# **Масштабируемость** в распределенных **in-memory** системах

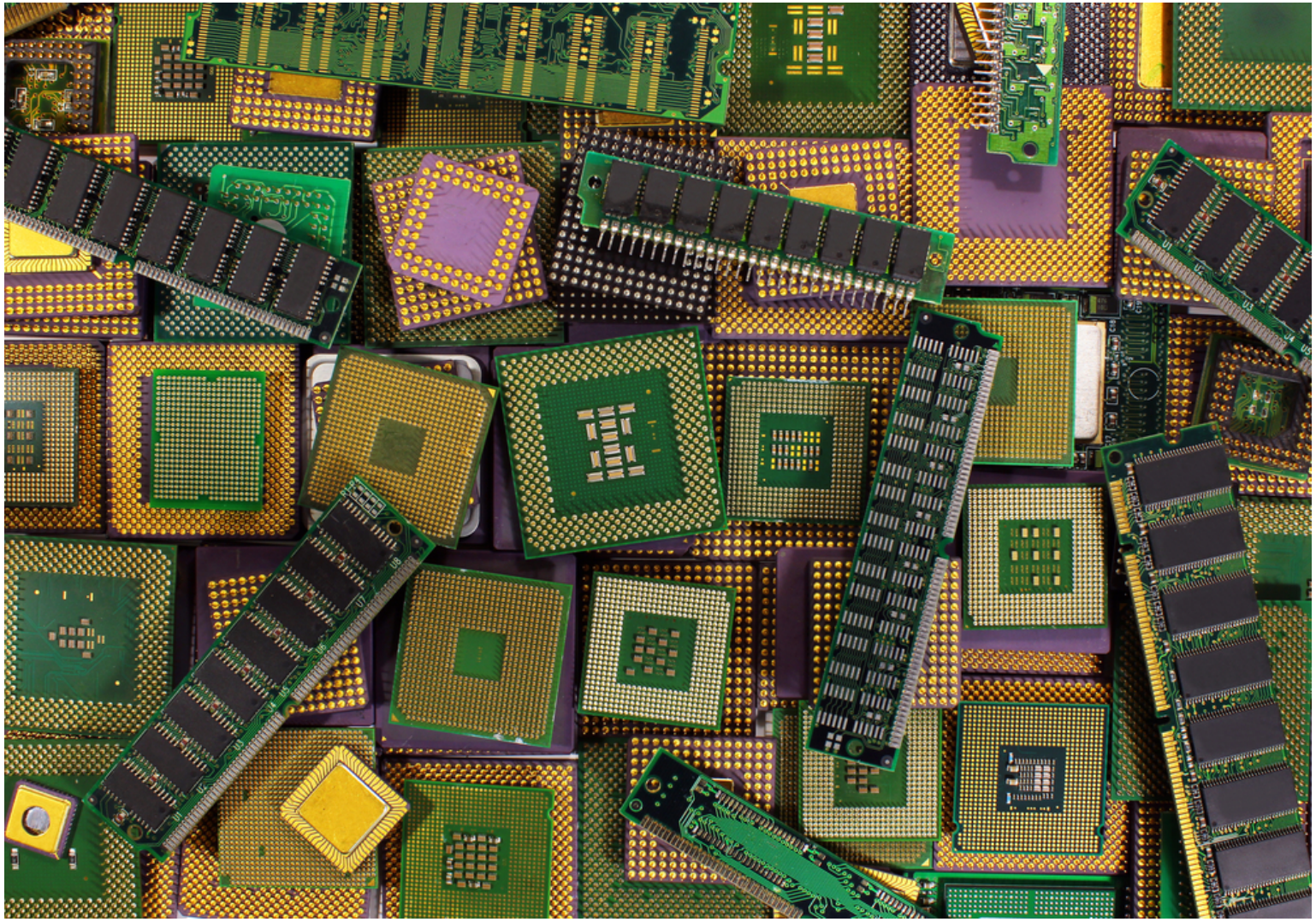
Владимир Озеров

Яков Жданов

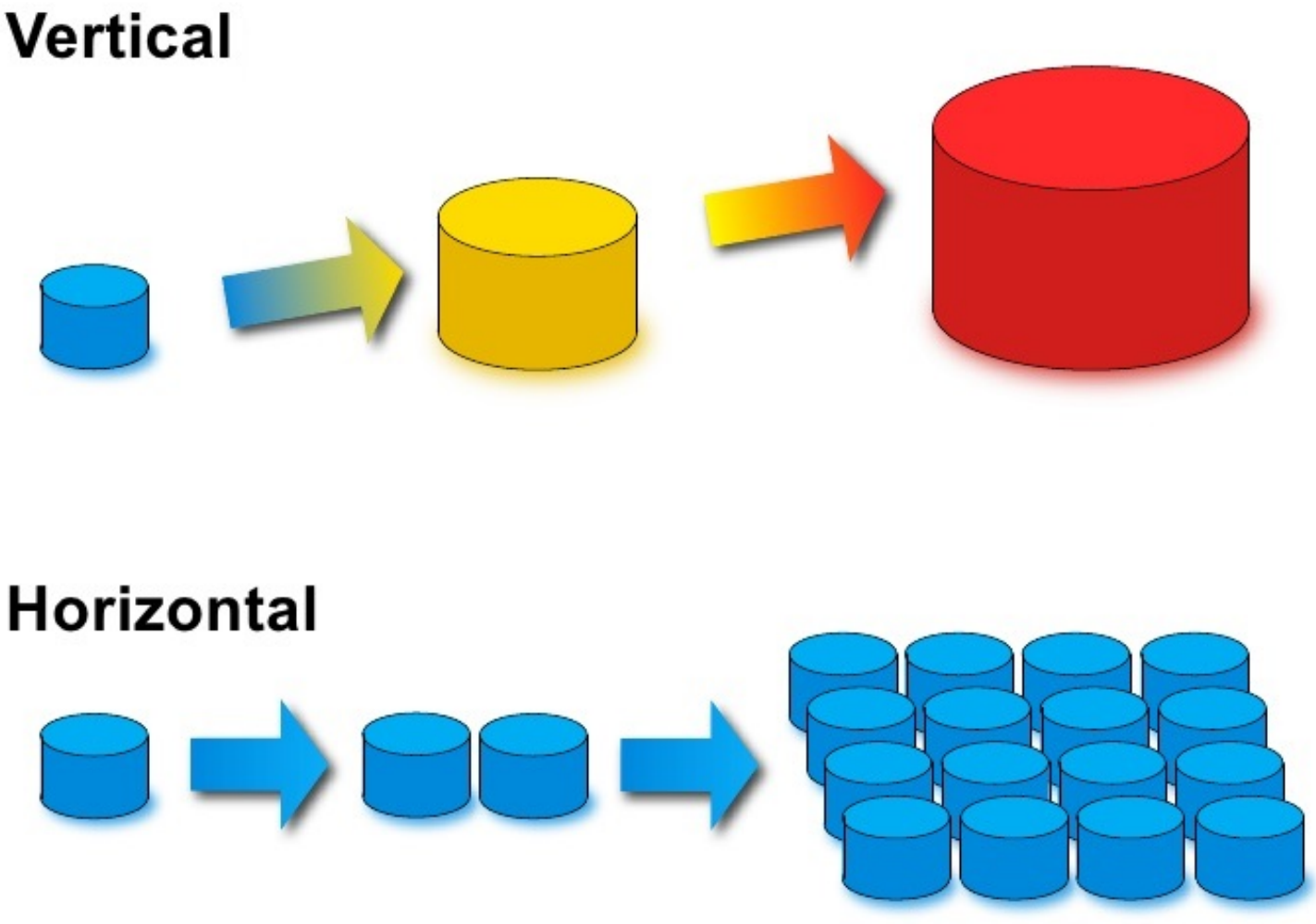
GridGain

# Кто?


# Почему in-memory?



# Масштабируем



# План

- Алгоритмы шардирования

# План

- Алгоритмы шардирования
- Data co-location

# План

- Алгоритмы шардирования
- Data co-location
- Синхронизация в кластере

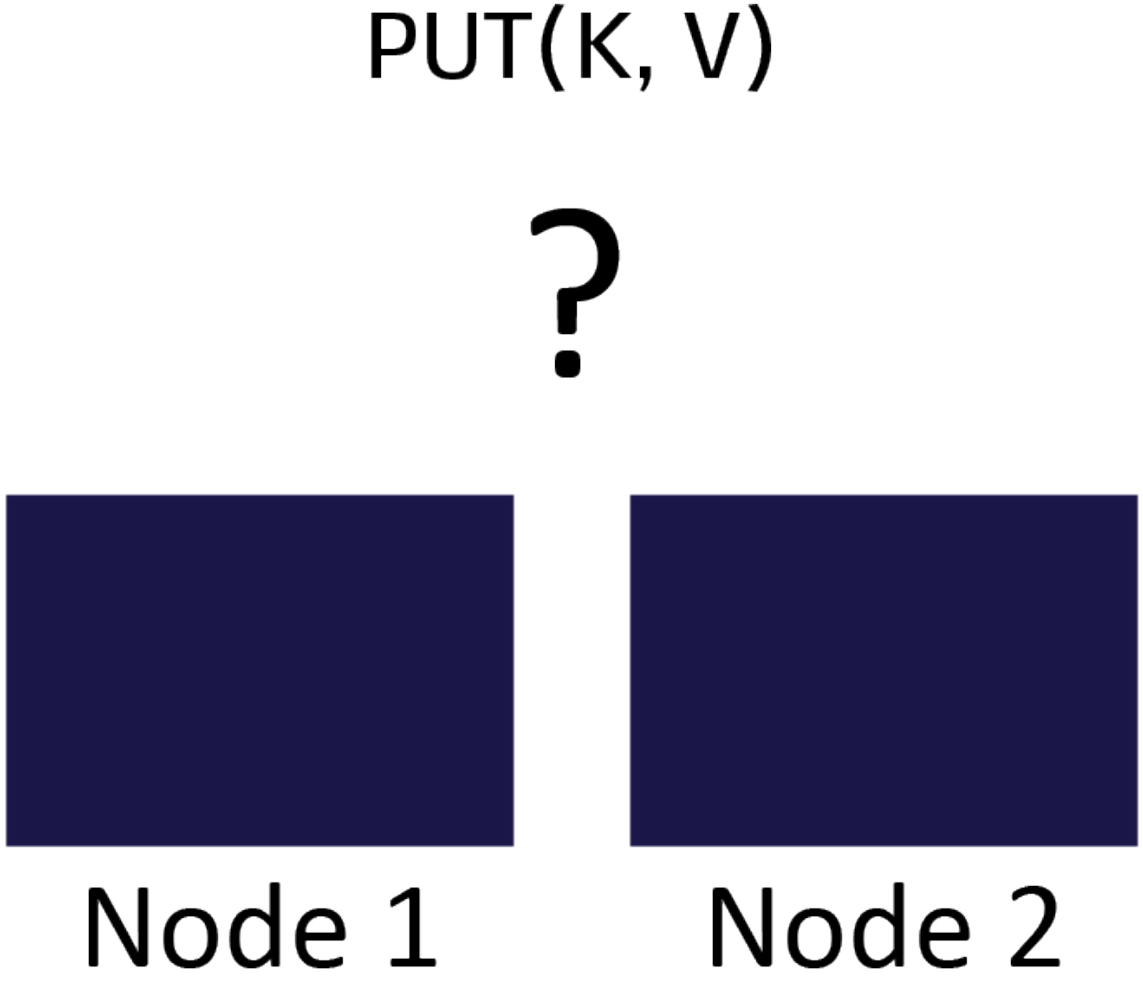
# План

- Алгоритмы шардирования
- Data co-location
- Синхронизация в кластере
- Локальная архитектура многопоточности

# План

- Алгоритмы шардирования
- Data co-location
- Синхронизация в кластере
- Локальная архитектура многопоточности

# Куда?



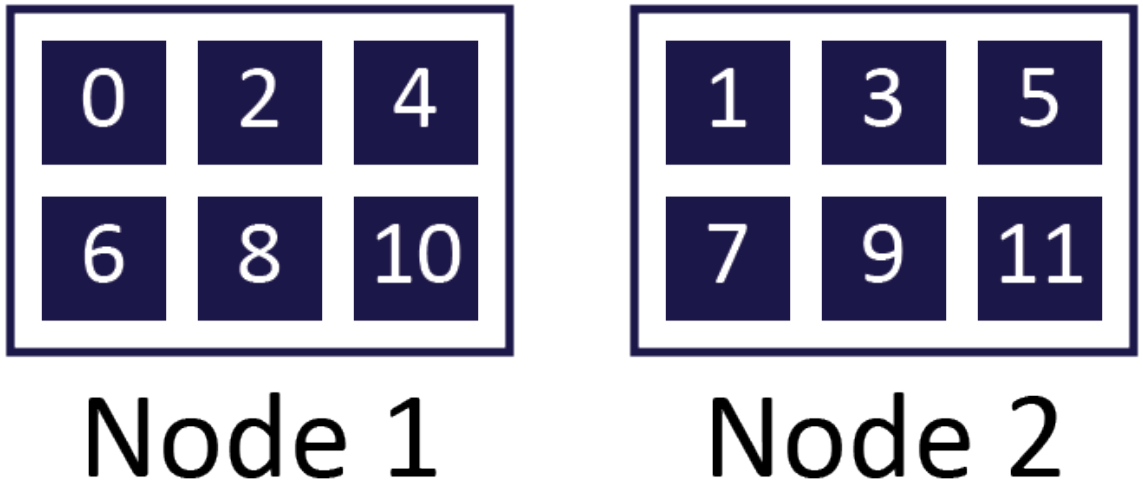
# Affinity

partition => node

# Affinity

PUT(K, V)

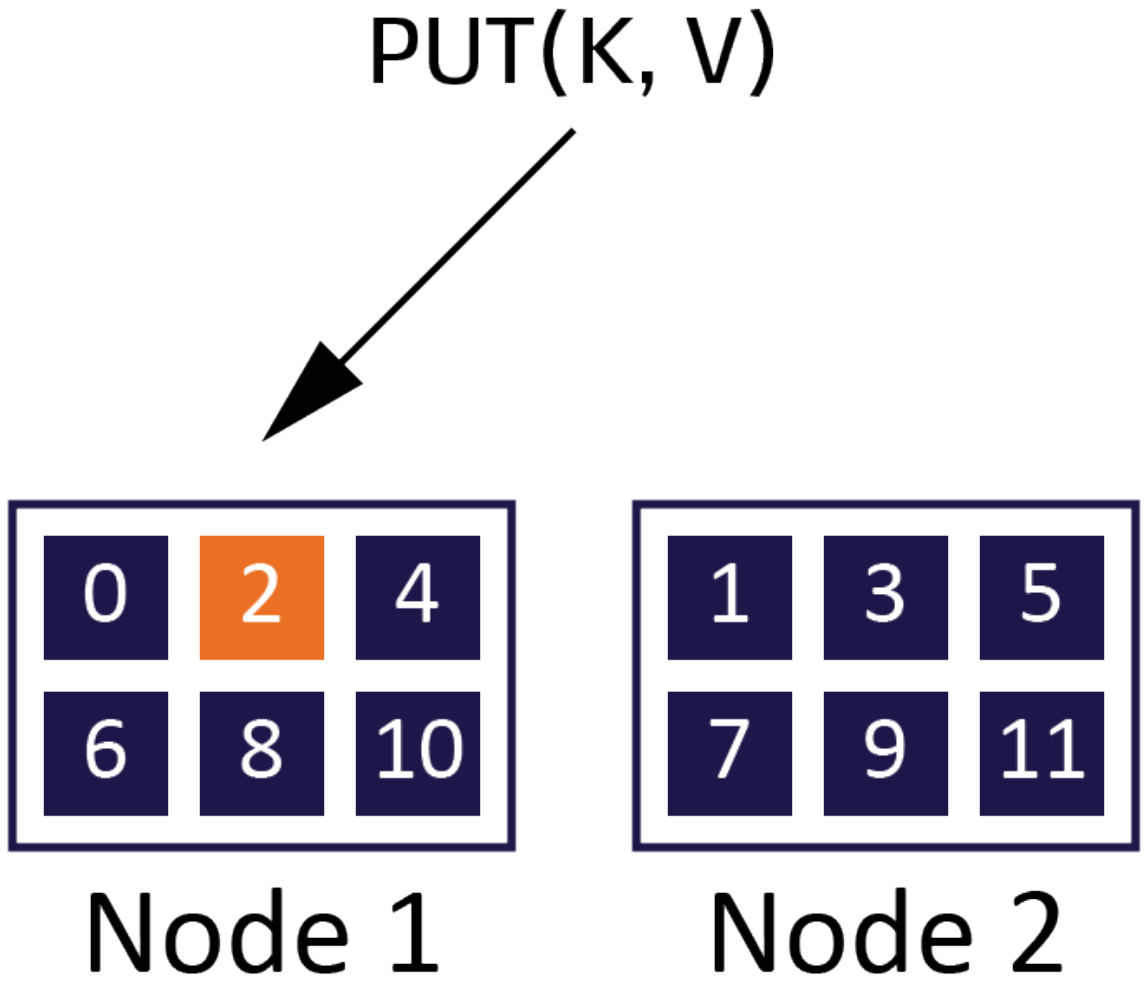
?



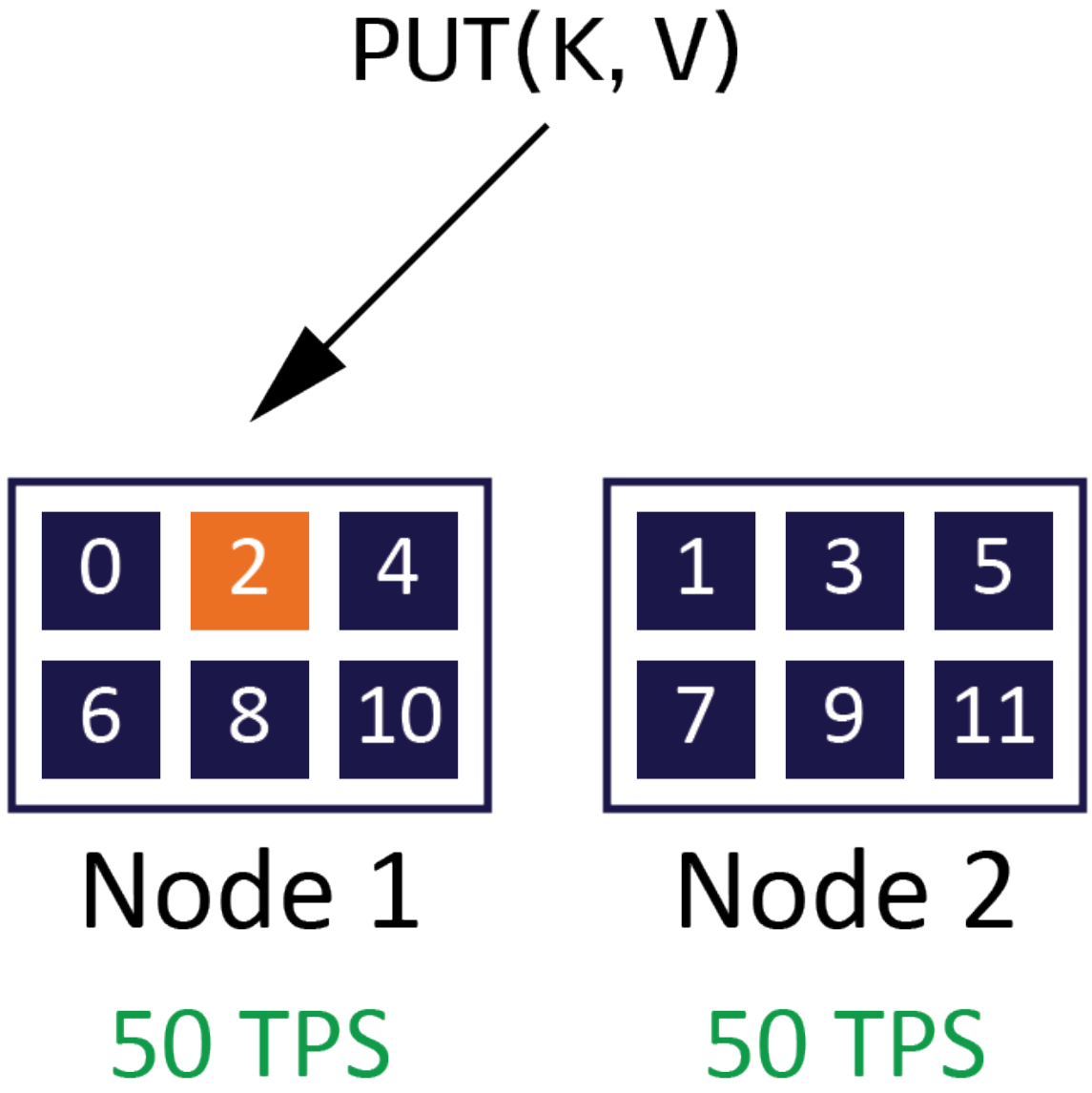
# Affinity

key => partition => node

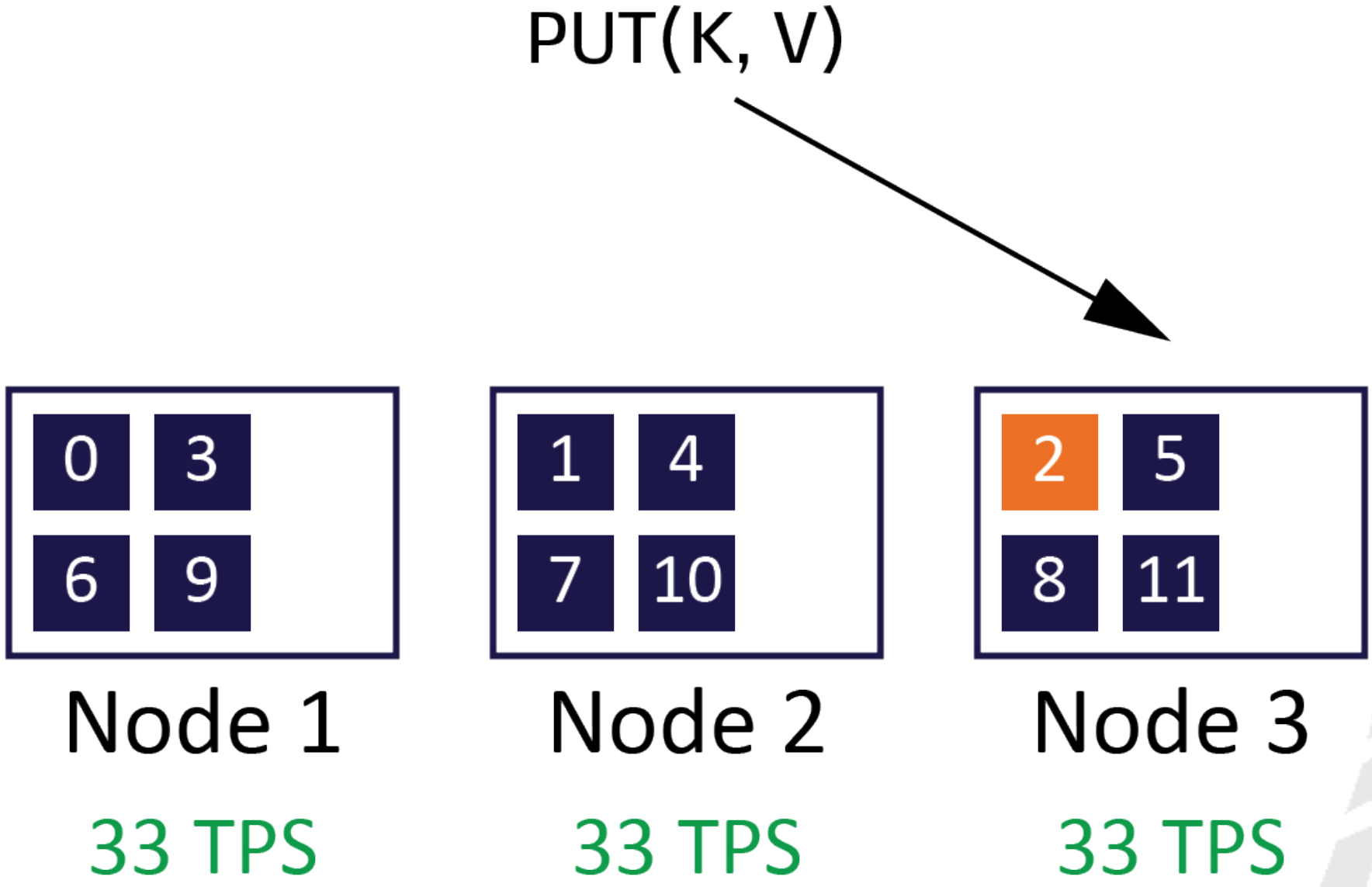
# Affinity



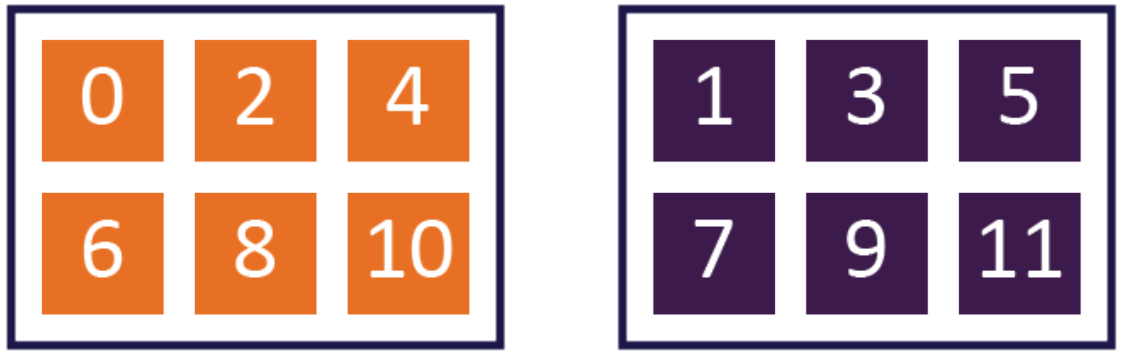
# Naïve affinity



# Naïve affinity

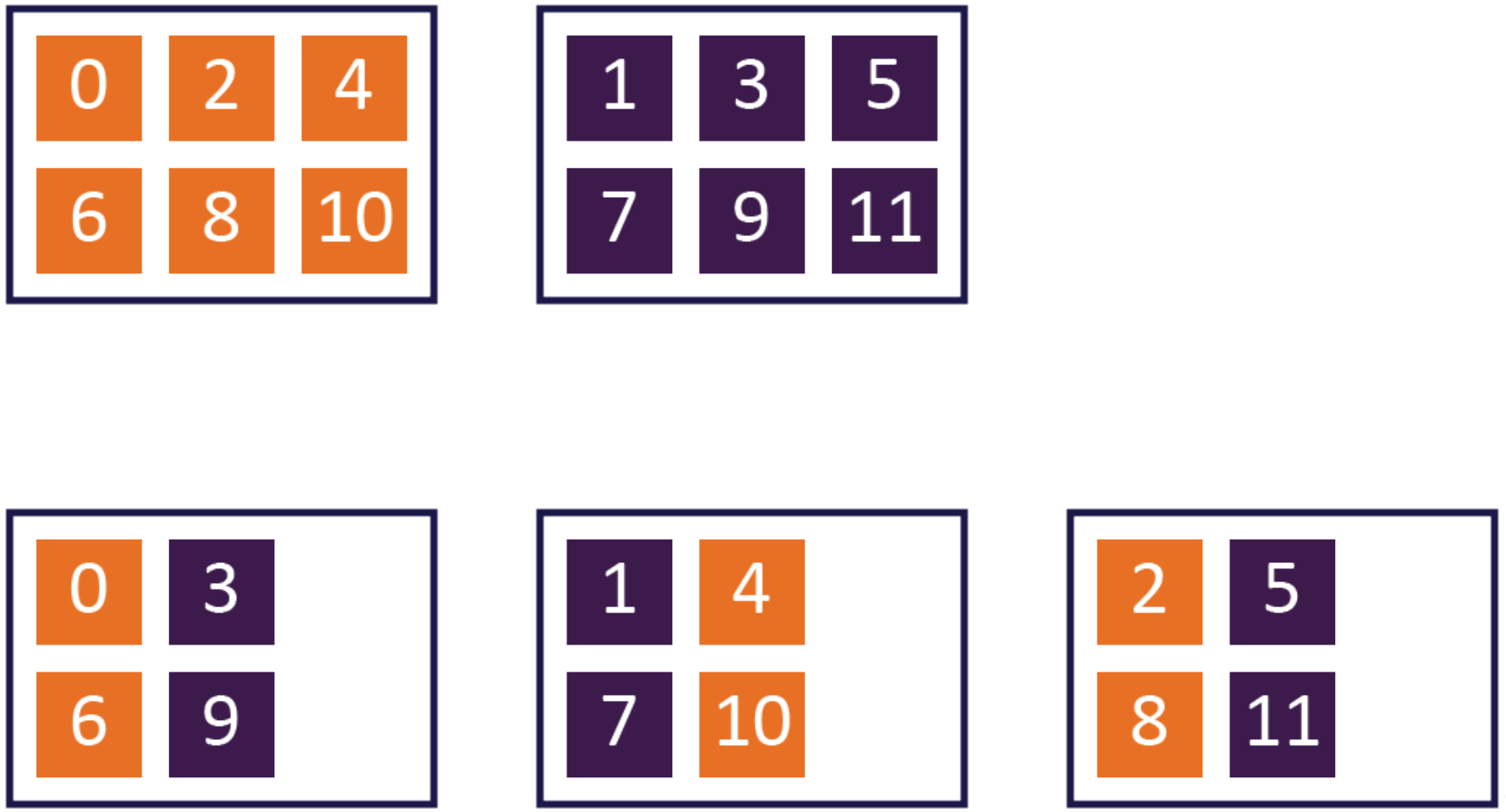


# Naïve affinity

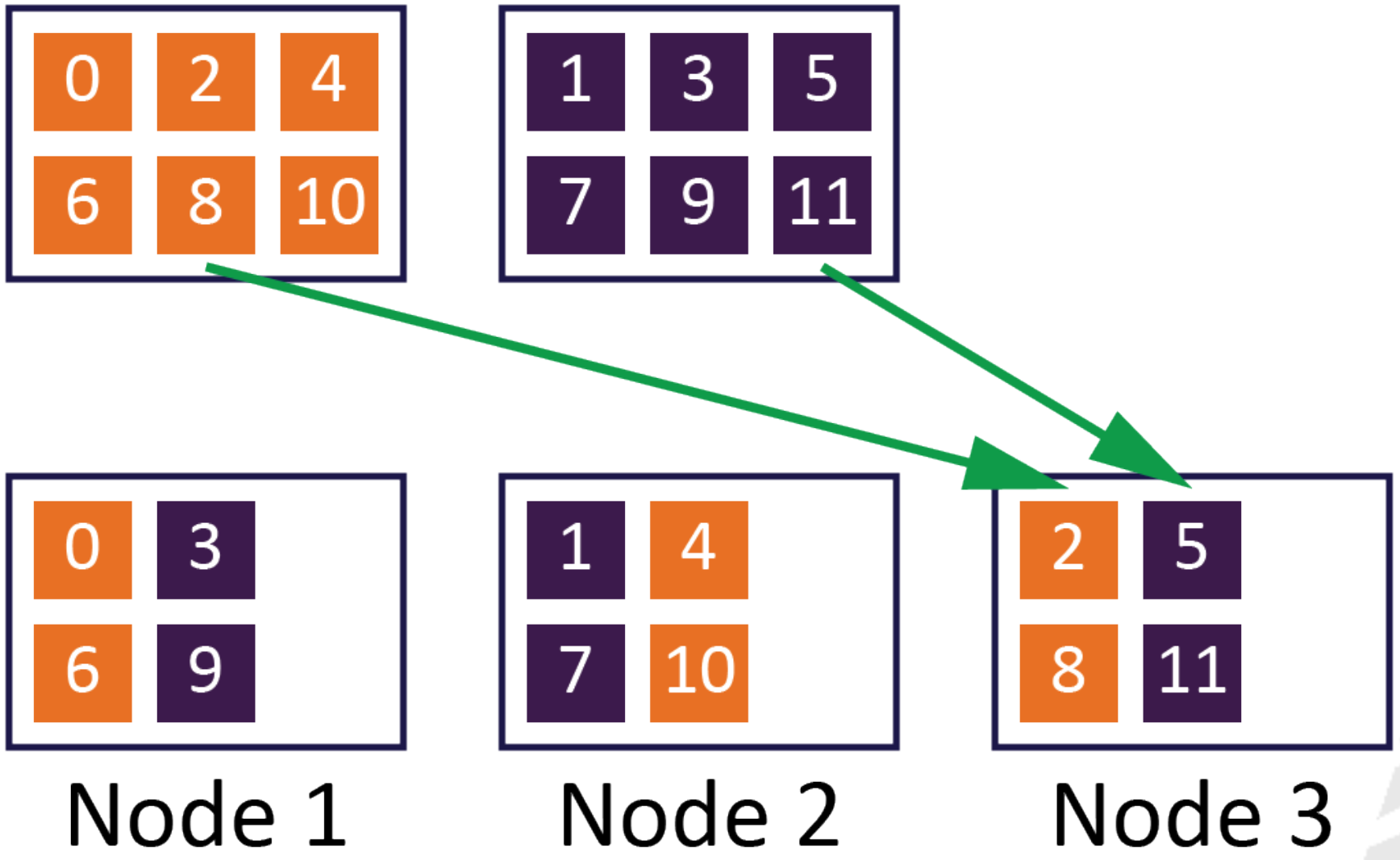



Node 1 Node 2

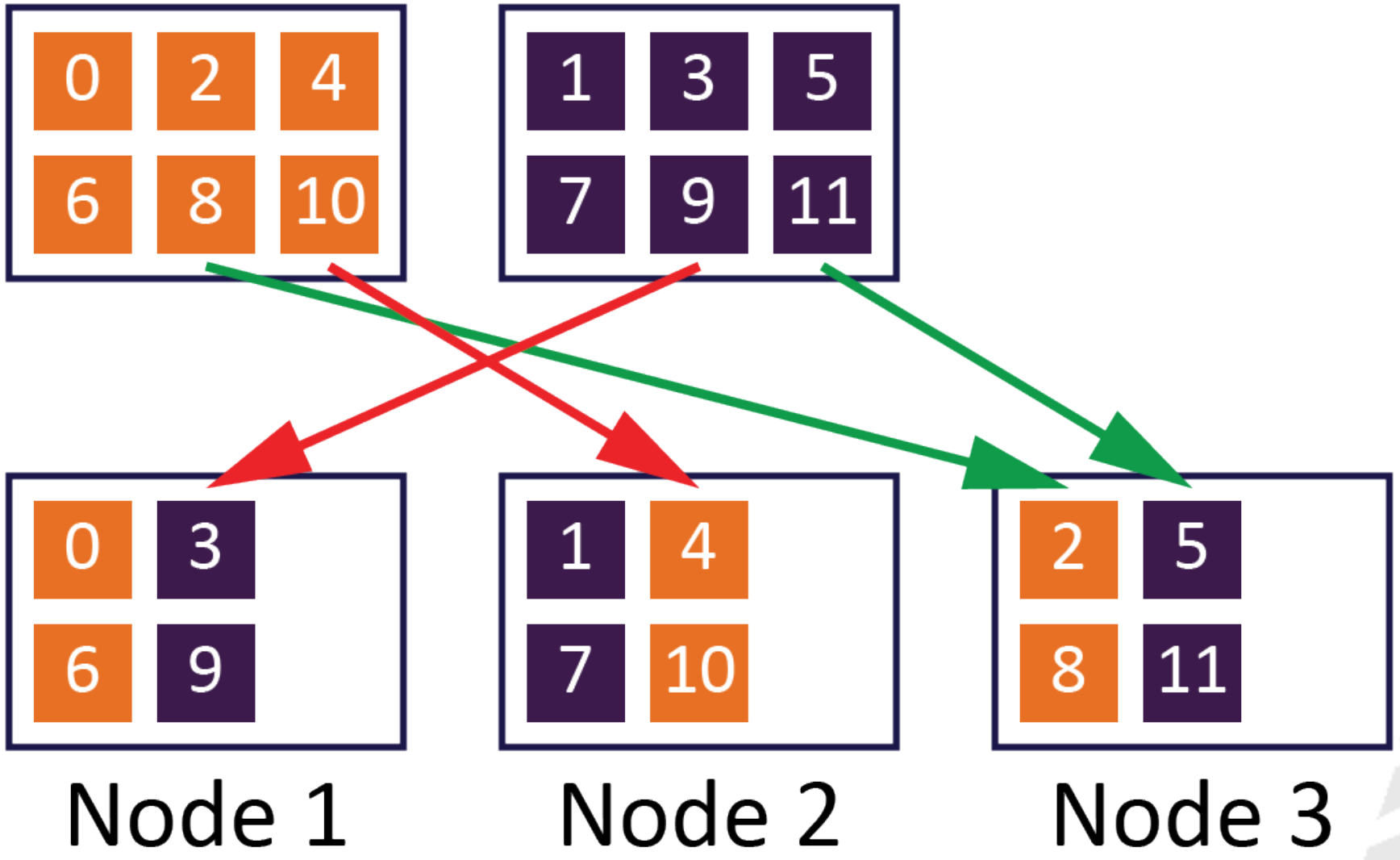
# Naïve affinity



# Naïve affinity



# Naïve affinity



# Rendezvous affinity

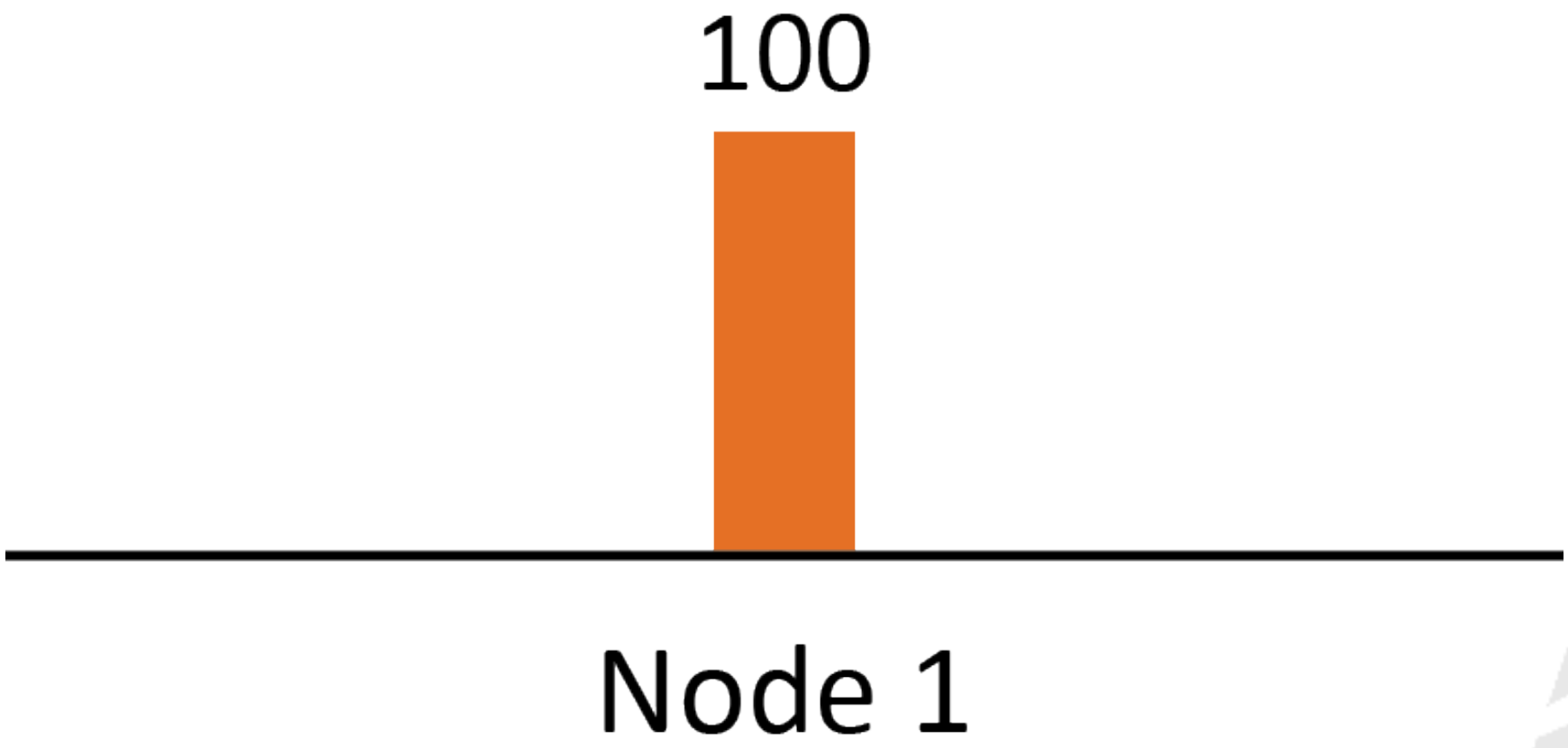
- Consistent hashing [1]
- Rendezvous hashing (HRW) [2]

[1] https://en.wikipedia.org/wiki/Consistent_hashing
[2] https://en.wikipedia.org/wiki/Rendezvous_hashing

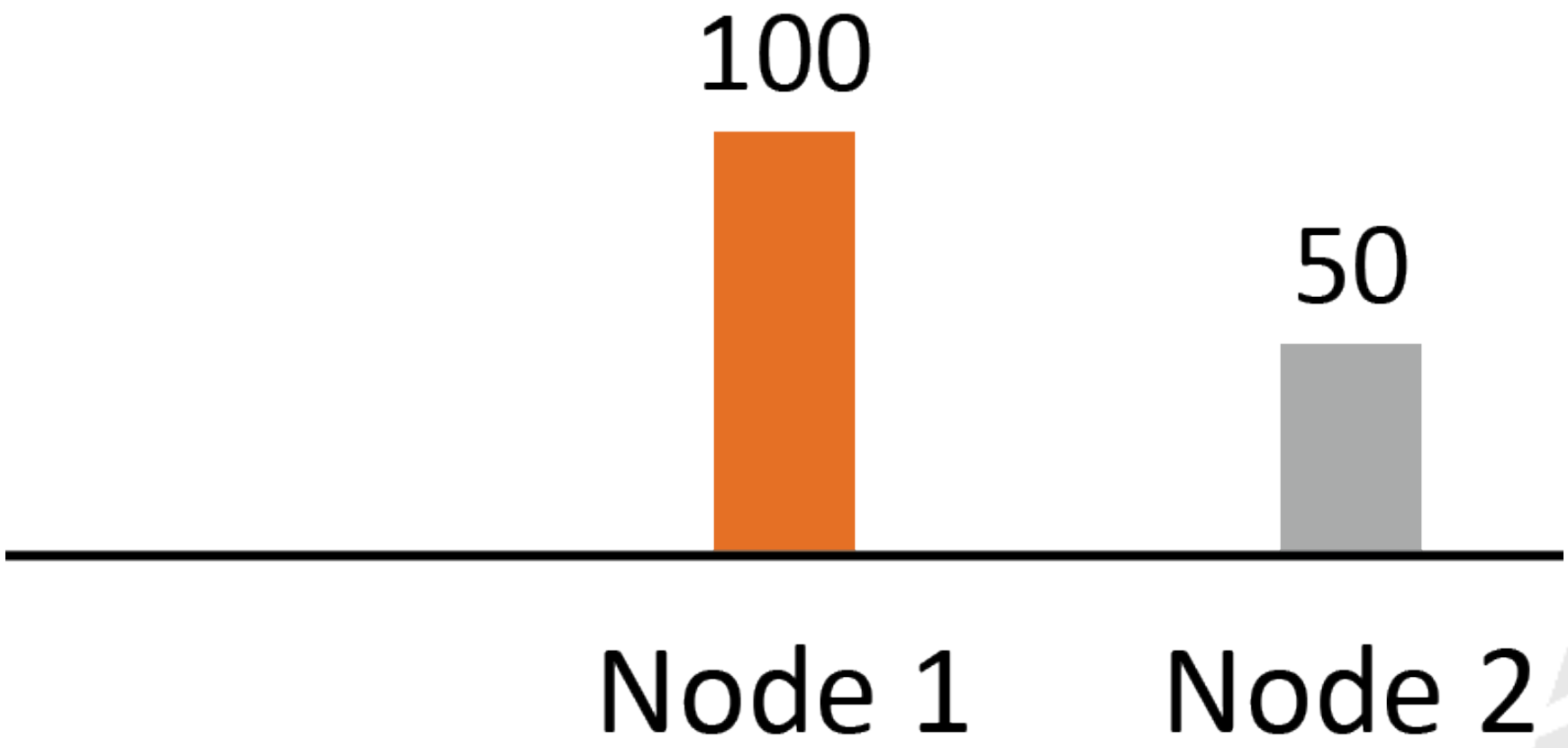
# Rendezvous affinity

```
WEIGHT(partition,
node)
```

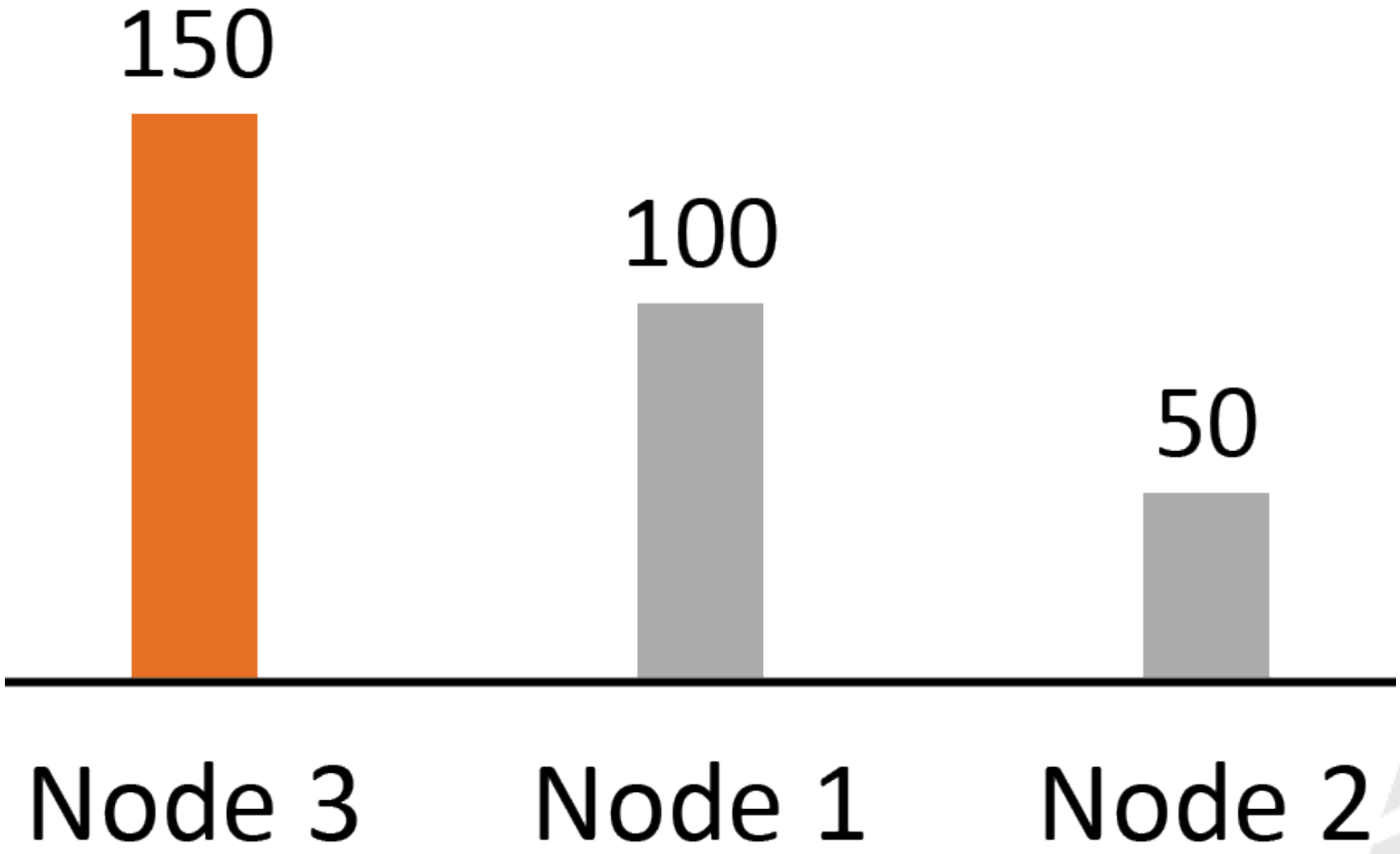
# Rendezvous affinity



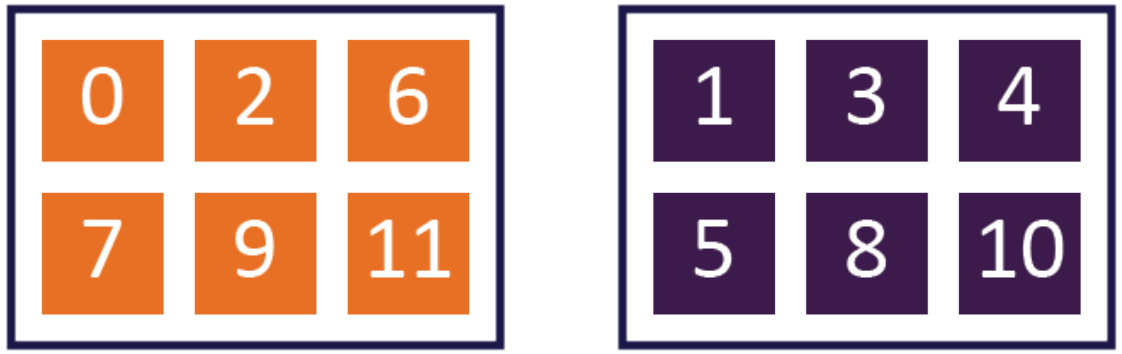
# Rendezvous affinity



# Rendezvous affinity

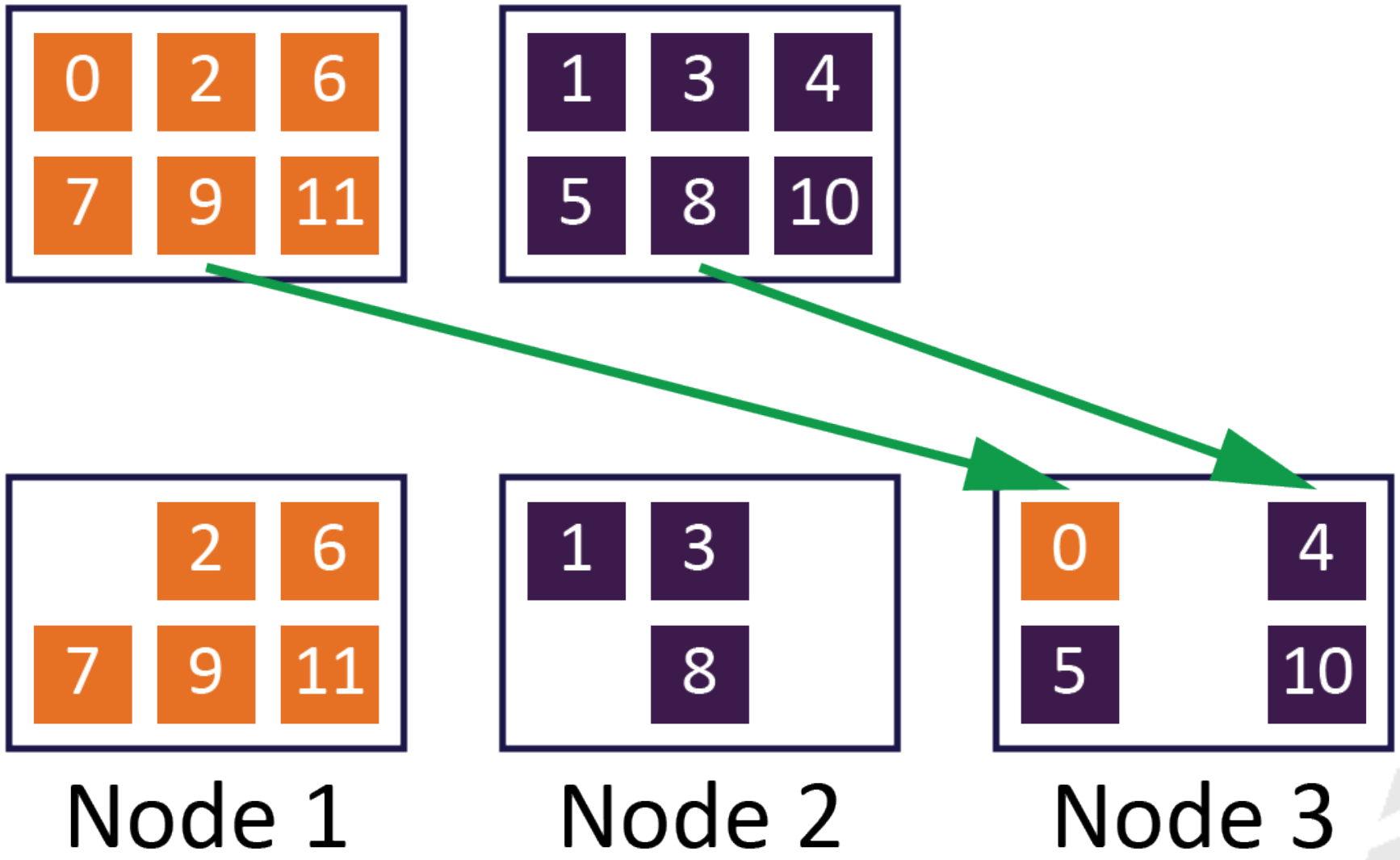


# Rendezvous affinity

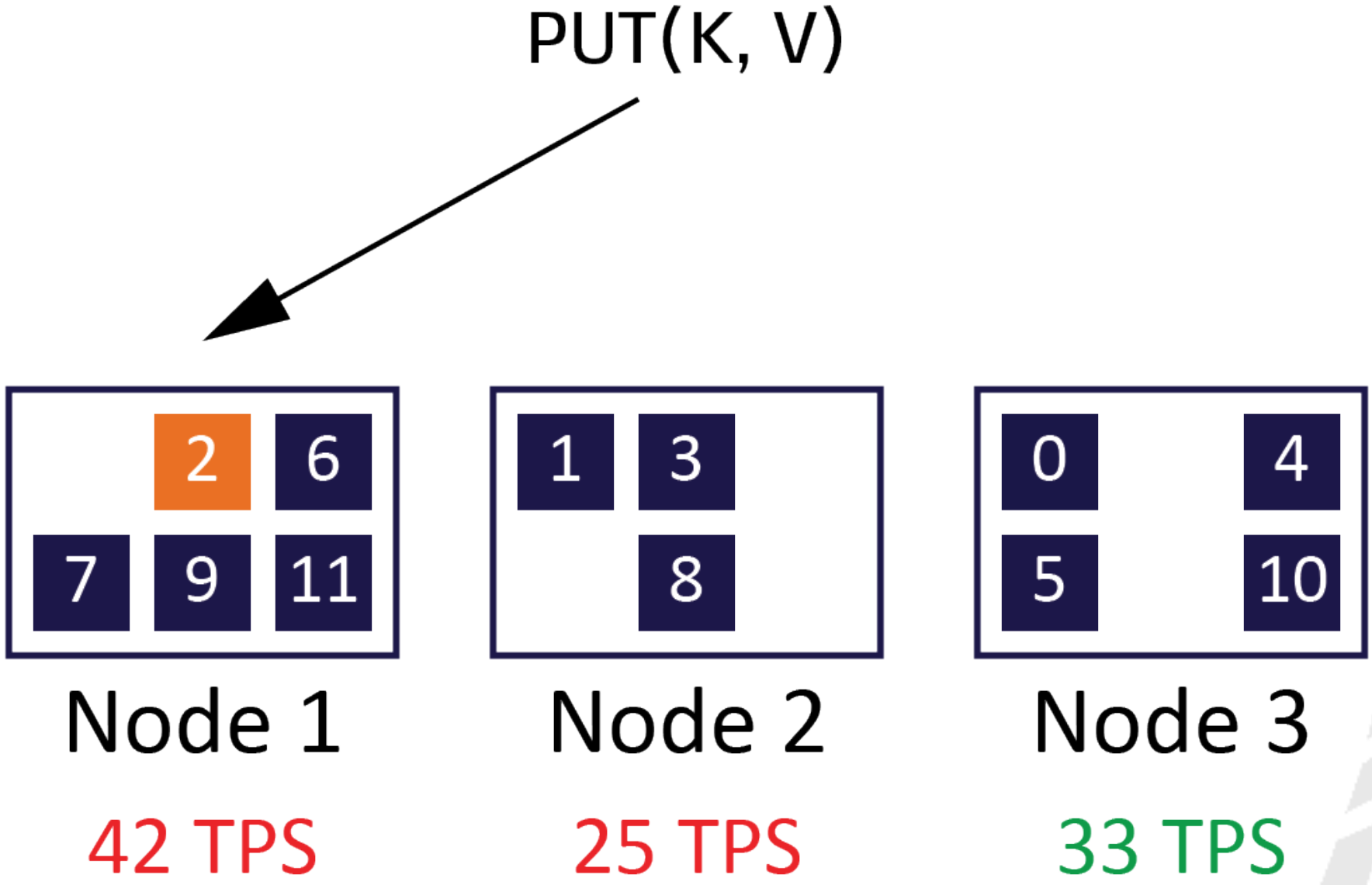



Node 1 Node 2

# Rendezvous affinity



# Rendezvous affinity

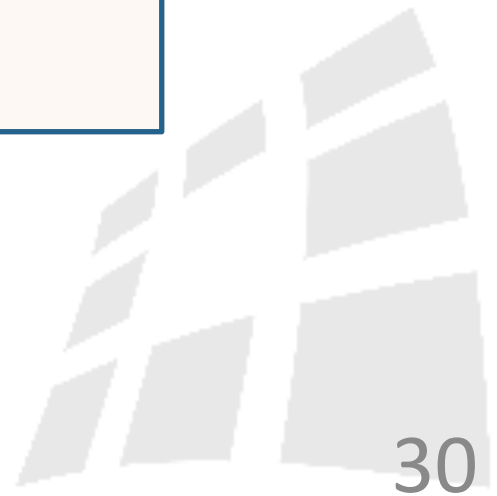


# План

- Алгоритмы шардирования
- Data co-location
- Shared-nothing архитектура
- Синхронизация в кластере

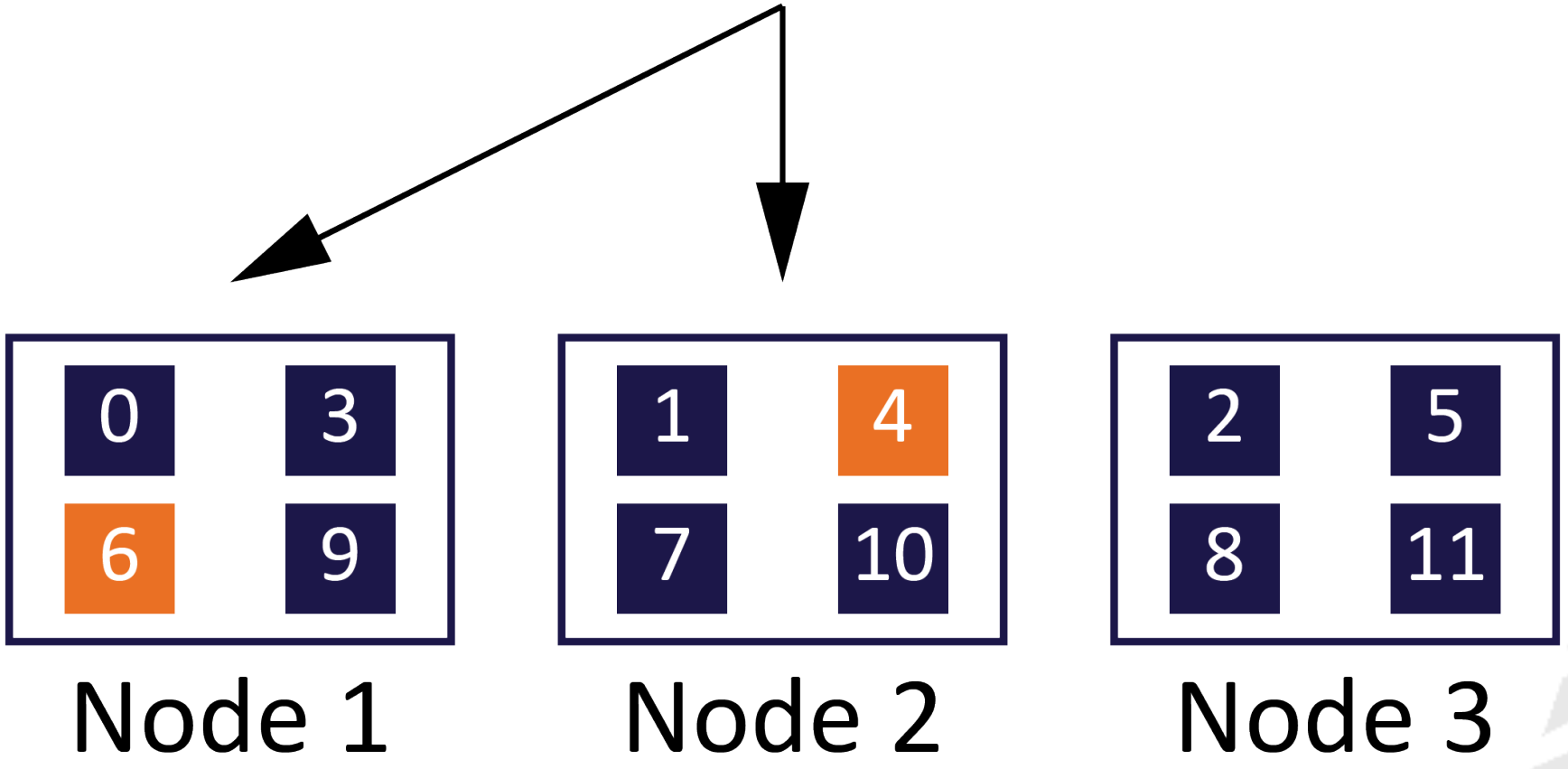
# Без co-location: транзакции

```
1: class Customer {
2:     long id;
3:     City city;
4: }
```

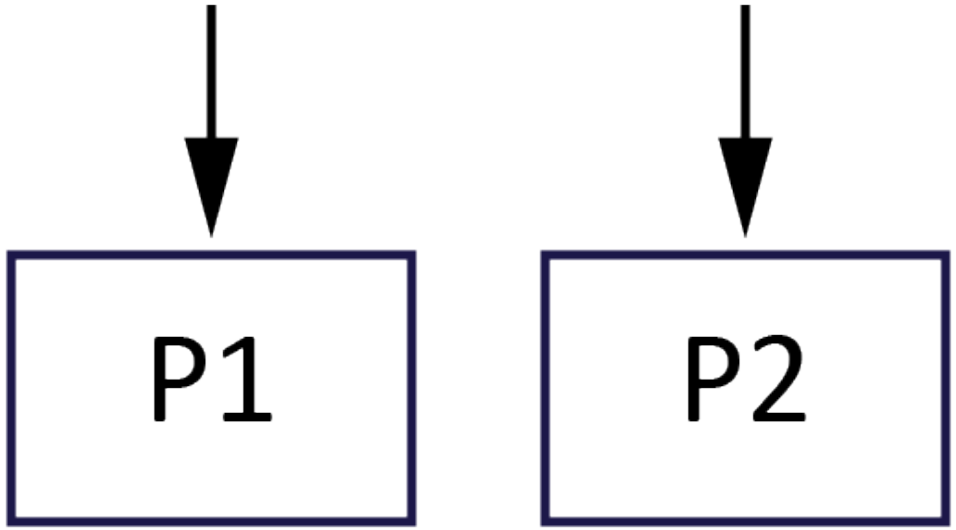


# Без co-location: транзакции

UPDATE(id=1, city=MSK)
UPDATE(id=2, city=MSK)

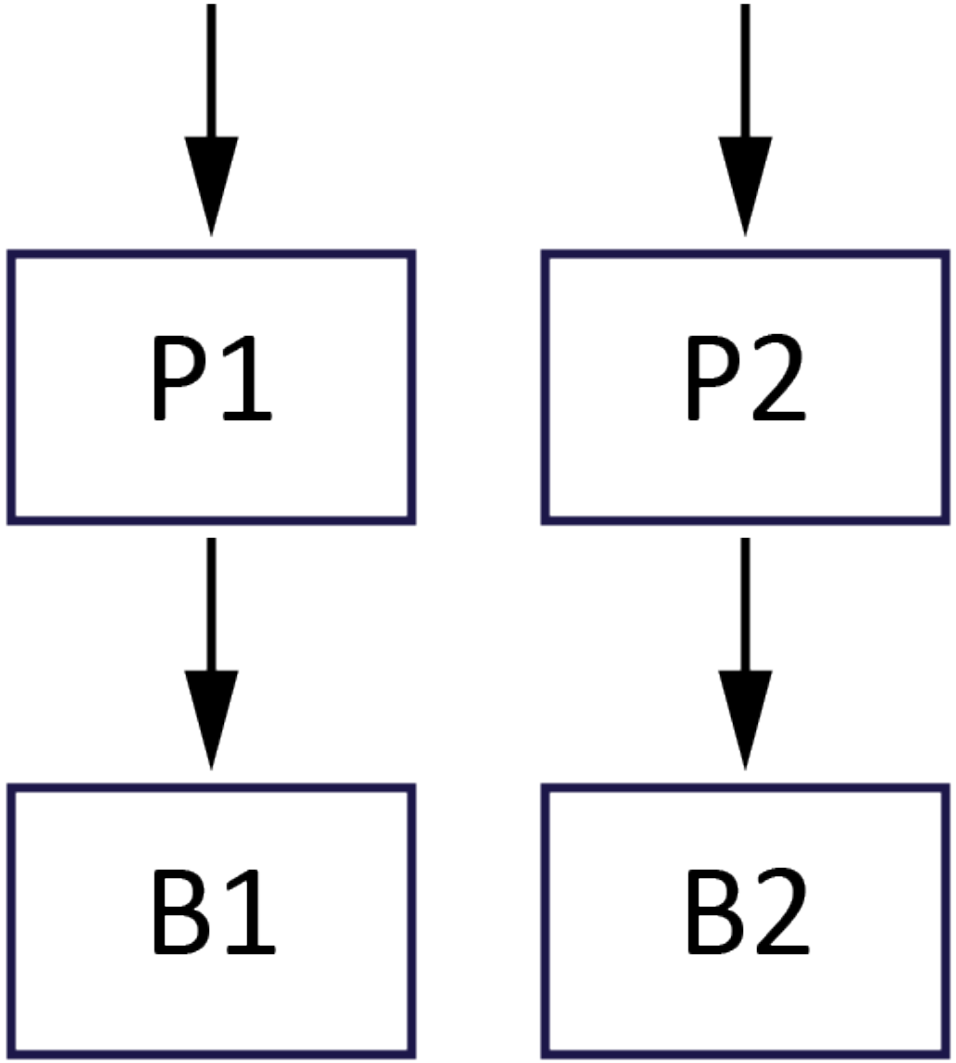


# Без co-location: транзакции



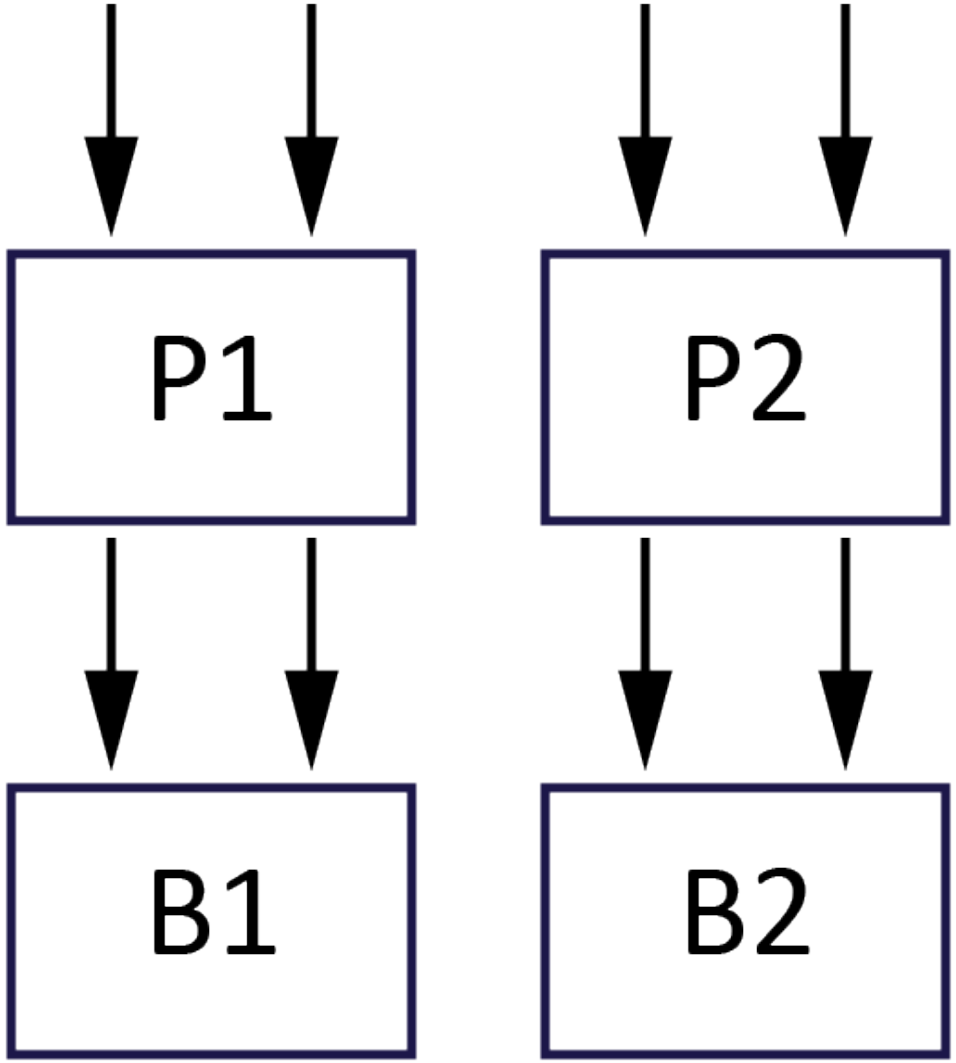

**2** (2 узла)

# Без co-location: транзакции



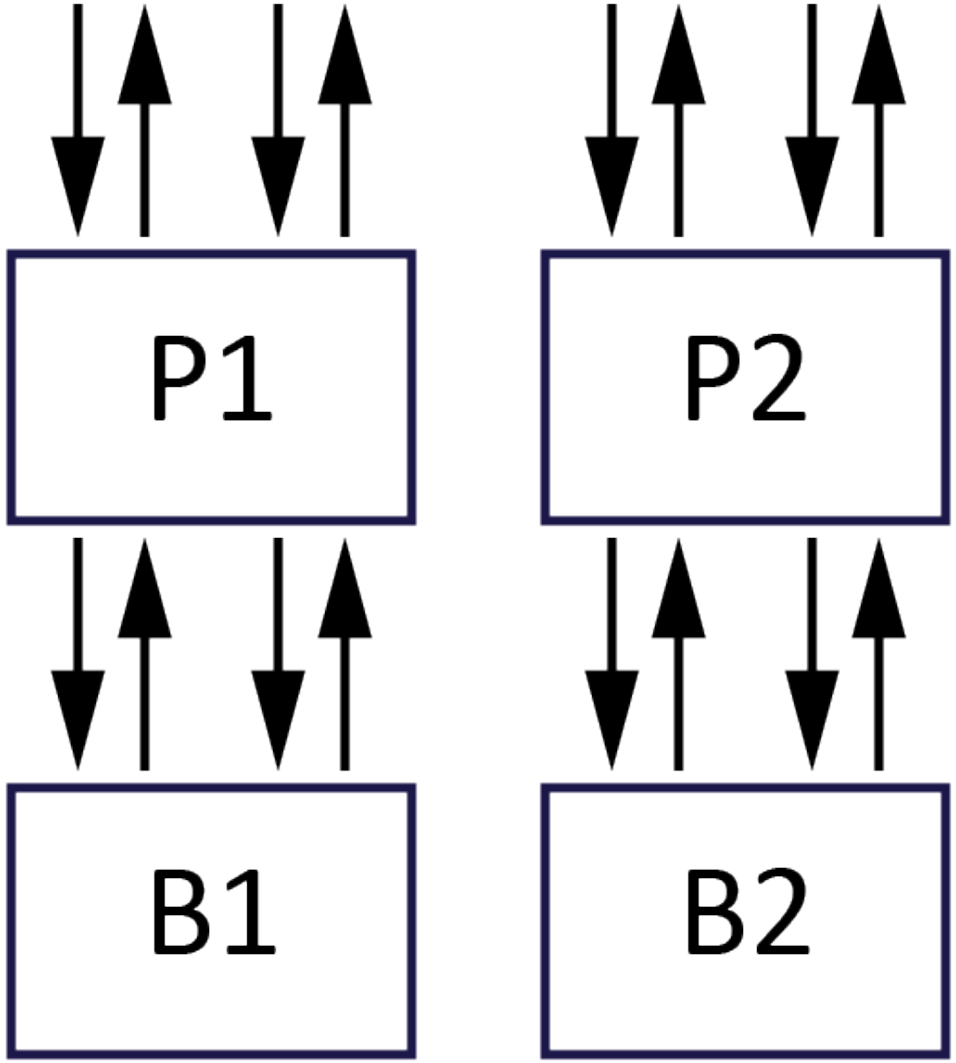

**2** (2 узла)
**2** (primary + backup)

# Без co-location: транзакции



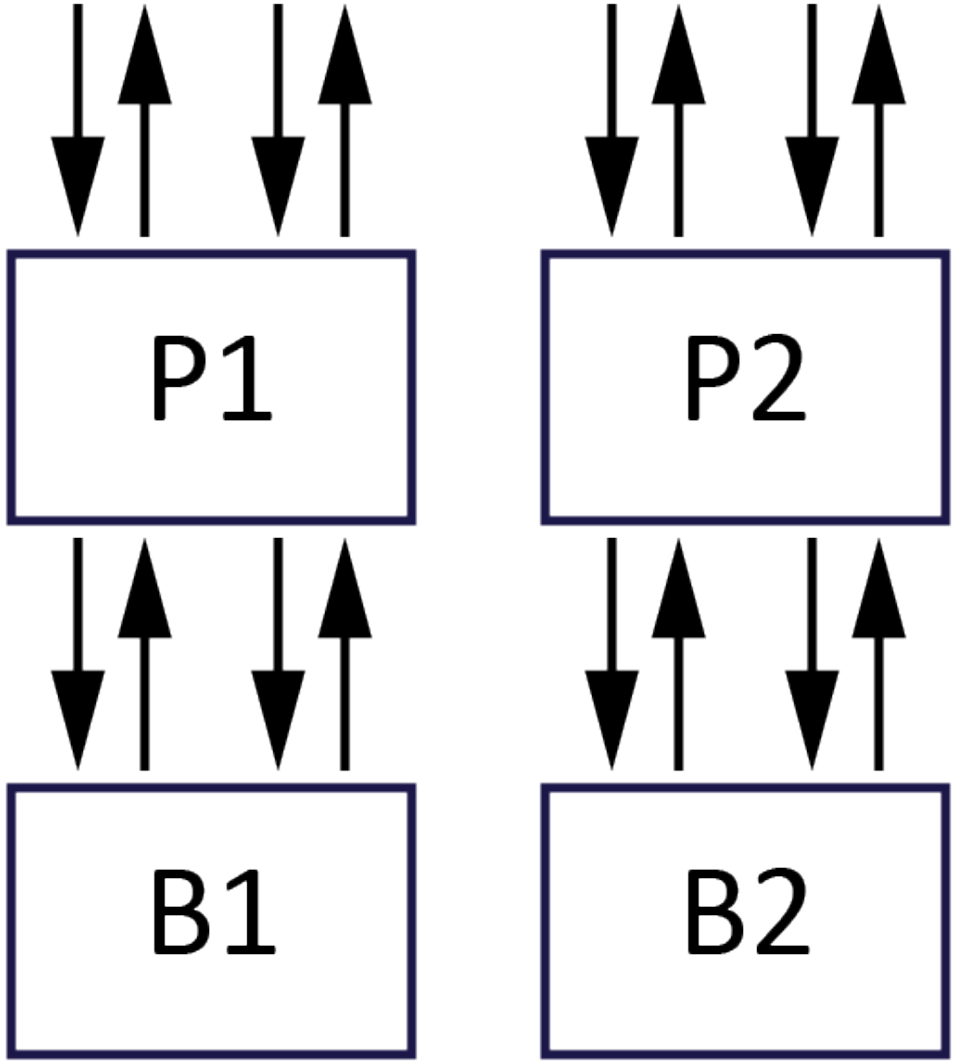

**2** (2 узла)
**2** (primary + backup)
**2** (two-phase commit)

# Без co-location: транзакции




**2** (2 узла)
**2** (primary + backup)
**2** (two-phase commit)
**2** (request-response)

# Без co-location: транзакции



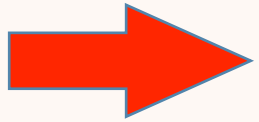

```
  2  (2 узла)
  2  (primary + backup)
  2  (two-phase commit)
  2  (request-response)
--
```

**16 сообщений**

# Провод

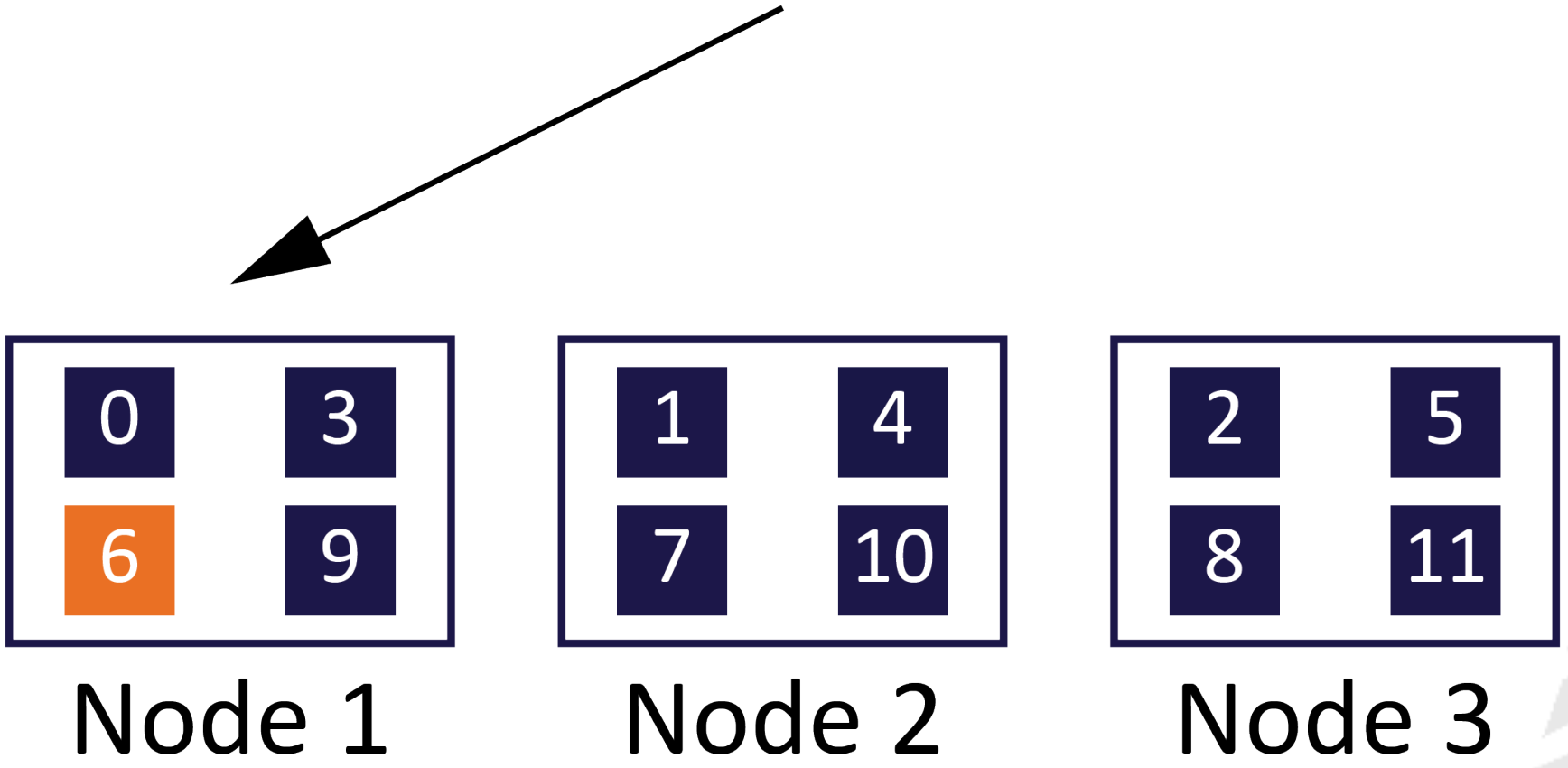
# Co-location: транзакции

```
1: class Customer {
2:     long id;
3:
4:     @AffinityKeyMapped
5:     City city;
6: }
```

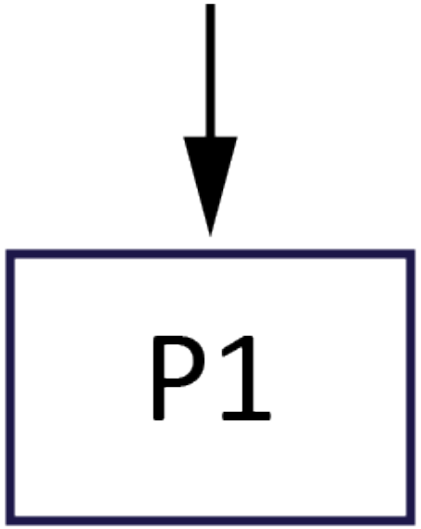


# Co-location: транзакции

UPDATE(id=1, city=MSK)
UPDATE(id=2, city=MSK)

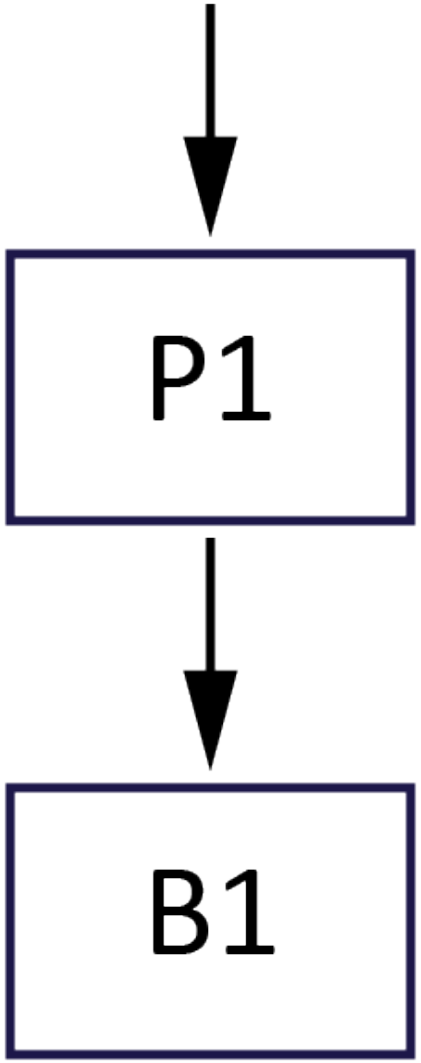


# Co-location: транзакции



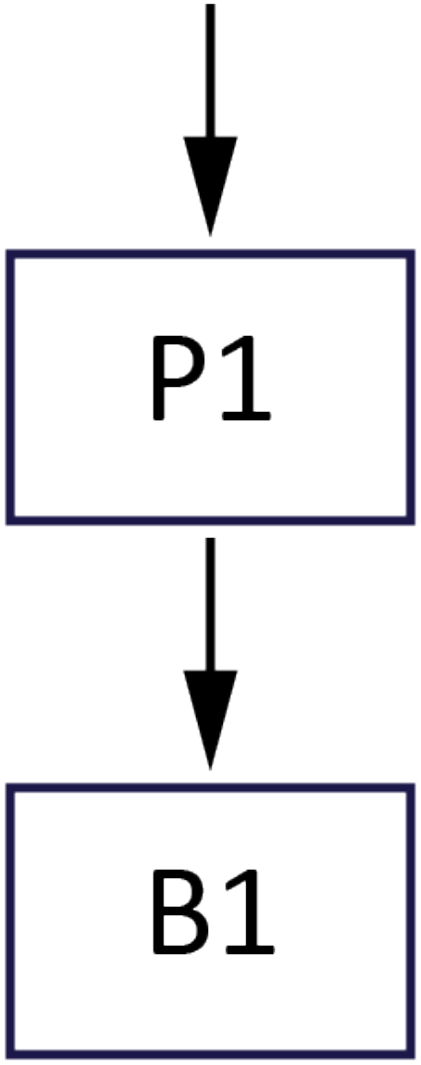

**1** (1 узел)

# Co-location: транзакции



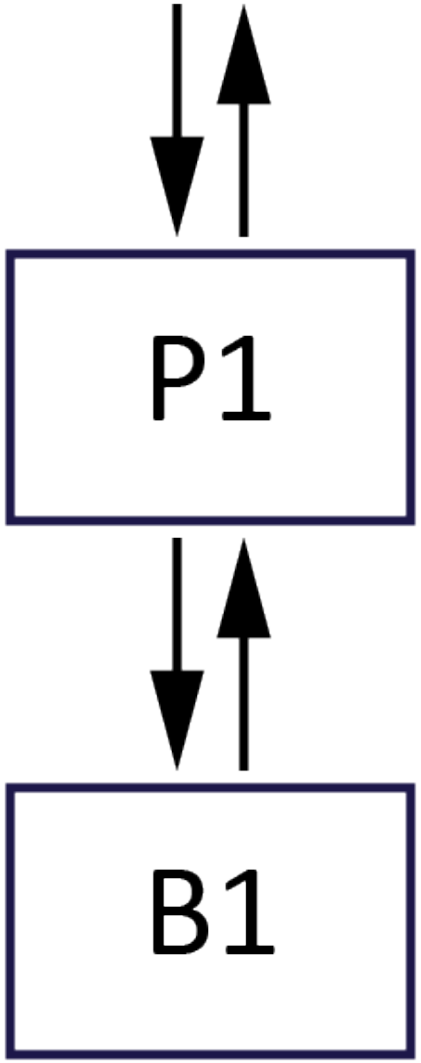

```
1  (1 узел)
2  (primary + backup)
```

# Co-location: транзакции



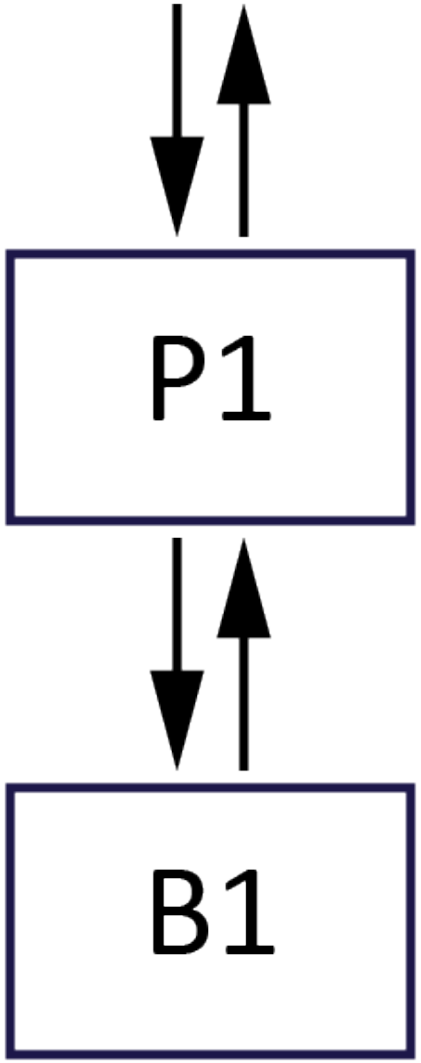

```
1  (1 узел)
2  (primary + backup)
1  (one-phase commit)
```

# Co-location: транзакции



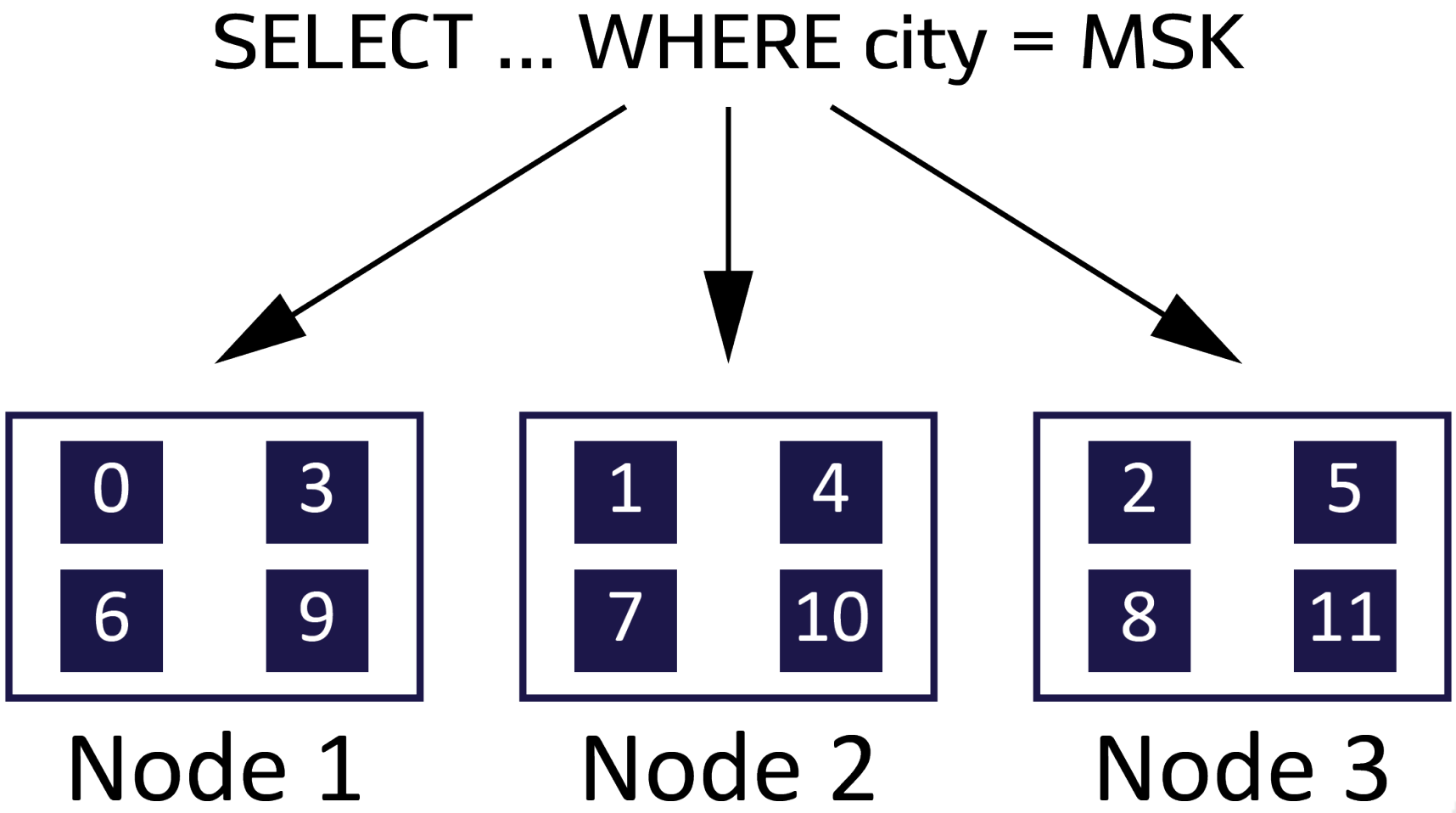

**1** (1 узел)
**2** (primary + backup)
**1** (one-phase commit)
**2** (request-response)

# Co-location: транзакции




```
1  (1 узел)
2  (primary + backup)
1  (one-phase commit)
2  (request-response)
--
4  сообщения
```

# Запустим SQL

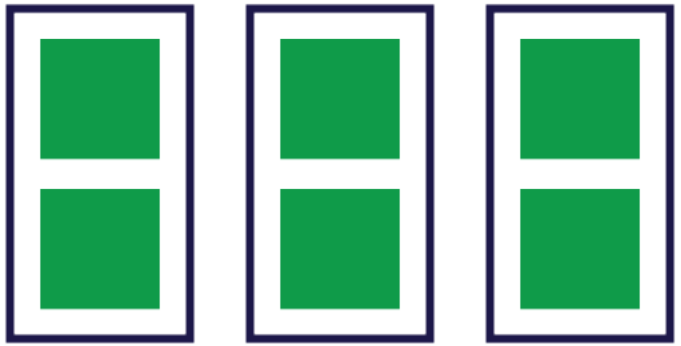


# Без co-location: full scan

1 node

# Без co-location: full scan

1 node



3 nodes

# Без co-location: full scan

- **1/3x** latency!

# Без co-location: full scan

- **1/3x** latency!
- **3x** capacity!

# Индексы

А с индексами же еще круче будет?

# Индексы

1 node

# Индексы

1 node

N nodes

# Посчитаем сложность

$$\log 1\_000\_000 \approx 20$$

# Посчитаем сложность

$$\log 1\_000\_000 \approx 20$$

**vs**

$$\log 333\_333 \approx 18$$

$$\log 333\_333 \approx 18$$

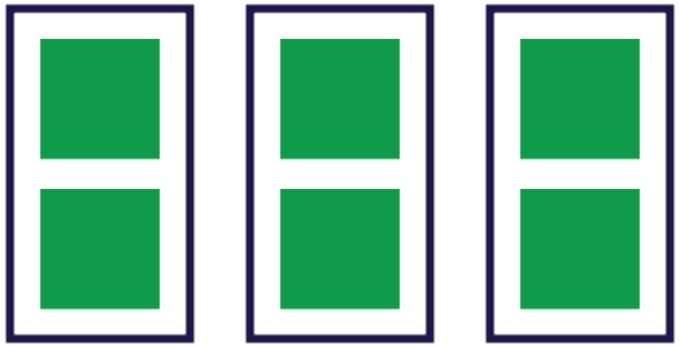
$$\log 333\_333 \approx 18$$

# Посчитаем сложность

$$\log N < M * \log N/M$$

# Без co-location: full scan

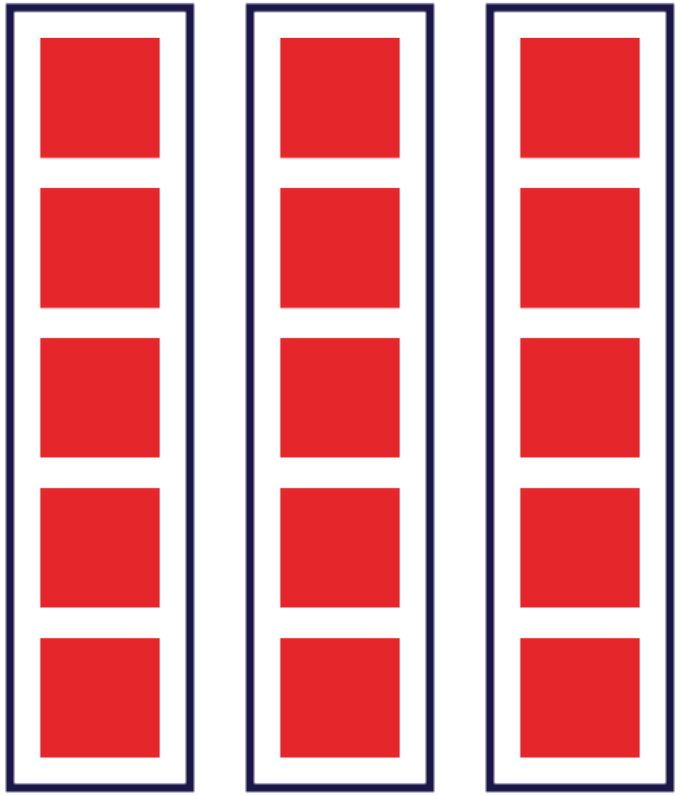
1 node



3 nodes

# Без co-location: index seek

1 node

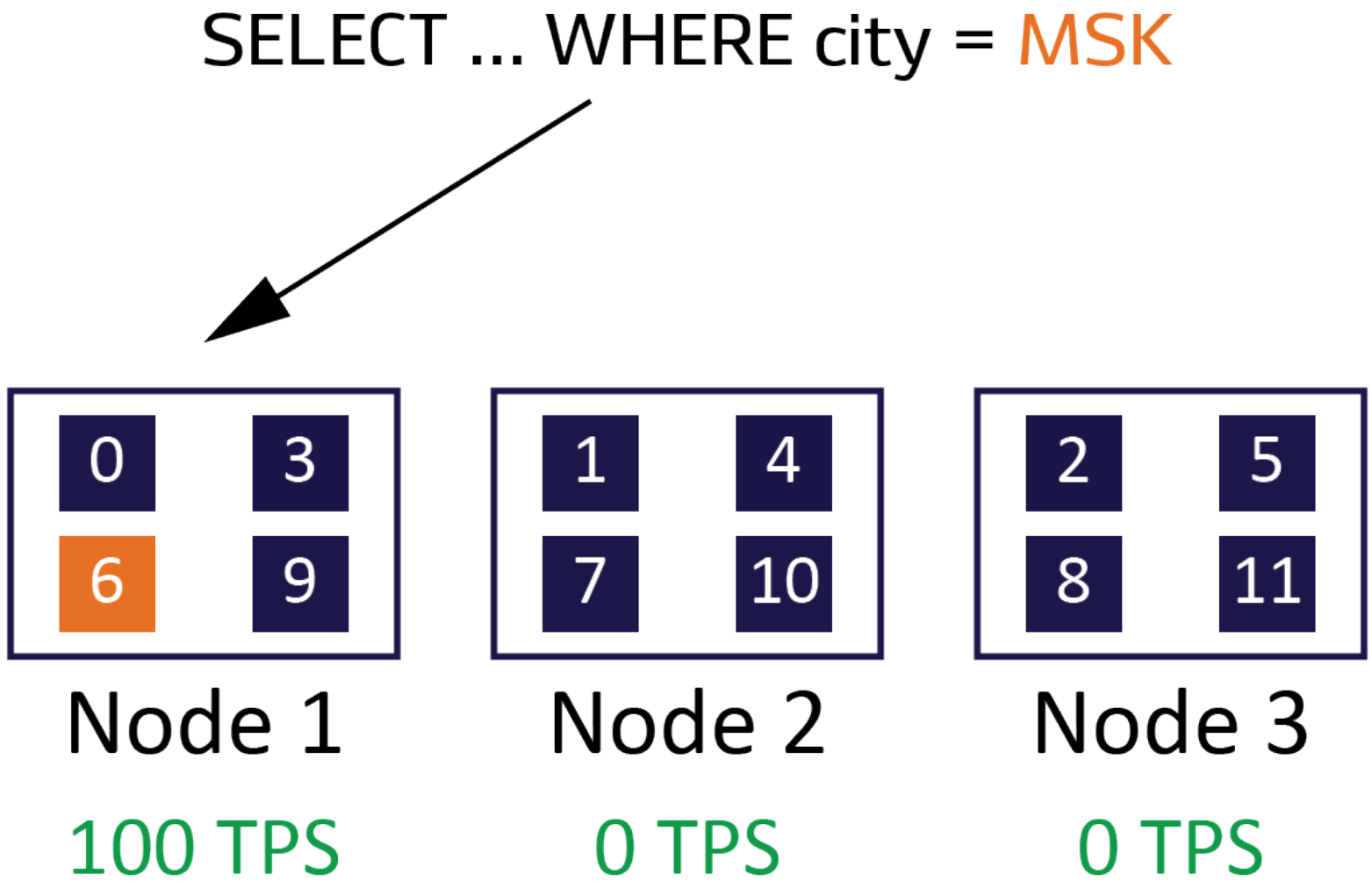


3 nodes

# Без co-location: index seek

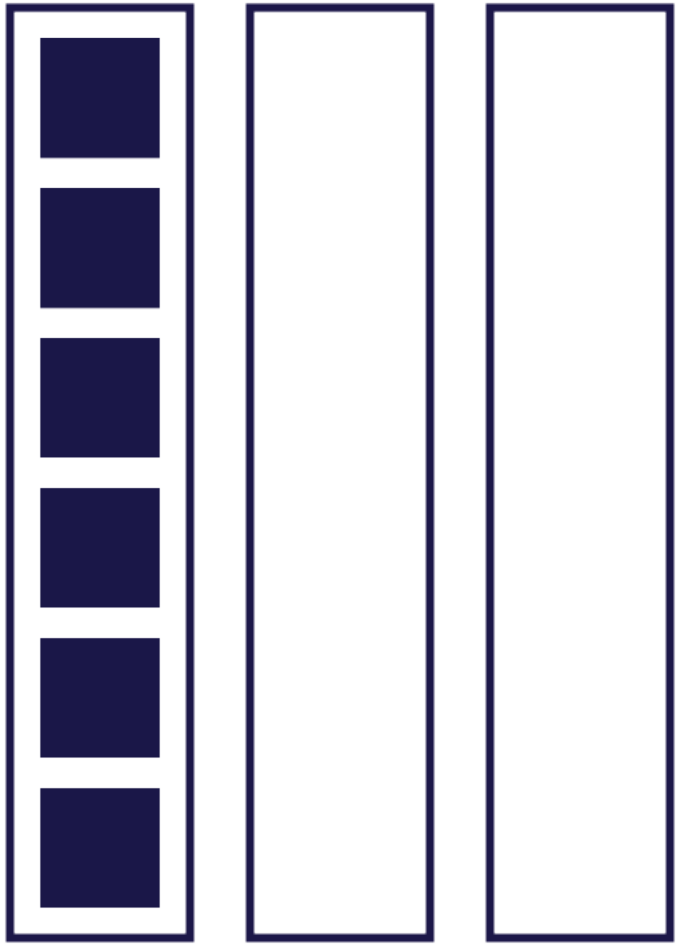
- **такой же** latency?
- **такой же** capacity?

# Co-location: index seek



# Co-location: index seek

1 node



3 nodes

# Co-location: index seek

- **такой же** latency?
- **но 3x** capacity!

# Дисбаланс


Node 1

100 TPS

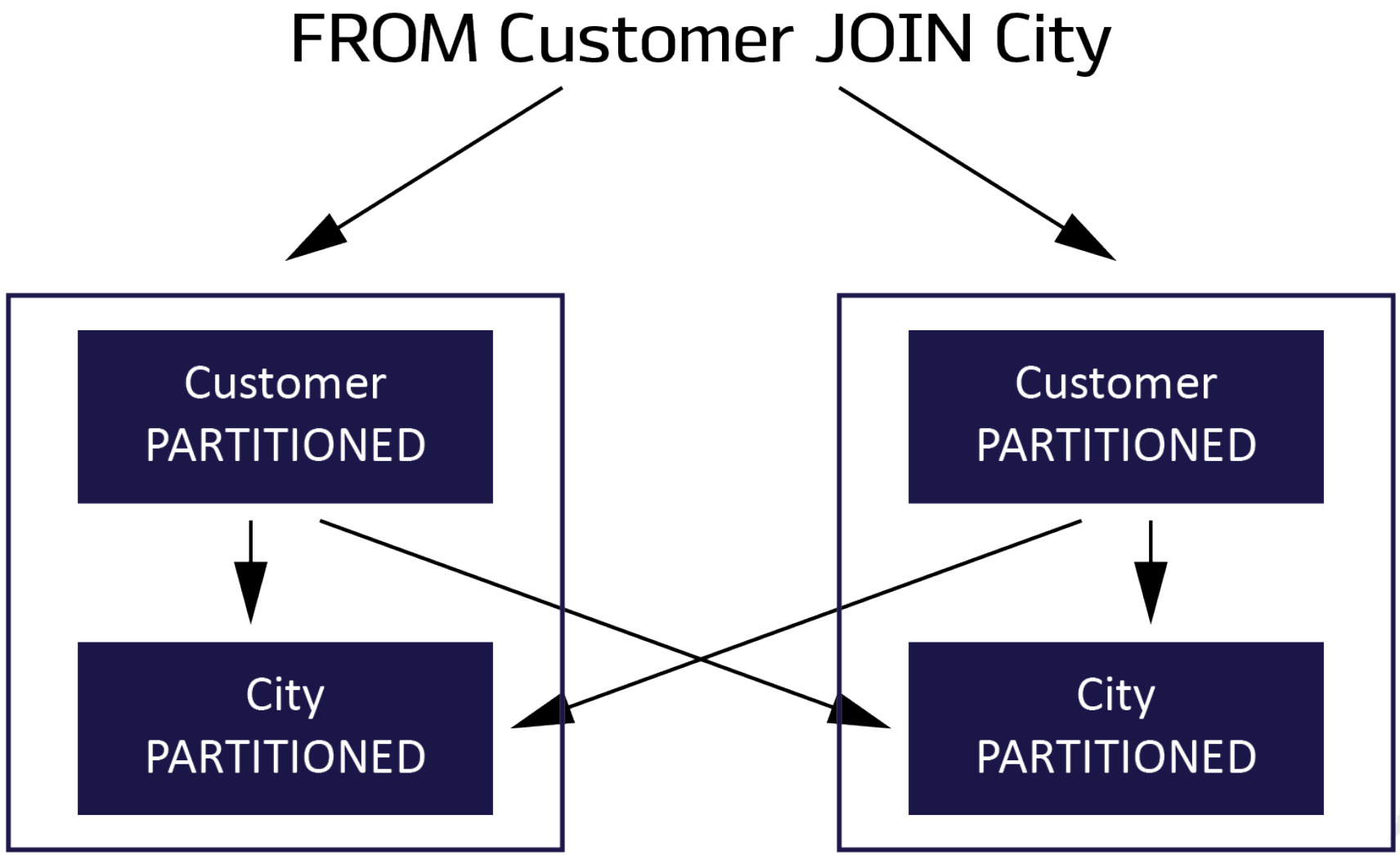

Node 2

50 TPS

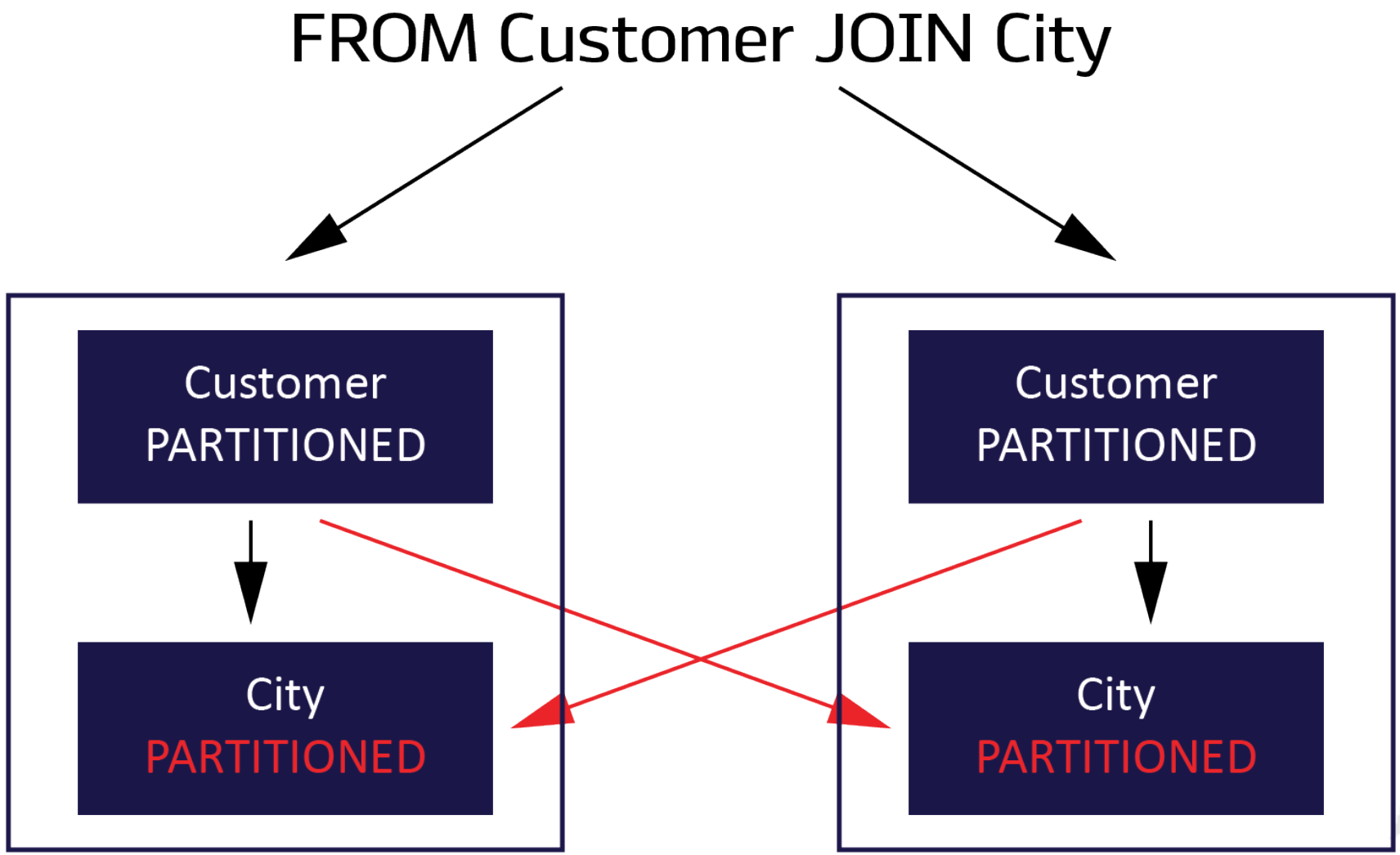

Node 3

10 TPS

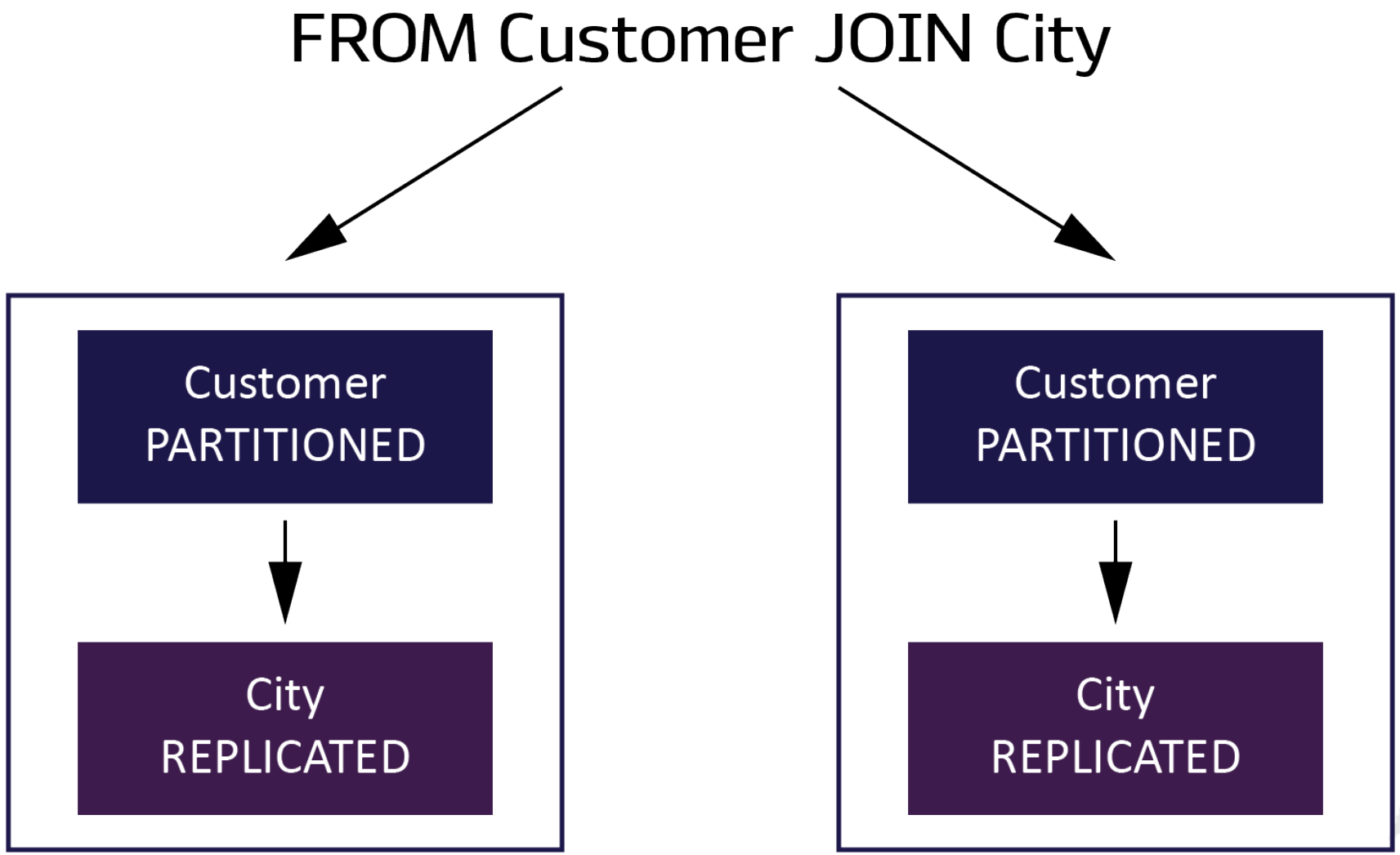
# PARTITIONED cache



# PARTITIONED cache



# REPLICATED cache



# Адаптация модели

Local-distributed **impedance mismatch**?

# План

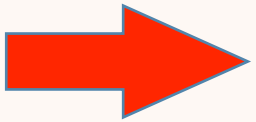
- Алгоритмы шардирования
- Data co-location
- Синхронизация в кластере
- Локальная архитектура многопоточности

# Счетчик: локально

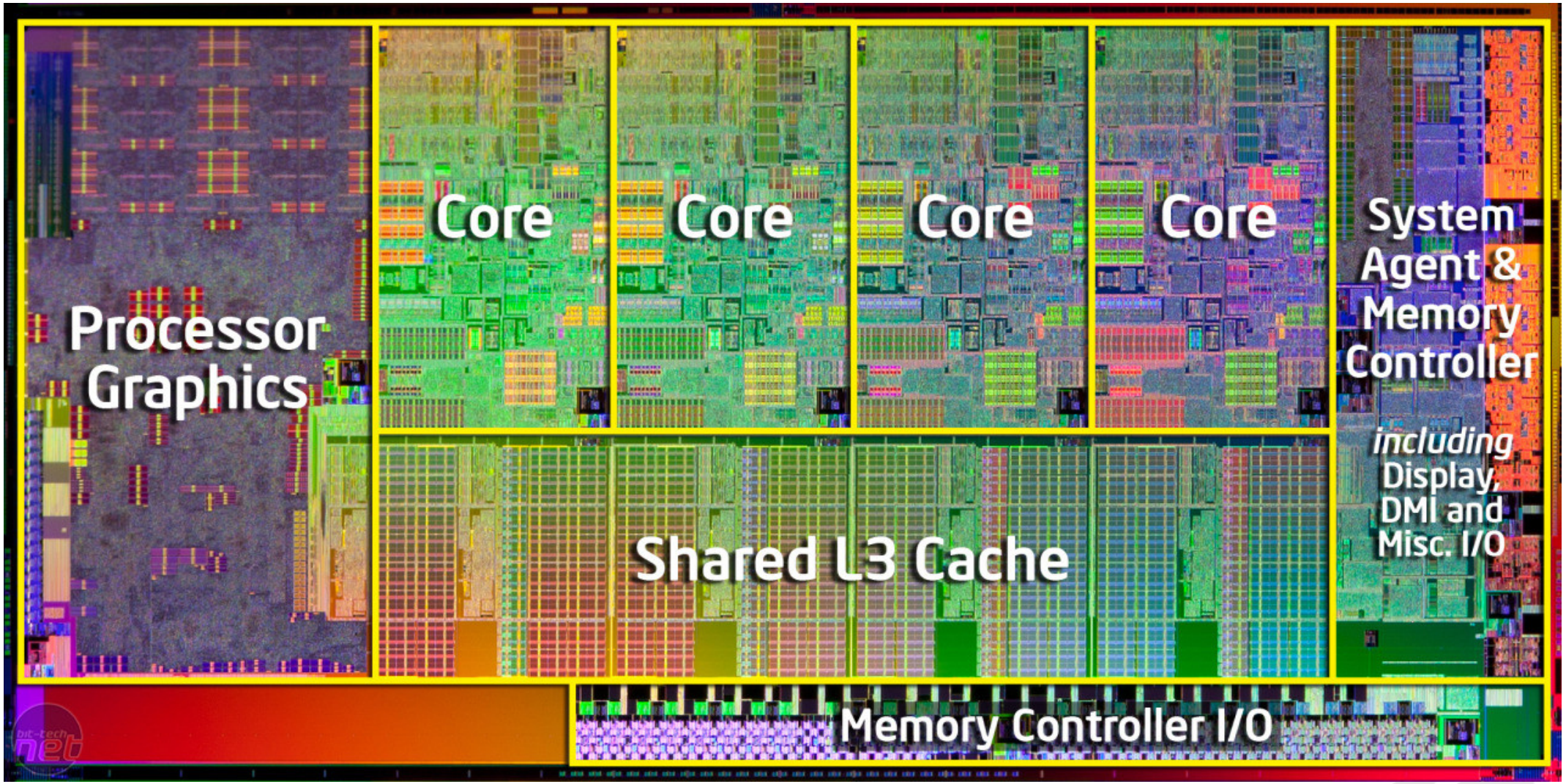
```
1: AtomicLong ctr;
2:
3: long getNext() {
4:     return ctr.incrementAndGet();
5: }
```

# Счетчик: локально, велосипедно

```
 1: AtomicLong ctr;
 2: ThreadLocal<Long> localCtr;
 3:
 4: long getNext() {
 5:     long res = localCtr.get();
 6:
 7:     if (res % 1000 == 0)
 8:         res = ctr.getAndAdd(1000);
 9:
10:     localCtr.set(++res);
11:
12:     return res;
13: }
```



# Процессор

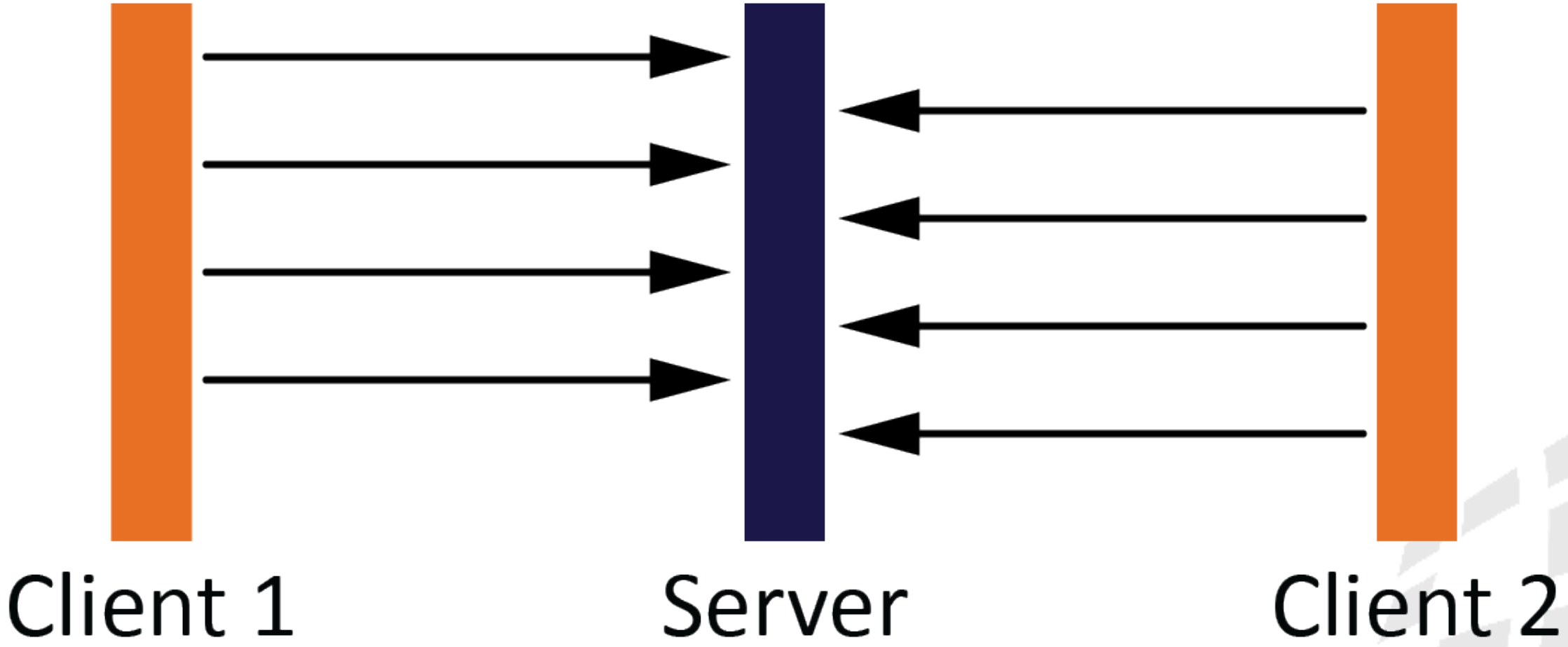

Processor Graphics
Core
Core
Core
Core
System Agent & Memory Controller
including Display, DMI and Misc. I/O
Shared L3 Cache
Memory Controller I/O

# Провод

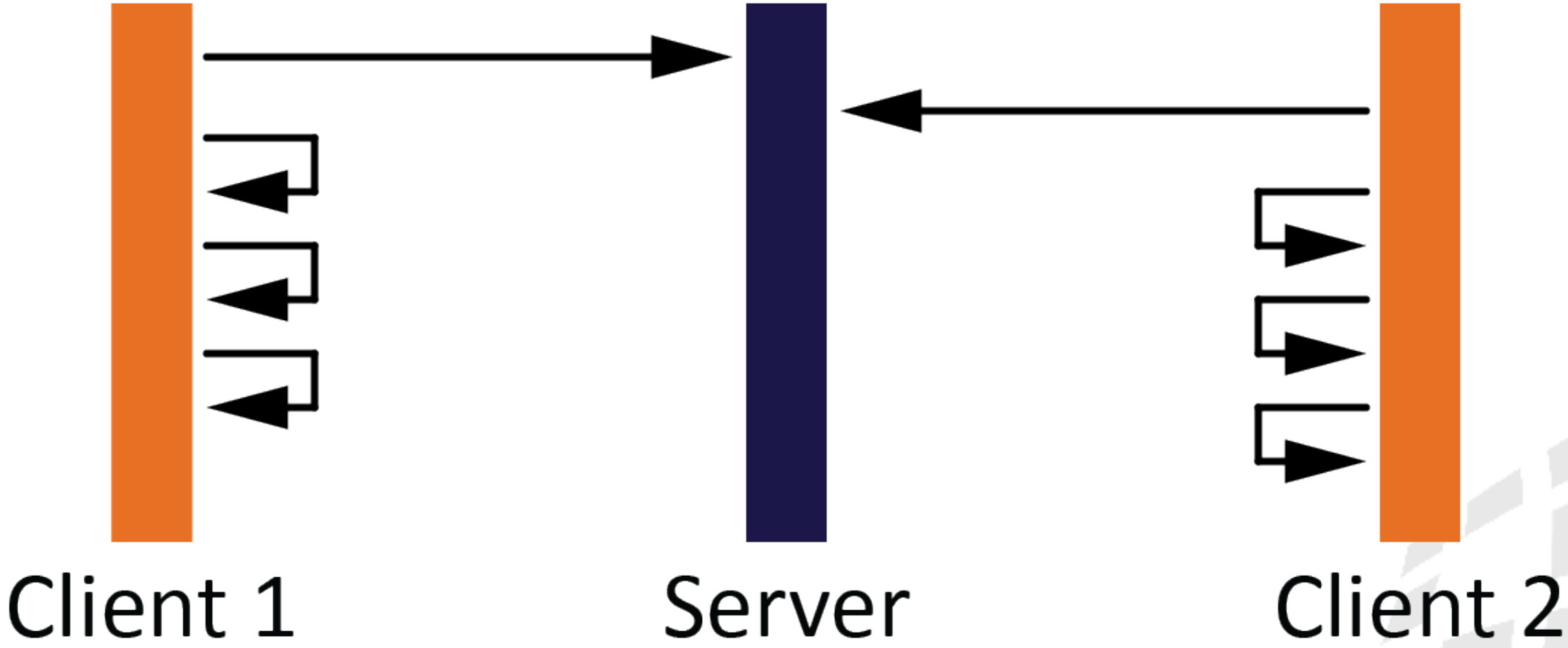
# Счетчик: в кластере

- Локально: **миллионы**
- Распределенно: **тысячи**

# IgniteAtomicLong



# IgniteAtomicSequence

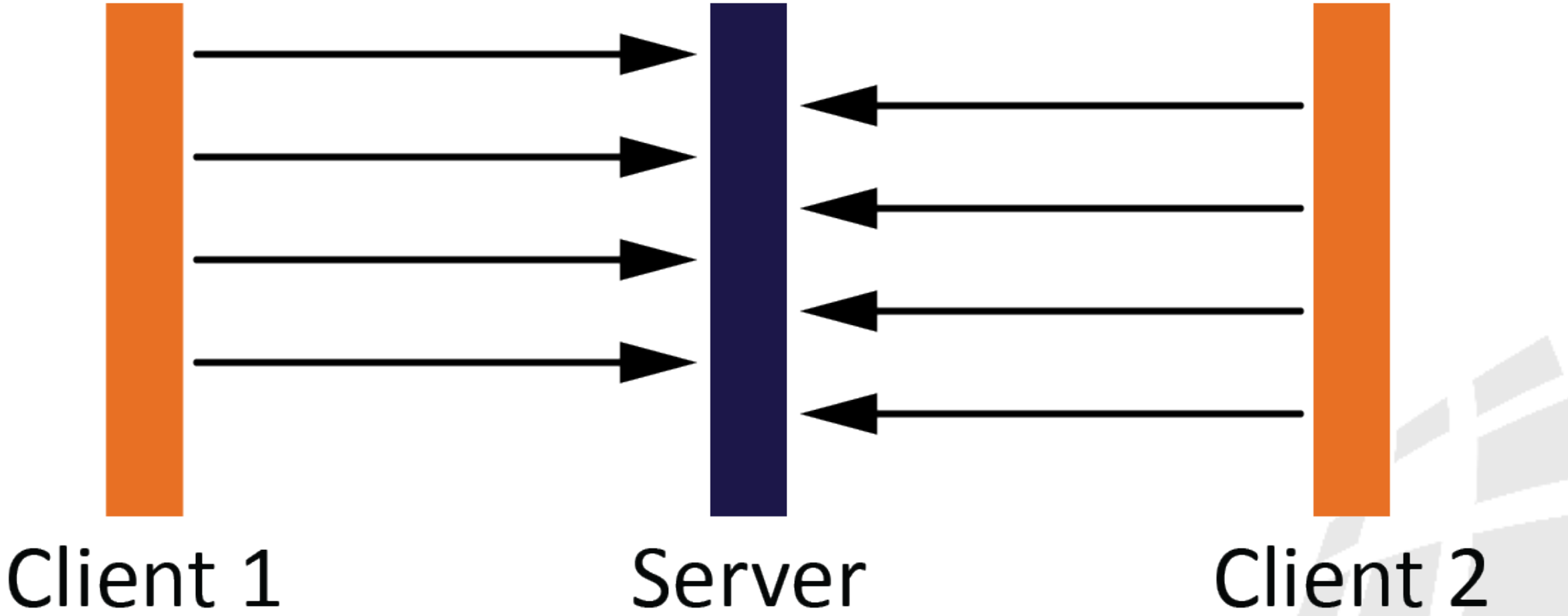


# Разумные требования: счетчик

Уникальный, монотонно возрастающий, 8 байт!

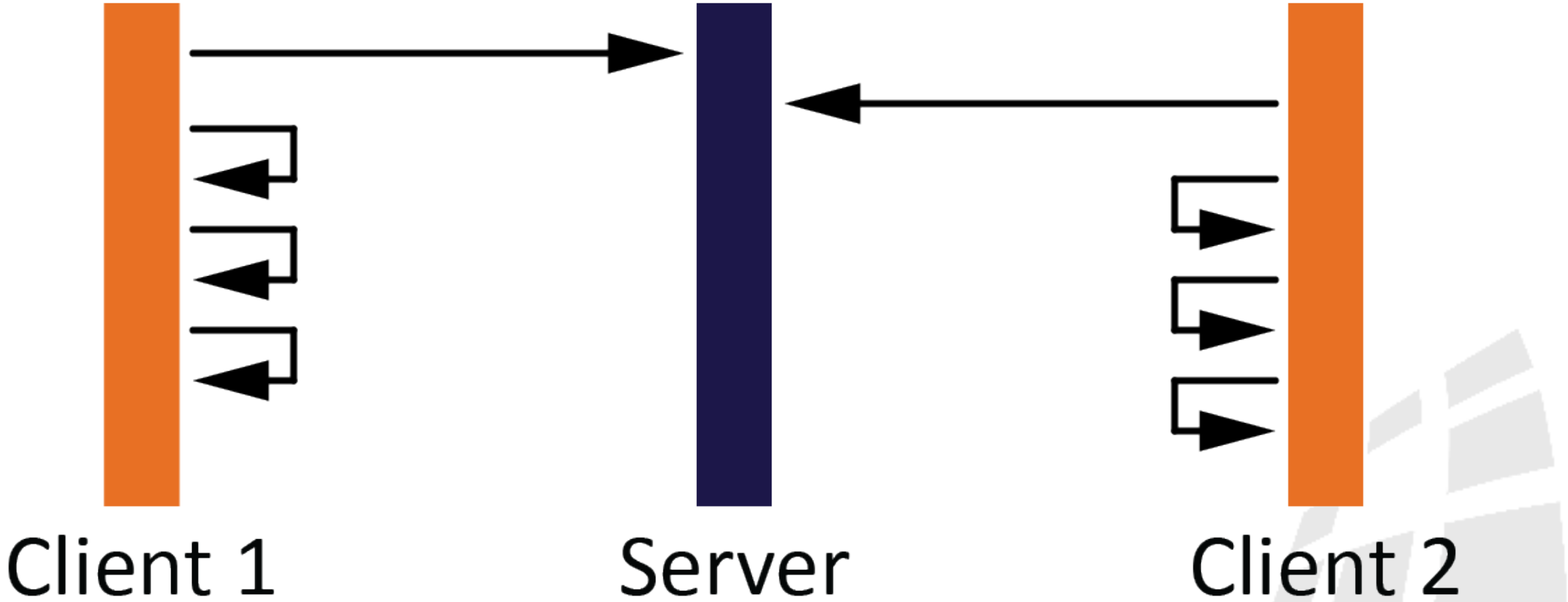
# Разумные требования: счетчик

Уникальный, монотонно возрастающий, 8 байт!



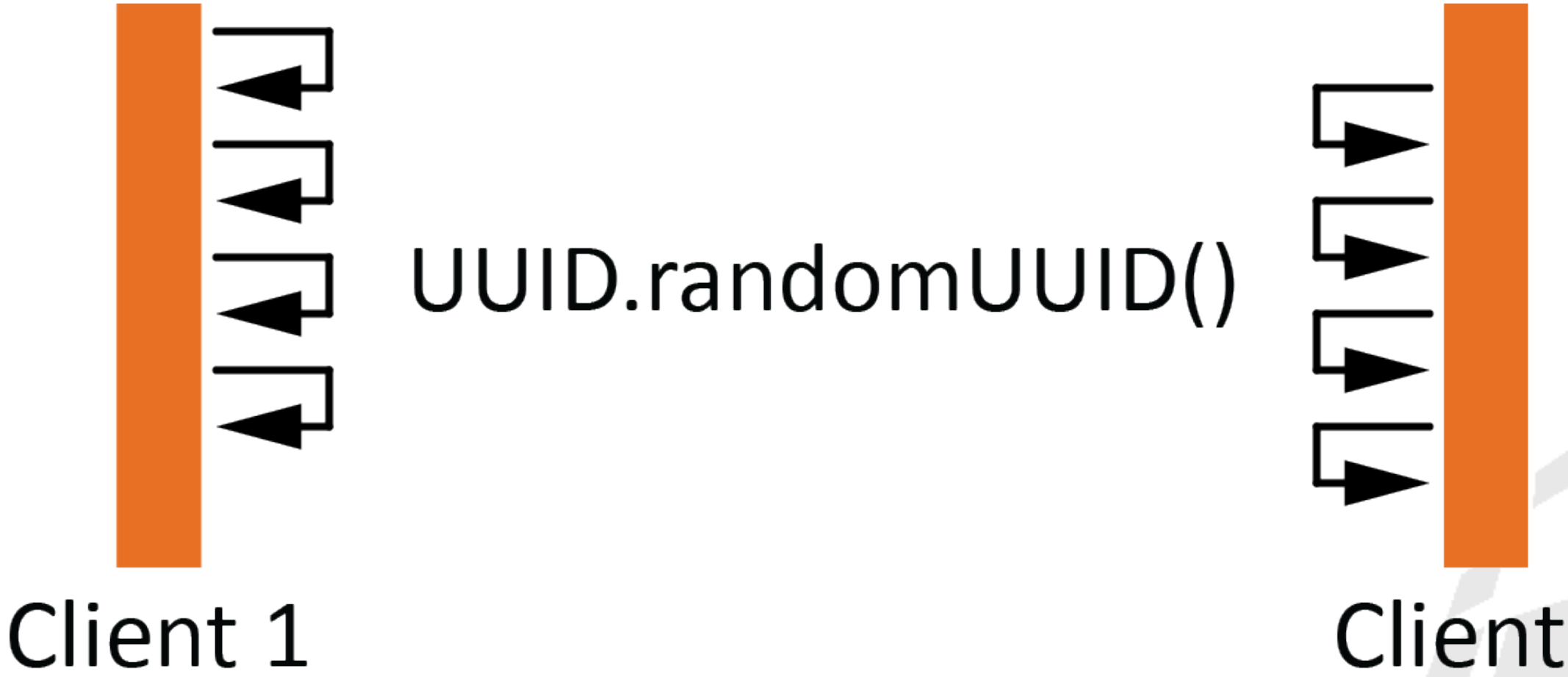
# Разумные требования: счетчик

Уникальный, ~~монотонно возрастающий~~, 8 байт!



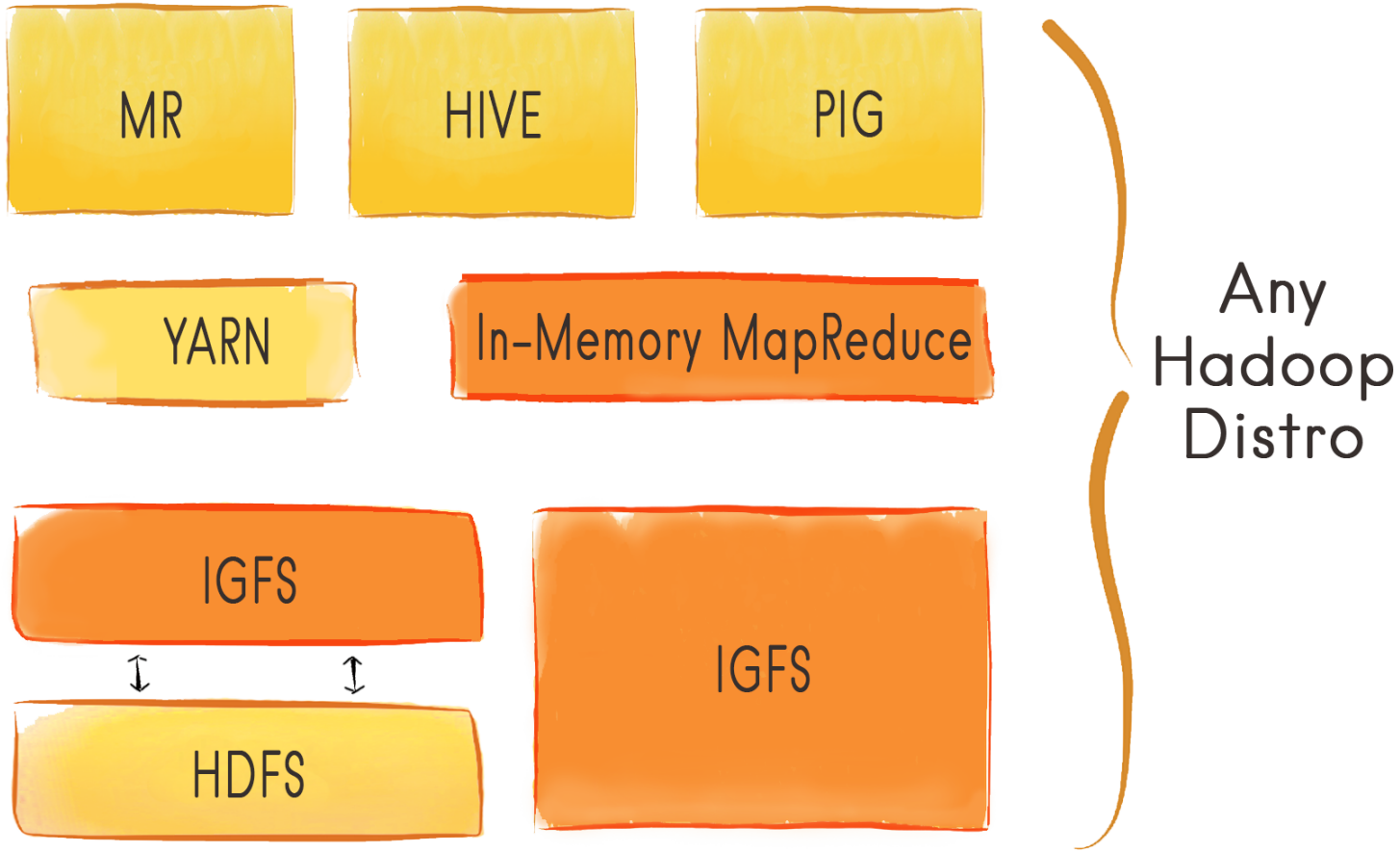
# Разумные требования: счетчик

Уникальный, ~~монотонно возрастающий~~, ~~8 байт~~!

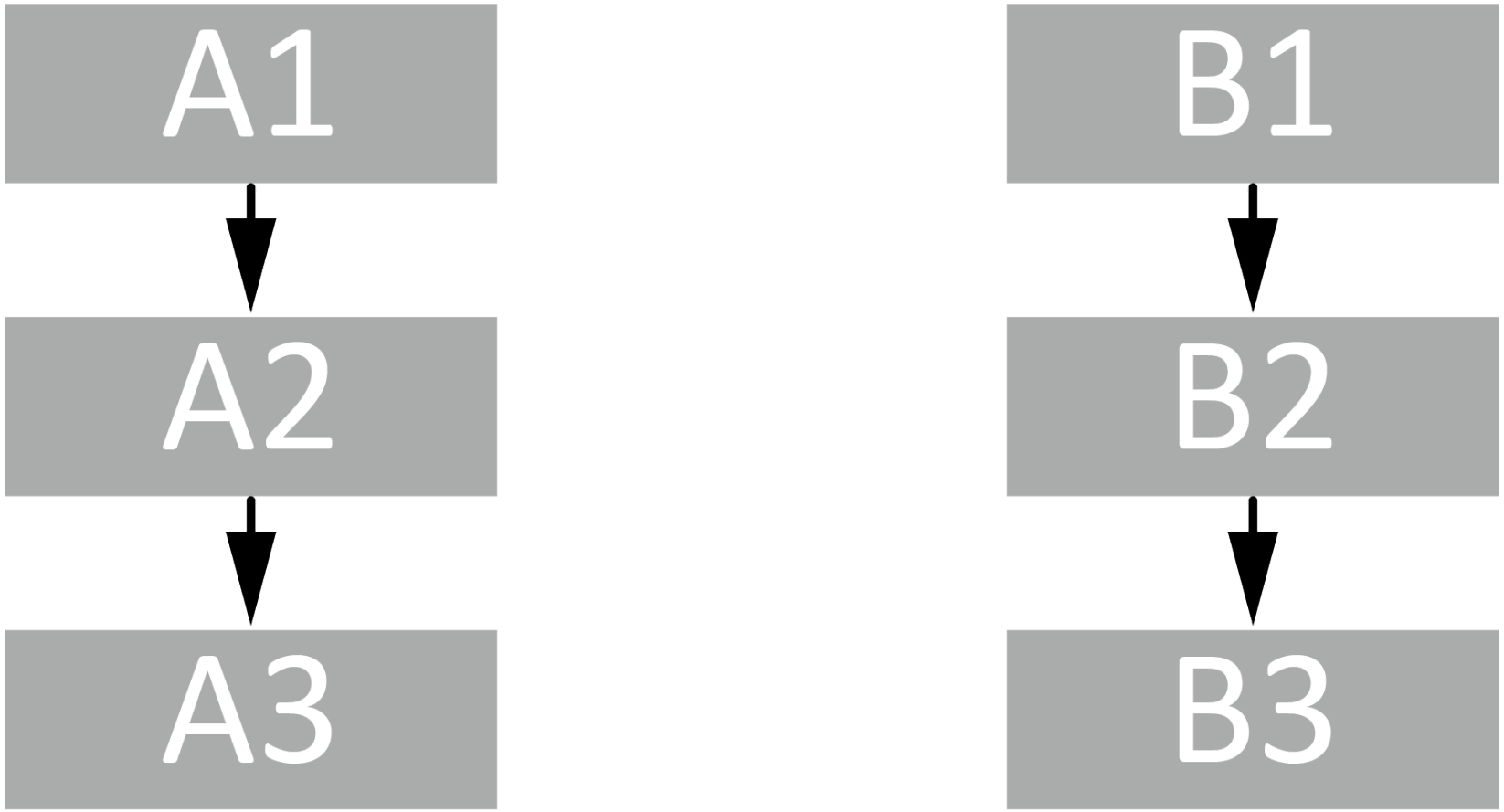


# Координация

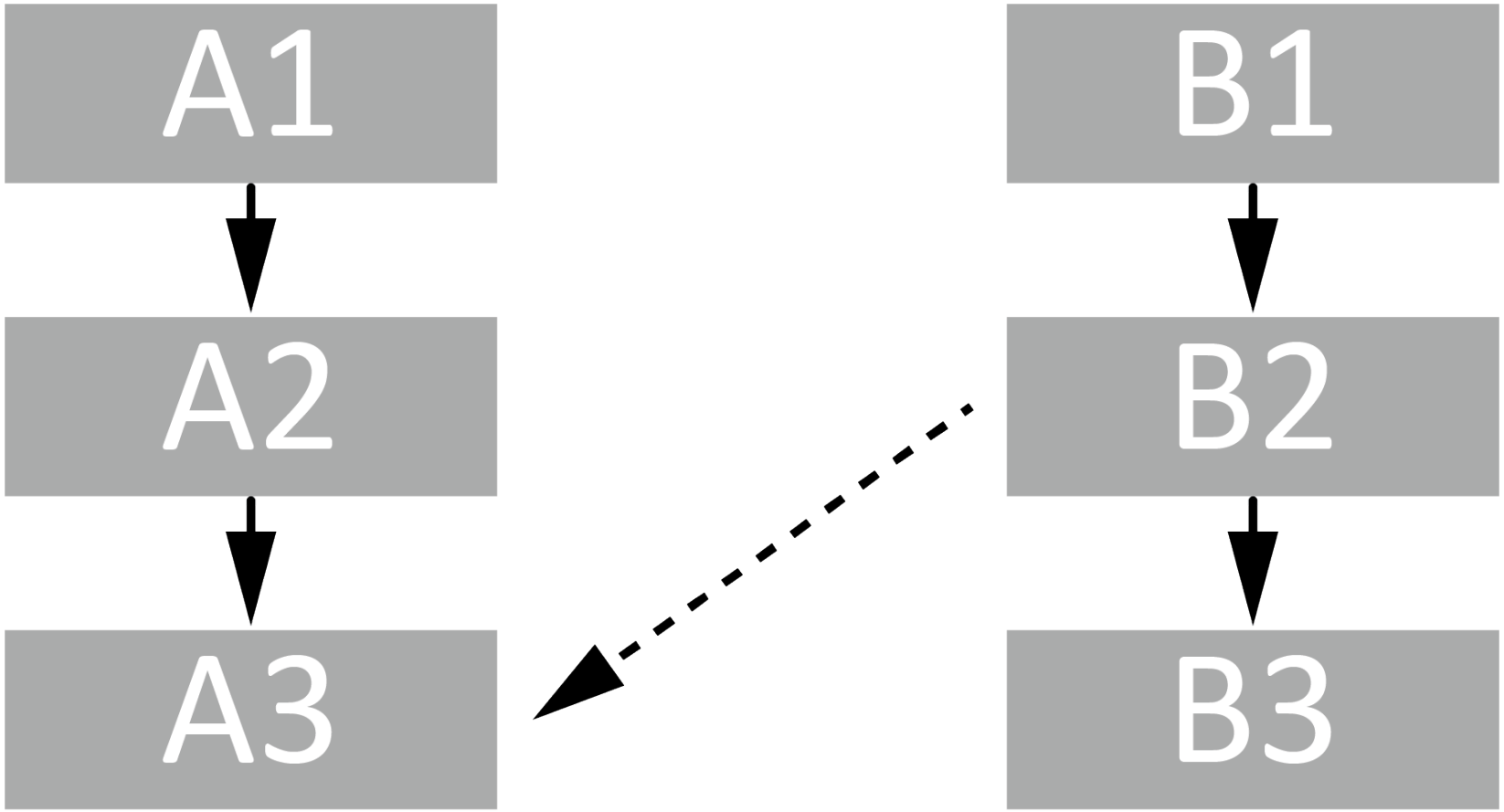
# Разумные требования: IGFS



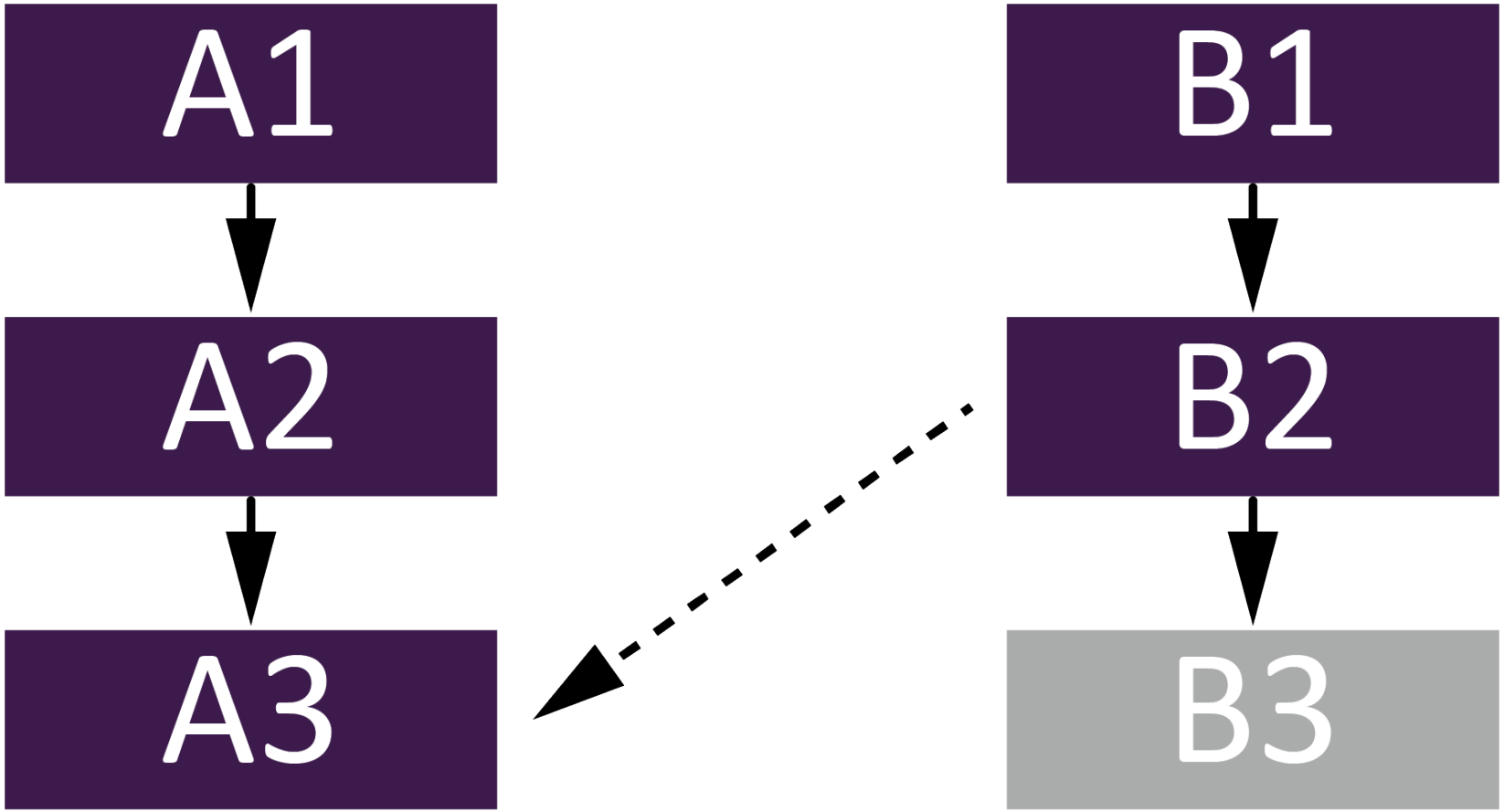
# Разумные требования: IGFS



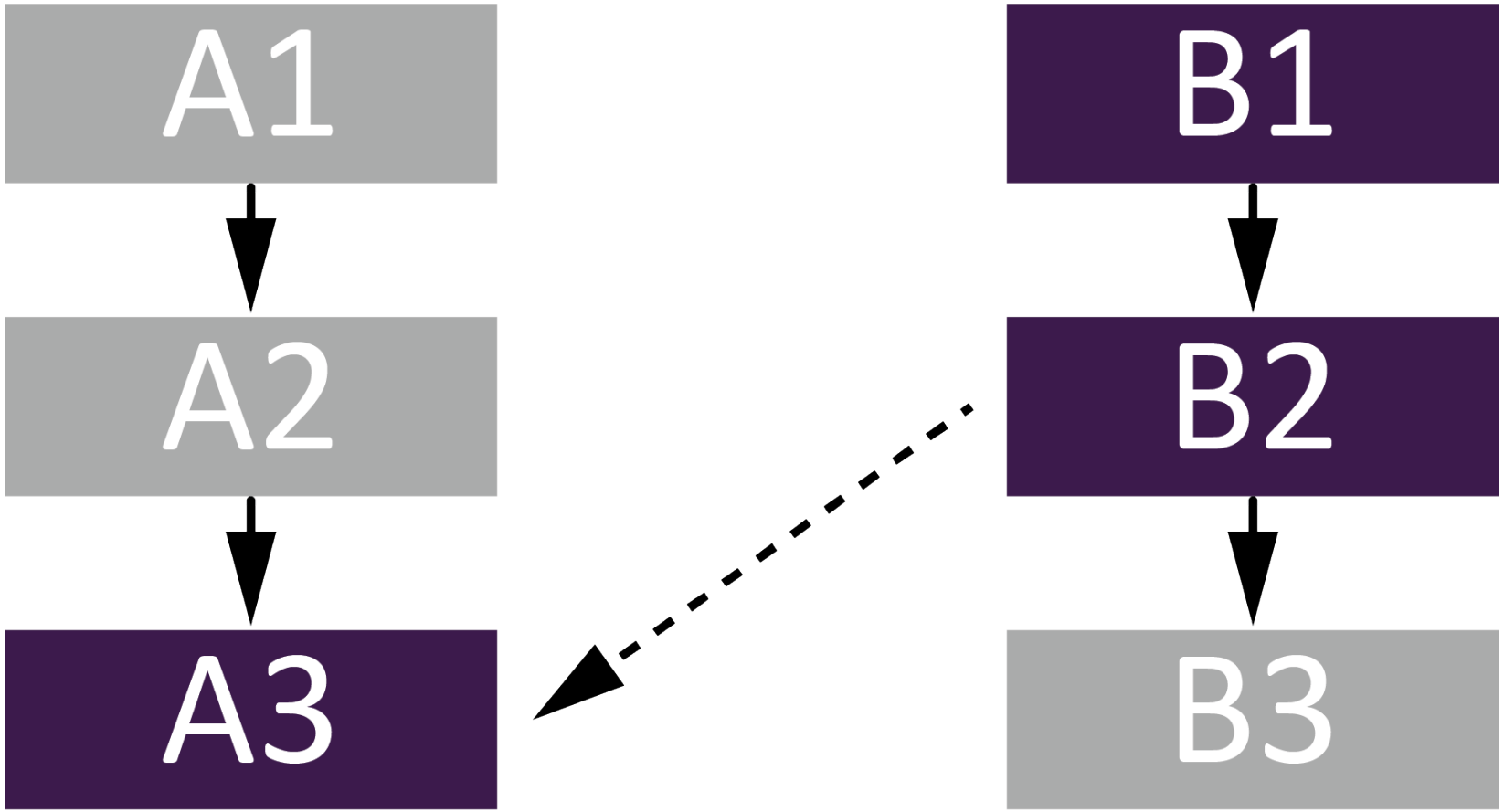
# Разумные требования: IGFS



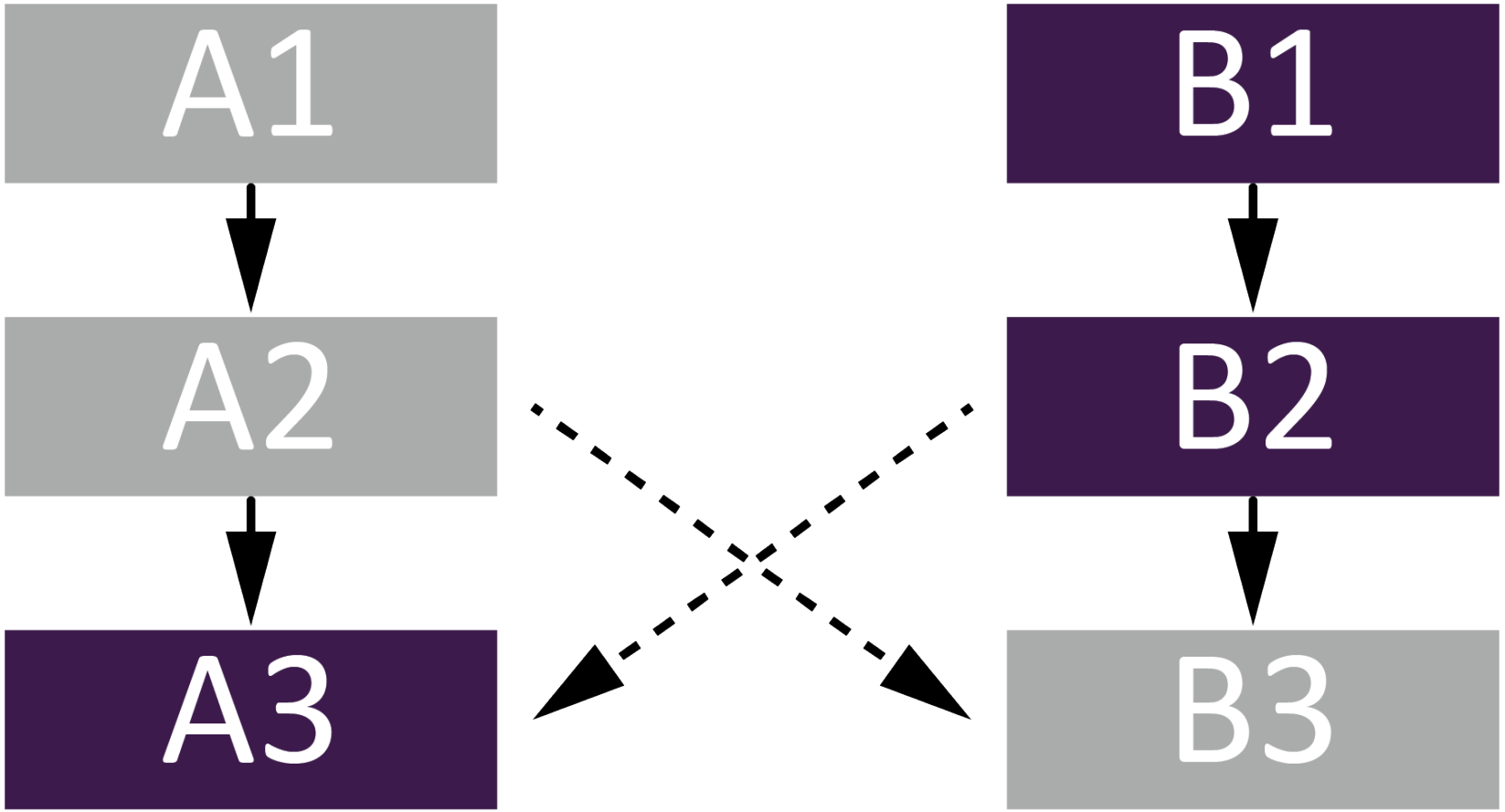
# Разумные требования: IGFS



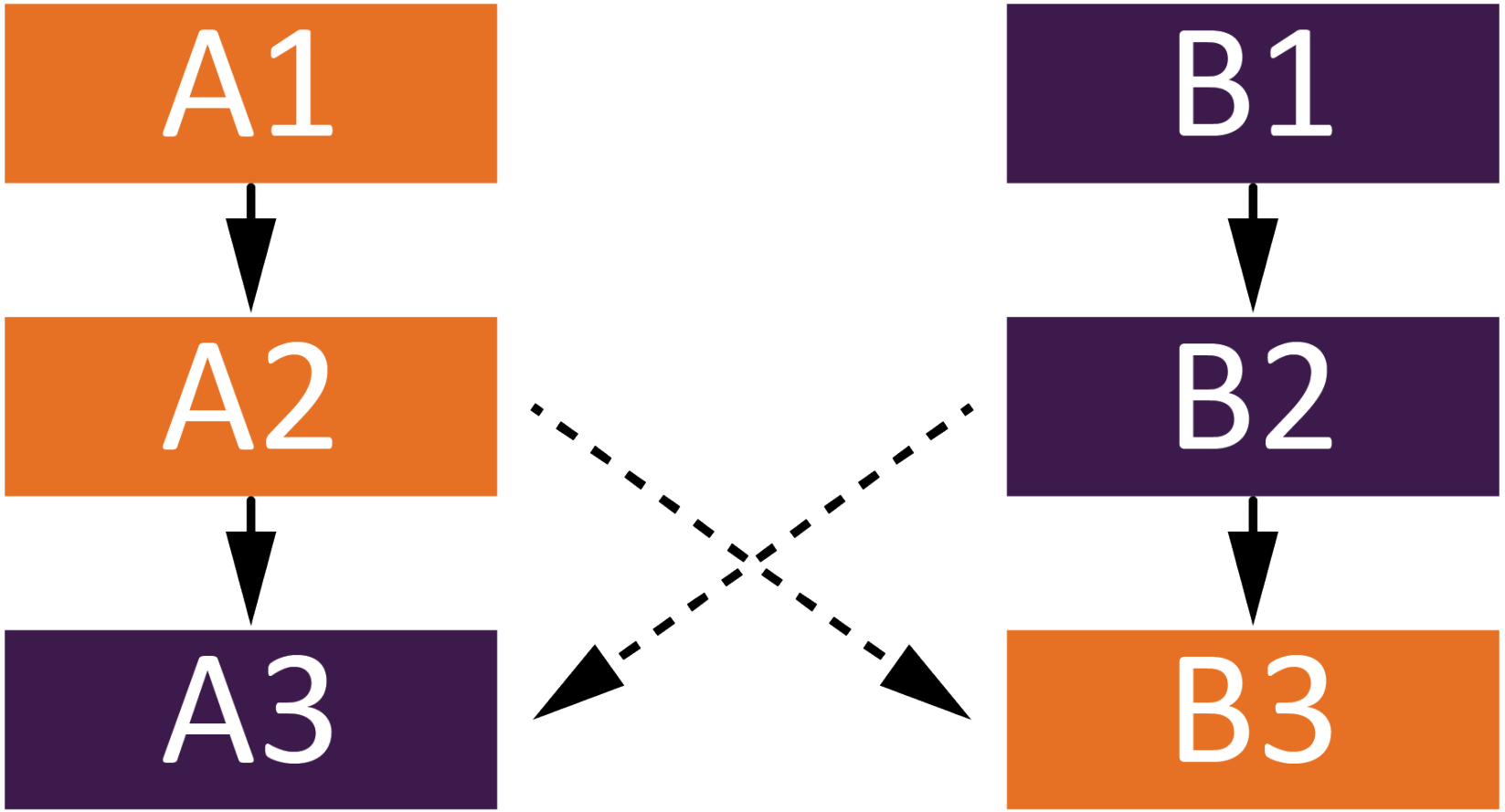
# Разумные требования: IGFS



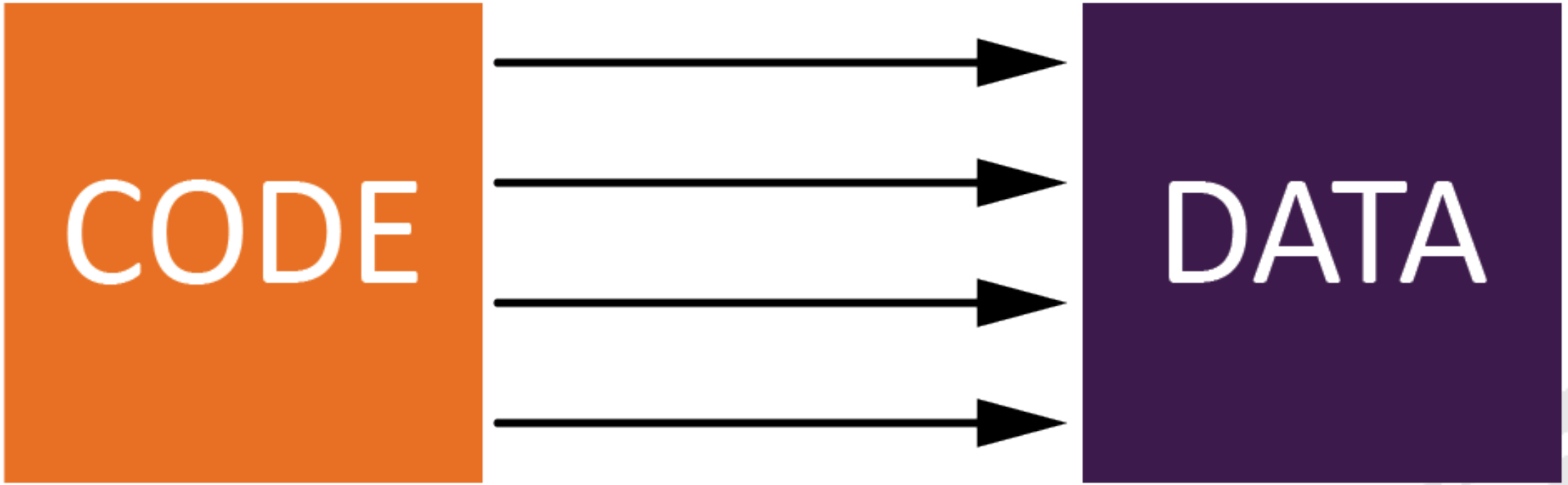
# Разумные требования: IGFS



# Разумные требования: IGFS



# Данные к коду



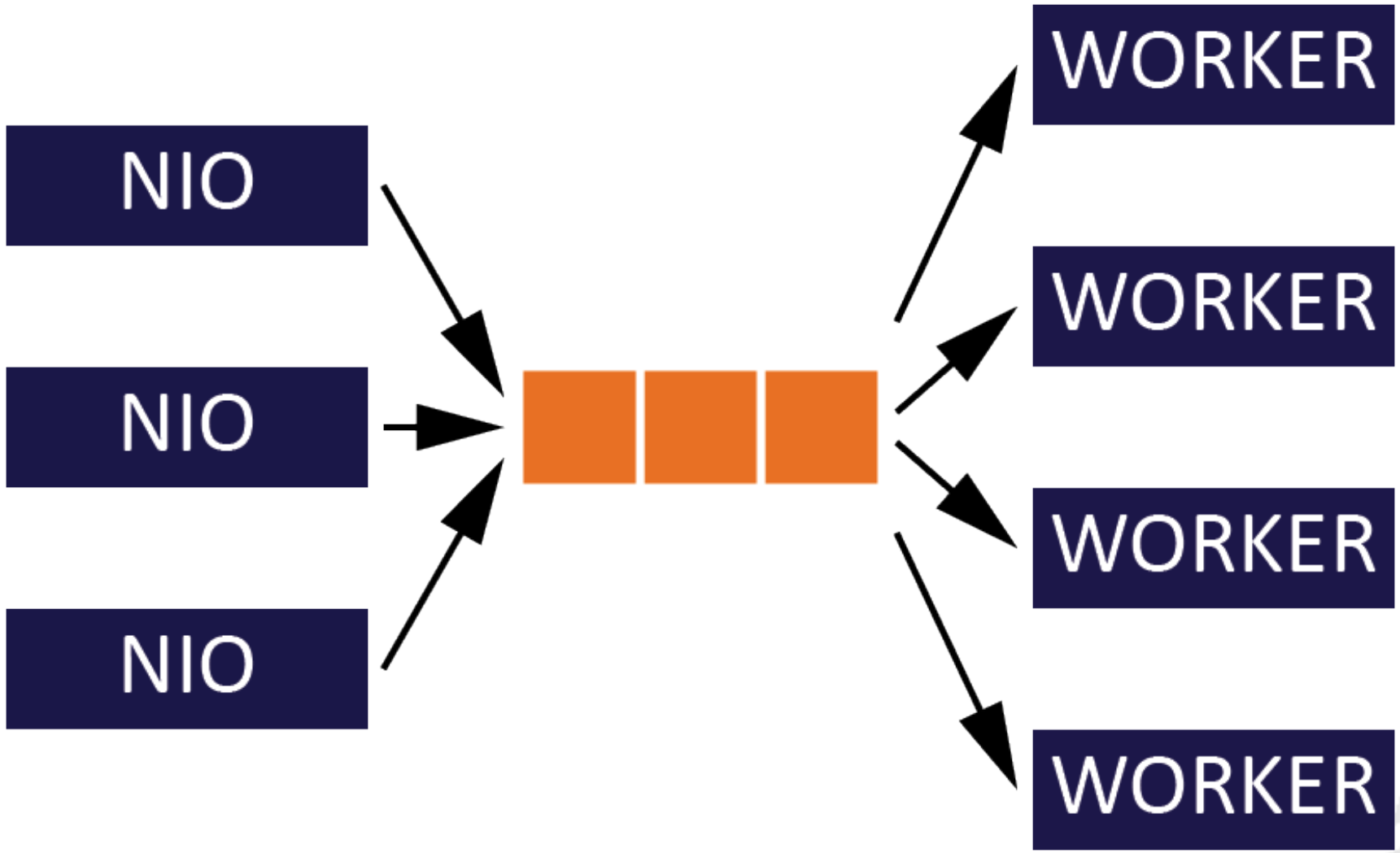
# Код к данным

# Код к данным: а если баг?

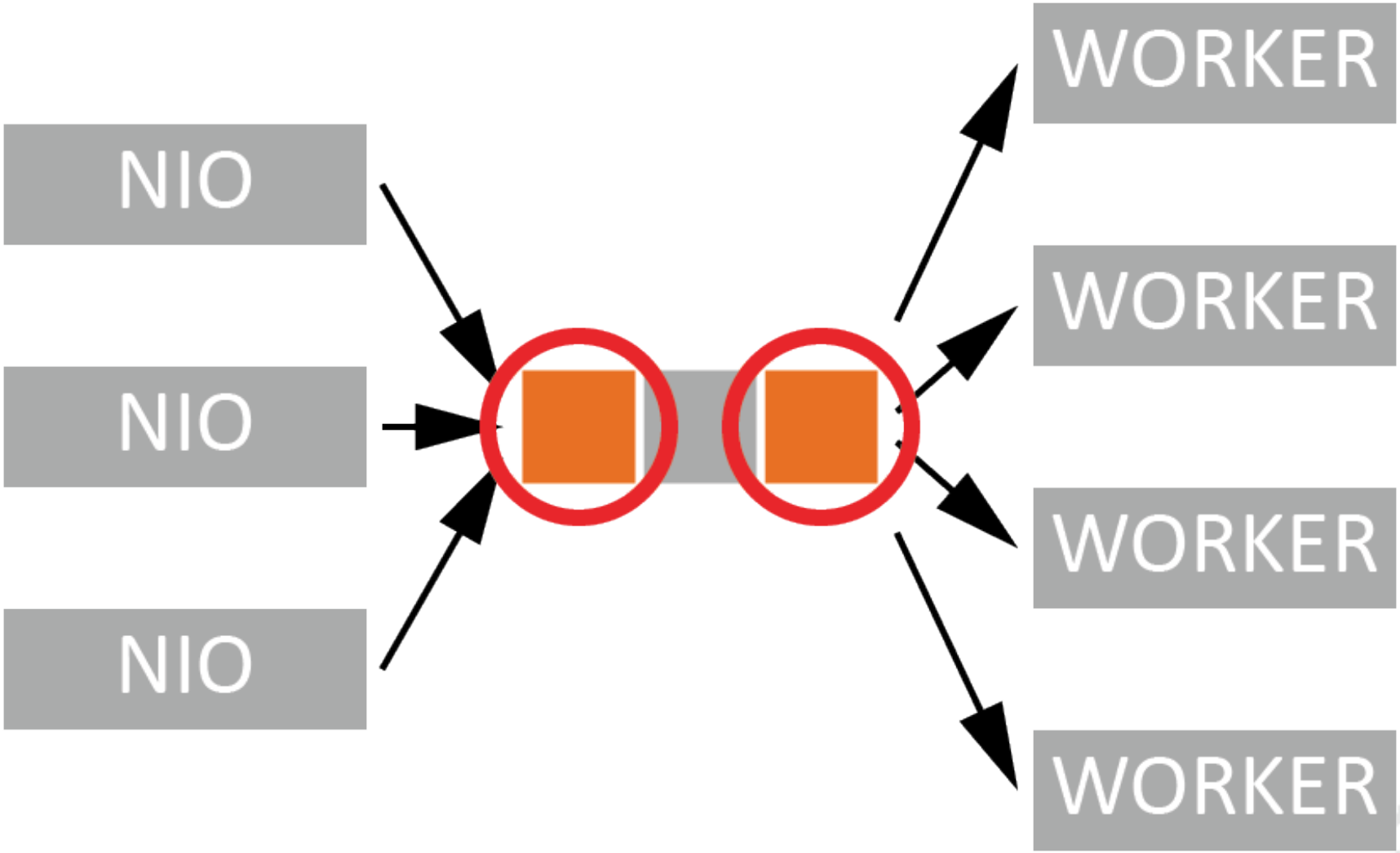
# План

- Алгоритмы шардирования
- Data co-location
- Синхронизация в кластере
- Локальная архитектура многопоточности

# Локальное распределение задач

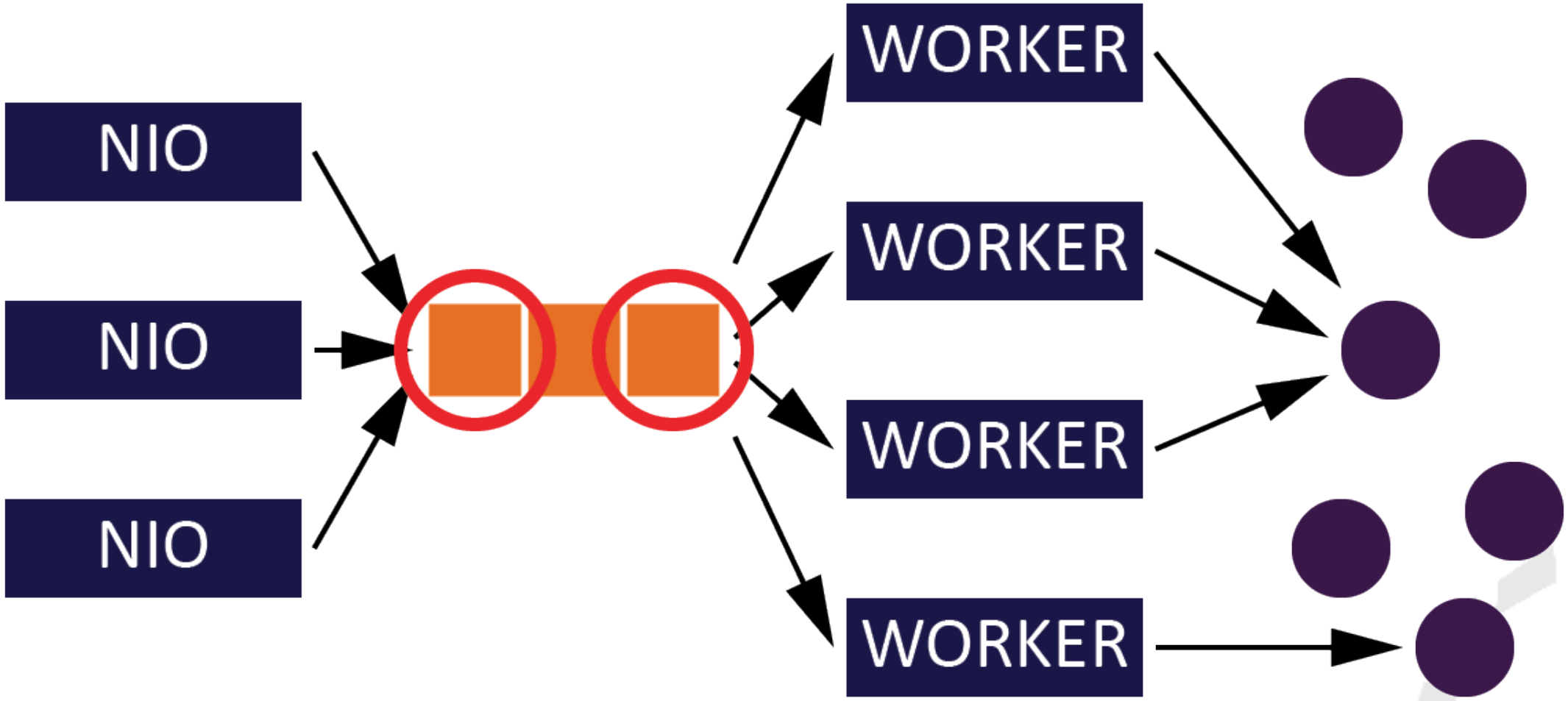


# Локальное распределение задач

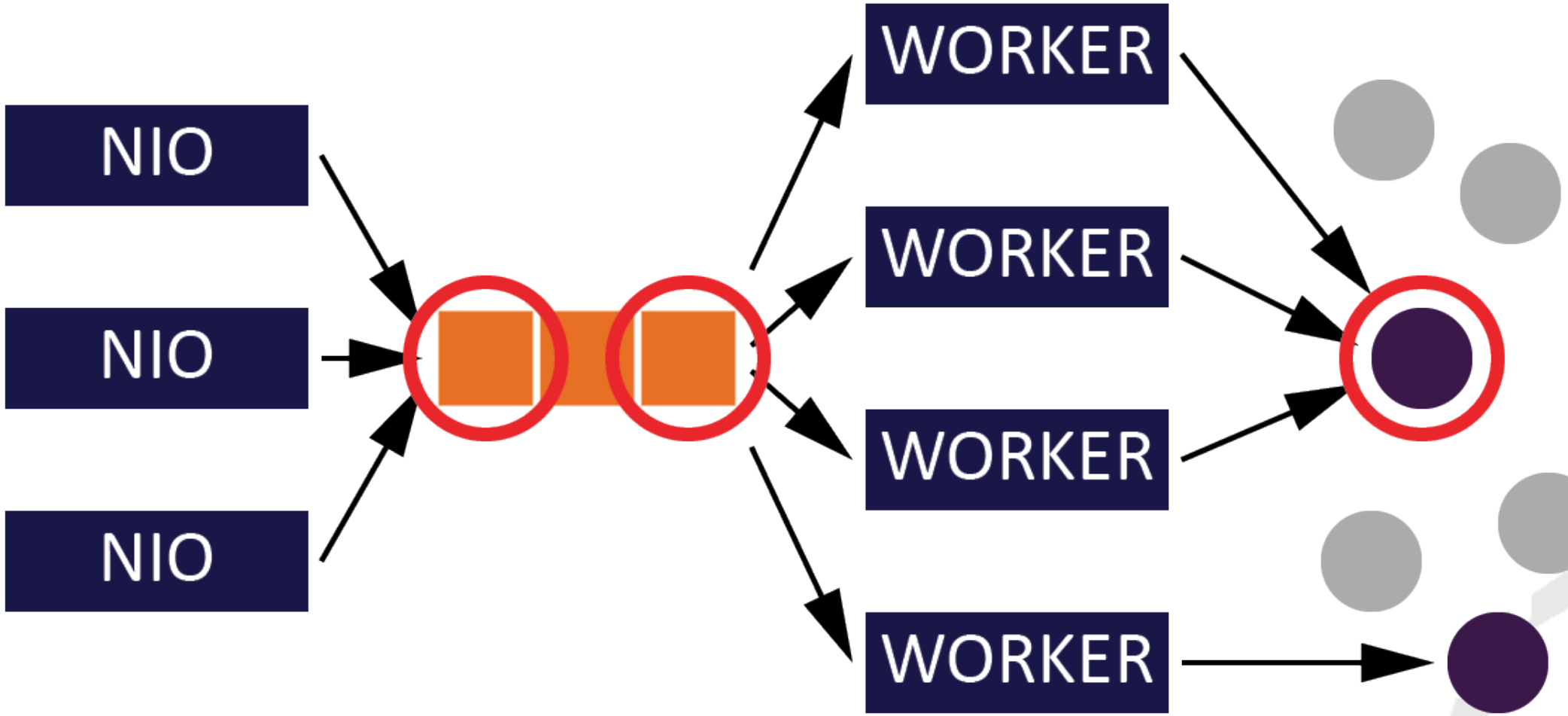


# Локальное распределение задач

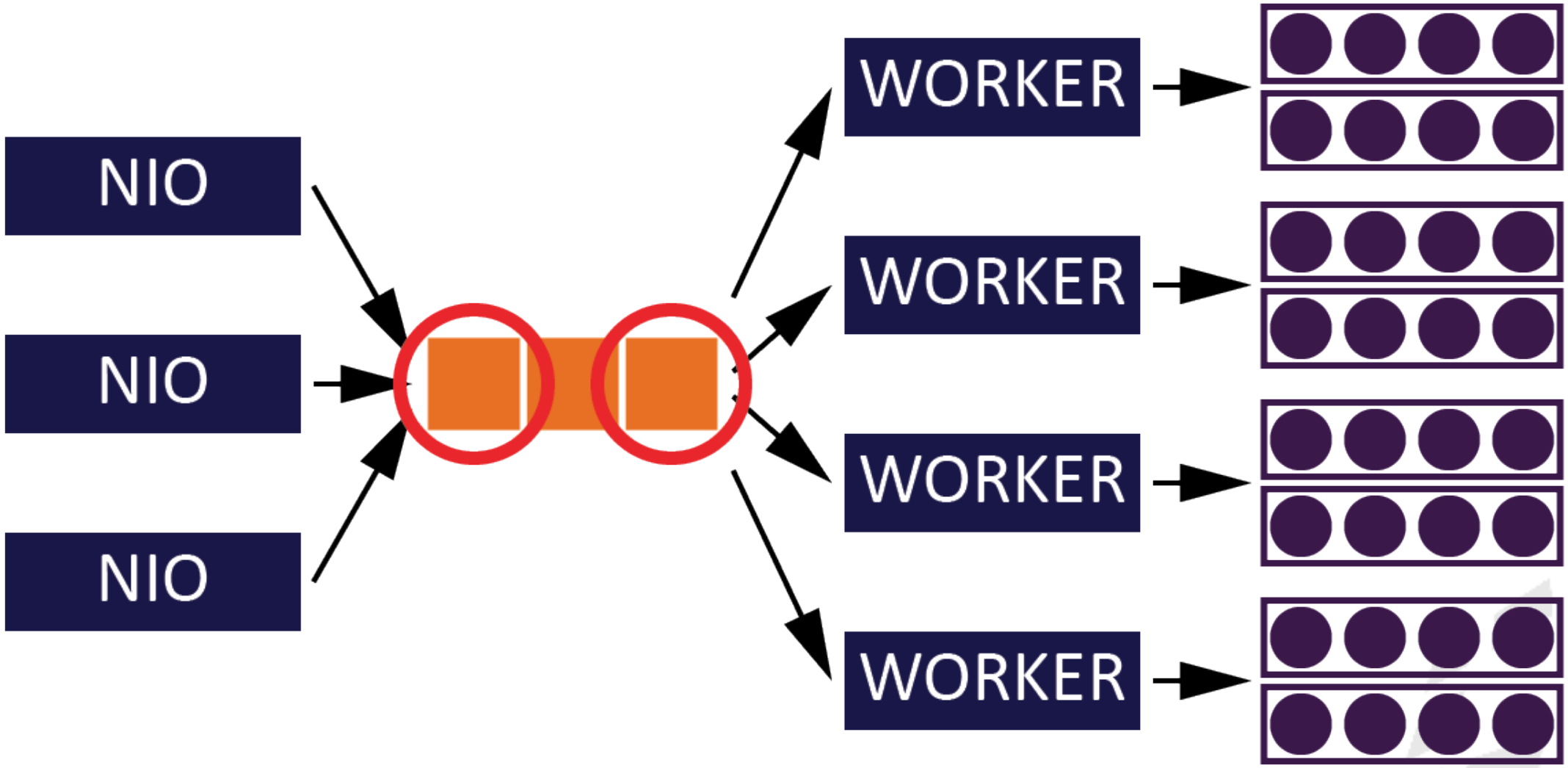
# Локальное распределение задач



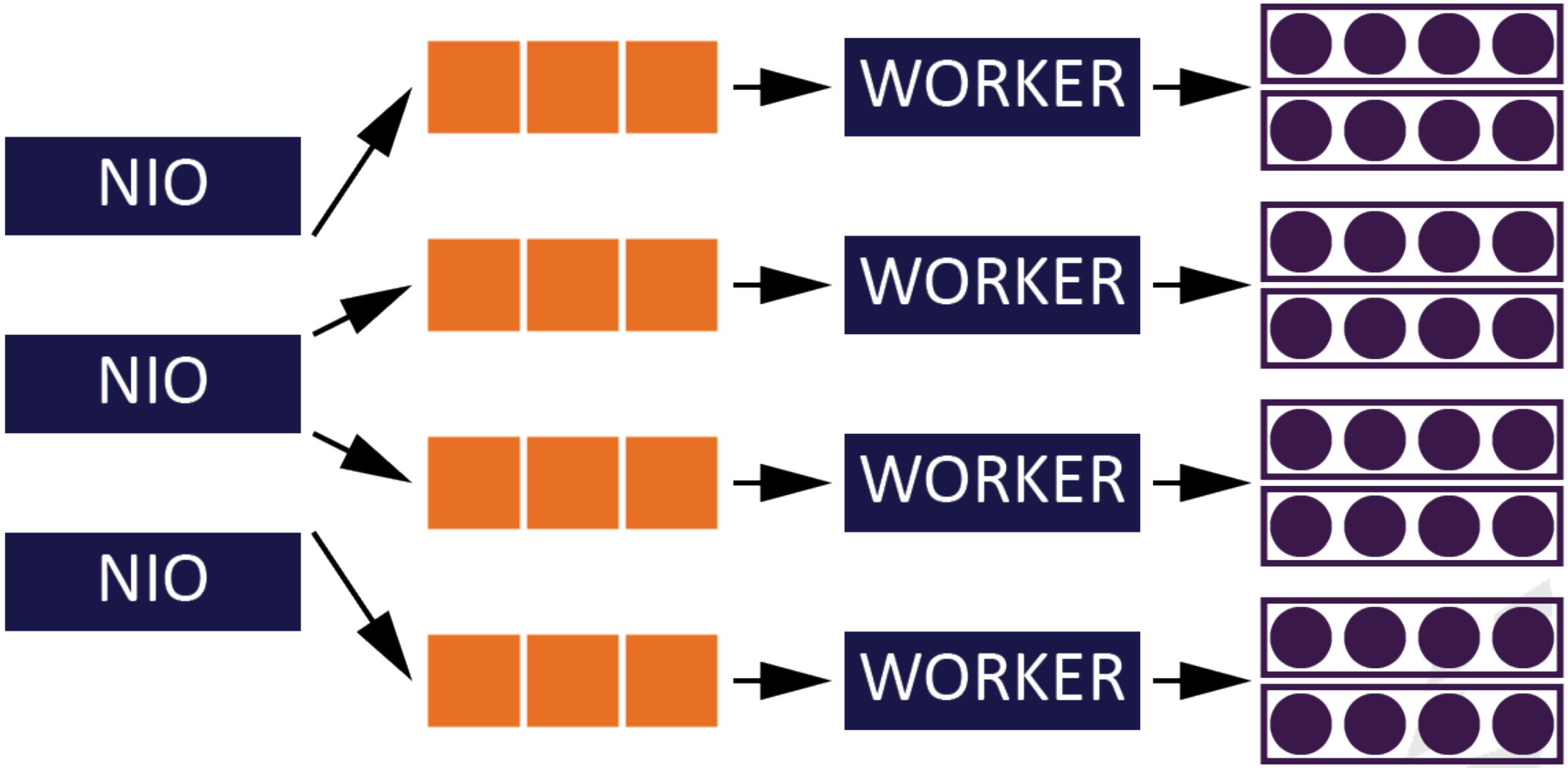
# Локальное распределение задач



# Thread-per-partition



# Thread-per-partition



# Thread-per-partition


Node 1

100 TPS

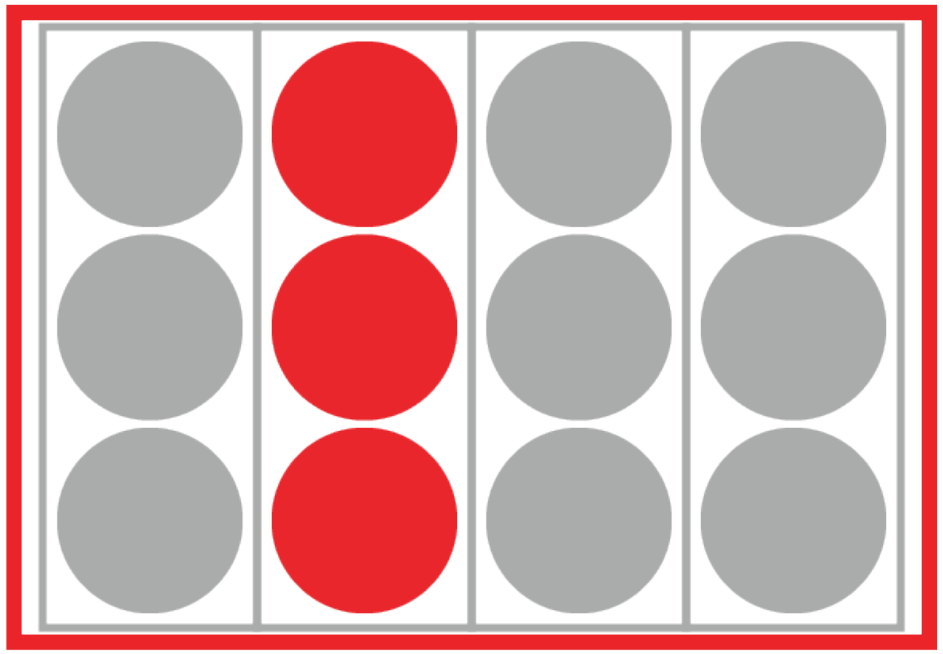

Node 2

50 TPS


Node 3

10 TPS

# Thread-per-partition



Node 1

100 TPS

SPB

Node 2

50 TPS

NSK

Node 3

10 TPS

# Итого

- Шардинг: важен не только баланс, но и стабильность

# Итого

- Шардинг: важен не только баланс, но и стабильность
- Co-location: баланс или эффективность

# Итого

- Шардинг: важен не только баланс, но и стабильность
- Co-location: баланс или эффективность
- Модель данных должна быть адаптирована

# Итого

- Шардинг: важен не только баланс, но и стабильность
- Co-location: баланс или эффективность
- Модель данных должна быть адаптирована
- Координация – враг производительности

# Итого

- Шардинг: важен не только баланс, но и стабильность
- Co-location: баланс или эффективность
- Модель данных должна быть адаптирована
- Координация – враг производительности
- Thread-per-partition – ускоряет простые сценарии, может замедлить сложные

# Вопросы?