جامعة نيويورك أبوظبي

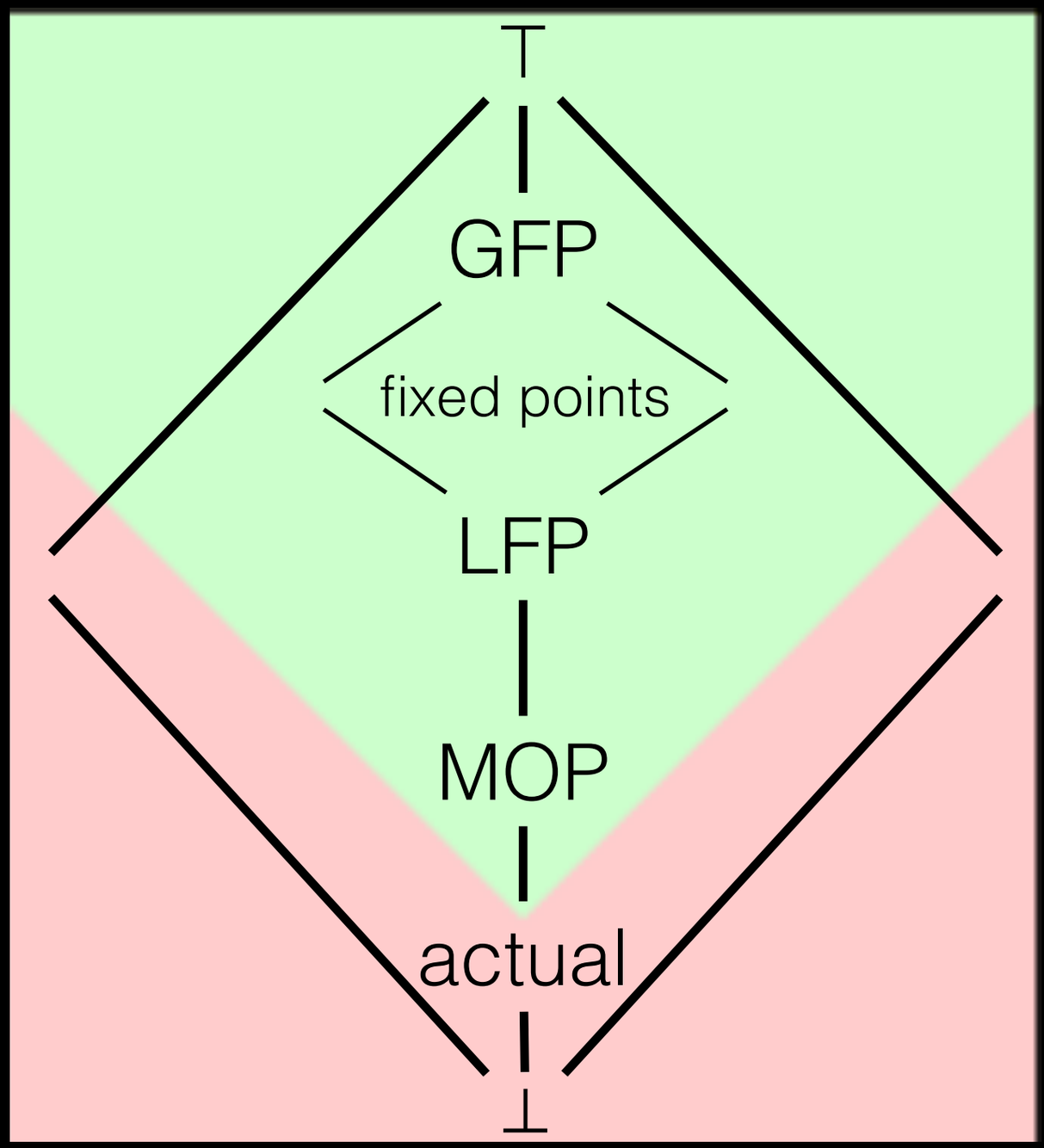
NYU ABU DHABI

Call Graph Construction

CS-UH 3260
Static Program Analysis

Karim Ali
@karimhamdanali

Previously: MOP ⊑ LFP



- Every solution S ⊒ *actual* is "safe" (i.e., sound).
- MOP ⊒ *actual*
- LFP ⊒ MOP
- A flow function `f` is distributive if `f(x) ⊔ f(y) = f(x ⊔ y)`
- If all flow functions are distributive, then LFP = MOP
- Initializing using ⊤ instead of ⊥ causes earlier termination, but yields more imprecise fixed-point

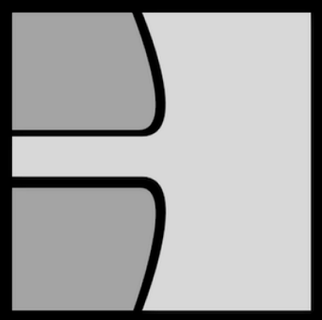
Previously: Designing a Dataflow Analysis

1. Forwards or backwards?
2. What is the domain of the analysis info (lattice elements)?
3. What's the effect a statement has on the info? (flow functions)
4. What values hold at program entry points?
5. What's the initial estimate? It's the unique element $\bot$ such that $\forall_x \bot \sqcup x = x$.
6. How to merge info? (join operator)


julia


Soot


UNIVERSITY OF
18
56
MARYLAND
FindBugs
because it's easy
™




CHECKER
framework

julia

Soot

UNIVERSITY OF MARYLAND 18 56

FindBugs™
because it's easy

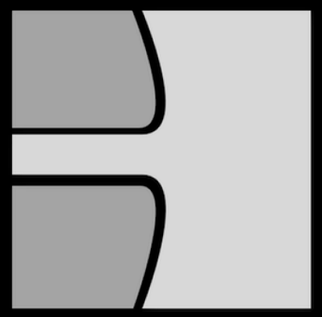
DOOP

CHECKMARX

coverity®
A Synopsys Company

AbsInt

FORTIFY



WALA
T. J. WATSON LIBRARIES FOR ANALYSIS

CHECKER framework


julia


Soot


UNIVERSITY OF
18
56
MARYLAND
FindBugs
because it's easy

Call Graph

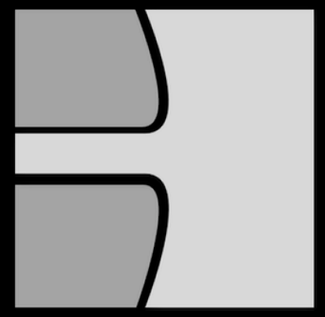

DOOP


CHECKMARX


Sources
Sinks


WALA
T. J. WATSON LIBRARIES FOR ANALYSIS


coverity
A Synopsys Company




AbsInt

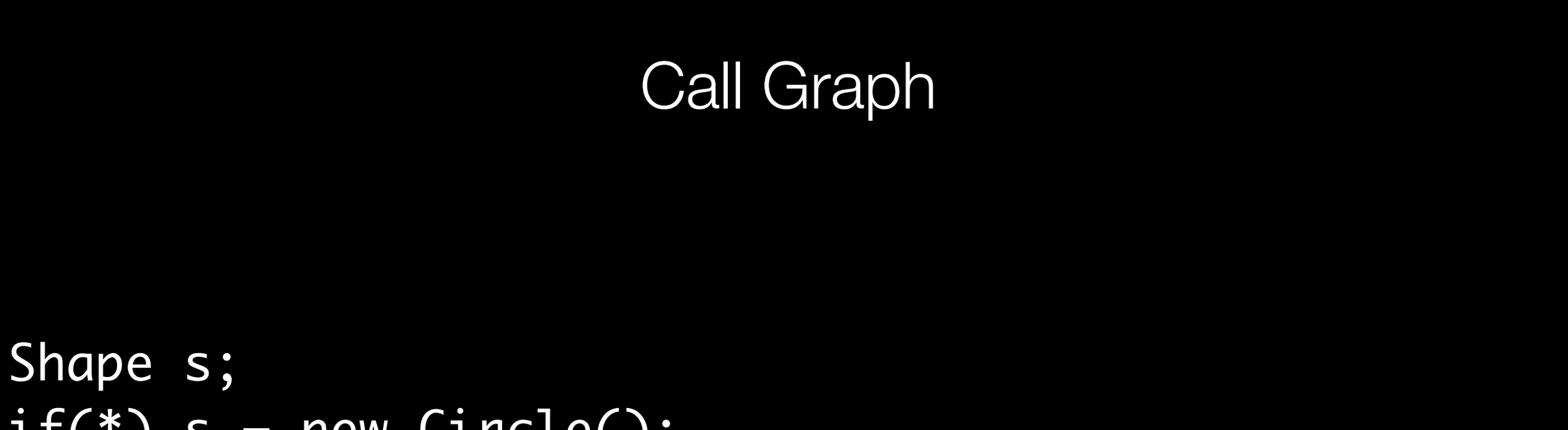

FORTIFY


CHECKER
framework

… so what is a Call Graph?

Call Graph

```
Shape s;
if(*) s = new Circle();
else s = new Square();
```



Call Graph

```
Shape s;
if(*) s = new Circle();
else s = new Square();
s.draw();
```

```
class Circle extends Shape
{ void draw() { ... } }

class Square extends Shape
{ void draw() { ... } }
```

required by every inter-procedural analysis

Let’s build a Call Graph

```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```

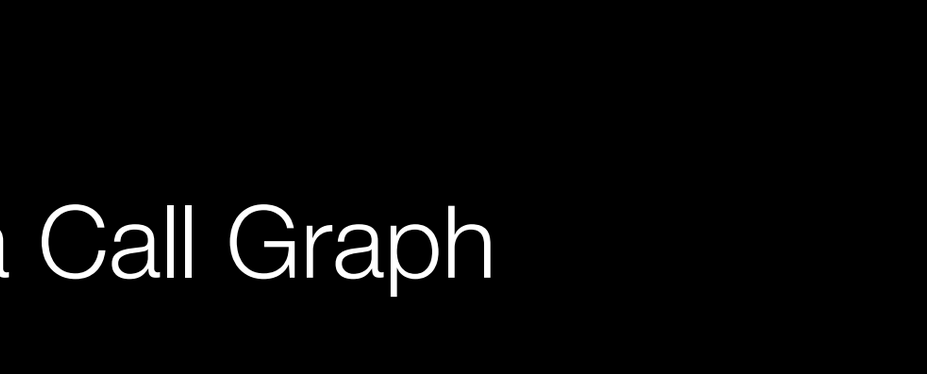
Let’s build a Call Graph

```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```

Main.main()

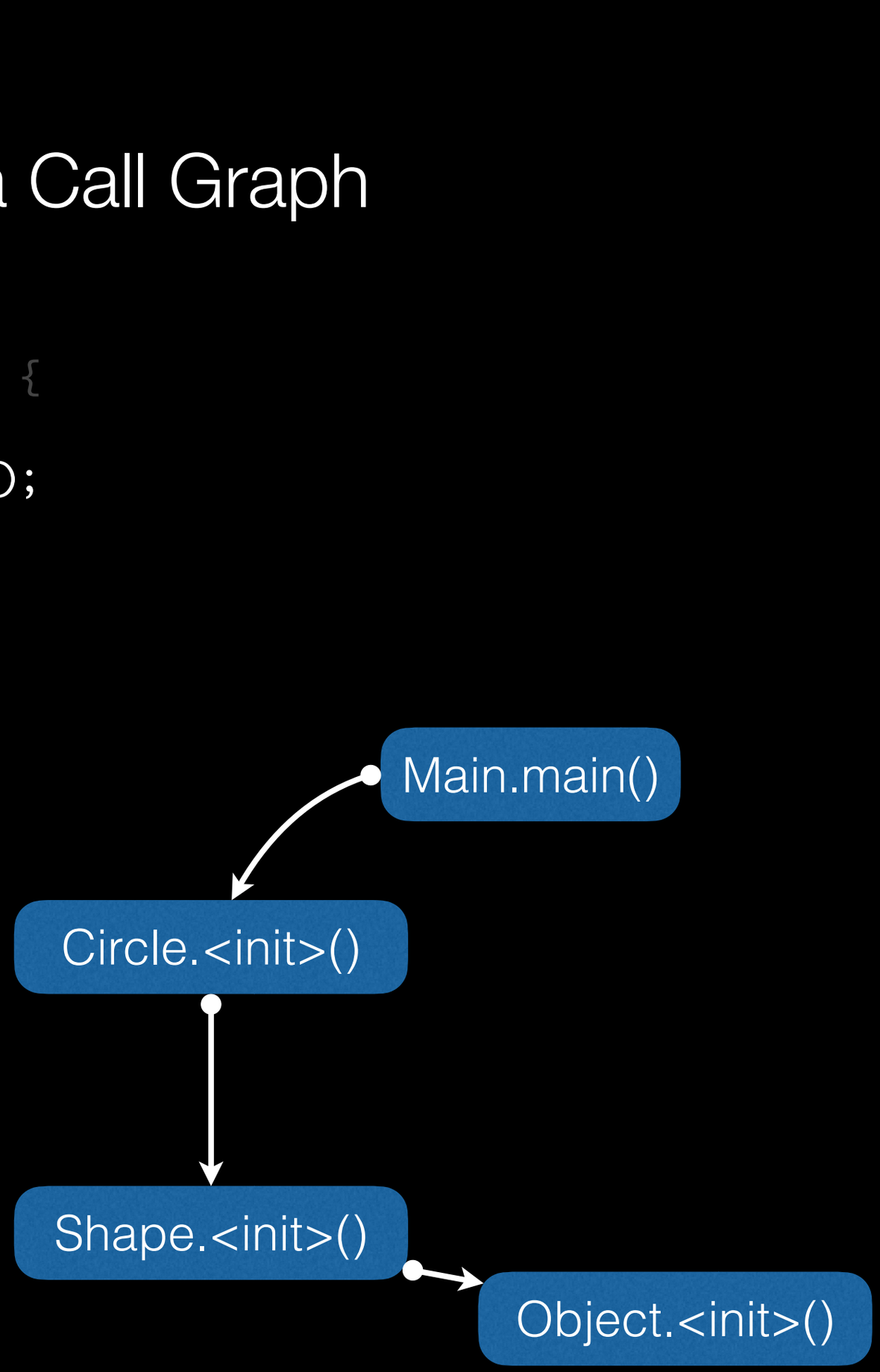
Let’s build a Call Graph

```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```



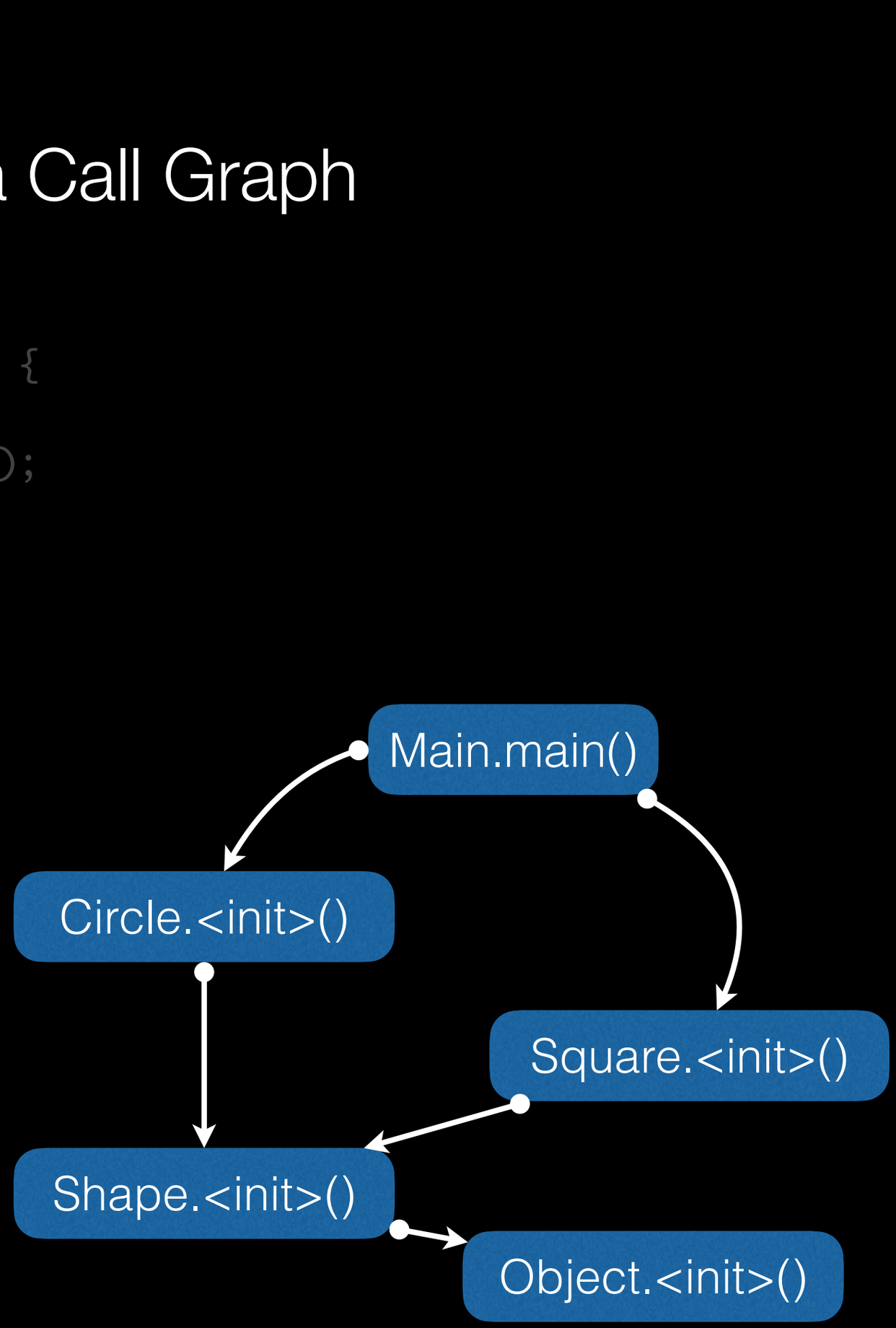
Let’s build a Call Graph

```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```



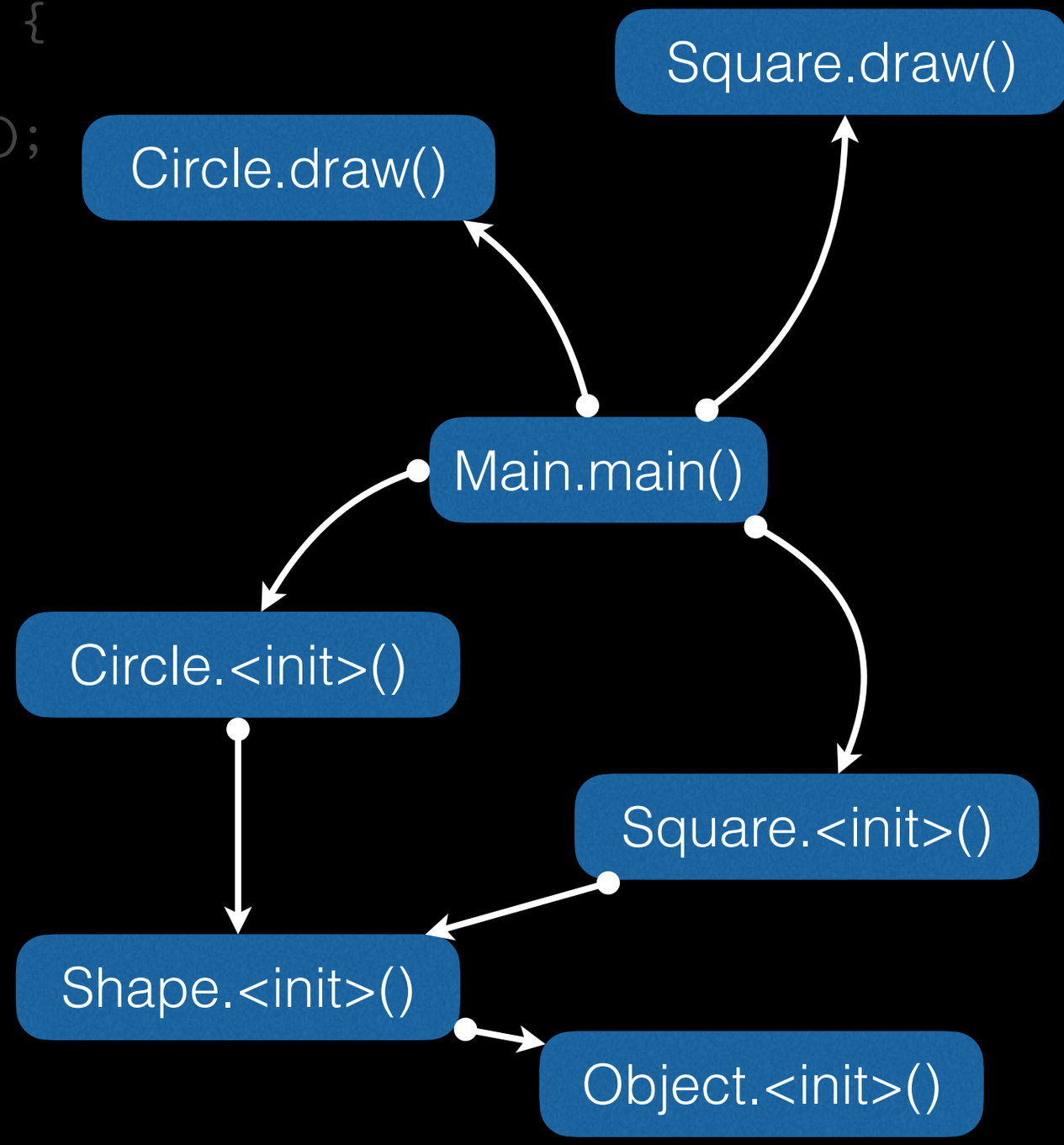
Let’s build a Call Graph

```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```

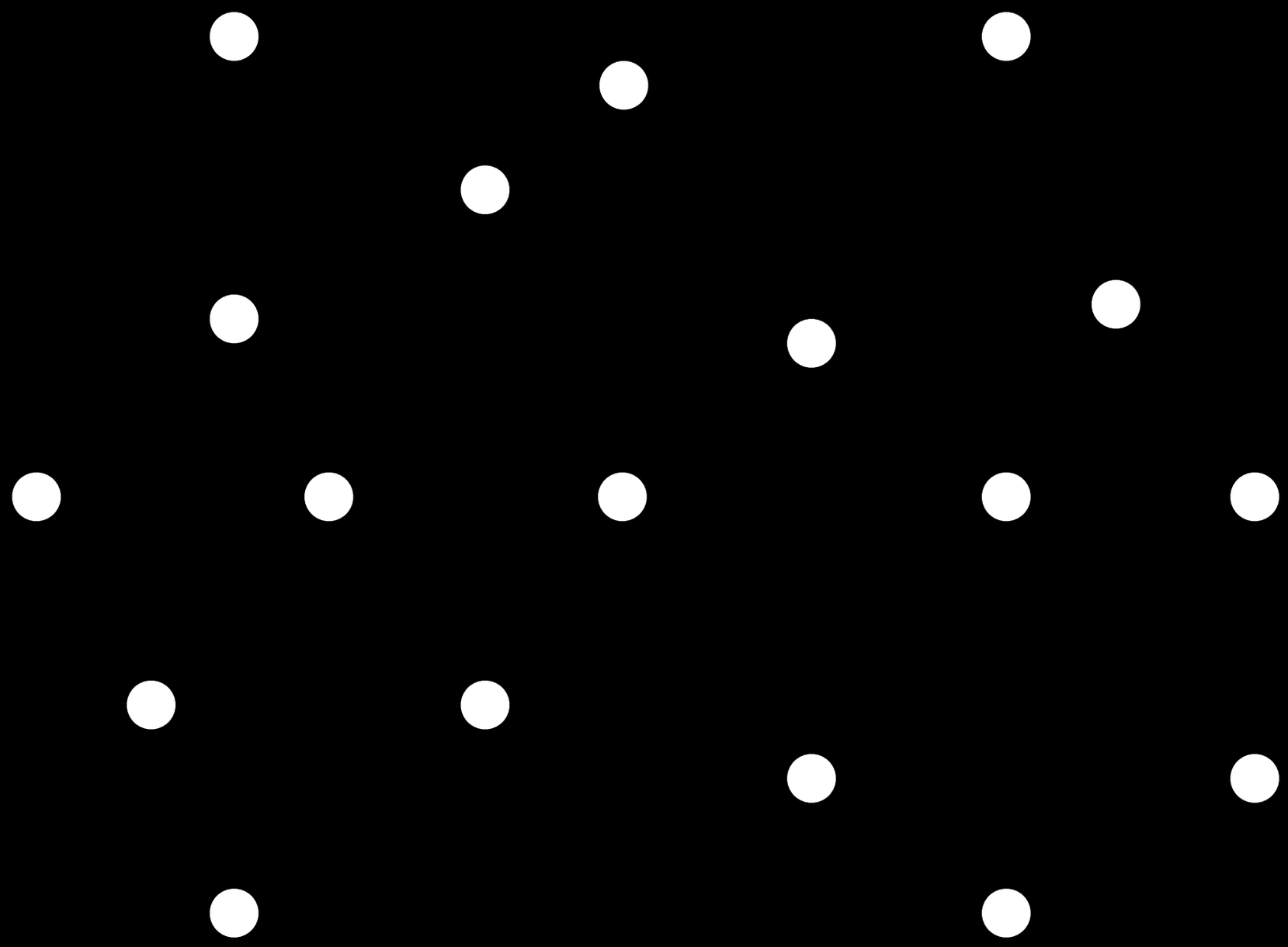


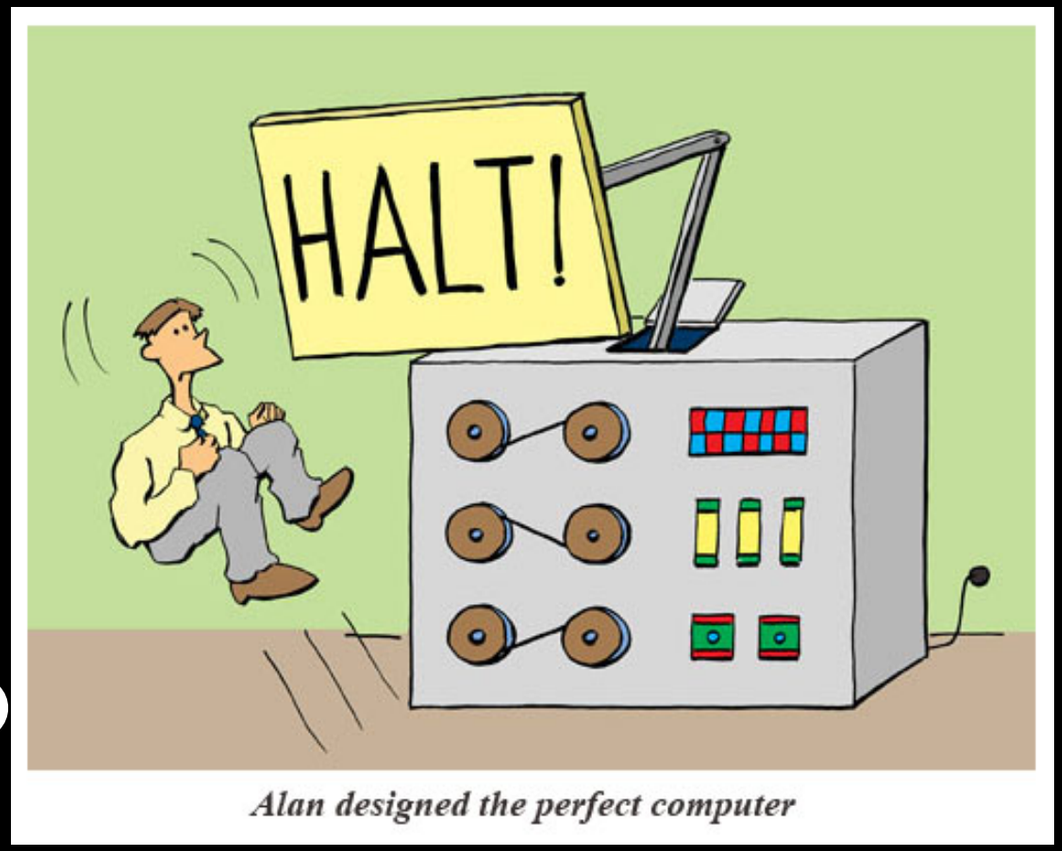
Let’s build a Call Graph

```
public class Main {
  public static void main(String[] args) {
    Shape s;
    if (args.length > 2) s = new Circle();
    else s = new Square();

    s.draw();
  }
}

abstract class Shape {
  abstract void draw();
}

class Circle extends Shape {
  void draw() { ... }
}

class Square extends Shape {
  void draw() { ... }
}
```



but … polymorphism!

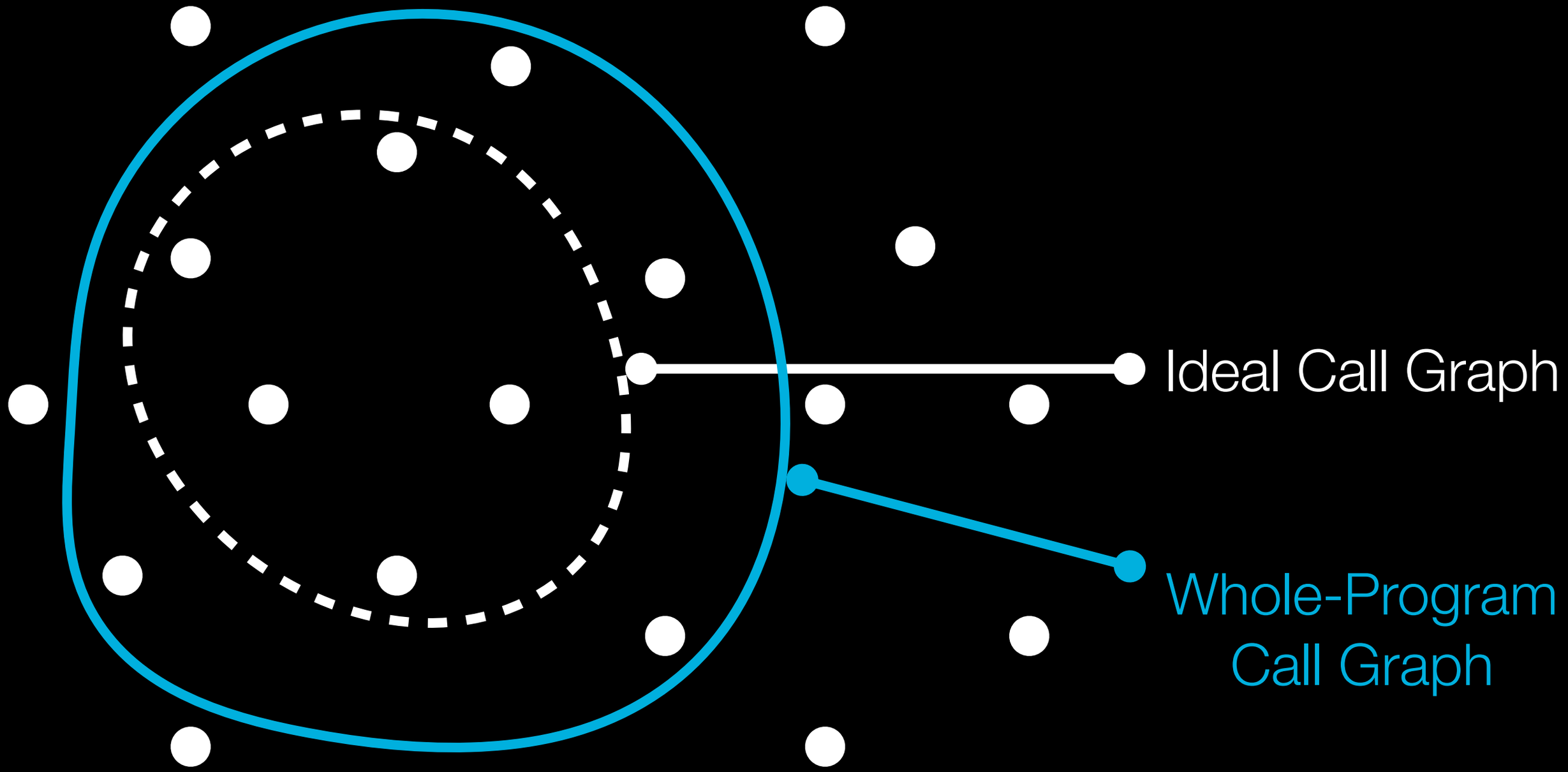


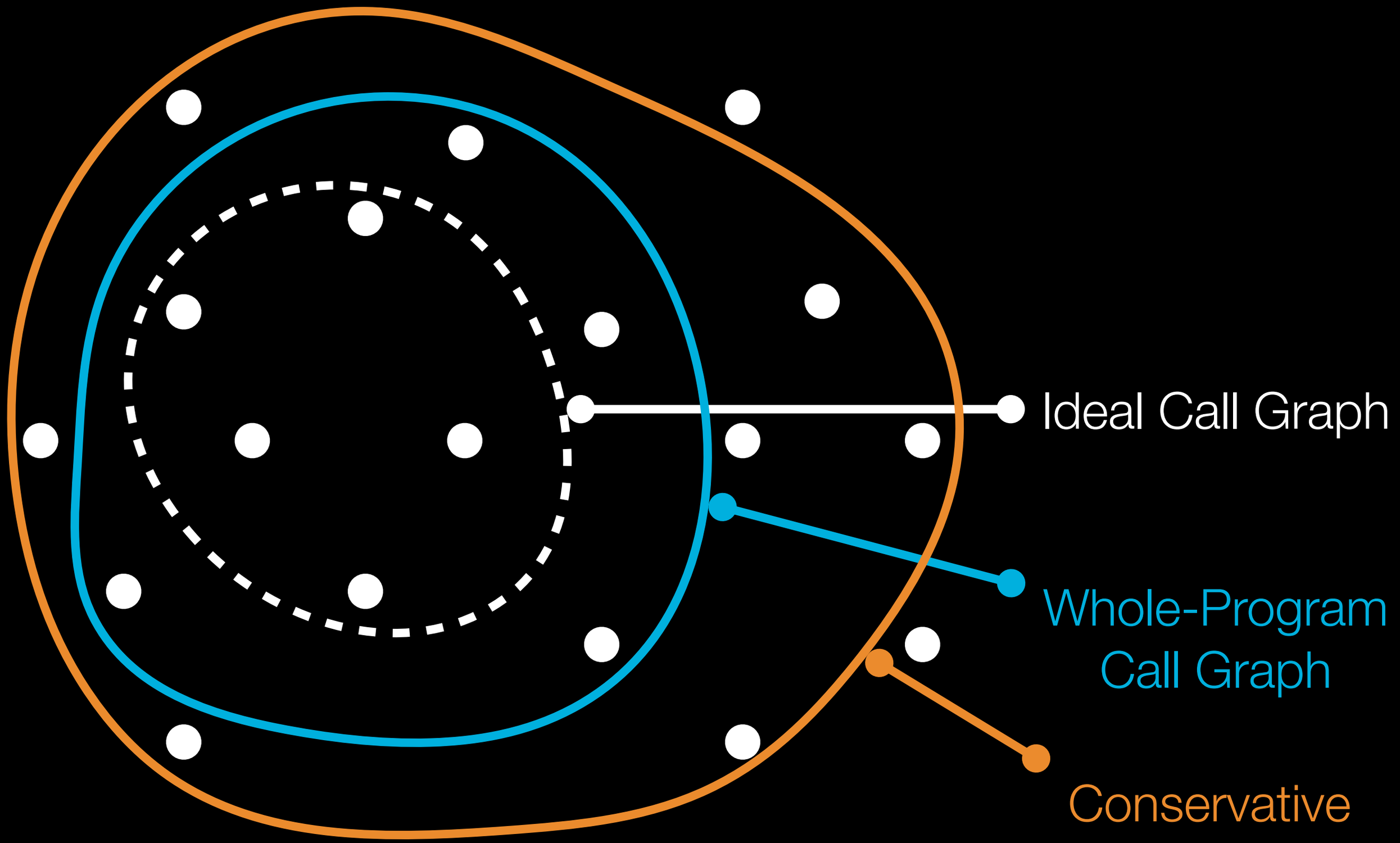


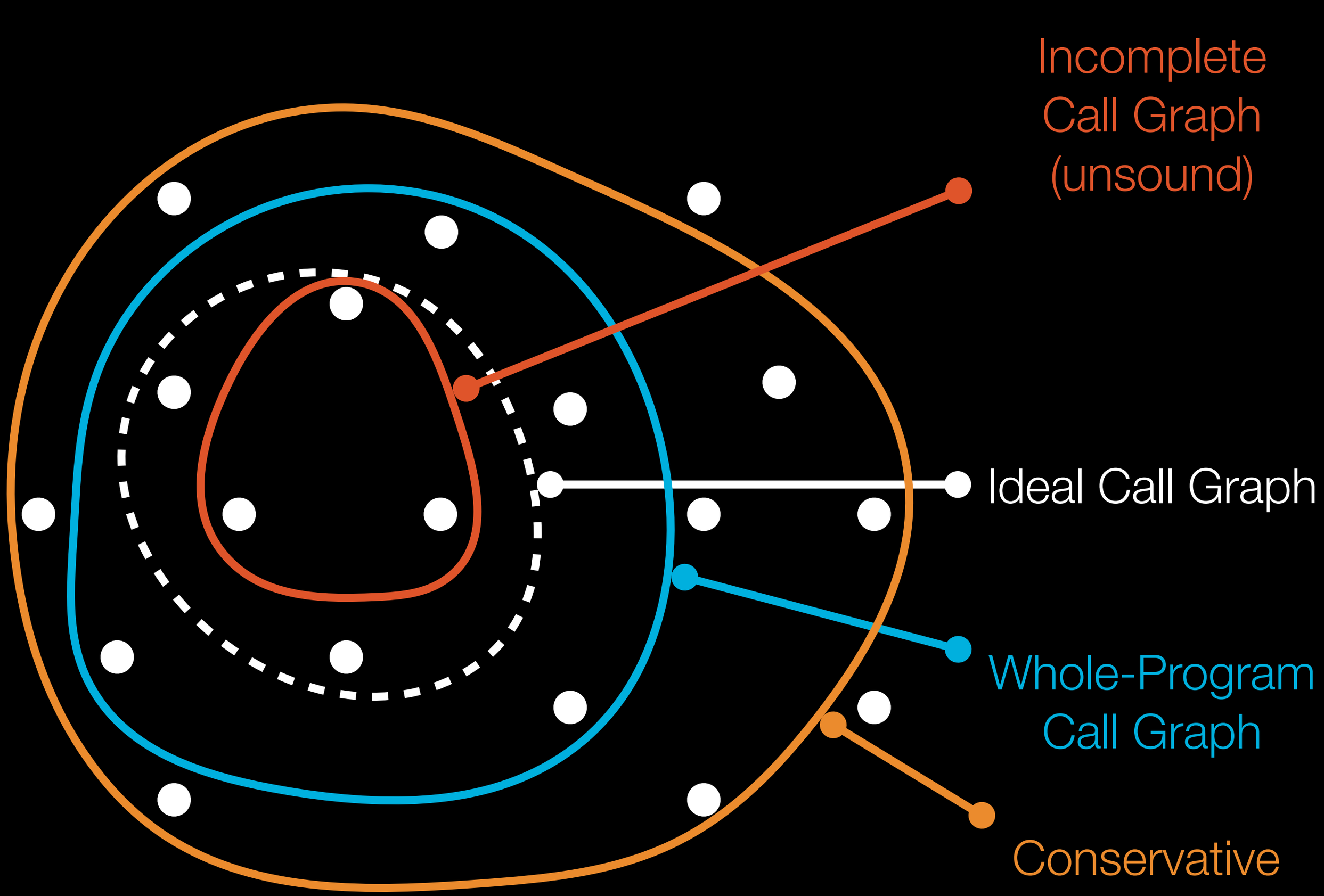
Alan designed the perfect computer

Ideal Call Graph

Image: CooperToons


Ideal Call Graph
Whole-Program
Call Graph


Ideal Call Graph
Whole-Program
Call Graph
Conservative
Call Graph
(highly imprecise)


Incomplete
Call Graph
(unsound)
Ideal Call Graph
Whole-Program
Call Graph
Conservative
Call Graph
(highly imprecise)

CG Algorithms

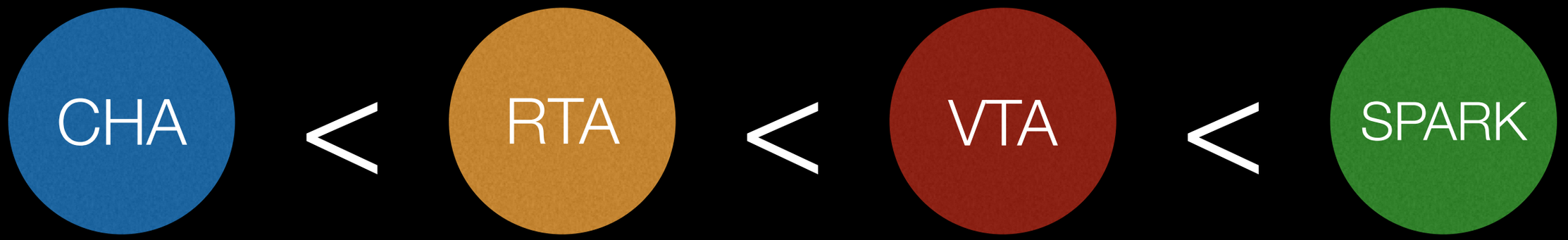
CHA

RTA

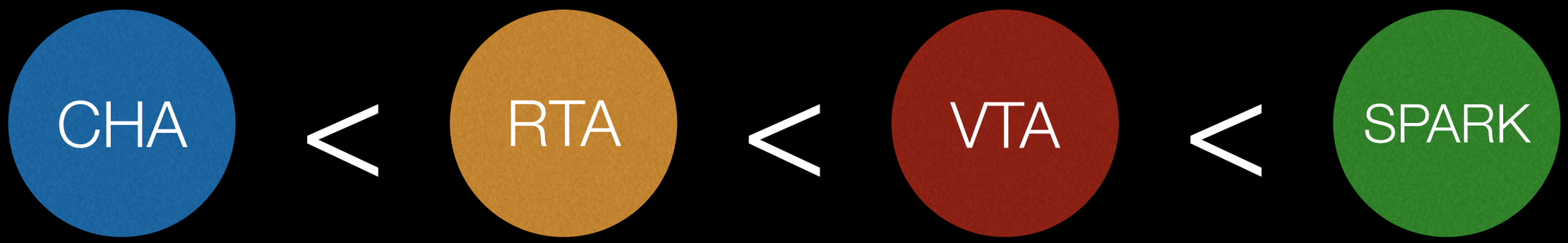
VTA

SPARK

CG Algorithms



CG Algorithms



Class Hierarchy Analysis

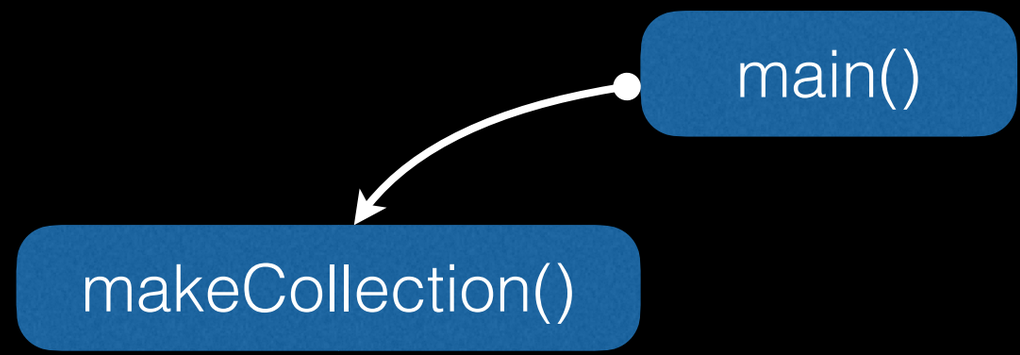
```
public static void main(String[] args) {
  Collection c = makeCollection(args[0]);
  c.add("elem");
}
```

```
static Collection makeCollection(String s) {
  if(s.equals("list")) {
    return new ArrayList();
  } else {
    return new HashSet();
  }
}
```

Jeffrey Dean, David Grove, and Craig Chambers. 1995. Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis. In *Proceedings of the 9th European Conference on Object-Oriented Programming* (ECOOP '95), 77-101.

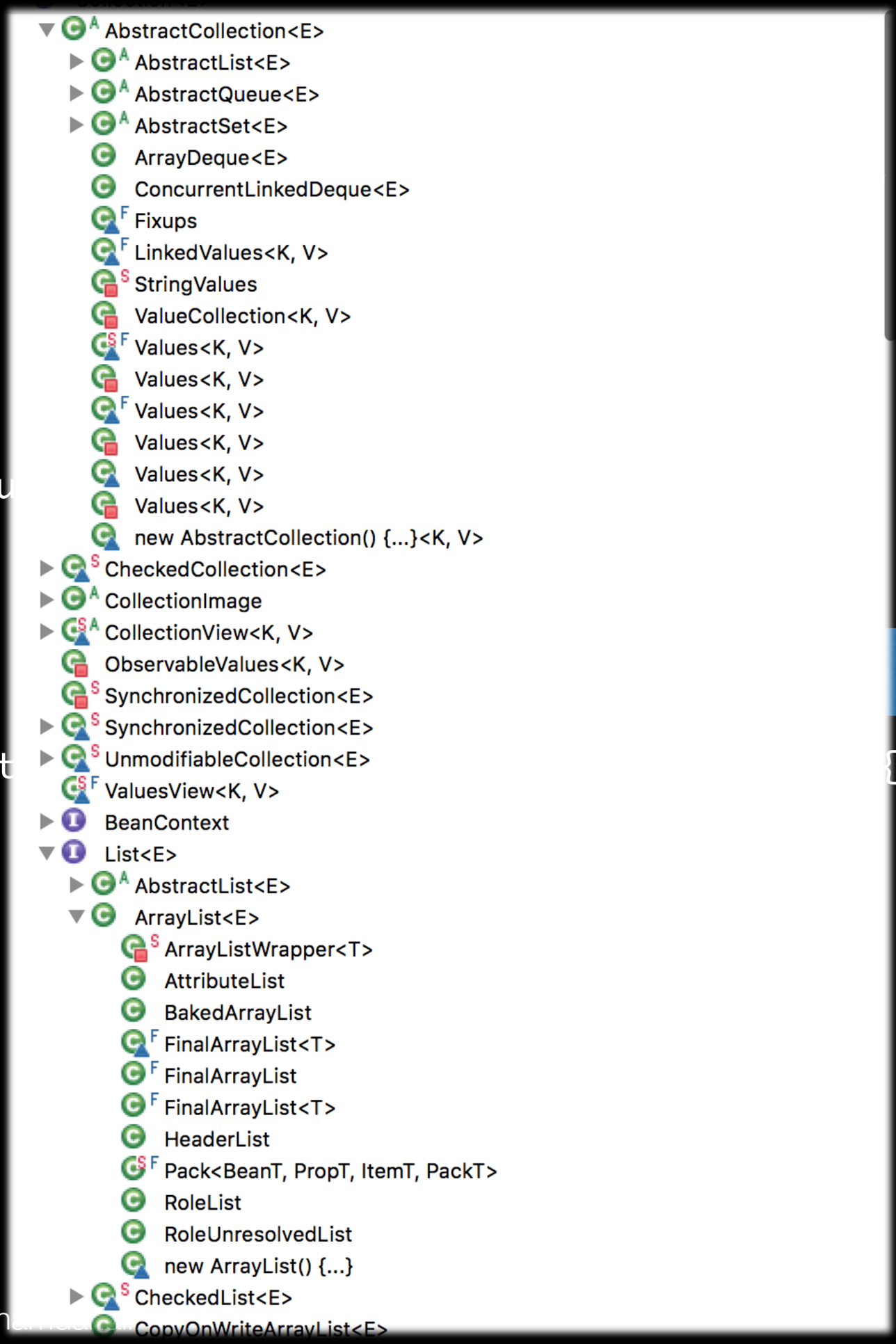
Class Hierarchy Analysis

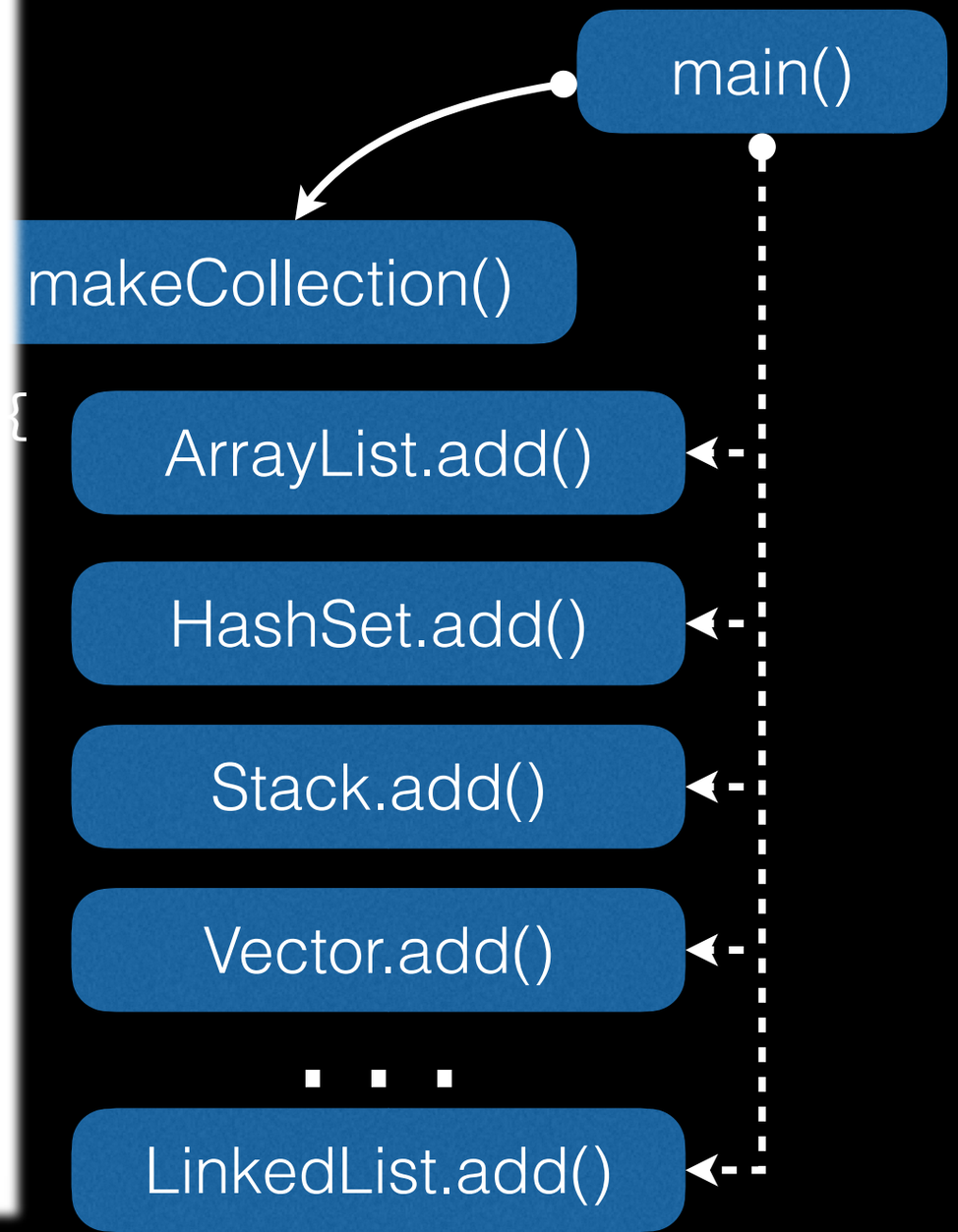
```
public static void main(String[] args) {
  Collection c = makeCollection(args[0]);
  c.add("elem");
}
```




```
static Collection makeCollection(String s) {
  if(s.equals("list")) {
    return new ArrayList();
  } else {
    return new HashSet();
  }
}
```

Analysis


AbstractCollection<E>
AbstractList<E>
AbstractQueue<E>
AbstractSet<E>
ArrayDeque<E>
ConcurrentLinkedDeque<E>
Fixups
LinkedValues<K, V>
StringValues
ValueCollection<K, V>
Values<K, V>
Values<K, V>
Values<K, V>
Values<K, V>
Values<K, V>
Values<K, V>
new AbstractCollection() {...}<K, V>
CheckedCollection<E>
CollectionImage
CollectionView<K, V>
ObservableValues<K, V>
SynchronizedCollection<E>
SynchronizedCollection<E>
UnmodifiableCollection<E>
ValuesView<K, V>
BeanContext
List<E>
AbstractList<E>
ArrayList<E>
ArrayListWrapper<T>
AttributeList
BakedArrayList
FinalArrayList<T>
FinalArrayList
FinalArrayList<T>
HeaderList
Pack<BeanT, PropT, ItemT, PackT>
RoleList
RoleUnresolvedList
new ArrayList() {...}
CheckedList<E>
CopyOnWriteArrayList<E>


main()
makeCollection()
ArrayList.add()
HashSet.add()
Stack.add()
Vector.add()
. . .
LinkedList.add()

Class Hierarchy Analysis

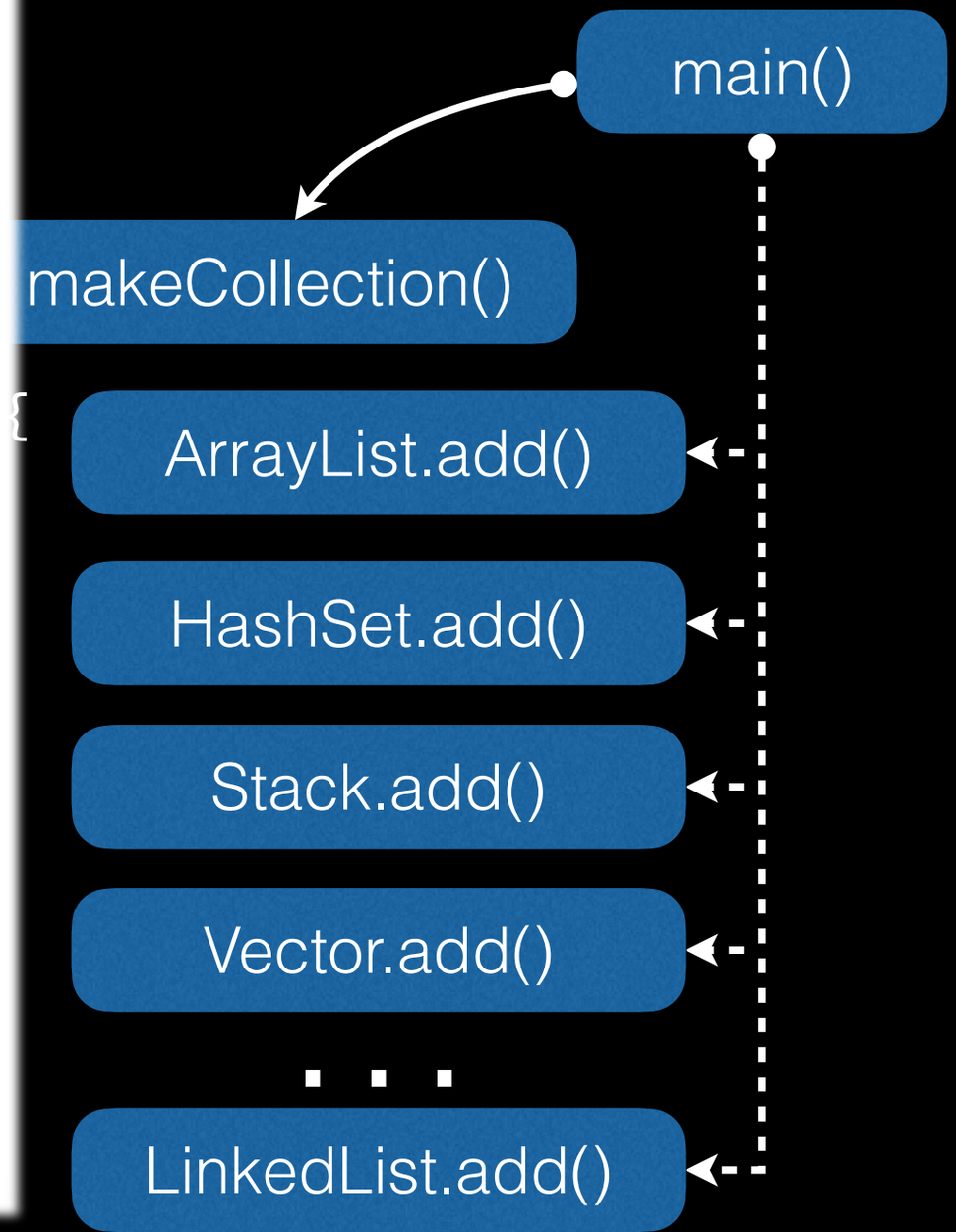
- ✅ Very simple
- ✅ Sound/correct call graph
- ✅ Pretty fast to compute
- ✅ Only input is class hierarchy

- ❌ Very imprecise (a lot of edges will never occur at runtime)

Rapid Type Analysis

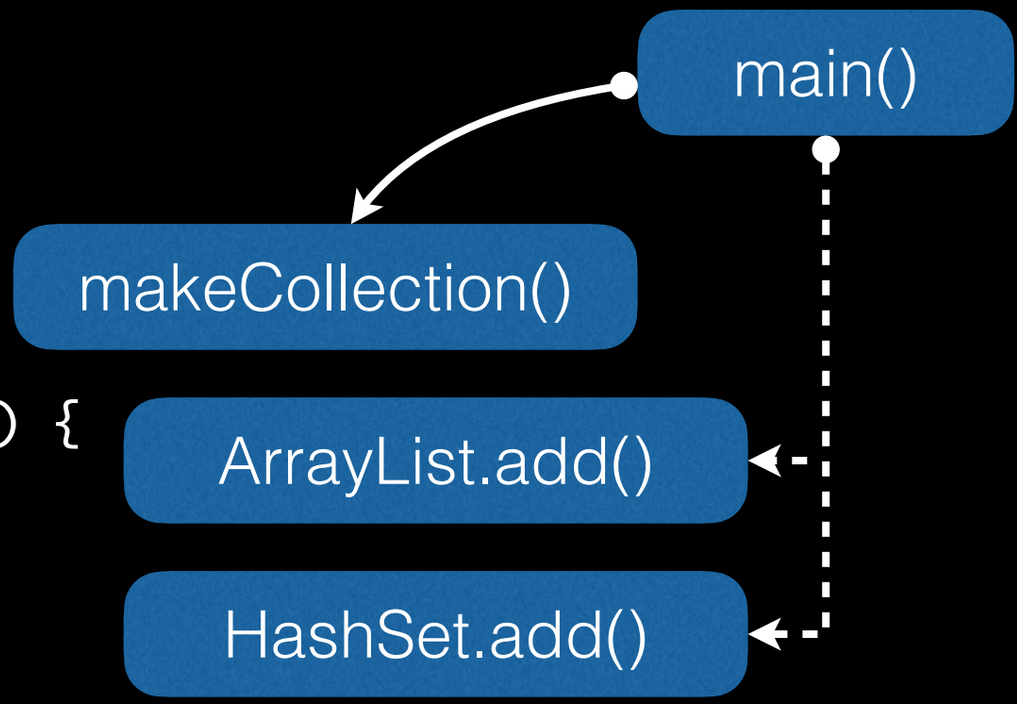
nalysis

- AbstractCollection<E>
  - AbstractList<E>
  - AbstractQueue<E>
  - AbstractSet<E>
  - ArrayDeque<E>
  - ConcurrentLinkedDeque<E>
  - Fixups
  - LinkedValues<K, V>
  - StringValues
  - ValueCollection<K, V>
  - Values<K, V>
  - Values<K, V>
  - Values<K, V>
  - Values<K, V>
  - Values<K, V>
  - Values<K, V>
  - new AbstractCollection() {...}<K, V>
- CheckedCollection<E>
- CollectionImage
- CollectionView<K, V>
- ObservableValues<K, V>
- SynchronizedCollection<E>
- SynchronizedCollection<E>
- UnmodifiableCollection<E>
- ValuesView<K, V>
- BeanContext
- List<E>
  - AbstractList<E>
  - ArrayList<E>
    - ArrayListWrapper<T>
    - AttributeList
    - BakedArrayList
    - FinalArrayList<T>
    - FinalArrayList
    - FinalArrayList<T>
    - HeaderList
    - Pack<BeanT, PropT, ItemT, PackT>
    - RoleList
    - RoleUnresolvedList
    - new ArrayList() {...}
  - CheckedList<E>
  - CopyOnWriteArrayList<E>



Rapid Type Analysis

```
public static void main(String[] args) {
  Collection c = makeCollection(args[0]);
  c.add("elem");
}

static Collection makeCollection(String s) {
  if(s.equals("list")) {
    return new ArrayList();
  } else {
    return new HashSet();
  }
}
```



David F. Bacon and Peter F. Sweeney. 1996. Fast static analysis of C++ virtual function calls. In Proceedings of the 11th ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '96), 324-341.

Rapid Type Analysis

- “Rapid”
- Still sound/correct call graph
- Call graph is much smaller than CHA
- Doesn't handle variable assignments

Variable Type Analysis

Variable Type Analysis

Main Idea: propagate **types** from allocation sites to potential **variable references**

Vijay Sundaresan, Laurie Hendren, Chrislain Razafimahefa, Raja Vallée-Rai, Patrick Lam, Etienne Gagnon, and Charles Godin. 2000. Practical virtual method call resolution for Java. In Proceedings of the 15th ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '00), USA, 264-280.

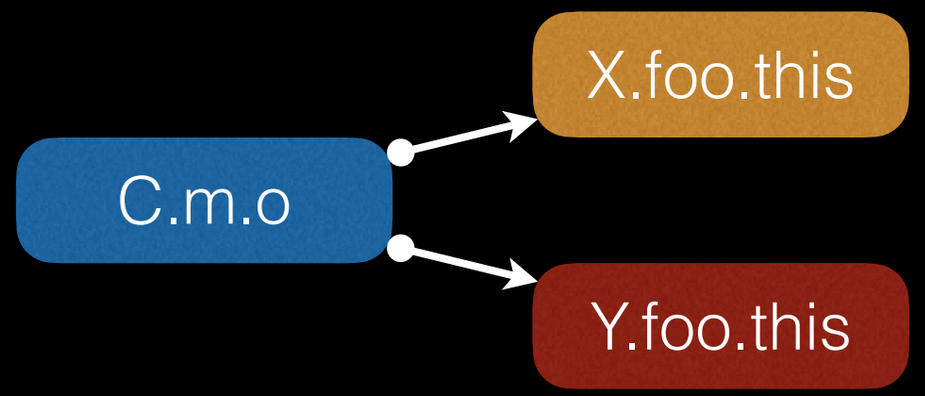
Variable Type Analysis

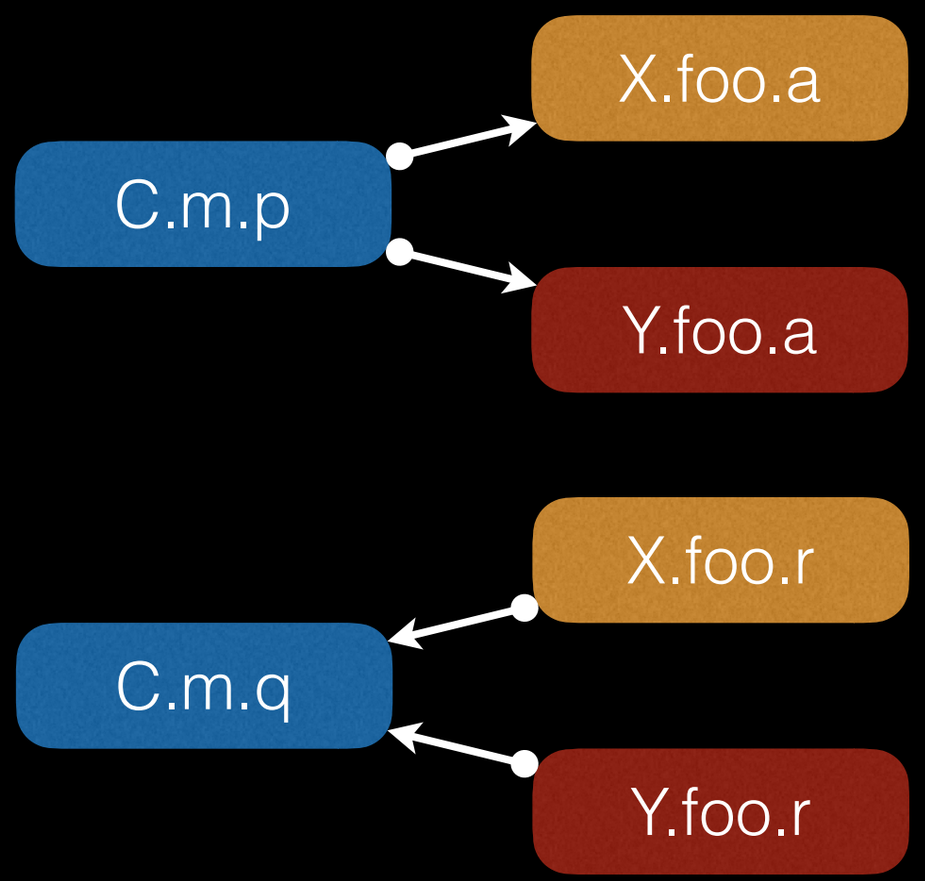
1. Start with a pre-computed call graph (e.g., CHA)
2. Build type propagation graph
3. Collapse strongly-connected components (SCCs)
4. Propagate types along the final Directed Acyclic Graph (DAG)

VTA - Step #1 … but why?

```
class X {
  D foo (A a){
    ...
    return(r);
  }
}

class Y {
  D foo (A a){
    ...
    return(r);
  }
}
```

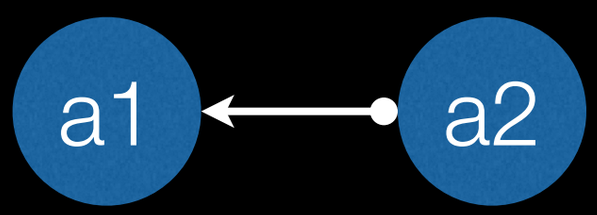
```
class C {
  E m() {
    ...
    q = o.foo(p);
  }
}
```





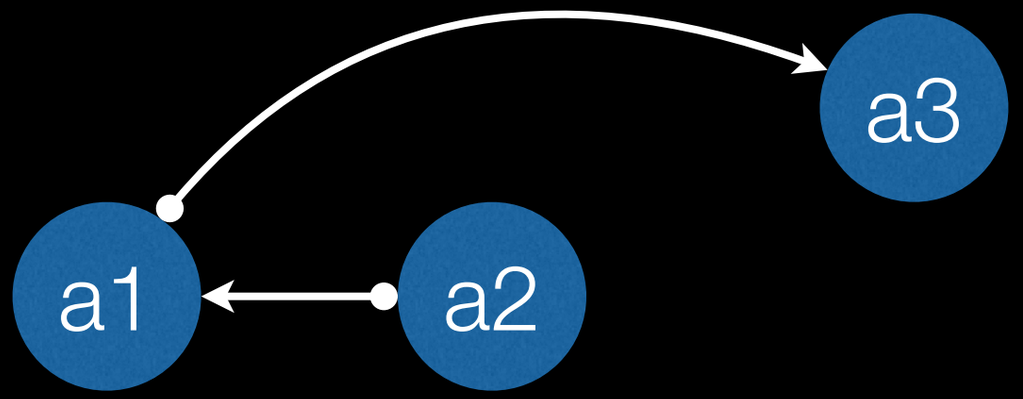
VTA - Step #2

```
A a1, a2, a3;
B b1, b3;
C c;


a1 = new A();
a2 = new A();
b1 = new B();
b3 = new B();
c = new C();

a1 = a2;
a3 = a1;
a3 = b3;
b3 = (B) a3;
b1 = c;
```



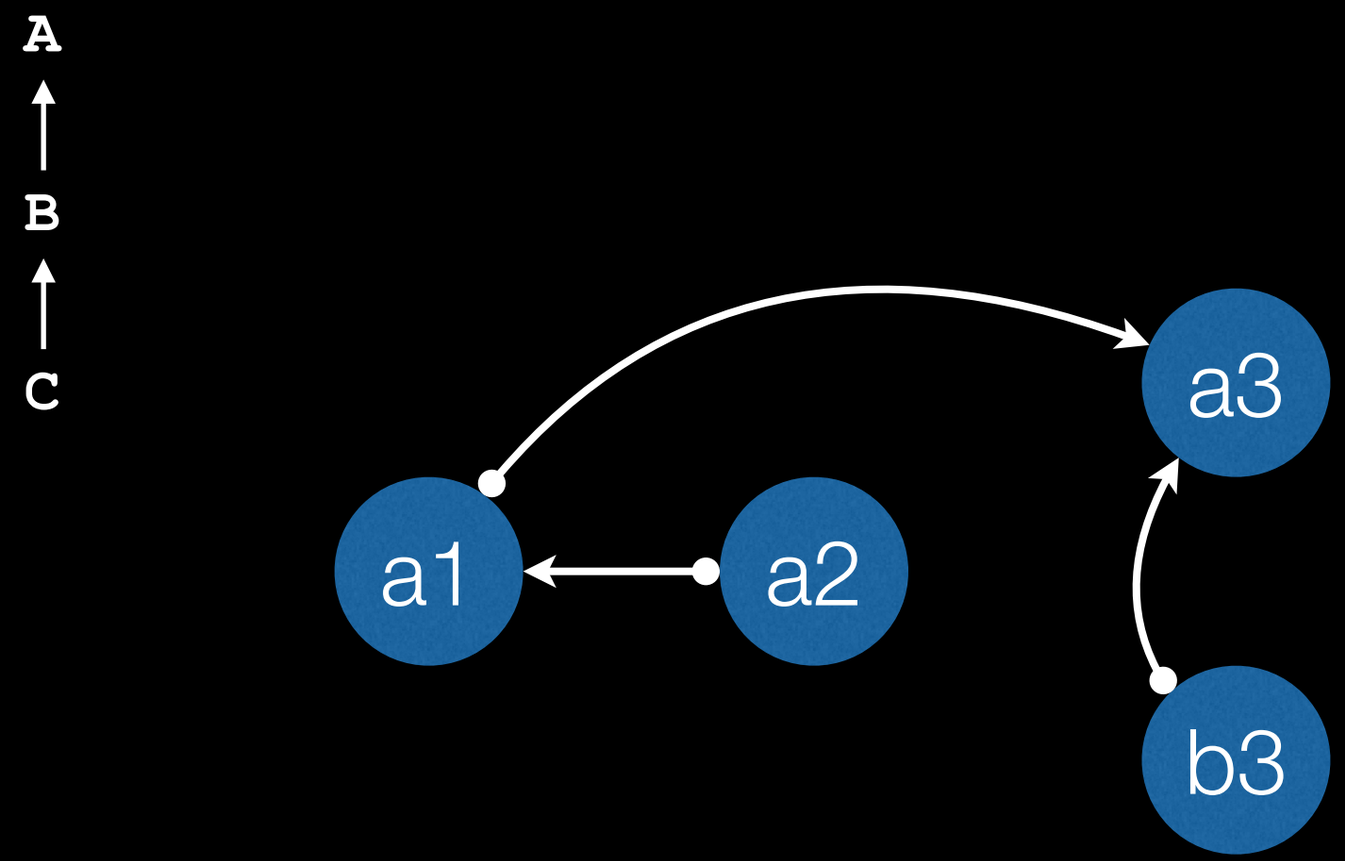
VTA - Step #2

```
A a1, a2, a3;
B b1, b3;
C c;

a1 = new A();
a2 = new A();
b1 = new B();
b3 = new B();
c = new C();

a1 = a2;
a3 = a1;
a3 = b3;
b3 = (B) a3;
b1 = c;
```



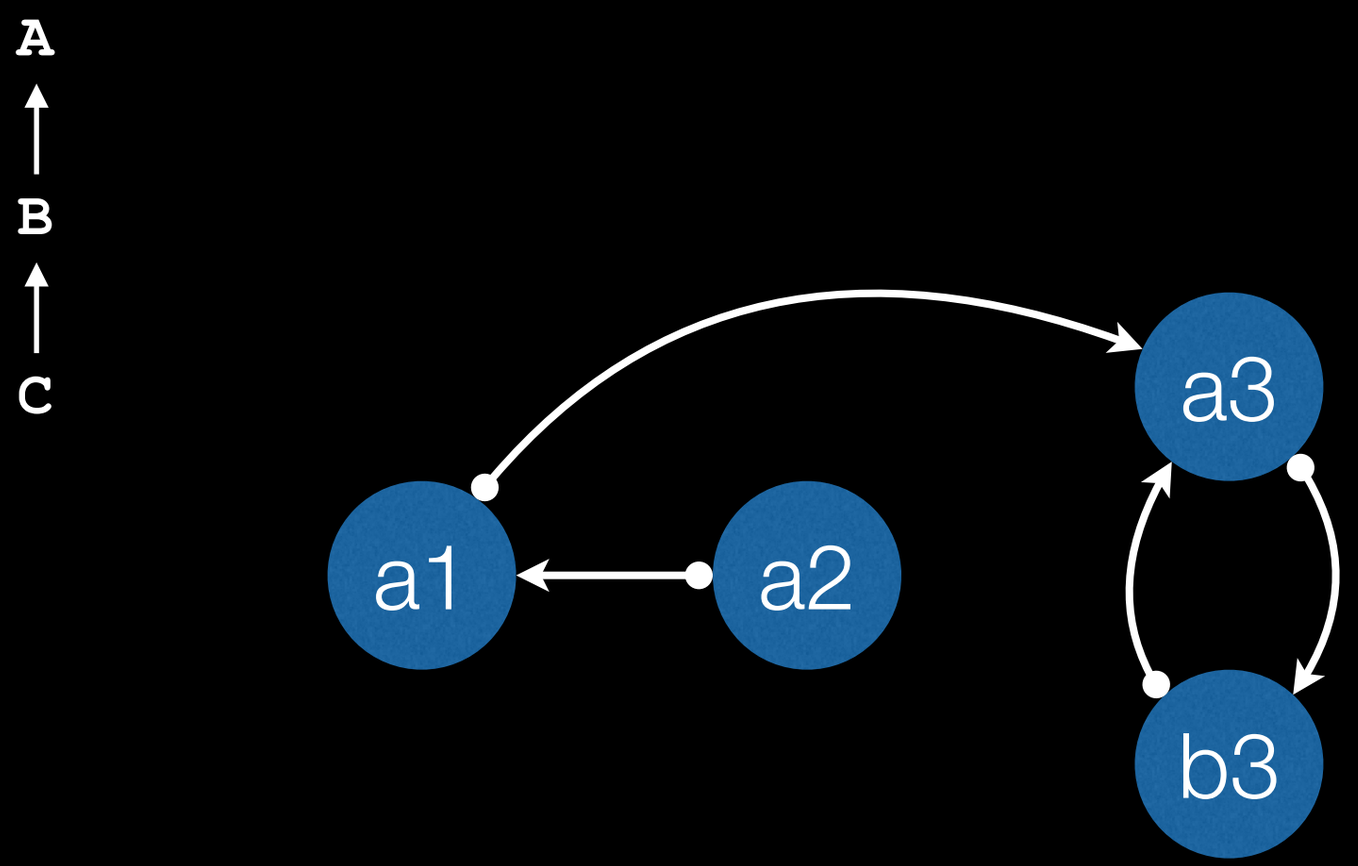
VTA - Step #2

```
A a1, a2, a3;
B b1, b3;
C c;

a1 = new A();
a2 = new A();
b1 = new B();
b3 = new B();
c = new C();

a1 = a2;
a3 = a1;
a3 = b3;
b3 = (B) a3;
b1 = c;
```



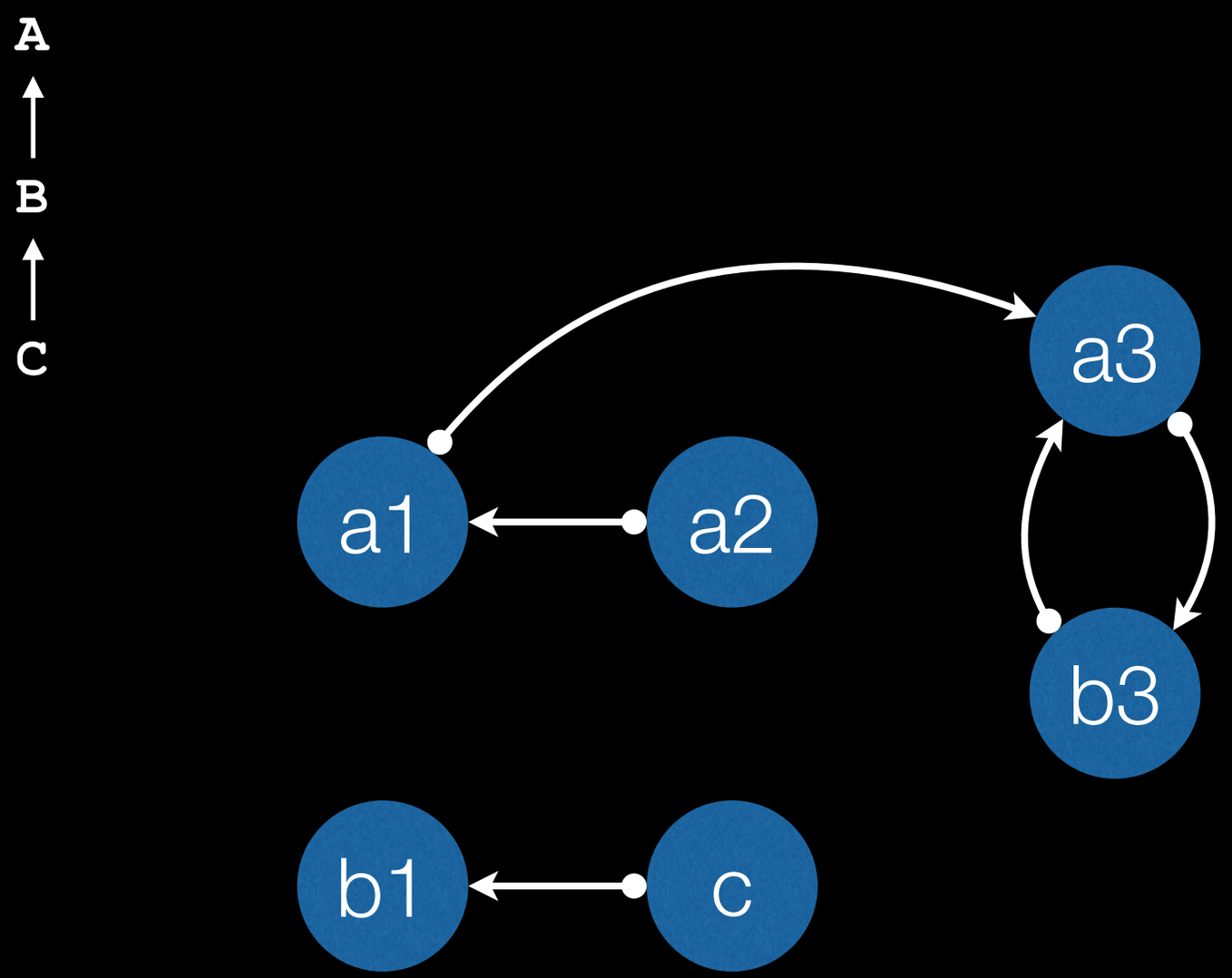
VTA - Step #2

```
A a1, a2, a3;
B b1, b3;
C c;

a1 = new A();
a2 = new A();
b1 = new B();
b3 = new B();
c = new C();

a1 = a2;
a3 = a1;
a3 = b3;
b3 = (B) a3;
b1 = c;
```



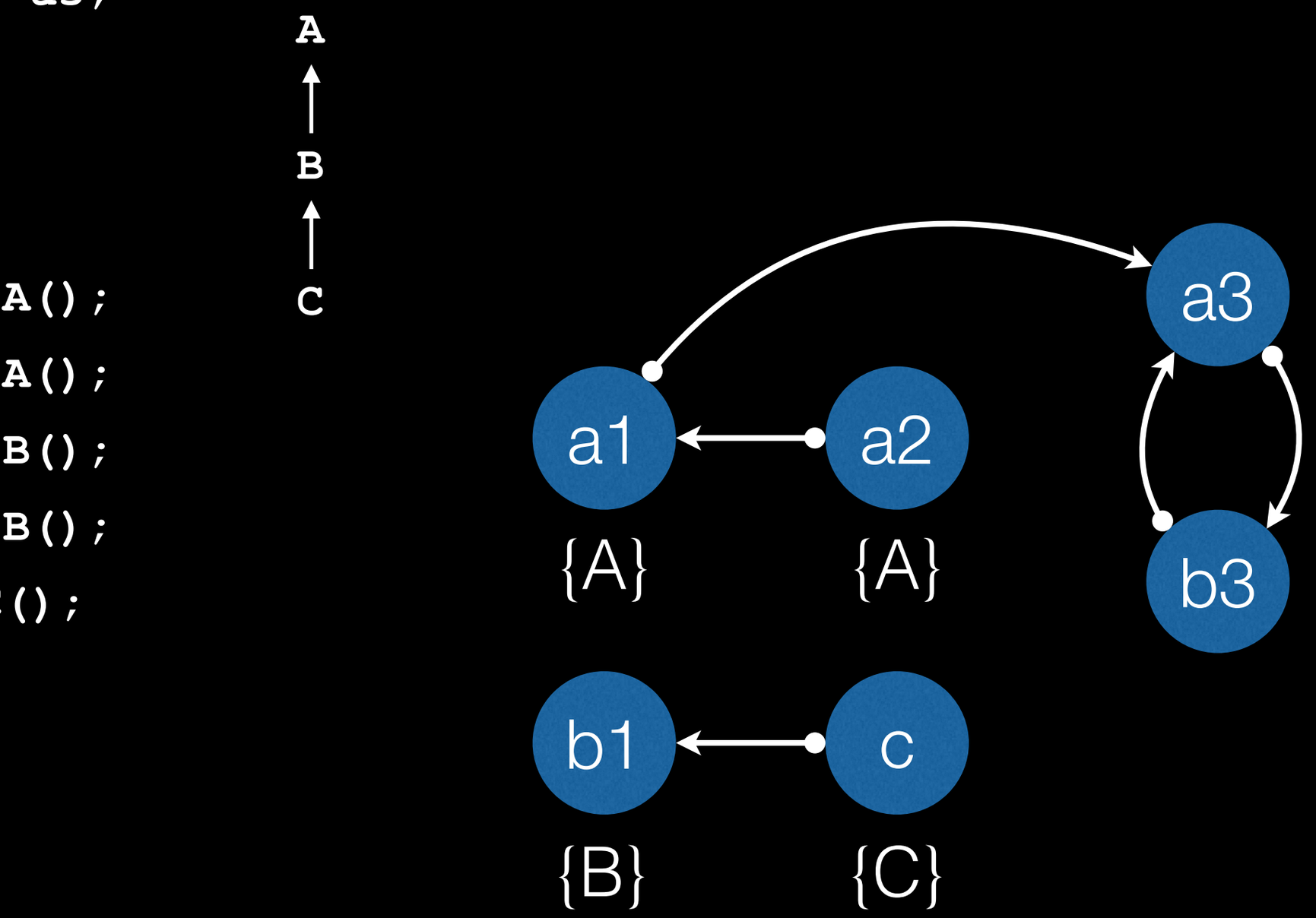
VTA - Step #2

```
A a1, a2, a3;
B b1, b3;
C c;

a1 = new A();
a2 = new A();
b1 = new B();
b3 = new B();
c = new C();

a1 = a2;
a3 = a1;
a3 = b3;
b3 = (B) a3;
b1 = c;
```



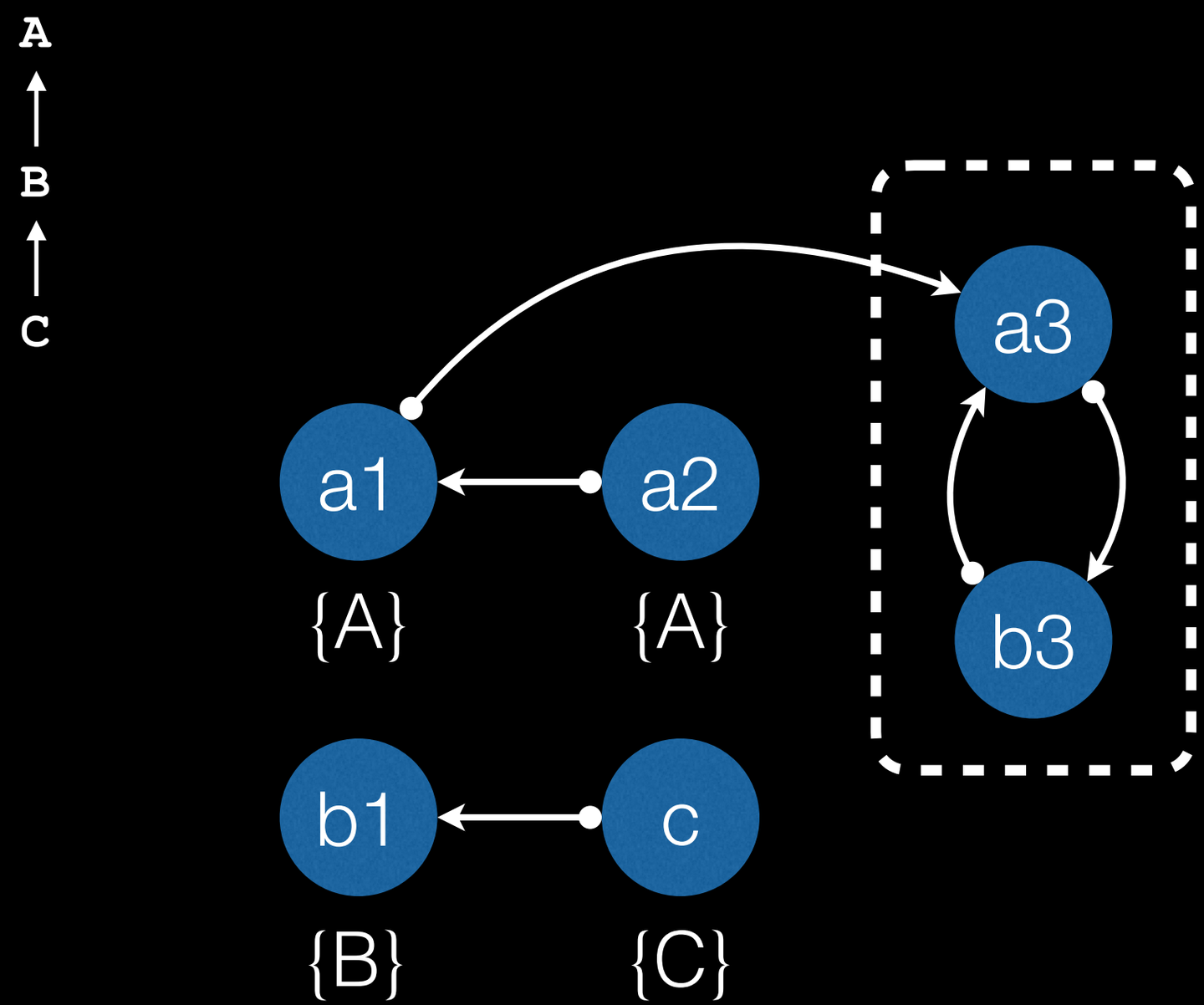
VTA - Step #2

```
A a1, a2, a3;
B b1, b3;
C c;

a1 = new A();
a2 = new A();
b1 = new B();
b3 = new B();
c = new C();

a1 = a2;
a3 = a1;
a3 = b3;
b3 = (B) a3;
b1 = c;
```



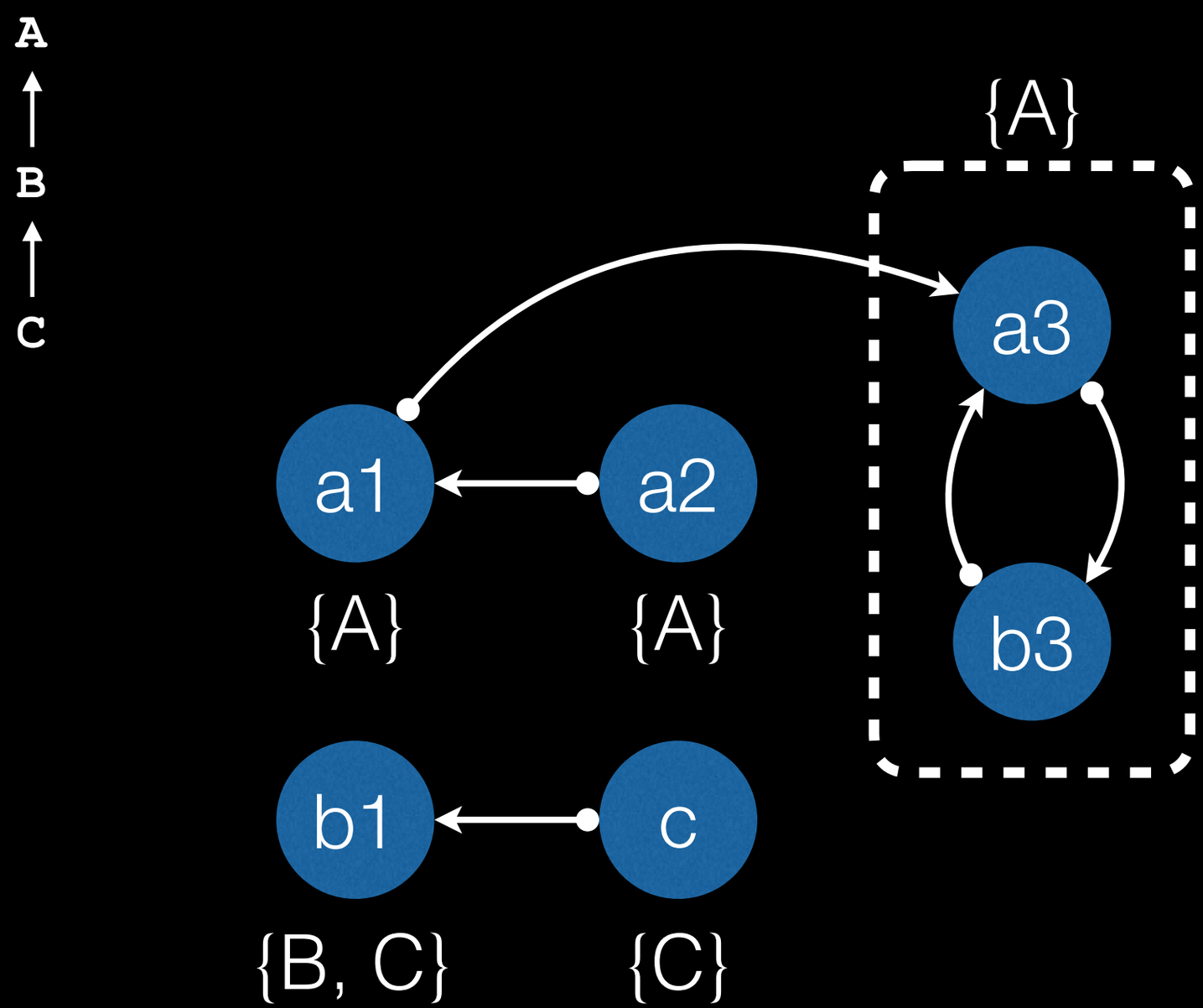
VTA - Step #3

```
A a1, a2, a3;
B b1, b3;
C c;

a1 = new A();
a2 = new A();
b1 = new B();
b3 = new B();
c = new C();

a1 = a2;
a3 = a1;
a3 = b3;
b3 = (B) a3;
b1 = c;
```



VTA - Step #4

```
A a1, a2, a3;
B b1, b3;
C c;

a1 = new A();
a2 = new A();
b1 = new B();
b3 = new B();
c = new C();

a1 = a2;
a3 = a1;
a3 = b3;
b3 = (B) a3;
b1 = c;
```



Variable Type Analysis

- ✅ More precise than RTA
- ✅ Relatively fast
- ❌ Requires an initial CG
- ❌ Field-based

SPARK

SPARK

Main Idea: propagate **information** along edges of **pointer assignment graph (PAG)**

Ondřej Lhoták and Laurie Hendren. 2003. Scaling Java points-to analysis using SPARK. In *Proceedings of the 12th international conference on Compiler Construction* (CC'03), 153-169.

SPARK - PAG

Nodes

Alloc new A

SPARK - PAG

Nodes

Alloc new A

Var p

SPARK - PAG

Nodes

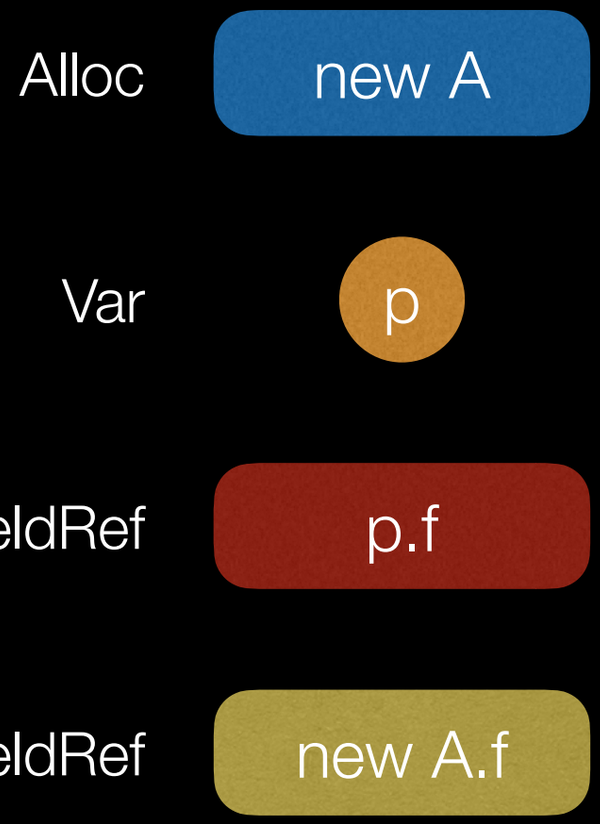
Alloc new A

Var p

FieldRef p.f

SPARK - PAG

Nodes



SPARK - PAG

Nodes

Alloc new A

Var p

FieldRef p.f

ConcreteFieldRef new A.f

Edges

SPARK - PAG

Nodes

Alloc new A

Var p

FieldRef p.f

ConcreteFieldRef new A.f

Edges

Alloc new A → p

Assign q → p

SPARK - PAG

Nodes

Alloc new A

Var p

FieldRef p.f

ConcreteFieldRef new A.f

Edges

Alloc new A → p

Assign q → p

Store q → p.f

SPARK - PAG

Nodes

Alloc new A

Var p

FieldRef p.f

ConcreteFieldRef new A.f

Edges

Alloc new A → p

Assign q → p

Store q → p.f

Load q.f → p

PAG - AllocNode

- Models allocations sites
- For each `new T` statement
- Has a type
- `AnyType` means type is unknown

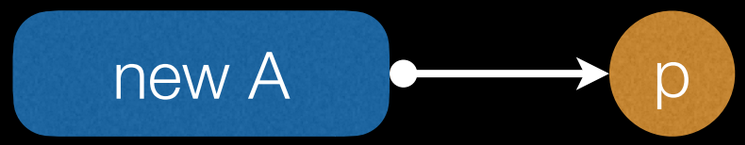
PAG - VarNode

- Represents a variable holding pointers to objects
- Local variables, method parameters, throwables
- Might have a type

PAG - FieldRefNode

- Represents a pointer dereference
- Base is a VarNode
- a.<elements> to model array contents
- Might have a type

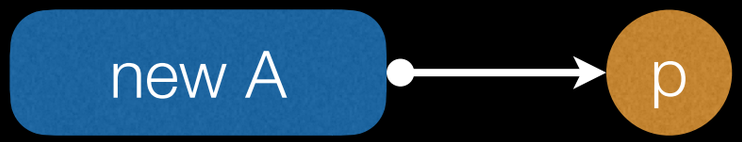
PAG - AllocEdge



- Models the statement `p = new A`
- E.g., `hm = new HashMap`

Allocation is independent of calling the constructor

PAG - AllocEdge



- Models the statement `p = new A`
- E.g., `hm = new HashMap`
- Induces the constraint

  pts-to(new A) ⊆ pts-to(p)

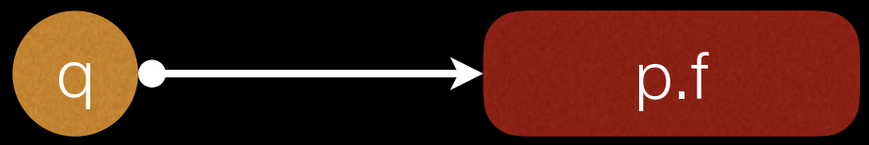
Why subset?

PAG - AssignEdge

- Models the assignment `p = q`
- Induces the constraint

pts-to(q) ⊆ pts-to(p)

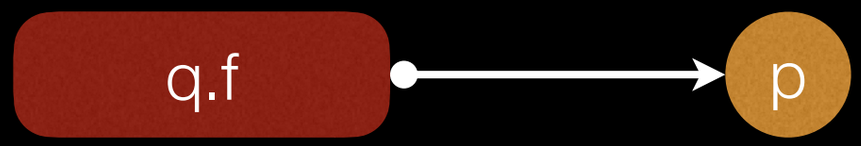
PAG - StoreEdge



- Models the assignment `p.f = q`
- Induces the constraint

  pts-to(q) ⊆ pts-to(p.f)

PAG - LoadEdge



- Models the assignment `p = q.f`
- Induces the constraint

pts-to(q.f) ⊆ pts-to(p)

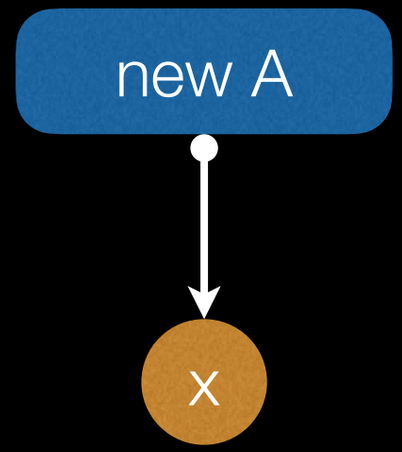
Let’s build a PAG!

PAG - Example

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```

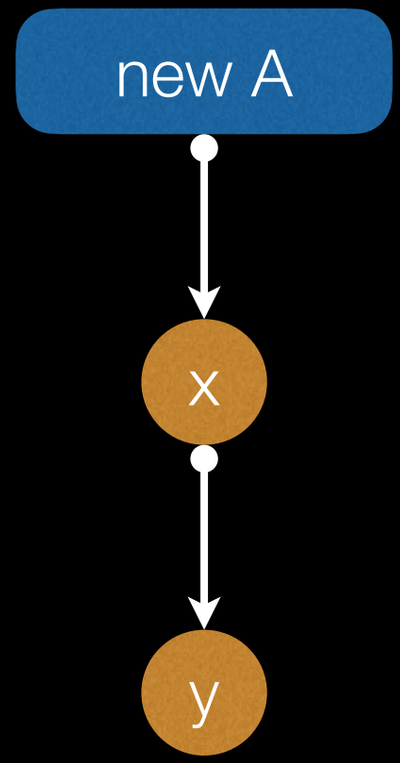
PAG - Example

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```



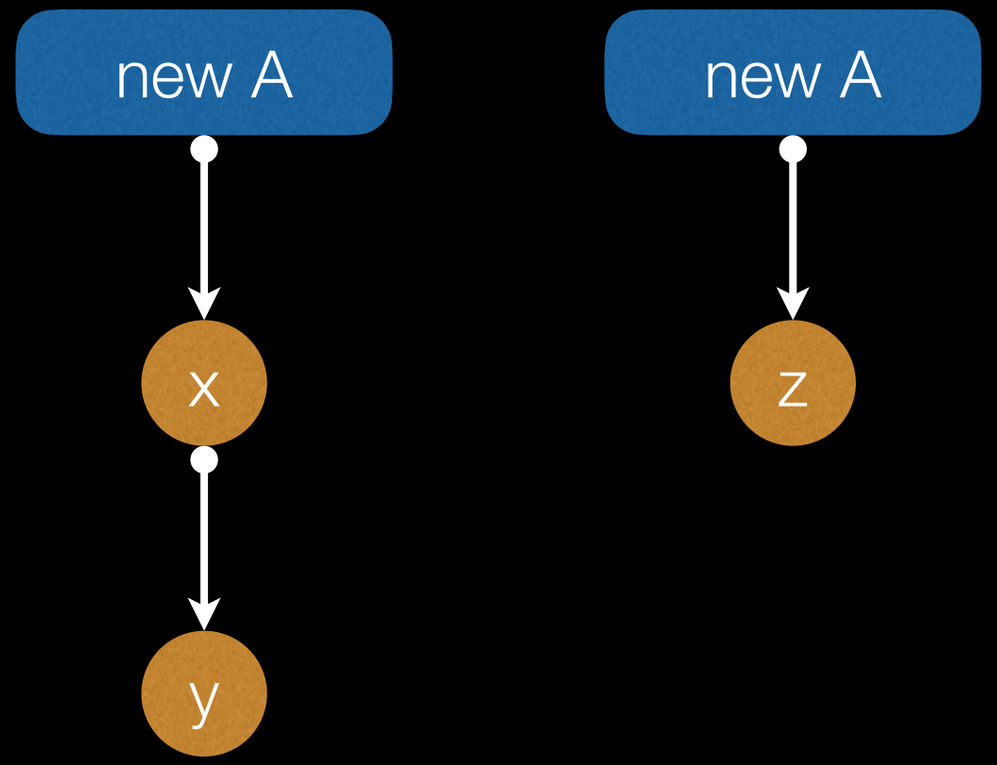
PAG - Example

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```



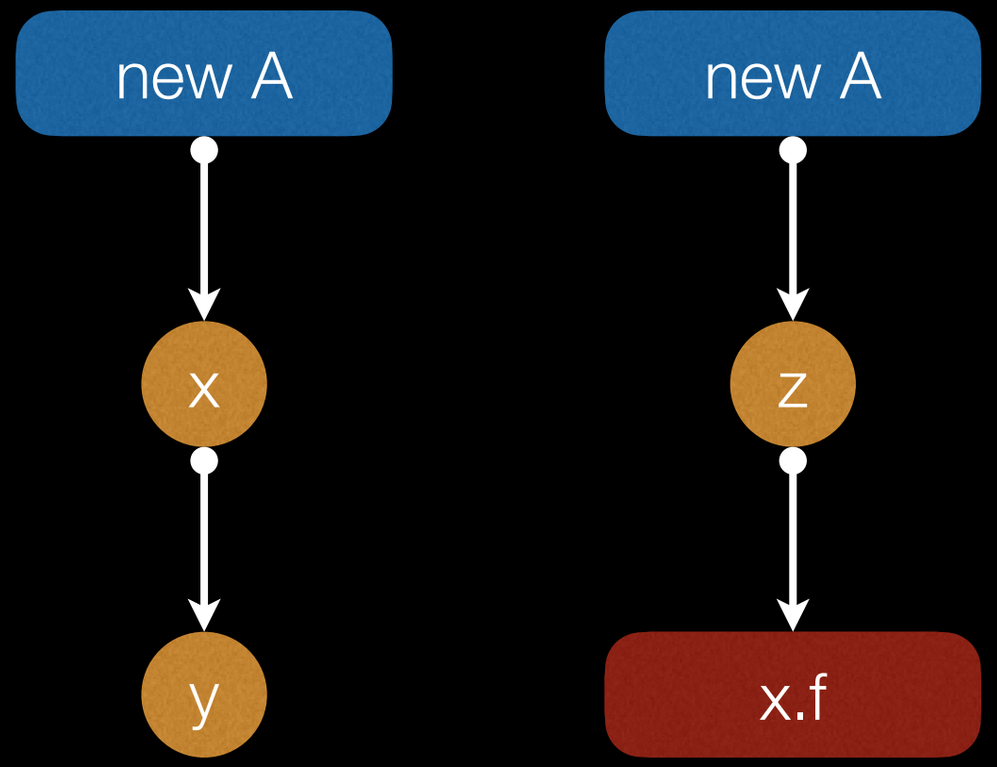
PAG - Example

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```



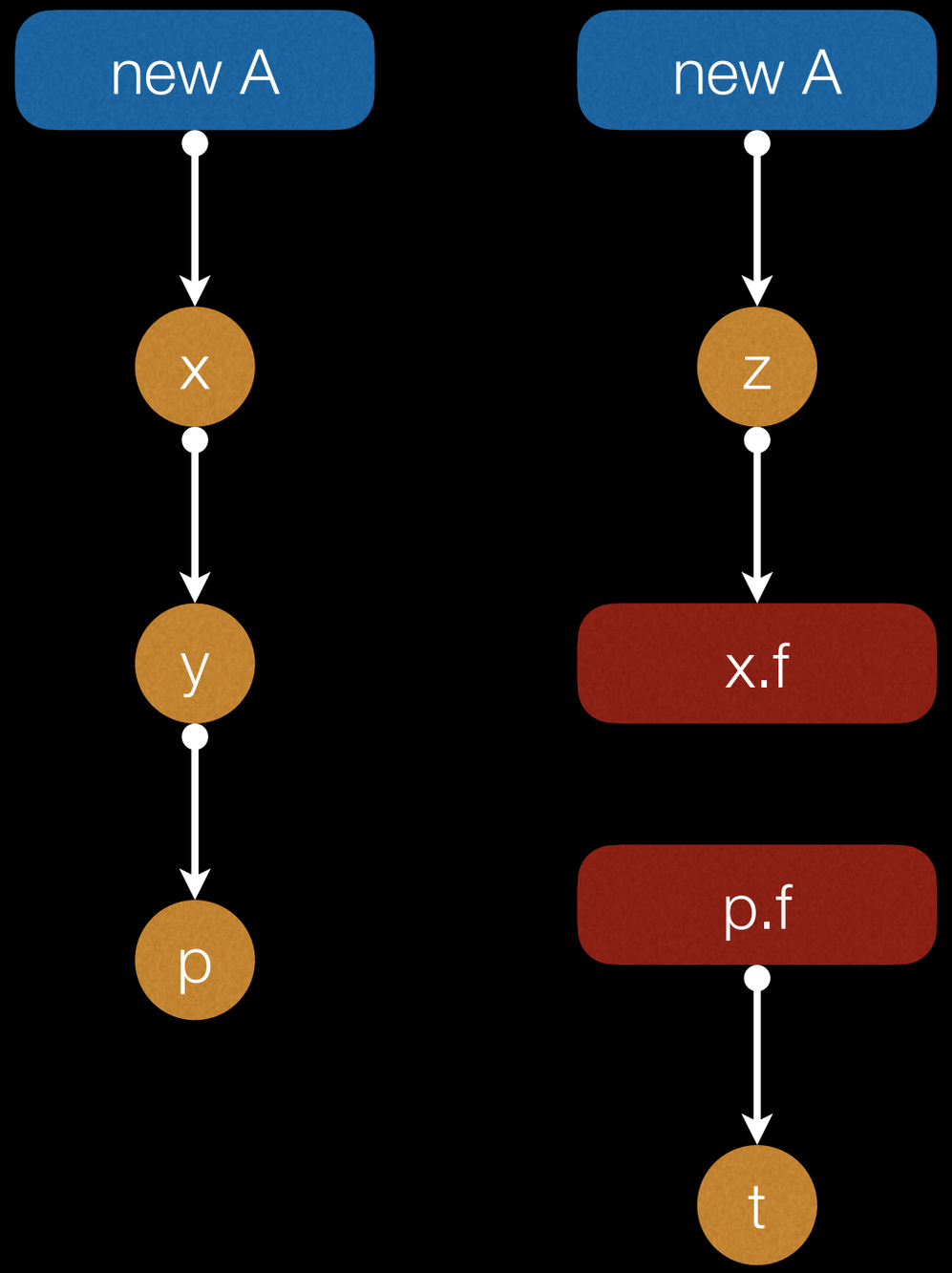
PAG - Example

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```



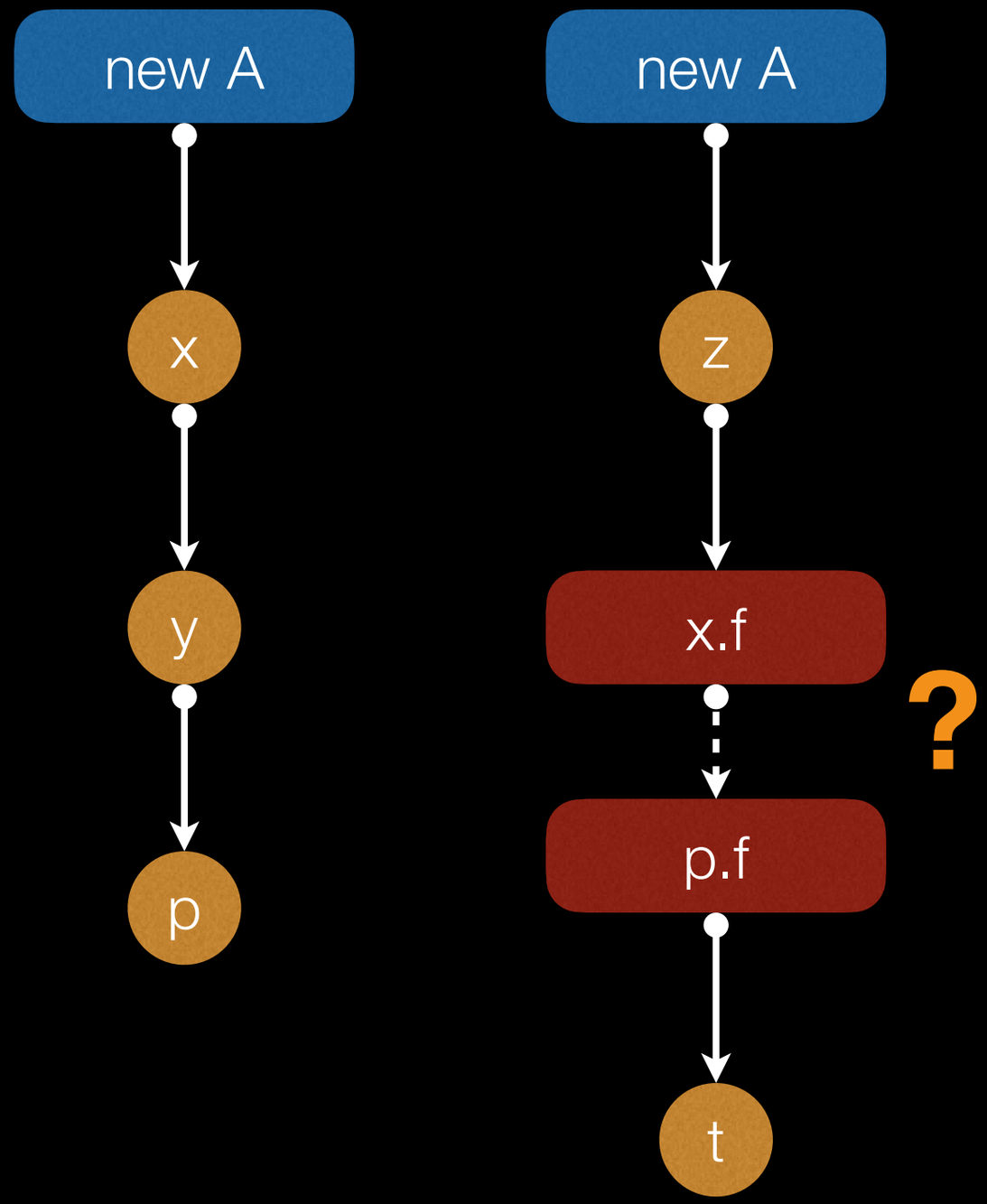
PAG - Example

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```



PAG - Example

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```



PAG - Fields

a.f

a.*

A.f

Field-Sensitive

Field-Insensitive

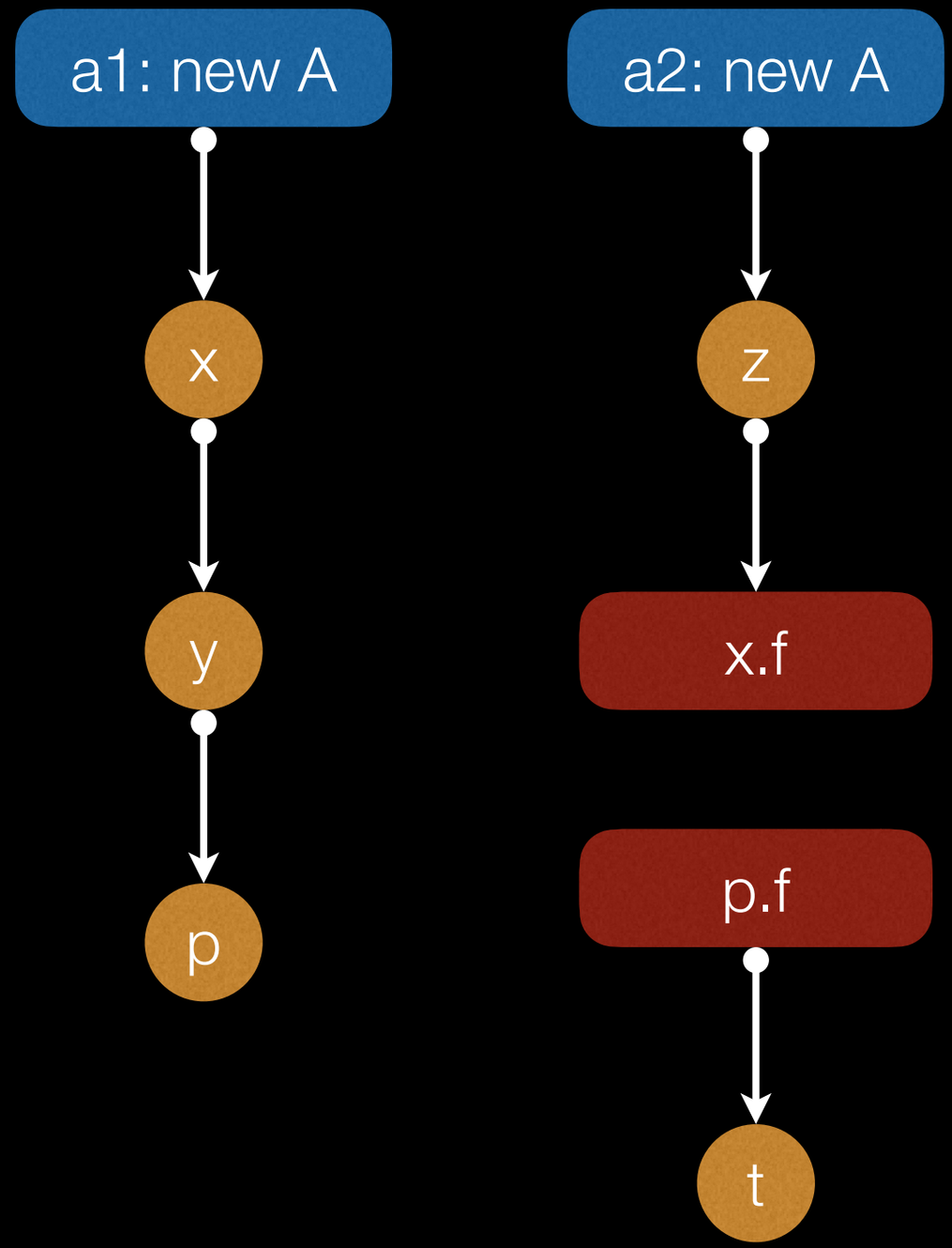
Field-Based

PAG - Other Considerations

- Static initializers
- Object.finalize()
- Thread.start()
- Reflection
- Object sensitivity

SPARK
Points-to Propagation

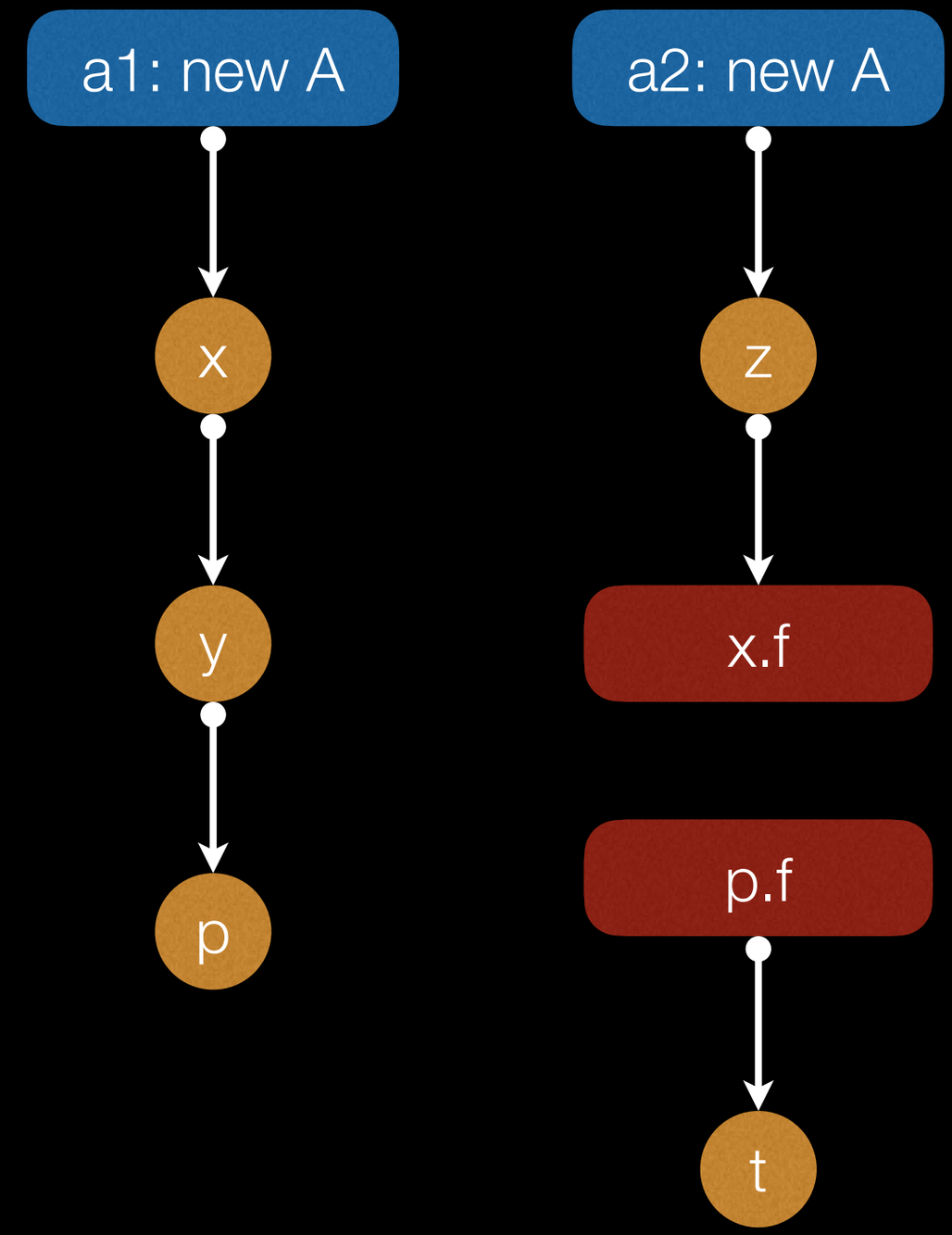
```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```



SPARK
Points-to Propagation

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```

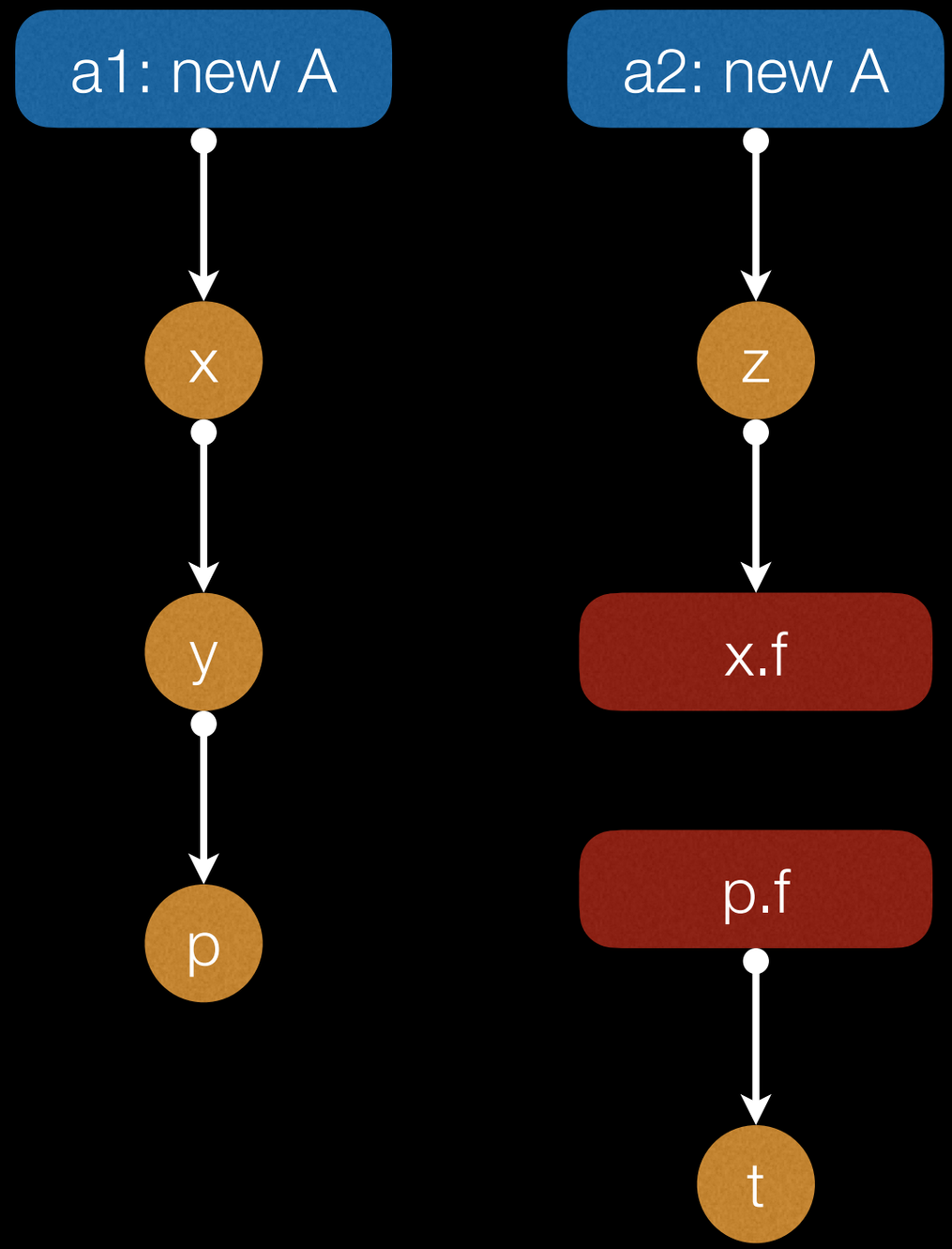
pts-to(x) = {a1}



SPARK
Points-to Propagation

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```

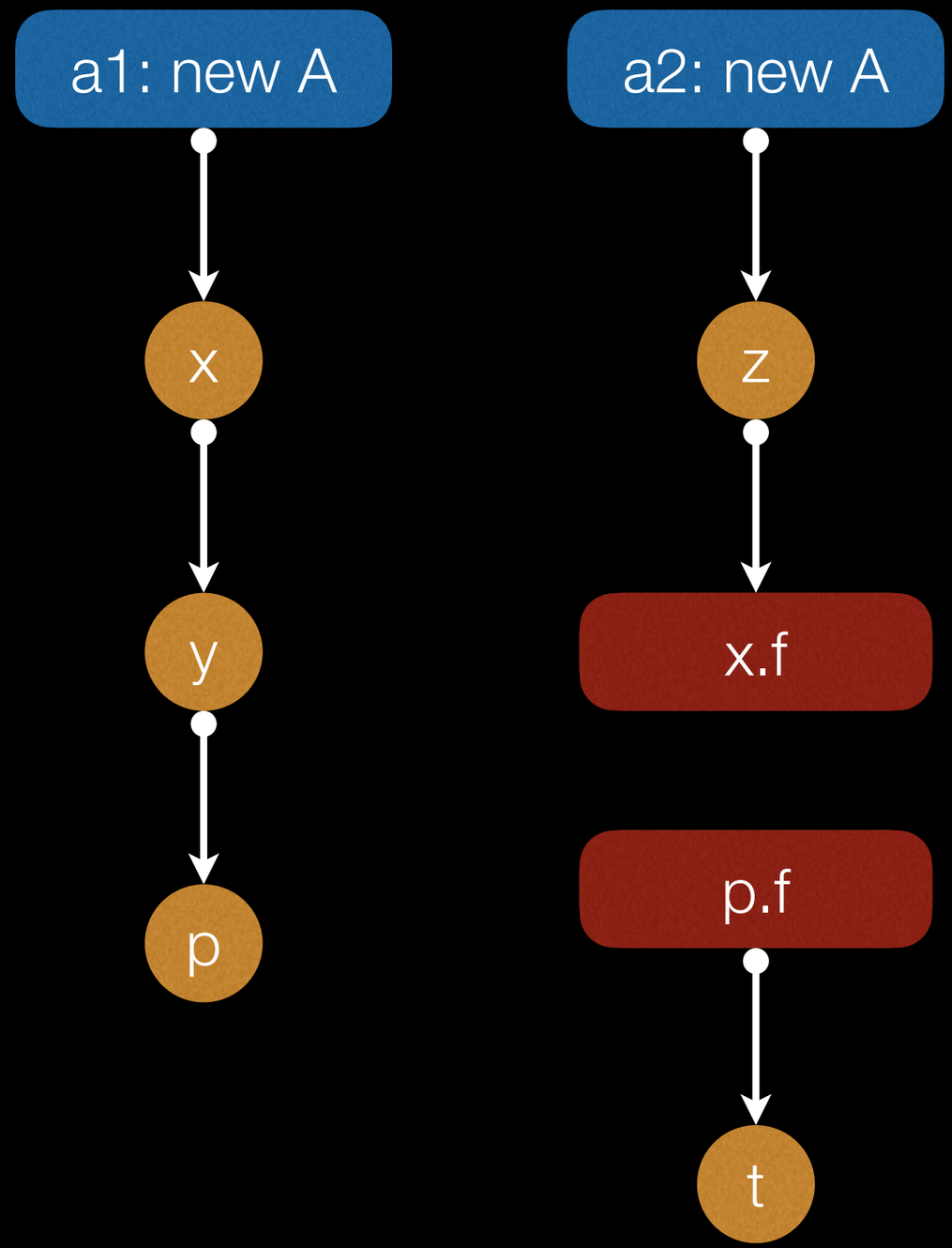
```
pts-to(x) = {a1}
pts-to(y) = {a1}
```



SPARK
Points-to Propagation

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```

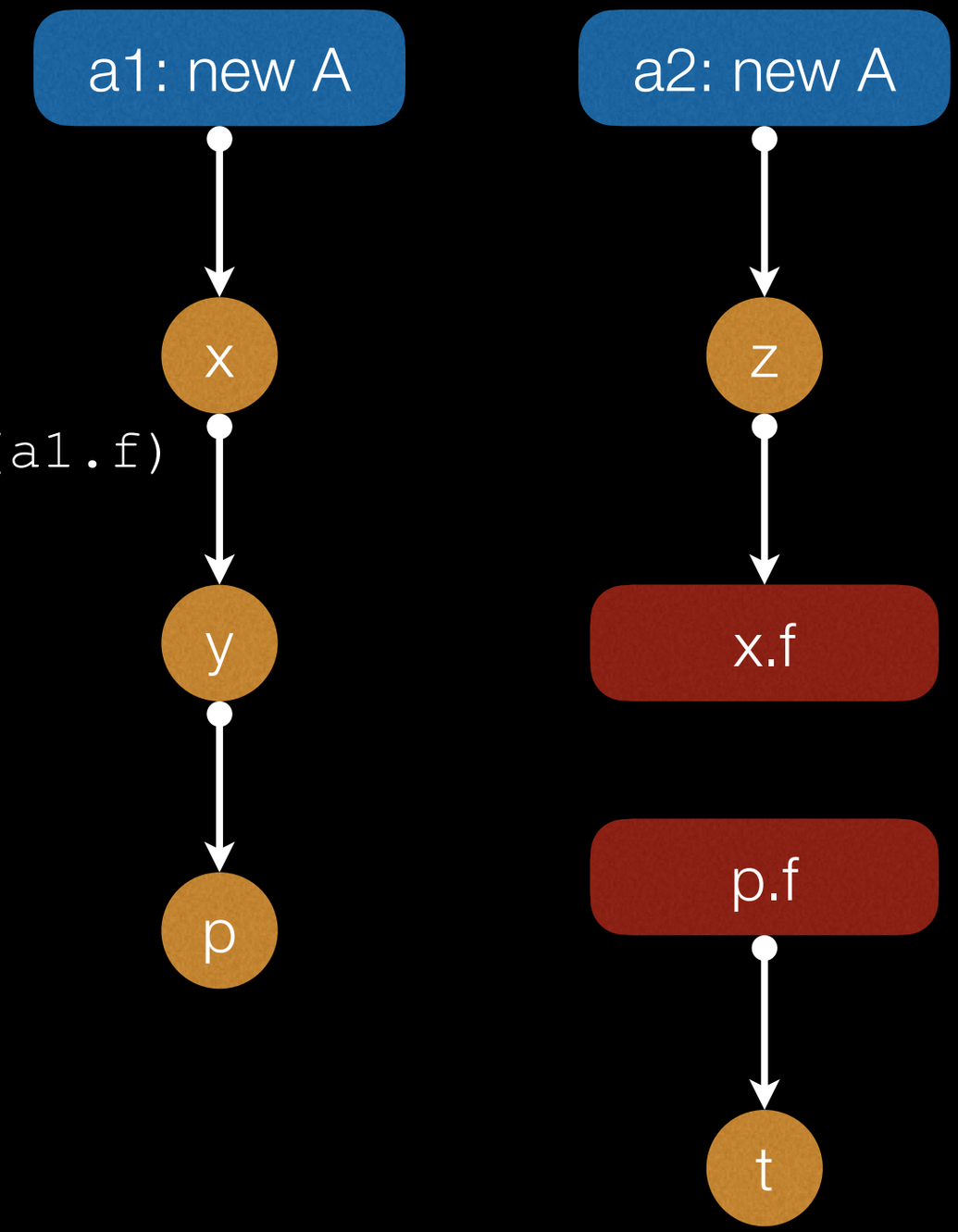
```
pts-to(x) = {a1}
pts-to(y) = {a1}
pts-to(z) = {a2}
```



SPARK
Points-to Propagation

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```

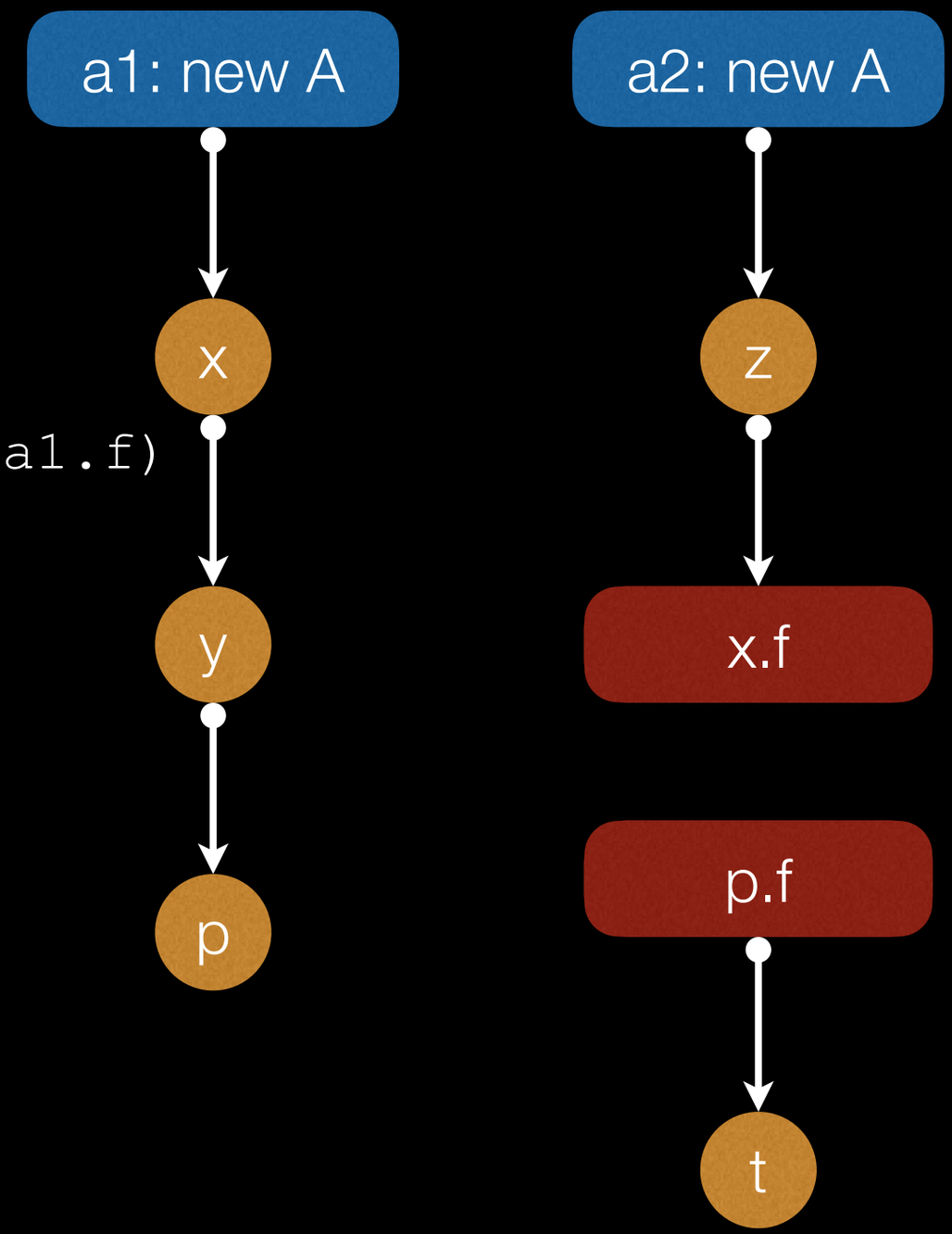
```
pts-to(x) = {a1}
pts-to(y) = {a1}
pts-to(z) = {a2}
pts-to(x.f) = pts-to(a1.f)
            = {a2}
```



SPARK
Points-to Propagation

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```

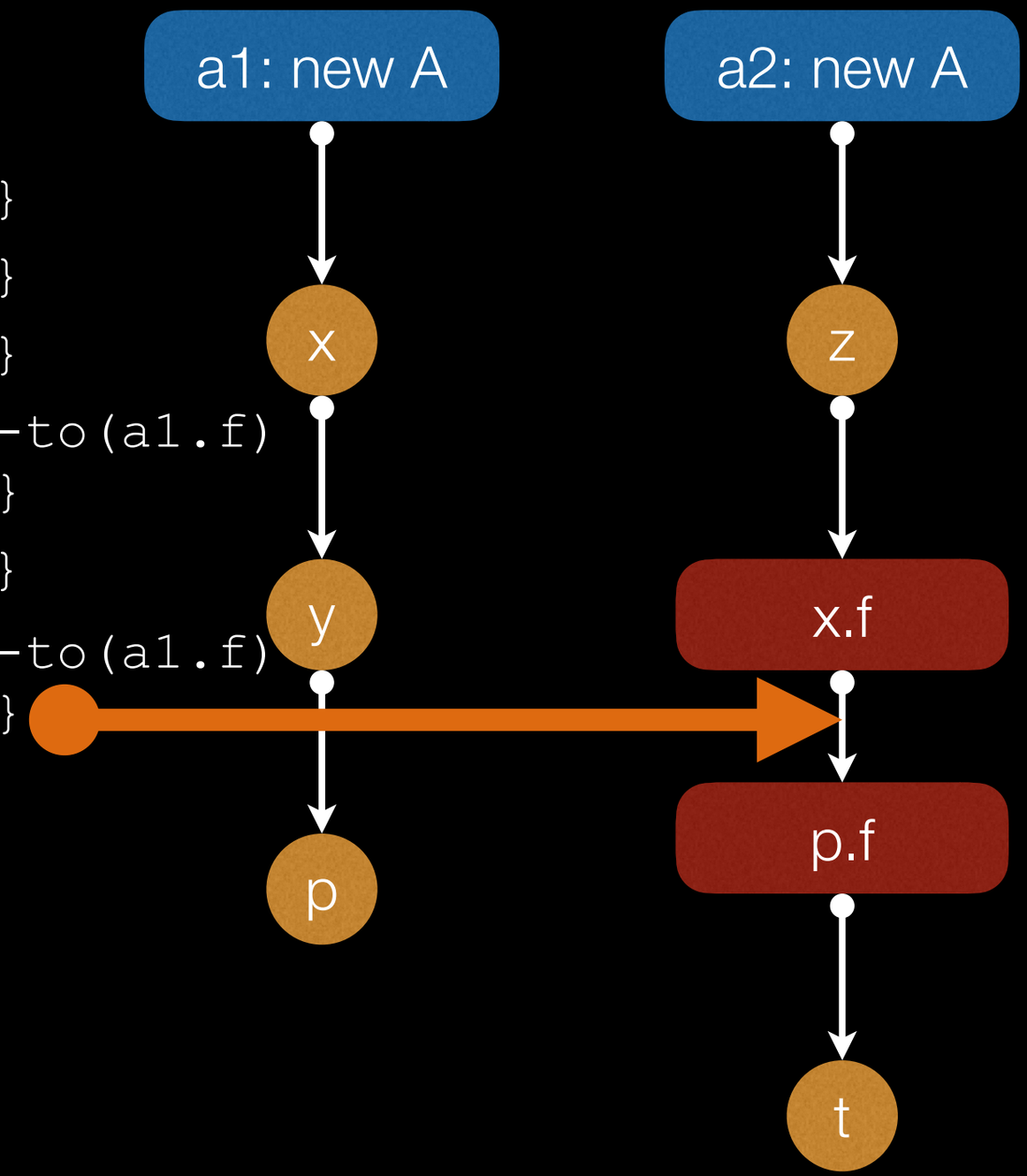
```
pts-to(x) = {a1}
pts-to(y) = {a1}
pts-to(z) = {a2}
pts-to(x.f) = pts-to(a1.f)
            = {a2}
pts-to(p) = {a1}
```



SPARK
Points-to Propagation

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```

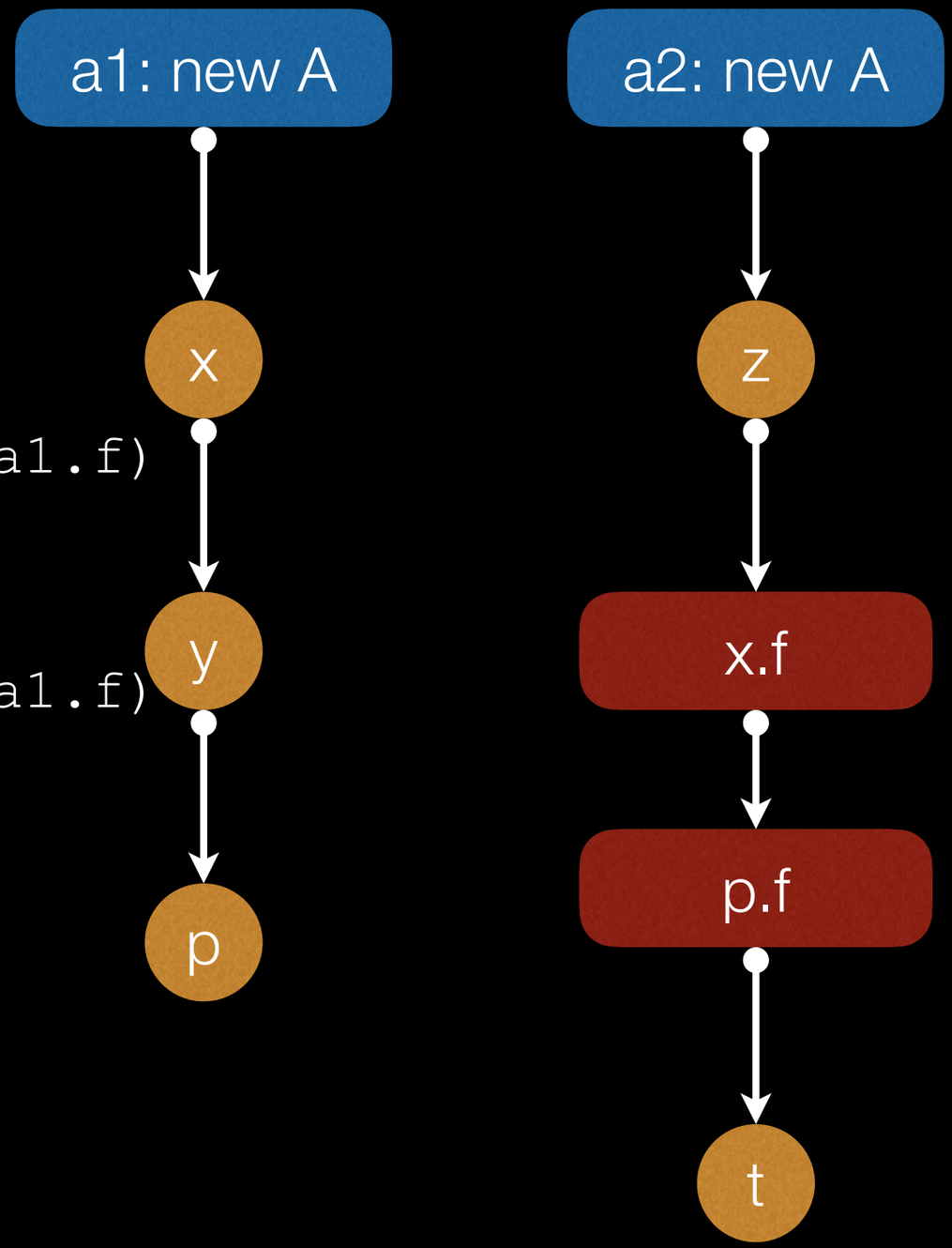
```
pts-to(x) = {a1}
pts-to(y) = {a1}
pts-to(z) = {a2}
pts-to(x.f) = pts-to(a1.f)
            = {a2}
  pts-to(p) = {a1}
pts-to(p.f) = pts-to(a1.f)
            = {a2}
```



SPARK
Points-to Propagation

```
void foo {
  x = new A();
  y = x;
  z = new A();
  x.f = z;
  t = bar(y);
}

A bar(A p) {
  return p.f;
}
```

```
pts-to(x) = {a1}
pts-to(y) = {a1}
pts-to(z) = {a2}
pts-to(x.f) = pts-to(a1.f)
            = {a2}
  pts-to(p) = {a1}
pts-to(p.f) = pts-to(a1.f)
            = {a2}

  pts-to(t) = {a2}
```



SPARK

- ✅ Very precise
- ✅ Highly customizable
- ✅ No initial call graph
- ❌ Flow-insensitive
- ❌ Quite expensive
- ❌ Large PAGs

On-the-fly CG Construction

Determining Target Calls

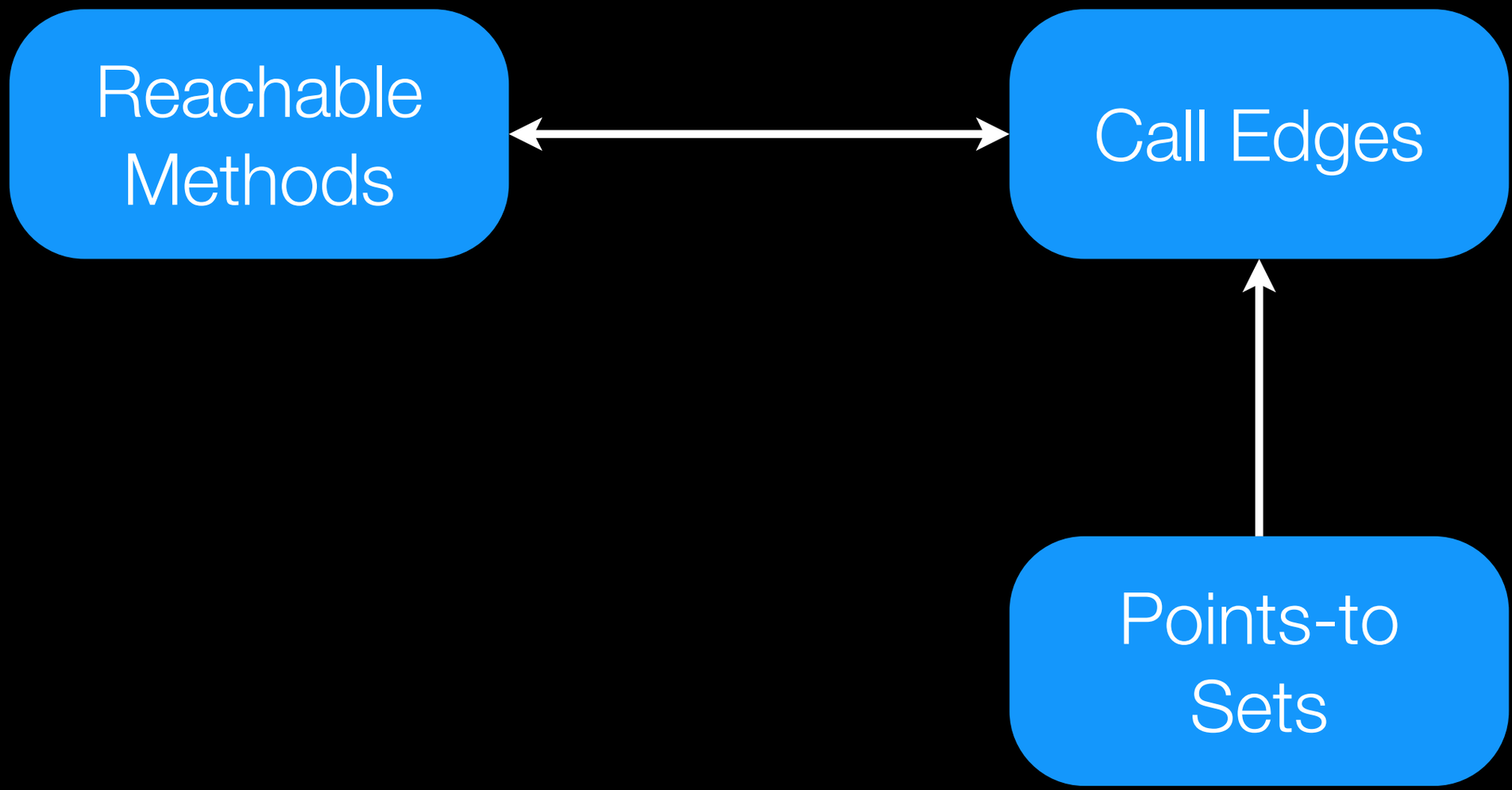
Points-to Analysis

On-the-fly CG Construction

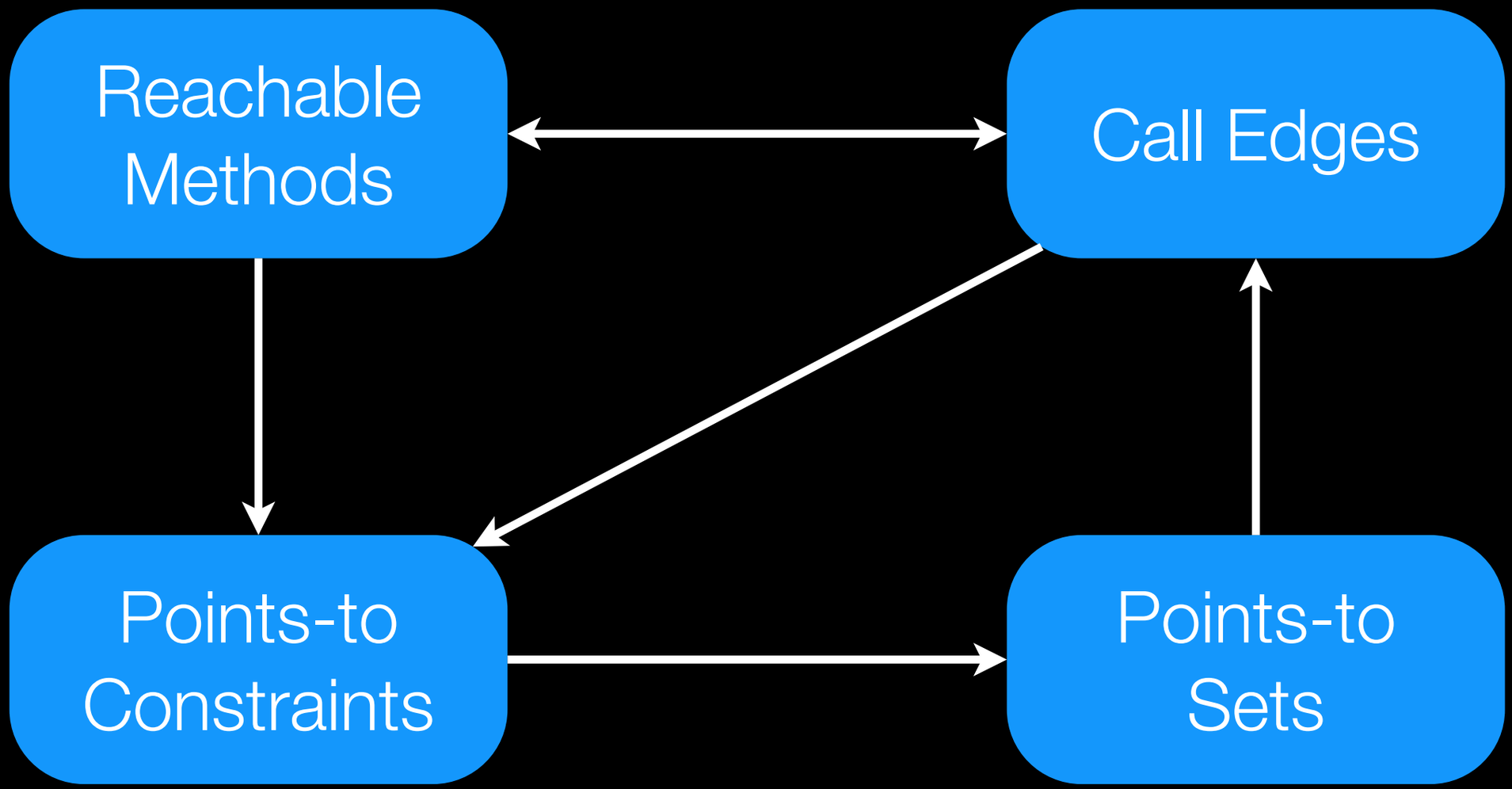
Call Edges

On-the-fly CG Construction

On-the-fly CG Construction



On-the-fly CG Construction



OTF CG Construction Algorithm

- Add entry points (e.g., main method) to the work-list and the list of reachable methods
- **Repeat until work-list is empty**
  - Pick a reachable method from the work-list
  - Find allocations sites in that method
  - Propagate those allocations along PAG edges
  - Re-resolve relevant calls in reachable methods using the updated pts-to sets
  - Add any newly-reachable methods to the work-list, and to the list of reachable methods

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```

Main.main()

Worklist

Main.main()

Reachable

Main.main()

Points-to

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```

Main.main()

Worklist

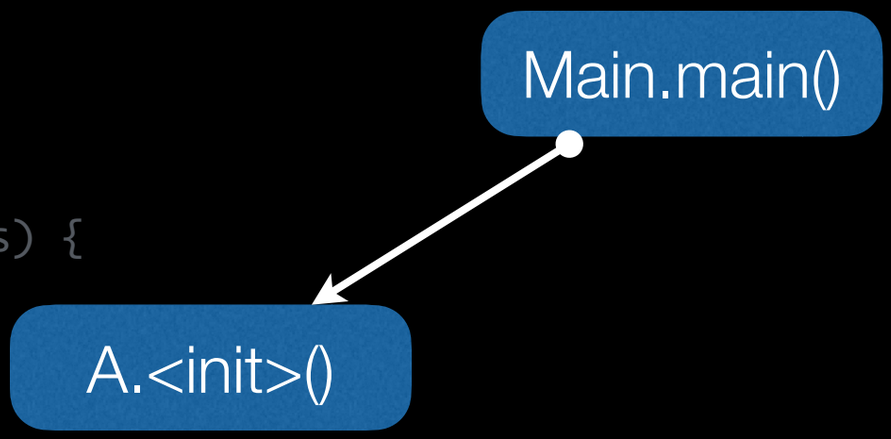
Reachable

Main.main()

Points-to

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```



Worklist

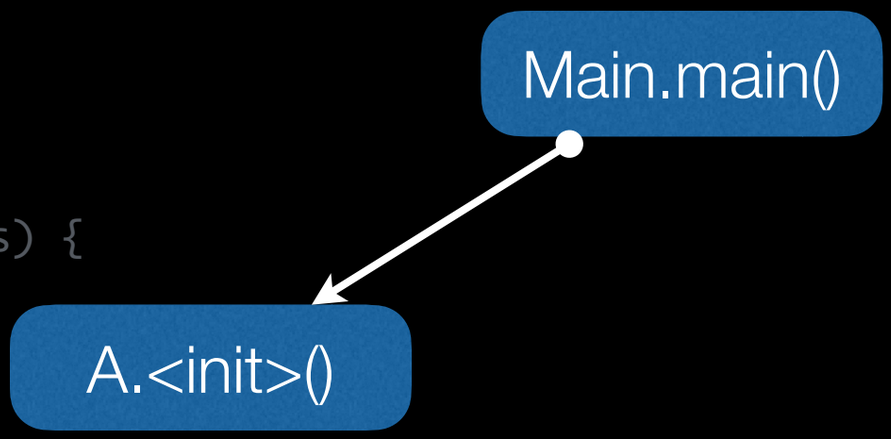
Reachable

Main.main()

Points-to `pts-to(a) = {new A}`

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```



Worklist

A.<init>()

Reachable

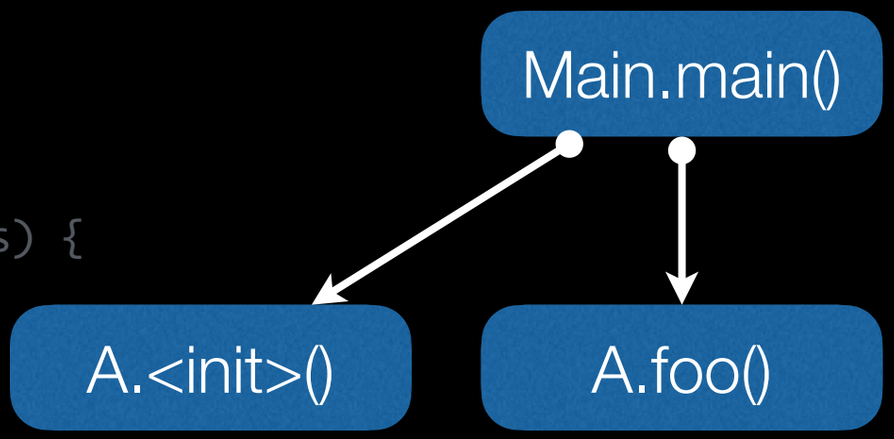
Main.main()

A.<init>()

Points-to `pts-to(a) = {new A}`

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```



Worklist

Reachable

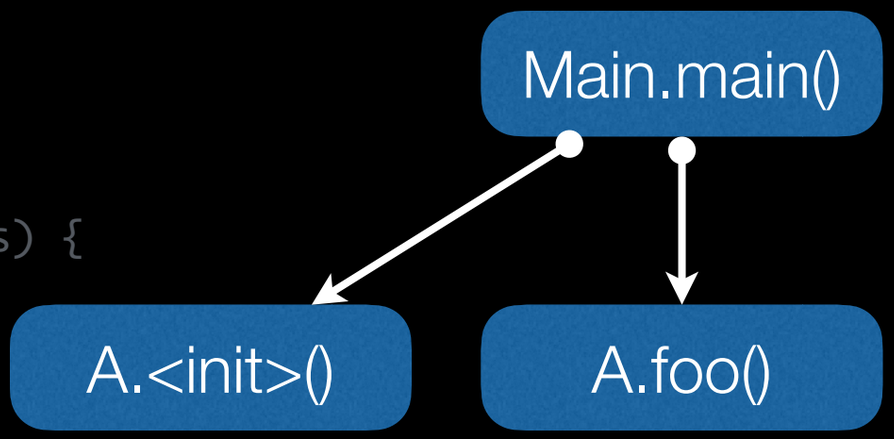
Main.main()

A.<init>()

Points-to `pts-to(a) = {new A}`

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```



Worklist

A.foo()

Reachable

Main.main()

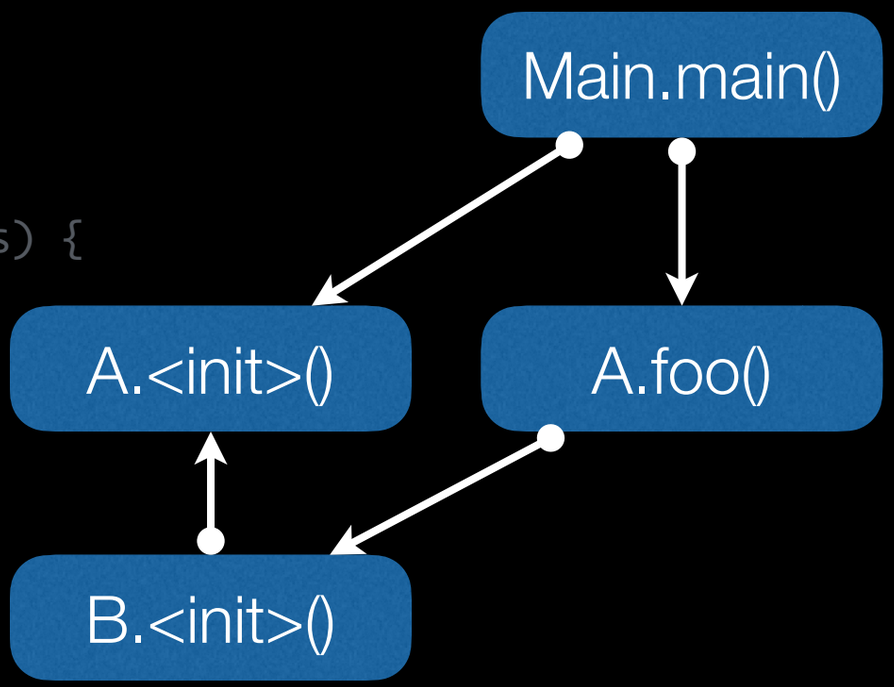
A.<init>()

A.foo()

Points-to `pts-to(a) = {new A}`

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```



Worklist

Reachable

Main.main()

A.<init>()

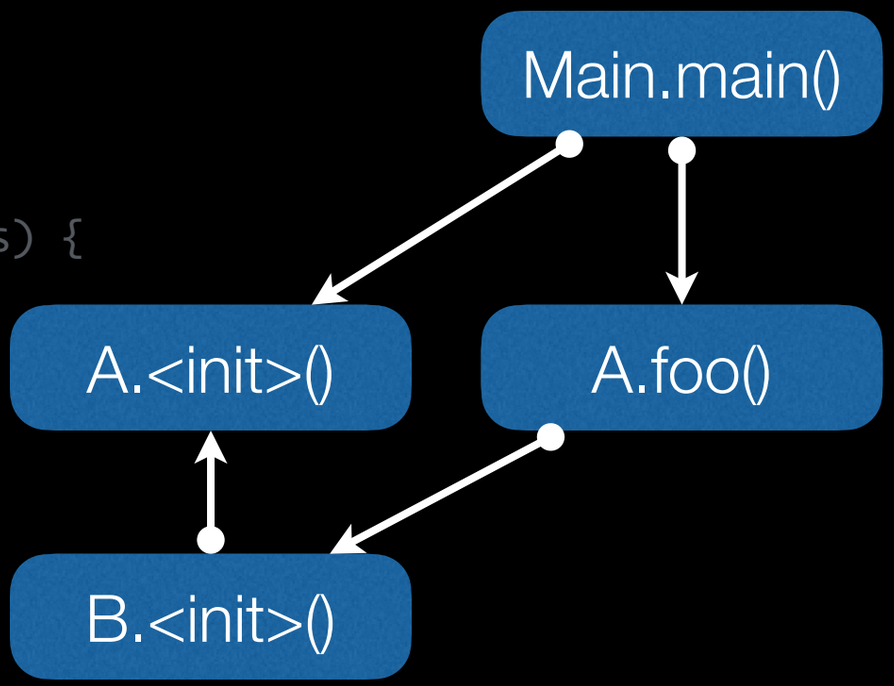
B.<init>()

A.foo()

Points-to `pts-to(a) = {new A}`

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```



Worklist

Reachable

Main.main()

A.<init>()

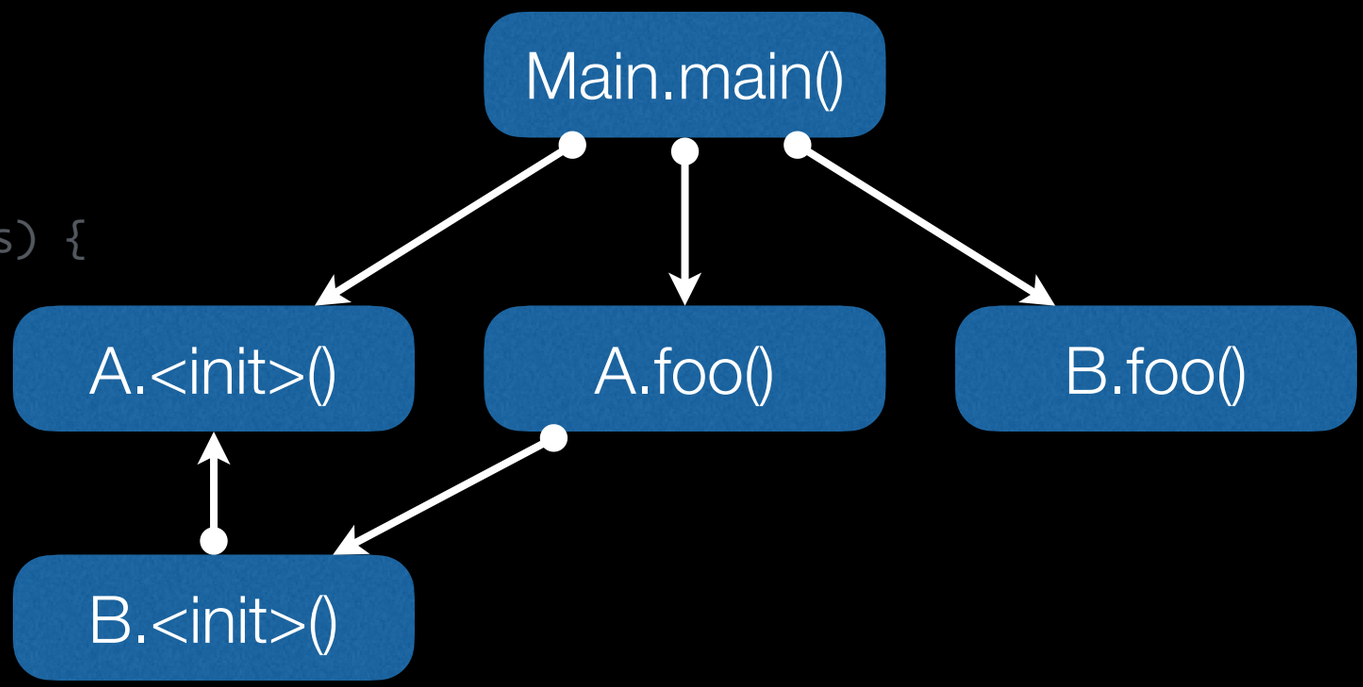
B.<init>()

A.foo()

Points-to `pts-to(a) = {new A, new B}`

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```



Worklist

Reachable

Main.main()

A.<init>()

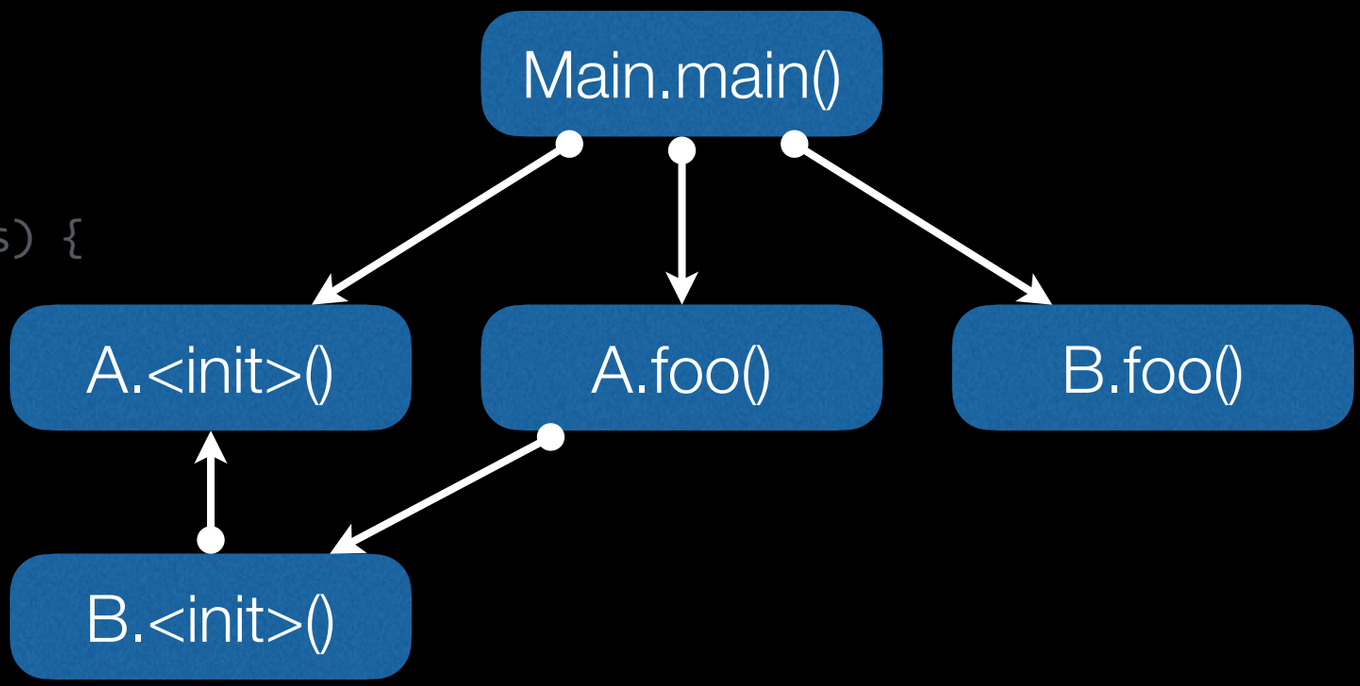
B.<init>()

A.foo()

Points-to `pts-to(a) = {new A, new B}`

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```



Worklist

B.foo()

Reachable

Main.main()

B.foo()

A.<init>()

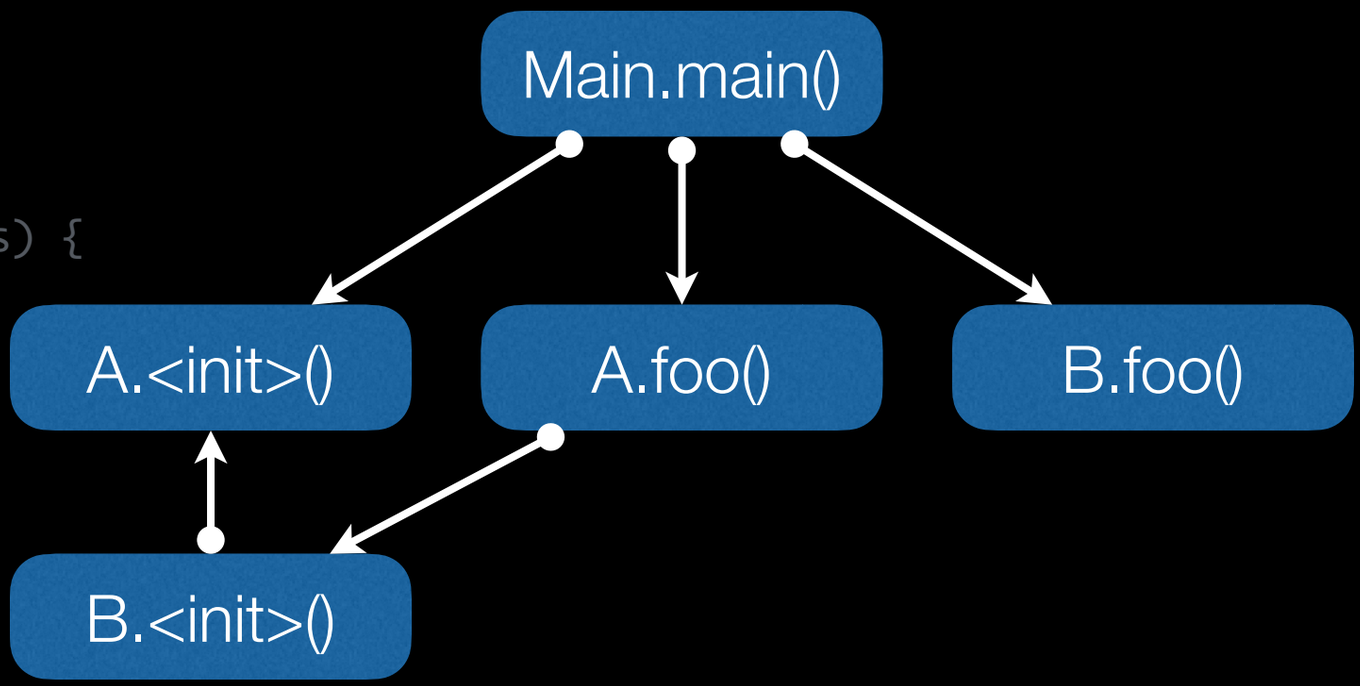
B.<init>()

A.foo()

Points-to `pts-to(a) = {new A, new B}`

SPARK - OTF CG Construction

```
public class Main {
  static A a;

  public static void main(String[] args) {
    a = new A();
    a.foo();
    a.foo();
  }

  static class A {
     void foo() { a = new B(); }
  }

  static class B extends A {
     void foo() {}
  }
}
```

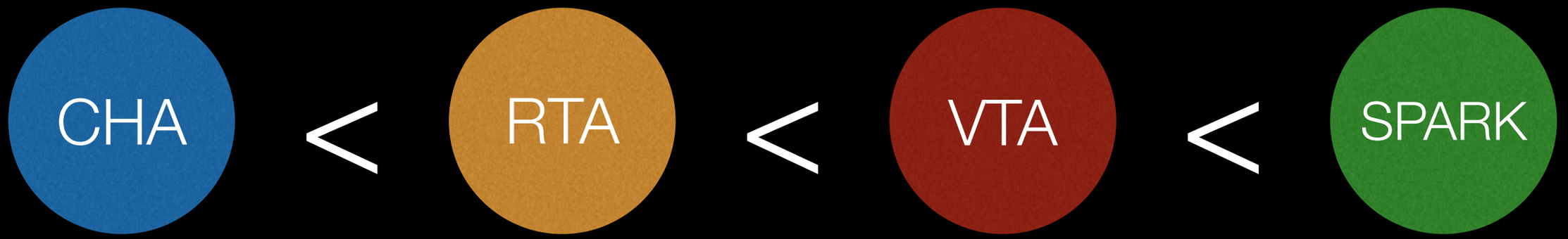


Worklist

Reachable

- Main.main()
- B.foo()
- A.<init>()
- B.<init>()
- A.foo()

Points-to `pts-to(a) = {new A, new B}`


CHA
<
RTA
<
VTA
<
SPARK

Next

- Call graph hands-on