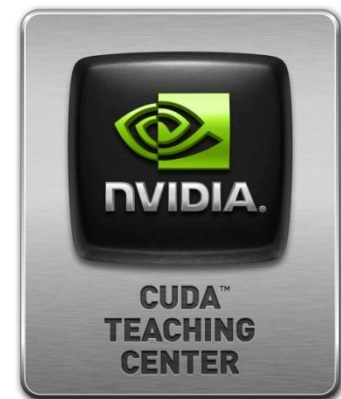


# Programación masivamente paralela en procesadores gráficos (GPUs)

E. Dufrechou, P. Ezzatti y M. Pedemonte



UNIVERSIDAD  
DE LA REPUBLICA  
URUGUAY



# Clase 2

## Computación Paralela

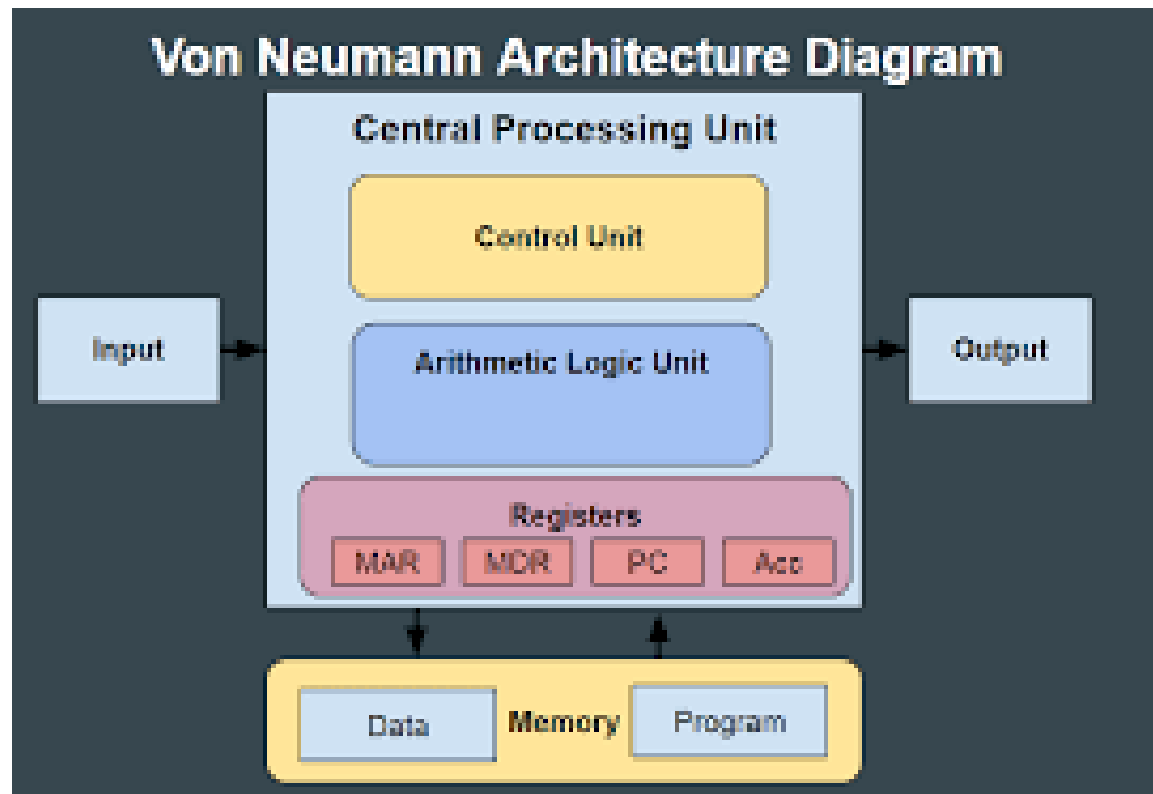
# Contenido

- **Paralelismo en máquinas secuenciales**
- **Máquinas paralelas**
- **Modelos y Estrategias para Programación Paralela**
- **Paralelismo de Memoria Compartida**
- **Paralelismo de Memoria Distribuida**

# Paralelismo en máquinas secuenciales

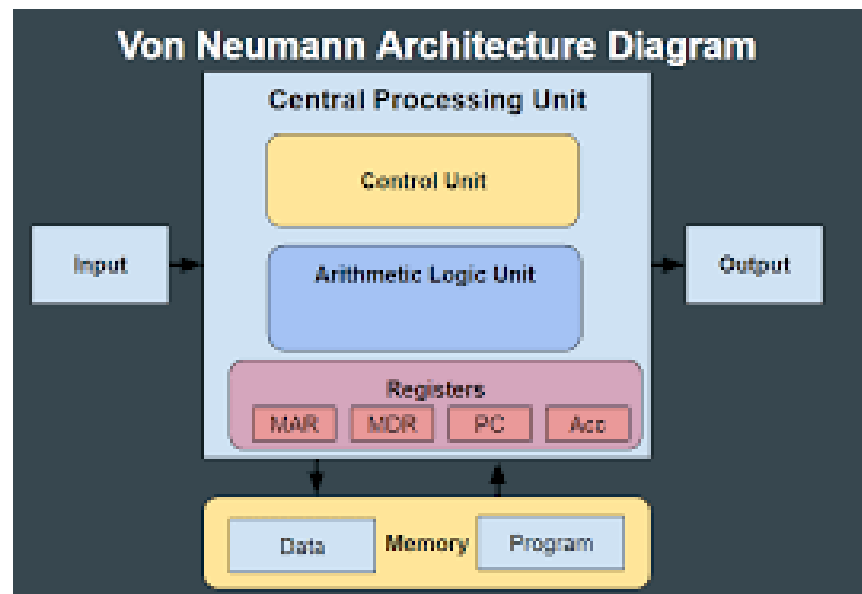
# Máquina de Von Neumann

- La arquitectura de la computadora tradicional de Von Neumann se compone de:
  - Una unidad funcional
  - Una unidad de control
  - Una única memoria.

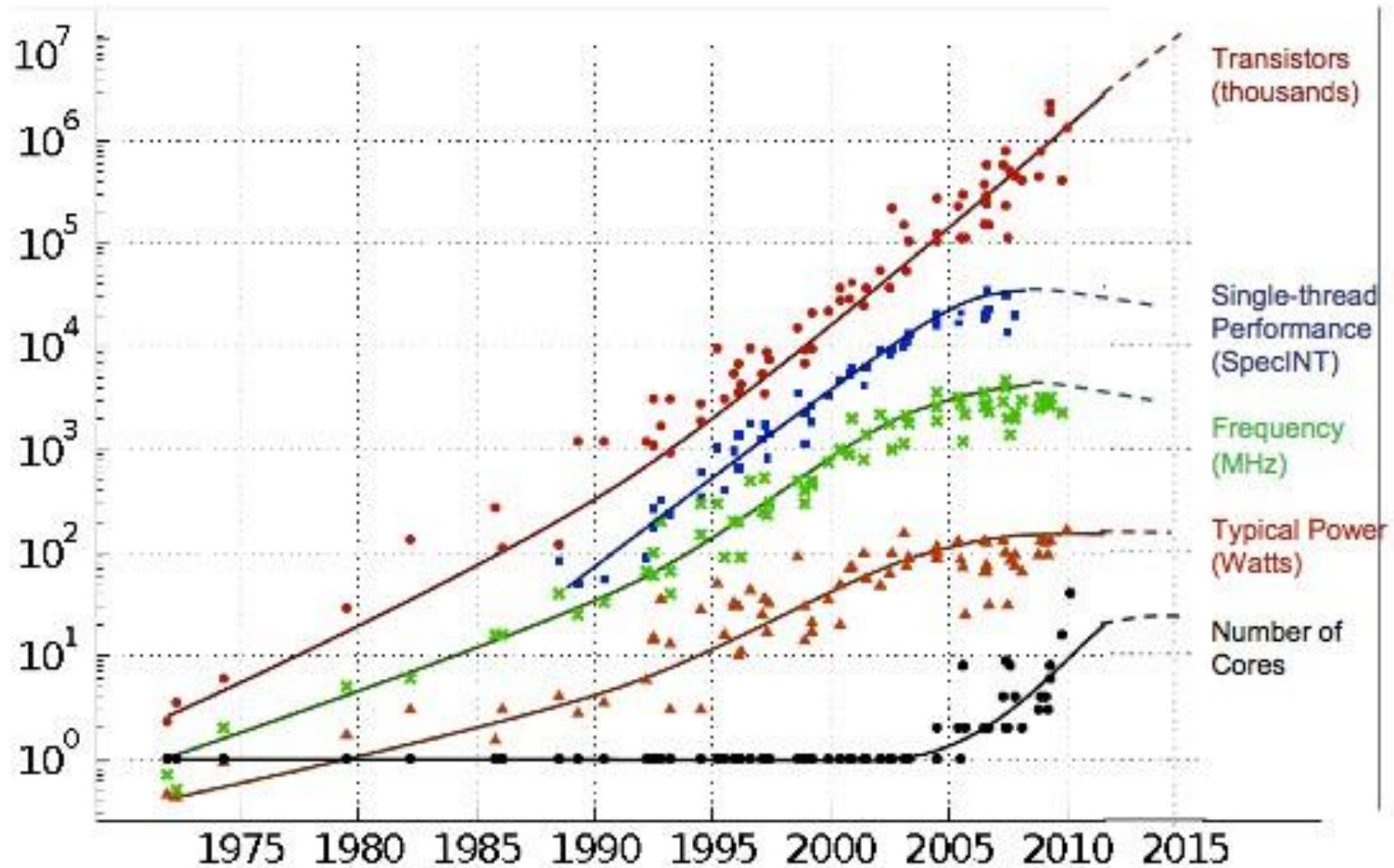


# Máquina de Von Neumann

- Realizar un cómputo implica los siguientes pasos:
  - Se trae una instrucción del programa desde la memoria y se decodifica.
  - Se calculan las direcciones de memoria de los datos requeridos.
  - Se traen los datos requeridos desde la memoria.
  - La operación es computada por la unidad funcional.
  - Los resultados son almacenados en la memoria.



# Necesidad de Paralelismo



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten  
Dotted line extrapolations by C. Moore

# Necesidad de Paralelismo

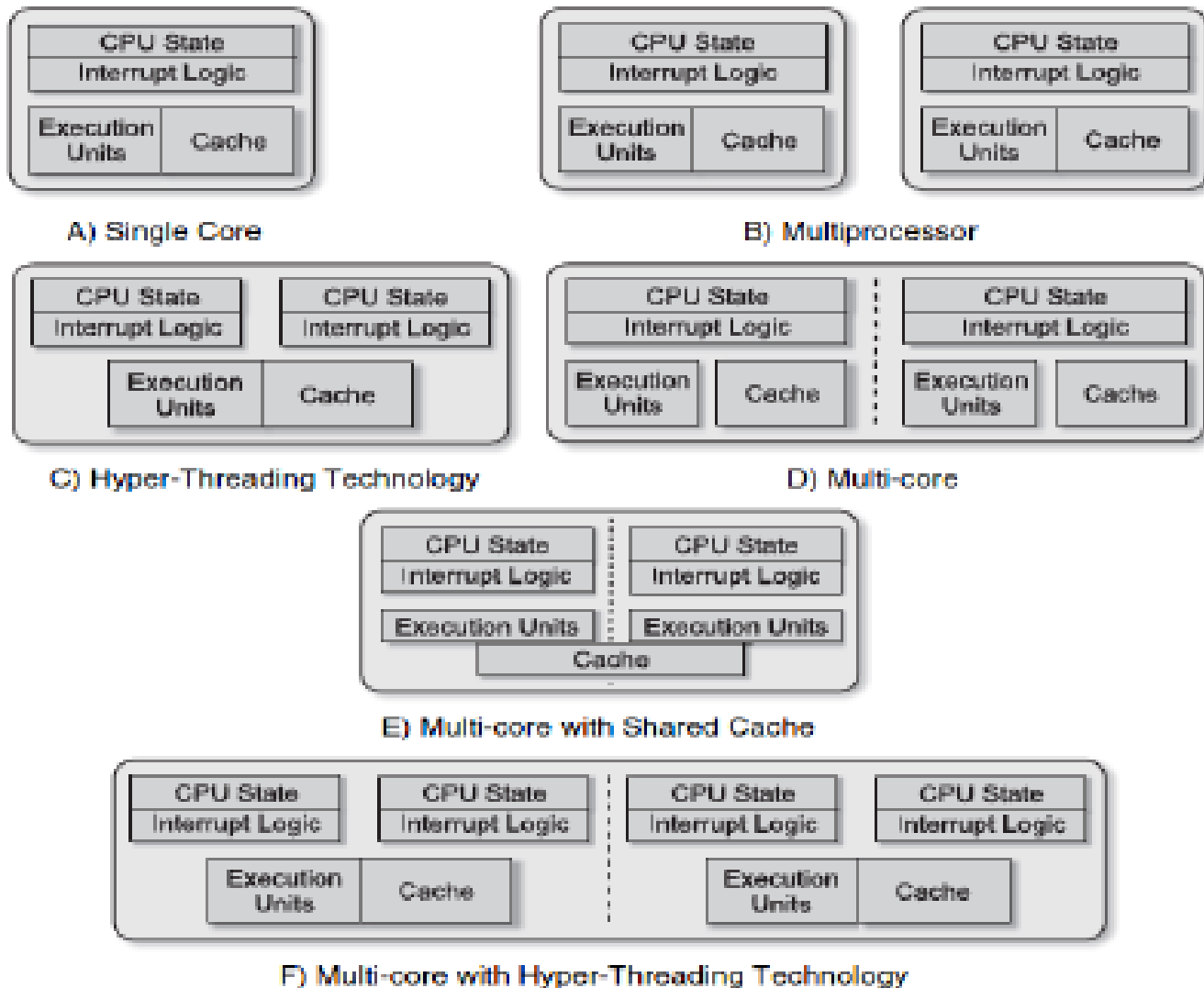
- La "Ley de Moore" se sigue cumpliendo ya que el número de transistores por chip se continúa incrementando exponencialmente.
- Sin embargo, las limitaciones térmicas impiden continuar aumentando las velocidades de relojes.
- De lo contrario, los chips comenzarían a fundirse.
- ¿Cómo se puede aumentar más el rendimiento de las aplicaciones ?
  - Aumentar el número de núcleos por chip
  - Explotar el paralelismo



# Paralelismo en Máquinas Secuenciales

- Existen diversas formas de incluir paralelismo en la arquitectura de Von Neumann:
  - 1) Incorporar múltiples unidades funcionales:
    - Multi-core: Una sola unidad de cómputo compuesta por al menos dos procesadores (con sus unidades funcionales y de control independientes)

# Paralelismo en Máquinas Secuenciales

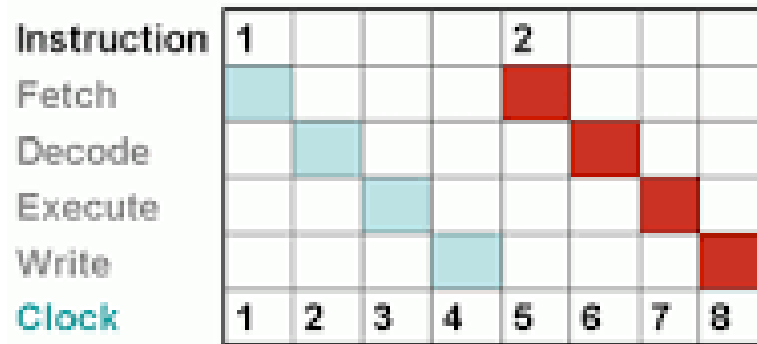


# Paralelismo en Máquinas Secuenciales

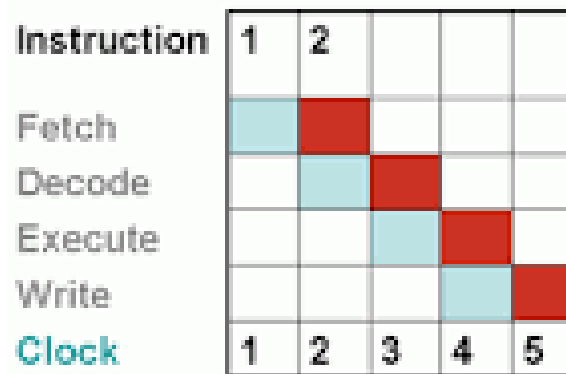
- Existen diversas formas de incluir paralelismo en la arquitectura de Von Neumann:

## 2) Pipelining:

- Las tareas se pueden dividir en sub-tareas, por lo que se definen sub-segmentos.
- Cada sub-tarea es ejecutada en un ciclo de reloj, pero diferentes sub-tareas son ejecutadas al mismo tiempo por diferentes segmentos.
- De esta forma, cuando se llena el pipeline, en cada ciclo se completa una instrucción.



Non-Pipelined



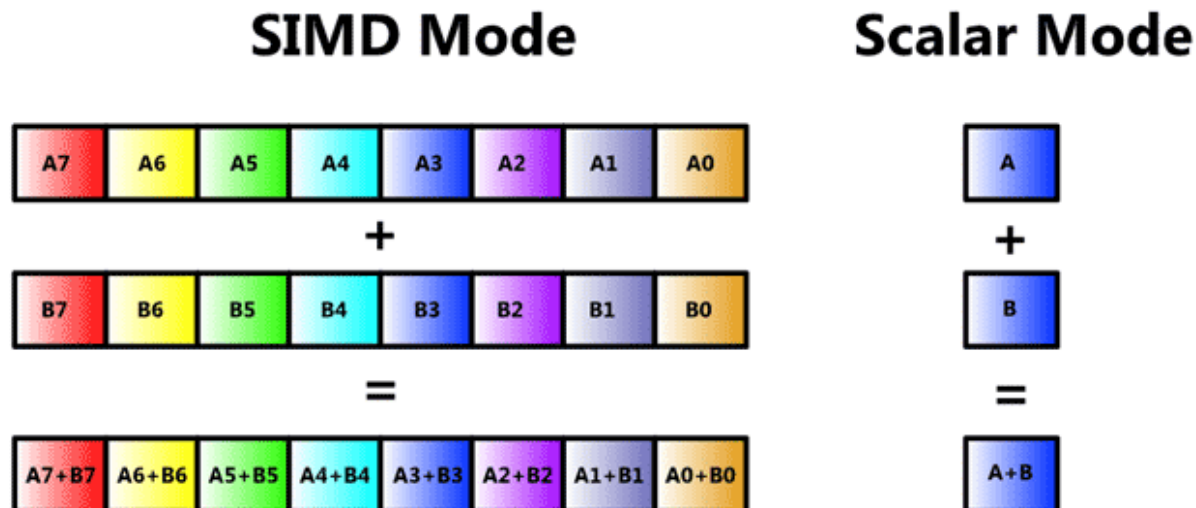
Pipelined

# Paralelismo en Máquinas Secuenciales

- Existen diversas formas de incluir paralelismo en la arquitectura de Von Neumann:

## 3) Instrucciones vectoriales:

- Se incorporan al repertorio de instrucciones operaciones que pueden operar sobre un array de datos (registros vectoriales).
- Ejemplos: las operaciones MMX, SSE y AVX de Intel.

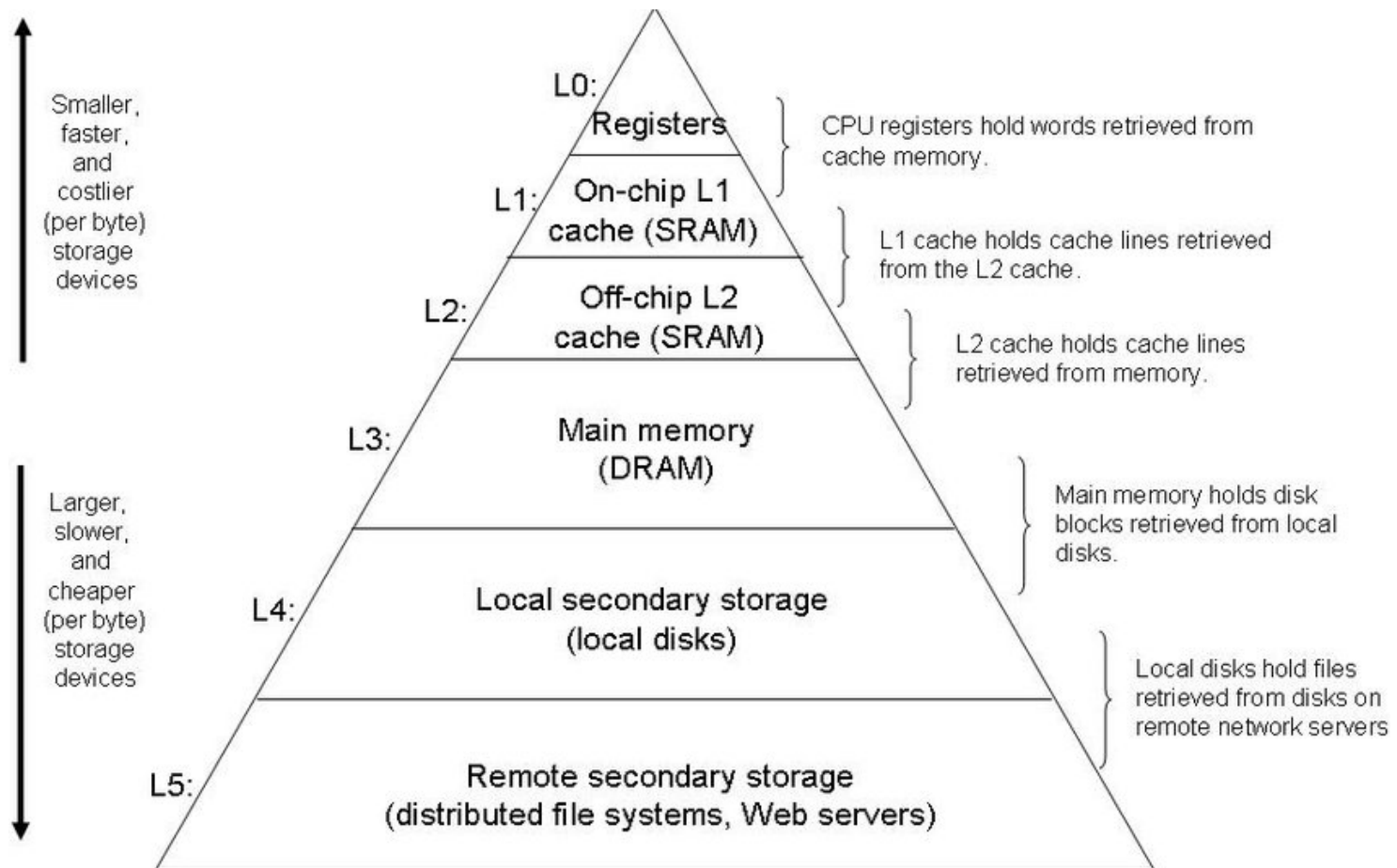


# Paralelismo en Máquinas Secuenciales

- **Existen diversas formas de incluir paralelismo en la arquitectura de Von Neumann:**
  - 4) Paralelismo a nivel de memoria:**
    - Se refiere a la capacidad de tener múltiples operaciones de memoria pendientes al mismo tiempo.
    - Es decir que un procesador puede emitir varias solicitudes de memoria al mismo tiempo para cada uno de sus núcleos.
    - Es posible realizar múltiples lecturas a la vez, así como operaciones de lectura y escritura a la vez
    - La memoria está organizada jerárquicamente.

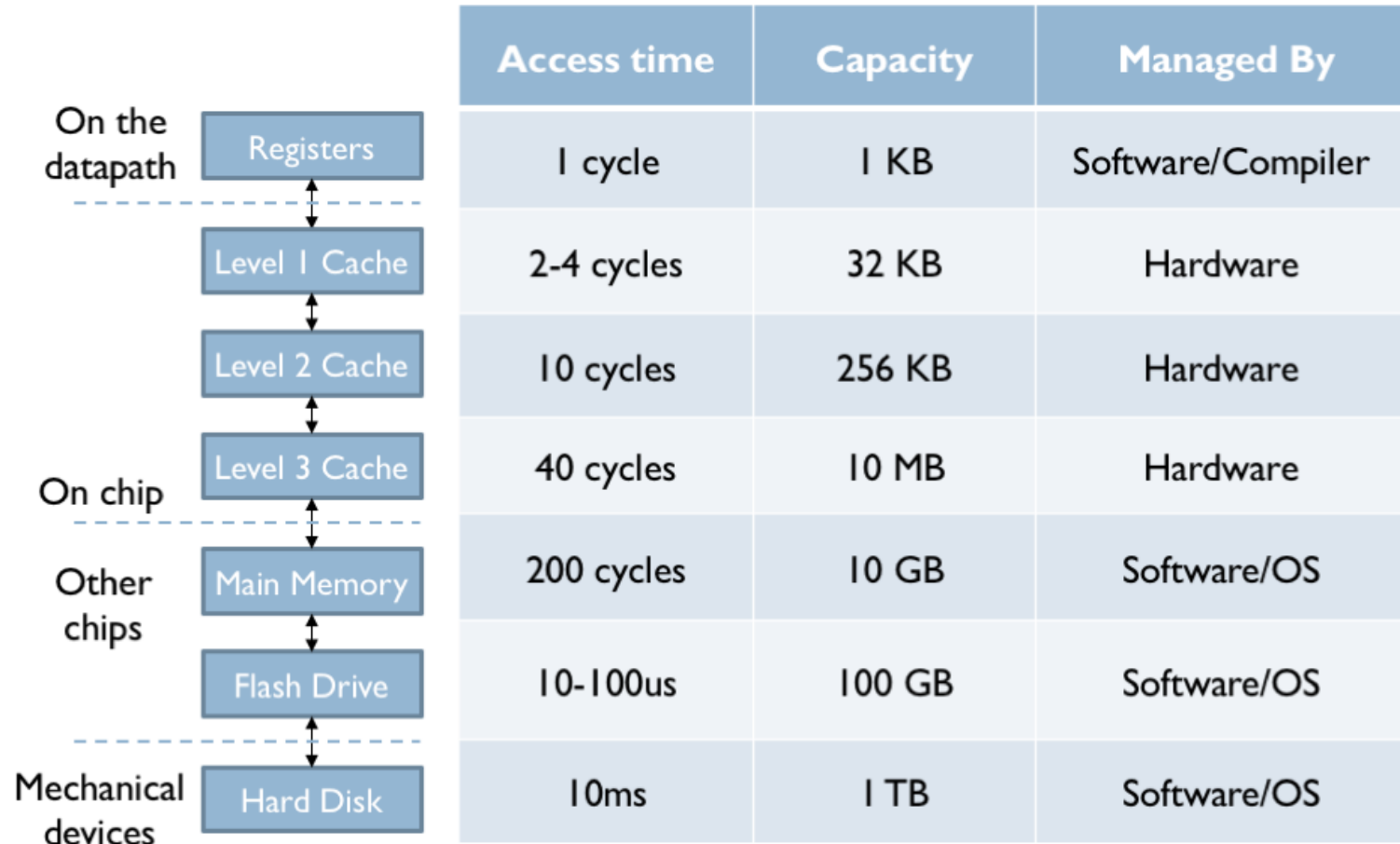
# Paralelismo en Máquinas Secuenciales

- Jerarquía de memoria:



# Paralelismo en Máquinas Secuenciales

- Jerarquía de memoria:



# Paralelismo en Máquinas Secuenciales

- **Almacenamiento en memoria:**

- La memoria es una secuencia de bytes:

1 2 3 4 5 6 7 8 9 10 11

- Los datos de los arreglos se almacenan en forma contigua.
- Los datos de las matrices dependen del lenguaje de programación:
  - C almacena por fila: 1ro los datos de la fila 1, después los datos de la fila 2 y así sucesivamente.
  - Fortran almacena por columnas: 1ros los datos de la columna 1, después los datos de la columna 2 y así sucesivamente.

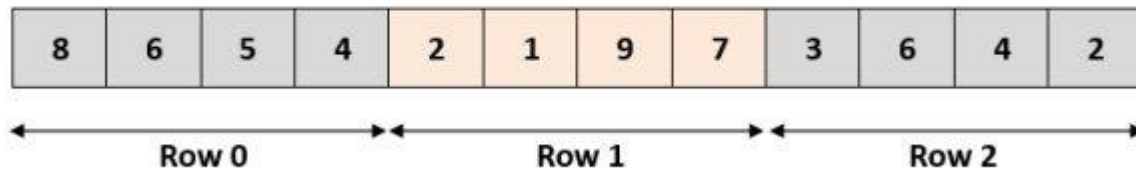


# Paralelismo en Máquinas Secuenciales

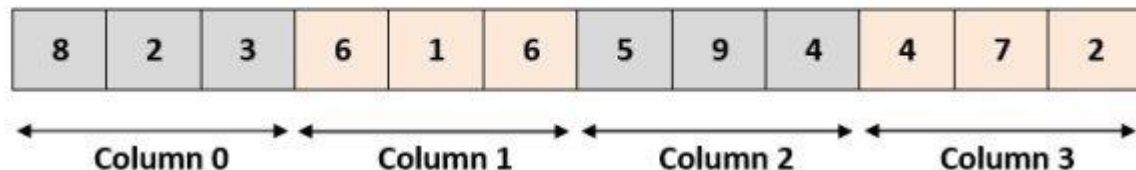
- Almacenamiento en memoria:

	0	1	2	3
0	8	6	5	4
1	2	1	9	7
2	3	6	4	2

Row-Major (Row Wise Arrangement)



Column-Major (Column Wise Arrangement)



# Paralelismo en Máquinas Secuenciales

- **Almacenamiento en memoria:**

- Por ejemplo, si se tiene una matriz de  $m$  filas y  $n$  columnas de reales (es decir que cada dato es de 4 bytes).
- La posición en bytes de la celda  $(i,j)$  en la memoria es:

Por filas – indexado desde 0:  $\text{inicio\_matriz} + (i * n) + j * 4$

Por filas – indexado desde 1:  $\text{inicio\_matriz} + (((i - 1) * n) + (j - 1)) * 4$

Por columnas – indexado desde 0:  $\text{inicio\_matriz} + (j * m) + i * 4$

Por columnas – indexado desde 1:  $\text{inicio\_matriz} + (((j - 1) * m) + (i - 1)) * 4$

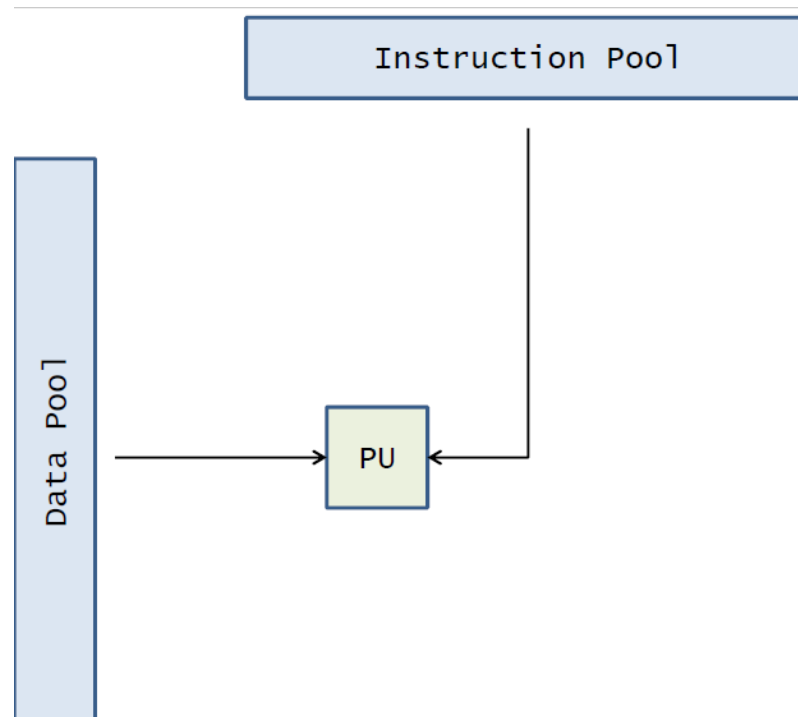
# Máquinas Paralelas

# Máquinas Paralelas

- **La taxonomía más conocida para clasificar las arquitecturas de las computadoras fue propuesta por Flynn (1966).**
- **Se basa en considerar en forma independiente los flujos de instrucciones y de datos.**
- **A partir de ello se clasifica a las computadoras en cuatro grupos: SISD, SIMD, MISD and MIMD.**

# Máquinas Paralelas

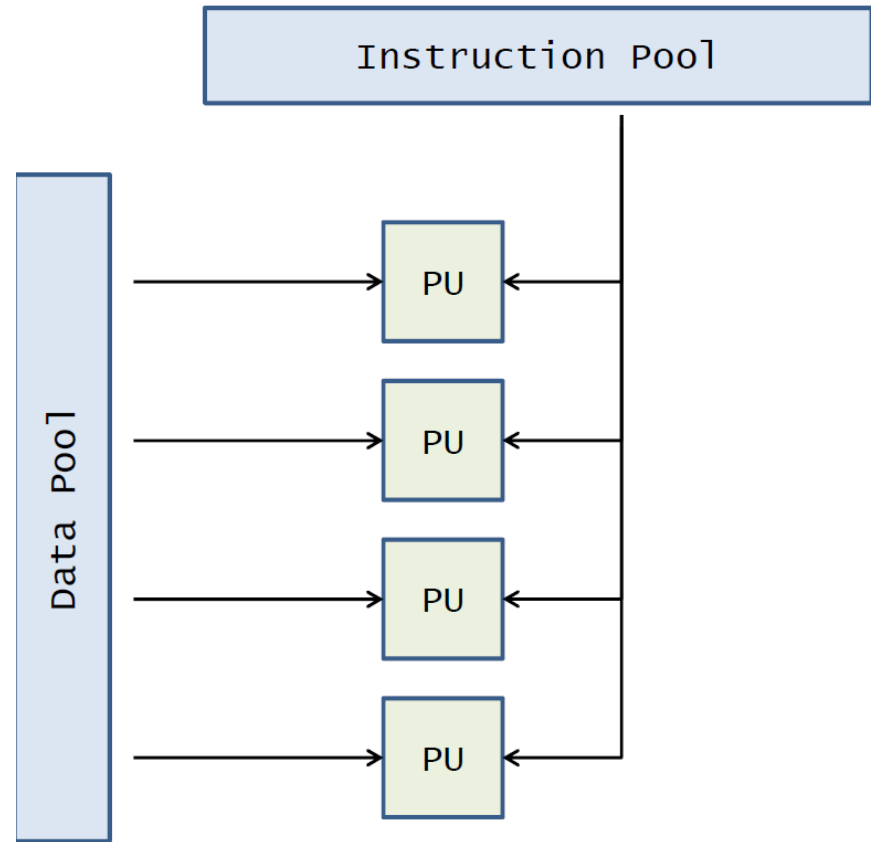
- **SISD: Single Instruction flow - Single Data flow**
  - Corresponde a la arquitectura tradicional (secuencial) de Von Neumann.



# Máquinas Paralelas

- **SIMD: Single Instruction flow - Multiple Data flow**

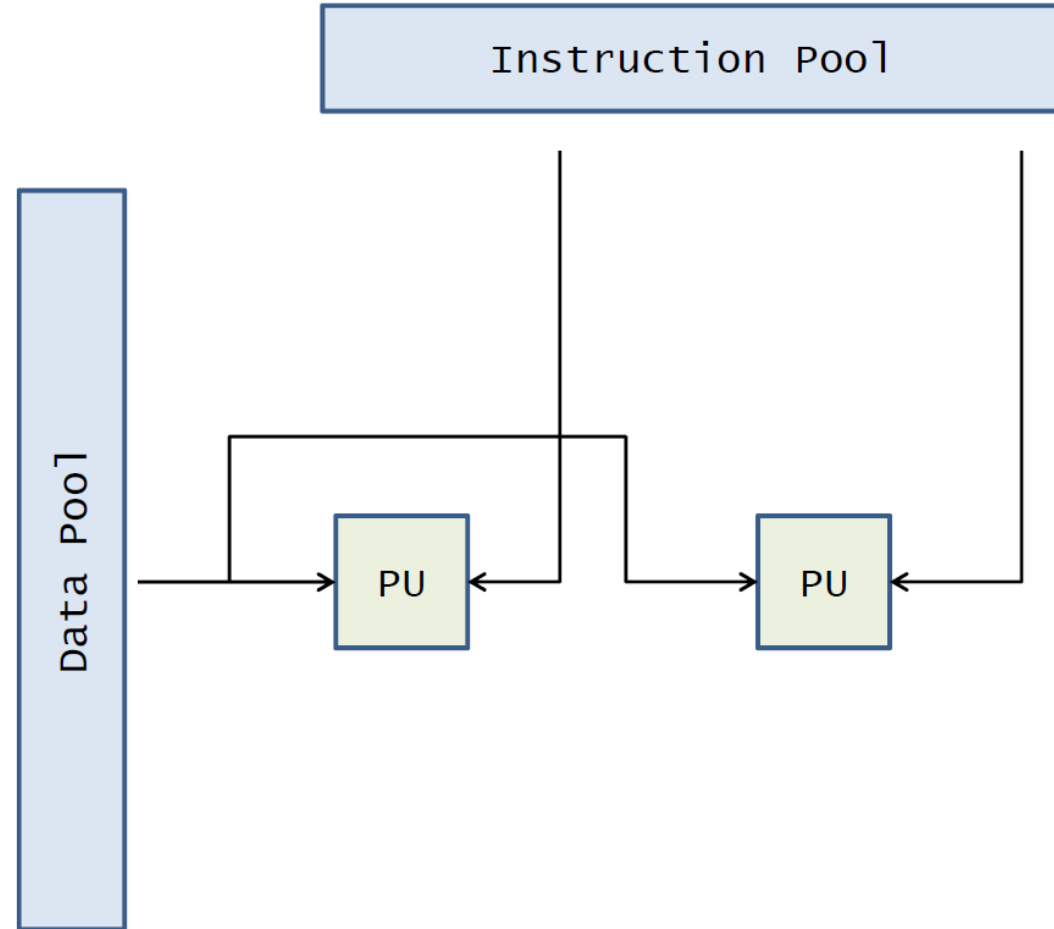
- Se caracterizan por tener múltiples unidades de procesamiento que ejecutan la misma instrucción en diferentes datos en el mismo instante de tiempo.
- Las unidades de procesamiento no tienen autonomía de procesamiento y son controlados centralmente.



# Máquinas Paralelas

- **MISD: Multiple Instruction flow - Single Data flow**

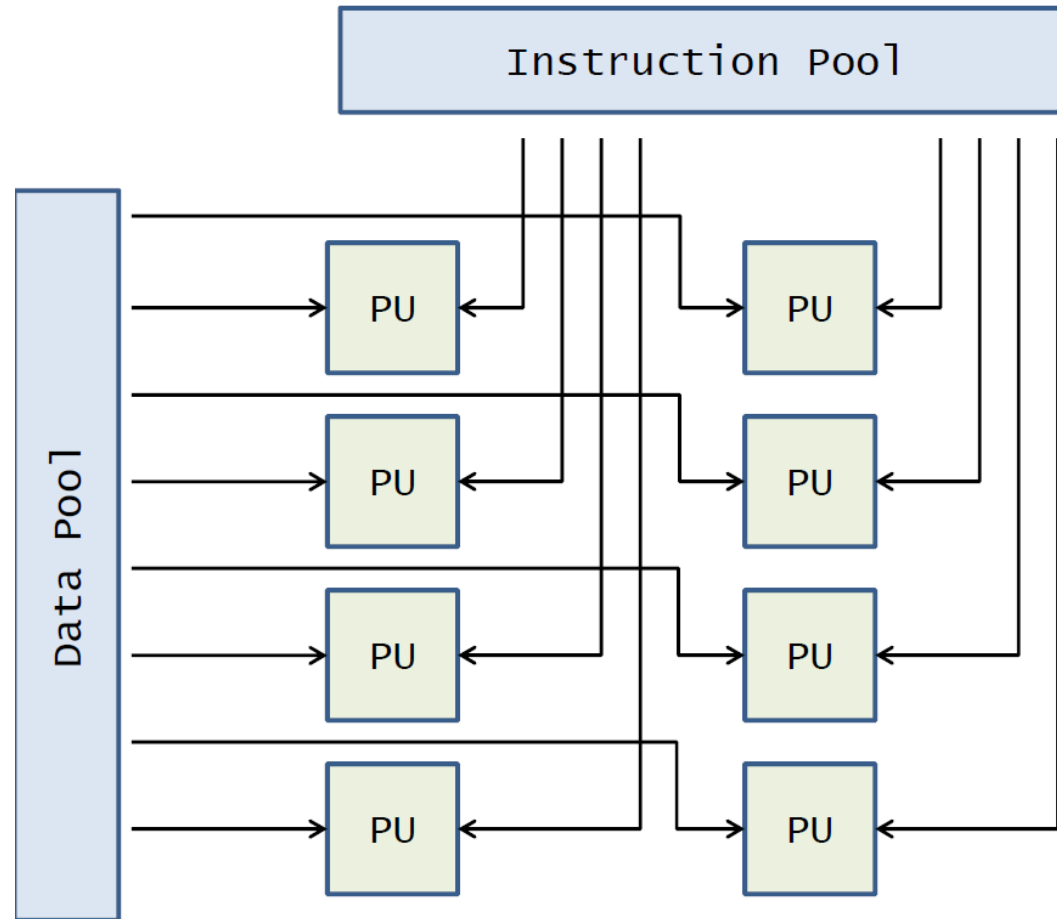
- Es una arquitectura muy poco común en la práctica.
- Sistemas a pruebas de fallos: cada unidad de procesamiento opera con los mismos datos y los resultados obtenidos por cada unidad tienen que ser los mismos.
- También los array sistólicos.



# Máquinas Paralelas

- **MIMD: Multiple Instruction flow - Multiple Data flow**

- Varias unidades de procesamiento ejecutan diferentes instrucciones en diferentes datos.
- Las unidades de procesamiento son autónomas y el control es completamente descentralizado.
- Las computadoras modernas en general caen en esta categoría.





# Modelos y Estrategias para Programación Paralela

# Niveles de Paralelismo

- **A nivel de trabajo:**
  - Manejado por el OS. Ej: Multiprogramming, multiprocessing
- **A nivel de programa:**
  - Manejado por software. Es el paralelismo a nivel de una aplicación. A su vez se puede dividir en:
    - fine grain tasks: por ejemplo a nivel de loops
    - coarse grain tasks: ejecución de partes o fracciones del código.
- **A nivel inter-instrucción:**
  - Realizado por el compilador.
  - Requiere análisis de dependencias entre instrucciones.
- **A nivel intra-instrucción:**
  - Realizado por el hardware. Ej: Pipelining, chaining, etc.

# Clasificación de los Programas Paralelos

- **Se puede clasificar de diferentes formas:**
  - **Granularidad:**
    - **Grano fino (fine grain) vs grano grueso (coarse grain)**
  - **Homogéneo – heterogéneo:**
    - **La misma tarea la ejecutan diversos procesadores sobre diferentes datos vs diferentes tareas.**
  - **Estrategia de paralelismo:**
    - **paralelismo de control, paralelismo de flujo, paralelismo de datos**
  - **Grado de programación:**
    - **paralelismo implícito, paralelismo explícito.**

# Modelos de Programación Paralela

- El desarrollo de programas paralelos está fuertemente condicionado por el hardware donde se ejecutará.
- En general se divide en dos grandes paradigmas:
  - Memoria compartida
    - Este modelo requiere memoria de acceso común a todas las unidades de procesamiento y herramientas para sincronización.
    - Un ejemplo es la programación multi-hilo.
  - Memoria distribuida (pasaje de mensajes)
    - Típicamente computadoras con acceso local a memoria (MPP, clusters, sistemas distribuidos, etc).
    - En cada procesador ejecuta un programa con sus propios datos y la comunicación y coordinación entre los procesos se realiza mediante mensajes.

# Descomposición de Problemas

- Para realizar implementaciones paralelas se sigue la estrategia de *divide and conquer* para definir sub-problemas más sencillos
- La división puede ser:
  - Descomposición funcional
    - Si la resolución del problema requiere la aplicación de diferentes funcionales independientes.
    - Cada procesador ejecuta una función diferente.
  - Descomposición de dominio
    - Se aplica cuando el problema implique un dominio donde se pueden dividir cálculos independientes.
    - Cada procesador ejecuta la misma función sobre un distinto conjunto de datos.

# Aspectos de la Descomposición de Problemas

- **Cuando hay múltiples unidades de cómputo trabajando en forma coordinada se debe tener en cuenta:**
  - Comunicación
  - Sincronización
- **Para tener un buen desempeño computacional resultan fundamentales:**
  - El scheduling de las tareas
  - La distribución de datos/cálculos
  - El control de sincronizaciones
- **Si bien parece obvio, además de buen desempeño es necesario correctitud.**
- **Sin embargo, este aspecto es más complejo de comprobar que en programas secuenciales.**

# Aspectos de la Descomposición de Problemas

- **Existen diversos criterios para optimizar la partición del cómputo:**
  - **Balance de carga: distribuir equitativamente los cálculos.**
  - **Minimizar las comunicaciones/sincronizaciones**
  - **Minimizar el ancho de banda utilizado**
  - **Minimizar la cantidad de nodos con los que me comunico**
  - **Etc.**
- **Mapeo:**
  - **¿Cómo relacionar las unidades de cómputo con los cálculos a realizar?**
  - **Son importantes aspectos como “cercanía” de unidades y localidad de datos (especialmente importante en arquitecturas multi-core).**

# Paralelismo de Memoria Compartida



# Paralelismo de Memoria Compartida

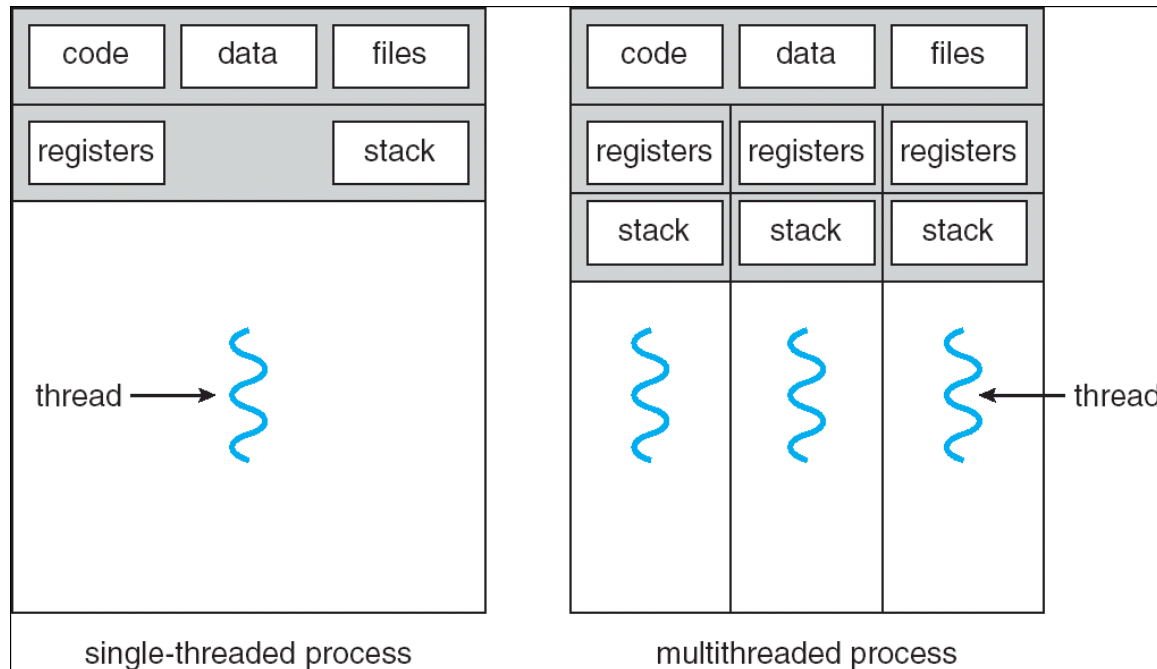
- **Todas las comunicaciones y sincronizaciones se realizan a través del recurso común (memoria).**
- **Es necesario sincronizar el acceso y garantizar la exclusión mutua de secciones compartidas.**
- **Paralelismo multithreading**
- **Bibliotecas estándares (e.g., en lenguaje C).**
- **Bibliotecas específicas.**
- **Ejemplos:**
  - **Hilos**
  - **OpenMP**

# Hilos

- **A partir de la existencia de sistemas multiprocesadores (en los años 90), surgieron sistemas operativos que permitían a un proceso del sistema manejar más de un hilo de ejecución (a través del uso de más de un contador de programa).**
- **De esta forma, se permite que un proceso ejecute en más de un procesador en forma simultánea, logrando paralelismo dentro del proceso.**

# Hilos

- Los hilos de un mismo proceso ejecutan sobre un mismo espacio de direccionamiento.
- Con el surgimiento de hardware multi-core de los últimos años, la utilización de múltiples hilos de ejecución en un mismo proceso reflató.



# Hilos

- **Los sistemas operativos modernos brindan una interfaz de programación que permite la utilización de varios hilos de ejecución dentro de un mismo proceso.**
- **Brindan herramientas básicas para el manejo de hilos: creación, destrucción y sincronización.**
- **La programación con hilos permite tener un mayor control sobre el paralelismo, pero requiere que el paralelismo sea implementado (o declarado) por el programador.**
- **Por lo tanto, requiere de un mayor conocimiento y dedicación que otras herramientas de programación de alto nivel.**

# Hilos

- **A diferencia de otros paradigmas, la programación con hilos no necesita herramientas de intercambio de información, ya sea a través de mensajes o primitivas a nivel del sistema operativo, ya que los procesos comparten el espacio de direccionamiento.**
- **Esto permite una mayor eficiencia frente a la programación con procesos que no comparten el espacio de direccionamiento.**
- **La gran desventaja del uso exclusivo de esta programación es que no escalan más allá de una máquina. Para esto se deben utilizar otros paradigmas de la programación paralela distribuida.**

# OpenMP

- **Es una API que permite aplicar paralelismo sobre memoria compartida utilizando multi-threads.**
- **Portable: C/C++ y Fortran, multiplataforma**
- **Utiliza directivas para el compilador:**
  - **Se pierde en eficiencia y flexibilidad.**
  - **Se gana en portabilidad.**

# OpenMP

- **Modelo de Ejecución paralela: fork-join:**
  - Un programa comienza su ejecución con un proceso único (hilo maestro)
  - Cuando se encuentra la primera construcción paralela (directiva) crea un conjunto de hilo
  - El trabajo se reparte entre todos los hilos incluido el maestro
  - Cuando termina la región paralela solo el hilo maestro continua la ejecución.

# OpenMP

- **Modelo de Ejecución:**
  - Los hilos se comunican a través de variables compartidas.
  - Compartir datos puede llevar a un mal comportamiento del programa debido al acceso simultáneo.
  - Para evitarlo se utilizan directivas de sincronización que protegen los datos de conflictos.
  - Las sincronizaciones son costosas, por lo que hay que tratar de evitarlas.
  - Las directivas se aplican a bloques estructurados



# Paralelismo de Memoria Distribuida

# Paralelismo de Memoria Distribuida

- En general se utiliza Message Passing Interface (MPI).
- Utiliza el paradigma de mailbox y se basa en el pasaje de mensajes.
- MPI es un estándar que funciona en una amplia variedad de arquitecturas de computación paralela.
- El estándar define la sintaxis y la semántica de un conjunto de rutinas que permiten escribir programas basados en pasaje de mensajes.
- Se usa un lenguaje secuencial estándar: Fortran, C, (F90, C++)...
- Cada proceso es un programa secuencial por separado.
- Todos los datos son privados a los procesos.
- MPI es SPMD (Single Program Multiple Data).