ExChain: Exception Dependency Analysis for Root Cause Diagnosis

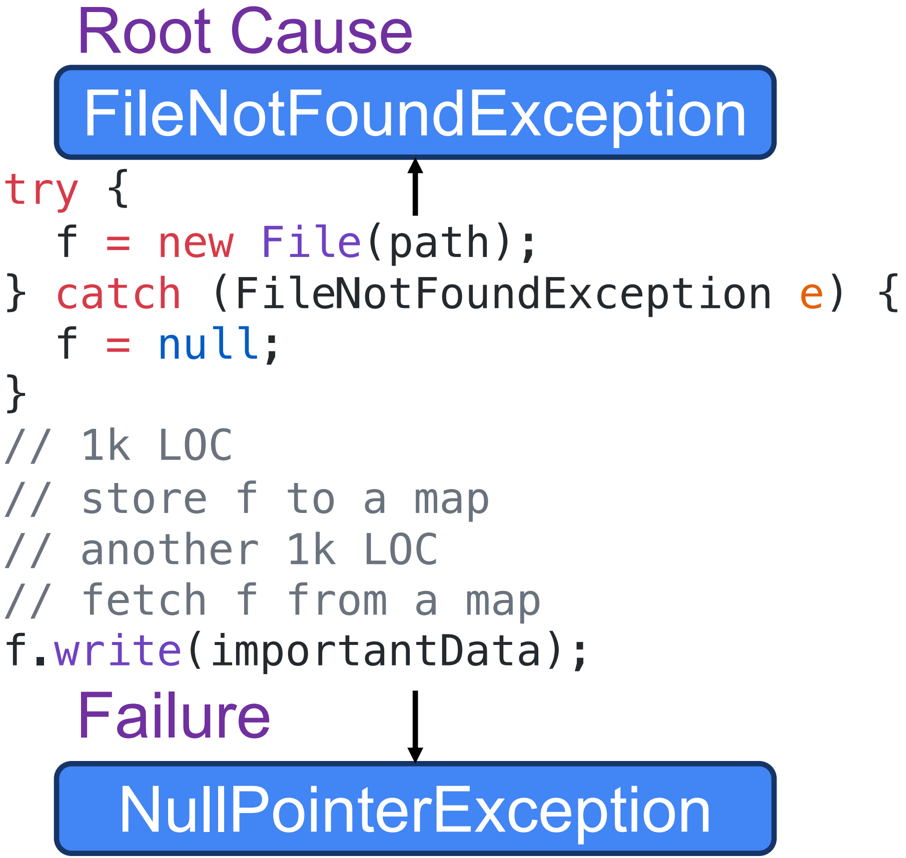
Ao Li, Shan Lu, Suman Nath, Rohan Padhye, Vyas Sekar

Exception/Error Handling Makes System Robust

```
try {
   f = new File(path);
} catch (FileNotFoundException e) {
   f = null;
}
```

Have you written this code before?

Mishandled Exceptions Cause Exception Dependent Failures



Root Cause

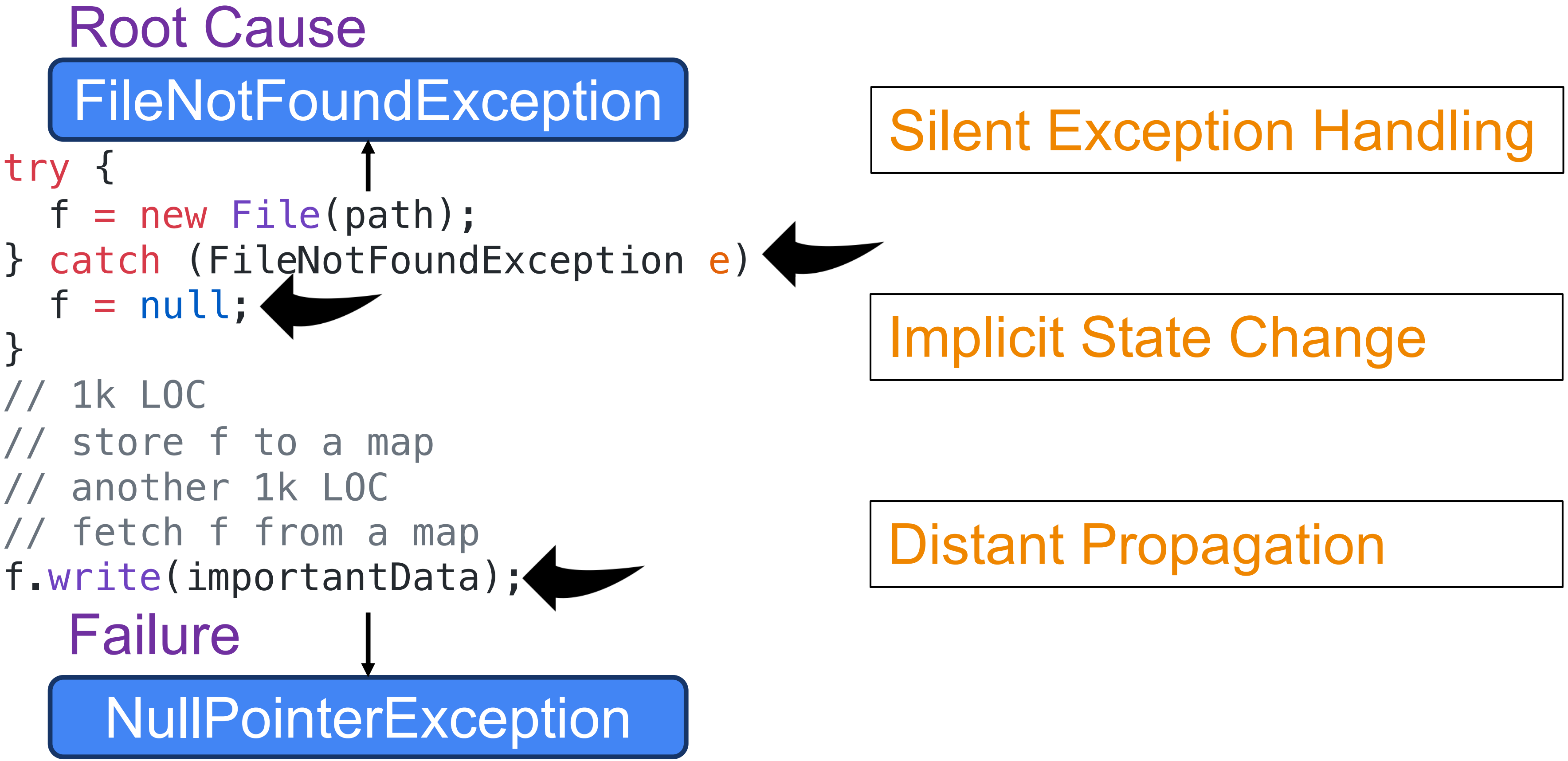
FileNotFoundException

```
try {
  f = new File(path);
} catch (FileNotFoundException e) {
  f = null;
}
// 1k LOC
// store f to a map
// another 1k LOC
// fetch f from a map
f.write(importantData);
```

Failure

NullPointerException

Acquiring the Exception Dependency Chain is Challenging



Root Cause

FileNotFoundException

```
try {
   f = new File(path);
} catch (FileNotFoundException e)
   f = null;
}
// 1k LOC
// store f to a map
// another 1k LOC
// fetch f from a map
f.write(importantData);
```

Failure

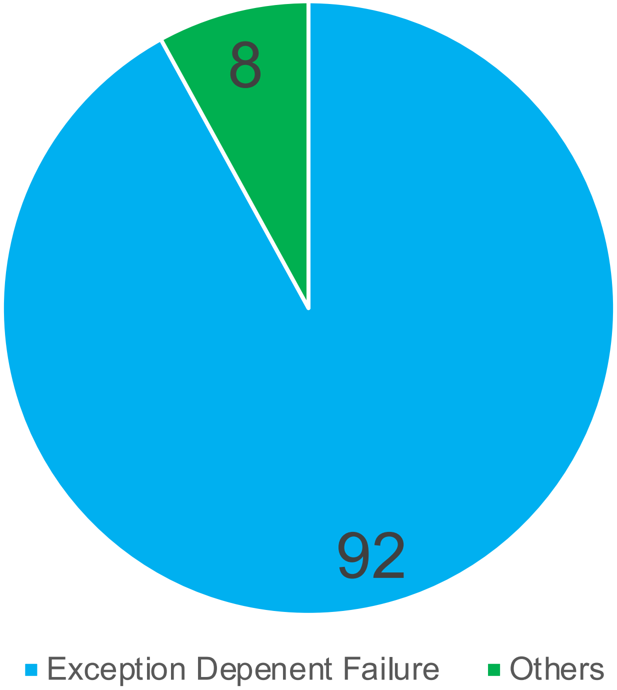
NullPointerException

Silent Exception Handling

Implicit State Change

Distant Propagation

Exception Dependent Failures Are Prevalent

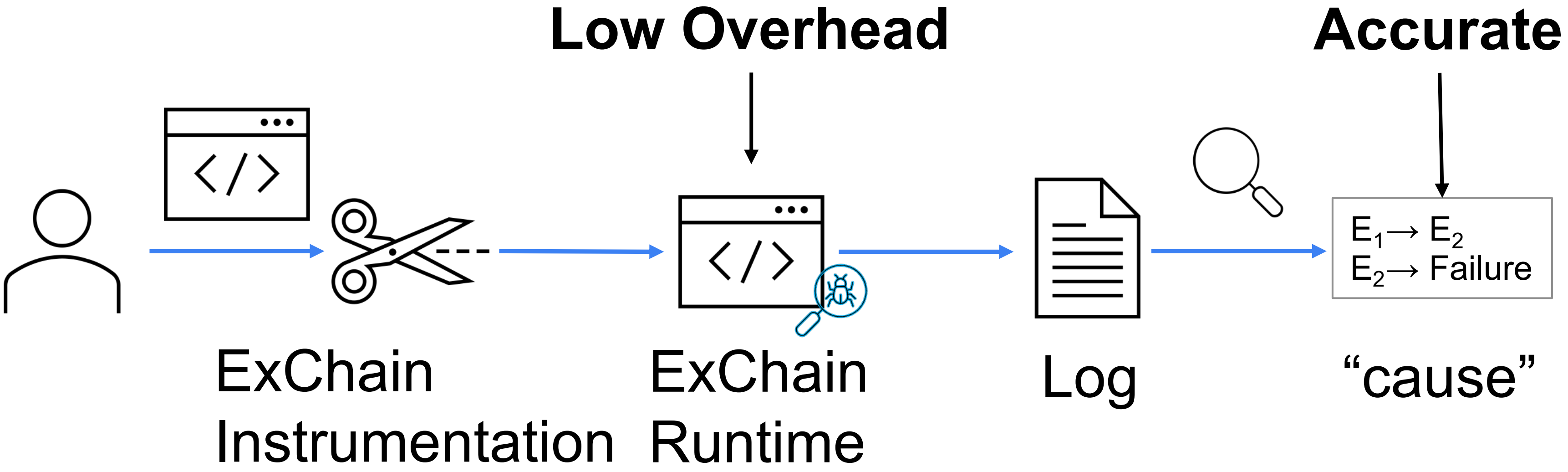


*Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. [OSDI 14']

Existing Techniques Don’t Handle EDFs

	Silent	Implicit	Distant
Log Analysis	✗	?	✗
Failure Monitoring	✓	?	✗
Request Tracing	✗	✗	?
Exception Analysis	✓	✗	✗
ExChain	✓	✓	✓

Our Work: ExChain



Low Overhead

Accurate

ExChain Instrumentation

ExChain Runtime

Log

$E_1 \rightarrow E_2$

$E_2 \rightarrow$ Failure

“cause”

Using low overhead instrumentation to the code, ExChain reports all exceptions related to the EDFs automatically.

ExChain: Key Ideas

Silent Exception Handling	Proactive Exception Monitoring
Implicit State Change	Affected/Responsible State Analysis
Distant Propagation	Hybrid Taint Analysis

This Talk

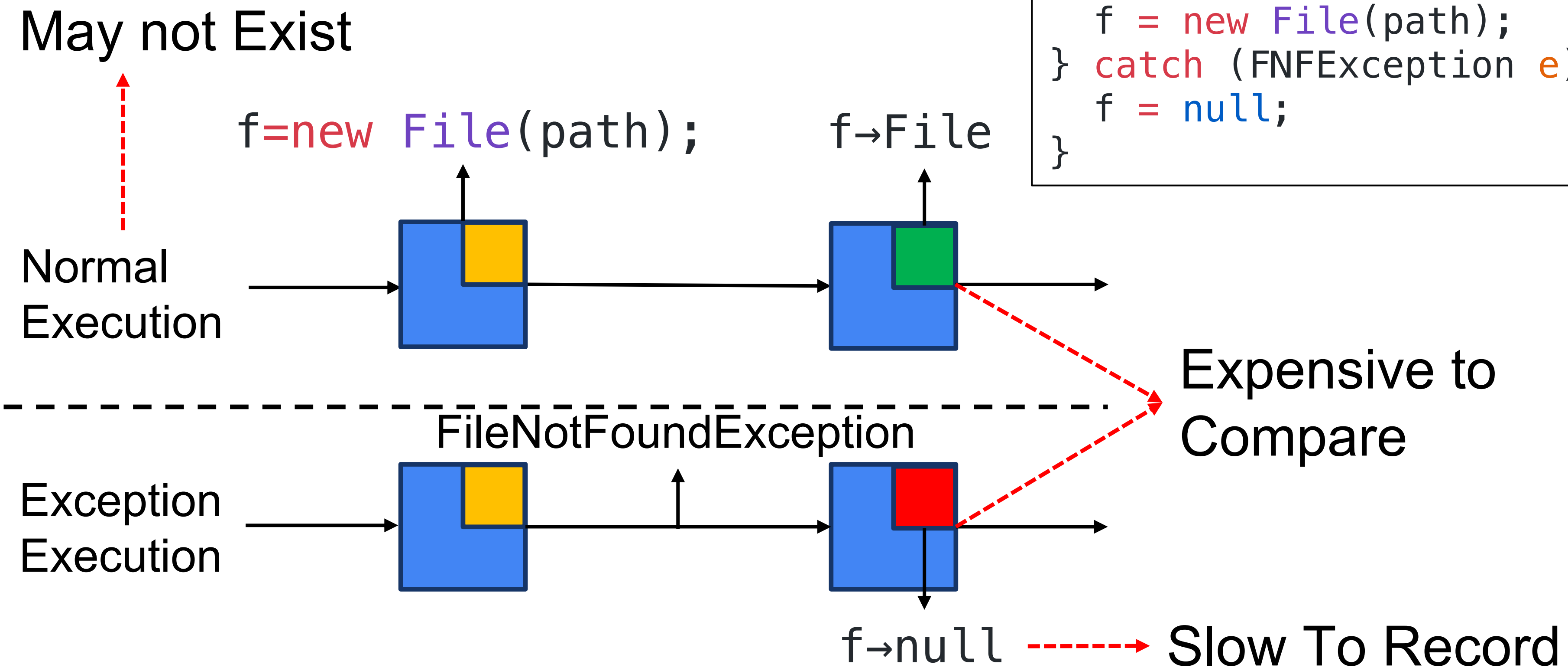
Silent Exception Handling — Proactive Exception Monitoring

Implicit State Change → Affected/Responsible State Analysis

Distant Propagation — Hybrid Taint Analysis

Problem: Affected State Analysis

```
try {
   f = new File(path);
} catch (FNFException e) {
   f = null;
}
```



Low Overhead Approximation Using Liveness Analysis

```
try {
  f = new File(path);
} catch (FileNotFoundException e) {
  f = null;
}
```

Only live when exception happens!

Insight: Compare “live” variables in exception control flow vs. normal control flow.

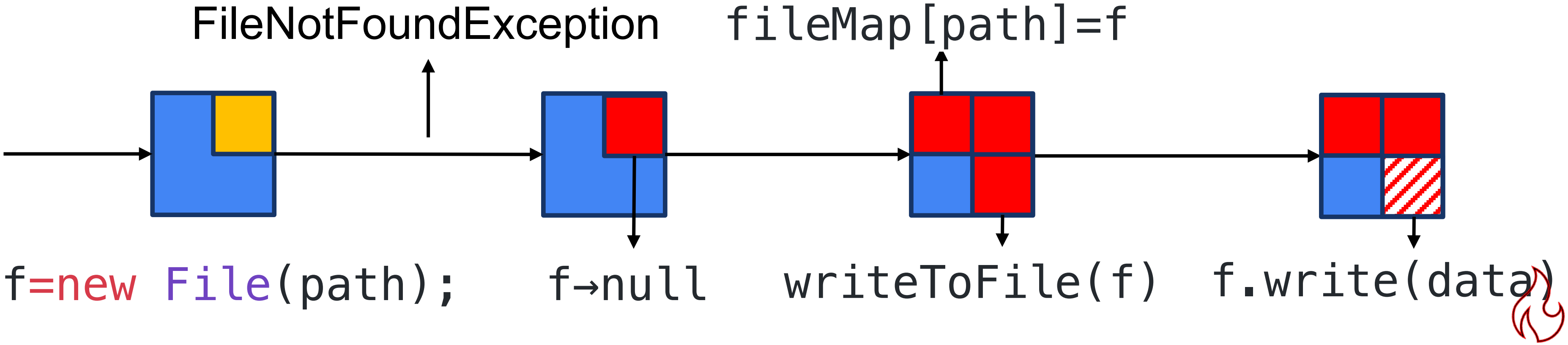
This Talk

Silent Exception Handling — Proactive Exception Monitoring

Implicit State Change — Affected/Responsible State Analysis

Distant Propagation — Hybrid Taint Analysis

Track the Propagation of Affected States



FileNotFoundException

`fileMap[path]=f`

`f=new File(path);`

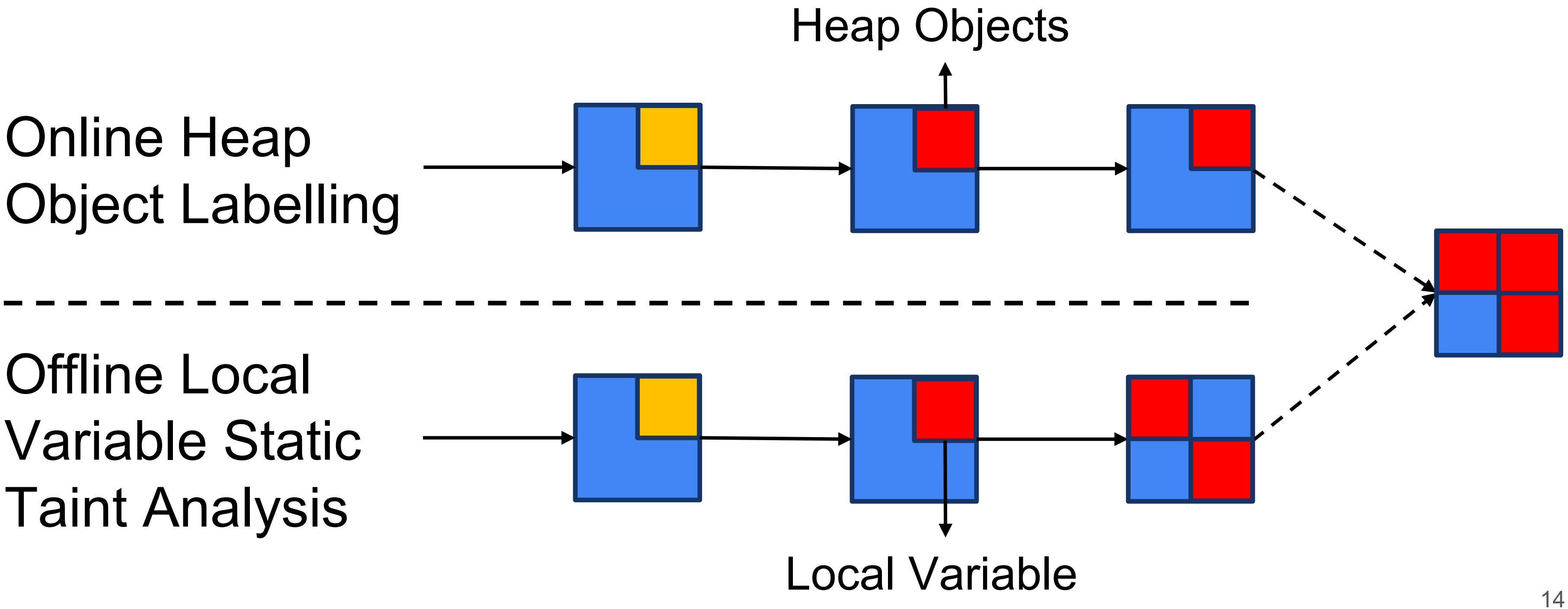
`f→null`

`writeToFile(f)`

`f.write(data)`

Problem: efficient and accurate information flow analysis for production systems!

Idea: Hybrid Taint Analysis Using Dynamic Labeling + Static Taint Tracking



Online Heap Object Labelling

Offline Local Variable Static Taint Analysis

Heap Objects

Local Variable

ExChain Summary of Key Ideas

- Proactively monitors all exceptions thrown by the application
- Identifies affected states using liveness analysis
- Track the propagation of affected state using hybrid taint analysis
- Identifies responsible states using backward data-flow analysis

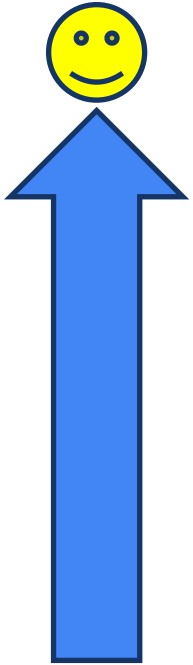
Evaluation

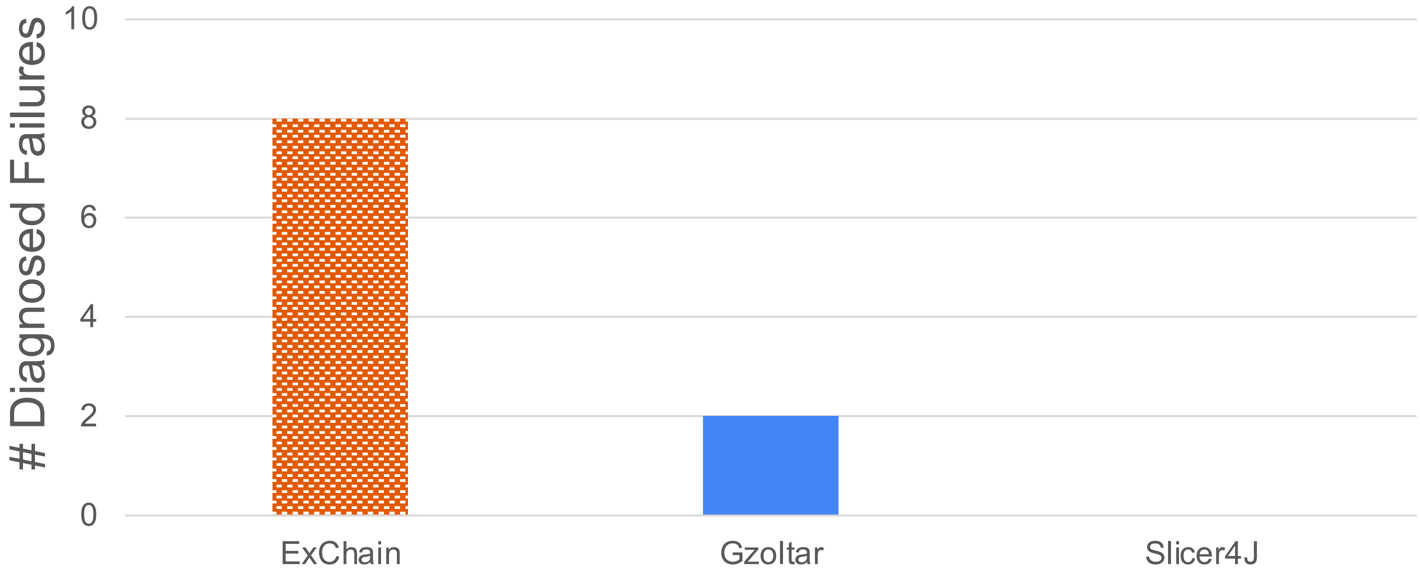
- 11 reproducible exception-dependent failures in real-world systems: e.g., Hadoop, MapReduce, Fineract

- How does ExChain compare to two state-of-the-art (SoTA) failure diagnosis tools for exception-dependent failures?

- How do key ideas contribute to low overhead and accuracy?
  - Is hybrid taint analysis better than static/dynamic taint analysis tool?

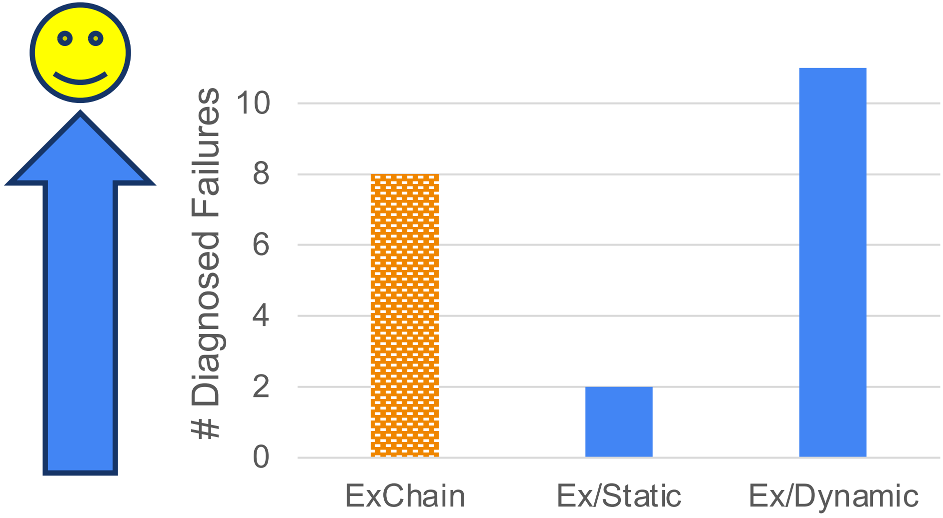
ExChain Successfully Diagnosed 8/11 Failures

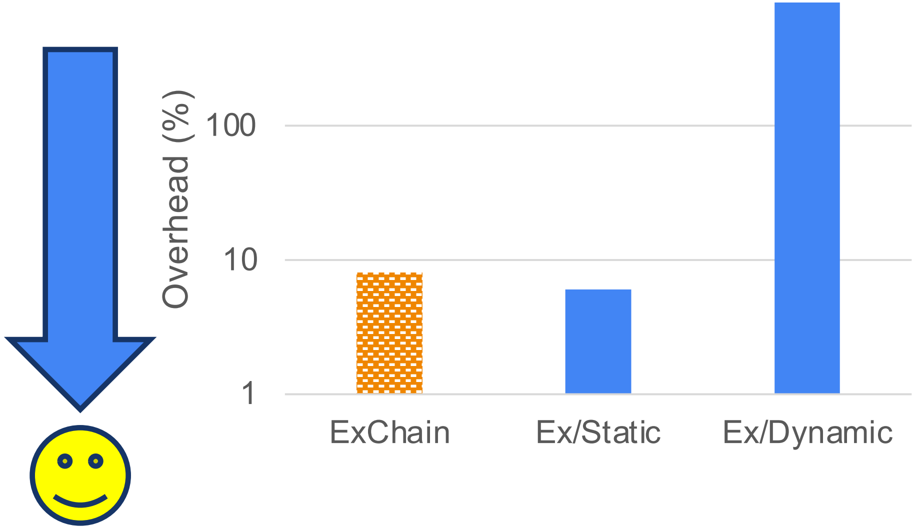




*Slicer4J failed to analyze 10/11 applications due to incompatible Java version and missing features.

ExChain: High Accuracy and Low Overhead





*Ex/Static, Ex/Dynamic uses ExChain affected/responsible state analysis algorithm with static/dynamic taint analysis.

Conclusions

- Exception Dependent Failures: Good software engineering practice has unintended consequences

- EDFs are challenging to debug with existing tools:
  Silent handling, Implicit state change, Distant effects

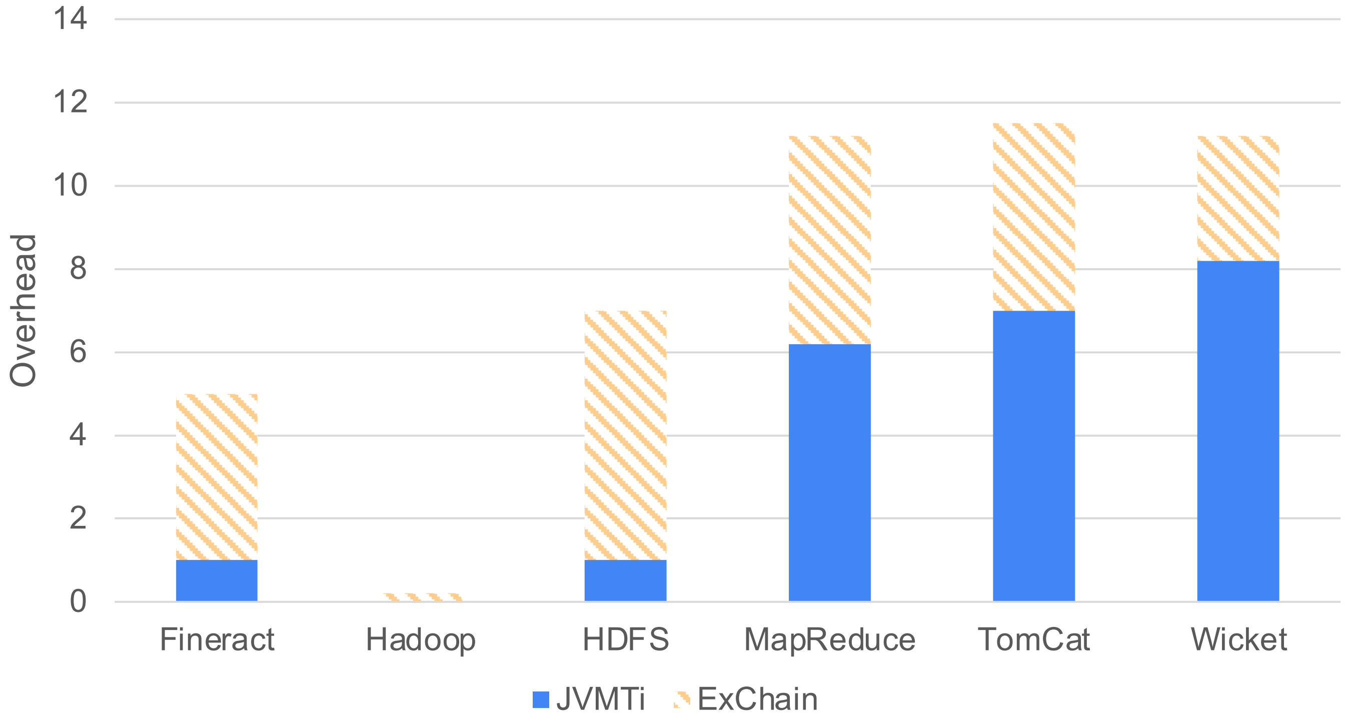
- ExChain: Low overhead + Accurate syste
  - Synthesis of static, hybrid analysis for

- Open Source!

Paper

Code

Debuggability and Observability Language/VM Design


Overhead
Fineract
Hadoop
HDFS
MapReduce
TomCat
Wicket
JVMTi
ExChain