

# Paradigmas y Lenguajes

## Paralelismo y Concurrency - 1



**Lic. Ricardo Monzón**

# **PROGRAMACION CONCURRENTE Y PARALELA INTRODUCCION.**

# CONCEPTOS Y DEFINICIONES

La programación a la que estamos más acostumbrados, la secuencial, estuvo en sus inicios fuertemente influenciada por las arquitecturas de único procesador, en las cuales disponíamos como características base:

- de un único procesador (CPU),
- de los programas y los datos que están almacenados en memoria RAM, y
- de los procesadores, que se dedican básicamente a obtener un determinado flujo de instrucciones desde la RAM, ejecutando una instrucción por unidad de tiempo

En este sentido los programas secuenciales son totalmente ordenados, éstos nos indican en qué orden serán ejecutadas las instrucciones.

# CONCEPTOS Y DEFINICIONES

Y una particularidad importante, los programas secuenciales son deterministas:

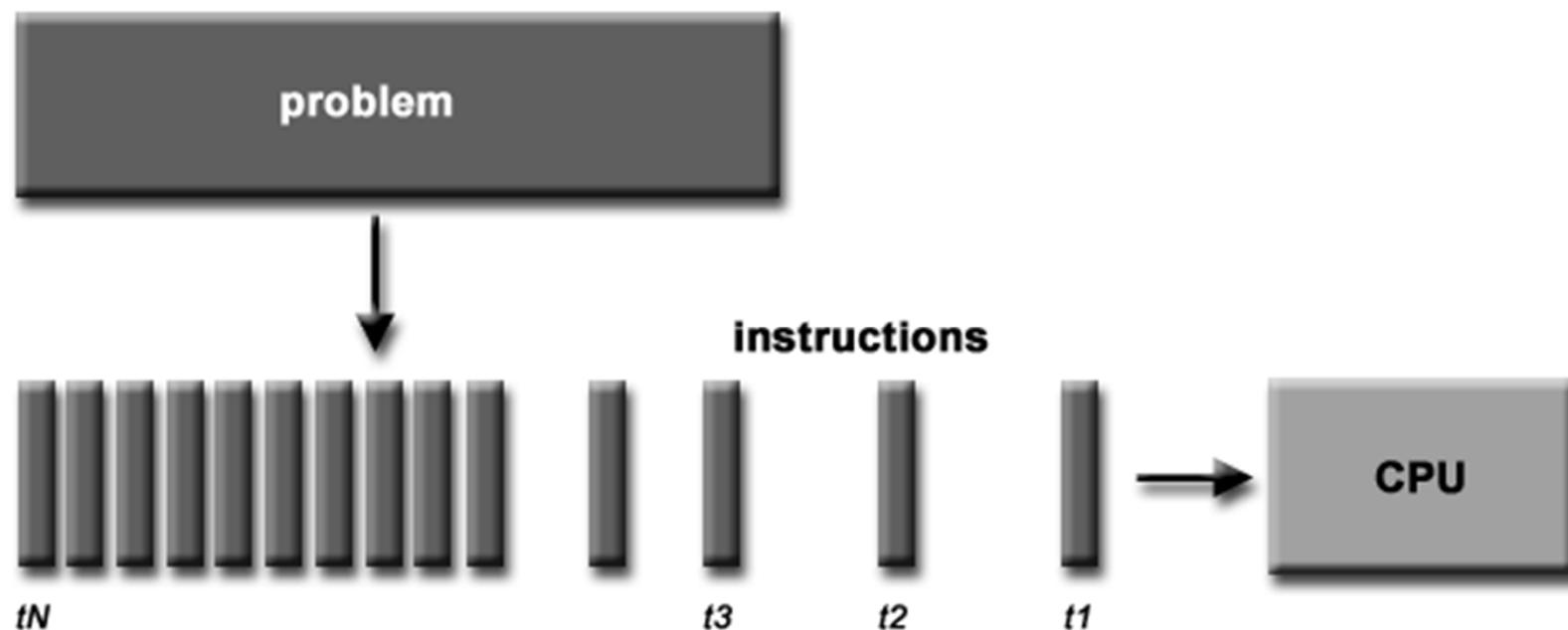
mediante una misma secuencia de datos de entrada, se ejecutará la misma secuencia de instrucciones y se producirá el mismo resultado (salvo errores de ejecución causados por causas externas).

*Esta afirmación no es cierta para la programación concurrente.*

# CONCEPTOS Y DEFINICIONES

Tradicionalmente, el software es escrito para ser computado en serie.  
Esto significa:

- Es ejecutado en una **única computadora** con una **única CPU**
- El **problema** se divide en una **sucesión de instrucciones**
- Las instrucciones **se ejecutan una después de otra**
- Solo una instrucción puede ser ejecutada en un determinado instante de tiempo



# CONCEPTOS Y DEFINICIONES

## *Características de Programas Secuenciales*

- Es el estilo de programación que corresponde al modelo conceptual de **Von Newmann**.
- Un programa secuencial tiene una **línea simple** de control de flujo.
- Las operaciones de un programa secuencial están ordenadas de acuerdo con un **orden estricto**.
- El comportamiento de un programa es solo función de las sentencias que lo componen y del orden en que se ejecutan.
- **El tiempo** que tarda en ejecutarse cada operación **no influye** en el resultado de un programa secuencial.
- La **verificación** de un programa secuencial **es sencilla**:
  - Cada sentencia da la respuesta correcta.
  - Las sentencias se ejecutan en el orden adecuado.

# CONCEPTOS Y DEFINICIONES

La **Programación concurrente** tiene sus raíces en los sistemas operativos y en la programación de sistemas (años 60's).

Sistemas operativos fueron organizados con una colección de procesos ejecutándose concurrentemente (“al mismo tiempo”), algunos se ejecutaban en el procesador principal y otros en los controladores llamados canales.

La administración de estos sistemas se hacía a bajo nivel, en 1972 apareció el lenguaje **Pascal concurrente**, era el **primer lenguaje de alto nivel para este objetivo**.

# CONCEPTOS Y DEFINICIONES

La concurrencia surge ante la necesidad de plantear soluciones con el máximo rendimiento.

Existen varias aplicaciones computacionales que requieren una gran velocidad de cálculo:

- visualización
- bases de datos distribuidas
- simulaciones
- predicciones científicas.

Dentro de las áreas afines de la concurrencia tenemos:

- sistemas distribuidos
- sistemas en tiempo real

# CONCEPTOS Y DEFINICIONES

Acontecimientos que impulsaron la programación concurrente:

- El concepto de **hilo** hizo que los programas puedan ejecutarse con mayor velocidad comparados con los procesos utilizados anteriormente.
- Los lenguajes orientados objetos como Java que daban mejor soporte a la programación con la inclusión de primitivas.
- La aparición de Internet dadas las aplicaciones como el navegador, el chat están programados mediante técnicas de concurrencia

# CONCEPTOS Y DEFINICIONES: CONCURRENCIA

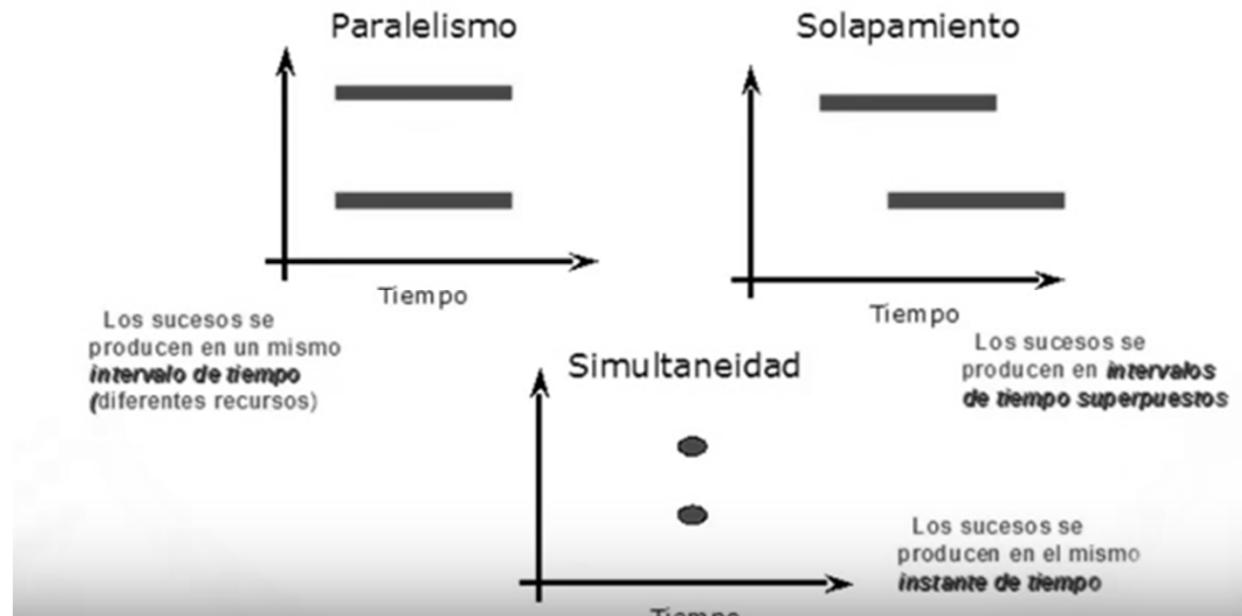
*«Es el conjunto de anotaciones y técnicas utilizadas para describir mediante programas el paralelismo potencial de los problemas, así como para resolver los problemas de comunicación y sincronización que se presentan cuando varios procesos que se reportan concurrentemente comparten recursos».*

Una propiedad básica de este tipo de programación es el **no determinismo**, se desconoce ante un instante de tiempo que va ocurrir en el siguiente instante:

- En el caso de un único procesador se desconoce si después la ejecución de una instrucción específica habrá alguna interrupción para brindar el procesador a otro proceso.
- En el caso del sistema multiprocesador las velocidades de los procesadores no están sincronizadas, por lo que se desconoce a priori cuál procesador va ser el primero en ejecutar su siguiente instrucción.

# CONCEPTOS Y DEFINICIONES: CONCURRENCIA

- Por tanto es necesario disponer de un modelo abstracto que permita verificar y corregir los sistemas concurrentes.
- Cada problema concurrente presenta un tipo distinto de paralelismo, la implementación depende de la arquitectura.
- Si se requiere trabajar en un ambiente independiente de la arquitectura se debe plantear un modelo para verificar que sea correcto independientemente del hardware en el que se ejecute.



# CONCEPTOS Y DEFINICIONES

La programación concurrente nos permite desarrollar aplicaciones que pueden ejecutar múltiples actividades de forma paralela o simultánea.

La programación concurrente es necesaria por varias razones:

- ***Ganancia de procesamiento***, ya sea en hardware multiprocesador o bien en un conjunto dado de computadoras. Mediante la obtención de ganancias en recursos, en capacidad de cómputo, memoria, recursos de almacenamiento, etc.
- ***Incremento del throughput de las aplicaciones*** (solapamiento E/S con cómputo).
- ***Incrementar la respuesta de las aplicaciones hacia el usuario***, poder atender estas peticiones frente al cómputo simultáneo. Permitiendo a los usuarios y computadores, mediante las capacidades extras, colaborar en la solución de un problema.

# CONCEPTOS Y DEFINICIONES

- *Es una estructura más apropiada* para programas que controlan múltiples actividades y gestionan múltiples eventos, con el objetivo de capturar la estructura lógica de un problema.
- *Da un soporte específico a los sistemas distribuidos.*

Otro punto con cierta controversia, son las diferencias entre los términos de *programación concurrente*, *distribuida*, *paralela*, y *multithread*. Todos ellos son usados en mayor o menor medida en la construcción de sistemas distribuidos y/o paralelos.

Usaremos *programación concurrente* como marco global a los diferentes tipos de programación distribuida, paralela y *multithread*, tendencia que viene siendo habitual en la literatura del campo distribuido.

Aun así, hay una serie de diferencias que hay que puntualizar:

# CONCEPTOS Y DEFINICIONES

**Concurrencia frente a paralelismo:** en **paralelismo** hay **procesamiento de cómputo simultáneo físicamente**, en **concurrencia el cómputo es simultáneo lógicamente**, ya que el paralelismo implica la existencia de múltiples elementos de procesamiento (ya sean CPU, o computadores enteros) con funcionamiento independiente; mientras que la concurrencia no implica necesariamente que existan múltiples elementos de procesamiento, ya que **puede ser simulada mediante las capacidades de multiprocesamiento del SO sobre la CPU**.

Un caso típico es la programación *multithread*, múltiples hilos de ejecución en una misma aplicación compartiendo memoria, pudiendo ser esta ejecución paralela si se ejecuta en una máquina multiprocesador, o concurrente, lógicamente si se ejecuta en un solo procesador. Aun existiendo estas diferencias, en la práctica las mismas técnicas son aplicables a ambos tipos de sistemas.

# CONCEPTOS Y DEFINICIONES

## CONCURRENCIA FRENTE A PARALELISMO

**Concurrente:** “Coincidencia, concurso simultáneo de varias circunstancias (RAE)”

- Es la capacidad de **algunas de las tareas** que resuelven un **problema** de poder ser realizadas en **paralelo**.
- Está presente en la naturaleza, la vida diaria, los sistemas de cómputo, etc.

**Paralelo:** “A la vez, a un mismo tiempo, simultáneamente (RAE)”

- Es la condición en que **dos o más tareas** que resuelven un **problema** se realizan **exactamente al mismo tiempo**.

Mientras la concurrencia es una característica de las tareas, el paralelismo se asocia a los recursos que se disponen para poder realizar esas tareas simultáneamente.

# CONCEPTOS Y DEFINICIONES: Paralelo y Distribuido

***La palabra paralelismo*** suele asociarse con altas prestaciones en cómputo científico en una determinada plataforma hardware, por ej. supercomputadores, aunque cada vez se desplaza más a plataformas basadas en *clusters* locales (computadoras enlazadas por redes de propósito general, o bien redes de alta velocidad optimizadas para procesamiento paralelo). Estas computadoras están físicamente en un mismo armario o sala de cómputo.

***La palabra distribuido*** se ha referido típicamente a aplicaciones, ejecutándose en múltiples computadoras (normalmente se implica además heterogeneidad de recursos), que no tenían por qué estar físicamente en un mismo espacio. Este término también relaja su significado debido a la extensión de los modelos paralelos, a ambientes como *multicluseter* (múltiples *clusters* enlazados), o *grid*, que, aun siendo generalmente considerados como paralelos, se acercan más a los sistemas distribuidos.

# CONCEPTOS Y DEFINICIONES

En general estos términos son ampliamente difundidos, y usados con significados diferentes, aunque las distinciones tienden a desaparecer, en especial cuando se consideran cada vez sistemas más potentes de multiprocesamiento, y la fusión de sistemas paralelos y distribuidos aumenta con entornos como *grid*. Llegando en la mayoría de los casos actuales a sistemas híbridos donde se integran los tres tipos de programación (o de sistemas).

Podemos asimismo partir de una definición de *sistema distribuido* de computación como:

*Una colección de elementos de cómputo autónomos, ejecutándose en uno o más computadores, enlazados por una red de interconexión, y soportados por diferentes tipos de comunicaciones que permiten observar la colección como un sistema integrado.*

## CONCEPTOS Y DEFINICIONES

Los sistemas distribuidos operan sobre redes de computadoras, pero también se puede construir uno que tenga sus elementos funcionando en un computador simple multitarea.

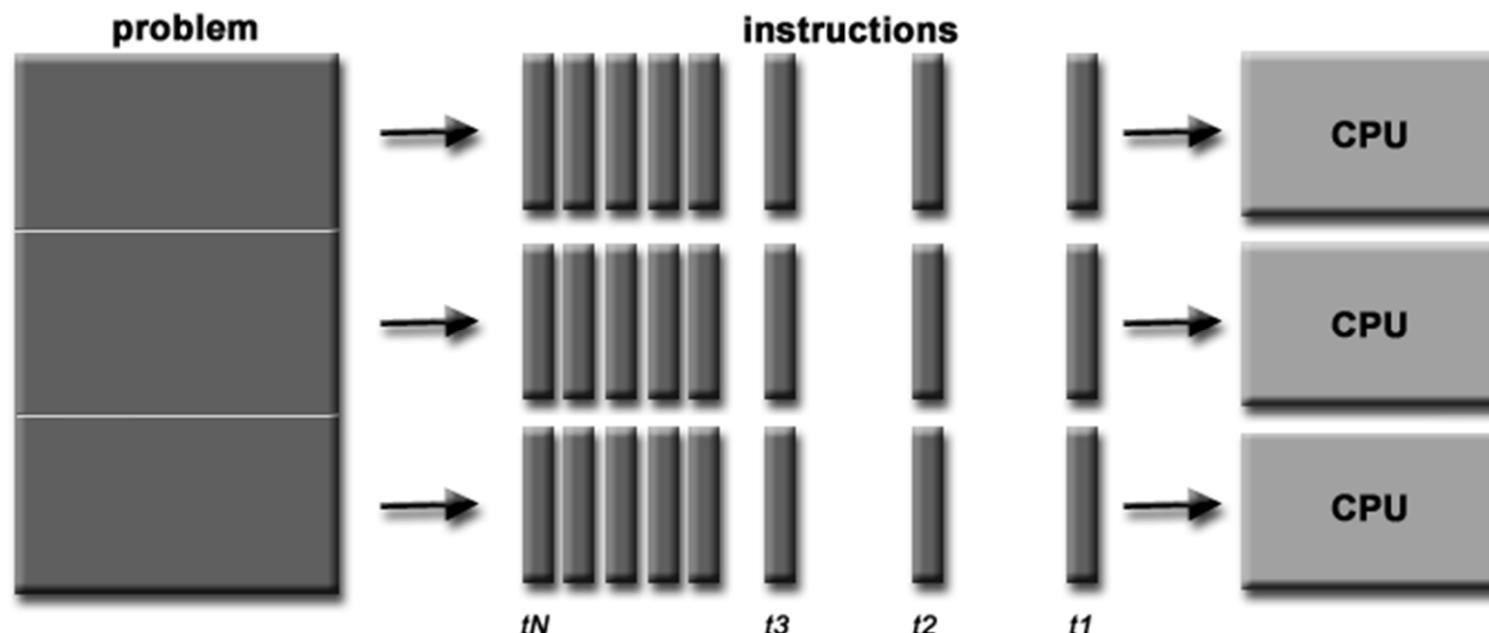
Además del clásico basado en redes, se pueden incluir los computadores paralelos, y especialmente los servidores en *cluster*, y otros ambientes como las redes *wireless*, y las redes de sensores. Por otra parte, la computación *grid* abarca coordinación de recursos que no están sujetos a control centralizado mediante el uso de protocolos e interfaces abiertas, de propósito general, para proporcionar diferentes servicios.

Uno de los parámetros más interesantes a la hora de enfocar la programación concurrente sobre un determinado sistema distribuido es conocer los diferentes estilos arquitectónicos en los que se basa, así como su modelo de interacción y organización interna de los componentes.

# CONCEPTOS Y DEFINICIONES

La **Computación Paralela** es el uso simultáneo de múltiples recursos de cómputo para resolver un problema computacional. Esto es:

- Se ejecuta utilizando **múltiples procesadores**
- El **problema es dividido** en partes que pueden resolverse **concurrentemente**
- Cada parte es dividida en una serie de instrucciones, que a su vez se ejecutan **simultáneamente** en diferentes procesadores
- Debe emplearse un mecanismo de **coordinación general**

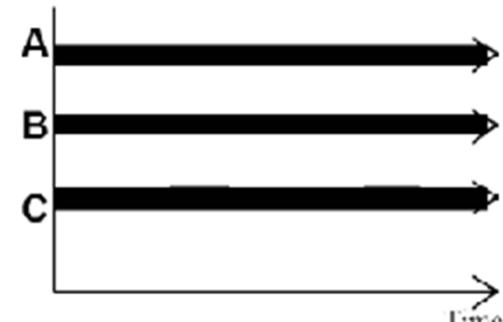
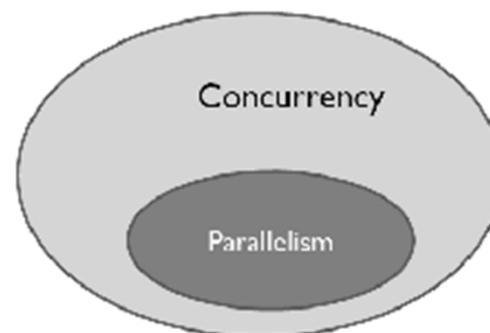
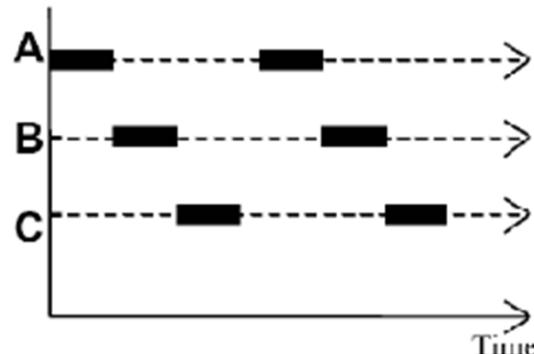


# CONCEPTOS Y DEFINICIONES

## *Procesamiento paralelo*

Un programa concurrente puede ser ejecutado por:

- **Multiprogramación:** los procesos comparten uno o más Procesadores
- **Multiprocesamiento:** cada proceso corre en su propio procesador pero con memoria compartida
- **Procesamiento Distribuido:** cada proceso corre en su propio procesador conectado a los otros a través de una red



# CONCEPTOS Y DEFINICIONES

## ***CARACTERISTICAS DE LOS PROGRAMAS CONCURRENTES***

- Son programas que tienen múltiples líneas de flujo de control.
- Las sentencias de un programa concurrente se ejecutan de acuerdo con un orden no estricto.
- La secuencialización de un programa concurrente es entre hitos o puntos de sincronización.
- Un programa concurrente se suele concebir como un conjunto de procesos que colaboran y compiten entre sí.
- Para validar un programa concurrente:
  - Las operaciones se pueden validar individualmente si las variables no son actualizadas concurrentemente.
  - El resultado debe ser independiente de los tiempos de ejecución de las sentencias.
  - El resultado debe ser independiente de la plataforma en que se ejecuta.

# CONCEPTOS Y DEFINICIONES

## APLICACIONES DE LOS PROGRAMAS CONCURRENTES

- Aplicaciones clásicas:
  - Programación de sistemas multicomputadores.
  - Sistemas operativos.
  - Control y monitorización de sistemas físicos.
- Aplicaciones actuales:
  - Servicios WEB.
  - Sistemas multimedia.
  - Cálculo numérico.
  - Procesamientos entrada/salida.
  - Simulación de sistemas dinámicos.
  - Interacción operador/máquina (GUIs)
  - Tecnologías de componentes.
  - Sistemas embebidos.

# CONCEPTOS Y DEFINICIONES

## VENTAJAS DE LA PROGRAMACION CONCURRENTE

- Proporciona el modelo más simple y natural de concebir muchas aplicaciones.
- Facilita el diseño orientado a objetos de las aplicaciones, ya que los objetos reales son concurrentes.
- Hace posible compartir recursos y subsistemas complejos.
- En sistemas monoprocesadores optimiza el uso de los recursos.
- Reduce tiempos de ejecución en plataformas multiprocesadoras.
- Facilita la realización de programas fiables replicando componentes y/o procesos.

# CLASIFICACIONES ARQUITECTURALES

Para la concepción de sistemas concurrentes y sus paradigmas de programación, hay que tener en cuenta que tanto en el caso distribuido, como en el paralelo los sistemas disponen de múltiples procesadores, que son capaces de trabajar conjuntamente en una o varias tareas para resolver un problema computacional.

Existen muchas formas diferentes de construir y clasificar en diferentes taxonomías los sistemas distribuidos y paralelos.

Presentaremos varias clasificaciones basadas en diferentes conceptos.

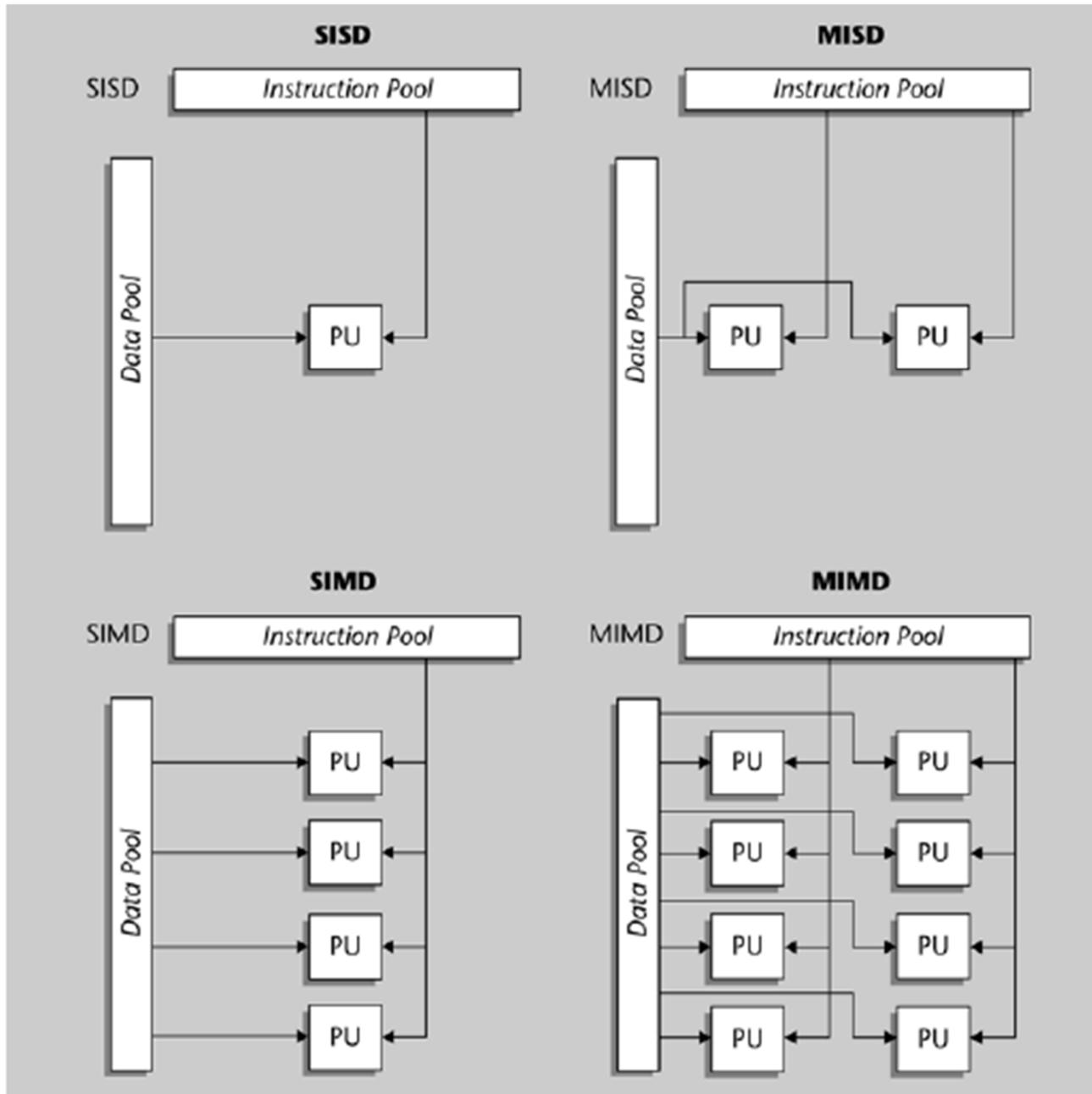
# CLASIFICACIONES ARQUITECTURALES

## Taxonomía de Flynn

Ésta es una clasificación de sistemas arquitecturales muy usada en paralelismo y en cómputo científico, que nos sirve para identificar tipos de aplicaciones y sistemas en función de los flujos de instrucciones (ya sean *threads*, procesos o tareas) y los recursos disponibles para ejecutarlos.

S I S D <b>Single Instruction Stream Single Data Stream</b>	S I M D <b>Single Instruction Stream Multiple Data Stream</b>
M I S D <b>Multiple Instruction Stream Single Data Stream</b>	M I M D <b>Multiple Instruction Stream Multiple Data Stream</b>

# CLASIFICACIONES ARQUITECTURALES



# CLASIFICACIONES ARQUITECTURALES

En función de los conjuntos de instrucciones y datos, y frente a la arquitectura secuencial que denominaríamos ***SISD (single instruction single data)***, podemos clasificar las diferentes arquitecturas paralelas (o distribuidas) en diferentes grupos:

- ***SISD (single instruction single data)***
- ***SIMD (single instruction multiple data)***,
- ***MISD (multiple instruction single data)*** y,
- ***MIMD (multiple instruction multiple data)***,
  - ***SPMD (single program multiple data)***.
  - ***MIMD Shared Memory (SMP o MPP)***.
    - ***UMA***
    - ***NUMA***
  - ***MIMD Distributed Memory (Paso de Mensajes y MPP)***.

# CLASIFICACIONES ARQUITECTURALES

**SIMD:** un solo flujo de instrucciones es aplicado a múltiples conjuntos de datos de forma concurrente. Unos procesos homogéneos (con el mismo código) sincrónicamente ejecutan la misma instrucción sobre sus datos, o bien la misma operación se aplica sobre unos vectores de tamaño fijo o variable. *El modelo es válido para procesamientos matriciales y vectoriales, siendo las máquinas paralelas con procesadores vectoriales un ejemplo de esta categoría.*

**MISD:** el mismo conjunto de datos se trata de forma diferente por los procesadores. Son útiles en casos donde sobre el mismo conjunto de datos se deban realizar muchas operaciones diferentes. *En la práctica no se han construido máquinas de este tipo por las dificultades en su concepción.*

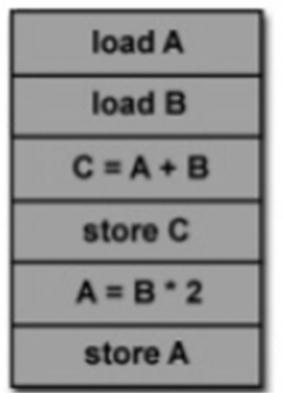
# CLASIFICACIONES ARQUITECTURALES

**MIMD:** paralelismo funcional y/o de datos. No sólo distribuimos datos, sino también las tareas a realizar entre los diferentes procesadores/nodos. Varios flujos (posiblemente diferentes) de ejecución son aplicados a diferentes conjuntos de datos. Esta categoría no es muy concreta, ya que existe una gran variedad de arquitecturas posibles con estas características, incluyendo máquinas con varios procesadores vectoriales o sistemas de centenares de procesadores o bien con unos pocos.

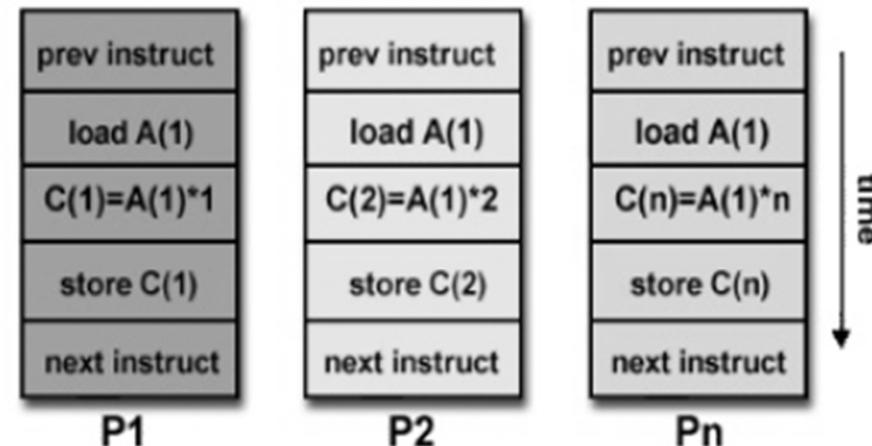
**SPMD** (variante de MIMD): en paralelismo de datos, utilizamos mismo código con distribución de datos. Hacemos varias instancias de las mismas tareas, cada uno ejecutando el código de forma independiente. SPMD puede verse como una extensión de SIMD o bien una restricción del MIMD. A veces suele tratarse más como un paradigma de programación, en el cual el mismo código es ejecutado por todos los procesadores (u nodos) del sistema, pero en la ejecución se pueden seguir diferentes caminos en los diferentes procesadores.

# CLASIFICACIONES ARQUITECTURALES

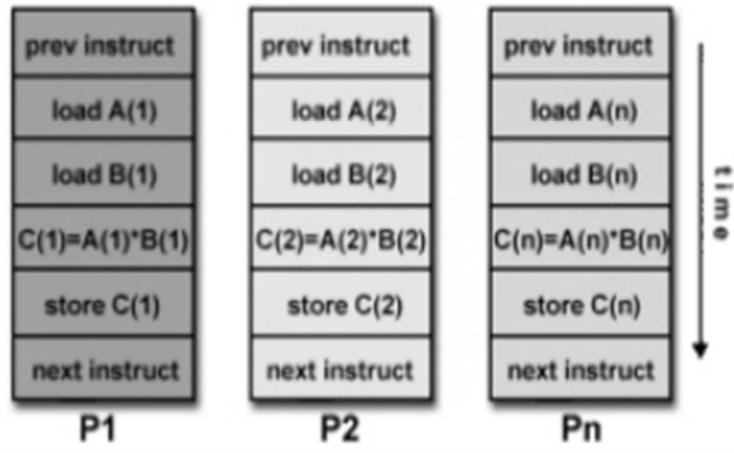
SISD: A serial computer



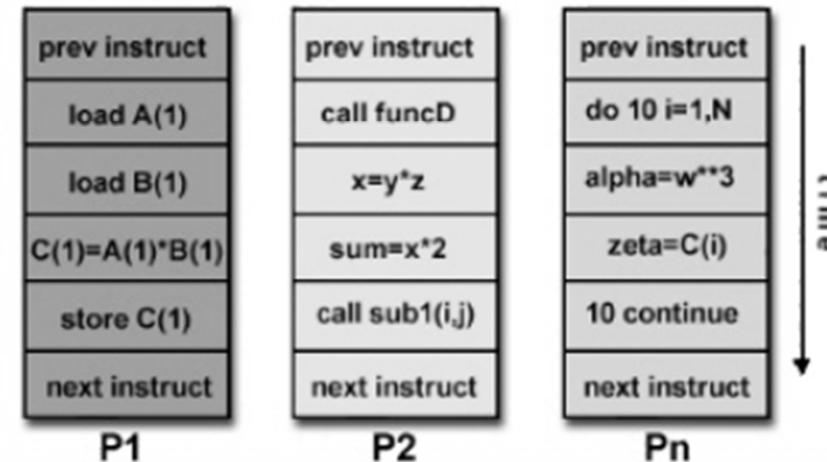
MISD: Cryptographic Decoding



SIMD: GPUs



MIMD: Clusters, Supercomputers



# CLASIFICACIONES ARQUITECTURALES

## Por control y comunicación (Variantes MIMD)

Otra extensión de las clasificaciones anteriores de los modelos arquitecturales es la basada en los mecanismos de control y la organización del espacio de direcciones en el caso de las arquitecturas MIMD, las de más amplia repercusión hoy en día (y que se ajusta bien a diferentes sistemas distribuidos):

1. *MIMD con memoria compartida (shared memory, MIMD)*
2. *Sistemas MIMD con memoria distribuida (distributed memory, MIMD)*

# CLASIFICACIONES ARQUITECTURALES

1. ***MIMD con memoria compartida o multiprocesadores:*** En este modelo, los procesadores comparten el acceso a una memoria común. Existe un único espacio de direcciones de memoria para todo el sistema, accesible a todos los procesadores. Los procesadores se comunican a través de esta memoria compartida. Los procesadores tienen el mismo espacio de acceso a los mismos.

Como ejemplo, se destaca a los ***sistemas SMP (symmetric multiprocessing)***, refiriéndose a la combinación de MIMD y memoria (físicamente) compartida. La limitación principal es el número de procesadores dependiendo de la red de interconexión, siendo habitual sistemas entre 2 a 16 procesadores, incluidos en una misma máquina. Aunque también existe la opción, de unir algunos de estos sistemas (formando un sistema híbrido), en lo que se conoce como ***SPP (scalable parallel processing)*** o ***CLUMP (clusters of multiprocessors)***, para formar un sistema mayor.

# CLASIFICACIONES ARQUITECTURALES

Pueden ser de dos tipos:

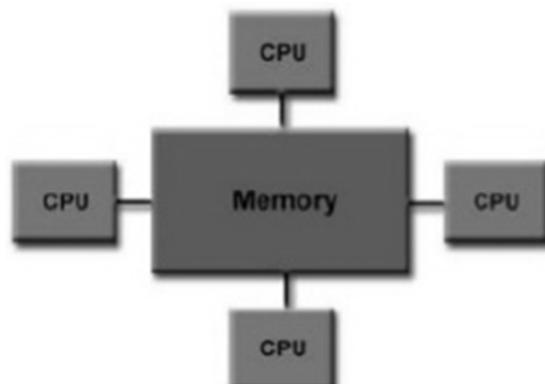
*Sistema fuertemente acoplado*, el sistema ofrece un mismo tiempo de acceso a memoria para cada procesador. Este sistema puede ser implementado a través de un gran módulo único de memoria, o por un conjunto de módulos de memoria de forma que se pueda acceder a ellos en paralelo por los diferentes procesadores. **Tenemos un acceso uniforme a memoria (UMA).**

*Sistema ligeramente acoplado*, el sistema de memoria está repartido entre los procesadores, disponiendo cada uno de su memoria local. Cada procesador puede acceder a su memoria local y a la de los otros procesadores, pero el tiempo de acceso a las memorias no locales es superior a la de la local, y no necesariamente igual en todas. **Tenemos un acceso no uniforme a la memoria (NUMA).**

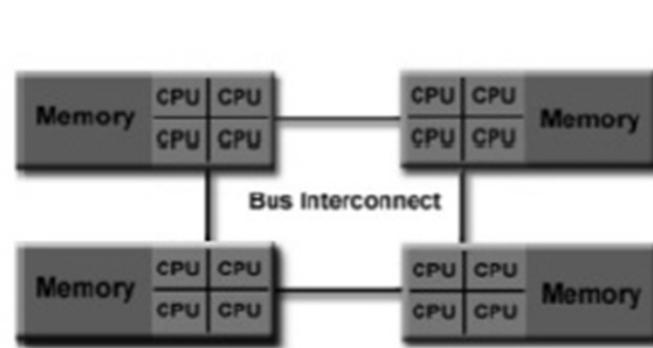
# CLASIFICACIONES ARQUITECTURALES

Estos sistemas pueden tener distinta arquitectura, pero tienen en común que cada uno de sus procesadores pueden acceder a toda la memoria del sistema a través de un espacio de direcciones global

- ▶ Múltiples procesadores pueden funcionar de forma independiente, pero comparten los mismos recursos de memoria
- ▶ Los cambios en una ubicación de memoria efectuado por un procesador son visibles para todos los demás procesadores
- ▶ Las máquinas de memoria compartida se clasifican como UMA y NUMA, basándose en los tiempos de acceso a memoria



Shared Memory (UMA)



Shared Memory (NUMA)

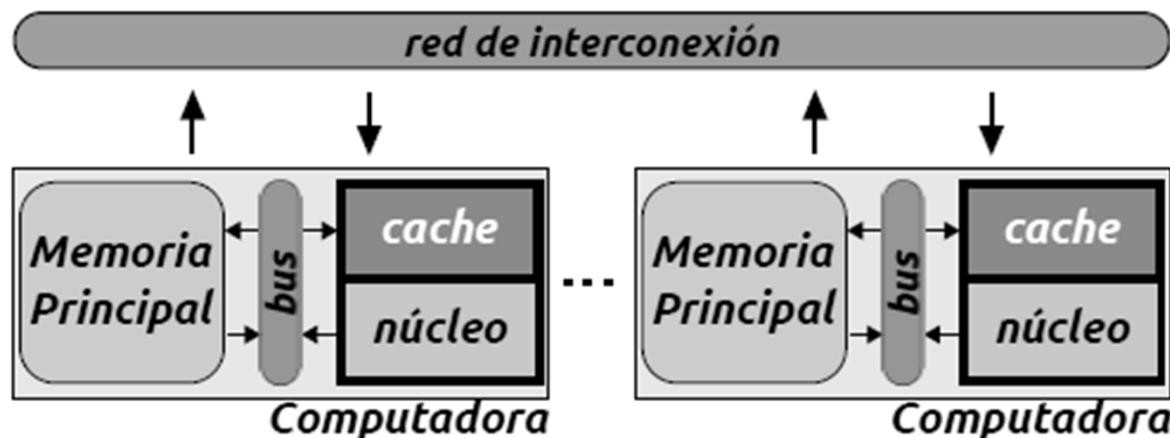
# CLASIFICACIONES ARQUITECTURALES

2. *Sistemas MIMD con memoria distribuida (distributed memory, MIMD)*, o también llamados multicomputadores: En este modelo, cada procesador ejecuta un conjunto separado de instrucciones sobre sus propios datos locales. La memoria no centralizada está distribuida entre los procesadores (o nodos) del sistema, cada uno con su propia memoria local, en la que poseen su propio programa y los datos asociados. El espacio de direcciones de memoria no está compartido entre los procesadores. Una red de interconexión conecta los procesadores (y sus memorias locales), mediante enlaces (*links*) de comunicación, usados para el intercambio de mensajes entre los procesadores. Los procesadores intercambian datos entre sus memorias cuando se pide el valor de variables remotas.

# CLASIFICACIONES ARQUITECTURALES

También son conocidas como máquinas de paso de mensajes:

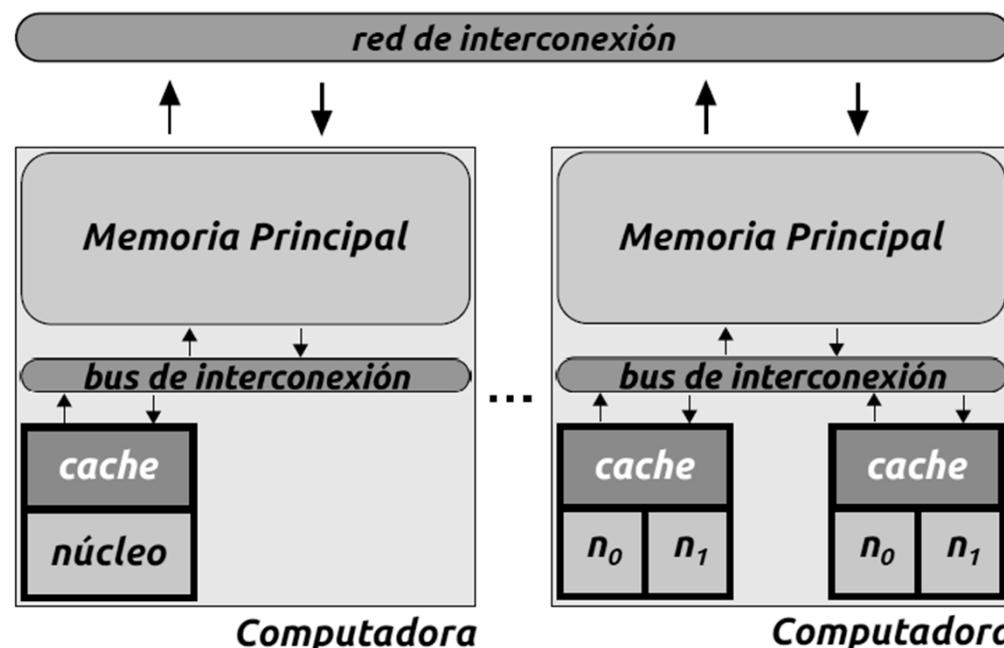
- ▶ Los procesadores tienen su propia memoria local y no hay un espacio de direcciones global
- ▶ Se requiere una red de interconexión para comunicar a los procesadores. Hay varias tecnologías de red disponibles y distintas topologías a considerar.
- ▶ Normalmente el programador es responsable de definir **cuándo, cómo y qué** datos **enviar y recibir** desde un procesador a otro. La sincronización entre tareas es también responsabilidad del programador.
- ▶ No se aplica el concepto de coherencia de caché ya que las memorias son locales



# CLASIFICACIONES ARQUITECTURALES

Una última variante MIMD con memoria físicamente distribuida y red de interconexión dedicada, especialmente diseñada, son los *sistemas MPP* (*massive parallel processing*). En éstos, el número de procesadores puede variar desde unos pocos a miles de ellos. Normalmente, los procesadores están organizados formando una cierta topología (tanto física como lógica, como: anillo, árbol, malla, hipercubo, etc.), disponiendo de un red de interconexión entre ellos de baja latencia y gran ancho de banda.

Esquema híbrido



# CARACTERISTICAS DESEADAS DE UNA APLICACIÓN PARALELA

- Alto Rendimiento
- Eficiencia
- Escalabilidad
- Transparencia
- Portabilidad
- Tolerante a fallos



# **Alto Rendimiento**

Se intenta reducir el tiempo de ejecución de los programas:

- La mejor solución puede diferir totalmente de la sugerida por los algoritmos secuenciales existentes.
- Los aspectos independientes de la máquina tales como la concurrencia deben ser considerados tempranamente, y los aspectos específicos de la máquina se deben demorar.
- Algunos temas pueden ser manejados automáticamente por las herramientas de especificación/diseño o por el S.O.



## **Metodología de diseño de algoritmos paralelos**



Enfoque metódico para maximizar el rango de opciones consideradas, brindar mecanismos para evaluar las alternativas, y reducir el costo de backtracking por malas elecciones.

# Eficiencia

Se considera con respecto a:

- **Uso de recursos de cómputo:** fracción de tiempo en que los procesadores están siendo utilizados.
- **Energía:** potencia y energía (estática y dinámica) que se utilizan.



Indica qué tan bien se utilizan los recursos.

# Escalabilidad

Es la capacidad de mantener/incrementar la eficiencia del programa en proporción al tamaño del problema (número de elementos de procesamiento).



Se debe considerar el efecto de incrementar el número de procesadores, la potencia de cada procesador y/o el volumen de datos a procesar.

# Transparencia

Significa ocultar al usuario/programador detalles que dan funcionalidad al sistema. Puede ser con respecto:

- a la **Ubicación** → no distinguir entre recursos locales o remotos.
- a la **Concurrencia** → El usuario no “ve” la competencia y sincronización por los recursos.
- al **Parallelismo y su implementación** → Partir de la especificación del problema y no detenerse en el modo de descomponerlo en tareas o de ejecutarlo en múltiples procesadores → mayor abstracción → muy difícil obtener alta eficiencia.

La transparencia acelera el proceso de producción y administración de software paralelo pero disminuye su eficiencia.

# Portabilidad

- La **portabilidad de código** se obtiene a través de estándares que permiten que el desarrollo de programas paralelos/distribuidos se ejecuten sobre una variedad de arquitecturas (ejemplos: OpenMP, MPI, Java)
- La **portabilidad de performance** es mucho más difícil de lograr → significa que en diferentes arquitecturas el programa logre un rendimiento o eficiencia similar.

**La portabilidad de performance no se consigue al mismo tiempo que la portabilidad de código.**

# Tolerancia a Fallos

Es la habilidad de recuperar el sistema desde fallos de sus componentes sin perder la estabilidad del sistema.

Un problema con las comunicaciones, entre los procesadores, reinicio del sistema, etc. no debería afectar el funcionamiento del sistema.



Normalmente, implica introducir algún grado de redundancia a costo del rendimiento.

# Modelos de Programación (Paralela/Distribuida)

Son formas de estructurar un algoritmo seleccionando una técnica de descomposición y mapeo en búsqueda de minimizar interacciones entre procesos.

Para los modelos de programación, remarcamos cómo solucionan la distribución de código y la interconexión entre las unidades de ejecución y las tareas. Cualquiera de estos modelos de programación puede ser usado para implementar los paradigmas de programación.

Pero las prestaciones de cada combinación resultante (paradigma en un determinado modelo) dependerán del modelo de ejecución subyacente (la combinación de hardware, red de interconexión y software de sistema disponibles).

## Modelo de MEMORIA COMPARTIDA

En este modelo los programadores ven sus programas como una colección de procesos accediendo a variables locales y un conjunto de variables compartidas. Cada proceso accede a los datos compartidos mediante una lectura o escritura asíncrona. Por tanto, como más de un proceso puede realizar las operaciones de acceso a los mismos datos compartidos en el mismo tiempo, es necesario implementar mecanismos para resolver los problemas de exclusiones mutuas que se puedan plantear, mediante mecanismos de semáforos o bloqueos.

La aplicación se ve como una colección de tareas que normalmente son asignadas a *threads* de ejecución en forma asíncrona.

OpenMP es una de las implementaciones más utilizadas para programar bajo este modelo en sistemas de tipo SMP (o SH-MIMD).

# Modelo de MEMORIA COMPARTIDA

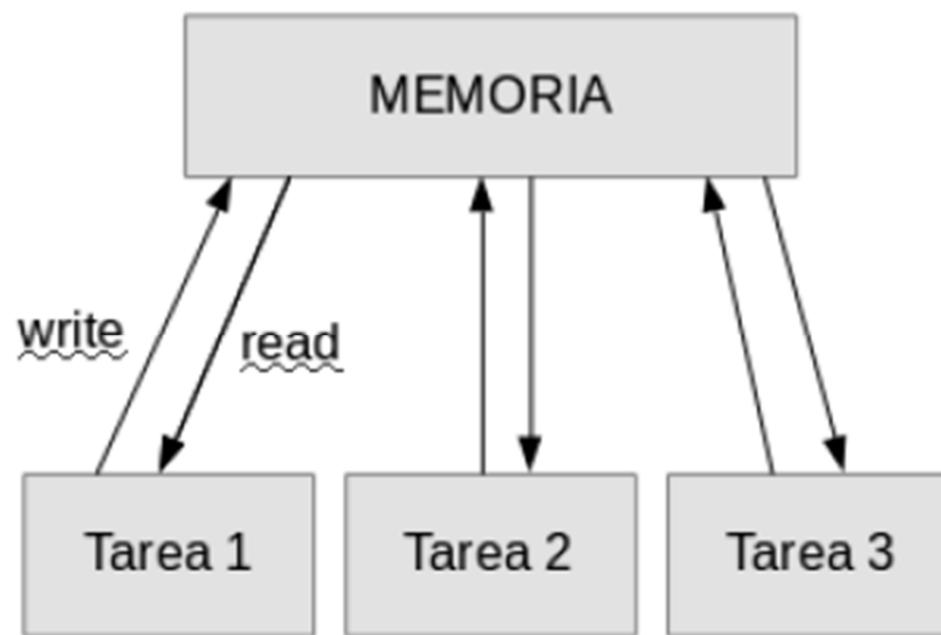
OpenMP (*open specifications for multi processing*) define directivas y primitivas de librería para controlar la paralelización de bucles y otras secciones de un código en lenguajes como Fortran, C y C++. La ejecución se basa en la creación de *thread* principal, juntamente con la creación de *threads* esclavos cuando se entra en una sección paralela. Al liberar la sección, se reasume la ejecución secuencial. OpenMP, que normalmente se implementa a partir librerías de bajo nivel de *threads*.

En OpenMP se usa un modelo de ejecución paralela denominado *fork-join*, básicamente el *thread* principal, que comienza como único proceso, realiza en un momento determinado una operación *fork* para crear una región paralela (directivas PARALLEL, END PARALLEL) de un conjunto de *threads*, que acaba mediante una sincronización por una operación *join*, reasumiéndose el *thread* principal de forma secuencial, hasta la siguiente región paralela. Todo el paralelismo de OpenMP se hace explícito mediante el uso de directivas de compilación que están integradas en el código fuente (Fortran, o C/C++).

# Modelo de MEMORIA COMPARTIDA

Todos los datos de la aplicación están en una memoria global a la que puede acceder cada tarea de forma independiente.

Se requiere sincronización para preservar la integridad de las estructuras de datos compartidas.



## **Modelo de MEMORIA DISTRIBUIDA (PASO DE MENSAJES)**

Éste es uno de los modelos más ampliamente usado. En él los programadores organizan sus programas como una colección de tareas con variables locales privadas y la habilidad de enviar, y recibir datos entre tareas por medio del intercambio de mensajes. Definiéndose así por sus dos atributos básicos: Un espacio de direcciones distribuido y soporte únicamente al paralelismo explícito.

Los mensajes pueden ser enviados vía red o usando memoria compartida si está disponible. Las comunicaciones entre dos tareas ocurren a dos bandas, donde los dos participantes tienen que invocar una operación. Podemos denominar a estas comunicaciones como operaciones cooperativas, ya que deben ser realizadas por cada proceso, el que tiene los datos y el proceso que quiere acceder a los datos. En algunas implementaciones, también pueden existir comunicaciones de tipo *one-sided*, si es sólo un proceso el que invoca la operación, colocando todos los parámetros necesarios y la sincronización se hace de forma implícita.

# **Modelo de MEMORIA DISTRIBUIDA (PASO DE MENSAJES)**

**Ventajas:** el paso de mensajes permite un enlace con el hardware existente, ya que se corresponde bien con arquitecturas que tengan una serie de procesadores conectados por una red de comunicaciones.

- En cuanto a la funcionalidad, incluye una mayor expresión disponible para los algoritmos concurrentes, proporcionando control no habitual en el paralelismo de datos, o en modelos basados en paralelismo implícito por compilador.
- En cuanto a prestaciones, especialmente en las CPU modernas, el manejo de la jerarquía de memoria es un punto a tener en cuenta; en el caso del paso de mensajes, deja al programador la capacidad de tener un control explícito sobre la localidad de los datos.

**Desventaja**, el principal problema del paso de mensajes es la responsabilidad que el modelo hace recaer en el programador.