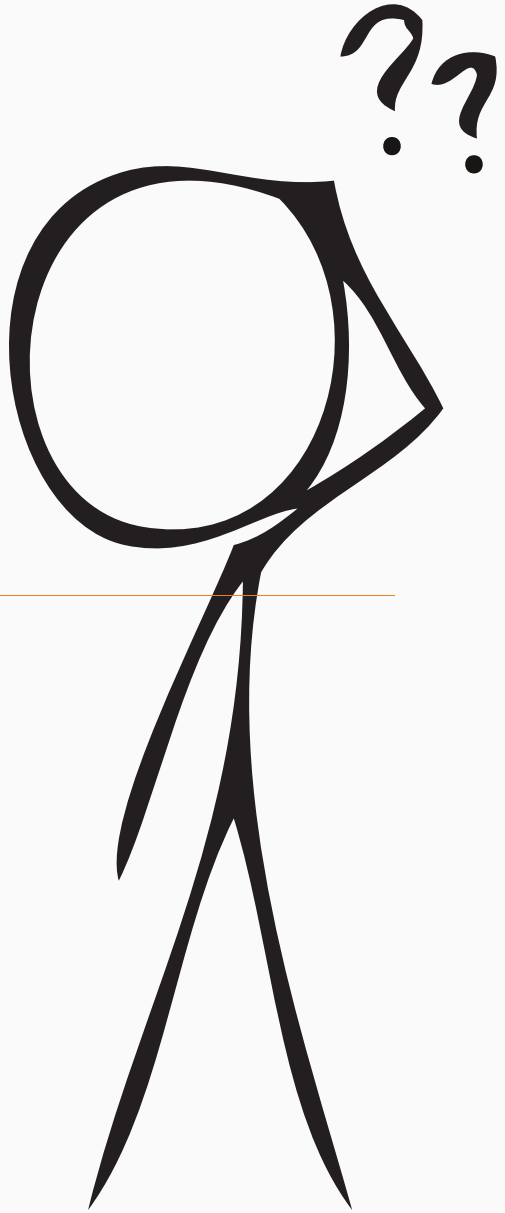
VERIFICATION OF CONCURRENT AND DISTRIBUTED SYSTEMS



Nickolai Novik

February 18, 2018

http://github.com/jettify

ABOUT ME

- **[Software Engineer]** DataRobot
- **[Github]** http://github.com/jettify
- **[Twitter]** https://twitter.com/isinf
- **[aio-libs]** https://github.com/aio-libs
- **[Projects]** *aiomonitor*, *aiohttp-debugtoolbar*, *aiobotocore*, *aiomysql*, *aioodbc*, *aiohttp-admin*, *aiorwlock*, *aiozipkin*, *etc*

AGENDA

1. Problem Statement. Motivational Example
2. Model Checking
3. TLA+ Basics
4. Aiorwlock Spec
5. Multithreading Queue Spec
6. Conclusions

POLL

How many of you heard of formal methods?

- I used one on of: Coq, Isable, TLA+, Alloy, Spin.
- I heard about it, but never used.
- I think formal methods are kinda cool.

PROBLEM STATEMENT. MOTIVATIONAL EXAMPLE

BRAVE NEW WORLD OF (MICROSERVISES) DISTRIBUTED SYSTEMS

How to be confidant that critical software works correctly?

1. Processor speed saturated, parallel execution is an answer
2. Concurrent/Parallel program is often requirement
3. Program complexity only raising
4. Simplified (monolith) application out of fashion
5. Due to micro services, everyone should be distributed systems expert

QA APPROACHES

Most of industry uses following techniques for quality assurance:

1. Design review
2. Static code analysis
3. Unit/Integration/Functional testing
4. Code coverage
5. Code review
6. Stress testing
7. Fault-injection testing

DISTRIBUTED/PARALLEL ALGORITHMS EXTREM HARD

Chord

is popular algorithm for P2P systems, paper published in 2001 by strong team of MIT researchers, 10 years later bug found in specification [10, 11]. Paper won best paper award.

Snark

non-blocking deque algorithm, published by well known researchers from Sun, clearly written sketch proof of the correctness of the algorithm. Later significant issue was found in algorithm [2, 6].

SOMETIMES COST OF ERROR IS VERY HIGH

Mars Pathfinder

rover, the mission was jeopardised by a concurrent software bug in the lander. [9]

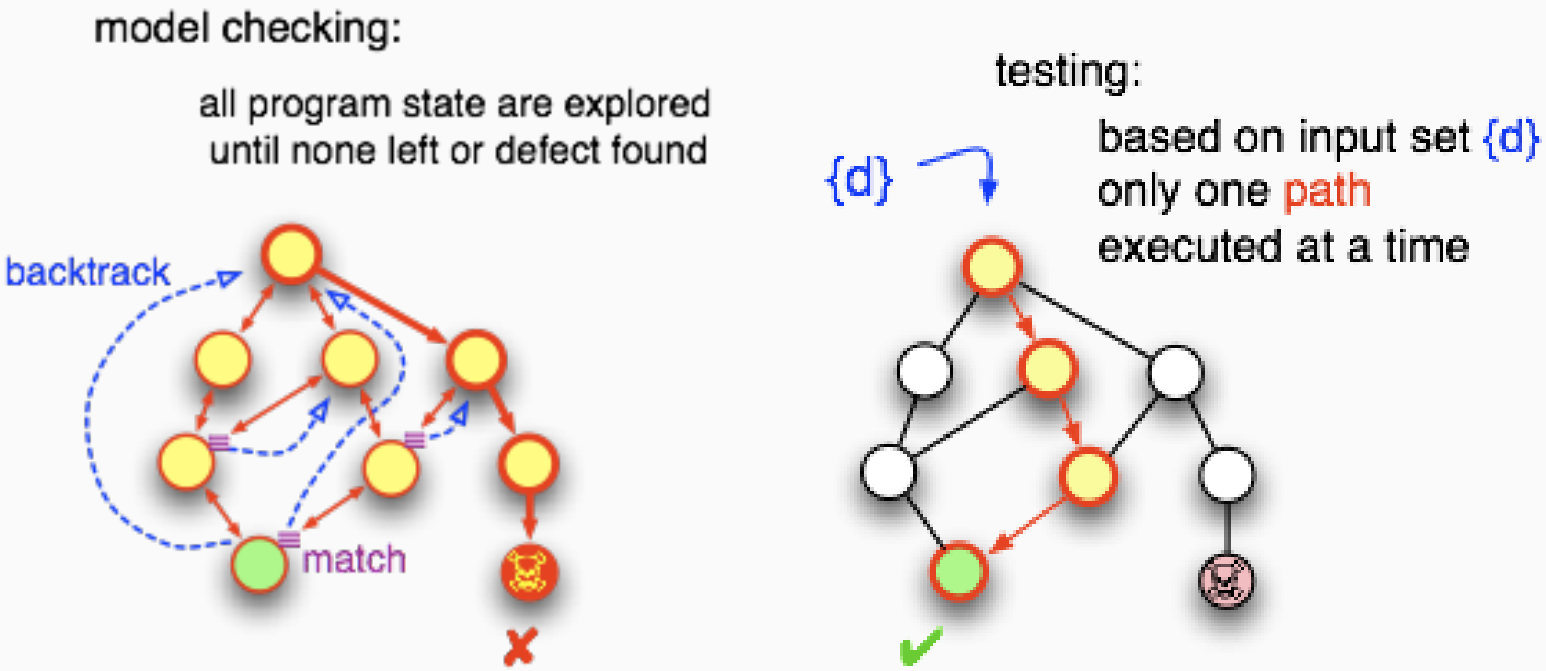
Therac-25

radiation therapy machine, because of concurrent programming errors, it sometimes gave its patients radiation doses that were hundreds of times greater than normal [1]

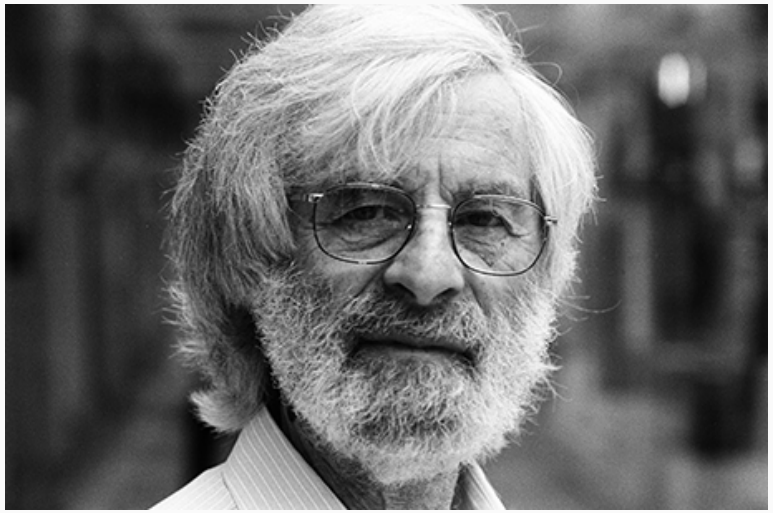
MODEL CHECKING

SLIDE NAME

Model checking is a technique for automatically verifying correctness properties of finite-state systems.



TLA+ – TEMPORAL LOGIC OF ACTIONS



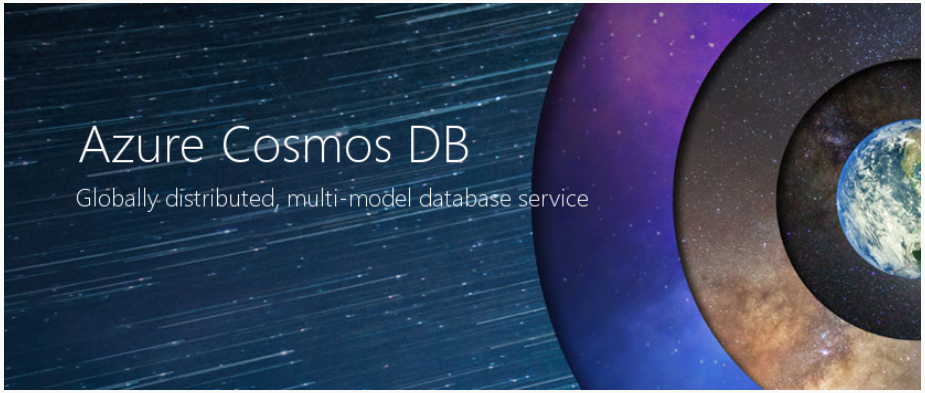
TLA+ language developed by **Leslie Lamport**. It is used to design, model, document, and verify concurrent systems, has been described as exhaustively-testable pseudocode and blueprints for software systems.

TLA+ INDUSTRY USAGE: AWS

TLA+ helped to find design bugs in **S3, Dynamo, EBS, EC2**, etc, some requiring traces of 35 steps. [8]

TLA+ INDUSTRY USAGE: MICROSOFT

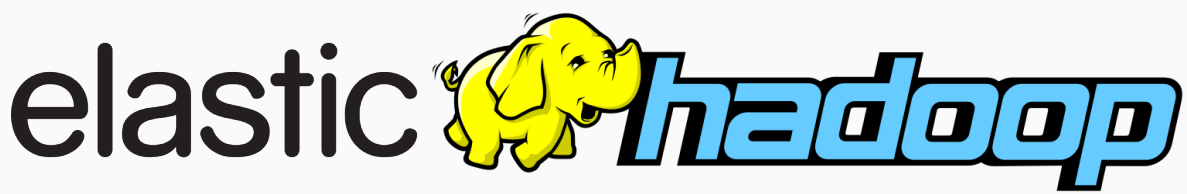
MS used TLA+ to define consistency protocol for CosmosDB and memory allocator for XBox [7]



TLA+ INDUSTRY USAGE: OPENS SOURCE

Number open source projects use TLA+ to verify complex algorithms:

- Elastic – data replication protocol [3]
- Mongodb – data replication protocol [12]
- Hadoop/YARN – registry of long lived processes [4]



TLA+ BASICS

TLA+ HELLO WORLDS

```
1  ---------------------- MODULE HourClock ----------------------
2  EXTENDS Naturals
3  VARIABLE hr
4  HCini  ==  hr \in (1 .. 12)
5  HCnxt  ==  hr' = IF hr # 12 THEN hr + 1 ELSE 1
6  HC  ==  HCini /\ [][HCnxt]_hr
7  ------------------------------------------------------------
8  THEOREM  HC => []HCini
9  ============================================================
```

```
1  ──────────────────── MODULE HourClock ────────────────────
2  EXTENDS Naturals
3  VARIABLE hr
4  HCini  ≜  hr ∈ (1 .. 12)
5  HCnxt  ≜  hr′ = IF hr ≠ 12 THEN hr + 1 ELSE 1
6  HC  ≜  HCini ∧ □[HCnxt]_hr
7  ├──────────────────────────────────────────────────────────┤
8  THEOREM  HC ⇒ □HCini
9  └──────────────────────────────────────────────────────────┘
```

TLA+ SYNTAX. LOGIC

Basic logical operators:

- $\vee$ logical OR, *or* in python
- $\wedge$ logical AND, *and* and in python
- $\neg$ logical NOT, *and* and in python
- $=$ boolean operator, checks equality, it is not an assignment operator.
- $\triangleq$ boolean operator, checks equality, it is not an assignment operator.

TLA+ SYNTAX. MORE LOGIC

More logic operators:

- $\exists$ means "there exists", written as E in ASCII
- $\forall$ means "for all", written as A in ASCII
- : colon reads as "such that"

$\exists x \in 1, 2, 3, 4, 5 : x > 3$ - exists x in set of integers $1, 2, 3, 4, 5$ such that $x > 3$ expression evaluates to TRUE.

TLA+ SYNTAX. MORE LOGIC

More logic operators:

$\Box$ formula is TRUE on each step

$\Diamond$ eventually TRUE

$\Longrightarrow$ logical implication

$'$ reads as prime, state of variable on next step

$\exists x \in 1, 2, 3, 4, 5 : x > 3$ – exists x in set of integers $1, 2, 3, 4, 5$ such that $x > 3$ expression evaluates to TRUE.

TLA+ SPEC TEMPLATE

```
1  ---------------------- MODULE ModuleName ----------------------
2  (* Imports and variable declarations*)
3  EXTENDS Naturals, Sequences, Integers, FiniteSets
4  ------------------------------------------------------------------
5  (* Initial Conditions *)
6  Init == ...
7  TypeOK == ...
8  ------------------------------------------------------------------
9  (* Body of the spec *)
10 Next == ...
11 Invariant == ...
12 ------------------------------------------------------------------
13 (* Invariant declaration with temporal formula *)
14 THEOREM  Spec => ...
15 ==================================================================
```

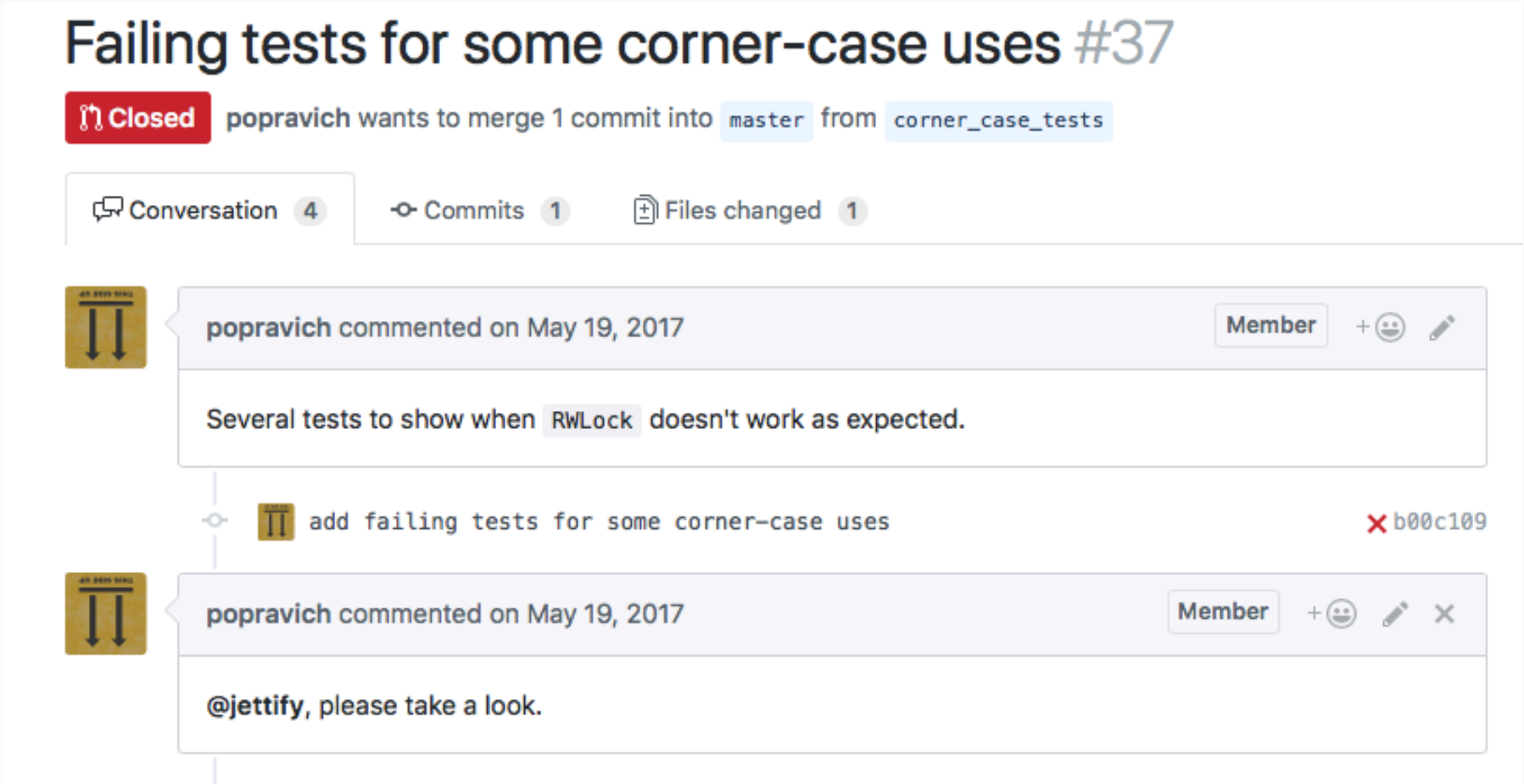
AIORWLOCK SPEC

AIORWLOCK – READ WRITE LOCK FOR ASYNCIO

An RW lock allows concurrent access for read-only operations, while write operations require exclusive access, simple example:

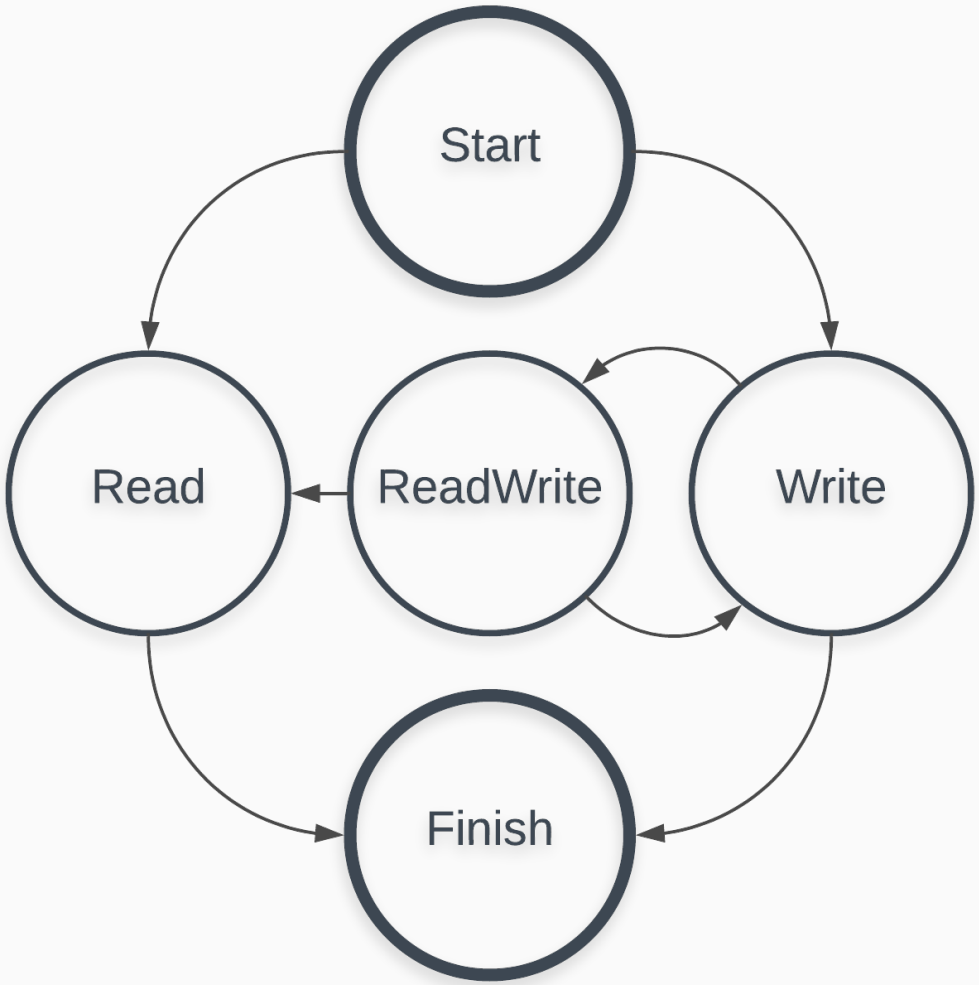
```
1  import asyncio
2  import aiorwlock
3
4  async def go():
5      rwlock = aiorwlock.RWLock()
6
7      async with rwlock.writer:
8          print("inside writer: only one writer is possible")
9
10     async with rwlock.reader:
11         print("inside reader: multiple reader possible")
12
13 loop = asyncio.get_event_loop()
14 loop.run_until_complete(go())
```

AIORWLOCK – BUG



AIORWLOCK POSSIBLE STATES

Read Write lock state machine:



AIORWLOCK IMPLEMENTATION SKETCH

Full implementation available in [5]

```
class _RWLockCore:
    def __init__(self, fast, loop):
        self._state = 0  # positive is shared count, negative exclusive count
        self._owning = []  # tasks will be few, so a list is not inefficient

    @property
    def read_locked(self):
        return self._state > 0
    @property
    def write_locked(self):
        return self._state < 0

    async def acquire_read(self):
        if not self._write_waiters and self._state >= 0:
            self._state += 1
            self._owning.append(me)
            return True
        # ...

```

AIORWLOCK TLA SPEC: PART 1

```
1   ---------------------------- MODULE aiorwlock ----------------------------
2   EXTENDS Naturals, Sequences, Integers, FiniteSets
3   CONSTANTS Task
4   ASSUME /\ Task # {}
5
6   VARIABLES State,
7             Lock
8
9   ---------------------------------------------------------------------------
10  TypeOK == /\ Lock \in [Task ->
11                         {"Read", "Write", "WriteRead", "Waiting", "Finished"}]
12            /\ State >= -1
13  LockInit == Lock = [t \in Task |-> "Waiting"] /\ State = 0
14  ---------------------------------------------------------------------------
15  RLocked == State > 0
16  WLocked == State < 0
17  Unlocked == State = 0
18  OwnWrite(t) == Lock[t] \in {"Write"}
19
```

AIORWLOCK TLA SPEC: PART 2

```
20  RAquire(t) == \/ /\  ~WLocked
21                   /\ Lock' = [Lock EXCEPT ![t] = "Read"]
22                   /\ State' = State + 1
23                   /\ Lock[t] \in {"Waiting"}
24                \/ /\ OwnWrite(t)
25                   /\ Lock' = [Lock EXCEPT ![t] = "WriteRead"]
26                   /\ State' = State + 1
27
28  WAquire(t) == /\ Unlocked
29                /\ Lock' = [Lock EXCEPT ![t] = "Write"]
30                /\ State' = State - 1
31                /\ Lock[t] \in {"Waiting"}
```

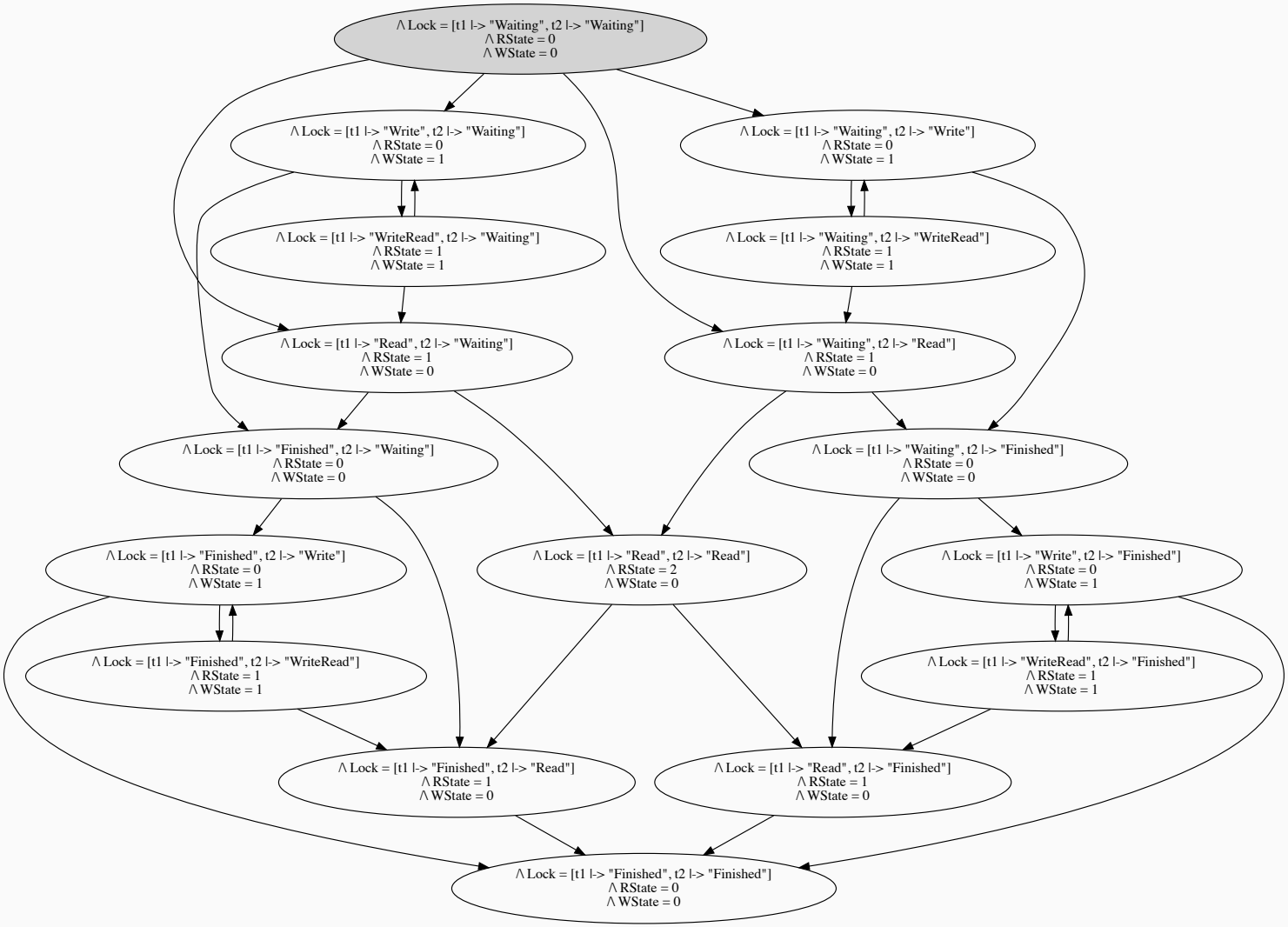
AIORWLOCK TLA SPEC: PART 3

```
34  RRelease(t) == \/ /\ RLocked /\ Lock[t] = "Read"
35                    /\ State' = State - 1
36                    /\ Lock' = [Lock EXCEPT ![t] = "Finished"]
37                 \/ /\ RLocked /\ Lock[t] = "WriteRead"
38                    /\ State' = State - 1
39                    /\ Lock' = [Lock EXCEPT ![t] = "Write"]
40
41  WRelease(t) == \/ /\ WLocked /\ Lock[t] = "Write"
42                    /\ State' = State - 1
43                    /\ Lock' = [Lock EXCEPT ![t] = "Finished"]
44                 \/ /\ WLocked /\ Lock[t] = "WriteRead"
45                    /\ State' = State - 1
46                    /\ Lock' = [Lock EXCEPT ![t] = "Read"]
47  ----------------------------------------------------------------------
```

AIORWLOCK TLA SPEC: PART 4

```
47  ----------------------------------------------------------------------------
48  Next == \E t \in Task: RAquire(t) \/ WAquire(t) \/ RRelease(t) \/ WRelease(t)
49  Spec == LockInit /\ [][Next]_<<State, Lock>>
50  LockInv ==
51      \A t1 \in Task : \A t2 \in (Task \ {t1}): ~
52          (/\ Lock[t1] \in {"Write", "WriteRead"}
53           /\ Lock[t2] \in {"Read", "Write", "WriteRead"})
54  ----------------------------------------------------------------------------
55  THEOREM Spec => [](TypeOK /\ LockInv)
56  ============================================================================
```

AIORWLOCK POSSIBLE STATES



AIORWLOCK TRACE

TLC shows all steps that leads to *invariant violation*.

Name	Value
▼ <Initial predicate>	State (num = 1)
▶ Lock	[t1 |-> "Waiting", t2 |-> "Waiting", t3 |-> "Waiting"]
State	0
▼ <Action line 30, col 15 to line 33, col 40 of mod...	State (num = 2)
▼ Lock	[t1 |-> "Write", t2 |-> "Waiting", t3 |-> "Waiting"]
t1	"Write"
t2	"Waiting"
t3	"Waiting"
State	-1
▼ <Action line 26, col 18 to line 28, col 38 of mod...	State (num = 3)
▼ Lock	[t1 |-> "WriteRead", t2 |-> "Waiting", t3 |-> "Waiting"]
t1	"WriteRead"
t2	"Waiting"
t3	"Waiting"
State	0
▼ <Action line 22, col 18 to line 25, col 43 of mod...	State (num = 4)
▼ Lock	[t1 |-> "WriteRead", t2 |-> "Read", t3 |-> "Waiting"]
t1	"WriteRead"
t2	"Read"
t3	"Waiting"
State	1

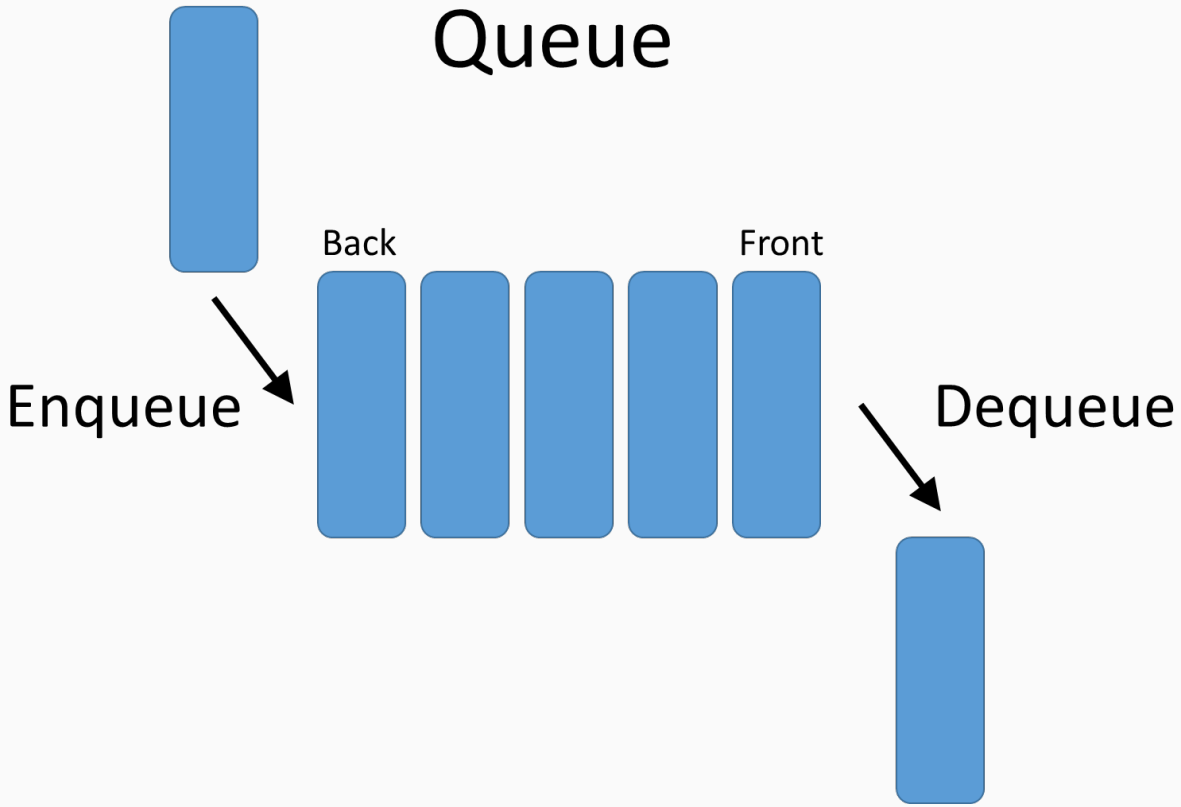
AIORWLOCK SPEC RESULTS

- In fact bug reveled itself on specification phase, before running any models
- Only three steps required to reproduce issue

MULTITHREADING QUEUE SPEC

MULTITHREADING QUEUE

Classic *bounded buffer*, attempts to put an element into a full queue or take from empty will block.



QUEUE IMPLEMENTATION, PART 1

```
1   import threading
2
3   class BoundedQueue:
4       def __init__(self, capacity=3):
5           self.capacity = capacity
6           self.buffer = [None] * capacity
7           self.mutex = threading.Lock()
8           self.condition = threading.Condition(self.mutex)
9           self.size = 0
10          self.head = 0
11          self.tail = 0
12
13      def is_full(self):
14          return self.size == self.capacity
15
16      def is_empty(self):
17          return self.size == 0
18
19      def _next(self, x):
20          return (x + 1) % self.capacity
```

QUEUE IMPLEMENTATION, PART 2

```
22      def put(self, item):
23          with self.condition:
24              while self.is_full():
25                  self.condition.wait()
26              self.buffer[self.tail] = item
27              self.tail = self._next(self.tail)
28              self.size += 1
29              self.condition.notify()
30
31      def get(self):
32          with self.condition:
33              while self.is_empty():
34                  self.condition.wait()
35              item = self.buffer[self.head]
36              self.buffer[self.head] = None
37              self.head = self._next(self.head)
38              self.size -= 1
39              self.condition.notify()
40              return item
```

QUEUE TLA SPEC: PART 1

```
1   ------------------------------ MODULE buffer ------------------------------
2   EXTENDS Naturals, Sequences
3
4   CONSTANTS Producers,
5             Consumers,
6             BufCapacity,
7             Data
8
9   ASSUME /\ Producers # {}
10         /\ Consumers # {}
11         /\ Producers \intersect Consumers = {}
12         /\ BufCapacity > 0
13         /\ Data # {}
14  VARIABLES buffer,
15            waitSet
16  ---------------------------------------------------------------------------
```

QUEUE TLA SPEC: PART 2

```
16  ----------------------------------------------------------------------------
17  Participants == Producers \union Consumers
18  RunningThreads == Participants \ waitSet
19
20  TypeInv == /\ buffer \in Seq(Data)
21             /\ Len(buffer) \in 0..BufCapacity
22             /\ waitSet \subseteq Participants
23
24  Notify == IF waitSet # {}
25            THEN \E x \in waitSet : waitSet' = waitSet \ {x}
26            ELSE UNCHANGED waitSet
27
28  NotifyAll == waitSet' = {}
29
30  Wait(t) == waitSet' = waitSet \union {t}
31  ----------------------------------------------------------------------------
```

QUEUE TLA SPEC: PART 3

```
32  Init == buffer = <<>> /\ waitSet = {}
33  Put(t,m) == IF Len(buffer) < BufCapacity
34              THEN /\ buffer' = Append(buffer, m)
35                   /\ Notify
36              ELSE /\ Wait(t)
37                   /\ UNCHANGED buffer
38  Get(t) == IF Len(buffer) > 0
39            THEN /\ buffer' = Tail(buffer)
40                 /\ Notify
41            ELSE /\ Wait(t)
42                 /\ UNCHANGED buffer
43  Next == \E t \in RunningThreads : \/ t \in Producers /\ \E m \in Data : Put(t,m
44                                    \/ t \in Consumers /\ Get(t)
45
46  Prog == Init /\ [][Next]_<<buffer, waitSet>>
47  ----------------------------------------------------------------------------
48  NoDeadlock == [](RunningThreads # {})
49
50  THEOREM Prog => []TypeInv /\ NoDeadlock
```

MULTITHREADING QUEUE SPEC RESULTS

- In fact bug reveled itself on specification phase, before running any models
- More than **40 steps** required to reproduce deadlock!
- Thread programming is hard!

QUEUE TRACE

TLC shows all steps that leads to *invariant violation*.

Name	Value
▼ <Initial predicate>	State (num = 1)
▶ Lock	[t1 |-> "Waiting", t2 |-> "Waiting", t3 |-> "Waiting"]
State	0
▼ <Action line 30, col 15 to line 33, col 40 of mod...	State (num = 2)
▼ Lock	[t1 |-> "Write", t2 |-> "Waiting", t3 |-> "Waiting"]
t1	"Write"
t2	"Waiting"
t3	"Waiting"
State	-1
▼ <Action line 26, col 18 to line 28, col 38 of mod...	State (num = 3)
▼ Lock	[t1 |-> "WriteRead", t2 |-> "Waiting", t3 |-> "Waiting"]
t1	"WriteRead"
t2	"Waiting"
t3	"Waiting"
State	0
▼ <Action line 22, col 18 to line 25, col 43 of mod...	State (num = 4)
▼ Lock	[t1 |-> "WriteRead", t2 |-> "Read", t3 |-> "Waiting"]
t1	"WriteRead"
t2	"Read"
t3	"Waiting"
State	1

CONCLUSIONS

LIMITATIONS OF MODEL-CHECKING

- State space explosion number of states reachable by a system can quickly become huge, or even infinite
- Used as an adjunct to, not a replacement for, standard quality assurance methods
- Formal methods are not a panacea, but can increase confidence in a product's reliability if applied with care and skill
- Very useful for consistency checks, but can not assure completeness

Questions?

http://github.com/jettify

REFERENCES i

W. contributors.
Therac-25 — wikipedia, the free encyclopedia, 2018.
https://en.wikipedia.org/w/index.php?title=Therac-25&oldid=820552180.

S. Doherty, D. L. Detlefs, L. Groves, C. H. Flood, V. Luchangco, P. A. Martin, M. Moir, N. Shavit, and G. L. Steele Jr.
Dcas is not a silver bullet for nonblocking algorithm design.
In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 216–224. ACM, 2004.

Elasticsearch.
Formal models of core elasticsearch algorithms, 2017.
https://github.com/elastic/elasticsearch-formal-models.

A. Hadoop.
Introduce coordination engine interface, 2017.
https://issues.apache.org/jira/browse/HADOOP-10641.

jettify.
aiorwlock implementation with bug, 2017.
https://github.com/aio-libs/aiorwlock/blob/v0.4.0/aiorwlock/__init__.py.

L. Lamport.
Checking a multithreaded algorithm with+ cal.
In *DISC*, volume 4167, pages 151–163. Springer, 2006.

REFERENCES ii

Microsoft.
A technical overview of azure cosmos db, 2017.
https://azure.microsoft.com/en-us/blog/a-technical-overview-of-azure-cosmos-db/.

C. Newcombe.
Why amazon chose tla+.
In Y. Ait Ameur and K.-D. Schewe, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 25–39, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

Rapitasystems.
What really happened to the software on the mars pathfinder spacecraft?, 2013.
https://www.rapitasystems.com/blog/what-really-happened-to-the-software-on-the-mars-pathfinder-spacecraft.

I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan.
Chord: A scalable peer to peer lookup service for internet applications.
ACM SIGCOMM Computer Communication Review, 31(4):149–160, 2001.

P. Zave.
How to make chord correct (using a stable base).
CoRR, abs/1502.06461, 2015.

S. Zhou.
Tla+ spec of a simplified part of mongodb replication system, 2017.
https://github.com/visualzhou/mongo-repl-tla.