



Java Programowanie II

wielowątkowość

Autor: Grzegorz Witczak

Prawa do korzystania z materiałów posiada Software Development Academy

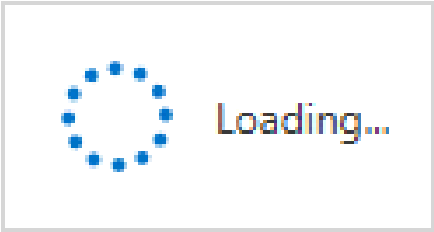
LOADING...



DOWNLOADING...

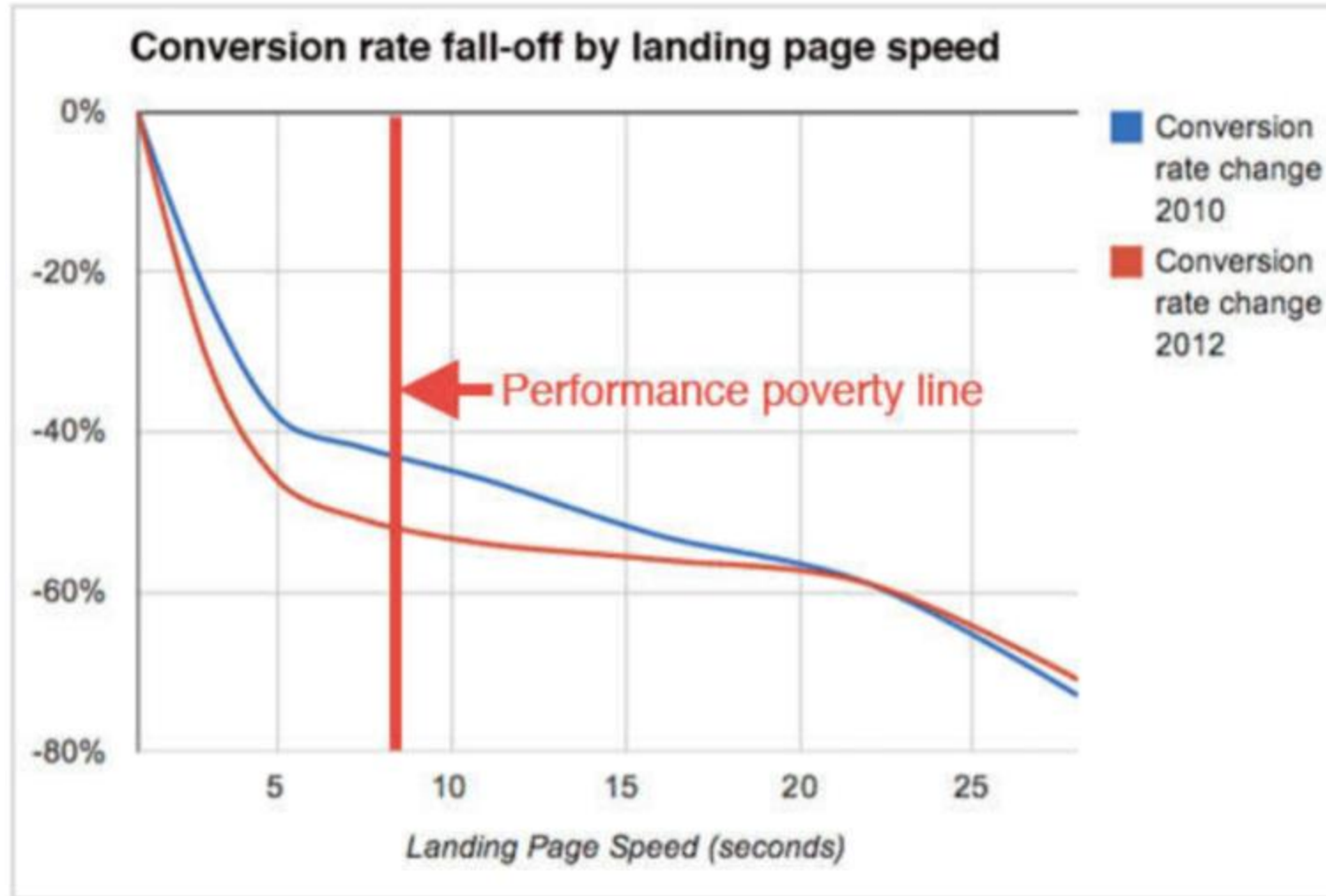


75 %











"1 second of load lag time would cost Amazon **\$1.6 billion in sales** per year" ¹

- Amazon

"When load times jump from 1 seconds to 4 seconds, conversions decline sharply. For every 1 second of improvement, we experience a 2% conversion increase" ³

- Walmart

"A lag time of 400ms results in a decrease of 0.44% traffic - In real terms this amounts to **440 million abandoned sessions/month** and a massive loss in advertising revenue for Google" ²

- Google

"An extra 0.5 seconds in each search page generation would cause traffic to drop by 20%" ²

- Google

Walmart.com

found that when load times jump from **1 second** to **4 seconds**, conversions decline sharply.

And for every **1 second** of improvement, they experienced up to a **2% conversion increase**.



Latency Numbers Every Programmer Should Know

■ 1 ns

■ L1 cache reference: 0.5 ns

■ Branch mispredict: 5 ns

■ L2 cache reference: 7 ns

■ Mutex lock/unlock: 25 ns

■ = 100 ns

■ Main memory reference: 100 ns

■ = 1 μ s

■ Compress 1 KB with Zippy: 3 μ s

■ = 10 μ s

■ Send 1 KB over 1 Gbps network: 10 μ s

■ SSD random read (1 Gb/s SSD): 150 μ s

■ Read 1 MB sequentially from memory: 250 μ s

■ Round trip in same datacenter: 500 μ s

■ = 1 ms

■ Read 1 MB sequentially from SSD: 1 ms

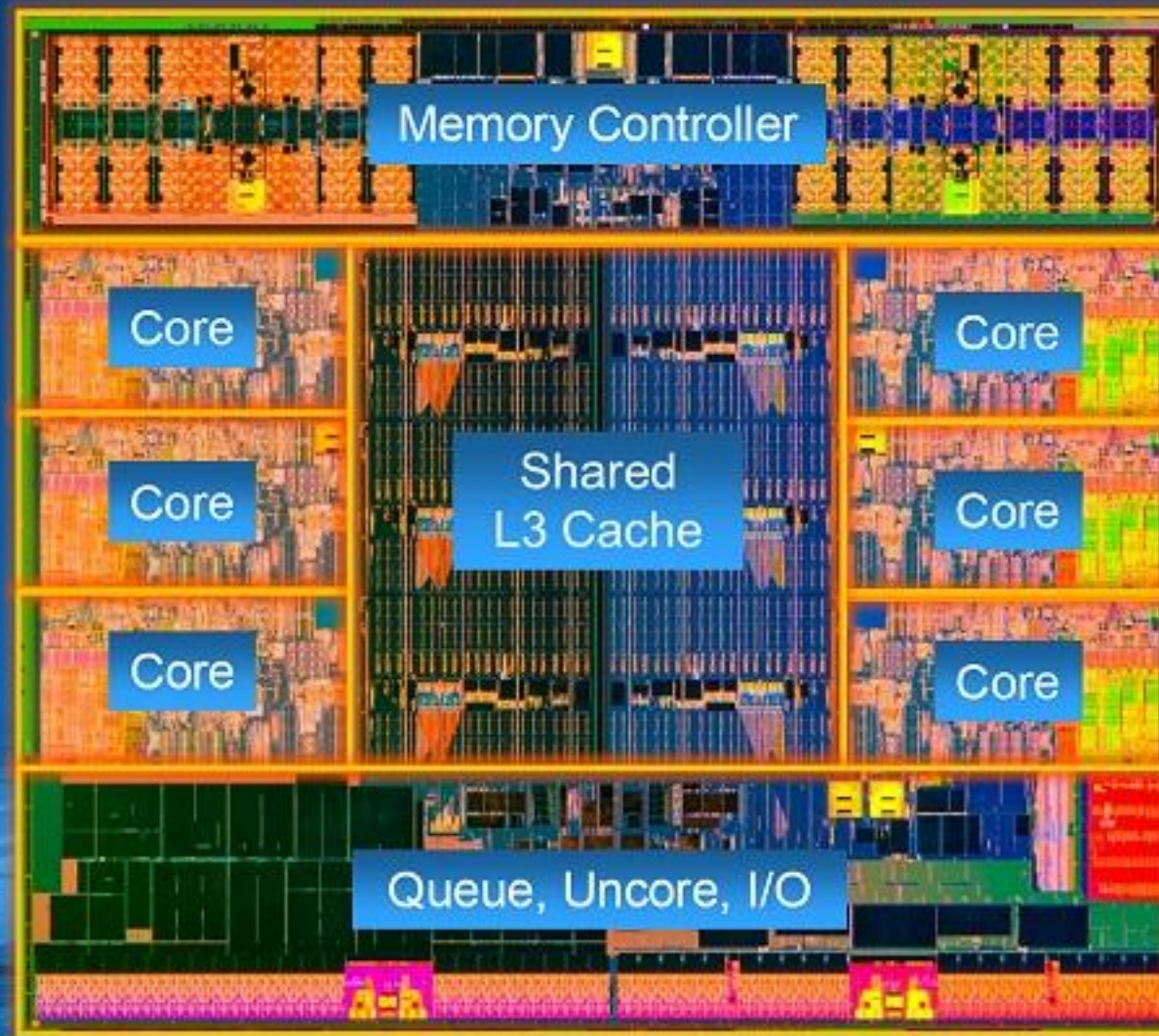
■ Disk seek: 10 ms

■ Read 1 MB sequentially from disk: 20 ms

■ Packet roundtrip CA to Netherlands: 150 ms

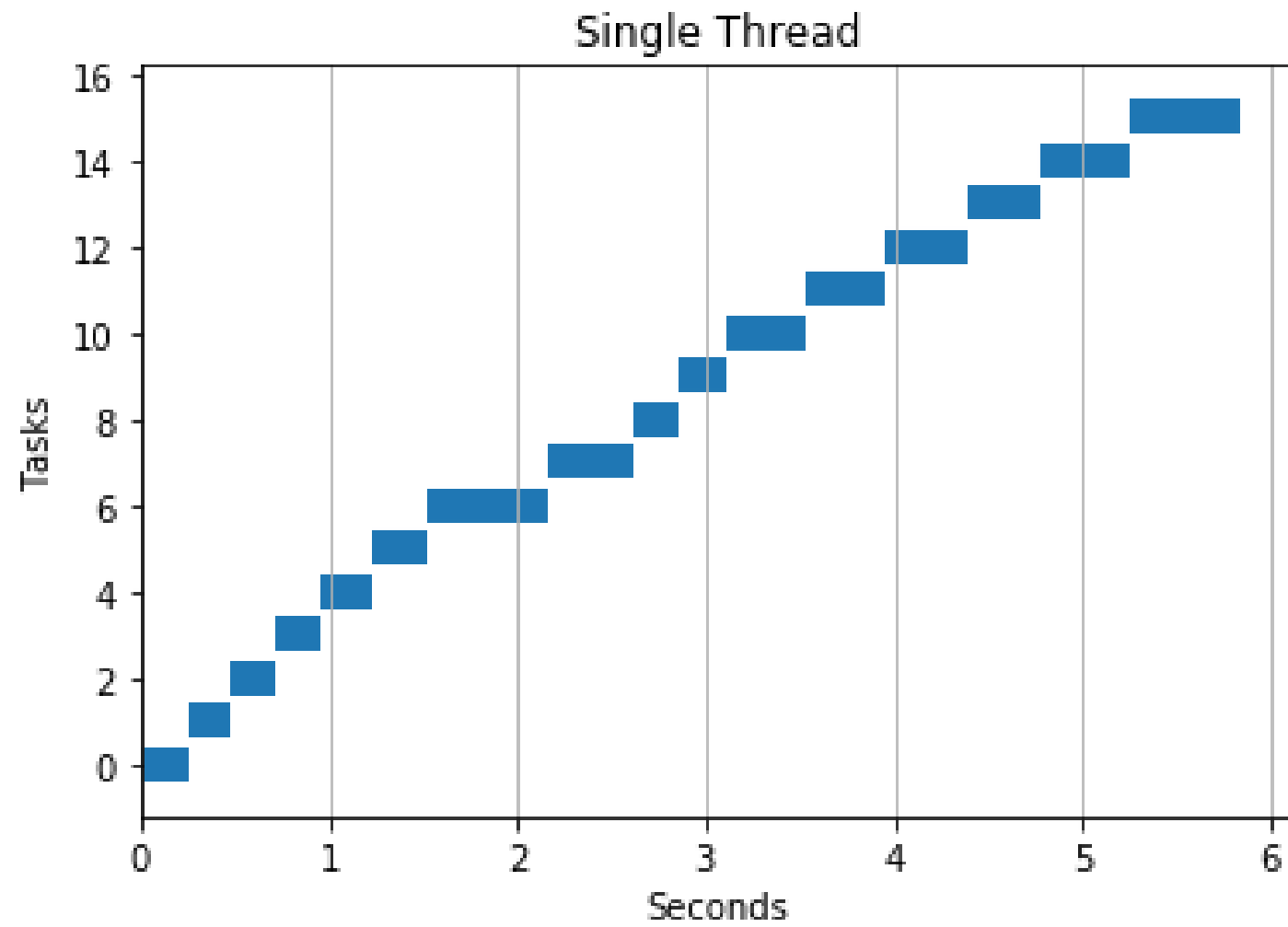
Source: <https://gist.github.com/2841832>

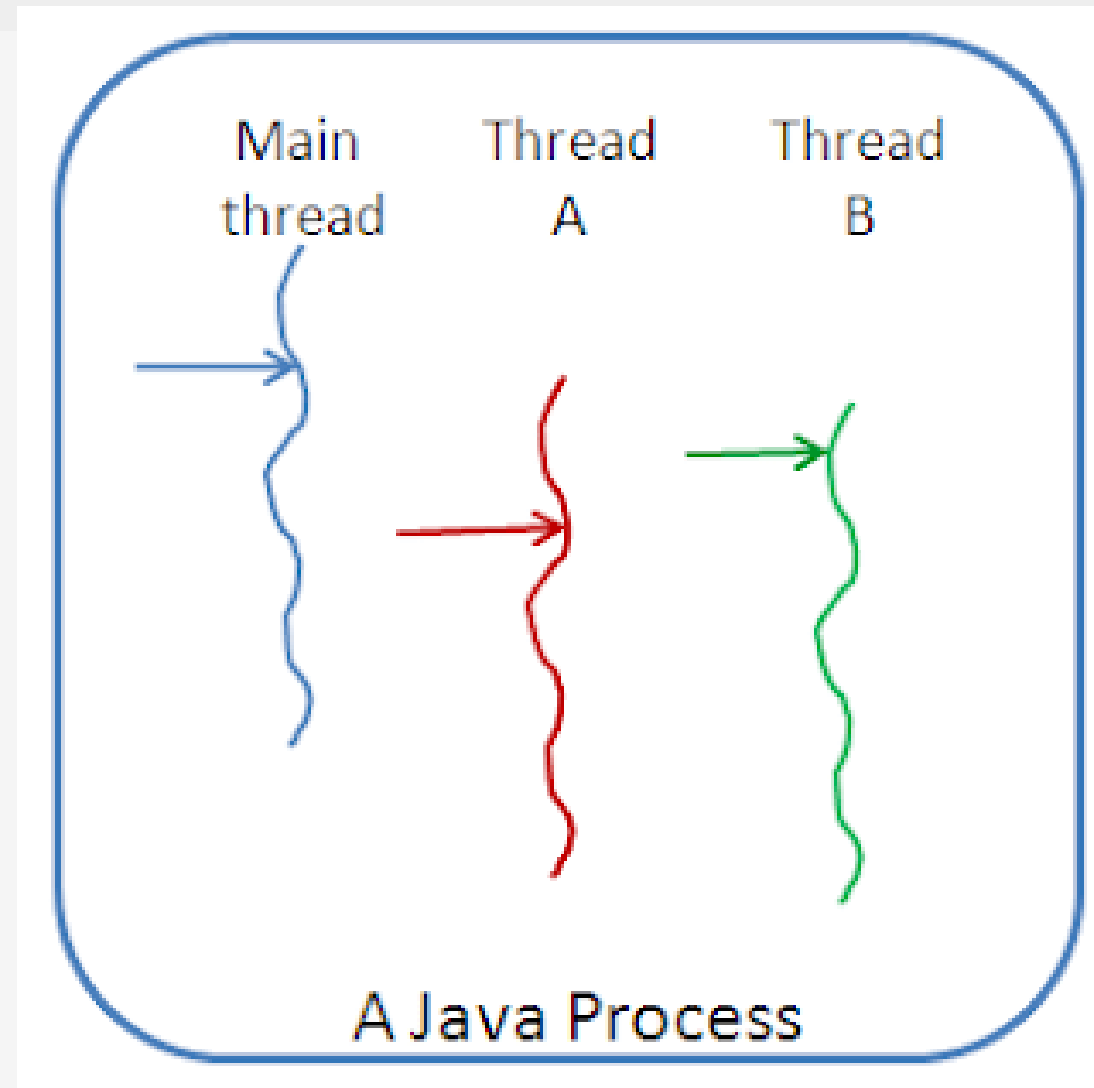
Intel® Core™ i7-4960X Processor Die Detail

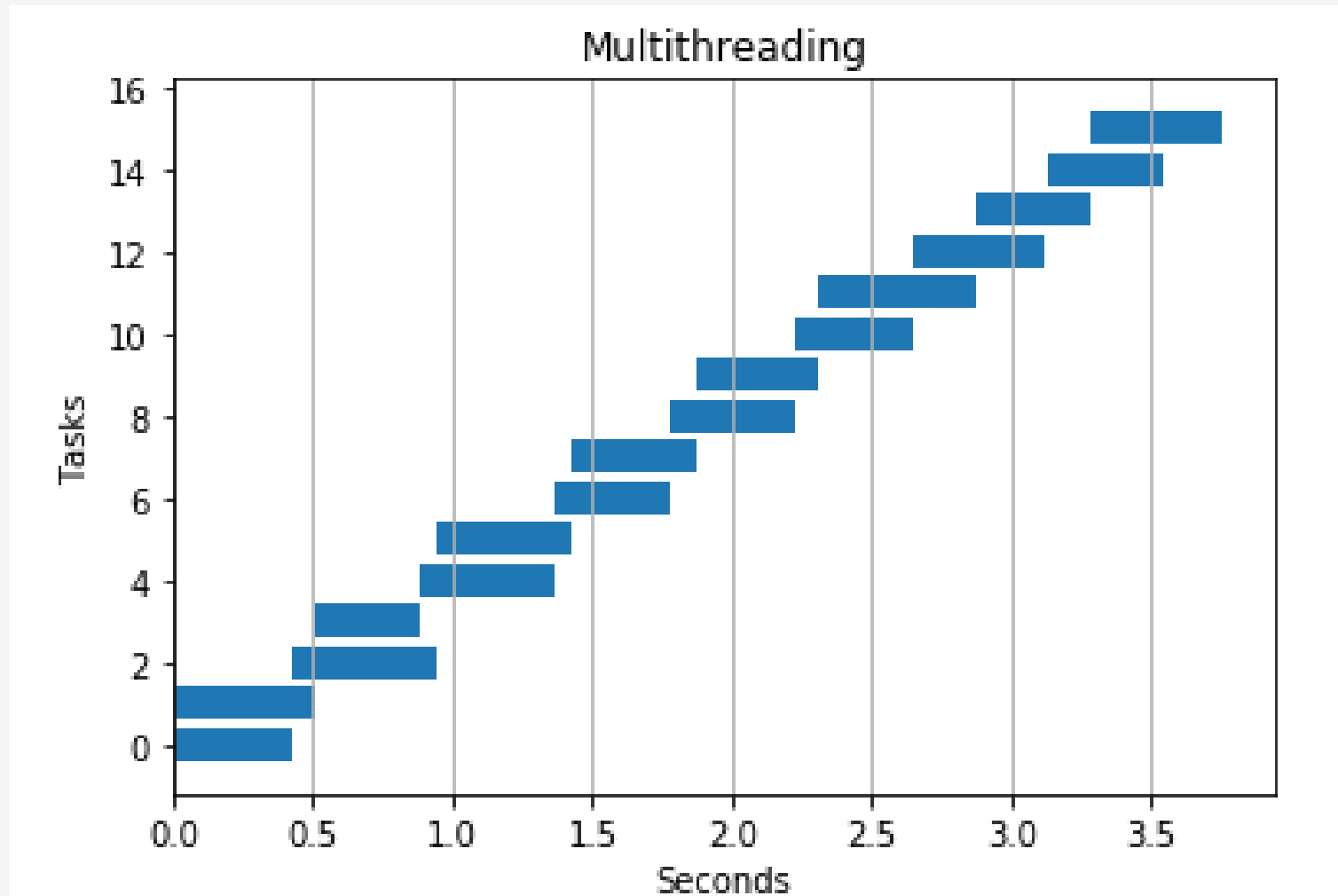


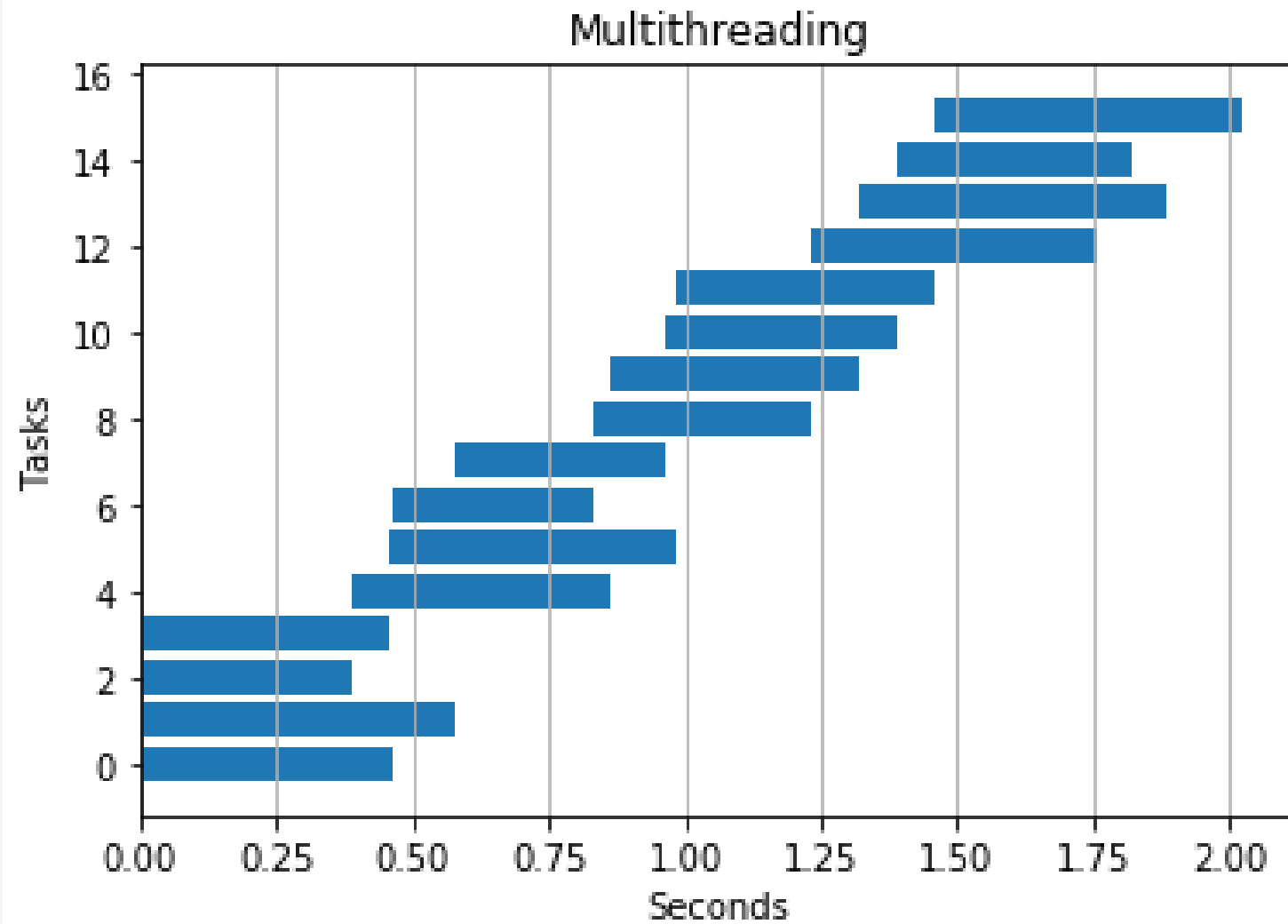
Total number of transistors 1.86B

Die size dimensions 15.0 mm x 17.1 mm [257 mm²]











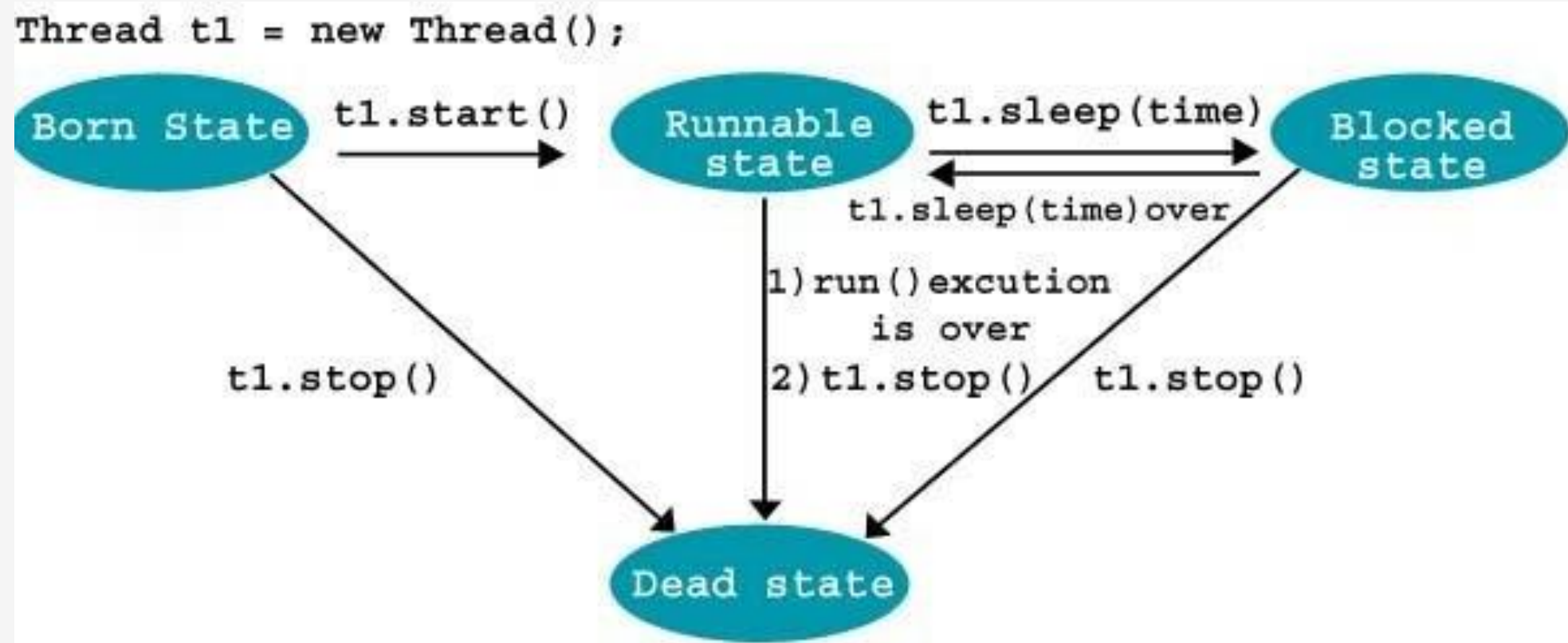
```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```



```
public static void main(String[] args) {  
    Thread t = new Thread();  
    t.start();  
    System.out.println("Hello World!");  
}
```

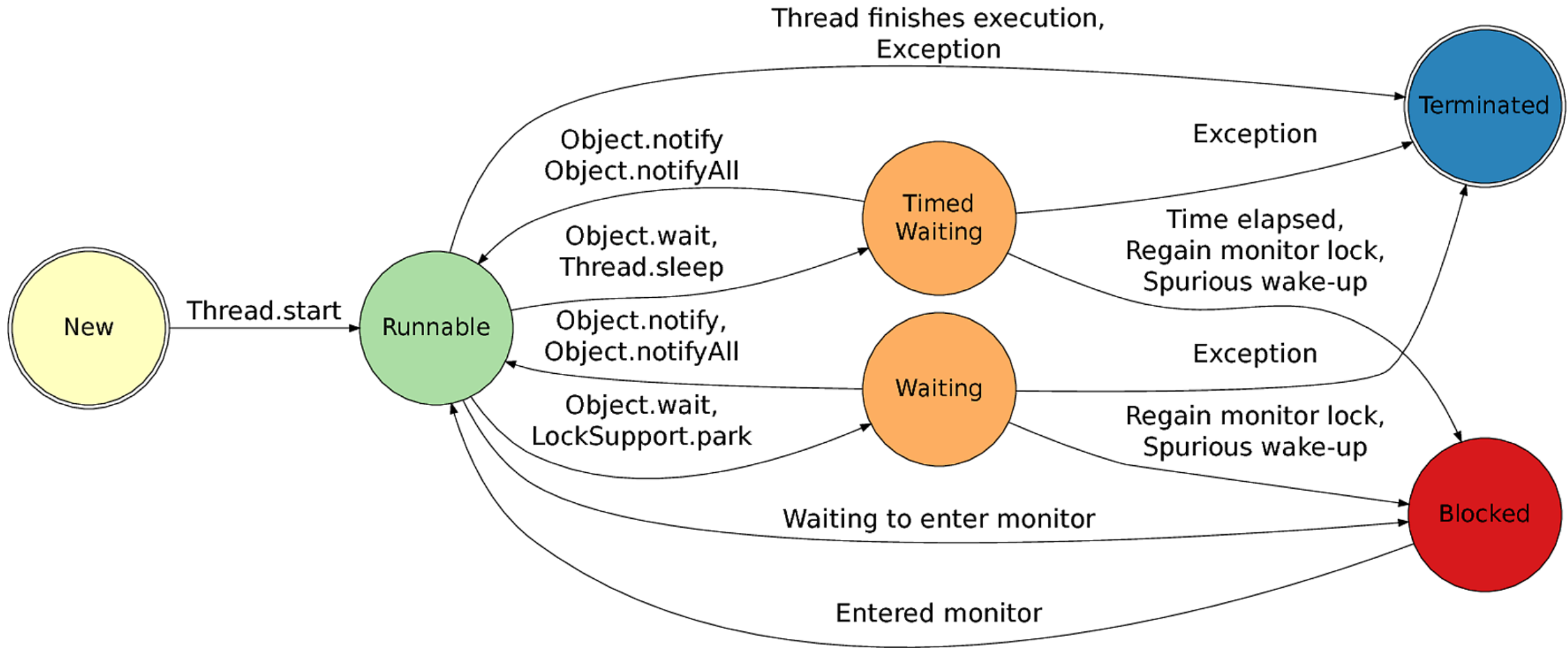



Cykl życia wątku w Javie





Cykl życia wątku w Javie





Interfejs Runnable i metoda .run()

- Interfejs **Runnable**, ma jedną metodę — **.run()**, która nic nie zwraca i nic nie przyjmuje.
- To centralny punkt naszego wątku - wątek uruchomi tę metodę i będzie działał, dopóki nie skończy się ona wykonywać. Po wykonaniu ostatniej linijki jej kodu – wątek kończy swoje działanie (umiera)
- Wewnątrz metody **.run()** zapisujemy cały kod, który ma zostać wykonany przez nasz wątek

```
public interface Runnable {  
    public abstract void run();  
}
```



Hello world + wątek



```
private static class MyRunnable implements Runnable {  
    @Override  
    public void run() {  
        System.out.println("Hello I'm thread!");  
    }  
}
```



```
public static void main(String[] args) {  
    Thread t = new Thread(new MyRunnable());  
    t.start();  
    System.out.println("Hello World!");  
}
```


Hello world + wątek



```
private static class MyThread extends Thread {  
    @Override  
    public void run() {  
        System.out.println("Hello I'm thread!");  
    }  
}
```

```
public static void main(String[] args) {  
    Thread t = new MyThread();  
    t.start();  
    System.out.println("Hello World!");  
}
```



Zatrzymanie wątku (1)



```
public static void main(String[] args) throws Exception {  
    Thread t = new Thread(new Runnable() {  
        @Override  
        public void run() {  
            int i = 0;  
            while (true)  
                i++;  
        }  
    });  
    t.start();  
    Thread.sleep(1000);  
    t.stop();  
}
```



Zatrzymanie wątku (2)

```
private static boolean stopRequested = false;

public static void main(String[] args) throws Exception {
    Thread t = new Thread(new Runnable() {
        @Override
        public void run() {
            int i = 0;
            while (!stopRequested)
                i++;
        }
    });

    t.start();
    Thread.sleep(1000);
    stopRequested = true;
}
```

volatile

vs

synchronized



- Wskazanie JVM, że wartość zmiennej będzie modyfikowana przez różne wątki (więc **nie będzie cache'owana** przez poszczególne wątki, które zamiast tego skorzystają z odwołania bezpośrednio do pamięci)
- Nigdy nie spowoduje zakleszczenia
- Używane do prostych rzeczy

- Wskazanie JVM, że dany blok kodu ma być **wykonywany przez tylko jeden wątek w tym samym czasie** (tj. jest sekcją krytyczną)
- Nieprawidłowo użyte – może spowodować zakleszczenie
- Używane do prostych i bardziej skomplikowanych rzeczy

Zatrzymanie wątku (3)



```
private static volatile boolean stopRequested = false;

public static void main(String[] args) throws Exception {
    Thread t = new Thread(new Runnable() {
        @Override
        public void run() {
            int i = 0;
            while (!stopRequested)
                i++;
        }
    });

    t.start();
    Thread.sleep(1000);
    stopRequested = true;
}
```



Zatrzymanie wątku (4)



```
private static boolean stopRequested = false;

private static synchronized void requestStop() {
    stopRequested = true;
}

private static synchronized boolean isStopRequested() {
    return stopRequested;
}

public static void main(String[] args) throws Exception {
    Thread t = new Thread(new Runnable() {
        @Override
        public void run() {
            int i = 0;
            while (!isStopRequested())
                i++;
        }
    });
    t.start();
    Thread.sleep(1000);
    requestStop();
}
```



Zatrzymanie wątku (5)



```
public static void main(String[] args) throws Exception {  
    Thread t = new Thread(new Runnable() {  
        @Override  
        public void run() {  
            int i = 0;  
            while (!Thread.interrupted())  
                i++;  
        }  
    });  
    t.start();  
    Thread.sleep(1000);  
    t.interrupt();  
}
```





Zadanie 1: Jaś i Małgosia

- Przygotuj program, w którym będą żyły równocześnie dwa wątki – **Jaś i Małgosia**
- Wykonują oni różne rzeczy w domu, które zajmują różny czas (jedne krócej, inne dłużej)
- Wskazówka: wykonywanie czynności potraktuj bardzo „symbolicznie”:

```
System.out.println("Jaś zaczyna prysznic...");  
Thread.sleep(3000);  
System.out.println("Jaś skończył prysznic!");
```

- Zadanie dodatkowe: Jak oboje skończą realizować swój plan dnia, wyświetl tekst „Koniec dnia!”

- Plan dnia Jasia:
 - Przygotowanie i jedzenie śniadania (5 sekund)
 - Prysznic (3 sekundy)
 - Ubranie się (1 sekunda)
 - Wyjście na zakupy (15 sekund)
 - Granie na konsoli (5 sekund)
- Plan dnia Małgosi:
 - Poranne bieganie (6 sekund)
 - Prysznic (2 sekundy)
 - Jedzenie śniadania (1 sekunda)
 - Ubranie się (1 sekunda)
 - Spotkanie z koleżanką (25 sekund)