

# Многопоточность

Вадим Калашников  
Wildberries, Lead Software Engineer



# Проверка связи





## Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включён звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти



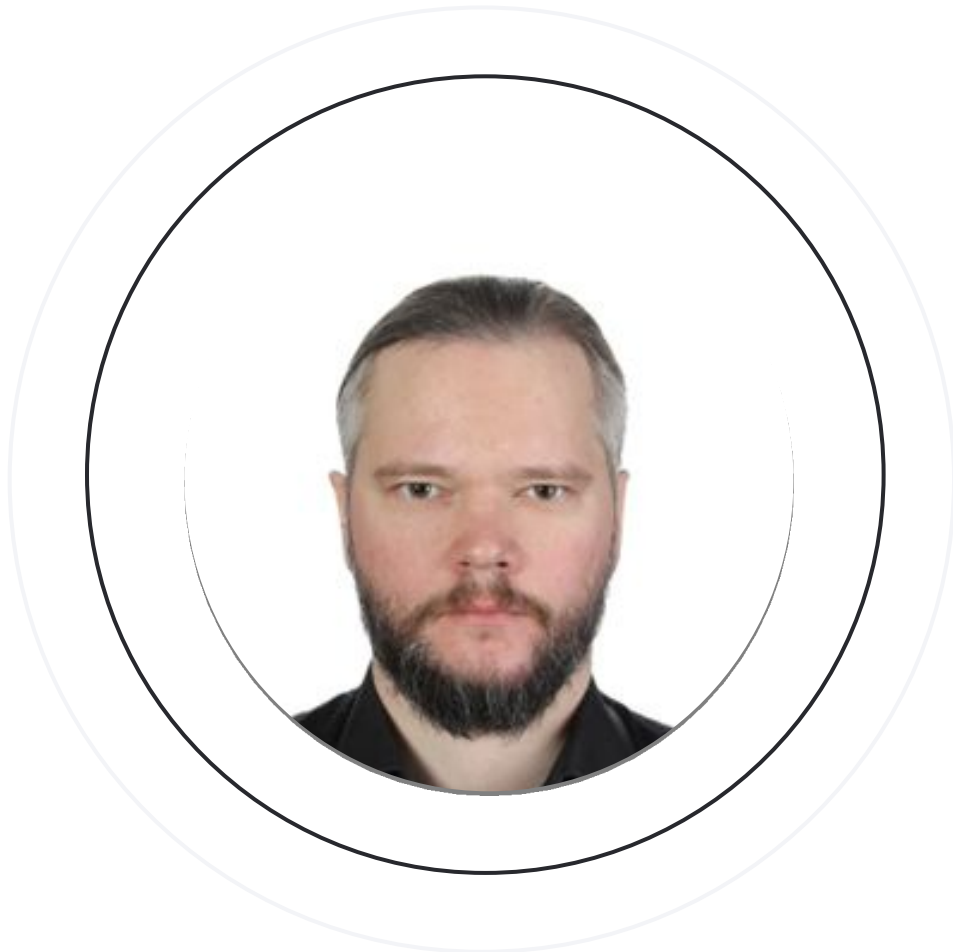
## Поставьте в чат:

-  если меня видно и слышно
-  если нет

# Вадим Калашников

О спикере:

- Разработчик на C++ более 15 лет
- Опыт разработки в областях: backend, embedded, kernel development, системное программирование, сети.
- Wildberries, Lead Software Engineer



# Вспоминаем прошрое занятие

**Вопрос:** Что такое полная специализация  
шаблона класса?



# Вспоминаем прошрое занятие

**Вопрос:** Что такое полная специализация шаблона класса?

**Ответ:** Перегрузка шаблона класса для конкретных типов данных



# Вспоминаем прошрое занятие

**Вопрос:** Какие шаблоны умных указателей вы знаете?



# Вспоминаем прошрое занятие

**Вопрос:** Какие шаблоны умных указателей вы знаете?

**Ответ:** `auto_ptr<>`, `unique_ptr<>`, `shared_ptr<>`, `weak_ptr<>`



# Вспоминаем прошрое занятие

**Вопрос:** Какие параметры чаще всего принимают функции-алгоритмы из библиотеки STL?





# Вспоминаем прошрое занятие

**Вопрос:** Какие параметры чаще всего принимают функции-алгоритмы из библиотеки STL?

**Ответ:** Итератор на начало контейнера, итератор на конец контейнера, функтор для обработки



# Вспоминаем прошрое занятие

**Вопрос:** Какое главное отличие l-value от r-value?



# Вспоминаем прошрое занятие

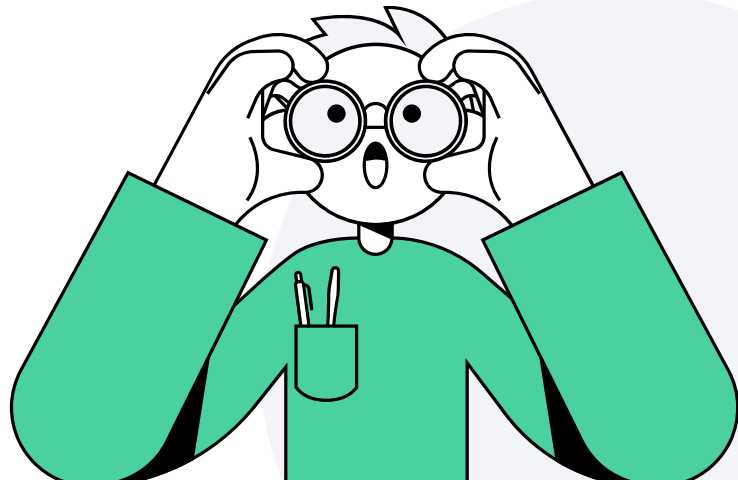
**Вопрос:** Какое главное отличие l-value от r-value?

**Ответ:** l-value это значение, на которое можно получить ссылку, r-value – на которое нельзя



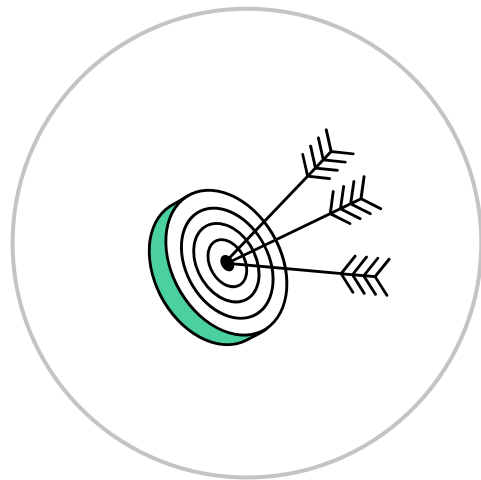
# Что вас ждет на курсе

- 1 Многопоточность
- 2 Конкуренция, состояние гонки. Race condition
- 3 Асинхронное программирование
- 4 Рефакторинг
- 5 Упаковка приложения в контейнер
- 6 Разбор задач и вопросов



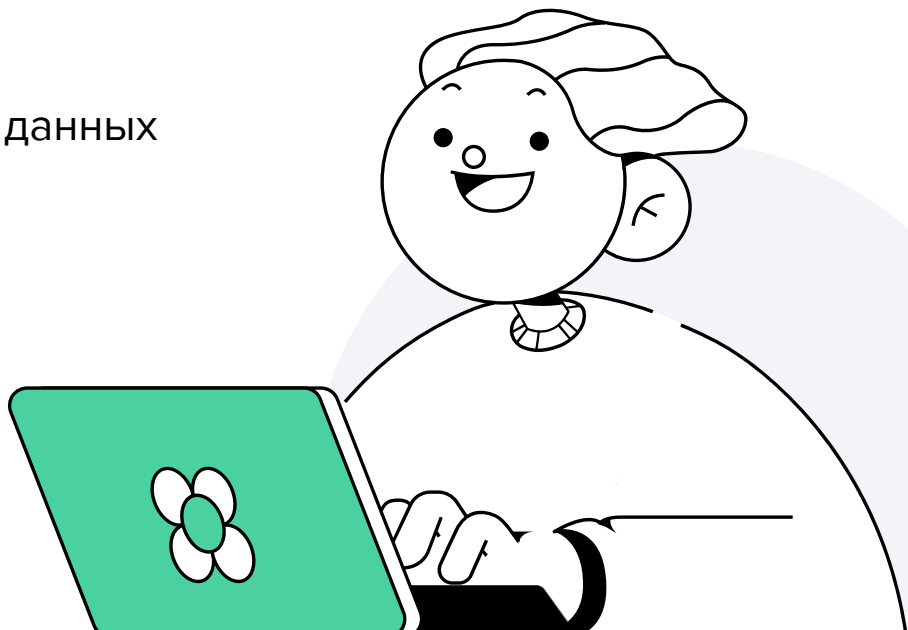
# Цели занятия

- Узнаем что такое поток и чем он отличается от процесса
- Сделаем первые программы с разделением на потоки
- Разберём принципы передачи данных в потоки и возврат значений из потоков



# План занятия

- 1 Параллельность
- 2 Измерение времени исполнения кода
- 3 Создание и запуск потоков
- 4 Передача данных в потоки и возврат данных
- 5 Итоги
- 6 Домашнее задание



\*Нажми на нужный раздел для перехода

# Параллельность



1

# Задачи параллельных вычислений

- 1 Обеспечить эффективное решение задач на той или иной параллельной системе, и какими критериями эффективности следует пользоваться
- 2 Описать класс тех задач, которые естественно решать на данной параллельной системе, а также класс задач, не поддающихся эффективному распараллеливанию
- 3 Обеспечить преобразование данного алгоритма в подходящую для рассматриваемой параллельной системы форму
- 4 Обеспечить поддержку переносимости полученной программы на систему с другой архитектурой
- 5 Сохранять работоспособность программы и улучшить ее характеристики при модификации данной системы; в частности, как обеспечить работоспособность программы при увеличении количества параллельных модулей



# Задачи параллельных вычислений

- 1 Обеспечить эффективное решение задач на той или иной параллельной системе, и какими критериями эффективности следует пользоваться
- 2 Описать класс тех задач, которые естественно решать на данной параллельной системе, а также класс задач, не поддающихся эффективному распараллеливанию
- 3 Обеспечить преобразование данного алгоритма в подходящую для рассматриваемой параллельной системы форму
- 4 Обеспечить поддержку переносимости полученной программы на систему с другой архитектурой
- 5 Сохранять работоспособность программы и улучшить ее характеристики при модификации данной системы; в частности, как обеспечить работоспособность программы при увеличении количества параллельных модулей

# Задачи параллельных вычислений

- 1 Обеспечить эффективное решение задач на той или иной параллельной системе, и какими критериями эффективности следует пользоваться
- 2 Описать класс тех задач, которые естественно решать на данной параллельной системе, а также класс задач, не поддающихся эффективному распараллеливанию
- 3 Обеспечить преобразование данного алгоритма в подходящую для рассматриваемой параллельной системы форму
- 4 Обеспечить поддержку переносимости полученной программы на систему с другой архитектурой
- 5 Сохранять работоспособность программы и улучшить ее характеристики при модификации данной системы; в частности, как обеспечить работоспособность программы при увеличении количества параллельных модулей

# Задачи параллельных вычислений

- 1 Обеспечить эффективное решение задач на той или иной параллельной системе, и какими критериями эффективности следует пользоваться
- 2 Описать класс тех задач, которые естественно решать на данной параллельной системе, а также класс задач, не поддающихся эффективному распараллеливанию
- 3 Обеспечить преобразование данного алгоритма в подходящую для рассматриваемой параллельной системы форму
- 4 Обеспечить поддержку переносимости полученной программы на систему с другой архитектурой
- 5 Сохранять работоспособность программы и улучшить ее характеристики при модификации данной системы; в частности, как обеспечить работоспособность программы при увеличении количества параллельных модулей

# Задачи параллельных вычислений

- 1 Обеспечить эффективное решение задач на той или иной параллельной системе, и какими критериями эффективности следует пользоваться
- 2 Описать класс тех задач, которые естественно решать на данной параллельной системе, а также класс задач, не поддающихся эффективному распараллеливанию
- 3 Обеспечить преобразование данного алгоритма в подходящую для рассматриваемой параллельной системы форму
- 4 Обеспечить поддержку переносимости полученной программы на систему с другой архитектурой
- 5 Сохранять работоспособность программы и улучшить ее характеристики при модификации данной системы; в частности, как обеспечить работоспособность программы при увеличении количества параллельных модулей

# Параллельная форма алгоритма

Для реализации алгоритма на параллельной системе его следует представить в виде последовательности групп операций.

Отдельные операции в каждой группе должны обладать следующим свойством: их можно выполнять одновременно на имеющихся в системе функциональных устройствах



**Представление алгоритма в таком виде называется параллельной формой алгоритма**



**Ярус** – каждая группа операций

Число таких ярусов – **высота параллельной формы**

**Ширина параллельной формы** – максимальное число операций в ярусах (число привлекаемых процессов в ярусах)

# Пример 1

Вычислить выражение  $(a_1 * a_2 + a_3 * a_4) * (a_5 * a_6 + a_7 * a_8)$

Данные	$a_1, a_2$	$a_3, a_4$	$a_5, a_6$	$a_7, a_8$
Ярус 1	$a_1 * a_2$	$a_3 * a_4$	$a_5 * a_6$	$a_7 * a_8$
Ярус 2	$a_1 * a_2 + a_3 * a_4$		$a_5 * a_6 + a_7 * a_8$	
Ярус 3	$(a_1 * a_2 + a_3 * a_4) * (a_5 * a_6 + a_7 * a_8)$			

Высота этой формы равна 3, а ширина 4

Данные	$a_1, a_2$	$a_3, a_4$	$a_5, a_6$	$a_7, a_8$
Ярус 1	$a_1 * a_2$	$a_3 * a_4$		
			$a_5 * a_6$	$a_7 * a_8$
Ярус 2	$a_1 * a_2 + a_3 * a_4$		$a_5 * a_6 + a_7 * a_8$	
Ярус 3	$(a_1 * a_2 + a_3 * a_4) * (a_5 * a_6 + a_7 * a_8)$			

Высота этой формы равна 4, а ширина 2

# Параллельная форма алгоритма

Для эффективного распараллеливания процесса стремятся

- ① к увеличению загрузки системы процессоров
- ② к отысканию параллельной формы с заданными свойствами



# Пример 2

Требуется перемножить числа  $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$

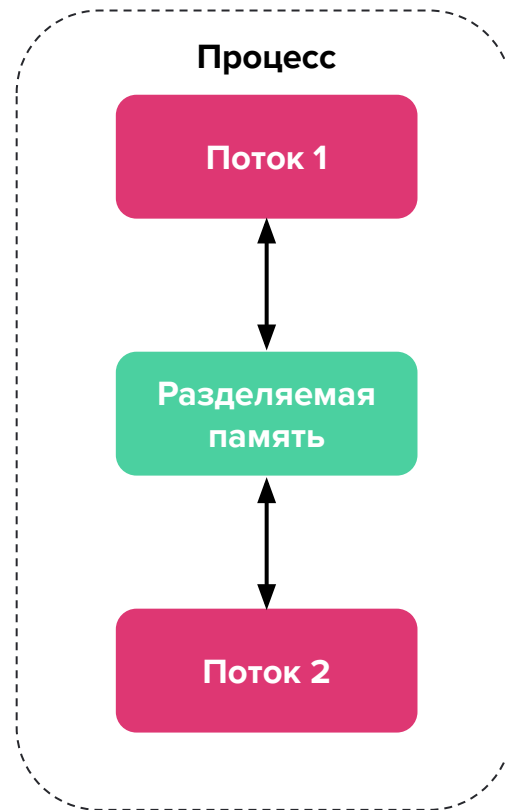
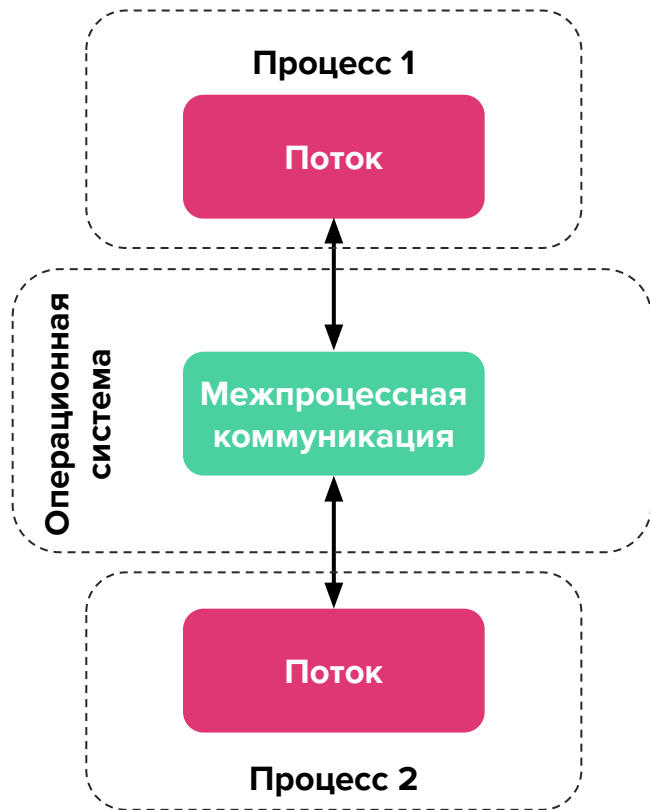
Данные	$a_1, a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$	$a_8$
Ярус 1	$a_1 * a_2$						
Ярус 2	$(a_1 * a_2) * a_3$						
Ярус 3	$(a_1 * a_2 * a_3) * a_4$						
Ярус 4	$(a_1 * a_2 * a_3 * a_4) * a_5$						
Ярус 5	$(a_1 * a_2 * a_3 * a_4 * a_5) * a_6$						
Ярус 6	$(a_1 * a_2 * a_3 * a_4 * a_5 * a_6) * a_7$						
Ярус 7	$(a_1 * a_2 * a_3 * a_4 * a_5 * a_6 * a_7) * a_8$						

Высота этой формы равна 7, а ширина 1

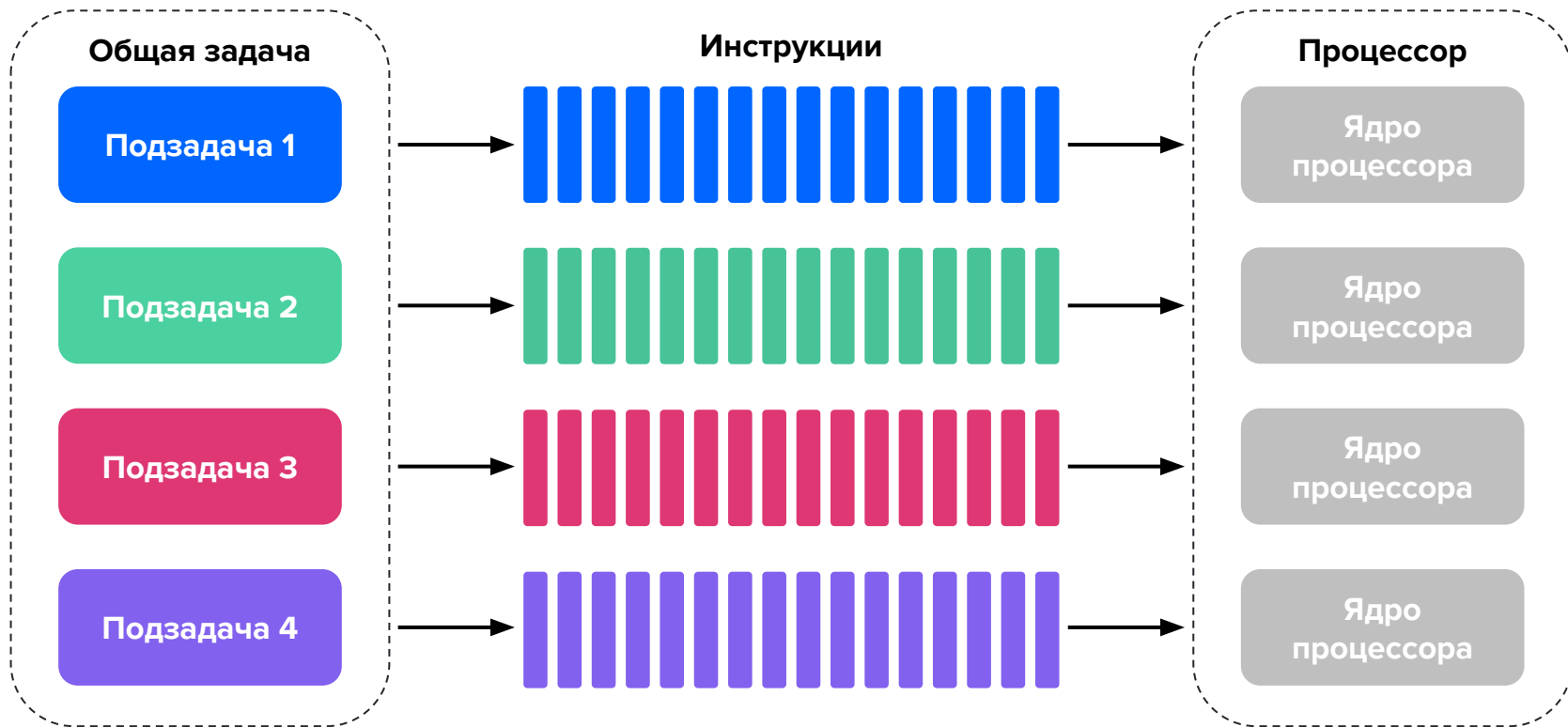
Схема сдваивания				
Данные	а1, а2	а3, а4	а5, а6	а7, а8
Ярус 1	а1 * а2	а3 * а4	а5 * а6	а7 * а8
Ярус 2	(а1 * а2) * (а3 * а4)		(а5 * а6) * (а7 * а8)	
Ярус 3	(а1 * а2 * а3 * а4) * (а5 * а6 * а7 * а8)			

Высота этой формы равна 3, а ширина 4

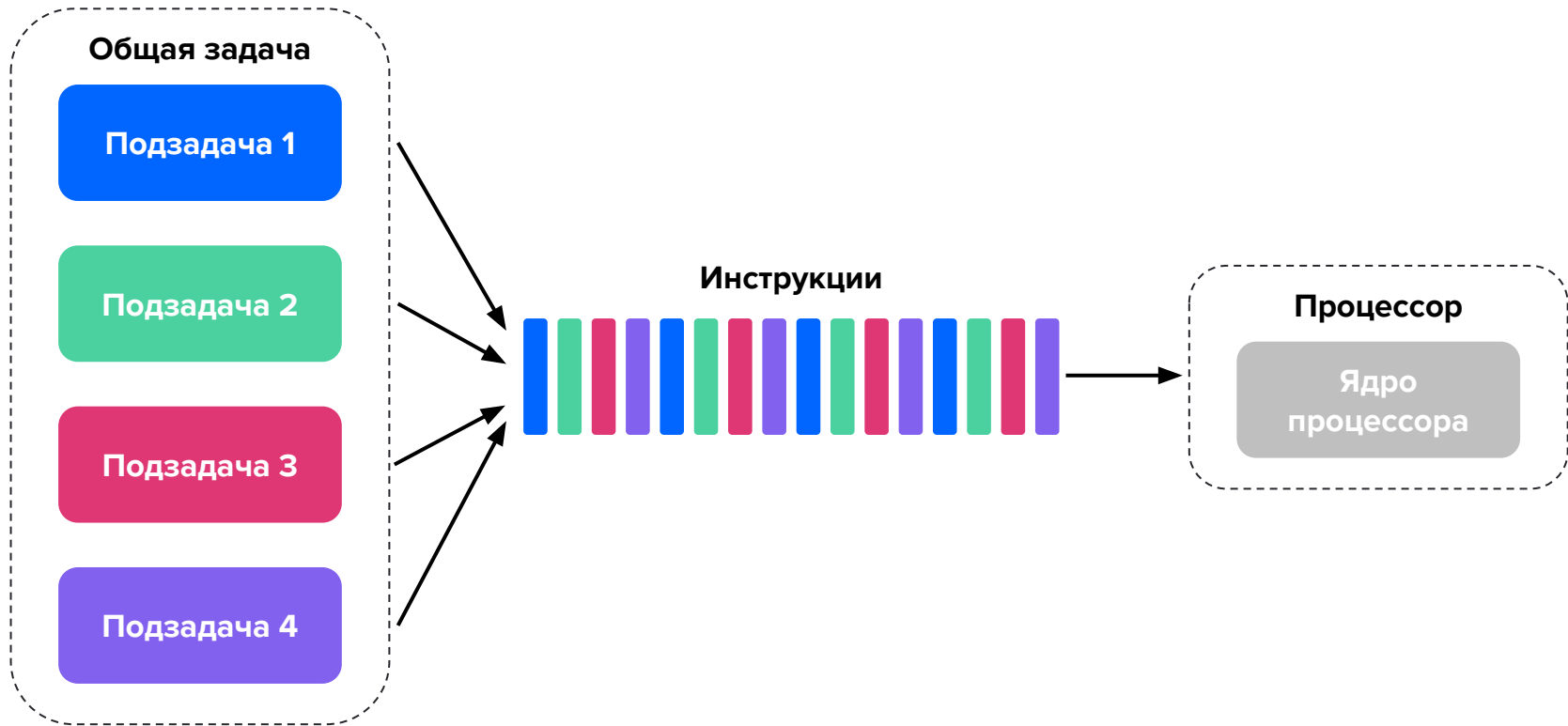
# Два вида многозадачности



# Аппаратный параллелизм



# Программный параллелизм



# Измерение времени исполнения кода



2

# Библиотека std::chrono

Библиотека std::chrono реализует следующие концепции:

- интервалы времени – duration
- моменты времени – time\_point
- таймеры – clock

```
1  auto start = std::chrono::steady_clock::now();  
2  do_something();  
3  auto end = std::chrono::steady_clock::now();  
4  std::chrono::duration<double> elapsed_seconds = end - start;  
5  std::cout << "elapsed time: " << elapsed_seconds.count() << "s\n";
```

# Автоматическое распараллеливание

В стандарте C++ 17 появилось одно крупное расширение политики выполнения для стандартных алгоритмов. Шестьдесят девять алгоритмов были дополнены возможностью параллельного исполнения на нескольких ядрах.

# Автоматическое распараллеливание

Для пользователя это означает возможность простого распараллеливания без использования `std::thread`

```
1  #include <algorithm>
2  #include <execution>
3
4  int main()
5  {
6      sort(v.begin(), v.end()); // без распараллеливания
7      sort(execution::par, v.begin(), v.end()); // с распараллеливанием
8
9      return 0;
10 }
```



# Создание и запуск потоков



3

# Полезные функции

Полезные функции, в пространстве имен `std::this_thread` и в классе `thread`

<code>std::this_thread::get_id</code>	возвращает id текущего потока
<code>std::this_thread::sleep_for</code>	блокирует выполнение текущего потока в течение установленного периода
<code>std::this_thread::sleep_until</code>	блокирует выполнение текущего потока, пока не будет достигнут указанный момент времени
<code>std::thread::hardware_concurrency</code>	возвращает число потоков, которые могут работать по-настоящему параллельно

\*Нажми на функции для перехода по ссылкам

# Запуск потока

Вне зависимости от того, что поток будет делать и откуда он запускается, сам запуск потока в стандартном C++ всегда сводится к конструированию объекта `std::thread`

```
1  #include<thread>
2  void foo()
3  {
4      // do something
5  }
6
7  int main()
8  {
9      std::thread t1(foo);
10     t1.join();
11     return 0;
12 }
```

# Ожидание завершения потока

Для описания отношений запущенных потоков с основным потоком используются функции `join()` и `detach()`

Функция `join()` завершает основной поток только после того, как завершатся все связанные с ним потоки

Функция `detach()` отсоединяет основной поток от всех остальных и завершает его не дожидаясь завершения других потоков.

# Передача данных в потоки и возврат данных



4

# Передача аргументов функции потока

Передача аргументов вызываемому объекту или функции сводится просто к передаче дополнительных аргументов конструктору `std::thread`

Однако важно иметь в виду, что по умолчанию эти аргументы копируются в память объекта, где они доступны вновь созданному потоку, причем так происходит даже в том случае, когда функция ожидает на месте соответствующего параметра ссылку

# Передача аргументов функции потока

Для гарантированной передачи аргумента по ссылке необходимо обернуть передаваемый параметр в функцию [std::ref](#)

Поток не имеет прямой возможности возвращать результат из запускаемой функции

# Передача владения потоком

Тип `std::thread` как и, например, `std::unique_ptr` является перемещаемым, но не копируемым. Это означает, что владение потоком можно передавать от одного экземпляра `std::thread` другому, только через операцию `std::move`

```
1  #include<thread>
2  void foo() { // do something }
3
4  int main()
5  {
6      std::thread t1(foo);
7      thread t2;
8      t2 = move(t1);
9      t1 = thread(foo);
10     t1.join();
11     t2.join();
12     return 0;
13 }
```



# Найдите ошибку

```
1 void func1()  
2 {  
3     cout << "New thread" << endl;  
4 }  
5  
6 int main()  
7 {  
8     thread t1(func1);  
9     t1.join();  
10    t1.detach();  
11    return 0;  
12 }
```



# Найдите ошибку

```
1  void func1(int &x)
2  {
3      x = x * 10;
4  }
5
6  int main()
7  {
8      int k = 10;
9      thread t1(func1, k);
10     t1.join();
11     cout << k << endl;
12     return 0;
13 }
```



# Итоги



# Итоги занятия

- 1 Узнали что такое поток и чем он отличается от процесса
- 2 Сделали первые программы с разделением на потоки
- 3 Разобрали принципы передачи данных в потоки и возврат значений из потоков



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



# Дополнительные материалы

- «C++ Concurrency in Action» Anthony Williams
- «Эффективный и современный C++» Скотт Мейерс
- [std::thread](#)



# Задавайте вопросы и пишите отзыв о лекции

Вадим Калашников  
Старший бэкенд разработчик на C++ в международной  
компании Orion Innovation

