COMPUTACIÓN DE ALTA PERFORMANCE

Curso 2017

Sergio Nesmachnow (sergion@fing.edu.uy)
Santiago Iturriaga (siturria@fing.edu.uy)
Nestor Rocchetti (nrocchetti@fing.edu.uy)

Centro de Cálculo

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

TEMA 3
PROGRAMACIÓN PARALELA

CONTENIDO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

TEMA 3: PROGRAMACIÓN PARALELA

1. Objetivos y enfoques de la computación paralela
2. Modelo de programación paralela
3. Problemas con la computación paralela – distribuida
4. Técnicas de programación paralela
5. Diseño y paralelización de aplicaciones
6. Modelos de comunicación entre procesos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

PROGRAMACIÓN PARALELA

3.1: OBJETIVOS Y ENFOQUES DE LA COMPUTACIÓN PARALELA

OBJETIVOS DE LA PROGRAMACIÓN PARALELA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La computación paralela tiene como principal objetivo la resolución eficiente de instancias de grandes dimensiones de problemas complejos.
- Además, desde el punto de vista del usuario, la computación paralela debe proveer:
  - Transparencia de la arquitectura y de los mecanismos de interconexión
  - Simplicidad de uso y confiabilidad
  - Manejo de excepciones y tolerancia a fallos
  - Mecanismos para asegurar la portabilidad y la ejecución en entornos heterogéneos
  - Soporte para lenguajes tradicionales de alto nivel

ENFOQUES DE LA PROGRAMACIÓN PARALELA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Paralelismo implícito
  - El usuario no controla el control ni el procesamiento
  - Optimizadores de código y compiladores

- Paralelismo explícito
  - El programador es responsable de implementar la lógica del programa paralelo
  - Lenguajes y bibliotecas para el diseño de aplicaciones paralelas
  - Permite explotar las características de cada problema

ESTRATEGIAS DE PROGRAMACIÓN PARALELA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

1. Paralelismo implícito o automático
   - Automático: el usuario no controla el control ni el procesamiento
   - Se parte de un código existente
   - Cambios muy menores en el código
   - Optimizadores de código y compiladores
2. Paralelismo explícito utilizando bibliotecas
   - Se parte de un código existente.
   - Se implementa la lógica utilizando rutinas de bibliotecas ya paralelizadas
   - Permite explotar ciertas características de cada subproblema
3. Paralelismo explícito con rediseño de código
   - No se utiliza código existente
   - Se trabaja rediseñando la aplicación, para tomar la mayor ventaja de las tareas a realizar concurrentemente
   - Se implementa la lógica utilizando bibliotecas de programación paralela

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

PROGRAMACIÓN PARALELA

3.2: MODELO DE COMPUTACIÓN PARALELA

MODELO DE PROGRAMACIÓN PARALELA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Programación en máquina de Von Neumann
  - Secuencia de operaciones (aritméticas, read/write de memoria, avance de program counter)
  - Abstracciones de datos e instrucciones
  - Técnicas de programación modular

- Programación en máquina paralela
  - Incluye complicaciones adicionales:
    - Multiejecución simultánea
    - Comunicaciones y sincronización
  - La modularidad pasa a ser fundamental, para manejar la (potencialmente) enorme cantidad de procesos en ejecución simultánea

MODELO CONCEPTUAL DE PARALELISMO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El paradigma de diseño y programación difiere del utilizado para diseñar y programar aplicaciones secuenciales.
  - Una buena estrategia de división del problema puede determinar la eficiencia de un algoritmo paralelo.
  - Es importante considerar el tipo y la disponibilidad de hardware.

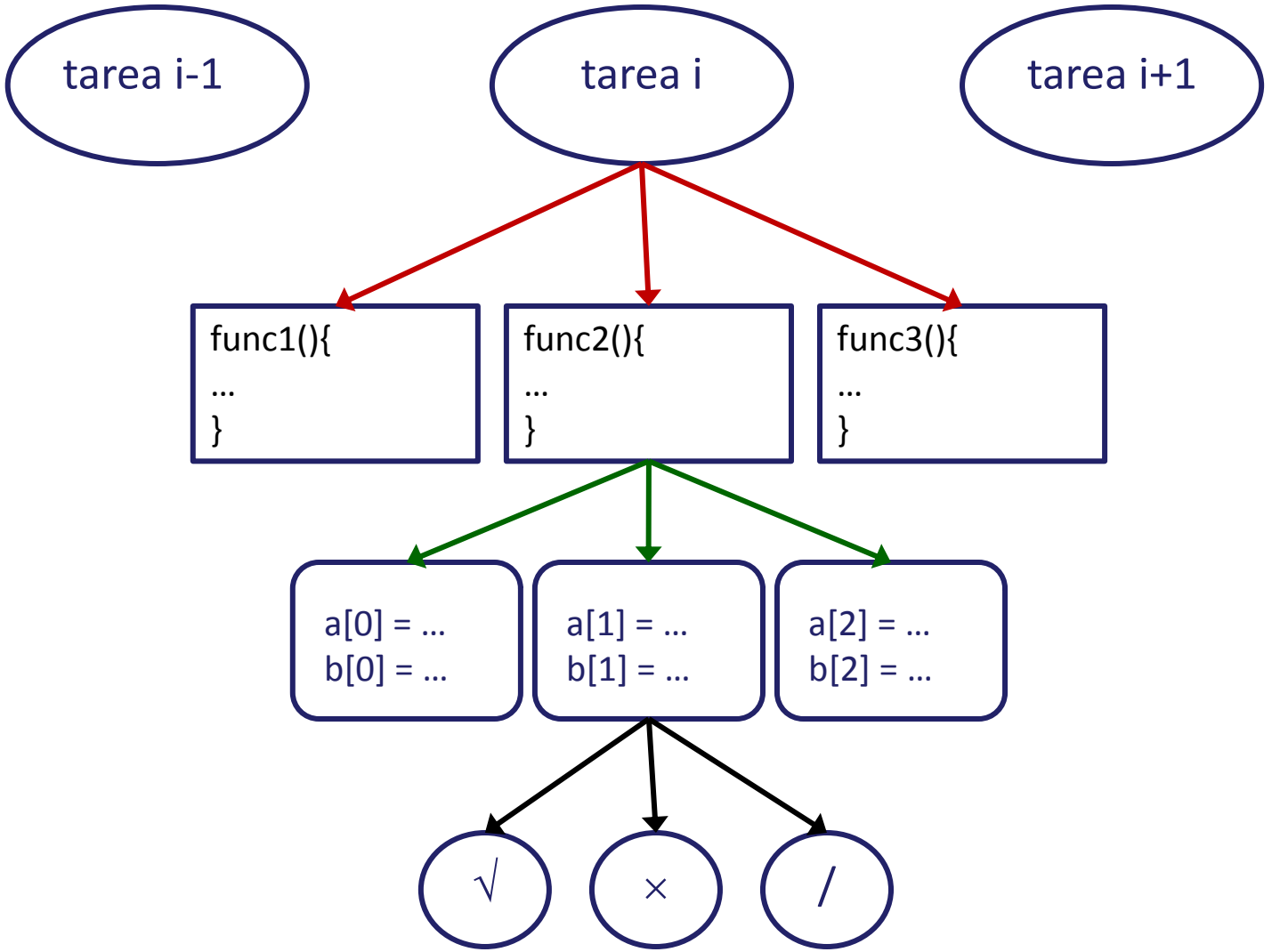
- Respecto a los mecanismos de comunicación entre procesos, existen dos paradigmas de computación paralela
  - MEMORIA COMPARTIDA
  - PASAJE DE MENSAJES
  - Existen también MODELOS HÍBRIDOS, que combinan ambas técnicas.

NIVELES DE PARALELISMO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El paralelismo puede aplicarse:
  1. A nivel intrainstrucción (hardware, pipelines)
  2. A nivel interinstrucción (SO, optimización de código, compilador)
  3. A nivel de procedimientos o tareas
     - Algorítmico funcional
  4. A nivel de programas o trabajos
     - Algorítmico tareas
- Los niveles 1 y 2 corresponden a cursos de arquitectura de sistemas y sistemas operativos
- A lo largo del presente curso, básicamente consideraremos los niveles 3 y 4 de aplicación de las técnicas de paralelismo (enfocado al diseño y programación de algoritmos paralelos)

NIVELES DE PARALELISMO



Algorítmico tareas

Algorítmico funcional

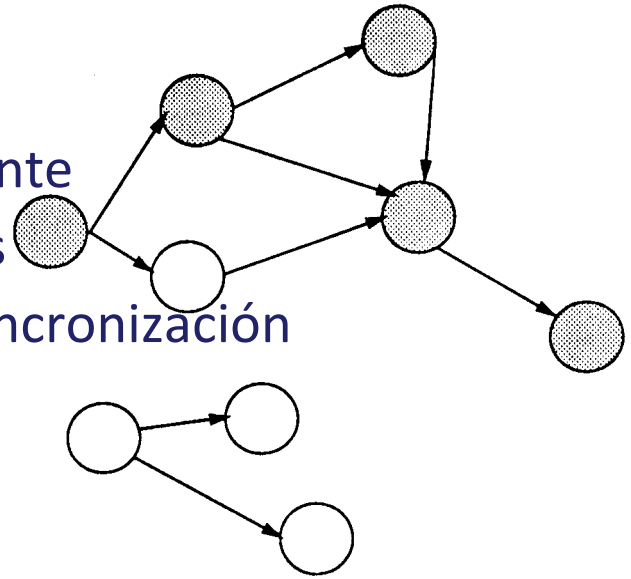
Inter-instrucción

Intra-instrucción

MODELO CONCEPTUAL DE PARALELISMO

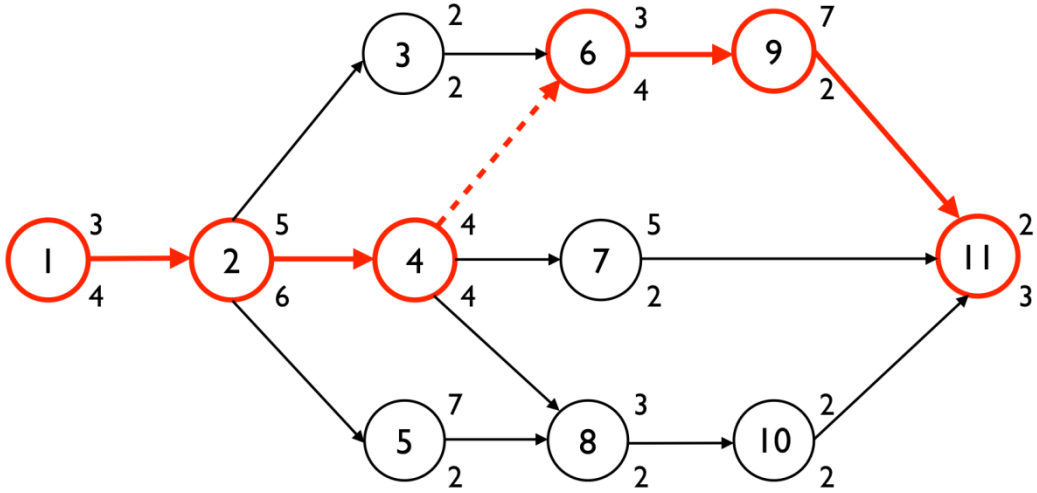
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- GRAFOS DIRIGIDOS ACICLICOS (ADGs)
  - Los nodos representan tareas o procesos (partes de código que ejecutan secuencialmente)
  - Las aristas representan dependencias (algunas tareas preceden a otras)
- El problema a resolver se divide en tareas a ejecutar cooperativamente en múltiples procesadores.
- El diseño e implementación de la aplicación debe:
  - Definir tareas que pueden ejecutar concurrentemente
  - Lanzar la ejecución y detener la ejecución de tareas
  - Implementar los mecanismos de comunicación y sincronización



MODELO CONCEPTUAL DE PARALELISMO

- Las dependencias entre tareas imponen limitaciones al nivel de paralelismo
  - Dependencia de datos
  - Sincronizaciones
- El camino crítico en el grafo de tareas permite determinar el mínimo tiempo posible de ejecución
  - Técnicas de investigación operativa permiten determinar los niveles de paralelismo (Gantt, planificación)



DISEÑO DE ALGORITMOS PARALELOS

- No siempre es una buena decisión partir de un algoritmo secuencial (“paralelizar” una aplicación)
- En ocasiones será necesario diseñar un nuevo algoritmo, que puede ser muy diferente al secuencial
- Resumiendo las etapas de diseño de aplicaciones paralelas:
  - Identificar el trabajo que puede hacerse en paralelo
  - Dividir el trabajo y los datos entre los procesadores
  - Resolver los accesos a los datos, las comunicaciones entre procesos y las sincronizaciones
  - Asignar recursos de cómputo a los procesos que ejecutarán en paralelo

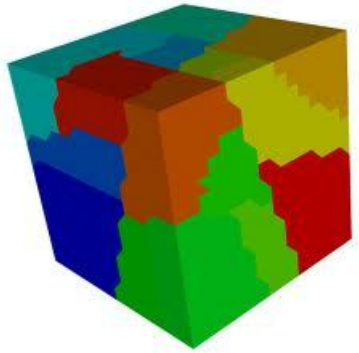
DESCOMPOSICIÓN – ASIGNACIÓN – ORQUESTACIÓN – MAPEO

DISEÑO DE ALGORITMOS PARALELOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- DESCOMPOSICIÓN
  - Identifica los procesos concurrentes y decide el nivel y la forma en la cual se explotará la concurrencia
  - Decisiones:
    - Cantidad de tareas
    - Tareas estáticas o dinámicas
    - Criterios de división y utilización de recursos

- ASIGNACIÓN
  - Asigna los datos a los procesos concurrentes
  - Criterios de asignación:
    - Estáticos o dinámicos
    - Balance de cargas
    - Reducción de costos de comunicación y sincronización



DISEÑO DE ALGORITMOS PARALELOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- ORQUESTACIÓN
  - Se toman decisiones sobre los parámetros de arquitectura, el modelo de programación, los lenguajes o bibliotecas a utilizar
  - Resuelve:
    - Las estructuras de datos, la localidad de referencias
    - La optimización de comunicaciones y sincronizaciones

- MAPEO (SCHEDULING)
  - Asigna los procesos a los recursos (procesadores).
  - Criterios de asignación:
    - Desempeño
    - Utilización
    - Reducción de costos de comunicación y sincronización
  - El mapeo puede ser estático, dinámico o adaptativo

DISEÑO DE ALGORITMOS PARALELOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Los mecanismos para definir, controlar y sincronizar tareas deben formar parte del lenguaje a utilizar o ser intercalados por el compilador.
- Estos mecanismos deben permitir especificar:
  - El control de flujo de ejecución de tareas
  - El particionamiento de los datos
- Algunos ejemplos:
  - Definición y ejecución de tareas: parbegin-parend (Pascal concurrente), task (Ada), spawn (PVM), fork & join (C)
  - Comunicación y sincronización: pipes, semáforos, barreras (en memoria compartida), mensajes sincrónicos (rendezvouz Ada) y mensajes asincrónicos (C/C++, PVM, MPI) (en memoria distribuida)
  - El particionamiento de los datos es una tarea usualmente asignada al diseñador del algoritmo paralelo
- El modelo se complica por características peculiares de la computación paralela – distribuida

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

PROGRAMACIÓN PARALELA

3.3: PROBLEMAS DE LA COMPUTACIÓN PARALELA-DISTRIBUIDA

PROBLEMAS
DE LA COMPUTACIÓN PARALELA-DISTRIBUIDA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- NO DETERMINISMO EN LA EJECUCIÓN
  - Los programas tienen muchos estados posibles
  - Impide utilizar herramientas de diseño del estilo de los "diagramas de flujo (en el tiempo)" para especificar secuencias de control

- CONFIABILIDAD
  - Varios componentes del sistema pueden fallar:
    - nodos, interfaces, tarjetas, caches, bridges, routers, repeaters, gateways, medios físicos
  - Problemas de utilizar equipamiento no dedicado:
    - problemas de uso y tráfico (propio y ajeno), etc.
    - no repetibilidad de condiciones de ejecución

PROBLEMAS
DE LA COMPUTACIÓN PARALELA-DISTRIBUIDA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- SEGURIDAD
  - Conexión y acceso a equipos remotos
  - Accesos a datos distribuidos
- DIFICULTAD DE ESTIMAR LA PERFORMANCE DE UNA APLICACIÓN
  - Como consecuencia del no determinismo en la ejecución
  - Deben utilizarse criterios estadísticos
- DIFICULTAD DE ANALIZAR Y VERIFICAR PROGRAMAS
  - Es difícil reproducir escenarios (múltiples estados, asincronismo)
  - Los datos y procesos no están centralizados
  - Implica la necesidad de un “debugger” en cada nodo utilizado

PROBLEMAS
DE LA COMPUTACIÓN PARALELA-DISTRIBUIDA

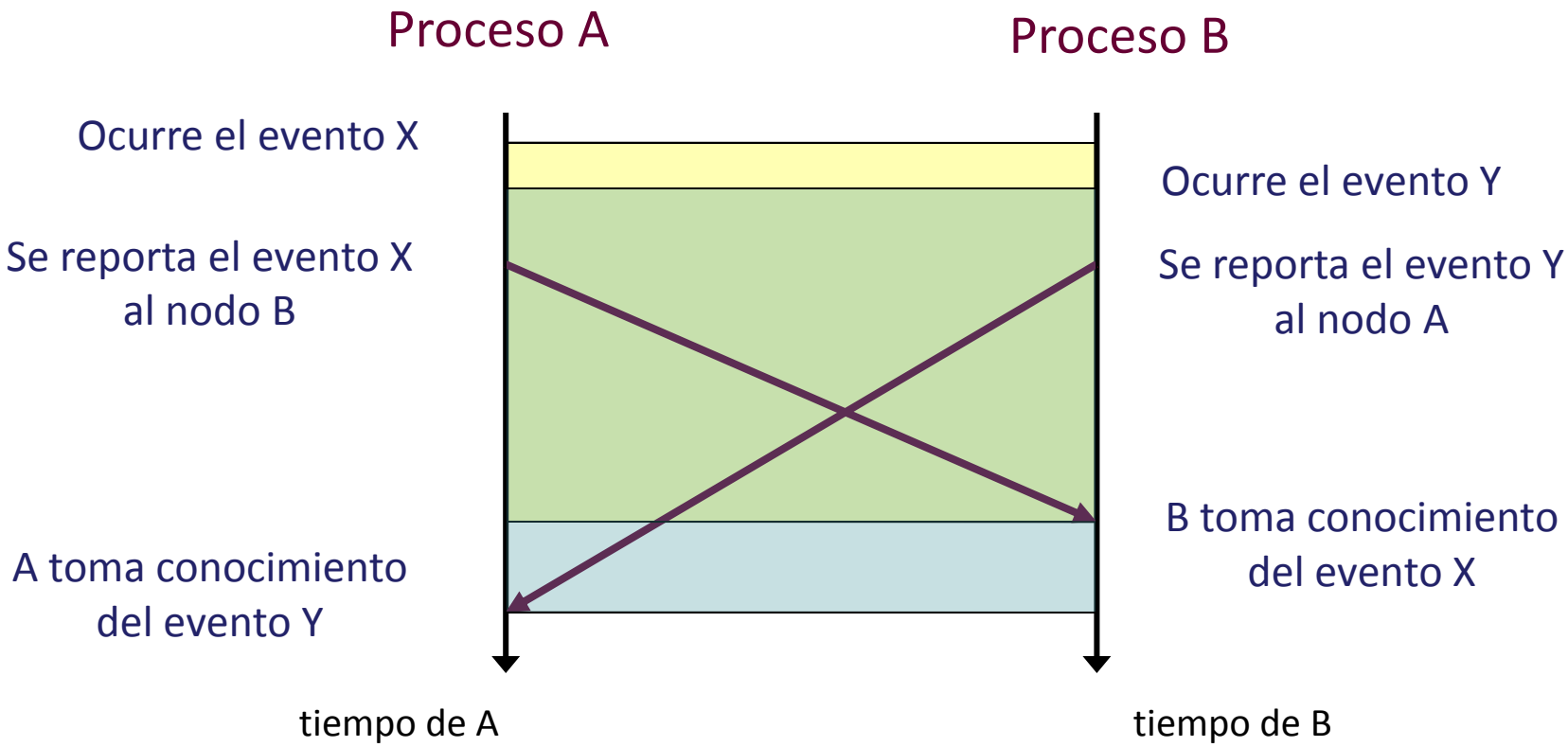
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- INCOMPATIBILIDAD POTENCIAL ENTRE PRODUCTOS Y PLATAFORMAS (para sistemas heterogéneos)
  - Sistemas Operativos: Linux y otros (variantes personalizadas de Unix, Windows, etc.)
  - Protocolos de comunicaciones: TCP/IP, protocolos propietarios
  - Herramientas de software y lenguajes: PVM/MPI, C/Java, bibliotecas de threads, etc....

- USO RACIONAL DE RECURSOS DE CÓMPUTO
  - En sistemas no dedicados

ASINCRONISMO

DEJA DE EXISTIR EL CONCEPTO DE "TIEMPO UNIVERSAL"



"el evento X sucedió antes que el evento Y"

"el evento Y sucedió antes que el evento X"

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

PROGRAMACIÓN PARALELA

3.4: TÉCNICAS DE PROGRAMACIÓN PARALELA

TÉCNICAS DE PROGRAMACIÓN PARALELA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Introducción
- Técnica de descomposición de dominio
- Técnica de descomposición funcional
- Técnicas de paralelismo optimista
- Técnicas Híbridas
- Ejemplos

TÉCNICAS de COMPUTACIÓN PARALELA/DISTRIBUIDA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Las técnicas de programación paralela/distribuida aplican estrategias de DESCOMPOSICIÓN o PARTICIONAMIENTO de los datos y del cómputo, que permiten dividir un problema en subproblemas de menor complejidad, a resolver en paralelo
- El objetivo primario de la descomposición será dividir en forma equitativa tanto los cálculos asociados con el problema como los datos sobre los cuales opera el algoritmo

- ¿ Cómo lograr el objetivo de la descomposición ?
  - Definir al menos un orden de magnitud más de tareas que de procesadores disponibles (utilización de recursos)
  - Evitar cálculos y almacenamientos redundantes.
  - Generar tareas de tamaño comparable.
  - Generar tareas escalables con el tamaño del problema.
  - Considerar varias alternativas de descomposición, en caso de ser posible.

- Según se enfoque principalmente en la descomposición de datos o de tareas, resultará una técnica diferente de programación paralela.

- Las técnicas más difundidas son las de descomposición de dominio y descomposición funcional.

DESCOMPOSICIÓN de DOMINIO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Se concentra en el particionamiento de los datos del problema (*paralelismo de datos - data parallel*)
- Se trata de dividir los datos en piezas pequeñas, de (aproximadamente) el mismo tamaño
- Luego se dividen los cálculos a realizar, asociando a cada operación con los datos sobre los cuales opera
- Los datos a dividir pueden ser:
  - La entrada del programa
  - La salida calculada por el programa
  - Datos intermedios calculados por el programa

DESCOMPOSICIÓN de DOMINIO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Si bien no existe una regla general para determinar como realizar la división de datos, existen algunas sugerencias obvias dadas por:
  - La estructura o “geometría” del problema.
  - La idea de concentrarse primero en las estructuras de datos más grandes o las accedidas con mayor frecuencia.
- Ejemplo 1: paralelizar un join entre tablas en una BD.
  - Debe particionarse la tabla más grande.
- Ejemplo 2 : Procesamiento de imágenes
  - División de dominios de cálculo.
  - Mismo programa en cada dominio.
  - Comunicación necesaria para cálculos en los bordes.
- La técnica de descomposición de dominio se asocia comúnmente con la estrategia de Divide&Conquer.

DESCOMPOSICIÓN DE DOMINIO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

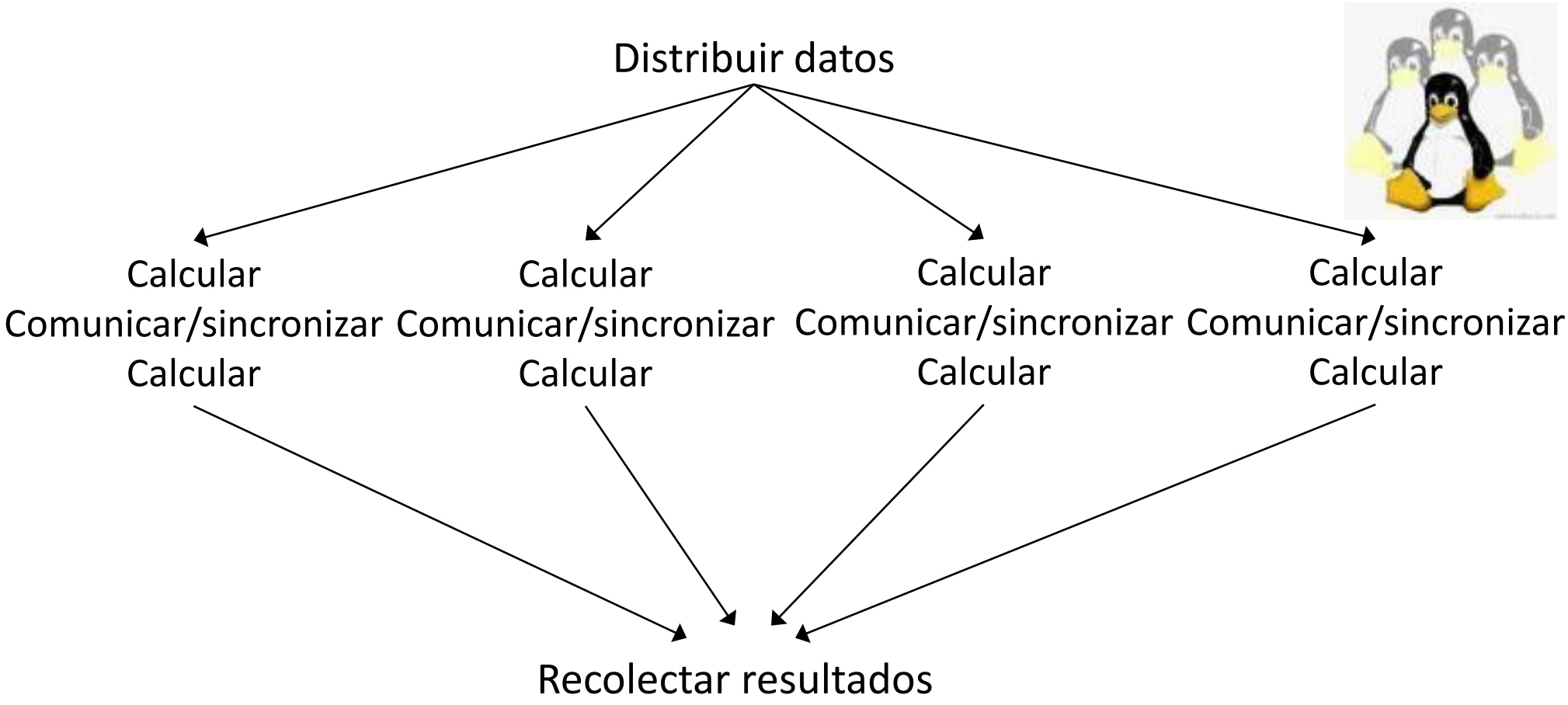
- Aplicable en los modelos SIMD y SPMD de programas paralelos:
  - SIMD: Single Instruction Multiple Data
  - SPMD: Single Program Multiple Data

- Un único programa [eventualmente con varias copias en ejecución] controla el procesamiento

- En general
  - SIMD sobre arquitecturas de memoria compartida
  - SPMD sobre arquitecturas de memoria distribuida

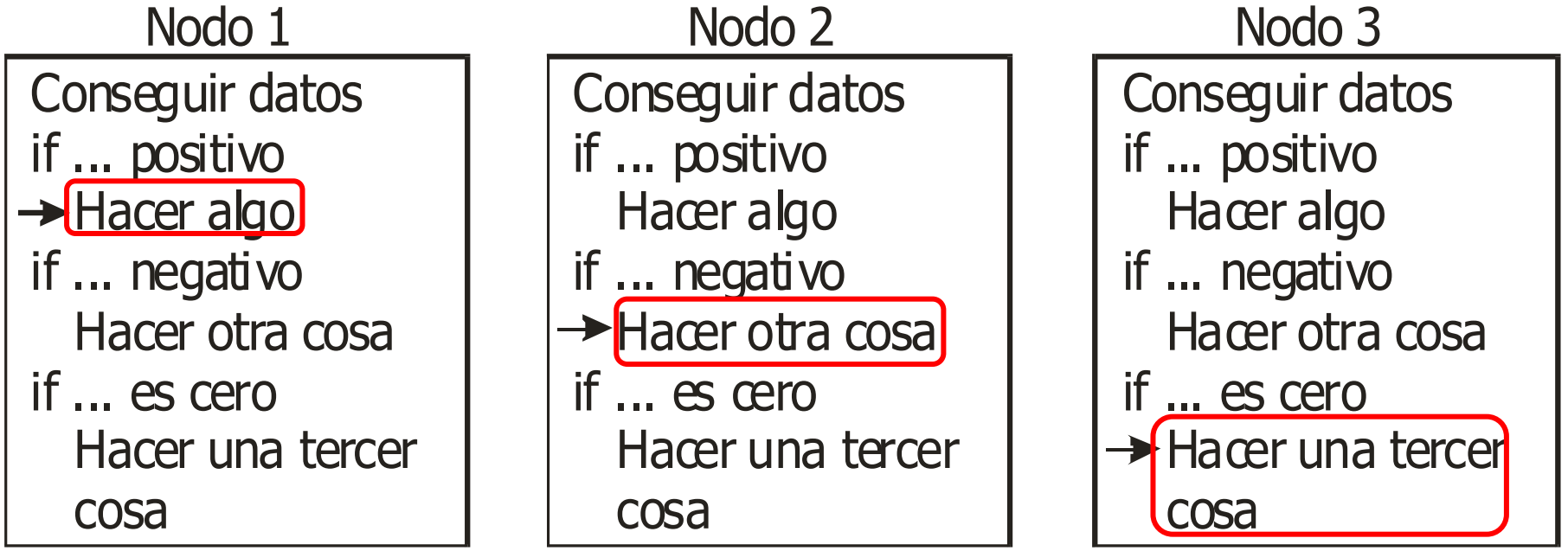
DESCOMPOSICIÓN de DOMINIO

- Modelo de programación SPMD (Single Program, Multiple Data)



DESCOMPOSICIÓN de DOMINIO

- Modelo de programación SPMD (Single Program, Multiple Data)
- Un mismo programa se ejecuta sobre diferentes conjuntos de datos



Todos los nodos ejecutan el mismo programa,
pero no necesariamente las mismas instrucciones

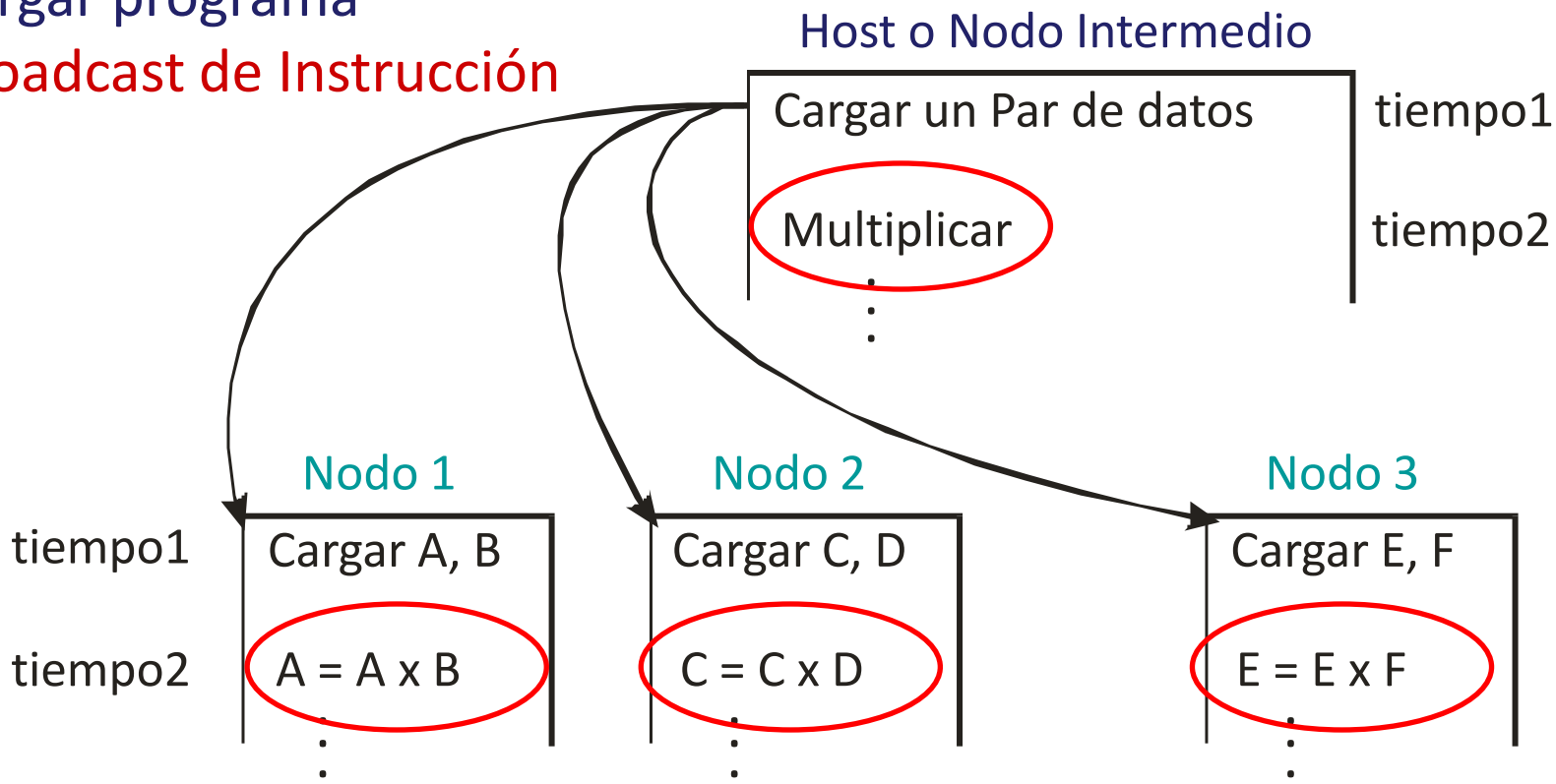
- Es un modelo muy utilizado en clusters y sistemas grid

DESCOMPOSICIÓN DE DOMINIO

DESCOMPOSICIÓN DE DOMINIO

- Modelo de programación SIMD (Single Instruction, Multiple Data)

1. Cargar programa
2. Broadcast de Instrucción



Los nodos reciben instrucciones y ejecutan

DESCOMPOSICIÓN DE DOMINIO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Modelo de programación SIMD (Single Instruction, Multiple Data)
- Aplicable en la resolución de problemas físicos homogéneos:
  - Estructura geométrica regular, con interacciones limitadas.
  - La homogeneidad permite la distribución uniforme de datos.
  - Las comunicaciones y sincronizaciones son reducidas y su costo es proporcional al tamaño de las fronteras entre los subdominios de datos.
  - El patrón de comunicaciones es usualmente estructurado y altamente predecible.
- Si el problema es no-homogéneo, se requieren mecanismos adicionales para la distribución de datos y el balance de cargas.
- El paradigma es altamente sensible al fallo en algún proceso:
  - Puede causar un deadlock, o incluso una caída del sistema.
  - Para resolverlo, deben aplicarse técnicas de tolerancia a fallos.
- Es el modelo adoptado por la biblioteca MPI (Message Passing Interface)

DESCOMPOSICIÓN DE DOMINIO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Método de Strassen para multiplicar matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

Método tradicional

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{aligned}$$

Complejidad

$$n^3 = n^{\log_2 8}$$

Método de Strassen

$$\begin{aligned} \mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\ \mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\ \mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\ \mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\ \mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\ \mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\ \mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2}) \end{aligned}$$

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\ \mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\ \mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\ \mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6 \end{aligned}$$

Complejidad $n^{\log_2 7} \approx n^{2.807}$

Referencias:

http://mathworld.wolfram.com/StrassenFormulas.html , http://en.wikipedia.org/wiki/Strassen_algorithm

DESCOMPOSICIÓN DE DOMINIO

- SETI@HOME

Distribuir datos

Calcular
Enviar resultados

Calcular
Enviar resultados

Calcular
Enviar resultados

Calcular
Enviar resultados

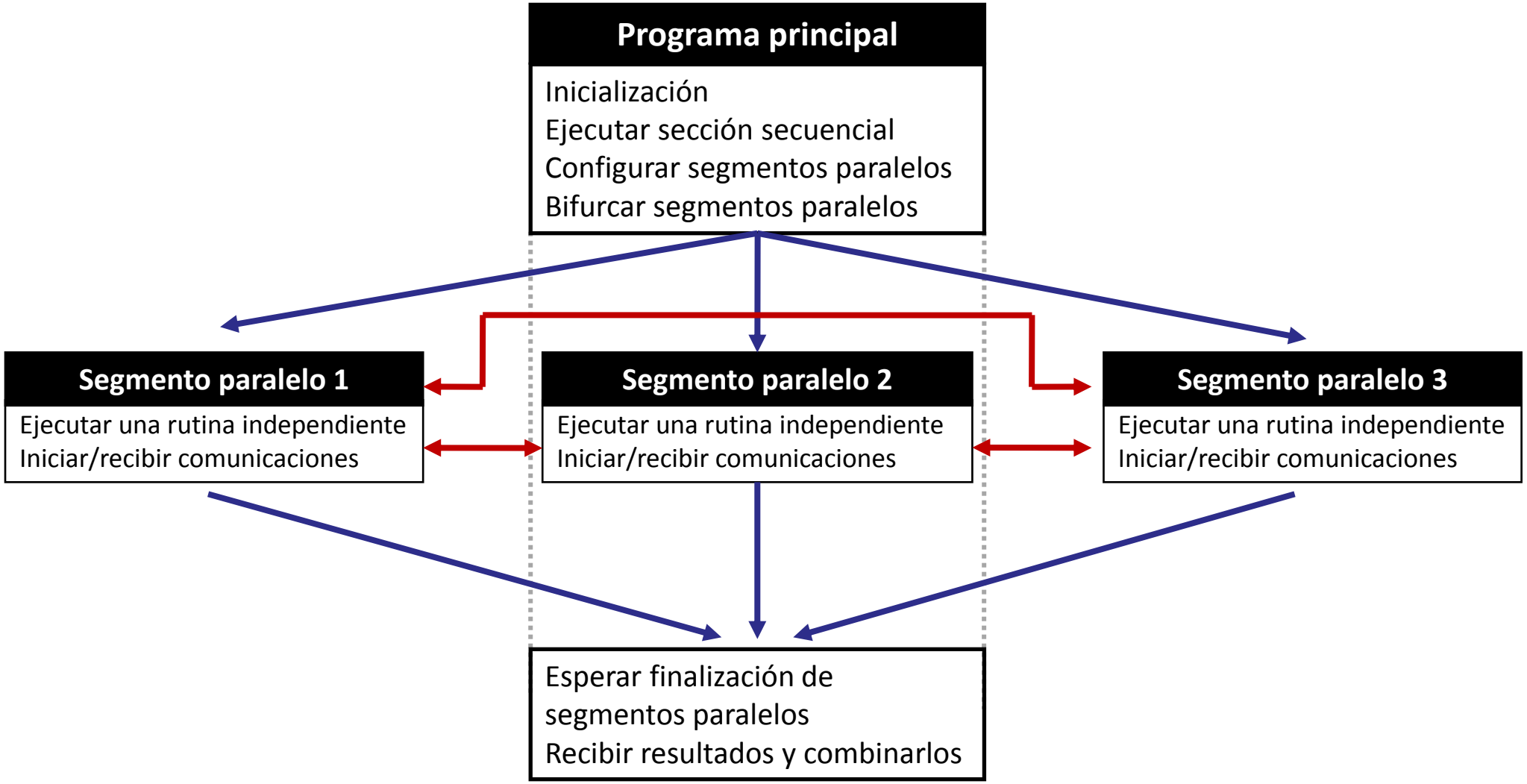
Recolectar resultados

DESCOMPOSICIÓN FUNCIONAL

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

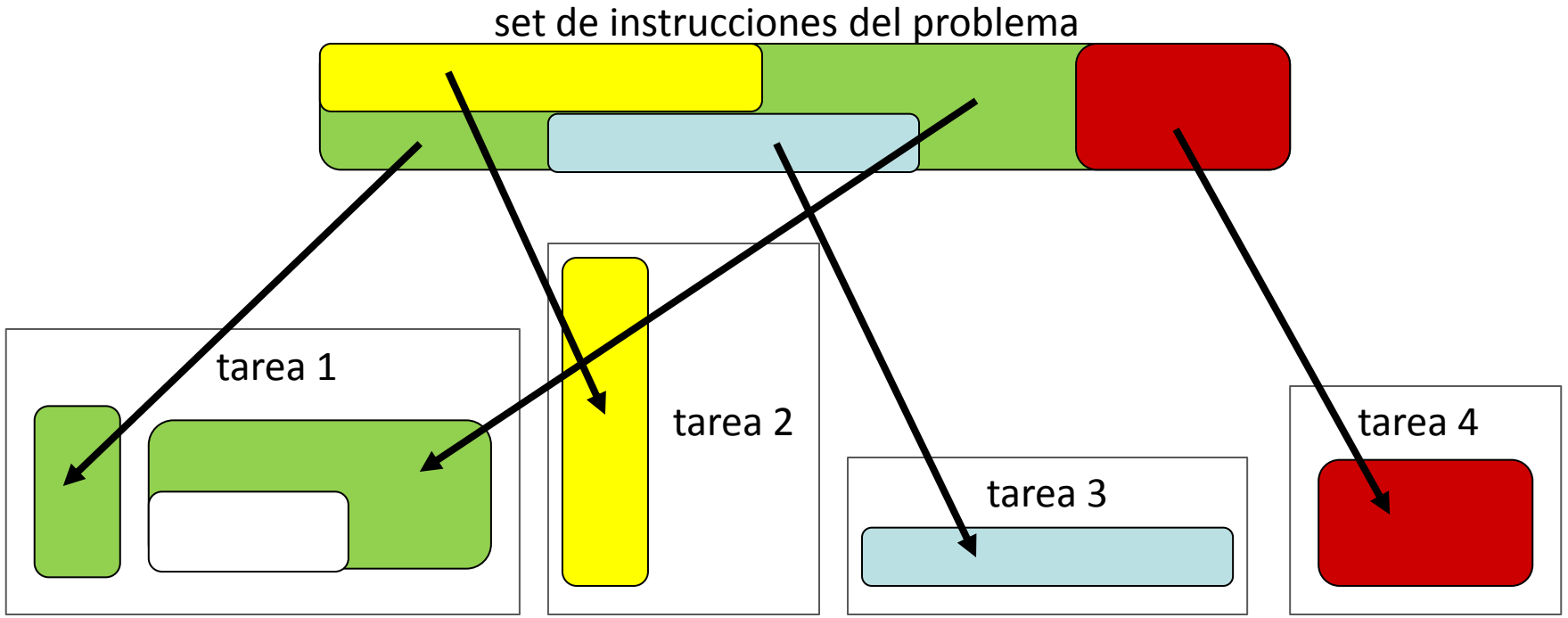
- Se concentra en el particionamiento de las operaciones del problema (*paralelismo de control - control parallel*).
- Se trata de dividir el procesamiento en tareas disjuntas.
- Luego se examinan los datos que serán utilizados por las tareas definidas.
- Si los datos son disjuntos, resulta un PARTICIONAMIENTO COMPLETO.
- Si los datos NO son disjuntos, resulta un PARTICIONAMIENTO INCOMPLETO (el caso más usual). Se requiere replicar los datos o comunicarlos entre los procesos asociados a las diferentes tareas.

DESCOMPOSICIÓN FUNCIONAL



DESCOMPOSICIÓN FUNCIONAL

- Casos típicos:
  - Distribuir código para asociar requerimientos de procesos a recursos locales
    - Cada tarea trabaja temporalmente con sus datos locales, pero se requiere comunicación para lograr la cooperación
  - Procesamiento de datos en bases de datos relacionales
    - Diferentes tareas realizan diferentes operaciones



PARALELISMO OPTIMISTA

- Realizar operaciones adicionales previendo que deban ser ejecutadas en el futuro (*look-ahead execution*)
  - Bajo la hipótesis de que hay recursos disponibles para ejecutarlas.

```
if condicion then
        Funcion1()
else
        Funcion2()
end if
```

  - Evaluar Función1() y Funcion2() de antemano, independientemente del valor de la condición.
- Típico en aplicaciones de tiempo real.
  - Ejemplo : sistemas que requieren conexión.
- Típico mecanismo para proveer tolerancia a fallos.
  - En aplicaciones distribuidas en Internet.

MODELOS HÍBRIDOS

- Incluyen dos o más tipos de programación paralela para la resolución de una misma aplicación.

- Comúnmente utilizados en programas paralelos distribuidos en Internet (donde casi siempre existe la posibilidad de conseguir recursos ociosos adicionales).
  - Se combina una estrategia “tradicional” de descomposición con una estrategia de paralelismo optimista, para mejorar la eficiencia o proveer mecanismos para tolerancia a fallos.
  - Las principales ventajas son eficiencia y robustez, aunque el manejo de los distintos modelos de paralelismo puede incluir complejidades adicionales.

ALGUNOS EJEMPLOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Cracking de passwords y claves publicas
  - www.distributed.net (R.I.P.)
- Calculo de mínimos globales
  - Función ‘constr’ de Matlab, algoritmos de optimización
- Multiplicación de matrices y resolución de sistemas lineales
  - Aplicado a métodos paralelizables: Jacobi, gradiente conjugado, etc.
- Aplicación de filtros a imágenes
  - Aplicaciones uniforme modelo SIMD
- Modelos de corrientes del Río de la Plata
  - Proyectos en Facultad: PTIDAL, RMA-10, RMA-11
- Procesamiento gráfico
  - PVMPOV para visualización y generación de animaciones
- Sistemas de procesamiento distribuido
  - Procesamiento transaccional en Internet

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

PROGRAMACIÓN PARALELA

3.5: DISEÑO Y PARALELIZACIÓN DE APLICACIONES

DISEÑO Y “PARALELIZACIÓN”

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Introducción
- Paralelizando aplicaciones existentes
- Estrategias de descomposición y estructura temporal
- Modelos de comunicación ente procesos:
  - Modelo maestro-esclavo (master-slave)
  - Modelo cliente-servidor
  - Modelo peer-to-peer

DISEÑO Y "PARALELIZACIÓN"

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- No siempre se dispone de recursos (en especial de tiempo) para diseñar una aplicación paralela-distribuida de acuerdo a los criterios y técnicas de programación especificadas
- En múltiples ocasiones, se trata de obtener resultados de eficiencia computacional aceptable adaptando programas secuenciales a los modelos de programación paralela
  - "Paralelizar" una aplicación existente
- Problemas de este enfoque:
  - Se utiliza un código existente que no fue diseñado para ejecutar sobre múltiples recursos computacionales
  - Los códigos existentes suelen ser enmarañados, y se sufre la "contaminación" del código heredado

PARALELIZANDO APLICACIONES EXISTENTES

- Deben analizarse varios aspectos:
  - ¿Existe una partición funcional evidente?
    - El código modular es el más fácil de paralelizar.
  - ¿Existe forma de particionar los datos?
    - Si existe, ¿cuál es la relación entre procesamiento y datos?
  - ¿Existen muchas variables globales?
    - Cuidado !!! El manejo de recursos compartidos es un problema a resolver.
    - Considerar el uso de un servidor de variables globales.

PARALELIZANDO APLICACIONES EXISTENTES

- Deben analizarse varios aspectos (continuación):
  - ¿Es la seguridad un requerimiento importante?
    - Autenticación en ambientes distribuidos
  - ¿Qué nivel de tolerancia a fallas se requiere?
    - En los recursos de cómputo (reejecución de tareas)
    - En la red (reenvío de mensajes)
  - ¿La aplicación utiliza otras formas de IPC?
    - No todos estos servicios existen al trabajar en entornos de memoria distribuida

ESTRATEGIAS DE DESCOMPOSICIÓN

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En *Parallel Programming for Scientist and Engineers*, G. Wilson identifica cinco técnicas de descomposición:
  1. Descomposición geométrica: el dominio de datos del problema se divide en subdominios y cada proceso ejecuta un [mismo] algoritmo sobre cada subdominio. Corresponde al modelo de descomposición de dominio.
  2. Descomposición funcional: se divide la aplicación en "fases", que aplican diferentes algoritmos que cooperativamente permiten resolver el problema. Incluye a la descomposición funcional y las estrategias de pipeline.
  3. Descomposición iterativa: para aplicaciones basadas en ejecución de ciclos donde cada iteración puede realizarse independientemente. Suele aplicarse un modelo de paralelismo maestro-esclavo.
  4. Descomposición recursiva: el problema original se divide en varios subproblemas de menor complejidad que se resuelven en paralelo. Corresponde al enfoque Divide&Conquer.
  5. Descomposición especulativa: se ejecutan N "intentos" [con diferentes métodos o con diferentes parámetros] y se utiliza(n) el/los mejor(es) resultado(s) [calidad/eficiencia], descartándose los restantes. Corresponde a los modelos de paralelismo optimista y de computación tolerante a fallos.

ESTRUCTURA TEMPORAL

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La estructura temporal de los problemas permite diferenciar entre:
  - Problemas sincrónicos: que se resuelven aplicando modelos de computación uniforme del tipo SIMD o SPMD sincrónico (paralelismo de datos)
    - Ejemplo: procesamiento multimedia
  - Problemas débilmente asincrónicos: que introducen pequeñas secciones asincrónicas en el modelo de paralelismo de datos (SPMD asincrónico)
    - Ejemplo: cómputos iterativos en dominios irregulares
  - Problemas asincrónicos: caracterizados por diversos procesos en ejecución asincrónica, usualmente siguiendo el modelo de descomposición funcional
    - Ejemplos: cómputos complejos en dominios irregulares, simulaciones basadas en eventos
  - Problemas trivialmente paralelos: caracterizados por la ejecución independiente de diversos procesos o múltiples copias de un mismo proceso, sin comunicación entre sí
    - Ejemplo: cómputos simples sobre grandes conjuntos de datos. Modelos desconexos o maestro-esclavo

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

PROGRAMACIÓN PARALELA

3.6: MODELOS DE COMUNICACIÓN ENTRE PROCESOS

MODELOS DE COMUNICACIÓN ENTRE PROCESOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Utilizados para comunicar y/o sincronizar procesos paralelos
- Modelos de comunicación:
  - Modelo maestro-esclavo (master-slave)
  - Modelo cliente-servidor
  - Modelo peer-to-peer

MODELO MAESTRO-ESCLAVO

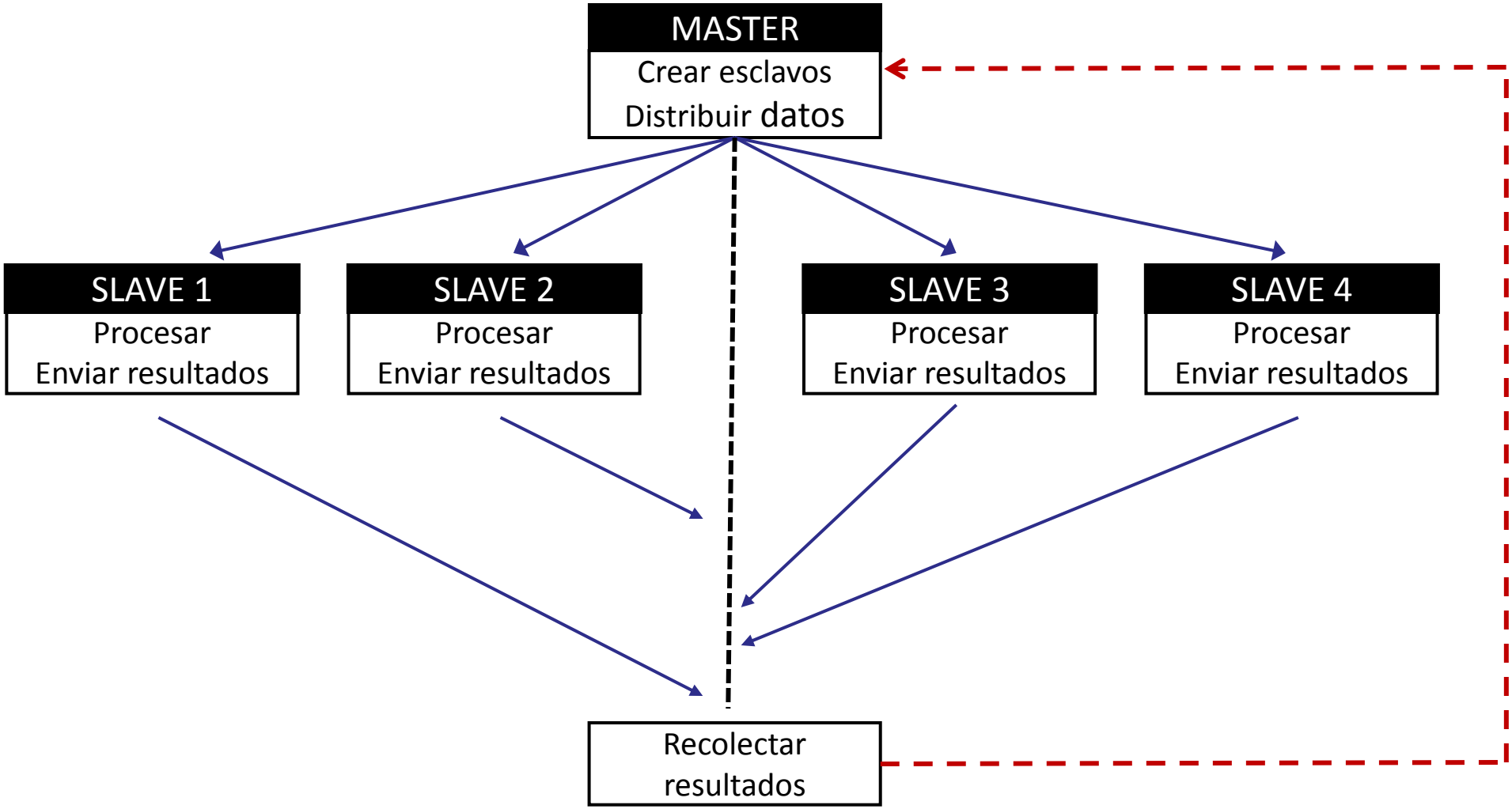
- Master-slave
  - Uno de los paradigmas de comunicación más sencillo.
  - Se basa en generar un conjunto de subproblemas y procesos que los resuelven.
  - Utiliza un proceso distinguido (master, maestro) y uno o varios procesos usualmente idénticos (slaves, esclavos).
  - El proceso master controla a los procesos slave.
  - Los procesos slave procesan.

MODELO MAESTRO-ESCLAVO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El proceso master lanza a los esclavos y les envía datos.
- Luego de establecida la relación master/slave, la jerarquía impone la dirección del flujo de control del programa (siempre del master sobre los slaves).
- La única comunicación desde los esclavos es para enviar los resultados de la tarea asignada.
- Habitualmente no hay dependencias fuertes entre las tareas realizadas por los esclavos (poca o nula comunicación entre esclavos).
- La manera más eficiente de implementarlo es con un pool de procesos esclavo.

MODELO MAESTRO-ESCLAVO



MODELO MAESTRO-ESCLAVO

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Clasificaciones
  - Sincrónico o asincrónico
    - Operativa sincrónica entre esclavos o no
  - Activo o pasivo
    - El maestro procesa o no procesa datos (solo control y monitoreo)
  - De procesos estáticos o incremental
    - Respecto al mecanismo de creación de procesos por parte del master
  - Estático o dinámico
    - Respecto al mecanismo de asignación de datos por parte del master

MODELO MAESTRO-ESCLAVO

- Características:
  - El paradigma es sencillo, pero requiere programar el mecanismo para lanzar tareas, la distribución de datos, el control del master sobre los slaves y la sincronización en caso de ser necesaria.
  - La selección de recursos es fundamental para la performance de las aplicaciones master/slave.
  - Métodos de balance de cargas en modelos estáticos y dinámicos.
- Típicas aplicaciones:
  - Técnicas de simulación Monte Carlo.
  - Aplicaciones criptográficas.
  - Procesamiento de datos masivos.
  - Modelos de partición sencilla de tareas y datos (SETI@home).
  - Modelos fork–join (C).
  - Modelos spawn con creación de procesos dinamicos.

MODELO MAESTRO-ESCLAVO

- Proceso maestro (modelo pasivo, sin pool de esclavos)

```
main ( ) {
   Inicializar;
   Fijar número de esclavos (num_esclavos);
   para i=0 hasta i=num_esclavos {
     datos=Determinar_datos(i);
     Lanzar_tarea(i, datos);
   }
  respuestas=0;
  para i=0 hasta i=num_esclavos {
    res=Obtener_Respuesta();
    resultado=Procesar_resultado(res);
  }
  Desplegar_resultado(resultado);
}
```

MODELO MAESTRO-ESCLAVO

- Proceso maestro (modelo pasivo, con pool de esclavos)

```
main ( ) {
   Inicializar y fijar número de esclavos (num_esclavos);
   para i=0 hasta i=num_esclavos {
     Lanzar_tarea(i, datos);
   }
   mientras (no fin){
     para i=0 hasta i=num_esclavos {
                datos=Determinar_datos(i);
       Asignar_Datos(i)
     }
   respuestas=0;
    Obtener_Respuestas(num_esclavos);
    resultado=Procesar_resultado_parcial(res);
   }
  Desplegar_resultado(resultado);
}
```

MODELO MAESTRO-ESCLAVO

- Proceso esclavo

```
main() {
  /* El proceso esclavo se crea con una referencia
     al proceso master, o puede obtenerla mediante
     una invocación a una función específica */
  datos=Esperar_datos(master);
  Procesar(datos);
  /* No hay comunicación entre procesos esclavos */
  /* De existir, se incluiría aquí, o dentro del
     procesamiento */
  Enviar_resultado(master);
}
```

- Si el esclavo fuera un proceso demonio (siempre en ejecución), tendría un loop infinito y solo finalizaría cuando se lo indique el master

MODELO CLIENTE-SERVIDOR

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Modelo que identifica dos clases de procesos, caracterizados por el tipo de tareas que realizan y servicios que brindan.
- Procesos de una clase (los *clientes*) que realizan peticiones solicitando servicios a procesos de otra clase (los *servidores*), que actúan como proveedores de recursos o servicios y atienden los pedidos de aquellos.
- El paradigma puede ser utilizado para comunicar procesos que ejecutan en un único equipo, pero es una idea potencialmente más poderosa para comunicar procesos distribuidos en una red.
- En el sistema cliente-servidor, las gestiones se concentran en el servidor, que centraliza los requerimientos provenientes de los clientes, maneja esquemas de prioridad de atención, almacena datos de uso público y de uso restringido, instrumenta políticas de acceso a los datos, etc.
- La capacidad de procesamiento se encuentra distribuida entre los clientes y los servidores.

MODELO CLIENTE-SERVIDOR

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

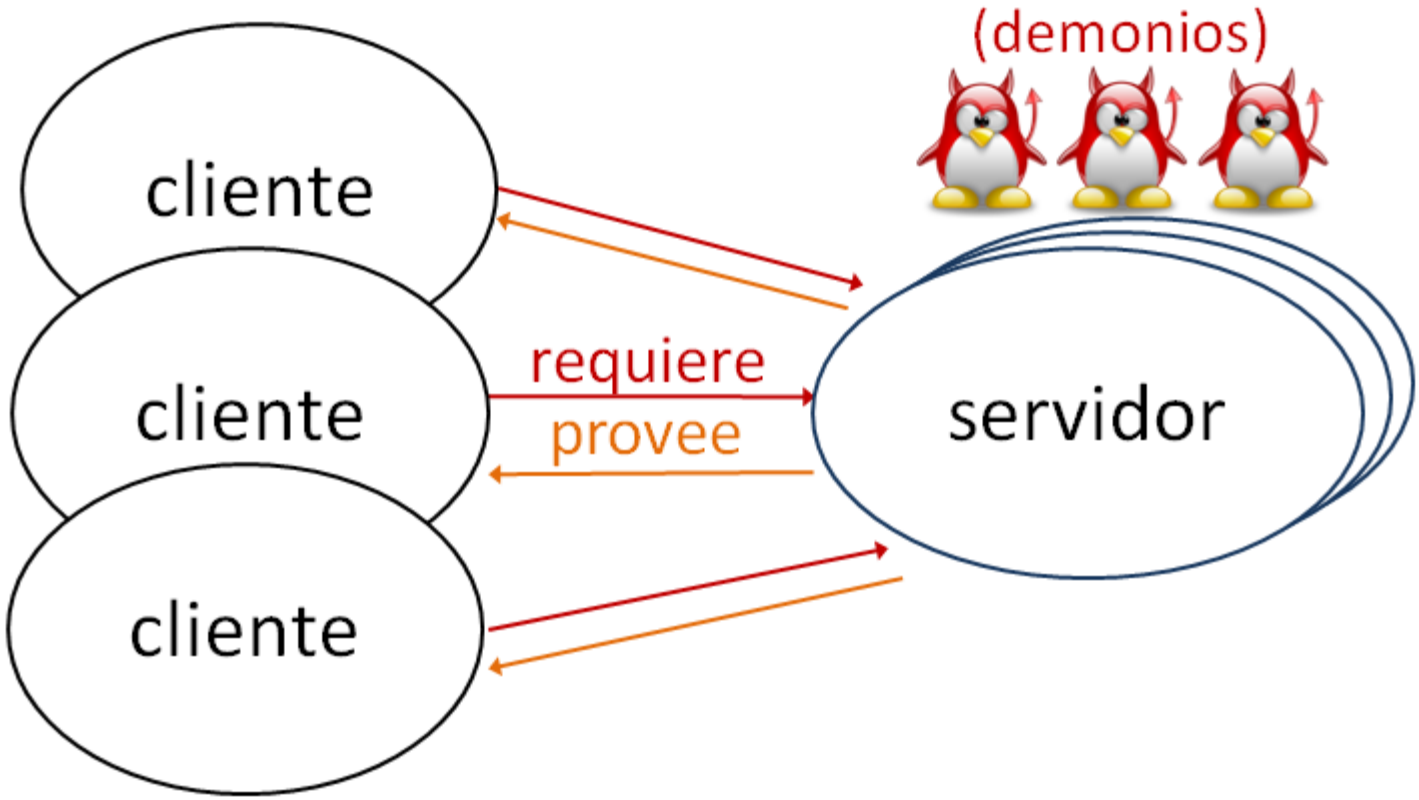
- Provee un mecanismo eficiente para comunicar aplicaciones distribuidas, que pueden funcionar simultáneamente como clientes y servidores para diferentes servicios.
- Ofrece ventajas organizativas por la gestión centralizada de la información y la separación de roles y responsabilidades, facilitando y clarificando el funcionamiento del sistema y el diseño y desarrollo de aplicaciones.
- La separación entre cliente y servidor es una separación lógica. El servidor no se ejecuta sobre una máquina ni es necesariamente un sólo programa.
- El modelo cliente-servidor se opuso durante años al modelo centralizado de los mainframes, hasta que éste cambió e incorporó características de cliente-servidor para tomar ventaja de la computación distribuida.
- El modelo cliente-servidor se ha convertido en una de las tecnologías dominantes para aplicaciones distribuidas y es muy común en la implementación de procesos que siguen modelos transaccionales realizando consultas a bases de datos.

MODELO CLIENTE-SERVIDOR

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En la arquitectura cliente-servidor, cada proceso servidor (usualmente implementado como un "*proceso demonio*") está permanentemente activo aguardando solicitudes de parte del cliente. Típicamente, muchos procesos cliente comparten los servicios de un único proceso servidor. Cuando se recibe una petición, el servidor crea una copia de si mismo para atender el pedido del cliente, a la vez que queda esperando recibir nuevas solicitudes.
- Ejemplos típicos de aplicaciones que funcionan como servidor son los habituales procesos de servidores web, los servidores de archivo, los servidores de correo, etc.
  - Los propósitos específicos varían al proveer uno u otro tipo de servicio, pero la arquitectura básica del modelo sigue siendo la misma.
- Otros ejemplos de este modelo son los navegadores web, los servicios web, el protocolo FTP, aplicaciones comerciales sobre Internet, el propio protocolo TCP/IP, aplicaciones basadas en sockets, juegos online, etc.

MODELO CLIENTE-SERVIDOR



MODELO CLIENTE-SERVIDOR

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Principales ventajas:
  - *la centralización del control*: los accesos, recursos y la integridad de los datos son controlados por el servidor, de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema. Esta centralización también facilita la tarea de actualizar datos o recursos.
  - *la escalabilidad*: es posible modificar la capacidad de clientes y servidores, que pueden ser aumentados/mejorados en cualquier momento y por separado. Además, se pueden incorporar recursos adicionales a la red, en el rol de cliente o de servidor, de manera transparente para el sistema.
  - *la facilidad de mantenimiento*: contar con las funciones y responsabilidades distribuidas entre varios recursos de procesamiento independientes permite reemplazar, reparar, actualizar, o incluso trasladar físicamente un servidor, sin gran impacto en los clientes.
- Actualmente existen tecnologías consolidadas diseñadas para el modelo cliente-servidor que aseguran seguridad/integridad en las transacciones, amigabilidad de las interfaces, y facilidad de uso del modelo.

MODELO CLIENTE-SERVIDOR

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

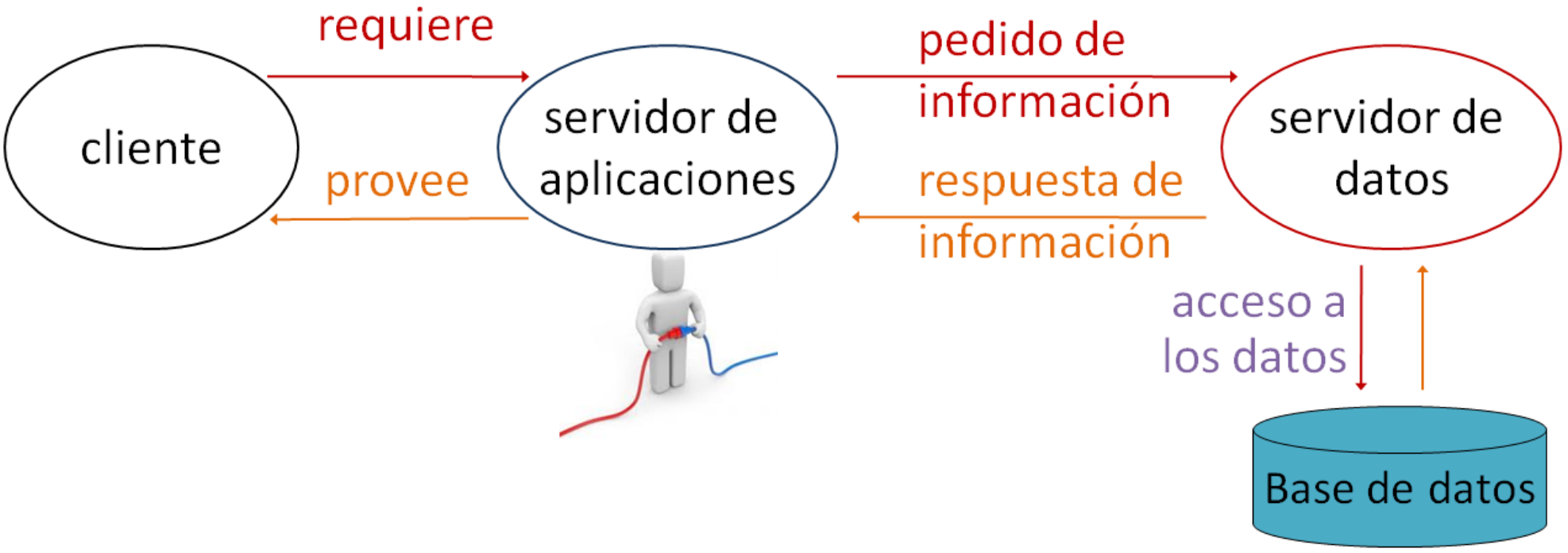
- Principales desventajas:
  - La congestión de tráfico es un inconveniente típico en escenarios donde muchos clientes envían peticiones simultáneas al mismo servidor. Este problema no es crítico en arquitecturas P2P, donde cada nodo opera como servidor y un mayor número de nodos proporciona mejor ancho de banda.
  - No tiene la robustez de las redes formadas por recursos con mayor autonomía de trabajo. En caso de caída de un servidor, las peticiones realizadas no pueden ser atendidas o derivadas automáticamente a otros.
  - Software y hardware de servidores condicionan la aplicabilidad del modelo. El diseño de la plataforma es muy importante. Una infraestructura pobremente dimensionada puede no dar servicio a un número elevado de clientes; una infraestructura sobredimensionada tendrá costos innecesarios de instalación, administración y mantenimiento.
  - El cliente no suele disponer de los recursos en el servidor, solo puede utilizarlos a través de los servicios expuestos. Por ejemplo, en una aplicación Web, no es posible escribir en el disco del cliente o imprimir directamente sin utilizar una interfaz de servicio (e.g., escritura, impresión) del navegador.

MODELO CLIENTE-SERVIDOR de 3 NIVELES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Propone una arquitectura multicapa para el sistema distribuido.
- El sistema se descompone en diferentes programas y/o servicios que pueden ser ejecutados en diferentes equipos, aumentando así el grado de distribución del sistema.
- La arquitectura típicamente utilizada en aplicaciones distribuidas proporciona tres niveles pero el modelo puede extenderse para contemplar más niveles en la jerarquía:
  - el cliente, que requiere servicios remotos
  - el servidor de aplicaciones, que provee la lógica transaccional
  - el servidor de datos, que se encarga de manejar y administrar el acceso a la información, instrumentado mediante enlaces con bases de datos

MODELO CLIENTE-SERVIDOR de 3 NIVELES



MODELO CLIENTE-SERVIDOR de 3 NIVELES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

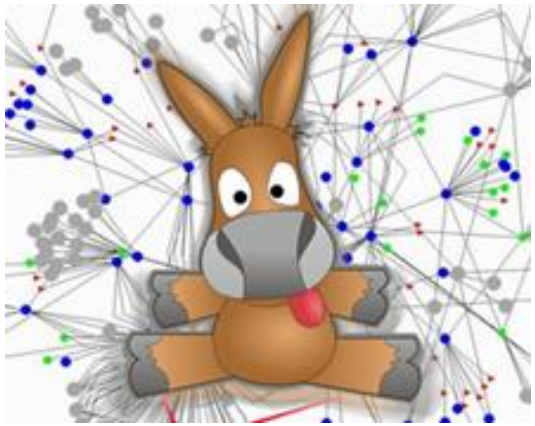
- Los clientes pueden ser procesos paralelos/distribuidos, navegadores universales usando páginas HTML estáticas o dinámicas, objetos Java (JVM), interfaces CGI, protocolos de comunicación, componentes ActiveX (Microsoft), objetos distribuidos mediante .NET, etc.
- El servidor de aplicaciones es el componente del sistema que soporta la lógica de transacciones necesaria para satisfacer los requerimientos de las aplicaciones. Garantiza que los sistemas tengan las características deseables de los sistemas distribuidos: que sean transaccionales, seguros, escalables, con la máxima disponibilidad, etc.
- El servidor de datos provee el acceso a los datos, mediante el uso de formatos estándar de datos (e.g., XML), y permite el enlace con bases de datos, sistemas ERP, monitores transaccionales y otros sistemas propios de la organización. En un sistema completamente distribuido, debe tenerse en cuenta que pueden existir múltiples repositorios de datos distribuidos e interfaces específicas para su acceso.

REDES de PARES (PEER-TO-PEER, P2P)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Implementa una organización y un modelo de comunicación en el cual cada recurso tiene las mismas capacidades para establecer una comunicación e iniciar la operativa del sistema.
- El sistema es completamente descentralizado. No existen clientes ni servidores fijos, todos los nodos de la red P2P actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red.

- Este modelo habitualmente se implementa asignando a los dispositivos las capacidades de cliente y servidor conjuntamente y la red se implementa mediante redes superpuestas construidas en la capa de aplicación de redes públicas tomadas como base (e.g., Internet).



REDES de PARES (PEER-TO-PEER, P2P)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Principales ventajas:
  - El *aprovechamiento*, la correcta *administración* y la *optimización* de recursos, en especial del uso del ancho de banda por medio de la conectividad entre usuarios, obteniendo más rendimiento en las conexiones y transferencias de datos que los métodos centralizados convencionales.
  - La *escalabilidad*, que permite obtener un alcance global, con cientos de millones de usuarios potenciales. Combinada con la característica de optimización de uso de recursos, la escalabilidad asegura que cuantos más nodos se conecten a una red P2P y compartan sus propios recursos, mejor será el funcionamiento del sistema distribuido en su conjunto. Esta característica constituye una gran ventaja sobre una arquitectura cliente-servidor, en los cuales la adición de clientes puede significar una degradación en los niveles globales de calidad de servicio.
  - La *descentralización*, que permite el manejo autónomo de los componentes, considerados todos iguales entre sí. Al no existir nodos con funciones especiales, ningún nodo es imprescindible para que funcione el sistema.

REDES de PARES (PEER-TO-PEER, P2P)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Principales ventajas (continuación):
  - La *robustez*; la distribución incrementa la tolerancia a fallos físicos (caída de pares o enlaces de comunicación), situaciones dinámicas imprevistas (salida de pares de la red) o fallos lógicos (transmisión de datos). Aunque algunos pares abandonen la red, otros pares pueden garantizar el servicio (por ejemplo, completar una descarga) utilizando la información y los servicios disponibles sin realizar peticiones a un servidor centralizado o activar mecanismos complejos de recuperación. En un sistema totalmente distribuido y autónomo, no existirá en la red ningún punto singular de falla.
  - La *distribución de costos* entre los usuarios, posibilitado por un mecanismo de computación colaborativa voluntaria, que permite compartir o donar recursos (archivos, ancho de banda, ciclos de CPU, espacio de disco) a cambio de servicios (almacenamiento, cómputo, etc.) o de otros recursos.
  - El *anonimato*, que permite ocultar los autores, editores y lectores de cada contenido, el servidor que lo aloja y la petición realizada para encontrar la información, siempre que así lo necesiten los usuarios.

REDES de PARES (PEER-TO-PEER, P2P)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Principales desventajas: *seguridad* y *direccionamiento*
  - La seguridad es una de las características deseables de los sistemas distribuidos que está menos implementada en redes P2P.
  - Los objetivos ideales de una red P2P segura incluirían identificar y evitar los nodos maliciosos, evitar contenidos infectados, evitar el espionaje de las comunicaciones entre nodos, y permitir la creación de grupos seguros de nodos y la protección de los recursos de la red.
  - Las características deseables de seguridad en sistemas P2P aún están bajo investigación, existiendo propuestas para avances en cifrado multiclave, sistemas de aislamiento de procesos (cajas de arena, *sandboxes*), la utilización de máquinas virtuales para la ejecución encapsulada de procesos, los mecanismos de reputación, las comunicaciones seguras, etc.

REDES de PARES (PEER-TO-PEER, P2P)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Principales desventajas: *seguridad* y *direccionamiento*
  - Los problemas de direccionamiento están causados porque en general los nodos de una red P2P no disponen de una dirección IP fija o accesible para otros nodos de Internet [porque se conectan a través de redes LAN o inalámbricas, porque están detrás de un firewall y/o utilizan traducción de direcciones de red (Network Address Translation, NAT), o forman parte de la red de un proveedor de servicios de Internet (ISP) que utiliza direccionamiento dinámico a demanda].
  - Para resolver estos inconvenientes, se utilizan mecanismos de conexión a un servidor inicial de IP fija y conocida (que puede ser distribuido o replicado en varios servidores físicos) para integrarse a la red, y la utilización de proxys de conexión para la comunicación entre nodos que no cuentan con dirección IP pública entre ellos.

SISTEMAS P2P

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Descentralizados, centralizados e híbridos
  - Los sistemas *totalmente descentralizados* son los más comunes y los más versátiles, ya que no requieren de un mecanismo de gestión central ni servidores centralizados para manejar las conexiones, enrutar o administrar el direccionamiento. Los propios nodos de la red realizan procesamiento, almacenamiento y gestión de las comunicaciones, que se realizan directamente entre usuarios, utilizando nodos intermedios de enlace. Todos los nodos tienen las potestades para actuar como cliente y como servidor.
  - Los sistemas P2P *centralizados* permiten proporcionar servicios y compartir recursos (procesamiento, datos) que utilizan un servidor central para manejar las transacciones y comunicaciones. Implementaciones de estas redes fueron muy populares, incluyendo el primer sistema de distribución de archivos P2P de popularidad masiva (Napster, 1999–2001). En general poseen una administración muy dinámica y una disposición más permanente de contenido, pero tienen como inconvenientes la falta de escalabilidad, de tolerancia a fallos y una seguridad débil.

SISTEMAS P2P

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Descentralizados, centralizados e híbridos
  - Los sistemas *híbridos* combinan la utilización de un servidor central como enlace para las comunicaciones entre nodos y que administra los recursos de banda ancha, pero sin conocer la identidad de los nodos ni almacenar información sobre las comunicaciones.
  - Combinan ventajas de los modelos centralizados, como la administración dinámica de peticiones y la disposición de contenido, con características de las redes descentralizadas, como la tolerancia a fallos, ya que un grupo de nodos puede seguir en comunicación a pesar de que se caiga un servidor o un grupo de servidores. Los nodos son responsables de almacenar la información, mientras que el enrutamiento es resuelto por el servidor mediante índices administrados para obtener direcciones absolutas.

SISTEMAS P2P

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Las principales aplicaciones de los sistemas P2P incluyen las redes para compartir archivos (audio, video, software), la telefonía de voz sobre IP (VoIP) que permite obtener una mayor eficiencia en la transmisión de datos en tiempo real, los sistemas de moneda virtual, etc.
- Los sistemas grid utilizados para compartir recursos de cómputo (grids computacionales) y de datos (grids de datos) también siguen el paradigma de computación P2P mediante un modelo híbrido concebido para una alta integración de los peers para compartir eficientemente los recursos.