Class Hierarchy Complementation: Soundly Completing a Partial Type Graph

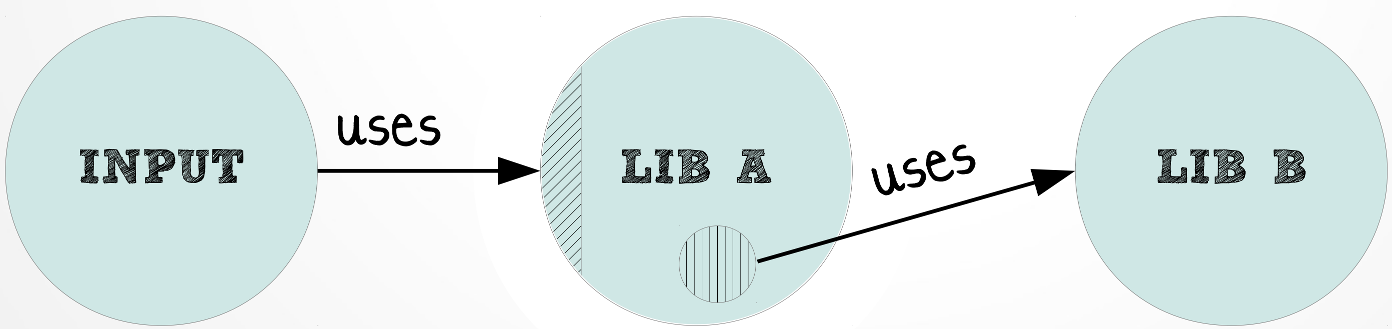
George Balatsouras
Yannis Smaragdakis
University of Athens

OOPSLA 2013

Motivation: Static Analysis

- Static Analysis using the Doop framework
  - Analyzes Java programs
  - Uses Soot to analyze bytecode
  - Whole-program analysis
  - External dependencies
  - Missing libraries / class defs

PHANTOM CLASSES



Motivation: Static Analysis

Soot FAQ

> How do I modify the code in order to enable soot to continue loading a class even if it doesn’t find some of it[s] references? Can I create a dummy soot class so it can continue with the load? How?

Motivation: Static Analysis

Soot FAQ

> How do I modify the code in order to enable soot to continue loading a class even if it doesn't find some of it[s] references? Can I create a dummy soot class so it can continue with the load? How?

> "You can try -use-phantom-refs but often that does not work because not all analyses can cope with such references.„

Complementation Problem

Partial Program

+

Phantom Classes

???

Complete Program

- Valid Java bytecode
- JVM Standard
- Verifiable

Complementation Problem

Partial Program

+

Phantom Classes

JPhantom

Complete Program

- Valid Java bytecode
- JVM Standard
- Verifiable

JPhantom: Approach

1. Detect every phantom reference
2. Generate minimal classes (empty method bodies) that respect the:
   i. referenced member signatures
   ii. implied type hierarchy

Motivating Example

X, Y, Z phantom classes

```
public void foo(X, Y) :
aload_2    // load 2nd arg (Y) into stack
aload_1    // load 1st arg (X) into stack
invokevirtual `Z  X.bar(A)`
invokevirtual `void B.baz()`
...
```

Symbolic Execution: Step 1

X, Y, Z phantom classes

```
public void foo(X, Y) :
aload_2      // load 2nd arg (Y) into stack
aload_1      // load 1st arg (X) into stack
invokevirtual `Z  X.bar(A)`
invokevirtual `void B.baz()`
...
```

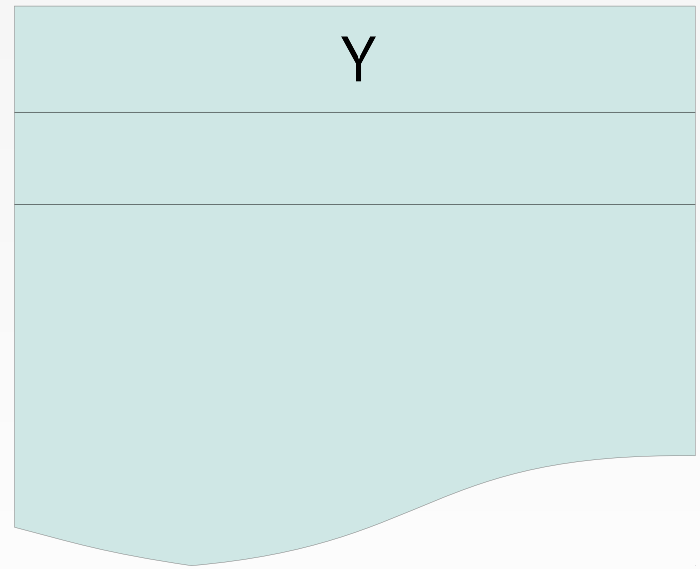
Stack

Symbolic Execution: Step 2

X, Y, Z phantom classes

```
public void foo(X, Y) :
aload_2     // load 2nd arg (Y) into stack
aload_1     // load 1st arg (X) into stack
invokevirtual `Z  X.bar(A)`
invokevirtual `void B.baz()`
...
```

Stack

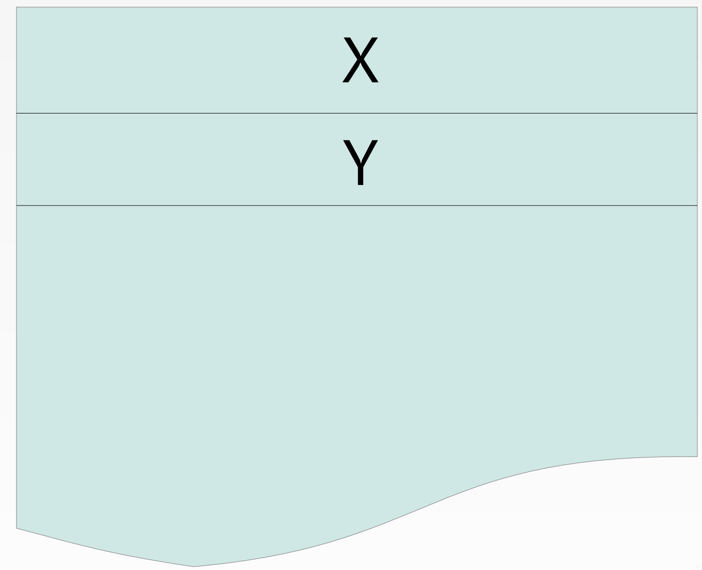


Symbolic Execution: Step 3

X, Y, Z phantom classes

```
public void foo(X, Y) :
aload_2    // load 2nd arg (Y) into stack
aload_1    // load 1st arg (X) into stack
invokevirtual `Z  X.bar(A)`
invokevirtual `void B.baz()`
...
```

Stack



Symbolic Execution: Step 3

X, Y, Z phantom classes

public void foo(X, Y) :

aload_2 *// load 2nd arg (Y) into stack*

aload_1 *// load 1st arg (X) into stack*

invokevirtual `Z X.bar(A)`

invokevirtual `void B.baz()`

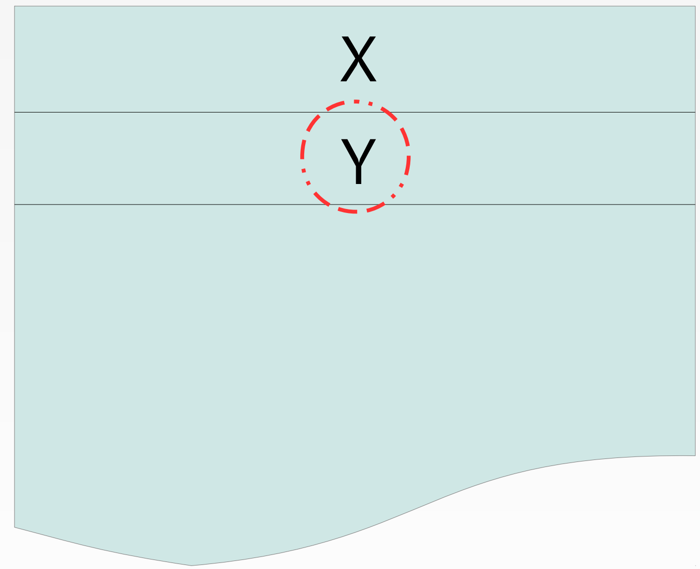
...

Method bar:

- expects an argument of type A
- receives an argument of type Y

upcast

Stack



Constraints

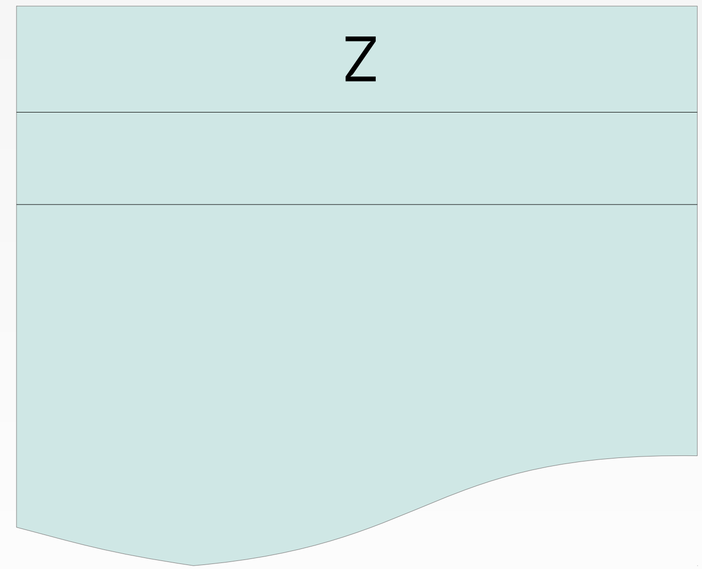
- X has to be a class (and not an interface).
- X has to provide a method: Z bar(A)
- Y has to be a subtype of A

Symbolic Execution: Step 4

X, Y, Z phantom classes

```
public void foo(X, Y) :
aload_2      // load 2nd arg (Y) into stack
aload_1      // load 1st arg (X) into stack
invokevirtual `Z  X.bar(A)`
invokevirtual `void B.baz()`
...
```

Stack



Constraints

- X has to be a class (and not an interface).
- X has to provide a method: Z bar(A)
- Y has to be a subtype of A

Symbolic Execution: Step 4

X, Y, Z phantom classes

public void foo(X, Y) :

aload_2 *// load 2nd arg (Y) into stack*

aload_1 *// load 1st arg (X) into stack*

invokevirtual `Z X.bar(A)`

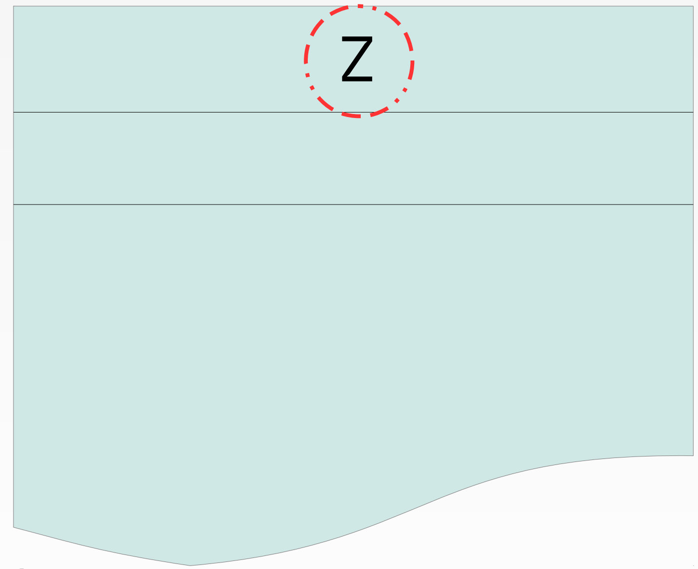
invokevirtual `void B.baz()`

...

Method baz:

- declares a receiver of type B
- is called by an object of type Z

Stack



Constraints

- X has to be a class (and not an interface).
- X has to provide a method: Z bar(A)
- Y has to be a subtype of A
- Z has to be a subtype of B

Symbolic Execution: Step 5

X, Y, Z phantom classes

```
public void foo(X, Y) :
aload_2    // load 2nd arg (Y) into stack
aload_1    // load 1st arg (X) into stack
invokevirtual `Z  X.bar(A)`
invokevirtual `void B.baz()`
...
```

Stack

Constraints

- X has to be a class (and not an interface).
- X has to provide a method: Z bar(A)
- Y has to be a subtype of A
- Z has to be a subtype of B

Hierarchy Complementation

Partial Type Graph

+

subtyping constraints

E.g., <A> has to be a (transitive) subtype of <B>

Complete Hierarchy

JPhantom

Multiple Inheritance

Multiple Inheritance

Wait a minute. Aren't we talking about Java?

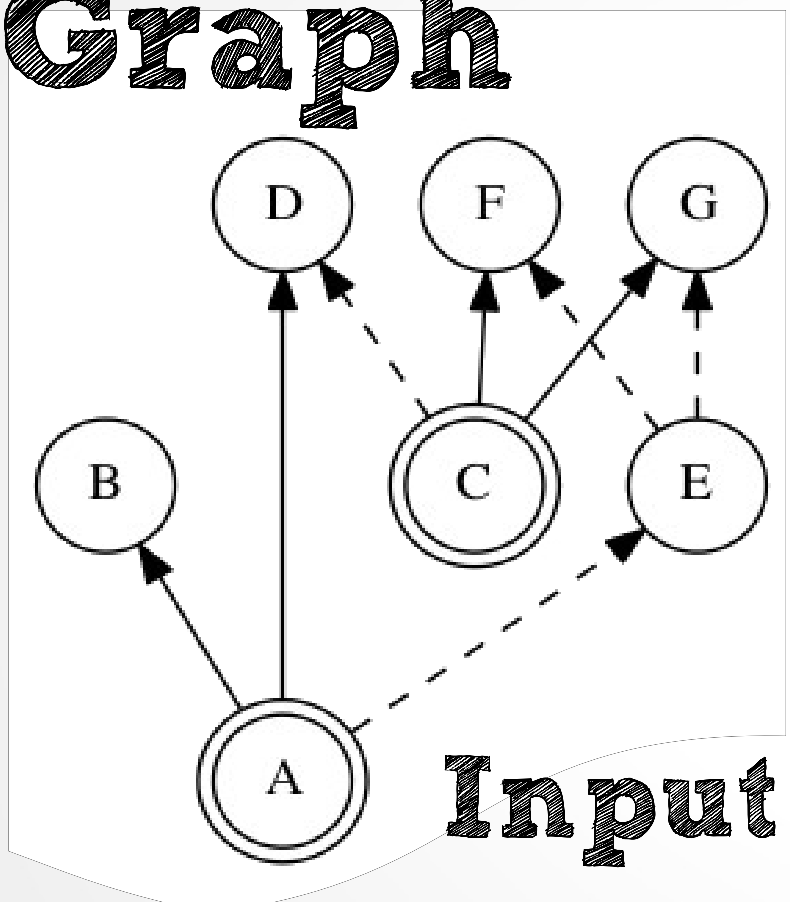
Multiple Inheritance

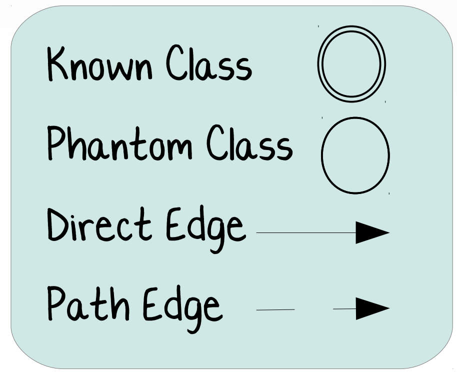
Interfaces

Wait a minute. Aren't we talking about Java?

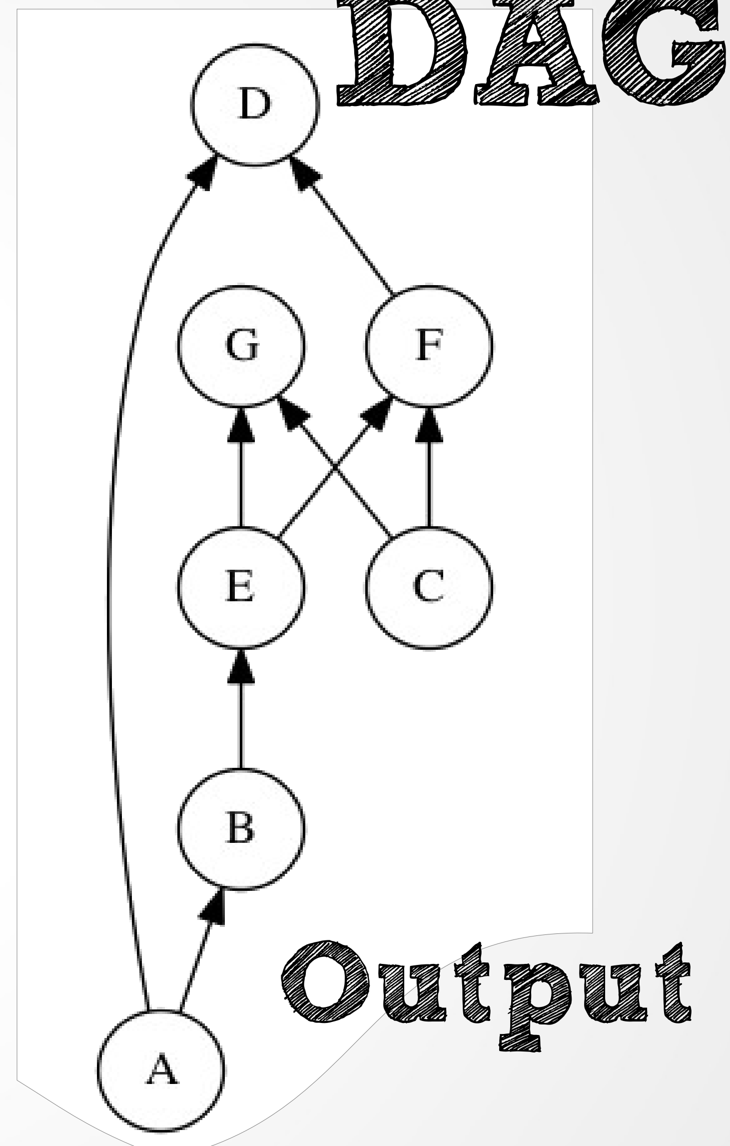
Multiple Inheritance Problem

Constraint Graph



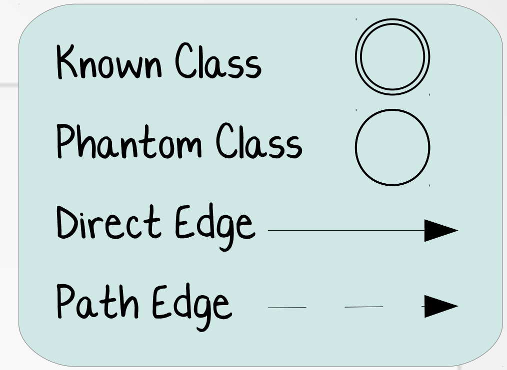


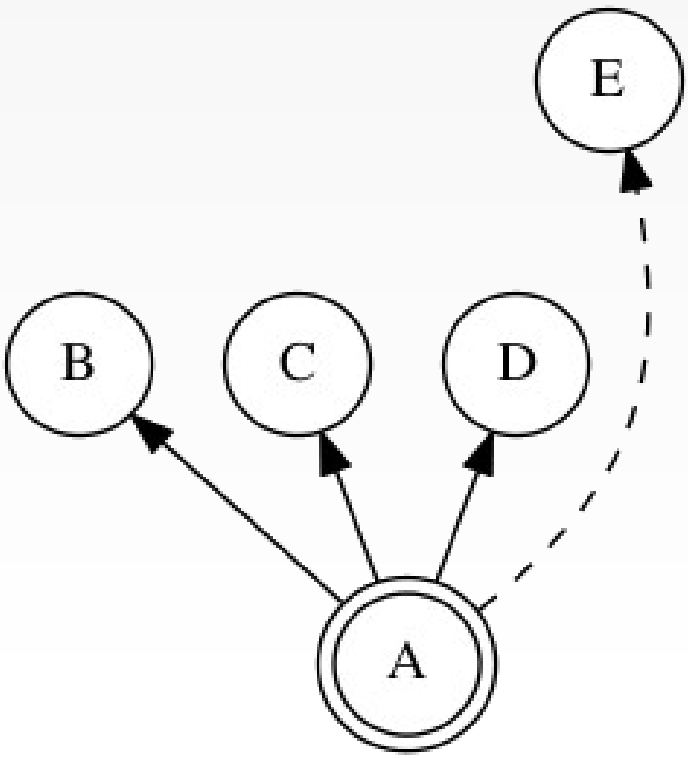
DAG



Cannot alter outgoing edges of known nodes

Projection Sets





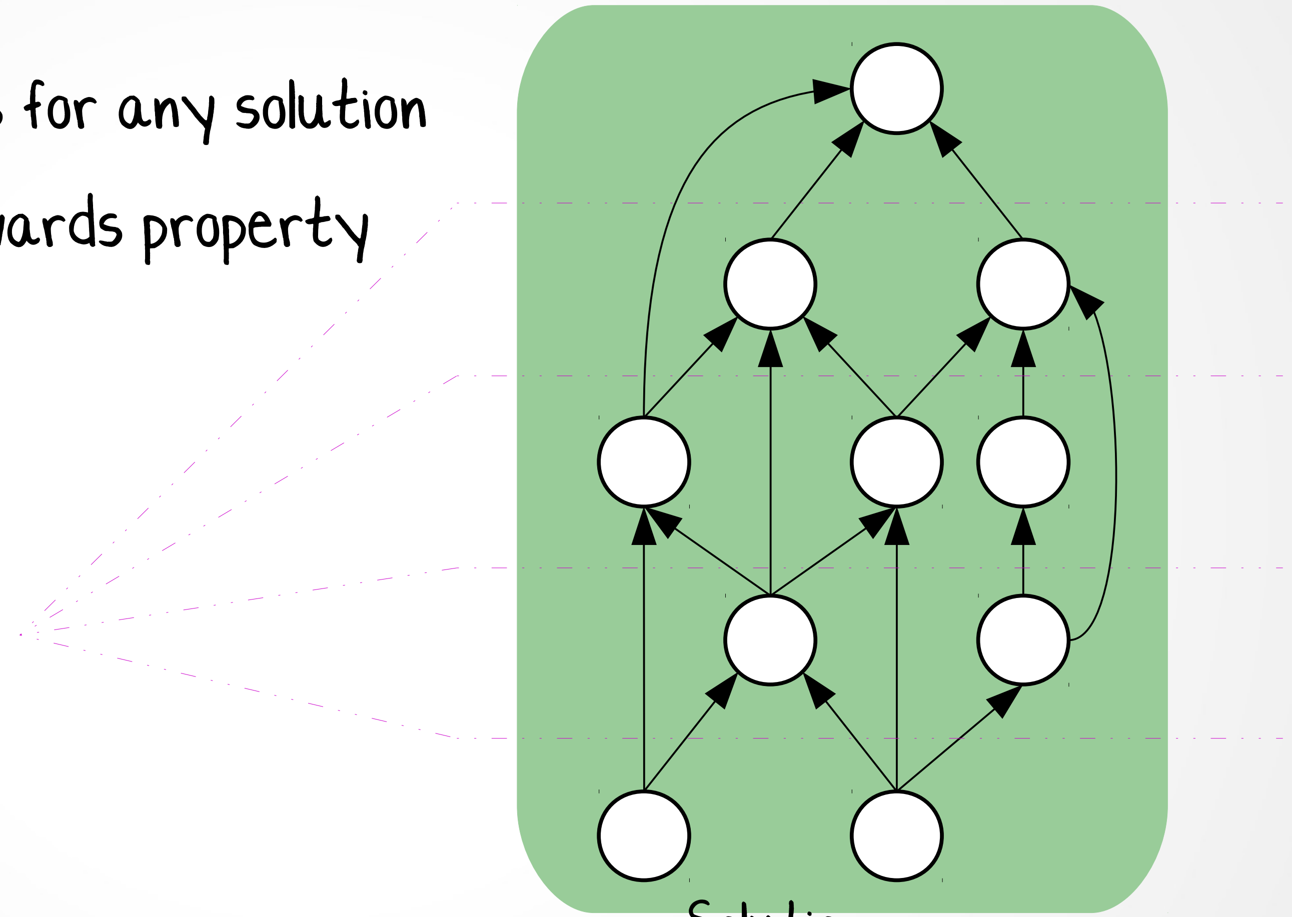
- The phantom projection set of A is {B,C,D}.
- In order to satisfy path-edge (A,E) we can either add an edge (B,E), (C,E), or (D,E).

Multiple Inheritance

Key Idea

- Stratification exists for any solution
  - Edges facing upwards property
    - – No cycles

strata



Solution

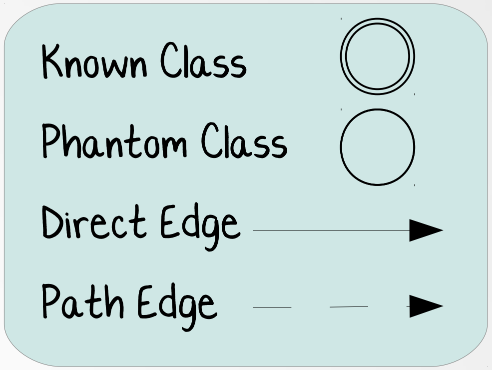
Multiple Inheritance

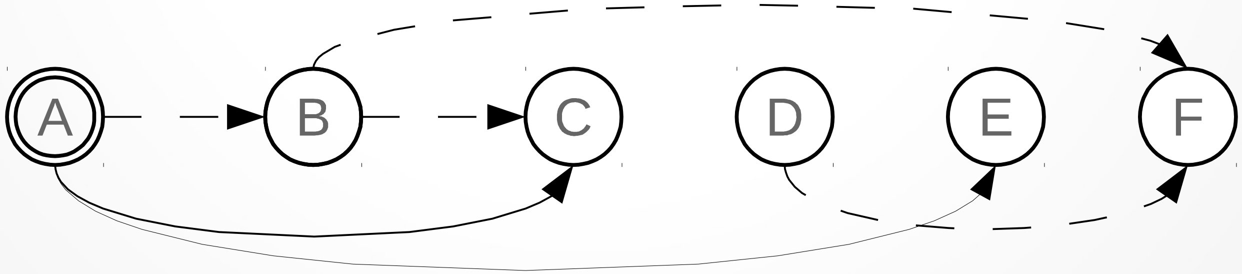
Algorithm

- Construct valid stratification iteratively
  - Keep nodes at minimum height
  - Keep edges facing upwards
  - Advance node only to satisfy constraint
  - Fixpoint
- Add upward edges to satisfy path constraints

Multiple Inheritance Example

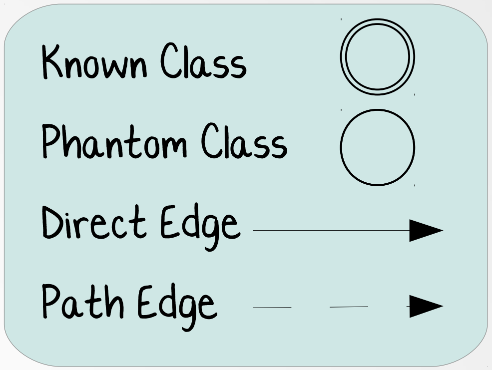
Step 1

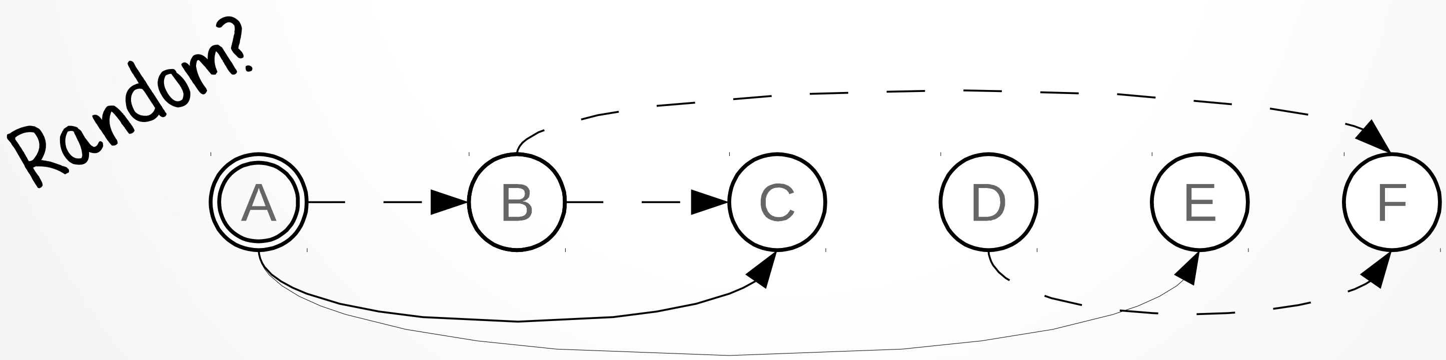




Multiple Inheritance Example

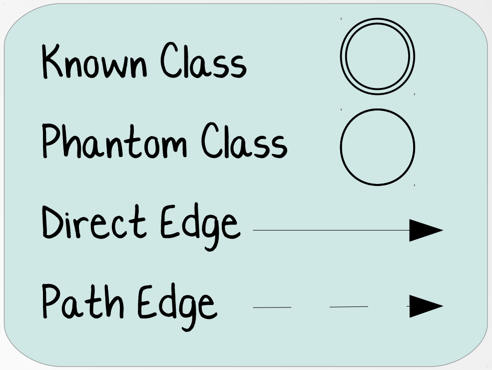
Step 1



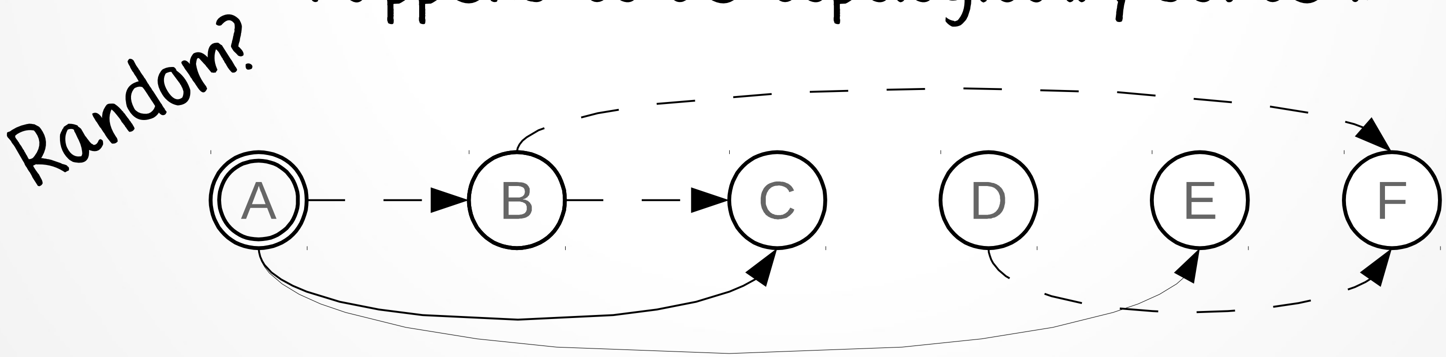


Multiple Inheritance Example

Step 1

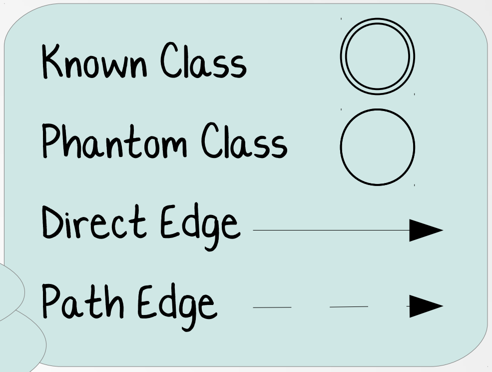


Not really.
Happens to be topologically sorted.



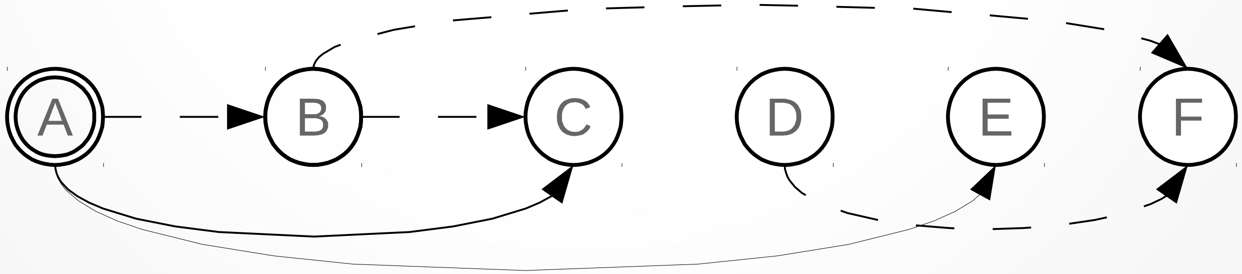
Multiple Inheritance Example

Step 1



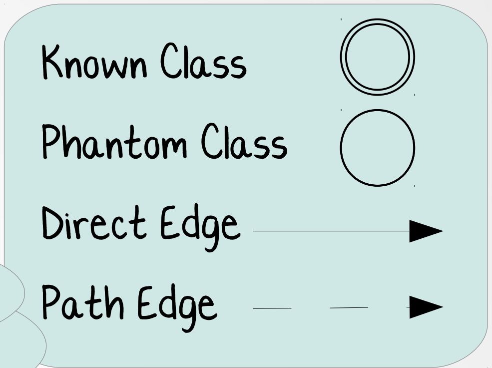
Isn't that enough?

Happens to be topologically sorted.



Multiple Inheritance Example

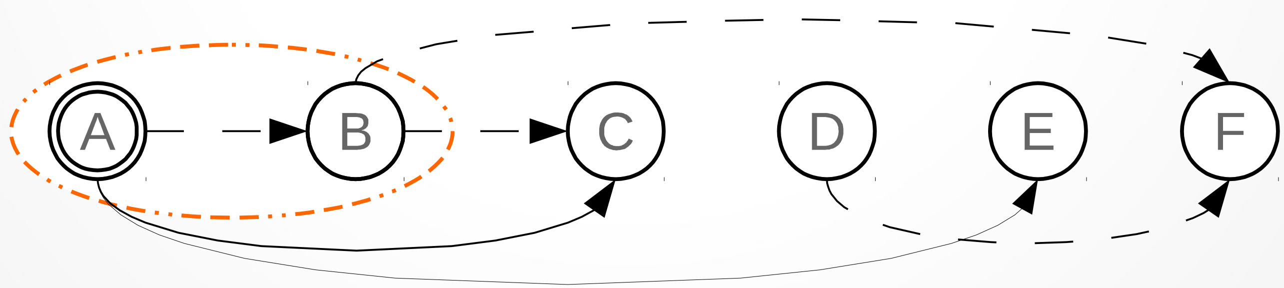
Step 1



Isn't that enough?

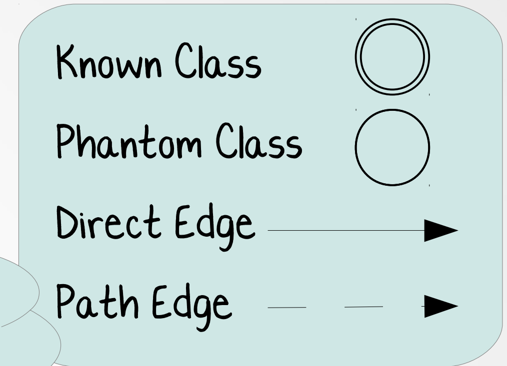
Path-edge (A,B) cannot be satisfied since both C and E are after B.

Happens to be topologically sorted.



Multiple Inheritance Example

Step 1

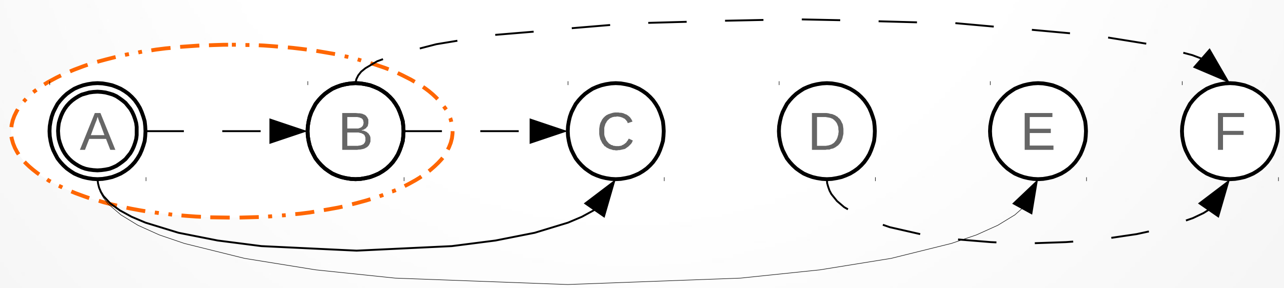


Isn't that enough?

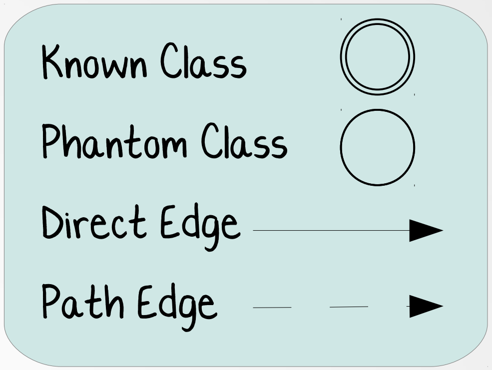
NO!!

Path-edge (A,B) cannot be satisfied since both C and E are after B.

Happens to be topologically sorted.

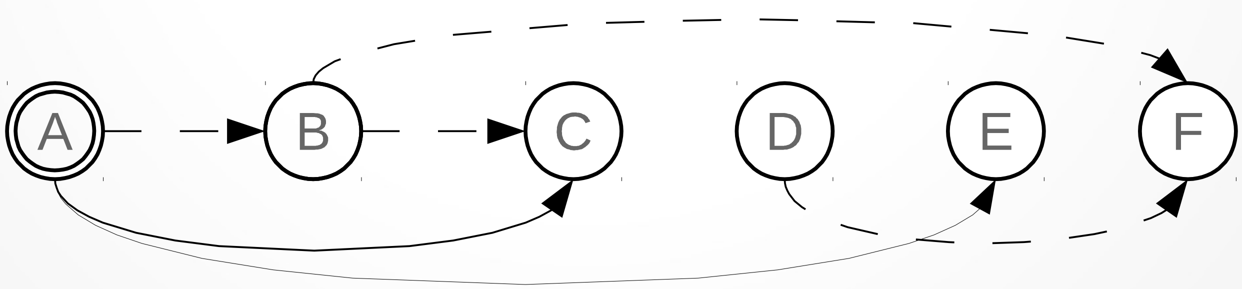


Multiple Inheritance Example



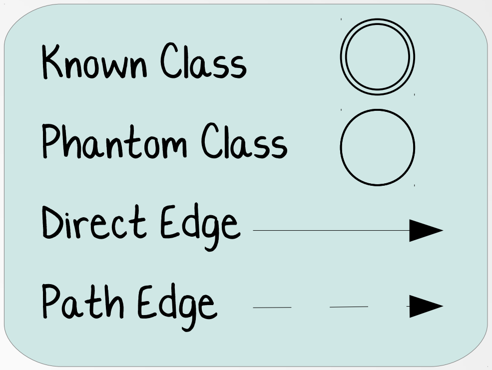
Step 1

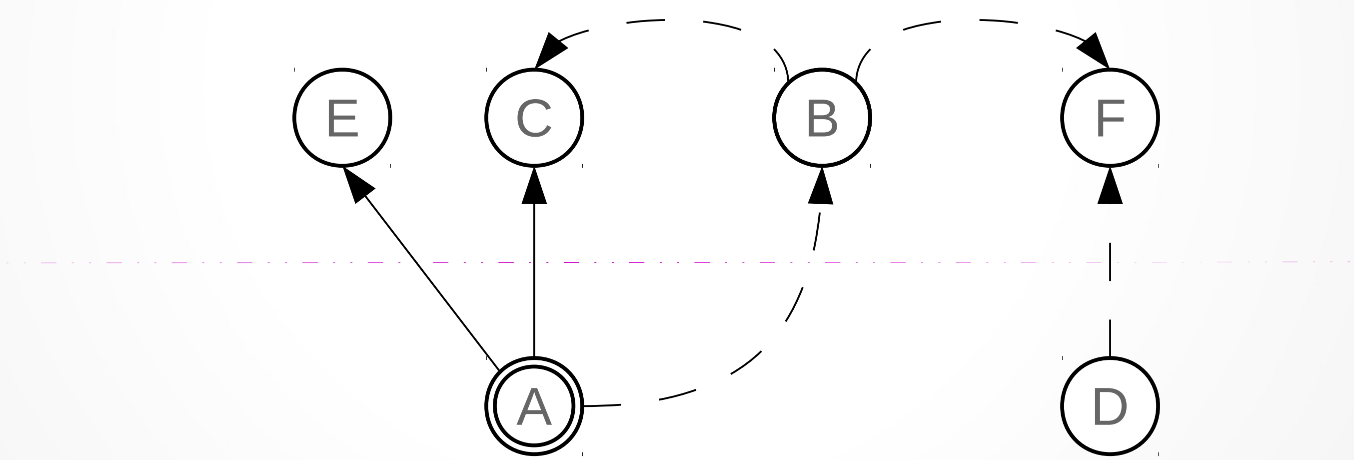
- All nodes except A, and D have incoming (horizontal) egdes



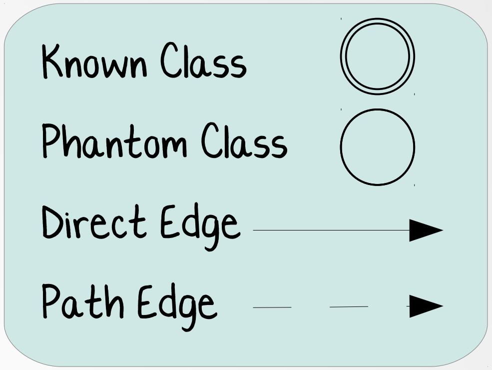
Multiple Inheritance Example

Step 2



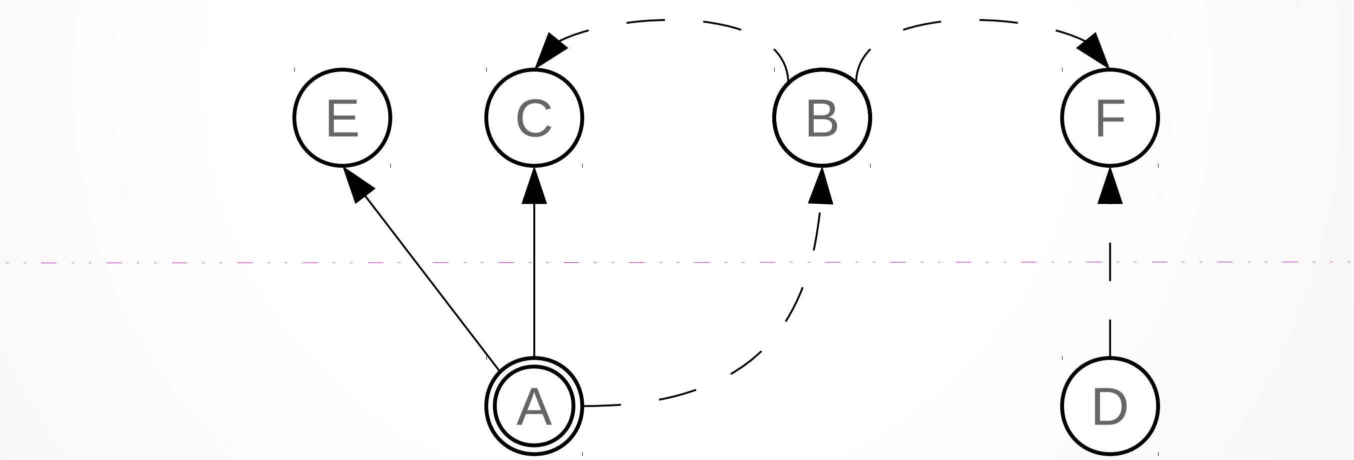


Multiple Inheritance Example



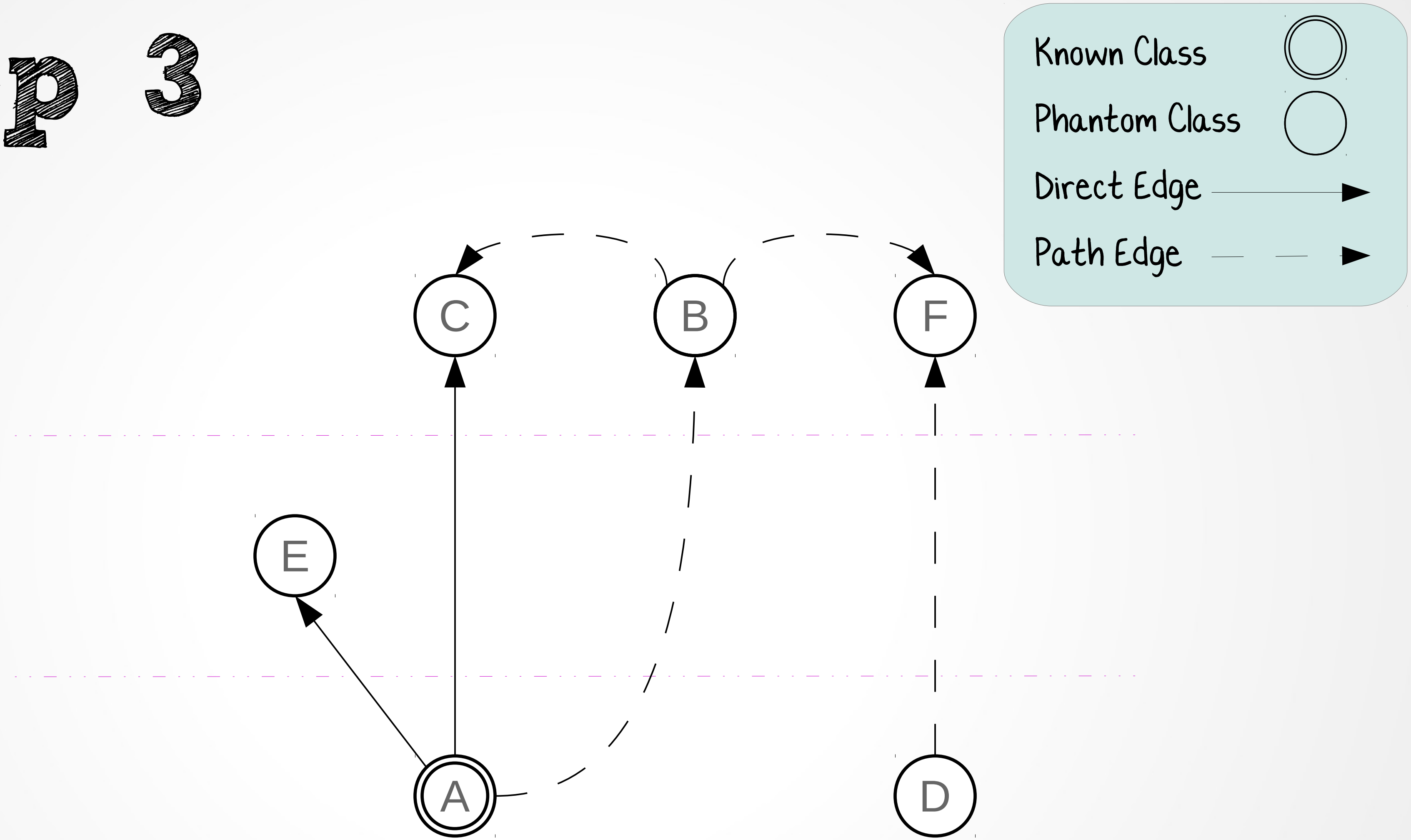
Step 2

- Nodes C, F have incoming (horizontal) egdes
- Node B is not yet higher than neither C nor E



Multiple Inheritance Example

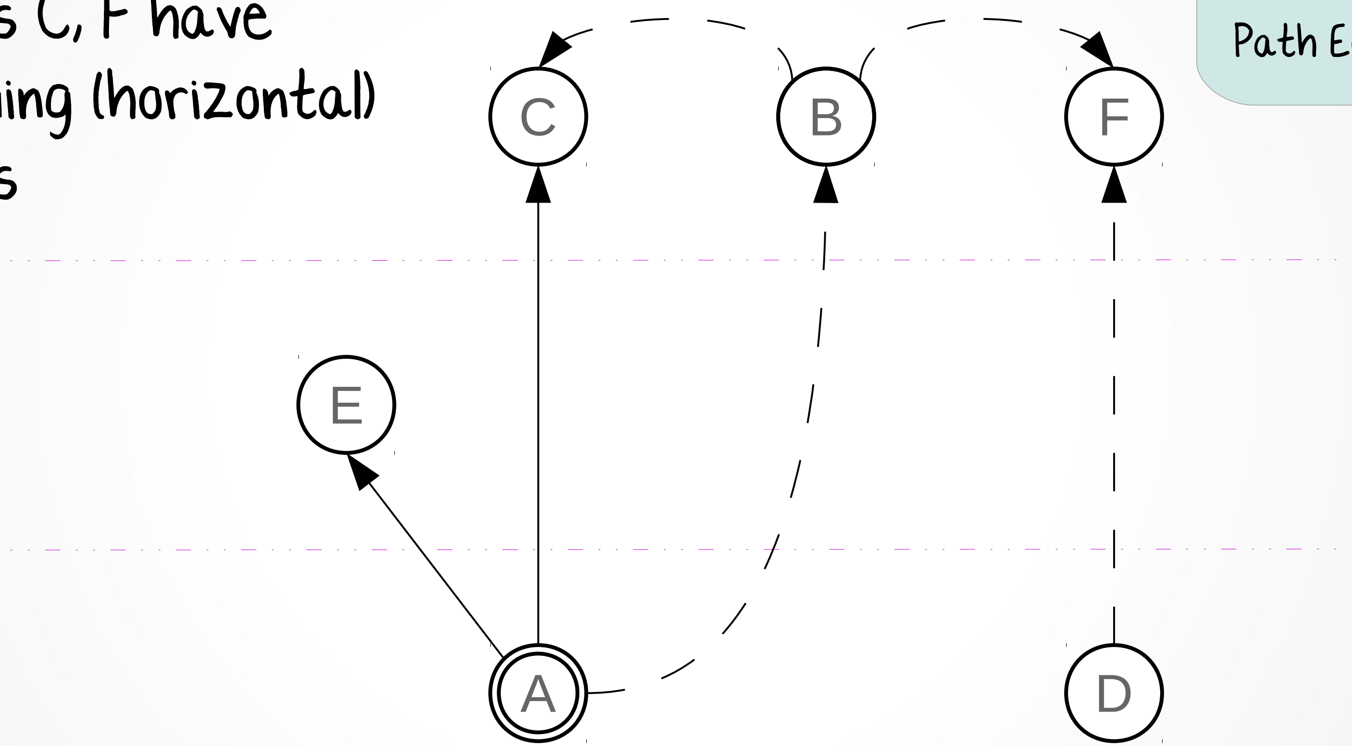
Step 3

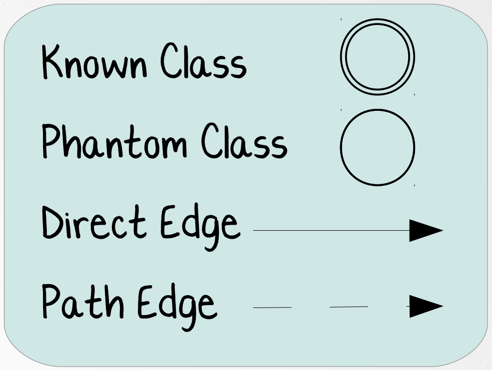


Multiple Inheritance Example

Step 3

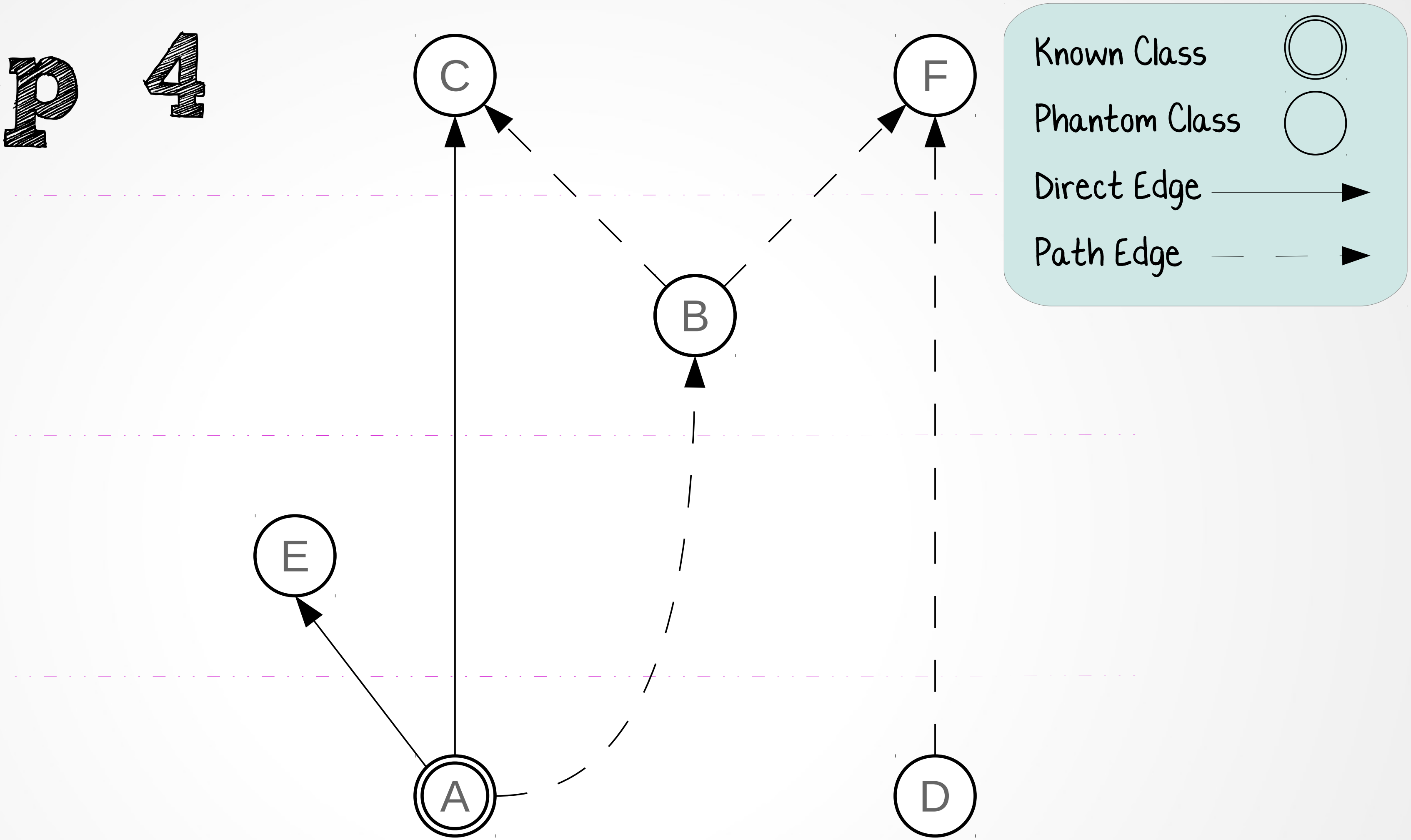
- Nodes C, F have incoming (horizontal) egdes





Multiple Inheritance Example

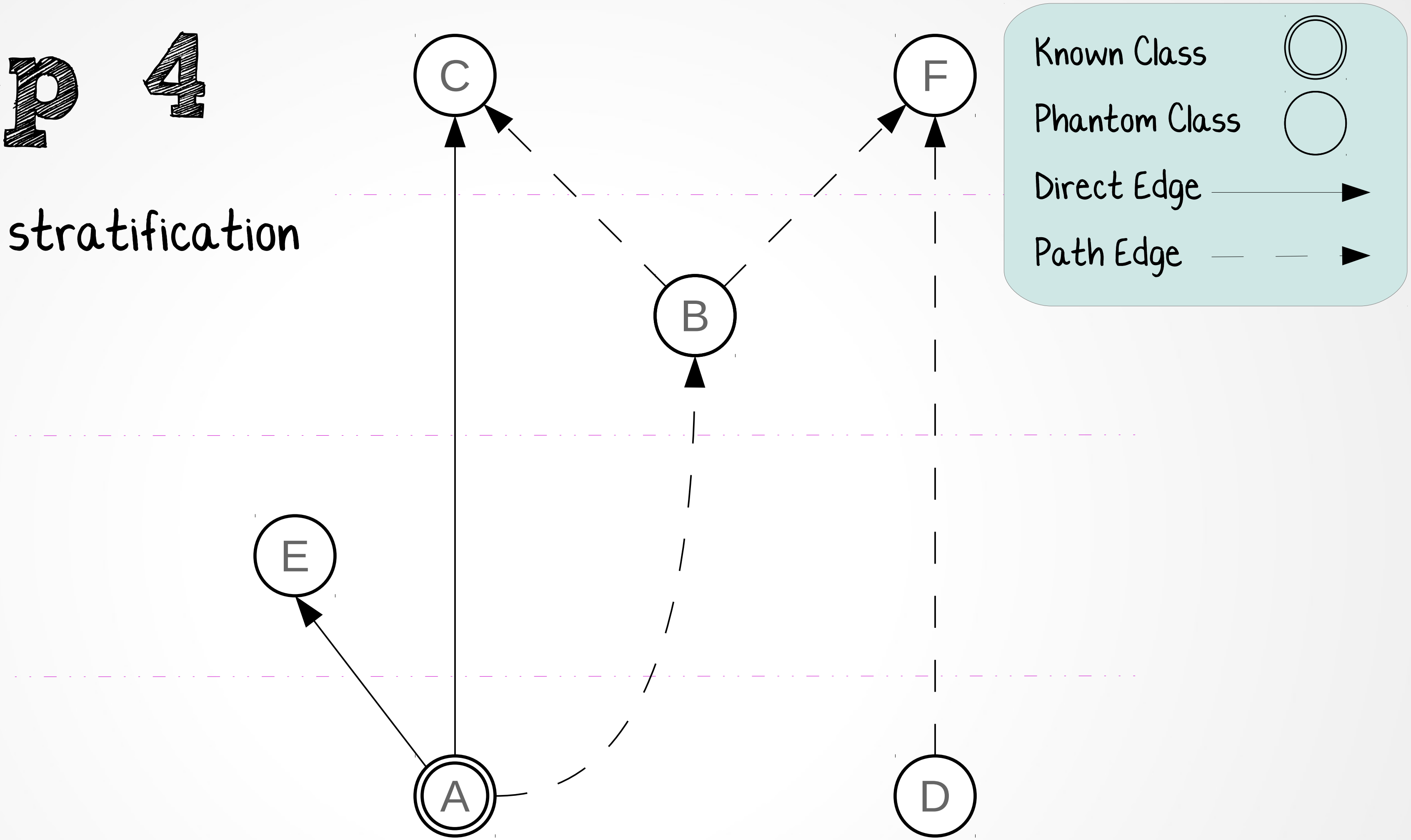
Step 4



Multiple Inheritance Example

Step 4

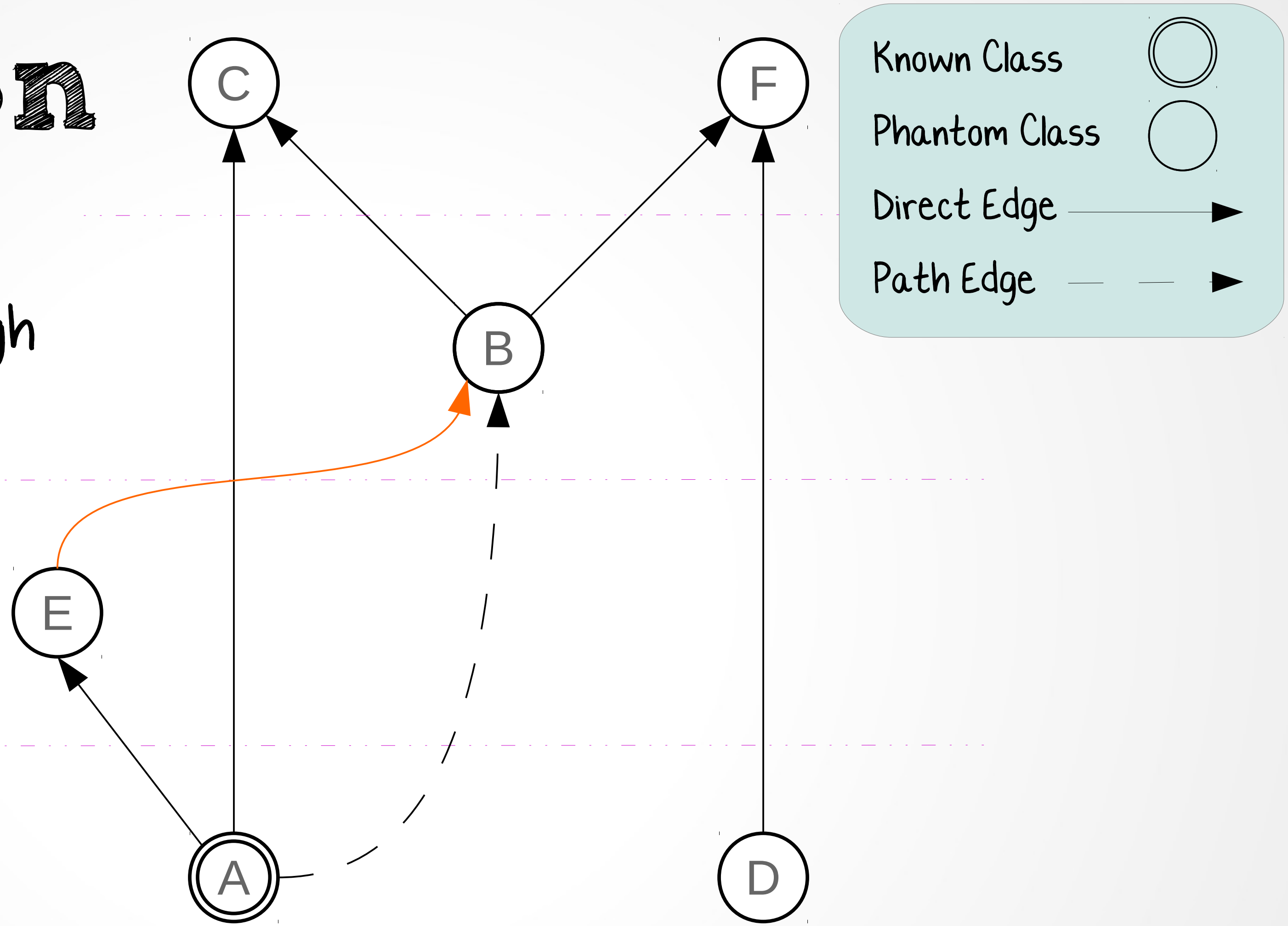
- ✔ Final stratification



Multiple Inheritance Example

Solution

- ✓ Path-edge (A,B) satisfied through path (A,E,B)



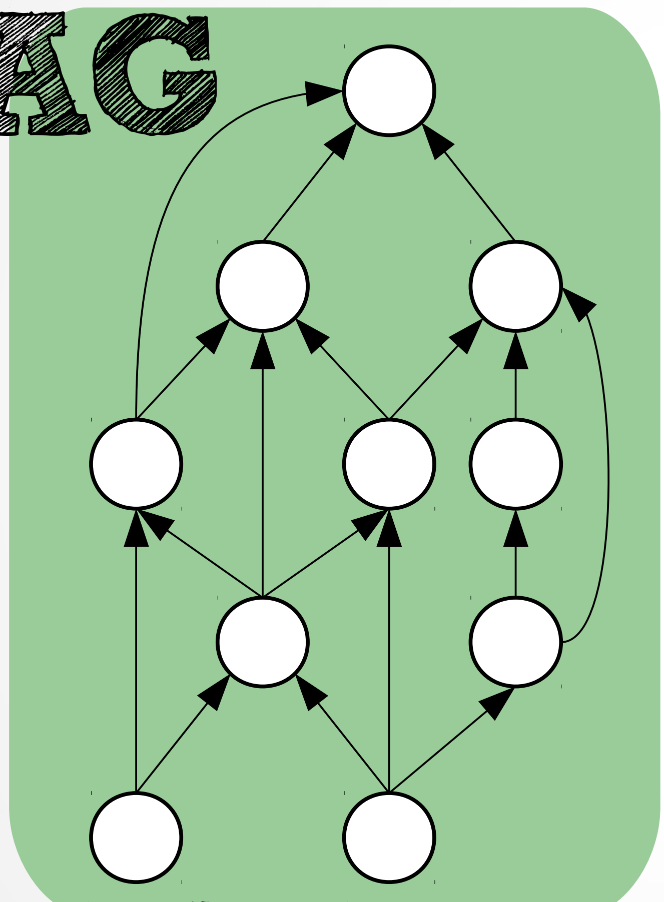
Single Inheritance

Classes

Multiple vs Single Inheritance

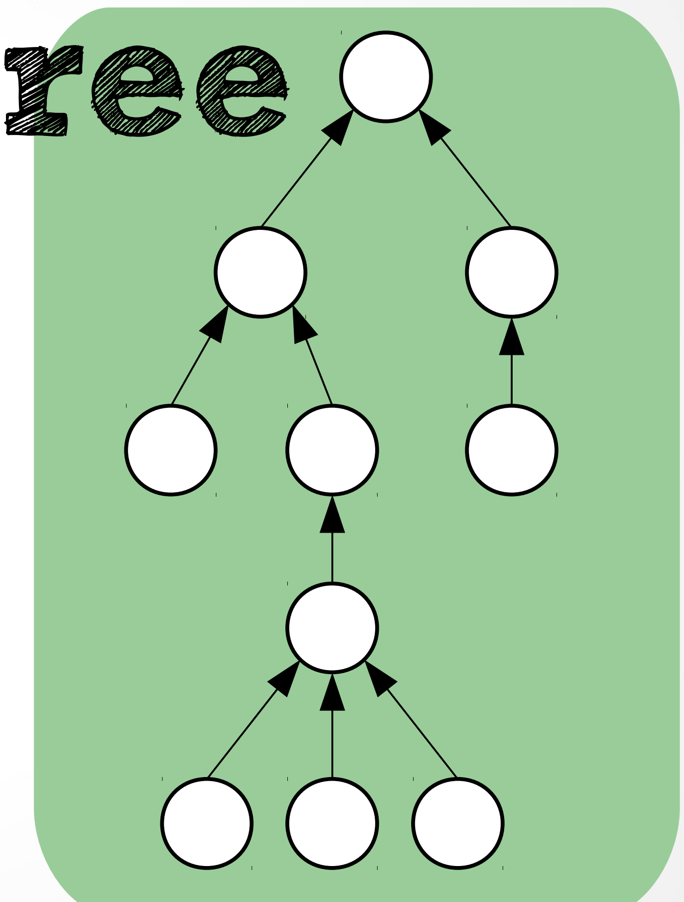
Just one additional constraint on the output
...which now has to form a

DAG



Multiple Inheritance

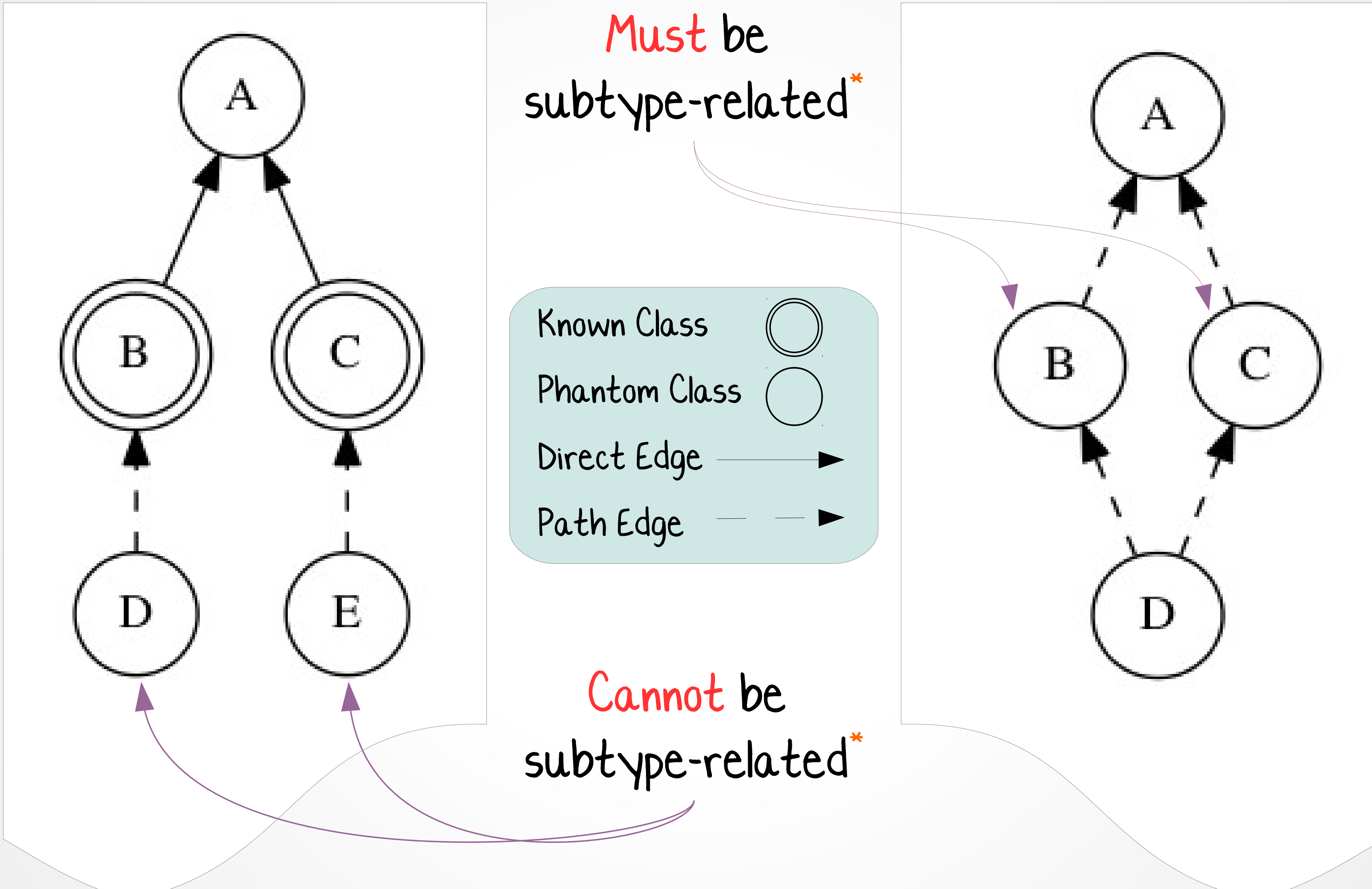
Tree



Single Inheritance

Single Inheritance: Examples

* X, Y subtype-related if X subtype or supertype of Y



JPhantom: Algorithms

Solvers

- Single inheritance
  - Polynomial if no direct-edges to phantom-nodes
  - Worst-case exponential (backtracking)
  - Quite effective in practice
- Multiple inheritance
  - Polynomial
- Single inheritance, multiple subtyping (e.g., Java)
  - Decompose into a single and a multiple inheritance subproblems

JPhantom: Overview

About JPhantom

- Solves the hierarchy complementation problem for all 3 settings
- Uses the ASM framework to operate on bytecode
- Constraint Extraction Step
  - Detects type constraints and missing member references
- Code Generation Step
  - Generates dummy classes, yet consistent with our input

JPhantom: Performance

About JPhantom

- Highly Scalable
  - runs in mere seconds even for large applications and complex constraints
  - 148 phantom classes and 212 constraints, where execution time < 2sec, for *logback-classic*
  - Maximum execution time of 14s for *JRuby*
    - 19MB binary

Summary

In summary, we:

- Introduce the class hierarchy complementation problem
- Provide algorithms for:
  i. single inheritance
  ii. multiple inheritance, and
  iii. single inheritance multiple subtyping
- Implement our algorithms in JPhantom, a practical tool for program complementation
  - highly scalable
  - meets Java bytecode requirements