Learning-Based Controlled Concurrency Testing

Suvam Mukherjee, *Microsoft*

Pantazis Deligiannis, *Microsoft Research*

Arpita Biswas, *Harvard*

Akash Lal, *Microsoft Research*

Concurrent Programs

…are mainstream

…are extremely hard to get right

- uncontrolled non-determinism

…"Heisenbugs" (hard to detect, hard to replay)

…are difficult to test using traditional techniques

- exponentially large space of possible behaviors
- stress-tests are ineffective
- inability to deterministically replay bugs

Controlled Concurrency Testing

Systematically explore space of program behaviors

…by *serializing* concurrent program executions

…using a *scheduler* which resolves control non-determinism

Example

```
public class TestThreads {

    public static volatile int Value;

    public static void T1Proc () {
      Thread.Sleep(100);
      Value = 3;
    }

    public static void T2Proc () {
      Thread.Sleep(100);
      Value = 5;
    }
```

```
    public static void main(String[] args) {

        Thread t1 = new Thread(new ThreadStart(T1Proc));
        Thread t2 = new Thread(new ThreadStart(T2Proc));

        t1.Start();
        t2.Start();
        t1.Join();
        t2.Join();

        Assert(Value == 5);
    }
}
```

Controlled Concurrency Testing

Stateful

- Zing [*Andrews et al, 2004*]
- SPIN [*Holzmann, 1997*]
- DFS
- BFS

↓ Requires full-program state

Stateless

- Random
- PCT [*Burckhardt et al, 2010*]
- Delay Bounding [*Emmi et al, 2011]*
- Preemption bounding [*Musuvathi et al, 2007*]

↑ PCT and Random known to be effective on real-world programs

↓ Search heuristics based on *empirical observations* of bug patterns

…few context switches

…few ordering constraints

…few deviations from a deterministic scheduler

↓ What about unknown patterns?

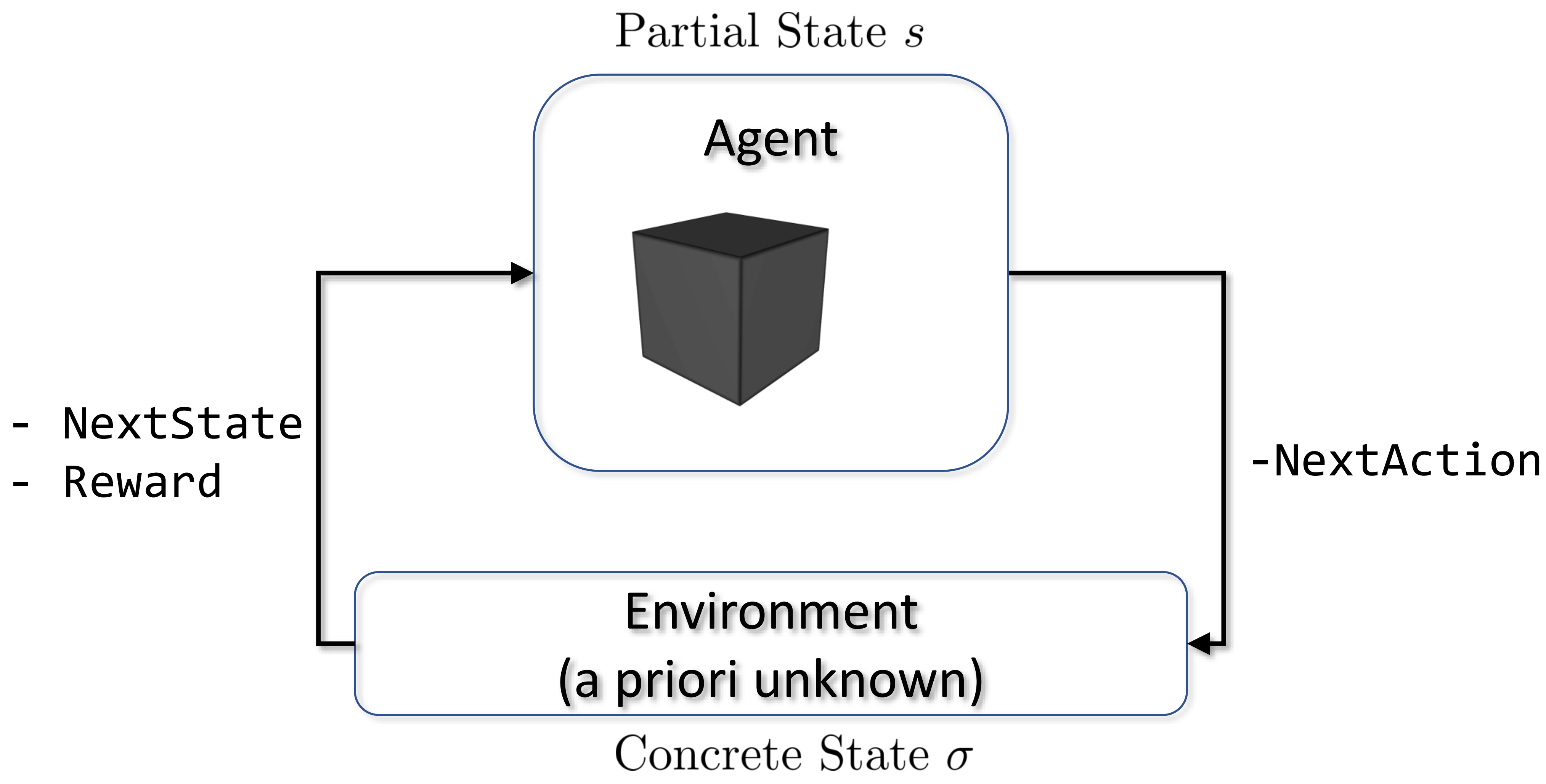
Our Contributions

- Systematic exploration strategy based on *Q-Learning*
  - …focus on *coverage*
  - …strike a balance between *exploration* (randomly choose the next action)
  - …and *exploitation* (learn from previously taken decisions)

- Highly customizable search strategy
  - …that adapts to the program under test!

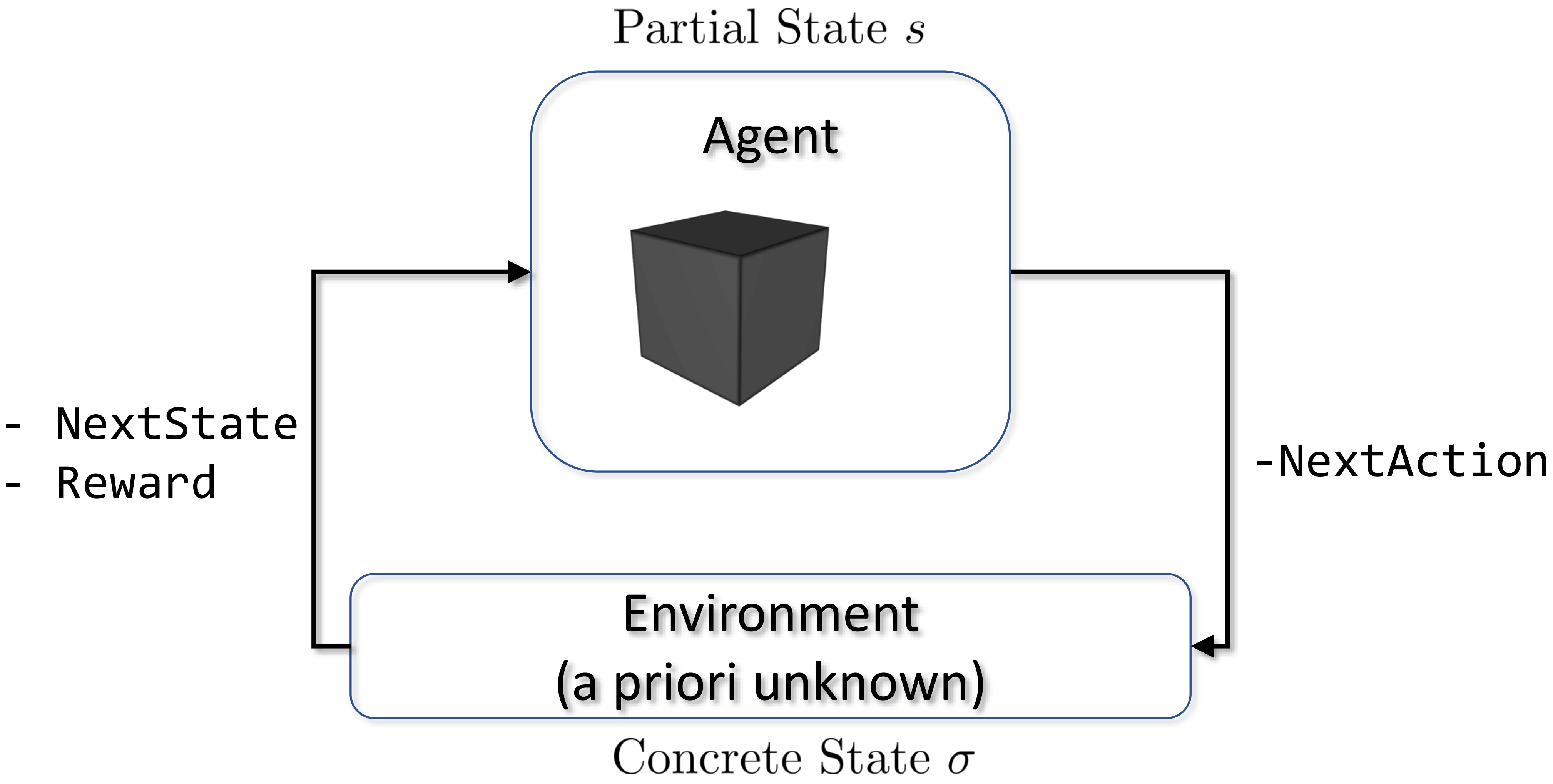
- Implemented in Coyote
  - Evaluated on micro-benchmarks and *production services* from Azure

Reinforcement Learning

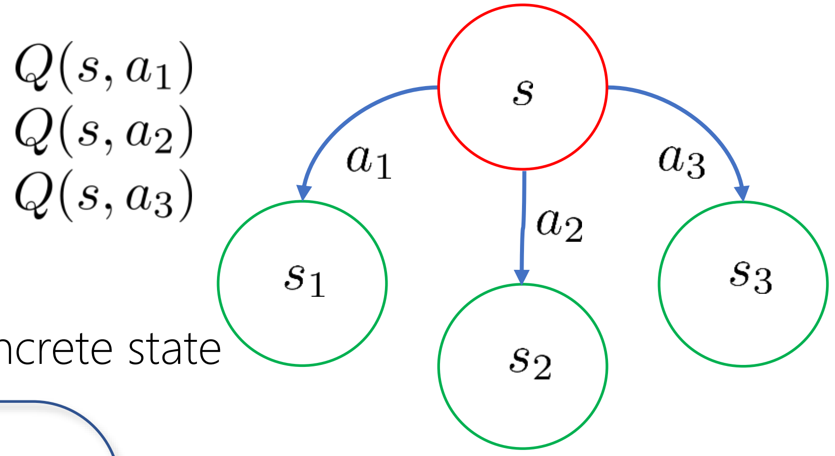


Reinforcement Learning

Goal of Agent: Learn an *optimal* policy, which *maximizes* expected reward

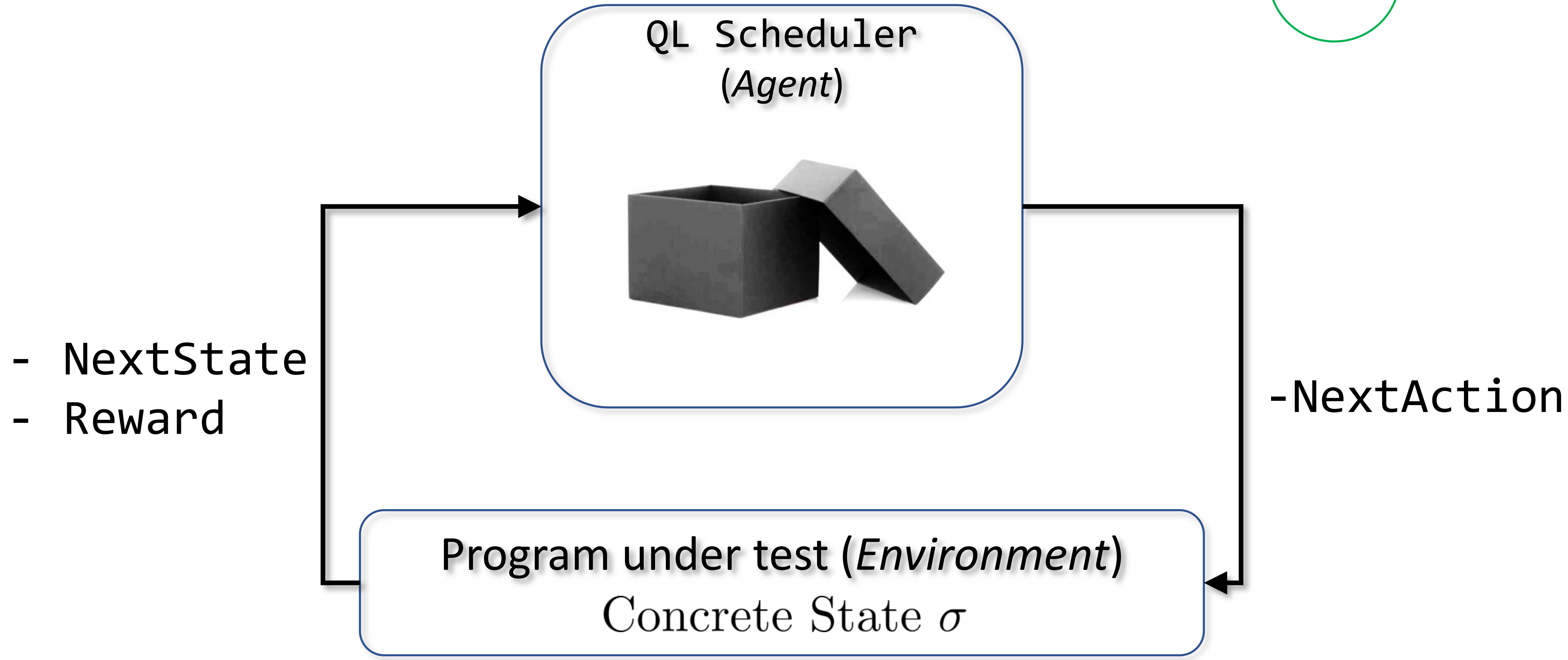


Learning-based CCT



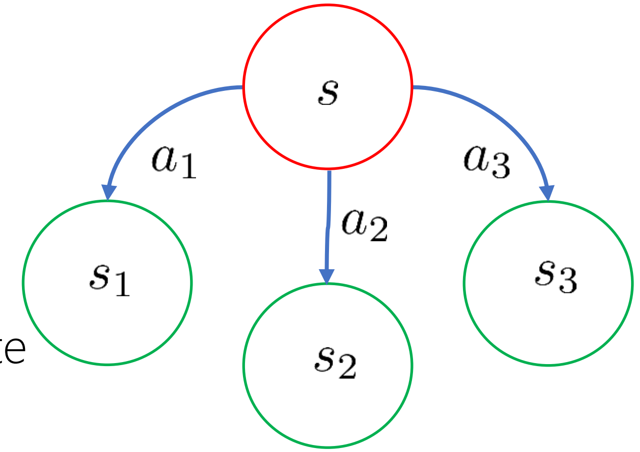
$$s = \mathcal{H}(\sigma)$$

User-defined **abstraction** of concrete state



Learning-based CCT

$Q(s, a_1)$
$Q(s, a_2)$
$Q(s, a_3)$



$$s = \mathcal{H}(\sigma)$$

User-defined **abstraction** of concrete state

`QL Scheduler`
(*Agent*)

- `NextState`
- `Reward`

`-NextAction`

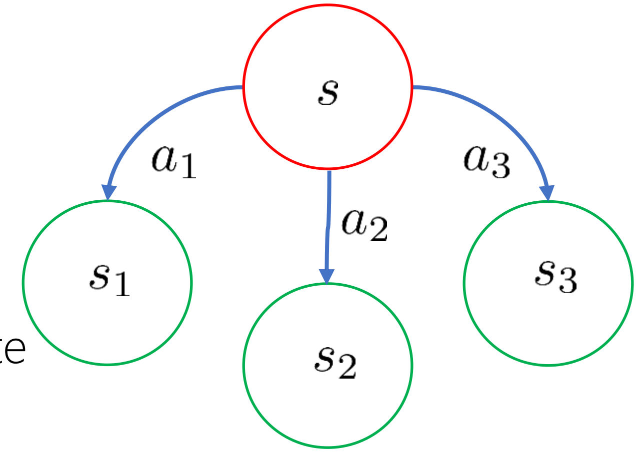
$$P_s(a_i) \leftarrow \frac{e^{Q(s,a_i)}}{\sum_{j=1}^{3} e^{Q(s,a_j)}}$$

Softmax

Program under test (*Environment*)
Concrete State σ

Learning-based CCT

$Q(s, a_1)$
$Q(s, a_2)$
$Q(s, a_3)$

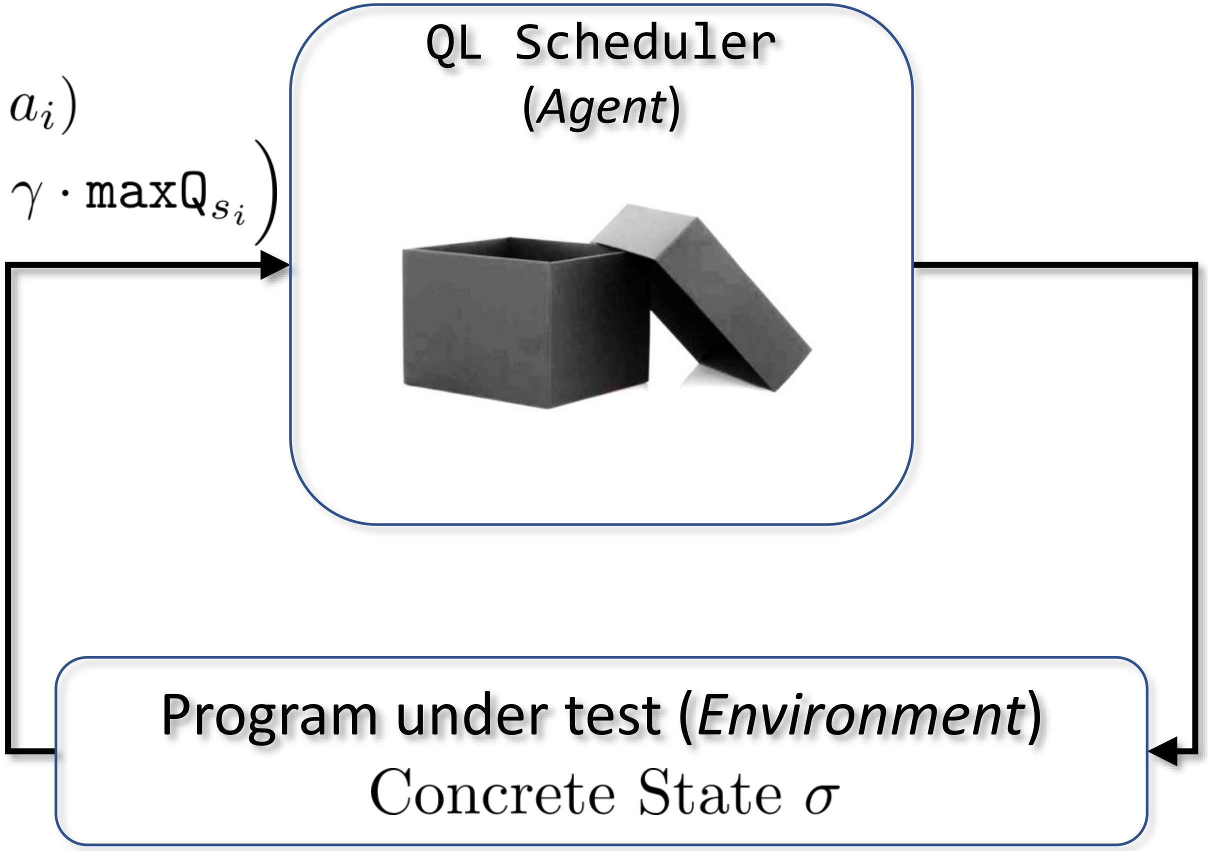


$$s = \mathcal{H}(\sigma)$$

User-defined **abstraction** of concrete state

Value Update

$$Q(s, a_i) \leftarrow (1 - \alpha) \cdot Q(s, a_i) + \alpha \cdot \left(R(s, a_i) + \gamma \cdot \mathtt{maxQ}_{s_i}\right)$$



- NextState
- Reward
  (Penalty)

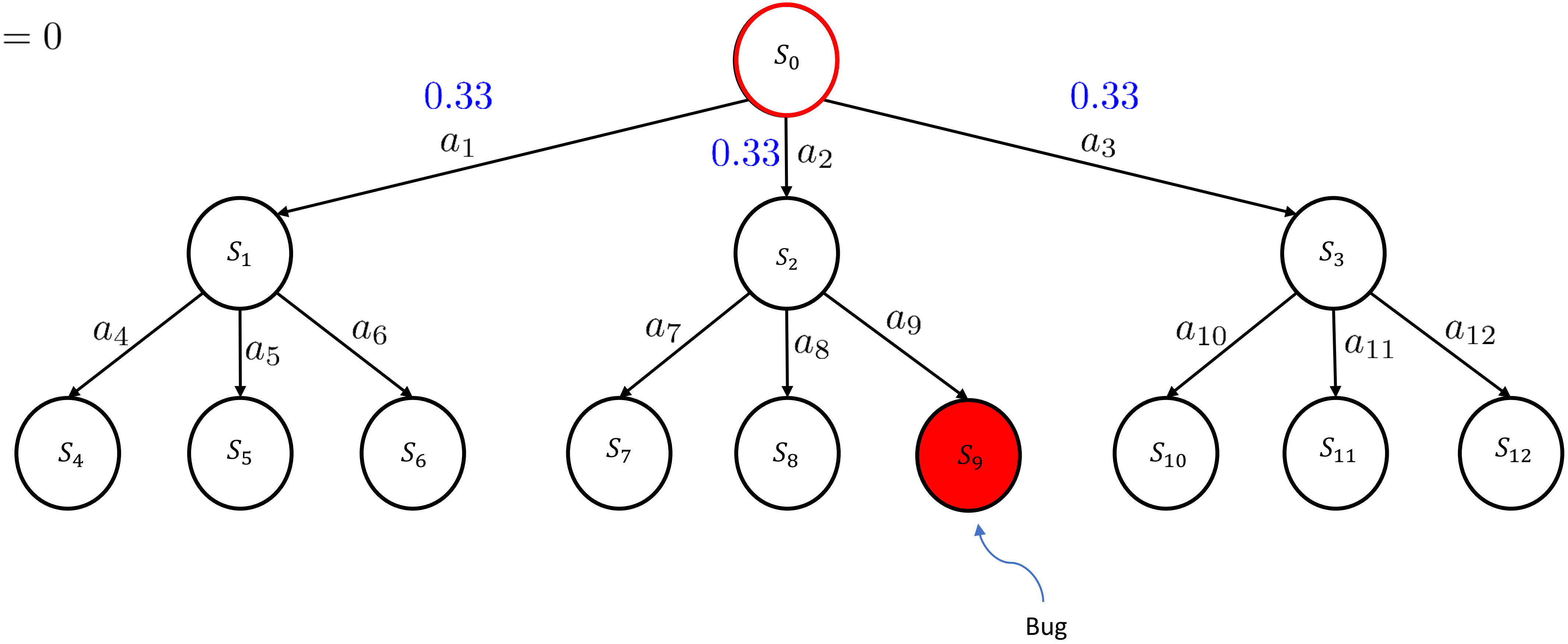
-NextAction

$$P_s(a_i) \leftarrow \frac{e^{Q(s,a_i)}}{\sum_{j=1}^{3} e^{Q(s,a_j)}}$$

Softmax

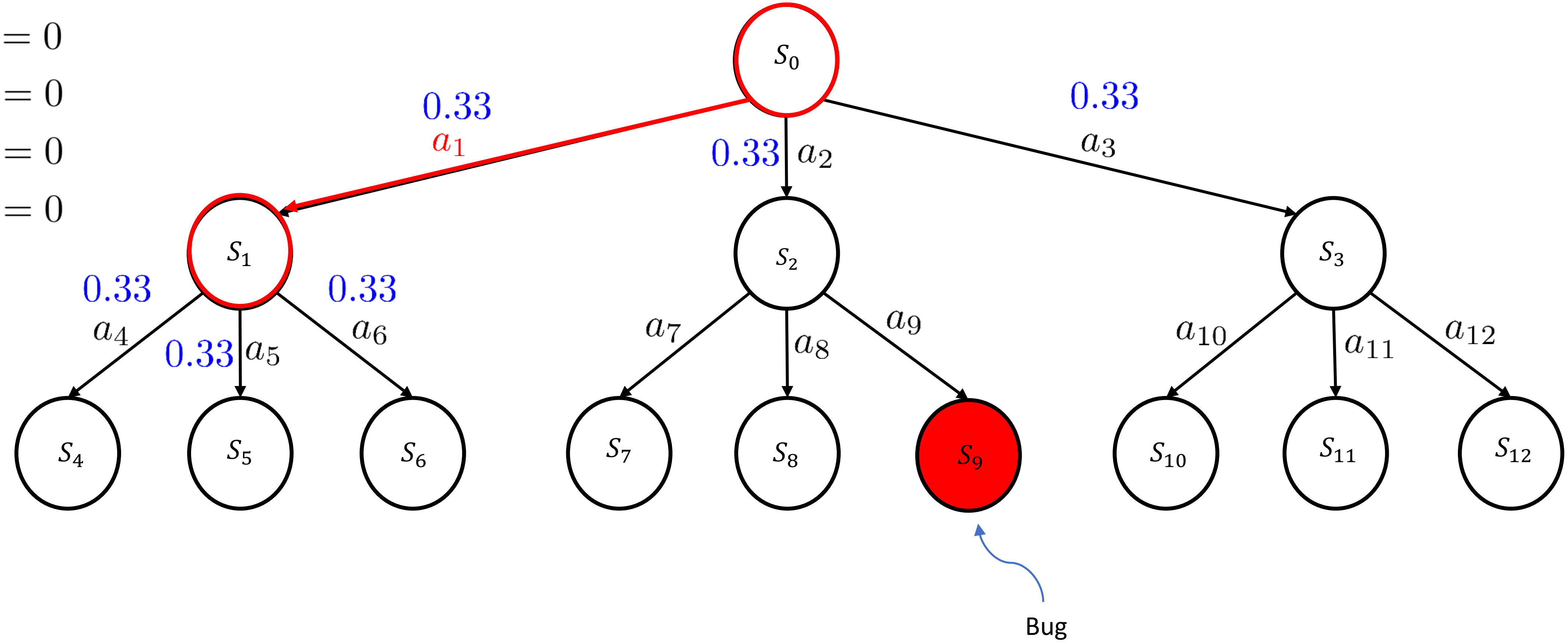
Controlled Concurrency Testing

$Q(s_0, a_1) = 0$
$Q(s_0, a_2) = 0$
$Q(s_0, a_3) = 0$



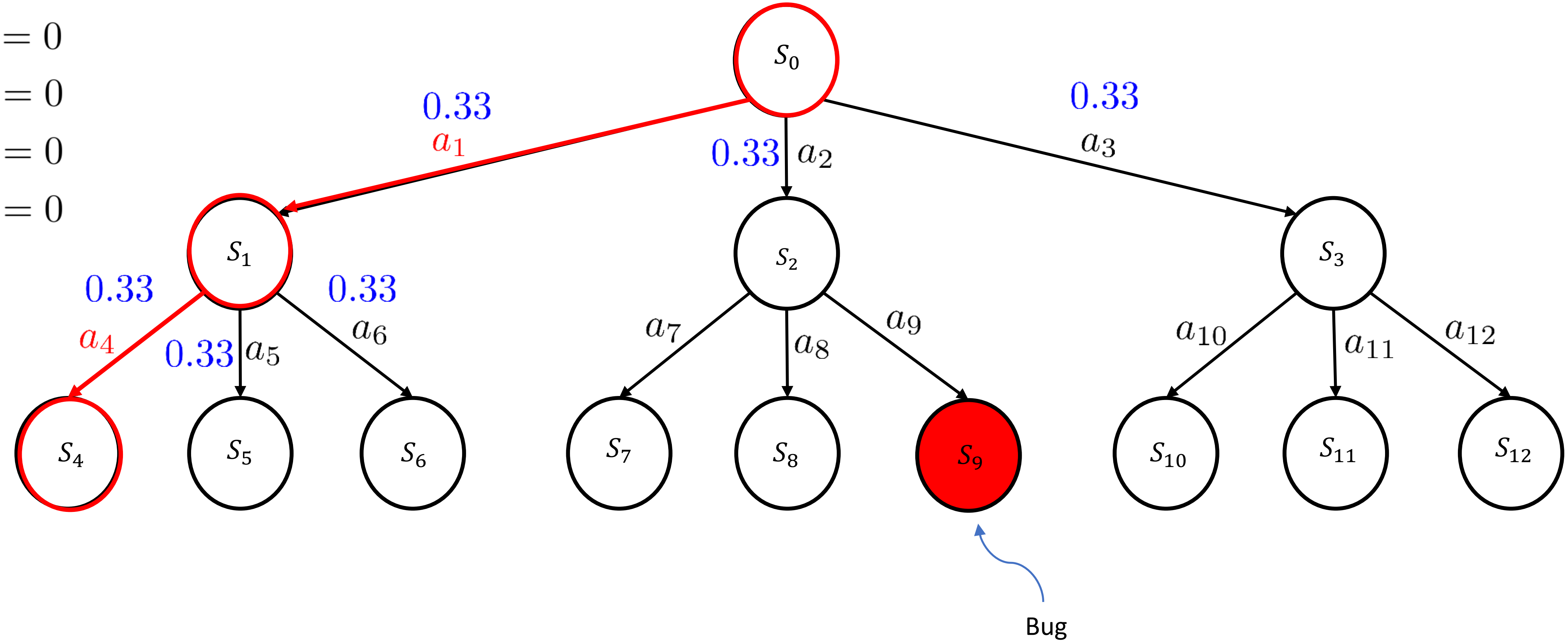
Controlled Concurrency Testing

$Q(s_0, a_1) = 0$
$Q(s_0, a_2) = 0$
$Q(s_0, a_3) = 0$
$Q(s_1, a_4) = 0$
$Q(s_1, a_5) = 0$
$Q(s_1, a_6) = 0$



Controlled Concurrency Testing

$Q(s_0, a_1) = 0$
$Q(s_0, a_2) = 0$
$Q(s_0, a_3) = 0$
$Q(s_1, a_4) = 0$
$Q(s_1, a_5) = 0$
$Q(s_1, a_6) = 0$



Controlled Concurrency Testing

$Q(s_0, a_1) = -0.3$

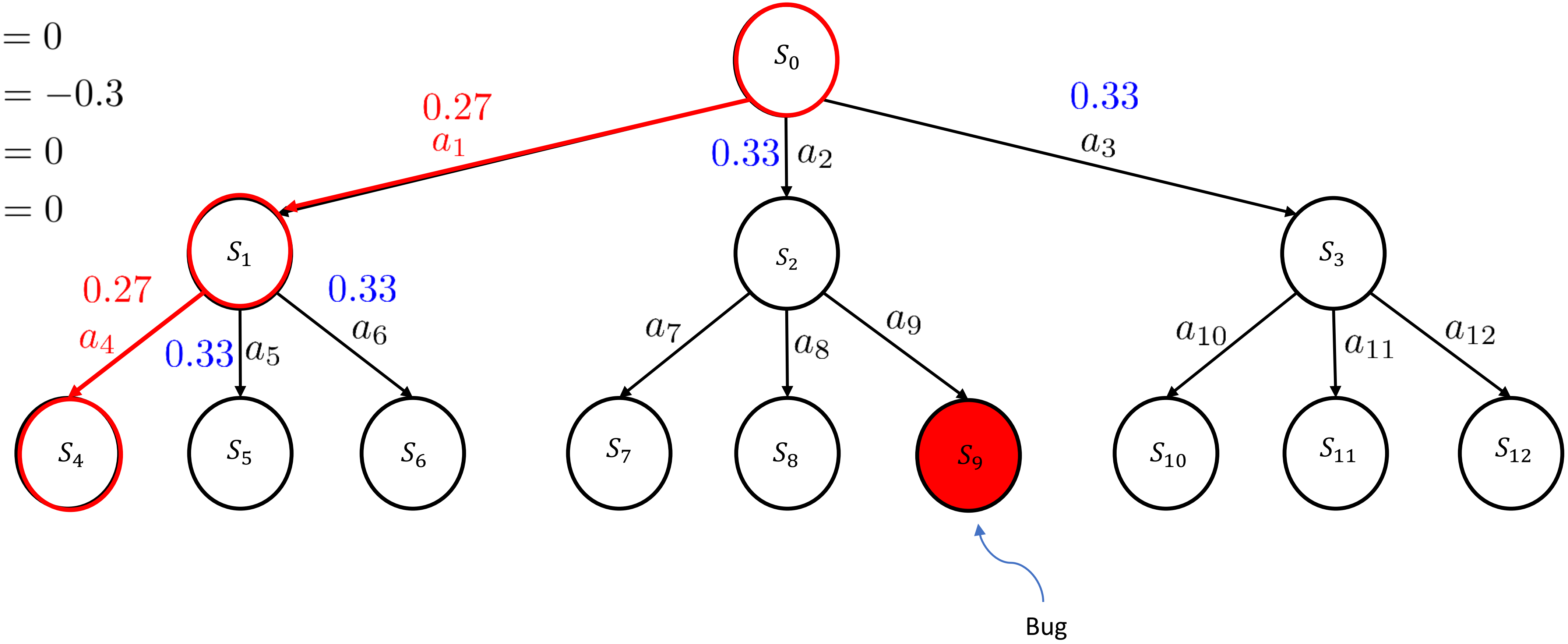
$Q(s_0, a_2) = 0$

$Q(s_0, a_3) = 0$

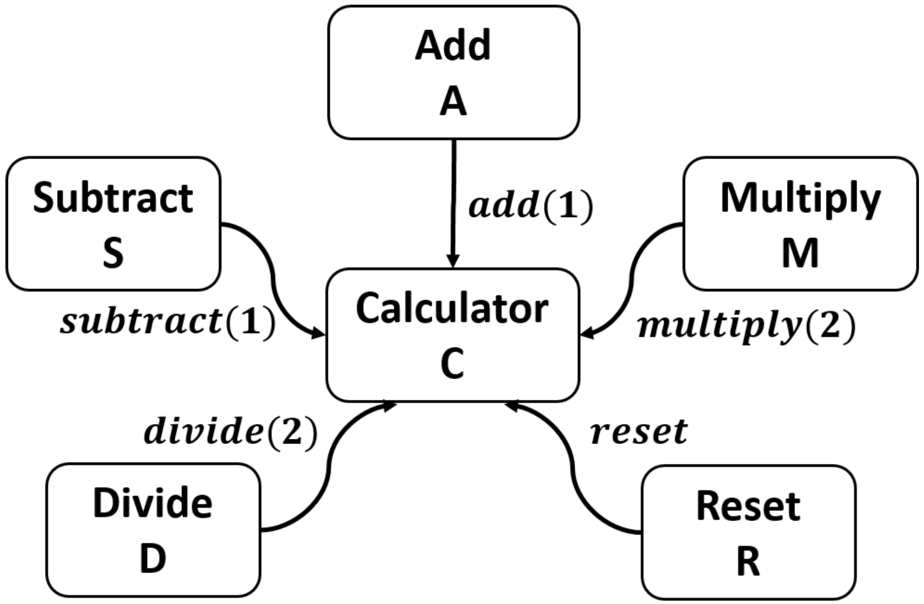
$Q(s_1, a_4) = -0.3$

$Q(s_1, a_5) = 0$

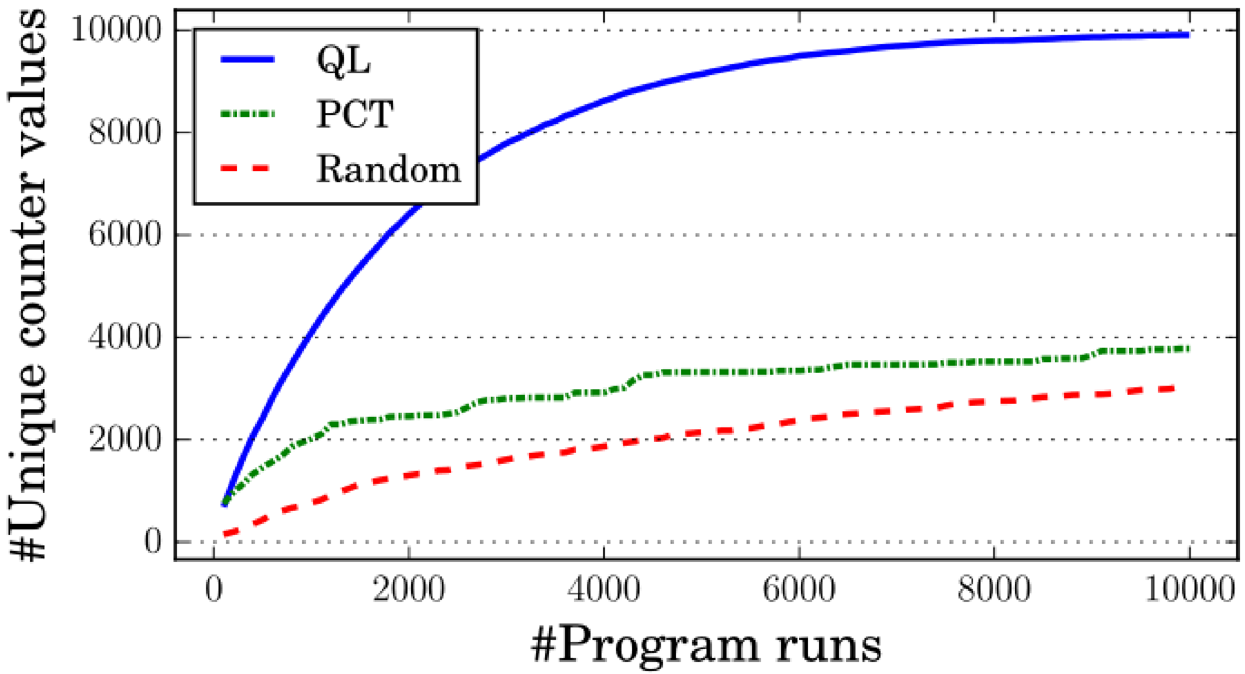
$Q(s_1, a_6) = 0$



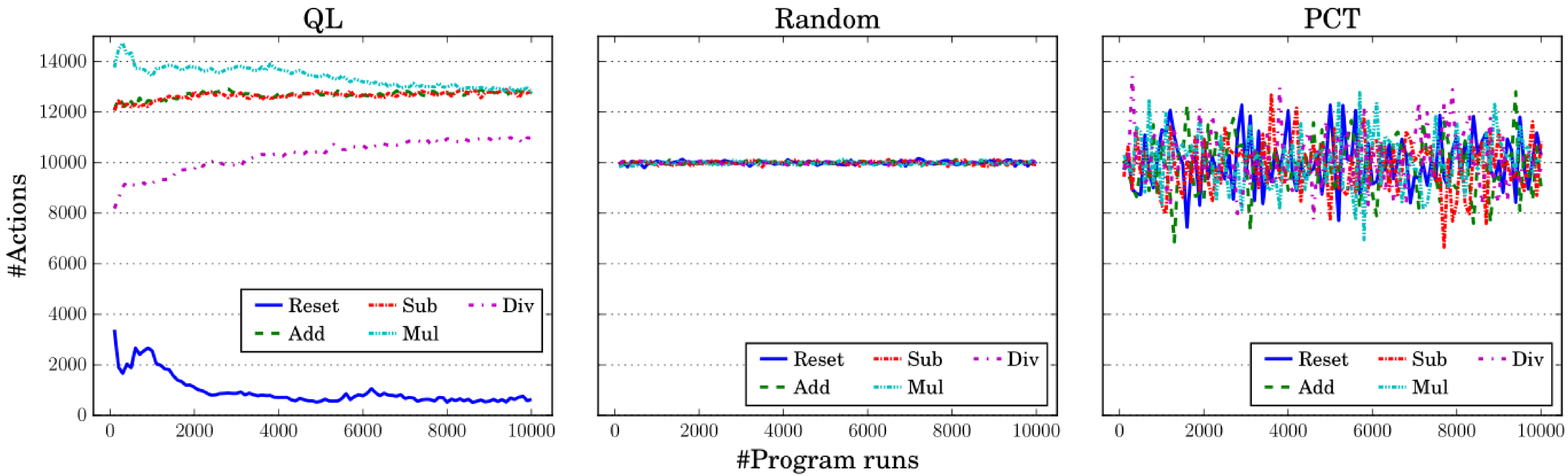
Optimizing for Coverage



-5000 <= C.counter <= 5000



Optimizing for Coverage



Raft Consensus Protocol

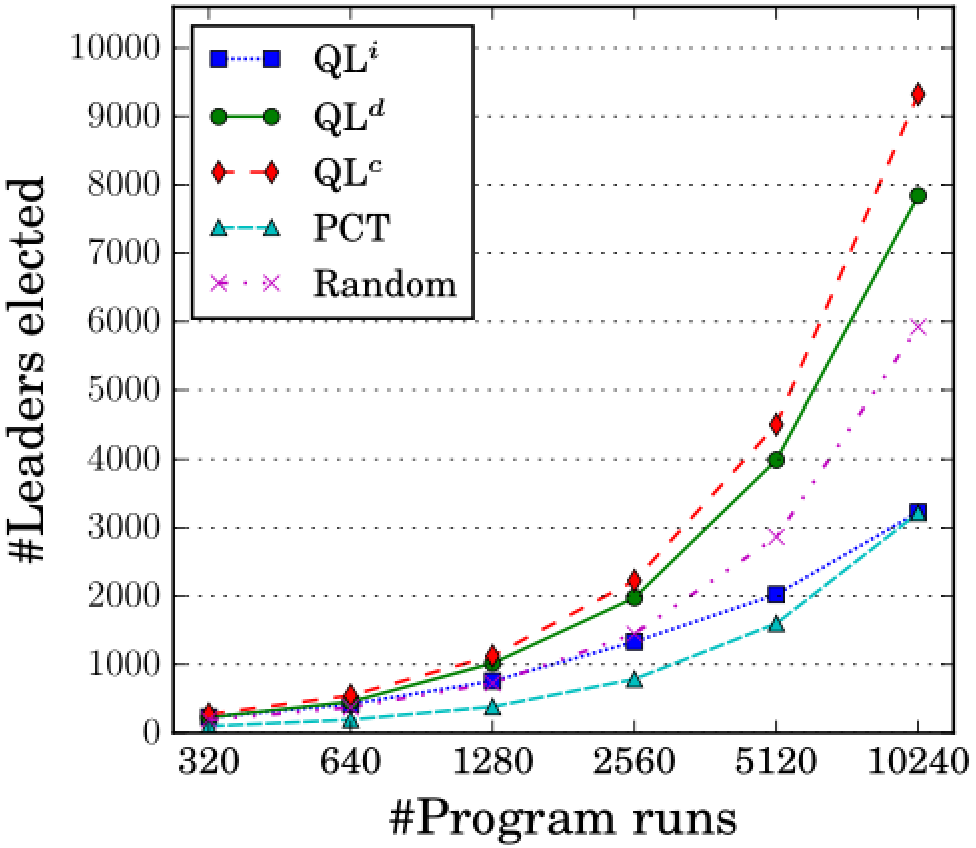
- Nodes:
  - *Leader*: receive and replicate client requests
  - *Follower*: Track all requests received from leader
  - *Candidate*: start leader election process at any time

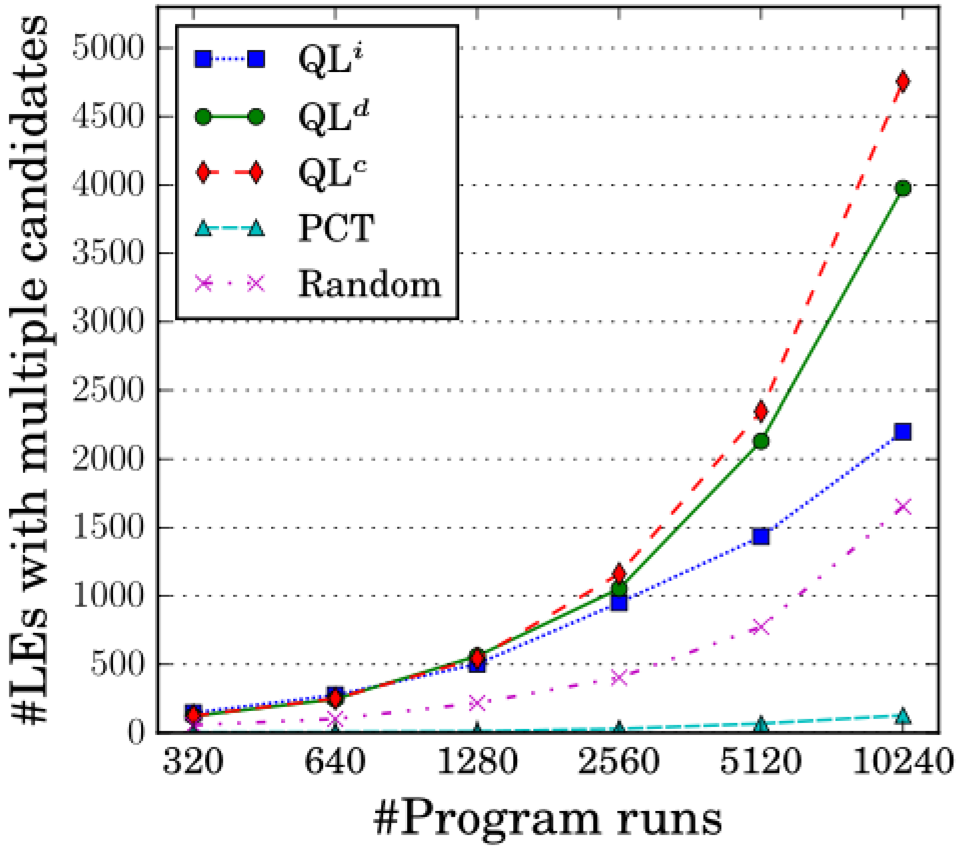
- Invariant:
  - *At most one leader at any point in time*.

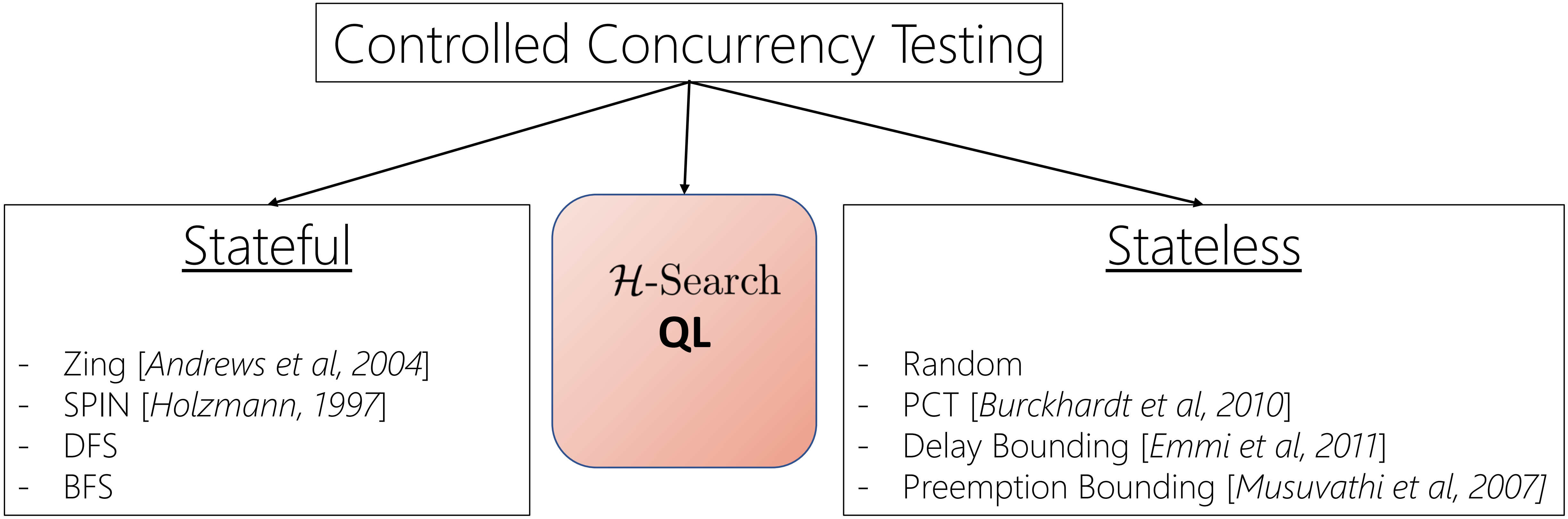
- Buggy implementation: violates the invariant

- To increase likelihood of bug, increase:
  - At least one leader must be elected
  - Leader election round should have *multiple* candidates

Raft Consensus Protocol






Controlled Concurrency Testing
Stateful
- Zing [Andrews et al, 2004]
- SPIN [Holzmann, 1997]
- DFS
- BFS
$\mathcal{H}$-Search
QL
Stateless
- Random
- PCT [Burckhardt et al, 2010]
- Delay Bounding [Emmi et al, 2011]
- Preemption Bounding [Musuvathi et al, 2007]

Experimental Evaluation

Effectiveness at bug-finding

	Benchmarks	LoC	#T	Bugs^{100} QL^d	Random	Greedy	PCT-3	PCT-10	PCT-30	IDB
Protocols	Raft-v1	1194	17	99	**100**	83	✗	12	45	28
	Raft-v2	1194	17	**95**	4	3	✗	✗	✗	1
	Paxos	849	10	66	8	20	19	91	**92**	33
	Chord	859	7	**34**	✗	✗	✗	✗	✗	✗
	FailureDetector	692	5	99	✗	✗	11	**100**	99	31
Multithreaded	Fib-Bench-2	55	3	**100**	**100**	**100**	✗	82	**100**	**100**
	Fib-Bench-Longest-2	55	3	**100**	**100**	**100**	✗	✗	✗	**100**
	Triangular-2	73	3	**100**	86	**100**	✗	✗	2	70
	Triangular-Longest-2	73	3	**100**	✗	79	✗	✗	✗	✗
	SafeStack	253	6	1	43	23	✗	✗	21	**46**
	BoundedBuffer	284	9	**53**	12	36	✗	✗	✗	✗
Production	ProdService1	56649	27	**79**	14	24	37	29	25	23
	ProdService2-v1	33827	15	**100**	✗	✗	**100**	**100**	37	✗
	ProdService2-v2	33827	28	97	✗	✗	**100**	36	✗	✗
	ProdService3-v1	18663	17	92	**100**	16	76	96	80	✗
	ProdService3-v2	19771	17	**100**	**100**	10	64	**100**	90	✗
	G-Mean Bugs^{100}			**63.9**$_{(16)}$	37.4$_{(11)}$	32.2$_{(12)}$	45.0$_{(7)}$	59.2$_{(9)}$	40.4$_{(10)}$	30.3$_{(9)}$

Experimental Evaluation

Effectiveness at bug-finding

	Benchmarks	LoC	#T	Bugs^{100} QL^{d}	Random	Greedy	PCT-3	PCT-10	PCT-30	IDB
Protocols	Raft-v1	1194	17	99	**100**	83	✗	12	45	28
	Raft-v2	1194	17	**95**	4	3	✗	✗	✗	1
	Paxos	849	10	66	8	20	19	91	**92**	33
	Chord	859	7	**34**	✗	✗	✗	✗	✗	✗
	FailureDetector	692	5	99	✗	✗	11	**100**	99	31
Multithreaded	Fib-Bench-2	55	3	**100**	**100**	**100**	✗	82	**100**	**100**
	Fib-Bench-Longest-2	55	3	**100**	**100**	**100**	✗	✗	✗	**100**
	Triangular-2	73	3	**100**	86	**100**	✗	✗	2	70
	Triangular-Longest-2	73	3	**100**	✗	79	✗	✗	✗	✗
	SafeStack	253	6	1	43	23	✗	✗	21	**46**
	BoundedBuffer	284	9	**53**	12	36	✗	✗	✗	✗
Production	ProdService1	56649	27	**79**	14	24	37	29	25	23
	ProdService2-v1	33827	15	**100**	✗	✗	**100**	**100**	37	✗
	ProdService2-v2	33827	28	97	✗	✗	**100**	36	✗	✗
	ProdService3-v1	18663	17	92	**100**	16	76	96	80	✗
	ProdService3-v2	19771	17	**100**	**100**	10	64	**100**	90	✗
	G-Mean Bugs^{100}			**63.9**$_{(16)}$	37.4$_{(11)}$	32.2$_{(12)}$	45.0$_{(7)}$	59.2$_{(9)}$	40.4$_{(10)}$	30.3$_{(9)}$

Experimental Evaluation

Effectiveness at bug-finding

	Benchmarks	LoC	#T	Bugs[100] QL[d]	Random	Greedy	PCT-3	PCT-10	PCT-30	IDB
Protocols	Raft-v1	1194	17	99	**100**	83	✗	12	45	28
	Raft-v2	1194	17	**95**	4	3	✗	✗	✗	1
	Paxos	849	10	66	8	20	19	91	**92**	33
	Chord	859	7	**34**	✗	✗	✗	✗	✗	✗
	FailureDetector	692	5	99	✗	✗	11	**100**	99	31
Multithreaded	Fib-Bench-2	55	3	**100**	**100**	**100**	✗	82	**100**	**100**
	Fib-Bench-Longest-2	55	3	**100**	**100**	**100**	✗	✗	✗	**100**
	Triangular-2	73	3	**100**	86	**100**	✗	✗	2	70
	Triangular-Longest-2	73	3	**100**	✗	79	✗	✗	✗	✗
	SafeStack	253	6	1	43	23	✗	✗	21	**46**
	BoundedBuffer	284	9	**53**	12	36	✗	✗	✗	✗
Production	ProdService1	56649	27	**79**	14	24	37	29	25	23
	ProdService2-v1	33827	15	**100**	✗	✗	**100**	**100**	37	✗
	ProdService2-v2	33827	28	97	✗	✗	**100**	36	✗	✗
	ProdService3-v1	18663	17	92	**100**	16	76	96	80	✗
	ProdService3-v2	19771	17	**100**	**100**	10	64	**100**	90	✗
	G-Mean Bugs[100]			**63.9**$_{(16)}$	37.4$_{(11)}$	32.2$_{(12)}$	45.0$_{(7)}$	59.2$_{(9)}$	40.4$_{(10)}$	30.3$_{(9)}$

Summary

- Novel controlled concurrency testing, based on Q-Learning
  - balance between taking random steps and informed decisions based on previous explorations
- Evaluation: outperforms state-of-the-art strategies on **production Azure services**

Big Picture

- ➢ Project Coyote: https://github.com/microsoft/coyote
  - Making testing of concurrent programs as easy as testing sequential programs
  - Used by many teams in Azure for writing distributed services
  - Includes great learning material for teaching concurrency-related concepts

- Extending beyond .NET [ASE'21]
  - Cross-platform solution for controlled-concurrency testing: https://github.com/microsoft/coyote-scheduler