Asignatura 780014
Programación Avanzada

TEMA 2 – INTRODUCCIÓN A LA CONCURRENCIA

"El hardware es lo que hace a una máquina rápida; el software es lo que hace que una máquina rápida se vuelva lenta"

Bruce Craig (empresario)

Introducción a la concurrencia

- Objetivo del tema:
  - Presentar el concepto de concurrencia, sus inicios y sus principales características.

Índice

1. Definición
2. Historia
3. Beneficios / riesgos
4. HW y SO
5. Notación
6. Java

Definición

- Definición de Concurrencia
  - “Notaciones y técnicas usadas para expresar en programas la posibilidad potencial de ejecución simultánea de varios algoritmos y resolver los problemas de sincronización y comunicación derivados de realizar dicha implementación”.

Definición

- Definición de Concurrencia
  - "**Notaciones** y **técnicas** usadas para expresar en programas la **posibilidad potencial** de ejecución simultánea de varios algoritmos y resolver los problemas de sincronización y comunicación derivados de realizar dicha implementación".

Notaciones, forma de representar los conceptos.

Técnicas, para descubrir la concurrencia potencial de los algoritmos.

Posibilidad potencial de ejecución simultánea: algoritmos con concurrencia "latente", pero existente.

Definición

- Definición de Concurrencia
  - "Notaciones y técnicas usadas para expresar en programas la posibilidad potencial de ejecución simultánea de varios algoritmos y **resolver los problemas** de **sincronización** y **comunicación** derivados de realizar dicha implementación".

Ejecución simultánea de tareas **de forma armoniosa**:

- **Comunicación** entre procesos que cooperan.
- **Sincronización** de procesos.

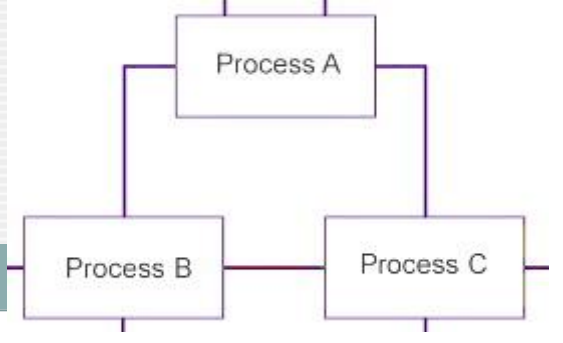
Resolver los problemas utilizando **herramientas** que permitan diseñar la concurrencia y ejecutarla según el diseño.

De forma segura:

el resultado no debe ser función del número de procesadores o del tiempo.

Historia de la concurrencia (1º)

- Aparece en el diseño de Sistemas Operativos
  - En fase temprana de su evolución
  - Con los siguientes objetivos:
    - Optimizar el uso de algunos recursos
      - La CPU es mucho más rápida que las unidades de E/S
    - Realizar un reparto más justo de recursos
      - Un programa no tendrá que esperar a otros para comenzar
    - Simplificar el desarrollo
      - Un programa que se puede dividir en tareas que cooperan, es más sencillo de codificar

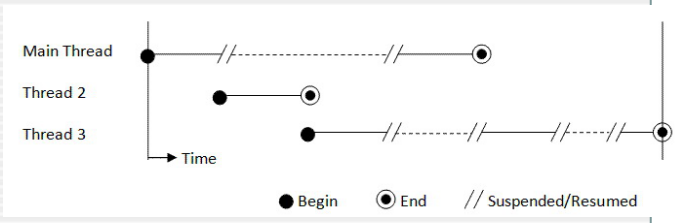


Historia de la concurrencia (2º)

- Requiere cambios en los lenguajes de programación
  - Los primeros lenguajes no permitían crear prog. concurrentes
  - Se crean librerías del SO para crear la concurrencia
    - Esto dificulta la creación de programas porque la concurrencia no está integrada en el lenguaje
  - Se crean compiladores que convierten los programas en concurrentes analizando lo que hacen
    - Esta técnica desaprovecha posibilidades: los programas no estaban pensados concurrentes
  - Finalmente aparecen lenguajes que contemplan la concurrencia en su propio diseño

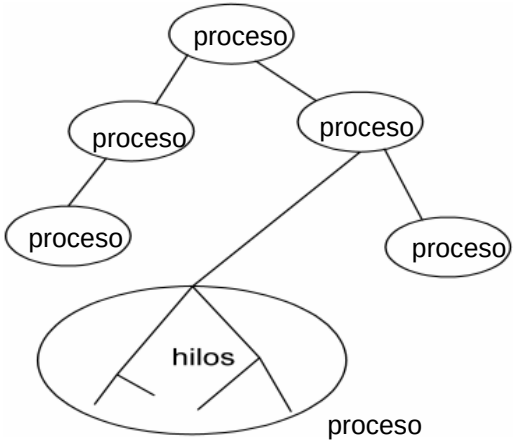
Historia de la concurrencia (3º)

- La concurrencia que ofrece el sistema operativo ofrece programas secuenciales que son concurrentes entre sí (llamados procesos)
  - Los procesos se comunican entre sí mediante mecanismos del SO (semáforos, sockets y otros)
  - Esta concurrencia limita las opciones para crear programas
- En los años 80 aparecen sistemas operativos con:
  - Múltiples procesos concurrentes
  - Múltiples tareas dentro de cada proceso
    - Son los hilos
    - Comparten el mismo espacio de direcciones



Historia de la concurrencia (4º)

- Hilos (Threads) = "Procesos ligeros"
  - Unidad básica de ejecución concurrente
  - Ejecución simultánea y asíncrona
  - Necesidad de coordinar su acceso a los datos que comparten
    - Evitar corrupción de los datos (seguridad o safety)
  - Necesidad de coordinar la disponibilidad de los recursos
    - La CPU es crítica (vitalidad o liveness)
- Usaremos un modelo basado en hilos
  - Como programadores, nos permite controlar el flujo de ejecución



Beneficios de los hilos

- **Aprovecha múltiples CPU** (↑ rendimiento)
  - Es más barato añadir otra CPU que aumentar la velocidad
  - Permite aprovechar varios procesadores para un programa
- Mejora el **rendimiento** incluso en sistemas con una sola CPU
  - Si, en un momento dado, en el programa se solicita una operación de E/S:
    - Si en el programa hay **un solo hilo**: el procesador está *idle*, esperando a que se complete la operación de E/S
    - Si el programa es **multi-hilo**: otro hilo puede ejecutarse mientras el primer hilo está esperando a que se complete la E/S, permitiendo a la aplicación continuar durante el bloqueo por E/S

Beneficios de los hilos

- **Simplifica el modelado** de los programas
  - Asignando una tarea específica a cada hilo, el código es:
    - Más fácil de escribir
    - Menos propenso a errores
    - Más fácil de testear
  - Cada tarea (hilo):
    - Se programa de forma independiente (menos complejo cada uno)
    - Se prueba, se mantiene y repara de forma independiente
- Permite atender **varias solicitudes simultáneas** con un solo programa activo
  - Asignando un hilo a cada conexión
    - Ejemplo: RMI

Beneficios de los hilos

- Facilita la **gestión de eventos asíncronos**
  - Gracias al propio diseño de los hilos y su comportamiento
  - Facilita la creación de servidores (Web, BBDD, etc.)
    - Un cliente no acapara un servidor, sólo un hilo del mismo
  - Permite crear interfaces de usuario más amigables
    - Se evita que la interfaz se pare si el programa está ocupado
    - Siempre hay un hilo dispuesto a detectar las acciones del usuario sobre los componentes

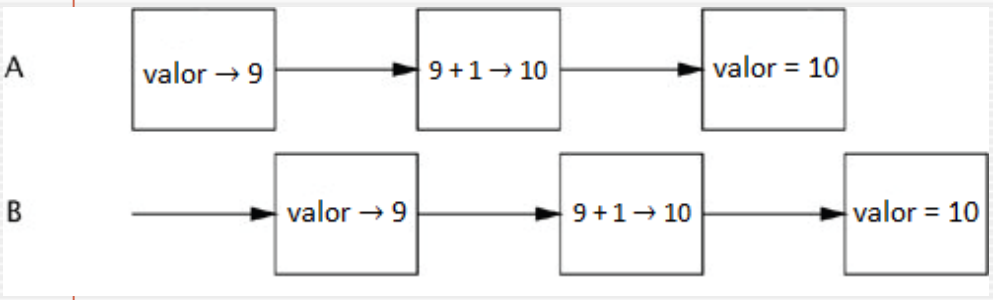
Riesgos de los hilos

- Seguridad (safety)
  - El acceso simultáneo a elementos compartidos puede ocasionar inconsistencia por:
    - "**Condiciones de carrera**" (race condition): el resultado del programa depende del orden de ejecución de los hilos
  - Un dato inconsistente puede invalidar todo un programa
  - Se evita garantizando el acceso a variables en **exclusión mutua**
    - En Java, mediante métodos o bloques "`synchronized`"

Riesgos de los hilos

- Ejemplo Java:

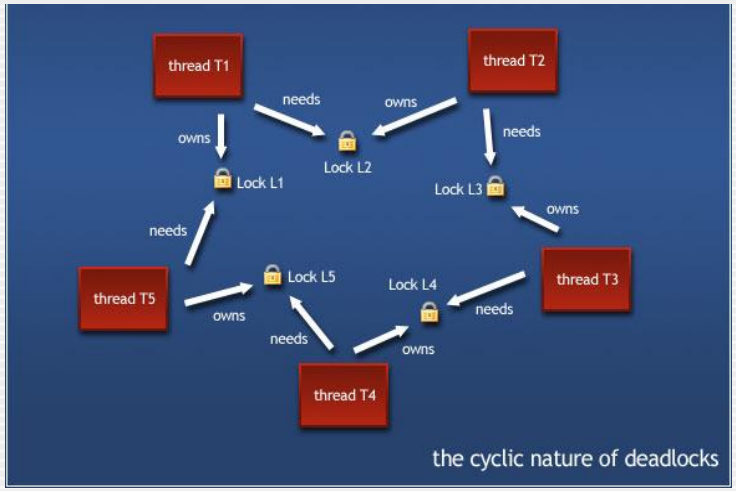
```
public class SecuenciaInsegura
{
  private int valor;
  public int getSiguiente()
  {/*Puede devolver valores
diferentes*/
      return valor++;
  }
}
```




```
public class Secuencia
{
   private int valor;
   public synchronized int getSiguiente()
   {/*Devuelve un único valor*/
       return valor++;
   }
}
```

Riesgos de los hilos

- Interbloqueo (deadlock)
  - Si tenemos varios recursos y varios hilos que los necesitan
  - Si el uso de los recursos implica bloquearlo en exclusividad
  - Si un hilo espera por un recurso hasta que queda libre
  - Puede ocurrir que se cree un ciclo de necesidad de recursos y bloqueos



  - Puede ser difícil de detectar y bloquear todo un sistema

Riesgos de los hilos

- Inanición (starvation)
  - Ocurre cuando un hilo entra en un estado en el cual es, permanentemente, **incapaz de progresar**
    - Ejemplo: **mala planificación**, con varios hilos intentando utilizar la CPU, y alguno nunca la obtiene
  - La inanición puede bloquear en cascada todo el programa
- Escaso rendimiento
  - Un sistema mal diseñado puede tener **mayor coste en concurrencia que lo que ahorra** su uso
    - La creación de hilos implica un coste de CPU y memoria
    - El control de hilos implica un coste de CPU y memoria
      - Cambios de contexto, tiempo de CPU en planificación de hilos, etc.

HW y SO: Niveles de implementación

- Nivel de operación (CPU) (Hardware):
  - Procesamiento a nivel de palabra.
- Nivel de instrucción (CPU) (Hardware):
  - Ejecución simultánea de varias instrucciones (‘pipelining’).

HW

SW

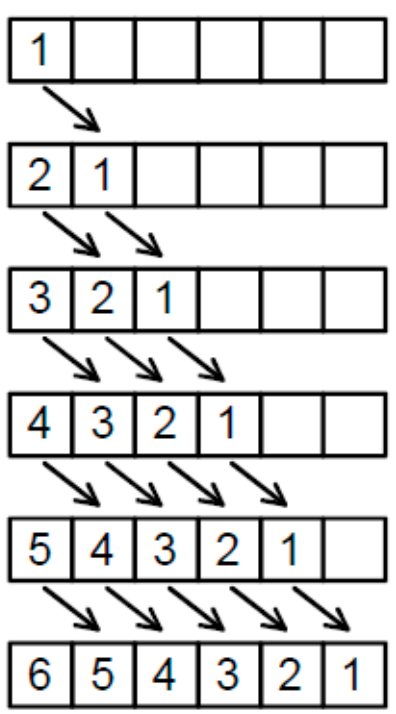
- Nivel de programa (cooperación) (Software):
  - Un mismo programa con varias tareas (hilos).
- Nivel de aplicaciones (competencia) (Software):
  - Varias aplicaciones en un mismo sistema.

HW y SO: Niveles de implementación

- Nivel de operación (CPU) (Hardware)
  - Procesamiento simultáneo de varios **bits** por operaciones aritméticas y lógicas dentro del procesador
    - Actualmente: 64 bits
  - Se consigue mediante la duplicidad de los componentes internos de los procesadores
    - Ejemplo: sumadores paralelos para acelerar los cálculos
  - Este nivel es transparente para nosotros

HW y SO: Niveles de implementación

- Nivel de instrucción (CPU) (Hardware)
  - Ejecución simultánea de varias **instrucciones**
  - Se consigue con uso de:
    - Varios procesadores
    - O un procesador diseñado para ejecutar varias instrucciones mediante solapamiento o 'pipelining'
  - Este nivel es transparente para nosotros



HW y SO: Niveles de implementación

- Nivel de programa (cooperación) (Software)
  - Realización simultánea de varias subtareas (**hilos**)
    - Ejemplo: procesador de textos
  - Este nivel implica la existencia de un software (SO, compilador) adecuado para la implantación de la concurrencia
    - Ejemplo: se encarga de repartir los procesadores cuando en el ordenador existan menos procesadores que procesos (hilos)
  - Este es el nivel en el que trabajaremos

HW y SO: Niveles de implementación

- Nivel de aplicaciones (competencia) (Software)
  - Ejecución simultánea de varios **programas**
  - El SO debe soportarlo
    - Al principio sólo los grandes sistemas lo soportaban, ahora también los equipos personales
    - Ejemplo: UNIX/Linux y Windows
  - Este nivel no es relevante para nosotros

HW y SO: Tipos de hardware concurrente

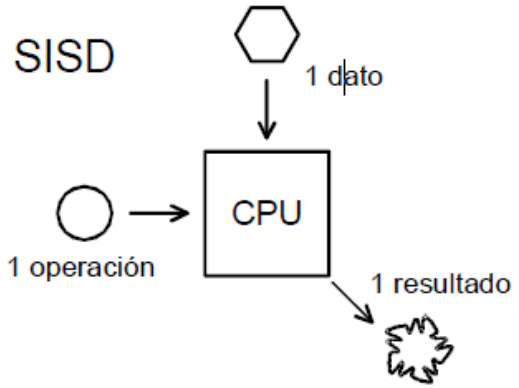
- 4 grupos de ordenadores (Flynn, 1966), basados en la estructura de los procesadores
  - Unidad de control (CU): encargada de la decodificación de las instrucciones máquina
  - Unidad de cálculo (ALU): encargada de realizar las operaciones aritméticas y lógicas indicadas en las instrucciones
  - CPU=(1..n)CU + (1..n)ALU
- Tipos:
  - **SISD** (*Single Instruction Single Data*)
  - **SIMD** (*Single Instruction Multiple Data*)
  - **MISD** (*Multiple Instruction Single Data*)
  - **MIMD** (*Multiple Instruction Multiple Data*)

http://en.wikipedia.org/wiki/Flynn%27s_taxonomy

HW y SO: Tipos de hardware concurrente

SISD (*Single Instruction Single Data*):

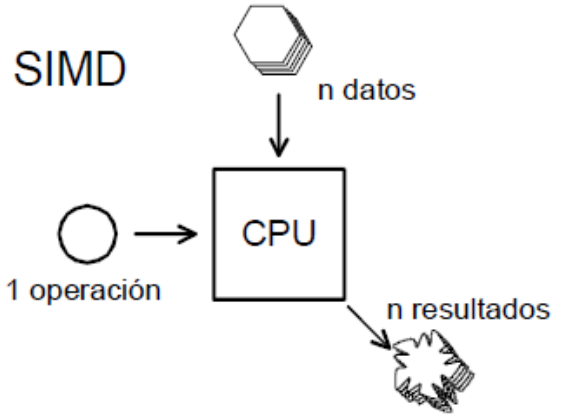
- Computador secuencial con procesador único
  - 1 CU + 1 ALU
- Capaz de ejecutar una instrucción por vez, y ésta aplicada sobre un dato único
- Se corresponden con esta arquitectura:
  - Los antiguos equipos de sobremesa
  - Pequeños ordenadores de uso específico



HW y SO: Tipos de hardware concurrente

SIMD (*Single Instruction Multiple Data*)

- Primeros intentos de obtener rendimientos elevados
- Podía ejecutar una instrucción sobre un gran número de datos (p.e. varias sumas)
  - 1 CU + n ALUs
- Se corresponden con esta arquitectura:
  - Ordenadores vectoriales
  - Recientemente disponible en la mayoría de los ordenadores en su GPU



HW y SO: Tipos de hardware concurrente

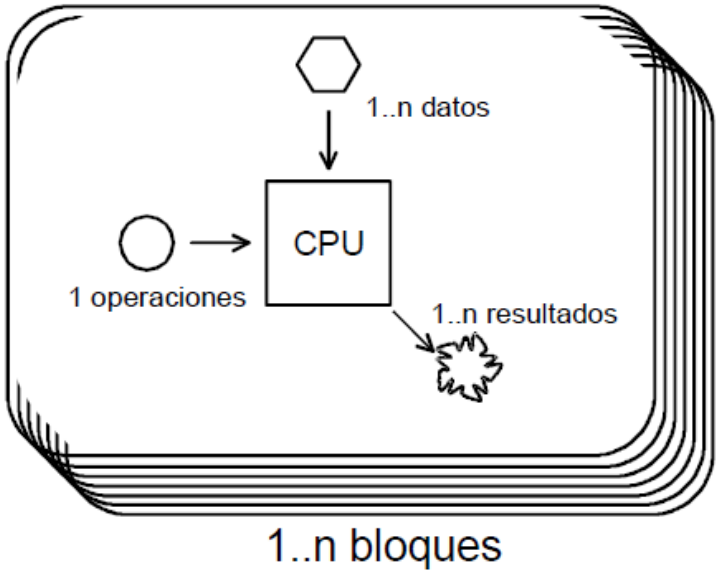
MISD (*Multiple Instruction Single Data*):

- Una secuencia de datos se transmite a una línea de procesadores, cada uno controlado por una CU propia
- Esta posibilidad no es muy común
- Los sistemas tolerantes a fallos la utilizan
  - Realizan diferentes cálculos con los mismos datos y los resultados deben coincidir para ser aceptados
    - Ejemplo: tecnología usada para las naves espaciales, máquinas criptográficas, etc.

HW y SO: Tipos de hardware concurrente

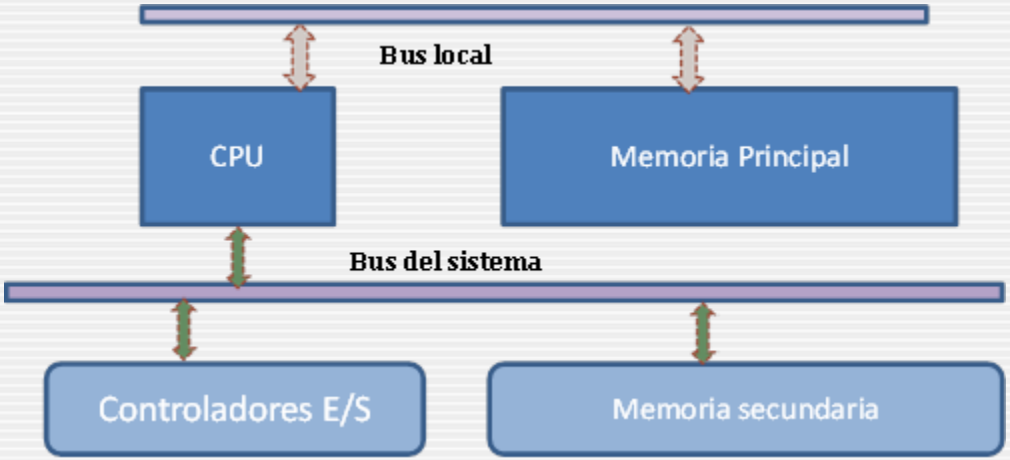
MIMD (*Multiple Instruction Multiple Data*):

- HW realmente concurrente: ejecución simultánea de varias instrucciones con diferentes datos
- Hay varios procesadores y, por tanto:
  - Varias unidades de control (n CUs)
  - Varias unidades de cálculo (n ALUs)
- Categorías:
  - Multiprocesadores
  - Multicomputadores
  - Sistemas distribuidos



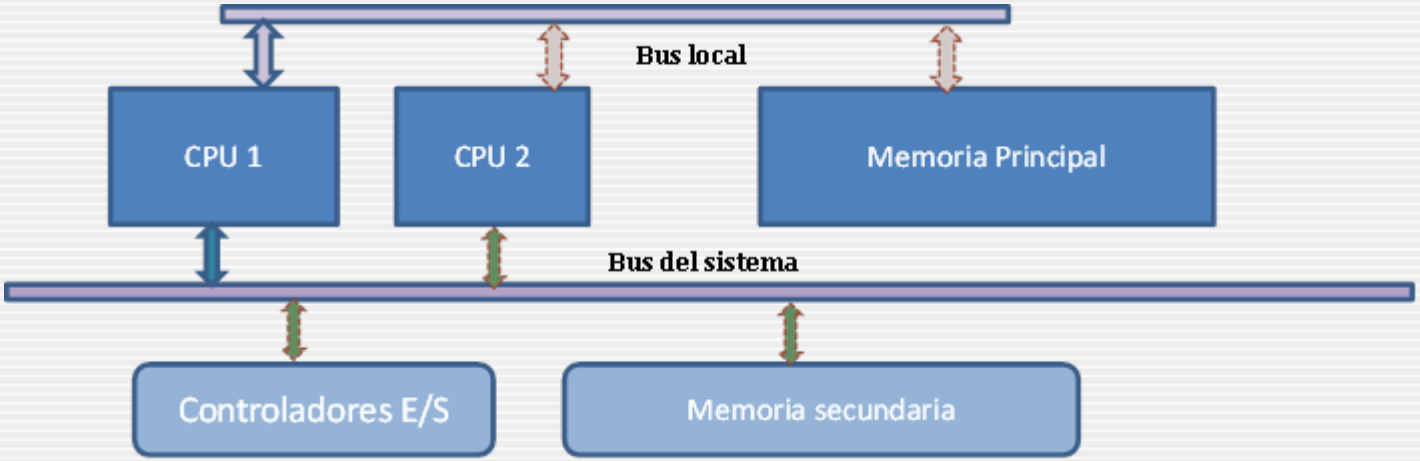
HW y SO: Concurrencia en monoprocesador

- **Concurrencia simulada** que mejora el rendimiento del conjunto
- La aplicación concurrente puede o no ser ejecutada más eficientemente
- Para que el SW (razones):
  - Optimice la utilización de los recursos.
  - Sirva a múltiples usuarios.
  - Consiga un diseño más simple y comprensible (al dividir el problema en tareas pequeñas).



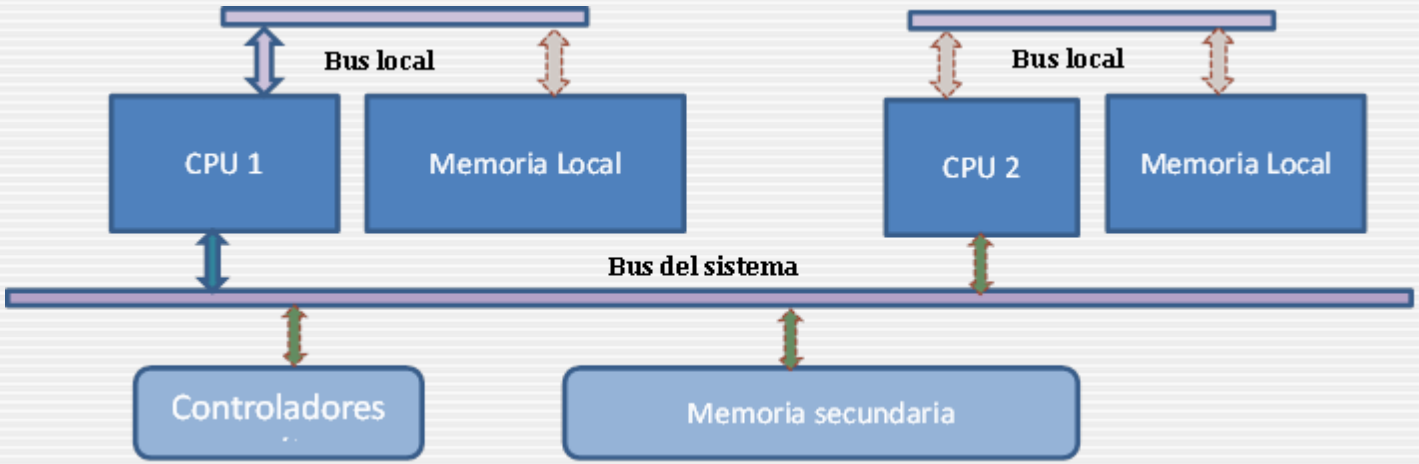
HW y SO: Concurrencia en multiprocesador

- Los procesadores **comparten** una **memoria común**
- Los procesos se ejecutan con **concurrencia física**
- El programa concurrente se ejecuta más eficientemente



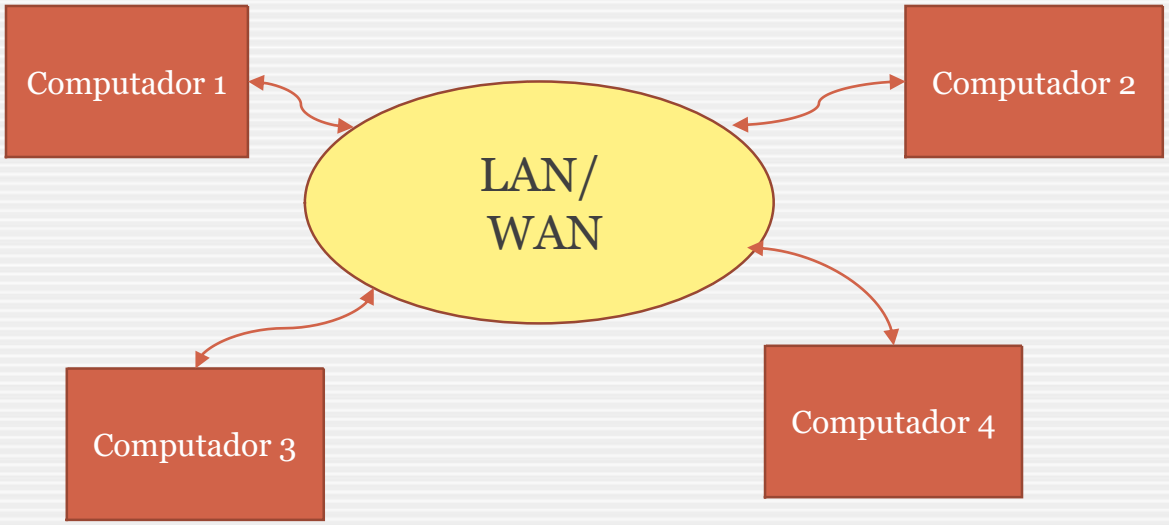
HW y SO: Concurrencia en multicomputador

- **Cada procesador tiene una memoria local**
- Comparten controladores, discos, etc. (*clusters*)
- Los procesos se ejecutan con **concurrencia física**
- No puede haber hilos de un mismo proceso en dos CPUs



HW y SO: Concurrencia en distribuido

- **Cada computador** es **independiente** del resto
- Se comunican a través de LAN o WAN
- Los procesos pueden intercambiar información a través de la red



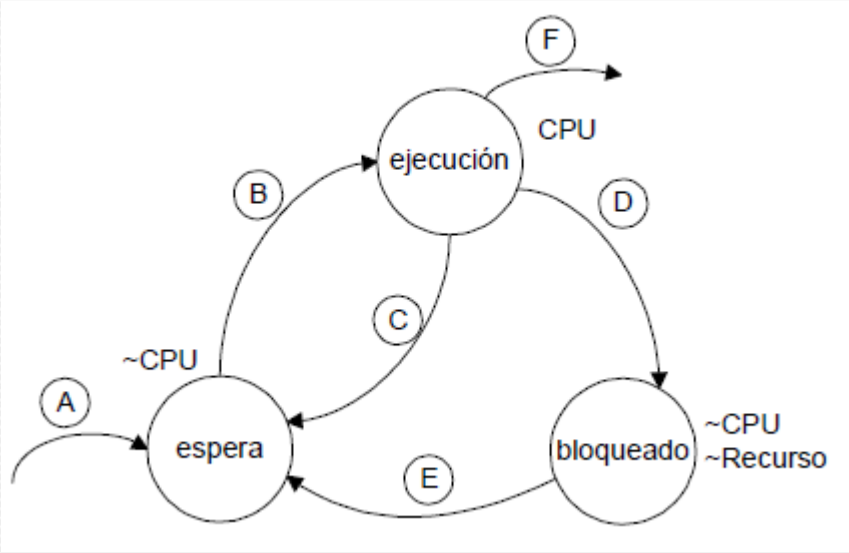
HW y SO: Tipos de concurrencia

- Independientemente de la arquitectura, un programa es **concurrente** si consigue que el ordenador ejecute más de una actividad de forma **simultánea**
  - Dos procesos (hilos) son concurrentes si la primera instrucción de uno de ellos (Pr1) se ejecuta en el tiempo que va entre la ejecución de la primera y de la última instrucción del otro (Pr2)

- La simultaneidad puede ser:
  - **Real**: la arquitectura dispone de hardware redundante (**más de un procesador**) y puede tener varias instrucciones ejecutándose simultáneamente en los componentes duplicados
  - **Simulada o aparente**: reparto del **único procesador** para simular el avance en paralelo de todas las tareas

HW y SO: Estados de un proceso

- **En ejecución**: el proceso está ocupando un procesador que ejecuta sus instrucciones
- **Preparado/Listo/En espera**: el proceso está listo para que se le asigne un procesador para ejecutarse sobre él
- **Bloqueado**: el proceso está a la espera de algún recurso y no puede pasar a ejecutarse



Notación

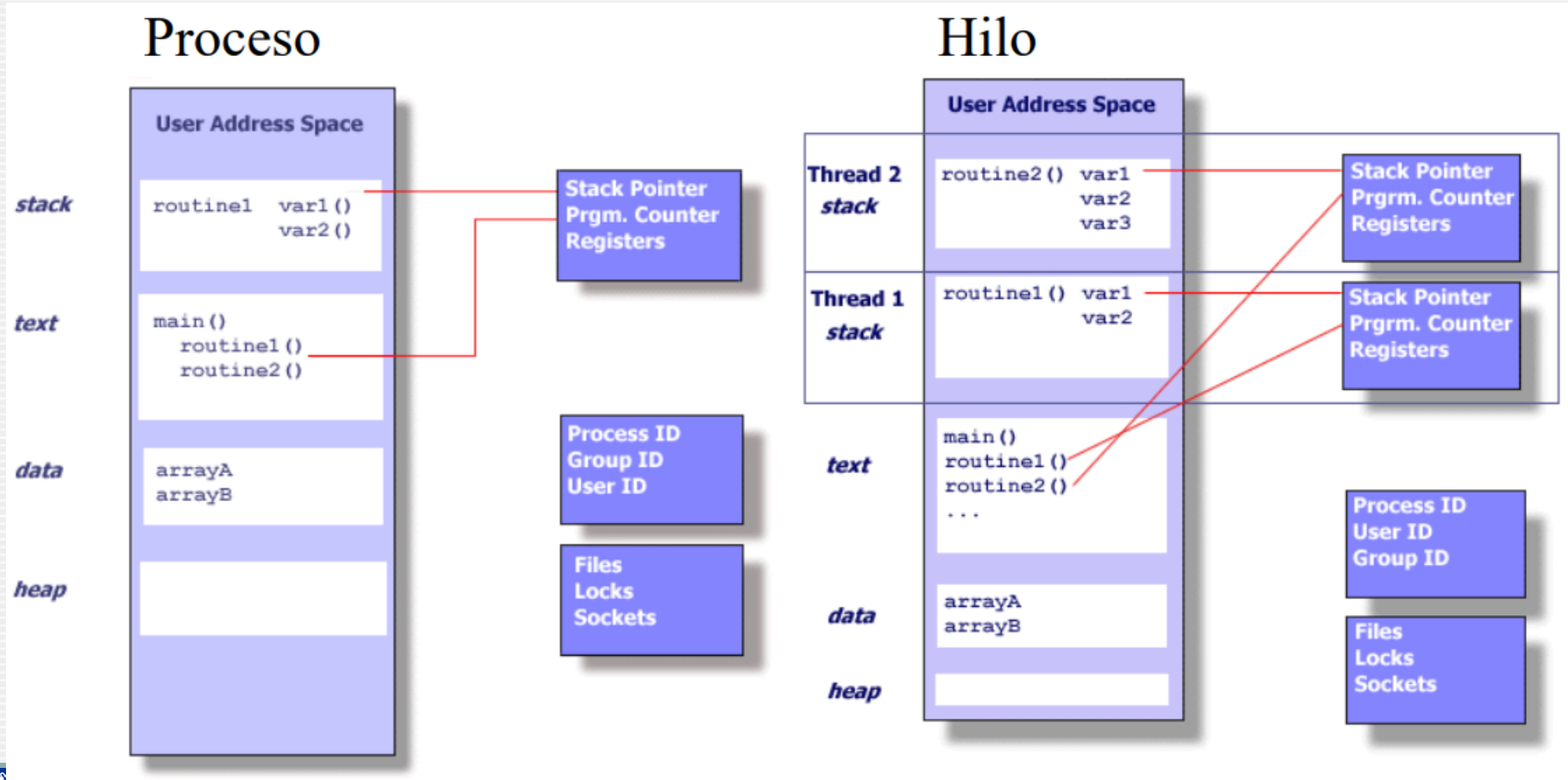
- El término “hilo”, en ocasiones, se confunde con “proceso” o “tarea”
- Muy importante la nomenclatura utilizada:
  - Procesador (CPU)
  - Multiprocesador (varias CPUs)
  - Programa (código fuente o código compilado)
  - Proceso (programa en ejecución)
  - Subproceso (denota jerarquía)
  - Hilo (*thread*) (proceso ligero)
  - Tarea (alto nivel, a nivel conceptual)
  - Multiproceso (S.O. capaz de manejar varios procesos a la vez)
  - Multitarea (sinónimo del anterior, que hace varias cosas a la vez)

Notación

- Diferencia entre proceso e hilo
  - Los **procesos** son entidades **independientes** para el SO
    - Tienen su propia memoria, código, montículo, etc.
  - Los **hilos comparten** memoria, código, montículo, etc.
    - Lo único que no comparten es el puntero de instrucción
  - Para el SO es más sencillo crear hilos

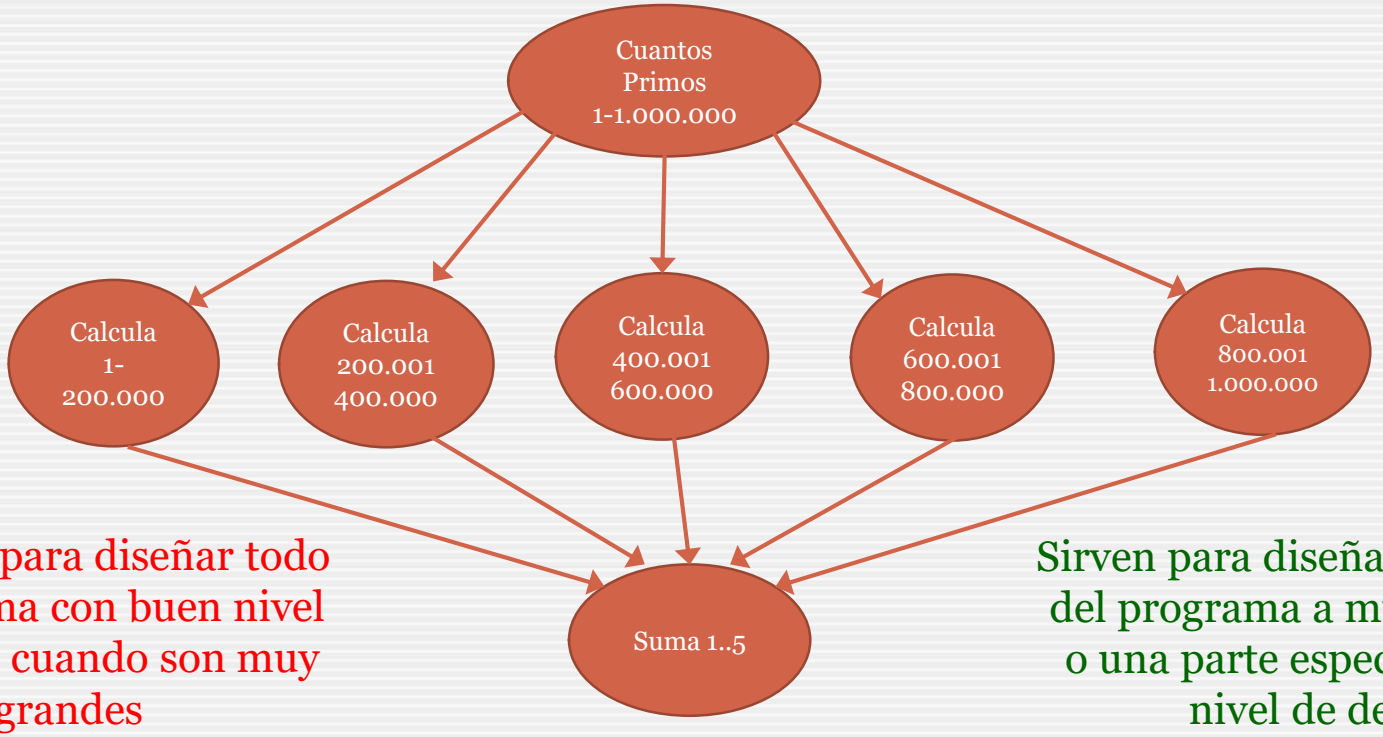
Notación

- Diferencia entre proceso e hilo



Notación: Diagramas de precedencia

- **Diagrama de precedencia**: es un grafo dirigido donde:
  - Los **nodos** representan secuencias de **instrucciones**
  - Las **flechas** indican el **orden** en que deben ejecutarse

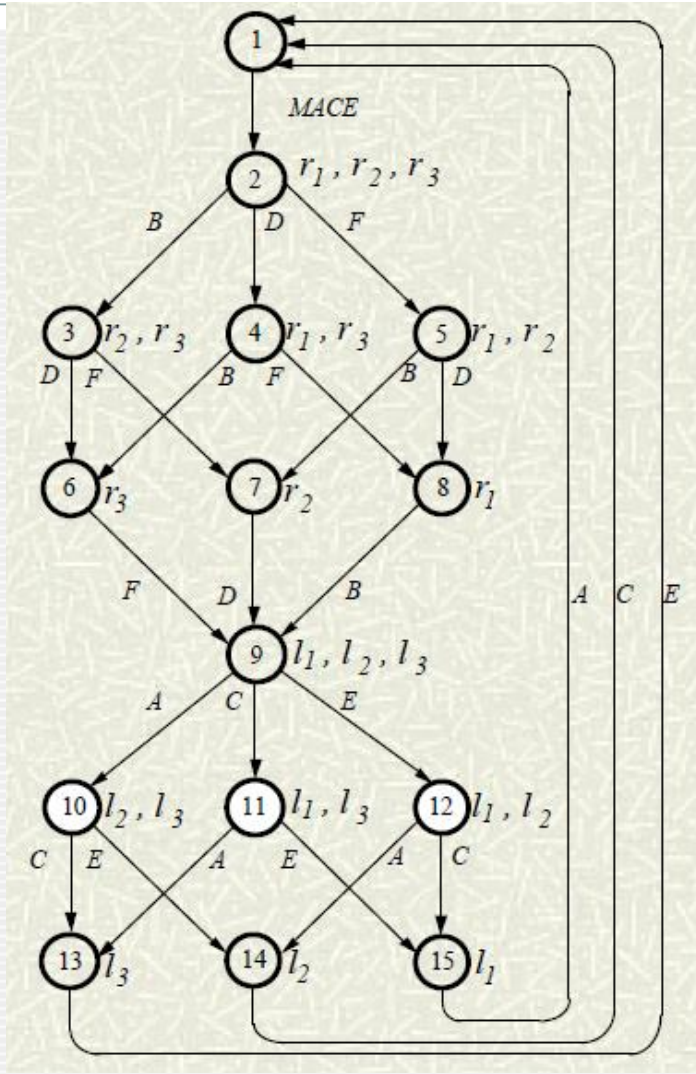


No sirven para diseñar todo el programa con buen nivel de detalle cuando son muy grandes

Sirven para diseñar el conjunto del programa a muy alto nivel o una parte específica a bajo nivel de detalle

Notación: Diagramas de estados

- El espacio de estados se hace muy complejo cuando se tratan sistemas concurrentes
  - Para N elementos, 2^{N+1}-1 estados
- Poco flexibles: cambios en la especificación implican cambios drásticos del modelo
- Se requieren otros métodos formales: Redes de Petri

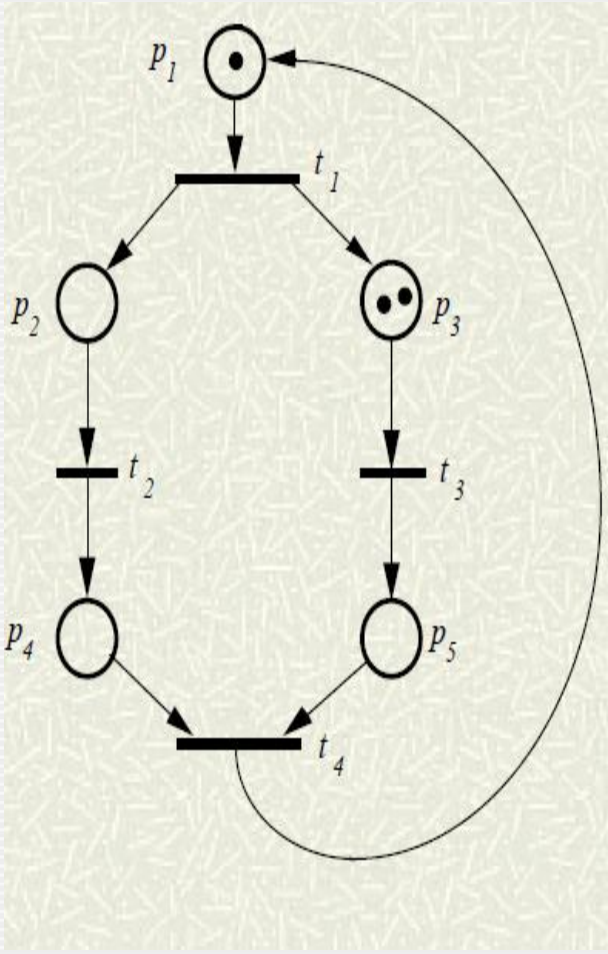


Notación: Redes de Petri

- Las **redes de Petri** (PN) (C.A. Petri, 1962) son una herramienta de modelado muy efectiva para la representación y el análisis de procesos concurrentes
- Simplicidad
  - Pero la representación de grandes sistemas es costosa
- Extensiones:
  - Red de Petri Temporizada: introduce el tiempo, para modelar el comportamiento de los sistemas dinámicos.
  - Red de Petri Estocástica: especifica el comportamiento temporal con variables aleatorias exponenciales.
  - Red de Petri Coloreada (CPN): permite modelar sistemas concurrentes descritos mediante flujos de datos.

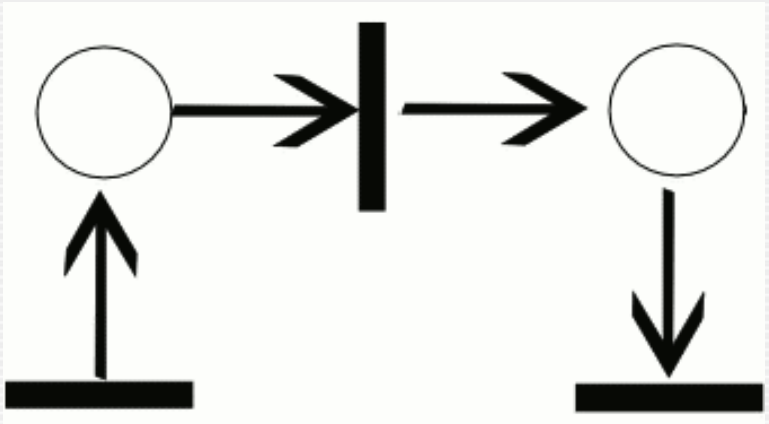
Notación: Redes de Petri

- Una red de Petri es un grafo orientado con dos clases de nodos: **lugares** (circunferencias) y **transiciones** (barras). Los **arcos** unen un lugar con una transición o viceversa.
- Un lugar puede contener un número positivo o nulo de **marcas** (bolitas).
- Distribución de marcas en los lugares: **marcado** → estado de la red.



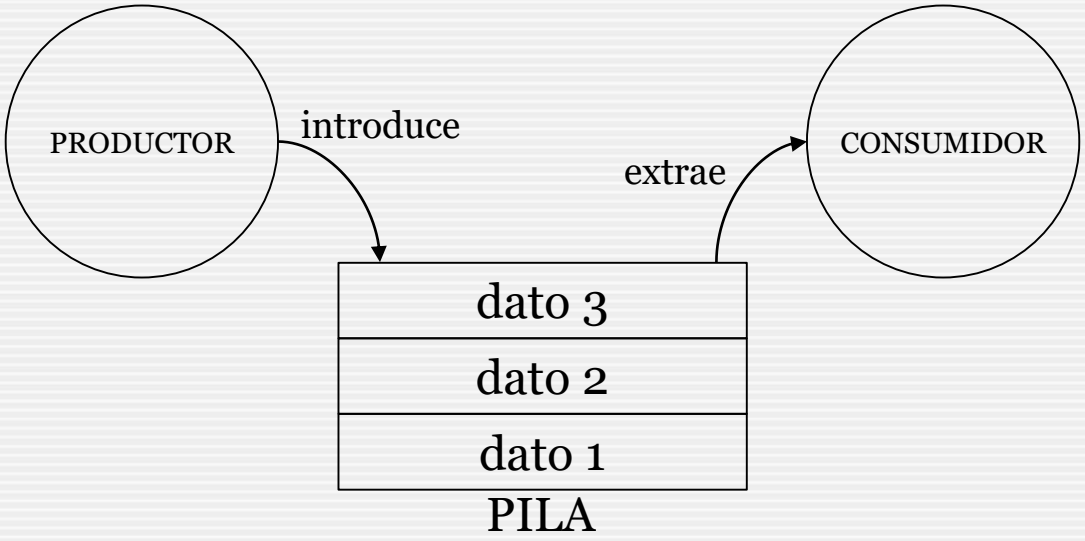
Notación: Redes de Petri

- Mediante una red de Petri puede modelarse un sistema concurrente:
  - Las **marcas** se interpretan como **recursos**
  - Las **transiciones** (acción a ejecutar)
    - Necesitan **precondiciones**
    - Generan **postcondiciones**



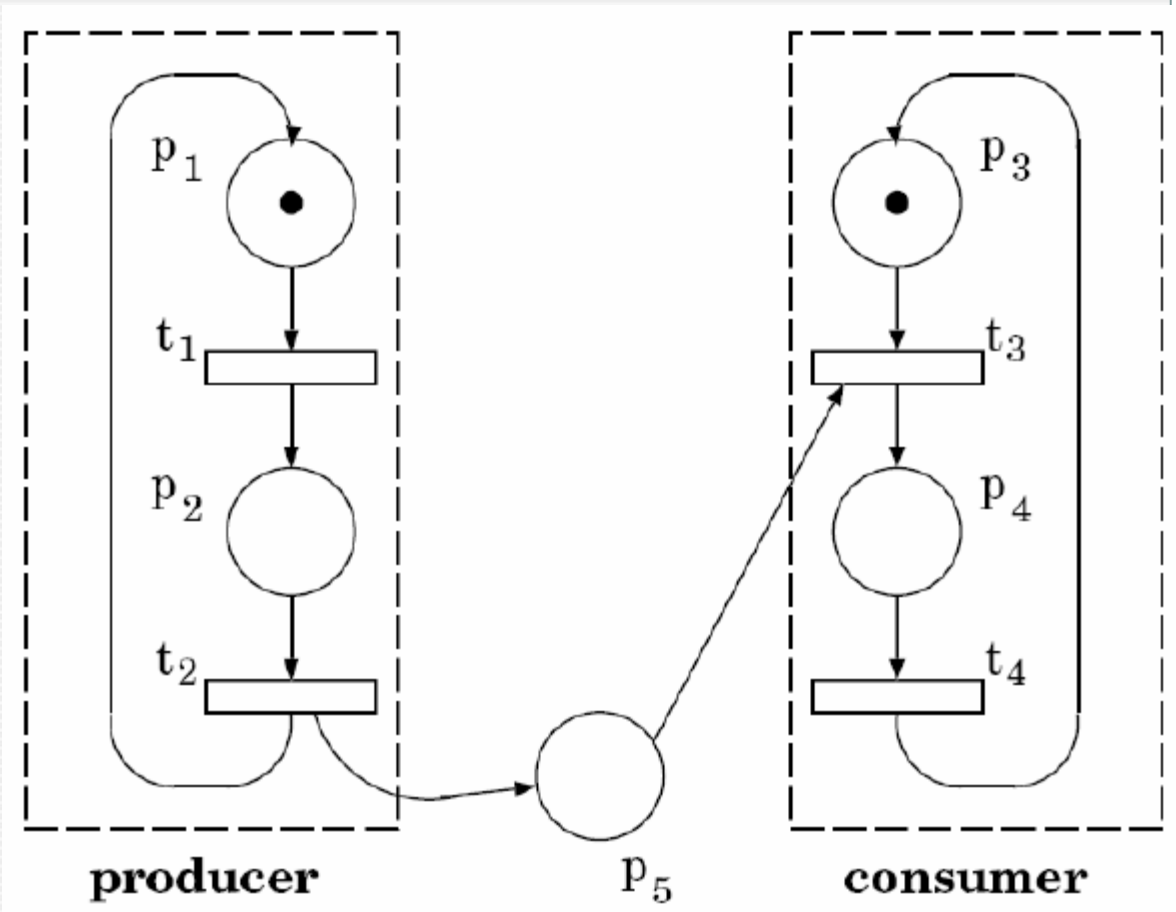
Notación: Modelo de Productor-Consumidor

- Problema del "Productor-Consumidor":
  - Dos **procesos concurrentes** utilizan una **estructura de datos**
  - La estructura puede ser desde un valor simple hasta una pila
  - Uno de los procesos produce datos y los deposita en la estructura
  - El otro proceso extrae los datos para utilizarlos

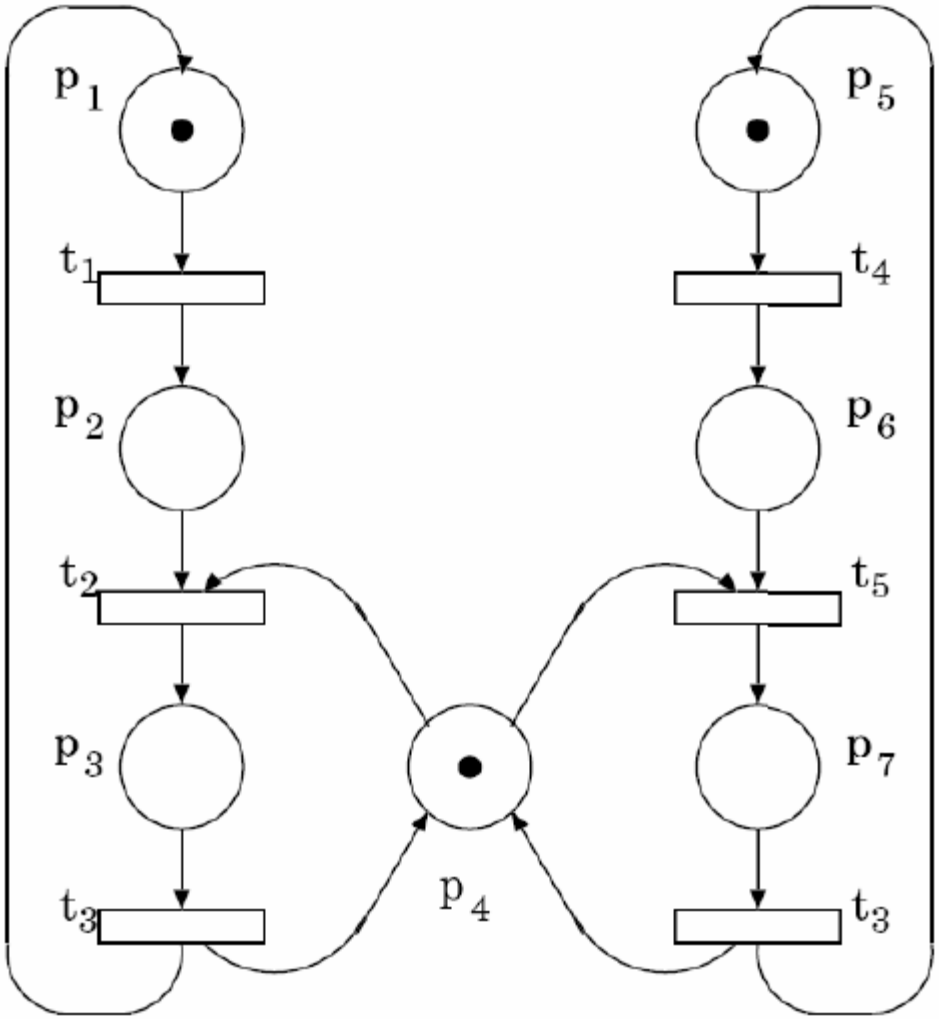


Notación: Modelo de Productor-Consumidor

P1: Dispuesto a producir
T1: Produce elemento
P2: Dispuesto a entregar
T2: Entrega elemento
P3: Dispuesto a recibir
T3: Recibe elemento
P4: Dispuesto a consumir
T4: Consume elemento
P5: Buffer



Notación: Exclusión mutua



T2: Toma control del recurso

P3: Usa el recurso

T3: Devuelve el recurso

P4: Recurso crítico

T5: Toma control del recurso

P7: Usa el recurso

T3: Devuelve el recurso

Java: Todo son hilos

- Todas las aplicaciones de Java usan hilos:
  - Hilo que ejecuta main()
  - Hilos de AWT y Swing, “listener()”, etc.
  - Garbage collector…
- Java facilita el uso de hilos en las aplicaciones:
  - Timer: permite programar una serie de hilos para que se ejecuten al cabo de cierto tiempo, o cada cierto tiempo
  - Servlets y JavaServer Pages (JSPs): para manejar las peticiones remotas de clientes HTTP
  - RMI: llamadas a métodos que se ejecutan remotamente

Ejercicios

- 1.- Buscar ejemplos de código donde haya condiciones de carrera.
- 2.- Localizar información y describir los aspectos más interesantes de las Redes de Petri.
- 3.- Definir el concepto de thread-safe. Poner un ejemplo de código thread-safe y otro que no lo sea.
- 4.- Definir el concepto de liveness en concurrencia. Poner algún ejemplo.