CMPT 745
Software Engineering

Measurement & Performance

Nick Sumner
wsumner@sfu.ca

Performance & Measurement

- Real development must manage resources

Performance & Measurement

- Real development must manage resources
  - Time
  - Memory
  - Open connections
  - VM instances
  - Energy consumption
  - …

Performance & Measurement

- Real development must manage resources
  - Time
  - Memory
  - Open connections
  - VM instances
  - Energy consumption
  - …
- Resource usage is one form of performance
  - ***Performance*** – a measure of nonfunctional behavior of a program

Performance & Measurement

- Real development must manage resources
  - Time
  - Memory
  - Open connections
  - VM instances
  - Energy consumption
  - …
- Resource usage is one form of performance
  - ***Performance*** – a measure of nonfunctional behavior of a program
- We often need to assess performance or a change in performance

Data Structure A vs Data Structure B

Performance & Measurement

- Real development must manage resources
  - Time
  - Memory
  - Open connections
  - VM instances
  - Energy consumption
  - …
- Resource usage is one form of performance
  - ***Performance*** – a measure of nonfunctional behavior of a program
- We often need to assess performance or a change in performance

Data Structure A vs Data Structure B

How would you approach this in a data structures course?

Performance & Measurement

- Performance assessment is deceptively hard

[Demo/Exercise]

Performance & Measurement

- Performance assessment is deceptively hard
  - Modern systems involve complex actors

Performance & Measurement

- Performance assessment is deceptively hard
  - Modern systems involve complex actors
  - Theoretical models may be too approximate

Performance & Measurement

- Performance assessment is deceptively hard
  - Modern systems involve complex actors
  - Theoretical models may be too approximate
  - Even with the best intentions we can deceive ourselves

Performance & Measurement

- Performance assessment is deceptively hard
  - Modern systems involve complex actors
  - Theoretical models may be too approximate
  - Even with the best intentions we can deceive ourselves
- Good performance evaluation should be rigorous & scientific

Performance & Measurement

- Performance assessment is deceptively hard
  - Modern systems involve complex actors
  - Theoretical models may be too approximate
  - Even with the best intentions we can deceive ourselves
- Good performance evaluation should be rigorous & scientific
  - The same process applies in development as in ***good*** research

Performance & Measurement

- Performance assessment is deceptively hard
  - Modern systems involve complex actors
  - Theoretical models may be too approximate
  - Even with the best intentions we can deceive ourselves
- Good performance evaluation should be rigorous & scientific
  - The same process applies in development as in ***good*** research

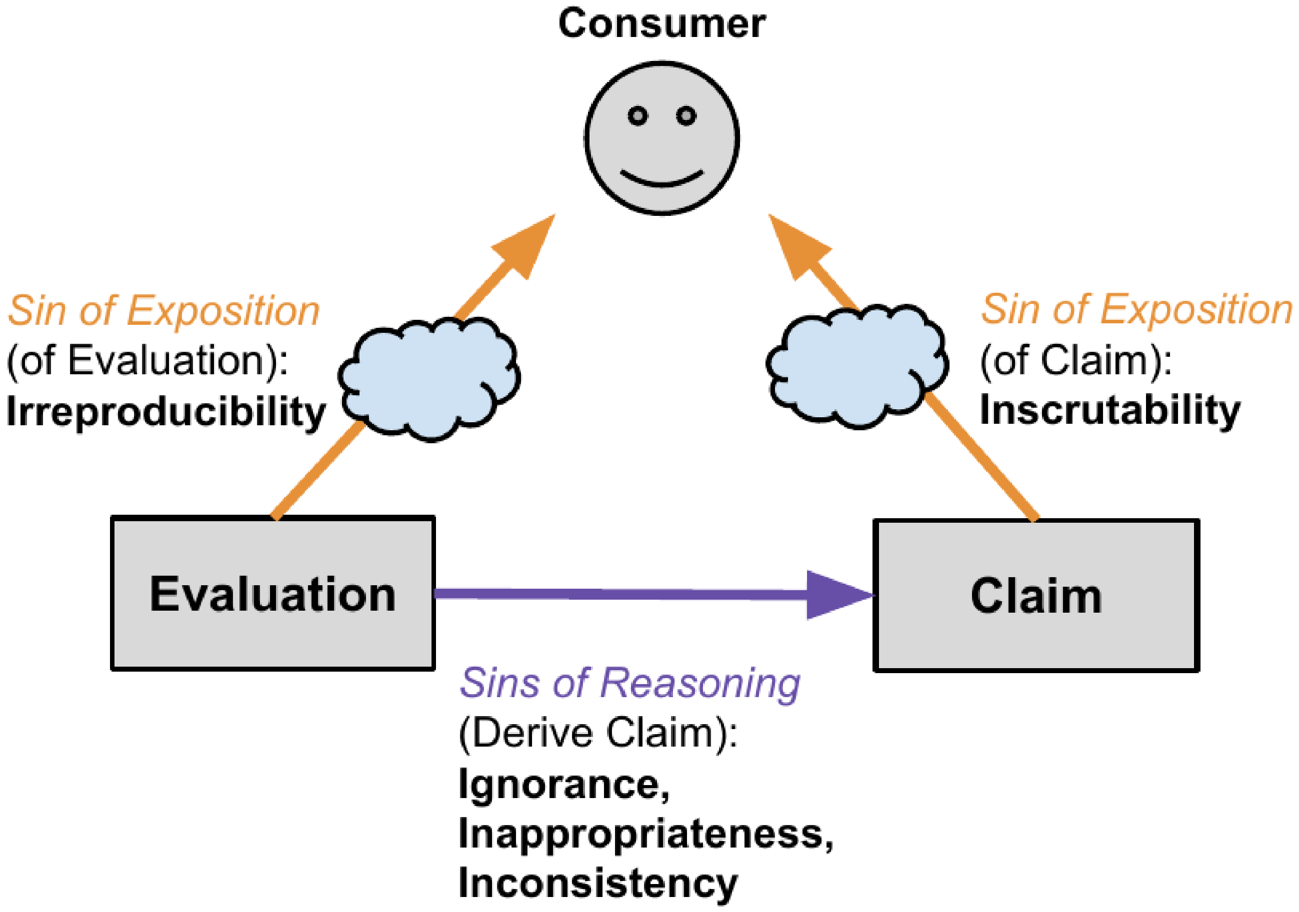
Performance & Measurement

- Performance assessment is deceptively hard
  - Modern systems involve complex actors
  - Theoretical models may be too approximate
  - Even with the best intentions we can deceive ourselves
- Good performance evaluation should be rigorous & scientific
  - The same process applies in development as in ***good*** research
    1) Clear claims
    2) Clear evidence
    3) Correct reasoning from evidence to claims

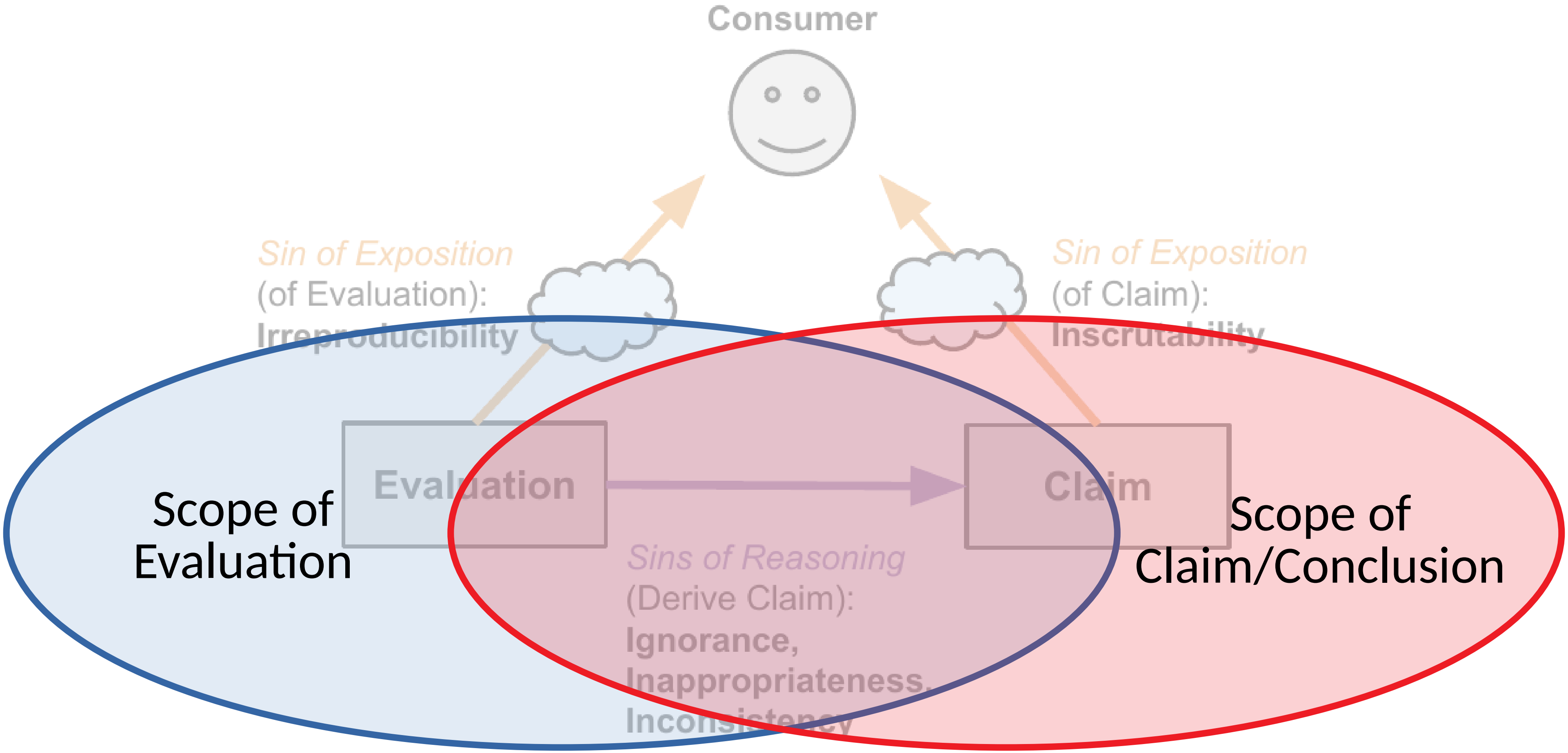
Performance & Measurement

- Performance assessment is deceptively hard
  - Modern systems involve complex actors
  - Theoretical models may be too approximate
  - Even with the best intentions we can deceive ourselves
- Good performance evaluation should be rigorous & scientific
  - The same process applies in development as in ***good*** research
    1) Clear claims
    2) Clear evidence
    3) Correct reasoning from evidence to claims
  - And yet this is challenging to get right!

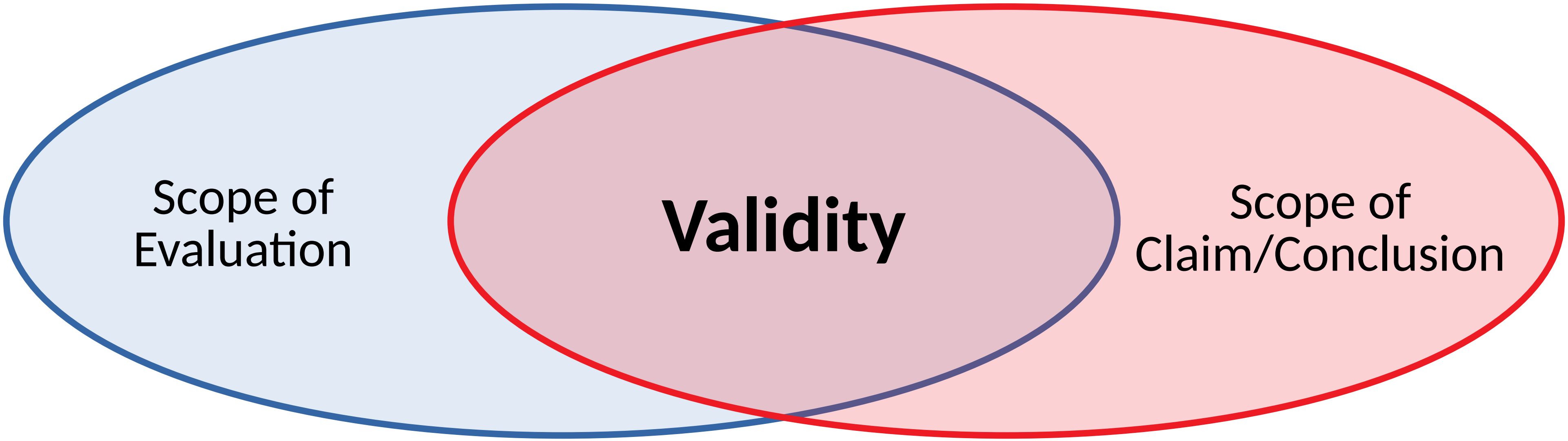
Performance & Measurement [Blackburn et al.]



Performance & Measurement [Blackburn et al.]



Performance & Measurement [Blackburn et al.]



Performance & Measurement [Blackburn et al.]

- Inscrutability
  - Lack of clarity on actors or relationships
  - Omission, Ambiguity, Distortion

Performance & Measurement [Blackburn et al.]

- Inscrutability
  - Lack of clarity on actors or relationships
  - Omission, Ambiguity, Distortion
- Irreproducibility
  - Lack of clarity in steps taken or data
  - Causes:
    - Omission of steps

Performance & Measurement [Blackburn et al.]

- Inscrutability
  - Lack of clarity on actors or relationships
  - Omission, Ambiguity, Distortion
- Irreproducibility
  - Lack of clarity in steps taken or data
  - Causes:
    - Omission of steps
    - Incomplete understanding of factors

Performance & Measurement [Blackburn et al.]

- Inscrutability
  - Lack of clarity on actors or relationships
  - Omission, Ambiguity, Distortion
- Irreproducibility
  - Lack of clarity in steps taken or data
  - Causes:
    - Omission of steps
    - Incomplete understanding of factors
    - Confidentiality & omission of data

Example …

Performance & Measurement [Blackburn et al.]

```
static int i = 0, j = 0, k = 0;
int main() {
  int g = 0, inc = 1;
  for (; g<65536; g++) {
    i += inc;
    j += inc;
    k += inc;
  }
  return 0;
}
```

Compare gcc -O2 vs -O3

Performance & Measurement [Blackburn et al.]

```
static int i = 0, j = 0, k = 0;
int main() {
  int g = 0, inc = 1;
  for (; g<65536; g++) {
    i += inc;
    j += inc;
    k += inc;
  }
  return 0;
}
```

Compare gcc -O2 vs -O3

One person may see a deterministic improvement..

Performance & Measurement [Blackburn et al.]

```
static int i = 0, j = 0, k = 0;
int main() {
  int g = 0, inc = 1;
  for (; g<65536; g++) {
    i += inc;
    j += inc;
    k += inc;
  }
  return 0;
}
```

Compare gcc -O2 vs -O3

One person may see a deterministic improvement..

Another may see a deterministic degradation.

Performance & Measurement [Blackburn et al.]

```
static int i = 0, j = 0, k = 0;
int main() {
  int g = 0, inc = 1;
  for (; g<65536; g++) {
    i += inc;
    j += inc;
    k += inc;
  }
  return 0;
}
```

Compare gcc -O2 vs -O3

One person may see a deterministic improvement..

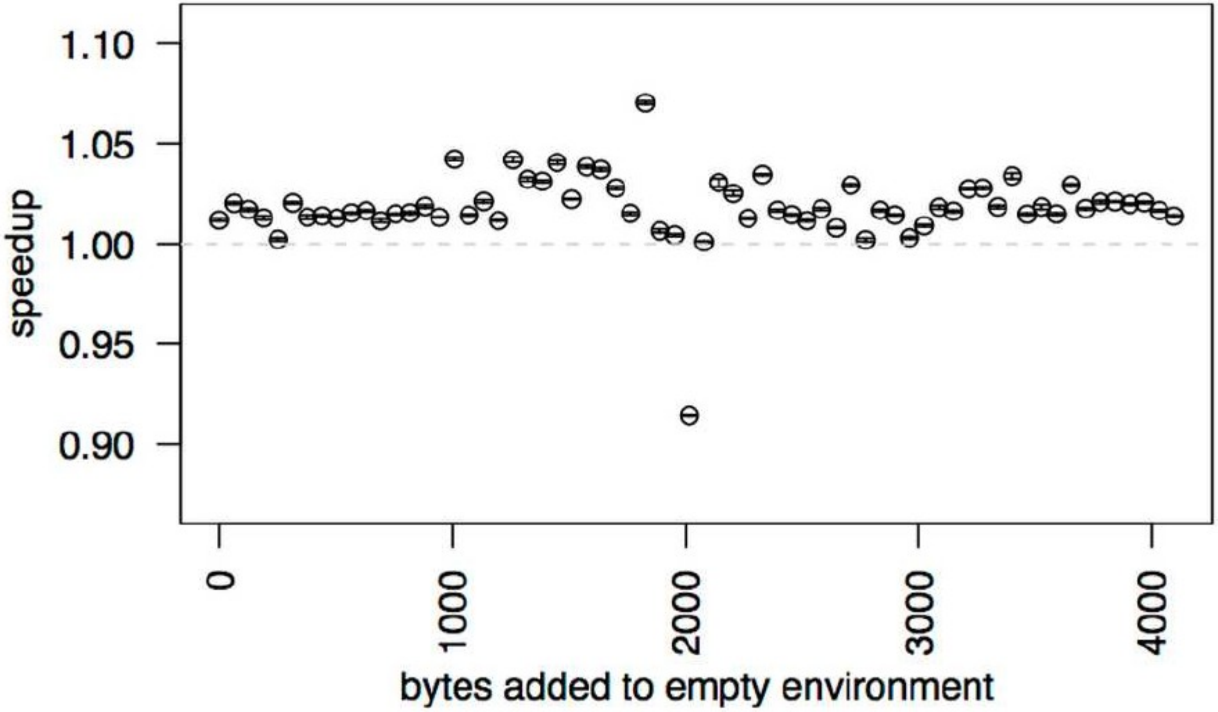
Another may see a deterministic degradation.

Both are right.

Performance & Measurement [Blackburn et al.]

```
static int i = 0, j = 0, k = 0;
int main() {
  int g = 0, inc = 1;
  for (; g<65536; g++) {
    i += inc;
    j += inc;
    k += inc;
  }
  return 0;
}
```

Compare gcc -O2 vs -O3

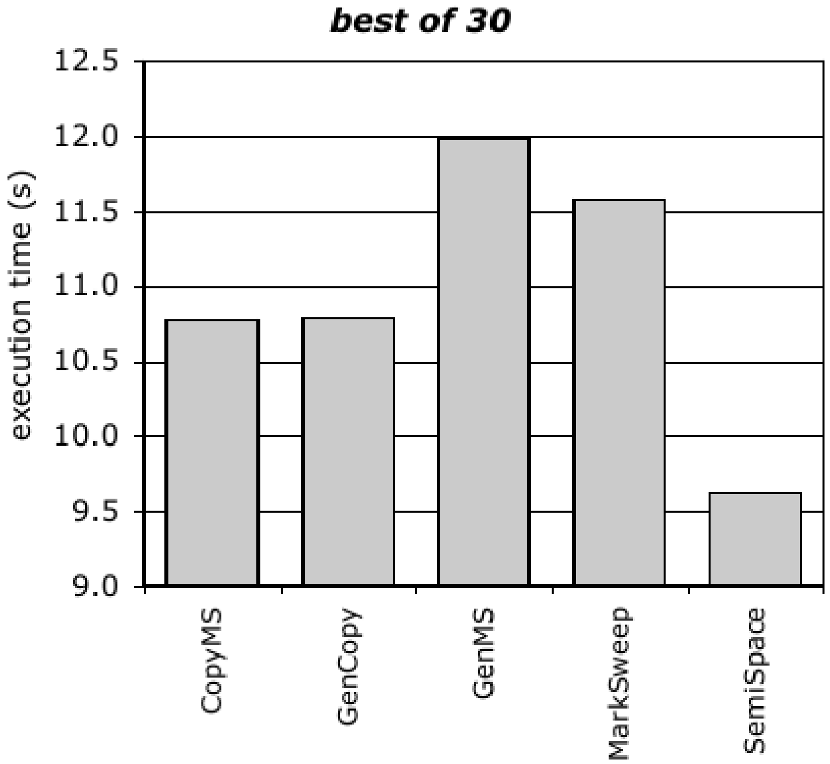


Performance & Measurement [Blackburn et al.]

- Ignorance – disregarding data or evidence against a claim
  - Ignoring data points

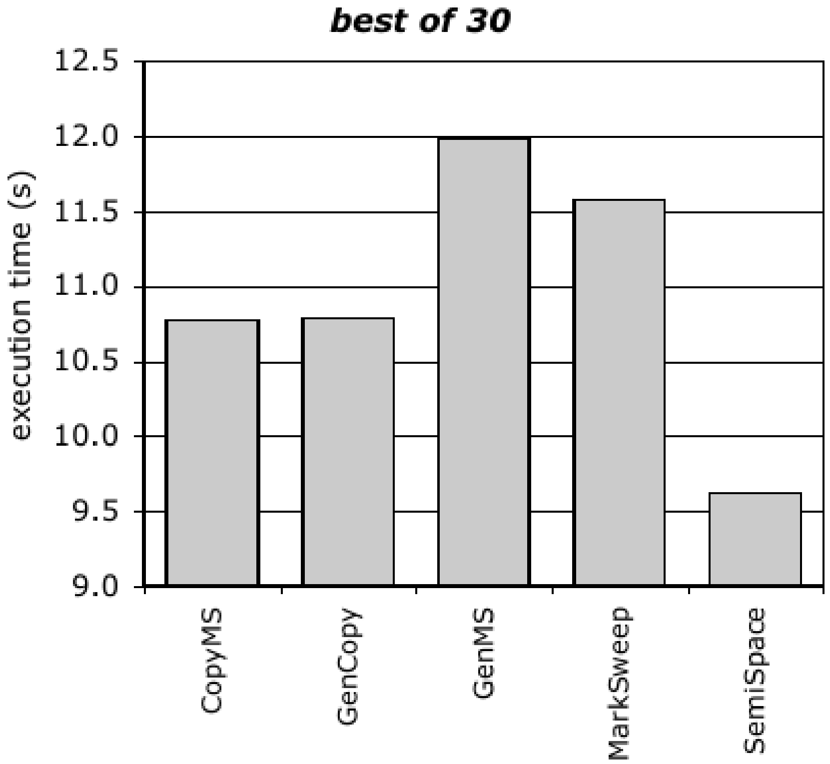
Performance & Measurement [Blackburn et al.]

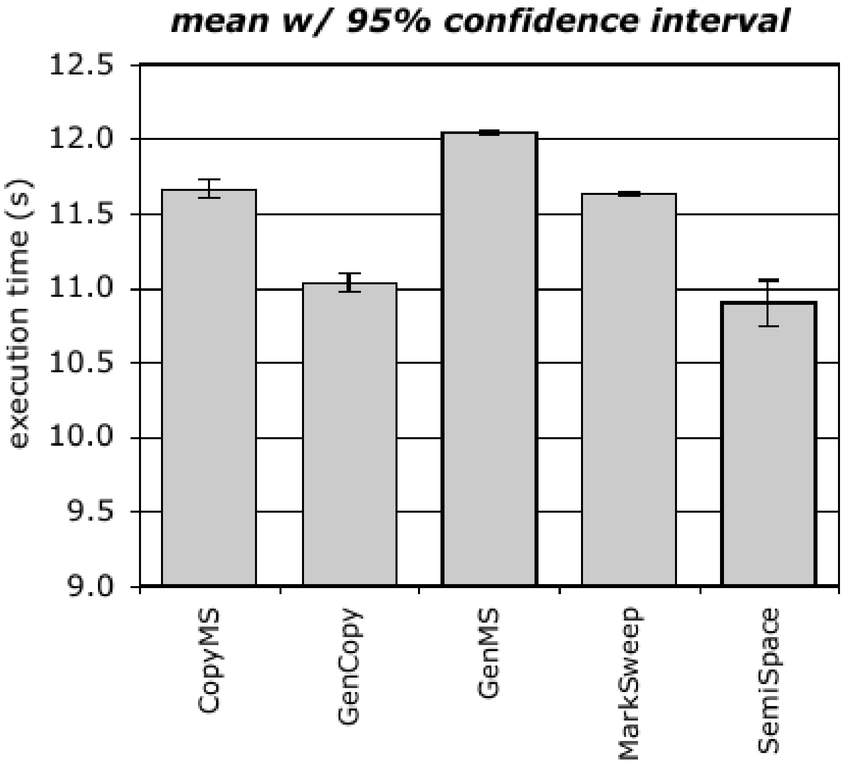
- Ignorance – disregarding data or evidence against a claim
  - Ignoring data points



Performance & Measurement [Blackburn et al.]

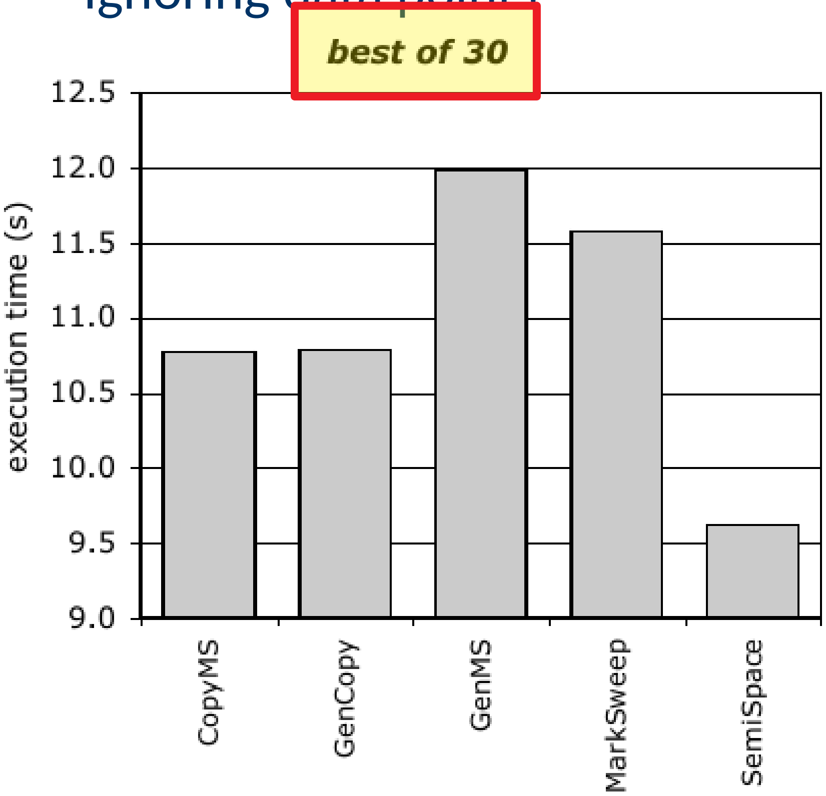
- Ignorance – disregarding data or evidence against a claim
  - Ignoring data points

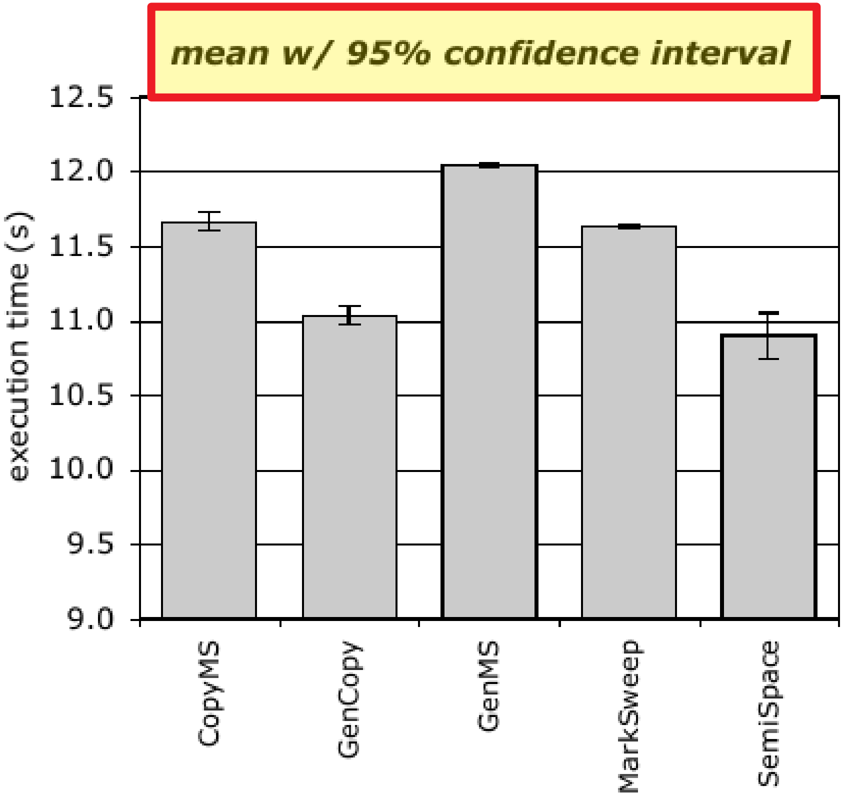




Performance & Measurement [Blackburn et al.]

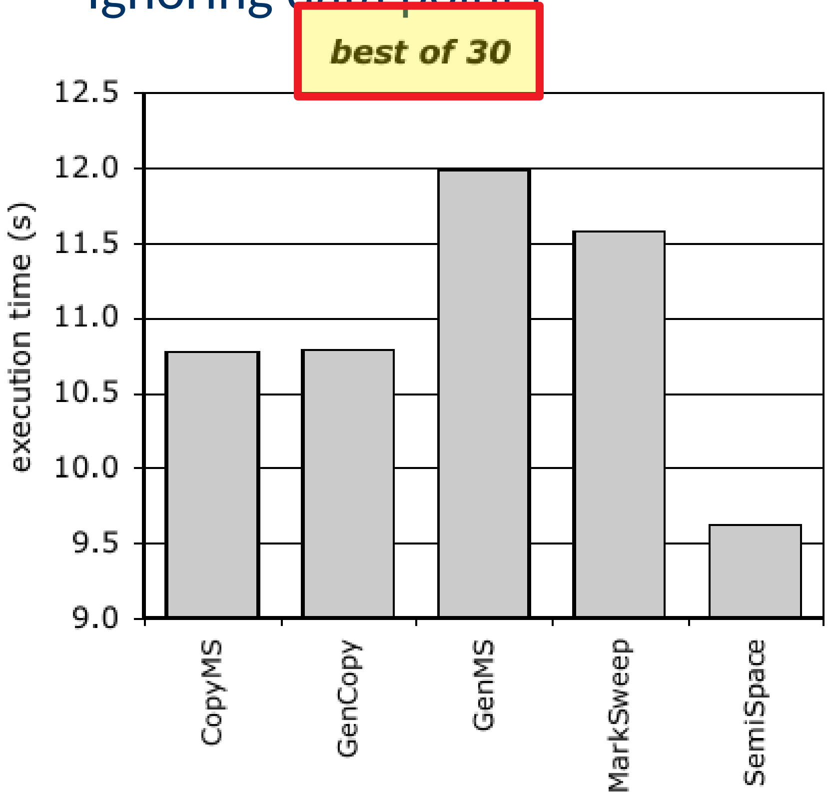
- Ignorance – disregarding data or evidence against a claim
  - Ignoring data points

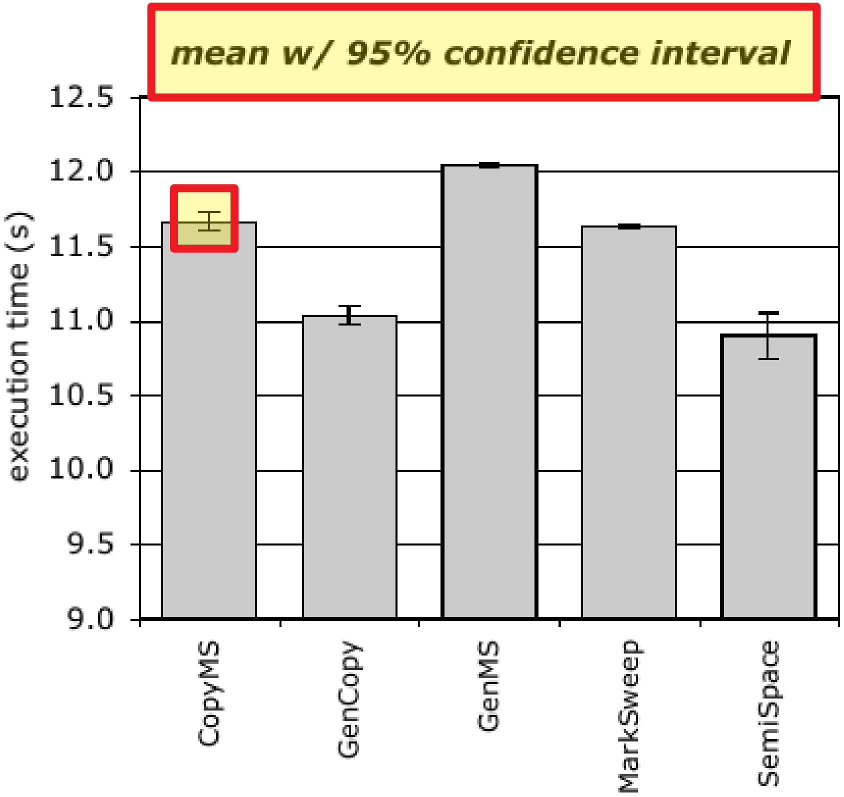




Performance & Measurement [Blackburn et al.]

- Ignorance – disregarding data or evidence against a claim
  - Ignoring data points



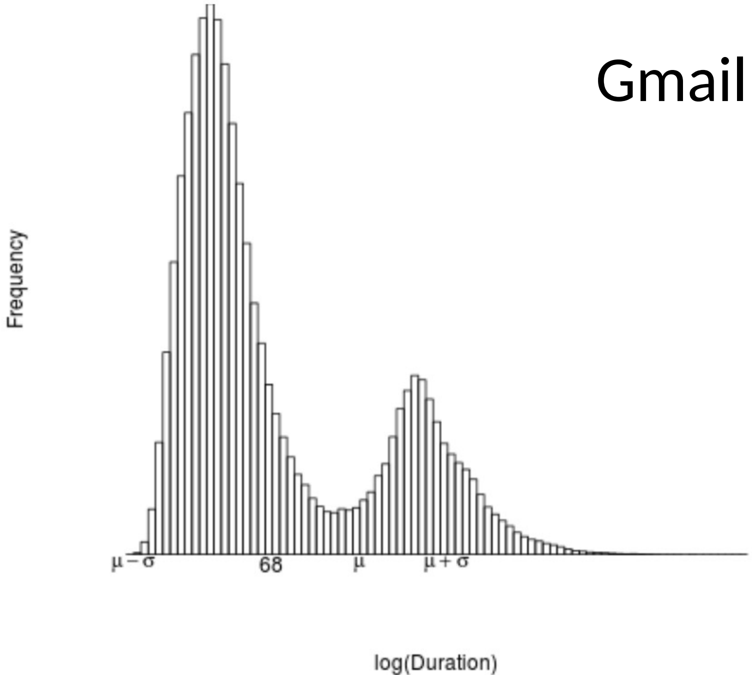


Performance & Measurement [Blackburn et al.]

- Ignorance – disregarding data or evidence against a claim
  - Ignoring data points
  - Ignoring distributions

Performance & Measurement [Blackburn et al.]

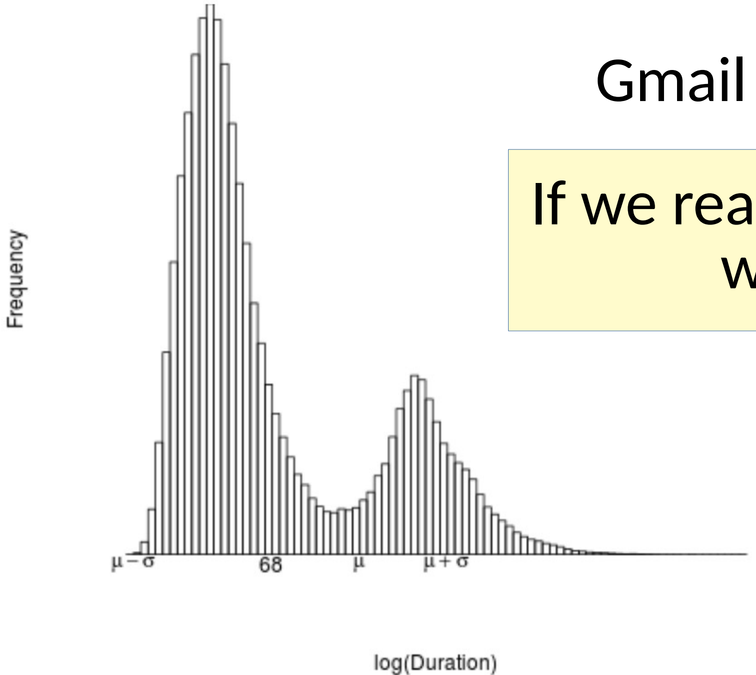
- Ignorance – disregarding data or evidence against a claim
  - Ignoring data points
  - Ignoring distributions



Gmail latency

Performance & Measurement [Blackburn et al.]

- Ignorance – disregarding data or evidence against a claim
  - Ignoring data points
  - Ignoring distributions

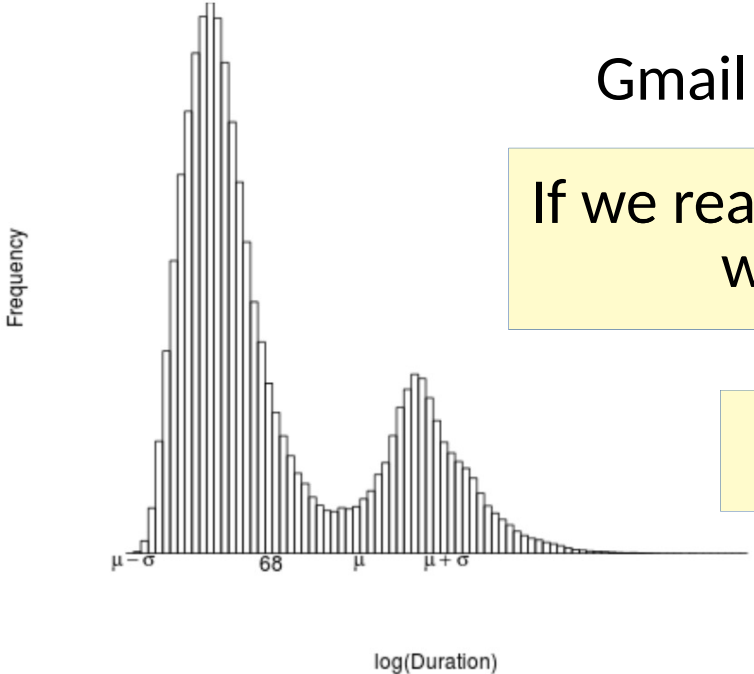


Gmail latency

If we reason about average latency, why is it misleading?

Performance & Measurement [Blackburn et al.]

- Ignorance – disregarding data or evidence against a claim
  - Ignoring data points
  - Ignoring distributions



Gmail latency

If we reason about average latency, why is it misleading?

What is better?

Performance & Measurement [Blackburn et al.]

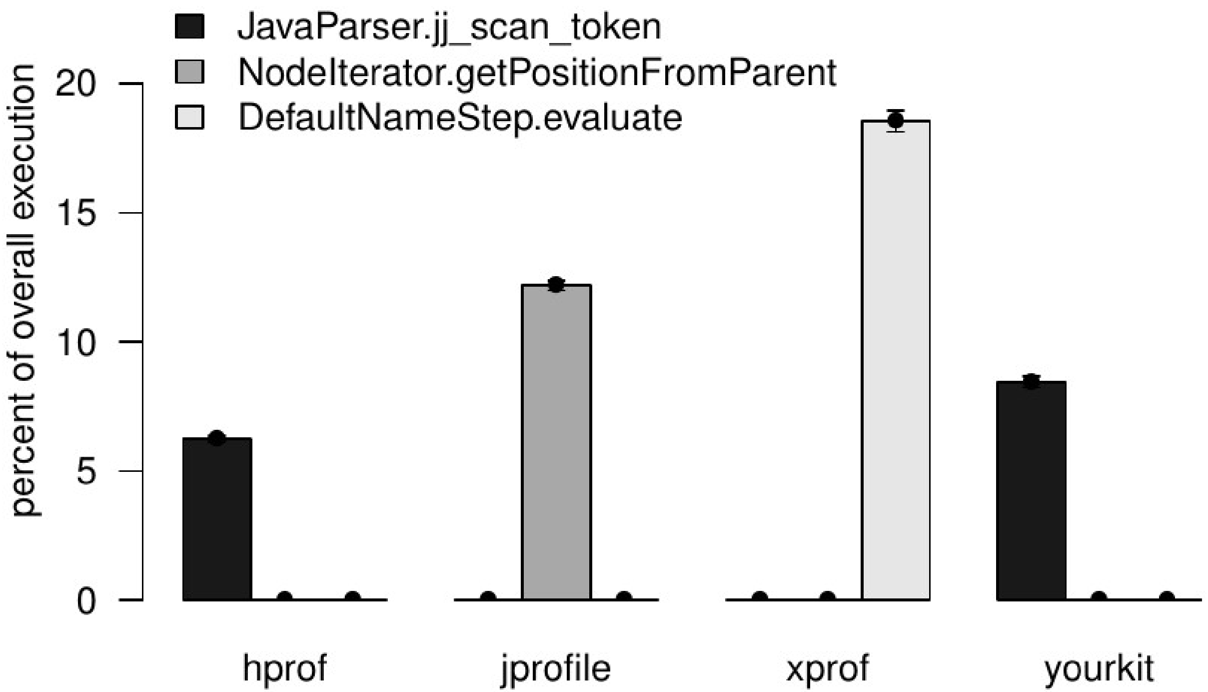
- Inappropriateness – claim is derived from facts not present

Performance & Measurement [Blackburn et al.]

- Inappropriateness – claim is derived from facts not present
  - Bad metrics (e.g. execution time vs. power)

Performance & Measurement [Blackburn et al.]

- Inappropriateness – claim is derived from facts not present
  - Bad metrics
  - Biased samples



Performance & Measurement [Blackburn et al.]

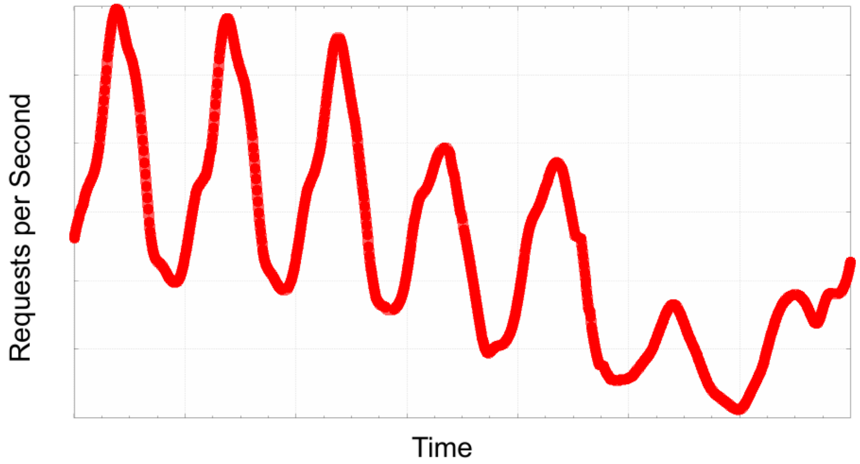
- Inappropriateness – claim is derived from facts not present
  - Bad metrics
  - Biased samples
  - …

Performance & Measurement [Blackburn et al.]

- Inappropriateness – claim is derived from facts not present
  - Bad metrics
  - Biased samples
  - …
- Inconsistency – comparing apples to oranges

Performance & Measurement [Blackburn et al.]

- Inappropriateness – claim is derived from facts not present
  - Bad metrics
  - Biased samples
  - …
- Inconsistency – comparing apples to oranges
  - Workload variation (e.g. learner effects, time of day, day of week, …)



Performance & Measurement [Blackburn et al.]

- Inappropriateness – claim is derived from facts not present
  - Bad metrics
  - Biased samples
  - …
- Inconsistency – comparing apples to oranges
  - Workload variation (e.g. learner effects, time of day)
  - Incompatible measures (e.g. performance counters across platforms)

Assessing Performance

Benchmarking

- We must reason rigorously about performance during assessment, investigation, & improvement

Benchmarking

- We must reason rigorously about performance during assessment, investigation, & improvement
- Assessing performance is done through benchmarking

Benchmarking

- We must reason rigorously about performance during assessment, investigation, & improvement
- Assessing performance is done through benchmarking
  - *Microbenchmarks*
    - Focus on cost of an operation in isolation
    - Can help identify core performance details & explain causes

Benchmarking

- We must reason rigorously about performance during assessment, investigation, & improvement
- Assessing performance is done through benchmarking
  - *Microbenchmarks*
    - Focus on cost of an operation in isolation
    - Can help identify core performance details & explain causes
  - *Macrobenchmarks*
    - Real world system performance

Benchmarking

- We must reason rigorously about performance during assessment, investigation, & improvement
- Assessing performance is done through benchmarking
  - *Microbenchmarks*
    - Focus on cost of an operation in isolation
    - Can help identify core performance details & explain causes
  - *Macrobenchmarks*
    - Real world system performance
  - Workloads (inputs) must be chosen carefully either way.
    - representative, pathological, scenario driven, …

Benchmarking

- We must reason rigorously about performance during assessment, investigation, & improvement
- Assessing performance is done through benchmarking
  - *Microbenchmarks*
    - Focus on cost of an operation in isolation
    - Can help identify core performance details & explain causes
  - *Macrobenchmarks*
    - Real world system performance
  - Workloads (inputs) must be chosen carefully either way.
    - representative, pathological, scenario driven, ...

Let’s dig into a common approach to consider issues

Benchmarking

- Suppose we want to run a microbenchmark

```
startTime = getCurrentTimeInSeconds();
doWorkloadOfInterest();
endTime = getCurrentTimeInSeconds();
reportResult(endTime – startTime);
```

Benchmarking

- Suppose we want to run a microbenchmark

```
startTime = getCurrentTimeInSeconds();
doWorkloadOfInterest();
endTime = getCurrentTimeInSeconds();
reportResult(endTime – startTime);
```

What possible issues do you observe?

Benchmarking

- Suppose we want to run a microbenchmark

```
startTime = getCurrentTimeInSeconds();
doWorkloadOfInterest();
endTime = getCurrentTimeInSeconds();
reportResult(endTime – startTime);
```

  - Granularity of measurement
  - Warm up effects
  - Nondeterminism
  - Size of workload
  - System interference
  - Frequency scaling?
  - Interference of other workloads?
  - Alignment?

Benchmarking

- Granularity & Units
  - Why is granularity a problem?
  - What are alternatives to `getCurrentTimeInSeconds()`?

Benchmarking

- Granularity & Units
  - Why is granularity a problem?
  - What are alternatives to `getCurrentTimeInSeconds()`?
  - What if I want to predict performance on a different machine?

Benchmarking

- Granularity & Units
  - Why is granularity a problem?
  - What are alternatives to `getCurrentTimeInSeconds()`?
  - What if I want to predict performance on a different machine?
    - Using *cycles* instead of wall clock time can be useful, but has its own limitations
    - Remember the sins of measurement

Benchmarking

- Warm up time
  - Why is warm up time necessary *in general*?

Benchmarking

- Warm up time
  - Why is warm up time necessary *in general*?
  - Why is it especially problematic for systems like Java?

Benchmarking

- Warm up time
  - Why is warm up time necessary *in general*?
  - Why is it especially problematic for systems like Java?
  - How can we modify our example to facilitate this?

Benchmarking

- Warm up time
  - Why is warm up time necessary *in general*?
  - Why is it especially problematic for systems like Java?
  - How can we modify our example to facilitate this?

```
for (…) doWorkloadOfInterest();
startTime = getCurrentTimeInSeconds();
doWorkloadOfInterest();
endTime = getCurrentTimeInSeconds();
reportResult(endTime – startTime);
```

Benchmarking

- Warm up time
  - Why is warm up time necessary *in general*?
  - Why is it especially problematic for systems like Java?
  - How can we modify our example to facilitate this?

Quiescence? Post-JIT hooks? …?

```
for (…) doWorkloadOfInterest();
startTime = getCurrentTimeInSeconds();
doWorkloadOfInterest();
endTime = getCurrentTimeInSeconds();
reportResult(endTime – startTime);
```

Benchmarking

- Warm up time
  - Why is warm up time necessary *in general*?
  - Why is it especially problematic for systems like Java?
  - How can we modify our example to facilitate this?

Quiescence? Post-JIT hooks? …?
It is ***complicated***. [Tratt 2018]

```
for (…) doWorkloadOfInterest();
startTime = getCurrentTimeInSeconds();
doWorkloadOfInterest();
endTime = getCurrentTimeInSeconds();
reportResult(endTime – startTime);
```

Benchmarking

- Nondeterministic behavior
  - Will `getCurrentTimeInSeconds()` always return the same number?

Why/why not?

Benchmarking

- Nondeterministic behavior
  - Will `getCurrentTimeInSeconds()` always return the same number?
  - So what reflects a *meaningful* result?
    - Hint: The Law of Large Numbers!

Benchmarking

- Nondeterministic behavior
  - Will `getCurrentTimeInSeconds()` always return the same number?
  - So what reflects a *meaningful* result?
    - Hint: The Law of Large Numbers!
- By running the same test many times,
  the sample arithmetic mean will converge on the real one

Benchmarking

- Nondeterministic behavior
  - Will `getCurrentTimeInSeconds()` always return the same number?
  - So what reflects a *meaningful* result?
    - Hint: The Law of Large Numbers!
- By running the same test many times,
  the sample arithmetic mean will converge on the real one

Is this always what you want?

Benchmarking

- A revised (informal) approach:

```
for (…) doWorkloadOfInterest();
startTime = getCurrentTimeInNanos();
for (…) doWorkloadOfInterest();
endTime = getCurrentTimeInNanos();
reportResult(endTime – startTime);
```

Benchmarking

- A revised (informal) approach:

```
for (…) doWorkloadOfInterest();
startTime = getCurrentTimeInNanos();
for (…) doWorkloadOfInterest();
endTime = getCurrentTimeInNanos();
reportResult(endTime - startTime);
```

- This still does not solve everything
  - Frequency scaling?
  - Interference of other workloads?
  - Alignment?

Benchmarking

- Now we have a benchmark, how do we interpret/report it?
  - We must ***compare***

Benchmarking

- Now we have a benchmark, how do we interpret/report it?
  - We must ***compare***
    - Benchmark vs expectation/mental model
    - Different solutions
    - Over time

Benchmarking

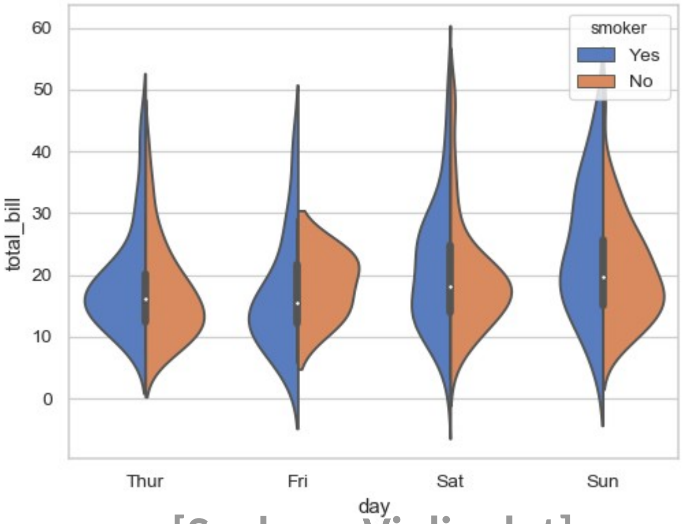
- Now we have a benchmark, how do we interpret/report it?
  - We must ***compare***
    - Benchmark vs expectation/mental model
    - Different solutions
    - Over time
    - Results are often normalized against the baseline

Benchmarking

- Now we have a benchmark, how do we interpret/report it?
  - We must ***compare***
  - We must remember results are ***statistical***

Benchmarking

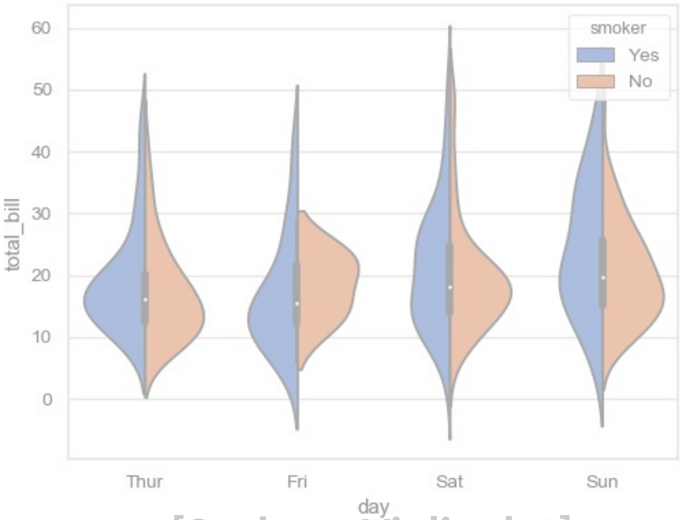
- Now we have a benchmark, how do we interpret/report it?
  - We must ***compare***
  - We must remember results are ***statistical***
    - Show the distribution (e.g. violin plots)



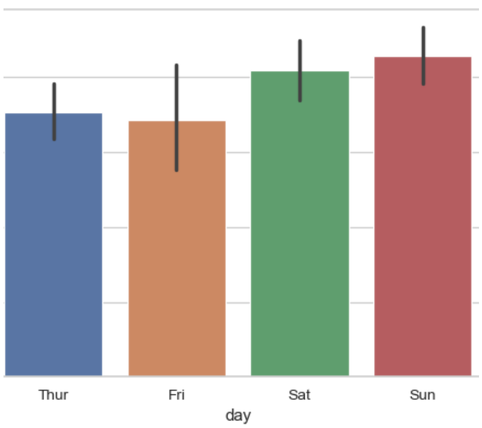
[Seaborn Violinplot]

Benchmarking

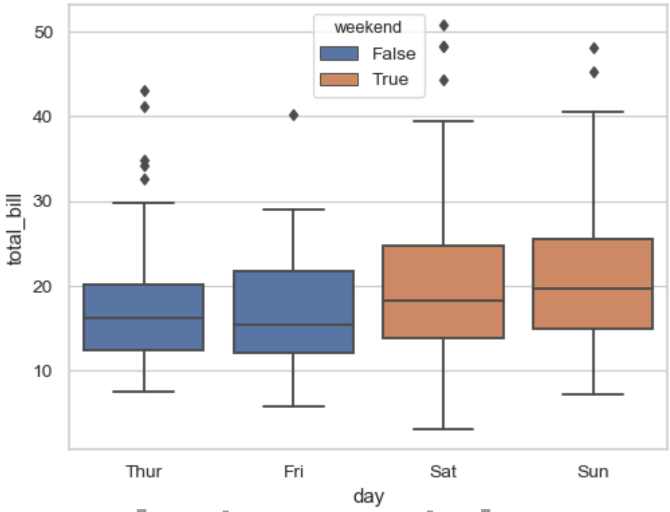
- Now we have a benchmark, how do we interpret/report it?
  - We must ***compare***
  - We must remember results are ***statistical***
    - Show the distribution (e.g. violin plots)
    - Summarize the distribution (e.g. mean and confidence intervals, box & whisker)



[Seaborn Violinplot]



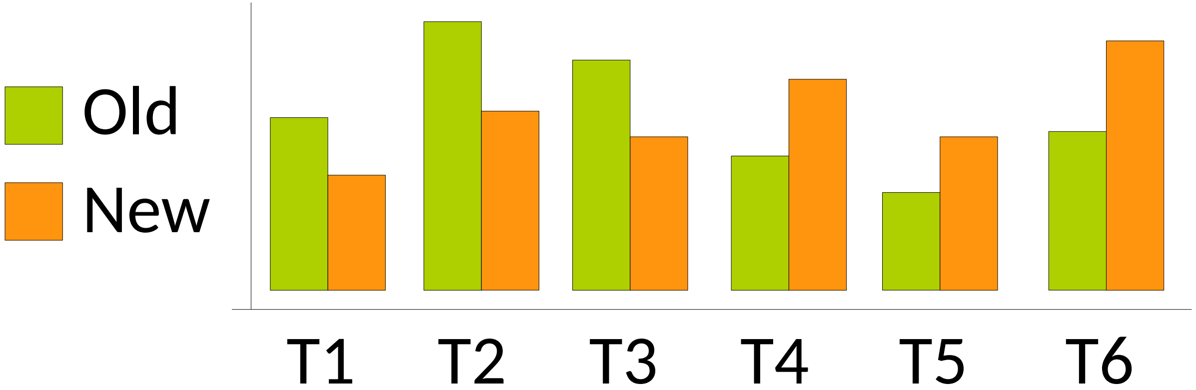
[Seaborn Barplot]



[Seaborn Boxplot]

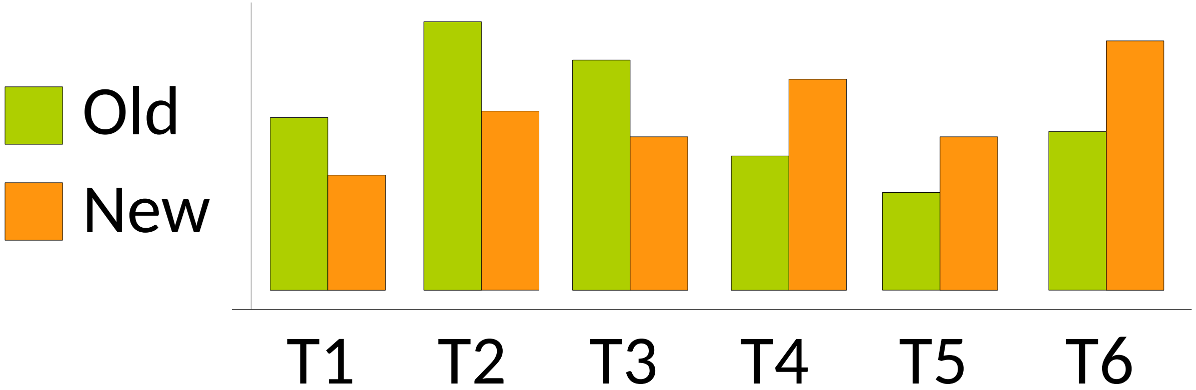
Benchmarking

- A ***benchmark suite*** comprises multiple benchmarks



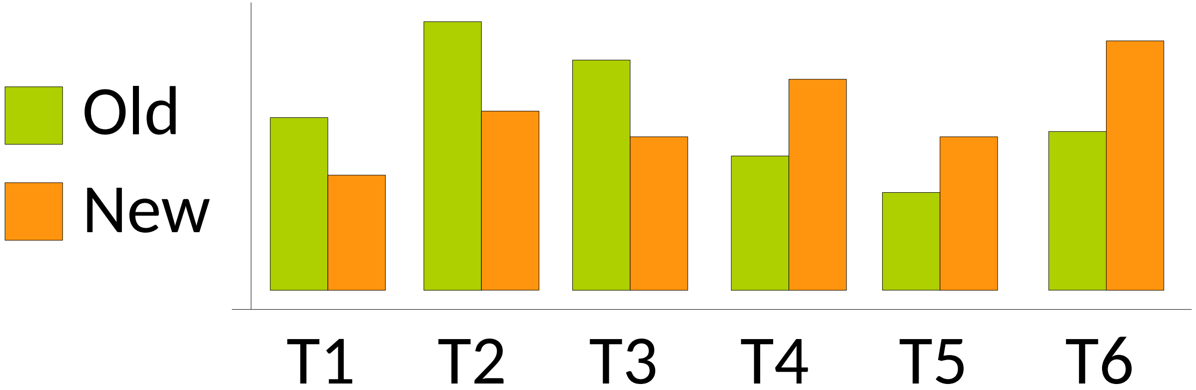
Benchmarking

- A benchmark suite comprises multiple benchmarks
- Now we have multiple results, how should we consider them?



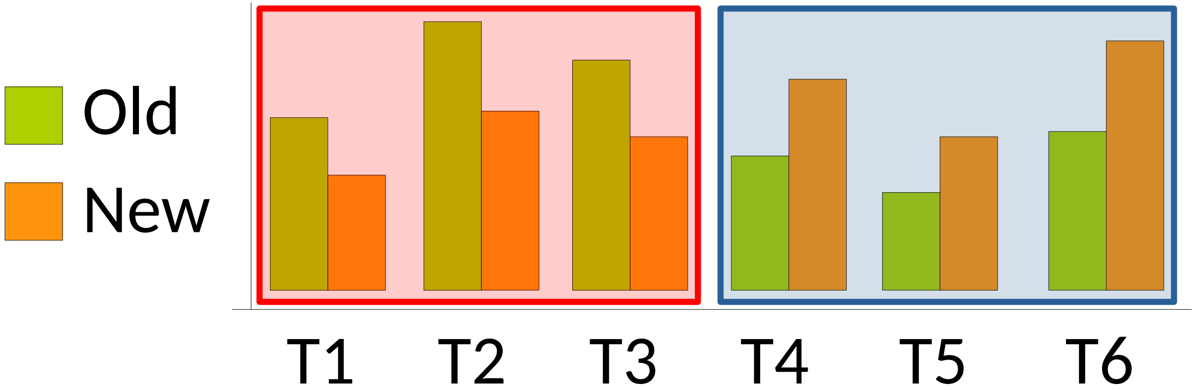
Benchmarking

- A benchmark suite comprises multiple benchmarks
- Now we have multiple results, how should we consider them?
  - 2 major senarios
    - *Hypothesis testing*
      - Is solution A different than B?



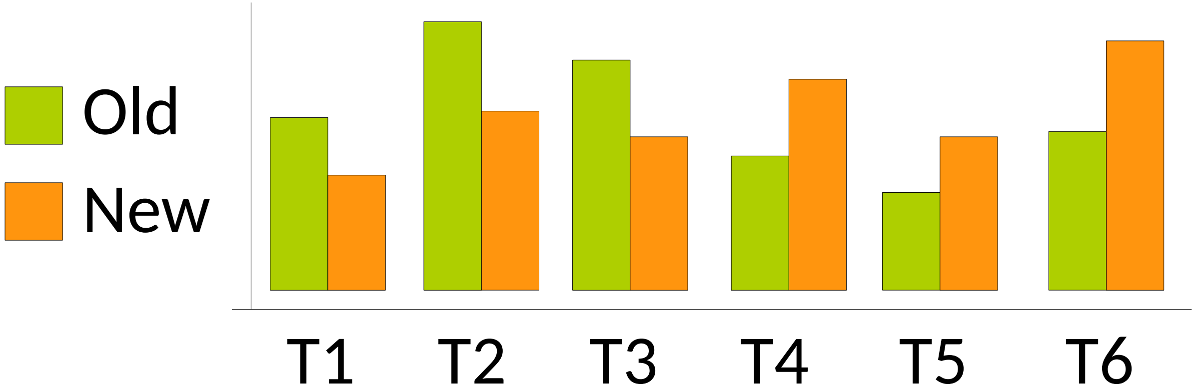
Benchmarking

- A benchmark suite comprises multiple benchmarks
- Now we have multiple results, how should we consider them?
  - 2 major senarios
    - *Hypothesis testing*
      - Is solution A different than B?



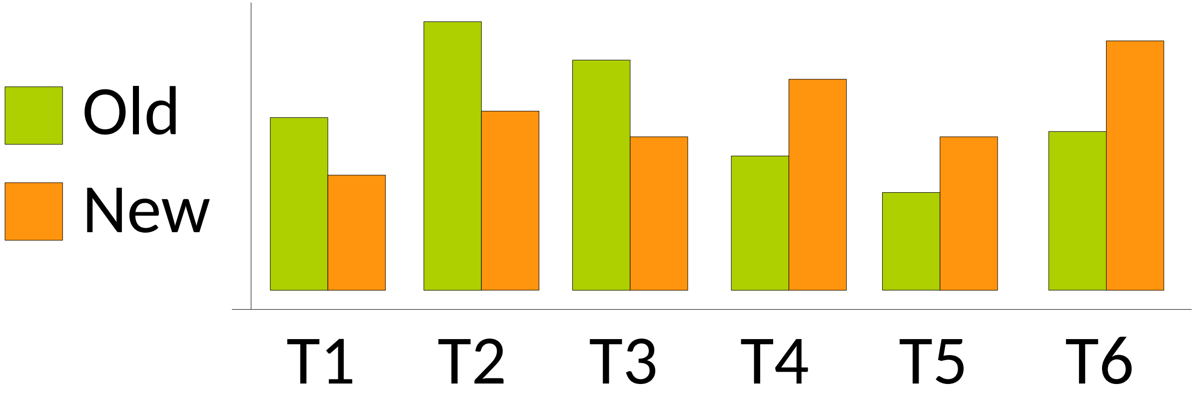
Benchmarking

- A benchmark suite comprises multiple benchmarks
- Now we have multiple results, how should we consider them?
  - 2 major senarios
    - *Hypothesis testing*
      - Is solution A different than B?
      - You can use ANOVA



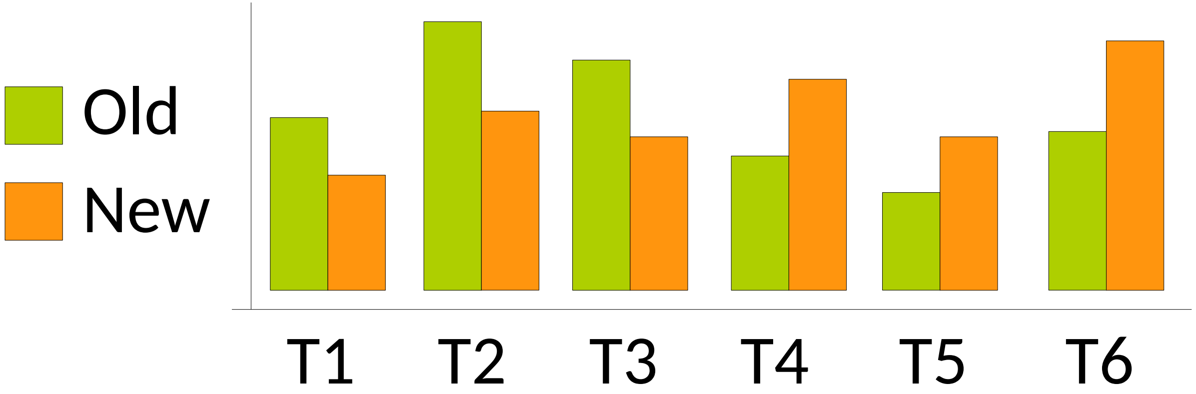
Benchmarking

- A benchmark suite comprises multiple benchmarks
- Now we have multiple results, how should we consider them?
  - 2 major senarios
    - *Hypothesis testing*
    - *Summary statistics*



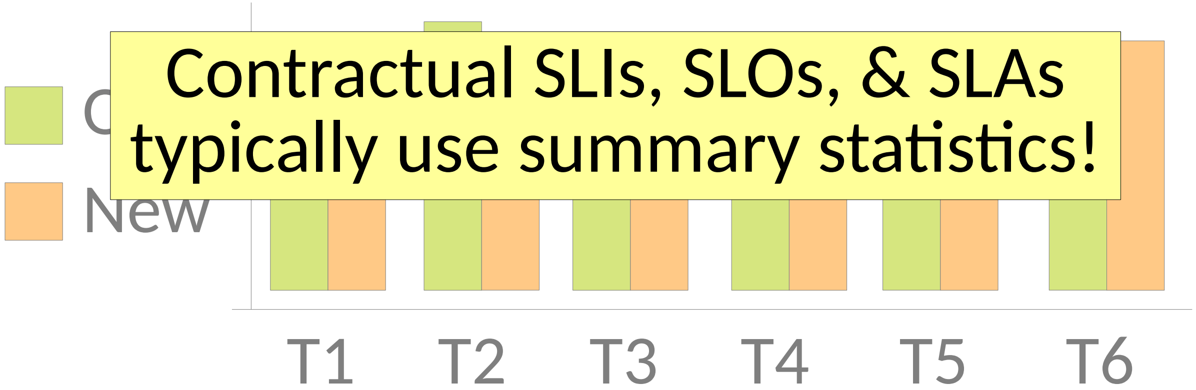
Benchmarking

- A benchmark suite comprises multiple benchmarks
- Now we have multiple results, how should we consider them?
  - 2 major senarios
    - *Hypothesis testing*
    - *Summary statistics*
      - Condensing a suite to a single number
      - Intrinsically lossy, but can still be useful



Benchmarking

- A benchmark suite comprises multiple benchmarks
- Now we have multiple results, how should we consider them?
  - 2 major senarios
    - *Hypothesis testing*
    - *Summary statistics*
      - Condensing a suite to a single number
      - Intrinsically lossy, but can still be useful



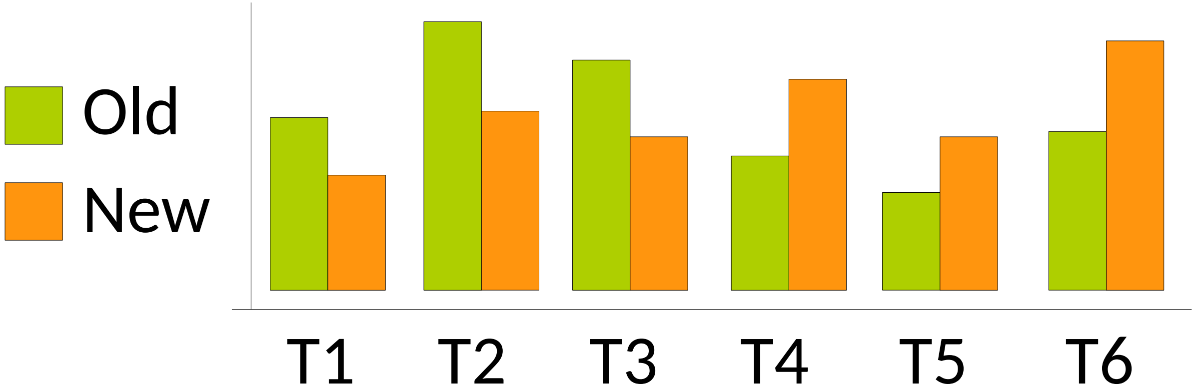
Contractual SLIs, SLOs, & SLAs typically use summary statistics!

Benchmarking

- A benchmark suite comprises multiple benchmarks
- Now we have multiple results, how should we consider them?
  - 2 major senarios
    - *Hypothesis testing*
    - *Summary statistics*
      - Condensing a suite to a single number
      - Intrinsically lossy, but can still be useful

Old: ?
New: ?

$\frac{New}{Old}$: ?



Summary Statistics

Averages of $r_1, r_2, \ldots, r_N$

- Many ways to measure *expectation* or *tendency*

Summary Statistics

Averages of $r_1, r_2, \ldots, r_N$

- Many ways to measure *expectation* or *tendency*
- Arithmetic Mean

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

Summary Statistics

Averages of $r_1, r_2, \ldots, r_N$

- Many ways to measure *expectation* or *tendency*
- Arithmetic Mean

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Harmonic Mean

$$\frac{N}{\sum_{i=1}^{N} \frac{1}{r_i}}$$

Summary Statistics

Averages of $r_1, r_2, \ldots, r_N$

- Many ways to measure *expectation* or *tendency*
- Arithmetic Mean

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Harmonic Mean

$$\frac{N}{\sum_{i=1}^{N}\frac{1}{r_i}}$$

- Geometric Mean

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$

Summary Statistics

Averages of $r_1, r_2, \ldots, r_N$

- Many ways to measure *expectation* or *tendency*
- Arithmetic Mean

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Harmonic Mean

$$\frac{N}{\sum_{i=1}^{N} \frac{1}{r_i}}$$

- Geometric Mean

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$

Each type means something different and has valid uses

Summary Statistics

- Arithmetic Mean
  - Good for reporting averages of numbers that mean the same thing

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

Summary Statistics

- Arithmetic Mean
  - Good for reporting averages of numbers that mean the same thing
  - Used for computing *sample means*

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

Summary Statistics

- Arithmetic Mean
  - Good for reporting averages of numbers that mean the same thing
  - Used for computing sample means
  - e.g. Timing the same workload many times

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

Summary Statistics

- Arithmetic Mean
  - Good for reporting averages of numbers that mean the same thing
  - Used for computing sample means
  - e.g. Timing the same workload many times

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

Handling Nondeterminism

```
for (x in 0 to ...)
  times[x] = doWorkloadOfInterest();
```

E(time) = arithmean(times)

Summary Statistics

- Arithmetic Mean
  - Good for reporting averages of numbers that mean the same thing
  - Used for computing sample means
  - e.g. Timing the same workload many times

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Harmonic Mean
  - Good for reporting *rates*

$$\frac{N}{\sum_{i=1}^{N} \frac{1}{r_i}}$$

Summary Statistics

- Arithmetic Mean
  - Good for reporting averages of numbers that mean the same thing
  - Used for computing sample means
  - e.g. Timing the same workload many times

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Harmonic Mean
  - Good for reporting *rates*
  - e.g. Required throughput for a set of tasks

$$\frac{N}{\sum_{i=1}^{N} \frac{1}{r_i}}$$

Summary Statistics

Given tasks t1, t2, & t3 serving 40 pages each:

thoughput(t1) = 10 pages/sec
thoughput(t2) = 20 pages/sec
thoughput(t3) = 20 pages/sec

What is the average throughput? What should it mean?

ean the same thing

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Good for reporting rates
- e.g. Required throughput for a set of tasks

$$\frac{N}{\sum_{i=1}^{N}\frac{1}{r_i}}$$

Summary Statistics

Given tasks t1, t2, & t3 serving 40 pages each:

thoughput(t1) = 10 pages/sec

thoughput(t2) = 20 pages/sec

thoughput(t3) = 20 pages/sec

What is the average throughput? What should it mean?

Arithmetic = 16.7 p/s

ean the same thing

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Good for reporting rates
- e.g. Required throughput for a set of tasks

$$\frac{N}{\sum_{i=1}^{N}\frac{1}{r_i}}$$

Summary Statistics

Given tasks t1, t2, & t3 serving 40 pages each:

thoughput(t1) = 10 pages/sec
thoughput(t2) = 20 pages/sec
thoughput(t3) = 20 pages/sec

What is the average throughput? What should it mean?

Arithmetic = 16.7 p/s   Harmonic = 15 p/s

ean the same thing

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Good for reporting rates
- e.g. Required throughput for a set of tasks

$$\frac{N}{\sum_{i=1}^{N}\frac{1}{r_i}}$$

$$\frac{3}{\frac{1}{10}+\frac{1}{20}+\frac{1}{20}}=15\,p/s$$

Summary Statistics

Given tasks t1, t2, & t3 serving 40 pages each:

thoughput(t1) = 10 pages/sec
thoughput(t2) = 20 pages/sec
thoughput(t3) = 20 pages/sec

What is the average throughput? What should it mean?

Arithmetic = 16.7 p/s — Harmonic = 15 p/s

120/16.7 = 7.2 — 120/15 = 8

ean the same thing

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Good for reporting rates
- e.g. Required throughput for a set of tasks

$$\frac{N}{\sum_{i=1}^{N} \frac{1}{r_i}}$$

Summary Statistics

Given tasks t1, t2, & t3 serving 40 pages each:

thoughput(t1) = 10 pages/sec
thoughput(t2) = 20 pages/sec
thoughput(t3) = 20 pages/sec

What is the average throughput? What should it mean?

Arithmetic = 16.7 p/s — 120/16.7 = 7.2

Harmonic = 15 p/s — 120/15 = 8

ean the same thing

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

– Good for reporting rates
– e.g. Required throughput for a set of tasks

Identifies the constant rate required for the same time

$$\frac{N}{\sum_{i=1}^{N} \frac{1}{r_i}}$$

Summary Statistics

Given tasks t1, t2, & t3 serving 40 pages each:

thoughput(t1) = 10 pages/sec
thoughput(t2) = 20 pages/sec
thoughput(t3) = 20 pages/sec

What is the average throughput? What should it mean?

Arithmetic = 16.7 p/s   Harmonic = 15 p/s
120/16.7 = 7.2   120/15 = 8

ean the same thing

$$\frac{1}{N}\sum_{i=1}^{N} r_i$$

- Good for reporting rates
- e.g. Required throughput for a set of tasks

Identifies the constant rate required for the same time

$$\frac{N}{\sum_{i=1}^{N} \frac{1}{r_i}}$$

CAVEAT: If the size of each workload changes, a weighted harmonic mean is required!

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

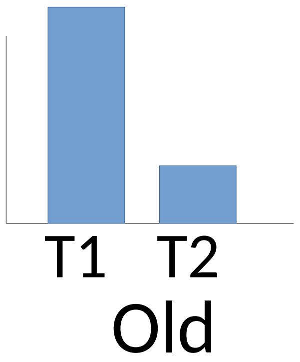
$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$

Any idea why it may be useful here?
(A bit of a thought experiment)

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

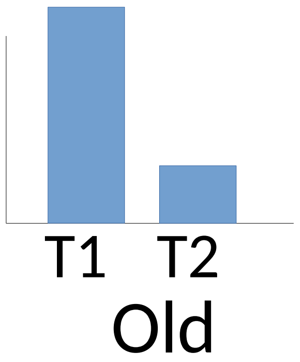
$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$

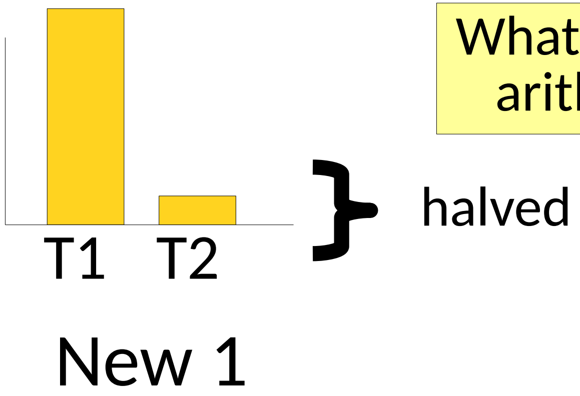


Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$



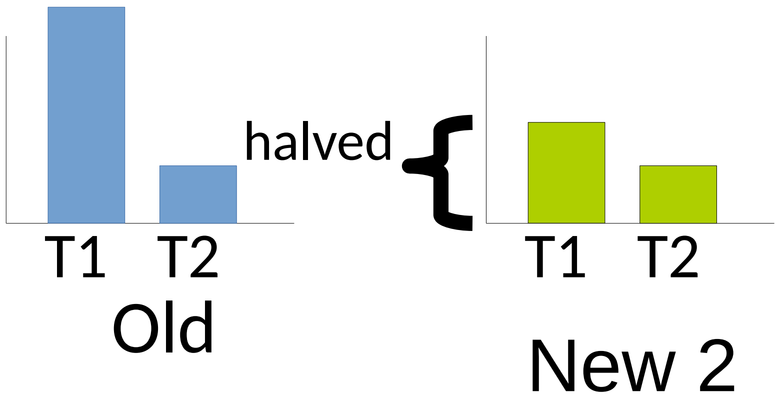


What happens to the arithmetic mean?

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$

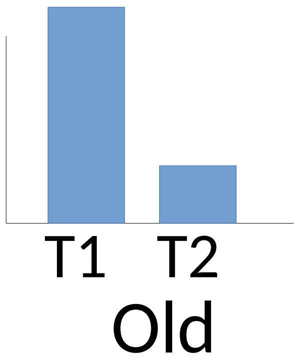


What happens to the arithmetic mean?

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$

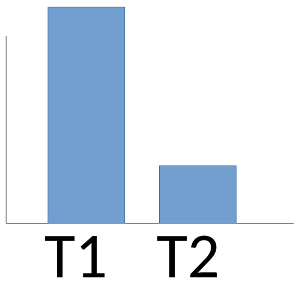


The (non) change to T1 dominates any behavior for T2!

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$



Old

Geometric:

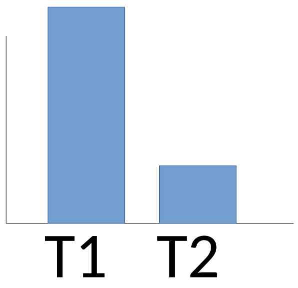
$$\sqrt{r_1 \times r_2}$$

Old

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$



Old

Geometric:

$$\sqrt{r_1 \times r_2}$$

Old

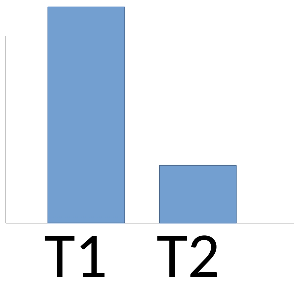
$$\sqrt{r_1 \times (\frac{1}{2} r_2)}$$

New 1

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$



Old

Geometric:

$$\sqrt{r_1 \times r_2}$$

Old

$$\sqrt{r_1 \times (\frac{1}{2} r_2)}$$

New 1

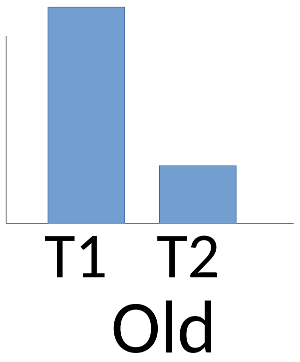
$$\sqrt{(\frac{1}{2} r_1) \times r_2}$$

New 2

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$



Geometric:

$$\sqrt{r_1 \times r_2}$$

Old

$$\sqrt{r_1 \times (\frac{1}{2} r_2)} = \sqrt{\frac{1}{2} \times r_1 \times r_2} = \sqrt{(\frac{1}{2} r_1) \times r_2}$$

New 1

New 2

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks
  - A 10% difference in any benchmark affects the final value the same way

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$

Summary Statistics

- Geometric Mean
  - Good for reporting results that mean different things
  - e.g. Timing results across *many different* benchmarks
  - A 10% difference in any benchmark affects the final value the same way

$$\sqrt[N]{\prod_{i=1}^{N} r_i}$$

Note: It doesn't have an *intuitive* meaning!
It does provides a balanced *score* of performance.

See [Mashey 2004] for deeper insights.

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure *pathological* cases?

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …

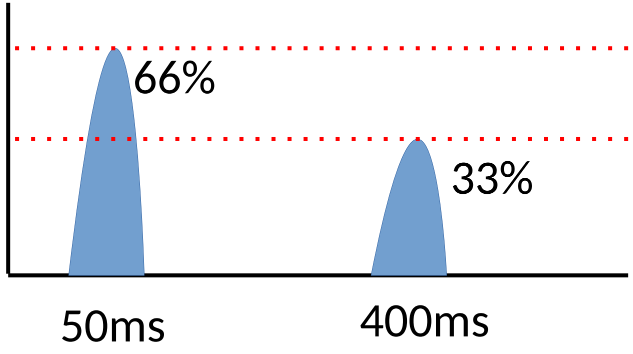
Again, these are commonly tied to SLAs!

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user’s batter? (energy)
  - …
- Averages in these scenarios are simply misleading

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …
- Averages in these scenarios are simply misleading

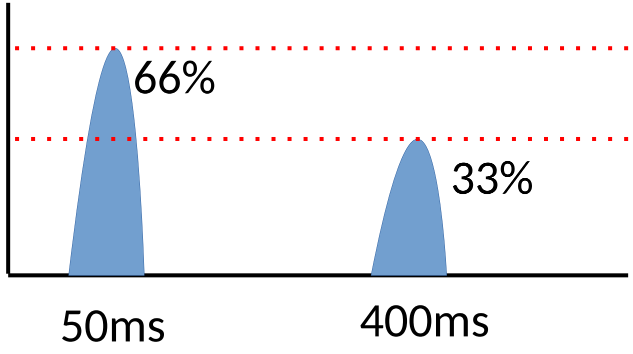


Suppose <200ms response is okay.

An arithmetic mean yields **167ms**.

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …
- Averages in these scenarios are simply misleading



Suppose <200ms response is okay.

An arithmetic mean yields 167ms.

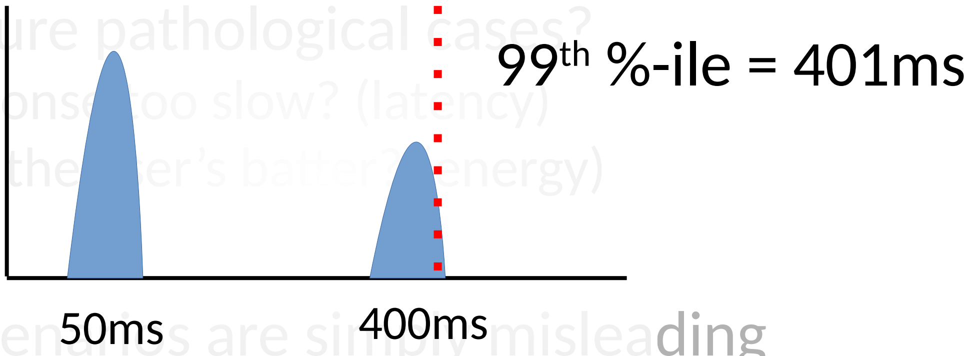
But **1/3** of responses are bad!

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …
- Averages in these scenarios are simply misleading
- *Percentiles*
  - n^{th} percentile - The score below which n% of a population resides

Summary Statistics

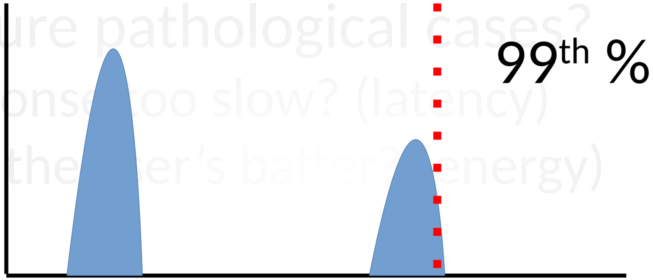
- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's battery? (energy)
  - …



- Averages in these scenarios are simply misleading
- *Percentiles*
  - n^{th} percentile - The score below which n% of a population resides

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's battery? (energy)
  - …



- Averages in these scenarios are simply misleading
- *Percentiles*
  - nth percentile - The score below which n% of a population resides

Percentiles better capture adherence to minimum standards.

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …
- Averages in these scenarios are simply misleading
- *Percentiles*
  - n^{th} percentile - The score below which n% of a population resides
- Even percentiles can be misleading when misused

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …
- Averages in these scenarios are simply misleading
- *Percentiles*
  - n^{th} percentile - The score below which n% of a population resides
- Even percentiles can be misleading when misused

How many server requests does a web page make?

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …
- Averages in these scenarios are simply misleading
- *Percentiles*
  - n^th percentile - The score below which n% of a population resides
- Even percentiles can be misleading when misused

How many server requests does a web page make?

Median of 69-75 [Hubspot 2019]

p(99th %-ile experience) = $1-0.99^{75}$ ~ **0.5**

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …
- Averages in these scenarios are simply misleading
- *Percentiles*
  - n^{th} percentile - The score below which n% of a population resides
- Even percentiles can be misleading when misused

How many server requests does a web page make?

How do you measure percentiles over time?

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …
- Averages in these scenarios are simply misleading
- *Percentiles*
  - n^{th} percentile - The score below which n% of a population resides
- Even percentiles can be misleading when misused

For more see:

How not to measure latency
Latency SLOs Done Right

Summary Statistics

- What if the goal is not to measure tendency, but instead to measure pathological cases?
  - Is my web site response too slow? (latency)
  - Does my app drain the user's batter? (energy)
  - …
- Averages in these scenarios are simply misleading
- *Percentiles*
  - n^{th} percentile - The score below which n% of a population resides
- Even percentiles can be misleading when misused
- Typical SLIs, SLOs, & SLAs are driven by percentiles.
  - These become your contractual obligations!

Summary Statistics

- At the end of the day,
  you cannot sit down and follow a boilerplate process.

Summary Statistics

- At the end of the day,
  you cannot sit down and follow a boilerplate process.
- Assess the goal. Assess the data. Determine what is meaningful.

Benchmarking

- In practice applying good benchmarking & statistics is made easier via frameworks
  - Google benchmark (C & C++)
  - Google Caliper (Java)
  - JMH (Java)
  - Nonius
  - Celero
  - Easybench
  - Pyperf
  - …

Investigating Performance

Profiling

- When benchmark results do not make sense, you should investigate ***why***

Profiling

- When benchmark results do not make sense, you should investigate why
  - For resource X, where is X being *used*, *acquired*, and or *released*?

Profiling

- When benchmark results do not make sense, you should investigate why
  - For resource X, where is X being *used*, *acquired*, and or *released*?
- Sometimes microbenchmarks provide sufficient insight

Profiling

- When benchmark results do not make sense, you should investigate why
  - For resource X, where is X being *used*, *acquired*, and or *released*?
- Sometimes microbenchmarks provide sufficient insight
- In other cases you will want to ***profile***

Profiling

- When benchmark results do not make sense, you should investigate why
  - For resource X, where is X being *used*, *acquired*, and or *released*?
- Sometimes microbenchmarks provide sufficient insight
- In other cases you will want to ***profile***
  - Collect additional information about resources in an execution
  - The nature of the tool will depend on the resource and the objective

Profiling

- When benchmark results do not make sense, you should investigate why
  - For resource X, where is X being *used*, *acquired*, and or *released*?
- Sometimes microbenchmarks provide sufficient insight
- In other cases you will want to profile
  - Collect additional information about resources in an execution
  - The nature of the tool will depend on the resource and the objective

You should already be familiar with tools like gprof or jprofile.
We'll examine some more advanced profilers now.

Heap profiling

- Suppose I have a task and it consumes all memory
  - **Note:** This is not hypothetical. This often happens with grad students!

Heap profiling

- Suppose I have a task and it consumes all memory
  - **Note:** This is not hypothetical. This often happens with grad students!
  - If I can identify where & why memory is consumed, I can remediate
    - Maybe better algorithm
    - Maybe competent use of data structures….

Heap profiling

- Suppose I have a task and it consumes all memory
  - **Note:** This is not hypothetical. This often happens with grad students!
  - If I can identify where & why memory is consumed, I can remediate
    - Maybe better algorithm
    - Maybe competent use of data structures….
- Heap profilers track the allocated memory in a program & their provenance

Heap profiling

- Suppose I have a task and it consumes all memory
  - **Note:** This is not hypothetical. This often happens with grad students!
  - If I can identify where & why memory is consumed, I can remediate
    - Maybe better algorithm
    - Maybe competent use of data structures....
- Heap profilers track the allocated memory in a program & their provenance
  - Can identify hotspots, bloat, leaks, short lived allocations, ...

Heap profiling

- Suppose I have a task and it consumes all memory
  - **Note:** This is not hypothetical. This often happens with grad students!
  - If I can identify where & why memory is consumed, I can remediate
    - Maybe better algorithm
    - Maybe competent use of data structures….

- Heap profilers track the allocated memory in a program & their provenance
  - Can identify hotspots, bloat, leaks, short lived allocations, …

Some people mistakenly believe that using managed languages like Java prevents these.

In practice, they look different….

leaks, …

bloat, latency spikes, OutOfMemoryError

Heap profiling

- Suppose I have a

Effective Java Item 7
Eliminate obsolete object references.

  - **Note:** This is not hypothetical. This often happens with grad students!
  - If I can ide… emediate
    - Maybe
    - Maybe
- Heap profile… m
  & their prove…
  - Can identi… …

```
public Integer pop() {
  if (size == 0) {
    throw new EmptyStackException();
  }
  size -= 1;
  Integer result = data[size];
  data[size] = null;
  return result
}
```

managed languages like Java prevents these.

In practice, they look different….

leaks, … bloat, latency spikes, OutOfMemoryError

Heap profiling

Effective Java Item 7
Eliminate obsolete object references.

```
public Integer pop() {
  if (size == 0) {
    throw new EmptyStackException();
  }
  size -= 1;
  Integer result = data[size];
  data[size] = null;
  return result
}
```

Very common defect in callbacks & caches
(nullification & deregistration)

- Suppose I have a
  - **Note:** This is not hypothetical. This often happens with grad students!
  - If I can ide … remediate
    - Maybe
    - Maybe
- Heap profile … m & their prove
  - Can identi … …

managed languages like Java prevents these.

In practice, they look different….

leaks, … bloat, latency spikes, OutOfMemoryError

Heap profiling

- Suppose I have a task and it consumes all memory
  - **Note:** This is not hypothetical. This often happens with grad students!
  - If I can identify where & why memory is consumed, I can remediate
    - Maybe better algorithm
    - Maybe competent use of data structures....
- Heap profilers track the allocated memory in a program & their provenance
  - Can identify hotspots, bloat, leaks, short lived allocations, ...
  - Commonly *sample* based, but sometimes *event* based
  - e.g. Massif, Heaptrack, ...

Heap profiling

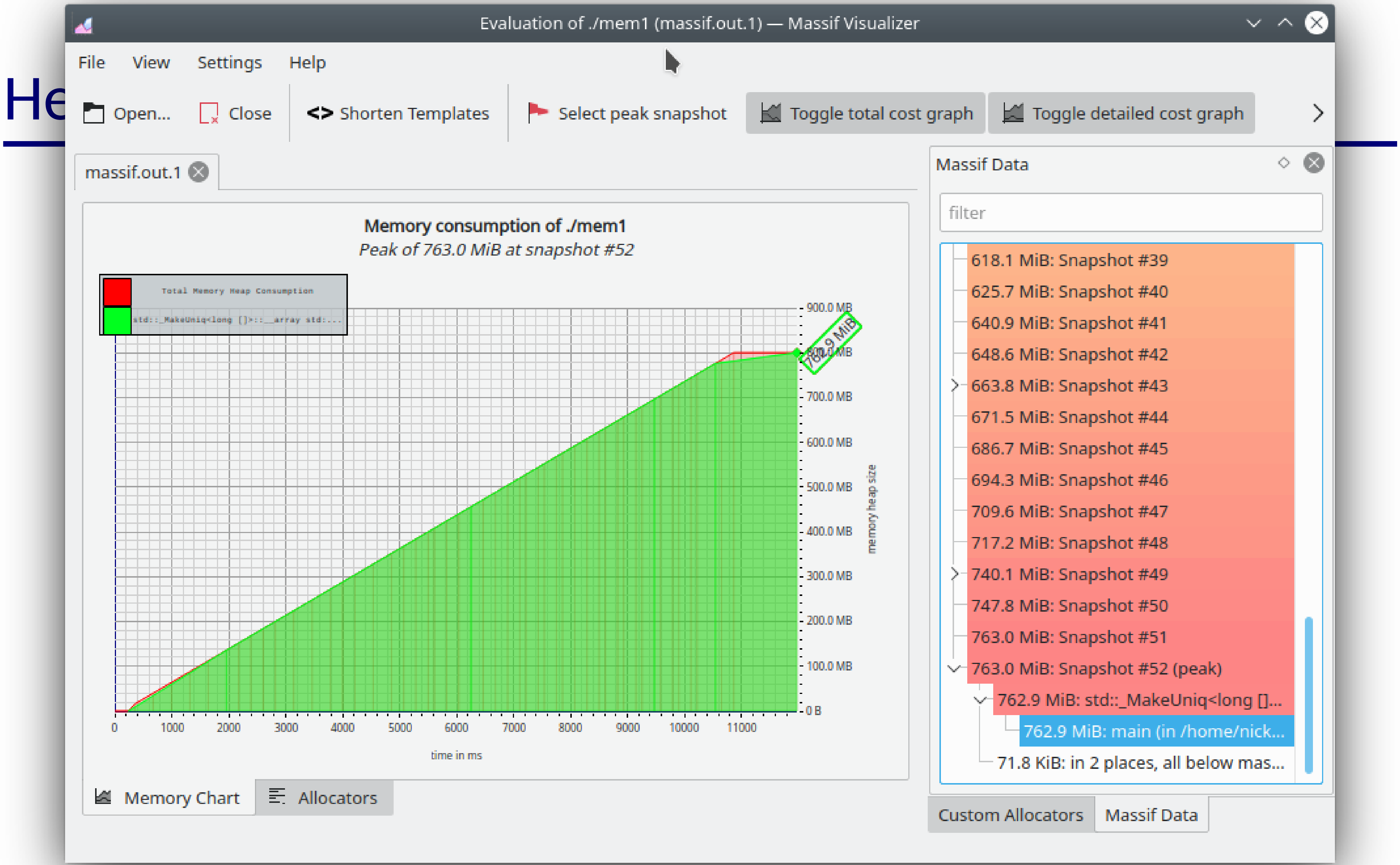
```
int
main() {
  std::vector<std::unique_ptr<long[]>> data{DATA_SIZE};

  for (auto &element : data) {
    element = std::make_unique<long[]>(BLOCK_SIZE);
    // do something with element
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
  }

  std::this_thread::sleep_for(std::chrono::seconds(1));
  return 0;
}
```

Heap profiling

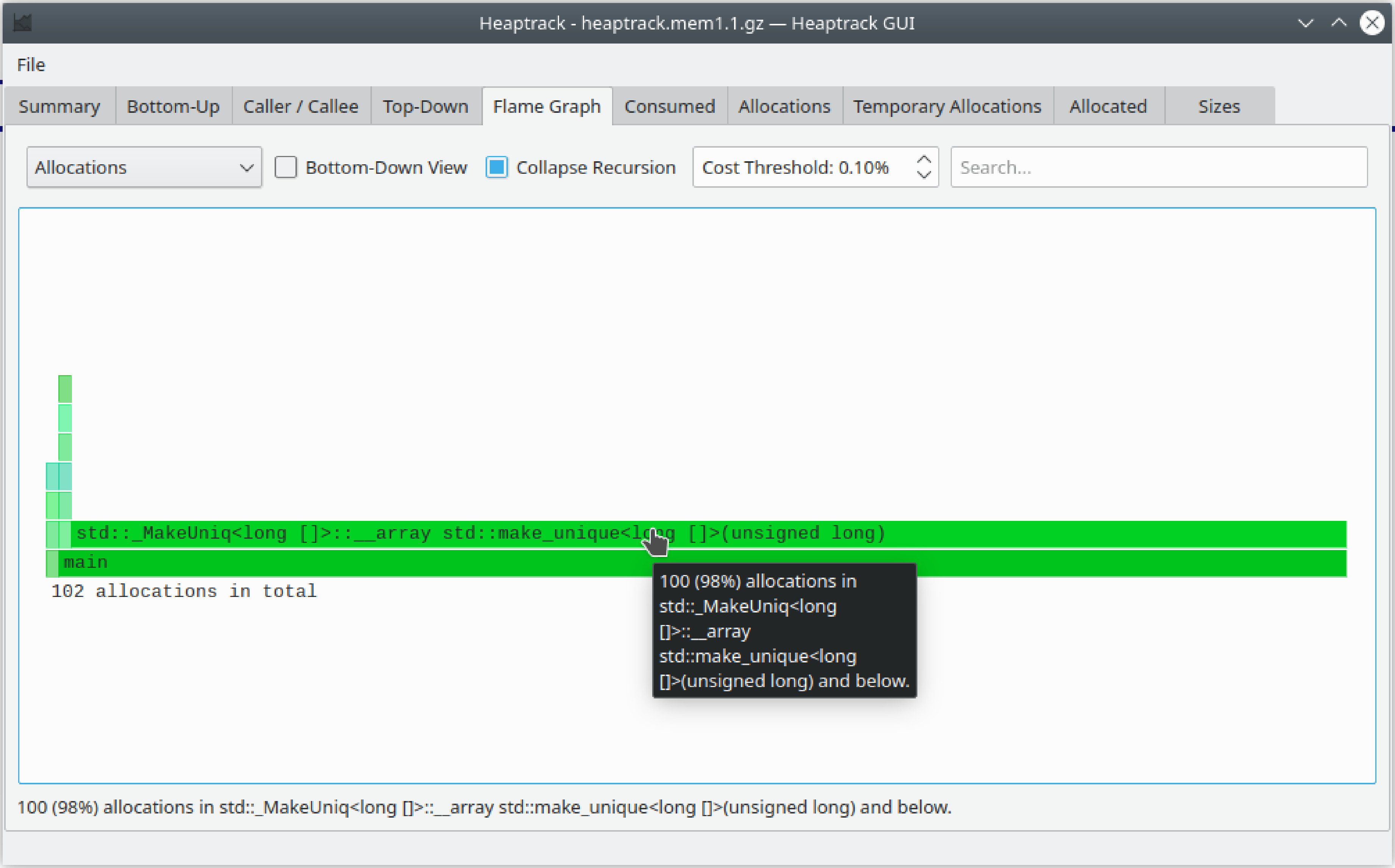
```
int
main() {
  std::vector<std::unique_ptr<long[]>> data{DATA_SIZE};

  for (auto &element : data) {
    element = std::make_unique<long[]>(BLOCK_SIZE);
    // do something with element
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
  }

  std::this_thread::sleep_for(std::chrono::seconds(1));
  return 0;
}
```

valgrind --time-unit=ms --tool=massif <program invocation>
heaptrack <program invocation>

massif-visualizer massif.out.<PID>
heaptrack_gui <path to data>


He
Evaluation of ./mem1 (massif.out.1) — Massif Visualizer
File
View
Settings
Help
Open...
Close
Shorten Templates
Select peak snapshot
Toggle total cost graph
Toggle detailed cost graph
massif.out.1
Memory consumption of ./mem1
Peak of 763.0 MiB at snapshot #52
Total Memory Heap Consumption
std::_MakeUniq<long []>::__array std::...
762.9 MiB
900.0 MB
800.0 MB
700.0 MB
600.0 MB
500.0 MB
400.0 MB
300.0 MB
200.0 MB
100.0 MB
0 B
memory heap size
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
time in ms
Memory Chart
Allocators
Massif Data
filter
618.1 MiB: Snapshot #39
625.7 MiB: Snapshot #40
640.9 MiB: Snapshot #41
648.6 MiB: Snapshot #42
663.8 MiB: Snapshot #43
671.5 MiB: Snapshot #44
686.7 MiB: Snapshot #45
694.3 MiB: Snapshot #46
709.6 MiB: Snapshot #47
717.2 MiB: Snapshot #48
740.1 MiB: Snapshot #49
747.8 MiB: Snapshot #50
763.0 MiB: Snapshot #51
763.0 MiB: Snapshot #52 (peak)
762.9 MiB: std::_MakeUniq<long []...
762.9 MiB: main (in /home/nick...
71.8 KiB: in 2 places, all below mas...
Custom Allocators
Massif Data

Heaptrack - heaptrack.mem1.1.gz — Heaptrack GUI
File
Summary
Bottom-Up
Caller / Callee
Top-Down
Flame Graph
Consumed
Allocations
Temporary Allocations
Allocated
Sizes
900.0 MB
800.0 MB
700.0 MB
600.0 MB
500.0 MB
400.0 MB
300.0 MB
200.0 MB
100.0 MB
0 B
Memory Consumed
0s
1s
2s
3s
4s
5s
6s
7s
8s
9s
10s
11s
12s
Elapsed Time

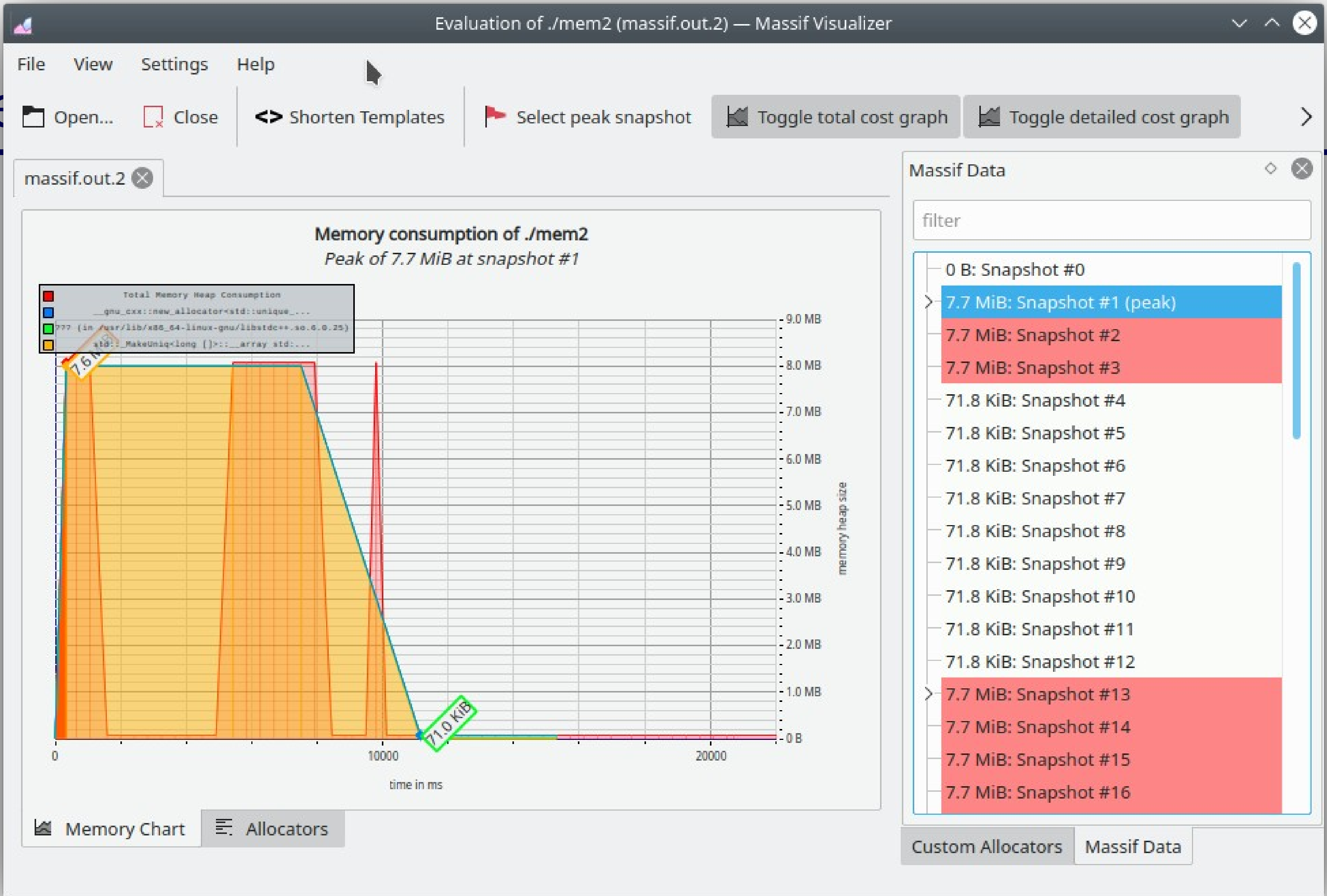

Heaptrack - heaptrack.mem1.1.gz — Heaptrack GUI
File
Summary
Bottom-Up
Caller / Callee
Top-Down
Flame Graph
Consumed
Allocations
Temporary Allocations
Allocated
Sizes
Allocations
Bottom-Down View
Collapse Recursion
Cost Threshold: 0.10%
Search...
std::_MakeUniq<long []>::__array std::make_unique<long []>(unsigned long)
main
102 allocations in total
100 (98%) allocations in std::_MakeUniq<long []>::__array std::make_unique<long []>(unsigned long) and below.
100 (98%) allocations in std::_MakeUniq<long []>::__array std::make_unique<long []>(unsigned long) and below.

Heap profiling

```
int
main() {
  std::vector<std::unique_ptr<long[]>> data{DATA_SIZE};

  for (auto &element : data) {
    element = std::make_unique<long[]>(BLOCK_SIZE);
    // do something with element
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    element.reset();
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
  }

  std::this_thread::sleep_for(std::chrono::seconds(1));
  return 0;
}
```

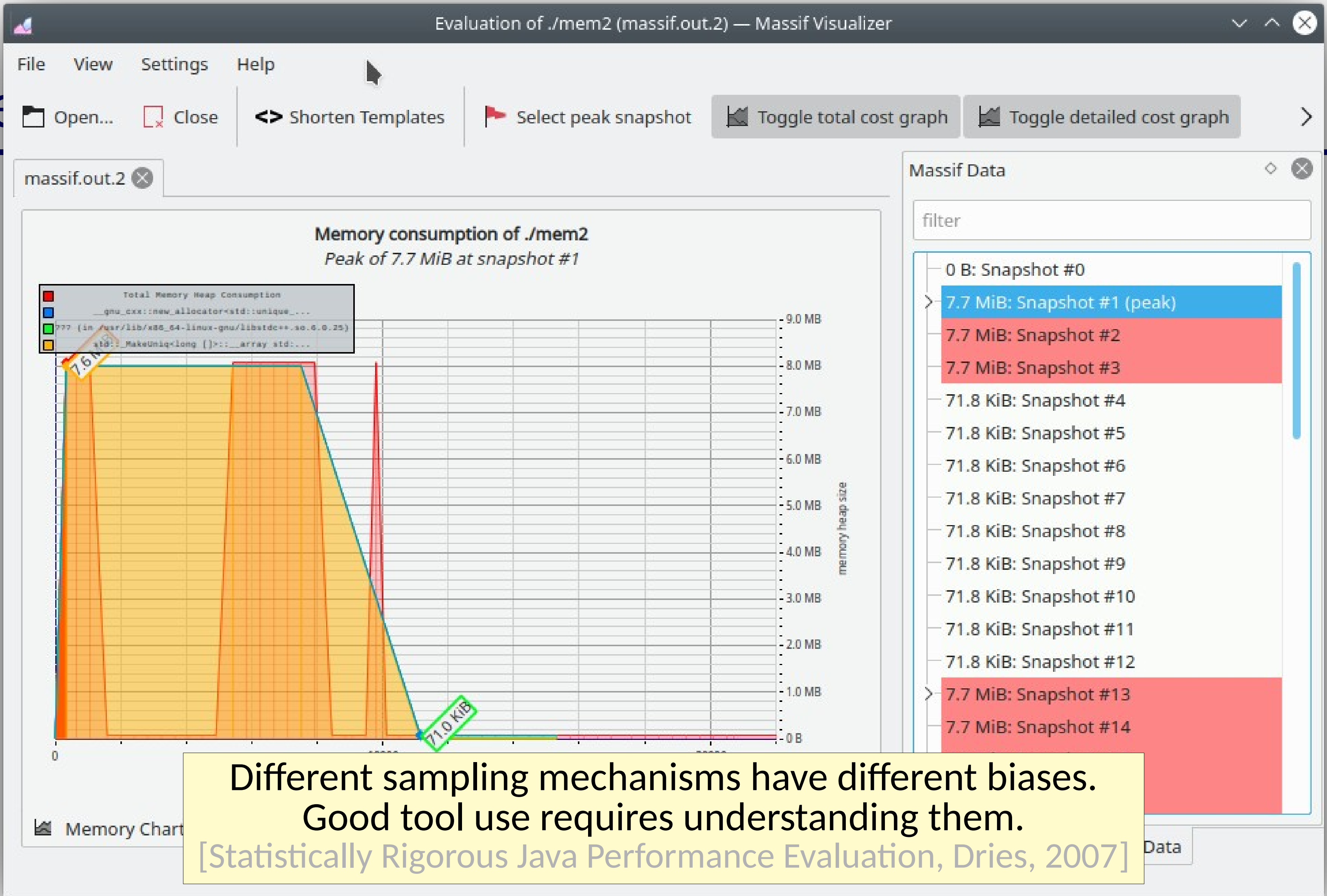
How do we expect this to differ?

He

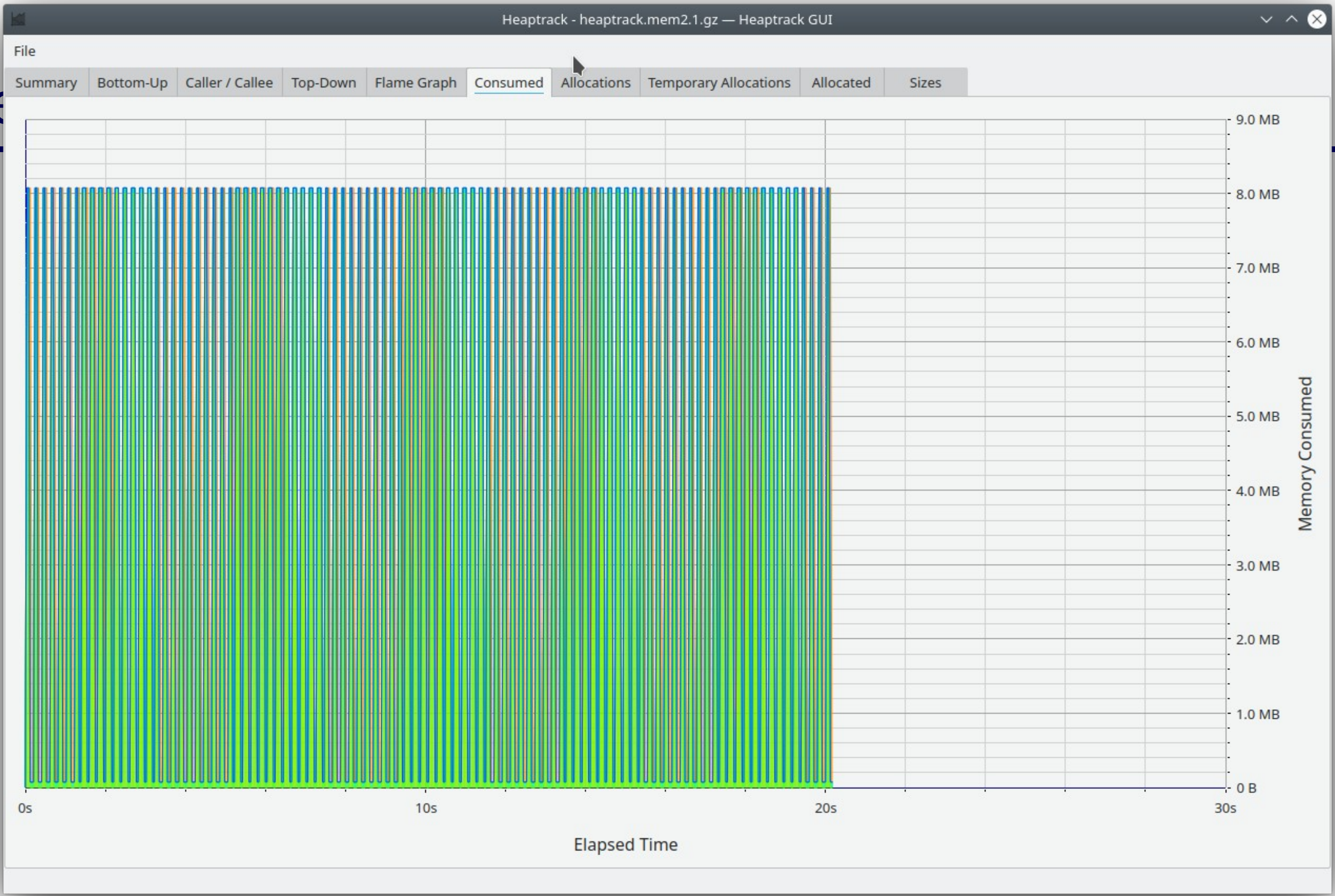


Evaluation of ./mem2 (massif.out.2) — Massif Visualizer
File View Settings Help
Open... Close Shorten Templates Select peak snapshot Toggle total cost graph Toggle detailed cost graph
massif.out.2
Memory consumption of ./mem2
Peak of 7.7 MiB at snapshot #1
Total Memory Heap Consumption
7.6 MiB
71.0 KiB
9.0 MB
8.0 MB
7.0 MB
6.0 MB
5.0 MB
4.0 MB
3.0 MB
2.0 MB
1.0 MB
0 B
memory heap size
0
10000
20000
time in ms
Memory Chart
Allocators
Massif Data
filter
0 B: Snapshot #0
7.7 MiB: Snapshot #1 (peak)
7.7 MiB: Snapshot #2
7.7 MiB: Snapshot #3
71.8 KiB: Snapshot #4
71.8 KiB: Snapshot #5
71.8 KiB: Snapshot #6
71.8 KiB: Snapshot #7
71.8 KiB: Snapshot #8
71.8 KiB: Snapshot #9
71.8 KiB: Snapshot #10
71.8 KiB: Snapshot #11
71.8 KiB: Snapshot #12
7.7 MiB: Snapshot #13
7.7 MiB: Snapshot #14
7.7 MiB: Snapshot #15
7.7 MiB: Snapshot #16
Custom Allocators
Massif Data

He



Different sampling mechanisms have different biases.
Good tool use requires understanding them.
[Statistically Rigorous Java Performance Evaluation, Dries, 2007]



Heaptrack - heaptrack.mem2.1.gz — Heaptrack GUI
File
Summary
Bottom-Up
Caller / Callee
Top-Down
Flame Graph
Consumed
Allocations
Temporary Allocations
Allocated
Sizes
9.0 MB
8.0 MB
7.0 MB
6.0 MB
5.0 MB
4.0 MB
3.0 MB
2.0 MB
1.0 MB
0 B
Memory Consumed
0s
10s
20s
30s
Elapsed Time

CPU Profiling & Flame Graphs

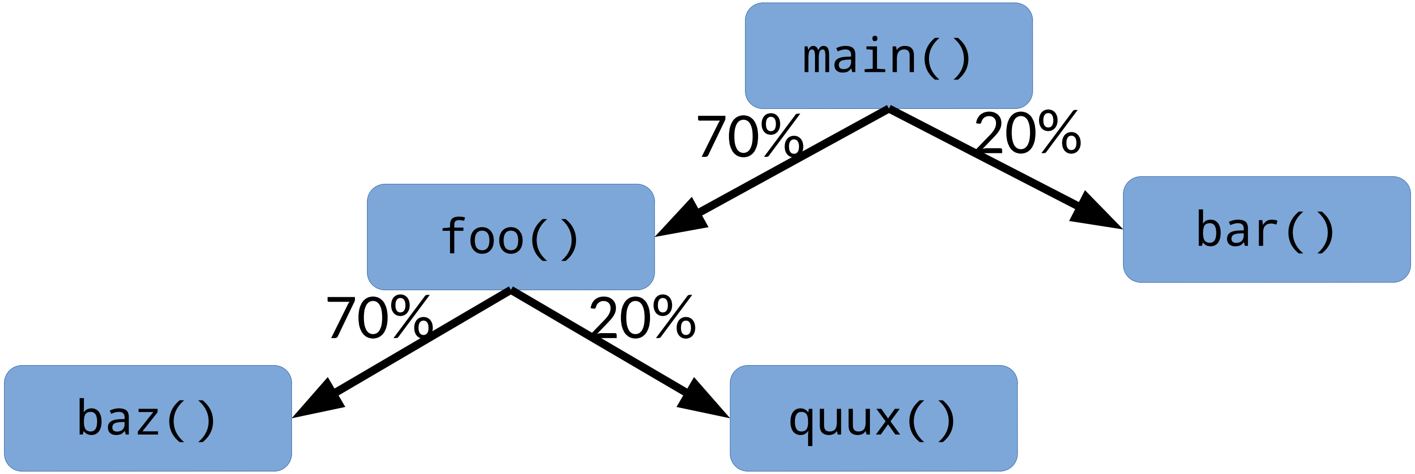
- When CPU is the resource, investigate where the CPU is spent
  - Classic profilers – gprof, oprofile, jprof, …

CPU Profiling & Flame Graphs

- When CPU is the resource, investigate where the CPU is spent
  - Classic profilers – gprof, oprofile, jprof, ...
- Classic CPU profilers capture a lot of data and force the user to explore & explain it manually

CPU Profiling & Flame Graphs

- When CPU is the resource, investigate where the CPU is spent
  - Classic profilers – gprof, oprofile, jprof, ...
- Classic CPU profilers capture a lot of data and force the user to explore & explain it manually

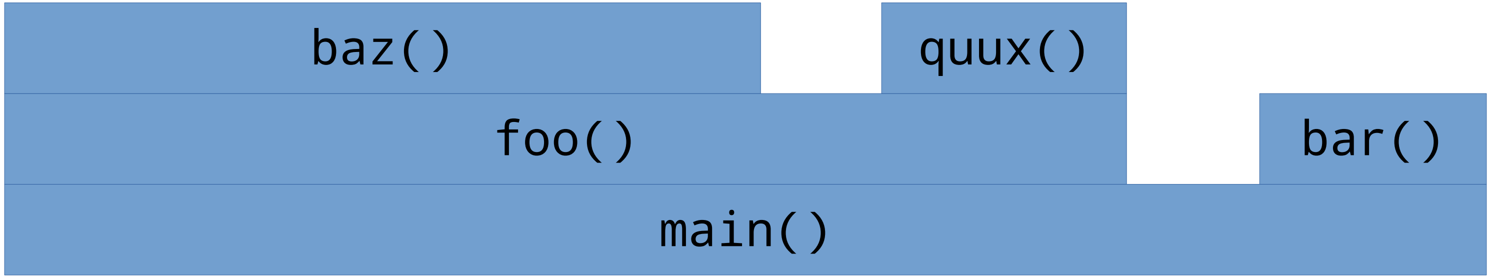


CPU Profiling & Flame Graphs

- When CPU is the resource, investigate where the CPU is spent
  - Classic profilers – gprof, oprofile, jprof, …
- Classic CPU profilers capture a lot of data and force the user to explore & explain it manually
- Flame graphs provide a way of structuring and visualizing substantial profiling information

CPU Profiling & Flame Graphs

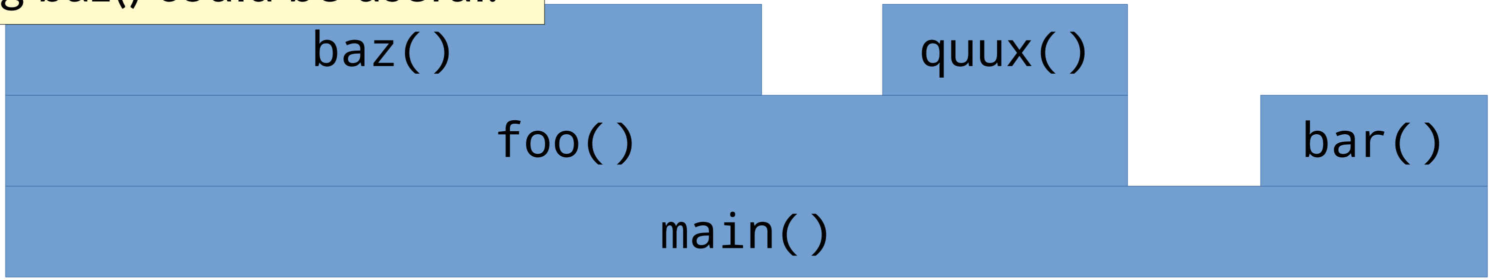
- When CPU is the resource, investigate where the CPU is spent
  - Classic profilers – gprof, oprofile, jprof, …
- Classic CPU profilers capture a lot of data and force the user to explore & explain it manually
- Flame graphs provide a way of structuring and visualizing substantial profiling information



CPU Profiling & Flame Graphs

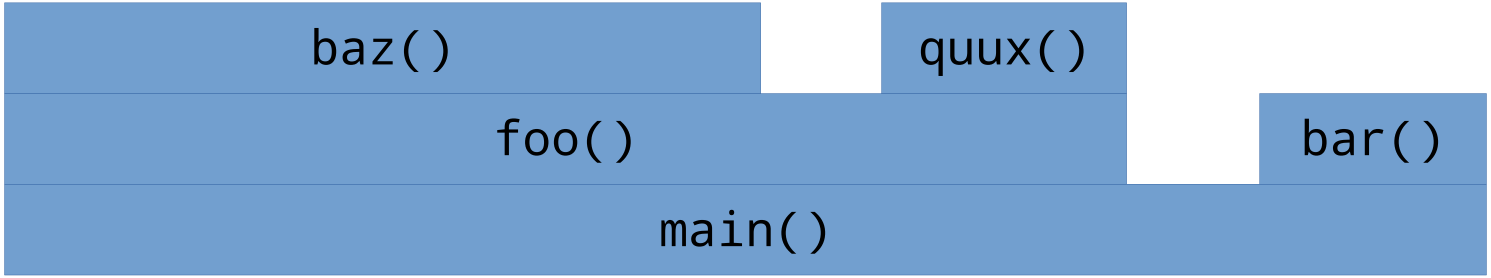
- When CPU is the resource, investigate where the CPU is spent
  - Classic profilers – gprof, oprofile, jprof, …
- Classic CPU profilers capture a lot of data and force the user to explore & explain it manually
- Flame graphs provide a way of structuring and visualizing substantial profiling information

It is easier to see that optimizing baz() could be useful.



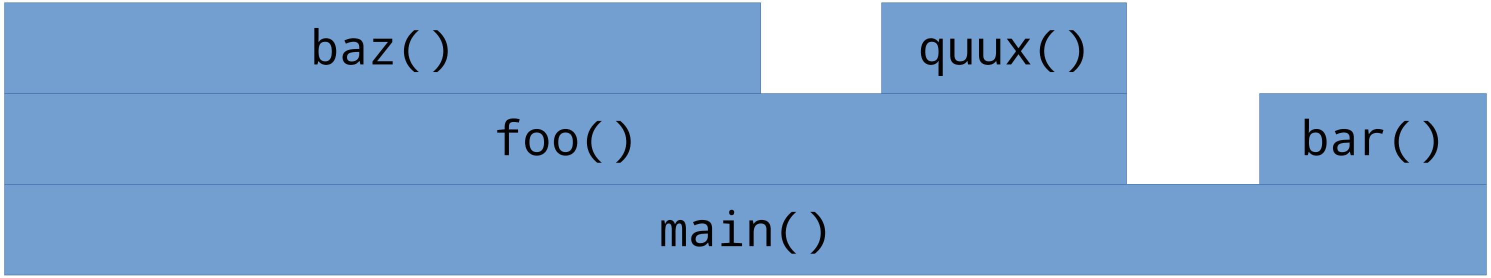
CPU Profiling & Flame Graphs

- When CPU is the resource, investigate where the CPU is spent
  - Classic profilers – gprof, oprofile, jprof, …
- Classic CPU profilers capture a lot of data and force the user to explore & explain it manually
- Flame graphs provide a way of structuring and visualizing substantial profiling information
  - Consumers of CPU on top



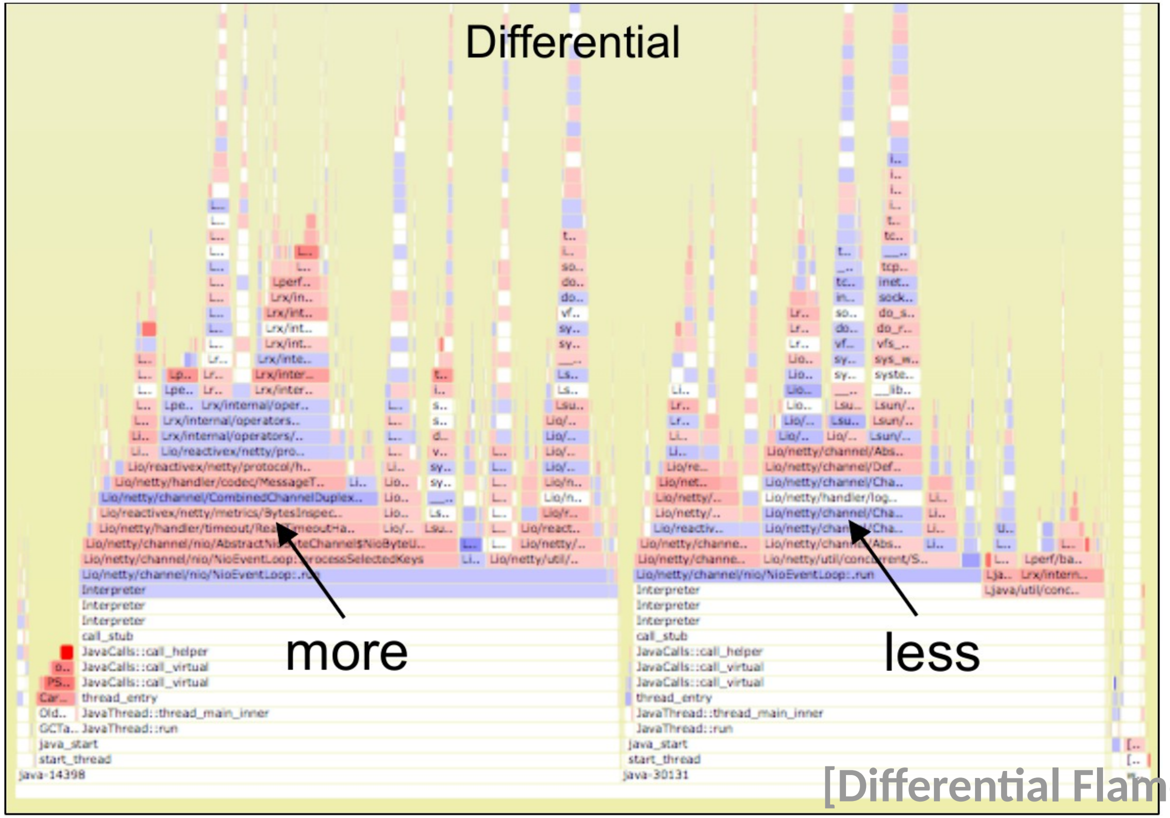
CPU Profiling & Flame Graphs

- When CPU is the resource, investigate where the CPU is spent
  - Classic profilers – gprof, oprofile, jprof, …
- Classic CPU profilers capture a lot of data and force the user to explore & explain it manually
- Flame graphs provide a way of structuring and visualizing substantial profiling information
  - Consumers of CPU on top
  - ancestry, proportions, components can all be clearly identified



CPU Profiling & Flame Graphs

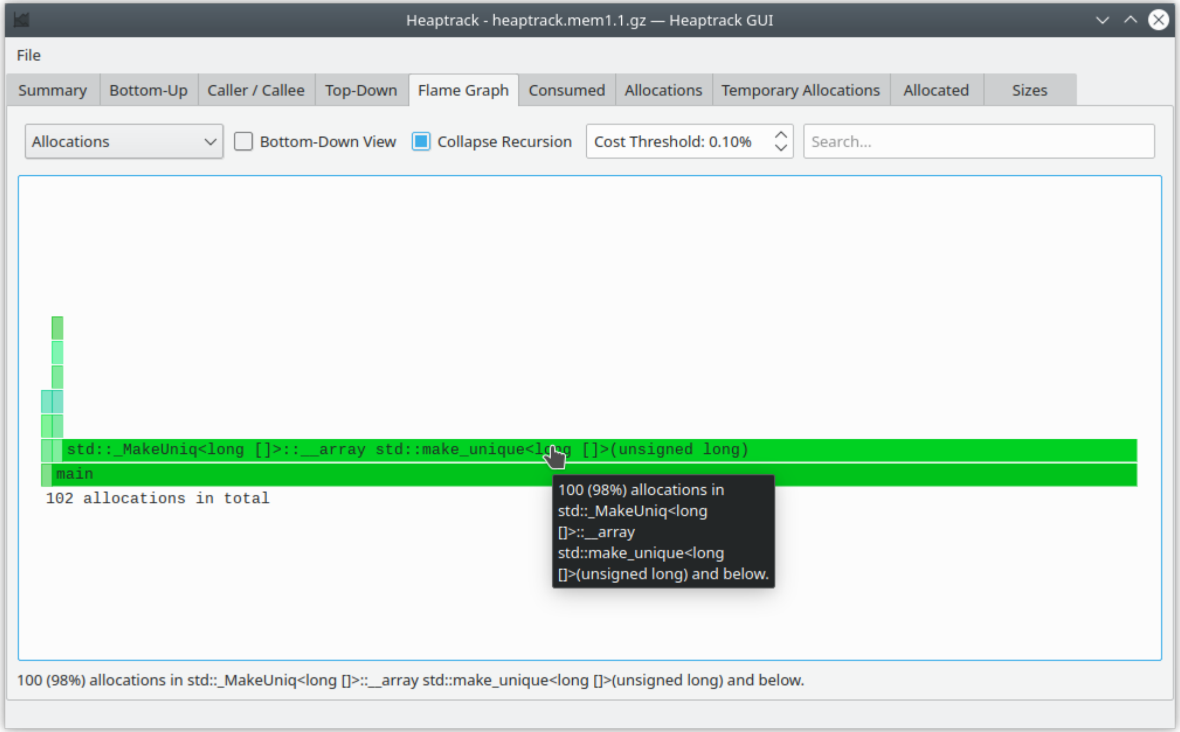
- Can extract rich information by embedding interesting things in colors



[Gregg, ATC 2017]
[Differential Flame Graphs, Gregg, 2014]

CPU Profiling & Flame Graphs

- Flame graphs are not just limited to CPU time!
  - Any countable resource or event can be organized & visualized



CPU Profiling & Flame Graphs

- Flame graphs are not just limited to CPU time!
  - Any countable resource or event can be organized & visualized
- You can also automatically generate them with clang & chrome
  - See project X-Ray in clang
  - See Chrome Trace Viewer

Perf & event profiling

- Sometimes low-level architectural effects determine the performance
  - Cache misses
  - Misspeculations
  - TLB misses

Perf & event profiling

- Sometimes low-level architectural effects determine the performance
  - Cache misses
  - Misspeculations
  - TLB misses

  How well does sample based profiling work for these?

Perf & event profiling

- Sometimes low-level architectural effects determine the performance
  - Cache misses
  - Misspeculations
  - TLB misses

  How well does sample based profiling work for these?

- Instead, we can leverage low level system counters via tools like `perf`

Perf & event profiling

- Sometimes low-level architectural effects determine the performance
  - Cache misses
  - Misspeculations
  - TLB misses

  How well does sample based profiling work for these?

- Instead, we can leverage low level system counters via tools like `perf`

```
perf stat -e <events> <command>
perf record -e <events> -g <command>
perf report
perf annotate
perf list
```

Perf & event profiling

- Sometimes low-level architectural effects determine the performance
  - Cache misses
  - Misspeculations
  - TLB misses

  How well does sample based profiling work for these?

- Instead, we can leverage low level system counters via tools like `perf`

```
perf stat -e <events> <command>
perf record -e <events> -g <command>
perf report
perf annotate
perf list
```

Perf & event profiling

- Sometimes low-level architectural effects determine the performance
  - Cache misses
  - Misspeculations
  - TLB misses

  How well does sample based profiling work for these?

- Instead, we can leverage low level system counters via tools like `perf`

```
perf stat -e <events> <command>
perf record -e <events> -g <command>
perf report
perf annotate
perf list
```

Perf & event profiling

- Sometimes low-level architectural effects determine the performance
  - Cache misses
  - Misspeculations
  - TLB misses

  How well does sample based profiling work for these?

- Instead, we can leverage low level system counters via tools like `perf`

```
perf stat -e <events> <command>
perf record -e <events> -g <command>
perf report
perf annotate
perf list
```

events like

```
task-clock,context-switches,cpu-migrations,page-faults,
cycles,branches,branch-misses,cache-misses,
cycle_activity.stalls_mem_any,icache_64b.iftag_stall
```

Perf & event profiling

```
Sequence s;
size_t count = // size of workload
auto value = randomInts.begin();

while (count) {
  const auto &v = *value;
  auto pos = std::find_if(s.begin(), s.end(),
    [&v] (auto elt) { return !(elt < v); });
  s.insert(pos, v);
  ++value;
  --count;
}
```

Perf & event profiling

```
Sequence s;
size_t count = // size of workload
auto value = randomInts.begin();

while (count) {
  const auto &v = *value;
  auto pos = std::find_if(s.begin(), s.end(),
    [&v] (auto elt) { return !(elt < v); });
  s.insert(pos, v);
  ++value;
  --count;
}
```

Perf & event profiling

```
perf stat -e ... bin/sequenceTest  --benchmark_filter=vector
```

```
 Performance counter stats for 'bin/sequenceTest --benchmark_filter=vector':

               203      page-faults                    #    0.216 K/sec
     3,633,972,445      cycles                         #    3.871 GHz
    11,103,176,853      instructions                   #    3.06  insn per cycle
     3,878,166,469      branches                       # 4130.852 M/sec
            938.83 msec task-clock                     #    0.981 CPUs utilized
                 3      context-switches               #    0.003 K/sec
                 0      cpu-migrations                 #    0.000 K/sec
         1,895,927      branch-misses                  #    0.05% of all branches
           398,844      cache-misses
       135,089,499      cycle_activity.stalls_total #  143.891 M/sec
```

805K insertions/sec @~1 second

Perf & event profiling

```
perf stat -e ... bin/sequenceTest  --benchmark_filter=list
```

```
 Performance counter stats for 'bin/sequenceTest --benchmark_filter=list':

              302      page-faults                      #    0.015 K/sec
   78,686,515,379      cycles                           #    3.953 GHz
   11,813,349,494      instructions                     #    0.15  insn per cycle
    4,695,891,137      branches                         #  235.899 M/sec
        19,906.35 msec task-clock                       #    0.999 CPUs utilized
               76      context-switches                 #    0.004 K/sec
                0      cpu-migrations                   #    0.000 K/sec
        1,344,413      branch-misses                    #    0.03% of all branches
        2,949,410      cache-misses
   73,920,774,866      cycle_activity.stalls_total # 3713.427 M/sec
```

28.7K insertions/sec @~20 seconds

Perf & event profiling

```
perf record -e ... -g bin/sequenceTest  --benchmark_filter=list
perf annotate
```

```
Samples: 177  of event 'cache-misses', 4000 Hz, Event count (approx.): 268564

Percent│      [&v] (auto elt) { return !(elt < v); });
       │        mov        0x0(%r13),%eax
       │      operator()<int>():
       │        nop
       │      __find_if<std::_List_iterator<int>, __gnu_cxx::__ops::_Iter_pred<testNaiveInsert<s
  1.48 │        cmp        %eax,0x10(%rbp)
       │      → jge        17a30 <void testNaiveInsert<std::__cxx11::list<int, std::allocator<in
       │       ZNSt14 List iteratorIiEppEv():
       │       _M_node = _M_node->_M_next;
 85.01 │        mov        0x0(%rbp),%rbp
       │      __find_if<std::_List_iterator<int>, __gnu_cxx::__ops::_Iter_pred<testNaiveInsert<s
 12.13 │        cmp        %r12,%rbp
       │      → jne        17a70 <void testNaiveInsert<std::__cxx11::list<int, std::allocator<in
Press 'h' for help on key
```

85-97% of stalls are on the linked list traversal

Similar profilers across languages

- These sorts of profiling concerns are not just for native code
  - Python – scalene (https://github.com/emeryberger/scalene)
  - Javascript – Chrome Dev Tools, Firebug, JitProf, …
  - Java – VisualVM, Mission Control, XRebel, …
  - …

Similar profilers across languages

- These sorts of profiling concerns are not just for native code
  - Python – scalene (https://github.com/emeryberger/scalene)
  - Javascript – Chrome Dev Tools, Firebug, JitProf, …
  - Java – VisualVM, Mission Control, XRebel, …
  - …

What *events* you care about may change, but the need for profiling is ubiquitous.

Similar profilers across languages

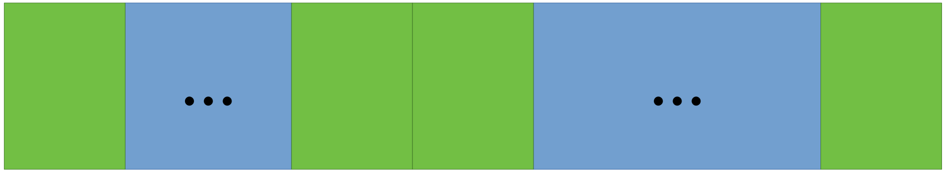
- These sorts of profiling concerns are not just for native code
  - Python – scalene (https://github.com/emeryberger/scalene)
  - Javascript – Chrome Dev Tools, Firebug, JitProf, …
  - Java – VisualVM, Mission Control, XRebel, …
  - …
- As new languages and use cases emerge,
  figuring out *what* to profile &
  developing new profiling tools is critical

Similar profilers across languages

- These sorts of profiling concerns are not just for native code
  - Python – scalene (https://github.com/emeryberger/scalene)
  - Javascript – Chrome Dev Tools, Firebug, JitProf, …
  - Java – VisualVM, Mission Control, XRebel, …
  - …
- As new languages and use cases emerge,
  figuring out *what* to profile &
  developing new profiling tools is critical
- Whatever event, resource, or problem you are interested in,
  a custom profiler can provide a useful investigative tool

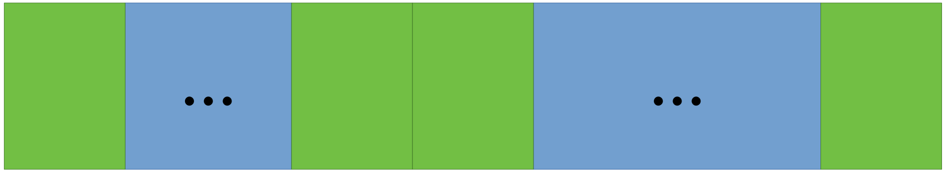
Profiling for *opportunities*

- Causal profiling (e.g. Coz)



Profiling for *opportunities*

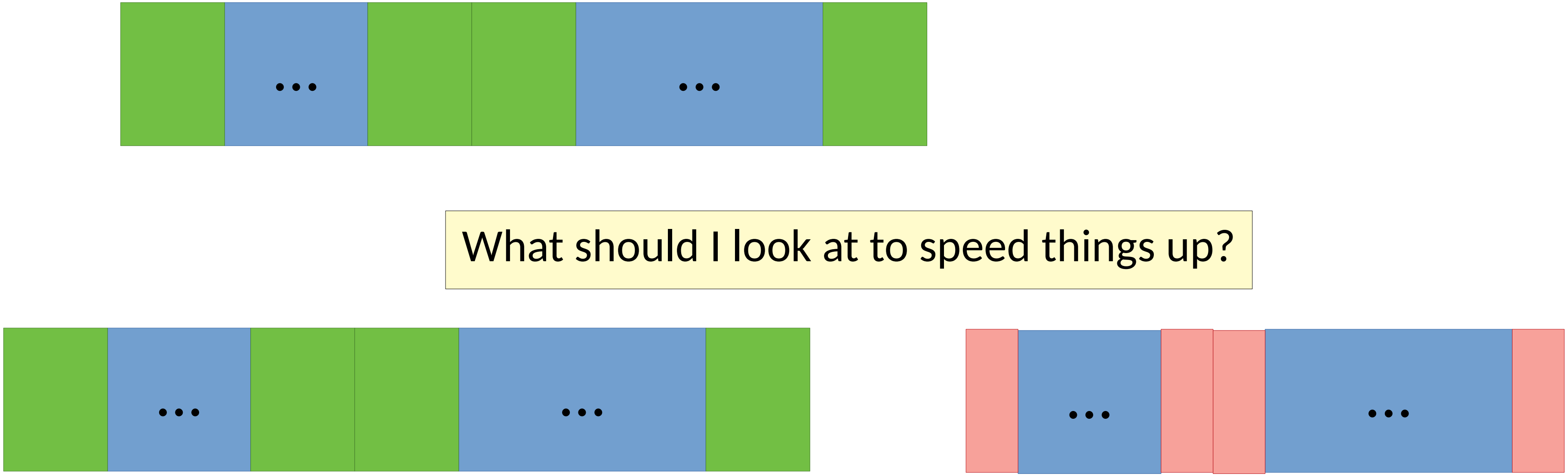
- Causal profiling (e.g. Coz)



What should I look at to speed things up?

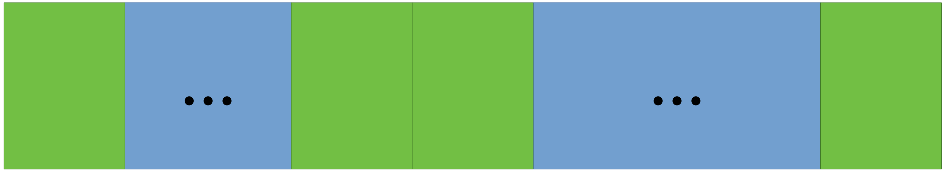
Profiling for *opportunities*

- Causal profiling (e.g. Coz)

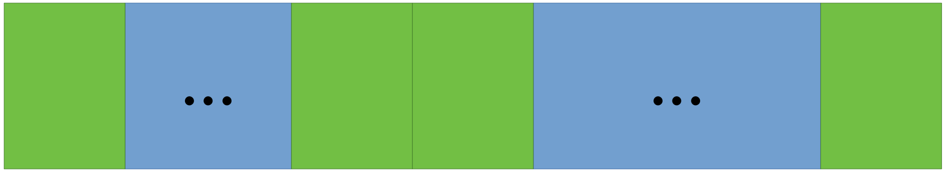


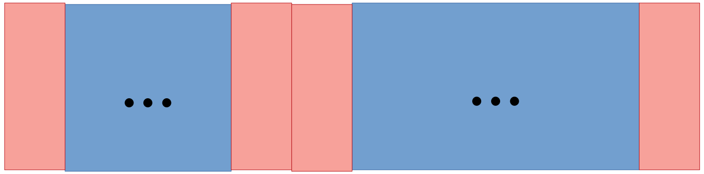
Profiling for *opportunities*

- Causal profiling (e.g. Coz)



What should I look at to speed things up?

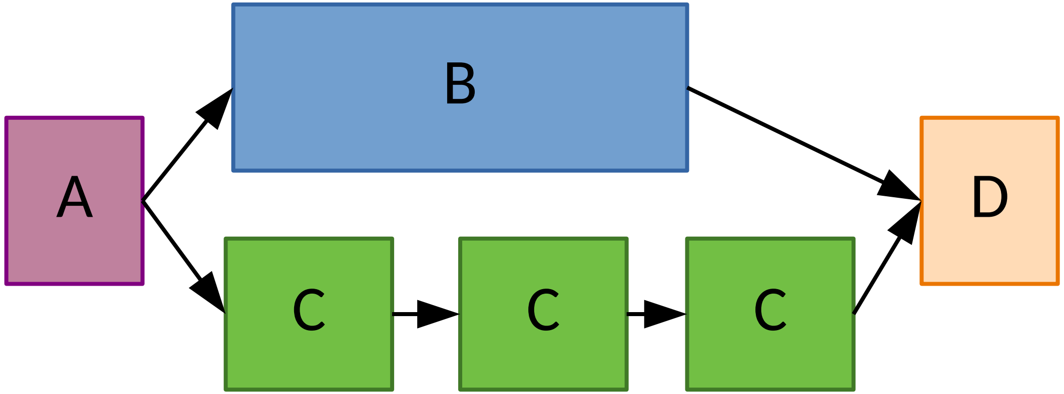




How would you implement such a tool?

Profiling for *opportunities*

- Causal profiling (e.g. Coz)

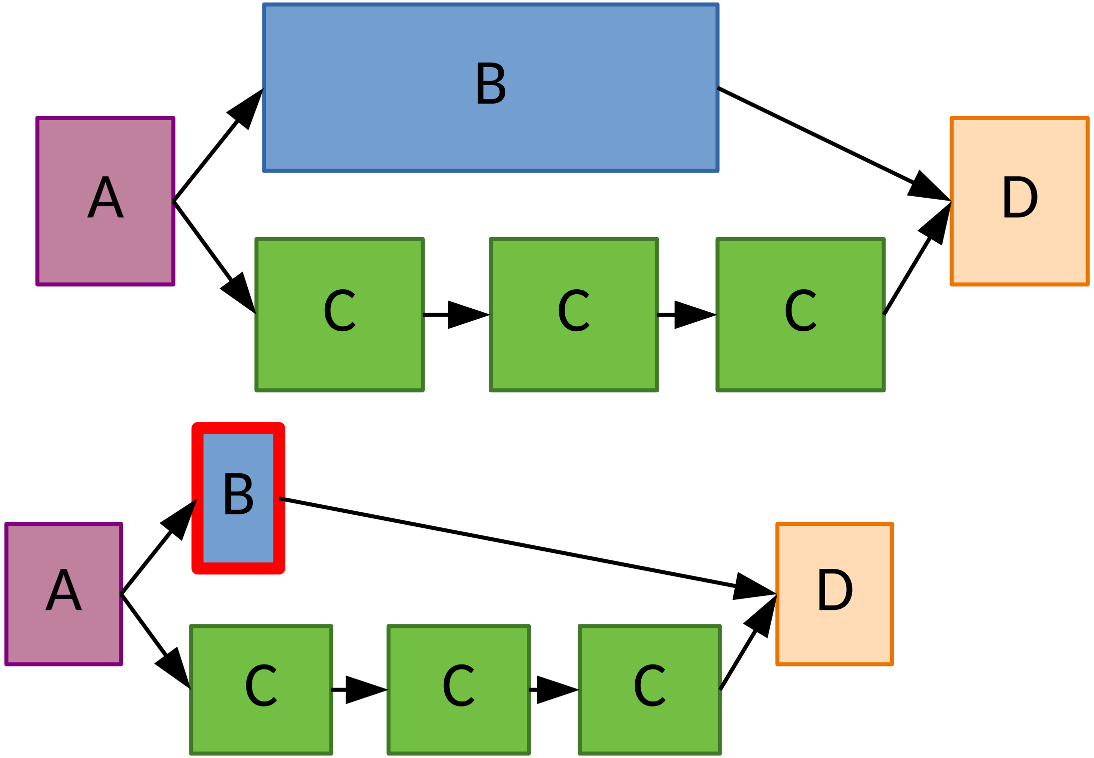


What about parallel code?

Profiling for *opportunities*

- Causal profiling (e.g. Coz)

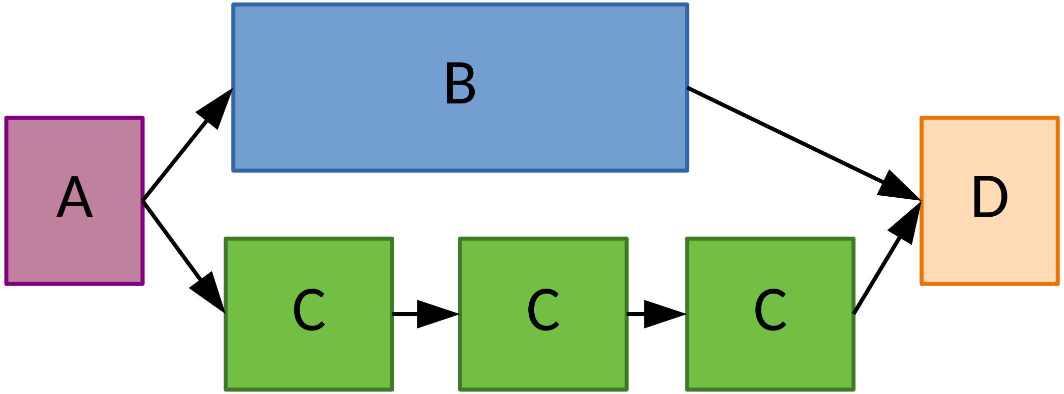
What about parallel code?

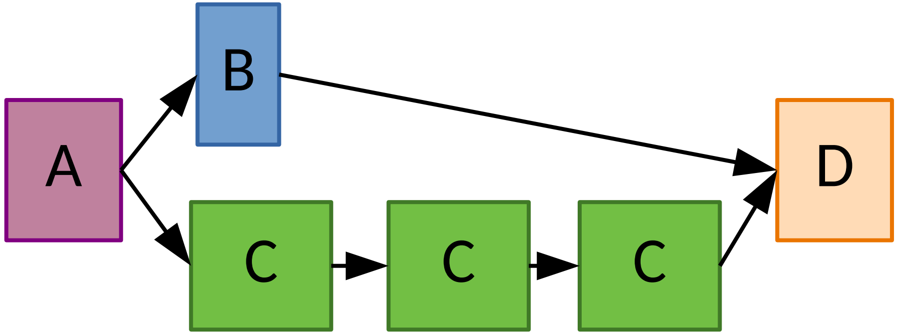


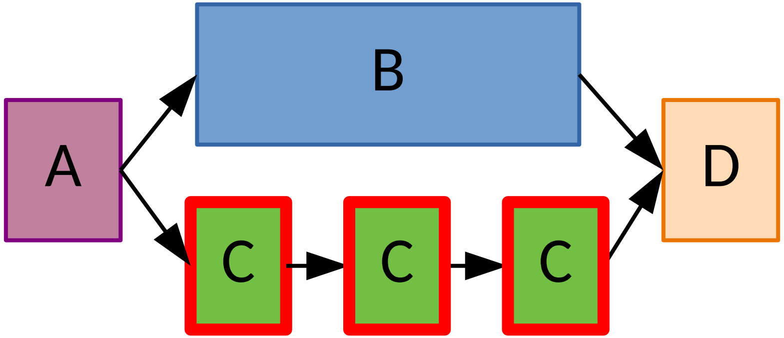
Profiling for *opportunities*

- Causal profiling (e.g. Coz)

What about parallel code?

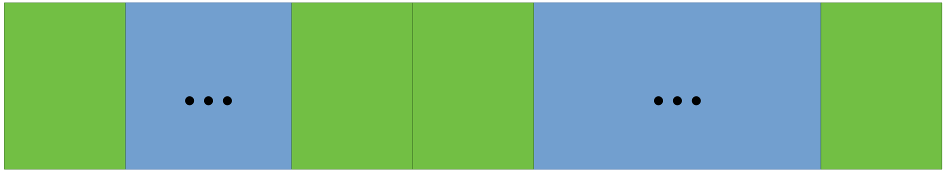






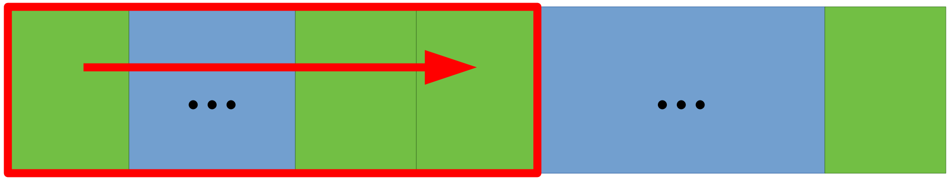
Profiling for *opportunities*

- Causal profiling
- Profiling *for parallelism* (1, 2, 3, 4, 5, …)



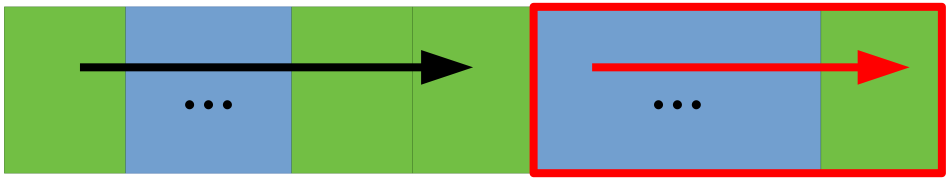
Profiling for *opportunities*

- Causal profiling
- Profiling for parallelism (1, 2, 3, 4, 5, …)



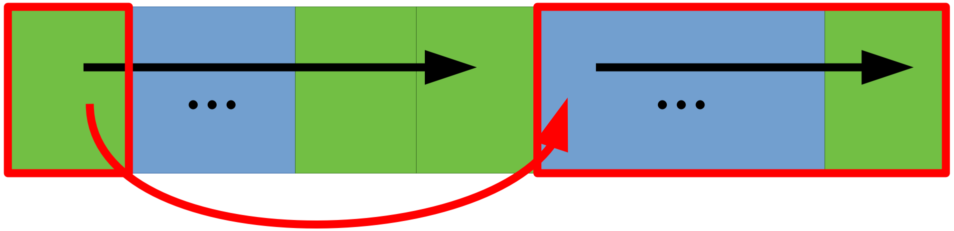
Profiling for *opportunities*

- Causal profiling
- Profiling for parallelism (1, 2, 3, 4, 5, …)



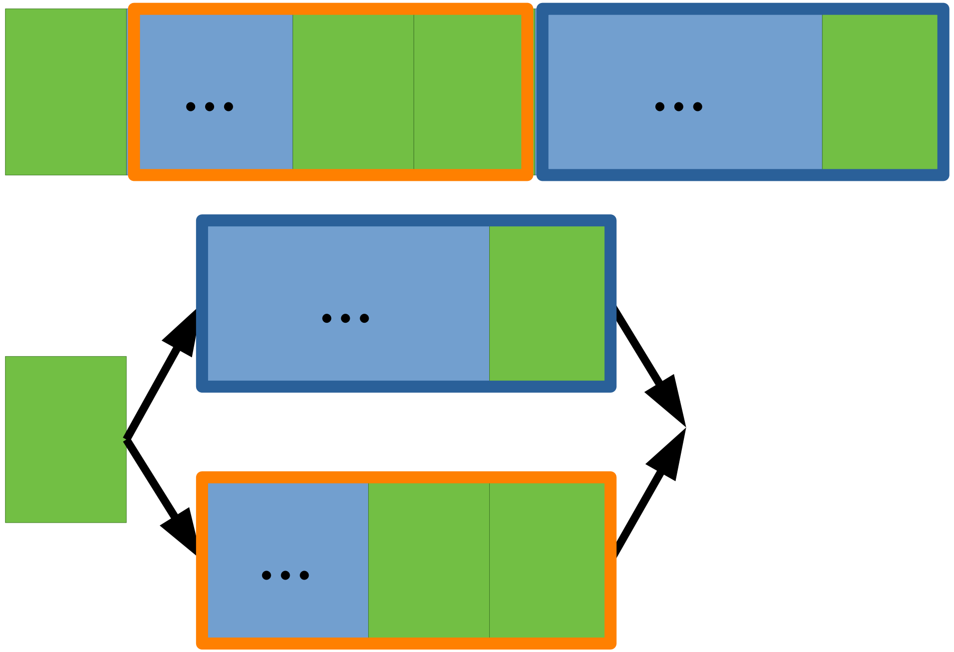
Profiling for *opportunities*

- Causal profiling
- Profiling for parallelism (1, 2, 3, 4, 5, …)



Profiling for *opportunities*

- Causal profiling
- Profiling for parallelism (1, 2, 3, 4, 5, ...)



Improving Performance

Improving Performance

- We can attack performance at several levels

Improving Performance

- We can attack performance at several levels
  - Compilers & tuning the build process

Improving Performance

- We can attack performance at several levels
  - Compilers & tuning the build process
  - Managing the organization of data

Improving Performance

- We can attack performance at several levels
  - Compilers & tuning the build process
  - Managing the organization of data
  - Managing the organization of code

Improving Performance

- We can attack performance at several levels
  - Compilers & tuning the build process
  - Managing the organization of data
  - Managing the organization of code
  - Better algorithms & algorithmic modeling

Improving Performance

- We can attack performance at several levels
  - Compilers & tuning the build process
  - Managing the organization of data
  - Managing the organization of code
  - Better algorithms & algorithmic modeling
- In all cases, we only care about improving performance of *hot code*

Improving Performance

- We can attack performance at several levels
  - Compilers & tuning the build process
  - Managing the organization of data
  - Managing the organization of code
  - Better algorithms & algorithmic modeling
- In all cases, we only care about improving performance of hot code
- Optimizing cold code can *hurt* software
  - You need to understand your trade offs, goals, & business value

Improving Performance

- We can attack performance at several levels
  - Compilers & tuning the build process
  - Managing the organization of data
  - Managing the organization of code
  - Better algorithms & algorithmic modeling
- In all cases, we only care about improving performance of hot code
- Optimizing cold code can *hurt* software
  - You need to understand your trade offs, goals, & business value
  - But also *do not ignore* basic performance awareness

Compiling for performance

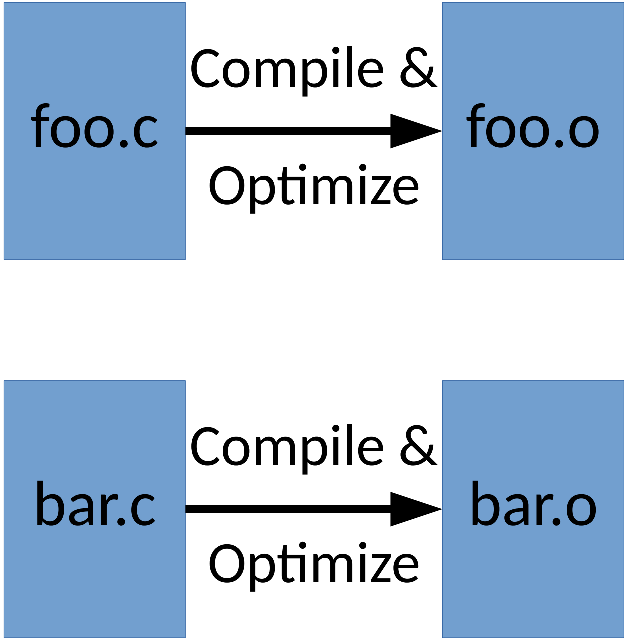
- Enabling optimizations…

Compiling for performance

- Enabling optimizations…
- LTO (Link Time Optimization / Whole Program Optimization)

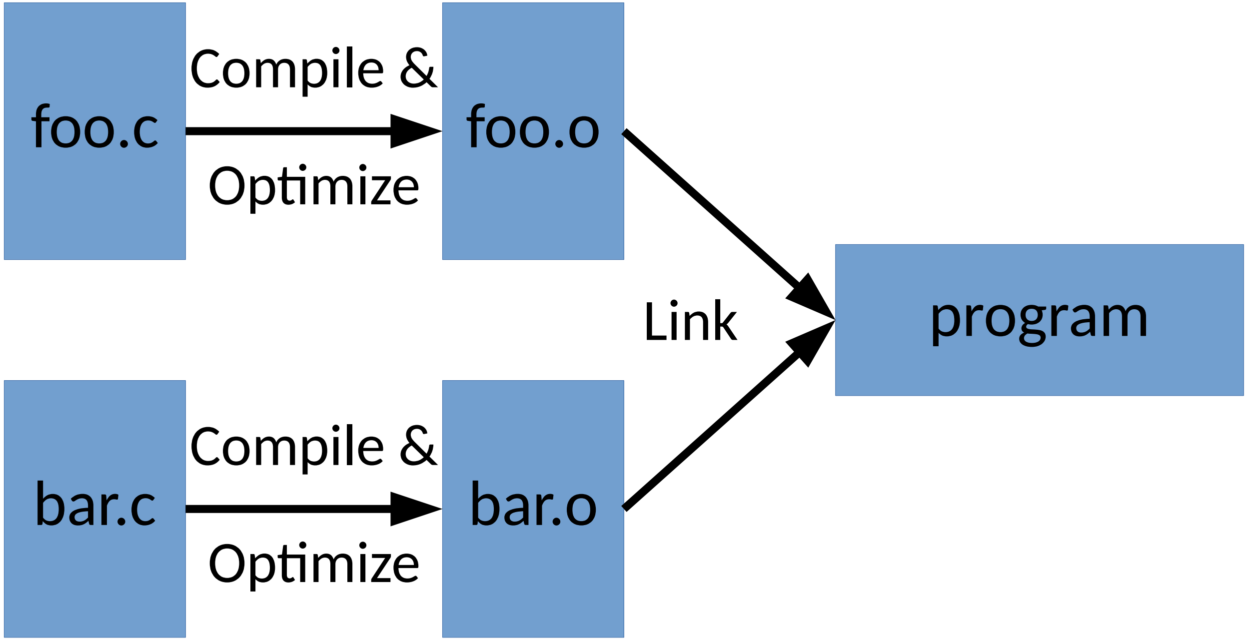
Compiling for performance

- Enabling optimizations…
- LTO (Link Time Optimization / Whole Program Optimization)



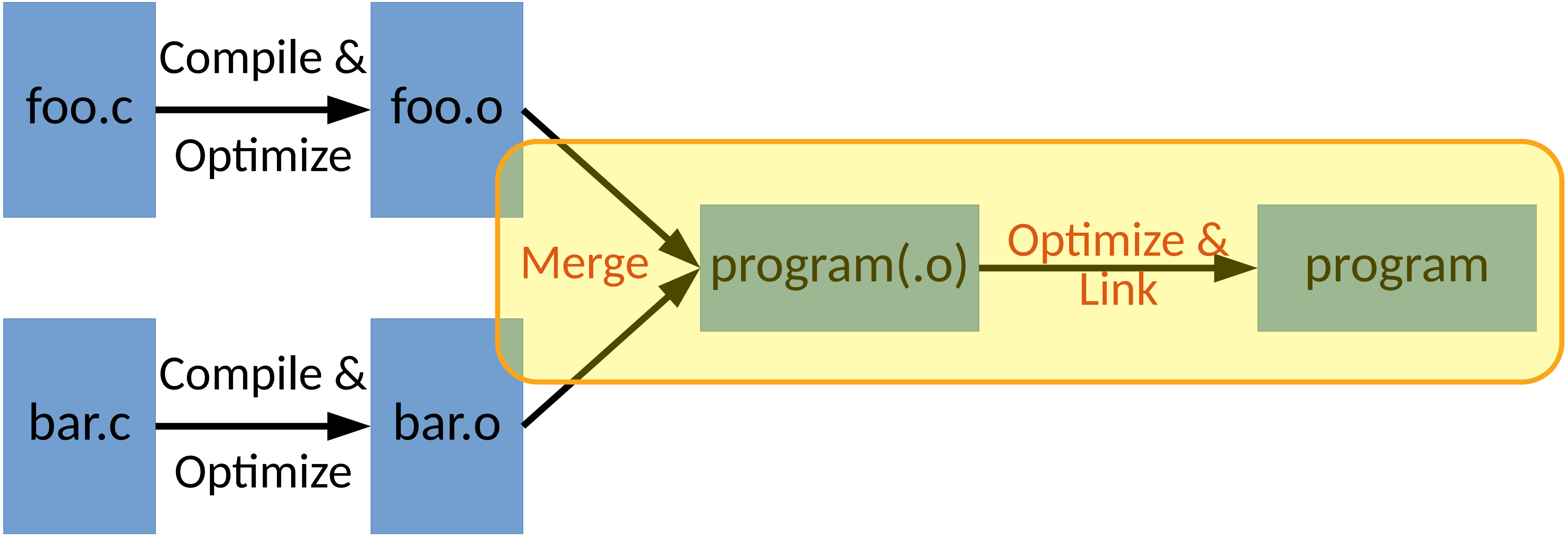
Compiling for performance

- Enabling optimizations...
- LTO (Link Time Optimization / Whole Program Optimization)



Compiling for performance

- Enabling optimizations...
- LTO (Link Time Optimization / Whole Program Optimization)



Compiling for performance

- Enabling optimizations…
- LTO
- PGO/FDO (Profile Guided Optimization/Feedback Directed Optimization)
  - Incorporate profile information in optimization decisions

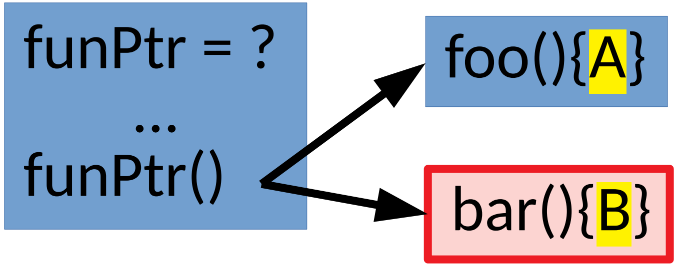
Compiling for performance

- Enabling optimizations…
- LTO
- PGO/FDO (Profile Guided Optimization/Feedback Directed Optimization)
  - Incorporate profile information in optimization decisions

```
funPtr = ?
      …
funPtr()
```

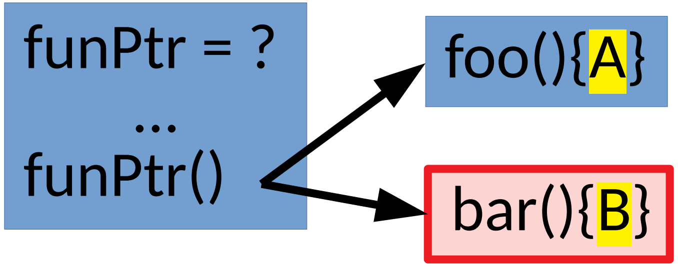
Compiling for performance

- Enabling optimizations…
- LTO
- PGO/FDO (Profile Guided Optimization/Feedback Directed Optimization)
  - Incorporate profile information in optimization decisions



Compiling for performance

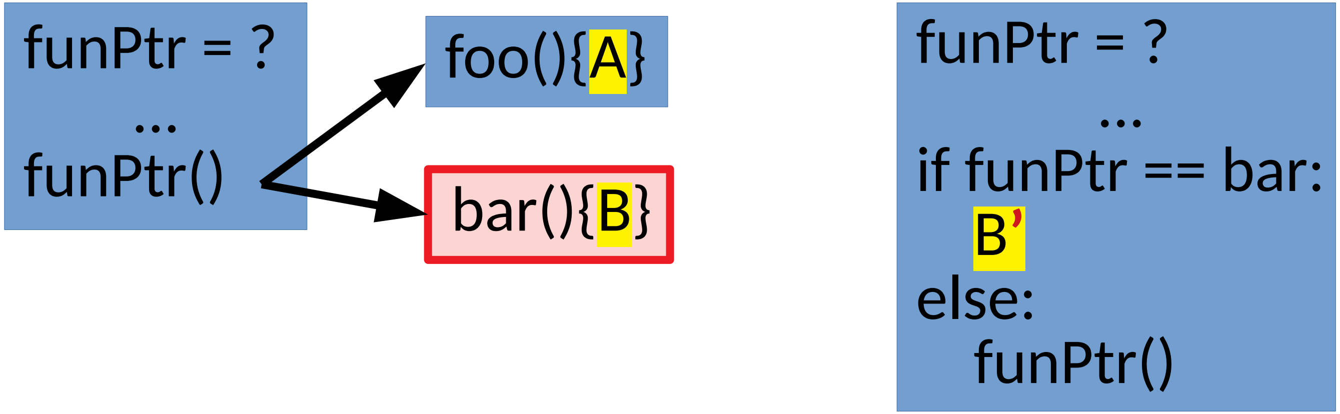
- Enabling optimizations…
- LTO
- PGO/FDO (Profile Guided Optimization/Feedback Directed Optimization)
  - Incorporate profile information in optimization decisions




```
funPtr = ?
      …
if funPtr == bar:
    B'
else:
    funPtr()
```

Compiling for performance

- Enabling optimizations…
- LTO
- PGO/FDO (Profile Guided Optimization/Feedback Directed Optimization)
  - Incorporate profile information in optimization decisions



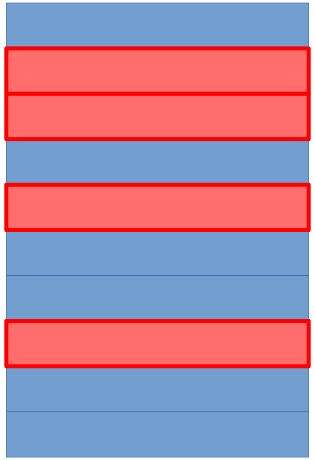
[Visual Studio profile guided optimizations]

Compiling for performance

- Enabling optimizations…
- LTO
- PGO
- Layout optimization (BOLT and otherwise)

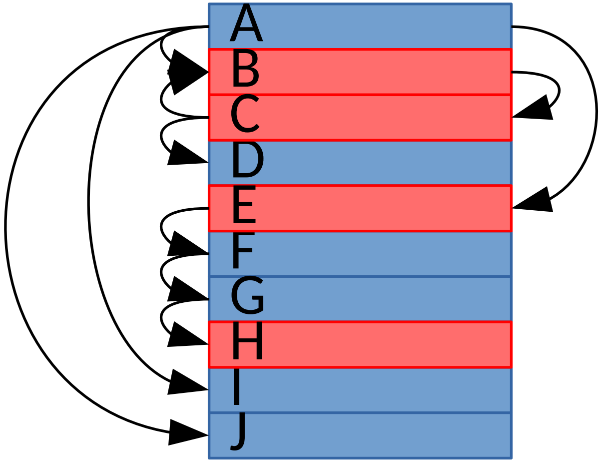
Compiling for performance

- Enabling optimizations…
- LTO
- PGO
- Layout optimization (BOLT and otherwise)



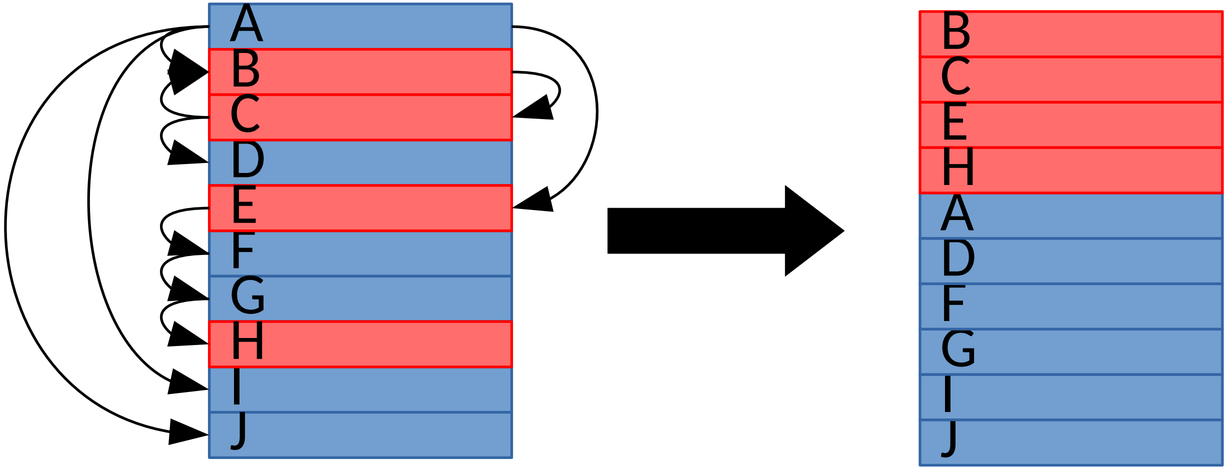
Compiling for performance

- Enabling optimizations…
- LTO
- PGO
- Layout optimization (BOLT and otherwise)



Compiling for performance

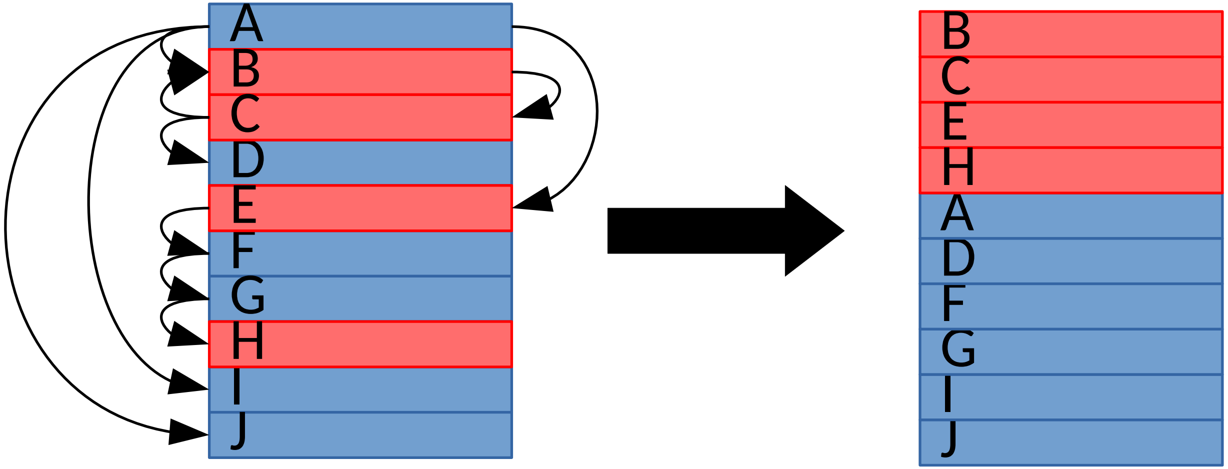
- Enabling optimizations…
- LTO
- PGO
- Layout optimization (BOLT and otherwise)



Compiling for performance

- Enabling optimizations…
- LTO
- PGO
- Layout optimization (BOLT and otherwise)

Google & Facebook found it useful on servers.
Apple has found it useful in embedded devices & apps.
Why? [Hot-Cold Splitting in LLVM]



Compiling for performance

- Enabling optimizations…
- LTO
- PGO
- Layout optimization (BOLT and otherwise)
- Polyhedral analysis, tiling, etc.
  - Transforming complex operations on, e.g., matrices to maximize locality

Compiling for performance

- Enabling optimizations…
- LTO
- PGO
- Layout optimization (BOLT and otherwise)
- Polyhedral analysis, tiling, etc.
  - Transforming complex operations on, e.g., matrices to maximize locality

Even for web apps, these techniques make a difference when applied to the hot path.
[Google Developer Updates]

Optimizing Your Data

- The basic directions of data optimizations

Optimizing Your Data

- The basic directions of data optimizations
  - Ensure the data you *want* is *available* for the tasks you have

Optimizing Your Data

- The basic directions of data optimizations
  - Ensure the data you want is available for the tasks you have
  - Do not spend time processing you do not need

Optimizing Your Data

- The basic directions of data optimizations
  - Ensure the data you want is available for the tasks you have
  - Do not spend time processing you do not need
  - Do not spend extra time managing the data at the system level

Optimizing Your Data

- The basic directions of data optimizations
  - Ensure the data you want is available for the tasks you have
  - Do not spend time processing you do not need
  - Do not spend extra time managing the data at the system level

Several aspects of high level design may be in tension with these

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache

```
struct S1 {
  char a;
};
```

sizeof(S1) == 1

```
struct S2 {
  uint32_t b;
};
```

sizeof(S2) == 4

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache

```
struct S1 {
  char a;
};
```

sizeof(S1) == 1

```
struct S2 {
  uint32_t b;
};
```

sizeof(S2) == 4

```
struct S3 {
  char a;
  uint32_t b;
  char c;
};
```

sizeof(S3) == ?

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache

```
struct S1 {
  char a;
};
```

sizeof(S1) == 1

```
struct S2 {
  uint32_t b;
};
```

sizeof(S2) == 4

```
struct S3 {
  char a;
  uint32_t b;
  char c;
};
```

sizeof(S3) == **12**

uint32_t must be 4 byte aligned.
Padding is inserted!

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache

```
struct S1 {
  char a;
};
```

sizeof(S1) == 1

```
struct S2 {
  uint32_t b;
};
```

sizeof(S2) == 4

```
struct S3 {
  char a;
  uint32_t b;
  char c;
};
```

sizeof(S3) == 12

```
struct S4 {
  char a;
  char c;
  uint32_t b;
};
```

sizeof(S3) == **8**

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache

```
struct S1 {
  char a;
};
```

sizeof(S1) == 1

```
struct S2 {
  uint32_t b;
};
```

sizeof(S2) == 4

```
struct S3 {
  char a;
  uint32_t b;
  char c;
};
```

sizeof(S3) == 12

```
struct S4 {
  char a;
  char c;
  uint32_t b;
};
```

sizeof(S3) == **8**

Careful ordering improves cache utilization

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache
  - Carefully *encoding* data or *reusing* storage can do more

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache
  - Carefully *encoding* data or *reusing* storage can do more
    - Operate on compressed data

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache
  - Carefully *encoding* data or *reusing* storage can do more
    - Operate on compressed data
    - Steal low/high order bits of pointers

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache
  - Carefully *encoding* data or *reusing* storage can do more
    - Operate on compressed data
    - Steal low/high order bits of pointers

```
template <class PointedTo>
class PointerValuePair<PointedTo,int> {
  uintptr_t compact;
  PointedTo* getP() {
    return reinterpret_cast<PointedTo*>(compact & ~0xFFFFFFF8);
  }
  Value getV() { return compact & 0x00000007; }
};
```

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache
  - Carefully *encoding* data or *reusing* storage can do more
    - Operate on compressed data
    - Steal low/high order bits of pointers

```
template <class PointedTo>
class PointerValuePair<PointedTo,int> {
  uintptr_t compact;
  PointedTo* getP() {
    return reinterpret_cast<PointedTo*>(compact & ~0xFFFFFFF8);
  }
  Value getV() { return compact & 0x00000007; }
};
```

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache
  - Carefully *encoding* data or *reusing* storage can do more
    - Operate on compressed data
    - Steal low/high order bits of pointers

```
template <class PointedTo>
class PointerValuePair<PointedTo,int> {
  uintptr_t compact;
  PointedTo* getP() {
    return reinterpret_cast<PointedTo*>(compact & ~0xFFFFFFF8);
  }
  Value getV() { return compact & 0x00000007; }
};
```

Optimizing Your Data

- Basic structure packing
  - Smaller aggregates consume less cache
  - Carefully *encoding* data or *reusing* storage can do more
    - Operate on compressed data
    - Steal low/high order bits of pointers

```
template <class PointedTo>
class PointerValuePair<PointedTo,int> {
  uintptr_t compact;
  PointedTo* getP() {
    return reinterpret_cast<PointedTo*>(compact & ~0xFFFFFFF8);
  }
  Value getV() { return compact & 0x00000007; }
};
```

Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

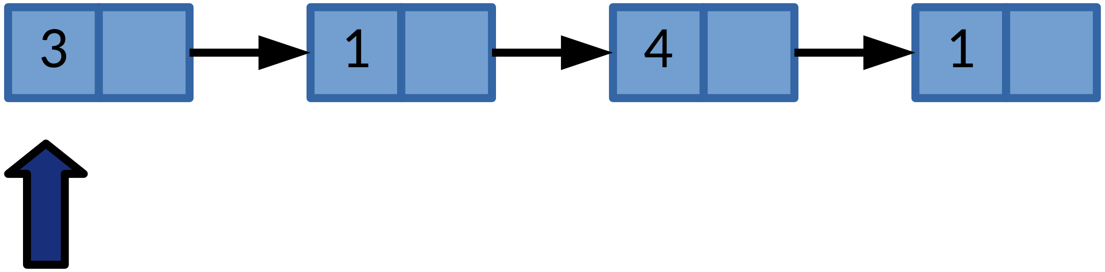
```
std::list numbers = ...
for (auto& i : numbers) {
   ...
}
```

We already saw this.
Traversing a linked list is expensive!

Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

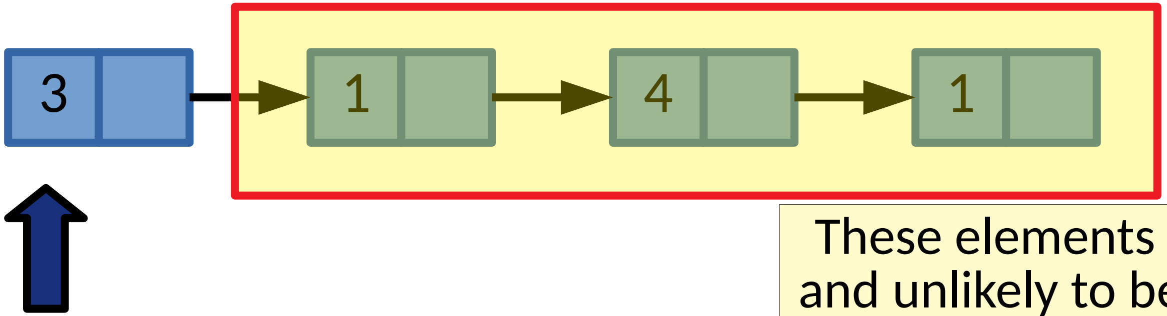
```
std::list numbers = ...
for (auto& i : numbers) {
  ...
}
```



Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

```
std::list numbers = ...
for (auto& i : numbers) {
  ...
}
```

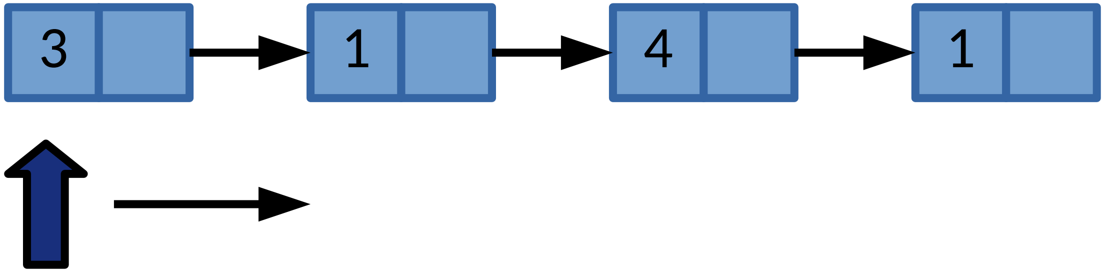


These elements are unlikely to be in cache and unlikely to be prefetched automatically.

Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

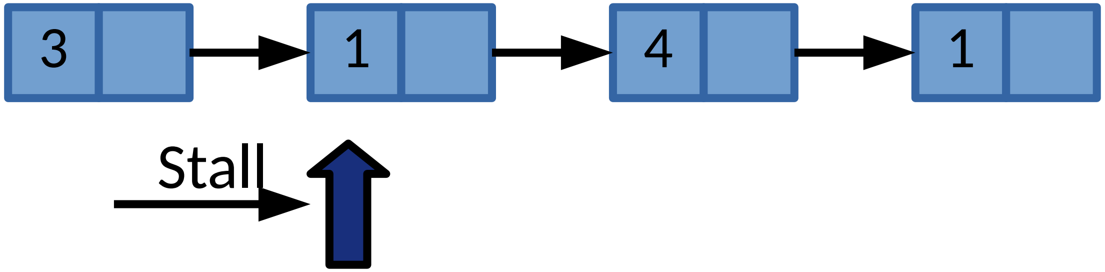
```
std::list numbers = ...
for (auto& i : numbers) {
  ...
}
```



Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

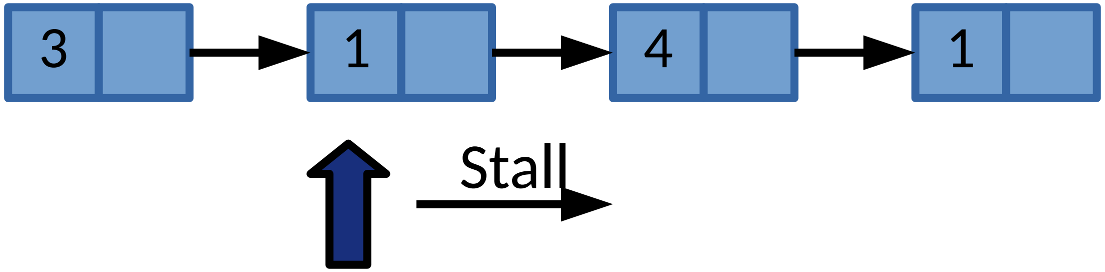
```
std::list numbers = ...
for (auto& i : numbers) {
  ...
}
```



Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

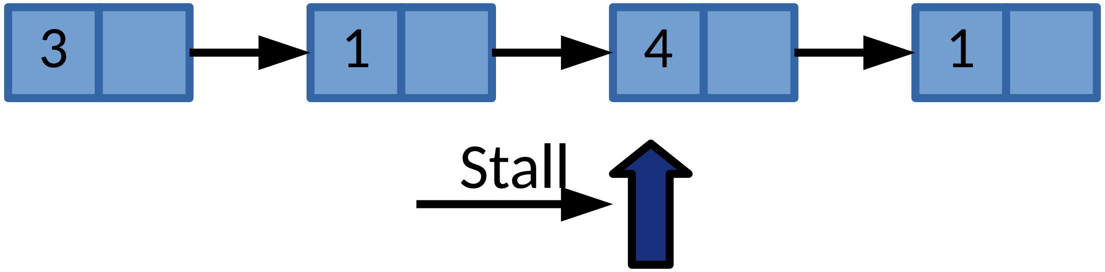
```
std::list numbers = ...
for (auto& i : numbers) {
  ...
}
```



Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

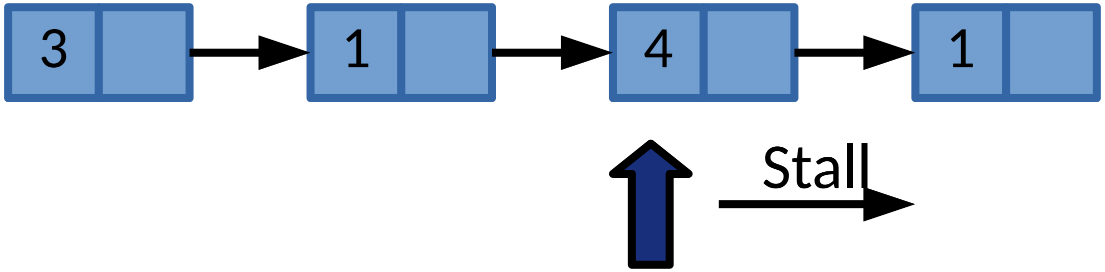
```
std::list numbers = ...
for (auto& i : numbers) {
  ...
}
```



Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

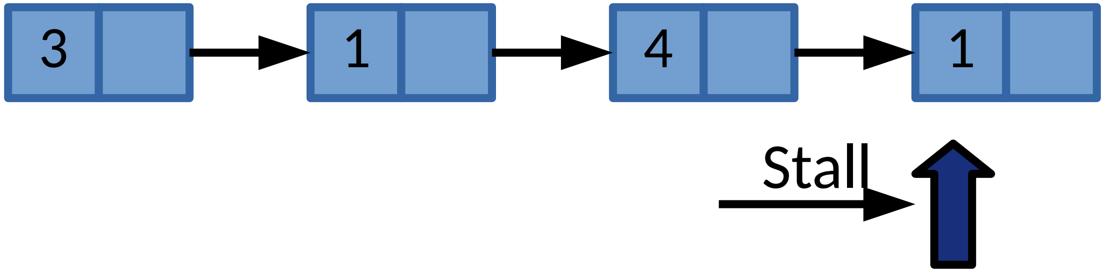
```
std::list numbers = ...
for (auto& i : numbers) {
  ...
}
```



Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

```
std::list numbers = ...
for (auto& i : numbers) {
  ...
}
```



Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

How does this relate to
design tools that we have seen?

Optimizing Your Data

- Managing indirection
  - Pointers and indirection can stall the CPU while waiting on memory

How does this relate to
design tools that we have seen?

How does this relate to
language selection & performance?

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

```
for (Dog& d : dogs) {
  play(d.friendliness, d.hobby);
}
```

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

```
for (Dog& d : dogs) {
  play(d.friendliness, d.hobby);
}
```

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

```
for (Dog& d : dogs) {
  play(d.friendliness, d.hobby);
}
```

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

```
for (Dog& d : dogs) {
  play(d.friendliness, d.hobby);
}
```

We can try to push the cold fields out of the cache

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

```
for (Dog& d : dogs) {
  play(d.friendliness, d.hobby);
}
```

```
struct HotDog {
  double friendliness;
  std::string hobby;
  unique_ptr<Cold> cold;
};
```

```
struct Cold {
  uint32_t age;
  uint32_t ownerID;
  Food treats[10];
};
```

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

```
for (Dog& d : dogs) {
  play(d.friendliness, d.hobby);
}
```

```
struct HotDog {
  double friendliness;
  std::string hobby;
  unique_ptr<Cold> cold;
};
```

```
struct Cold {
  uint32_t age;
  uint32_t ownerID;
  Food treats[10];
};
```

Benefits depend on
the size of Cold & the access patterns

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

```
for (Dog& d : dogs) {
  play(d.friendliness, d.hobby);
}
```

```
struct HotDog {
  double friendliness;
  std::string hobby;
  unique_ptr<Cold> cold;
};
```

```
struct Cold {
  uint32_t age;
  uint32_t ownerID;
  Food treats[10];
};
```

Benefits depend on
the size of Cold & the access patterns

Again, profilers can guide the process.
[Le, 2019]

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

```
struct DogManager {
  std::vector<uint32_t> friendliness;
  std::vector<uint32_t> age;
  std::vector<uint32_t> ownerID;
  std::vector<std::string> hobby;
  std::vector<std::array<Food,10>> treats;
};
```

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)

```
struct Dog {
  uint32_t friendliness;
  uint32_t age;
  uint32_t ownerID;
  std::string hobby;
  Food treats[10];
};
```

```
struct DogManager {
  std::vector<uint32_t> friendliness;
  std::vector<uint32_t> age;
  std::vector<uint32_t> ownerID;
  std::vector<std::string> hobby;
  std::vector<std::array<Food,10>> treats;
};
```

```
for (auto i : range(dogs)) {
  play(friendliness[i], hobby[i]);
}
```

Optimizing Your Data

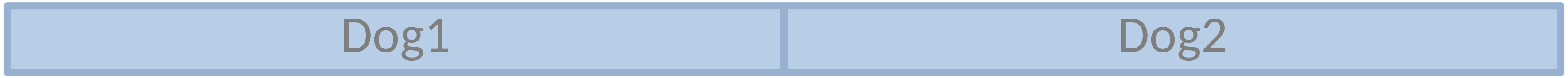
- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)

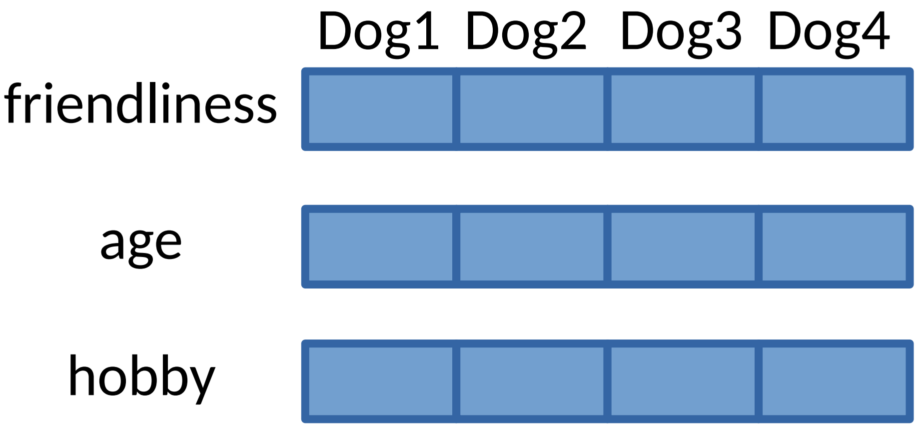
Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)

Optimizing Your Data

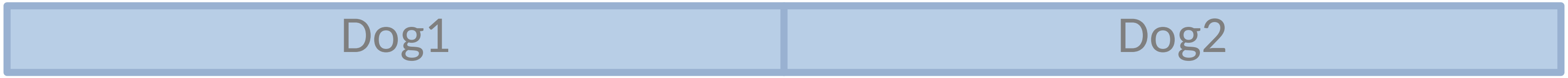
- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)

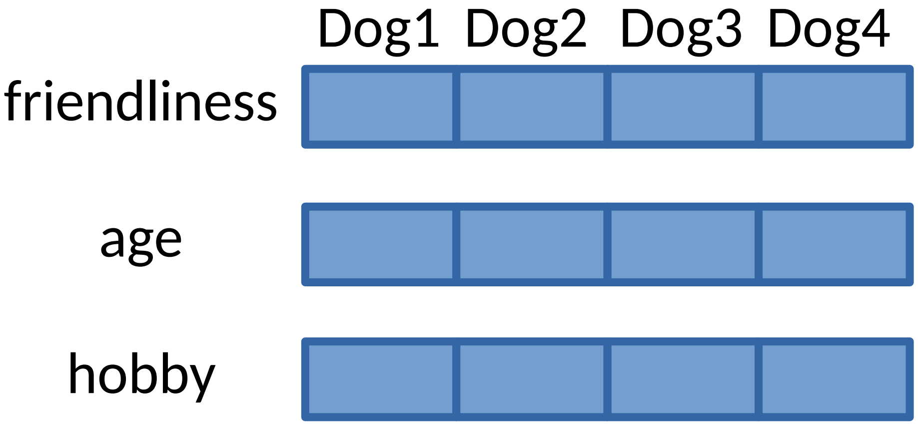




Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)

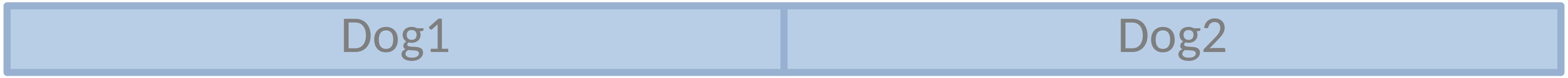


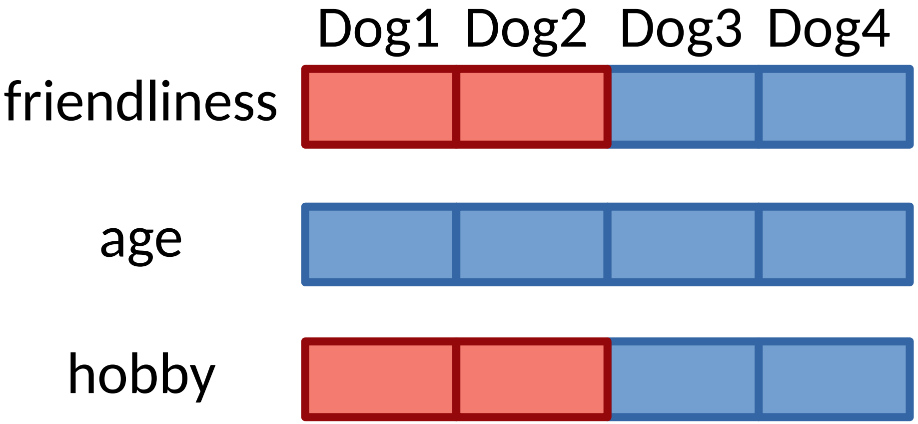


You can pick and choose while still getting good locality

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)

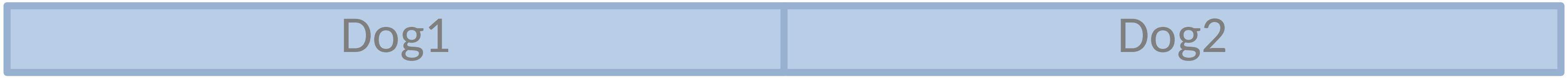


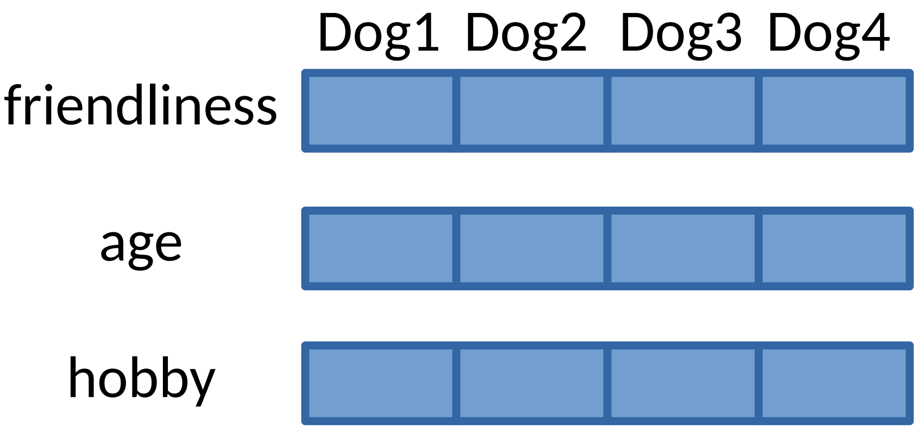


You can pick and choose while still getting good locality

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)



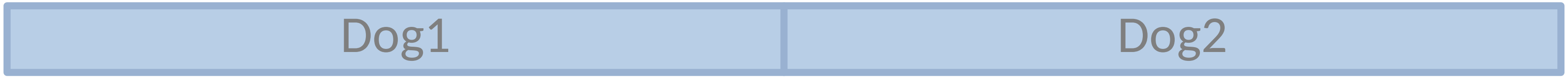


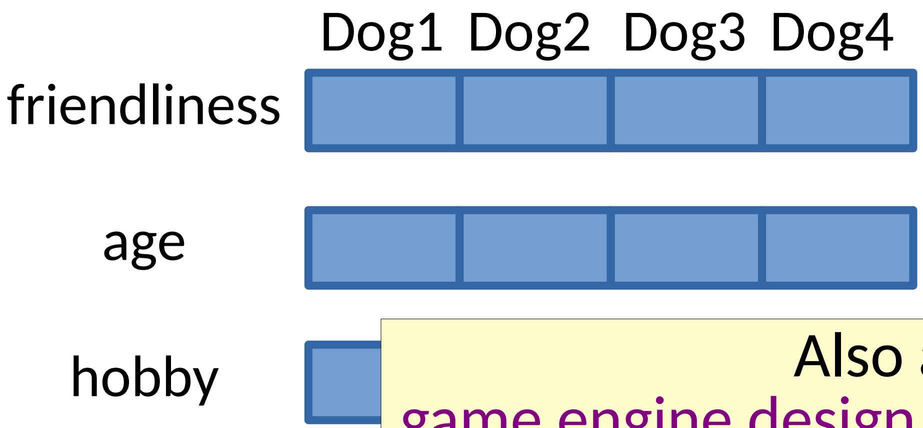
You can pick and choose while still getting good locality

Easier for compilers to vectorize

Optimizing Your Data

- Grouping things that are accessed together
  - Guiding spatial design by temporal locality can improve cache utilization
  - Cold field outlining
  - AoS vs SoA (Array of Structs vs Struct of Arrays)





You can pick and choose while still getting good locality

Easier for compilers to vectorize

Also a foundation of modern game engine design (ECS) & data processing (columnar DB)

Optimizing Your Data

- Loop invariance
  - Avoid recomputing the same values inside a loop

Optimizing Your Data

- Loop invariance
  - Avoid recomputing the same values inside a loop

```
for (auto i : ...) {
  auto sqrt2 = sqrt(2);
  auto x = f(i, sqrt2);
  ...
}
```

```
auto sqrt2 = sqrt(2);
for (auto i : ...) {
  auto x = f(i, sqrt2);
  ...
}
```

Optimizing Your Data

- Loop invariance
  - Avoid recomputing the same values inside a loop
  - Compilers automate this but cannot always succeed (LICM)

```
for (auto i : ...) {
  auto sqrt2 = sqrt(2);
  auto x = f(i, sqrt2);
  ...
}
```

```
auto sqrt2 = sqrt(2);
for (auto i : ...) {
  auto x = f(i, sqrt2);
  ...
}
```

Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

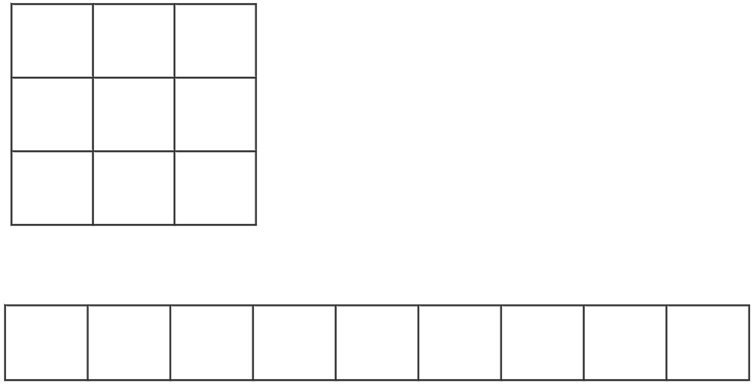
```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```

Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```

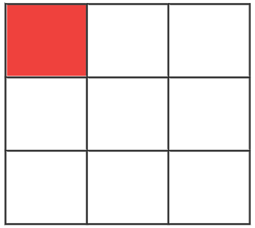


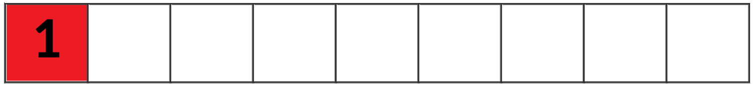
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```



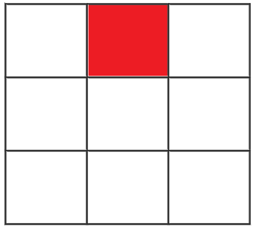


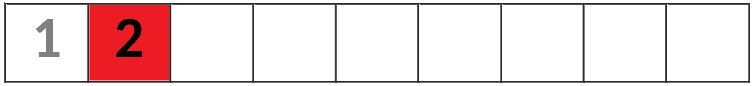
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```



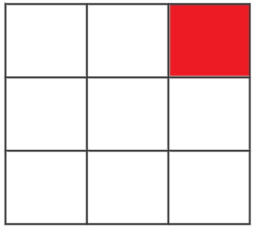


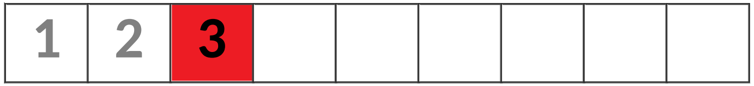
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```



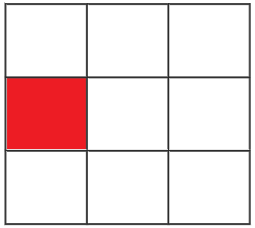


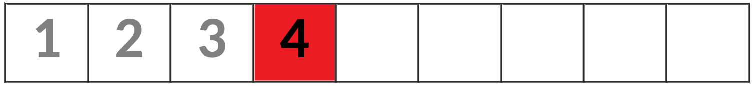
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```

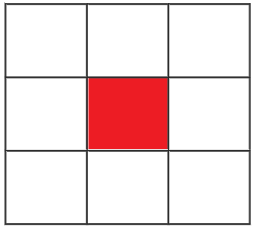




Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```



Memory accesses are consecutive!

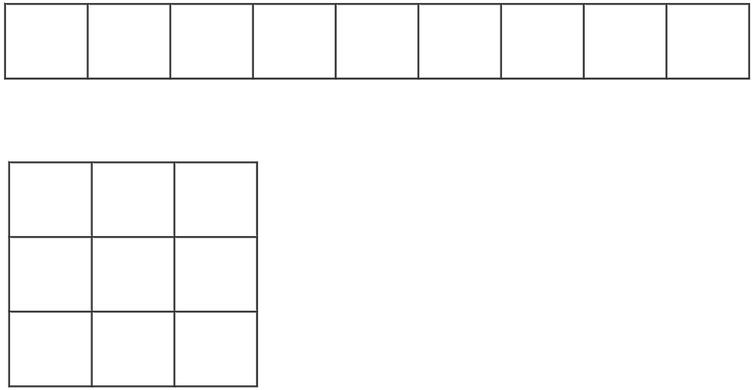

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```

Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```

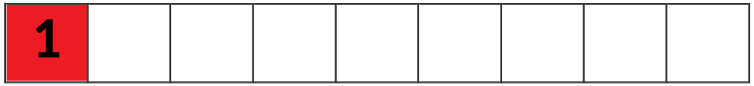


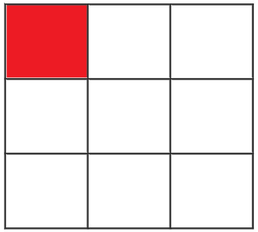
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```



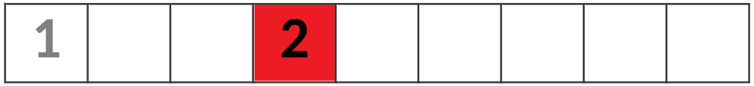


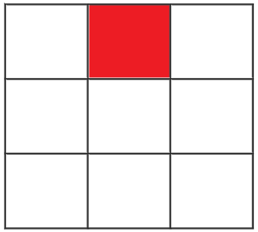
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```



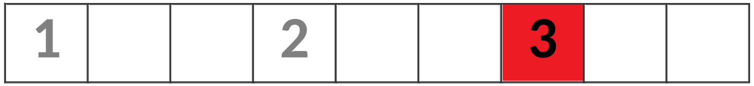


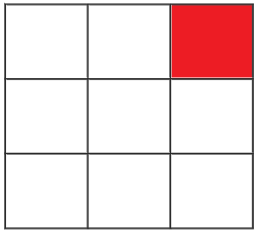
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```



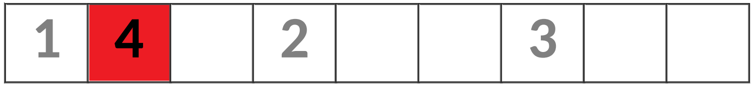


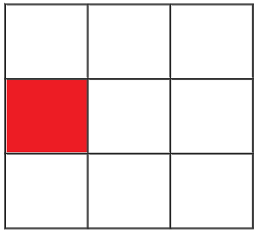
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```



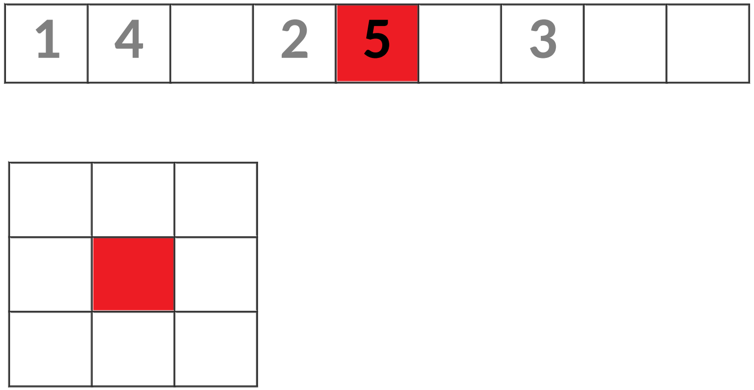


Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```

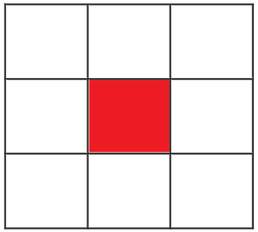


Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw

```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[cols*row + col]);
  }
}
```

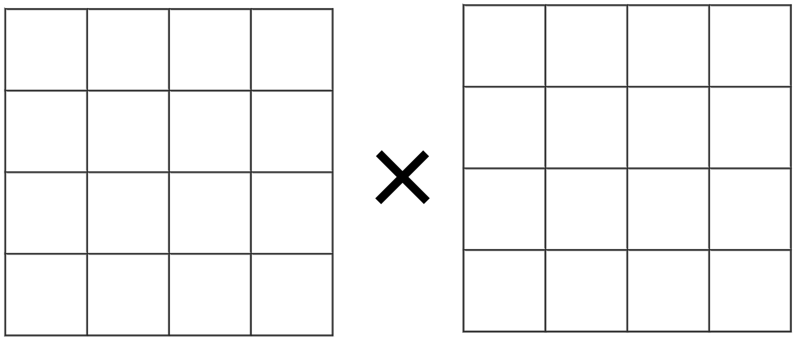
```
uint32_t marix[rows*cols];
for (size_t row = 0; row < rows; ++row) {
  for (size_t col = 0; col < cols; ++col) {
    foo(matrix[rows*col + row]);
  }
}
```



Memory accesses jump around & thrash the cache!

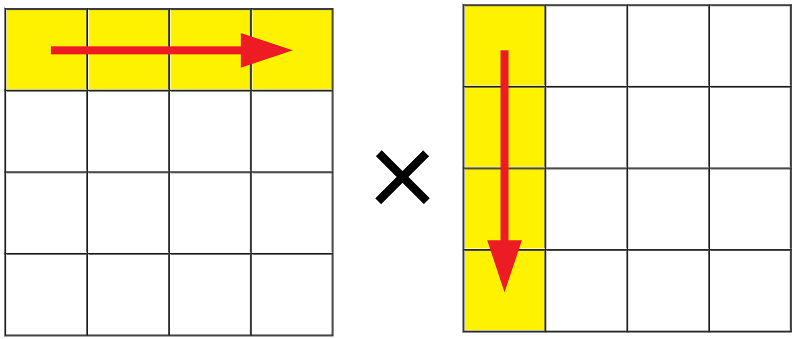
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work



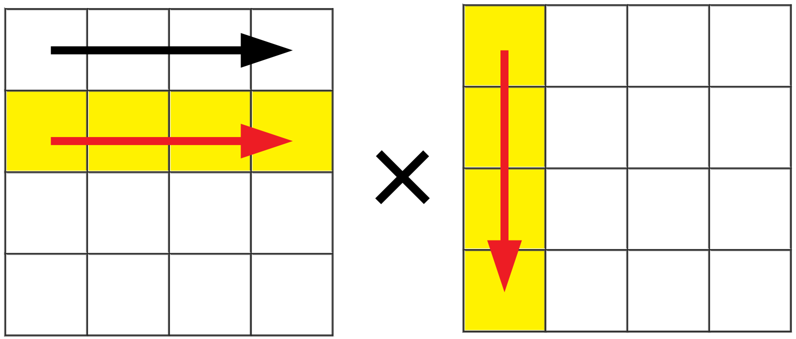
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work



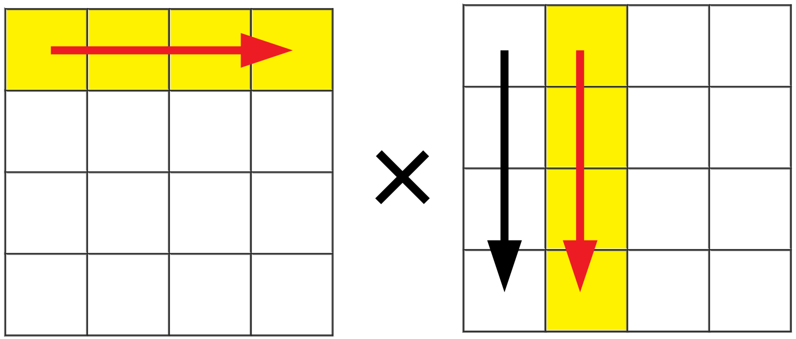
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work



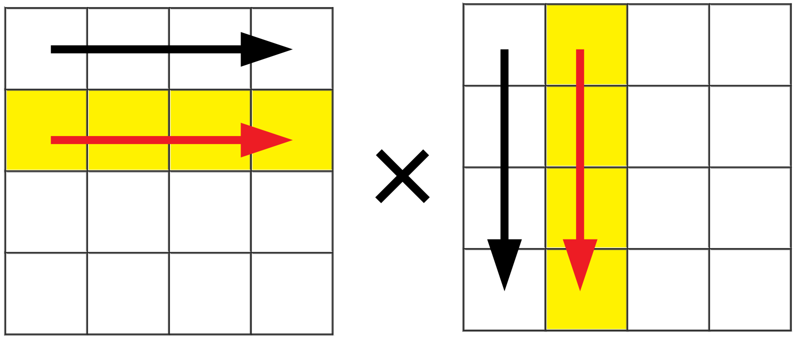
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work



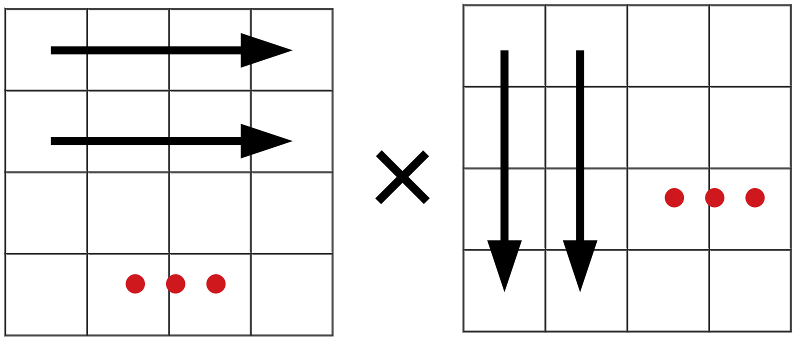
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work



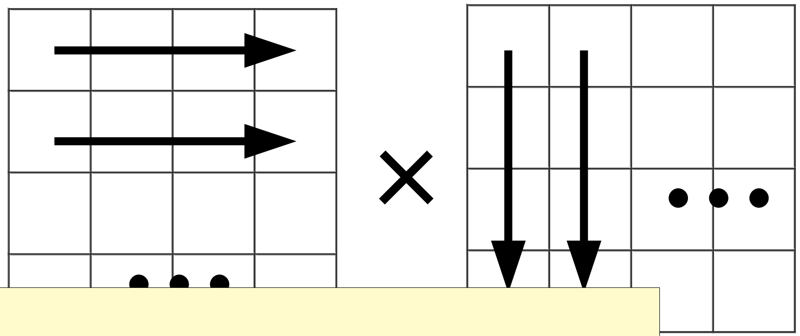
Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work



Optimizing Your Data

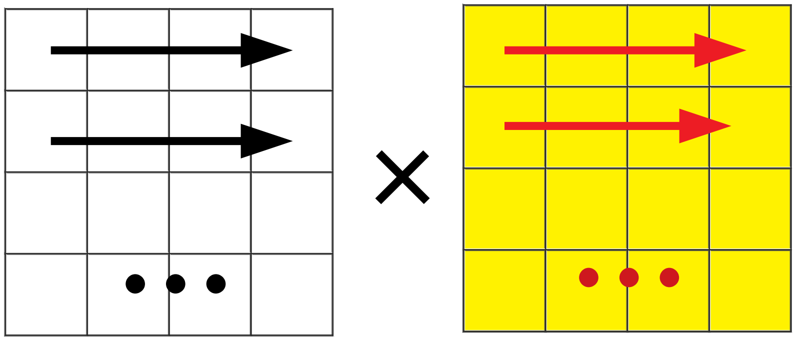
- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work



Problem:
Using the same layout creates bad locality.

Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work

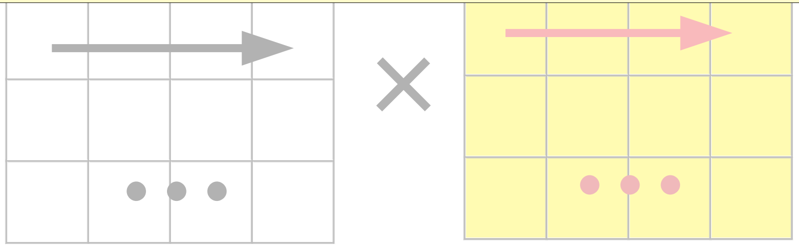


Solution: Transpose first.
Implement over the transpose instead.

Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work

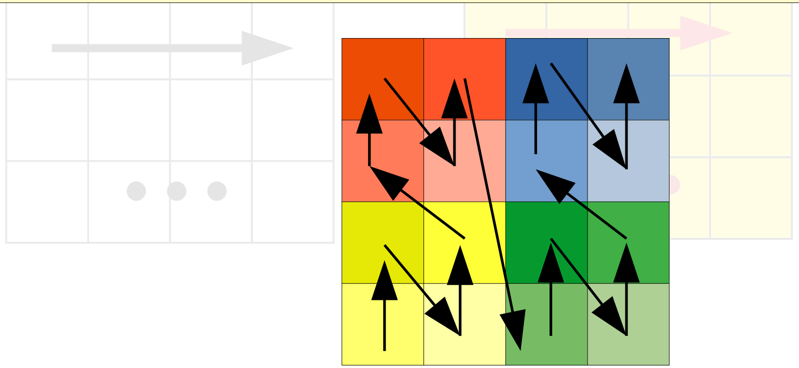
Note: Better solutions further leverage layout & parallelization.



Optimizing Your Data

- Inner loop locality
  - The simplest scenarios are like the matrix example we first saw
  - Matrix operations (e.g. multiplication) can require extra work

Note: Better solutions further leverage layout & parallelization.



tiling, polyhedral analysis, …

[polyhedral.info]

Optimizing Your Data

- Memory management effects
  - Data structure packing & access patterns affect deeper system behavior

Optimizing Your Data

- Memory management effects
  - Data structure packing & access patterns affect deeper system behavior
    - What about virtual memory, page tables, & the TLB?

Optimizing Your Data

- Memory management effects
  - Data structure packing & access patterns affect deeper system behavior
    - What about virtual memory, page tables, & the TLB?
    - What about allocation strategies & fragmentation?

Optimizing Your Data

- Memory management effects
  - Data structure packing & access patterns affect deeper system behavior
    - What about virtual memory, page tables, & the TLB?
    - What about allocation strategies & fragmentation?

Object/Memory pools?

Optimizing Your Data

- Memory management effects
  - Data structure packing & access patterns affect deeper system behavior
    - What about virtual memory, page tables, & the TLB?
    - What about allocation strategies & fragmentation?

      Object/Memory pools?
      Per class allocation?

Optimizing Your Data

- Memory management effects
  - Data structure packing & access patterns affect deeper system behavior
    - What about virtual memory, page tables, & the TLB?
    - What about allocation strategies & fragmentation?

      Object/Memory pools?
      Per class allocation?
      Region based allocation?

Optimizing Your Data

- Memory management effects
  - Data structure packing & access patterns affect deeper system behavior
    - What about virtual memory, page tables, & the TLB?
    - What about allocation strategies & fragmentation?

      Object/Memory pools?
      Per class allocation?
      Region based allocation?
      Bump pointer allocators?
      Cyclic buffers?
      Precomputed allocation requirements & scheduling?

Optimizing Your Data

- Memory management effects
  - Data structure packing & access patterns affect deeper system behavior
    - What about virtual memory, page tables, & the TLB?
    - What about allocation strategies & fragmentation?
  - Data structure inlining
    - folly::small_vector
      absl::InlinedVector
      rust-smallvec
      …

Optimizing Your Data

- Memory management effects
  - Data structure packing & access patterns affect deeper system behavior
    - What about virtual memory, page tables, & the TLB?
    - What about allocation strategies & fragmentation?
  - Data structure inlining
    - folly::small_vector
      absl::InlinedVector
      rust-smallvec
      …

```
small_vector<int,2> vec;
vec.push_back(0); // Stored in-place on stack
vec.push_back(1); // Still on the stack
vec.push_back(2); // Switches to heap buffer
```

[facebook's small_vector]

Optimizing Your Data

- Designing with clear ownership policies in mind

Optimizing Your Data

- Designing with clear ownership policies in mind
  - Resource acquisition should not happen in hot code

Optimizing Your Data

- Designing with clear ownership policies in mind
  - Resource acquisition should not happen in hot code
  - Use APIs that express intent & prevent copying

Optimizing Your Data

- Designing with clear ownership policies in mind
  - Resource acquisition should not happen in hot code
  - Use APIs that express intent & prevent copying

“std::string is responsible for almost half of all allocations in the Chrome”

```
foo(const std::string& s) {
  bar(s.c_str());
}
```

```
bar(const char* s) {
  baz(std::string{s});
}
```

```
baz(const std::string& s) {
  quux(s.c_str());
}
```

```
quux(const char* s) {
  quuz(std::string{s});
}
```

Optimizing Your Data

- Designing with clear ownership policies in mind
  - Resource acquisition should not happen in hot code
  - Use APIs that express intent & prevent copying

"std::string is responsible for almost half of all allocations in the Chrome"

```
template<class E>
struct Span {
    template<class E, auto N>
    Span(const std::array<E,N>& c);

    template<class E>
    Span(const std::vector<E>& c);

    E* first;
    size_t count;
};
```

Optimizing Your Code

- Basic ideas for code optimization (we'll walk through examples shortly)

Optimizing Your Code

- Basic ideas for code optimization
  - Avoid branching whenever possible

Optimizing Your Code

- Basic ideas for code optimization
  - Avoid branching whenever possible

Mis-speculating over a branch is costly

Optimizing Your Code

- Basic ideas for code optimization
  - Avoid branching whenever possible
  - Make code that does the same thing occur close together temporally

Optimizing Your Code

- Basic ideas for code optimization
  - Avoid branching whenever possible
  - Make code that does the same thing occur close together temporally

Leverage the instruction cache if you can

Optimizing Your Code

- Branch prediction & speculation

Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements

```
for (...) {
  if (foo(c)) {
    bar();  A
  } else {
    baz();  B
  }
}
```

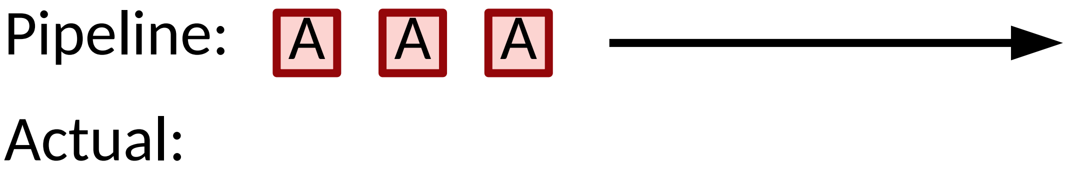
90%

10%

Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements

```
for (...) {
  if (foo(c)) {
    bar();    A    90%
  } else {
    baz();    B    10%
  }
}
```



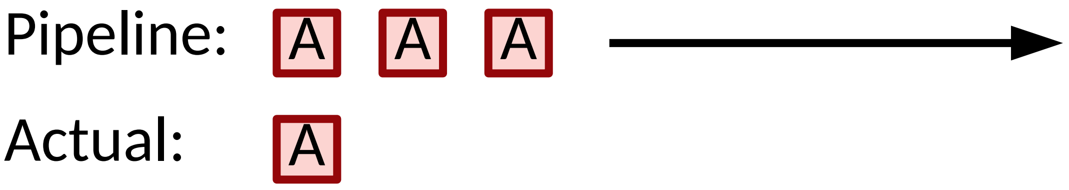
Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements

```
for (...) {
  if (foo(c)) {
    bar();   A
  } else {
    baz();   B
  }
}
```

90%

10%



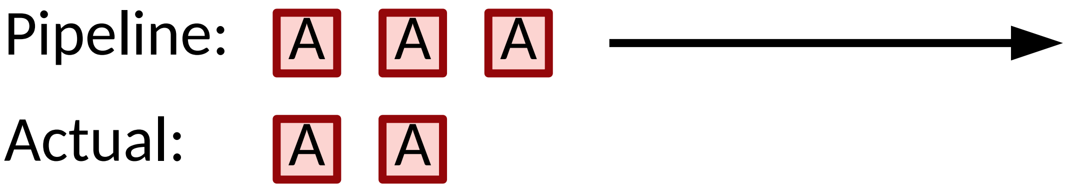
Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements

```
for (...) {
  if (foo(c)) {
    bar();   A
  } else {
    baz();   B
  }
}
```

90%

10%



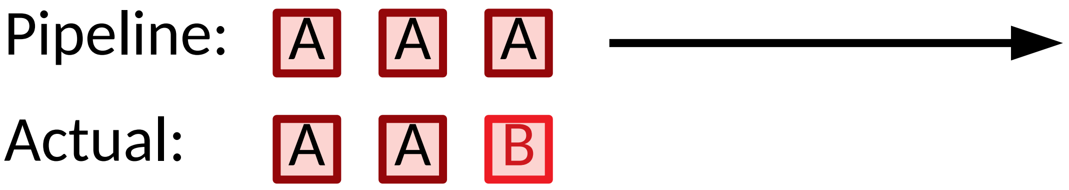
Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements

```
for (...) {
  if (foo(c)) {
    bar();   A
  } else {
    baz();   B
  }
}
```

A 90%

B 10%



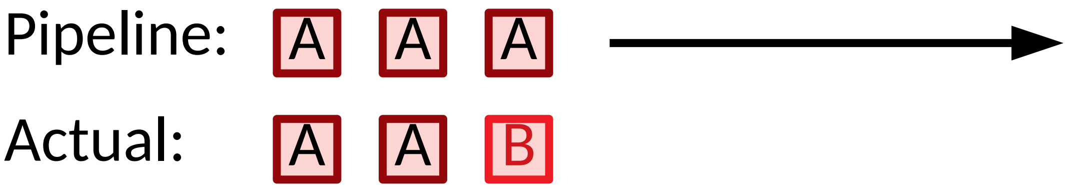
Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements

```
for (...) {
  if (foo(c)) {
    bar();  [A]
  } else {
    baz();  [B]
  }
}
```

90%

10%



Stall, but relatively infrequently

Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements

```
for (...) {
  if (foo(c)) {
    bar(); A
  } else {
    baz(); B
  }
}
```

51%

49%

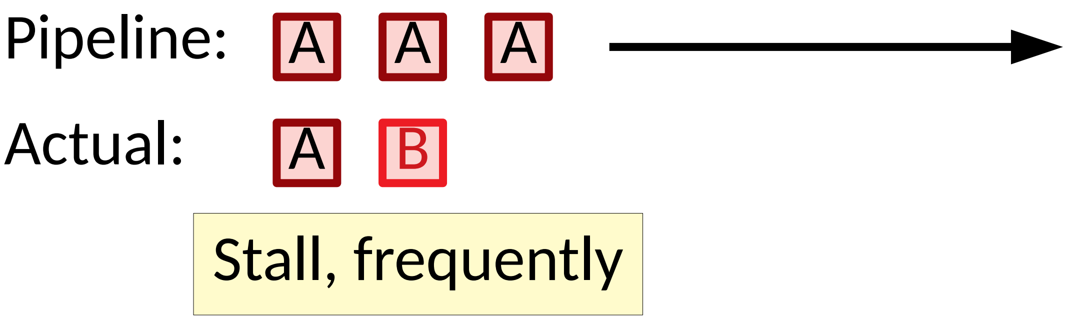
Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements

```
for (...) {
  if (foo(c)) {
    bar();   A
  } else {
    baz();   B
  }
}
```

51%

49%



Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements

```
for (...) {
  if (foo(c)) {
    bar();    A
  } else {
    baz();    B
  }
}
```

51%

49%

Pipeline: A A A →

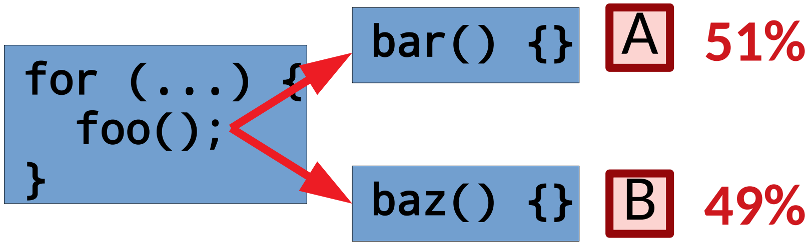
Actual: A B

Stall, frequently

How would you fix it?

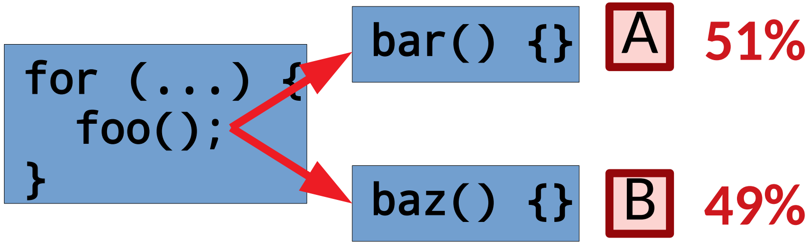
Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements
  - On function pointers!



Optimizing Your Code

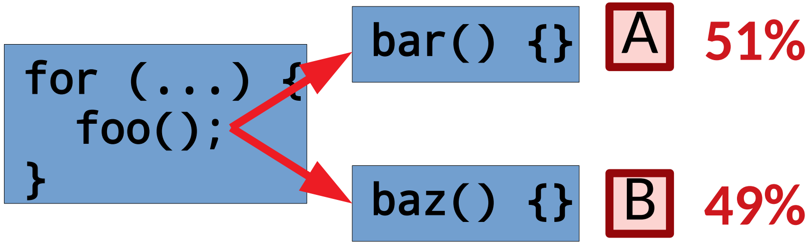
- Branch prediction & speculation
  - On `if` statements
  - On function pointers!



The same problems arise

Optimizing Your Code

- Branch prediction & speculation
  - On `if` statements
  - On function pointers!



The same problems arise

Consistent call targets perform better

Optimizing Your Code

- Designing away checks
  - Repeated checks can be removed by maintaining invariants

Optimizing Your Code

- Designing away checks
  - Repeated checks can be removed by maintaining invariants

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

[Wikipedia’s Insertion Sort]

Optimizing Your Code

- Designing away checks
  - Repeated checks can be removed by maintaining invariants

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

[Wikipedia’s Insertion Sort]

Optimizing Your Code

- Designing away checks
  - Repeated checks can be removed by maintaining invariants

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

[Wikipedia’s Insertion Sort]

Can we turn the semantic check into a bounds check?

Optimizing Your Code

- Designing away checks
  - Repeated checks can be removed by maintaining invariants

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

[Wikipedia's Insertion Sort]

We just guarantee that **A** starts with the smallest element!

Optimizing Your Code

- Designing away checks
  - Repeated checks can be removed by maintaining invariants

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

[Wikipedia's Insertion Sort]

```
k ← find_smallest(A)
swap A[0] and A[k]
i ← 1
while i < length(A)
    j ← i
    while A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

We just guarantee that **A** starts with the smallest element!

Optimizing Your Code

- Designing away checks
  - Repeated checks can be removed by maintaining invariants

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

[Wikipedia's Insertion Sort]

```
k ← find_smallest(A)
swap A[0] and A[k]
i ← 1
while i < length(A)
    j ← i
    while A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

On an Intel i7@ 2.20GHz, uniformly random data

8192 elements:	269k items/s	415k items/s
32768 elements:	68k items/s	104k items/s
131072 elements:	17k items/s	26k items/s

Optimizing Your Code

- Designing away checks
  - Repeated checks can be removed by maintaining invariants

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

[Wikipedia’s Insertion Sort]

```
A[-1] ← MIN_VALUE
i ← 1
while i < length(A)
    j ← i
    while A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    i ← i + 1
```

Extra domain knowledge may allow this in different ways.

Values that do not appear?
Shape & distribution?
…

Optimizing Algorithms

- Improving real world algorithmic performance comes from recognizing the ***interplay*** between ***theory*** and ***hardware***

Optimizing Algorithms

- Improving real world algorithmic performance comes from recognizing the ***interplay*** between ***theory*** and ***hardware***
- Hybrid algorithms – use multiple algorithms & choose

Optimizing Algorithms

- Improving real world algorithmic performance comes from recognizing the ***interplay*** between ***theory*** and ***hardware***
- Hybrid algorithms – use multiple algorithms & choose
  - Constants matter. Use thresholds to select algorithms.

Optimizing Algorithms

- Improving real world algorithmic performance comes from recognizing the ***interplay*** between ***theory*** and ***hardware***
- Hybrid algorithms – use multiple algorithms & choose
  - Constants matter. Use thresholds to select algorithms.
  - Use general N logN sorting for N above 300 [Alexandrescu 2019]

Optimizing Algorithms

- Improving real world algorithmic performance comes from recognizing the ***interplay*** between ***theory*** and ***hardware***
- Hybrid algorithms
  - Constants matter. Use thresholds to select algorithms.
  - Use general N logN sorting for N above 300 [Alexandrescu 2019]
- Caching & Precomputing

Optimizing Algorithms

- Improving real world algorithmic performance comes from recognizing the ***interplay*** between ***theory*** and ***hardware***
- Hybrid algorithms
  - Constants matter. Use thresholds to select algorithms.
  - Use general N logN sorting for N above 300 [Alexandrescu 2019]
- Caching & Precomputing
  - If you will reuse results, save them and avoid recomputing

Optimizing Algorithms

- Improving real world algorithmic performance comes from recognizing the ***interplay*** between ***theory*** and ***hardware***
- Hybrid algorithms
  - Constants matter. Use thresholds to select algorithms.
  - Use general N logN sorting for N above 300 [Alexandrescu 2019]
- Caching & Precomputing
  - If you will reuse results, save them and avoid recomputing
  - If all possible results are compact, just compute a table up front

Optimizing Algorithms

- Improving real world algorithmic performance comes from recognizing the ***interplay*** between ***theory*** and ***hardware***
- Hybrid algorithms
  - Constants matter. Use thresholds to select algorithms.
  - Use general N logN sorting for N above 300 [Alexandrescu 2019]
- Caching & Precomputing
  - If you will reuse results, save them and avoid recomputing
  - If all possible results are compact, just compute a table up front
- Predictability & Speculation
  - Can you determine data & behaviors early?
  - Can you fetch/perform them during an early lull?

Optimizing Algorithms

- Improving real world algorithmic performance comes from recognizing the ***interplay*** between ***theory*** and ***hardware***
- Hybrid algorithms
  - Constants matter. Use thresholds to select algorithms.
  - Use general N logN sorting for N above 300 [Alexandrescu 2019]
- Caching & Precomputing
  - If you will reu… ting
  - If all possible … up front

e.g., determine resources for a web page,
& fetch them when initially loading
[Mickens 2010, Netravali 2018, Ko 2021]

- Predictability & Speculation
  - Can you determine data & behaviors early?
  - Can you fetch/perform them during an early lull?

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice
  - It uses an ***abstract machine model*** that is too approximate!

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice
  - It uses an abstract machine model that is too approximate!
  - Constants and artifacts of scale can actually dominate the real world performance

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice
  - It uses an abstract machine model that is too approximate!
  - Constants and artifacts of scale can actually dominate the real world performance

A ***uniform cost model***
throws necessary information away

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice
  - It uses an abstract machine model that is too approximate!
  - Constants and artifacts of scale can actually dominate the real world performance
  - We want modeling & algorithms that account for artifacts like: memory, I/O, consistency & speculation, shapes of workloads

Optimizing Algorithms

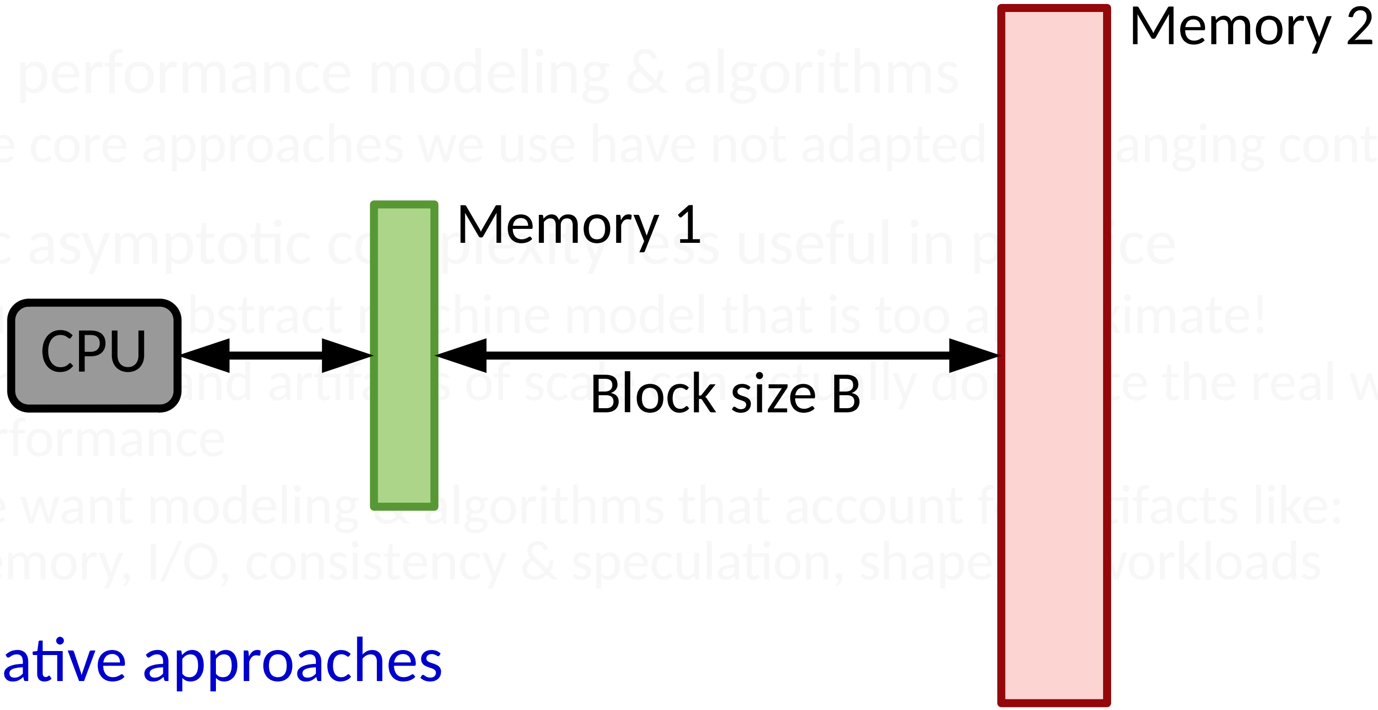
- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice
  - It uses an abstract machine model that is too approximate!
  - Constants and artifacts of scale can actually dominate the real world performance
  - We want modeling & algorithms that account for artifacts like: memory, I/O, consistency & speculation, shapes of workloads
- Alternative approaches

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice
  - It uses an abstract machine model that is too approximate!
  - Constants and artifacts of scale can actually dominate the real world performance
  - We want modeling & algorithms that account for artifacts like: memory, I/O, consistency & speculation, shapes of workloads
- Alternative approaches
  - I/O complexity, I/O efficiency and cache awareness

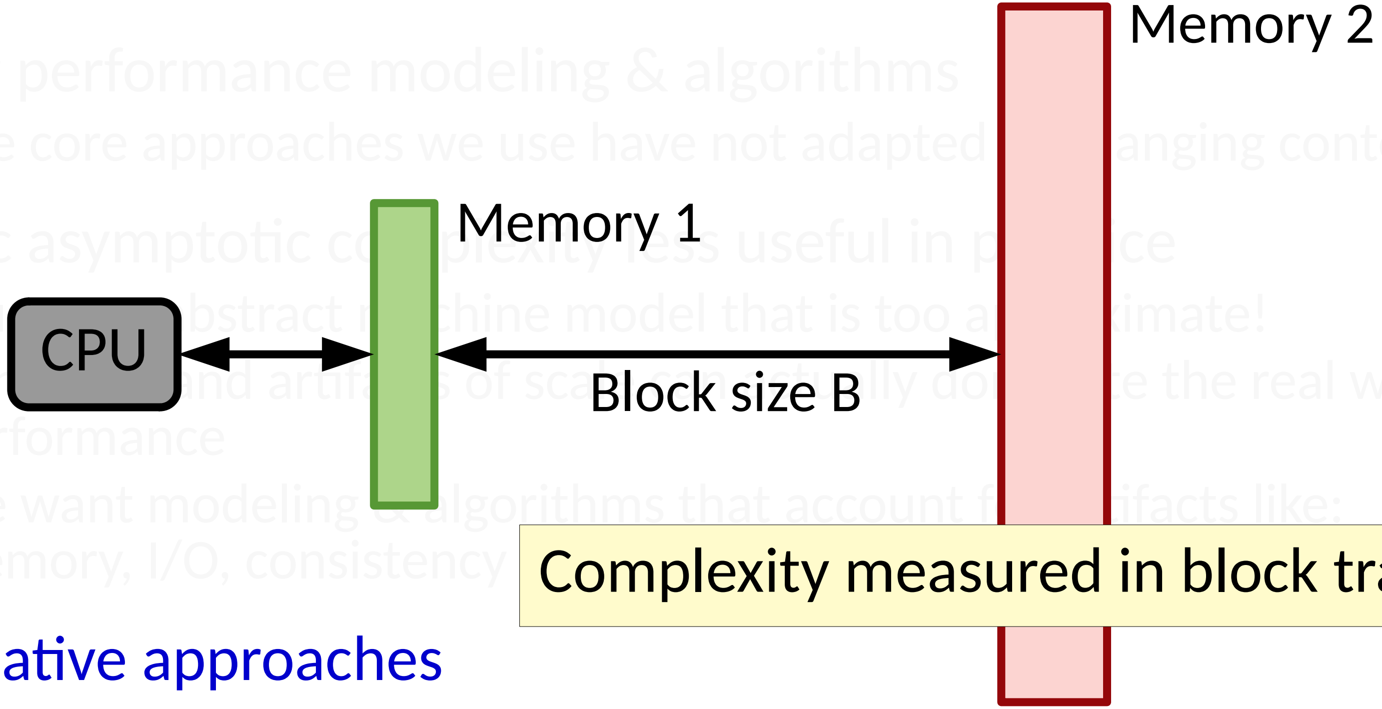
Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted [illegible]anging contexts
- Classic asymptotic co[illegible]plexity less useful in p[illegible]ce
  - It u[illegible]bstract m[illegible]chine model that is too a[illegible]timate!
  - Co[illegible]and artifa[illegible] of sca[illegible]lly do[illegible]e the real world performance
  - We want modeling & algorithms that account f[illegible]ifacts like: memory, I/O, consistency & speculation, shape[illegible]orkloads



- Alternative approaches
  - I/O complexity, I/O efficiency and cache awareness

Optimizing Algorithms



Complexity measured in block transfers

- Alternative approaches
  - I/O complexity, I/O efficiency and cache awareness

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice
  - It uses an abstract machine model that is too approximate!
  - Constants and artifacts of scale can actually dominate the real world performance
  - We want modeling & algorithms that account for artifacts like: memory, I/O, consistency & speculation, shapes of workloads
- Alternative approaches
  - I/O complexity, I/O efficiency and cache awareness
  - Cache oblivious algorithms & data structures

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice
  - It uses an abstract machine model that is too approximate!
  - Constants and artifacts of scale can actually dominate the real world performance
  - We want modeling & algorithms that account for artifacts like: memory, I/O, consistency & speculation, shapes of workloads
- Alternative approaches
  - I/O complexity, I/O efficiency and cache awareness
  - Cache oblivious algorithms & data structures

Similar to I/O, but agnostic to block size

Optimizing Algorithms

- Better performance modeling & algorithms
  - The core approaches we use have not adapted to changing contexts
- Classic asymptotic complexity less useful in practice
  - It uses an abstract machine model that is too approximate!
  - Constants and artifacts of scale can actually dominate the real world performance
  - We want modeling & algorithms that account for artifacts like: memory, I/O, consistency & speculation, shapes of workloads
- Alternative approaches
  - I/O complexity, I/O efficiency and cache awareness
  - Cache oblivious algorithms & data structures
  - Parameterized complexity

Optimizing Algorithms

- Classic design mistakes [Lu 2012]

Optimizing Algorithms

- Classic design mistakes [Lu 2012]
  - Uncoordinated functions (e.g. lack of batching)

```
for (auto& action : actions) {
  action.do()
}
```

```
Action::do() {
  acquire(mutex)
  ...
  release(mutex)
}
```

Optimizing Algorithms

- Classic design mistakes [Lu 2012]
  - Uncoordinated functions (e.g. lack of batching)

```
for (auto& action : actions) {
  action.do()
}
```

```
Action::do() {
  acquire(mutex)
  ...
  release(mutex)
}
```

VS

```
acquire(mutex)
for (auto& action : actions) {
  action.do()
}
release(mutex)
```

Optimizing Algorithms

- Classic design mistakes [Lu 2012]
  - Uncoordinated functions (e.g. lack of batching)
  - Skippable functions (e.g. transparent draws)

Optimizing Algorithms

- Classic design mistakes [Lu 2012]
  - Uncoordinated functions (e.g. lack of batching)
  - Skippable functions (e.g. transparent draws)
  - Poor/unclear synchronization

Optimizing Algorithms

- Classic design mistakes [Lu 2012]
  - Uncoordinated functions (e.g. lack of batching)
  - Skippable functions (e.g. transparent draws)
  - Poor/unclear synchronization

```
foo() {
  bar()
}
```

```
bar() {
  baz()
}
```

```
baz() {
  quux()
}
```

```
quux() {
  random()
}
```

```
random() {
  acquire(mutex)
  ...
  release(mutex)
}
```

Optimizing Algorithms

- Classic design mistakes [Lu 2012]
  - Uncoordinated functions (e.g. lack of batching)
  - Skippable functions (e.g. transparent draws)
  - Poor/unclear synchronization

```
foo() {
  bar()
}
```

```
bar() {
  baz()
}
```

```
baz() {
  quux()
}
```

```
quux() {
  random()
}
```

```
random() {
  acquire(mutex)
  ...
  release(mutex)
}
```

Consider
shallow, broad, & explicit designs, or
designing the resource away.

Optimizing Algorithms

- Classic design mistakes **[Lu 2012]**
  - Uncoordinated functions (e.g. lack of batching)
  - Skippable functions (e.g. transparent draws)
  - Poor/unclear synchronization
  - Poor data structure selection

Optimizing Algorithms

- Classic design mistakes **[Lu 2012]**
  - Uncoordinated functions (e.g. lack of batching)
  - Skippable functions (e.g. transparent draws)
  - Poor/unclear synchronization
  - Poor data structure selection

This sounds simple,
but it can become quite challenging.
[Loncaric 2016, Idreos 2018, Loncaric 2018]

Summary

- Reasoning rigorously about performance is challenging

Summary

- Reasoning rigorously about performance is challenging
- Good tooling can allow you to investigate performance well

Summary

- Reasoning rigorously about performance is challenging
- Good tooling can allow you to investigate performance well
- We can improve performance through
  - compilers
  - managing data
  - managing code
  - better algorithmic thinking