Тестирование lock-free алгоритмов,

или Поиск иголки в стоге сена

Никита Коваль

nkoval@devexperts.com
twitter.com/nkoval_

DEVEXPERTS

О себе

- Инженер-исследователь в лаборатории dxLab, Devexperts
- Преподаю курс по многопоточному программированию в ИТМО
- Заканчиваю магистратуру ИТМО

DEVEXPERTS

Concurrency – это сложно!

DEVEXPERTS

Concurrency – это сложно!

… но тестировать многопоточные алгоритмы не легче!

DEVEXPERTS

Счётчик

```
class Counter {
    private int x;

    int incAndGet() {
        return ++x;
    }
}
```

Вызовем `incAndGet` параллельно

Счётчик: исполняем

```
val c = new Counter();
```

`c.incAndGet():`
`  int t1 = x;`
`  t1 = t1 + 1;`
`  x = t1;`
`  return t1;`
	`c.incAndGet():`
	`  int t2 = x;`
	`  t2 = t2 + 1;`
	`  x = t2;`
	`  return t2;`

DEVEXPERTS

Счётчик: исполняем

```
val c = new Counter();
```

```
c.incAndGet():
1 int t1 = x; // 0

3 t1 = t1 + 1; // 1
4 x = t1; // 1

8 return t1; // 1
```

```
c.incAndGet():

2 int t2 = x; // 0

5 t2 = t2 + 1; // 1
6 x = t2; // 1
7 return t2; // 1
```


DEVEXPERTS

Счётчик: исполняем

```
val c = new Counter();
```

`c.incAndGet()`:	`c.incAndGet()`:
1 `int t1 = x; // 0`	2 `int t2 = x; // 0`
3 `t1 = t1 + 1; // 1`	5 `t2 = t2 + 1; // 1`
4 `x = t1; // 1`	6 `x = t2; // 1`
	7 `return t2; // 1`
8 `return t1; // 1`	

DEVEXPERTS

Счётчик: что ожидаем

```
val c = new Counter();
```

1	2
`c.incAndGet(); // 1`	`c.incAndGet(); // 2`

ИЛИ

```
val c = new Counter();
```

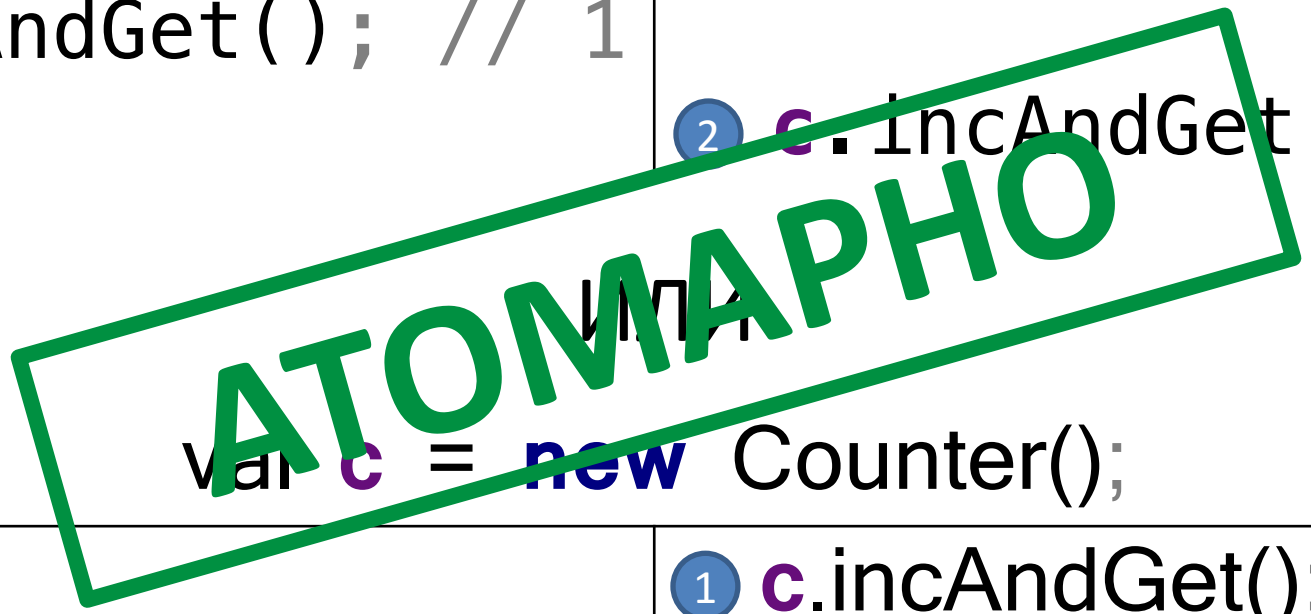
1	2
`c.incAndGet(); // 1`	`c.incAndGet(); // 2`

DEVEXPERTS

Счётчик: что ожидаем

```
val c = new Counter();
```

1 `c.incAndGet(); // 1`	2 `c.incAndGet(); // 2`
2 `c.incAndGet(); // 2`	1 `c.incAndGet(); // 1`



DEVEXPERTS

Корректность

- Последовательный алгоритм
  - Последовательная спецификация операций
- Многопоточный алгоритм
  - Линеаризуемость (aka атомарность)

Линеаризуемость

Исполнение линеаризуемо ⇔ ∃ эквивалентное последовательное исполнение *(на самом деле чуть сложнее)*

Линеаризуемость

Исполнение линеаризуемо ⇔ ∃ эквивалентное последовательное исполнение *(на самом деле чуть сложнее)*

```
val m = new ConcurrentHashMap<Int, Int>();
```

`m.put(4, 5);`	`m.get(4);`	`m.put(2, 4);`
`m.get(2);`	`m.put(1, 3);`	`m.get(4);`
`m.put(1, 1);`		`m.get(1);`

DEVEXPERTS

Линеаризуемость

Исполнение линеаризуемо ⇔ ∃ эквивалентное последовательное исполнение *(на самом деле чуть сложнее)*

```
val m = new ConcurrentHashMap<Int, Int>();
```

`m.put(4, 5); // 0`	`m.get(4); // 0`	`m.put(2, 4); // 0`
`m.get(2); // 4`	`m.put(1, 3); // 0`	`m.get(4); // 5`
`m.put(1, 1); // 3`		`m.get(1); // 1`

DEVEXPERTS

Линеаризуемость

Исполнение линеаризуемо ⇔ ∃ эквивалентное последовательное исполнение *(на самом деле чуть сложнее)*

```
val m = new ConcurrentHashMap<Int, Int>();
```

	1 `m.get(4); // 0`	
2 `m.put(4, 5); // 0`		
		3 `m.put(2, 4); // 0`
		4 `m.get(4); // 5`
	5 `m.put(1, 3); // 0`	
6 `m.get(2); // 4`		
7 `m.put(1, 1); // 3`		
		8 `m.get(1); // 1`

DEVEXPERTS

JCStress: счётчик

```
@JCStressTest
@Outcome(id = "1, 2", expect = Expect.ACCEPTABLE)
@Outcome(id = "2, 1", expect = Expect.ACCEPTABLE)
@State
public class CounterTest {
  Counter c = new Counter();

  @Actor
  public void actor1(IntResult2 r) { r.r1 = c.incAndGet(); }

  @Actor
  public void actor2(IntResult2 r) { r.r2 = c.incAndGet(); }
}
```


DEVEXPERTS

JCStress: счётчик

```
@JCStressTest
@Outcome(id = "1, 2", expect = Expect.ACCEPTABLE)
@Outcome(id = "2, 1", expect = Expect.ACCEPTABLE)
@State
public class CounterTest {
  Counter c = new Counter();

  @Actor
  public void actor1(IntResult2 r) { r.r1 = c.incAndGet(); }

  @Actor
  public void actor2(IntResult2 r) { r.r2 = c.incAndGet(); }
}
```


DEVEXPERTS

JCStress: счётчик

```
@JCStressTest
@Outcome(id = "1, 2", expect = Expect.ACCEPTABLE)
@Outcome(id = "2, 1", expect = Expect.ACCEPTABLE)
@State
public class CounterTest {
  Counter c = new Counter();

  @Actor
  public void actor1(IntResult2 r) { r.r1 = c.incAndGet(); }

  @Actor
  public void actor2(IntResult2 r) { r.r2 = c.incAndGet(); }
}
```


DEVEXPERTS

JCStress: счётчик

```
@JCStressTest
@Outcome(id = "1, 2", expect = Expect.ACCEPTABLE)
@Outcome(id = "2, 1", expect = Expect.ACCEPTABLE)
@State
public class CounterTest {
  Counter c = new Counter();

  @Actor
  public void actor1(IntResult2 r) { r.r1 = c.incAndGet(); }

  @Actor
  public void actor2(IntResult2 r) { r.r2 = c.incAndGet(); }
}
```


DEVEXPERTS

JCStress: счётчик

```
@JCStressTest
@Outcome(id = "1, 2", expect = Expect.ACCEPTABLE)
@Outcome(id = "2, 1", expect = Expect.ACCEPTABLE)
@State
public class CounterTest {
  Counter c = new Counter();

  @Actor
  public void actor1(IntResult2 r) { r.r1 = c.incAndGet(); }

  @Actor
  public void actor2(IntResult2 r) { r.r2 = c.incAndGet(); }
}
```


DEVEXPERTS

JCStress: счётчик

```
@JCStressTest
@Outcome(id = "1, 2", expect = Expect.ACCEPTABLE)
@Outcome(id = "2, 1", expect = Expect.ACCEPTABLE)
@State
public class CounterTest {
  Counter c = new Counter();

  @Actor
  public void actor1(IntResult2 r) { r.r1 = c.incAndGet(); }

  @Actor
  public void actor2(IntResult2 r) { r.r2 = c.incAndGet(); }
}
```


DEVEXPERTS

JCStress: счётчик

```
@JCStressTest
@Outcome(id = "1, 2", expect = Expect.ACCEPTABLE)
@Outcome(id = "2, 1", expect = Expect.ACCEPTABLE)
@State
public class CounterTest {
  Counter c = new Cou

  @Actor
  public void actor1(                    1 = c.incAndGet(); }

  @Actor
  public void actor2(IntResult2 r) { r.r2 = c.incAndGet(); }
}
```

Запустим

DEVEXPERTS

JCStress: счётчик

```
@JCStressTest
@Outcome(id = "1, 2", expect = Expect.ACCEPTABLE)
@Outc
@Stat
publi
  Cou

  @Ac
  pub                                                    ); }

  @Ac
  public void actor2(IntResult2 r) { r.r2 = c.incAndGet(); }
}
```

State	Occurences	Expectation
1, 1	5,003,109	FORBIDDEN
1, 2	3,830,773	ACCEPTABLE
2, 1	2,396,268	ACCEPTABLE

DEVEXPERTS

JCStress: тестируем ConcurrentHashMap

Сценарии:

- Одновременное добавление и удаление по ключу
- Параллельное обновление одного bucket-а
- Параллельное обновление разных bucket-ов
- Параллельное добавление/удаление и перехеширование
- …

DEVEXPERTS

JCStress: тестируем ConcurrentHashMap

Сценарии:

- Одновременное добавление и удаление по ключу
- Параллельное обновление одного bucket-а
- Параллельное обновление разных bucket-ов
- Параллельное добавление/удаление и перехеширование
- …

Возможные исполнения:

- 2 операции x 2 потока ⇒ 6 последовательных исполнений
- 2 операции x 3 потока ⇒ **90 (!)** последовательных исполнений

DEVEXPERTS

JCStress: тестируем ConcurrentHashMap

Сценарии:

- Одновременное добавление и удаление по ключу
- Параллельное обновле[illegible] одного bi[illegible]
- Параллельное обновлени[illegible]
- Параллельное доба[illegible]
- …

Возможные [illegible]

- 2 операции x 2 потока ⇒ [illegible] последовательных исполнений
- 2 операции x 3 потока ⇒ **90 (!)** последовательных исполнений

DEVEXPERTS

JCStress: резюмируем

- Проверяет корректность заданного сценария
- Универсален: не только линеаризуемость

- Нужно продумать сценарии
  - Это сложно
- Нужно описать корректные исходы
  - Муторно и долго

DEVEXPERTS

JCStress: резюмируем

Автоматизировать?

- Нужно продумать сценарии
  - Это сложно
- Нужно описать корректные исходы
  - Муторно и долго

DEVEXPERTS

Идеальный инструмент

1. **Задать состояние**

```
public class CHMTest {
  private Map<Int, Int> m =
        new ConcurrentHashMap<>();
}
```


DEVEXPERTS

Идеальный инструмент

1. Задать состояние
2. Задать операции

```
public class CHMTest {
  private Map<Int, Int> m =
        new ConcurrentHashMap<>();

  @Operation
  public int put(Int k, Int v) {
    return m.put(k, v);
  }

  @Operation
  public int get(Int k) {
    return m.get(k);
  }


}
```


DEVEXPERTS

Идеальный инструмент

1. Задать состояние
2. Задать операции
3. Проверить на линеаризуемость

Сделай хорошо!

```
public class CHMTest {
  private Map<Int, Int> m =
        new ConcurrentHashMap<>();

  @Operation
  public int put(Int k, Int v) {
    return m.put(k, v);
  }

  @Operation
  public int get(Int k) {
    return m.get(k);
  }

  public void test() { check(this); }
}
```


DEVEXPERTS

Идеальный инструмент

1. Задать состояние
2. Задать операции
3. Проверить на линеаризуемость

Сделай хорошо!

```
public class CHMTest {
  private Map<Int, Int> m =
        new ConcurrentHashMap<>();

  @Operation
  public int put(Int k, Int v) {
    return m.put(k, v);
  }

  @Operation
  public int get(Int k) {
    return m.get(k);
  }

  public void test() { check(this); }
}
```


DEVEXPERTS

Lin-Check

- Lin-Check = **Lin**earization **Check**er
  - https://github.com/Devexperts/lin-check
- Генерирует сценарии выполнения, запускает и проверяет на линеаризуемость

DEVEXPERTS

Lin-Check: «Hello World»

```
@StressCTest
public class CounterTest {
    private Counter counter;

    @Reset
    public void reload() { counter = new Counter(); }

    @Operation
    public int incAndGet() {
        return counter.incAndGet();
    }

    @Test // JUnit test
    public void test() { LinChecker.check(CounterTest.class);
}
}
```


DEVEXPERTS

Lin-Check: «Hello World»

```
@StressCTest
public class CounterTest {
    private Counter counter;

    @Reset
    public void reload() { counter = new Counter(); }

    @Operation
    public int incAndGet() {
        return counter.incAndGet();
    }

    @Test // JUnit test
    public void test() { LinChecker.check(CounterTest.class);
}
}
```

Начальное состояние

DEVEXPERTS

Lin-Check: «Hello World»

```
@StressCTest
public class CounterTest {
    private Counter counter;

    @Reset
    public void reload() { counter = new Counter(); }

    @Operation
    public int incAndGet() {
        return counter.incAndGet();
    }

    @Test // JUnit test
    public void test() { LinChecker.check(CounterTest.class);
}
}
```

Операции

DEVEXPERTS

Lin-Check: «Hello World»

```
@StressCTest
public class CounterTest {
    private Counter counter;

    @Reset
    public void reload() { counter = new Counter(); }

    @Operation
    public int incAndGet() {
        return counter.incAndGet();
    }

    @Test // JUnit test
    public void test() { LinChecker.check(CounterTest.class);
}
}
```

Проверка на линеаризуемость

DEVEXPERTS

Lin-Check: «Hello World»

```
@StressCTest
public class CounterTest {
    private Counter counter;

    @Reset
    public void reload() { counter = new Counter(); }

    @Operation
    public int incAndGet() {
        return counter.incAndGet();
    }

    @Test // JUnit test
    public void test() { LinChecker.check(CounterTest.class);
}
}
```


DEVEXPERTS

Lin-Check: «Hello World»

```
@StressCTest
public class CounterTest {
    private Counter counter;

    @Reset
    public void reload() { counter = new Counter(); }

    @Operation
    public int incAnd
        return counte
    }

    @Test // JUnit test
    public void test() { LinChecker.check(CounterTest.class);
}
}
```

Запустим

DEVEXPERTS

Lin-Check: «Hello World»

```
= Iteration 1 / 200 =
Actors per thread:
[incAndGet(), incAndGet(), incAndGet()]
[incAndGet(), incAndGet(), incAndGet(), incAndGet()]

Non-linearizable execution:
[2, 3, 5]
[1, 3, 4, 6]

Possible linearizable executions:
[3, 6, 7]
[1, 2, 4, 5]

[1, 2, 5]
[3, 4, 6, 7]
...
```


DEVEXPERTS

Lin-Check: «Hello World»

```
= Iteration 1 / 200 =
Actors per thread:
[incAndGet(), incAndGet(), incAndGet()]
[incAndGet(), incAndGet(), incAndGet(), incAndGet()]

Non-linearizable execution:
[2, 3, 5]
[1, 3, 4, 6]

Possible linearizable executions:
[3, 6, 7]
[1, 2, 4, 5]

[1, 2, 5]
[3, 4, 6, 7]
...
```

Сценарий выполнения

DEVEXPERTS

Lin-Check: «Hello World»

```
= Iteration 1 / 200 =
Actors per thread:
[incAndGet(), incAndGet(), incAndGet()]
[incAndGet(), incAndGet(), incAndGet(), incAndGet()]
```

Non-linearizable execution:

```
[2, 3, 5]
[1, 3, 4, 6]
```

Не могут два incAndGet вернуть одно и то же

```
Possible linearizable executions:
[3, 6, 7]
[1, 2, 4, 5]

[1, 2, 5]
[3, 4, 6, 7]
...
```


DEVEXPERTS

Lin-Check: чуть сложнее

```
@<Strategy>CTest
@Param(name = "key", gen = IntGen.class, conf = "1:3")
@Param(name = "value", gen = IntGen.class)
public class CHMTest {
    private Map<Integer, Integer> m;

    @Reset
    public void reload() { m = new ConcurrentHashMap<>(); }

    @Operation(params = {"key", "value"})
    public int put(Integer k, Integer v) { return m.put(k, v); }

    @Operation
    public int get(@Param(name = "key") Integer k) { return m.get(k); }

    @Test // JUnit test
    public void test() throws Exception { LinChecker.check(CHMTest.class); }
}
```

Lin-Check: чуть сложнее

```
@<Strategy>CTest
@Param(name = "key", gen = IntGen.class, conf = "1:3")
@Param(name = "value", gen = IntGen.class)
public class CHMTest {
    private Map<Integer, Integer> m;

    @Reset
    public void reload() { m = new ConcurrentHashMap<>(); }

    @Operation(params = {"key", "value"})
    public int put(Integer k, Integer v) { return m.put(k, v); }

    @Operation
    public int get(@Param(name = "key") Integer k) { return m.get(k); }

    @Test // JUnit test
    public void test() throws Exception { LinChecker.check(CHMTest.class); }
}
```

Начальное состояние

DEVEXPERTS

Lin-Check: чуть сложнее

```
@<Strategy>CTest
@Param(name = "key", gen = IntGen.class, conf = "1:3")
@Param(name = "value", gen = IntGen.class)
public class CHMTest {
    private Map<Integer, Integer> m;

    @Reset
    public void reload() { m = new ConcurrentHashMap<>(); }

    @Operation(params = {"key", "value"})
    public int put(Integer k, Integer v) { return m.put(k, v); }

    @Operation
    public int get(@Param(name = "key") Integer k) { return m.get(k); }

    @Test // JUnit test
    public void test() throws Exception { LinChecker.check(CHMTest.class); }
}
```

Операции

DEVEXPERTS

Lin-Check: чуть сложнее

```
@<Strategy>CTest
@Param(name = "key", gen = IntGen.class, conf = "1:3")
@Param(name = "value", gen = IntGen.class)
public class CHMTest {
    private Map<Integer, Integer> m;

    @Reset
    public void reload() { m = new ConcurrentHashMap<>(); }

    @Operation(params = {"key", "value"})
    public int put(Integer k, Integer v) { return m.put(k, v); }

    @Operation
    public int get(@Param(name = "key") Integer k) { return m.get(k); }

    @Test // JUnit test
    public void test() throws Exception { LinChecker.check(CHMTest.class); }
}
```

Генераторы параметров операций

DEVEXPERTS

Генератор параметров операций

```
/**
 * The implementation of this interface is used
 * to generate parameter values for
 * {@link Operation operations}.
 */
public interface ParameterGenerator<T> {
    T generate();
}
```


DEVEXPERTS

Lin-Check: чуть сложнее

```
@<Strategy>CTest
@Param(name = "key", gen = IntGen.class, conf = "1:3")
@Param(name = "value", gen = IntGen.class)
public class CHMTest {
    private Map<Integer, Integer> m;

    @Reset
    public void reload() { m = new ConcurrentHashMap<>(); }

    @Operation(params = {"key", "value"})
    public int put(Integer k, Integer v) { return m.put(k, v); }

    @Operation
    public int get(@Param(name = "key") Integer k) { return m.get(k); }

    @Test // JUnit test
    public void test() throws Exception { LinChecker.check(CHMTest.class); }
}
```

Генераторы параметров операций

DEVEXPERTS

Lin-Check: чуть сложнее

```
@<Strategy>CTest
@Param(name = "key", gen = IntGen.class, conf = "1:3")
@Param(name = "value", gen = IntGen.class)
public class CHMTest {
    private Map<Integer, Integer> m;

    @Reset
    public void reload() { m = new ConcurrentHashMap<>(); }

    @Operation(params = {"key", "value"})
    public int put(Integer k, Integer v) { return m.put(k, v); }

    @Operation
    public int get(@Param(name = "key") Integer k) { return m.get(k); }

    @Test // JUnit test
    public void test() throws Exception { LinChecker.check(CHMTest.class); }
}
```

Проверка на линеаризуемость

DEVEXPERTS

Lin-Check: чуть сложнее

```
@<Strategy>CTest
@Param(name = "key", gen = IntGen.class, conf = "1:3")
@Param(name = "value", gen = IntGen.class)
public class CHMTest {
    private Map<Integer, Integer> m;

    @Reset
    public void reload() { m = new ConcurrentHashMap<>(); }

    @Operation(params = {"key", "value"})
    public int put(Integer k, Integer v) { return m.put(k, v); }

    @Operation
    public int get(@Param(name = "key") Integer k) { return m.get(k); }

    @Test // JUnit test
    public void test() throws Exception { LinChecker.check(CHMTest.class); }
}
```

Генерация сценариев выполнения

- Тестируем ConcurrentHashMap
- Генерируем байт-код для каждого из потоков (ASM)
- Сохраняем результат каждой операции

```
val m = new ConcurrentHashMap<Int, Int>();
```

```
// start thread #1
r1[1] = m.put(4, 5);
r1[2] = m.get(2);
r1[3] = m.put(1, 1);
// done
```

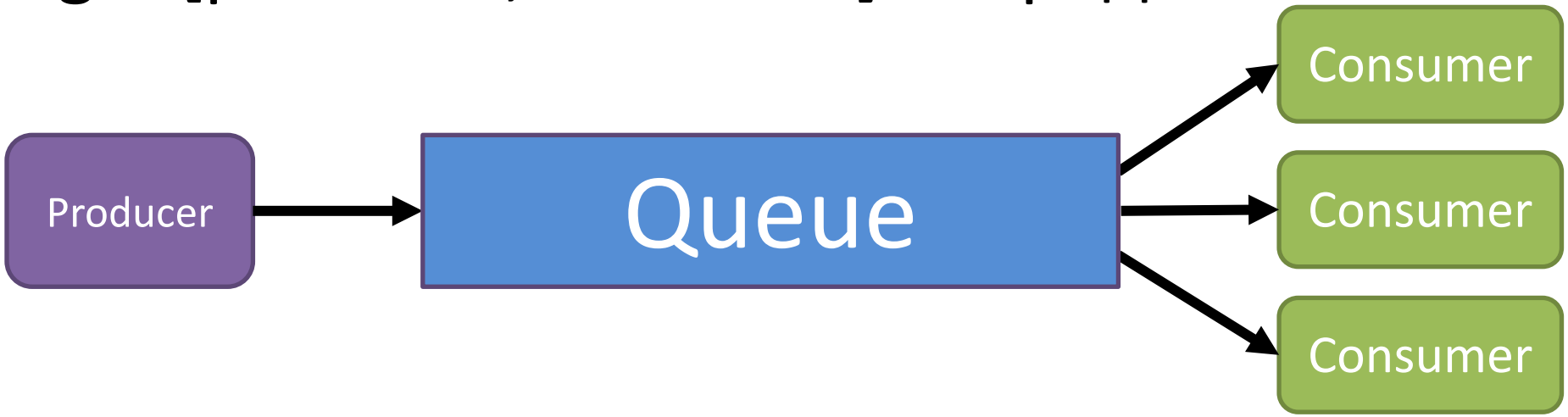
```
// start thread #2
r2[0] = m.get(4);
r2[1] = m.put(1, 3);
// done
```

```
// start thread #3
r3[0] = m.put(2, 4);
r3[1] = m.get(4);
r3[2] = m.get(1);
// done
```


DEVEXPERTS

Один писатель/читатель

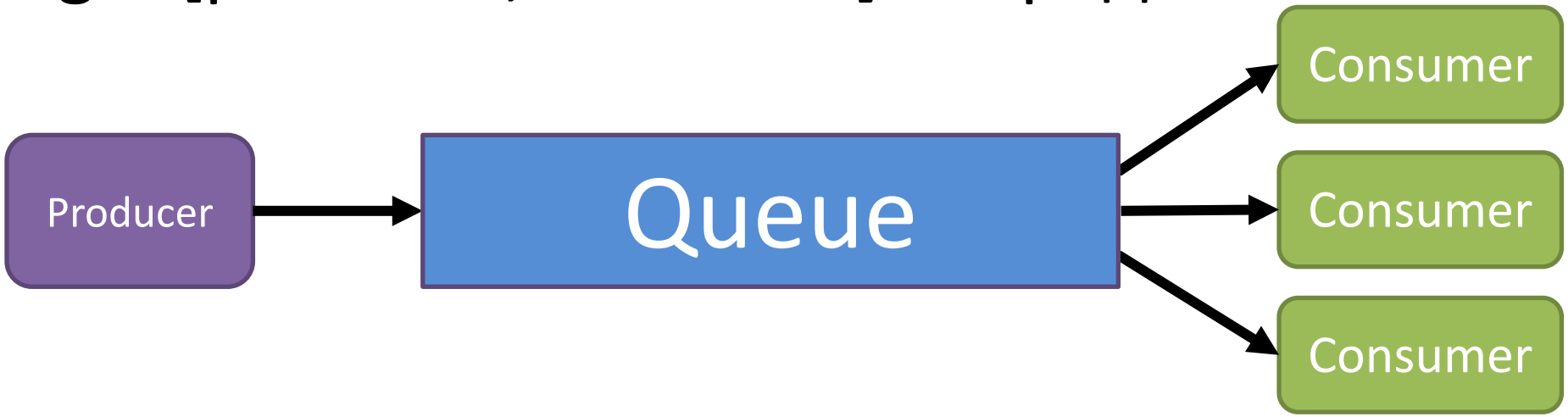
- Написали свой `SingleWriterHashMap`
- Или single-[producer,consumer] очередь



DEVEXPERTS

Один писатель/читатель

- Написали свой `SingleWriterHashMap`
- Или single-[producer,consumer] очередь



Как гарантировать, что только один писатель/читатель?

DEVEXPERTS

Один писатель/читатель: API

```
@OpGroupConfig(name = "producer", nonParallel = true)
@StressCTest
public class SPMCQueueTest {
    private SPMCQueue<Integer> q;

    @Reset
    public void reset() { q = new SPSCQueue<>(); }

    @Operation(group = "producer”)
    public void offer(@Param(gen = IntGen.class) Integer x) { q.offer(x); }

    @Operation
    public Integer poll() { return q.poll(); }

    @Test public void test() { LinChecker.check(SPMCQueueTest.class); }
}
```


DEVEXPERTS

Один писатель/читатель: API

```
@OpGroupConfig(name = "producer", nonParallel = true)
@StressCTest
public class SPMCQueueTest {
    private SPMCQueue<Integer> q;

    @Reset
    public void reset() { q = new SPSCQueue<>(); }

    @Operation(group = "producer”)
    public void offer(@Param(gen = IntGen.class) Integer x) { q.offer(x); }

    @Operation
    public Integer poll() { return q.poll(); }

    @Test public void test() { LinChecker.check(SPMCQueueTest.class); }
}
```


DEVEXPERTS

Один писатель/читатель: API

```
@OpGroupConfig(name = "producer", nonParallel = true)
@StressCTest
public class SPMCQueueTest {
    private SPMCQueue<Integer> q;

    @Reset
    public void reset() {

    @Operation(group = "pr
    public void offer(@Par                    Integer x) { q.offer(x); }

    @Operation
    public Integer poll() { return q.poll(); }

    @Test public void test() { LinChecker.check(SPMCQueueTest.class); }
}
```

Запустим

DEVEXPERTS

Один писатель/читатель: сценарии

```
= Iteration 1 / 200 =
Actors per thread:
[offer(2), poll(), poll()]
[poll(), poll(), poll(), poll()]

= Iteration 2 / 200 =
Actors per thread:
[offer(1), poll(), offer(4)]
[poll(), poll(), poll()]

= Iteration 3 / 200 =
Actors per thread:
[offer(-8), offer(-2), offer(-8), poll()]
[poll(), poll(), poll(), poll()]
```

`offer()` только в одном потоке

DEVEXPERTS

Одноразовая операция

- Некоторые операции можно выполнять только один раз
  - `shutdown()`
  - `close()`

DEVEXPERTS

Одноразовая операция

- Некоторые операции можно выполнять только один раз
  - shutdown()
  - close()

```
@Operation(runOnce = true)
public void shutdown() {
    job.shutdown();
}
```


DEVEXPERTS

Результат в качестве параметра

- Результат одной операции = параметр другой

```
class MyStack {
    Node push(int val) { ... }

    int pop() { ... }

    boolean remove(Node node) { ... }
}
```


DEVEXPERTS

Результат в качестве параметра

- Результат одной операции = параметр другой

```
class MyStack {
    Node push(int val) { ... }

    int pop() { ... }

    boolean remove(Node node) { ... }
}
```


DEVEXPERTS

Результат в качестве параметра: желание

- Хотим примерно такое исполнение:

```
                val s = new MyStack();
-------------------------------------------------------
Node node = s.push(1);  | s.pop();
s.remove(node);         |
```

- Но как его получить?

Результат в качестве параметра: API

```
@OpGroupConfig(name = "push_remove", nonParallel = true)
@StressCTest
public class MyStackTest {
    private MyStack stack; private Node lastPushNode;

    @Reset public void reset() { stack = new MyStack(); lastPushNode = null; }

    @Operation(group = "push_remove")
    public void push(@Param(gen = IntGen.class) int x) { lastPushNode = stack.push(x); }

    @Operation
    public int pop() { return stack.pop(); }

    @Operation(group = "push_remove")
    public boolean remove() {
        Node node = lastPushNode;
        return node != null ? stack.remove(node) : false;
    }

    @Test public void test() { LinChecker.check(MyStackTest.class); }
}
```


DEVEXPERTS

Результат в качестве параметра: API

```
@OpGroupConfig(name = "push_remove", nonParallel = true)
@StressCTest
public class MyStackTest {
    private MyStack stack; private Node lastPushNode;

    @Reset public void reset() { stack = new MyStack(); lastPushNode = null; }

    @Operation(group = "push_remove")
    public void push(@Param(gen = IntGen.class) int x) { lastPushNode = stack.push(x); }

    @Operation
    public int pop() { return stack.pop(); }

    @Operation(group = "push_remove")
    public boolean remove() {
        Node node = lastPushNode;
        return node != null ? stack.remove(node) : false;
    }

    @Test public void test() { LinChecker.check(MyStackTest.class); }
}
```

Запоминаем результат push-a

DEVEXPERTS

Результат в качестве параметра: API

```
@OpGroupConfig(name = "push_remove", nonParallel = true)
@StressCTest
public class MyStackTest {
    private MyStack stack; private Node lastPushNode;

    @Reset public void reset() { stack = new MyStack(); lastPushNode = null; }

    @Operation(group = "push_remove")
    public void push(@Param(gen = IntGen.class) int x) { lastPushNode = stack.push(x); }

    @Operation
    public int pop() { return stack.pop(); }

    @Operation(group = "push_remove")
    public boolean remove() {
        Node node = lastPushNode;
        return node != null ? stack.remove(node) : false;
    }

    @Test public void test() { LinChecker.check(MyStackTest.class); }
}
```

Если увидели результат push-а, удаляем

DEVEXPERTS

Результат в качестве параметра: API

```
@OpGroupConfig(name = "push_remove", nonParallel = true)
@StressCTest
public class MyStackTest {
    private MyStack stack; private Node lastPushNode;

    @Reset public void reset() { stack = new MyStack(); lastPushNode = null; }

    @Operation(group = "push_remove")
    public void push(@Param(gen = IntGen.class) int x) { lastPushNode = stack.push(x); }

    @Operation
    public int pop() { return stack.pop(); }

    @Operation(group = "push_remove")
    public boolean remove() {
        Node node = lastPushNode;
        return node != null ? stack.remove(node) : false;
    }

    @Test public void test() { LinChecker.check(MyStackTest.class); }
}
```

Выполняем push и remove в одном потоке

DEVEXPERTS

Проверка корректности

- Хотим проверить, что результаты выполнения операций (**r1**, **r2**, **r3**) возможны в линеаризуемом исполнении

DEVEXPERTS

Линеаризуемость

Исполнение линеаризуемо ⇔ ∃ эквивалентное последовательное исполнение *(на самом деле чуть сложнее)*

Проверка корректности

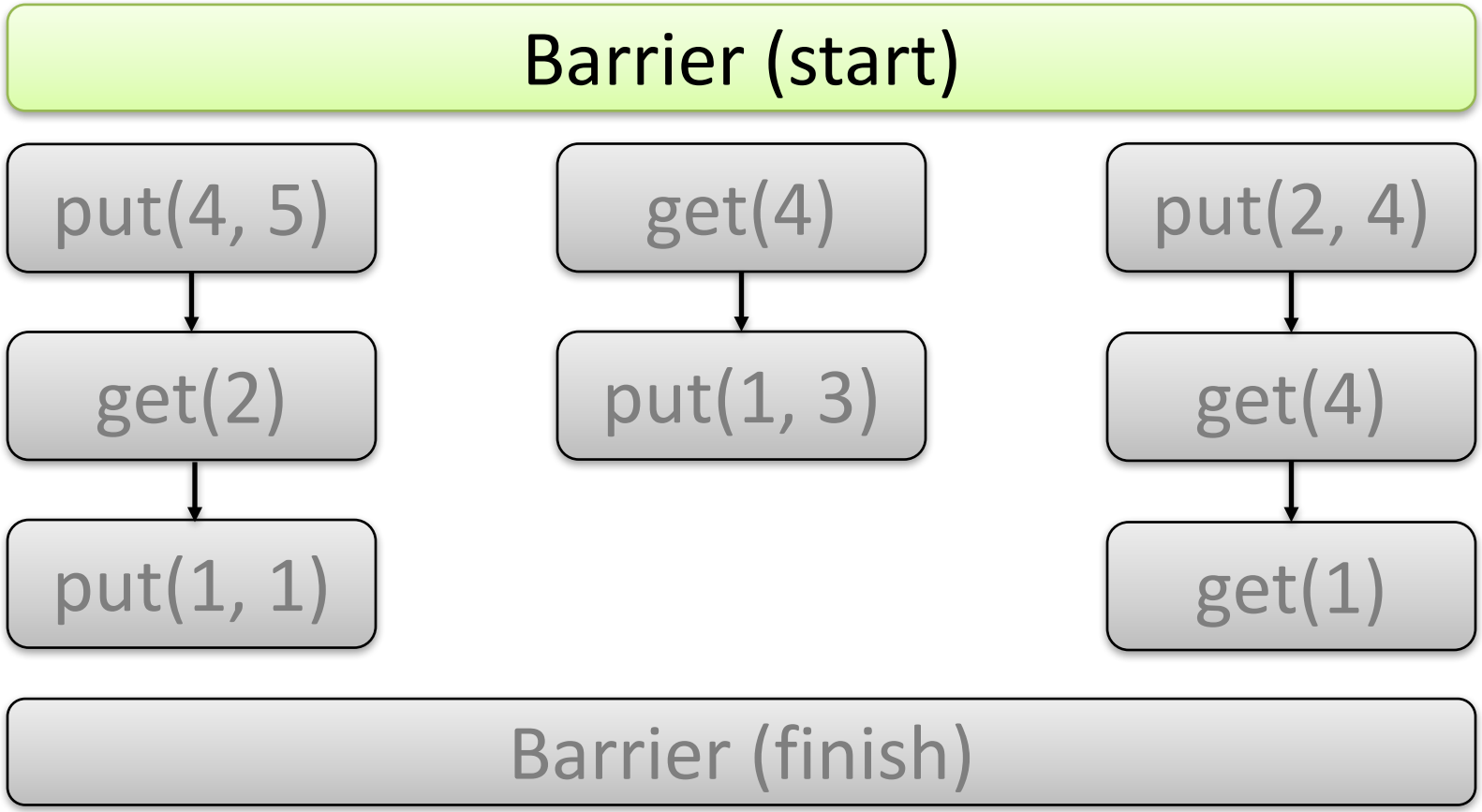
- Хотим проверить, что результаты выполнения операций (**r1**, **r2**, **r3**) возможны в линеаризуемом исполнении
- Ищем перестановку операций, результат последовательного выполнения которой совпадает с результатом многопоточного исполнения (**r1**, **r2**, **r3**)
- Если такой перестановки нет, нашли ошибку

Как запускать сценарий?

- Стресс режим
  - Запускаем потоки одновременно, ждем завершения и смотрим на результат
  - Делаем это много раз (привет, JCStress)
- Управляемое исполнение
  - Об этом позже

DEVEXPERTS

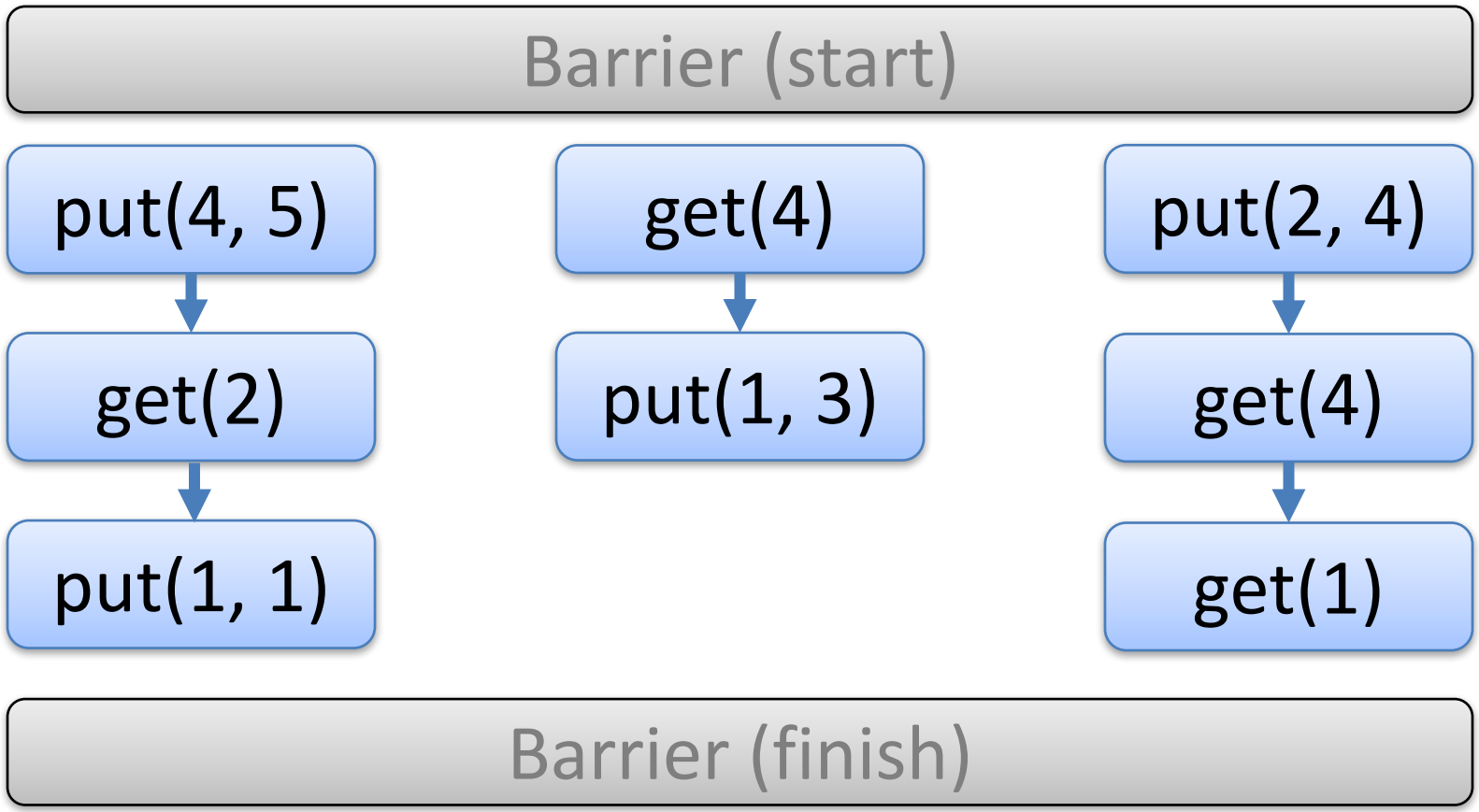
Стресс-режим: пример



«Одновременный» старт

DEVEXPERTS

Стресс-режим: пример

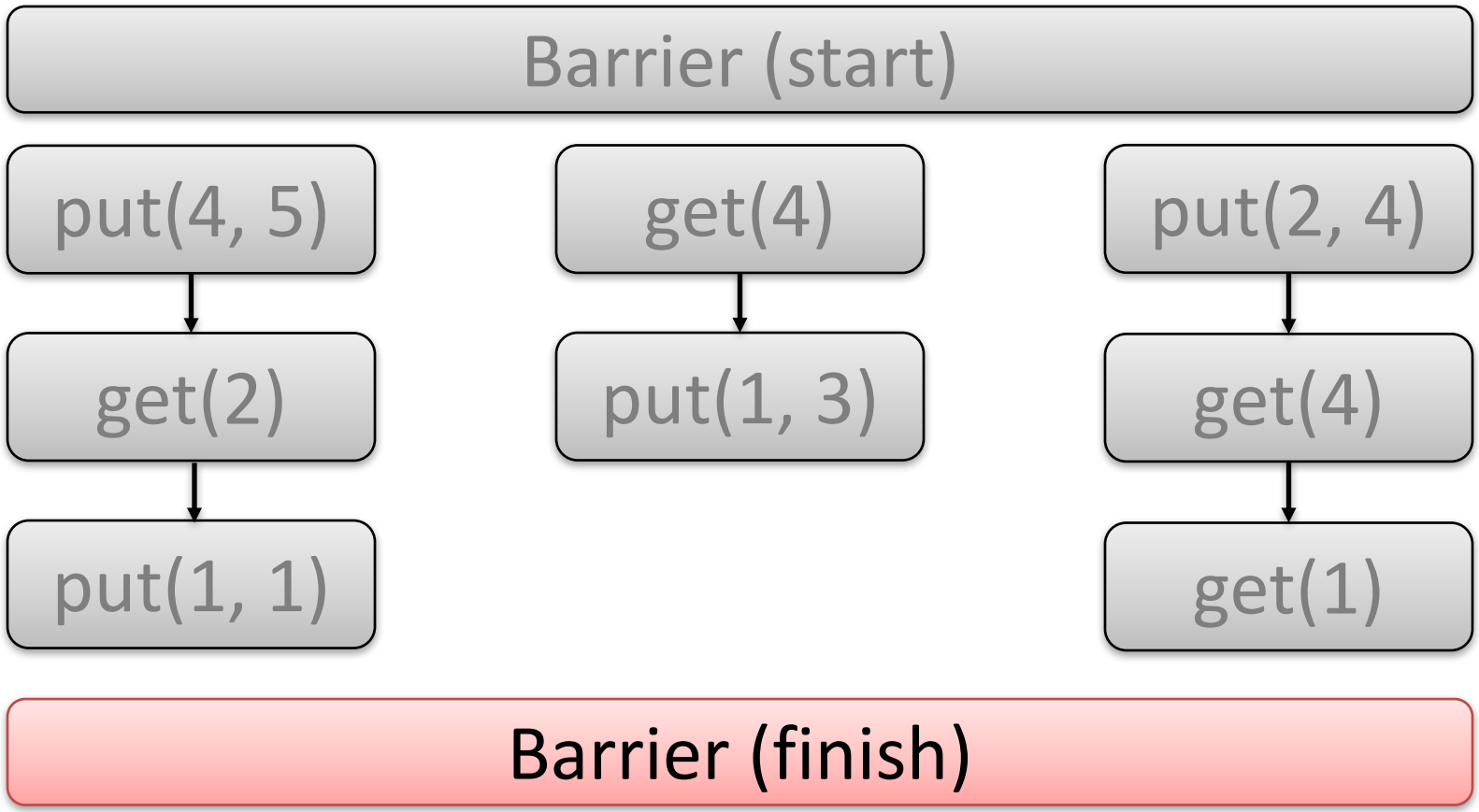


«Одновременный» старт

Параллельное исполнение операций: происходят гонки, ошибки синхронизации и т.д.

DEVEXPERTS

Стресс-режим: пример



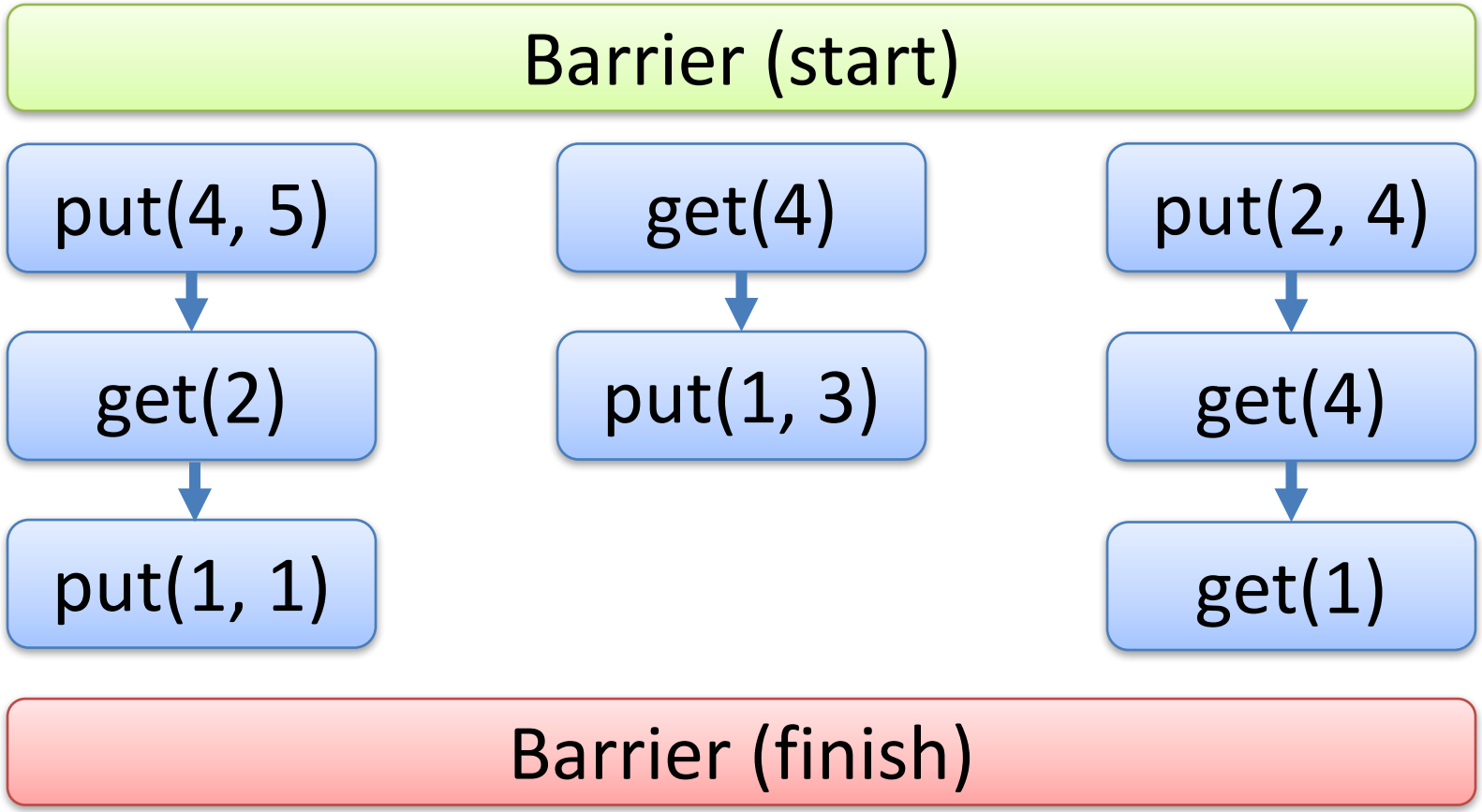
«Одновременный» старт

Параллельное исполнение операций: гонки, ошибки синхронизации и т.д.

Все закончили работу

DEVEXPERTS

Стресс-режим: пример



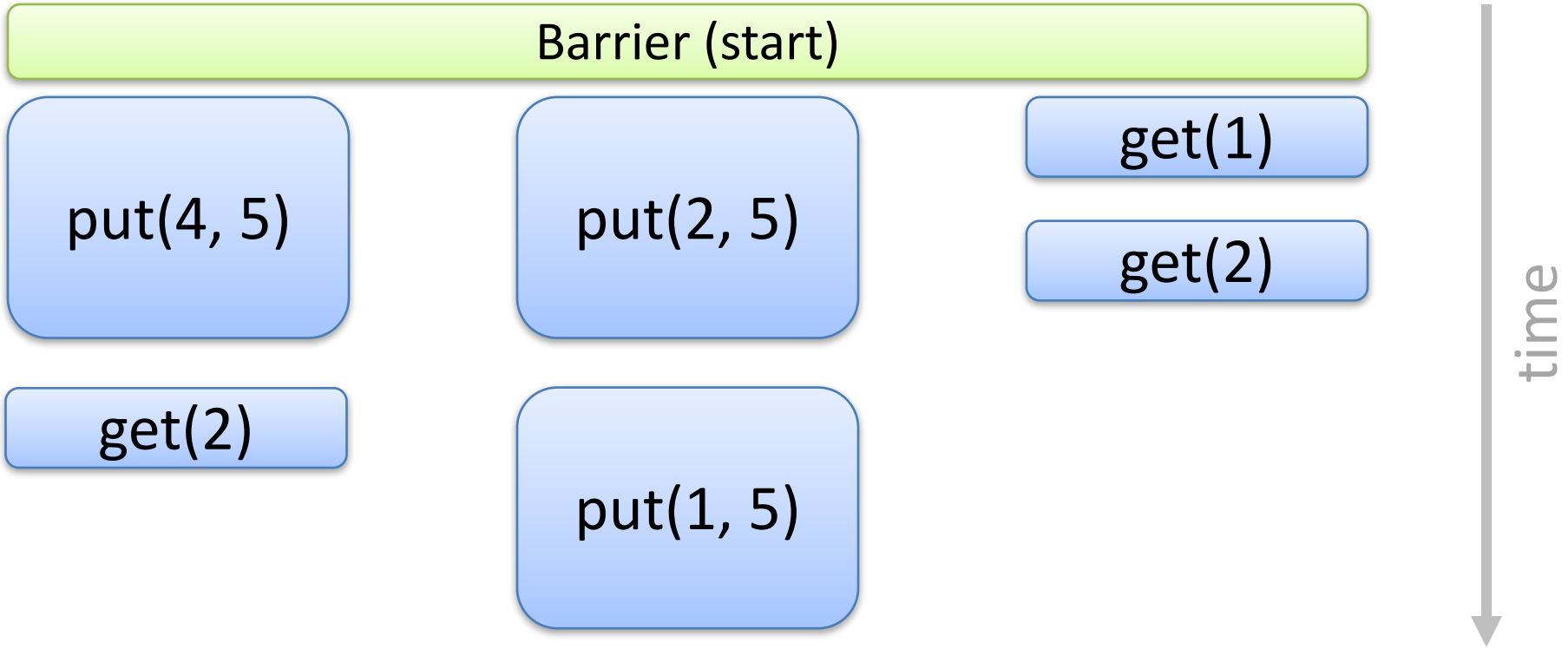
«Одновременный» старт

Параллельное исполнение операций: гонки, ошибки синхронизации и т.д.

Все закончили работу

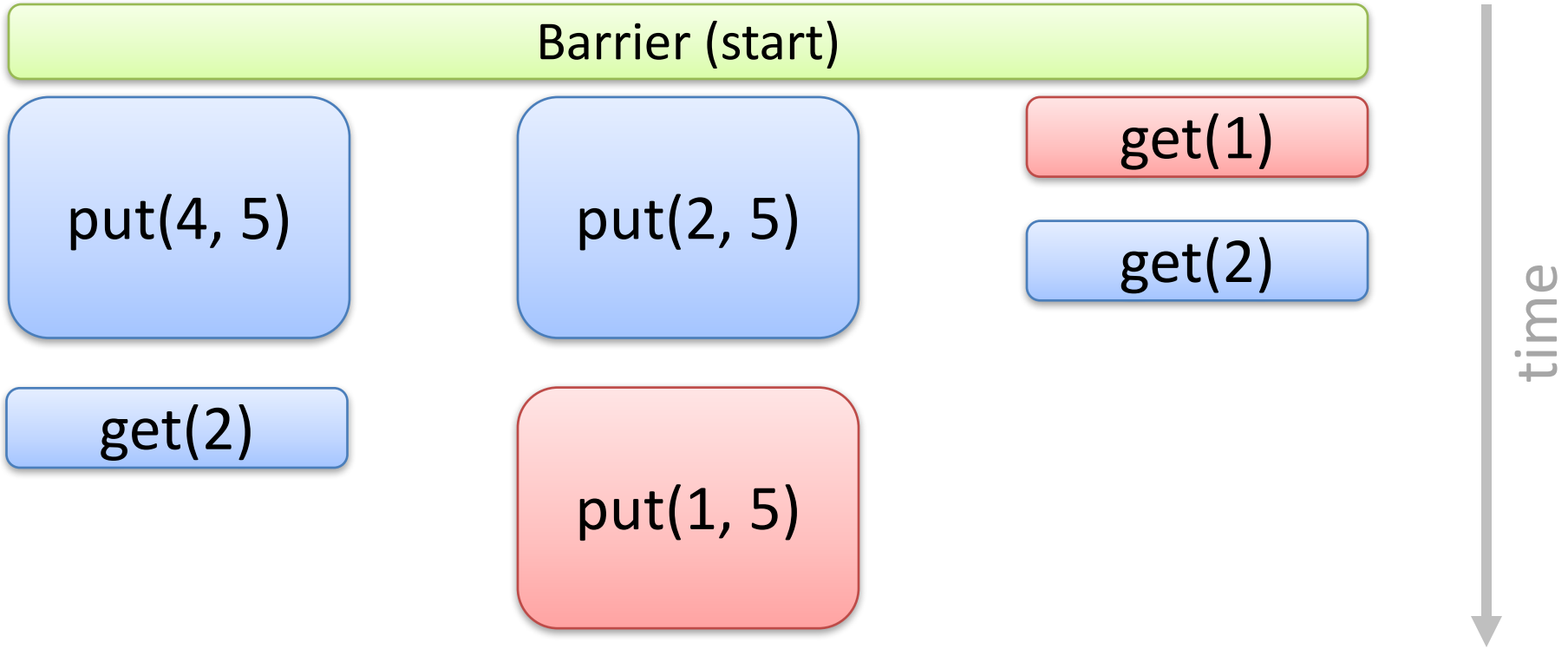
DEVEXPERTS

Стресс-режим: улучшаем покрытие



DEVEXPERTS

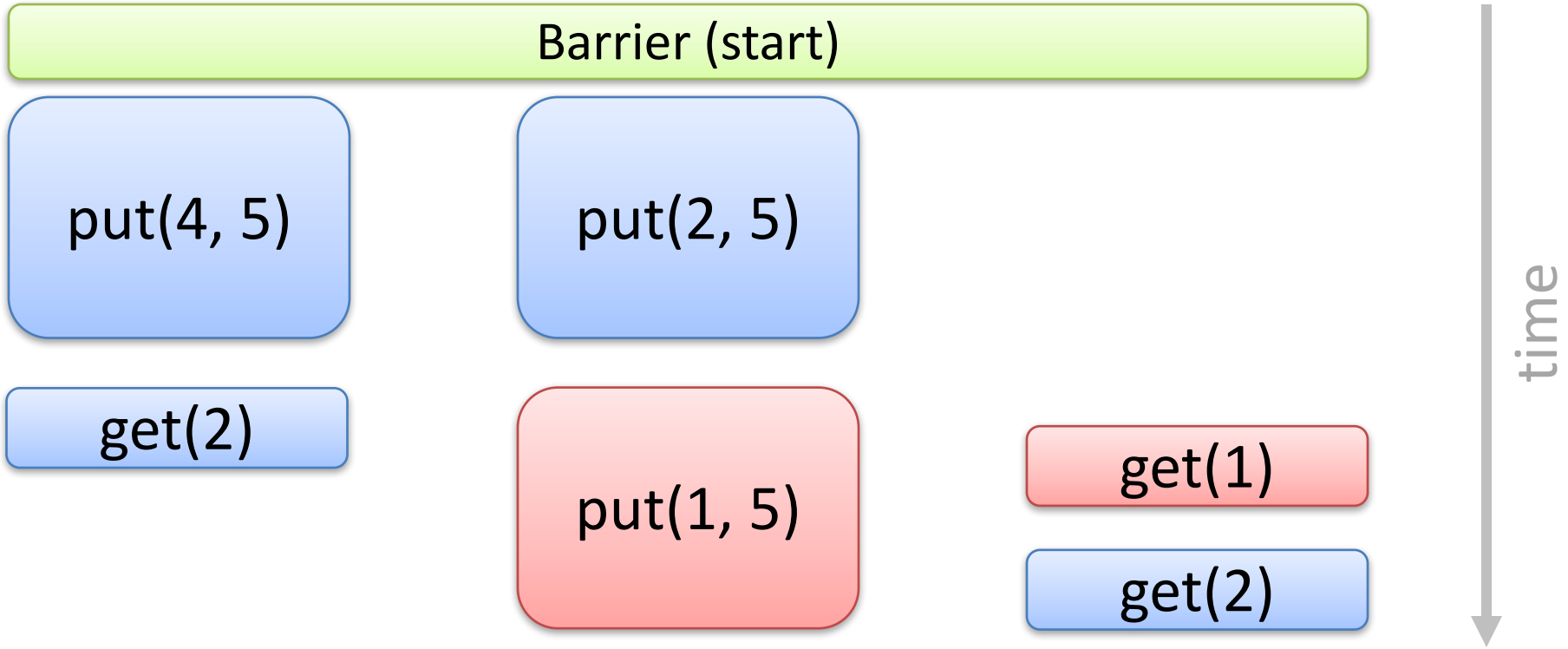
Стресс-режим: улучшаем покрытие



На практике почти всегда get(1) до put(1, 5)

DEVEXPERTS

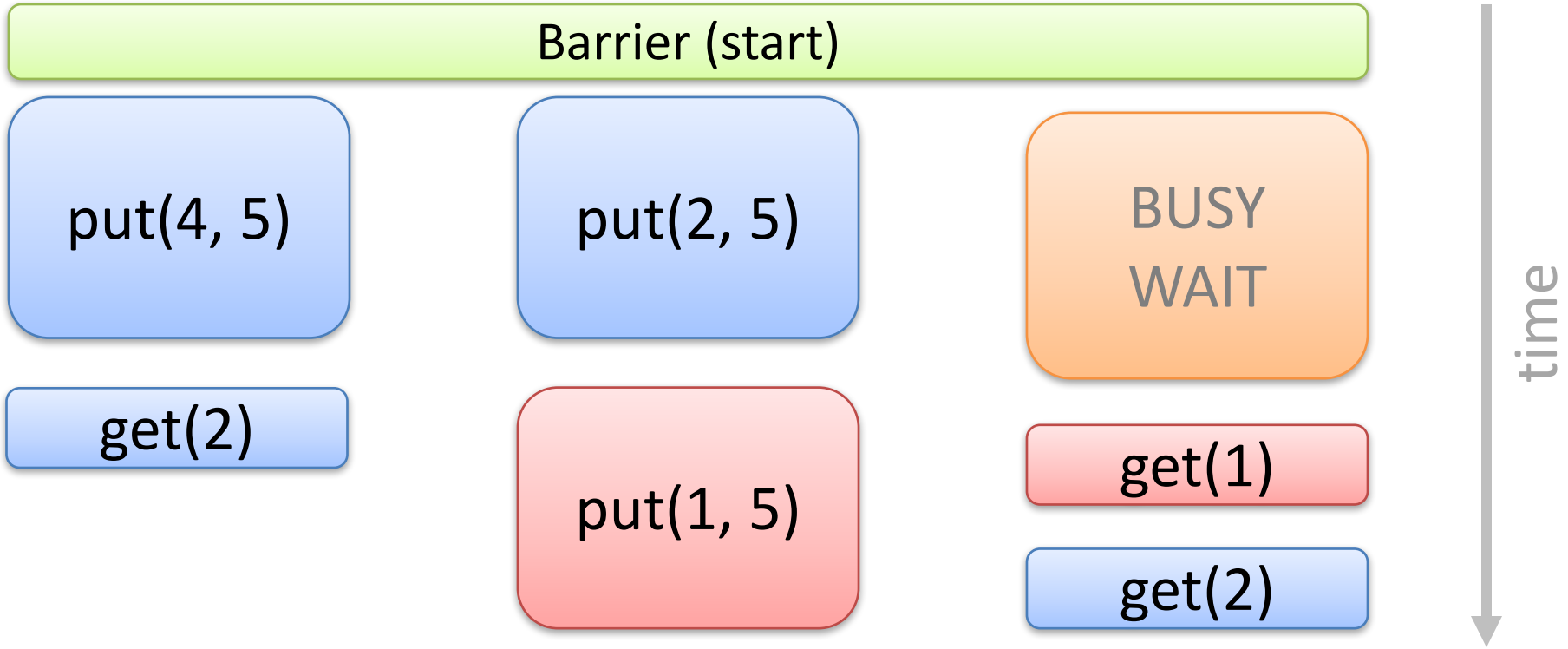
Стресс-режим: улучшаем покрытие



Хотелось бы получить такое исполнение

DEVEXPERTS

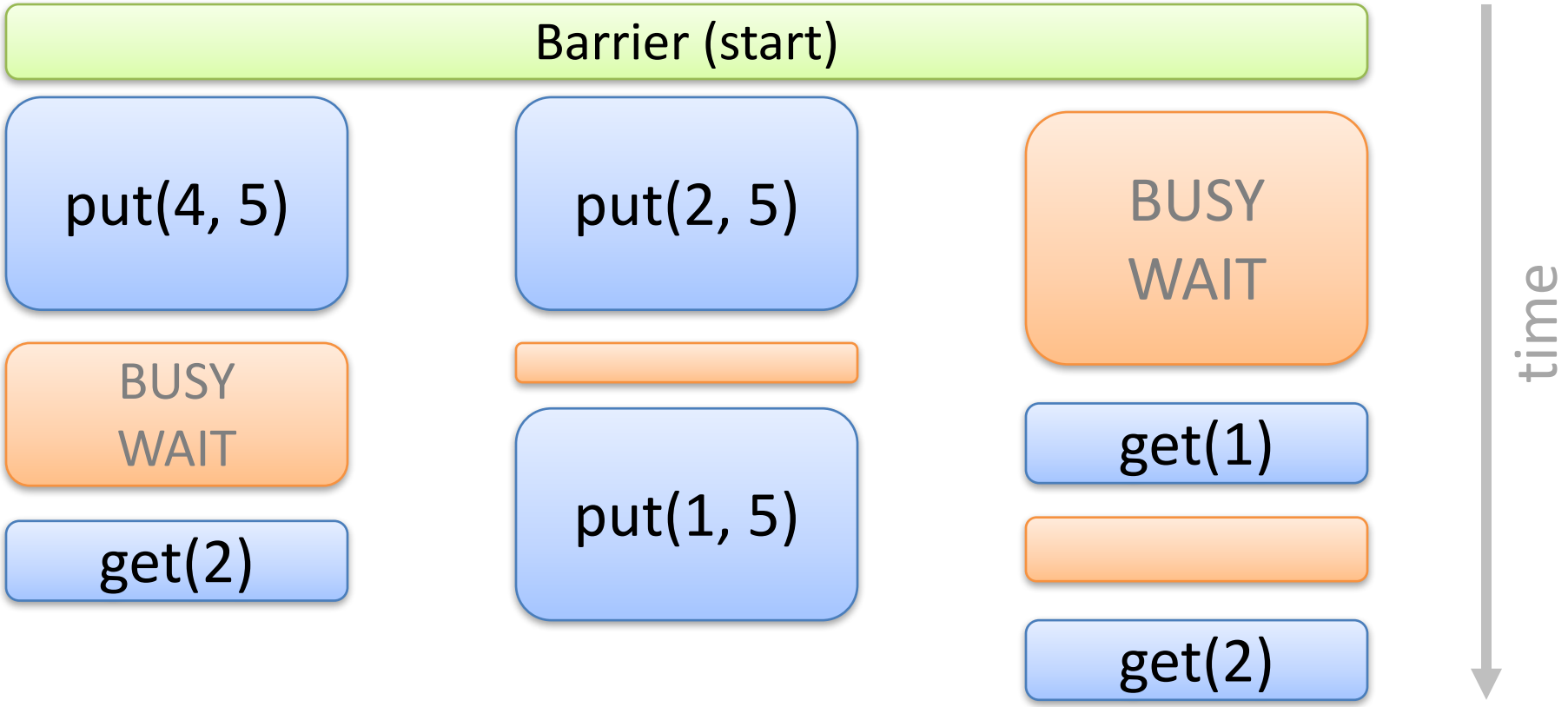
Стресс-режим: улучшаем покрытие



Вставим бесполезную работу

DEVEXPERTS

Стресс-режим: улучшаем покрытие



Везде… и случайную!

DEVEXPERTS

Стресс-режим: резюмируем

- Добавляем задержки не только между операциями, но и внутри них
- Пробуем запускать с задержками разной величины

Стресс-режим: резюмируем

- Добавляем задержки не только между операциями, но и внутри них
- Пробуем запускать с задержками разной величины

Как ещё улучшить покрытие?

DEVEXPERTS

Управляемые стратегии

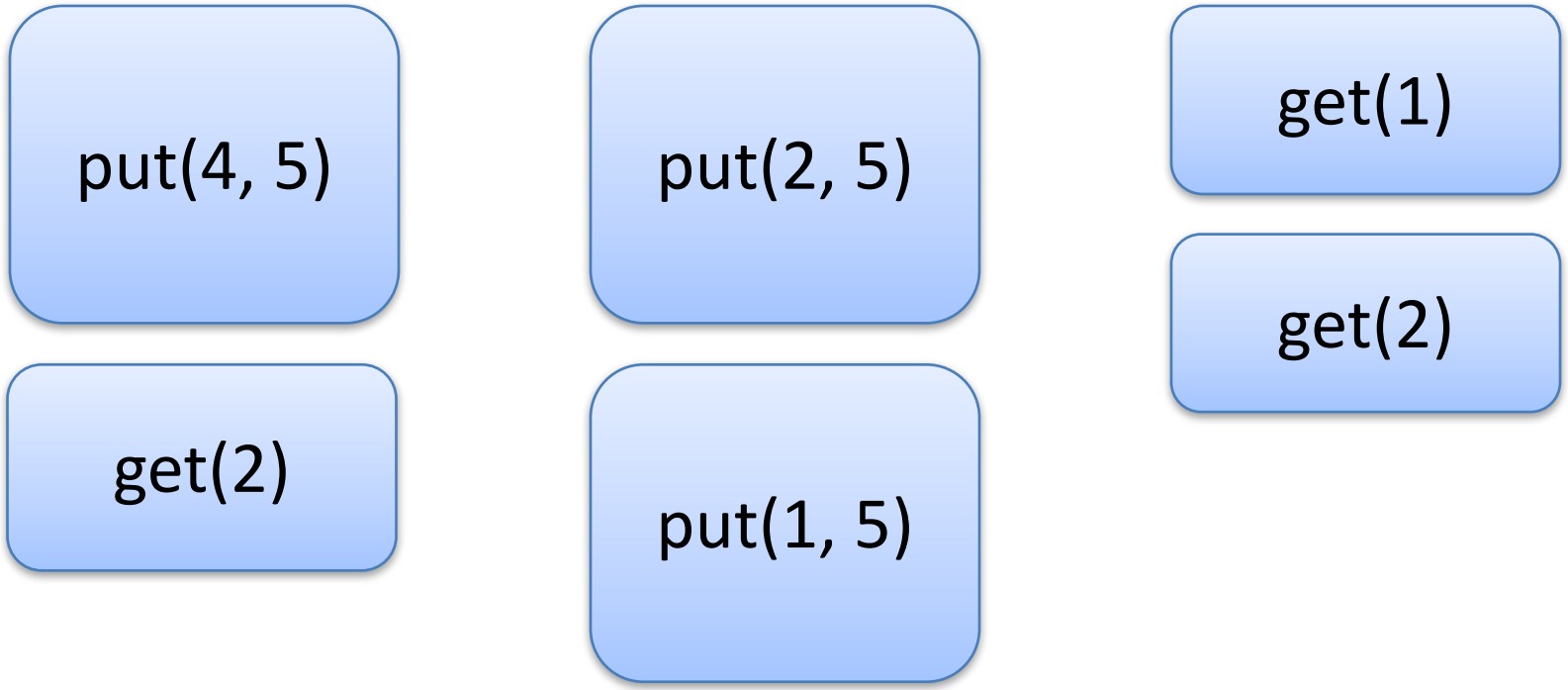
- Не будем надеяться на задержки, сами переключаем потоки в нужных местах
- Есть куча стратегий для переключения

- Позволяют находить логические ошибки
- Исполнение не содержит гонок

DEVEXPERTS

One-switch стратегия

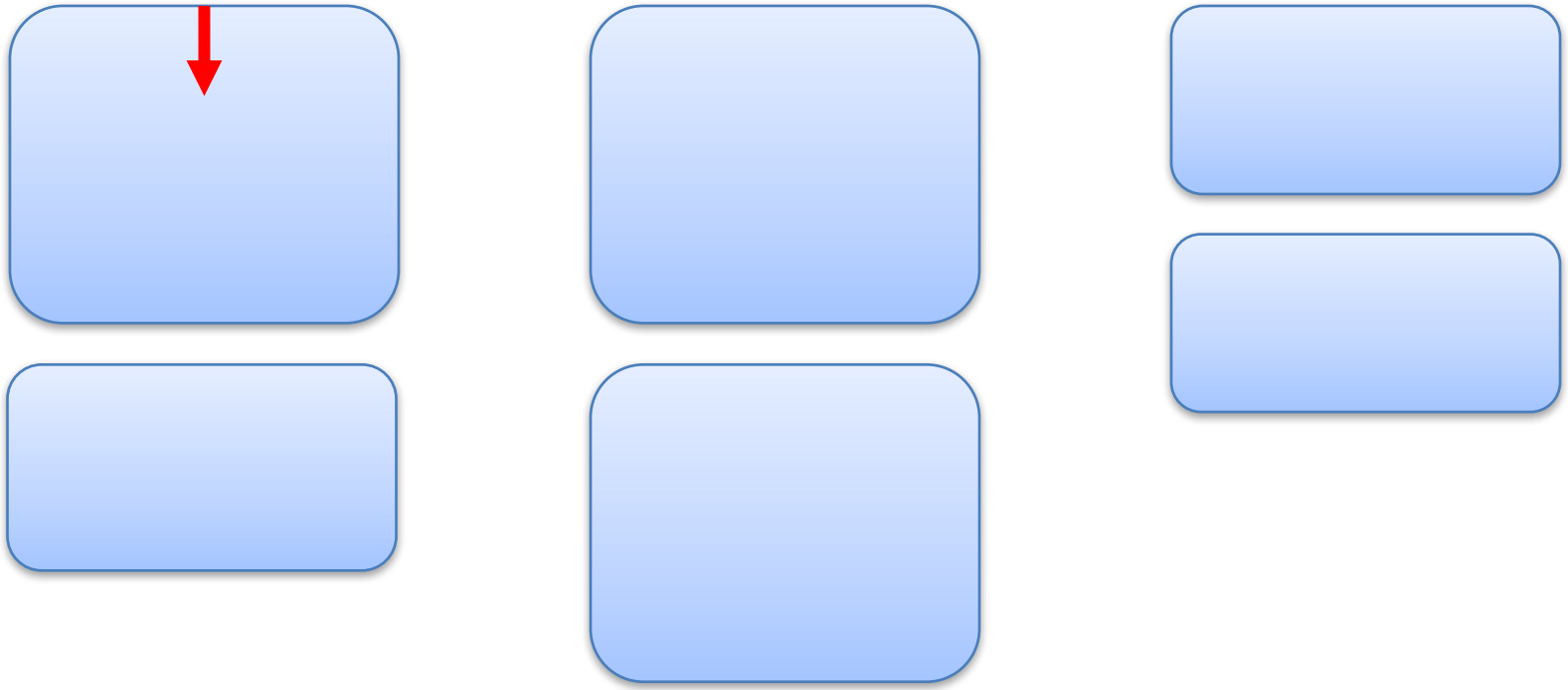
- Есть такой сценарий:



DEVEXPERTS

One-switch стратегия

- Хочется уметь исполнять как-то так:

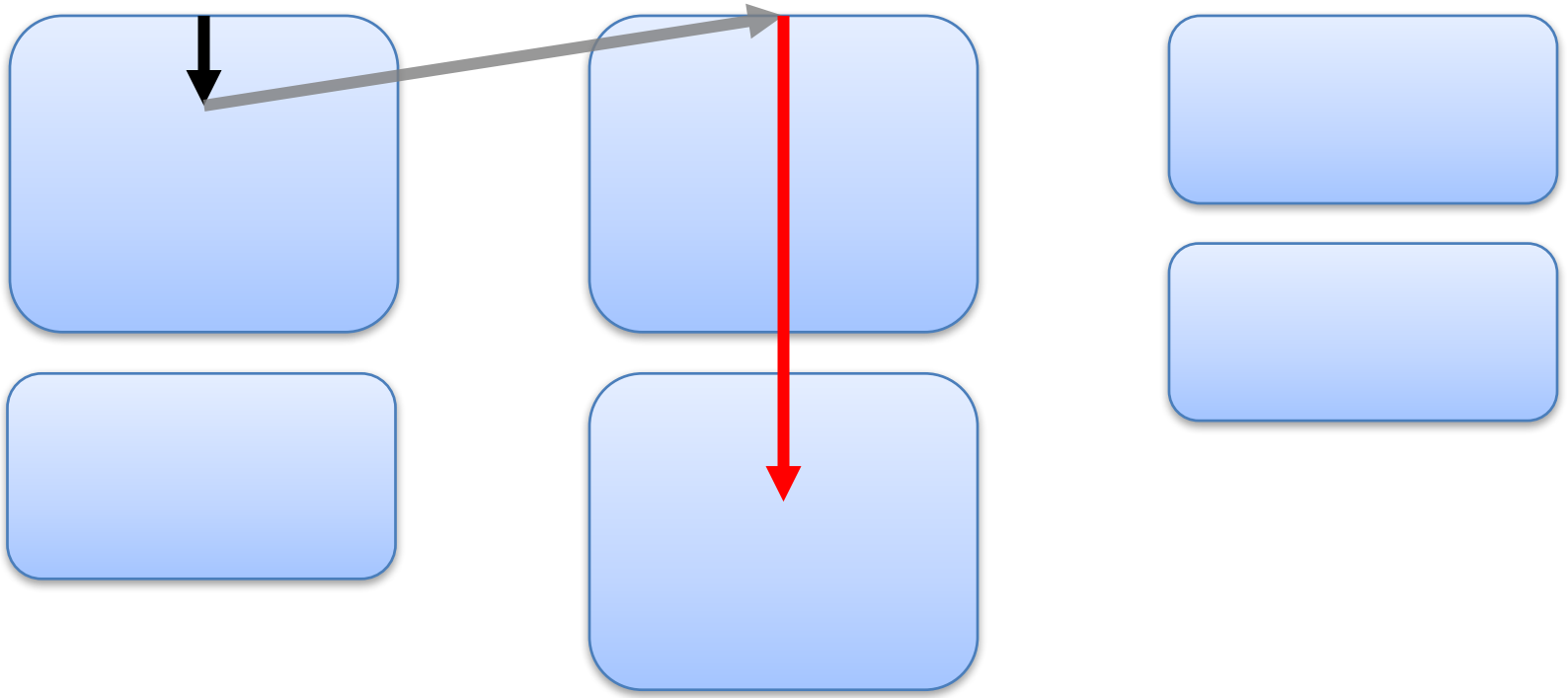


Выполним часть первого потока

DEVEXPERTS

One-switch стратегия

- Хочется уметь исполнять как-то так:

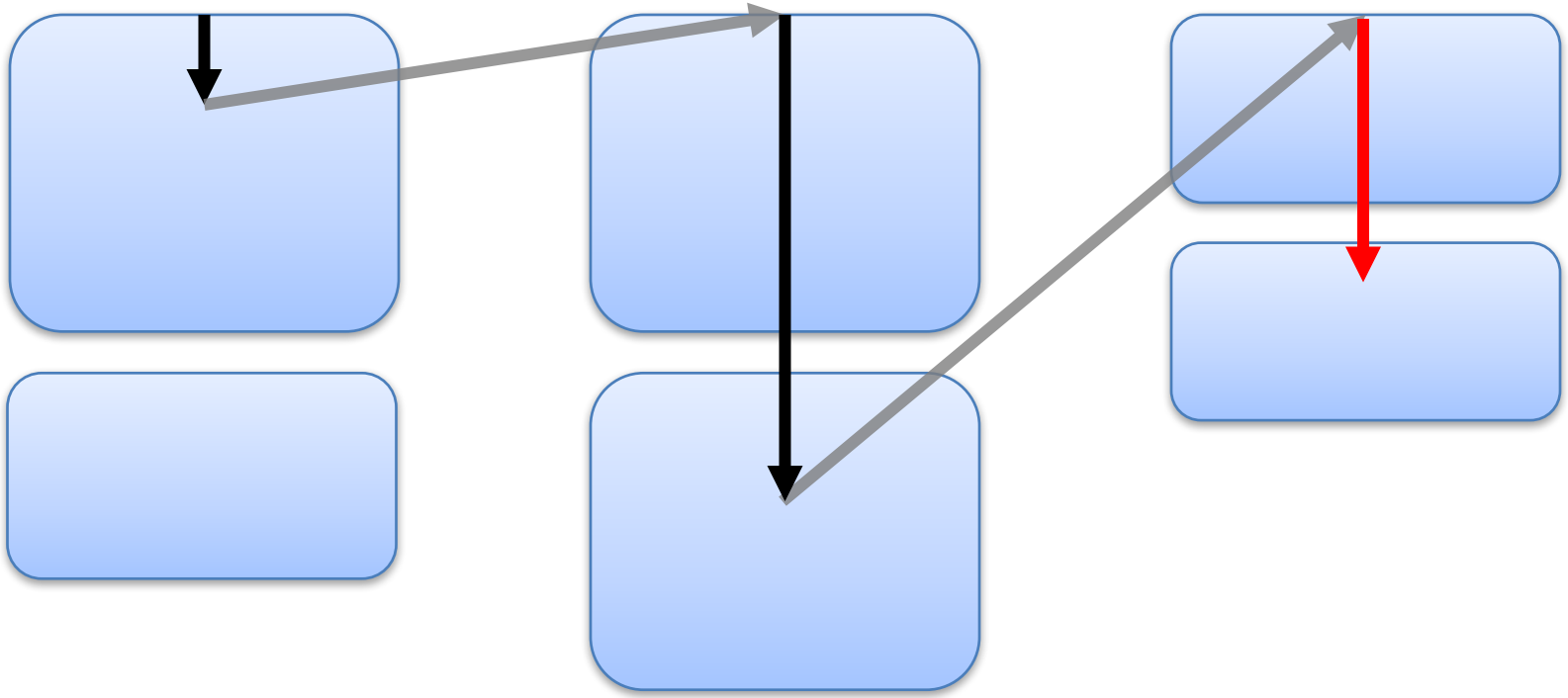


Переключимся на второй

DEVEXPERTS

One-switch стратегия

- Хочется уметь исполнять как-то так:

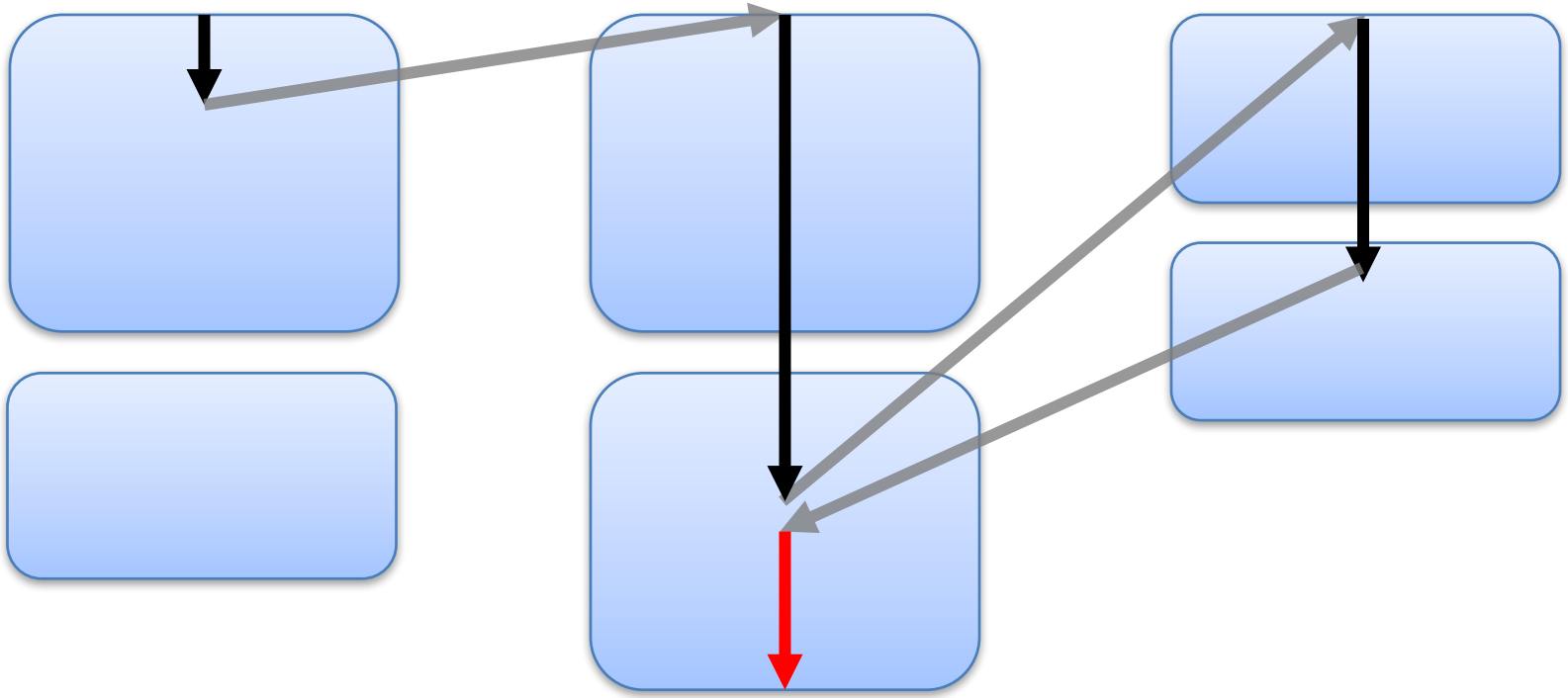


Теперь на третий

DEVEXPERTS

One-switch стратегия

- Хочется уметь исполнять как-то так:

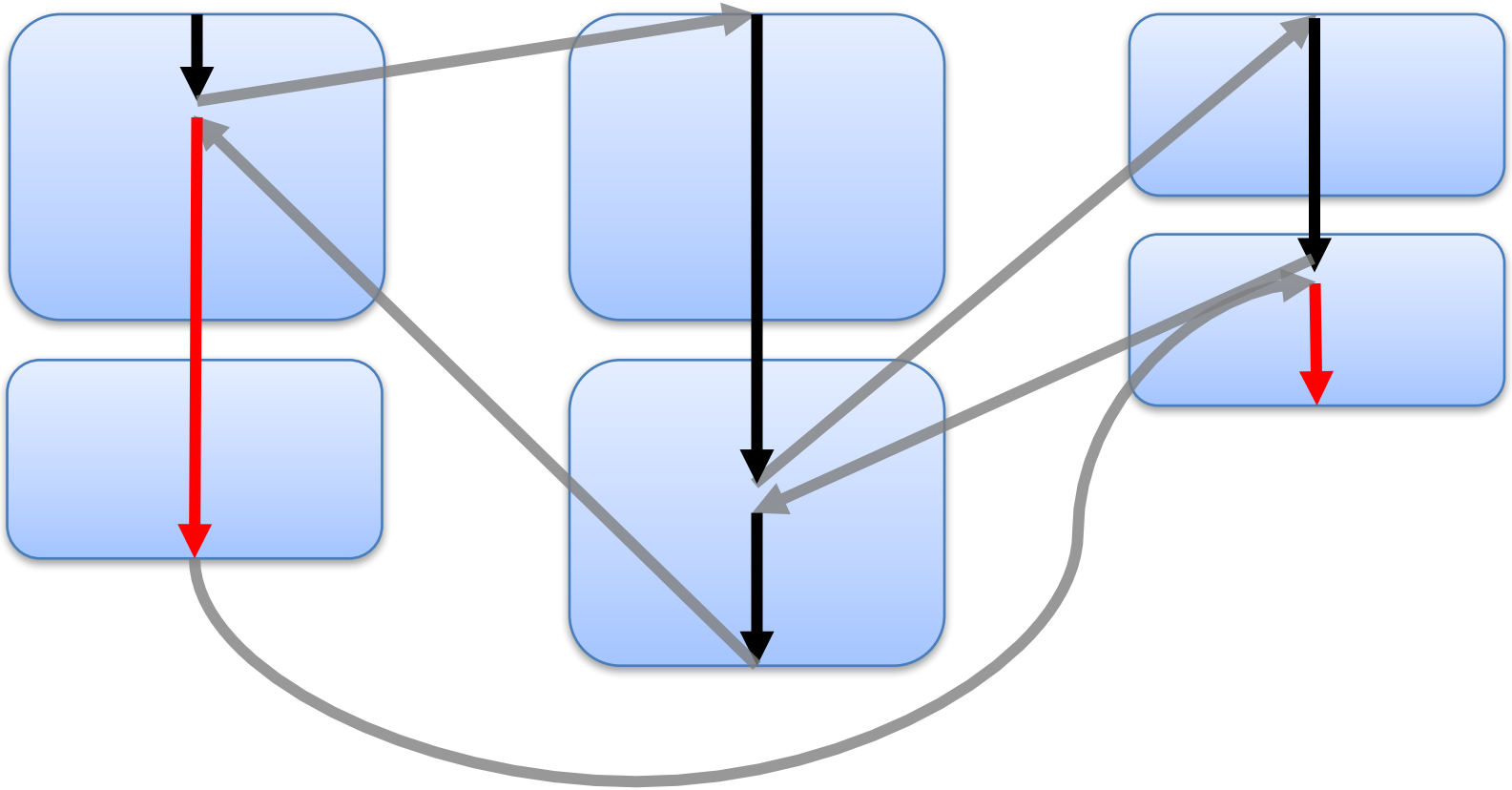


Обратно на второй

DEVEXPERTS

One-switch стратегия

- Хочется уметь исполнять как-то так:

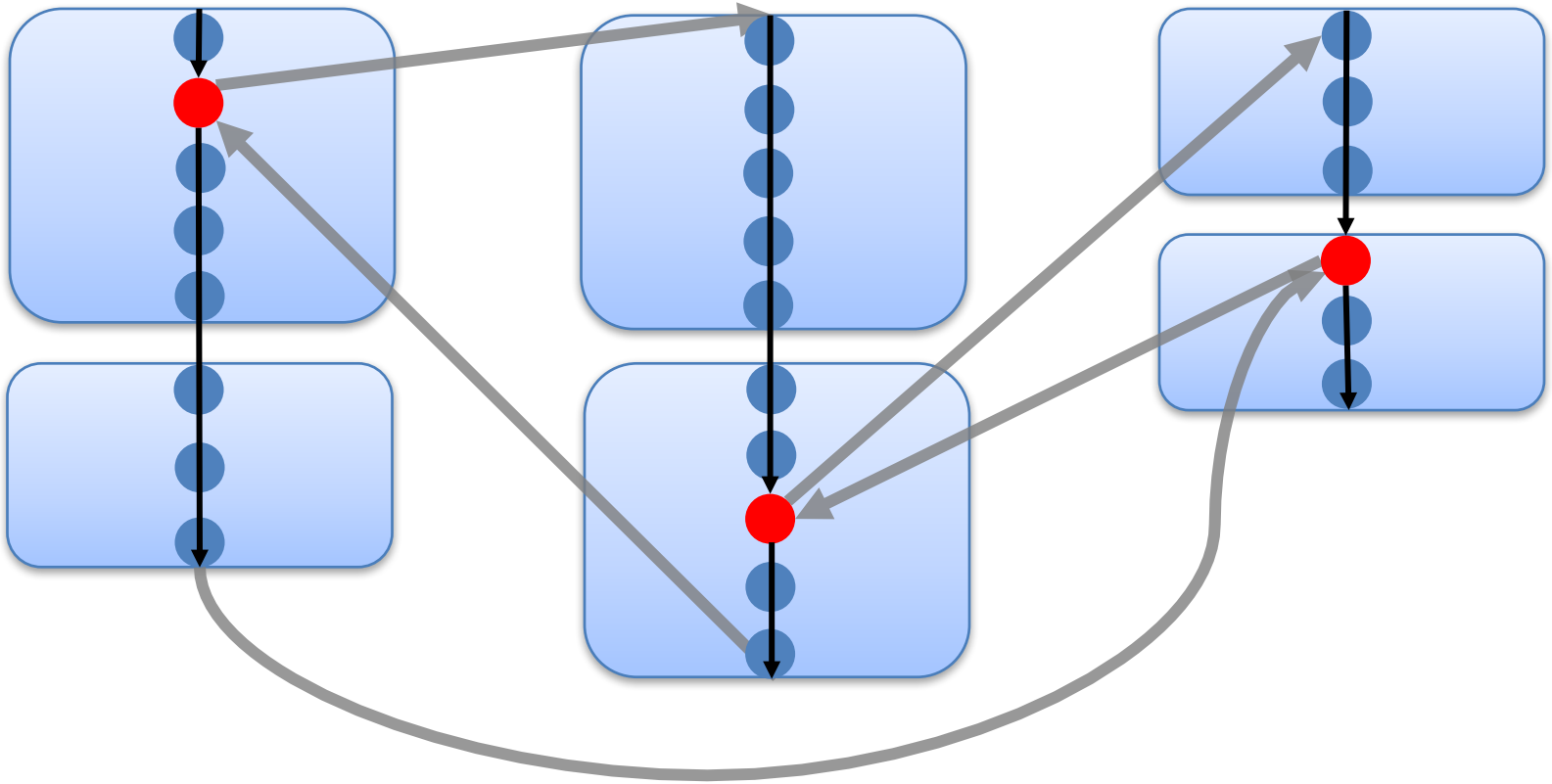


На первый,
на третий

DEVEXPERTS

One-switch: точки переключения

- Где потенциальные точки переключения?



DEVEXPERTS

One-switch: точки переключения

Доступ к разделяемым переменным:

- Поля
  - `getfield`, `putfield`, `getstatic`, `putstatic`
- Элементы массива
  - `Xaload`, `Xastore`, `X={a,i,d,...}`

DEVEXPERTS

One-switch: приоритезация

- Возможных сценариев переключения может быть много
- Хочется ограничить их количество

One-switch: приоритезация

- Возможных сценариев переключения может быть много
- Хочется ограничить их количество

Ограничим количество исполнений, выполняя более «интересные» раньше

DEVEXPERTS

Переключение потоков

- В управляемых стратегиях исполнение де-факто **последовательное**
- Много ресурсов тратится на переключение потоков

DEVEXPERTS

Переключение потоков

- В управляемых стратегиях исполнение де-факто **последовательное**
- Много ресурсов тратится на переключение потоков

Идея: легковесные потоки!

Легковесные потоки

- Fibers, green threads, [co,go,…]routines, …
- Фреймворк Quasar
  - http://www.paralleluniverse.co/quasar/
- При небольшой глубине стека переключение потоков намного дешевле
  - Наш случай!

DEVEXPERTS

Стресс vs Управляемые

- Управляемые стратегии ещё в разработке
- One-switch чуть медленнее Stress режима
- One-switch + Fibers = в 1.5 раза быстрее
- One-switch + Fibers + Parallel = ???

Стресс vs Управляемые

- Управляемые стратегии ещё в разработке
- One-switch медленнее Stress режима
- One-switch + Fibers = примерно так же
- One-switch + Fibers + Parallel = ???

В управляемых стратегиях нет гонок!

DEVEXPERTS

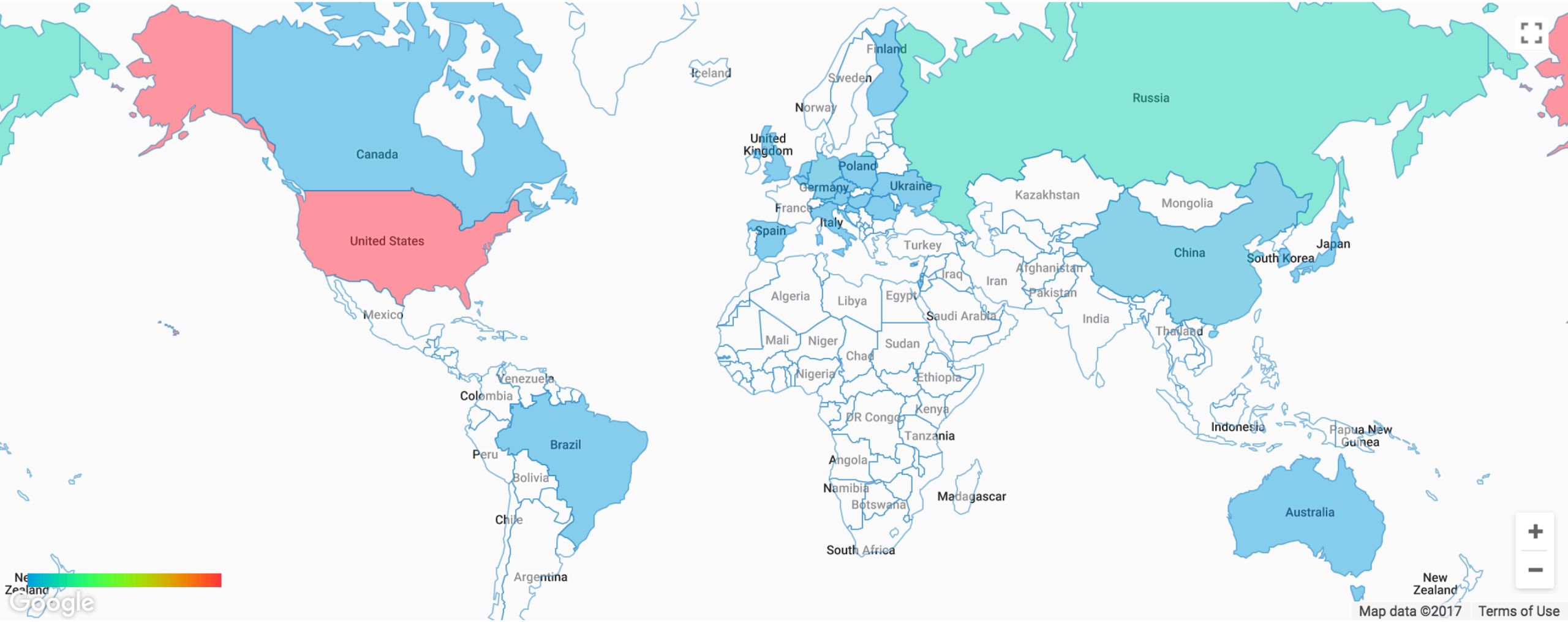
Итого

- Тестировать многопоточные программы непросто, но теперь есть Lin-Check
  - https://github.com/Devexperts/lin-check
- Используется в Devexperts и Jetbrains
  - И еще кто-то скрывается ☺

DEVEXPERTS

lin-check downloads

Total downloads: 2,048



DEVEXPERTS

Бонус

В многопоточном мире **корректность = линеаризуемость**

DEVEXPERTS

Бонус

В многопоточном мире **корректность = линеаризуемость**

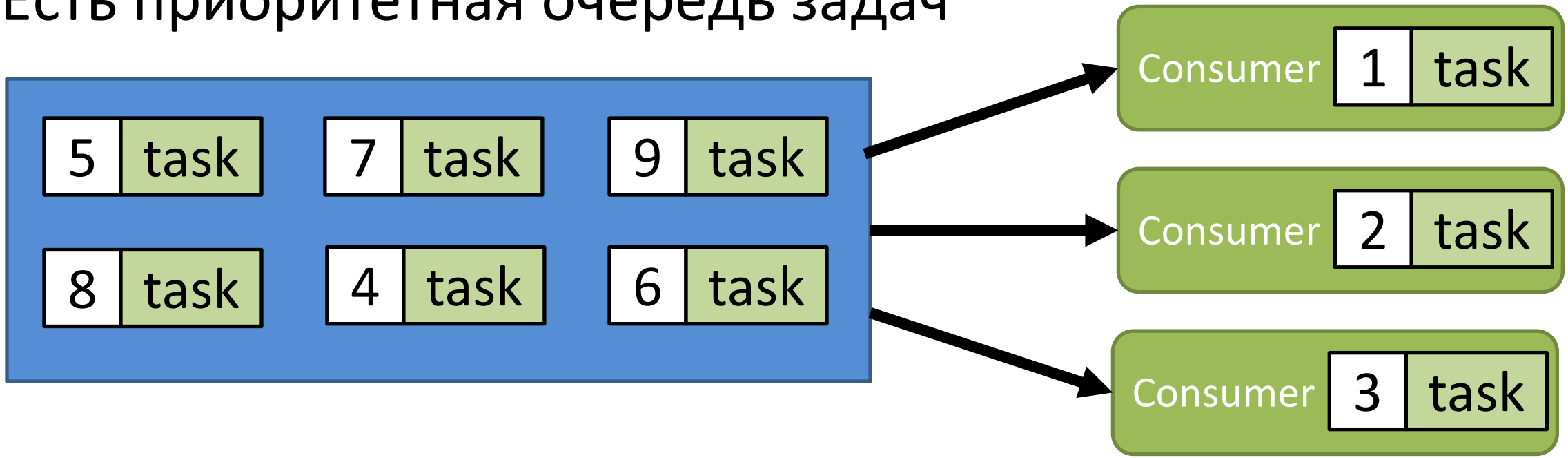
Наглая ложь!*

* Почти всегда правда ☺

DEVEXPERTS

Бонус: relaxed priority queue[1]

- Есть приоритетная очередь задач

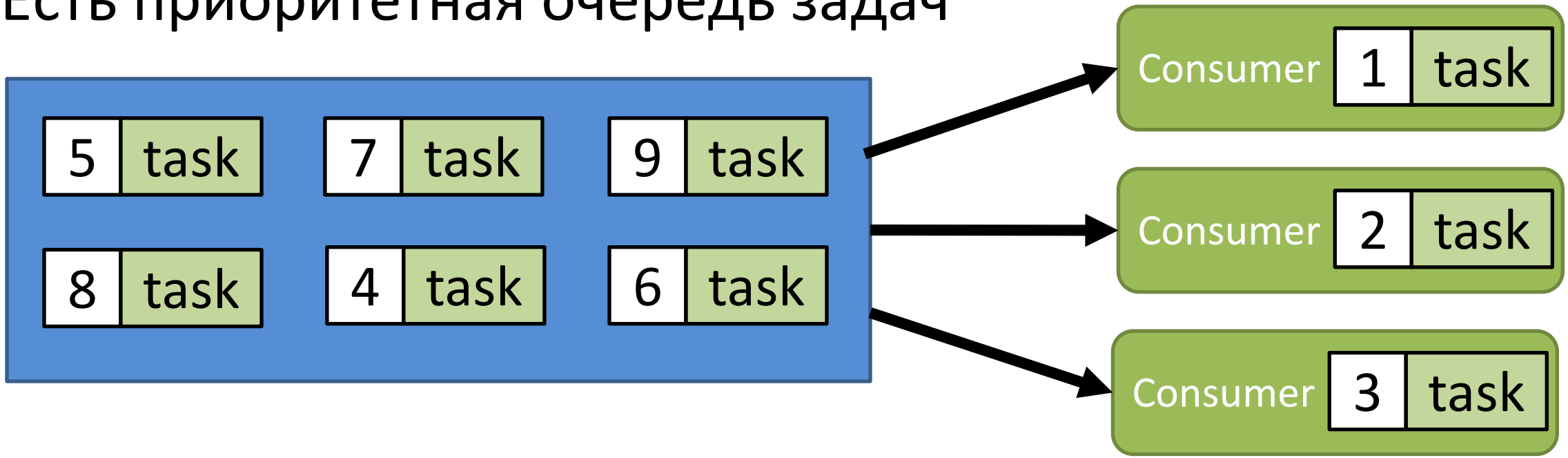


[1] Alistarh, Dan, et al. "The SprayList: A scalable relaxed priority queue." *ACM SIGPLAN Notices* 50.8 (2015): 11-20.

DEVEXPERTS

Бонус: relaxed priority queue

- Есть приоритетная очередь задач

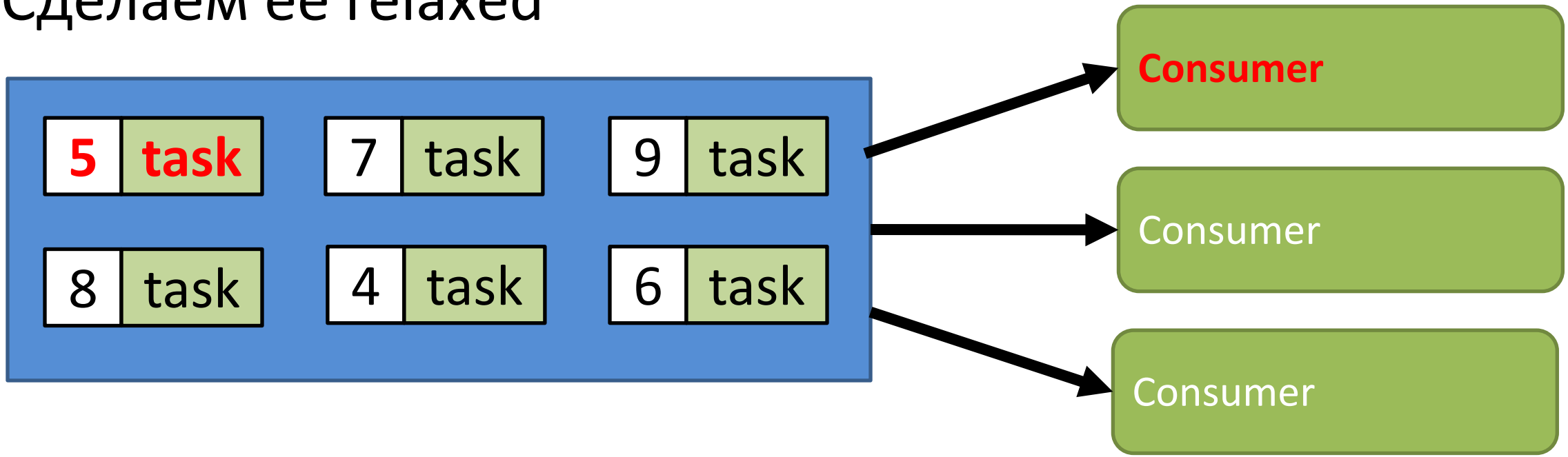


Выполняем задачи **параллельно** ⇒
можем выполнить их в «неправильном» порядке!

DEVEXPERTS

Бонус: relaxed priority queue

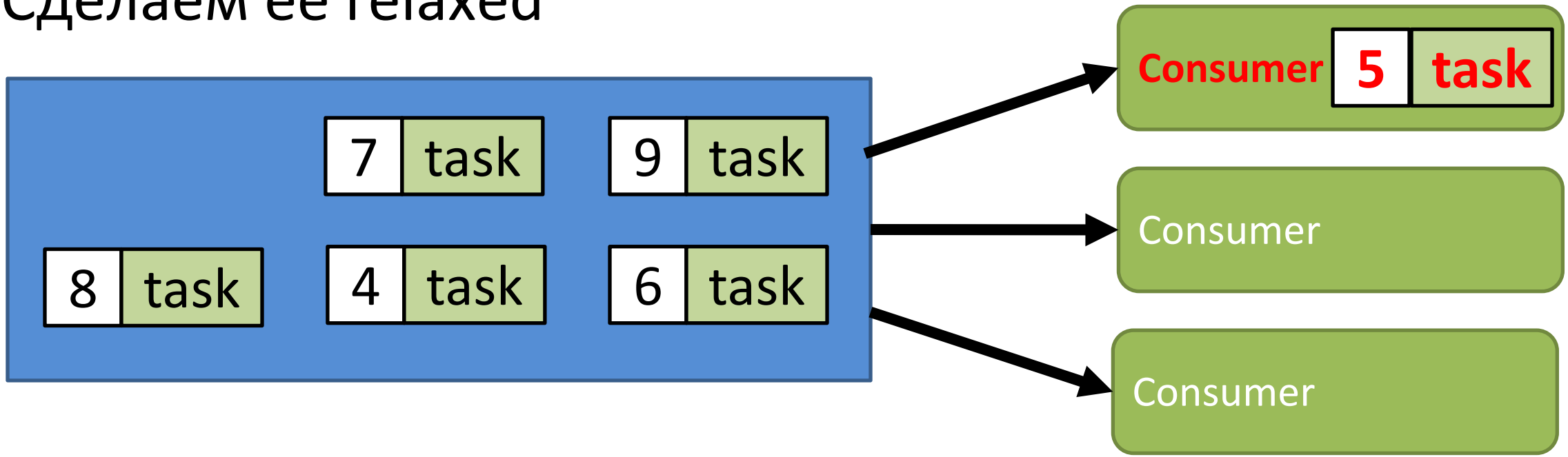
- Сделаем её relaxed



DEVEXPERTS

Бонус: relaxed priority queue

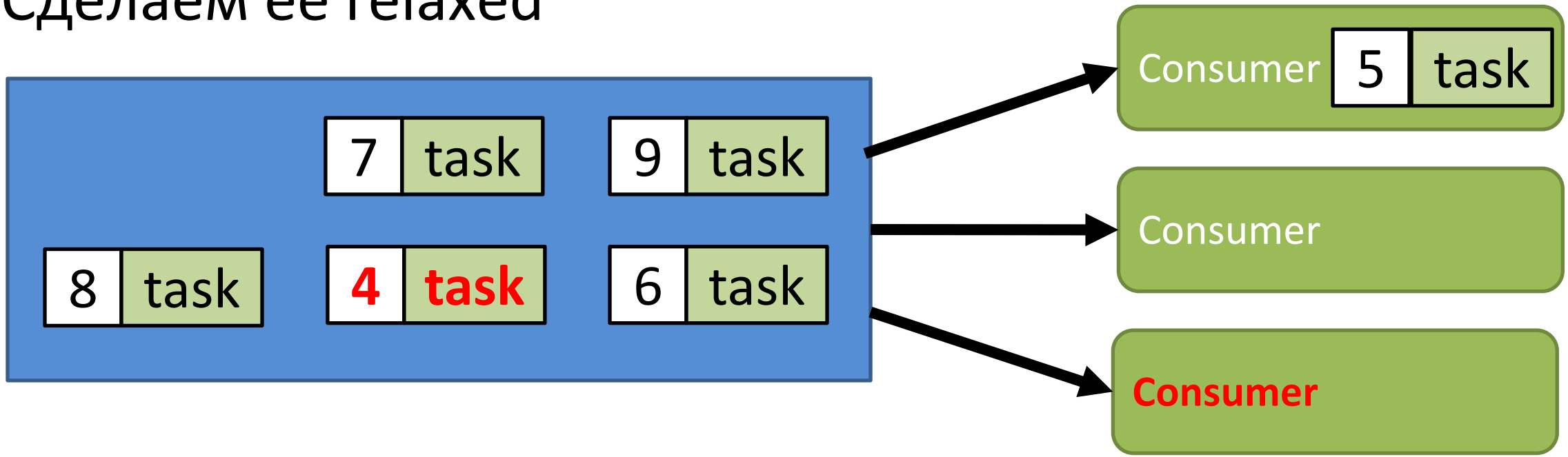
- Сделаем её relaxed



DEVEXPERTS

Бонус: relaxed priority queue

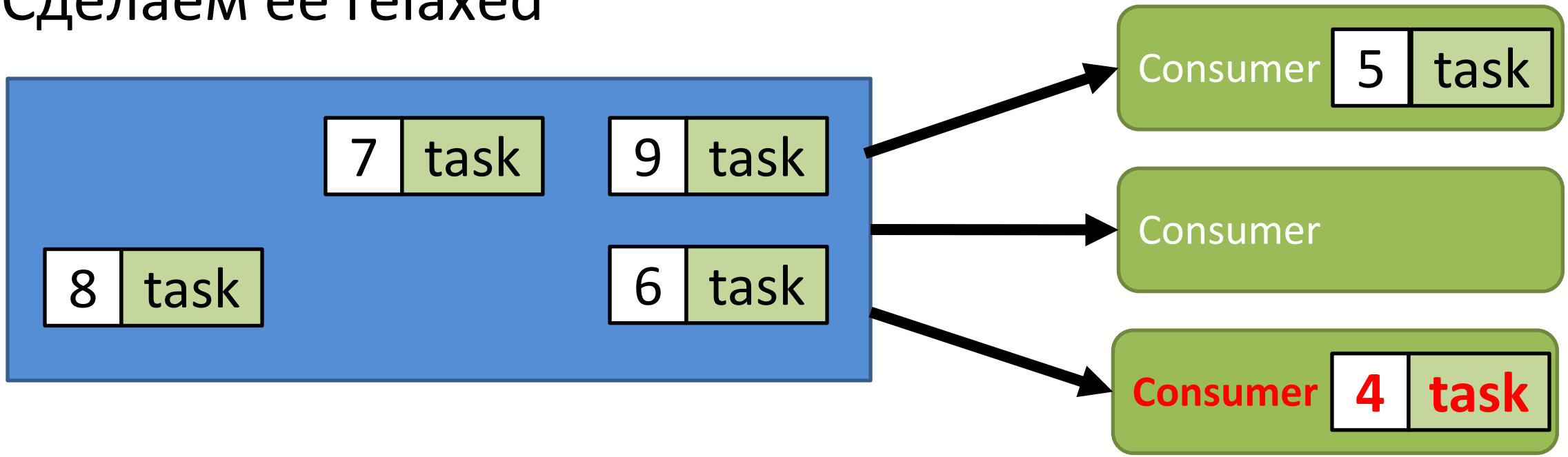
- Сделаем её relaxed



DEVEXPERTS

Бонус: relaxed priority queue

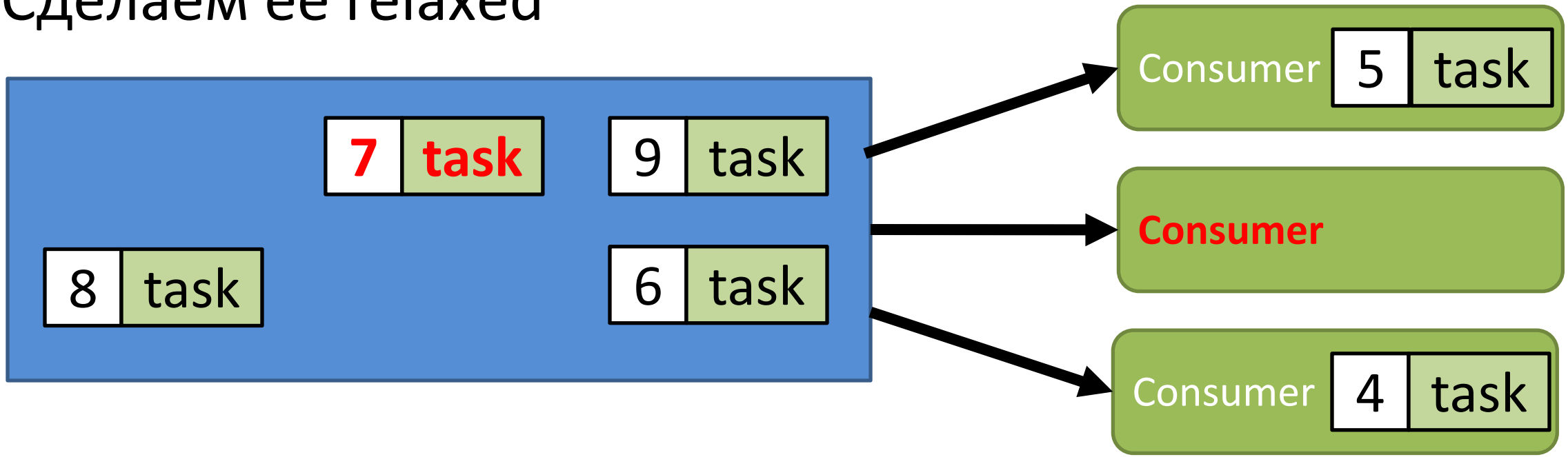
- Сделаем её relaxed



DEVEXPERTS

Бонус: relaxed priority queue

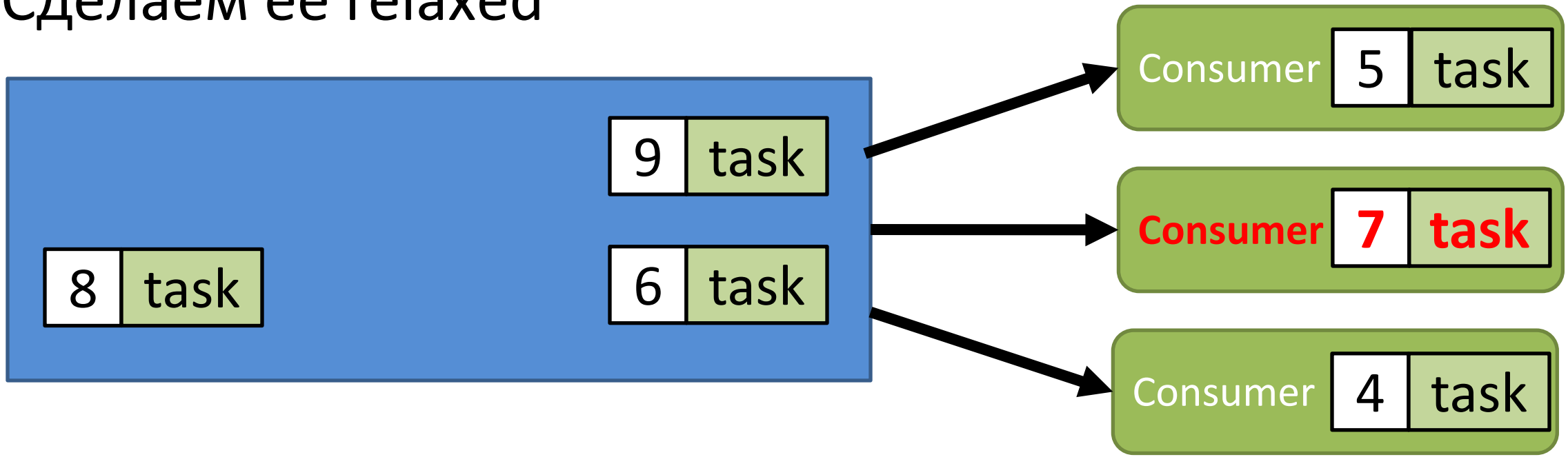
- Сделаем её relaxed



DEVEXPERTS

Бонус: relaxed priority queue

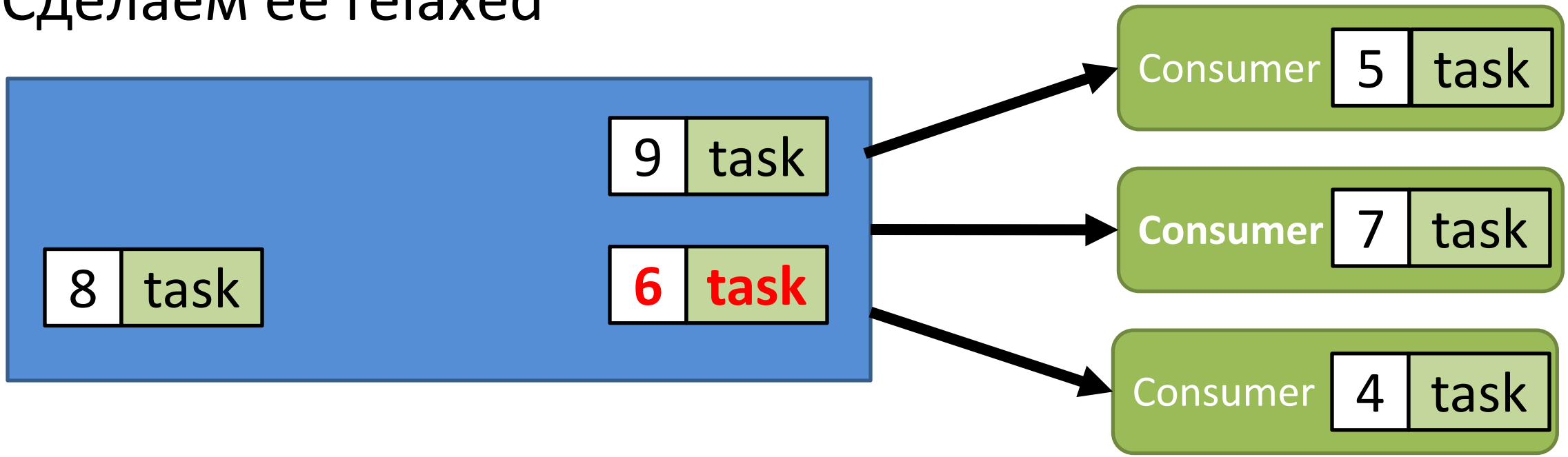
- Сделаем её relaxed



DEVEXPERTS

Бонус: relaxed priority queue

- Сделаем её relaxed



Задачу «6» пока никто не взял.
И это нормально!

DEVEXPERTS

Бонус: Verifier

В Lin-Check можно указать свой верификатор

```
StressCTest(verifier = MyTest.MyVerifier.class)
public class MyTest {
    ...
    public static class MyVerifier extends Verifier {
        public MyVerifier(List<List<Actor>> actorsPerThread,
                          Object testInstance, Method resetMethod) {
            super(actorsPerThread, testInstance, resetMethod); }

        @Override
        public void verifyResults(List<List<Result>> results) {
            ...
        }
    }
}
```


DEVEXPERTS

Бонус: Verifier

В Lin-Check можно указать свой верификатор

```
StressCTest(verifier = MyTest.MyVerifier.class)
public class MyTest {
    ...
    public static class MyVerifier extends Verifier {
        public MyVerifier(List<List<Actor>> actorsPerThread,
                          Object testInstance, Method resetMethod) {
            super(actorsPerThread, testInstance, resetMethod); }

        @Override
        public void verifyResults(List<List<Result>> results) {
            ...
        }
    }
}
```


DEVEXPERTS

Бонус: Verifier

В Lin-Check можно указать свой верификатор

```
StressCTest(verifier = MyTest.MyVerifier.class)
public class MyTest {
    ...
    public static class MyVerifier extends Verifier {
        public MyVerifier(List<List<Actor>> actorsPerThread,
                          Object testInstance, Method resetMethod) {
            super(actorsPerThread, testInstance, resetMethod); }

        @Override
        public void verifyResults(List<List<Result>> results) {
            ...
        }
    }
}
```


DEVEXPERTS

Бонус: Verifier

В Lin-Check можно указать свой верификатор

```
StressCTest(verifier = MyTest.MyVerifier.class)
public class MyTest {
    ...
    public static class MyVerifier extends Verifier {
        public MyVerifier(List<List<Actor>> actorsPerThread,
                          Object testInstance, Method resetMethod) {
            super(actorsPerThread, testInstance, resetMethod); }

        @Override
        public void verifyResults(List<List<Result>> results) {
            ...
        }
    }
}
```


DEVEXPERTS

Бонус: Verifier

В Lin-Check можно указать свой верификатор

```
StressCTest(verifier = MyTest.MyVerifier.class)
public class MyTest {
    ...
    public static class MyVerifier extends Verifier {
        public MyVerifier(List<List<Actor>> actorsPerThread,
                          Object testInstance, Method resetMethod) {
            super(actorsPerThread, testInstance, resetMethod); }

        @Override
        public void verifyResults(List<List<Result>> results) {
            ...
        }
    }
}
```


DEVEXPERTS

Спасибо за внимание!

Никита Коваль

nkoval@devexperts.com
twitter.com/nkoval_

DEVEXPERTS