CMPT 473
Software Quality Assurance

Graph Coverage

Nick Sumner

Recall: Coverage/Adequacy

- Can't look at all possible inputs.
- Need to determine if a test suite *covers / is adequate* for our quality objectives.

Recall: Coverage/Adequacy

- Can't look at all possible inputs.
- Need to determine if a test suite *covers / is adequate* for our quality objectives.
- So far: Input & Requirements based

Recall: Coverage/Adequacy

- Can't look at all possible inputs.
- Need to determine if a test suite *covers / is adequate* for our quality objectives.
- So far: Input & Requirements based

Efficiently sort a provided list in under X seconds

```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```

Recall: Coverage/Adequacy

- Can't look at all possible inputs.
- Need to determine if a test suite *covers / is adequate* for our quality objectives.
- So far: Input & Requirements based

Efficiently sort a provided list in under X seconds

```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```

How well can input based techniques test this?

Recall: Coverage/Adequacy

- Can't look at all possible inputs.
- Need to determine if a test suite *covers / is adequate* for our quality objectives.
- So far: Input & Requirements based

Efficiently sort a provided list in under X seconds

```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```

How well can input based techniques test this?

How might we do better?

White Box / Black Blox

- Considering only the requirements or input is a *black box* approach
  - Treats the program like an opaque box
  - No deep knowledge of the program's structure

White Box / Black Blox

- Considering only the requirements or input is a *black box* approach
  - Treats the program like an opaque box
  - No deep knowledge of the program's structure
- Techniques that use artifacts of the program structure are *white box* approaches
  - They can 'see into' the program's implementation

White Box Testing

- What is a simple approach that solves our problem here?

```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```

White Box Testing

- What is a simple approach that solves our problem here?
- Statement Coverage
  - How many of the statements did the suite test?

```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```

White Box Testing

- What is a simple approach that solves our problem here?
- Statement Coverage
  - How many of the statements did the suite test?
- Branch Coverage
  - How many of the condition outcomes were tested?

```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```

White Box Testing

- In this course, we'll mostly look at graph coverage based techniques

White Box Testing

- In this course, we'll mostly look at graph coverage based techniques
  - Most commonly used metrics in the real world

So a bit of review…

White Box Testing

- In this course, we'll mostly look at graph coverage based techniques
  - Most commonly used metrics in the real world
  - Most concepts can be modeled through graphs
    e.g. programs, protocols, use patterns, designs, …

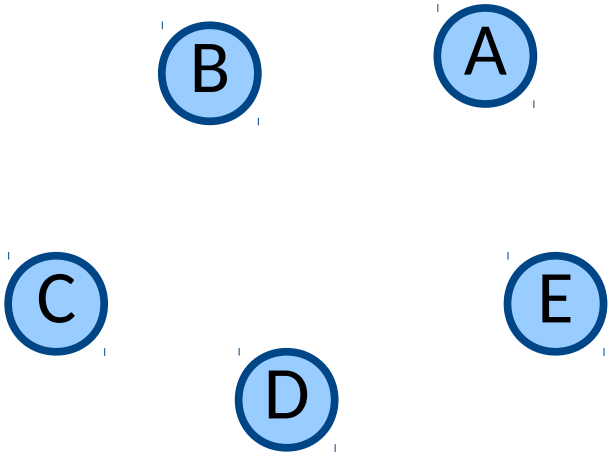
So a bit of review…

Graphs

- What is a *graph* G?

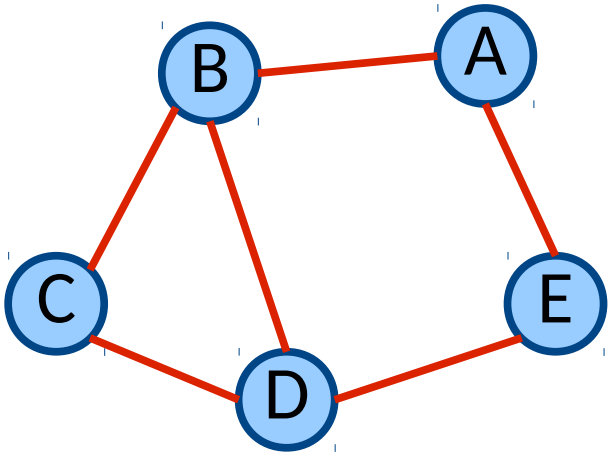
Graphs

- What is a *graph* G?
  - A set N of nodes



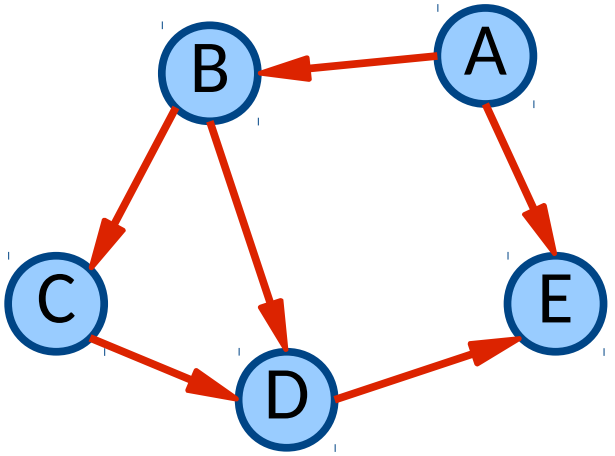
Graphs

- What is a *graph* G?
  - A set N of nodes
  - A set E of edges



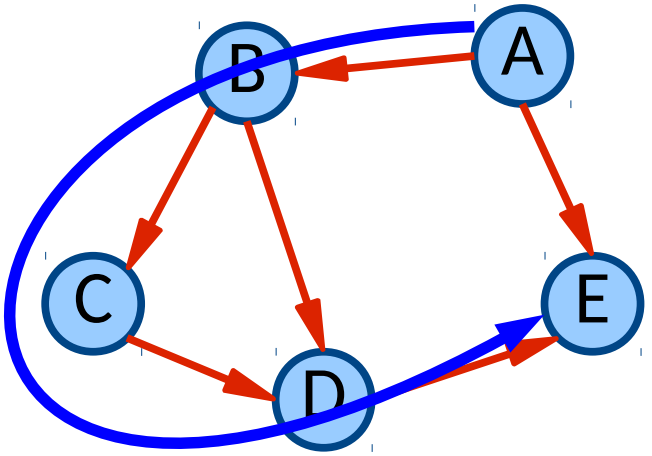
Graphs

- What is a *graph* G?
  - A set N of nodes
  - A set E of edges
- When edges are directed from one node to another, the graph is a *directed graph*



Graphs

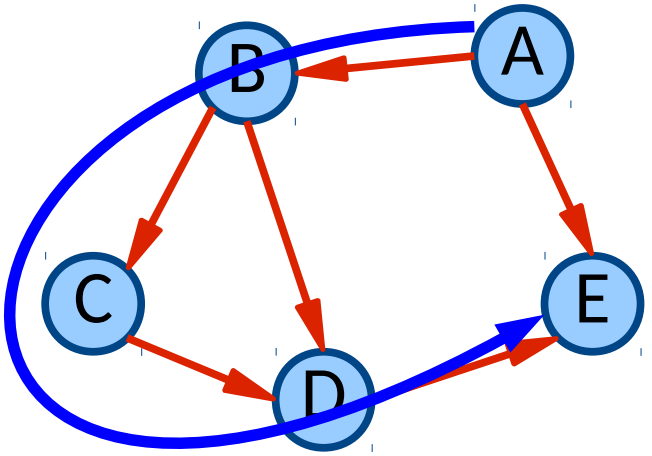
- What is a *graph* G?
  - A set N of nodes
  - A set E of edges
- When edges are directed from one node to another, the graph is a *directed graph*
- A *path* is a list of pairwise connected nodes



Graphs

- What is a *graph* G?
  - A set N of nodes
  - A set E of edges
- When edges are directed from one node to another, the graph is a *directed graph*
- A *path* is a list of pairwise connected nodes

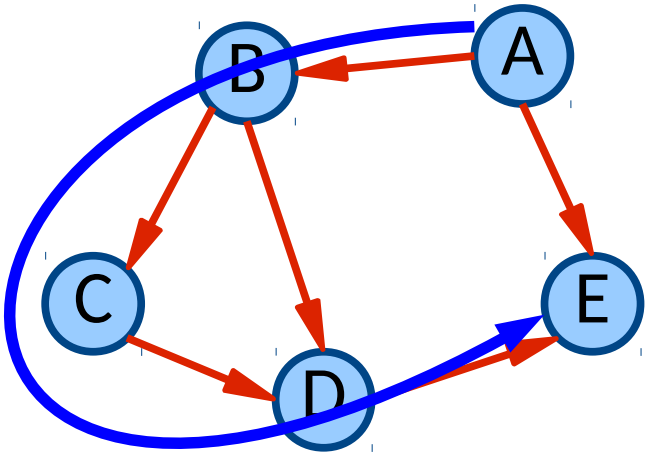
ABCDE



Graphs

- What is a *graph* G?
  - A set N of nodes
  - A set E of edges
- When edges are directed from one node to another, the graph is a *directed graph*
- A *path* is a list of pairwise connected nodes

ABCDE
ABDE
AE
BCDE
BD



Control Flow Graphs (CFGs)

- Programs can be modeled as graphs
  - Used extensively in *compilers*

Control Flow Graphs (CFGs)

- Programs can be modeled as graphs
  - Used extensively in compilers
  - Also used in testing!

Control Flow Graphs (CFGs)

- Programs can be modeled as graphs
  - Used extensively in compilers
  - Also used in testing!
- Control Flow Graphs
  - Nodes comprise the code of a program

Control Flow Graphs (CFGs)

- Programs can be modeled as graphs
  - Used extensively in compilers
  - Also used in testing!
- Control Flow Graphs
  - Nodes comprise the code of a program
  - Edges show the paths that an execution may take through a program

Control Flow Graphs

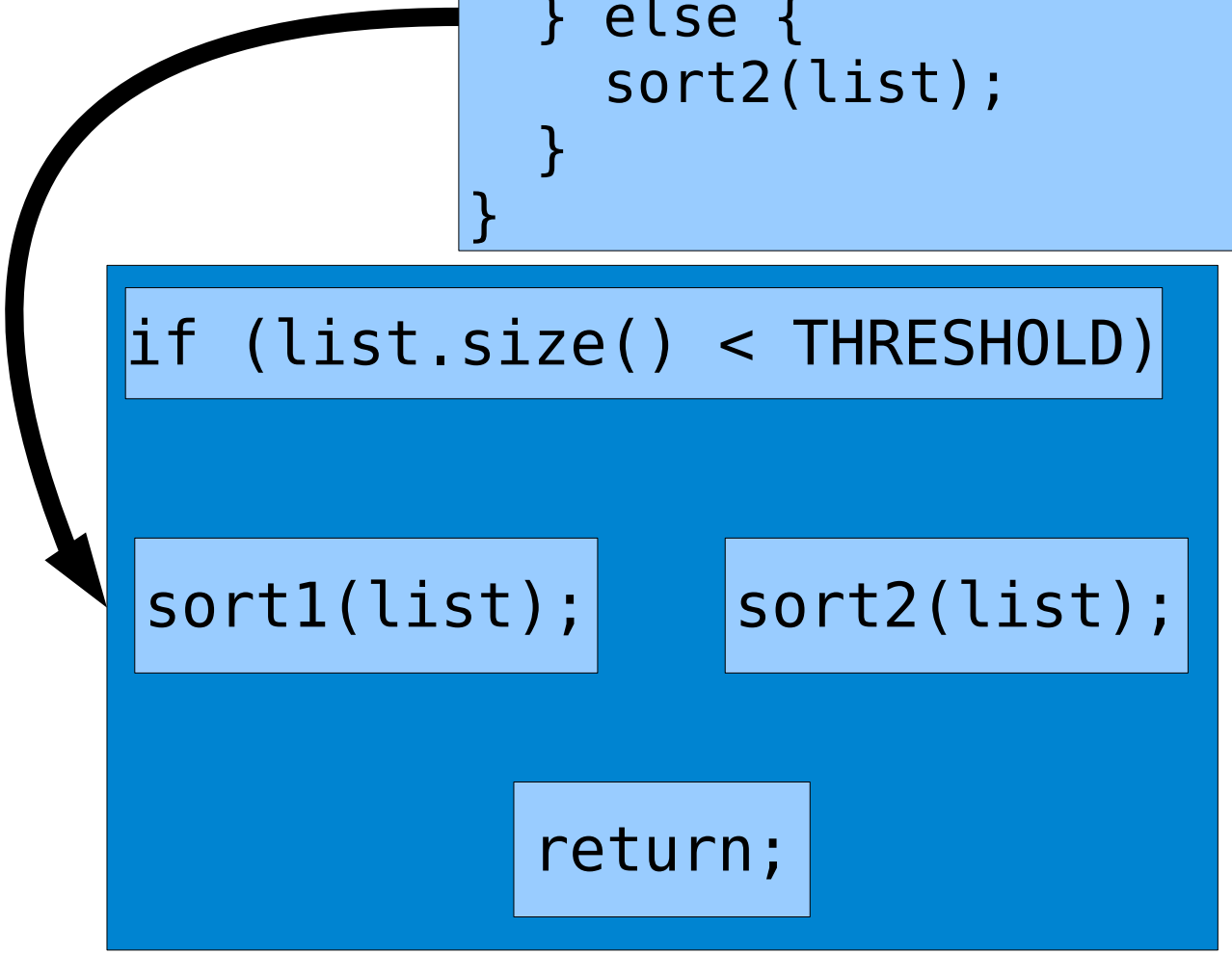
Example:

```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```

Control Flow Graphs

Example:

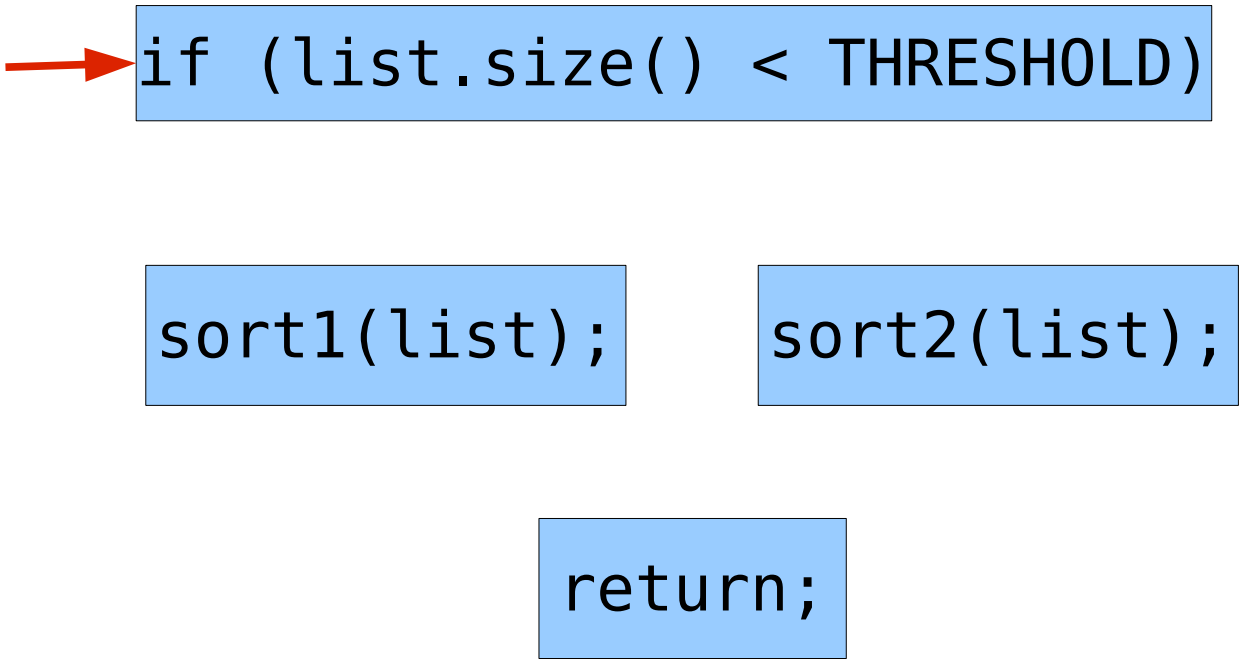
```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```



Control Flow Graphs

Example:

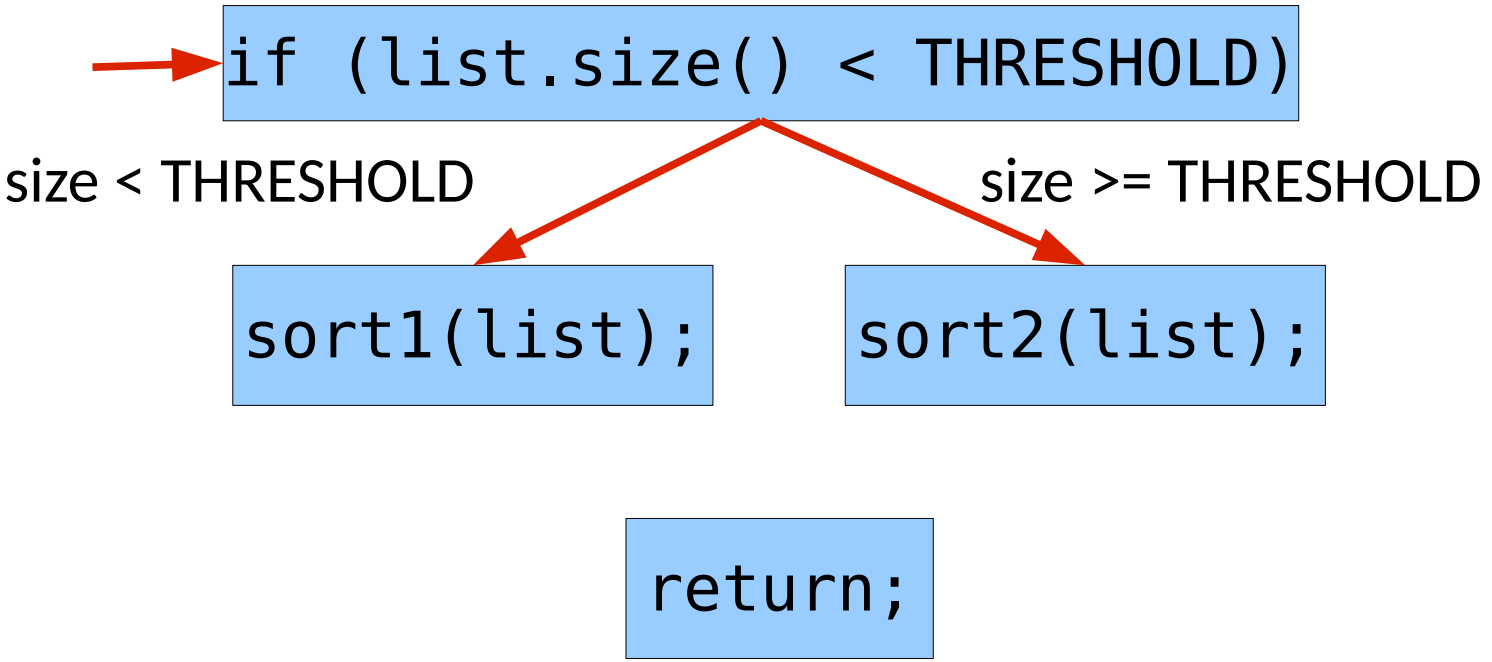
```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```



Control Flow Graphs

Example:

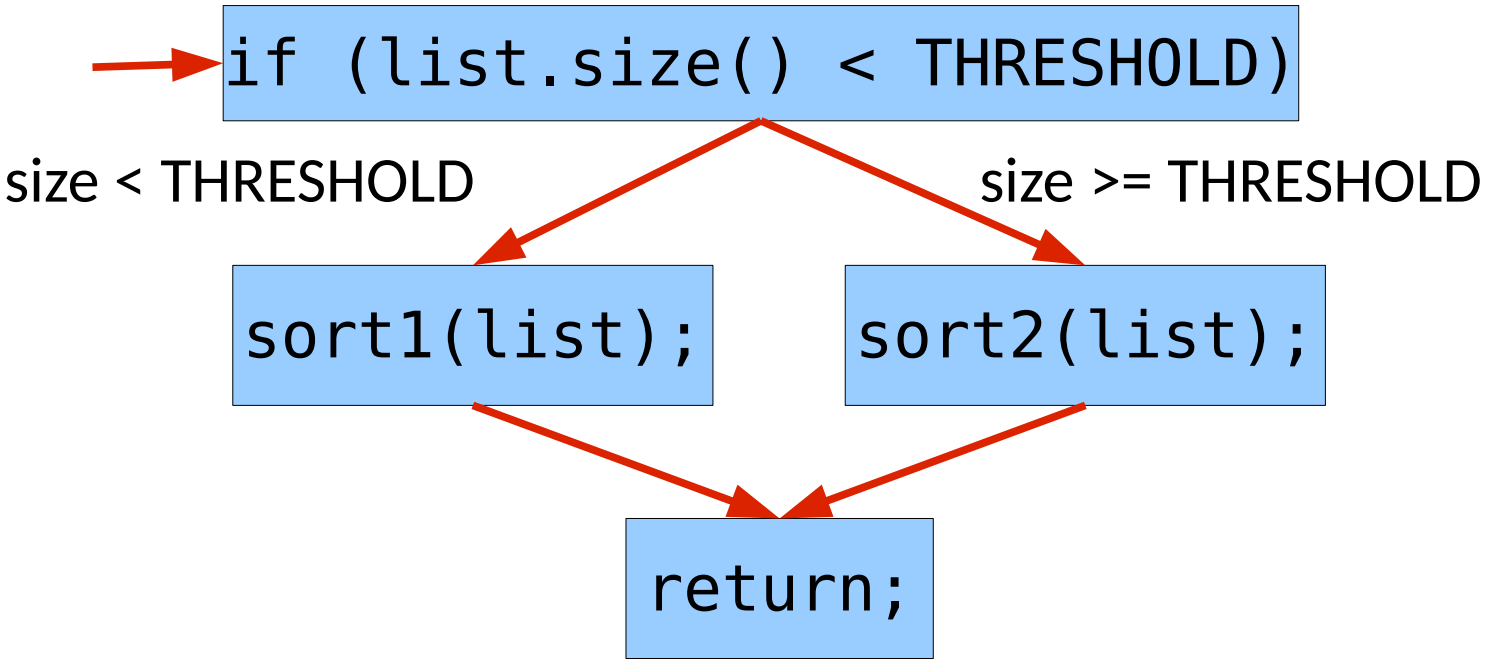
```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```



Control Flow Graphs

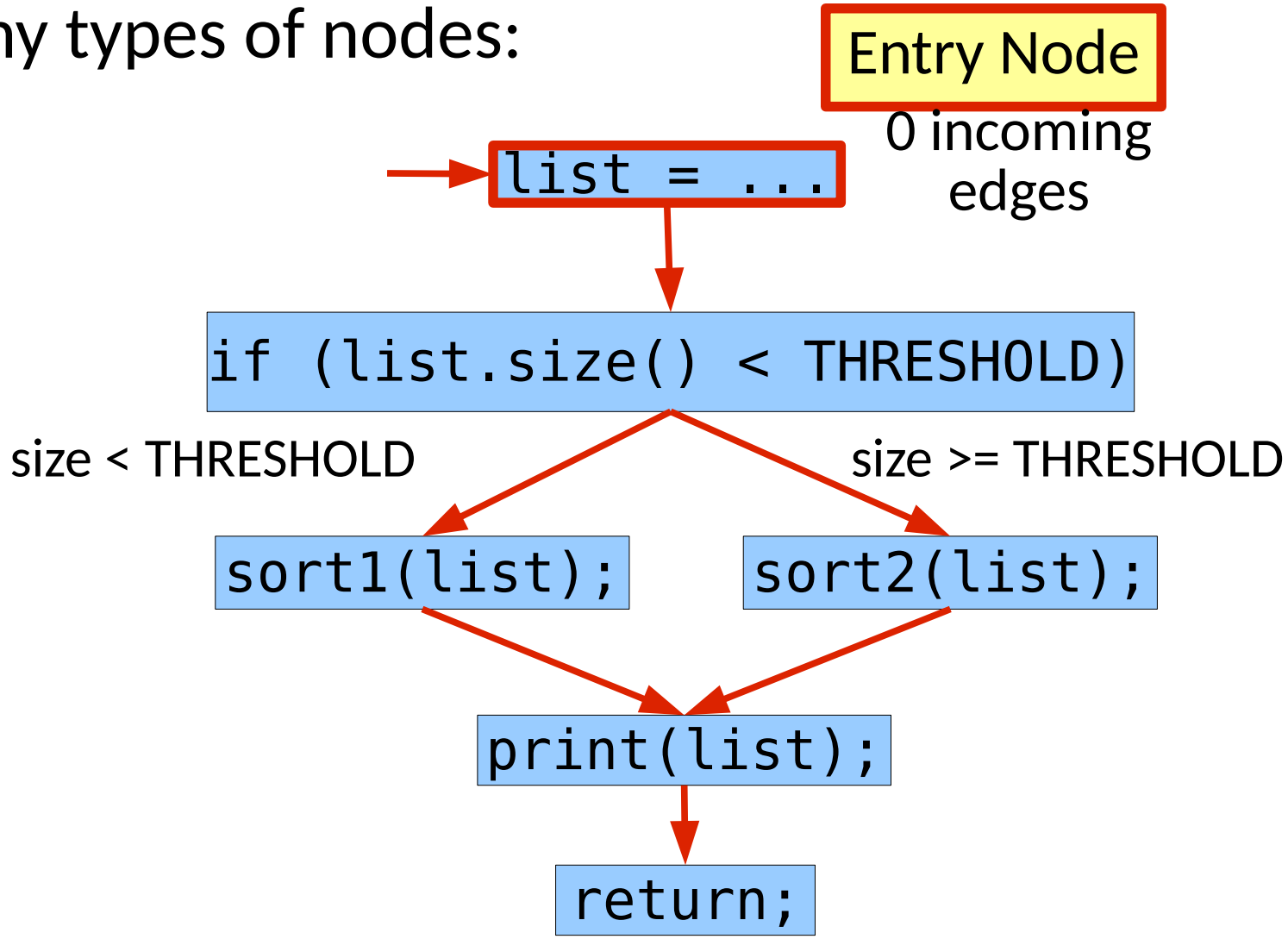
Example:

```
void sortEfficiently(List list) {
  if (list.size() < THRESHOLD) {
    sort1(list);
  } else {
    sort2(list);
  }
}
```



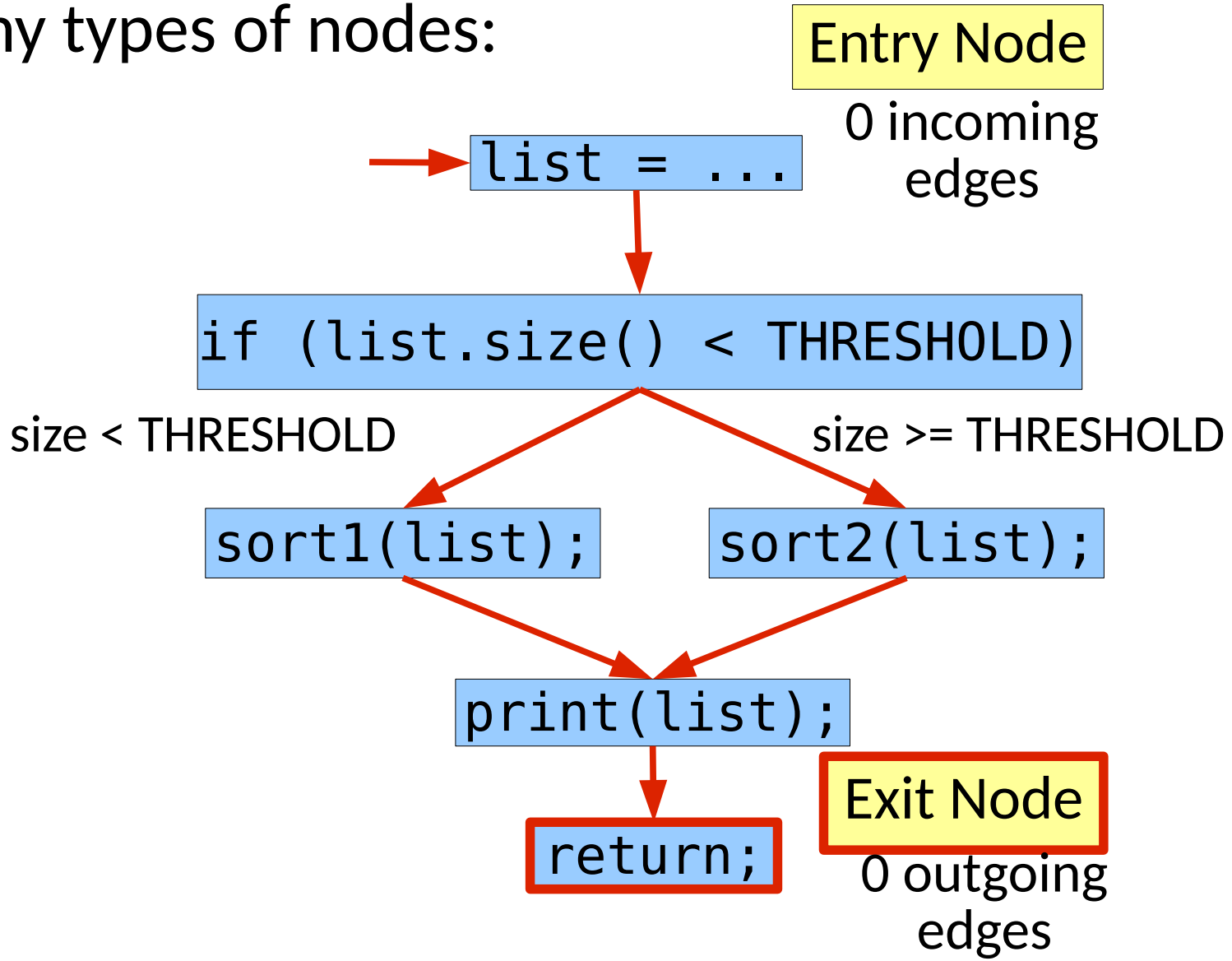
Control Flow Graphs

- Many types of nodes:



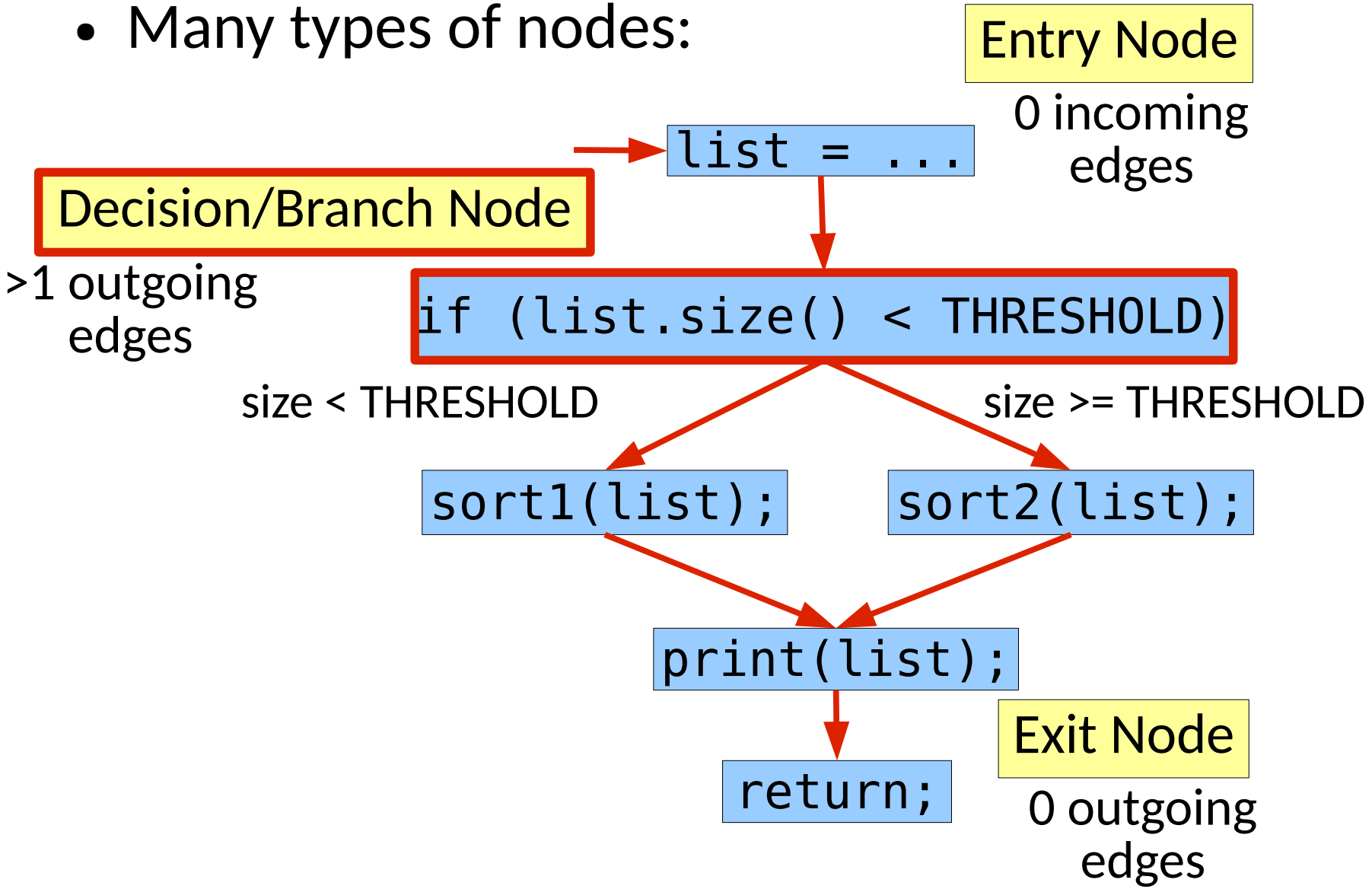
Control Flow Graphs

- Many types of nodes:



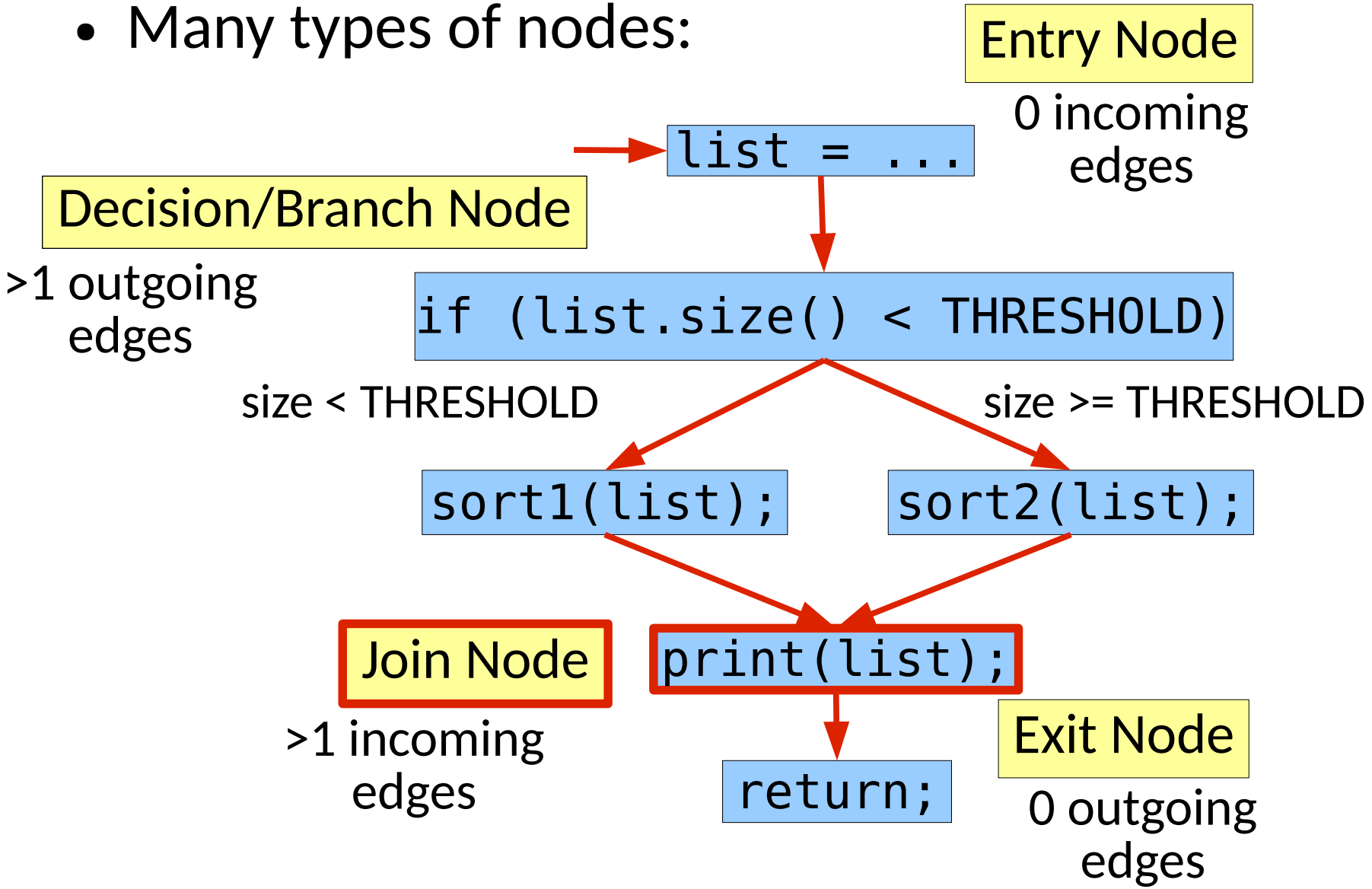
Control Flow Graphs

- Many types of nodes:



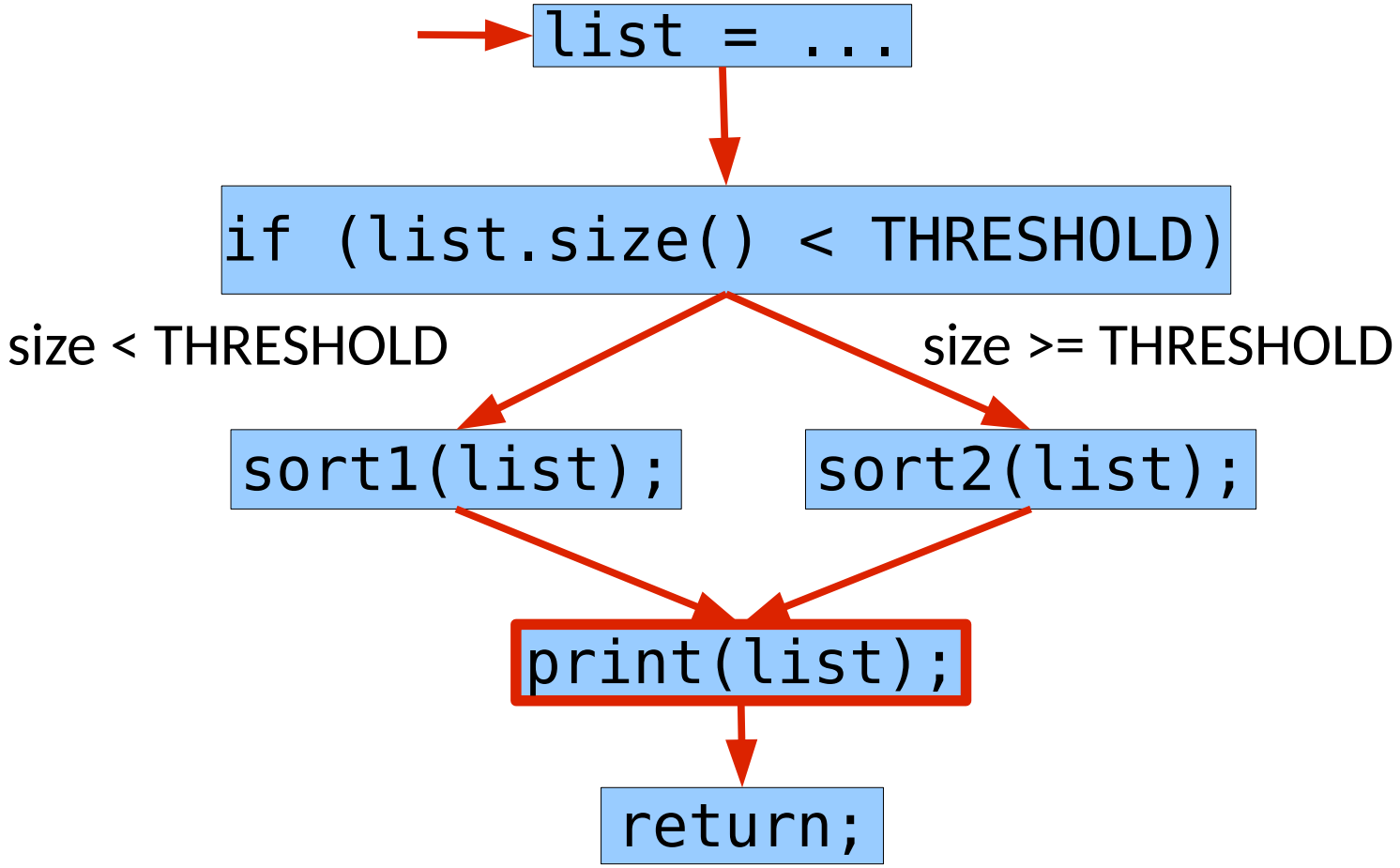
Control Flow Graphs

- Many types of nodes:



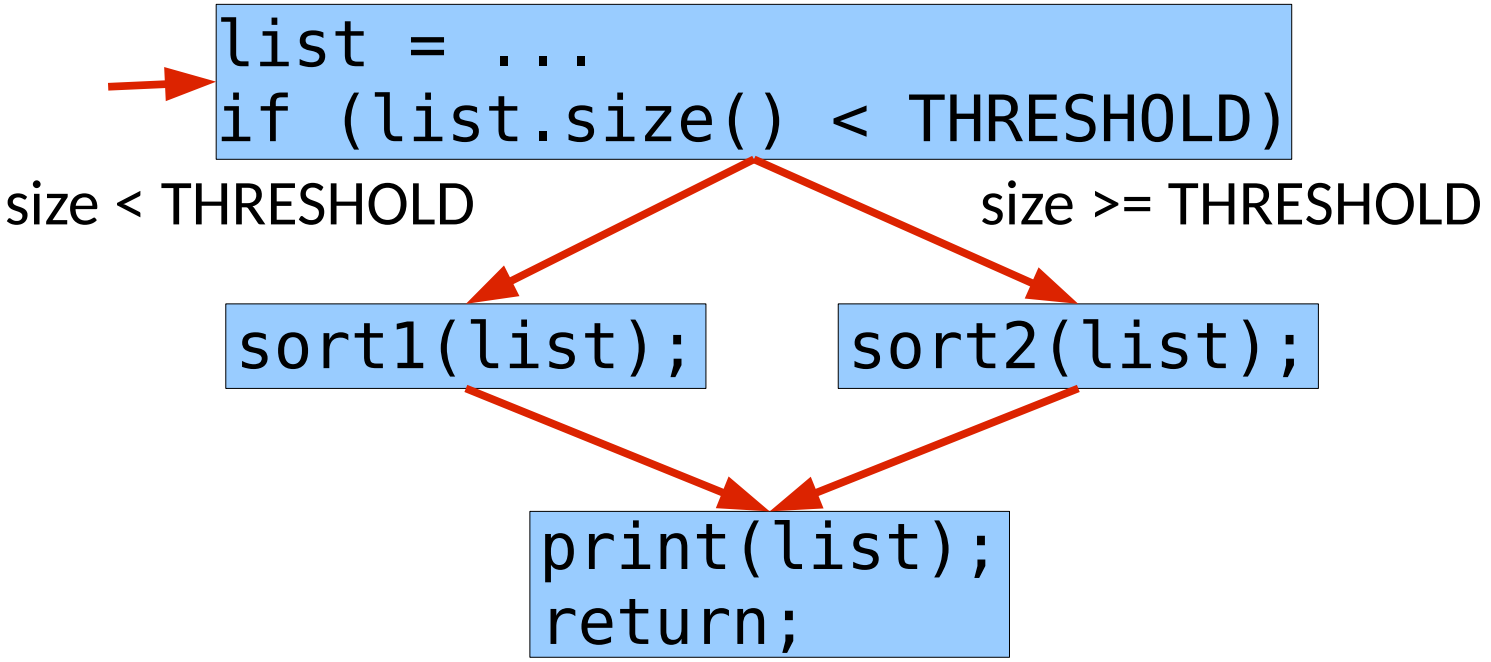
Control Flow Graphs

- Straight-line sequences of code are grouped into basic blocks:



Control Flow Graphs

- Straight-line sequences of code are grouped into basic blocks:



Control Flow Graphs

- Many patterns arise from common constructs

Control Flow Graphs

- Many patterns arise from common constructs
- What is the CFG for this program?

```
q = 0;
r = x;
while r >= y {
  r = r - y;
  q = q + 1;
}
```

Control Flow Graphs

- What is the CFG?

```
for (i = 0; i < n; i++) {
   foo(i);
}
```

Control Flow Graphs

- What is the CFG?
  - Don't forget implicit behavior like `default`!

```
switch  (x) {
   case a: foo(x); break;
   case b: bar(x);
   case c: baz(x); break;
}
```

Control Flow Graphs

- What is the CFG?
  - Short circuit operators can lead to subtle behavior!

```
if (x == 0 || y/x > 1) {
    foo(x, y);
}
```

Control Flow Graphs

- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:

```
x = 0;
while (x < y) {
   y = f (x, y);
   if (y == 0) {
      break;
   } else if (y < 0) {
      y = y*2;
      continue;
   }
   x = x + 1;
}
print (y);
```

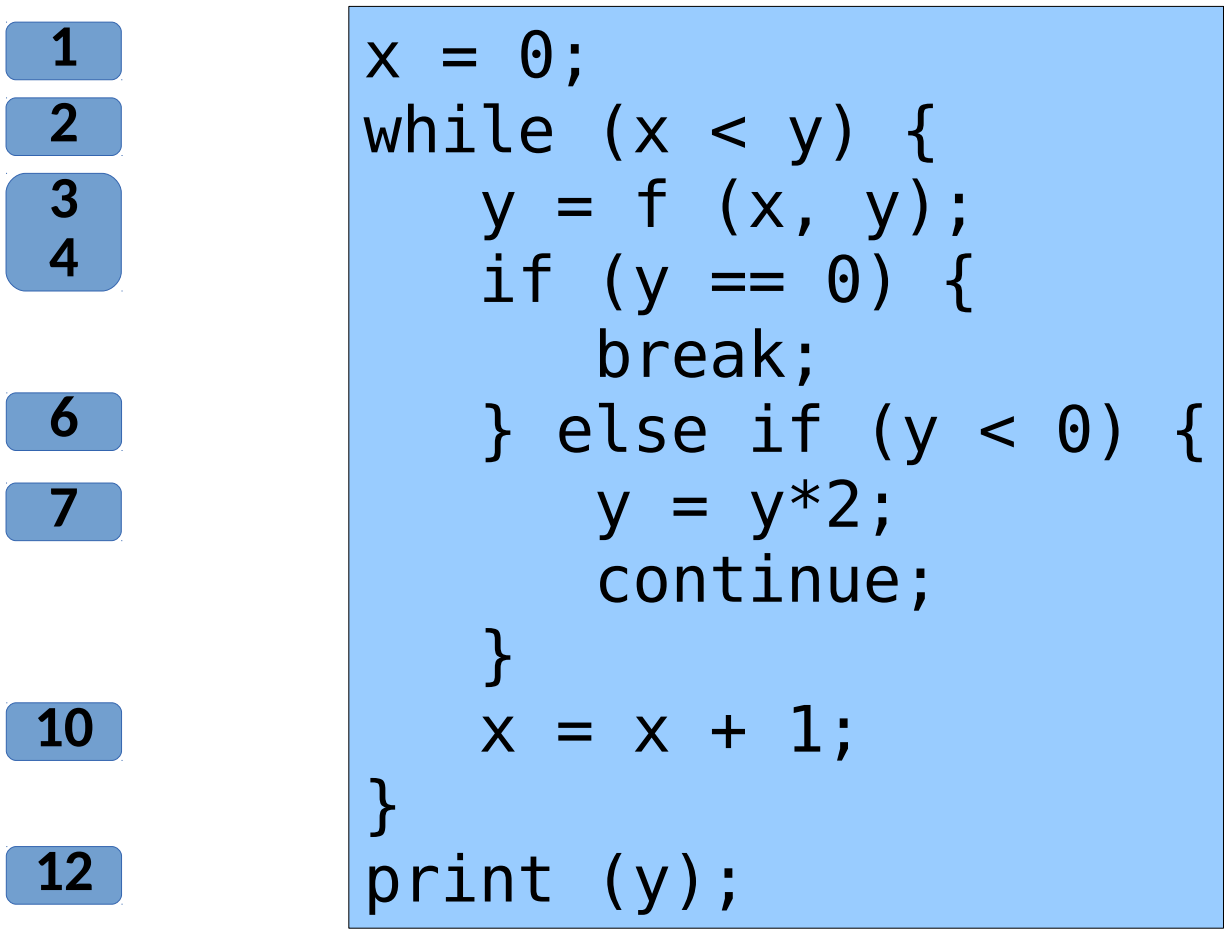
Control Flow Graphs

- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:

```
 1  x = 0;
 2  while (x < y) {
 3      y = f (x, y);
 4      if (y == 0) {
 5          break;
 6      } else if (y < 0) {
 7          y = y*2;
 8          continue;
 9      }
10      x = x + 1;
11  }
12  print (y);
```

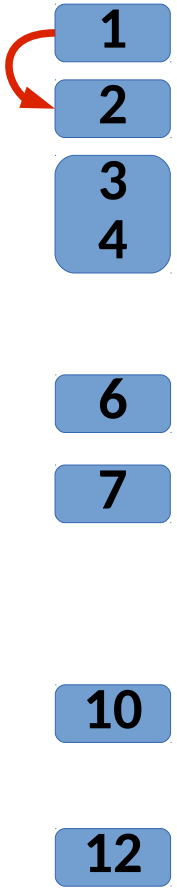
Control Flow Graphs

- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:



Control Flow Graphs

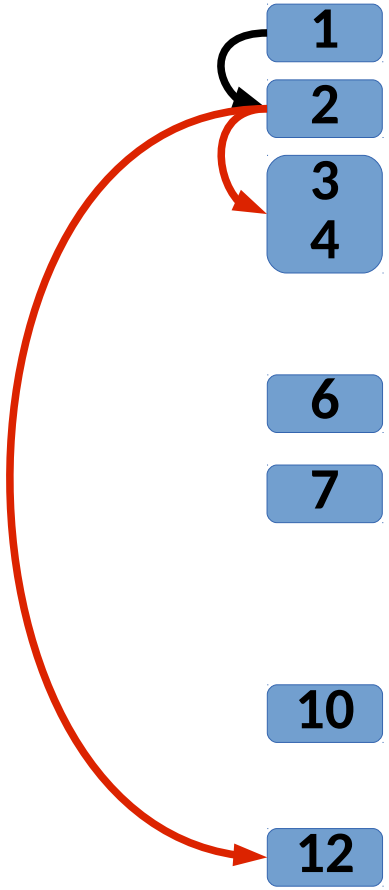
- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:




```
x = 0;
while (x < y) {
    y = f (x, y);
    if (y == 0) {
        break;
    } else if (y < 0) {
        y = y*2;
        continue;
    }
    x = x + 1;
}
print (y);
```

Control Flow Graphs

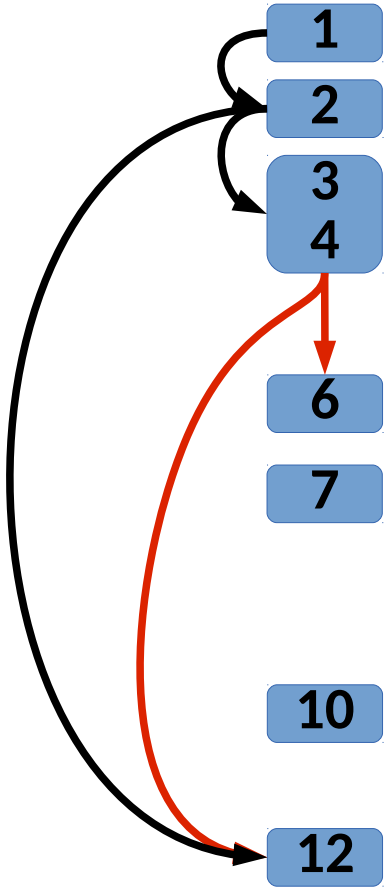
- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:




```
x = 0;
while (x < y) {
    y = f (x, y);
    if (y == 0) {
        break;
    } else if (y < 0) {
        y = y*2;
        continue;
    }
    x = x + 1;
}
print (y);
```

Control Flow Graphs

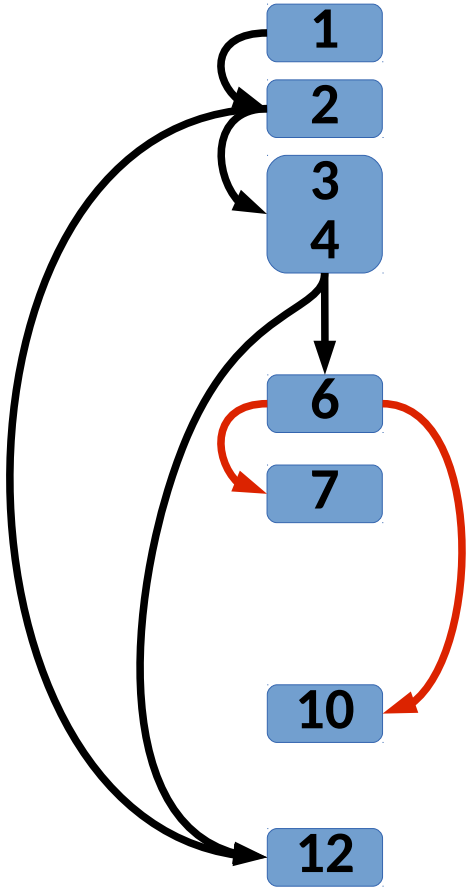
- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:




```
x = 0;
while (x < y) {
    y = f (x, y);
    if (y == 0) {
        break;
    } else if (y < 0) {
        y = y*2;
        continue;
    }
    x = x + 1;
}
print (y);
```

Control Flow Graphs

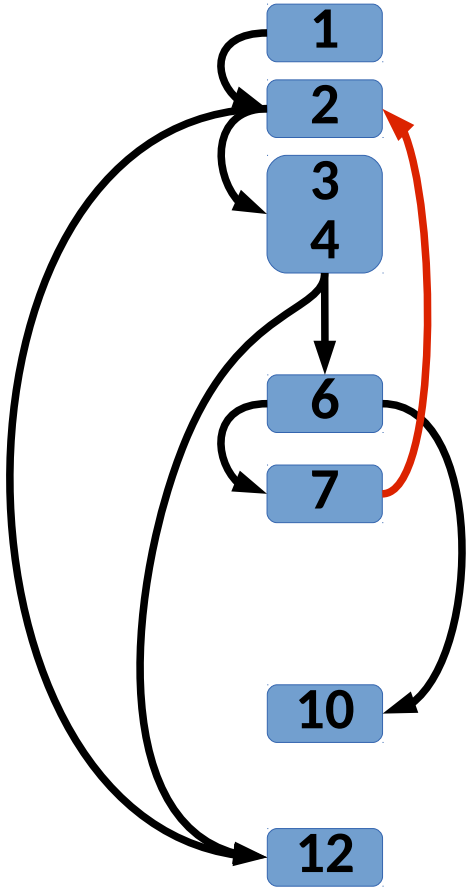
- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:




```
x = 0;
while (x < y) {
    y = f (x, y);
    if (y == 0) {
        break;
    } else if (y < 0) {
        y = y*2;
        continue;
    }
    x = x + 1;
}
print (y);
```

Control Flow Graphs

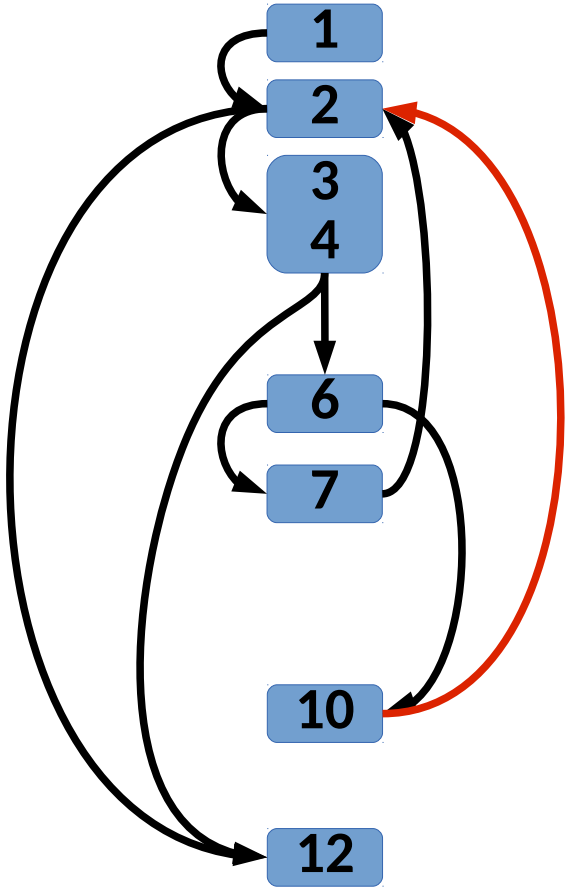
- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:




```
x = 0;
while (x < y) {
    y = f (x, y);
    if (y == 0) {
        break;
    } else if (y < 0) {
        y = y*2;
        continue;
    }
    x = x + 1;
}
print (y);
```

Control Flow Graphs

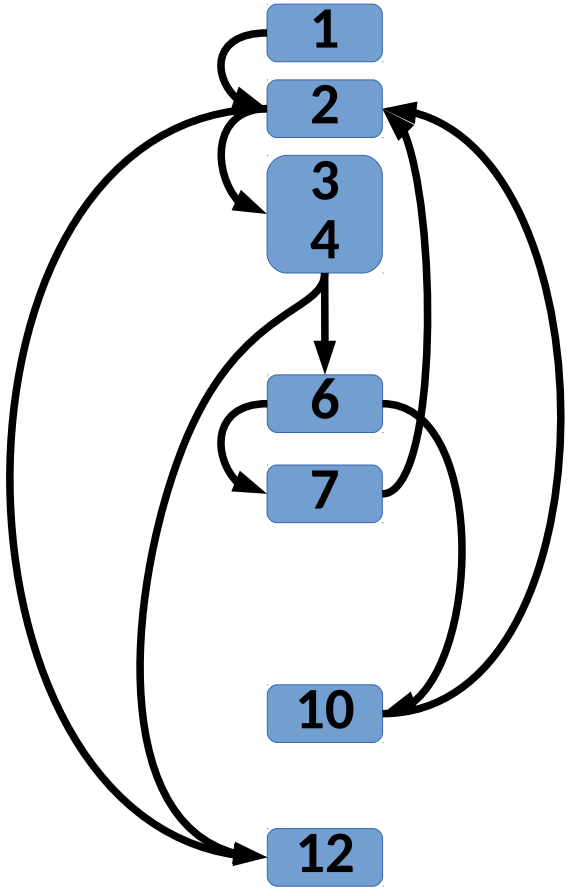
- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:




```
x = 0;
while (x < y) {
    y = f (x, y);
    if (y == 0) {
        break;
    } else if (y < 0) {
        y = y*2;
        continue;
    }
    x = x + 1;
}
print (y);
```

Control Flow Graphs

- What is the CFG?
  - Control flow can become complex!
    From Ammann & Offutt:




```
x = 0;
while (x < y) {
    y = f (x, y);
    if (y == 0) {
        break;
    } else if (y < 0) {
        y = y*2;
        continue;
    }
    x = x + 1;
}
print (y);
```

CFG Coverage

- Statement Coverage → Node Coverage

Try to cover all reachable basic blocks.

CFG Coverage

- Statement Coverage → Node Coverage

Thinking in terms of node coverage can be more efficient. **Why**?

CFG Coverage

- Statement Coverage → Node Coverage
- Branch Coverage → Edge Coverage

Try to cover all reachable paths of length ≤ 1.

CFG Coverage

- Statement Coverage → Node Coverage
- Branch Coverage → Edge Coverage

How do node & edge coverage compare? **Why**?

CFG Coverage

- Statement Coverage → Node Coverage
- Branch Coverage → Edge Coverage

How do node & edge coverage compare? **Why**?

Are these notions of coverage enough? **Why**?

Pragmatic Concerns

- The *goal* is full coverage (of whatever criteria)

Pragmatic Concerns

- The *goal* is full coverage (of whatever criteria)

Is that reasonable in practice?
Why?

Pragmatic Concerns

- The *goal* is full coverage (of whatever criteria)
- We must consider *reachability*

Pragmatic Concerns

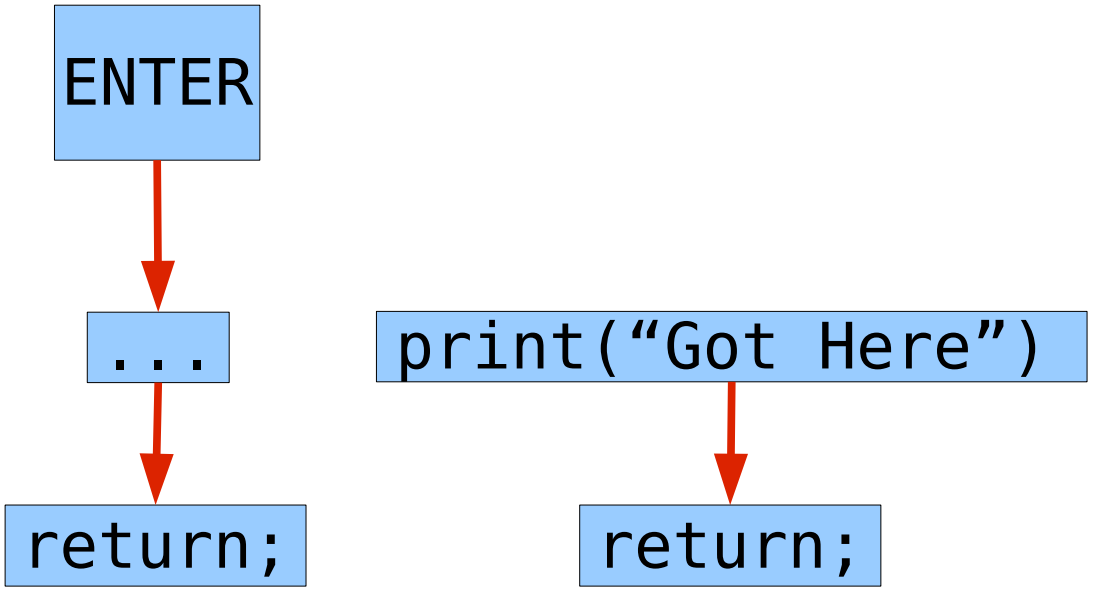
- The *goal* is full coverage (of whatever criteria)
- We must consider *reachability*

Try to cover ***all reachable*** basic blocks.

Try to cover ***all reachable*** paths of length ≤ 1.

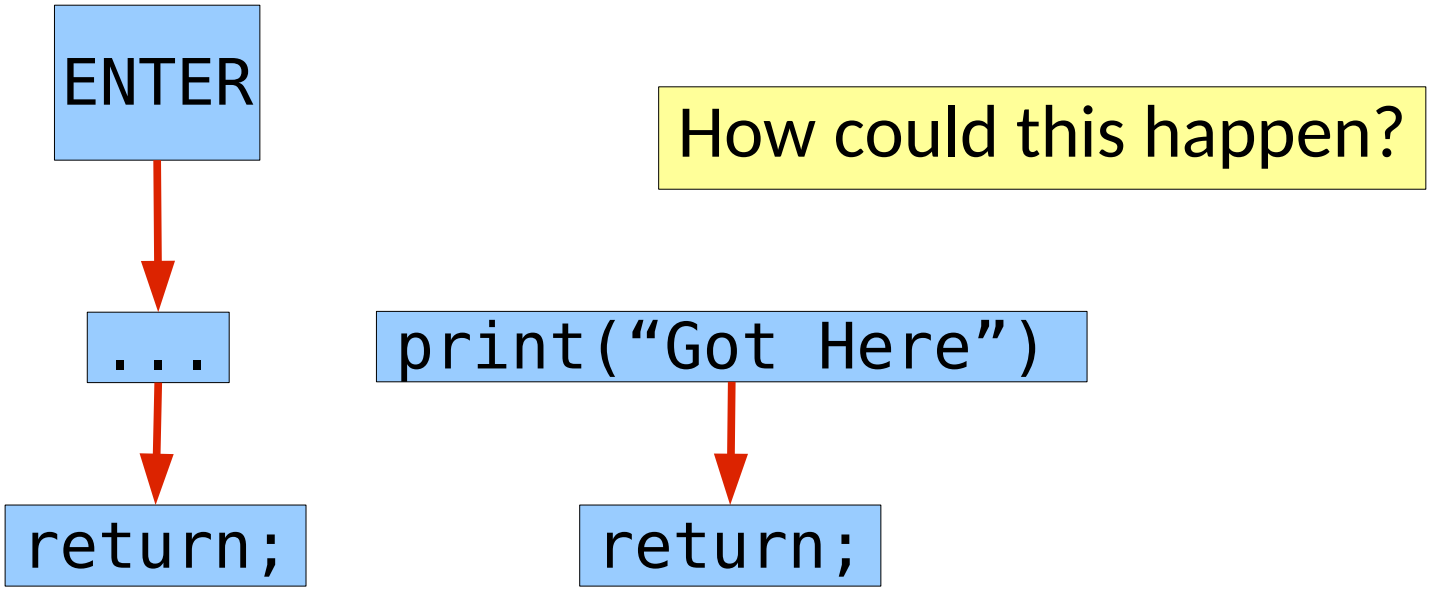
Pragmatic Concerns

- The *goal* is full coverage (of whatever criteria)
- We must consider *reachability*
  - Syntactic Reachability
    - Based on the structure of the code



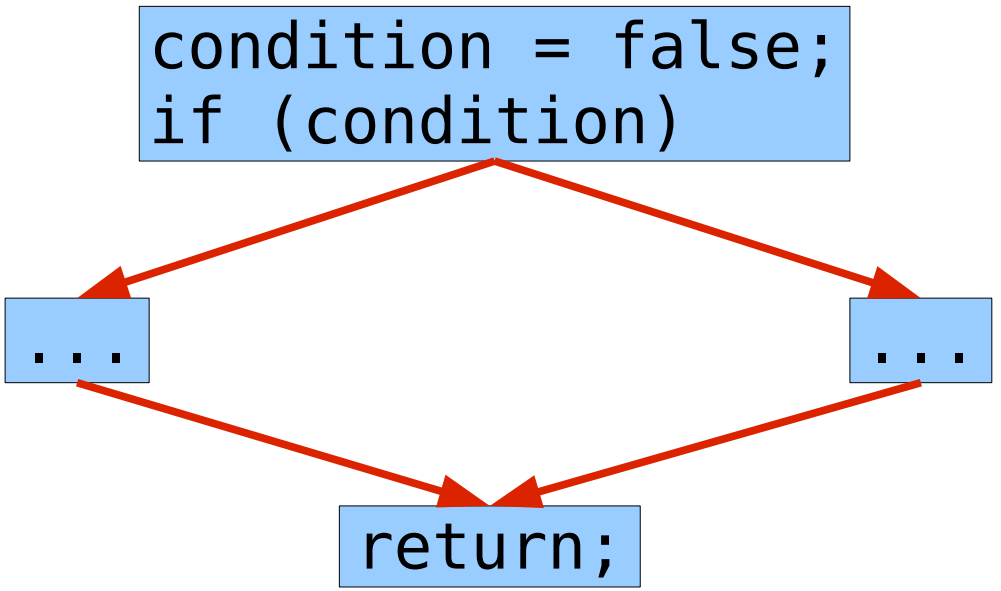
Pragmatic Concerns

- The *goal* is full coverage (of whatever criteria)
- We must consider *reachability*
  - Syntactic Reachability
    - Based on the structure of the code



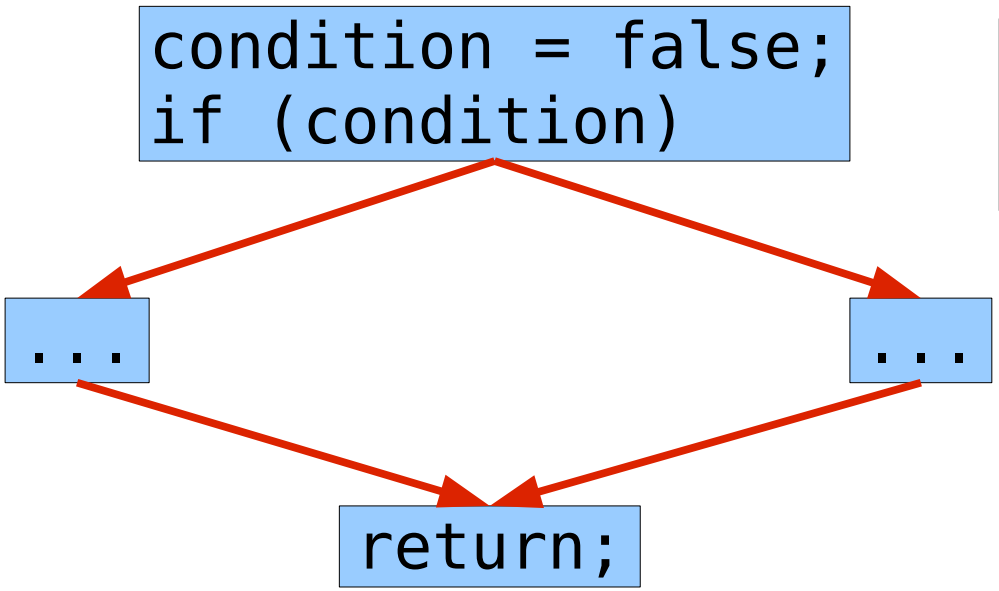
Pragmatic Concerns

- The *goal* is full coverage (of whatever criteria)
- We must consider *reachability*
  - Syntactic Reachability
    - Based on the structure of the code
  - Semantic Reachability
    - Based on the meaning of the code



Pragmatic Concerns

- The *goal* is full coverage (of whatever criteria)
- We must consider *reachability*
  - Syntactic Reachability
    - Based on the structure of the code
  - Semantic Reachability
    - Based on the meaning of the code



This can be ***undecidable***!

Pragmatic Concerns

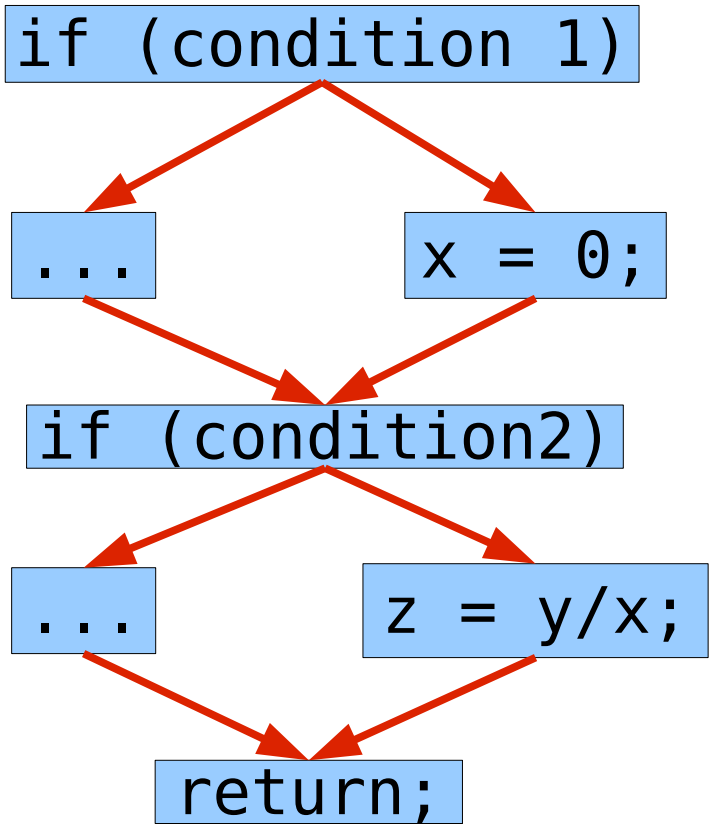
- The *goal* is full coverage (of whatever criteria)
- We must consider *reachability*
  - Syntactic Reachability
    - Based on the structure of the code
  - Semantic Reachability
    - Based on the meaning of the code
- So what do you do in practice?
  - No, really. What have *you* done in practice?

Pragmatic Concerns

- The *goal* is full coverage (of whatever criteria)
- We must consider *reachability*
  - Syntactic Reachability
    - Based on the structure of the code
  - Semantic Reachability
    - Based on the meaning of the code
- So what do you do in practice?
  - No, really. What have *you* done in practice?
  - *Relative* degrees of coverage matter (40%? 80%?)

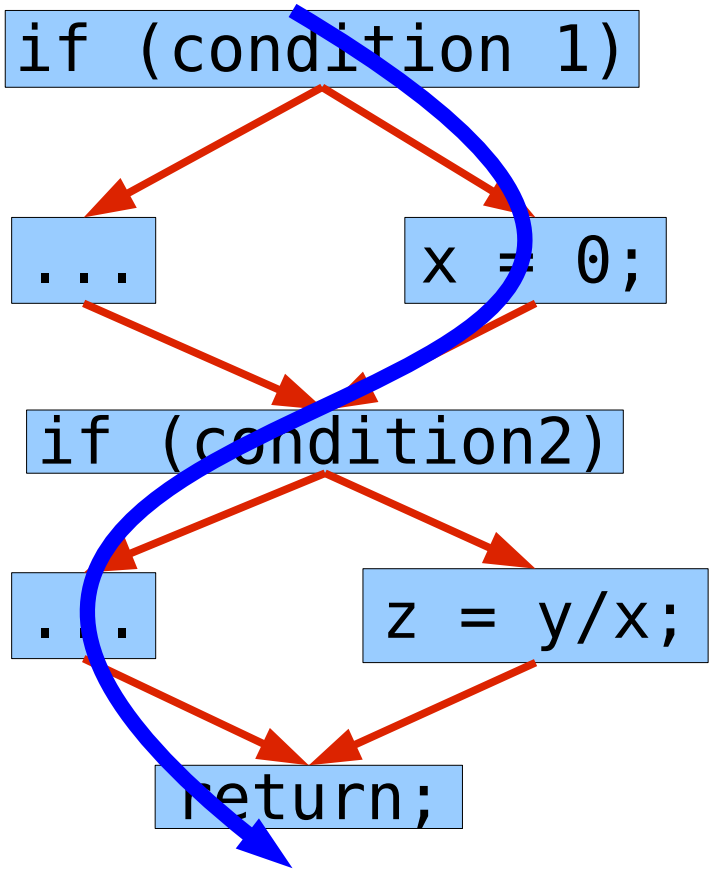
CFG Coverage

- The path taken by each test can matter



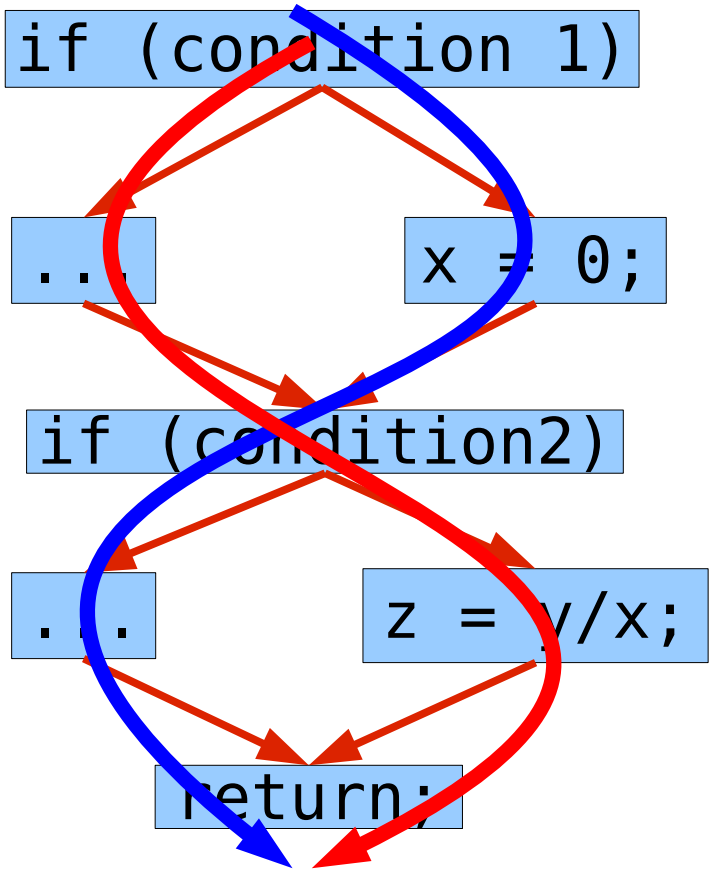
CFG Coverage

- The path taken by each test can matter



CFG Coverage

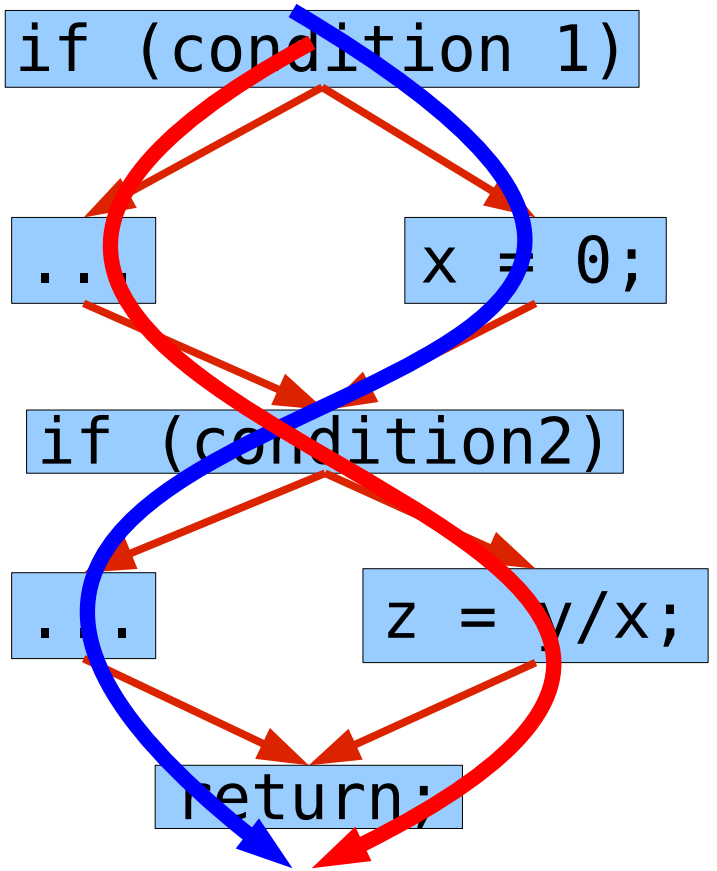
- The path taken by each test can matter



CFG Coverage

- The path taken by each test can matter

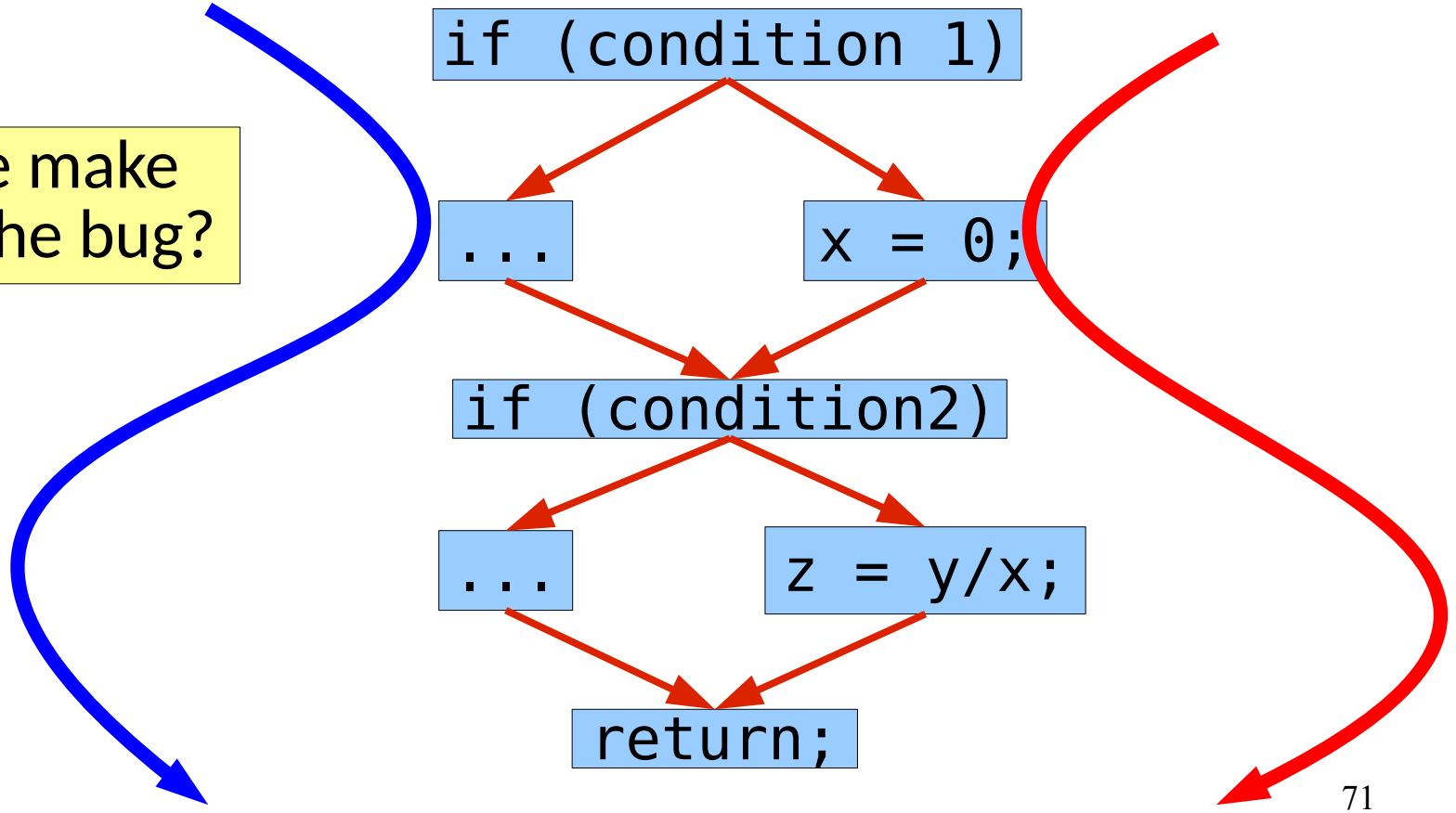
Full edge coverage
& no bugs found



CFG Coverage

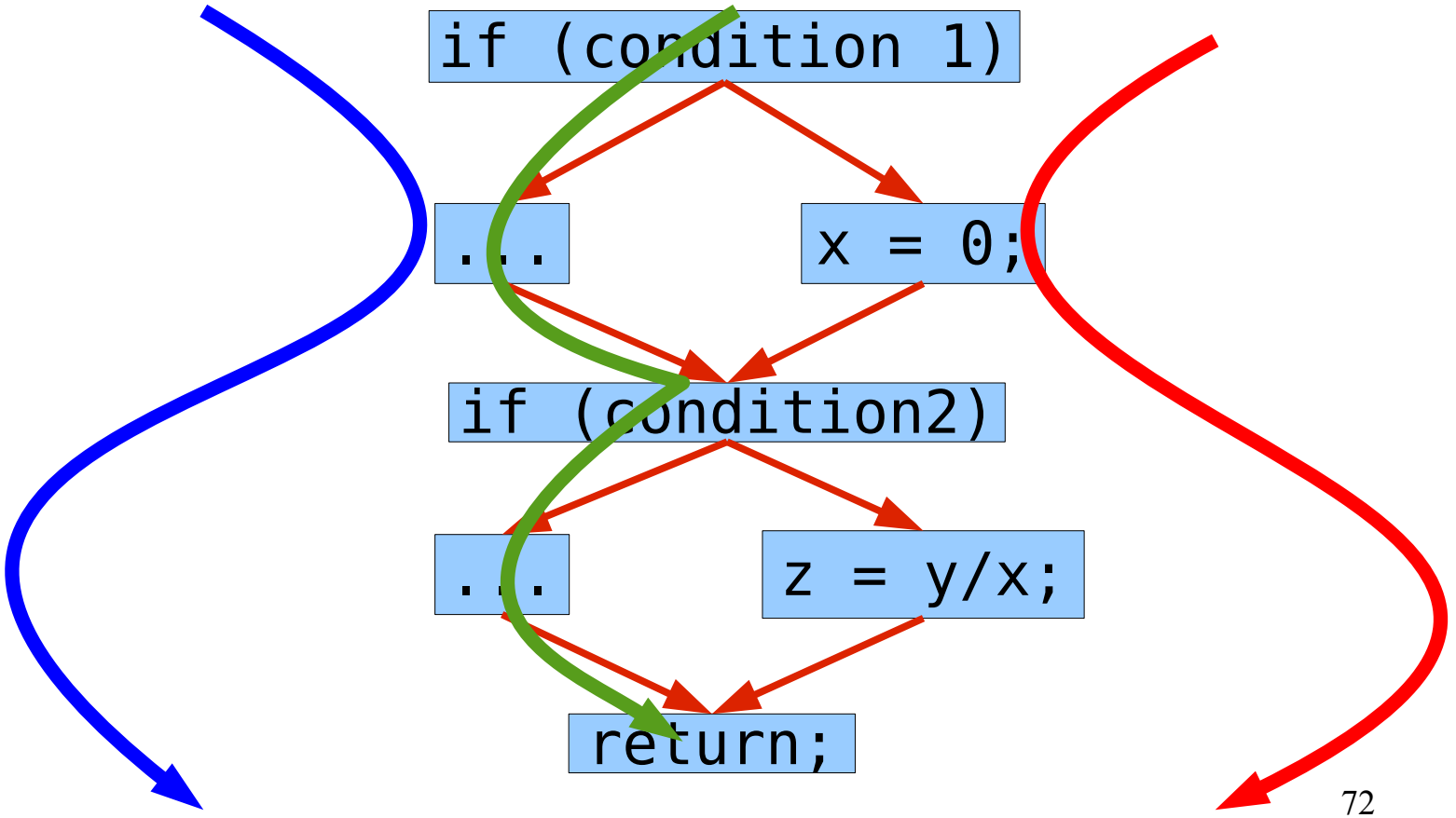
- The path taken by each test can matter

How can we make sure to find the bug?



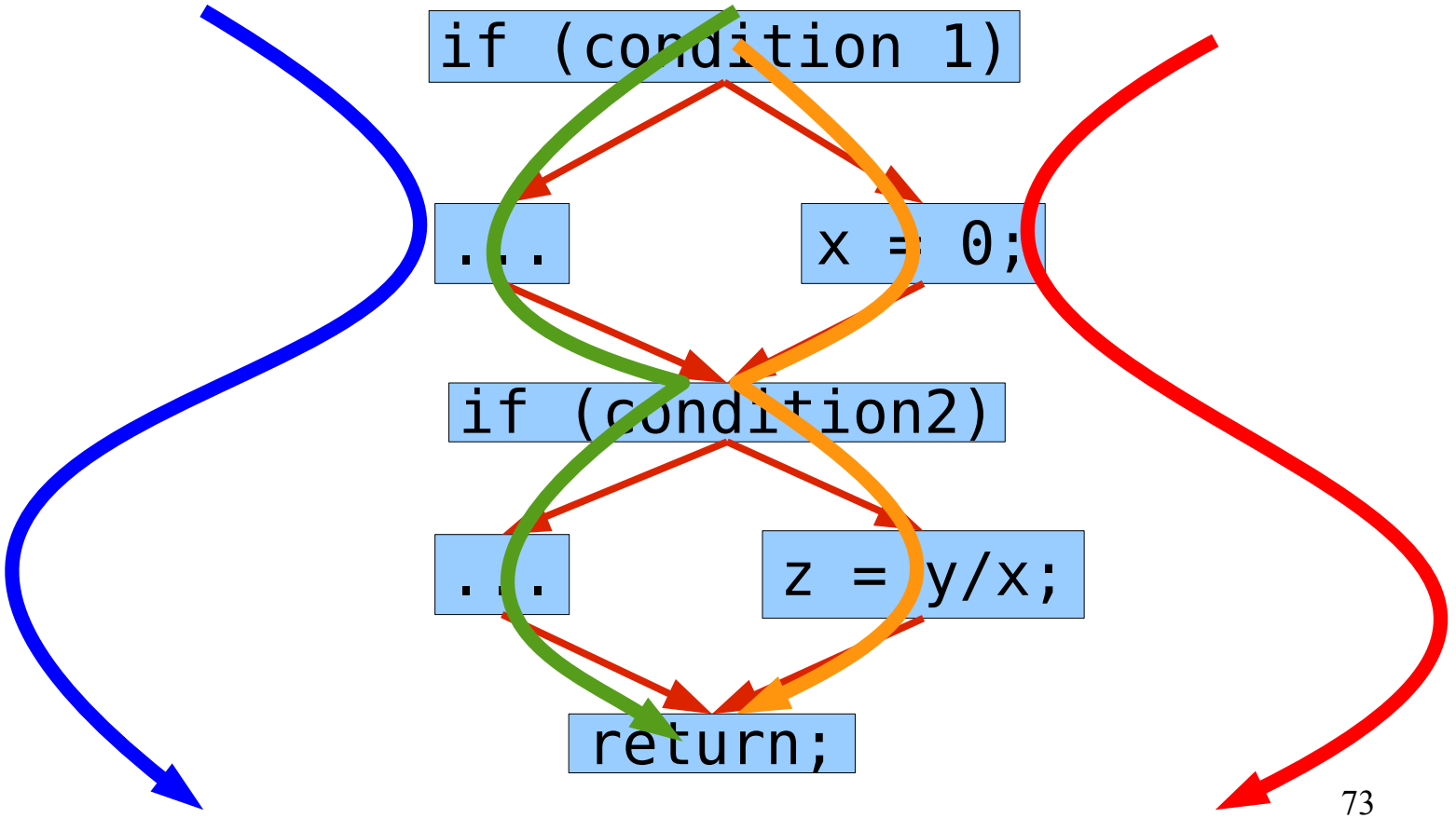
CFG Coverage

- The path taken by each test can matter



CFG Coverage

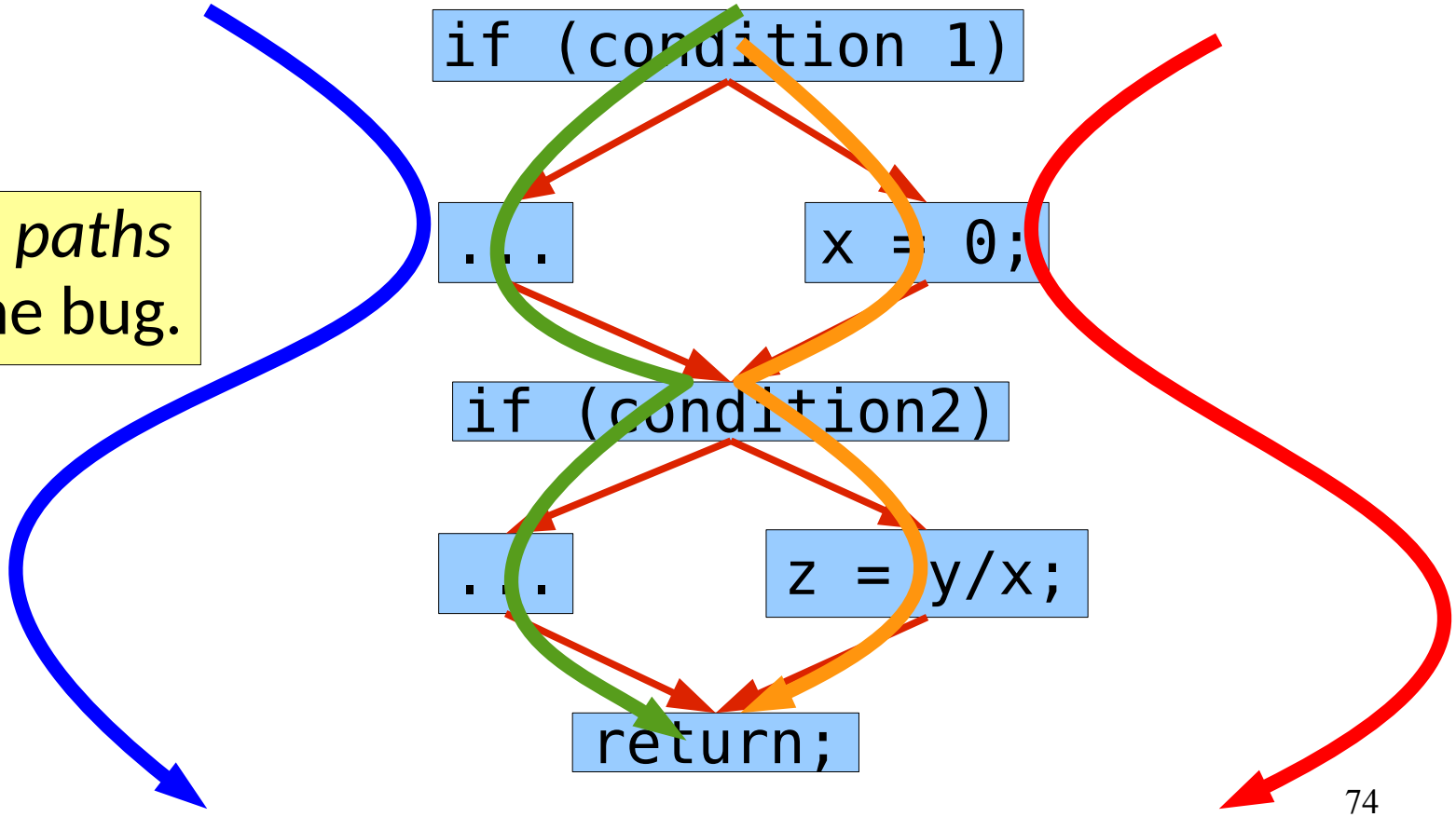
- The path taken by each test can matter



CFG Coverage

- The path taken by each test can matter

Testing *all paths* exposes the bug.



Path Coverage

- Complete Path Coverage
  - Test all paths through the graph

Path Coverage

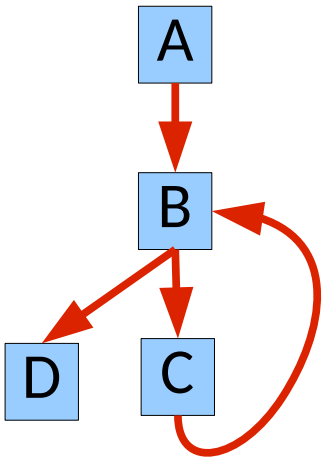
- Complete Path Coverage
  - Test all paths through the graph

Is this reasonable?
Why?

Path Coverage

- Complete Path Coverage
  - Test all paths through the graph

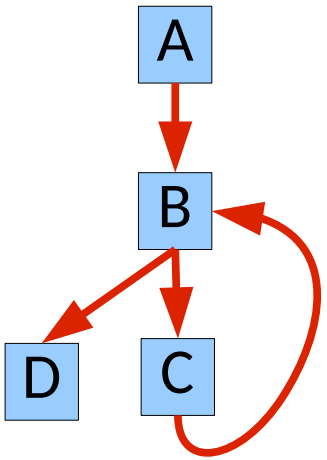
Is this reasonable?
Why?



Path Coverage

- Complete Path Coverage
  - Test all paths through the graph

Is this reasonable?
Why?

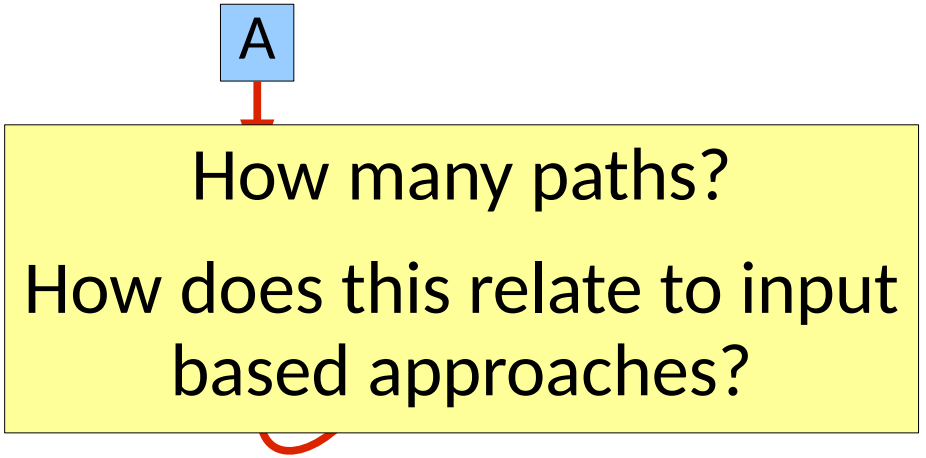


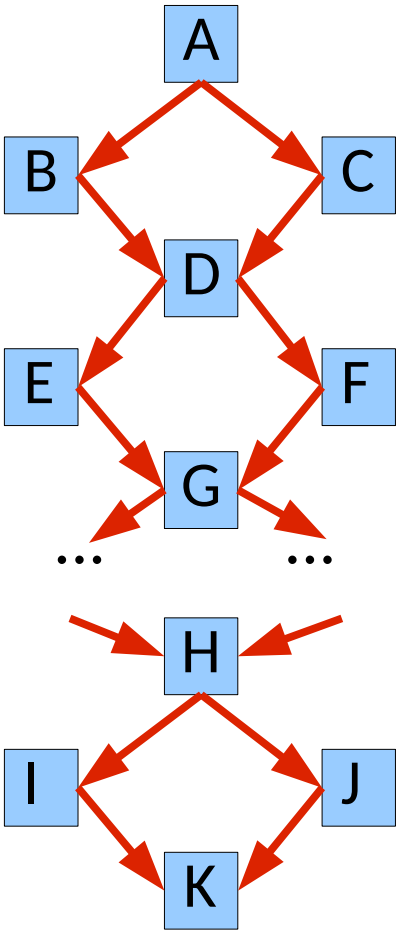
Infeasibility

Path Coverage

- Complete Path Coverage
  - Test all paths through the graph

Is this reasonable?
Why?

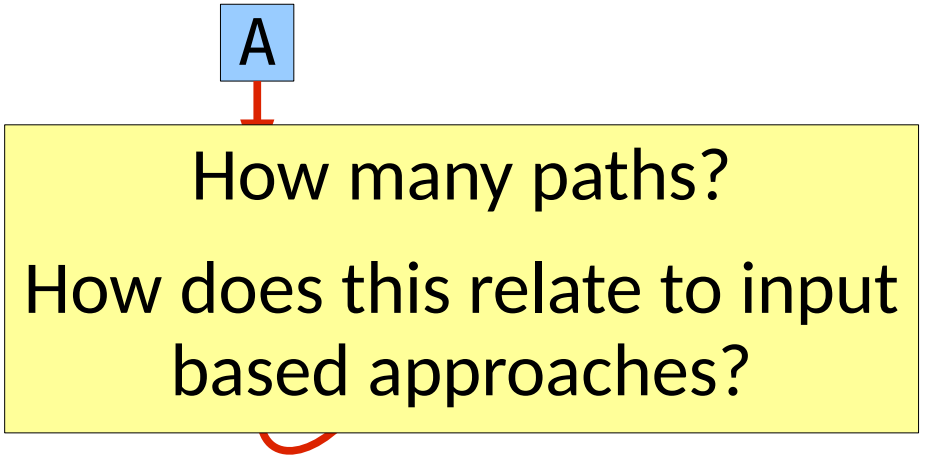


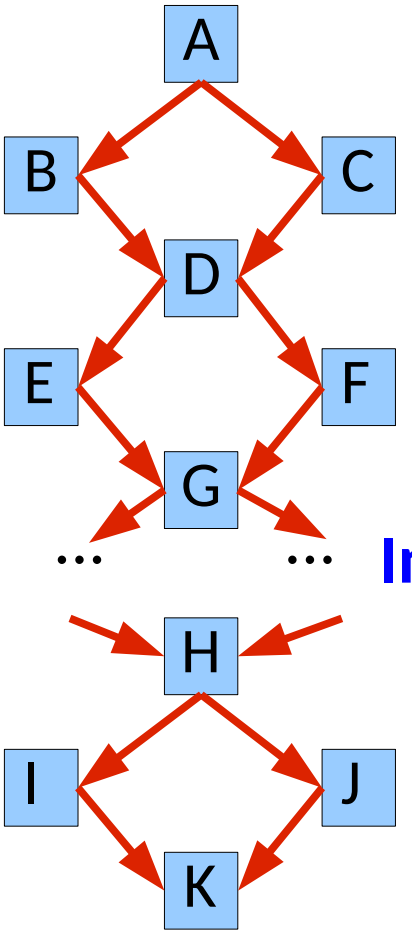


Path Coverage

- Complete Path Coverage
  - Test all paths through the graph

Is this reasonable?
Why?





Intractability

Compromises?

- What could we do instead?

  (How did we handle the input based approaches?)

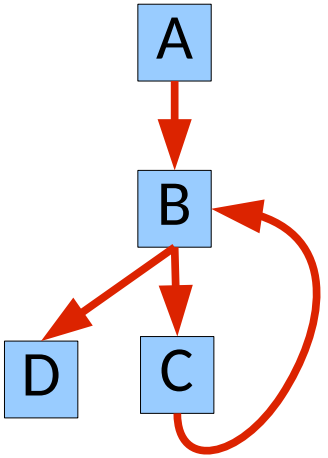
Compromises?

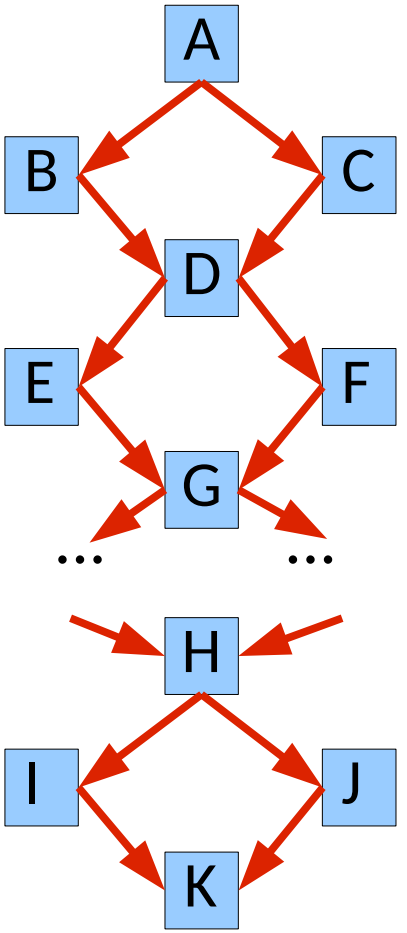
- Edge Pair Coverage
  - Each path of length <= 2 is tested.
- Specified Path Coverage
  - Given a number k, test k paths

Compromises?

- Edge Pair Coverage
  - Each path of length <= 2 is tested.
- Specified Path Coverage
  - Given a number k, test k paths

What do these look like?



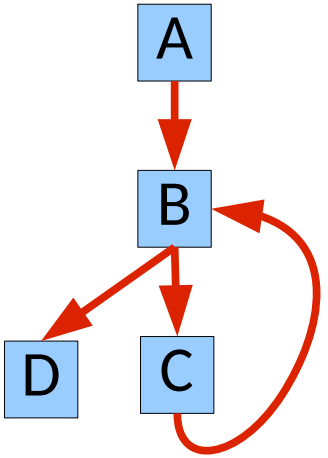


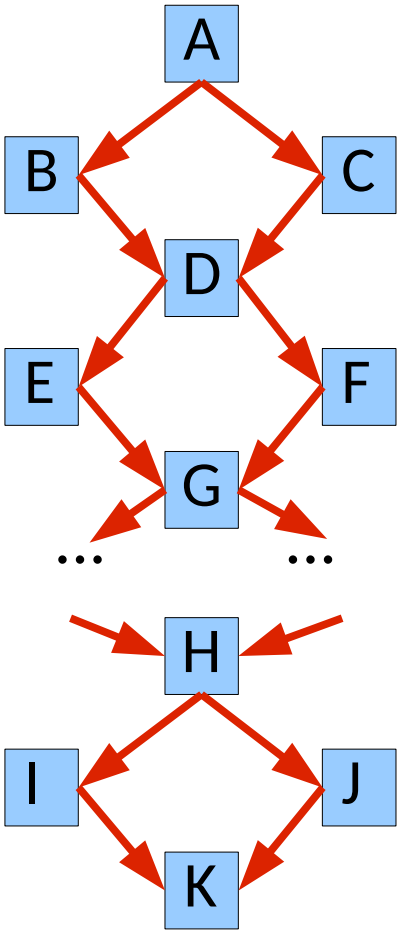
Compromises?

- Edge Pair Coverage
  - Each path of length <= 2 is tested.
- Specified Path Coverage
  - Given a number k, test k paths

What do these look like?

Are they good?





Coping With Loops

- What criteria do *you* use when testing loops?

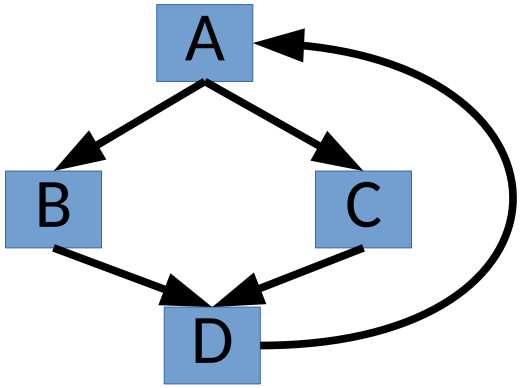
Coping With Loops

- What criteria do *you* use when testing loops?
- Simple Paths
  - A path between nodes is simple if no node appears more than once. (Except maybe the first and last)
  - Captures the acyclic behaviors of a program

Coping With Loops

- What criteria do *you* use when testing loops?
- Simple Paths
  - A path between nodes is simple if no node appears more than once. (Except maybe the first and last)
  - Captures the acyclic behaviors of a program

How many may there be?



Coping With Loops

- What criteria do *you* use when testing loops?
- Simple Paths
  - A path between nodes is simple if no node appears more than once. (Except maybe the first and last)
  - Captures the acyclic behaviors of a program
- Prime Paths
  - A simple path that is not a subpath of any other simple path

Coping With Loops

- What criteria do *you* use when testing loops?
- Simple Paths
  - A path between nodes is simple if no node appears more than once. (Except maybe the first and last)
  - Captures the acyclic behaviors of a program
- Prime Paths
  - A simple path that is not a subpath of any other simple path

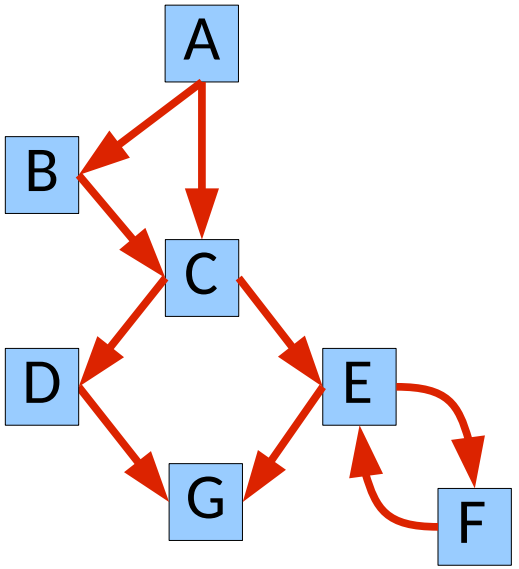
What does this provide?
What do they look like?

Coping With Loops

- Prime Path Coverage
  - Cover all prime paths

Coping With Loops

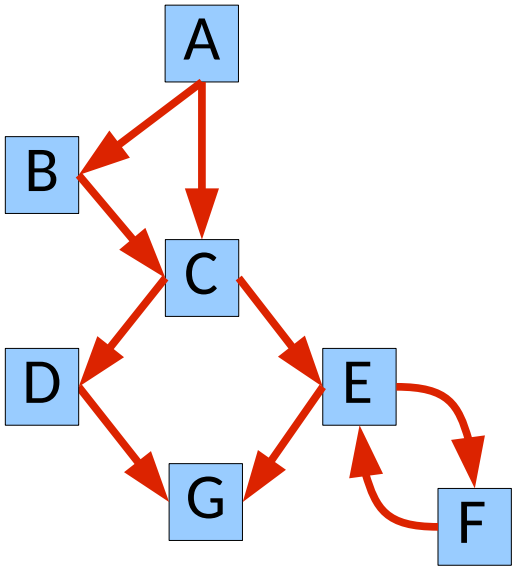
- Prime Path Coverage
  - Cover all prime paths



Example from Ammann & Offutt

Coping With Loops

- Prime Path Coverage
  - Cover all prime paths

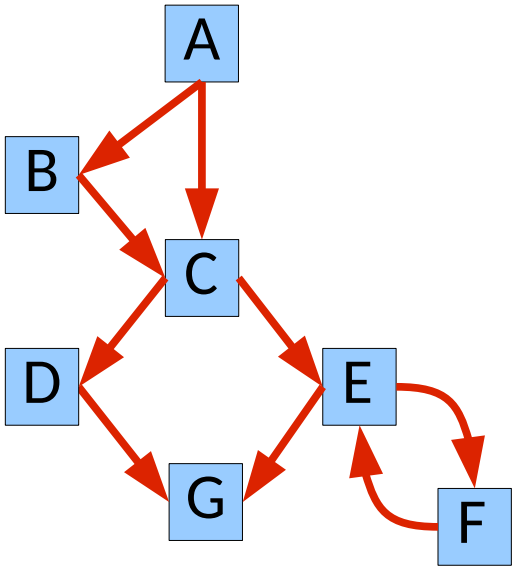


What are the prime paths?

Example from Ammann & Offutt

Coping With Loops

- Prime Path Coverage
  - Cover all prime paths



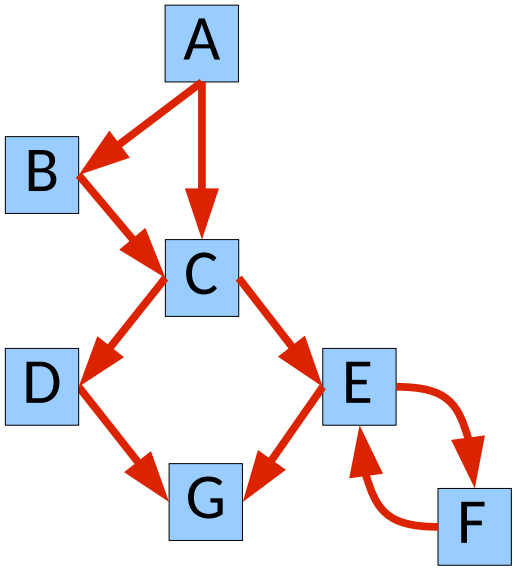
What are the prime paths?

How many simple paths?

Example from Ammann & Offutt

Coping With Loops

- Prime Path Coverage
  - Cover all prime paths



What are the prime paths?

How many simple paths?

Can you intuitively explain what prime paths capture?

Example from Ammann & Offutt

Coping With Loops

- Prime Path Coverage
  - Cover all prime paths

Are these tests good or bad?

Coping With Loops

- Prime Path Coverage
  - Cover all prime paths

Are these tests good or bad?

Do they address all of the problems with path coverage?

Coping With Loops

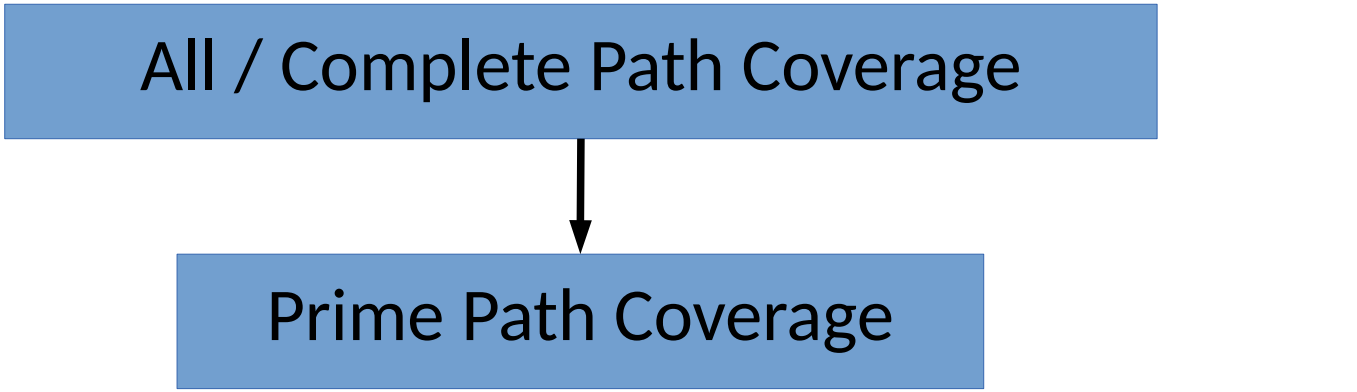
- Prime Path Coverage
  - Cover all prime paths

Are these tests good or bad?

Do they address all of the problems with path coverage?

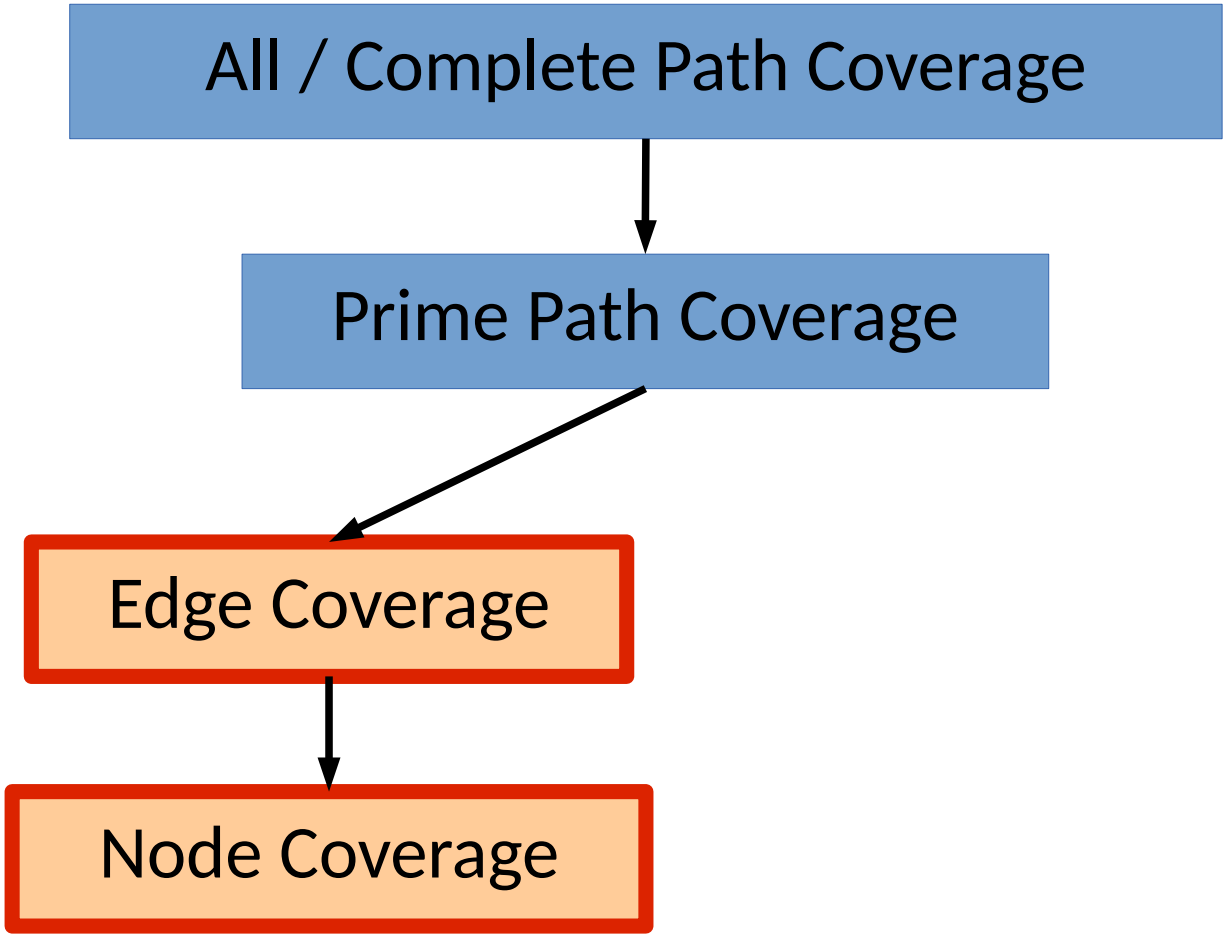
Can you think of things they miss?

How Do They Relate?

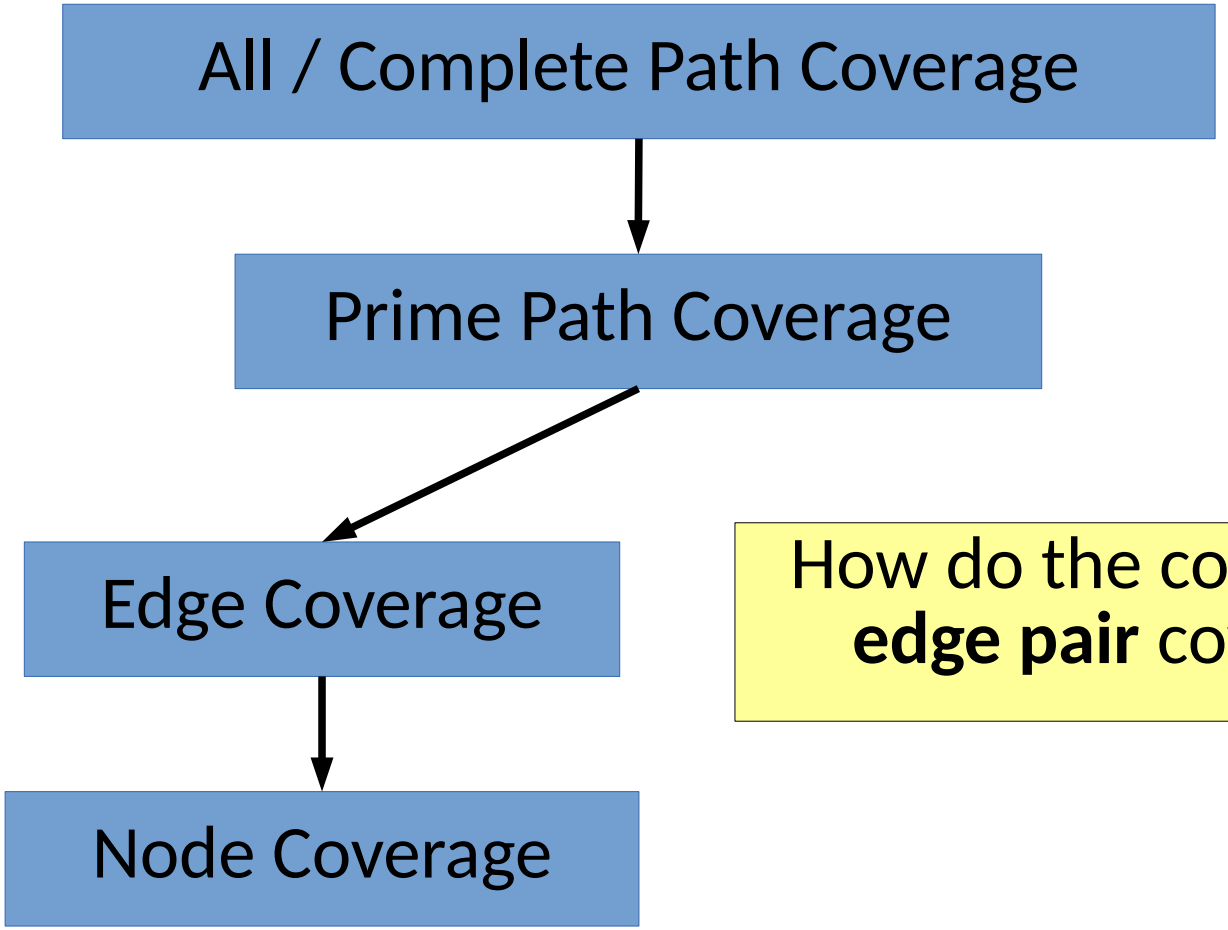


How do they compare w/ edge coverage?

How Do They Relate?

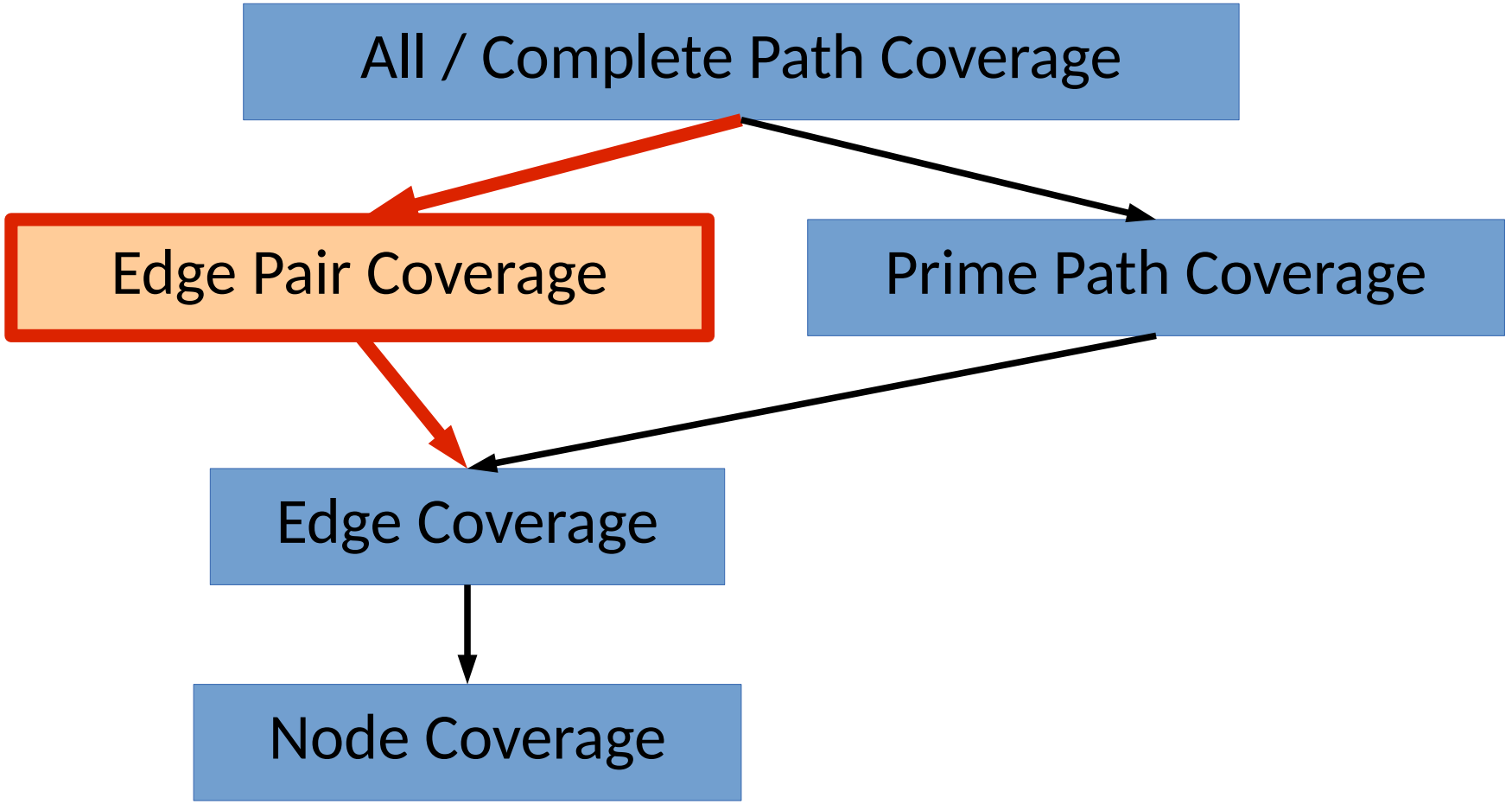


How Do They Relate?

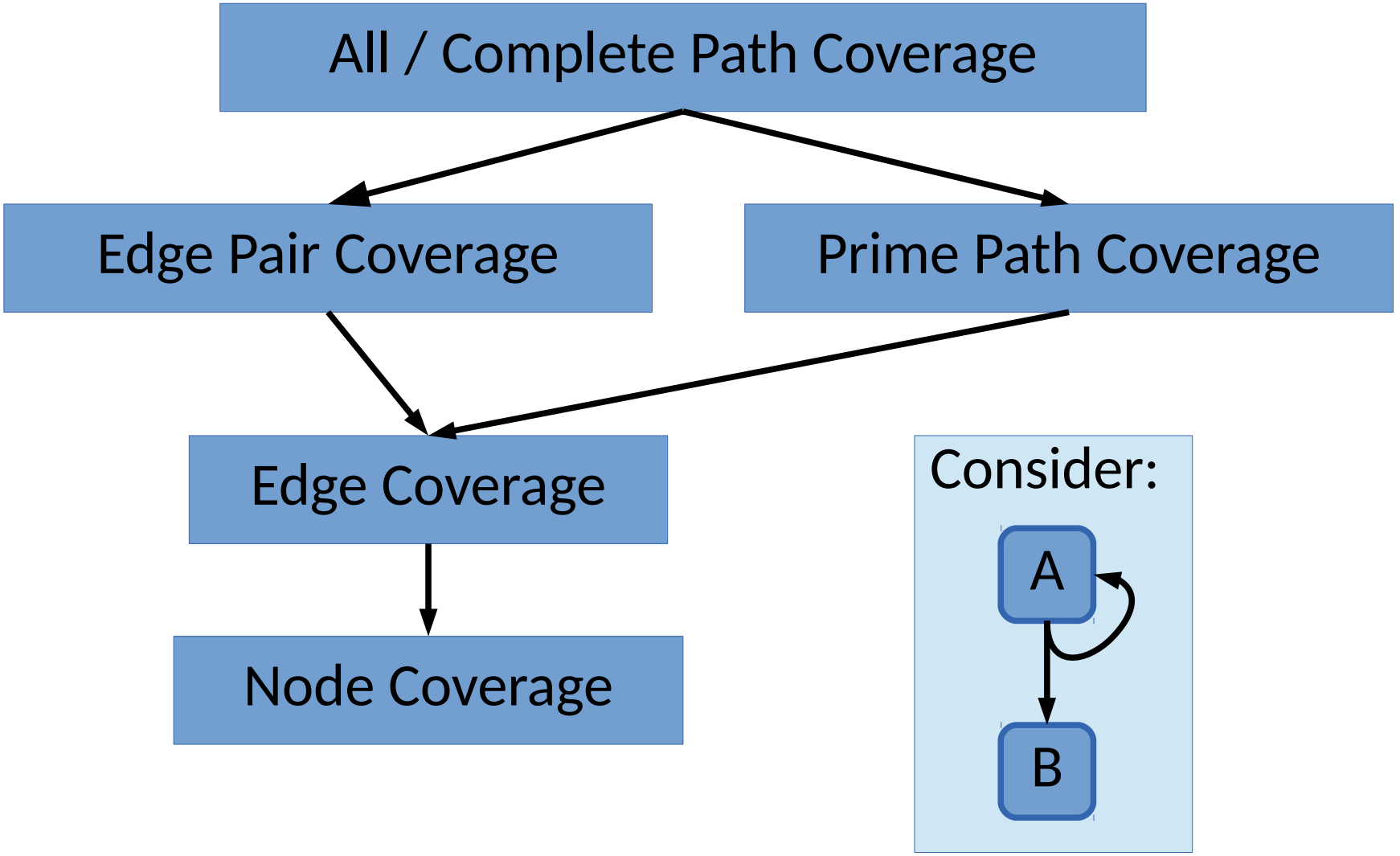


How do the compare w/ **edge pair** coverage?

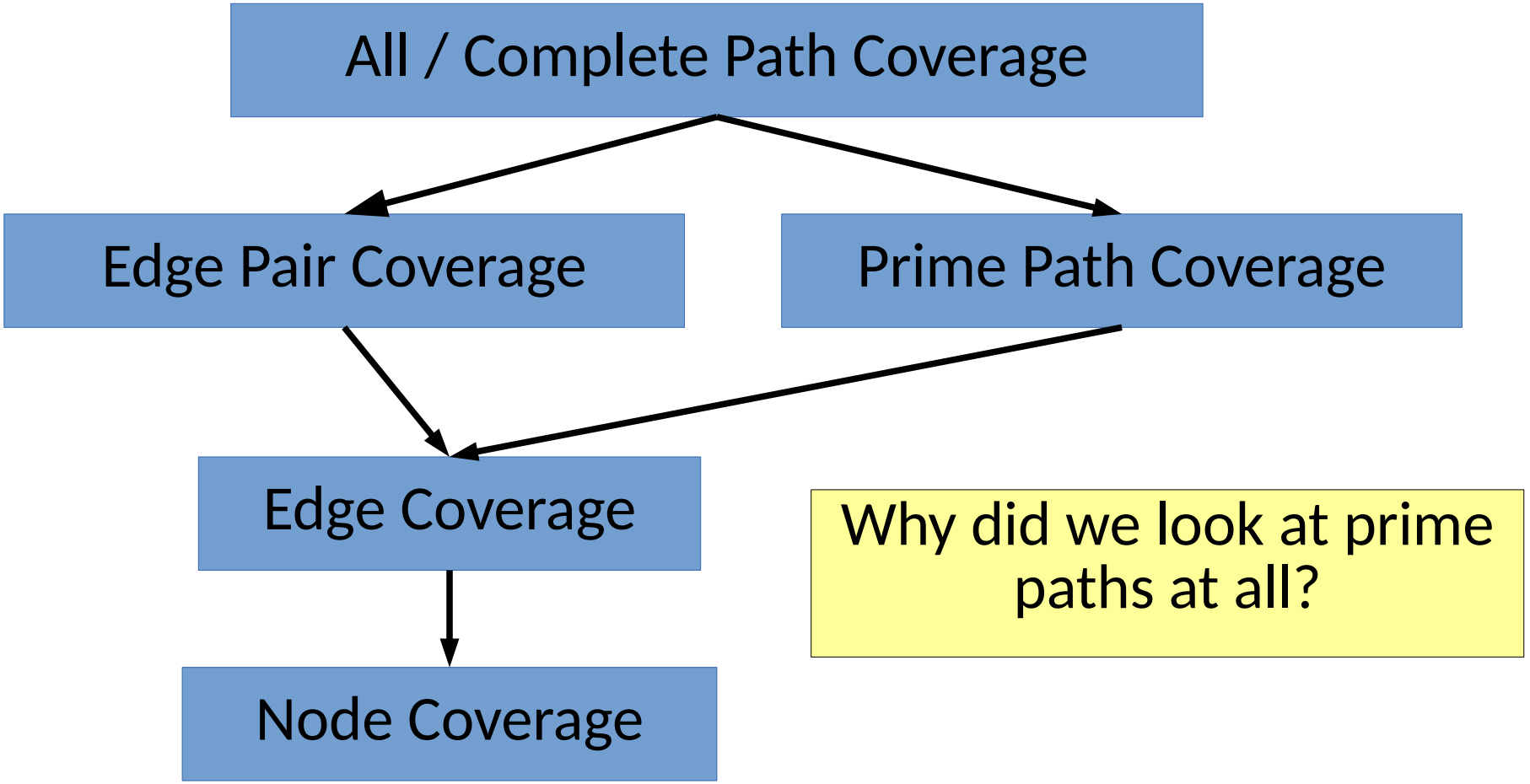
How Do They Relate?



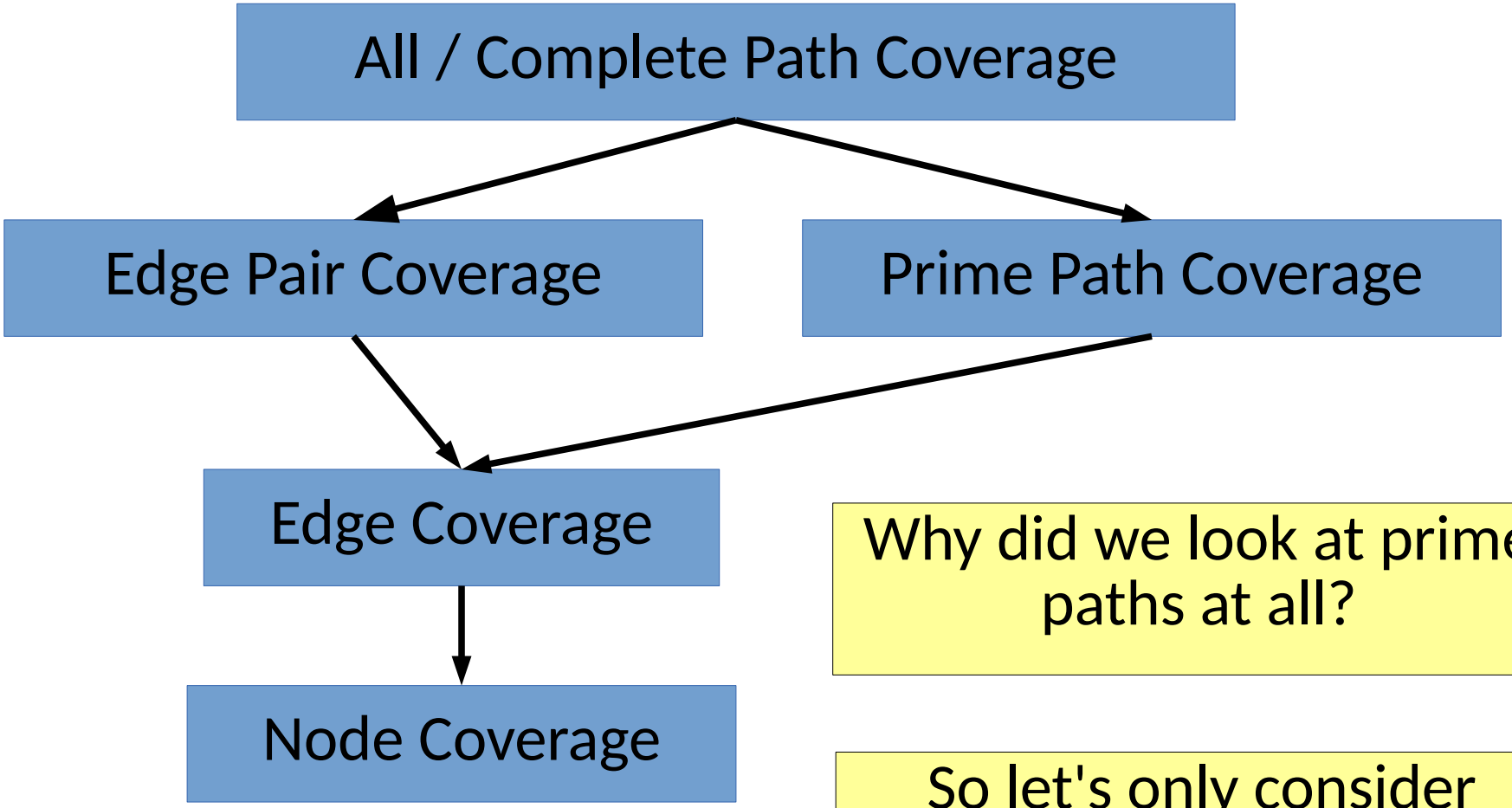
How Do They Relate?



How Do They Relate?



How Do They Relate?



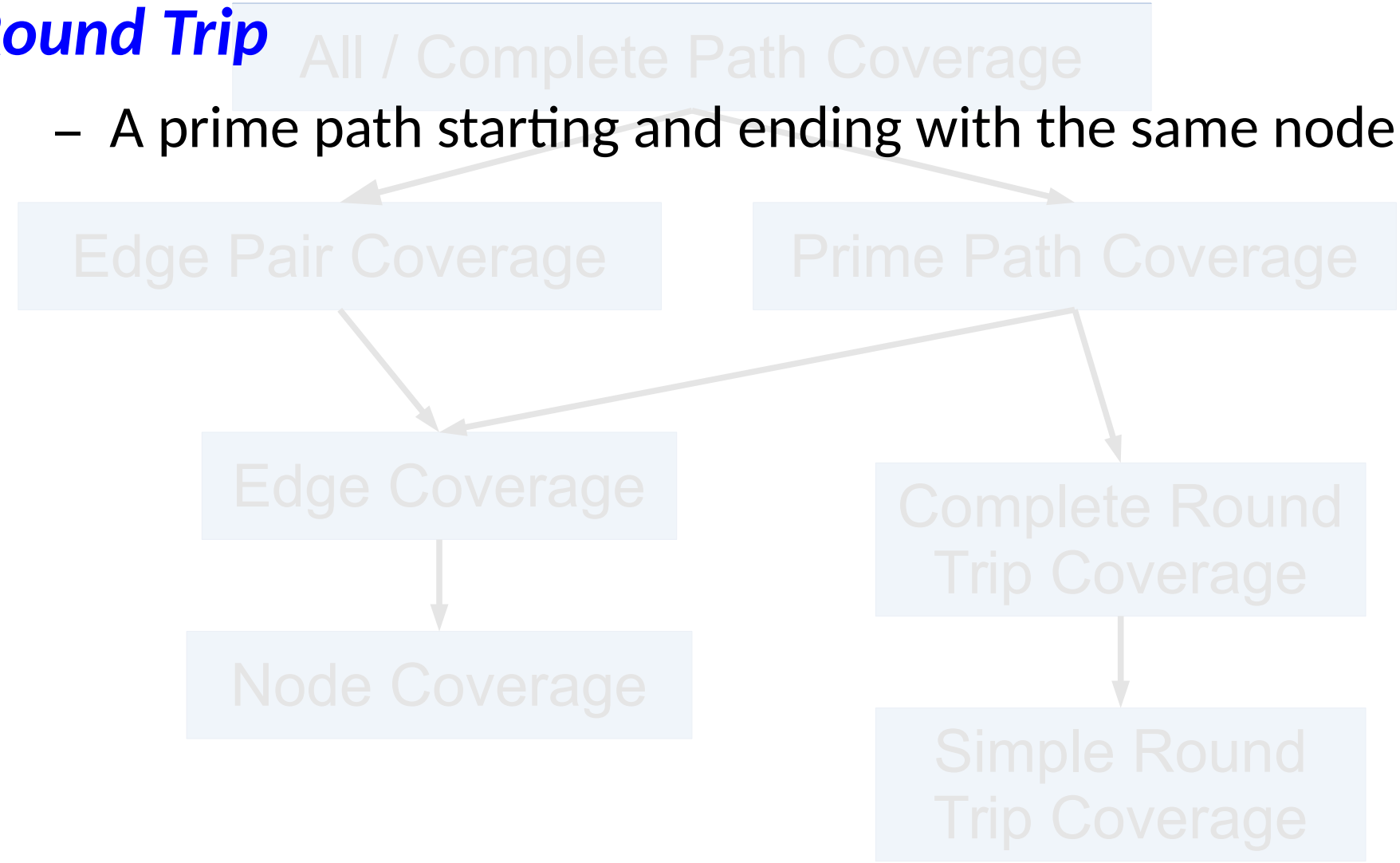
Why did we look at prime paths at all?

So let's only consider loops....

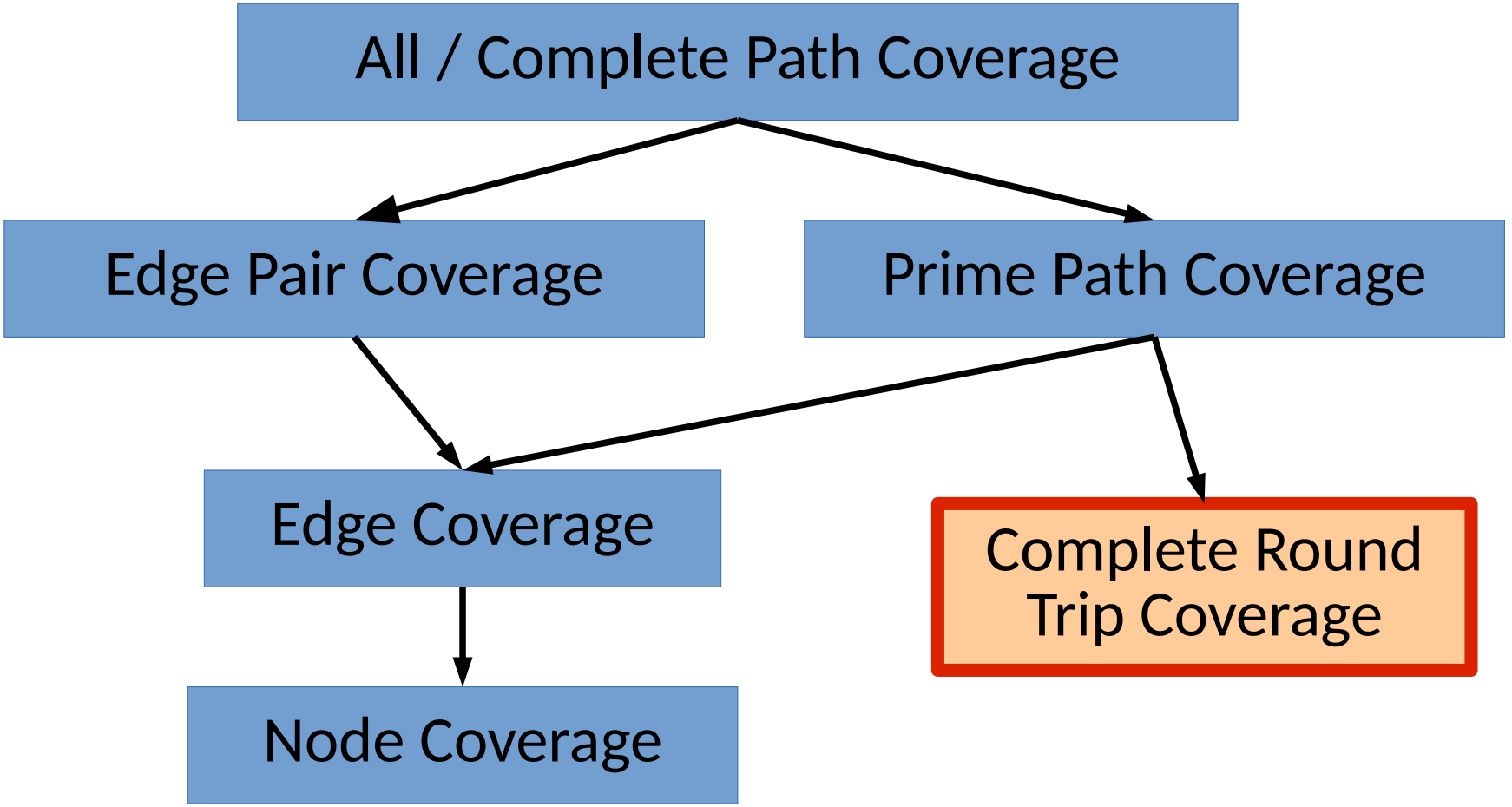
How Do They Relate?

Round Trip

– A prime path starting and ending with the same node

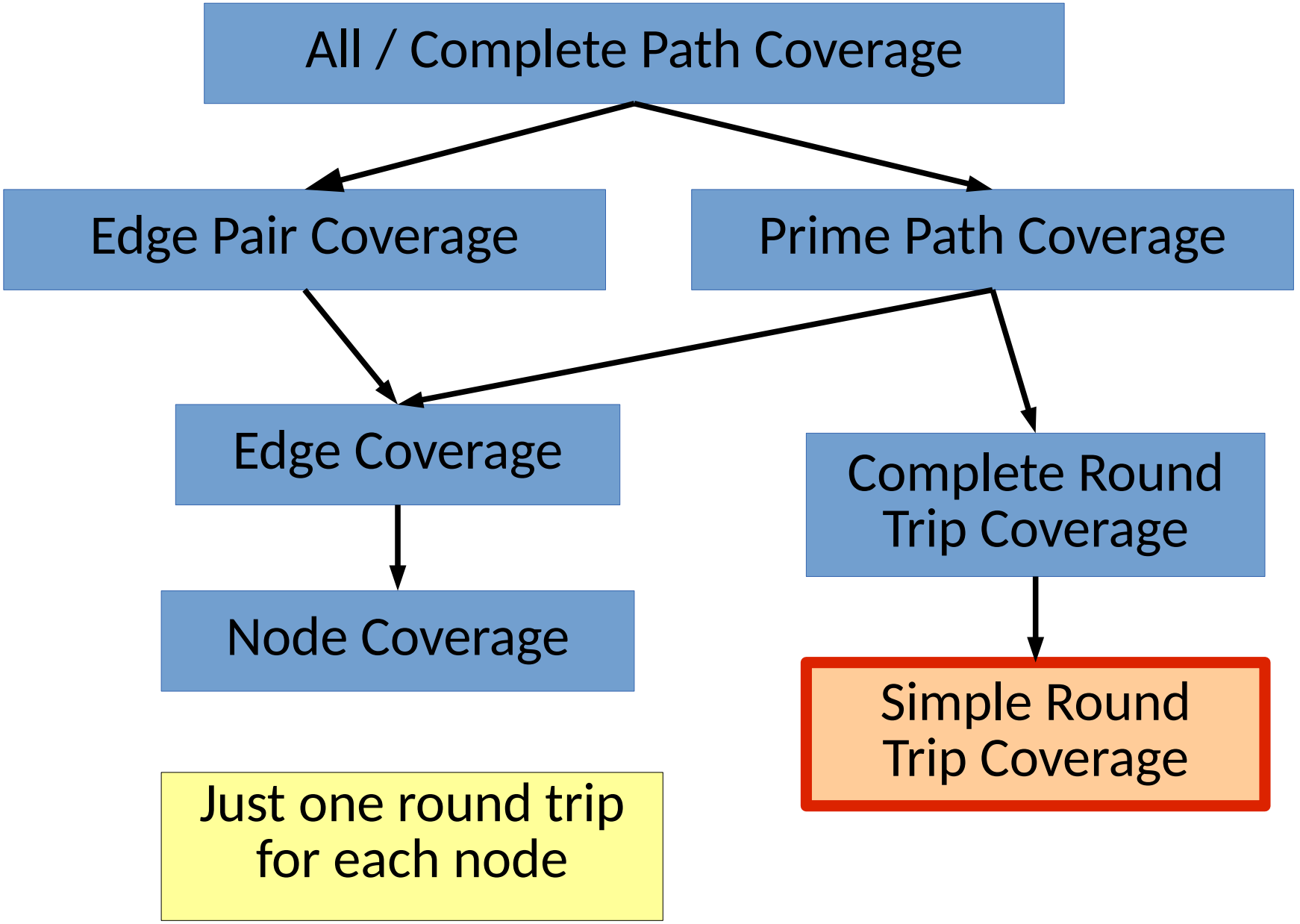


How Do They Relate?



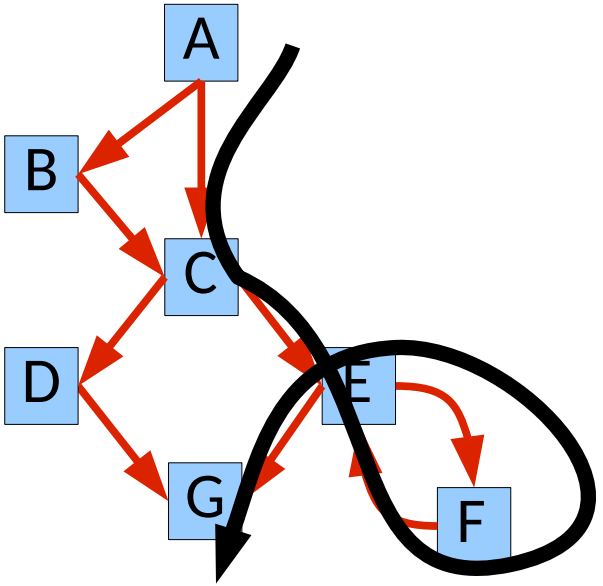
All round trips for each node

How Do They Relate?



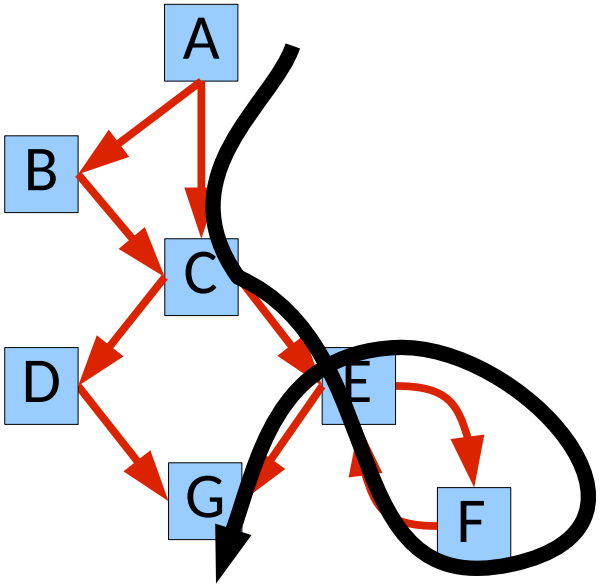
Turning Them Into Tests

- Reconsider:



Turning Them Into Tests

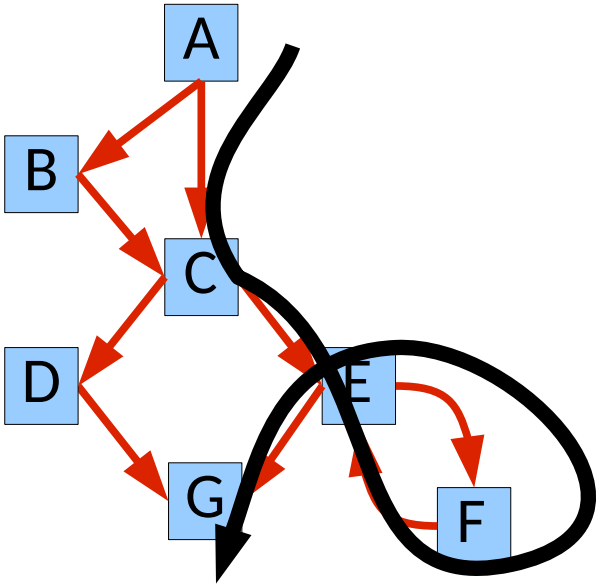
- Reconsider:



Is this path prime?

Turning Them Into Tests

- Reconsider:

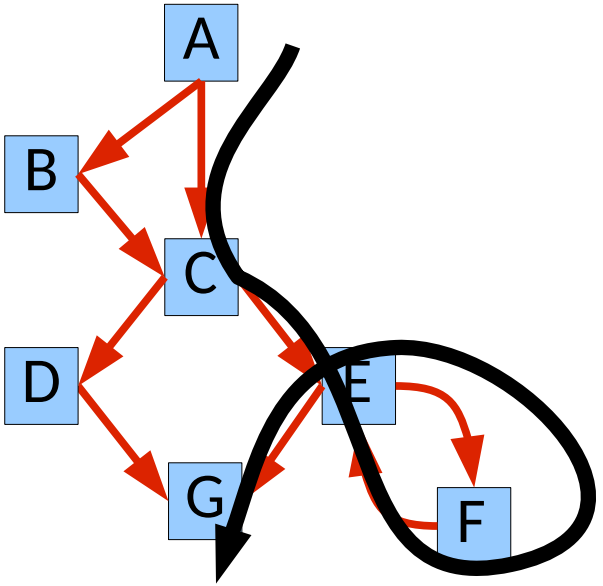


Is this path prime?

Is it still useful?

Turning Them Into Tests

- Reconsider:



Is this path prime?

Is it still useful?

One test may cover multiple prime paths!

Requirements ≠ Tests

Turning Them Into Tests

- Relaxing our path requirements can help, too

Turning Them Into Tests

- Relaxing our path requirements can help, too
- Tour
  - A path *p* tours path *q* if *q* is a subpath of *p*
  - A test covers any prime path it tours

Turning Them Into Tests

- Relaxing our path requirements can help, too
- Tour
  - A path *p* tours path *q* if *q* is a subpath of *p*
  - A test covers any prime path it tours

This is strict!
Can we relax it?

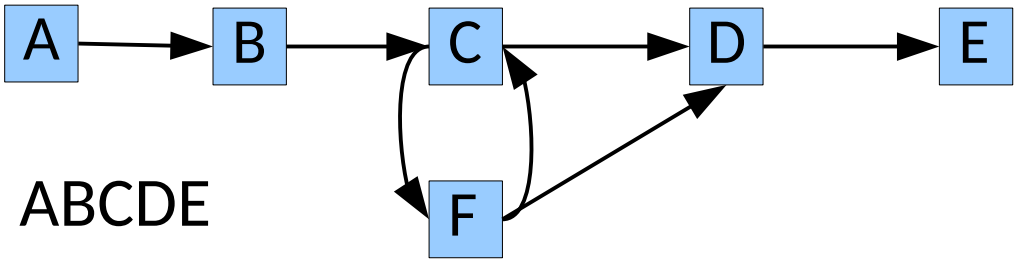
Turning Them Into Tests

- Relaxing our path requirements can help, too
- Tour
  - A path *p* tours path *q* if *q* is a subpath of *p*
  - A test covers any prime path it tours
- Tour with sidetrips
  - Iff every ***edge*** of *q* appears in the same order in *p*
  - "Return to where you left"

Turning Them Into Tests

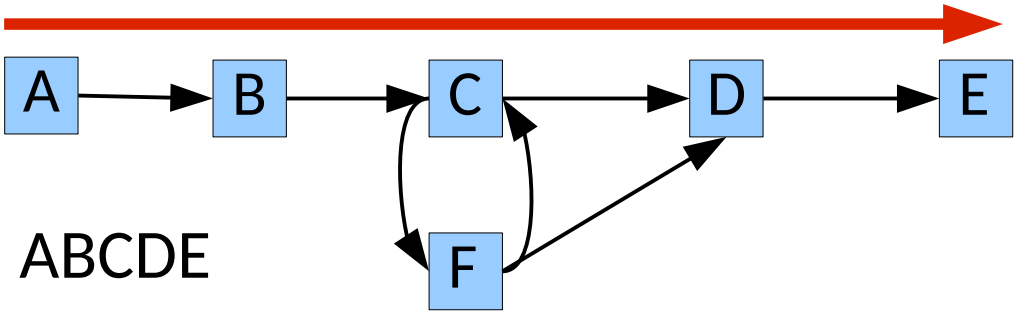
- Relaxing our path requirements can help, too
- Tour
  - A path *p* tours path *q* if *q* is a subpath of *p*
  - A test covers any prime path it tours
- Tour with sidetrips
  - Iff every ***edge*** of *q* appears in the same order in *p*
  - "Return to where you left"
- Tour with detours
  - Iff every ***node*** of *q* appears in the same order in *p*

Turning Them Into Tests



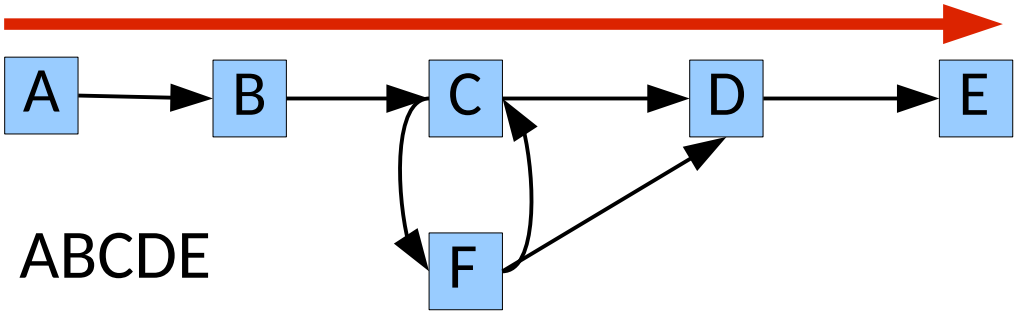
Strict Tour: ABCDE

Turning Them Into Tests

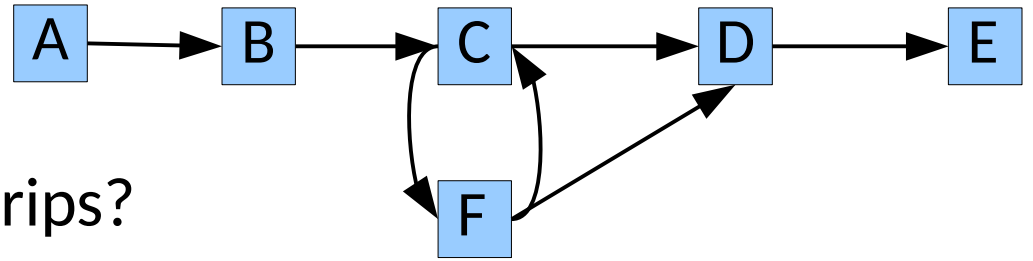


Strict Tour: ABCDE

Turning Them Into Tests

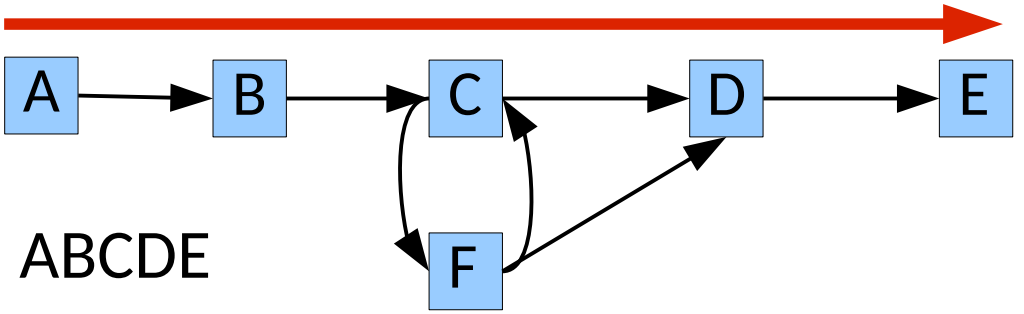


Strict Tour: ABCDE

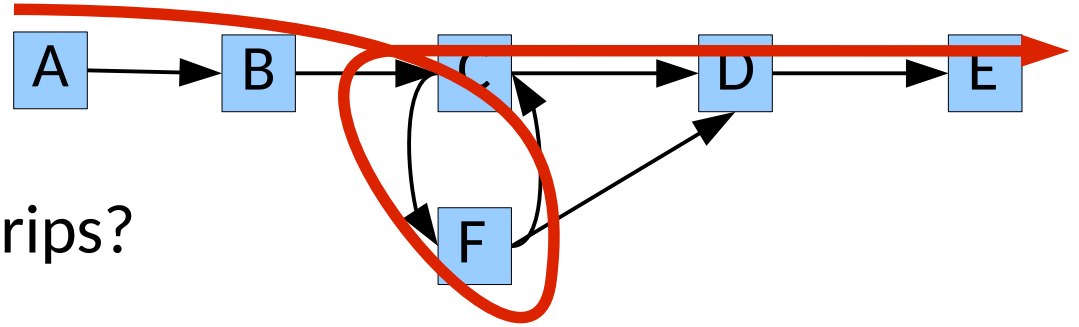


With Sidetrips?

Turning Them Into Tests

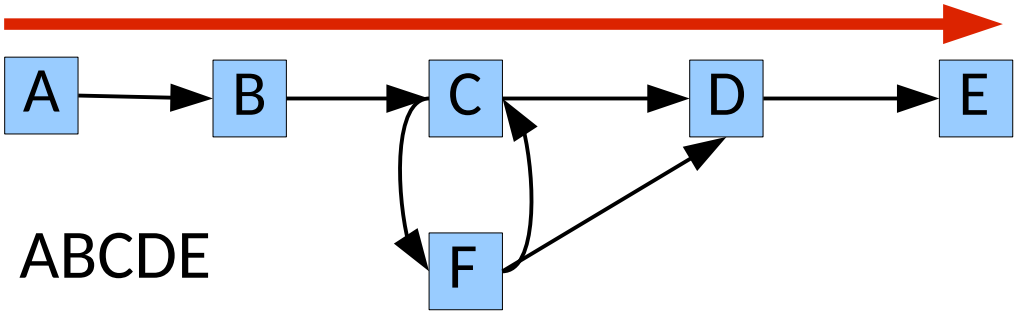


Strict Tour: ABCDE

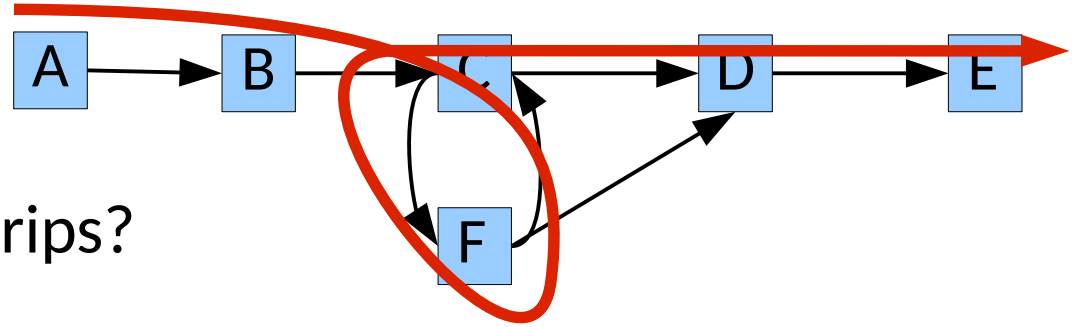


With Sidetrips?

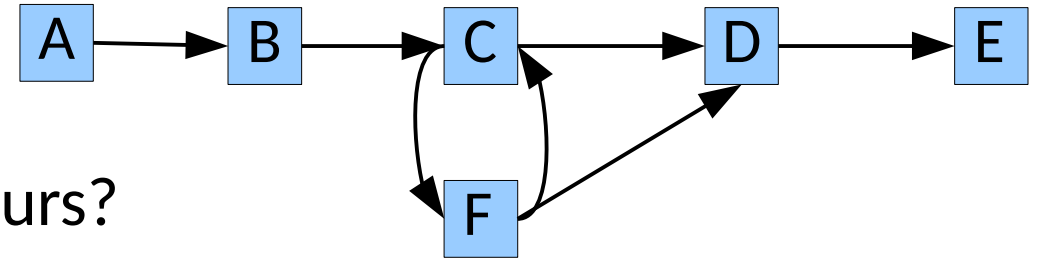
Turning Them Into Tests



Strict Tour: ABCDE

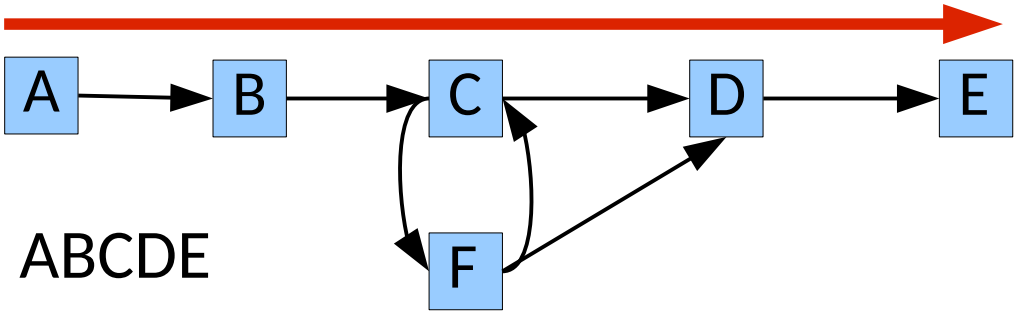


With Sidetrips?

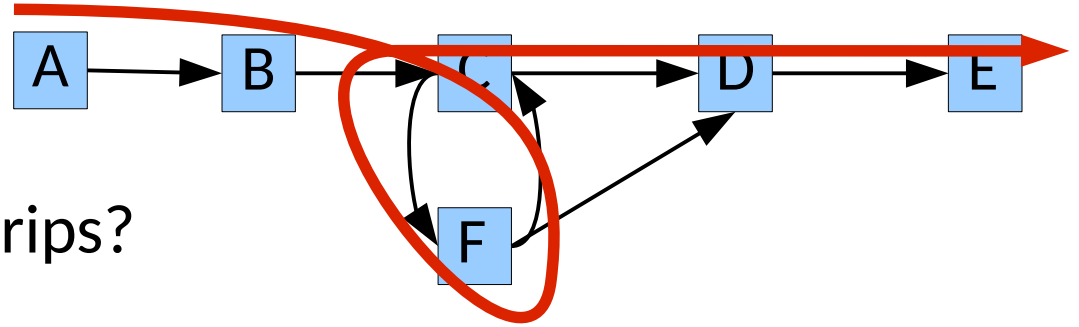


With Detours?

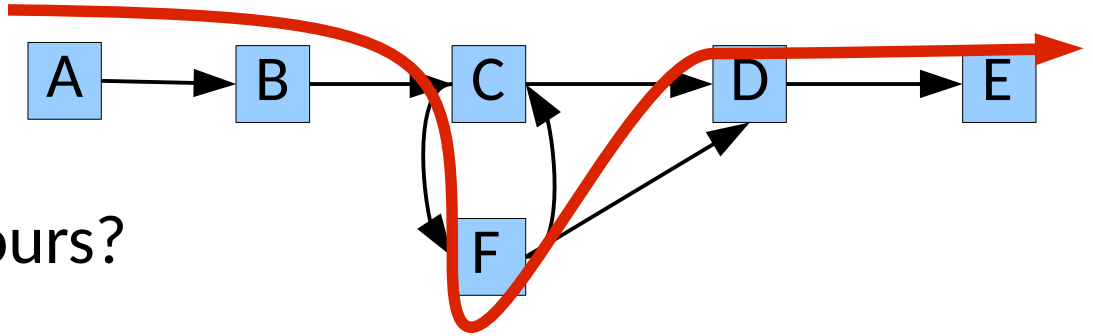
Turning Them Into Tests



Strict Tour: ABCDE



With Sidetrips?



With Detours?

Turning Them Into Tests

- Do these relaxations help us with problems we have seen?

Turning Them Into Tests

- Do these relaxations help us with problems we have seen?
- Can you see any problems they may introduce?

Turning Them Into Tests

- Do these relaxations help us with problems we have seen?
- Can you see any problems they may introduce?
- How might this affect how you use them?

Onward

- But sometimes the structure of the CFG is not enough…