Неблокирующая синхронизация

Владимир Озеров

GridGain

План

- Мотивация
- Структура
- Производительность

Кто?

План

- Мотивация
- Структура
- Производительность

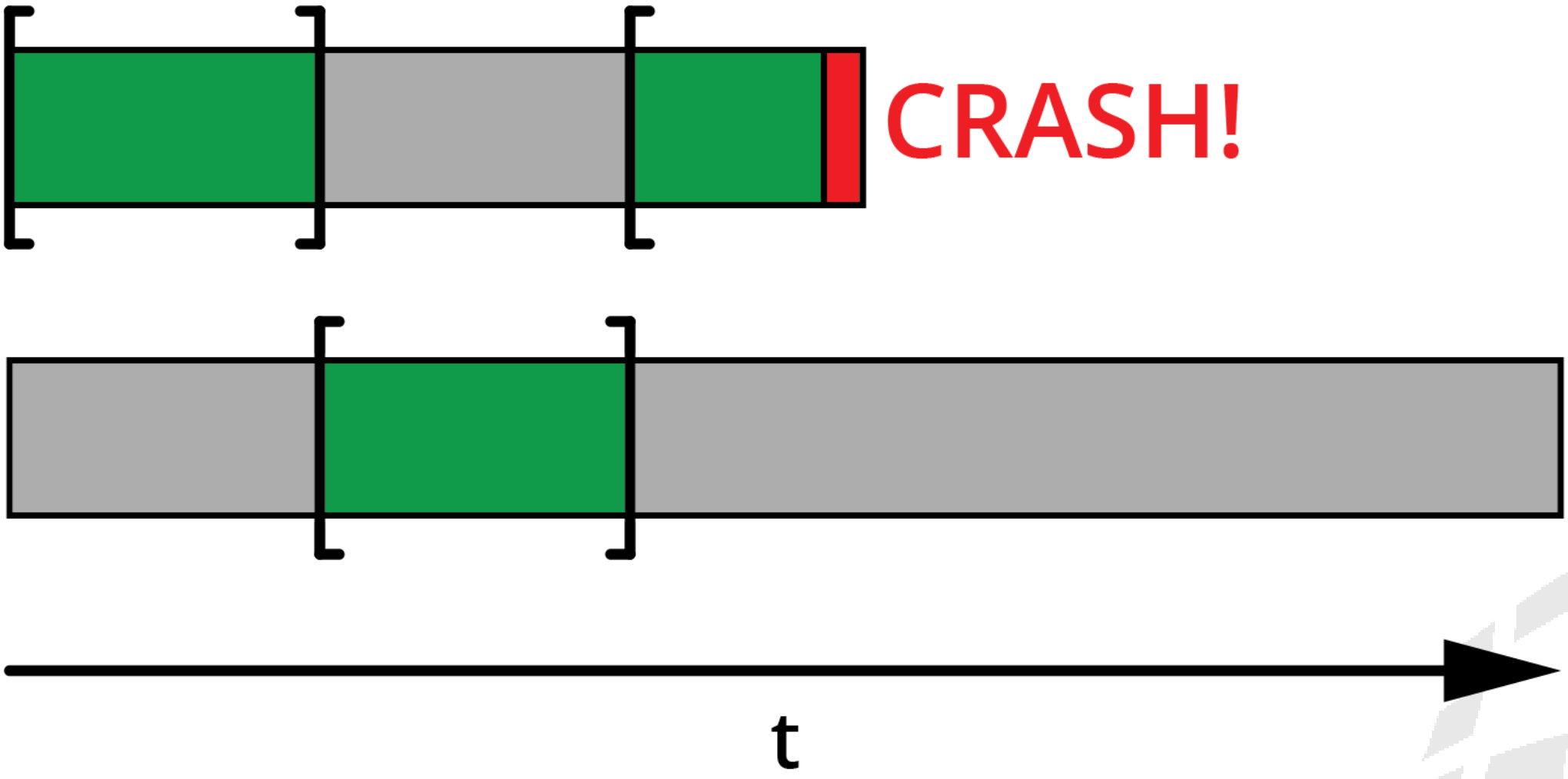
Мотивация: определения

- BLOCKING – один поток может не давать работать другим потокам неограниченно долго

Мотивация: определения

- BLOCKING – один поток может не давать работать другим потокам неограниченно долго
- **NON**-BLOCKING = !BLOCKING

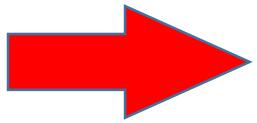
Мотивация: безопасность



Мотивация: безопасность в Java

```
1: lock.lock();
2:
3: try {
4:     doSomething();
5: } finally {
6:     lock.unlock();
7: }
```

Мотивация: безопасность в Java



```
1: lock.lock();
2:
3: try {
4:     doSomething();
5: } finally {
6:     lock.unlock();
7: }
```

Stack overflow!

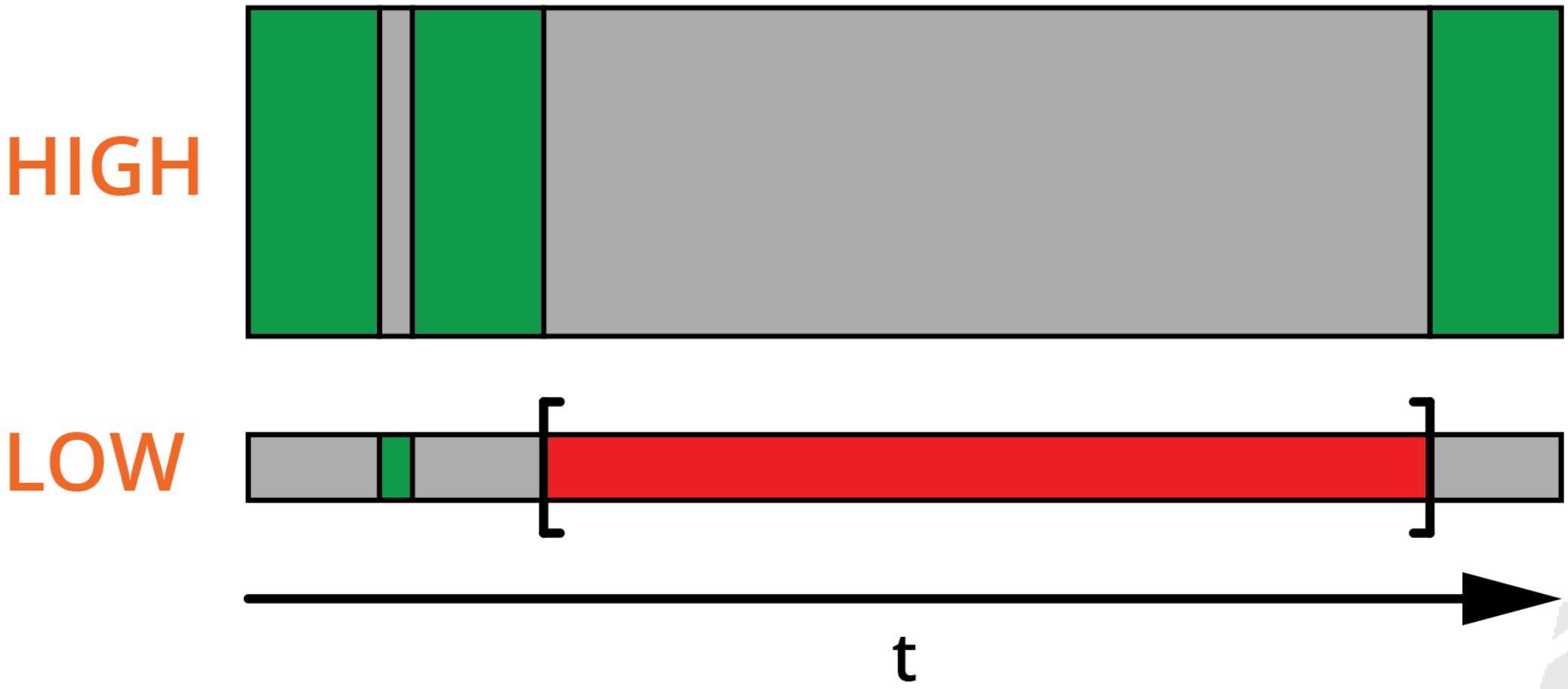
http://openjdk.java.net/jeps/270

Мотивация: безопасность в .NET :-)

```
 1: try {
 2: } finally {
 3:     lock.lock();
 4:
 5:     try {
 6:         doSomething();
 7:     } finally {
 8:         lock.unlock();
 9:     }
10: }
```

MSDN: “Unexecuted finally blocks are executed before the thread is aborted.”

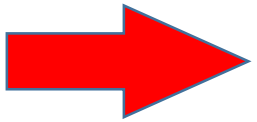
Мотивация: инверсия приоритетов



Мотивация: инверсия приоритетов

```
1: Connection conn = pool.acquire();
2:
3: try {
4:     execute(conn);
5: } finally {
6:     pool.release(conn);
7: }
```

Мотивация: инверсия приоритетов

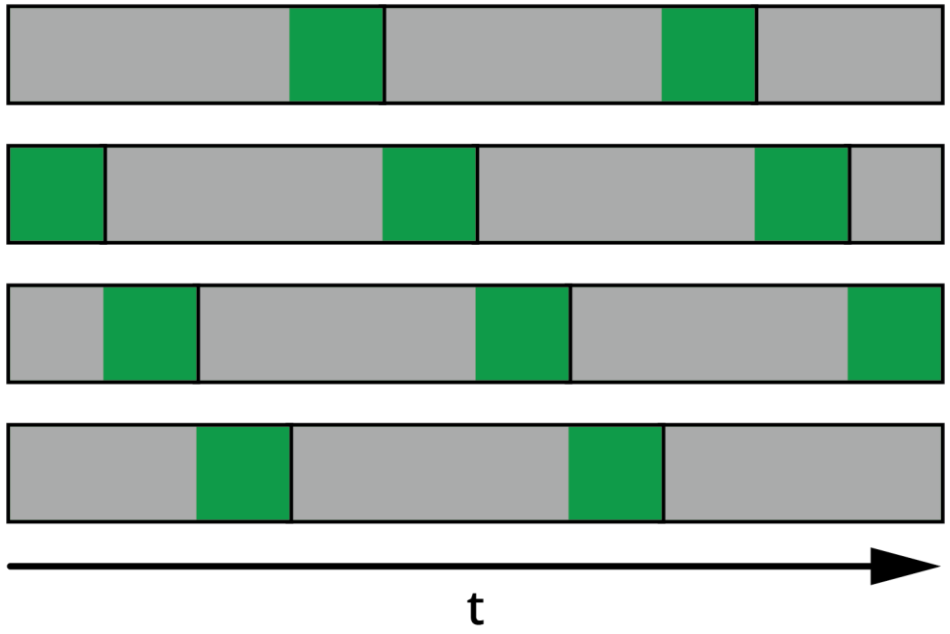


```
1: Connection conn = pool.tryAcquire(5000L);
2:
3: if (conn != null) {
4:     ...
5: } else {
6:     conn = Connection.create(...);
7:     ...
8: }
```

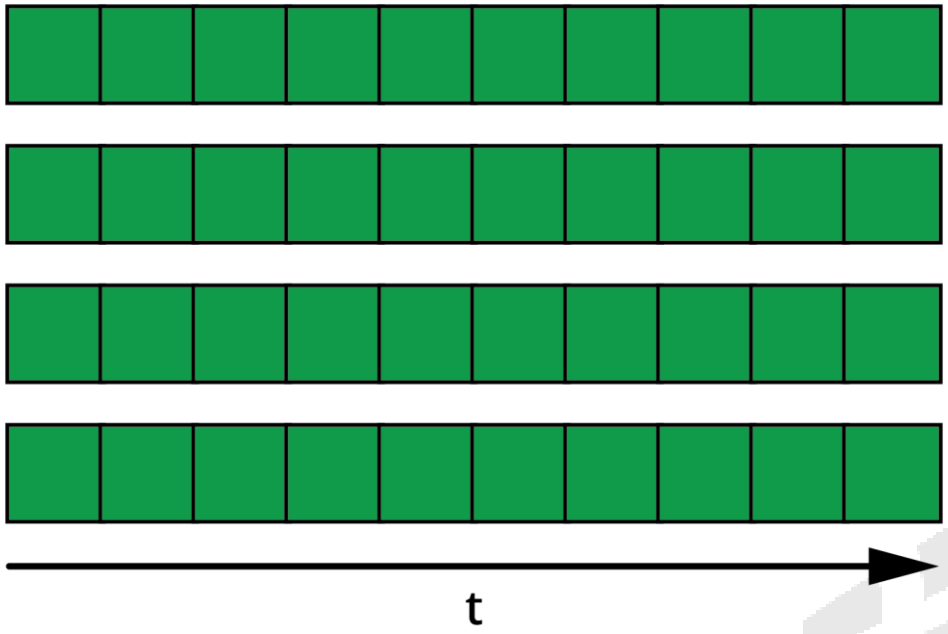
Мотивация: deadlocks

```
"ContainersLauncher #27":
  at AllocatorPerContext.getLocalPathForWrite(LocalDirAllocator.java:331)
  - waiting to lock <B> (AllocatorPerContext)

  [3 other calls]

  at LocalDirsHandlerService.getLocalPathForWrite(LocalDirsHandlerService.java:262)
  - locked <A> (Configuration)

"New I/O server worker #1-3":
  at Configuration.getProps(Configuration.java:1373)
  - waiting to lock <A> (Configuration)

  [7 other calls]

  at AllocatorPerContext.getLocalPathToRead(LocalDirAllocator.java:425)
  - locked <B> (AllocatorPerContext)
```

Мотивация: производительность



BLOCKING



IDEAL

Мотивация: итоги

Безопасность	Low
Инверсия приоритетов	Low
Deadlock	Low
Производительность	High

Мотивация: итоги

Безопасность	Low
Инверсия приоритетов	Low
Deadlock	Low
Производительность	High
Удобство	High

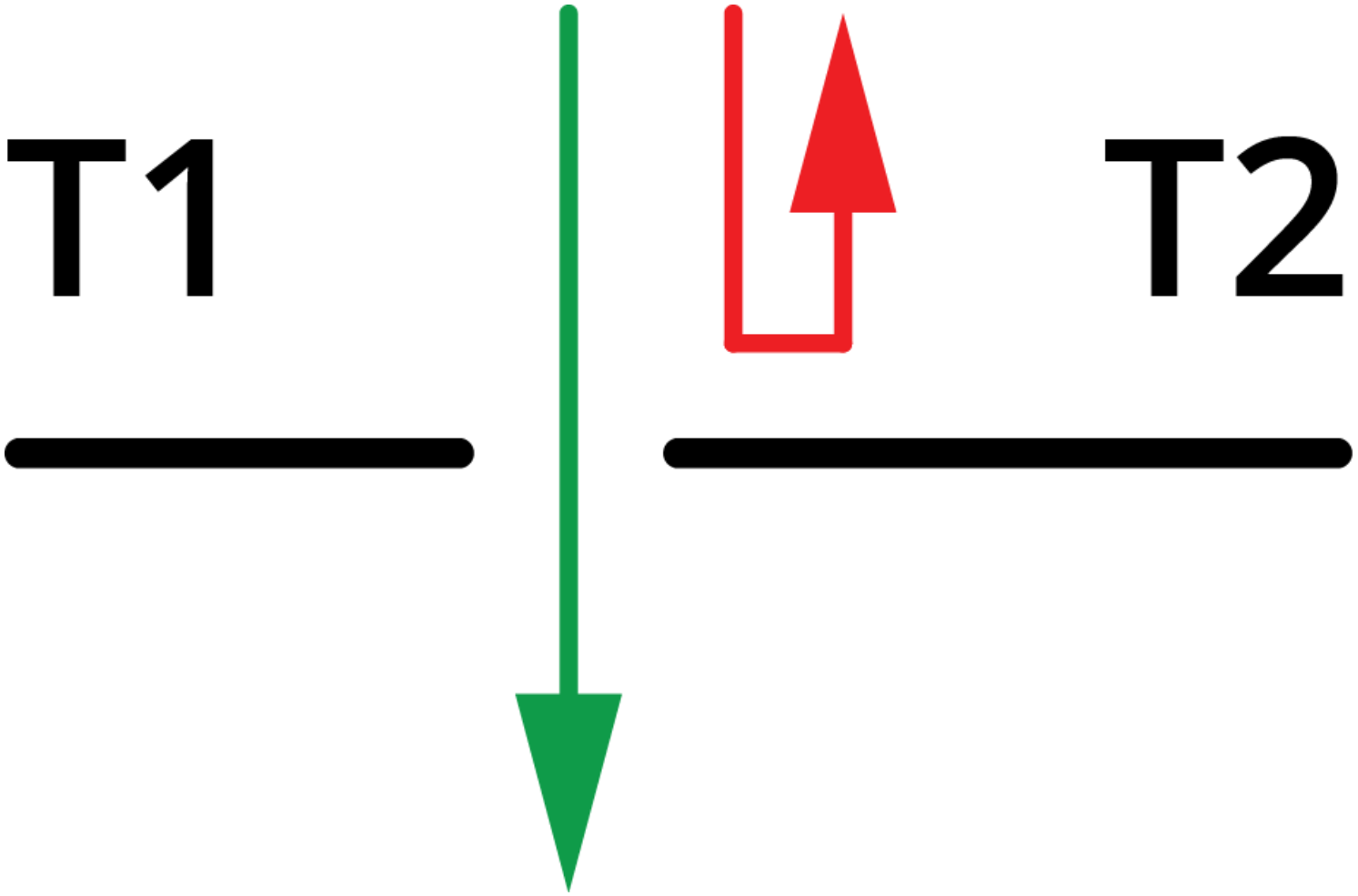
План

- Мотивация
- Структура
- Производительность

Структура: compare-and-set

```
bool CAS(T* address, T expected, T new) {
    if (*address == expected) {
        *address = new;

        return true;
    } else
        return false;
}
```

Структура: compare-and-set



Структура: atomics

```
1: synchronized (...) {
2:     if (starting)
3:         throw new AlreadyStartedException();
4:
5:     starting = true;
6: }
7:
8: doStart();
```

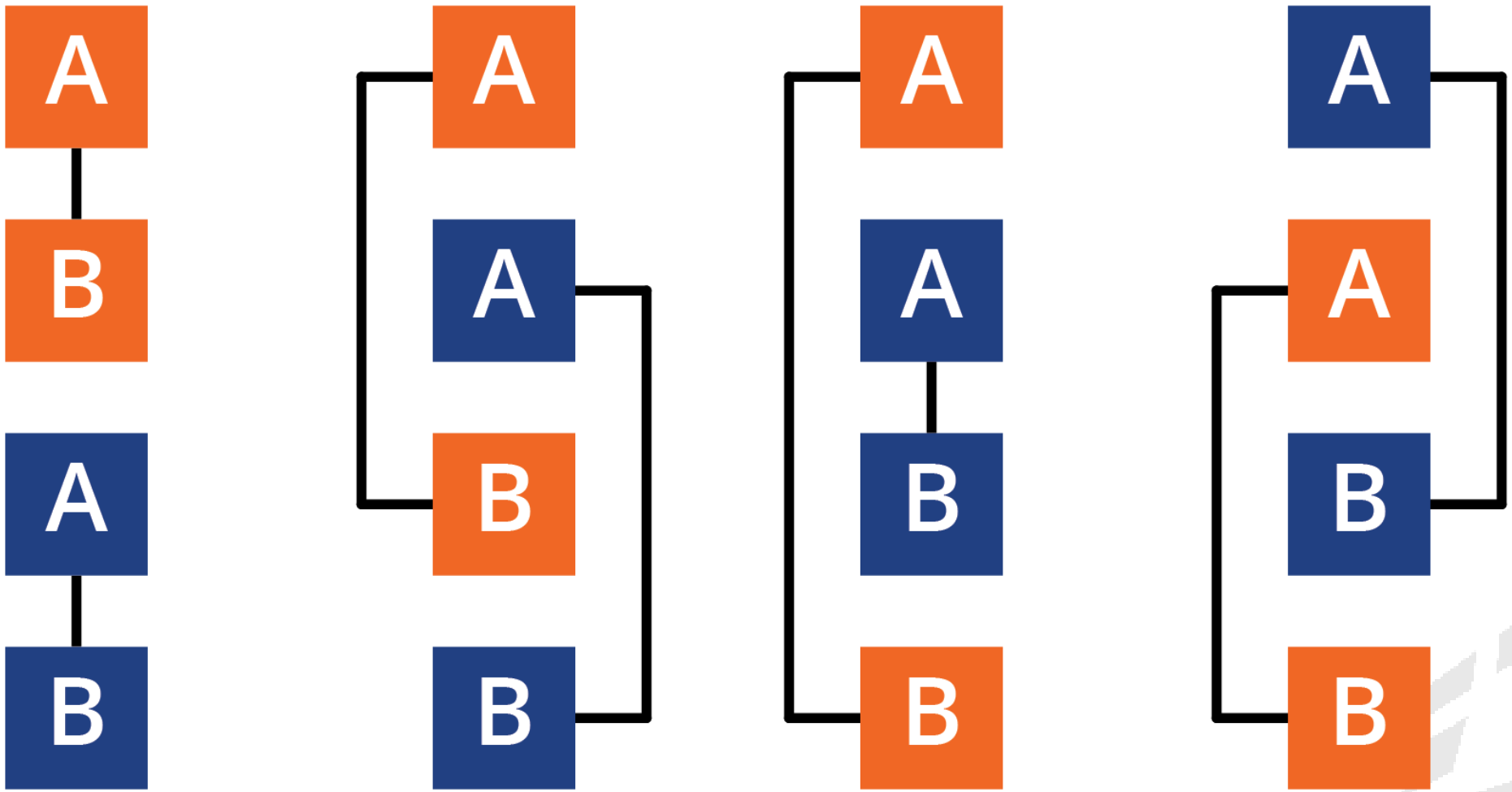
Структура: atomics

```
1: synchronized (...) {
2:     if (starting)
3:         throw new AlreadyStartedException();
4:
5:     starting = true;
6: }
7:
8: doStart();
```

```
1: if (!starting.compareAndSet(false, true))
2:     throw new AlreadyStartedException();
3:
4: doStart();
```

Структура: races

Структура: races



Упрощаем: меньше shared state

```
1: volatile int a;
2: volatile int b;
```

Упрощаем: меньше shared state

```
1: volatile int a;
2: volatile int b;
```

```
1: class State {
2:     final int a;
3:     final int b;
4: }
5:
6: volatile State state;
```

Упрощаем: обратно в blocking

```
 1: Node node = new Node(key, val);
 2:
 3: if (table.compareAndSet(index, null, node))
 4:     return null;
 5: else {
 6:     Node oldNode = table.get(index);
 7:
 8:     synchronized (oldNode) {
 9:         ...
10:     }
11: }
```

ConcurrentHashMap

Упрощаем: меньше гарантий

```
1: private final ConcurrentLinkedQueue queue;
2: private final LongAccumulator size;
3:
4: public boolean offer(E e) {
5:     size.accumulate(1L);
6:
7:     return queue.offer(e);
8: }
```

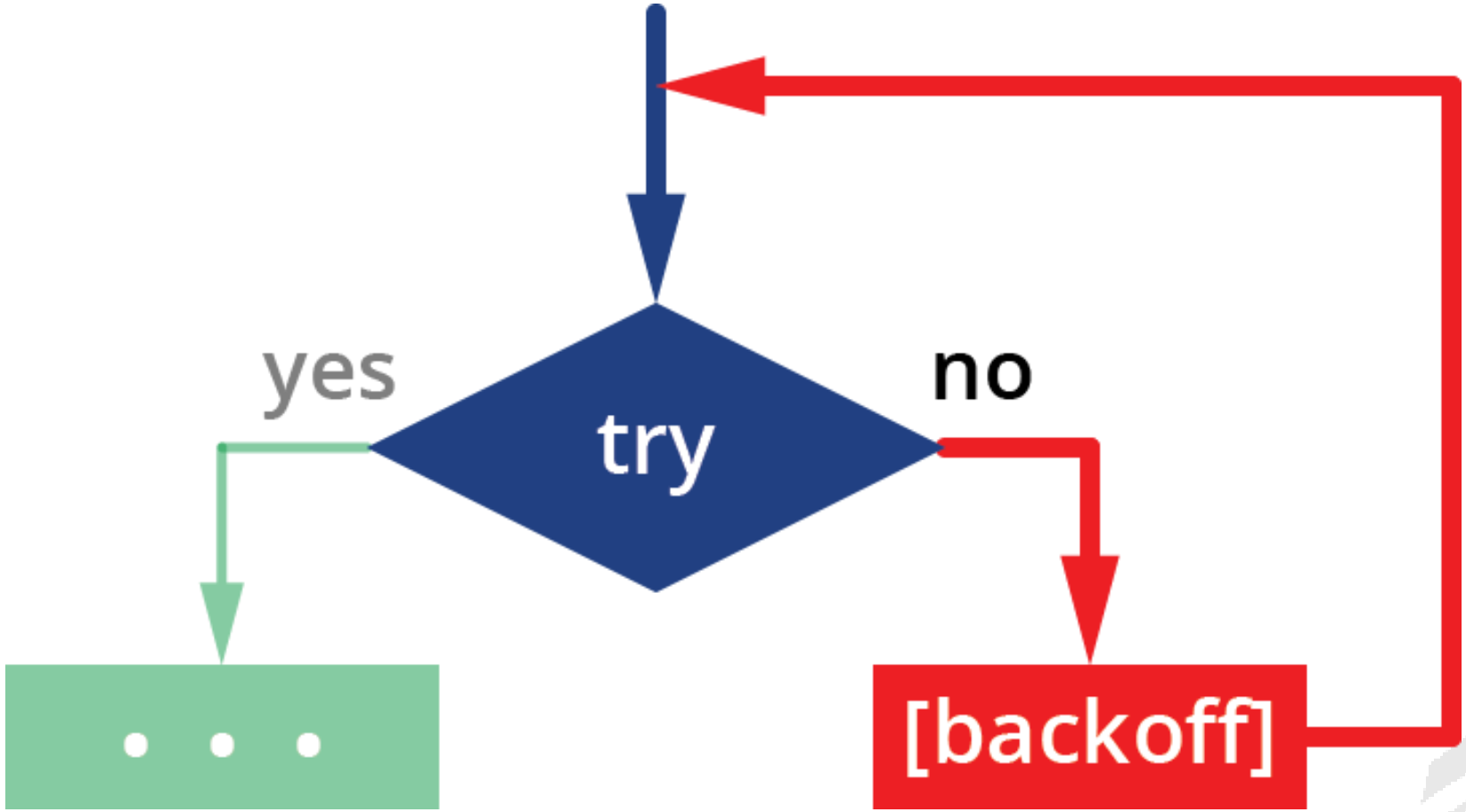
Упрощаем: меньше гарантий

```
1: private final ConcurrentLinkedQueue queue;
2: private final LongAccumulator size;
3:
4: public boolean offer(E e) {
5:     size.accumulate(1L);
6:
7:     return queue.offer(e);
8: }
```

```
assert q.size() > 0 && q.peek() != null;
```

FAIL!

Структура: lock-free



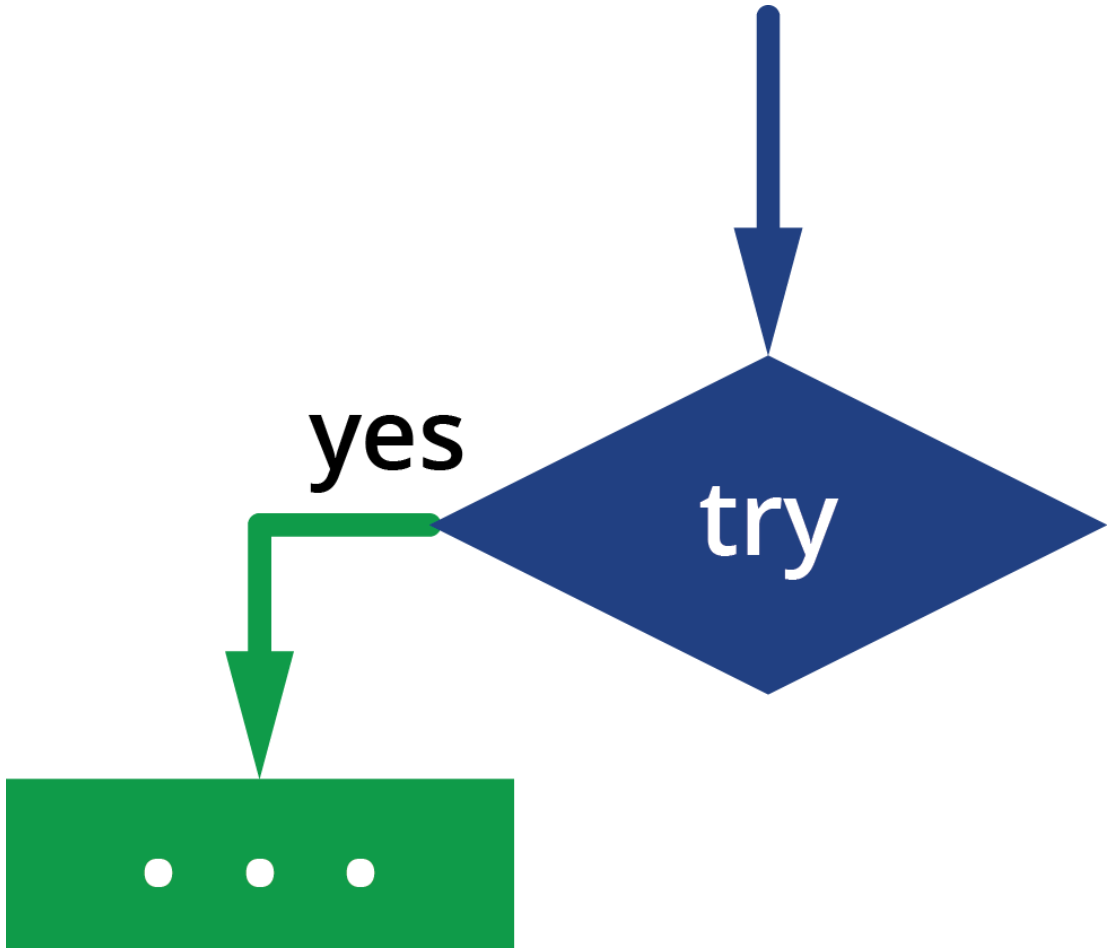
Пример: busy lock

```
1: class BusyLock {
2:     boolean tryAcquire();
3:     void release();
4:
5:     void block();
7: }
```

Структура: lock-free

```
final AtomicInteger state = new AtomicInteger();

boolean tryAcquire() {
    while (true) {
        int state0 = state.get();

        if (isBlocked(state0))
            return false;

        if (state.compareAndSet(state0, state0 + 1))
            return true;
    }
}
```

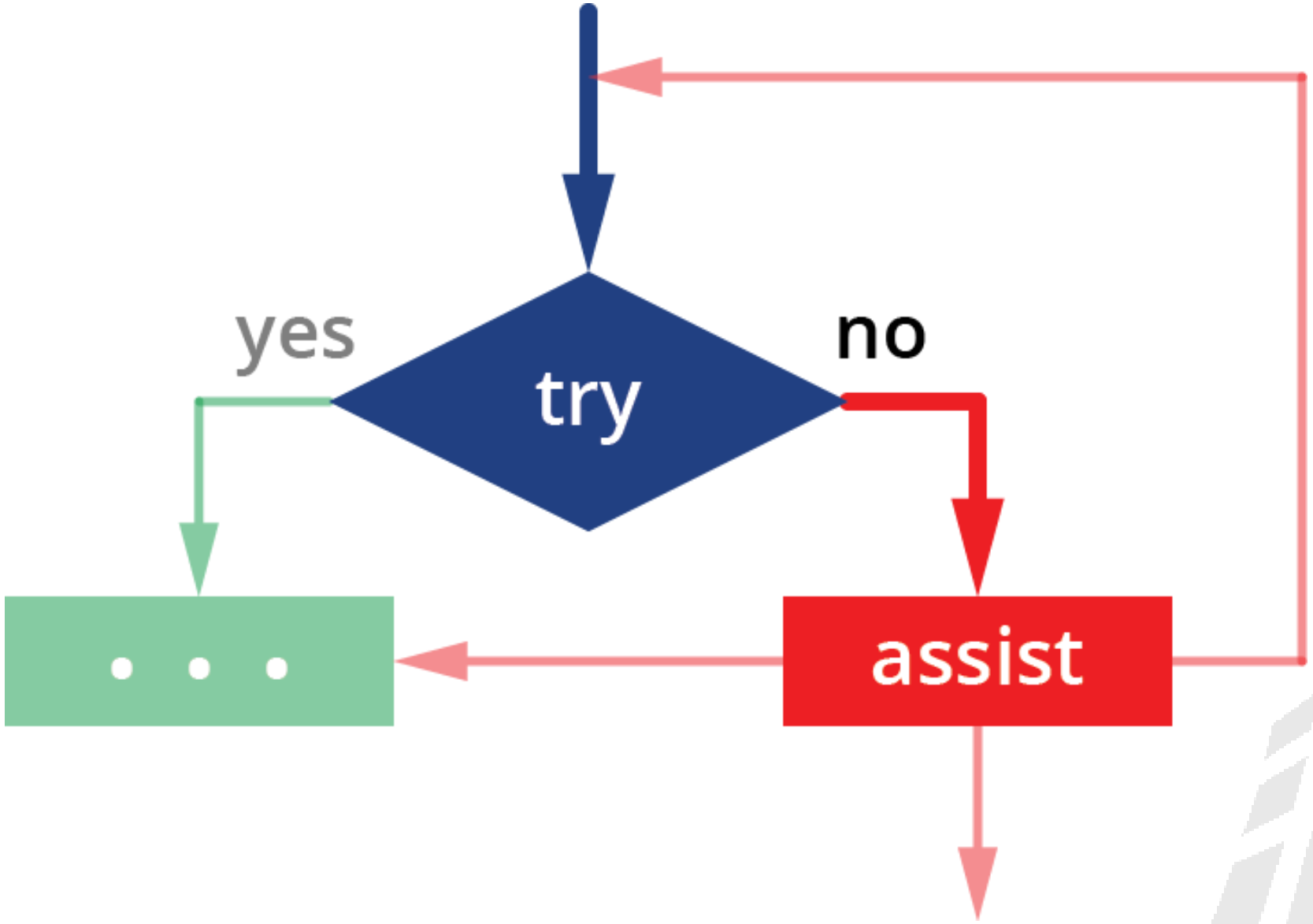
Структура: wait-free



Структура: wait-free

```
 1: final AtomicInteger acquired = new AtomicInteger();
 2: volatile boolean blocked;
 3:
 4: boolean tryAcquire() {
 5:     acquired.incrementAndGet();
 6:
 7:     if (blocked) {
 8:         acquired.decrementAndGet();
 9:
10:         return false;
11:     }
12:
13:     return true;
14: }
```

Структура: assist



Структура: assist

```
 1: AtomicReference<Batch> batch;
 2:
 3: void addToBatch(Object item) {
 4:     while (true) {
 5:         Batch batch0 = batch.get();
 6:
 7:         if (batch0.tryAdd(item))
 8:             return;
 9:
10:
11:     }
12: }
```

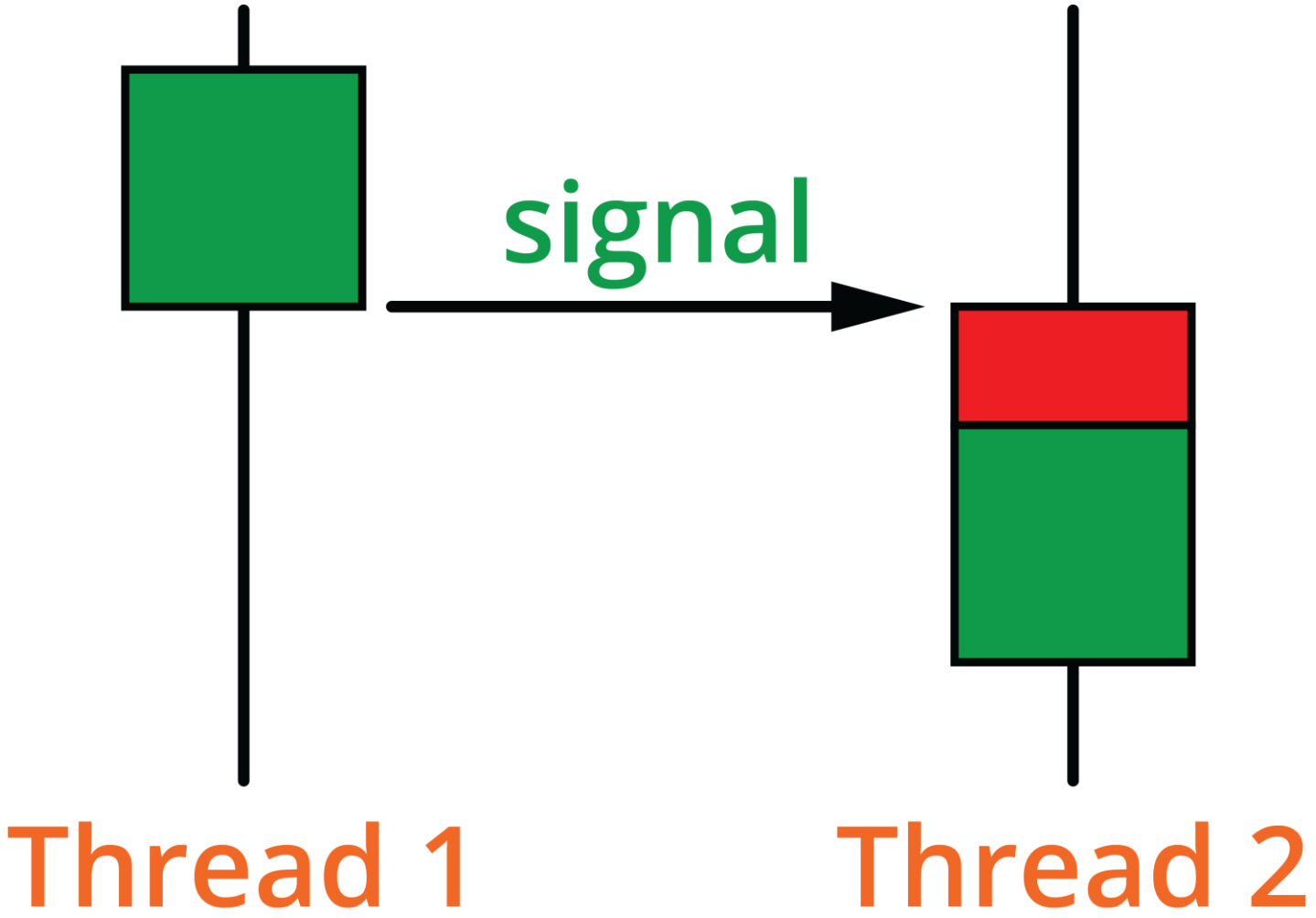
Структура: assist

```
AtomicReference<Batch> batch;

void addToBatch(Object item) {
    while (true) {
        Batch batch0 = batch.get();

        if (batch0.tryAdd(item))
            return;
        else
            batch.compareAndSet(batch0, new Batch());
    }
}
```

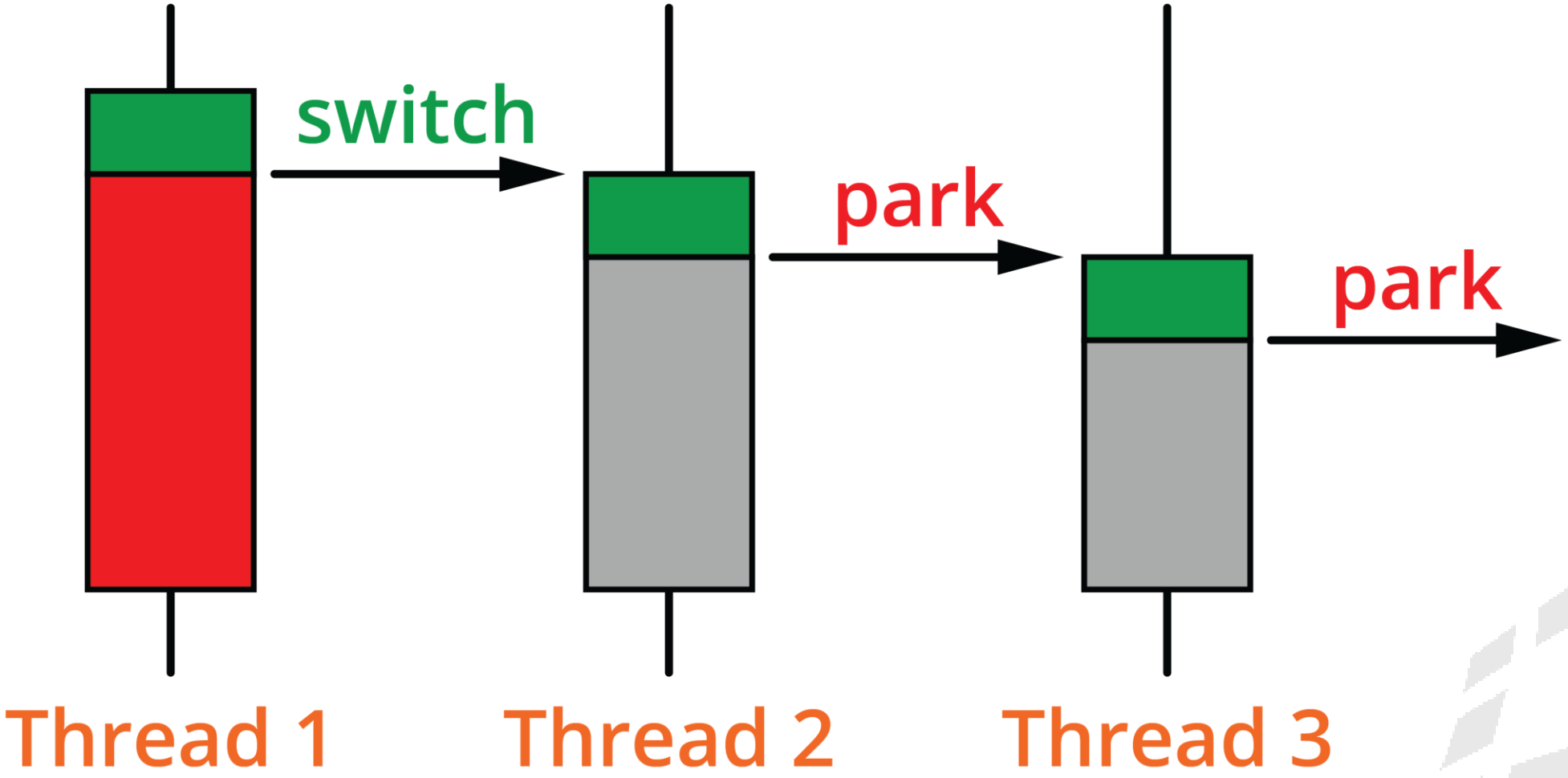
План

- Мотивация
- Структура
- Производительность

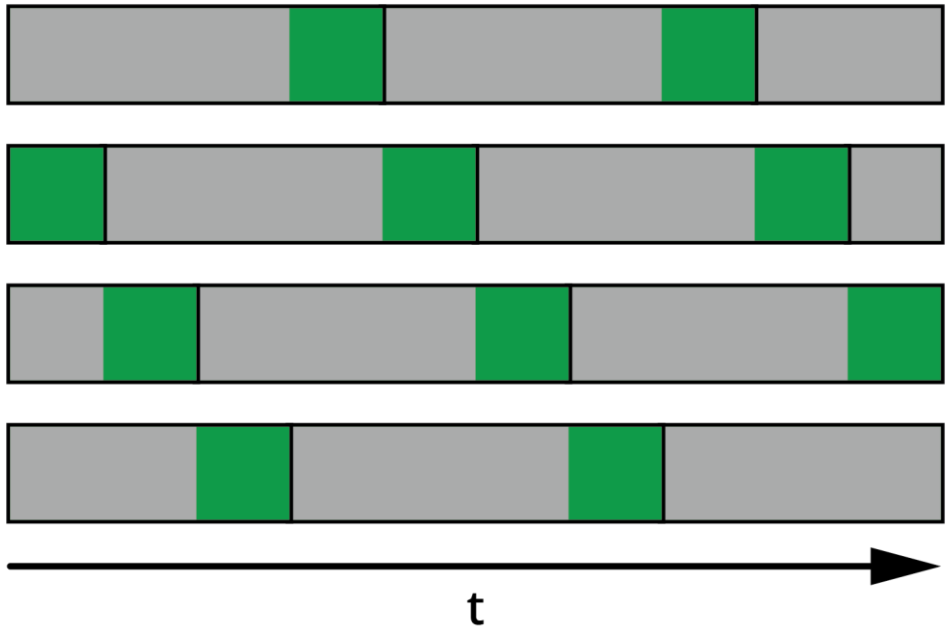
Производительность: latency



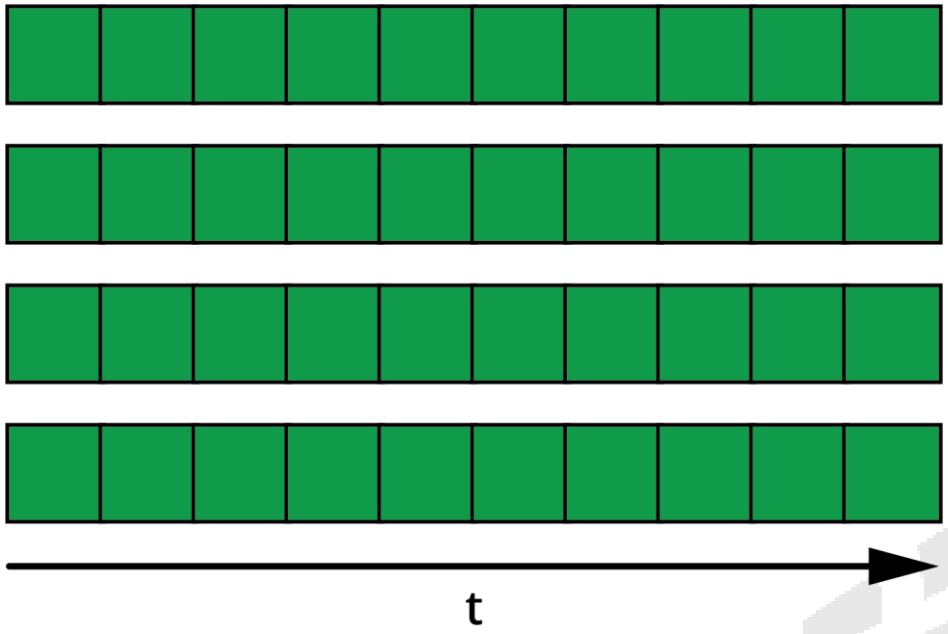
Производительность: lock convoy



Производительность: throughput



BLOCKING

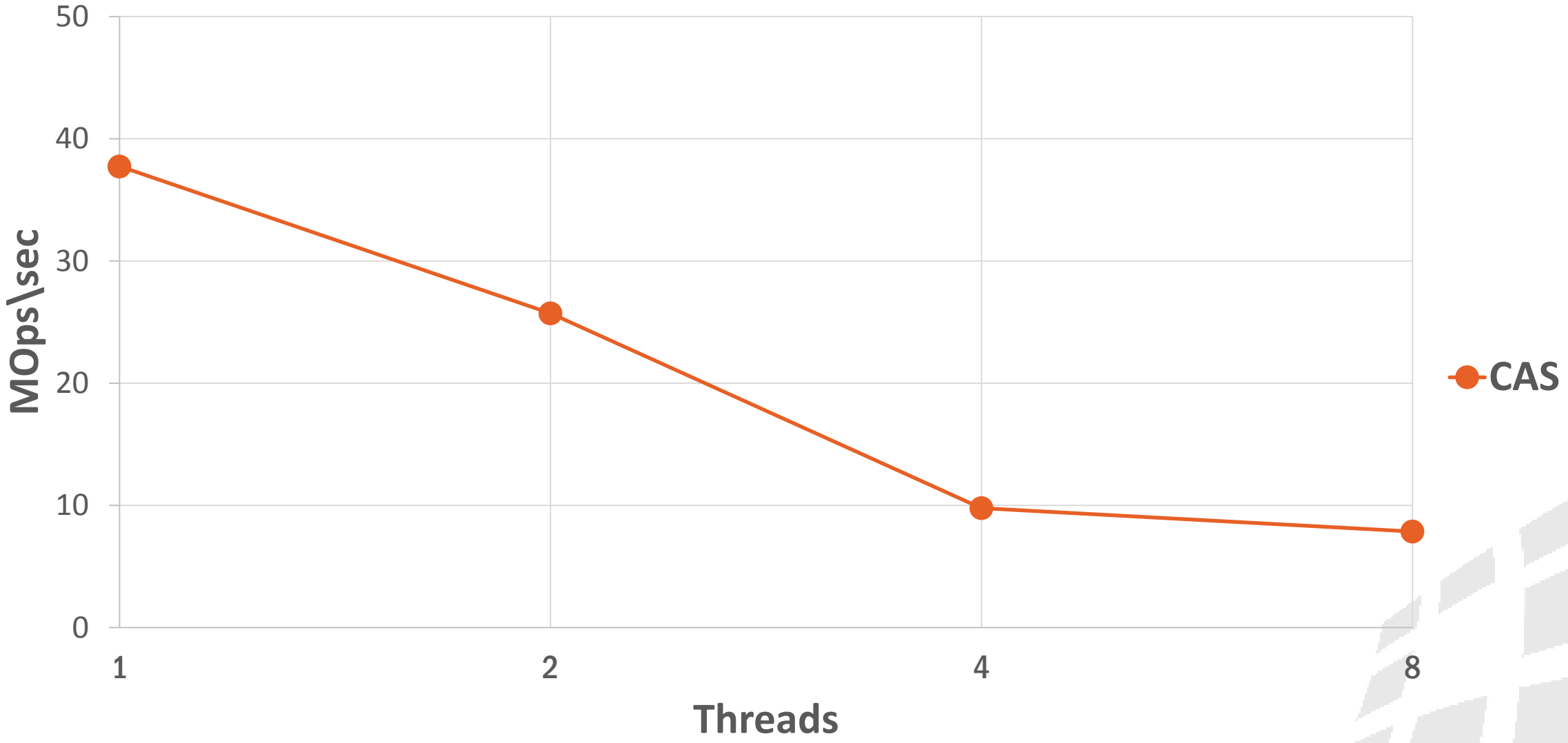


IDEAL

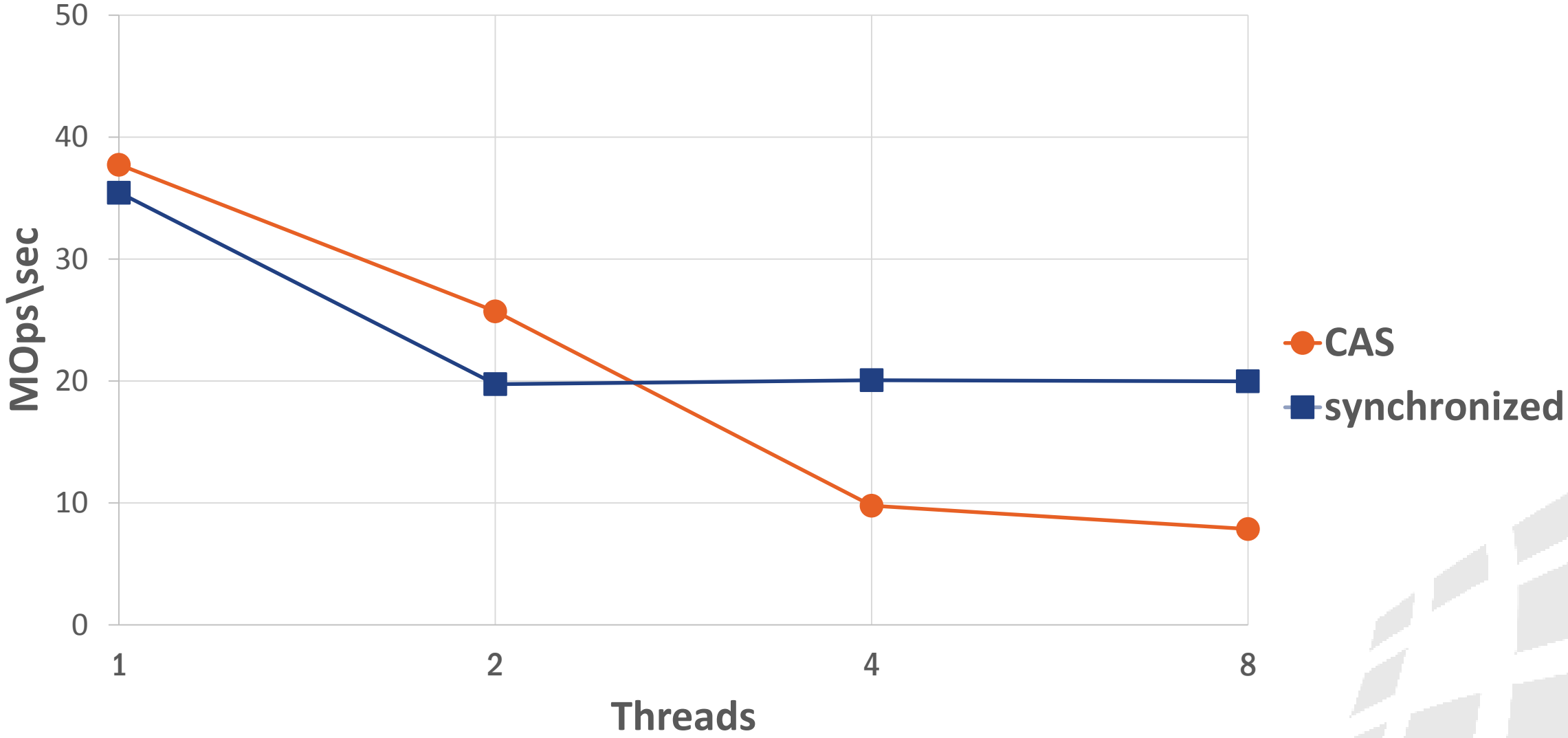
Производительность: throughput

```
 1: BusyLock lock;
 2:
 3: void doBenchmark() {
 4:     if (lock.tryAcquire()) {
 5:         try {
 6:             payload(); // e.g. consumeCPU(x)
 7:         } finally {
 8:             lock.release();
 9:         }
10:     }
11: }
```

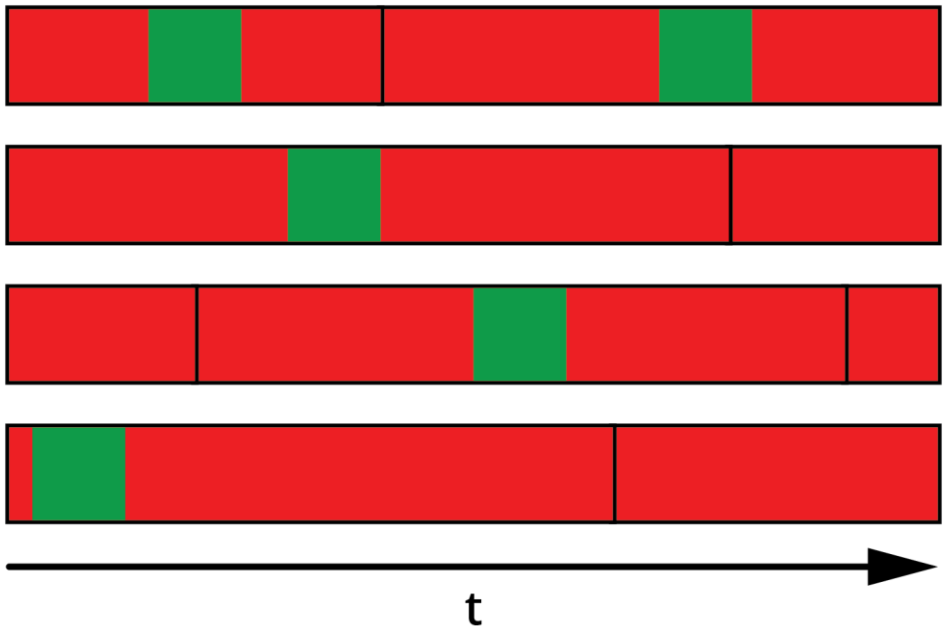
Производительность: CAS



Производительность: synchronized

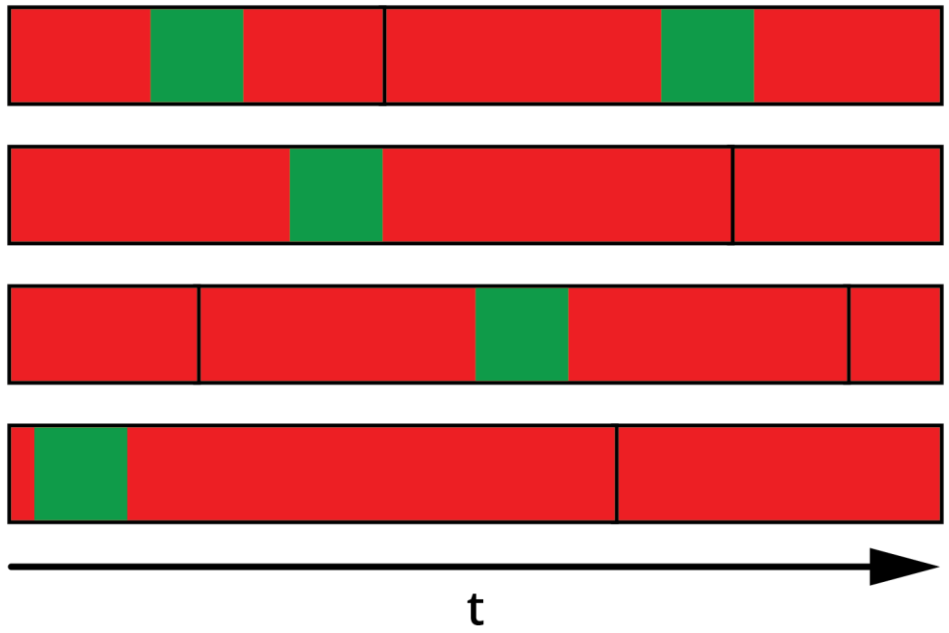


Производительность: contention

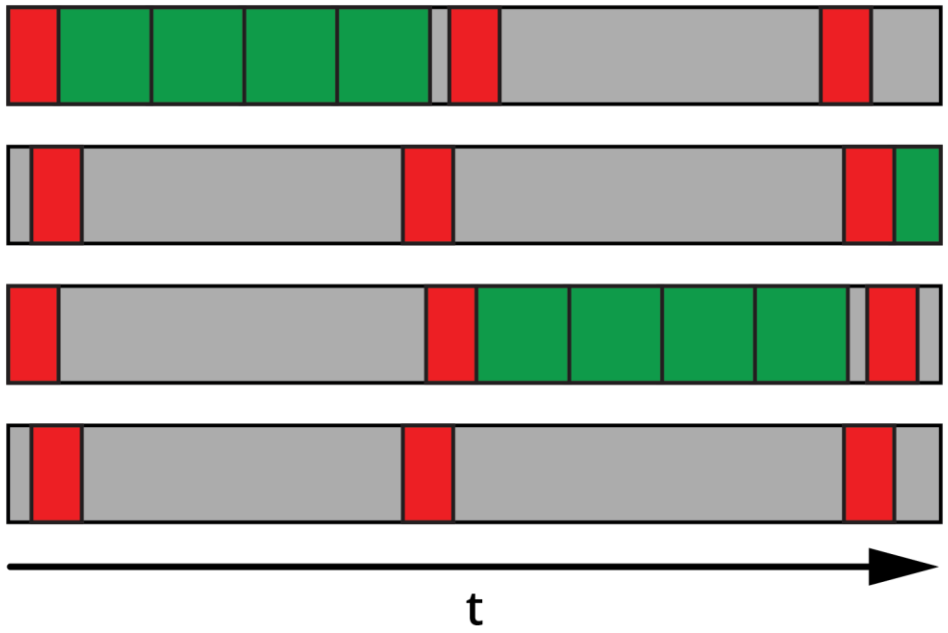


CAS/increment

Производительность: contention



CAS/increment



synchronized

http://mechanical-sympathy.blogspot.ru/2013/01/further-adventures-with-cas.html
http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/xeon-lock-scaling-analysis-paper.pdf

Производительность: batching

```
1: AtomicLong generator;
2:
3: return generator.getAndIncrement();
```

Производительность: batching

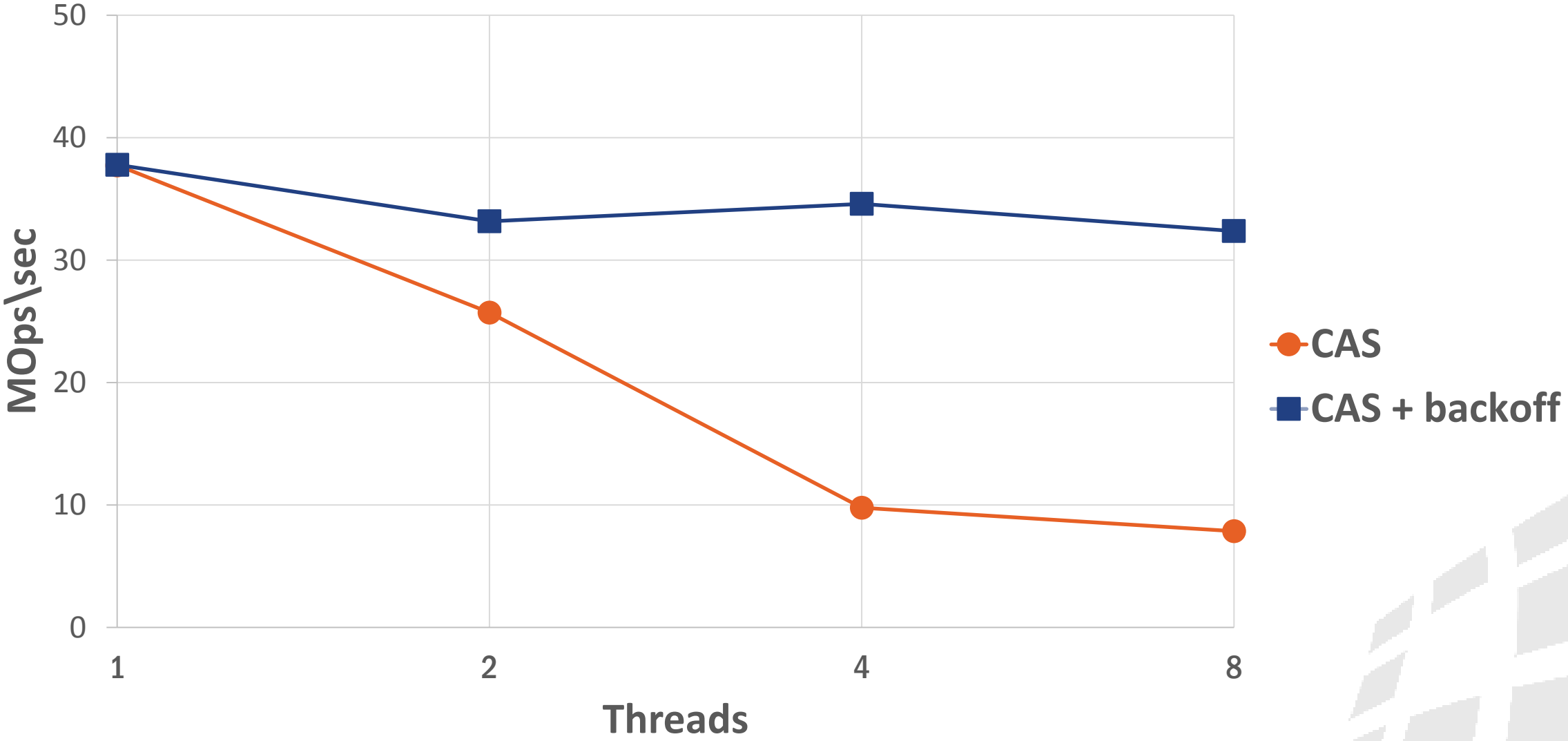
```
1: AtomicLong generator;
2:
3: return generator.getAndIncrement();
```

```
1: AtomicLong generator;
2:
3: return generator.getAndAdd(1000);
```

Производительность: backoff

```
boolean tryAcquire() {
    for (int i = 0;; i++) {
        ...

        if (isSpin(i))
            for (...)
                Runtime.getRuntime().onSpinWait();
        else if (isYield(i))
            Thread.sleep(0);
        else
            Thread.sleep(1);
    }
}
```

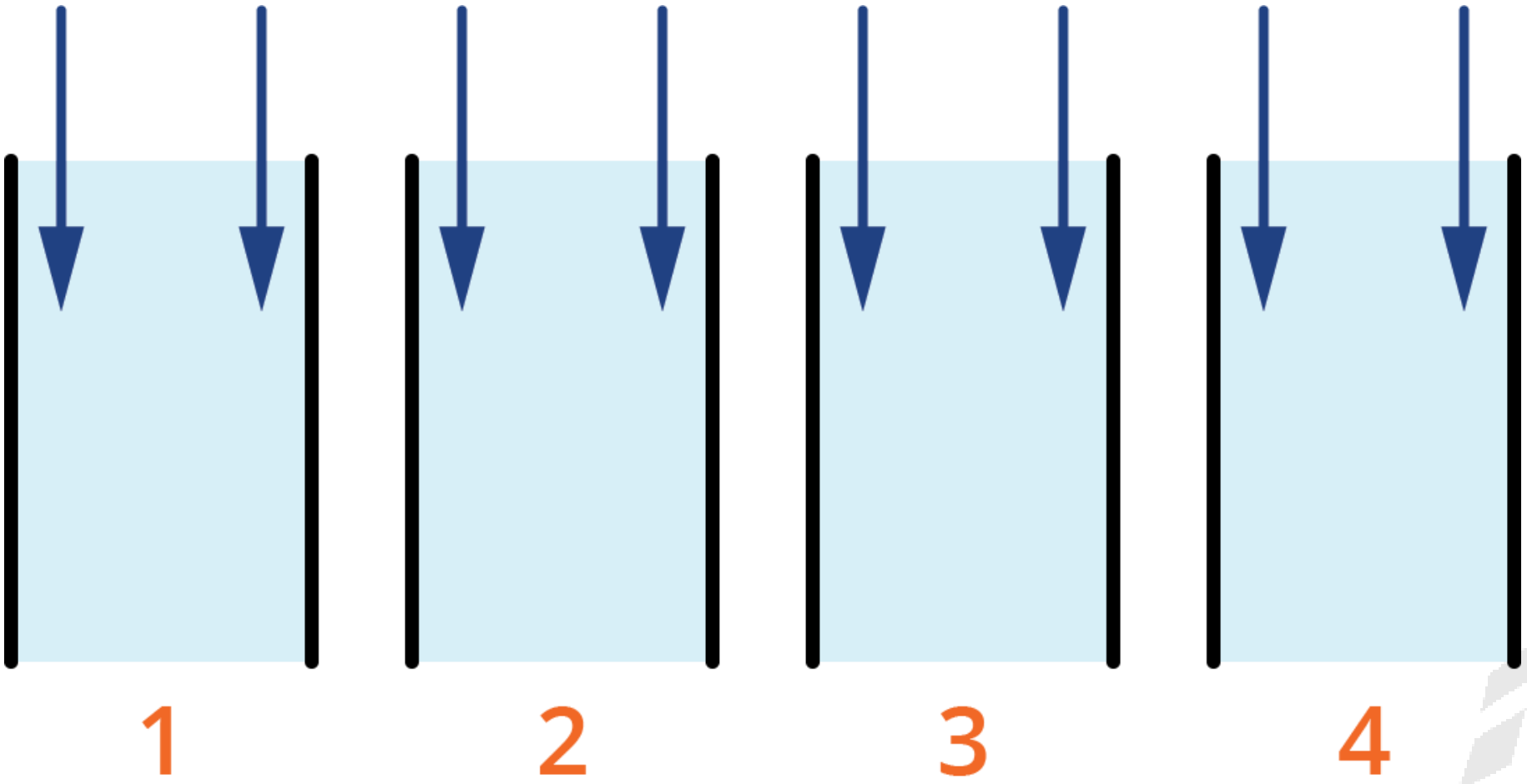
Производительность: backoff



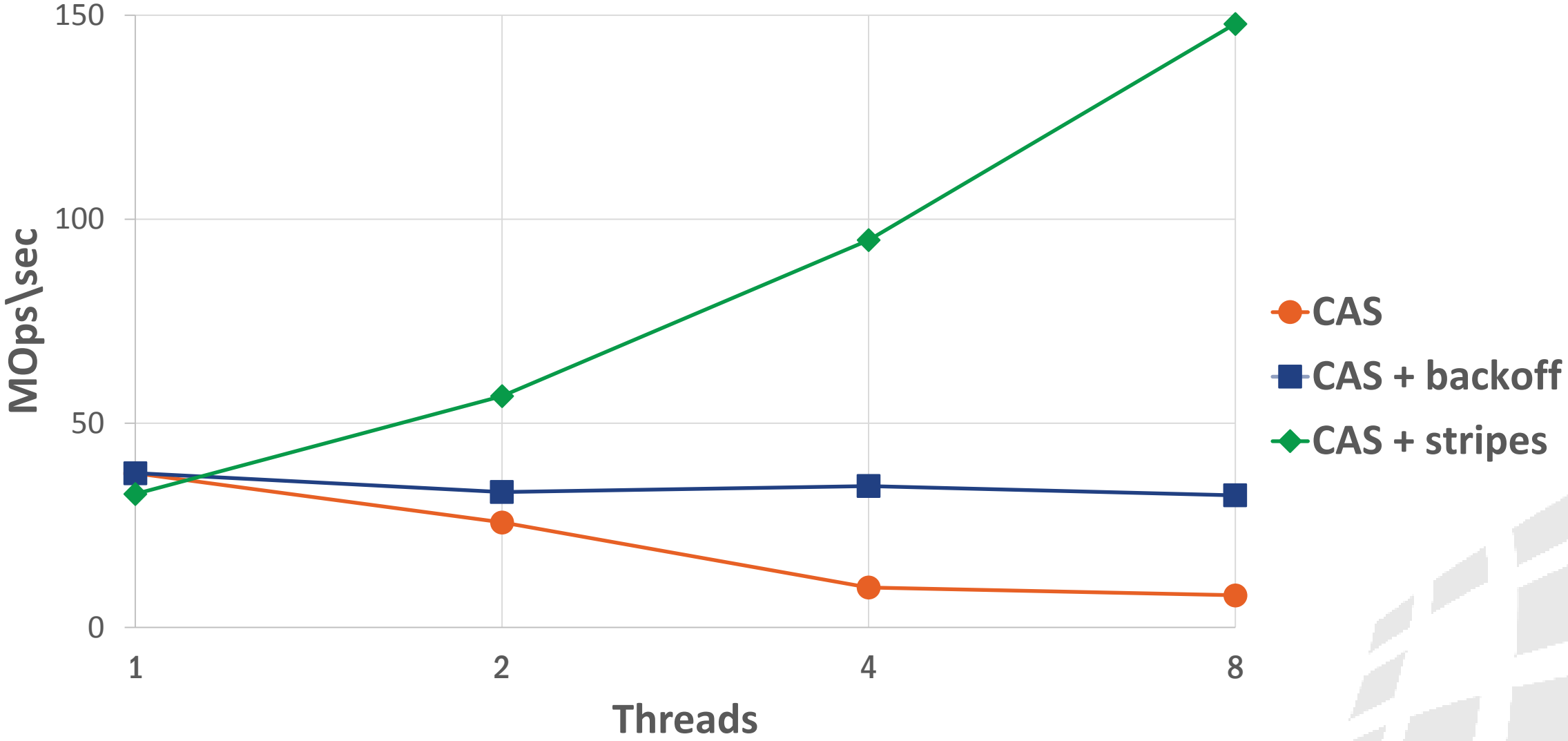
Производительность: no stores

```
 1: volatile HashMap<K, V> map;
 2:
 3: V get(K key) {
 4:     return map.get(key);
 5: }
 6:
 7: V put(K key, V val) {
 8:     synchronized (this) {
 9:         map = copyAndPut(key, val);
10:     }
11: }
```

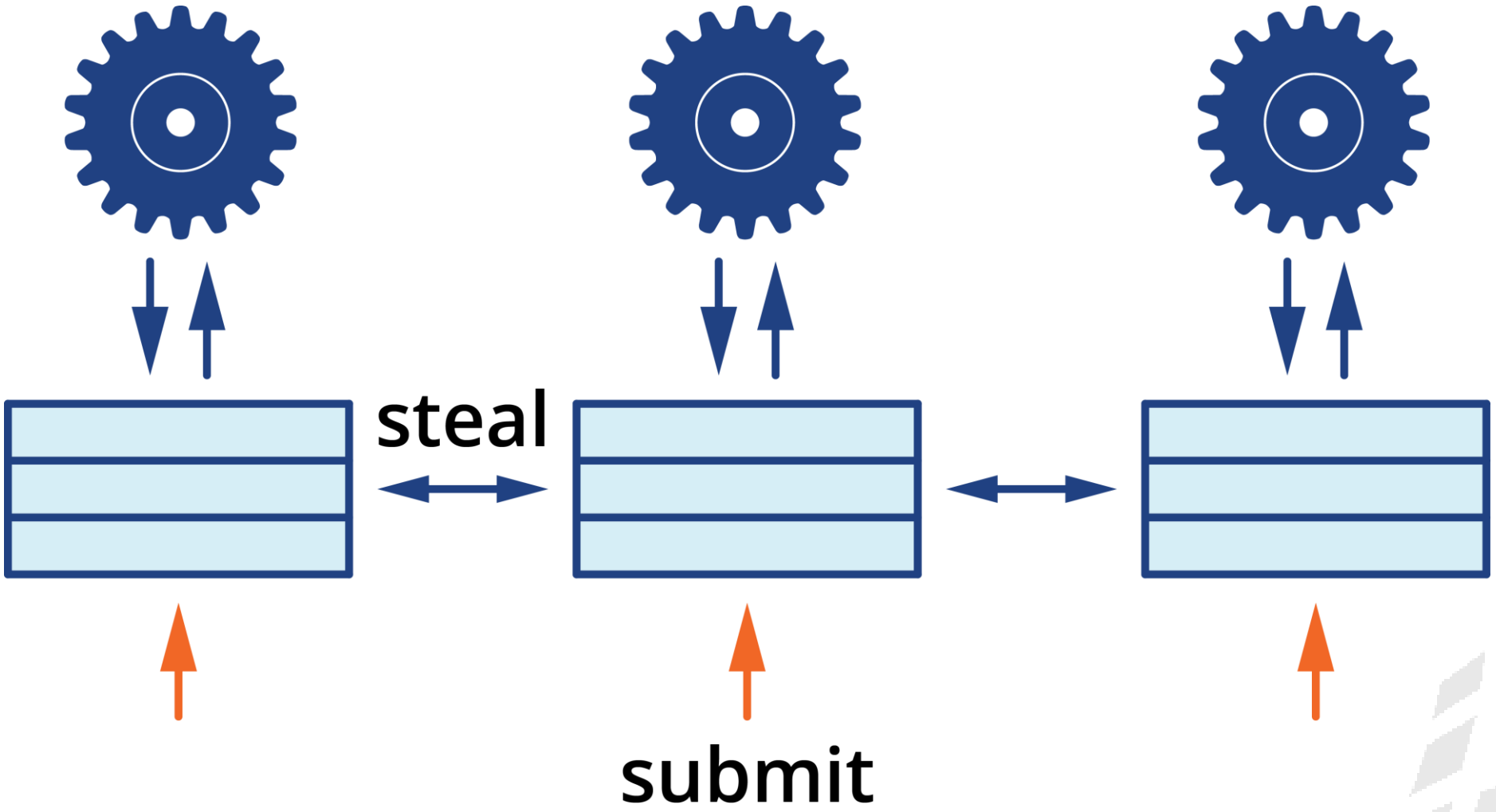
Производительность: stripes



Производительность: stripes



Производительность: local state



Производительность: консенсус

NonBlockingHashMap

```
1: void resize(Node[] oldTable, int newSize) {
2:     Node[] newTable = new Node[newSize];
3:
4:     newTable.compareAndSet(oldTable, newTable);
5:
6:     ...
7: }
```

https://github.com/boundary/high-scale-lib/blob/master/src/main/java/org/cliffc/high_scale_lib/NonBlockingHashMap.java#L752

Выводы

Мотивация:

- В первую очередь – удобство и performance
- Во вторую – дедлоки, priority inversion, safety

Структура:

- Минимум операций над shared variables
- Не удерживаем non-blocking, если оно не нужно
- Гарантии: lock-free vs wait-free

Performance:

- Latency: непрерывное выполнение кода потоком
- Throughput: боремся с contention, а не с блокировками

Ссылки

Площадки:

- http://jsr166-concurrency.10961.n7.nabble.com/
- https://groups.google.com/forum/#!forum/mechanical-sympathy

Люди:

- **Dmitry Vyukov** – http://www.1024cores.net/
- **Nitsan Wakart** – http://psy-lob-saw.blogspot.ru/
- **Martin Thompson** – http://mechanical-sympathy.blogspot.ru/

Проекты:

- **JCTools** – https://github.com/JCTools/JCTools
- **Agrona (Aeron)** – https://github.com/real-logic/Agrona

Контакты

Twitter:

- https://twitter.com/devozerov

GitHub:

- https://github.com/devozerov/ozerov_2016_jpoint

Вопросы?