

|| РСХБ | ЦИФРА

# ВИРТУАЛЬНЫЕ ПОТОКИ В JAVA 21+



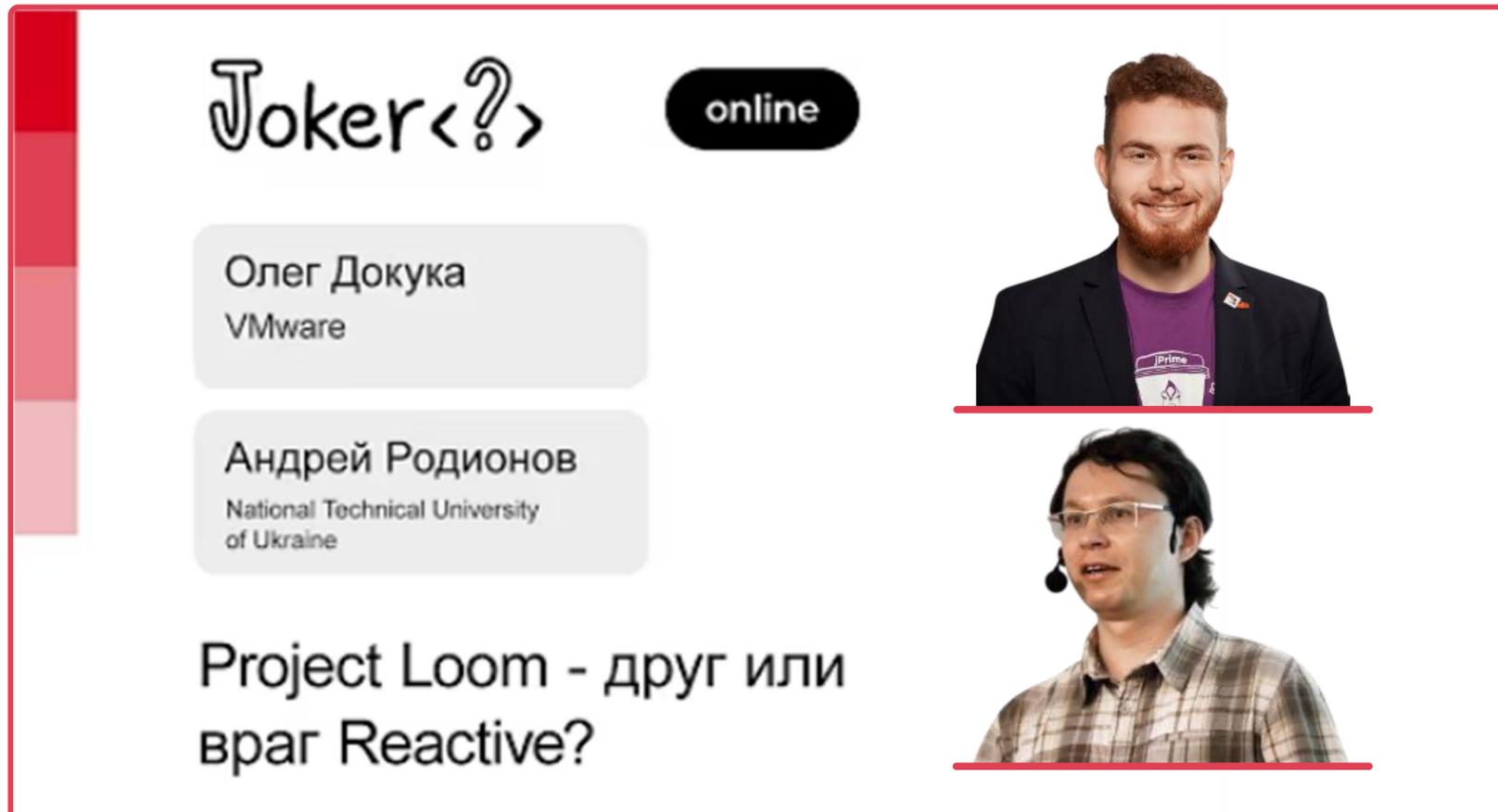
# НЕМНОГО ОБО МНЕ

- Последние 5 лет пишу код на Java
- Использовал в проде все последние LTS версии языка Java: 8, 11, 17, 21
- Моя команда занимается в основном интеграциями
- Последние несколько лет разрабатываю микросервисы



# ВИРТУАЛЬНЫЕ ПОТОКИ VS REACTIVE PROGRAMMING

Олег Докука, Андрей Родионов – Project Loom.  
Друг или враг Reactive?



The screenshot shows a virtual meeting interface. At the top left, the text 'Joker<?>' is displayed in a monospace font, with a black 'online' status indicator to its right. Below this, there are two participant cards. The first card identifies 'Олег Докука' from 'VMware'. The second card identifies 'Андрей Родионов' from 'National Technical University of Ukraine'. To the right of these cards are two video thumbnails. The top thumbnail shows a man with a beard and a purple t-shirt. The bottom thumbnail shows a man with glasses and a plaid shirt. At the bottom of the interface, the text 'Project Loom - друг или враг Reactive?' is visible.

# ВИРТУАЛЬНЫЕ ПОТОКИ **VS** GO-РУТИНЫ И КО-РУТИНЫ

Иван Углянский — Thread Wars:  
проект **Loom** наносит ответный удар



Thread Wars: проект  
Loom наносит  
ответный удар



Иван  
Углянский

Huawei

# ВИРТУАЛЬНЫЕ ПОТОКИ

**Формула:**

**ForkJoinPool + Continuations**

**Зачем:**

**Highload + Simplicity**

**Опасности:**

**Synchronized + Native Code**  
**Java 21                      Java 25**

# СОЗДАНИЕ ПОТОКОВ



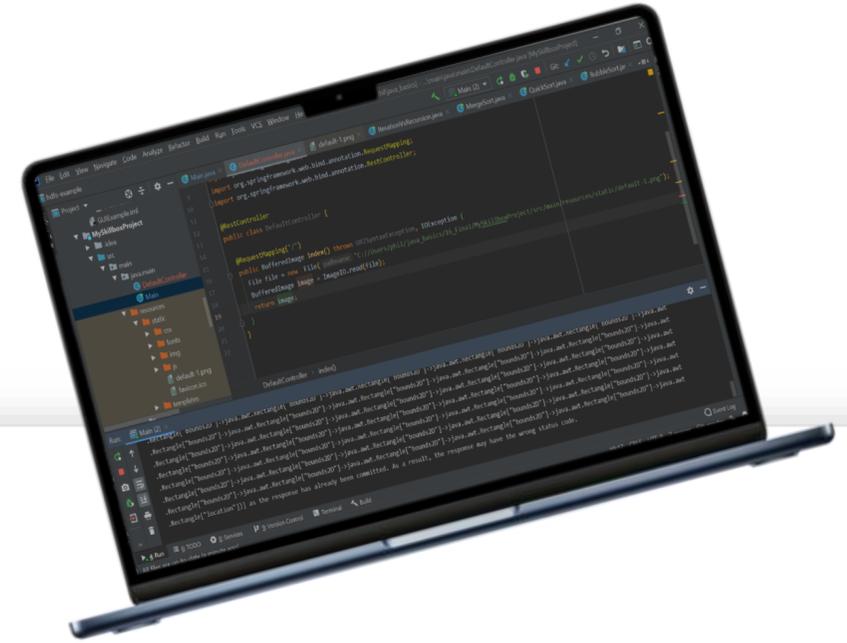
```
Runnable DO_SOMETHING = () -> {...};
```

## Платформенные потоки Через конструктор

```
var thread = new Thread(DO_SOMETHING);  
thread.start();
```

## Виртуальные потоки Через *static factory* метод

```
var thread = Thread.startVirtualThread(DO_SOMETHING);
```



# СОЗДАНИЕ ПОТОКОВ

```
Runnable DO_SOMETHING = () -> {...};
```

## Платформенные потоки Через конструктор

```
var thread = new Thread(DO_SOMETHING);  
thread.start();
```

## Через builder

```
var thread = Thread.ofPlatform().unstarted(DO_SOMETHING)
```

```
var thread = Thread.ofPlatform().start(DO_SOMETHING)
```

## Виртуальные потоки Через static factory метод

```
var thread = Thread.startVirtualThread(DO_SOMETHING);
```

## Через builder

```
var thread = Thread.ofVirtual().start(DO_SOMETHING)
```

```
var thread = Thread.ofVirtual().unstarted(DO_SOMETHING)
```





```
Runnable DO_SOMETHING = () -> {...};
```

## Платформенные потоки

### Через конструктор

```
var thread = new Thread(DO_SOMETHING);  
thread.start();
```

### Через builder

```
var thread = Thread.ofPlatform().unstarted(DO_SOMETHING)
```

```
var thread = Thread.ofPlatform().start(DO_SOMETHING)
```

### Через фабрику

```
var factory = Thread.ofPlatform().factory();  
var thread = factory.newThread(DO_SOMETHING);  
thread.start();
```

## Виртуальные потоки

### Через static factory метод

```
var thread = Thread.startVirtualThread(DO_SOMETHING);
```

### Через builder

```
var thread = Thread.ofVirtual().start(DO_SOMETHING)
```

```
var thread = Thread.ofVirtual().unstarted(DO_SOMETHING)
```

### Через фабрику

```
var factory = Thread.ofVirtual().factory();  
var thread = factory.newThread(DO_SOMETHING);  
thread.start();
```

# СОЗДАНИЕ ПОТОКОВ

```
Runnable DO_SOMETHING = () -> {...};
```

## Платформенные потоки Через конструктор

```
var thread = new Thread(DO_SOMETHING);  
thread.start();
```

## Через ExecutorService

```
try (var executor =  
Executors.newFixedThreadPool(CORES)) {  
    executor.submit(DO_SOMETHING);  
}
```

## Виртуальные потоки Через static factory метод

```
var thread = Thread.startVirtualThread(DO_SOMETHING);
```

## Через ExecutorService

```
try (var executor =  
Executors.newVirtualThreadPerTaskExecutor()) {  
    executor.submit(DO_SOMETHING);  
}
```



JVM

OS

Core

Core

CPU

Core

Core

**Thread  
Object**

**JVM**

**OS**

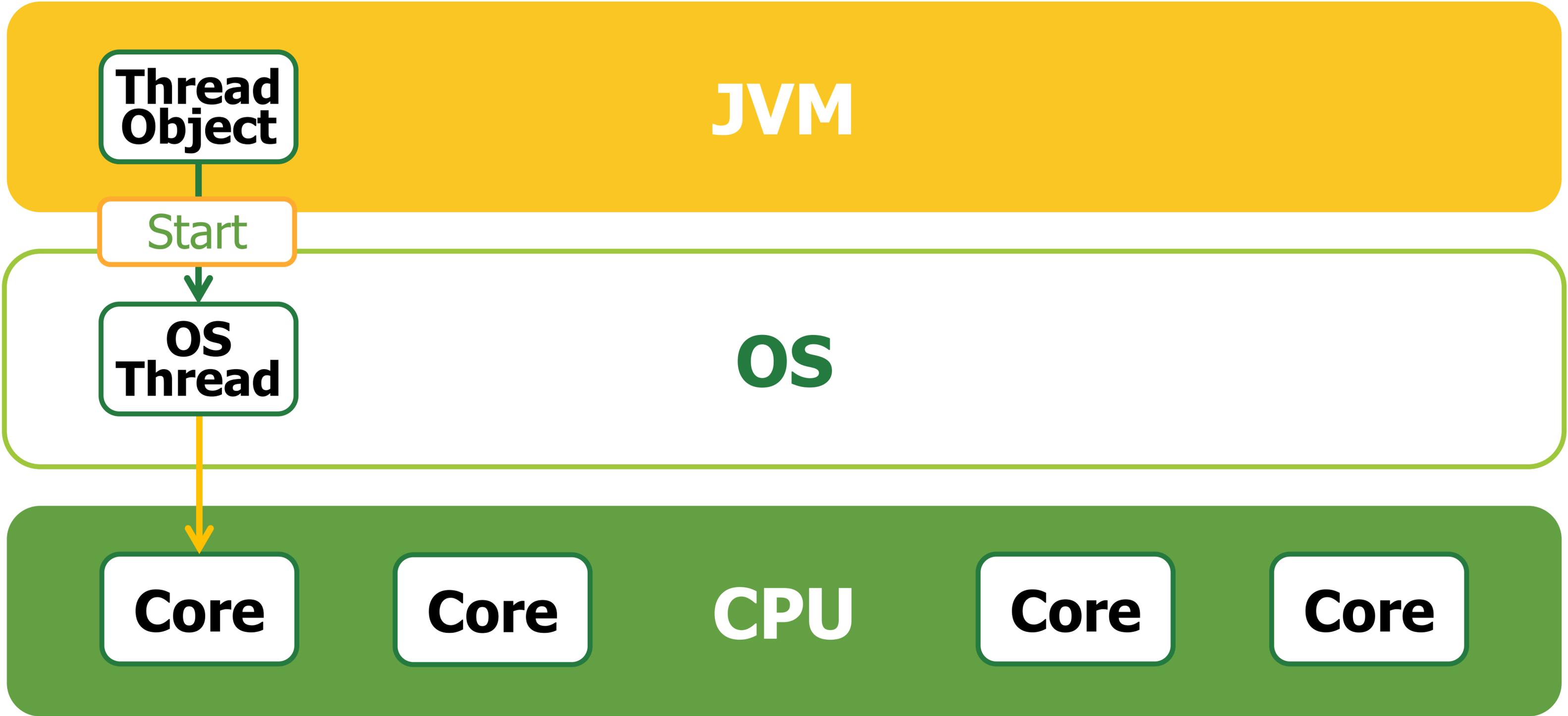
**Core**

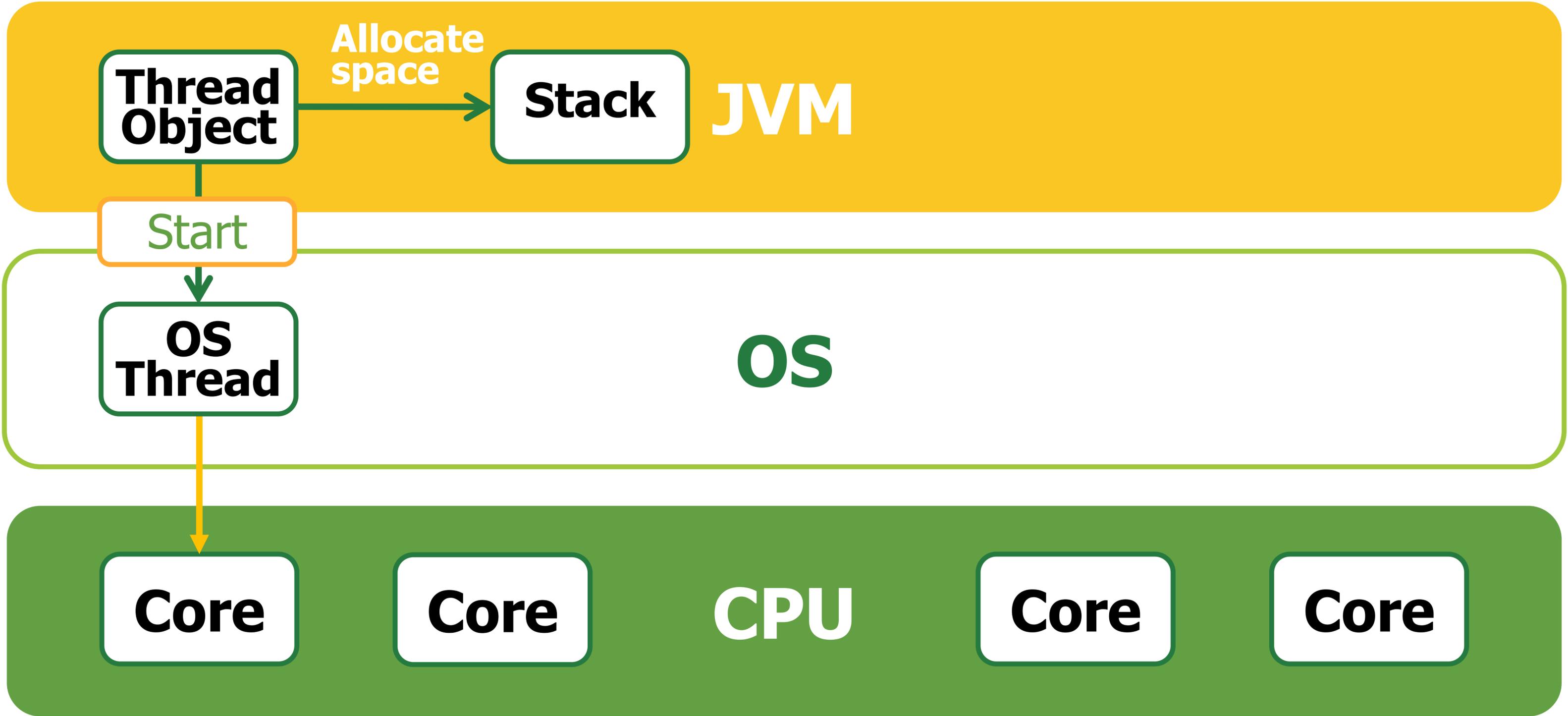
**Core**

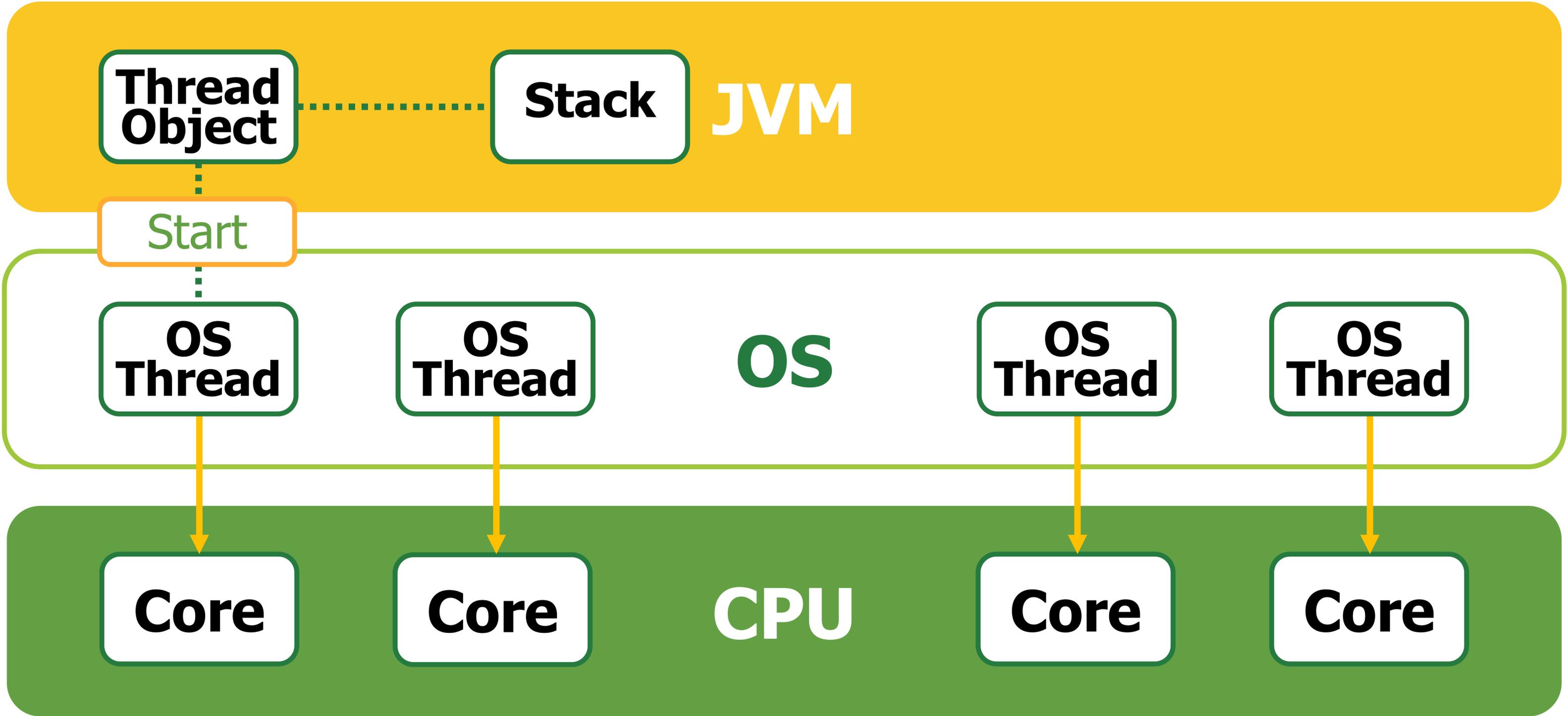
**CPU**

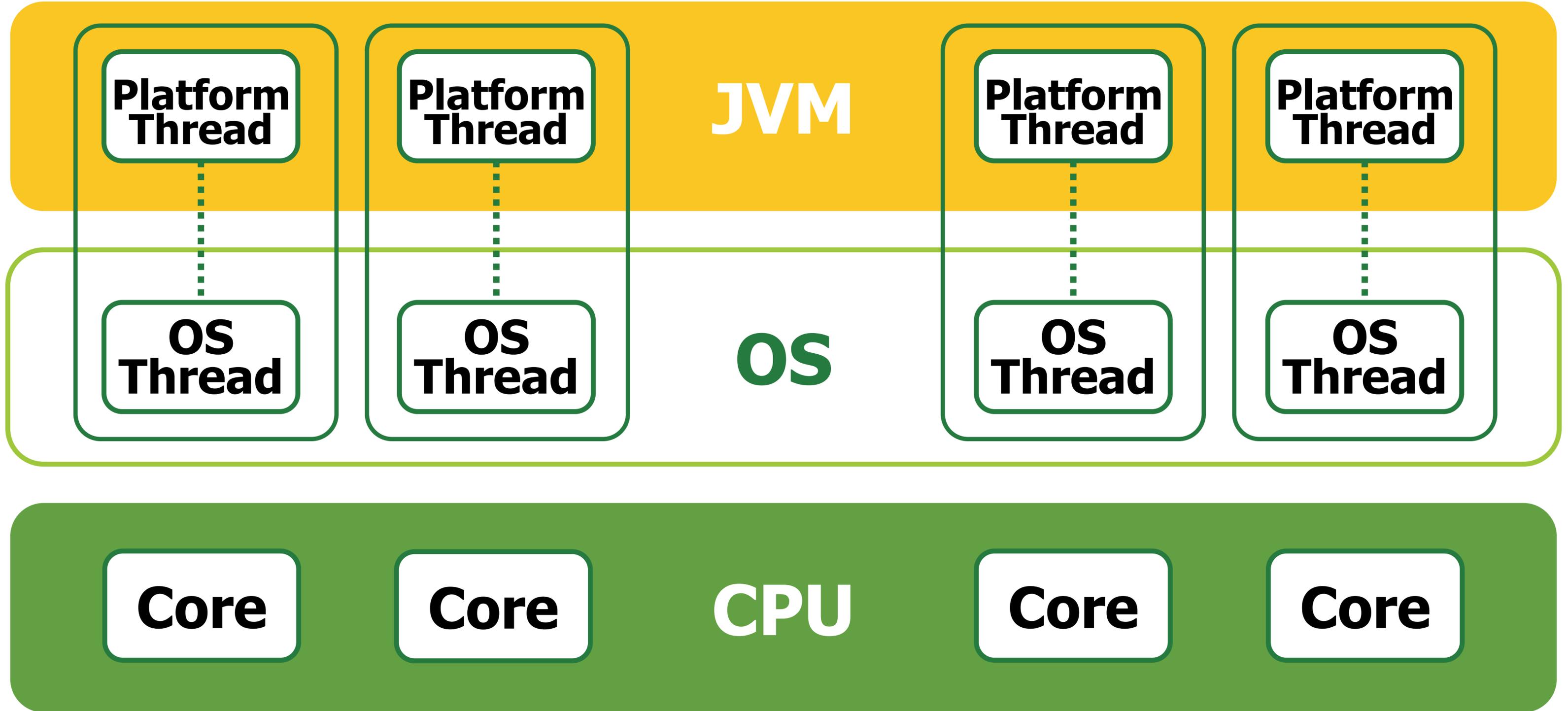
**Core**

**Core**









**Virtual  
thread**

**JVM**

**OS**

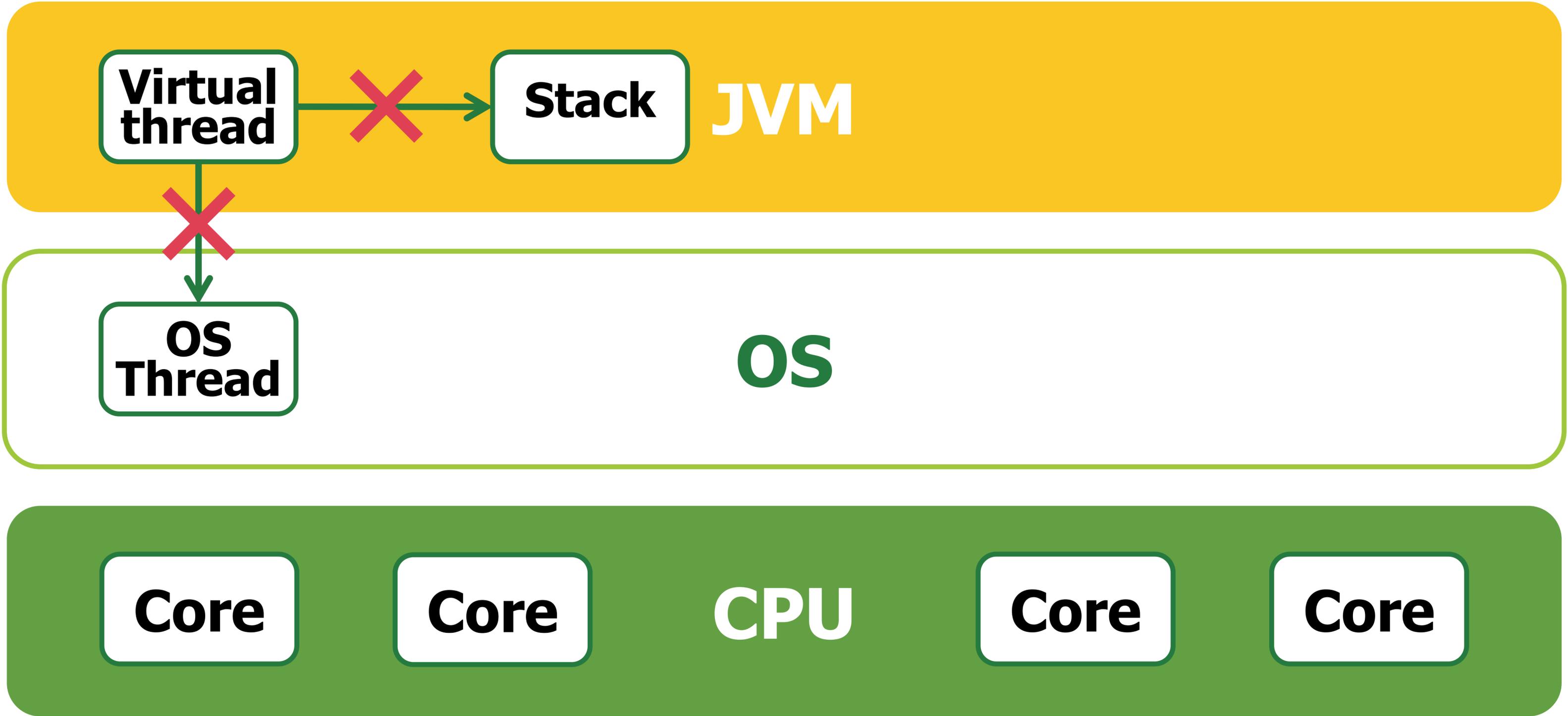
**Core**

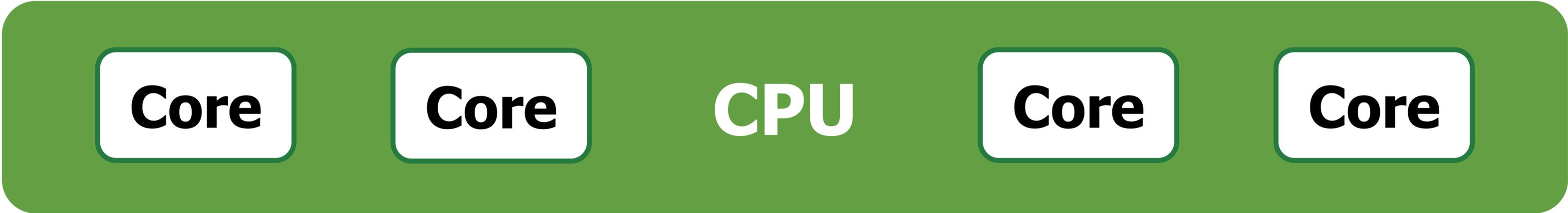
**Core**

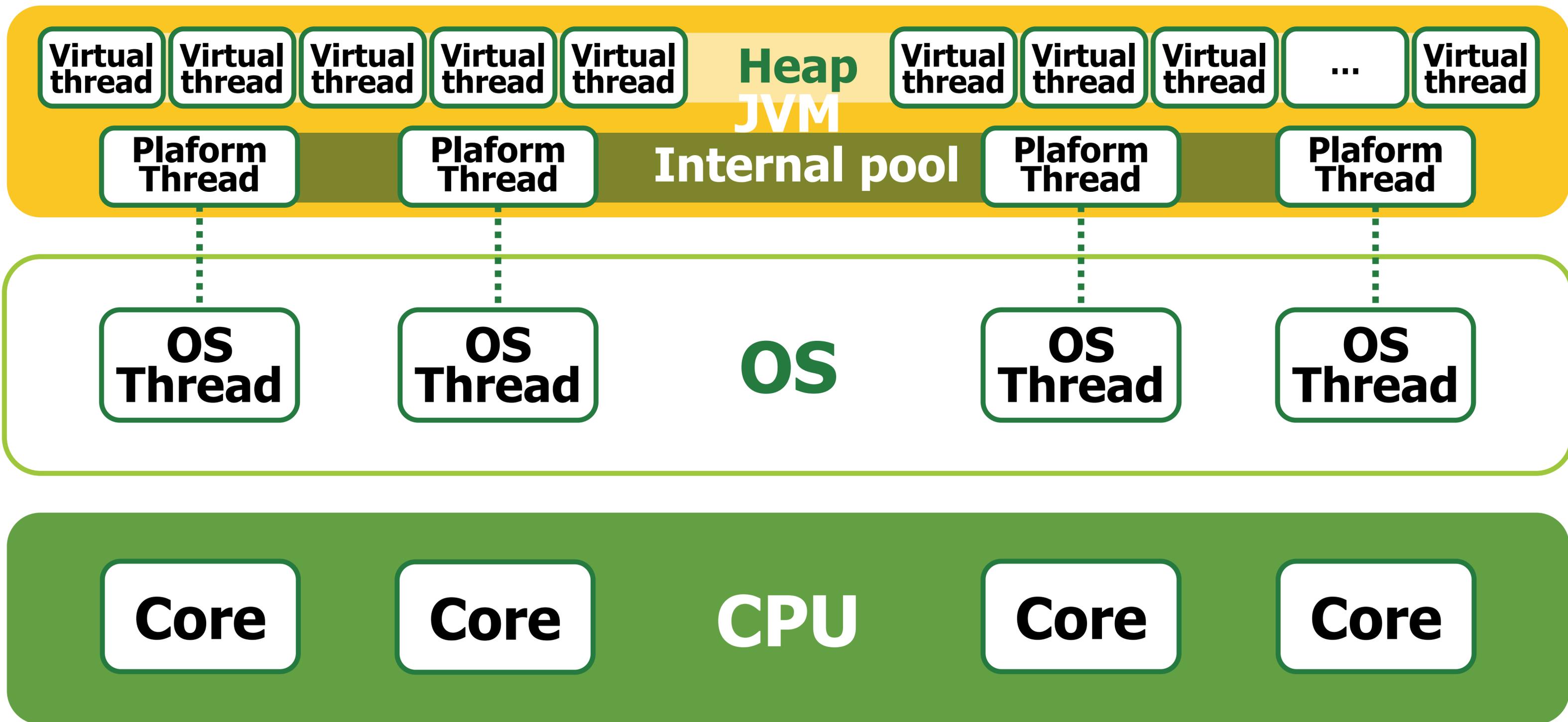
**CPU**

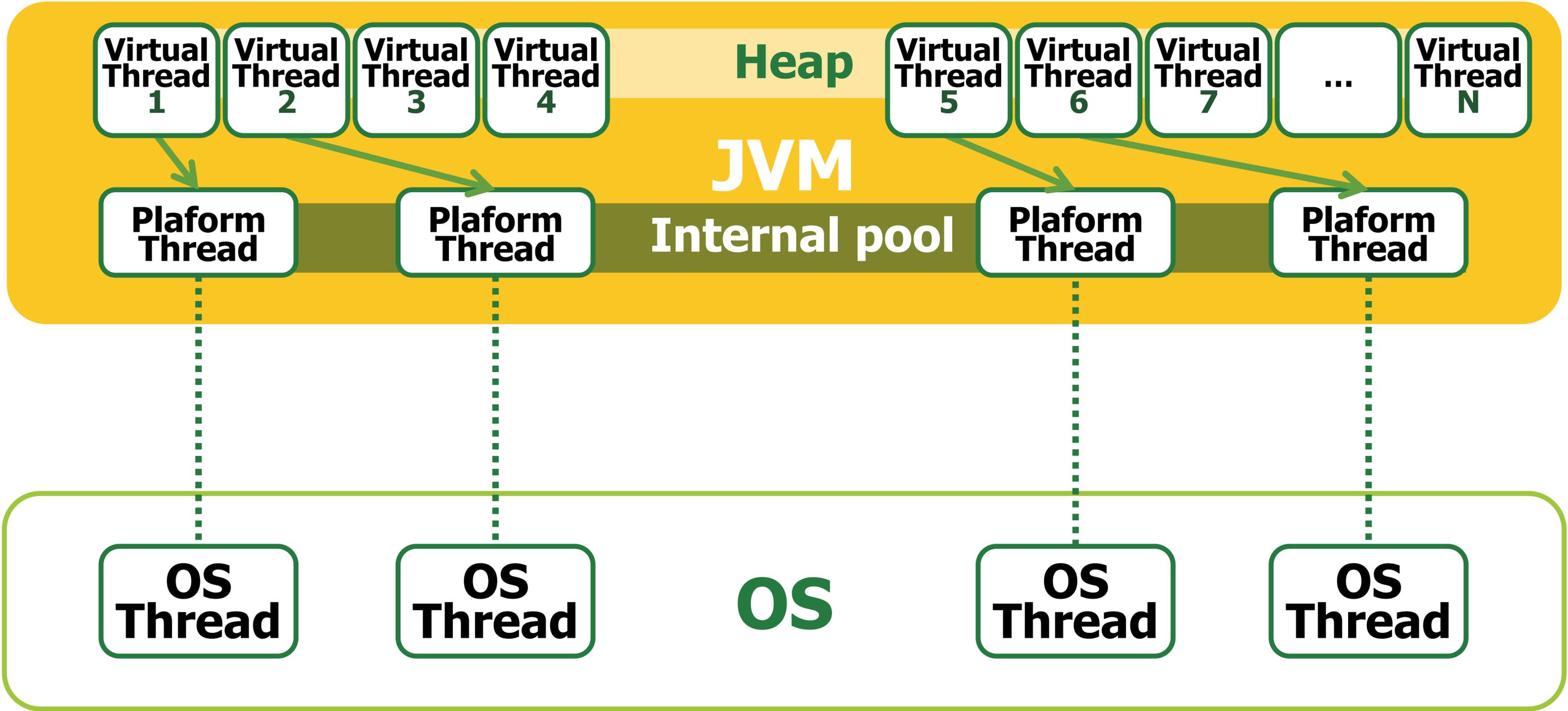
**Core**

**Core**











## ForkJoinPool





# ForkJoinPool





## ForkJoinPool





## ForkJoinPool





## ForkJoinPool





## ForkJoinPool





## ForkJoinPool





# ForkJoinPool





## ForkJoinPool





# ForkJoinPool





## ForkJoinPool





## ForkJoinPool





## ForkJoinPool





# ForkJoinPool

Completed





# ForkJoinPool





# ForkJoinPool





## ForkJoinPool





# ForkJoinPool





## ForkJoinPool



# ForkJoinPool



# ForkJoinPool

Blocking IO operation



# ForkJoinPool

Part unmount



# ForkJoinPool



v4

# ForkJoinPool



v4

Ждёт сигнала →

unparker

## ForkJoinPool



v4

# ForkJoinPool



v4

# ForkJoinPool



v4

# ForkJoinPool



v4

# ForkJoinPool

v10

p1

v7

v11

p2

v5

v12

p3

v6

v4

Получает сигнал →

unparker

## ForkJoinPool

v10

p1

v7

v11

p2

v5

v12

p3

v6

unparker

v4

# ForkJoinPool

unpark



# ForkJoinPool



# ForkJoinPool



# ForkJoinPool



Completed

# ForkJoinPool

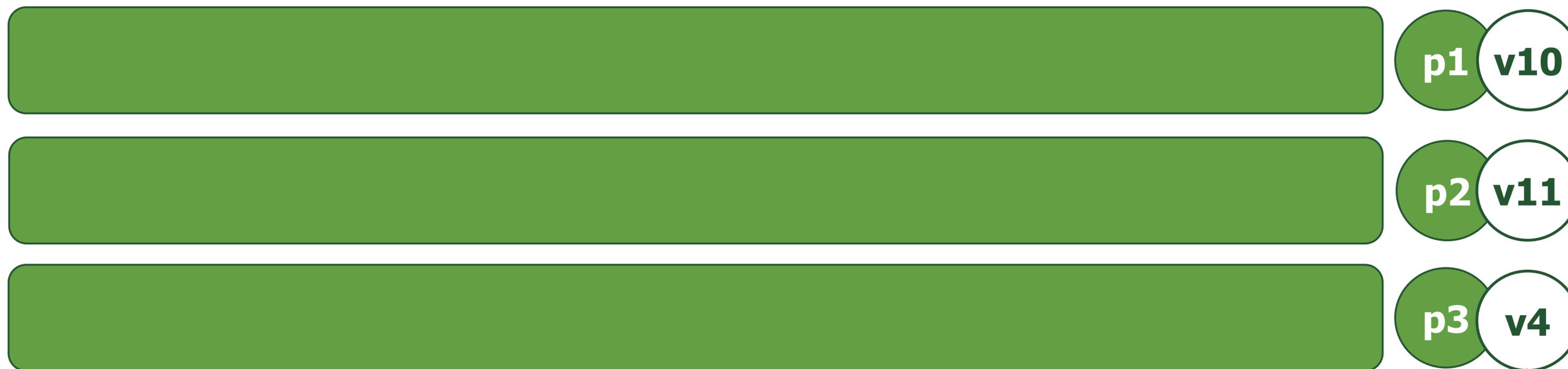


# ForkJoinPool



**Carrier thread changed!**

# ForkJoinPool



**Stack changed!**

# ForkJoinPool



**All task completed!**

# ВЕРНЁМСЯ НАЗАД. ЧТО МОГЛО ПОЙТИ НЕ ТАК?

## ForkJoinPool



# ВЕРНЁМСЯ НАЗАД. ЧТО МОГЛО ПОЙТИ НЕ ТАК?

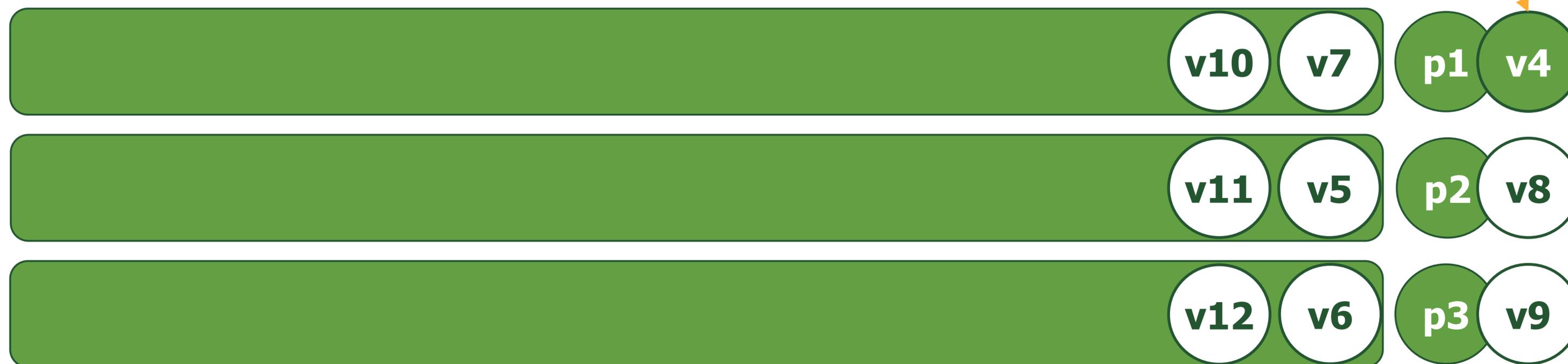
## ForkJoinPool SYNCHRONIZED BLOCK!



# ВЕРНЁМСЯ НАЗАД. ЧТО МОГЛО ПОЙТИ НЕ ТАК?

## ForkJoinPool

**PINNED!**



# ВЕРНЁМСЯ НАЗАД. ЧТО МОГЛО ПОЙТИ НЕ ТАК?

## ForkJoinPool BLOCKING IO OPERATION!



# ВЕРНЁМСЯ НАЗАД. ЧТО МОГЛО ПОЙТИ НЕ ТАК?

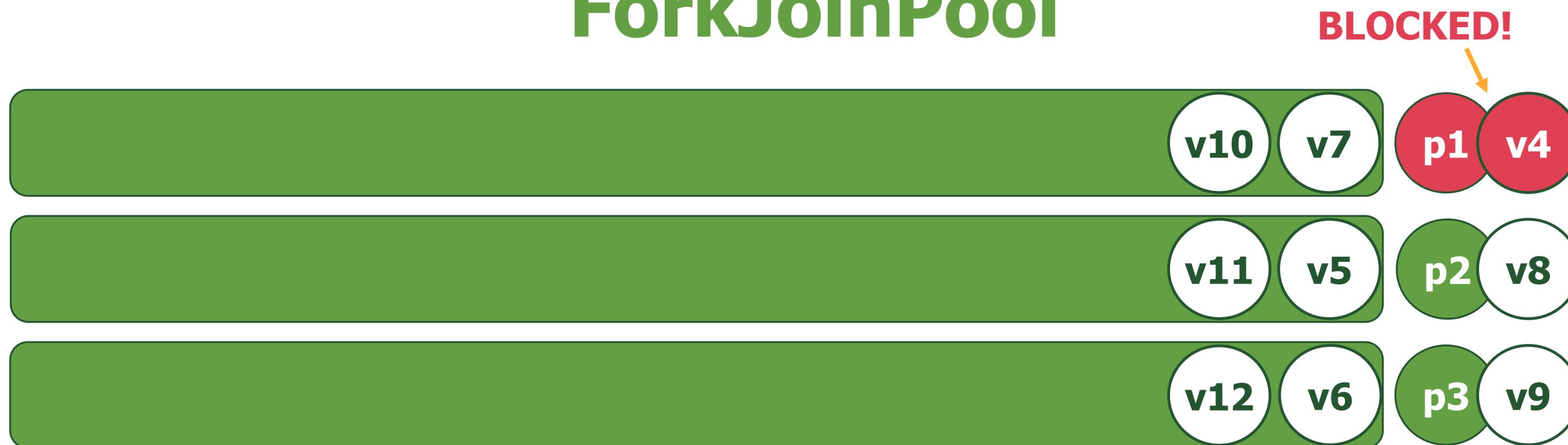
## ForkJoinPool

UNMOUNT IMPOSSIBLE!



# ВЕРНЁМСЯ НАЗАД. ЧТО МОГЛО ПОЙТИ НЕ ТАК?

## ForkJoinPool



$$\text{throughput} = n \times \frac{1000}{t}$$

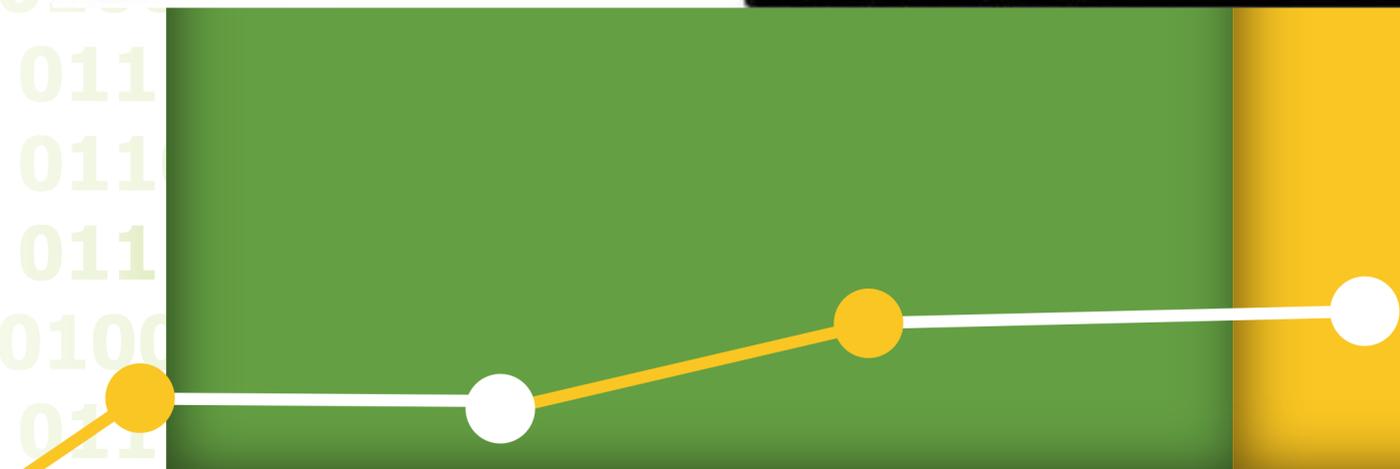
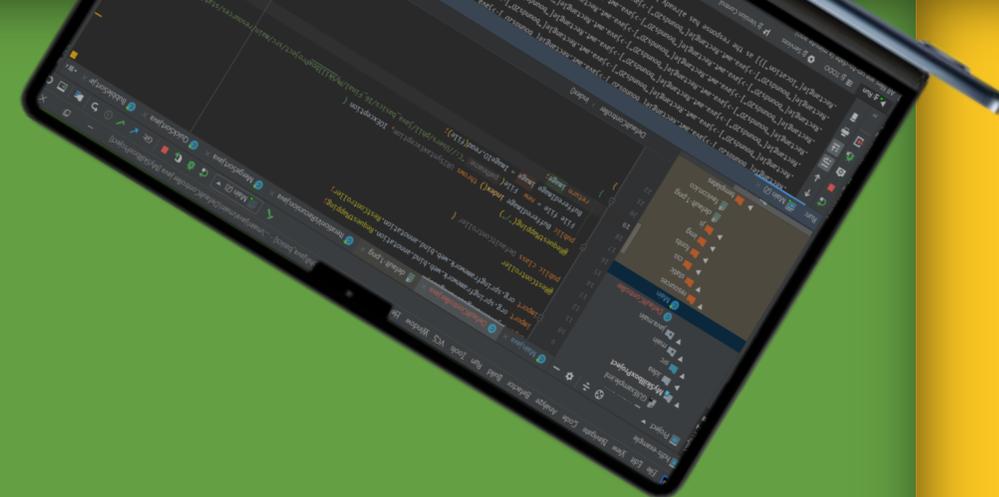
Запросов  
в секунду

Число  
ПОТОКОВ

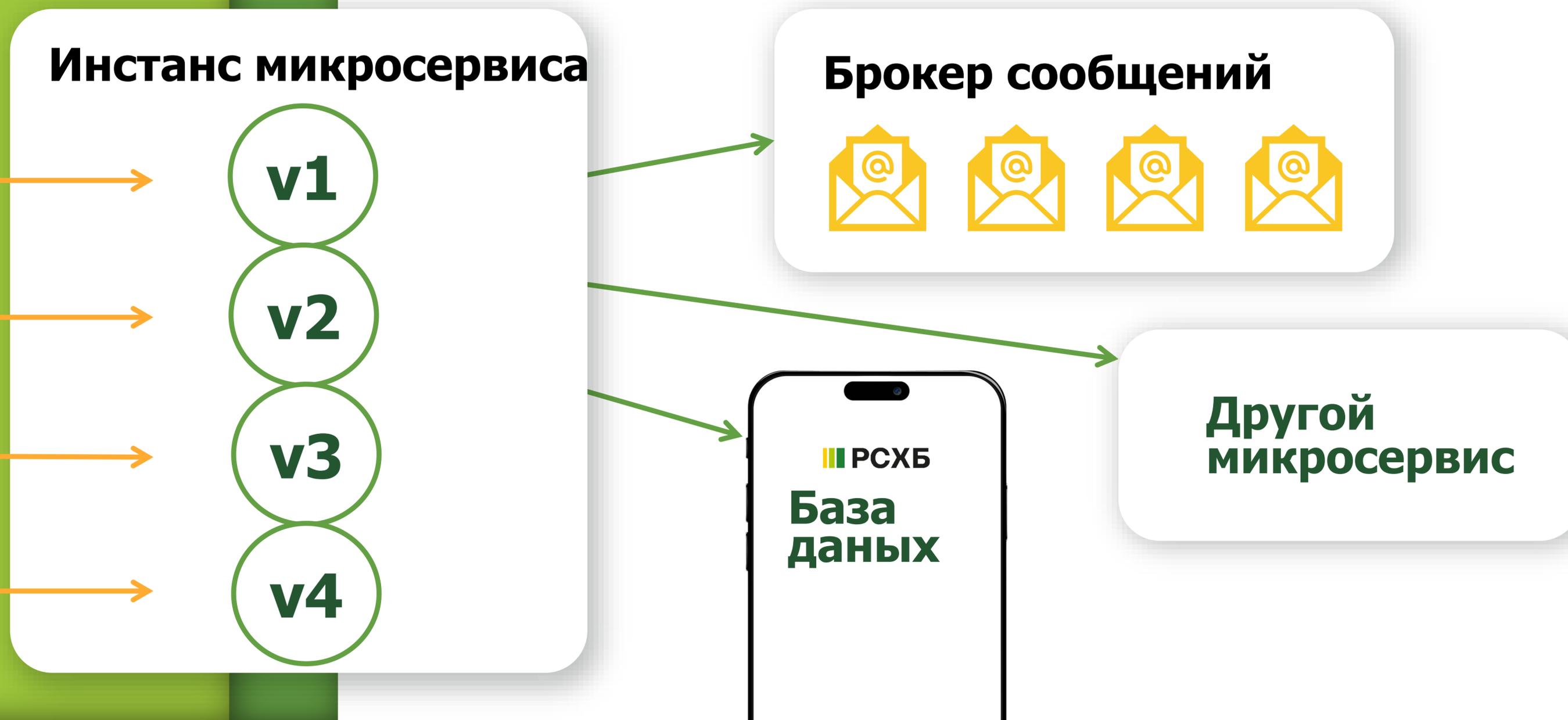
Среднее время  
обработки запроса  
(мс)

НЕЛЬЗЯ ПРОСТО ТАК ВЗЯТЬ

И УВЕЛИЧИТЬ КОЛИЧЕСТВО ПОТОКОВ



# ТРАДИЦИОННЫЙ ЗАПРОС – ПОТОК ПОДХОД



# ТРАДИЦИОННЫЙ ЗАПРОС – ПОТОК ПОДХОД

## Инстанс микросервиса



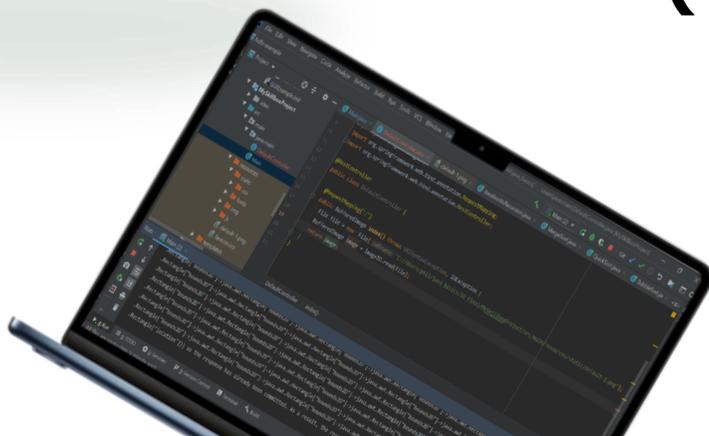
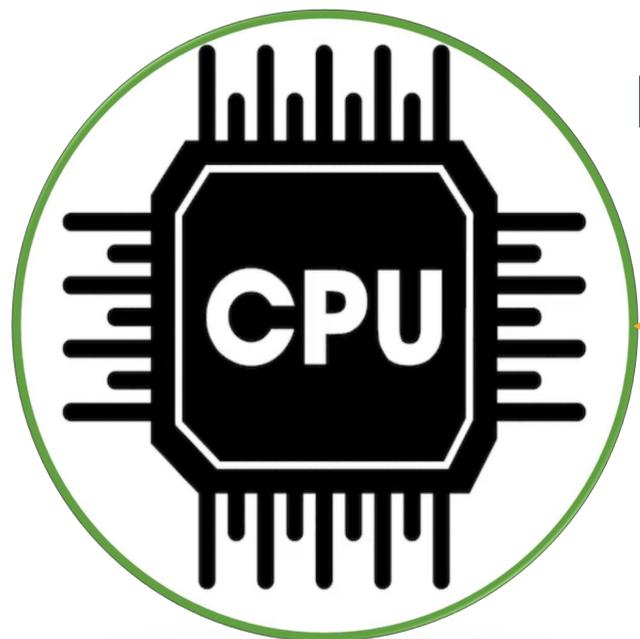
## Брокер сообщений



## Другой микросервис

||| РСХБ  
База  
данных





Выполнение

Переключение контекста

Выбор для исполнения

Вытеснение

Планировщик (часть ОС)

Таблица указателей потоков и их приоритетов

Thread #1 (high)

Thread #2 (low)

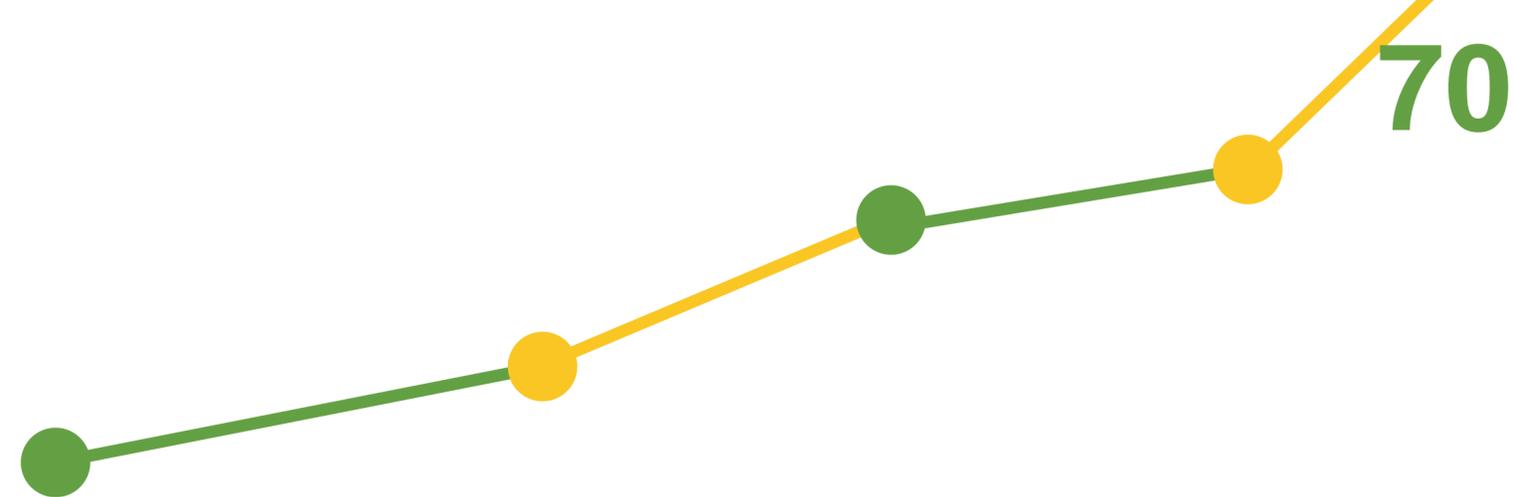
Thread #3 (medium)

Thread #4 (low)

...

Thread #K (high)

# КАТЕГОРИИ ЗАДАЧ



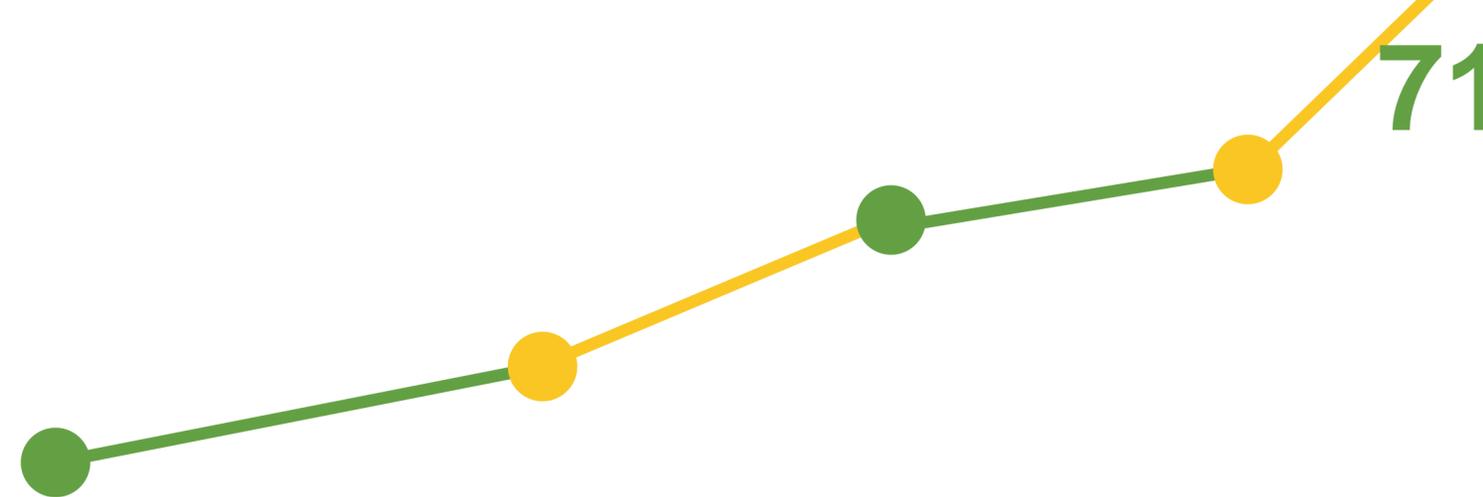
## I/O bound

➤ Доминируют (70-80%)

## CPU bound

➤ Встречаются реже (20-30%)

# КАТЕГОРИИ ЗАДАЧ



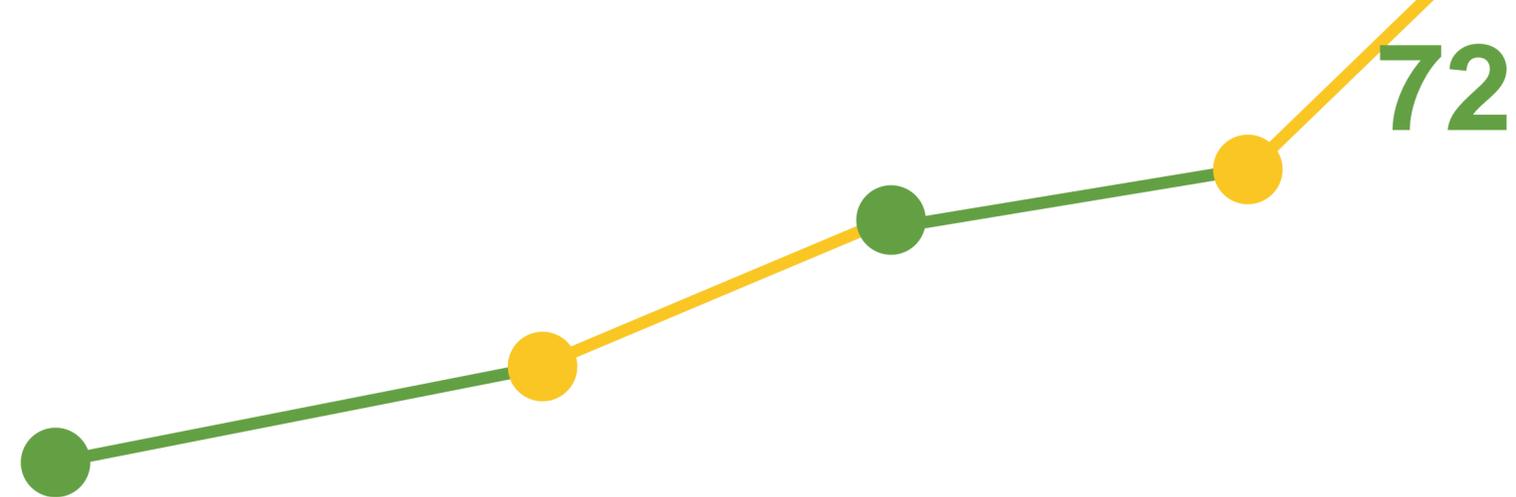
## I/O bound

- Доминируют (70-80%)
- Заставляют потоки/ядра CPU простаивать в ожидании I/O операций

## CPU bound

- Встречаются реже (20-30%)
- Интенсивно используют CPU

# КАТЕГОРИИ ЗАДАЧ



## I/O bound

- Доминируют (70-80%)
- Заставляют потоки/ядра CPU простаивать в ожидании I/O операций
- Являются Reactive Programming, Async Programming и Virtual Threads friendly

## CPU bound

- Встречаются реже (20-30%)
- Интенсивно используют CPU

# Заключение

➔ ForkJoinPool = эффективное распределение

# Заключение

- ForkJoinPool = эффективное распределение
- Continuation = yield, run, манипуляции стек <-> куча

# Заключение

- ForkJoinPool = эффективное распределение
- Continuation = yield, run, манипуляции стек <-> куча
- Java 21 – хорошо, Java 25 – ещё лучше

# Заключение

- ForkJoinPool = эффективное распределение
- Continuation = yield, run, манипуляции стек <-> куча
- Java 21 – хорошо, Java 25 – ещё лучше
- Spring Boot 3.2+ поддерживает виртуальные потоки:  
`spring.threads.virtual.enabled=true`

# Заключение

- ForkJoinPool = эффективное распределение
- Continuation = yield, run, манипуляции стек <-> куча
- Java 21 – хорошо, Java 25 – ещё лучше
- Spring Boot 3.2+ поддерживает виртуальные потоки:  
spring.threads.virtual.enabled=true
- Виртуальные потоки: вся магия спрятана, но работает не всегда.

# Заключение

- ForkJoinPool = эффективное распределение
- Continuation = yield, run, манипуляции стек <-> куча
- Java 21 – хорошо, Java 25 – ещё лучше
- Spring Boot 3.2+ поддерживает виртуальные потоки:  
`spring.threads.virtual.enabled=true`
- Виртуальные потоки: вся магия спрятана, но работает не всегда.
- Имеет смысл попробовать!

Measure, don't guess!