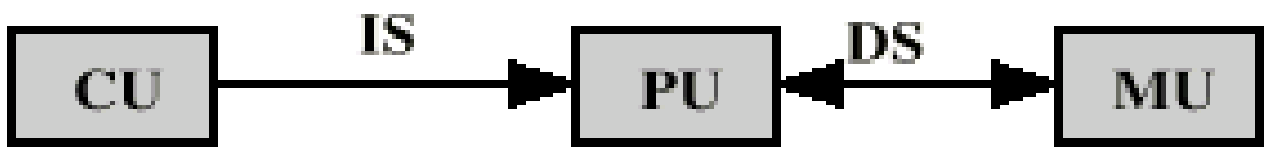
# Clasificación de Flynn de los computadores

- Single instruction, single data stream - SISD
- Single instruction, multiple data stream - SIMD
- Multiple instruction, single data stream - MISD
- Multiple instruction, multiple data stream- MIMD

# Single Instruction, Single Data Stream - SISD

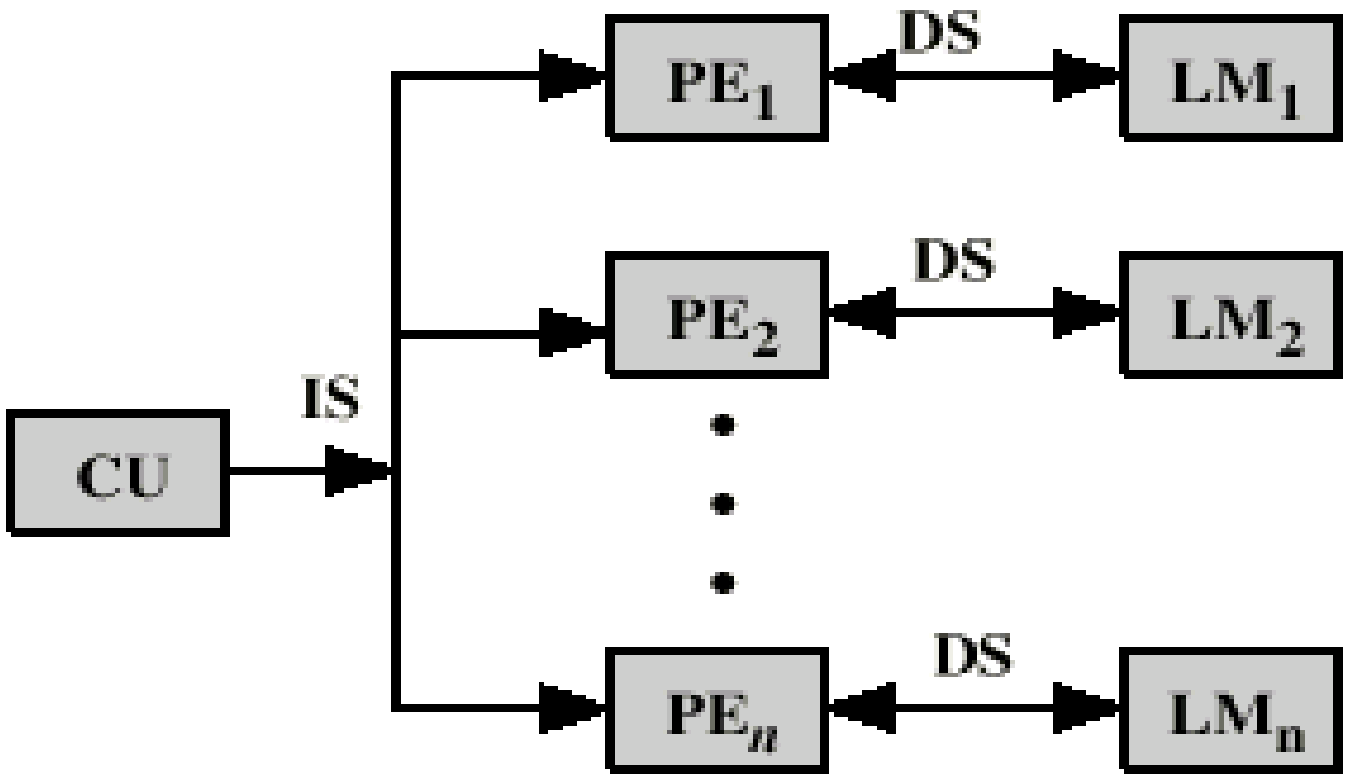
- Un único procesador (C.U. + A.L.U.)
- Un único flujo de instrucciones entre la memoria y la unidad de control
- Un único flujo de datos entre la memoria y la unidad procesadora (ALU o PU)



# Single Instruction, Multiple Data Stream - SIMD

- Una única unidad de control
  => "Single instruction stream"
- Varios elementos de proceso (ALU)
  => "Multiple data stream"
- Los elementos de proceso funcionan síncronamente
- Cada elemento de proceso tiene su propia memoria local de datos
- Procesadores matriciales

# Organización SIMD



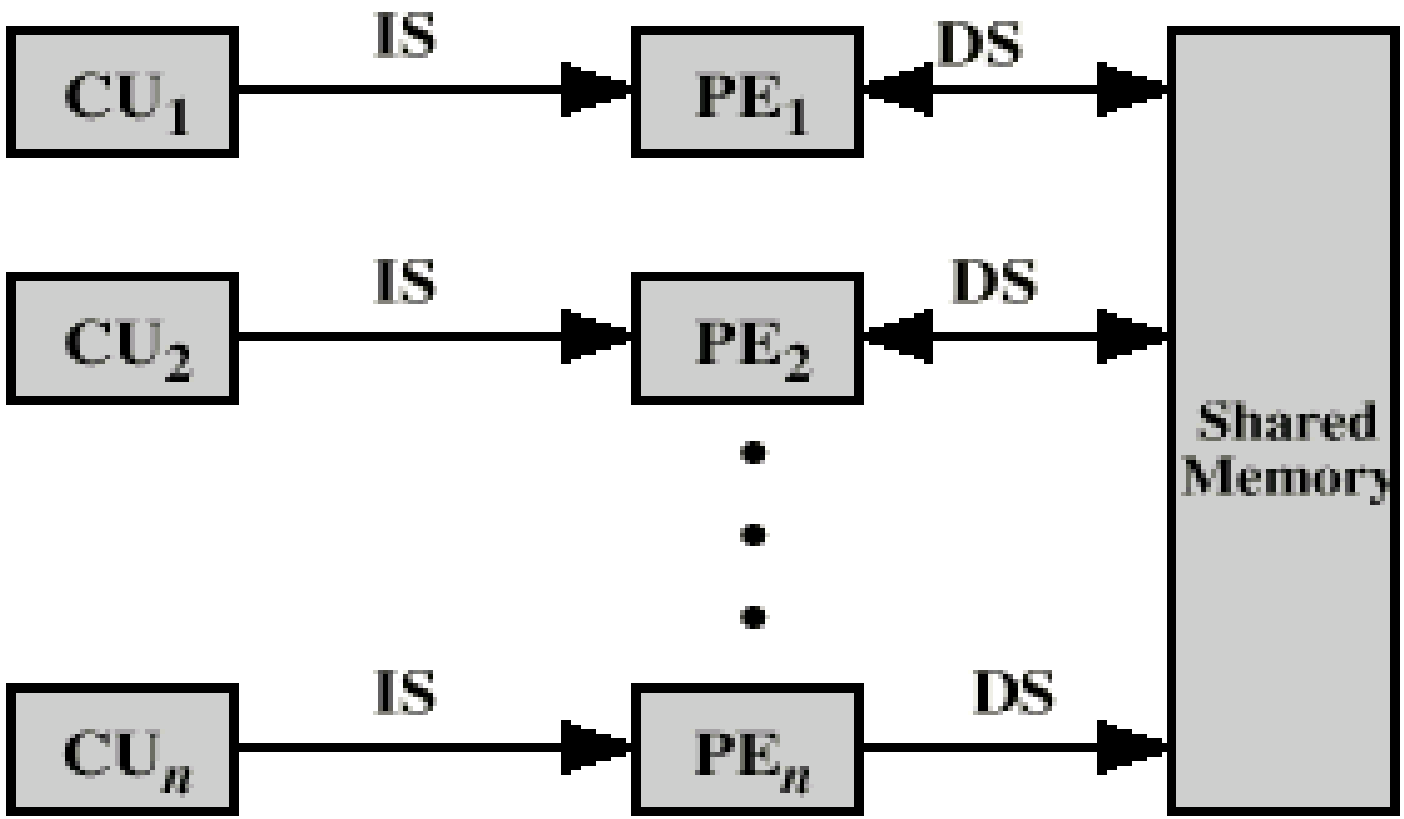
# Multiple Instruction, Single Data Stream - MISD

- Una única secuencia de datos es transmitida a través de una serie de elementos de proceso
- Cada procesador ejecuta una secuencia de instrucciones diferente sobre un mismo flujo de datos
- Algunos incluyen a los procesadores sistólicos en esta categoría
- Otros opinan que no se ha construido ninguna computadora con esta arquitectura

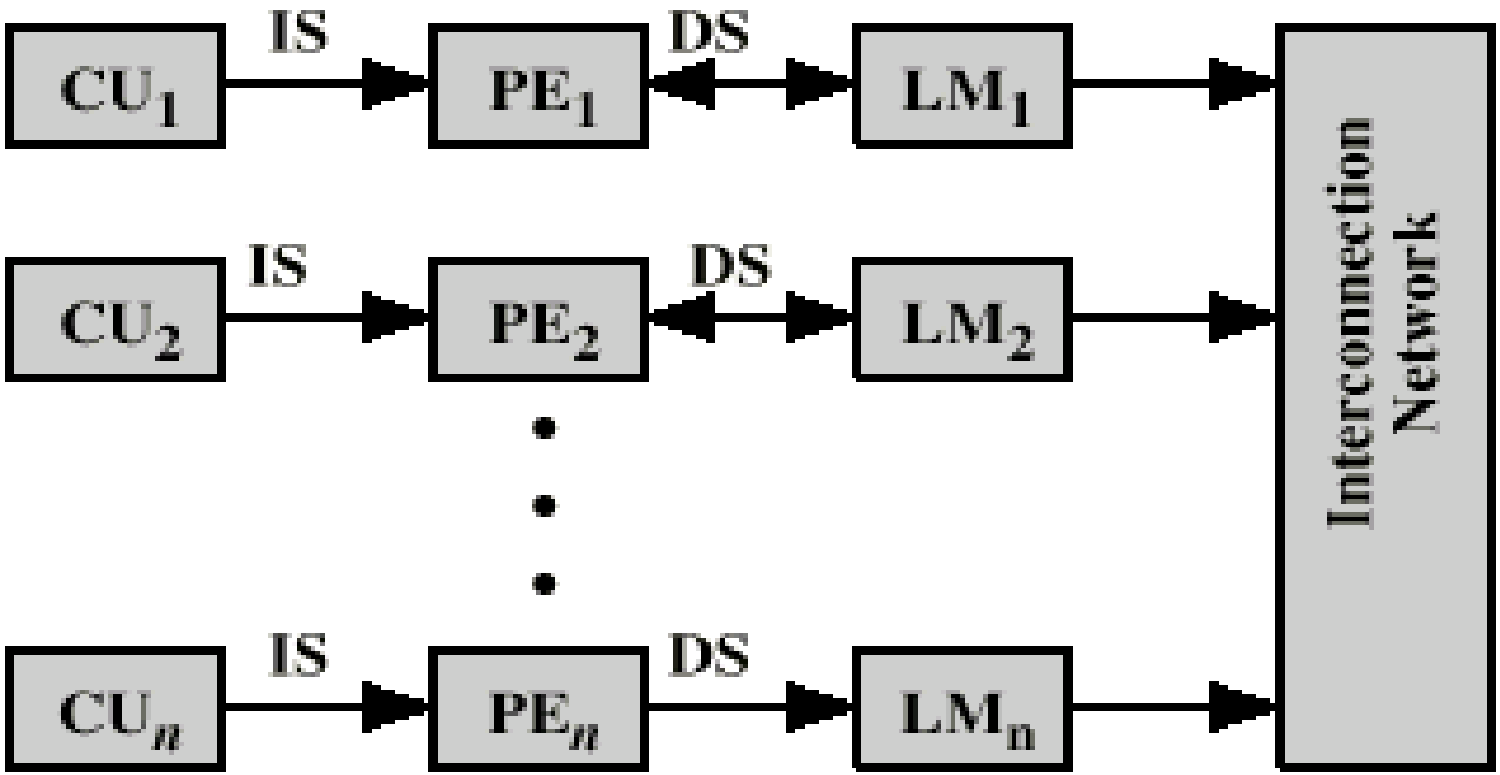
# Multiple Instruction, Multiple Data Stream- MIMD

- Varios procesadores
- Si
  - Memoria compartida entonces “multiprocesadores”
  - Memoria distribuida entonces “multicomputadores”

# Organización MIMD
# Memoria compartida



# Organización MIMD
# Memoria distribuida



# Clasificación de Sima, Fountain y Kacsuk [1997]

- Arquitecturas de paralelismo funcional
  - Arquitecturas ILP (paralelas a nivel de instrucción)
    - Procesadores encauzados
    - Procesadores VLIW
    - Procesadores superescalares
  - Arquitecturas paralelas a nivel de thread
  - Arquitecturas paralelas a nivel de proceso (MIMD)
    - MIMD de memoria compartida o multiprocesador
    - MIMD de memoria distribuida o multicomputador
- Arquitecturas de paralelismo de datos
  - Arquitecturas vectoriales
  - Arquitecturas asociativas y neuronales
  - Arquitecturas SIMD
  - Arquitecturas sistólicas

# Procesos y threads (hilos, procesos ligeros)

- Ambos consisten en la tarea de ejecutar un fragmento de programa y se crean para poder ejecutarse concurrentemente o en paralelo
- El S.O. asigna recursos distintos a cada proceso (espacio de memoria, procesador...) para su ejecución
- Los *threads* se crean dentro de un proceso y comparten los recursos del proceso
  - La existencia de *threads* supone un grano de paralelismo más fino que la existencia de procesos sin *threads*

# Arquitecturas multithread

- Grano más fino que las multitarea (thread más pequeño que proceso) =>
  - Más importante la **velocidad** en el **cambio de contexto**
  - Necesidad de **mecanismo hardware** para ese cambio
- Eficiencia procesador = busy /(busy+switching+idle)
  - *Busy, switching, idle*: cantidad de tiempo, medida sobre un largo intervalo, que el procesador está haciendo trabajo útil, cambiando de contexto y parado, respectivamente
  - Objetivo multithread: reducir *idle* sin que *switching* sufra un aumento sustancial (en todo caso es exigible que *switching* < *idle*)

# Arquitecturas multithread (cont)

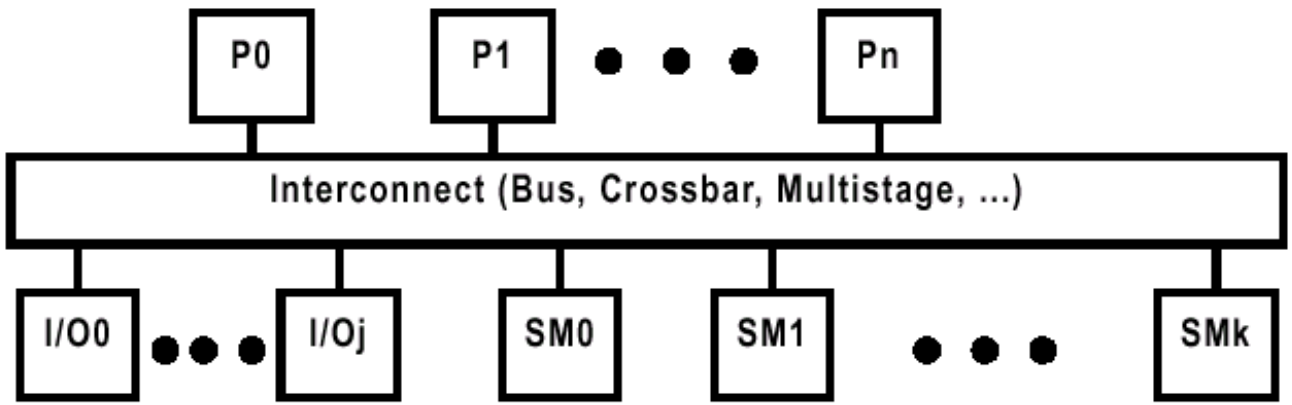
- Principales decisiones de diseño:
  - Modelo von Neumann / modelo híbrido (von Neumann-flujo de datos)
  - Granularidad fina/gruesa
  - Número de threads pequeño (4-10) / mediano (10-100) / grande (más de 100)
  - Memoria centralizada compartida / distribuida compartida

# Comparación de máquinas multithread

| Máquina | Modelo com-putacional | Granularidad | Organización de memoria | Threads por procesador | Año de aparición |
|---|---|---|---|---|---|
| **Denelcor HEP** | Von Neumann | Fina | Centralizada compartida | Hasta 64 | 1978 |
| **Tera** | Von Neumann | Fina | Distribuida compartida | Hasta 128 | 1990 |
| **MIT Alewife** | Von Neumann | Gruesa | Distribuida compartida | Hasta 3 | 1990 |
| **EM-4** | Híbrida-flujo de datos | Gruesa | Distribuida compartida | Ilimitado | 1990 |
| ***T** | Híbrida-Von Neumann | Fina | Distribuida compartida | Ilimitado | 1992 |

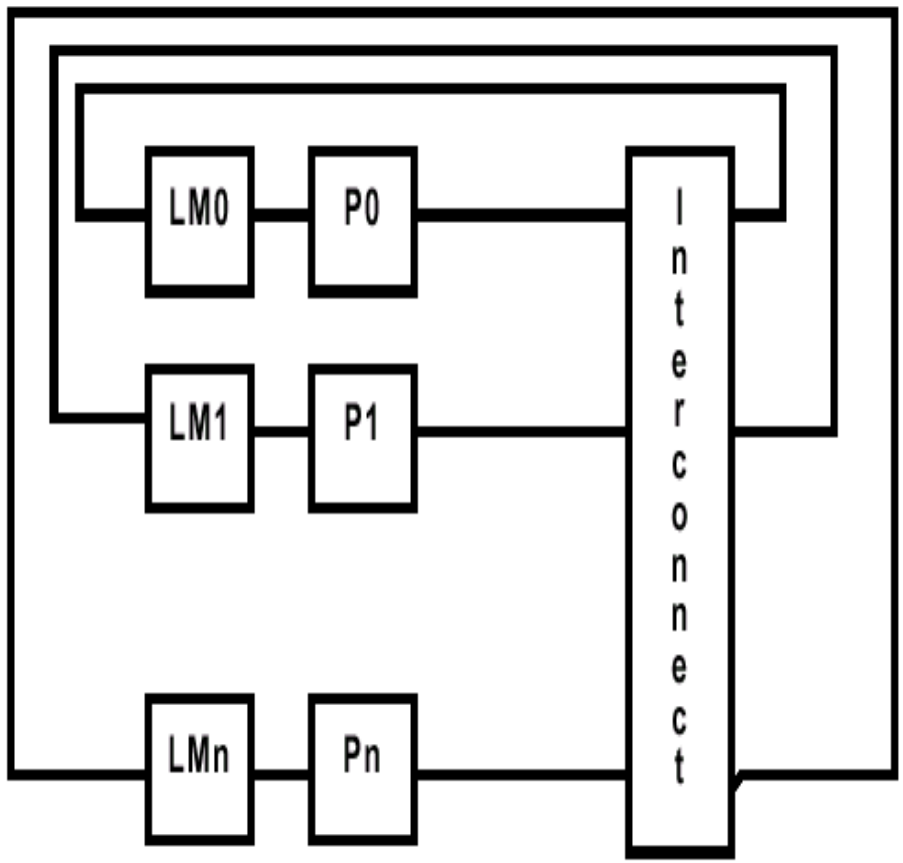
# Arquitecturas UMA (multiprocesadores simétricos)

- Todos los procesadores comparten
  - Memoria (con similares tiempos de acceso)
  - Entrada/salida
- Propia de multiprocesadores de pequeña escala
- Ejemplos: Sun Enterprise 6000, SGI Challenge, Intel SystemPro



# Arquitecturas NUMA

- El tiempo de acceso de un procesador a memoria es diferente en diferentes zonas de memoria
- Propia de multiprocesadores de gran escala
- Muchas variantes
- Ejemplos: Cray T3D, KSR-1, Convex SPP1000



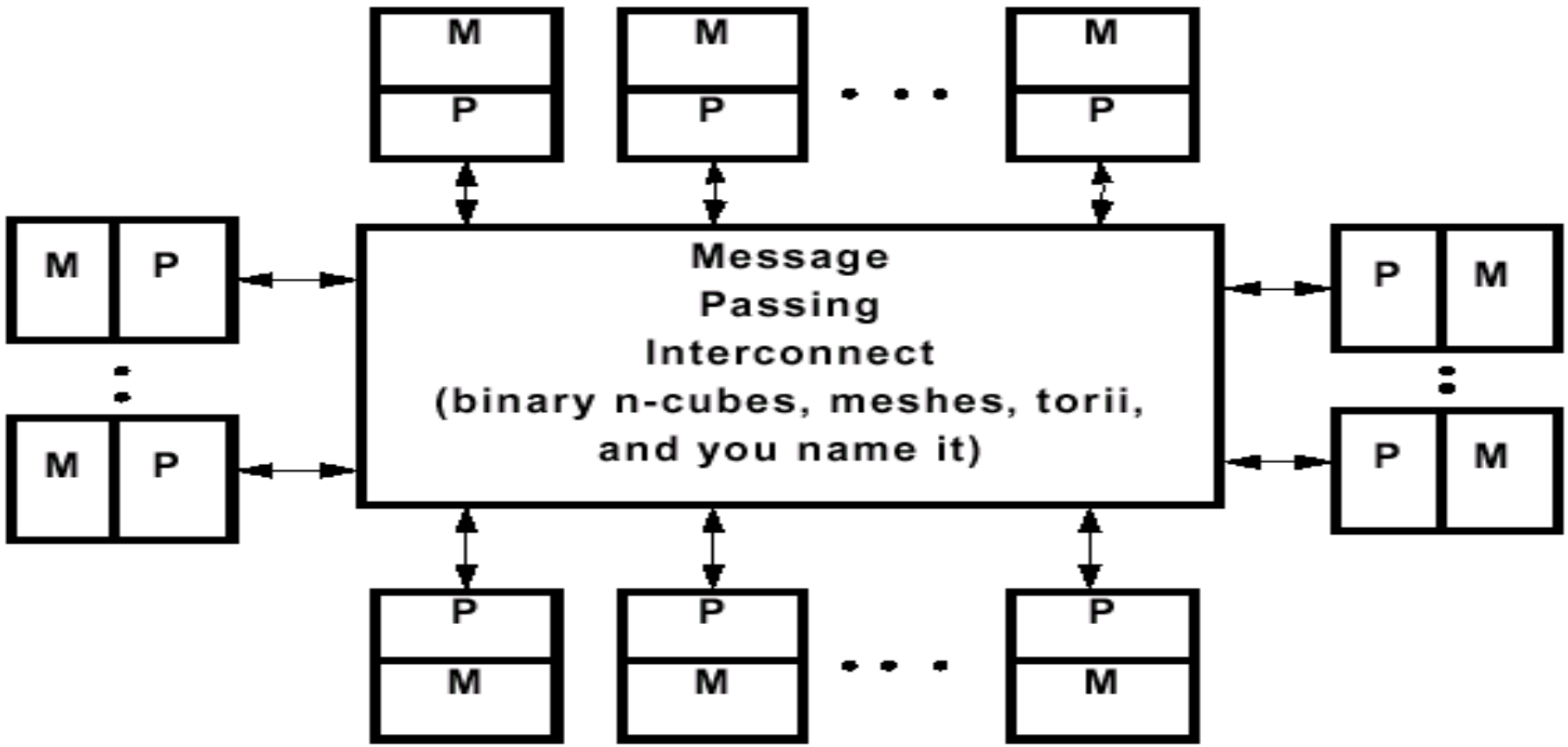
# Comunicación y sincronización en los multiprocesadores

- Comunicación mediante variables compartidas =>
  - Aparecen **secciones críticas**. Ej.:
  - *LOAD Reg[1],V* / *ADD Reg[1],#1* / *STORE V,Reg[1]* — *LOAD Reg[2],V* / *ADD Reg[2],#2* / *STORE V,Reg[2]*
- Sincronización mediante instrucciones ininterrumpibles específicas. Ej.:
  - *WAIT(S)* / Sección crítica 1 / *SIGNAL(S)* — *WAIT(S)* / Sección crítica 2 / *SIGNAL(S)*
  - S es una variable compartida (suponemos inicialmente S=1)
  - *WAIT(S) = [**If** S=0 **then** "impedir el paso" **else** S ← S-1]*
  - *SIGNAL(S) = [S ← S+1]*

# Multicomputadores

- Son una colección de computadores (CPU, memoria, elementos de E/S) que se comunican:
  - A través de una **red escalable, de alto rendimiento**
    - Muy importante la topología de la red
  - Mediante **operaciones explícitas de E/S**
    - Send: **SEND expresión TO proceso destino**
    - Receive: **RECEIVE variable FROM proceso fuente**
    - Cuando un proceso emisor y uno receptor **se nombran mutuamente** se establece un **canal lógico de comunicación** entre ambos
    - *Send* y *receive* resuelven la **sincronización** entre procesos
- Ejemplos: Schlumberger FAIM- 1, HPL Mayfly, NCUBE, Intel iPSC, Parsys SuperNode1000, Intel Paragon

# Multicomputadores



# Problemas de los MIMD escalables y posibles soluciones

- Problemas de latencia de memoria
- Pérdidas de tiempo debidas a la sincronización entre procesos
- Posibles soluciones
  - Uso de memoria **caché**
  - **Red de interconexión** eficiente
  - **Conmutación de procesos**: cuando un proceso quiere acceder a un recurso ocupado, al procesador se le asigna otro proceso para evitar que esté parado
    - Problema: el estado del proceso suspendido debe ser salvado y el del proceso activado debe ser cargado (cambio de contexto)
  - Utilización de **threads** junto a un **mecanismo rápido de cambio de contexto** entre threads

# Convergencia de las arquitecturas paralelas

- La evolución del hardware y del software ha conducido a una convergencia de las máquinas paralelas hacia una estructura común:
  - **Varios nodos que se comunican mediante una red escalable de propósito general y alto rendimiento**
  - **Cada nodo** incluye
    - **Uno o más procesadores (integran caché [$])**
    - **Memoria**
    - Un **controlador para las operaciones a través de la red (CA)**
- Desde esta perspectiva, la cuestión básica del diseño es
  - Qué funcionalidad debe tener el CA (*comunication assist*)
  - Cómo debe ser la interfaz entre el CA y el/los procesadores, el sistema de memoria y la red [caja negra en la fig. sgte.]

# Organización genérica de un MIMD escalable