

# ИМПУЛЬС ТИ

## Выжимаем максимум из ПОТОКОВ

Как увеличить производительность приложения  
в 2k25



Как дела с нагрузкой на вашем проекте?

Как дела с нагрузкой на вашем проекте?



У меня  
высоконагруженное  
приложение

Как дела с нагрузкой на вашем проекте?



У меня  
высоконагруженное  
приложение



У меня — нет

Давайте знакомиться:



# Вячеслав Чернышов

Backend-разработчик

СберТех

Давайте знакомиться:



# Вячеслав Чернышов

Backend-разработчик

СберТех

Автор почти 30 статей на Хабре

Давайте знакомиться:



# Вячеслав Чернышов

Backend-разработчик

СберТех

Автор почти 30 статей на Хабре

Автор Telegram-канала Chernyshoff++

Давайте знакомиться:



# Вячеслав Чернышов

Backend-разработчик

СберТех

Автор почти 30 статей на Хабре

Автор Telegram-канала Chernyshoff++

Преподаю в Центральном Университете (Москва)



Профиль на Хабре



Канал в Telegram



Личный сайт

Итак, как справиться с высокой нагрузкой?  
Какие рецепты?

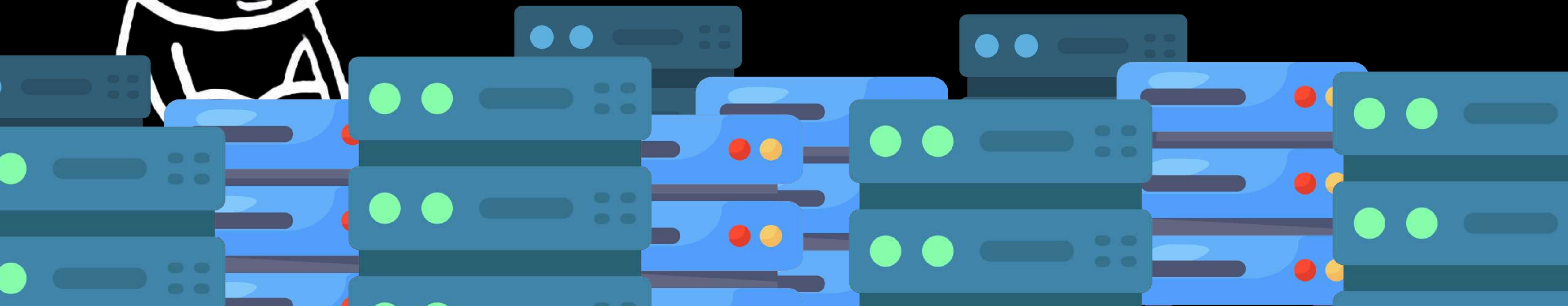
# ГОРИЗОНТАЛЬНОЕ МАСШТАБИРОВАНИЕ



Я подниму 10 000 потоков и  
справлюсь с нагрузкой.



Я подниму 10 000 потоков и  
справлюсь с нагрузкой.



Что мешает это сделать?

Что мешает это сделать?

Первое: ограничение на количество потоков на стороне операционной системы.

## Что не так с ограничением на количество потоков?

Каждая операционная система ограничивает допустимое количество потоков в пределах всей ОС, на одно ядро и на одно приложение.

## Что не так с ограничением на количество потоков?

Каждая операционная система ограничивает допустимое количество потоков в пределах всей ОС, на одно ядро и на одно приложение.

Узнать лимит для текущего приложения:

```
launchctl limit maxproc
```

Soft


Hard

4 000

6 000

Я подниму ~~10 000~~ 6 000 потоков  
и справлюсь с нагрузкой.





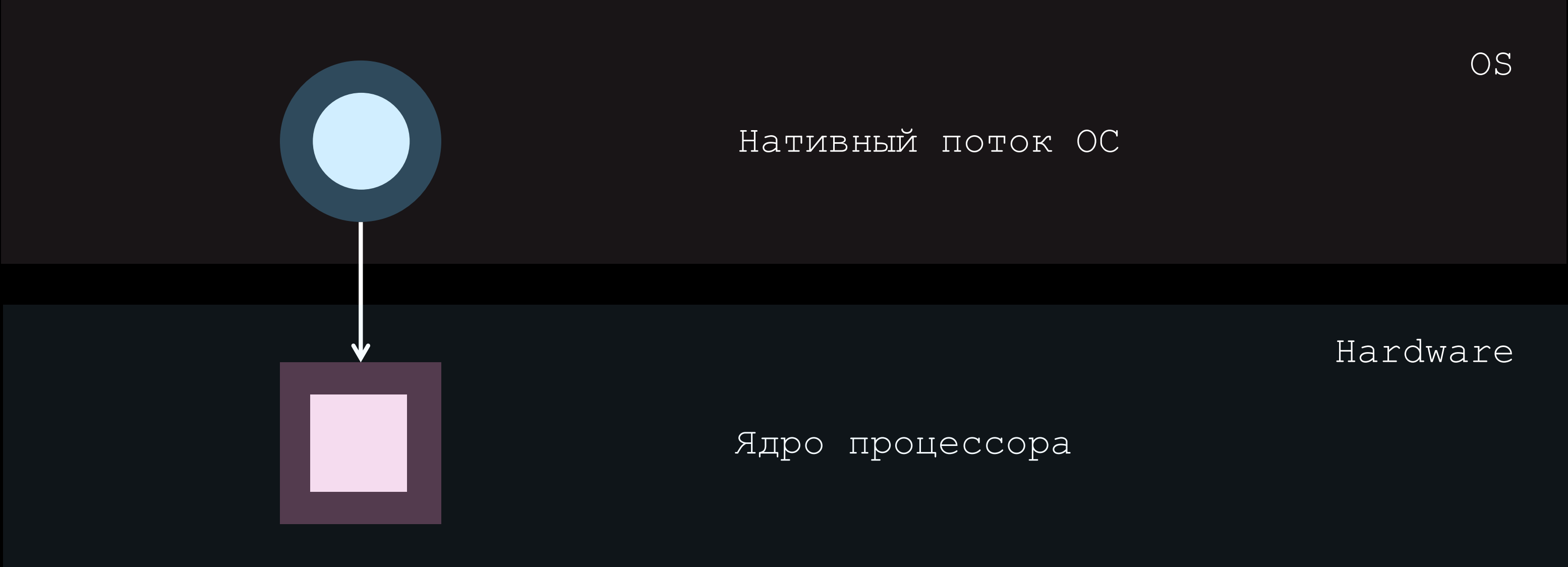
Я подниму ~~10 000~~ 6 000 потоков  
и справлюсь с нагрузкой.

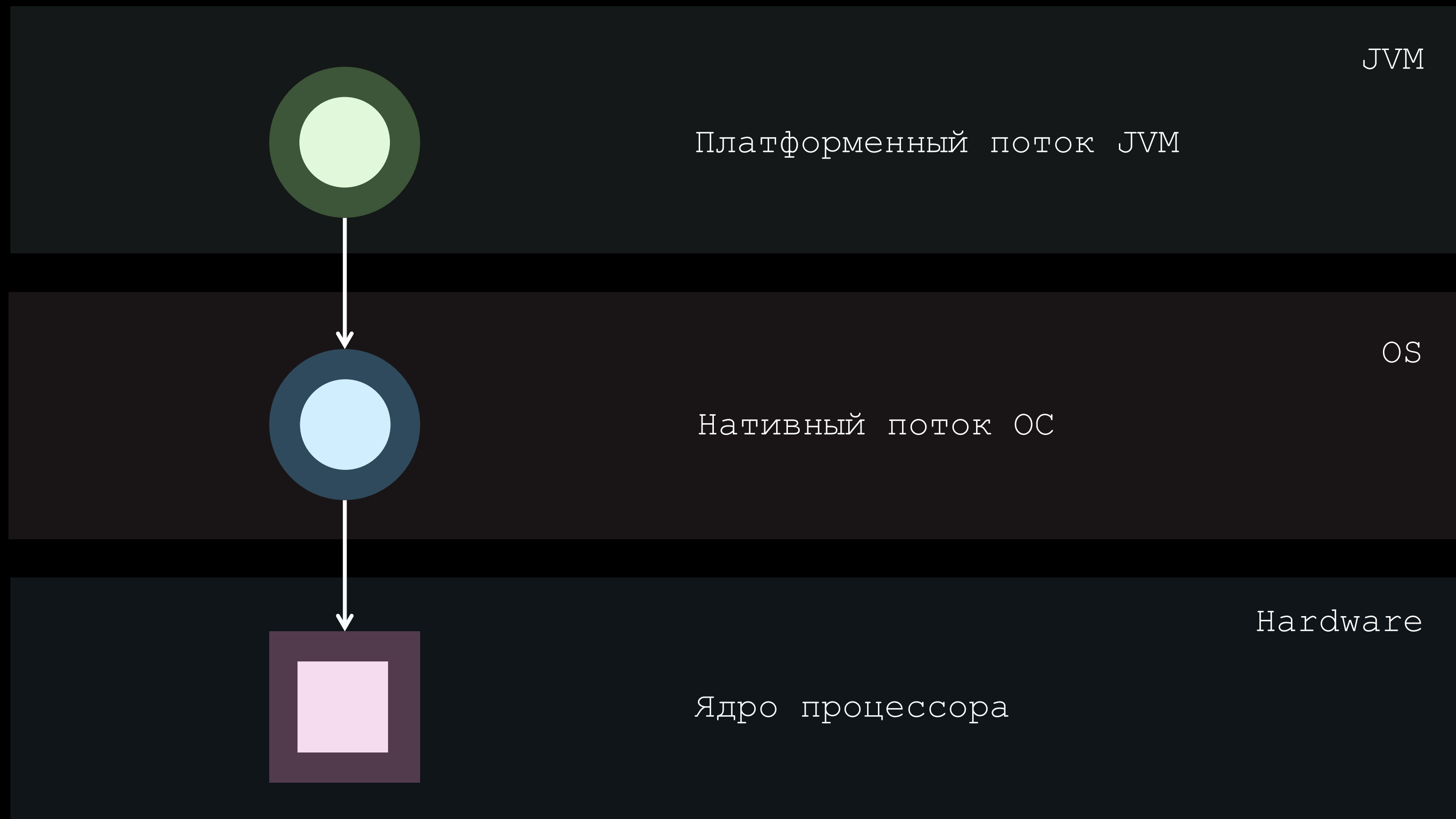
Это плохая идея, и вот  
почему:

Второе: вытесняющая многозадачность

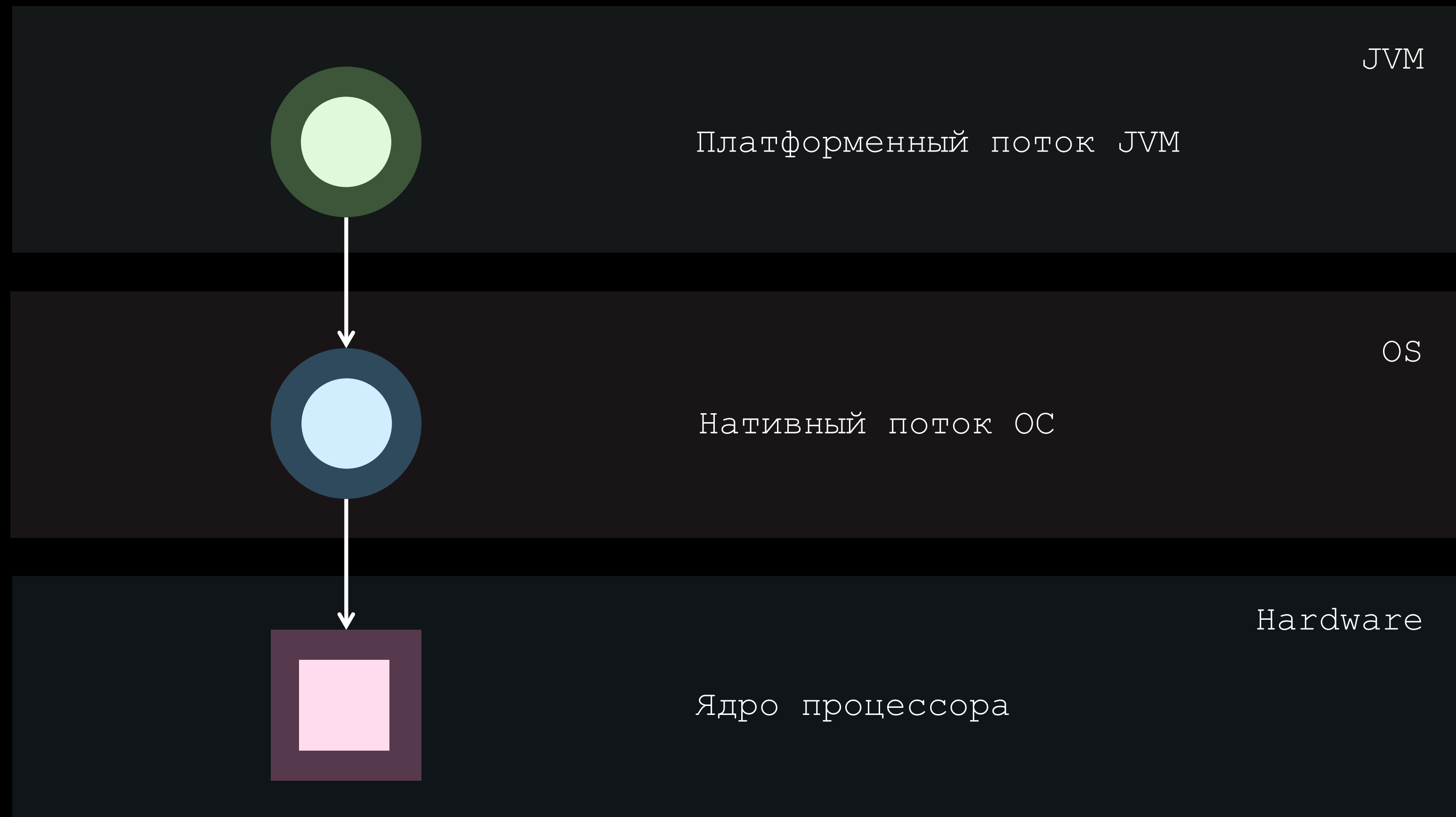
Hardware

Ядро процессора





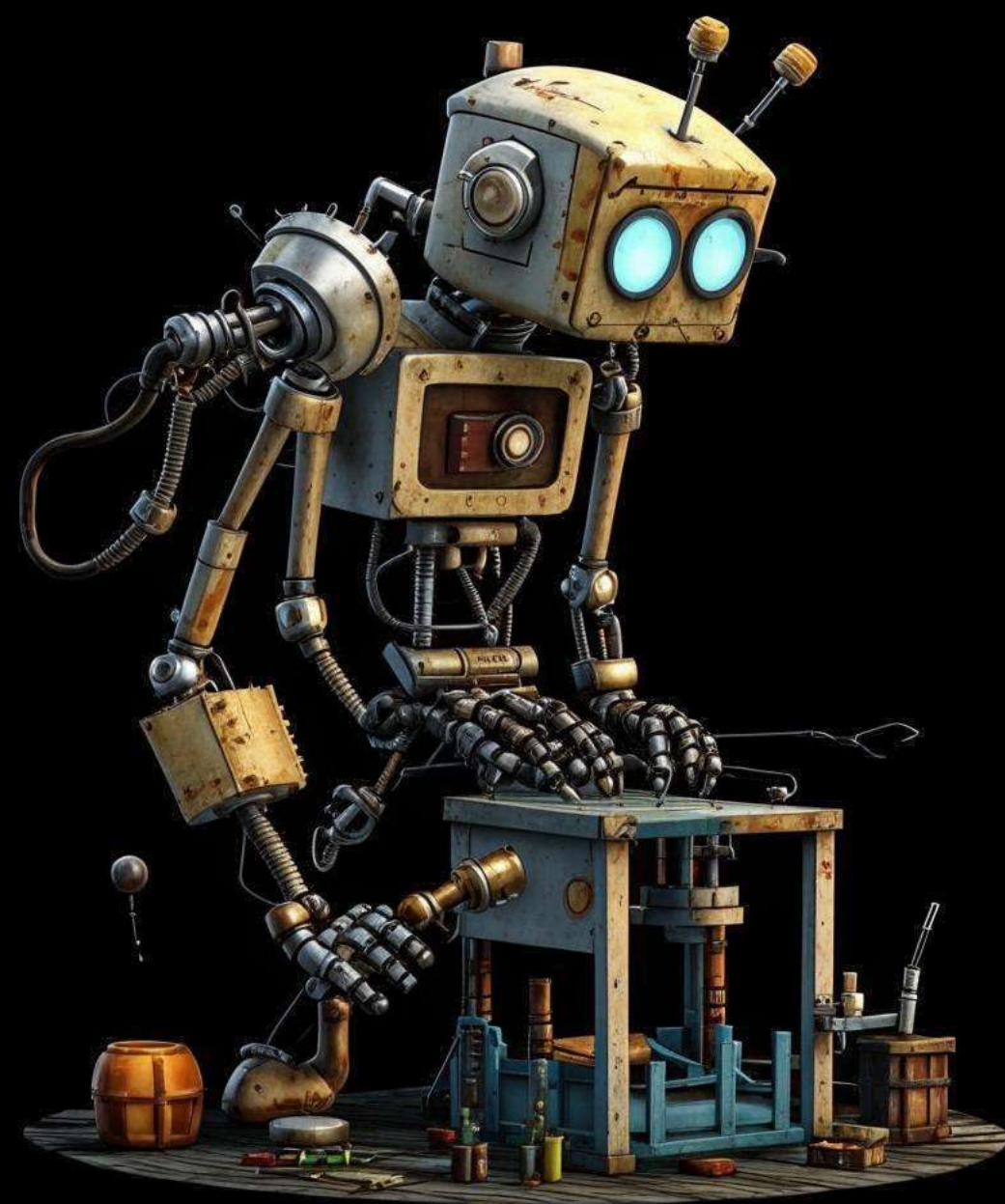
Платформенные потоки JVM – это обёртки над нативными потоками операционной системы 1-к-1.



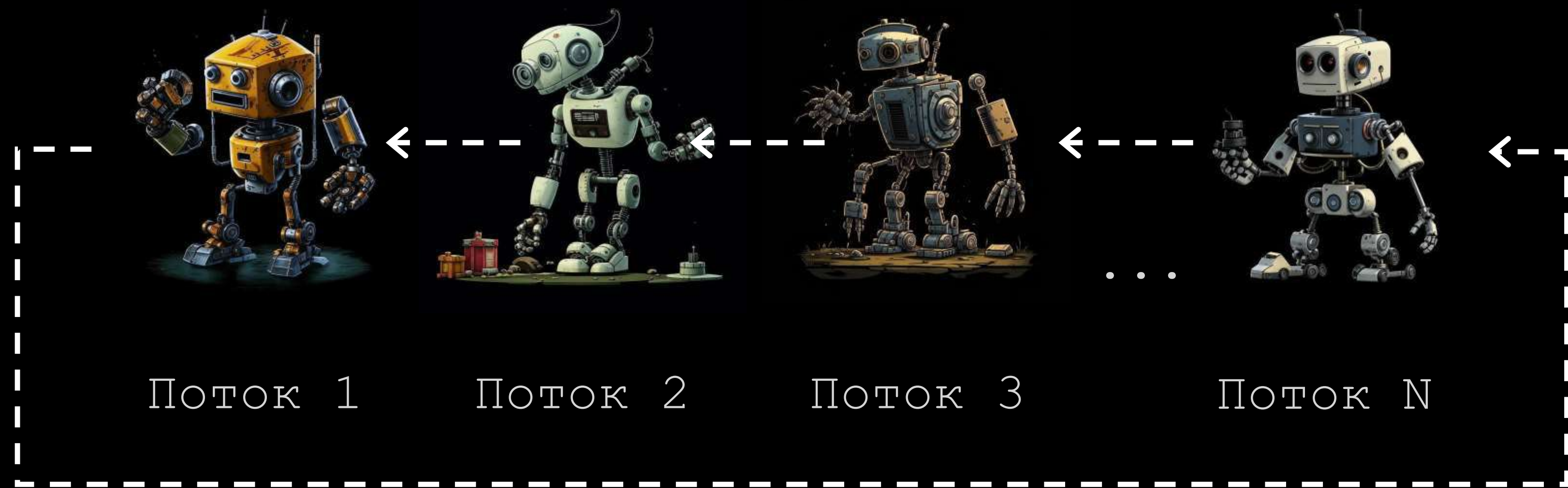
Почему нельзя взять и создать 6 000 потоков?

Каждому потоку выделяется квант времени, во время которого процессор работает с потоком.

Как только квант времени исчерпан, поток вытесняется планировщиком и перемещается в конец очереди.



Ядро процессора



Поток 1

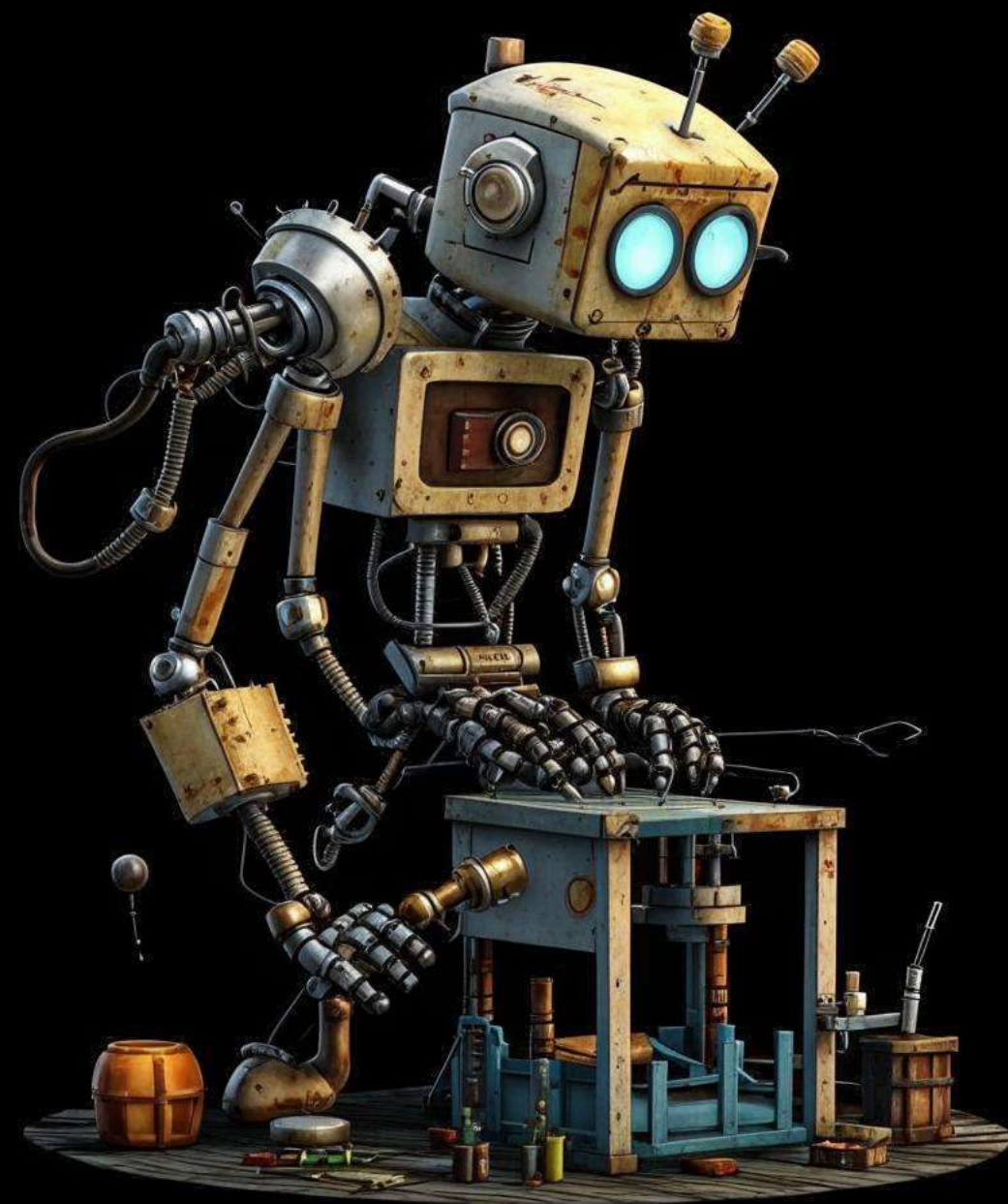
Поток 2

Поток 3

Поток N

Почему нельзя взять и создать 6 000 потоков?

Такое переключение — очень дорогостоящая операция.



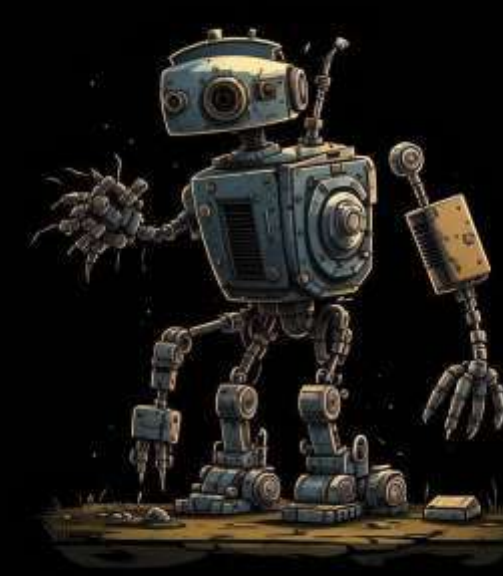
Ядро процессора



Поток 1



Поток 2



Поток 3

...



Поток N

**Вывод:** при множестве потоков на одно ядро процессор захлебнётся на переключении контекста.

Ещё причины?

Ещё причины?

Третье: размер платформенного потока

Из чего складывается размер оперативной памяти,  
потребляемой потоком?

Из чего складывается размер оперативной памяти,  
потребляемой потоком?

Стековая память

Сколько?

Из чего складывается размер оперативной памяти,  
потребляемой потоком?

Стековая память

32 bit: 320 KB

64 bit: 2 MB

Из чего складывается размер оперативной памяти,  
потребляемой потоком?

Стековая память

32 bit: 320 KB

64 bit: 2 MB

```
java -XX:+PrintFlagsFinal -version | grep ThreadStackSize
```

Из чего складывается размер оперативной памяти,  
потребляемой потоком?

Стековая память

32 bit: 320 KB

64 bit: 2 MB

```
% java -XX:+PrintFlagsFinal -version | grep ThreadStackSize
intx CompilerThreadStackSize          = 2048          {pd product} {default}
intx ThreadStackSize                  = 2048          {pd product} {default}
intx VMThreadStackSize                = 2048          {pd product} {default}
openjdk version "24.0.1" 2025-04-15
OpenJDK Runtime Environment (build 24.0.1+9-30)
OpenJDK 64-Bit Server VM (build 24.0.1+9-30, mixed mode, sharing)
```

Из чего складывается размер оперативной памяти,  
потребляемой потоком?

Стековая память

32 bit: 320 KB

64 bit: 2 MB

Нативные ресурсы ОС

~ 2-4 KB

Из чего складывается размер оперативной памяти,  
потребляемой потоком?

Стековая память	32 bit: 320 KB 64 bit: 2 MB
Нативные ресурсы ОС	~ 2-4 KB
Хранение состояния	~ 1-2 KB

Из чего складывается размер оперативной памяти,  
потребляемой потоком?

Стековая память	32 bit: 320 KB 64 bit: 2 MB
-----------------	--------------------------------

Нативные ресурсы ОС	~ 2-4 KB
---------------------	----------

Хранение состояния	~ 1-2 KB
--------------------	----------

---

Total	2.005 MB для Java 24
-------	----------------------

1 000 потоков потребует 2 ГБ RAM, и это уже немало, но терпимо.

1 000 потоков потребует 2 ГБ RAM, и это уже немало, но терпимо.

6 000 потоков потребует 12 ГБ RAM, и это уже много.

1 000 потоков потребует 2 ГБ RAM, и это уже немало, но терпимо.

6 000 потоков потребует 12 ГБ RAM, и это уже много.

Но возможно, системы столько и не требуют, и нам никогда не потребуется столько потоков?

Давайте посчитаем ещё раз.

# Плановая нагрузка на разные типы сервисов \*

## Плановая нагрузка на разные типы сервисов \*

Сервис	RPS
Один известный поисковик	30-60 К

## Плановая нагрузка на разные типы сервисов \*

Сервис	RPS
Один известный поисковик	30-60 К
Один известный видеохостинг	1-2 М

## Плановая нагрузка на разные типы сервисов \*

Сервис	RPS
Один известный поисковик	30-60 К
Один известный видеохостинг	1-2 М
Одна известная социальная сеть	2-5 М

## Плановая нагрузка на разные типы сервисов \*

Сервис	RPS
Один известный поисковик	30-60 K
Один известный видеохостинг	1-2 M
Одна известная социальная сеть	2-5 M
Один известный интернет-магазин	5-10 M

## Плановая нагрузка на разные типы сервисов \*

Сервис	RPS
Один известный поисковик	30-60 K
Один известный видеохостинг	1-2 M
Одна известная социальная сеть	2-5 M
Один известный интернет-магазин	5-10 M
Один известный мессенджер	10-20 M

# Плановая нагрузка на разные типы сервисов \*

Сервис

RPS

SaaS-стартап

100 – 10 К

# Плановая нагрузка на разные типы сервисов \*

Сервис	RPS
SaaS-стартап	100 – 10 К
Банковское приложение	1 К – 50 К

# Плановая нагрузка на разные типы сервисов \*

Сервис	RPS
SaaS-стартап	100 – 10 К
Банковское приложение	1 К – 50 К
Криптовбиржа	100 К – 1 М

Среднее время выполнения запроса: 150 ms.

Среднее время выполнения запроса: 150 ms.

Это значит, что один поток может выполнить 6,(6) запросов  
в секунду.

Среднее время выполнения запроса: 150 ms.

Это значит, что один поток может выполнить 6,(6) запросов  
в секунду.

Округлим до 6.

# Плановая нагрузка на разные типы сервисов \*

Время выполнения запроса

150 ms

Сервис

RPS

SaaS-стартап

100 – 10 K

Банковское приложение

1 K – 50 K

Криптовбиржа

100 K – 1 M

# Плановая нагрузка на разные типы сервисов \*

Время выполнения запроса

150 ms

Сервис

RPS

Сколько потоков потребуется

SaaS-стартап

100 – 10 K

15 – 1 500

Банковское приложение

1 K – 50 K

150 – 7 500

Криптовбиржа

100 K – 1 M

15 – 150 K

# Плановая нагрузка на разные типы сервисов \*

Сервис	RPS	Время выполнения запроса 150 ms	Сколько потоков потребуется	RAM на один поток 2 MB
SaaS-стартап	100 – 10 K		15 – 1 500	
Банковское приложение	1 K – 50 K		150 – 7 500	
Криптовбиржа	100 K – 1 M		15 – 150 K	

## Плановая нагрузка на разные типы сервисов \*

Сервис	RPS	Время выполнения запроса 150 ms	Сколько потоков потребуется	RAM на один поток 2 MB	Сколько RAM они займут
SaaS-стартап	100 – 10 K		15 – 1 500		30 MB – 3 GB
Банковское приложение	1 K – 50 K		150 – 7 500		300 MB – 15 GB
Криптовбиржа	100 K – 1 M		15 – 150 K		30 – 300 GB

Для любого серьёзного сервиса потоки выглядят **ЗОЛОТЫМИ**.

Для любого серьёзного сервиса потоки выглядят **ЗОЛОТЫМИ**.

Что можно оптимизировать в первую очередь?

Для любого серьёзного сервиса потоки выглядят **ЗОЛОТЫМИ**.

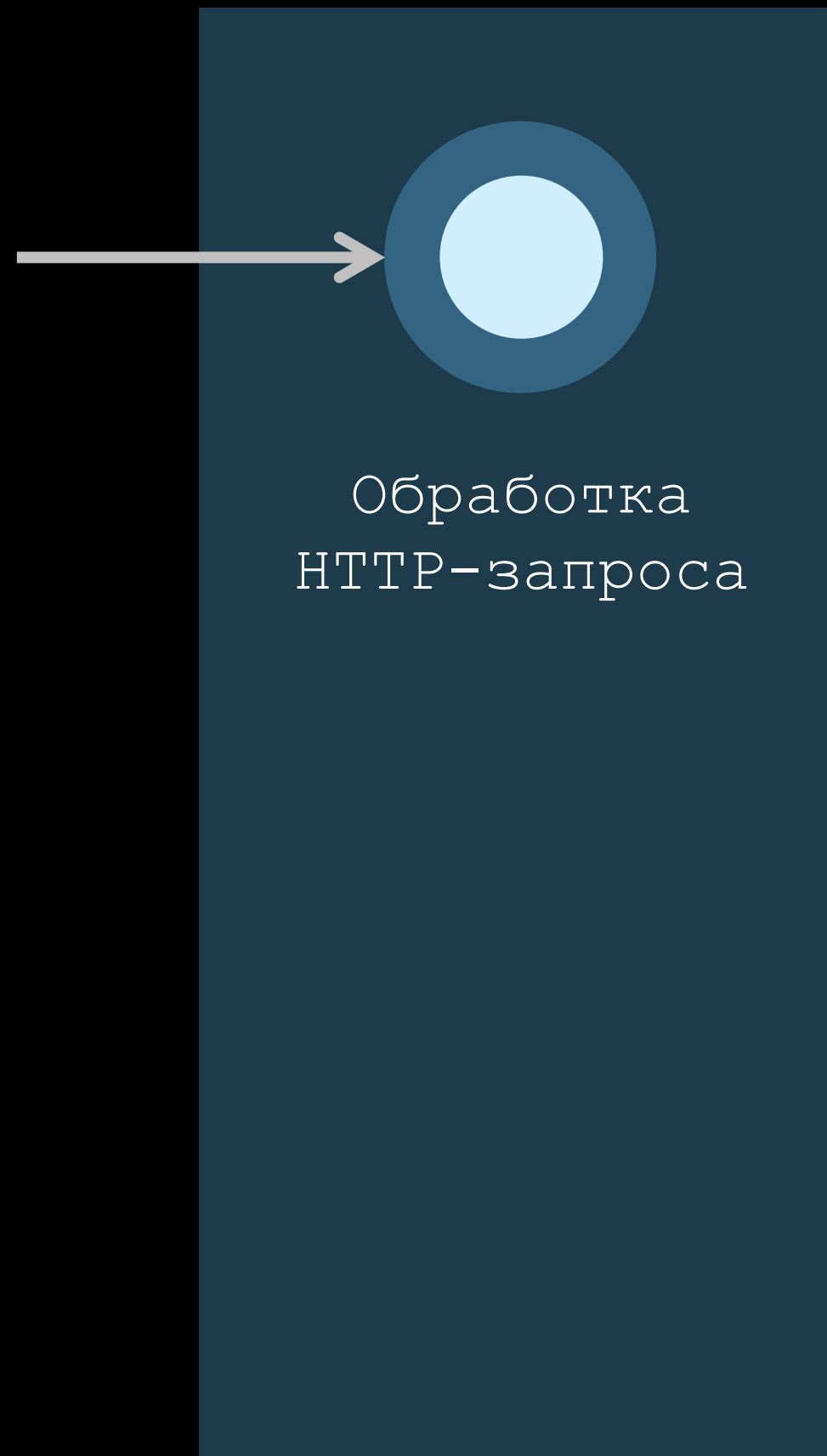
Что можно оптимизировать в первую очередь?

Время выполнения запроса.

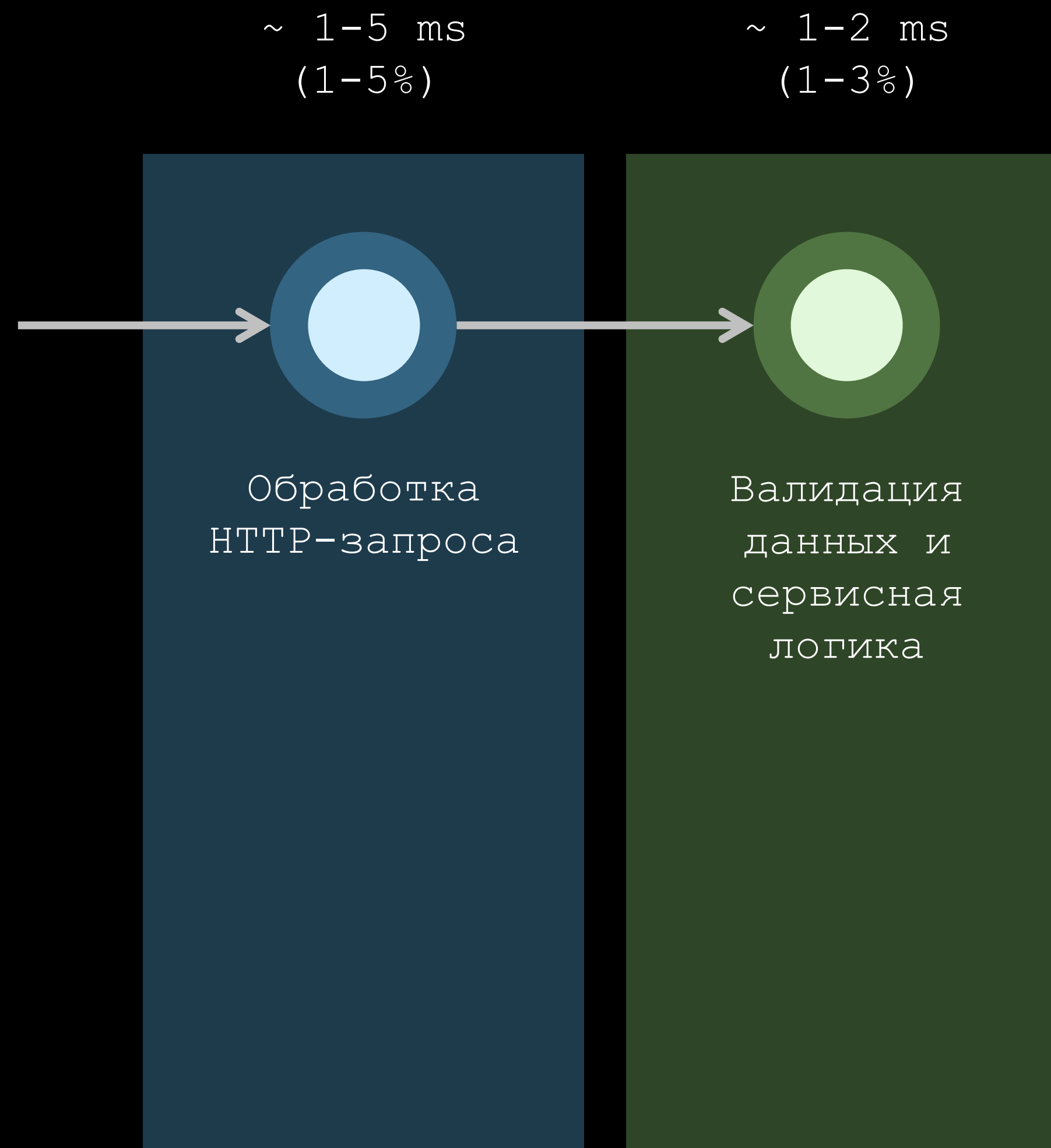
## Время выполнения запроса блокирующим потоком

## Время выполнения запроса блокирующим потоком

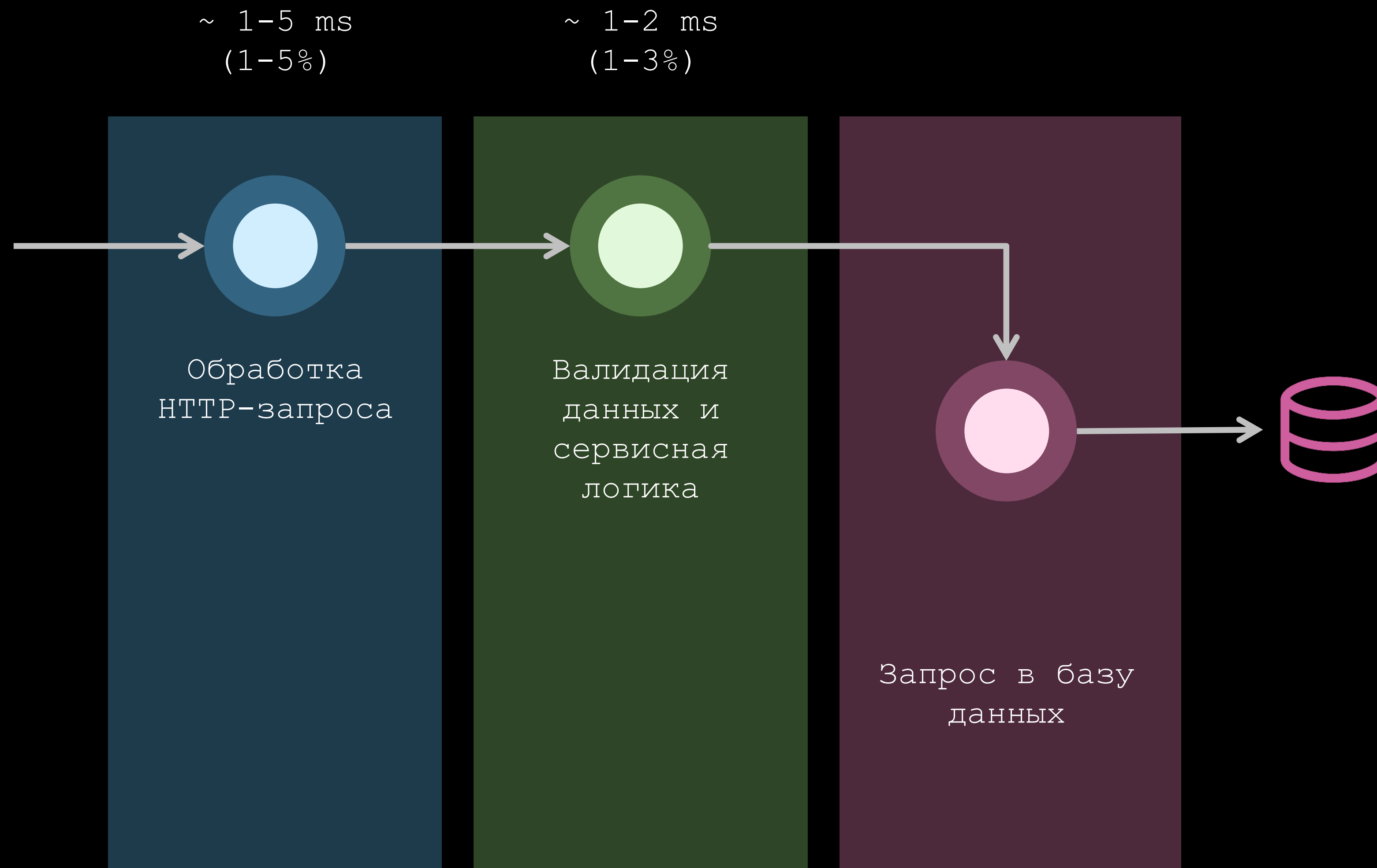
~ 1-5 ms  
(1-5%)



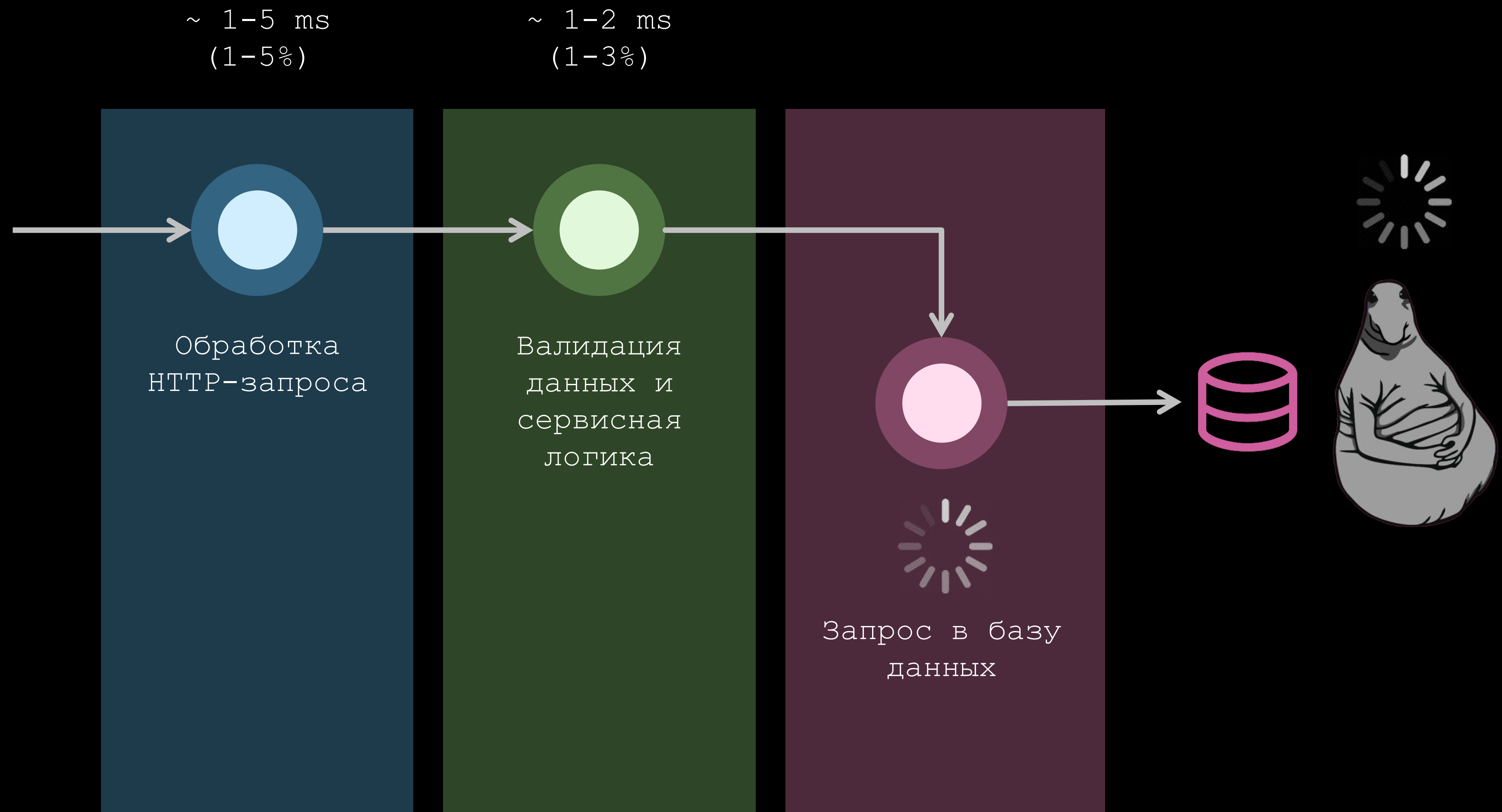
## Время выполнения запроса блокирующим потоком



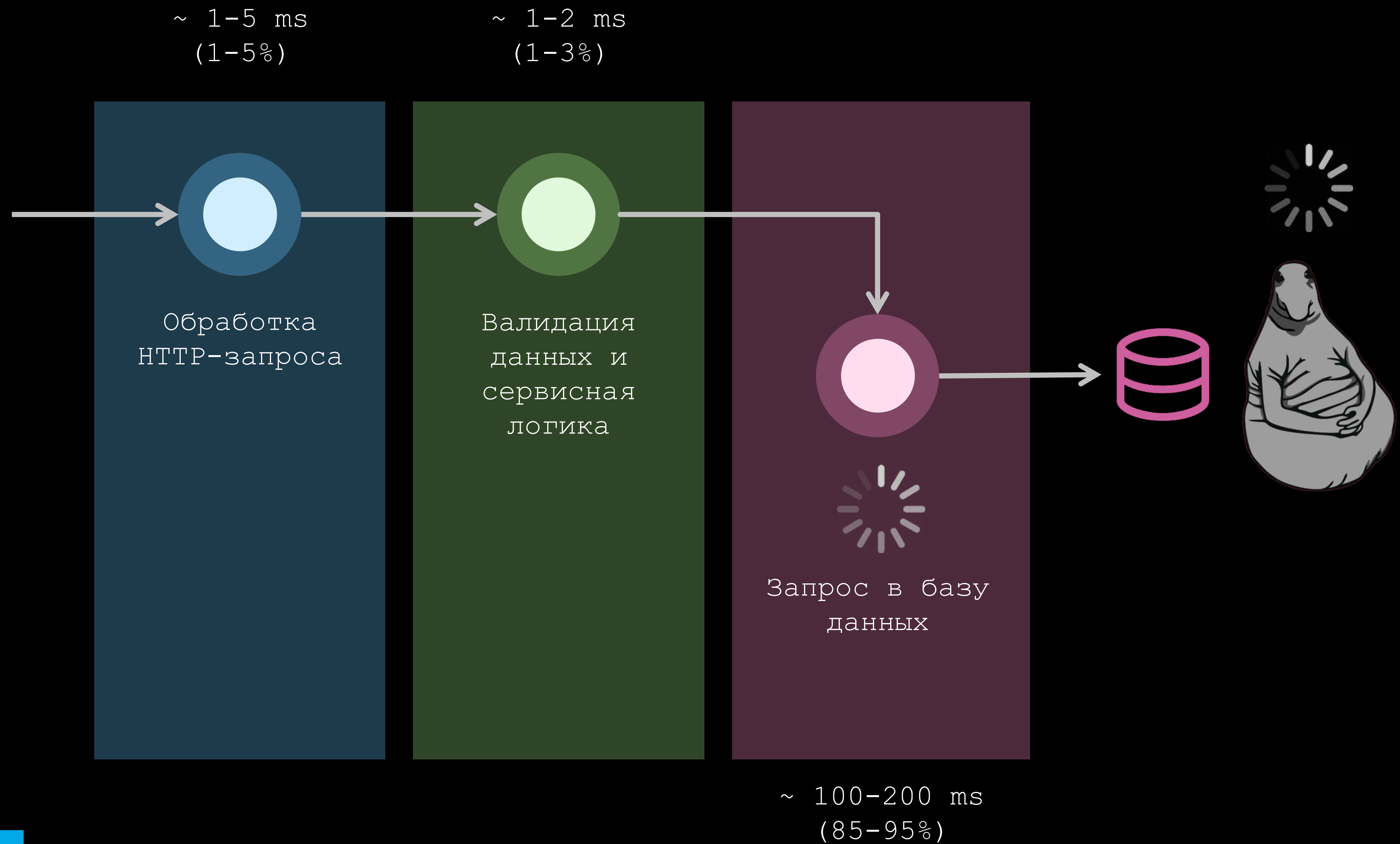
# Время выполнения запроса блокирующим потоком



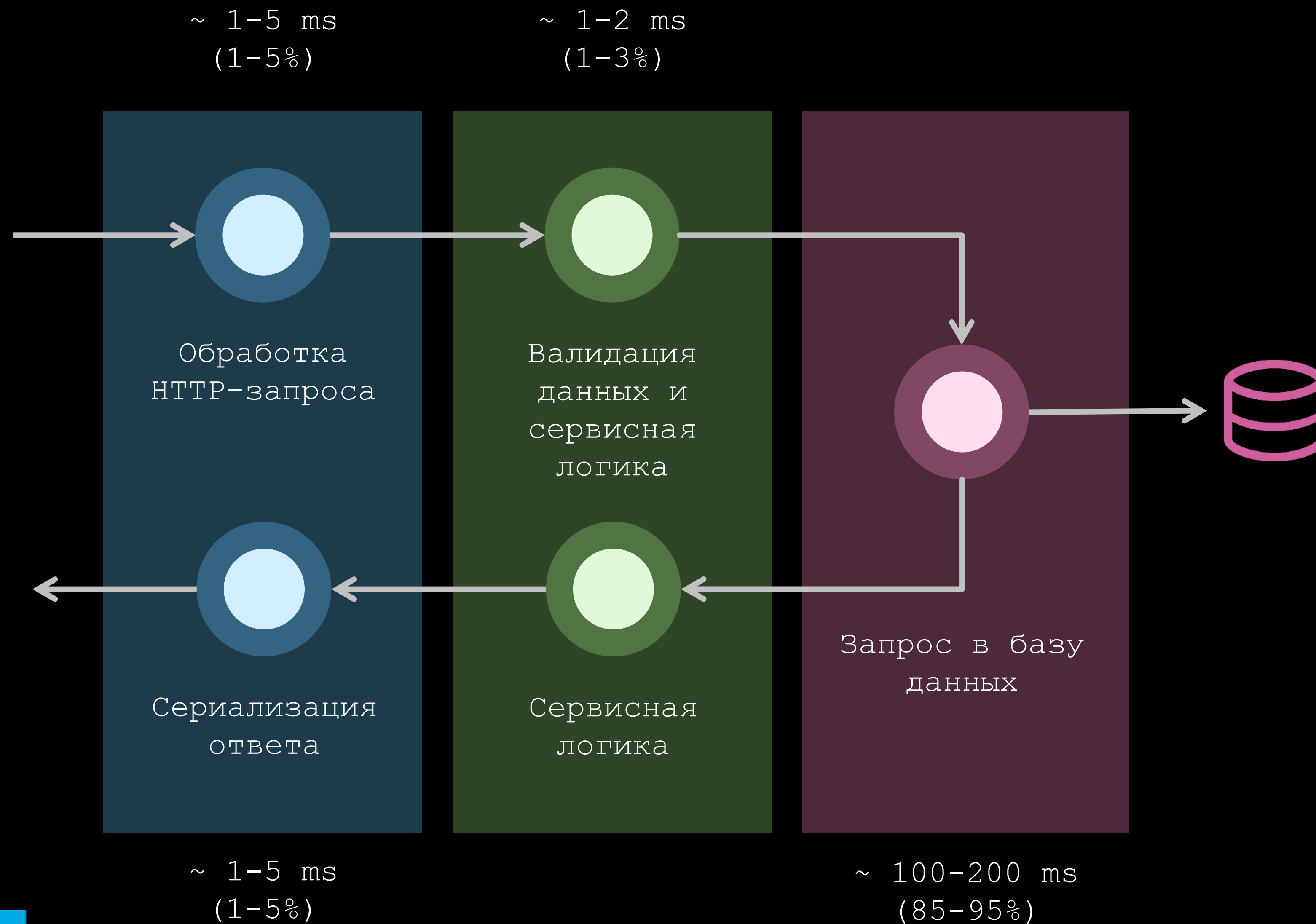
# Время выполнения запроса блокирующим потоком



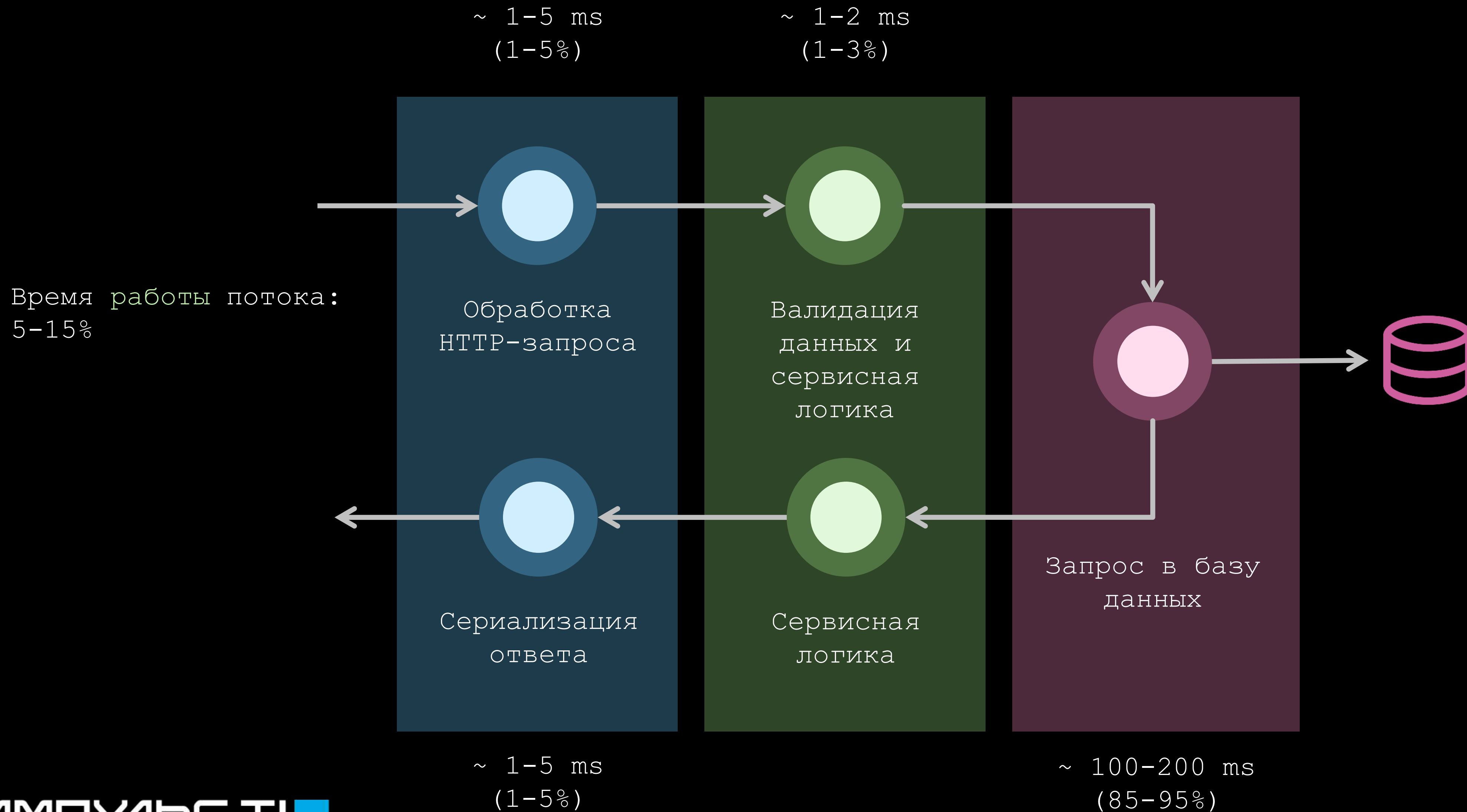
# Время выполнения запроса блокирующим потоком



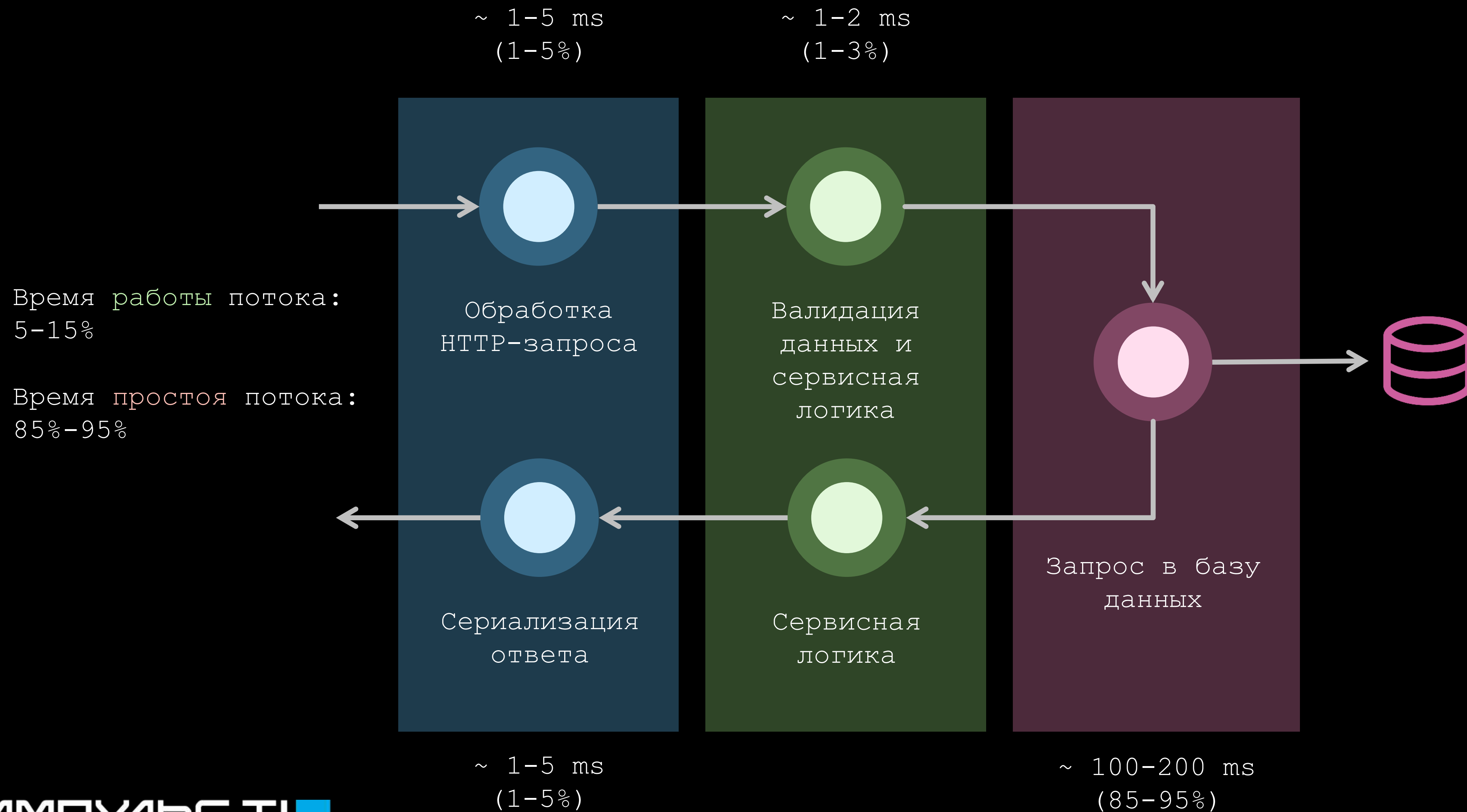
# Время выполнения запроса блокирующим потоком



# Время выполнения запроса блокирующим потоком



# Время выполнения запроса блокирующим потоком



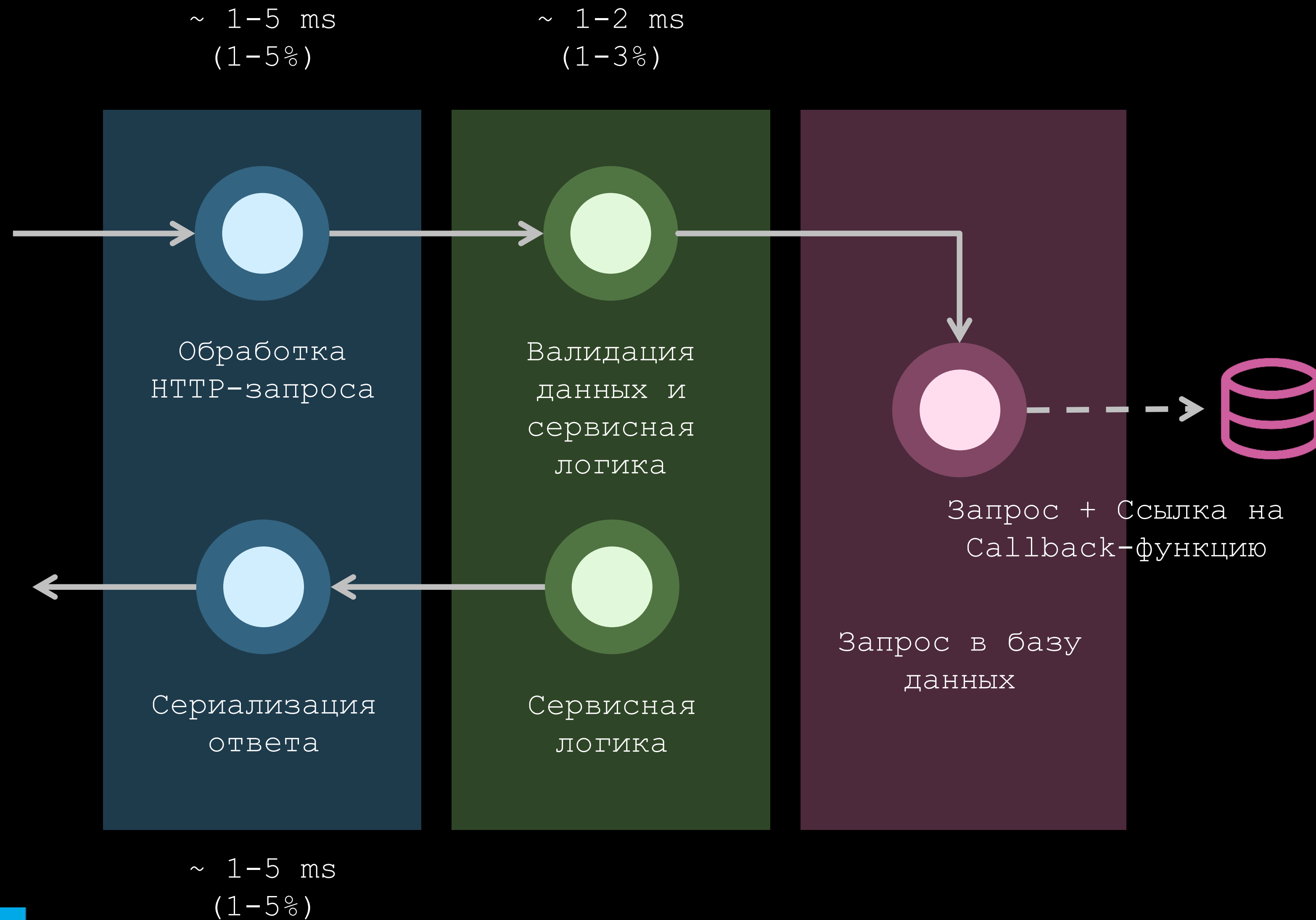
Кажется, что потоки едят свою оперативку незаслуженно.

Кажется, что потоки едят свою оперативку незаслуженно.

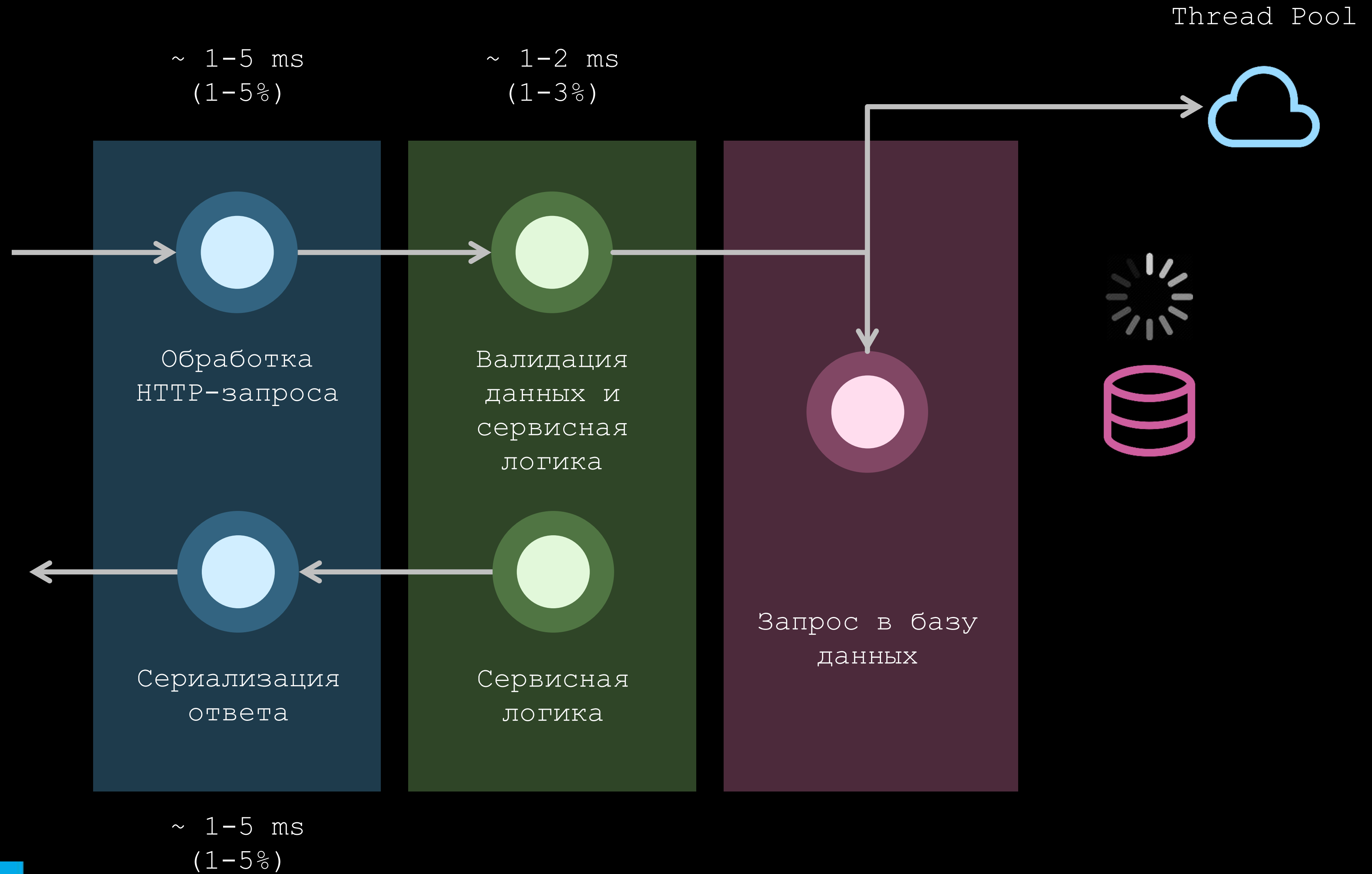
Поток не должен блокироваться на время ожидания выполнения операции ввода-вывода.

# Время выполнения запроса реактивным потоком

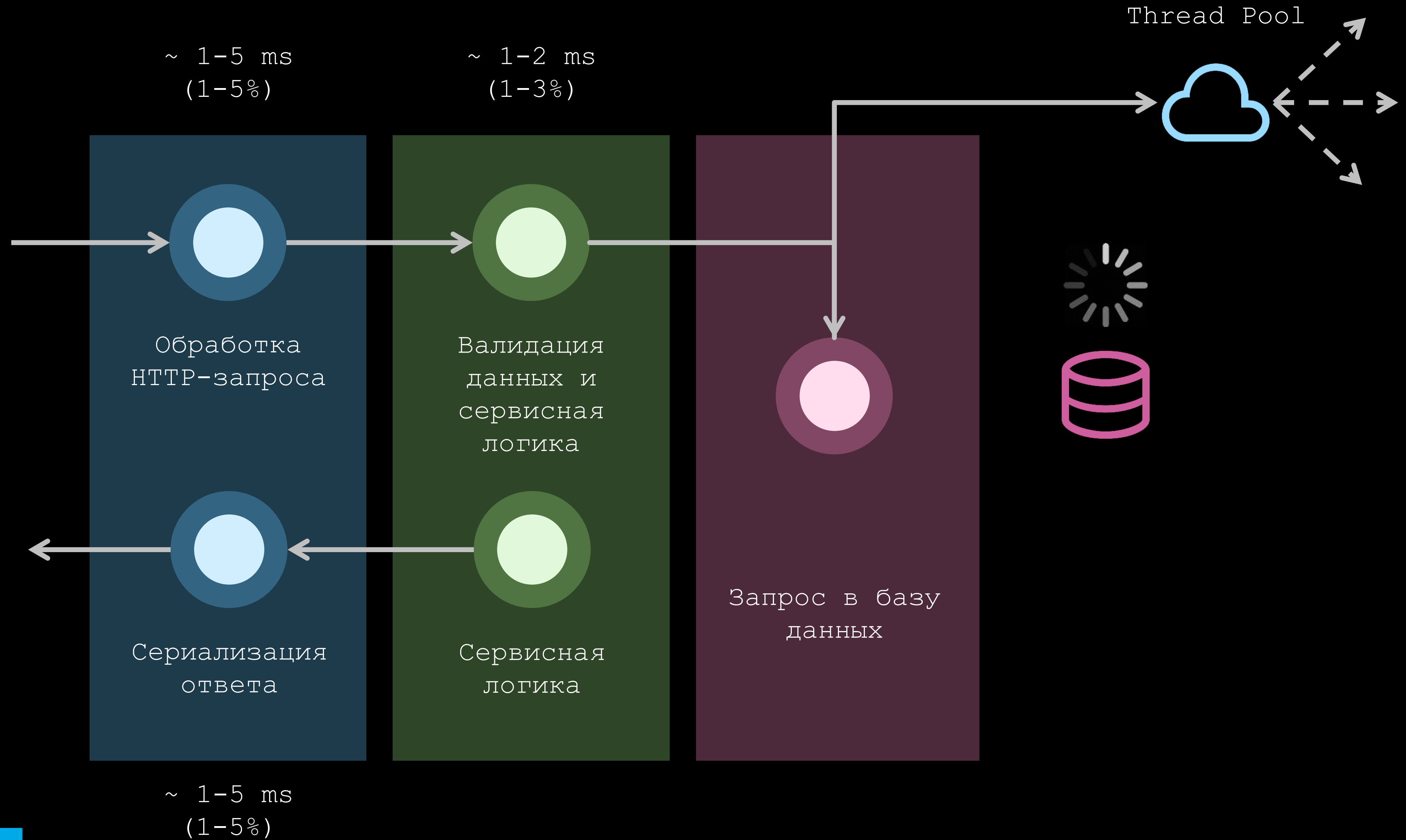
# Время выполнения запроса реактивным потоком



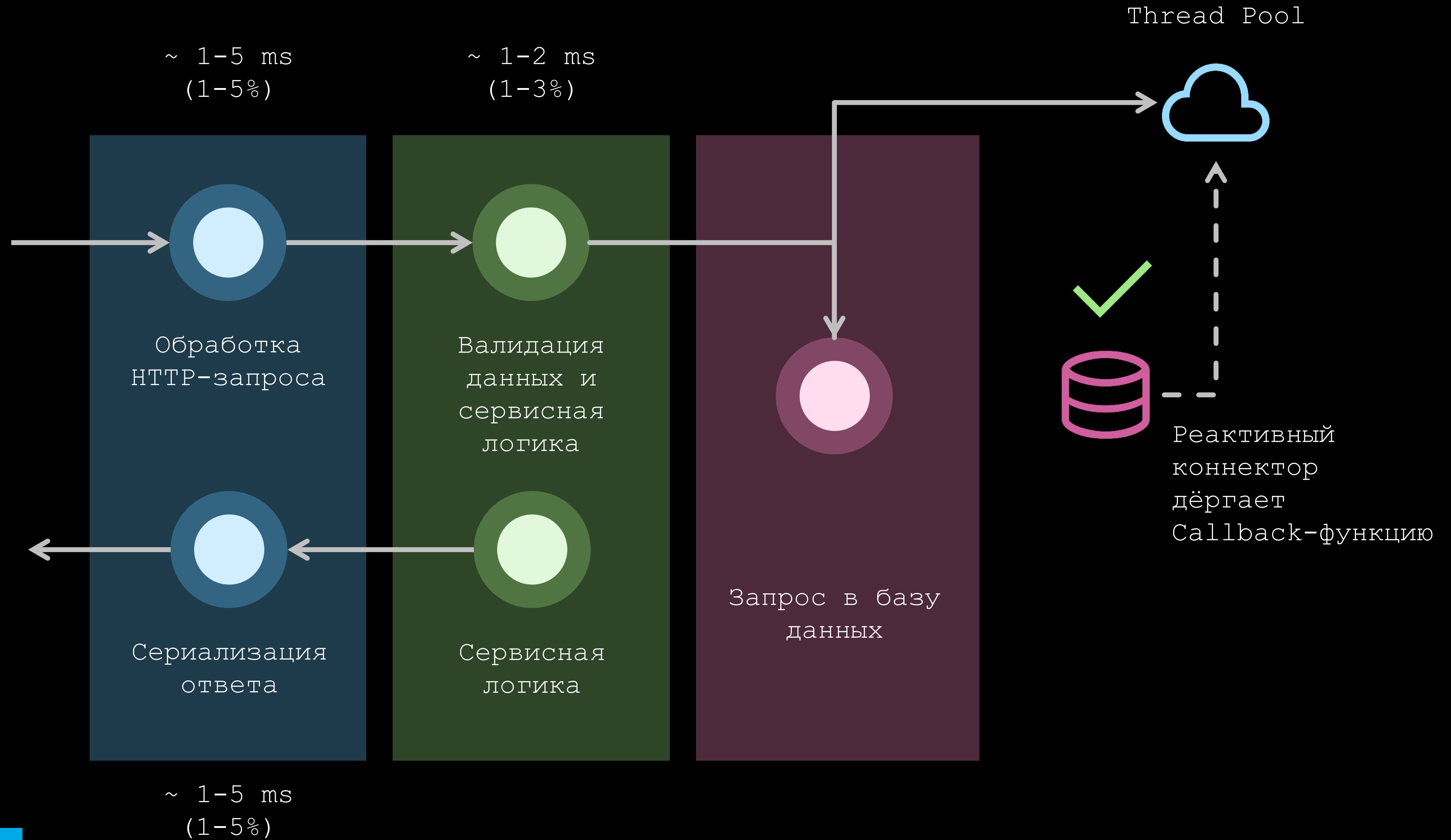
# Время выполнения запроса реактивным потоком



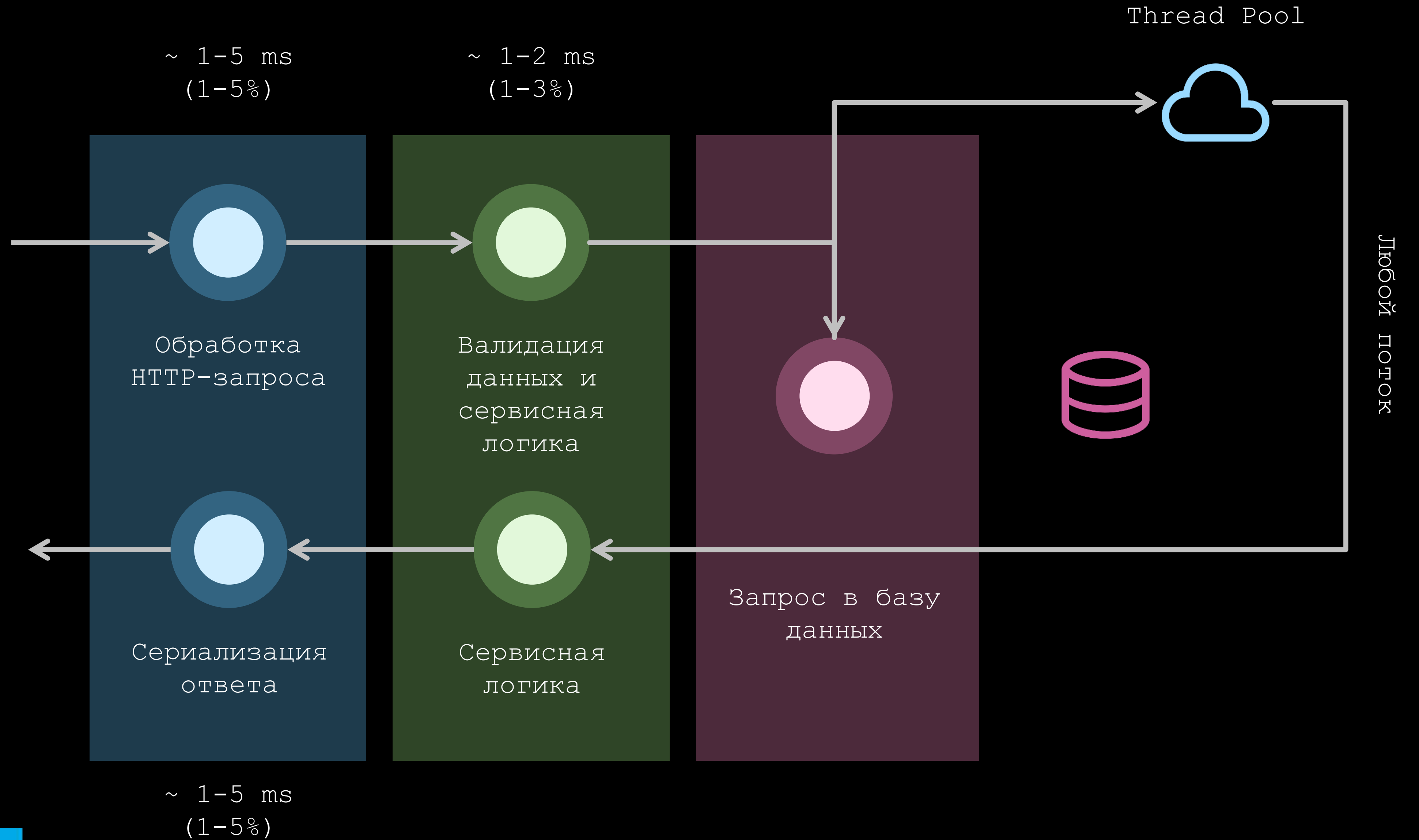
# Время выполнения запроса реактивным потоком



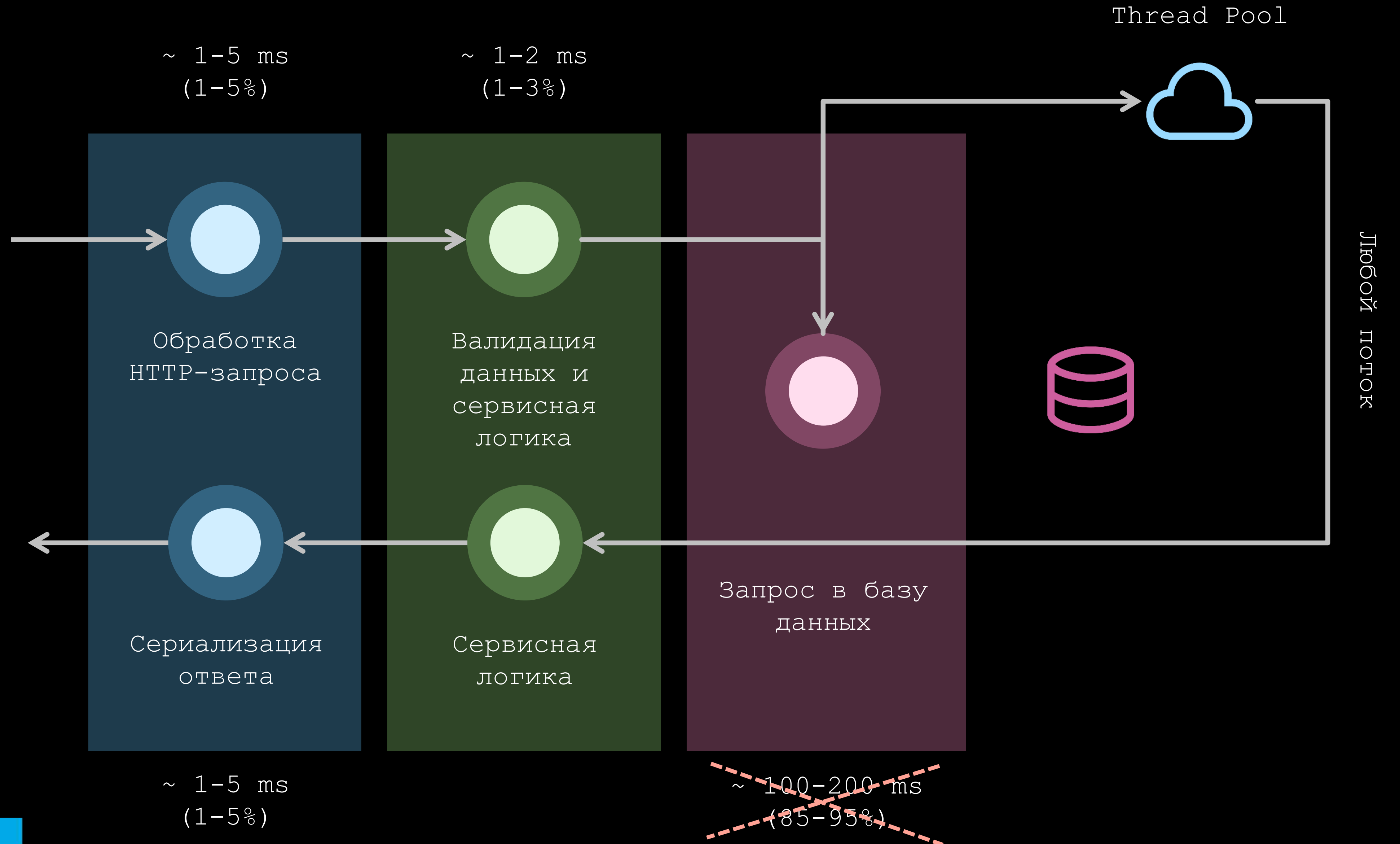
# Время выполнения запроса реактивным потоком



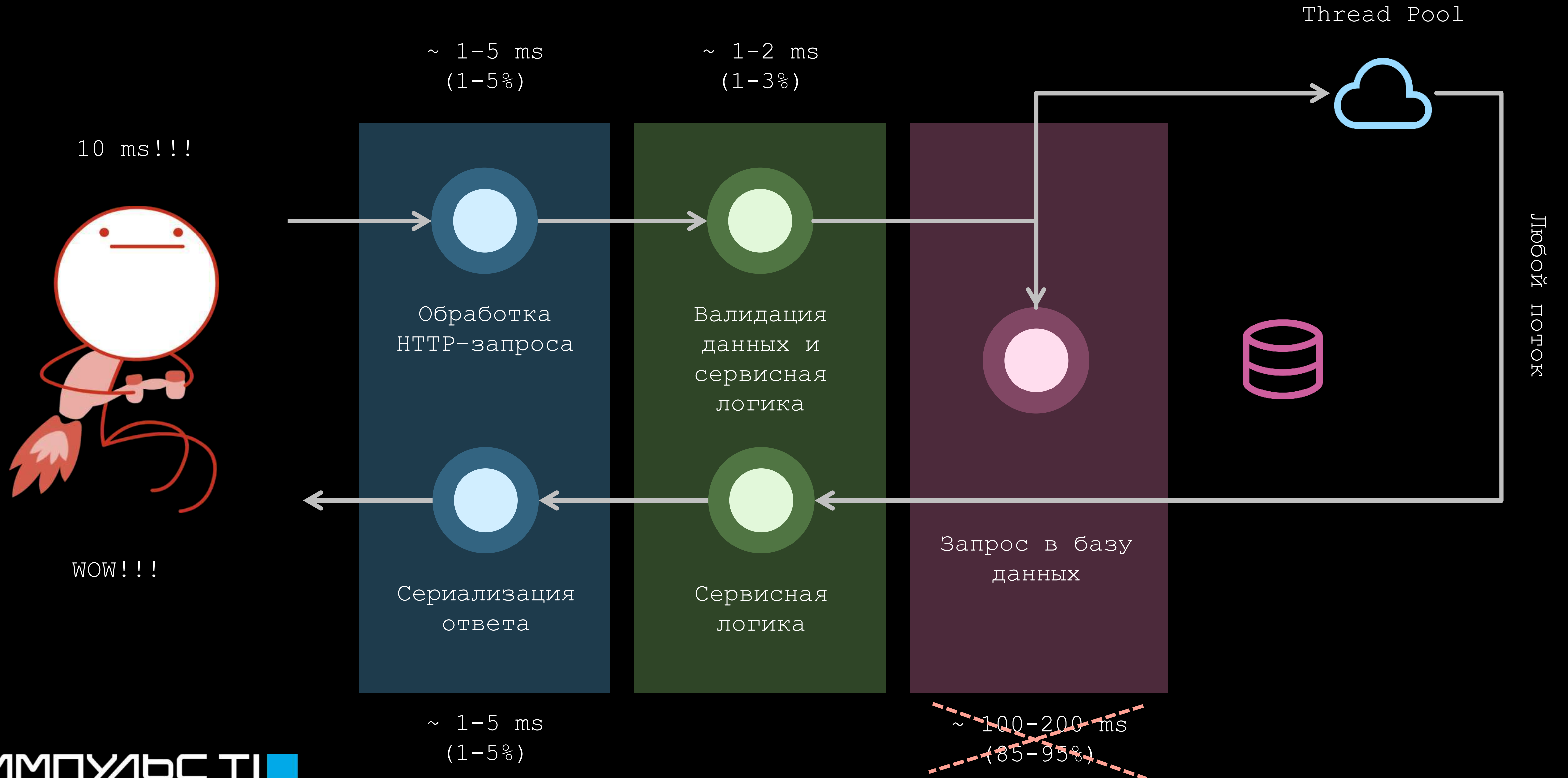
# Время выполнения запроса реактивным потоком



# Время выполнения запроса реактивным потоком



# Время выполнения запроса реактивным потоком



Первый шаг оптимизации: переход на реактивный стек.

Реактивные потоки едят свою оперативку заслуженно.

# Плановая нагрузка на разные типы сервисов

Время выполнения запроса

Блокирующий стек: 150 ms



	Кол-во потоков	RAM
Один известный поисковик	4.5 - 9 K	9 – 18 GB
Один известный видеохостинг	150 – 300 K	300 – 600 GB
Одна известная социальная сеть	300 – 750 K	600 GB – 1.5 TB
Один известный интернет-магазин	750 K – 1.5 M	1.5 – 3 TB
Один известный мессенджер	1.5 - 3 M	3 – 6 TB
SaaS-стартап	15 – 1 500	30 MB – 3 GB
Банковское приложение	150 – 7 500	300 MB – 15 GB
Криптовалютная биржа	15 – 150 K	30 – 300 GB

# Плановая нагрузка на разные типы сервисов

Время выполнения запроса

Блокирующий стек: 150 ms

Реактивный стек: 10 ms

	Блокирующий стек: 150 ms		~	Реактивный стек: 10 ms	
	Кол-во потоков	RAM		Кол-во потоков	RAM
Один известный поисковик	4.5 - 9 K	9 – 18 GB	~	300 - 600	0.6 – 1.2 GB
Один известный видеохостинг	150 – 300 K	300 – 600 GB	~	10 – 20 K	20 - 40 GB
Одна известная социальная сеть	300 – 750 K	600 GB – 1.5 TB	~	20 – 50 K	40 – 100 GB
Один известный интернет-магазин	750 K – 1.5 M	1.5 – 3 TB	~	50 – 100 K	100 – 200 GB
Один известный мессенджер	1.5 - 3 M	3 – 6 TB	~	100 – 200 K	200 – 400 GB
SaaS-стартап	15 – 1 500	30 MB – 3 GB			
Банковское приложение	150 – 7 500	300 MB – 15 GB			
Криптовалютная биржа	15 – 150 K	30 – 300 GB			

# Плановая нагрузка на разные типы сервисов

Время выполнения запроса


Блокирующий стек: 150 ms

Реактивный стек: 10 ms

	Блокирующий стек: 150 ms			~	Реактивный стек: 10 ms	
	Кол-во потоков	RAM			Кол-во потоков	RAM
Один известный поисковик	4.5 - 9 K	9 – 18 GB		~	300 - 600	0.6 – 1.2 GB
Один известный видеохостинг	150 – 300 K	300 – 600 GB		~	10 – 20 K	20 - 40 GB
Одна известная социальная сеть	300 – 750 K	600 GB – 1.5 TB		~	20 – 50 K	40 – 100 GB
Один известный интернет-магазин	750 K – 1.5 M	1.5 – 3 TB		~	50 – 100 K	100 – 200 GB
Один известный мессенджер	1.5 - 3 M	3 – 6 TB		~	100 – 200 K	200 – 400 GB
SaaS-стартап	15 – 1 500	30 MB – 3 GB		~	1 – 100	2 – 200 MB
Банковское приложение	150 – 7 500	300 MB – 15 GB		~	10 – 500	200 MB - 1 GB
Криптовалютная биржа	15 – 150 K	30 – 300 GB		~	1 – 10 K	2 – 20 GB

Я подниму 6 000 потоков и  
справлюсь с нагрузкой.





Я подниму 6 000 потоков и  
справлюсь с нагрузкой.

Мои 400 реактивных потоков будут  
выполнять работу твоих 6 000.

Хорошо, нам удалось существенно сэкономить ресурсы потоков за счёт реактивного стека.

Хорошо, нам удалось существенно сэкономить ресурсы потоков за счёт реактивного стека.

Но на самом деле, мы просто маскируем недостатки платформенных потоков за счёт уменьшения их количества.

# Недостатки платформенных потоков JVM

- Высокая стоимость потока

Под каждый поток резервируется 2 МВ оперативной памяти. Уменьшение количества потоков за счёт перехода на реактивный стек не уменьшает стоимости каждого потока.

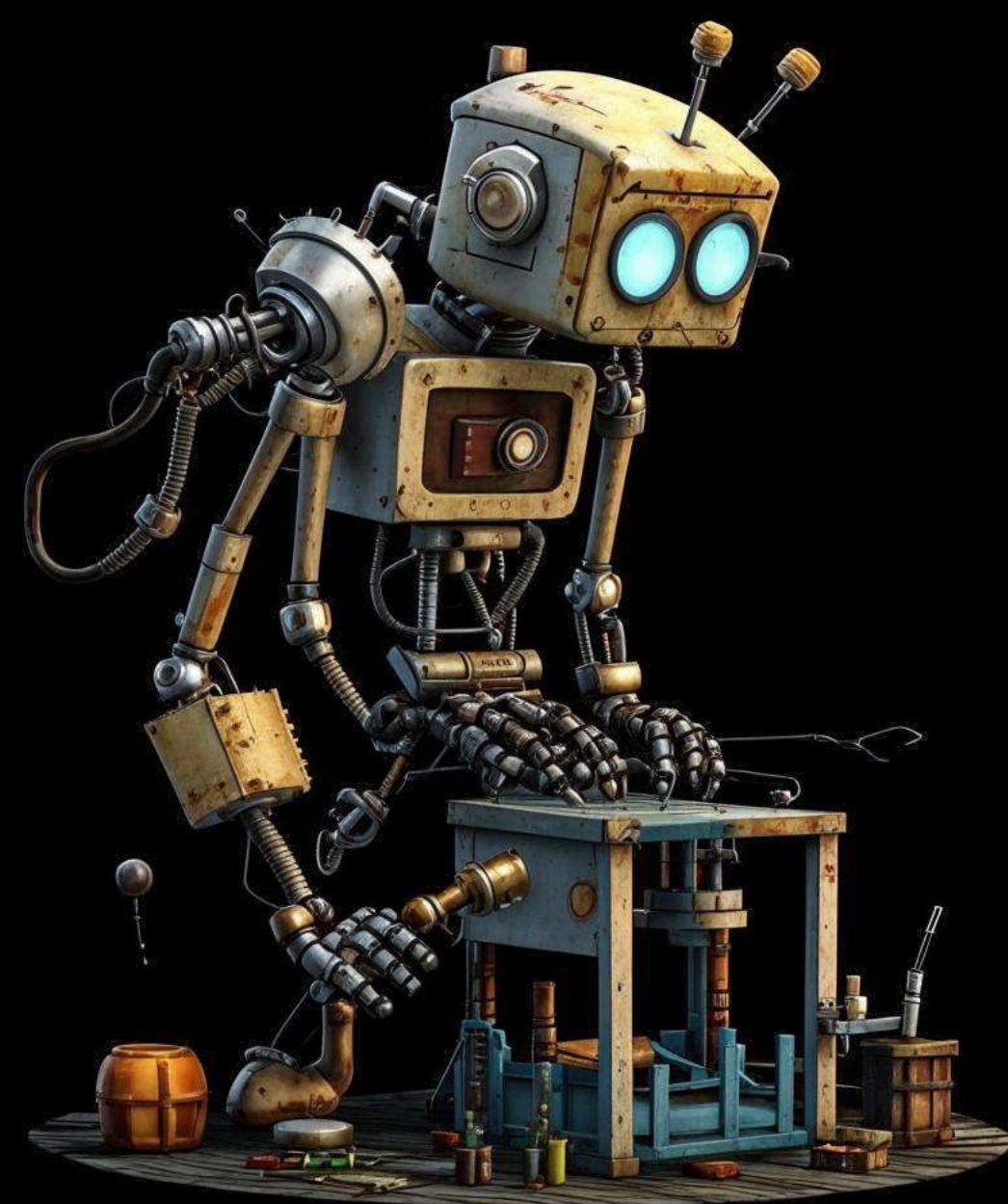


2 МВ — это по-прежнему очень дорого.

# Недостатки платформенных потоков JVM

- Высокая стоимость потока
- Вытесняющая многозадачность

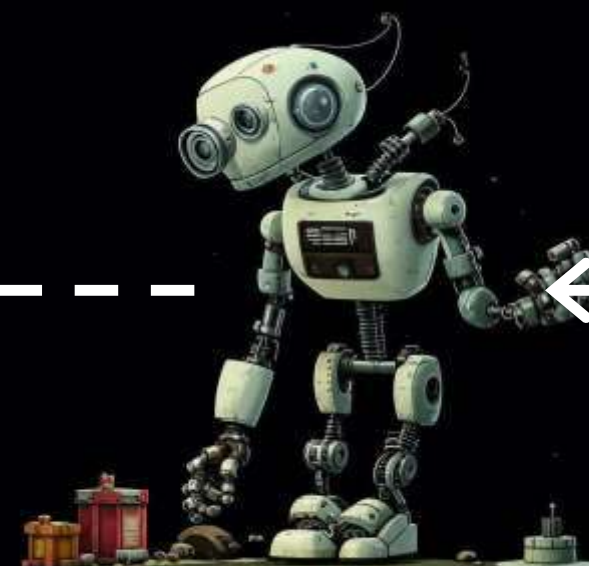
Чем больше потоков будет обслуживаться одним процессорным ядром, чем больше переключений контекста будет приходиться на одно квантовое время, выделенное конкретному потоку.



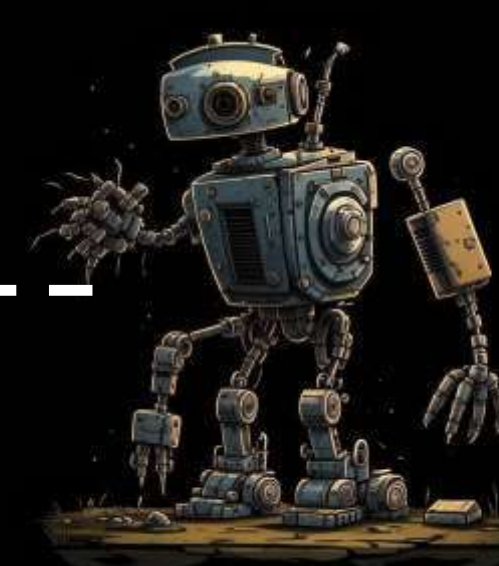
Ядро процессора



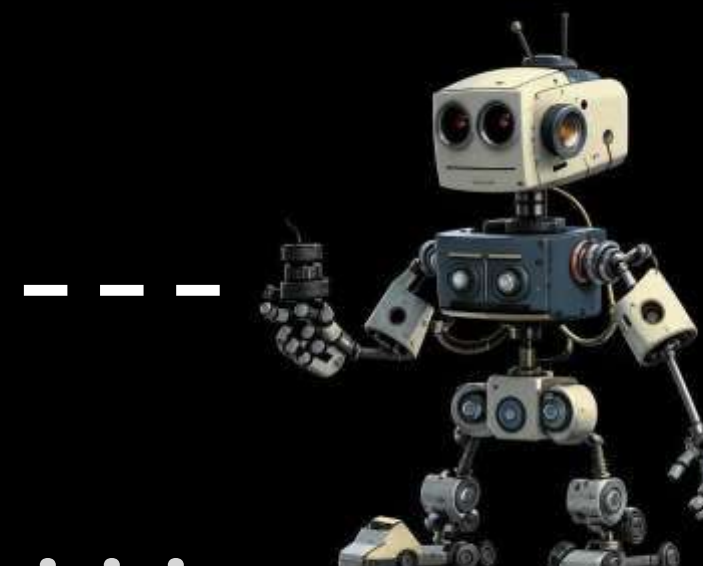
Поток 1



Поток 2

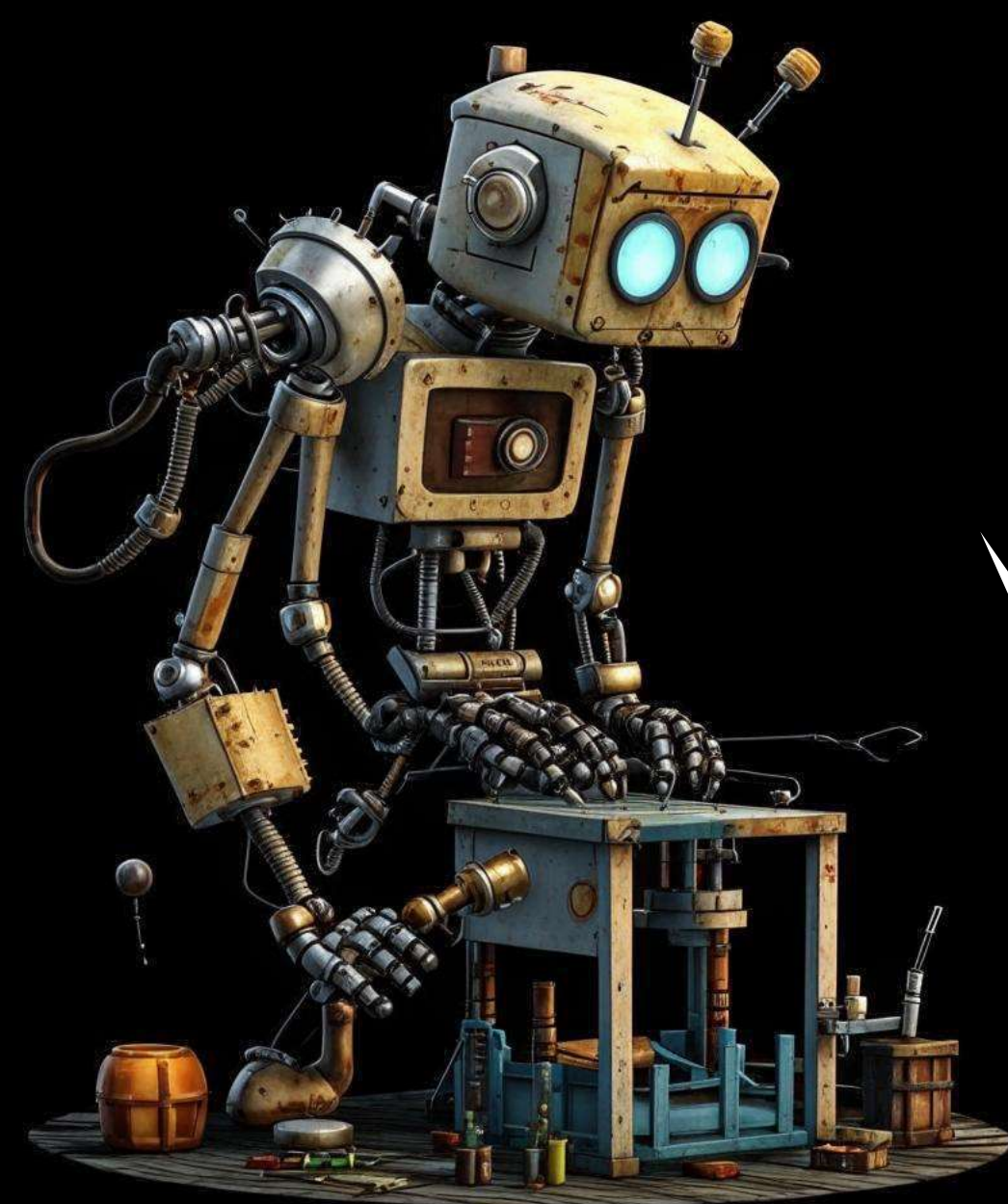


Поток 3



Поток N

При большом количестве потоков ядро просто захлебнётся на переключении  
контекста между ними.



Ядро процессора

Просто оставьте  
меня в покое

При большом количестве потоков ядро просто захлебнётся на переключении  
контекста между ними.

# Недостатки платформенных потоков JVM

- Высокая стоимость потока
- Вытесняющая многозадачность
- Ограниченная масштабируемость

Каждый платформенный поток является обёрткой над нативным потоком операционной системы. Соответственно, при создании платформенного потока на стороне JVM в операционной системе создаётся нативный поток, который платформенный поток будет использовать.

Количество потоков жёстко ограничено настройками ОС.

Приложение, которое пытается создать поток для каждой конкурентной задачи (например, HTTP-запроса), быстро упрётся в это ограничение и просто перестанет принимать запросы.

# Недостатки платформенных потоков JVM

- Высокая стоимость потока
- Вытесняющая многозадачность
- Ограниченная масштабируемость
- Сложности реактивного подхода

Разработчики в массе своей ненавидят реактивный подход за его сложность для понимания и отладки и потребность заточить весь стек под реактивный стиль.

Плюс проблемы с блокирующим кодом, работа на ограниченном количестве потоков из коробки и риск дедлоков как следствие.

- сложность кода и отладки
- проблемы с блокирующим кодом
- неоптимальная утилизация памяти

# Недостатки платформенных потоков JVM

- Высокая стоимость потока
- Вытесняющая многозадачность
- Ограниченная масштабируемость
- Сложности реактивного подхода

Разработчики в массе своей ненавидят реактивный подход за его сложность для понимания и отладки и потребность заточить весь стек под реактивный стиль.

Плюс проблемы с блокирующим кодом, работа на ограниченном количестве потоков из коробки и риск дедлоков как следствие.

- сложность кода и отладки
- проблемы с блокирующим кодом
- неоптимальная утилизация памяти



Место для отдельного доклада

# Недостатки платформенных потоков JVM

- Высокая стоимость потока
- Вытесняющая многозадачность
- Ограниченная масштабируемость
- Сложности реактивного подхода

Мы просто маскируем эти  
недостатки



Платформенные потоки не подходят для параллельного выполнения большого количества задач.

Платформенные потоки не подходят для параллельного выполнения большого количества задач.

Платформенные потоки не должны использоваться для решения бизнес-задач.

Платформенные потоки не подходят для параллельного выполнения большого количества задач.

Платформенные потоки не должны использоваться для решения бизнес-задач.

Тогда что?

Кто я?



Джавист

Кто я?



Джавист



Котлинист

Виртуальные потоки.

Виртуальные потоки.

Почему именно они?

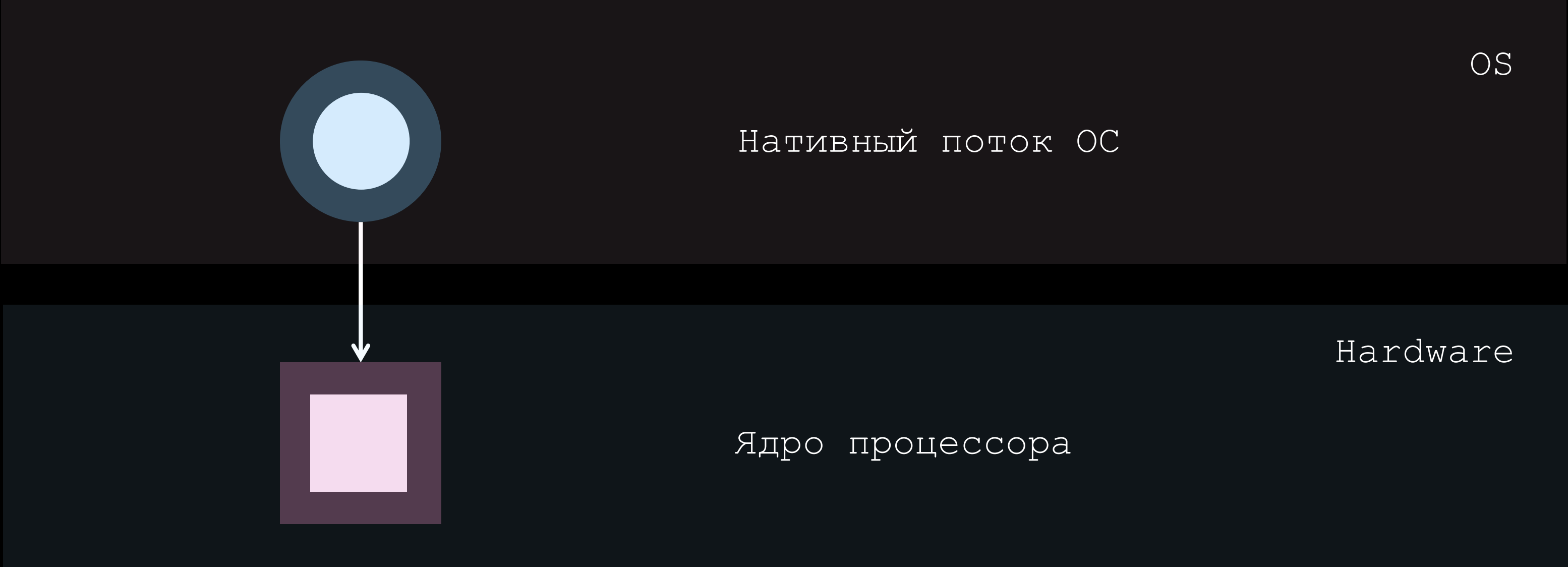
## Виртуальные потоки: что это?

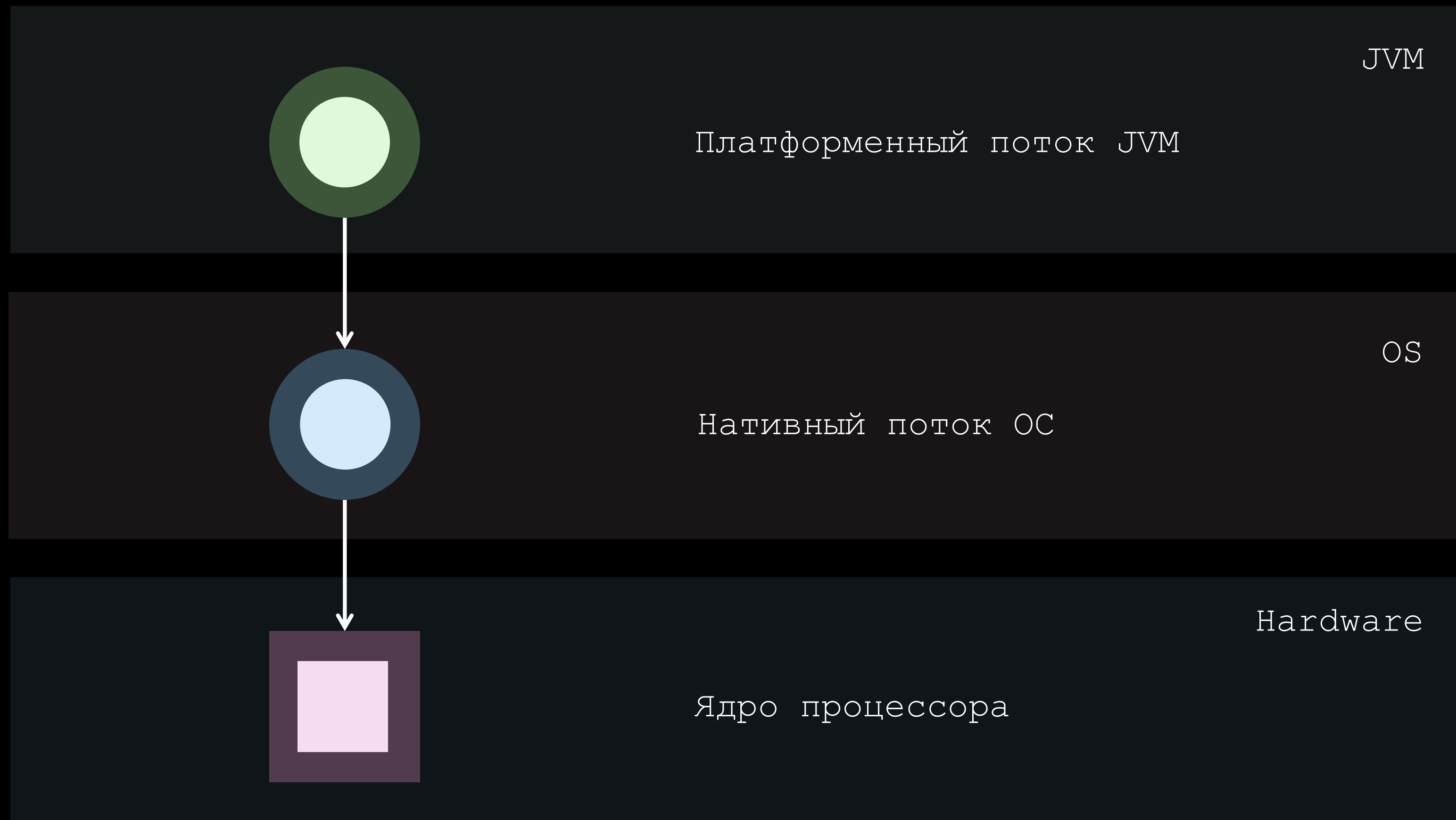
- Виртуальные потоки являются объектами JVM и никак не связаны с ядрами процессора.

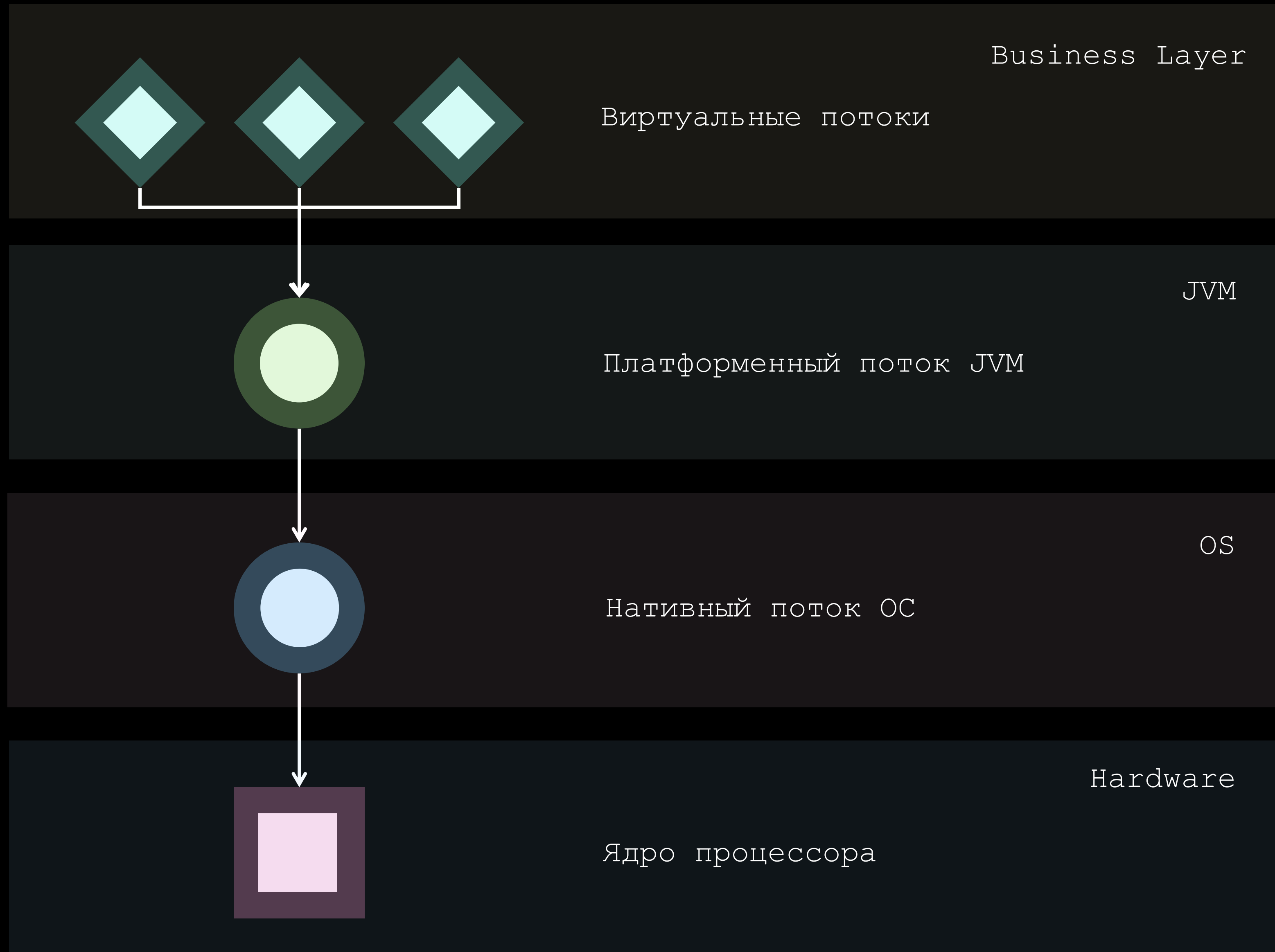
В отличие от платформенных потоков, виртуальные потоки никак не связаны с нативными потоками операционной системы.

Hardware

Ядро процессора





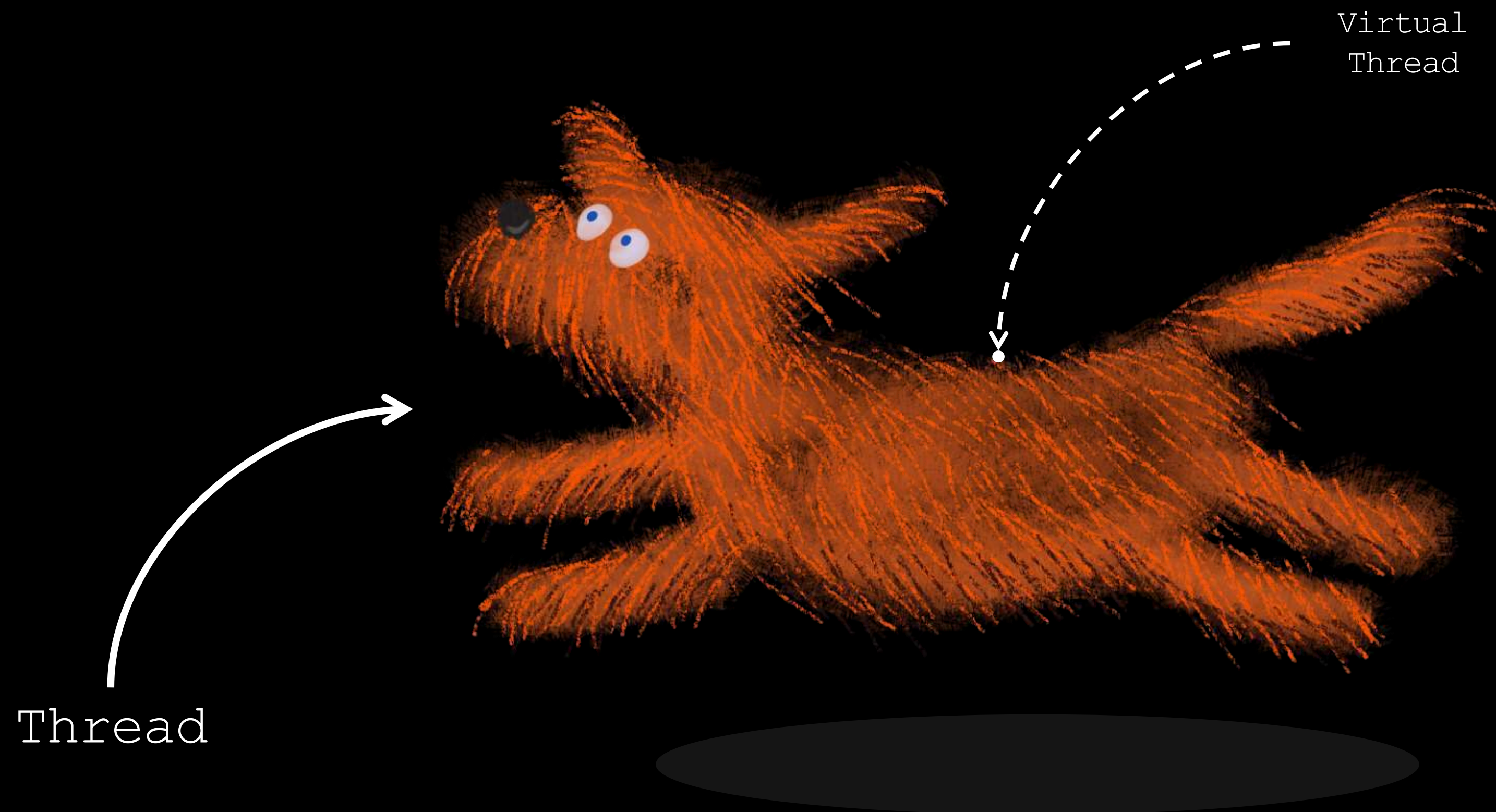


## Виртуальные потоки: что это?

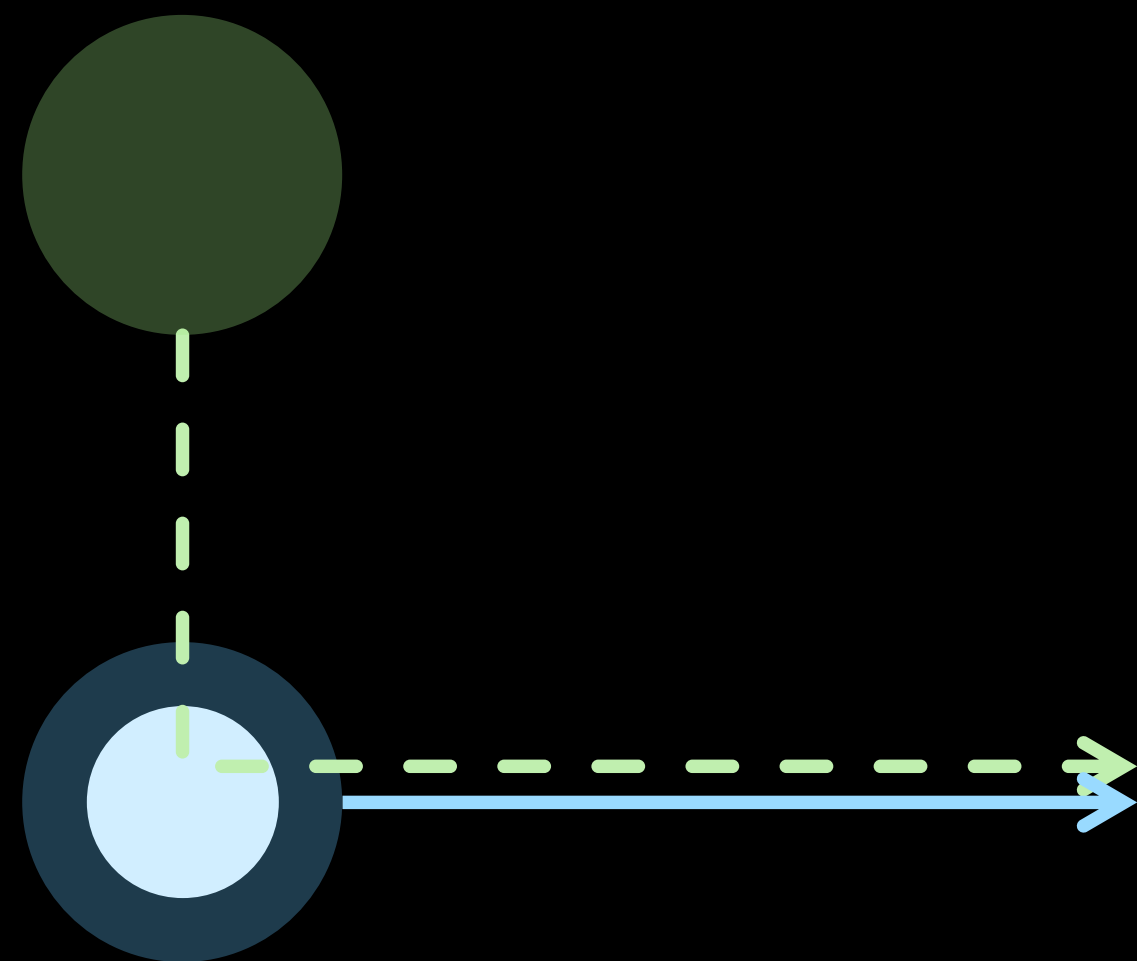
- Виртуальные потоки являются объектами JVM и никак не связаны с ядрами процессора.
- Виртуальные потоки используют платформенные потоки для своей работы.

Размер виртуального потока составляет 200 байт.

Виртуальные потоки не имеют собственного стека в привычном понимании. Для выполнения виртуальные потоки используют стек платформенного потока. А свои данные они хранят в куче.



Virtual Thread 1

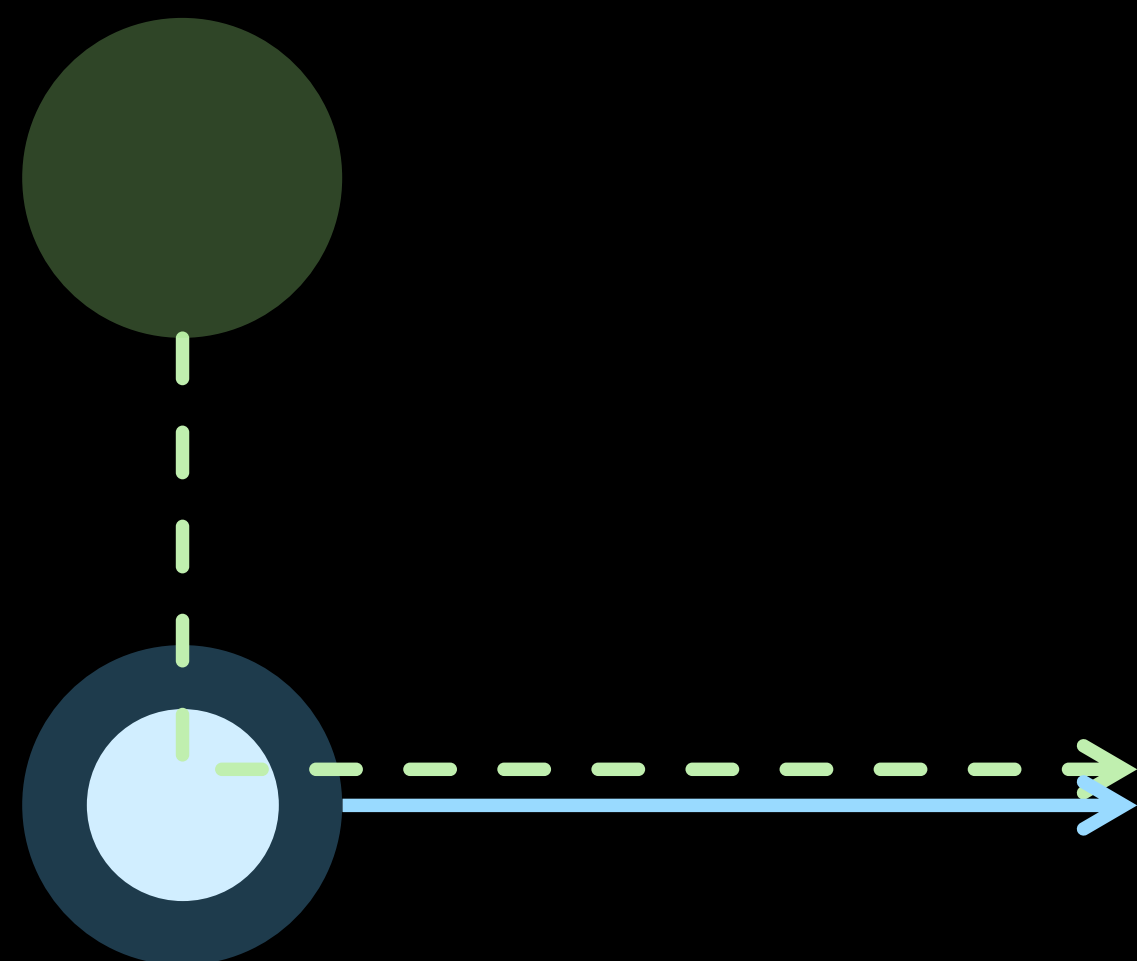


Career Thread 1

Шаг 1. Монтаж и исполнение.

Virtual Thread монтируется на носитель (Career Thread), после чего использует его для выполнения своих задач.

Virtual Thread 1



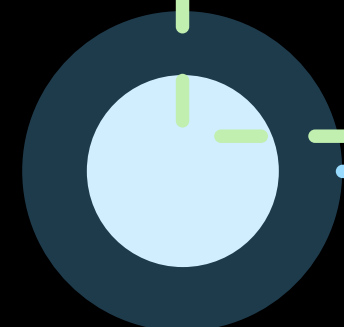
Career Thread 1

Платформенный поток предоставляет виртуальному потоку доступ к мощностям процессора.

Шаг 1. Монтаж и исполнение.

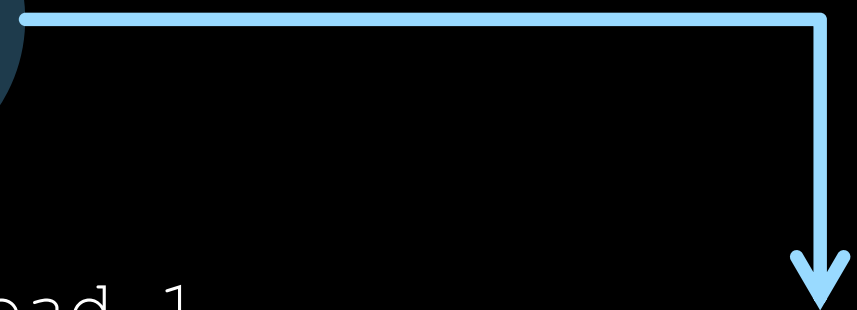
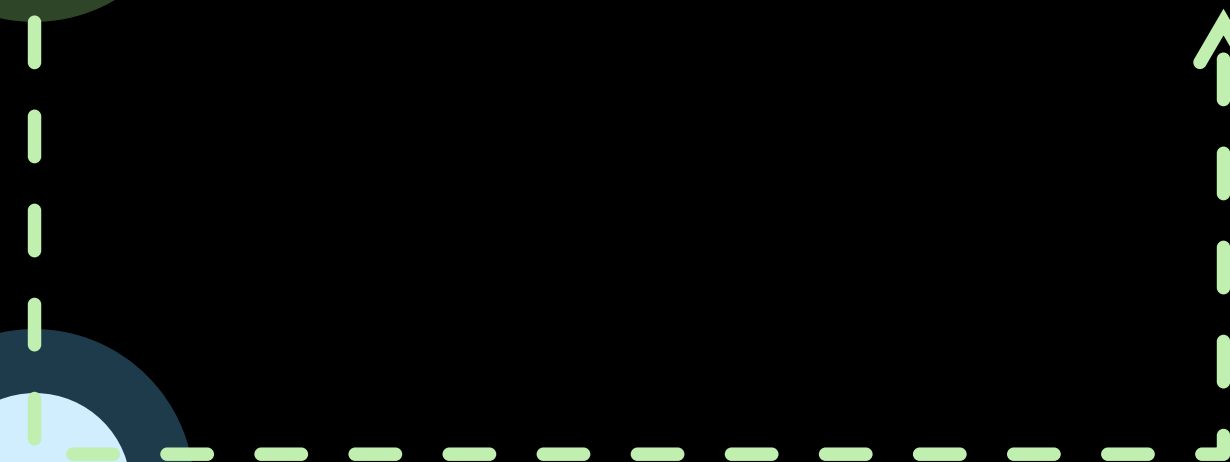
Virtual Thread монтируется на носитель (Career Thread), после чего использует его для выполнения своих задач.

Virtual Thread 1



Career Thread 1

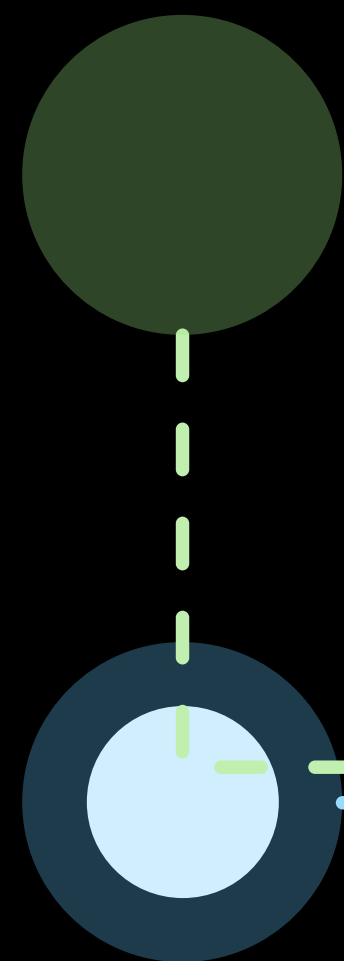
Java Heap



Шаг 2. I/O операция.

JVM определяет, что происходит I/O операция. Виртуальный поток демонтируется с платформенного потока.

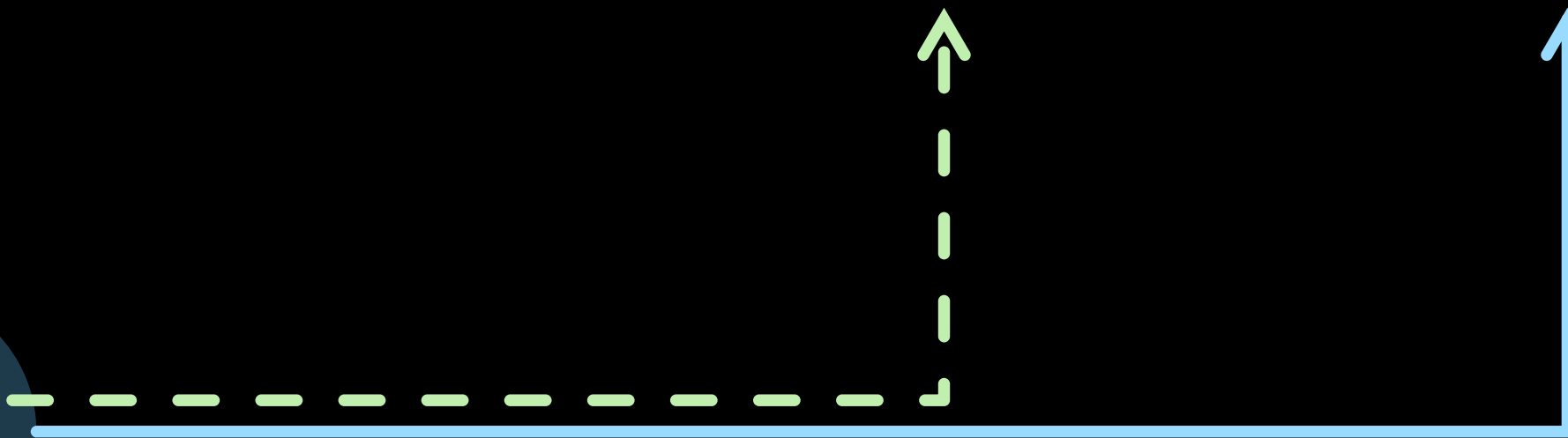
Virtual Thread 1



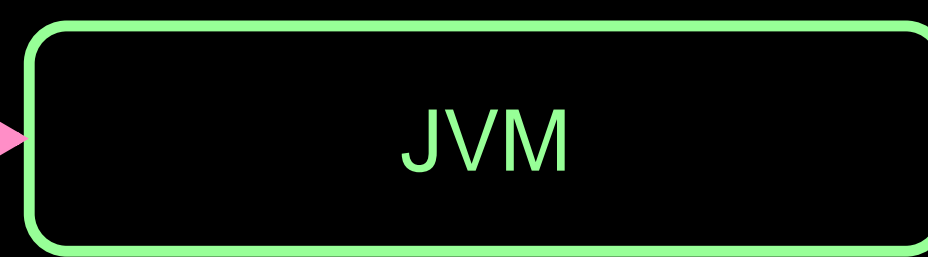
Java Heap



Thread Pool



Career Thread 1

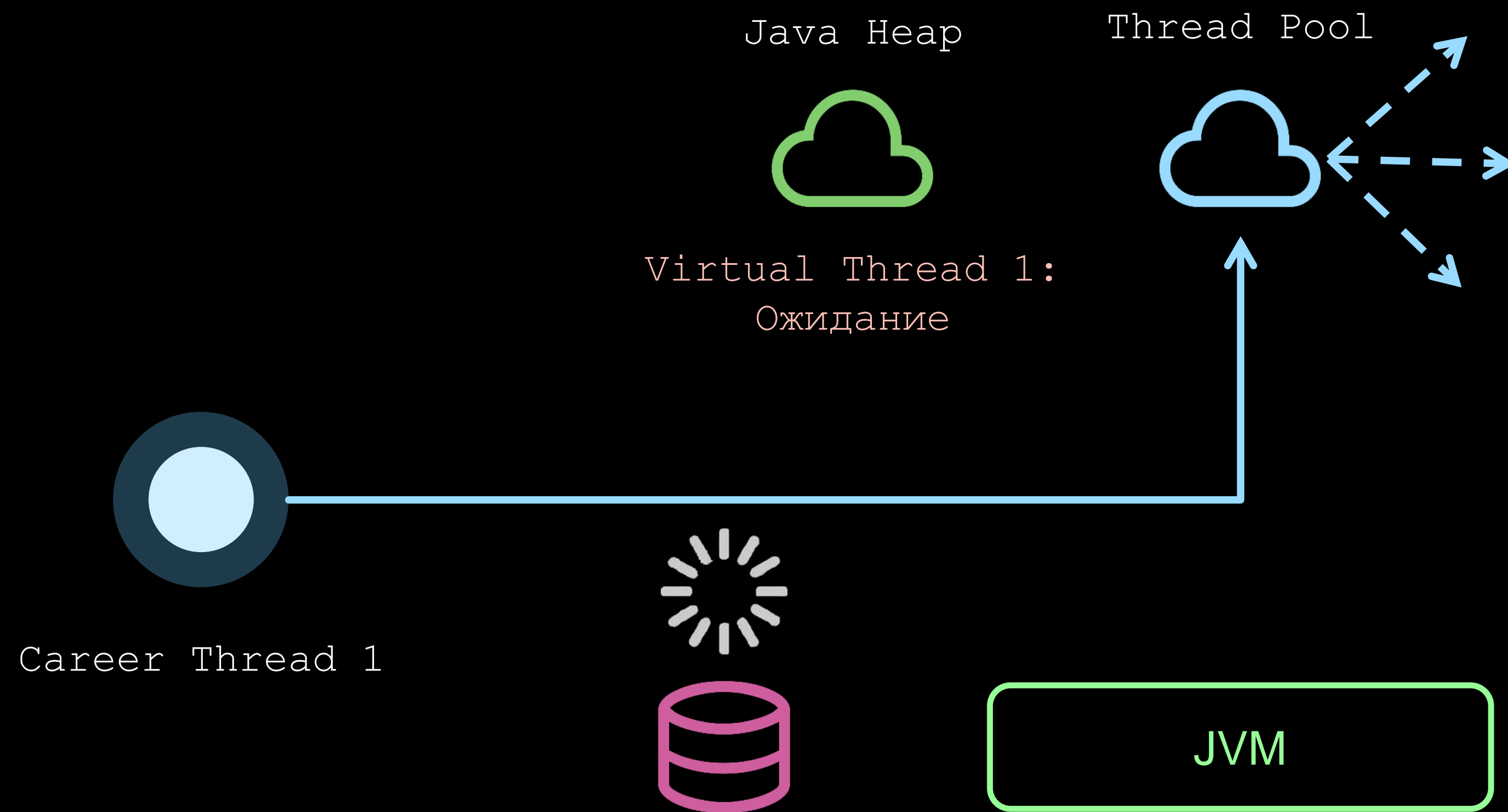


Шаг 3. Unmounting.

I/O запрос регистрируется в системе.

JVM переходит к использованию асинхронных механизмов ОС.

Поток высвобождается и возвращается в Thread Pool.

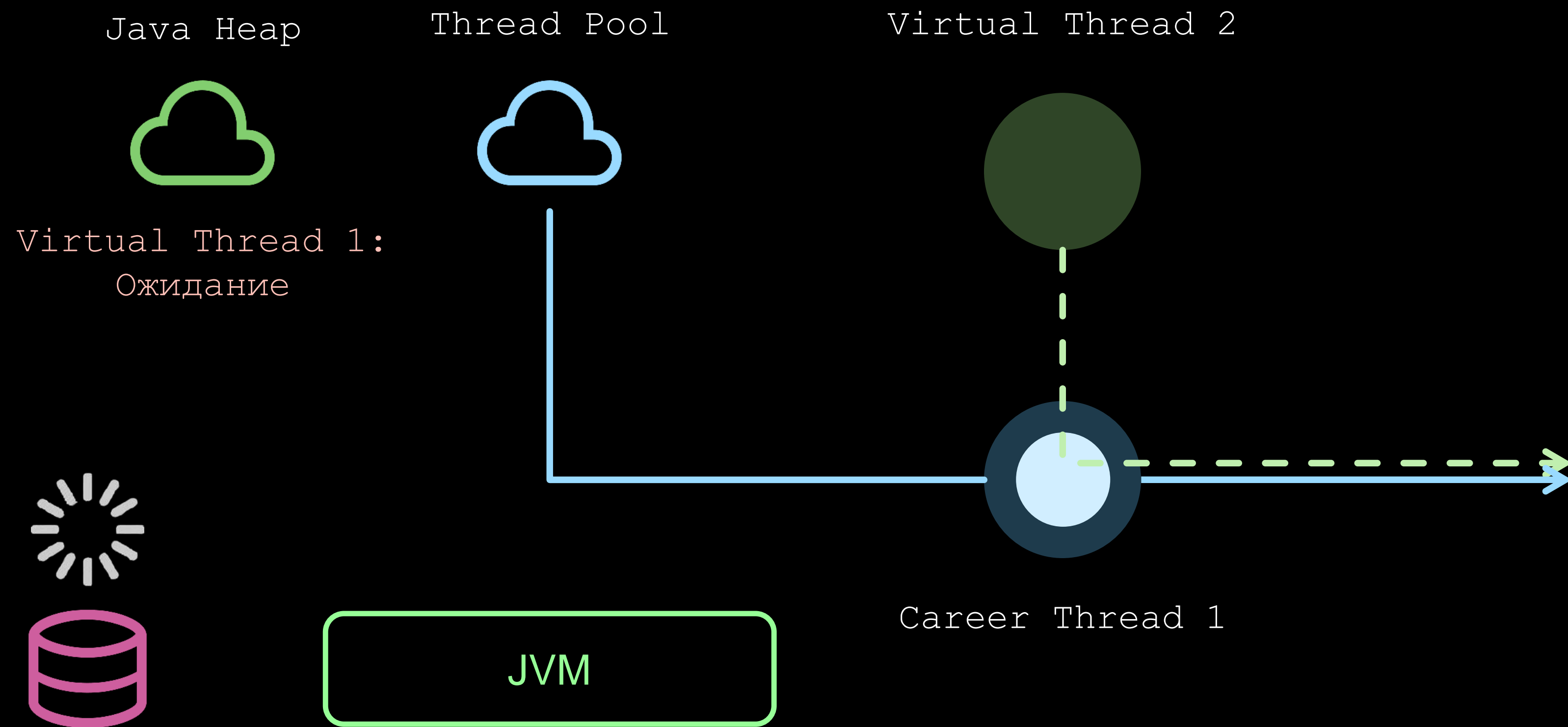


Шаг 3. Ожидание завершения операции.

Поток освобождается для других задач.

JVM ожидает выполнения операции ввода-вывода.

Виртуальный поток помечается как «ожидающий».



Шаг 1а. Монтаж и исполнение.

Любой другой поток монтируется на поток-носитель и использует его для выполнения своих задач.

Java Heap



Thread Pool



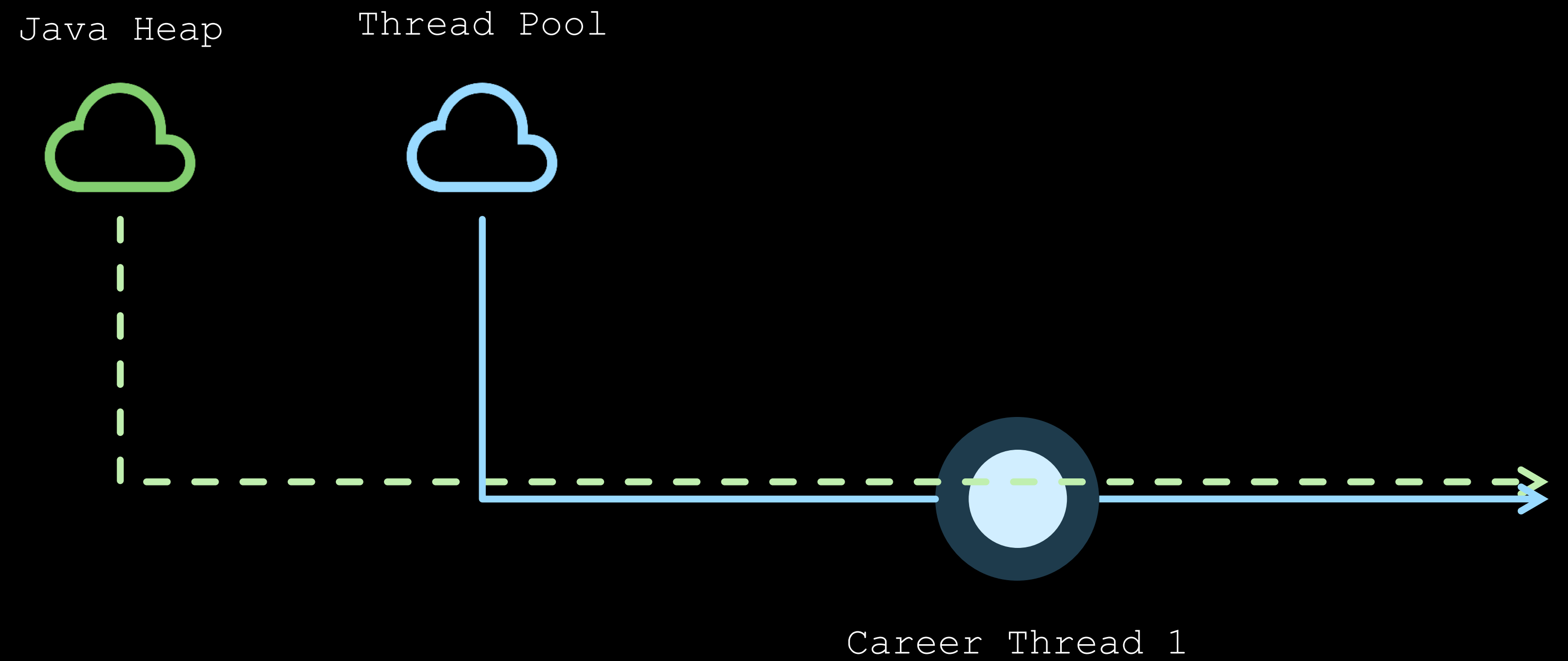
Virtual Thread 1:  
Готов к исполнению



Шаг 4. Завершение I/O операции.

Когда I/O операция завершается, JVM получает уведомление.

Планировщик виртуальных потоков помечает виртуальный поток как готовый к исполнению и виртуальный поток начинает конкурировать за носитель.



Шаг 5. Продолжение выполнения.

Когда становится доступен поток-носитель, виртуальный поток монтируется на него и продолжает исполнение.

Виртуальные потоки и платформенные потоки выполняют  
каждый свои задачи и не зависят друг от друга.

Виртуальные потоки и платформенные потоки выполняют каждый свои задачи и не зависят друг от друга.

При этом, платформенные потоки не выполняют бизнес-задач. Их выполняют виртуальные потоки.

Виртуальные потоки и платформенные потоки выполняют каждый свои задачи и не зависят друг от друга.

При этом, платформенные потоки не выполняют бизнес-задач. Их выполняют виртуальные потоки.

Платформенные потоки выступают как носители и предоставляют доступ к мощностям процессора.

## Виртуальные потоки: что это?

- Виртуальные потоки являются объектами JVM и никак не связаны с ядрами процессора.
- Виртуальные потоки используют платформенные потоки для своей работы.
- Количество виртуальных потоков не имеет строгого лимита.

Количество виртуальных потоков не ограничено ни лимитами операционной системы, ни лимитами JVM.

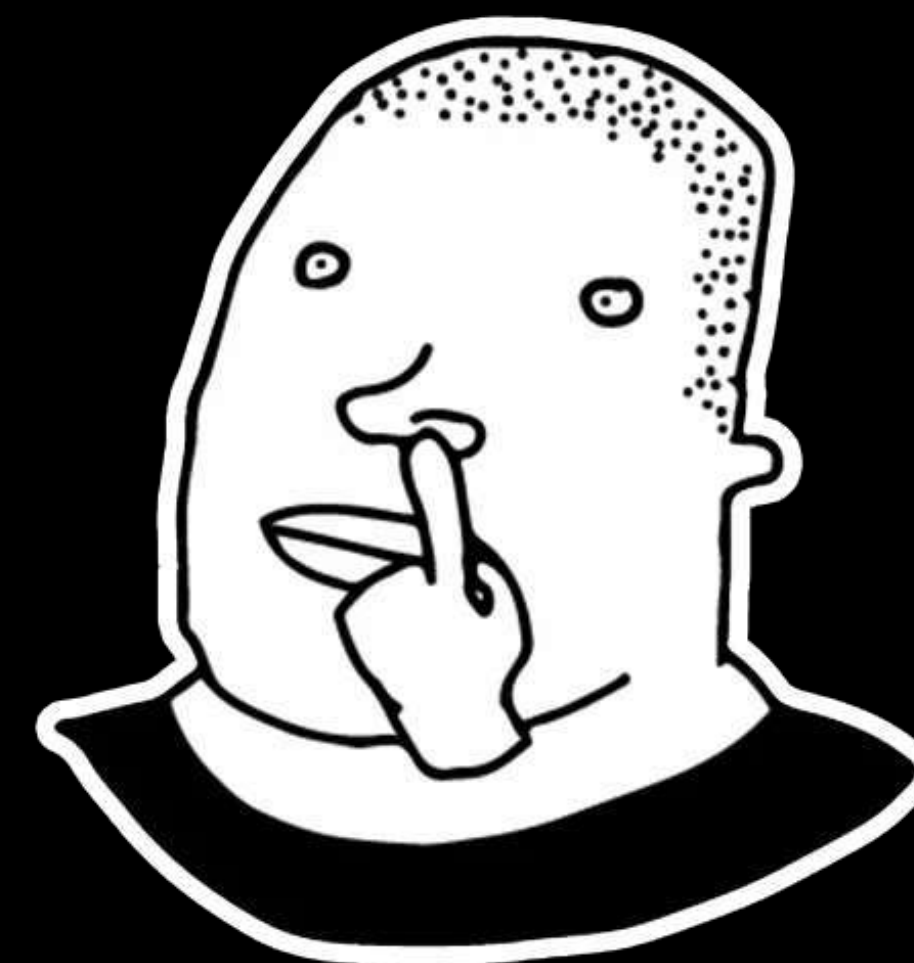
## Виртуальные потоки: что это?

- Виртуальные потоки являются объектами JVM и никак не связаны с ядрами процессора.
- Виртуальные потоки используют платформенные потоки для своей работы.
- Количество виртуальных потоков не имеет строгого лимита.

Количество виртуальных потоков не ограничено ни лимитами операционной системы, ни лимитами JVM.

При этом, один виртуальный поток занимает всего 200 байт. Мы можем создать миллионы виртуальных потоков без вреда для системы.

Действительно ли можно создать миллион виртуальных потоков?



# Виртуальные потоки: производительность

# Виртуальные потоки: производительность

Параллельное выполнение

100К

## Виртуальные потоки: производительность

Параллельное выполнение

Regular Threads

100К

54,649 ms

## Виртуальные потоки: производительность

Параллельное выполнение

Regular Threads

Virtual Threads

100К

54,649 ms

9,477 ms

## Виртуальные потоки: производительность

Параллельное выполнение	Regular Threads	Virtual Threads
100К	54,649 ms	9,477 ms
200К		

## Виртуальные потоки: производительность

Параллельное выполнение	Regular Threads	Virtual Threads
100К	54,649 ms	9,477 ms
200К	98,614 ms	12,260 ms

## Виртуальные потоки: производительность

Параллельное выполнение	Regular Threads	Virtual Threads
100K	54,649 ms	9,477 ms
200K	98,614 ms	12,260 ms



Regular threads VS Virtual threads

## Виртуальные потоки: производительность

Параллельное выполнение	Regular Threads	Virtual Threads	Coroutines
100K	54,649 ms	9,477 ms	5,302 ms
200K	98,614 ms	12,260 ms	5,415 ms



Regular threads VS Virtual threads

## Виртуальные потоки: производительность

Параллельное выполнение	Regular Threads	Virtual Threads	Coroutines
100K	54,649 ms	9,477 ms	5,302 ms
200K	98,614 ms	12,260 ms	5,415 ms



Regular threads VS Virtual threads



Virtual threads VS Coroutines

Подведём итог.

Что нужно сделать, чтобы нам не было стыдно в 2k25 перед нашим приложением?

Что нужно сделать, чтобы нам не было стыдно в 2k25 перед нашим приложением?

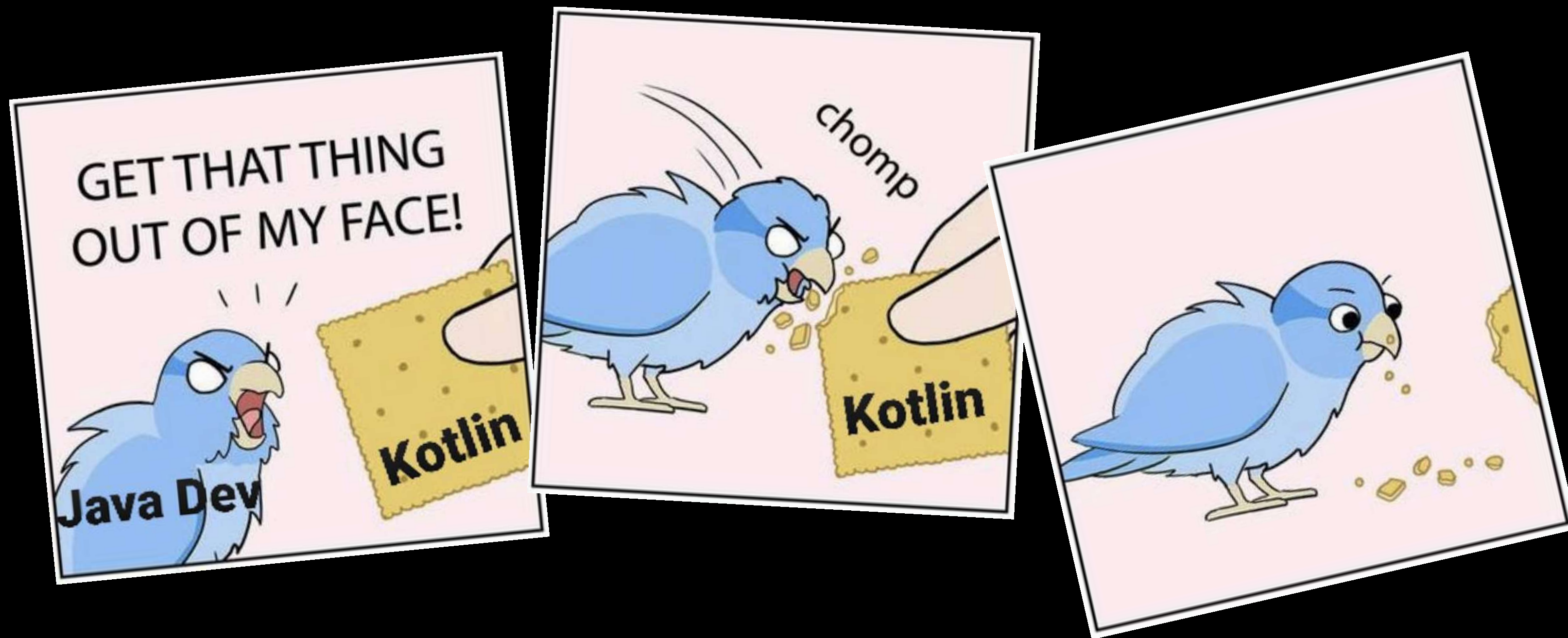
1. Отказаться от блокирующего стека, если вы ещё этого не сделали.

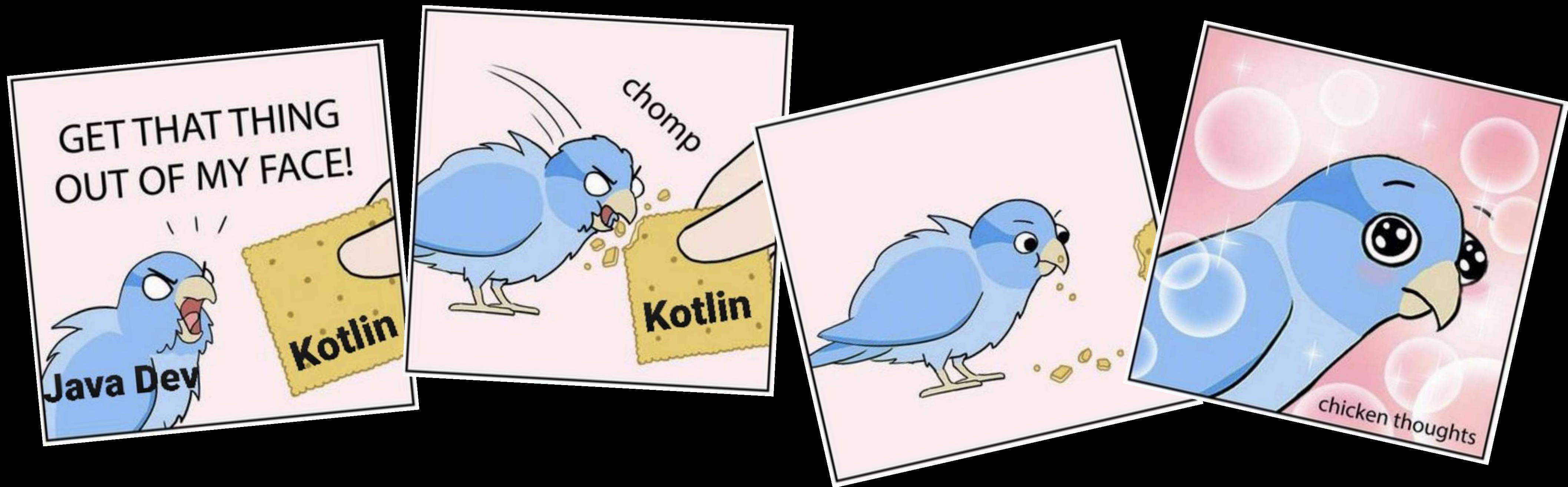
Что нужно сделать, чтобы нам не было стыдно в 2k25 перед нашим приложением?

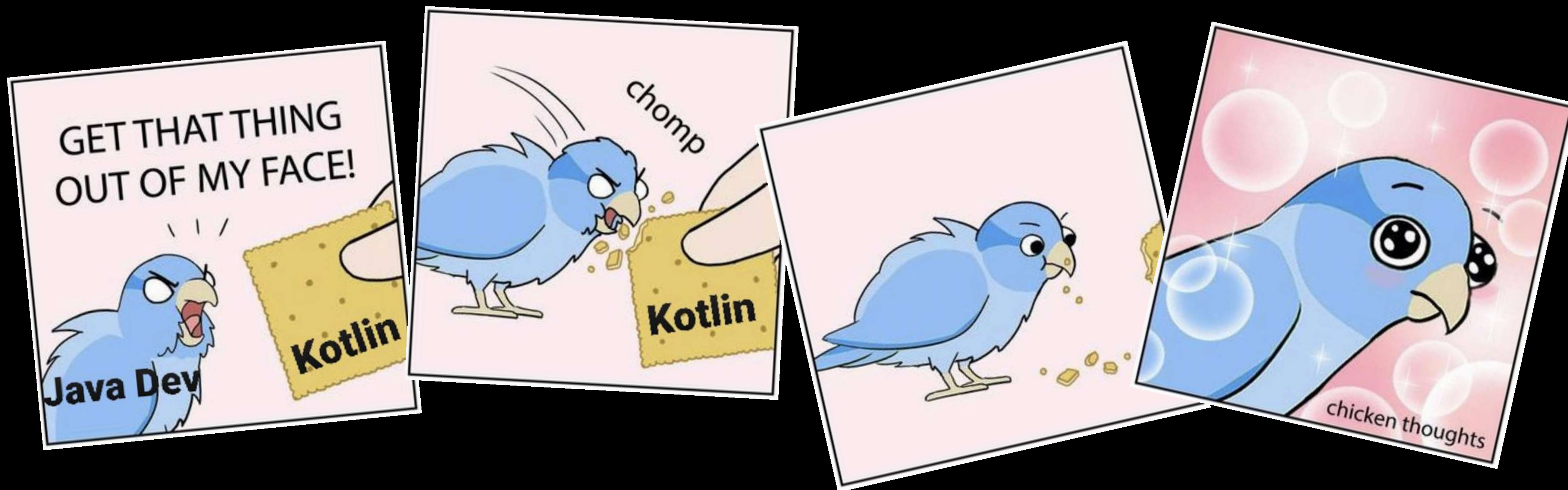
1. Отказаться от блокирующего стека, если вы ещё этого не сделали.
2. Перейти на виртуальные потоки, если у вас Java, и на корутины, если у вас Kotlin.



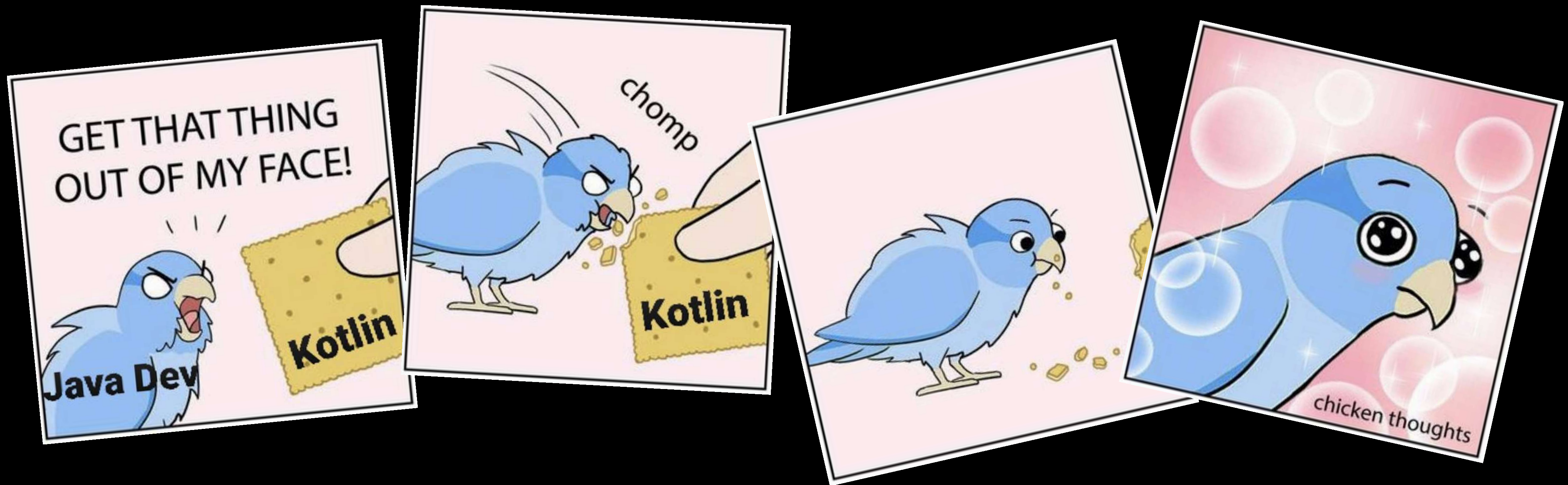








Это Kotlin, детка.



Kodee

Это Kotlin, детка.



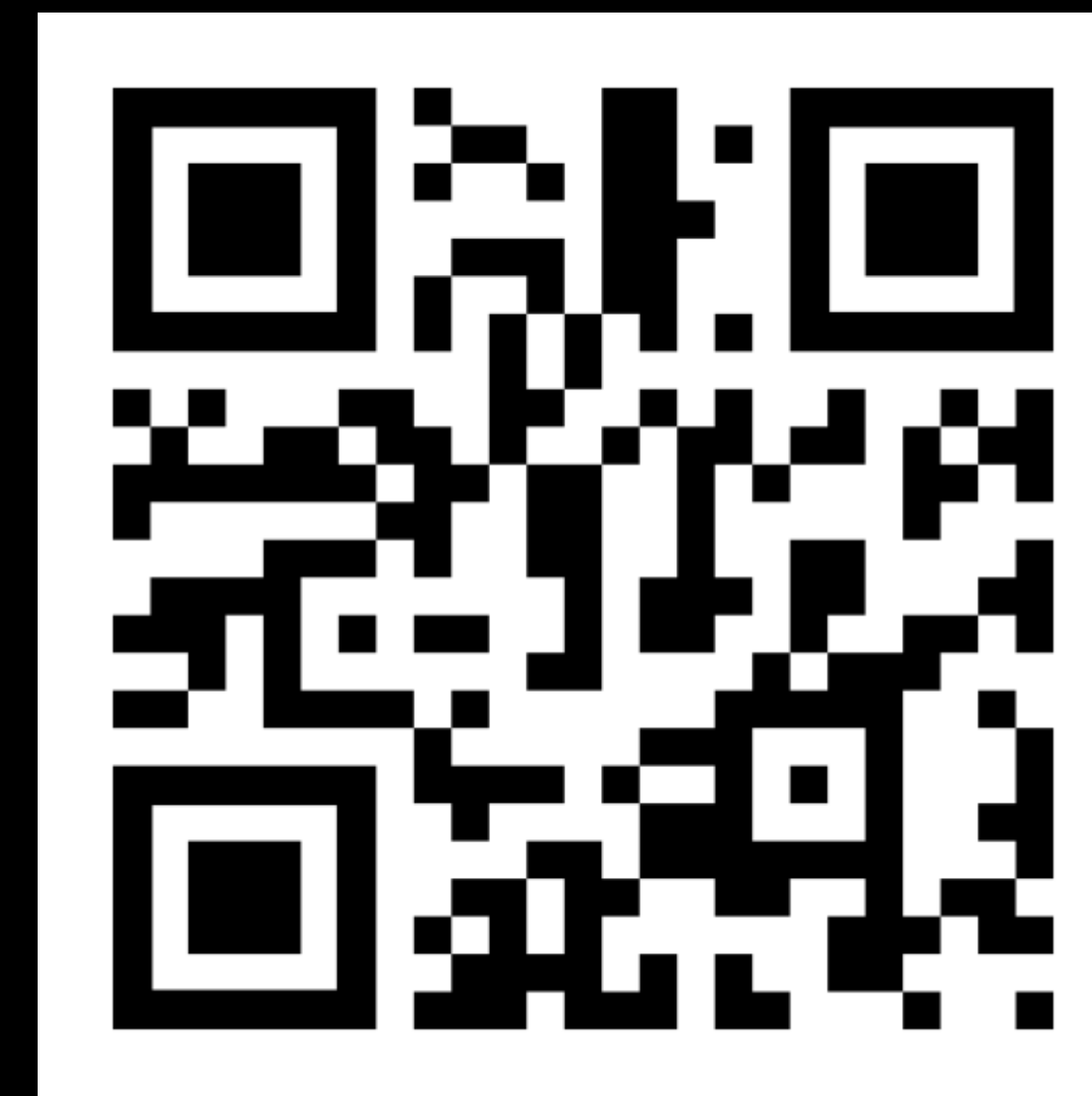
# Спасибо!



Профиль на Хабре

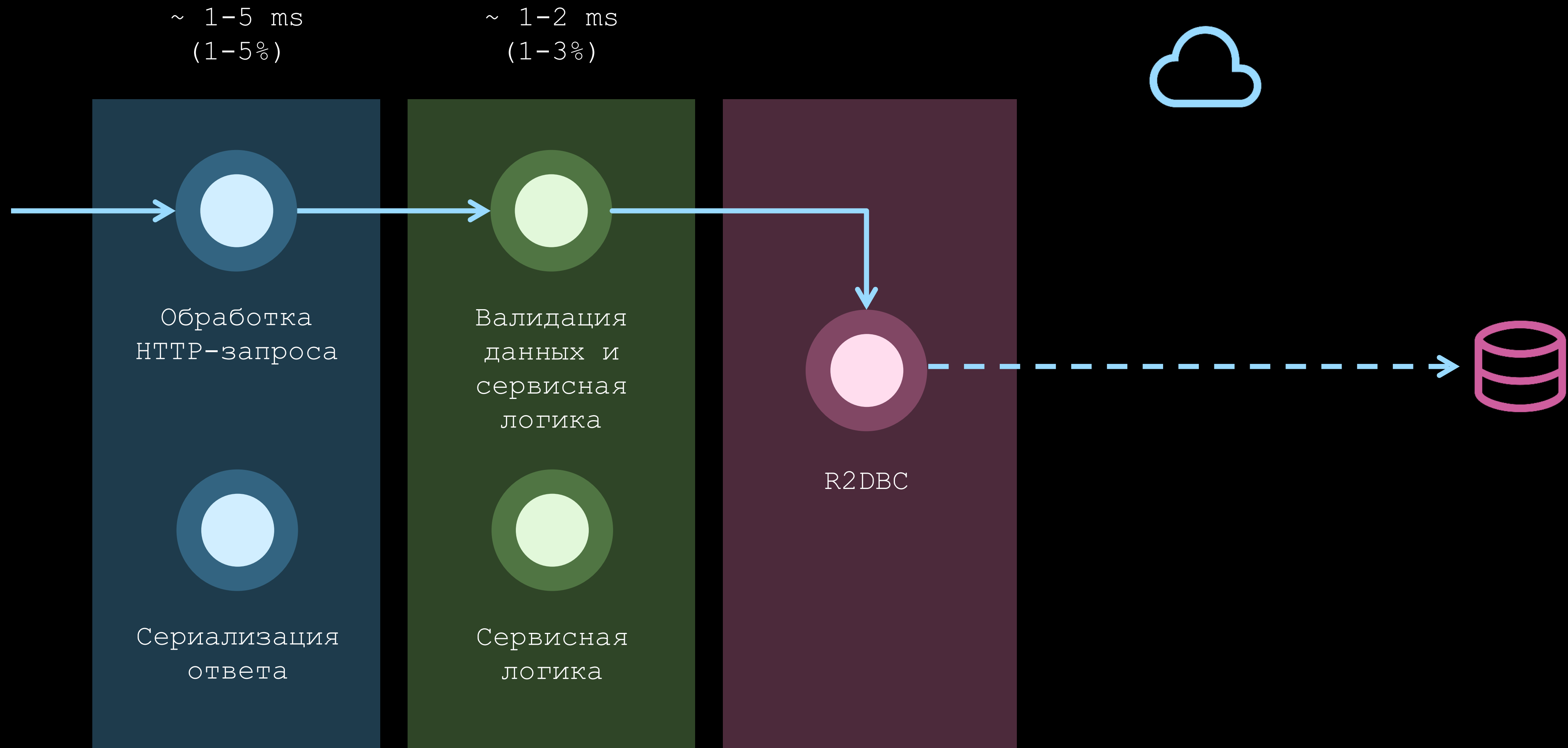


Канал в Telegram

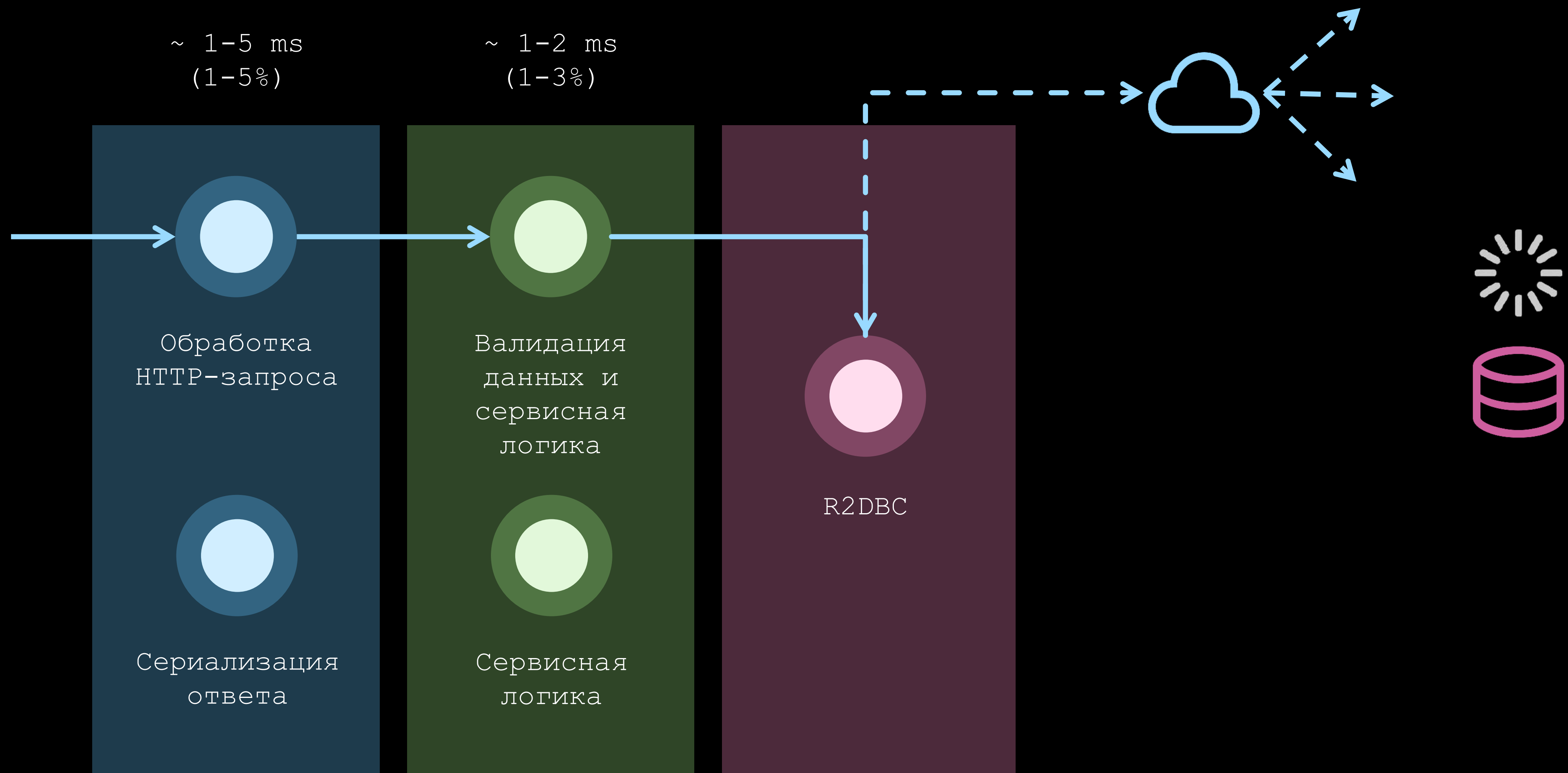


Личный сайт

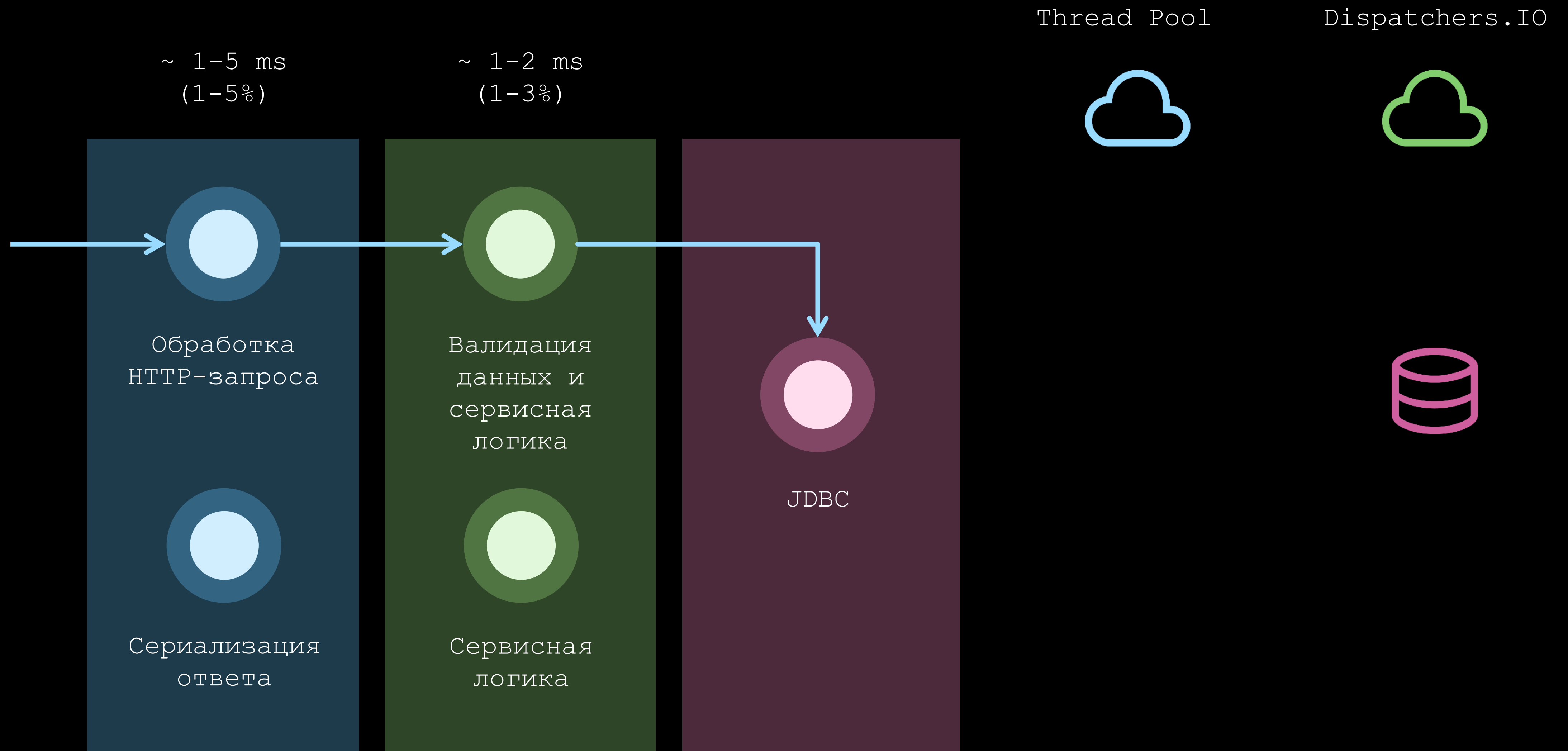
Корутины: операции ввода / вывода при неблокирующем стеке.



Корутины: операции ввода / вывода при неблокирующем стеке.



# Корутины: как они помогут нам с блокирующими операциями ввода-вывода?



## Диспетчеры корутин

Dispatchers.MAIN



Работа с UI  
(Android)

Dispatchers.IO



Блокирующие I/O  
операции

Dispatchers.Default



CPU-интенсивные  
задачи

Dispatchers.Unconfined



Без привязки к  
конкретному пулу

# Диспетчеры корутин

Dispatchers.MAIN



Работа с UI  
(Android)



Основной поток

Dispatchers.IO



Блокирующие I/O  
операции



До 64 потоков

Dispatchers.Default



CPU-интенсивные  
задачи



По количеству  
ядер

Dispatchers.Unconfined

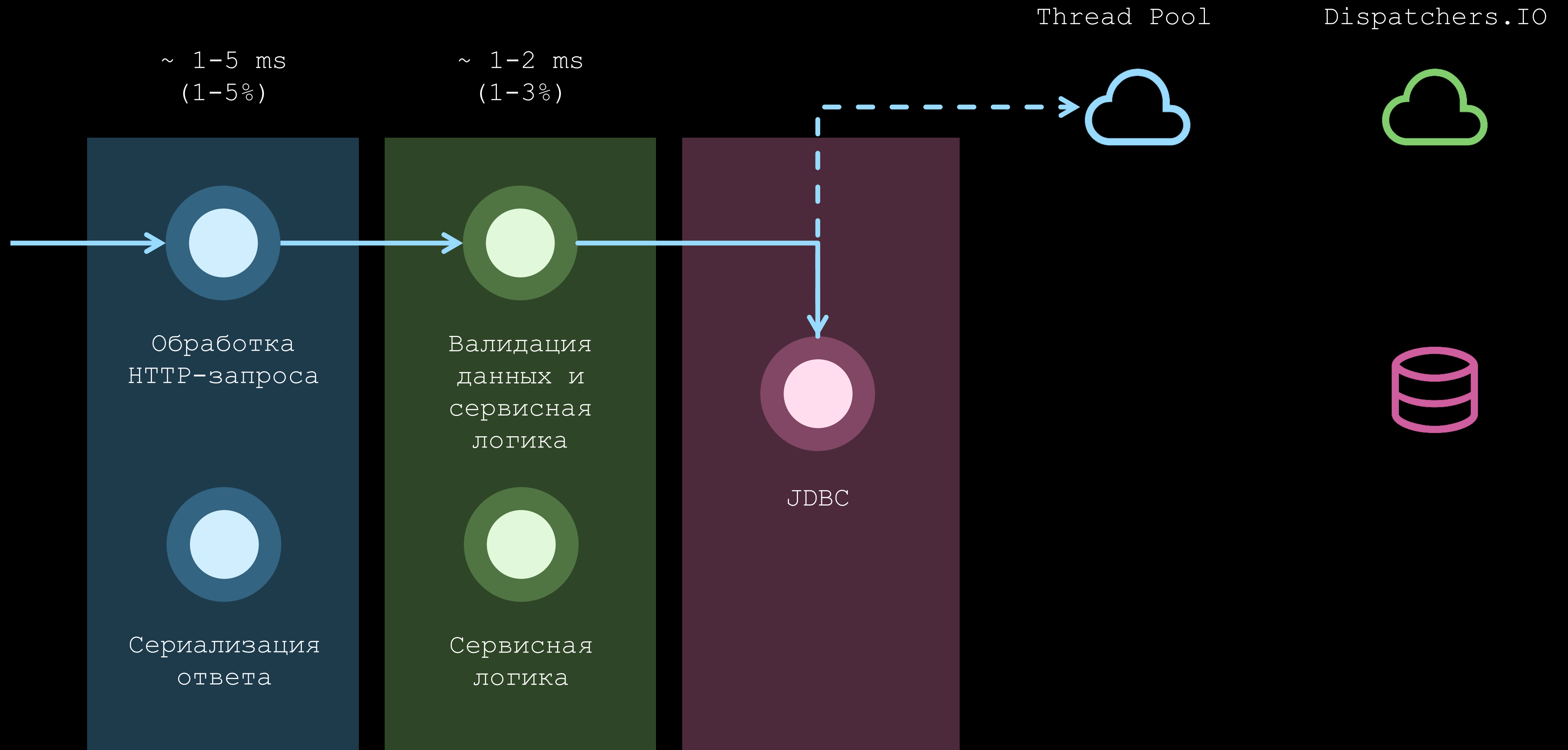


Без привязки к  
конкретному пулу

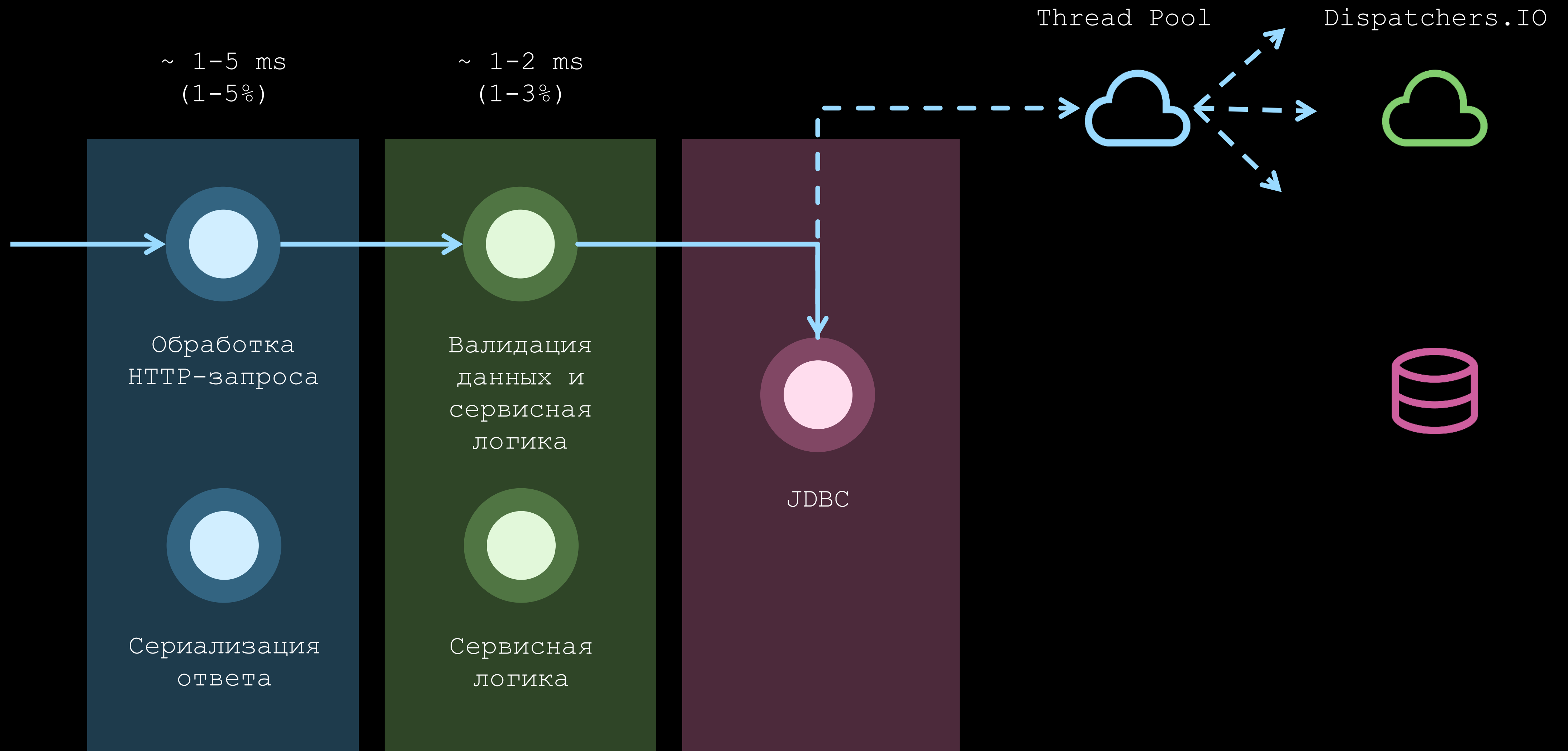


Любой поток

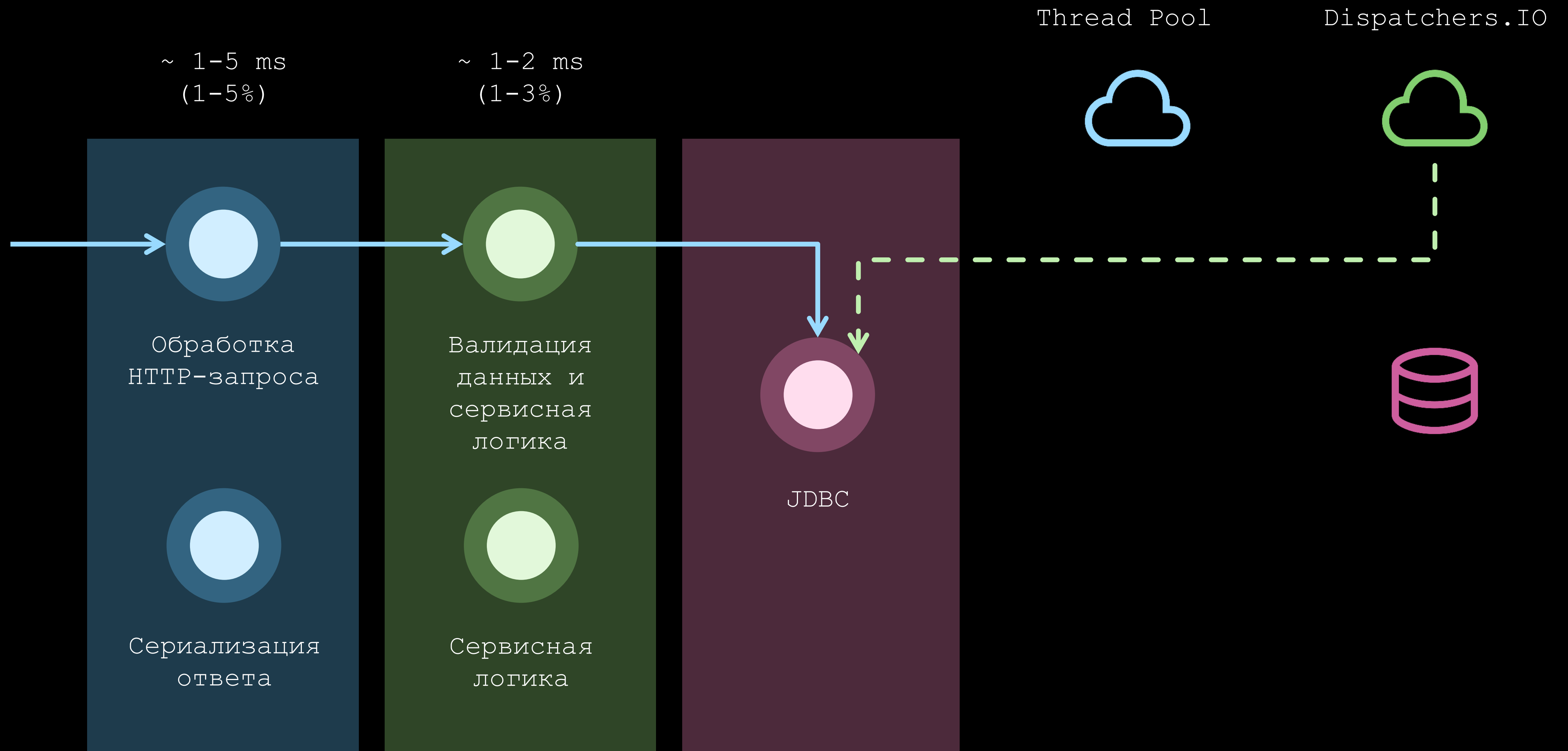
# Корутины: как они помогут нам с блокирующими операциями ввода-вывода?



# Корутины: как они помогут нам с блокирующими операциями ввода-вывода?



# Корутины: как они помогут нам с блокирующими операциями ввода-вывода?



# Корутины: как они помогут нам с блокирующими операциями ввода-вывода?

