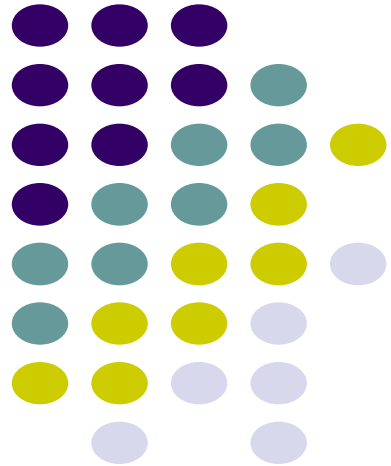
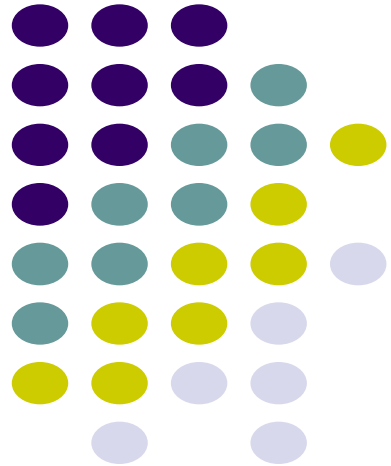
Course
Parallel Programming and Distributed Systems in Java

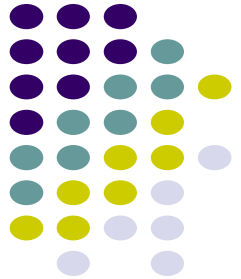


Лекция 1

Introduction to Parallel Computing



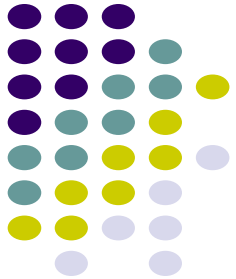
Что позволяет делать более быстрый компьютер?



- Существующие задачи решаются быстрее
  - Уменьшается время вычисления сложных задач
  - Увеличивается отзывчивость интерактивных приложений

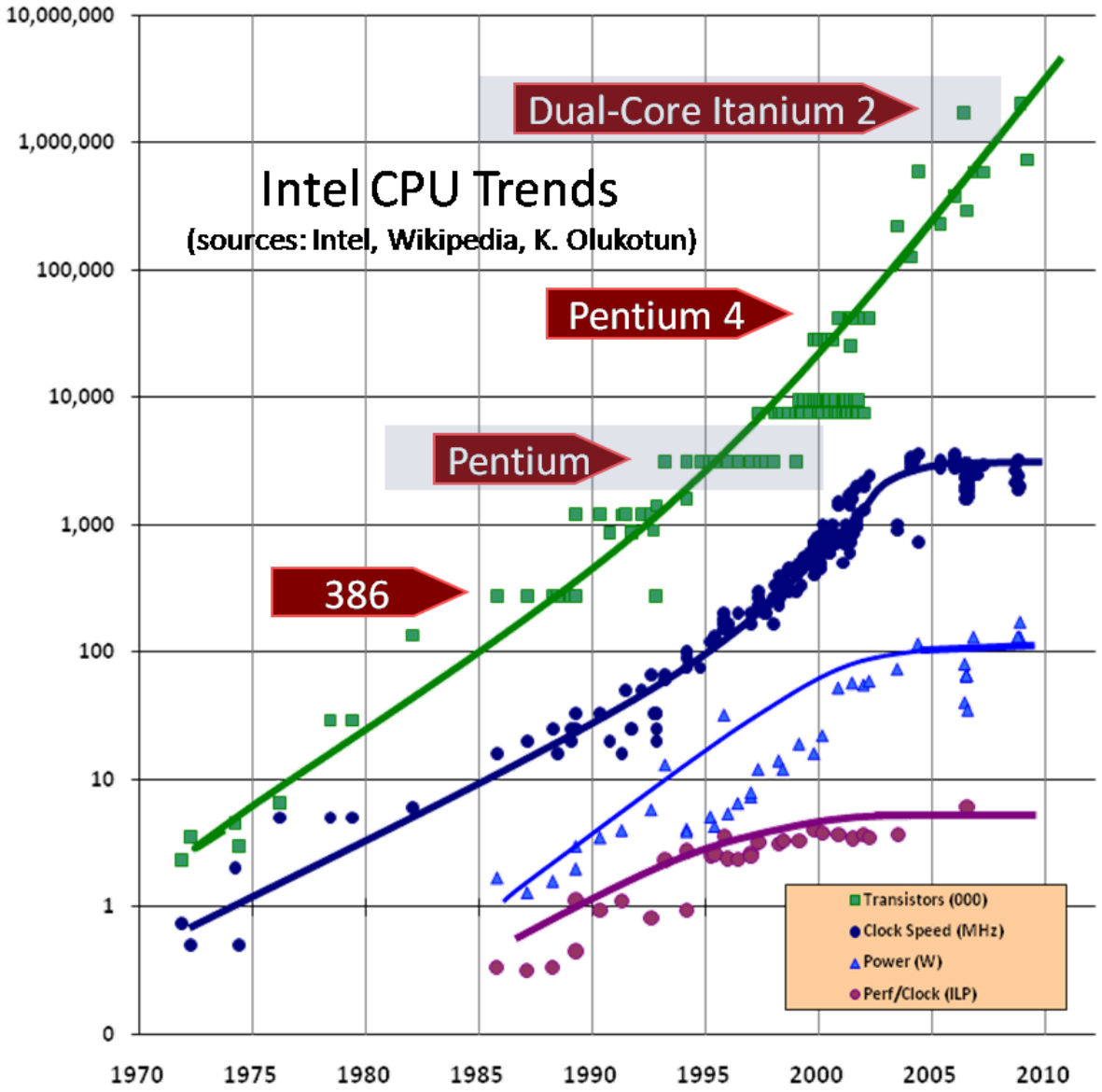
- Улучшенные решения за то же время
  - Увеличивается детализация моделей
  - Позволяет строить более сложные модели

Что такое параллельные вычисления?



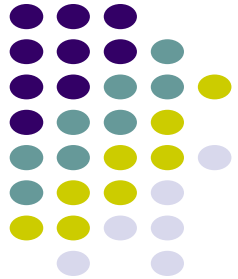
- Ускорение выполнения конкретной задачи путем:
  1. Разделения задачи на подзадачи
  2. Одновременного выполнения подзадач

Бесплатный обед?



- “Free Lunch Is Over”
  – *Herb Sutter*

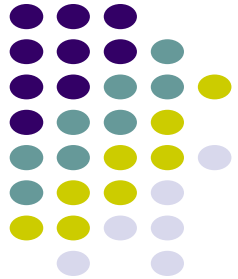
Увеличение частоты невозможно



- Проблемы вызываемые повышением частоты:
  - чрезмерное энергопотребление
  - выделение тепла
  - увеличение токов потерь

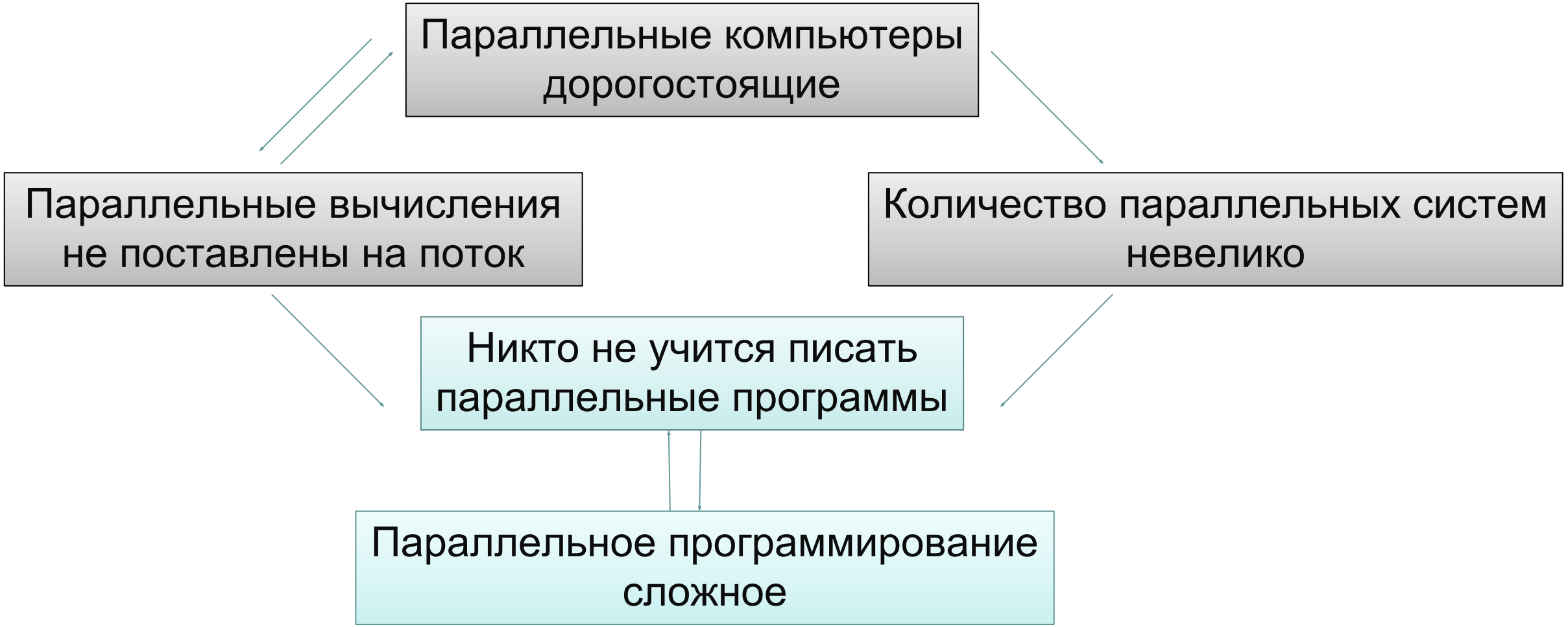
Энергопотребление критично в мобильных устройствах

Есть ли решения кроме параллелизма?

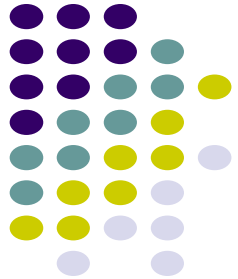


- Возможные оптимизации:
  - Instruction prefetching (предвыборка кода)
  - Instruction reordering (внеочередное исполнение)
  - Pipeline functional units (конвейерная обработка)
  - Branch prediction (предсказание переходов)
  - Hyper Threading
- Недостатки:
  - Усложнение схем => накладные расходы и потери на управление

Устаревшие мифы



Многоядерные процессоры



- Производительность ~= (core frequency) * (# cores)
- Стратегия:
  - Ограничить тактовую частоту и сложность архитектуры ядра
  - Расположить множество ядер на одном чипе

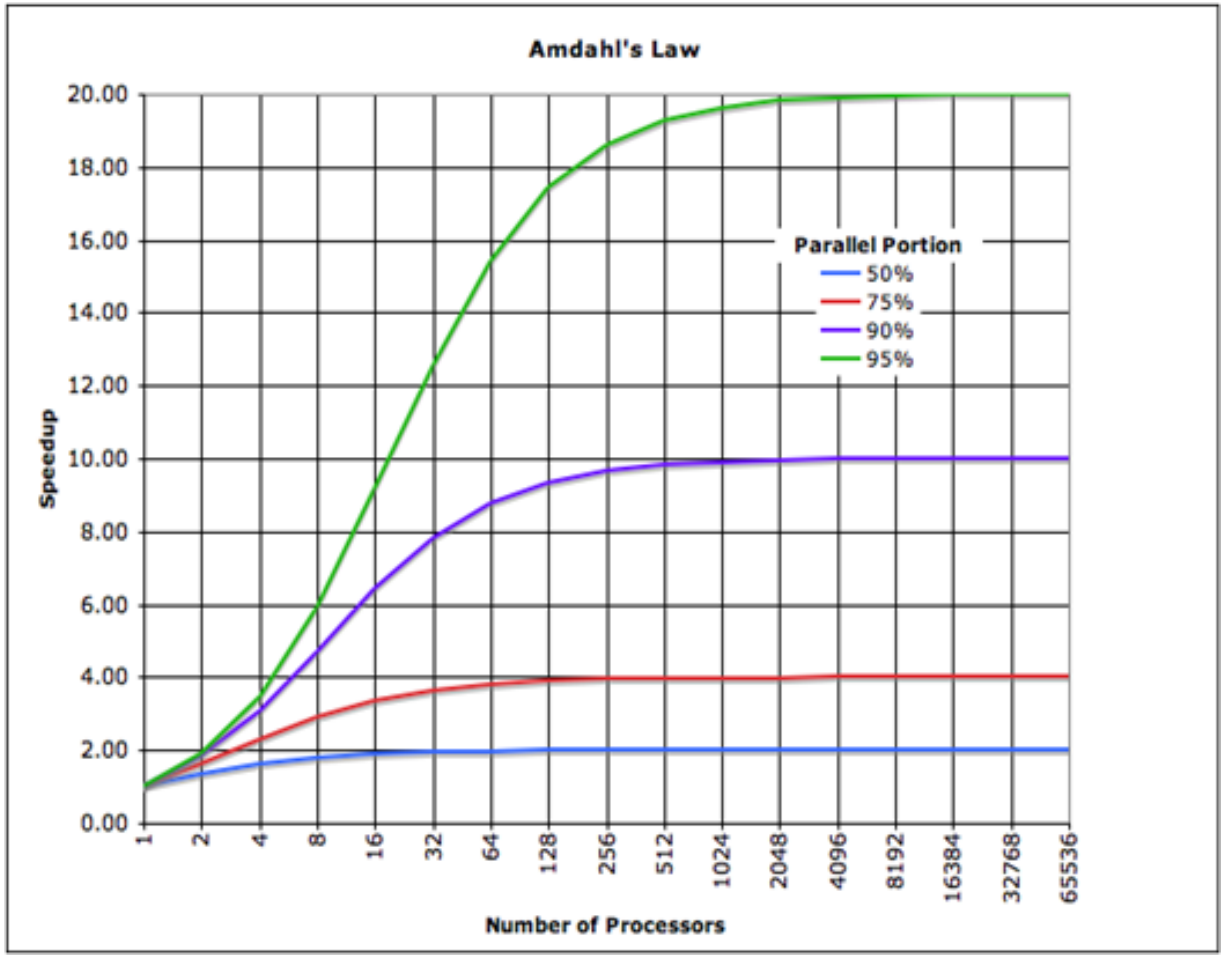
Классификации

Закон Амдала

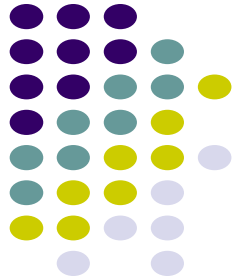
$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

S_p – ускорение
p – количество ядер
α – доля последовательного кода

α / p	10	100	1000
0	10	100	1000
10%	5.263	9.174	9.910
25%	3.077	3.883	3.988
40%	2.174	2.463	2.496



Методы декомпозиции

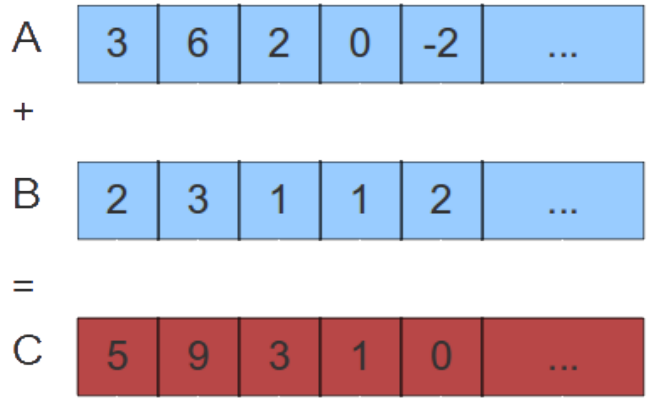


- По данным (domain decomposition)
- По задачам (task decomposition)
- Конвейер (pipelining)

Распараллеливание по данным

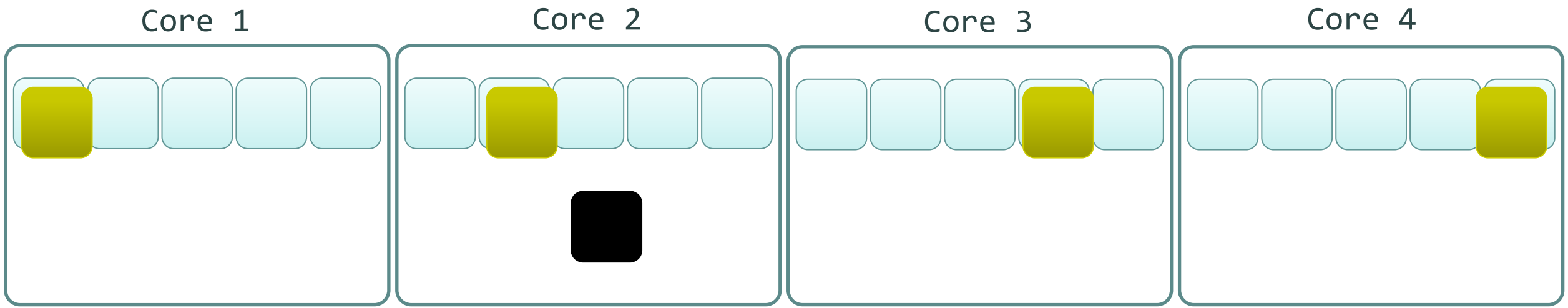
- Разделить блоки данных между ядрами
- Затем определить, какую задачу каждое ядро должно выполнять над блоком данных

Пример: сложение векторов



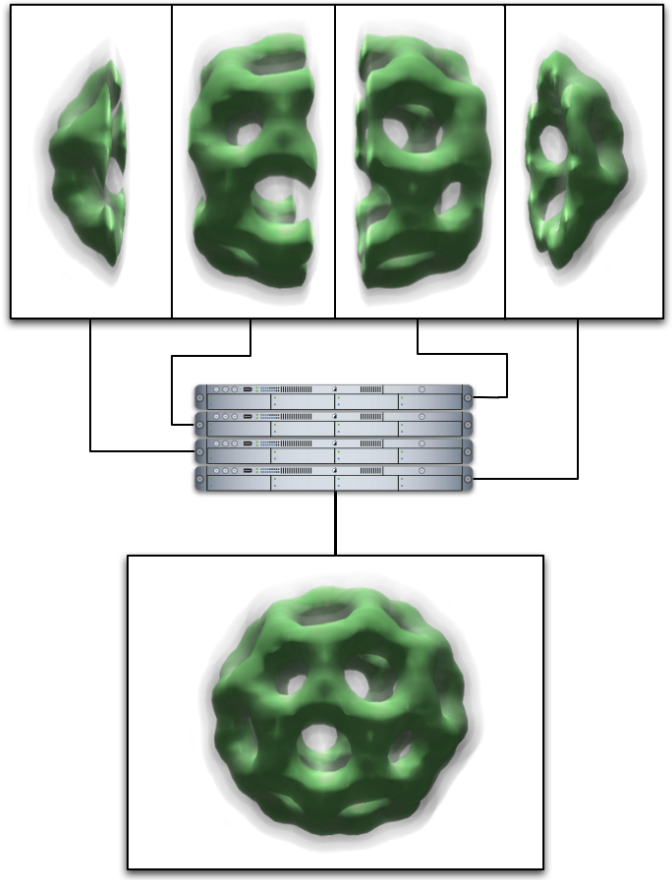
Распараллеливание по данным

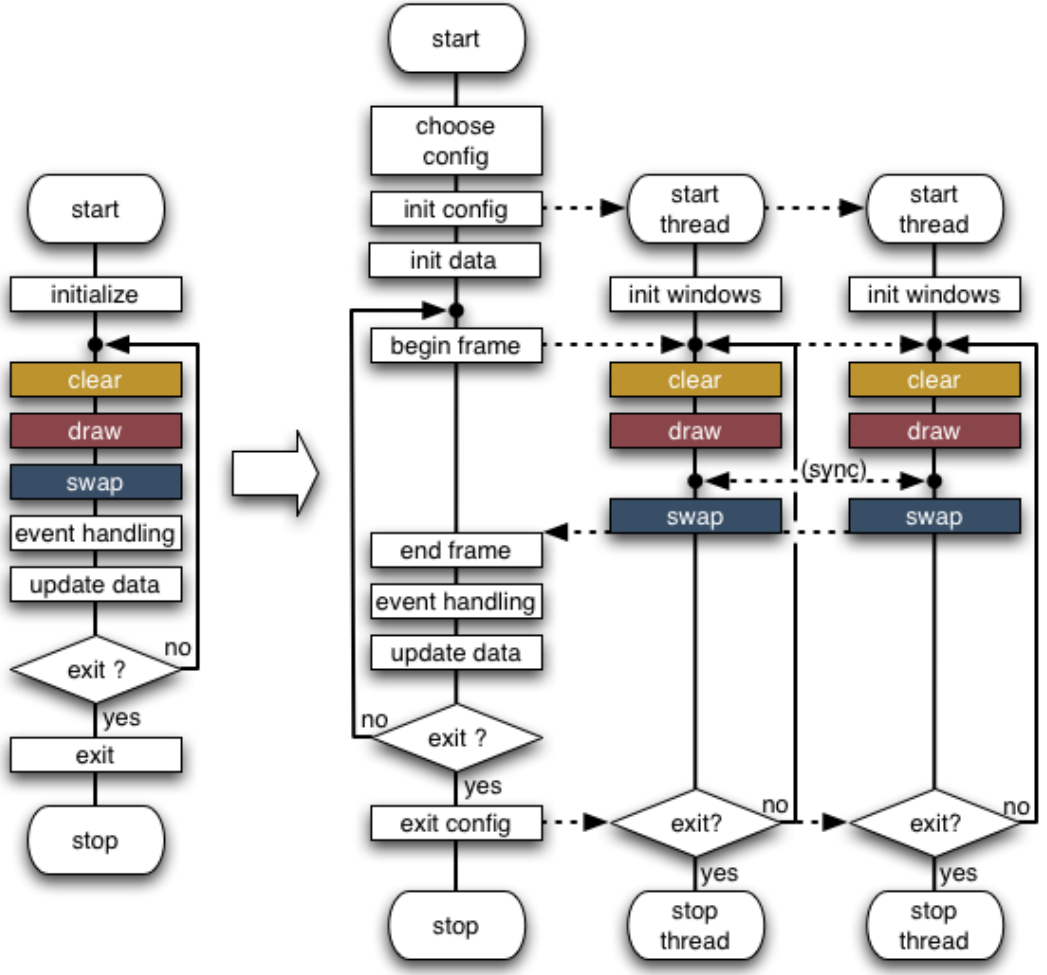
Поиск наибольшего элемента



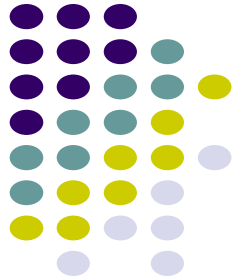
Распараллеливание по данным

Parallel Volume Rendering





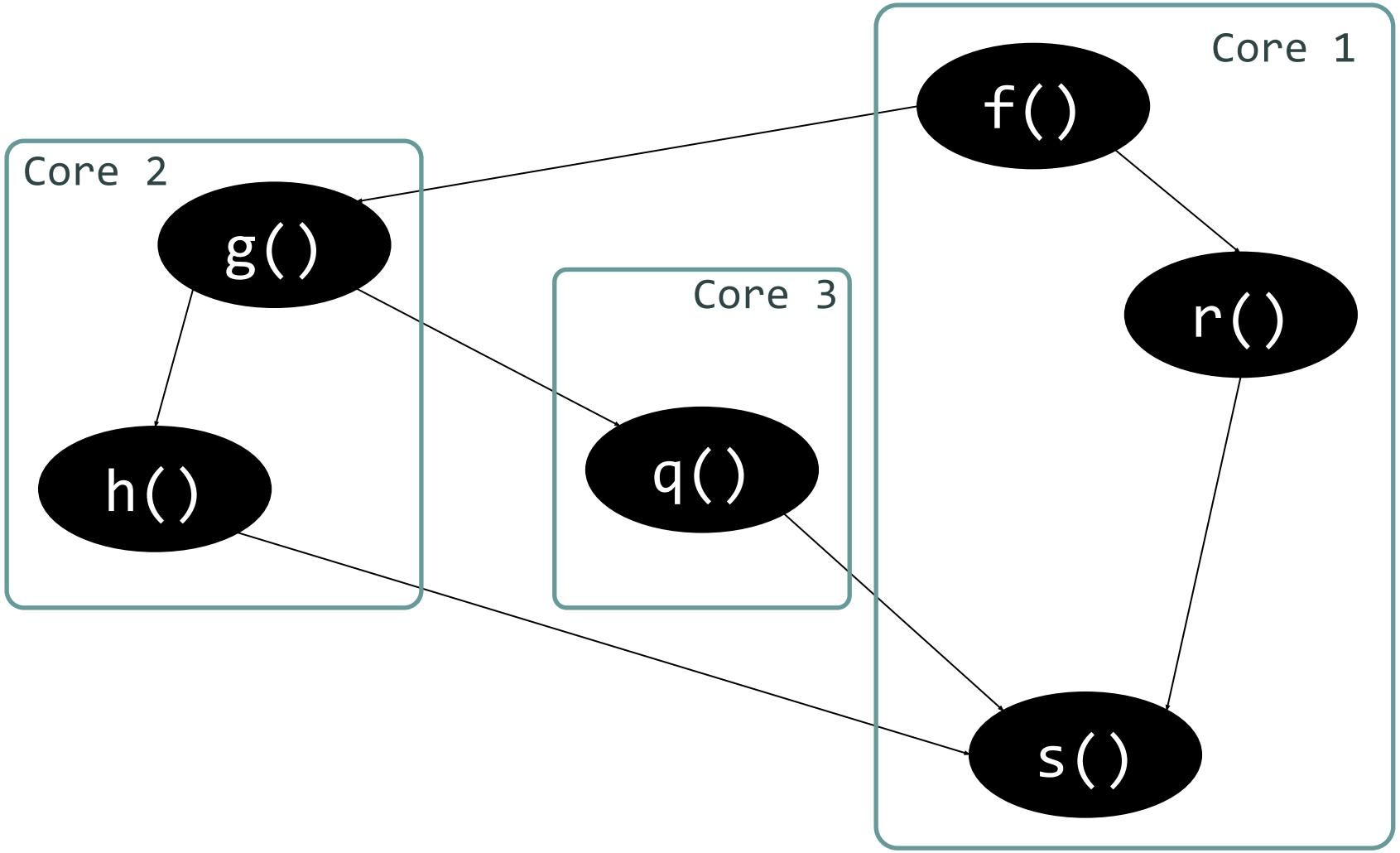
Распараллеливание по задачам



- Сначала разделить задачу на независимые подзадачи
- Затем определить блоки данных, к которым подзадача будет иметь доступ (чтение/запись)

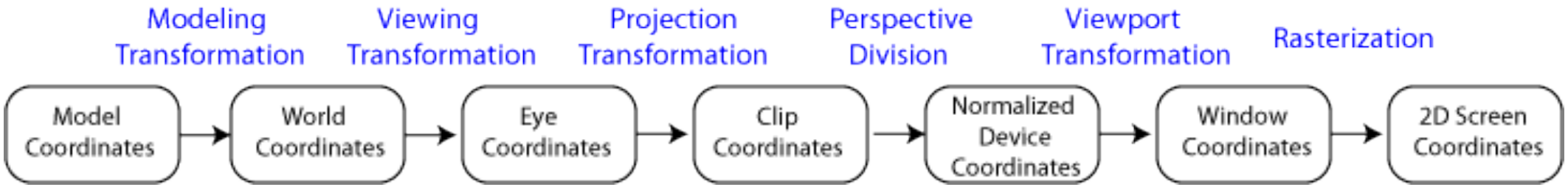
Пример: Обработчик событий GUI

Распараллеливание по задачам

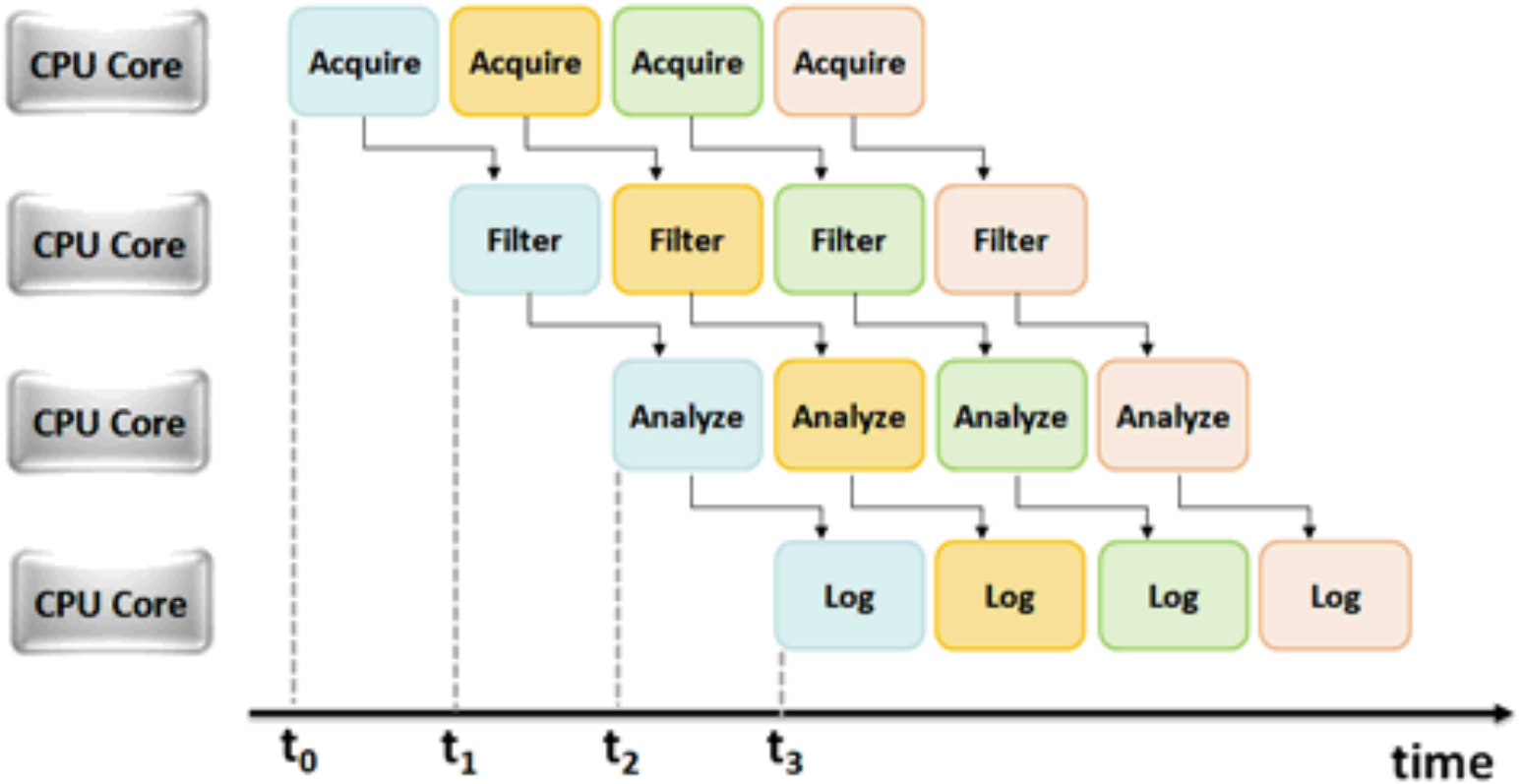


Конвейер

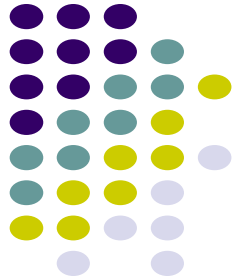
3D Rendering pipeline



Конвейер (множество ядер)



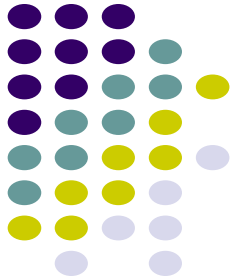
Как находить параллельные задачи?



- Работает ли программа с большим объемом данных?
- Есть ли участки кода, которые не имеют общего состояния и могут работать независимо?
- Есть ли последовательность вычислений, которые не взаимодействуют между собой, кроме входных и выходных данных?

Более формальный метод: построить граф зависимостей

Граф зависимостей

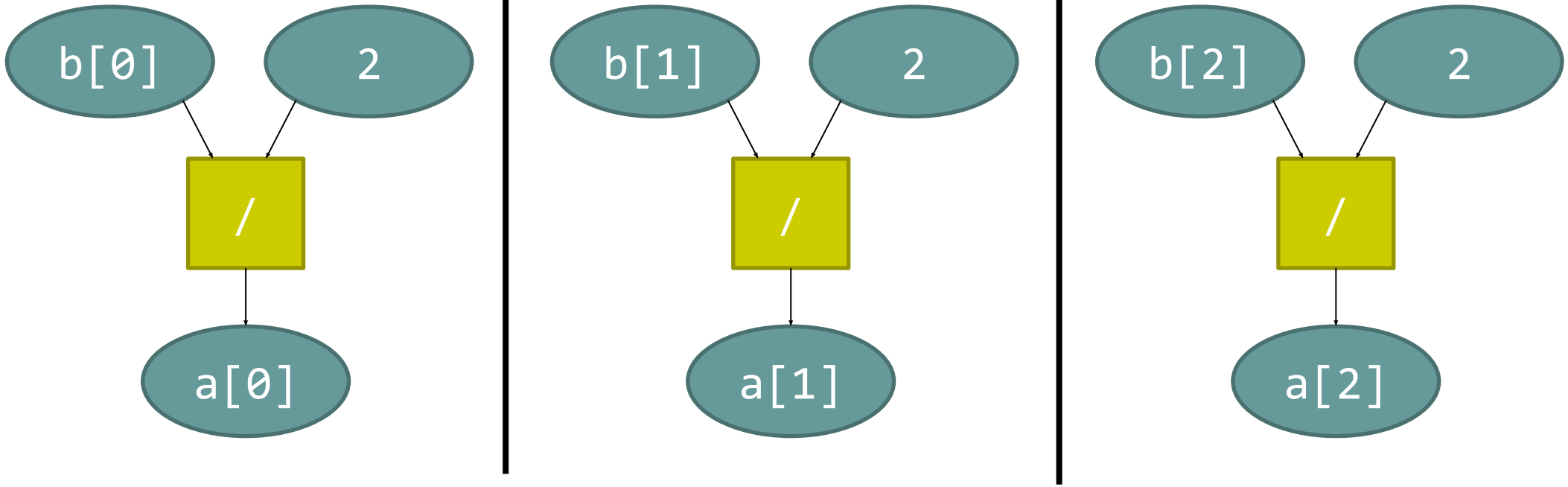


- Граф:
  - Узлы
  - Ребра (стрелки)
- Узлы представляют собой:
  - Присваивание значения (не учитывая индексы и счетчики)
  - Константы
  - Операторы или вызовы функций
- Стрелки:
  - Потоки данных и потоки выполнения

Граф зависимостей #1

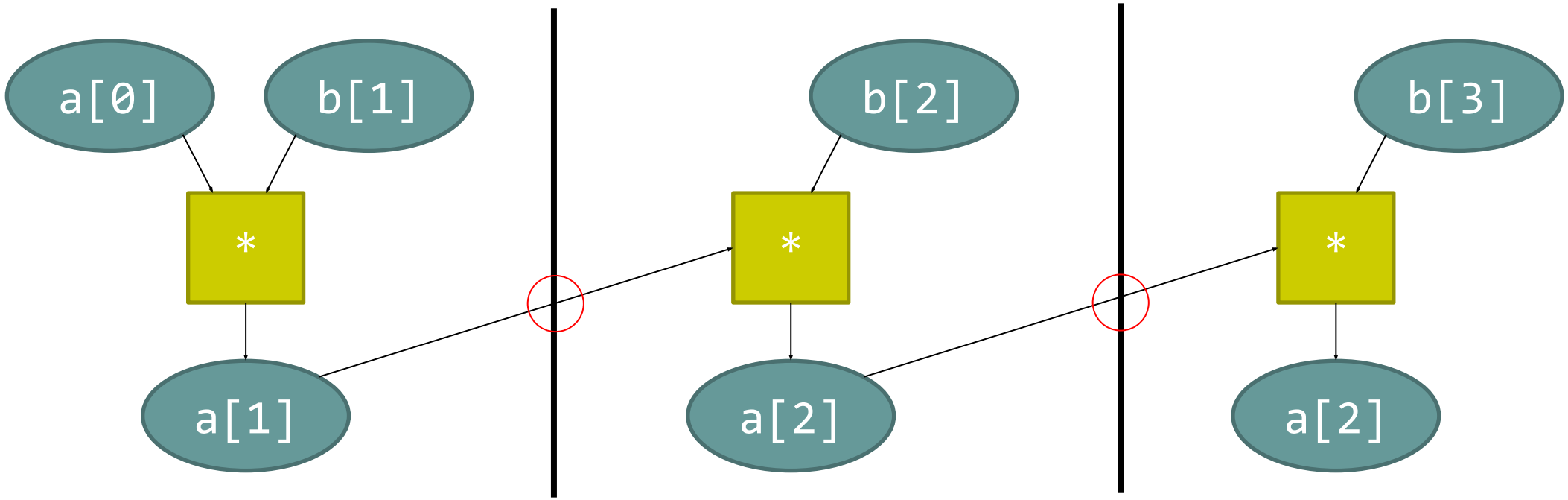
```
for (i = 0; i < 3; i++) {
   a[i] = b[i] / 2.0;
}
```

Распараллеливание по данным возможно



Граф зависимостей #2

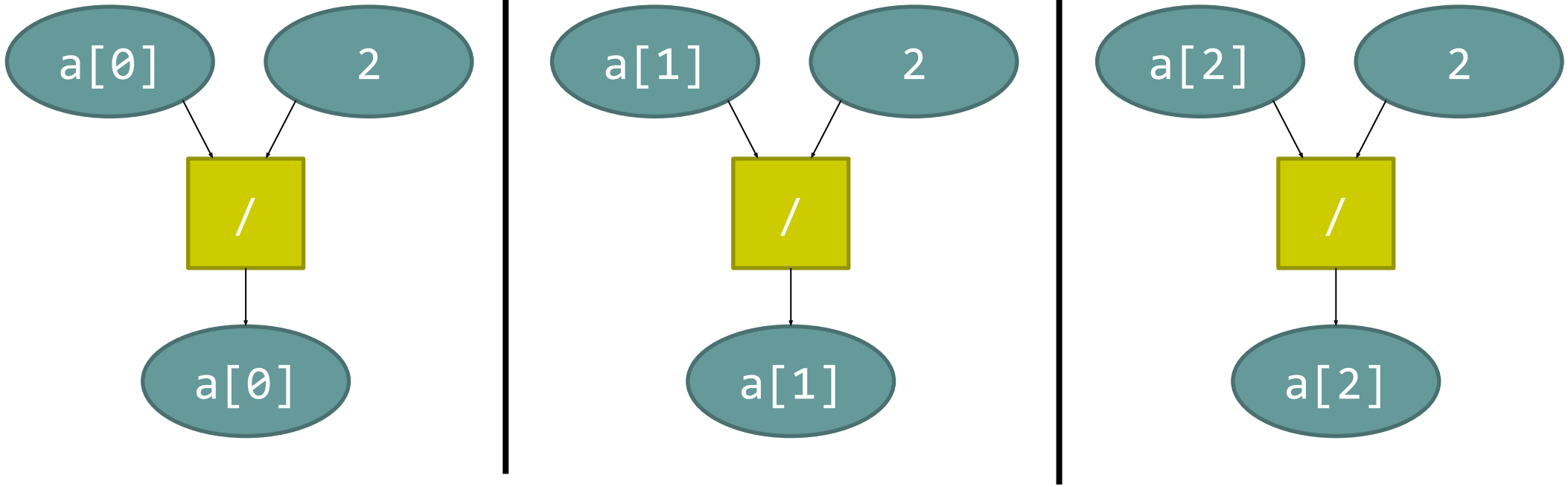
```
for (i = 1; i < 4; i++) {
   a[i] = a[i-1] * b[i];
}
```



Граф зависимостей #3

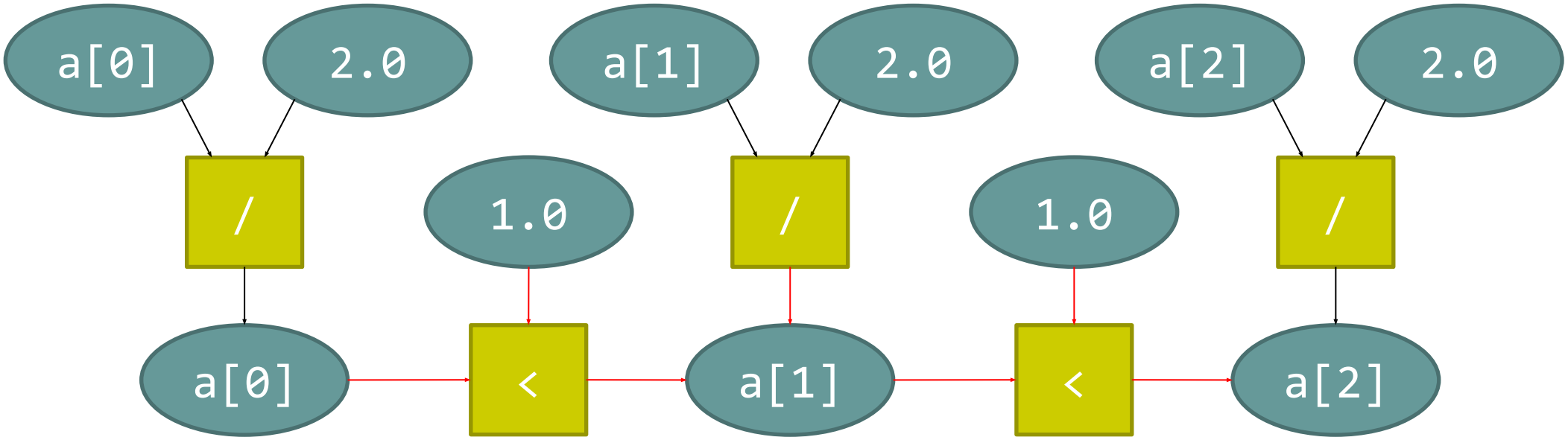
```
for (i = 0; i < 3; i++) {
   a[i] = a[i] / 2.0;
}
```

Распараллеливание по данным возможно

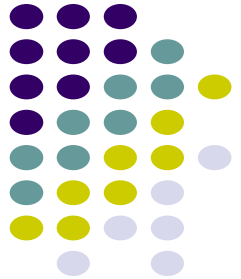


Граф зависимостей #4

```
for (i = 0; i < 3; i++) {
    a[i] = a[i] / 2.0;
    if(a[i] < 1.0) break;
}
```



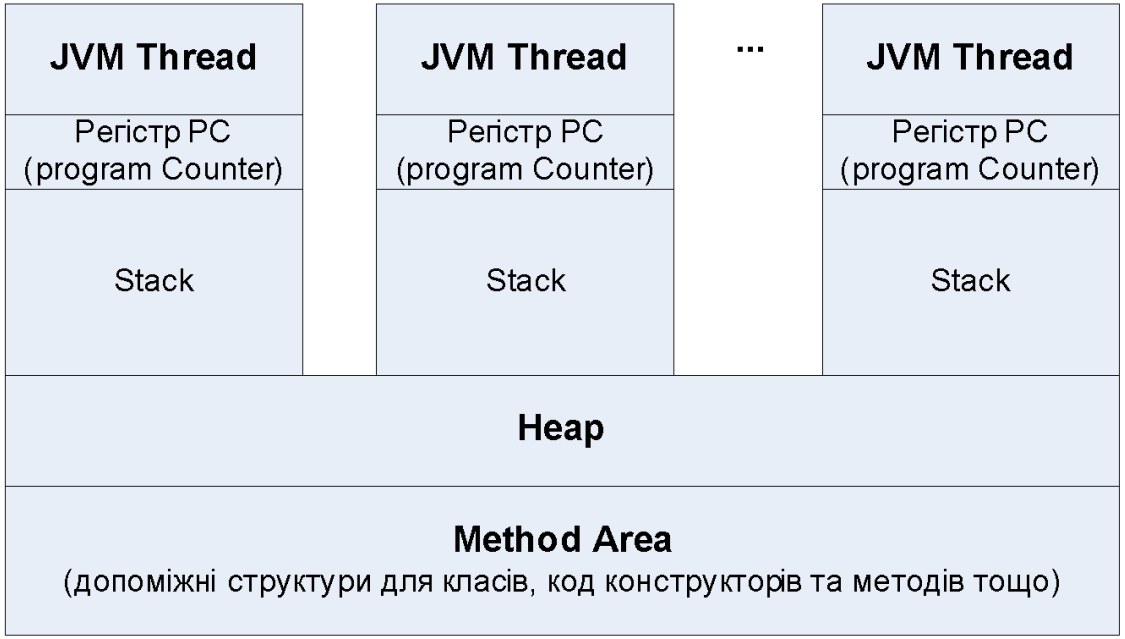
Возможно ли это распараллелить?



- Масштабирование изображения
- Поиск слова в документе
- Обновление полей электронной таблицы
- Компиляция программы
- Индексирование веб страниц поисковым роботом

Java и потоки

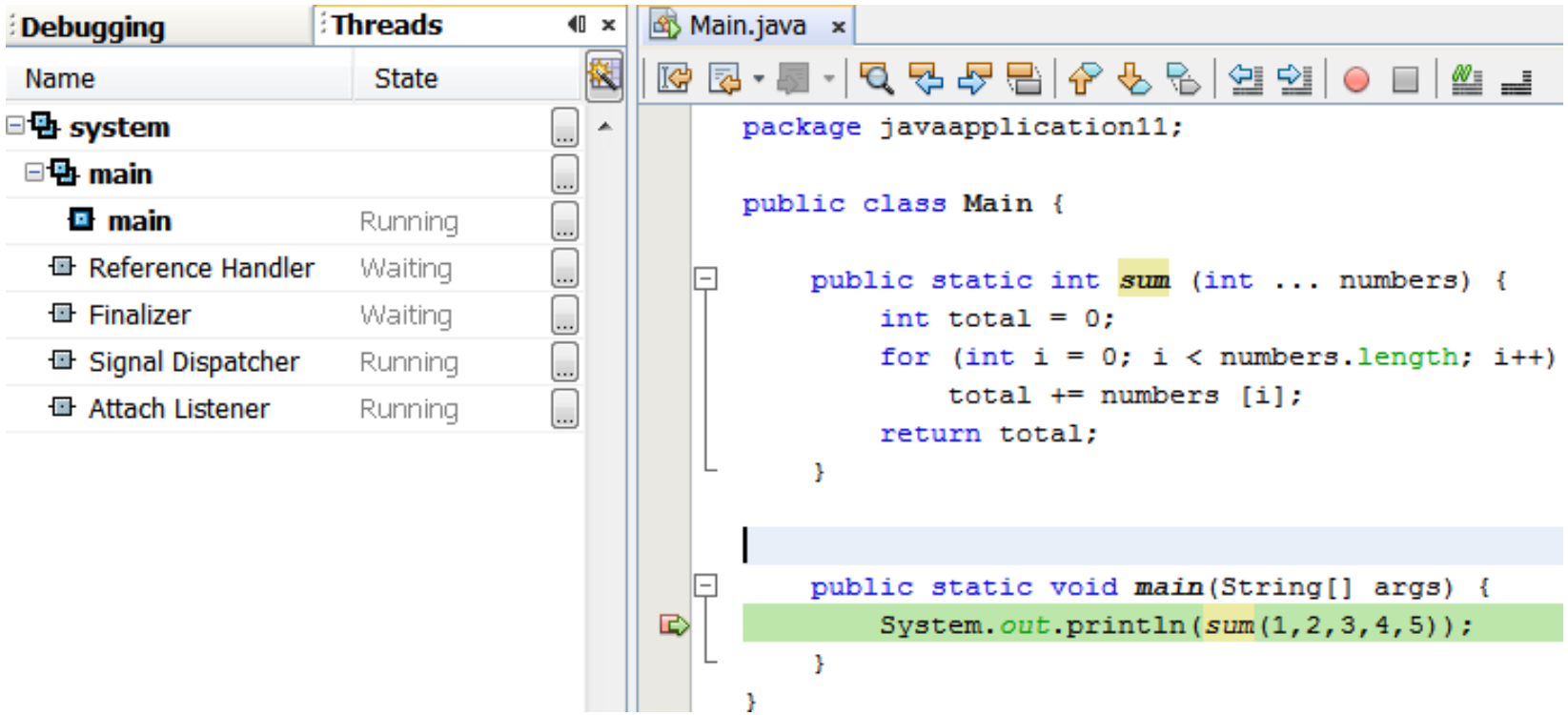
- Потоки (Threads) – это части программы, которые могут выполняться параллельно
- Java имеет встроенные средства для работы с потоками
- Это достигается за счет того, что JVM иммет собственную реализацию потоков (JVM Thread)
- Потоки JVM отображаются на потоки ОС и тем самым используют системные ресурсы



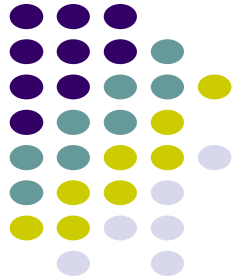
Структура JVM

Потоки для Java-программы

- Потоки для простой Java-программы:
  - Поток “Main” (с нем выполняется метод main)
  - Системные потоки (garbage collector, …)

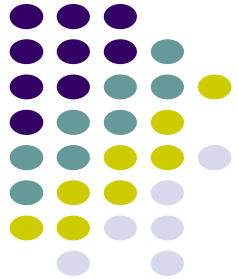


Мониторинг потоков

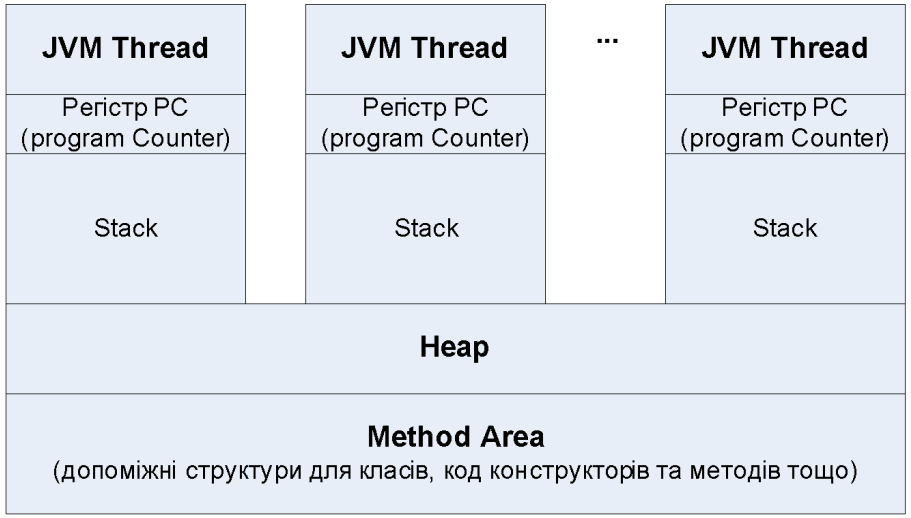


- jConsole
- jVisualVM
- Jstack

Понятие потока выполнения в Java

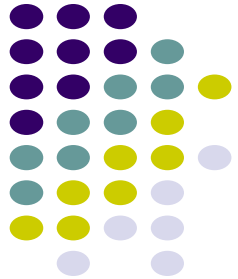


- Каждый поток имеет свой стек и регистр указателя программы program counter
- Могут выполнять один и тот же код (но каждый поток будет выполнять код со своим стеком)
- Имеют доступ к одному и тому же адерсному пространству (JVM Heap) и могут манипулировать одними и теме же данными



- Потоки Java – это экземпляры класса `java.util.Thread`

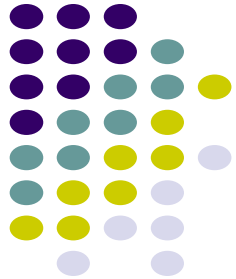
Запуск потока



1. Определить код который будет выполнять поток
2. Создать экземпляр потока и задать ему код для выполнения
3. Запустить поток

- Для старта потока используется метод **`start`** класса **`java.util.Thread`**

#1 Определение кода для выполнения



- **Определить код непосредственно в потоке**. Для этого расширить класс **`java.lang.Thread`**

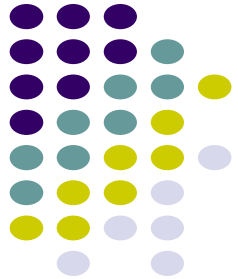
```
public class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("MyThread");
    }
}
```

- **Код в отдельном классе**. Для этого реализовать интерфейс **`java.lang.Runnable`**

```
class MyRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("MyRunnable");
    }
}
```

`Runnable` – это лучший способ для задания кода потока поскольку позволяет отделить «полезную работу», которую поток выполняет от деталей реализации управления

#2. Создание экземпляра потока



- если код в самом потоке

```
MyThread t = new MyThread();
```

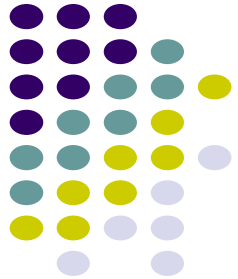
- если код определен в **Runnable**

```
MyRunnable r = new MyRunnable();
Thread t = new Thread(r);
```

Один экземлпяр **Runnable** можно передать нескольким потокам

```
MyRunnable r = new MyRunnable();
Thread t1 = new Thread(r);
Thread t2 = new Thread(r);
Thread t3 = new Thread(r);
```

#3. Запуск потока



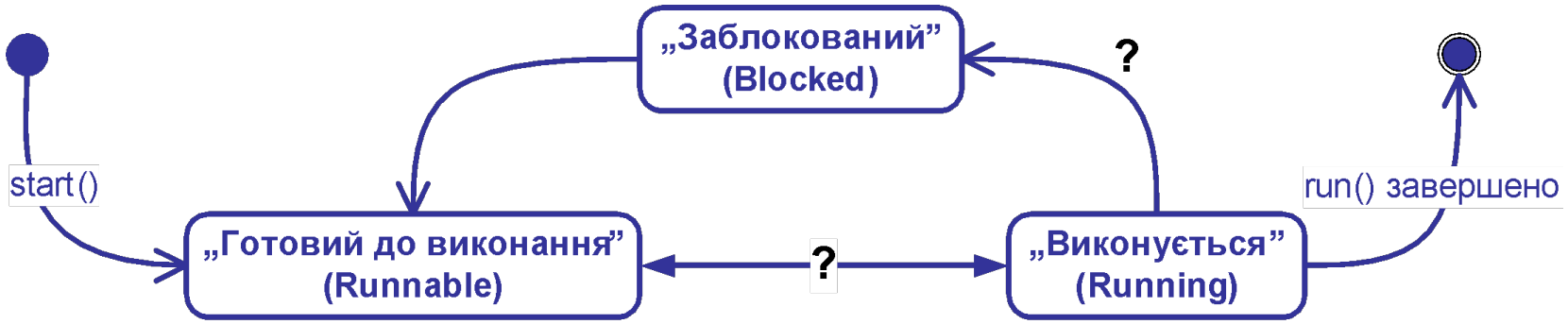
- С помощью метода **`start`**()

```
public static void main(String[] args) {
    MyRunnable r = new MyRunnable();
    Thread t = new Thread(r);
    t.start();
}
```

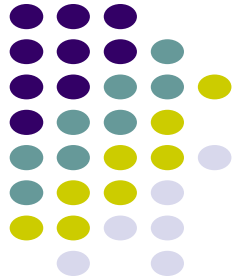
Основные состояния потоков

- После старта поток не не сразу получает системные ресурсы
- Основные состояния:
  - **Готов к выполнению** – переходит в это состояние после вызова `start`
  - **Выполняется** – означает что потоку были выделены системные ресурсы
  - **Заблокирован** – выполнение этого потока было приостановлено

- После завершения перезапусить тот же поток нельзя
- Проверить состояние потока можно методом `isAlive`

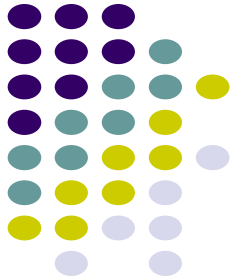


Ожидание завершения. Join



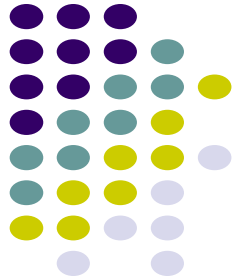
```
public static void main(String[] args) {

    Thread t = new Thread(new NamedRunnable());
    t.start();

    ...

    // Ожидаем пока поток не завершит работу
    try {
        t.join();
    }
    catch (InterruptedException e) {
        ...
    }
     ...
}
```

Остановка потока



- Корректного метода принудительного завершения потока извне – НЕТ
- Метод `stop()` считается неподдерживаемым (deprecated)
- Метод `interrupt()` устанавливает флаг завершения, но не завершает. Поток должен сам проверить этот флаг и завершиться корректно

Подытожим. Класс Thread



Основні методи	Конструктори
static **currentThread** static **dumpStack** static **getAllStackTraces** **getId/setId** **getName/setName** **getPriority/setPriority** **getState** **interrupt** **isAlive** **isDaemon/setDaemon** **join** **run** **sleep** **start** **yield**	**Thread**() **Thread**(String name) **Thread**(Runnable runnable) **Thread**(Runnable runnable, String name) **Thread**(ThreadGroup g, Runnable runnable) **Thread**(ThreadGroup g, Runnable runnable, String name) **Thread**(ThreadGroup g, String name)