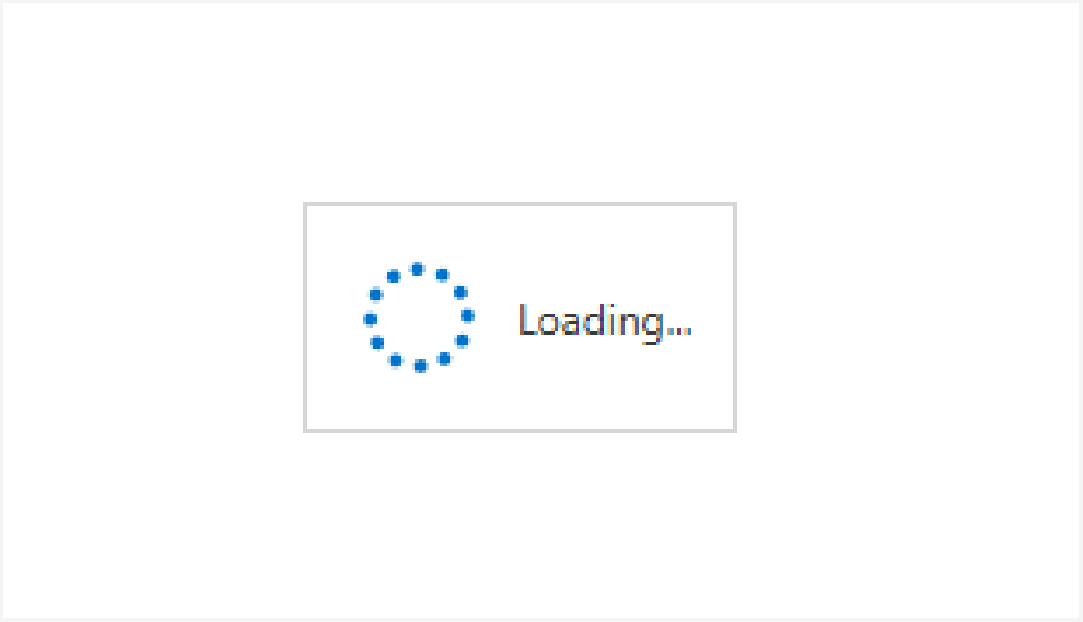
Java Programowanie II

wielowątkowość

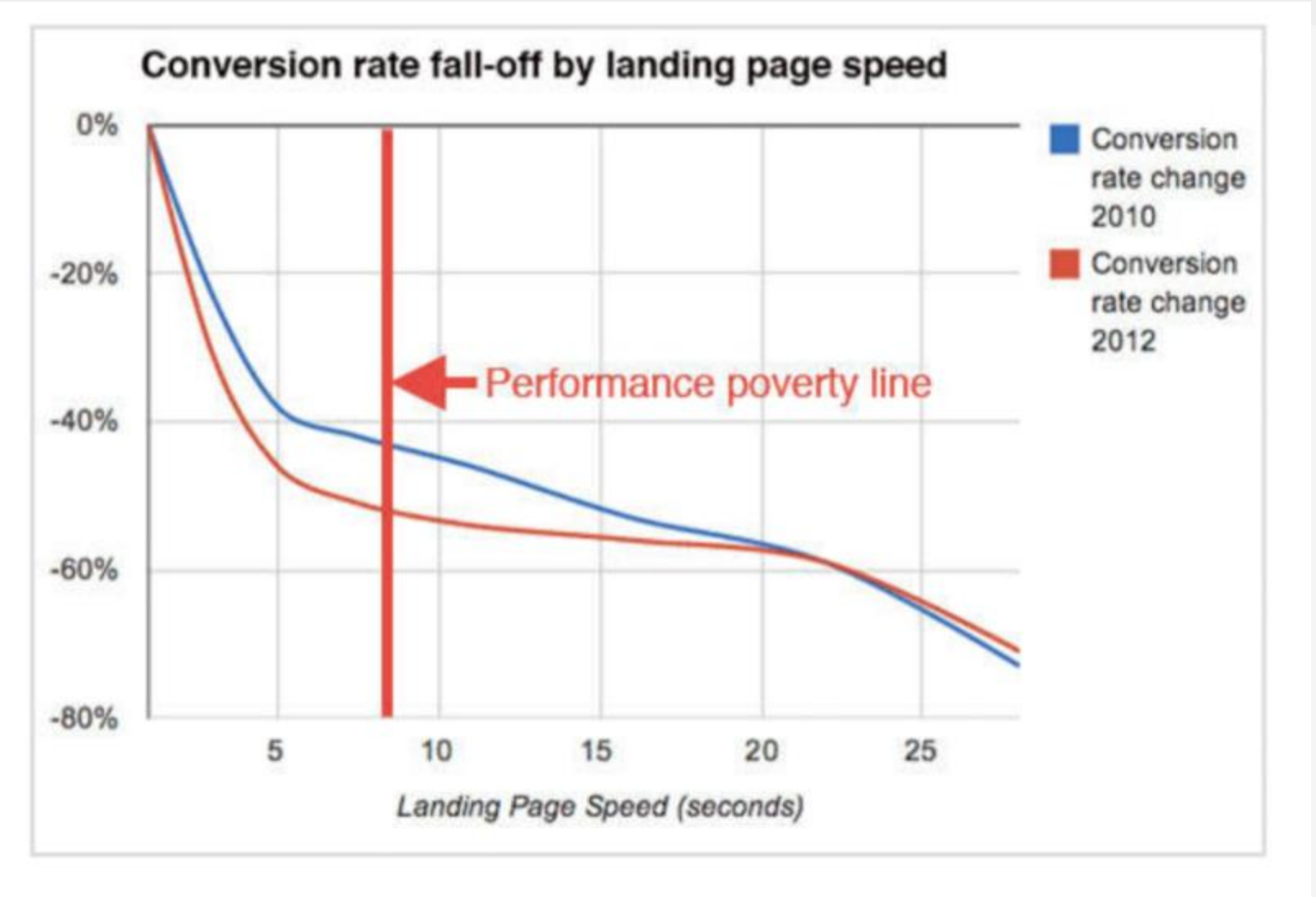
Autor: Grzegorz Witczak


LOADING...

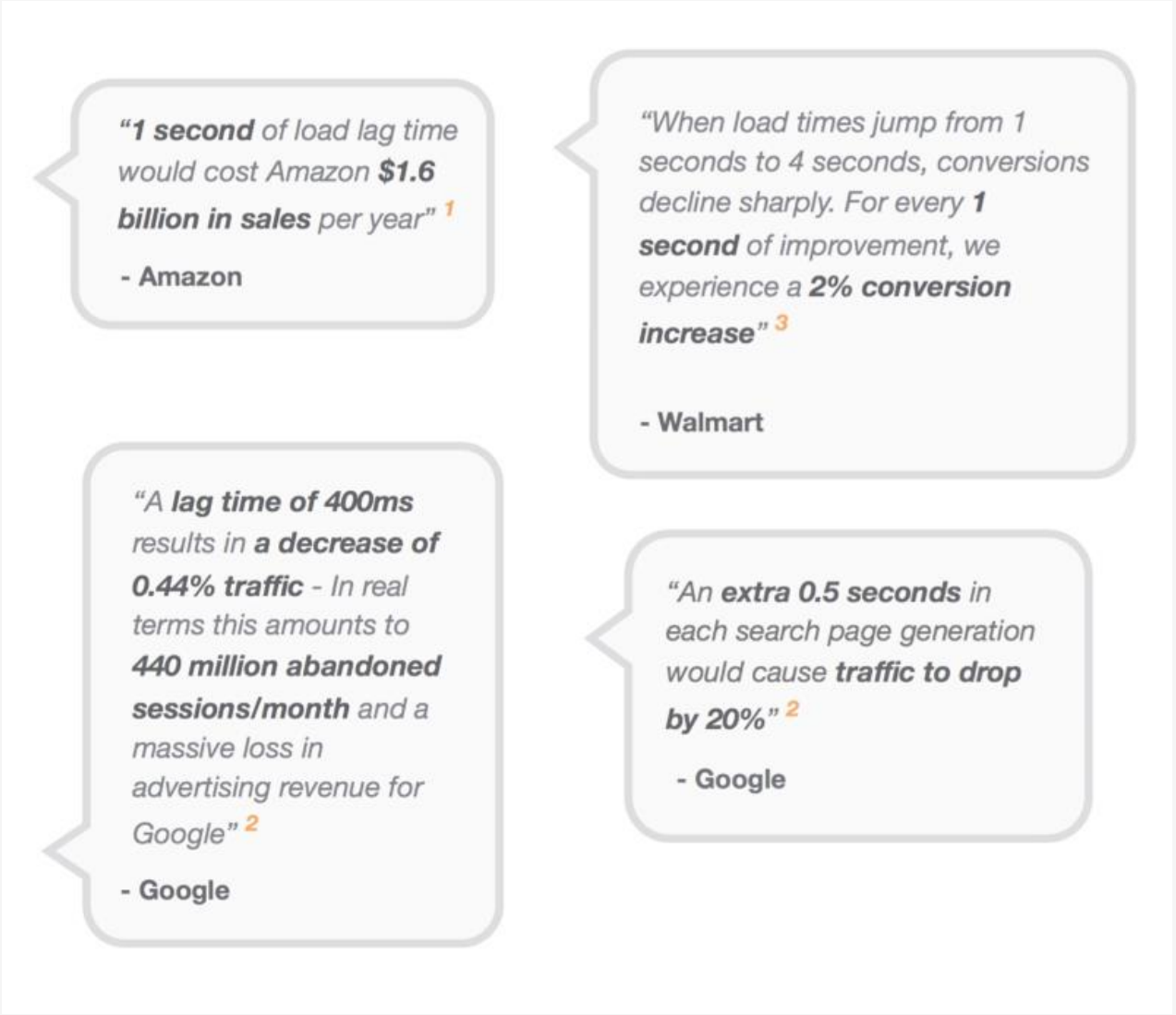

DOWNLOADING...
75 %


Loading...


E 87005581 G
WASHINGTON, D.C.
壹佰圓
WX51753386
100
ECB EZB EKT EKP 2002
200
Bank of England
I PROMISE TO PAY THE BEARER ON DEMAND THE SUM OF
FIVE
Pounds
JD70 915
50
£50
FIFTY
Pounds
THE UNITED STATES OF AMERICA
THIS NOTE IS LEGAL TENDER
FOR ALL DEBTS, PUBLIC AND PRIVATE
I 3841926
中国人民银行
伍圓
GG 96859801
SINGAPURA 新加坡 SINGAPORE
50
20
100

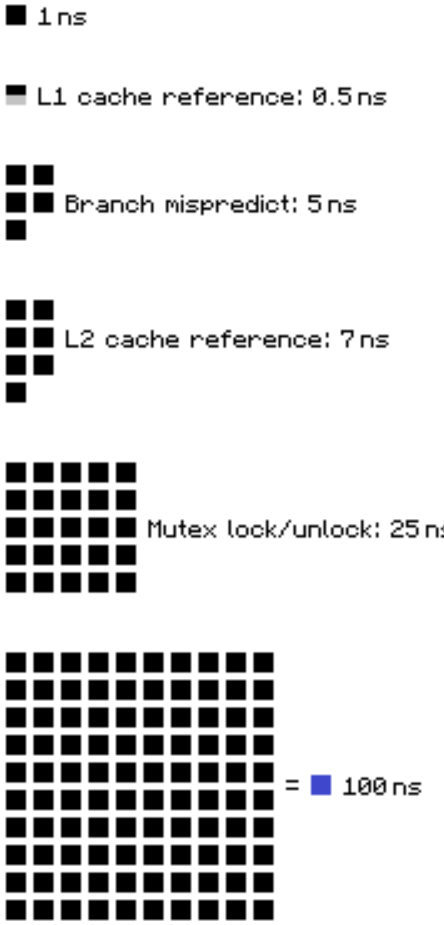


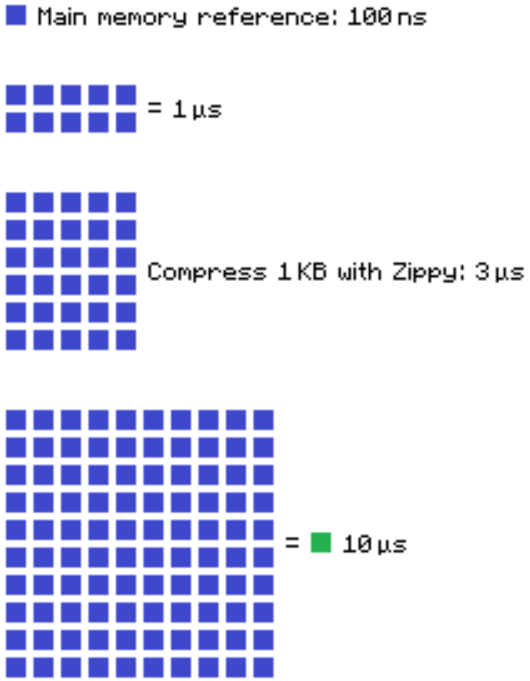
https://medium.com/@vikigreen/impact-of-slow-page-load-time-on-website-performance-40d5c9ce568a

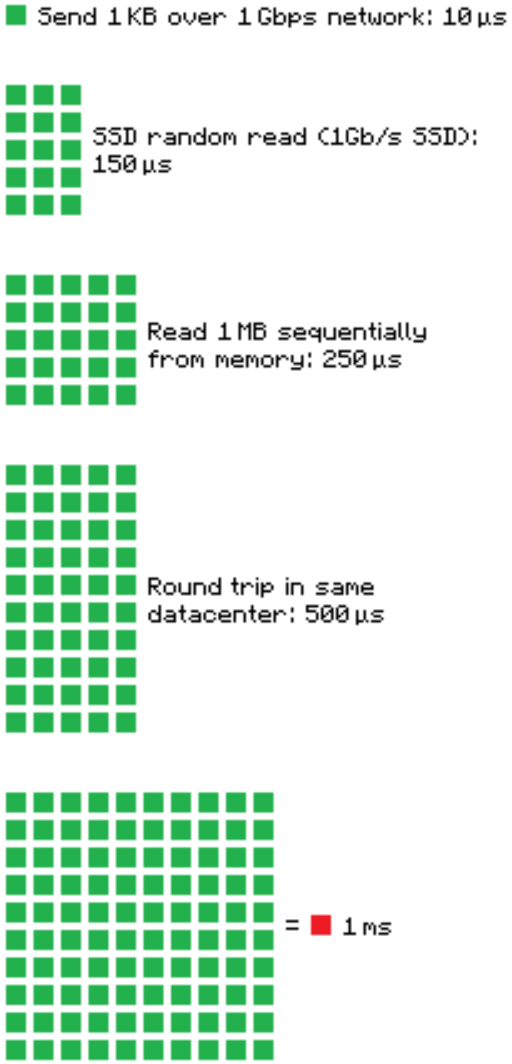


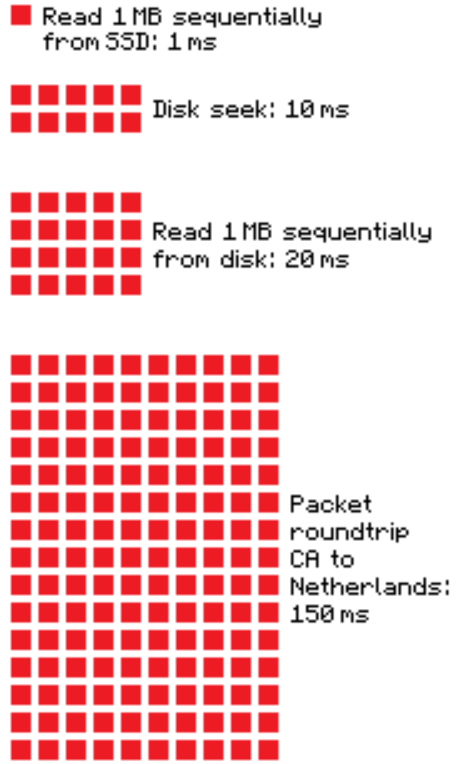
https://medium.com/@vikigreen/impact-of-slow-page-load-time-on-website-performance-40d5c9ce568a

Latency Numbers Every Programmer Should Know



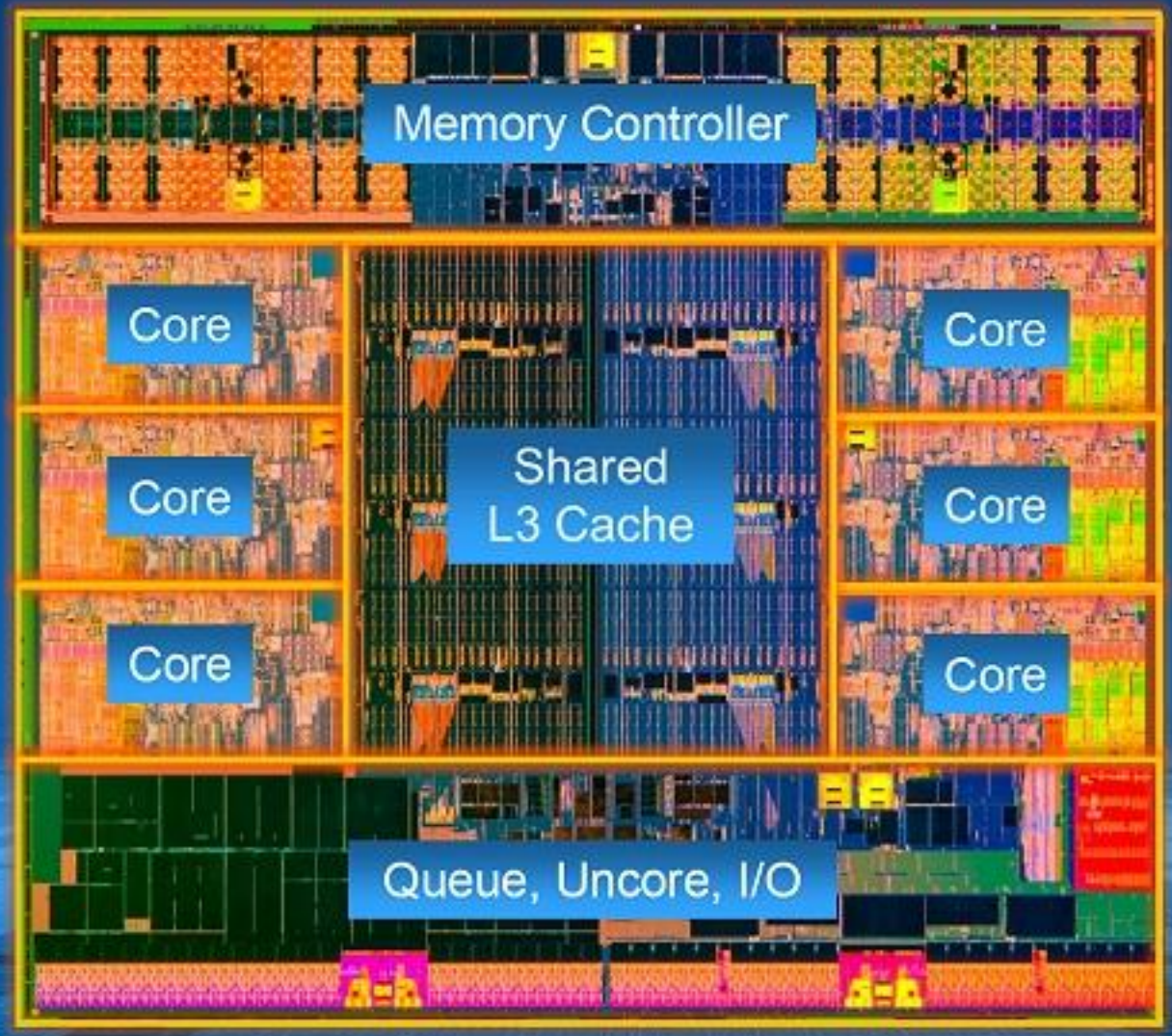






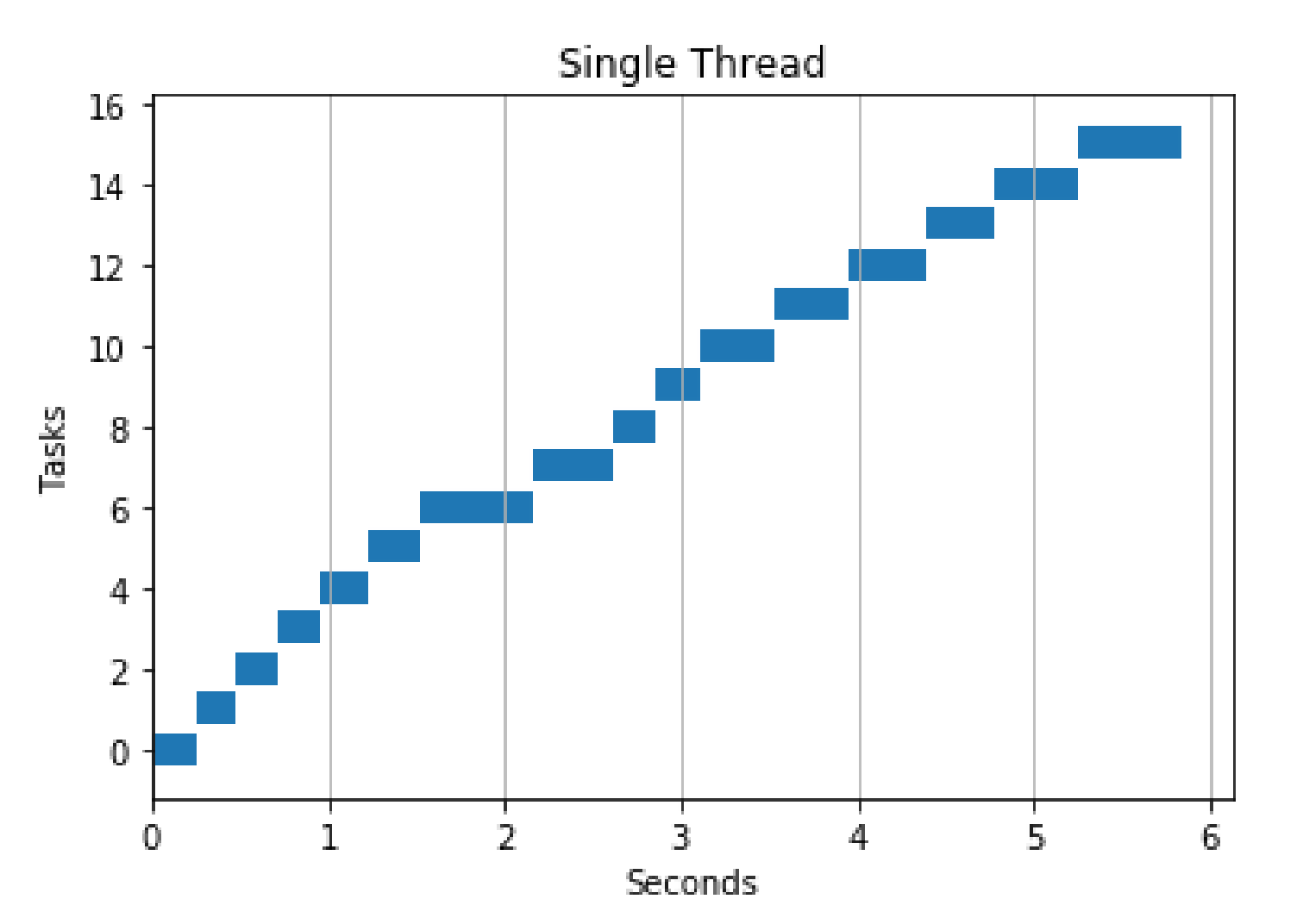
Source: https://gist.github.com/2841832

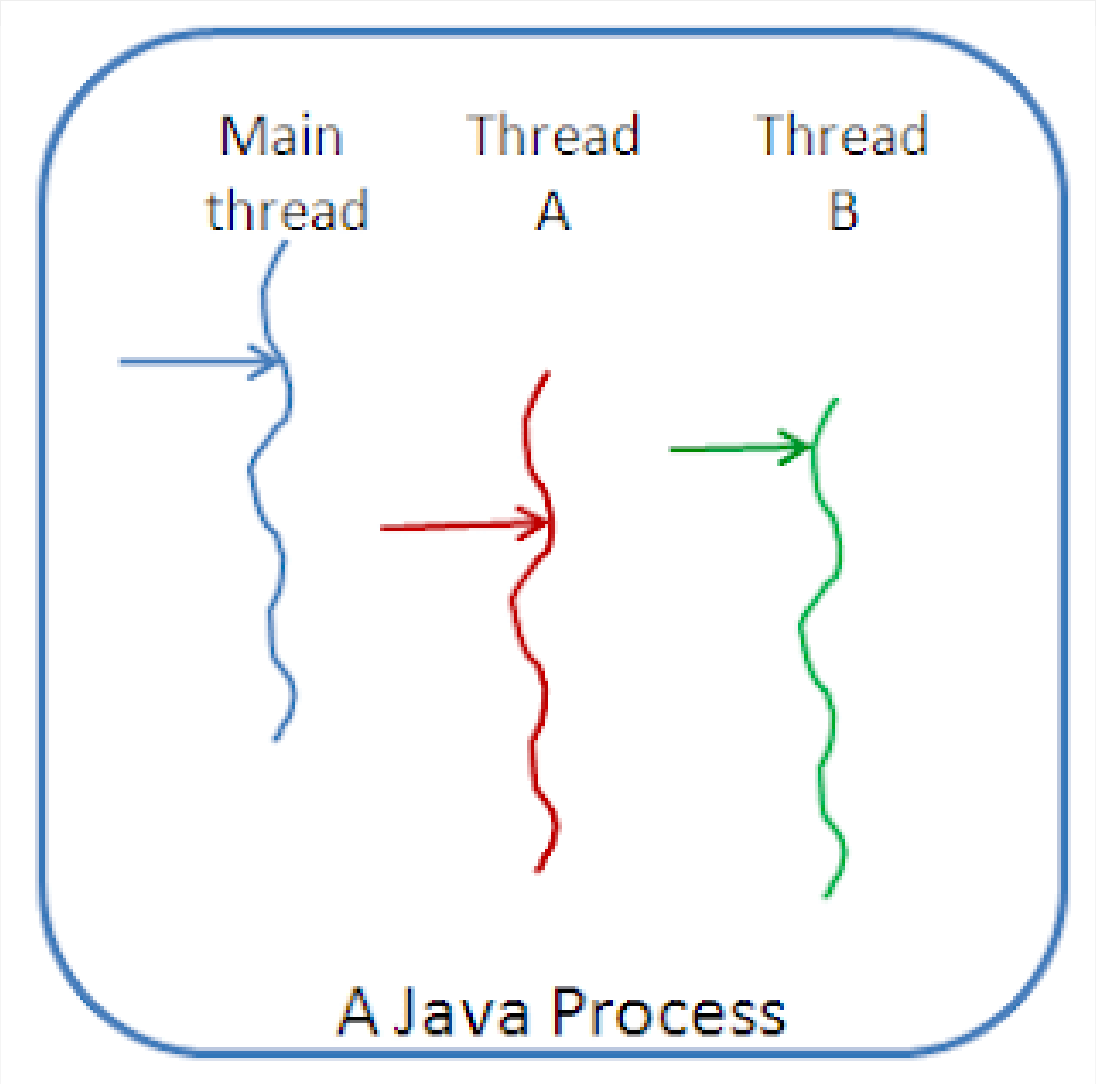
Intel® Core™ i7-4960X Processor Die Detail

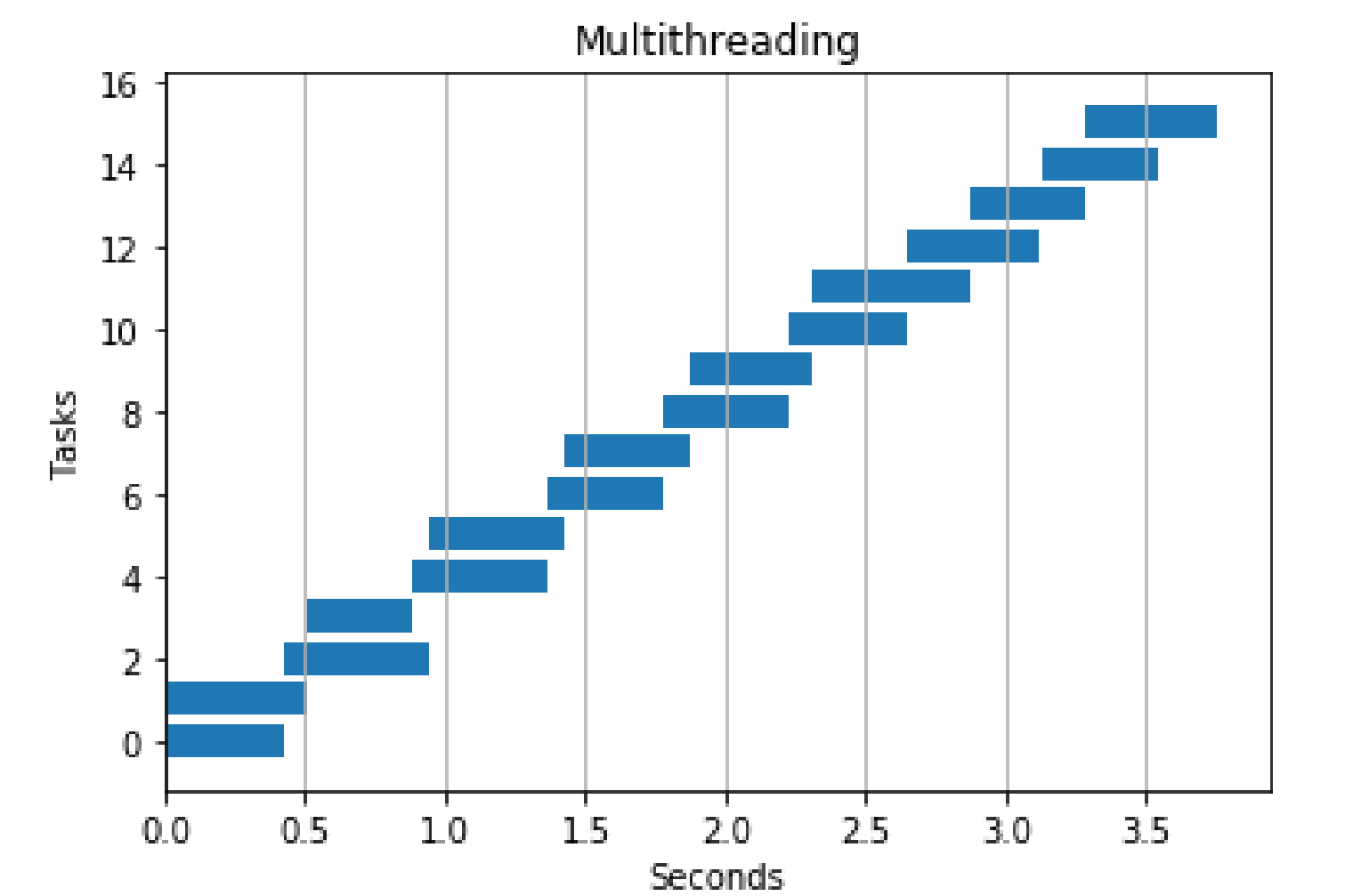


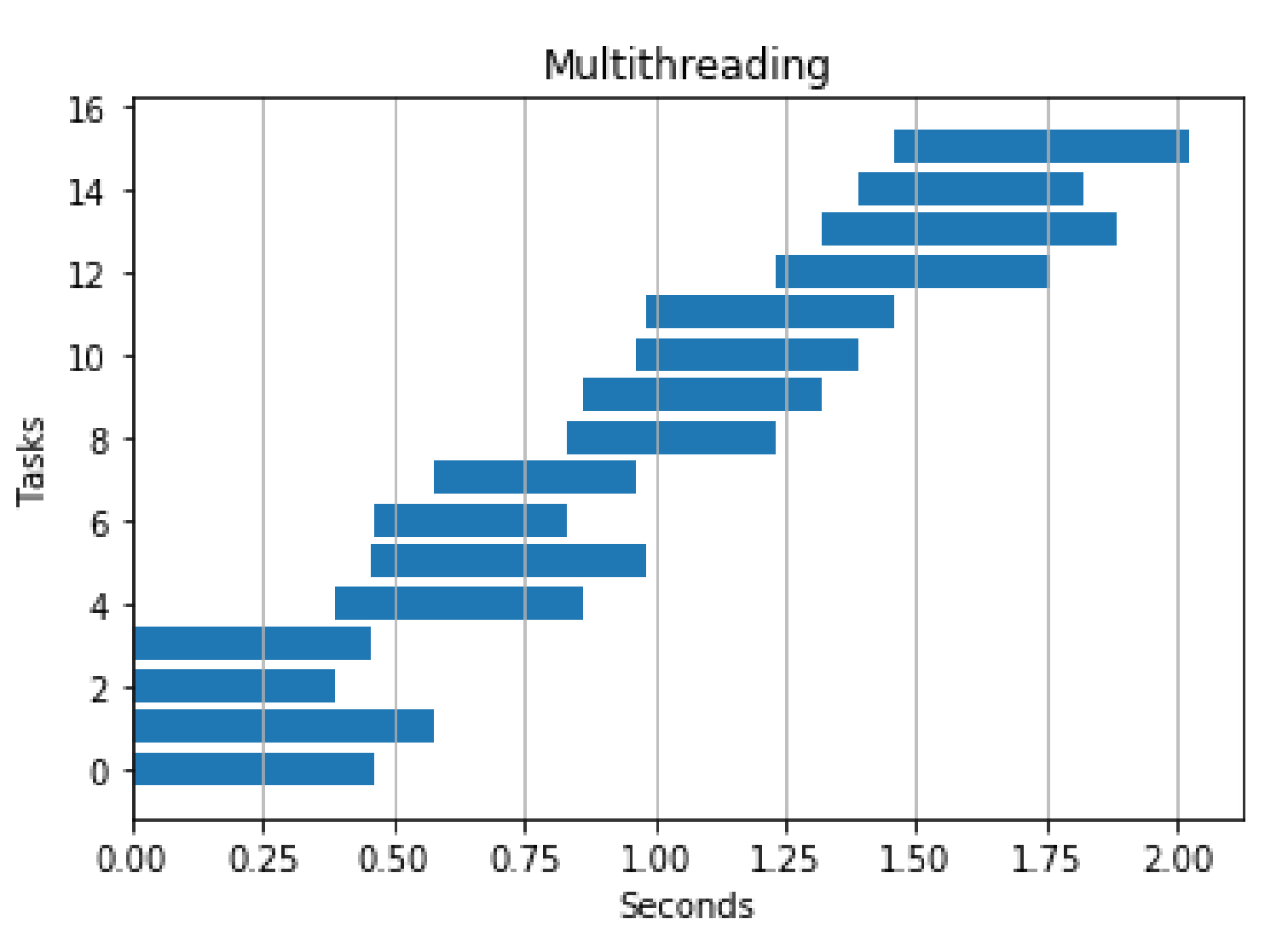
Total number of transistors 1.86B

Die size dimensions 15.0 mm x 17.1 mm [257 mm^2]


Single Thread
Tasks
Seconds
0
2
4
6
8
10
12
14
16
0
1
2
3
4
5
6


Main thread
Thread A
Thread B
A Java Process


Multithreading
Tasks
Seconds
0
2
4
6
8
10
12
14
16
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Multithreading
Tasks
16
14
12
10
8
6
4
2
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Seconds

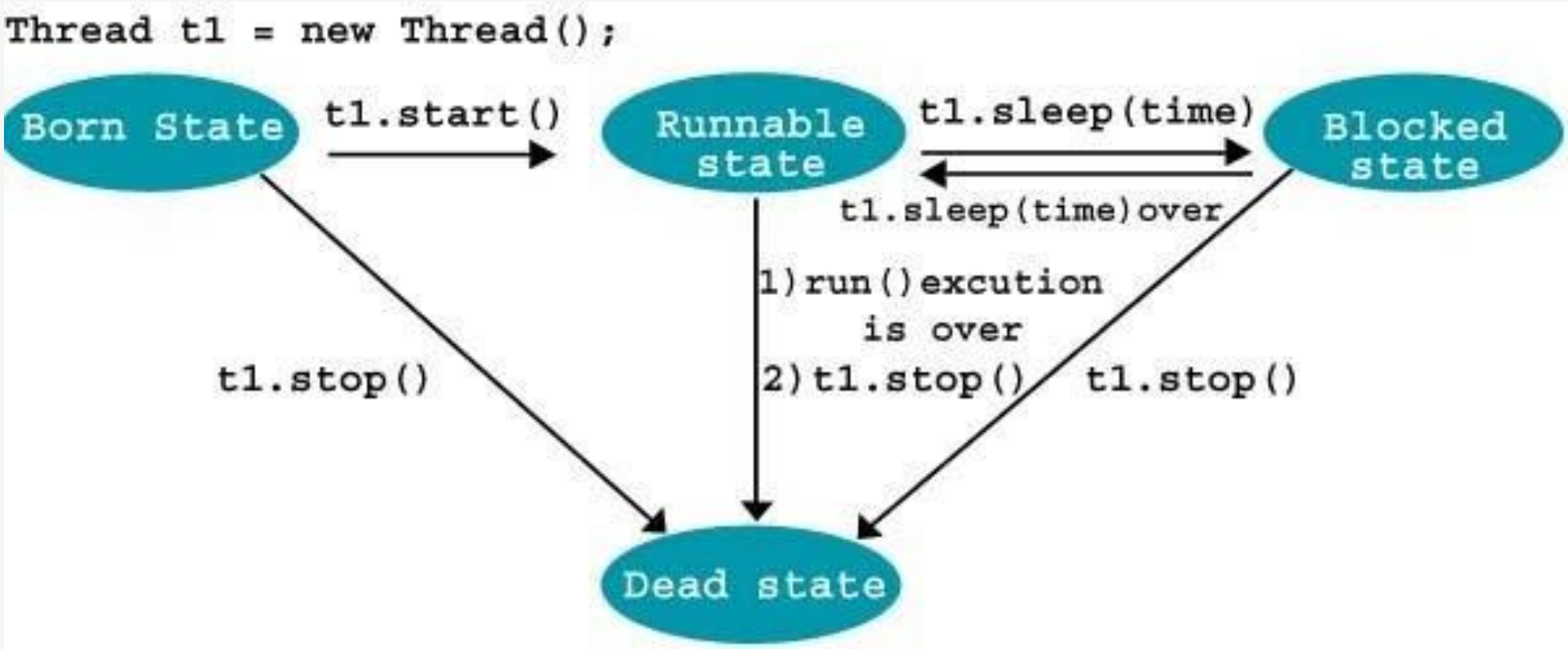
Hello world

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

Hello world + wątek

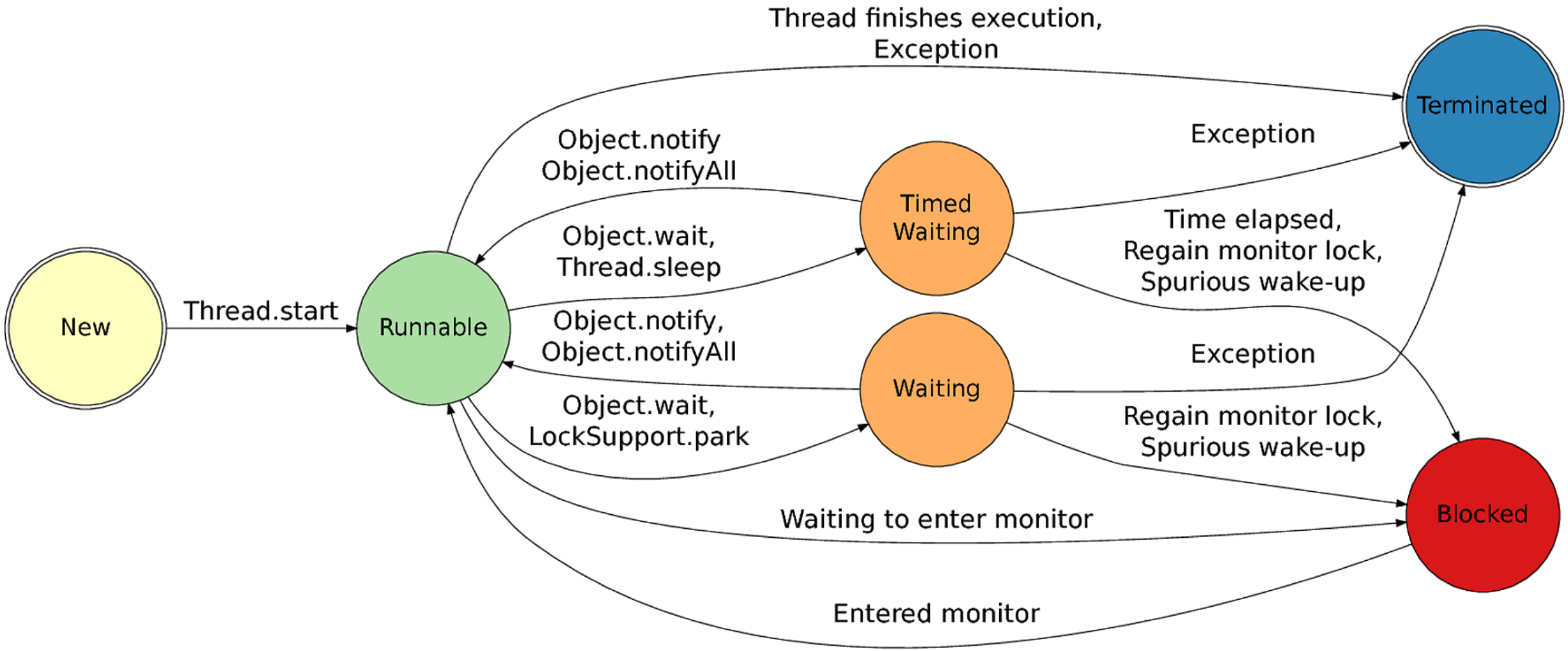
```
public static void main(String[] args) {
    Thread t = new Thread();
    t.start();
    System.out.println("Hello World!");
}
```

Cykl życia wątku w Javie



http://way2java.com/wp-content/uploads/2011/01/life-cycle.jpe

Cykl życia wątku w Javie



https://stackoverflow.com/a/11302334/1061543

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() zapisujmy 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)

Sposoby czekania na wątki: wait & notify

Sposoby czekania na wątki: wait & notify

- `monitor.wait()`

  Wątek przechodzi w stan **uśpienia** i oczekuje wybudzenia na monitorze

- `monitor.notify()`

  Inny wątek woła metodę notify na monitorze i **wybudza** jeden z oczekujących watków

- `monitor.notifyAll()`

  Inny wątek woła metodę notifyAll na monitorze i wybudza wszystkie z oczekujących watków

Sposoby czekania na wątki: wait & notify (2)

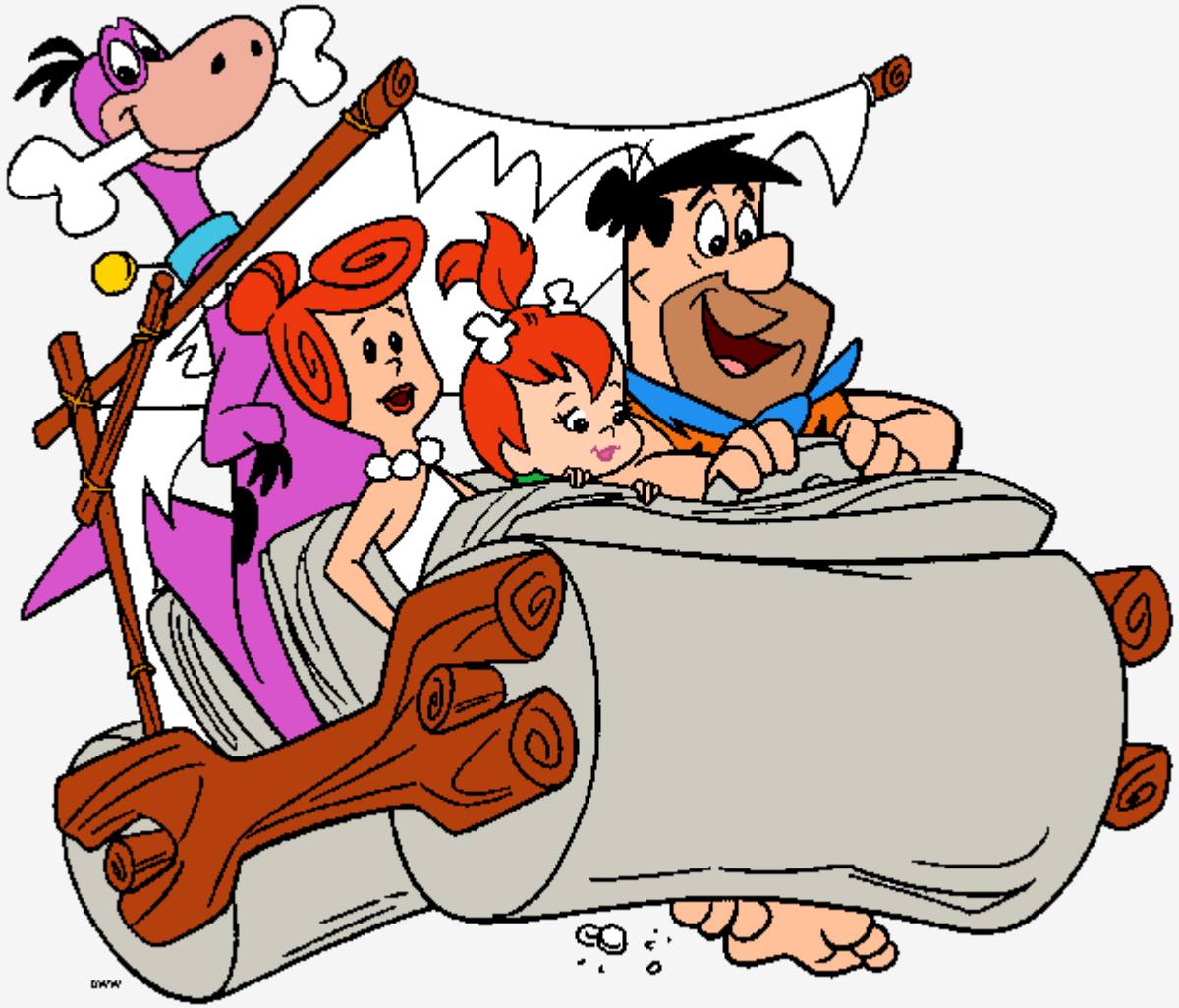
- Wait&notify są stosunkowo niskopoziomowe, a także (na dłuższą metę) **nieprzyjemne** i **trudne**. W zdecydowanej większości przypadków – **da się użyć czegoś fajniejszego!**
- Ale jeżeli już uparliśmy się na wait&notify…
  - Zawsze zamykać wait w pętli sprawdzającej warunek – i usypiać wątek dalej, jeżeli warunek nie jest spełniony
  - Nie zakładać, że wątki uśpią się w oczekiwanej przez nas kolejności – żeby wybudzić jeden konkretny, lepiej wybudzić wszystkie (i pozostałe zaraz się uśpią przez pętlę sprawdzającą warunek)

wait & notify - demo


JUST WHEN YOU THOUGHT
IT COULDN'T GET WORSE


THERE'S ALWAYS AN IDIOT
TO PROVE YOU WRONG

Sposoby czekania na wątki: no-wait



Sposoby czekania na wątki: no-wait

- Najgorszym z możliwych sposobów jest tzw. „aktywne czekanie”, które **nie usypia wątku**. Robi się to za pomocą pętli, która składa się tylko ze sprawdzenia warunku:

```
private static volatile boolean malgosiaGotowa = false;

while (malgosiaGotowa == false);
```

- To tragiczny sposób, bo pochłania (zupełnie niepotrzebnie) mnóstwo zasobów komputera – pytając dziesiątki tysięcy razy na sekundę, czy zmienna *malgosiaGotowa* ma wartość **true**.

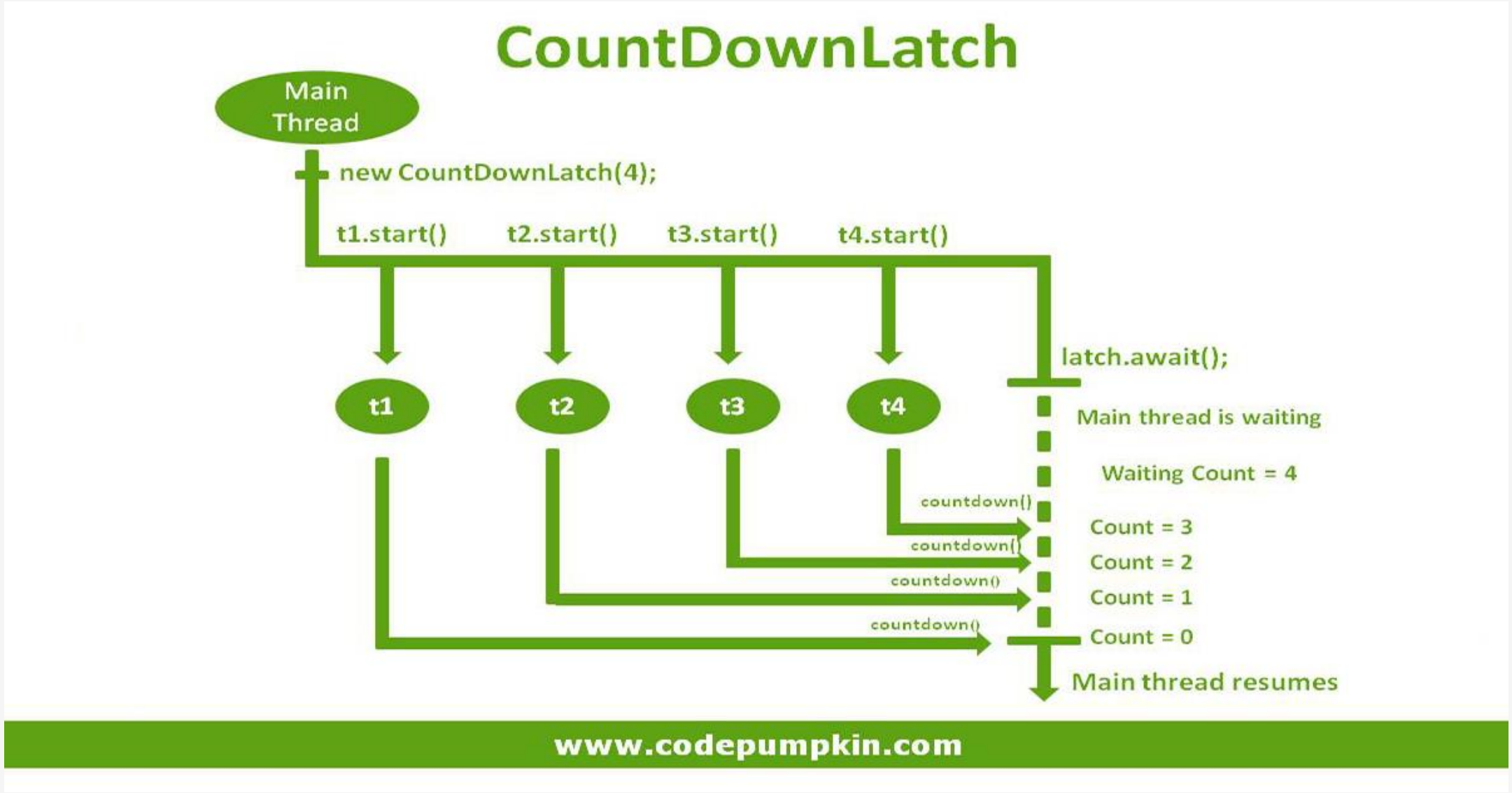

There's a way to do it better - find it.
Thomas A. Edison
quotefancy

Sposoby czekania na wątki: CountDownLatch

Sposoby czekania na wątki: CountDownLatch

- “Zatrzask odliczający”
- Inicjowany w konstruktorze liczbą, zmiejszaną o jeden z każdym wywołaniem `.countdown()`
- W momencie kiedy licznik dojdzie do zera, zwalnia wszystkie wątki oczekujące na zakończenie odliczania (tj. wątki, które wywołały `.await()`)

Sposoby czekania na wątki: CountDownLatch



https://codepumpkin.com/countdownlatch-java-concurrency/

CountDownLatch - demo