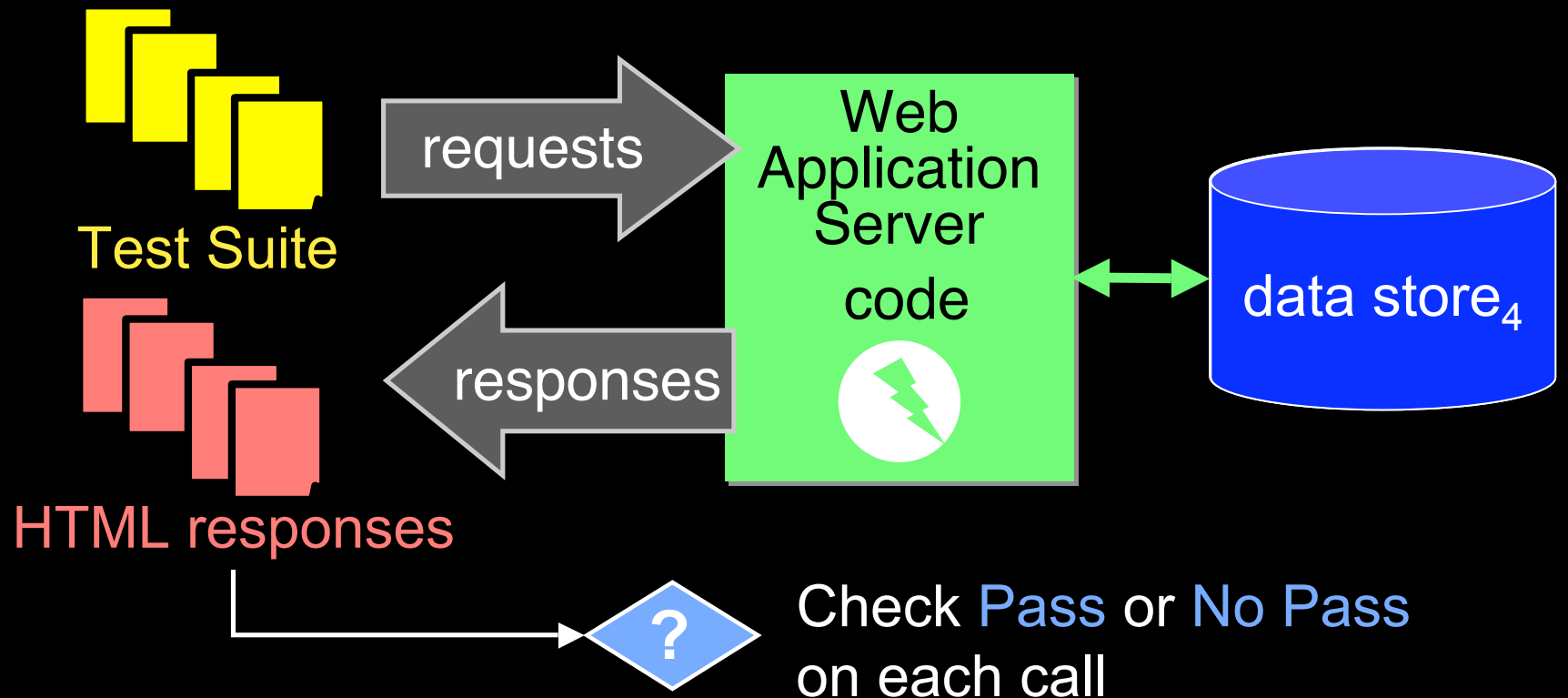
Automated Replay and Failure Detection for Web Applications

Sara Sprenkle, Emily Gibson,
Sreedevi Sampath, and Lori Pollock

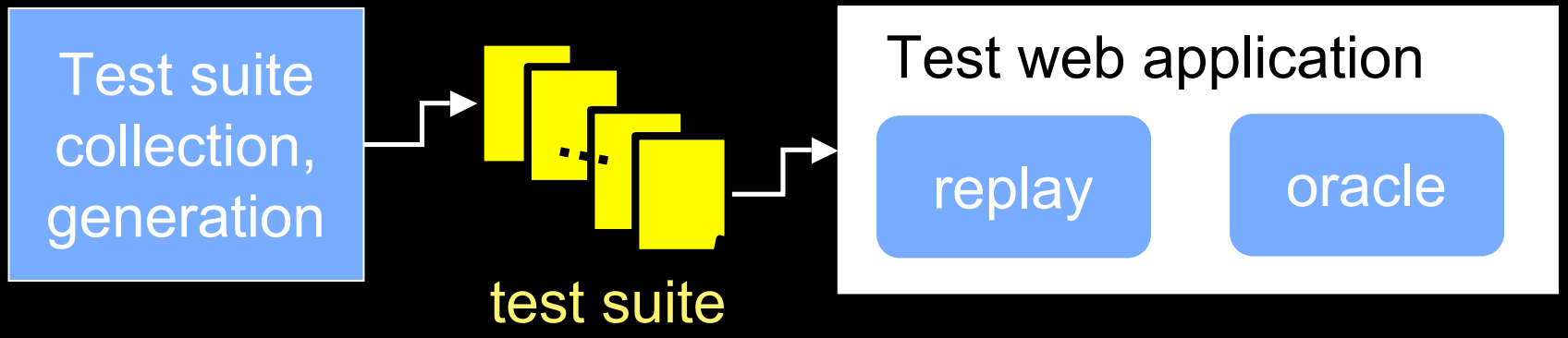
University of Delaware

Web Application Testing Process



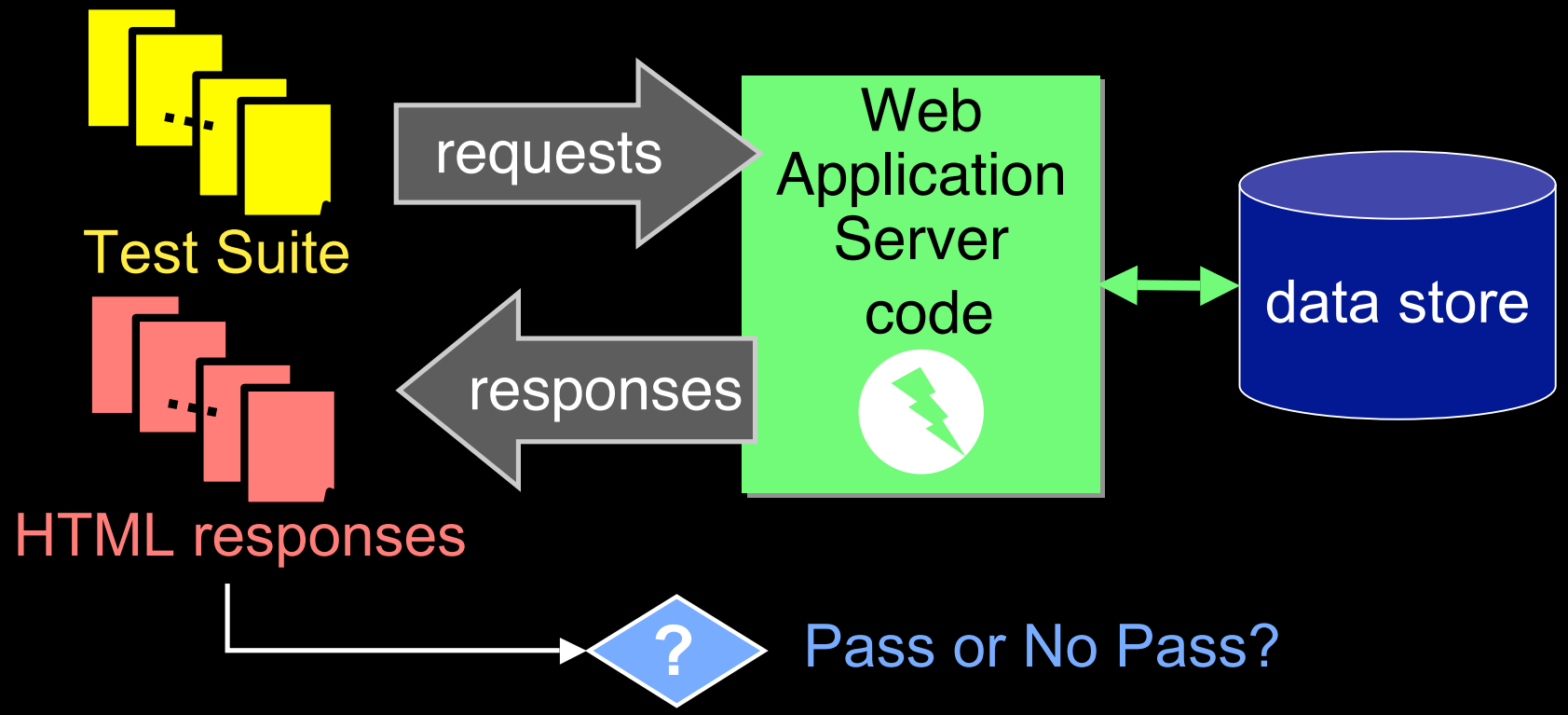
Motivates need for *automated* testing framework

Automated Framework Overview



- User-session-based testing
- Design Goals
  - Integrated components
  - General: test any web application
  - Flexible: customizable components
  - Portable: platform-independent test information

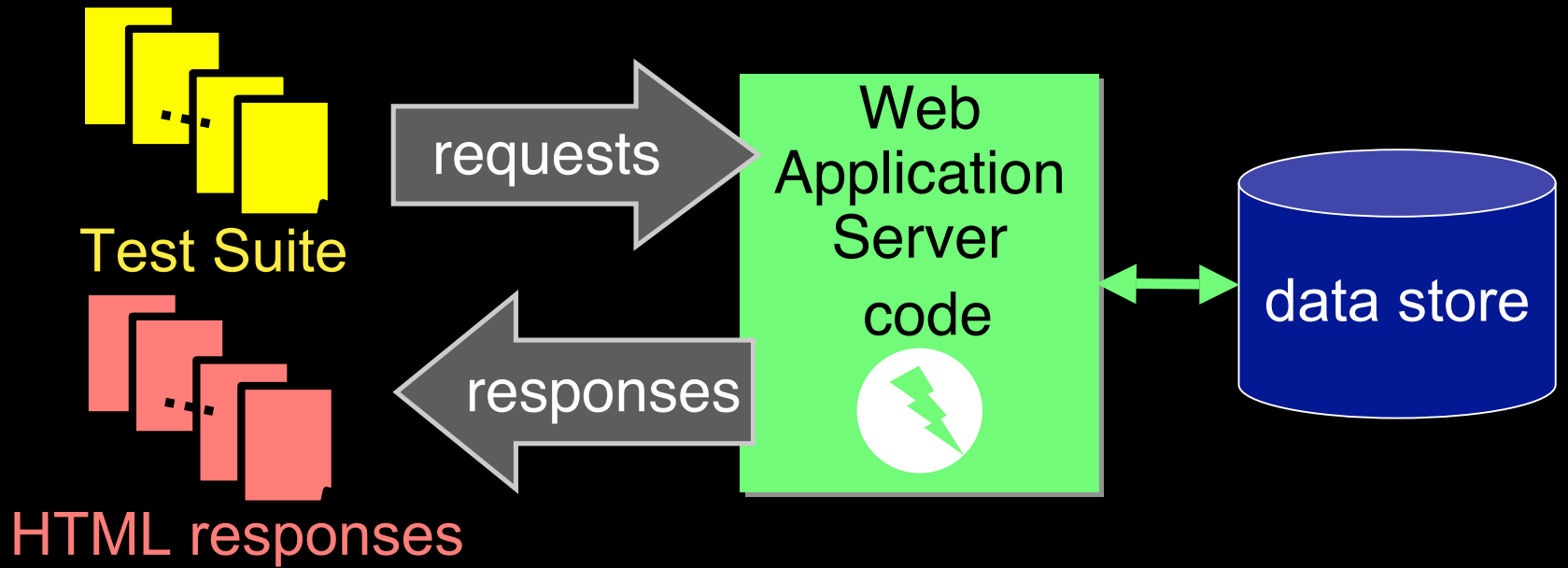
Web Application Testing Challenges



Oracle:
Is the application behavior correct?

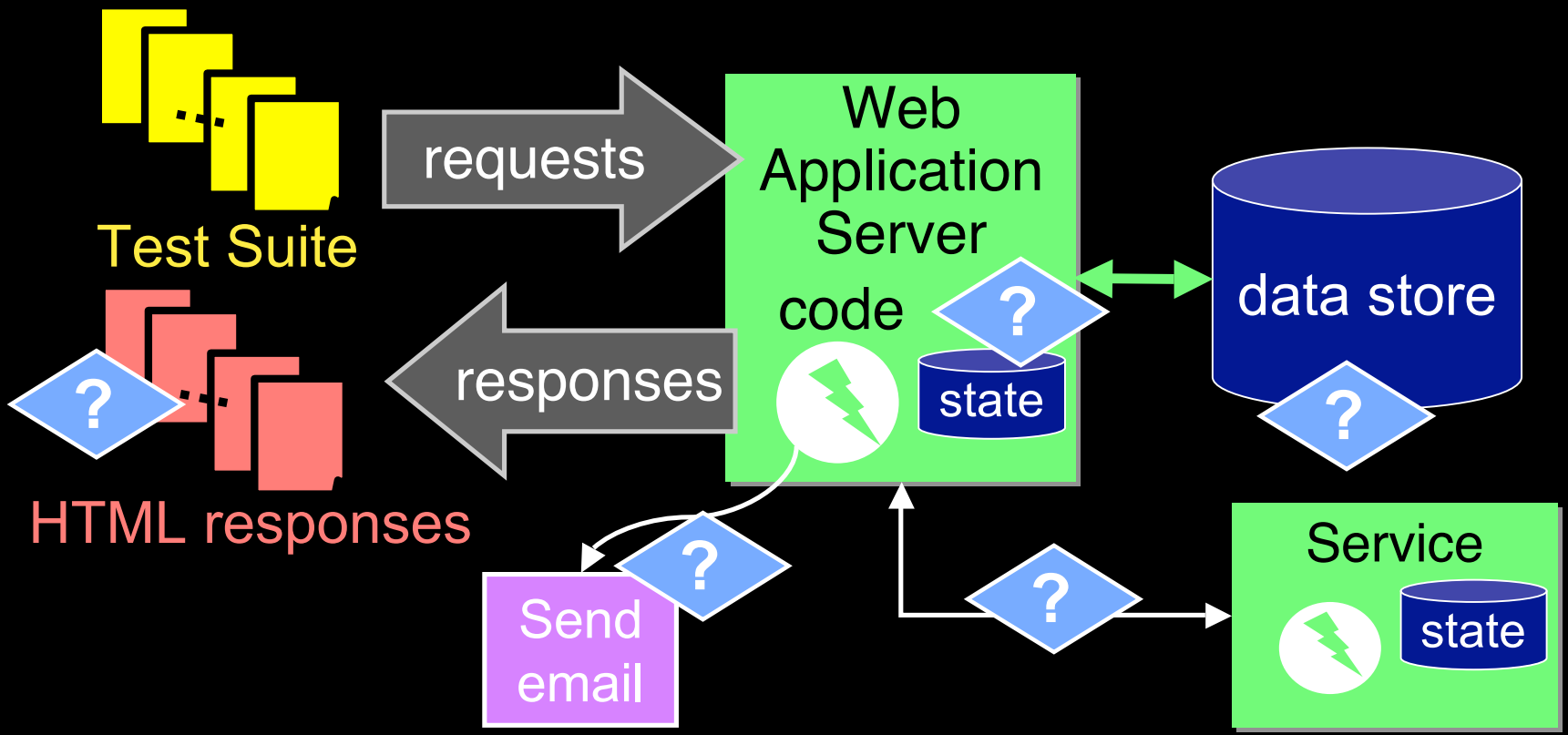
Replay:
How to control a persistent data store?

Challenge: Controlling Persistent State



- Difficulty of Controllability
- Dependence on replayed requests
  - e.g., order of replayed requests

Challenge: Application Correctness



- Low observability of outputs

Overview of Contributions

- Propose different combinations of

  replay techniques and *oracles*

  to execute different code, expose different faults
- Empirical study of techniques' effects
- Recommendations to testers

Replay Parameters

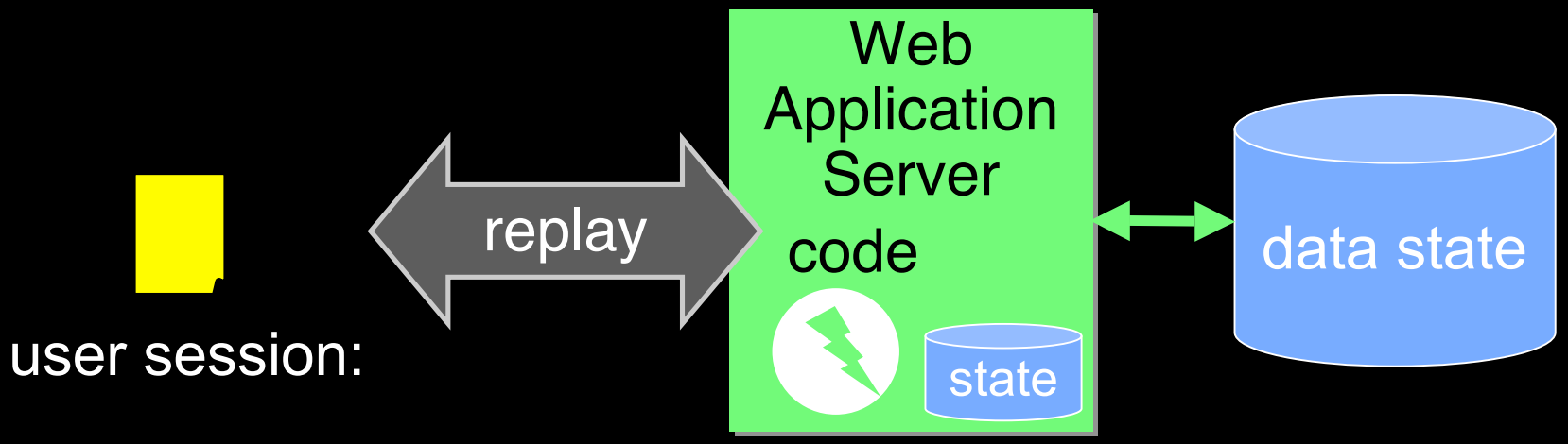
- Selected user sessions
  - Original, collected sessions
  - Reduced: selected subset by a heuristic
- Replay timing
  - Serially
  - Concurrently
- Replay order
  - Log-recorded order
  - Another established order
- Restoration of persistent state

Replay Parameters

- Selected user sessions
  - ✓ Original, collected sessions
  - ✓ Reduced: selected subset by a heuristic
- Replay timing
  - ✓ Serially
  - Concurrently
- Replay order
  - ✓ Log-recorded order
  - Another established order
- Restoration of persistent state

Replay Issue: Persistent State

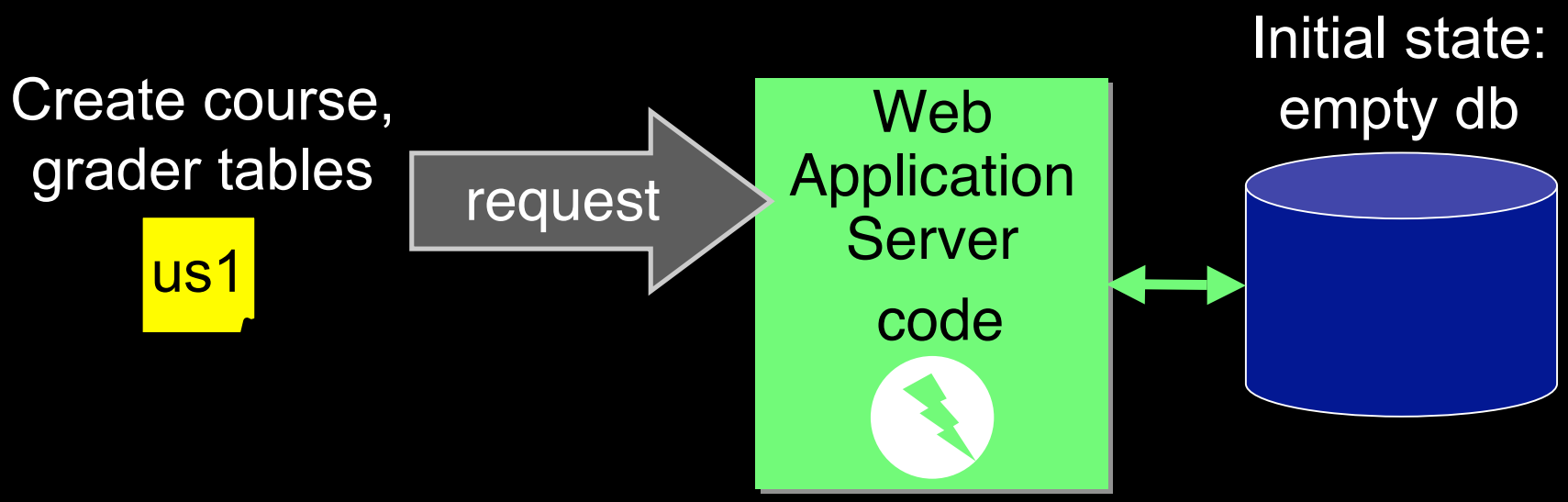
- Web application test case input
  - Explicit: name/value parameters
  - Implicit: data, server state



GraderLogin?name=grader&pswd=PASSWORD
ViewGrades?course=CISC105

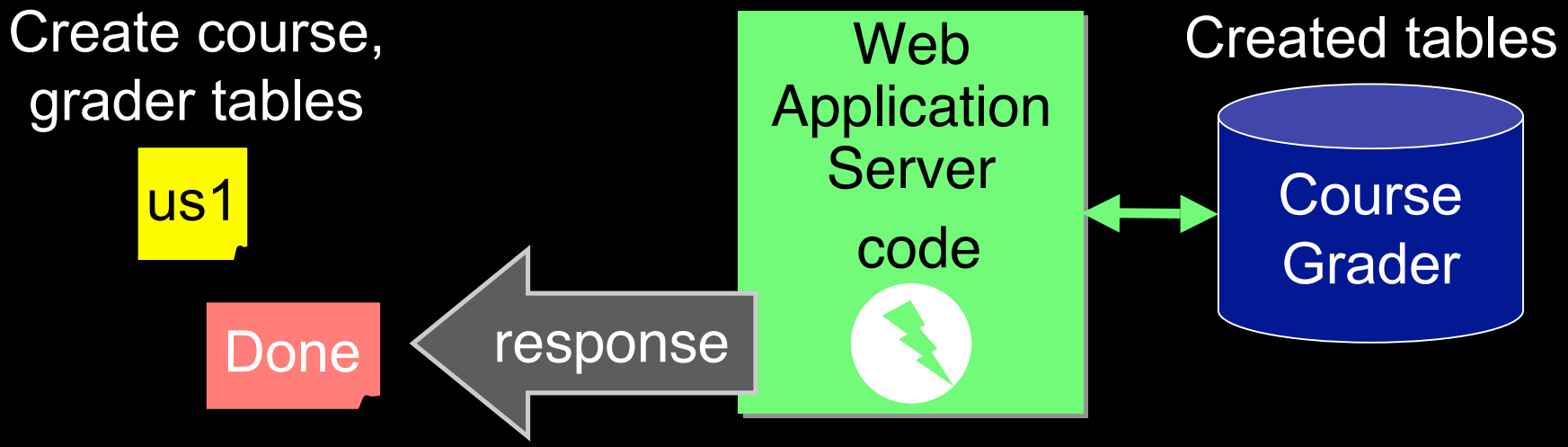
Replaying Original Suite

Replaying user sessions 1, 2, 3, and 4



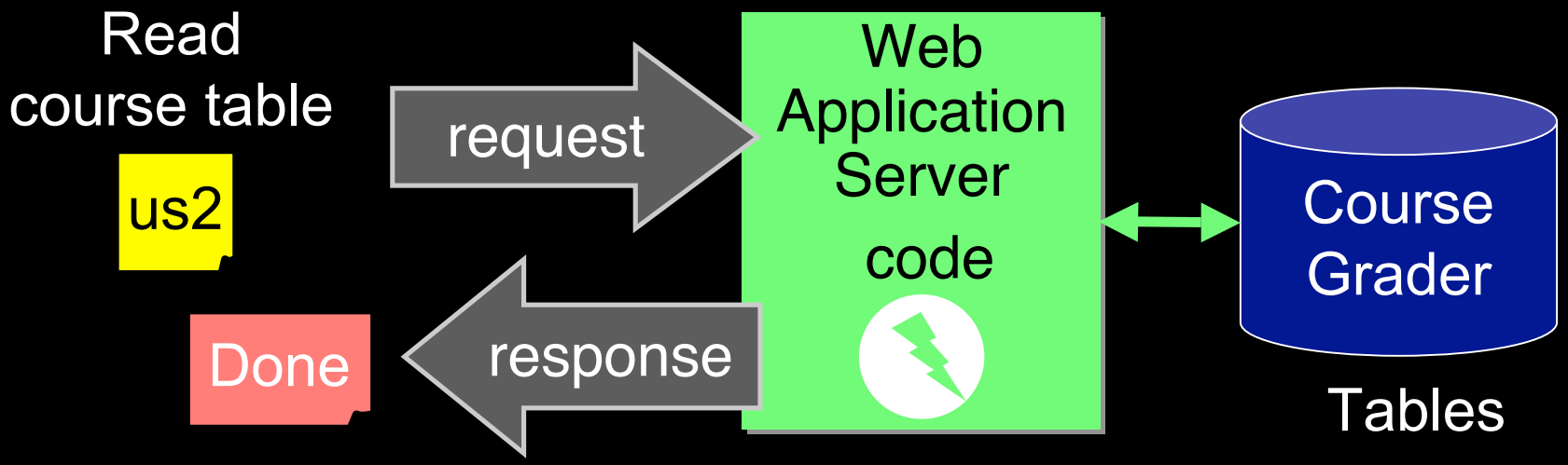
Replaying Original Suite

Replaying user sessions 1, 2, 3, and 4



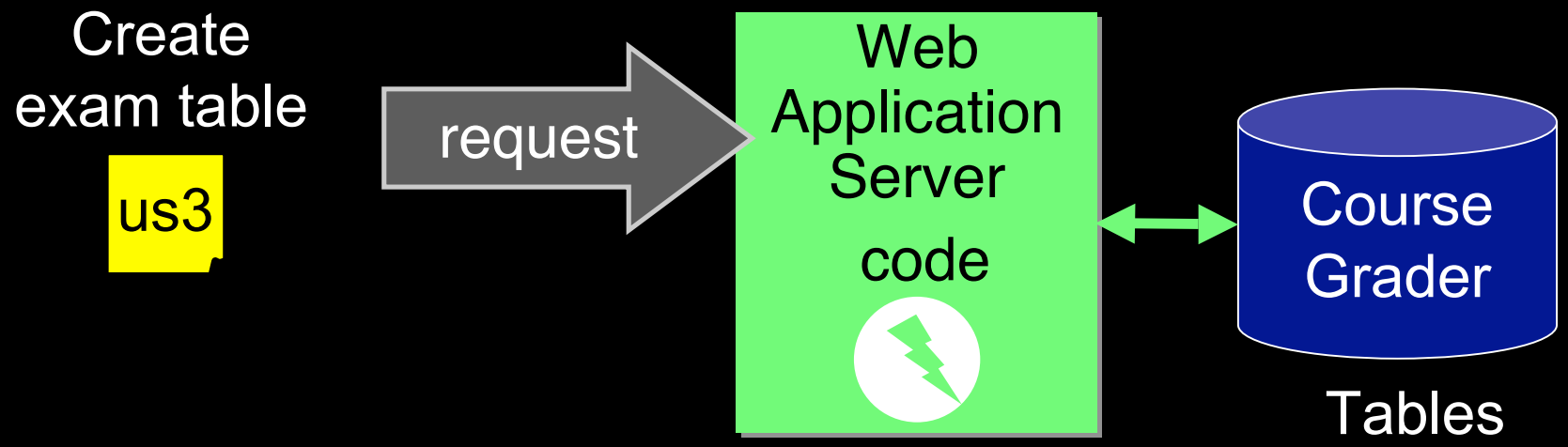
Replaying Original Suite

Replaying user sessions 1, 2, 3, and 4



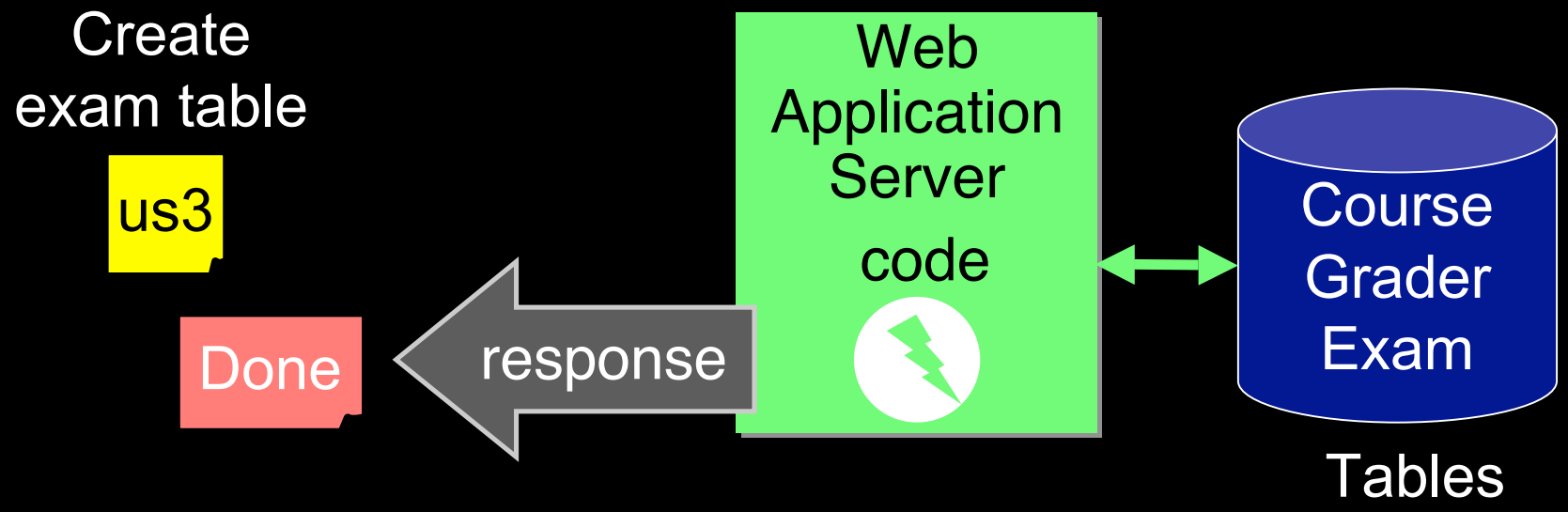
Replaying Original Suite

Replaying user sessions 1, 2, 3, and 4



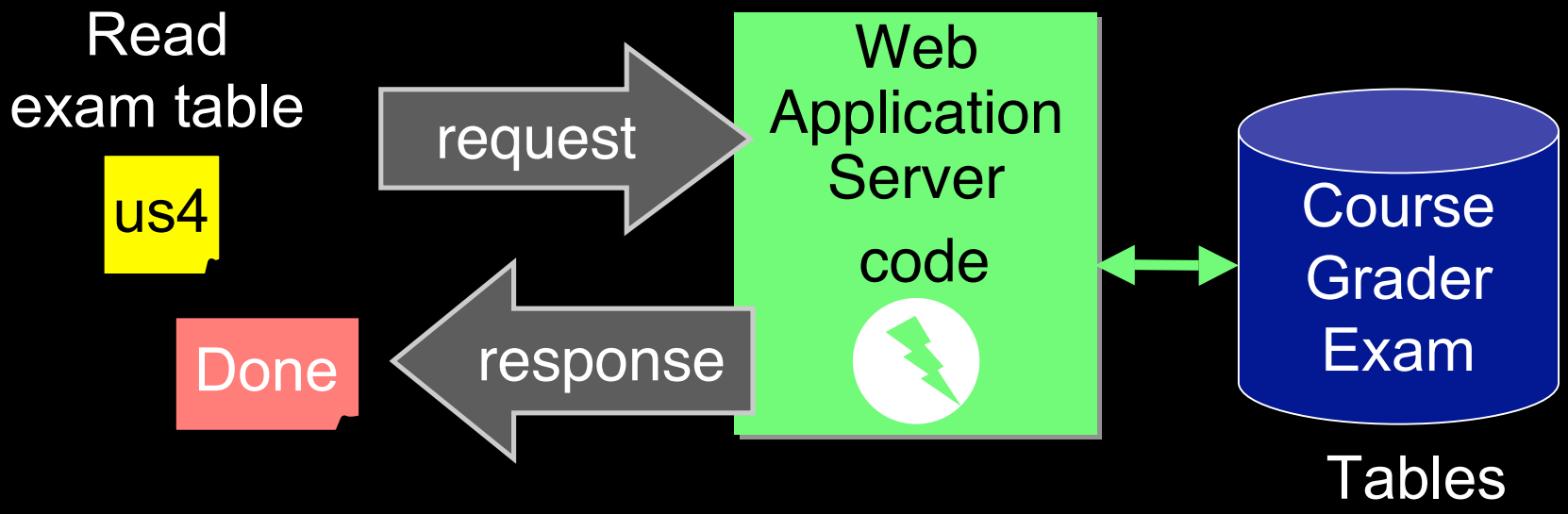
Replaying Original Suite

Replaying user sessions 1, 2, 3, and 4



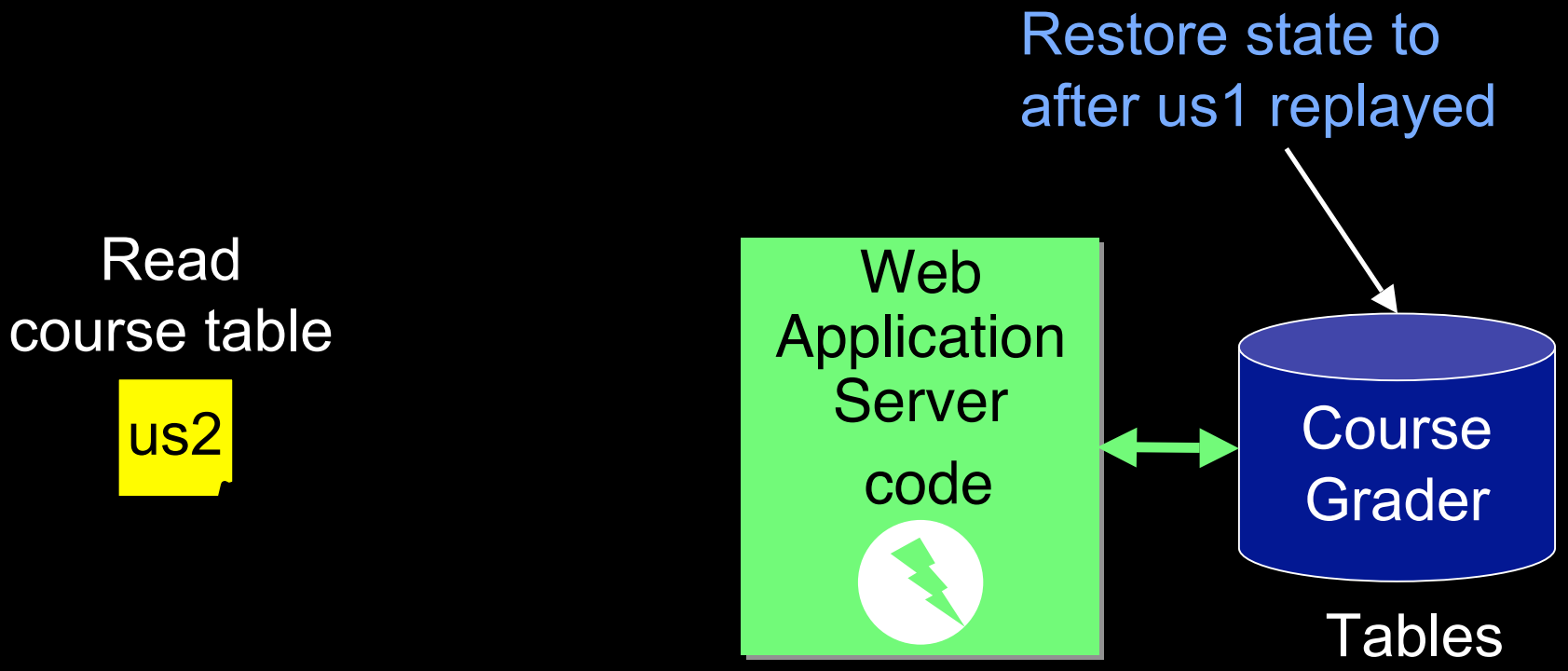
Replaying Original Suite

Replaying user sessions 1, 2, 3, and 4



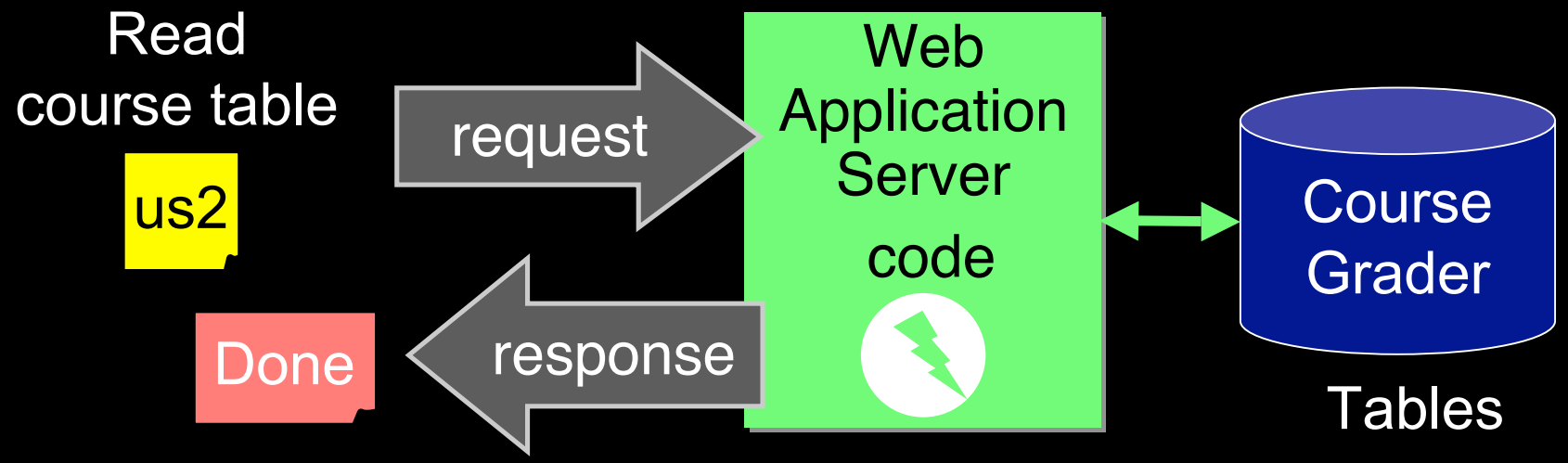
Replay *with* Restoring State

Replaying reduced suite, user sessions 2 and 4



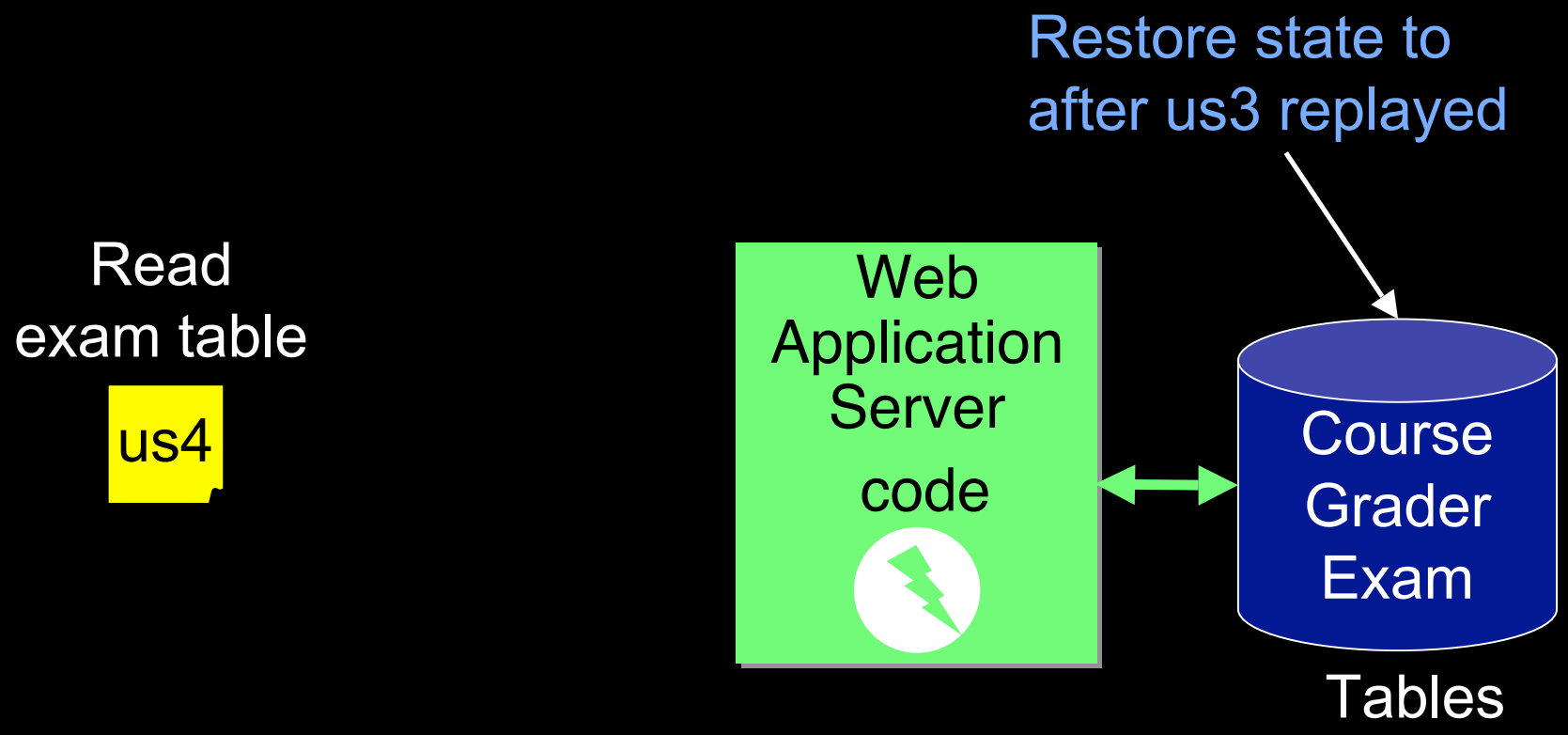
Replay *with* Restoring State

Replaying reduced suite, user sessions 2 and 4



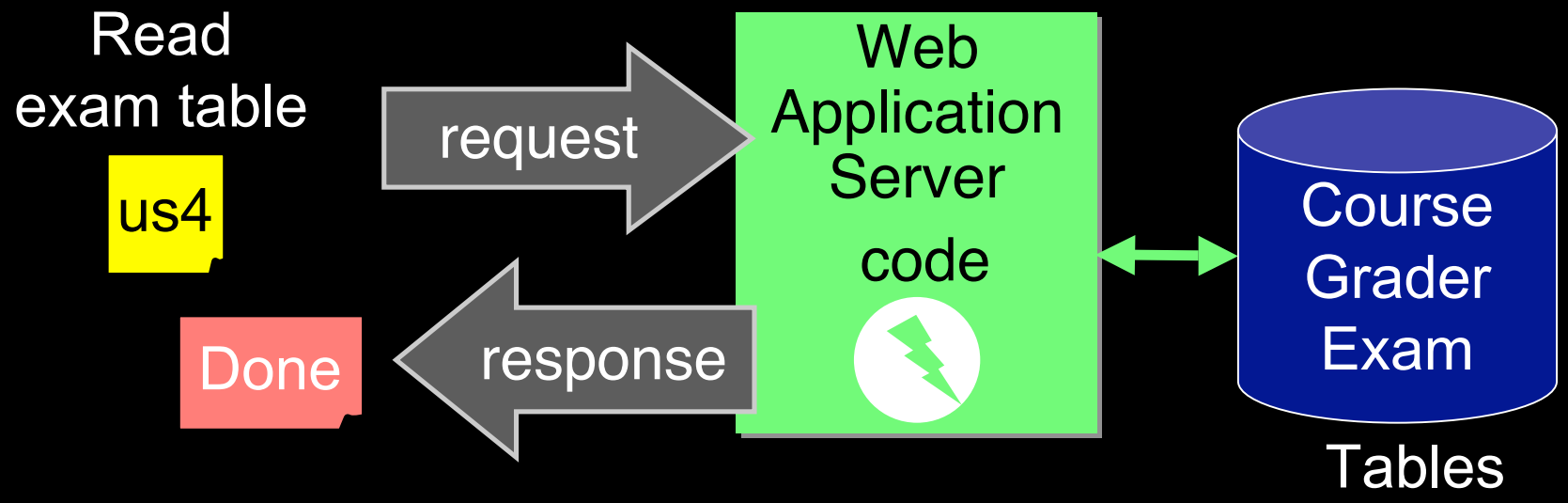
Replay *with* Restoring State

Replaying reduced suite, user sessions 2 and 4



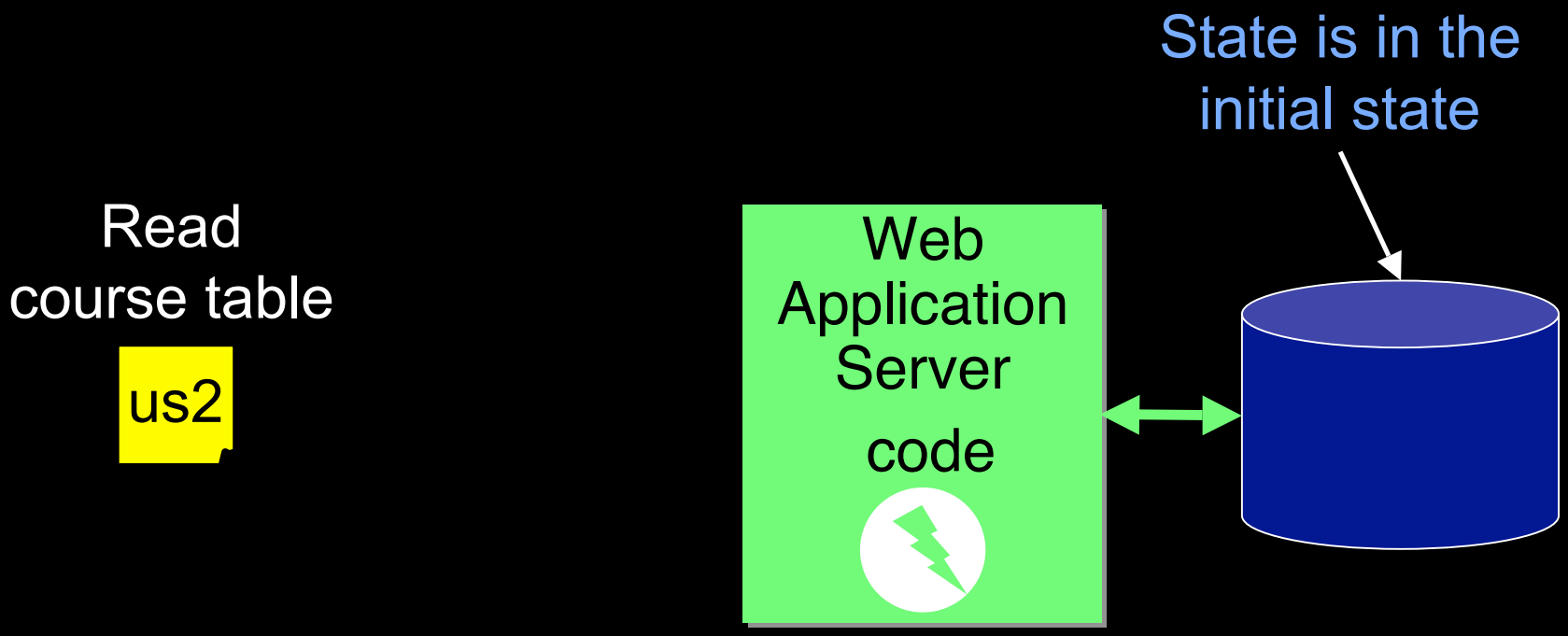
Replay *with* Restoring State

Replaying reduced suite, user sessions 2 and 4



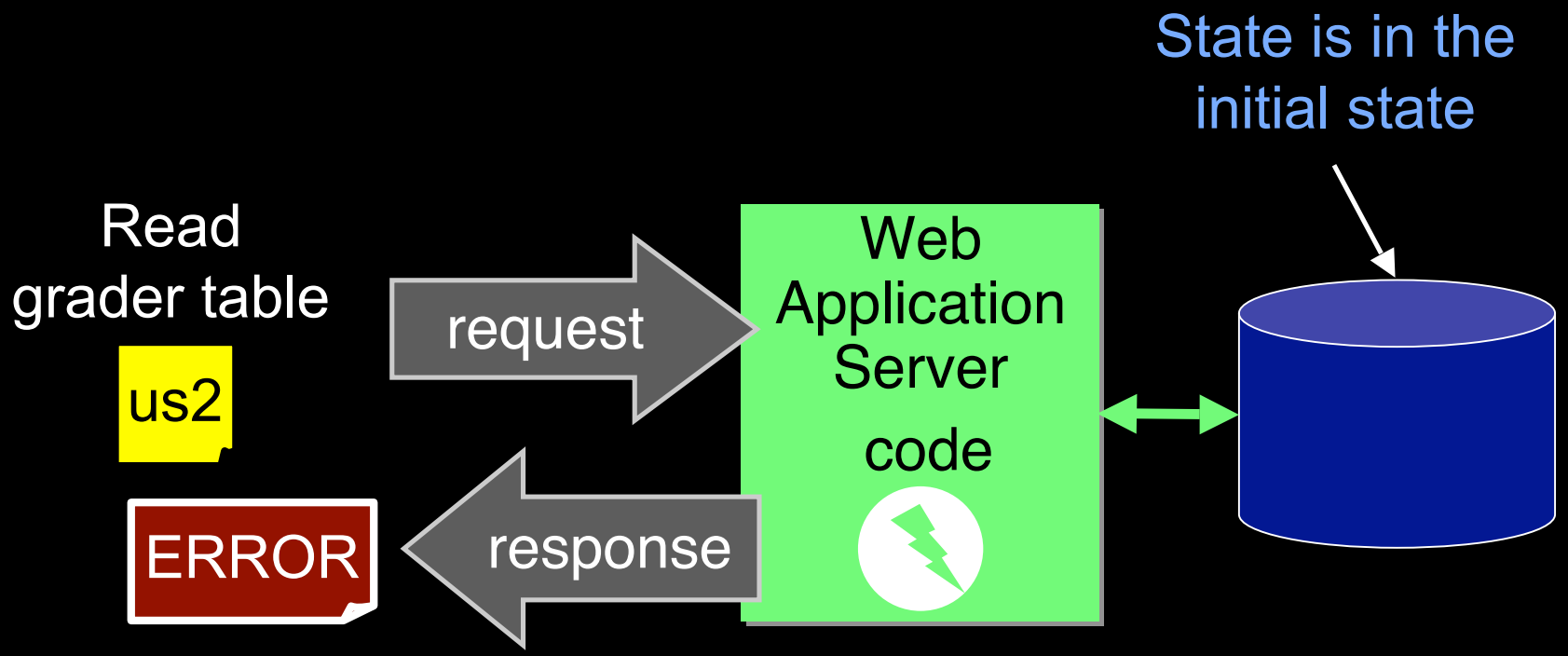
Replay *without* Restoring State

Replaying reduced suite, user sessions 2 and 4



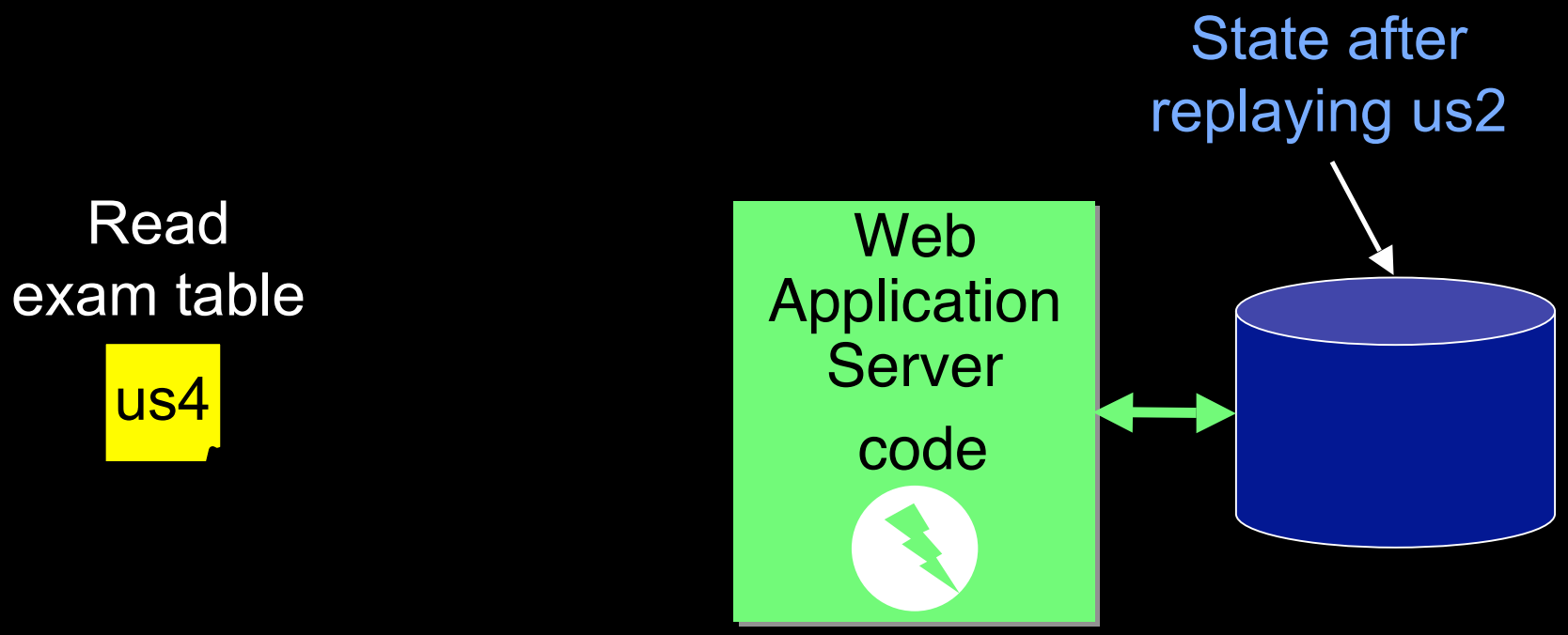
Replay *without* Restoring State

Replaying reduced suite, user sessions 2 and 4



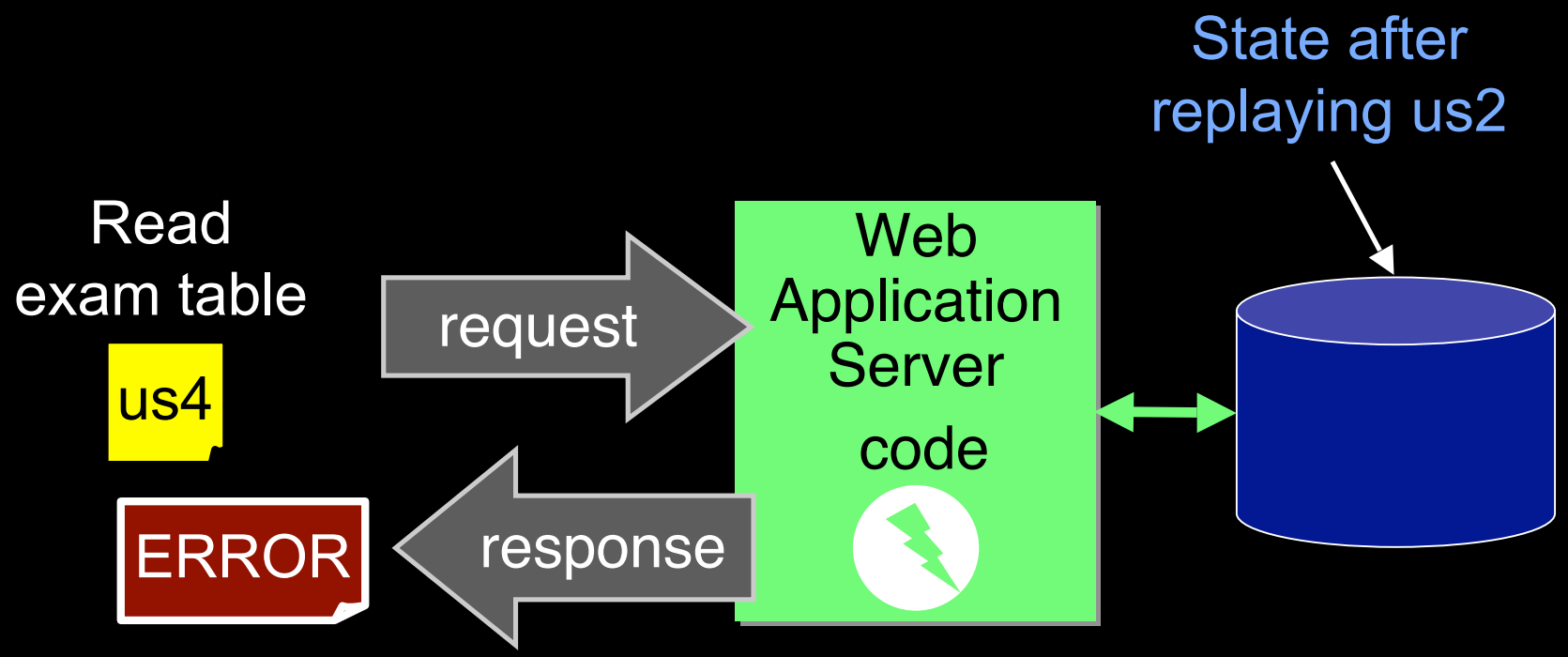
Replay *without* Restoring State

Replaying reduced suite, user sessions 2 and 4



Replay *without* Restoring State

Replaying reduced suite, user sessions 2 and 4



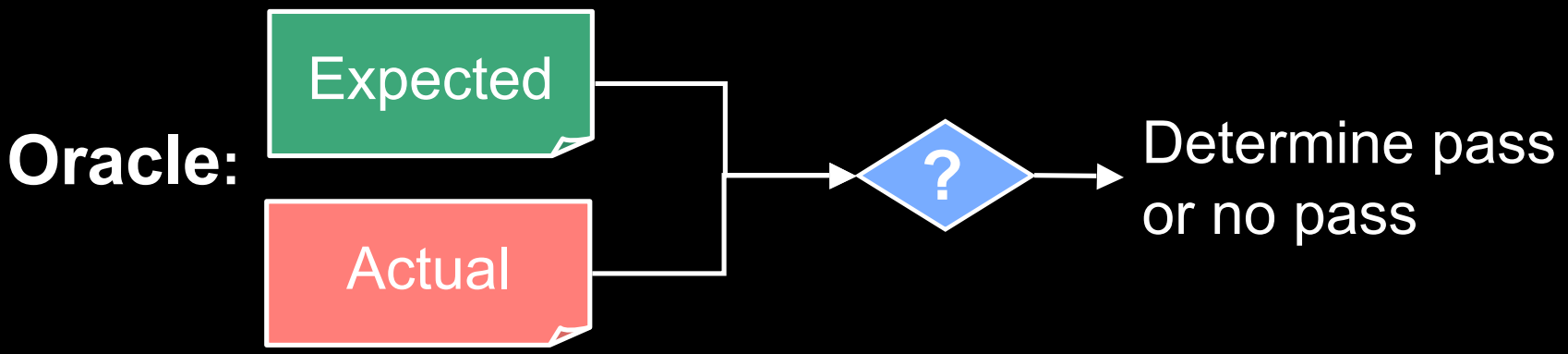
Persistent State: Proposed Solutions

- Replaying **without** restoring state
  - May execute error code
- Replaying **with** restoring state
  - Closely matches execution of original suite
  - Requires running full suite once, saving state
- Augment test suite
  - Add user sessions that affect the state for later user sessions
  - Worst case: include all user sessions

Persistent State: Proposed Solutions

- ✓Replaying **without** restoring state
  - ➢ May execute error code
- ✓Replaying **with** restoring state
  - ➢ Closely matches execution of original suite
  - ➢ Requires running full suite once, saving state
- Augment test suite
  - ➢ Add user sessions that affect the state for later user sessions
  - ➢ Worst case: include all user sessions

Oracles: Challenges



- Generating expected output
- Validating actual output
  - False negatives: miss reporting a fault
  - False positives: report a fault but not faulty behavior
    - Spend time "debugging" correct code

Oracles: Proposed Solutions

- Use original app version to generate expected results
  - Gold Standard
- Automated comparison algorithms
  - HTML: common web application output
  - Propose 4 HTML comparison algorithms
  - False negative: faults that do not manifest themselves in HTML output

Example of HTML Output

```
<html>
<head><title>My Page</title>
<link style></head>
<body>
<h1>Intro</h1>
<p> Text…
<p> Today is November 10.
<a href="link.html">
<h2>Subsection</h2>
<form method=post …>…</form>
</body>
```

- Structure
  - Tags
    - Name
    - Attribute, Value
- Content
  - Text enclosed between tags

Comparison Algorithm: Raw

- Diff entire document
  - Cheap, thorough

```
<html>
<head><title>My Page</title>
<link style></head>
<body>
<h1>Intro</h1>
<p> Text…
<p> Today is November 10.
<a href="link.html">
<h2>Subsection</h2>
<form method=post …>…</form>
</body>
```

```
<html>
<head><title>My Page</title>
<link style></head>
<body>
<h1>Intro</h1>
<p> Text…
<br> Today is November 11.
<a href="link2.html">
<h2>Subsection</h2>
<form method=post …>…</form>
</body>
```

Not a fault

Appearance fault

Link fault

Comparison Algorithm: Content

- Diff document's text

My Page
Intro
 Text…
 Today is November 10.
Subsection

My Page
Intro
 Text…
 Today is November 11.
Subsection

- Misses link fault

Comparison Algorithm: Structure

- Diff document's tags

```
<html>
<head><title></title>
<link style></head>
<body>
<h1></h1>
<p>
<p>
<a href="link.html">
<h2></h2>
<form method=post …>…</form>
</body>
```

```
<html>
<head><title></title>
<link style></head>
<body>
<h1></h1>
<p>
<br>
<a href="link2.html">
<h2></h2>
<form method=post …>…</form>
</body>
```

HTML Comparison Algorithms

- Raw: compare the entire document
  - Diff documents
- Content: compare text between tags
  - Filter document's text; diff
- Structure: compare tags
  - Filter document's tags; diff
- Flist: compare list of downloaded URLs
  - Diff downloads' directory listing

HTML Comparison Algorithms

False negatives: faults not manifested in HTML

Comparison Algorithm	False Positives	False Negatives
Raw	Dynamic, real-time changes	———
Content	Dynamic, real-time changes	Errors in structure, e.g. forms
Structure	Display changes that do not affect app behavior	Errors in content
Flist	———	Structure & content

Expected Tradeoffs

- Computation Cost
  - Flist < Raw < Structure, Content
- False Positives
  - Flist < Structure, Content < Raw
- False Negatives
  - Raw < Structure, Content < Flist
  - Future work to study empirically

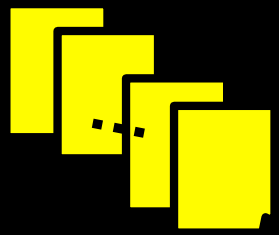
Empirical Study Questions

- How does the replay technique affect the test suite's program coverage?
- How do the replay technique and oracle comparator affect the set of reported faults?
- What are the time and space costs of each replay technique and oracle comparator?

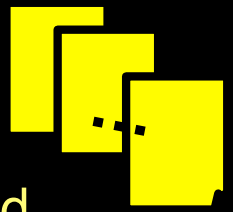
Empirical Study Questions

- How does the replay technique affect the test suite's program coverage?
- How do the replay technique and oracle comparator affect the set of reported faults?
- What are the time and space costs of each replay technique and oracle comparator?

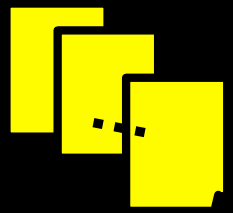
Empirical Study Methodology

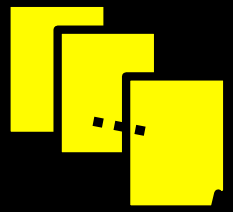


Original test suite



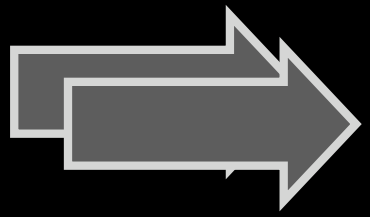
Reduced suites



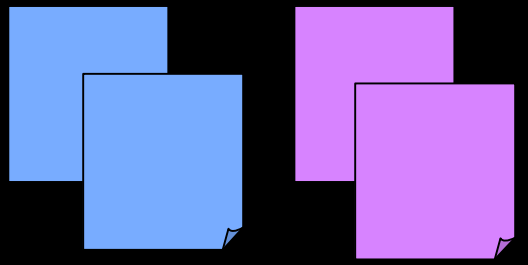


Performed study on 2 web applications

Seeded fault types: logic, data store, appearance, form, link



Replay with and without restoring state



Generate coverage, fault detection reports using each **oracle**

Summary of Replay Results

- Techniques covered different code, exposed different faults
  - With state covered more code, faults
  - Without state covered & exposed faults in error handling code
- No correlation between reduced test suite size and difference in coverage

Summary of Oracle Results

- Reported faults
  - Raw > Structure, Content > Flist
  - Form faults
    - Structure > Content
  - Flist detects mostly logic & data faults

- Observed false positives
  - Raw: form input (e.g., a.m. → p.m.)
  - Content: demos printed in different order
  - Structure: none in our study

Guidance to Testers: Replay

- Both with and without state
  - For maximum coverage & fault exposure
  - If time permitting
- With state
  - More closely matches original suite's behavior
- Without state
  - If little persistent state maintained
  - Want to cover error code & faults in error code

Guidance to Testers: Oracles

- For fewer false positives
  - Combine use of Structure, Content
- For more faults detected
  - Raw but must look at false positives
- Quick filter: Flist
- Detect different fault types
  - Form: Raw, Structure
  - Appearance: Raw, Structure, Content

Our Contributions

- Automated strategies for handling persistent state during replay
  - Expose different faults with same suite
- Automated, pluggable oracle comparators
  - Detect different types of faults
- Experimental study of strategies' and comparators' effects on fault detection
- Guidance to testers about strategies

Future Work

- Investigate additional replay techniques
  - Order: Concurrent replay
  - State: User-session dependencies
- Further study of oracles' false positives, false negatives
- Improve oracle accuracy
  - Better heuristic to quantify *equivalent* results
- Automated fault seeding