SCARPE: A Technique and Tool for Selective Capture and Replay of Program Executions

Shrinivas Joshi

Advanced Micro Devices
Shrinivas.Joshi@amd.com

Alessandro Orso

Georgia Institute of Technology
orso@cc.gatech.edu

This work was supported in part by NSF awards
CCF-0541080 and CCR-0205422 to Georgia Tech.

Collecting Field Data

In house

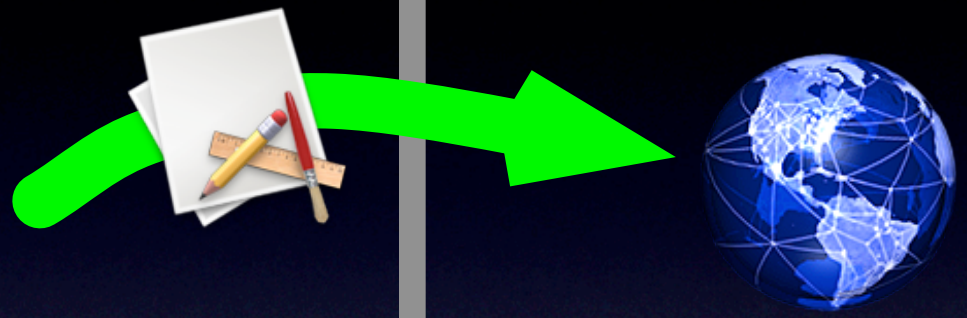
Developers

In the field

Collecting Field Data

In house

In the field

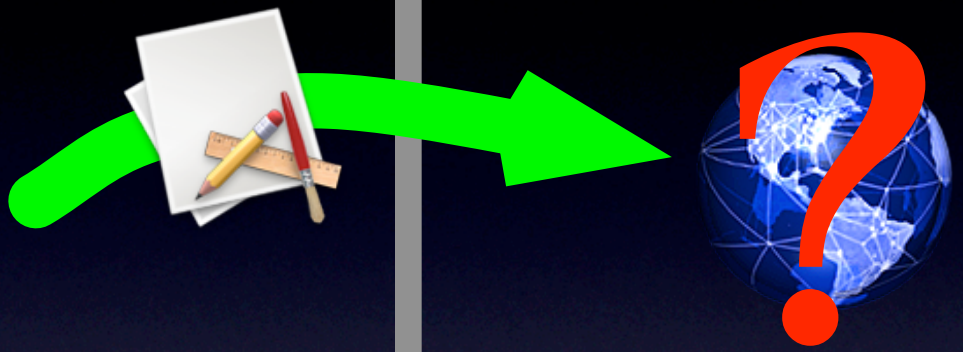


Developers

Collecting Field Data

In house

In the field



Developers

Collecting Field Data

In house

In the field

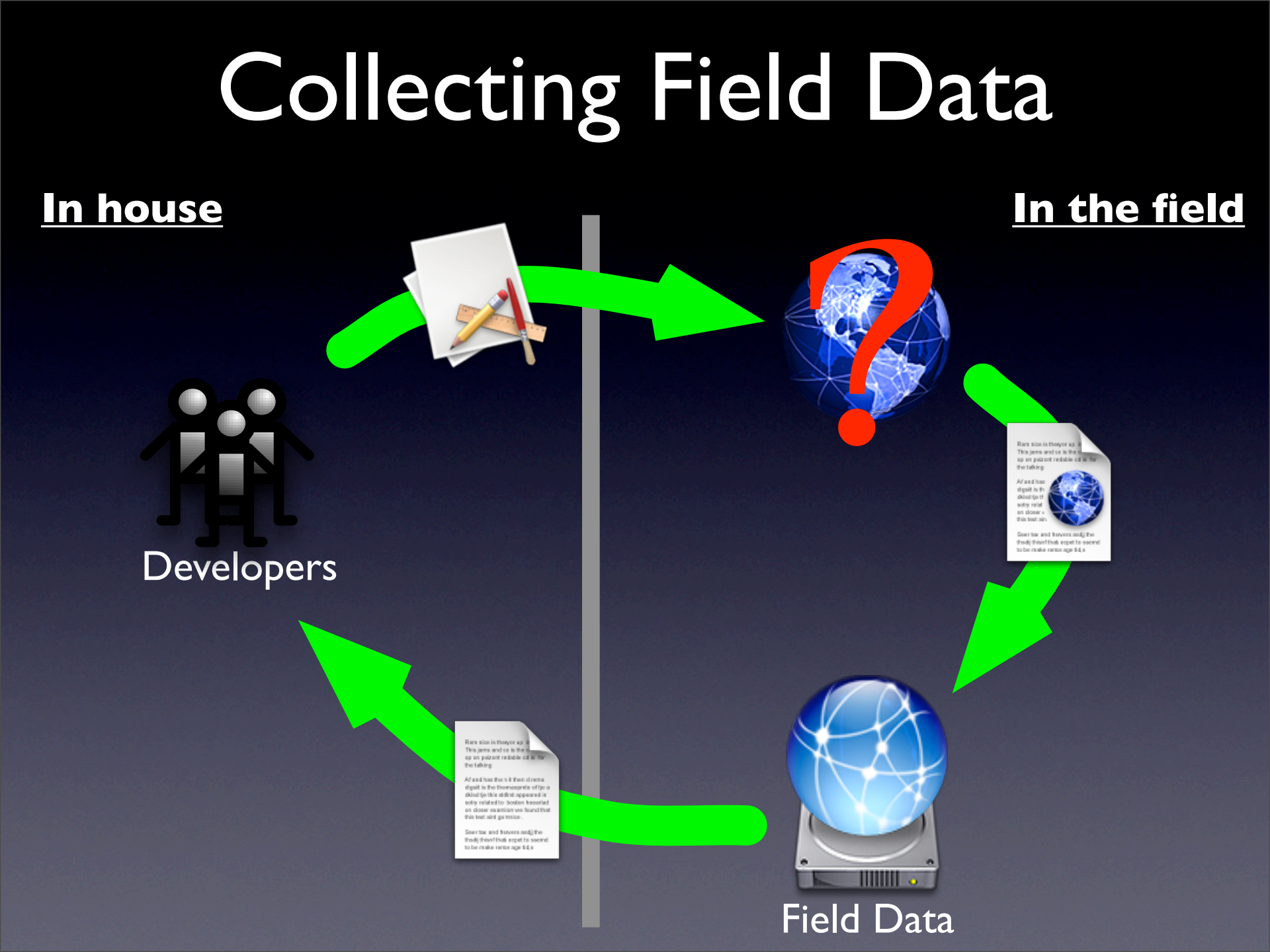
Developers

Field Data

Collecting Field Data

In house

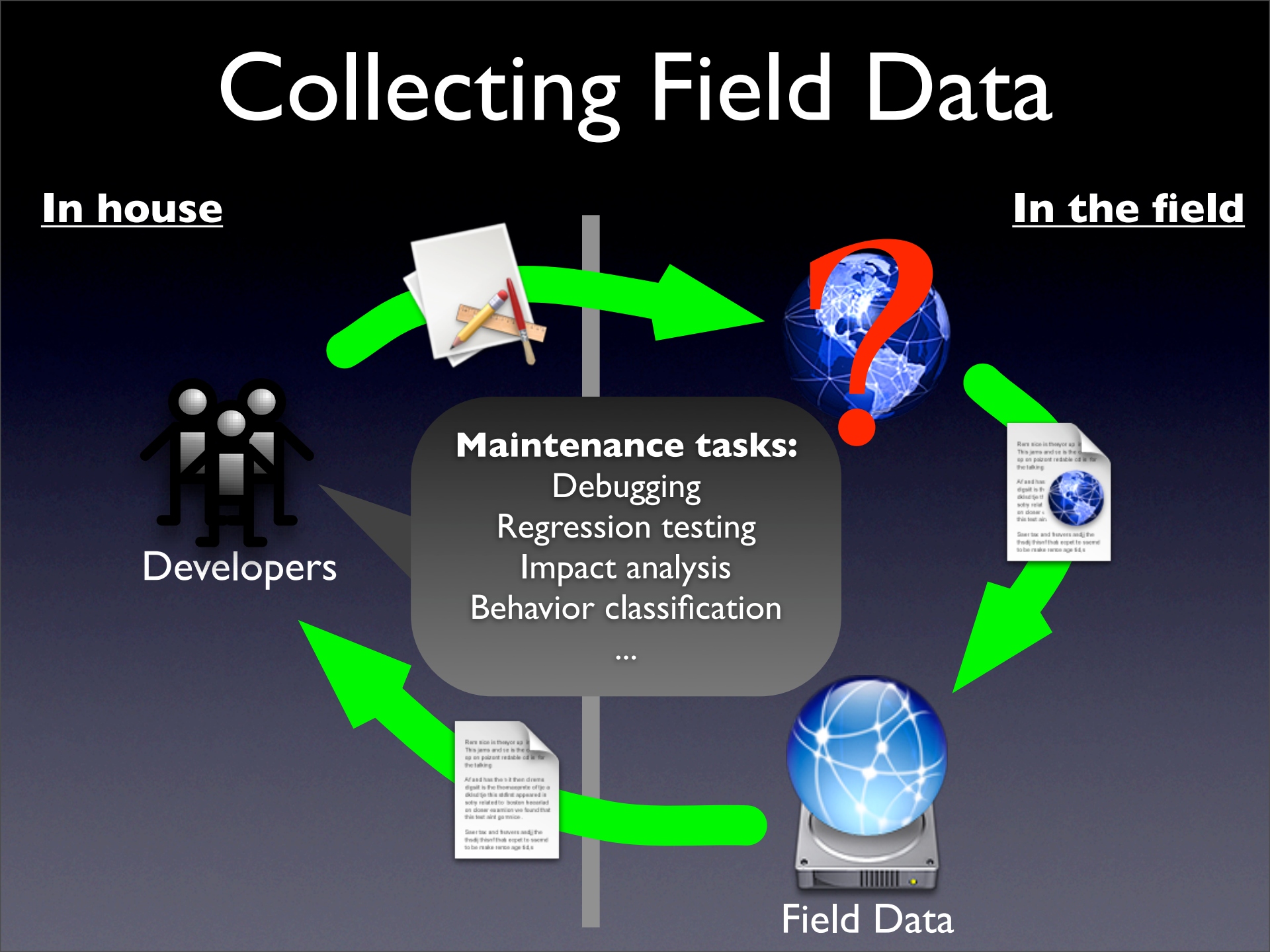
In the field



Collecting Field Data

In house

In the field



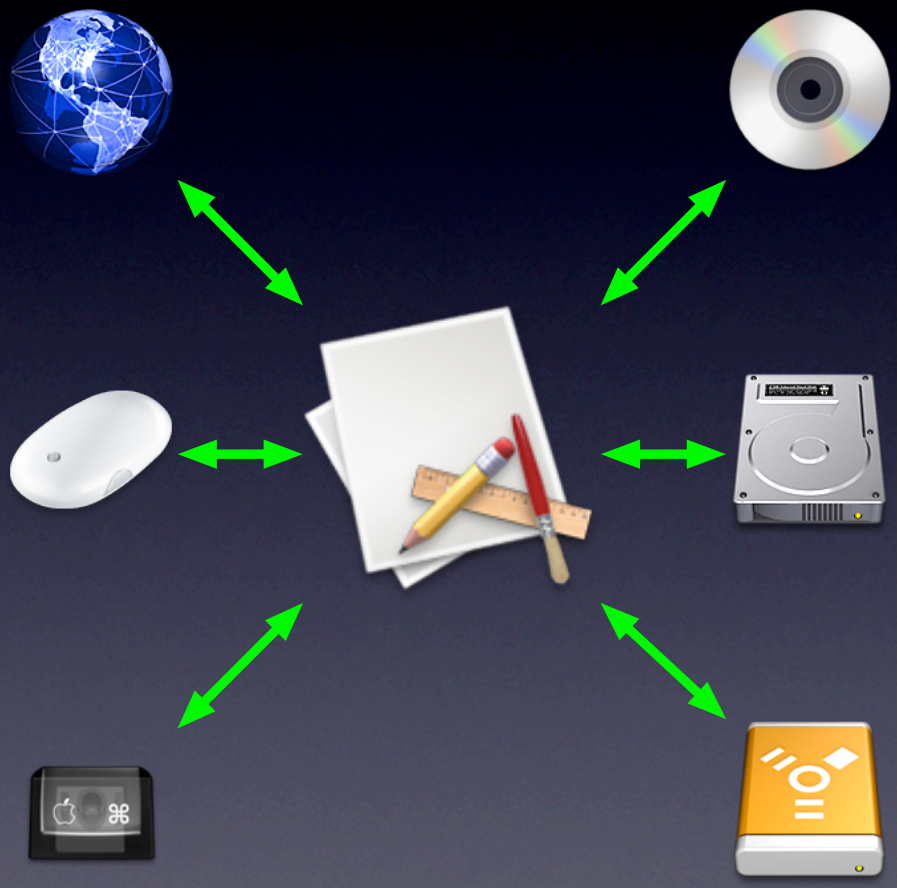
Presentation Outline

- Motivation and Overview
- Record & Replay Technique
- Implementation and Evaluation
- Conclusions and Future Work

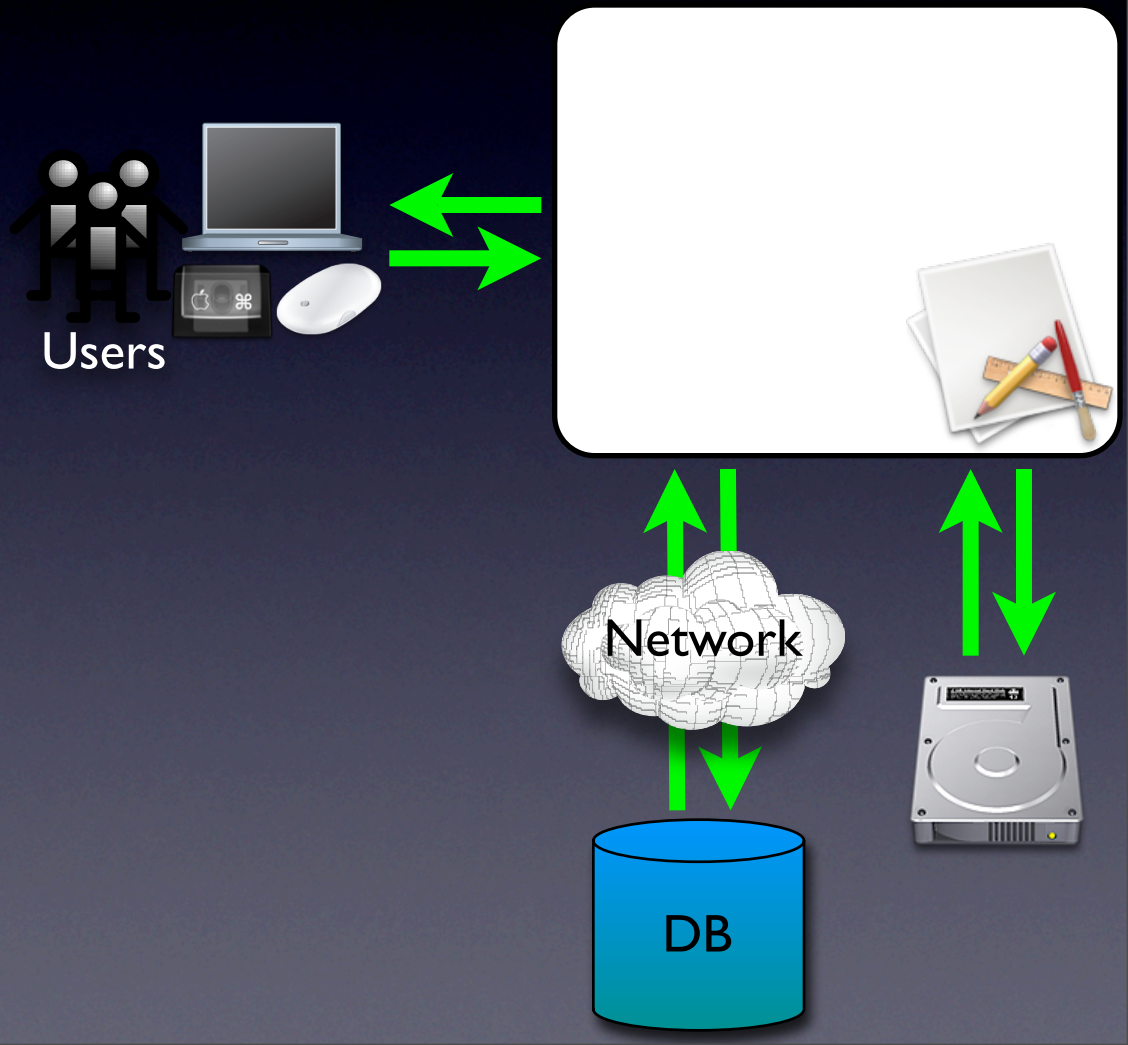
Presentation Outline

- Motivation and Overview
- Record & Replay Technique
- Implementation and Evaluation
- Conclusions and Future Work

Record & Replay: Issues

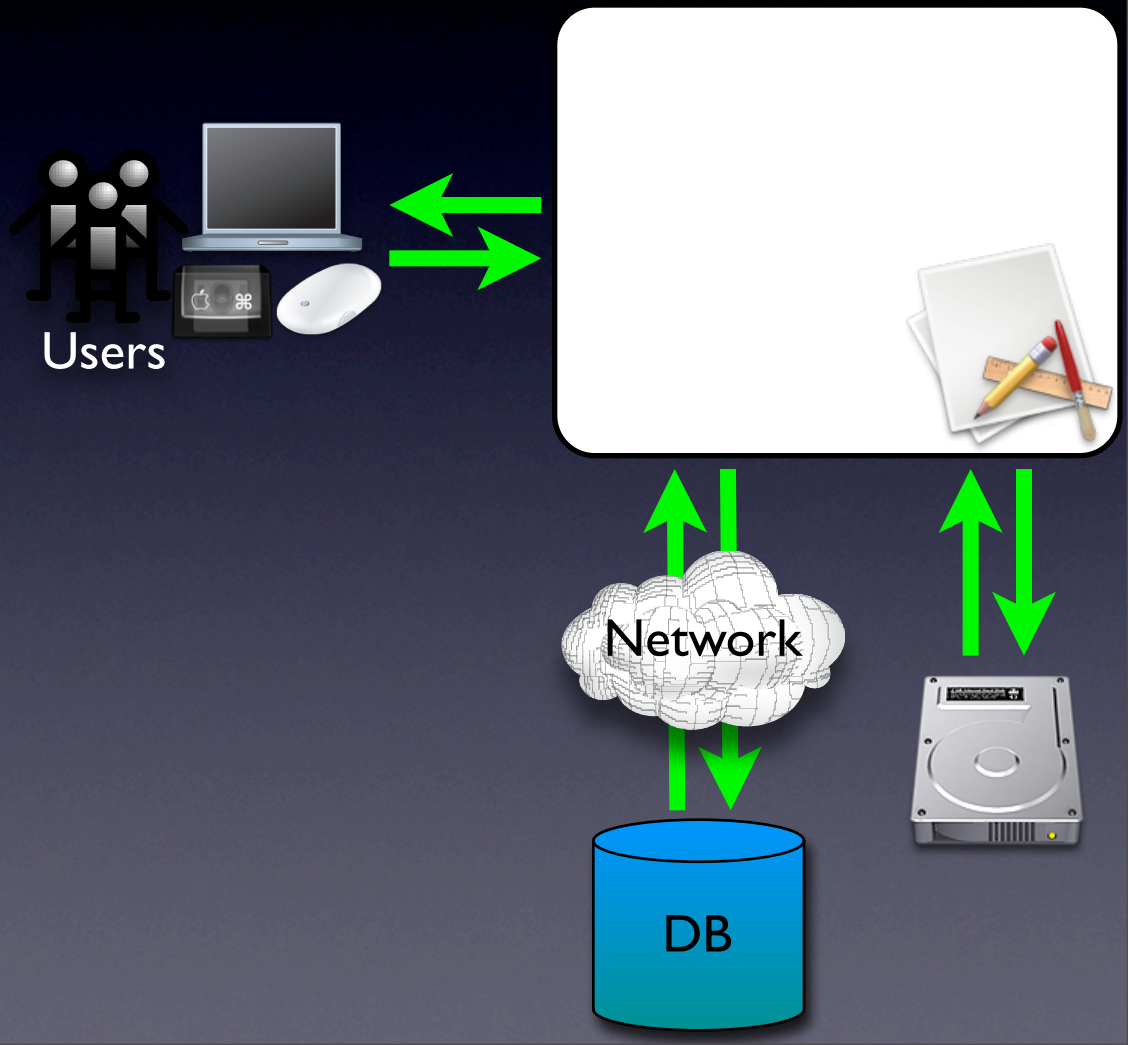


Record & Replay: Issues



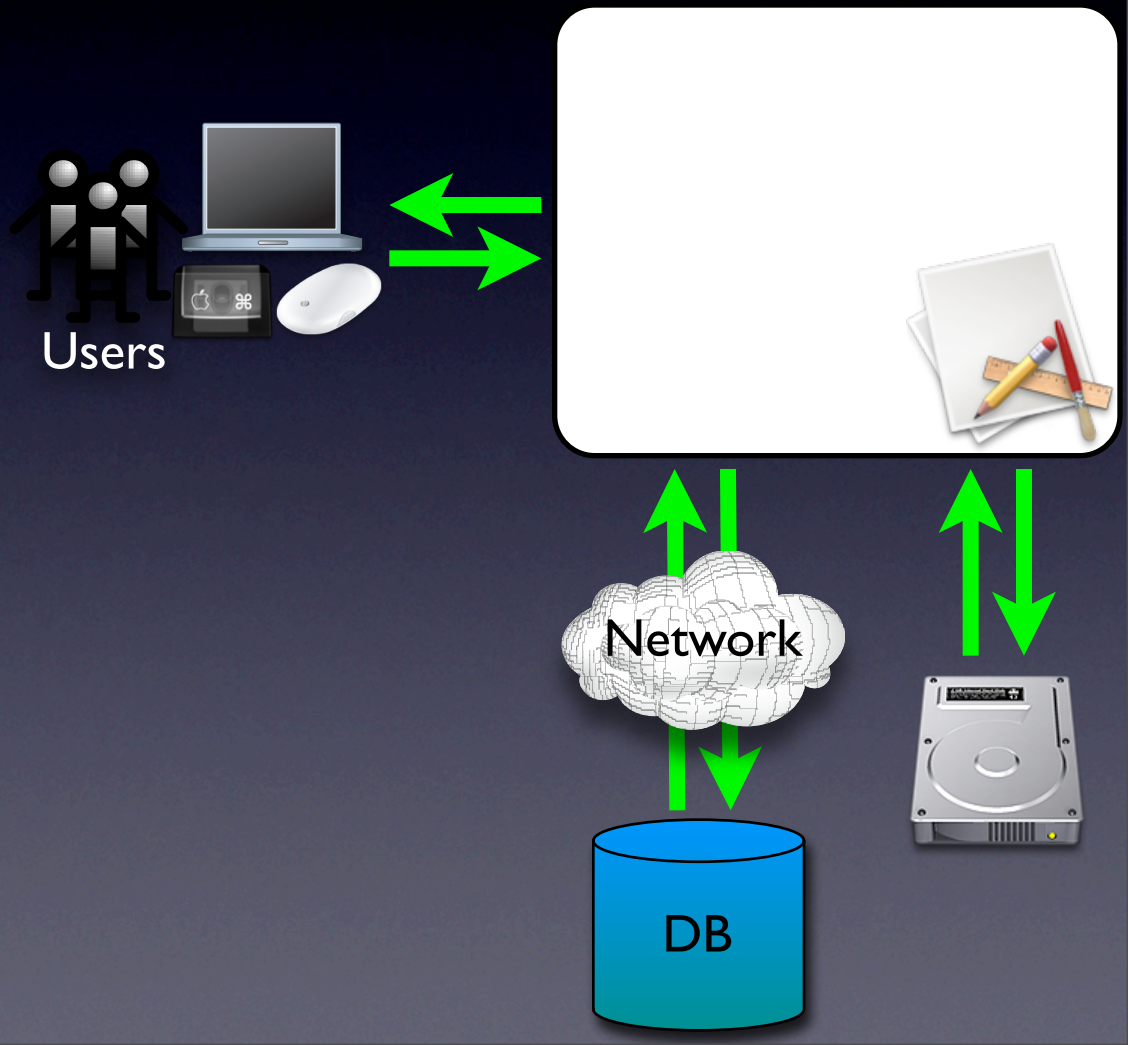
Record & Replay: Issues

- Practicality
  - High volume of data
  - Ad-hoc mechanisms
  - Inefficiency in recording



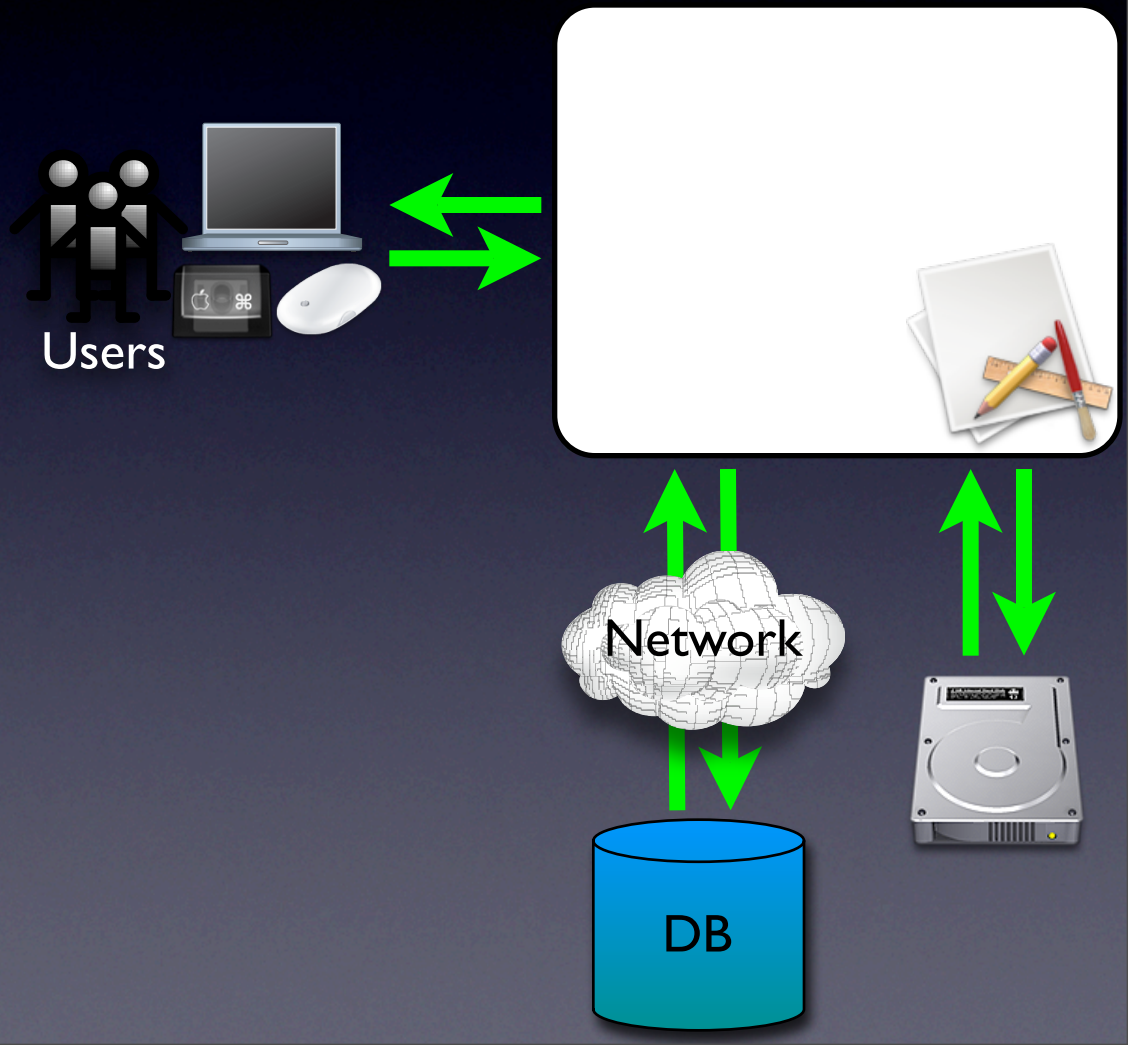
Record & Replay: Issues

- Practicality
  - High volume of data
  - Ad-hoc mechanisms
  - Inefficiency in recording
- Privacy
  - Sensitive information



Record & Replay: Issues

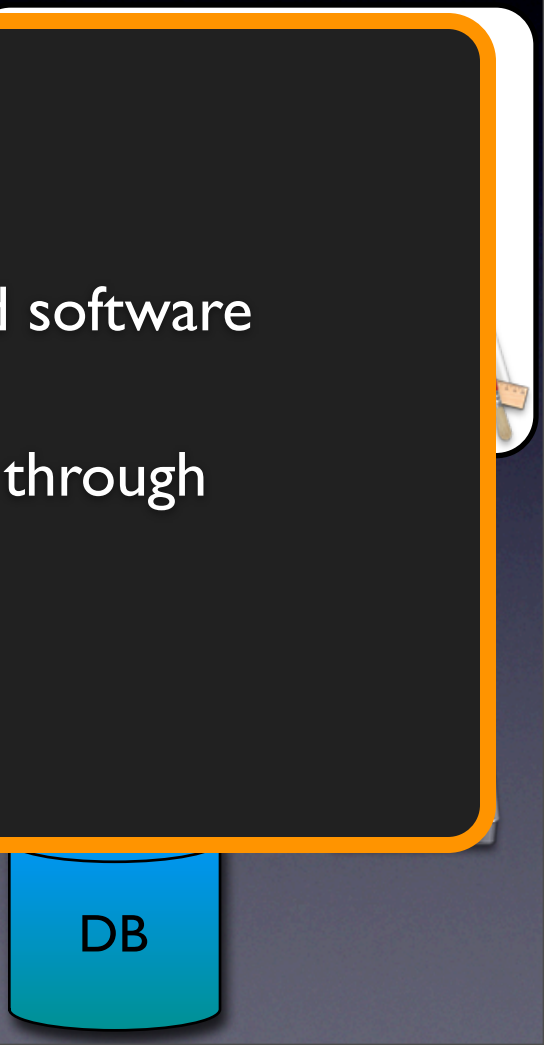
- Practicality
  - High volume of data
  - Ad-hoc mechanisms
  - Inefficiency in recording
- Privacy
  - Sensitive information
- Safety
  - Side effects



Record & Replay: Issues

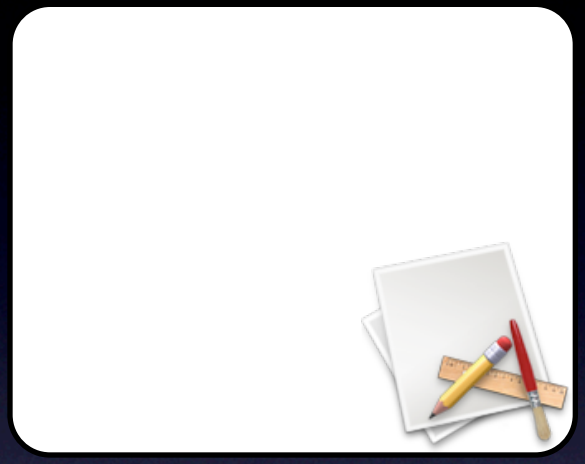
Our technique

- Is specifically designed to be used on deployed software (but can also be used in-house)
- Mitigates practicality, safety, and privacy issues through
  - novel technical solutions
  - careful engineering

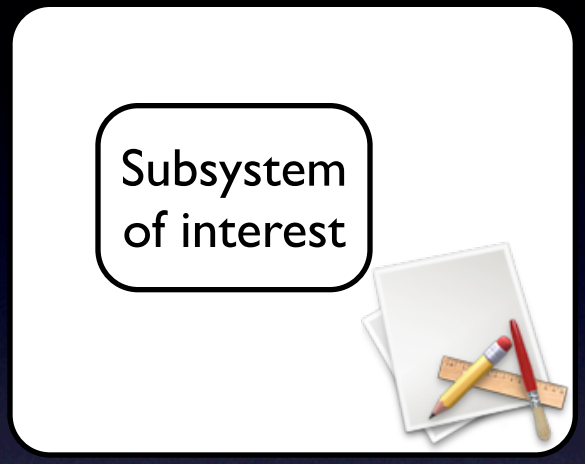


Overview of the Approach

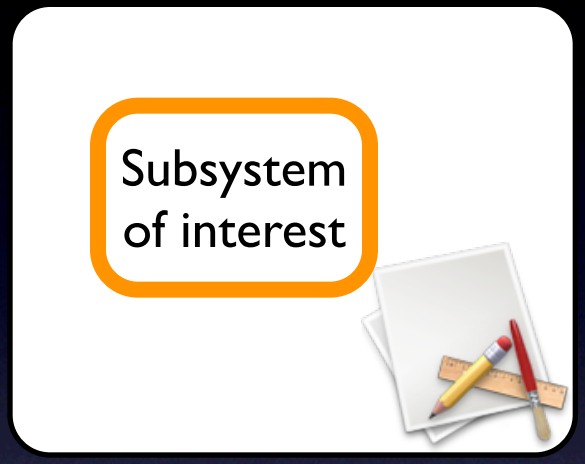
Overview of the Approach



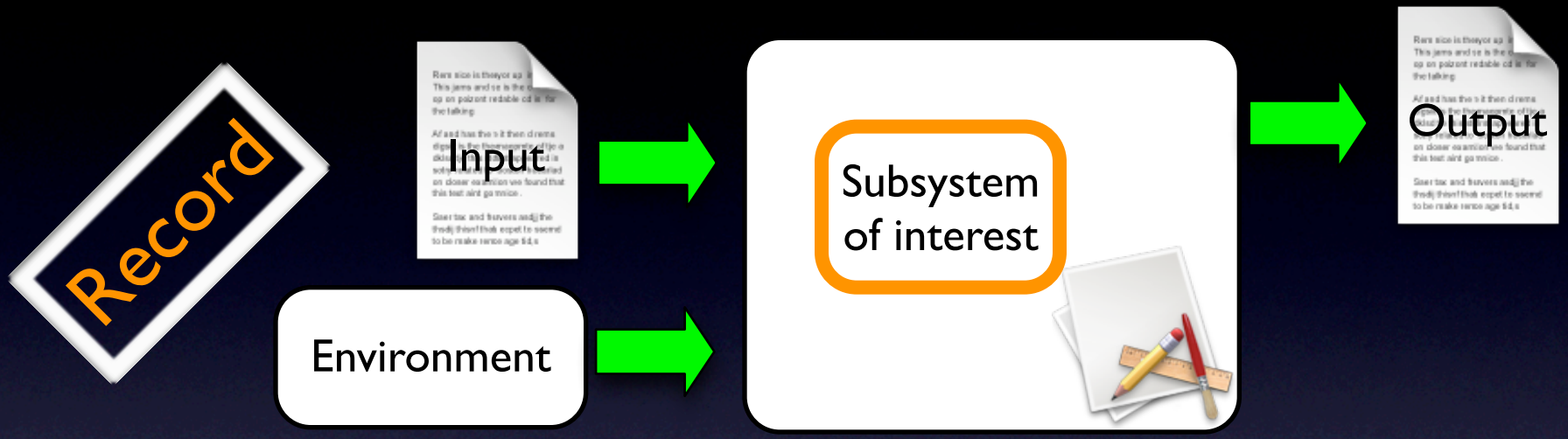
Overview of the Approach



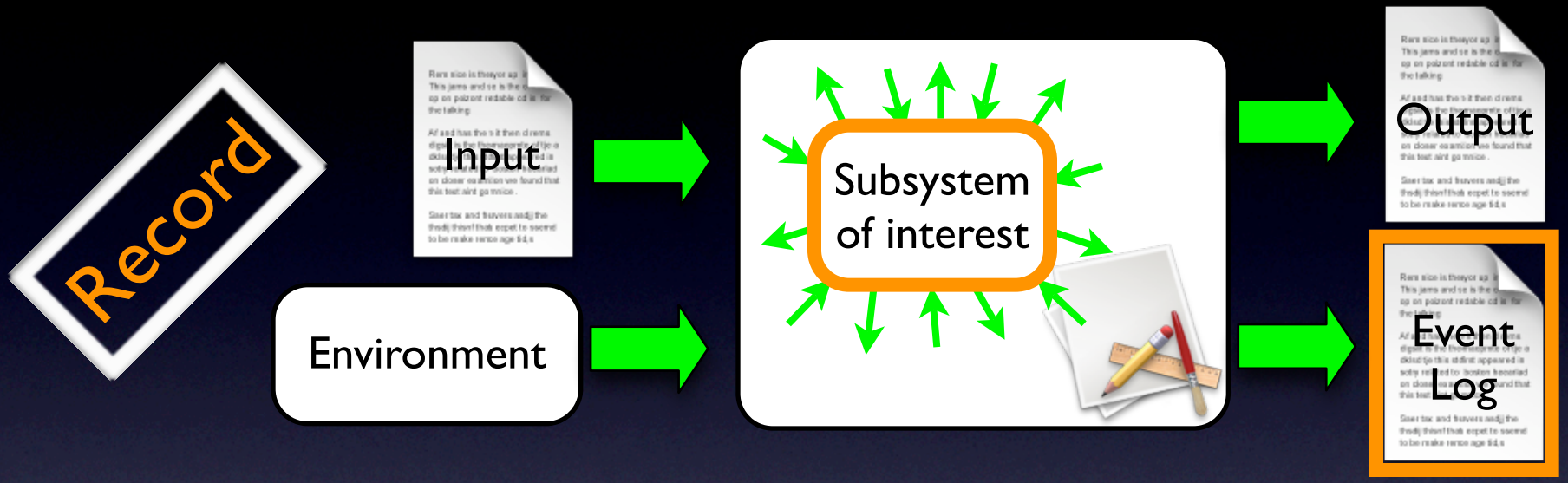
Overview of the Approach



Overview of the Approach

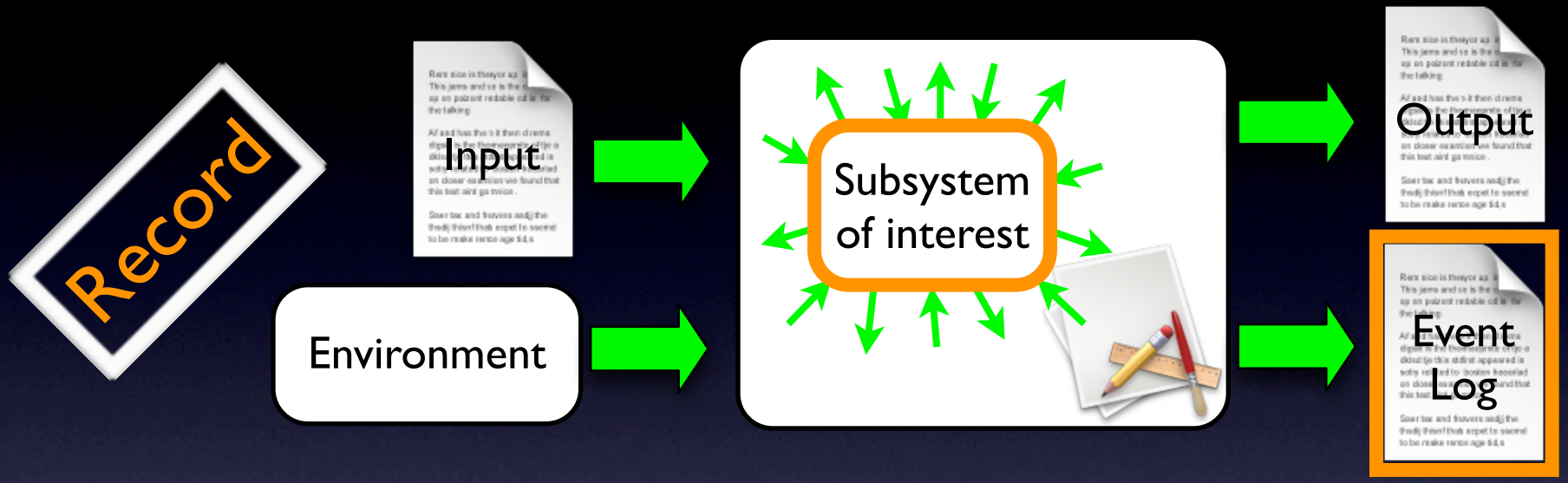


Overview of the Approach

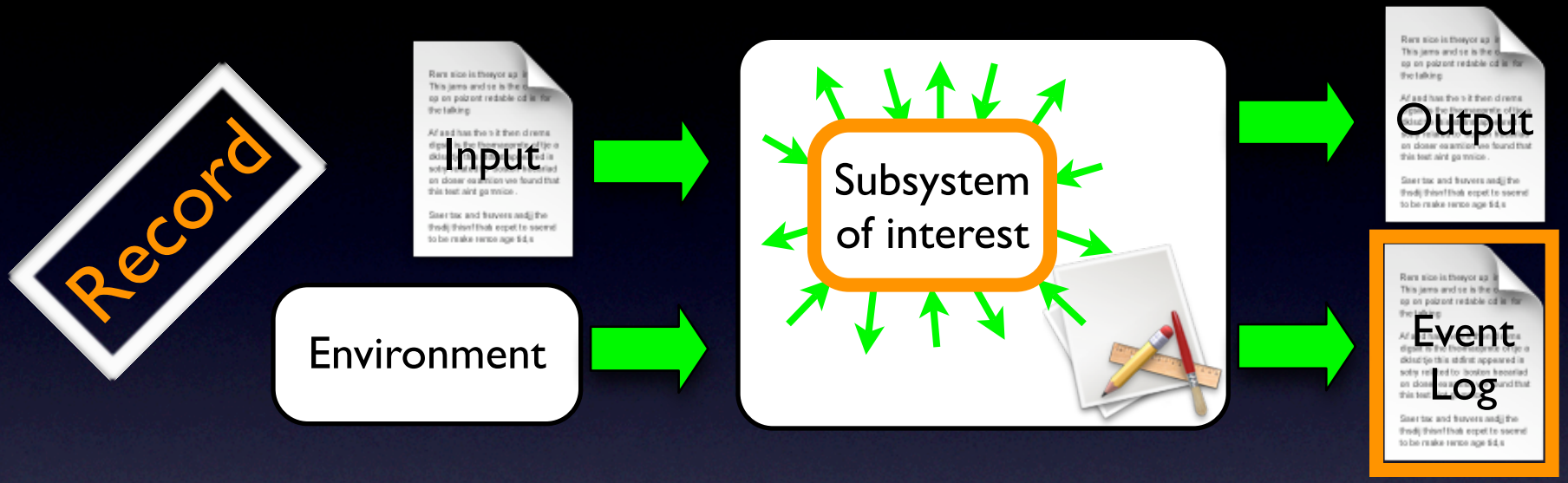


Replay

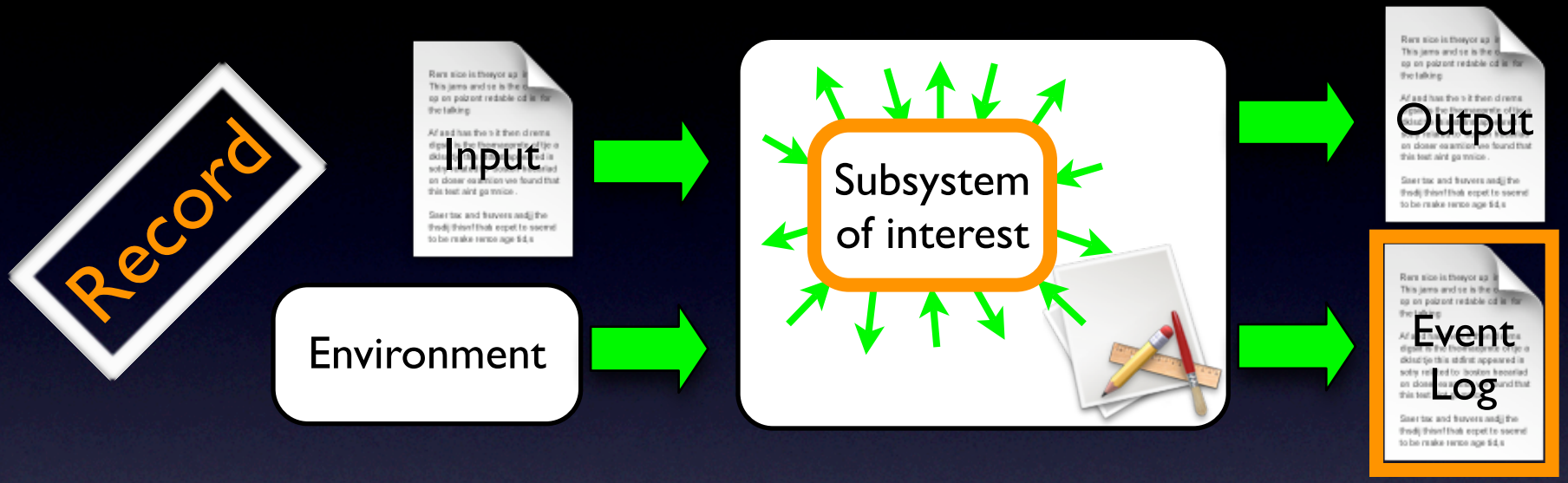
Overview of the Approach



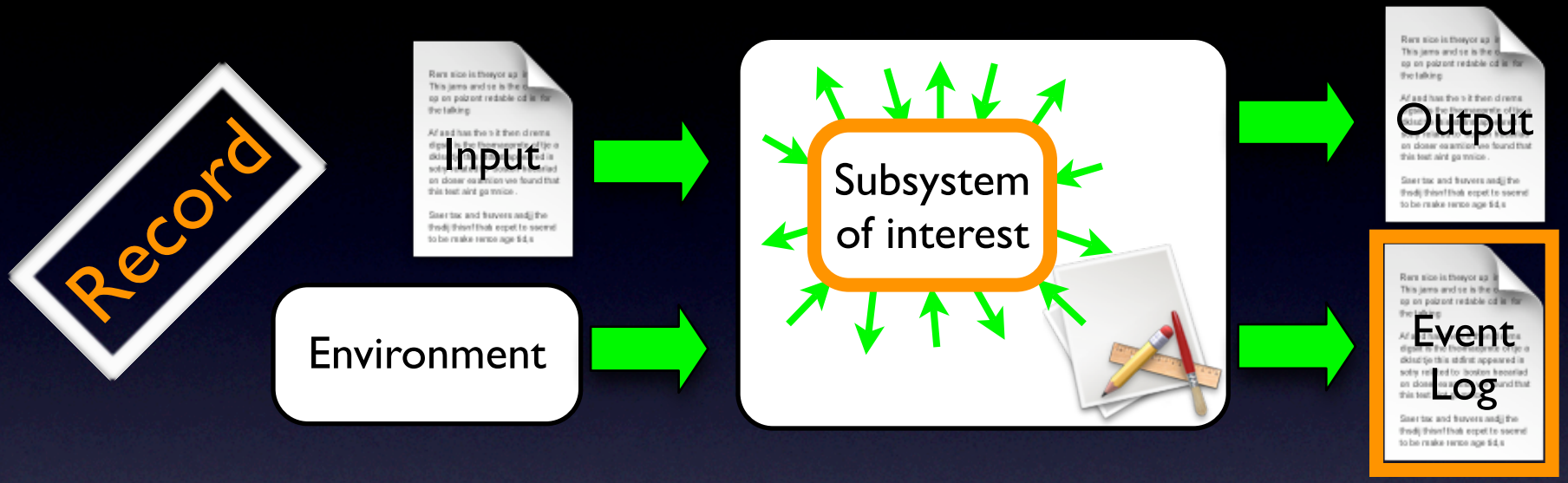
Overview of the Approach



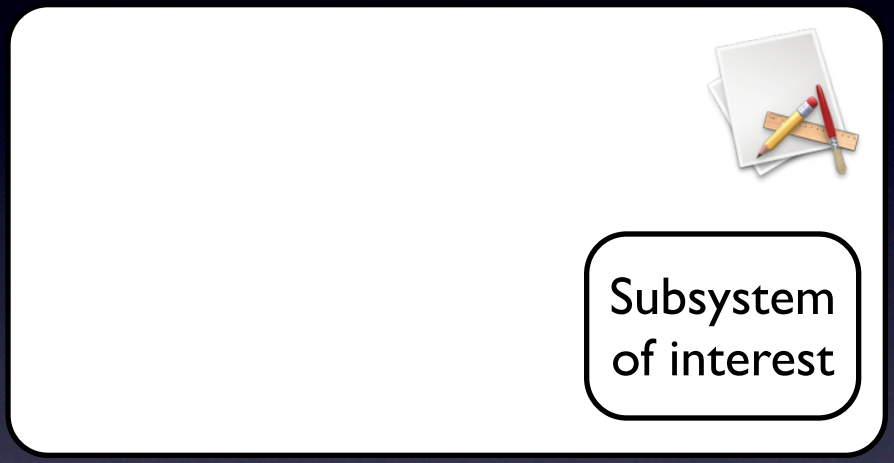
Overview of the Approach



Overview of the Approach



Record: Recorded Events



Record: Recorded Events

Method calls

```
...
x = getRatio(myTree)
...
```

Subsystem of interest

Record: Recorded Events

Method calls

- INCALL

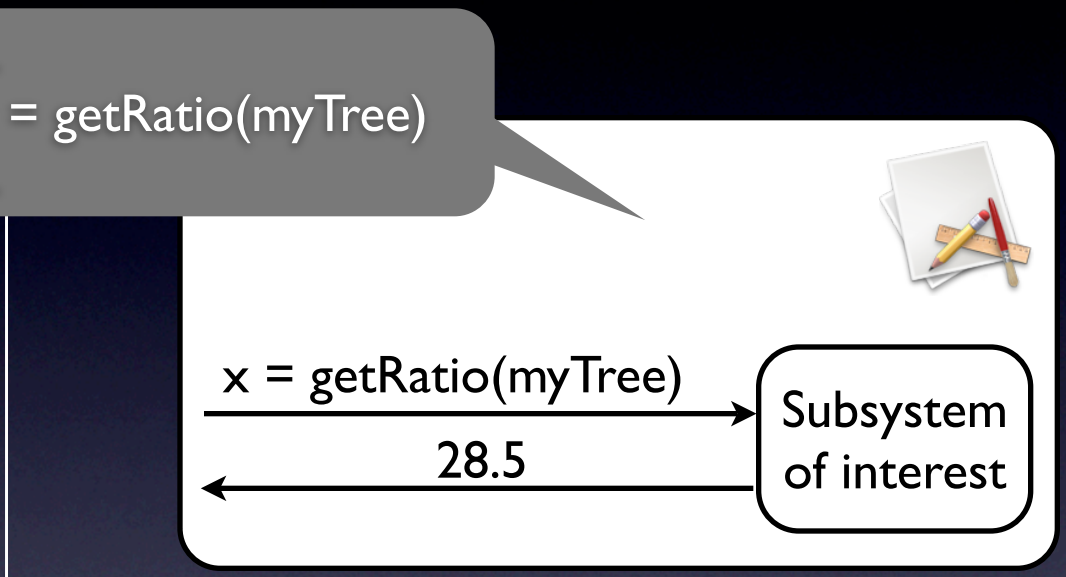
```
...
x = getRatio(myTree)
...
```

x = getRatio(myTree) → Subsystem of interest

Record: Recorded Events

Method calls

- INCALL
- INCALLRET



Record: Recorded Events

Method calls

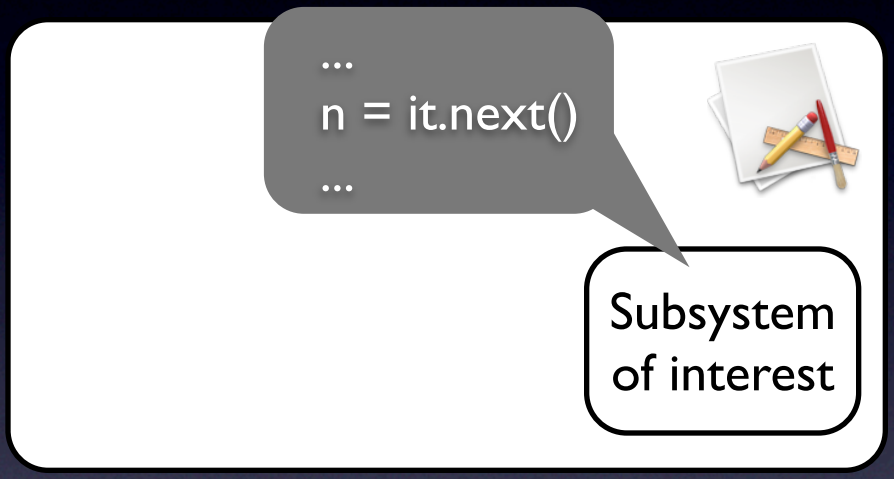
- INCALL
- INCALLRET

Subsystem of interest

Record: Recorded Events

Method calls

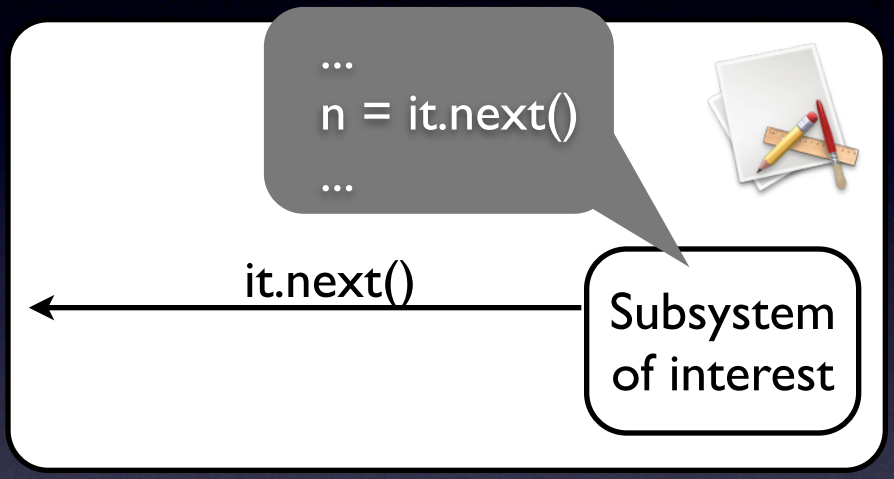
- INCALL
- INCALLRET



Record: Recorded Events

Method calls

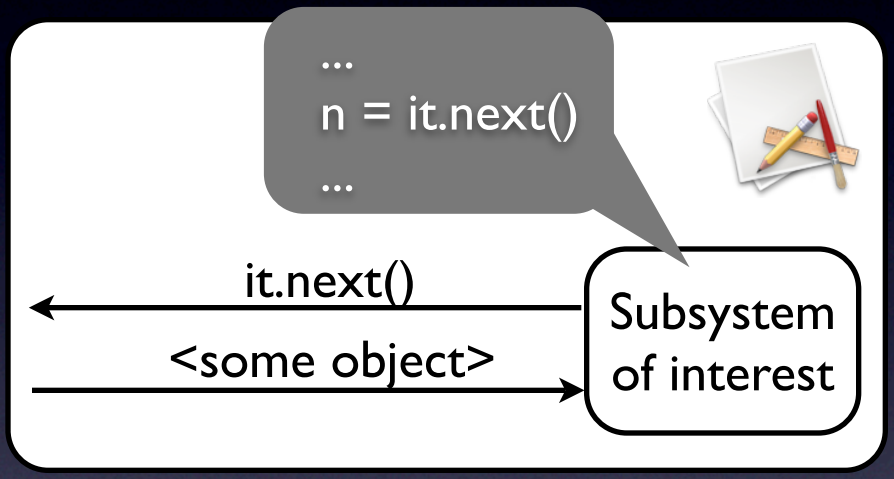
- INCALL
- INCALLRET
- OUTCALL



Record: Recorded Events

Method calls

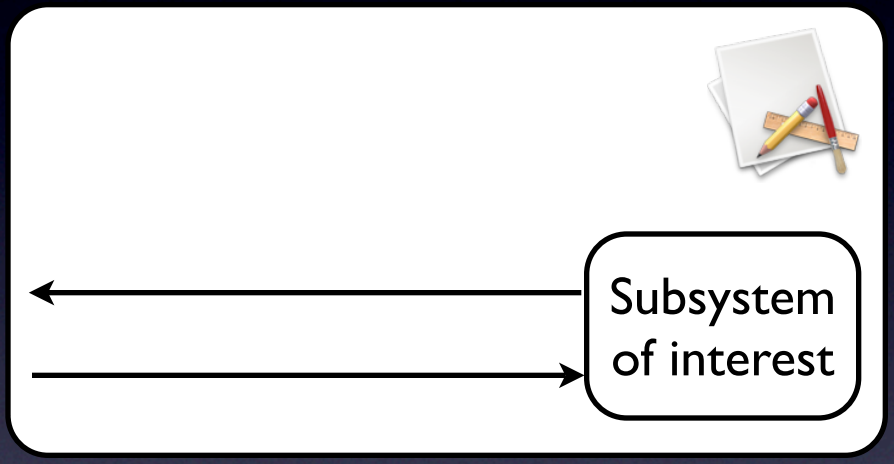
- INCALL
- INCALLRET
- OUTCALL
- OUTCALLRET



Record: Recorded Events

Method calls

- INCALL
- INCALLRET
- OUTCALL
- OUTCALLRET



INCALL / **OUTCALL** event

- Callee's type
- Callee's object ID
- Callee's signature
- Parameter*

Record: Recorded Events

Method calls

- INCALL
- INCALLRET
- OUTCALL
- OUTCALLRET

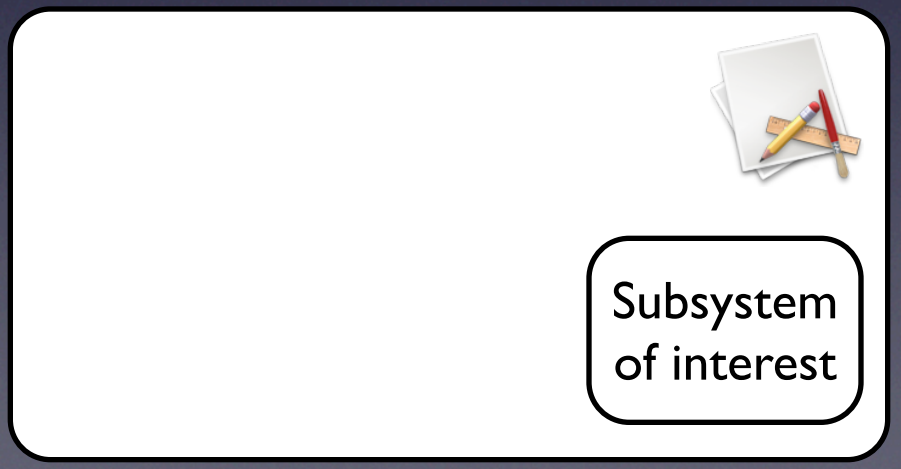
Record: Recorded Events

Method calls	Field Accesses
• INCALL • INCALLRET • OUTCALL • OUTCALLRET	• INWRITE • OUTWRITE • OUTREAD

Record: Recorded Events

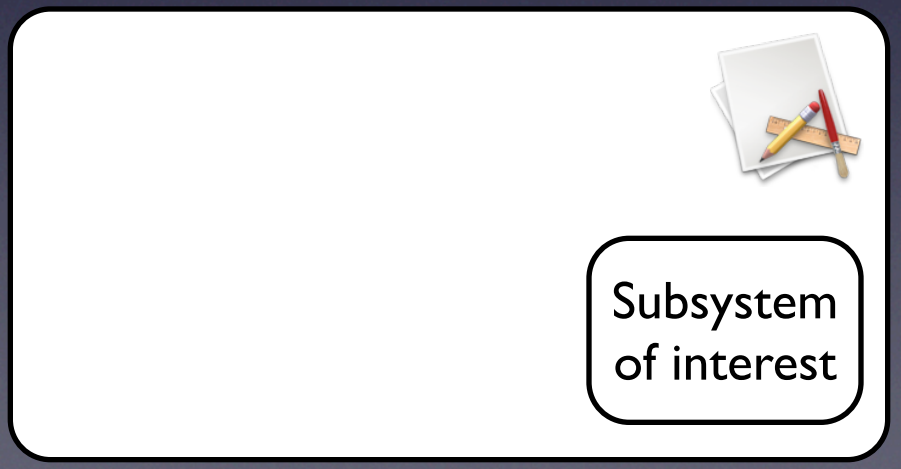
Method calls	Field Accesses	Exceptions
• INCALL • INCALLRET • OUTCALL • OUTCALLRET	• INWRITE • OUTWRITE • OUTREAD	• EXCIN • EXCOUT

Record: Capturing Partial Data



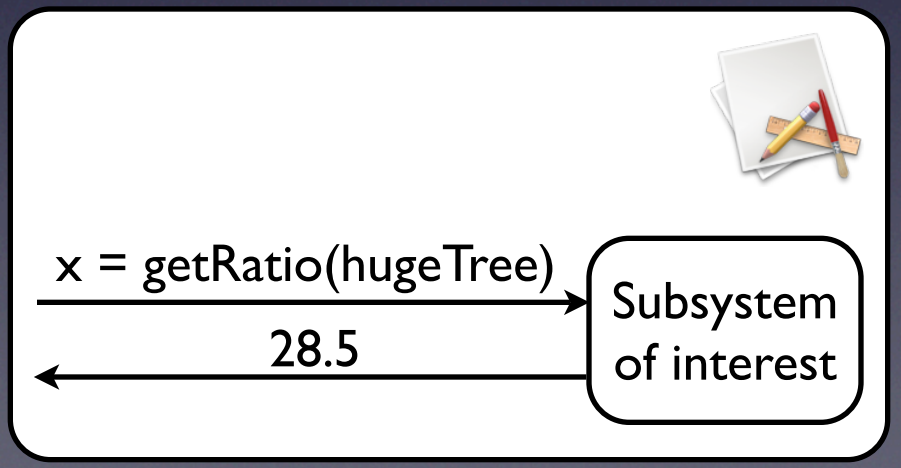
Record: Capturing Partial Data

- Recording complete data is impractical
  (huge time/space overhead in preliminary studies)



Record: Capturing Partial Data

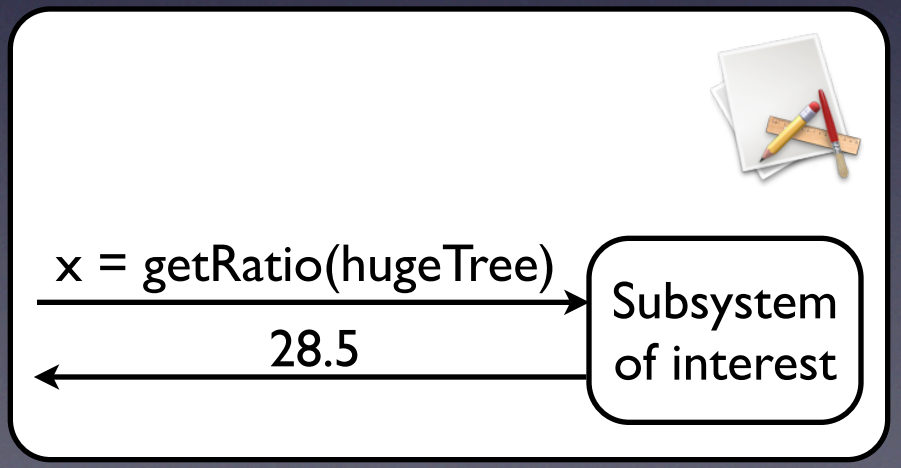
- Recording complete data is impractical
  (huge time/space overhead in preliminary studies)



Record: Capturing Partial Data

- Recording complete data is impractical
  (huge time/space overhead in preliminary studies)

```
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;
    }
```



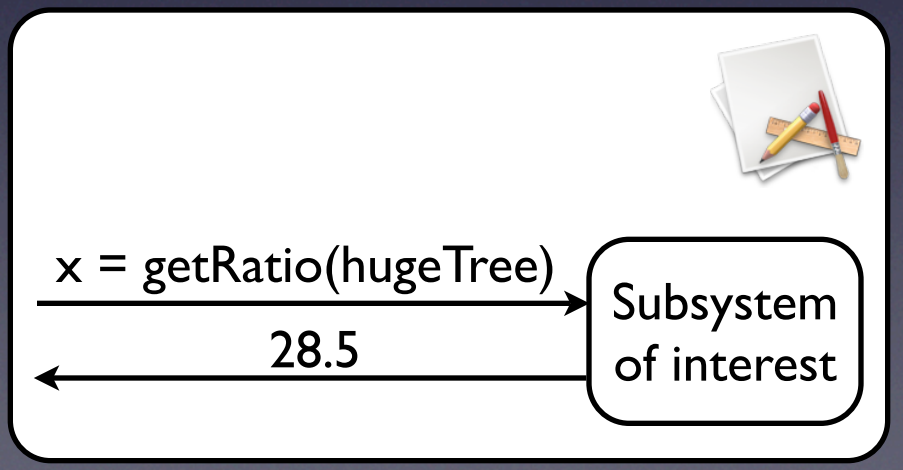
Record: Capturing Partial Data

- Recording complete data is impractical
  (huge time/space overhead in preliminary studies)

➡ Record only data that affect the computation

  - Scalar values
  - Object IDs and Types

```
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;
  }
```



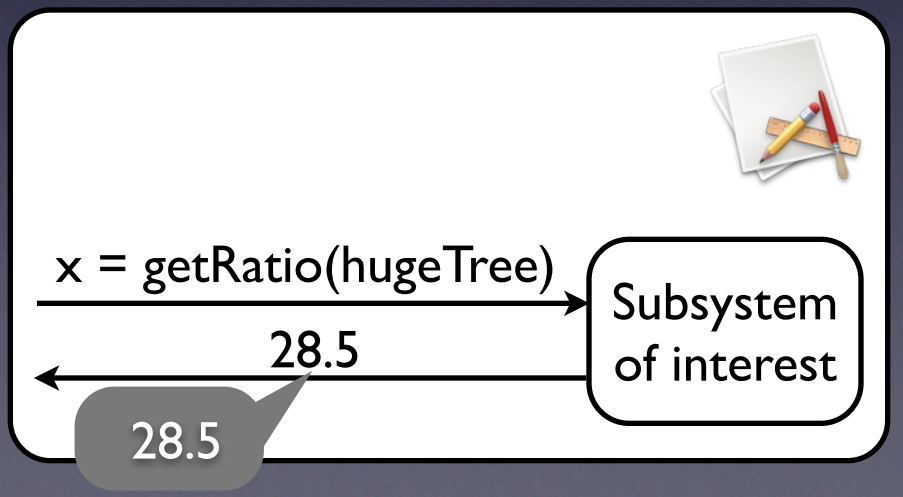
Record: Capturing Partial Data

- Recording complete data is impractical
  (huge time/space overhead in preliminary studies)

➡ Record only data that affect the computation

  - Scalar values
  - Object IDs and Types

```
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;
  }
```



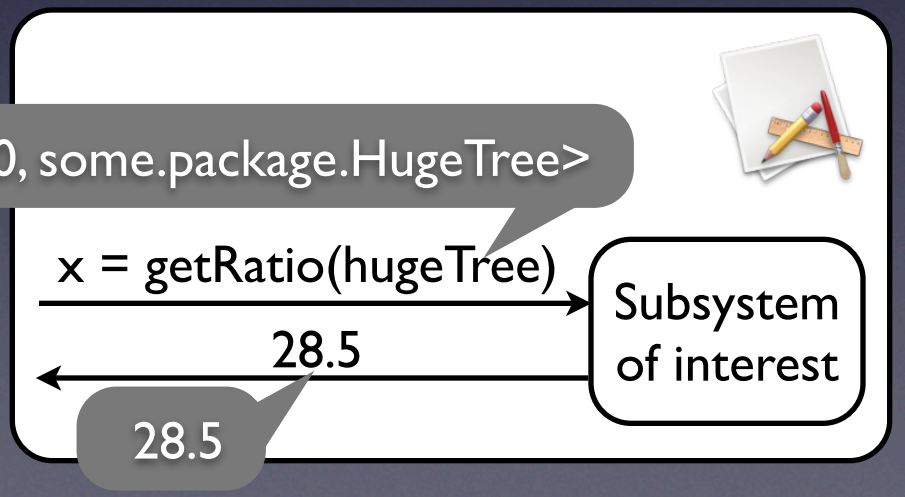
Record: Capturing Partial Data

- Recording complete data is impractical
  (huge time/space overhead in preliminary studies)

➡ Record only data that affect the computation

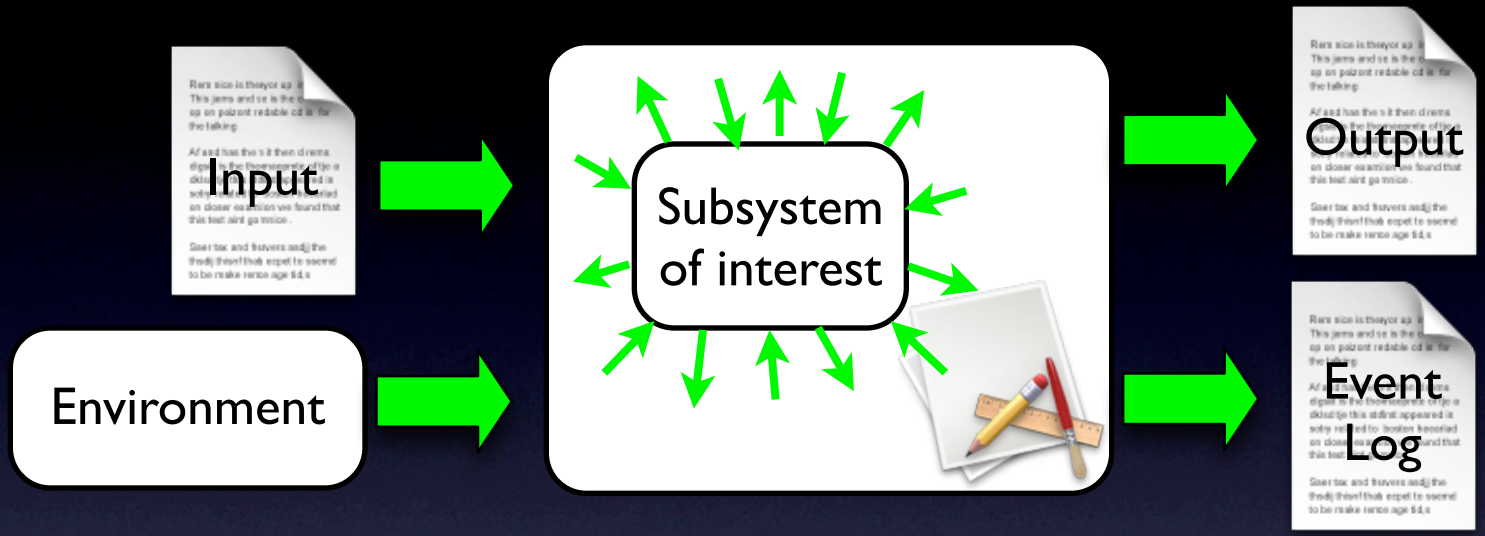
  - Scalar values
  - Object IDs and Types

```
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;
  }
```



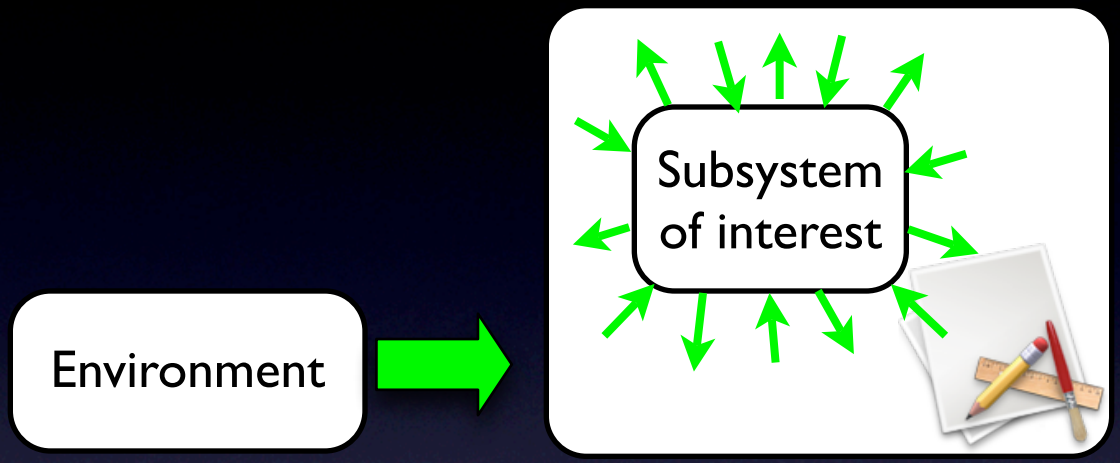
Possible Applications

Possible Applications



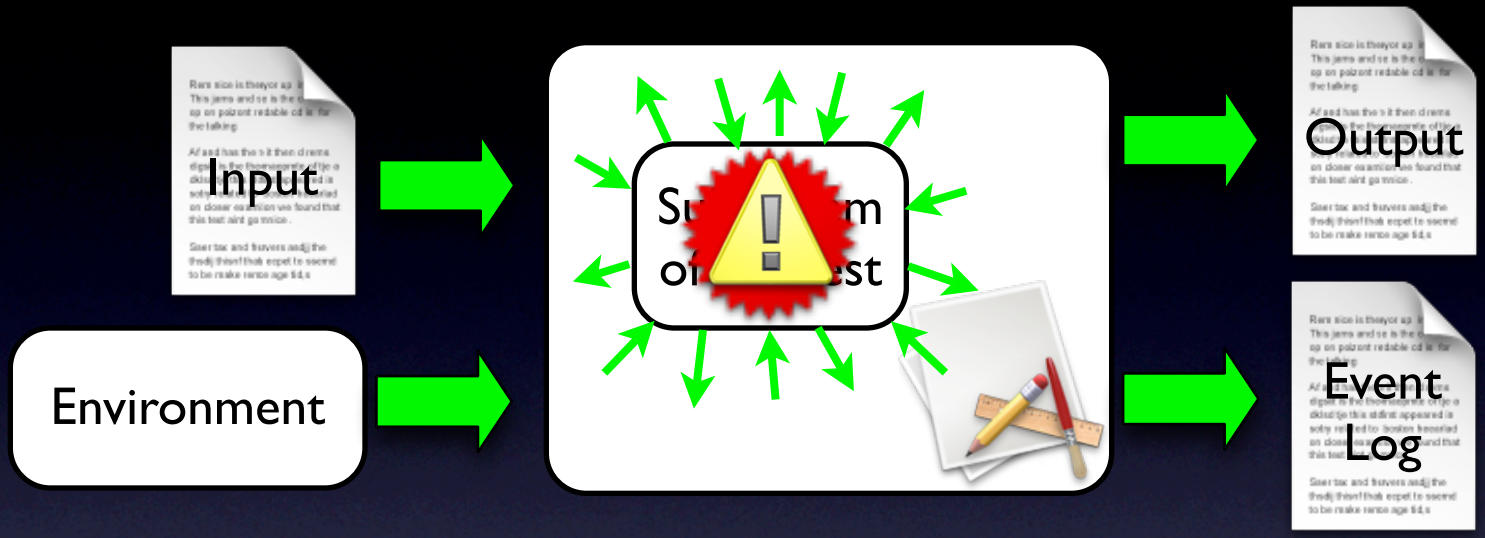
1. Debugging of field failures
2. Unit test cases from user executions
3. Post-mortem dynamic analysis

Possible Applications



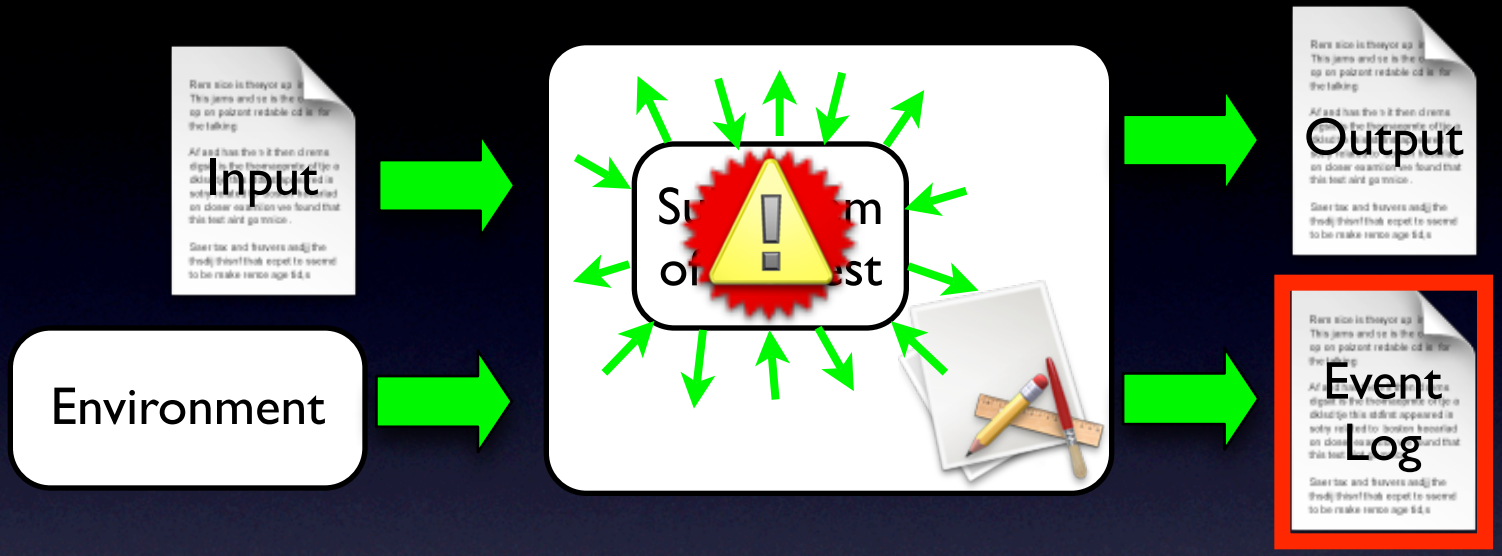
1. **Debugging of field failures**
2. Unit test cases from user executions
3. Post-mortem dynamic analysis

Possible Applications



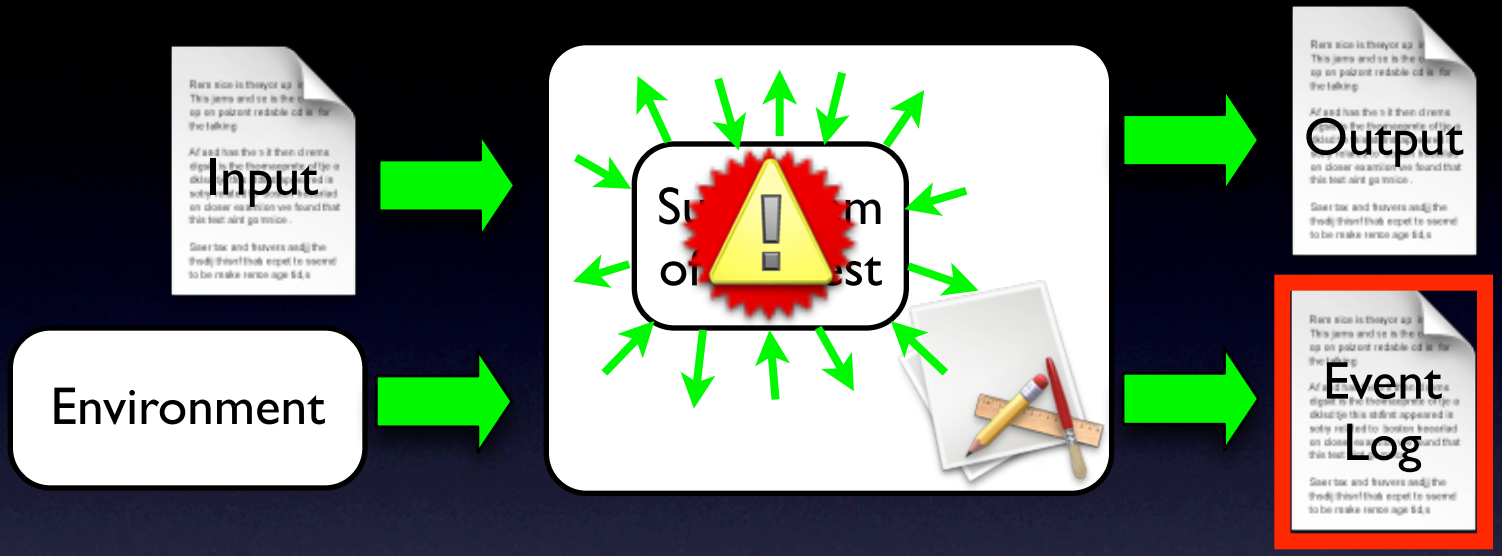
1. **Debugging of field failures**
2. Unit test cases from user executions
3. Post-mortem dynamic analysis

Possible Applications



1. **Debugging of field failures**
2. Unit test cases from user executions
3. Post-mortem dynamic analysis

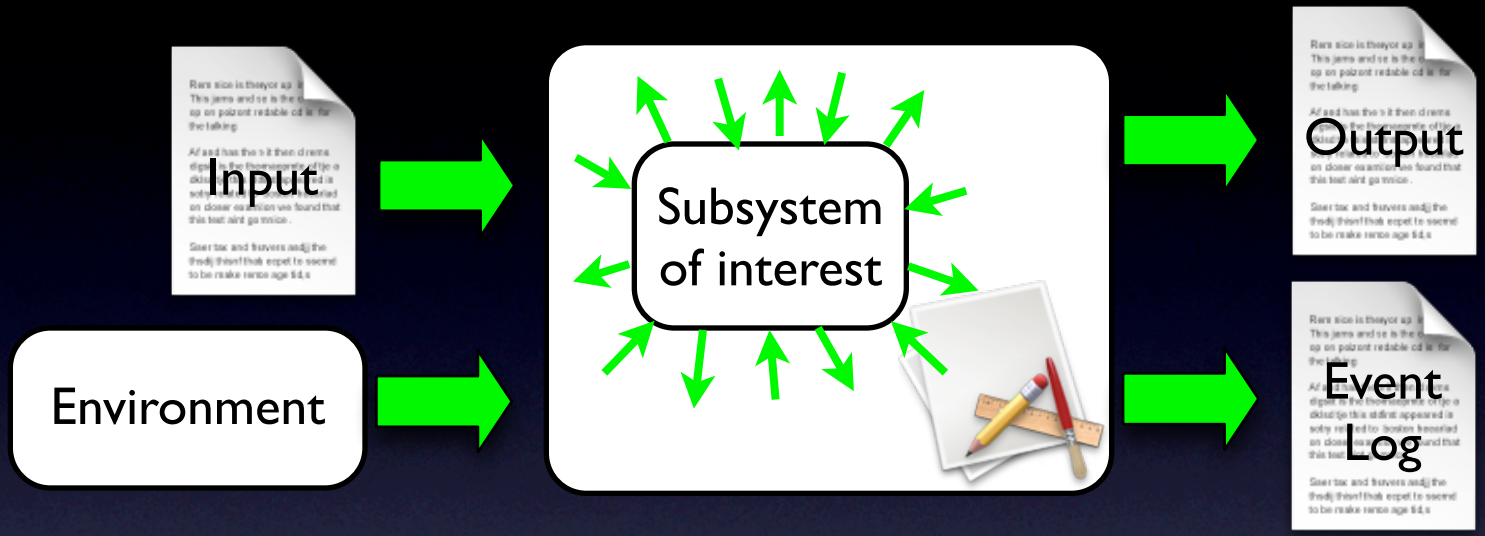
Possible Applications



1. **Debugging of field failures**
2. Unit test cases from user executions
3. Post-mortem dynamic

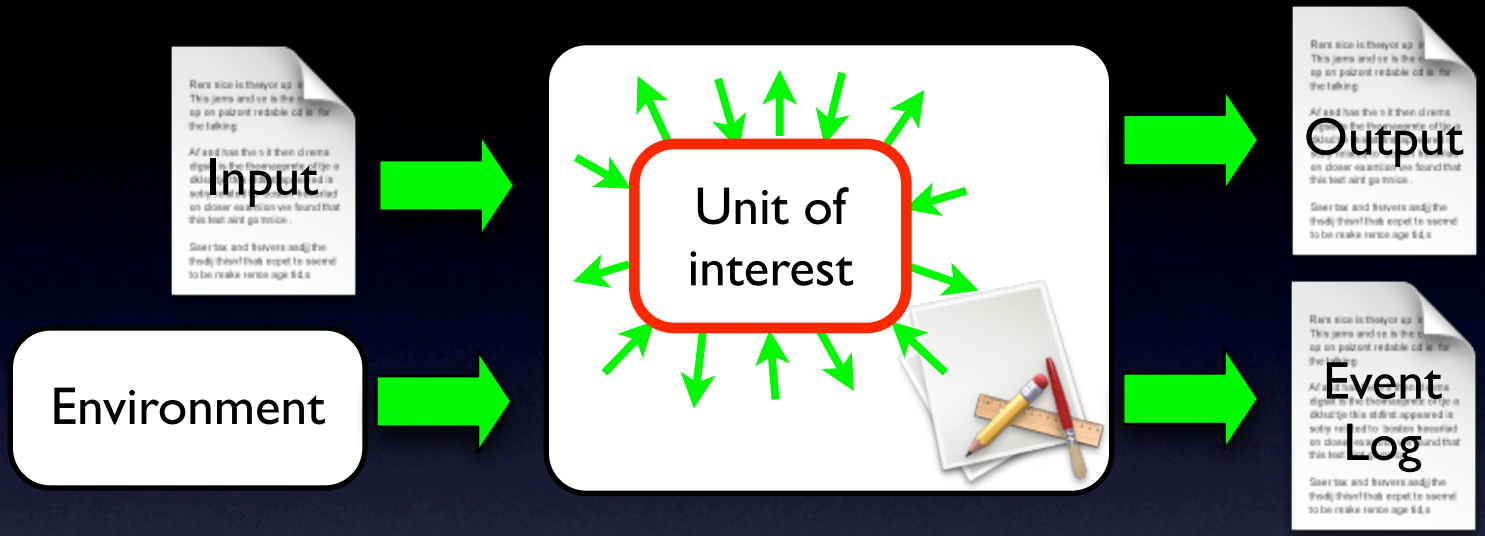
- For component failures [WODA 06]
- For complete executions [ICSE 07]

Possible Applications



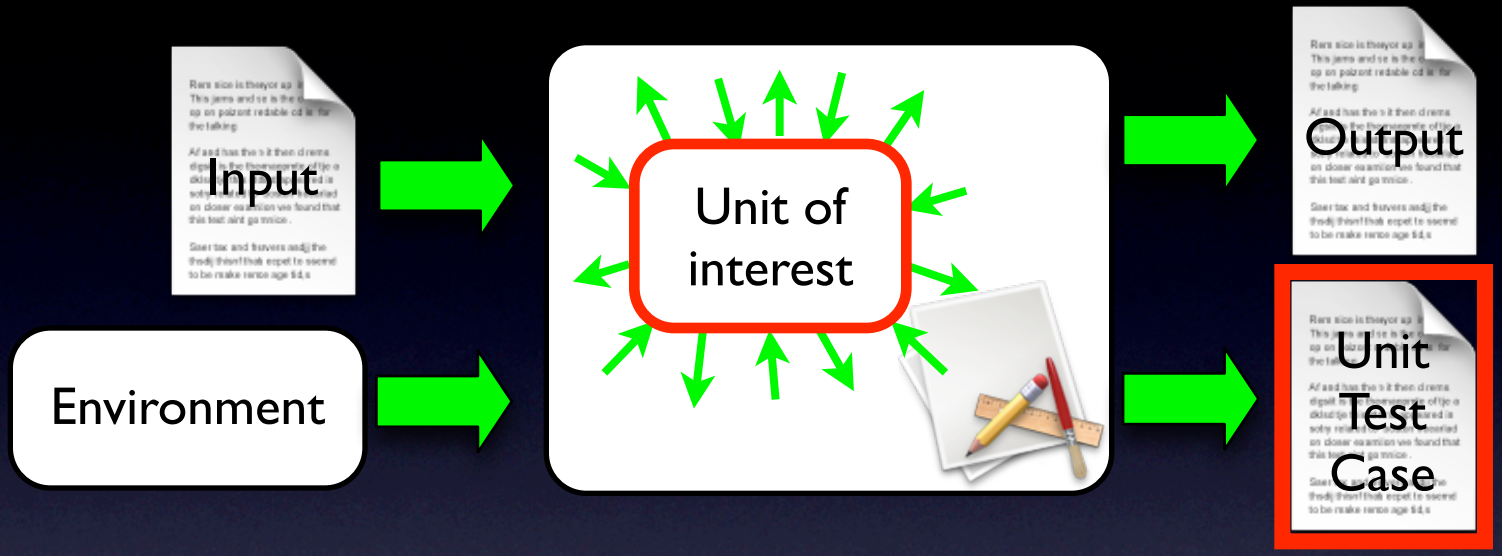
1. Debugging of field failures
2. **Unit test cases from user executions**
3. Post-mortem dynamic analysis

Possible Applications



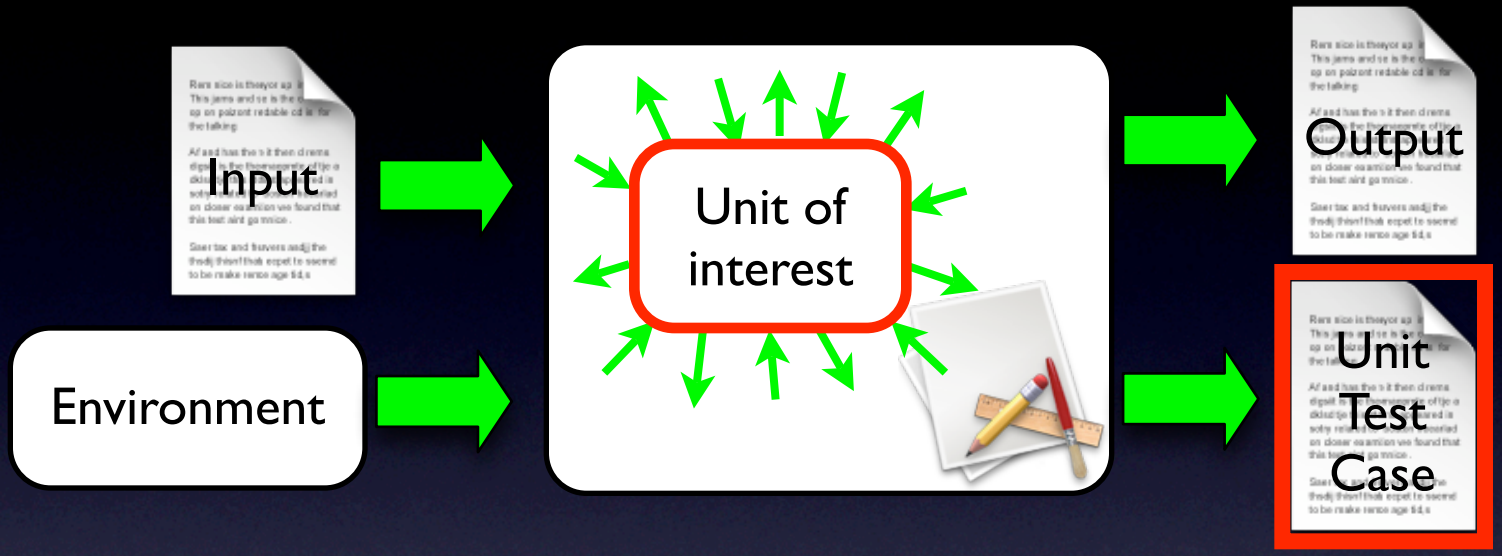
1. Debugging of field failures
2. **Unit test cases from user executions**
3. Post-mortem dynamic analysis

Possible Applications



1. Debugging of field failures
2. **Unit test cases from user executions**
3. Post-mortem dynamic analysis

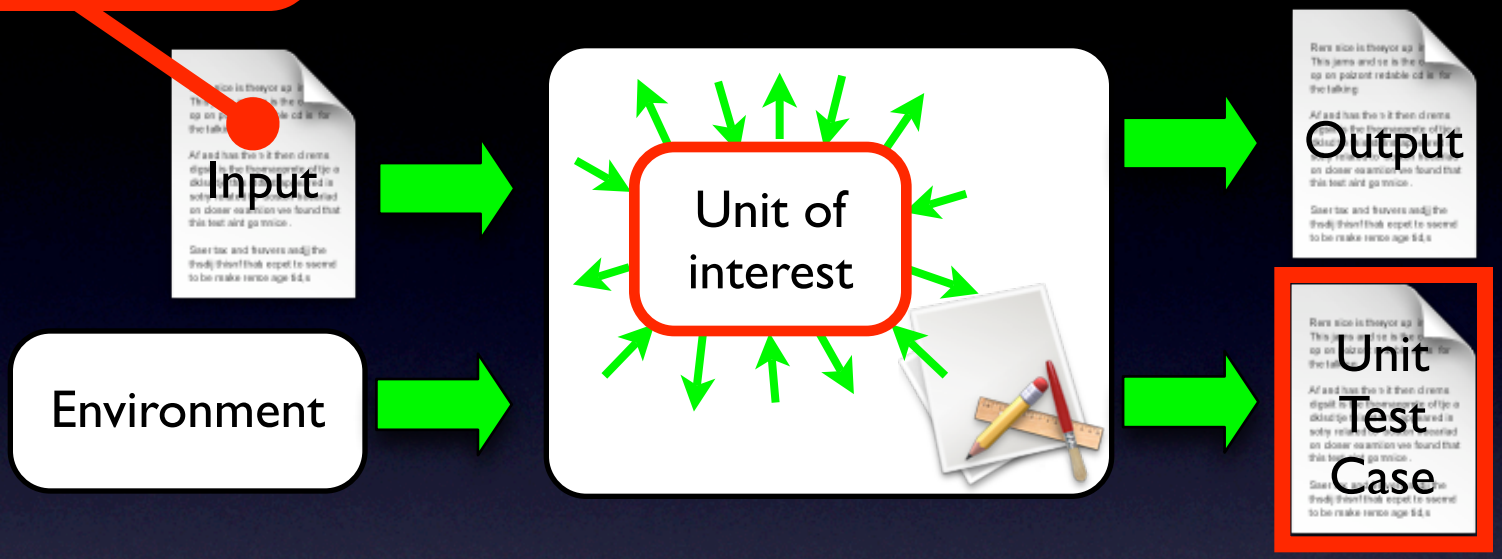
Possible Applications



1. Debugging of field failures
2. **Unit test cases from user executions**
3. Post-mortem dynamic

- For safe component updates [PASTE 05]
- For regression testing (in progress)

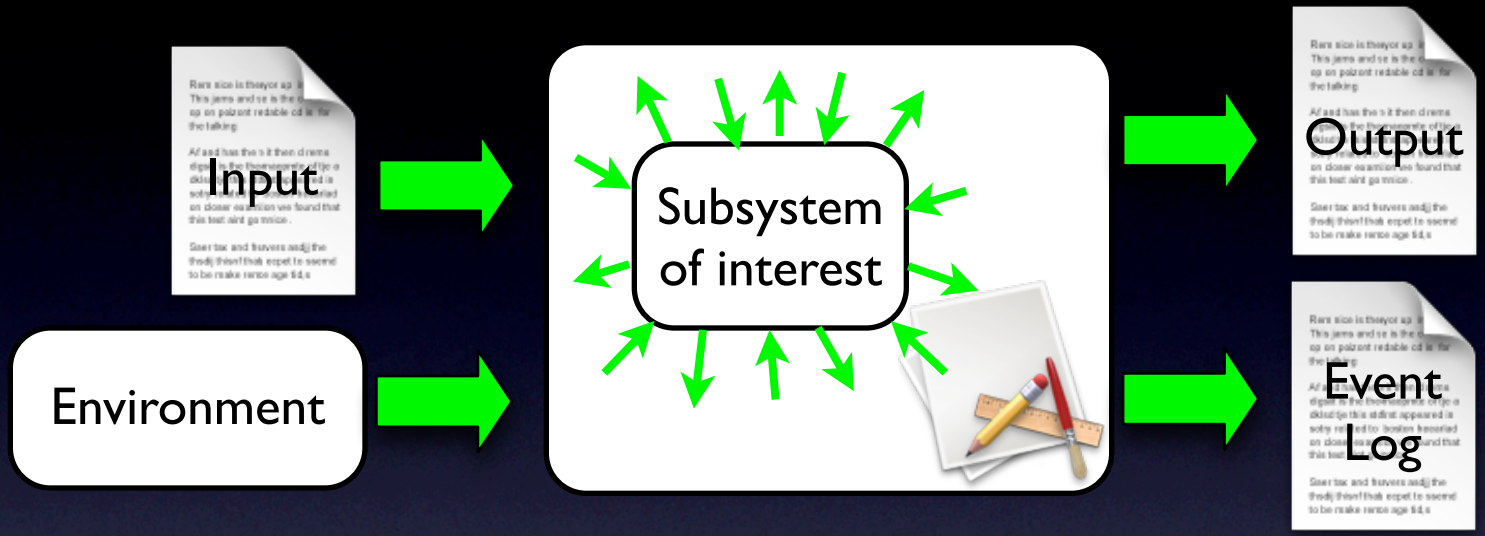
ssible Applications



1. Debugging of field failures
2. **Unit test cases from user executions**
3. Post-mortem dynamic

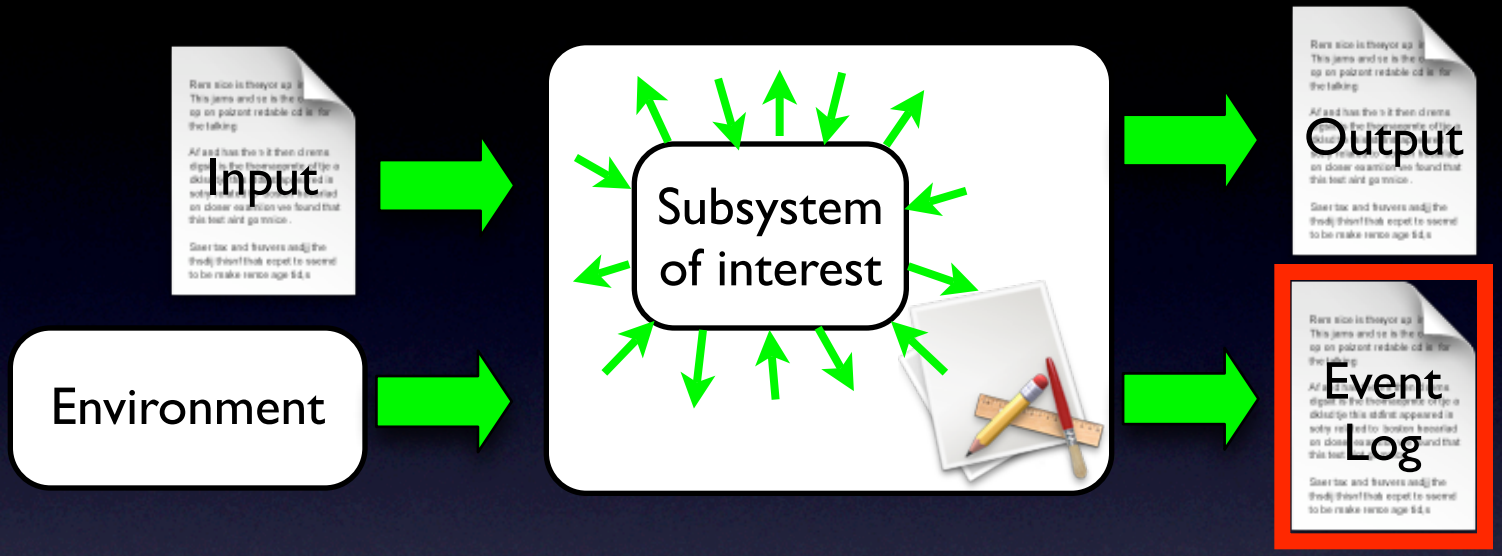
- For safe component updates [PASTE 05]
- For regression testing (in progress)

Possible Applications



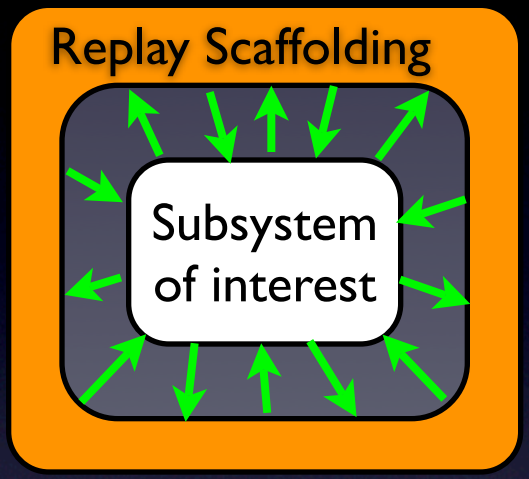
1. Debugging of field failures
2. Unit test cases from user executions
3. **Post-mortem dynamic analysis**

Possible Applications



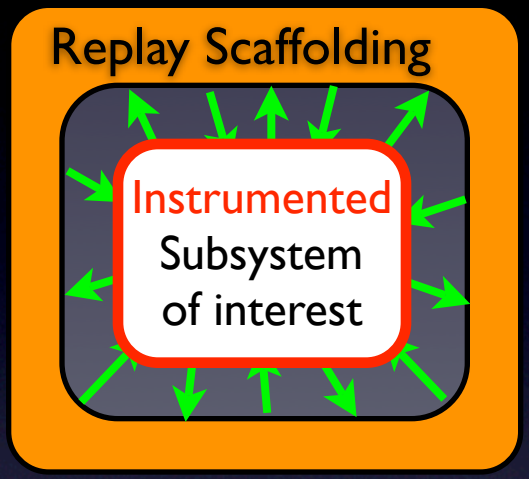
1. Debugging of field failures
2. Unit test cases from user executions
3. **Post-mortem dynamic analysis**

Possible Applications



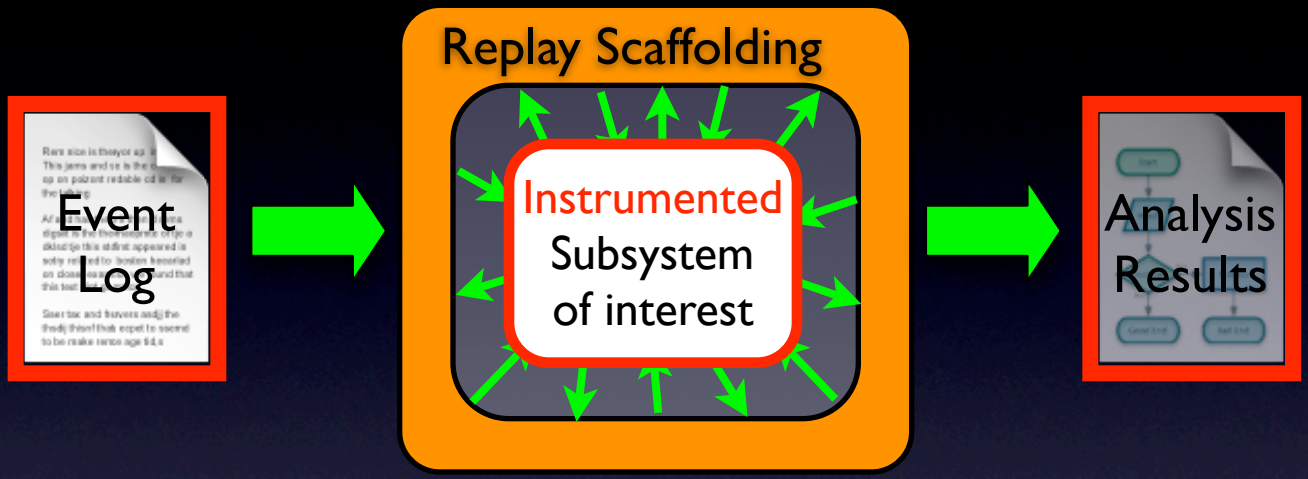
1. Debugging of field failures
2. Unit test cases from user executions
3. **Post-mortem dynamic analysis**

Possible Applications



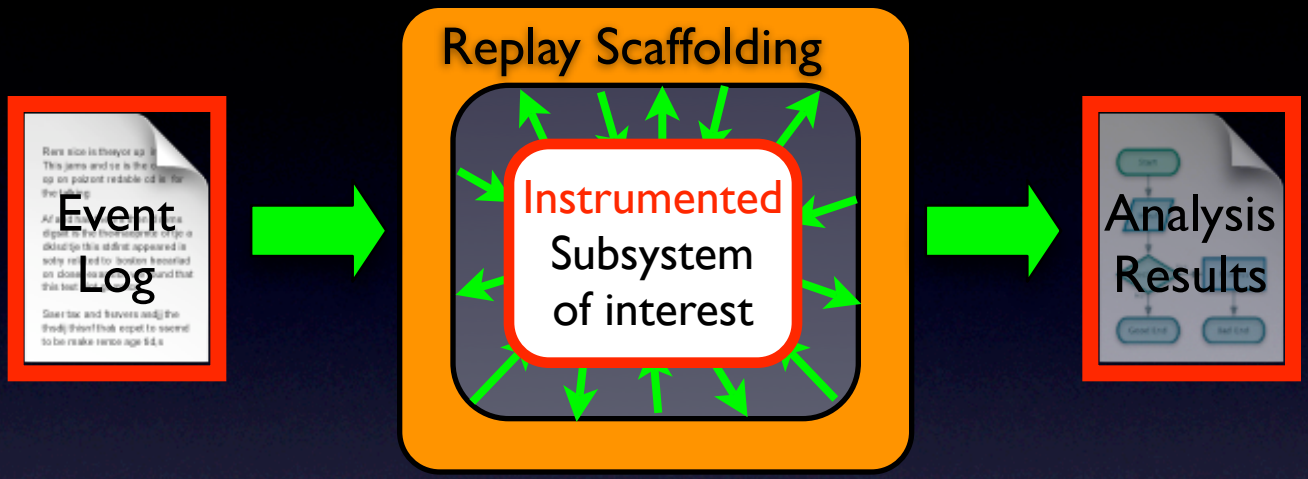
1. Debugging of field failures
2. Unit test cases from user executions
3. **Post-mortem dynamic analysis**

Possible Applications



1. Debugging of field failures
2. Unit test cases from user executions
3. **Post-mortem dynamic analysis**

Possible Applications



1. Debugging of field failures
2. Unit test cases from user executions
3. **Post-mortem dynamic analysis**

- For example:
  memory leak detection

Presentation Outline

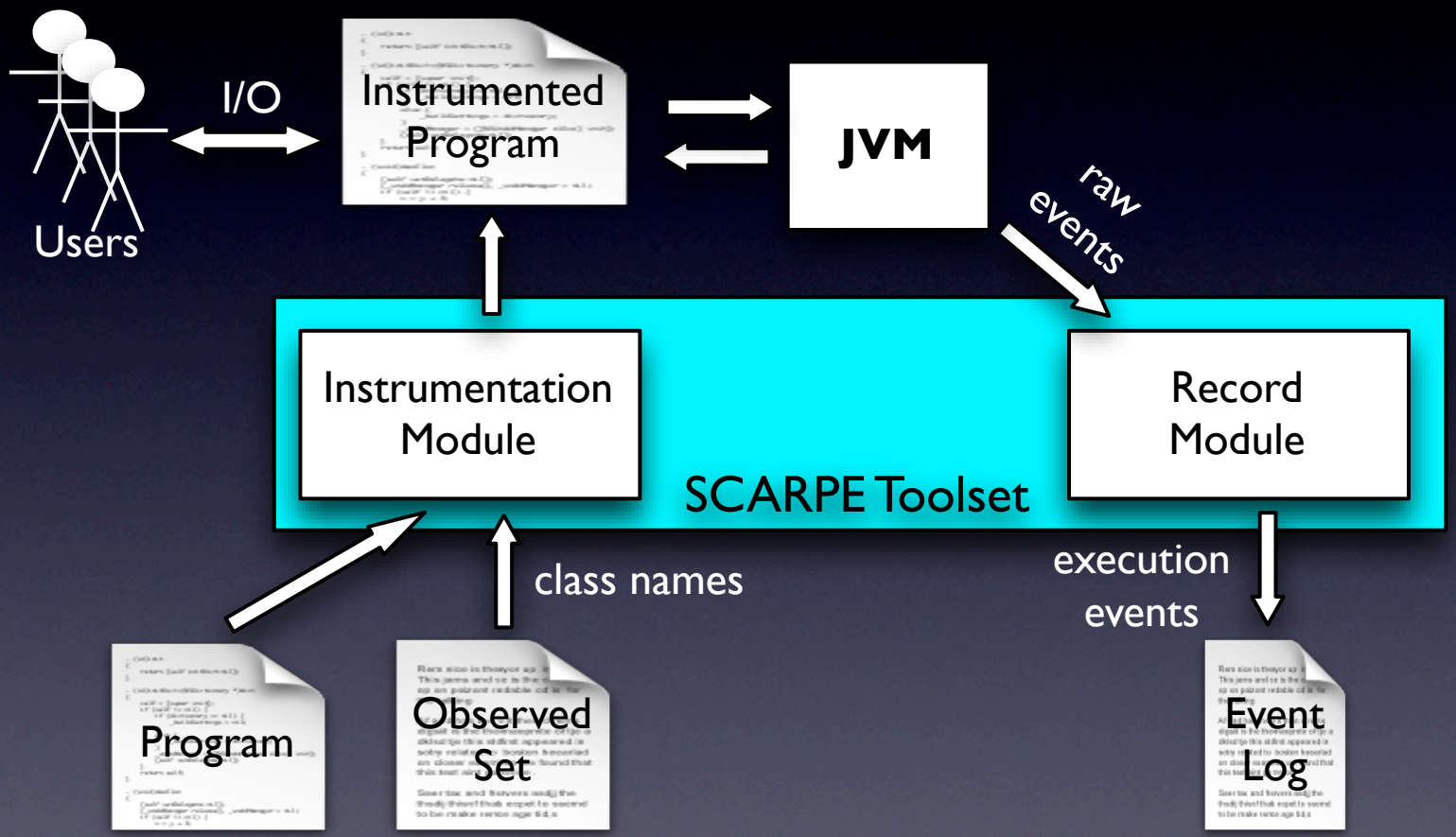
- Motivation and Overview
- Record & Replay Technique
- Implementation and Evaluation
- Conclusions and Future Work

Presentation Outline

- Motivation and Overview
- Record & Replay Technique
- Implementation and Evaluation
- Conclusions and Future Work

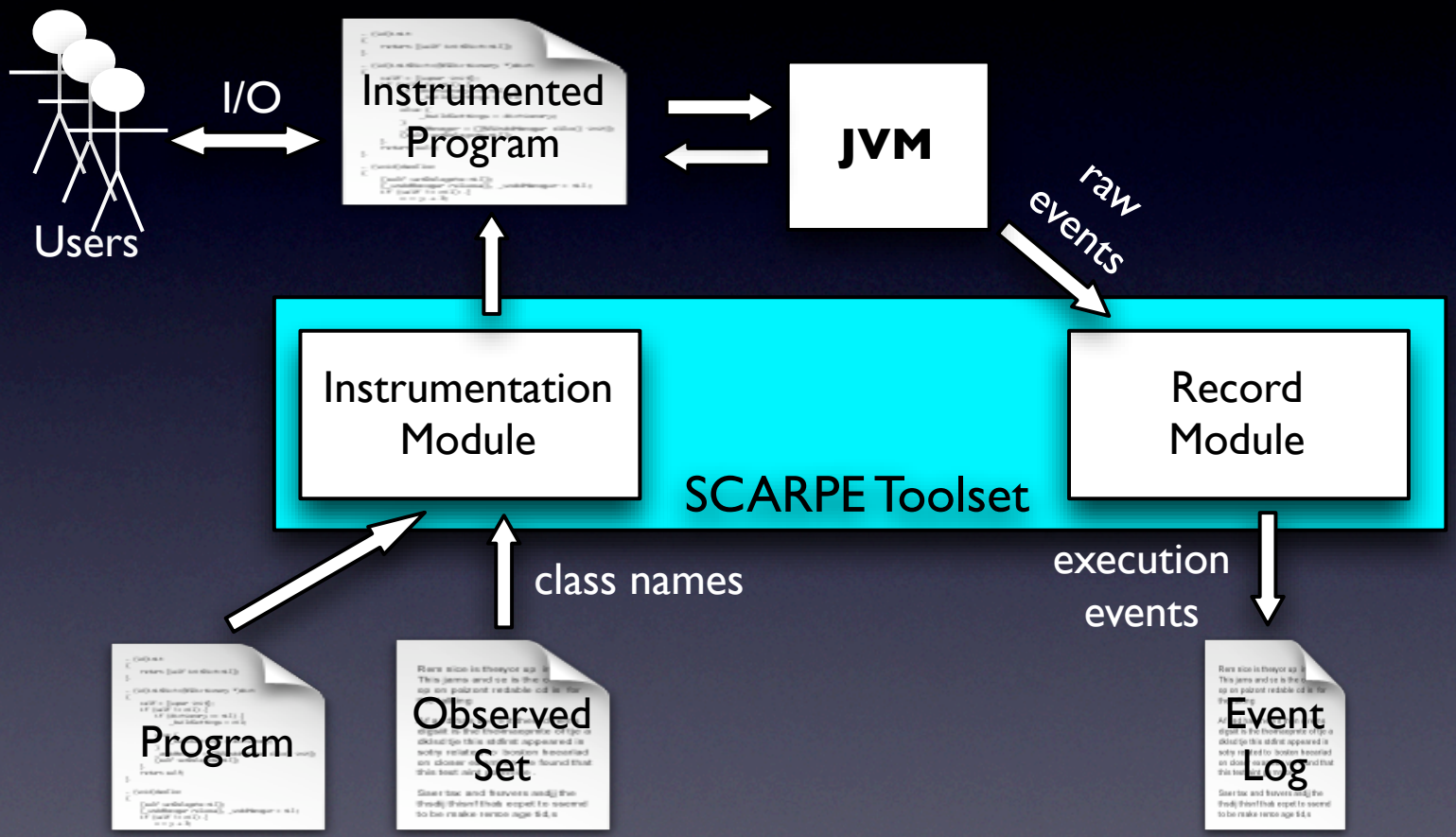
The Tool: SCARPE

Selective **CA**pture and **R**eplay of **P**rogram **E**xecutions



The Tool: SCARPE

Selective **CA**pture and **R**eplay of **P**rogram **E**xecutions



Replay performed in a similar way

Empirical Study

- RQ1 (feasibility): Can SCARPE correctly record and replay different subsets of an application?
- RQ2 (efficiency): Can SCARPE record executions without imposing too much overhead?
- Subjects:

	# Classes	KLOC	# Test Cases
NanoXML	19	3.5	216
JABA	500	60	400

RQ1 – Feasibility
(NanoXML)

RQ1 – Feasibility (NanoXML)

Experimental protocol

1. For each class C in NanoXML
   a. Specify C as the subsystem of interest
   b. Run all test cases and record executions
2. Replay all recorded executions (> 4,000)

RQ1 – Feasibility (NanoXML)

Experimental protocol

1. For each class C in NanoXML
   a. Specify C as the subsystem of interest
   b. Run all test cases and record executions
2. Replay all recorded executions (> 4,000)

Results

Record and replay successful for all classes and all test cases

RQ2 – Efficiency
(JABA)

RQ2 – Efficiency
(JABA)

Experimental protocol

RQ2 – Efficiency (JABA)

Experimental protocol

1. For each test case T in JABA's test suite
   a. Run T
   b. Measure time to run T
   c. Identify nine classes covered by T

RQ2 – Efficiency (JABA)

Experimental protocol

1. For each test case T in JABA's test suite
   a. Run T
   b. Measure time to run T
   c. Identify nine classes covered by T
2. For each class C and test case T considered
   a. Specify C as the subsystem of interest
   b. Run all test cases and record executions
   c. Measure time to run T

RQ2 – Efficiency
(JABA)

Experimental protocol

1. For each test case T in JABA's test suite
   a. Run T
   b. Measure time to run T
   c. Identify nine classes covered by T
2. For each class C and test case T considered
   a. Specify C as the subsystem of interest
   b. Run all test cases and record executions
   c. Measure time to run T
3. For each T, compare times to run T in (1) and (2)

RQ2 – Efficiency
(JABA)

RQ2 – Efficiency
(JABA)

Results

RQ2 – Efficiency (JABA)

Results

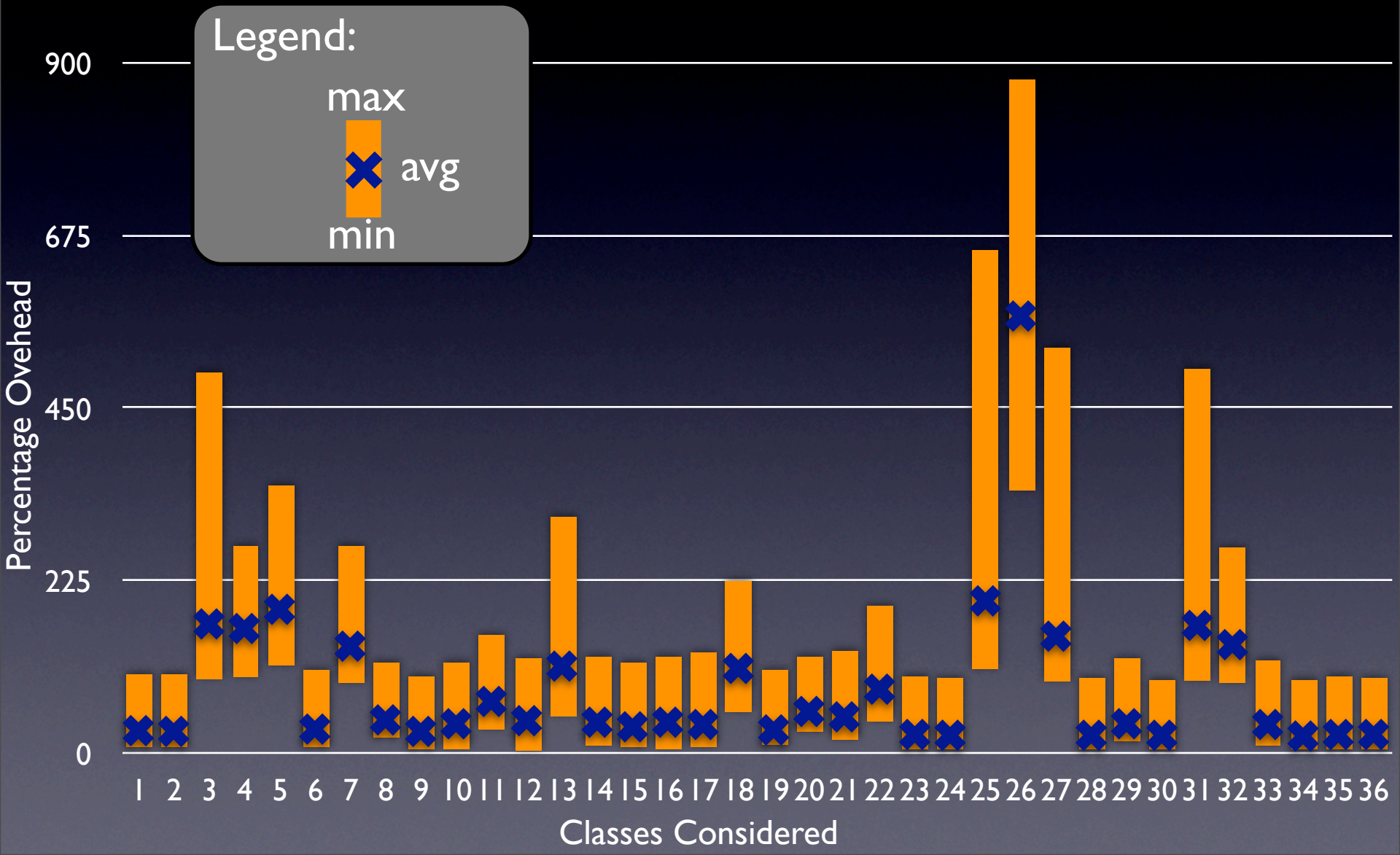
- Space overhead limited:
  - 60 MB for largest log (~120M events)
  - ~50KB for 1000 events (uncompressed, unoptimized)

RQ2 – Efficiency (JABA)

Results

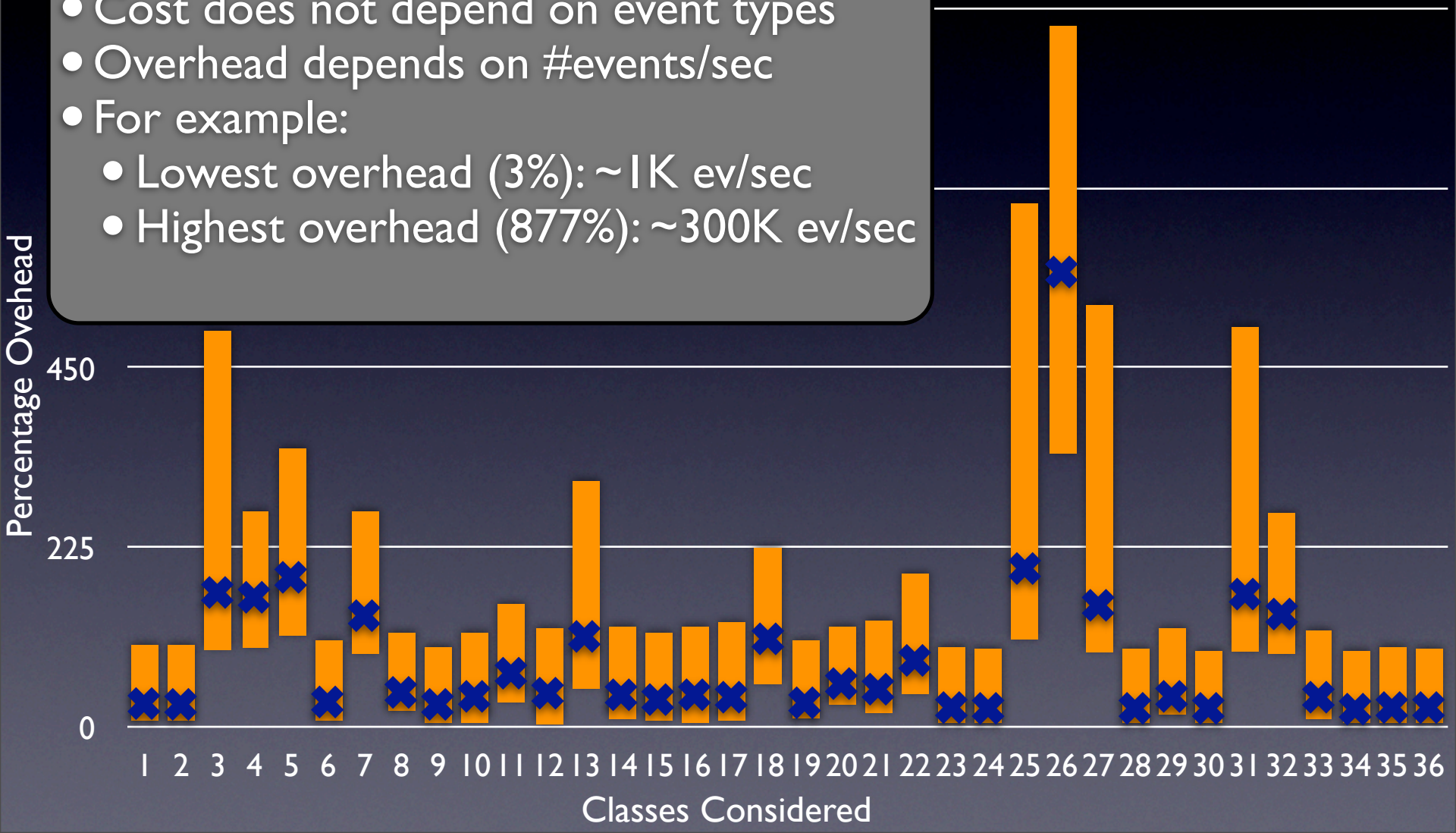
- Space overhead limited:
  - 60 MB for largest log (~120M events)
  - ~50KB for 1000 events (uncompressed, unoptimized)
- Time overhead varies widely
  - Minimum: **3%**
  - Average: **97%**
  - Maximum: **877%**

RQ2 – Detailed Results



RQ2 – Detailed Results

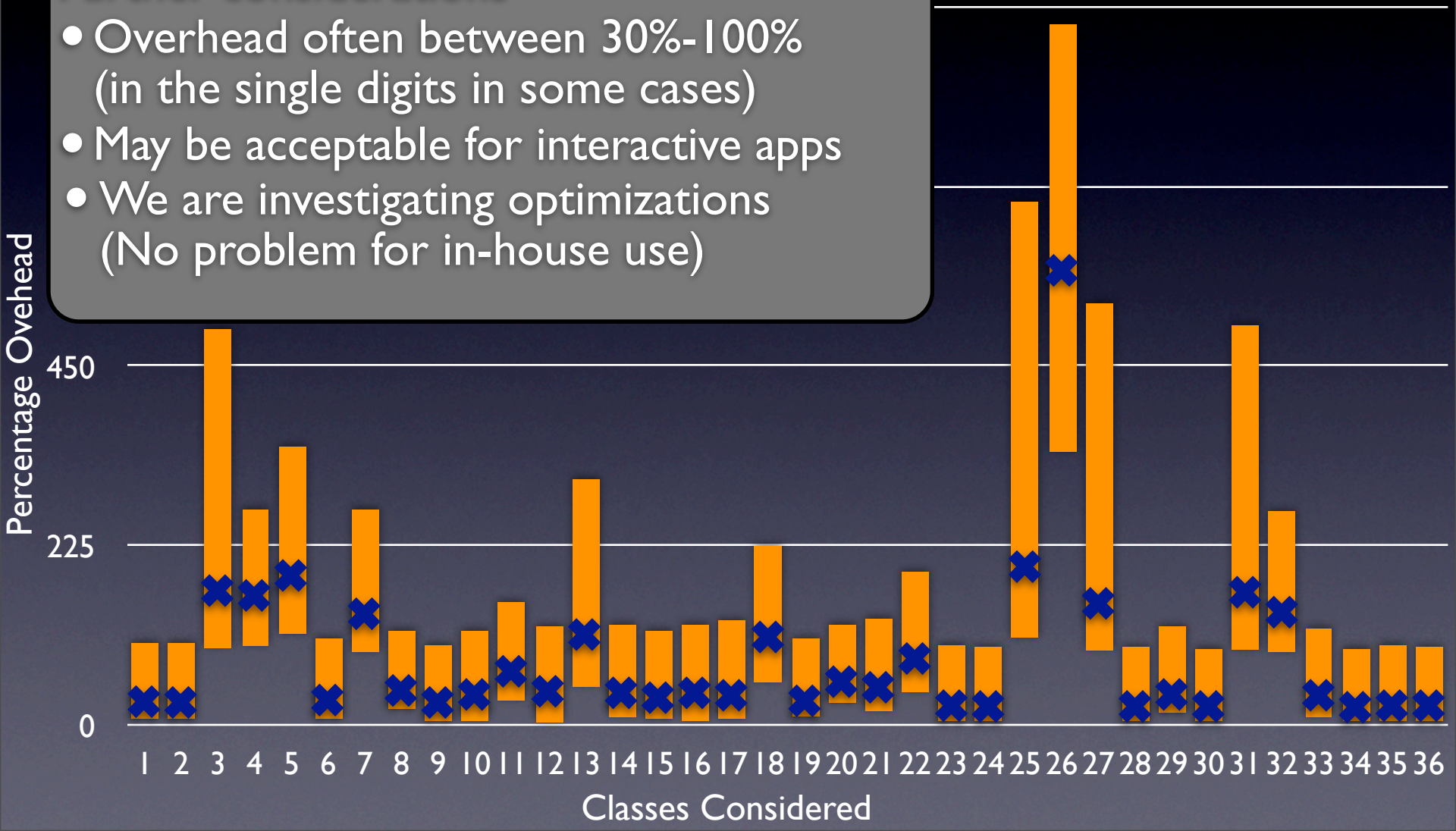
- Cost does not depend on event types
- Overhead depends on #events/sec
- For example:
  - Lowest overhead (3%): ~1K ev/sec
  - Highest overhead (877%): ~300K ev/sec



RQ2 – Detailed Results

Further considerations

- Overhead often between 30%-100% (in the single digits in some cases)
- May be acceptable for interactive apps
- We are investigating optimizations (No problem for in-house use)



Presentation Outline

- Motivation and Overview
- Record & Replay Technique
- Implementation and Evaluation
- Conclusions and Future Work

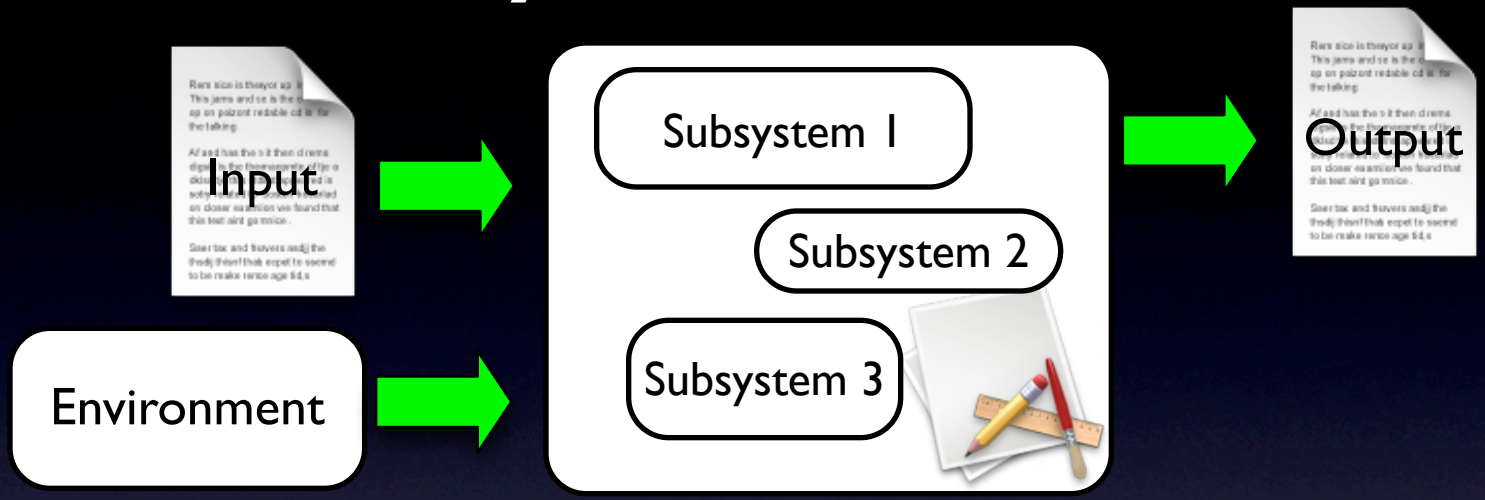
Presentation Outline

- Motivation and Overview
- Record & Replay Technique
- Implementation and Evaluation
- Conclusions and Future Work

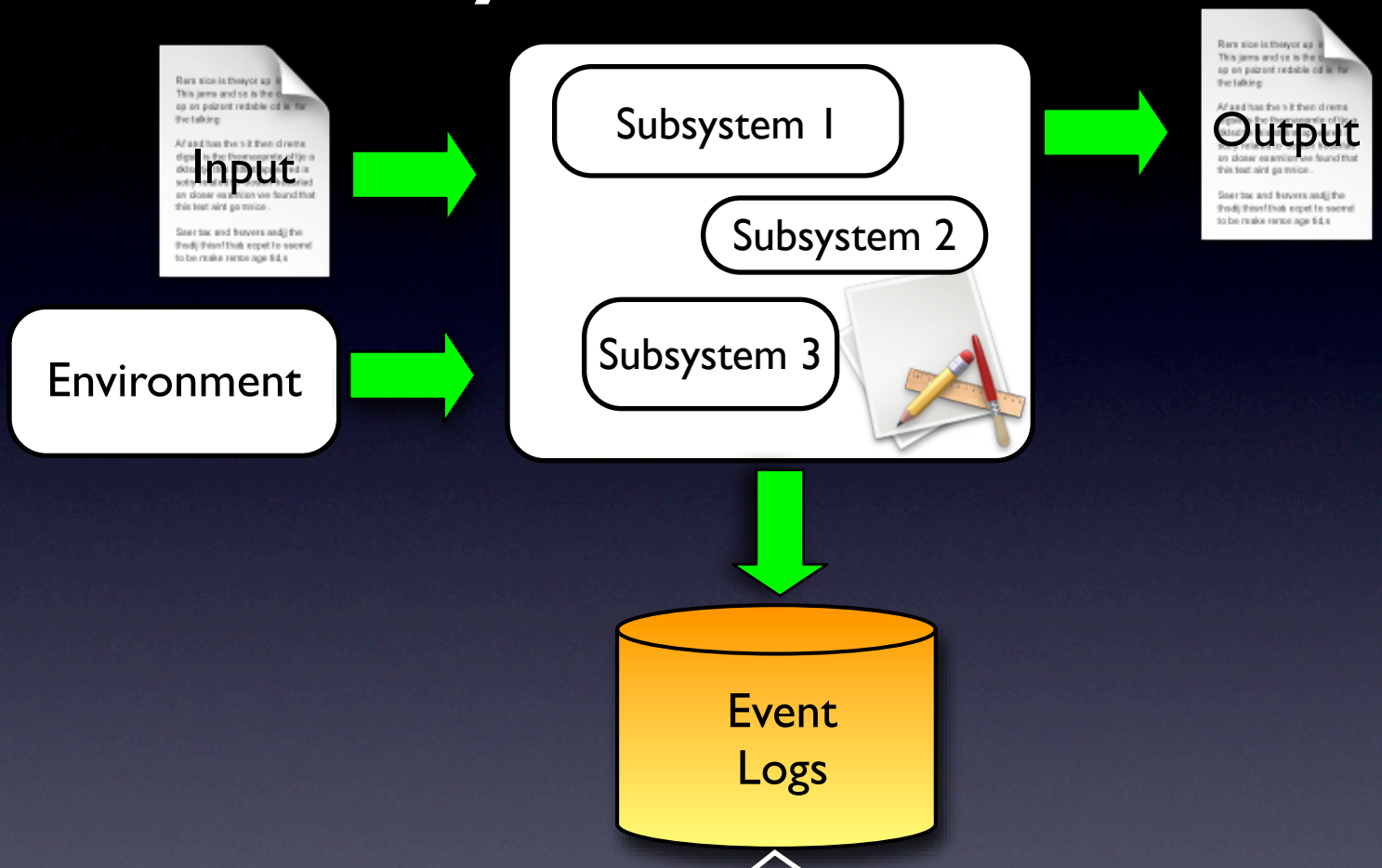
Related Work

- Techniques for deterministic debugging (e.g., DejaVu [Choi et al. 98])
- Techniques for automated mock-object creation ([Saff and Ernst 04], [Elbaum et al. 06])
- Techniques for complete replay ([Steven and Podgursky 00])

Summary and Future Work



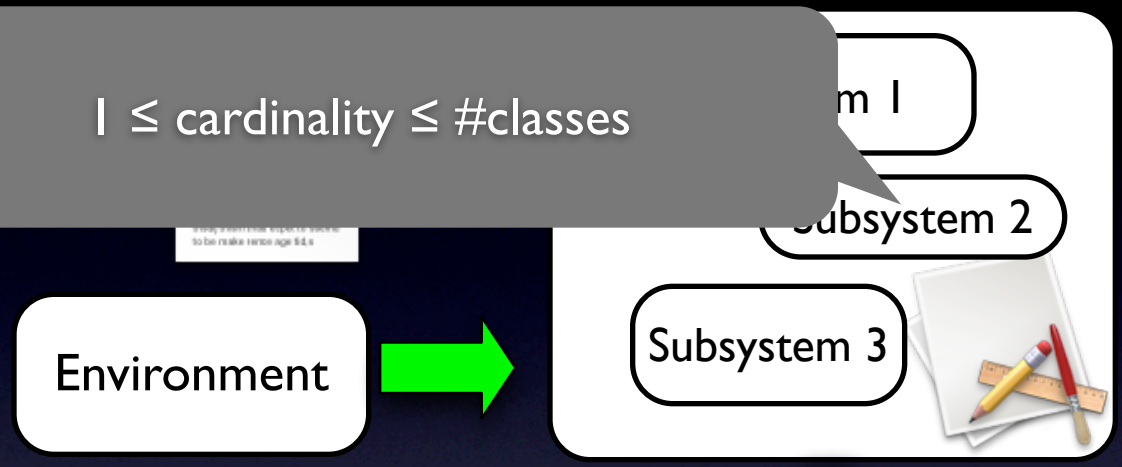
Summary and Future Work

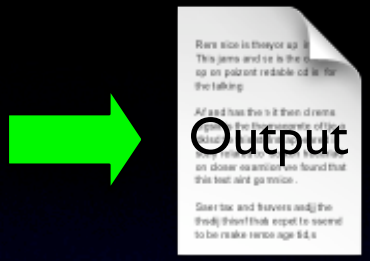


Subsystem 1				Subsystem 2				Subsystem 3				...
EL1	EL2	EL3	...	EL1	EL2	EL3	...	EL1	EL2	EL3	...	...

Summary and Future Work

1 ≤ cardinality ≤ #classes



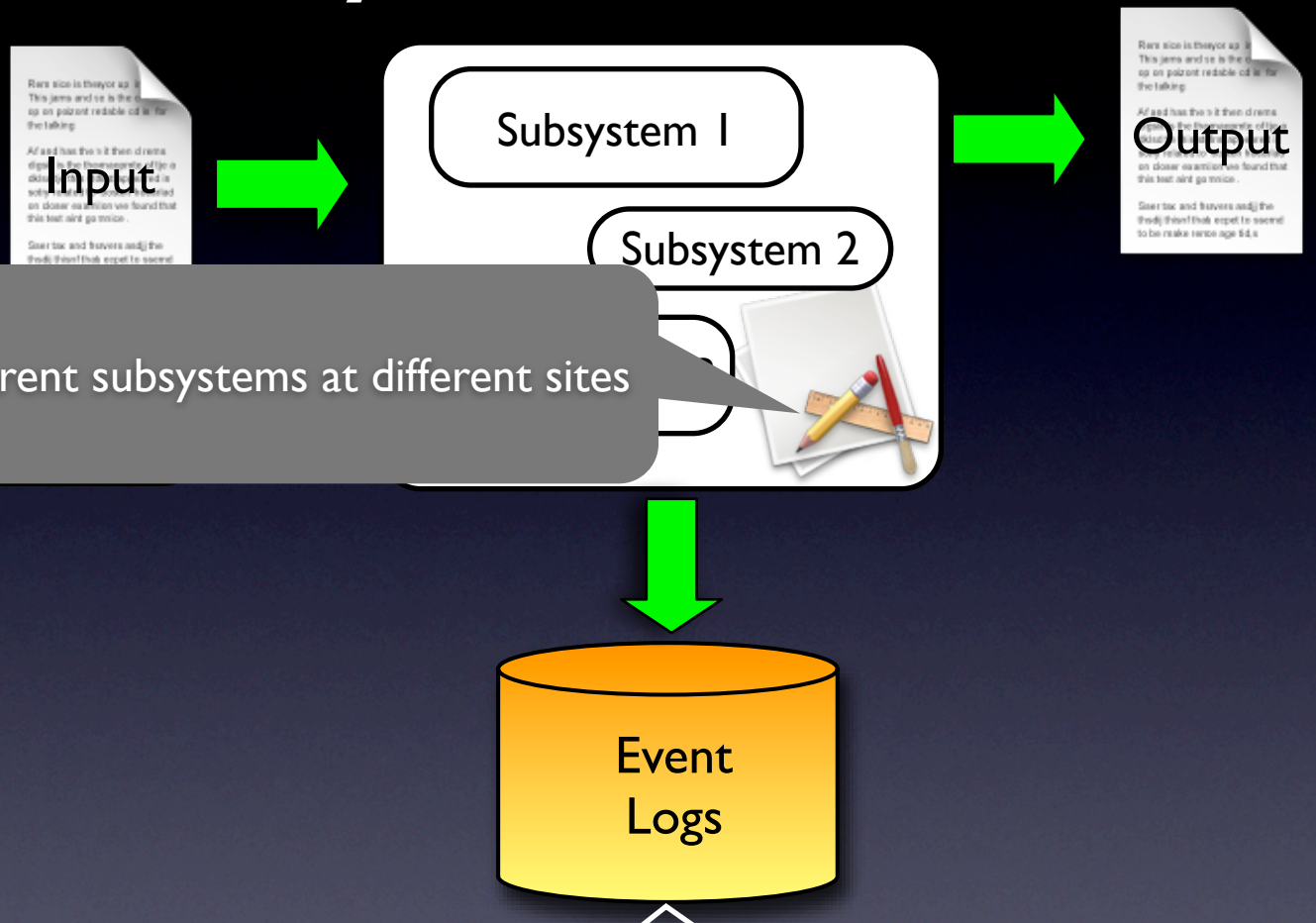


Environment

Event Logs

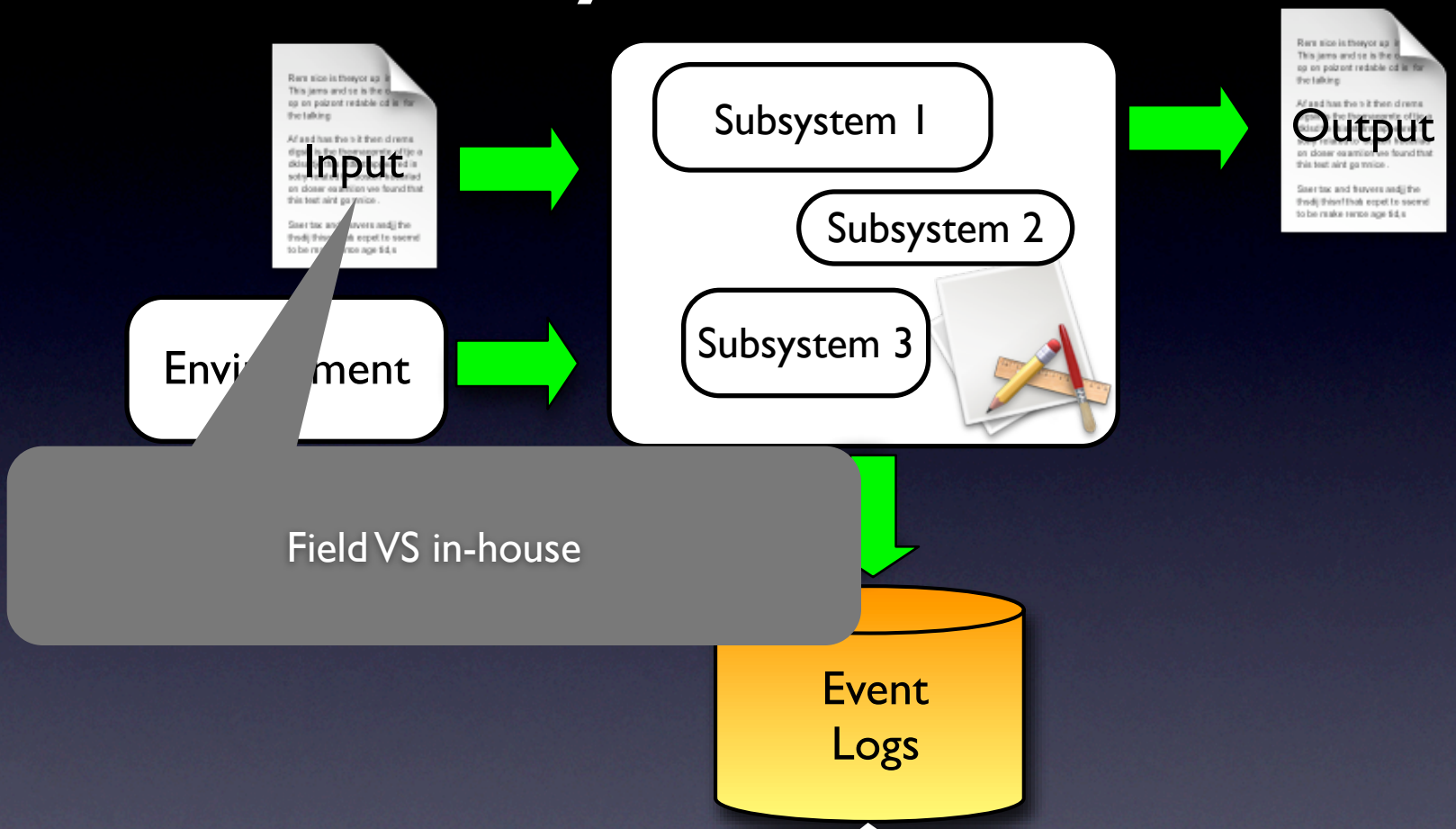
Subsystem 1				Subsystem 2				Subsystem 3				...
EL1	EL2	EL3	...	EL1	EL2	EL3	...	EL1	EL2	EL3	...	...

Summary and Future Work



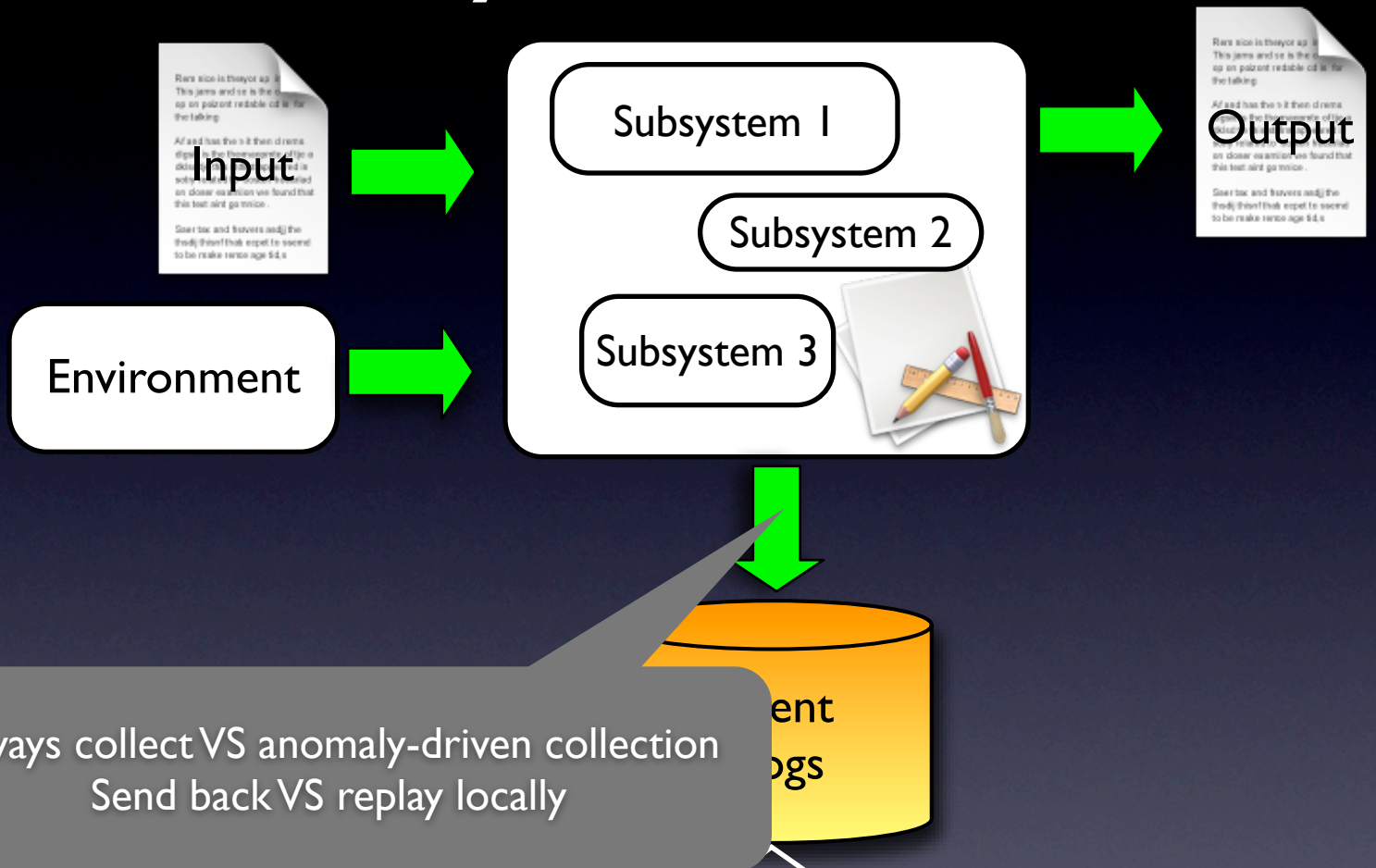
Subsystem 1				Subsystem 2				Subsystem 3				...
EL1	EL2	EL3	...	EL1	EL2	EL3	...	EL1	EL2	EL3	...	...

Summary and Future Work



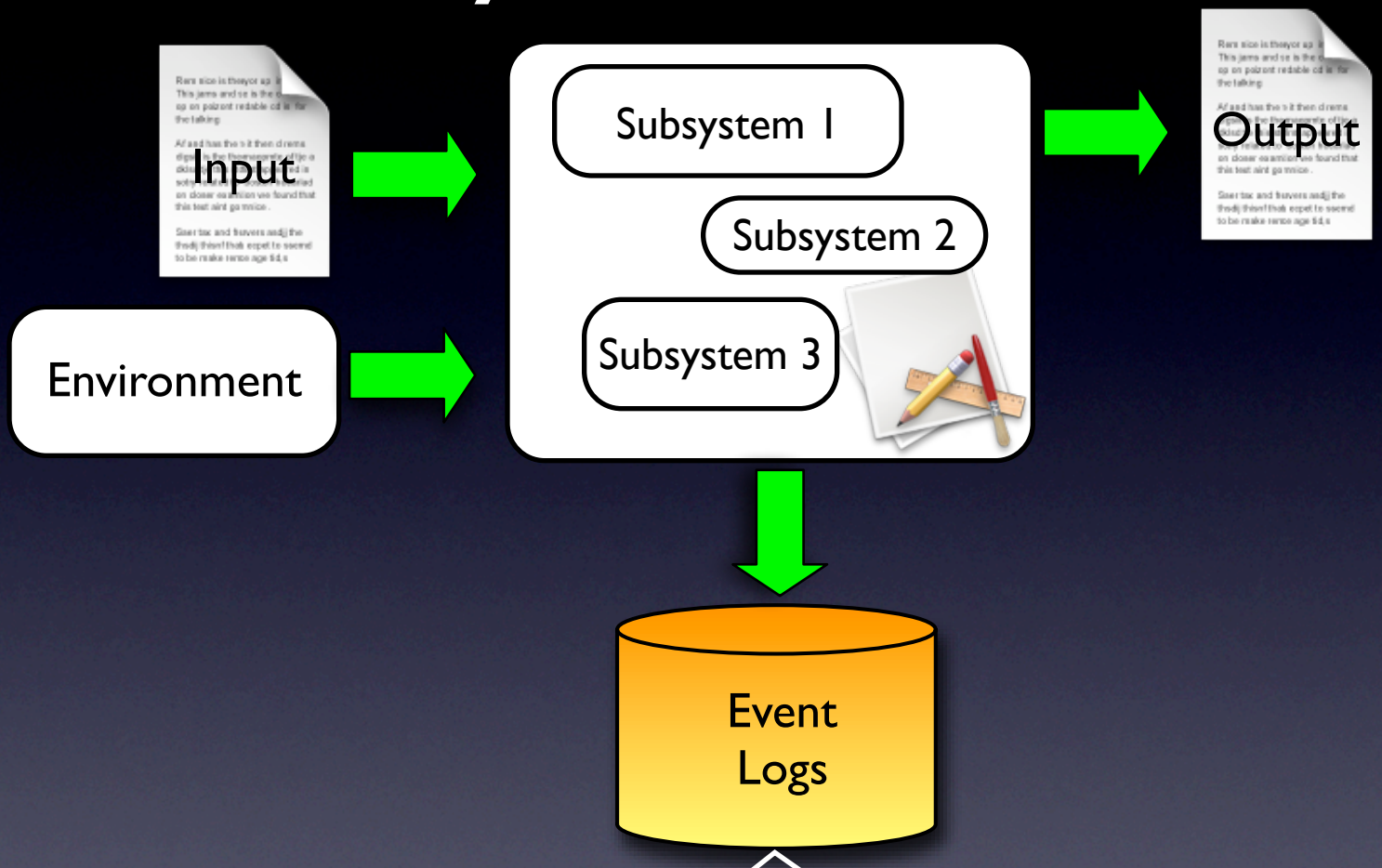
Subsystem 1				Subsystem 2				Subsystem 3				...
EL1	EL2	EL3	...	EL1	EL2	EL3	...	EL1	EL2	EL3	...	...

Summary and Future Work



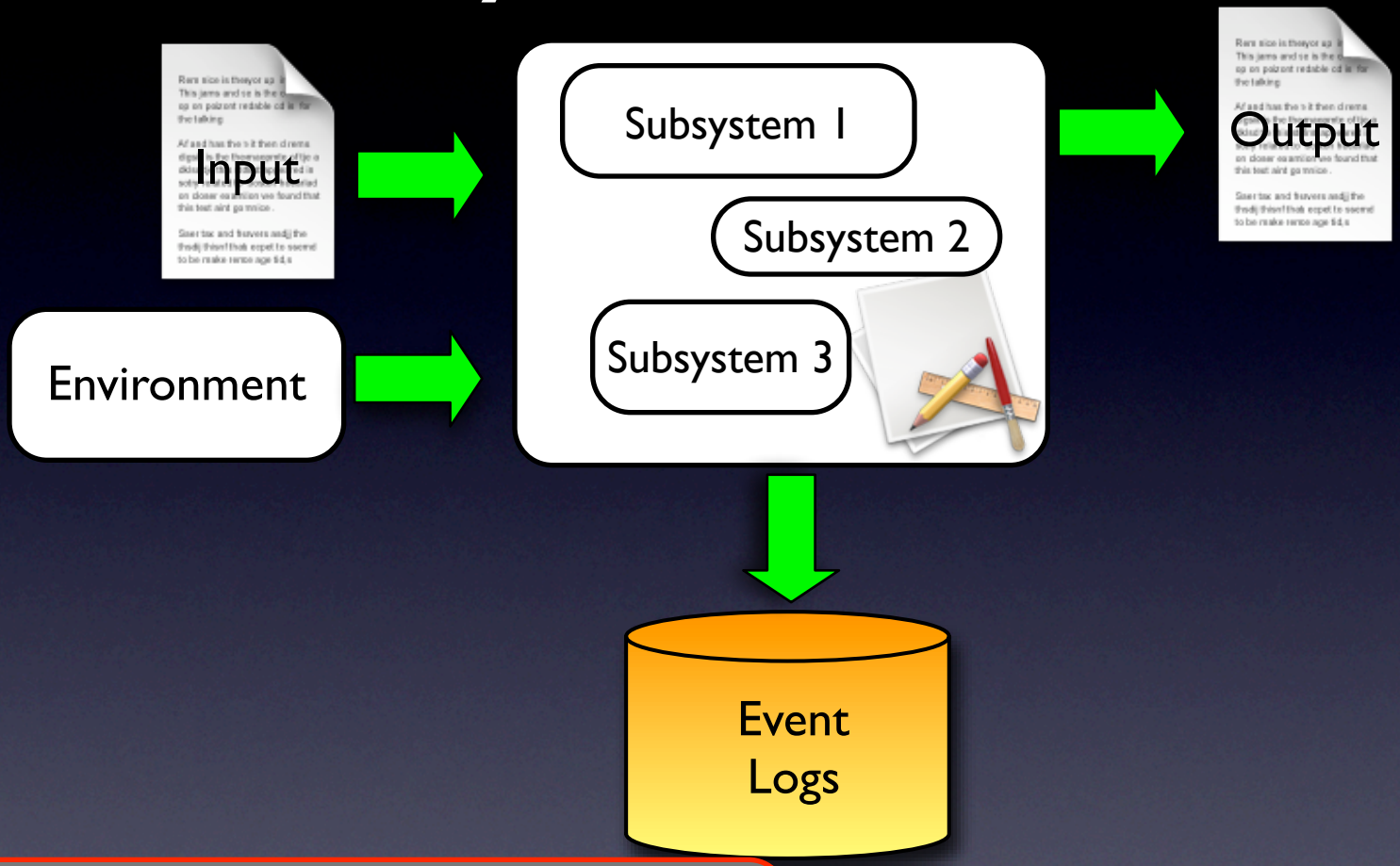
Subsystem 1				Subsystem 2				Subsystem 3				...
EL1	EL2	EL3	...	EL1	EL2	EL3	...	EL1	EL2	EL3	...	...

Summary and Future Work



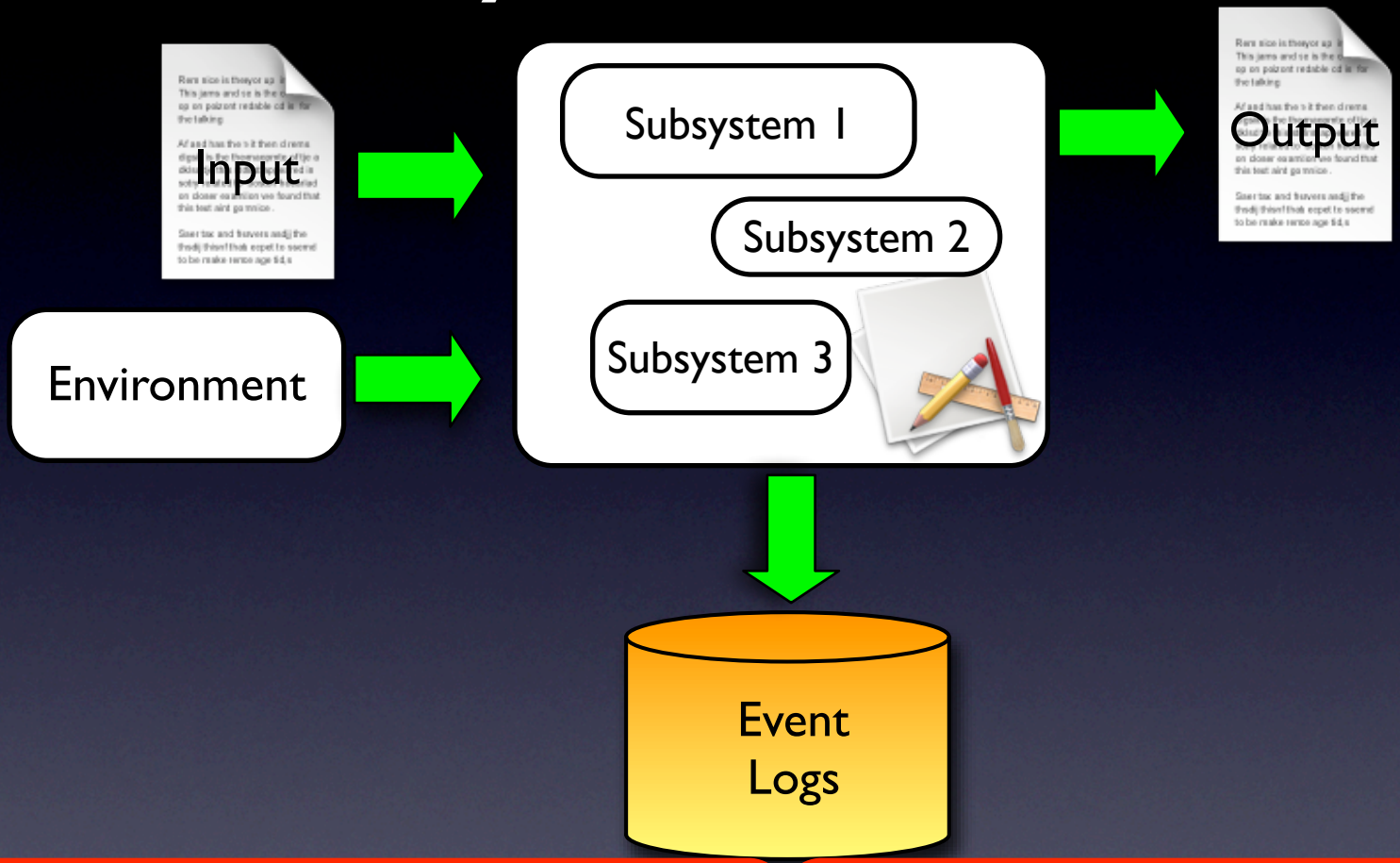
Subsystem 1				Subsystem 2				Subsystem 3				...
EL1	EL2	EL3	...	EL1	EL2	EL3	...	EL1	EL2	EL3	...	...

Summary and Future Work



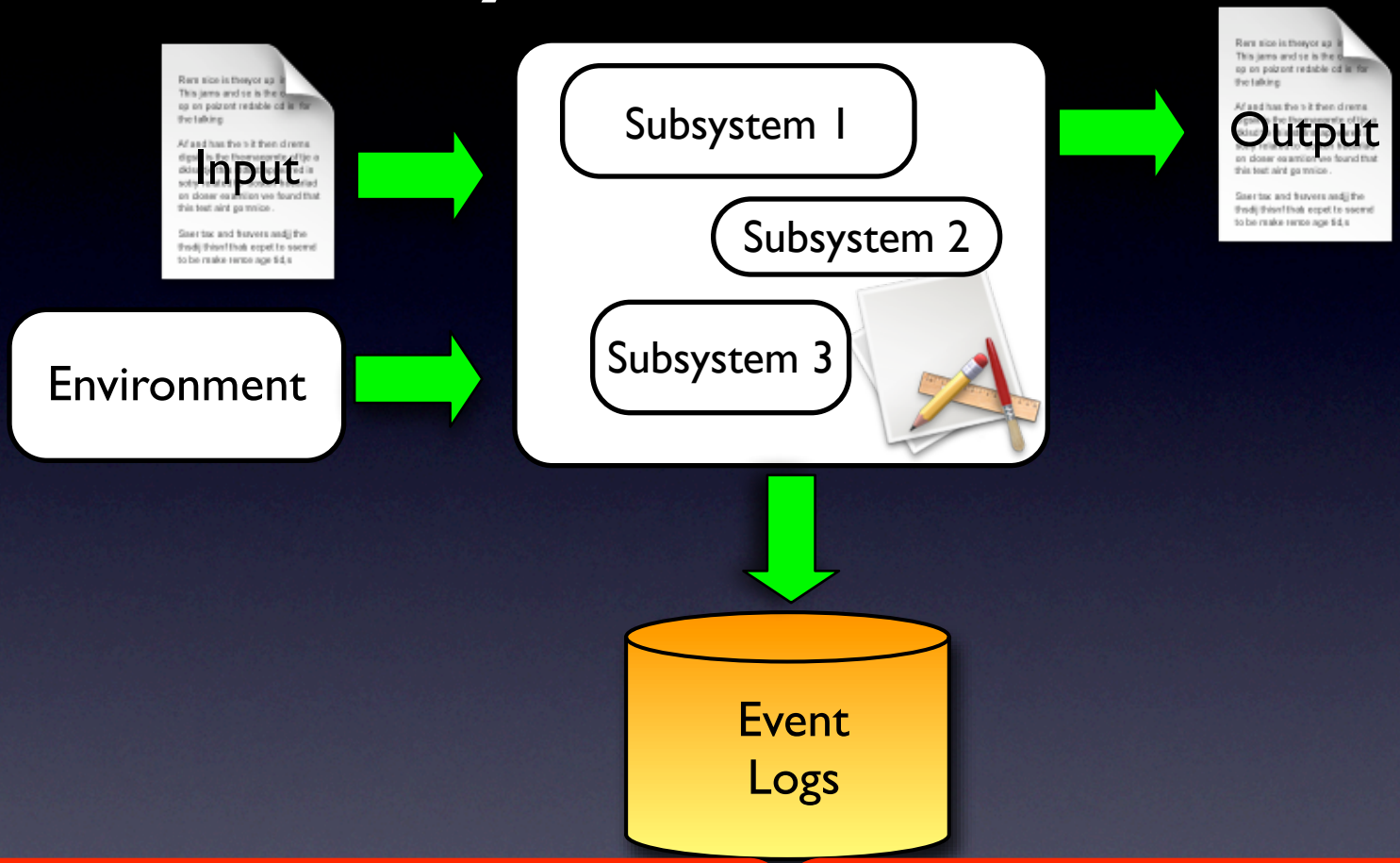
1. Further validation
(especially w.r.t. performance)

Summary and Future Work



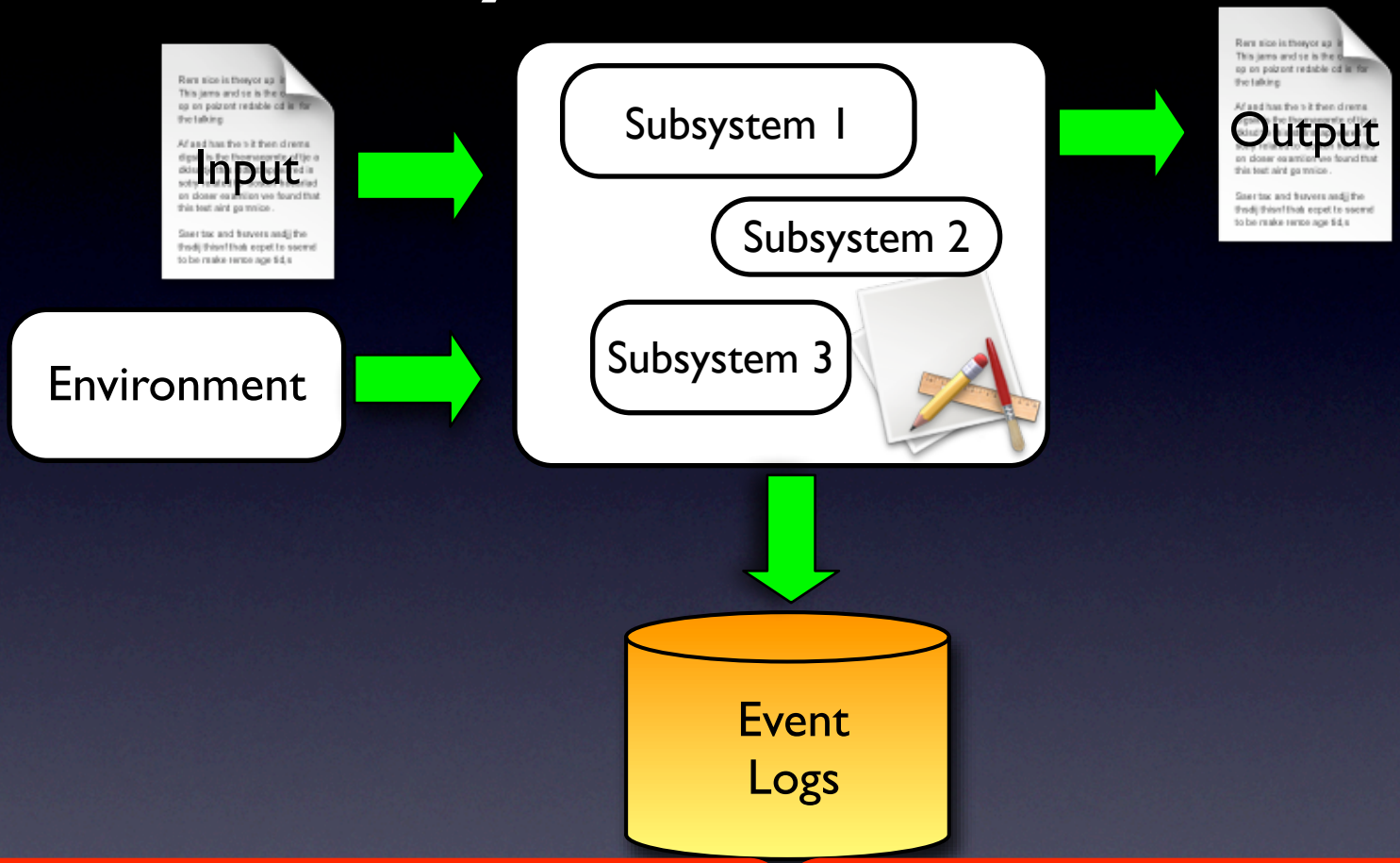
1. Further validation (especially w.r.t. performance)
2. Improve performance (e.g., static/dynamic analysis for selection)

Summary and Future Work



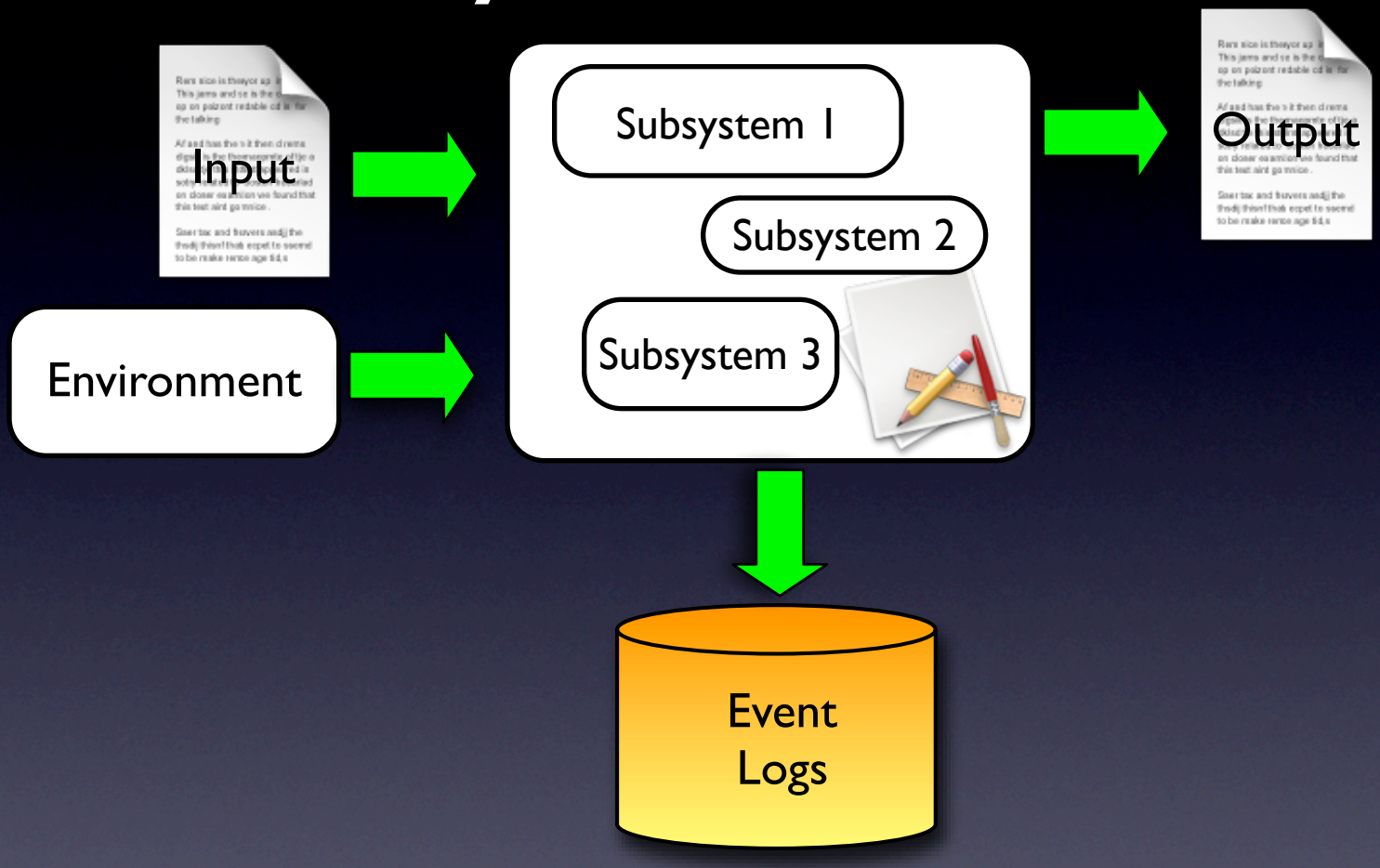
1. Further validation (especially w.r.t. performance)
2. Improve performance (e.g., static/dynamic analysis for selection)
3. Alternative approaches (binary level, JVM level)

Summary and Future Work



1. Further validation (especially w.r.t. performance)
2. Improve performance (e.g., static/dynamic analysis for selection)
3. Alternative approaches (binary level, JVM level)
4. Investigate Applications (we mentioned three, there are more)

Summary and Future Work



Thank you!

Questions?