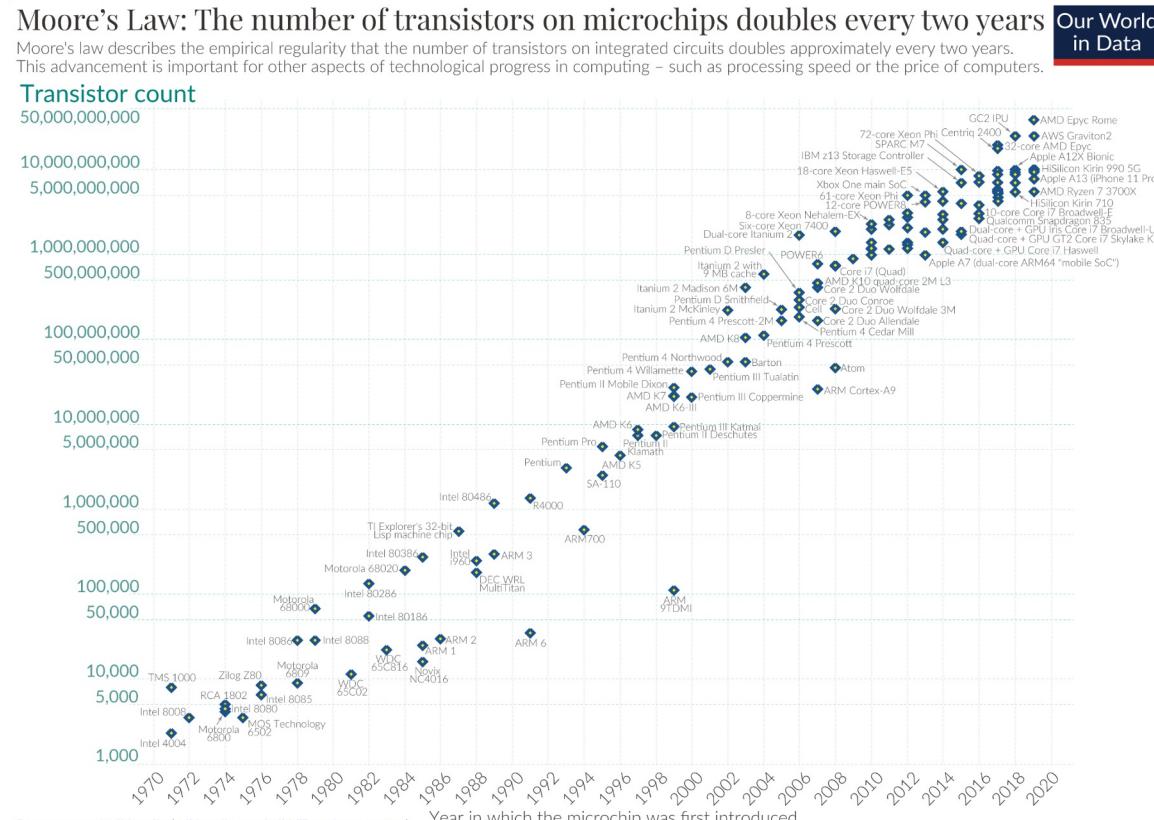


PROGRAMACIÓN EN PARALELO Y CONCURRENCIA

Computación en paralelo – por qué?

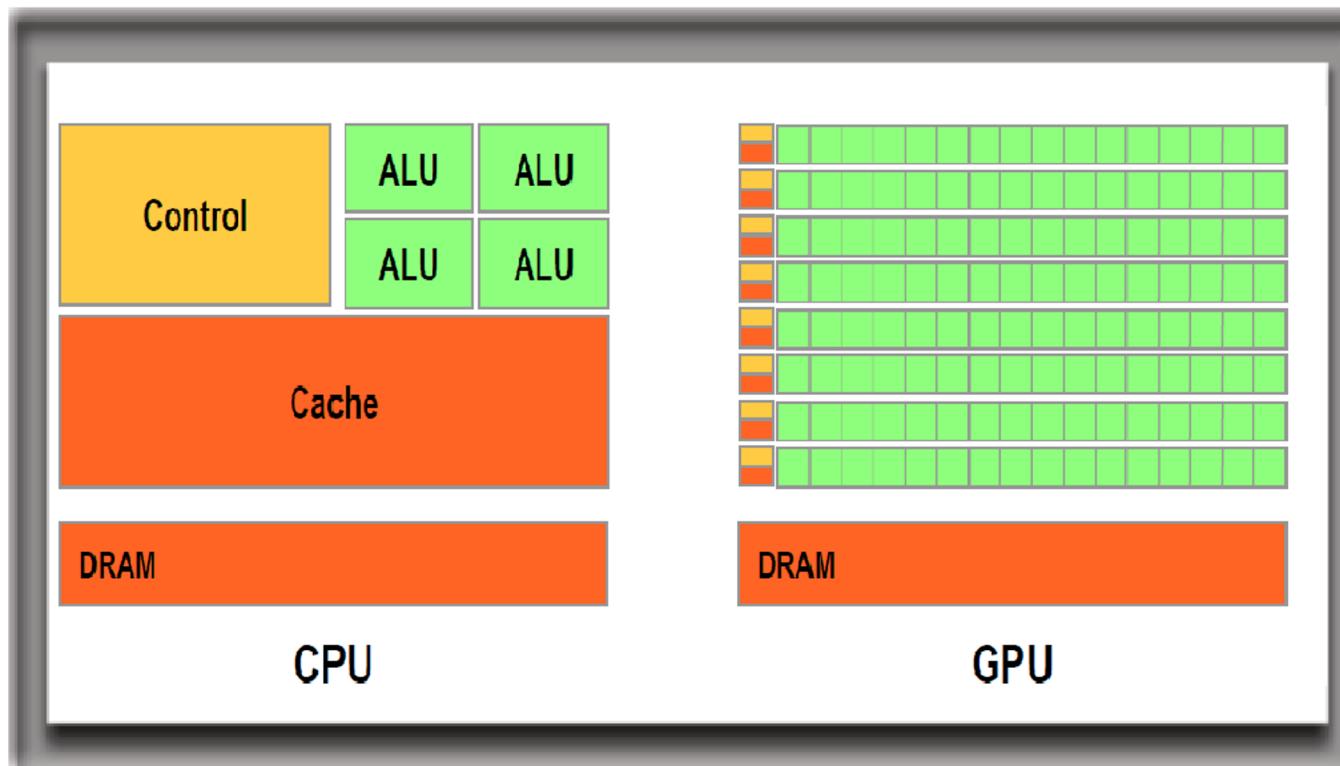
- Capacidad de cómputo para el futuro?



Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

Por qué?

- Paralelismo



Por qué?

- Paralelismo



Por qué?

- Paralelismo



Por qué?

- Paralelismo



Paralelismo - GPU

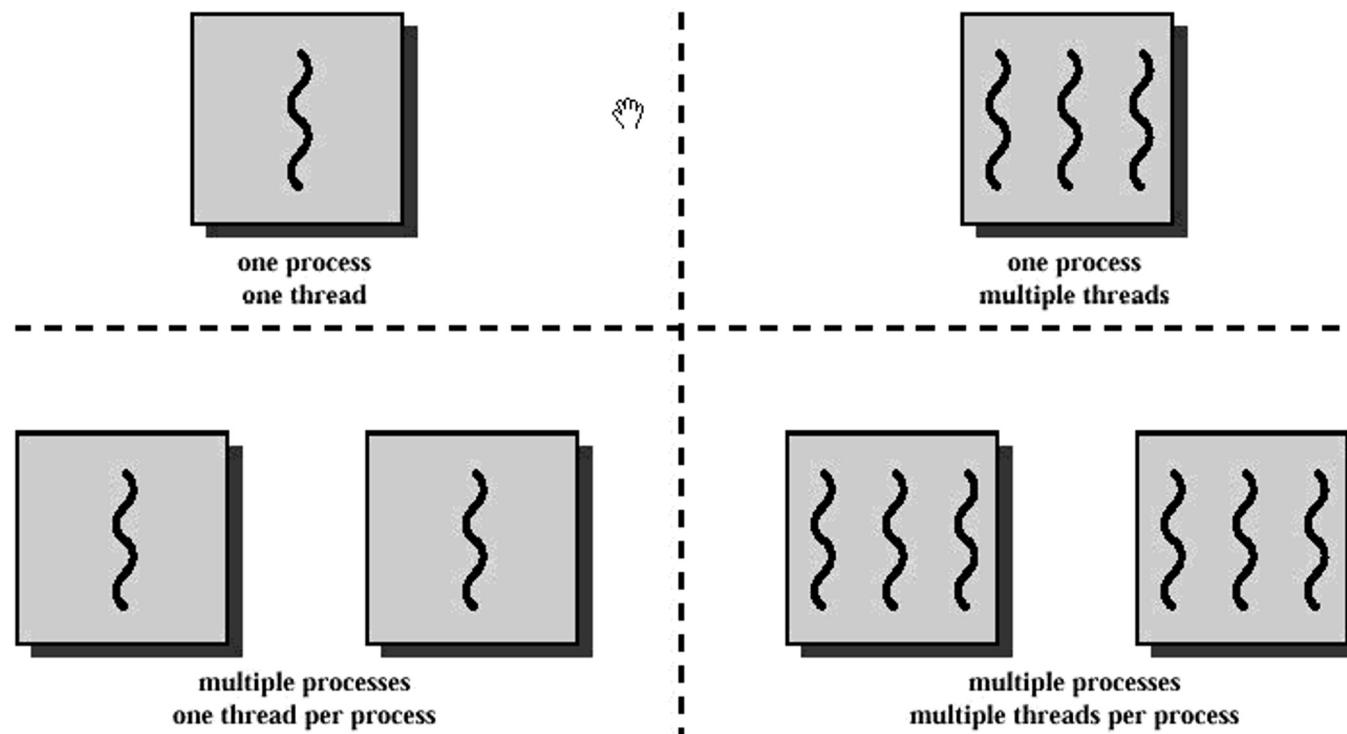
The screenshot shows a dark-themed product page for the GeForce GTX 1080 Ti. At the top, there's a navigation bar with links for PRODUCTS, GEFORCE EXPERIENCE, DRIVERS, GAMES, NEWS, COMMUNITY, SUPPORT, and CHAT. A large green "X" icon is located in the top right corner. The main title "GEFORCE GTX 1080 Ti" is displayed prominently in green. Below it, the heading "GPU Engine Specs:" is followed by a table of specifications. The table has two columns: the left column lists the specification names, and the right column lists their values. The same layout is used for the "Memory Specs:" section below.

NVIDIA CUDA® Cores	3584
Boost Clock (MHz)	1582

Memory Speed	11 Gbps
Standard Memory Config	11 GB GDDR5X
Memory Interface Width	352-bit
Memory Bandwidth (GB/sec)	484

Procesamiento en paralelo con 1 CPU: Procesos vs Hilos

- Comparación
-



Procesos vs Hilos

- Comunicación:
 - Procesos: memoria compartida.
 - Hilos: variables compartidas.

Estados de los hilos

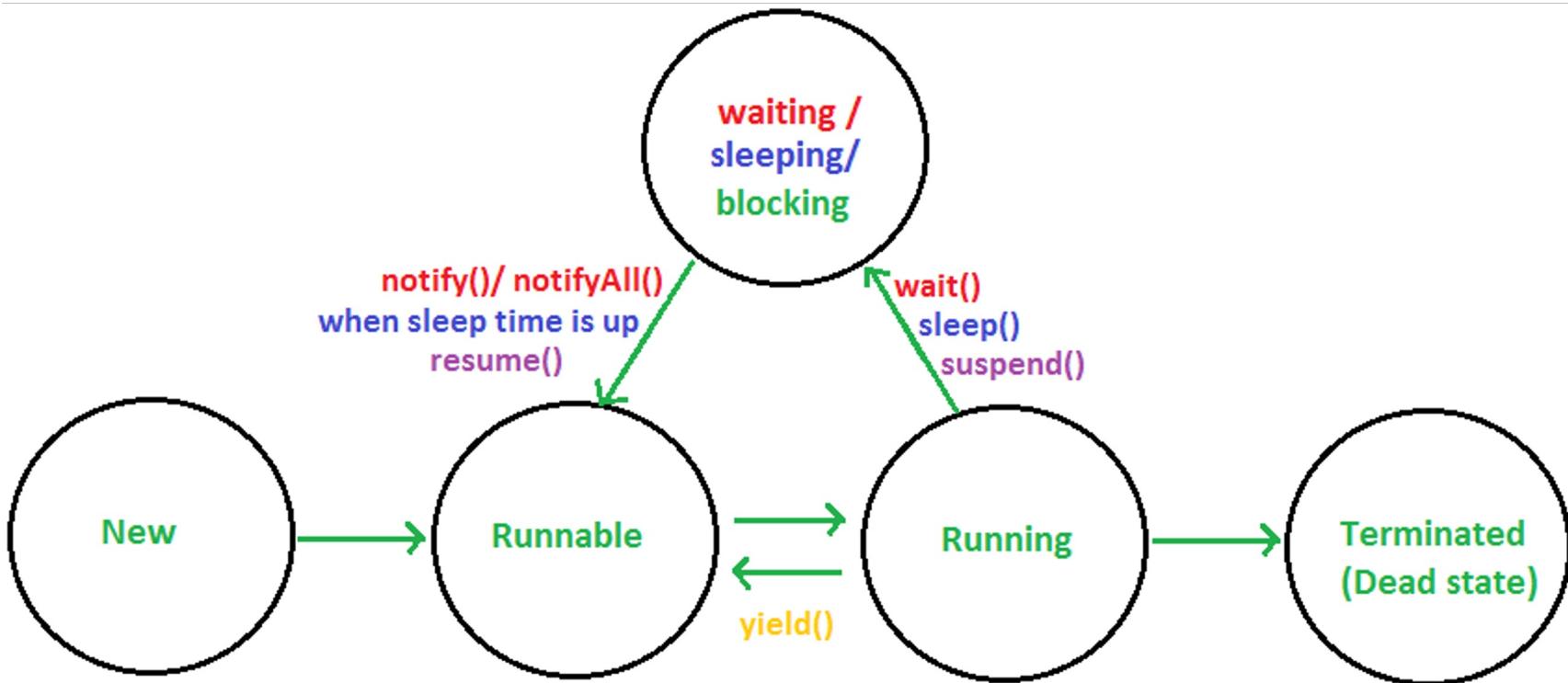


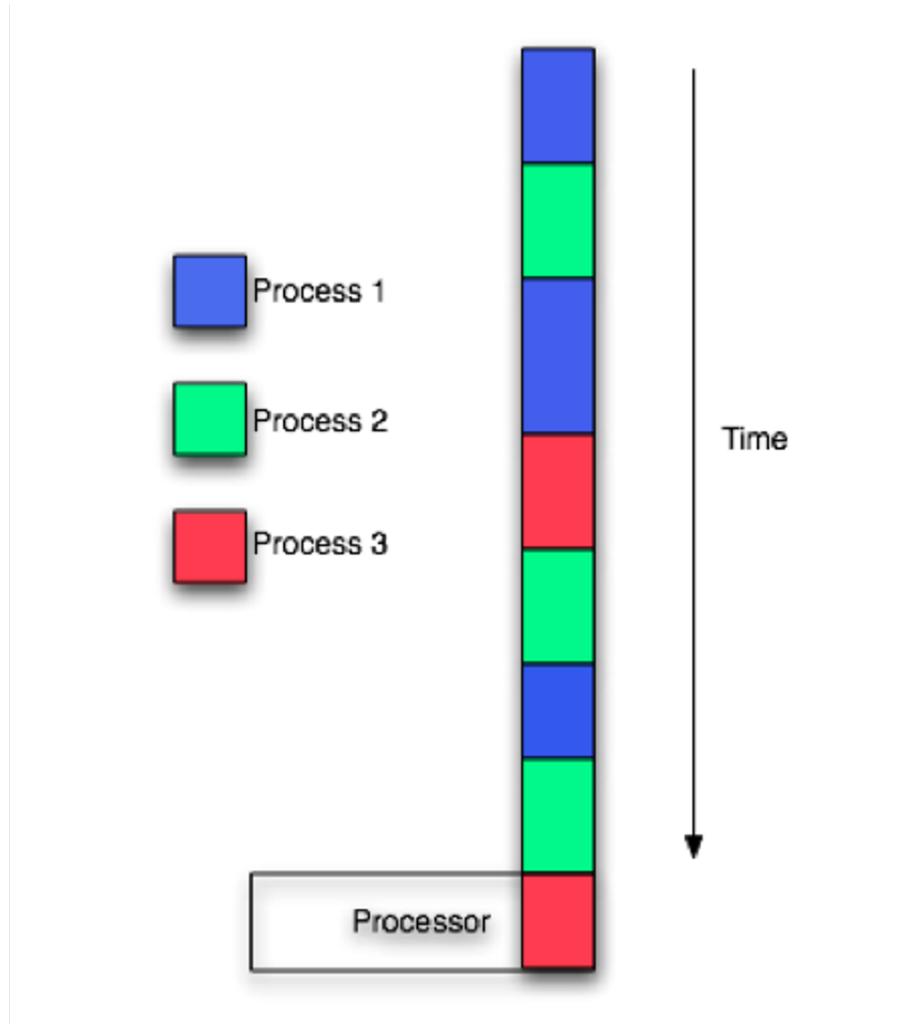
Fig. THREAD STATES

Paralelismo vs Concurrencia

Sun's *Multithreaded Programming Guide*:

- Concurrency: A condition that exists when at least two threads are making progress. A more generalized form of parallelism that can include time-slicing as a form of virtual parallelism.
- Parallelism: A condition that arises when at least two threads are executing simultaneously.

Concurrencia – ‘Interleaving’

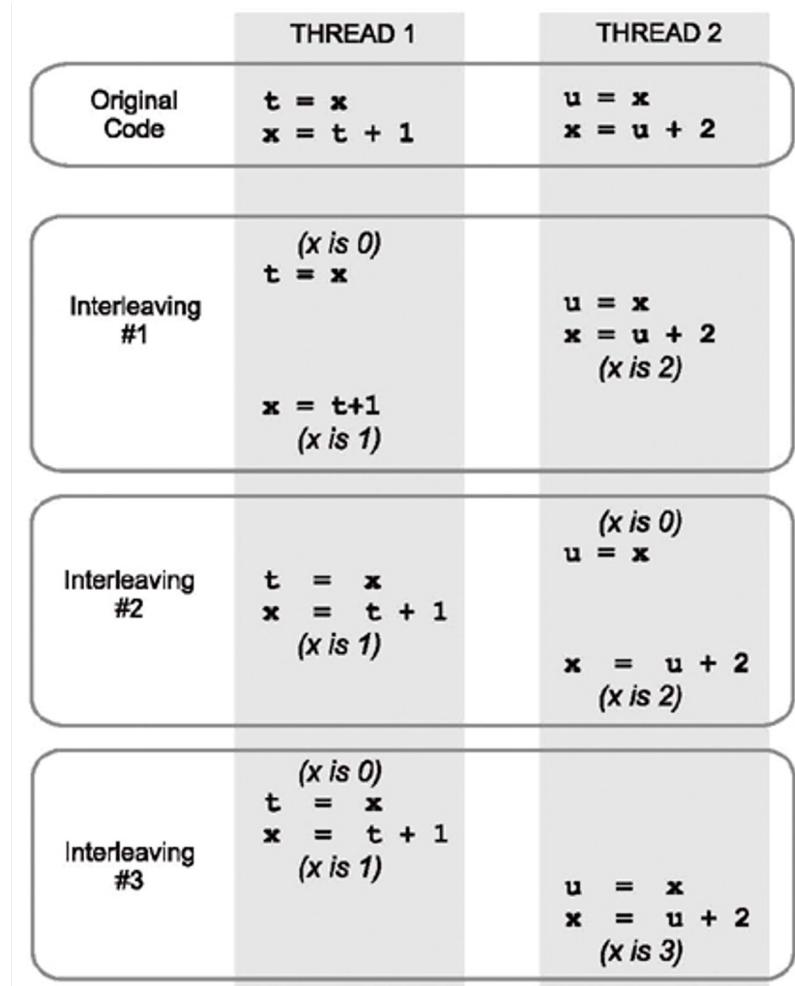


Conceptos clave

- Condición de carrera.
- Región Crítica.
- Sincronización:
 - Mutex.
 - Thread state: running, waiting.
 - Thread wakeup.

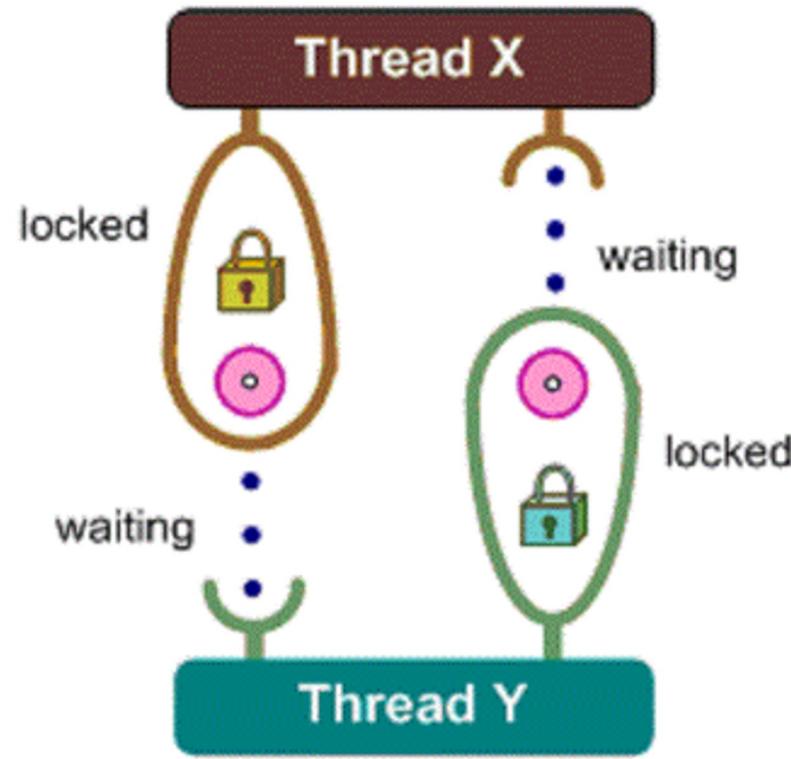
Recapitulación

- Programación en paralelo:
 - Condiciones de carrera.
 - No-determinismo.



Recapitulación

- Mecanismos de sincronización (Java).
 - Bloques sincronizados.



Recapitulación

- Métodos sincronizados (azúcar sintáctico):

```
public class SynchronizedCounter {  
    private int c = 0;  
  
    public synchronized void increment() {  
        c++;  
    }  
}
```

Cooperación entre hilos

- Escenario 1:
 - Un hilo ‘coordinador’ y N hilos ‘trabajadores’.
 - El coordinador activa a los trabajadores, pero debe esperar a que éstos terminen para consolidar sus resultados.
- Ejemplos?
- Estrategia?

Cooperación entre hilos

- Escenario 2:
 - El trabajo se realiza por ‘rondas’, donde:
 - Cada trabajador hace su tarea y espera para la siguiente ‘ronda’.
 - El coordinador espera a que todos los trabajadores realicen su trabajo, y consolida los resultados.
 - Cuando esto se ha hecho, el coordinador inicia la siguiente ronda diciéndole a los trabajadores que se activen de nuevo.
- Ejemplos?
- Estrategia?

Amdahl vs Moore

- Moore's law



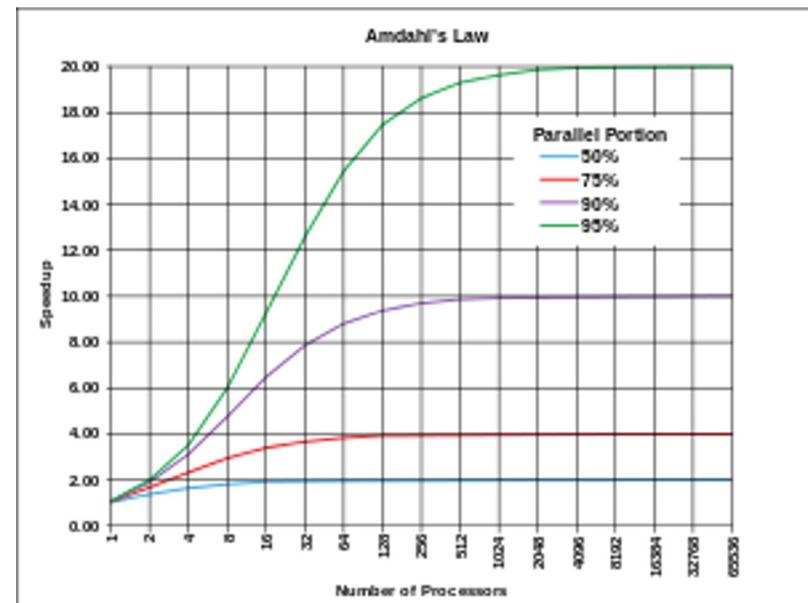
- Amdahl's law

$$T_N = \alpha + \frac{(1 - \alpha)}{N}$$

N : No. of

T_N : Time with N cores

α : Fraction of instructions in serial code



Thread-safety

- Si un objeto de tipo A será compartido por múltiples hilos, A debe ser '*thread-safe*'.
- El tipo A es '*thread-safe*' si:
 - Garantiza consistencia frente a una manipulación concurrente:
 - No comparta la variable de estado entre subprocessos.
 - Hacer que la variable de estado sea inmutable.
 - Utilice la sincronización cada vez que acceda a la variable de estado(encapsulamiento).

Thread-safety

- Tipos atómicos

Class AtomicInteger

```
java.lang.Object  
java.lang.Number  
java.util.concurrent.atomic.AtomicInteger
```

Class ConcurrentLinkedQueue<E>

```
java.lang.Object  
java.util.AbstractCollection<E>  
java.util.AbstractQueue<E>  
java.util.concurrent.ConcurrentLinkedQueue<E>
```

Thread-safety

- Colecciones sincronizadas

synchronizedCollection

```
public static <T> Collection<T> synchronizedCollection(Collection<T> c)

Collection c = Collections.synchronizedCollection(myCollection);
...
synchronized (c) {
    Iterator i = c.iterator(); // Must be in the synchronized block
    while (i.hasNext())
        foo(i.next());
}
```

Thread-safety

- Colecciones concurrentes

Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms*

Maged M. Michael Michael L. Scott

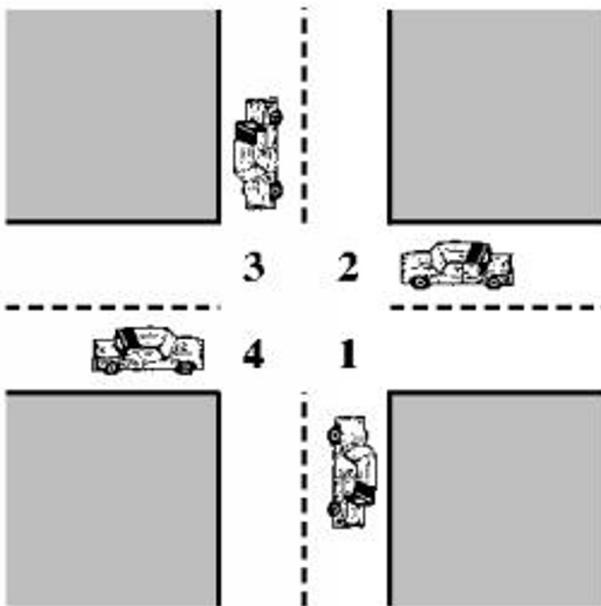
Department of Computer Science

University of Rochester

Rochester, NY 14627-0226

{michael, scott}@cs.rochester.edu

Deadlocks



(a) Deadlock possible



(b) Deadlock

Deadlocks

- Cómo prevenirlos?
 - Trabajar secuencialmente, o con un enfoque no-bloqueante.
 - Evitar ‘locks’ anidados.
 - Si lo anterior es inevitable, adquirir los locks en el mismo orden!

Deadlocks

- CuentaBancaria
 - depositar(int monto)
 - retirar(int monto)
- Transacción:
 - transferencia(CuentaBancaria c1, CuentaBancaria c2, monto)
 - Condición de carrera?
 - Cómo evitarla?