Testing of Java Web Services for Robustness

Chen Fu, Ana Milanova,
David Wonnacott, Barbara Ryder

Availability of Internet Services

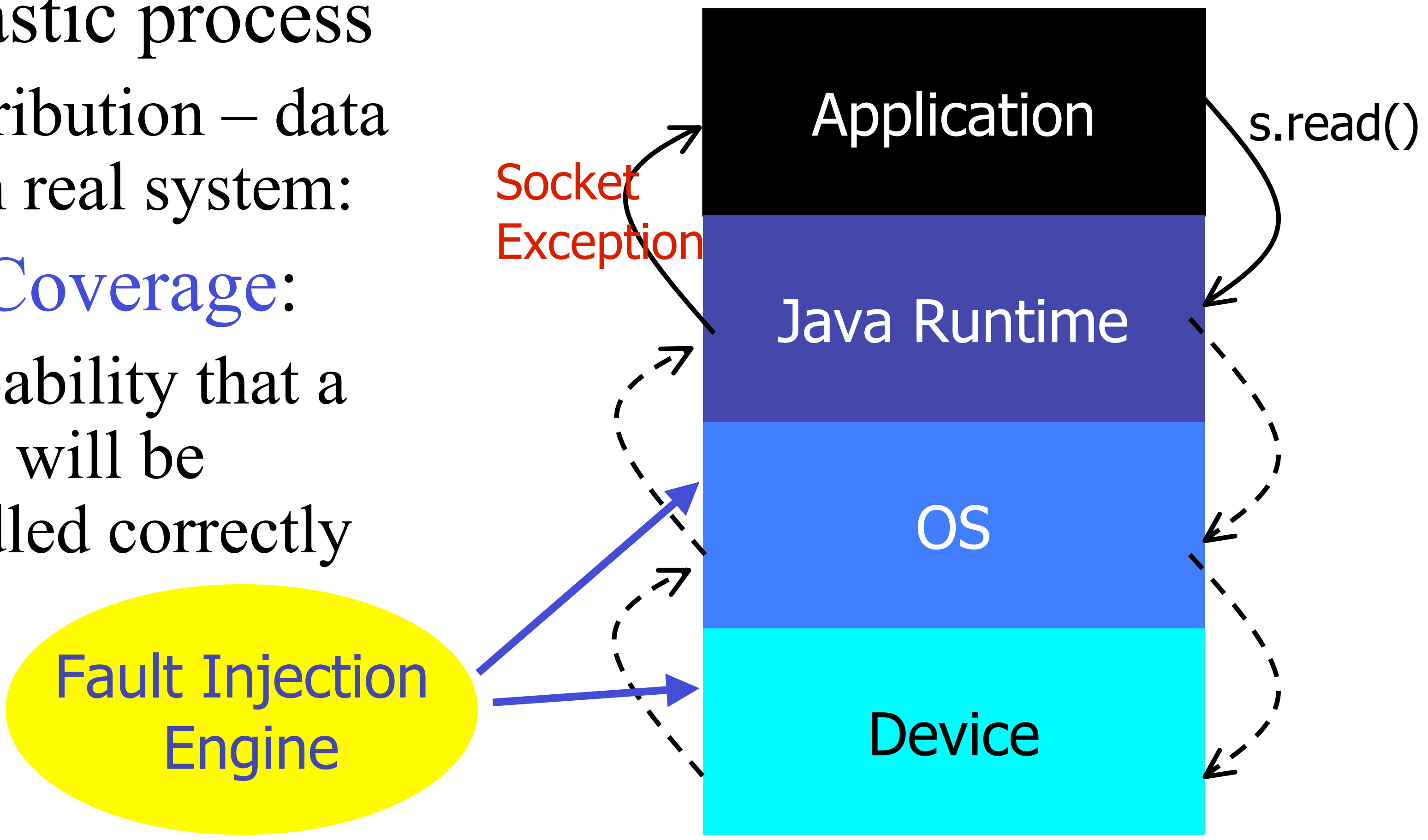
- Internet Service: New Kid in 24x7 domain
  - Public Telephone System: 99.999%
  - Internet Services: 99% ~ 99.9%
- Why?
  - Hardware:
    - Heterogeneous Cluster-based, complex system
  - Software
    - Short lifecycle caused by market pressure.
    - Components from various vendors.
  - Faults are unavoidable (Disk/Network/OS)

Fault Injection

- Motivation:
  - Redundant components are used to mask individual faults.
    - But would the software be able to take advantage of that?
  - Testing program reaction to hardware/software problems
    - Disk Crash, Network congestion, OS resources depletion, OS bug …
  - Waiting for real fault to see the reaction of the system?
    - Actual Problems happen in rare basis
- Solution
  - Special software components to simulate "faulty conditions".

Fault Injection – Current Approach

- Stochastic process
  - Distribution – data from real system:
- Fault Coverage:
  - Probability that a fault will be handled correctly

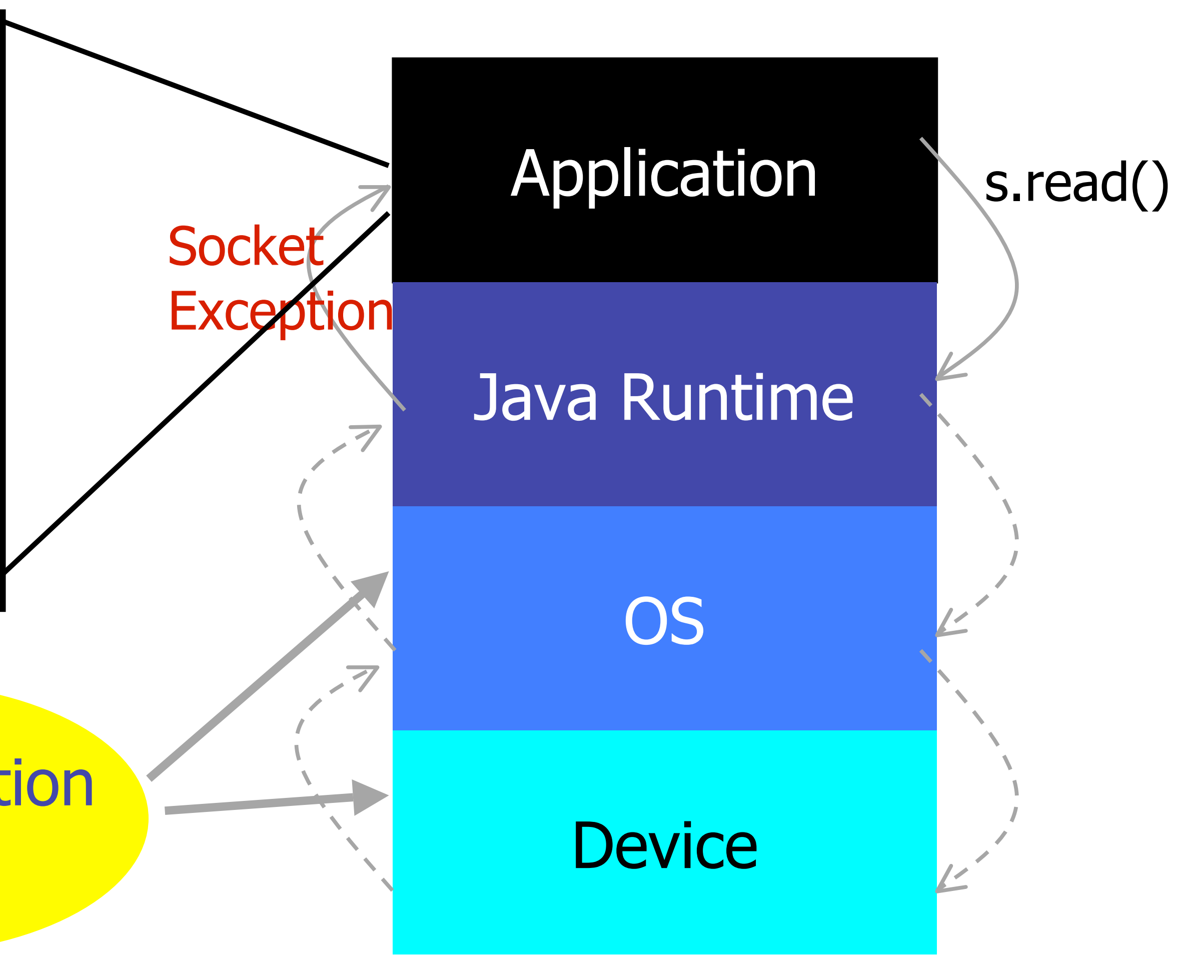


Fault Injection – White box test?

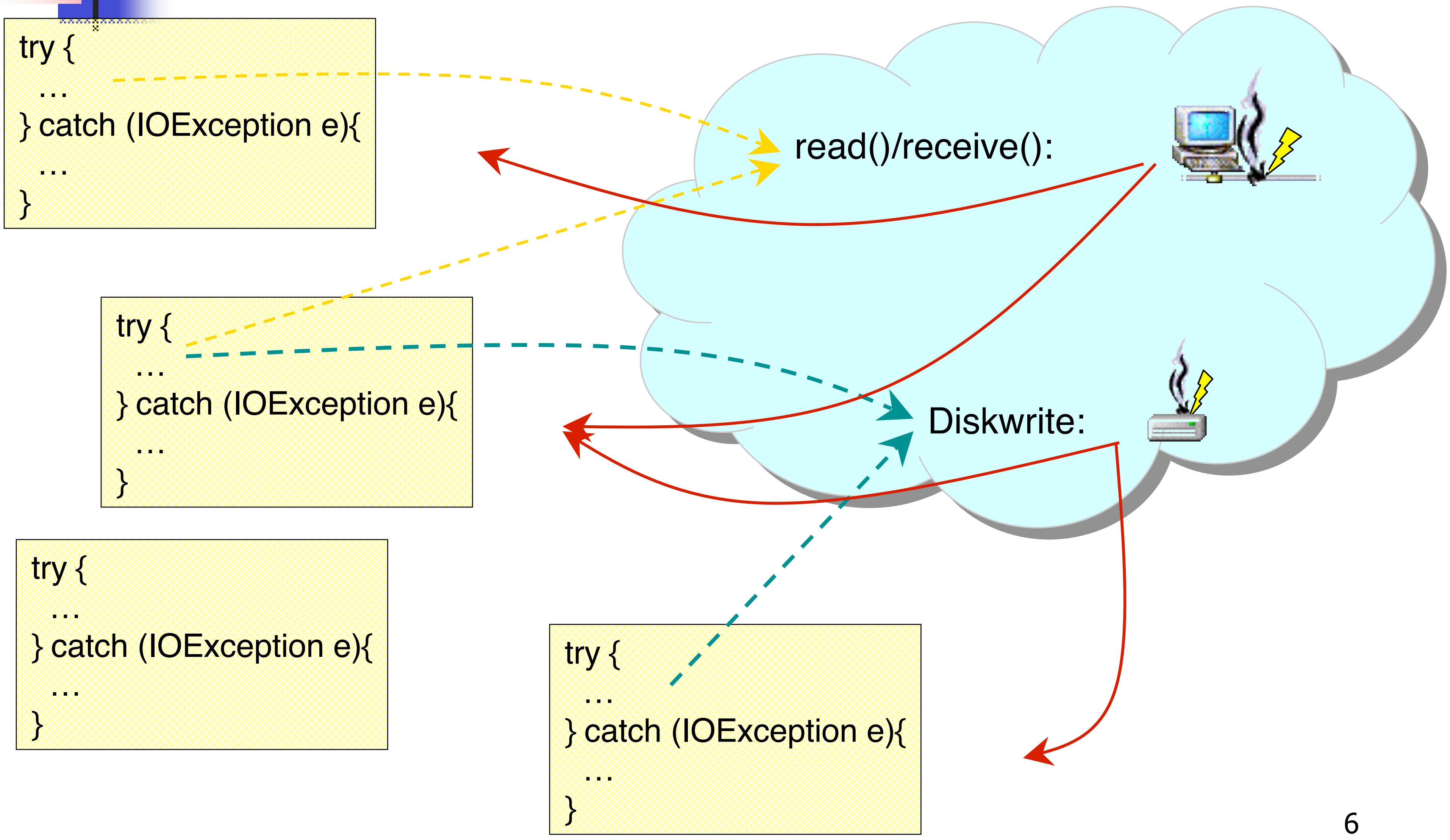
```
try{
  …

  process(…)
  …

} catch (IOException e){
  // recovery code
}
```



Exception-Catch Links



```
try {
  …
} catch (IOException e){
  …
}
```

```
try {
  …
} catch (IOException e){
  …
}
```

```
try {
  …
} catch (IOException e){
  …
}
```

```
try {
  …
} catch (IOException e){
  …
}
```

Coverage Metric

- Exception Def-Catch Coverage is: $\frac{|E|}{|F|}$

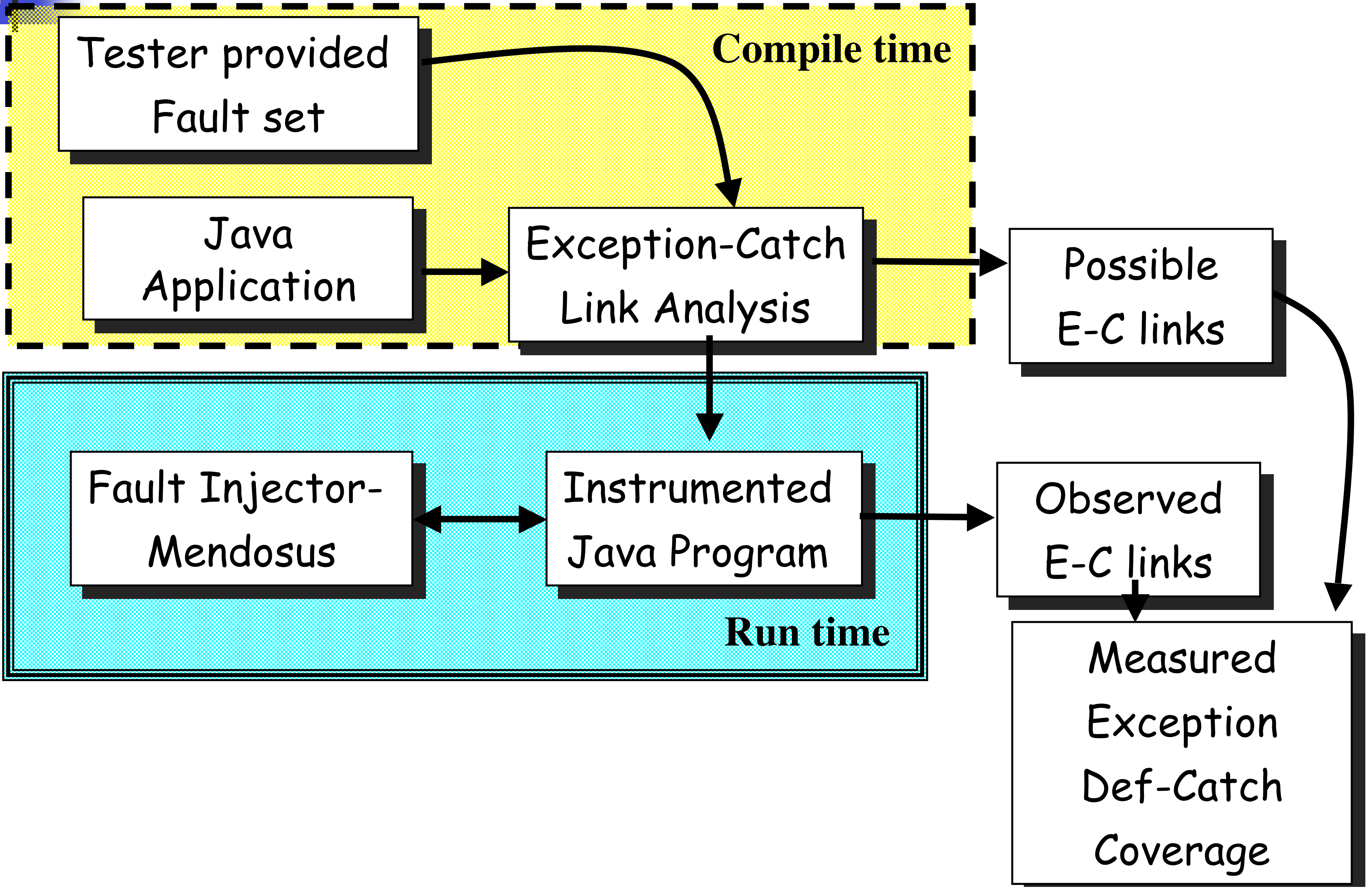
Static

- ***F*** – Set of possible *e-c links* (starting from a set of *fault-sensitive* operations)

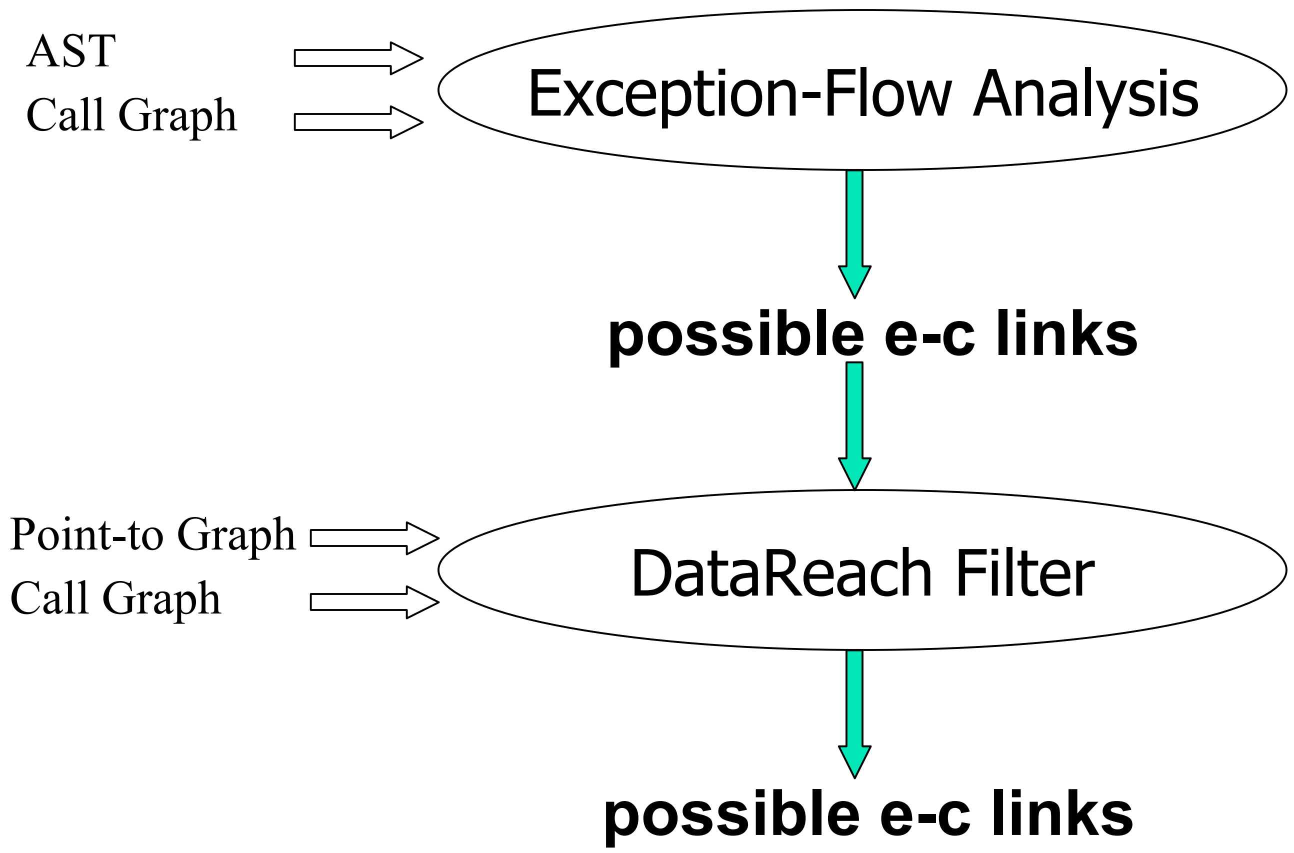
Dynamic

- ***E*** – Set of *e-c links* that are actually experienced in a set of test runs $(E \subseteq F)$

Framework


Compile time
Tester provided Fault set
Java Application
Exception-Catch Link Analysis
Possible E-C links
Fault Injector-Mendosus
Instrumented Java Program
Run time
Observed E-C links
Measured Exception Def-Catch Coverage

Analysis: Finding *e-c links*



Exception-flow Analysis

Finding e-c links

```
void foo() throws Exception{
...
  try{
    bar();
  }catch (IOException ioe){…}
}
```

Set of **throws** can be handled here?

Set of **throws** that can reach **bar()** without being handled?

ReachingThrown

Exception-flow Analysis

```
void foo() throws Exception{
...
  try{
    bar();
  }catch (IOException ioe){…}
}

void bar() throws Exception{
.  .  .
  throw new SocketException();
.  .  .
  throw new OtherException();
.  .  .
}
```

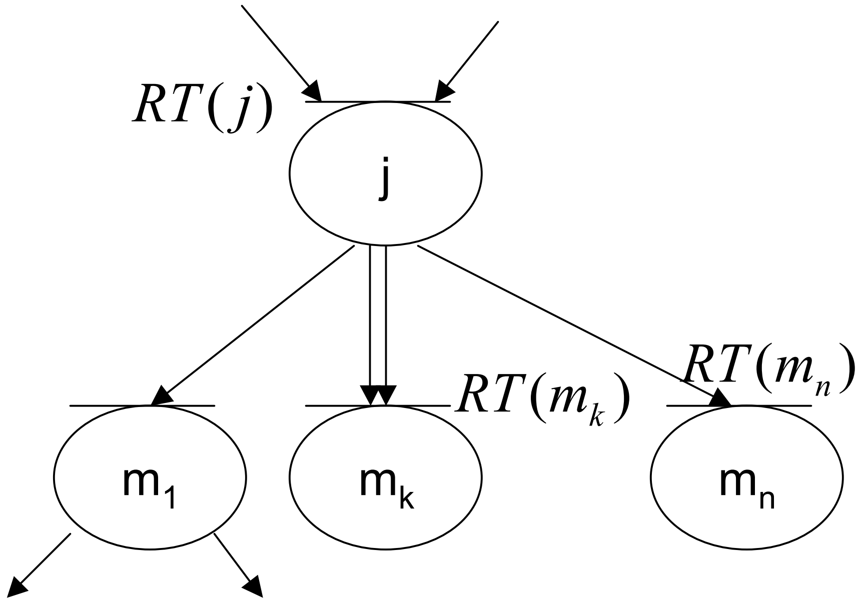
ReachingThrow(`foo`)
  `OtherException` thrown in **`bar`**

ReachingThrow(`bar`)
  `SocketException` thrown in **`bar`**
  `OtherException` thrown in **`bar`**

$$RT(j) = \bigcup_{t \in T} (gen(t) - kill(trynest(t))) \cup \bigcup_{cs \in CS} \bigcup_{m \in \text{target}(cs)} (RT(m) - kill(trynest(cs)))$$

Exception-flow Analysis

- Dataflow Problem defined on call graph (backward)
- Varies call graph algorithm can be used:
  - CHA, RTA, Points-To (context insensitive, context sensitive)



SocketException thrown in **bar**

Call Chain

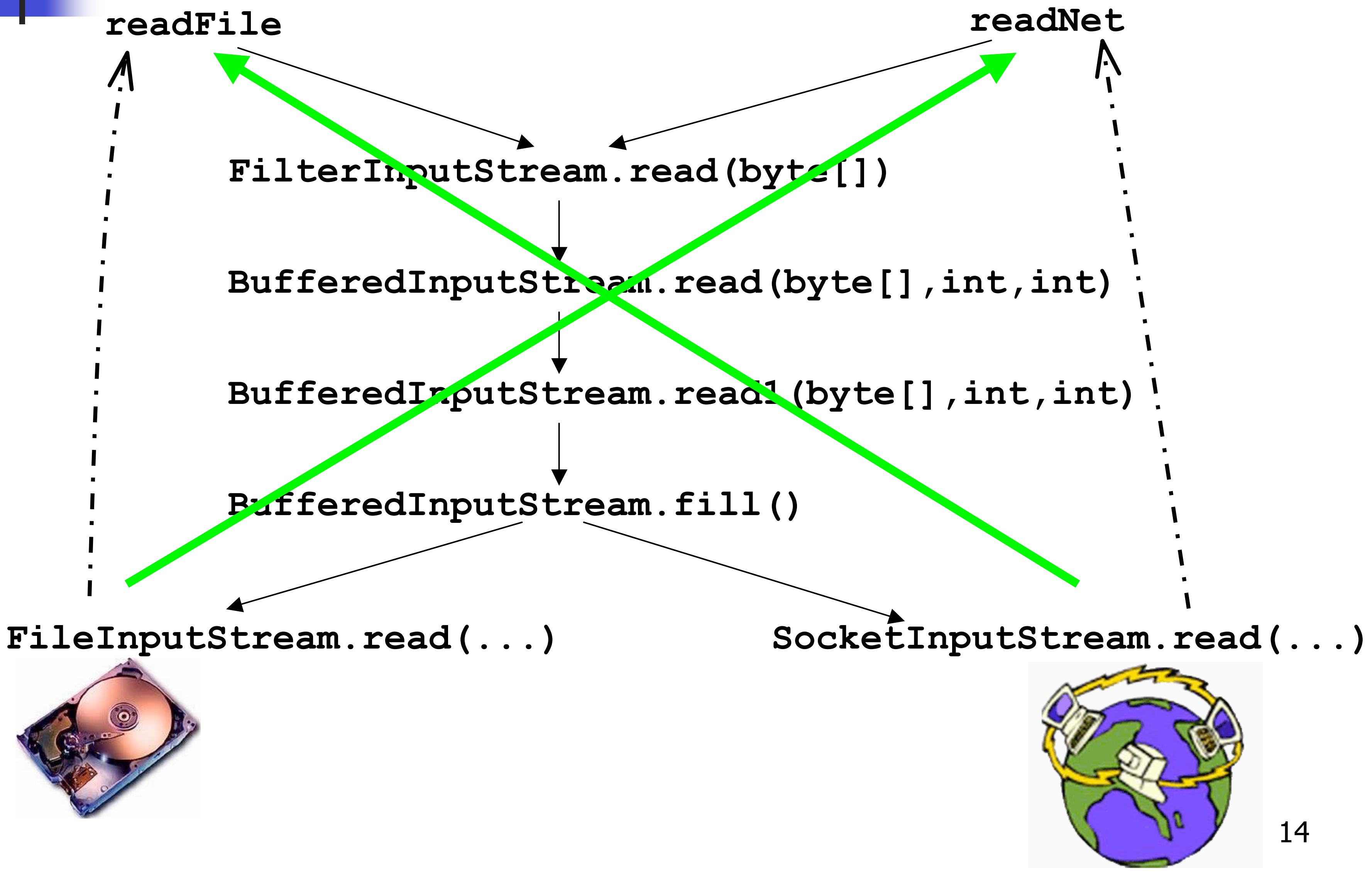
catch (IOException ioe) in **foo**

Data-Reach ____ Motivation

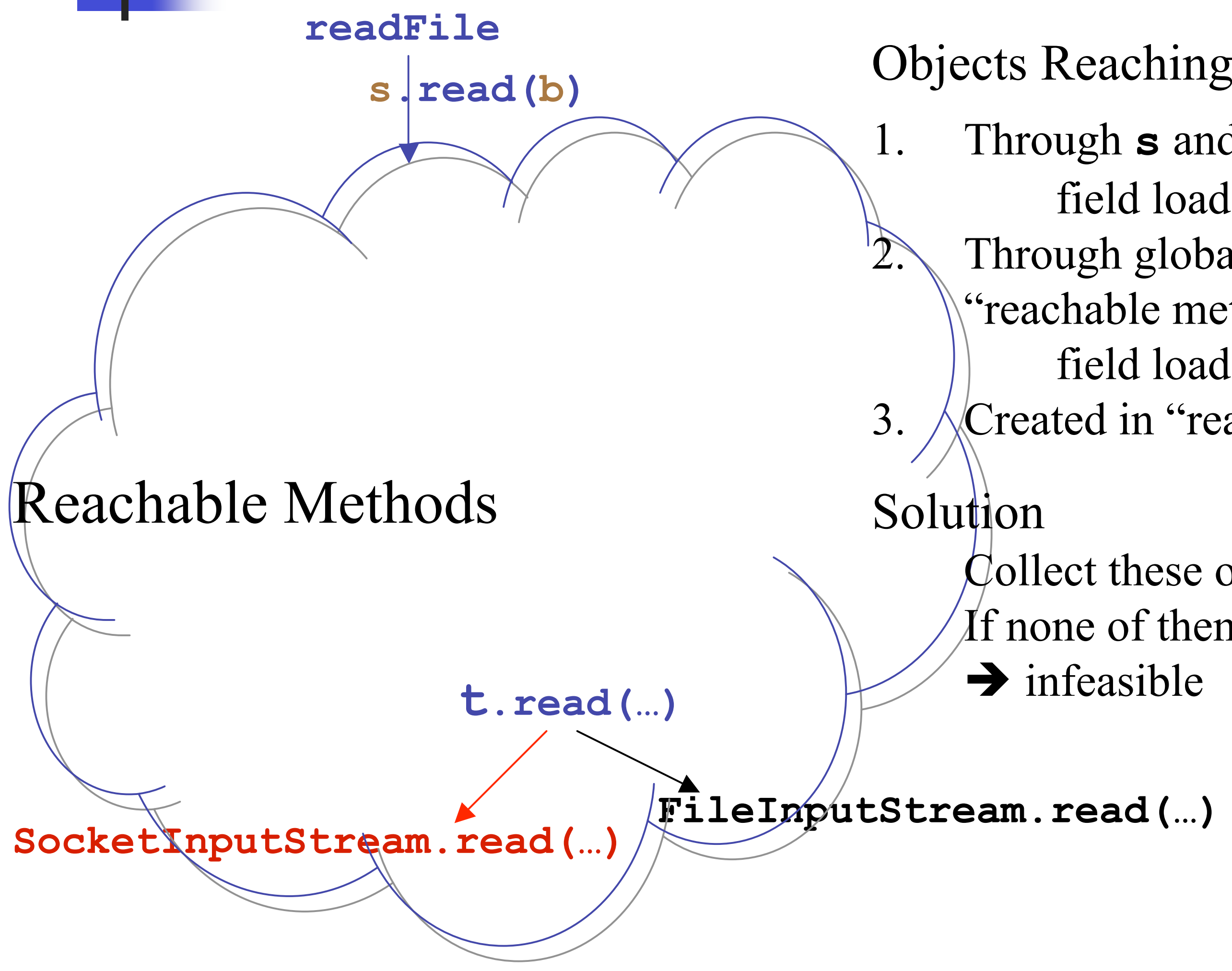
```
void readFile(String s){
 byte[] buffer = new byte[256];
 try{
  InputStream f =new FileInputStream(s);
  InputStream source=new
BufferedInputStream(f);
  for (...)
   c = source.read(buffer);
 }catch (IOException e){ ...}
}
```

```
void readNet(Socket s){
 byte[] buffer = new byte[256];
 try{
  InputStream n =s.getInputStream();
  InputStream source=new
BufferedInputStream(n);
  for (...)
   c = source.read(buffer);
 }catch (IOException e){ ...}
}
```

Data-Reach ___ Motivation


readFile
readNet
FilterInputStream.read(byte[])
BufferedInputStream.read(byte[],int,int)
BufferedInputStream.read1(byte[],int,int)
BufferedInputStream.fill()
FileInputStream.read(...)
SocketInputStream.read(...)

Feasibility of a call chain



Objects Reaching **t**:

1. Through **s** and **b**
   field loads in "reachable methods"
2. Through global variables accessed in "reachable methods"
   field loads in "reachable methods"
3. Created in "reachable methods"

Solution

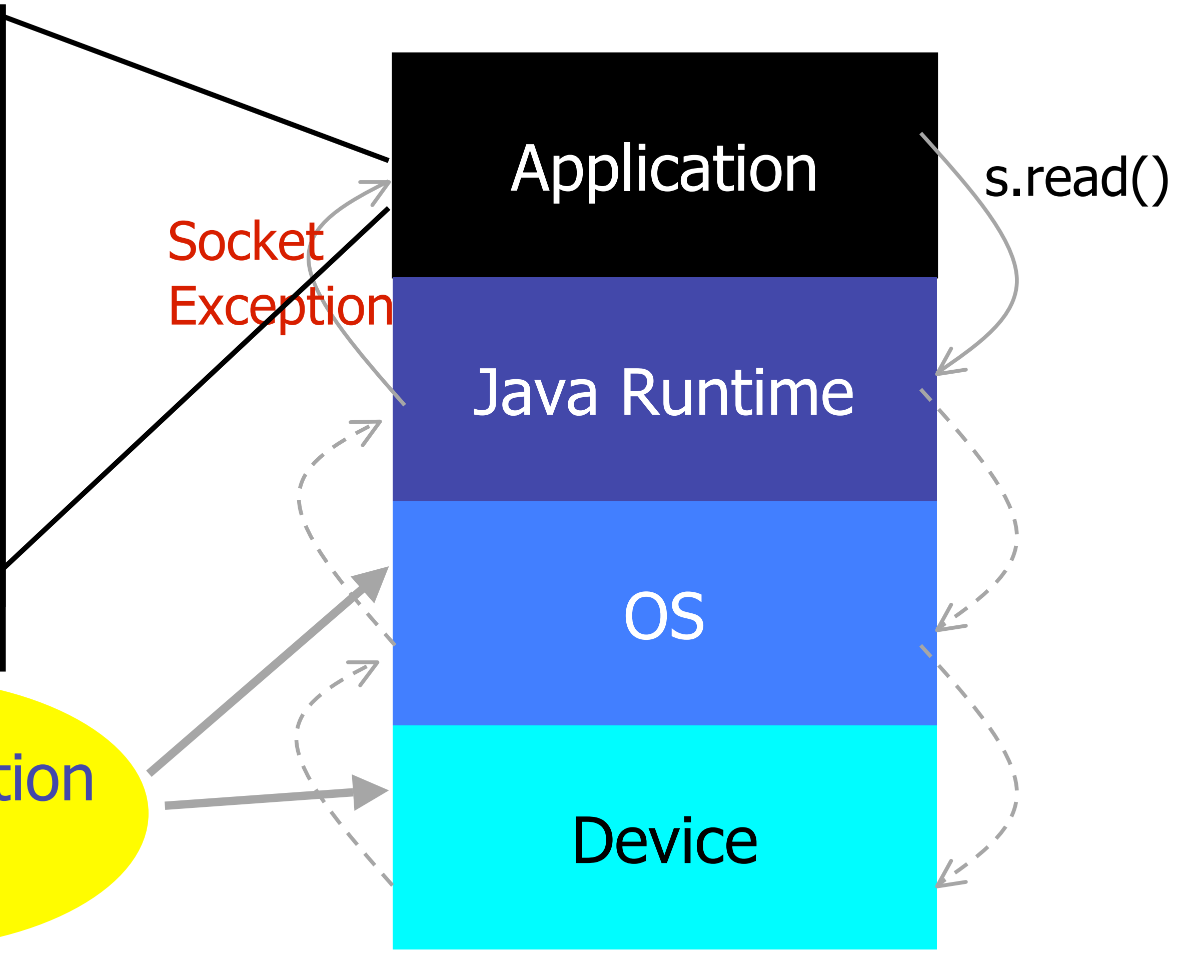
Collect these objects using **Points-to Graph**

If none of them has appropriate type

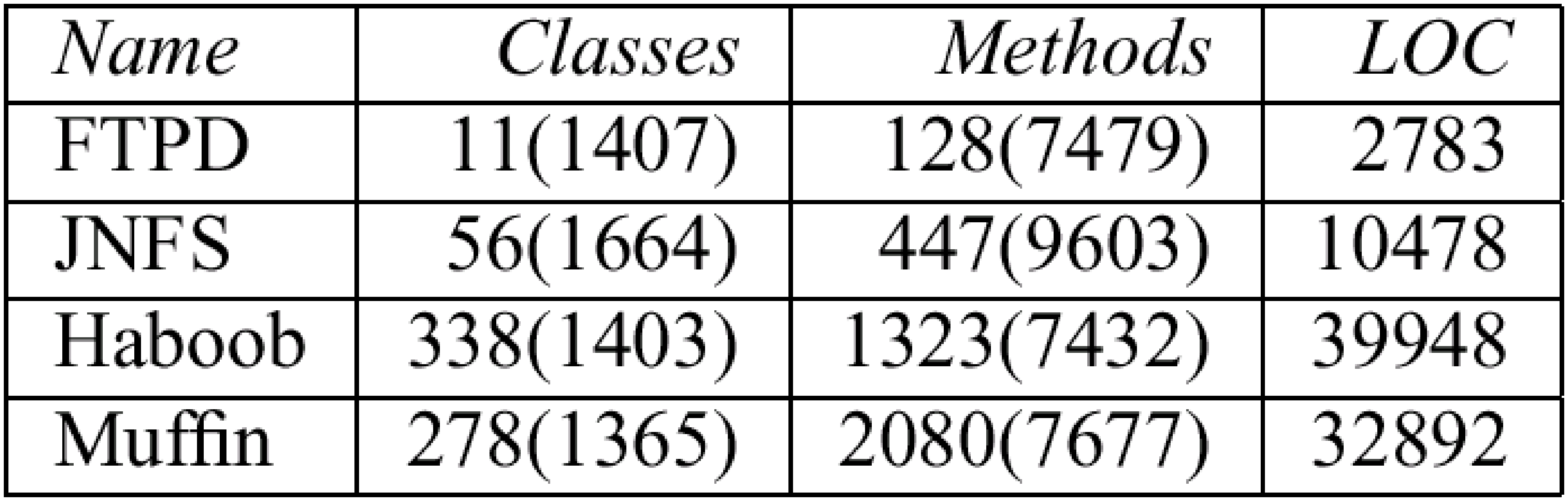
➔ infeasible

Instrumentation

```
try{
  …
  inject_fault();
  process(…)
  …
  cancel_fault();
} catch (IOException e){
  record_current_fault();
  // recovery code
}
```



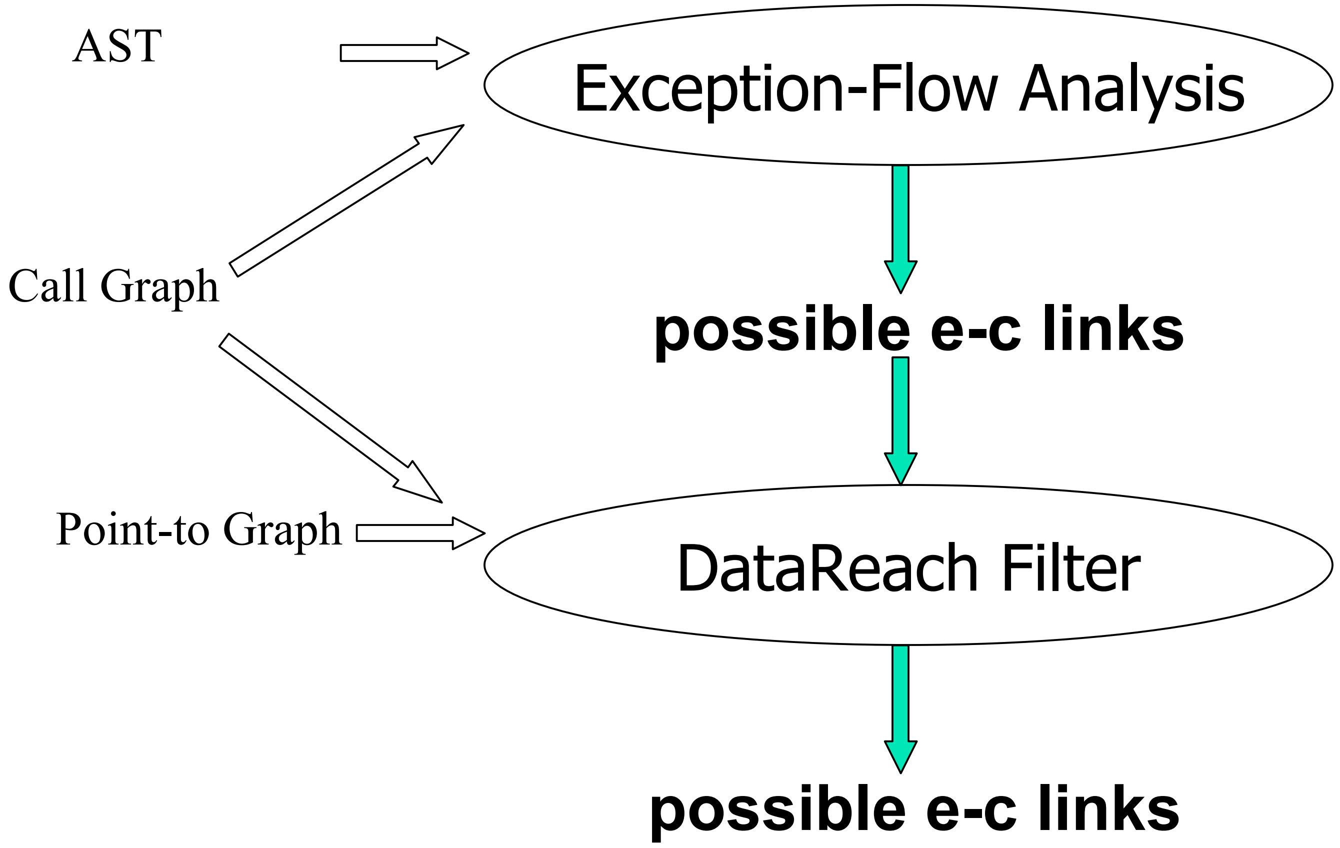
Benchmarks



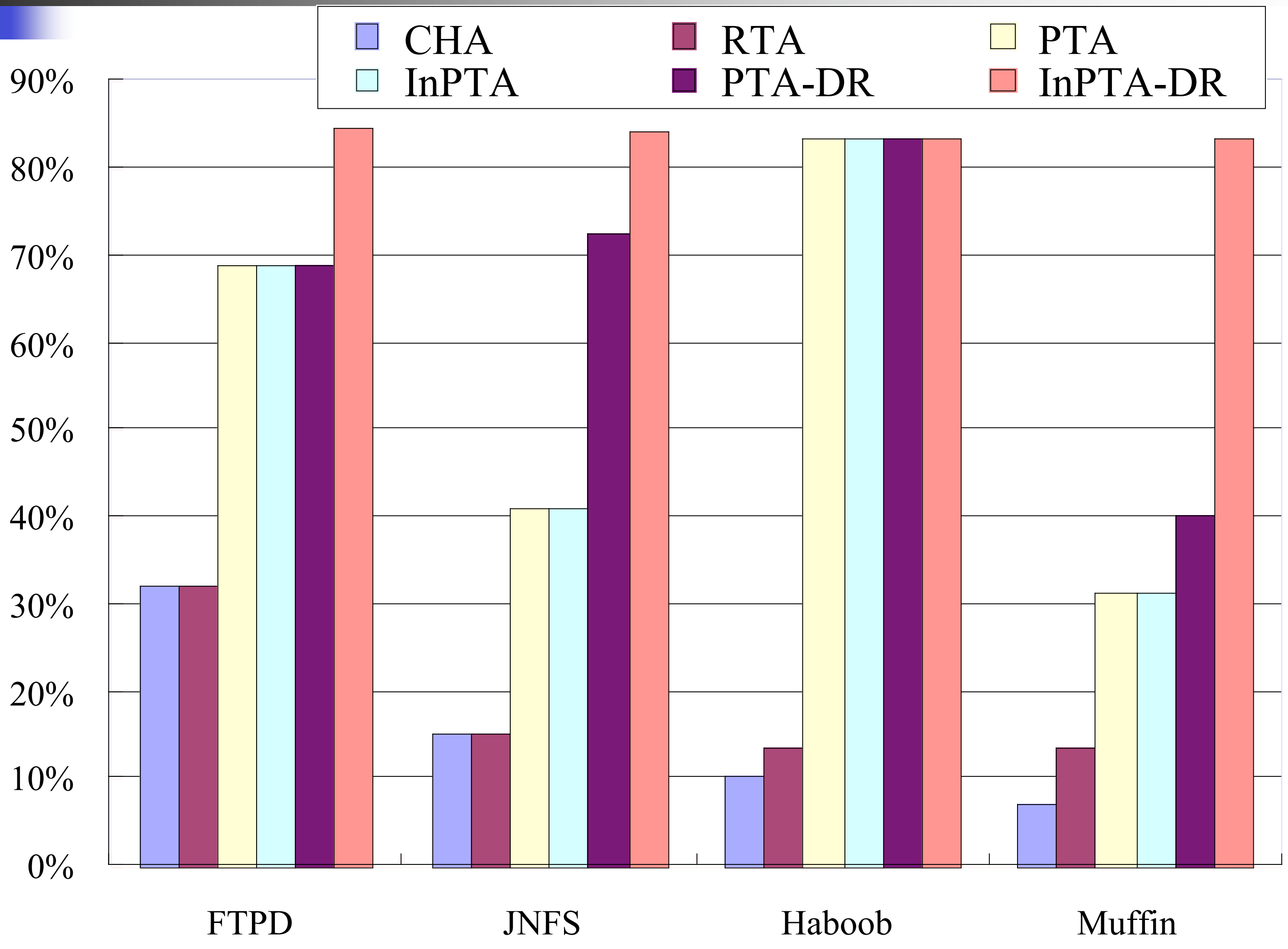
Name	*Classes*	*Methods*	*LOC*
FTPD	11(1407)	128(7479)	2783
JNFS	56(1664)	447(9603)	10478
Haboob	338(1403)	1323(7432)	39948
Muffin	278(1365)	2080(7677)	32892

Configurations

- 🟦 CHA
- 🟪 RTA
- 🟨 PTA
- 🟦 InPTA
- 🟪 PTA-DR
- 🟥 InPTA-DR

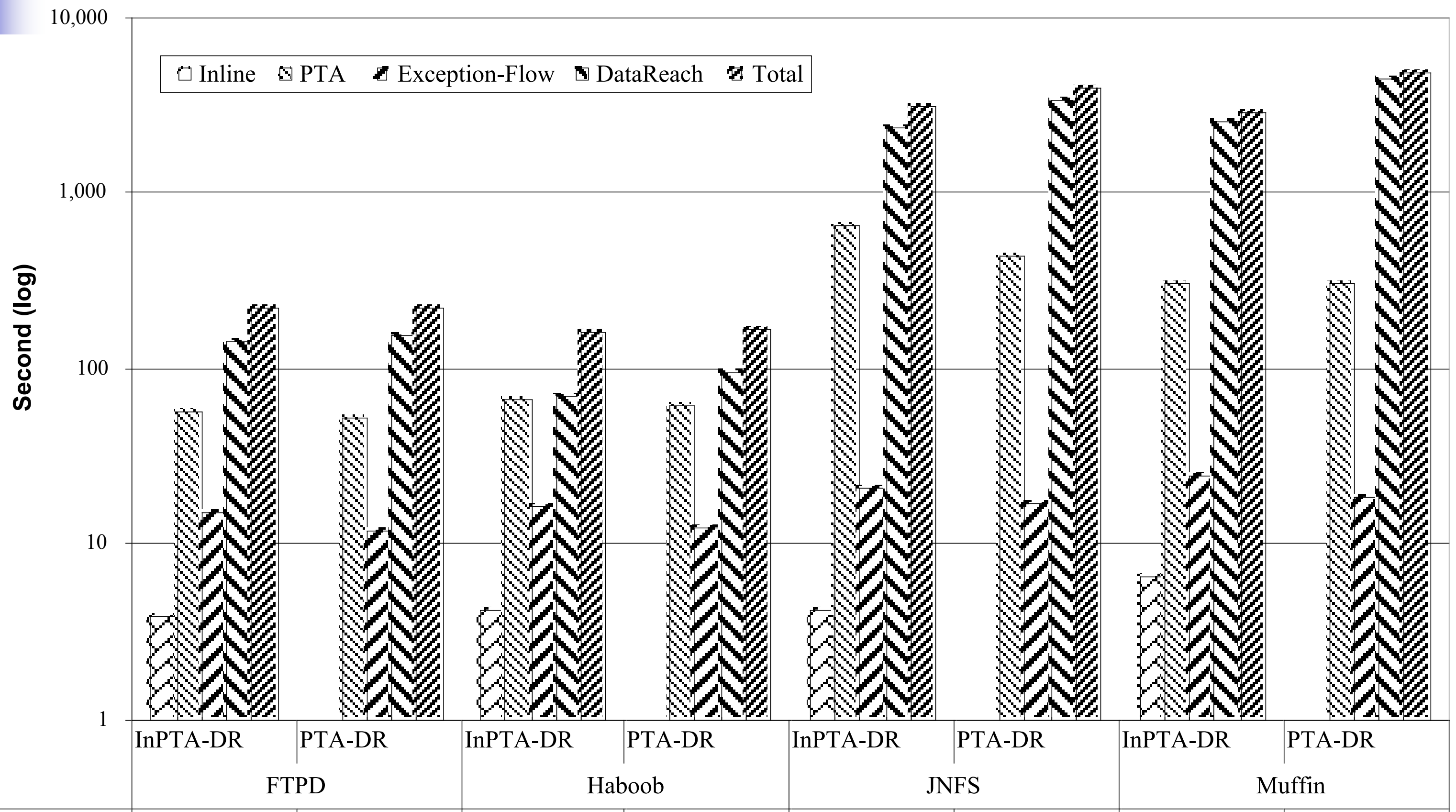


Coverage



CHA
RTA
PTA
InPTA
PTA-DR
InPTA-DR
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
FTPD
JNFS
Haboob
Muffin

Time Cost

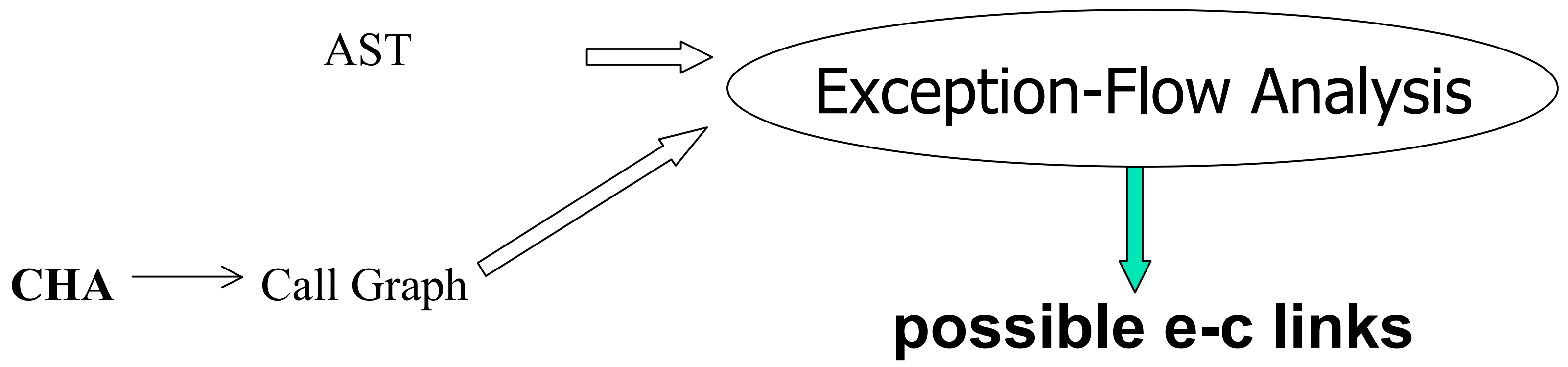


	FTPD		Haboob		JNFS		Muffin	
	InPTA-DR	PTA-DR	InPTA-DR	PTA-DR	InPTA-DR	PTA-DR	InPTA-DR	PTA-DR
Inline	3.9		4.2		4.2		6.7	
PTA	56.7	53.7	67.7	62.2	659.4	453.1	304.5	307.2
Exception-Flow	15.4	12.1	16.8	12.5	20.9	17.5	25.1	19.2
DataReach	144.6	155.8	71.7	97.2	2,419.2	3,490.4	2,618.0	4,581.3
Total	220.6	221.6	160.4	171.9	3,103.7	3,961.0	2,954.3	4,907.7

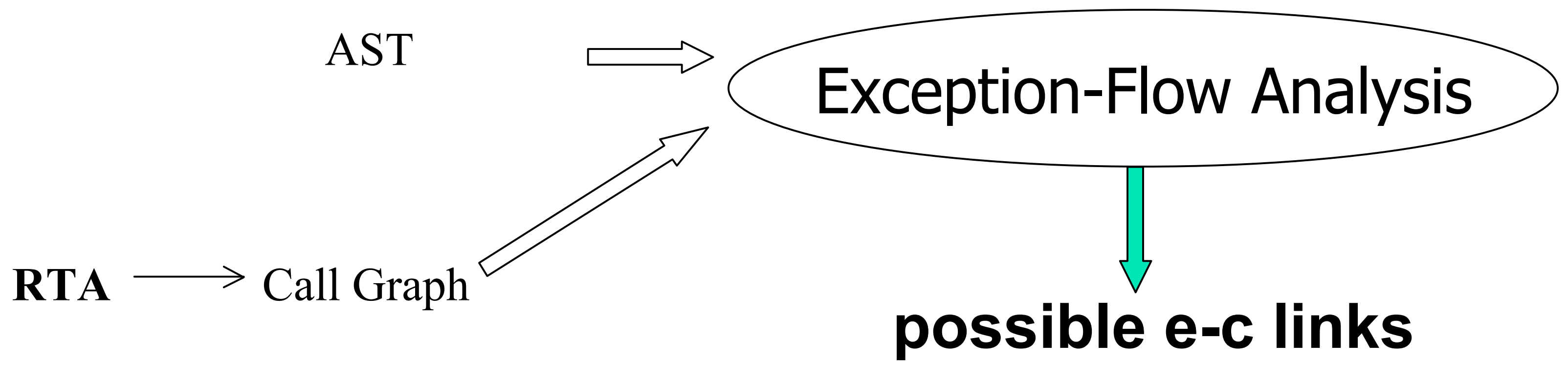
Benchmarks

Thanks!

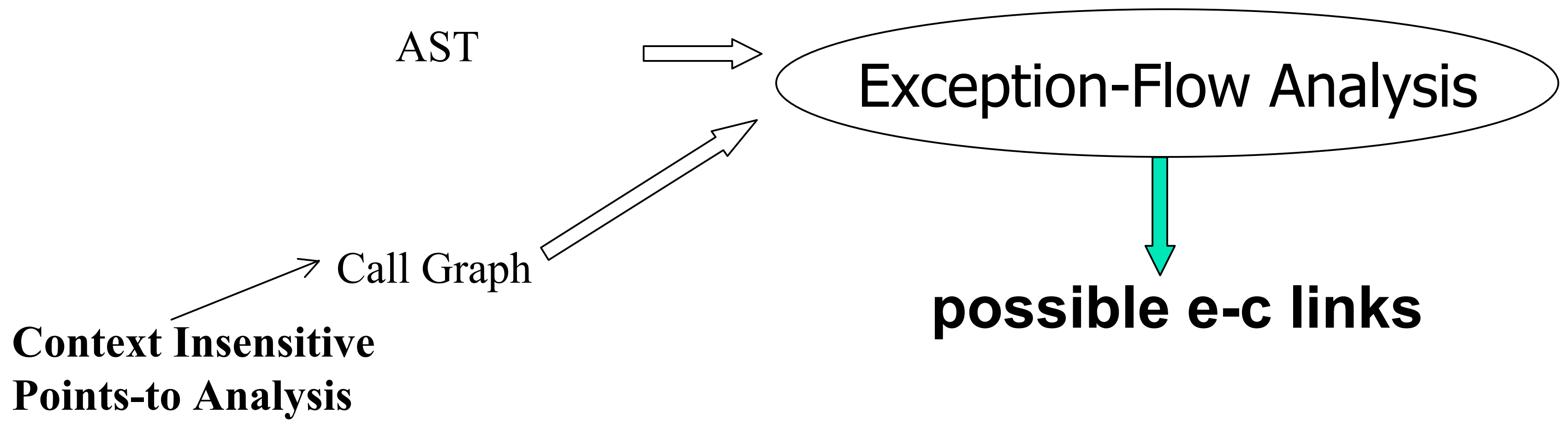
Configurations -- CHA



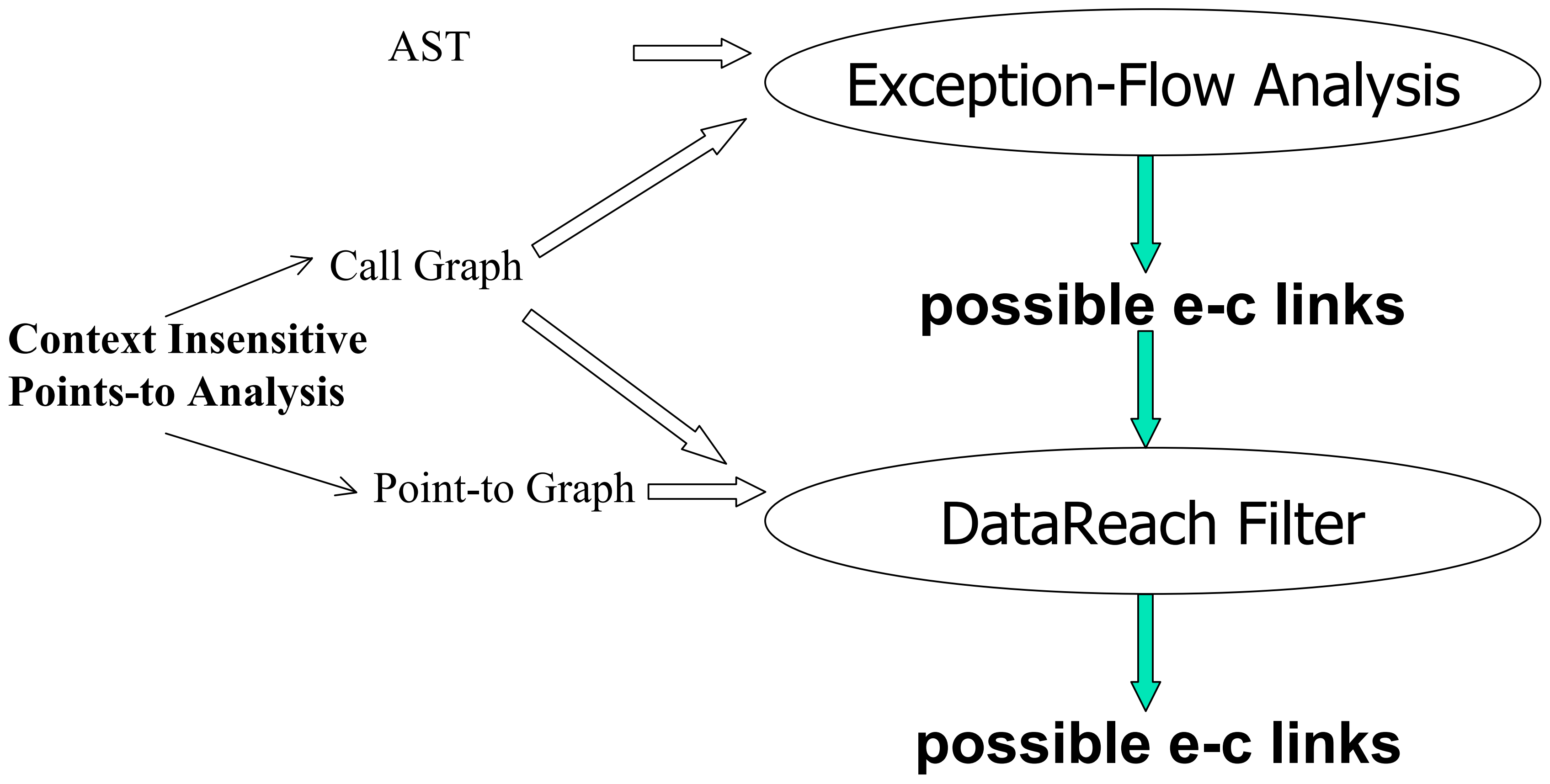
Configurations -- RTA



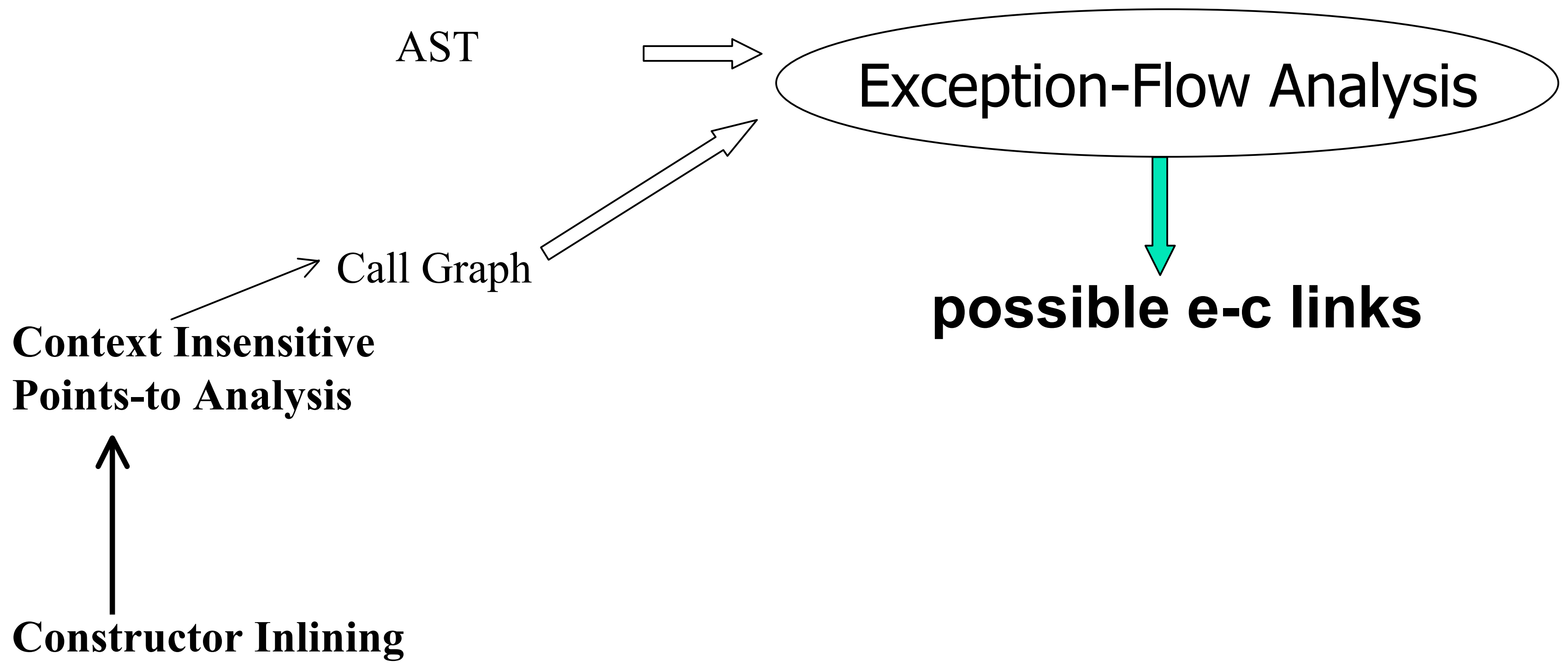
Configurations -- PTA



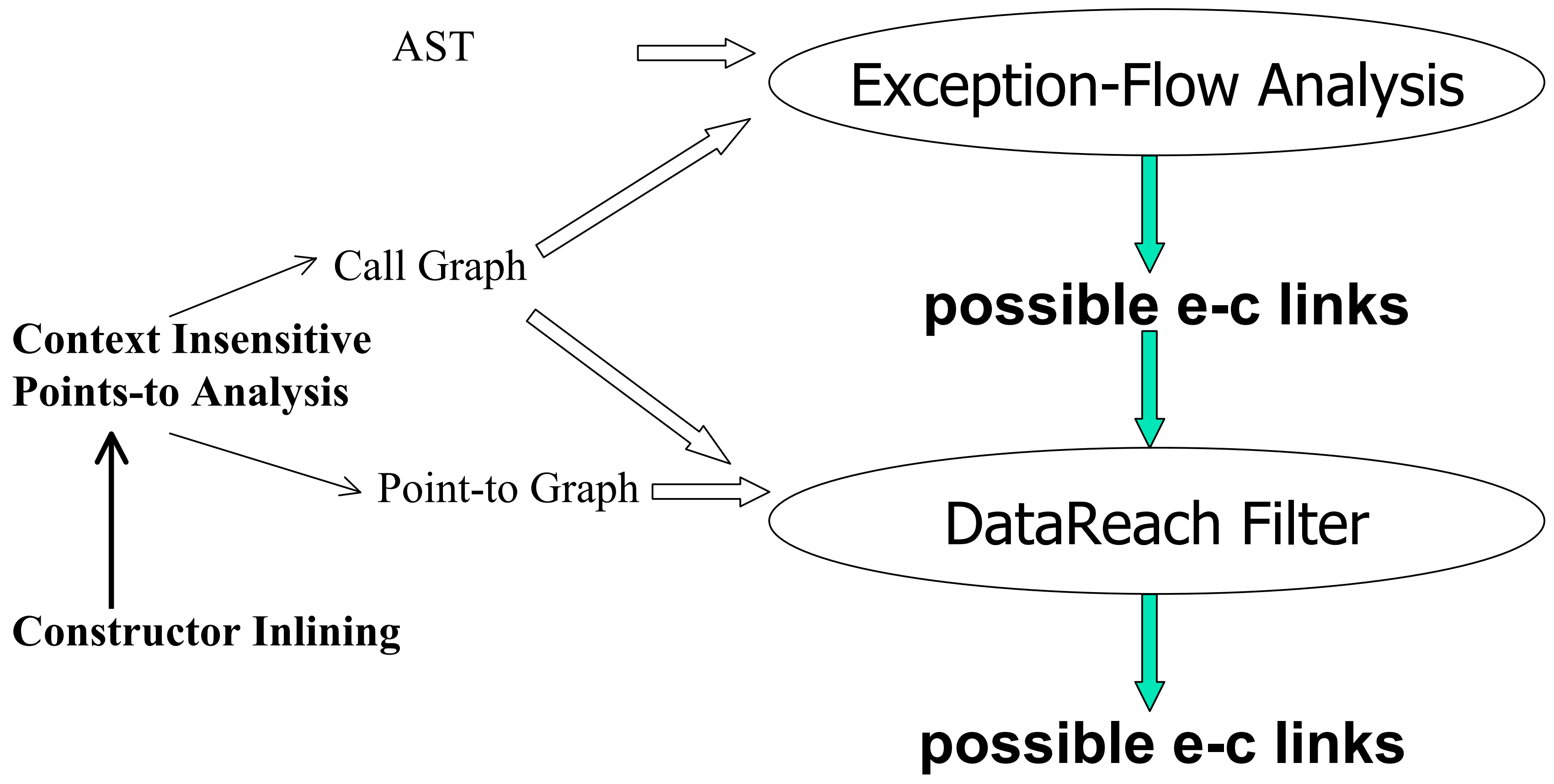
Configurations -- PTA-DR



Configurations -- InPTA



Configurations -- InPTA-DR


AST
Exception-Flow Analysis
Call Graph
possible e-c links
Context Insensitive Points-to Analysis
Point-to Graph
DataReach Filter
Constructor Inlining
possible e-c links