

# Programación Concurrente 2015

## Clase 1



Facultad de Informática  
UNLP

# Metodología del curso

- ♦ **Comunicación:** Plataforma WebUNLP ([webunlp.unlp.edu.ar](http://webunlp.unlp.edu.ar)).
  - Solicitar inscripción.
- ♦ **Bibliografía / material:**
  - *Libro base:* Foundations of Multithreaded, Parallel, and Distributed Programming. Gregory Andrews. Addison Wesley.  
([www.cs.arizona.edu/people/greg/mpdbook](http://www.cs.arizona.edu/people/greg/mpdbook)).
  - Material de lectura adicional: bibliografía, web.
    - Principles of Concurrent and Distributed Programming, 2/E. Ben-Ari. Addison-Wesley
    - An Introduction to Parallel Computing. Design and Analysis of Algorithms, 2/E. Grama, Gupta, Karypis, Kumar. Pearson Addison Wesley.
    - The little book of semaphores. Downey.  
<http://www.cs.ucr.edu/~kishore/papers/semaphores.pdf>.
  - Planteo de temas/ejercicios (recomendado hacerlos).

# Metodología del curso

## ◆ **Horarios:**

- **Teoría (2 turnos):**
  - Lunes de 8 a 11 - aula 5.
  - Jueves de 15 a 18 - aula 4.
  
- **Prácticas:**
  - Martes de 8 a 10 - aula 9.
  - Jueves de 18 a 20 - aula 4.
  - Sábado de 9:30 a 11:30 - aula 1.
  
- **Explicación de Ejercicios:**
  - Se hará en las clases de teoría.

# Metodología del curso

- ◆ **Cursada:** la cursada se aprueba con un parcial que cuenta con dos recuperatorios.
- ◆ Para la promoción práctica (optativa) se deben rendir 3 parcialitos prácticos (en la teoría de los lunes):
  - Si los 3 parcialitos son aprobados con nota  $\geq 8 \rightarrow$  se obtiene la cursada.
  - Si el promedio de los parcialitos es  $\geq 6$  y al menos 2 son aprobados  $\rightarrow$  rinde un parcial reducido (la condición vale tanto para el parcial como para los recuperatorios).
- ◆ **Final:** se aprueba por medio de un final tradicional teorico-práctico.
- ◆ Para la promoción teórica (optativa) se debe:
  - Rendir al menos 2 de 3 parcialitos teóricos (en la teoría de los lunes).
  - Una prueba teórica (en febrero de 2016). Dependiendo de la nota hay tiempo hasta marzo de 2018 para:
    - Nota  $\geq 7 \rightarrow$  coloquio en mesa de final.
    - Nota  $\geq 4$  y  $< 7 \rightarrow$  trabajo individual.

# Motivaciones del curso

- ◆ ¿Por qué es importante la concurrencia?
- ◆ ¿Cuáles son los problemas de concurrencia en los sistemas?
- ◆ ¿Cómo se resuelven usualmente esos problemas?
- ◆ ¿Cómo se resuelven los problemas de concurrencia a diferentes niveles (HW, SO, lenguajes, aplicaciones)?
- ◆ ¿Cuáles son las herramientas?

# Objetivos del curso

- ◆ Plantear los fundamentos de programación concurrente, estudiando sintaxis y semántica, así como herramientas y lenguajes para la resolución de programas concurrentes.
- ◆ Analizar el concepto de sistemas concurrentes que integran la arquitectura de Hardware, el Sistema Operativo y los algoritmos para la resolución de problemas concurrentes.
- ◆ Estudiar los conceptos fundamentales de comunicación y sincronización entre procesos, por Memoria Compartida y Pasaje de Mensajes.
- ◆ Vincular la concurrencia en software con los conceptos de procesamiento distribuido y paralelo, para lograr soluciones multiprocesador con algoritmos concurrentes.

# Temas del curso

- ◆ **Conceptos básicos.** Concurrencia y arquitecturas de procesamiento. Multithreading, Procesamiento Distribuido, Procesamiento Paralelo.
- ◆ **Concurrencia por memoria compartida.** Procesos y sincronización. Locks y Barreras. Semáforos. Monitores. Resolución de problemas concurrentes con sincronización por MC.
- ◆ **Concurrencia por pasaje de mensajes (MP).** Mensajes asincrónicos. Mensajes sincrónicos. Remote Procedure Call (RPC). Rendezvous. Paradigmas de interacción entre procesos.
- ◆ **Lenguajes que soportan concurrencia.** Características. Similitudes y diferencias.
- ◆ **Introducción a la programación paralela.** Conceptos, herramientas de desarrollo, aplicaciones.

# Concurrencia

## ¿Que es?

- ◆ Concurrencia es la capacidad de ejecutar múltiples actividades en paralelo o simultáneamente.
- ◆ Permite a distintos objetos actuar al mismo tiempo.
- ◆ Un concepto clave dentro de la ciencia de la computación. Factor relevante para el diseño de hardware, sistemas operativos, multiprocesadores, computación distribuida, programación y diseño.
- ◆ La necesidad de sistemas de cómputo cada vez más poderosos y flexibles atenta contra la simplificación de asunciones de secuencialidad.

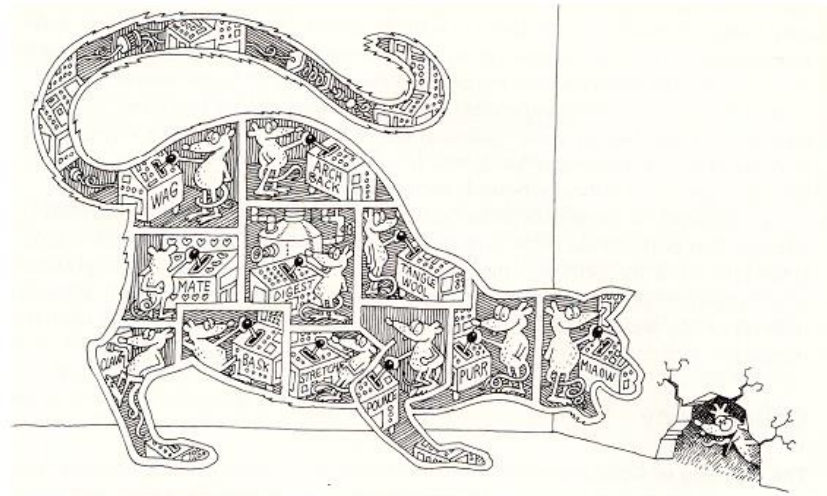
## ¿Donde está?

- ◆ Navegador Web accediendo una página mientras atiende al usuario.
- ◆ Varios navegadores accediendo a la misma página.
- ◆ Acceso a disco mientras otras aplicaciones siguen funcionando.
- ◆ Impresión de un documento mientras se consulta.
- ◆ El teléfono avisa recepción de llamada mientras se habla.
- ◆ Varios usuarios conectados al mismo sistema (reserva de pasajes).
- ◆ Cualquier objeto más o menos “inteligente” exhibe concurrencia.
- ◆ Juegos, automóviles, etc.



# Concurrencia

- ◆ Los sistemas biológicos suelen ser masivamente concurrentes: comprenden un gran número de células, evolucionando simultáneamente y realizando (independientemente) sus procesos.



- ◆ En el mundo biológico los sistemas secuenciales rara vez se encuentran.
- ◆ En algunos casos se tiende a pensar en sistemas secuenciales en lugar de concurrentes para simplificar el proceso de diseño. Pero esto va en contra de la necesidad de sistemas de cómputo cada vez más poderosos y flexibles.

# Concurrencia “natural”

- ♦ **Problema:** Desplegar cada 3 segundos un cartel ROJO.
- ♦ **Solución secuencial:**

*Programa Cartel*

Mientras (true)

Demorar (3 seg)

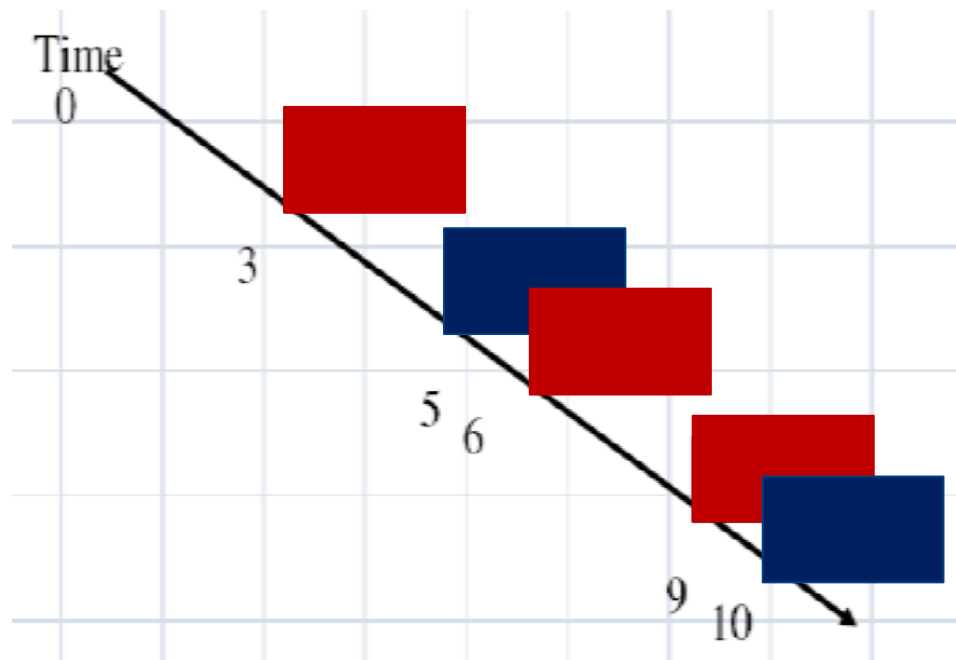
Desplegar cartel

Fin mientras

*Fin programa*

# Concurrencia “natural”

- ◆ **Problema:** Desplegar cada 3 segundos un cartel ROJO y cada 5 segundos un cartel AZUL.



# Concurrencia “natural”

## *Programa Carteles*

Proximo\_Rojo = 3

Proximo\_Azul = 5

Actual = 0

Mientras (true)

Si (Proximo\_Rojo < Proximo\_Azul)

Demorar (Proximo\_Rojo – Actual)

Desplegar cartel ROJO

Actual = Proximo\_Rojo

Proximo\_Rojo = Proximo\_Rojo +3

sino

Demorar (Proximo\_Azul – Actual)

Desplegar cartel AZUL

Actual = Proximo\_Azul

Proximo\_Azul = Proximo\_Azul +5

Fin mientras

*Fin programa*

# Concurrencia “natural”

- ◆ Obliga a establecer un orden en el despliegue de cada cartel.
- ◆ Código más complejo de desarrollar y mantener.
- ◆ ¿Que pasa si se tienen más de dos carteles?
- ◆ ***Más natural:*** cada cartel es un elemento independiente que actúa concurrentemente con otros → *es decir, ejecutar dos o más algoritmos simples concurrentemente.*

```
Programa Cartel (color, tiempo)  
    Mientras (true)  
        Demorar (tiempo segundos)  
        Desplegar cartel (color)  
    Fin mientras  
Fin programa
```

- ◆ No hay un orden preestablecido en la ejecución

# ¿Por qué es necesaria la Programación Concurrente?

- No hay más ciclos de reloj → Multicore → ¿por qué? y ¿para qué?
- Aplicaciones con estructura más natural.
  - El mundo no es secuencial.
  - Más apropiado programar múltiples actividades independientes y concurrentes.
  - Reacción a entradas asincrónicas (ej: sensores en un STR).
- Mejora en la respuesta
  - No bloquear la aplicación completa por E/S.
  - Incremento en el rendimiento de la aplicación por mejor uso del hardware (ejecución paralela).
- Sistemas distribuidos
  - Una aplicación en varias máquinas.
  - Sistemas C/S o P2P.

# Objetivos de los sistemas concurrentes

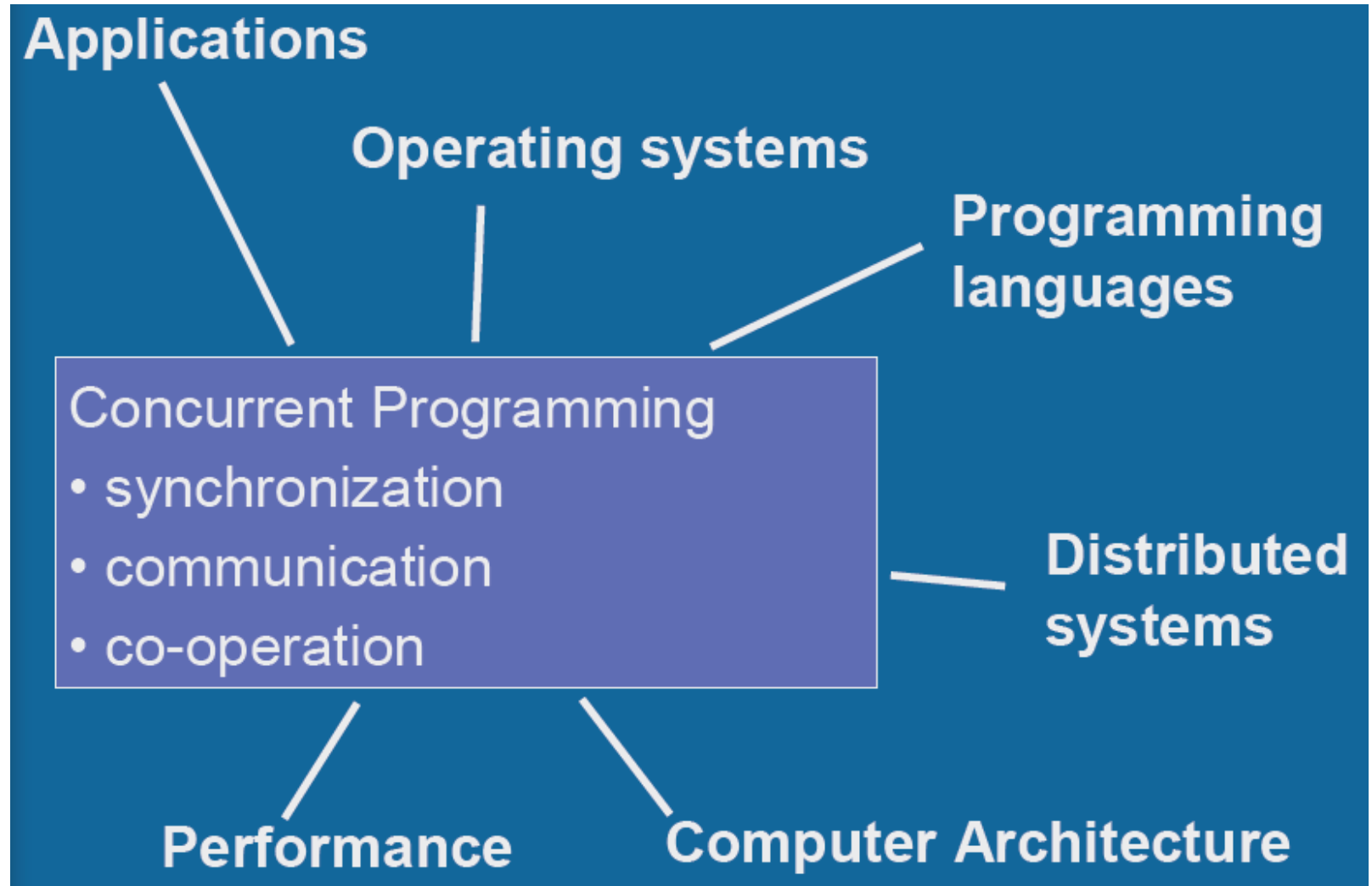
*Ajustar el modelo de arquitectura de hardware y software al problema del mundo real a resolver.*

*Incrementar la performance, mejorando los tiempos de respuesta de los sistemas de cómputo, a través de un enfoque diferente de la arquitectura física y lógica de las soluciones.*

## Algunas ventajas ⇒

- La velocidad de ejecución que se puede alcanzar.
- Mejor utilización de la CPU de cada procesador.
- Explotación de la concurrencia inherente a la mayoría de los problemas reales.

# Conexiones





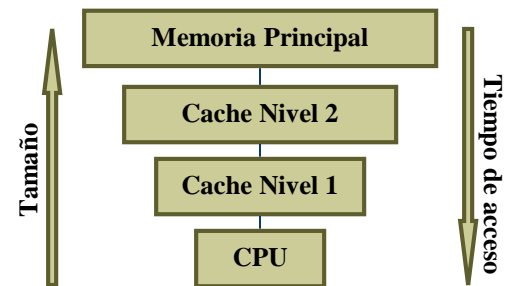
# Concurrencia a nivel de hardware

## Límite físico en la velocidad de los procesadores

- Máquinas monoprocesador ya no pueden mejorar.
- Más procesadores por chip para mayor potencia de cómputo.
- Multicores → Cluster de multicores → Consumo.
- Uso eficiente → Programación concurrente y paralela.

## Niveles de memoria.

- Jerarquía de memoria. ¿Consistencia?
- Diferencias de tamaño y tiempo de acceso.
- Localidad temporal y espacial de los datos.

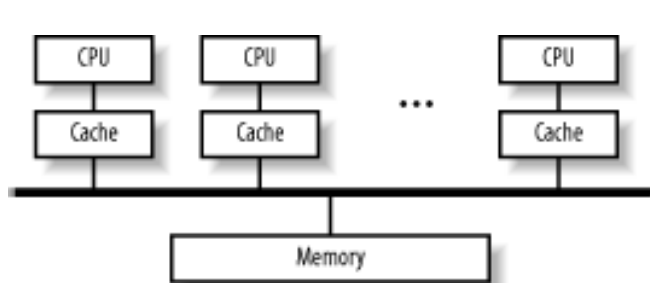


## Máquinas de memoria compartida vs memoria distribuida.

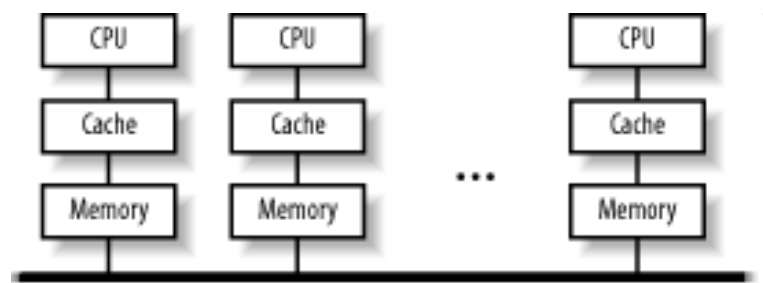
# Concurrencia a nivel de hardware

## Multiprocesadores de memoria compartida.

- Interacción modificando datos en la memoria compartida.
- Esquemas UMA con bus o crossbar switch (SMP, multiprocesadores simétricos). Problemas de sincronización y consistencia.
- Esquemas NUMA para mayor número de procesadores distribuidos.
- Problema de consistencia.



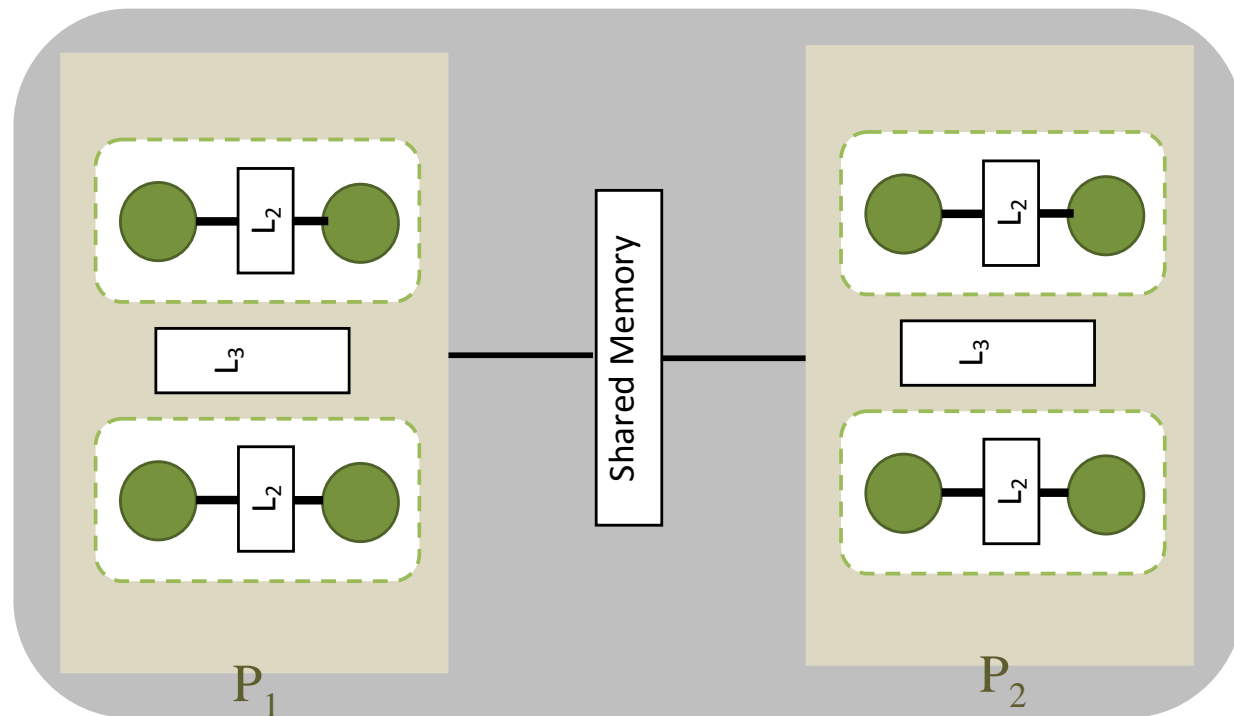
Esquema UMA



Esquema NUMA

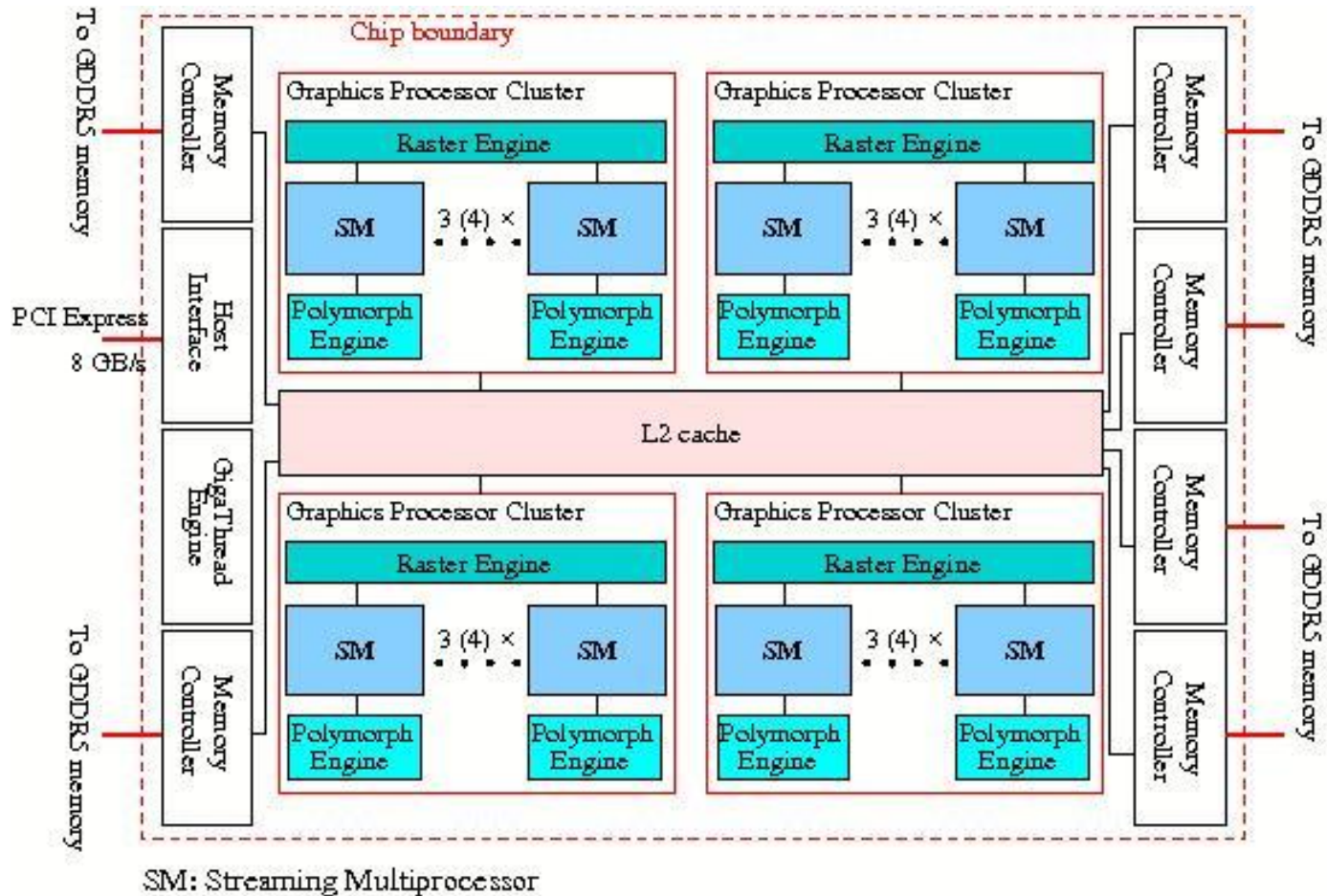
# Concurrencia a nivel de hardware

- Ejemplo de multiprocesador de memoria compartida: *multicore de 8 núcleos*.



# Concurrencia a nivel de hardware

- Ejemplo de multiprocesador de memoria compartida: *GPU*.



# Concurrencia a nivel de hardware

## Multiprocesadores con memoria distribuida.

- Procesadores conectados por una red.
- Memoria local (no hay problemas de consistencia).
- Interacción es sólo por pasaje de mensajes.
- Grado de acoplamiento de los procesadores:
  - Multicomputadores (máquinas fuertemente acopladas). Procesadores y red físicamente cerca. Pocas aplicaciones a la vez, cada una usando un conjunto de procesadores. Alto ancho de banda y velocidad.
  - Memoria compartida distribuida.
  - Clusters.
  - Redes (multiprocesador debilmente acoplado).



# ¿Que es un proceso?

**PROCESO:** programa secuencial (un único flujo de control que ejecuta una instrucción y cuando esta finaliza ejecuta la siguiente)

Un único hilo o flujo de control

→ programación secuencial, monoprocesador.

Múltiples hilos o flujos de control

→ programa concurrente.

→ procesos paralelos.

Los procesos cooperan y compiten...



# Posibles comportamientos de los procesos

## Procesos independientes

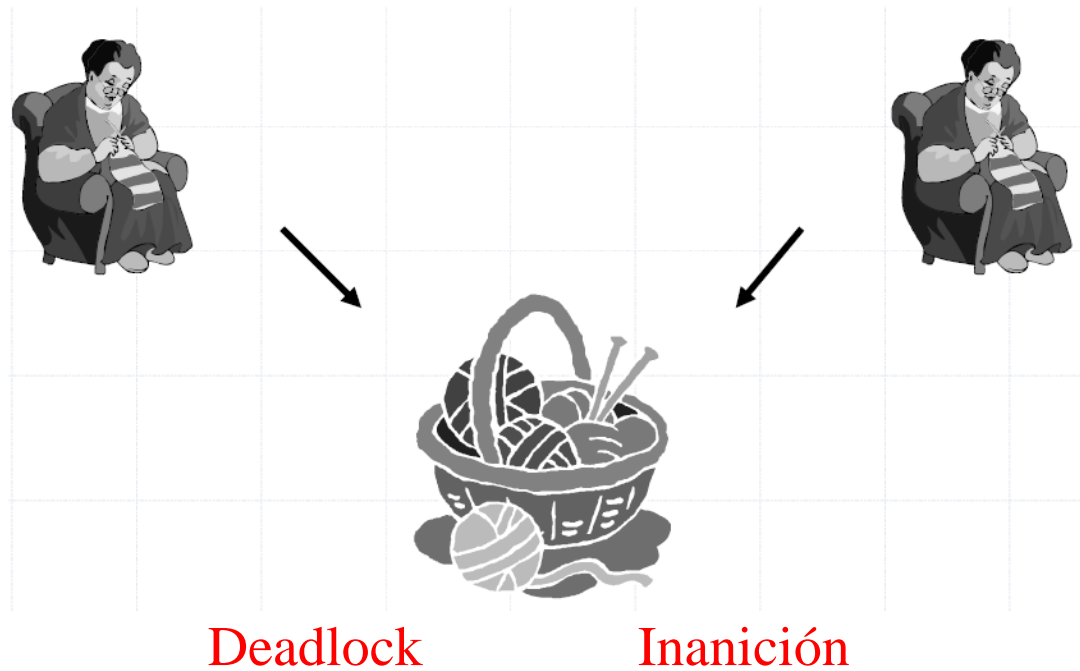
- Relativamente raros.
- Poco interesante.



# Posibles comportamientos de los procesos

## Competencia

- Típico en Sistemas Operativos y Redes, debido a recursos compartidos.

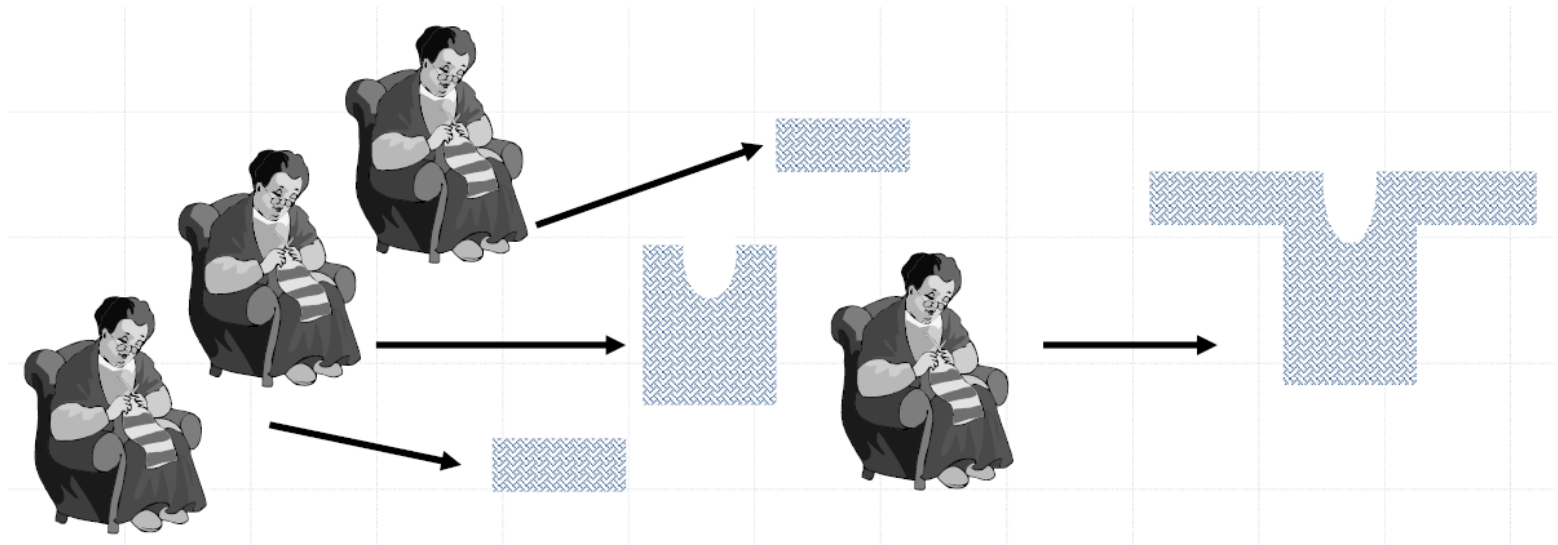




# Posibles comportamientos de los procesos

## Cooperación

- Los procesos se combinan para resolver una tarea común.
- Sincronización.



# Procesamiento secuencial, concurrente y paralelo

Analicemos la solución secuencial y monoprocesador (*UN ALBAÑIL*) para construir un barrio de  $N$  casas.

La solución secuencial **nos fuerza** a establecer un **estricto orden temporal**.

Al disponer de un sólo albañil, el trabajo se termina luego de  $N$  pasos (casas construidas).

# Procesamiento secuencial, concurrente y paralelo

Si disponemos de  $N$  *albañiles* para construir las casas, y **no hay dependencia** de la materia prima, cada uno puede trabajar *al mismo tiempo* en una casa.

## Consecuencias $\Rightarrow$

- Menor tiempo para completar el trabajo.
- Menor esfuerzo individual.
- Paralelismo del hardware.

## Dificultades $\Rightarrow$

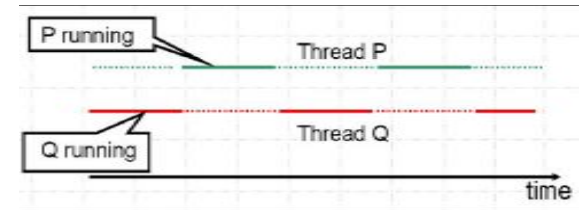
- Distribución de la carga de trabajo (tamaño de la casa, velocidad del albañil).
- Necesidad de compartir recursos.
- Necesidad de esperarse en puntos clave (cosas que comparten en el barrio).
- Necesidad de comunicarse.
- Tratamiento de las fallas.
- Asignación de uno de los albañiles para las cosas compartidas por todas las casas (¿Cual?).

# Procesamiento secuencial, concurrente y paralelo

**Otro enfoque:** *un sólo albañil* dedica una parte del tiempo a cada componente del barrio  $\Rightarrow$  **Concurrencia sin paralelismo de hardware**  $\Rightarrow$  Menor speedup.

## Dificultades $\Rightarrow$

- Distribución de carga de trabajo.
- Necesidad de compartir recursos.
- Necesidad de esperarse en puntos clave.
- Necesidad de comunicarse.
- Necesidad de recuperar el “estado” de cada proceso al retomarlo.



**CONCURRENCIA**  $\Rightarrow$  Concepto de software no restringido a una arquitectura particular de hardware ni a un número determinado de procesadores.

# Procesamiento secuencial, concurrente y paralelo

## Este último caso sería multiprogramación en un procesador

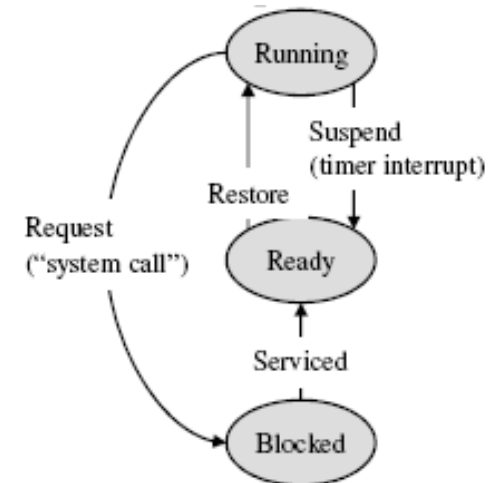
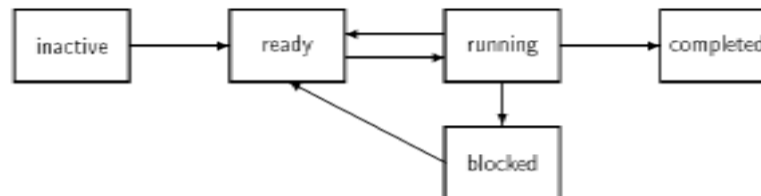
- El tiempo de CPU es compartido entre varios procesos, por ejemplo por *time slicing*.
- El sistema operativo controla y planifica procesos: si el slice expiró o el proceso se bloquea el sistema operativo hace *context (process) switch*.

*Process switch: suspender el proceso actual y restaurar otro*

1. Salvar el estado actual en memoria. Agregar el proceso al final de la cola de *ready* o una cola de *wait*.
2. Sacar un proceso de la cabeza de la cola *ready*. Restaurar su estado y ponerlo a correr.

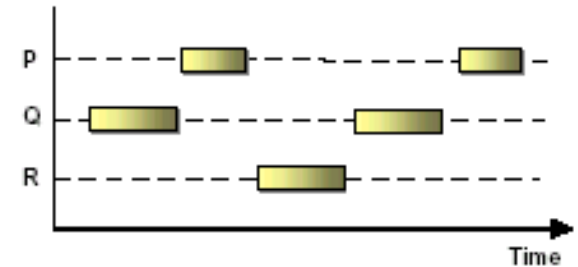
*Reanudar un proceso bloqueado: mover un proceso de la cola de wait a la de ready.*

- Estados de los Procesos



# Programa Concurrente

*Un programa concurrente especifica dos o más programas secuenciales que pueden ejecutarse concurrentemente en el tiempo como tareas o procesos.*



Un proceso o tarea es un elemento concurrente abstracto que puede ejecutarse simultáneamente con otros procesos o tareas, si el hardware lo permite (por ejemplo los TASKs de ADA).

Un programa concurrente puede tener  $N$  **procesos** habilitados para ejecutarse concurrentemente y un sistema concurrente puede disponer de  $M$  **procesadores** cada uno de los cuales puede ejecutar uno o más procesos.

Características importantes:

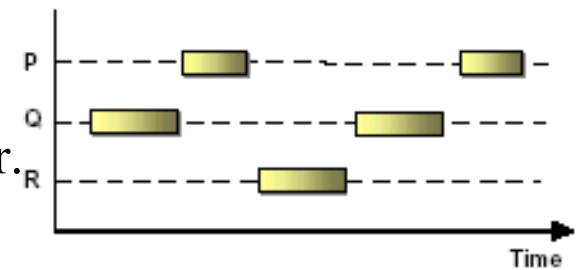
- interacción
- no determinismo  $\Rightarrow$  dificultad para la interpretación y debug
- ejecución infinita

# Programa Concurrente: *Concurrencia vs. Paralelismo*

La concurrencia no implica paralelismo

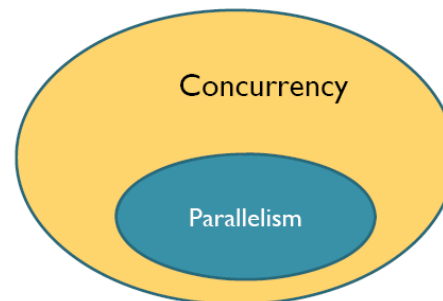
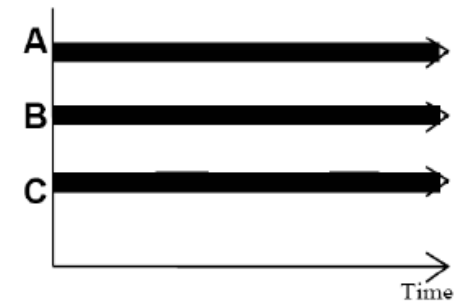
## ◆ Concurrencia “*Interleaved*” →

- Procesamiento lógicamente simultáneo.
- Ejecución intercalada en un único procesador.
- Pseudo-paralelismo.



## ◆ Concurrencia “*Simultánea*” →

- Procesamiento físicamente simultáneo.
- Requiere un sistema multiprocesador o multicore.
- Paralelismo “full”.



# Procesos e Hilos

- Todos los Sistemas Operativos soportan *Procesos*.
  - Cada proceso tiene su propio espacio de direcciones y recursos.
- Algunos Sistemas Operativos soportan procesos livianos (*threads o hilos*).
  - Proceso “liviano” que tiene su propio contador de programa y su pila de ejecución, pero no controla el “contexto pesado” (por ejemplo, las tablas de página).
  - Todos los hilos de un proceso comparten el mismo espacio de direcciones y recursos.
  - El programador o el lenguaje deben proporcionar mecanismos para evitar interferencias.
  - La concurrencia puede estar provista por el lenguaje (ADA, Java) o por el Sistema Operativo (C/POSIX).



# Secuencialidad y concurrencia

- **Programa secuencial**  $\Rightarrow$

- Totalmente ordenado
- Determinístico (para los mismos datos de entrada, ejecuta siempre la misma secuencia de instrucciones y obtiene la misma salida)

- **Programa concurrentes**  $\Rightarrow$  ¿que sucede en este caso?

**Ejemplo**  $\Rightarrow$

- $x=0$ ; //P
- $y=0$ ; //Q
- $z=0$ ; //R

- Instrucciones lógicamente concurrentes  $\rightarrow$  Orden de ejecución es irrelevante.

$$\left. \begin{array}{l} P \rightarrow Q \rightarrow R \\ Q \rightarrow P \rightarrow R \\ R \rightarrow Q \rightarrow P \end{array} \right\} \text{ Darán el mismo resultado}$$

# Secuencialidad y concurrencia

**Ejemplo**  $\Rightarrow$   $\left. \begin{array}{l} \bullet P \equiv p_1, p_2, p_3, p_4, \dots \\ \bullet Q \equiv q_1, q_2, q_3, q_4, \dots \\ \bullet R \equiv r_1, r_2, r_3, r_4, \dots \end{array} \right\} P, Q, R \text{ independientes.}$

- Orden parcial  $\rightarrow$  La única regla es que  $p_i$  ejecuta antes que  $p_j$  si  $i < j$  (idem con R y Q).
- Los siguientes ordenamientos darán el mismo resultado.

$p_1, p_2, q_1, r_1, q_2 \dots$   
 $q_1, r_1, q_2, p_1 \dots$

**Ejemplo**  $\Rightarrow$  Suponemos que  $x = 5$ .

- $x = 0; //P$
- $x = x + 1; //Q$

- Orden  $\rightarrow$  Diferentes ejecuciones, pueden dar distintos resultados.  
 $P \rightarrow Q \Rightarrow x = 1 \quad Q \rightarrow P \Rightarrow x = 0$

Los programas concurrentes pueden ser **no-determinísticos**: pueden dar distintos resultados al ejecutarse sobre los mismos datos de entrada.

# Clases de aplicaciones

## Programación Concurrente $\Rightarrow$

- organizar software que consta de partes (relativamente) independientes.
- usar uno o múltiples procesadores.

## 3 grandes clases (superpuestas) de aplicaciones

- Sistemas **multithreaded**.
- Sistemas de **cómputo distribuido**.
- Sistemas de **cómputo paralelo**.

# Clases de aplicaciones

**Un sistema de software “multithreading”** maneja simultáneamente múltiples actividades independientes, asignando los procesadores de acuerdo a alguna política.  
*Ejecución de  $N$  procesos independientes en  $M$  procesadores ( $N > M$ ).*

## Ejemplos:

- Sistemas de ventanas en PCs o WS
- Sistemas Operativos time-shared y multiprocesador
- Sistemas de tiempo real (por ejemplo, en plantas industriales o medicina)

# Clases de aplicaciones

## Cómputo distribuido

Una red de comunicaciones vincula procesadores diferentes sobre los que se ejecutan procesos que se comunican esencialmente por mensajes. Cada componente del sistema distribuido puede hacer a su vez multithreading.

### Ejemplos:

- Servidores de archivos en una red.
- Sistemas de BD en bancos y aerolíneas (acceso a datos remotos).
- Servidores Web distribuidos (acceso a datos remotos).
- Sistemas corporativos que integran componentes de una empresa.
- Sistemas *fault-tolerant* que incrementan la confiabilidad.

# Clases de aplicaciones

## **Procesamiento paralelo**

Resolver un problema en el menor tiempo (o un problema más grande en el mismo tiempo) usando una arquitectura multiprocesador en la que se pueda distribuir la tarea global en tareas que puedan ejecutarse en distintos procesadores.

Paralelismo de datos y paralelismo de procesos.

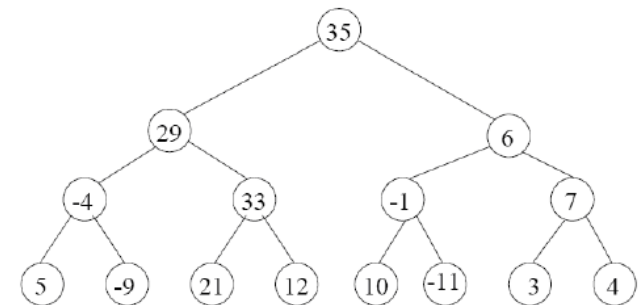
### **Ejemplos:**

- Cálculo científico para modelar y simular sistemas naturales.
- Procesamiento gráfico y de imágenes, efectos especiales, procesamiento de video, realidad virtual.
- Problemas combinatorios y de optimización lineal o no lineal. Modelos econométricos.

# Ejemplo simple de concurrencia

## Sumar 32 números

- Una persona (*Secuencial*): 31 sumas, cada una 1 unidad de tiempo.
- Dos personas: cada una suma 16 números. Reducción del tiempo a la mitad. ¿Cuántas unidades de tiempo lleva?
- ¿Que sucede si al aumentar la cantidad de personas?
- Máxima concurrencia: 16 personas
  - Árbol binario
  - ¿Cuántas unidades de tiempo?



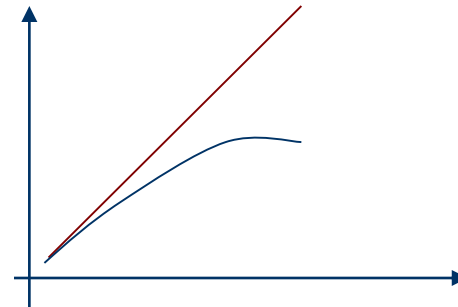
El *cómputo concurrente* con mayor grado de concurrencia resulta en un menor tiempo de ejecución

# ¿Cómo se mide el incremento de performance?

El procesamiento paralelo lleva a los conceptos de **speedup** y **eficiencia**.

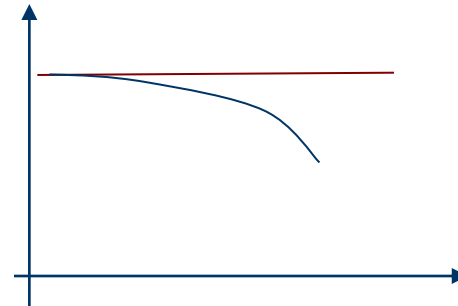
◆ *Speedup*  $\Rightarrow S = T_s / T_p$

- Significado.
- Rango de valores.



◆ *Eficiencia*  $\Rightarrow E = S / p$

- Significado.
- Rango de valores.



En la ejecución concurrente, el **speedup** es menor.



# Tareas propuestas

- ◆ Leer los capítulos 1 y 2 del libro de Andrews
- ◆ Leer los ejemplos de paralelismo recursivo, productores y consumidores, clientes y servidores y el código de la multiplicación de matrices distribuida del Capítulo 1 (Andrews).
- ◆ Investigar las primitivas de programación concurrente de algún lenguaje de programación.