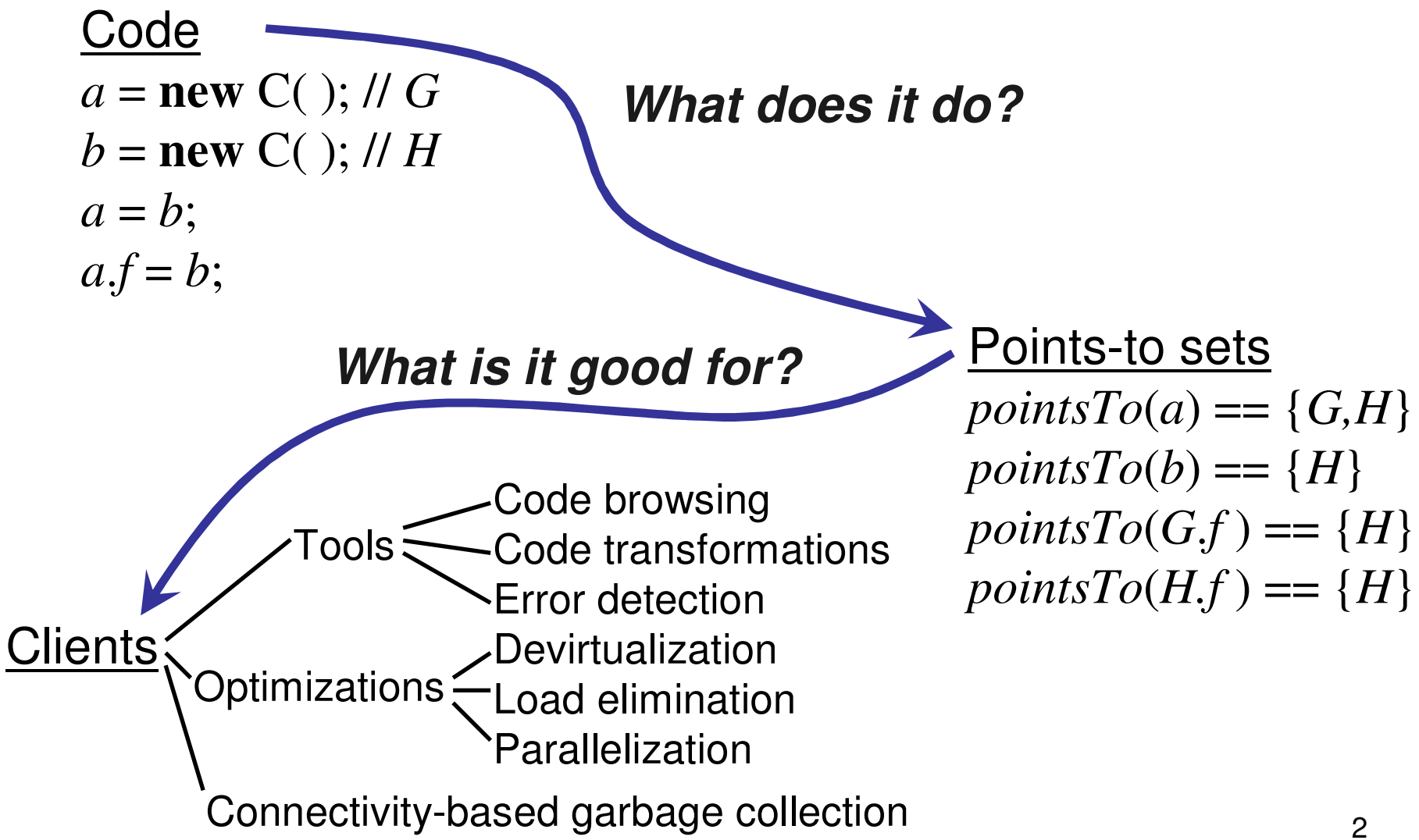
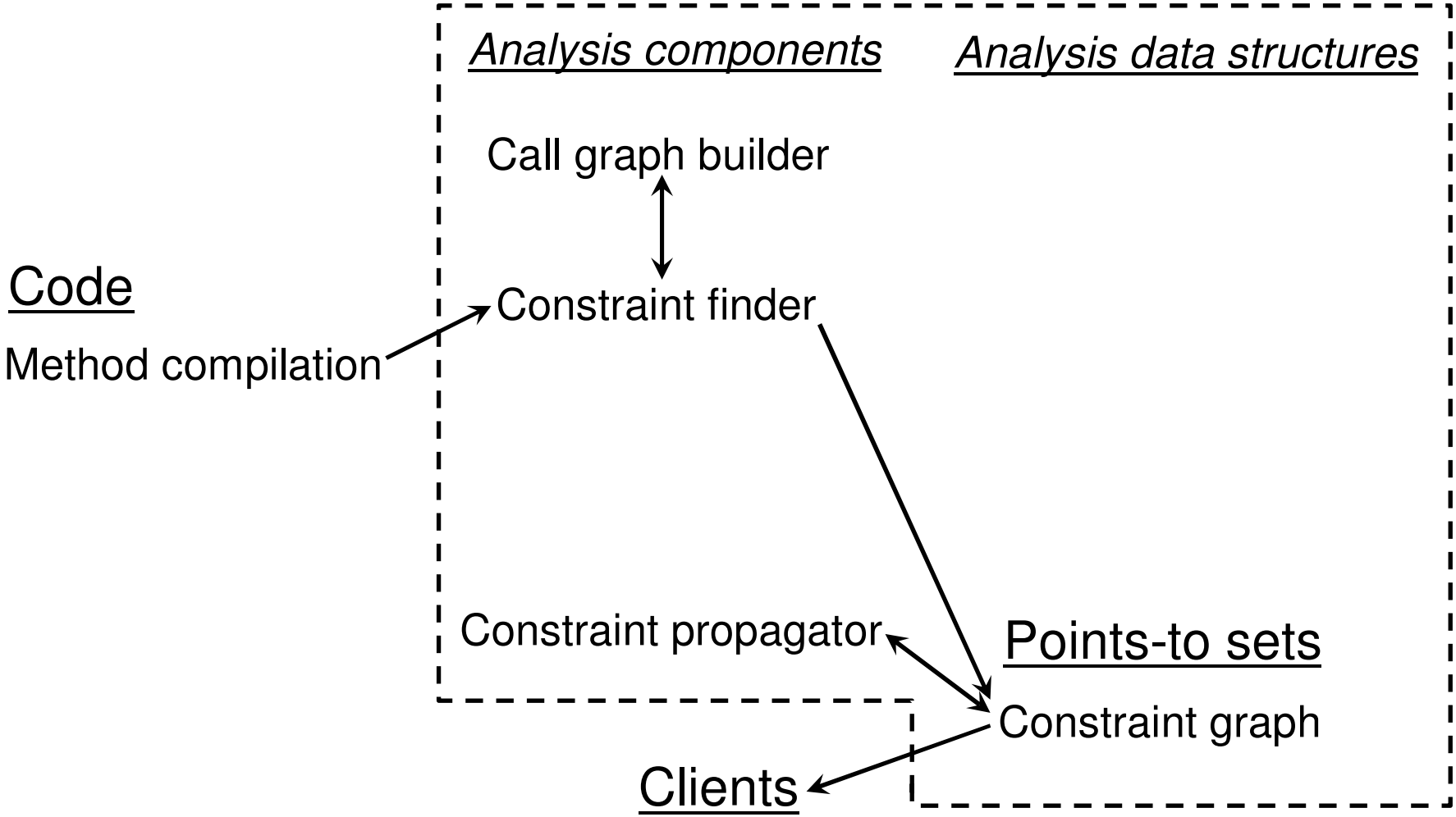
Pointer Analysis in the Presence of Dynamic Class Loading

Martin Hirzel, Amer Diwan
University of Colorado at Boulder
Michael Hind
IBM T.J. Watson Research Center

Pointer analysis motivation



Static flow- and context-insensitive pointer analysis by Andersen

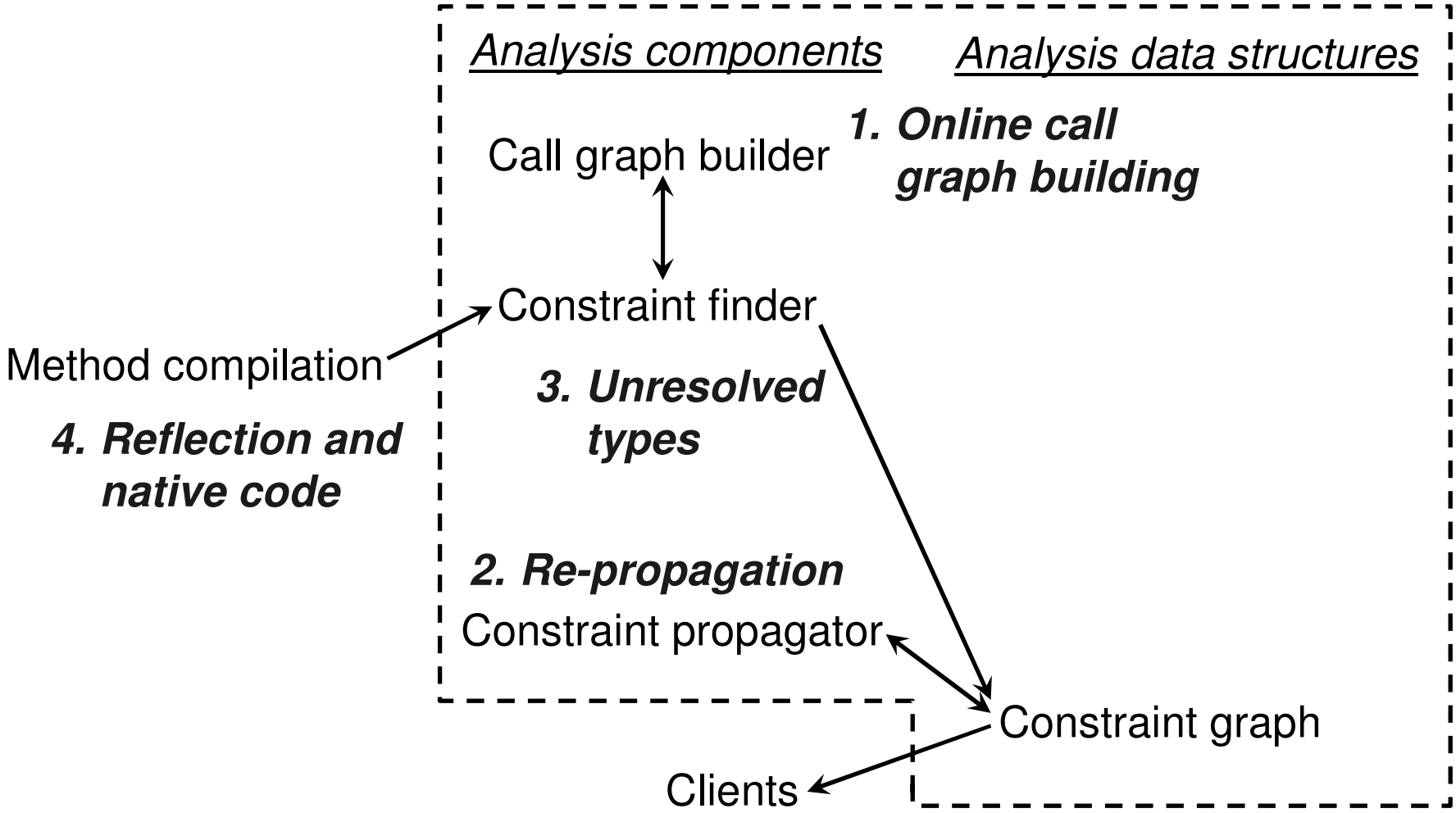


Static analysis can not deal with all of Java

- Class loading may be implicitly triggered by any …
  - Constructor call
  - Static field access
  - Static method call
- Classes may come from the web or be generated on the fly

➔ ***Pretending a "static world" fails for most real-world applications***

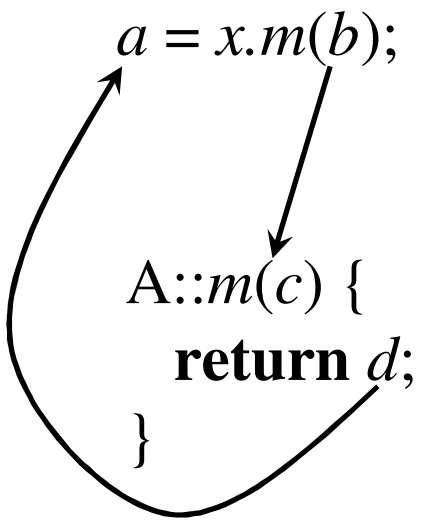
Java challenges



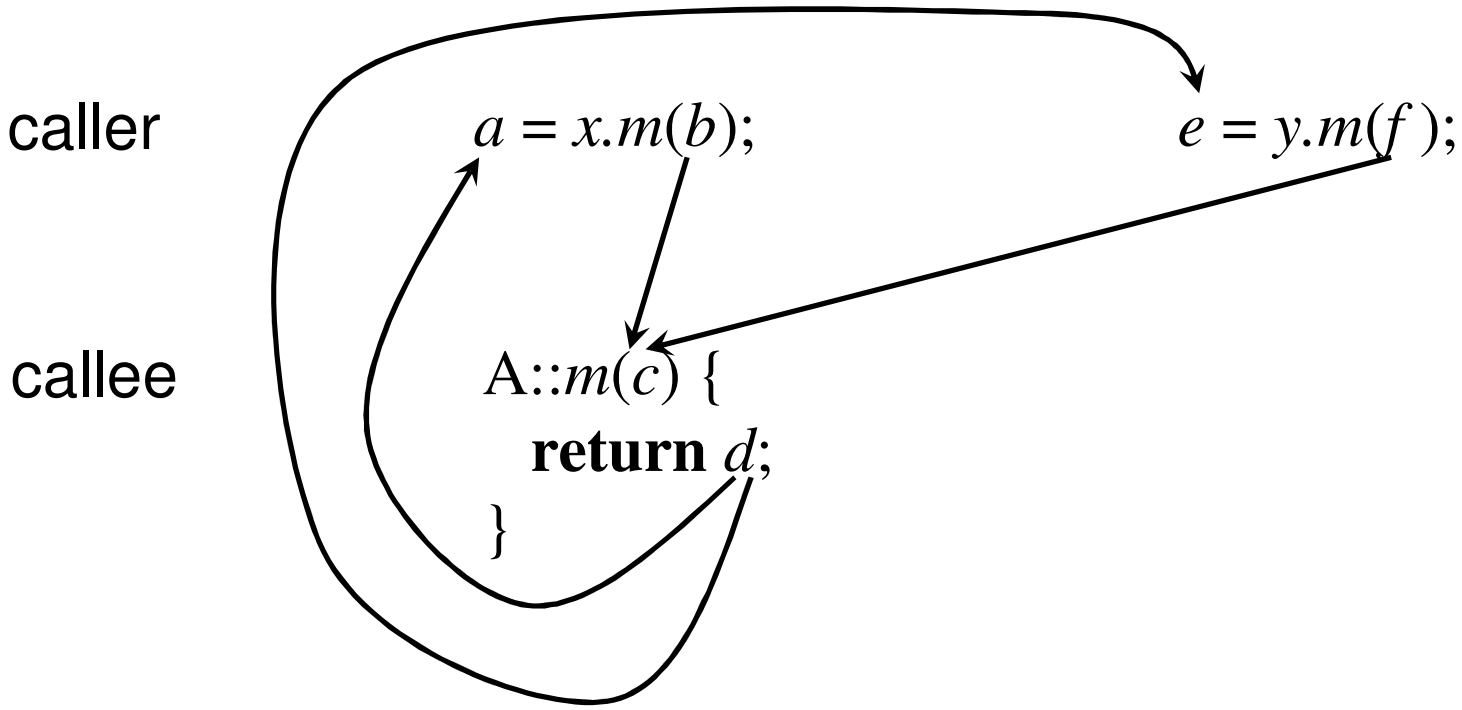
1. Online call graph building

caller

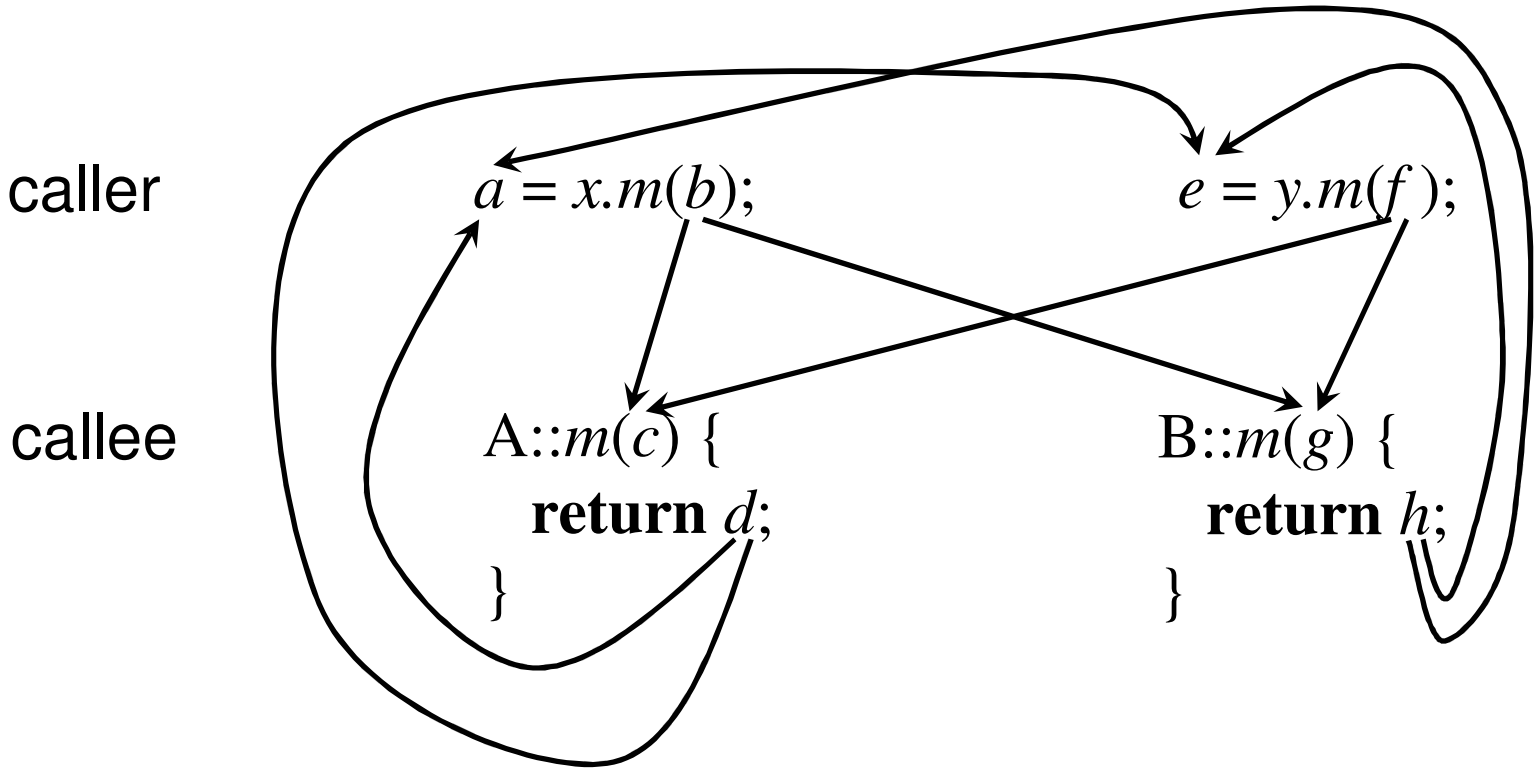
callee



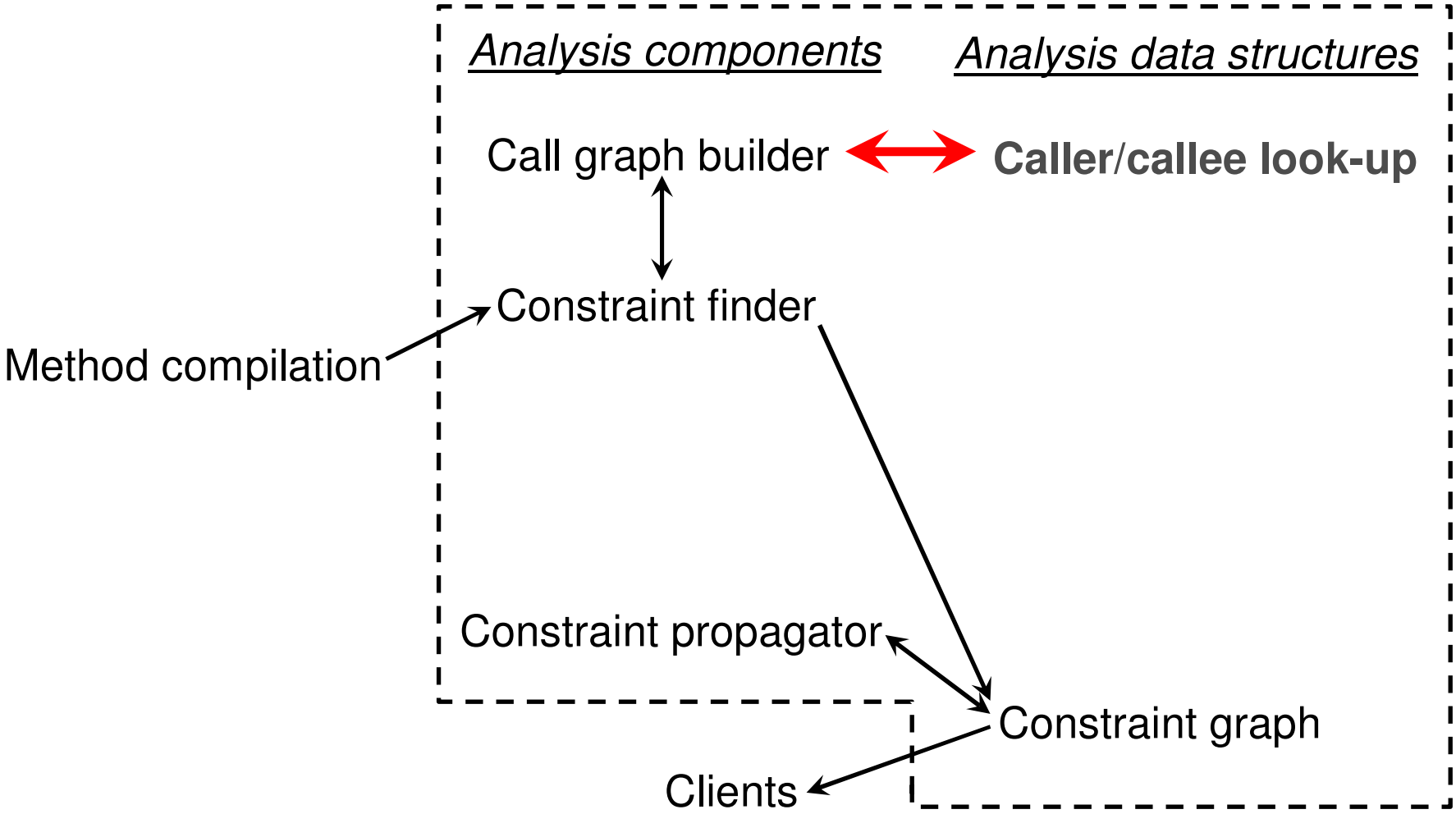
1. Online call graph building



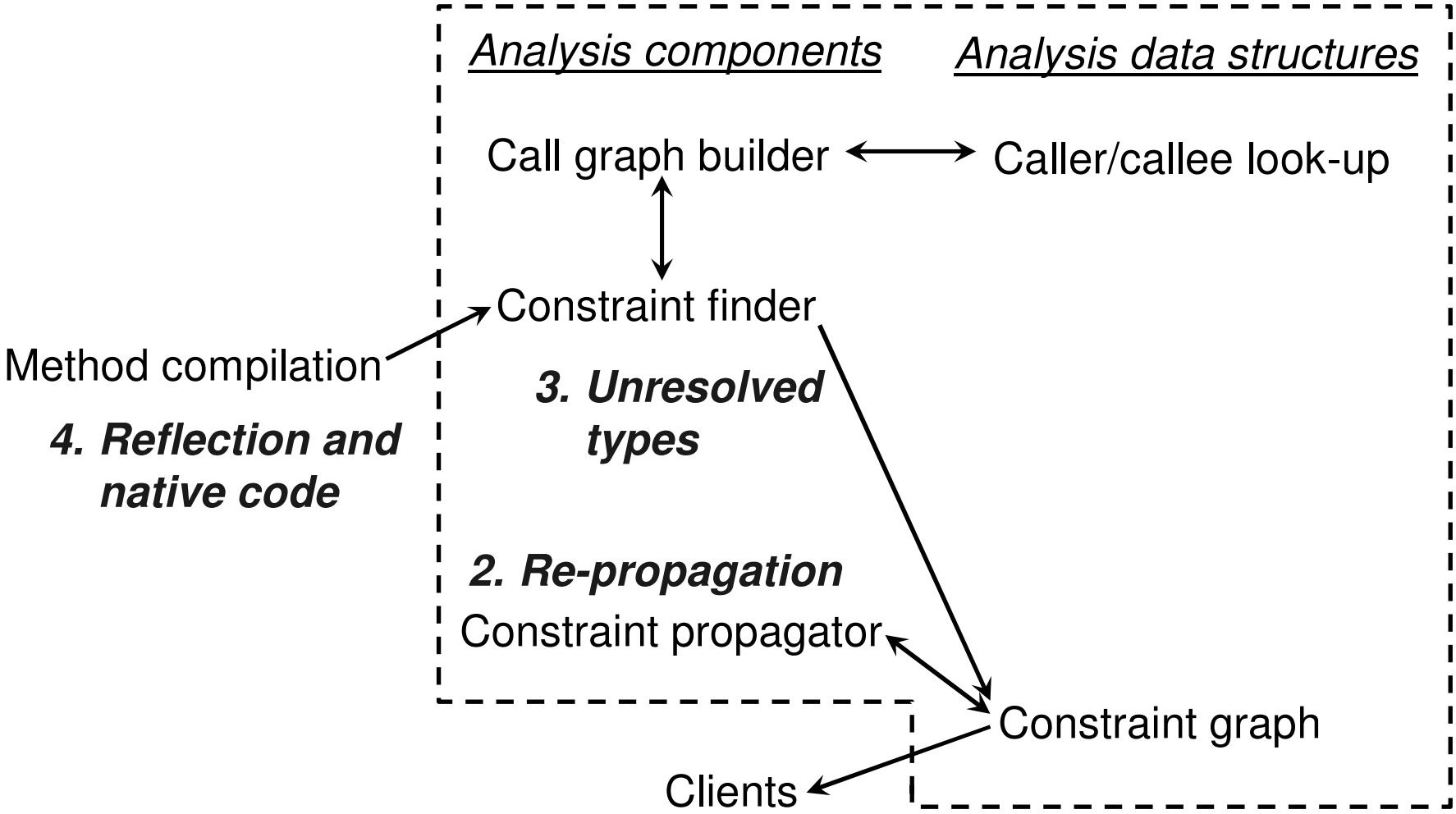
1. Online call graph building



Architecture for online call graph building



Java challenges



2. Focused re-propagation

Code

a = **new** C( ); // G

b = **new** C( ); // H

$a.f = b$;

a = b;

Points-to sets

$pointsTo(a) == \{G\}$

$pointsTo(b) == \{H\}$

$pointsTo(G.f) == \{H\}$

$pointsTo(H.f) == \{\ \}$

2. Focused re-propagation

Code

a = **new** C( ); // G
b = **new** C( ); // H
$a.f = b$;

a = b;

Points-to sets

$pointsTo(a) == \{G,H\}$
$pointsTo(b) == \{H\}$
$pointsTo(G.f) == \{H\}$
$pointsTo(H.f) == \{\ \}$

2. Focused re-propagation

Code

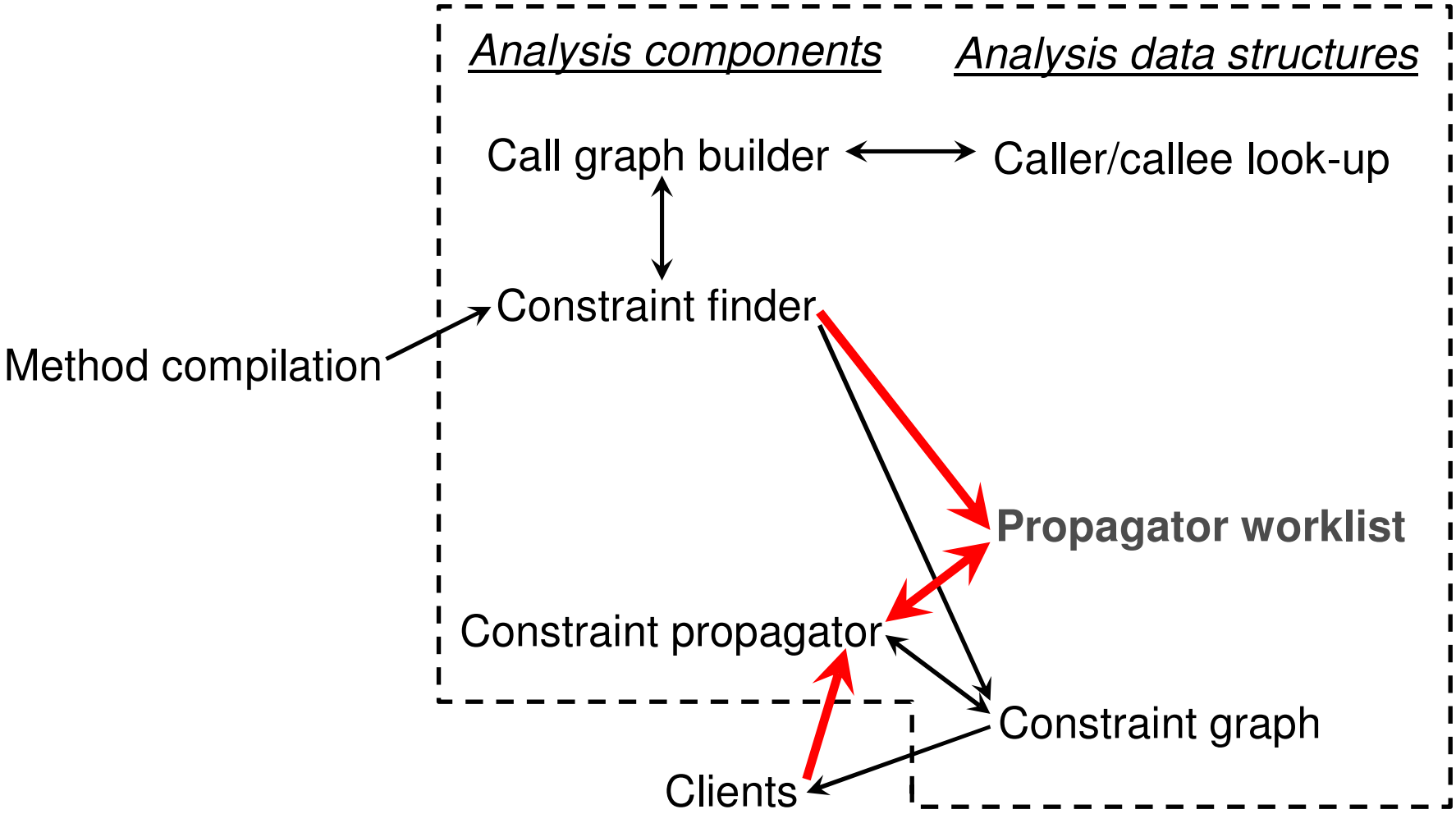
a = **new** C( ); // G
b = **new** C( ); // H
$a.f = b$;

a = b;

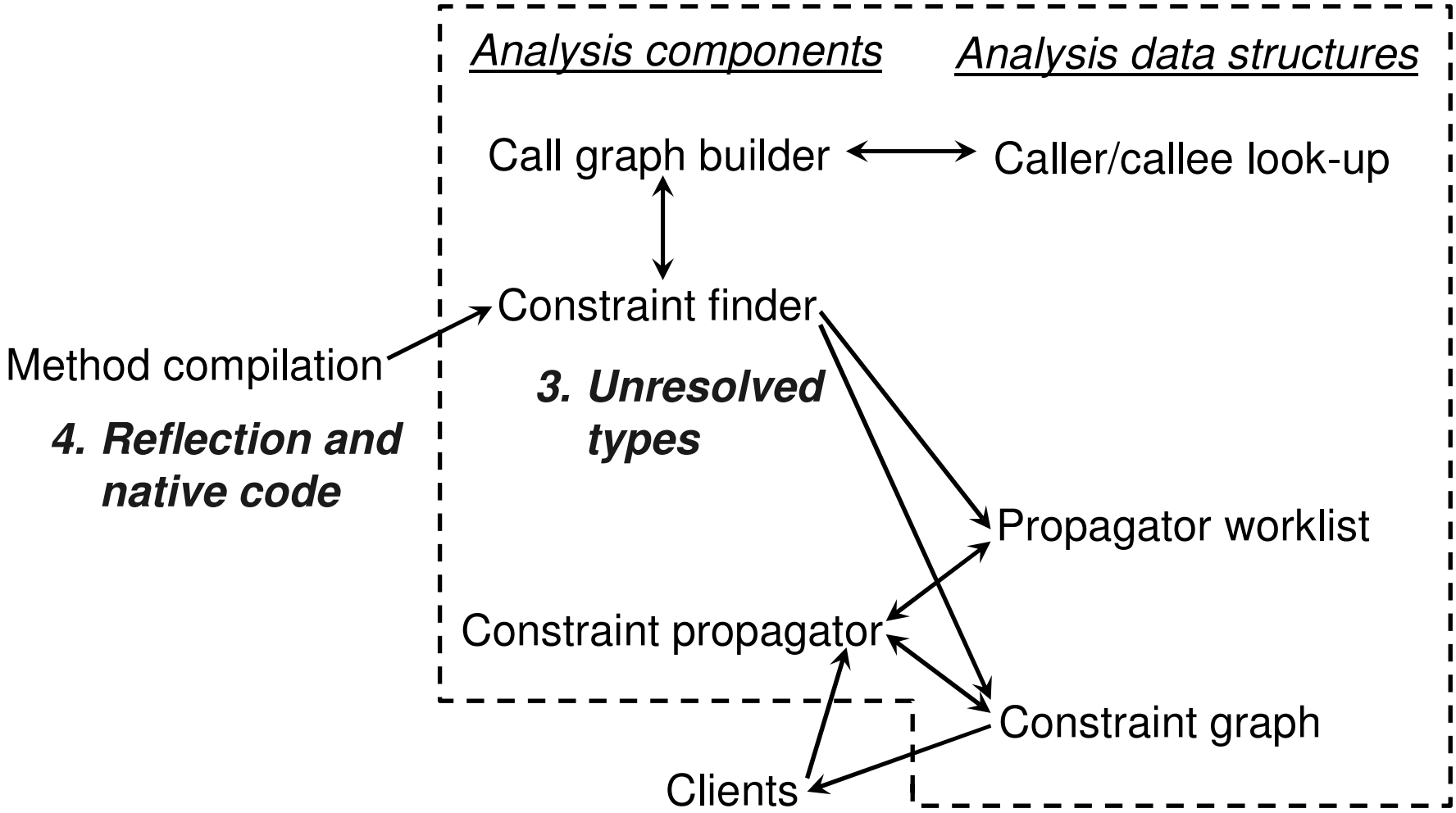
Points-to sets

$pointsTo(a) == \{G,H\}$
$pointsTo(b) == \{H\}$
$pointsTo(G.f) == \{H\}$
$pointsTo(H.f) == \{H\}$

Architecture for focused re-propagation



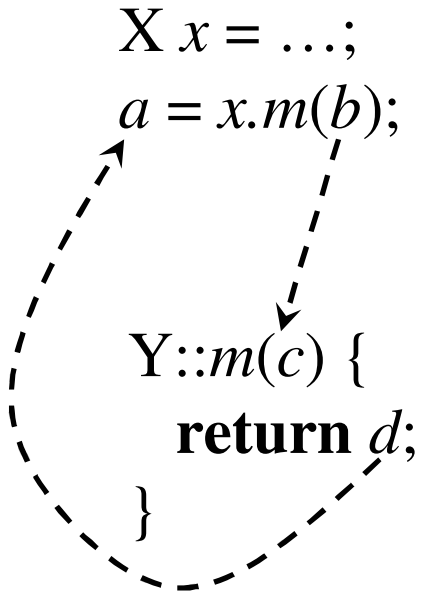
Java challenges



3. Unresolved types

caller

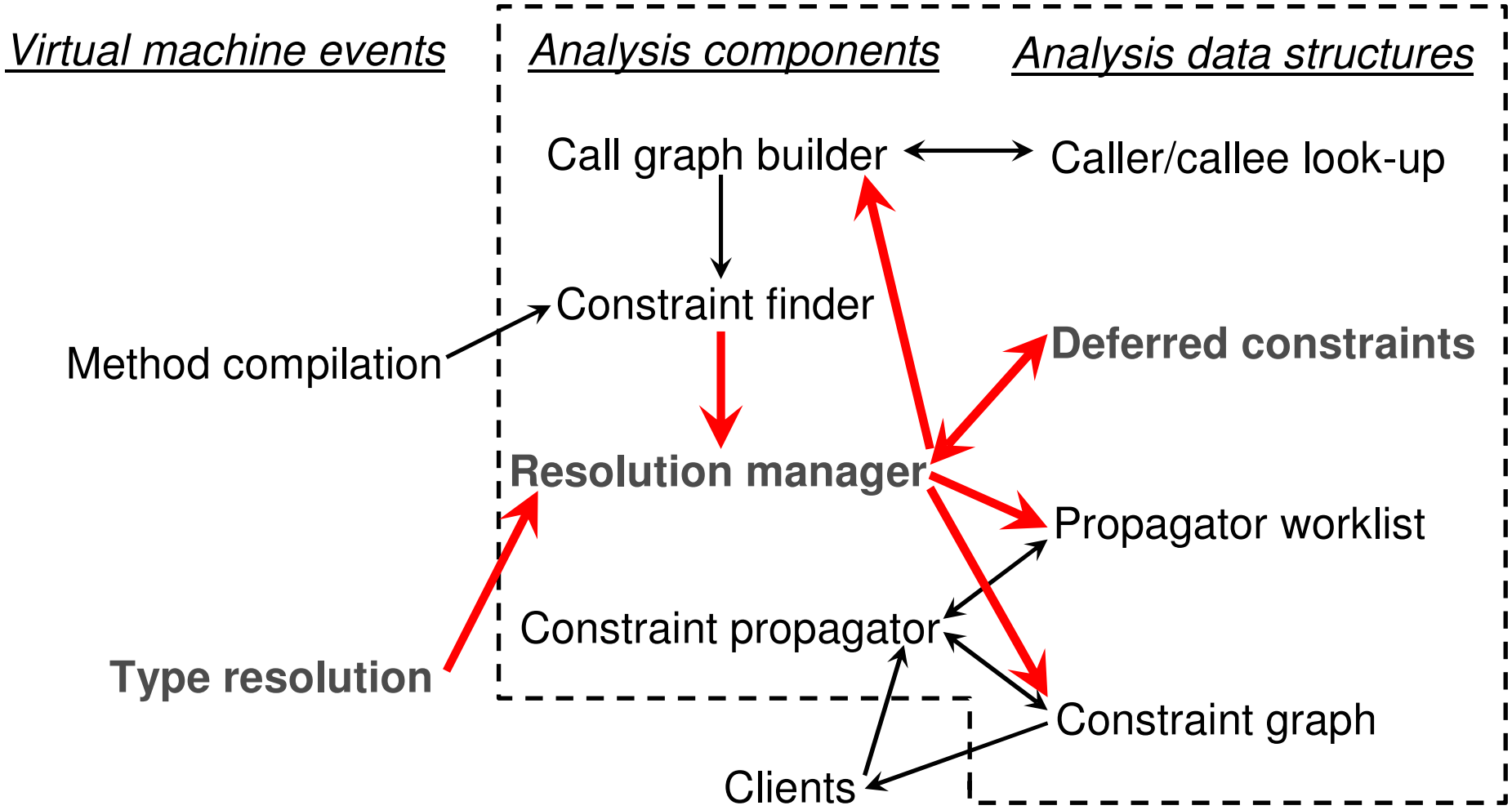
callee



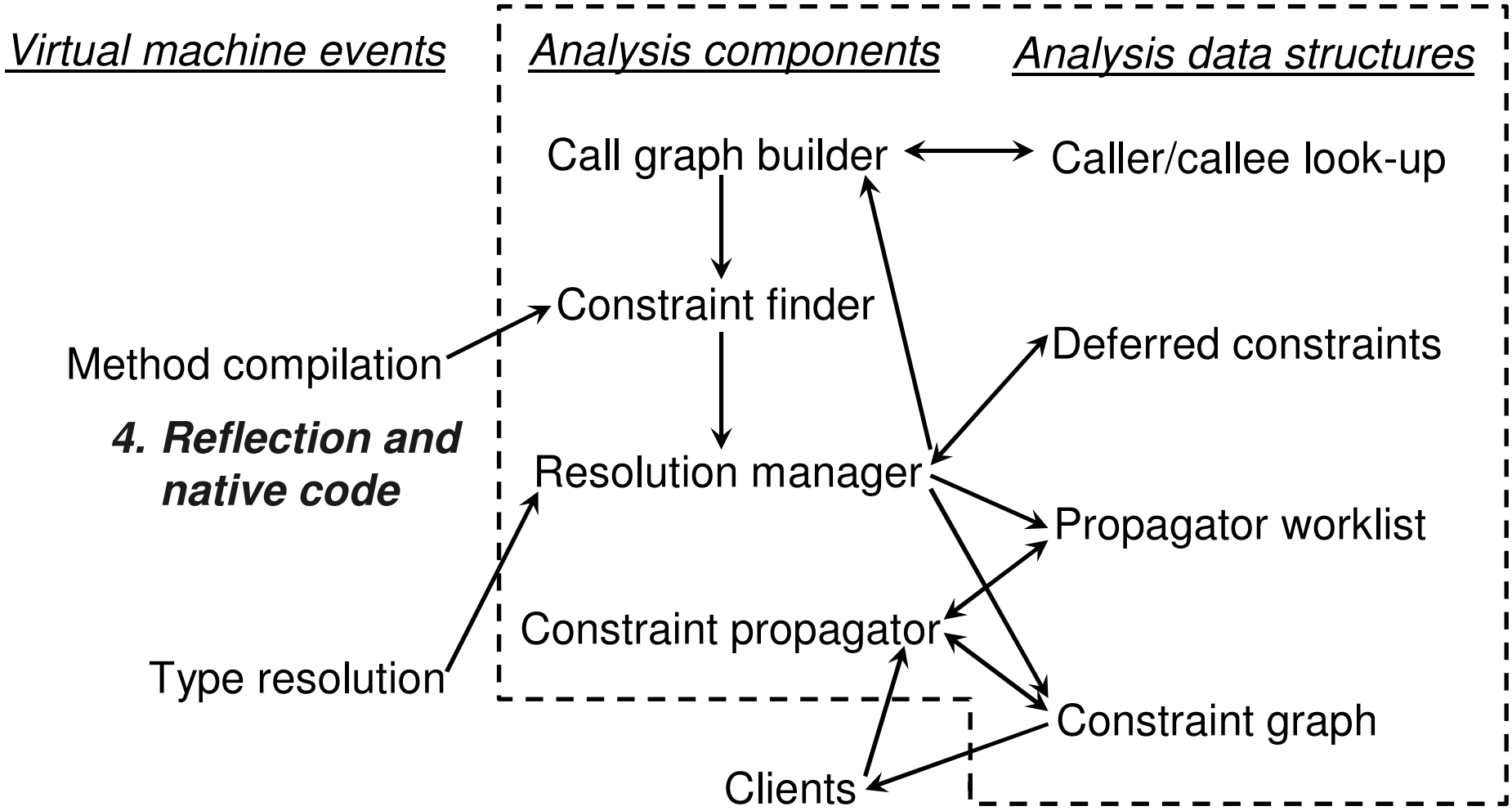
? Can X have a subclass that inherits *m* from Y?

! Cannot tell before X is resolved!

Architecture for managing unresolved types



Java challenges

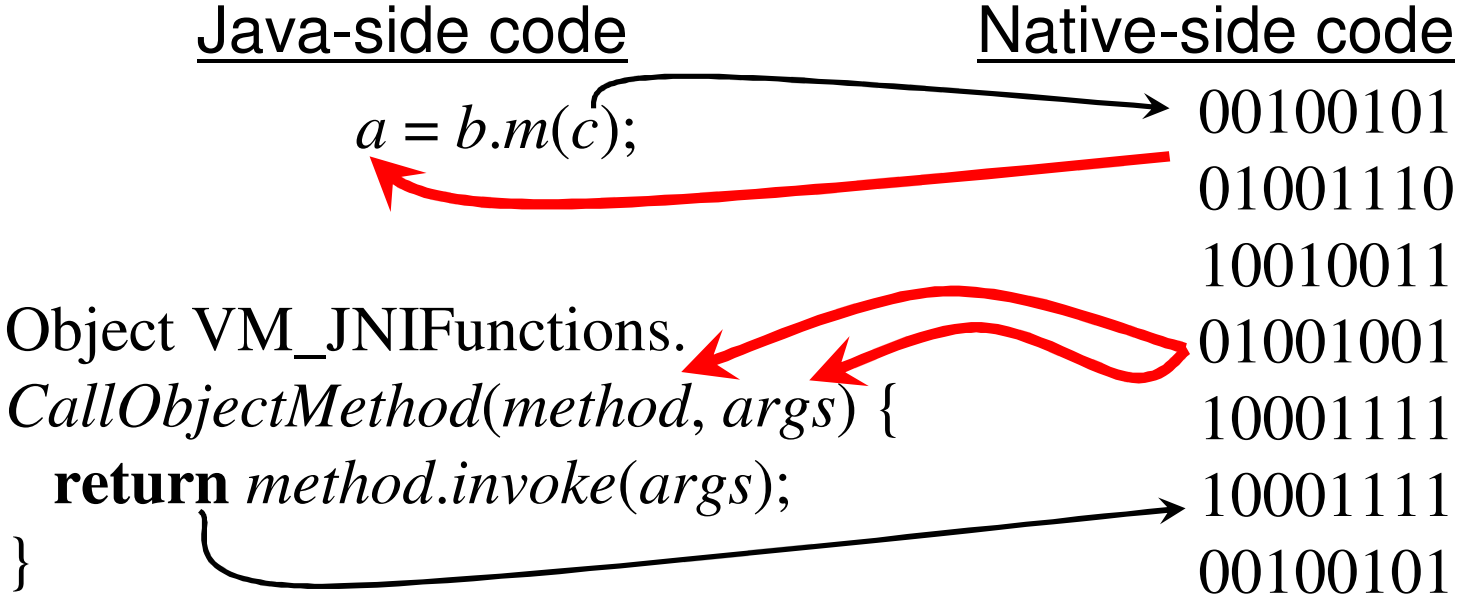


4. Reflection and native code

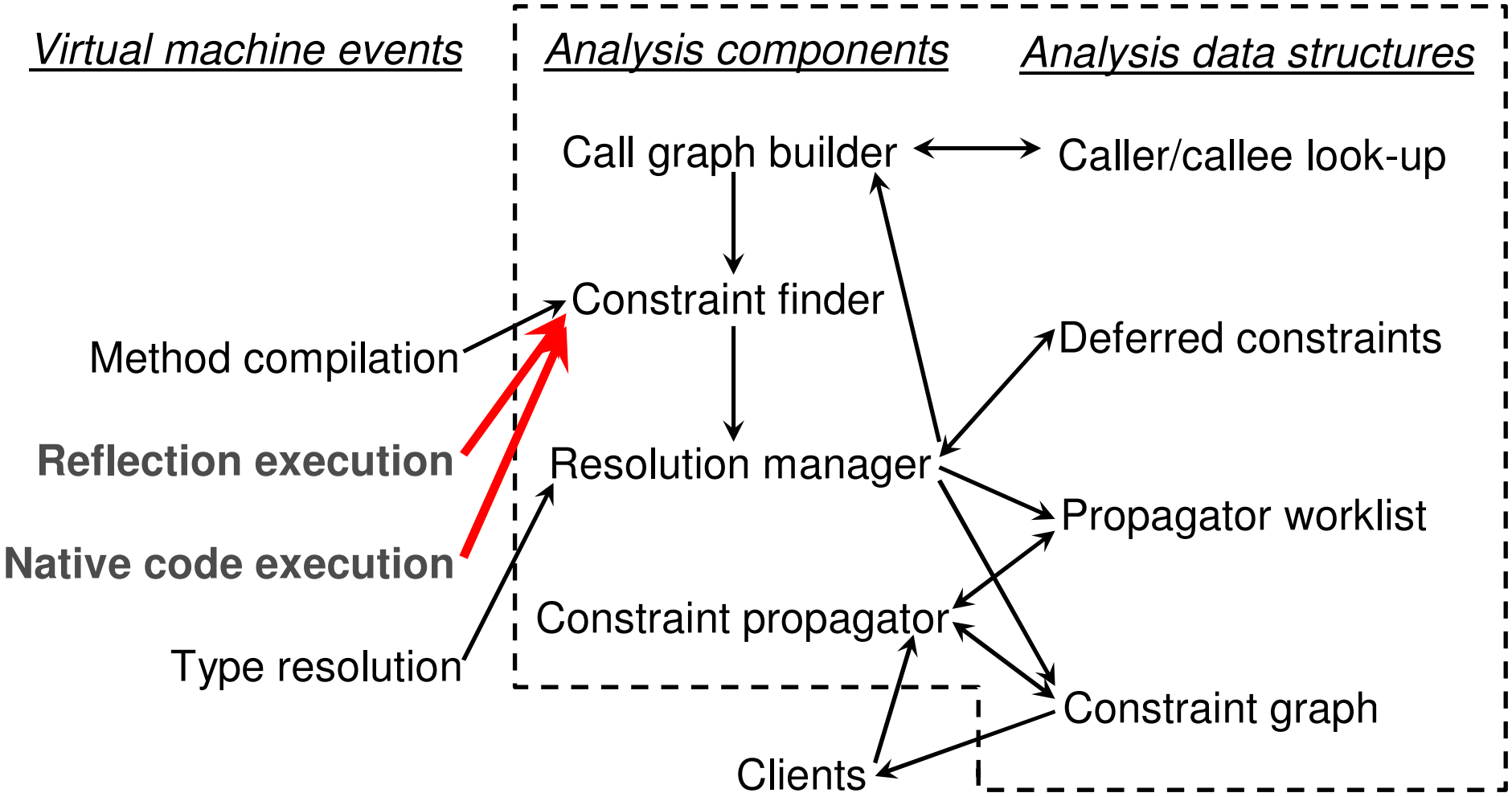
Reflection

```
Field f = B.class.getField("…");
B b = …;
f.set(b,v);
```

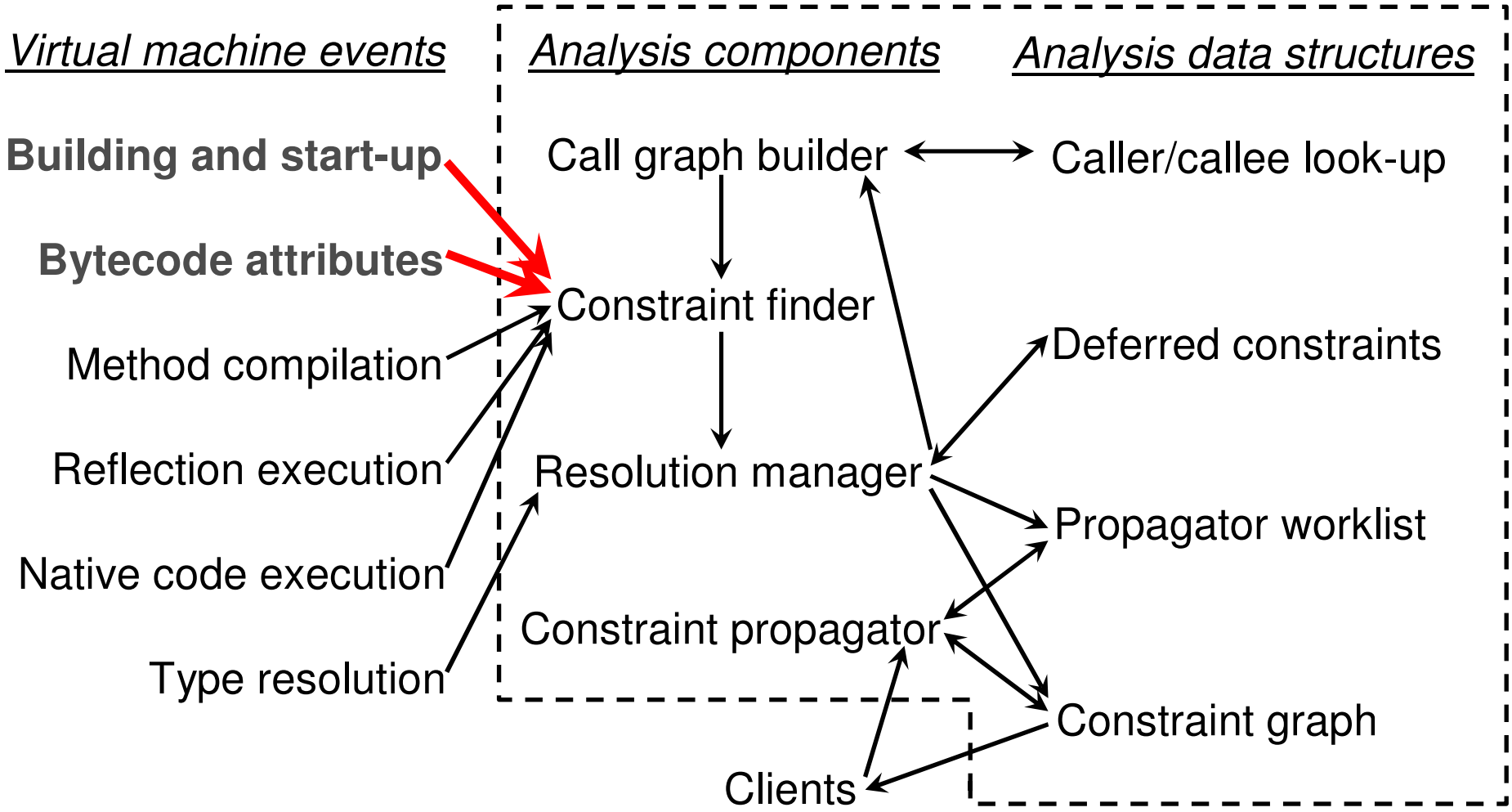
Native code



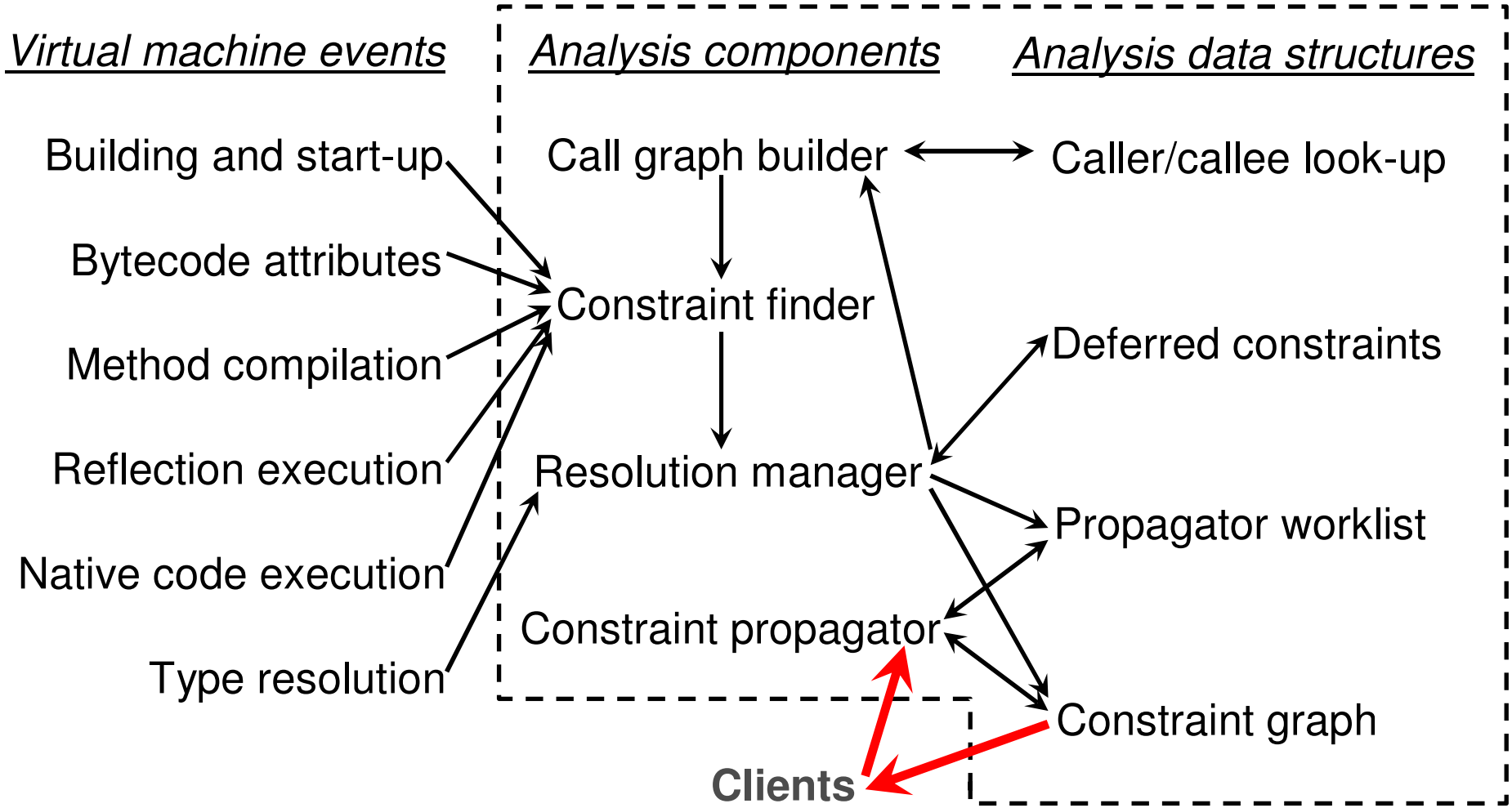
Architecture for dealing with reflection and native code



Other events leading to constraints



Clients using our pointer analysis



Dealing with invalidated results

Many techniques from prior work

- Guard optimized code (extant analysis)
- Pre-existence based inlining
- On-stack replacement
- and more

Connectivity-based garbage collection

- Trigger propagator only before collection
- Merge partitions if necessary

Evaluation methodology

Java virtual machine

- Jikes RVM from IBM, is itself written in Java

Benchmarks

- SPECjvm98 suite, xalan, hsql

Results not comparable to static analysis

- Analyze more code:
  Jikes RVM adds a lot of Java code
- Analyze less code:
  Not all application classes get loaded

Propagation cost

	Eager			At GC			At End
	Count	Avg.	Total	Count	Avg.	Total	Total
hsql	391	10.1s	1h06m	6	1m17s	7m40s	7m07s
jess	734	16.8s	3h26m	3	1m58s	5m53s	3m02s
javac	1,103	12.5s	3h50m	5	1m54s	9m32s	6m27s
xalan	1,726	11.2s	5h22m	1	2m01s	2m01s	7m45s

- ➔ ***Eagerness trades off average cost against total cost***
- ➔ ***On average, focused re-propagation is much cheaper than full propagation***
- ➔ ***Total cost is a function of code size and propagator eagerness***

How long does a program have to run to amortize the analysis cost?

	Eager			At GC			At End
	Count	Avg.	Total	Count	Avg.	Total	Total
hsql	391	10.1s	1h06m	6	1m17s	7m40s	7m07s
jess	734	16.8s	3h26m	3	1m58s	5m53s	3m02s
javac	1,103	12.5s	3h50m	5	1m54s	9m32s	6m27s
xalan	1,726	11.2s	5h22m	1	2m01s	2m01s	7m45s

Overall analysis overhead

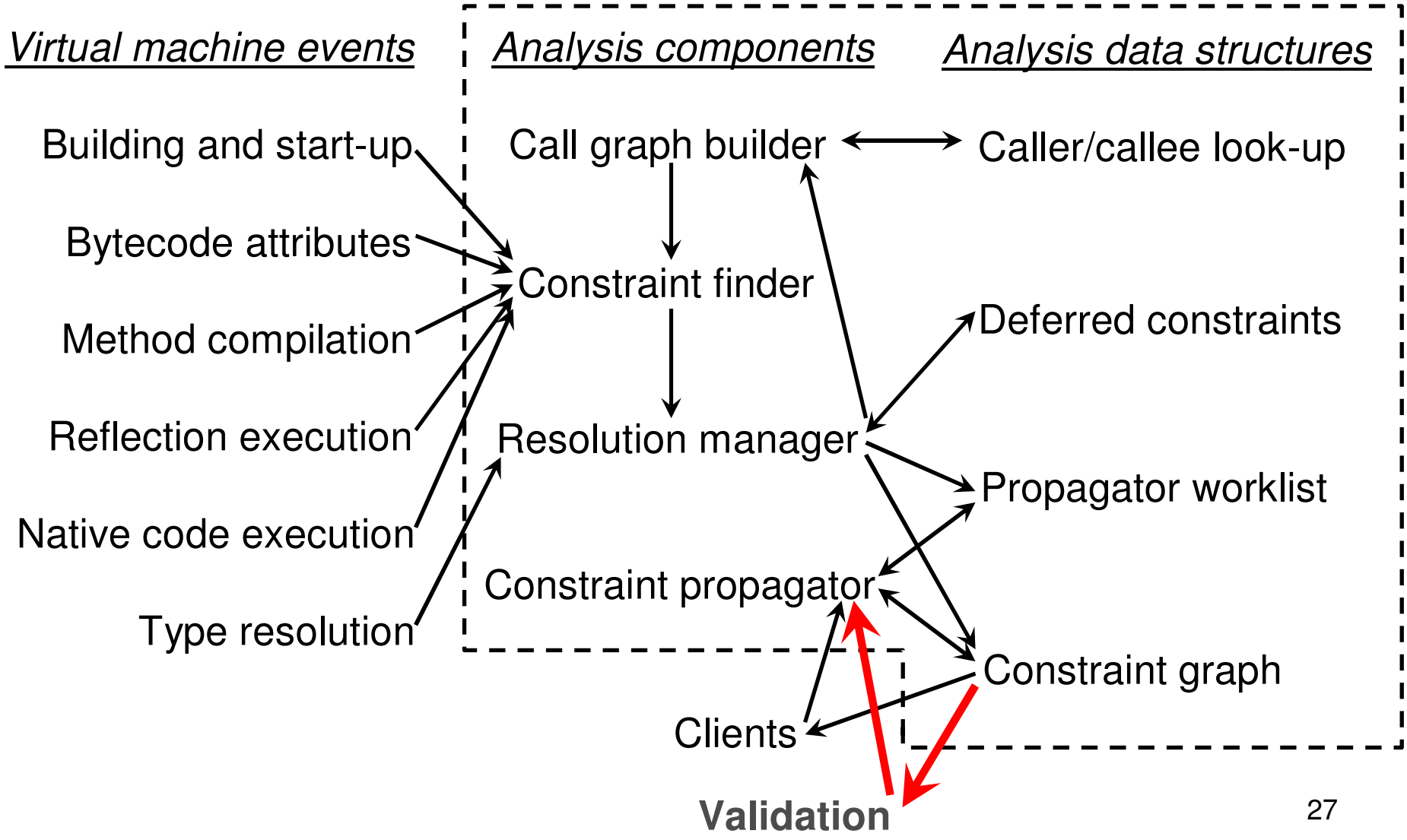
Application runtime →

Analysis cost to amortize

	10%	5%	2.5%
5m	50m	1h40m	3h20m
15m	2h30m	5h	10h
1h	10h	20h	1d16h
5h	1d16h	4d04h	8d08h

➔ ***Long-running applications can amortize not-too-eager analysis cost***

Validation


Virtual machine events
Analysis components
Analysis data structures
Building and start-up
Bytecode attributes
Method compilation
Reflection execution
Native code execution
Type resolution
Call graph builder
Constraint finder
Resolution manager
Constraint propagator
Clients
Caller/callee look-up
Deferred constraints
Propagator worklist
Constraint graph
Validation

Validation

- Piggy-back validation on garbage collection
- For each pointer, check consistency with analysis results
- Incorrect analysis would lead to tricky bugs in clients

Related work

- Andersen's analysis for "static Java"
  - [RountevMilanovaRyder'01]
  - [LiangPenningsHarrold'01]
  - [WhaleyLam'02]
  - [LhotakHendren'03]
- Weaker analyses with dynamic class loading
  - DOIT – [PechtchanskiSarkar'01]
  - XTA – [QianHendren'04]
  - Ruf's escape analysis – [BogdaSingh'01, King'03]
- Demand-driven / incremental analysis

Conclusions

- 1st non-trivial pointer analysis for all of Java
- Identified and solved the challenges:
  1. ***Online call graph building***
  2. ***Focused re-propagation***
  3. ***Managing unresolved types***
  4. ***Reflection and native code***
- Evaluated efficiency
- Validated correctness