LeakChecker: Practical Static Memory Leak Detection for Managed Languages

Dacong (Tony) Yan[1], Guoqing Xu[2], Shengqian Yang[1], Atanas Rountev[1]

[1] **Ohio State University**
[2] **University of California, Irvine**

PRESTO: Program Analyses and Software Tools Research Group, Ohio State University

Memory Leaks in Managed Languages

- Languages such as Java still have memory leaks: unnecessary references keep unused objects alive
- Static leak detection
  - Widely used for unmanaged languages such as C
  - Cannot be applied to managed languages: no explicit memory deallocation
- General definition of leaks
  - Precision: difficult to compute object liveness precisely
  - Performance: limited scalability for large programs
- Our approach
  - Shift the focus, and identify common leak patterns

Proposed Leak Detection for Java

- Observation: leaks are often related to frequently occurring events (e.g., loop iterations)
- Solution: focus on a user-specified event loop
- Observation: a leaking object is often
  - created by one loop iteration
  - escapes this iteration
  - never used in later iterations
- Solution: interprocedural tracking of
  - whether an object escapes to a memory location outside of the loop
  - whether an escaping object flows from the outside location back into a later loop iteration

Example

```
“main”:
Transaction t = new Transaction();
for (int i = 0; i < X; ++i) {
  t.display();
  Order order = new Order(...);
  t.process(order);
}
```

```
class Transaction {
  Order prev;
  Customer[] custs = new Customer[…];
  void display() {
    Order r = this.prev;
    … // display r
    this.prev = null; // remove
  }
  void process(Order p) {
    this.prev = p;
    Customer c = this.custs[…];
    c.addOrder(p);
    ... // process p
  }  }
```

```
class Customer {
  Order[] orders = new Order[…];
  void addOrder(Order q) {
    this.orders[...] = q;
  }
}
```

An example adapted from SPECjbb2000

Example

```
“main”:
Transaction t = new Transaction();
for (int i = 0; i < X; ++i) {
  t.display();
  Order order = new Order(...);
  t.process(order);
}
```

```
class Transaction {
  Order prev;
  Customer[] custs = new Customer[…];
  void display() {
    Order r = this.prev;
    … // display r
    this.prev = null; // remove
  }
  void process(Order p) {
    this.prev = p;
    Customer c = this.custs[…];
    c.addOrder(p);
    ... // process p
  }  }
```

```
class Customer {
  Order[] orders = new Order[…];
  void addOrder(Order q) {
    this.orders[...] = q;
  }
}
```

An example adapted from SPECjbb2000

Example

```
"main":
Transaction t = new Transaction();
for (int i = 0; i < X; ++i) {
  t.display();
  Order order = new Order(...);
  t.process(order);
}
```

loop object

```
class Transaction {
  Order prev;
  Customer[] custs = new Customer[…];
  void display() {
    Order r = this.prev;
    … // display r
    this.prev = null; // remove
  }
  void process(Order p) {
    this.prev = p;
    Customer c = this.custs[…];
    c.addOrder(p);
    ... // process p
  }  }
```

```
class Customer {
  Order[] orders = new Order[…];
  void addOrder(Order q) {
    this.orders[...] = q;
  }
}
```

Example

outside object

```
“main”:
Transaction t = new Transaction();
for (int i = 0; i < X; ++i) {
  t.display();
  Order order = new Order(...);
  t.process(order);
}
```

loop object

```
class Transaction {
  Order prev;
  Customer[] custs = new Customer[…];
  void display() {
    Order r = this.prev;
    … // display r
    this.prev = null; // remove
  }
  void process(Order p) {
    this.prev = p;
    Customer c = this.custs[…];
    c.addOrder(p);
    ... // process p
  }  }
```

```
class Customer {
  Order[] orders = new Order[…];
  void addOrder(Order q) {
    this.orders[...] = q;
  }
}
```

Example

outside object

“main”:
```
Transaction t = new Transaction();
for (int i = 0; i < X; ++i) {
  t.display();
  Order order = new Order(...);
  t.process(order);
}
```

loop object

```
class Transaction {
  Order prev;
  Customer[] custs = new Customer[…];
  void display() {
    Order r = this.prev;
    … // display r
    this.prev = null; // remove
  }
  void process(Order p) {
    this.prev = p;
    Customer c = this.custs[…];
    c.addOrder(p);
    ... // process p
  }  }
```

```
class Customer {
  Order[] orders = new Order[…];
  void addOrder(Order q) {
    this.orders[...] = q;
  }
}
```

Example

outside object

loop object

“main”:

```
Transaction t = new Transaction();
for (int i = 0; i < X; ++i) {
  t.display();
  Order order = new Order(...);
  t.process(order);
}
```

```
class Transaction {
  Order prev;
  Customer[] custs = new Customer[…];
  void display() {
    Order r = this.prev;
    … // display r
    this.prev = null; // remove
  }
  void process(Order p) {
    this.prev = p;
    Customer c = this.custs[…];
    c.addOrder(p);
    ... // process p
  } }
```

```
class Customer {
  Order[] orders = new Order[…];
  void addOrder(Order q) {
    this.orders[...] = q;
  }
}
```

Example

“main”:

```
Transaction t = new Transaction();
for (int i = 0; i < X; ++i) {
  t.display();
  Order order = new Order(...);
  t.process(order);
}
```

```
class Transaction {
  Order prev;
  Customer[] custs = new Customer[…];
  void display() {
    Order r = this.prev;
    … // display r
    this.prev = null; // remove
  }
  void process(Order p) {
    this.prev = p;
    Customer c = this.custs[…];
    c.addOrder(p);
    ... // process p
  }  }
```

```
class Customer {
  Order[] orders = new Order[…];
  void addOrder(Order q) {
    this.orders[...] = q;
  }
}
```

Example

```
"main":
Transaction t = new Transaction();
for (int i = 0; i < X; ++i) {
  t.display();
  Order order = new Order(...);
  t.process(order);
}
```

```
class Transaction {
  Order prev;
  Customer[] custs = new Customer[…];
  void display() {
    Order r = this.prev;
    … // display r
    this.prev = null; // remove
  }
  void process(Order p) {
    this.prev = p;
    Customer c = this.custs[…];
    c.addOrder(p);
    ... // process p
  }  }
```

```
class Customer {
  Order[] orders = new Order[…];
  void addOrder(Order q) {
    this.orders[...] = q;
  }
}
```

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }

21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

```
7 class Transaction {
8    Order prev;
9    Customer[] custs = new Customer[…];
10   void display() {
11     Order r = this.prev;
12     … // display r
13     this.prev = null; // remove
14   }
15   void process(Order p) {
16     this.prev = p;  store16
17     Customer c = this.custs[…];
18     c.addOrder(p);
19     ... // process p
20   } }
```

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }

21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

```
7 class Transaction {
8   Order prev;
9   Customer[] custs = new Customer[…];
10  void display() {
11    Order r = this.prev;
12    … // display r
13    this.prev = null; // remove
14  }
15  void process(Order p) {
16    this.prev = p;
17    Customer c = this.custs[…];
18    c.addOrder(p);
19    ... // process p
20 } }
```

store_{16}

$$\begin{tikzcd}
\mathrm{Order}_4 \arrow[r, "\mathit{store}_{16}"] \arrow[r, dashed, bend right, "\mathit{escape}"'] & \mathrm{Transaction}_1
\end{tikzcd}$$

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }
```

```
7 class Transaction {
8   Order prev;
9   Customer[] custs = new Customer[…];
10  void display() {
11    Order r = this.prev;
12    … // display r
13    this.prev = null; // remove
14  }
15  void process(Order p) {
16    this.prev = p;
17    Customer c = this.custs[…];
18    c.addOrder(p);
19    ... // process p
20  } }
```

```
21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

$store_{24}$

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }
```

```
7 class Transaction {
8   Order prev;
9   Customer[] custs = new Customer[…];
10   void display() {
11     Order r = this.prev;
12     … // display r
13     this.prev = null; // remove
14   }
15   void process(Order p) {
16     this.prev = p;
17     Customer c = this.custs[…];
18     c.addOrder(p);
19     ... // process p
20 }  }
```

```
21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;  store_24
25   }
26 }
```

$$\begin{tikzcd}
\mathtt{Order}_4 \arrow[r, "store_{24}"] \arrow[rrrr, dashed, bend right, "escape"'] & \mathtt{Order[]}_{23} \arrow[r, "store_{22}"] & \mathtt{Customer}_? \arrow[r, "store_?"] & \mathtt{Customer[]}_{10} \arrow[r, "store_9"] & \mathtt{Transaction}_1
\end{tikzcd}$$

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }
```

```
 7 class Transaction {
 8   Order prev;
 9   Customer[] custs = new Customer[…];
10   void display() {
11     Order r = this.prev;
12     … // display r
13     this.prev = null; // remove
14   }
15   void process(Order p) {
16     this.prev = p;
17     Customer c = this.custs[…];
18     c.addOrder(p);
19     ... // process p
20   } }
```

```
21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

$$\begin{tikzcd}
\texttt{Order}_4 \arrow[r, bend left, dashed, "\mathit{store}_{24}/\mathrm{orders},\ \ldots,\ \mathit{store}_{9}/\mathrm{custs}"] \arrow[r, bend right, dashed, "\mathit{store}_{16}/\mathrm{prev}"'] & \texttt{Transaction}_1
\end{tikzcd}$$

Example

```
“main”:
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }
```

```
 7 class Transaction {
 8   Order prev;
 9   Customer[] custs = new Customer[…];
10   void display() {
11     Order r = this.prev;
12     … // display r
13     this.prev = null; // remove
14   }
15   void process(Order p) {
16     this.prev = p;
17     Customer c = this.custs[…];
18     c.addOrder(p);
19     ... // process p
20   }  }
```

```
21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

$$\begin{tikzcd}
\text{Order}_4 \arrow[r, dashed, bend left, "\textit{store}_{24}/\text{orders}, \ldots, \textit{store}_{9}/\text{custs}"] \arrow[r, phantom, "\textit{leaking?}", pos=0.9, yshift=2ex] & \text{Transaction}_1
\end{tikzcd}$$

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }
```

```
7 class Transaction {
8   Order prev;
9   Customer[] custs = new Customer[…];
10  void display() {
11    Order r = this.prev;
12    … // display r
13    this.prev = null; // remove
14  }
15  void process(Order p) {
16    this.prev = p;
17    Customer c = this.custs[…];
18    c.addOrder(p);
19    ... // process p
20  } }
```

```
21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

$$\begin{tikzcd}
\mathtt{Order}_4 \arrow[r, dashed, bend right, "store_{16}/prev"'] & \mathtt{Transaction}_1
\end{tikzcd}$$

leaking?

Example

```
“main”:
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }

21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

```
7 class Transaction {
8   Order prev;
9   Customer[] custs = new Customer[…];
10  void display() {
11    Order r = this.prev;
12    … // display r
13    this.prev = null; // remove
14  }
15  void process(Order p) {
16    this.prev = p;
17    Customer c = this.custs[…];
18    c.addOrder(p);
19    ... // process p
20  }  }
```

$\boxed{\texttt{Order r = this.prev;}}$ *$load_{11}$*

$$\begin{tikzcd}\texttt{Order}_4 \arrow[r, dashed, bend right, "store_{16}/prev"'] & \texttt{Transaction}_1\end{tikzcd}$$

leaking?

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }
```

```
7 class Transaction {
8   Order prev;
9   Customer[] custs = new Customer[…];
10  void display() {
11    Order r = this.prev;
12    … // display r
13    this.prev = null; // remove
14  }
15  void process(Order p) {
16    this.prev = p;
17    Customer c = this.custs[…];
18    c.addOrder(p);
19    ... // process p
20  } }
```

$load_{11}$

$store_{16}$

```
21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

$$\begin{tikzcd}\mathtt{Order}_4 \arrow[r, dashed, bend right, "store_{16}/prev"'] & \mathtt{Transaction}_1\end{tikzcd}$$

leaking?

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }
```

```
7 class Transaction {
8   Order prev;
9   Customer[] custs = new Customer[…];
10  void display() {
11    Order r = this.prev;          load11
12    … // display r
13    this.prev = null; // remove
14  }
15  void process(Order p) {
16    this.prev = p;                store16
17    Customer c = this.custs[…];
18    c.addOrder(p);
19    ... // process p
20  } }
```

```
21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

$$\begin{tikzcd} \mathrm{Order}_4 \arrow[r, bend right, dashed, "store_{16}/prev"'] & \mathrm{Transaction}_1 \arrow[l, bend right, dashed, "load_{11}/prev"'] \end{tikzcd}$$

leaking?

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }
```

```
7 class Transaction {
8   Order prev;
9   Customer[] custs = new Customer[…];
10  void display() {
11    Order r = this.prev;
12    … // display r
13    this.prev = null; // remove
14  }
15  void process(Order p) {
16    this.prev = p;
17    Customer c = this.custs[…];
18    c.addOrder(p);
19    ... // process p
20  } }
```

$load_{11}$

$store_{16}$

```
21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

(i+1)-th iteration

$$\begin{tikzcd}
\mathtt{Order}_4 \arrow[r, bend right, dashed, "store_{16}/prev"'] & \mathtt{Transaction}_1 \arrow[l, bend right, dashed, "load_{11}/prev"']
\end{tikzcd}$$

i-th iteration

leaking?

Example

```
"main":
1 Transaction t = new Transaction();
2 for (int i = 0; i < X; ++i) {
3   t.display();
4   Order order = new Order(...);
5   t.process(order);
6 }
```

```
 7 class Transaction {
 8   Order prev;
 9   Customer[] custs = new Customer[…];
10   void display() {
11     Order r = this.prev;
12     … // display r
13     this.prev = null; // remove
14   }
15   void process(Order p) {
16     this.prev = p;
17     Customer c = this.custs[…];
18     c.addOrder(p);
19     ... // process p
20   } }
```

```
21 class Customer {
22   Order[] orders = new Order[…];
23   void addOrder(Order q) {
24     this.orders[...] = q;
25   }
26 }
```

$$\begin{tikzcd}
\mathtt{Order}_4 \arrow[r, bend left, dashed, "\mathit{store}_{24}/\text{orders}, \ldots, \mathit{store}_{9}/\text{custs}"] \arrow[r, bend right, dashed, "\mathit{store}_{16}/\text{prev}"'] & \mathtt{Transaction}_1
\end{tikzcd}$$

leaking

non-leaking

Static Analysis Outline

- Object: pair of allocation site and calling context

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow

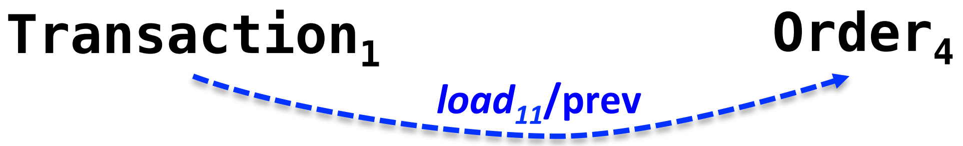
$$\begin{tikzcd}
\texttt{Order}_4 \arrow[r, "store_{24}"] & \texttt{Order[]}_{23} \arrow[r, "store_{22}"] & \texttt{Customer}_? \arrow[r, "store_?"] & \texttt{Customer[]}_{10} \arrow[r, "store_9"] & \texttt{Transaction}_1
\end{tikzcd}$$

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow
- Flows-in path
  - Escaping object flows from outside object into the loop
  - Sequence of load statements causing the flow

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow
- Flows-in path
  - Escaping object flows from outside object into the loop
  - Sequence of load statements causing the flow


$$\begin{tikzcd}\texttt{Transaction}_1 \arrow[r, dashed, bend right, "load_{11}/\text{prev}"'] & \texttt{Order}_4\end{tikzcd}$$

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow
- Flows-in path
  - Escaping object flows from outside object into the loop
  - Sequence of load statements causing the flow
- Loop iteration constraints
  - Flows-in valid only if the corresponding Flows-out occurs in an earlier iteration

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow
- Flows-in path
  - Escaping object flows from outside object into the loop
  - Sequence of load statements causing the flow
- Loop iteration constraints
  - Flows-in valid only if the corresponding Flows-out occurs in an earlier iteration

outside object

`Transaction`$_1$

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow
- Flows-in path
  - Escaping object flows from outside object into the loop
  - Sequence of load statements causing the flow
- Loop iteration constraints
  - Flows-in valid only if the corresponding Flows-out occurs in an earlier iteration

outside object

`Order`$_4$ `Transaction`$_1$

i-th iteration

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow
- Flows-in path
  - Escaping object flows from outside object into the loop
  - Sequence of load statements causing the flow
- Loop iteration constraints
  - Flows-in valid only if the corresponding Flows-out occurs in an earlier iteration

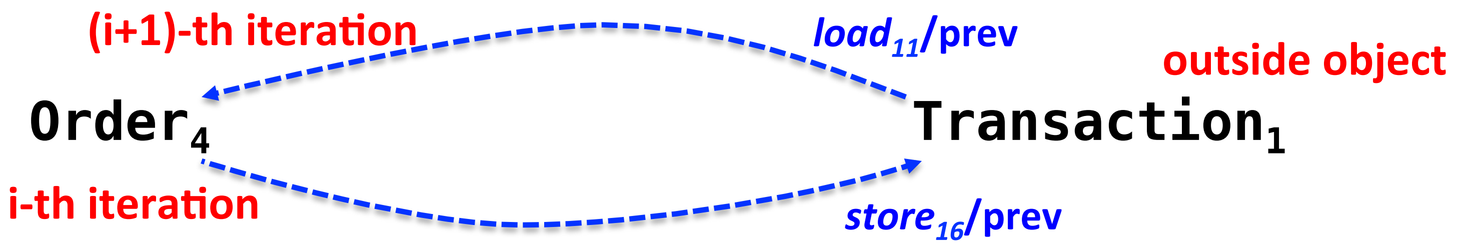
outside object

$$\begin{tikzcd} \mathtt{Order}_4 \arrow[r, dashed, bend right, "\mathit{store}_{16}/\mathbf{prev}"'] & \mathtt{Transaction}_1 \end{tikzcd}$$

i-th iteration

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow
- Flows-in path
  - Escaping object flows from outside object into the loop
  - Sequence of load statements causing the flow
- Loop iteration constraints
  - Flows-in valid only if the corresponding Flows-out occurs in an earlier iteration


$$\begin{tikzcd}
\texttt{Order}_4 \arrow[r, bend right, dashed, "\textit{store}_{16}\text{/prev}"'] & \texttt{Transaction}_1 \arrow[l, bend right, dashed, "\textit{load}_{11}\text{/prev}"']
\end{tikzcd}$$

Static Analysis Outline

- Object: pair of allocation site and calling context
- Flows-out path
  - Loop object escaping to outside object
  - Sequence of store statements for the flow
- Flows-in path
  - Escaping object flows from outside object into the loop
  - Sequence of load statements causing the flow
- Loop iteration constraints
  - Flows-in valid only if the corresponding Flows-out occurs in an earlier iteration
  - Captured by extended recency abstraction (ERA)

Static Analysis Outline (cont.)

- On-demand analysis
  - Detect leaks only for objects allocated in specified loops
  - Context-free language reachability to match relevant store/load and call/return
- Conservative handling of thread lifetime
  - Assume the lifetime of each thread exceeds the loop lifetime
  - Capture objects leaking to long-running threads

Analysis Implementation

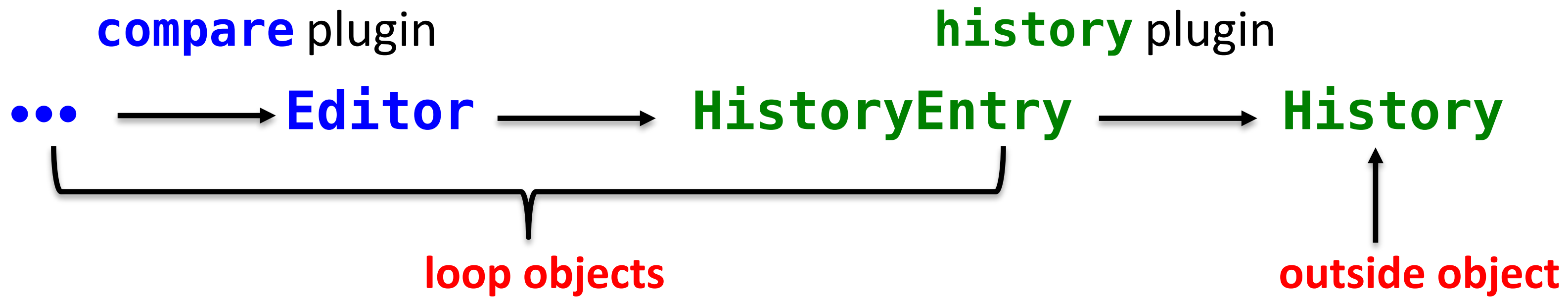
- Memory leak
  - An object leaks if it starts a flows-out path, but does not have a matching flows-in path
- Reporting leaks
  - Leaking object, with calling context of its allocation
  - Outside target object, with calling context of its allocation
  - Escape-causing heap write (store) statement, with its calling context
- LeakChecker: leak detection tool built using Soot

Evaluation

- 8 real-world Java programs
  - Enterprise trading, software tools, databases, logging
  - 3 programs never studied by existing work
- Effective for leak detection?
  - LeakChecker detected both known and new leaks
- Suitable for practical use?
  - Analysis running time (all < 35 mins)
  - Reasonable false positive rate (avg < 50%)
- Case studies
  - Performed to understand quality of leak report
  - For each leak defect, pinpoint root cause and fix the problem in < 2 hours

Eclipse Diff

- Scenario: compare two large JAR files
  - `runCompare` method in **`compare`** plugin
  - An artificial loop to invoke `runCompare` multiple times
- Loop objects
  - `Editor`: display comparison results
  - `HistoryEntry`: represent list of opened editors, and allow users to navigate them backward/forward
- Outside object
  - `History`: managed by another plugin to save `HistoryEntry`



Conclusions

- LeakChecker is both effective and practical
- Key insights
  - Capture common patterns of leaking behavior
  - Focus on user-specified event loops
  - Report leaks with detailed context information

Thank you