

Testing Distributed Systems

A systematic approach to
reliability

Jack Vanlightly
Software Engineer/Tester
at SII Concatel

 RabbitMQ

 kafka

 PULSAR

Distributed Systems Are Hard...

- Multiple processes, actors and the environment interacting concurrently
- Hard to reason about, hard to test:
 - combinatorial explosion of possible interactions

Race Conditions

There is no now (clock skew)

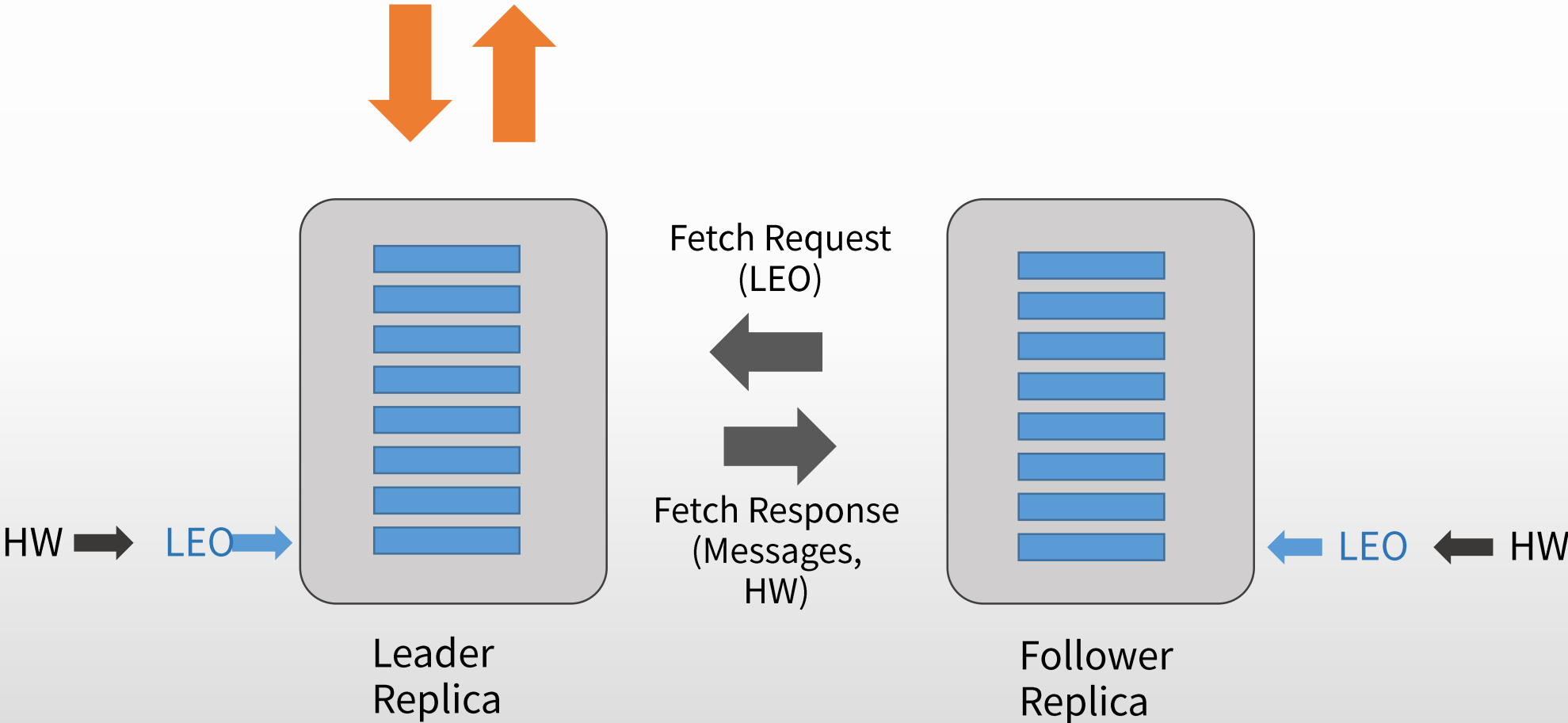
Failures

Concurrency

Unanticipated sequences of events

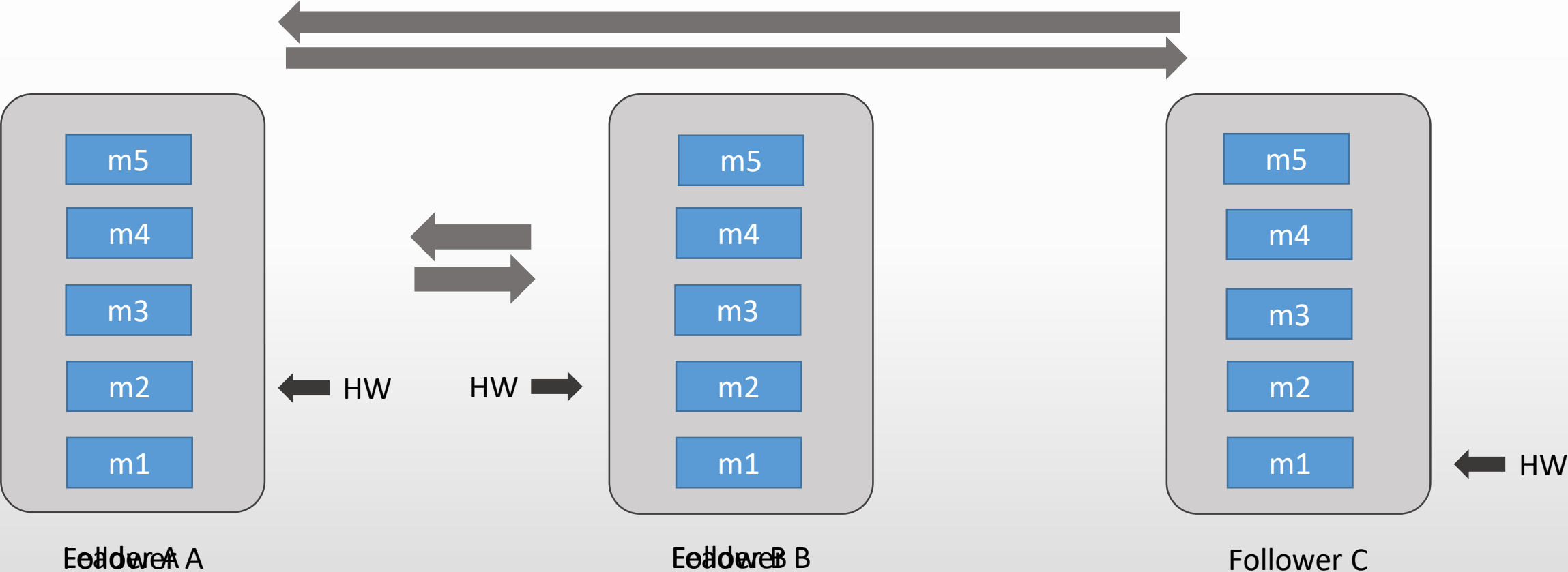
Case Study – Apache Kafka Replication

Introduction to message replication



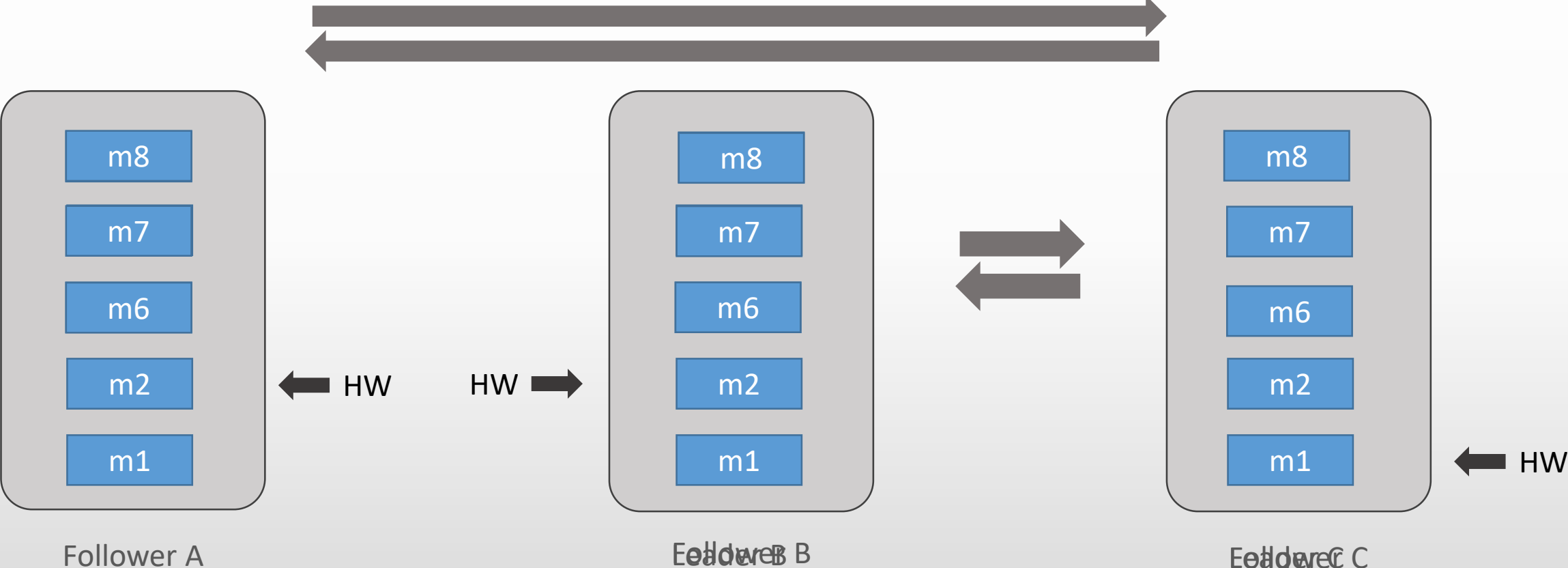
Case Study – Apache Kafka Replication

Leader Fail-Over and Without Message Loss



Case Study – Apache Kafka Replication

Leader Fail-Over and With Message Loss



Case Study – Apache Kafka Replication

See the Protocol Defect?



Distributed Systems Are Hard...

A Test Strategy for Distributed
Systems...

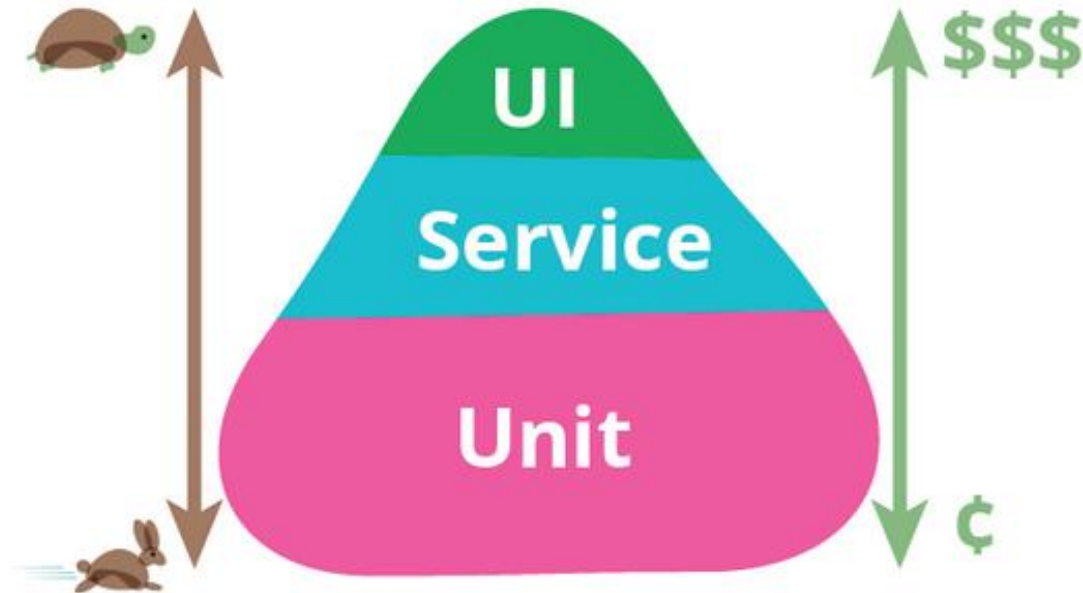
The Test Pyramid

TestPyramid



Martin Fowler
1 May 2012

The test pyramid is a way of thinking about different kinds of automated tests should be used to create a balanced portfolio. Its essential point is that you should have many more low-level **UnitTests** than high level **BroadStackTests** running through a GUI.



The Testing Trophy

 **Guillermo Rauch** ✓
@rauchg


Following ▾

Write tests. Not too many. Mostly integration.

8:43 AM - 10 Dec 2016 from San Francisco, CA

111 Retweets 357 Likes

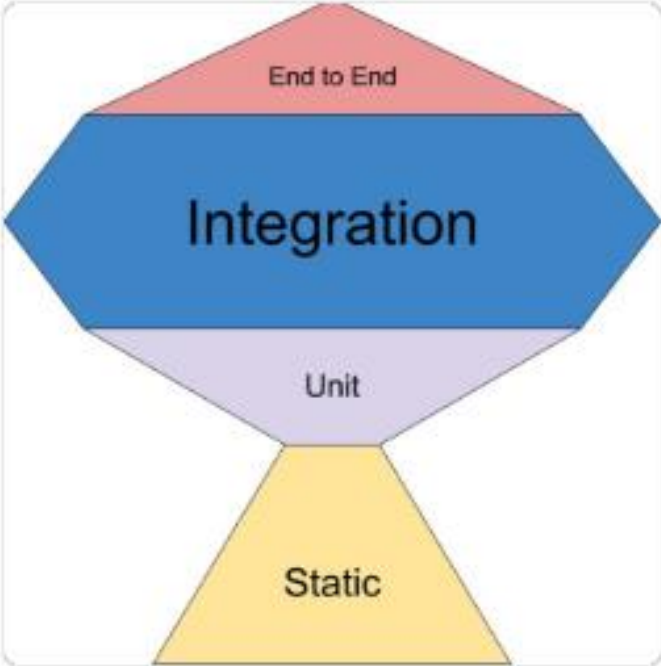
16 111 357

 **Kent C. Dodds**
@kentcdodds **Follow** ▾

"The Testing Trophy" 🏆

A general guide for the ****return on investment**** 🤖 of the different forms of testing with regards to testing JavaScript applications.

- End to end w/ @Cypress_io ●
- Integration & Unit w/ @fbjest 🏆
- Static w/ @flowtype *F* and @geteslint ●

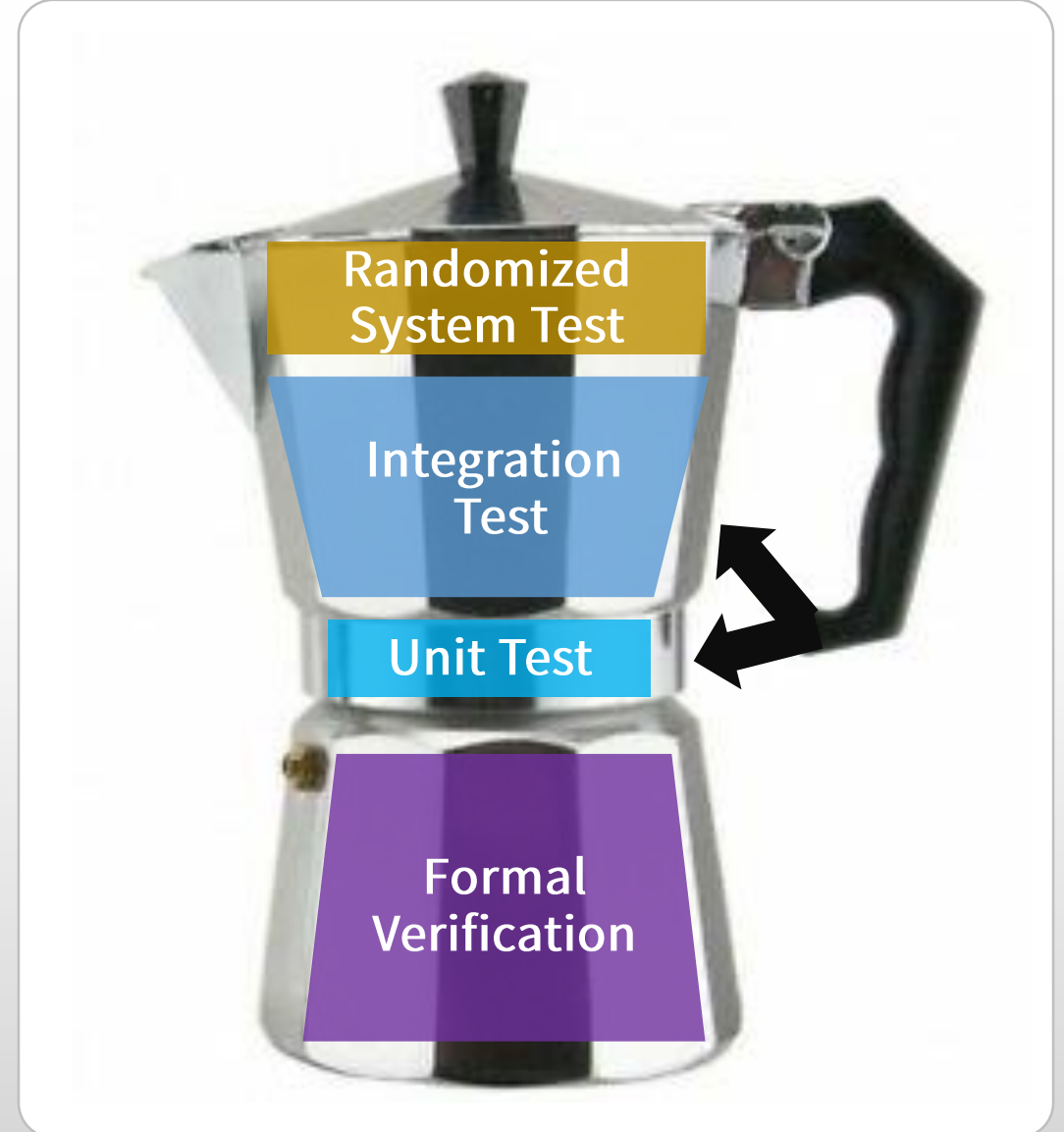


7:53 PM - 5 Feb 2018

The Testing Cafetiere



A multi-layered test strategy for distributed systems.



Why is Formal Verification needed?

“ human fallibility means that some of the more subtle, dangerous bugs turn out to be errors in design;

the code faithfully implements the intended design, but the design fails to correctly handle a particular ‘rare’ scenario.

We have found that testing the code is inadequate as a method to find subtle errors in design.

Use of Formal Methods at Amazon Web Services (Paper)

Designs need to be tested too

What is Formal Verification?

“ ... formal verification is the act of proving or disproving the correctness of intended algorithms...

with respect to a certain formal specification or property,

using formal methods of mathematics.

Wikipedia

Sounds kinda scary...

What is a Formal Verification language?

“ TLA+ is a formal specification and verification language that helps engineers design, specify, reason about and verify complex, real-life algorithms and software or hardware systems.

Ron Pressler

Less scary...

What is a Formal Verification language?

“ Exhaustively testable pseudo-code

Use of Formal Methods at Amazon Web Services (Paper)

Sounds pretty cool...

... it's precise designs

“

In order to find subtle bugs in a system design, it is necessary to have a precise description of that design ...

In contrast, conventional design documents consist of prose, static diagrams, and perhaps pseudo-code in an ad hoc untestable language.

Such descriptions are far from precise; they are often ambiguous, or omit critical aspects.

Use of Formal Methods at Amazon Web Services (Paper)

Complex designs need precision...

... it's verifying correctness

“ Correctness of the algorithm means that the program satisfies a desired property.

Leslie Lamport

Correctness via Properties...

What are properties?

- Committed messages are not lost

Safety

- Messages are stored in temporal order

Safety

- Up to the High Watermark, of a given partition, each replica consists of the same messages in the same order

Safety

A safety property is one that is true for every reachable state

AKA an Invariant

What are properties?

- If a message is sent to the leader replica, it must eventually be replicated to all followers, as long as the leader does not die

Liveness

- If the leader dies and there are followers, then eventually a follower will be elected leader

Liveness

A liveness property describes what must eventually happen

TLA+ and TLC Together

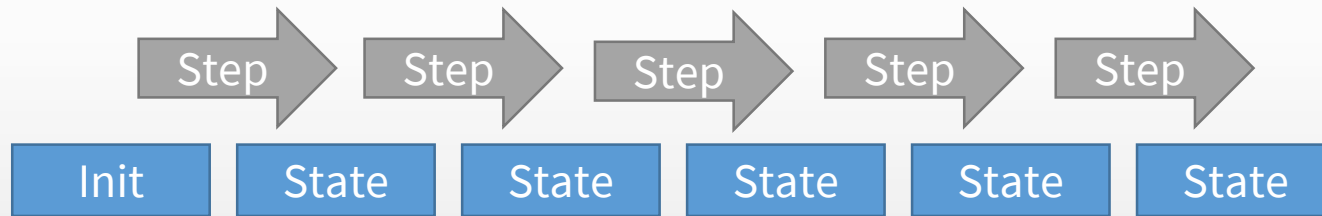
- **A TLA+ specification describes:**
 - The behaviour of the system
 - Properties
 - Safety properties (invariants)
 - Liveness properties
- **TLC (The model checker)**
 - Executes the specification
 - Exhaustively checks all possible action sequences and while verifying that invariants hold

Wow, a testing framework for designs...

TLA+: A State Machine of States and Steps

Steps and States

TLA+ models time as a sequence of discrete steps, where one step transitions from one state to another.



Each step consists of one or more actions.

Back to Apache Kafka Replication...

Define the Actions

The actions in our system:

1. Start a node
2. Shutdown a node
3. Become the partition leader
4. Receive a message
5. Become a follower
6. Send a fetch request
7. Send a fetch response

Back to Apache Kafka Replication...

Define the Actions

What states are in our system?

1. A single node, without role, no messages, HW 0
2. A single node, leader, no messages, HW 0
3. A single node, leader, 1 message, HW 1
4. A single node, leader, 2 messages, HW 2
5. Two nodes:
 - leader, 2 messages, HW 2
 - none, 0 messages, HW 0

Back to Apache Kafka Replication...

Define the Actions

What states are in our system?

232. Two nodes:

- leader, 2 messages, HW 0
- follower, 1 message, HW 0

11,348. Three nodes:

- leader, 2 messages, HW 1
- follower, 2 messages, HW 0
- follower, 1 message, HW 0

Back to Apache Kafka Replication...

Define the Actions

What states are in our system?

87,348. Three nodes:

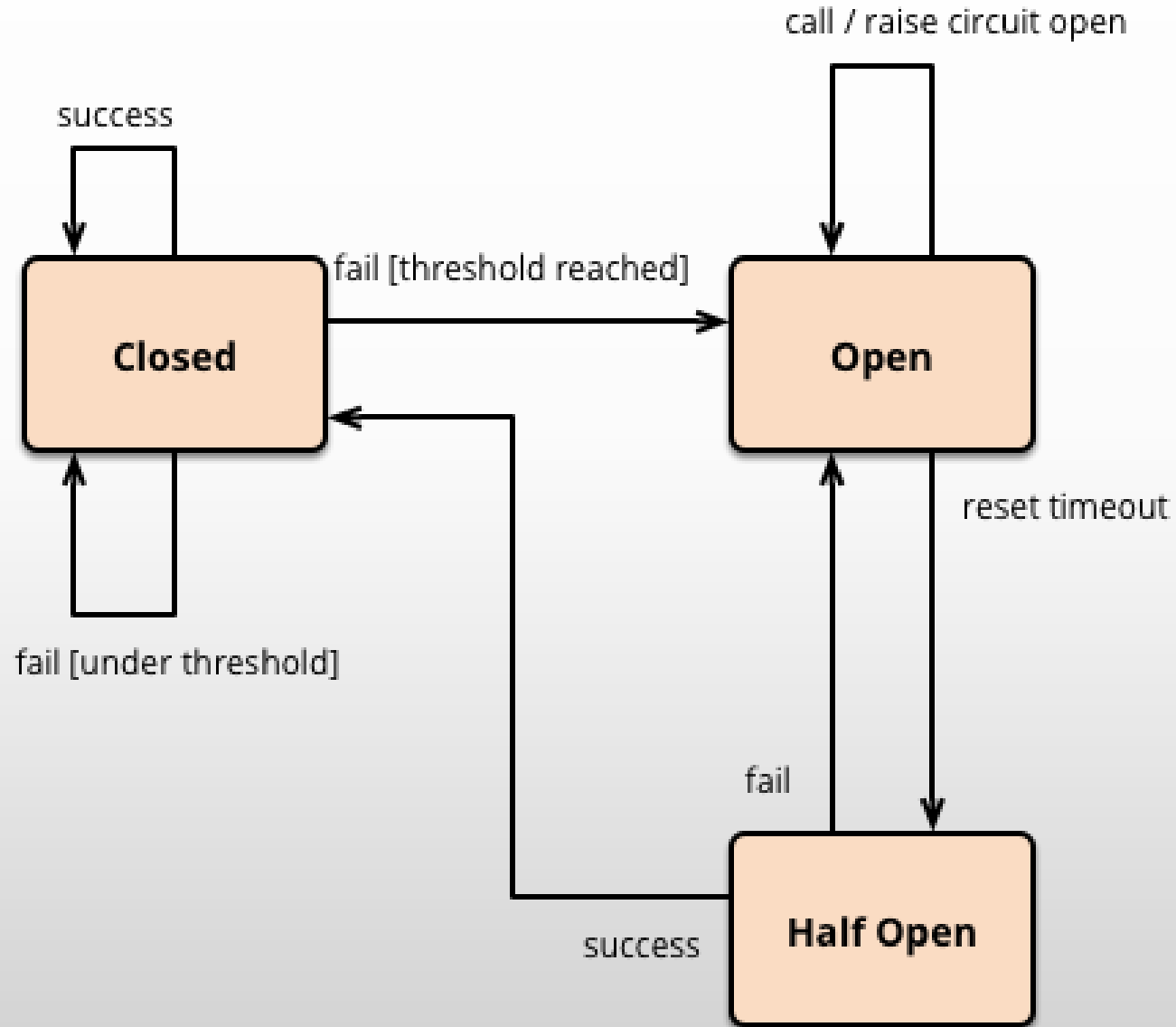
- leader, 8 messages, HW 6
- follower, 6 messages, HW 4
- follower, 6 messages, HW 4



State
Explosion

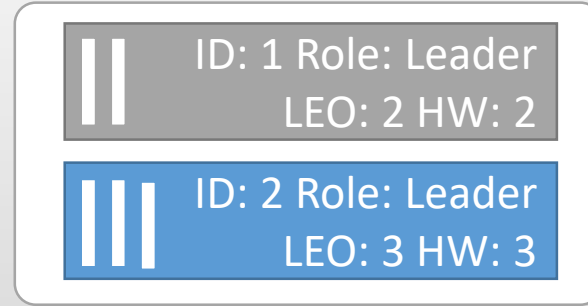
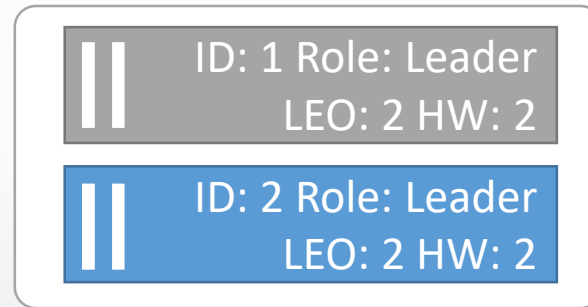
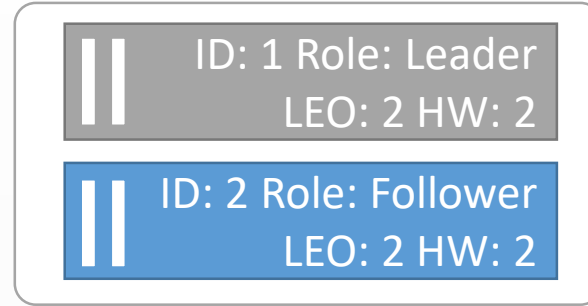
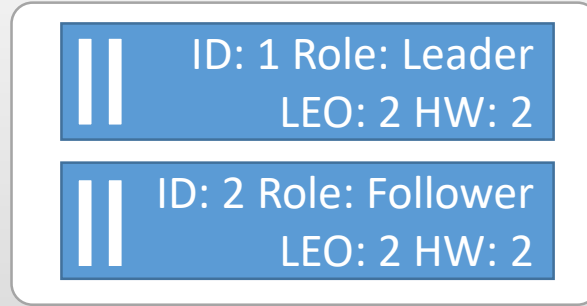
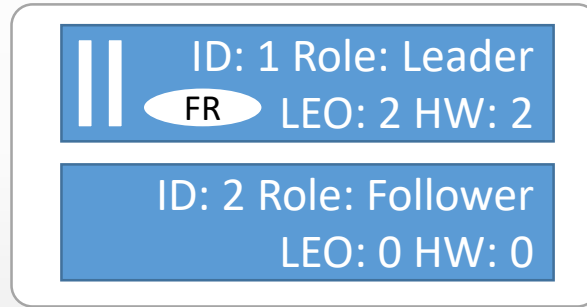
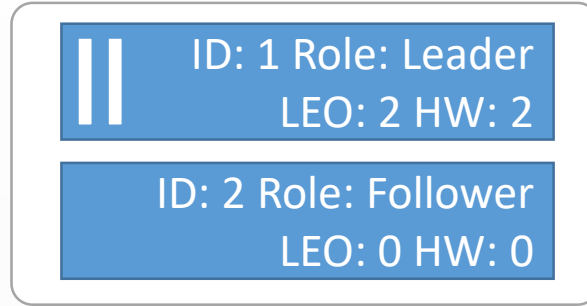
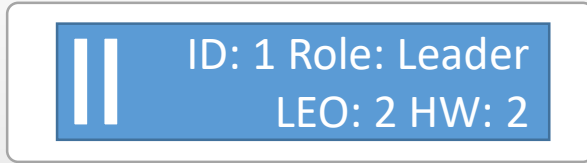
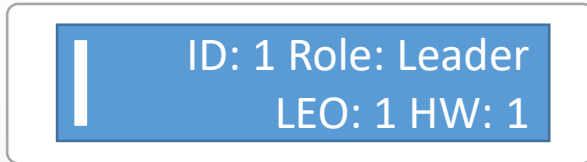
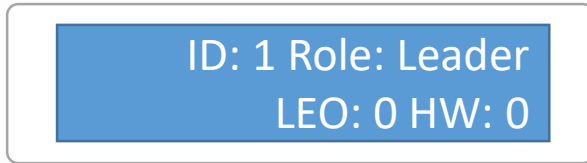
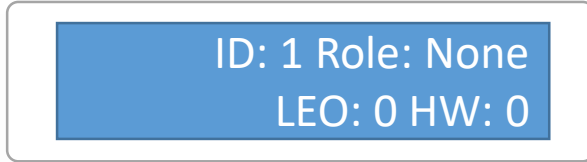
TLA+: A State Machine of States and Steps

Not your traditional state machine



TLA+: A State Machine of States and Steps

How many states are there?



- Start a node
- Become leader
- Receive message
- Receive Message
- Start a node
- Become follower
- Send fetch request
- Send fetch response
- Shutdown node
- Become leader
- Receive message

TLA+: A State Machine of States and Steps

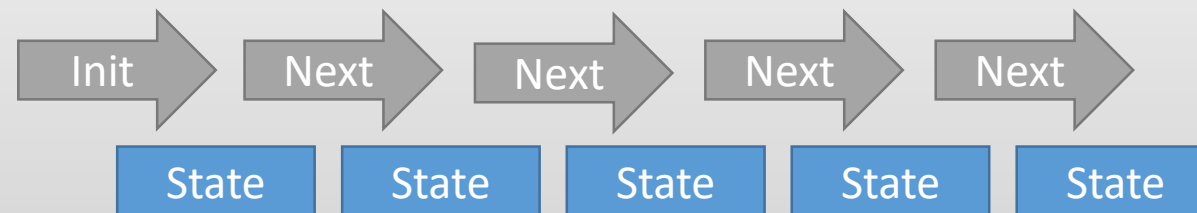
How does TLA+ represent states and steps?

State: A TLA+ specification has variables

Steps: A TLA+ specification is formed by two principal formulae:

Initial State Formula

Next State Formula



Back to Apache Kafka Replication...

Finally, some actual TLA+

What is the model?

Specify the values of declared constants.

```
N <- { "n1", "n2" }
```

```
CONSTANTS N \* the set of Kafka nodes
VARIABLES node_data,
             node_ldr,
             confirmed_leo,
             pending_req,
             \* meta data variables
             curr_val,
             committed_msgs,
```

Back to Apache Kafka Replication...

Define the Initial State

What is the model?

Specify the values of declared constants.

```
N ← { "n1", "n2" }
```

```
Init ==  
  /\ node_ldr = "-"  
  /\ node_data = [n \in N |-> [id |-> n,  
                                role |-> "offline",  
                                hw |-> 0, leo |-> 0,  
                                log |-> {}]]  
  
  /\ pending_req = [n \in N |-> 0]  
  /\ confirmed_leo = [n \in N |-> 0]  
  /\ committed_msgs = {}  
  /\ curr_val = 1
```

Back to Apache Kafka Replication...

Define the Actions

\wedge AND
 \vee OR
 \exists there exists

```
Next ==  
  \/\ \E n \in OfflineNodes : StartNode(n)  
  \/\ \E n \in LiveNodes :  
    \/\ ShutdownNode(n)  
    \/\ BecomeLeader(n)  
    \/\ BecomeFollower(n)  
    \/\ SendFetchRequest(n)  
    \/\ SendFetchResponse(n)  
  
  \/\ ReceiveMessage
```


Back to Apache Kafka Replication...

Define the Actions

/\ AND
\/ OR
\E there exists

```
LiveNodes ==  
{ n \in N : node_data[n].role # "offline"}
```

```
def get_live_nodes():  
    set live_nodes = set()  
    for node_id in N:  
        set of keys of node_data map : where role is not offline  
        if node_data[node_id].role != "offline":  
            live_nodes.add(node_id)  
  
    return live_nodes
```

```
OfflineNodes ==  
{ n \in N : node_data[n].role = "offline"}
```

Back to Apache Kafka Replication...

Anatomy of an action

/\ AND
\/ OR
\E there exists

```
StartNode(n) ==
  \* enabling condition -----
  /\ node_data[n].role = "offline"
  \* next state -----
  /\ node_data' = [node_data EXCEPT \* no role yet
                  ![n].role = "none",
                  \* truncate log to HW
                  ![n].leo = node_data[n].hw,
                  ![n].log =
                    { msg \in node_data[n].log
                      : msg.offset <= node_data[n].hw
                    }]
  ]
```

Back to Apache Kafka Replication...

Anatomy of an action

/\ AND
\ / OR
\E there exists

```
ShutdownNode(n) ==
  \* enabling condition -----
  /\ Cardinality(LiveNodes) > 1
  /\ node_data[n].role # "offline"
  \* next state -----
  /\ node_data' = [node_data EXCEPT ![n].role = "offline"]
  /\ IF n = node_ldr
      THEN node_ldr' = "-"
      ELSE node_ldr' = node_ldr
  /\ UNCHANGED << curr_val, confirmed_leo, committed_msgs,
```

Back to Apache Kafka Replication...

Anatomy of an action

/\ AND
\ / OR
not equal

```
ReceiveMessage ==
  \* enabling condition -----
  /\ node_ldr # "-"
  \* next state -----
  /\ LET msgOffset == node_data[node_ldr].leo + 1
     IN /\ node_data' = [node_data EXCEPT ![node_ldr].log = @ \cup {[offset |-> msgOffset,
                                                                    val |-> curr_val]}],
        ![node_ldr].leo = msgOffset]
        /\ curr_val' = curr_val + 1
        /\ confirmed_leo' = [confirmed_leo EXCEPT ![node_ldr] = msgOffset]
  /\ UNCHANGED << node_ldr, isr, pending_req, committed_msgs >>
```

Back to Apache Kafka Replication...

Define our Invariants - Safety Properties

/\ AND
\/ OR
\A for all

```
NoLossOfCommittedMsgsInvariant ==  
  \* it is valid for there to be no leader  
  \/ node_ldr = "-"  
  \* in case of a leader, it must have all committed messages  
  \/ /\ node_ldr # "-"  
      /\ \A msg_value \in committed_msgs :  
          msg_value \in LogValuesOfNodeLeader
```

```
LogValuesOfNodeLeader ==  
  { log_item.val : log_item \in node_data[node_ldr].log }
```

Back to Apache Kafka Replication...

Find invariant violations with TLC – The Model Checker

```
<< <<"StartNode", "n1">>,
  <<"StartNode", "n2">>,
  <<"BecomeLeader", "n1">>,
  <<"BecomeFollower", "n2">>,
  <<"ReceiveMessage">>,
  <<"SendFetchRequest", "n2">>,
  <<"SendFetchResponse", "n2">>,
  <<"SendFetchRequest", "n2">>,
  <<"ReceiveMessage">>,
  <<"SendFetchResponse", "n2">>,
  <<"SendFetchRequest", "n2">>,
  <<"ShutdownNode", "n2">>,
  <<"StartNode", "n2">>,
  <<"ShutdownNode", "n1">>,
  <<"BecomeLeader", "n2">> >>
```

TLA+: It's Weird and Wonderful...

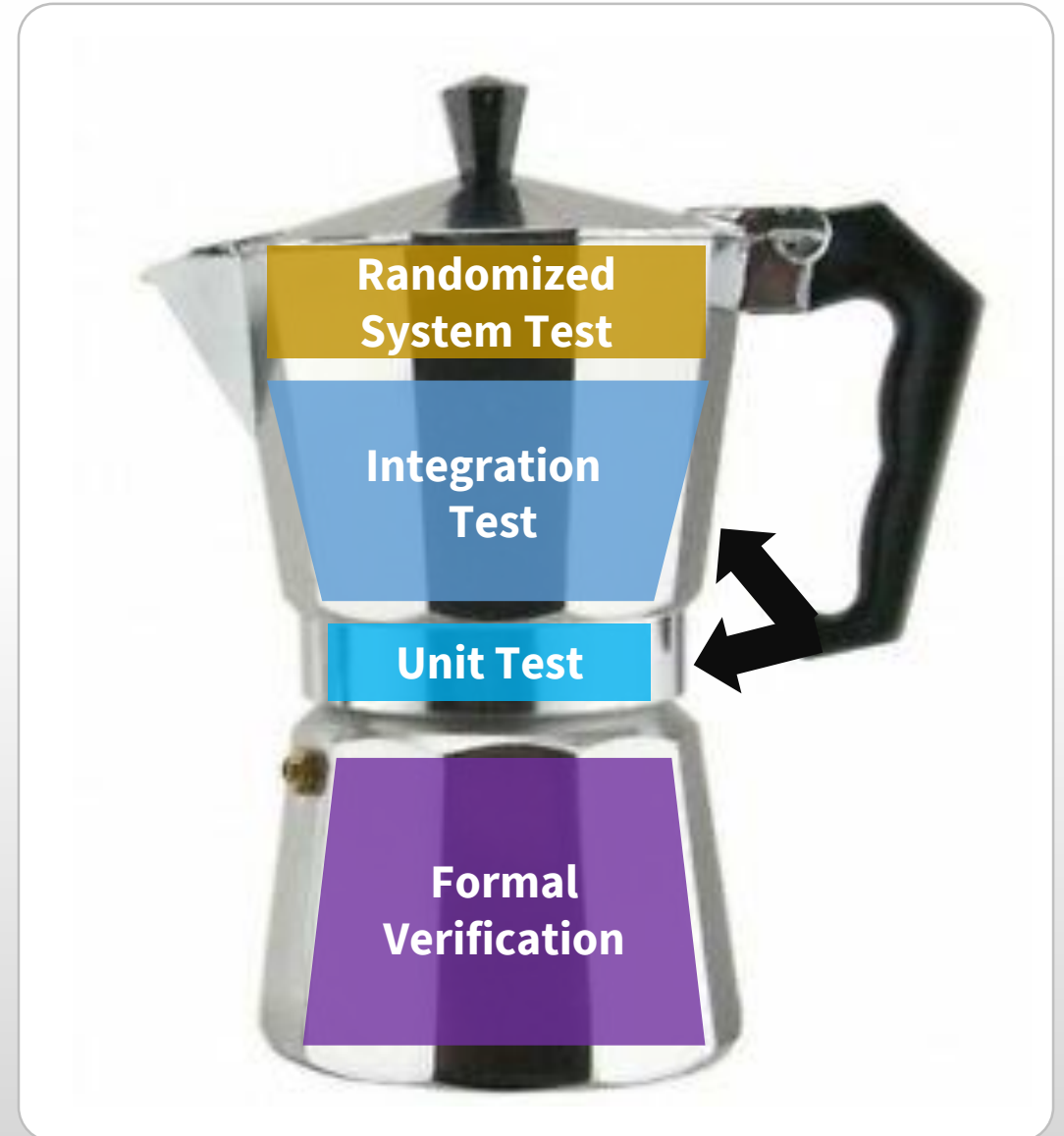
But that's ok

- Don't be put off by strange syntax
- You may already know different languages of different paradigms:
 - A procedural language
 - SQL (Set based)
 - HTML, CSS, YAML (Declarative)
 - A functional language
- You can be productive in 2-3 weeks

and... Testing Cafeterieres



A multi-layered test strategy for distributed systems.



Beyond unit and integration testing

Property Based Testing

Test that certain properties (invariants) hold while executing a program with different inputs.



Nicolas Rinaudo

@NicolasRinaudo

Follow



Studying TLA+ and property based tests, and the more I look into it, the more they look like two sides of the same coin.

5:33 AM - 10 Mar 2019

Randomized System Testing

Unleashing entropy, discovering the unexpected

Check that certain properties (invariants) hold while:

- executing randomly chosen actions
- at random intervals (including concurrently)

Start a node, stop a node, send a message, read a message, commit an offset, rewind an offset, add a partition, start a consumer, stop a consumer, add a second consumer group ...

Randomized System Testing

Unleashing entropy, discovering the unexpected

We can add more randomness..

- randomly injecting failure
- using random combinations of configurations and features

Randomized System Testing

Unleashing entropy, discovering the unexpected

Some failures:

- Kill TCP connections

- Add latency to TCP connections

- Add packet loss to TCP connections

- Kill nodes

- Destroy data on disk

- Corrupt data on disk

- Network Partitions

- Clock skew

Randomized System Testing

Unleashing entropy, discovering the unexpected

Apache Kafka configs and feature examples:

Replication Factor: 1, 3, 5?

Log Compaction: enabled/disabled?

Producer acks: 0, 1 or all?

Idempotent producer: Enabled, disabled?

Producer batch size? Max in-flight requests?

Consumer manual or auto-commit?

Min In-Sync Replicas?

Fsync period?

Transactions?

Randomized System Testing

Unleashing entropy, discovering the unexpected

... all while recording actions, events, crash dumps, metrics leading to test failures in order to:

- to understand root cause of failures
- and make test failures reproducible to create unit/integration test cases (regression)

Randomized System Testing

If you thought integration tests were slow...

Unit tests



Integration tests



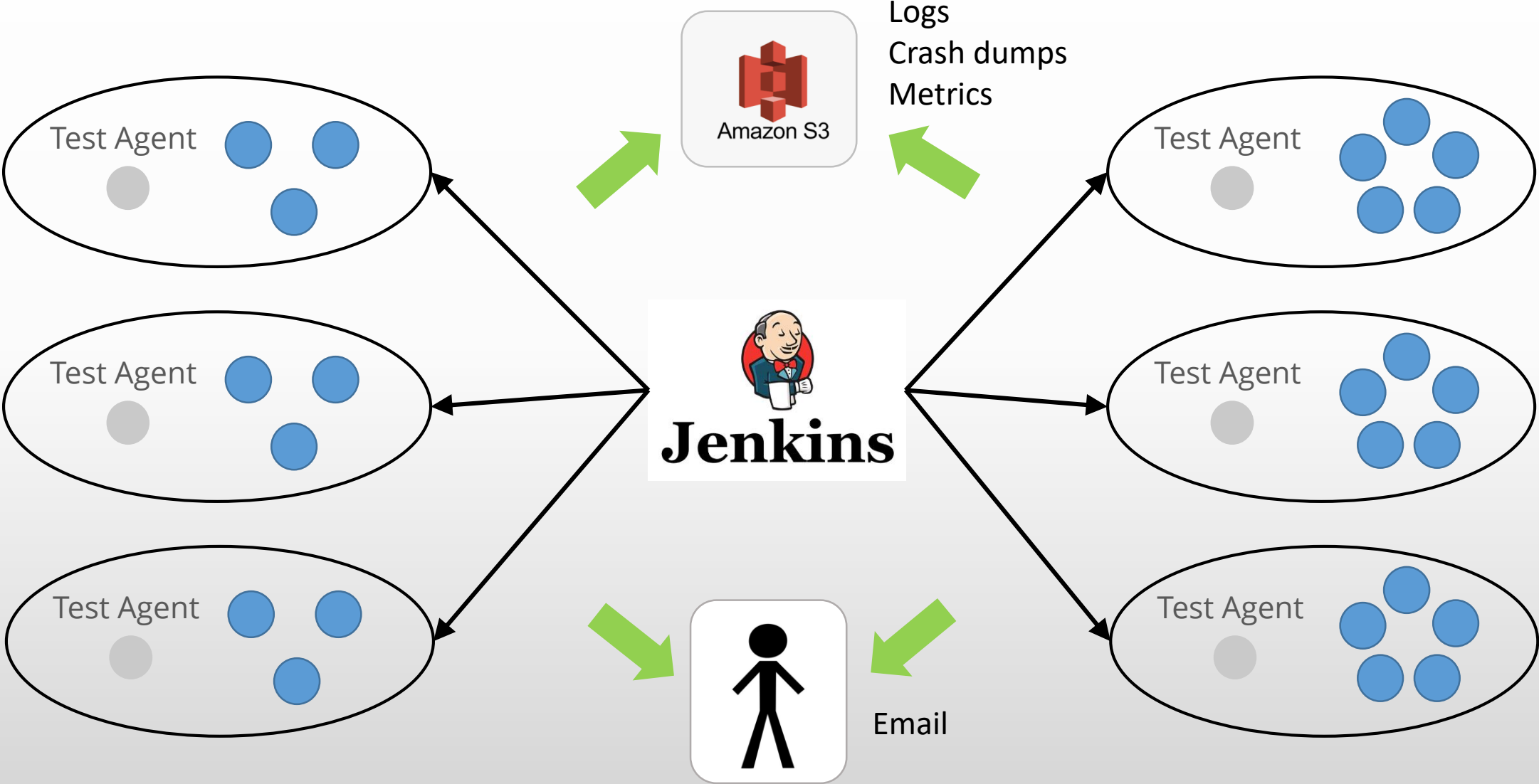
Randomized
System Tests



Not just slow to
execute,
but slow to analyze

Randomized System Testing

Unleashing entropy, discovering the unexpected

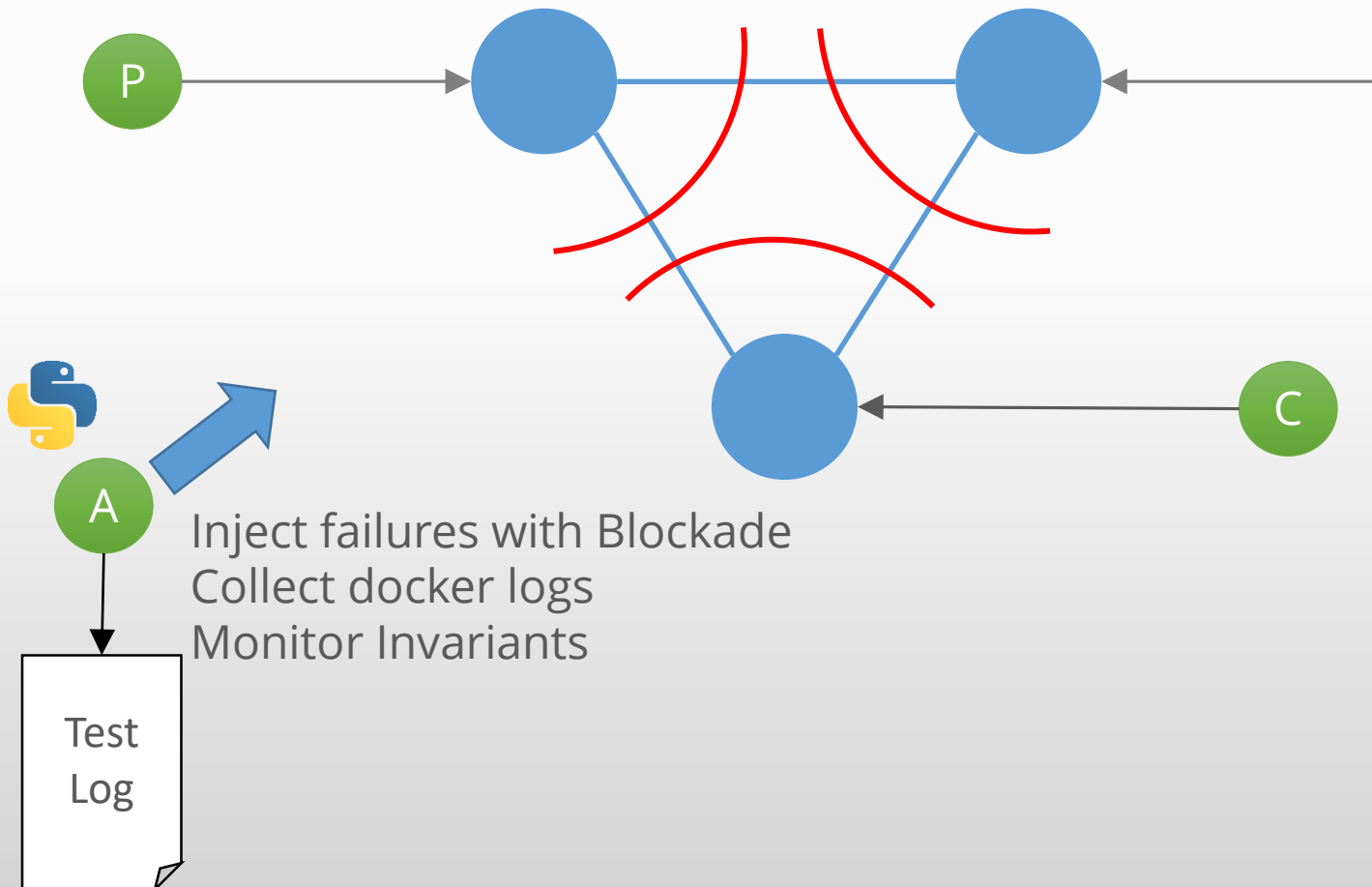


Case Study – RabbitMQ Quorum Queue Testing

Each test verifies both safety and liveness properties:

- One producer constantly trying to send messages with a monotonically increasing integer
- One consumer constantly trying to consume messages
- At random intervals, perform actions such as
 - Starting, stopping nodes
 - Stopping and starting the consumer
 - Killing and restarting the consumer
 - Injecting failures
- 5 minute grace period for stability to be established
- Check invariants:
 - Message loss
 - Message ordering (monotonically increasing numbers in messages)

Case Study – RabbitMQ Quorum Queue Testing



Actions

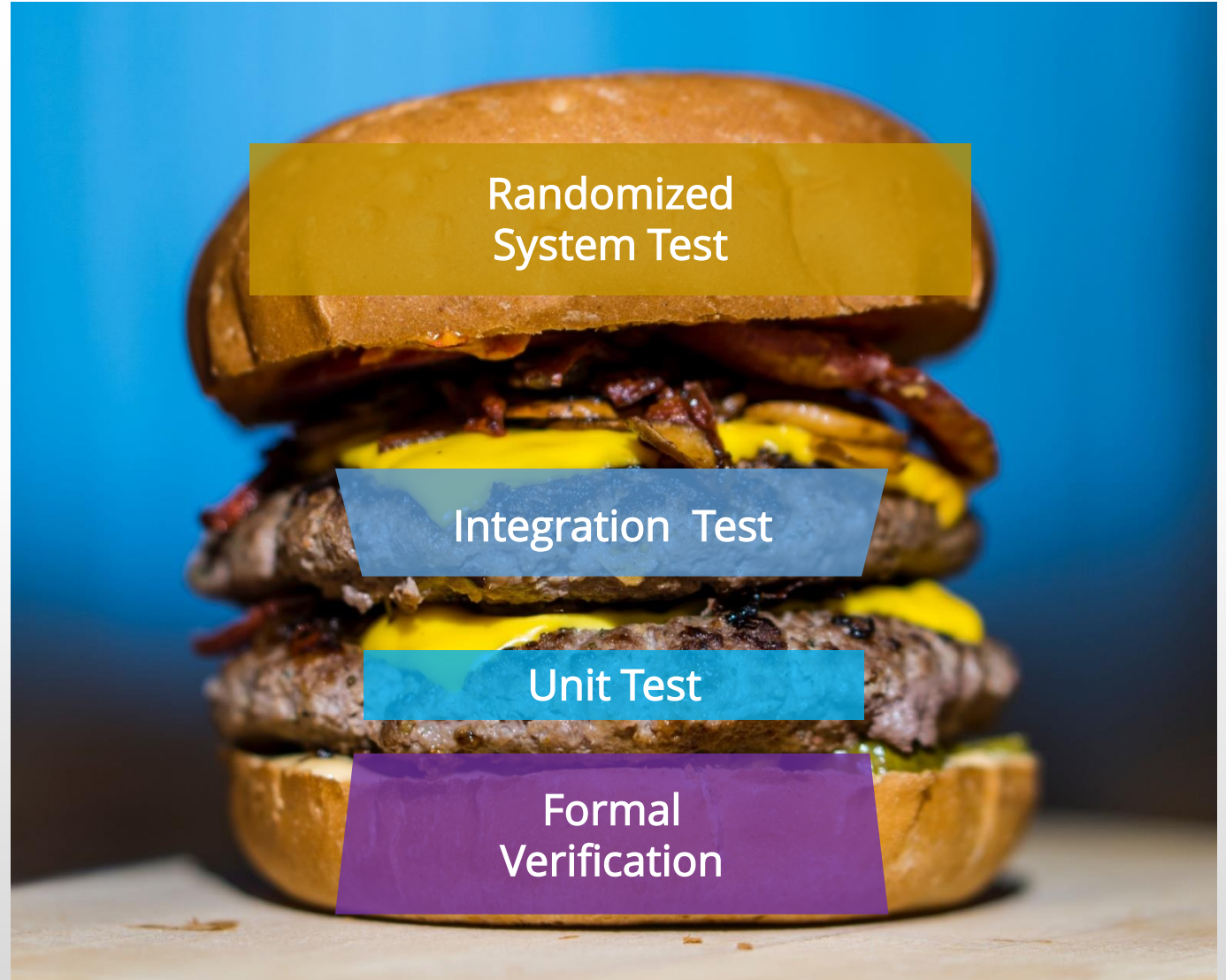
One publisher, one consumer constantly trying to publish and consume

1. Node 1 crashes
2. Node 3 network partitioned
3. Node 1 starts up and rejoins
4. Node 1-2 link congested
5. Network partition resolved
6. Node 2 catastrophic failure (including data)
7. Node 3 failure
8. Node 2 starts up and rejoins
9. Node 1 network partitioned
10. Node 3 starts up and rejoins
11. Partition resolved
12. Node 2 failure
13. Node 2 starts up and rejoins
14. Congestion on all links
15. Node 2 network partitioned
16. Partition resolved
17. Congestion alleviated

or... Testing Hamburgers

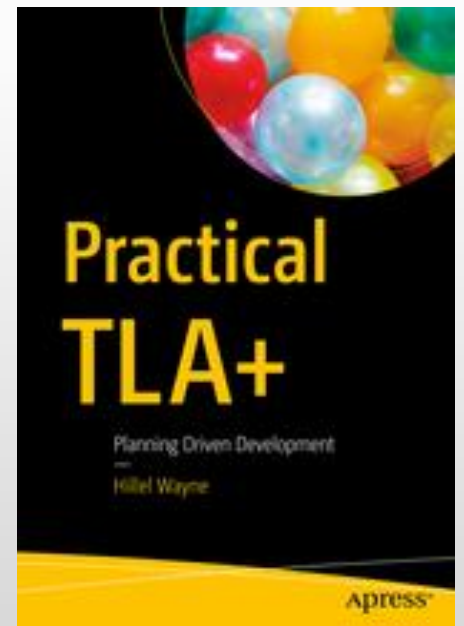
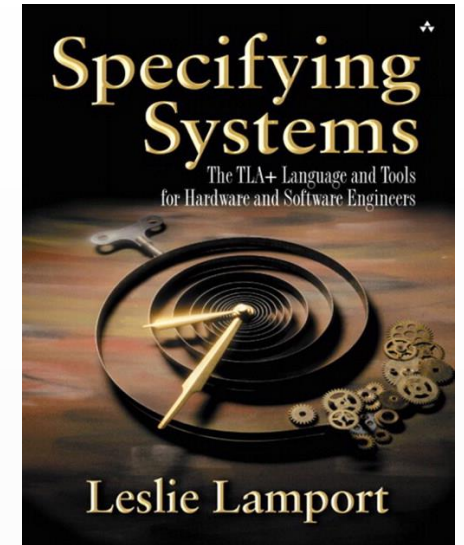
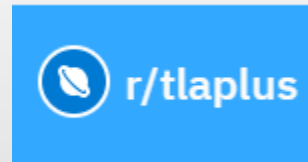


A multi-layered test strategy for distributed systems.



Learning TLA+

- Leslie Lamport's Video Series:
<https://lamport.azurewebsites.net/video/videos.html>
- Specifying Systems E-Book
<https://lamport.azurewebsites.net/tla/book.html>
- Hillel Wayne, Practical TLA+
<https://www.apress.com/gp/book/9781484238288>
- TLA+ subreddit
<https://www.reddit.com/r/tlaplus/>
- TLA+ Google Group
<https://groups.google.com/forum/#!forum/tlaplus>



Random Testing and Fault Injection Tools

Jepsen

<https://github.com/jepsen-io/jepsen>
<https://jepsen.io/analyses>

ToxiProxy

<https://github.com/Shopify/toxiproxy>

Blockade

<https://github.com/worstcase/blockade>

Chaos Monkey

<https://github.com/netflix/chaosmonkey>

RabbitMQ 3.8 Randomized Test Code:

<https://github.com/Vanlightly/ChaosTestingCode/blob/master/RabbitMqUdn/readme.md>

Andrey Satarin - Testing Distributed Systems

<https://asatarin.github.io/testing-distributed-systems/>

