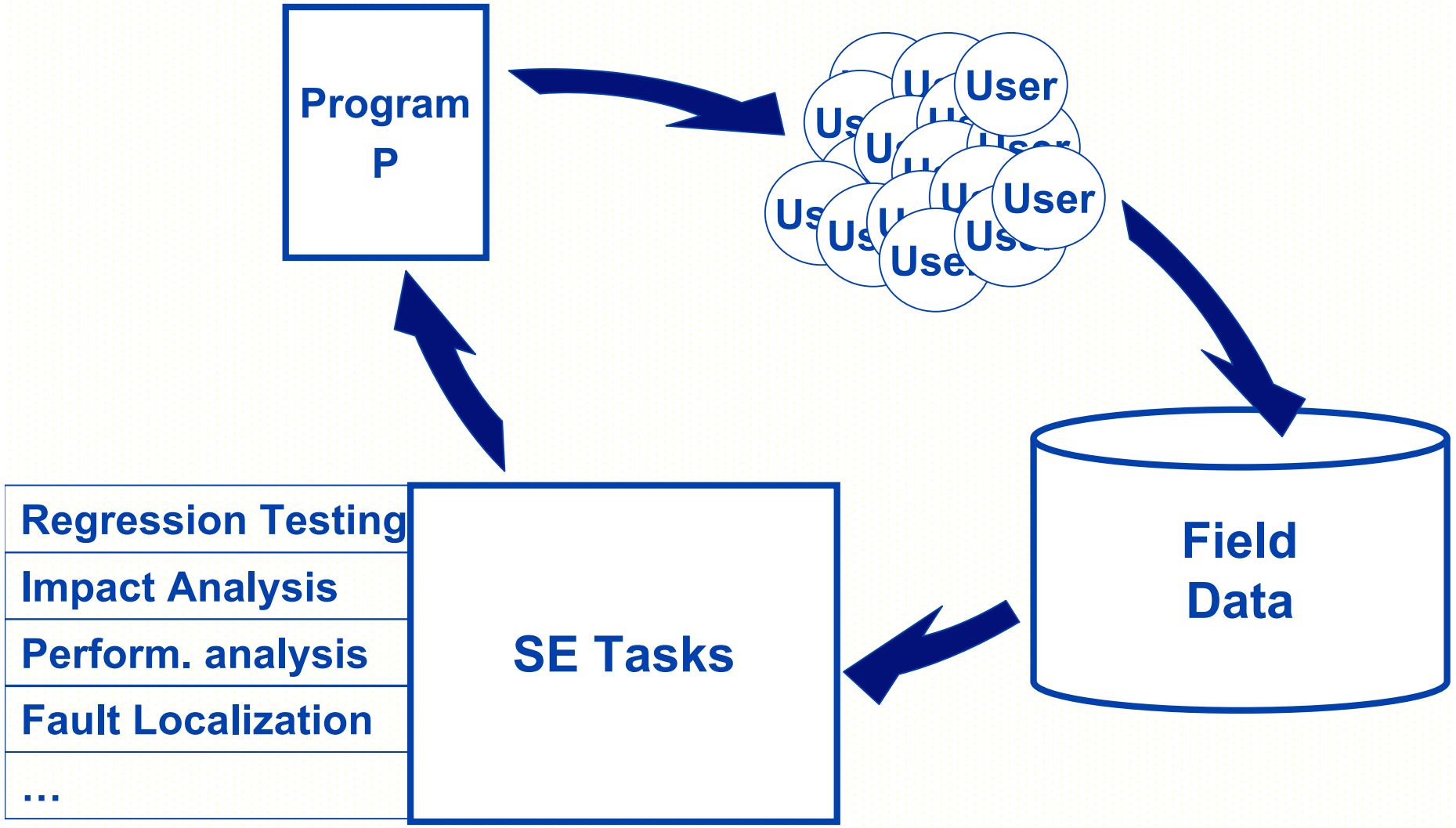
Selective Capture and Replay of Program Executions

Alex Orso
Bryan Kennedy
Georgia Institute of Technology
orso@cc.gatech.edu | bck@acm.org

This work was supported in part by NSF awards CCR-0205422, CCR-0306372, and CCR-0209322 to Georgia Tech.

SPARC

GAMMA: Overall Picture



Replaying Executions: Issues

Practicality

- High volume of data
- Hard to capture (custom)
- Rich environment

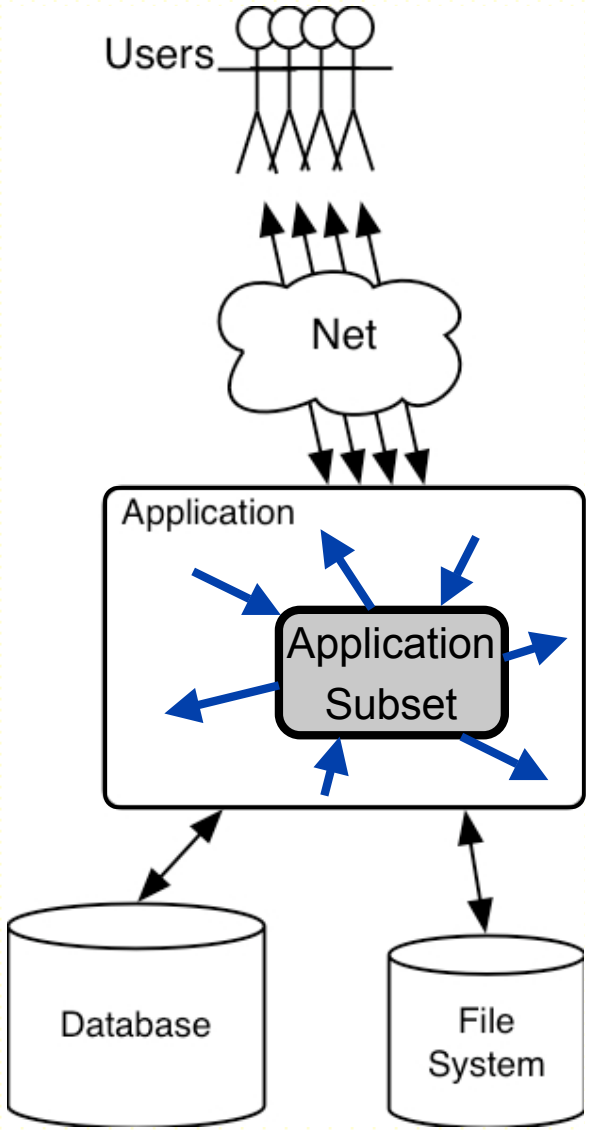
Privacy

- Sensitive information

Safety

- Side-effects

→ Partial replay



Replaying Executions: Applications

- Generation of test cases from users' executions
- Generation of subsystem/unit test cases from system test cases
- Off-line dynamic analysis
- Debugging
- …

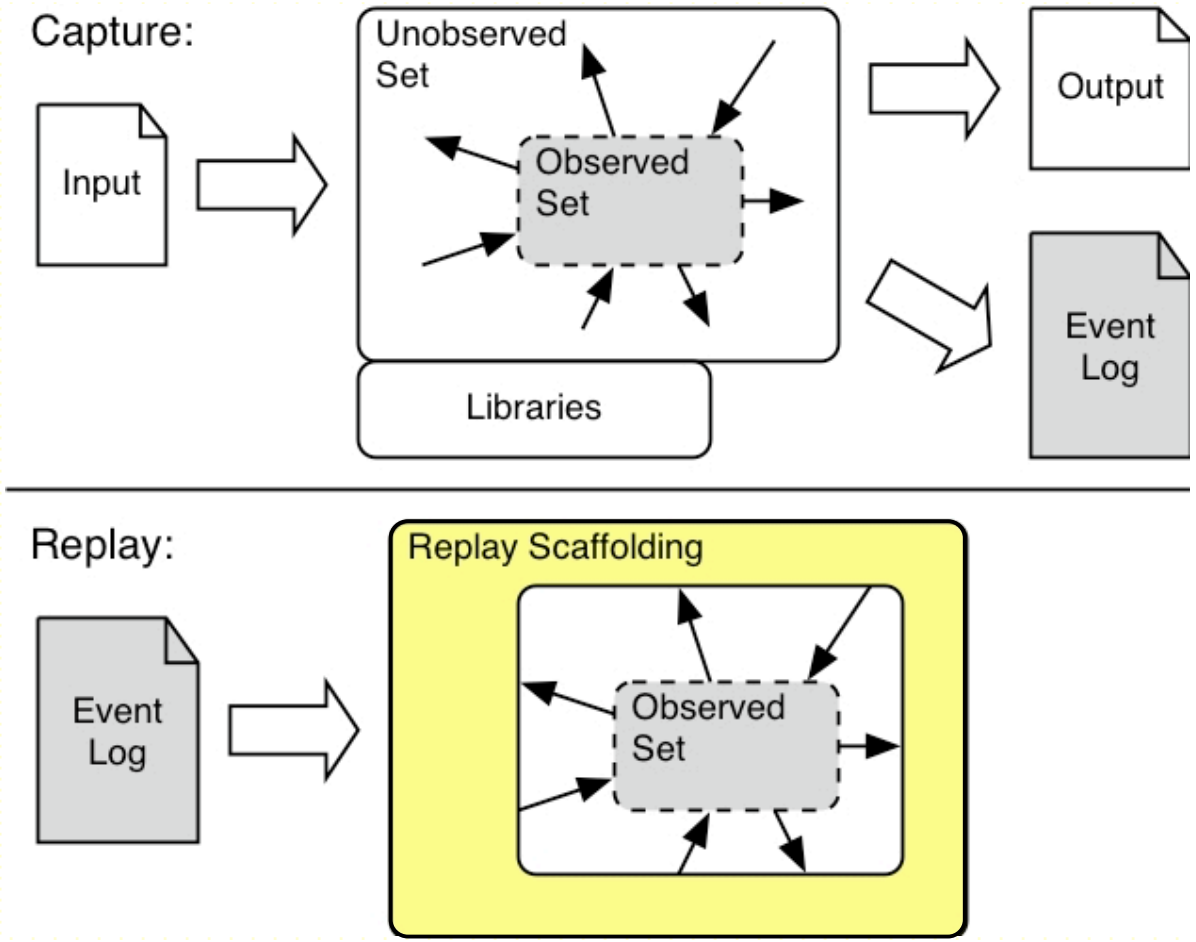
Outline

- Motivation
- Our approach
- Implementation and Evaluation
- Conclusions and Future Work

Outline

- Motivation
- Our approach
- Implementation and Evaluation
- Conclusions and Future Work

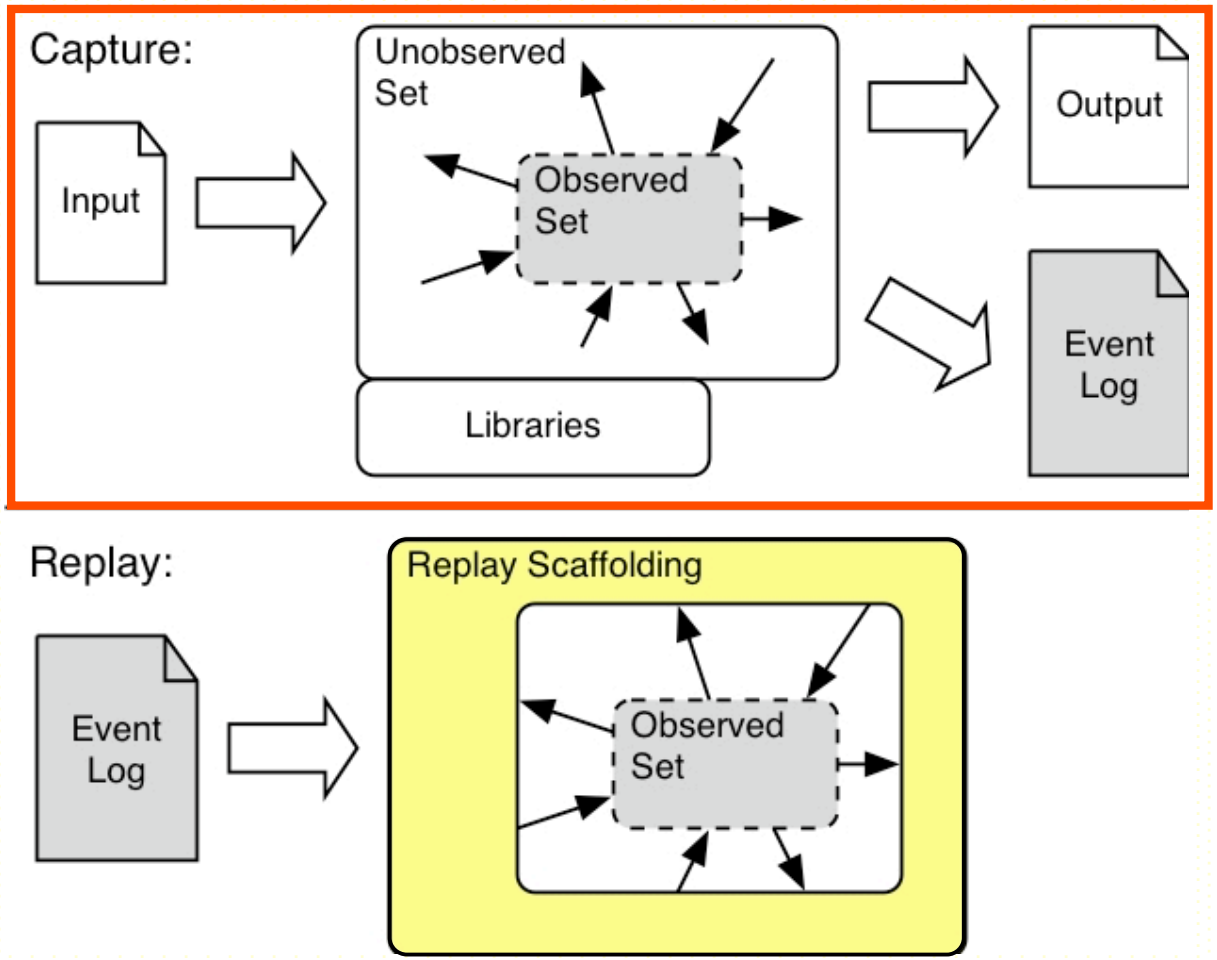
Overview


Capture:
Input
Unobserved Set
Observed Set
Libraries
Output
Event Log
Replay:
Event Log
Replay Scaffolding
Observed Set

Overview: Capture

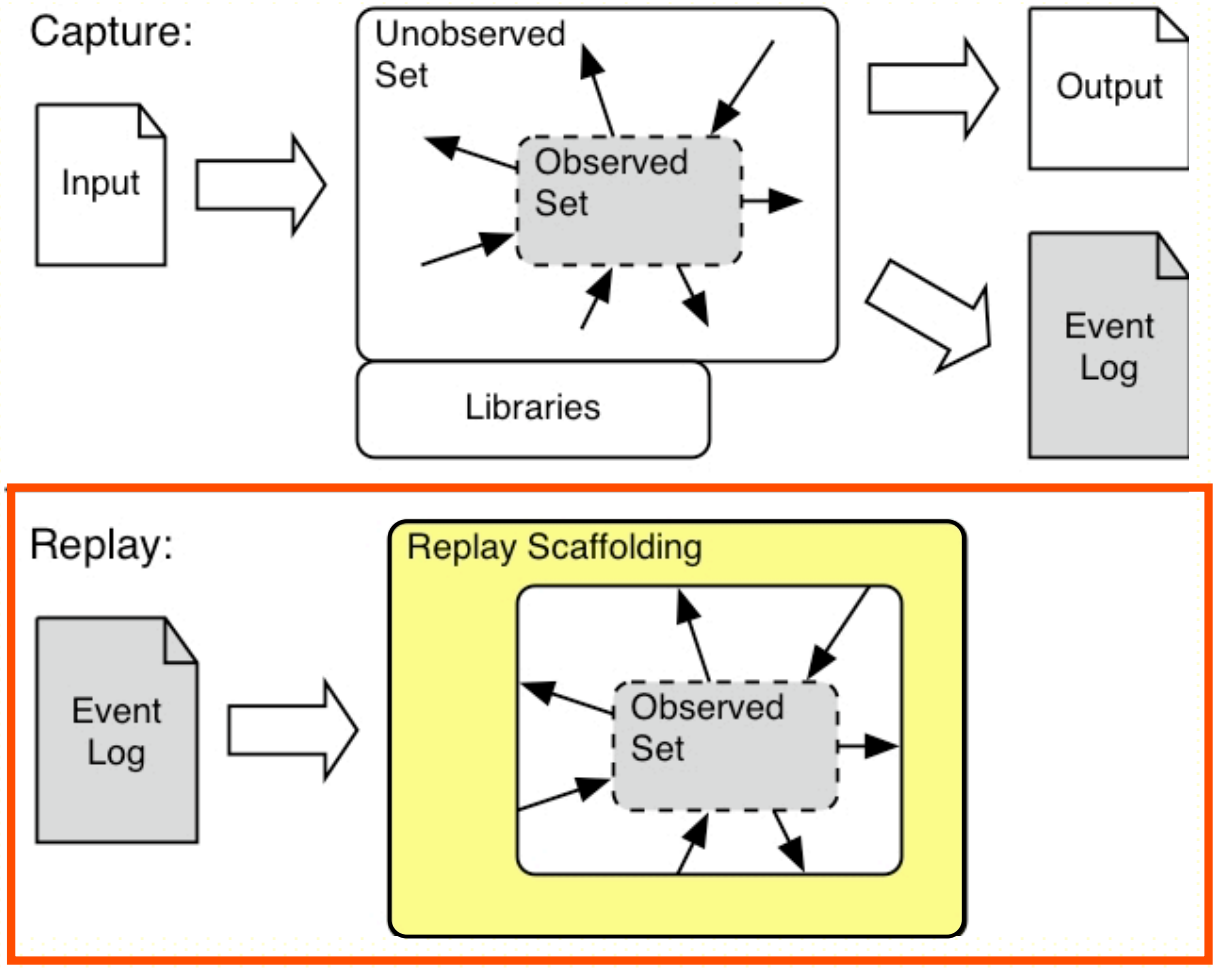
- Input *observed set*
- Identify observed-set's boundaries
- Collect interactions and data across boundaries

  => *event log*



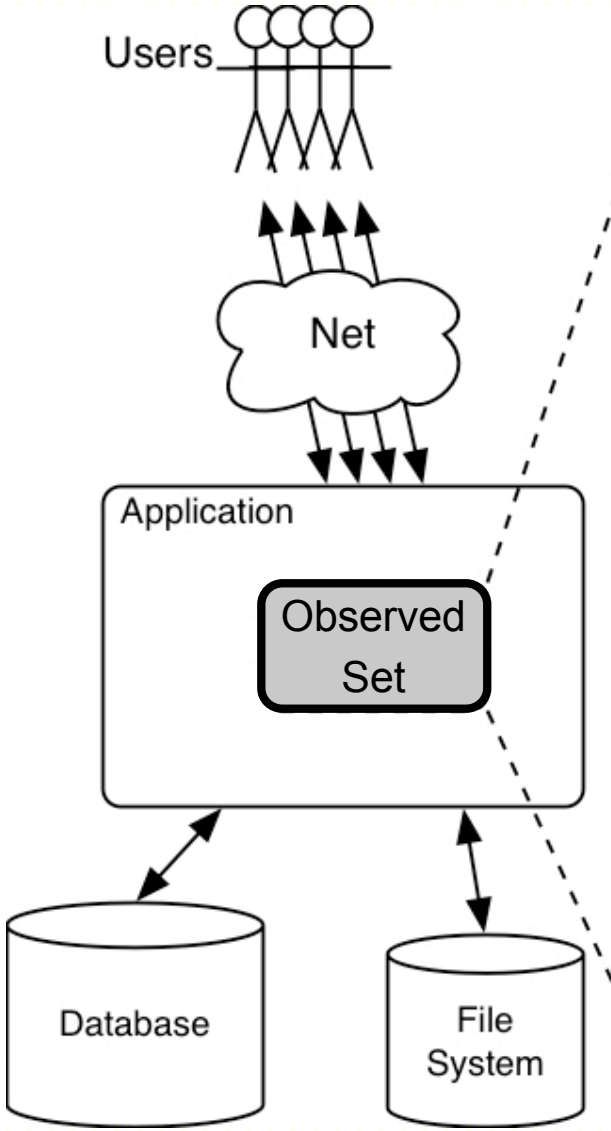
Overview: Replay

- Provide *replay scaffolding*
- Process *event log*
  - Create classes
  - Replay interactions



Characteristics of the Approach

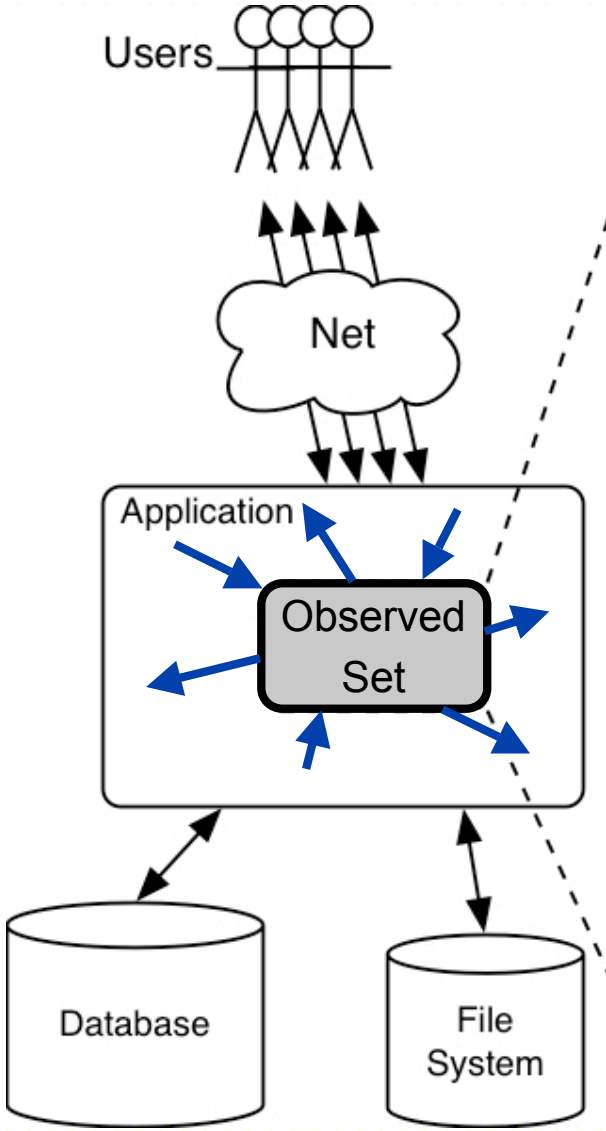
- Selective




```
class DB {...}
class Node {...}
class Compute {
  int norm = 0;
  DB db;
  ...
  void setup(int x) {
    ...
    int y = db.getSomeInt();
    norm = x - y;
    ...
  }
  ...
  double getRatio(HugeTree ht) {
    Iterator it = ht.iterator();
    while (it.hasNext()) {
      Node n = (Node)it.next();
      double res = n.val;
      if (res > 0)
        return res / norm;
    }
    return 0.0;
  }
}
```


Characteristics of the Approach

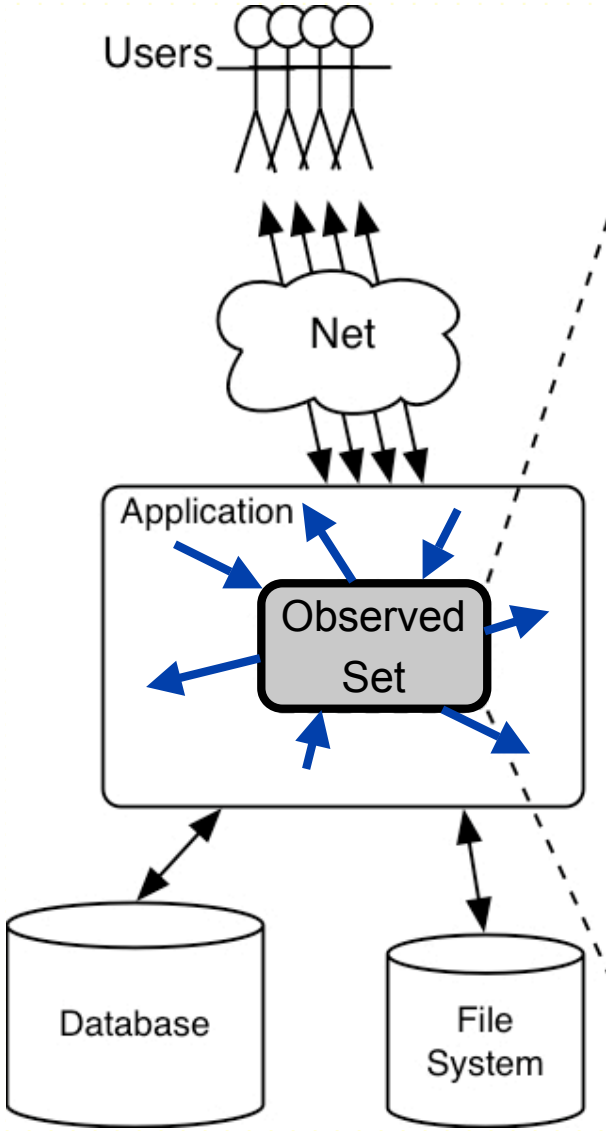
- Selective
- Event based




```
class DB {...}
class Node {...}
class Compute {
  int norm = 0;
  DB db;
  ...
  void setup(int x) {
    ...
    int y = db.getSomeInt();
    norm = x - y;
    ...
  }
  ...
  double getRatio(HugeTree ht) {
    Iterator it = ht.iterator();
    while (it.hasNext()) {
      Node n = (Node)it.next();
      double res = n.val;
      if (res > 0)
        return res / norm;
    }
    return 0.0;
  }
}
```

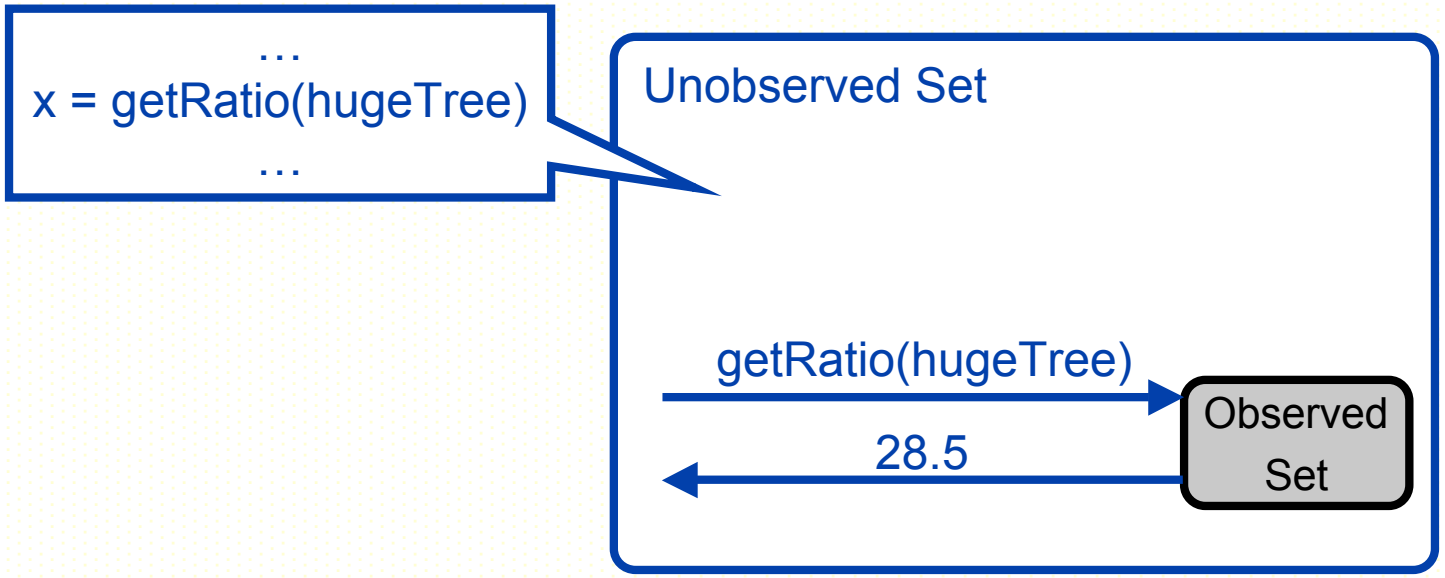

Characteristics of the Approach

- Selective
- Event based
- Efficient (partial data)

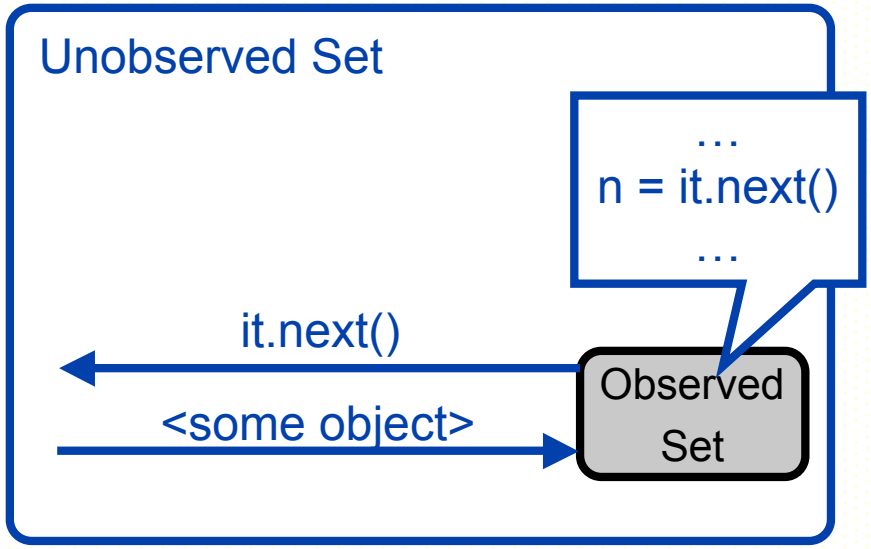



```
class DB {...}
class Node {...}
class Compute {
  int norm = 0;
  DB db;
  ...
  void setup(int x) {
    ...
    int y = db.getSomeInt();
    norm = x - y;
    ...
  }
  ...
  double getRatio(HugeTree ht) {
    Iterator it = ht.iterator();
    while (it.hasNext()) {
      Node n = (Node)it.next();
      double res = n.val;
      if (res > 0)
        return res / norm;
    }
    return 0.0;
  }
}
```


Capture Phase: Captured Events



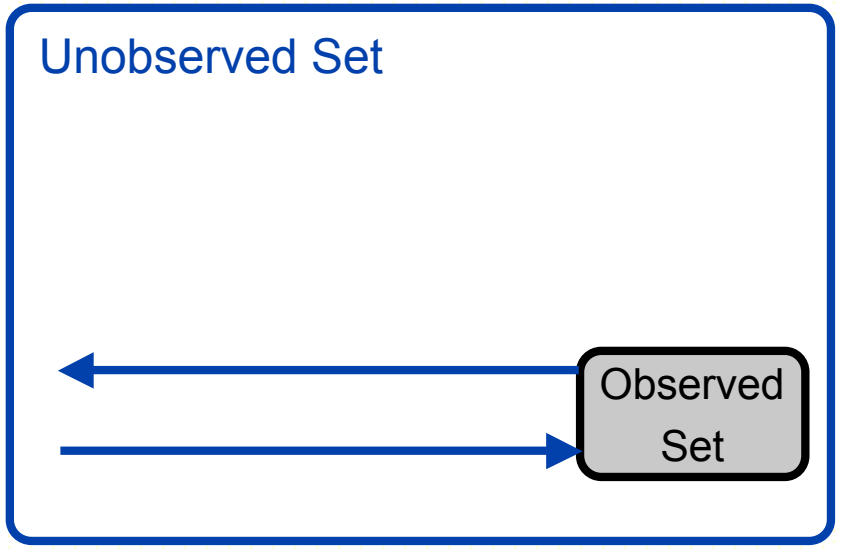
Capture Phase: Captured Events



Capture Phase: Captured Events

Method calls

- INCALL
- INCALLRET
- OUTCALL
- OUTCALLRET



Capture Phase: Captured Events

Method calls

- INCALL
- INCALLRET
- OUTCALL
- OUTCALLRET

Field Access

- INWRITE
- OUTWRITE
- OUTREAD

Exceptions

- EXCIN
- EXCOUT

INCALL / OUTCALL event

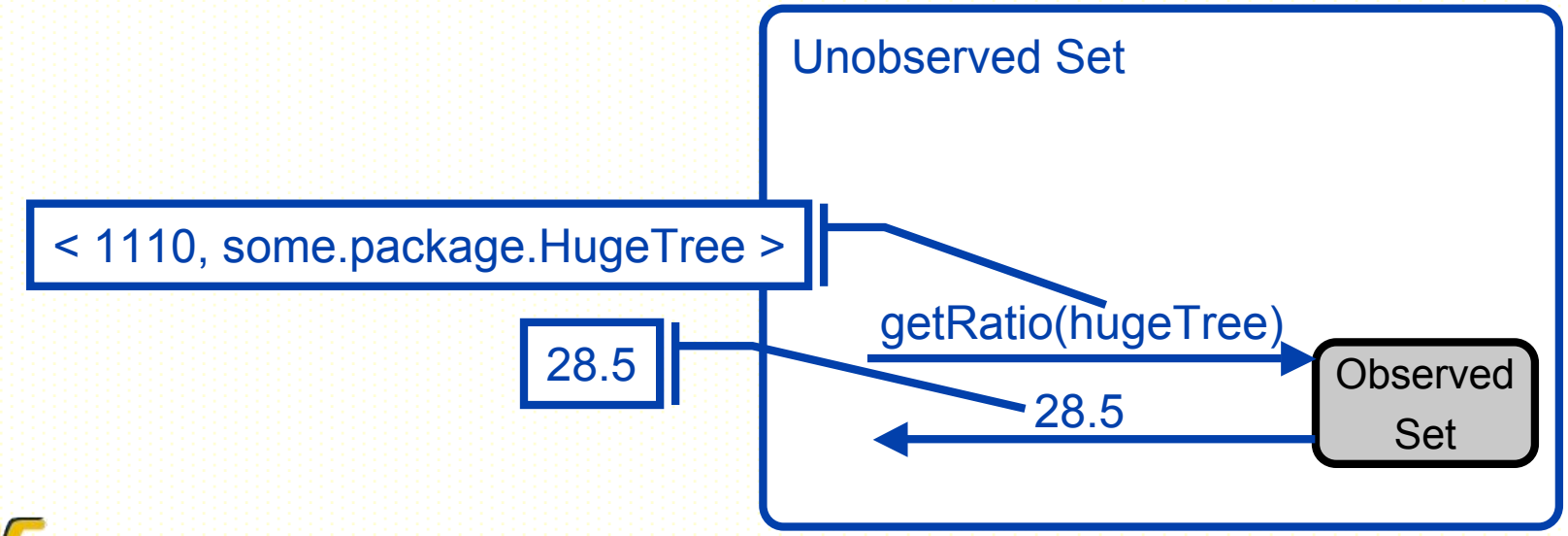
- callee's type
- callee's object ID
- callee signature
- parameter*

Capture Phase: Capturing Partial Data

- Capturing complete data is impractical (> 500% overhead in preliminary study)

⇒ only data that affect the computation

  - Scalar values
  - Object IDs and types



Mechanics

Capture/replay through instrumentation

- Probes
- Modifications of call sites and proxying
- Modification of field accesses
- Use of exception handling capabilities

Replaying Events

Technique acts as both driver and stub

- Produces events
  - INWRITE
  - INCALL, OUTCALLRET, and EXCIN (passing control to observed code)
- Consumes events
  - OUTCALL, INCALLRET, OUTWRITE, OUTREAD, EXCOUT
- Events from observed code are “optional”

Replaying Events

Technique acts as both driver and stub

- Produces events
  - INWRITE
  - INCALL, OUTCALLRET, and EXCIN (passing control to observed code)
- Consumes events
  - OUTCALL, INCALLRET, OUTWRITE, OUTREAD, EXCOUT
  - Checks for *out-of-sync* events
- Events from observed code are “optional”

Outline

- Motivation
- Our approach
- Implementation and Evaluation
- Conclusions and Future Work

Feasibility Study

Tool: SCARPE (Selective CApture and Replay of Program Executions)

- Two modalities: Online and offline instrumentation
- Uses BCEL

Subject: NanoXML

- 19 classes
- 3,300 LOCs,
- 216 test cases

Study setup: For each class in NanoXML

- Capture execution of the class for each test case
- Reply all such executions (> 4,000)

Results: all captures and replays were successful

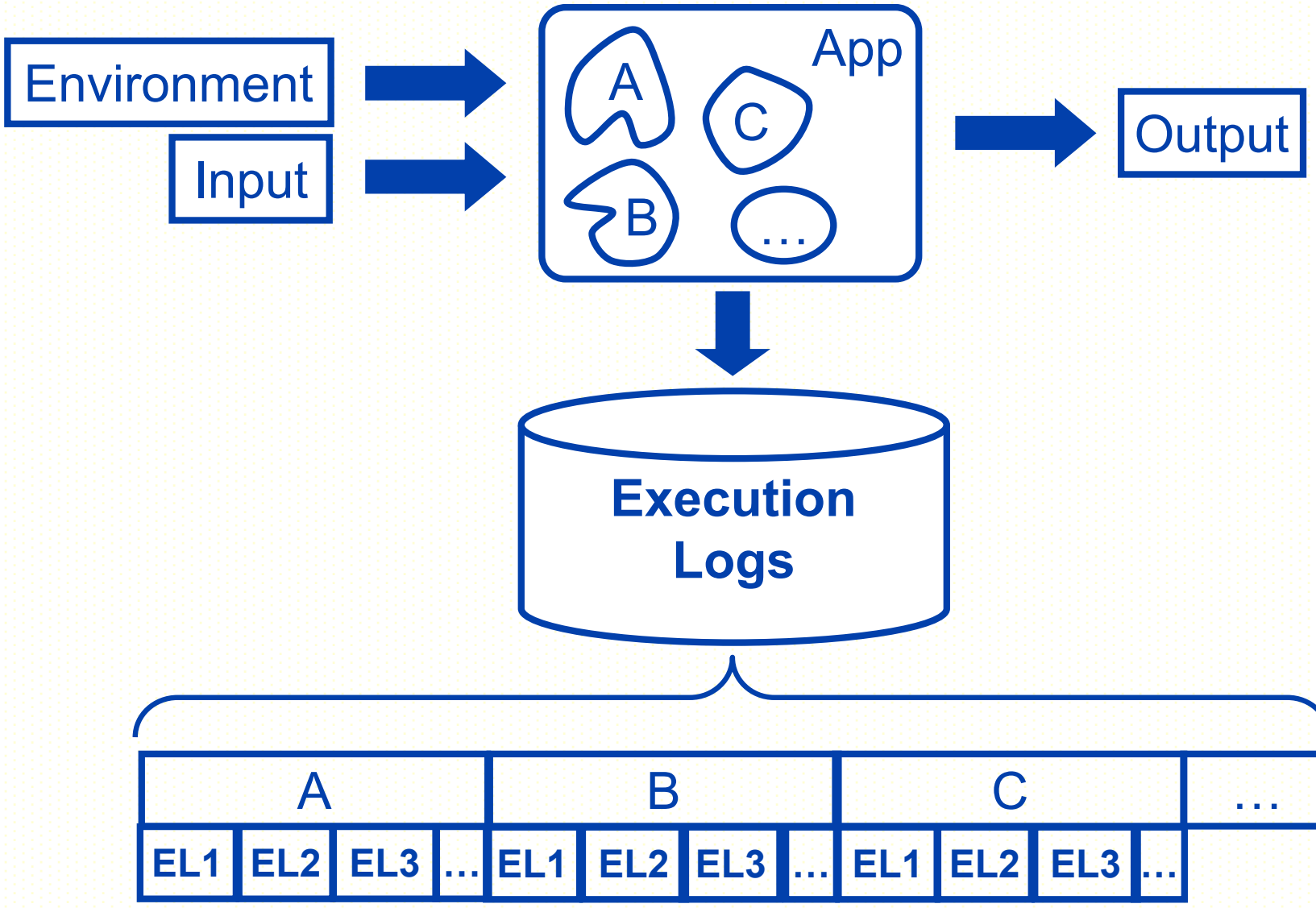
Outline

- Motivation
- Our approach
- Implementation and Evaluation
- Conclusions and Future Work

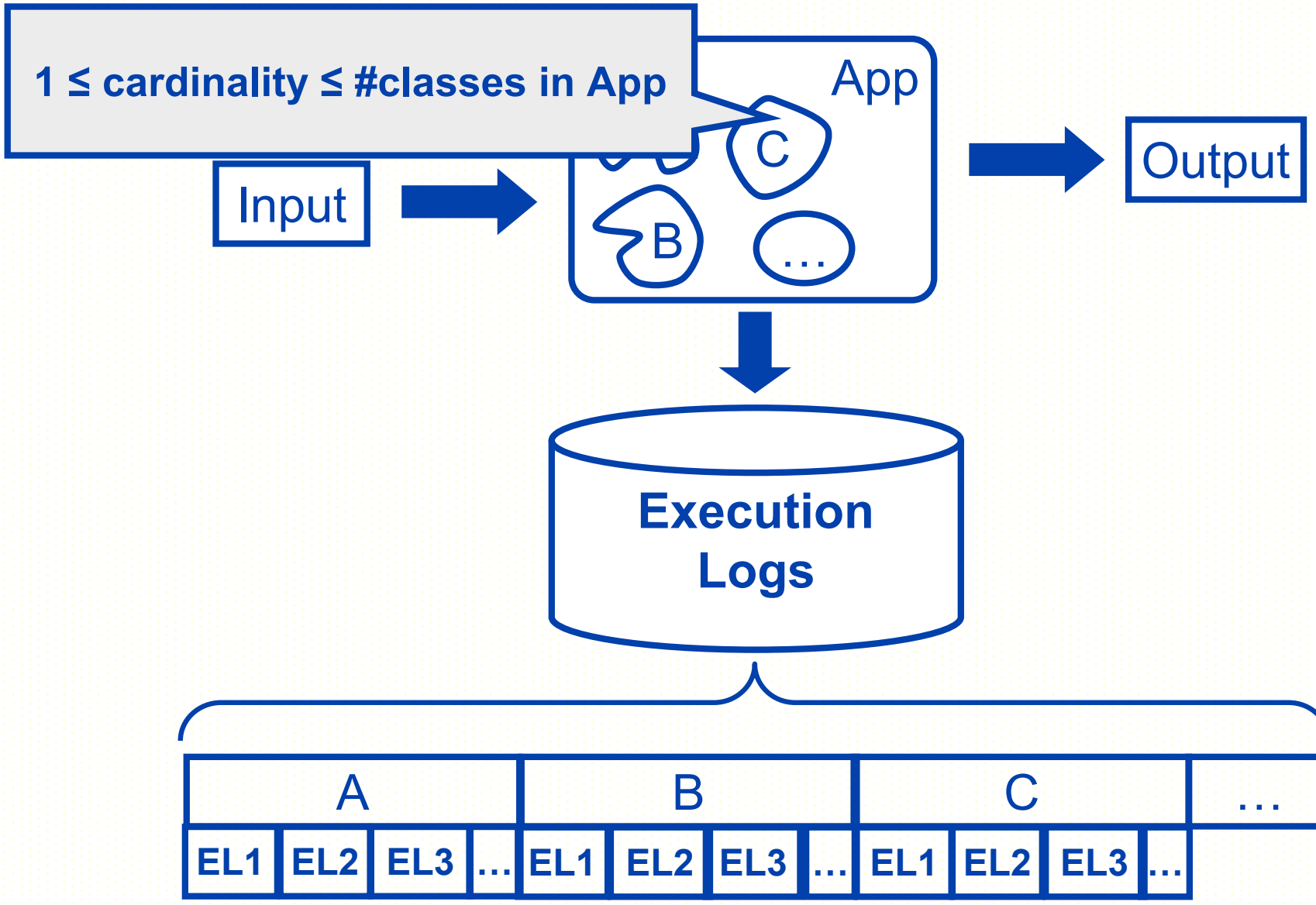
Related Work

- DejaVu [Choi98]
- jRapture [Steven00]
- Mock-object creation [Saff04]

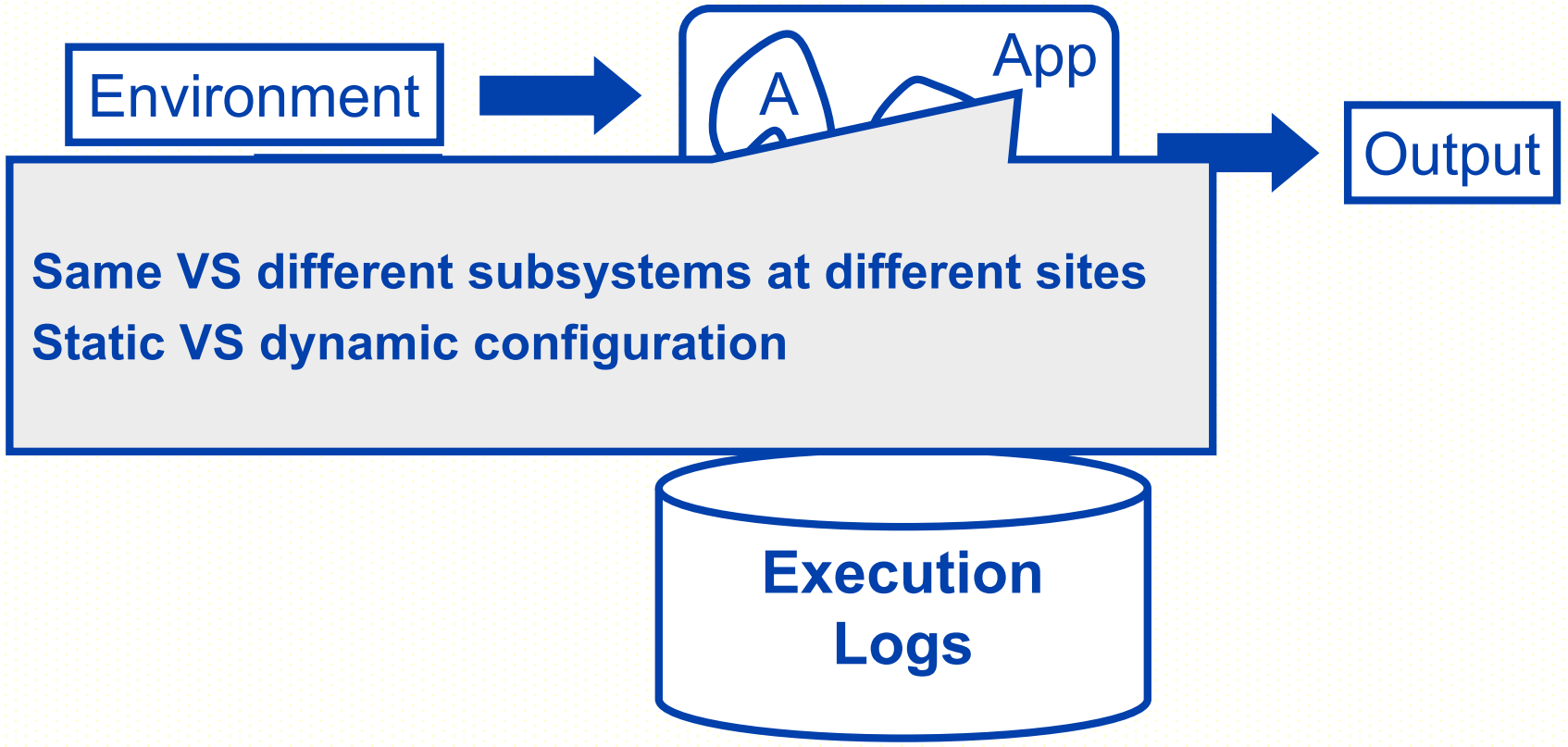
Summary

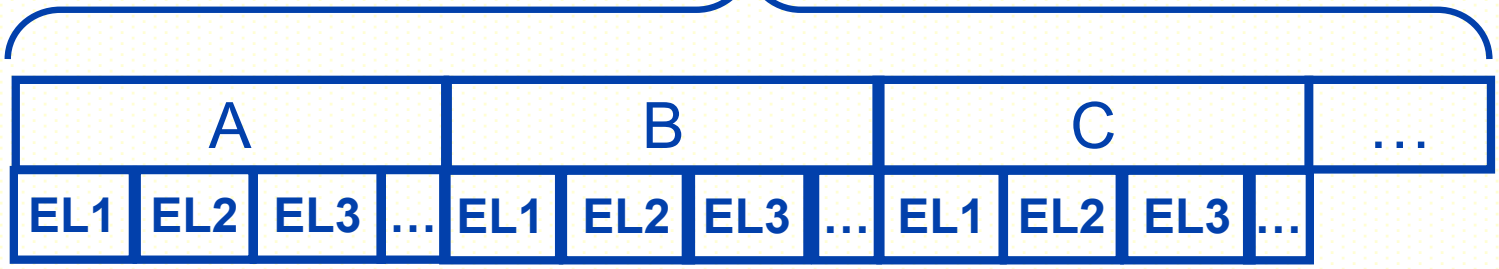

Environment
Input
App
A
C
B
…
Output
Execution Logs
A
B
C
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…

Summary

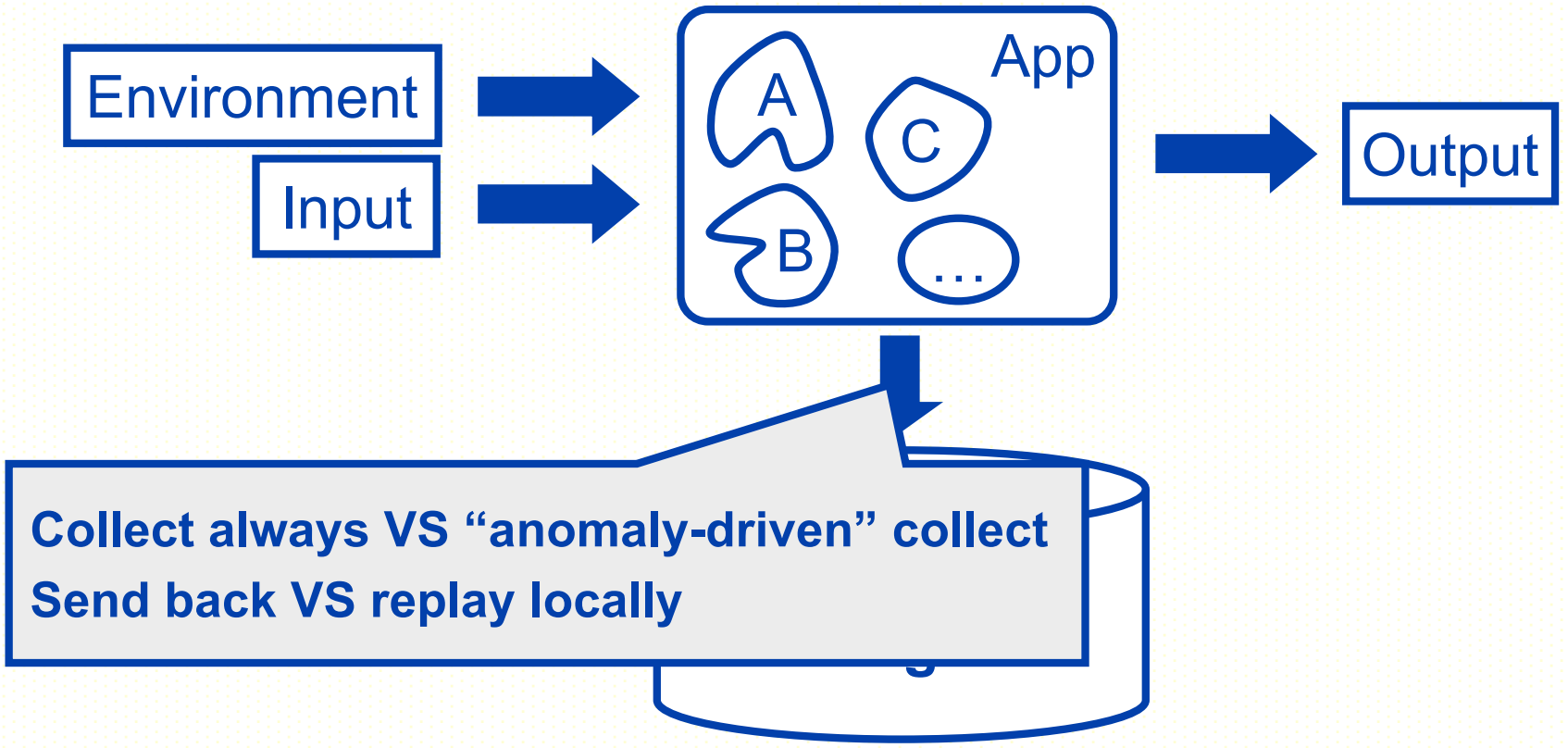

1 ≤ cardinality ≤ #classes in App
App
C
B
…
Input
Output
Execution Logs
A
B
C
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…

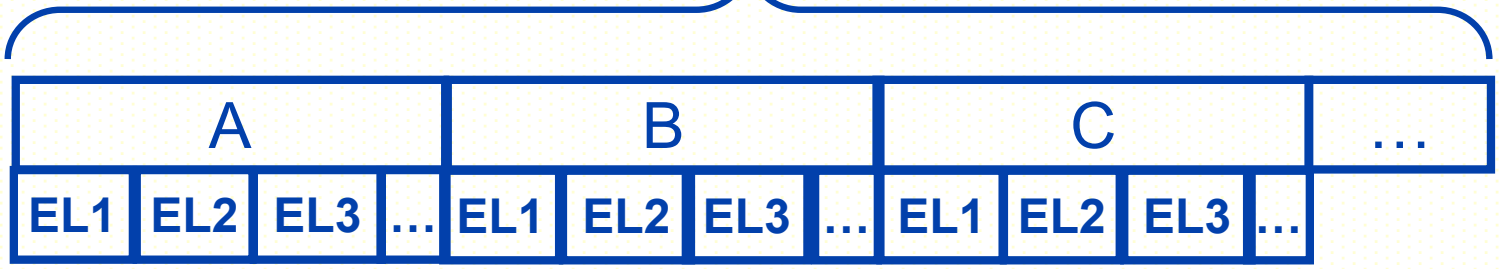
Summary


Environment
A
App
Output
Same VS different subsystems at different sites
Static VS dynamic configuration
Execution Logs

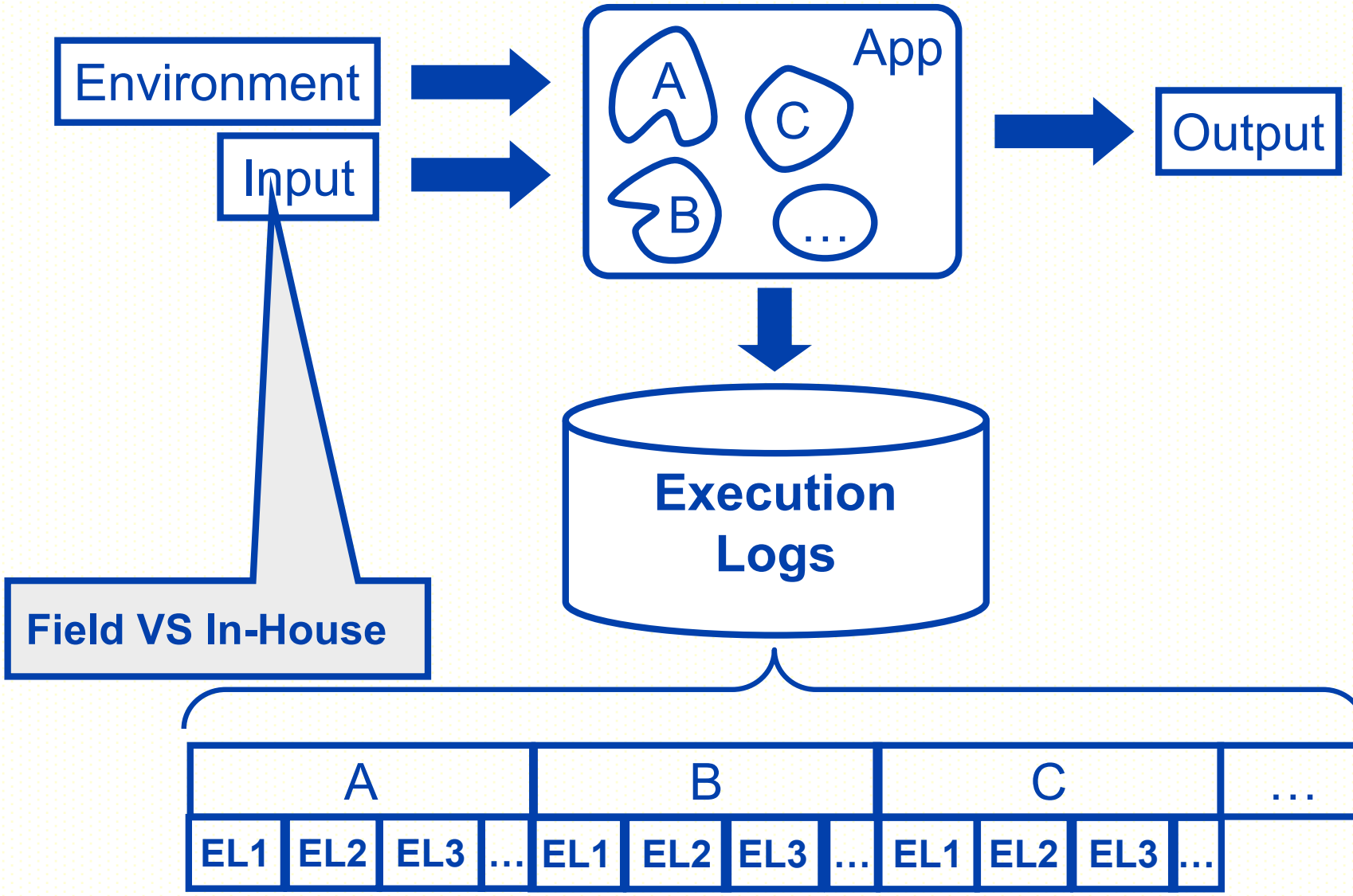

A
B
C
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…

Summary


Environment
Input
App
A
C
B
…
Output
Collect always VS “anomaly-driven” collect
Send back VS replay locally


A
B
C
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…

Summary


Environment
Input
App
A
C
B
…
Output
Execution Logs
Field VS In-House
A
B
C
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…
EL1
EL2
EL3
…

Future Work

- Further validation (especially w.r.t. performance)
- Post-mortem dynamic analysis of users' executions
  - Collection and replay in-house
  - Replay in the field
  - Conditional collection
- Regression testing
  - Automated generation of subsystem/unit test cases
  - Handling of out-of-sequence events
  - Possible extensions to the technique
- Debugging
- Static- and dynamic-analysis support for selection
- Application in other contexts (e.g., web services)
- Implementation at the JVM level

Questions?