ReproLite : A Lightweight Tool to Quickly Reproduce Hard System Bugs

Kaituo Li (U. Massachusetts, Amherst)
Pallavi Joshi (NEC Labs America)
Aarti Gupta (NEC Labs America)
Malay K. Ganai (NEC Labs America)

Outline

Reproduce what bugs? Why IReproLite?

Motivation

- programmable scheduler
  - e.g. PLDI 2013 "Concurrit"
- static-analysis
  - e.g. ASPLOS 2010 "Sherlog"

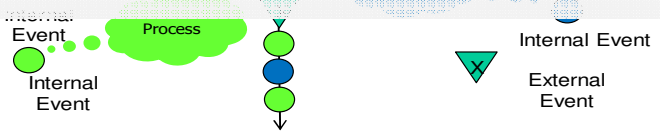
- Provides **assistance** to understand the cause of a bug from **logs**
- Focus on **system-level concurrency** issues

RE: Reproduction Engine

External events: cause them to happen

Internal events: wait for them

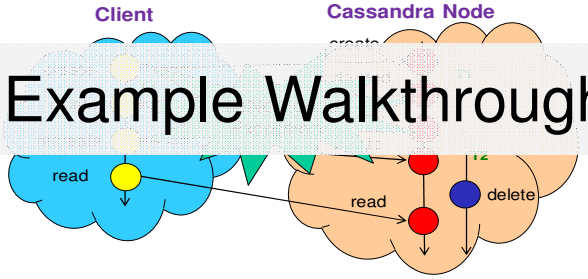
Structure



9

Cassandra bug 1477

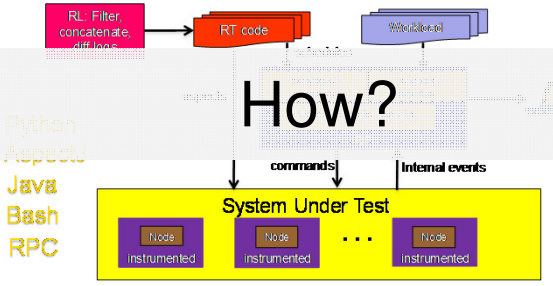
Example Walkthrough



9

Implementation

How?



18

Experiments

We experimented with using ReproLite to debug 11 bugs from

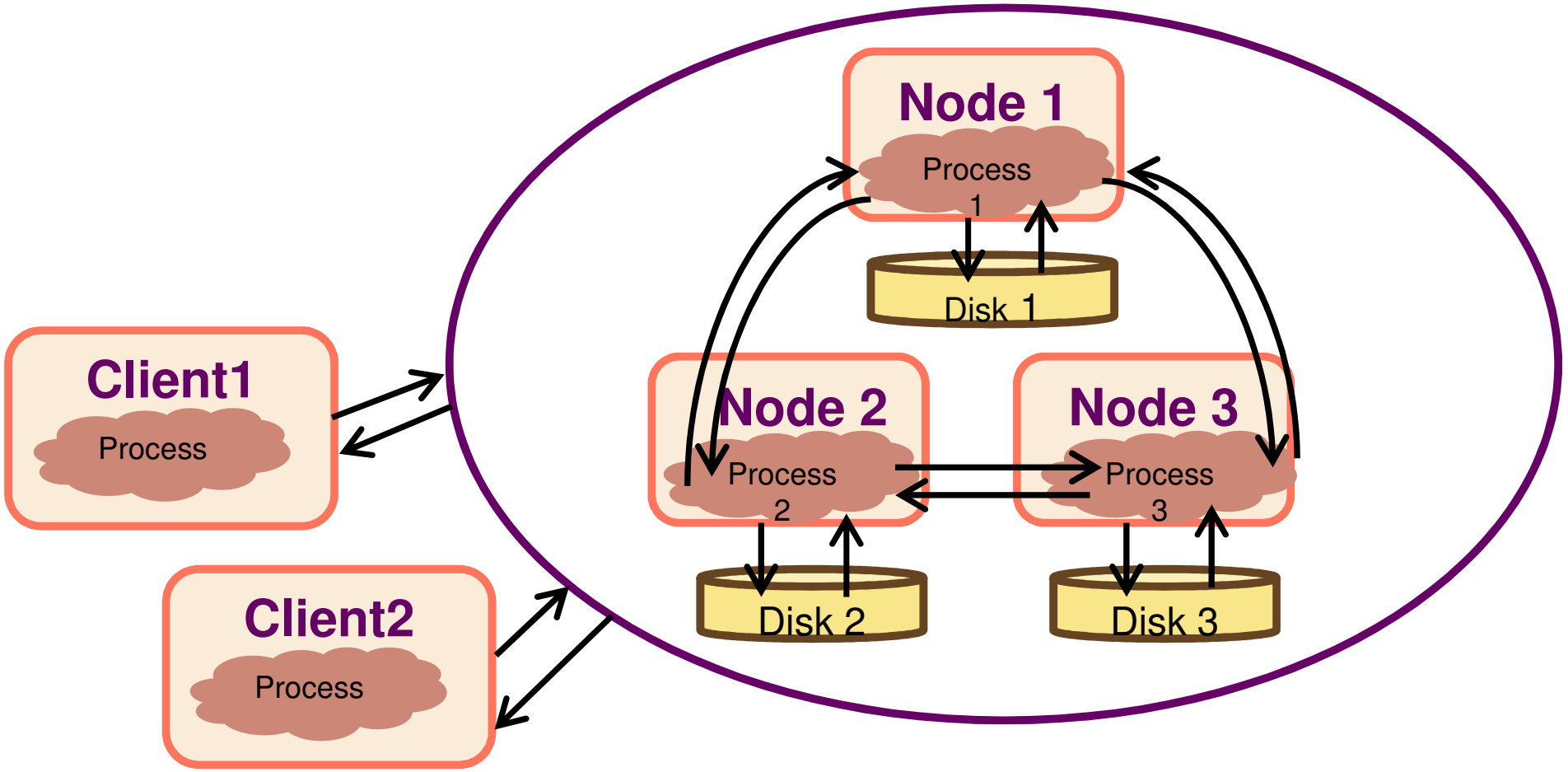
So what?

Instrumentation Overhead	Almost nil for half of the bugs
Usefulness of Logs	70% percent of bug causes hidden in logs.

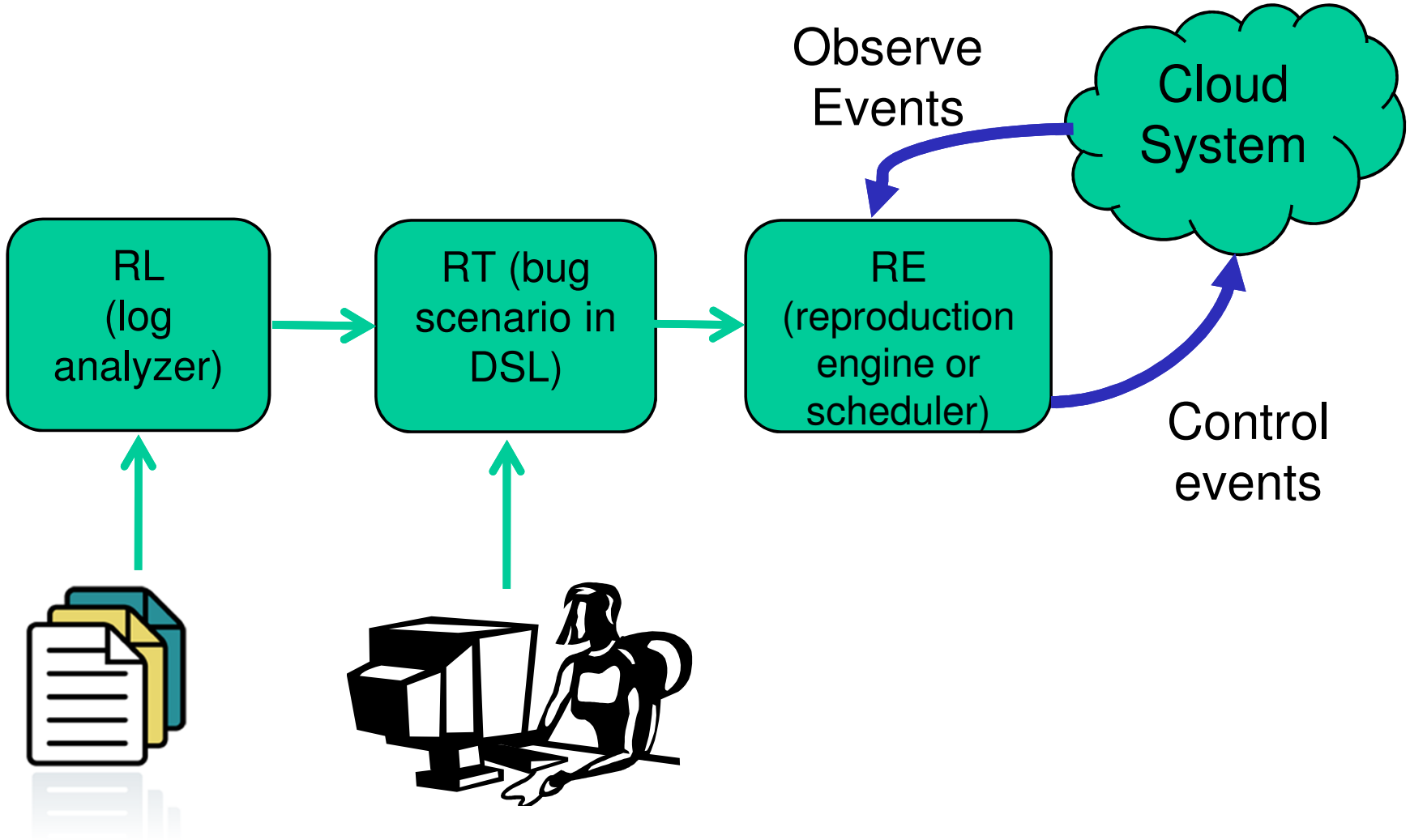
19

Motivation

Cloud Systems



ReproLite's Approach:
3 Main Elements (RL, RT, RE)



Reproduce what bugs? Why ReproLite?

- System-level bugs involving
  - Concurrency
  - System component interaction
  - Environment

Main new element:
DSL for scenarios + scheduler

- Benefits
  - RT (DSL)
    - Expressive for specifying bug scenarios in a deterministic manner
  - RL (log analyzer)
    - Provides assistance to understand the cause of a bug from logs
  - RE (scheduler)
    - Repeatedly reproduce
    - lightweight

Structure

RL: Parse, extract and diff buggy and non-buggy logs

Log level

Date & Time

Thread name

DEBUG [GC inspection] 2014-10-22 16:23:29,160 GCInspector.java (line 131) GC for ParNew: 5 ms, 266519024 reclaimed leaving 28339384 used; max is 1263271936

Line number

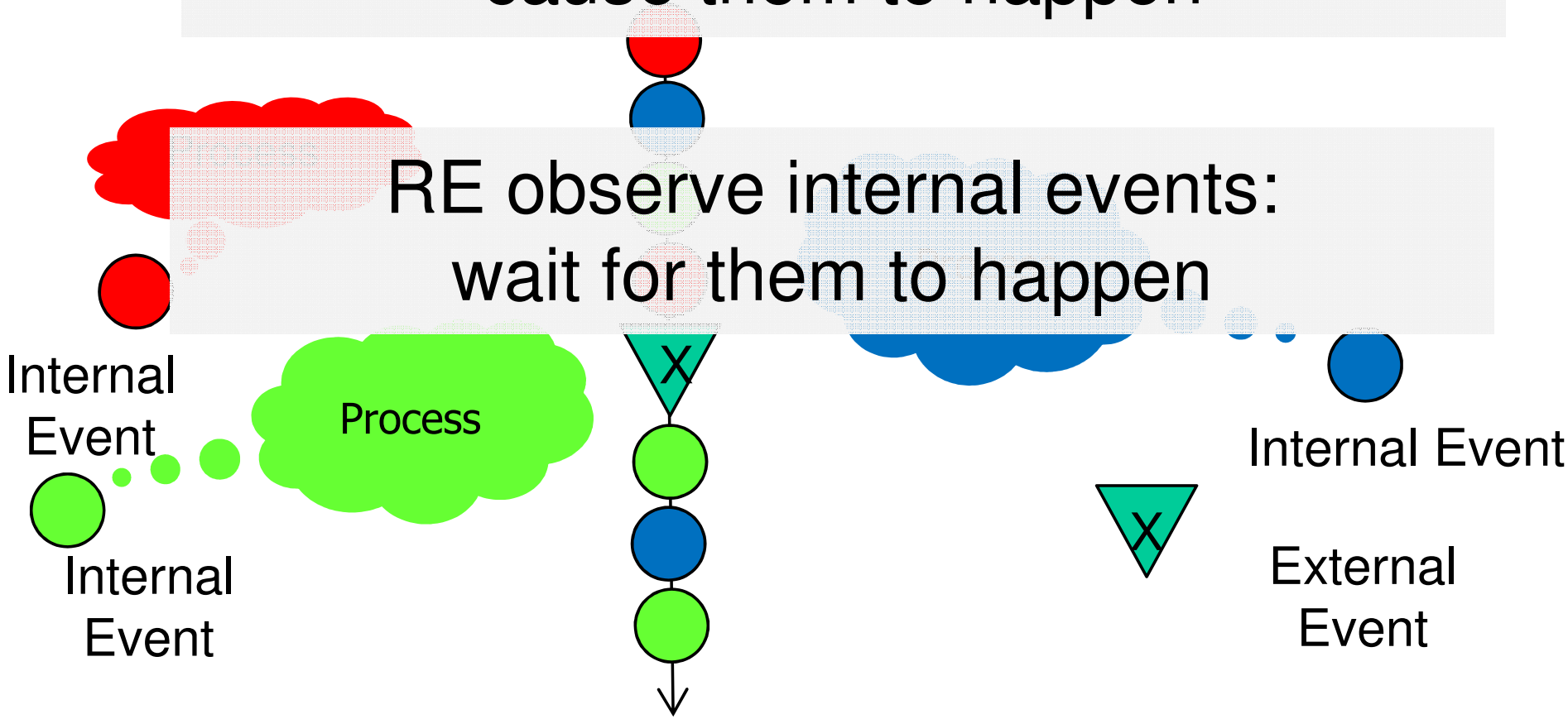
File name

Log message

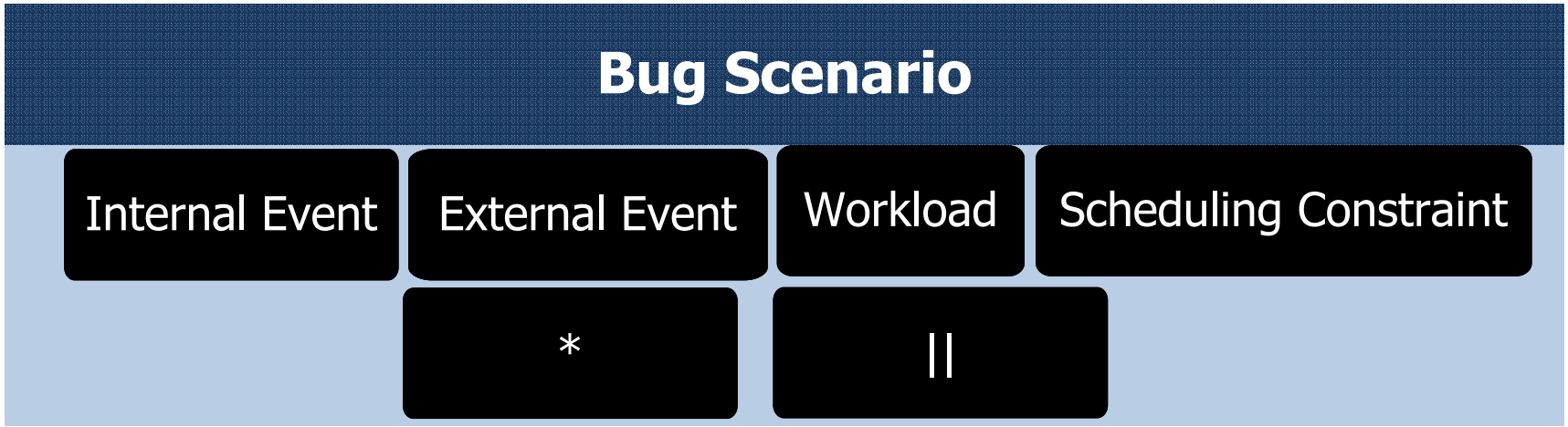
RE: Reproduction Engine

RE control external events:
cause them to happen

RE observe internal events:
wait for them to happen



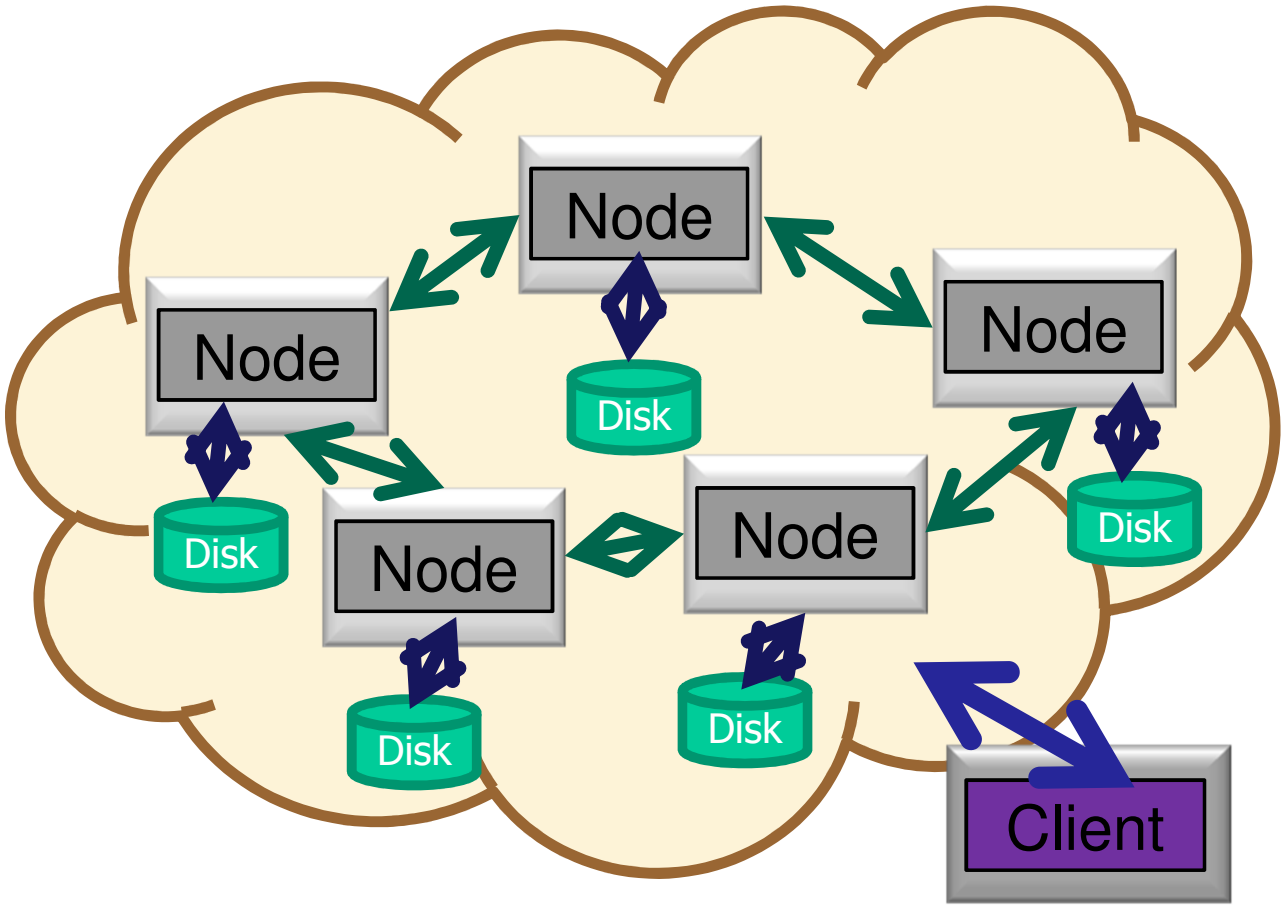
RT: DSL for expressing bug scenario



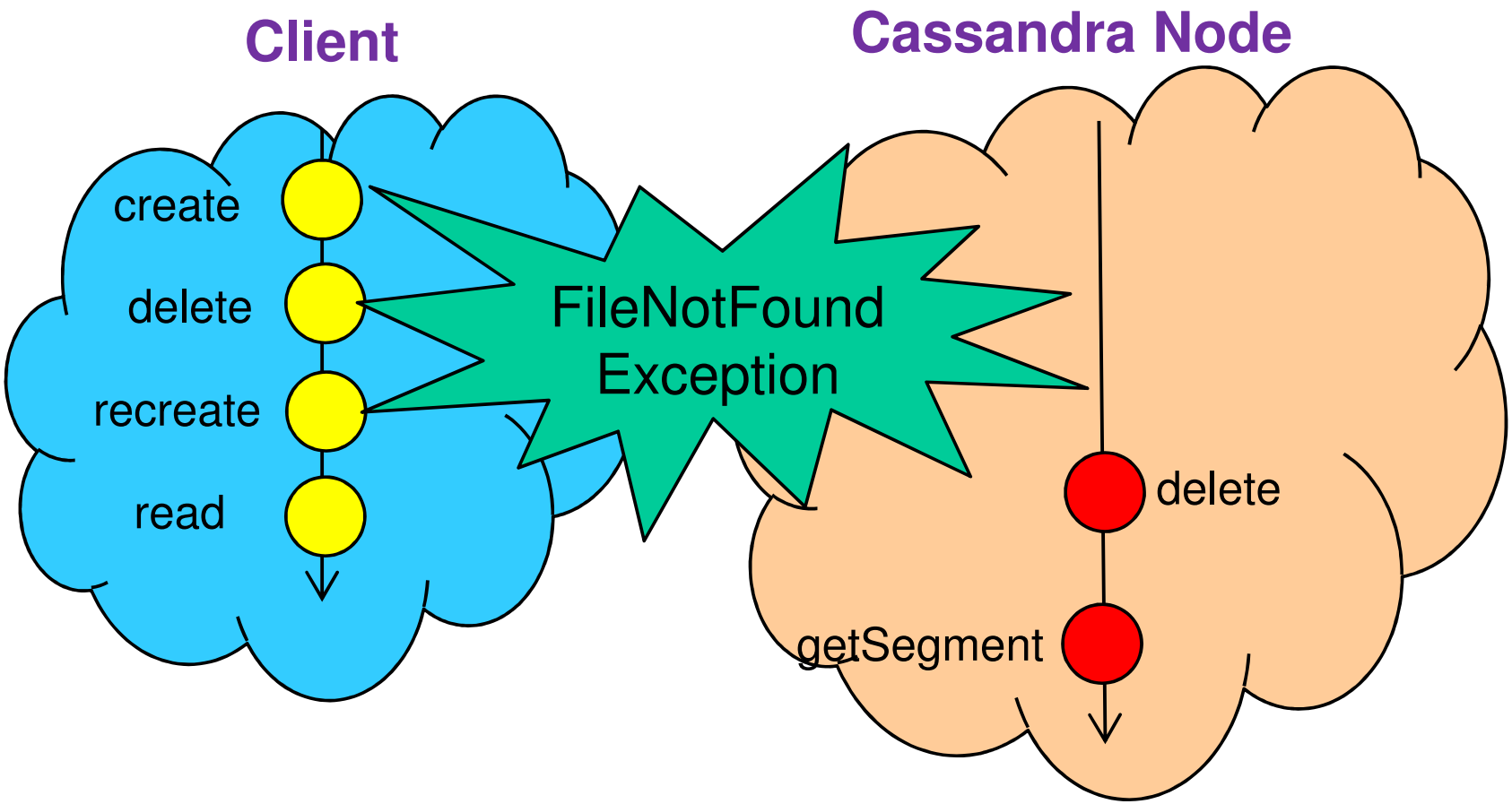
Run in sequence E1 * E2

Left and right run in parallel E1 || E2

Cassandra



Cassandra bug 1477



DSL example

```
W1 = 'create-ks.sh test'
W2 = 'add-cf.sh test cf'
W3 = 'write.sh 20 test cf'
W4 = 'flush.sh'
W5 = 'drop.sh cf'
W6 = 'add-cf.sh test cf'
W7 = 'read.sh test cf'
```

A workload given by bash script names and arguments

Stack traces to identify an internal event

```
E1 [ stack ]= 'File.delete`
E2 [ stack ]= 'BufferedSegmentedFile.getSegment'
```

Bug scenario

```
W1 * W2 * W3 * W4 * ( W5 * W6 * W7 || E1 * E2 )
```

Example Walkthrough

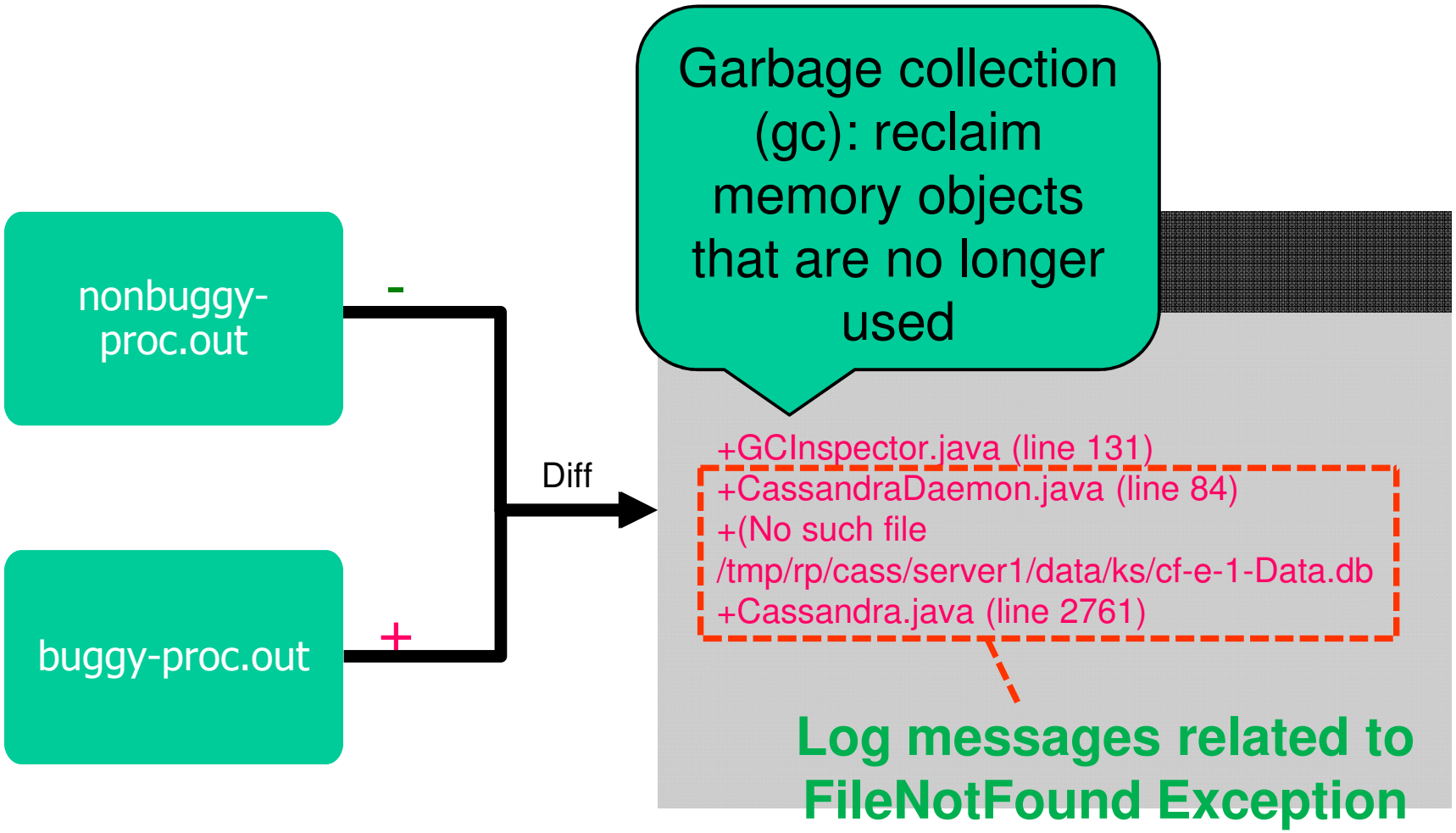
Users write code in DSL: First attempt

W1 = 'create-ks.sh test'
W2 = 'add-cf.sh test cf'
W3 = 'write.sh 20 test cf'
W4 = 'flush.sh'
W5 = 'drop.sh cf'
W6 = 'add-cf.sh test cf'
W7 = 'read.sh test cf'

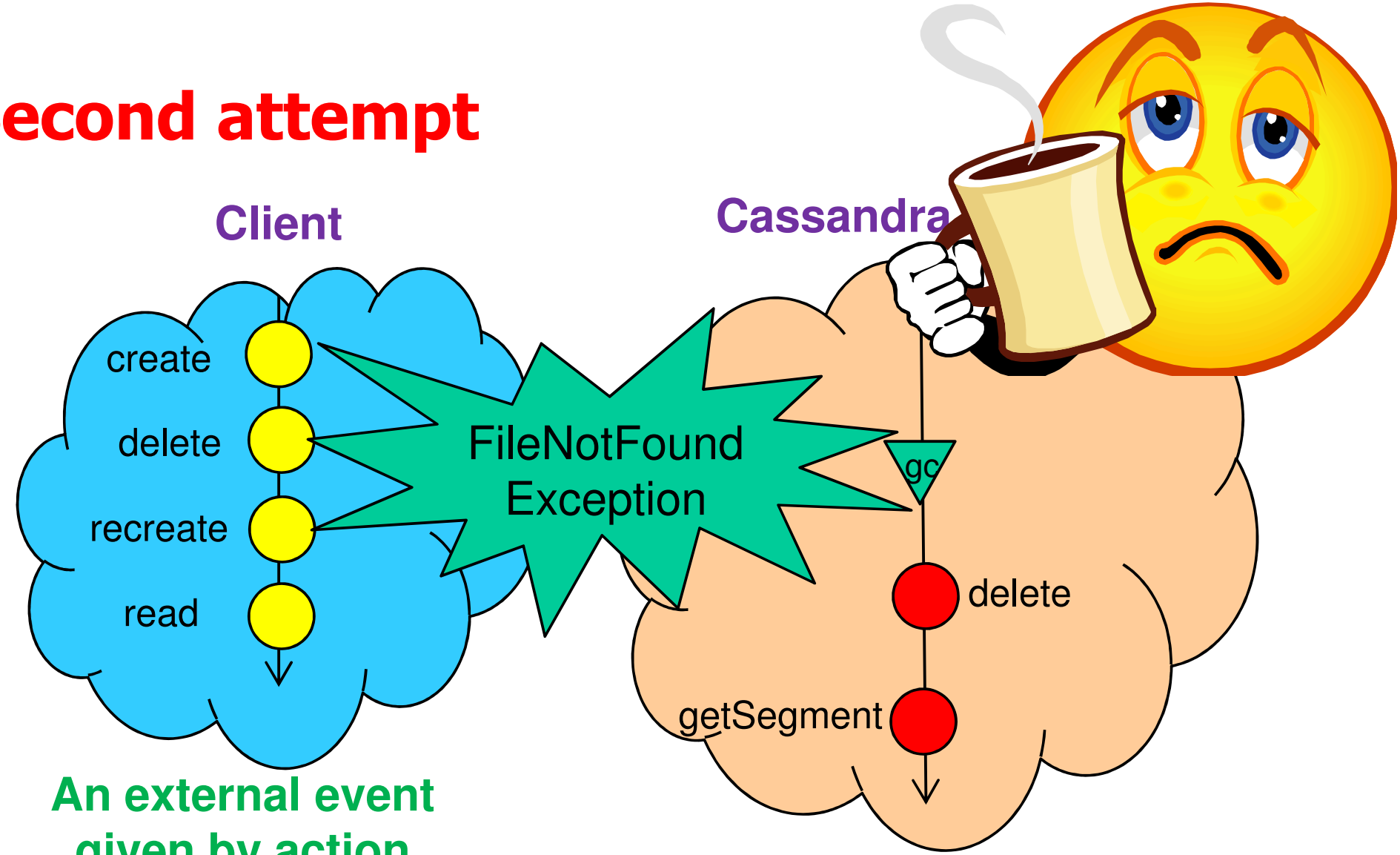
E1 [stack]= 'File.delete`
E2 [stack]= 'BufferedSegmentedFile.getSegment'

W1 * W2 * W3 * W4 * (W5 * W6 * W7 || E1 * E2)

Extract file name and line number from each log message and diff



Second attempt



An external event given by action and node ID

X1 = garbage collection 'cnode'

W1 * W2 * W3 * W4 * (W5 * W6 * W7 || **X1** * E1 * E2)

Rely on logs and source code whenever possible

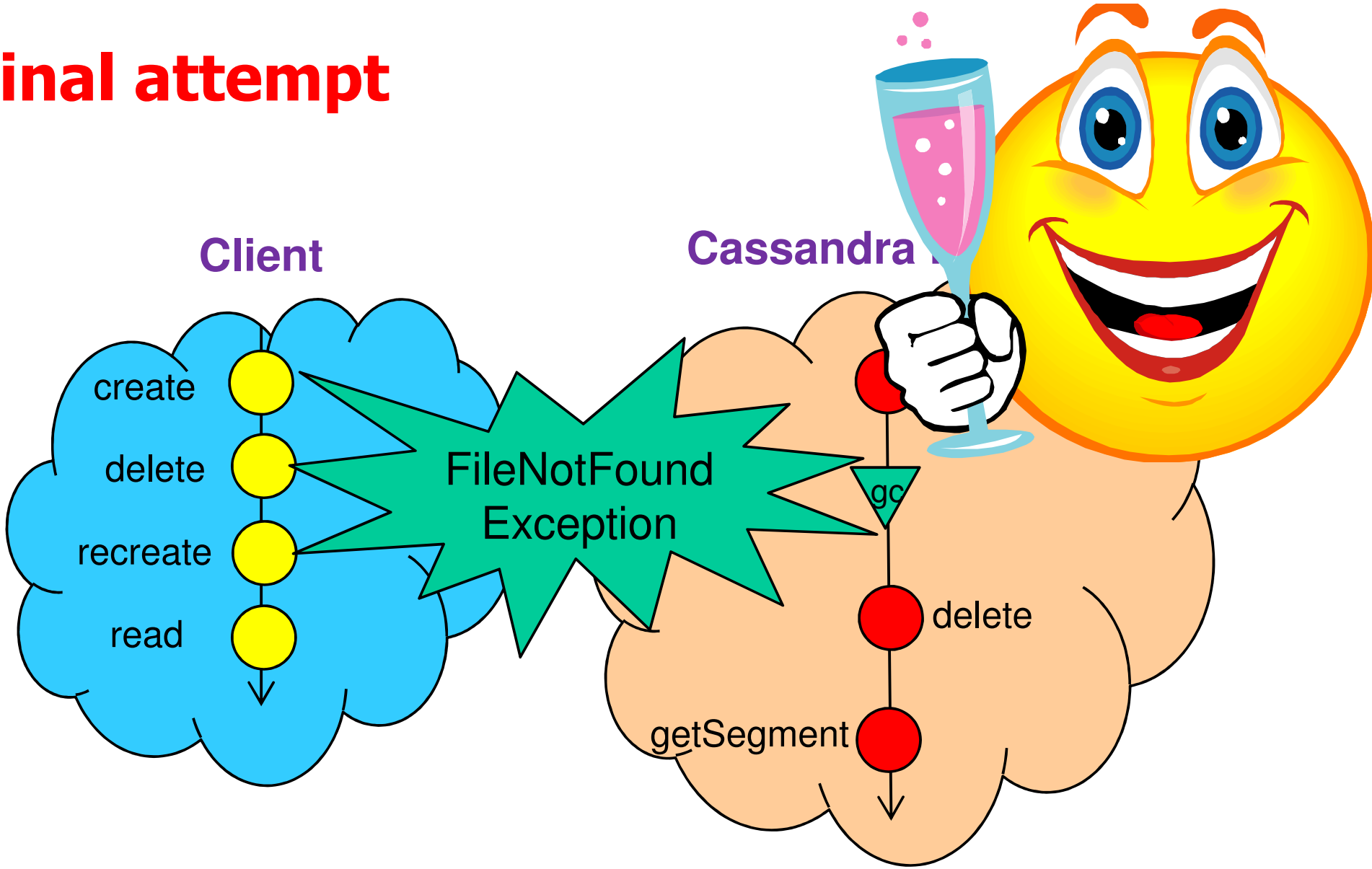
buggy-proc.out

```
SSTableReader.java (line 471)
GCInspector.java (line 131)
SSTableTracker.java (line 113)
Migration.java (line 120)
ColumnFamilyStore.java (line 418)
```

SSTableReader.java

```
471             logger.debug("Marking " + getFil
472         try
473         {
474             if (!new File(desc.filenameFor(C
475                 throw new IOException("Unabl
476         }
477         catch (IOException e)
478         {
479             throw new IOError(e);
480         }
481         phantomReference.deleteOnCleanup();
```

Final attempt

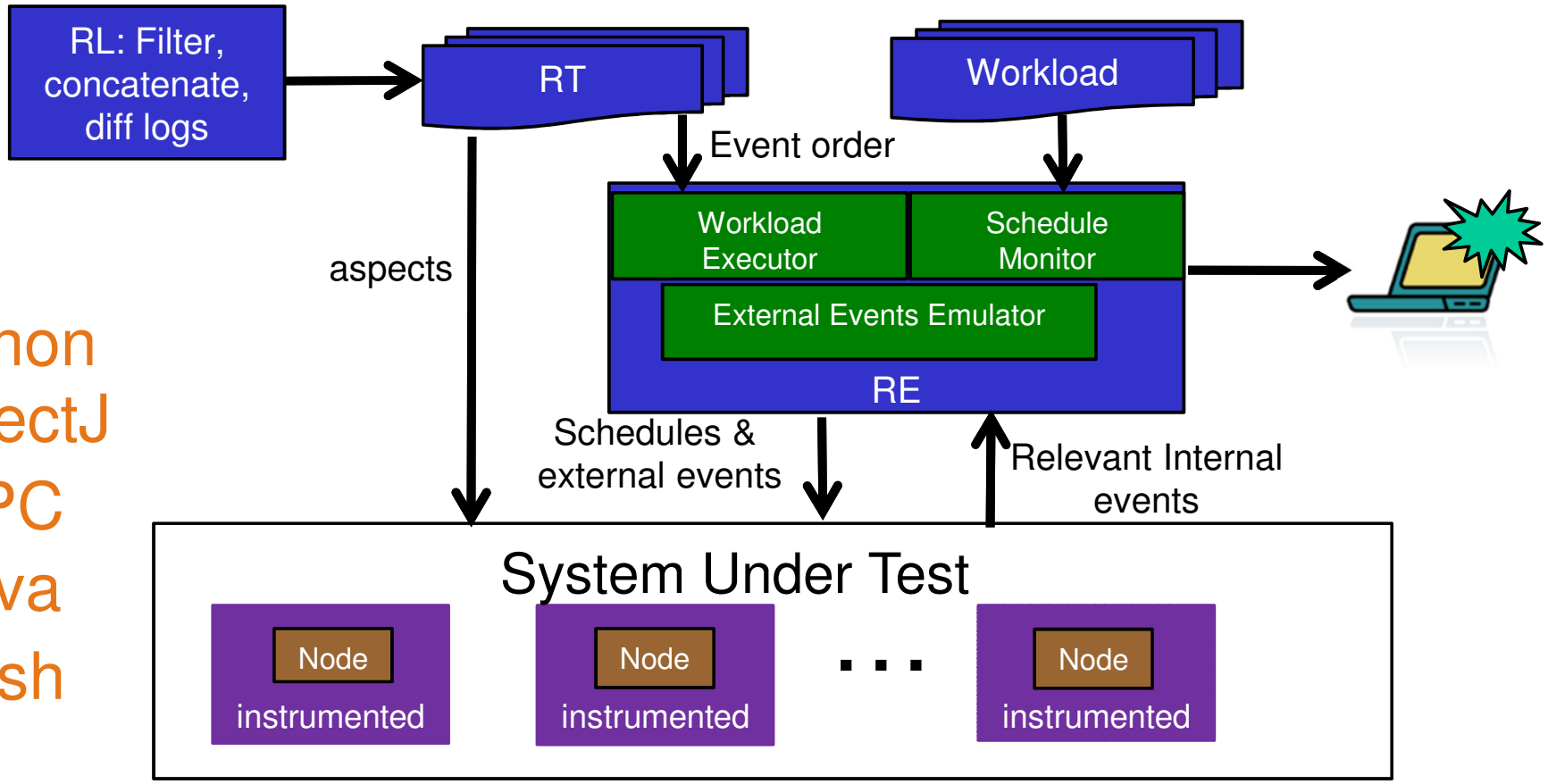


E3 [stack]= 'SSTableDeletingReference.deleteOnCleanup'

W1 * W2 * W3 * W4 * (W5 * W6 * W7 || **E3** * X1 * E1 * E2)

How? (Implementation)

Implementation



Python
AspectJ
RPC
Java
Bash

So What? (Experiments)

Experiments

We experimented with using ReproLite to debug 11 bugs from

Cassandra HBase

Complexity	<=6 events involved in each bug
Performance Overhead	0~132, close to 0 for half of the bugs
Usefulness of Logs	70% percent of bug causes hidden in logs.

Conclusion

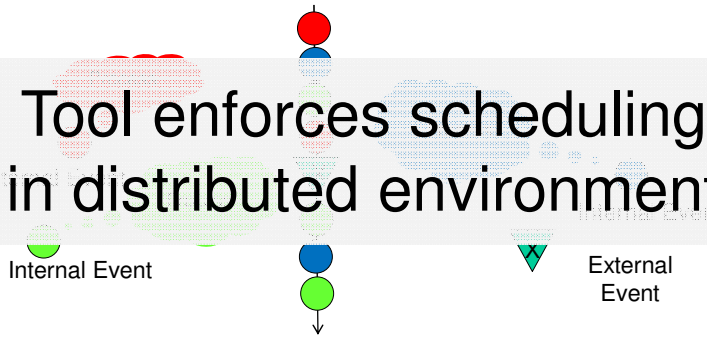
Users write code in DSL: Final attempt

```
W1 = 'create-ks.sh test'
W2 = 'add-cf.sh test cf'
W3 = 'write.sh 20 test cf'
E2 [ stack ]= 'BufferedSegmentedFile.getSegment'
E3 [ stack ]= 'SSTableDeletingReference.deleteOnCleanup'

W1 * W2 * W3 * W4 * ( W5 * W6 * W7 || E3 * X1 * E1 * E2 )
```

High-level language specifies bug scenario

RE: Reproduction Engine



Tool enforces scheduling in distributed environment

8

Experiments

We experimented with using ReproLite to debug 11 bugs from

Illustrated benefits with 11 hard system bugs

Usefulness of Logs — 70% percent of bug causes hidden in logs.

18

Thanks