Pip: Detecting the Unexpected in Distributed Systems

Patrick Reynolds, Charles Killian, Janet Wiener,
Jeffrey Mogul, Mehul Shah

Presented by:
Aaron Schulman

Some slides and images were borrowed from NSDI Talk

Introduction

- Distributed systems are complex
  - Harder to debug than centralized systems
- Often, deviations from expected behavior indicate bugs
  - Characterize system behavior

Distributed Systems (DS) are hard to test

- Distributed systems are subject to:
  - Independent node failures
  - Incorrect synchronization of parallel tasks
  - Network errors
  - Security Breaches

Goal

Develop tool to aid DS programmers by identifying bugs.

DS testing tools

DS testing tools

Approach	Scenario

DS testing tools

Approach	Scenario
gdb and gprof	low level bugs at a single node / core dumps

DS testing tools

Approach	Scenario
gdb and gprof	low level bugs at a single node / core dumps
black box testing	enough consistency to do statistical analysis

DS testing tools

Approach	Scenario
gdb and gprof	low level bugs at a single node / core dumps
black box testing	enough consistency to do statistical analysis
model checking	small system with hard to reproduce bug

DS testing tools

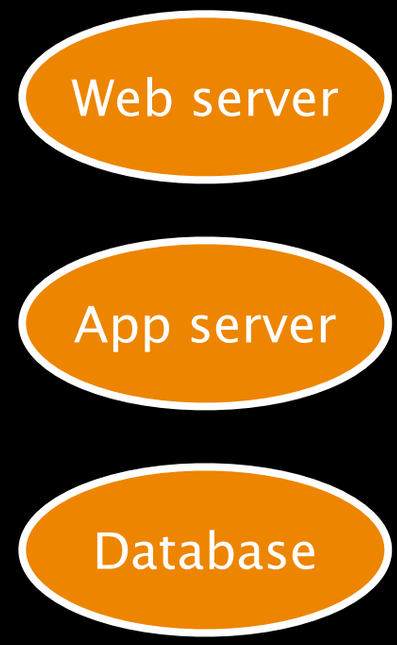
Approach	Scenario
gdb and gprof	low level bugs at a single node / core dumps
black box testing	enough consistency to do statistical analysis
model checking	small system with hard to reproduce bug
printf	bugs detected with local log

DS testing tools

Approach	Scenario
gdb and gprof	low level bugs at a single node / core dumps
black box testing	enough consistency to do statistical analysis
model checking	small system with hard to reproduce bug
printf	bugs detected with local log
Causal Path Analysis	distributed system does not behave as expected

Causal Path Analysis

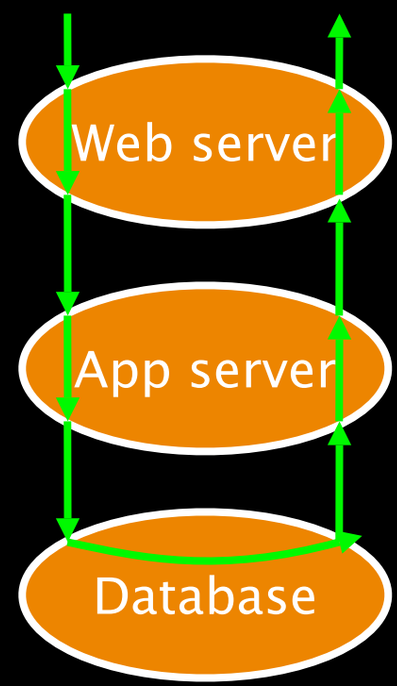
Example:
Web service system



Causal Path Analysis

- Path is caused by input to system
- e.g. user request from web service

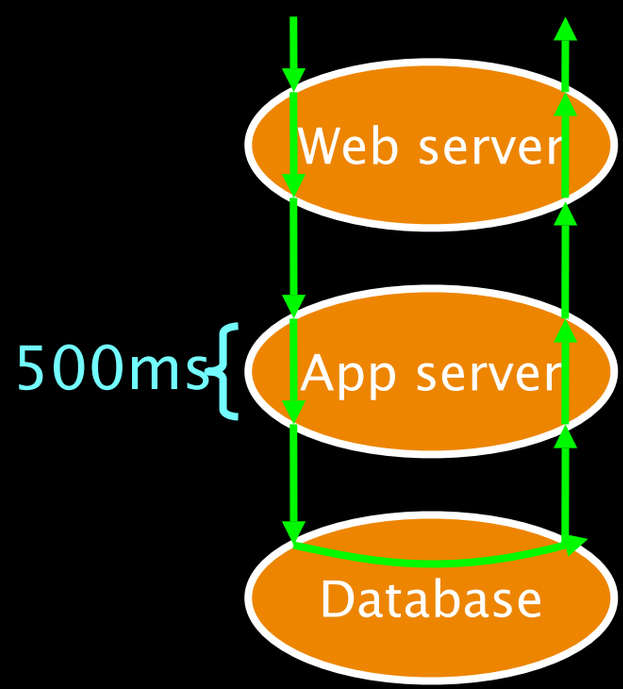
Example:
Web service system



Causal Path Analysis

- Path is caused by input to system
  - e.g. user request from web service
- Components delay

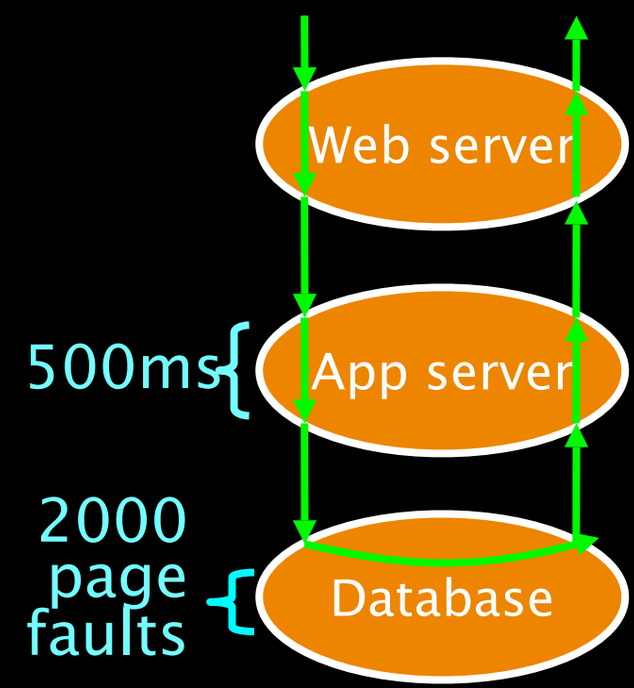
Example:
Web service system



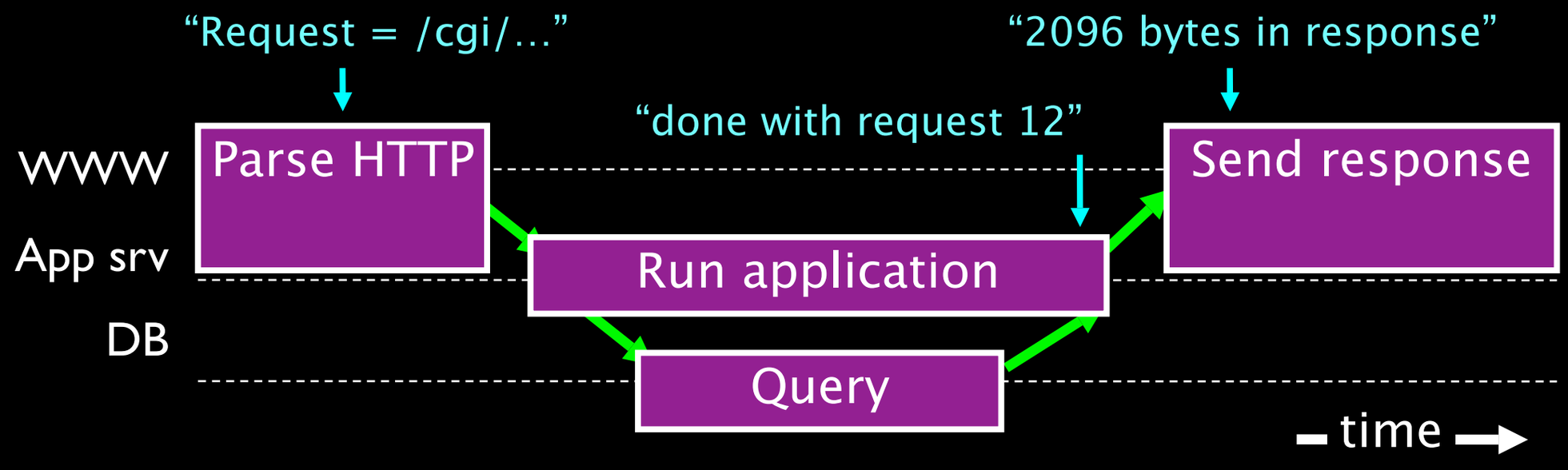
Causal Path Analysis

- Path is caused by input to system
  - e.g. user request from web service
- Components delay
- Attribution of resource consumption

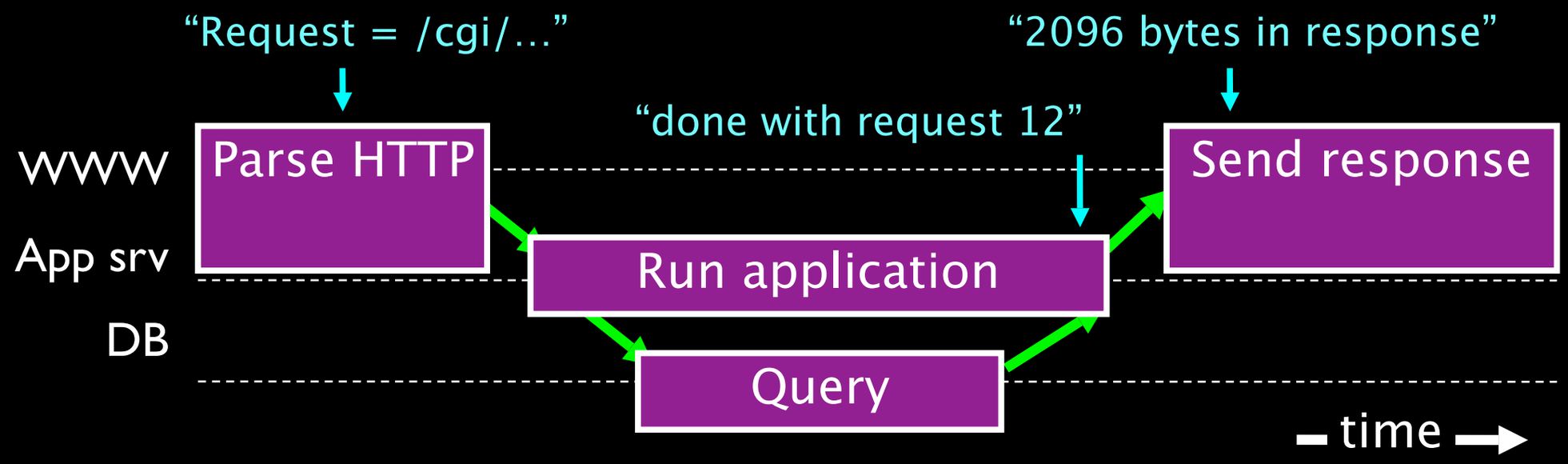
Example:
Web service system



Path Instances

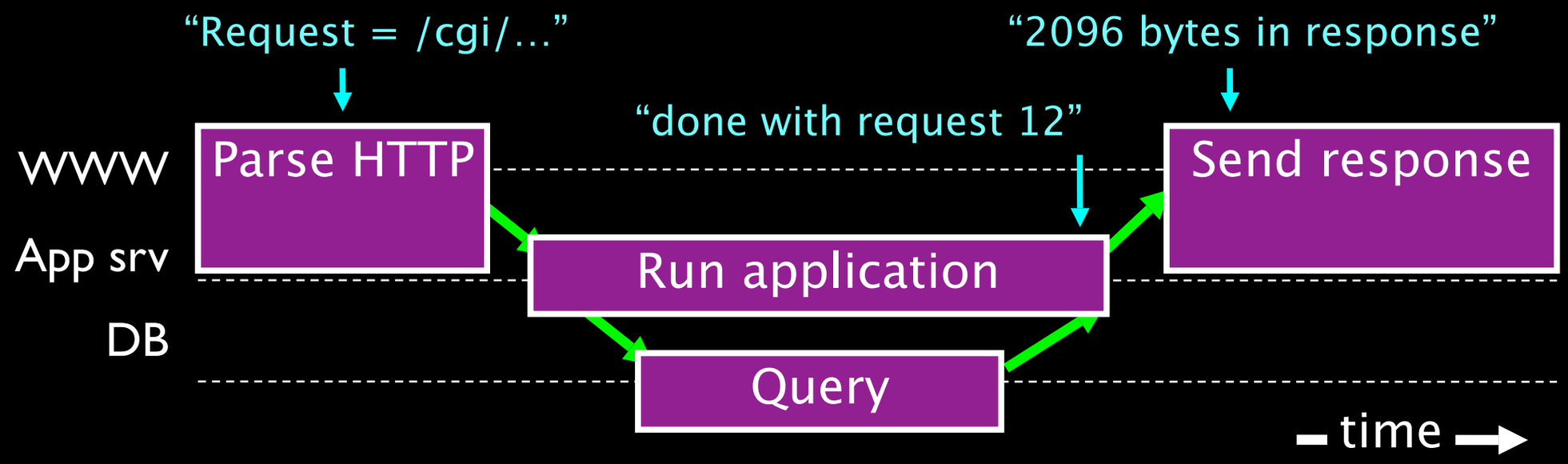


Path Instances



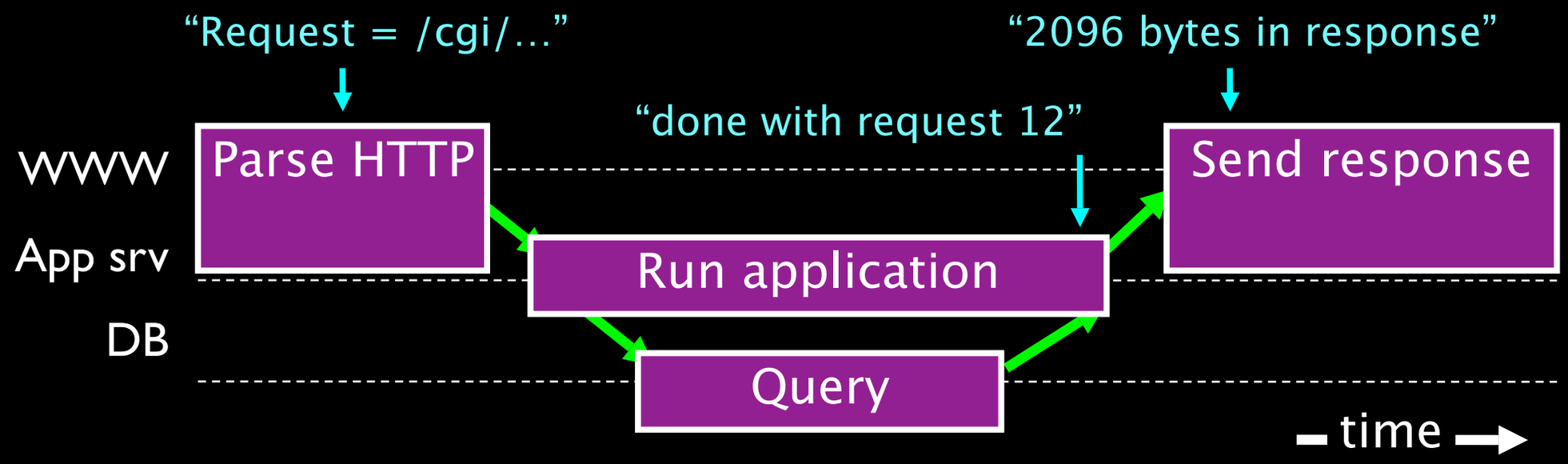
- Within paths are *tasks*, *messages,* and *notices*

Path Instances



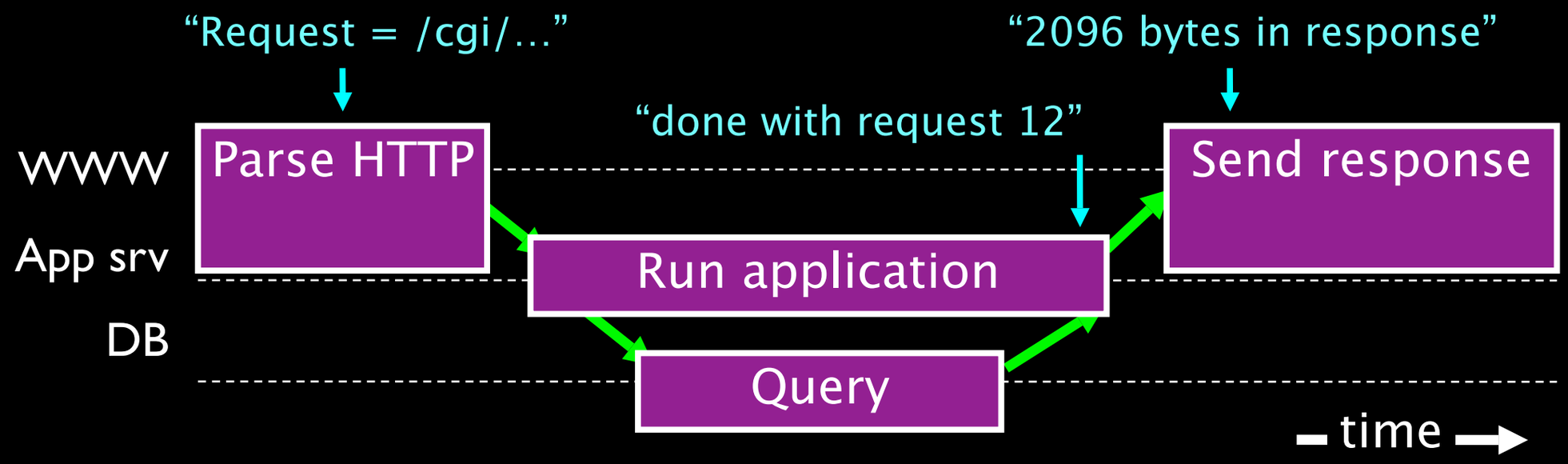
- Within paths are *tasks*, *messages,* and *notices*
  - Tasks: processing with start and end points

Path Instances



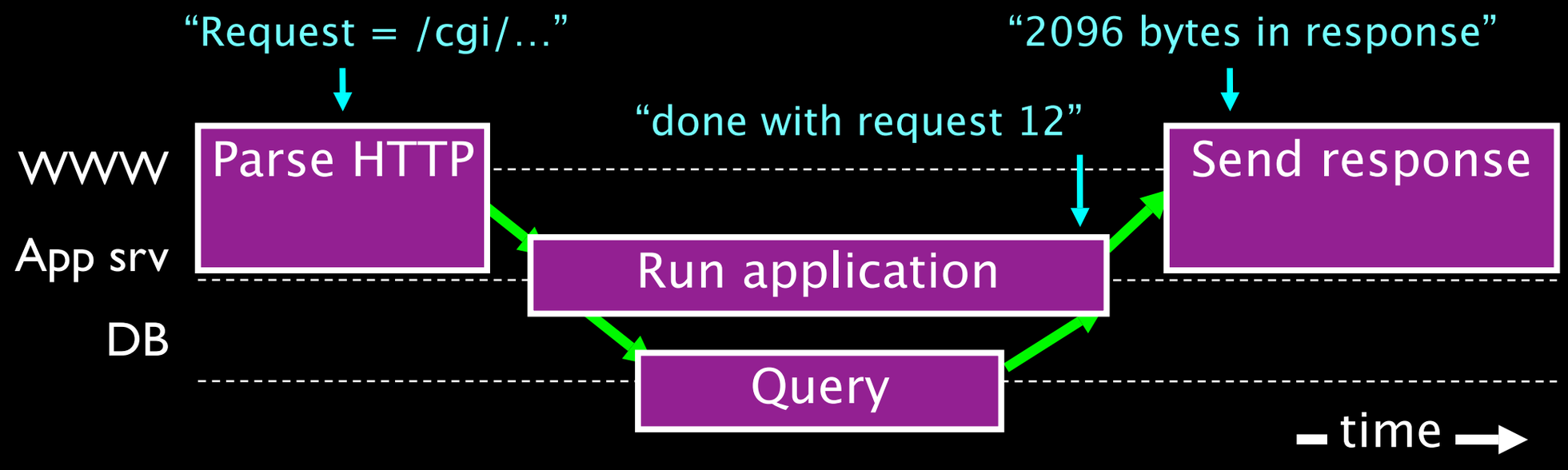
- Within paths are *tasks*, *messages,* and *notices*
  - Tasks: processing with start and end points
  - Messages: send and receive events for any communication

Path Instances



- Within paths are *tasks*, *messages,* and *notices*
  - Tasks: processing with start and end points
  - Messages: send and receive events for any communication
    - Includes network, synchronization (lock/unlock), and timers

Path Instances



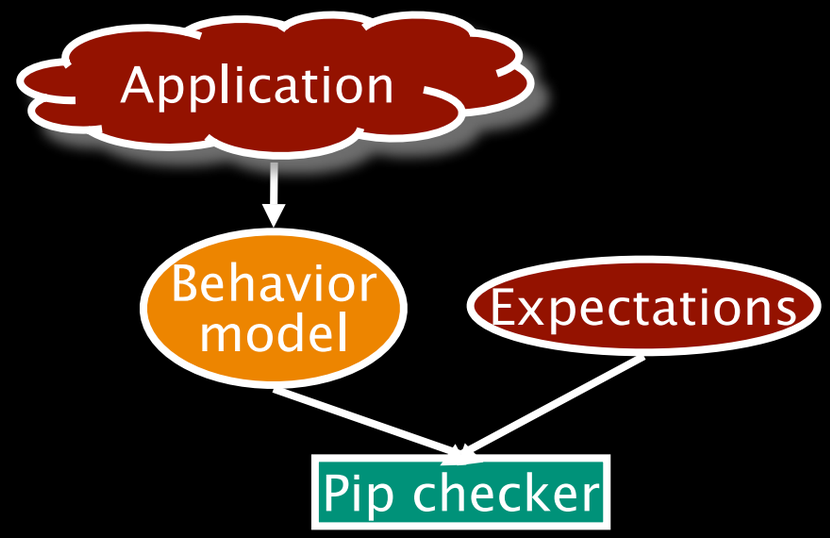
- Within paths are *tasks*, *messages,* and *notices*
  - Tasks: processing with start and end points
  - Messages: send and receive events for any communication
    - Includes network, synchronization (lock/unlock), and timers
  - Notices: time-stamped strings; essentially log entries

Pip workflow

Pip workflow

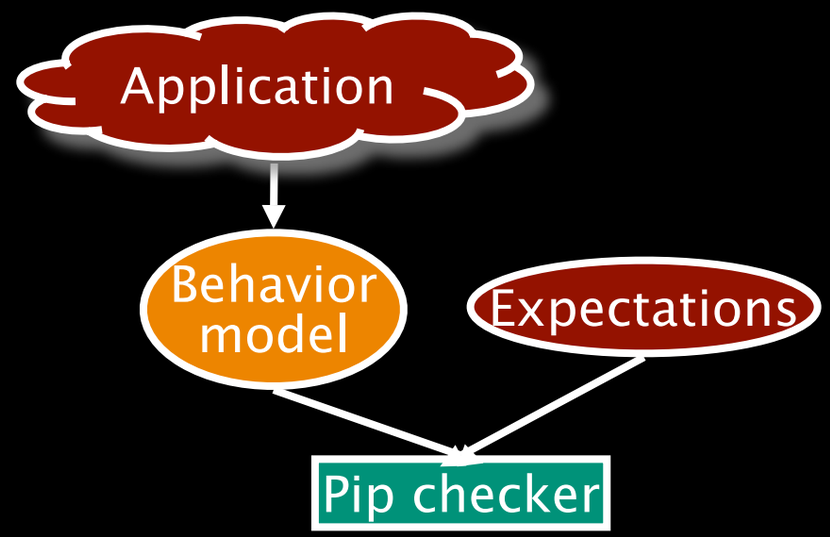
Pip workflow

1. Captures events from an instrumented system



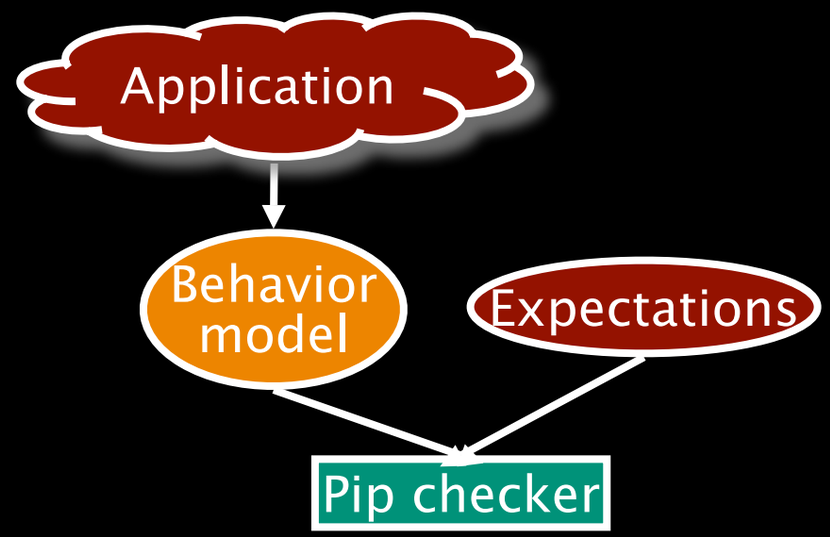
Pip workflow

1. Captures events from an instrumented system
2. Generate paths from events



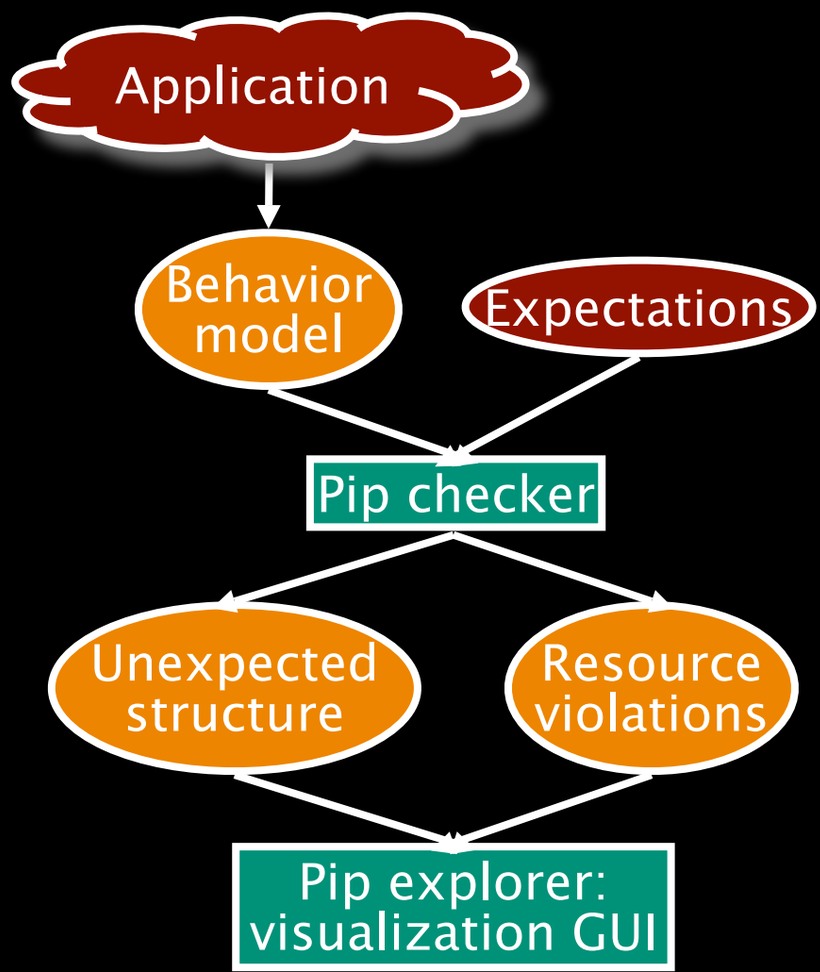
Pip workflow

1. Captures events from an instrumented system
2. Generate paths from events
3. Checks behavior against expectations



Pip workflow

1. Captures events from an instrumented system
2. Generate paths from events
3. Checks behavior against expectations
4. Displays unexpected behavior



Main Contribution: Expectations

Declarative language to describe a DS's expected path.

Describing Expected Behavior

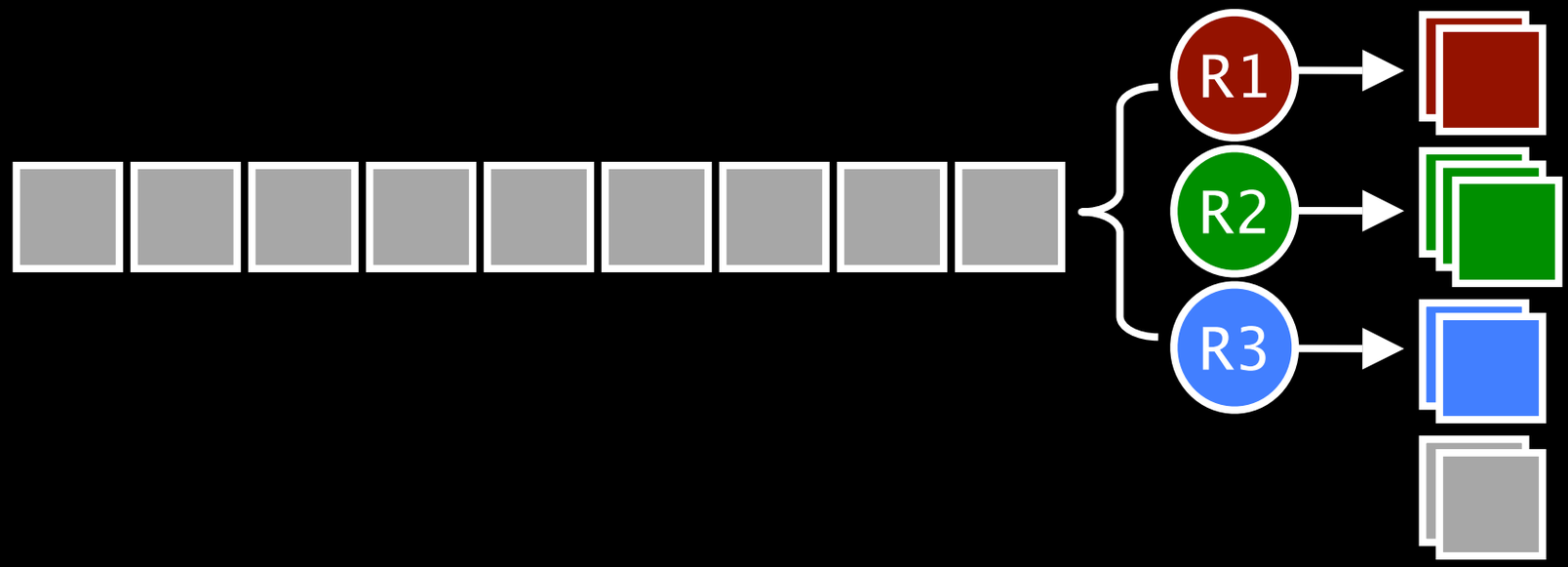
Recognizers
Description of Behavior
Structural or Performance

Aggregates
Assertions about sets of path instances

Expectation: Recognizers

- Validator, invalidator, building block
- Can match a complete path or fragment
  - Invalid paths are often represented as fragments

Expectation: Recognizers

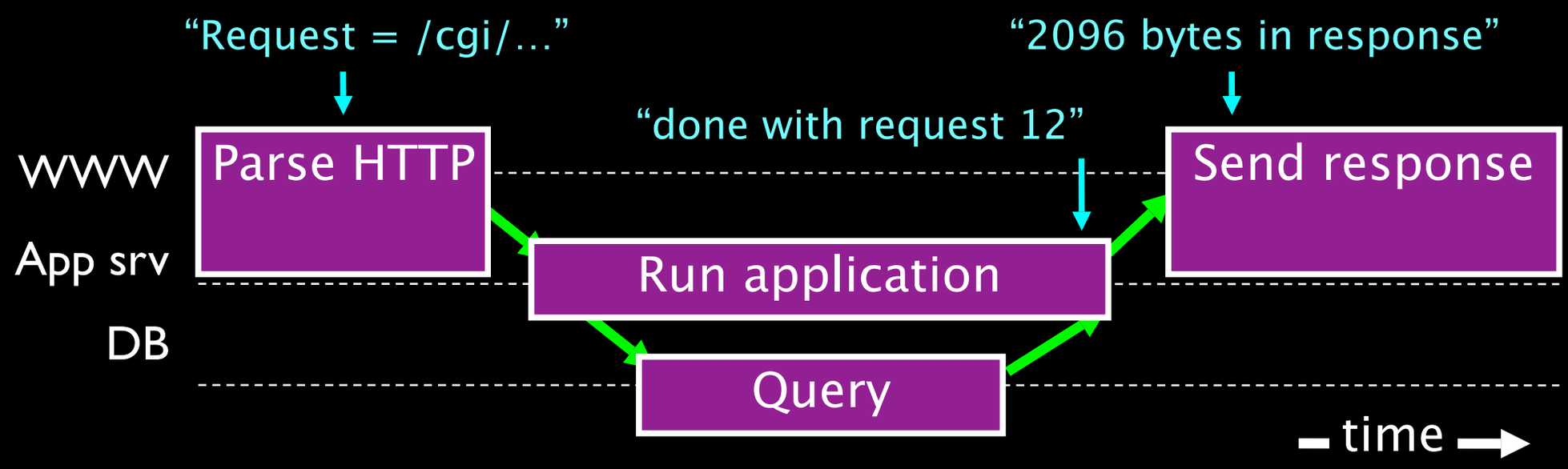


- Validator, invalidator, building block
- Can match a complete path or fragment
  - Invalid paths are often represented as fragments

Recognizer Example

```
validator CGIRequest
   thread WebServer(*, 1)
      task("Parse HTTP") limit(CPU_TIME, 100ms);
      notice(m/Request URL: .*/);
      send(AppServer);
      recv(AppServer);

invalidator DatabaseError
   notice(m/Database error: .*/);
```



Other Statements

- repeat: matches a ≤ n ≤ b copies of a block
- xor: matches any one of several blocks

```
xor {
      branch: …
      branch: …
}
```

- future: lets a block match now or later
  - done: forces the named block to match

Expectation: Aggregate Paths

```
assert(instances(CGIRequest) > 4);
assert(max(CPU_TIME, CGIRequest) < 500ms);
assert(max(REAL_TIME, CGIRequest) <=
    3*avg(REAL_TIME, CGIRequest));
```

- Recognizers categorize paths into sets
- Aggregates make assertions about sets of paths
  - Instances, unique instances, resource constraints
  - Simple math and set operators

Results

- Applied Pip to several distributed systems:
  - FAB: distributed block store
  - SplitStream: DHT-based multicast protocol
  - Others: RanSub, Bullet, SWORD, Oracle of Bacon
- We have found unexpected behavior in each system
- We have fixed bugs in some systems

… and used Pip to verify that the behavior was fixed

Conclusions

- Causal paths are a useful abstraction of distributed system behavior
- Expectations serve as a high-level description
  - Summary of inter-component behavior and timing
  - Regression test for structure and performance
- Finding unexpected behavior can help us find bugs
  - Both structure and performance bugs