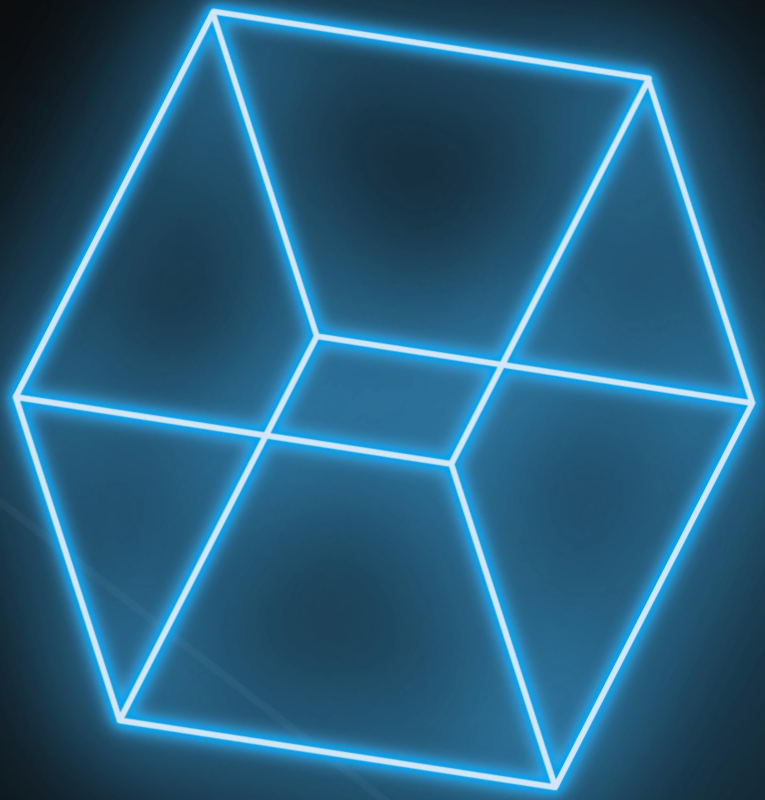
Нестандартные

расширения модели памяти на практике



МИР Plat.Form

Обо мне

linkedin.com/in/alantsov

Мир Plat.Form

mir-platform.ru

Все примеры и бенчмарки

github.com/lantalex/semantics-sandbox

В предыдущей серии ...

youtu.be/UZbPOtEgcoY

- **Семантика**
  спецификатор операции чтения/записи,
  который дает дополнительные гарантии
- **Если нет новых семантик**
  поведение программы полностью
  описывается существующей JMM
- **VarHandle**
  позволяет выбрать семантику
  при операциях записи/чтения
- **Opaque и Acquire/Release**
  две новых промежуточных семантики

Карта семантик

plain	⊆	opaque	⊆	acq-rel	⊆	volatile
type-safety		progress		causality		consensus
		coherence				
		bitwise atomicity				

Как эти семантики используются на практике?

Синглтон

правильный и не очень

Синглтон: правильный

```
class SingletonFactory {
    private static volatile MyObject instance;

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42);
            }
        }
        return instance;
    }
}
```

Синглтон: правильный

```
class SingletonFactory {
    private static volatile MyObject instance

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42);
            }
        }
        return instance;
    }
}
```

Синглтон: правильный

```
class SingletonFactory {
    private static volatile MyObject instance

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42)
            }
        }
        return instance
    }
}
```

Синглтон: правильный

```
class SingletonFactory {
    private static volatile MyObject instance

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42)
            }
        }
        return instance
    }
}
```

Синглтон: правильный

```
class SingletonFactory {
    private static volatile MyObject instance

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42)
            }
        }
        return instance
    }
}
```

Синглтон: правильный и не очень

```
class SingletonFactory {
    private static volatile MyObject instance

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42)
            }
        }
        return instance
    }
}
```

Синглтон: правильный и не очень

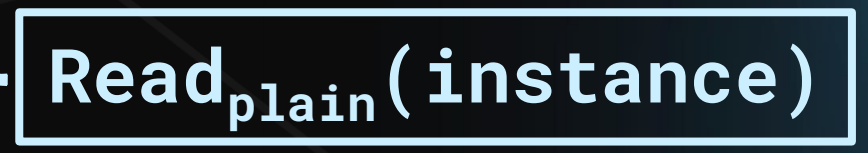
```
class SingletonFactory {
    private static volatile MyObject instance

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42)
            }
        }
        return instance
    }
}
```

Синглтон: правильный и не очень

```
class SingletonFactory {
    private static MyObject instance

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42)
            }
        }
        return instance
    }
}
```

Синглтон: правильный и не очень

```
class SingletonFactory {
    private static MyObject instance

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42)
            }
        }
        return instance
    }
}
```



Когерентность

```
MyObject instance = null
```

Thread 1

```
instance = new MyObject(42)
```

Thread 2

```
MyObject ref1 = instance
MyObject ref2 = instance

if (ref1 != null &&
        ref2 == null ) {

    throw new Error("WTF???")
}
```

Может ли произойти исключение?

Да!

Карта семантик

plain ⊆	opaque ⊆	acq-rel ⊆	volatile
type-safety	progress	causality	consensus
	coherence		
	bitwise atomicity		

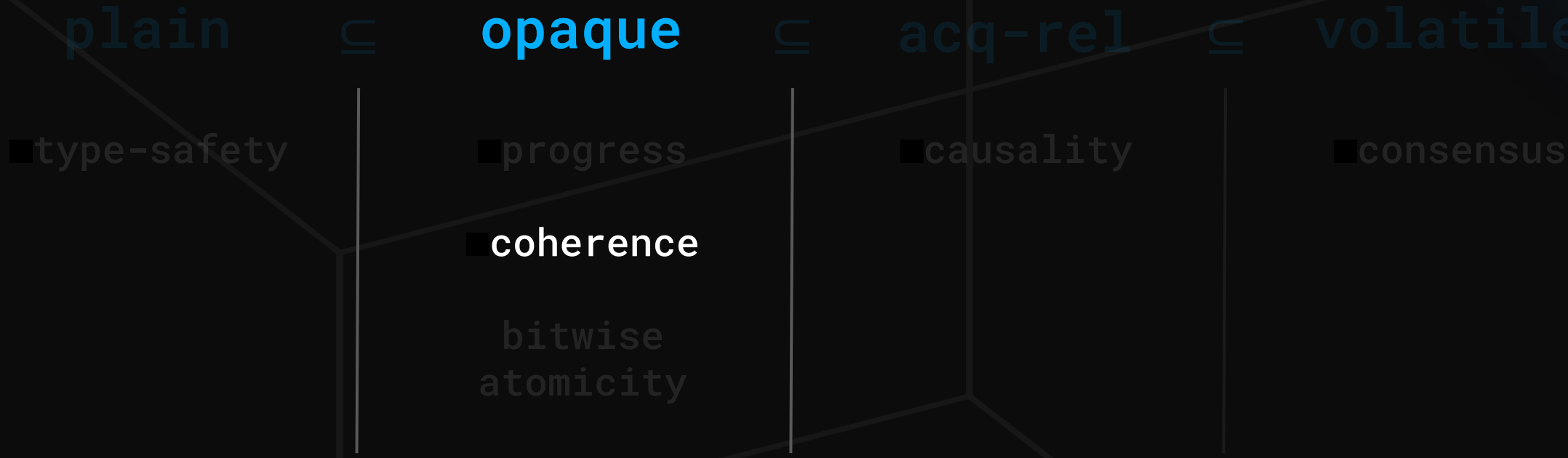
Карта семантик

plain ⊆ opaque ⊆ acq-rel ⊆ volatile

- type-safety
- progress
- coherence
- causality
- consensus

bitwise
atomicity

Карта семантик



Coherence: все операции чтения/записи по одной конкретной переменной **выполняются в глобальном порядке**, консистентном исходному коду

Когерентность

```
MyObject instance = null
VarHandle INSTANCE = ...
```

Thread 1

```
INSTANCE.setOpaque(new MyObject(42))
```

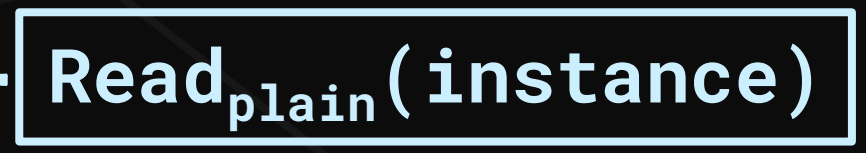
Может ли произойти исключение? Нет!

Thread 2

```
MyObject ref1 = INSTANCE.getOpaque()
MyObject ref2 = INSTANCE.getOpaque()

if (ref1 != null &&
        ref2 == null ) {

    throw new Error("WTF???")
}
```

Синглтон: чиним проблему № 1

```
class SingletonFactory {
    private static MyObject instance

    public static MyObject get() {
        if (instance == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    instance = new MyObject(42)
            }
        }
        return instance
    }
}
```



Синглтон: чиним проблему № 1

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        if (INSTANCE.getOpaque() == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setOpaque(new MyObject(42))
            }
        }
        return INSTANCE.getOpaque()
    }
}
```

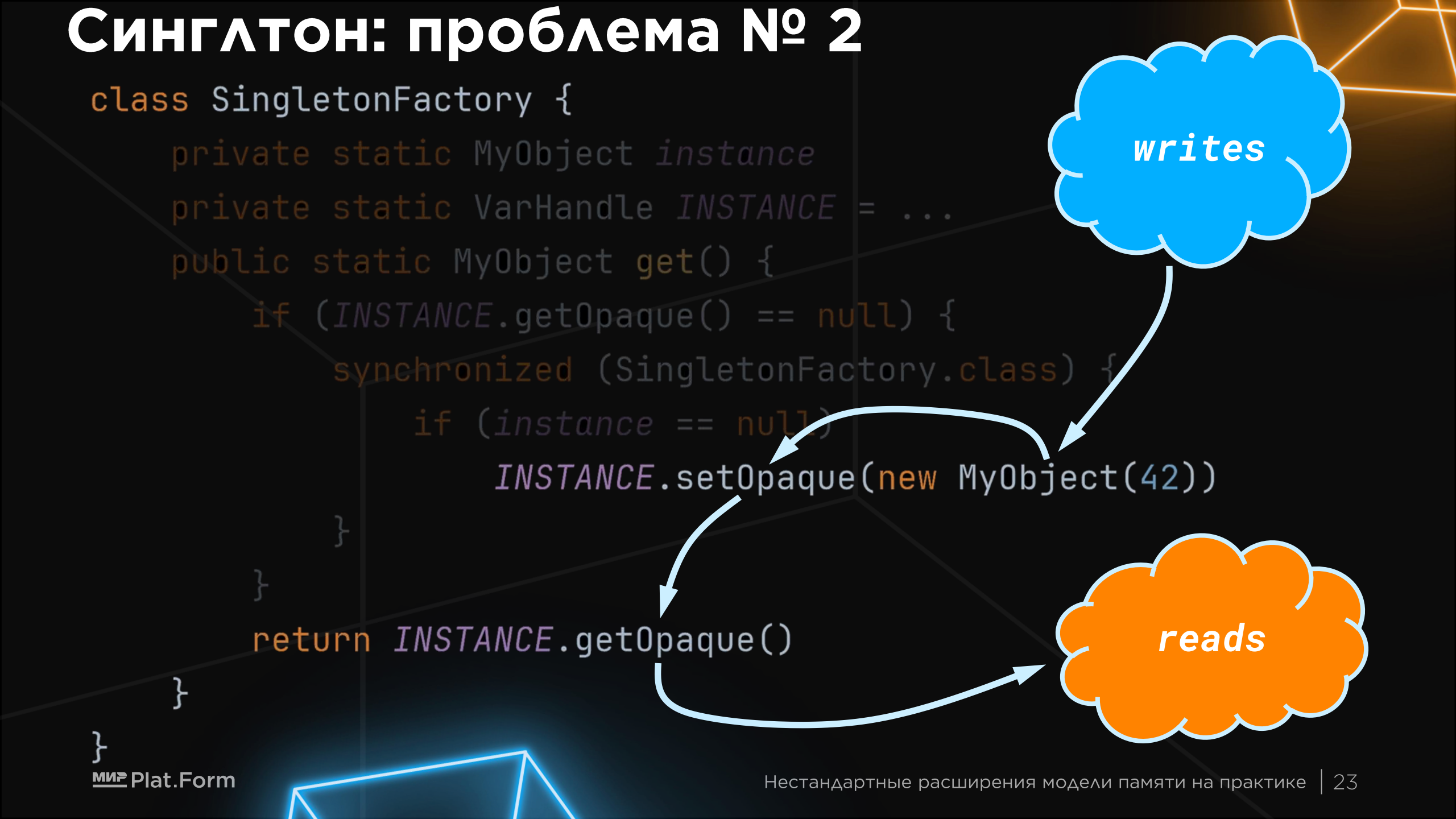
$\text{Read}_{\text{opaque}}(\text{instance})$

Синглтон: чиним проблему № 1

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        if (INSTANCE.getOpaque() == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setOpaque(new MyObject(42))
            }
        }
        return INSTANCE.getOpaque()
    }
}
```

Синглтон: проблема № 2

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        if (INSTANCE.getOpaque() == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setOpaque(new MyObject(42))
            }
        }
        return INSTANCE.getOpaque()
    }
}
```



Причинность

```
MyObject instance = null
VarHandle INSTANCE = ...
```

Thread 1

```
MyObject o = new MyObject(...)
o.someField = 42
INSTANCE.setOpaque(o)
```

Thread 2

```
MyObject ref
do {
    ref = INSTANCE.getOpaque()
} while (ref == null)

assert ref.someField == 42
```

Может ли упасть проверка?

Да!

Карта семантик

plain	⊆	opaque	⊆	acq-rel	⊆	volatile
type-safety		progress		causality		consensus
		coherence				
		bitwise atomicity				

Карта семантик

plain ⊆ opaque ⊆ acq-rel ⊆ volatile

- type-safety
- progress
- coherence

bitwise
atomicity

- causality
- consensus

Карта семантик

plain ⊆	opaque ⊆	acq-rel ⊆	volatile
type-safety	progress	causality	consensus
	coherence		
	bitwise atomicity		

Causality: все операции, предшествующие *release*-записи, **видны для всех операций**, идущих после парного *acquire*-чтения

Причинность

```
MyObject instance = null
VarHandle INSTANCE = ...
```

Thread 1

```
MyObject o = new MyObject(...)
o.someField = 42
INSTANCE.setRelease(o)
```

Thread 2

```
MyObject ref
do {
    ref = INSTANCE.getAcquire()
} while (ref == null)

assert ref.someField == 42
```

Может ли упасть проверка?

Нет!

Синглтон: чиним проблему № 2

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        if (INSTANCE.getOpaque() == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setOpaque(new MyObject(42))
            }
        }
        return INSTANCE.getOpaque()
    }
}
```

Синглтон: чиним проблему № 2

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        if (INSTANCE.getOpaque() == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setOpaque(new MyObject(42))
            }
        }
        return INSTANCE.getOpaque()
    }
}
```

Синглтон: чиним проблему № 2

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        if (INSTANCE.getOpaque() == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setRelease(new MyObject(42))
            }
        }
        return INSTANCE.getAcquire()
    }
}
```

Синглтон: чиним проблему № 2

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        if (INSTANCE.getOpaque() == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setRelease(new MyObject(42))
            }
        }
        return INSTANCE.getAcquire()
    }
}
```

Синглтон: финальные штрихи

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        if (INSTANCE.getOpaque() == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setRelease(new MyObject(42))
            }
        }
        return INSTANCE.getAcquire()
    }
}
```

Синглтон: финальные штрихи

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        if (INSTANCE.getOpaque() == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setRelease(new MyObject(42))
            }
        }
        return INSTANCE.getAcquire()
    }
}
```

Синглтон: финальные штрихи

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
```

Синглтон: финальные штрихи

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        MyObject p = INSTANCE.getAcquire()
```

Синглтон: финальные штрихи

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        MyObject p = INSTANCE.getAcquire()
        if (p == null) {



        }
        return p
```

Синглтон: финальные штрихи

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        MyObject p = INSTANCE.getAcquire()
        if (p == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setRelease(new MyObject(42))
                p = instance
            }
        }
        return p
```

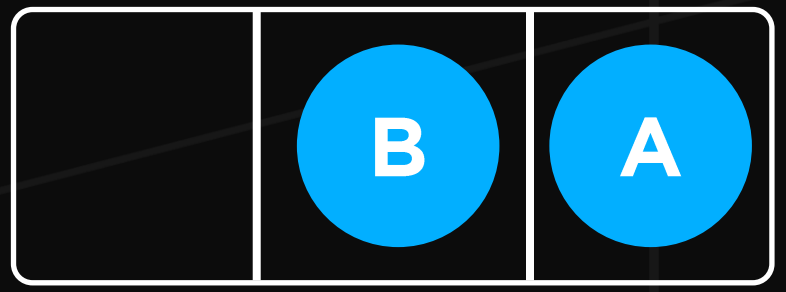
Синглтон: финальные штрихи

```
class SingletonFactory {
    private static MyObject instance
    private static VarHandle INSTANCE = ...
    public static MyObject get() {
        MyObject p = INSTANCE.getAcquire()
        if (p == null) {
            synchronized (SingletonFactory.class) {
                if (instance == null)
                    INSTANCE.setRelease(new MyObject(42))
                p = instance
            }
        }
        return p
```

SPSC-очередь
на пути к Disruptor

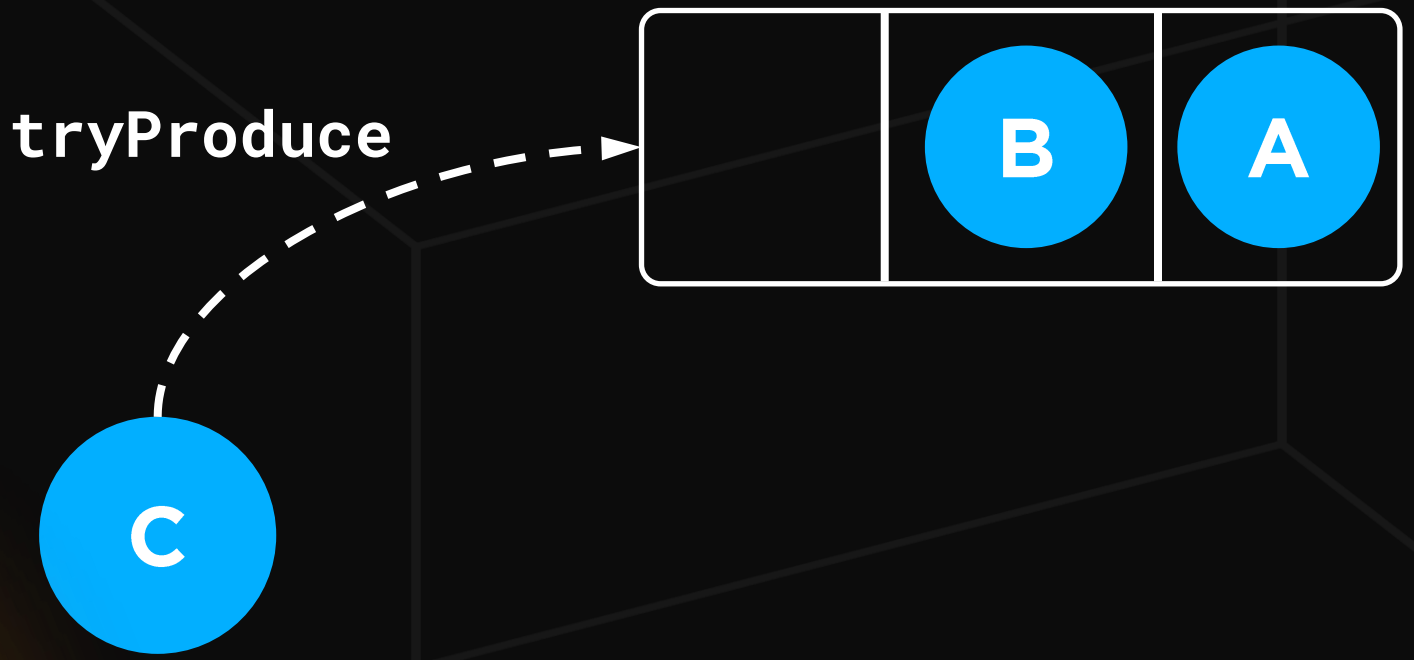
Простая очередь

Single Producer/Single Consumer Bounded Queue



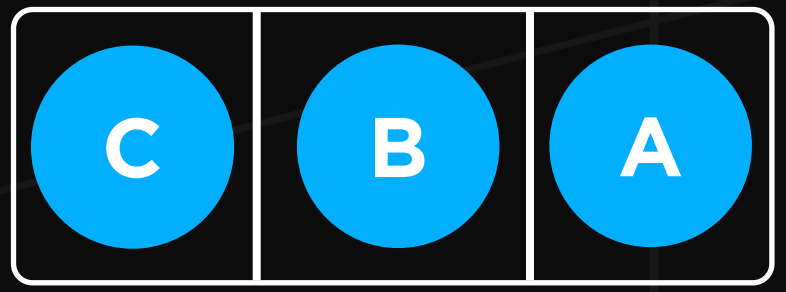
Простая очередь

Single Producer/Single Consumer Bounded Queue



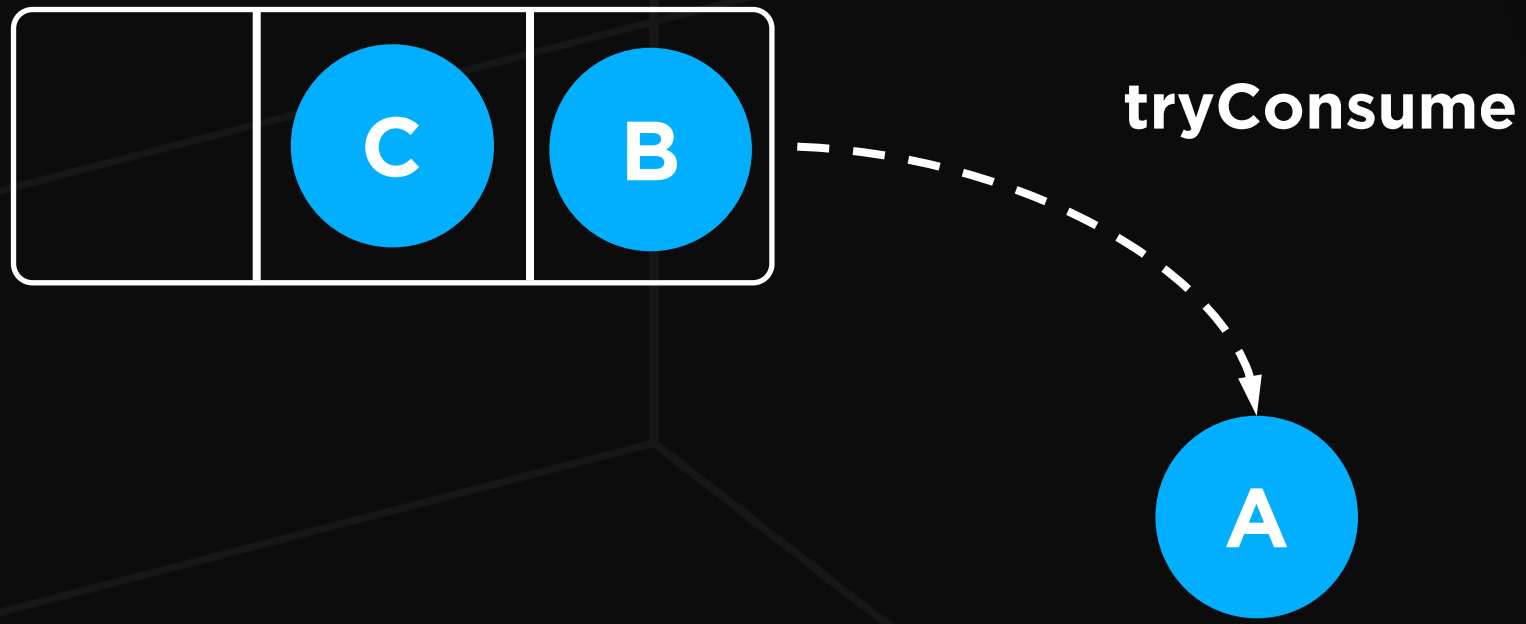
Простая очередь

Single Producer/Single Consumer Bounded Queue



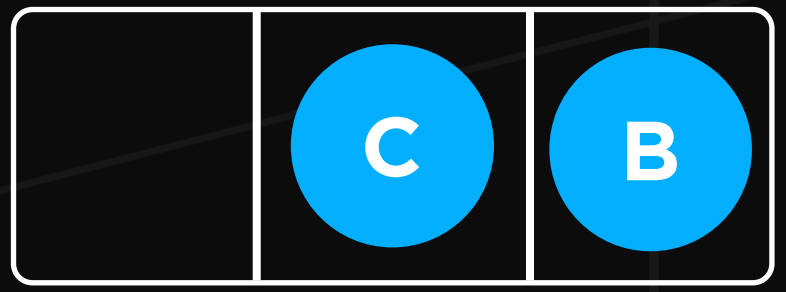
Простая очередь

Single Producer/Single Consumer Bounded Queue



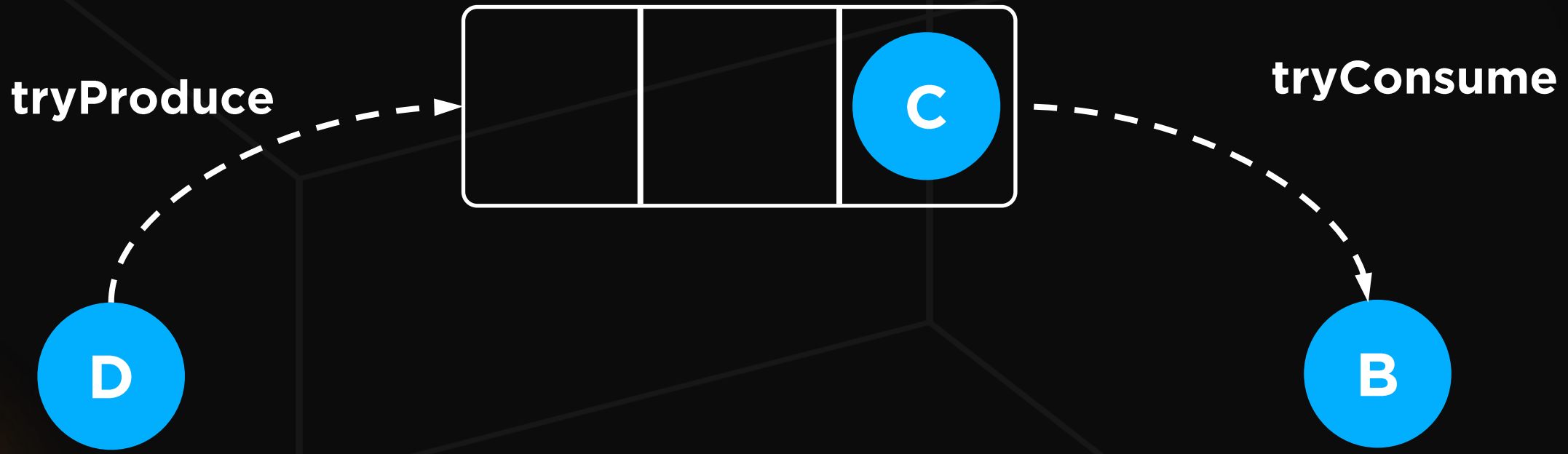
Простая очередь

Single Producer/Single Consumer Bounded Queue



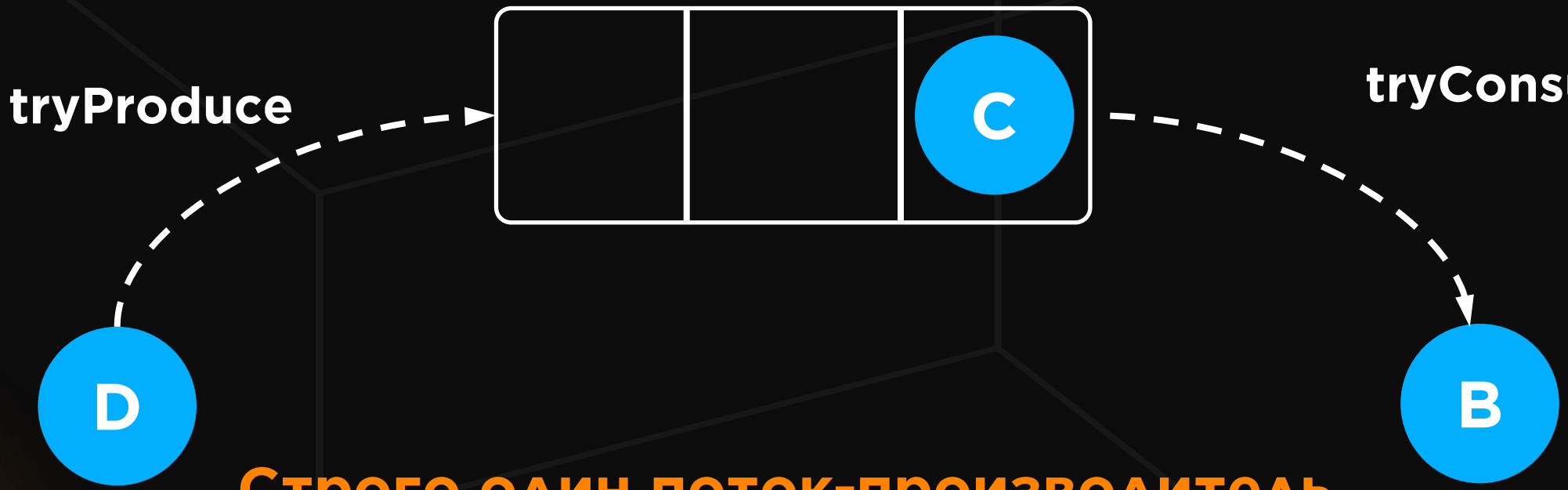
Простая очередь

Single Producer/Single Consumer Bounded Queue



Простая очередь

Single Producer/Single Consumer Bounded Queue



**Строго один поток-производитель,
строго один поток-потребитель!**

Базовый интерфейс

```
interface SPSC_BoundedQueue<E> {
    //метод для производителя
    boolean tryProduce(E value)

    //метод для потребителя
    boolean tryConsume(Consumer<E> consumer)
}
```

Сценарий использования

```
SPSC_BoundedQueue<E> queue = new SPSC_BoundedQueue<>(...)
```

Producing thread

```
E element = new E(...)
init(element)

queue.tryProduce(element)
```

Consuming thread

```
queue.tryConsume(element ->
        print(element)
)
```

Сценарий использования

```
SPSC_BoundedQueue<E> queue = new SPSC_BoundedQueue<>(...)
```

Producing thread


```
queue.tryProduce(element)
```

Consuming thread

```
queue.tryConsume(element ->
        print(element)
)
```

Сценарий использования

```
SPSC_BoundedQueue<E> queue = new SPSC_BoundedQueue<>(...)
```

Producing thread


```
queue.tryProduce(element)
```

Consuming thread

```
queue.tryConsume(
)
```

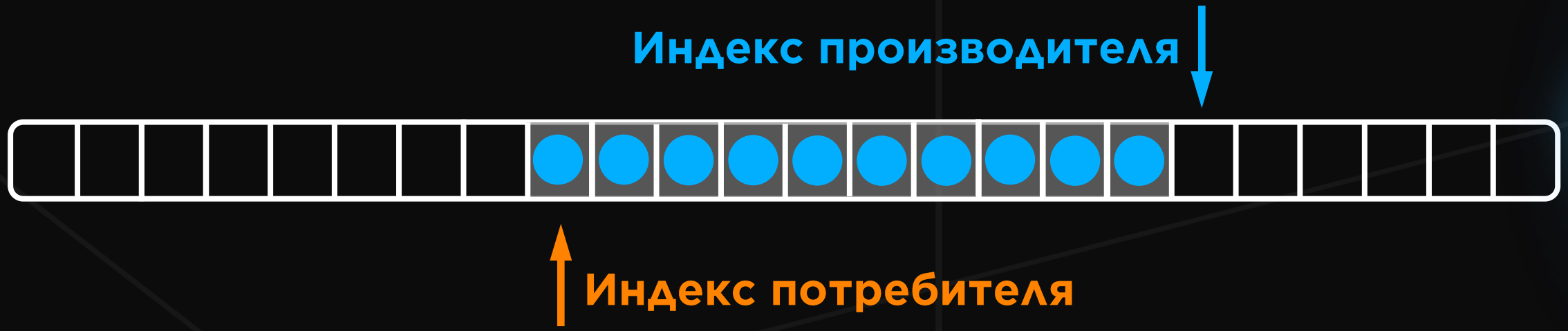
Сценарий использования

```
SPSC_BoundedQueue<E> queue = new SPSC_BoundedQueue<>(...)
```

Producing thread

Consuming thread

Потребитель должен видеть очередной элемент в полностью готовом виде...



- Индекс потребителя **не должен обгонять** индекс производителя
- Потребитель должен учитывать ситуацию, когда **очередь пустая**
- Производитель не должен писать в ячейки, которые потребитель **еще не успел считать**
- Производитель должен учитывать ситуацию, когда **очередь полная**

```
class SPSC_VolatileQueue {

}
```

```
class SPSC_VolatileQueue {
    //индекс потребителя
    volatile int consumerIdx = 0

}
```

```
class SPSC_VolatileQueue {
    //индекс потребителя
    volatile int consumerIdx = 0

    //индекс производителя
    volatile int producerIdx = 0


}
```

```
class SPSC_VolatileQueue {
    //индекс потребителя
    volatile int consumerIdx = 0

    //индекс производителя
    volatile int producerIdx = 0

    //массив для хранения ссылок
    MyObject[] buffer = new MyObject[capacity]
}
```

```
boolean tryProduce(MyObject value) {

}
```

```
boolean tryProduce(MyObject value) {
    int pIdx = producerIdx
    int cIdx = consumerIdx
}
```

```
boolean tryProduce(MyObject value) {
    int pIdx = producerIdx
    int cIdx = consumerIdx

    if (isFull(cIdx, pIdx)) {
        return false
    }


}
```

```
boolean tryProduce(MyObject value) {
    int pIdx = producerIdx
    int cIdx = consumerIdx

    if (isFull(cIdx, pIdx)) {
        return false
    }

    buffer[pIdx] = value


}
```

```
boolean tryProduce(MyObject value) {
    int pIdx = producerIdx
    int cIdx = consumerIdx

    if (isFull(cIdx, pIdx)) {
        return false
    }

    buffer[pIdx] = value

    producerIdx = (pIdx + 1) % capacity
    return true
}
```

```
boolean tryConsume(Consumer<MyObject> consumer) {

}
```

```
boolean tryConsume(Consumer<MyObject> consumer) {
    int cIdx = consumerIdx
    int pIdx = producerIdx


}
```

```
boolean tryConsume(Consumer<MyObject> consumer) {
    int cIdx = consumerIdx
    int pIdx = producerIdx

    if (isEmpty(cIdx, pIdx)) {
        return false
    }


}
```

```
boolean tryConsume(Consumer<MyObject> consumer) {
    int cIdx = consumerIdx
    int pIdx = producerIdx

    if (isEmpty(cIdx, pIdx)) {
        return false
    }

    consumer.accept(buffer[cIdx])


}
```

```
boolean tryConsume(Consumer<MyObject> consumer) {
    int cIdx = consumerIdx
    int pIdx = producerIdx

    if (isEmpty(cIdx, pIdx)) {
        return false
    }

    consumer.accept(buffer[cIdx])

    consumerIdx = (cIdx + 1) % capacity
    return true
}
```

Producing thread

Consuming thread

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

producerIdx =
        (pIdx + 1) % capacity
return true
```

Consuming thread

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

producerIdx =
        (pIdx + 1) % capacity
return true
```

Consuming thread

```
int cIdx = consumerIdx
int pIdx = producerIdx

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumer.accept(buffer[cIdx])

consumerIdx =
        (cIdx + 1) % capacity
return true
```

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

producerIdx =
        (pIdx + 1) % capacity
return true
```

Consuming thread

```
int cIdx = consumerIdx
int pIdx = producerIdx

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumer.accept(buffer[cIdx])

consumerIdx =
        (cIdx + 1) % capacity
return true
```

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

producerIdx =
        (pIdx + 1) % capacity
return true
```

Consuming thread

```
int cIdx = consumerIdx
int pIdx = producerIdx

if (isEmpty(cIdx, pIdx)) {
    return false
}

consum      r[cIdx])

consumerIdx =
        (cIdx + 1) % capacity
return true
```

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

producerIdx =
        (pIdx + 1) % capacity
return true
```

writes

Consuming thread

```
int cIdx = consumerIdx
int pIdx = producerIdx

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumed = buffer[cIdx])

consumerIdx =
        (cIdx + 1) % capacity
return true
```

reads

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

PRODUCER.setRelease(
        (pIdx + 1) % capacity)
return true
```

writes

Consuming thread

```
int cIdx = consumerIdx
int pIdx = producerIdx

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumed = buffer[cIdx])

consumerIdx =
        (cIdx + 1) % capacity
return true
```

reads

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

PRODUCER.setRelease(
        (pIdx + 1) % capacity)
return true
```

writes

Consuming thread

```
int cIdx = consumerIdx
int pIdx = PRODUCER.getAcquire()

if (isEmpty(cIdx, pIdx)) {
    return false
}

consume[cIdx])

consumerIdx =
        (cIdx + 1) % capacity
return true
```

reads

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

PRODUCER.setRelease(
        (pIdx + 1) % capacity)
return true
```

Consuming thread

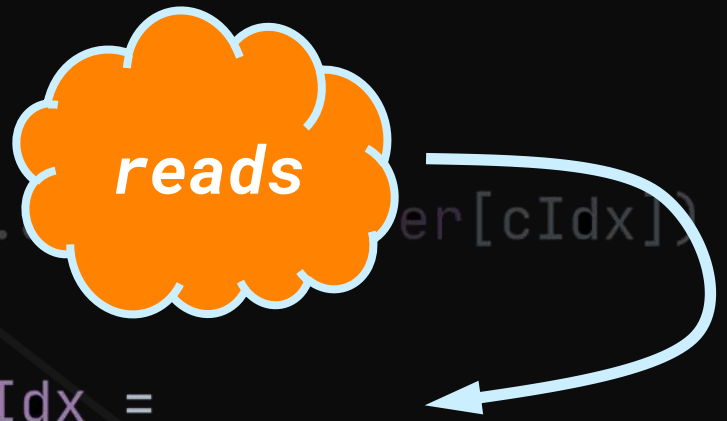
```
int cIdx = consumerIdx
int pIdx = PRODUCER.getAcquire()

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumer.accept(buffer[cIdx])

consumerIdx =
        (cIdx + 1) % capacity
return true
```

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

PRODUCER.setRelease(
        (pIdx + 1) % capacity)
return true
```

Consuming thread

```
int cIdx = consumerIdx
int pIdx = PRODUCER.getAcquire()

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumer.                er[cIdx]

consumerIdx =
        (cIdx + 1) % capacity
return true
```



Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffe      value

PRODUCER.setRelease(
        (pIdx + 1) % capacity)
return true
```

Consuming thread

```
int cIdx = consumerIdx
int pIdx = PRODUCER.getAcquire()

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumer.      er[cIdx]

consumerIdx =
        (cIdx + 1) % capacity
return true
```

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffe      value

PRODUCER.setRelease(
        (pIdx + 1) % capacity)
return true
```

writes

Consuming thread

```
int cIdx = consumerIdx
int pIdx = PRODUCER.getAcquire()

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumer.      er[cIdx]

consumerIdx =
        (cIdx + 1) % capacity
return true
```

reads

Producing thread

```
int pIdx = producerIdx
int cIdx = consumerIdx

if (isFull(cIdx, pIdx)) {
    return false
}

buffe      value

PRODUCER.setRelease(
        (pIdx + 1) % capacity)
return true
```

writes

Consuming thread

```
int cIdx = consumerIdx
int pIdx = PRODUCER.getAcquire()

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumer.      er[cIdx]

CONSUMER.setRelease(
        (cIdx + 1) % capacity)
return true
```

reads

Producing thread

```
int pIdx = producerIdx
int cIdx = CONSUMER.getAcquire()

if (isFull(cIdx, pIdx)) {
    return false
}

buffe      value

PRODUCER.setRelease(
        (pIdx + 1) % capacity)
return true
```

Consuming thread

```
int cIdx = consumerIdx
int pIdx = PRODUCER.getAcquire()

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumer.      er[cIdx]

CONSUMER.setRelease(
        (cIdx + 1) % capacity)
return true
```

Producing thread

```
int pIdx = producerIdx
int cIdx = CONSUMER.getAcquire()

if (isFull(cIdx, pIdx)) {
    return false
}

buffer[pIdx] = value

PRODUCER.setRelease(
        (pIdx + 1) % capacity)
return true
```

Consuming thread

```
int cIdx = consumerIdx
int pIdx = PRODUCER.getAcquire()

if (isEmpty(cIdx, pIdx)) {
    return false
}

consumer.accept(buffer[cIdx])

CONSUMER.setRelease(
        (cIdx + 1) % capacity)
return true
```

Queue benchmark

Тип очереди	Throughput, ops/ms

Queue benchmark

Тип очереди	Throughput, ops/ms
`ArrayBlockingQueue_Wrapper`	6,178

Queue benchmark

Тип очереди	Throughput, ops/ms
`ArrayBlockingQueue_Wrapper`	6,178

Queue benchmark

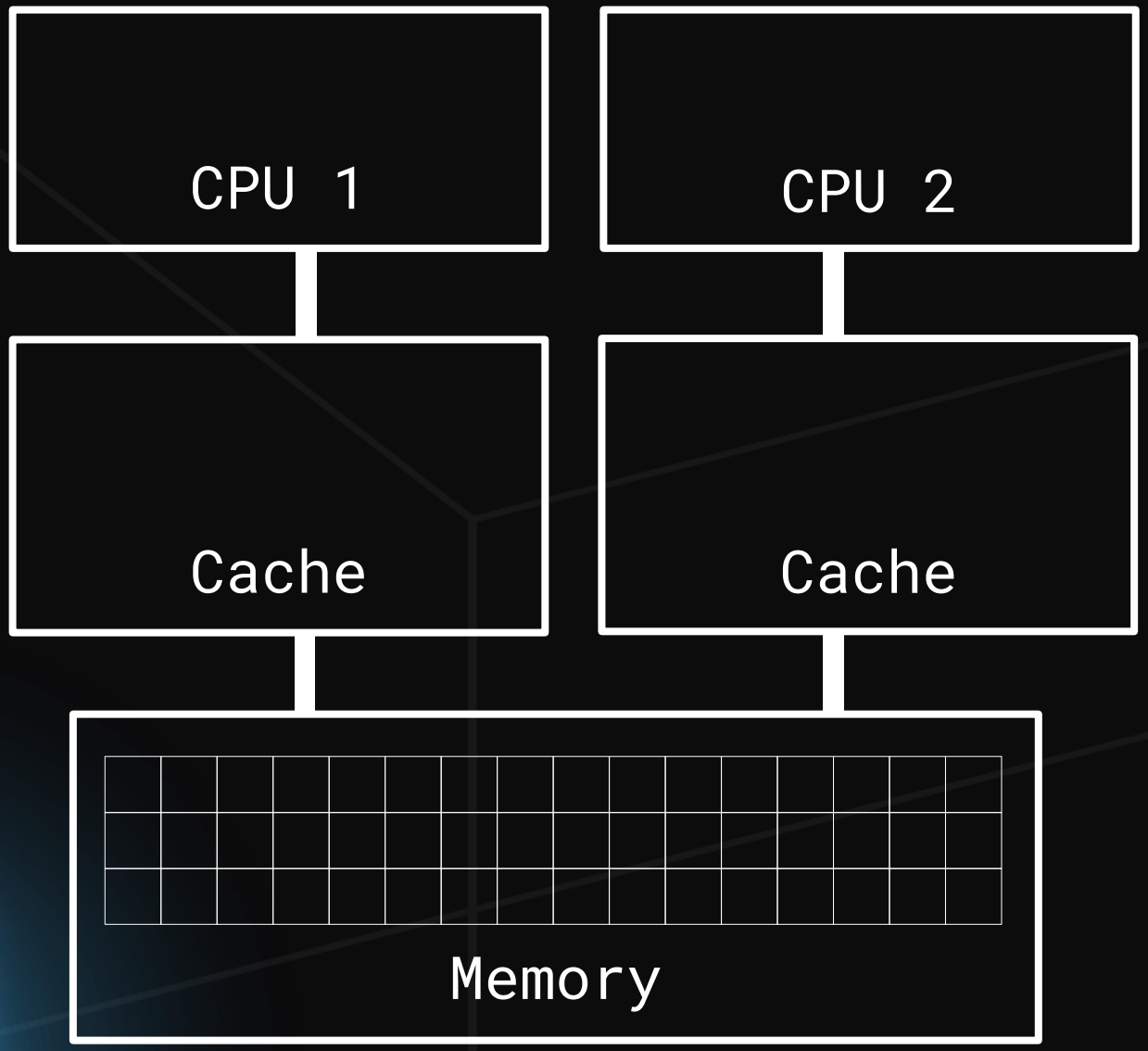
Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002

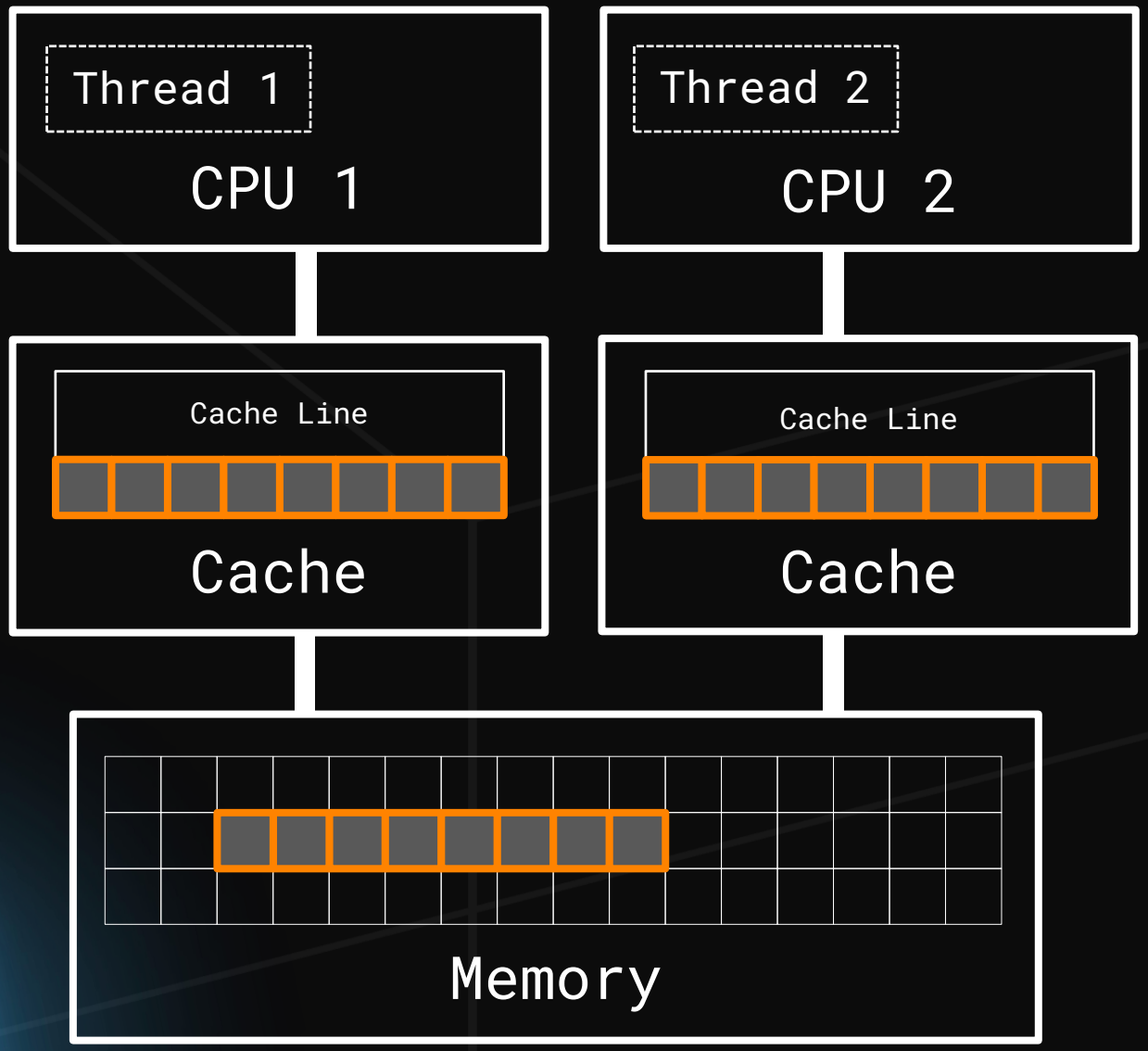
Queue benchmark

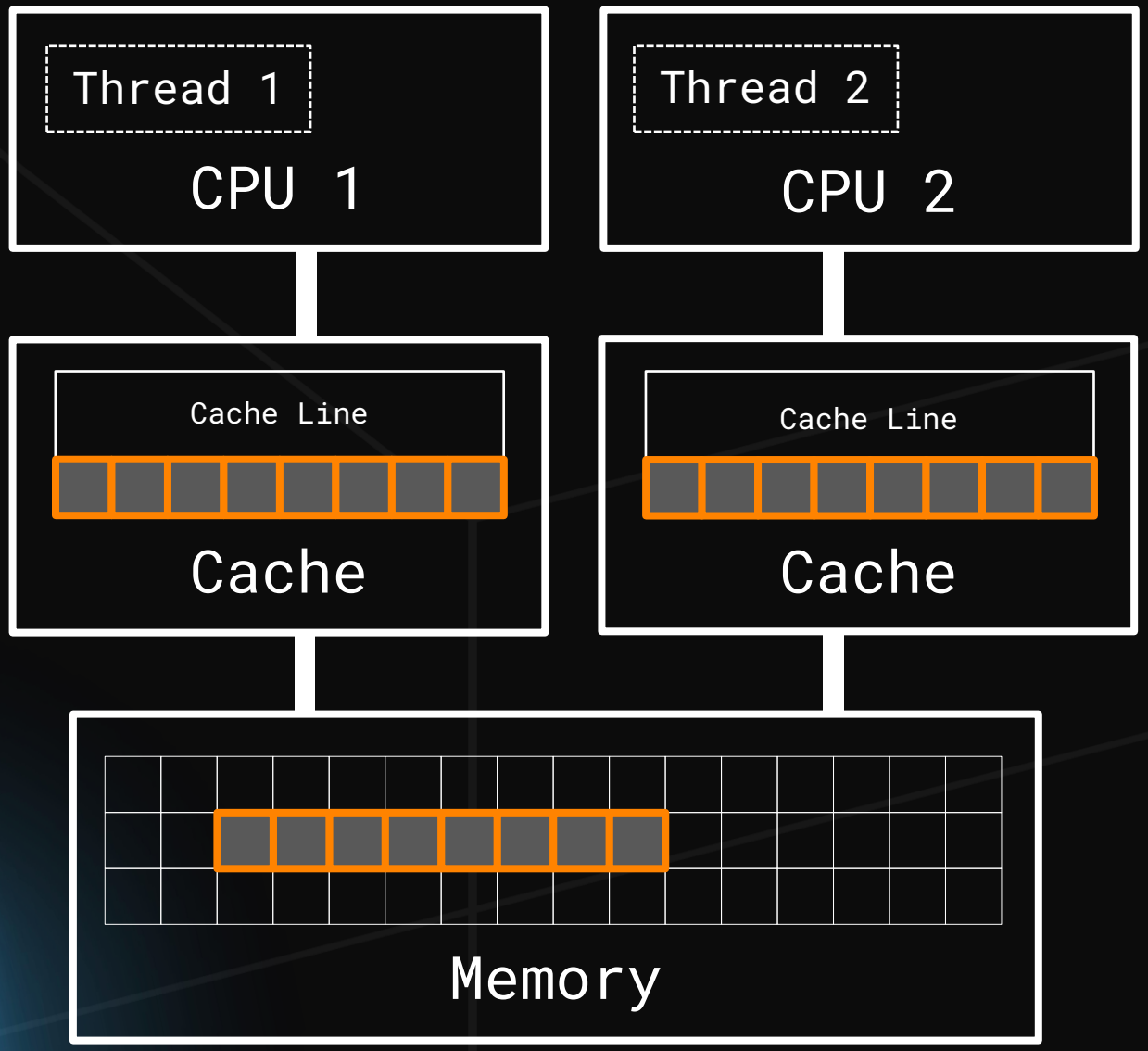
Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002

Queue benchmark

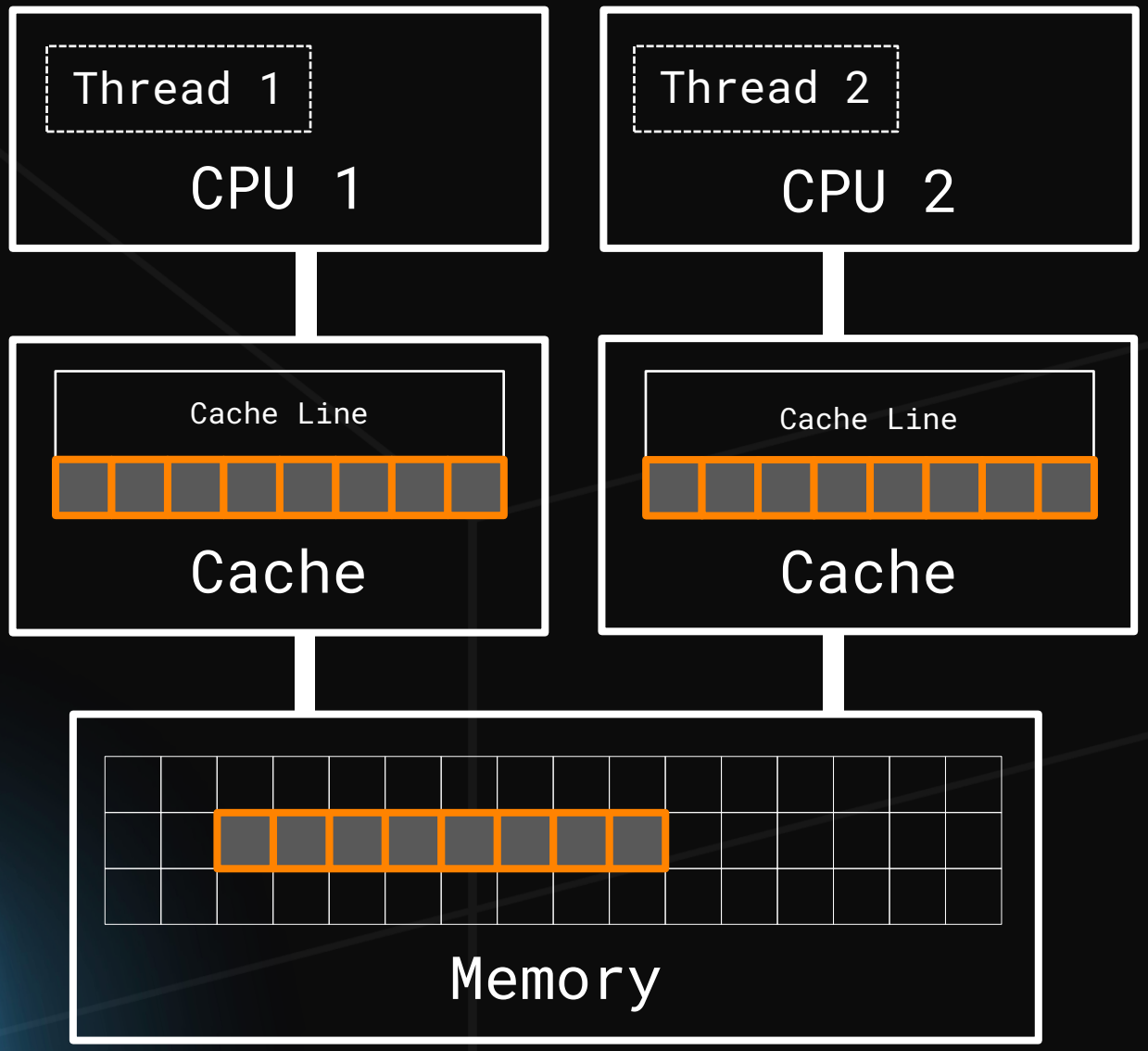
Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002
Acq_RelQueue	37,472


CPU 1
CPU 2
Cache
Cache
Memory

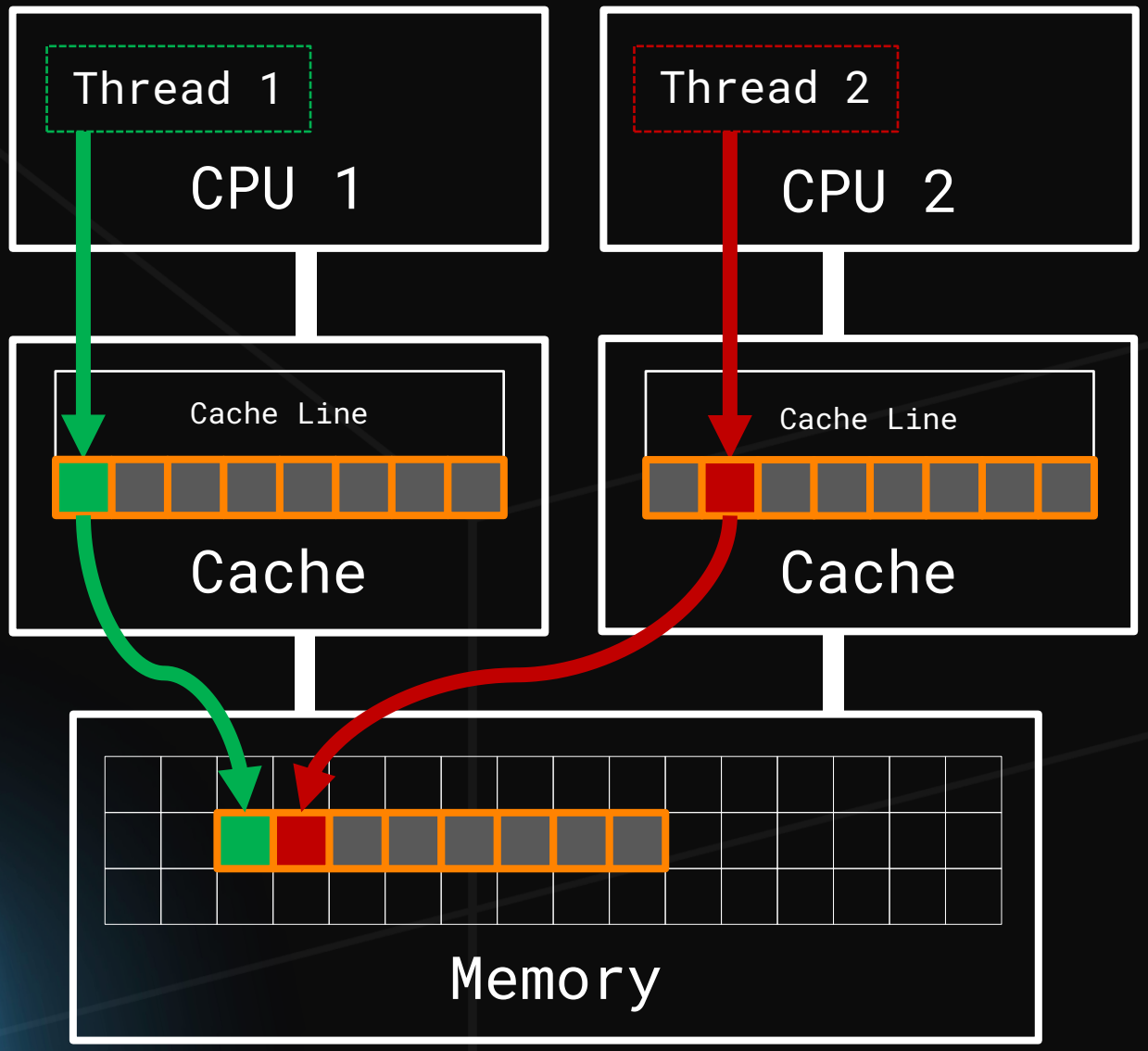

Thread 1
CPU 1
Thread 2
CPU 2
Cache Line
Cache
Cache Line
Cache
Memory



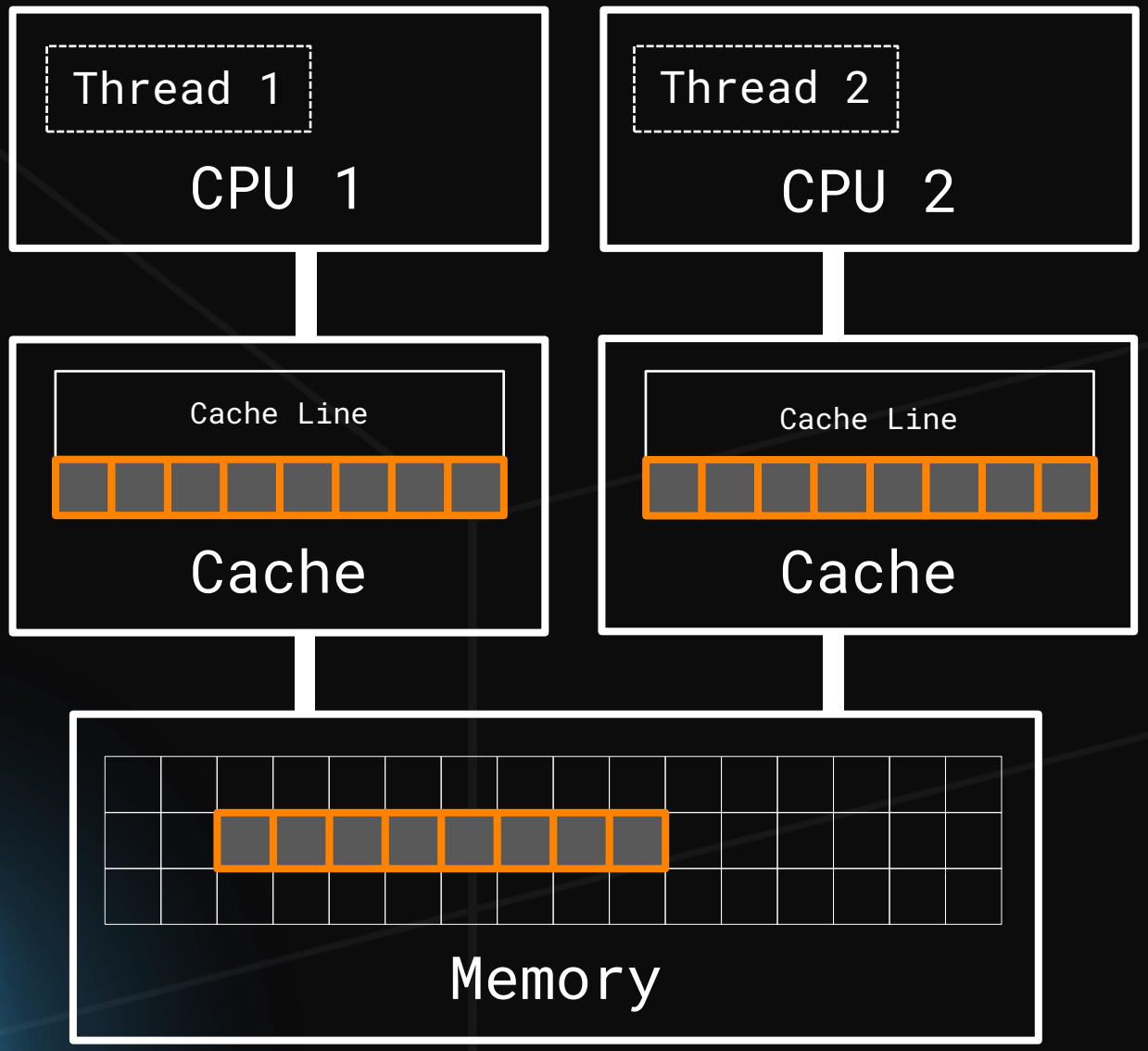
- Кэши взаимодействуют с помощью протокола когерентности



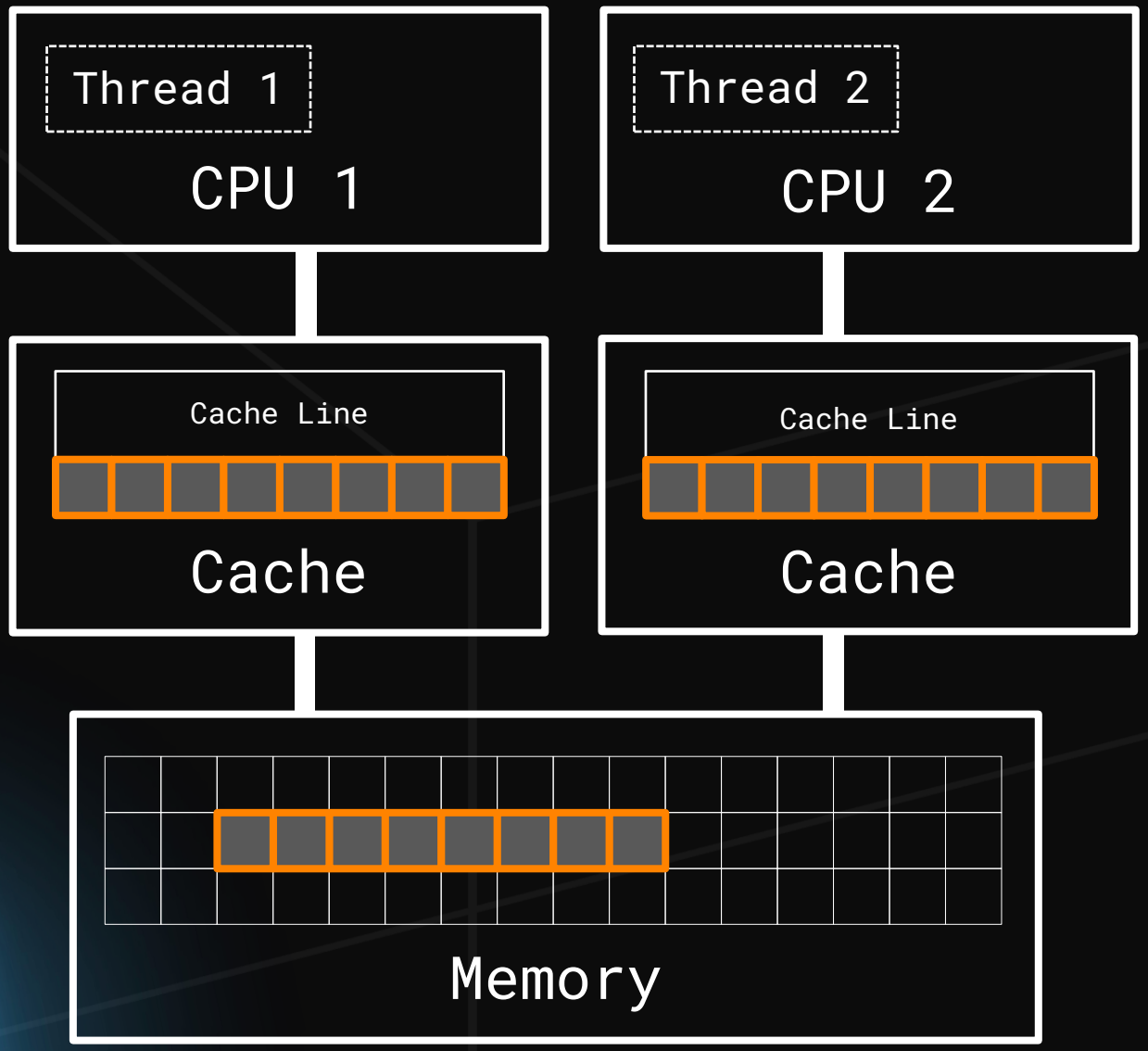
- Кэши взаимодействуют с помощью протокола когерентности
- **False sharing** взаимодействие с разными переменными в рамках одной кэш-линии из нескольких потоков



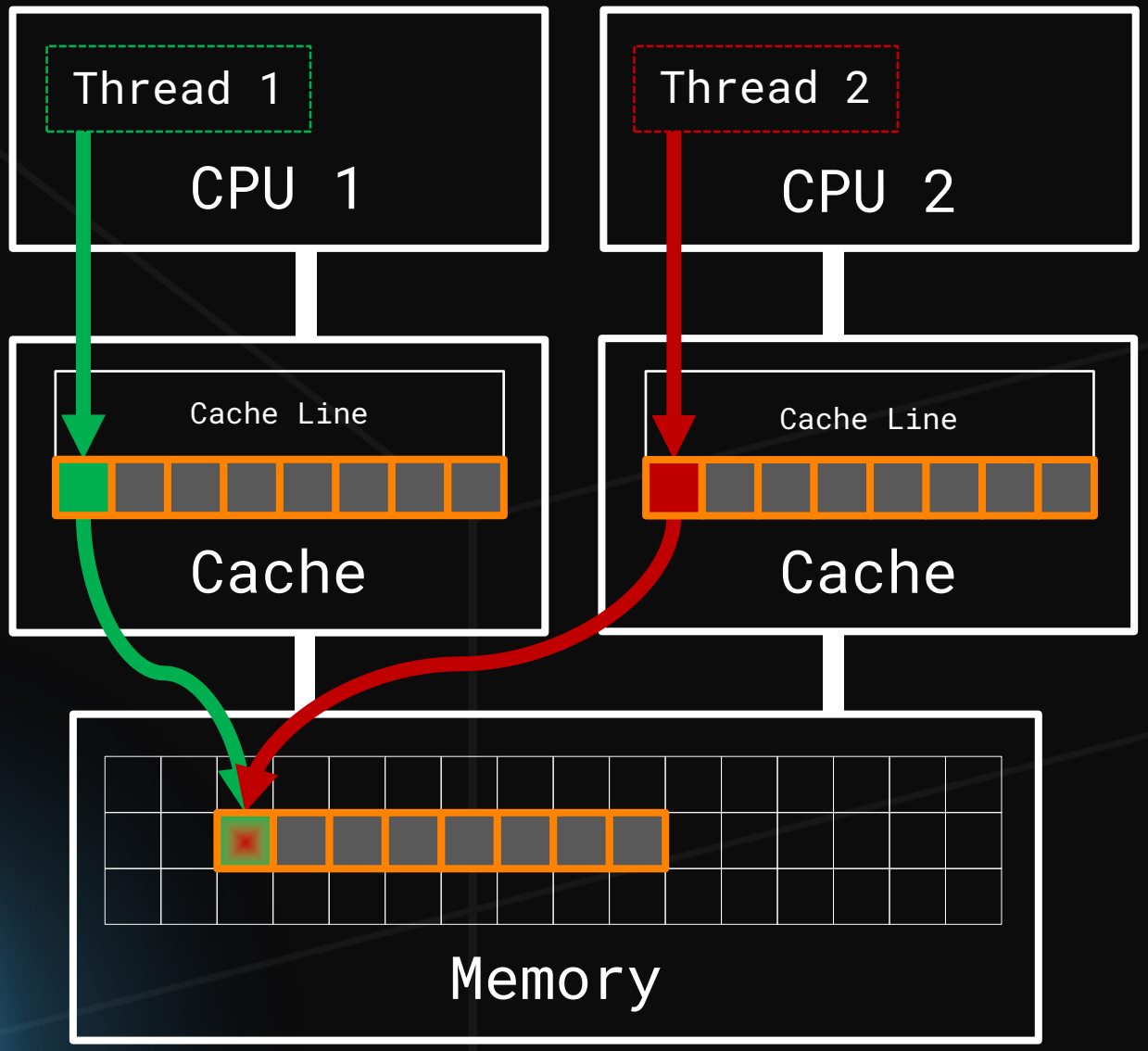
- Кэши взаимодействуют с помощью протокола когерентности
- **False sharing** взаимодействие с разными переменными в рамках одной кэш-линии из нескольких потоков



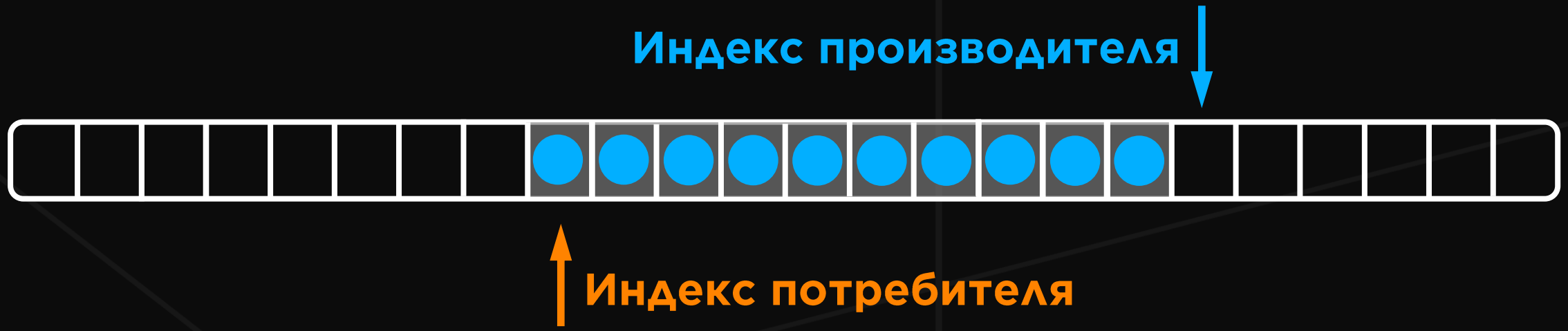
- Кэши взаимодействуют с помощью протокола когерентности
- **False sharing** взаимодействие с разными переменными в рамках одной кэш-линии из нескольких потоков



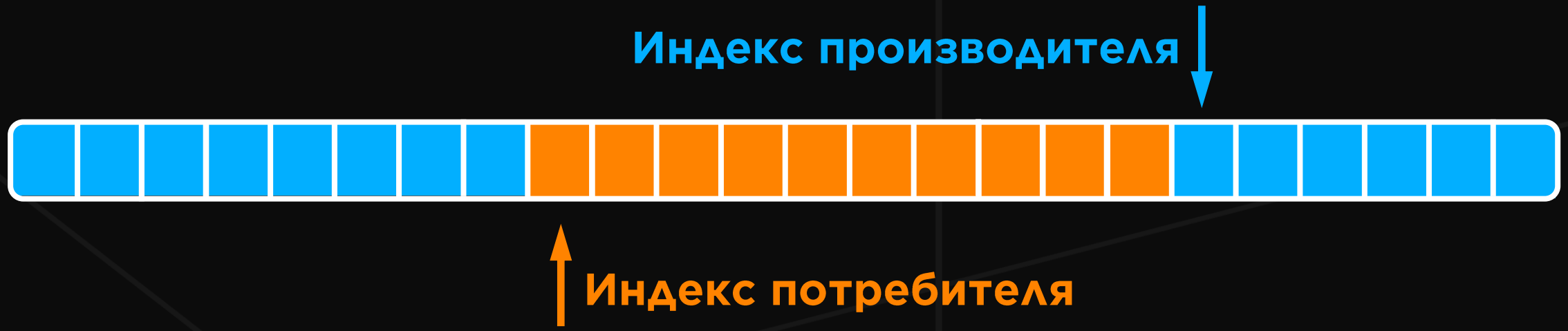
- Кэши взаимодействуют с помощью протокола когерентности
- **False sharing** взаимодействие с разными переменными в рамках одной кэш-линии из нескольких потоков
- **True sharing** взаимодействие с одной и той же переменной из разных потоков



- Кэши взаимодействуют с помощью протокола когерентности
- **False sharing** взаимодействие с разными переменными в рамках одной кэш-линии из нескольких потоков
- **True sharing** взаимодействие с одной и той же переменной из разных потоков



- Потребитель **постоянно читает позицию** производителя, а производитель – постоянно меняет свою позицию
- **Тоже самое** для производителя
- Настоящий **true sharing** – постоянный межъядерный трафик для обеспечения когерентности кэшей
- **Можно ли уменьшить количество трафика?**



- Производитель, считав индекс потребителя один раз – может спокойно писать во все слоты до этого индекса
- Потребитель, считав индекс производителя один раз – может спокойно читать все слоты до этого индекса

```
class SPSC_ImprovedQueue {

}
```

```
class SPSC_ImprovedQueue {
    //индекс потребителя
    int consumerIdx = 0

    //индекс производителя
    int producerIdx = 0

    //буфер для хранения ссылок
    MyObject[] buffer = new MyObject[capacity]

}
```

```
class SPSC_ImprovedQueue {
    //индекс потребителя
    int consumerIdx = 0

    //индекс производителя
    int producerIdx = 0

    //буфер для хранения ссылок
    MyObject[] buffer = new MyObject[capacity]

    //копия индекса потребителя, используется в производителе
    int consCached = 0

}
```

```
class SPSC_ImprovedQueue {
    //индекс потребителя
    int consumerIdx = 0

    //индекс производителя
    int producerIdx = 0

    //буфер для хранения ссылок
    MyObject[] buffer = new MyObject[capacity]

    //копия индекса потребителя, используется в производителе
    int consCached = 0

    //копия индекса производителя, используется в потребителе
    int prodCached = 0
}
```

Producing thread

Producing thread

```
int pIdx = producerIdx
int pNextIdx =
        (pIdx + 1) % capacity

if (pNextIdx == consCached) {
    consCached = CONSUMER.getAcquire()
}

if (isFull(consCached, pIdx)) {
    return false
}

buffer[pIdx] = value

PRODUCER.setRelease(pNextIdx)
return true
```

Producing thread

```
int pIdx = producerIdx
int pNextIdx =
        (pIdx + 1) % capacity

if (pNextIdx == consCached) {
    consCached = CONSUMER.getAcquire()
}

if (isFull(consCached, pIdx)) {
    return false
}

buffer[pIdx] = value

PRODUCER.setRelease(pNextIdx)
return true
```

Producing thread

```
int pIdx = producerIdx
int pNextIdx =
        (pIdx + 1) % capacity

if (pNextIdx == consCached) {
    consCached = CONSUMER.getAcquire()
}

if (isFull(consCached, pIdx)) {
    return false
}

buffer[pIdx] = value

PRODUCER.setRelease(pNextIdx)
return true
```

Producing thread

```
int pIdx = producerIdx
int pNextIdx =
        (pIdx + 1) % capacity

if (pNextIdx == consCached) {
    consCached = CONSUMER.getAcquire()
}

if (isFull(consCached, pIdx)) {
    return false
}

buffer[pIdx] = value

PRODUCER.setRelease(pNextIdx)
return true
```

Consuming thread

```
int cIdx = consumerIdx
int cNextIdx =
        (cIdx + 1) % capacity

if (cIdx == prodCached) {
    prodCached = PRODUCER.getAcquire()
}

if (isEmpty(cIdx, prodCached)) {
    return false
}

consumer.accept(buffer[cIdx])

CONSUMER.setRelease(cNextIdx)
return true
```

Queue benchmark

Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002
AcqRel_Queue	37,472

Queue benchmark

Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002
AcqRel_Queue	37,472
Improved_Volatile_Queue	48,678

Queue benchmark

Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002
AcqRel_Queue	37,472
Improved_Volatile_Queue	48,678
Improved_AcqRel_Queue	165,217

Финальные штрихи

- Размер буфера – степень 2
- Индексы – long с только инкрементацией (иначе ABA проблема для нескольких потребителей/производителей)
- Переиспользование объектов в производителе

Queue benchmark

Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002
AcqRel_Queue	37,472
Improved_Volatile_Queue	48,678
Improved_AcqRel_Queue	165,217

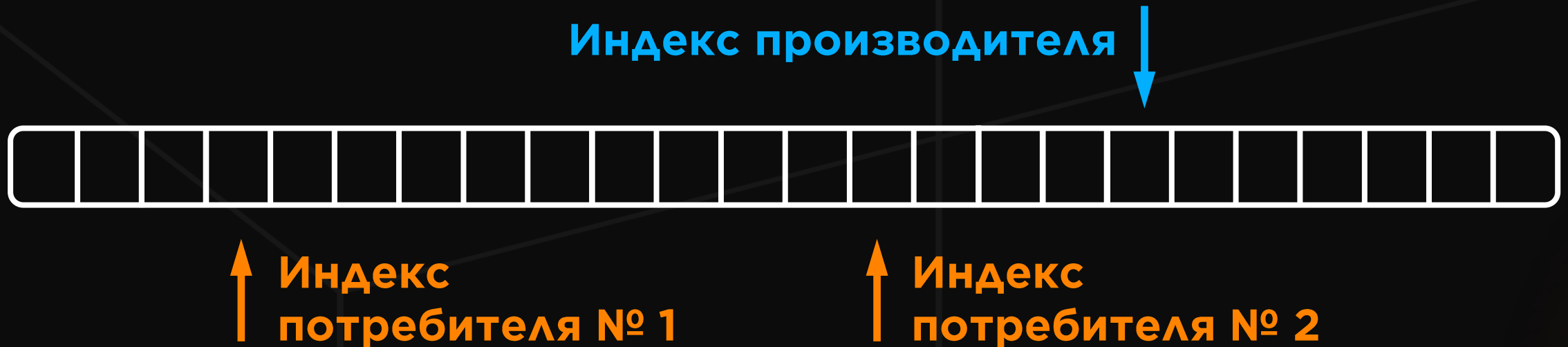
Queue benchmark

Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002
AcqRel_Queue	37,472
Improved_Volatile_Queue	48,678
Improved_AcqRel_Queue	165,217
Disruptor (Single Producer)	183,308

Queue benchmark

Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002
AcqRel_Queue	37,472
Improved_Volatile_Queue	48,678
Improved_AcqRel_Queue	165,217
Disruptor (Single Producer)	183,308
Disrupted_AcqRel_Queue	190,000

Disruptor: нюансы использования



- Самый медленный потребитель тормозит **всех** производителей и потребителей
- Иногда такое нужно, иногда – нет
- Недокументированный интерфейс **EventReleaser** (удален в Disruptor 4.0)

Seqlock

```
class DateTimeSnapshot {
    int year, month, day;
    int hour, minute, second;
    int millis, micros, nanos;



}
```

- Один писатель, неограниченное количество читателей
- Медленный читатель не должен приводить к тормозам писателя/других читателей
- Минимальное количество блокировок

```
class DateTimeSnapshot {
    int year, month, day;
    int hour, minute, second;
    int millis, micros, nanos;

    void update(DateTimeSnapshot from)
    boolean tryRead(DateTimeSnapshot into)
}
```

- Один писатель, неограниченное количество читателей
- Медленный читатель не должен приводить к тормозам писателя/других читателей
- Минимальное количество блокировок

```
class DateTimeSnapshot {
    int year, month, day;
    int hour, minute, second;
    int millis, micros, nanos;

    long version = 0;

    void update(DateTimeSnapshot from)
    boolean tryRead(DateTimeSnapshot into)
}
```

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year  = from.year
    this.month = from.month
              ...
    this.nanos = from.nanos

    version++
}
```

WARNING: это
псевдокод с гонками

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year  = from.year
    this.month = from.month
              ...
    this.nanos = from.nanos

    version++
}
```

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = version
    if (vBefore % 2 == 1)
        return false



}
```

WARNING: это псевдокод с гонками

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year  = from.year
    this.month = from.month
              ...
    this.nanos = from.nanos

    version++
}
```

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = version
    if (vBefore % 2 == 1)
        return false

    into.year  = this.year
    into.month = this.month
              ...
    into.nanos = this.nanos


}
```

WARNING: это псевдокод с гонками

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year  = from.year
    this.month = from.month
               ...
    this.nanos = from.nanos

    version++
}
```

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = version
    if (vBefore % 2 == 1)
        return false

    into.year  = this.year
    into.month = this.month
               ...
    into.nanos = this.nanos

    return version == vBefore
}
```

WARNING: это псевдокод с гонками

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.y                    ear
    this                      nth

    this.n                    .nanos

    version++
}
```

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = version
    if (vBefore % 2 == 1)
        return false

    into.year  = this.year
    into.month = this.month
                ...
    into.nanos = this.nanos

    return version == vBefore
}
```

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year
    this.month

    this.nanos

    version++
}
```

writes

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = version
    if (vBefore % 2 == 1)
        return false

    into.year
    into.month

    into.nanos

    return version == vBefore
}
```

reads

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year
    this.month

    this.nanos

    version++
}
```

writes

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = version
    if (vBefore % 2 == 1)
        return false

    into.year
    into.month

    into.nanos

    return version == vBefore
}
```

reads

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year
    this.month

    this.nanos

    VERSION.getAndAddRelease(1)
}
```

writes

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = version
    if (vBefore % 2 == 1)
        return false

    into.year
    into.month

    into.nanos

    return version == vBefore
}
```

reads

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year
    this.month

    this.nanos

    VERSION.getAndAddRelease(1)
}
```

writes

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = VERSION.getAcquire()
    if (vBefore % 2 == 1)
        return false

    into.year
    into.month

    into.nanos

    return version == vBefore
}
```

reads

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year
    this.month

    this.nanos

    VERSION.getAndAddRelease(1)
}
```

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = VERSION.getAcquire()
    if (vBefore % 2 == 1)
        return false

    into.year
    into.month

    into.nanos

    return version == vBefore
}
```

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year
    this.month

    this.nanos

    VERSION.getAndAddRelease(1)
}
```

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = VERSION.getAcquire()
    if (vBefore % 2 == 1)
        return false

    into.year
    into.month

    into.nanos

    return version == vBefore
}
```

Как обеспечить упорядоченность между этими операциями?

Writer

```
void update(DateTimeSnapshot from) {
    version++

    this.year  = from.year
    this.month = from.month
              ...
    this.nanos = from.nanos

    VERSION.getAndAddRelease(1)
}
```

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = VERSION.getAcquire()
    if (vBefore % 2 == 1)
        return false

    into.year  = this.year
    into.month = this.month
              ...
    into.nanos = this.nanos

    return version == vBefore
}
```

Writer

```
void update(DateTimeSnapshot from) {
    version++
    VarHandle.fullFence()
    this.year  = from.year
    this.month = from.month
              ...
    this.nanos = from.nanos

    VERSION.getAndAddRelease(1)
}
```

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = VERSION.getAcquire()
    if (vBefore % 2 == 1)
        return false

    into.year  = this.year
    into.month = this.month
              ...
    into.nanos = this.nanos

    return version == vBefore
}
```

Writer

```
void update(DateTimeSnapshot from) {
    version++
    VarHandle.fullFence()
    this.year  = from.year
    this.month = from.month
              ...
    this.nanos = from.nanos

    VERSION.getAndAddRelease(1)
}
```

Reader

```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = VERSION.getAcquire()
    if (vBefore % 2 == 1)
        return false

    into.year  = this.year
    into.month = this.month
              ...
    into.nanos = this.nanos
    VarHandle.fullFence()
    return version == vBefore
}
```

Writer

```
void update(DateTimeSnapshot from) {
    version++
    VarHandle.fullFence()
    this.year  = from.year
    this.month = from.month
              ...
    this.nanos = from.nanos

    VERSION.getAndAddRelease(1)
}
```

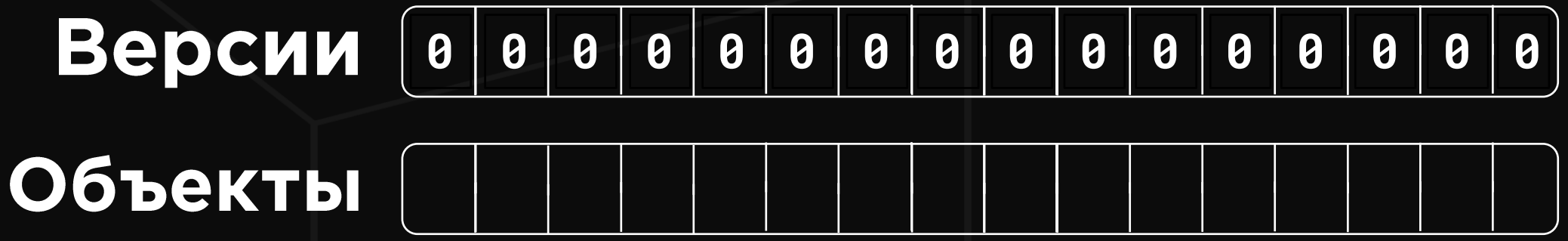
Reader

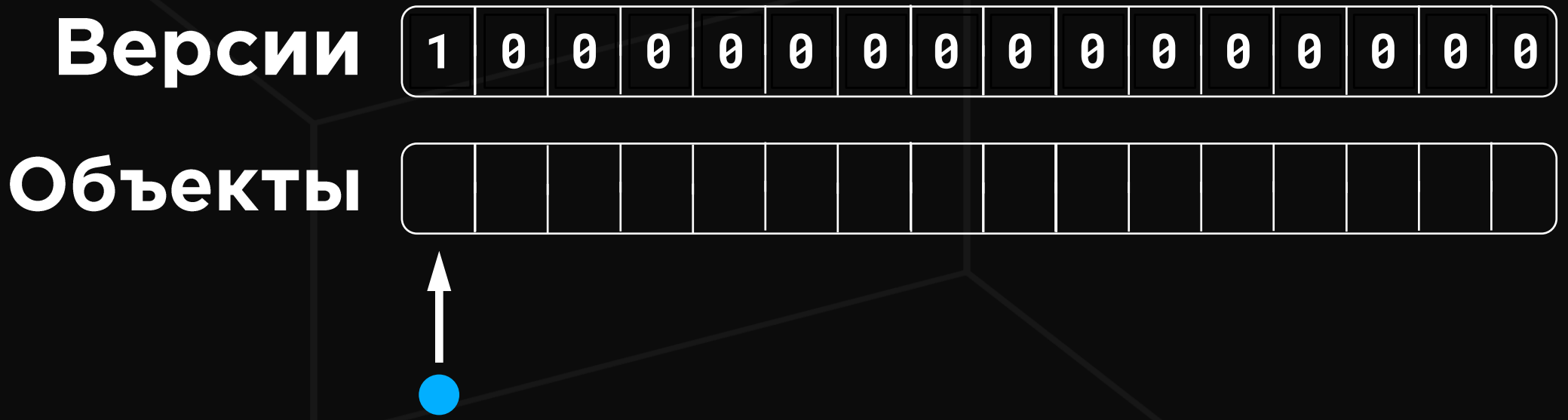
```
boolean tryRead(DateTimeSnapshot into) {
    long vBefore = VERSION.getAcquire()
    if (vBefore % 2 == 1)
        return false

    into.year  = this.year
    into.month = this.month
              ...
    into.nanos = this.nanos
    VarHandle.fullFence()
    return version == vBefore
}
```

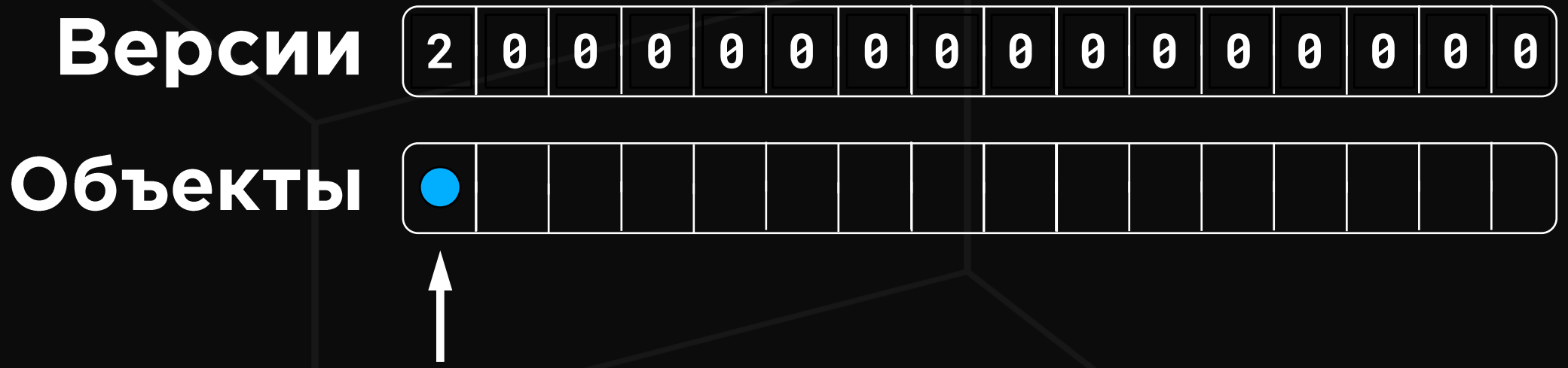
Обратите внимание: в методе читателя нет ни одной записи, только чтения

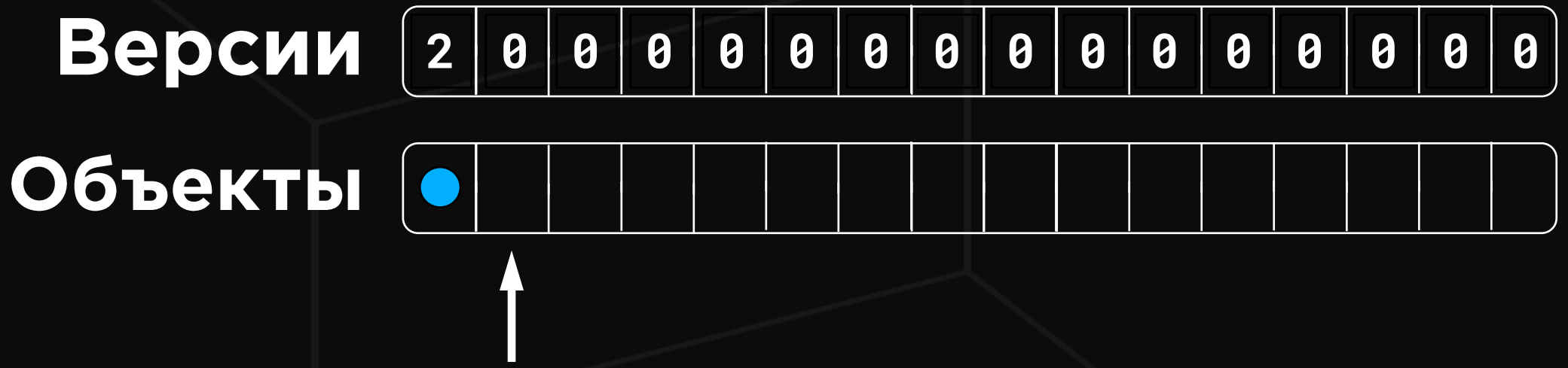
Seqlock-очередь

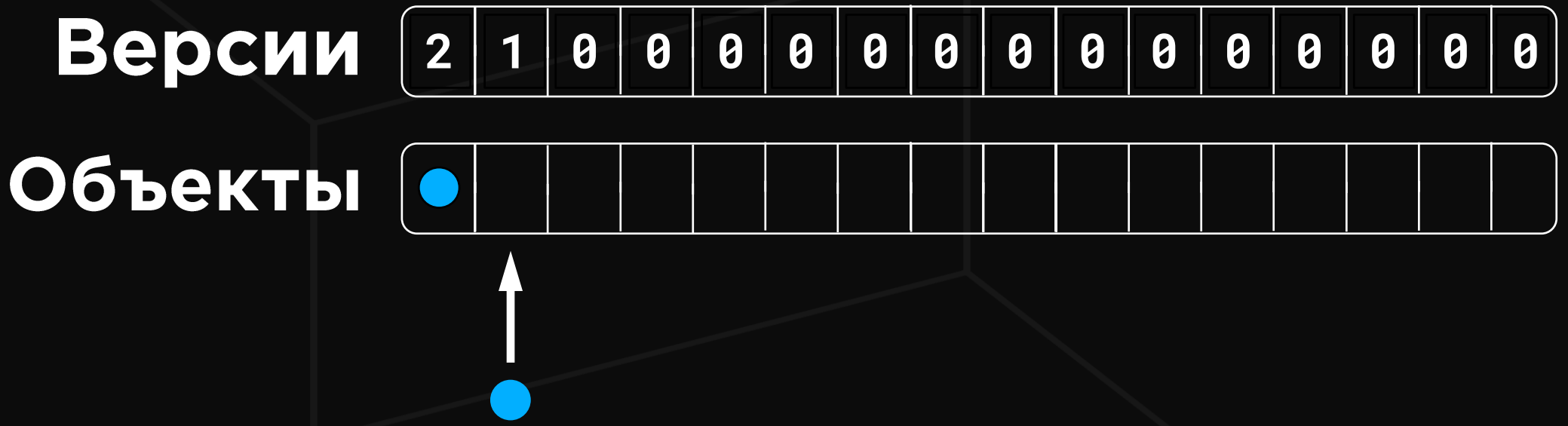
Объекты

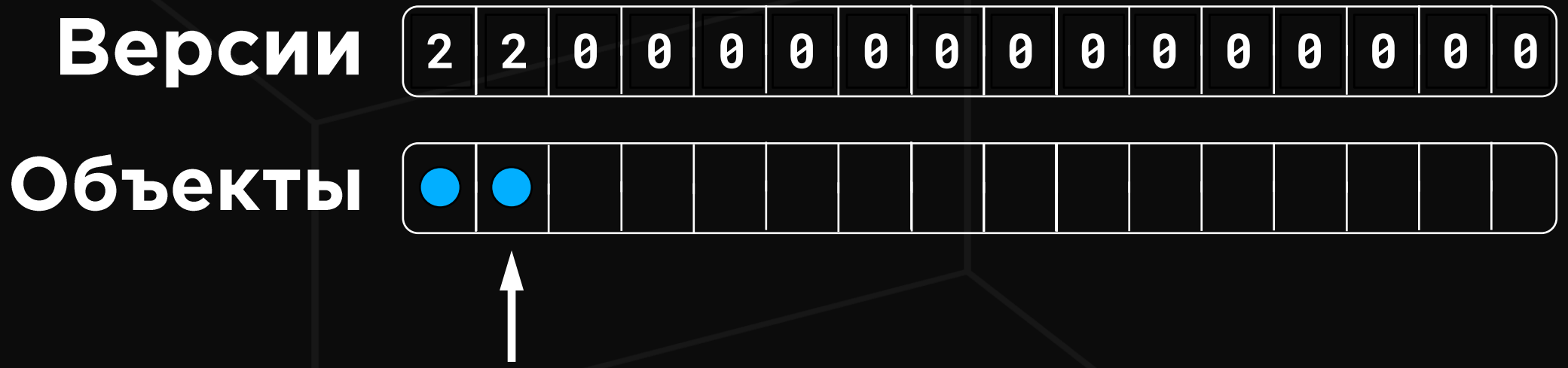

Версии
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Объекты

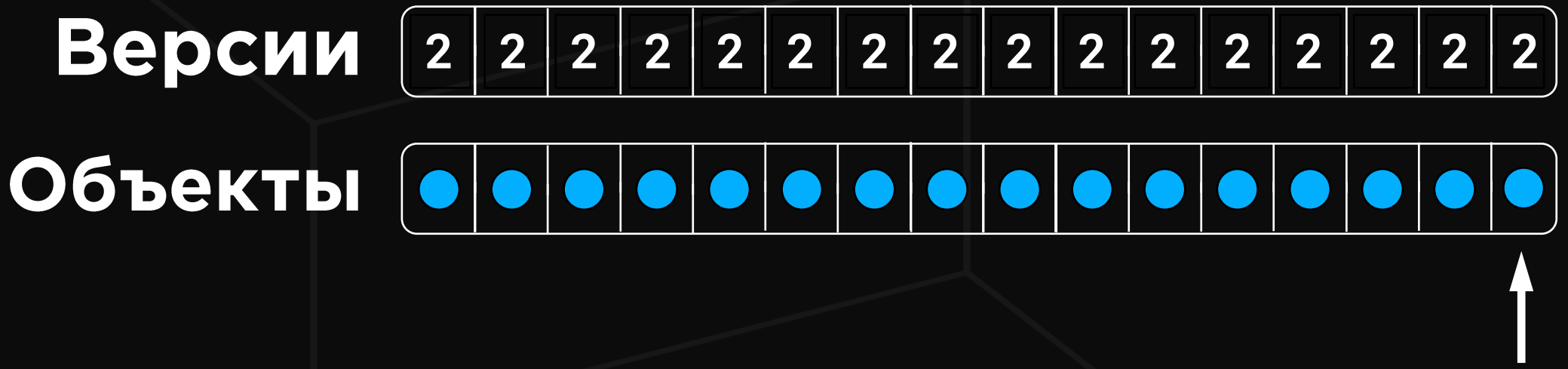

Версии
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Объекты

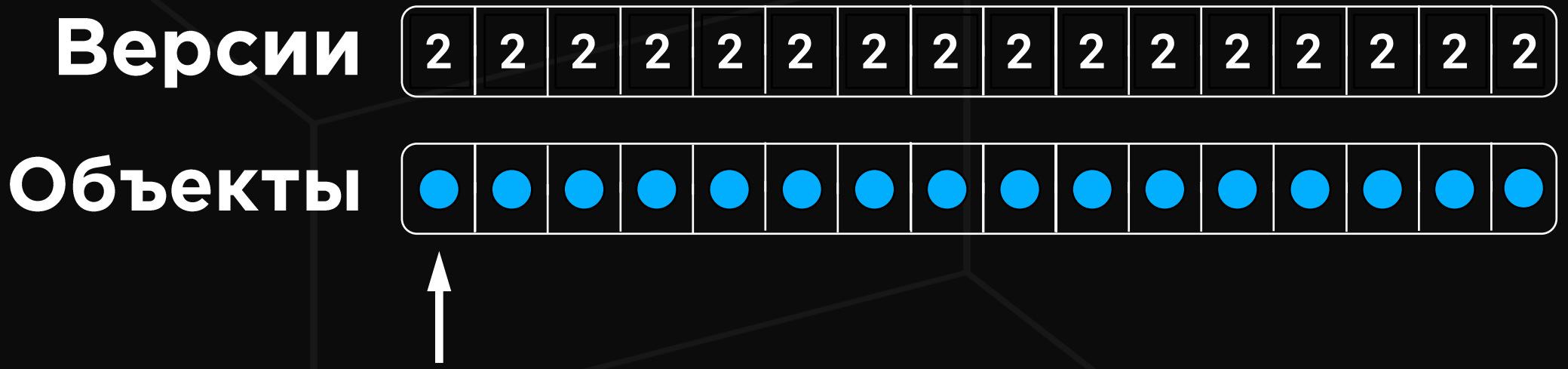

Версии
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Объекты

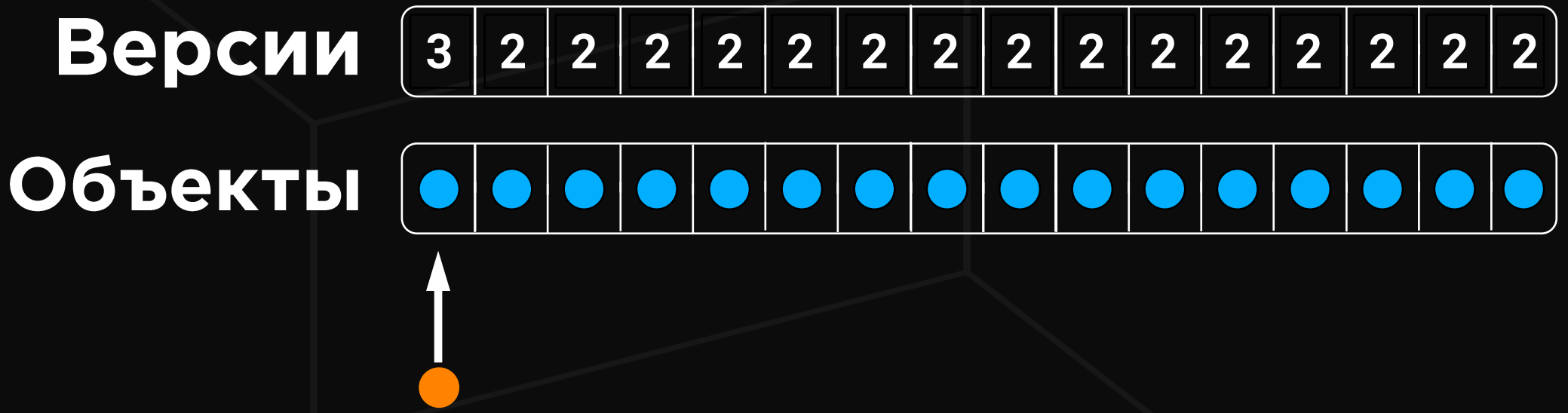

Версии
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Объекты

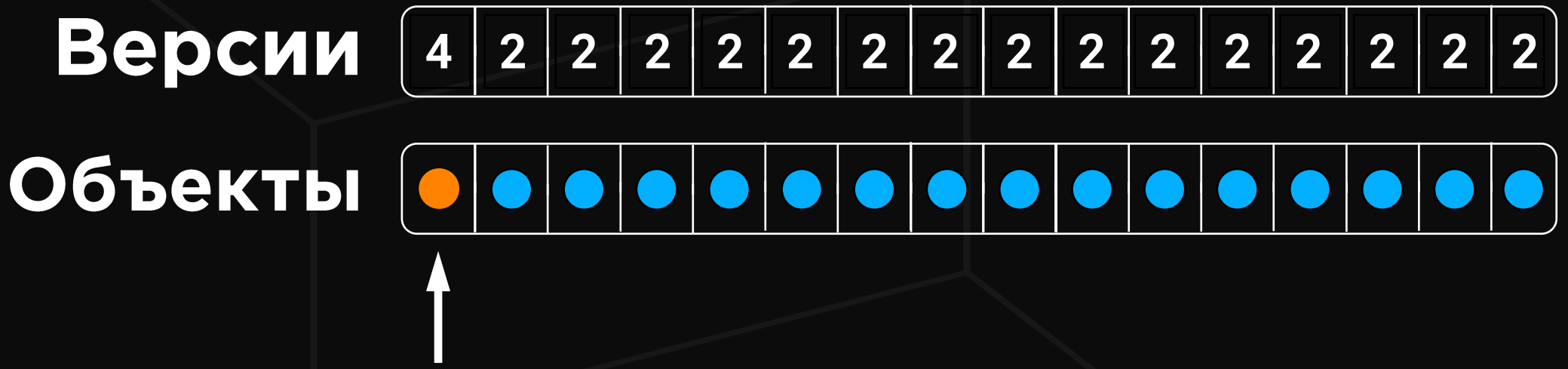

Версии
2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Объекты

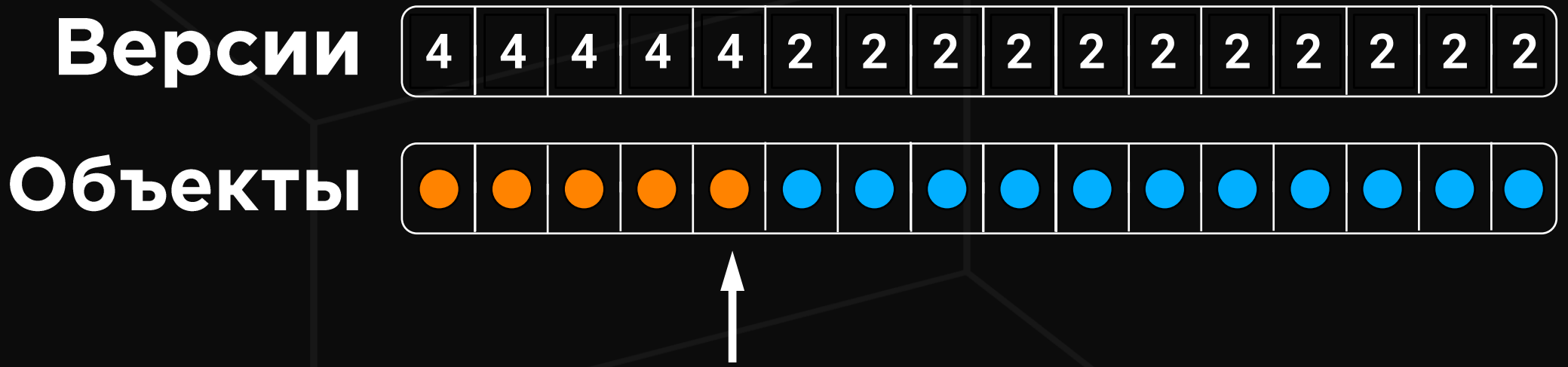

Версии
2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Объекты


Версии
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Объекты

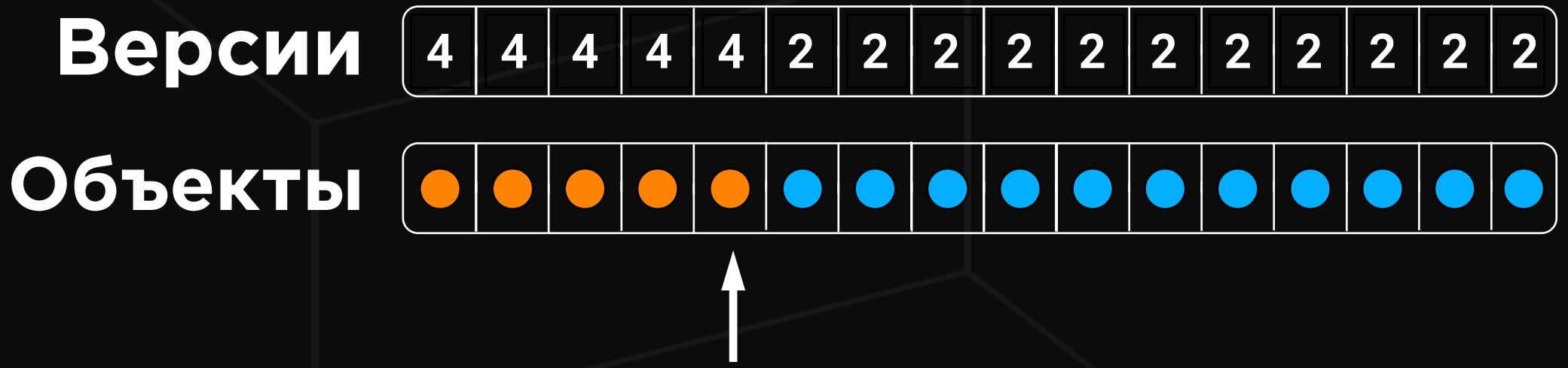

Версии
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Объекты


Версии
3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Объекты


Версии
4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Объекты


Версии
4 4 4 4 4 2 2 2 2 2 2 2 2 2 2 2
Объекты

А потребитель то где?



Queue benchmark

Тип очереди	Throughput, ops/ms
ArrayBlocking_QueueWrapper	6,178
Volatile_Queue	17,002
AcqRel_Queue	37,472
Improved_Volatile_Queue	48,678
Improved_AcqRel_Queue	165,217
Disruptor (Single Producer)	183,308
Disrupted_AcqRel_Queue	190,000

Queue benchmark

Тип очереди	Throughput, ops/ms
ArrayBlockingQueue_Wrapper	6,178
Volatile_Queue	17,002
AcqRel_Queue	37,472
Improved_Volatile_Queue	48,678
Improved_AcqRel_Queue	165,217
Disruptor (Single Producer)	183,308
Disrupted_AcqRel_Queue	190,000
Seqlock_Queue	198,000

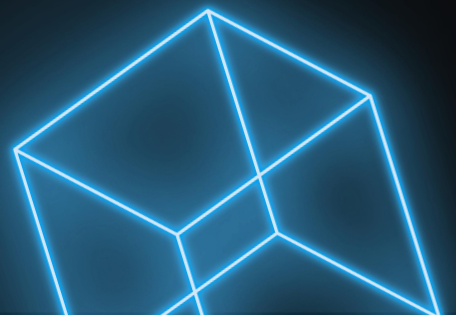
Lincheck: A Practical Framework for Testing Concurrent Data Structures on JVM

by JetBrains Research: *https://link.springer.com/chapter/10.1007/978-3-031-37706-8_8*

- **Баги находятся до сих пор!**
- `ConcurrentLinkedDeque`, `AbstractQueuedSynchronizer`
- `NonBlockingHashMapLong`
- **Некоторые конкурентные структуры позволяют нарушение линеаризуемости из-за ослабленных семантик**

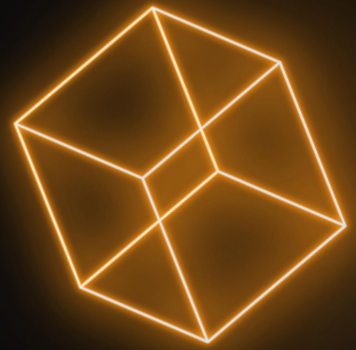
Итоги

- Посмотрели на практическое применение семантик
- Последовательно дошли до архитектуры Disruptor
- Посмотрели на внутреннее устройство Seqlock
- Велосипеды зло! За каждой реальной библиотекой – годы разработки и исправлений багов...
- Но, зная семантики, можно понимать, как они устроены внутри



Спасибо

за внимание!



Все примеры и бенчмарки

github.com/lantalex/semantics-sandbox