# Java PathFinder: going to Mars without bugs and deadlocks

Alexander Filatov

# Bio

- 2015 - 2019 Software engineer, **Excelsior**
  Performance tuning of proprietary JVM
  - Scalability analysis of automatic memory management
  - Design and analysis of concurrent data structures
- 2019 - `Present` Principal Engineer, **Huawei**
  - Development of managed runtimes
  - Design and implementation of memory manager improvements

# My motivation for this talk

What I say in resume

What I actually mean

# My motivation for this talk

| What I say in resume | What I actually mean |
| --- | --- |
| Researcher of automatic memory management | I love to write prototypes |

# My motivation for this talk

| What I say in resume | What I actually mean |
| --- | --- |
| Researcher of automatic memory management | I love to write prototypes |
| Engineer of scalable concurrent data structures | I like to implement complicated algorithms |

# My motivation for this talk

| What I say in resume | What I actually mean |
| --- | --- |
| Researcher of automatic memory management | I love to write prototypes |
| Engineer of scalable concurrent data structures | I like to implement complicated algorithms |
| Interested in reliable multithreaded software | I hate debugging concurrency |

# My motivation for this talk

| What I say in resume | What I actually mean |
| --- | --- |
| Researcher of automatic memory management | I love to write prototypes |
| Engineer of scalable concurrent data structures | I like to implement complicated algorithms |
| Interested in reliable multithreaded software | I hate debugging concurrency |

# Scope

**Concurrency** vs **parallelism**

# Scope

**Concurrency** vs ~~**parallelism**~~

# Scope

**Concurrency**

# Scope

**Concurrency**

Concept

Problem

# Scope

**Concurrency**

| Concept | Problem |
| --- | --- |
| Threads + Mutable shared memory | Data races |

# Scope

**Concurrency**

| Concept | Problem |
| --- | --- |
| Threads + Mutable shared memory | Data races |
| Concurrent data structures: | Incorrect uses: |
| Mutex | Deadlock |
| Semaphore | Livelock |
| Monitor | Priority inversion |
| Latch | ABA problem |
| Barrier | Memory model constraints |

# Scope

**Concurrency**

| Concept | Problem |
| --- | --- |
| Threads + Mutable shared memory | Data races |
| | |
| Concurrent data structures: | Incorrect uses: |
| Mutex | Deadlock |
| Semaphore | Livelock |
| Monitor | Priority inversion |
| Latch | ABA problem |
| Barrier | Memory model constraints |

Aleksey Shipilëv: Close Encounters of The Java Memory Model Kind
`https://shipilev.net/blog/2016/close-encounters-of-jmm-kind`
`https://www.youtube.com/watch?v=C6b_dFtujKo`

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
   }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
   }
}
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0;
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0;
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0;
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0;
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0;
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

W1: x = 0, y = 0; W2: x = 1, y = 1;

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

W1: x = 0, y = 0; W2: x = 1, y = 1;

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

W1: x = 0, y = 0; W2: x = 1, y = 1;

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

W1: x = 0, y = 0; W2: x = 1, y = 1;

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

W1: x = 0, y = 0; W2: x = 1, y = 1;

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

W1: x = 0, y = 0; W2: x = 1, y = 1;

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

W1: x = 0, y = 0; W2: x = 1, y = 1;

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

W1: x = 0, y = 0; W2: x = 1, y = 1;

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0; W2: x = 1, y = 1;
W1: x = 0, y = 0;
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0; W2: x = 1, y = 1;
W1: x = 0, y = 0;
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0; W2: x = 1, y = 1;
W1: x = 0, y = 0;
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0; W2: x = 1, y = 1;
W1: x = 0, y = 0; W2: x = 0, y = 0;
```

# Execution trace

```
class W1 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W1", l_x, l_y);
  }
}
```

```
class W2 extends Thread {
  public void run() {
    int l_x = x;
    int l_y = y;
    x++;
    y++;
    log("W2", l_x, l_y);
  }
}
```

```
W1: x = 0, y = 0; W2: x = 0, y = 0;
W1: x = 0, y = 0; W2: x = 0, y = 1;
W1: x = 0, y = 0; W2: x = 1, y = 0;
W1: x = 0, y = 0; W2: x = 1, y = 1;
W1: x = 0, y = 1; W2: x = 0, y = 0;
W1: x = 1, y = 0; W2: x = 0, y = 0;
W1: x = 1, y = 1; W2: x = 0, y = 0;
```

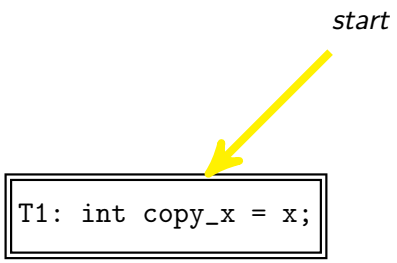
# State space exploration

*start*

```
    Thread 1
int copy_x = x;
x++;
```

```
    Thread 2
int copy_x = x;
x++;
```

# State space exploration



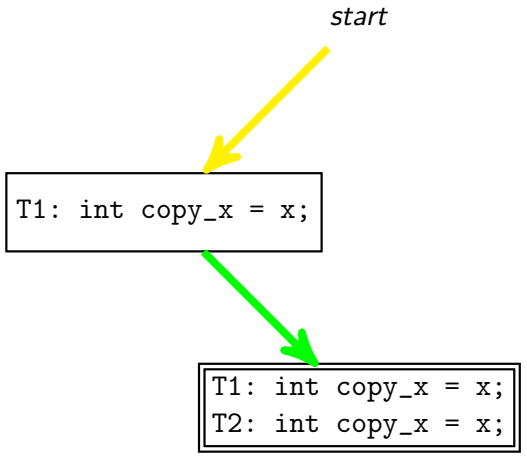

Thread 1

```
int copy_x = x;
x++;
```

Thread 2

```
int copy_x = x;
x++;
```

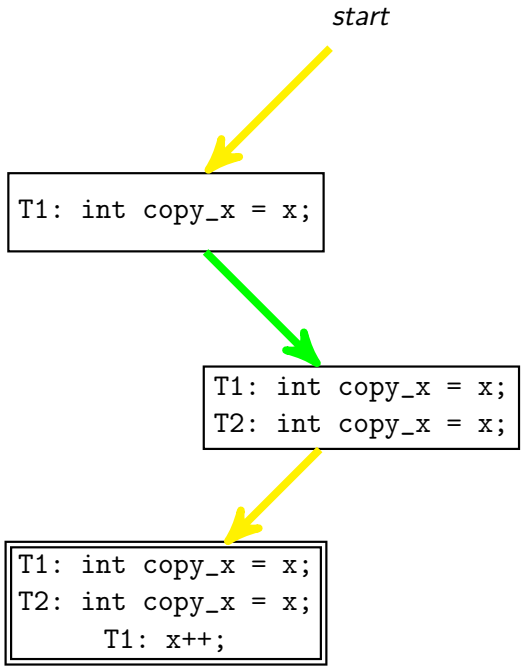
# State space exploration




```
    Thread 1
int copy_x = x;
x++;
```

```
    Thread 2
int copy_x = x;
x++;
```

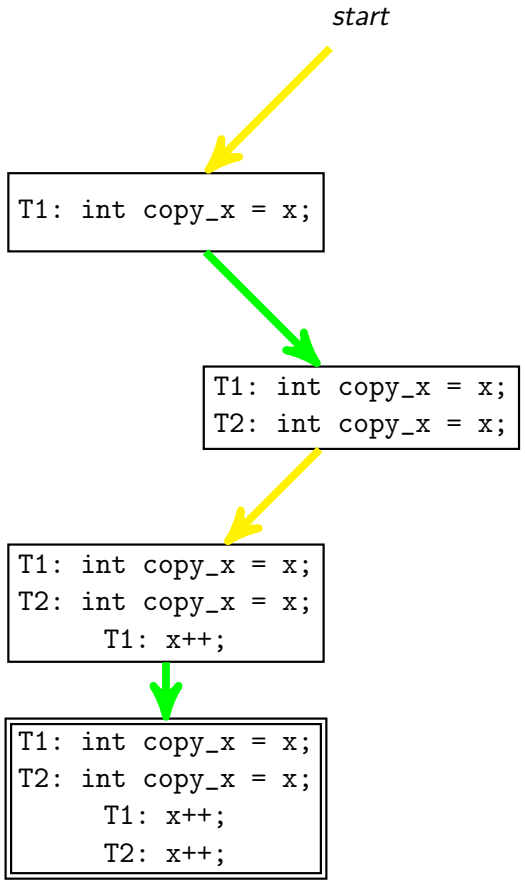
# State space exploration




```
    Thread 1
int copy_x = x;
x++;
```

```
    Thread 2
int copy_x = x;
x++;
```

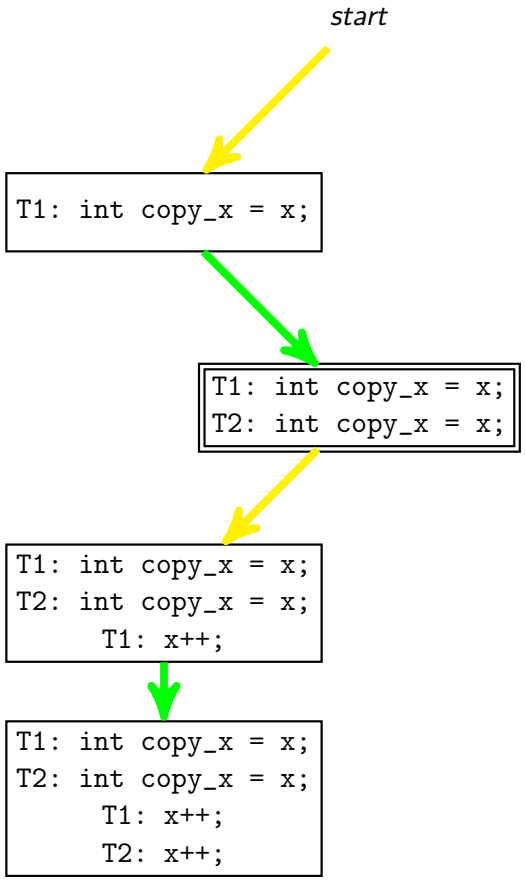
# State space exploration




```
    Thread 1
int copy_x = x;
x++;
```

```
    Thread 2
int copy_x = x;
x++;
```

# State space exploration



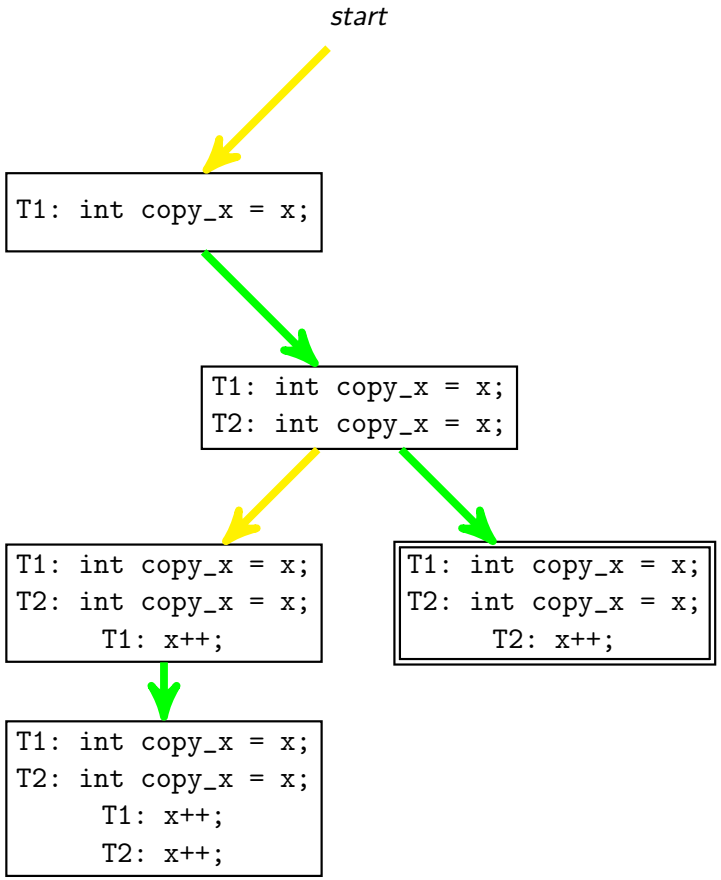

Thread 1

```
int copy_x = x;
x++;
```

Thread 2

```
int copy_x = x;
x++;
```

# State space exploration



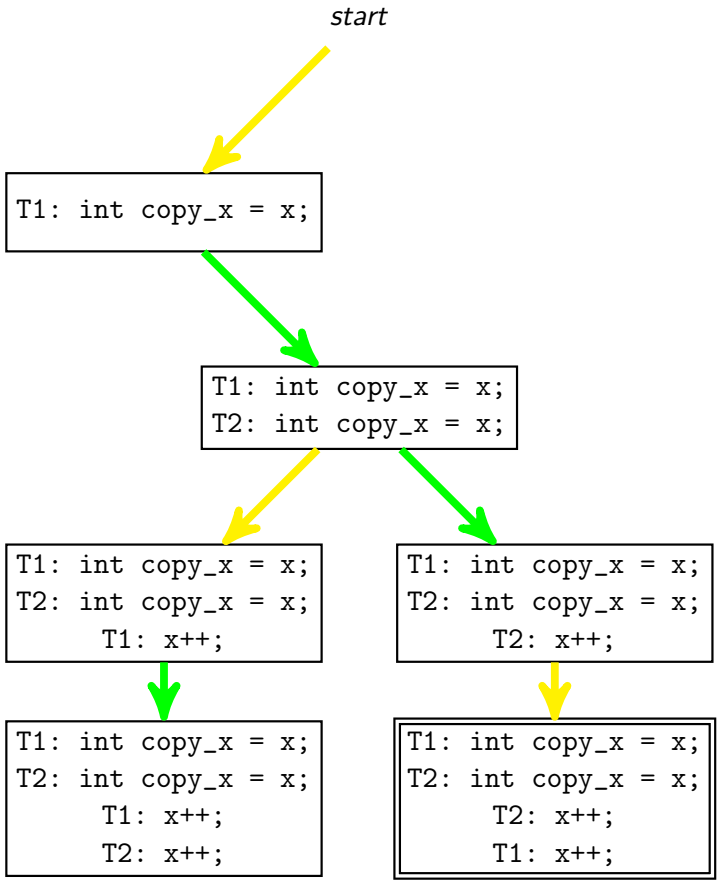

Thread 1

```
int copy_x = x;
x++;
```

Thread 2

```
int copy_x = x;
x++;
```

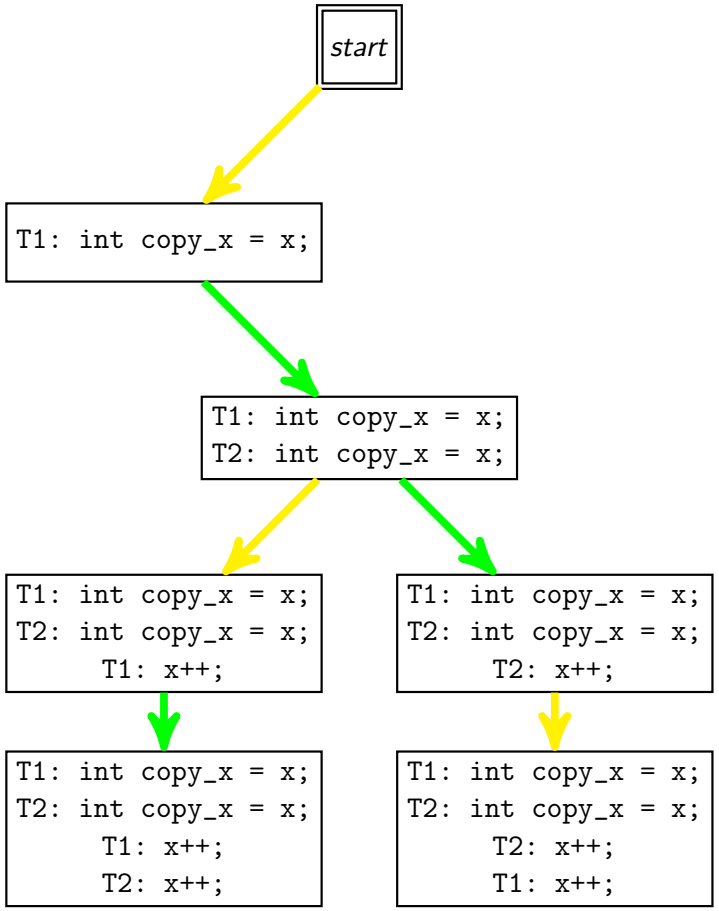
# State space exploration




```
    Thread 1
int copy_x = x;
x++;
```

```
    Thread 2
int copy_x = x;
x++;
```

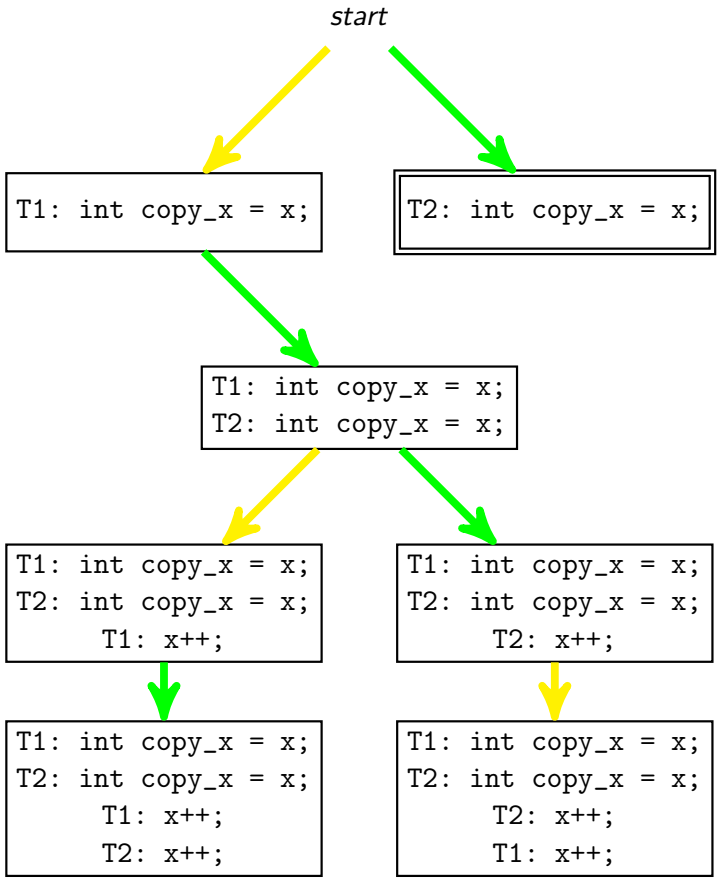
# State space exploration




```
    Thread 1
int copy_x = x;
x++;
```

```
    Thread 2
int copy_x = x;
x++;
```

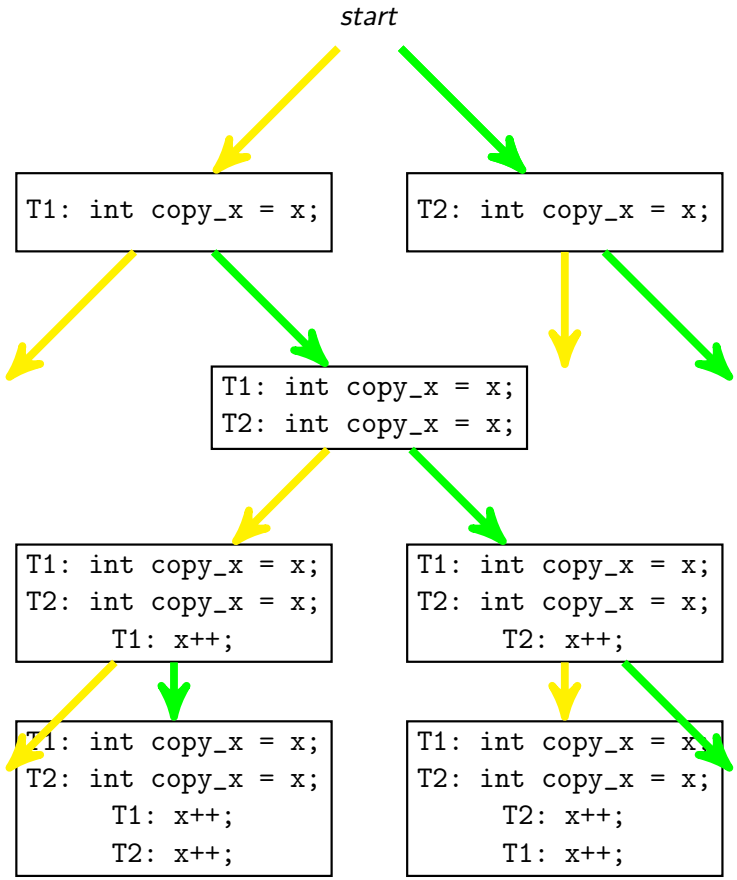
# State space exploration




```
    Thread 1
int copy_x = x;
x++;
```

```
    Thread 2
int copy_x = x;
x++;
```

# State space exploration




```
    Thread 1
int copy_x = x;
x++;
```

```
    Thread 2
int copy_x = x;
x++;
```

# State space exploration

Idea: analyze **all** reachable program states.

# State space exploration

Idea: analyze **all** reachable program states.
Traverse a tree of all allowed instruction permutations.

# State space exploration

Idea: analyze **all** reachable program states.
Traverse a tree of all allowed instruction permutations.
Every leaf in a tree contains a program trace. Check it for

- Assertions
- Unexpected exceptions
- Deadlocks

# State space exploration

Idea: analyze **all** reachable program states.
Traverse a tree of all allowed instruction permutations.
Every leaf in a tree contains a program trace. Check it for

- Assertions
- Unexpected exceptions
- Deadlocks

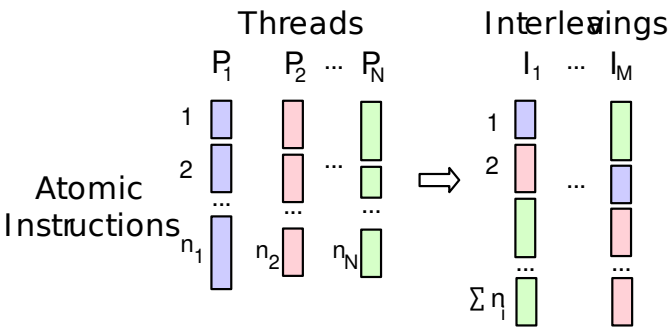
Hypothesis: every leaf is ok $\rightarrow$ program is correct

# State space exploration

Idea: traverse a tree of all allowed instruction permutations
Hypothesis: every leaf is ok $\rightarrow$ program is correct

# State space exploration

Idea: traverse a tree of all allowed instruction permutations
Hypothesis: every leaf is ok $\rightarrow$ program is correct
Practical consideration:




$$M = \frac{(\sum_{i=1}^{N} n_i)!}{\prod_{i=1}^{N} (n_i!)}$$

# State space exploration

Idea: traverse a tree of all allowed instruction permutations
Hypothesis: every leaf is ok $\rightarrow$ program is correct
Horror story of the day: Java Memory Model is not sequentially consistent

JLS 17.4.3:
«If we were to use sequential consistency as our memory model, many of the compiler and processor optimizations that we have discussed would be illegal.»

# State space exploration

Idea: traverse a tree of all allowed instruction permutations
~~Hypothesis: every leaf is ok $\rightarrow$ program is correct~~
Horror story of the day: Java Memory Model is not sequentially consistent

JLS 17.4.3:
«If we were to use sequential consistency as our memory model, many of the compiler and processor optimizations that we have discussed would be illegal.»

**Corrected hypothesis**: there exists trace with an error $\rightarrow$ program has a bug and a reproducing scenario

# Software model checking

Allen Emerson, E.; Clarke, Edmund M. (1980) «Characterizing correctness properties of parallel programs using fixpoints»

# Software model checking

Allen Emerson, E.; Clarke, Edmund M. (1980) «Characterizing correctness properties of parallel programs using fixpoints»

Turing Award 2007 for Clarke, Emerson, Sifakis: «... for their seminal work founding and developing the field of model checking»

# Software model checking

Allen Emerson, E.; Clarke, Edmund M. (1980) «Characterizing correctness properties of parallel programs using fixpoints»

Turing Award 2007 for Clarke, Emerson, Sifakis: «... for their seminal work founding and developing the field of model checking»

Significant tools:

- SPIN (Plan 9 kernel: verified `sleep` and `wakeup` primitives)
- TLA+ `http://research.microsoft.com/en-us/um/people/lamport/tla/formal-methods-amazon.pdf`

# Software model checking

Allen Emerson, E.; Clarke, Edmund M. (1980) «Characterizing correctness properties of parallel programs using fixpoints»

Turing Award 2007 for Clarke, Emerson, Sifakis: «... for their seminal work founding and developing the field of model checking»

Significant tools:

- SPIN (Plan 9 kernel: verified `sleep` and `wakeup` primitives)
- TLA+ `http://research.microsoft.com/en-us/um/people/lamport/tla/formal-methods-amazon.pdf`

TLA on Hydra:
`https://2021.hydraconf.com/2021/msk/talks/5oob1c0bvepx2cgfm8botq`

# Software model checking

Pro:

- Analysis = graph search

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample
- Better hardware – faster analysis

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample
- Better hardware – faster analysis
- Academic

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample
- Better hardware – faster analysis
- Academic

Cons:

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample
- Better hardware – faster analysis
- Academic

Cons:

- Combinatorial explosion

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample
- Better hardware – faster analysis
- Academic

Cons:

- Combinatorial explosion
- «Special» languages for model description

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample
- Better hardware – faster analysis
- Academic

Cons:

- Combinatorial explosion
- «Special» languages for model description

https://github.com/elastic/elasticsearch-formal-models/blob/master/ZenWithTerms/tla/ZenWithTerms.tla

```
CommitHasQuorumVsPreviousCommittedConfiguration ==
  \A mc \in messages: mc.method = Commit
    => (\A mprq \in messages: (/\ mprq.method  = PublishRequest
                               /\ mprq.term    = mc.term
                               /\ mprq.version = mc.version)

          => IsQuorum({mprs.source: mprs \in {mprs \in messages: /\ mprs.method = PublishResponse
                                                                 /\ mprs.term = mprq.term
                                                                 /\ mprs.version = mprq.version
                      }}, mprq.commConf))
```

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample
- Better hardware – faster analysis
- Academic

Cons:

- Combinatorial explosion
- «Special» languages for model description
- Academic

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample
- Better hardware – faster analysis
- Academic

Cons:

- Combinatorial explosion
- «Special» languages for model description
- Academic

`https://en.wikipedia.org/wiki/Linear_temporal_logic`
In logic, linear temporal logic or linear-time temporal logic (LTL) is a modal temporal logic with modalities referring to time. In LTL, one can encode formulae about the future of paths, e.g., a condition will eventually be true, a condition will be true until another fact becomes true, etc.

# Software model checking

Pro:

- Analysis = graph search
- Provides counterexample
- Better hardware – faster analysis
- Academic

Cons:

- Combinatorial explosion
- «Special» languages for model description
- Academic
- Production-ready code $\neq$ it's specification

`https://probablydance.com/2020/10/31/using-tla-in-the-real-world-to-understand-a-glibc-bug`

«... so to start off I cut everything that we didn't strictly need: Support for timeouts and shared memory, support for thread cancellation, a spin-wait optimization, all mentions of memory orderings, LIBC_PROBE and other things I'm probably forgetting»

# Java Path Finder

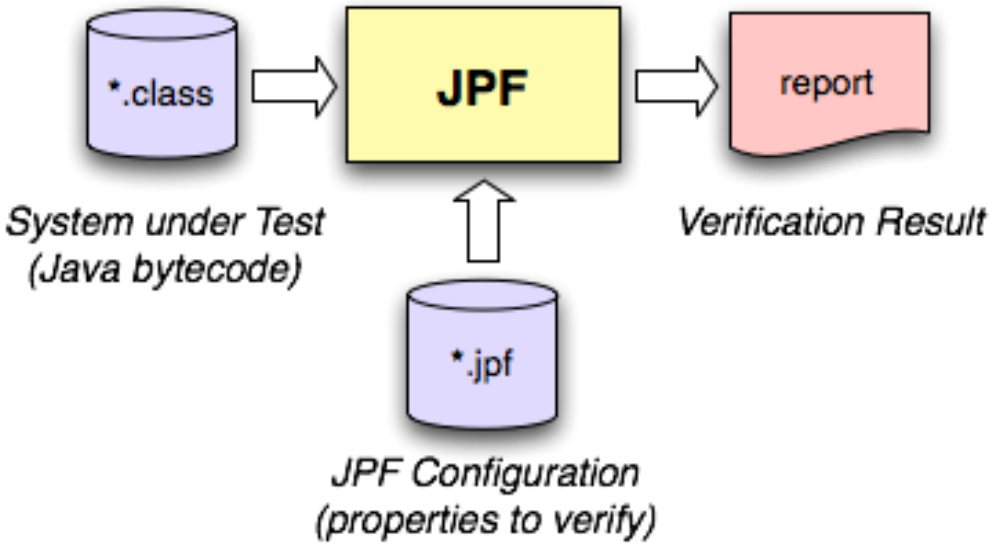
Extensible model checking framework for Java bytecode programs

# Java Path Finder

Extensible model checking framework for Java bytecode programs
Developed at the NASA Ames Research Center. Open sourced in 2005 (Apache-2.0)
`https://github.com/javapathfinder/jpf-core`

# Java Path Finder

Extensible model checking framework for Java bytecode programs
Developed at the NASA Ames Research Center. Open sourced in 2005 (Apache-2.0)
`https://github.com/javapathfinder/jpf-core`
Effectively – research JVM written in Java



# We are not going to Mars literally, aren‘t we?

Framework from NASA Ames Research Center.

# We are not going to Mars literally, aren't we?

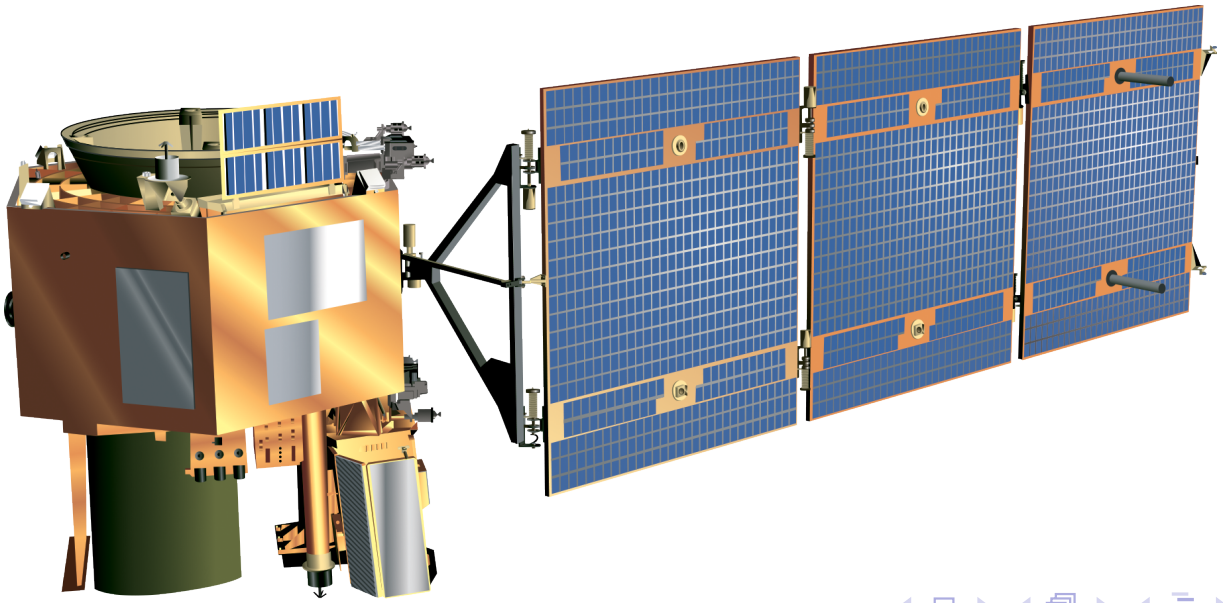
Framework from NASA Ames Research Center.

https://www.nasa.gov/centers/ames/news/releases/2005/05_28AR.html
«JPF was used to detect inconsistencies in the executive software for the K9 Rover at NASA Ames. The K9 rover is a six-wheeled, solar-powered rover developed jointly at NASA Ames and NASA's Jet Propulsion Laboratory»

# We are not going to Mars literally, aren't we?

Framework from NASA Ames Research Center.

https://www.nasa.gov/centers/ames/news/releases/2005/05_28AR.html
https://en.wikipedia.org/wiki/Earth_Observing-1
« ... scientists used elements of JPF to develop verification computer code for Livingstone 2 software, a diagnosis system now flying on the EO-1 spacecraft».



# Multiple producer multiple consumer intrusive queue

Task: implement concurrent queue

```
public abstract class Queue {

    public static class Node {
        public Queue.Node next;
    }

    public abstract void enqueue(Node element);

    // null if queue is empty
    public abstract Node dequeue();
}
```

# Baseline implementation

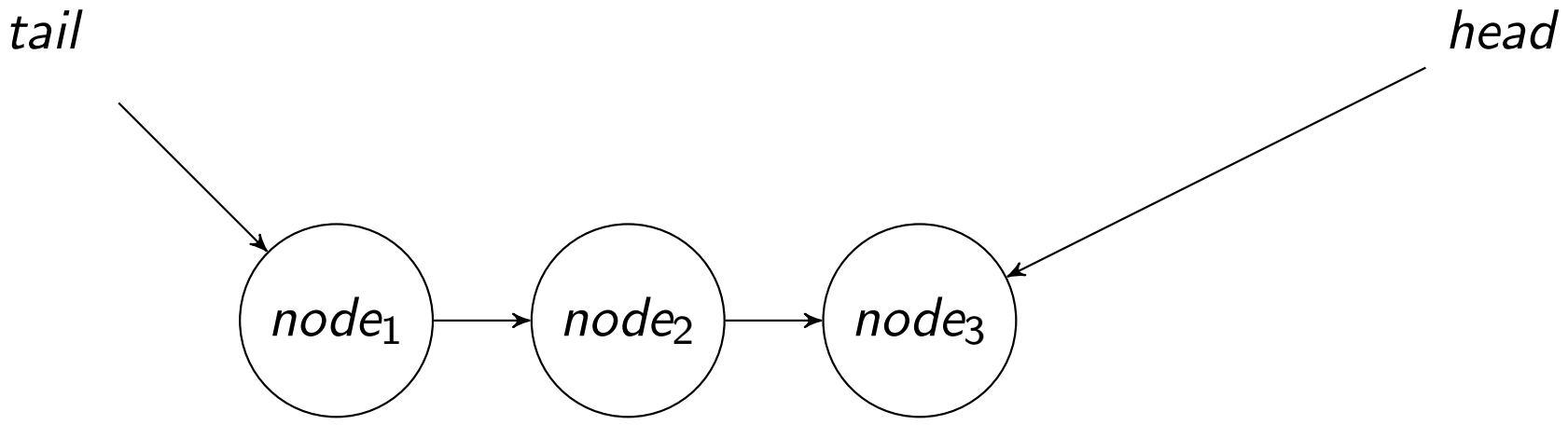
```
public class BaselineQueue extends Queue {

    final LinkedList<Queue.Node> impl = new LinkedList<>();

    @Override
    public synchronized void enqueue(Queue.Node element) {
        impl.add(element);
    }

    @Override
    public synchronized Queue.Node dequeue() {
        return impl.poll();
    }
}
```

Straightforward

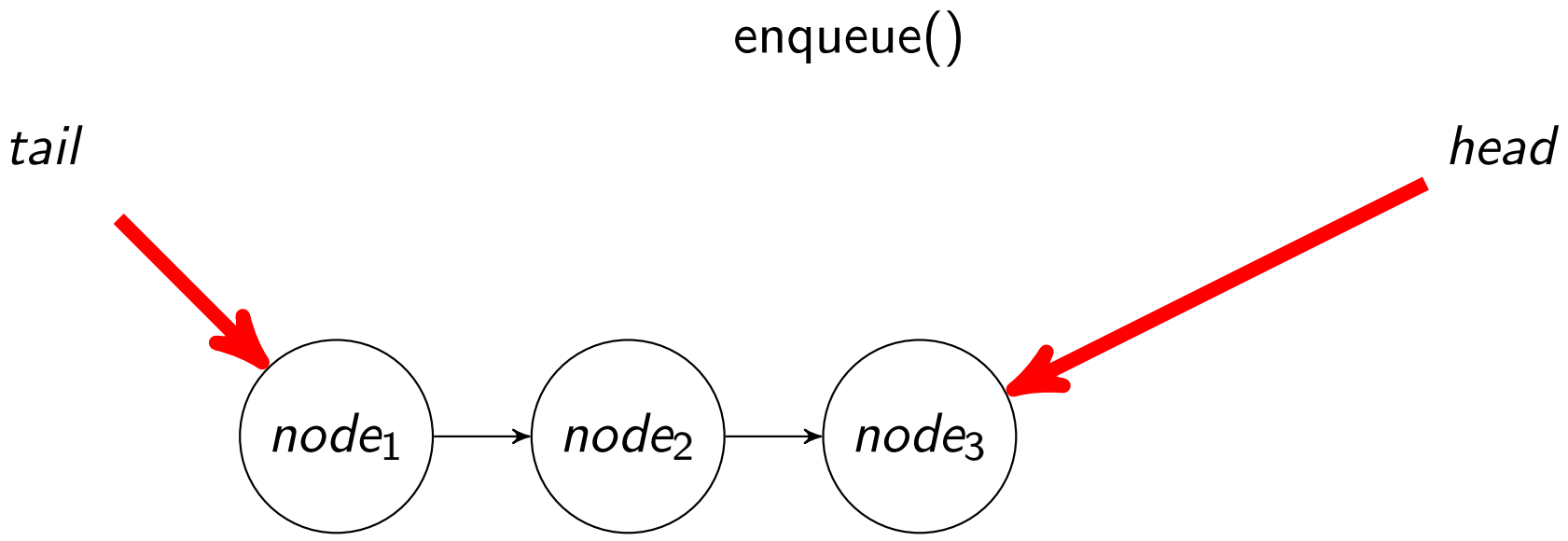
Memory consumption
Scalability [1]

[1] with all due respect to built-in assymetric synchronization in JVM

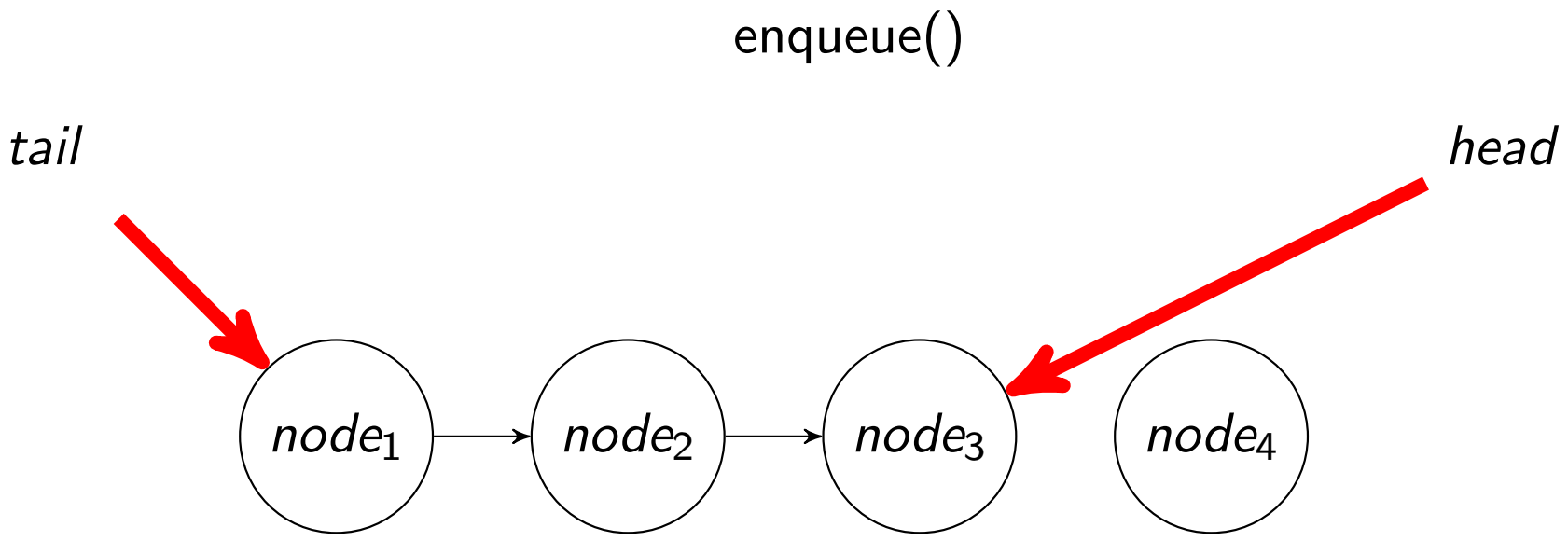
# Contention problem: enqueue



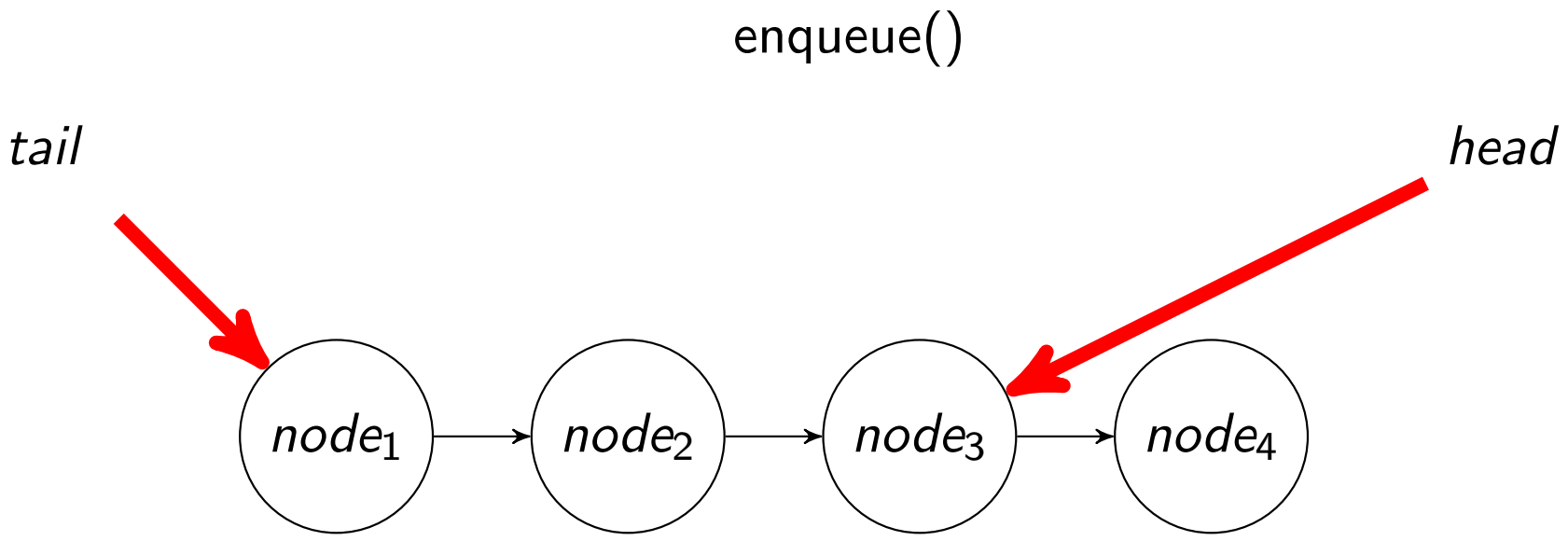
# Contention problem: enqueue



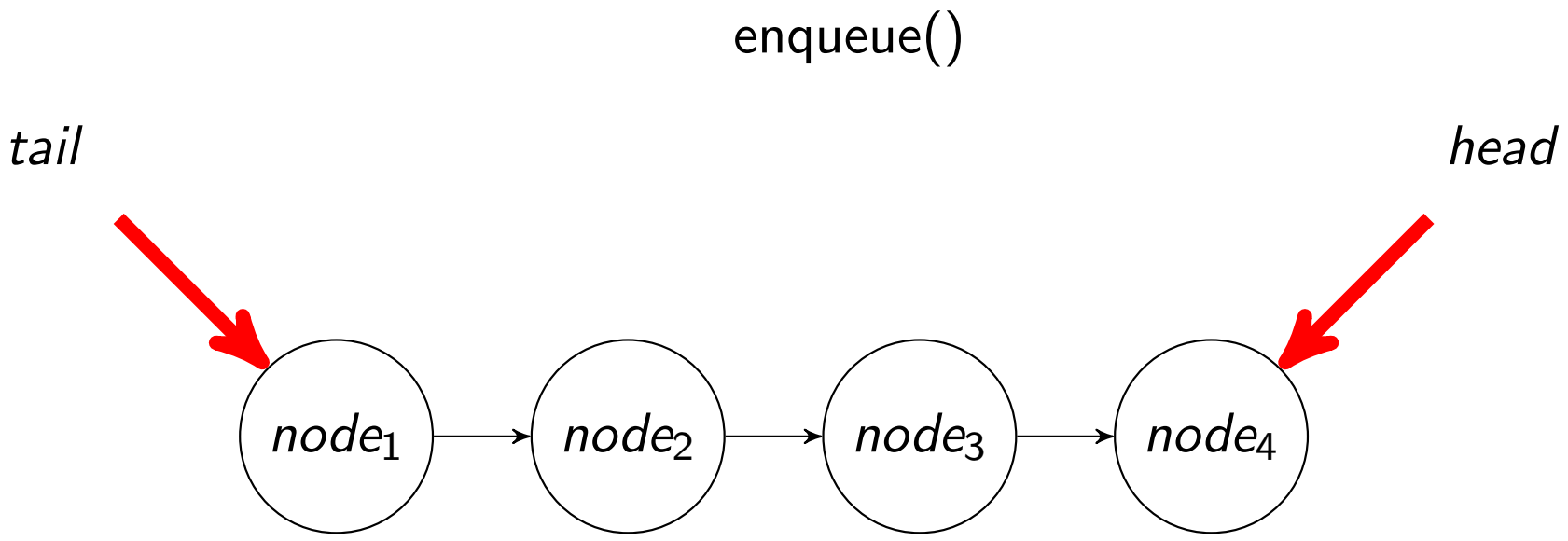
# Contention problem: enqueue



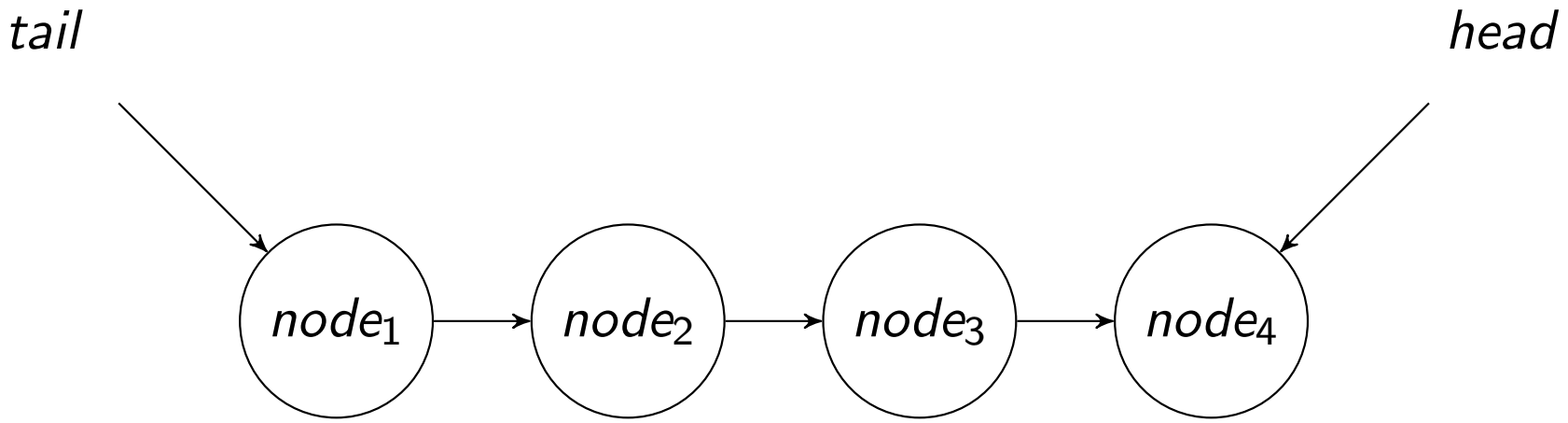
# Contention problem: enqueue



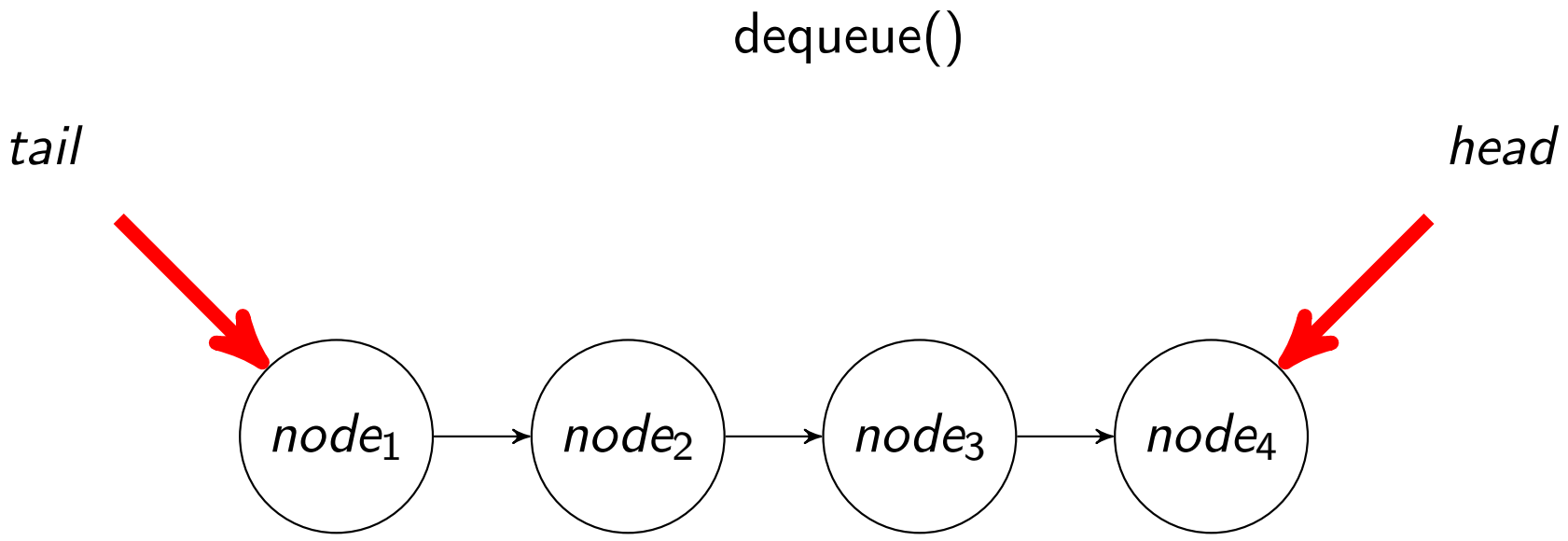
# Contention problem: enqueue



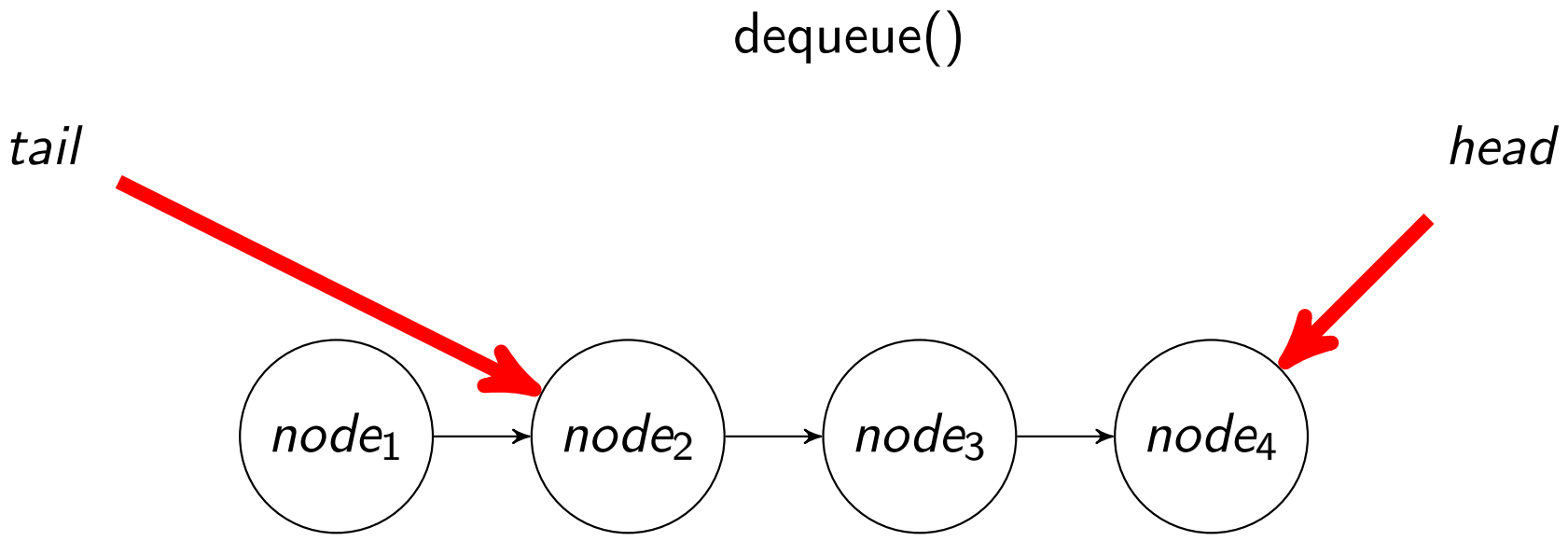
# Contention problem: dequeue



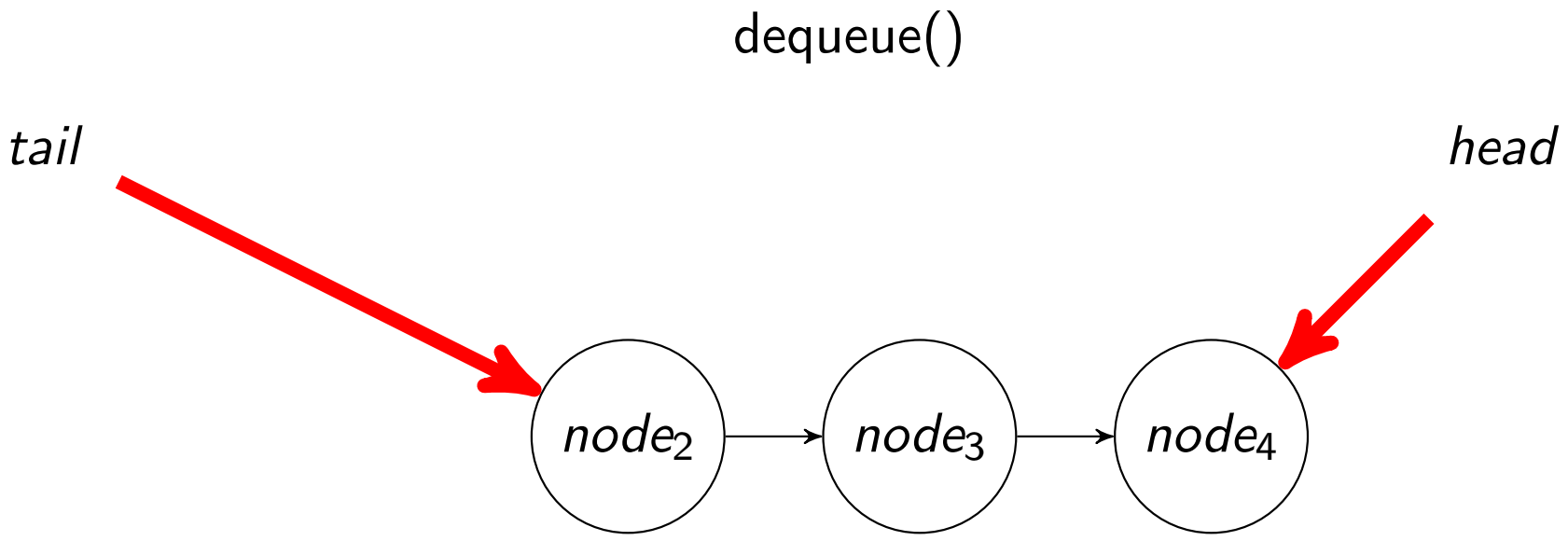
# Contention problem: dequeue



# Contention problem: dequeue



# Contention problem: dequeue



# Separate locks: enqueue

```
void enqueue(Node element) {
  synchronized (H_LOCK)
  {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

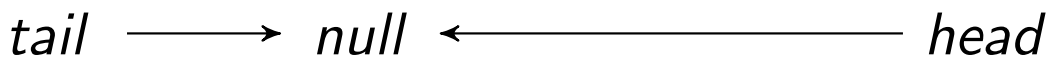
```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

$$tail \longrightarrow null \longleftarrow head$$

# Separate locks: enqueue

```
void enqueue(Node element) {
  synchronized (H_LOCK)
  {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```



# Separate locks: enqueue

```
void enqueue(Node element) {
  synchronized (H_LOCK)
  {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

# Separate locks: enqueue

```
void enqueue(Node element) {
  synchronized (H_LOCK)
  {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

# Separate locks: enqueue

```
void enqueue(Node element) {
  synchronized (H_LOCK)
  {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

# Separate locks: enqueue

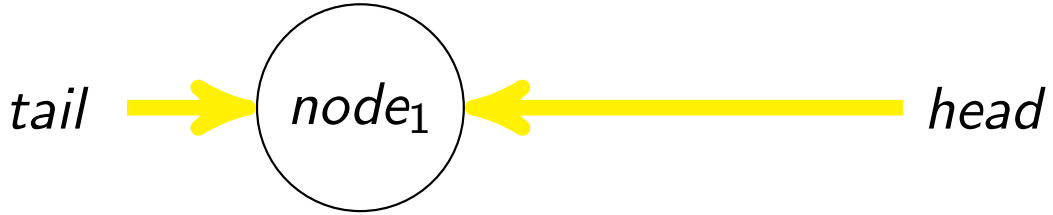
```
void enqueue(Node element) {
  synchronized (H_LOCK)
  {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

# Separate locks: enqueue

```
void enqueue(Node element) {
  synchronized (H_LOCK)
  {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

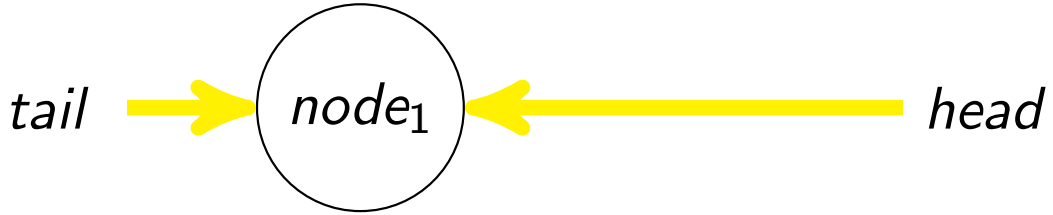
```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```



# Separate locks: enqueue

```
void enqueue(Node element) {
  synchronized (H_LOCK)
  {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

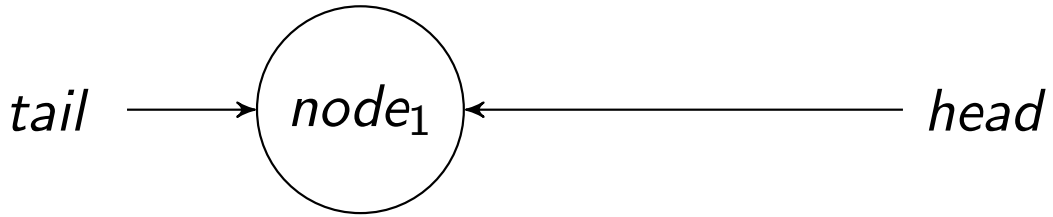
```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```



# Separate locks: enqueue

```
void enqueue(Node element) {
  synchronized (H_LOCK)
  {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

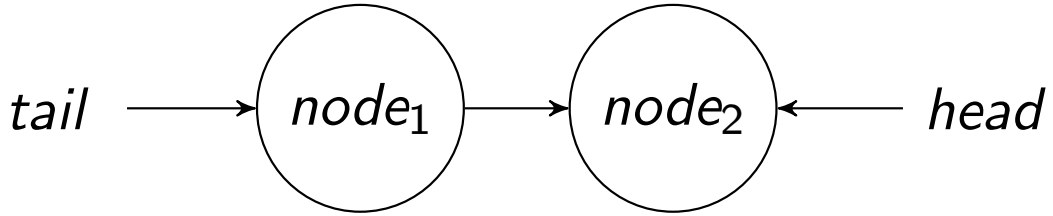
```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```



# Separate locks: dequeue

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

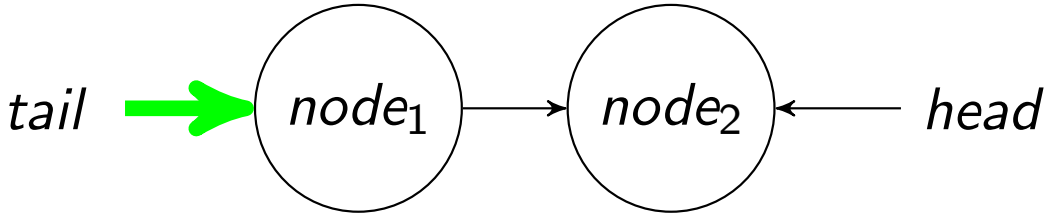
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK) {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: dequeue

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

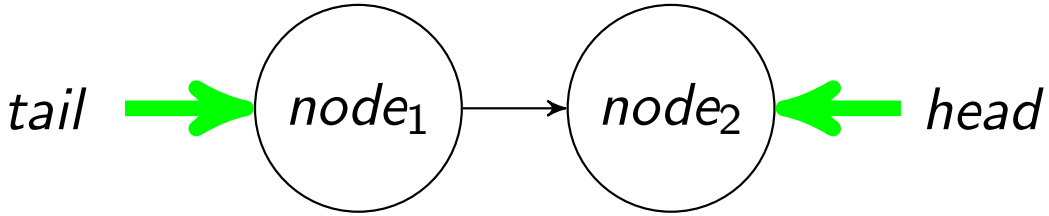
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK) {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: dequeue

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

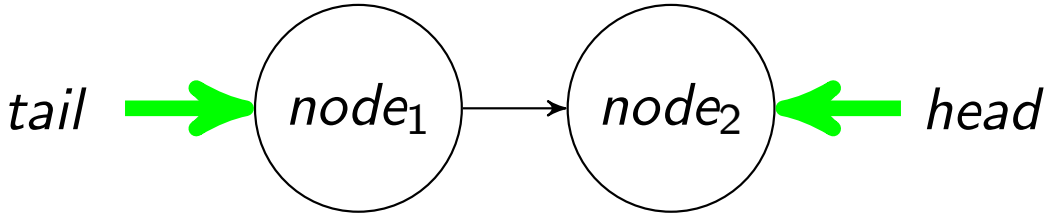
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK) {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: dequeue

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

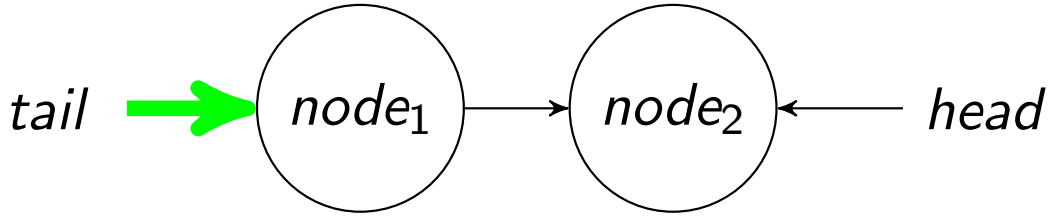
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK) {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: dequeue

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

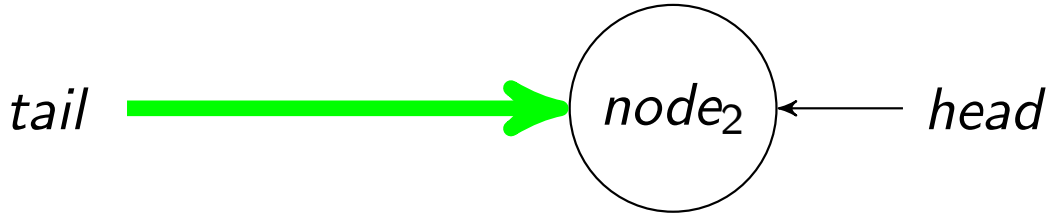
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK) {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: dequeue

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

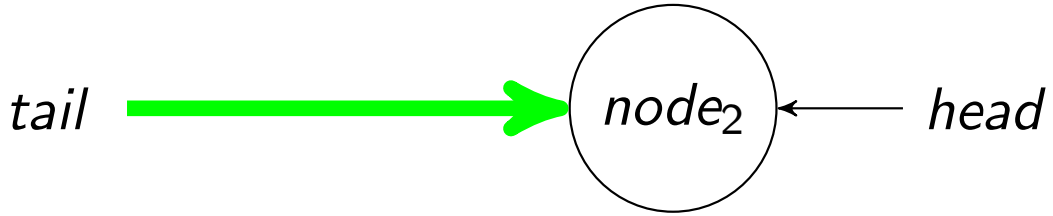
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK) {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: dequeue

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

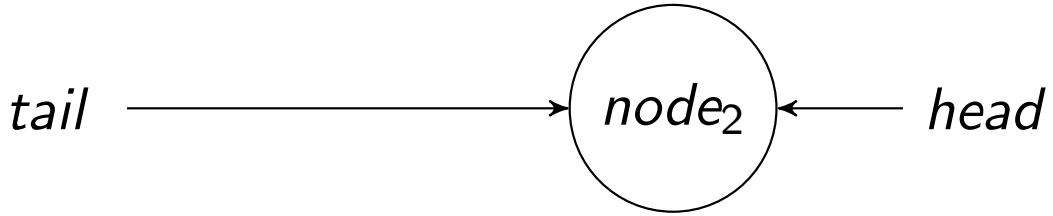
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK) {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: dequeue

```
init {
  Node head = null;
  Node tail = null;
  Object H_LOCK = new Object();
  Object T_LOCK = new Object();
}
```

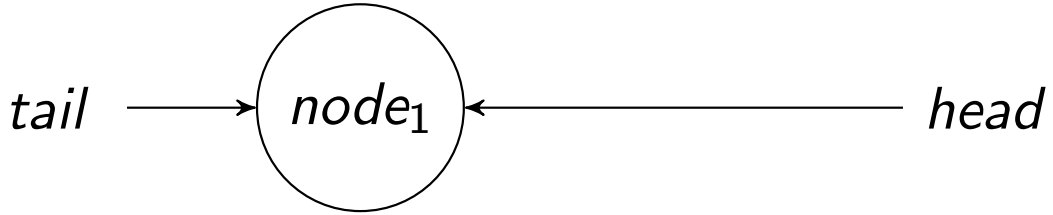
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK) {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

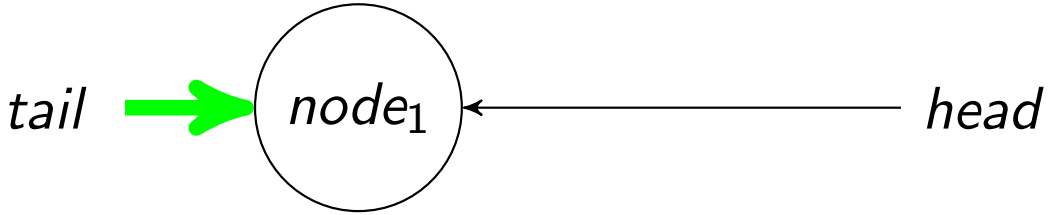
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

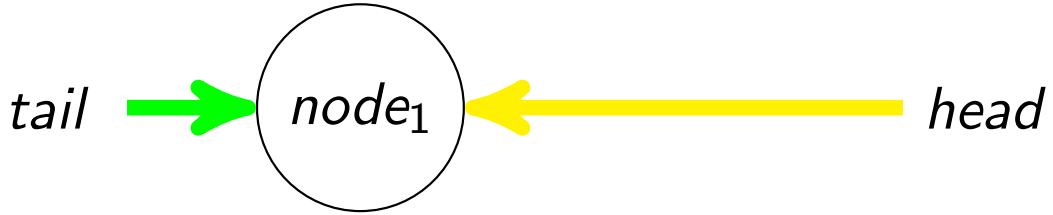
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

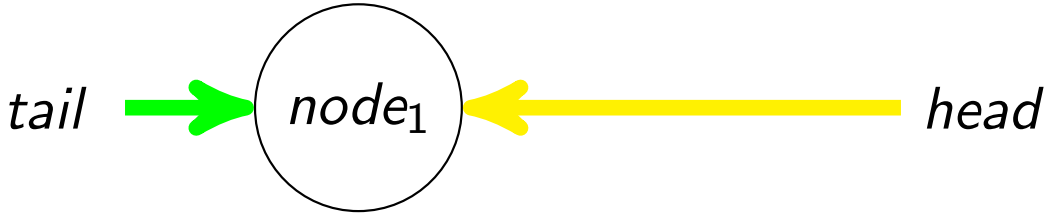
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

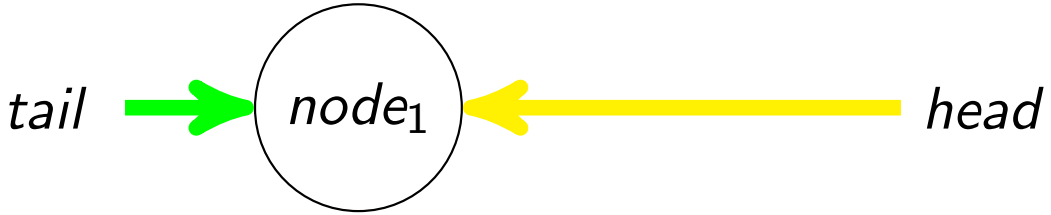
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

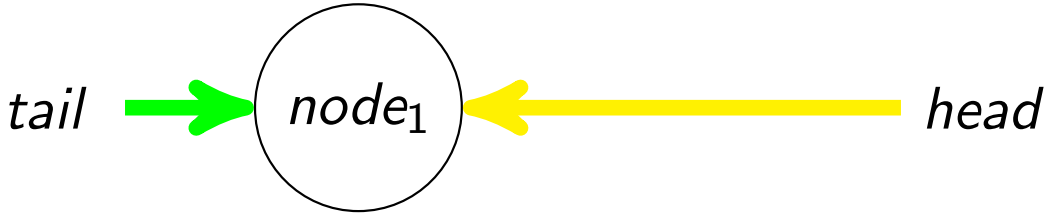
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

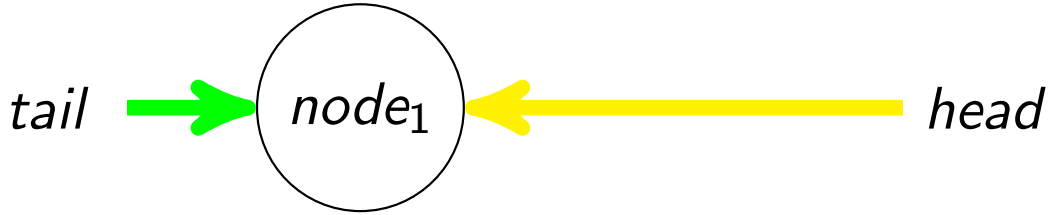
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

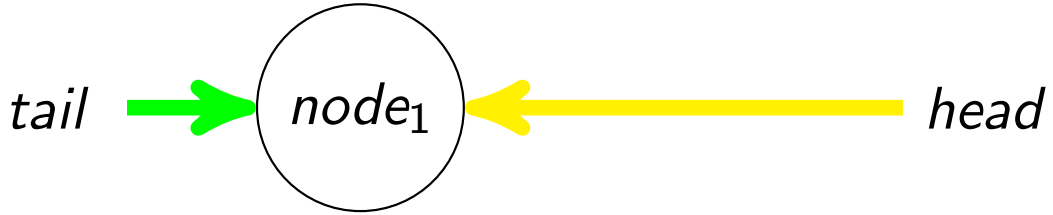
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

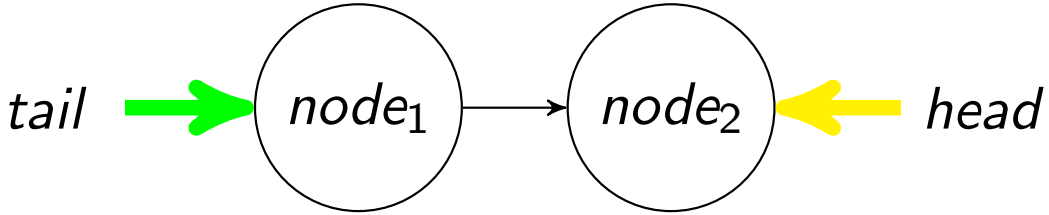
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

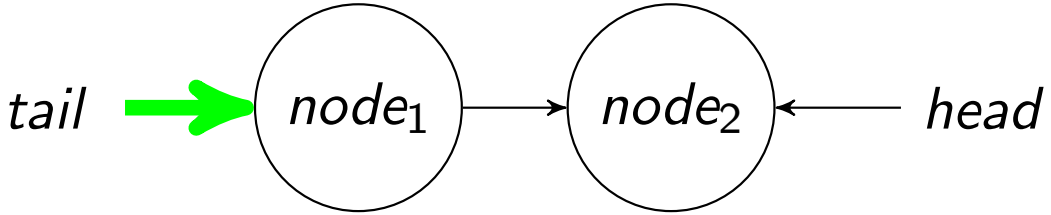
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

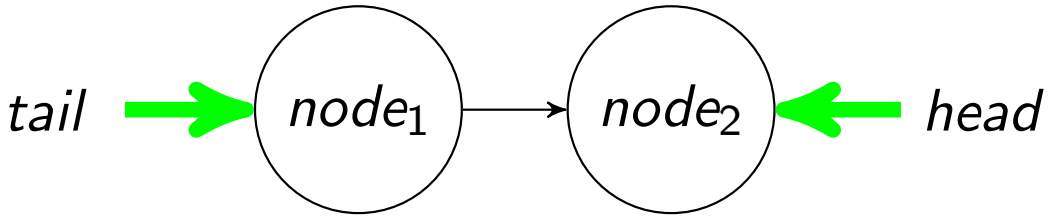
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

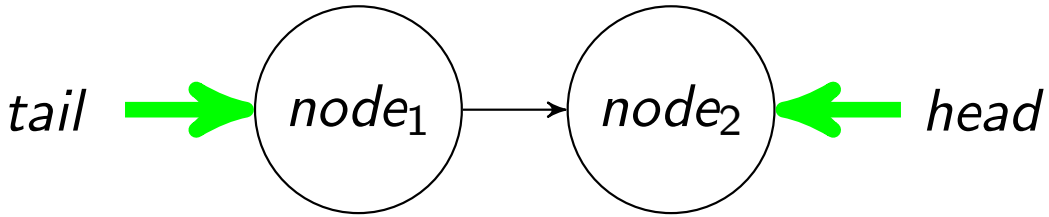
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

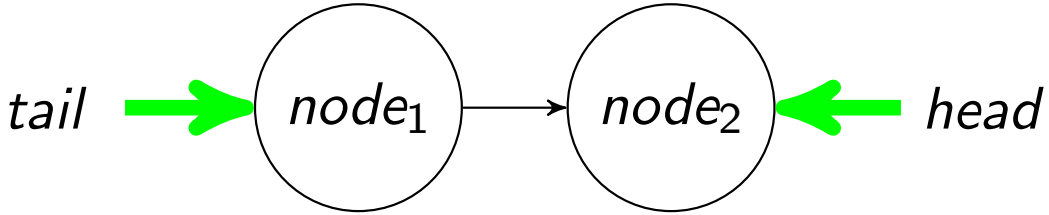
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

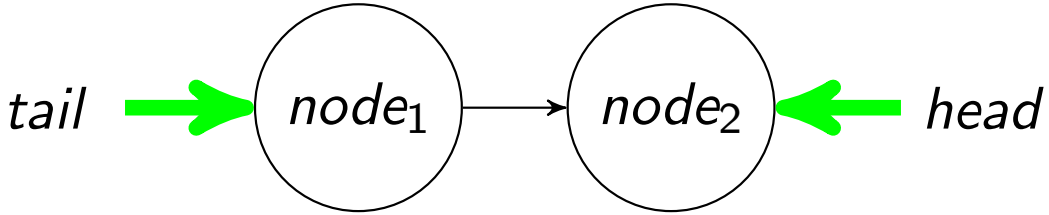
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

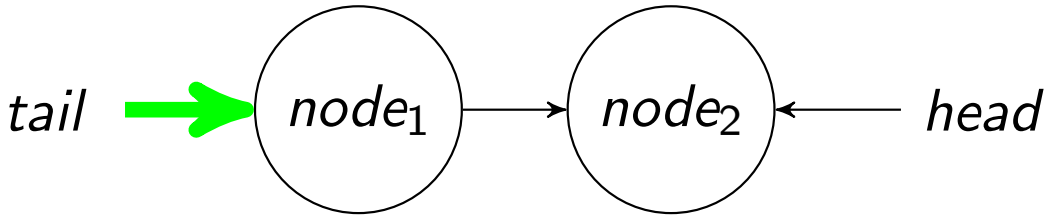
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

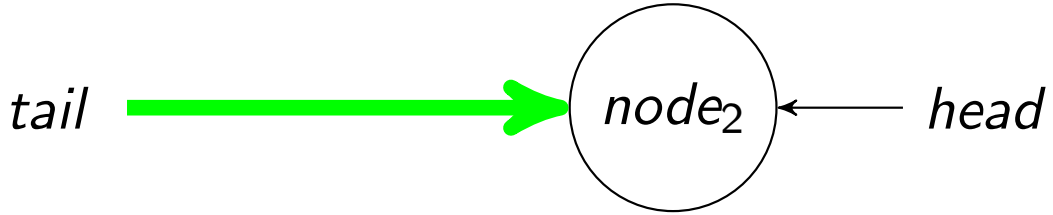
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

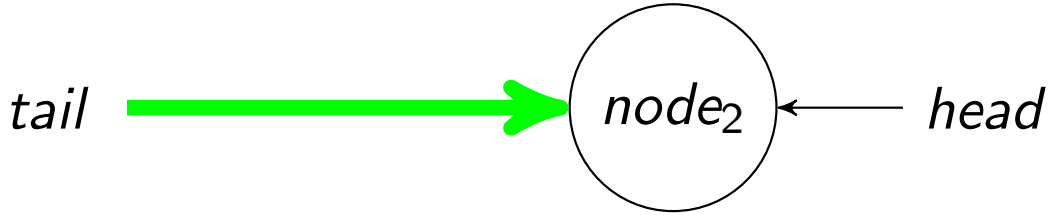
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

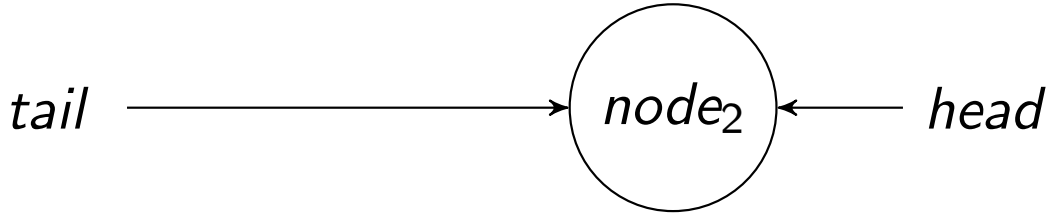
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: concurrent enqueue/dequeue

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK) {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Running JPF: config

```
====================Test.jpf====================
target=Test
target.args=

classpath=.
sourcepath=.

report.console.property_violation = output,trace
```

# Running JPF: console

```
/home/JPF/jpf-core/bin/jpf Test.jpf
```

# Running JPF: console

## /home/JPF/jpf-core/bin/jpf Test.jpf

```
JavaPathfinder core system v8.0 (rev 32) - (C) 2005-2014 United States Government.

All rights reserved.

. . .

================================= search started: 10/24/19 10:43 PM

================================= trace 1

----------------- transition 0 thread: 0

gov.nasa.jpf.vm.choice.ThreadChoiceFromSet id:"ROOT 1/1,isCascaded:false

[3157 insn w/o sources]

Test.java:64 : private static final Random rand = new Random(123);

[2 insn w/o sources]

Test.java:64 : private static final Random rand = new Random(123);

. . .
```

# Running JPF: console

```
/home/JPF/jpf-core/bin/jpf Test.jpf
JavaPathfinder core system v8.0 (rev 32) - (C) 2005-2014 United States Government.
All rights reserved.
. . .
================================= search started: 10/24/19 10:43 PM
================================= trace 1
----------------- transition 0 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet id:"ROOT 1/1,isCascaded:false
[3157 insn w/o sources]
Test.java:64 : private static final Random rand = new Random(123);
[2 insn w/o sources]
Test.java:64 : private static final Random rand = new Random(123);
. . .
```

**270 lines: every step of every running thread**

# Running JPF: error report

```
========================================================= results

error 1: gov.nasa.jpf.vm.NotDeadlockedProperty

"deadlock encountered: thread java.lang.Thread: ..."


========================================================= statistics

elapsed time: 00:00:00

states: new=603,visited=447,backtracked=1017,end=3

search: maxDepth=54,constraints=0

choice generators: thread=602 (signal=0,lock=108,sharedRef=349,threadApi=91,reschedule=54), data=0

heap: new=409,released=295,maxLive=390,gcCycles=971

instructions: 20933

max memory: 77MB

loaded code: classes=69,methods=1506
```

# Running JPF: error report

```
========================================================= results

error 1: gov.nasa.jpf.vm.NotDeadlockedProperty
"deadlock encountered: thread java.lang.Thread: ..."


========================================================= statistics

elapsed time: 00:00:00

states: new=603,visited=447,backtracked=1017,end=3

search: maxDepth=54,constraints=0

choice generators: thread=602 (signal=0,lock=108,sharedRef=349,threadApi=91,reschedule=54), data=0

heap: new=409,released=295,maxLive=390,gcCycles=971

instructions: 20933

max memory: 77MB

loaded code: classes=69,methods=1506
```

# Separate locks: bug visualization

```
void enqueue(Node element) {
   synchronized (H_LOCK) {
     Node curHead = this.head;
     if (curHead == null) {
       synchronized (T_LOCK)
       {
         head = tail = element;
         return;
       }
     }
     curHead.next = element;
     element.next = null;
     this.head = element;
   }
}
```

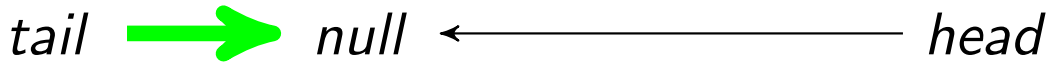
```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```

$$tail \longrightarrow null \longleftarrow head$$

# Separate locks: bug visualization

```
void enqueue(Node element) {
   synchronized (H_LOCK) {
     Node curHead = this.head;
     if (curHead == null) {
       synchronized (T_LOCK)
       {
         head = tail = element;
         return;
       }
     }
     curHead.next = element;
     element.next = null;
     this.head = element;
   }
}
```

```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```



# Separate locks: bug visualization

```
void enqueue(Node element) {
  synchronized (H_LOCK) {
    Node curHead = this.head;
    if (curHead == null) {
      synchronized (T_LOCK)
      {
        head = tail = element;
        return;
      }
    }
    curHead.next = element;
    element.next = null;
    this.head = element;
  }
}
```

```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```

# Separate locks: bug visualization

```
void enqueue(Node element) {
   synchronized (H_LOCK) {
     Node curHead = this.head;
     if (curHead == null) {
       synchronized (T_LOCK)
       {
         head = tail = element;
         return;
       }
     }
     curHead.next = element;
     element.next = null;
     this.head = element;
   }
}
```

```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```

# Separate locks: bug visualization

```
void enqueue(Node element) {
   synchronized (H_LOCK) {
     Node curHead = this.head;
     if (curHead == null) {
       synchronized (T_LOCK)
       {
         head = tail = element;
         return;
       }
     }
     curHead.next = element;
     element.next = null;
     this.head = element;
   }
}
```

```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```

# Separate locks: bug visualization

```
void enqueue(Node element) {
   synchronized (H_LOCK) {
     Node curHead = this.head;
     if (curHead == null) {
       synchronized (T_LOCK)
       {
         head = tail = element;
         return;
       }
     }
     curHead.next = element;
     element.next = null;
     this.head = element;
   }
}
```

```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```

# Separate locks: bug visualization

```
void enqueue(Node element) {
   synchronized (H_LOCK) {
     Node curHead = this.head;
     if (curHead == null) {
       synchronized (T_LOCK)
       {
         head = tail = element;
         return;
       }
     }
     curHead.next = element;
     element.next = null;
     this.head = element;
   }
}
```

```
Node dequeue() {
  synchronized (T_LOCK) {
    synchronized (H_LOCK)
    {
      if (tail == head) {
        Node result = this.head;
        head = tail = null;
        return result;
      }
    }
    Node result = this.tail;
    this.tail = result.next;
    return result;
  }
}
```

# Second attempt: minimize lock surface

```
private Node getHead() {
    synchronized (HEAD_LOCK) {
        return head;
    }
}

private void setHead(Node newHead) {
    synchronized (HEAD_LOCK) {
        head = newHead;
    }
}

private Node getTail() {
    synchronized (TAIL_LOCK) {
        return tail;
    }
}

private void setTail(Node newTail) {
    synchronized (TAIL_LOCK) {
        tail = newTail;
    }
}
```

# Running JPF

```
====================================================== results

error 1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty "java.lang.NullPointerException: ..."


====================================================== statistics

elapsed time: 00:00:00

states: new=126,visited=70,backtracked=162,end=2

search: maxDepth=44,constraints=0

choice generators: thread=125 (signal=0,lock=42,sharedRef=45,threadApi=26,reschedule=12), data=0

heap: new=419,released=103,maxLive=390,gcCycles=183

instructions: 8910

max memory: 61MB

loaded code: classes=73,methods=1540
```

**320 lines: every step of every running thread**

# Running JPF

```
====================Test.jpf====================
target=Test
target.args=

classpath=.
sourcepath=.

# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
search.class = gov.nasa.jpf.search.heuristic.BFSHeuristic
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

report.console.property_violation = output,trace
```

# Running JPF

```
========================================================= results

error 1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty "java.lang.NullPointerException: ..."


========================================================= statistics

elapsed time: 00:00:00

states: new=351,visited=305,backtracked=655,end=0

search: maxDepth=16,constraints=0

choice generators: thread=281 (signal=0,lock=155,sharedRef=123,threadApi=3,reschedule=0), data=0

heap: new=591,released=44,maxLive=389,gcCycles=586

instructions: 14320

max memory: 61MB

loaded code: classes=73,methods=1540
```

**180 lines instead of 320 lines**

# Running JPF

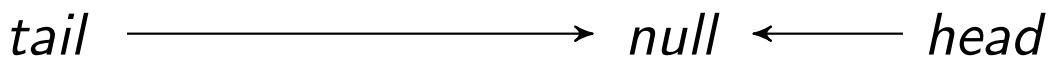
```
========================================================= results
error 1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
"java.lang.NullPointerException: ..."


========================================================= statistics

elapsed time: 00:00:00

states: new=351,visited=305,backtracked=655,end=0

search: maxDepth=16,constraints=0

choice generators: thread=281 (signal=0,lock=155,sharedRef=123,threadApi=3,reschedule=0), data=0

heap: new=591,released=44,maxLive=389,gcCycles=586

instructions: 14320

max memory: 61MB

loaded code: classes=73,methods=1540
```

**180 lines instead of 320 lines**

# Synchronized accessors: bug visualization

```
void enqueue(Node element) {
    Node curHead = getHead();
    if (curHead == null) {
        setHead(element);

        setTail(element);
        return;
    }

    curHead.next = element;
    element.next = null;
    setHead(element);
}
```

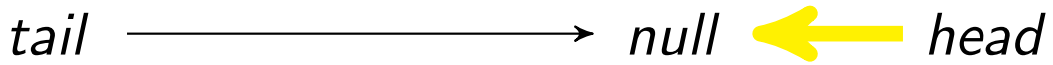
```
Node dequeue() {
    Node tailCopy = getTail();
    Node headCopy = getHead();
    if (tailCopy == headCopy) {
        setHead(null);
        setTail(null);
        return headCopy;
    }

    setTail(tailCopy.next);
    return tailCopy;
}
```



# Synchronized accessors: bug visualization

```
void enqueue(Node element) {
    Node curHead = getHead();
    if (curHead == null) {
        setHead(element);

        setTail(element);
        return;
    }

    curHead.next = element;
    element.next = null;
    setHead(element);
}
```

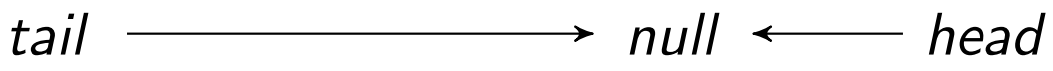
```
Node dequeue() {
    Node tailCopy = getTail();
    Node headCopy = getHead();
    if (tailCopy == headCopy) {
        setHead(null);
        setTail(null);
        return headCopy;
    }

    setTail(tailCopy.next);
    return tailCopy;
}
```



# Synchronized accessors: bug visualization

```
void enqueue(Node element) {
    Node curHead = getHead();
    if (curHead == null) {
        setHead(element);

        setTail(element);
        return;
    }

    curHead.next = element;
    element.next = null;
    setHead(element);
}
```

```
Node dequeue() {
    Node tailCopy = getTail();
    Node headCopy = getHead();
    if (tailCopy == headCopy) {
        setHead(null);
        setTail(null);
        return headCopy;
    }

    setTail(tailCopy.next);
    return tailCopy;
}
```



# Synchronized accessors: bug visualization

```
void enqueue(Node element) {
    Node curHead = getHead();
    if (curHead == null) {
        setHead(element);

        setTail(element);
        return;
    }

    curHead.next = element;
    element.next = null;
    setHead(element);
}
```

```
Node dequeue() {
    Node tailCopy = getTail();
    Node headCopy = getHead();
    if (tailCopy == headCopy) {
        setHead(null);
        setTail(null);
        return headCopy;
    }

    setTail(tailCopy.next);
    return tailCopy;
}
```

# Synchronized accessors: bug visualization

```
void enqueue(Node element) {
    Node curHead = getHead();
    if (curHead == null) {
        setHead(element);

        setTail(element);
        return;
    }

    curHead.next = element;
    element.next = null;
    setHead(element);
}
```

```
Node dequeue() {
    Node tailCopy = getTail();
    Node headCopy = getHead();
    if (tailCopy == headCopy) {
        setHead(null);
        setTail(null);
        return headCopy;
    }

    setTail(tailCopy.next);
    return tailCopy;
}
```

# Synchronized accessors: bug visualization

```
void enqueue(Node element) {
    Node curHead = getHead();
    if (curHead == null) {
        setHead(element);

        setTail(element);
        return;
    }

    curHead.next = element;
    element.next = null;
    setHead(element);
}
```

```
Node dequeue() {
    Node tailCopy = getTail();
    Node headCopy = getHead();
    if (tailCopy == headCopy) {
        setHead(null);
        setTail(null);
        return headCopy;
    }

    setTail(tailCopy.next);
    return tailCopy;
}
```

# Synchronized accessors: bug visualization

```
void enqueue(Node element) {
    Node curHead = getHead();
    if (curHead == null) {
        setHead(element);

        setTail(element);
        return;
    }

    curHead.next = element;
    element.next = null;
    setHead(element);
}
```

```
Node dequeue() {
    Node tailCopy = getTail();
    Node headCopy = getHead();
    if (tailCopy == headCopy) {
        setHead(null);
        setTail(null);
        return headCopy;
    }

    setTail(tailCopy.next);
    return tailCopy;
}
```

# Synchronized accessors: bug visualization

```
void enqueue(Node element) {
    Node curHead = getHead();
    if (curHead == null) {
        setHead(element);

        setTail(element);
        return;
    }

    curHead.next = element;
    element.next = null;
    setHead(element);
}
```

```
Node dequeue() {
    Node tailCopy = getTail();
    Node headCopy = getHead();
    if (tailCopy == headCopy) {
        setHead(null);
        setTail(null);
        return headCopy;
    }

    setTail(tailCopy.next);
    return tailCopy;
}
```

# Synchronized accessors: bug visualization

```
void enqueue(Node element) {
    Node curHead = getHead();
    if (curHead == null) {
        setHead(element);

        setTail(element);
        return;
    }

    curHead.next = element;
    element.next = null;
    setHead(element);
}
```

```
Node dequeue() {
    Node tailCopy = getTail();
    Node headCopy = getHead();
    if (tailCopy == headCopy) {
        setHead(null);
        setTail(null);
        return headCopy;
    }

    setTail(tailCopy.next);
    return tailCopy;
}
```

# Real-world example

MPSC queue by Dmitry Vyukov

- http://www.1024cores.net/home/lock-free-algorithms/queues/intrusive-mpsc-node-based-queue
- https://www.boost.org/doc/libs/1_63_0/boost/fiber/detail/context_mpsc_queue.hpp

Java port (https://github.com/JCTools):
java.org.jctools.queues.MpscLinkedQueue

# Data integrity problem

What if dequeue() would return same object to different threads?

```
class CheckedNode extends Queue.Node {
   volatile int x = 0;

   void beforeEnqueue() {
       assert x == 0;        x = 1;        assert x == 1;
   }

   void afterDequeue() {
       assert x == 1;        x = 0;        assert x == 0;
   }
}

static void task(Queue q) {
   CheckedNode n = new CheckedNode();
   n.beforeEnqueue();
   q.enqueue(n);
   CheckedNode v = q.dequeue();
   if (v != null) v.afterDequeue();
}

static void testTwoThreads(Queue q) {
   startInNewThread(() -> task(q));
   startInNewThread(() -> task(q));
}
```

# Running JPF

```
========================================================= results

error 1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty

"java.lang.AssertionError at ..."

========================================================= statistics

elapsed time: 00:00:00

states: new=720,visited=637,backtracked=1356,end=35

search: maxDepth=19,constraints=0

choice generators: thread=623 (signal=0,lock=90,sharedRef=457,threadApi=2,reschedule=74), data=0

heap: new=802,released=1890,maxLive=395,gcCycles=1259

instructions: 25342

max memory: 77MB

loaded code: classes=72,methods=1553
```

# Running JPF

```
========================================================= results

error 1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
"java.lang.AssertionError at ..."

========================================================= statistics

elapsed time: 00:00:00

states: new=720,visited=637,backtracked=1356,end=35

search: maxDepth=19,constraints=0

choice generators: thread=623 (signal=0,lock=90,sharedRef=457,threadApi=2,reschedule=74), data=0

heap: new=802,released=1890,maxLive=395,gcCycles=1259

instructions: 25342

max memory: 77MB

loaded code: classes=72,methods=1553
```

# MpscLinkedQueue bug

Scenario: two different threads simultaneously `dequeue` object from non-empty queue.

# MpscLinkedQueue bug

Scenario: two different threads simultaneously `dequeue` object from non-empty queue.

MPSC – Multiple Producer Single Consumer

# MpscLinkedQueue bug

Scenario: two different threads simultaneously `dequeue` object from non-empty queue.

MPSC – Multiple Producer **Single** Consumer

# MpscLinkedQueue bug

Scenario: two different threads simultaneously `dequeue` object from non-empty queue.

MPSC – Multiple Producer **Single** Consumer

Correct data structure was used in inappropriate multithreaded context.

# Recap

Discovered errors:

- Deadlock
- Internal invariants violation (`NullPointerException`)
- Logical error caught by `assert`: inappropriate usage

# Recap

Discovered errors:

- Deadlock
- Internal invariants violation (`NullPointerException`)
- Logical error caught by `assert`: inappropriate usage

Still need to write tests and `assertions`.

# Recap

Discovered errors:

- Deadlock
- Internal invariants violation (`NullPointerException`)
- Logical error caught by `assert`: inappropriate usage

Still need to write multithreaded tests and `assertions`.

# Recap

Discovered errors:

- Deadlock
- Internal invariants violation (`NullPointerException`)
- Logical error caught by `assert`: inappropriate usage

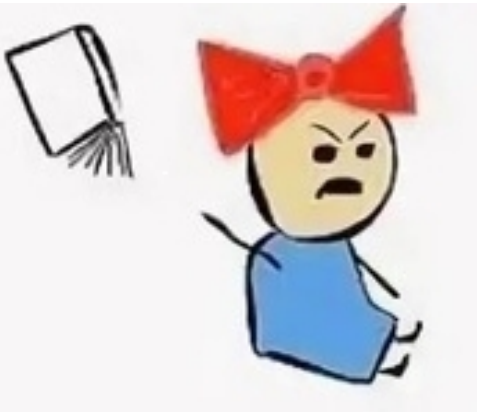
Still need to write multithreaded tests and elaborate `assertions`.

# Recap

Discovered errors:

- Deadlock
- Internal invariants violation (`NullPointerException`)
- Logical error caught by `assert`: inappropriate usage

Still need to write multithreaded tests and elaborate `assertions`.



# KSUH

"A Fair Fast Scalable Reader-Writer Lock ", 1993
Orran **K**rieger, Michael **S**tumm, Ron **U**nrau, Jonathan **H**anna
In Proc. of the International Conference on Parallel Processing

# KSUH

"A Fair Fast Scalable Reader-Writer Lock ", 1993
Orran **K**rieger, Michael **S**tumm, Ron **U**nrau, Jonathan **H**anna
In Proc. of the International Conference on Parallel Processing

Implemented in C.
Verified by SPIN (formal model written in Promela language).

# KSUH

"A Fair Fast Scalable Reader-Writer Lock ", 1993
Orran **K**rieger, Michael **S**tumm, Ron **U**nrau, Jonathan **H**anna
In Proc. of the International Conference on Parallel Processing

Implemented in C.
Verified by SPIN (formal model written in Promela language).

Different languages for implementation and verification.

# KSUH

"Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm", 2013
Dave Dice, Yossi Lev, Yujie Liu, Victor Luchangco, Mark Moir

# KSUH

"Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm", 2013
Dave Dice, Yossi Lev, Yujie Liu, Victor Luchangco, Mark Moir

20 years later.

# KSUH

"Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm", 2013
Dave Dice, Yossi Lev, Yujie Liu, Victor Luchangco, Mark Moir

20 years later.
KSUH implementation contained a bug connected with manual memory management.

# KSUH

"Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm", 2013
Dave Dice, Yossi Lev, Yujie Liu, Victor Luchangco, Mark Moir

20 years later.
KSUH implementation contained a bug connected with manual memory management.
Was reproduced on real hardware.

# KSUH

"Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm", 2013
Dave Dice, Yossi Lev, Yujie Liu, Victor Luchangco, Mark Moir

20 years later.
KSUH implementation contained a bug connected with manual memory management.
Was reproduced on real hardware.

Java Path Finder:

- 250 lines of code for impl.
- 90 lines for test.
- Analysis found a problem in 7 minutes.
- It took 700 Mb RAM.

# KSUH

"Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm", 2013
Dave Dice, Yossi Lev, Yujie Liu, Victor Luchangco, Mark Moir

20 years later.
KSUH implementation contained a bug connected with manual memory management.
Was reproduced on real hardware.

Java Path Finder:

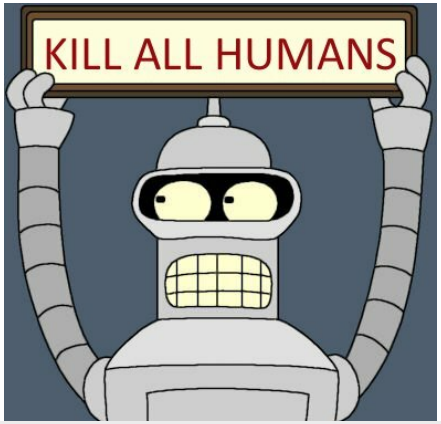
- 250 lines of code for impl.
- 90 lines for test.
- Analysis found a problem in 7 minutes.
- It took 700 Mb RAM.

# KSUH

"Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm", 2013
Dave Dice, Yossi Lev, Yujie Liu, Victor Luchangco, Mark Moir

20 years later.
KSUH implementation contained a bug connected with manual memory management.
Was reproduced on real hardware.

Java Path Finder:



# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

Spin lock

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
  }
}
```

```
void unlock() {
  boolean unlocked = cas(1, 0);
  assert unlocked;
}
```

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

Spin lock

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
  }
}
```

```
void unlock() {
  boolean unlocked = cas(1, 0);
  assert unlocked;
}
```

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

Spin lock

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
  }
}
```

```
void unlock() {
  boolean unlocked = cas(1, 0);
  assert unlocked;
}
```

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

Spin lock

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
  }
}
```

```
void unlock() {
  boolean unlocked = cas(1, 0);
  assert unlocked;
}
```

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

Spin lock

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
  }
}
```

```
void unlock() {
  boolean unlocked = cas(1, 0);
  assert unlocked;
}
```

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

Spin lock

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
  }
}
```

```
void unlock() {
  boolean unlocked = cas(1, 0);
  assert unlocked;
}
```

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

Spin lock

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
  }
}
```

```
void unlock() {
  boolean unlocked = cas(1, 0);
  assert unlocked;
}
```

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

Spin lock

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
  }
}
```

```
void unlock() {
  boolean unlocked = cas(1, 0);
  assert unlocked;
}
```

```
[SEVERE] JPF out of memory
```

# Grain of salt

Model checking analyse **model** of your program by explicitly enumerating program **states**.
What if there are infinite number of states?

Spin lock

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
  }
}
```

```
void unlock() {
  boolean unlocked = cas(1, 0);
  assert unlocked;
}
```

```
[SEVERE] JPF out of memory
```

```
listener=gov.nasa.jpf.listener.EndlessLoopDetector
```

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not change any program state.

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not change any program state.
Help them to «advance» any other thread

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not change any program state.

Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not change any program state.

Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not change any program state.
Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

Other spells I have used over different editions of JPF:

```
staticAtomicInteger.atomicLoad();
```

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not change any program state.
Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

Other spells I have used over different editions of JPF:

```
  staticAtomicInteger.atomicLoad();

  synchronized(STATIC_GUARD) {
    STATIC_GUARD.wait(1);
  }
```

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not change any program state.

Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

Other spells I have used over different editions of JPF:

```
  staticAtomicInteger.atomicLoad();

  synchronized(STATIC_GUARD) {
    STATIC_GUARD.wait(1);
  }
```

Warning: Sometimes they just obsure the «infinite states» problem.

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not check any program state.

Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not check any program state.
Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

Theory: we lost obstruction-freedom property

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not check any program state.

Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

Theory: we lost obstruction-freedom property

Production system: unfortunate scheduling may cost you

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not check any program state.

Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

Theory: we lost obstruction-freedom property

Production system: unfortunate scheduling may cost you

scalability

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not check any program state.
Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

Theory: we lost obstruction-freedom property
Production system: unfortunate scheduling may cost you

scalability/throughput

# Taming spin loops

Usually, model checkers are smart enough to see that spin loop iteration does not check any program state.
Help them to «advance» any other thread

```
void lock() {
  boolean locked = false;
  while (true) {
    locked = cas(0, 1);
    if (locked) {
      break;
    }
    Thread.yield(); // allow others to progress
  }
}
```

Theory: we lost obstruction-freedom property
Production system: unfortunate scheduling may cost you

scalability/throughput/vacation

# Limitations

Problem size

- 10 KLOC
- 3-4 «concurrent» threads
- 3-6 hours per one test case
- 10-20 GB RAM
- JPF itself is single-threaded

Keep in mind

- Cannot prove your program is correct
- Requires `asserts`
- Needs manually-written tests
- Spinloops may be a problem
- Cannot handle all pecularities of JMM

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | | | | | | |

https://github.com/openjdk/jcstress

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | | | | | |

`https://github.com/openjdk/jcstress`

Experimental harness and a suite of tests to aid the research in the correctness of concurrency support in the JVM, class libraries, and hardware.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | | | | |

`https://github.com/openjdk/jcstress`

Experimental harness and a suite of tests to aid the research in the correctness of concurrency support in the JVM, class libraries, and hardware.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | | | |

`https://github.com/openjdk/jcstress`

Experimental harness and a suite of tests to aid the research in the correctness of concurrency support in the JVM, class libraries, and hardware.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | | |

`https://github.com/openjdk/jcstress`

Experimental harness and a suite of tests to aid the research in the correctness of concurrency support in the JVM, class libraries, and hardware.

**Empirical** testing framework – focuses on observable results (OS-, hardware-, JVM-specific etc.)

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | |

`https://github.com/openjdk/jcstress`

Experimental harness and a suite of tests to aid the research in the correctness of concurrency support in the JVM, class libraries, and hardware.

**Empirical** testing framework – focuses on observable results (OS-, hardware-, JVM-specific etc.)

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |

`https://github.com/openjdk/jcstress`

Experimental harness and a suite of tests to aid the research in the correctness of concurrency support in the JVM, class libraries, and hardware.

**Empirical** testing framework – focuses on observable results (OS-, hardware-, JVM-specific etc.)

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |

`https://github.com/openjdk/jcstress`

Experimental harness and a suite of tests to aid the research in the correctness of concurrency support in the JVM, class libraries, and hardware.

**Empirical** testing framework – focuses on observable results (OS-, hardware-, JVM-specific etc.)

Aleksey Shipilev. Workshop: Java Concurrency Stress: `https://www.youtube.com/watch?v=koU38cczBy8`

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | | | | | |

`https://github.com/Kotlin/kotlinx-lincheck`

User should **specify** properties of operations. `lincheck` will generate test scenarios, execute them in concurrent environment and check that results are correct.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |

`https://github.com/Kotlin/kotlinx-lincheck`

User should **specify** properties of operations. `lincheck` will generate test scenarios, execute them in concurrent environment and check that results are correct.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |

`https://github.com/Kotlin/kotlinx-lincheck`

User should **specify** properties of operations. `lincheck` will generate test scenarios, execute them in concurrent environment and check that results are correct.

Nikita Koval. Lin-Check: Testing concurrent data structures in Java
`https://2019.hydraconf.com/2019/talks/1wyyozh4zrh0da1ctbcynm`

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | | | | | |

`https://github.com/javapathfinder/jpf-core/wiki/What-is-JPF`
The JPF core is a Virtual Machine (VM) for Java bytecode.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | + | | | | |

`https://github.com/javapathfinder/jpf-core/wiki/What-is-JPF`
The JPF core is a Virtual Machine (VM) for Java bytecode.

JPF (theoretically) explores all potential executions in a systematic way.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | + | - | | | |

`https://github.com/javapathfinder/jpf-core/wiki/What-is-JPF`
The JPF core is a Virtual Machine (VM) for Java bytecode.

JPF (theoretically) explores all potential executions in a systematic way.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | + | - | - | | |

`https://github.com/javapathfinder/jpf-core/wiki/What-is-JPF`
The JPF core is a Virtual Machine (VM) for Java bytecode.

JPF (theoretically) explores all potential executions in a systematic way.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | + | - | - | + | |

`https://github.com/javapathfinder/jpf-core/wiki/What-is-JPF`
The JPF core is a Virtual Machine (VM) for Java bytecode.

JPF (theoretically) explores all potential executions in a systematic way.

JPF is a tool for mission critical applications, where failure is not an option.
No surprise it was started by NASA.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | + | - | - | + | Small |

`https://github.com/javapathfinder/jpf-core/wiki/What-is-JPF`
The JPF core is a Virtual Machine (VM) for Java bytecode.

JPF (theoretically) explores all potential executions in a systematic way.

JPF is a tool for mission critical applications, where failure is not an option.
No surprise it was started by NASA.

After all, it is a heavyweight tool, not a quick and simple bug finder

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | + | - | - | + | Small |
| $\text{LinCheck}_{MC}$ | | | | | | |

`https://github.com/Kotlin/kotlinx-lincheck#model-checking`

Model checking mode that works under the sequentially consistent memory model. It studies all possible schedules with a bounded number of context switches by fully controlling the execution and putting context switches in different locations in threads.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | + | - | - | + | Small |
| $\text{LinCheck}_{MC}$ | auto | + | - | - | + | Tiny |

`https://github.com/Kotlin/kotlinx-lincheck#model-checking`

Model checking mode that works under the sequentially consistent memory model. It studies all possible schedules with a bounded number of context switches by fully controlling the execution and putting context switches in different locations in threads.

# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | + | - | - | + | Small |
| $\text{LinCheck}_{MC}$ | auto | + | - | - | + | Tiny |

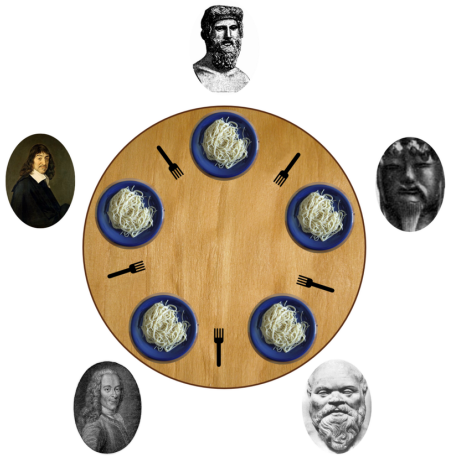
# Bigger picture

| | Tests | Sched | WeakMM | CPU aware | Error | Problem size |
|---|---|---|---|---|---|---|
| JCStress | manual | - | + | + | - | Large |
| LinCheck | auto | - | + | + | - | Large |
| JPF | manual | + | - | - | + | Small |
| $\text{LinCheck}_{MC}$ | auto | + | - | - | + | Tiny |

Select tool that suites your problem

# Give it a try

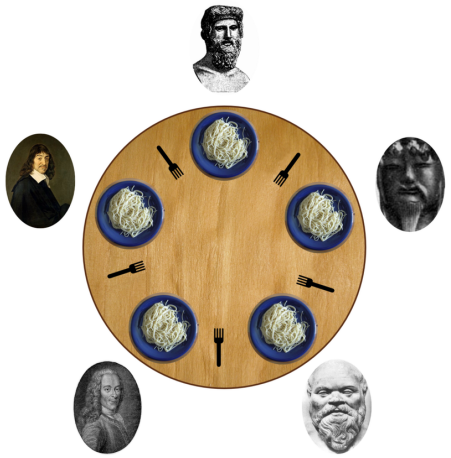
https://github.com/javapathfinder/jpf-core
https://github.com/javapathfinder/jpf-core/tree/master/src/examples
https://github.com/javapathfinder/jpf-core/blob/master/src/examples/DiningPhil.java



Go and find a concurrency bug in standard library.

# Give it a try

https://github.com/javapathfinder/jpf-core
https://github.com/javapathfinder/jpf-core/tree/master/src/examples
https://github.com/javapathfinder/jpf-core/blob/master/src/examples/DiningPhil.java



Go and find a concurrency bug in standard library.
What if? ...

Thank you