

THREAD WARS: USER-LEVEL THREADING STRIKES BACK



DISCLAIMERS

1. В этой лекции мы любим и ценим императивщину
(не отрицая наличие и плюсы других парадигм)

DISCLAIMERS

1. В этой лекции мы любим и ценим императивщину
(не отрицая наличие и плюсы других парадигм)
2. Т.к. весь курс был склонен к Java/JVM, сегодня мы
тоже сделаем акцент на этом домене
(хотя и вспомним о других мирах)

DISCLAIMERS

1. В этой лекции мы любим и ценим императивщину (не отрицая наличие и плюсы других парадигм)
2. Т.к. весь курс был склонен к Java/JVM, сегодня мы тоже сделаем акцент на этом домене (хотя и вспомним о других мирах)
3. Нам интересны и детали реализации (системщина) и дизайн языков! Впрочем, как и всегда.

A long time ago in a galaxy far,
far away....

THE PROBLEM

Хотим писать код последовательный:

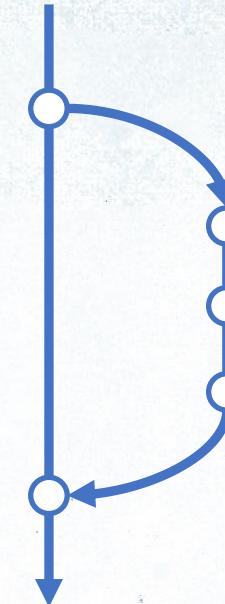
```
int v = foo();  
v += 2;  
bar(v);
```



THE PROBLEM

Хотим писать код параллельный:

```
Thread t = new Thread(() -> {  
    int v = foo();  
    v += 2;  
    bar(v);  
});  
t.start();  
...  
t.join();
```



THE PROBLEM

А как реализовать `java.lang.Thread`?

THE PROBLEM

А как реализовать `java.lang.Thread`?

А чего вообще хотим от этого класса?

THE PROBLEM

А как реализовать `java.lang.Thread`?

Чего вообще хотим от этого класса:

1. Чтобы тела потоков исполнялись «параллельно»
 - о Желательно «честно», т.е. равномерно деля процессорное время

THE PROBLEM

А как реализовать `java.lang.Thread`?

Чего вообще хотим от этого класса:

1. Чтобы тела потоков исполнялись «параллельно»
 - о Желательно «честно», т.е. равномерно деля процессорное время
2. При этом у каждого свои локальные переменные
3. При этом был доступ к общей памяти

THE PROBLEM

А как реализовать `java.lang.Thread`?

Чего вообще хотим от этого класса:

1. Оркестратор (**планировщик**), который будет решать, кому исполняться,
2. Система управления **стеками**,
3. Остаемся в рамках одного **процесса**



THE PROBLEM

А как реализовать `java.lang.Thread`?

Очевидное решение – встать на плечи гигантов!

`java.lang.Thread` <=> OS-thread
(1:1 схема, один к одному)



THE PROBLEM

А как реализовать `java.lang.Thread`?

Чего вообще хотим от этого класса:

1. Оркестратор (**планировщик**), который будет решать, кому исполняться,
2. Система управления **стеками**,
3. Остаемся в рамках одного **процесса**

Все это за нас сделает OS!



THE PROBLEM

`java.lang.Thread <=> OS-thread`

- + в JVM ничего делать не нужно
- + scheduler, над которым работают 50 лет

THE PROBLEM

`java.lang.Thread <=> OS-thread`

- + в JVM ничего делать не нужно
- + scheduler, над которым работают 50 лет

А минусы будут?

THE PROBLEM

`java.lang.Thread <=> OS-thread`

- + в JVM ничего делать не нужно
- + scheduler, над которым работают 50 лет

- ограничение по количеству
- долгое создание
- ограниченный стек

THE PROBLEM

```
final int count = 100_000;
final Thread[] threads = new Thread[count];

for (int i = 0; i < count; i++) {
    System.out.println("Starting thread #" + i);
    final int n = i;
    threads[i] = new Thread(() -> {
        Thread.sleep(Duration.ofSeconds(10));
        System.out.println("Thread #" + n + " finished");
    });
    threads[i].start();
}

for (int i = 0; i < count; i++) {
    threads[i].join();
}
```

THE PROBLEM

```
final int count = 100_000;
final Thread[] threads = new Thread[count];

for (int i = 0; i < count; i++) {
    System.out.println("Starting thread #" + i);
    final int n = i;
    threads[i] = new Thread(() -> {
        Thread.sleep(Duration.ofSeconds(10));
        System.out.println("Thread #" + n + " finished");
    });
    threads[i].start();
}

for (int i = 0; i < count; i++) {
    threads[i].join();
}
```

THE PROBLEM

```
final int count = 100_000;  
final Thread[] threads = new Thread[count];
```

```
Starting thread #10775  
Starting thread #10776  
Starting thread #10777  
Starting thread #10778  
Starting thread #10779  
Starting thread #10780  
[1.232s][warning][os,thread] Failed to start thread "Unknown thread" -  
pthread_create failed (EAGAIN) for attributes: stacksize: 1024k, guardsize: 0k, detached.  
[1.232s][warning][os,thread] Failed to start the native thread for java.lang.Thread  
"Thread-10780"
```

```
for (int i = 0; i < count; i++) {  
    threads[i].join();  
}
```



THE PROBLEM

```
final int count = 100_000;  
final Thread[] threads = new Thread[count];
```

```
Starting thread #10775  
Starting thread #10776  
Starting thread #10777  
Starting thread #10778  
Starting thread #10779  
Starting thread #10780  
[1.232s][warning][os,thread] Failed to start thread "Unknown thread" -  
pthread_create failed (EAGAIN) for attributes: stacksize: 1024k, guardsize: 0k, detached.  
[1.232s][warning][os,thread] Failed to start the native thread for java.lang.Thread
```

```
Starting thread #92148  
Starting thread #92149  
Starting thread #92150  
[457.214s][warning][os,thread] Failed to start thread - _beginthreadex failed (EINVAL) for attributes: stacksize: default, flags: CREATE_SUSPENDED STACK_SIZE_PARAM_IS.  
Exception in thread "main" java.lang.OutOfMemoryError Create breakpoint : unable to create native thread: possibly out of memory or process/resource limits reached  
at java.base/java.lang.Thread.start0(Native Method)  
at java.base/java.lang.Thread.start(Thread.java:802)  
at com.company.Main.main(Main.java:20)
```



OpenJDK Platform binary

4 550,8 MB

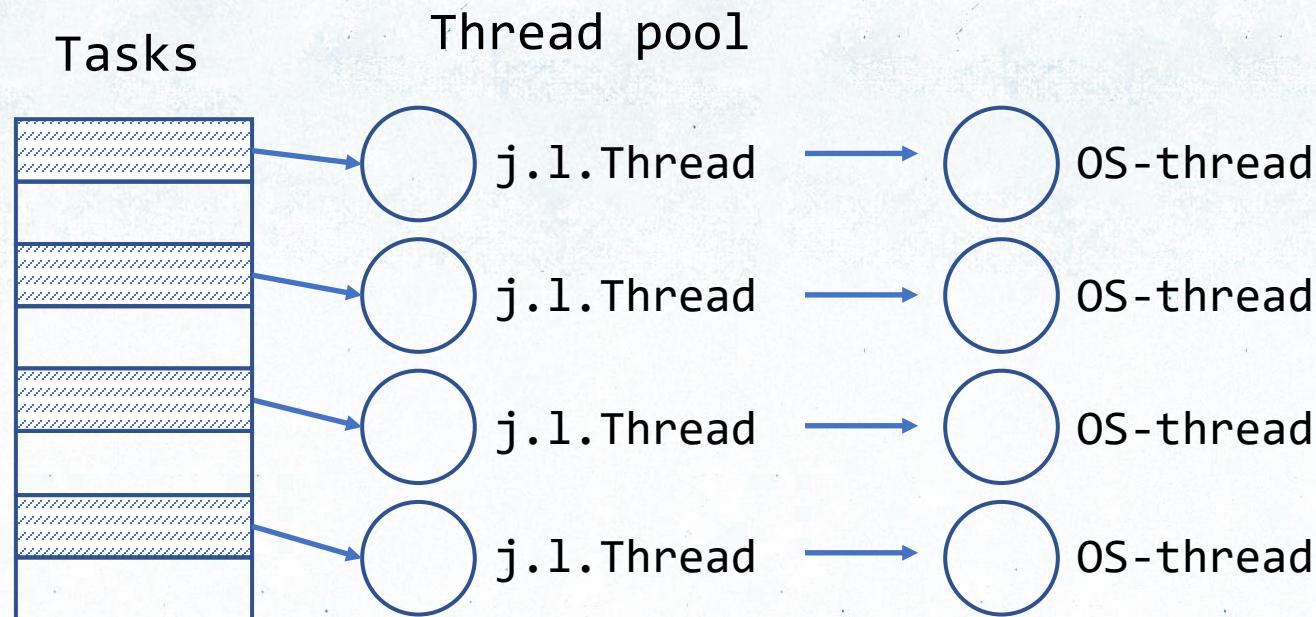
THE PROBLEM

`java.lang.Thread <=> OS-thread`

Следствие реализации:

- о Треды – **дорогие!** Лишний раз их создавать не стоит (а хочется)
- о Значит треды будем **пульть**

THREAD POOL TO THE RESCUE



THREAD POOL TO THE RESCUE

Вместо тредов таски в тредпуле:

- + Больше не боимся ограничений OS!

THREAD POOL TO THE RESCUE

```
final int count = 100_000;
final Thread[] threads = new Thread[count];

for (int i = 0; i < count; i++) {
    System.out.println("Starting thread #" + i);
    final int n = i;
    threads[i] = new Thread(() -> {
        Thread.sleep(Duration.ofSeconds(10));
        System.out.println("Thread #" + n + " finished");
    });
    threads[i].start();
}
```

```
Starting thread #92148
Starting thread #92149
Starting thread #92150
[457.214s][warning][os,thread] Failed to start thread - _beginthreadex failed (EINVAL) for attributes: stacksize: default, flags: CREATE_SUSPENDED STACK_SIZE_PARAM_IS_StackCommitSize
Exception in thread "main" java.lang.OutOfMemoryError Create breakpoint : unable to create native thread: possibly out of memory or process/resource limits reached
  at java.base/java.lang.Thread.start0(Native Method)
  at java.base/java.lang.Thread.start(Thread.java:802)
  at com.company.Main.main(Main.java:20)
```



OpenJDK Platform binary

4 550,8 MB

THREAD POOL TO THE RESCUE

```
ExecutorService es =  
    Executors.newFixedThreadPool(4);  
  
for (int i = 0; i < 100_000; i++) {  
    System.out.println("Submitting task #" + i);  
    es.execute(() -> {  
        Thread.sleep(Duration.ofSeconds(10));  
    });  
}
```

THREAD POOL TO THE RESCUE

```
ExecutorService es =  
    Executors.newFixedThreadPool(4);  
  
for (int i = 0; i < 100_000; i++) {  
    System.out.println("Submitting task #" + i);  
    es.execute(() -> {  
        Thread.sleep(Duration.ofSeconds(10));  
    });
```



```
Submitting task #99994  
Submitting task #99995  
Submitting task #99996  
Submitting task #99997  
Submitting task #99998  
Submitting task #99999
```

THREAD POOL TO THE RESCUE

Вместо тредов таски в тредпуле:

- + Больше не боимся ограничений OS!
- + Специализированные scheduler-ы
(теперь уже придется написать сами, ну что же)

THREAD POOL TO THE RESCUE

Вместо тредов таски в тредпуле:

- + Больше не боимся ограничений OS!
- + Специализированные scheduler-ы

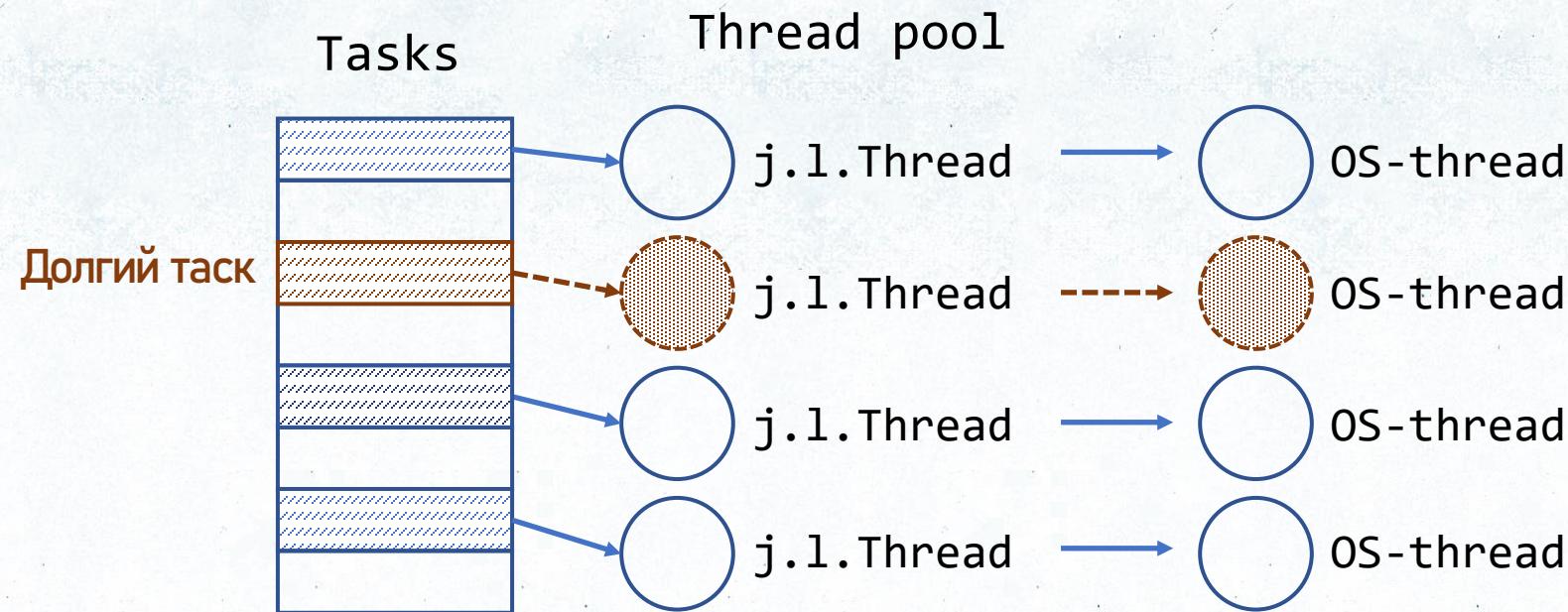
А минусы будут? 😊

THREAD POOL TO THE RESCUE

Вместо тредов таски в тредпуле:

- + Больше не боимся ограничений OS!
- + Специализированные scheduler-ы
- Долгий (бесконечный) таск забирает OS-thread

THREAD POOL TO THE RESCUE

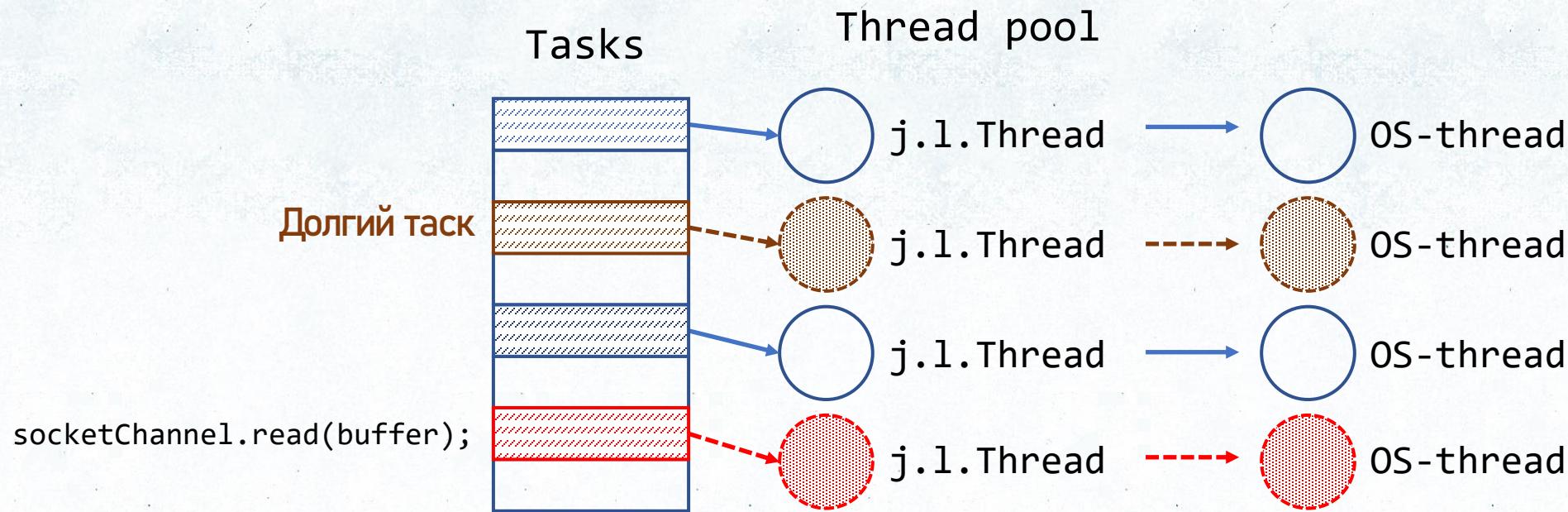


THREAD POOL TO THE RESCUE

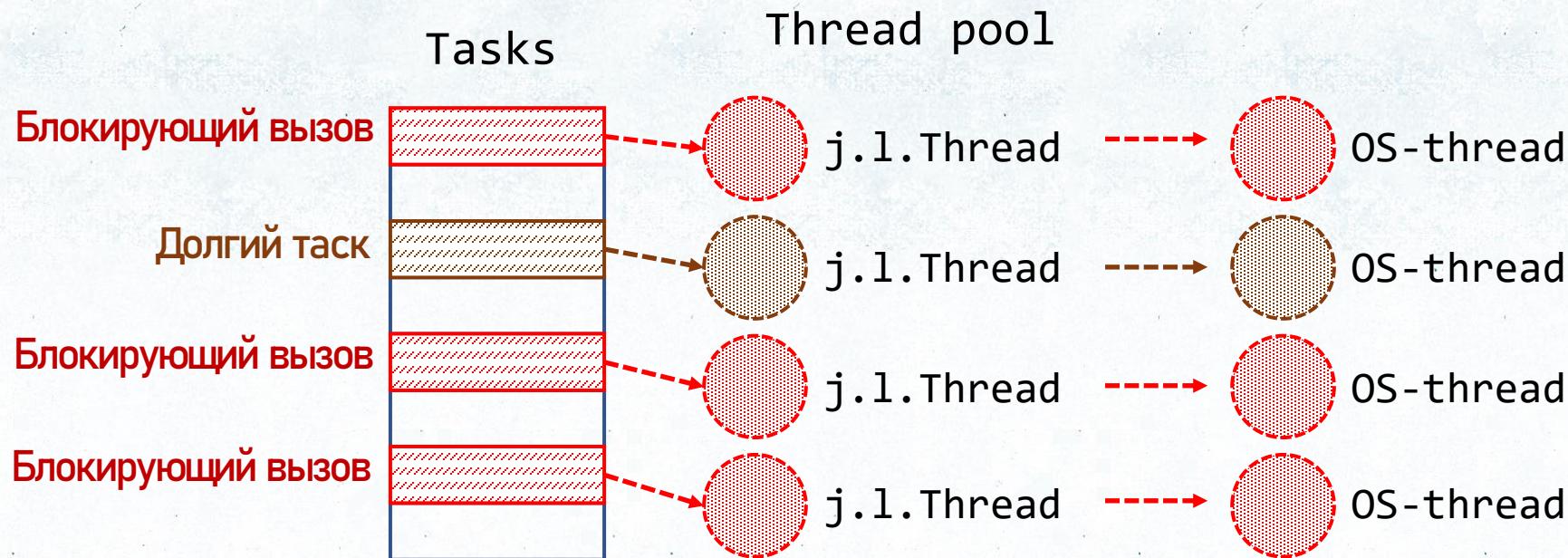
Блокирующие вызовы:

- Ожидание ответа по сети
- Ожидание окончания файловой операции
- Ожидание монитора
- Ожидание по таймауту (Thread.sleep)
- ...

THREAD POOL TO THE RESCUE



THREAD POOL TO THE RESCUE



THREAD POOL TO THE RESCUE

Вместо тредов таски в тредпуле:

- + Больше не боимся ограничений OS!
- + Специализированные scheduler-ы

- Долгий (бесконечный) таск забирает OS-thread
- Блокирующие вызовы, как крайний случай

TO BLOCK OR NOT TO BLOCK

Блокирующие вызовы:

- Ожидание ответа по сети
- Ожидание окончания файловой операции
- Ожидание монитора
- Ожидание по таймауту (Thread.sleep)
- ...

TO BLOCK OR NOT TO BLOCK

Блокирующие вызовы:

- Ожидание ответа по сети
- Ожидание окончания файловой операции
- Ожидание монитора
- Ожидание по таймауту (Thread.sleep)
- ...

Ожидание реакции OS.

TO BLOCK OR NOT TO BLOCK

```
int listen_sock = 0;
struct sockaddr_in serv_addr;
char buffer[SIZE] = {0};
...
listen_sock = socket(AF_INET, SOCK_STREAM, 0);
connect(listen_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
...
n = read(listen_sock, buf, sizeof(buf));
```

TO BLOCK OR NOT TO BLOCK

```
int listen_sock = 0;
struct sockaddr_in serv_addr;
char buffer[SIZE] = {0};

...
listen_sock = socket(AF_INET, SOCK_STREAM, 0);           ← Открыли сокет (получили FD)
connect(listen_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
...

n = read(listen_sock, buf, sizeof(buf)); ← Читаем, блокируемся!
```

TO BLOCK OR NOT TO BLOCK

```
int listen_sock = 0;
struct sockaddr_in serv_addr;
char buffer[SIZE] = {0};
...
listen_sock = socket(AF_INET, SOCK_STREAM, 0);
connect(listen_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
...
n = read(listen_sock, buf, sizeof(buf));
```

Открыли сокет (получили FD)

Читаем, блокируемся!

И пока данные не запишут в сокет, поток так и будет заблокирован.

Аналогичный API есть и в Java: это java.net.Socket (getInputStream)

TO BLOCK OR NOT TO BLOCK

Блокирующие вызовы:

- Ожидание ответа по сети
- Ожидание окончания файловой операции
- Ожидание монитора
- Ожидание по таймауту (Thread.sleep)
- ...

Ожидание реакции OS.

TO BLOCK OR NOT TO BLOCK

Блокирующие вызовы:

- Ожидание ответа по сети
- Ожидание окончания файловой операции
- Ожидание монитора
- Ожидание по таймауту (Thread.sleep)
- ...

Ожидание реакции OS. Но можно и не ждать!

TO BLOCK OR NOT TO BLOCK

OS предоставляет асинхронный API (AsyncIO + EPOLL)

Вместо занимающего тред ожидания ответа
подписываемся на событие.

Когда оно случится, вызывается колбек.



```
listen_sock = socket(AF_INET, SOCK_STREAM, 0);
fcntl(listen_sock, F_SETFL, fcntl(listen_sock, F_GETFL, 0) | O_NONBLOCK);
epfd = epoll_create(1);
struct epoll_event ev;
ev.events = EPOLLIN | EPOLLOUT | EPOLLET; ev.data.fd = listen_sock;
epoll_ctl(epfd, EPOLL_CTL_ADD, listen_sock, &ev);

...
// server itself
for (;;) {
    nfds = epoll_wait(epfd, events, MAX_EVENTS, -1);
    for (i = 0; i < nfds; i++) {
        ...
        if (events[i].events & EPOLLIN) {
            char buf[BUF_SIZE];
            for(;;) {
                n = read(events[i].data.fd, buf, sizeof(buf));
                if (n <= 0) break;
                // execute callback
            }
        }
    }
}
```

```
listen_sock = socket(AF_INET, SOCK_STREAM, 0); ← Открыли сокет (получили FD)
fcntl(listen_sock, F_SETFL, fcntl(listen_sock, F_GETFL, 0) | O_NONBLOCK);
epfd = epoll_create(1);                                Неблокирующий режим!
struct epoll_event ev;                                Если данные не готовы => read сразу вернется
ev.events = EPOLLIN | EPOLLOUT | EPOLLET; ev.data.fd = listen_sock;
epoll_ctl(epfd, EPOLL_CTL_ADD, listen_sock, &ev);

...
// server itself
for (;;) {
    nfds = epoll_wait(epfd, events, MAX_EVENTS, -1);
    for (i = 0; i < nfds; i++) {
        ...
        if (events[i].events & EPOLLIN) {
            char buf[BUF_SIZE];
            for(;;) {
                n = read(events[i].data.fd, buf, sizeof(buf));
                if (n <= 0) break;
                // execute callback
            }
        }
    }
}
```

```
listen_sock = socket(AF_INET, SOCK_STREAM, 0); ← Открыли сокет (получили FD)
fcntl(listen_sock, F_SETFL, fcntl(listen_sock, F_GETFL, 0) | O_NONBLOCK);
epfd = epoll_create(1);
struct epoll_event ev;
ev.events = EPOLLIN | EPOLLOUT | EPOLLET; ev.data.fd = listen_sock;
epoll_ctl(epfd, EPOLL_CTL_ADD, listen_sock, &ev);
...
// server itself
for (;;) {
    nfds = epoll_wait(epfd, events, MAX_EVENTS, -1);
    for (i = 0; i < nfds; i++) {
        ...
        if (events[i].events & EPOLLIN) {
            char buf[BUF_SIZE];
            for(;;) {
                n = read(events[i].data.fd, buf, sizeof(buf));
                if (n <= 0) break;
                // execute callback
            }
        }
    }
}
```

Неблокирующий режим!
Если данные не готовы => read сразу вернется

Но что же нам теперь, все время в цикле крутиться и ждать, пока read вернет все данные? Чем это лучше, чем заблокироваться?

```
listen_sock = socket(AF_INET, SOCK_STREAM, 0); ← Открыли сокет (получили FD)
fcntl(listen_sock, F_SETFL, fcntl(listen_sock, F_GETFL, 0) | O_NONBLOCK)
epfd = epoll_create(1);                                Неблокирующий режим!
struct epoll_event ev;
ev.events = EPOLLIN | EPOLLOUT | EPOLLET; ev.data.fd = listen_sock;
epoll_ctl(epfd, EPOLL_CTL_ADD, listen_sock, &ev); ← Будем следить за ивентами (включая
...                                              этот сокет)
...
// server itself
for (;;) {
    nfps = epoll_wait(epfd, events, MAX_EVENTS, -1);
    for (i = 0; i < nfps; i++) {
        ...
        if (events[i].events & EPOLLIN) {
            char buf[BUF_SIZE];
            for(;;) {
                n = read(events[i].data.fd, buf, sizeof(buf));
                if (n <= 0) break;
                // execute callback
            }
        }
    }
}
```

```
listen_sock = socket(AF_INET, SOCK_STREAM, 0); ← Открыли сокет (получили FD)
fcntl(listen_sock, F_SETFL, fcntl(listen_sock, F_GETFL, 0) | O_NONBLOCK)
epfd = epoll_create(1);                                Неблокирующий режим!
struct epoll_event ev;
ev.events = EPOLLIN | EPOLLOUT | EPOLLET; ev.data.fd = listen_sock;
epoll_ctl(epfd, EPOLL_CTL_ADD, listen_sock, &ev); ← Будем следить за ивентами (включая
...                                              этот сокет)
// server itself
for (;;) {
    nfds = epoll_wait(epfd, events, MAX_EVENTS, -1); ← Блокируемся и ждем, пока
    for (i = 0; i < nfds; i++) {                      что-то произойдет (смотрим
        ...                                              сразу за многими сокетами!)
        if (events[i].events & EPOLLIN) {
            char buf[BUF_SIZE];
            for(;;) {
                n = read(events[i].data.fd, buf, sizeof(buf));
                if (n <= 0) break;
                // execute callback
            }
        }
    }
}
```

```
listen_sock = socket(AF_INET, SOCK_STREAM, 0); ← Открыли сокет (получили FD)
fcntl(listen_sock, F_SETFL, fcntl(listen_sock, F_GETFL, 0) | O_NONBLOCK)
epfd = epoll_create(1);                                Неблокирующий режим!
struct epoll_event ev;
ev.events = EPOLLIN | EPOLLOUT | EPOLLET; ev.data.fd = listen_sock;
epoll_ctl(epfd, EPOLL_CTL_ADD, listen_sock, &ev); ← Будем следить за ивентами (включая
...                                              этот сокет)
// server itself
for (;;) {
    nfds = epoll_wait(epfd, events, MAX_EVENTS, -1); ← Блокируемся и ждем, пока
    for (i = 0; i < nfds; i++) {                      что-то произойдет (смотрим
        ...                                              сразу за многими сокетами!)
        if (events[i].events & EPOLLIN) {              Т.е. теперь не нужно
            char buf[BUF_SIZE];                         блокировать все трэды,
            for(;;) {                                  достаточно выделить один!
                n = read(events[i].data.fd, buf, sizeof(buf));
                if (n <= 0) break;
                // execute callback
            }
        }
    }
}
```

```
listen_sock = socket(AF_INET, SOCK_STREAM, 0); ← Открыли сокет (получили FD)
fcntl(listen_sock, F_SETFL, fcntl(listen_sock, F_GETFL, 0) | O_NONBLOCK)
epfd = epoll_create(1);                                Неблокирующий режим!
struct epoll_event ev;
ev.events = EPOLLIN | EPOLLOUT | EPOLLET; ev.data.fd = listen_sock;
epoll_ctl(epfd, EPOLL_CTL_ADD, listen_sock, &ev); ← Будем следить за ивентами (включая
...                                              этот сокет)
// server itself
for (;;) {
    nfds = epoll_wait(epfd, events, MAX_EVENTS, -1); ← Блокируемся и ждем, пока
    for (i = 0; i < nfds; i++) {                      что-то произойдет (смотрим
        ...                                              сразу за многими сокетами)
        if (events[i].events & EPOLLIN) {
            char buf[BUF_SIZE];
            for(;;) {
                n = read(events[i].data.fd, buf, sizeof(buf));
                if (n <= 0) break;                      Из тех, где что-то произошло, можем читать.
                // execute callback                  Делаем это только постфактум, когда
            }                                         случилось прерывание.
        }
    }
}
```

TO BLOCK OR NOT TO BLOCK

Через EPOLL реализованы асинхронные API для работы с сетью и файловой системой в managed языках.

AsynchronousSocketChannel

AsynchronousFileChannel

...

TO BLOCK OR NOT TO BLOCK

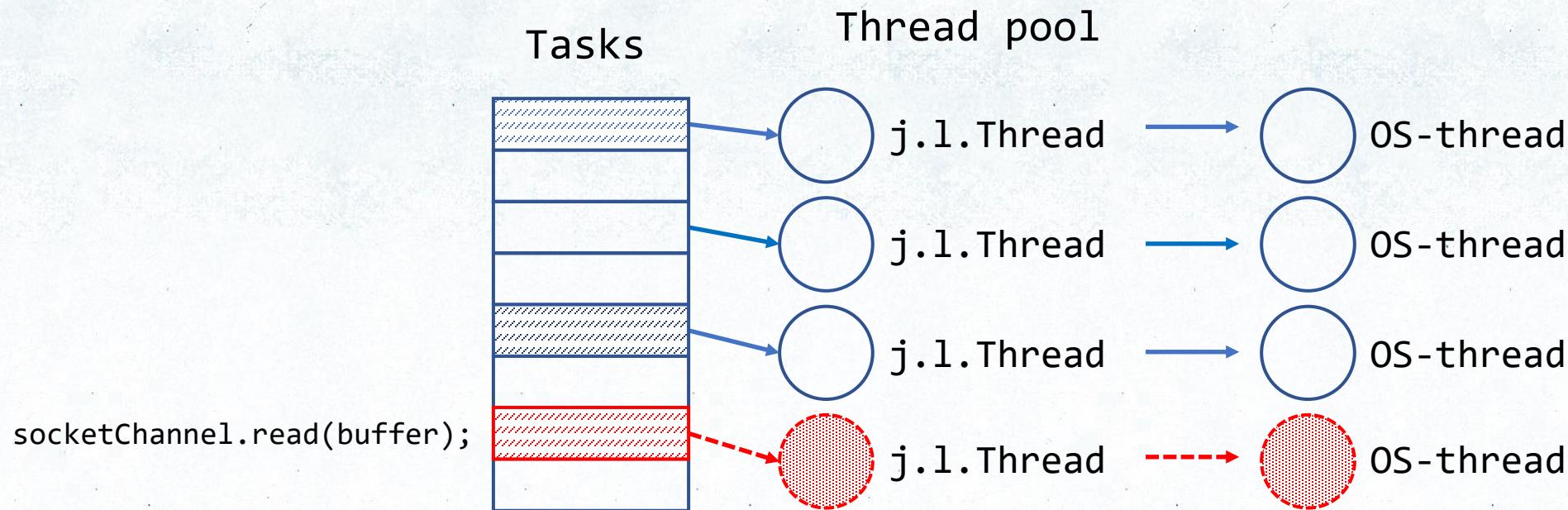
Через EPOLL реализованы асинхронные API для работы с сетью и файловой системой в managed языках.

```
<A> void read(ByteBuffer dst,  
               long position,  
               A attachment,  
               CompletionHandler<Integer,? super A> handler);
```

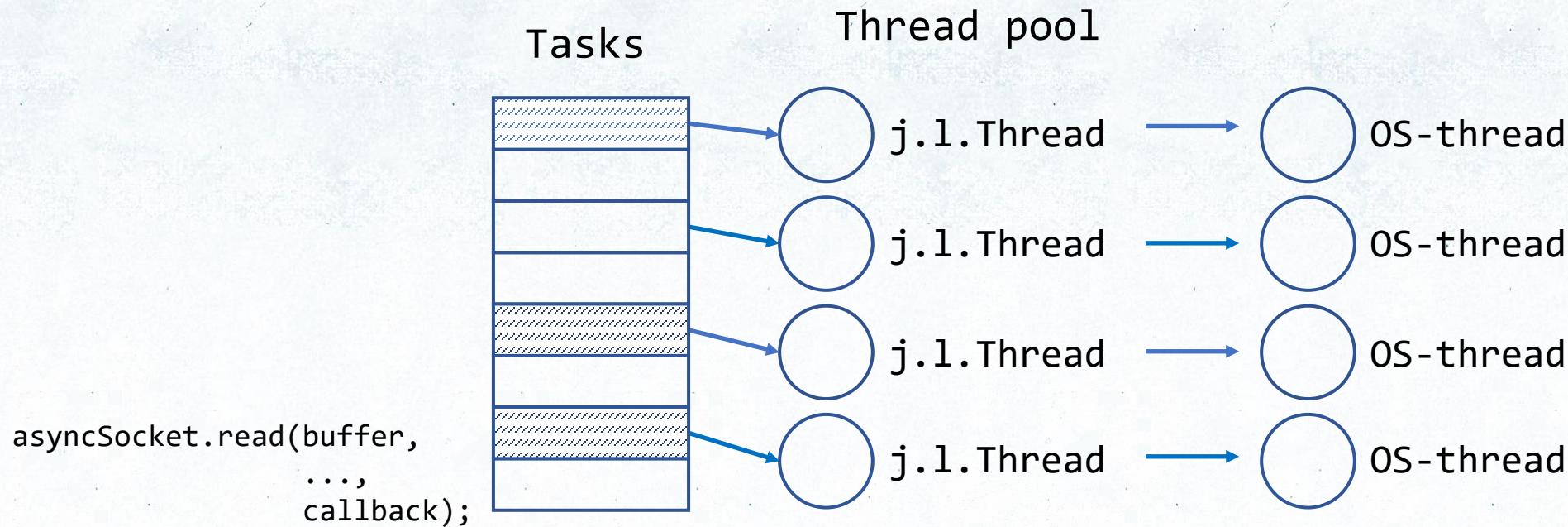


callback

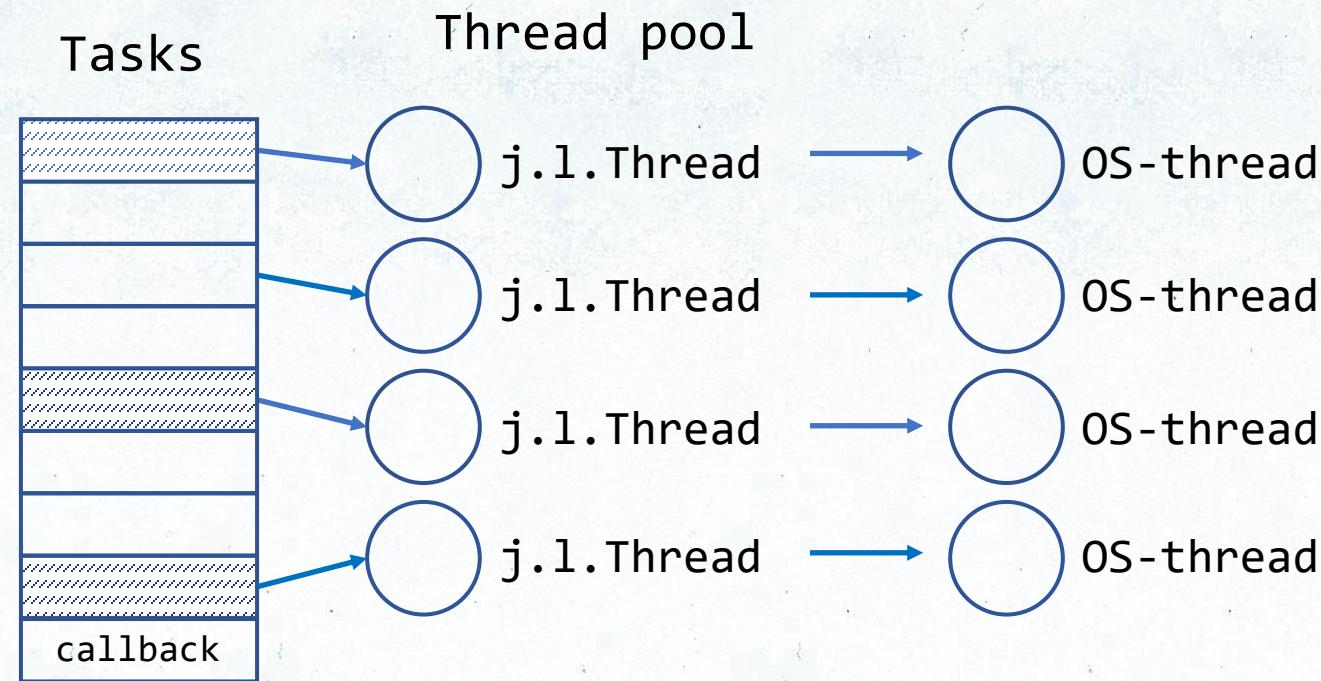
THREAD POOL TO THE RESCUE



THREAD POOL TO THE RESCUE

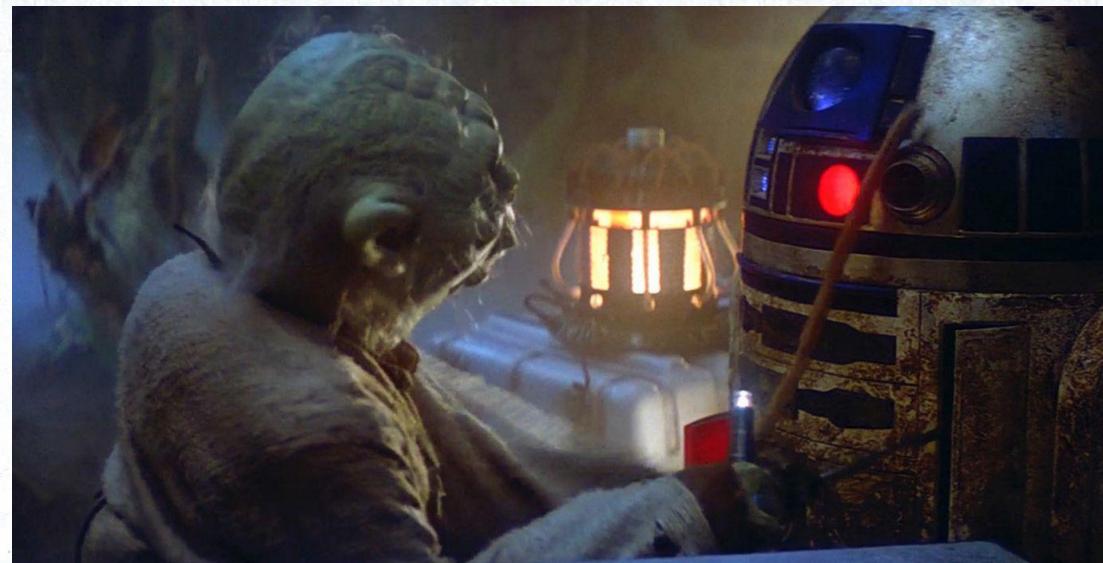


THREAD POOL TO THE RESCUE



THREAD POOL TO THE RESCUE

Какие еще могут быть проблемы с тредуплом и с callbacks?



THREAD POOL TO THE RESCUE

Главная проблема подхода с callback –
уродование кода и сложность отладки



THREAD POOL TO THE RESCUE

Главная проблема подхода с callback –
уродование кода и сложность отладки

```
int v = foo();
validate(v);
int k = bar(v);
validate(k);
baz(k);
```



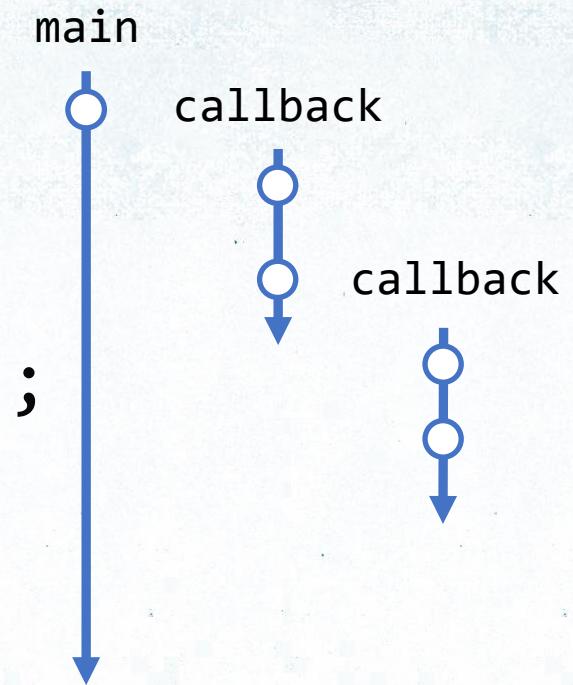
THREAD POOL TO THE RESCUE

Главная проблема подхода с callback –
уродование кода и сложность отладки

```
int v = foo();
validate(v);
int k = bar(v);
validate(k);
baz(k);
```



```
foo(v -> {
    validate(v);
    bar(v, k -> {
        validate(k);
        baz(k);
    });
});
```



```
2 var floppy = require('floppy');
3
4 floppy.load('disk1', function (data1) {
5   floppy.prompt('Please insert disk 2', function() {
6     floppy.load('disk2', function (data2) {
7       floppy.prompt('Please insert disk 3', function() {
8         floppy.load('disk3', function (data3) {
9           floppy.prompt('Please insert disk 4', function() {
10          floppy.load('disk4', function (data4) {
11            floppy.prompt('Please insert disk 5', function() {
12              floppy.load('disk5', function (data5) {
13                floppy.prompt('Please insert disk 6', function() {
14                  floppy.load('disk6', function (data6) {
15                    //if node.js would have existed in
16                    });
17                    });
18                    });
19                    });
20                    });
21                    });
22                    });
23                    });
24                    });
25                    });
26                    });


```

Эта проблема известна, как **callback hell**. Читать – сложно, отлаживать – почти невозможно.



Эта проблема известна, как **callback hell**. Читать – сложно, отлаживать – почти невозможно.

TO BLOCK OR NOT TO BLOCK

Альтернатива колбекам - Future

```
Future<Integer> read(ByteBuffer dst, long position);
```

TO BLOCK OR NOT TO BLOCK

Альтернатива колбекам - Future

```
Future<Integer> read(ByteBuffer dst, long position);
```

- Вызов `read` неблокирующий, сразу возвращает результат
- Вызов `Future.get` – блокирующий

TO BLOCK OR NOT TO BLOCK

Альтернатива колбекам - Future

```
Future<Integer> read(ByteBuffer dst, long position);
```

- Вызов `read` неблокирующий, сразу возвращает результат
- Вызов `Future.get` – блокирующий
- Само по себе это не поможет, но помогут комбинаторы
- Вместо вызова `get` описываем следующие шаги

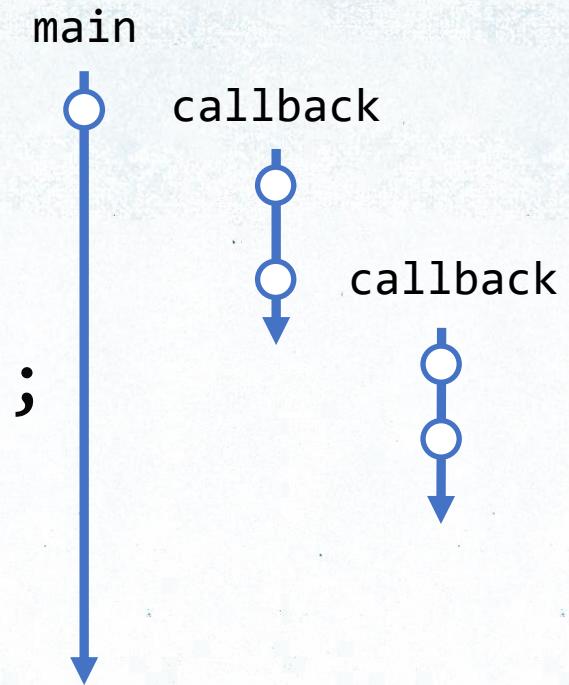
TO BLOCK OR NOT TO BLOCK

Главная проблема подхода с callback –
уродование кода и сложность отладки

```
int v = foo();
validate(v);
int k = bar(v);
validate(k);
baz(k);
```



```
foo(v -> {
    validate(v);
    bar(v, k -> {
        validate(k);
        baz(k);
    });
});
```



TO BLOCK OR NOT TO BLOCK

С комбинаторами – лучше. Callback hell больше нет, но код все еще другой и непривычный.

```
CompletableFuture.supplyAsync(this::foo).  
    thenApply(v -> { validate(v); return v; }).  
    thenApplyAsync(this::bar).  
    thenApply(k -> { validate(k); return k; }).  
    thenAcceptAsync(this::baz);
```



TO BLOCK OR NOT TO BLOCK

Итого: первый путь решения проблемы treadов –
изменение исходного кода

TO BLOCK OR NOT TO BLOCK

Итого: первый путь решения проблемы treadов – изменение исходного кода

- о Используем асинхронные API
- о Декомпозириуем код с помощью Future и комбинаторов
- о Избегаем кода с блокировками на мониторах

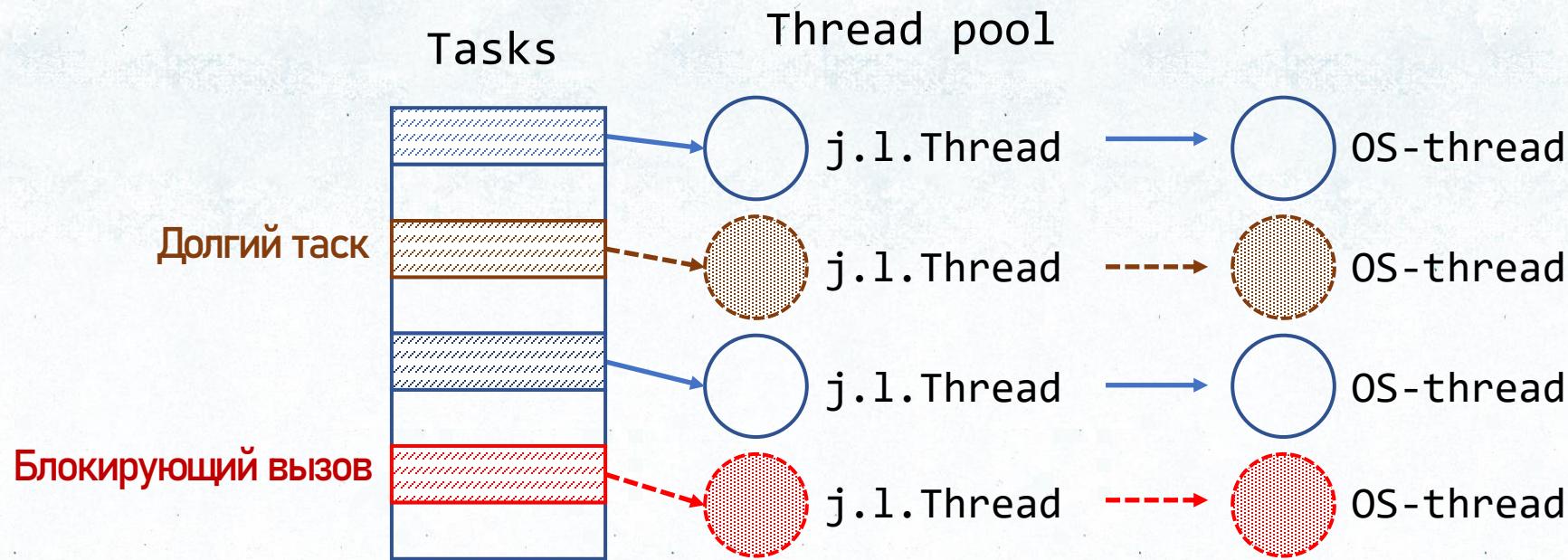
TO BLOCK OR NOT TO BLOCK

Итого: первый путь решения проблемы treadов – изменение исходного кода

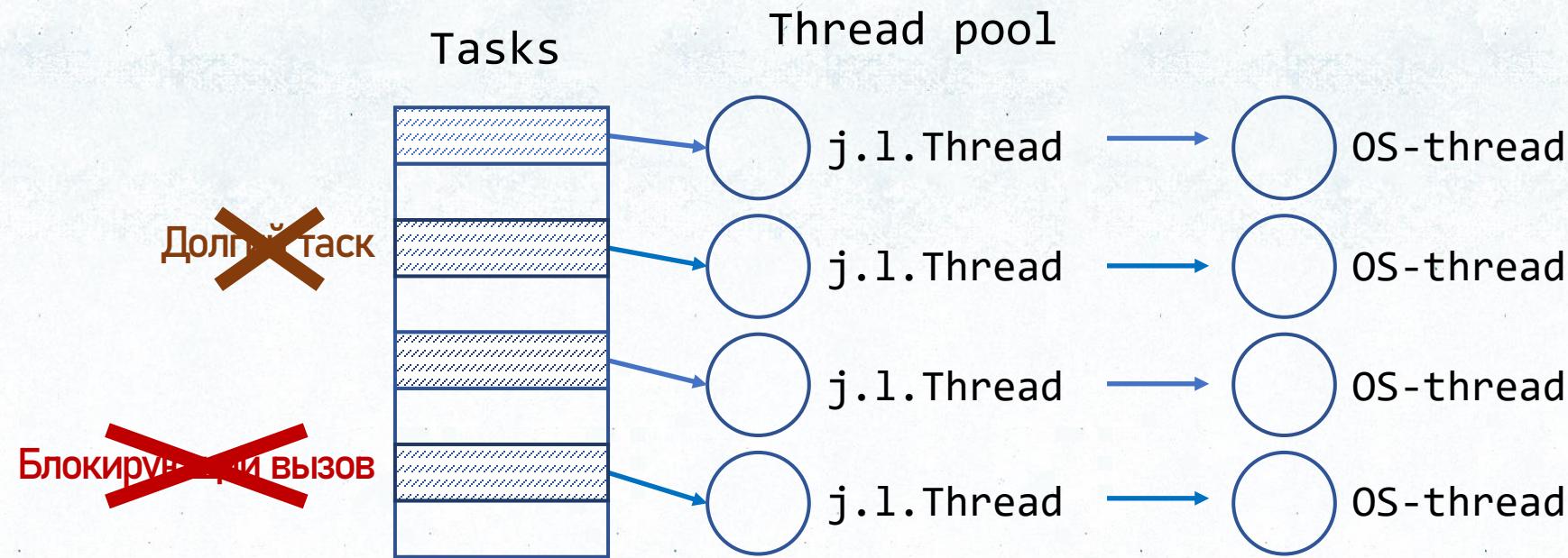
- о Используем асинхронные API
- о Декомпозириуем код с помощью Future и комбинаторов
- о Избегаем кода с блокировками на мониторах

После этого начинает работать подход с treadпулом.

TO BLOCK OR NOT TO BLOCK



TO BLOCK OR NOT TO BLOCK



РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

Основанные на ThreadPool и изменении
подхода к написанию кода:

- + Реализация на уровне библиотеки
- ± Сильное изменение кода

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

Основанные на ThreadPool и изменениях
подхода к написанию кода:

- + Реализация на уровне библиотеки
- ± Сильное изменение кода
- Хрупкое решение (в случае Java)



РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

Аналогичный подход в функциональных языках:

- Scala: ZIO, Cats Effect
- Pure functional

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

Аналогичный подход в функциональных языках:

- Scala: ZIO, Cats Effect
- Pure functional

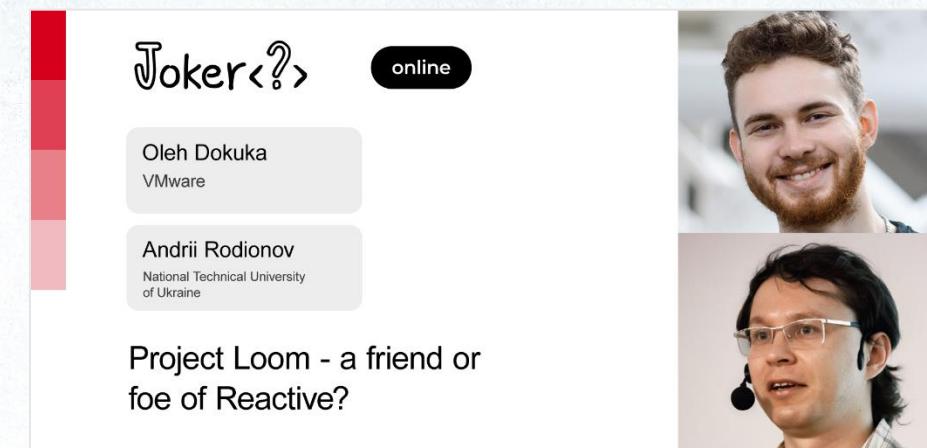
Реактивные фреймворки

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

Аналогичный подход в функциональных языках:

- Scala: ZIO, Cats Effect
- Pure functional

Реактивные фреймворки:



Project Loom - a friend or
foe of Reactive?

youtube.com/watch?v=YwG04UZP2a0

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

больше

Влияние на код и язык

меньше

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

ThreadPool +
Callbacks/Future/
Combinators

больше

Влияние на код и язык



меньше

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture

больше

Влияние на код и язык

меньше

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture

больше

Влияние на код и язык

меньше



THREAD WARS

EPISODE I

СКРЫТЫЙ
ТРЕДПУЛ



TO HIDE A THREAD POOL

И все же писать и отлаживать императивный
код хочется

TO HIDE A THREAD POOL

И все же писать и отлаживать императивный
код хочется

А что если преобразовывать код для работы с
тредпулом... автоматически?

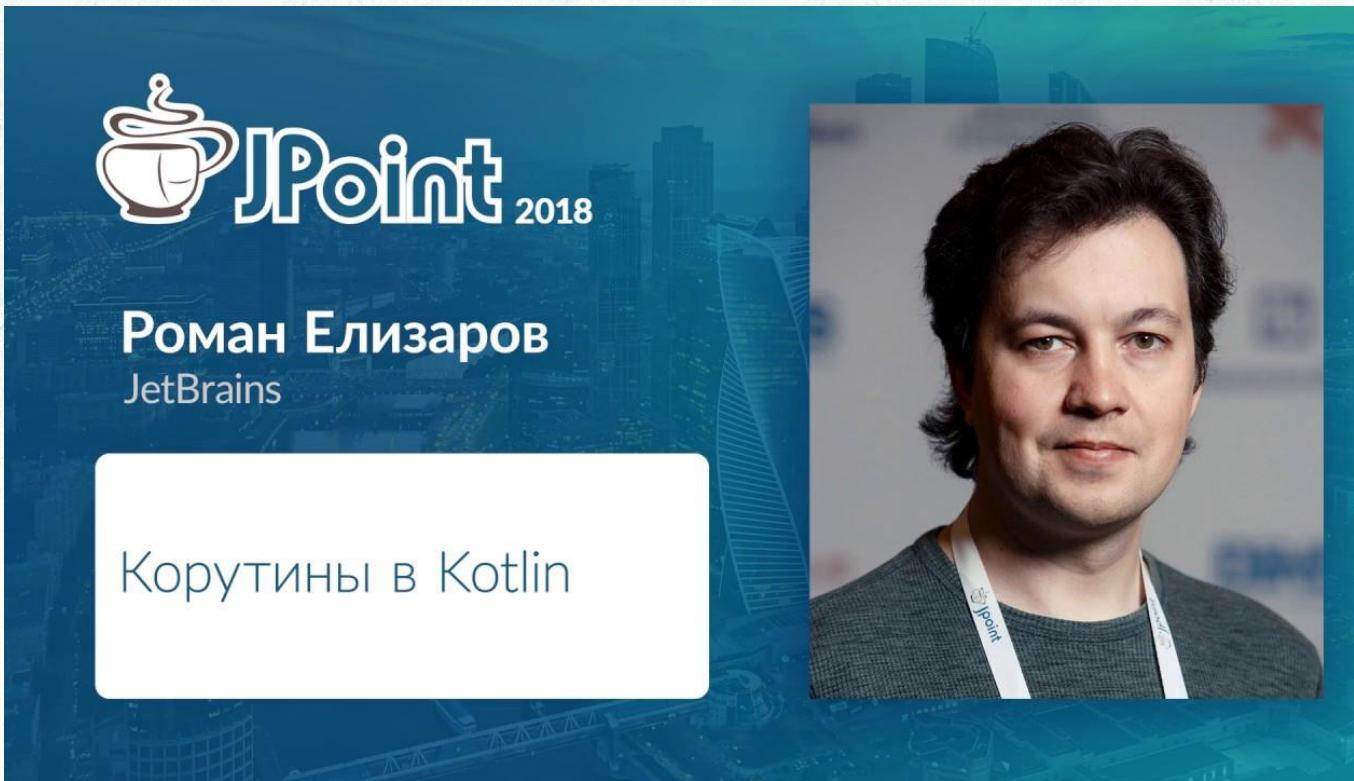
TO HIDE A THREAD POOL

И все же писать и отлаживать императивный
код хочется

А что если преобразовывать код для работы с
тредпулом... автоматически?

Так делают в C#/JS/C++/Rust/Kotlin

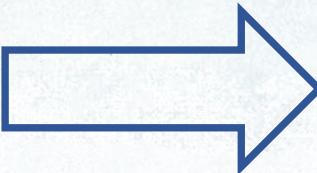
TO HIDE A THREAD POOL



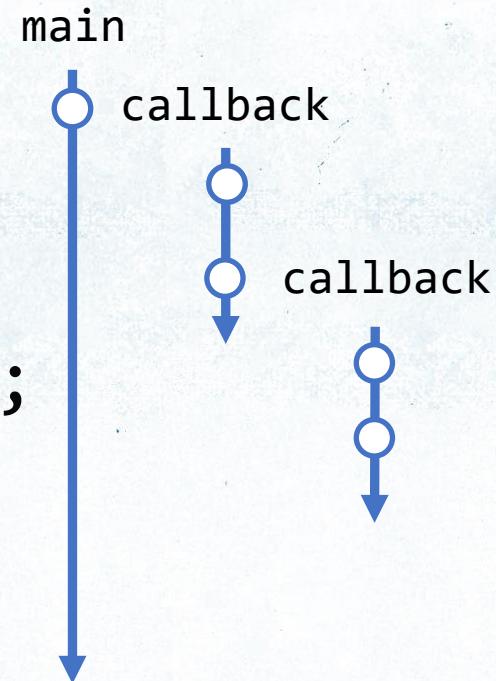
youtube.com/watch?v=rB5Q3y73FTo

TO HIDE A THREAD POOL

```
int v = foo();  
validate(v);  
int k = bar(v);  
validate(k);  
baz(k);
```



```
foo(v -> {  
    validate(v);  
    bar(v, k -> {  
        validate(k);  
        baz(k);  
    });  
});
```



TO HIDE A THREAD POOL

```
fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```

TO HIDE A THREAD POOL

```
fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```

```
public suspend fun delay(timeMillis: Long)
```

Delays coroutine for a given time without blocking a thread and resumes it after a specified time.

TO HIDE A THREAD POOL

```
fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```

```
public suspend fun delay(timeMillis: Long)
```

Delays coroutine for a given time without blocking a thread and resumes it after a specified time.

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```

```
public suspend fun delay(timeMillis: Long)
```

Delays coroutine for a given time without blocking a thread and resumes it after a specified time.

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    ...  
}  
  
fun main() = runBlocking {  
    for (i in 1..100_000) {  
        Launch {  
            val result = test(i)  
            println("Coroutine $i returned $result")  
        }  
    }  
}
```

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    ...  
}  
  
fun main() = runBlocking {  
    for (i in 1..100_000) {  
        Launch {  
            val result = test(i)  
            println("Coroutine $i returned $result")  
        }  
    }  
}
```

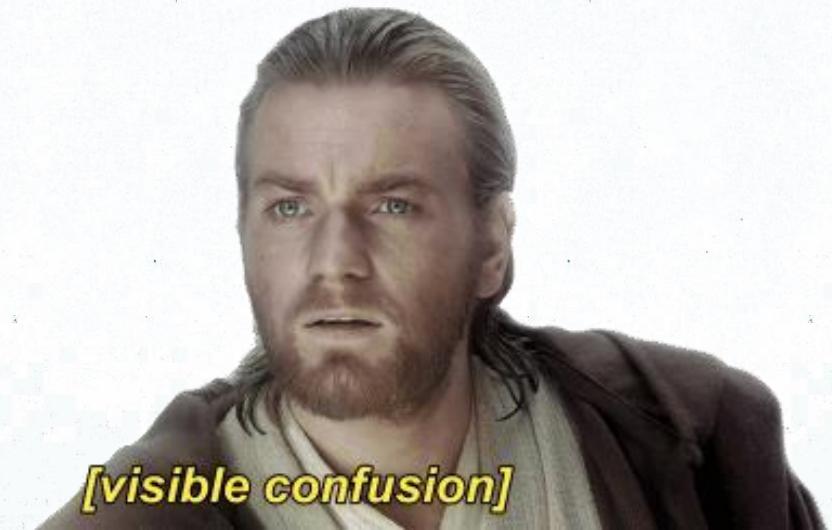
Coroutine 99997 returned -933214581
Coroutine 99998 returned 130246854
Coroutine 99999 returned 1250627867
Coroutine 100000 returned -850427097
Process finished with exit code 0

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```



А как это работает то?



TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    ...  
}  
  
fun main() = runBlocking {  
    for (i in 1..100_000) {  
        Launch { ←  
            val result = test(i)  
            println("Coroutine $i returned $result")  
        }  
    }  
}
```

Запуск таска
на тредпуле

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    ↳ delay(100)  
    print("$input -> $seed")  
    ↳ delay(100)  
    return input + seed  
}
```

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    ↳ delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```

} callback

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    ↳ delay(100)  
    return input + seed } callback  
}
```

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    ↳ delay(100)  
    print("$input -> $seed")  
    ↳ delay(100)  
    return input + seed  
}
```



```
Object test(int input,  
Continuation<?> cont) {  
    ...  
}
```

```
}
```

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    → delay(100)  
    print("$input -> $seed")  
    → delay(100)  
    return input + seed  
}
```



```
Object test(int input,  
Continuation<?> cont) {
```

```
    ...  
    switch (cont.label) {
```

```
}
```

```
    ...  
}
```

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```



```
Object test(int input,  
           Continuation<?> cont) {  
    ...  
    switch (cont.label) {  
        case 0:  
            cont.input = input;  
            cont.seed = Random.nextInt();  
            cont.label = 1;  
            delay(100, cont);  
            break;  
        ...  
    }  
    ...  
}
```

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```



```
Object test(int input,  
           Continuation<?> cont) {  
    ...  
    switch (cont.label) {  
        ...  
        case 1:  
            print("$cont.input ->  
                  $cont.seed")  
            cont.label = 2;  
            delay(100, cont);  
            break;  
        ...  
    }  
    ...  
}
```

TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```



```
Object test(int input,  
           Continuation<?> cont) {  
    ...  
    switch (cont.label) {  
        ...  
        case 2:  
            cont.caller.resumeWith(  
                cont.input + cont.seed);  
    }  
    ...  
}
```

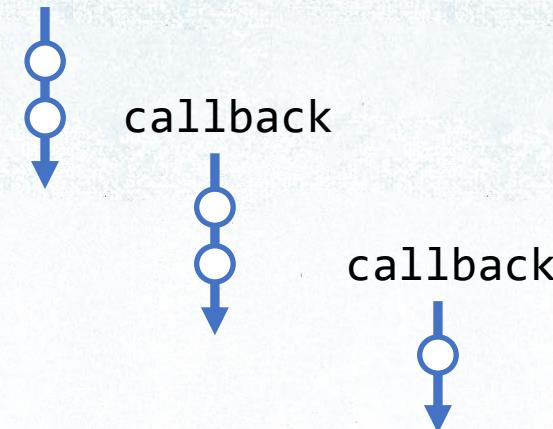
TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```



TO HIDE A THREAD POOL

```
suspend fun test(input: Int): Int {  
    val seed = Random.nextInt()  
    delay(100)  
    print("$input -> $seed")  
    delay(100)  
    return input + seed  
}
```



TO HIDE A THREAD POOL

Suspend функции:

- о Преобразуются компилятором в switch и работу с Continuation

TO HIDE A THREAD POOL

Suspend функции:

- о Преобразуются компилятором в switch и работу с Continuation
- о Изначально появляются в библиотеке или как обертки над async API (см. `suspendCoroutine`)

TO HIDE A THREAD POOL

Suspend функции:

- о Преобразуются компилятором в switch и работу с Continuation
- о Изначально появляются в библиотеке или как обертки над async API (см. `suspendCoroutine`)
- о Caller suspend функции тоже suspend

TO HIDE A THREAD POOL

Suspend функции:

- Преобразуются **компилятором** в switch и работу с Continuation
- Изначально появляются в библиотеке или как обертки над async API (см. **suspendCoroutine**)
- Caller **suspend** функции тоже **suspend** (и ответственность за расстановку suspend лежит на программисте! Компилятор подскажет)

TO HIDE A THREAD POOL



TO HIDE A THREAD POOL

Получается, программисты
красят все функции в два
цвета: *suspend* и остальные.



TO HIDE A THREAD POOL

Получается, программисты красят все функции в два цвета: *suspend* и остальные.

А почему бы не "объявить" все функции *suspend*?



TO HIDE A THREAD POOL

А чем за это все заплатим?

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
fun validate() { ... }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
fun validate() { ... }
```

```
@Benchmark  
fun baseline() =  
    runBlocking {  
        test()  
    }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
fun validate() { ... }
```

```
@Benchmark  
fun baseline() =  
    runBlocking {  
        test()  
    }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337	± 0,027	us/op

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
suspend fun validate() {...}
```

```
@Benchmark  
fun baseline() =  
    runBlocking {  
        test()  
    }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337 ± 0,027	us/op	

TO HIDE A THREAD POOL

```
suspend fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
suspend fun validate() {...}
```

```
@Benchmark  
fun testSuspend() =  
runBlocking {  
    test()  
}
```

```
suspend fun foo() {  
    validate(...)  
    ...  
}  
  
suspend fun bar(i: Int) {  
    foo()  
    ...  
}  
  
suspend fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337	± 0,027	us/op

TO HIDE A THREAD POOL

```
suspend fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
suspend fun validate() {...}
```

```
@Benchmark  
fun testSuspend() =  
runBlocking {  
    test()  
}
```

```
suspend fun foo() {  
    validate(...)  
    ...  
}  
  
suspend fun bar(i: Int) {  
    foo()  
    ...  
}  
  
suspend fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337	± 0,027	us/op
testSuspend	3,843	± 0,278	us/op

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
fun validate() { ... }
```

```
@Benchmark  
fun baseline() =  
    runBlocking {  
        test()  
    }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337	± 0,027	us/op
testSuspend	3,843	± 0,278	us/op

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
fun validate() { ... }
```

```
@Benchmark  
fun baseline() =  
    runBlocking {  
        test()  
    }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337	± 0,027	us/op
testSuspend	3,843	± 0,278	us/op

А вдруг дело чисто в том, что после преобразования к CSP функции перестали инлайниться? (и тогда стоило бы просто проточить инлайн, чтобы это исправить)

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
fun validate() { ... }
```

```
@Benchmark  
fun baseline() =  
    runBlocking {  
        test()  
    }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337	± 0,027	us/op
testSuspend	3,843	± 0,278	us/op

А вдруг дело чисто в том, что после преобразования к CSP функции перестали инлайниться? (и тогда стоило бы просто проточить инлайн, чтобы это исправить)

Проверим!

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + l)  
}
```

```
fun validate() { ... }
```

```
@Benchmark  
fun testNoInline() =  
    runBlocking {  
        test()  
    }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337	± 0,027	us/op
testSuspend	3,843	± 0,278	us/op

Запретим inline на всех функциях, но уберем при этом suspend

```
@CompilerControl(CompilerControl.Mode.DONT_INLINE)
```

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + 1)  
}
```

```
fun validate() { ... }
```

```
@Benchmark  
fun testNoInline() =  
    runBlocking {  
        test()  
    }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337	± 0,027	us/op
testSuspend	3,843	± 0,278	us/op
testNoInline	2,677	± 0,075	us/op

Запретим инлайн на всех функциях, но уберем при этом suspend. Получилось уже сильно хуже, чем baseline, но с CSP – еще хуже 😞

`@CompilerControl(CompilerControl.Mode.DONT_INLINE)`

TO HIDE A THREAD POOL

```
fun test(): Int {  
    val r = foo()  
    validate(r)  
    val l = bar(r + 42)  
    validate(l)  
    return baz(r + 1)  
}
```

```
fun validate() { ... }
```

```
@Benchmark  
fun testNoInline() =  
    runBlocking {  
        test()  
    }
```

```
fun foo() {  
    validate(...)  
    ...  
}  
  
fun bar(i: Int) {  
    foo()  
    ...  
}  
  
fun baz(i: Int) {  
    validate(i)  
    ...  
}
```

Benchmark	Score	Error	Units
baseline	0,337	± 0,027	us/op
testSuspend	3,843	± 0,278	us/op
testNoInline	2,677	± 0,075	us/op

Запретим инлайн на всех функциях, но уберем при этом suspend. Получилось уже сильно хуже, чем baseline, но с CSP – еще хуже 😞

`@CompilerControl(CompilerControl.Mode.DONT_INLINE)`

TO HIDE A THREAD POOL

А чем за это все заплатим?

о Производительностью преобразованного кода



TO HIDE A THREAD POOL

А чем за это все заплатим?

- о Производительностью преобразованного кода
- о Памятью (из-за создания Continuation-ов)



TO HIDE A THREAD POOL

А чем за это все заплатим?

- о Производительностью преобразованного кода
- о Памятью (из-за создания Continuation-ов)

Так что все объявить suspend никак нельзя, все будет с трудом ползать (и потреблять огромное количество памяти)

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

Автоматическое преобразование кода в CPS

- Используется в C#, JS, C++, **Kotlin**

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

Автоматическое преобразование кода в CPS

- Используется в C#, JS, C++, **Kotlin**
- Реализация на уровне компилятора и библиотеки
 - + Минимальное изменение исходного кода
 - Издержки по производительности, по памяти

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ

Автоматическое преобразование кода в CPS

- Используется в C#, JS, C++, **Kotlin**
- Реализация на уровне компилятора и библиотеки
 - + Минимальное изменение исходного кода
 - Издержки по производительности, по памяти
- ± Раскраска кода (suspend функциями)
- Все еще хрупкое решение!



РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture

больше

Влияние на код и язык

меньше

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture

Автоматическое
преобразование кода
в CPS a.k.a stackless
корутины

больше

Влияние на код и язык

меньше

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture

Автоматическое
преобразование кода
в CPS a.k.a stackless
корутины

C#/JS/C++

Kotlin

больше

меньше

Влияние на код и язык

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture



Автоматическое
преобразование кода
в CPS a.k.a stackless
корутины

C#/JS/C++

Kotlin

больше

меньше

Влияние на код и язык



LOOM

LOOM

В Java мире давно хотели решить проблему ограничений ОС тредов, изменив реализацию `java.lang.Thread`

LOOM

В Java мире давно хотели решить проблему ограничений ОС тредов, изменив реализацию `java.lang.Thread`

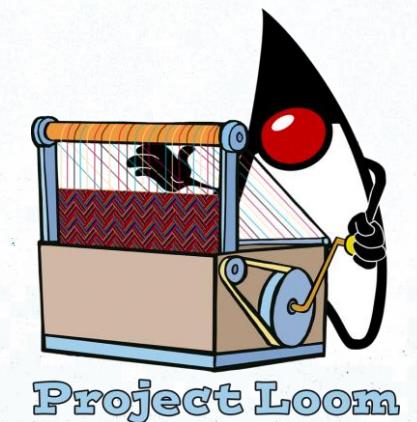
В 2017 году был начат проект Loom, вводящий в язык **виртуальные потоки**.

LOOM

В Java мире давно хотели решить проблему ограничений OS treadов, изменив реализацию `java.lang.Thread`

В 2017 году был начат проект Loom, вводящий в язык **виртуальные потоки**.

В Java 19 они появились, как preview фича, а в Java 20 (2023 год) зарелизились.



LOOM

```
final int count = 100_000;
final Thread[] threads = new Thread[count];

for (int i = 0; i < count; i++) {
    System.out.println("Starting thread #" + i);
    final int n = i;
    threads[i] = new Thread(() -> {
        Thread.sleep(Duration.ofSeconds(10));
        System.out.println("Thread #" + n + " finished");
    });
    threads[i].start();
}

for (int i = 0; i < count; i++) {
    threads[i].join();
}
```

LOOM

```
final int count = 100_000;
final Thread[] threads = new Thread[count];

for (int i = 0; i < count; i++) {
    System.out.println("Starting thread #" + i);
    final int n = i;
    threads[i] = new Thread(() -> {
        Thread.sleep(Duration.ofSeconds(10));
        System.out.println("Thread #" + n + " finished");
    });
    threads[i].start();
}

for (int i = 0; i < count; i++) {
    threads[i].join();
}
```

LOOM

```
final int count = 100_000;
final Thread[] threads = new Thread[count];
```

```
Starting thread #10775
Starting thread #10776
Starting thread #10777
Starting thread #10778
Starting thread #10779
Starting thread #10780
[1.232s][warning][os,thread] Failed to start thread "Unknown thread" -
pthread_create failed (EAGAIN) for attributes: stacksize: 1024k, guardsize: 0k, detached.
[1.232s][warning][os,thread] Failed to start the native thread for java.lang.Thread
"Thread-10780"
```

```
for (int i = 0; i < count; i++) {
    threads[i].join();
}
```

LOOM

```
final int count = 100_000;
final Thread[] threads = new Thread[count];

for (int i = 0; i < count; i++) {
    System.out.println("Starting thread #" + i);
    final int n = i;
    threads[i] = new Thread(() -> {
        Thread.sleep(Duration.ofSeconds(10));
        System.out.println("Thread #" + n + " finished");
    });
    threads[i].start();
}

for (int i = 0; i < count; i++) {
    threads[i].join();
}
```

LOOM

```
final int count = 100_000;
final Thread[] threads = new Thread[count];

for (int i = 0; i < count; i++) {
    System.out.println("Starting thread #" + i);
    final int n = i;
    threads[i] = Thread.ofVirtual().start(() -> {
        Thread.sleep(Duration.ofSeconds(10));
        System.out.println("Thread #" + n + " finished");
    });
}

for (int i = 0; i < count; i++) {
    threads[i].join();
}
```

LOOM

```
final int count = 100_000;
final Thread[] threads = new Thread[count];

Thread #99986 finished
Thread #99982 finished
Thread #99990 finished
Thread #99993 finished
Thread #99996 finished
Thread #99994 finished
Thread #99997 finished
Thread #99999 finished
Thread #99988 finished

    < count; i++) {
        System.out.println("Starting thread #" + i);
        i;
        Thread.ofVirtual().start(() -> {
            sleep(Duration.ofSeconds(10));
            System.out.println("Thread #" + n + " finished");
        });
    }

    for (int i = 0; i < count; i++) {
        threads[i].join();
    }
}
```

LOOM

```
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {  
    IntStream.range(0, 100_000).forEach(i -> {  
        executor.submit(() -> {  
            Thread.sleep(Duration.ofSeconds(1));  
            System.out.println("worker " + i + " finished");  
            return i;  
        });  
    });  
}
```

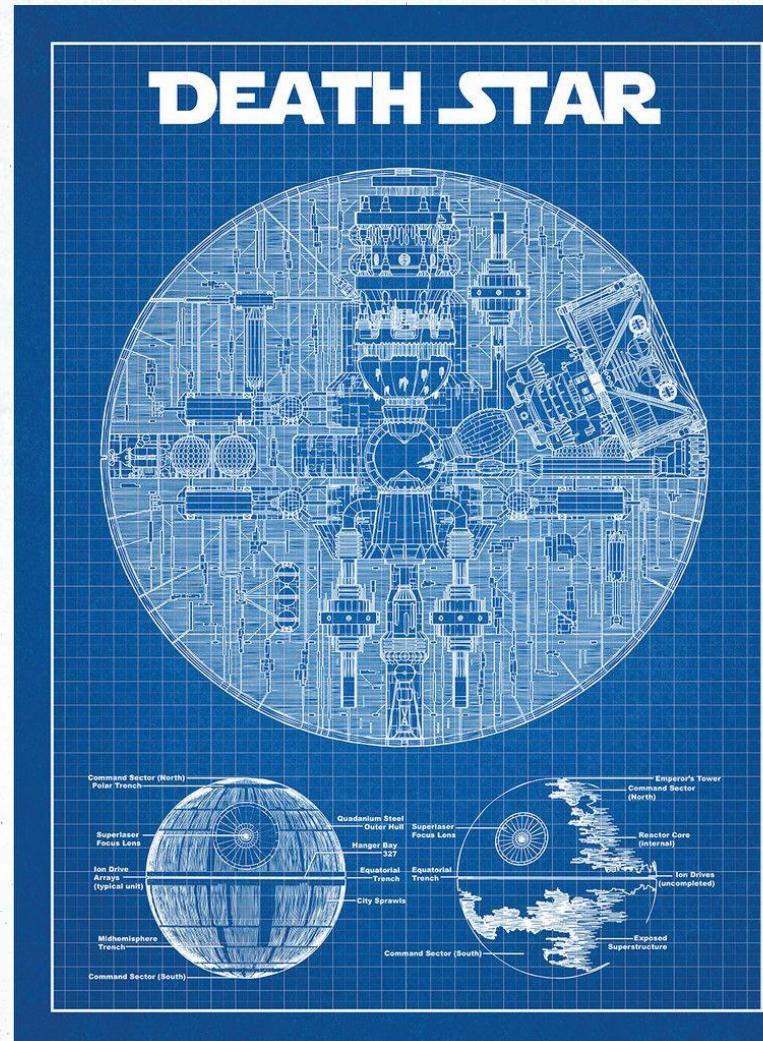
LOOM

```
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {  
    IntStream.range(0, 100_000).forEach(i -> {  
        executor.submit(() -> {  
            Thread.sleep(Duration.ofSeconds(1));  
            System.out.println("worker " + i + " finished");  
            return i;  
        });  
    });  
}
```

worker #99989 finished
worker #99982 finished
worker #99991 finished
worker #99994 finished
worker #99993 finished
worker #99999 finished
worker #99995 finished
worker #99992 finished
worker #99998 finished

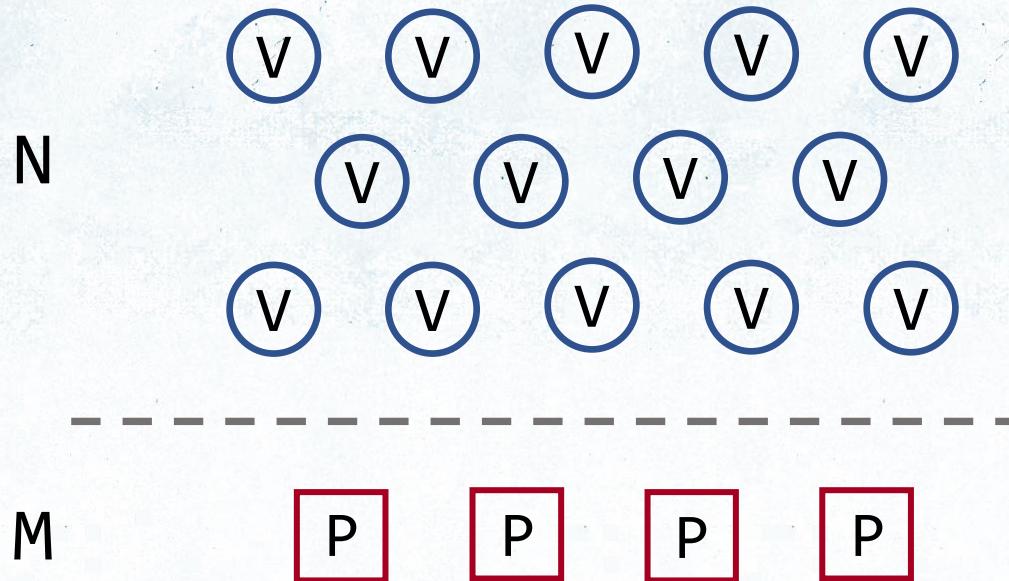


LOOM: UNDER THE HOOD



LOOM: UNDER THE HOOD

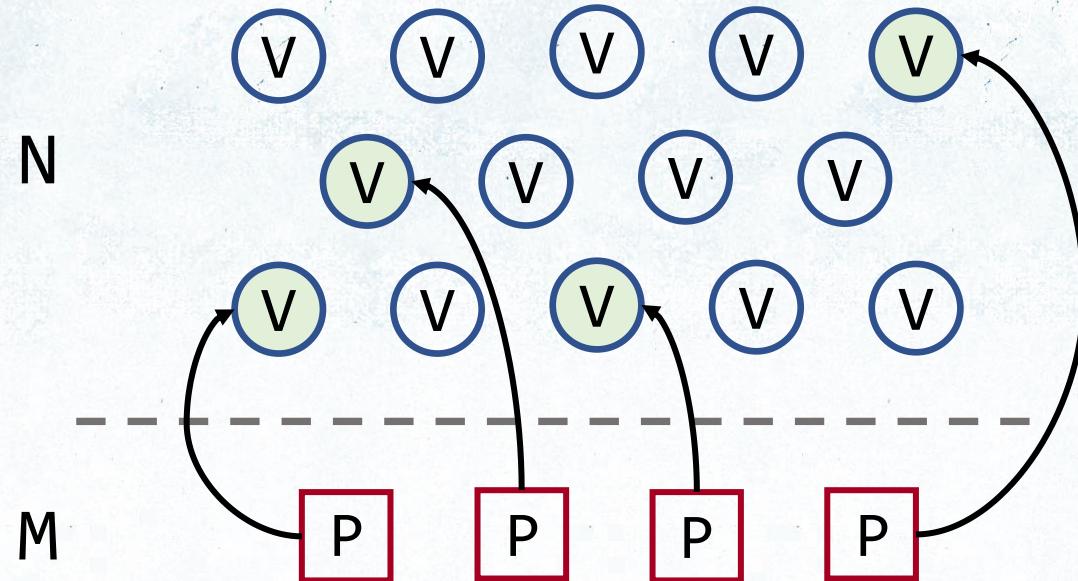
виртуальные потоки



o N:M threading

LOOM: UNDER THE HOOD

виртуальные потоки

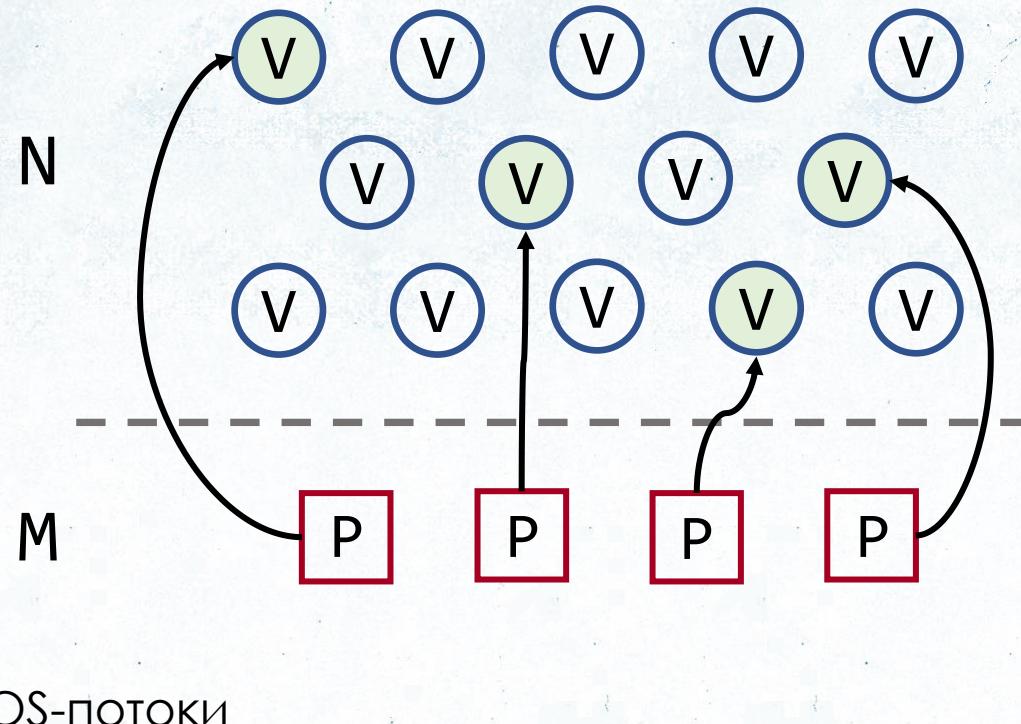


OS-потоки

- N:M threading
- Виртуальный поток:
 - исполняется (на OS-потоке)
 - ожидает

LOOM: UNDER THE HOOD

виртуальные потоки

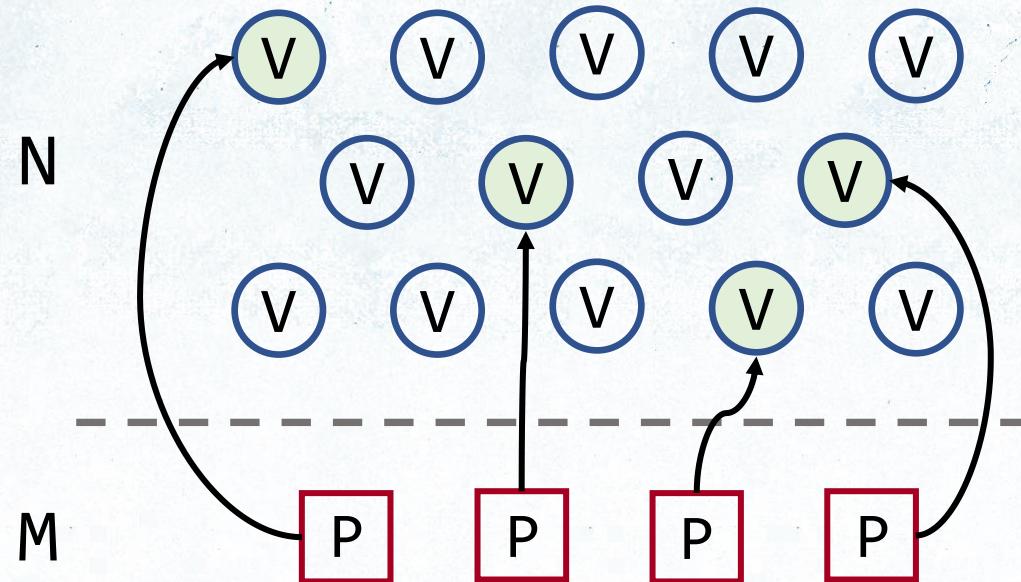


OS-потоки

- N:M threading
- Виртуальный поток:
 - исполняется (на OS-потоке)
 - ожидает
 - Снятие с OS-потока кооперативное

LOOM: UNDER THE HOOD

виртуальные потоки



OS-потоки

- N:M threading
- Виртуальный поток:
 - исполняется (на OS-потоке)
 - ожидает
- Снятие с OS-потока кооперативное
- Планировщик решает, кто следующий

LOOM: UNDER THE HOOD

Когда виртуальный поток снимается с OS потока?



LOOM: UNDER THE HOOD

Когда виртуальный поток снимается с OS потока?

- Thread.yield()

```
public static void yield() {
    if (currentThread() instanceof VirtualThread vthread) {
        vthread.tryYield();
    } else {
        yield0();
    }
}
```

LOOM: UNDER THE HOOD

Когда виртуальный поток снимается с OS потока?

- Thread.yield(), Thread.sleep(...)

LOOM: UNDER THE HOOD

Когда виртуальный поток снимается с OS потока?

- `Thread.yield()`, `Thread.sleep(...)`
- `LockSupport.park(...)`

LOOM: UNDER THE HOOD

Когда виртуальный поток снимается с OS потока?

- Thread.yield(), Thread.sleep(...)
- LockSupport.park(...)

```
public static void park(Object blocker) {
    Thread t = Thread.currentThread();
    setBlocker(t, blocker);
    try {
        if (t.isVirtual()) {
            VirtualThreads.park();
        } else {
            U.park(false, 0L);
        }
    } finally {
        setBlocker(t, null);
    }
}
```

LOOM: UNDER THE HOOD

Когда виртуальный поток снимается с OS потока?

- Thread.yield(), Thread.sleep(...)
- LockSupport.park(...)
- ReentrantLock, BlockingQueue, Mutex

LOOM: UNDER THE HOOD

Когда виртуальный поток снимается с OS потока?

- Thread.yield(), Thread.sleep(...)
- LockSupport.park(...)
 - ReentrantLock, BlockingQueue, Mutex
- java.net/java.nio.channels

LOOM: UNDER THE HOOD

Когда виртуальный поток снимается с OS потока?

- Thread.yield(), Thread.sleep(...)
 - LockSupport.park(...)
 - ReentrantLock, BlockingQueue, Mutex
 - java.net/java.nio.channels
- "Блокирующие" чтение и запись по сети вместо блокировки OS-потока снимают виртуальный поток

LOOM: UNDER THE HOOD

```
public int read(ByteBuffer buf) throws IOException {  
    ...  
    configureSocketNonBlockingIfVirtualThread();  
    n = IOUtil.read(fd, buf, -1, nd);  
    if (blocking) {  
        while (IOStatus.okayToRetry(n) && isOpen()) {  
            park(Net.POLLIN);  
            n = IOUtil.read(fd, buf, -1, nd);  
        }  
    }  
    ...  
}
```

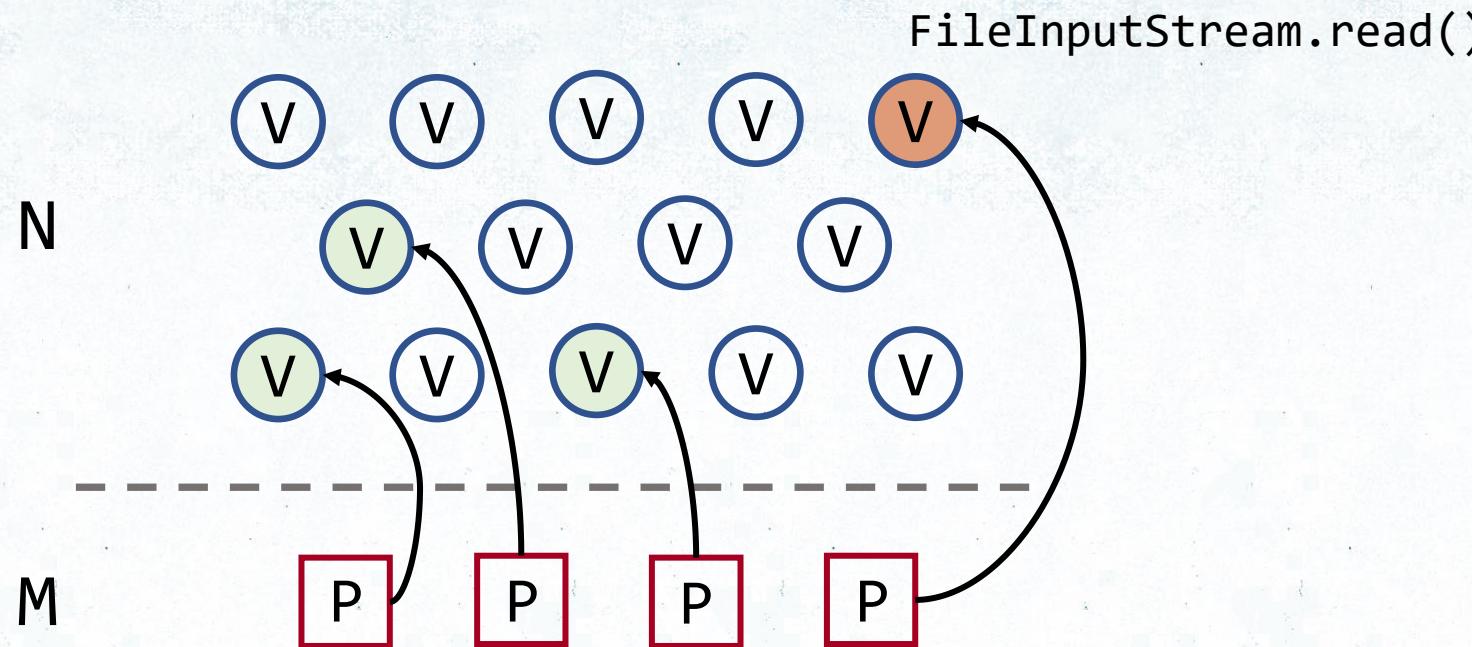
sun.nio.ch.SocketChannelImpl.java

LOOM: UNDER THE HOOD

А что, если поток заблокируется на IO?

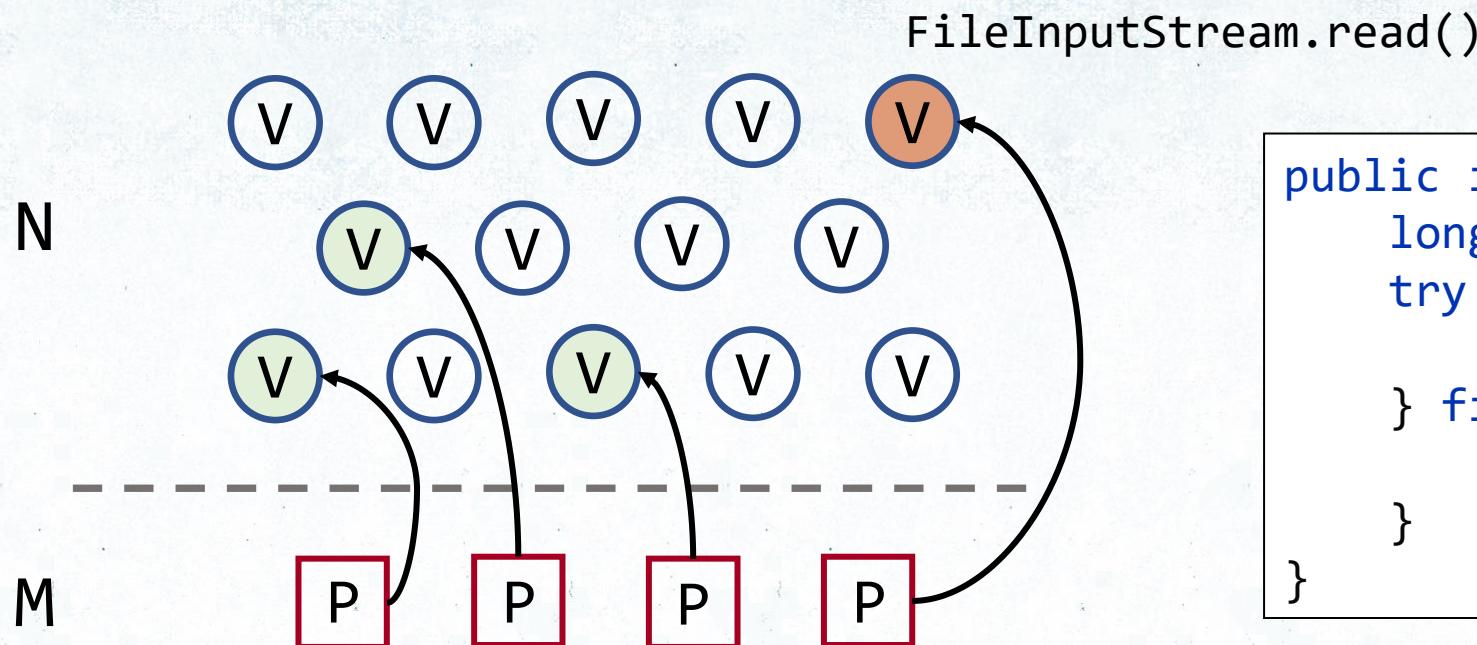
LOOM: UNDER THE HOOD

А что, если поток заблокируется на IO?



LOOM: UNDER THE HOOD

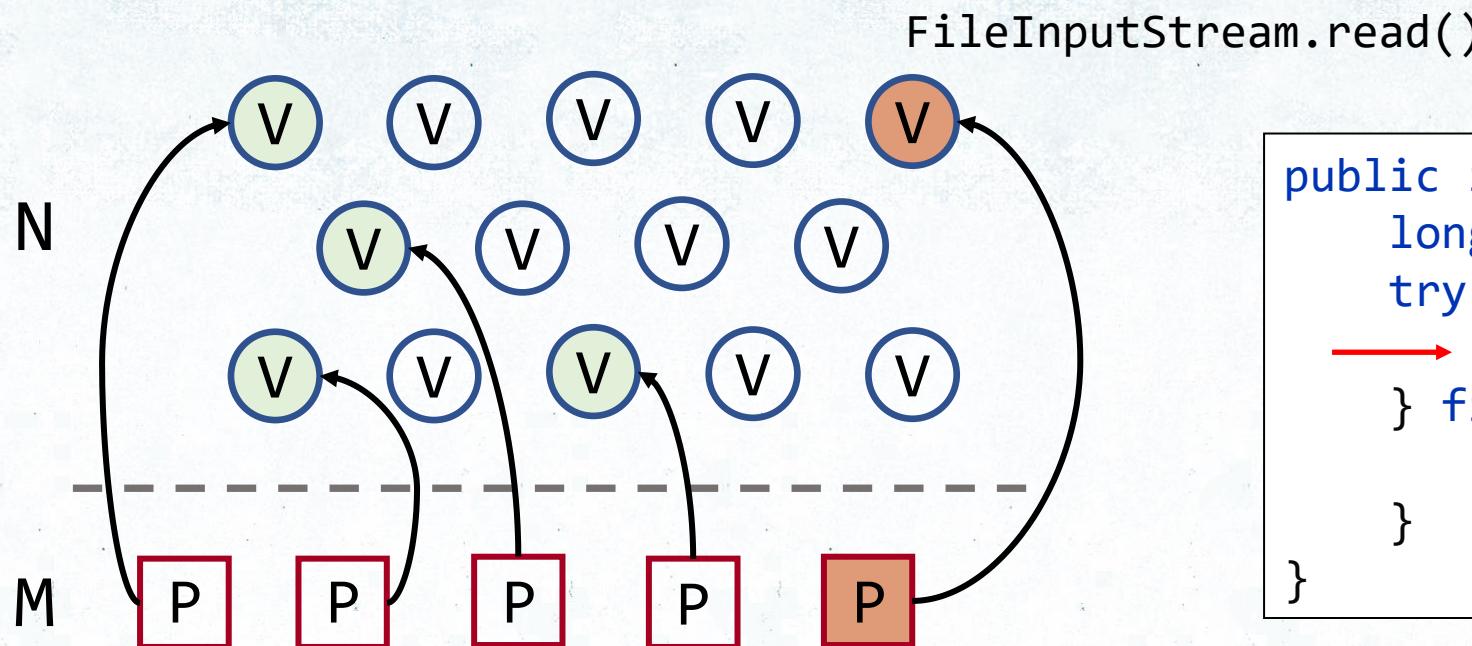
А что, если поток заблокируется на IO?



```
public int read() throws IOException {  
    long comp = Blocker.begin();  
    try {  
        return read0();  
    } finally {  
        Blocker.end(comp);  
    }  
}
```

LOOM: UNDER THE HOOD

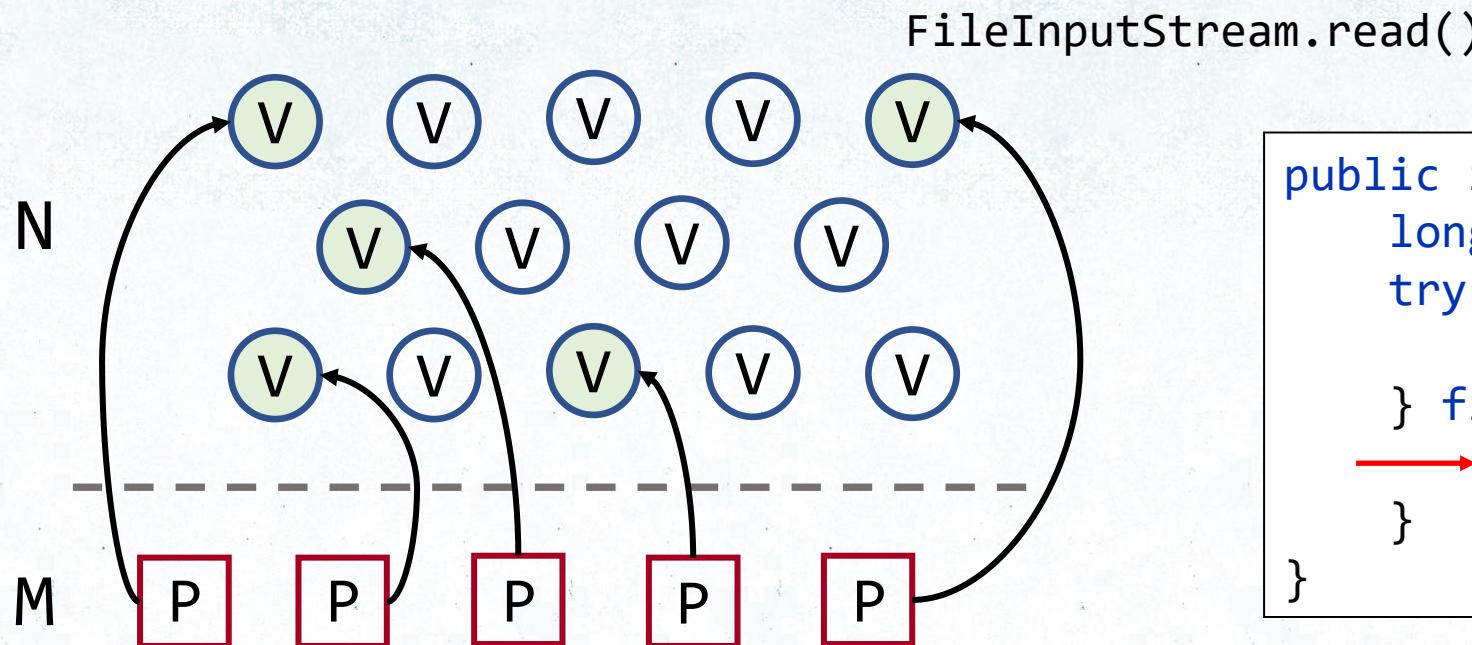
А что, если поток заблокируется на IO?



```
public int read() throws IOException {  
    long comp = Blocker.begin();  
    try {  
        → return read0();  
    } finally {  
        Blocker.end(comp);  
    }  
}
```

LOOM: UNDER THE HOOD

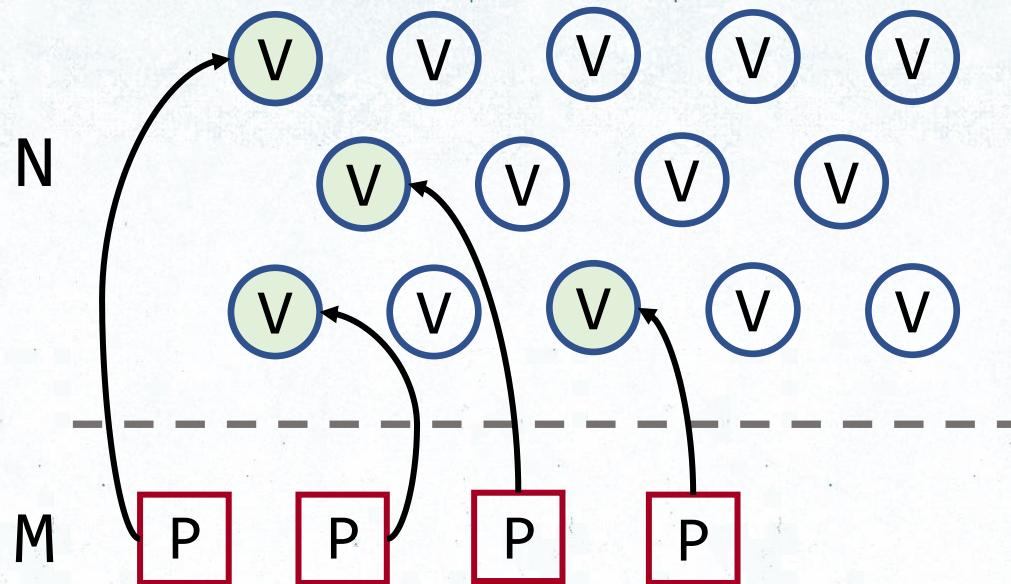
А что, если поток заблокируется на IO?



```
public int read() throws IOException {  
    long comp = Blocker.begin();  
    try {  
        return read0();  
    } finally {  
        → Blocker.end(comp);  
    }  
}
```

LOOM: UNDER THE HOOD

А что, если поток заблокируется на IO?



```
public int read() throws IOException {  
    long comp = Blocker.begin();  
    try {  
        return read0();  
    } finally {  
        Blocker.end(comp);  
    }  
}
```

LOOM: UNDER THE HOOD

При блокирующих операциях:

- Либо виртуальные потоки снимаются с OS-потоков
- Либо временно добавляются новые OS-потоки

LOOM: UNDER THE HOOD

При блокирующих операциях:

- Либо виртуальные потоки снимаются с OS-потоков
- Либо временно добавляются новые OS-потоки

Но есть **исключение!** Догадаетесь, какое?

LOOM: UNDER THE HOOD

При блокирующих операциях:

- Либо виртуальные потоки снимаются с OS-потоков
- Либо временно добавляются новые OS-потоки

Но есть **исключение!** Догадаетесь, какое?

Если хоть один из фреймов в вашем стеке вызовов
навигативный: Java => C++ => JNI => Java => ... => Thread.yield()

LOOM: UNDER THE HOOD

При блокирующих операциях:

- Либо виртуальные потоки снимаются с OS-потоков
- Либо временно добавляются новые OS-потоки

Но есть **исключение!** Догадаетесь, какое?

Если хоть один из фреймов в вашем стеке вызовов
нативный: Java => C++ => JNI => Java => ... => Thread.yield()

...то снятия виртуального потока не произойдет. Позже
станет понятно, почему.

LOOM: UNDER THE HOOD

При блокирующих операциях:

- Либо виртуальные потоки снимаются с OS-потоков
- Либо временно добавляются новые OS-потоки

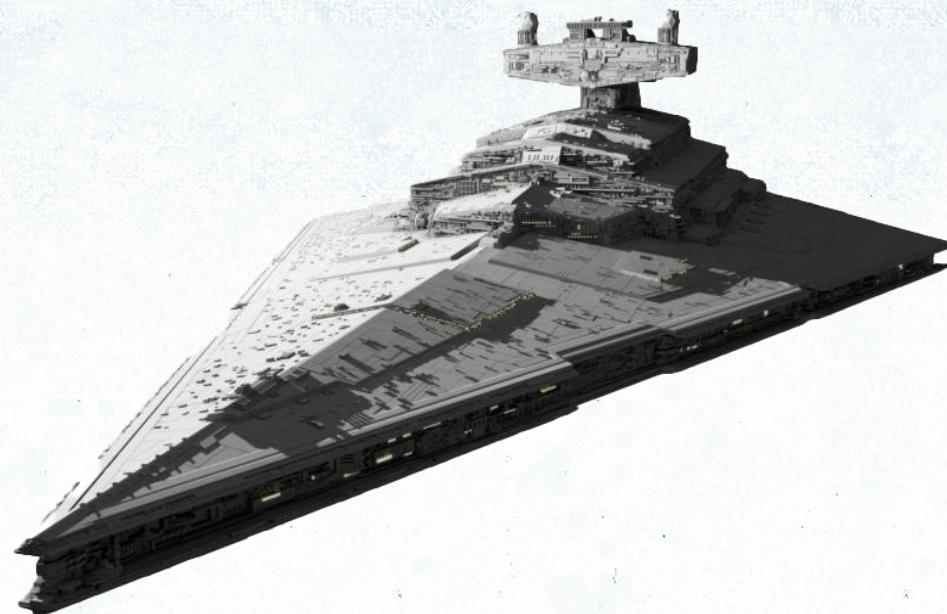
Но есть **исключение!** Догадаетесь, какое?

Если хоть один из фреймов в вашем стеке вызовов
нативный: Java => C++ => JNI => Java => ... => Thread.yield()

Раньше снятие не работало внутри synchronized блоков, но
это починили в Java 21 

LOOM: UNDER THE HOOD

А что, если в виртуальном потоке исполняется
просто долгий CPU intensive код?



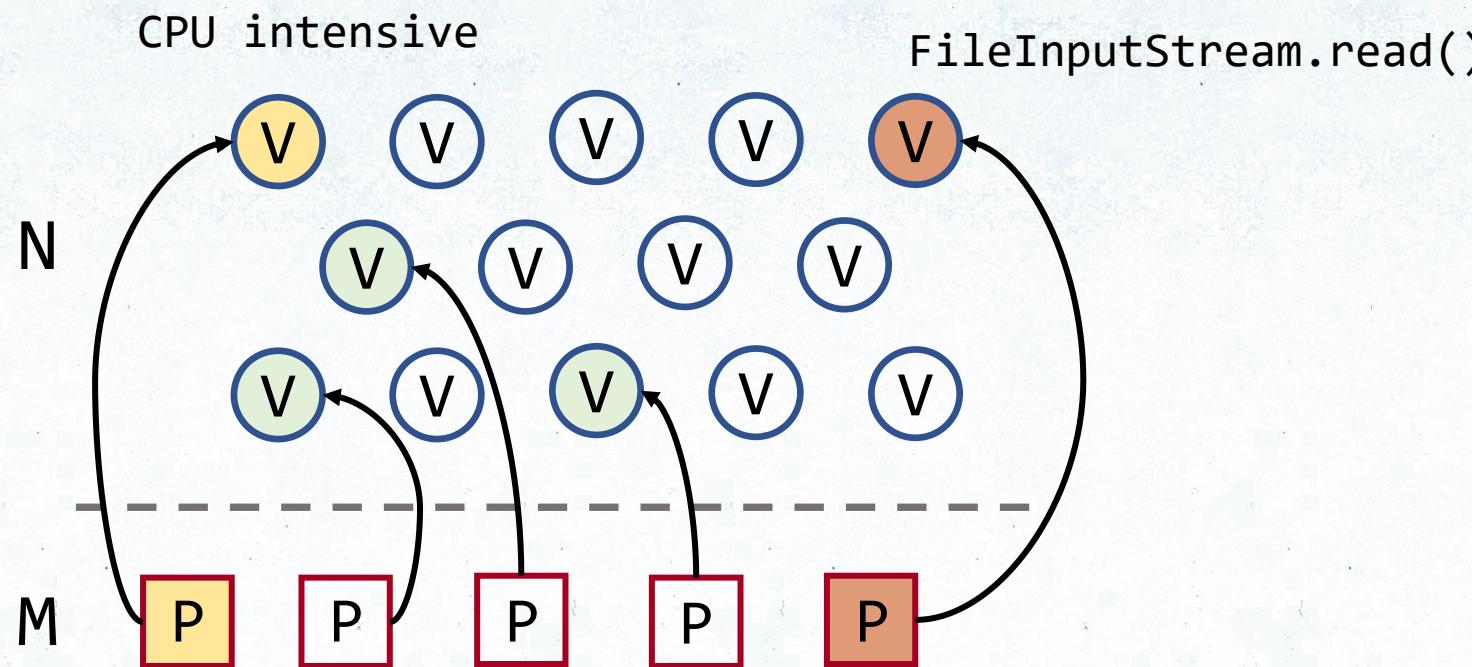
LOOM: UNDER THE HOOD

А что, если в виртуальном потоке исполняется просто долгий CPU intensive код?

```
for (int i = 0; i < 100_000; i++) {  
    for (int j = 0; j < 100_000; j++) {  
        while (z > 0) {  
            // CPU intensive делишки  
        }  
    }  
}
```

LOOM: UNDER THE HOOD

А что, если в виртуальном потоке исполняется просто долгий CPU intensive код?



LOOM: UNDER THE HOOD

```
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
    IntStream.range(0, 100_000).forEach(i -> {
        executor.submit(() -> {
            Thread.sleep(Duration.ofSeconds(1));
            System.out.println("worker " + i + " finished");
            return i;
        });
    });
}
```

LOOM: UNDER THE HOOD

```
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
    IntStream.range(0, 100_000).forEach(i -> {
        executor.submit(() -> {
            if (i < 8) {
                long result = getNthPrimeNumber(1_000_000 + i);
                System.out.println("Heavy worker " + i + " finished");
                return (int) result;
            } else {
                Thread.sleep(Duration.ofSeconds(1));
                System.out.println("worker " + i + " finished");
                return i;
            });
        });
    });
}
```

LOOM: UNDER THE HOOD

```
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {  
    IntStream.range(0, 100_000).forEach(i -> {  
        executor.submit(() -> {  
            if (i < 8) {  
                long result = getNthPrimeNumber(1_000_000);  
                System.out.println("Heavy worker " + i + " result: " + result);  
                return (int) result;  
            } else {  
                Thread.sleep(Duration.ofSeconds(1));  
                System.out.println("worker " + i + " sleeping");  
                return i;  
            }  
        });  
    });  
}
```

Heavy worker 6 finished
Heavy worker 4 finished
Heavy worker 0 finished
Heavy worker 1 finished
Heavy worker 5 finished
Heavy worker 3 finished
Heavy worker 2 finished
Heavy worker 7 finished
worker 14 finished
worker 15 finished
worker 51 finished
worker 53 finished
worker 55 finished
...

LOOM: UNDER THE HOOD

А ЧТО, ЕСЛИ В ВИРТУАЛЬНОМ ПОТОКЕ ИСПОЛНЯЕТСЯ ПРОСТО ДОЛГИЙ CPU intensive код?

Virtual threads

Thread also supports the creation of *virtual threads*. Virtual threads are typically *user-mode threads* scheduled by the Java runtime rather than the operating system. Virtual threads will typically require few resources and a single Java virtual machine may support millions of virtual threads. Virtual threads are suitable for executing tasks that spend most of the time blocked, often waiting for I/O operations to complete. Virtual threads are not intended for long running CPU intensive operations.

LOOM: UNDER THE HOOD

А что, если в виртуальном потоке исполняется просто долгий CPU intensive код?

- Это может быть проблемой (несколько CPU intensive потоков забьют весь пул)

LOOM: UNDER THE HOOD

А что, если в виртуальном потоке исполняется просто долгий CPU intensive код?

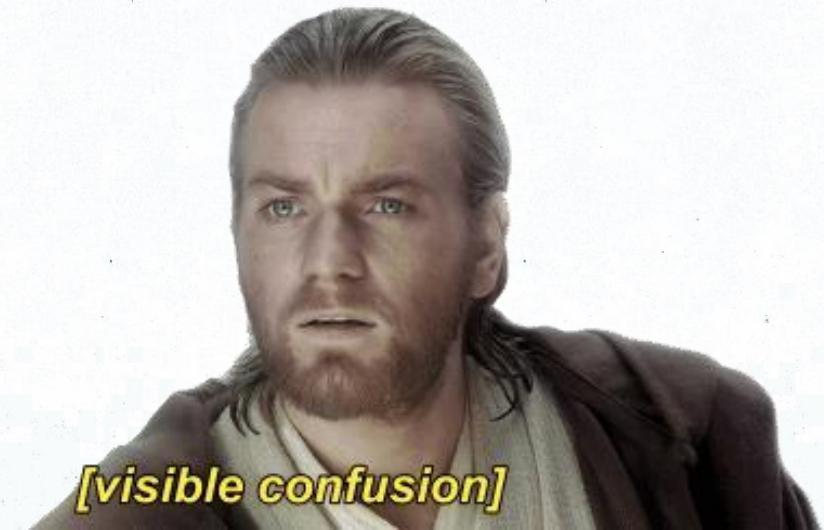
- Это может быть проблемой (несколько CPU intensive потоков забывают весь пул)
- Лечение:
 - Ручной Thread.yield()
 - Использование старых тредов

A dense collage of Star Wars characters and vehicles against a space background. In the foreground, a large crowd includes Obi-Wan Kenobi, Anakin Skywalker, Padmé Amidala, Yoda, Luke Skywalker, Han Solo, Chewbacca, Darth Vader, and various alien species like Wookiees, Gungan, and Ewoks. Above them, a massive Death Star looms over a fleet of X-wing and TIE fighters. In the center, a large AT-AT walker stands prominently. To the right, a massive Rancor monster is visible. The background is filled with more ships, a planet, and a bright starburst.

**THEY ARE
BILLIONS**

BETTER THREADS

Но почему миллион виртуальных потоков создать можно, а миллион OS-тредов – нет?



[visible confusion]

BETTER THREADS

Но почему миллион виртуальных потоков создать можно, а миллион OS-тредов – нет?

Стеки!



[visible happiness]

THE PROBLEM

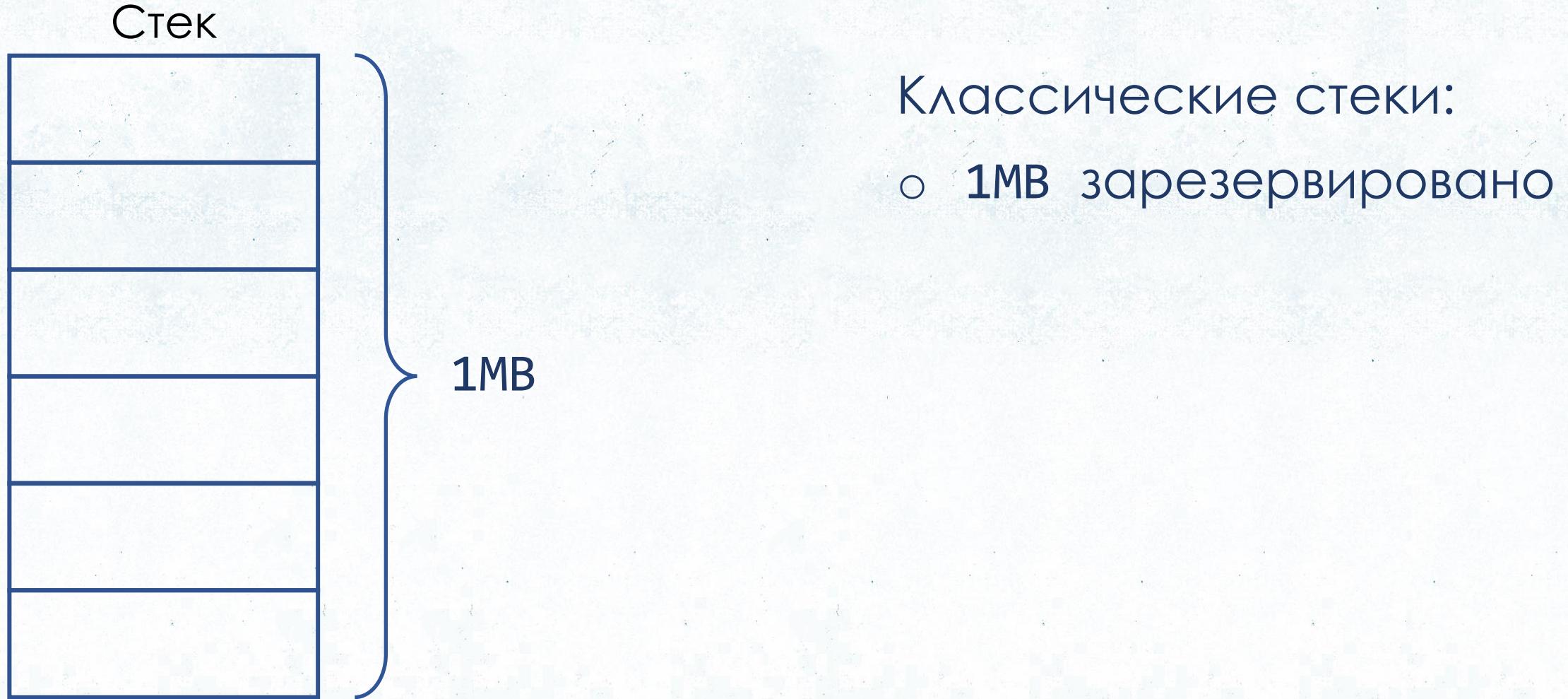
А как реализовать `java.lang.Thread`?

Чего вообще хотим от этого класса:

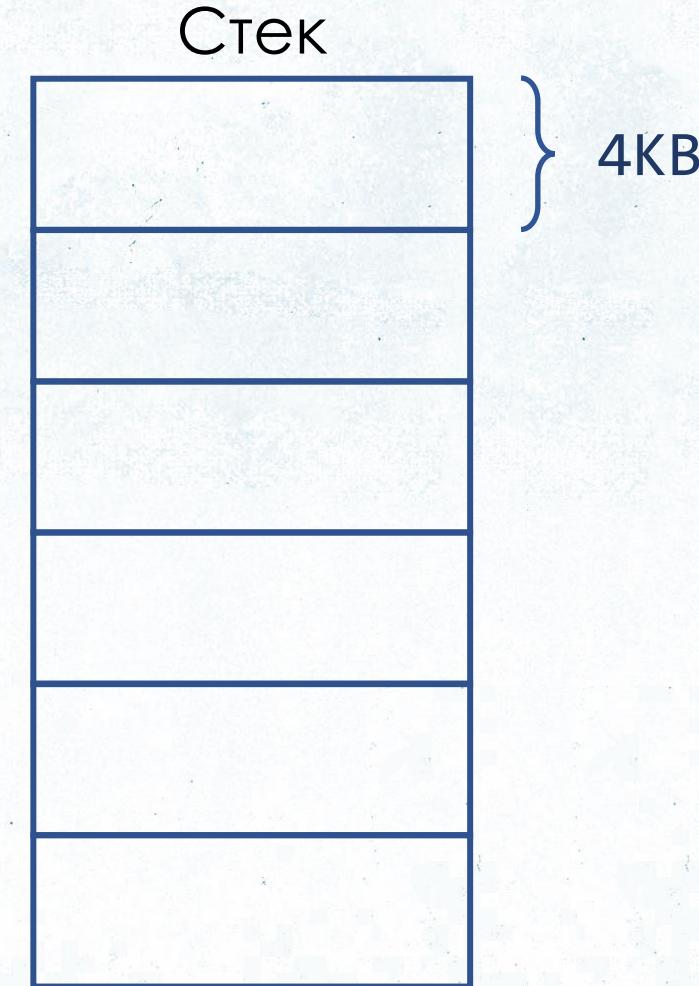
1. Оркестратор (**планировщик**), который будет решать, кому исполняться,
2. Система управления стеками,
3. Остаемся в рамках одного процесса



BETTER THREADS



BETTER THREADS

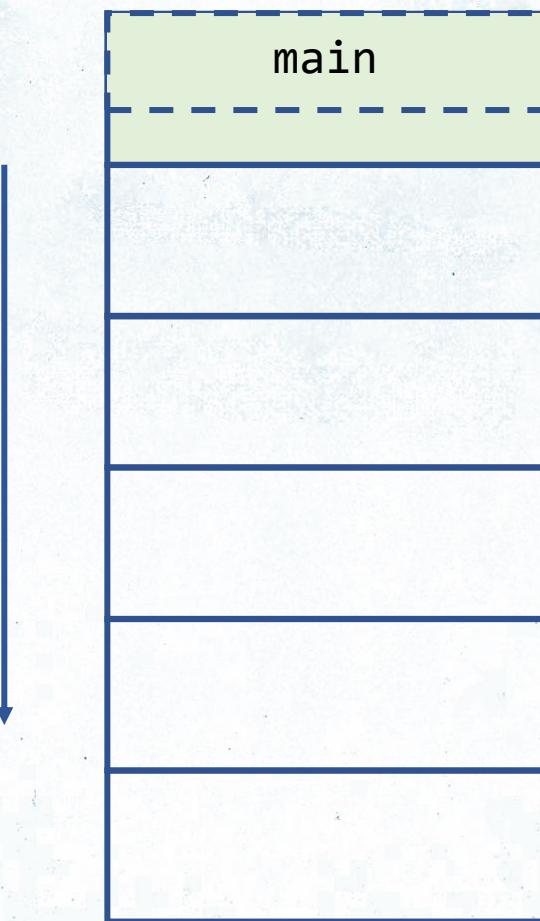


Классические стеки:

- 1МВ зарезервировано
- Большая часть страниц не подключена
- Страницы по 4KB или больше

BETTER THREADS

Стек



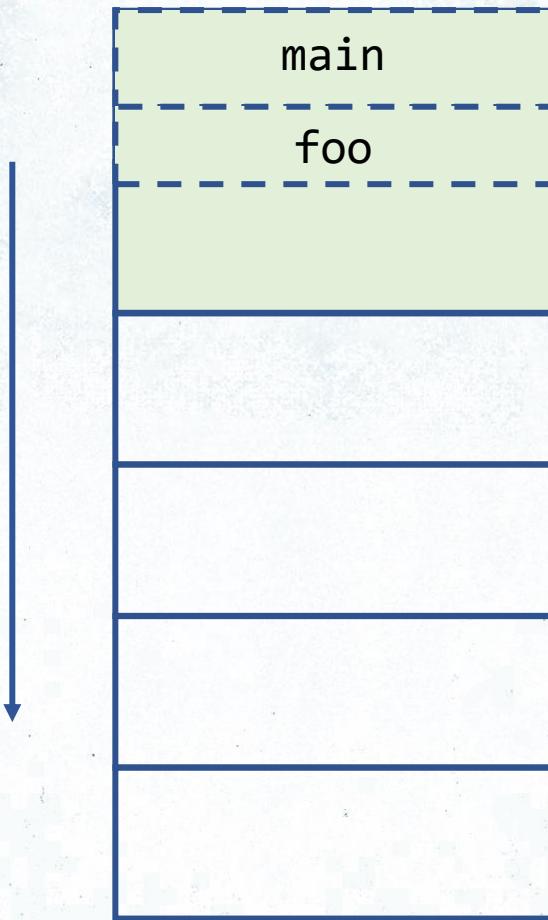
main()

Классические стеки:

- 1МВ зарезервировано
- Большая часть страниц не подключена
- Страницы по 4КВ или больше
- Подключаем лениво

BETTER THREADS

Стек



main()
↓
foo()

- Классические стеки:
- 1МВ зарезервировано
 - Большая часть страниц не подключена
 - Страницы по 4КВ или больше
 - Подключаем лениво

BETTER THREADS



main()
↓
foo()
↓
bar()

- Классические стеки:
- 1МВ зарезервировано
 - Большая часть страниц не подключена
 - Страницы по 4КВ или больше
 - Подключаем лениво

BETTER THREADS

Стек



main()
↓
foo()
↓
bar()
↓
baz()

- Классические стеки:
- 1МВ зарезервировано
 - Большая часть страниц не подключена
 - Страницы по 4КВ или больше
 - Подключаем лениво

BETTER THREADS

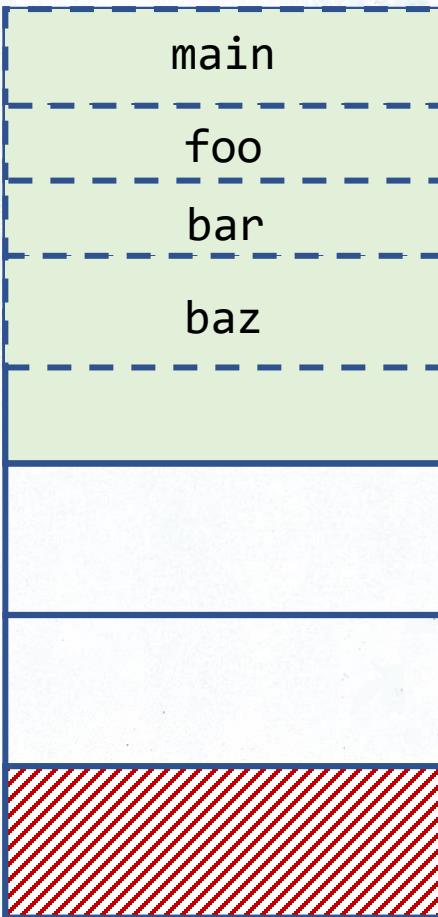


main()
↓
foo()
↓
bar()
↓
baz()

- Классические стеки:
- 1МВ зарезервировано
 - Большая часть страниц не подключена
 - Страницы по 4КВ или больше
 - Подключаем лениво
 - В конце guard page

BETTER THREADS

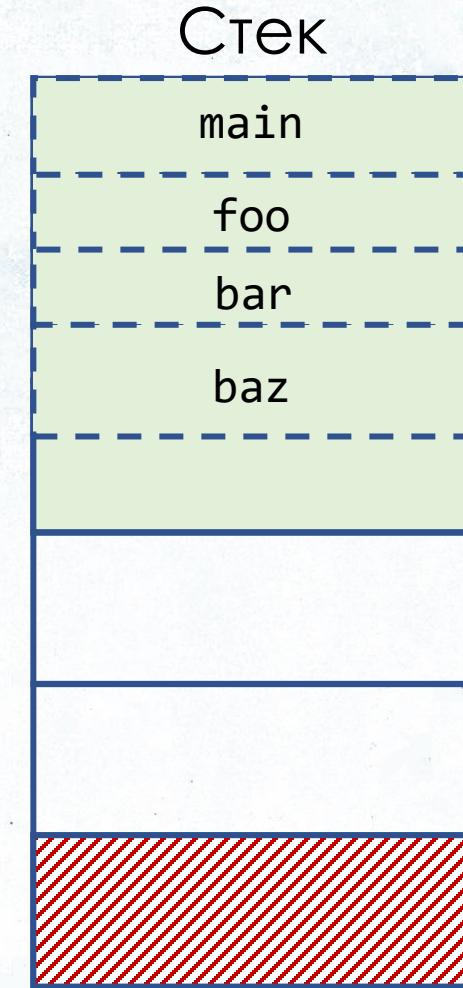
Стек



main()
↓
foo()
↓
bar()
↓
baz()

Почему не сопоставить
такой стек каждому
виртуальному потоку?

BETTER THREADS



main()
↓
foo()
↓
bar()
↓
baz()

Почему не сопоставить
такой стек каждому
виртуальному потоку?

- Дорогое создание

BETTER THREADS



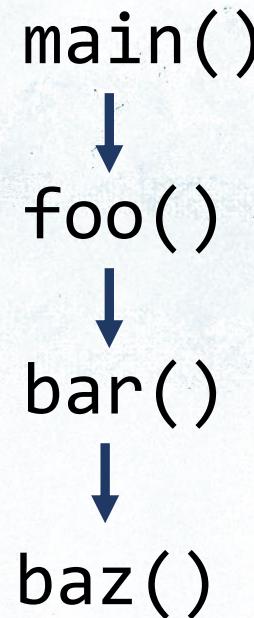
main()
↓
foo()
↓
bar()
↓
baz()

Почему не сопоставить
такой стек каждому
виртуальному потоку?

- Дорогое создание
- Потребление памяти
(страницы бывают и по 64к)

BETTER THREADS

Стек

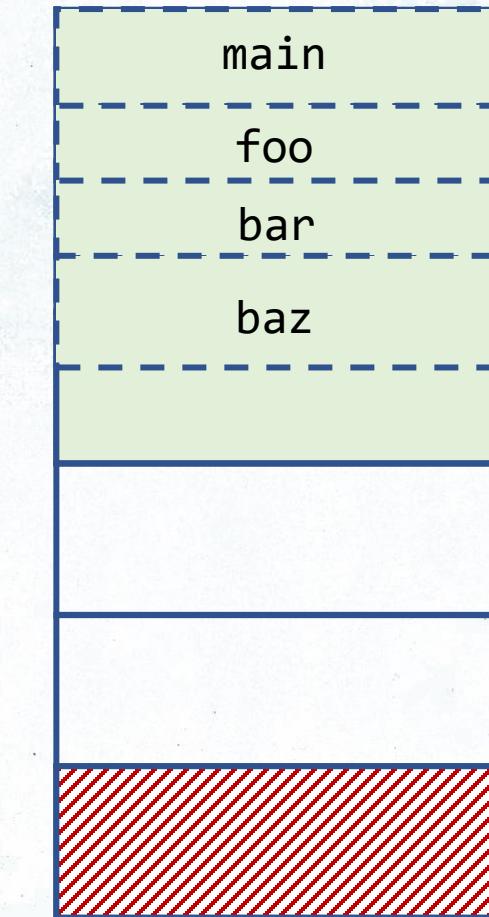


Почему не сопоставить
такой стек каждому
виртуальному потоку?

- Дорогое создание
- Потребление памяти
(страницы бывают и по 64к)
- Адресное пространство
конечно

BETTER THREADS

Стек



main()
↓
foo()
↓
bar()
↓
baz()

Почему не сопоставить
такой стек каждому
виртуальному потоку?

- Дорогое создание
- Потребление памяти
(страницы бывают и по 64к)
- Адресное пространство
конечно
- Ограничения OS
(vm.max_map_count)

BETTER THREADS

```
final class VirtualThread extends Thread {  
    ...  
    // scheduler and continuation  
    private final Executor scheduler;  
    private final Continuation cont;  
    private final Runnable runContinuation;  
    ...  
}
```



BETTER THREADS

```
final class VirtualThread extends Thread {  
    ...  
    // scheduler and continuation  
    private final Executor scheduler;  
    private final Continuation cont;  
    private final Runnable runContinuation;  
    ...  
}
```

```
public class Continuation {  
    ...  
    private StackChunk tail;  
    ...  
}
```



BETTER THREADS

```
final class VirtualThread extends Thread {  
    ...  
    // scheduler and continuation  
    private final Executor scheduler;  
    private final Continuation cont;  
    private final Runnable runContinuation;  
    ...  
}
```

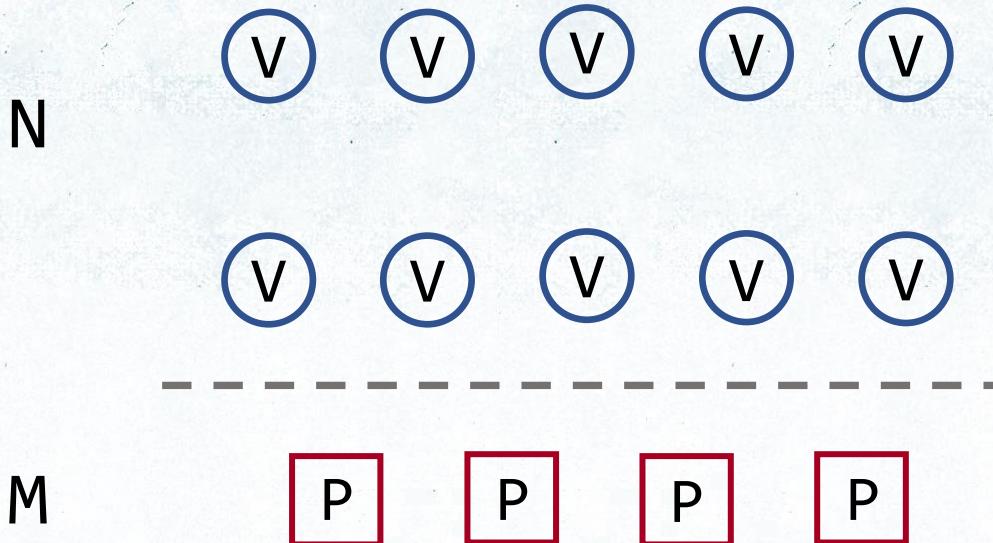
```
public class Continuation {  
    ...  
    private StackChunk tail;  
    ...  
}
```

```
public final class StackChunk {  
    public static void init() {}  
  
    private StackChunk parent;  
    private int size;      // in words  
    private int sp;        // in words  
    private int argsize;  // bottom stack-passed arguments, in words  
  
    // The stack itself is appended here by the VM, as well as some injected fields  
}
```



BETTER THREADS

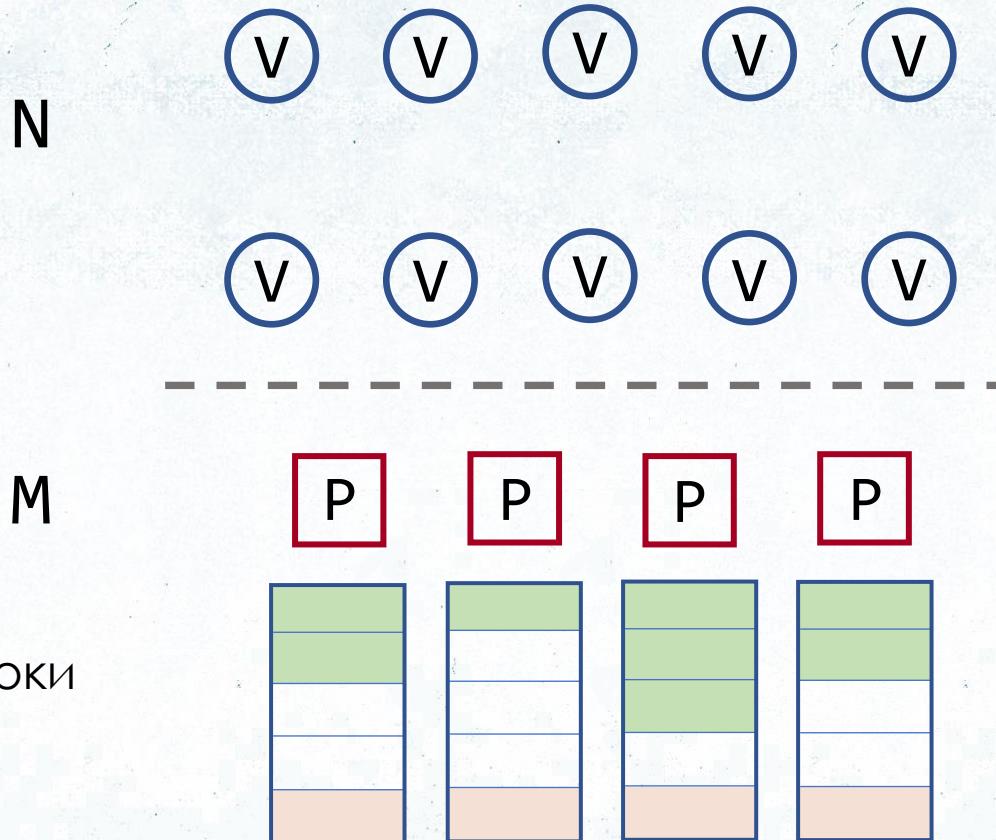
виртуальные потоки



OS-потоки

BETTER THREADS

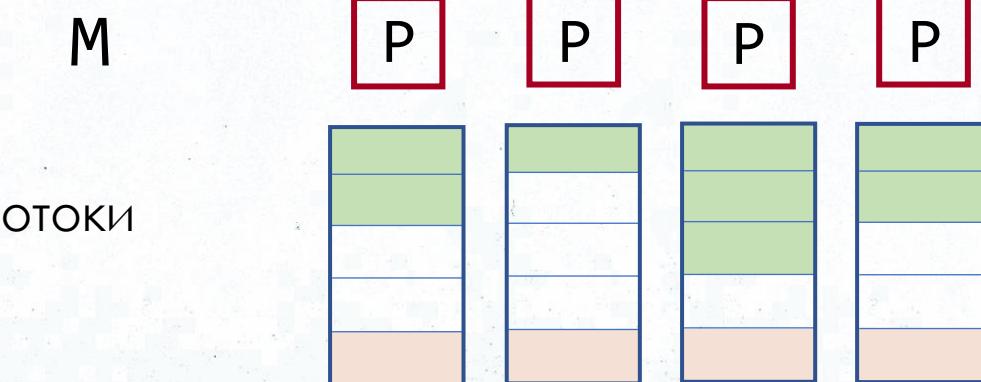
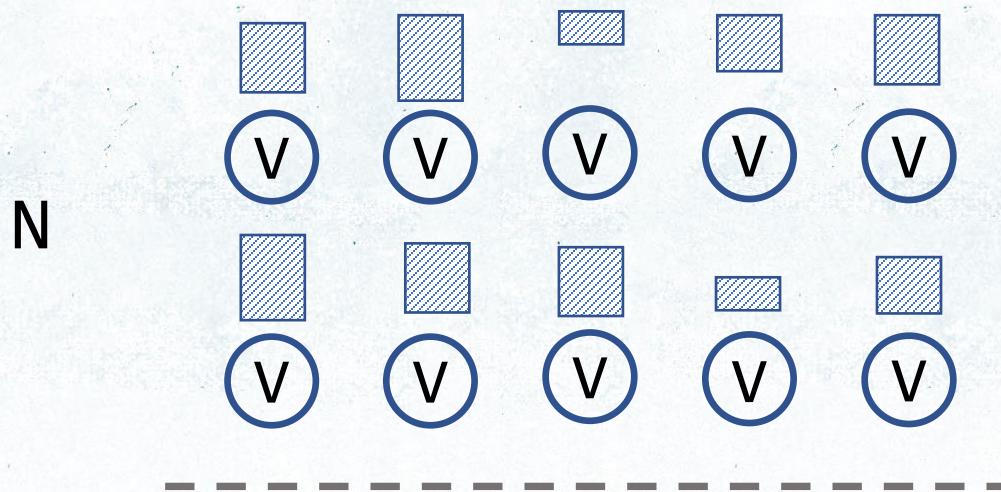
виртуальные потоки



- о Классические стеки у потоков носителей

BETTER THREADS

виртуальные потоки

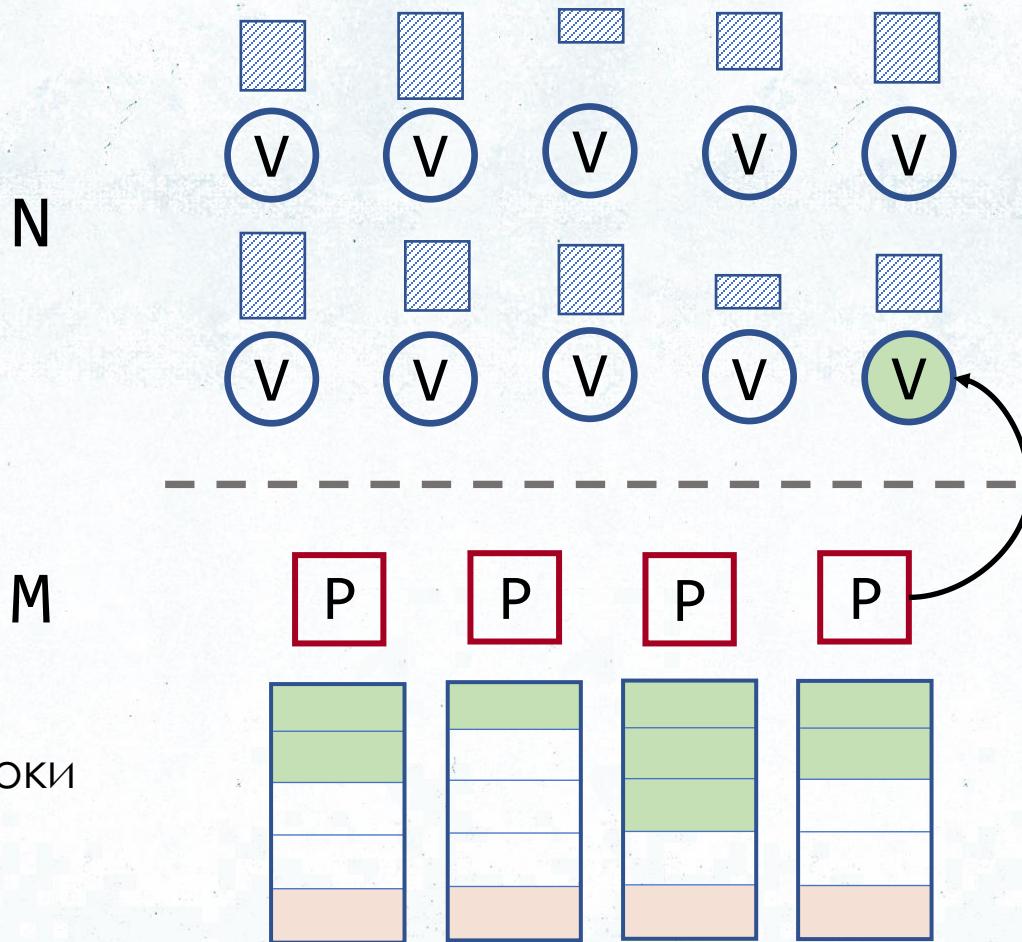


OS-потоки

- Классические стеки у потоков носителей
- Собственные маленькие стеки в хипе у виртуальных потоков

BETTER THREADS

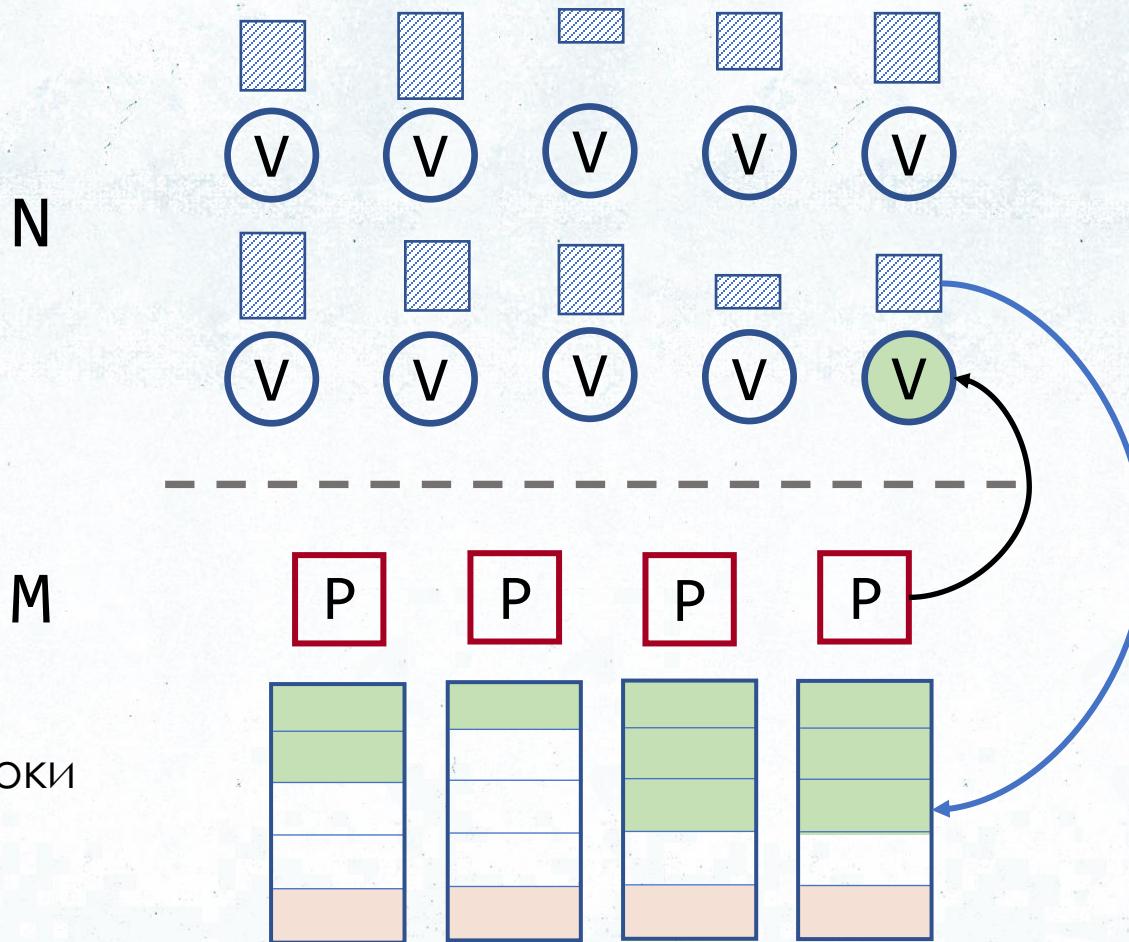
виртуальные потоки



- Классические стеки у потоков носителей
- Собственные маленькие стеки в хипе у виртуальных потоков
- Копирование стека при возобновлении

BETTER THREADS

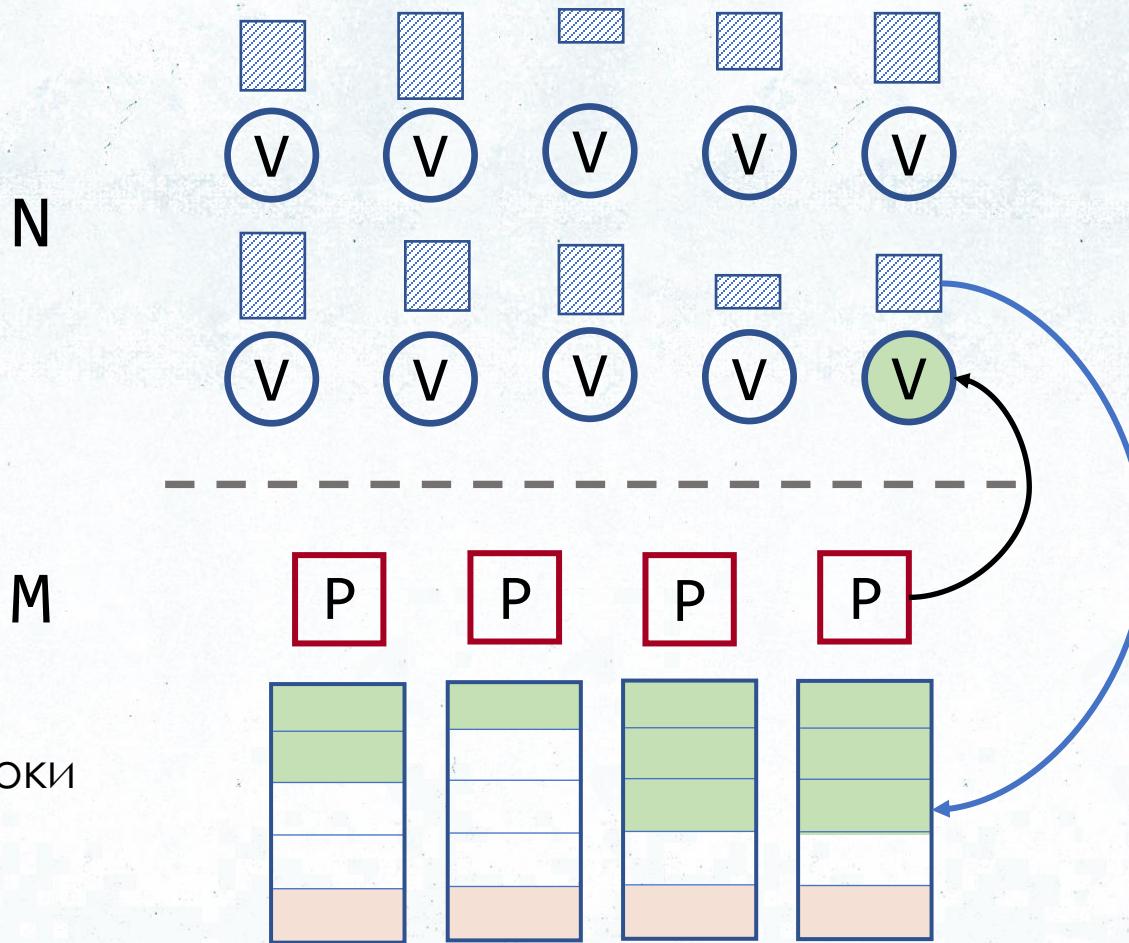
виртуальные потоки



- Классические стеки у потоков носителей
- Собственные маленькие стеки в хипе у виртуальных потоков
- Копирование стека при возобновлении

BETTER THREADS

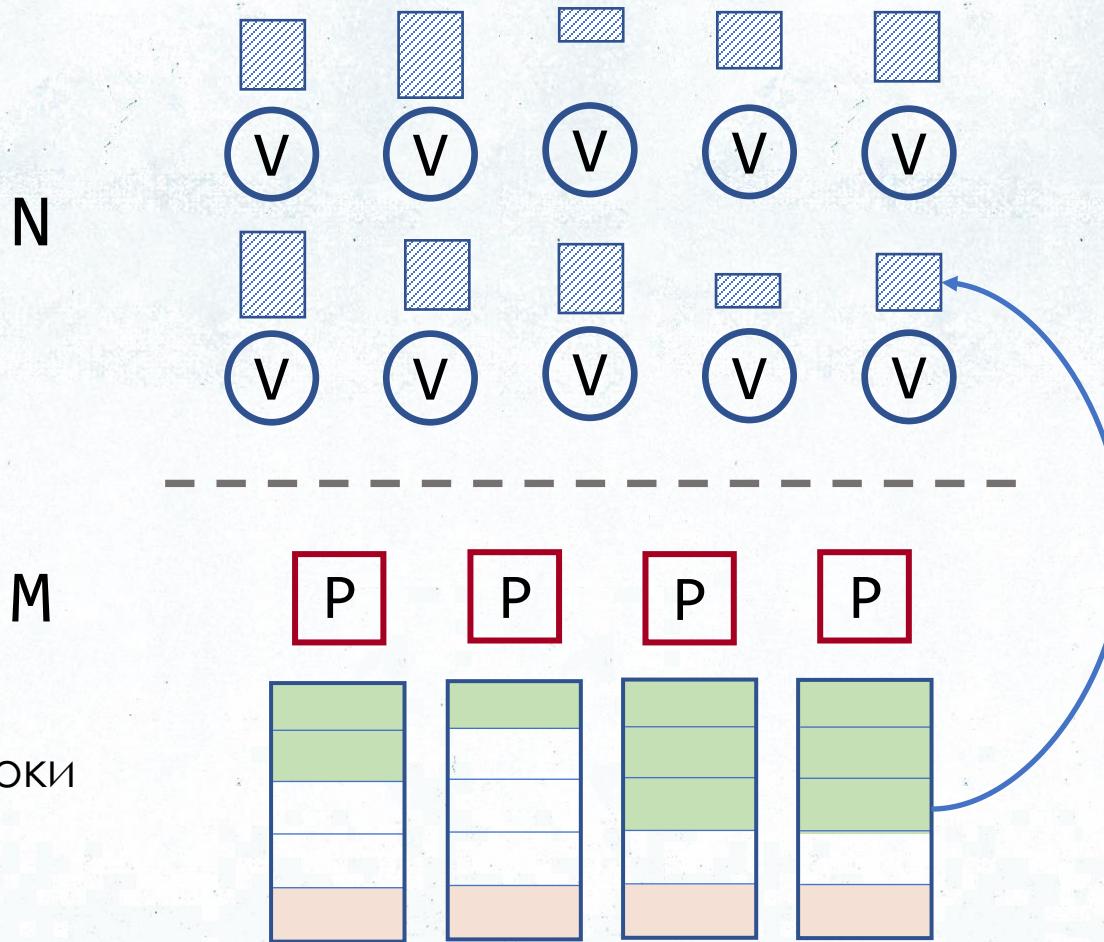
виртуальные потоки



- Классические стеки у потоков носителей
- Собственные маленькие стеки в хипе у виртуальных потоков
- Копирование стека при возобновлении
- Копирование обратно в хип при приостановке

BETTER THREADS

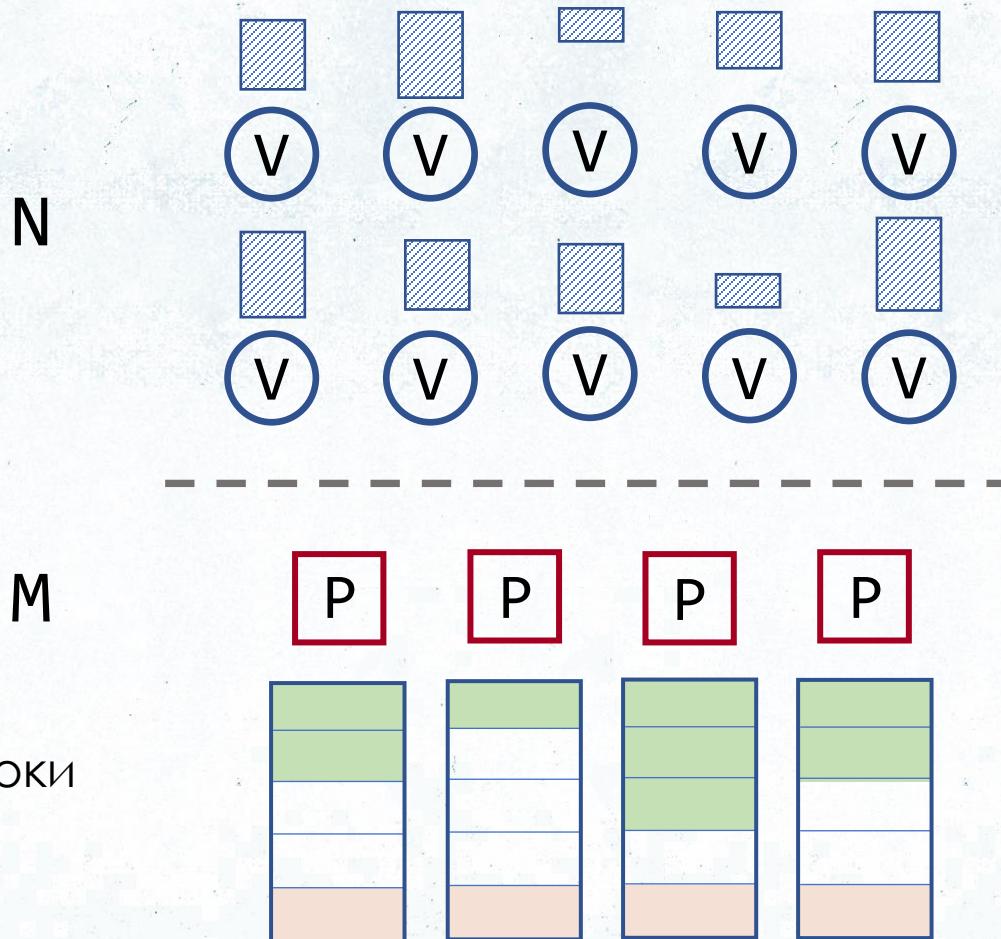
виртуальные потоки



- Классические стеки у потоков носителей
- Собственные маленькие стеки в хипе у виртуальных потоков
- Копирование стека при возобновлении
- Копирование обратно в хип при приостановке

BETTER THREADS

виртуальные потоки



- Классические стеки у потоков носителей
- Собственные маленькие стеки в хипе у виртуальных потоков
- Копирование стека при возобновлении
- Копирование обратно в хип при приостановке

BETTER THREADS

Теперь понятно, почему не работают нативы!
Там ведь могли быть указатели на стек!

```
struct test a;  
struct test* pa = &a;
```

BETTER THREADS

Теперь понятно, почему не работают нативы!
Там ведь могли быть указатели на стек!

```
struct test a;  
struct test* pa = &a;
```

Копирование стека сделает такие указатели невалидными.

BETTER THREADS

Получается, что context-switch – работает за
 $O(N)$ от размера стека?!

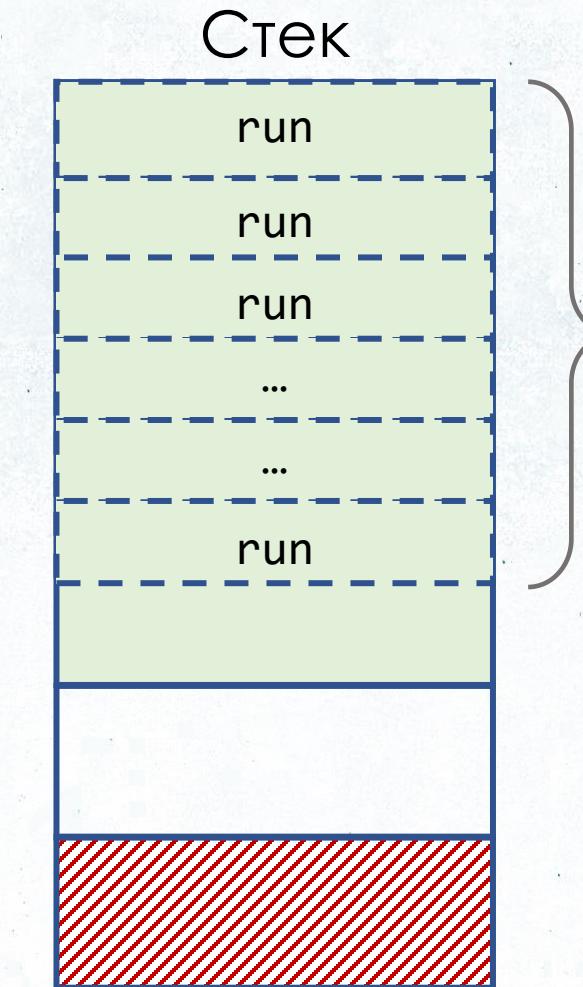
BETTER THREADS

```
run(depth)
↓
run(depth - 1)
↓
...
↓
run(0)
↓
for ( ; ; ) {
    Thread.yield()
}
```



BETTER THREADS

```
run(depth)
↓
run(depth - 1)
↓
...
↓
run(0)
↓
for ( ; ; ) {
    Thread.yield()
}
```



BETTER THREADS

```
import jdk.internal.vm.Continuation;
import jdk.internal.vm.ContinuationScope;

static class YieldAtLevel implements Runnable {

    private void run(int depth) {
        if (depth > 0) {
            run(depth - 1);
        } else if (depth == 0) {
            for (;;) {
                Continuation.yield(SCOPE);
            }
        }
    }
    ...
}
```

BETTER THREADS

```
import jdk.internal.vm.Continuation;
import jdk.internal.vm.ContinuationScope;

static class YieldAtLevel implements Runnable {

    private void run(int depth) {
        if (depth > 0) {
            run(depth - 1);
        } else if (depth == 0) {
            for (;;) {
                Continuation.yield(SCOPE);
            }
        }
    }
    ...
}
```

```
@Setup(Level.Trial)
public void setup() {
    cont = YieldAtLevel.cont(stackDepth);
    cont.run();
}

@Benchmark
public void yieldAtTheBottom() {
    cont.run();
}
```

BETTER THREADS

```
run(depth)
↓
run(depth - 1)
↓
...
↓
run(0)
↓
for ( ; ; ) {
    Thread.yield()
}
```



Benchmark	(stackDepth)	Score	Error	Units
yieldAtTheBottom	5	115.436	± 0.674	ns/op
yieldAtTheBottom	10	114.703	± 2.482	ns/op
yieldAtTheBottom	20	116.356	± 1.025	ns/op
yieldAtTheBottom	100	115.887	± 1.295	ns/op
yieldAtTheBottom	200	115.369	± 0.520	ns/op
yieldAtTheBottom	500	113.776	± 0.384	ns/op

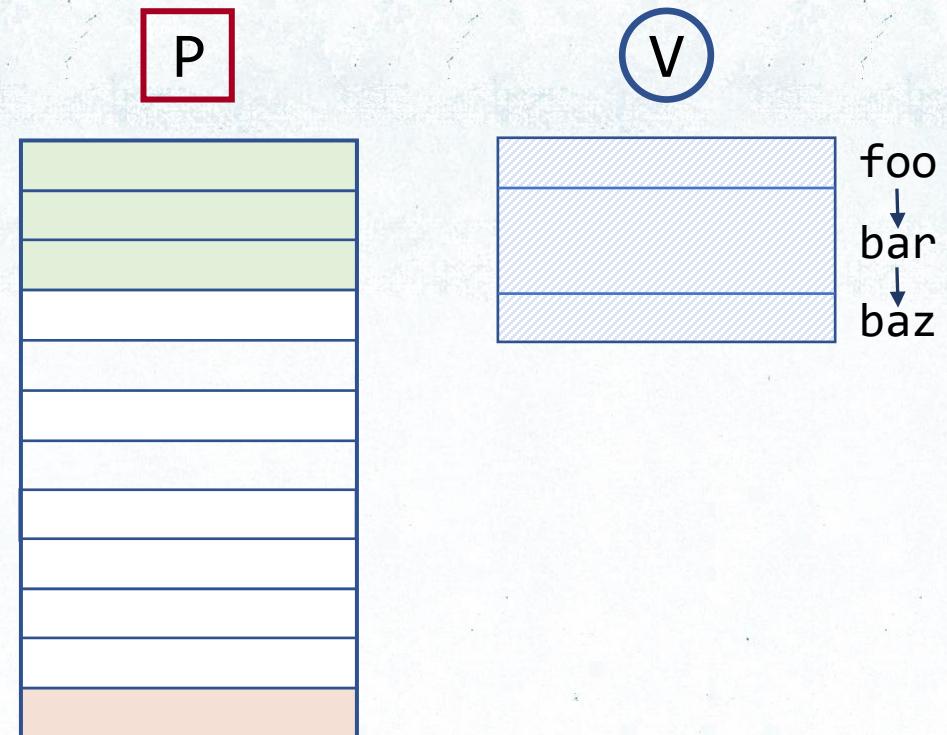


BETTER THREADS

Получается, что context-switch – работает за $O(N)$ от размера стека?!

Да, но это оптимизируется ленивым копированием:

- Копируются N фреймов
- Затем return барьер

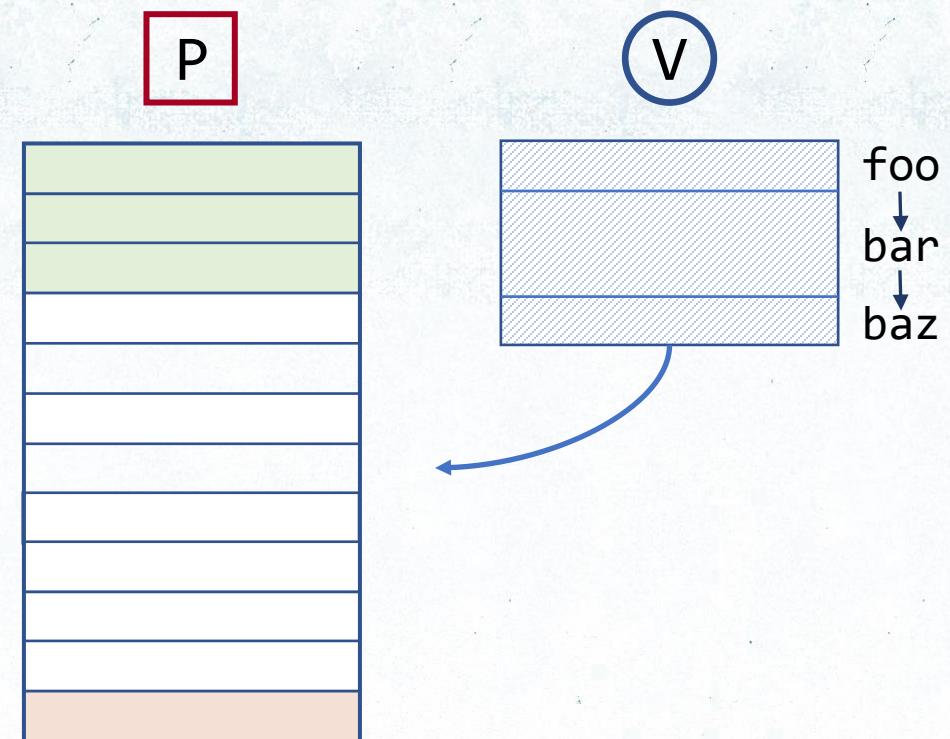


BETTER THREADS

Получается, что context-switch – работает за $O(N)$ от размера стека?!

Да, но это оптимизируется ленивым копированием:

- Копируются N фреймов
- Затем return барьер

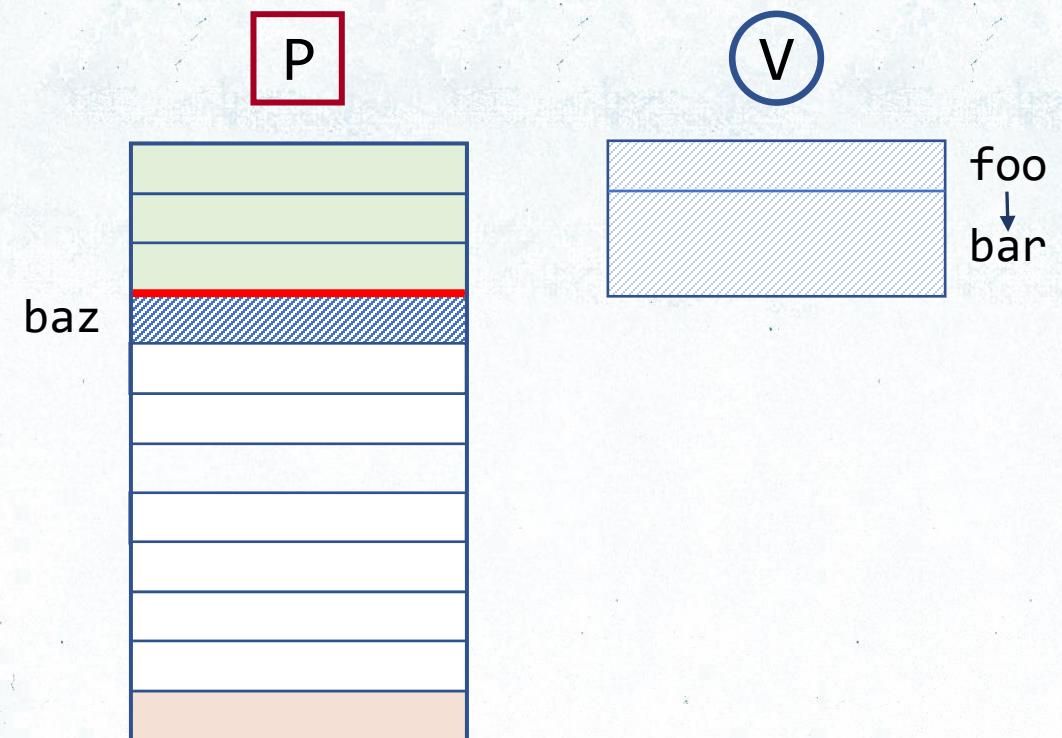


BETTER THREADS

Получается, что context-switch – работает за $O(N)$ от размера стека?!

Да, но это оптимизируется ленивым копированием:

- Копируются N фреймов
- Затем return барьер

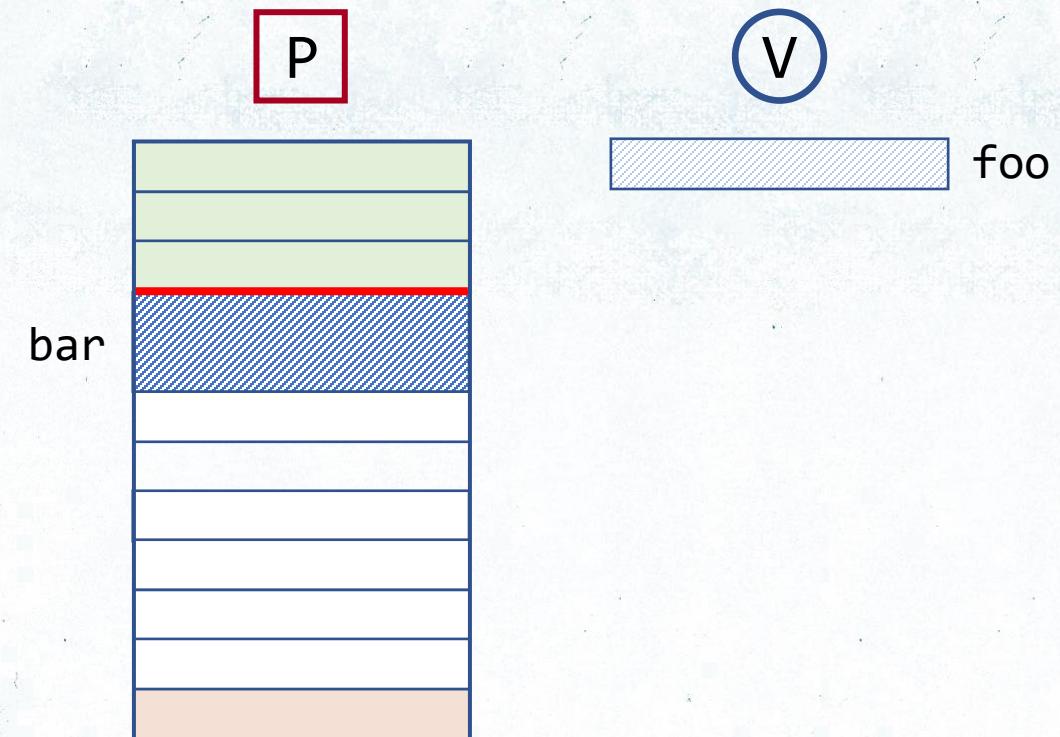


BETTER THREADS

Получается, что context-switch – работает за $O(N)$ от размера стека?!

Да, но это оптимизируется ленивым копированием:

- Копируется N фреймов
- Затем return барьер



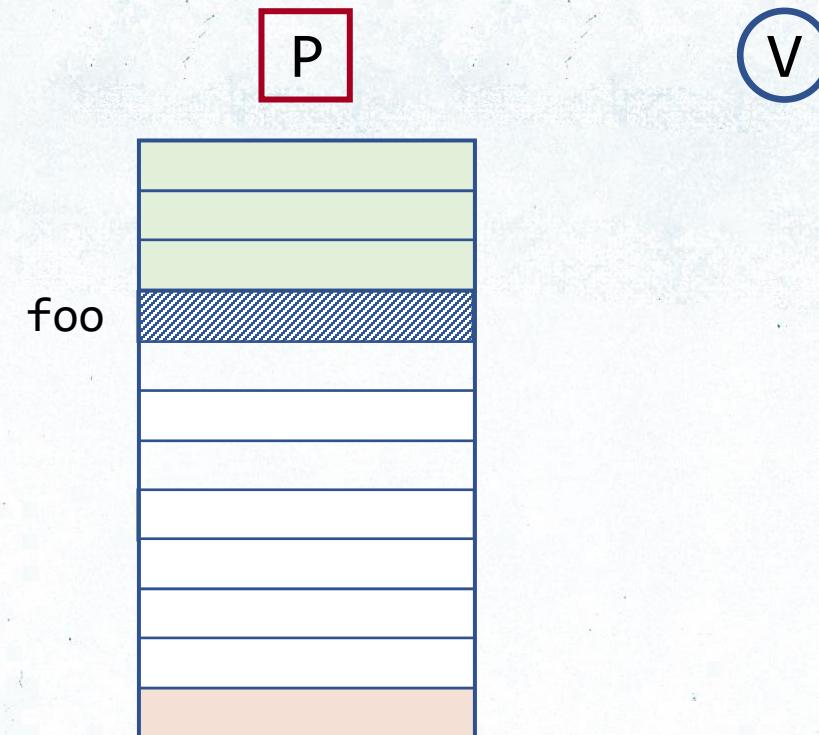
BETTER THREADS

Получается, что context-switch – работает за $O(N)$ от размера стека?!

Да, но это оптимизируется ленивым копированием:

- Копируется N фреймов
- Затем return барьер

Но есть и цена!



BETTER THREADS

```
run(depth)
↓
run(depth - 1)
↓
...
↓
run(0)
↓
for ( ; ; ) {
    Thread.yield()
}
```



BETTER THREADS

```
run(depth)
↓
run(depth - 1)
↓
...
↓
run(0)
↓
Thread.yield()
return;
```

BETTER THREADS

```
run(depth)
↓
run(depth - 1)
↓
...
↓
run(0)
↓
Thread.yield()
return;
```



BETTER THREADS

```
run(depth)
↓
run(depth - 1)
↓
...
↓
run(0)
↓
Thread.yield()
return;
```



A vertical sequence of method calls is shown on the left. It starts with `run(depth)`, followed by `run(depth - 1)`, then an ellipsis (`...`), then `run(0)`. Below `run(0)` is a red arrow pointing down to the code `Thread.yield()`, which is in blue. Below that is the word `return;` also in blue.

```
run(depth)
↓
run(depth - 1)
↓
...
↓
run(0)
return;
baseline
```



A vertical sequence of method calls is shown on the right. It starts with `run(depth)`, followed by `run(depth - 1)`, then an ellipsis (`...`), then `run(0)`. Below `run(0)` is the word `return;` in blue. Below that is the word `baseline` in blue. A large blue bracket is positioned on the right side of the sequence, spanning from the first `run(depth)` call down to the `return;` statement.



BETTER THREADS

Выводы по стекам:

- Context-switch не бесплатный, но оптимизации работают



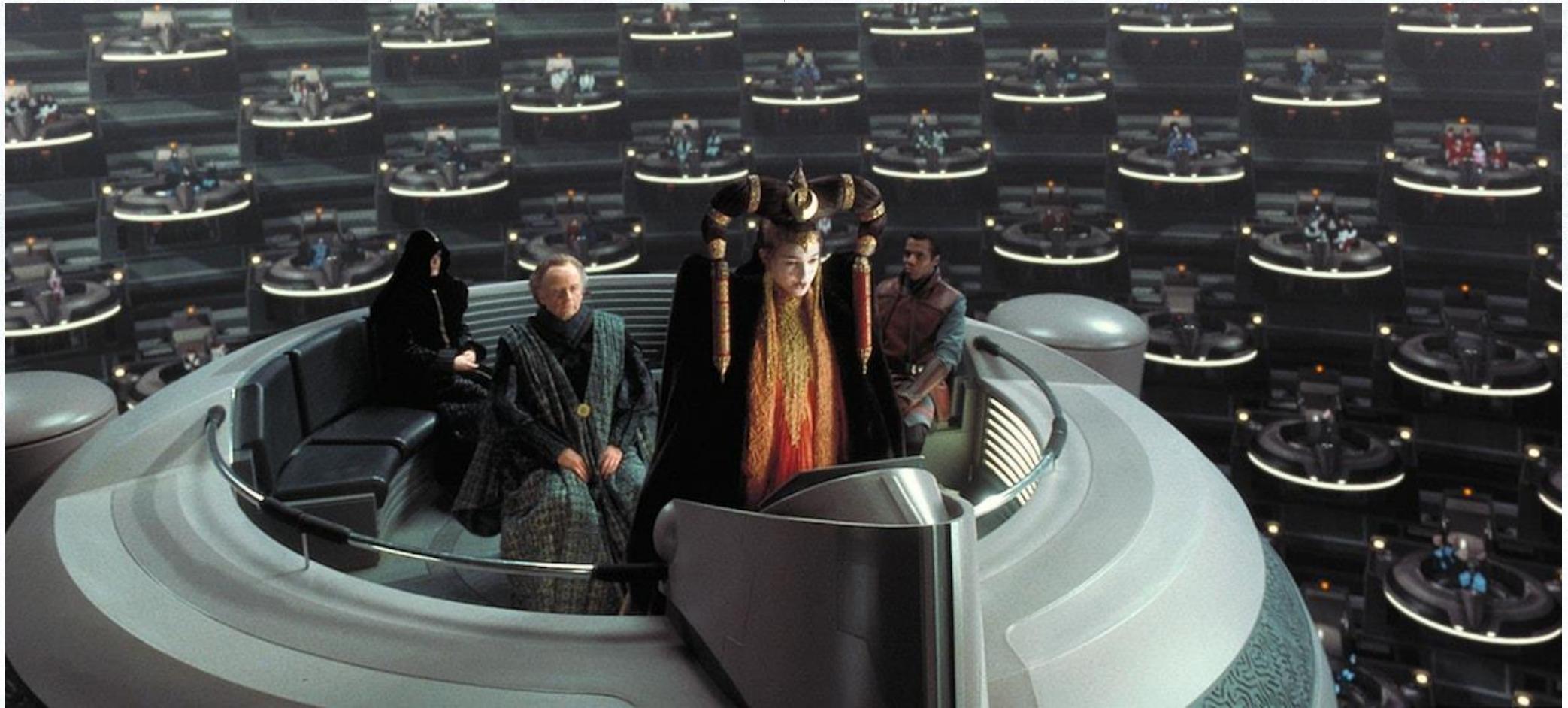
BETTER THREADS

Выводы по стекам:

- Context-switch не бесплатный, но оптимизации работают
- Контрпример: обратный ход глубокой рекурсии



MIGHTY SCHEDULER



THE PROBLEM

А как реализовать `java.lang.Thread`?

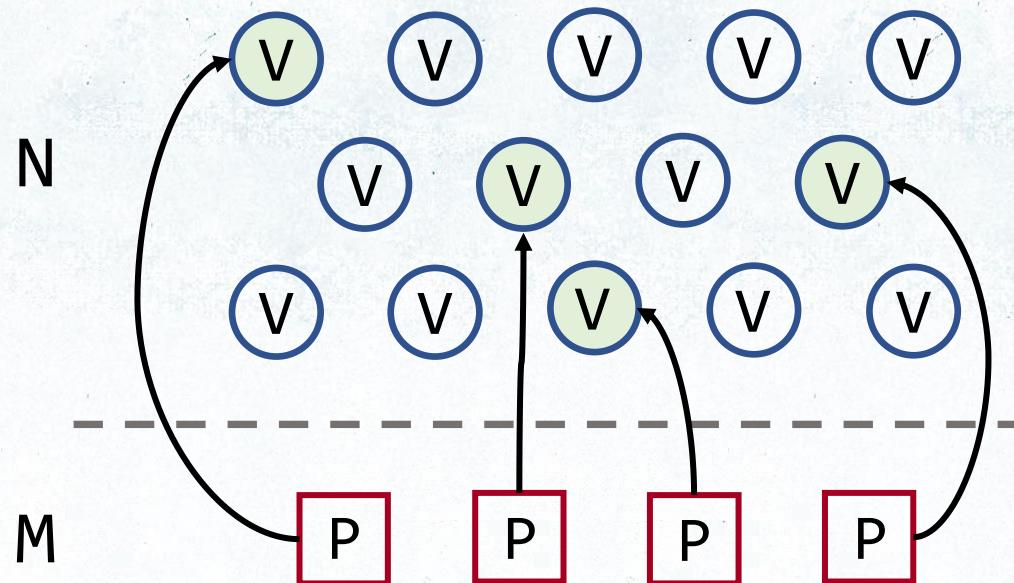
Чего вообще хотим от этого класса:

1. Оркестратор (**планировщик**), который будет решать, кому исполняться,
2. Система управления стеками,
3. Остаемся в рамках одного процесса



MIGHTY SCHEDULER

виртуальные потоки

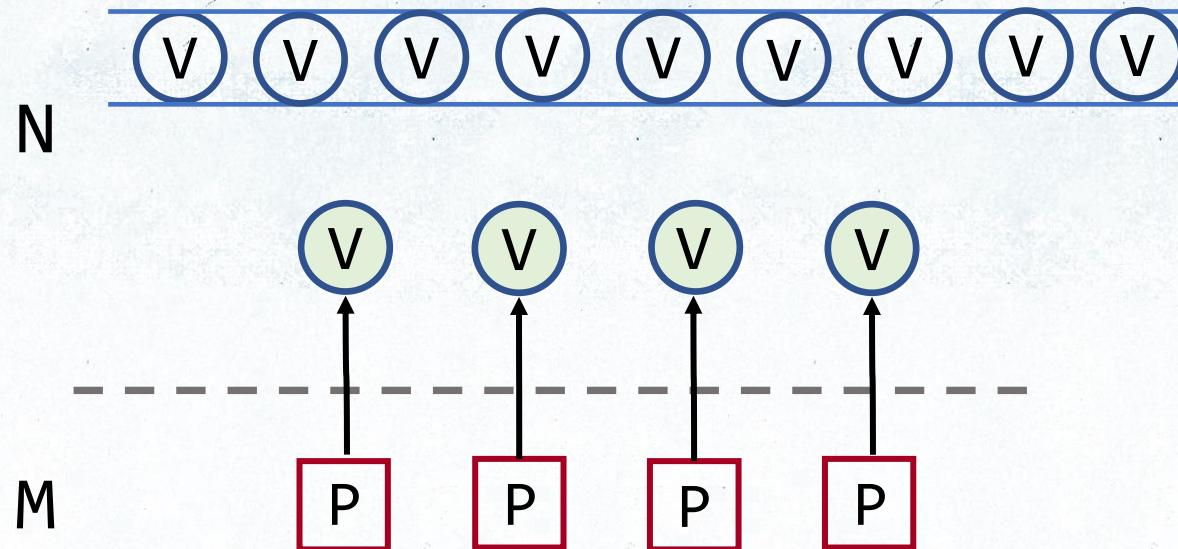


OS-потоки

Как понять, кто
следующий исполняется?

MIGHTY SCHEDULER

виртуальные потоки

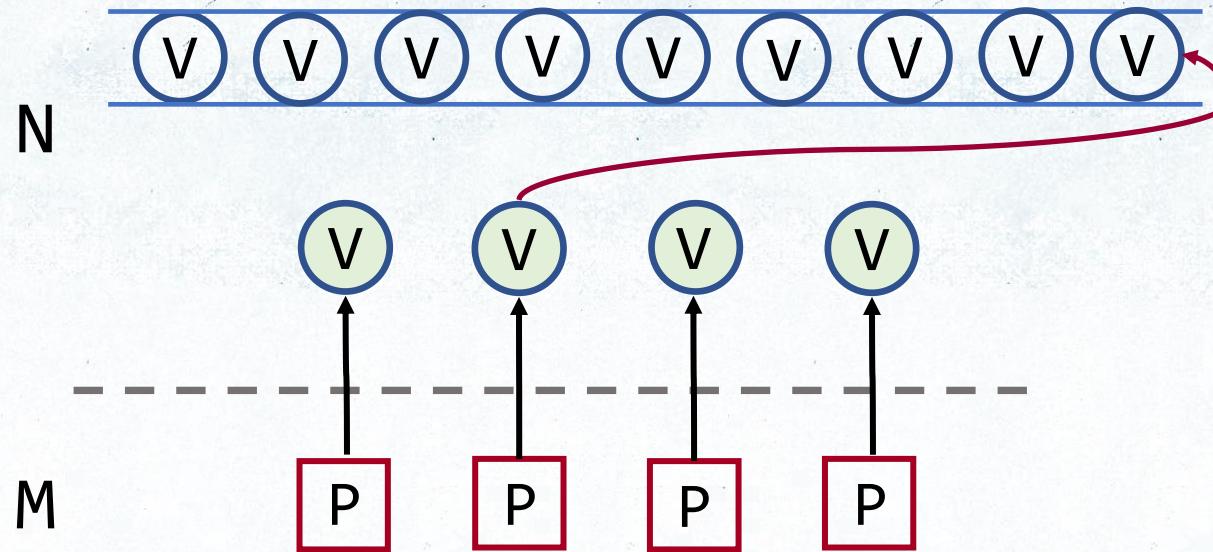


OS-потоки

Как понять, кто
следующий исполняется?
○ Очередь?

MIGHTY SCHEDULER

виртуальные потоки

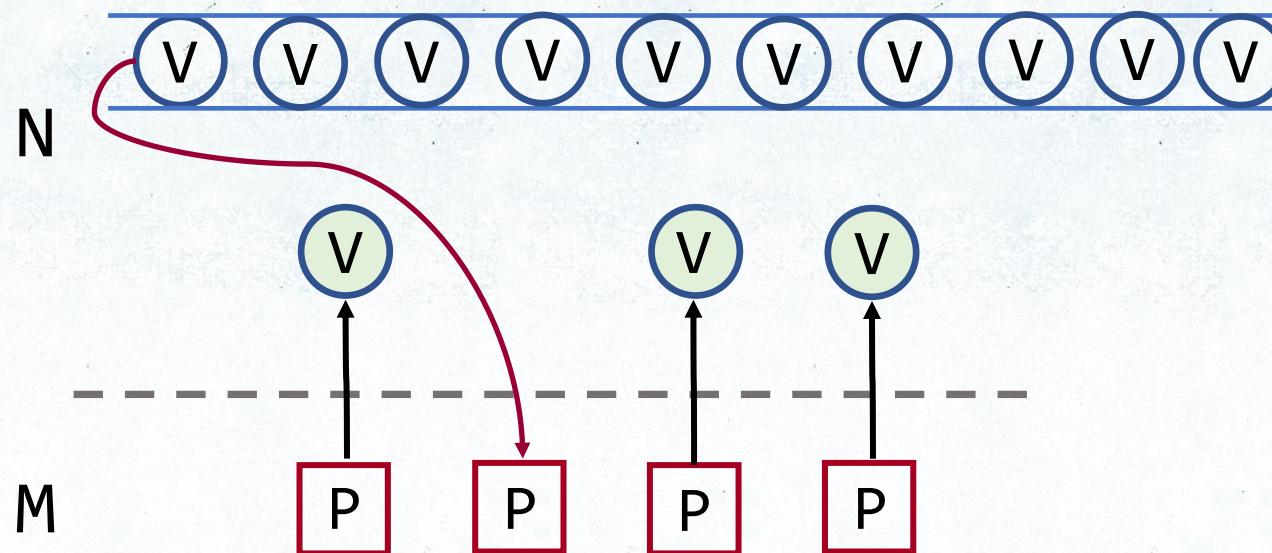


OS-потоки

Как понять, кто
следующий исполняется?
○ Очередь?

MIGHTY SCHEDULER

виртуальные потоки

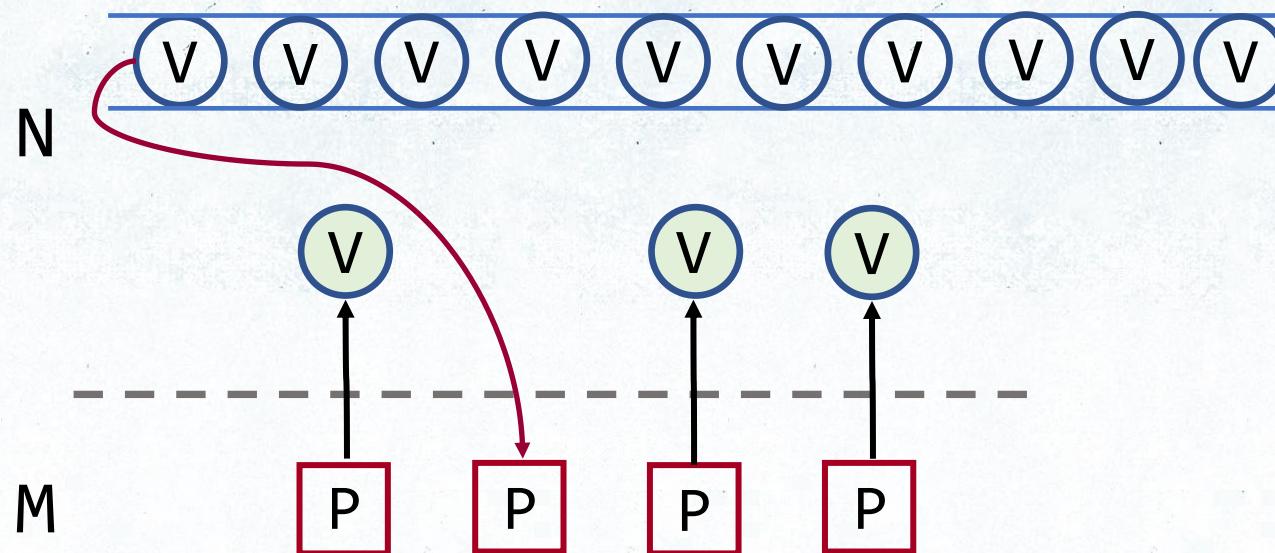


OS-потоки

Как понять, кто
следующий исполняется?
○ Очередь?

MIGHTY SCHEDULER

виртуальные потоки

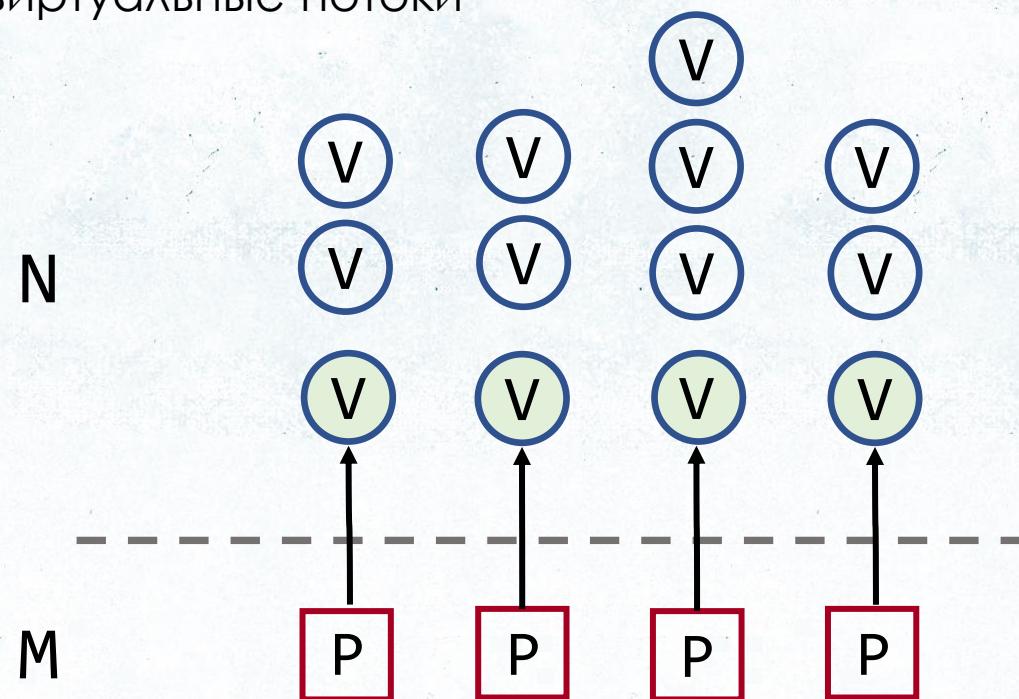


OS-потоки

Как понять, кто
следующий исполняется?
— Очередь?

MIGHTY SCHEDULER

виртуальные потоки



OS-потоки

Как понять, кто следующий исполняется?

- ~~Очередь?~~
- Много очередей!

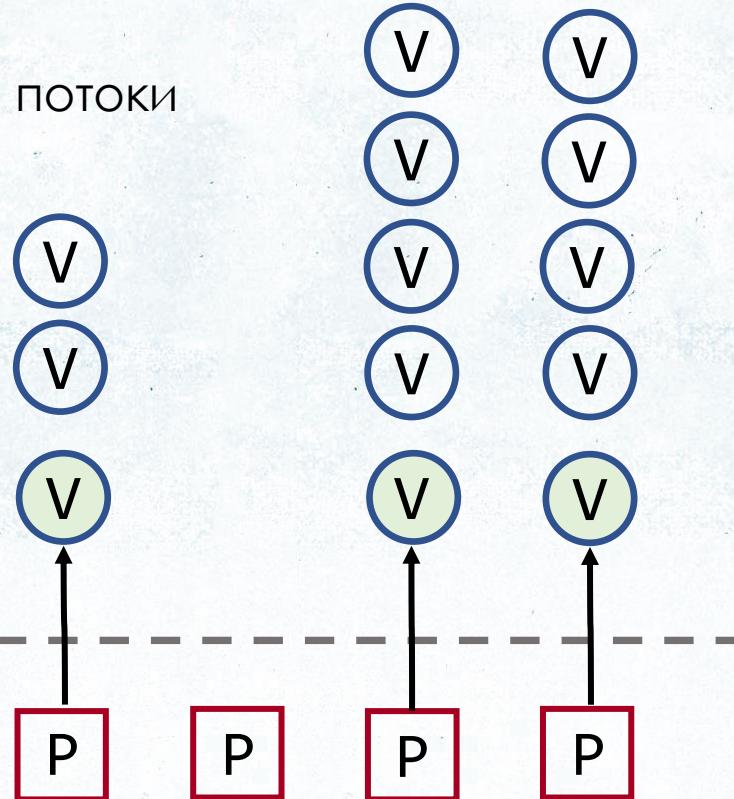
MIGHTY SCHEDULER

виртуальные потоки

N

M

OS-потоки



Как понять, кто
следующий исполняется?

- ~~Очередь?~~
- Много очередей!

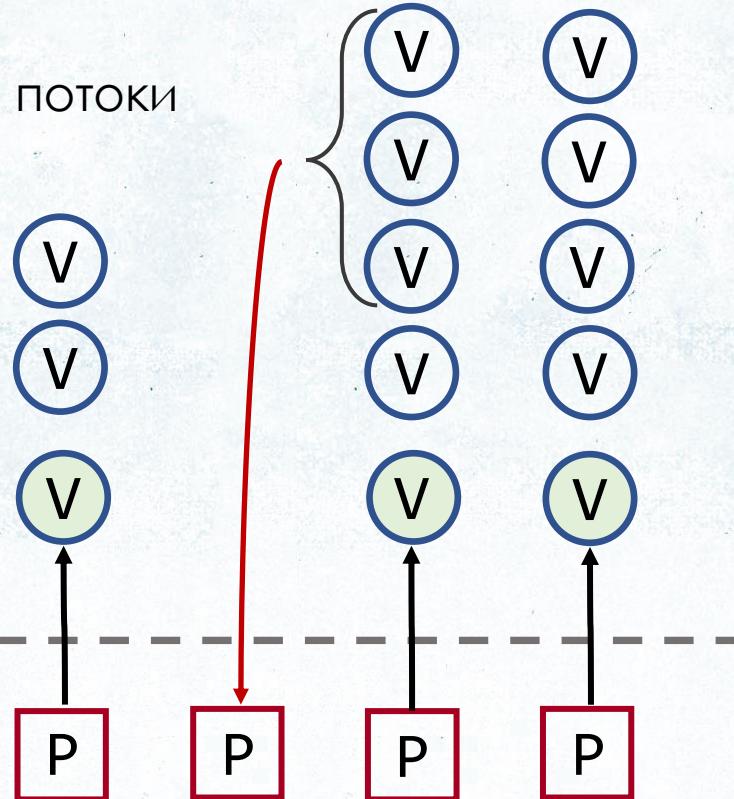
MIGHTY SCHEDULER

виртуальные потоки

N

M

OS-потоки



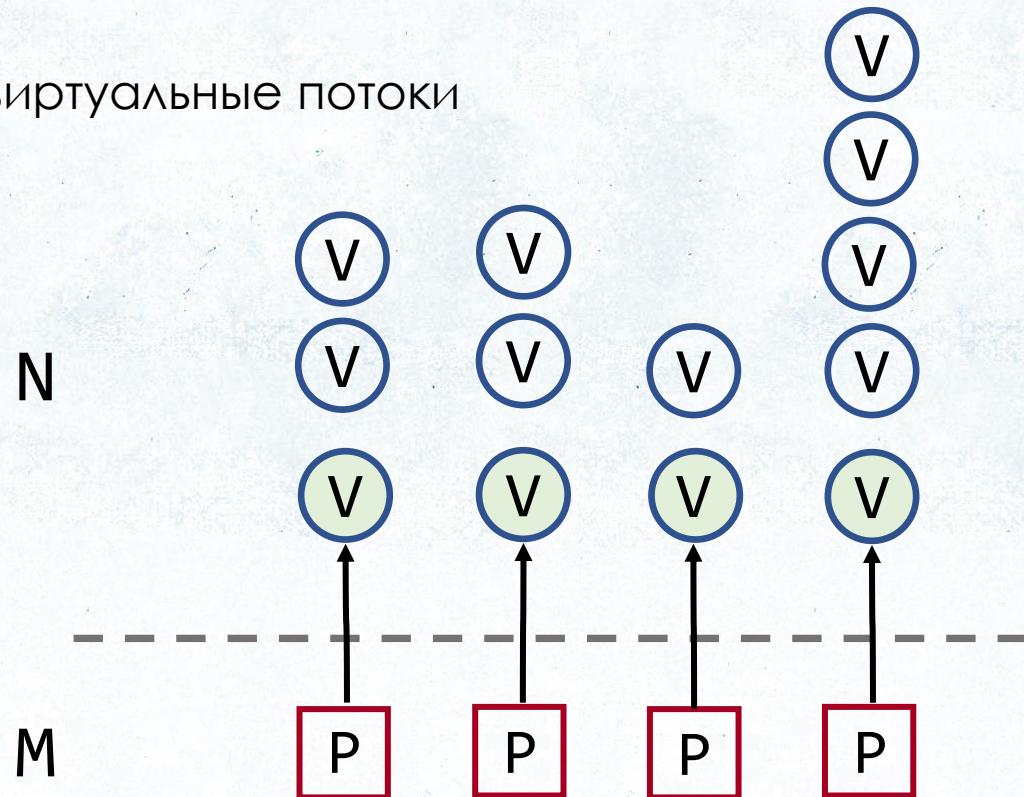
Как понять, кто
следующий исполняется?

○ ~~Очередь?~~

- Много очередей!
- Work stealing

MIGHTY SCHEDULER

виртуальные потоки



OS-потоки

Как понять, кто
следующий исполняется?

○ ~~Очередь?~~

- Много очередей!
- Work stealing

Такая функциональность
есть в ForkJoinPool!

MIGHTY SCHEDULER

В качестве планировщика используется ForkJoinPool

- Написан Дагом Ли!
- Мощно оптимизирован



Алексей
Шипилёв

ForkJoinPool
в Java 8

JUG.RU

youtube.com/watch?v=t0dGLFtRR9c

MIGHTY SCHEDULER

В качестве планировщика используется ForkJoinPool

- Написан Дагом Ли!
- Мощно оптимизирован

Но так ли он подходит именно для виртуальных потоков?

MIGHTY SCHEDULER

Раньше в Котлине тоже использовали FJP, как дефолтный планировщик.

MIGHTY SCHEDULER

Раньше в Котлине тоже использовали FJP, как дефолтный планировщик.

А теперь – нет. Почему?

MIGHTY SCHEDULER

PingPongActorBenchmark.kt

MIGHTY SCHEDULER

Benchmark	(dispatcher)	Score	Error	Units
PingPongActorBenchmark.coresCountPingPongs	scheduler	58,211	± 2,074	ms/op
PingPongActorBenchmark.coresCountPingPongs	fjp	58,506	± 10,942	ms/op
PingPongActorBenchmark.coresCountPingPongs	ftp_1	548,916	± 37,671	ms/op

PingPongActorBenchmark.singlePingPong	scheduler	23,855	± 1,436	ms/op
PingPongActorBenchmark.singlePingPong	fjp	81,871	± 1,816	ms/op
PingPongActorBenchmark.singlePingPong	ftp_1	29,627	± 0,290	ms/op

[PingPongActorBenchmark.kt](#)

MIGHTY SCHEDULER

Benchmark	(dispatcher)	Score	Error	Units
PingPongActorBenchmark.coresCountPingPongs	scheduler	58,211	± 2,074	ms/op
PingPongActorBenchmark.coresCountPingPongs	fjp	58,506	± 10,942	ms/op
PingPongActorBenchmark.coresCountPingPongs	ftp_1	548,916	± 37,671	ms/op

PingPongActorBenchmark.singlePingPong	scheduler	23,855	± 1,436	ms/op
PingPongActorBenchmark.singlePingPong	fjp	81,871	± 1,816	ms/op
PingPongActorBenchmark.singlePingPong	ftp_1	29,627	± 0,290	ms/op

[PingPongActorBenchmark.kt](#)

MIGHTY SCHEDULER

Эмпирические наблюдения:

- Зачастую приостановленная корутина скоро снова будет готова работать



MIGHTY SCHEDULER

Эмпирические наблюдения:

- Зачастую приостановленная корутина скоро снова будет готова работать
- Хорошо бы, чтобы она осталась на том же треде и на том же CPU (вспоминаем лекцию про кэши!)



MIGHTY SCHEDULER

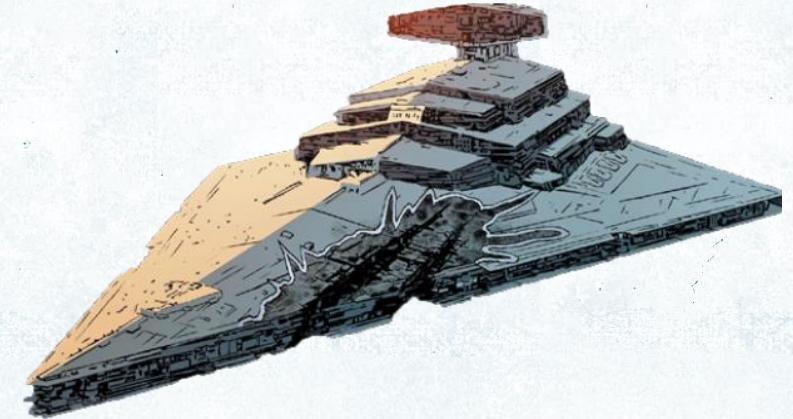
ForkJoinPool:

- Написан Дагом Ли!
- Мощно оптимизирован

MIGHTY SCHEDULER

ForkJoinPool:

- Написан Дагом Ли!
- Мощно оптимизирован
- Нужна более тонкая настройка очередности
- Слишком агрессивно ворует



MIGHTY SCHEDULER

ForkJoinPool:

- Написан Дагом Ли!
- Мощно оптимизирован
- Нужна более тонкая настройка очередности
- Слишком агрессивно ворует



У большинства JVM-based решений свой планировщик, который эти проблемы старается решать.

MIGHTY SCHEDULER

Нужна возможность писать и использовать
с виртуальными потоками свои планировщики!

MIGHTY SCHEDULER

Нужна возможность писать и использовать
с виртуальными потоками свои планировщики!



loom-dev@openjdk.java.net

...
Thus, full support for NUMA requires the ability **to create several different pools of carrier threads with the ability to configure them**. This is not possible in the current version of the API.

...

Vladimir Ogorodnikov

Yes, **the plan is to ultimately allow custom schedulers**, but we want to first let the ecosystem learn about virtual threads and their uses with the default scheduler.

...

Ron Pressler

IS LOOM A HERO?

IS LOOM A HERO?

Из хорошего:

- + Низкоуровневые и оптимизированные примитивы:
Continuation/Stack chunk
- + Переработана стандартная библиотека под
виртуальные потоки



IS LOOM A HERO?

Из хорошего:

- + Низкоуровневые и оптимизированные примитивы:
Continuation/Stack chunk
- + Переработана стандартная библиотека под
виртуальные потоки
- + Влияние на код минимальное!
Никакого coloring



IS LOOM A HERO?

Что нужно доделать:

- ? Synchronized
- ? Принудительное переключение потоков
- ? Humongous stack chunks (SOE при стеке >256КБ)
- ? Возможность делать свои планировщики



IS LOOM A HERO?

Что нужно доделать:

- ❓ ~~Synchronized~~ => fixed in Java 21!
- ❓ Принудительное переключение потоков
- ❓ Humongous stack chunks (SOE при стеке >256КБ)
- ❓ Возможность делать свои планировщики



IS LOOM A HERO?

Из плохого:

- Реализация сложна (кишки JVM + переделки в стандартной библиотеке)
- Сквозные нативы пыняют виртуальный тренд
- Обратный ход рекурсии может быть дорог
- Context-switch не бесплатен



РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture



Автоматическое
преобразование кода
в CPS a.k.a stackless
корутины

C#/JS/C++

Kotlin

больше

меньше

Влияние на код и язык

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture

больше



Автоматическое
преобразование кода
в CPS a.k.a stackless
корутины

C#/JS/C++

Kotlin

Влияние на код и язык



Виртуальные
потоки в Loom a.k.a
stackful корутины

меньше

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture

больше



Автоматическое
преобразование кода
в CPS a.k.a stackless
корутины

C#/JS/C++

Kotlin

Влияние на код и язык



Виртуальные
потоки в Loom a.k.a
stackful корутины

Loom
идеальный

меньше

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture

больше



Автоматическое
преобразование кода
в CPS a.k.a stackless
корутины

C#/JS/C++

Kotlin

Влияние на код и язык



Виртуальные
потоки в Loom a.k.a
stackful корутины

Loom
сейчас

Loom
идеальный

меньше

**ТОЛЬКО СИТХИ ВСЕ
ВОЗВОДЯТ В АБСОЛЮТ**



BRAVE NEW WORLD

```
func worker(id int) {
    fmt.Printf("Worker %d starting\n", id)
    time.Sleep(time.Second)
    fmt.Printf("Worker %d done\n", id)
}

func main() {
    for i := 1; i <= 10; i++ {
        go worker(i)
    }
}
```

BRAVE NEW WORLD

```
func main() {
    var wg sync.WaitGroup

    for i := 1; i <= 100000; i++ {
        wg.Add(1)
        i := i

        go func() {
            defer wg.Done()
            worker(i)
        }()
    }

    wg.Wait()
}
```

BRAVE NEW WORLD

```
func main() {
    var wg sync.WaitGroup

    for i := 1; i <= 100000; i++ {
        wg.Add(1)
        i := i

        go func() {
            defer wg.Done()
            worker(i)
        }()
    }

    wg.Wait()
}
```

...
Worker 99833 done
Worker 76674 done
Worker 99825 done
Worker 99821 done
Worker 99799 done
Worker 67867 done
Worker 99815 done
Worker 99804 done
Worker 99816 done
Worker 99863 done
Worker 87131 done
Worker 84239 done
Worker 99808 done

BRAVE NEW WORLD

Горутины в Go:

- о Любую функцию можно запустить параллельно



BRAVE NEW WORLD

Горутины в Go:

- о Любую функцию можно запустить параллельно
- о Специальных модификаторов/раскраски – нет
- о Ограничений на код – нет



BRAVE NEW WORLD

Горутины в Go:

- о Любую функцию можно запустить параллельно
- о Специальных модификаторов/раскраски – нет
- о Ограничений на код – нет
- о Миллион горутин создается и работает



BRAVE NEW WORLD



youtube.com/watch?v=-K11rY57K7k

BRAVE NEW WORLD

Реализация горутин в Go:

- N:M:P схема

BRAVE NEW WORLD

Реализация горутин в Go:

- N:M:P схема
- Бесконечные стеки
- Context-switch без копирования стеков

BRAVE NEW WORLD

Реализация горутин в Go:

- N:M:P схема
- Бесконечные стеки
- Context-switch без копирования стеков
 - + Бесплатный! За $O(1)$
 - + Нативы работают
 - Платим скоростью **каждой** функции

BRAVE NEW WORLD

Реализация горутин в Go:

- N:M:P схема
- Бесконечные стеки
- Context-switch без копирования стеков
- Preemption!
- Честность планирования горутин



РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



ThreadPool +
Callbacks/Future/
Combinators

ReactiveFrameworks
CompletableFuture

больше



Автоматическое
преобразование кода
в CPS a.k.a stackless
корутины

C#/JS/C++

Kotlin

Влияние на код и язык



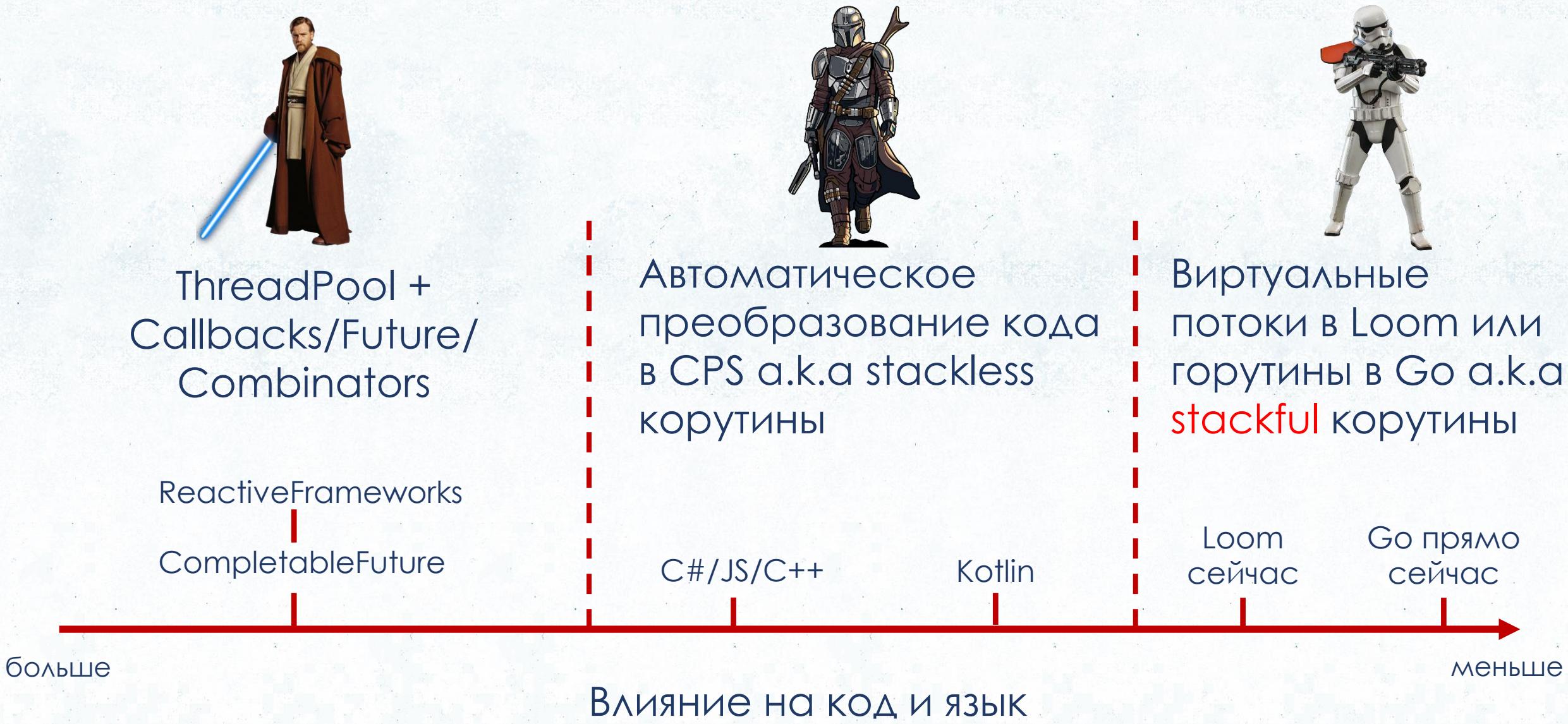
Виртуальные
потоки в Loom a.k.a
stackful корутины

Loom
сейчас

Loom
идеальный

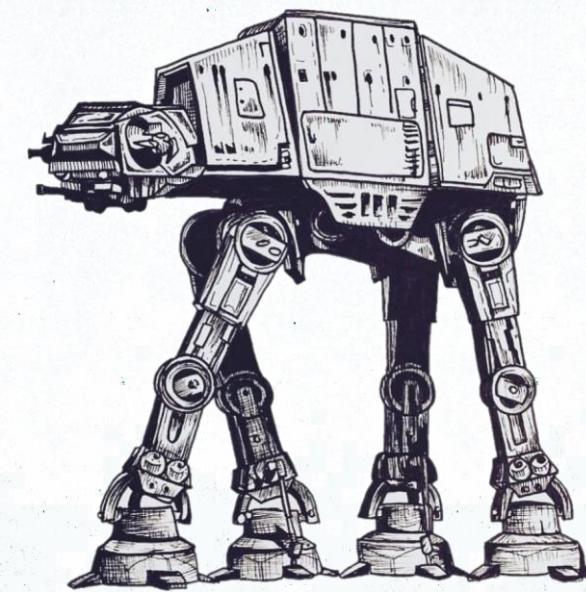
меньше

РЕШЕНИЯ ПРОБЛЕМЫ ТРЕДОВ



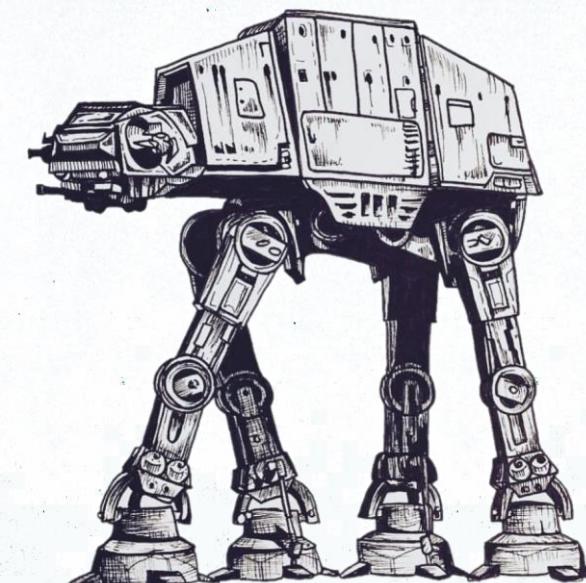
TAKEAWAYS

- Способы реализации j.l.Thread: 1:1 vs N:M схема, блокирующие и неблокирующие вызовы



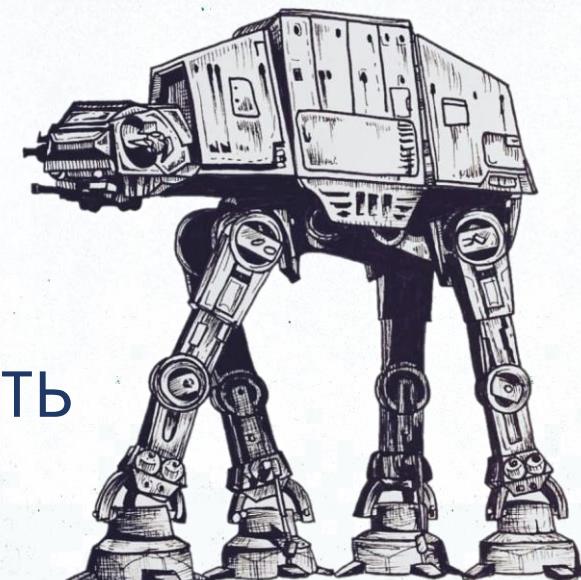
TAKEAWAYS

- Способы реализации `j.l.Thread`: 1:1 vs N:M схема, блокирующие и неблокирующие вызовы
- Реализация единиц многопоточности **влияет** на язык! В зависимости от способа реализации сильнее или слабее,



TAKEAWAYS

- Способы реализации `j.l.Thread`: 1:1 vs N:M схема, блокирующие и неблокирующие вызовы
- Реализация единиц многопоточности **влияет** на язык! В зависимости от способа реализации сильнее или слабее,
- Осторожнее с тредпулами, их легко сломать. В современном мире заводить по треду на запрос может быть лучше.



ВОПРОСЫ И ОТВЕТЫ!!



Бенчмарки и примеры здесь:

github.com/ugliansky/jpoint2022