Formal Modeling and Analysis of Distributed Systems

Finding Critical Bugs Early!

Ankush Desai (ankushpd@)

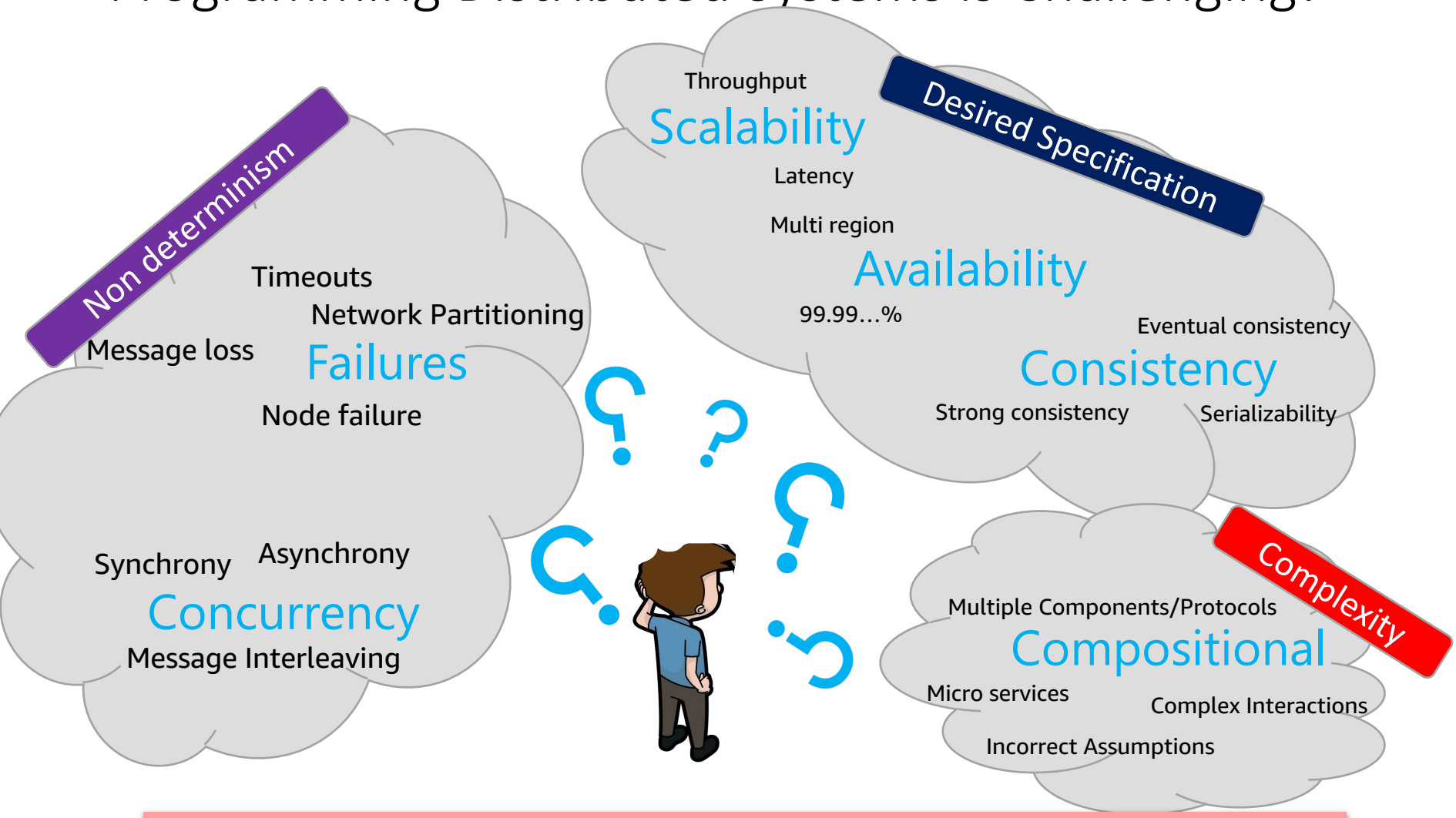
Sr. Applied Scientist,
Database Systems @ Amazon Web Services

https://github.com/p-org/P

Formal Methods

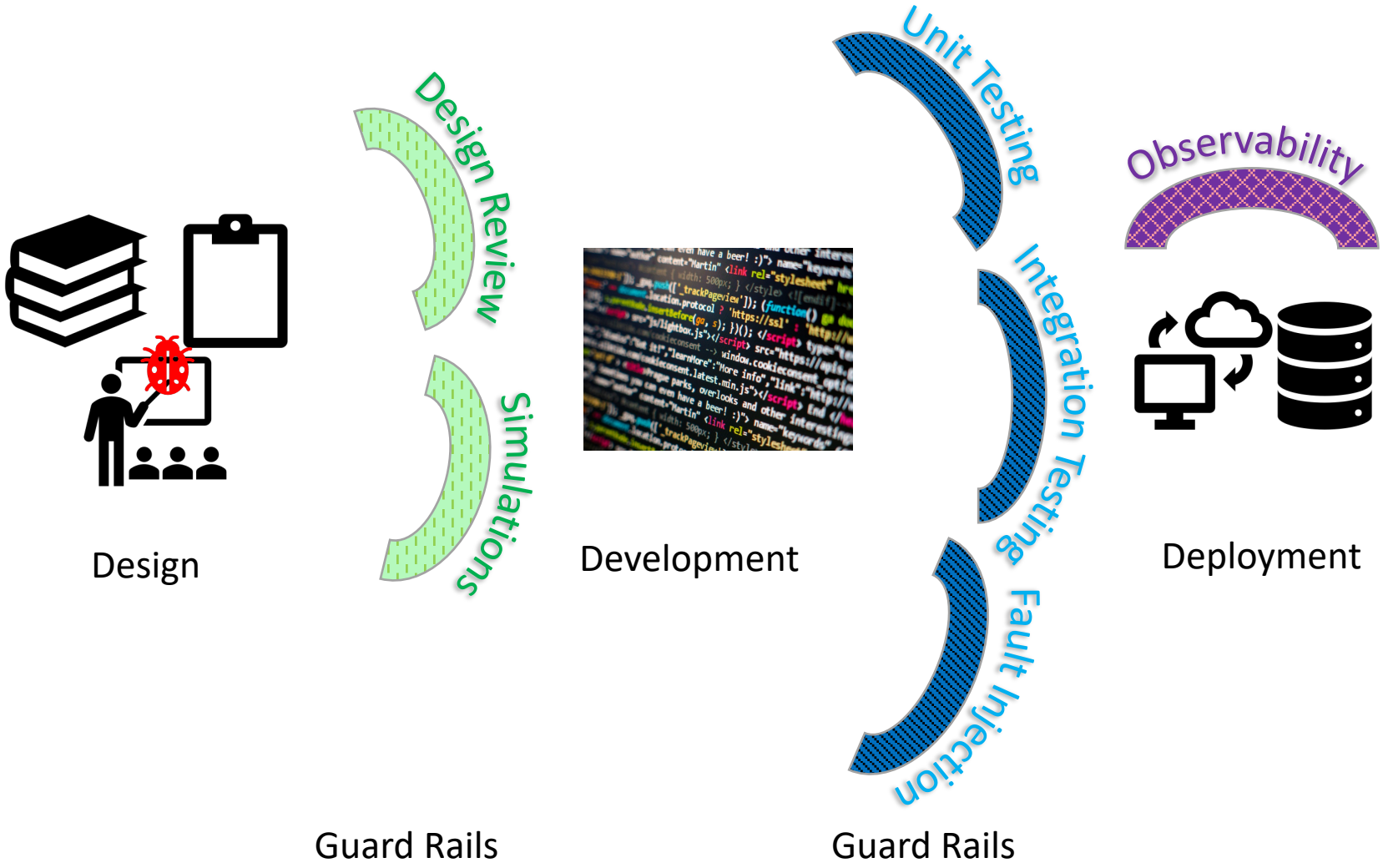
Its not just about the **Proofs**, its also about the **Process**!

Programming Distributed Systems is Challenging!



Traditional testing approaches fail to provide the strong assurance!

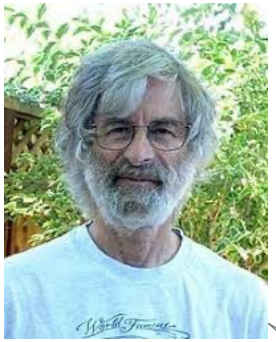
Not uncommon to find bugs in production after deployment



Formal Methods!

"*formal methods: any process that involves writing formal specifications and checking systems behavior against those specifications*"

Thinking abstractly, formally, above coding



People confuse Programming with Coding. Coding is to Programming what Typing is to Writing

Key is to abstract away unwanted details and think of the system above coding, that improves developers ability to understand systems better.

You don't understand something until you've written it down carefully – carefully enough to explain it to somebody else (formal models). And if you haven't done that, you're just thinking you understand it,

From Talks:

The Man Who Revolutionized Computer Science With Math

How to think clearly as a programmer with mathematics and TLA+ - Leslie Lamport @ HLF 2019

Challenges with wide spread adoption of Formal Methods!

- [Steep Learning Curve] : X is not programmer friendly ..
  - Mathematical modeling language, huge overhead of maintaining the models
  - Formal design and implementation go out of sync.
  - Developers think of distributed system as communicating state machines, instead of a monolithic state transition system.

- [Scalability]: X does not scale for verification of real-world distributed systems.
  - Fails to find deep or corner case bugs.

- [Connection to Implementation]: what about the gap between design and implementation.

X is any formal methods tool.

(https://github.com/p-org/P)

- P is a state-machine based programming framework for modeling, specifying, and formal reasoning of distributed systems.

- P program is a collection of asynchronously communicating state machines.

  Models and specifications are implemented as state machines.

- P supports a scalable backend checker.

  State-of-the-art techniques to scale analysis to large distributed systems

```
machine Client
{
  var server : Server;
  var nextReqId : int;
  var lastSuccessfulRespId: int;

  start state Init {
    entry (payload : Server)
    {...}
  }

  state StartPumpingRequests {
    entry
    {...}
    on eResponse do (resp: tResponse){...}
    on eRequest goto ServiceRequests;
  }
}
```

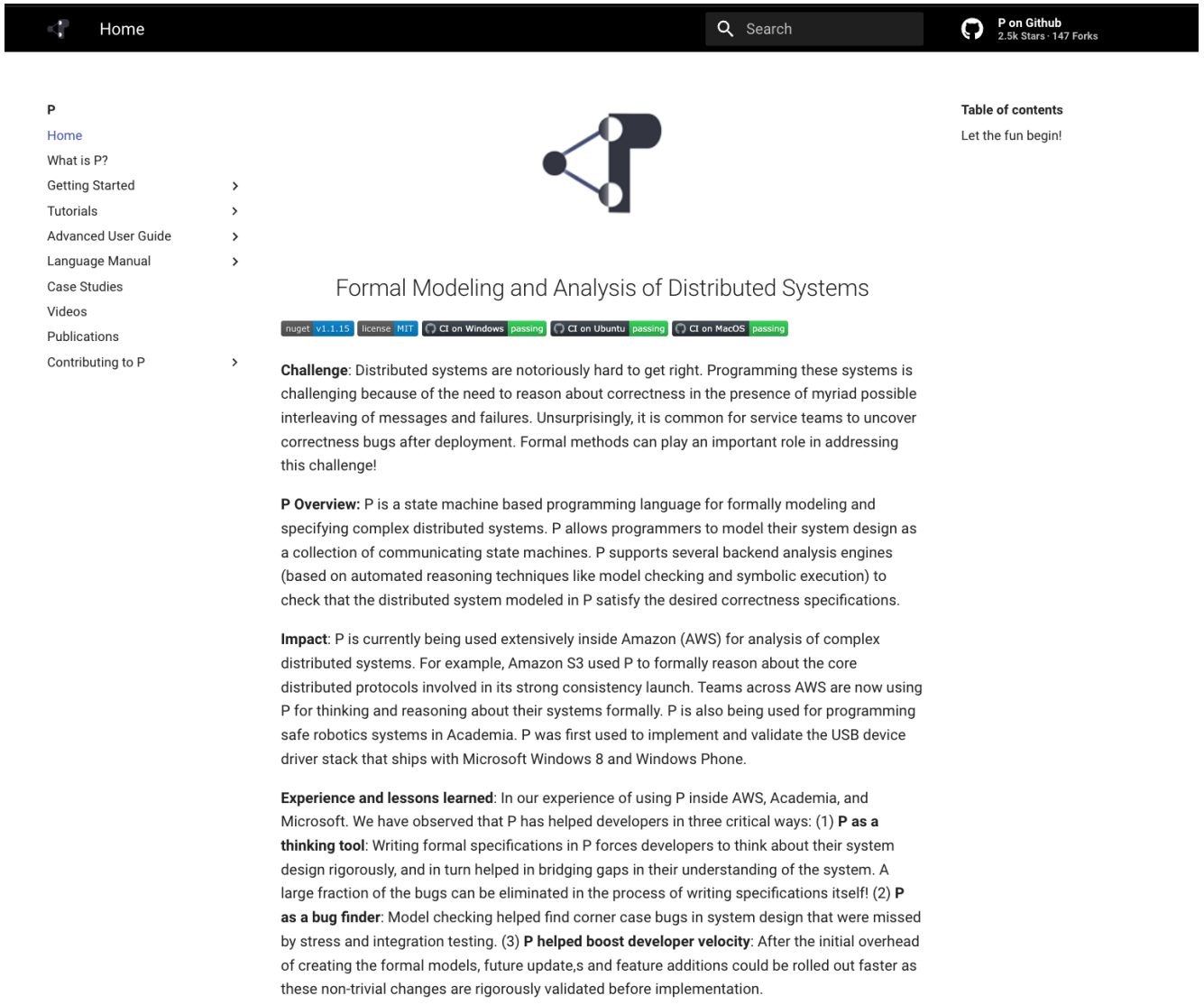
P Specifications: Runtime Monitors

```
event eA: int;
event eB: int;

spec AlternatingAB observes eA, eB {
  var lastVal: int; // local state
  start state WaitForA {
    on eA goto WaitForB with (payload: int) {
      assert payload > lastVal;
      lastVal = payload;
    }
  }

  state WaitForB {
    on eB goto WaitForA with (payload: int) {
      assert payload > lastVal;
      lastVal = payload;
    }
  }
}
```

- Specifications are runtime monitors that listen to messages, maintain local state, and assert global invariants!
- Specifications are programs!

P Tutorials and Documentation

https://p-org.github.io/P/



History of P (Slides taken from my PhD. talk)

Users in Microsoft

Windows (P)	Azure (P#)	Office
USB host	Node Service	Office Client
USB function	Batch Service	
UART class extension	Learning Service	
Hid class	AZSM	
USB Type C stack	CAT (Connected Car)	
Media Agnostic USB	Blockchains	
Bluetooth		

Users in Academia

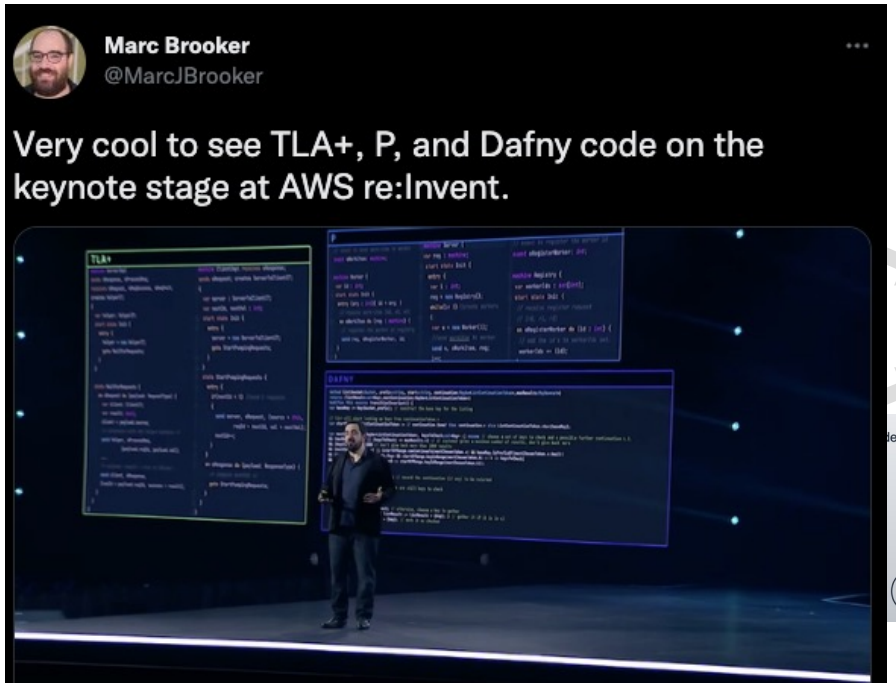
- Used P to model and verify distributed IEEE 1588 Precision time protocol [CAV 2015].
  - Used as a standard in telecommunication, industrial automation, and CERN LHC particle accelerator.
- We were able to reproduce a long debated bug of "*endlessly circulating frames*" in IEEE1588 using our verification approach.
  - The counter example was confirmed using a simulator and well received by the IEEE 1588 standards community.

Formal Reasoning of S3 Strong Consistency Design using P

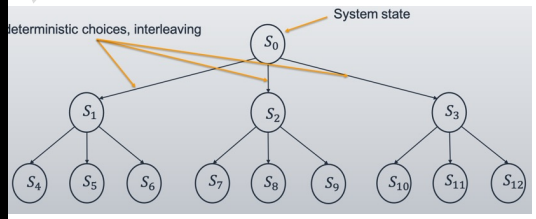
AWS News Blog

Amazon S3 Update – Strong Read-After-Write Consistency

by Jeff Barr | on 01 DEC 2020 | in Amazon Simple Storage Services (S3), Launch, News | Permalink | Comments | Share



Service teams across AWS (compute, storage, database) are using P to reason about their core distributed protocols.



systematically explore state space to check if model of a system satisfies desired specifications.

Lessons Learned (P as a Thinking Tool)

Formal Modeling: "P as a Thinking Tool"

Design as reference executable specification, bridges the gap in understanding.

Formal Models (PSrc)	**"How the system does what it is supposed to do?"** Think about **detailed design**, interaction with other components, messages exchanged, assumptions about other components.
Formal Specifications (PSpec)	**"Does the system do what it is supposed to do?"** Safety and liveness properties. **Global invariants** both external and internal that the system must satisfy.
Inputs and Fault Modeling (PTst)	**"What inputs and faults can the system tolerate?"** Think about the kind of inputs and faults the system must tolerate.

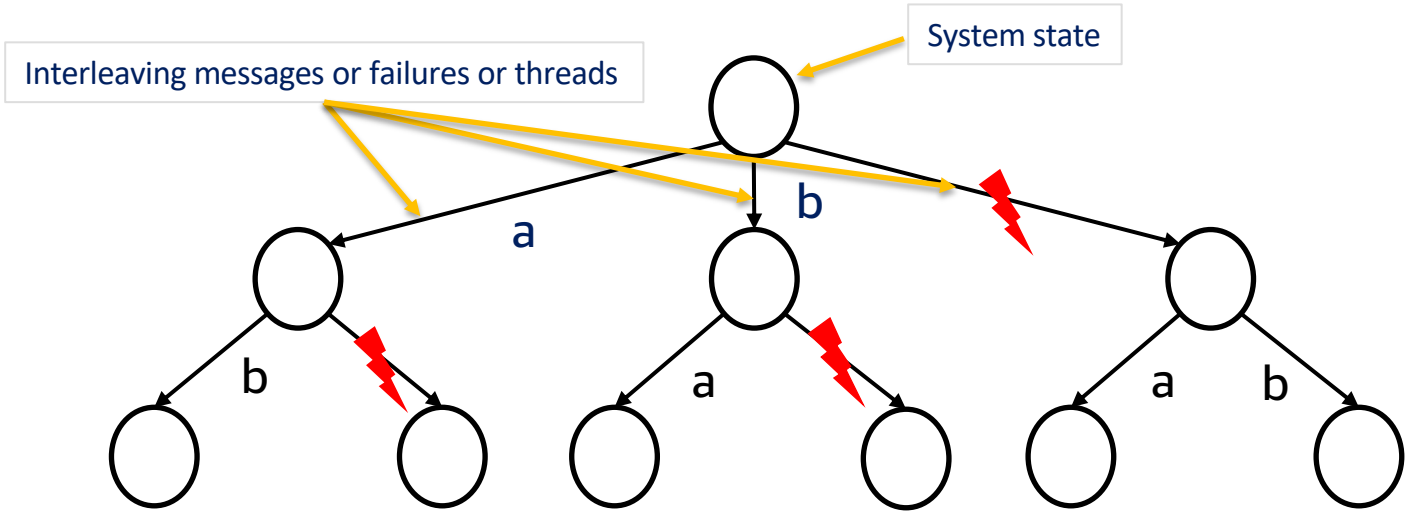
Lessons Learned (P as a systematic explorer or bug finder)

Model Checking: "P as a Systematic explorer or bug finder"
Uncover non-trivial bugs/scenarios, gain confidence in design.

P Checker: Systematically explores nondeterministic behaviors.

- Two forms of non-determinism:
  - Data non-determinism (e.g., all possible inputs, random(), timeouts)
  - Scheduling non-determinism (e.g., interleaving of messages, failures)

Model Checking as a search problem



Scalable Exploration: Partial Order Reduction, Search Prioritization, Symbolic Execution, ..

Continued Assurance

- Leverage AWS to scale systematic exploration:
  - Explore billions of states using 100s of compute nodes every day.
  - Explore combinations of rare events (extremely low probability).

- Integrate model checking into CI/CD pipeline.

Lessons Learned (P boosts velocity of developers)

- **Maintenance (quick iterations):** "P boosts velocity of developers"

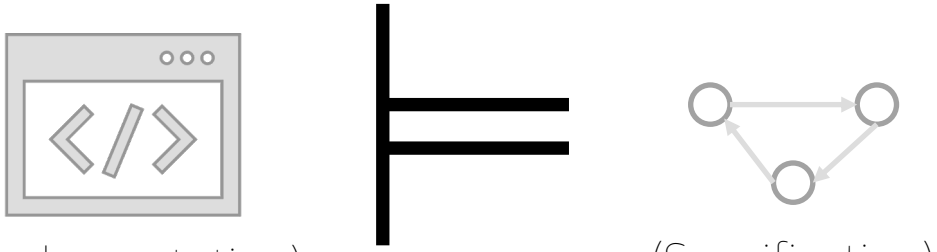
Rapid iterations, release features faster, eliminate bugs early instead of in production.

Service teams can try out risky optimizations or protocol changes first using P, gain confidence, and then mimic those changes in the implementation.

Higher confidence in design before moving to implementation.

What about connection to Implementation?

- We want to check specifications on the system implementation.

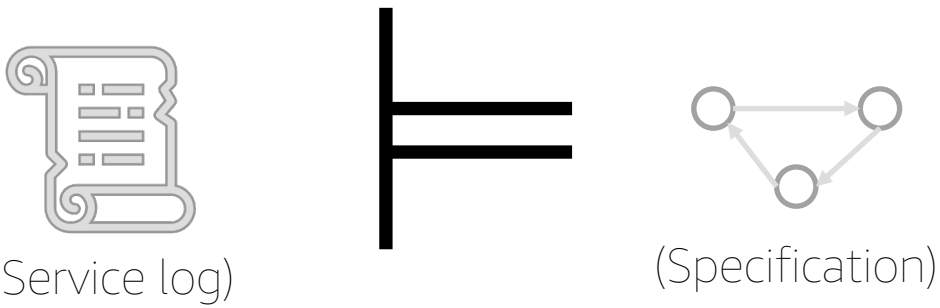


(Implementation) (Specification)

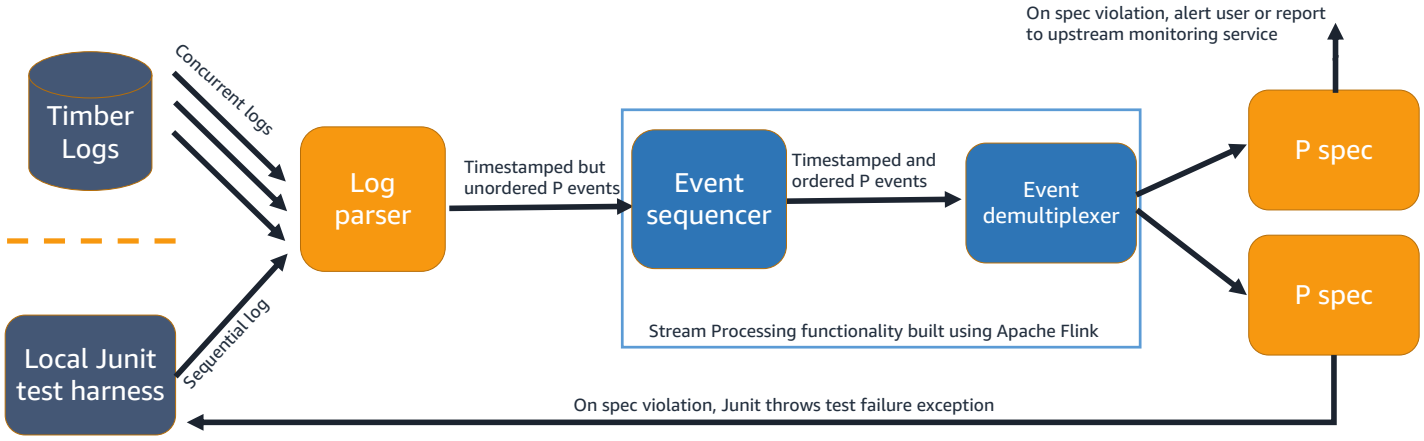
- Systems already produce service logs!

What about connection to Implementation?

- *Runtime monitoring* is a well-known technique for checking that a system *trace* satisfies a desired *property*.
  - Trace = Service log
  - Property = P specification



- **Idea:** Check P Specifications on service logs!



PObserve architecture; components in orange are service-specific and are provided by developers.

Improving observability of distributed systems using formal specifications.

We define *distributed systems' observability* as the ability to check formal specifications at all the phases of development lifecycle (design, testing, and production).

Formal Methods

Its not just about the **Proofs**, its also about the **Process**!

https://github.com/p-org/P