Java

Многопоточность

План

- Многозадачность/многопоточность
- Процессы и потоки
- Потоки в Java
- ThreadPool

“Однопоточные” компьютеры

- Одноядерные процессоры
- Могли выполнять только одну программу
- Взаимодействие выполнялось при помощи прерываний
- Можно ли на одном ядре выполнять несколько программ?

Современные компьютеры

- Для реализации многозадачности процессор (ядро) переключается между задачами
- Большинство процессоров сейчас имеют несколько ядер
- Часто UI поток отделен от основного для реализации плавного интерфейса

Для чего нужна многопоточность

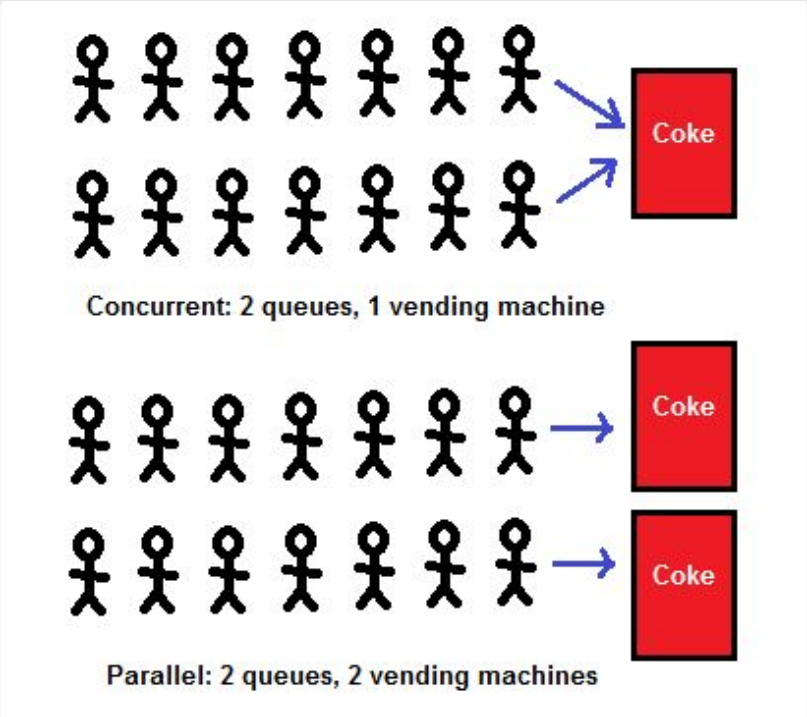
- многозадачность
- оптимальное использование процессорного времени
- асинхронная обработка
- повышение производительности

Многопоточное программирование

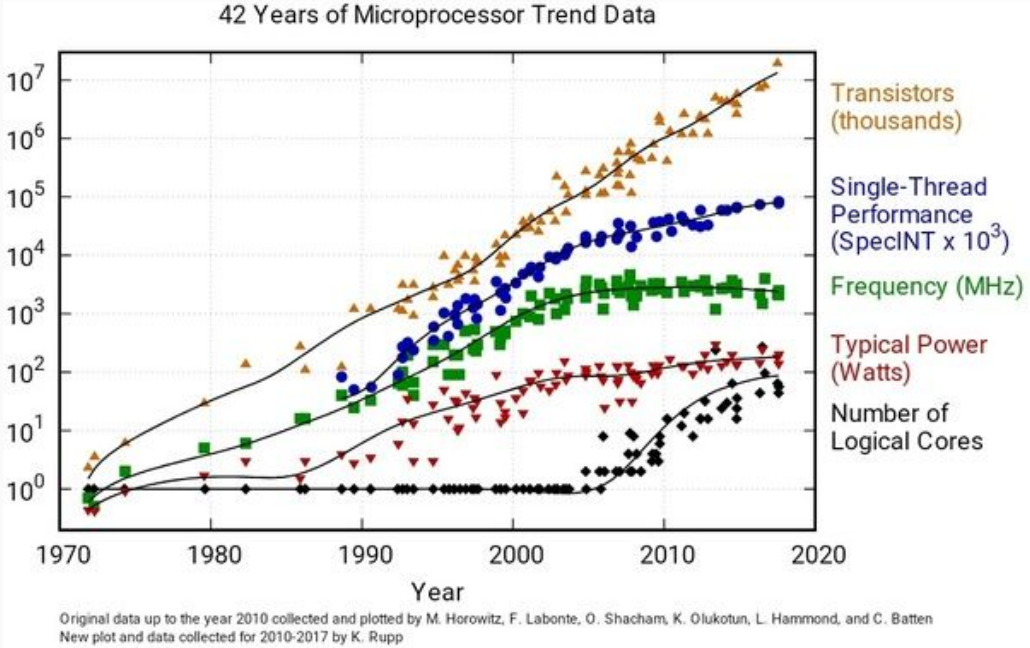
Процессы и потоки

- Процесс имеет свою память, к которой доступ есть только у него
  - Процесс состоит из потоков
  - Сложно осуществлять синхронизацию и общение
  - Больше занимает памяти
- Поток находится внутри процесса
  - У потоков общая память
  - Является более лёгким вариантом процесса
- Если убить поток то процесс останется жив, а если убить процесс то его потоки умрут

Concurrent vs Parallel



Закон Мура


42 Years of Microprocessor Trend Data
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1970
1980
1990
2000
2010
2020
Year
Transistors (thousands)
Single-Thread Performance (SpecINT x 10^3)
Frequency (MHz)
Typical Power (Watts)
Number of Logical Cores
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp


M1

Создание Runnable в Java

```java
public class MyRunnable implements Runnable
{
    @Override
    public void run()
    {
        for (int i = 0; i < 10; i++)
        {
            System.out.println("Number: " + i);
        }
    }
}
```

```java
public static void main2(String[] args)
{
    Thread thread = new Thread(new MyRunnable());
    thread.run();
}
```

Создание потока в Java

```java
public class MyThread extends Thread
{
    @Override
    public void run()
    {
        for (int i = 0; i < 10; i++)
        {
            System.out.println("Number: " + i);
        }
    }
}
```

```java
public static void main(String[] args)
{
    Thread thread = new MyThread();
    thread.start();
}
```

Создание задания через Timer

```
public static void main(String[] args)
{
    Timer timer = new Timer();
    timer.schedule(new TimerTask()
    {
        @Override
        public void run()
        {
            System.out.println("Timer task");
        }
    }, delay: 1000);
}
```

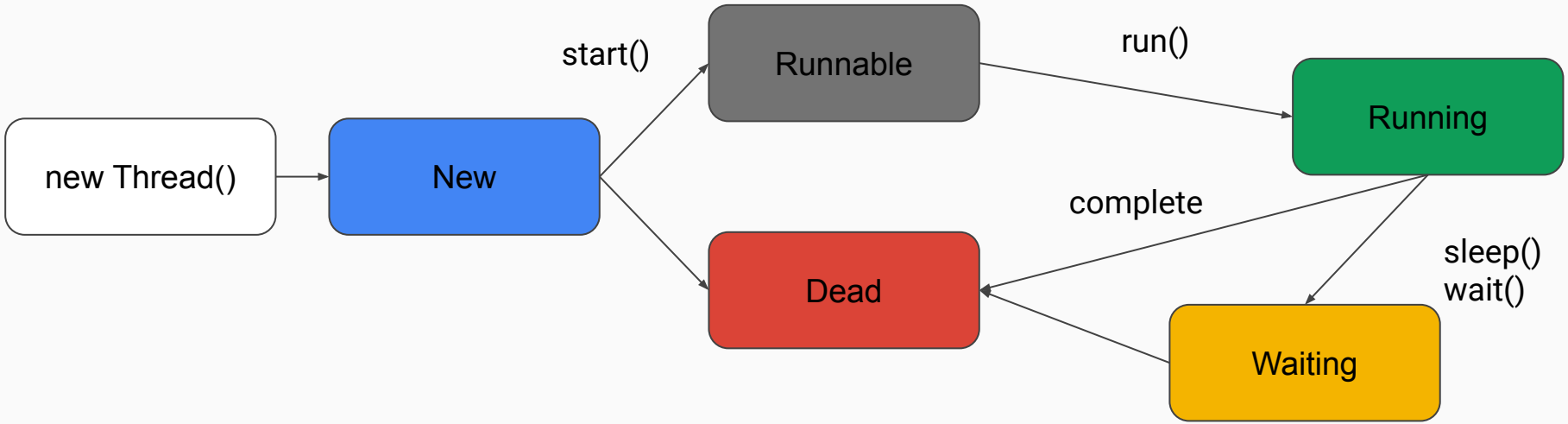
Методы работы с потоками (Thread)

- start() - Запуск потока
- sleep() - Пауза выполнения потока (статический)
- yield() - Сообщить о готовности отдать ресурсы процессора
- setDaemon() - Сделать демоном, не ожидаем его завершения
- join() - Объединить с текущим

Методы Object

- wait() - Ожидать notify
- notify() - Отправить notify одному (любому) потоку
- notifyAll() - Отправить notify всем потокам

Жизненный цикл потока



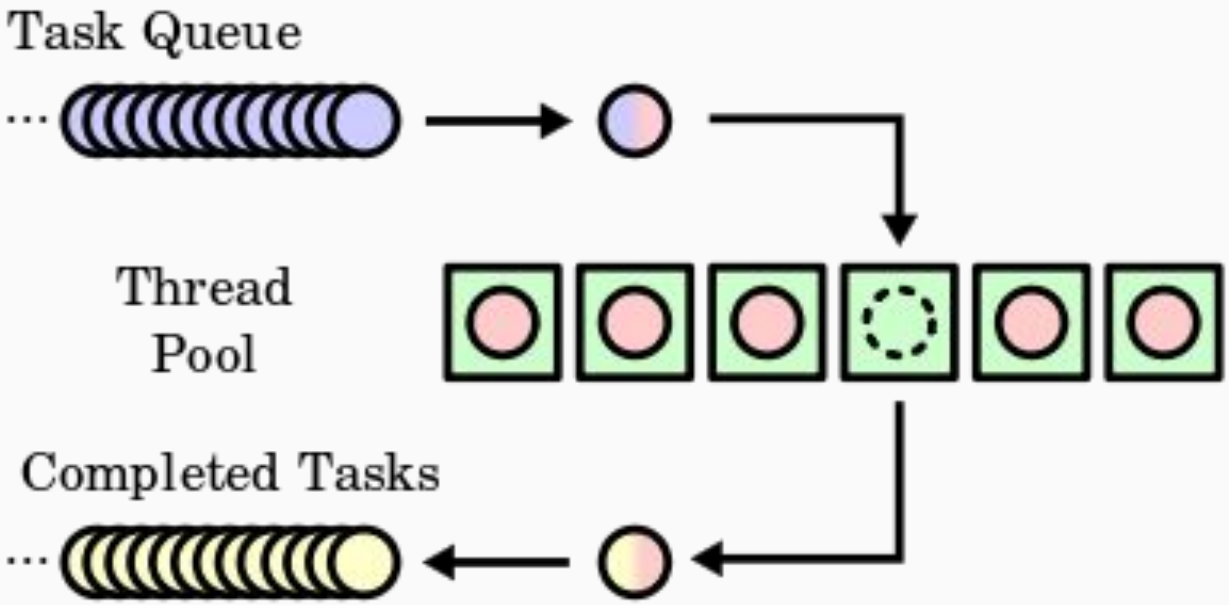
Как остановить поток

- stop() - @Deprecated, опасный
- interrupt()
  - Ставится флаг interrupted
  - Выбрасывается InterruptedException
  - Нужно обрабатывать код внутри
- В “общем” случае не остановить

Проблема потоков

- Создавать потоки не очень “дёшево”
- Управлять потоками для достижения равномерной работы на “железе” достаточно сложно

Thread pool



Thread pool

```
public static void main(String[] args)
{
    ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 12);
    executorService.execute(new MyRunnable());
    executorService.submit(new MyRunnable());

    int cores = Runtime.getRuntime().availableProcessors();
}
```