Identifying Information Disclosure in Web Applications with Retroactive Auditing

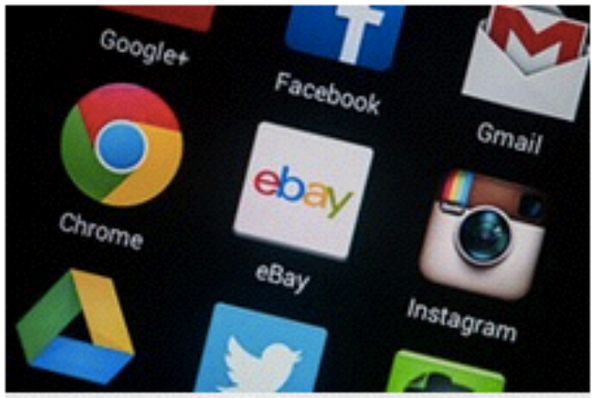
Haogang Chen, Taesoo Kim, Xi Wang
Nickolai Zeldovich, and M. Frans Kaashoek

MIT CSAIL

Data breach: an enduring problem

27 MAY 2014 | NEWS

New Web Vulnerabilities Expose eBay User Data Again



New Web Vulnerabilities Expose eBay User Data Again

Security researchers have warned that eBay user accounts could still be at risk, just days after the firm was forced to admit a major data breach, after spotting new critical web vulnerabilities.

The first was discovered by Stockton-on-Tees based researcher Jordan Jones, who took to Twitter to reveal he had managed to upload shellcode to eBay, which could give him remote control of the targeted server.

The online giant said in a message shared by Jones on Twitter that it has since resolved the problem and promised to add his name to its "acknowledgement page".

Data breach: an enduring problem

27 MAY 2014 | NEWS

New Web Vulnerabilities Expose eBay User Data

JPMorgan Says Data Breach Hit 76 Million Households

The Huffington Post | By Mark Gongloff

Posted: 10/02/2014 5:17 pm EDT | Updated: 10/02/2014 6:59 pm EDT

A cyber attack at America's biggest bank this summer affected more than half of all U.S. households -- far, far more than previously estimated, and the latest in a string of massive, unnerving data breaches.

The attack at JPMorgan Chase affected the data of 76 million households and 7 million businesses, the bank said in a regulatory filing on Thursday.

That impact was far bigger than earlier estimates that about 1 million customers had been affected, the New York Times noted. It represents more than half of the roughly 115 million households in America.

What to do about it?

What to do about it?

- Before data breach: prevention techniques
  - privilege separation
  - encryption
  - information flow control

What to do about it?

- Before data breach: prevention techniques
  - privilege separation
  - encryption
  - information flow control
- After a potential data breach: **damage control**

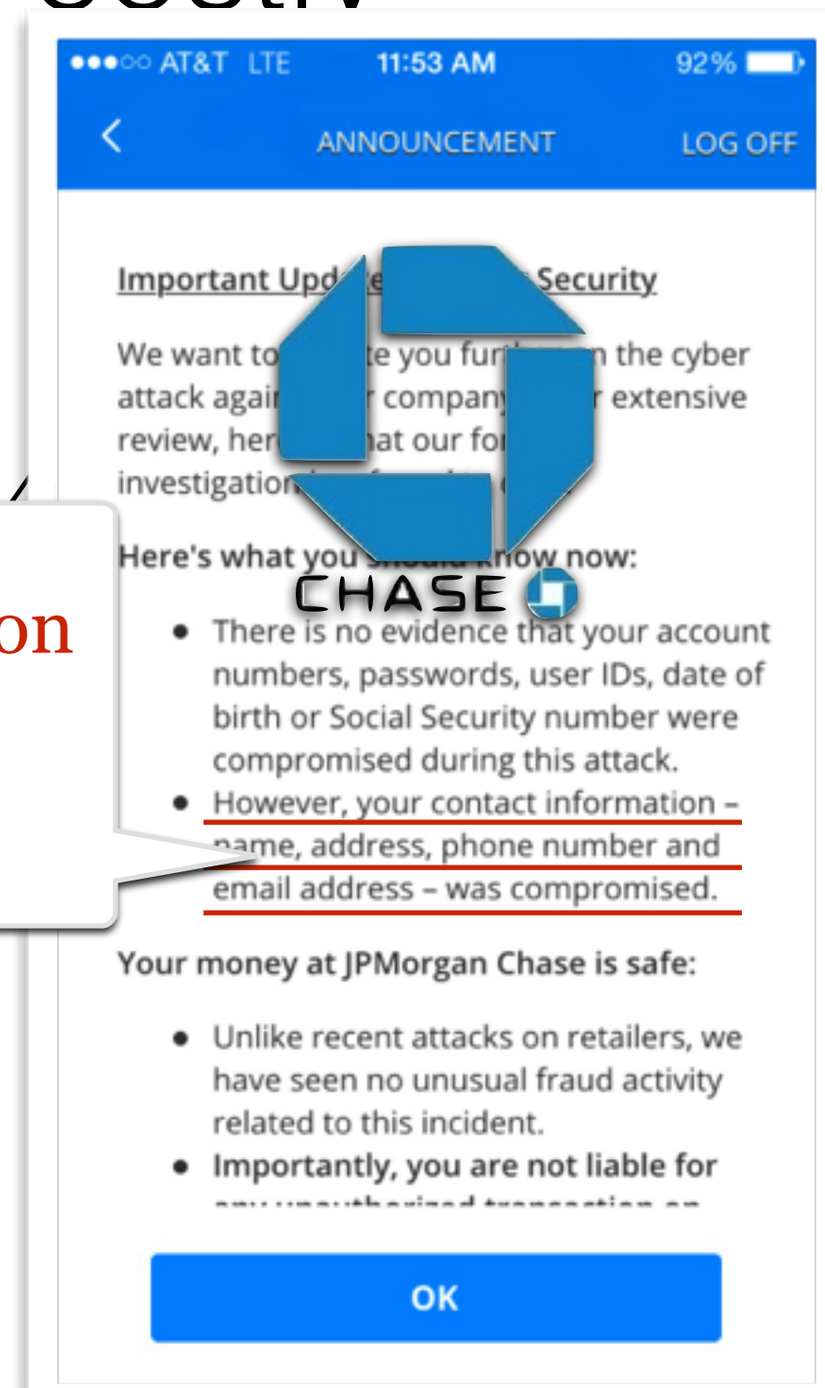
Observation:
Damage control is costly

- Notify the victims
  - Data Breach Notification Laws (40/50 states)

Observation:
Damage control is costly

- Notify the victims
  - Data Breach Notification Laws (4

"..... However, your contact information — name, address, phone number and email address — was compromised."



Observation:
Damage control is costly

- Notify the victims
  - Data Breach Notification Laws (40/50 states)
- Pay for credit monitoring & fraud protection

Observation:

- 

- e.g., University of Maryland pledges to offer its 309,079 victims for 5-year of credit monitoring

Opportunity:
Some data might not be leaked

Opportunity:
Some data might not be leaked

- The vulnerability might not have been exploited yet

Opportunity:
Some data might not be leaked

- The vulnerability might not have been exploited yet
- Attackers might not steal all data that they can

Opportunity:
Some data might not be leaked

- The vulnerability might not have been exploited yet
- Attackers might not steal all data that they can

- **Goal:** precisely identify breached data items

Opportunity:
Some data might not be leaked

- The vulnerability might not have been exploited yet
- Attackers might not steal all data that they can

- **Goal:** precisely identify breached data items
- Target damage control at real victims only

State of the art

- Log all accesses to sensitive data
- Inspect logs after an intrusion

State of the art

- Log all accesses to sensitive data
- Inspect logs after an intrusion
- **Problems**
  - Need to know what is sensitive data beforehand

State of the art

- Log all accesses to sensitive data
- Inspect logs after an intrusion
- **Problems**
  - Need to know what is sensitive data beforehand
  - Hard to tell legal v.s. illegal accesses

State of the art

- Log all accesses to sensitive data
- Inspect logs after an intrusion
- **Problems**
  - Need to know what is sensitive data beforehand
  - Hard to tell legal v.s. illegal accesses
  - Takes a long time:
    e.g., University of Maryland: one month to inspect 309,079 breached records

Solution: **Rail**

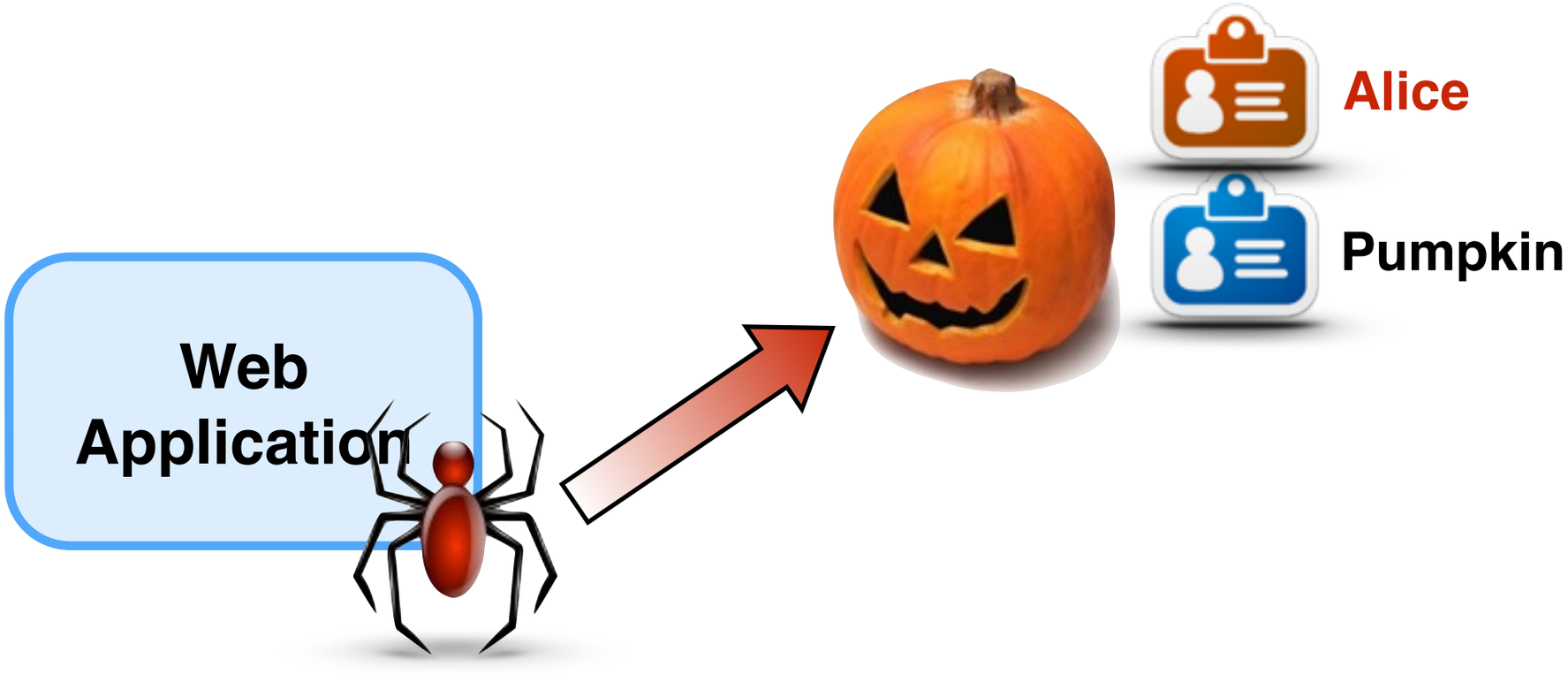
- **Goal:** precisely identify previously breached data *after* a vulnerability is fixed

Solution: **Rail**

- **Goal:** precisely identify previously breached data *after* a vulnerability is fixed

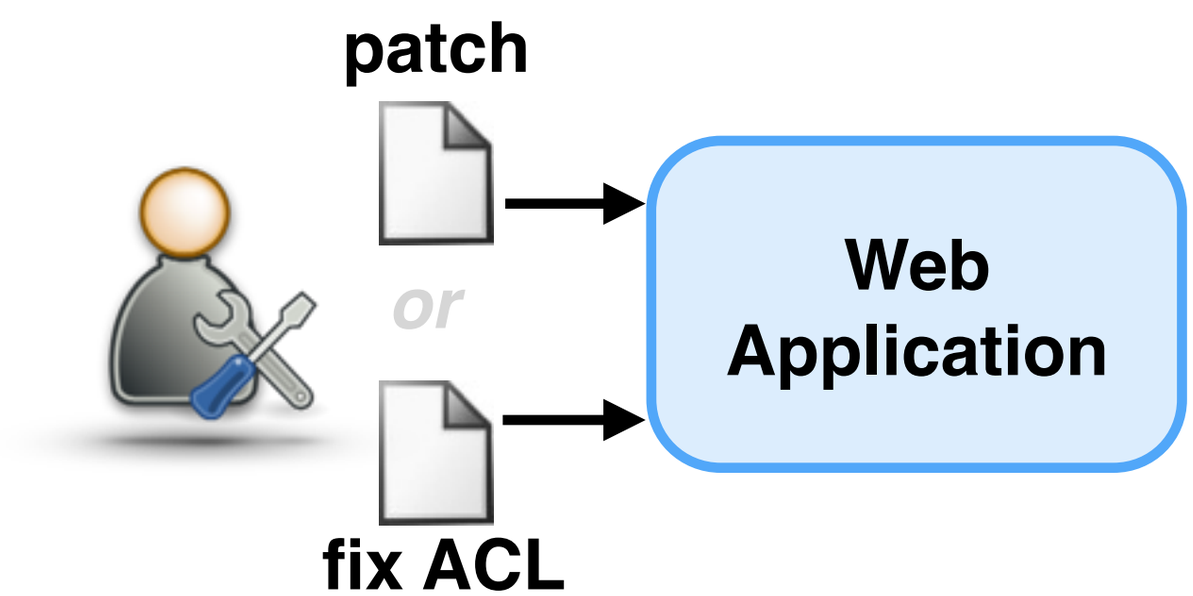
Solution: **Rail**

- **Goal:** precisely identify previously breached data *after* a vulnerability is fixed



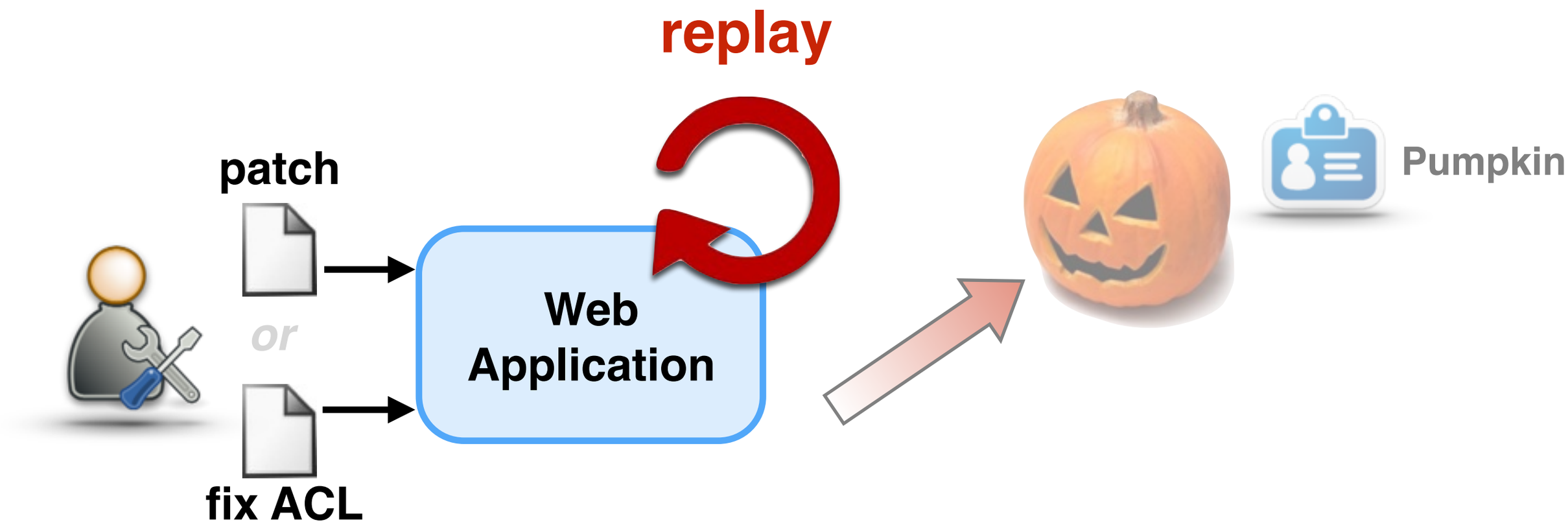
Solution: **Rail**

- **Goal:** precisely identify previously breached data *after* a vulnerability is fixed



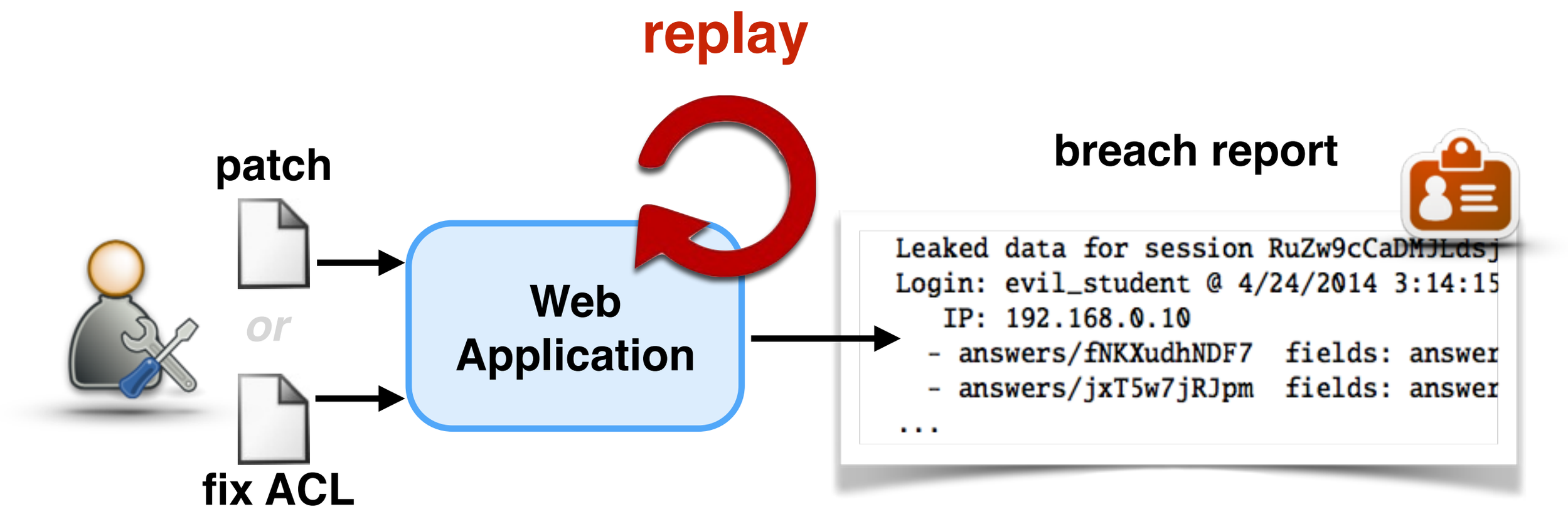
Solution: **Rail**

- **Goal:** precisely identify previously breached data *after* a vulnerability is fixed



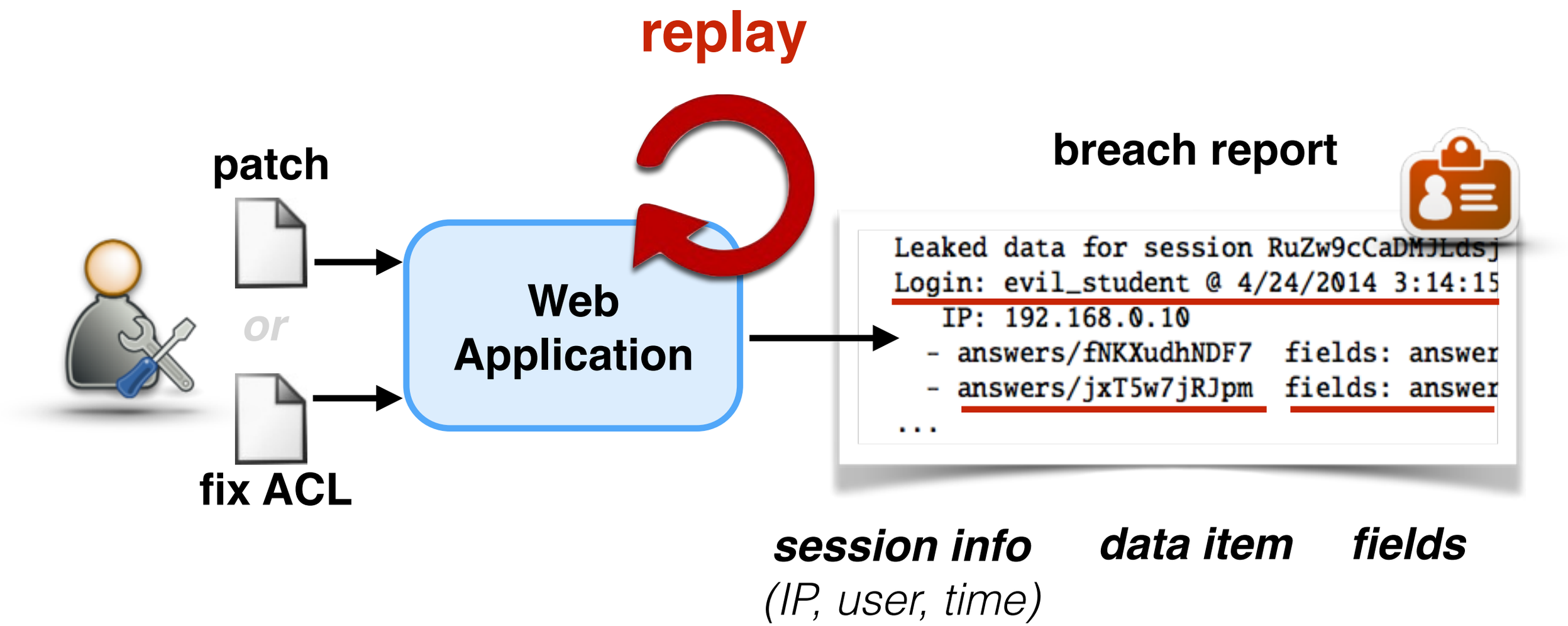
Solution: **Rail**

- **Goal:** precisely identify previously breached data *after* a vulnerability is fixed



Solution: **Rail**

- **Goal:** precisely identify previously breached data *after* a vulnerability is fixed



Challenge

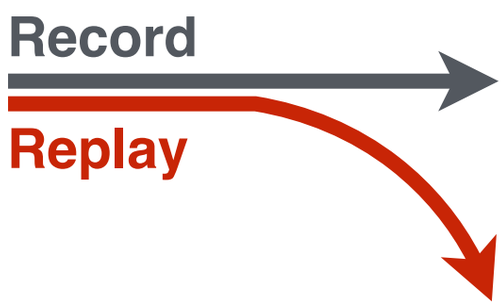
- State during replay can *diverge* from the original execution

Challenge

- State during replay can *diverge* from the original execution
- Prior systems use record and replay for *integrity*

Challenge

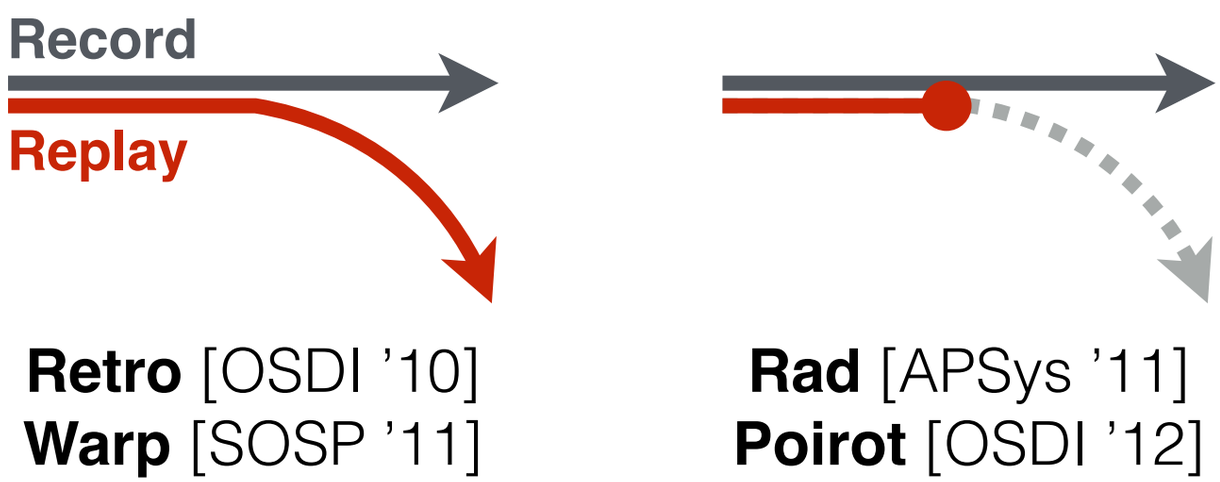
- State during replay can *diverge* from the original execution
- Prior systems use record and replay for *integrity*



Retro [OSDI '10]
Warp [SOSP '11]

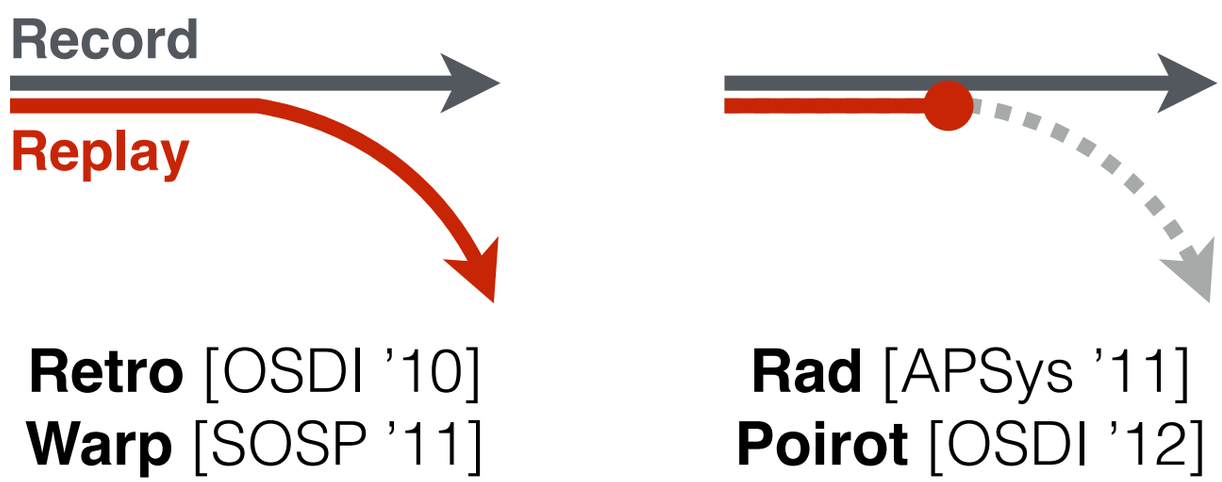
Challenge

- State during replay can *diverge* from the original execution
- Prior systems use record and replay for *integrity*



Challenge

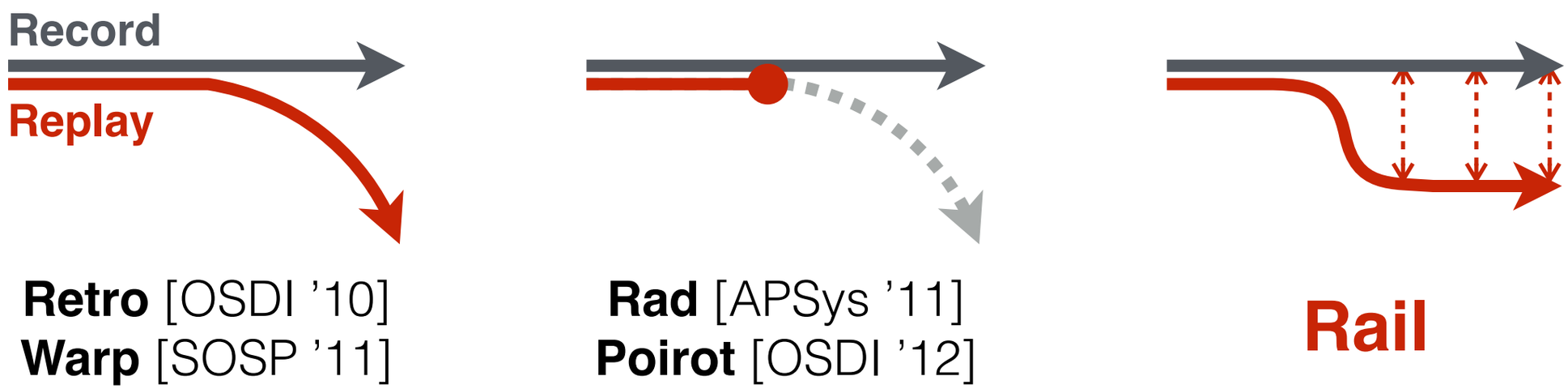
- State during replay can *diverge* from the original execution
- Prior systems use record and replay for *integrity*



- Rail focuses on *confidentiality*

Challenge

- State during replay can *diverge* from the original execution
- Prior systems use record and replay for *integrity*



- Rail focuses on *confidentiality*
- For precision, Rail must **match up state** and **minimize state divergence** between the two executions

Contribution

- Record and replay scheme for identifying data disclosures
  - APIs for application developers
  - Context matching to improve precision
- Prototype based on Meteor web framework
- Result: few changes to applications, precise, fast

Focus and assumptions

- **Focus: web applications**
  - Our prototype is based on, but not limited to, Meteor

Focus and assumptions

- **Focus: web applications**
  - Our prototype is based on, but not limited to, Meteor
- **Assumptions**
  - Trusts the software stack below the web application
    - TCB: web framework, runtime, DBMS, OS, etc.
  - Requests do not change during replay, except for fixes
  - Requests are serializable, etc.

Basic approach

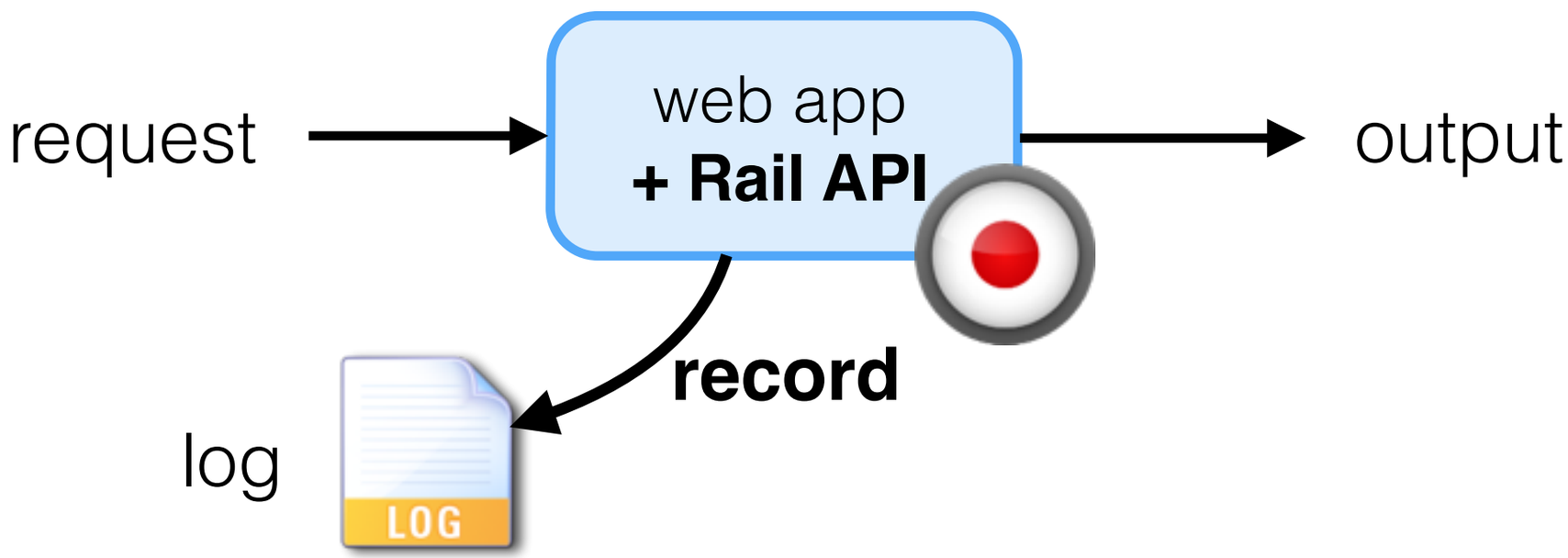
- Record and replay the web application
- Compare the outputs of two executions

Basic approach

- Record and replay the web application
- Compare the outputs of two executions

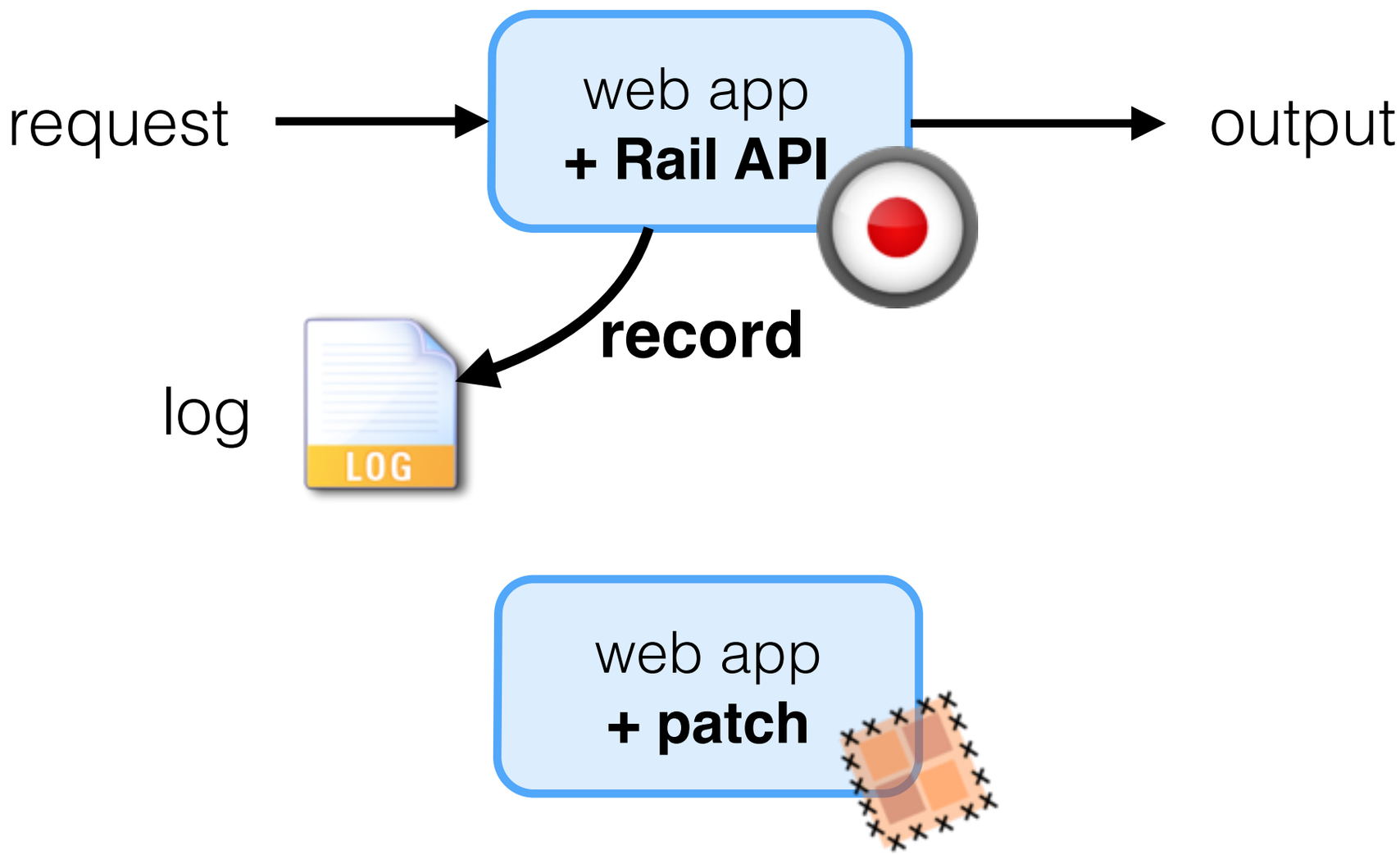
Basic approach

- Record and replay the web application
- Compare the outputs of two executions



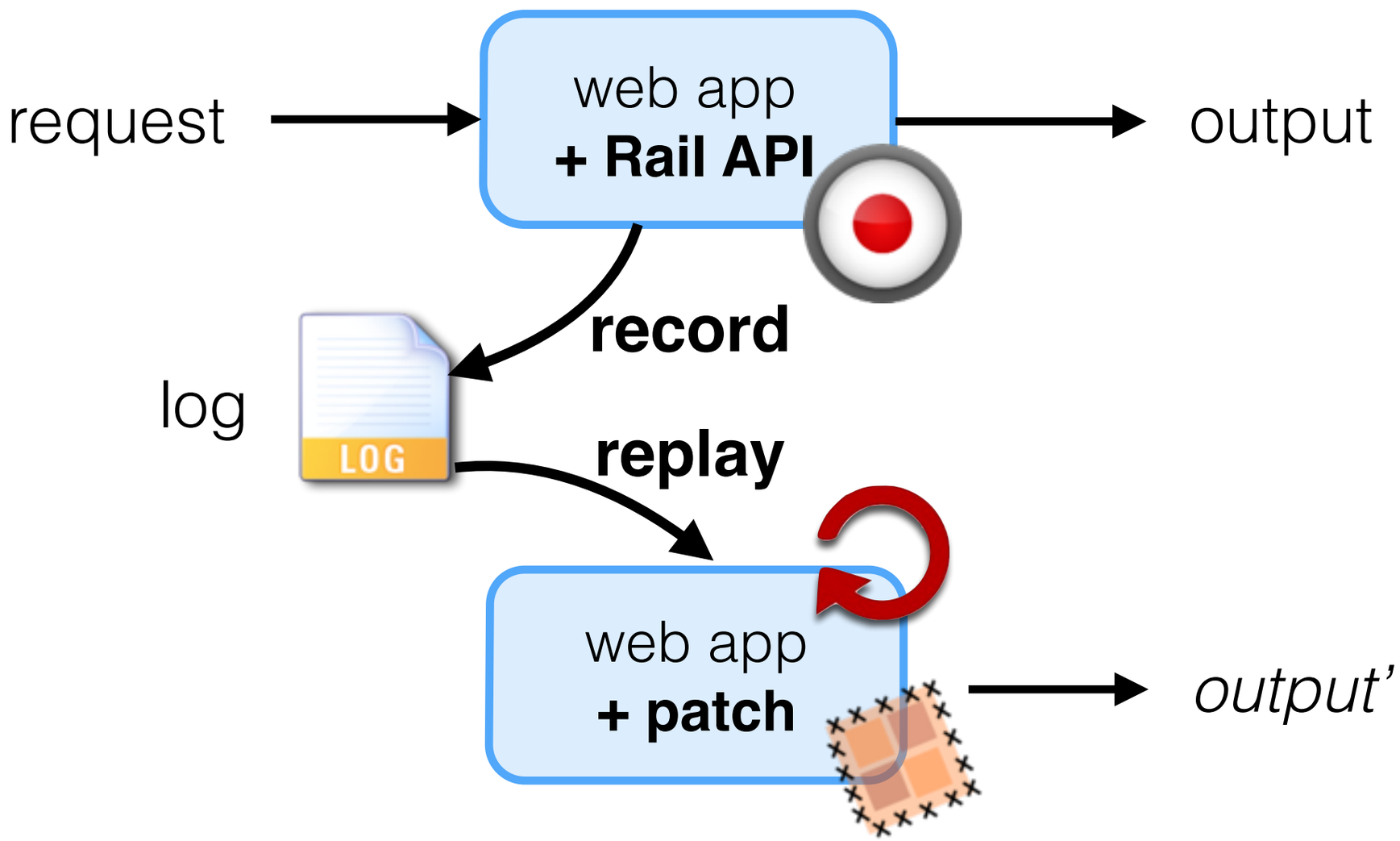
Basic approach

- Record and replay the web application
- Compare the outputs of two executions



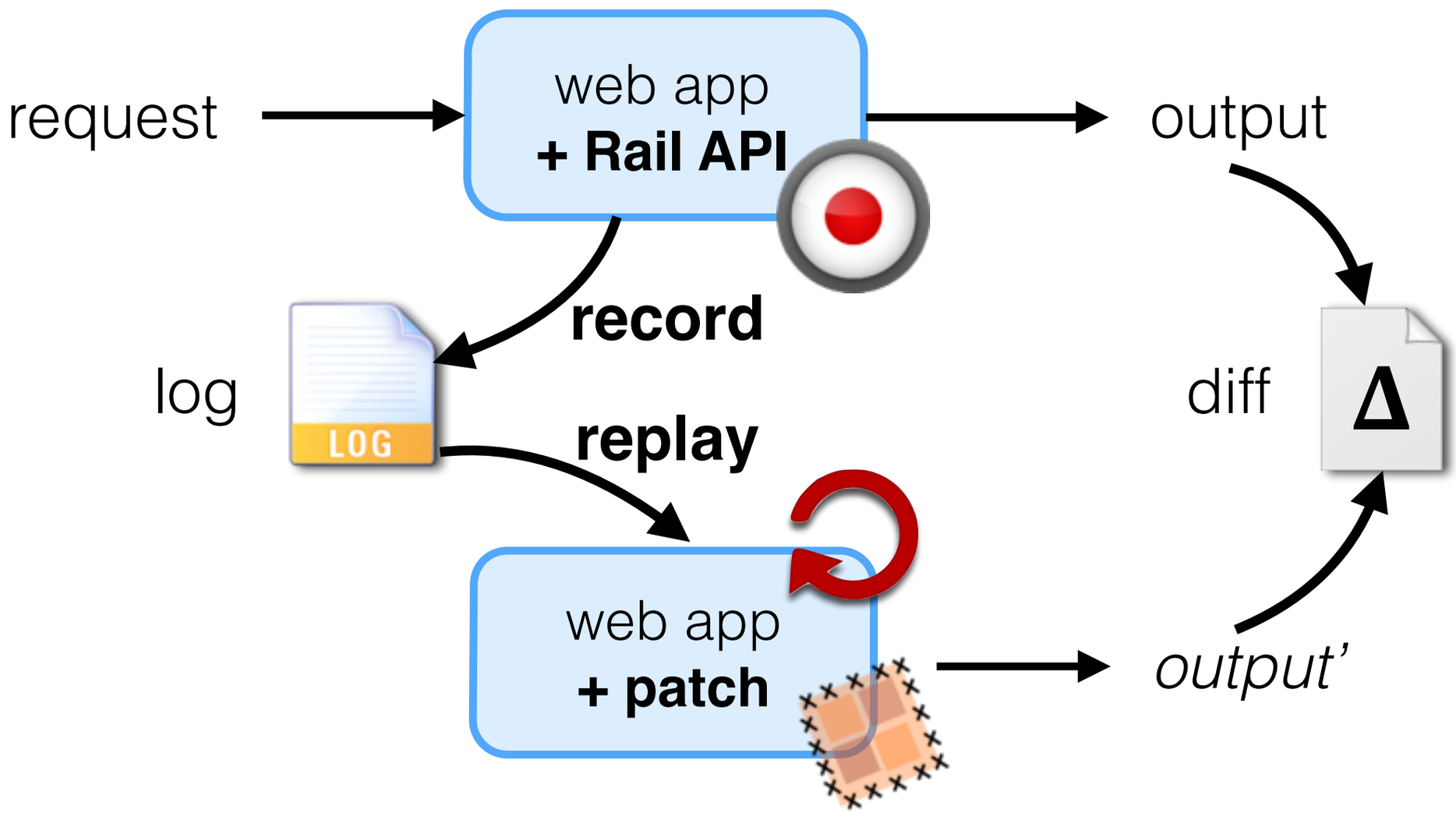
Basic approach

- Record and replay the web application
- Compare the outputs of two executions

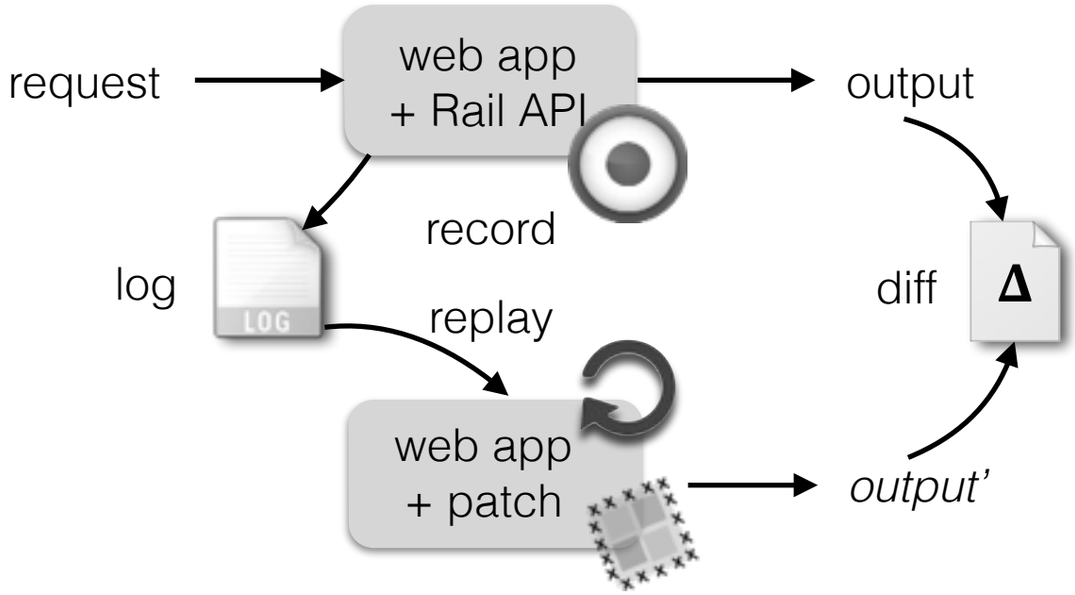


Basic approach

- Record and replay the web application
- Compare the outputs of two executions

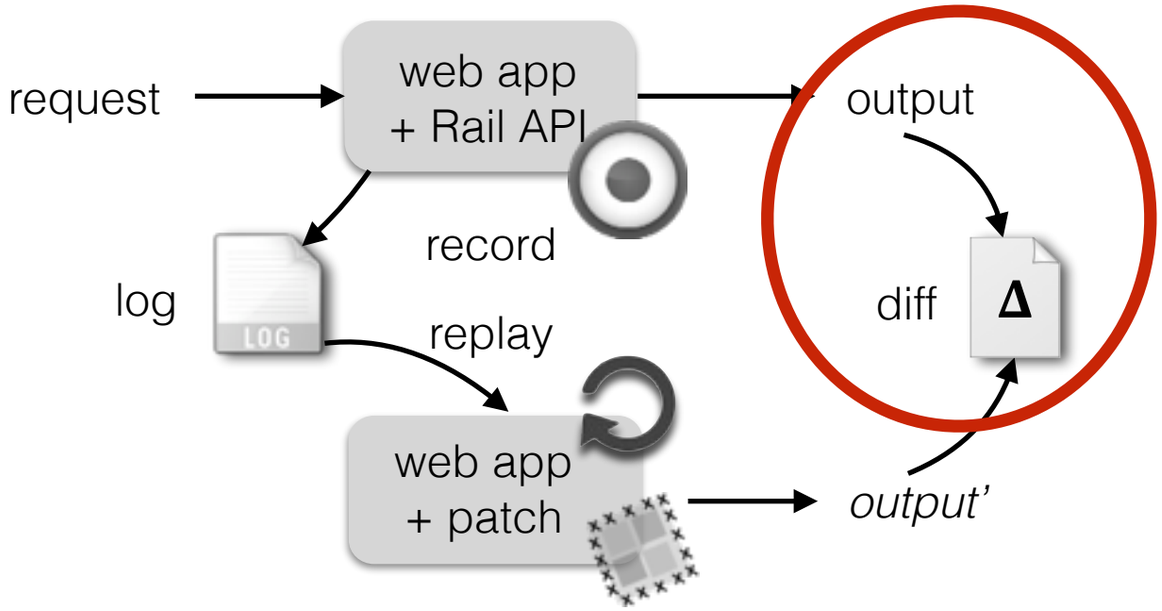


Technical challenges



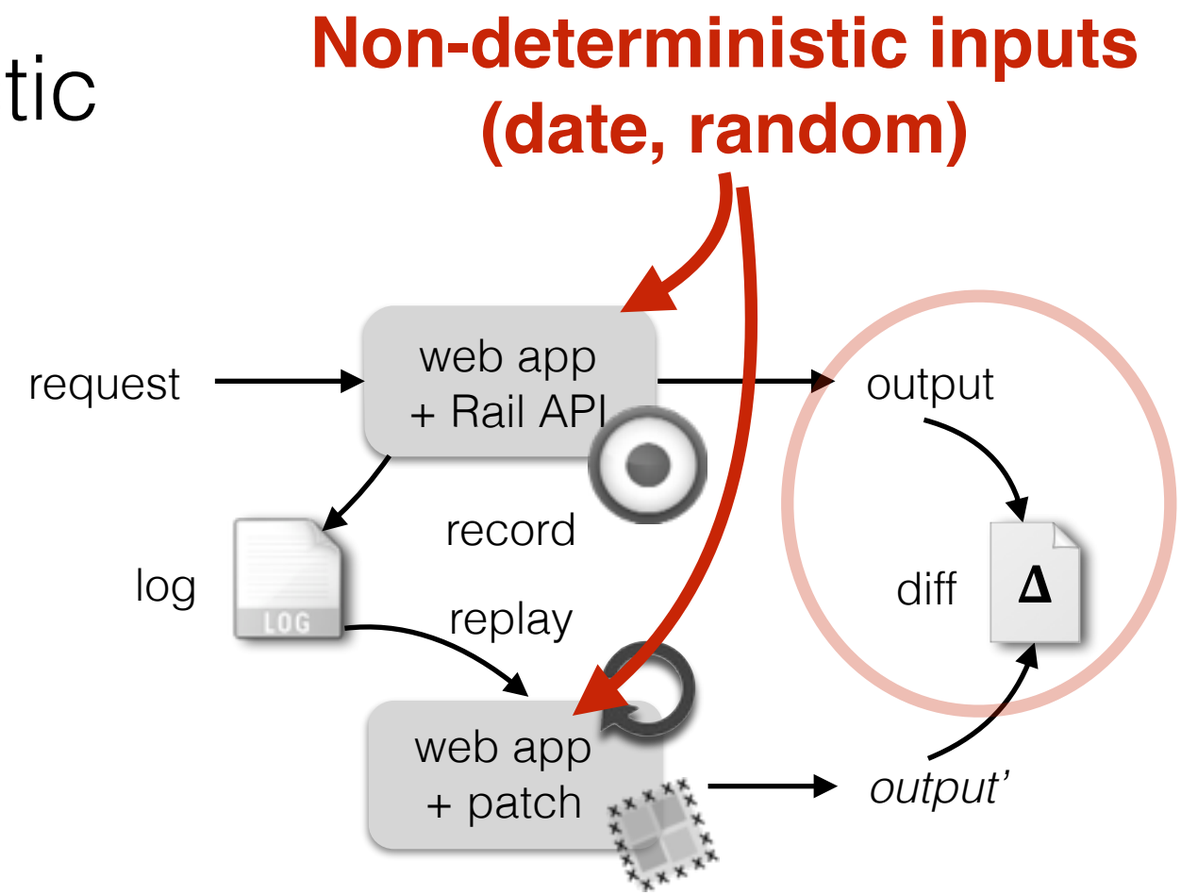
Technical challenges

- Compare & identify data items at object level



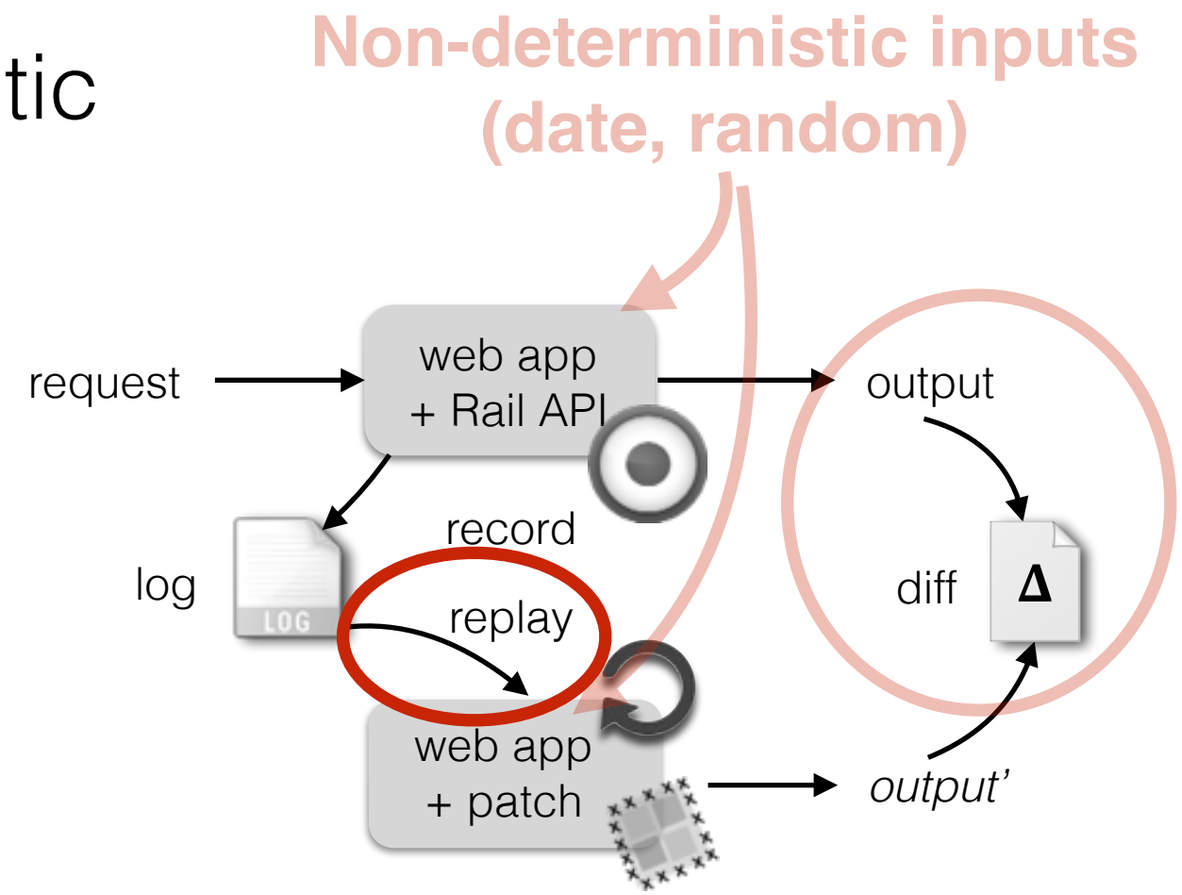
Technical challenges

- Compare & identify data items at object level
- Make replay deterministic



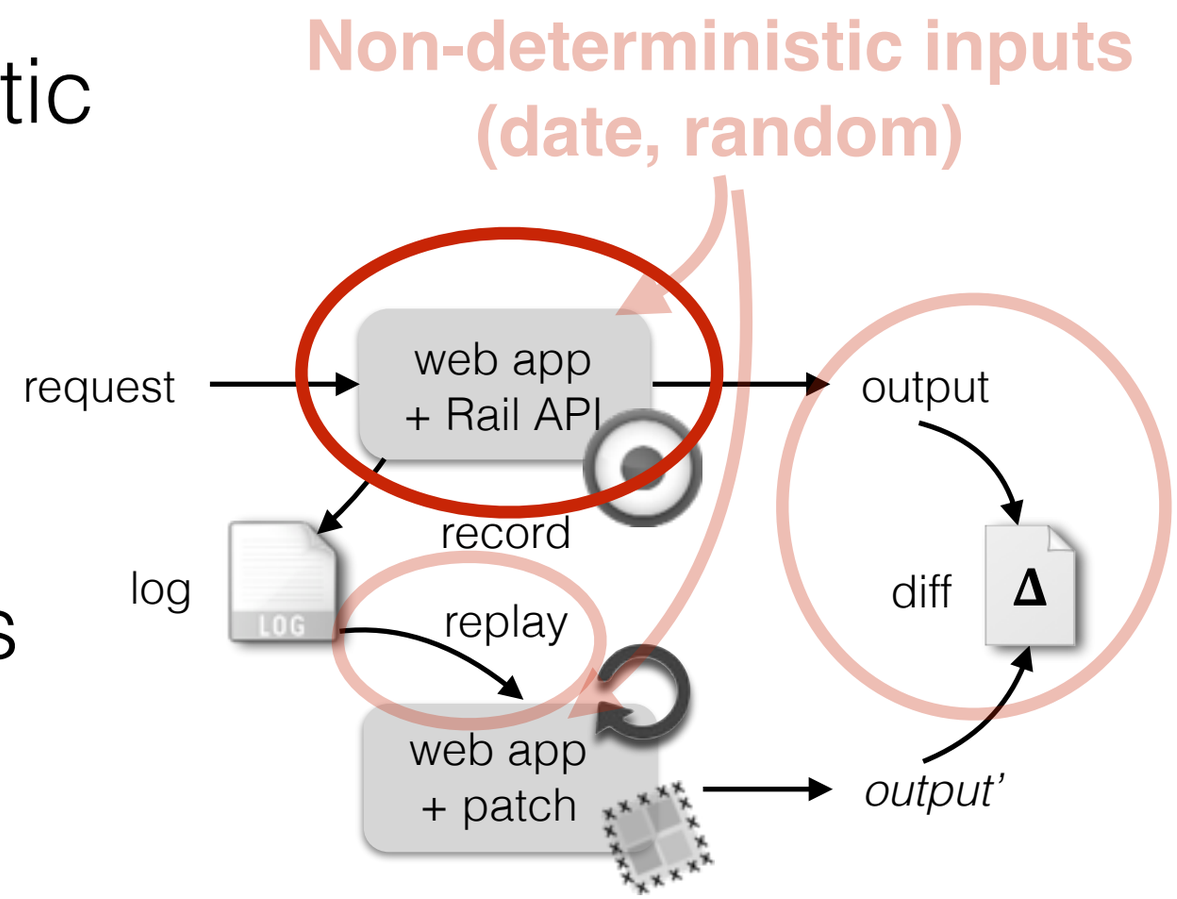
Technical challenges

- Compare & identify data items at object level
- Make replay deterministic
- Selective replay for performance



Technical challenges

- Compare & identify data items at object level
- Make replay deterministic
- Selective replay for performance
- Minimize code changes in the application

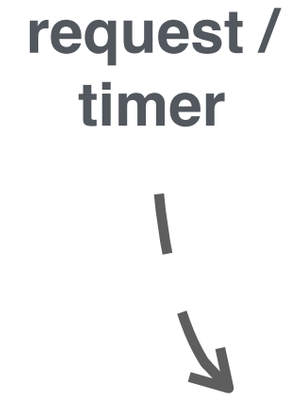


Design: action history graph (AHG)

- AHG [*OSDI '10*] tracks dependencies among **actions** and **objects**

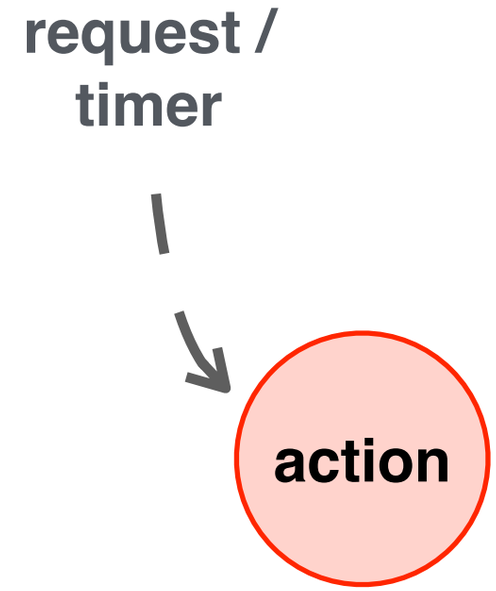
Design: action history graph (AHG)

- AHG [*OSDI '10*] tracks dependencies among **actions** and **objects**
- **Actions**
  - Triggered by external **events**



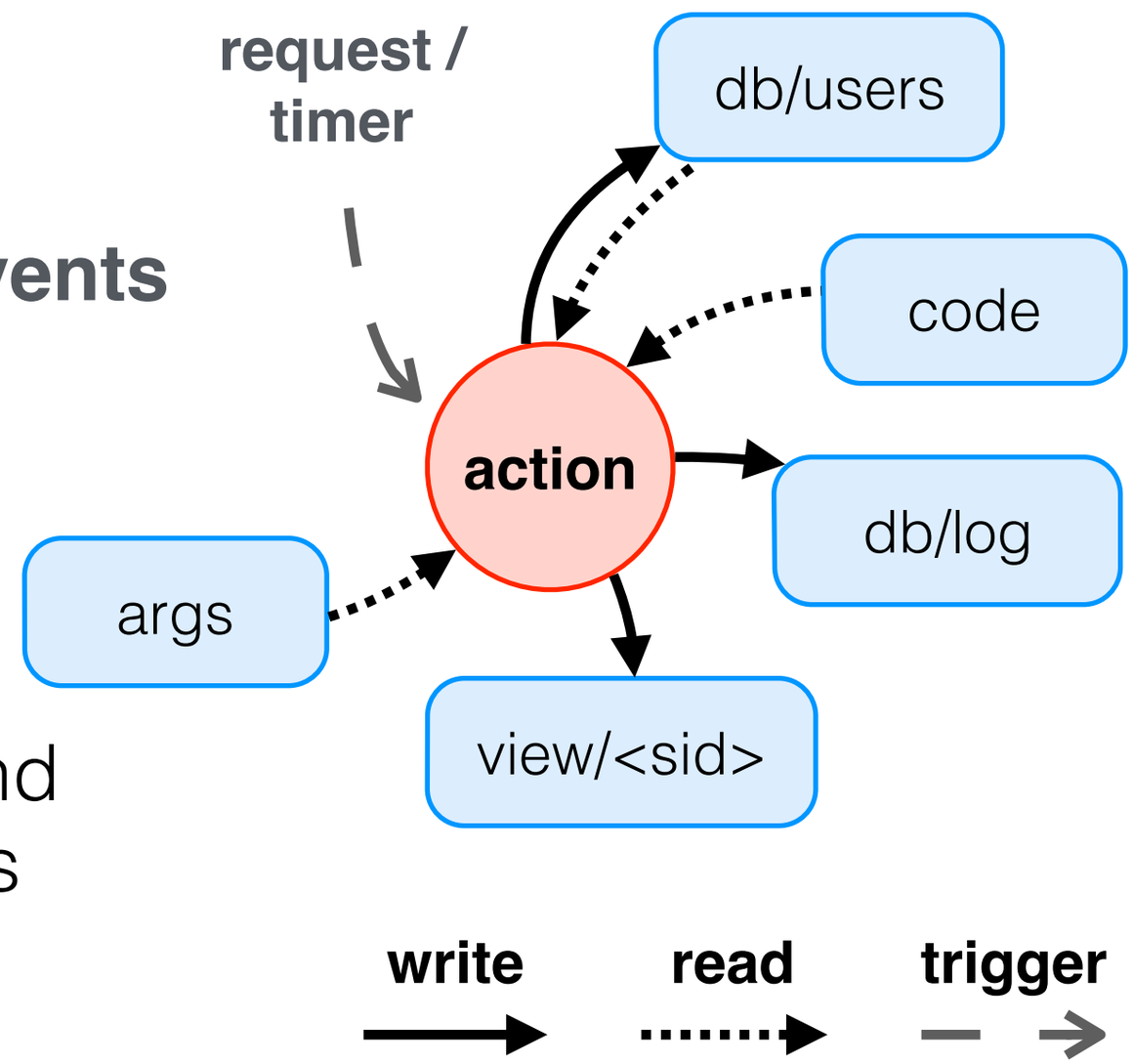
Design: action history graph (AHG)

- AHG [*OSDI '10*] tracks dependencies among **actions** and **objects**
- **Actions**
  - Triggered by external **events**
  - All application code is executed in the context of an *action*



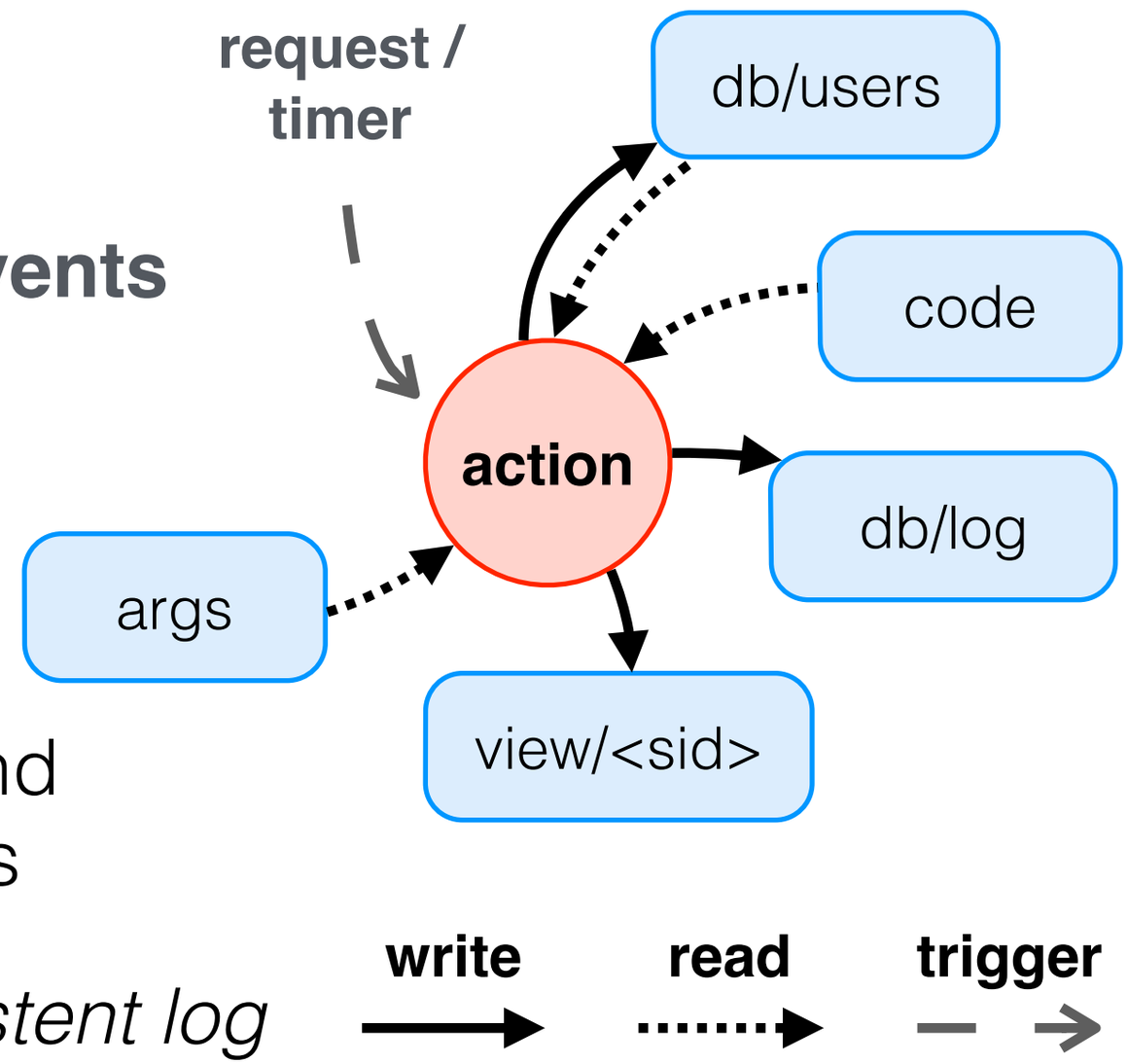
Design: action history graph (AHG)

- AHG [*OSDI '10*] tracks dependencies among **actions** and **objects**
- **Actions**
  - Triggered by external **events**
  - All application code is executed in the context of an *action*
  - Rail connects actions and objects as the code runs



Design: action history graph (AHG)

- AHG [*OSDI '10*] tracks dependencies among **actions** and **objects**
- **Actions**
  - Triggered by external **events**
  - All application code is executed in the context of an *action*
  - Rail connects actions and objects as the code runs
- Rail stores AHG in a *persistent log*

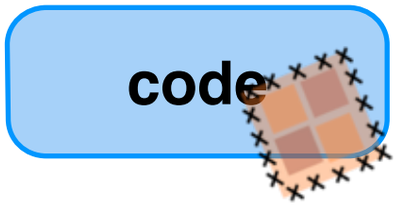


Selective replay using AHG

- Rail replays each action sequentially in the time order
- Replays an action if any of its inputs or outputs are changed

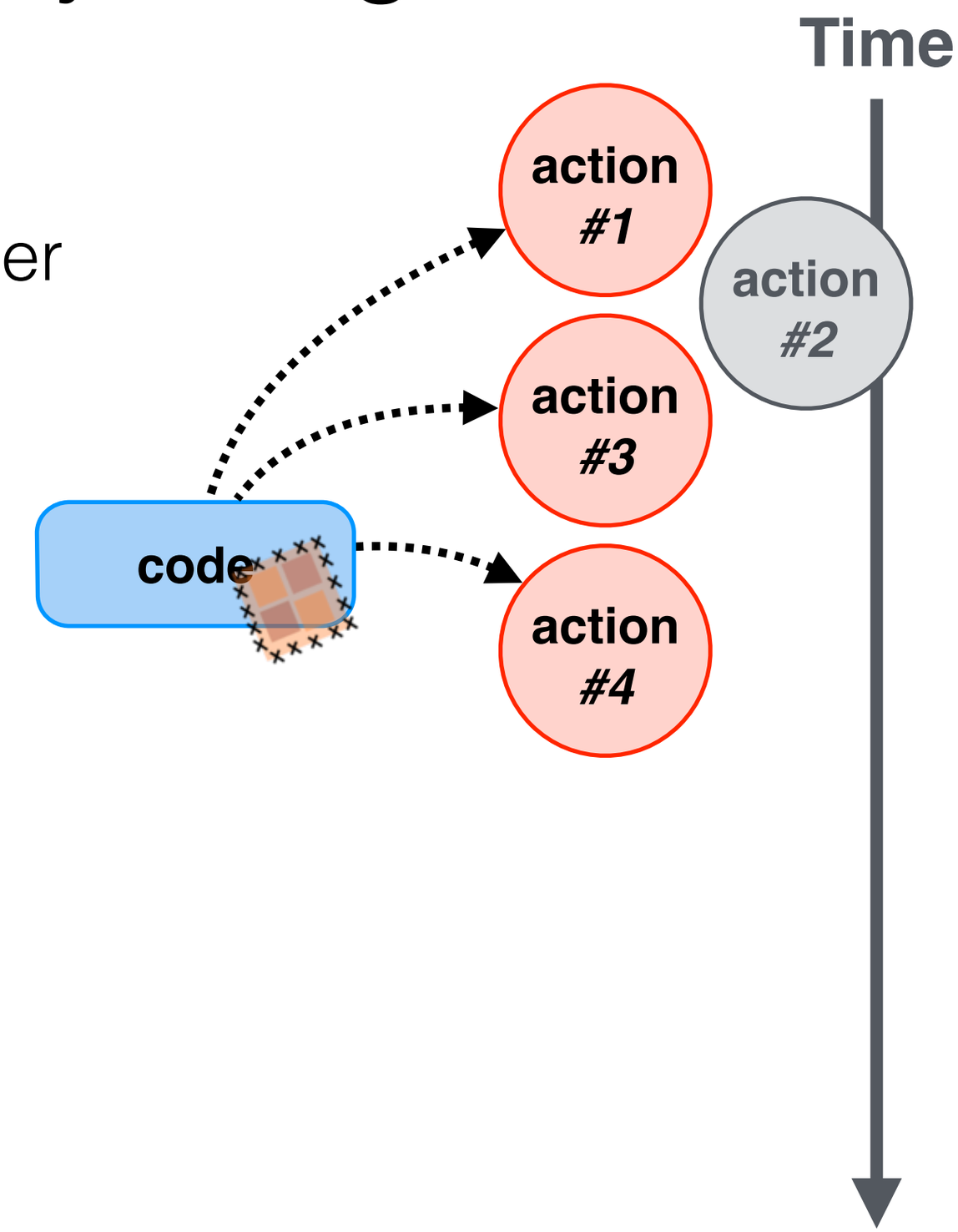
Selective replay using AHG

- Rail replays each action sequentially in the time order
- Replays an action if any of its inputs or outputs are changed



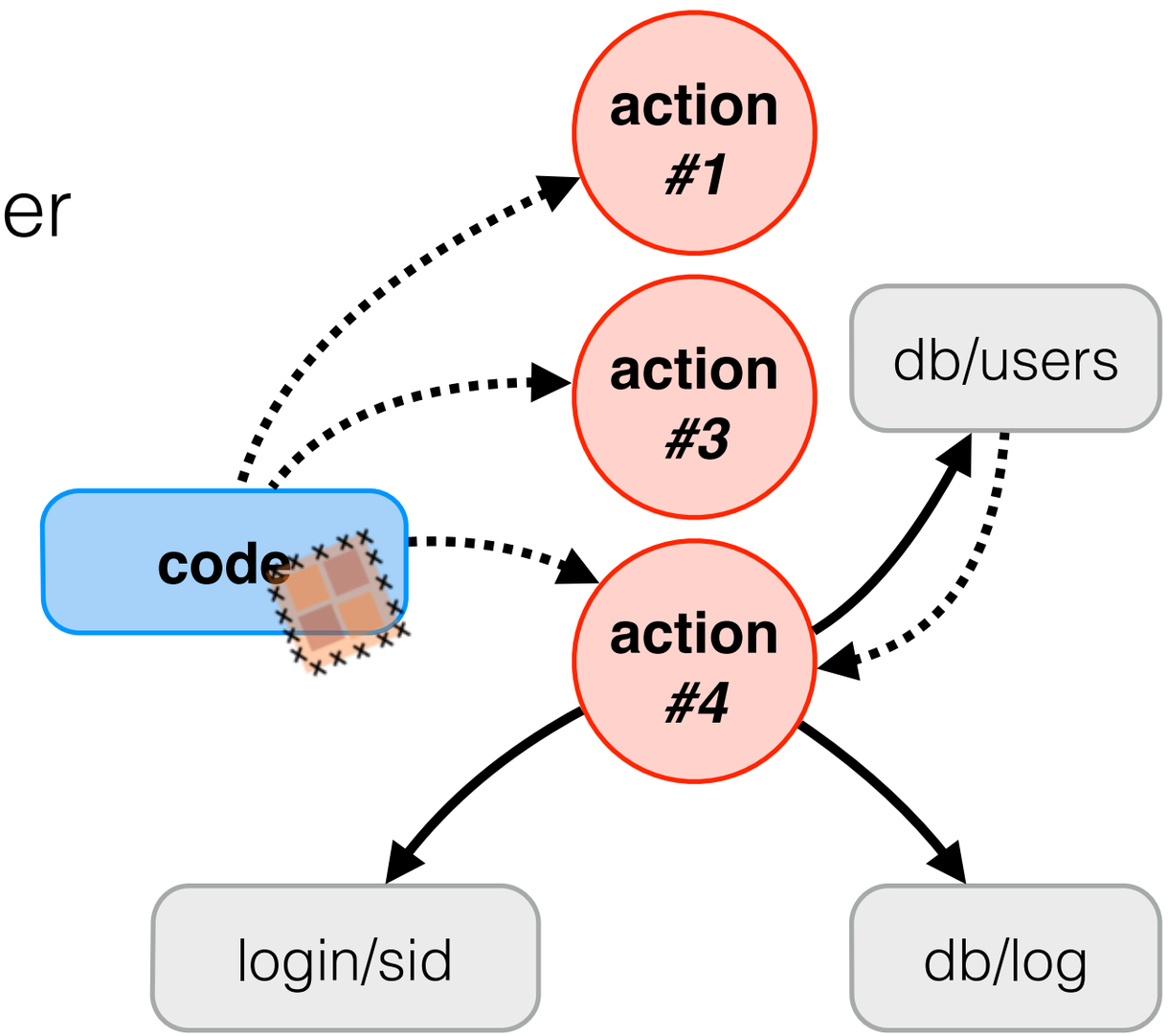
Selective replay using AHG

- Rail replays each action sequentially in the time order
- Replays an action if any of its inputs or outputs are changed



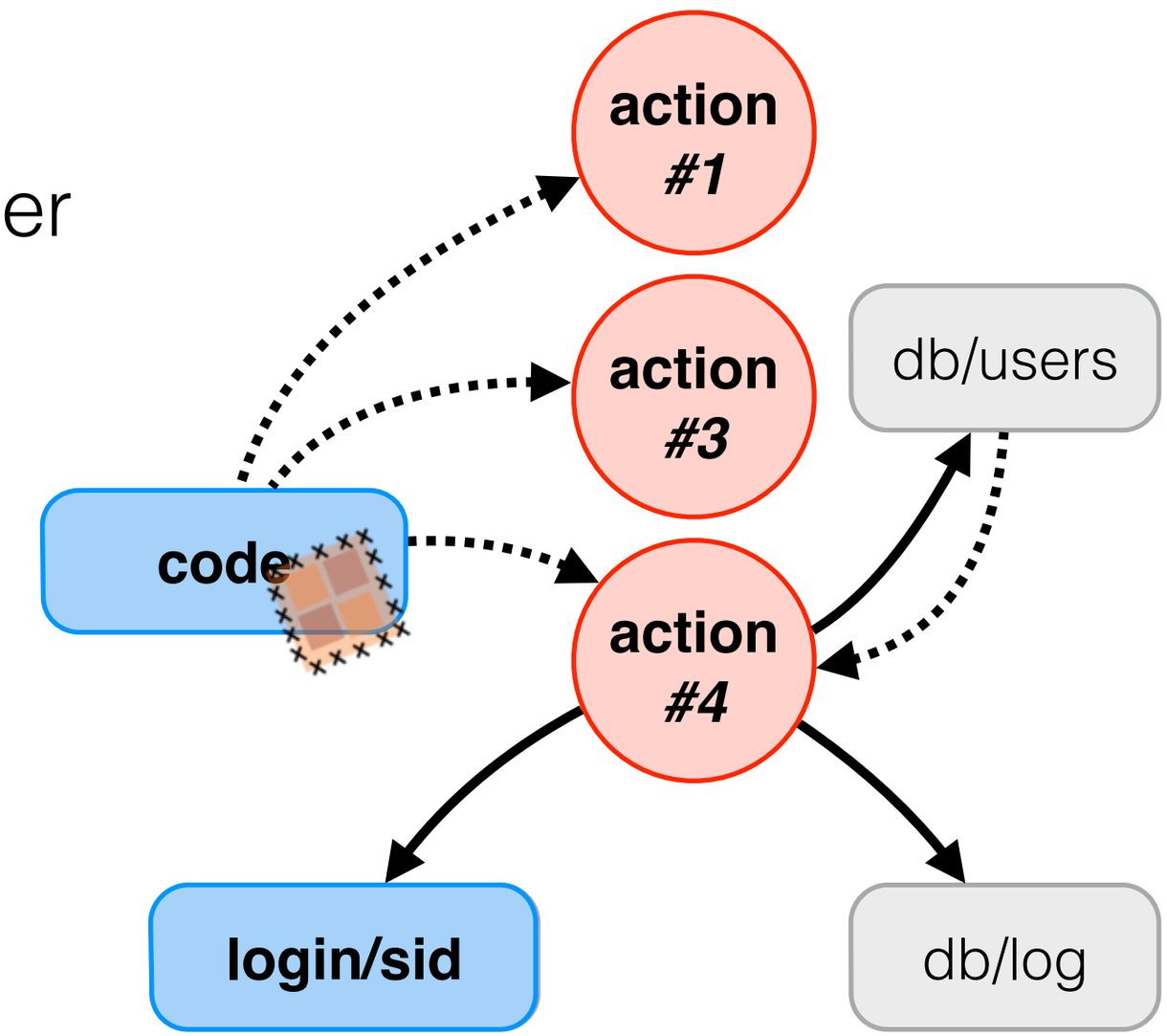
Selective replay using AHG

- Rail replays each action sequentially in the time order
- Replays an action if any of its inputs or outputs are changed



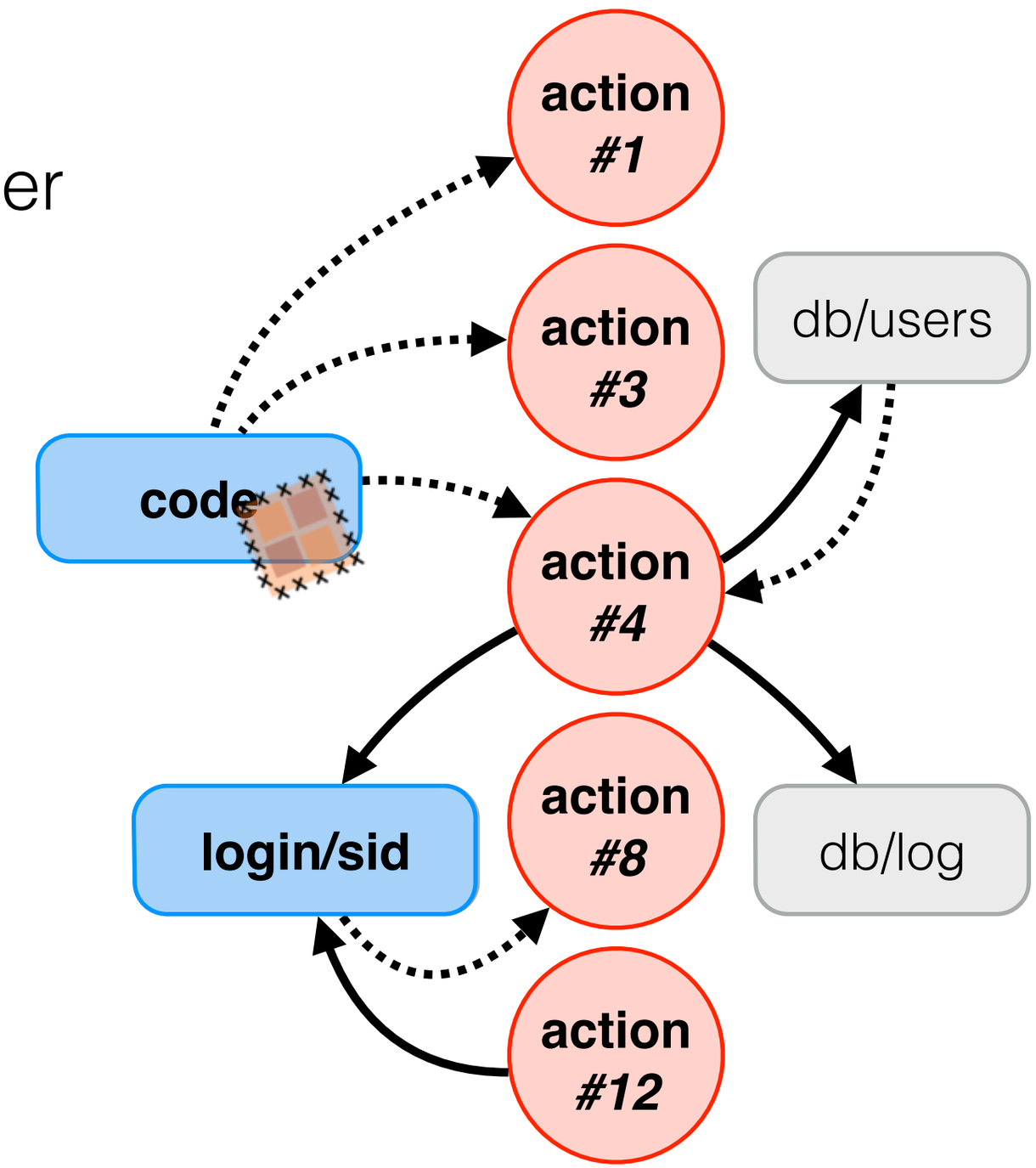
Selective replay using AHG

- Rail replays each action sequentially in the time order
- Replays an action if any of its inputs or outputs are changed



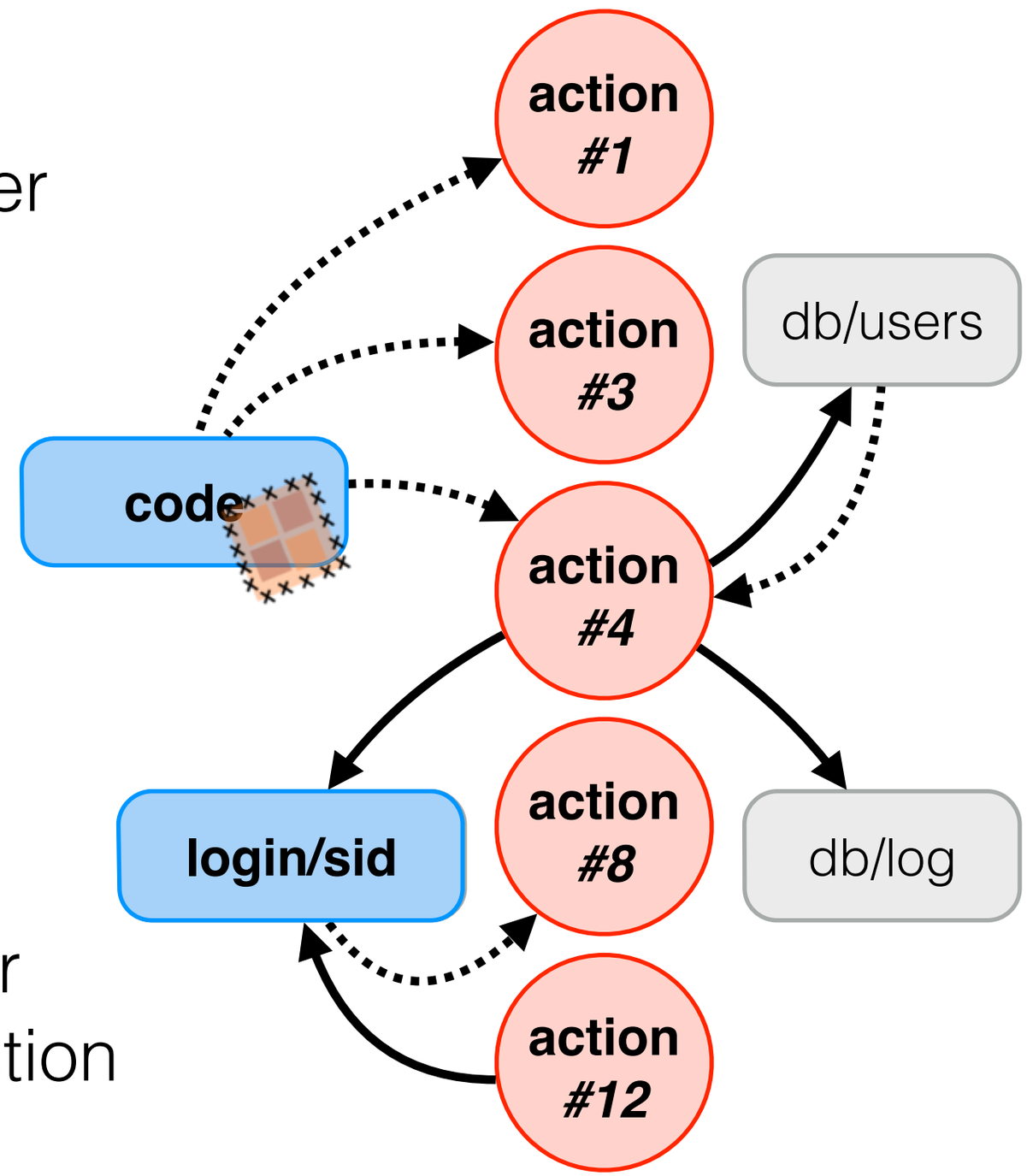
Selective replay using AHG

- Rail replays each action sequentially in the time order
- Replays an action if any of its inputs or outputs are changed



Selective replay using AHG

- Rail replays each action sequentially in the time order
- Replays an action if any of its inputs or outputs are changed
- Replay is guaranteed to terminate
  - Never runs actions earlier than current replaying action



Selective replay using Object API

- Rail must intercept all accesses to global objects
  - e.g., inputs, outputs, database items, session states, …

Selective replay using Object API

- Rail must intercept all accesses to global objects
  - e.g., inputs, outputs, database items, session states, …
- Reasons
  - to **track dependency** between actions
  - to make **continuous checkpoints** of object states

Selective replay using Object API

- Developer must wrap all global objects in the app using Rail's **object API**

Selective replay using Object API

- Developer must wrap all global objects in the app using Rail's **object API**

Selective replay using Object API

- Developer must wrap all global objects in the app using Rail's **object API**

- Global objects are quite standard in all web apps.

Selective replay using Object API

- Developer must wrap all global objects in the app using Rail's **object API**

- Global objects are quite standard in all web apps.
- Most wrappers can be done **once** in the framework

Example: homework submission

```
// Server side code
var Homeworks = App.getDBCollection('hws');
var Answers = App.getDBCollection('answers');

App.method('submit', function (hw_id, answer) {
  var uid = App.getSessionUserId();
  var hw = Homework.findOne( {_id: hw_id} );

  if (!uid || !hw || hw.dueDate < (new Date))
    throw new Error('Submission failed');
  Answers.insert( {_id: Math.random(),
    hw: hw_id, user: uid, answer: answer} );
});
```

Example: homework submission

```
// Server side code
var Homeworks = App.getDBCollection('hws');
var Answers = App.getDBCollection('answers');

App.method('submit', function (hw_id, answer) {
  var uid = App.getSessionUserId();
  var hw = Homework.findOne( {_id: hw_id} );

  if (!uid || !hw || hw.dueDate < (new Date))
    throw new Error('Submission failed');
  Answers.insert( {_id: Math.random(),
    hw: hw_id, user: uid, answer: answer} );
});
```

Example: homework submission

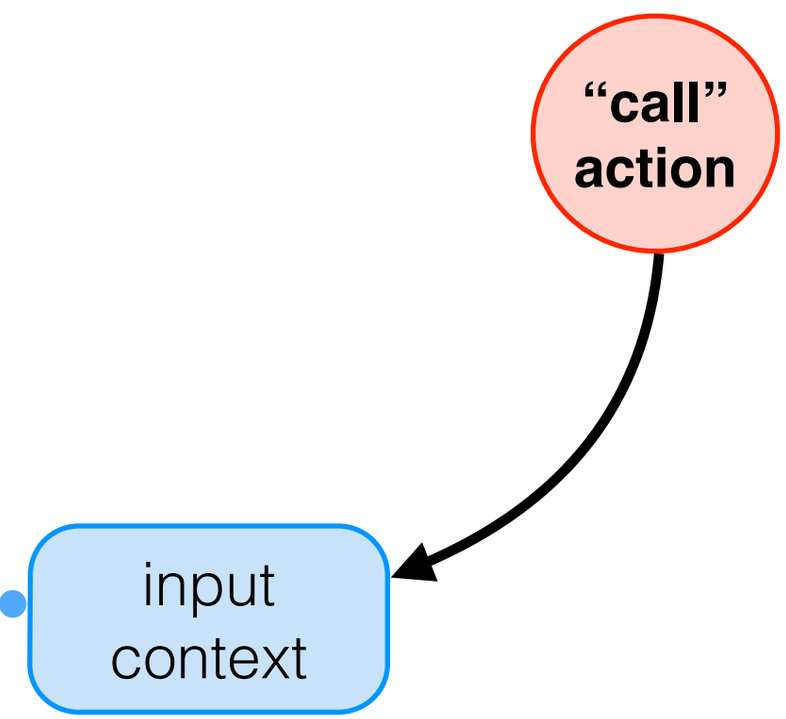
```
// Server side code using Rail API
var Homeworks = App.getDBCollection('hws');
var Answers = App.getDBCollection('answers');

App.method('submit', function (hw_id, answer) {
  var uid = App.getSessionUserId();
  var hw = Homework.findOne( {_id: hw_id} );
  var ctx = Rail.inputContext(hw_id, uid);
  if (!uid || !hw || hw.dueDate < ctx.date())
    throw new Error('Submission failed');
  Answers.insert( {_id: ctx.random(),
    hw: hw_id, user: uid, answer: answer} );
});
```

Example: homework submission

```
// Server side code using Rail API
var Homeworks = App.getDBCollection('hws');
var Answers = App.getDBCollection('answers');

App.method('submit', function (hw_id, answer) {
  var uid = App.getSessionUserId();
  var hw = Homework.findOne( {_id: hw_id} );
  var ctx = Rail.inputContext(hw_id, uid);
  if (!uid || !hw || hw.dueDate < ctx.date())
    throw new Error('Submission failed');
  Answers.insert( {_id: ctx.random(),
    hw: hw_id, user: uid, answer: answer} );
});
```

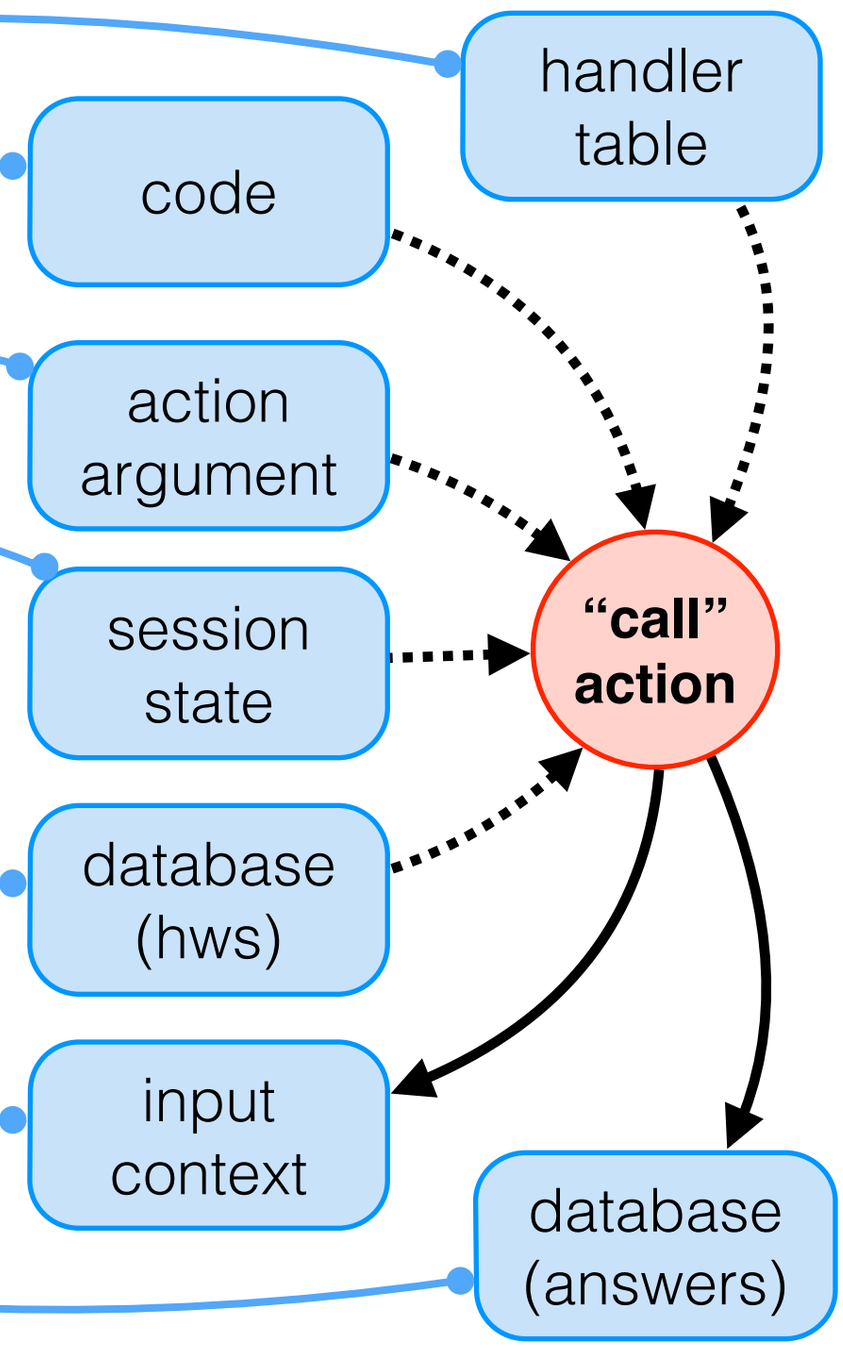
Example: homework submission

```
// Server side code using Rail API
var Homeworks = App.getDBCollection('hws');
var Answers = App.getDBCollection('answers');

App.method('submit', function (hw_id, answer) {
  var uid = App.getSessionUserId();
  var hw = Homework.findOne( {_id: hw_id} );
  var ctx = Rail.inputContext(hw_id, uid);
  if (!uid || !hw || hw.dueDate < ctx.date())
    throw new Error('Submission failed');
  Answers.insert( {_id: ctx.random(),
    hw: hw_id, user: uid, answer: answer} );
});
```



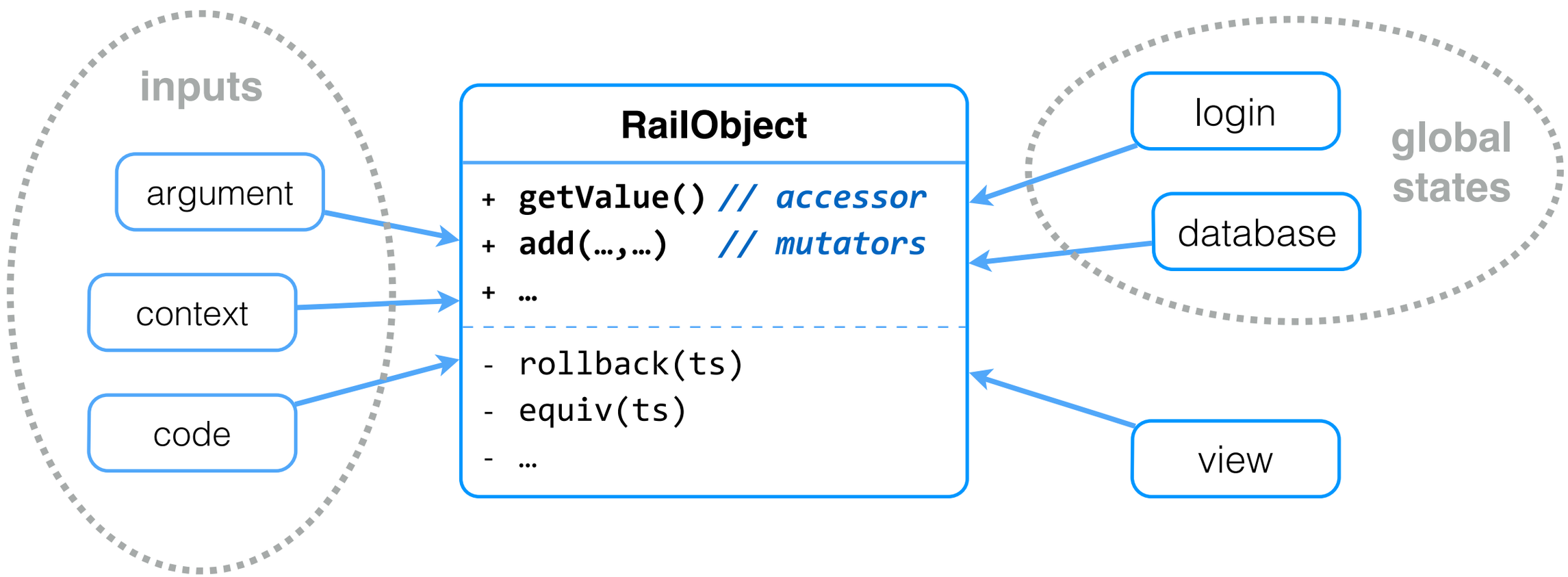
Example: homework submission

```
// Server side code using Rail API
var Homeworks = App.getDBCollection('hws');
var Answers = App.getDBCollection('answers');

App.method('submit', function (hw_id, answer) {
  var uid = App.getSessionUserId();
  var hw = Homework.findOne( {_id: hw_id} );
  var ctx = Rail.inputContext(hw_id, uid);
  if (!uid || !hw || hw.dueDate < ctx.date())
    throw new Error('Submission failed');
  Answers.insert( {_id: ctx.random(),
    hw: hw_id, user: uid, answer: answer} );
});
```



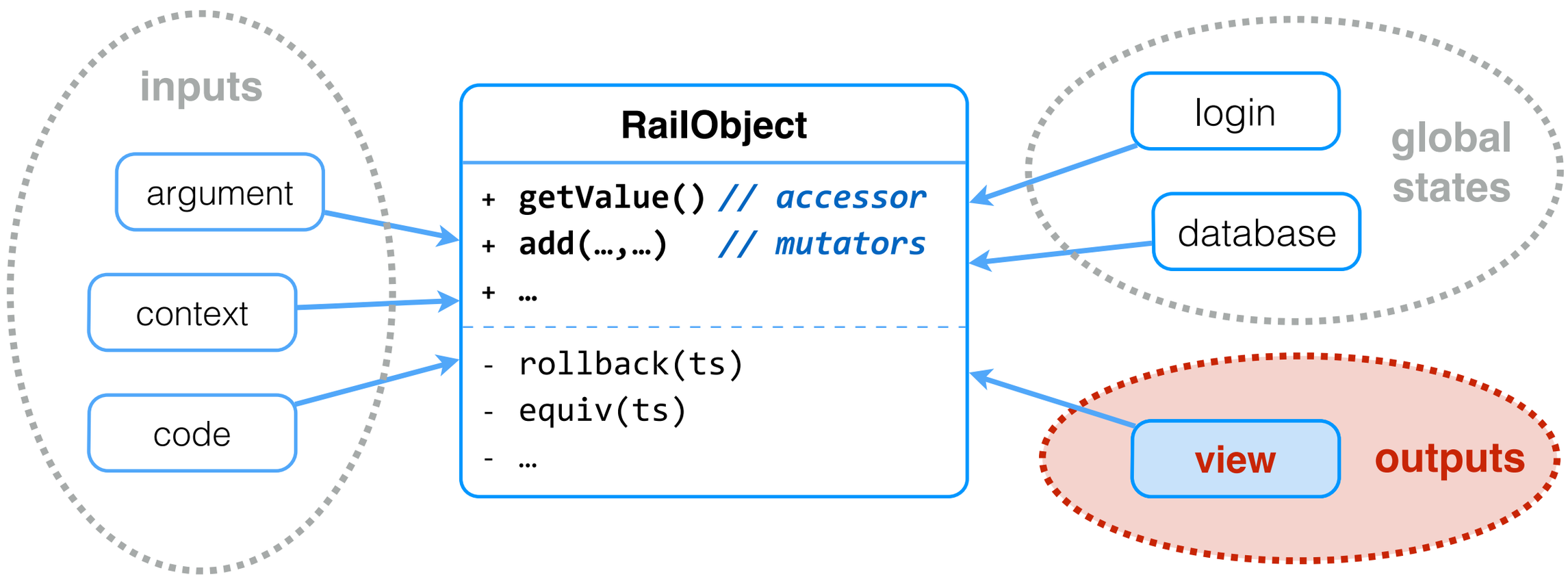
Uniform Object API

- Rail provides a *uniform* API for different types of objects
- Rail takes care of dependency tracking and checkpointing



Uniform Object API

- Rail provides a *uniform* API for different types of objects
- Rail takes care of dependency tracking and checkpointing



Tracking data items in output

Tracking data items in output

- Rail maintains a **view object** for every session
  - tracks all data items sent to the client

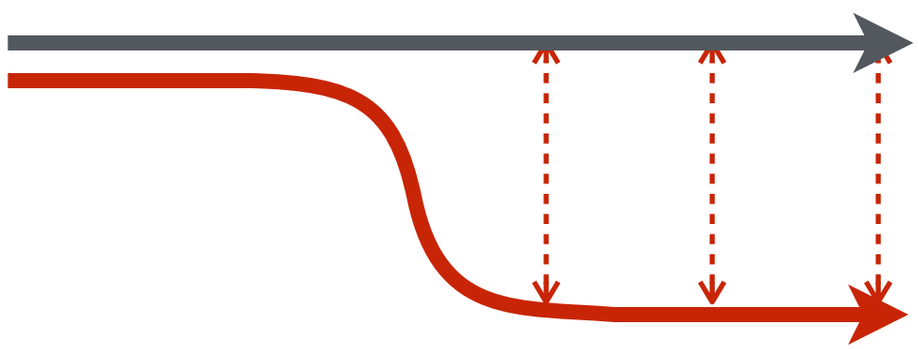
Tracking data items in output

- Rail maintains a **view object** for every session
  - tracks all data items sent to the client
- To do output book-keeping, one adds objects to the view
  - e.g., `view.add(“db/users/admin”, {“name”, “email”});`
  - change the template rendering system

Tracking data items in output

- Rail maintains a **view object** for every session
  - tracks all data items sent to the client
- To do output book-keeping, one adds objects to the view
  - e.g., `view.add("db/users/admin", {"name", "email"});`
  - change the template rendering system
- During replay, Rail reruns actions and re-compute the view objects for every session
  - if *old_view* – *new_view* ≠ ∅ ➜ **Breach!**

Replay with non-deterministic inputs



Goal: minimize state divergence

Replay with non-deterministic inputs

```
App.method('populate_admins', function () {
  var admins = ['Alice', 'Mallory', 'Bob'];

  for (var i = 0; i < admins.length; ++i) {
    var pwd = Math.random();
    Users.insert({name: admins[i], passwd: pwd});
  }
});
```

Replay with non-deterministic inputs

- How to handle randomness during replay?

```
App.method('populate_admins', function () {
- var admins = ['Alice', 'Mallory', 'Bob'];
+ var admins = ['Alice', 'Bob'];
  for (var i = 0; i < admins.length; ++i) {
    var pwd = Math.random();
    Users.insert({name: admins[i], passwd: pwd});
  }
});
```

Replay with non-deterministic inputs

- How to handle randomness during replay?
  - Strawman 1: return a new random number ✗

```
App.method('populate_admins', function () {
- var admins = ['Alice', 'Mallory', 'Bob'];
+ var admins = ['Alice', 'Bob'];
  for (var i = 0; i < admins.length; ++i) {
    var pwd = Math.random();
    Users.insert({name: admins[i], passwd: pwd});
  }
});
```

Replay with non-deterministic inputs

- How to handle randomness during replay?
  - Strawman 1: return a new random number ✗
  - Strawman 2: log and return random numbers in order ✗

```
App.method('populate_admins', function () {
- var admins = ['Alice', 'Mallory', 'Bob'];
+ var admins = ['Alice', 'Bob'];
  for (var i = 0; i < admins.length; ++i) {
    var pwd = Math.random();
    Users.insert({name: admins[i], passwd: pwd});
  }
});
```

Stabilize non-deterministic inputs with context identifiers

- To avoid false report, Rail must reconcile state divergence of two executions w.r.t. non-deterministic inputs

Stabilize non-deterministic inputs with context identifiers

- To avoid false report, Rail must reconcile state divergence of two executions w.r.t. non-deterministic inputs

- Solution: use **input context** object to access non-deterministic input

Stabilize non-deterministic inputs with context identifiers

- To avoid false report, Rail must reconcile state divergence of two executions w.r.t. non-deterministic inputs

- Solution: use **input context** object to access non-deterministic input

```
App.method('populate_admins', function () {
  var admins = ['Alice', 'Mallory', 'Bob'];
  for (var i = 0; i < admins.length; ++i) {
-    var pwd = Math.random();
+    var pwd = Rail.inputContext(
+        'populate', admins[i]).random();
     Users.insert({name: admins[i],
                   passwd: pwd});
  }
});
```

Stabilize non-deterministic inputs with context identifiers

- To avoid false report, Rail must reconcile state divergence of two executions w.r.t. non-deterministic inputs
- Solution: use **input context** object to access non-deterministic input
  - developer supplies a *stable* **context ID**
  - during replay: same context ID ➜ return same value

```
App.method('populate_admins', function () {
  var admins = ['Alice', 'Mallory', 'Bob'];
  for (var i = 0; i < admins.length; ++i) {
-    var pwd = Math.random();
+    var pwd = Rail.inputContext(
+        'populate', admins[i]).random();
     Users.insert({name: admins[i],
                   passwd: pwd});
  }
});
```

Stabilize non-deterministic inputs with context identifiers

- To avoid false report, Rail must reconcile state divergence of two executions w.r.t. non-deterministic inputs
- Solution: use **input context** object to access non-deterministic input
  - developer supplies a *stable* **context ID**
  - during replay: same context ID ➜ return same value

```
App.method('populate_admins', function () {
  var admins = ['Alice', 'Mallory', 'Bob'];
  for (var i = 0; i < admins.length; ++i) {
-    var pwd = Math.random();
+    var pwd = Rail.inputContext(
+        'populate', admins[i]).random();
     Users.insert({name: admins[i],
                   passwd: pwd});
  }
});
```

Other issues

- How to port other web frameworks to support Rail?
  - e.g., Django, Ruby, etc.
- How to choose context identifiers?
- What if developers misuse Rail API?
- … …

Evaluation

Benchmarks

Application	Description
Submit	homework grading
EndoApp	medical survey
Telescope	social news (open source)

Benchmarks

Application	Description	Attack workload
Submit	homework grading	**ACL error**: administrator erroneously granting "*staff*" privilege to a student
EndoApp	medical survey	**Stolen password**: attacker creating a malicious root account using a surgeon's weak password
Telescope	social news (open source)	**Code bugs**: permission checks based on client-supplied user ID —— a *real bug in commit history*

Porting applications to Rail is easy

LOC in JavaScript (only server-side code is changed)

Application	Changed	Server	Client
Submit	**24**	**769**	891
EndoApp	**2**	**599**	900
Telescope	**20**	**1,169**	1,781

- Most of the changes are related to non-deterministic inputs

Porting applications to Rail is easy

LOC in JavaScript (only server-side code is changed)

Application	Changed	Server	Client
Submit	**24**	**769**	891
EndoApp	**2**	**599**	900
Telescope	**20**	**1,169**	1,781

- Most of the changes are related to non-deterministic inputs
- Framework wrappers (422 lines in Meteor) offload most burdens from the application developer

Rail is more precise than access log based approaches

# of data items (run with benign workloads in the background)

Workload	Accessed	Reported	Missed	False
ACL error (Submit)	1,121	193	0	0
Stolen password (EndoApp)	3,521	197	0	1
Code bugs (Telescope)	23	10	0	0

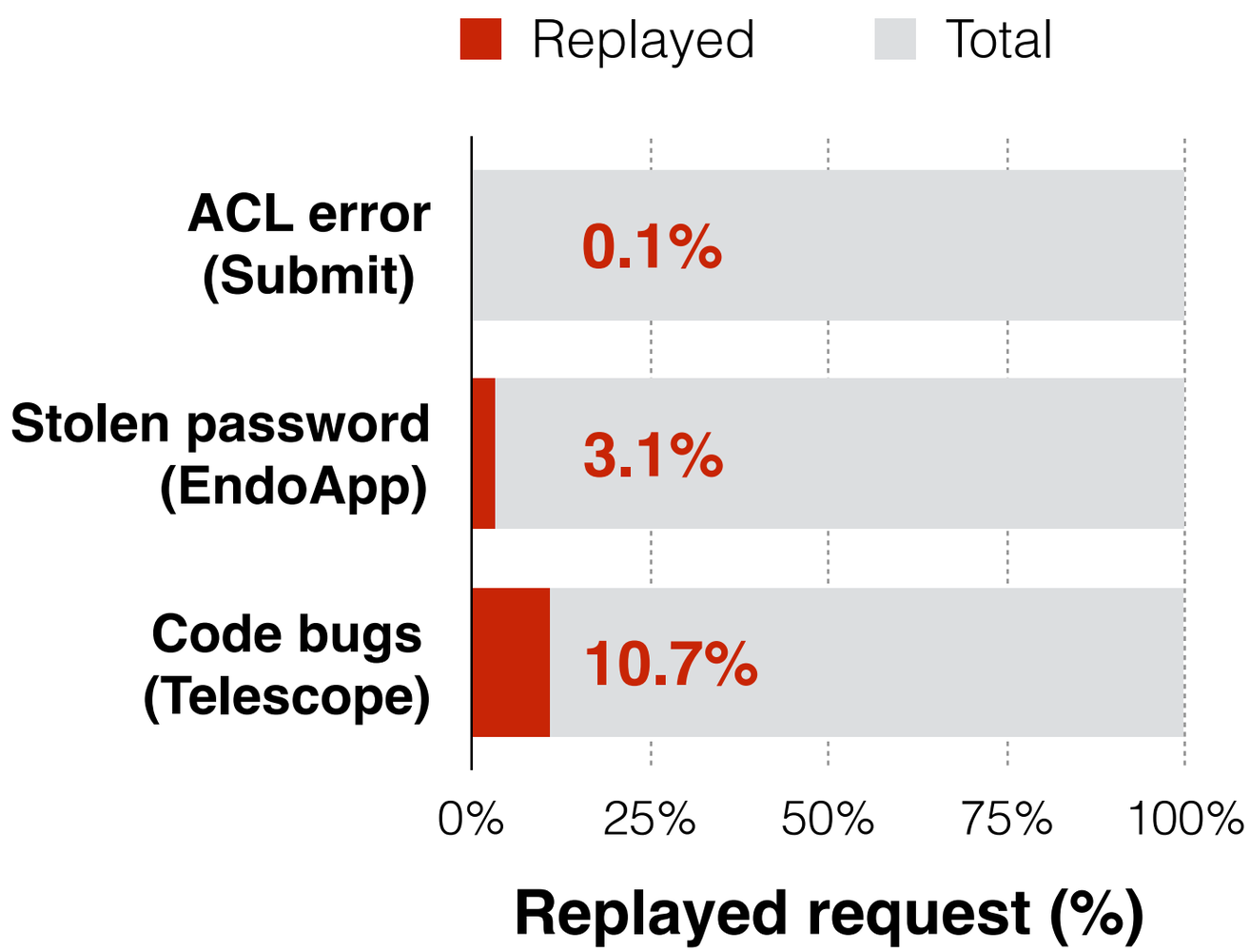
Rail is more precise than access log based approaches

# of data items (run with benign workloads in the background)

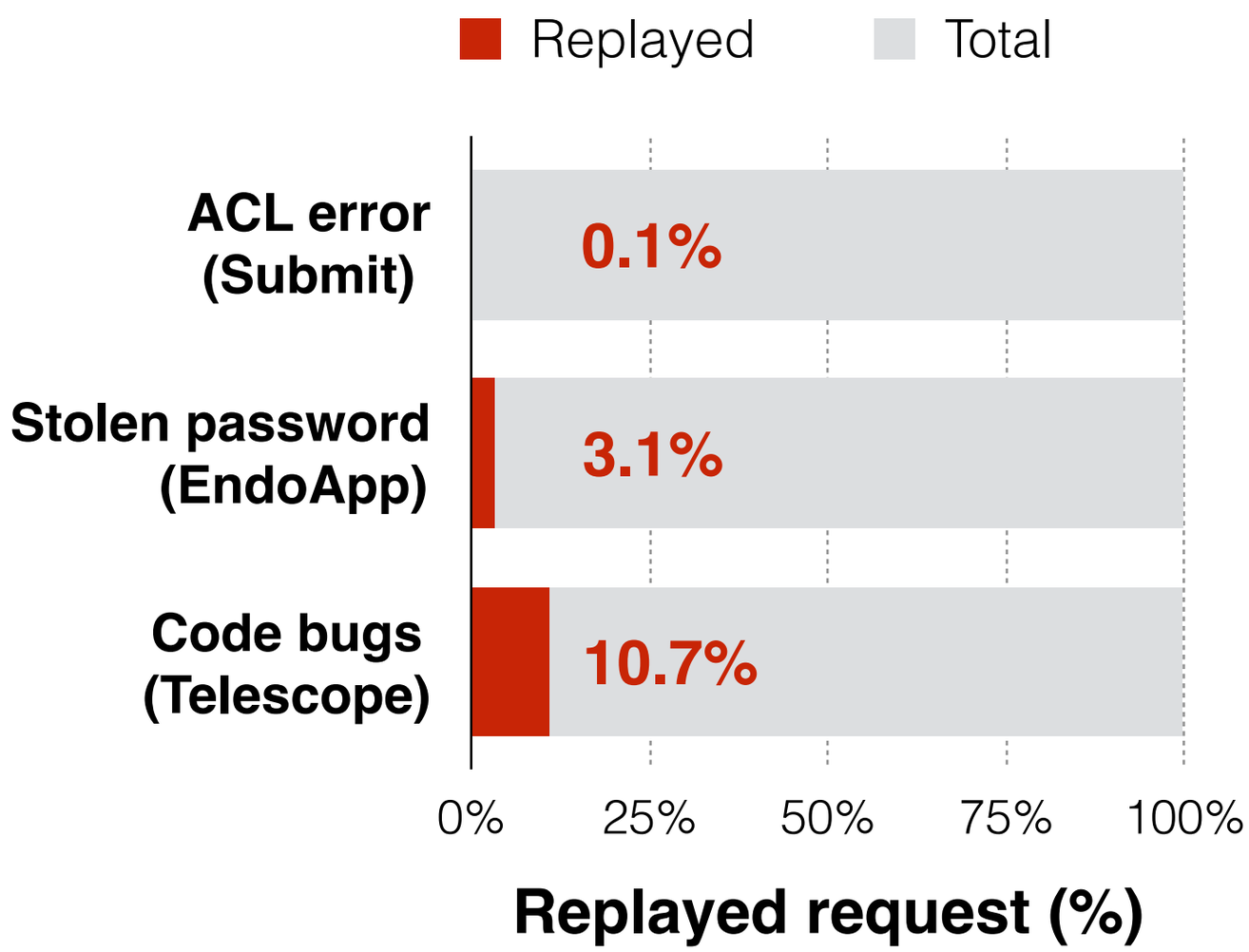
Workload	Accessed	Reported	Missed	False
ACL error (Submit)	1,121	193	0	0
Stolen password (EndoApp)	3,521	197	0	1
Code bugs (Telescope)	23	10	0	0

The malicious account created by the attacker (not a "breach", but related to the attack)

Rail replays only relevant requests

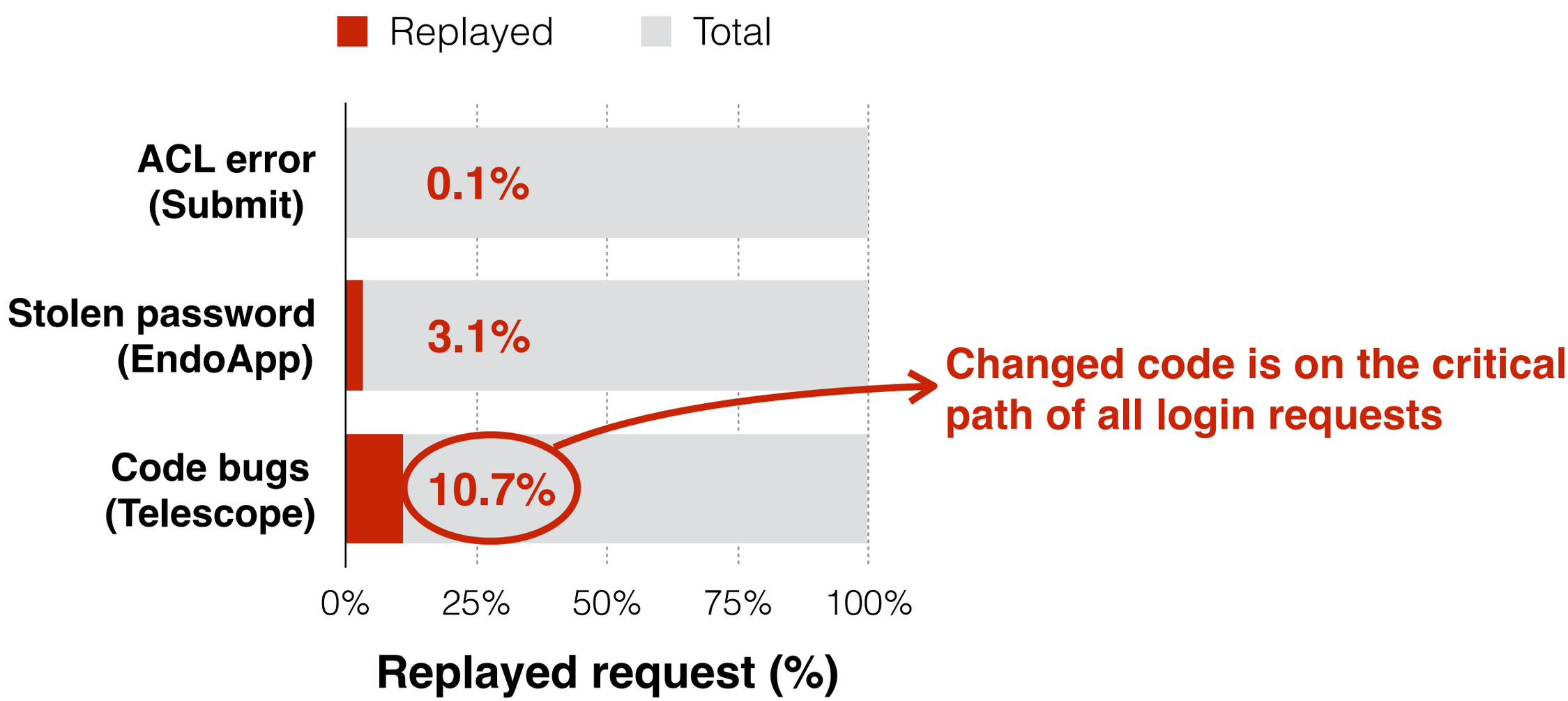


Rail replays only relevant requests



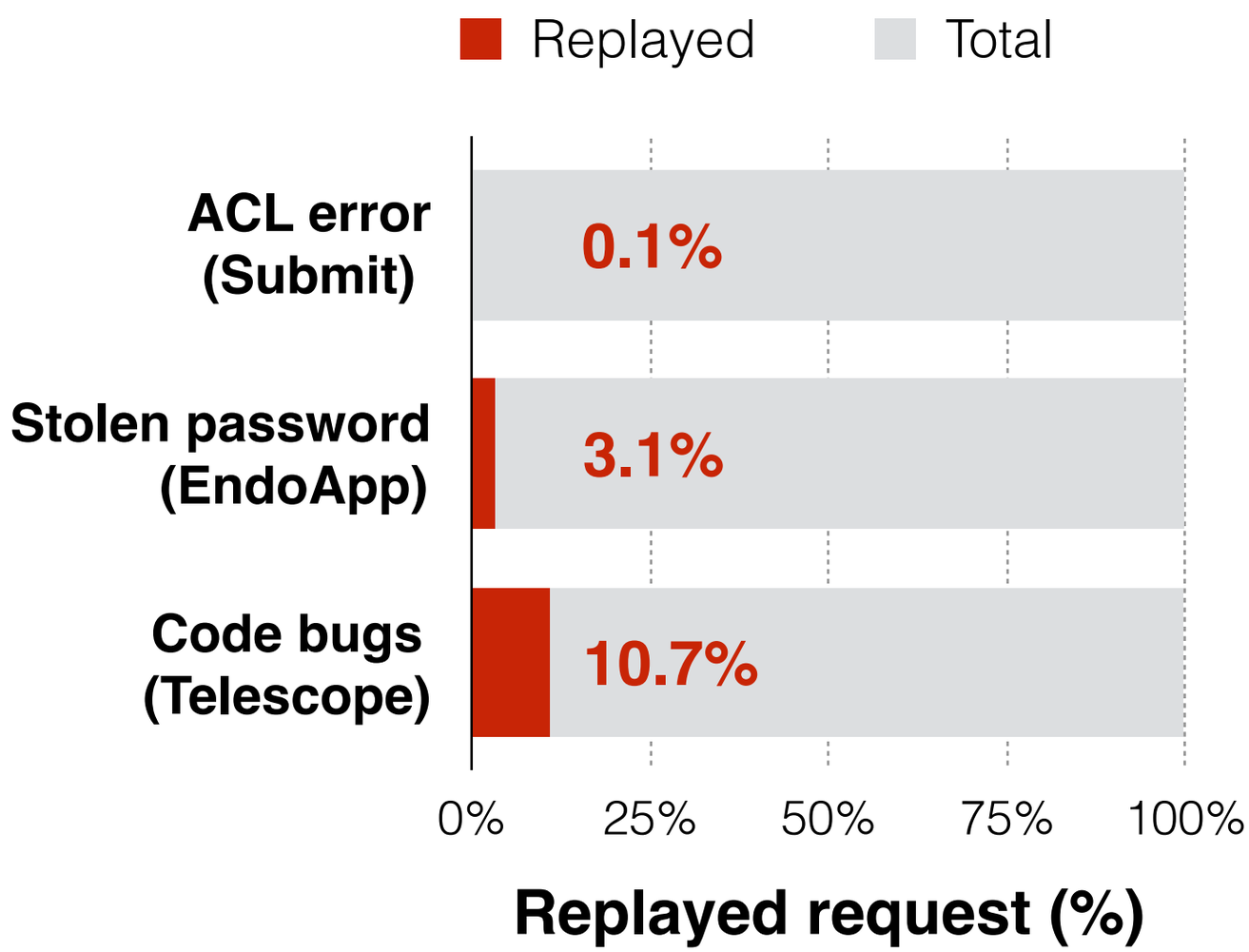
- **Rail replays only a small fraction of original requests that are related to the attack**

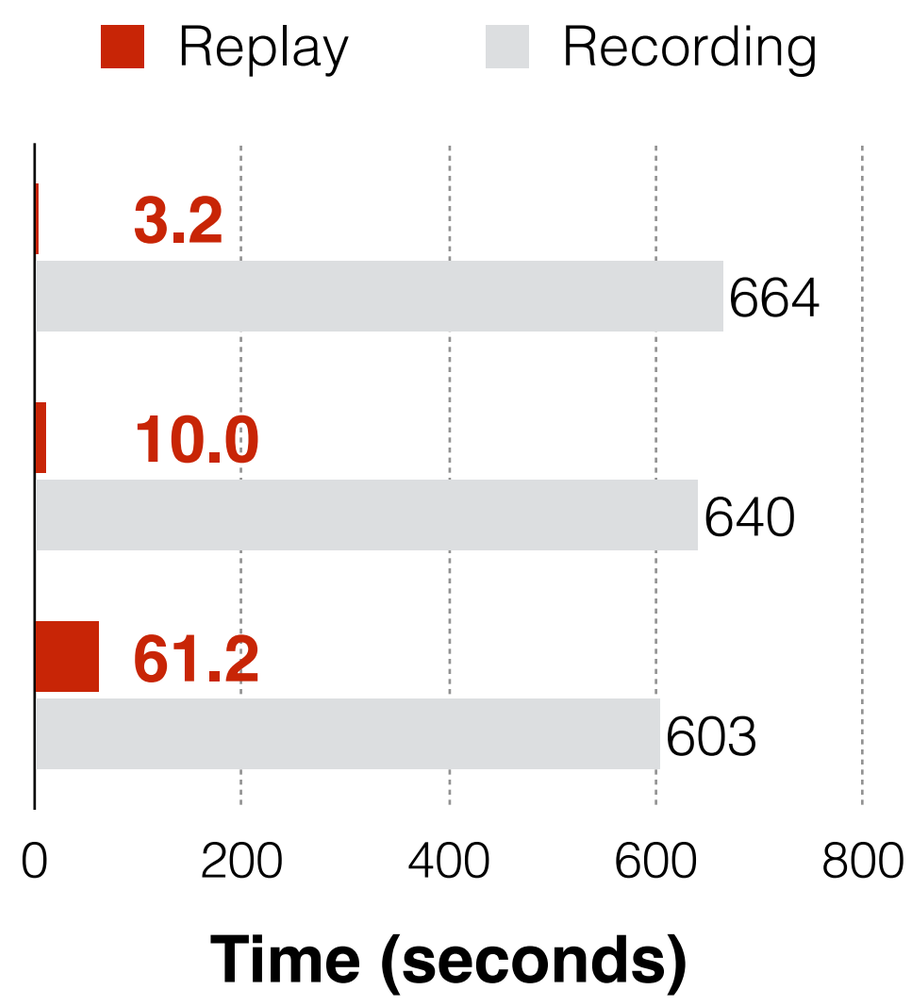
Rail replays only relevant requests



- **Rail replays only a small fraction of original requests that are related to the attack**

Rail replays only relevant requests

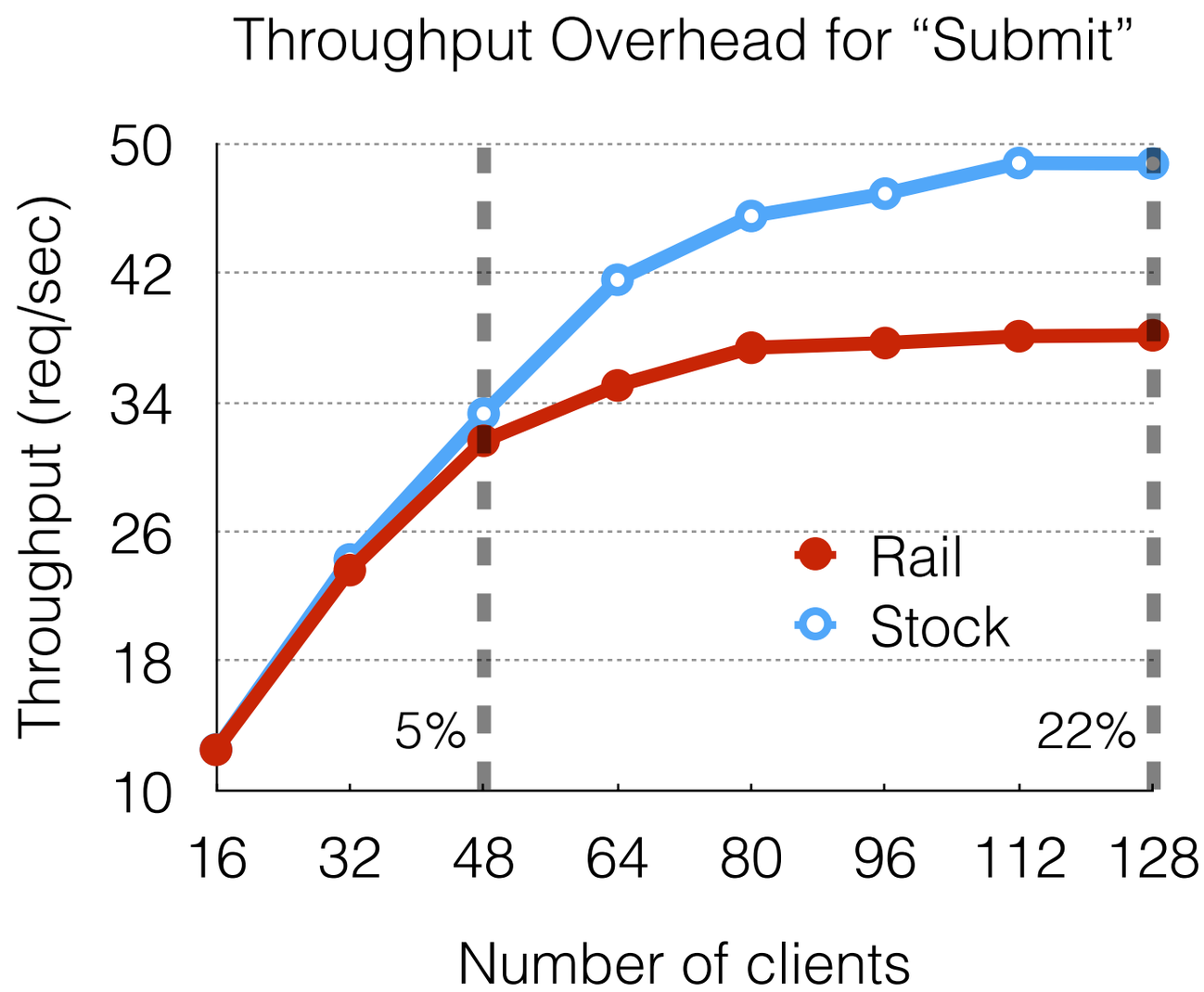




- **Rail replays only a small fraction of original requests that are related to the attack**
- **Replay time is proportional to the number of replayed requests**

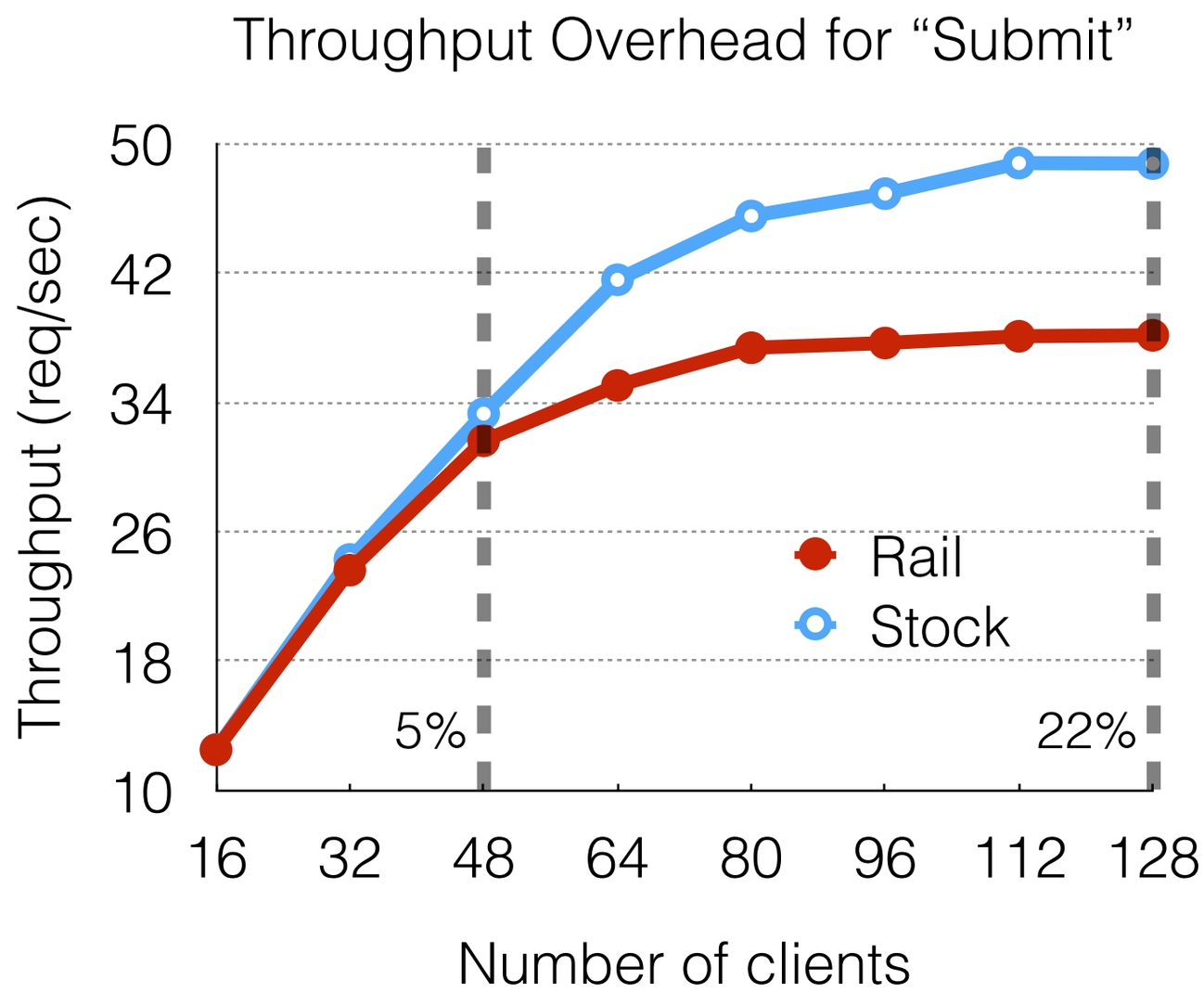
Rail's recording overhead is moderate

- **Performance**
  - 5% for an under-loaded server (< 90% CPU utilization)
  - 22% for an over-loaded server



Rail's recording overhead is moderate

- **Performance**
  - 5% for an under-loaded server (< 90% CPU utilization)
  - 22% for an over-loaded server
- **Storage**
  - ~ 0.5KB / request
  - or 500GB / year for a full-loaded server



Related Work

- Record and replay
  - *Recovery*: **Retro** [OSDI ’10]; **Warp** [SOSP ’11]
  - *Auditing*: **Rad** [APSys ’11]; **Poirot** [OSDI ’12]
- Detecting data breaches
  - *Access log*: **Keypad** [EuroSys ’11]; **Pasture** [OSDI ’12]
  - *Information flow*: **TightLip** [NSDI ’07]; **TaintDroid** [OSDI ’10]

Conclusion

- Rail can precisely identify breached data items *after* a disclosure in web applications
- Provides developers with APIs that help to identify data items, track dependencies, and match up states
- Requires few changes to applications
- Precise, efficient, and practical

Questions?