Lightweight and Modular Resource Leak Verification

Martin Kellogg[a] , **Narges Shadab**[b] , Manu Sridharan[b], Michael D. Ernst[a]

[a]University of Washington

[b]University of California, Riverside

What's a Resource Leak?

```
try {
    Socket s = new Socket(address, port);
    ...
    s.close();
} catch (IOException e) {

}
```

What's a Resource Leak?

```
try {
    Socket s = new Socket(address, port);
    ...

} catch (IOException e) {

}
```

Missing call to close()

Problems Caused by Resource Leaks

- Resource starvation
- Slowdowns
- System crashes
- Denial-of-service attack
  - E.g. CVE-1999-1127, CVE-2001-0830, CVE-2002-1372

Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias

Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias
- Previous approaches:

Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias
- Previous approaches:

Heuristic bug-finding tools

Ignore aliasing

Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias
- Previous approaches:

Heuristic bug-finding tools	**Ownership type systems**
Ignore aliasing	Enforce uniqueness

Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias
- Previous approaches:

Heuristic bug-finding tools	Ownership type systems	**Whole-program static analysis**
Ignore aliasing	Enforce uniqueness	Track all aliases

Key Insight

- Resource leak detection is an **accumulation** problem

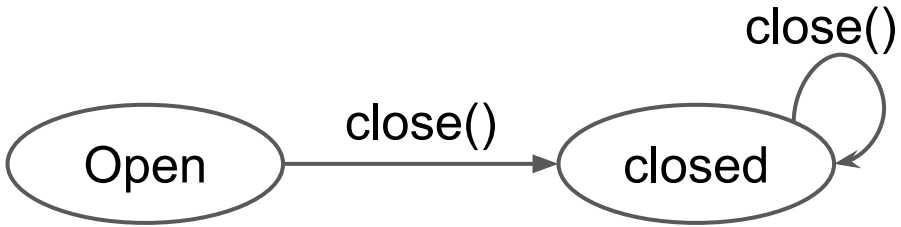
Key Insight

- Resource leak detection is an **accumulation** problem
  - FSM contains no loops

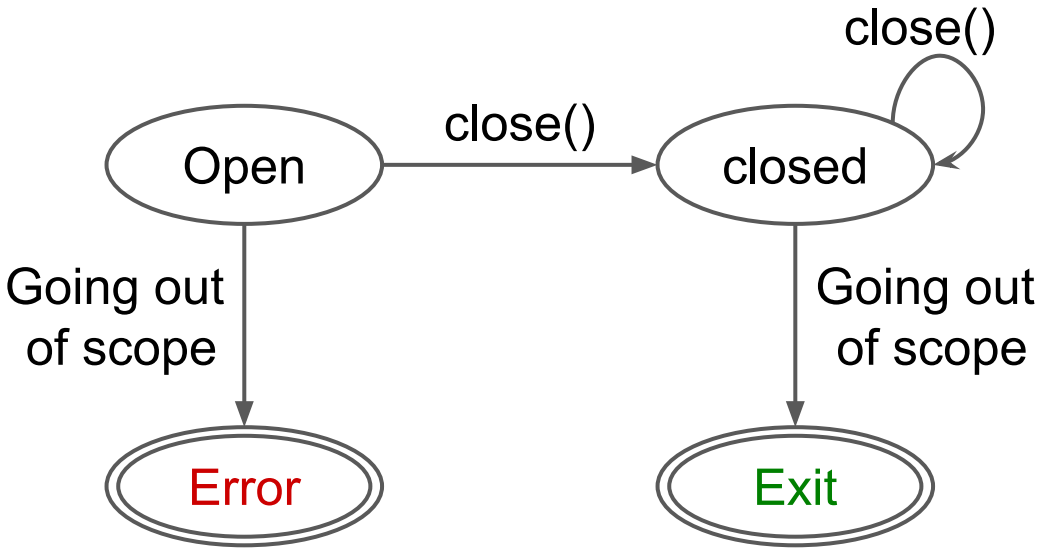
Key Insight

- Resource leak detection is an **accumulation** problem
  - FSM contains no loops
  - **Sound with no alias analysis**

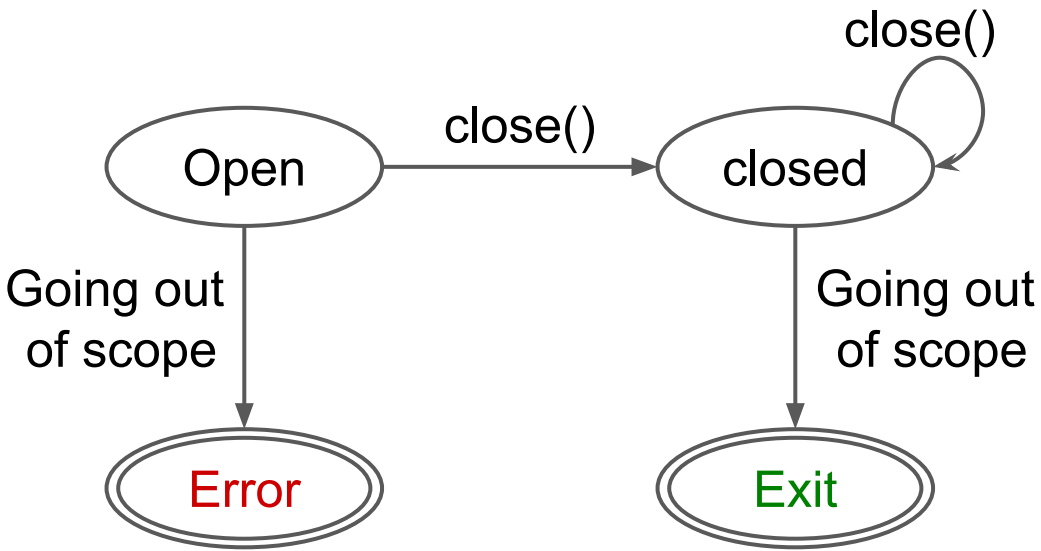
Resource Leaks as Accumulation



Resource Leaks as Accumulation

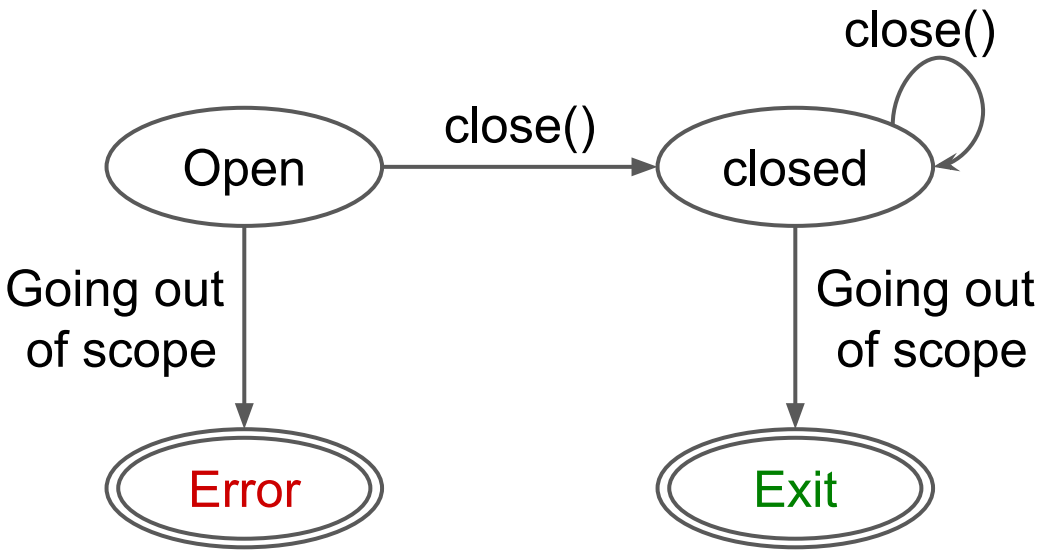


Resource Leaks as Accumulation



FSM contains no loops

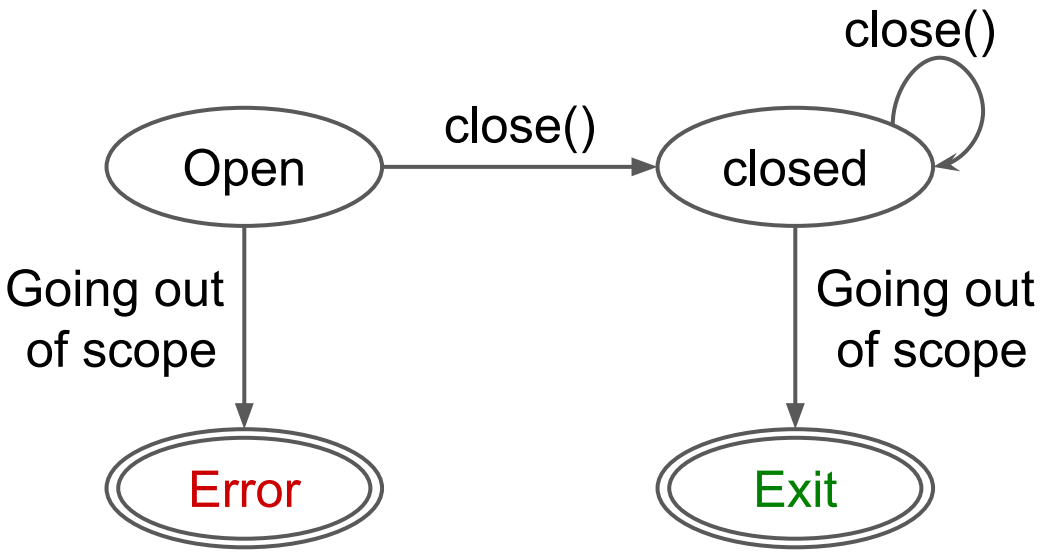
Resource Leaks as Accumulation



FSM contains no loops

Alias analysis not required for soundness

Resource Leaks as Accumulation



FSM contains no loops

Alias analysis not required for soundness

can be implemented modularly

Leak Detection Approach:

1. Compute what methods must be called
2. Compute what methods are called
3. Issue error if mismatch when going out of scope

Example

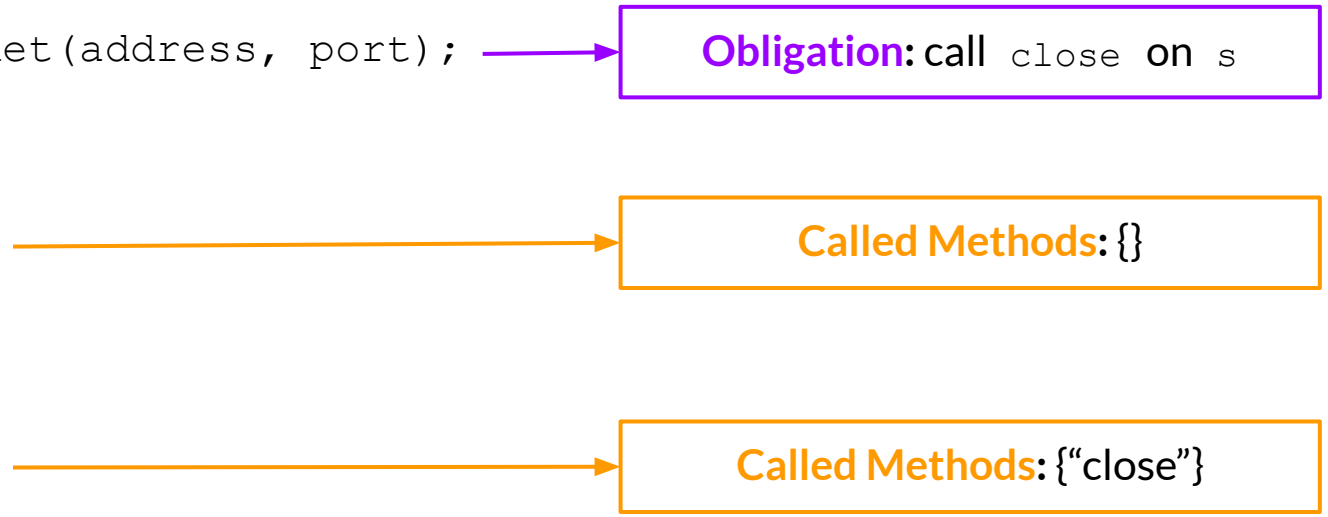
```
{
  s = new Socket(address, port);
  ...
  if (...) {
    s = ...;
  }
  s.close();
}
```

Example

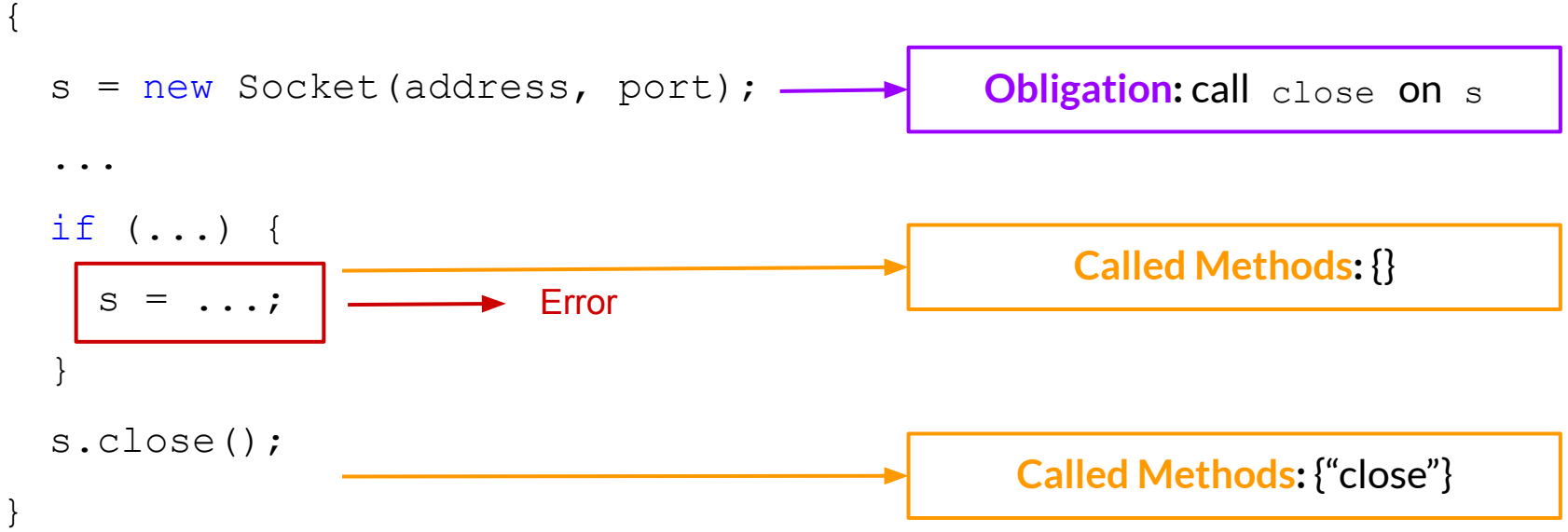
```
{
  s = new Socket(address, port);
  ...
  if (...) {
    s = ...;
  }
  s.close();
}
```

Example

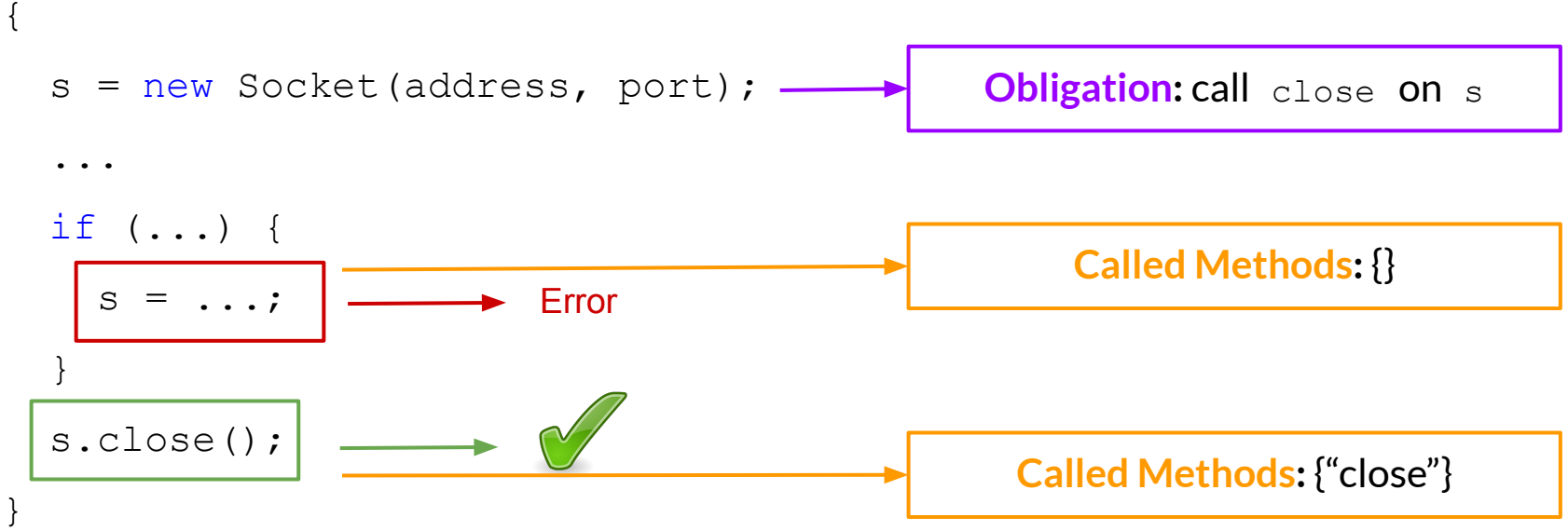
```
{
  s = new Socket(address, port);
  ...
  if (...) {
    s = ...;
  }
  s.close();
}
```



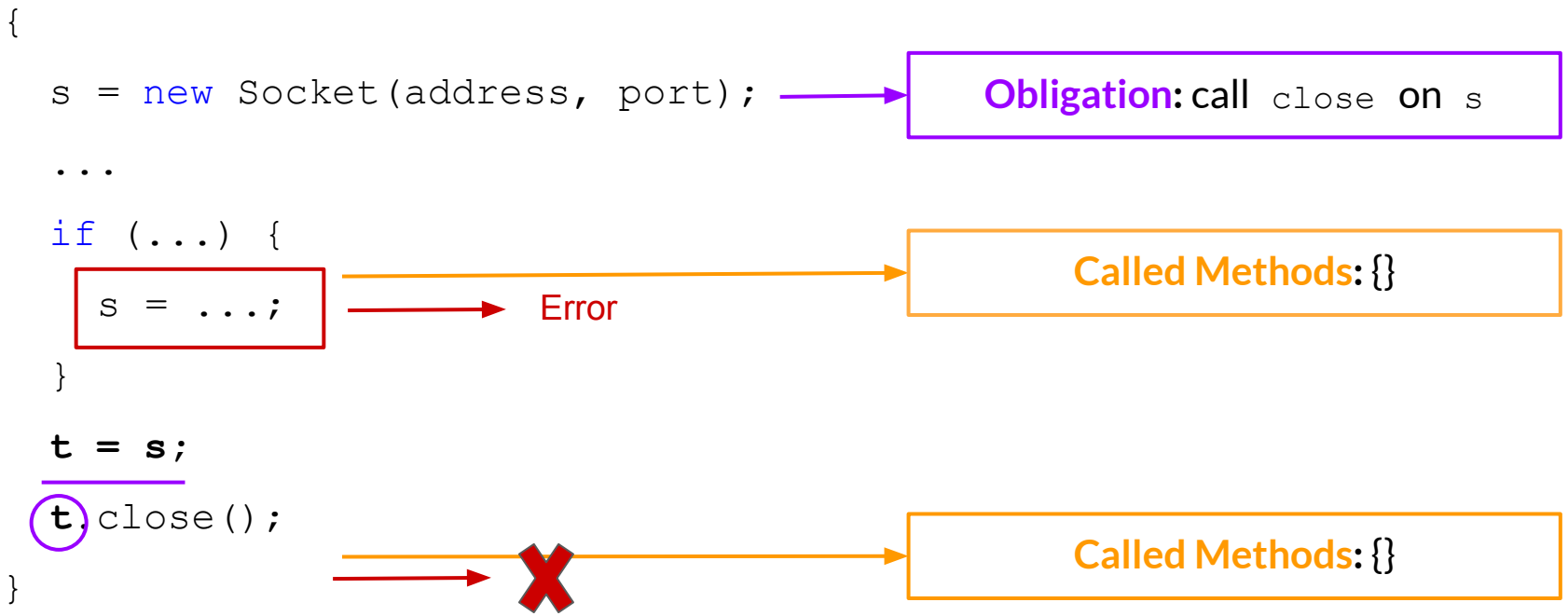
Example


{
s = new Socket(address, port);
...
if (...) {
s = ...;
}
s.close();
}
Obligation: call close on s
Called Methods: {}
Error
Called Methods: {“close”}

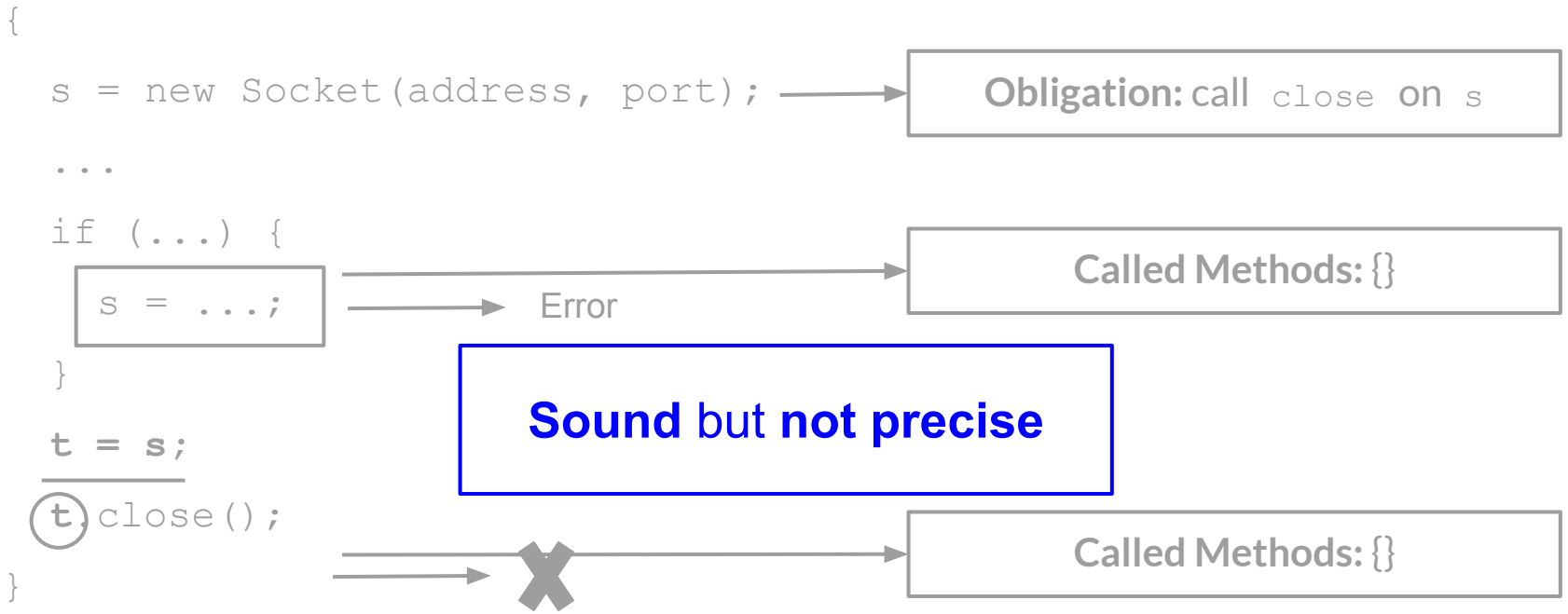
Example


{
s = new Socket(address, port);
...
if (...) {
s = ...;
}
s.close();
}
Obligation: call close on s
Called Methods: {}
Error
Called Methods: {“close”}

Example


{
s = new Socket(address, port);
...
if (...) {
s = ...;
}
t = s;
t.close();
}
Obligation: call close on s
Called Methods: {}
Error
Called Methods: {}

Example


{
s = new Socket(address, port);
...
if (...) {
s = ...;
}
t = s;
t.close();
}
Obligation: call close on s
Called Methods: {}
Error
Sound but not precise
Called Methods: {}

Precision via Local Alias Reasoning

- Local must-aliases
- Lightweight ownership
- Resource aliasing
- Obligation creation

Precision via Local Alias Reasoning

- Local must-aliases
- **Lightweight ownership**
- Resource aliasing
- Obligation creation

Lightweight Ownership

```
closeSocket(mySock);
```

Lightweight Ownership

Obligation: call `close` on `mySock`

```
closeSocket(mySock);
```

Obligation: call `close` on `mySock`

Lightweight Ownership

Obligation: call `close` on `mySock`

```
closeSocket(mySock);
```

Obligation: call `close` on `mySock`

```
void closeSocket(@Owning Socket s) {
```

Obligation: call `close` on `s`

```
  s.close();
}
```

Lightweight Ownership

Obligation: call `close` on `mySock`

```
closeSocket(mySock);
```

```
void closeSocket(@Owning Socket s) {
```

Obligation: call `close` on `s`

```
  s.close();
}
```

- Obligations are neither created nor destroyed
- Doesn't restrict privileges of other aliases

Precision via Local Alias Reasoning

- Local must-aliases
- Lightweight ownership
- **Resource aliasing**
- Obligation creation

Resource Aliasing

```
Socket socket = ...;
InputStreamReader stream =
    new InputStreamReader(socket.getInputStream());
...
```

Resource Aliasing

```
Socket socket = ...;
InputStreamReader stream =
    new InputStreamReader(socket.getInputStream());
...
```

Which of these should be closed?

Resource Aliasing

```
Socket socket = ...;
InputStreamReader stream =
    new InputStreamReader(socket.getInputStream());
...
```

Which of these should be closed?

- Closing either `socket` or `stream` is adequate
- Extensibility

Evaluation:

Evaluation: Case Studies

Four programs: zookeeper, hadoop-hdfs, hbase, plume-util

Lines of code	Resource Leaks Found	False positive warnings	Annotations
427,858	49	121	286

Evaluation: Case Studies

Four programs: zookeeper, hadoop-hdfs, hbase, plume-util

Lines of code	Resource Leaks Found	False positive warnings	Annotations
427,858	49	121	286

Precision: 29%

Evaluation: Case Studies

Four programs: zookeeper, hadoop-hdfs, hbase, plume-util

Lines of code	Resource Leaks Found	False positive warnings	Annotations
427,858	49	121	286

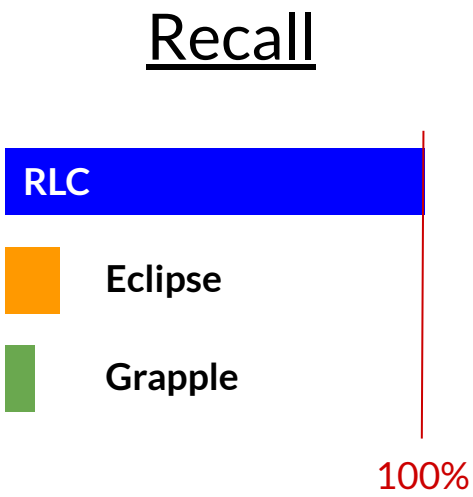
~1 per 1,500 LoC

Evaluation: Comparison

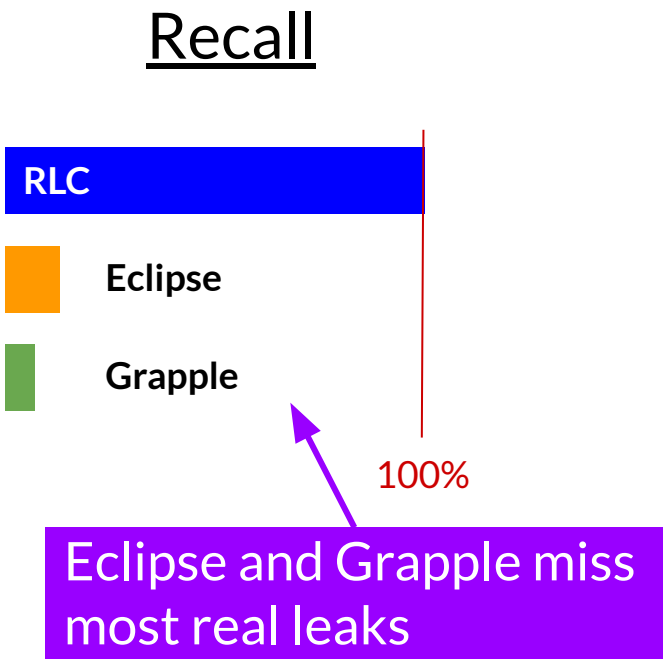
3 analyses:

- **RLC**, our type-based analysis
- **Eclipse**'s high-confidence heuristic bug-finder
- **Grapple**, a whole-program graph reachability analysis

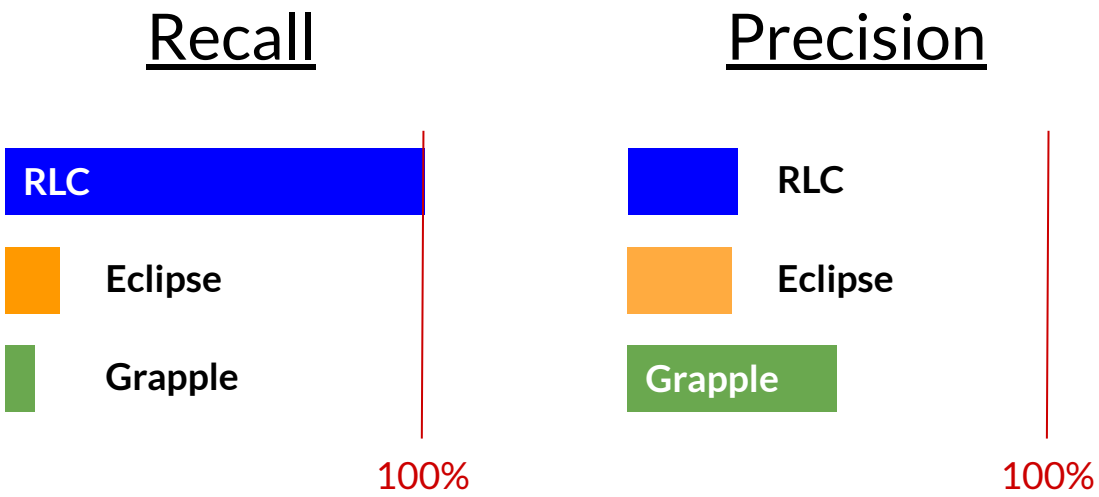
Evaluation: Comparison



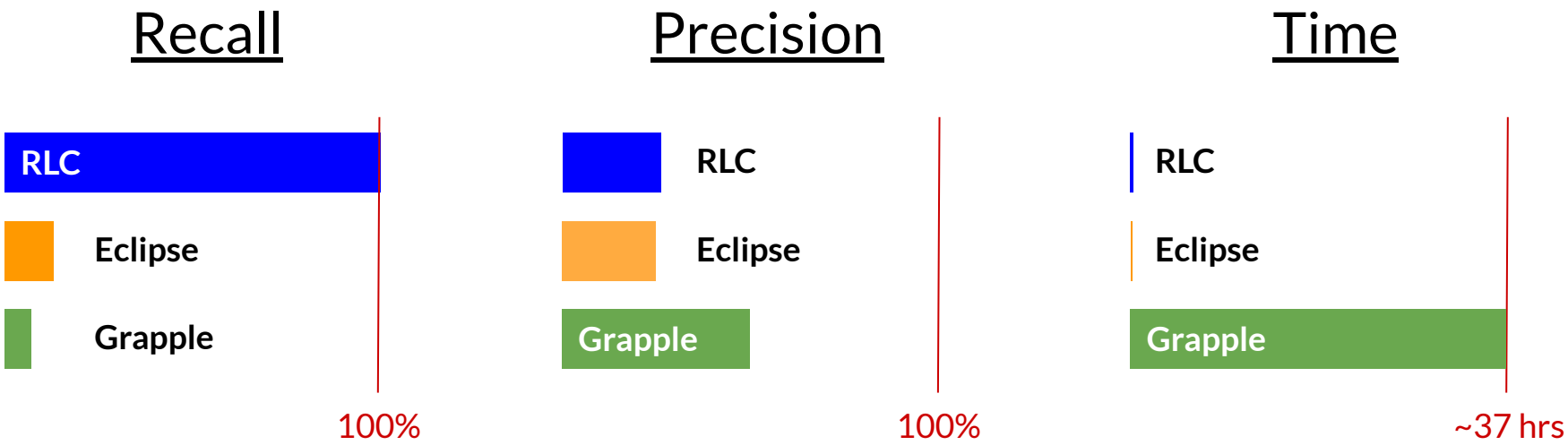
Evaluation: Comparison



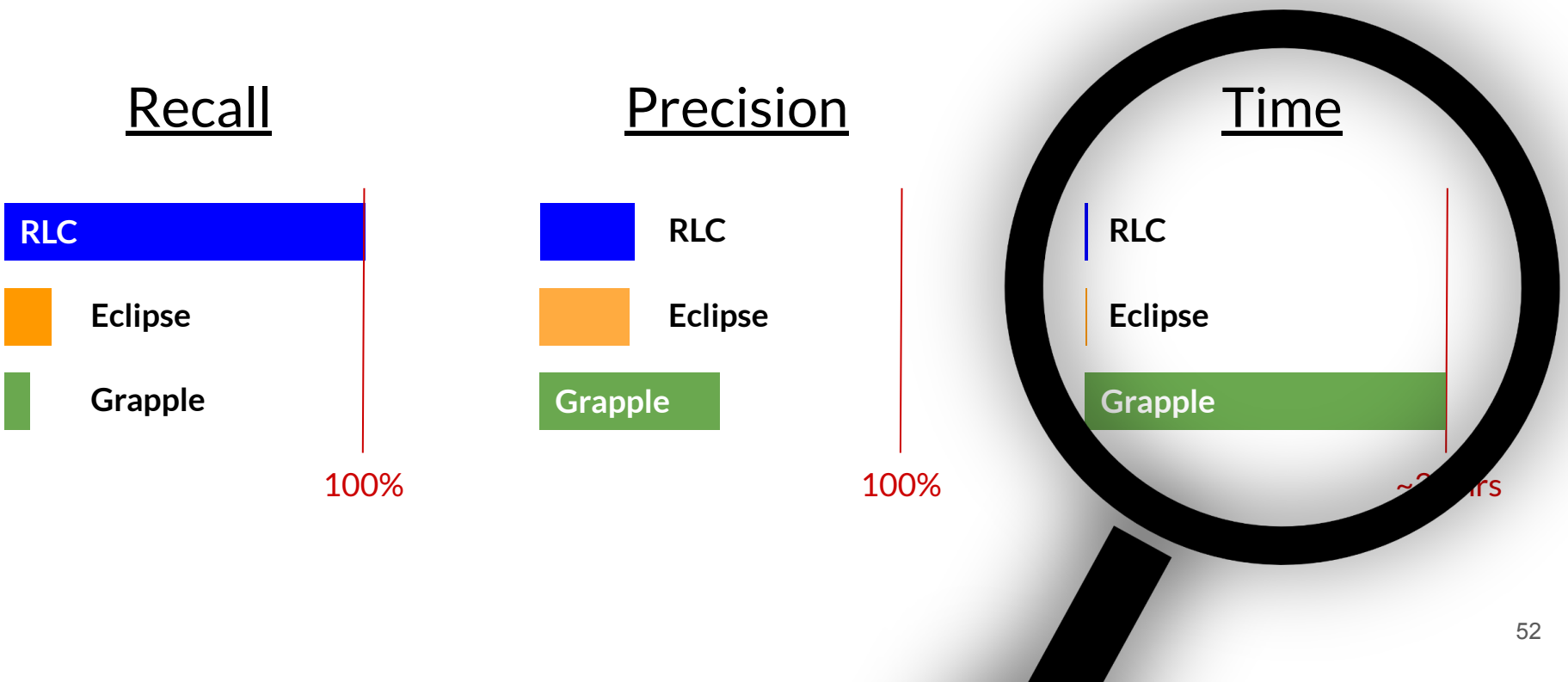
Evaluation: Comparison



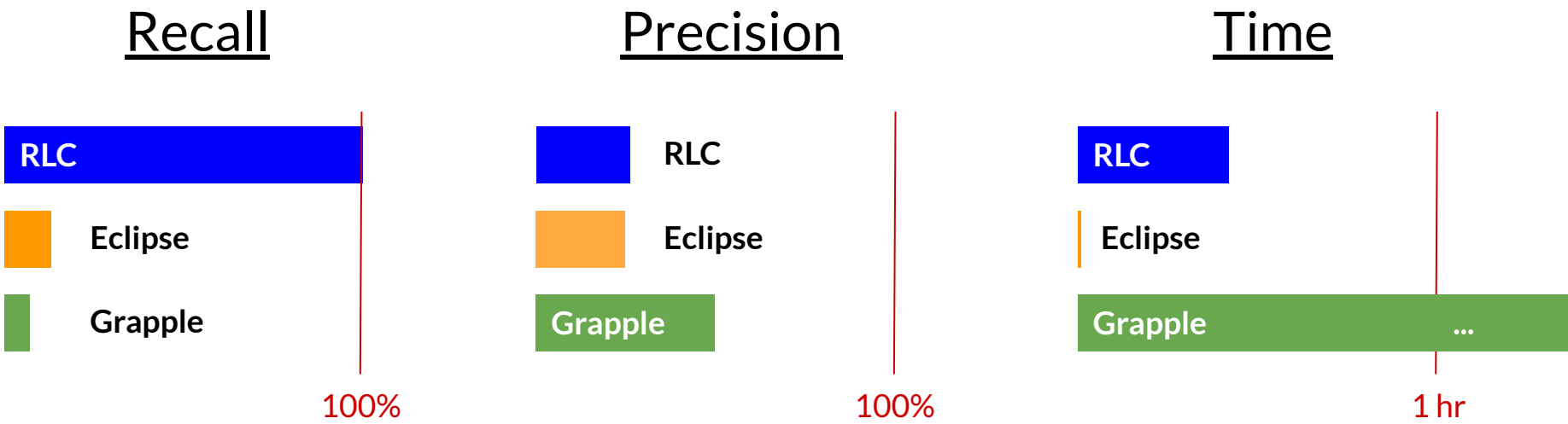
Evaluation: Comparison



Evaluation: Comparison



Evaluation: Comparison



Contributions

- **Lightweight and modular resource leak verification** via *accumulation analysis*
- **Local alias reasoning** for precision
- **Extensive evaluation**
- **Open-source implementation** at checkerframework.org