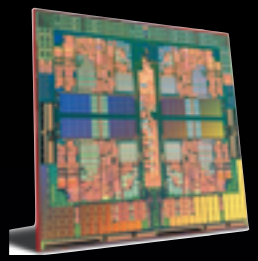


Programming Parallel and Distributed Systems

http://www.cs.sfu.ca/~ashriram/courses/CS880_479/

Monday : 2:30-3:20
Wednesday : 2:30-3:20
Fridays : 2:30-3:20

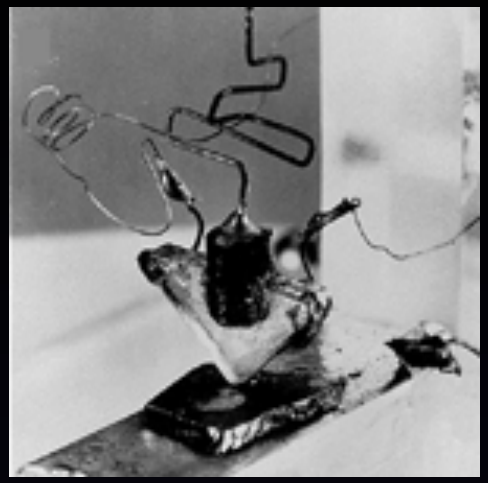
Who am I ?

- Arrvindh Shriraman
  - faculty at SFU since Jan 2011
  - graduated from University of Rochester, NY in 2010
- Areas of research
  - Multicore / Manycore Systems Architecture
  - Parallel Programming Models
  - Energy Management
  - Cache Subsystem

What's the class about?

- Multicore processors
  - current and future computing platform
  - lots of integration (e.g., GPUs)
  - various architectural tradeoffs
- The parallelism wall
  - programming models
  - how to structure communication and data structures
- Large-scale Distributed Systems
  - formal models of execution on warehouse-scale systems
  - Programming the warehouse

Technology

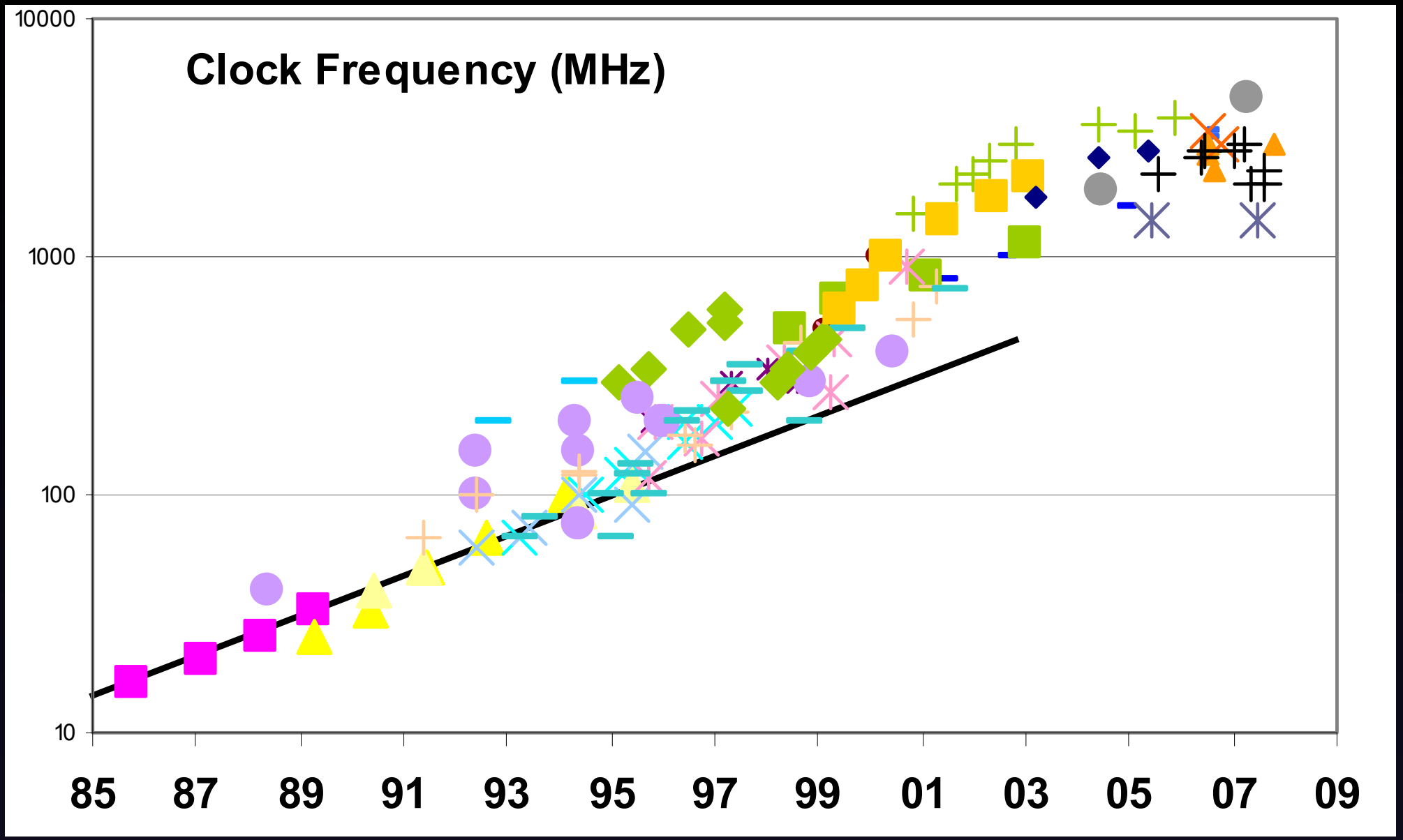


Transistor : 1947

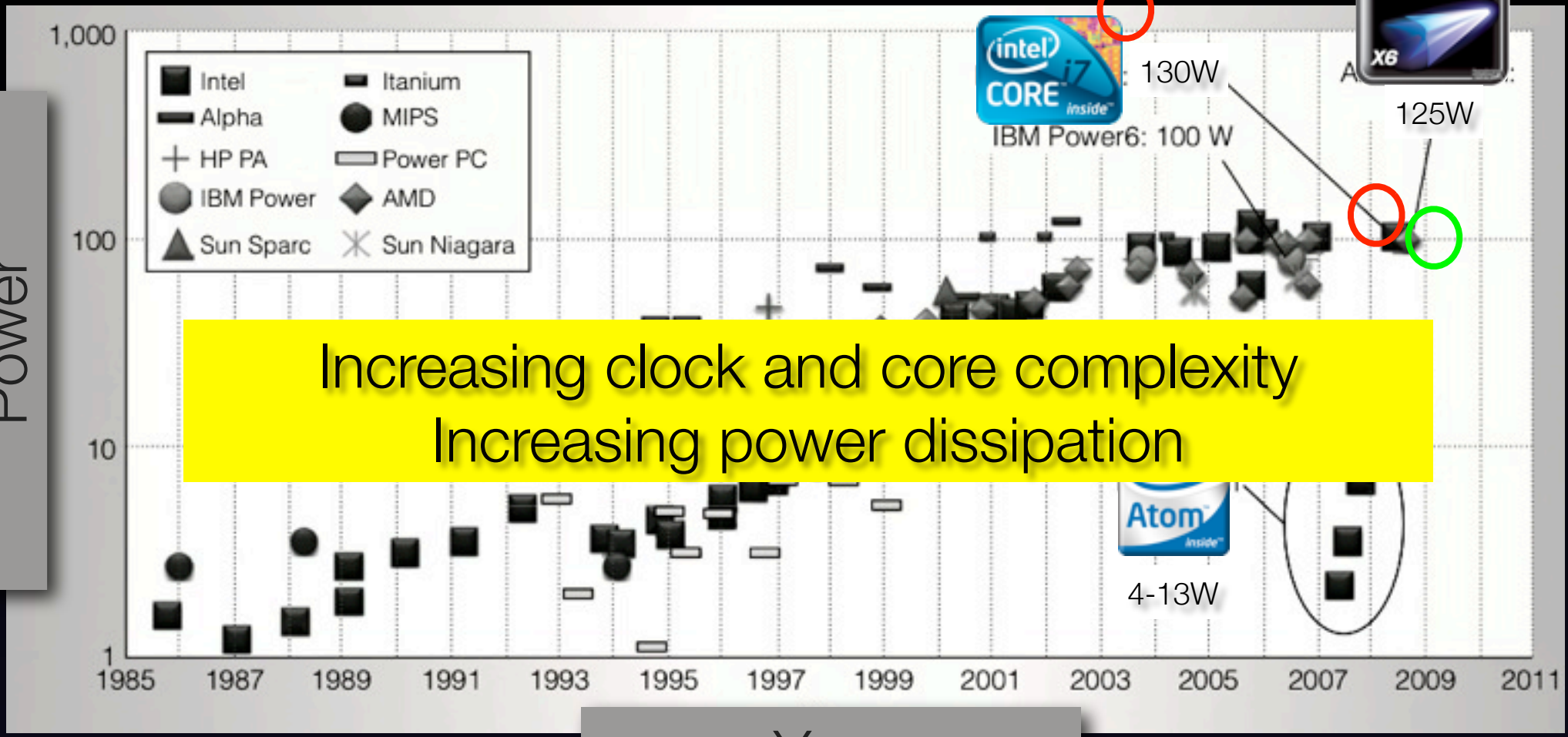
Chip : 1958

Moore’s Law : # of transistors double every 18months

Clock Frequency



CPUs consume a lot of power



Source: Rethinking digital design: Why design must change [IEEE Micro]

Development cycle

Multicore Revolution is here!

More cores on a chip

Each core ; 40% ↓ Ghz = 0.25x Power ↓

Overall Performance = 4 cores * 0.6x/core = 2.4x

Research avenues (hint: final papers)

- Architecture
  - many different types of cores (GPUs, Cryptos, etc..)
  - many caching strategies and memory models
- Programmability
  - Where are the threads?
  - Different programming models

- Distributed Systems
  - How to scale.
  - How scalable are multicores?
  - How to run parallel programs across O (100000) machines ?

Class Organization

- Module 1 (Weeks 1-4) : Parallelism
  - Today (Multicore systems)
  - Memory hierarchy, Locks, Parallel Programming
- Module 1.5 (Weeks 5-7): Multicore Systems
  - cache coherence, memory model
  - GPUs, Vector processors.
- Module 2 (Weeks 5-12): Distributed Systems
  - Google Datacenters
  - Distributed Transactions, Consensus Protocols etc
  - Recovery Protocols, Election Algorithms

Class structure

- Class participation (20%.) Extra Credit : 10% if you present
  - each student will lead the discussion in the class
  - peer-group evaluation
- 3 Prog. Assignments (20% each. Can form groups of 2)
  - Assignment 1: Threads and Synchronization
  - Assignment 2: Multicore Application (OpenMP, Cilk)
- Final Mini project (groups of 2)
  - **Parallelize your favorite Application.**

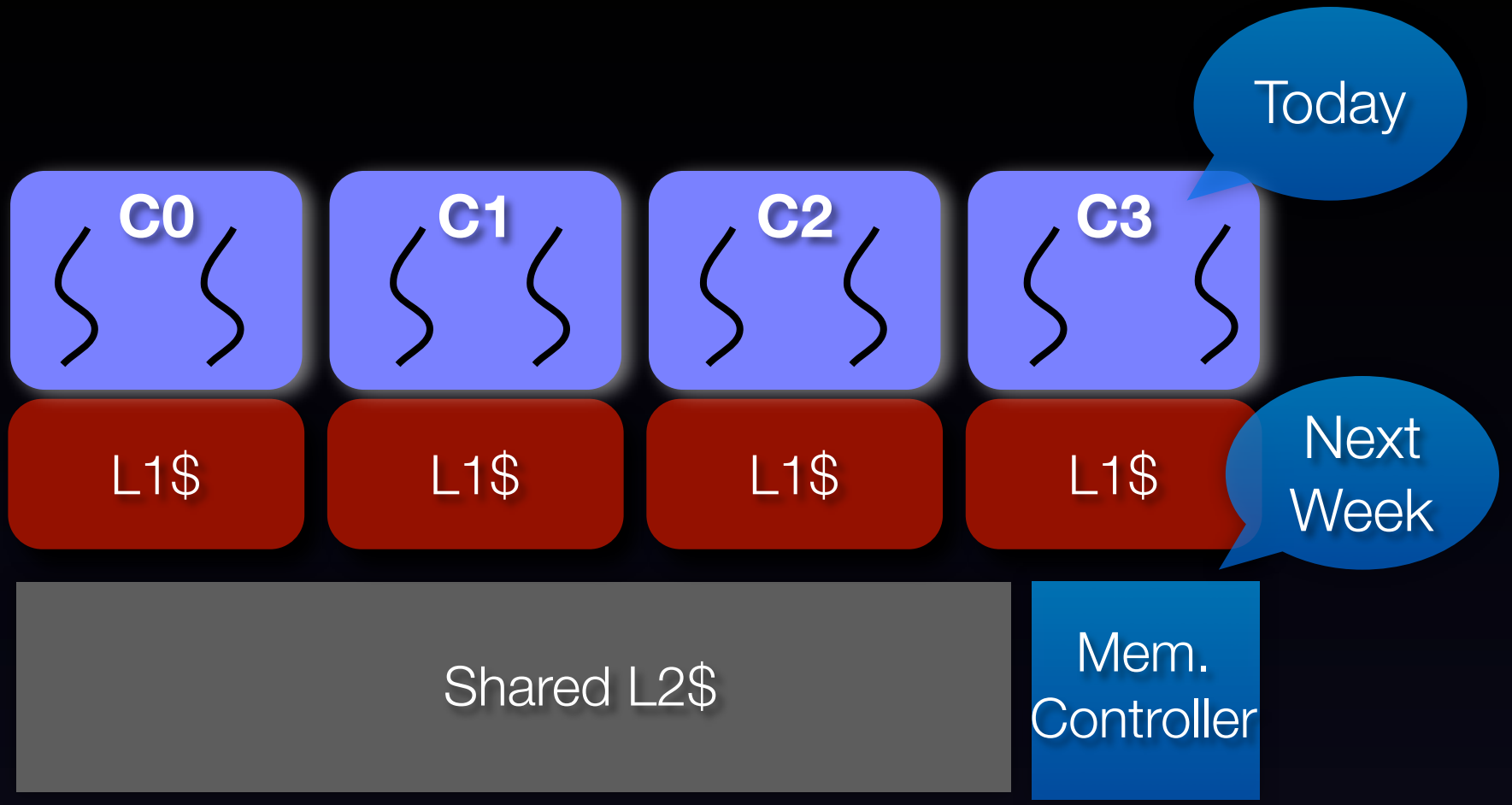
What's expected of you?

- Expect to dedicate about 25 hrs/week for class work
- Need to be self-motivated and work hard
  - self driven
  - I can help you help yourself (I can't do it for you)
- No Books! lots of programming.

Relation to other classes

- Summer CMPT 885
  - More details on cache coherence
  - Parallel Data Structures
  - Formal Memory Models
  - GPUs and Vectors
- CMPT 886
  - Similarities in Distributed Systems
  -

What' inside a multicore chip?

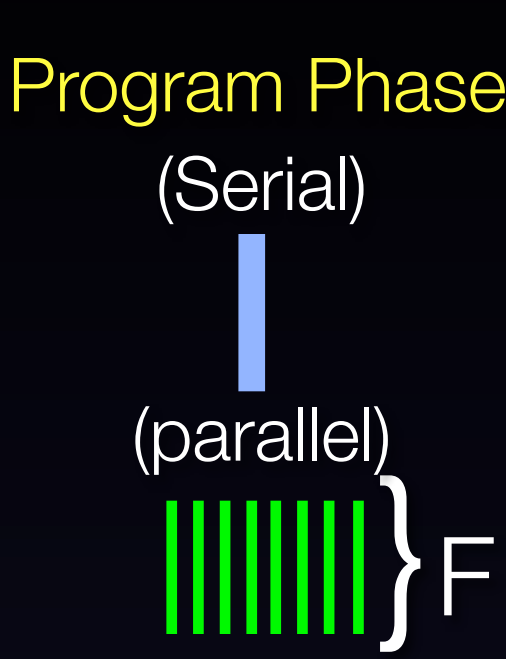


Outline

- How did we get here. Why Multicores?
- Amdahl's Law: How do Apps scale on Multicores?
  - Symmetric, Asymmetric, Morphing
- Dissecting a Processor Core
  - Different types, Performance etc.

Amdahl's Law [1967, Gene Amdahl]

- Maximum speedup achievable on a multicore



Time on 1 core = $\frac{1-F}{1} + \frac{F}{1}$

Time on N cores = $\frac{1-F}{1} + \frac{F}{N}$

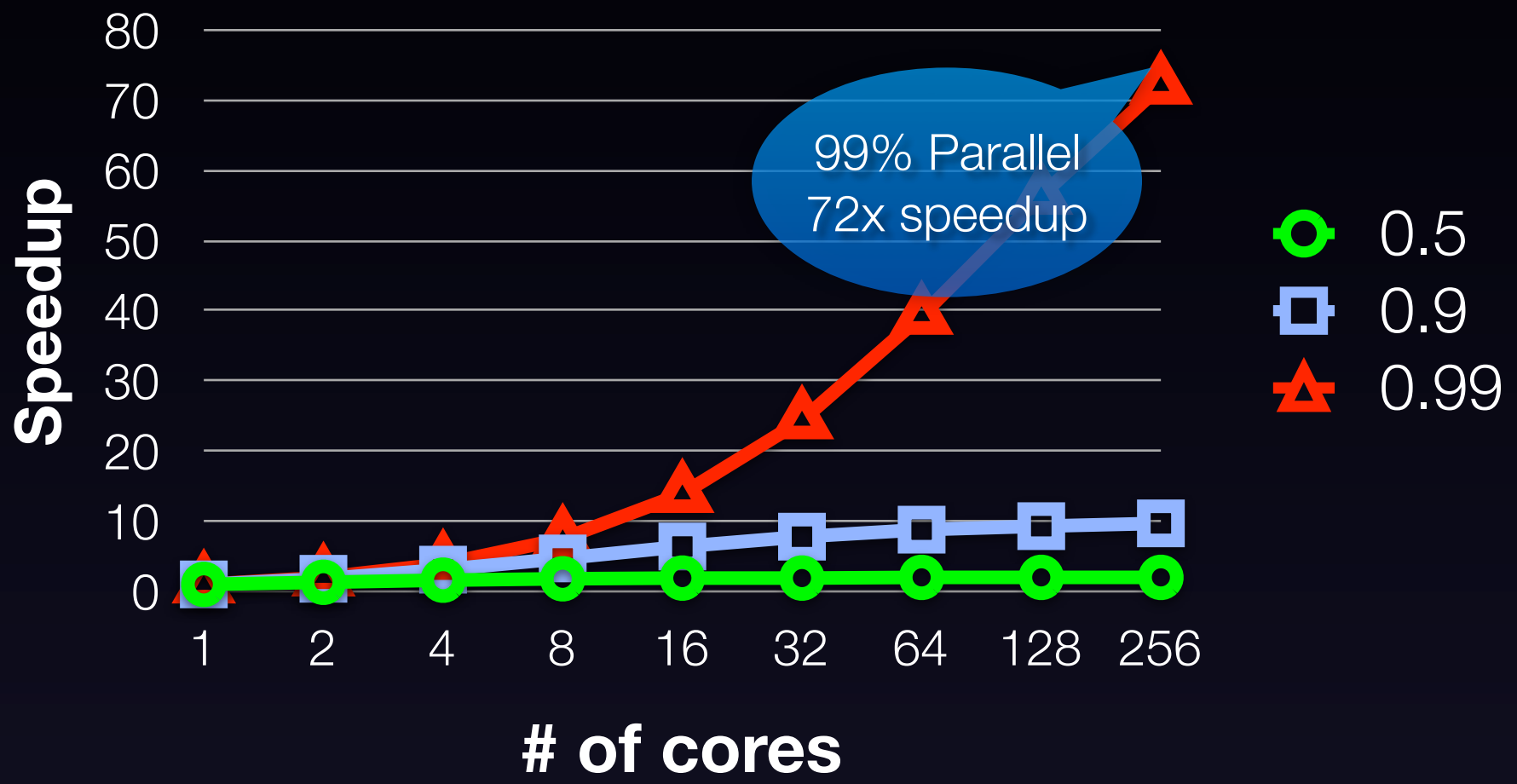
Serial
No Improvement

Speedup = $\frac{1}{\frac{1-F}{1} + \frac{F}{N}}$

If F = 0.35
@ 4 cores, speedup = 2
@ ∞ cores, speedup = 3

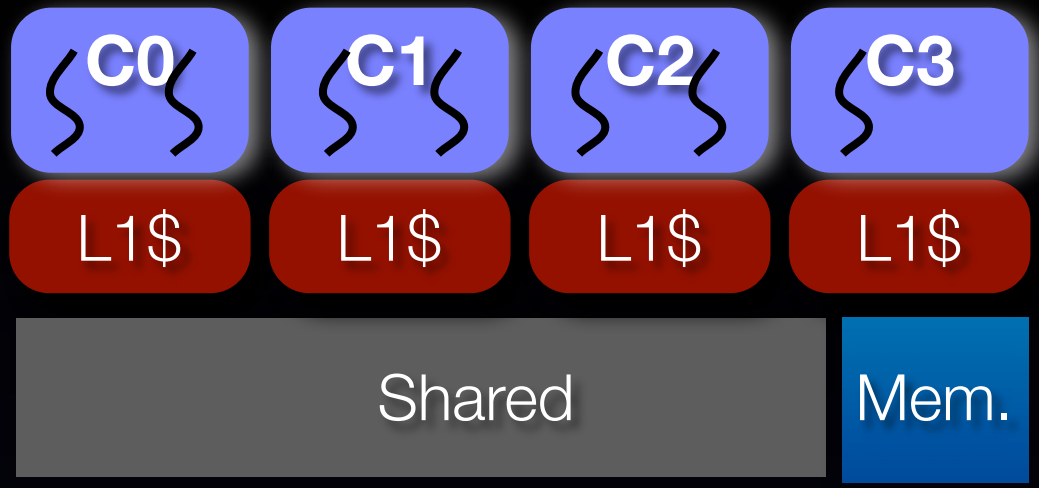
Strong scaling vs Weak scaling

- Strong Scaling : If new machine has K times more resources, how much does perf. improve ?
- Weak Scaling : If new machine has K times more resources, can we solve a bigger problem size ?



Amdahl's Law for Multicores

[Marty and Hill, 2009]



- Multicore Chip partitioned into
  - multiple cores (includes L1 cache)
  - uncore (Intel terminology for Shared L2 cache, L3)

- Resources per-chip bounded
  - Area, Power, $, or a combination
  - Bound of total N resources per-chip.
  - How many cores ? How big each ?

Core Types

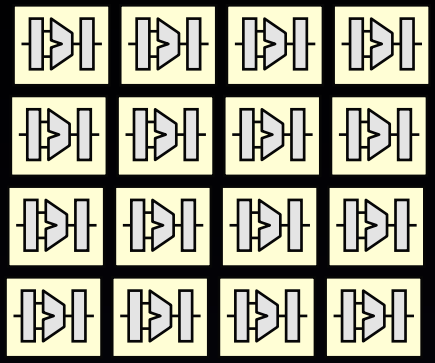
- Your favorite trick can be used to improve single-core performance using same resource
  - becoming increasingly hard to do power-efficiently
- Wimpy Core :
  - Consumes 1 CU (CU: measure of core resources)
  - performance = 1
- Hulk Core:
  - consumes R CUs
  - performance = perf(R)

Hulk Cores

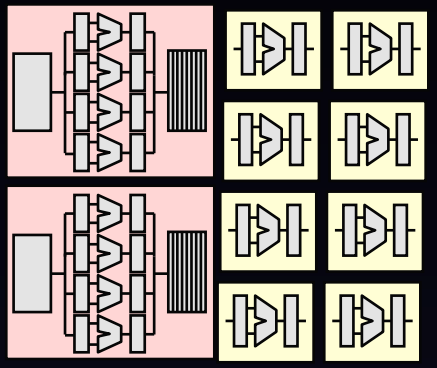
- If Perf (R) >= R ; always use the hulk cores.
  - speeds up everything
- Unfortunately, life isn't easy Perf (R) $<$ R
- Assume Perf (R) = $\sqrt{2}$
  - reasonable assumption?
  - Microprocessor examples seem to indicate
- How to design core for specific Perf (R)
  - coming up later in the latter 1/3rd of talk
  - basic idea: do many instructions in parallel

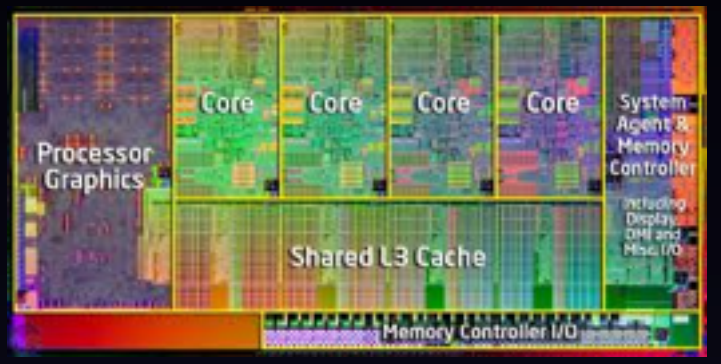
Multicores under consideration

Symmetric

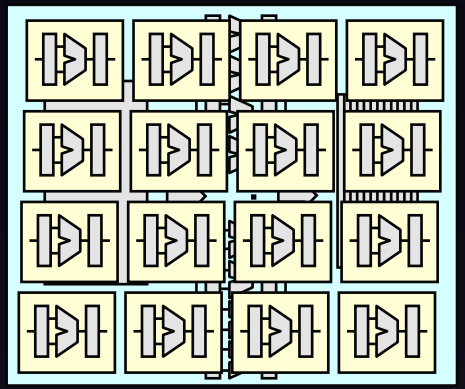


Asymmetric

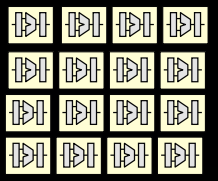




Morphing

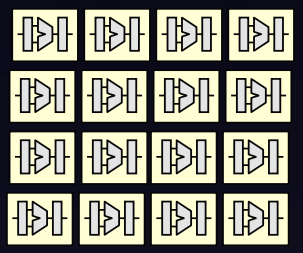


Symmetric Multicores

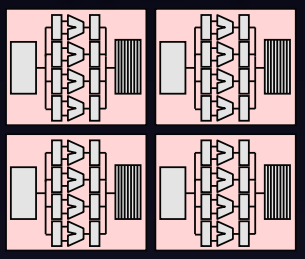


- How many cores ? How big each core ?
- Chip is bounded to N CUs
  - each core has R CUs
- Number of cores per-chip = N/R
- For example, lets say N = 16

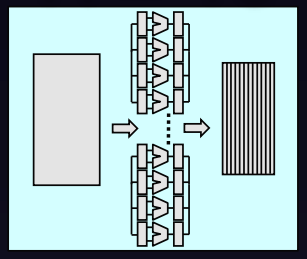
R = 1



R = 4



R = 16



Symmetric Multicore : Performance

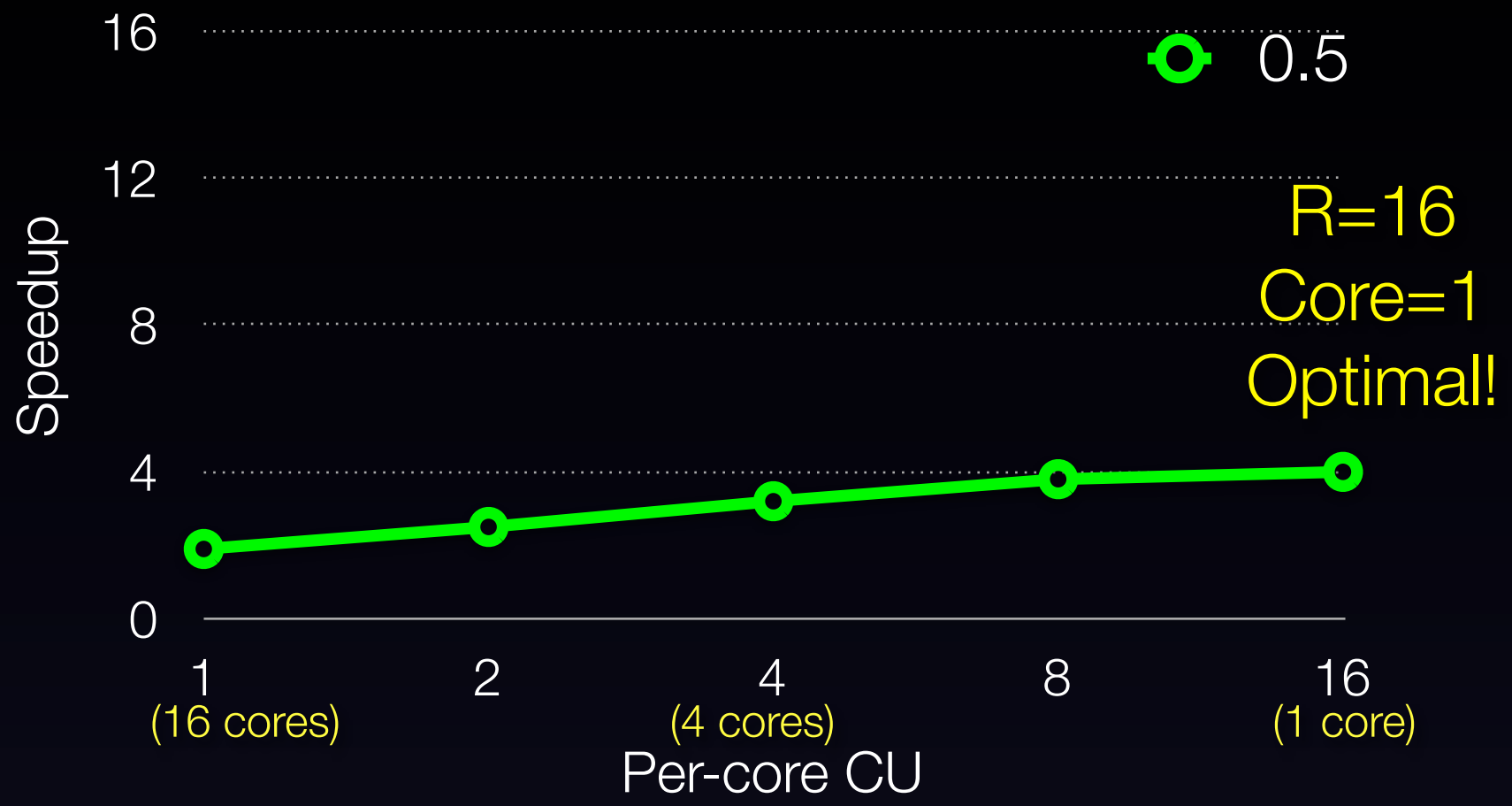
- Serial Phase (1-F) runs on 1 thread on 1 core
  - performance $\propto$ Perf (R)
  - Execution time = (1-F) / Perf (R)
- Parallel Phase uses all N/R cores. Core @ Perf (R)
  - Execution time = F / [Perf (R) * N/R]

$$\text{Speedup} = \frac{1}{\frac{1-F}{Perf(R)} + \frac{F*R}{Perf(R)*N}}$$

Serial Phase Perf(R) ↑

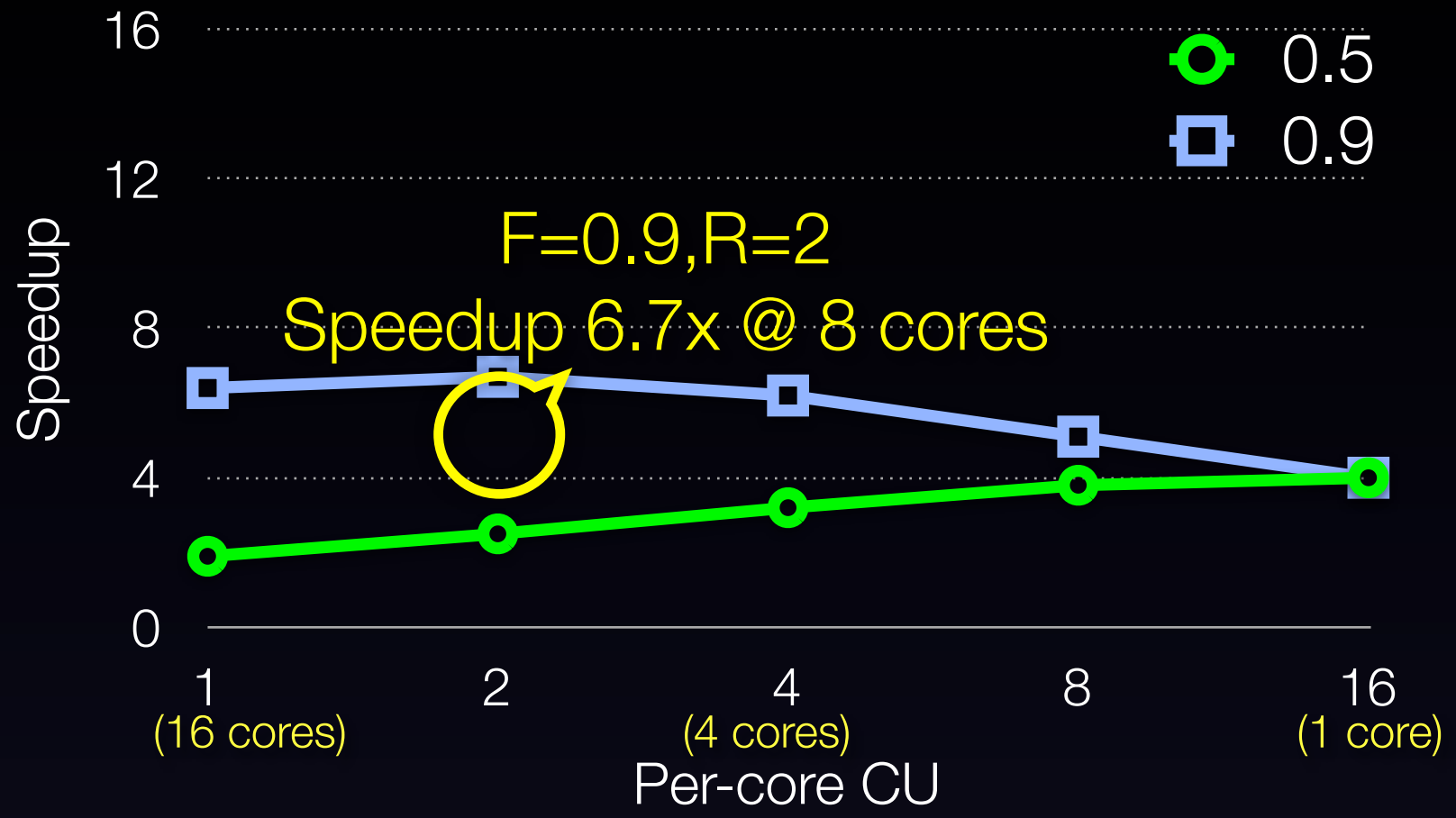
II Phase More cores!

Symmetric Multicore (Chip = 16 CUs)



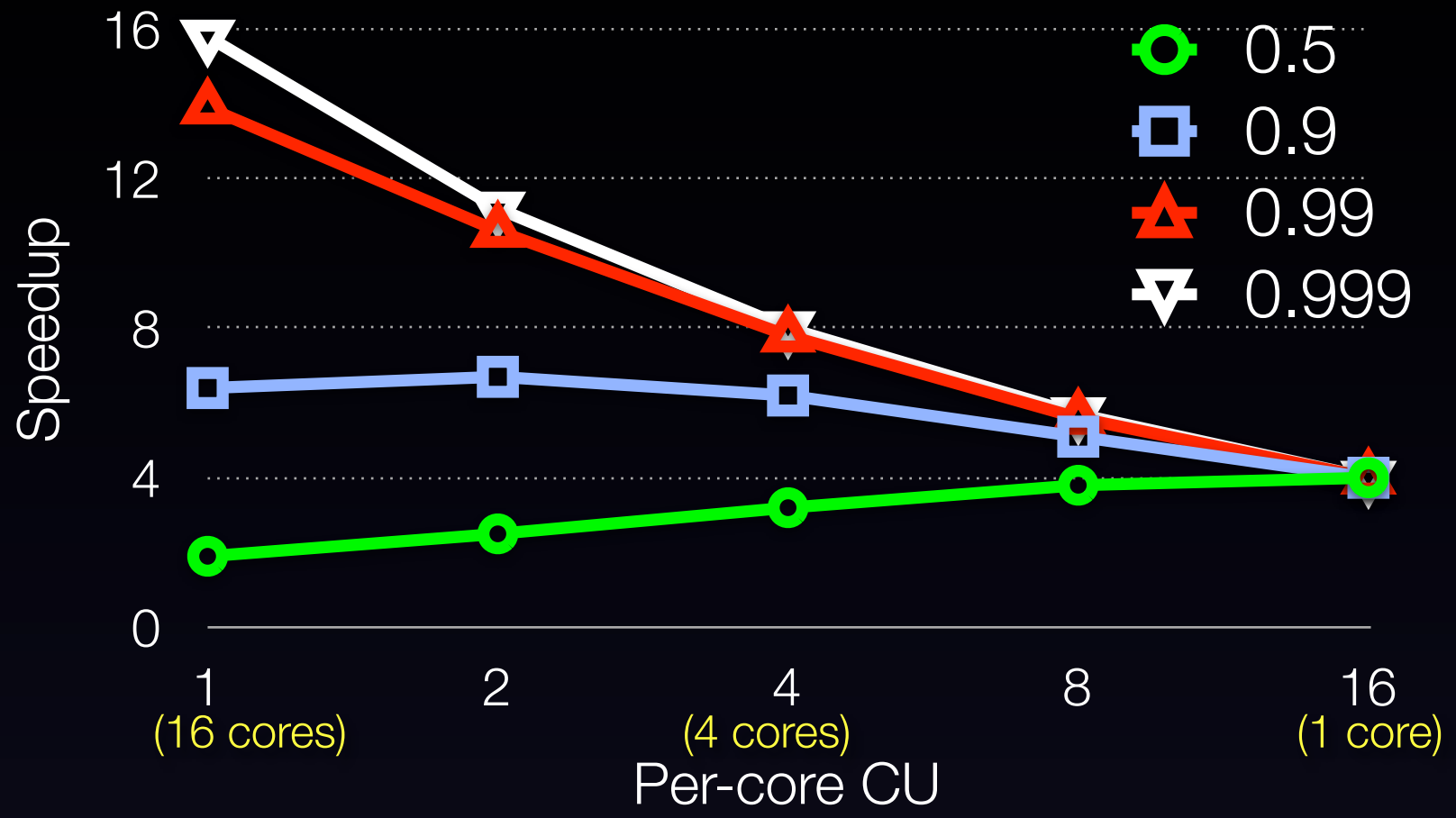
- Need lots of parallelism in multicore world!

Symmetric Multicore (Chip = 16 CUs)



- More parallelism helps; but limited speedup!

Symmetric Multicore (Chip = 16 CUs)



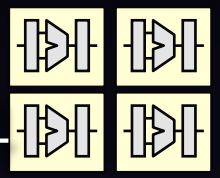
- Applications with high F;
  - significant performance loss with bigger cores
  - Performance loss $\propto \frac{R}{\sqrt{R}} = \sqrt{R}$

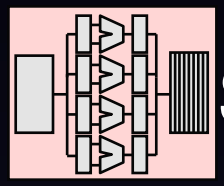
Model-bias towards parallelism

- Remember Perf (R) when scaling up CPU = $\sqrt{2}$
- Lets say 1st gen 1 CU system = 1 CU

- Now consider 2nd gen 4 CU system
  - Four 1CU cores or One 4CU core?
  - When F=0.999; always pick Four 1CU cores

F=0.999
Speedup ~4





Speedup = 2

- Even parallel fraction not perfectly parallel
  - Synchronization, Contention, Locks etc
  - Need SW-Perf(R) (depends on application)

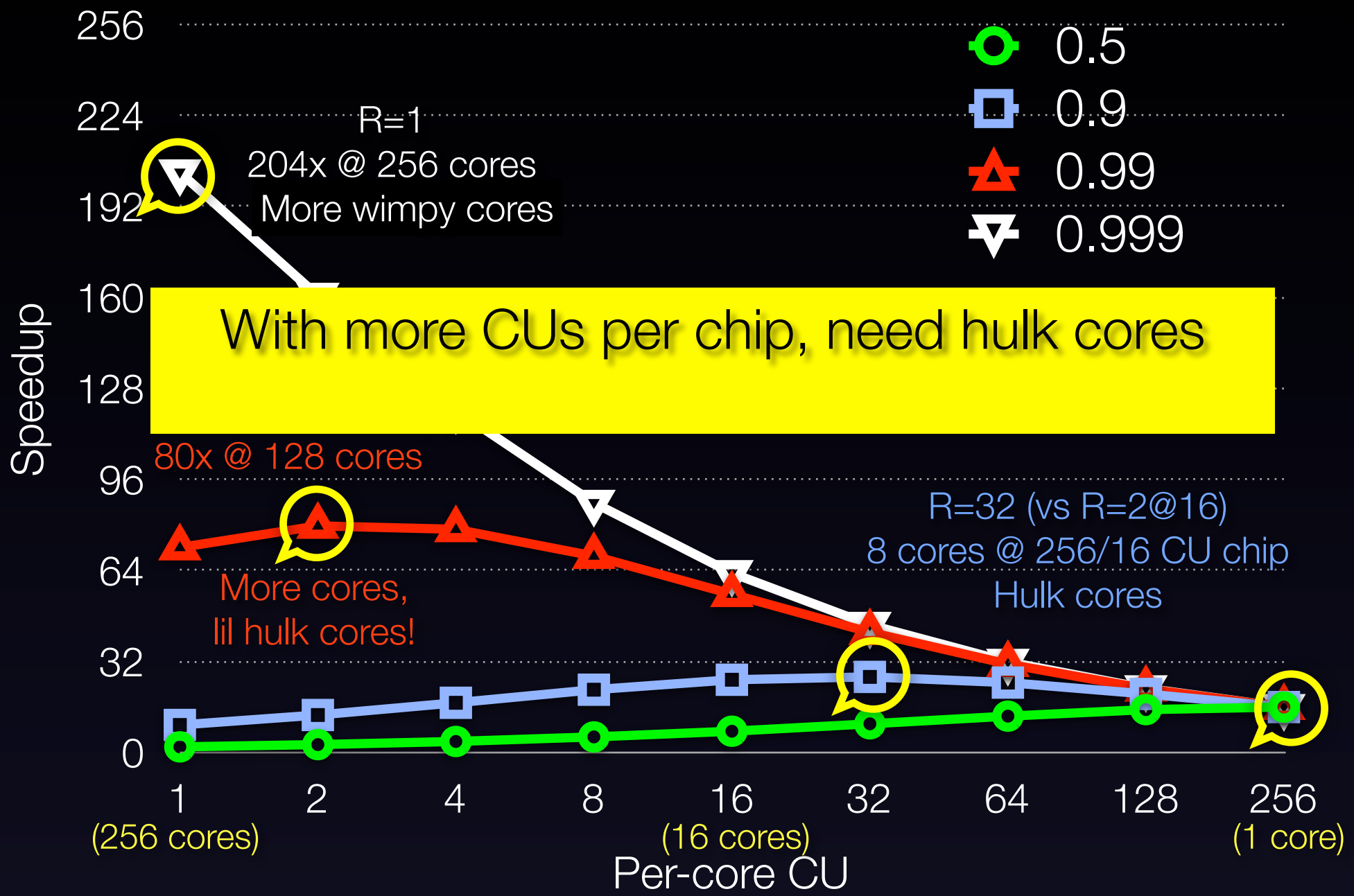
Multicore Moore's Law

- Since 1970s Technology Moore's Law
  - Double transistors every 2 years.
  - Should possibly continue....

- Microarchitect's Moore's Law
  - double single-thread performance every 2 years
  - Stopped due to power required

  - Multicore's Moore's Law
  - 2x cores every 2 years (1 in 2007- 8 in 2010)
  - Need to double software threads every two years
  - Need HW to enable 2x threads every two years

Symmetric Multicore (Chip = 256 CUs) Er



Cost-Effective Multicore Computing

- Is Speedup (N cores) < N that bad ?

- It depends on cost of adding cores.
  - $$$, Power
  - Cost-ratio = Cost (N_{cores}) / Cost (1)

- If chip budget is cost, Cost-ratio << 1.
  - Much of multicore cost outside core [IEEE 1995]
  - Caches, Memory Controller etc.
- If power is cost, cost-ratio can approach 1

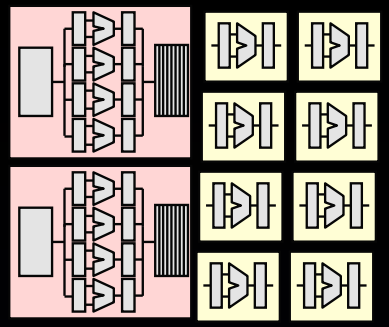
- Multicore computing effective if Cost-ratio > N
  - AMD 6 core = $470 ; AMD 8-core 580$
  - If 8-core speedup >1.2x, then cost-effective

Multicores in Servers and Clients

- Multicore parallelism where cost ratio is low and app
- Clients (high F is hard)
  - Smart-phones just moved to dual-cores
  - how many cores?
- Servers
  - can use vast parallelism (Mapreduce, data analysis)
  - natural overlap across clients

May cause move to cloud computing

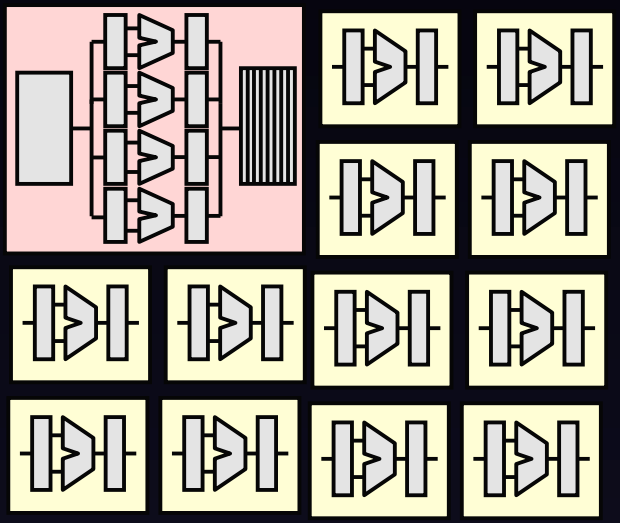
Asymmetric Multicores



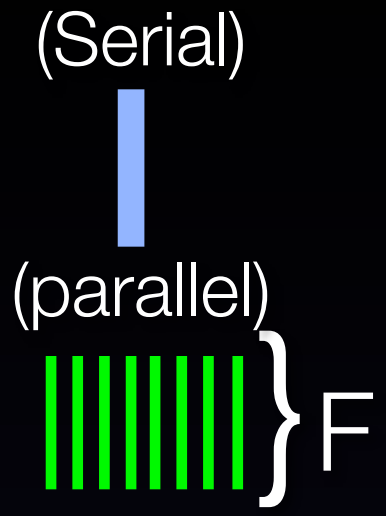
- Enhance some cores to improve performance for serial phase.
- Many designs possible (In this talk, 1 Hulk core)
- How to enhance core ?
  - coming up in last 1/3rd of class

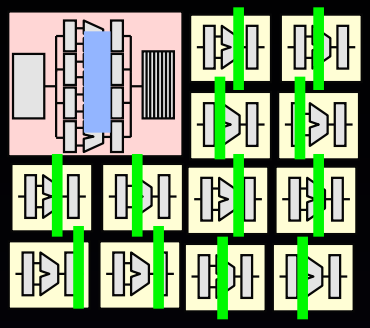
Asymmetric Multicores

- Total chip resources = N CUs
- Assume two-types of cores on-chip
  - One core = R CU, N-R 1 CU cores
  - Total cores = N-R+1



Asymmetric Cores : Performance





Serial Phase = (1-F) / K*Perf (R)

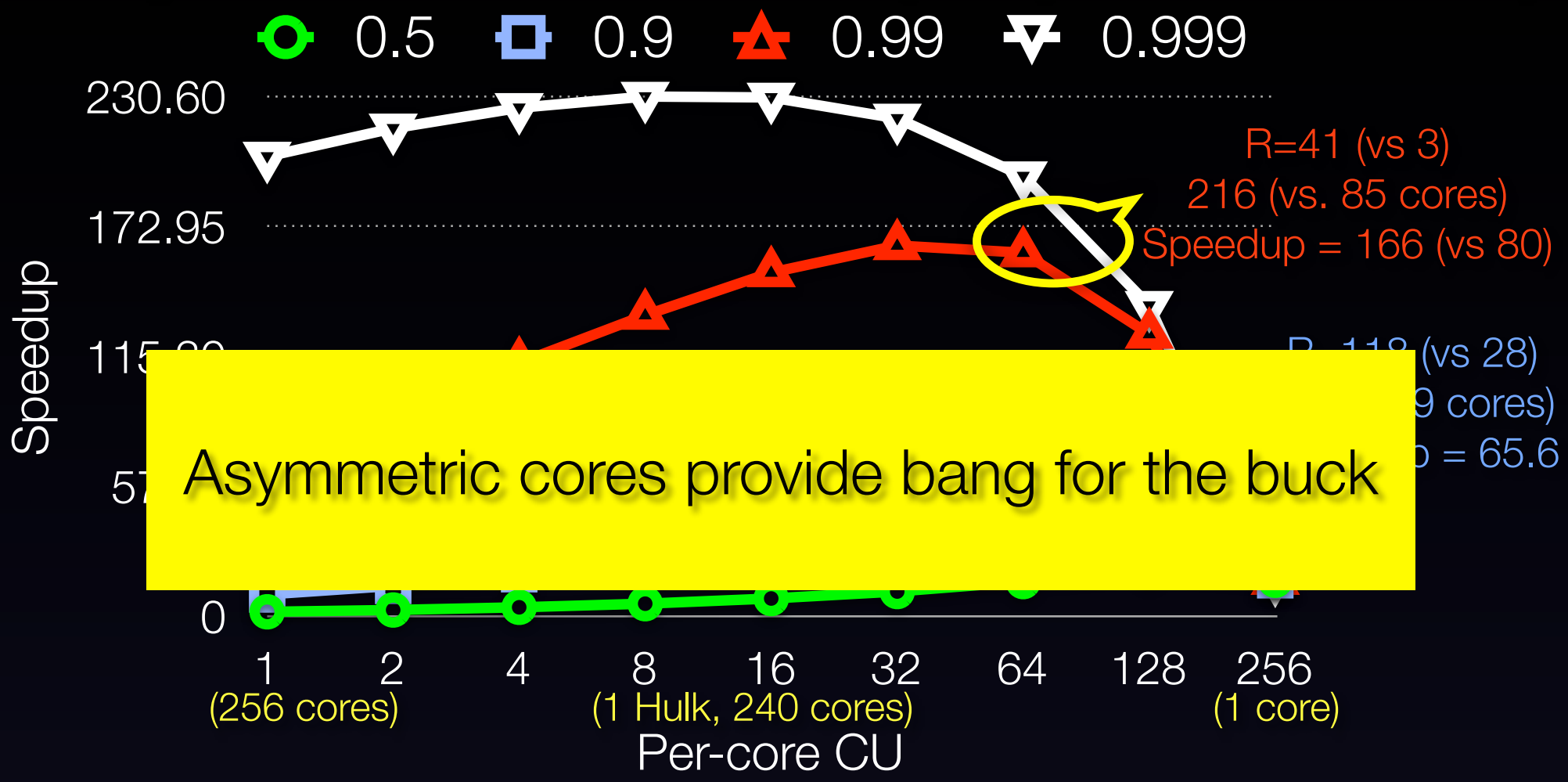
Parallel Phase = (F) / [K*Perf (R) + N-(K*R)]

where K is # of Hulk cores.

In our case, K = 1

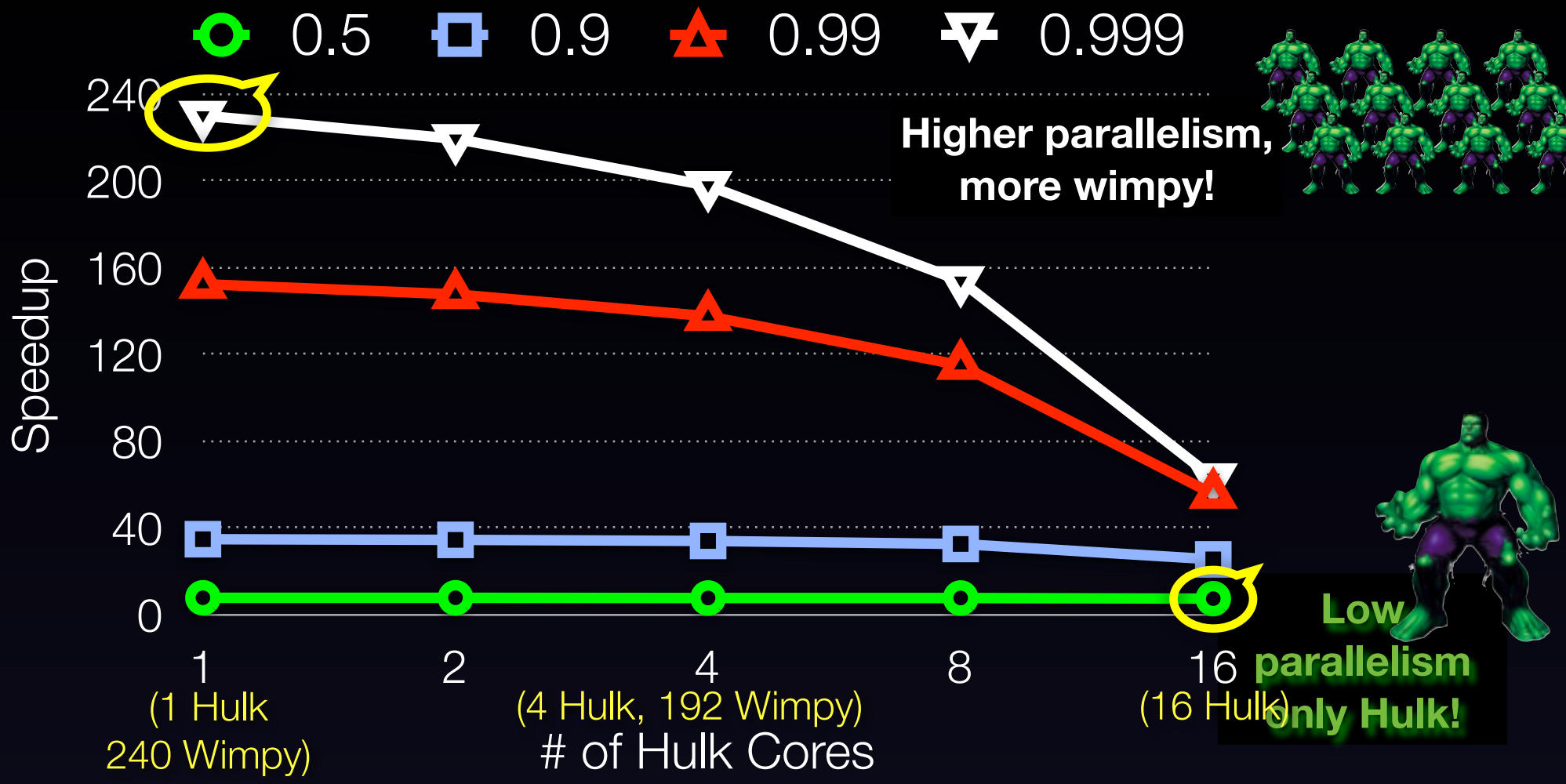
$$\text{Speedup} = \frac{1}{\frac{1-F}{Perf(R)} + \frac{F}{Perf(R)+N-R}}$$

Asymmetric Multicore (Chip = 256 CUs)



- Asymmetric cores offer great potential
  - with 1 Hulk core, speedup increases significantly
  - helps take care of Amdahl's law

Asymmetric Multicore (Chip = 256 CUs)



As F increases, always increase wimpy cores!

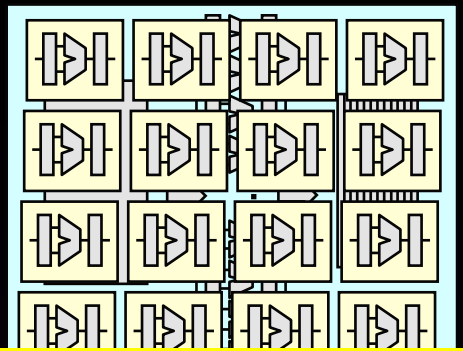
Asymmetric Multicores : Challenge

Task Management :
How to schedule computation?

Locality :
How to keep data close to task?

Coordinate Tasks :
How to synchronize data?

Morphing Multicores



- C
  - Advantage : Can harness all cores on the chip
  - Core optimized
- At runtime glue R 1CU cores to create R CU core
  - improves performance for serial phase
- How to dynamically glue cores ?
  - Not the focus; need's future research

Morphing Multicores : Performance

- N 1CU cores, from which R 1CU cores glued
- Serial phase uses R CU core at Perf (R)
  - execution time = (1-F)/R
- Parallel phases uses N cores
  - execution time = (1-F)/N

$$\text{Speedup} = \frac{1}{\frac{1-F}{Perf(R)} + \frac{F}{N}}$$

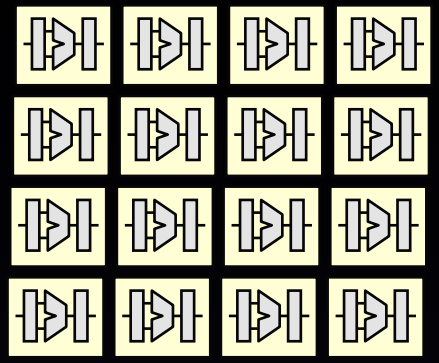
Morphing Multicore (Chip = 256 CUs)

Morphing multicores are awesome!
Especially at higher chip resource levels

How to glue!

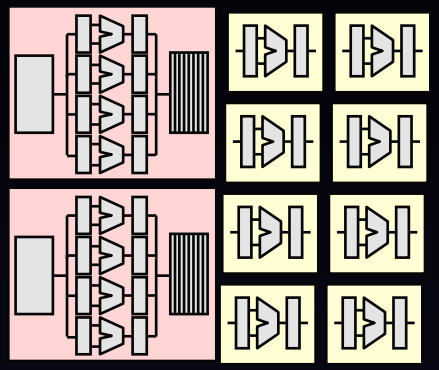
Multicore Amdahl's Law

Symmetric



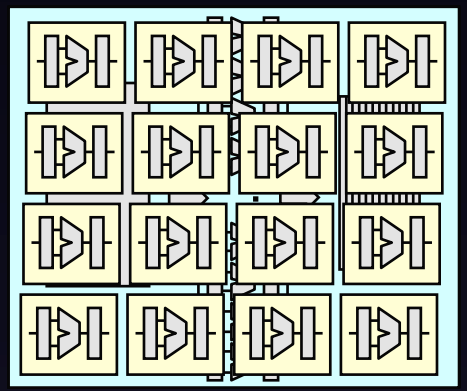
$$\frac{1}{\frac{1-F}{Perf(R)} + \frac{F*R}{Perf(R)*N}}$$

Asymmetric



$$\frac{1}{\frac{1-F}{Perf(R)} + \frac{F}{Perf(R)+N-R}}$$

Morphing



$$\frac{1}{\frac{1-F}{Perf(R)} + \frac{F}{N}}$$

Challenges (1/2)

- Serial Fraction (1-F) has fine-grain parallelism
- Parallel Fraction (F) has serialization overheads
  - You will learn in the next 2-3 weeks.
- Software challenges for asymmetric and dynamic multicores
- How much parallelism in future software?

Challenges (2/2)

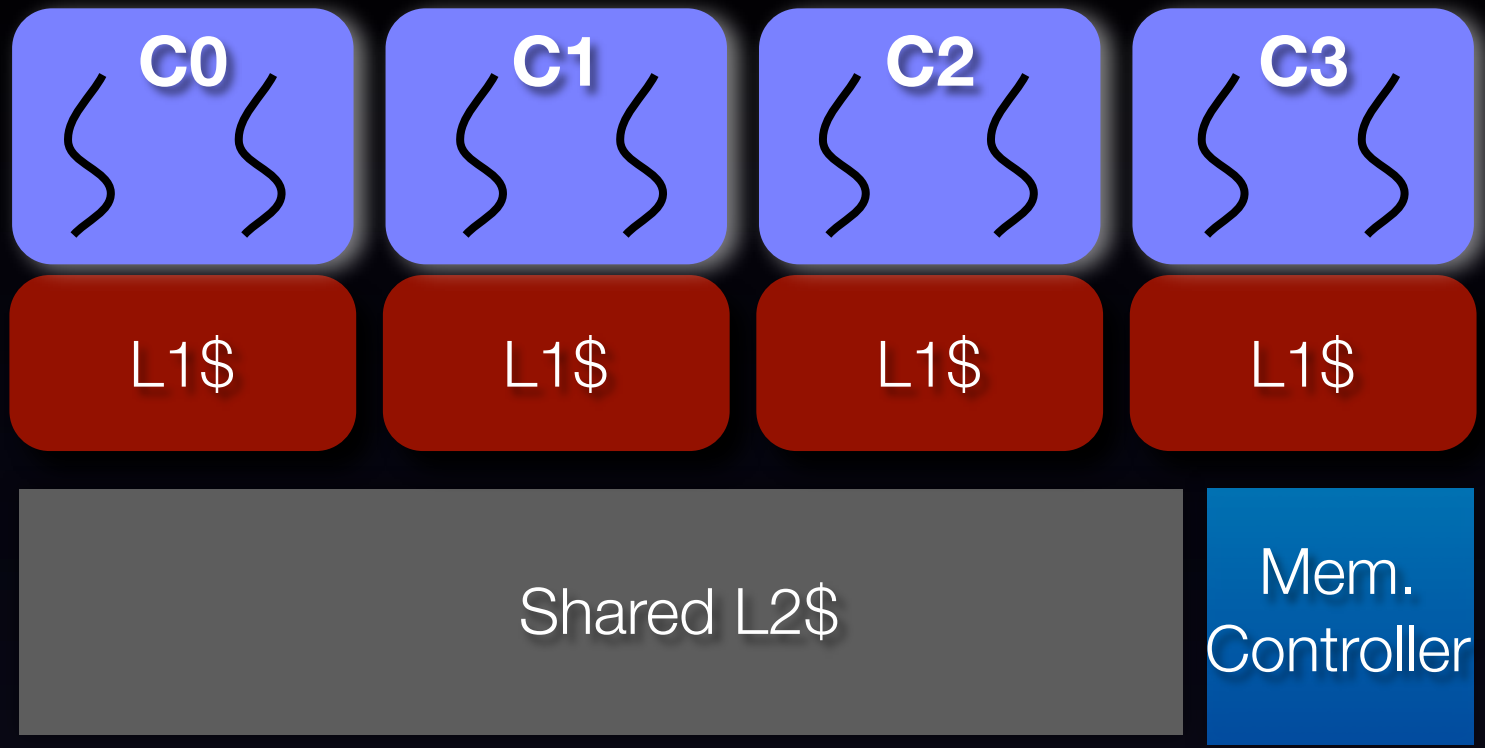
Parallelism all the time ?

Amdahl's Law affects serial fraction ?
Need to increase core speed.

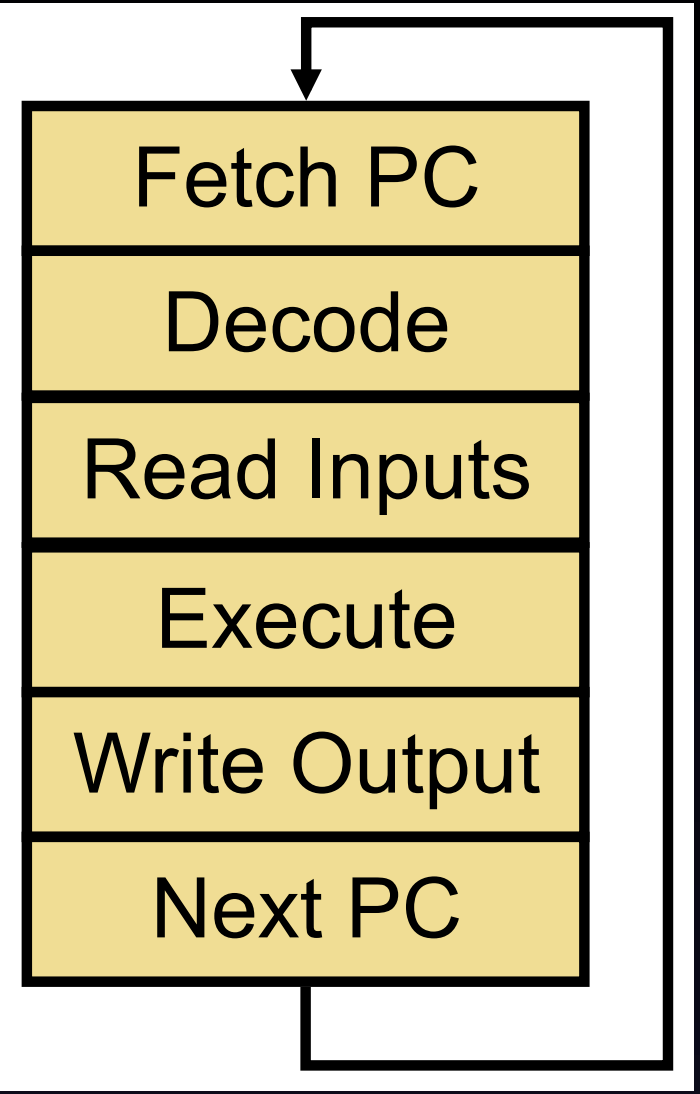
Lots of walls: Power, Area, Shared caches
How to scale CPU performance?

Outline

- How did we get here. Why Multicores?
- Amdahl's Law: How do Apps scale on Multicores?
  - Symmetric, Asymmetric, Morphing
- Dissecting a Processor Core
  - Different types, Performance etc.

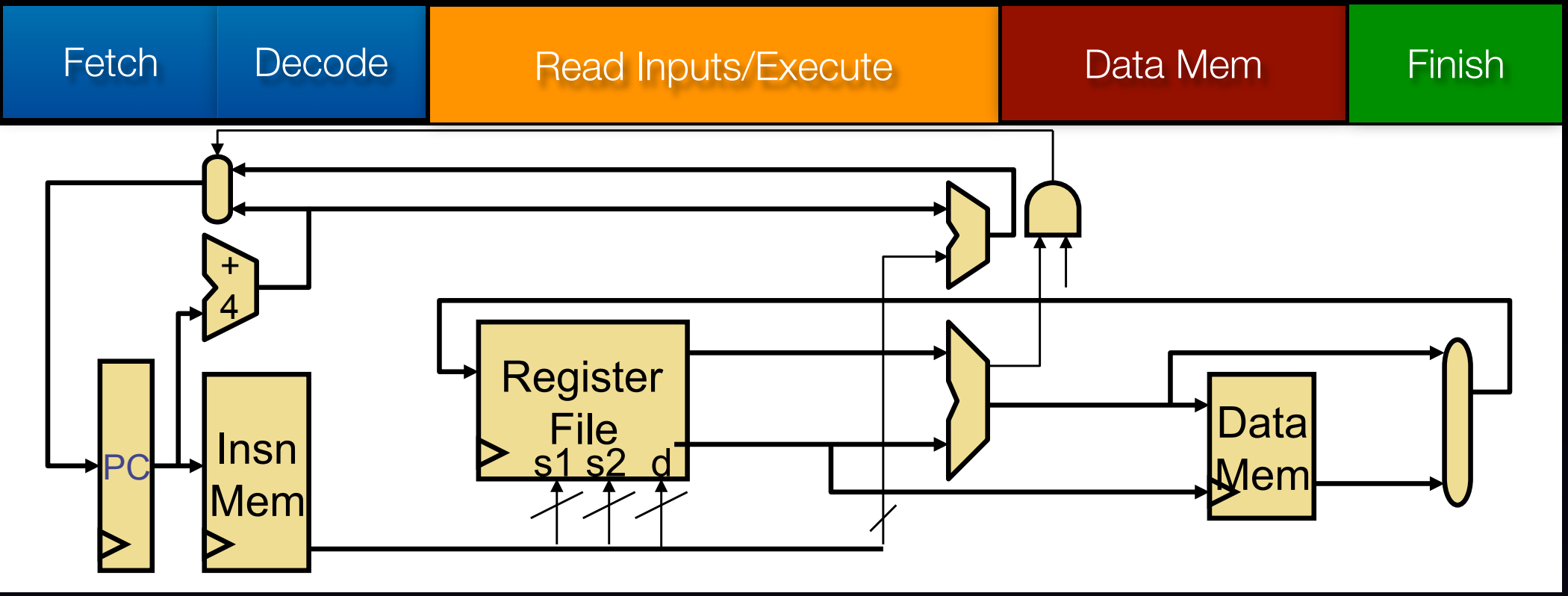

C0
C1
C2
C3
L1$
L1$
L1$
L1$
Shared L2$
Mem.
Controller

The core



- Basic loop in each core
  - program order on dynamic instructions
- Ins is Memory[PC]
  - typically, Next PC = PC+1;
- Atomic illusion
  - ins X finishes before X+1
  - can break constraint
- Iron-Law Performance =
  # ins * (cycles/ins) * (seconds/cycle)

How to implement the loop?

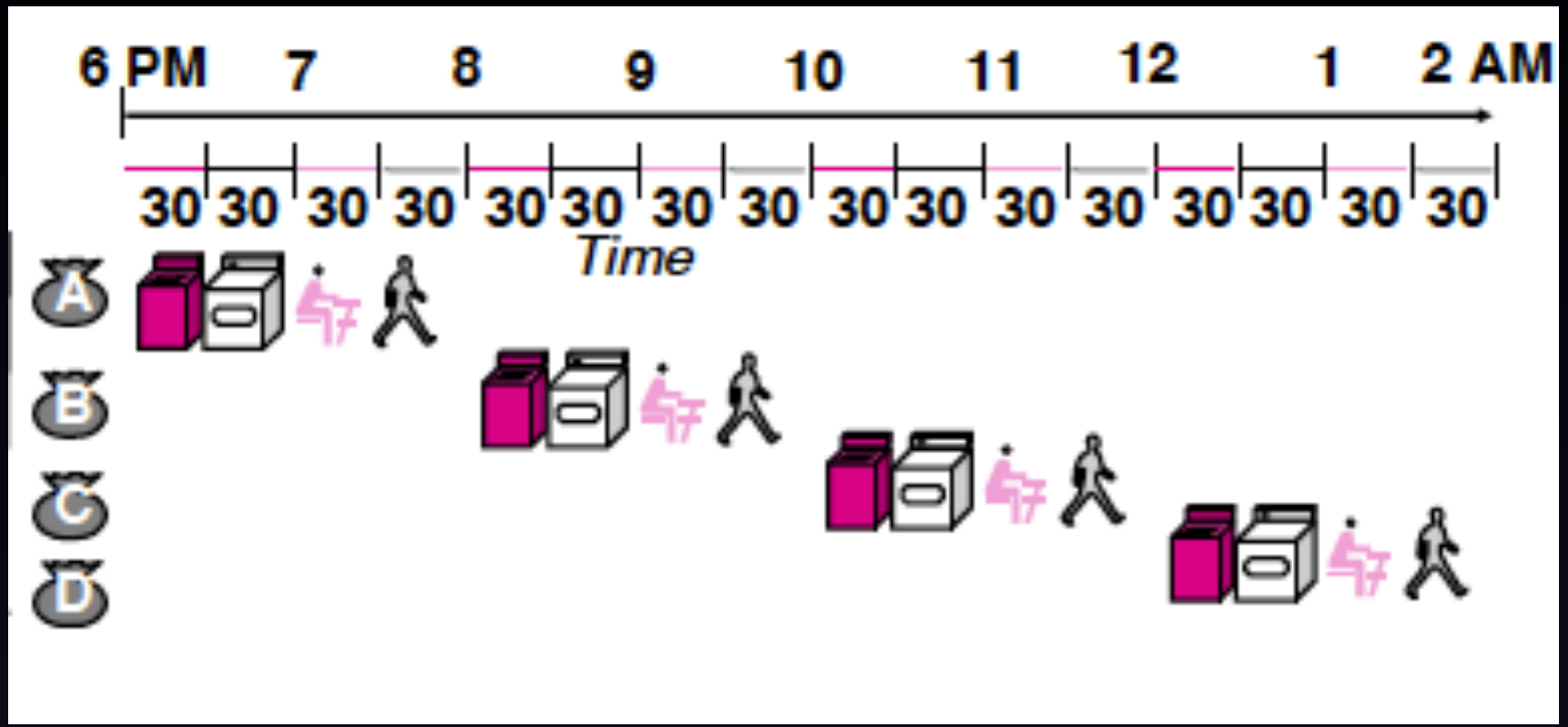


Pipelines (Laundry Analogy)

- Amy, Bob, Cathy, Dave each have to wash, dry and fold
  - Washer : 30minutes
  - Dryer : 30 minutes
  - Folder : 30 minutes
  - Stasher : 30 minutes
- Pipelining doesn't help latency.
  - Single task takes same time
- Speedup = # of stages
  - Pipeline only as fast as slowest stages

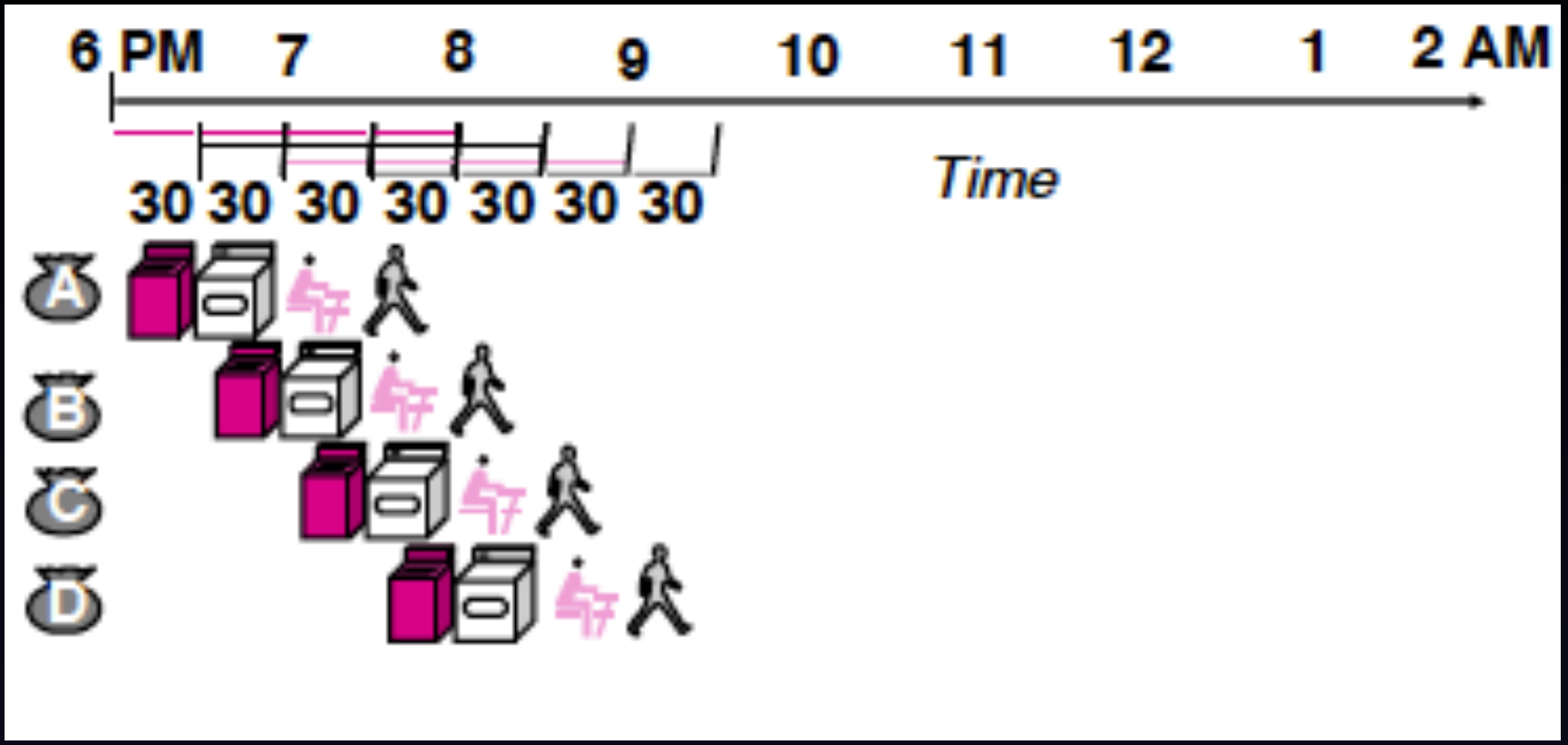
One-at-a-time Laundry

Total Time : 4 * (T-laundry+T-drier+T-Fold+T-stash.)



Pipelined Laundry

- Partition total work into stages
  - start specific stage ASAP

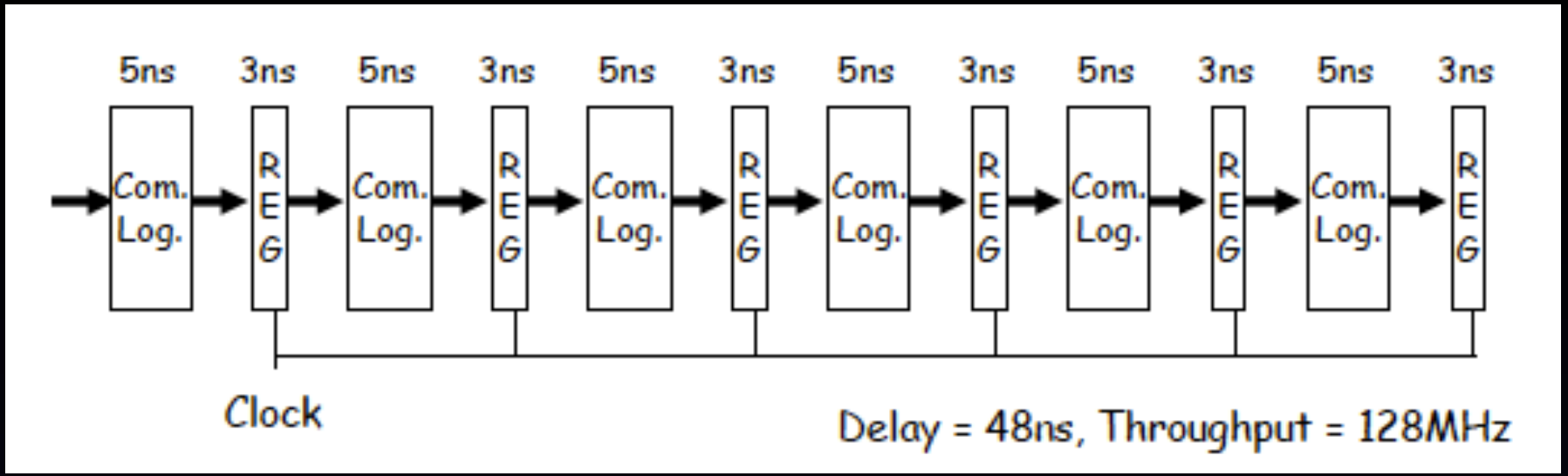


Total Time : (T-laundry+T-drier+T-Fold+T-stash) + (4 -1)*30m

Wimpy Core

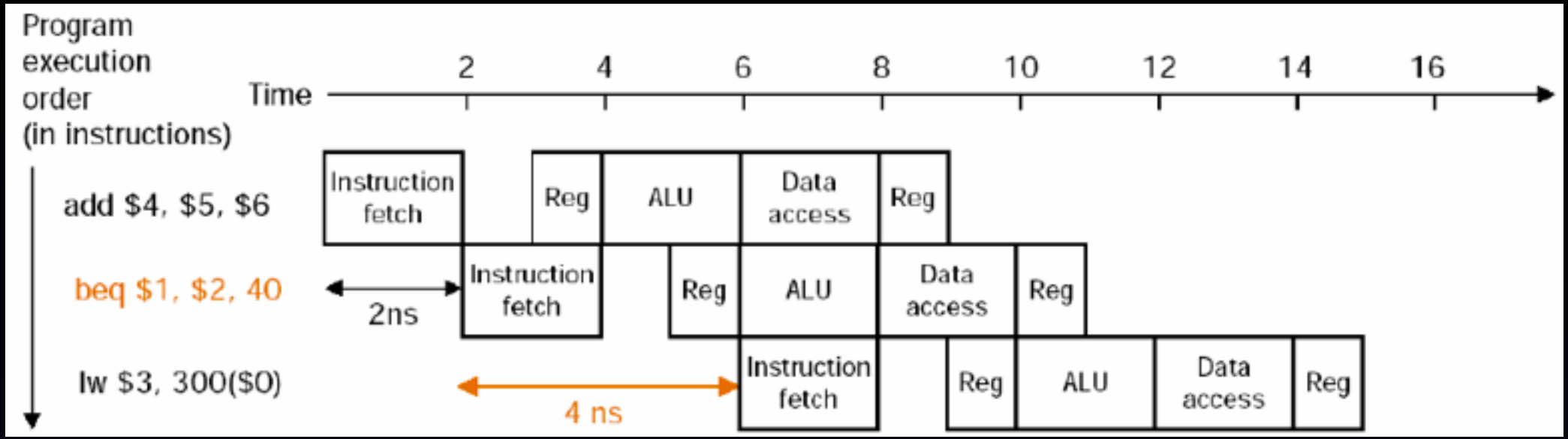
- Pipelined-only In-order CPUs offerings
  - MIPS, Niagara, Intel Atom, ARM (early cell-phones)
- Benefits
  - Simple design
  - Not too much wastage
- Challenges (performance brittle)
  - stalls due to control
  - stalls due to data dependencies
  - only # stages instructions in flight (limited ins. parallel)
  - overhead of pipeline stages

Deep pipelines

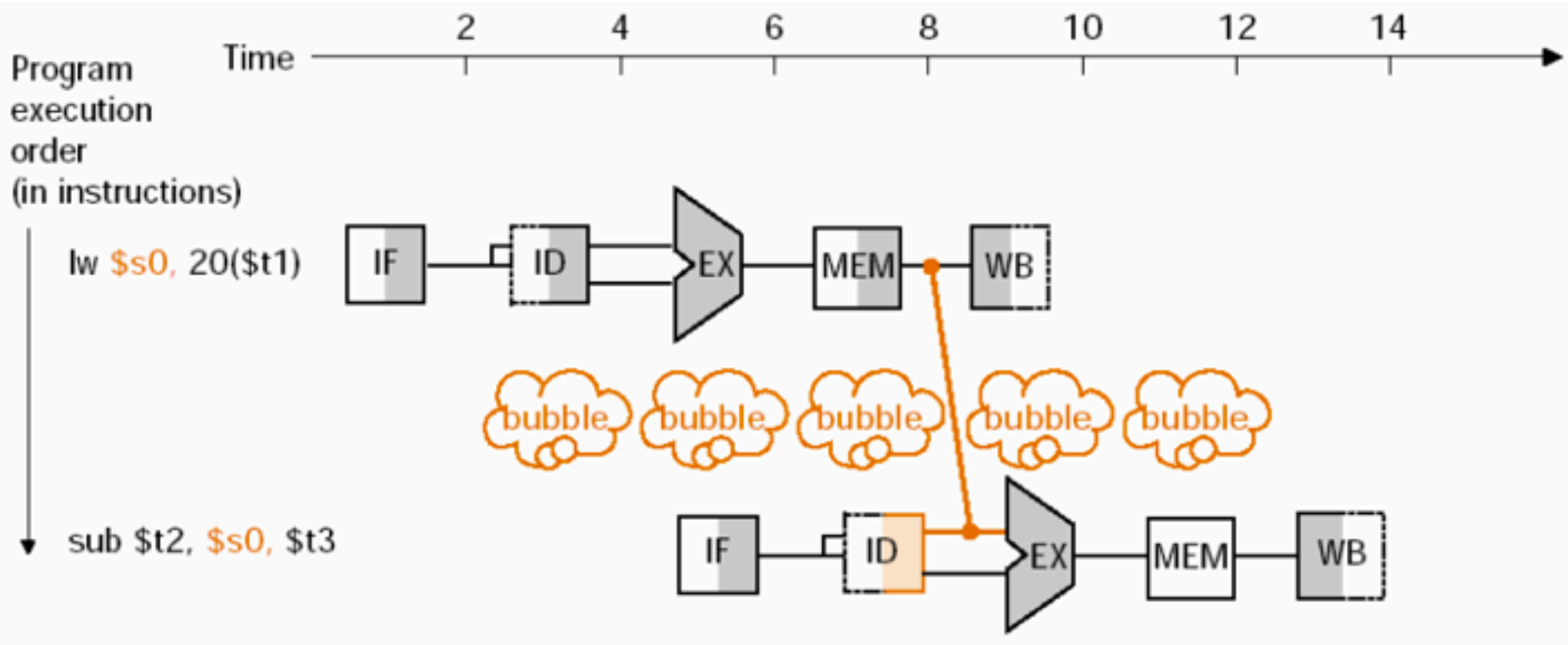


- Diminishing returns
  - overhead of pipeline interface
  - increased latency
  - Pentium 4 (22 stages)

Wimpy Challenge (1/3) : Branches

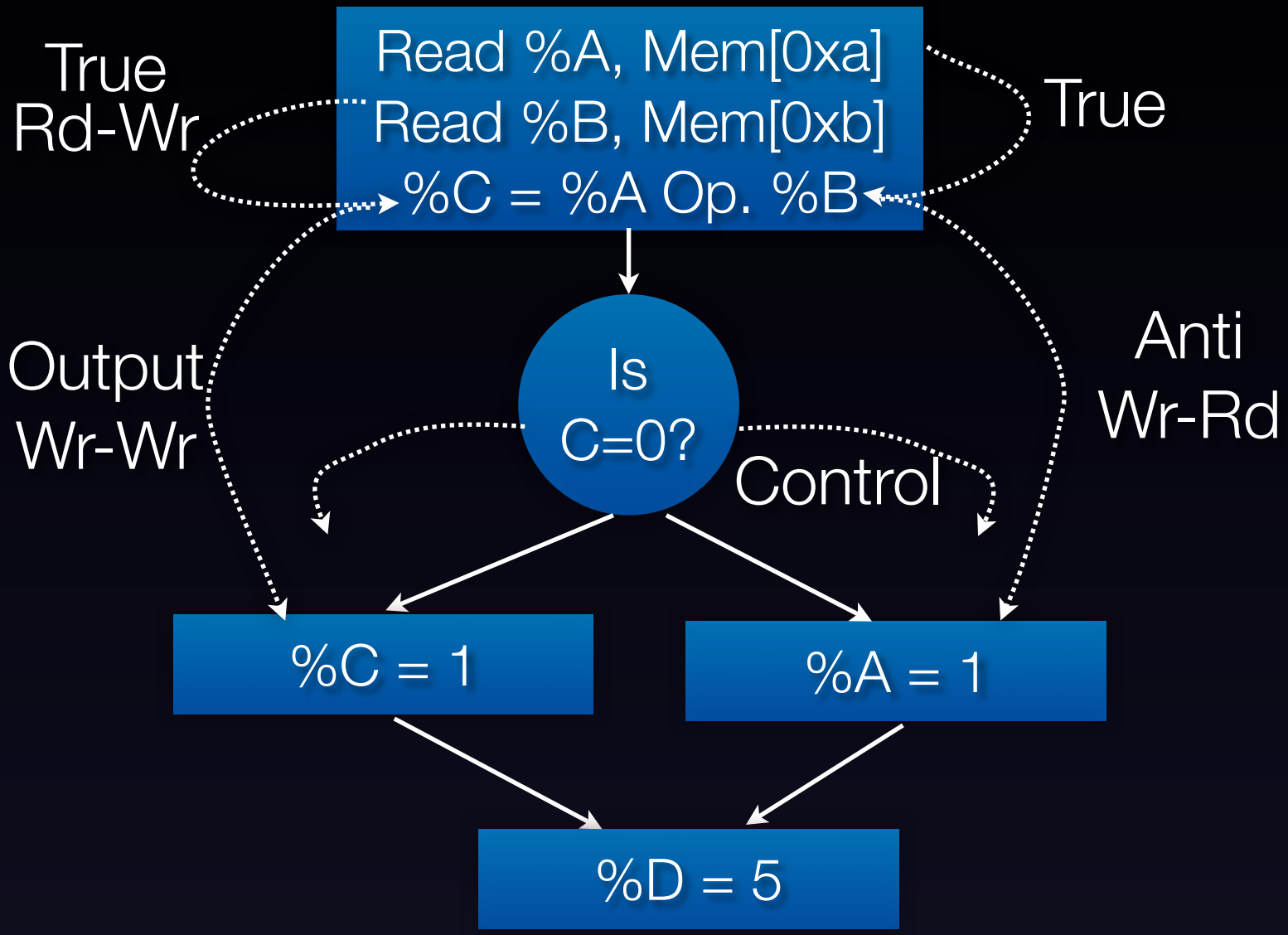


Wimpy Challenge (2/3) : Mem. Ops



Wimpy Challenge (3/3): Data Hazard

- Anti and Output dependencies (no need to wait)



Hulk Core : Basic Strategy

- Build flow-graph from window of instructions

Honor True Dep.

Read %A, Mem[0xa]
Read %B, Mem[0xb]
%C = %A Op. %B

Branches: Speculate

Is C=0?

%C = 1

%A = 1

Anti Dep. : Rename Reg.

C [1.1] = 1

C [1.1] = 1
D [1.0] = 5

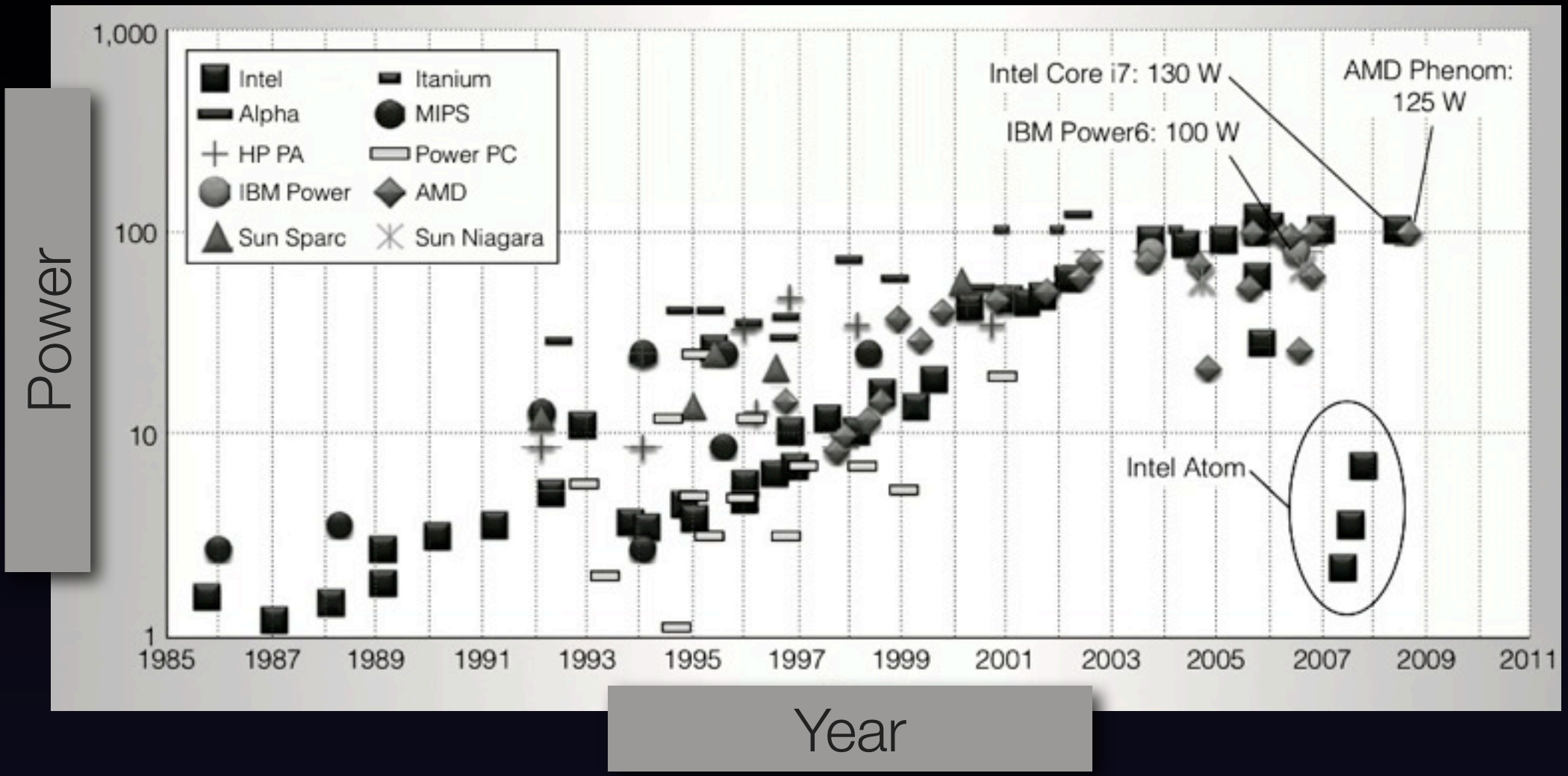
%D = 5

Hulk core : Challenges

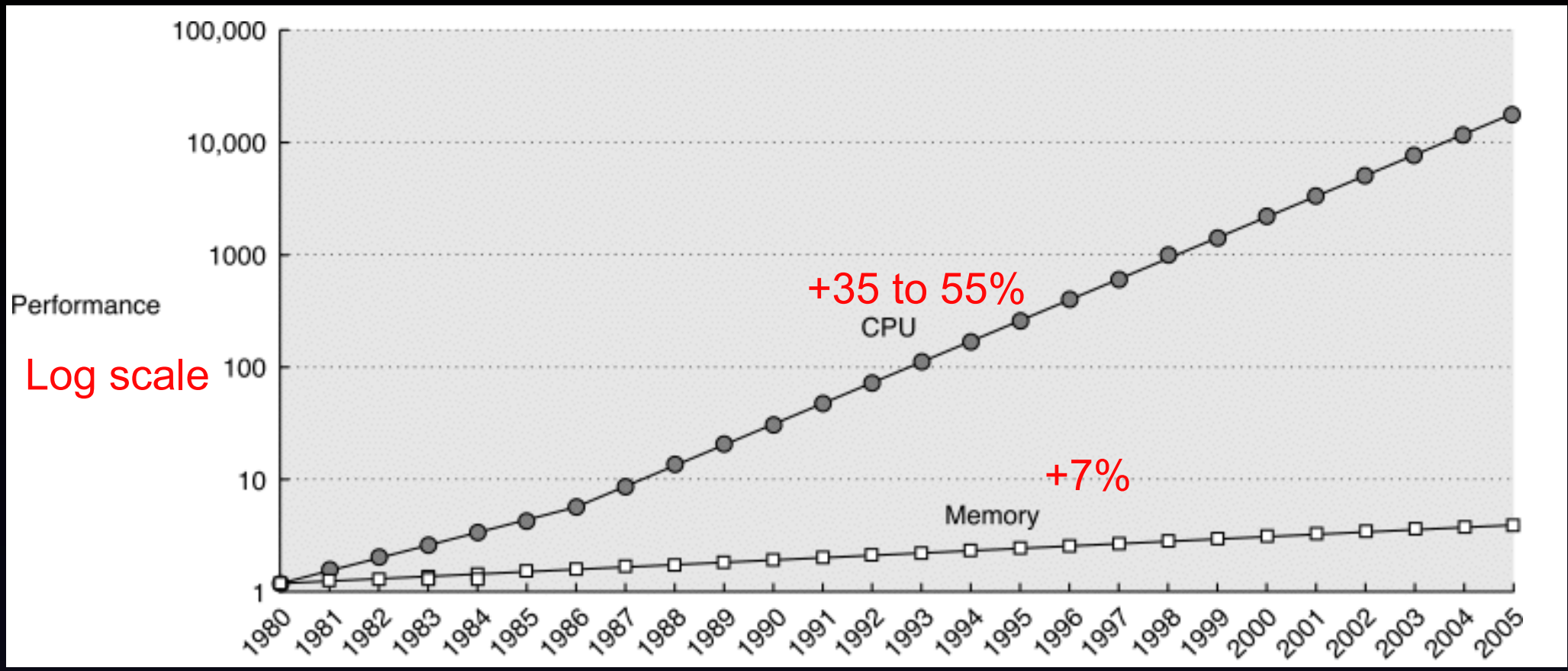
Power

- Cost of speculation
  - hard in data dependent programs (e.g., databases)
  - lots of buffering
  - deep pipelines (on Pentium 4 spec-fail costs 30+cycles)

- Dependence checks
  - storage cost (instructions window)^2
  - propagating dependency expensive (window size)

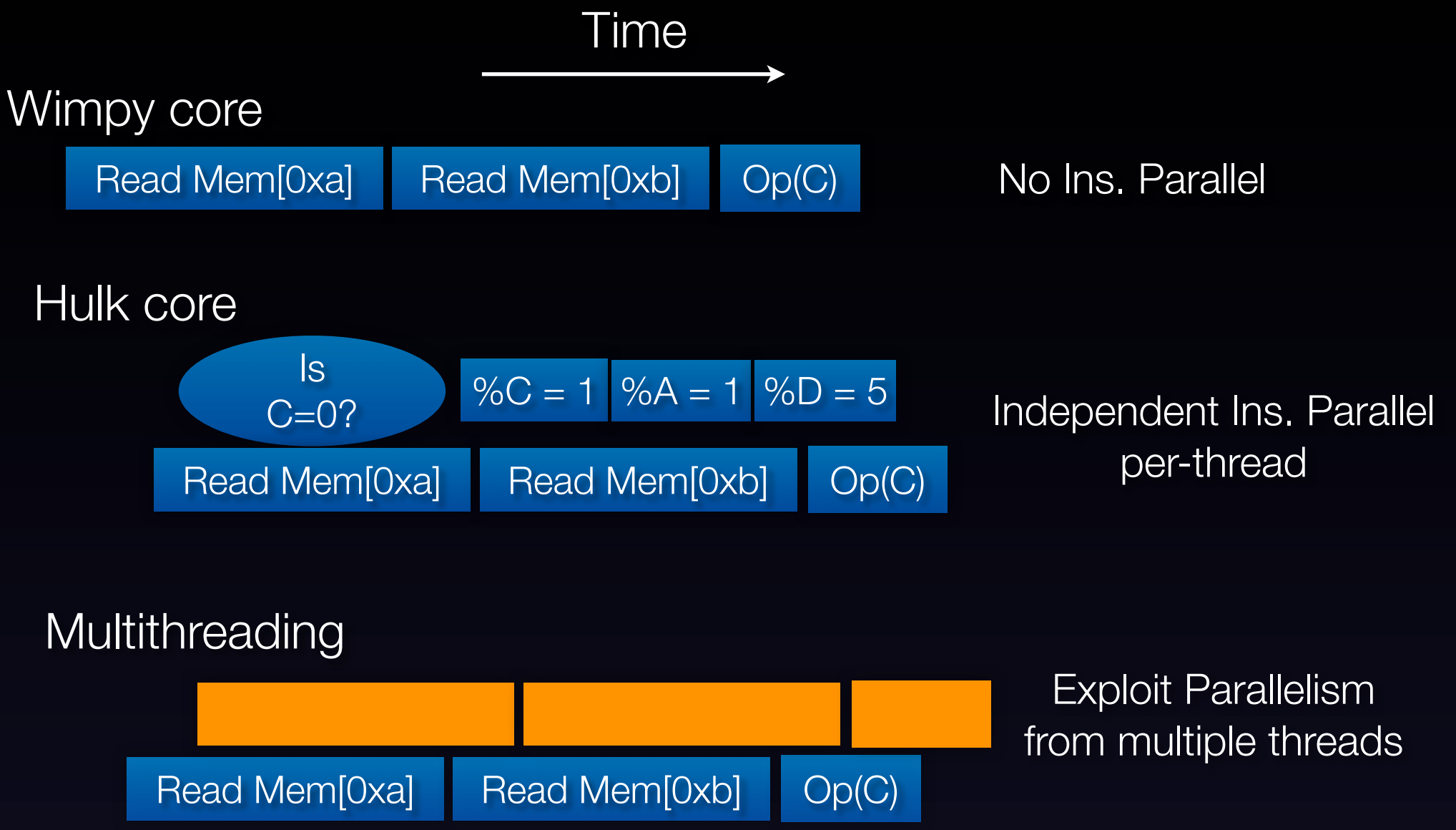

Power
1,000
100
10
1
Intel
Alpha
HP PA
IBM Power
Sun Sparc
Itanium
MIPS
Power PC
AMD
Sun Niagara
Intel Core i7: 130 W
IBM Power6: 100 W
AMD Phenom:
125 W
Intel Atom
1985
1987
1989
1991
1993
1995
1997
1999
2001
2003
2005
2007
2009
2011
Year

Memory is too slow!

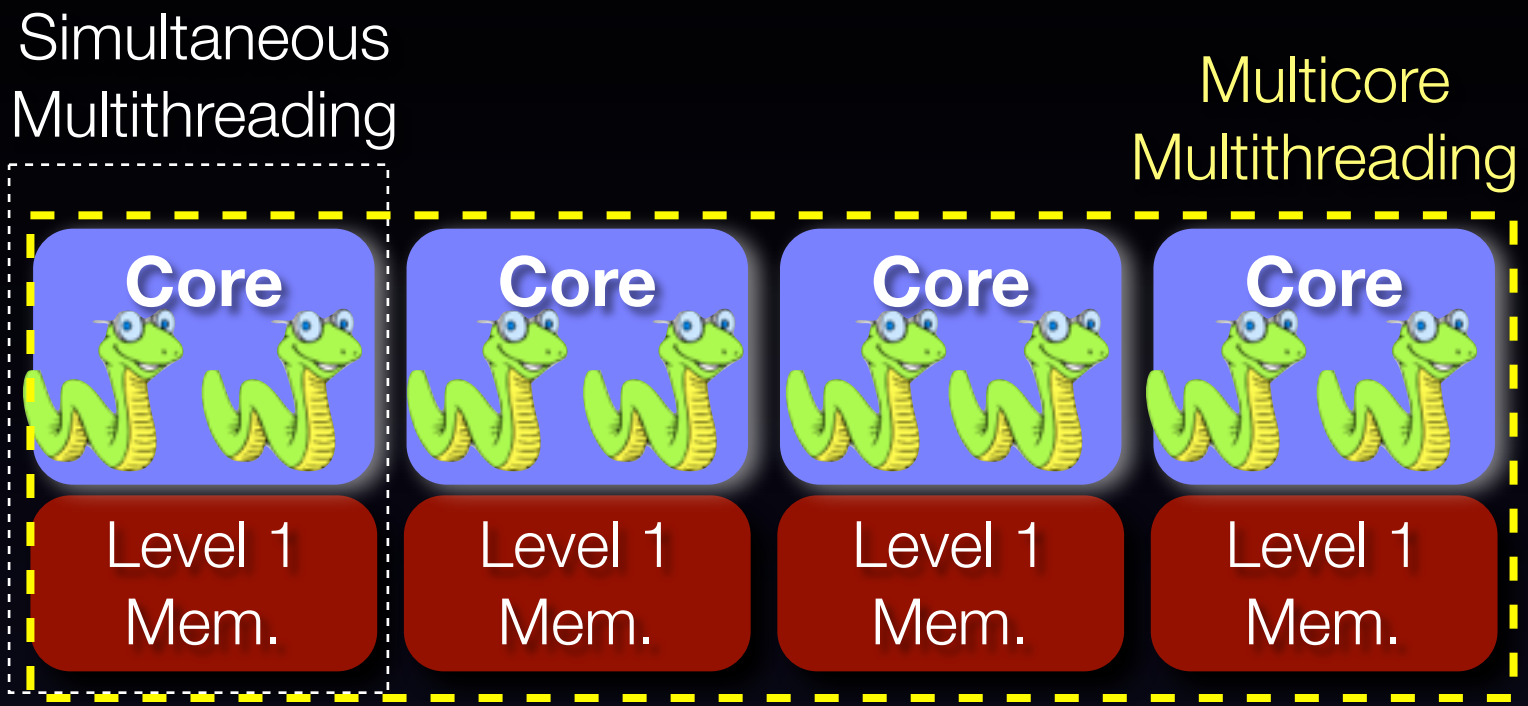


Hulk core window fills up;
Waiting.....It costs energy to wait

Keeping a core busy



What else can core do when waiting ?



What's are these red and orange rectangular regions?