Finding Complex Concurrency Bugs in Large Multi-Threaded Applications

Pedro Fonseca, Cheng Li, Rodrigo Rodrigues
MPI-SWS

EuroSys'11

April 22, 2011

Complex bugs

- Semantic bugs:
  - Return wrong results to clients
  - Manifestation is not obvious
  - May have higher impact than crash bugs
- Latent bugs:
  - Silently corrupt state
  - Manifest to users later
  - Require more requests to manifest

Detecting semantic and latent bugs

- Programmer can write a specification
  - Full specification
  - Partial specification (e.g., assertion):
- Hard for programmers

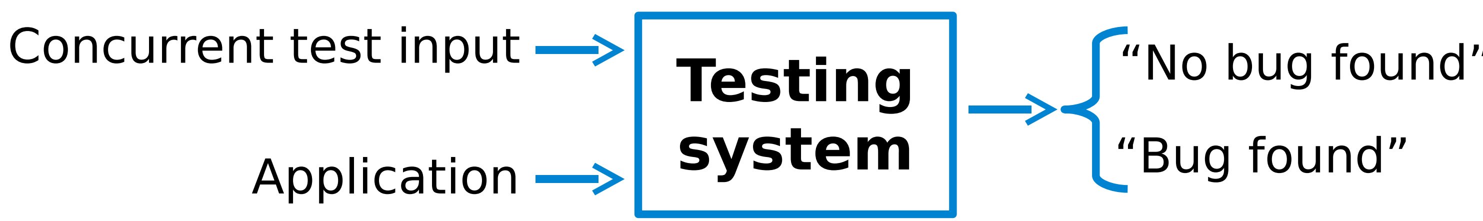
Key observation

- Concurrent applications
  - Important class of software in multi-core era
- Concurrency usually seen as a challenge
- Analyze behavior under different thread interleavings

Take advantage of concurrency to detect concurrency bugs

Goal

- Test concurrent applications
  - Find semantic bugs and latent bugs
- Bugs might not be caused by data races



Outline

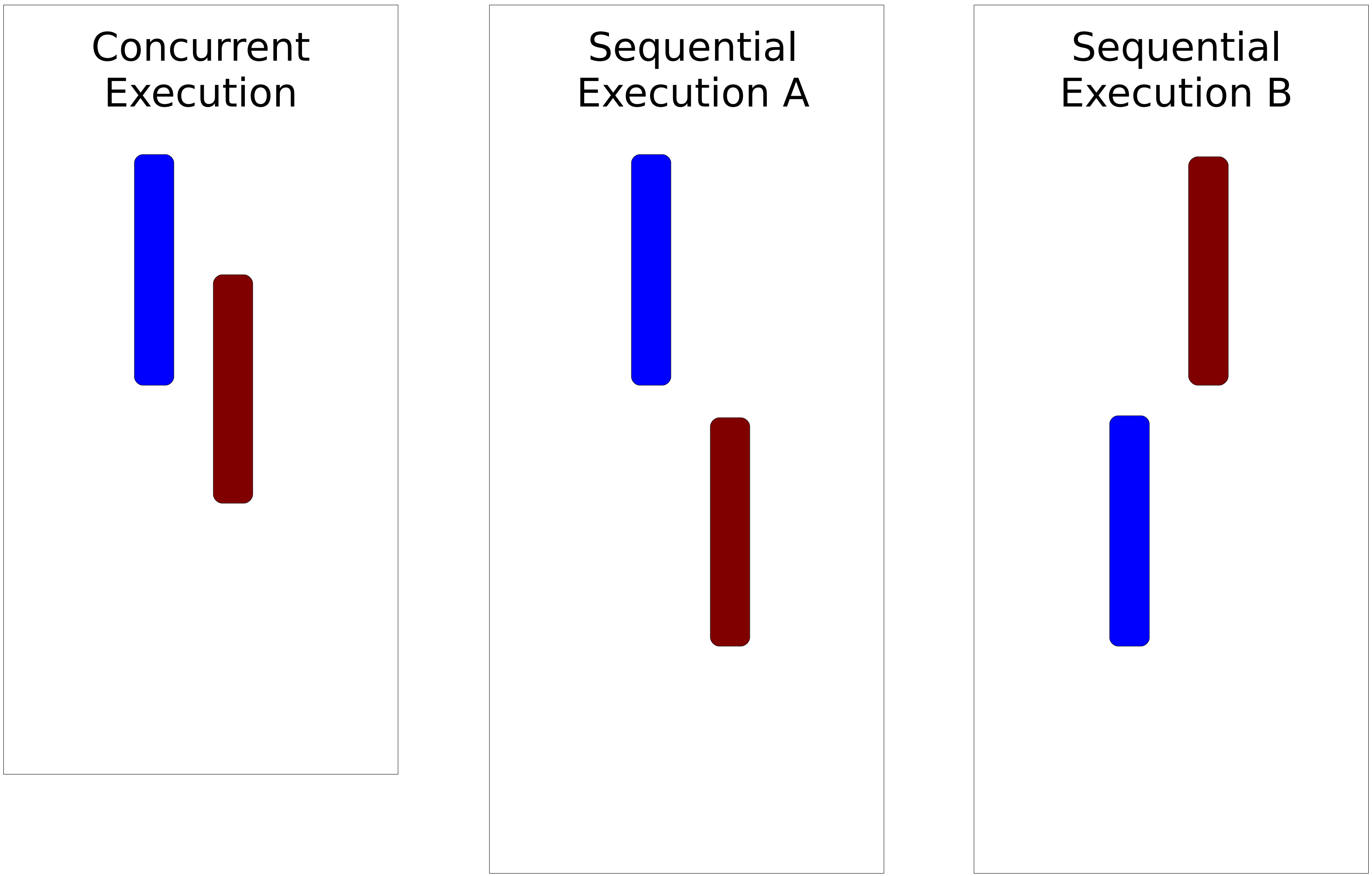
- Idea
- Pike: A tool to detect concurrency bugs
- Experience with Pike

Goal: Detecting concurrency bugs

- Hypothesis: A correct execution behaves in the same way as one of the sequential executions
- Might hold even for large and complex apps

Find concurrency bugs by checking for linearizability

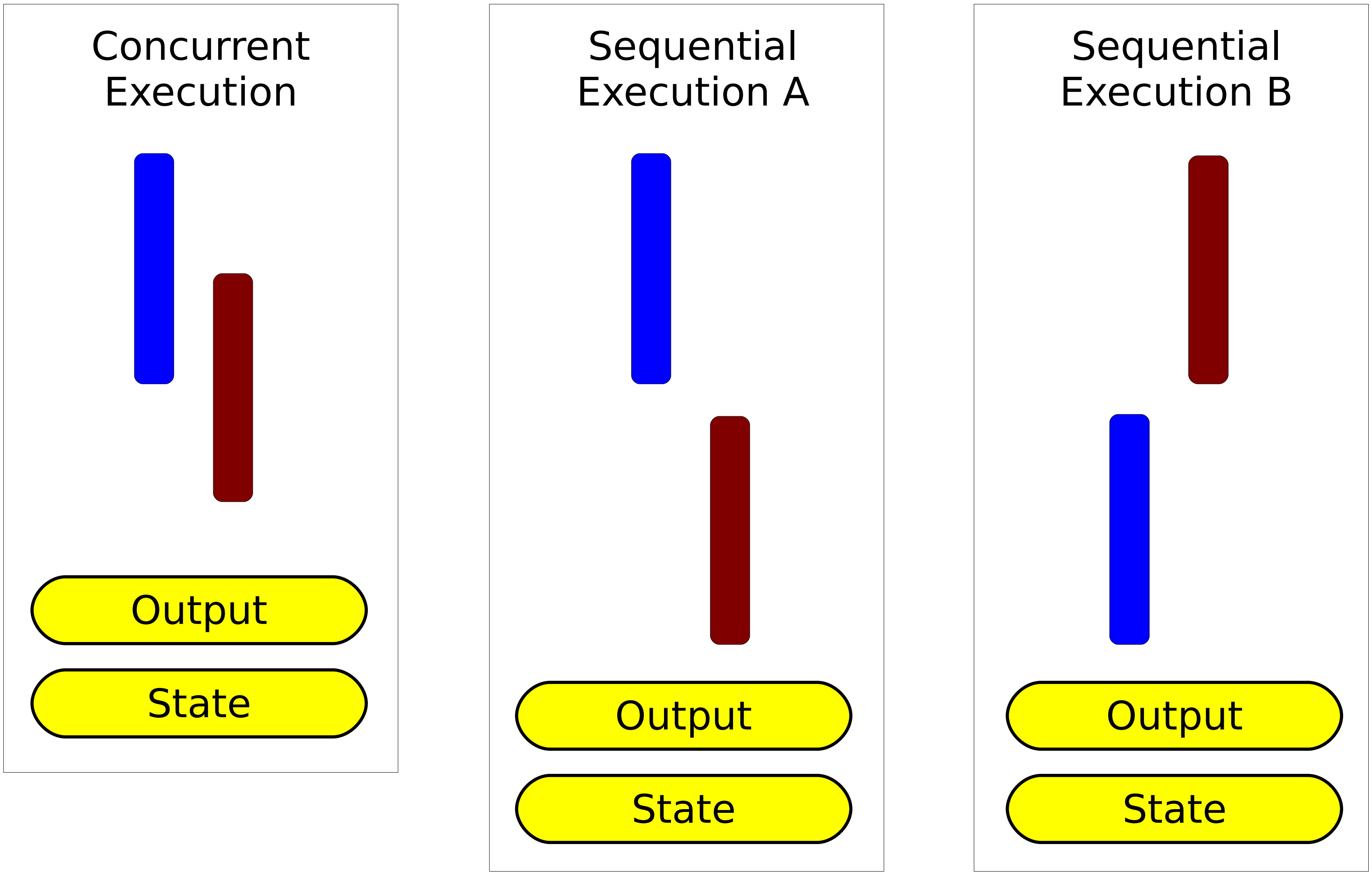
Checking for linearizability



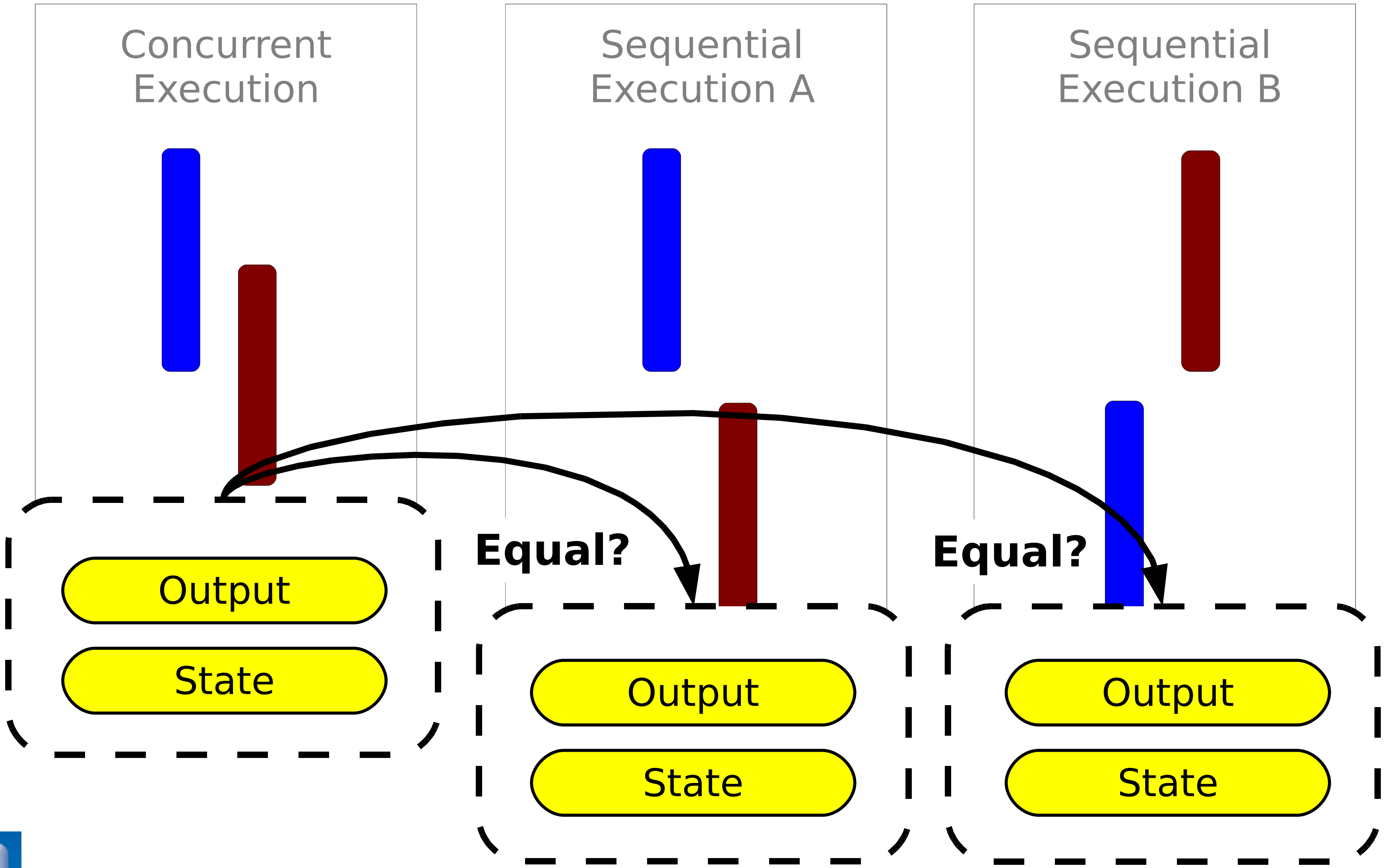
Which behavior to analyze?

- External behavior: application output
  - Check visible behavior
  - Detects semantic bugs
- Internal behavior: application state
  - Check for state corruption
  - Detects early on latent bugs

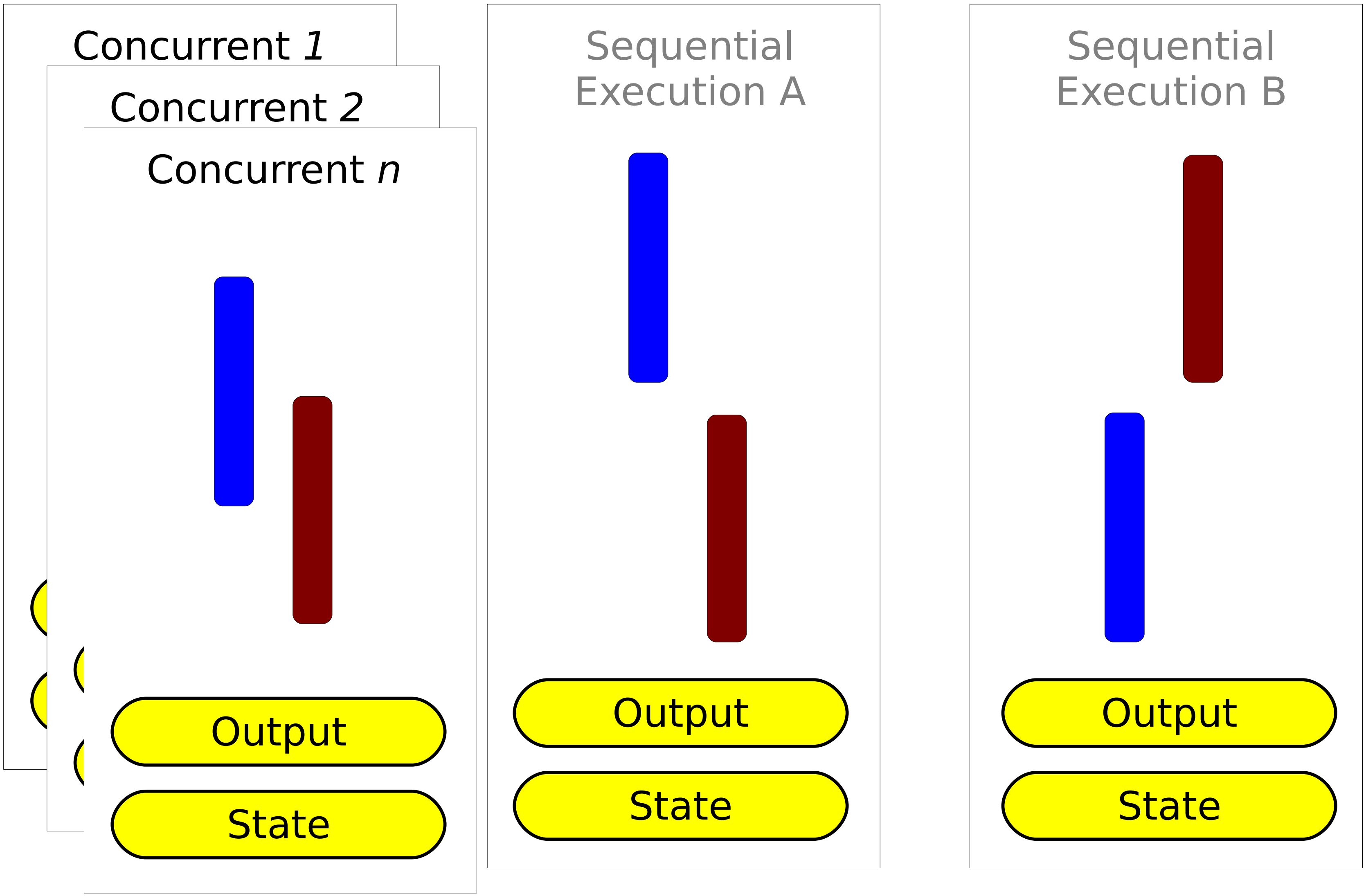
Checking for linearizability



Checking for linearizability



Checking for linearizability



Outline

- Idea: Assume linearizability to detect concurrency bugs
- Pike: A tool to detect concurrency bugs
- Experience with Pike

Pike

- We built Pike to find concurrency bugs
  - Pike runs for each test:
    - The sequential executions
    - Various concurrent executions (PCT algorithm)
  - State and output comparison
- Challenges:
  - Analyze the application state
  - Handle false positives

Analyzing the state

- Simple bitwise comparison does not work
  - E.g., pointers would cause false positives
- Need an abstraction of the application state
  - E.g., capture set

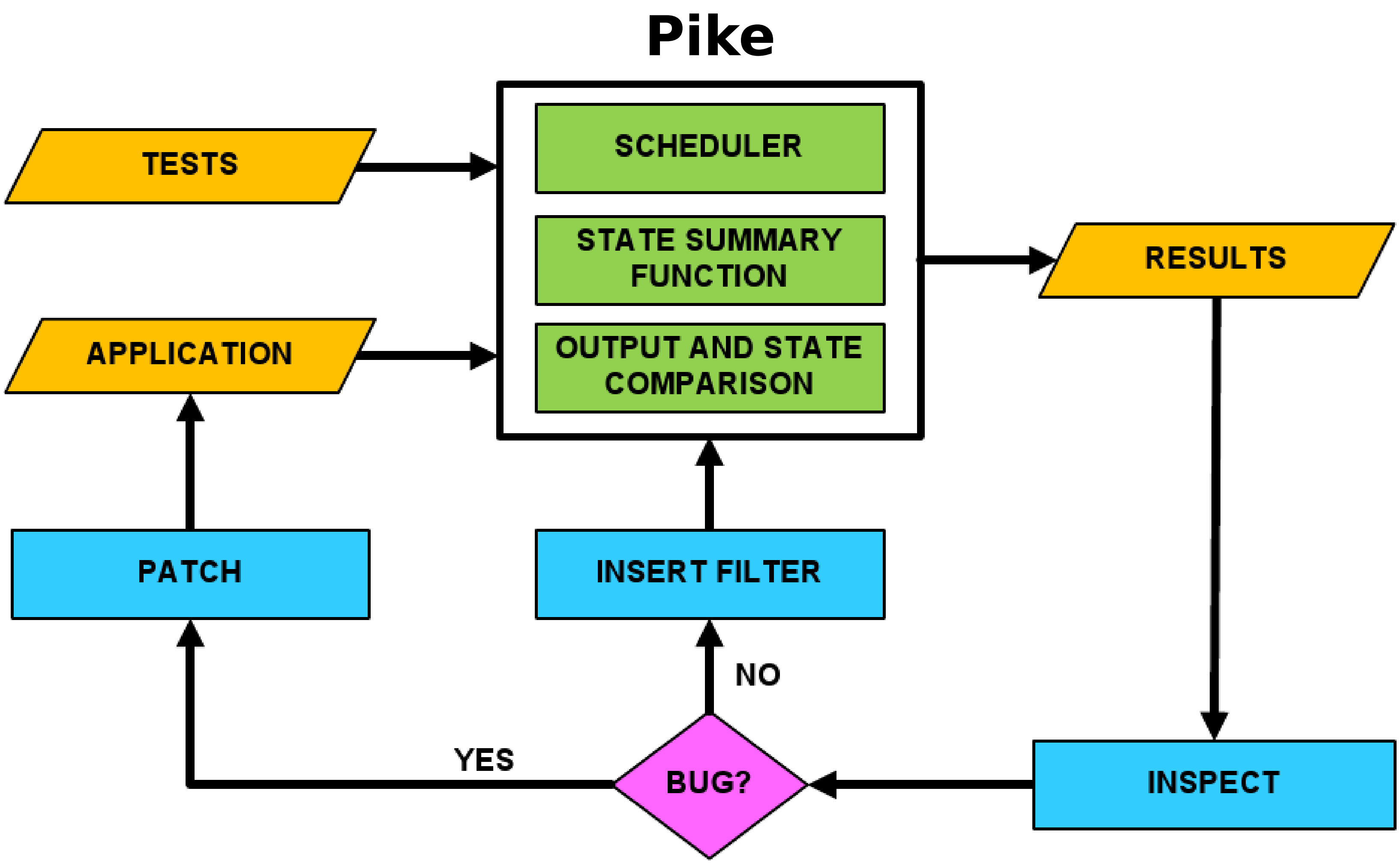
	0x00:	a		0x00:	**b**		0x00:	a
	0x01:	b		0x01:	**a**		0x01:	b
memory:	0x02:	c	≠	0x02:	c	≠	0x02:	
	0x03:			0x03:			0x03:	**c**
state summary:		{a,b,c}	=		{a,b,c}	=		{a,b,c}

  - Programmer writes simple state summary functions

False positives

- Deliberate violations of linearizability
  - Hypothesis does not hold
- Solution: developer introduces filters
  - Change comparison function
    - E.g., check for containment of sets instead of equality

Overview


Pike
TESTS
APPLICATION
SCHEDULER
STATE SUMMARY FUNCTION
OUTPUT AND STATE COMPARISON
RESULTS
PATCH
INSERT FILTER
NO
YES
BUG?
INSPECT

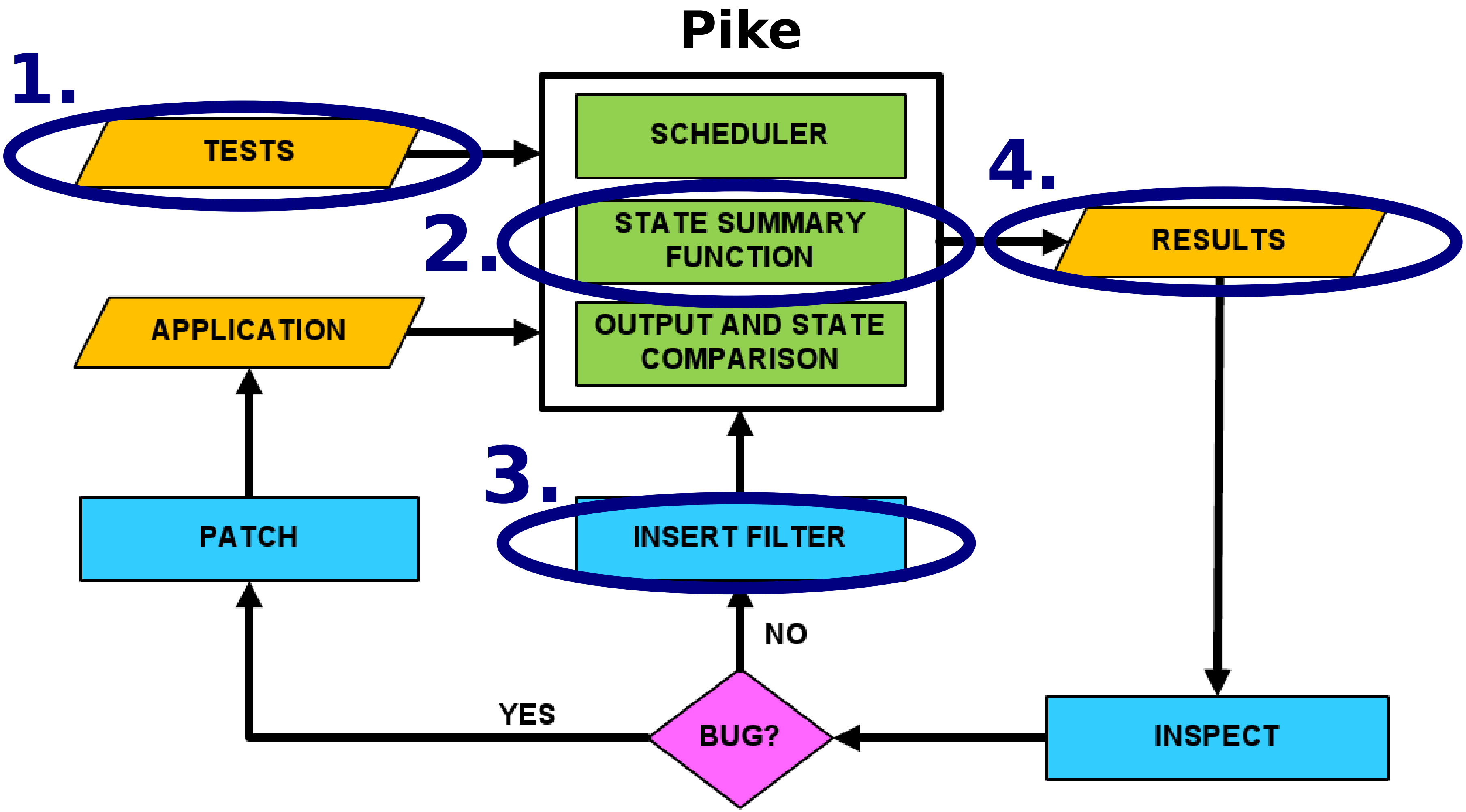
Outline

- Idea: Assume linearizability to detect concurrency bugs
- Pike: A tool to detect concurrency bugs
- Experience with Pike

Experience: Testing MySQL

- We applied Pike to a stable version of MySQL
- A large and complex multi-threaded application
  - 360,000 lines of code

Applying Pike to MySQL



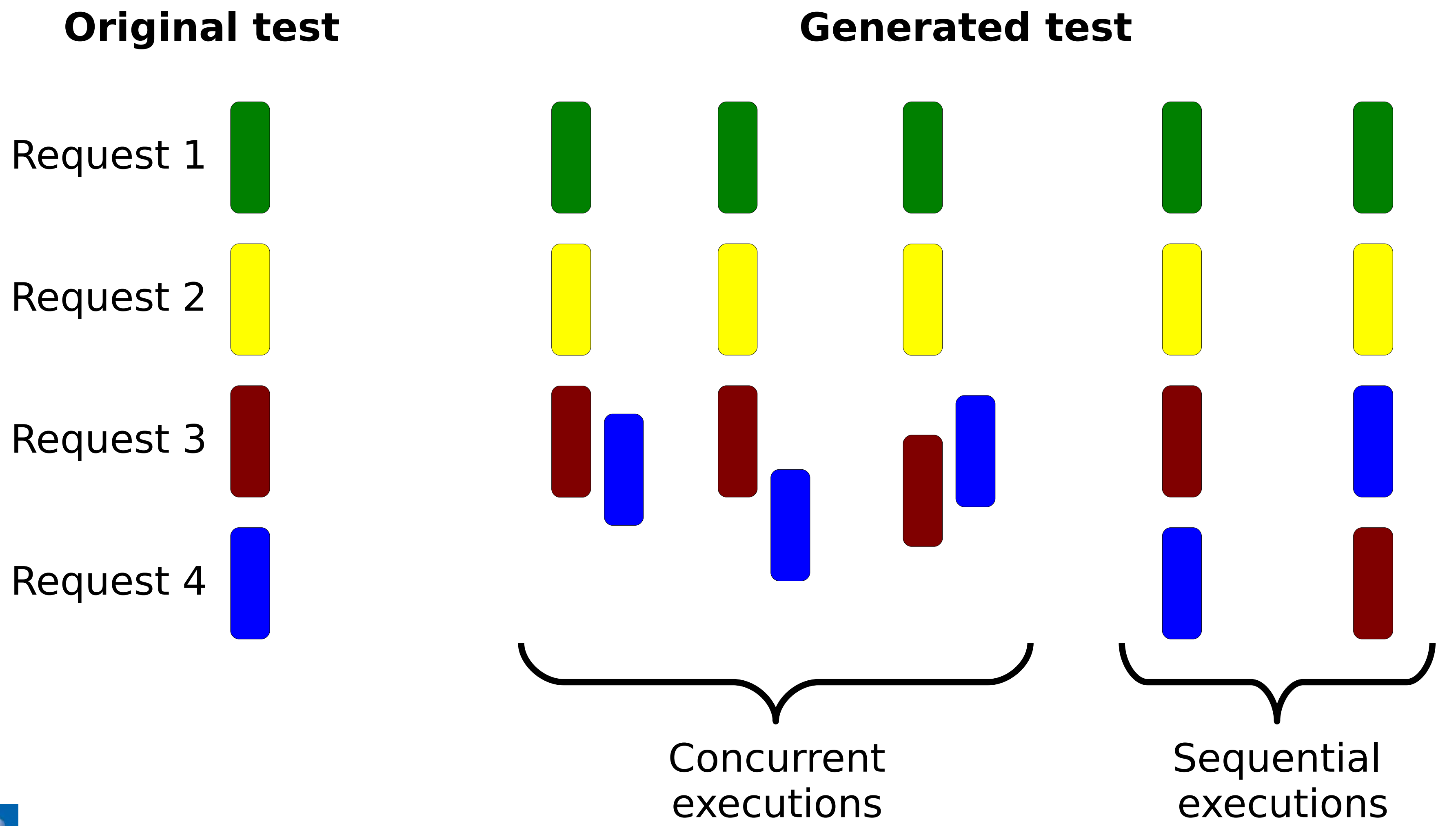
1. Test generation (1/3)

- Initial possibilities:
  - Manual test generation
  - Random grammar-assisted test generation
  - Automatic test generation (e.g., KLEE, DART)
- We plan to explore these possibilities further

1. Test generation (2/3)

- MySQL includes sequential tests
- We made MySQL's own test suite concurrent
  - We generated 1550 concurrent tests

1. Test generation (3/3)



2. Capturing MySQL state

- We created state summary functions for six data structures
  - E.g., caches and indexes
  - Represented sets or sequences
  - Around 600 lines of code
  - Around two man-months to understand and annotate MySQL

3. Dealing with false positives

- Initially 1/3 of the tests led to false positives
  - Caused by application caches
- Inserted two filters
  - Check for containment instead of equality
  - Significantly reduced false positives
- Only 27 false positives remained
  - Most of them were easy to rule out

4. Results

- We ran experiments on a cluster
  - Run 400 interleavings for each of the 1550 tests
- Found 12 tests that triggered concurrency bugs
  - 8 instances of memory corruption
  - 10 instances of wrong results

4. Examples of bugs found

- Inconsistent results
  - Requests: *DROP* and *SHOW TABLE STATUS*
  - *SHOW TABLE STATUS* returns invalid fields
- Stale results (latent)
  - Requests: *SELECT* and *INSERT*
  - Subsequent *SELECT*s return old contents

Conclusion

- Pike tests for semantic and latent bugs
  - Infers specification assuming linearizability
- Experience with MySQL
  - Modest effort to analyze state
  - Relatively close to linearizable semantics