Karlsruhe Institute of Technology

Speeding up context-, object- and field-sensitive SDG generation

Jürgen Graf

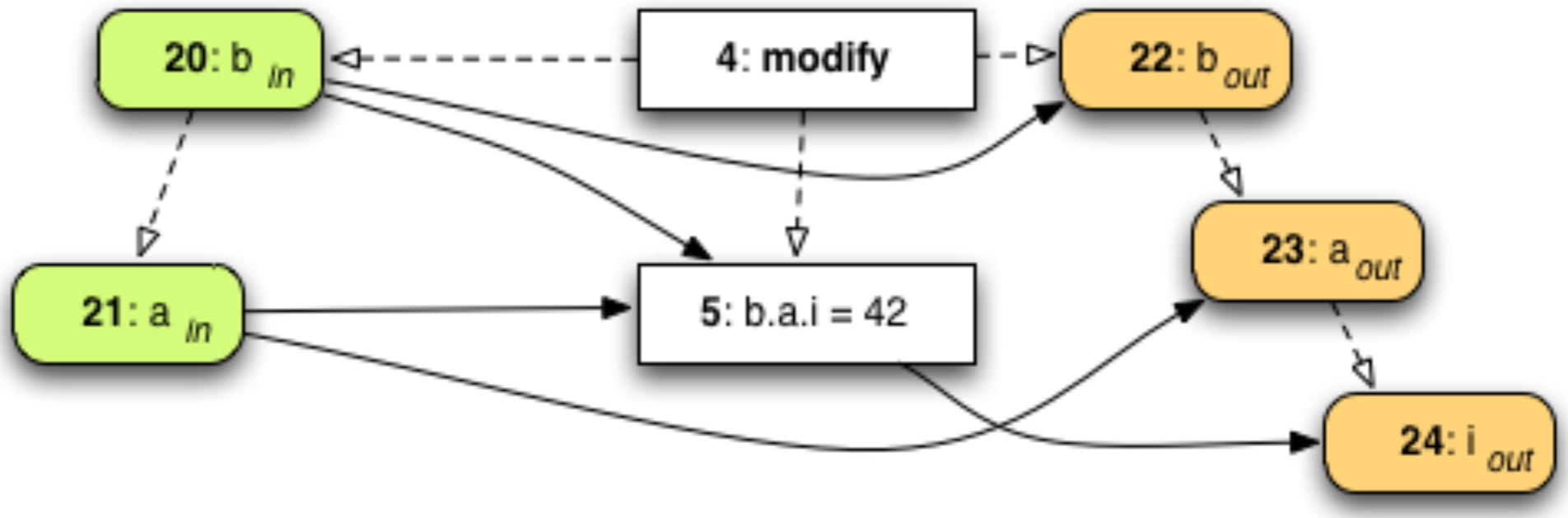

```
theorem nonInterferenceSecurity:
  assumes "[cf1] ≈L [cf2]" and "(-High-) ∉ ⌊HRB-slice (CFG-node (-Low-))⌋CFG" and "valid-edge a"
  and "sourcenode a = (-High-)" and "targetnode a = n" and "kind a = (λs. True)√" and "n ≙ c"
  and "final c'" and "⟨c,[cf1]⟩ ⇒ ⟨c',s1⟩" and "⟨c,[cf2]⟩ ⇒ ⟨c',s2⟩"
  shows "s1 ≈L s2"
proof –
  from High-target-Entry-edge obtain ax where "valid-edge ax" and "sourcenode ax = (-Entry-)"
    and "targetnode ax = (-High-)" and "kind ax = (λs. True)√" by blast
  from `n ≙ c` `⟨c,[cf1]⟩ ⇒ ⟨c',s1⟩` obtain n1 as1 cfs1 where "n –as1→√* n1" and "n1 ≙ c'" and "preds (kinds as1) [(cf1,undefined)]"
    and "transfers (kinds as1) [(cf1,undefined)] = cfs1" and "map fst cfs1 = s1" by(fastsimp dest:fundamental-property)
  from `n –as1→√* n1` `valid-edge a` `sourcenode a = (-High-)` `targetnode a = n` `kind a = (λs. True)√`
  have "(-High-) –a#as1→√* n1" by(fastsimp intro:Cons-path simp:vp-def valid-path-def)
  from `final c'` `n1 ≙ c'` obtain a1 where "valid-edge a1" and "sourcenode a1 = n1" and "targetnode a1 = (-Low-)" and "kind a1 = ↑id"
    by(fastsimp dest:final-edge-Low)
```

Precise System Dependence Graphs

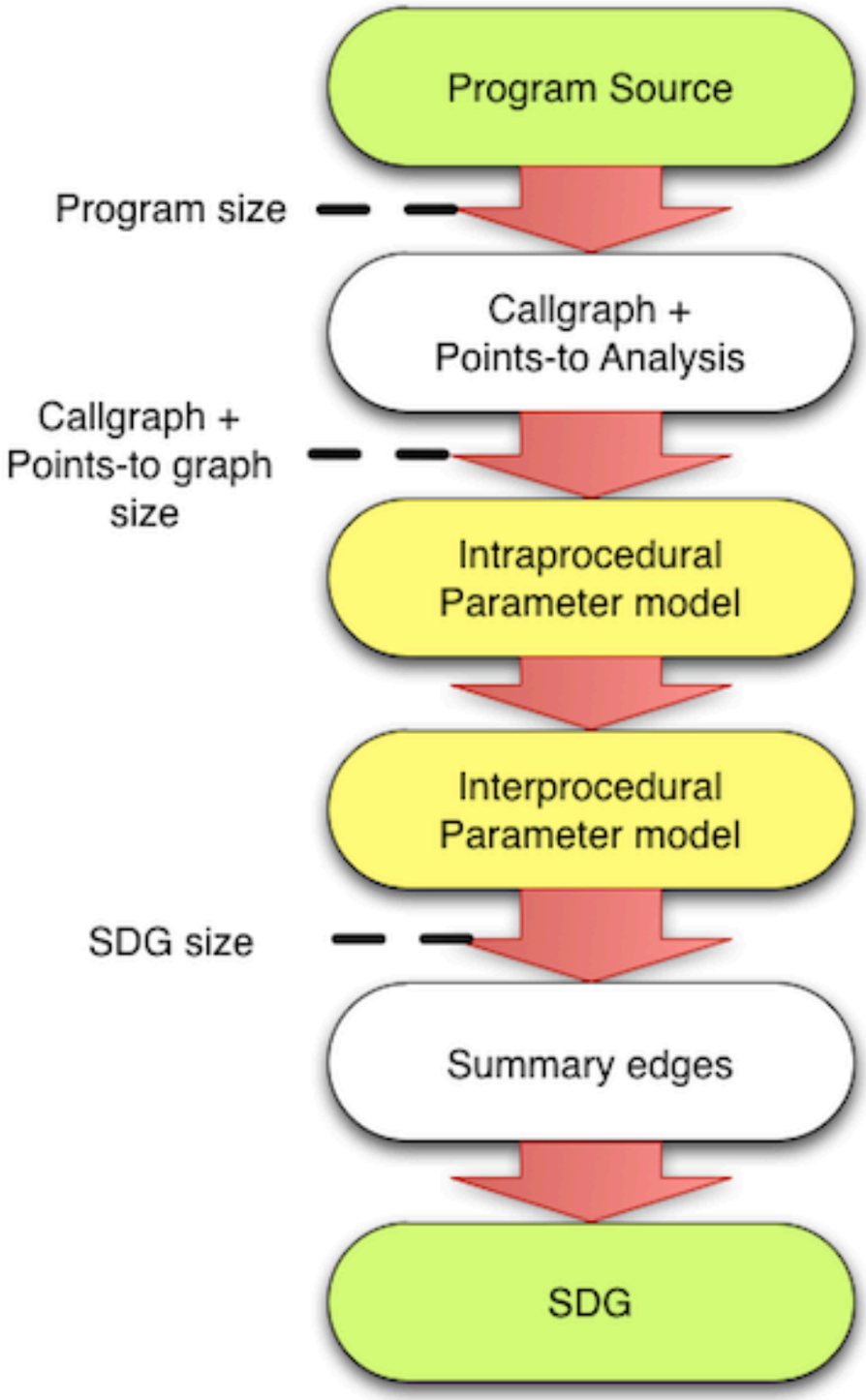
- **Heavy-weight** whole program analysis (**Interprocedural**)
- Object-oriented language (Java)
- System Dependence Graph (SDG)
  - Nodes: Statements
  - Edges: Dependencies between statements
    - A $\rightarrow$ B: A may influence B
    - A $\nrightarrow$ B: A certainly does not influence B
- Used for
  - **Information Flow Control** (IFC)
  - Concurrent Programs, ...

Context-, object- and field-sensitivity

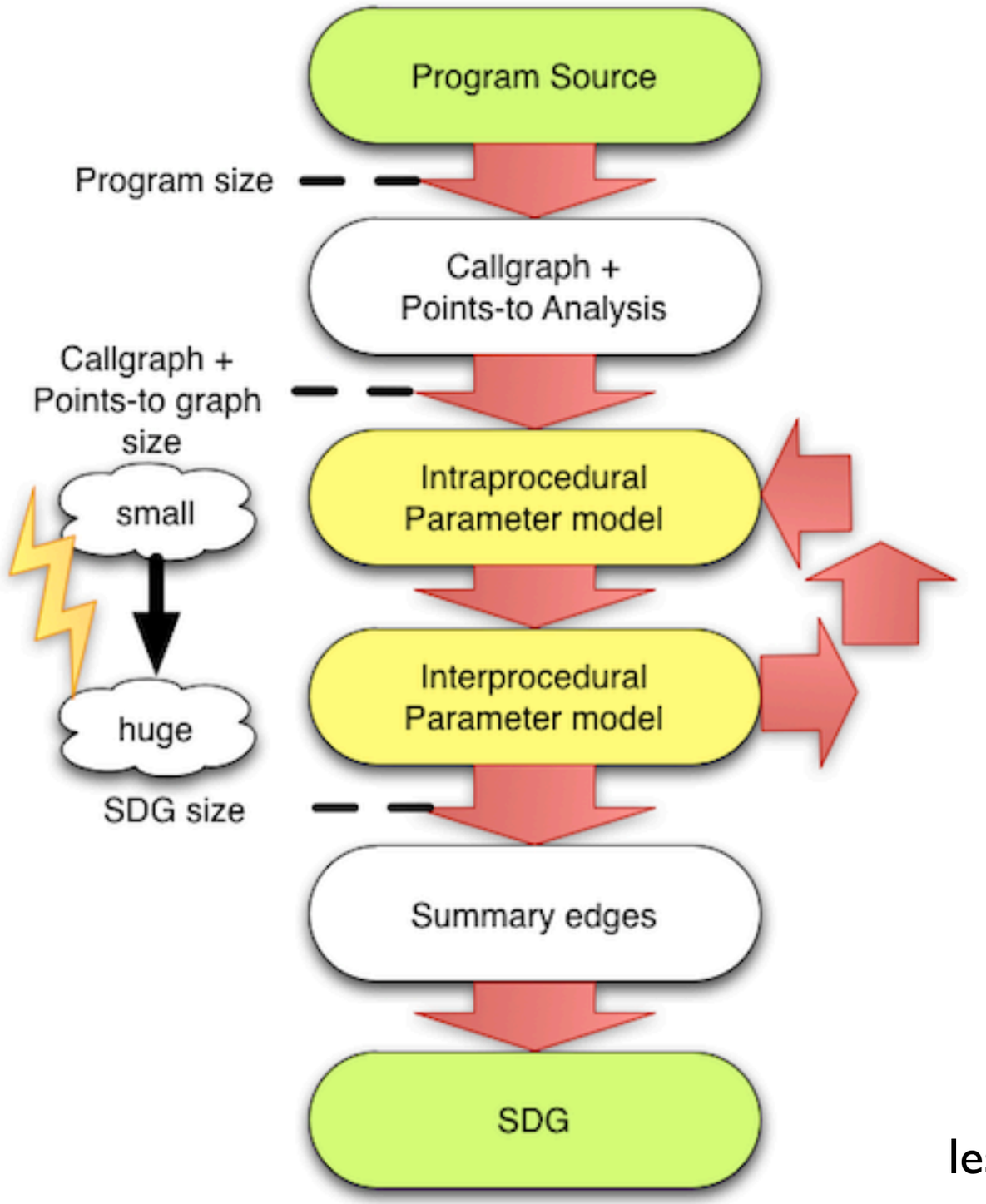
- **Parameter nodes**
  - Read (input) or modified (output) values of a method
  - For each **Parameter** and **Field** of a parameter (**many**!)
  - **Object Trees**: Tree structure for a parameter and its fields
- Points-to / May-alias Analysis

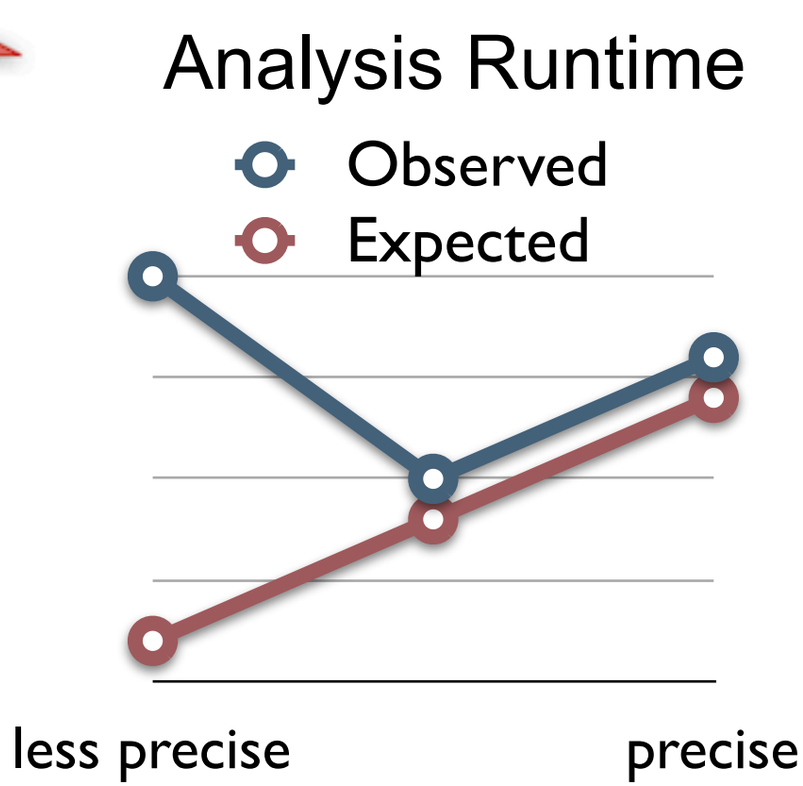


SDG computation

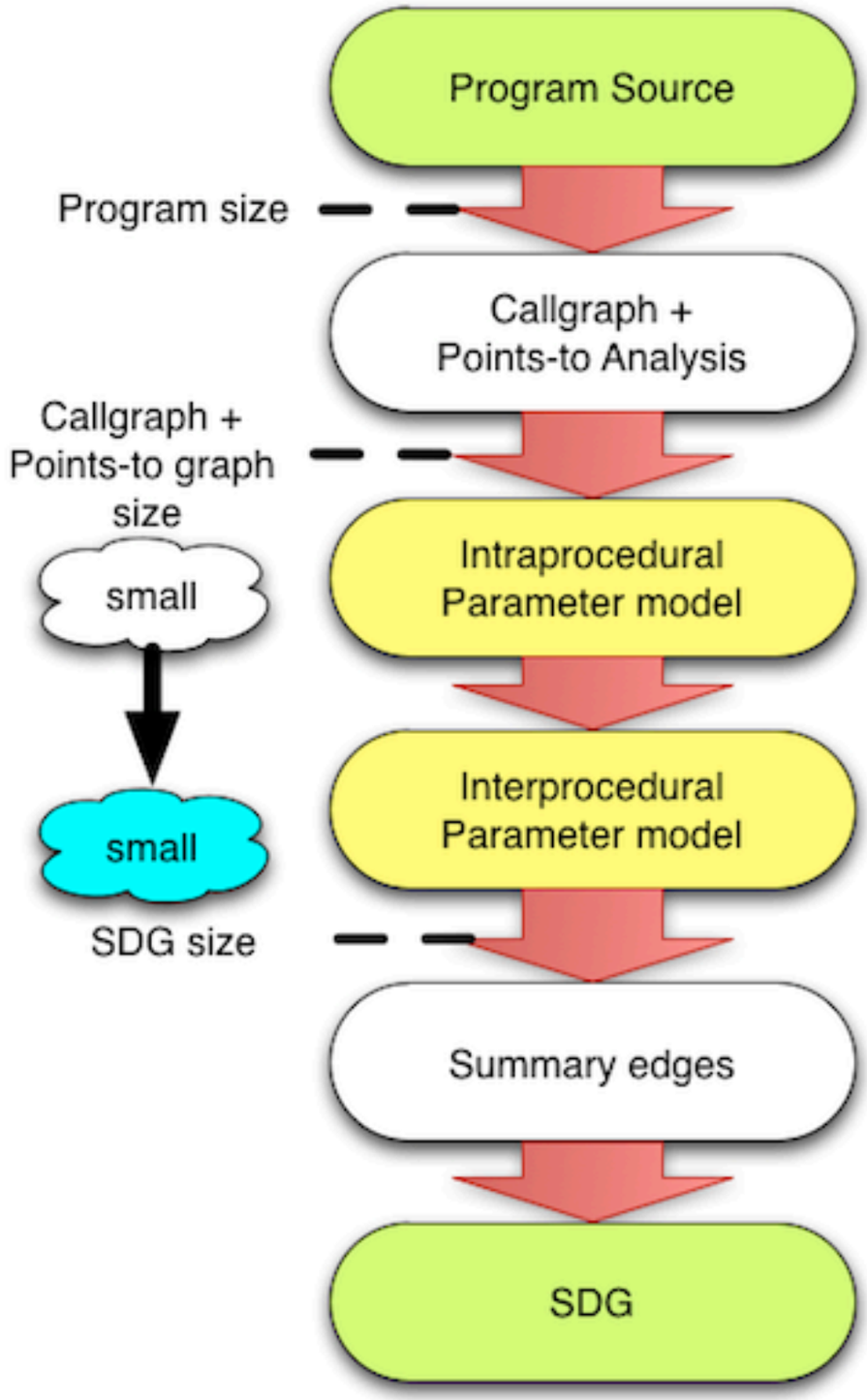


SDG computation with Object Trees





New Object Graph approach

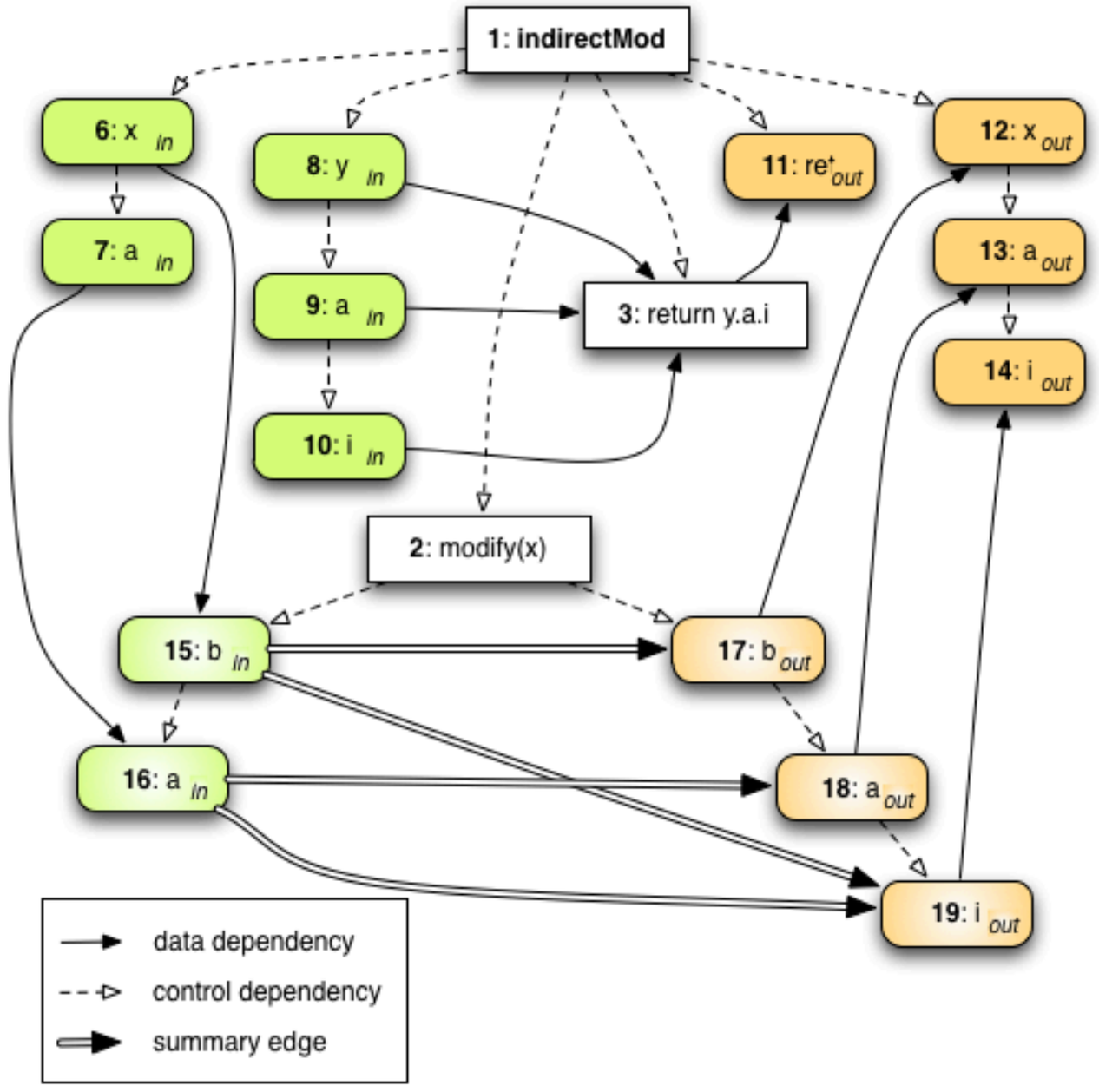


remove scalability problem

maintain precision

SDGs for object oriented programs

```
void main(String argv[]) {
  indirectMod(new B(), new B())
}


int indirectMod(B x, B y) {
  modify(x);
  return y.a.i;
}

void modify(B b) {
  b.a.i = 42;
}

class B { A a = new A(); }
```

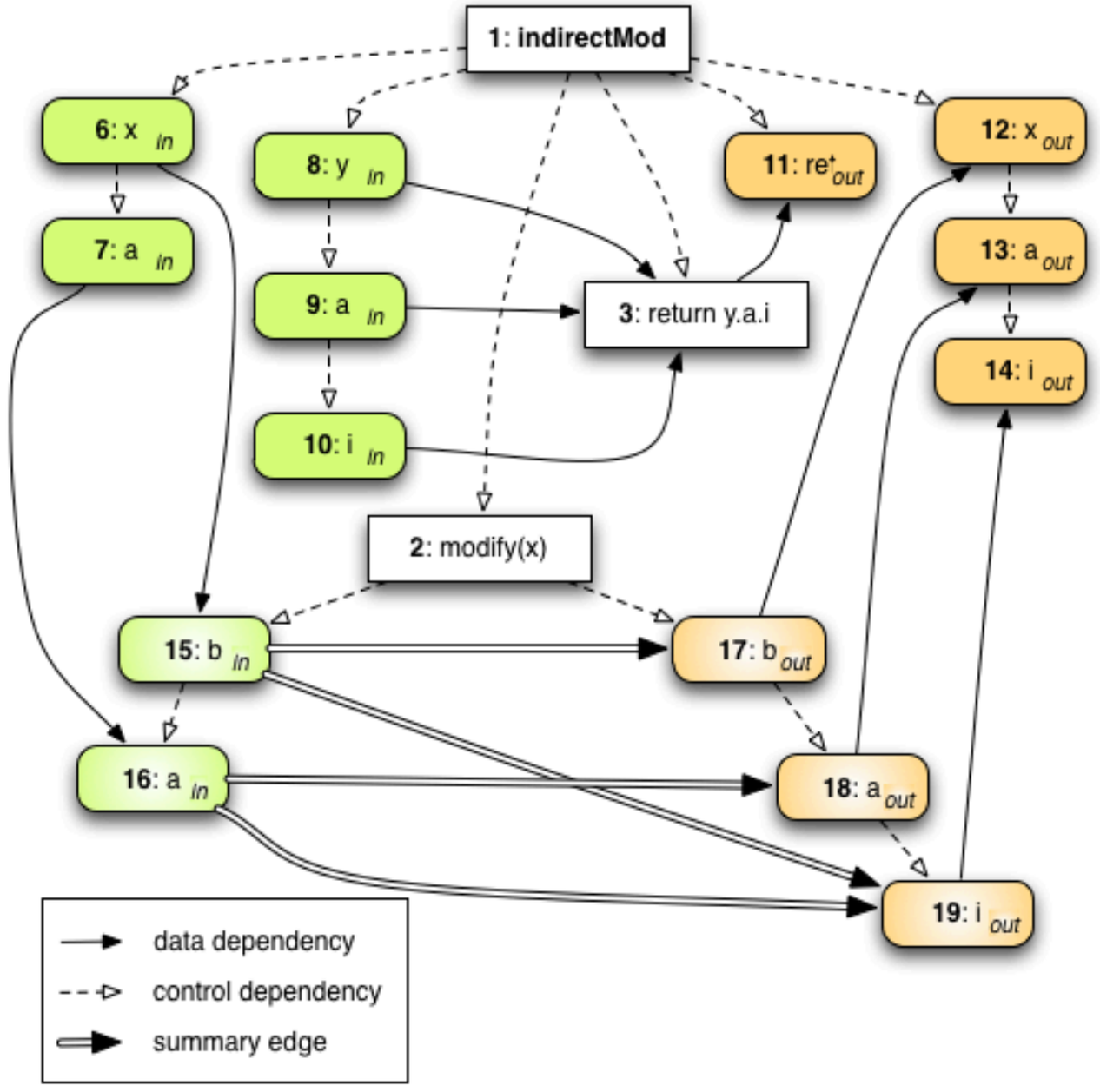


Object Trees for method parameters

```
void main(String argv[]) {
  indirectMod(new B(), new B());
}

int indirectMod(B x, B y) {
  modify(x);
  return y.a.i;
}

void modify(B b) {
  b.a.i = 42;
}

class B { A a = new A(); }
```

***precise** points-to*

⇒ x.a ≠ y.a



Object Trees for method parameters

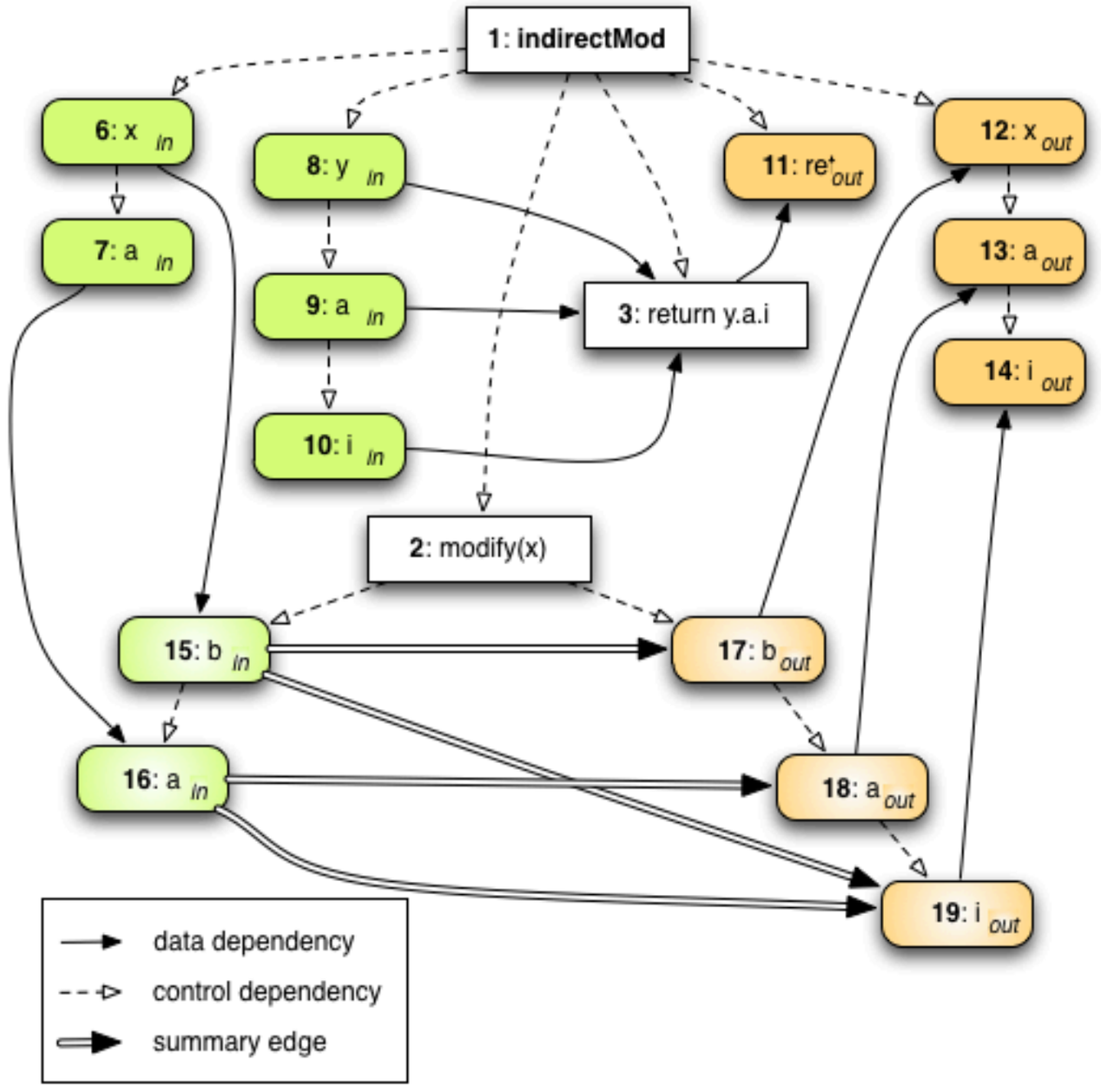
```
void main(String argv[]) {
  indirectMod(new B(), new B())
}

int indirectMod(B x, B y) {
  modify(x);
  return y.a.i;
}

void modify(B b) {
  b.a.i = 42;
}

class B { A a = new A(); }
```

***precise** points-to*

⇒ x.a ≠ y.a

modifies x.a.i

reads y.a.i

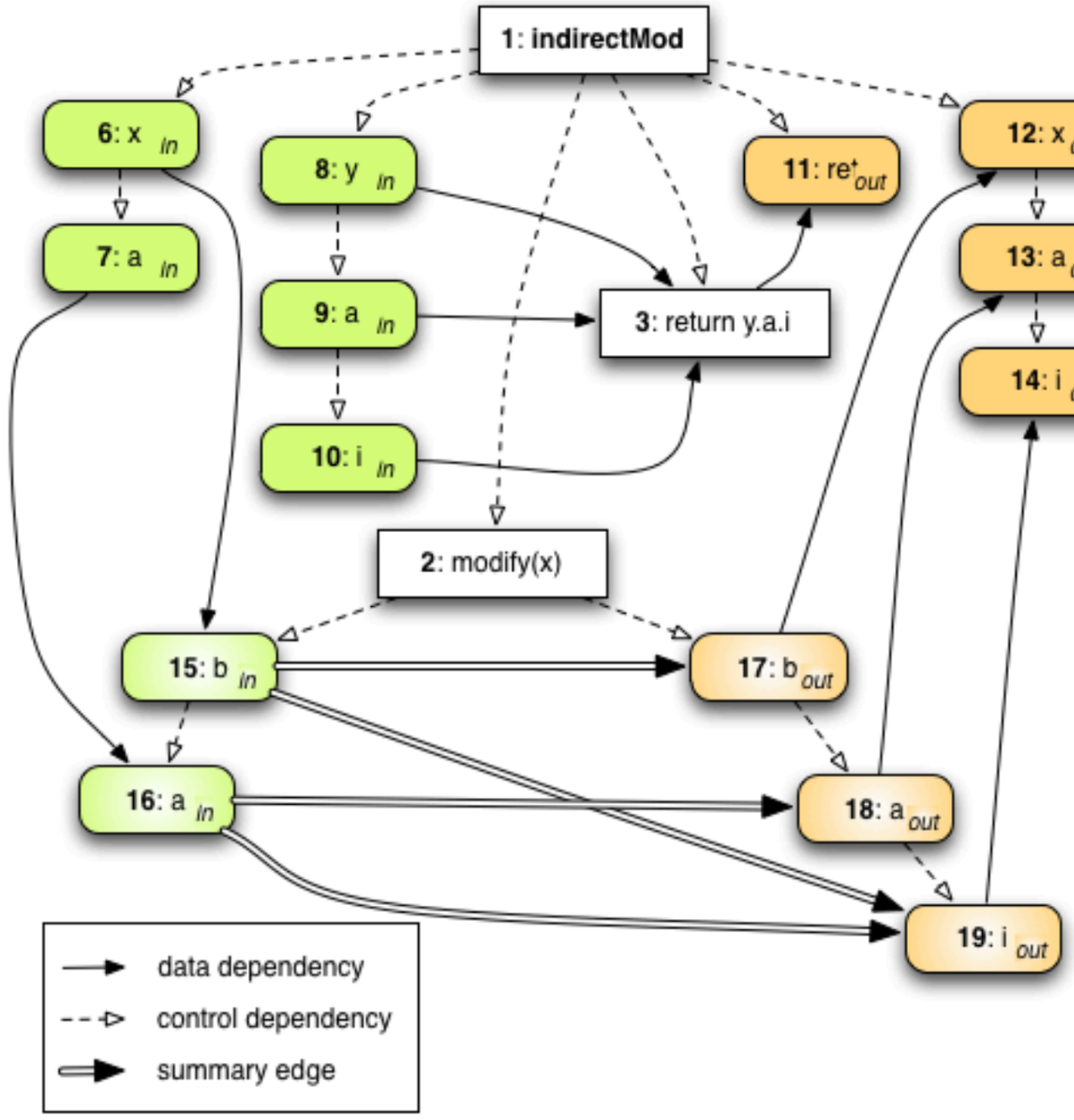


Object Trees for method parameters

```
void main(String argv[]) {
  indirectMod(new B(), new B())
}

int indirectMod(B x, B y) {
  modify(x);
  return y.a.i;
}

void modify(B b) {
  b.a.i = 42;
}

class B { A a = new A(); }
```

less precise *points-to*

⇒ x.a **?** y.a

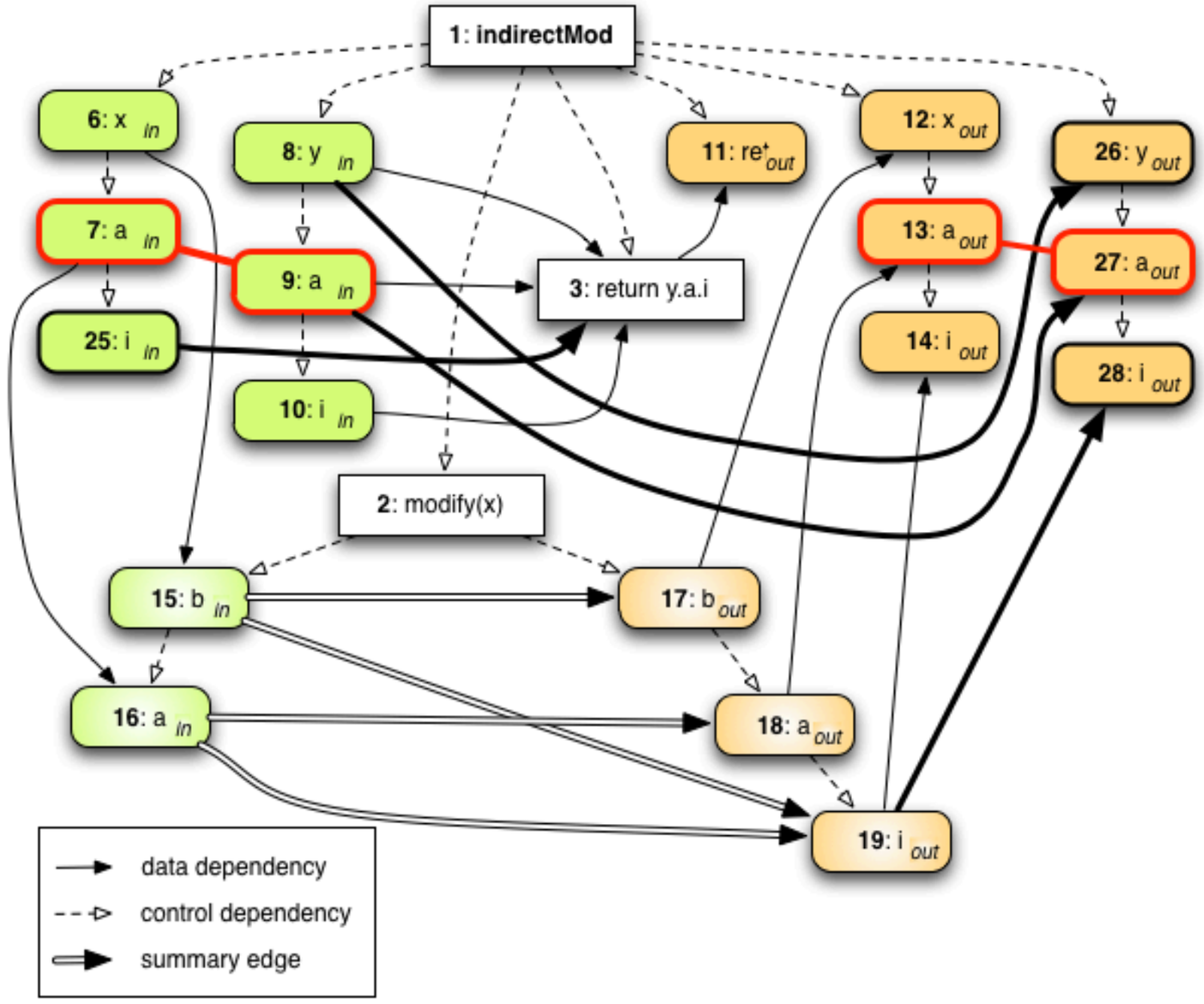


Object Tree grows with less precision

```
void main(String argv[]) {
  indirectMod(new B(), new B())
}

int indirectMod(B x, B y) {
  modify(x);
  return y.a.i;
}

void modify(B b) {
  b.a.i = 42;
}

class B { A a = new A(); }
```

less precise *points-to*

⇒ x.a **?** y.a



Object Tree grows with less precision

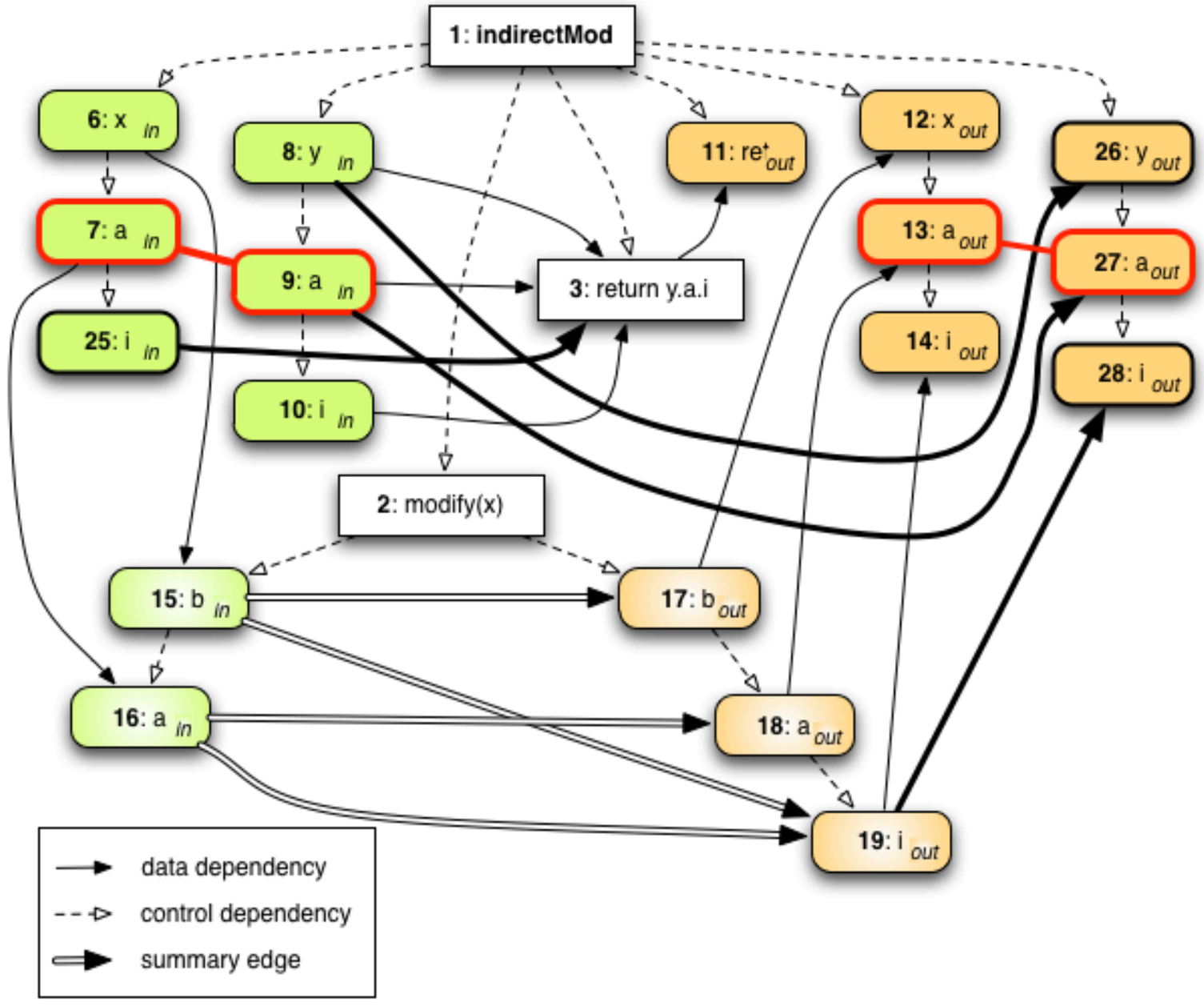
```
void main(String argv[]) {
  indirectMod(new B(), new B())
}

int indirectMod(B x, B y) {
  modify(x);
  return y.a.i;
}

void modify(B b) {
  b.a.i = 42;
}

class B { A a = new A(); }
```

less precise points-to

⇒ x.a **?** y.a

*modifies [**x**,**y**].a.i*

*reads [**x**,**y**].a.i*



Object Graphs share indistinguishable fields

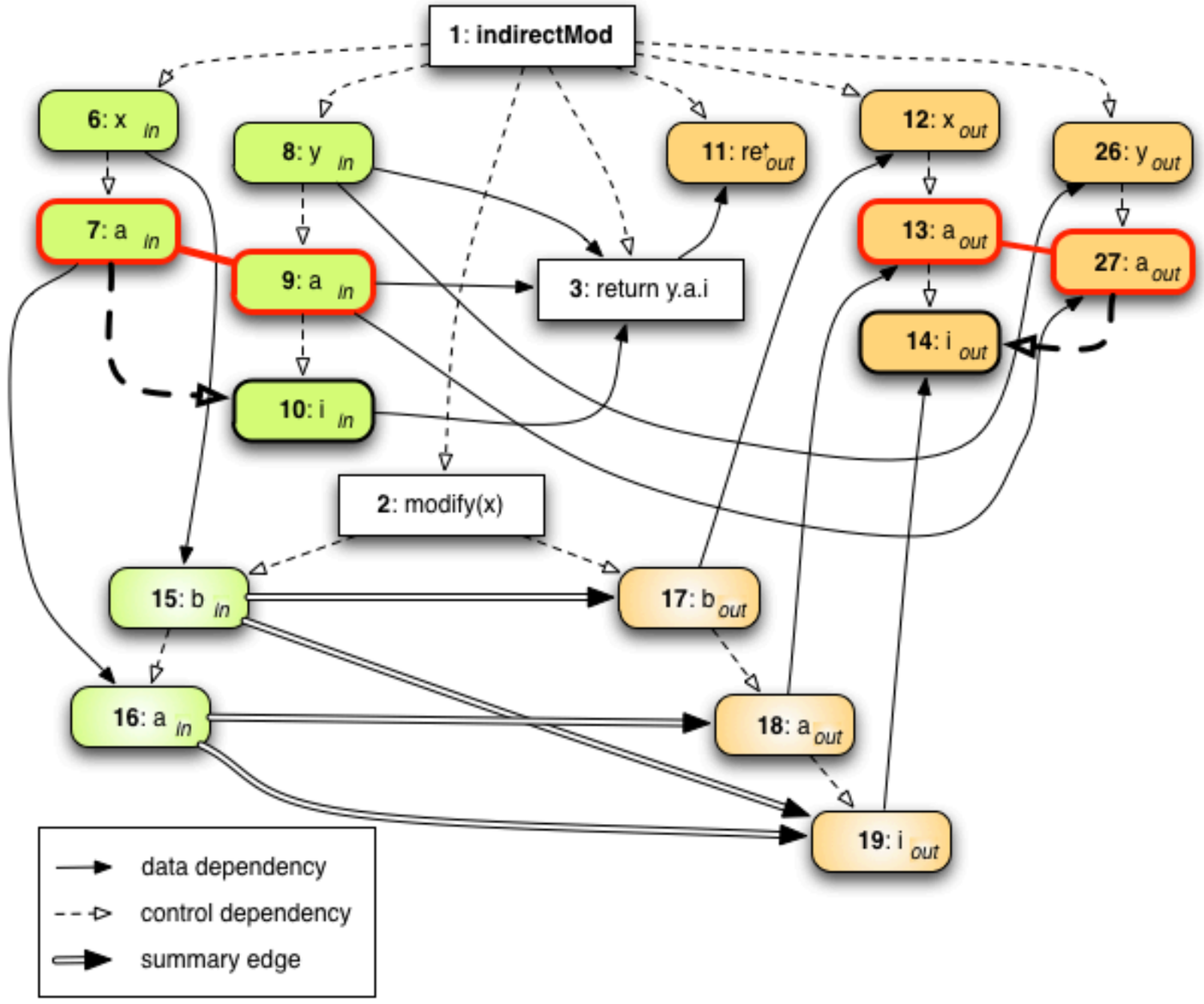
```
void main(String argv[]) {
  indirectMod(new B(), new B())
}

int indirectMod(B x, B y) {
  modify(x);
  return y.a.i;
}

void modify(B b) {
  b.a.i = 42;
}

class B { A a = new A(); }
```

less precise points-to

⇒ x.a **?** y.a

*modifies [**x**,**y**].a.i*

*reads [**x**,**y**].a.i*



Implementation and Evaluation

SDG Generator using WALA Framework (wala.sf.net)

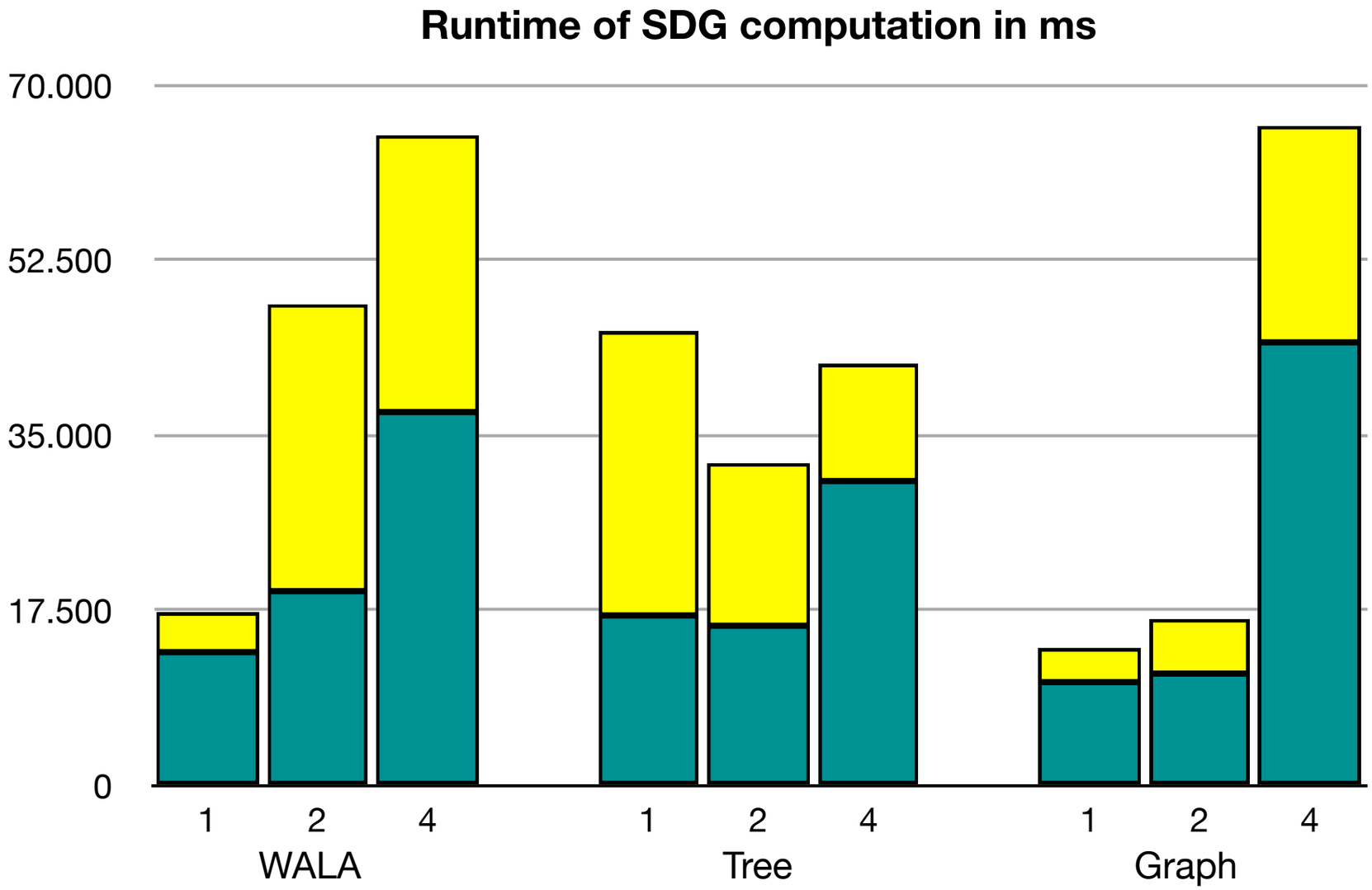
- **3 Parameter Models**
  - **Object Tree**
  - **Object Graph**
    - Optional escape analysis
    - Optimizations for interprocedural parameter node propagation
  - **WALA approach** (unstructured ⇒ less precision)

- **4 Points-to Analyses** (varying in precision)

Evaluation

- 20 programs from 120 - 63.000 LoC
- Many larger ones only with Object Graphs

Runtime


Runtime of SDG computation in ms
70.000
52.500
35.000
17.500
0
1
2
4
WALA
Tree
Graph
Computation phase:
Summary edge
SDG with parameters
Points-to Analysis:
1 Type based
2 Instance based
4 Object sensitive

Conclusion

- Parameter model has huge impact on runtime
- Object Trees suffer from scalability deadlock
- **Object Graphs**
  - Less nodes for less precision ⇒ **larger programs**
  - Less nodes for larger programs
  - **Maintain precision** of Object Trees
  - High precision points-to
    - Use Object Trees
    - Remove propagation optimization
- Additional results
  - Exception flow has a huge impact on precision
  - Precision benefit from points-to analysis varies
  - Advantage of structured parameter models: access path
    - No parameter nodes for side-effects that can not escape

The End

Karlsruhe Institute of Technology

Questions?

Statement

Static analyses will never be as precise and fast as dynamic analysis techniques. Why are we still trying?