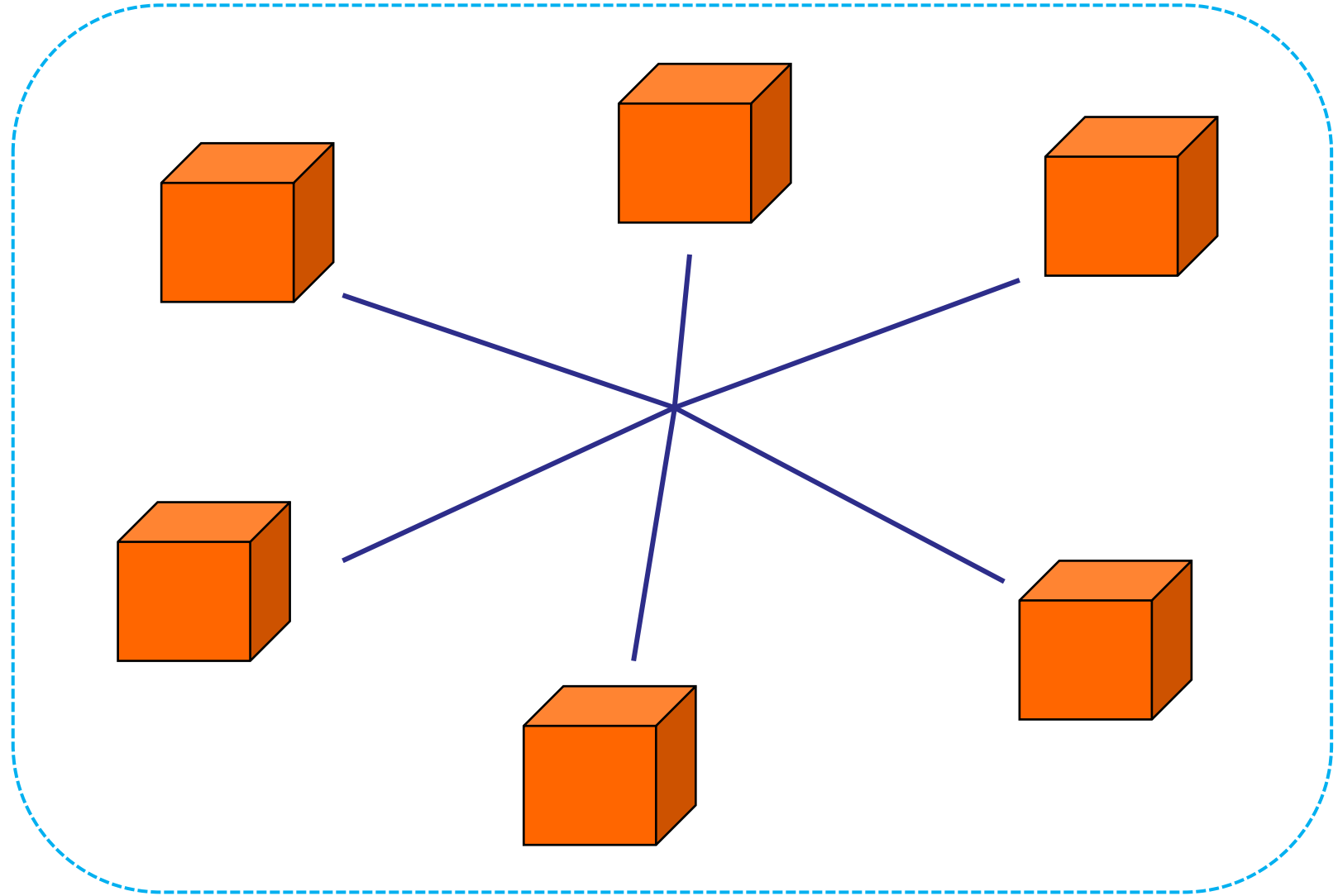


## Unidad N° 4:

# SINCRONIZACIÓN E INTERBLOQUEO EN SISTEMAS OPERATIVOS DISTRIBUIDOS



# SISTEMAS OPERATIVOS DISTRIBUIDOS

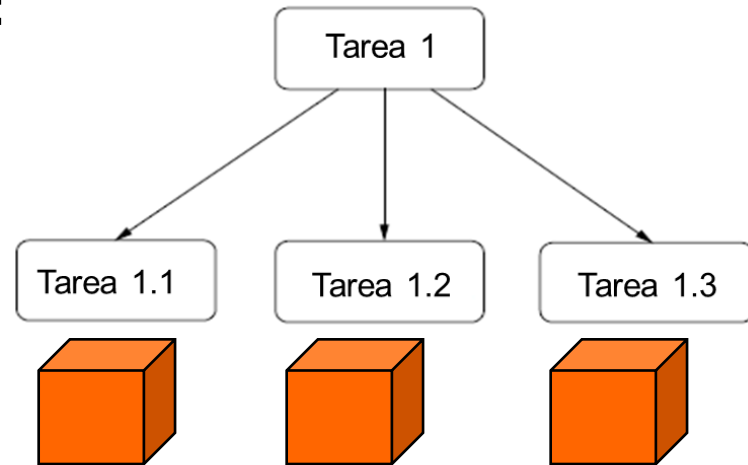


*Conjunto de computadoras que se integran para hacer desaparecer la dualidad local / remoto para ofrecer la visión de un «sistema único»*

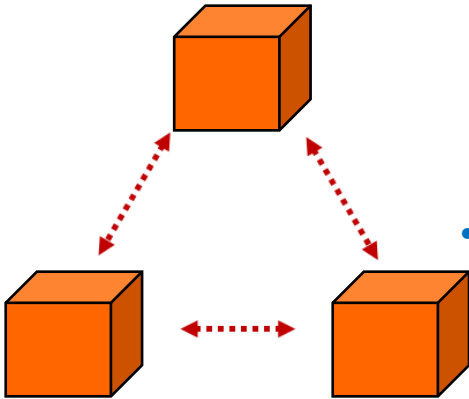
# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Objetivos de un Sistema Distribuido:

- Distribuir el Trabajo.



- Compartir Recursos.



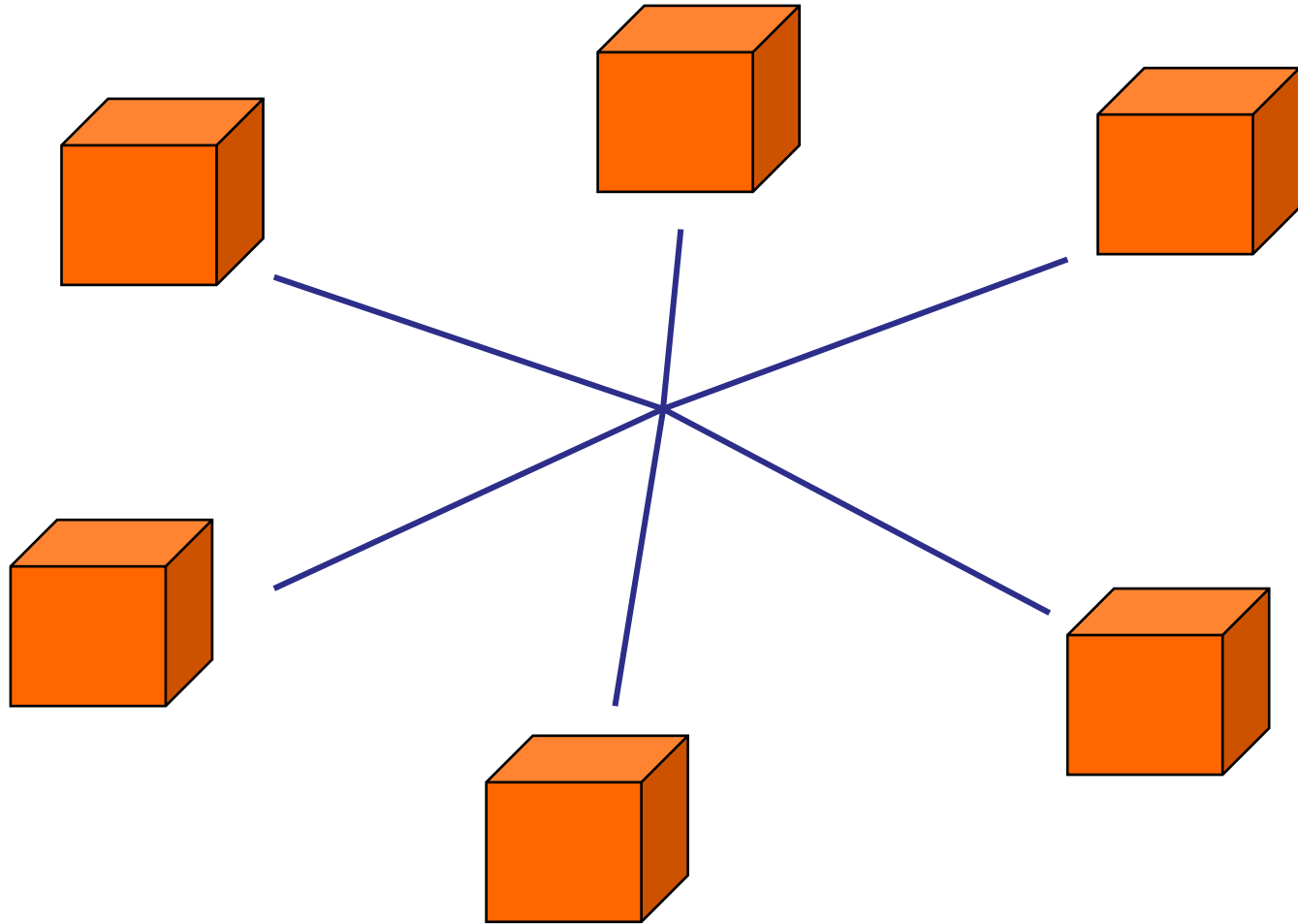
- Logrando:
  - ✓ Alto Rendimiento
  - ✓ Alta Escalabilidad
  - ✓ Alta Disponibilidad

# SISTEMAS OPERATIVOS DISTRIBUIDOS

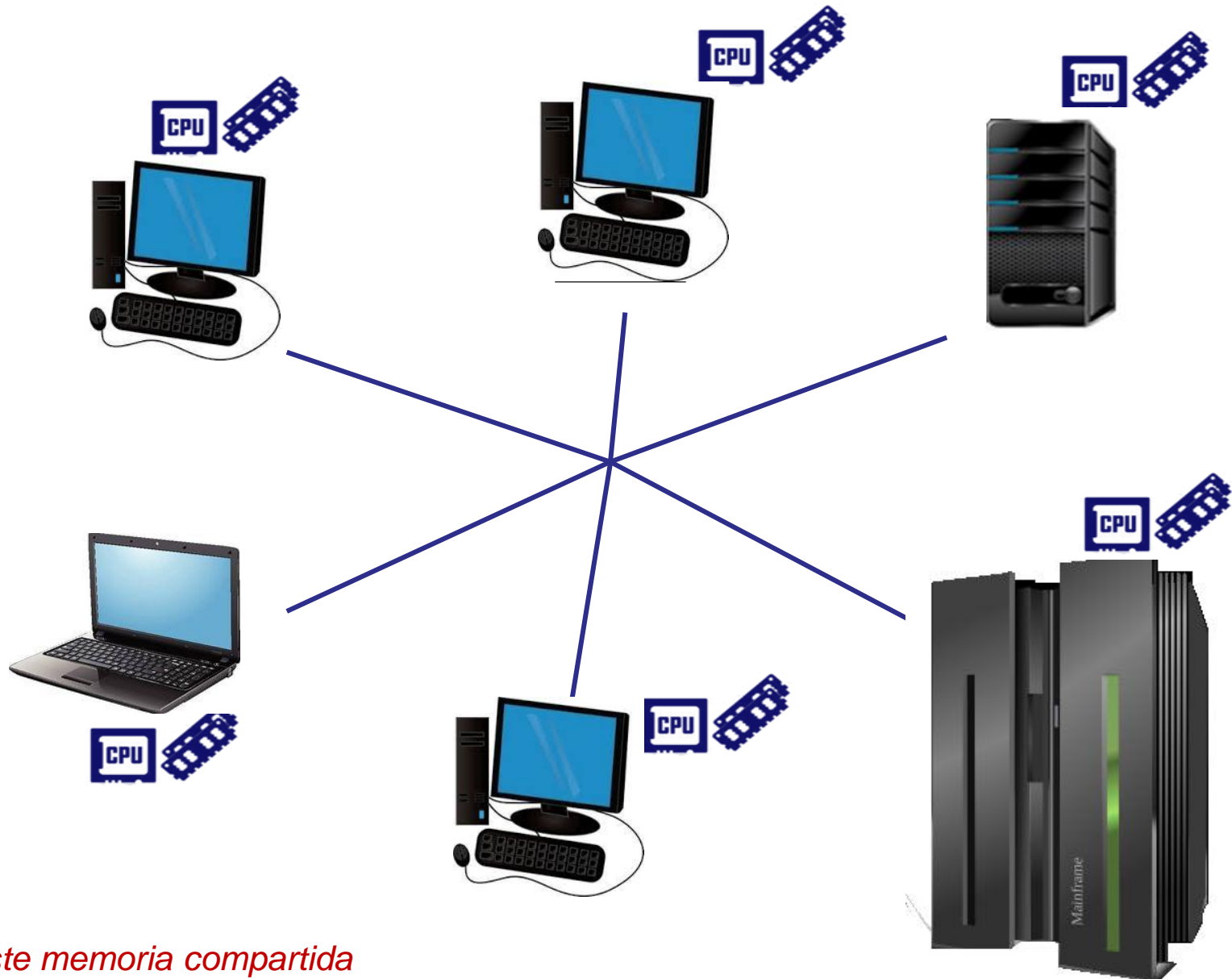
## ➤ Cuestiones para implementar un Sistema Distribuido:

- ❖ *¿Cómo distribuir la Carga de Trabajo?*
- ❖ *¿Cómo administrar los Recursos Compartidos?*
- ❖ *¿Cómo lograr la Sincronización de Procesos?*
- ❖ *¿Cómo manejar el Deadlock?*
- ❖ *¿Cómo lograr un 'Estado Consistente'?*
- ❖ *¿Cómo asegurar la Confiabilidad y Fiabilidad?*

# SISTEMAS OPERATIVOS DISTRIBUIDOS



# SISTEMAS OPERATIVOS DISTRIBUIDOS

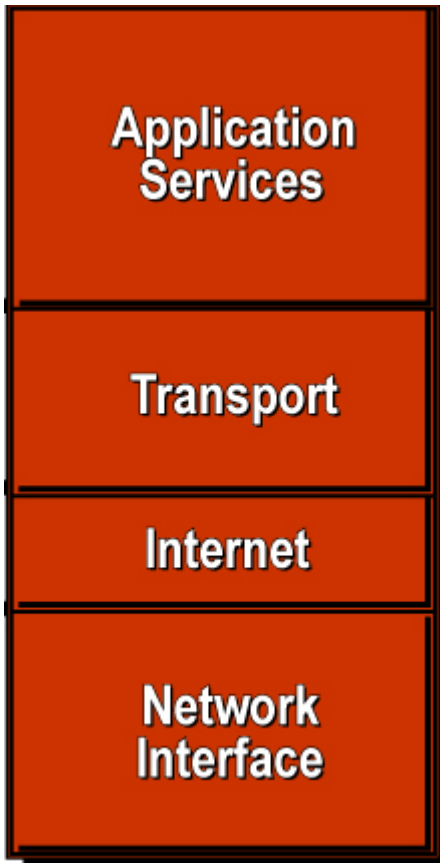


- *No existe memoria compartida*

# SISTEMAS OPERATIVOS DISTRIBUIDOS

- ¿Cómo compartir información entre procesos?

## TCP / IP



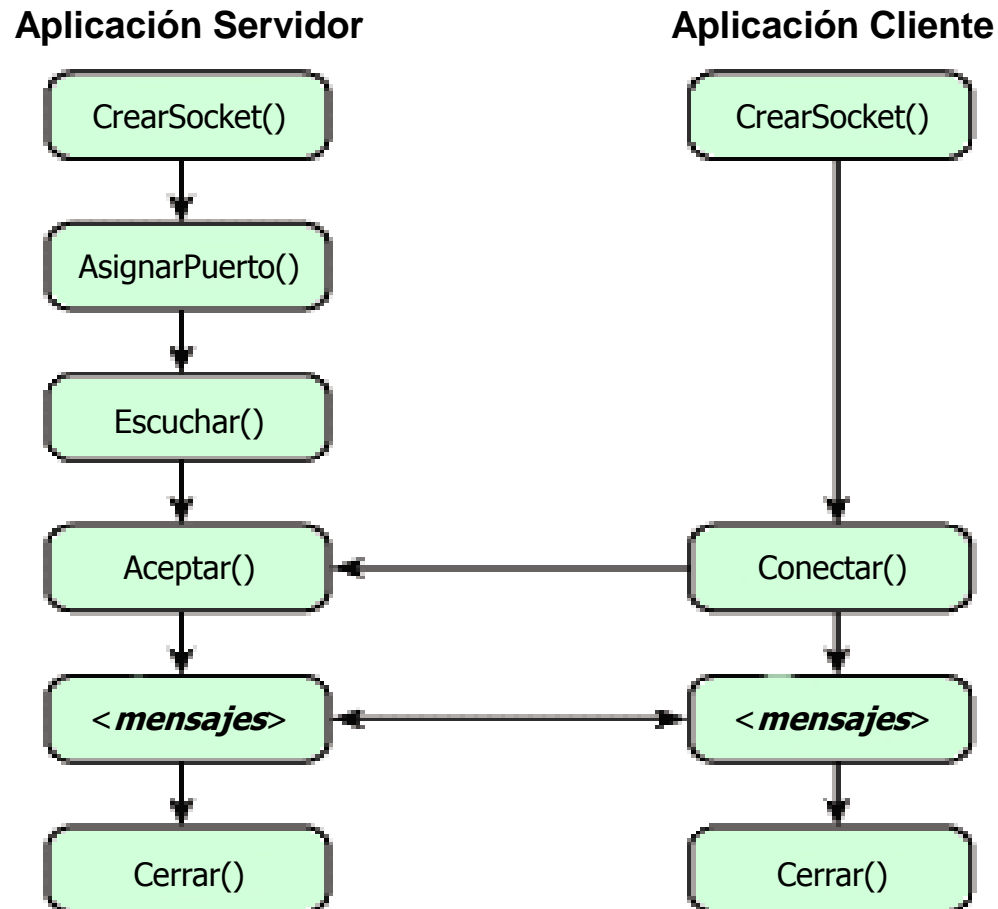
## □ Primitivas de Comunicación:

- Pipes
  - Sockets
  - Mensajes
- (The items Sockets and Mensajes are grouped within a light purple rounded rectangle in the original image.)
- Llamadas a Procedimientos Remotos ( RCP )

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo compartir información entre procesos?

- Sockets





# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo compartir información entre procesos?

- Mensajes

- función **Enviar**( socket, datos )

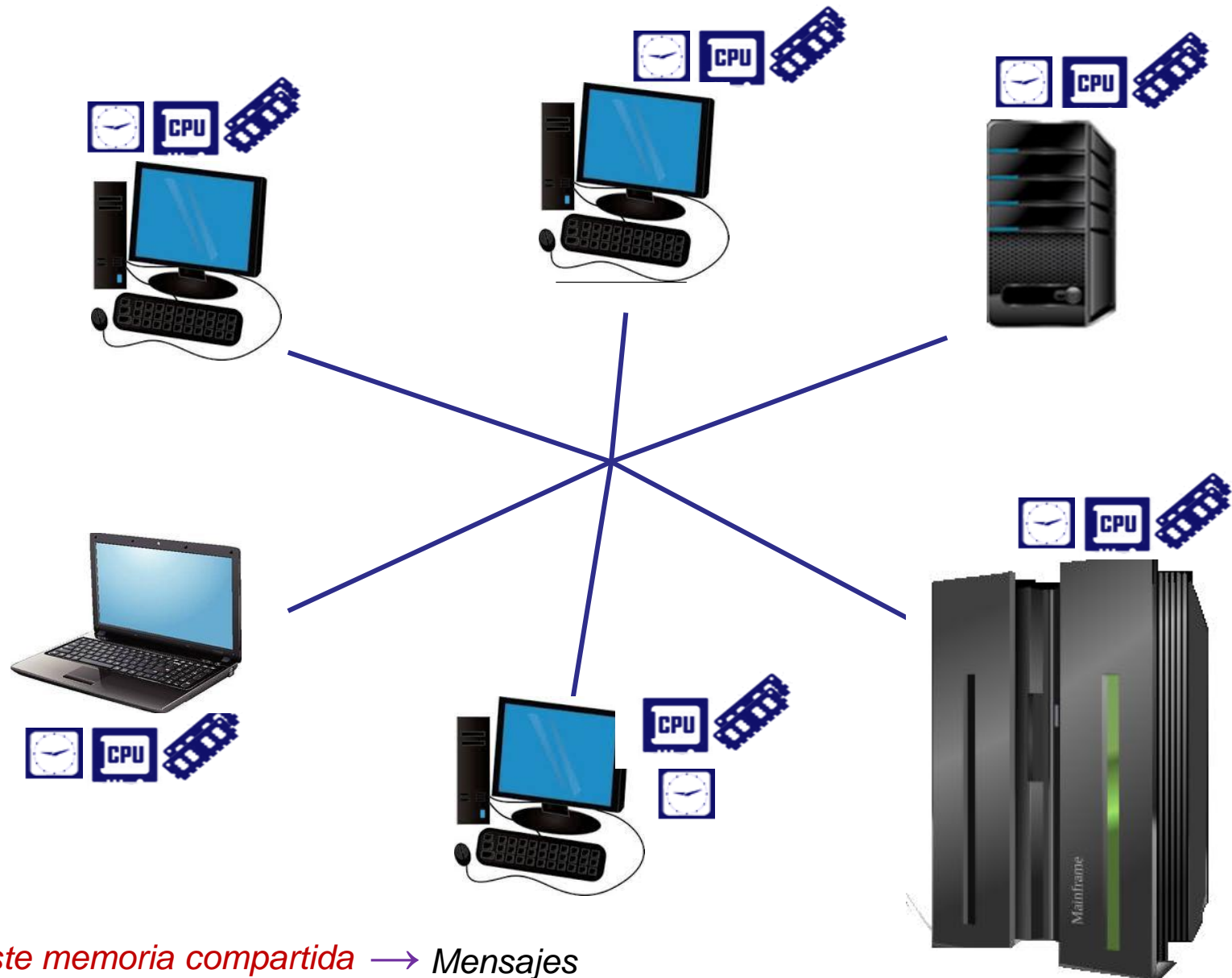
- función **Recibir**( socket ): datos

- Confiable vs No Confiable

- Sincrónico vs Asincrónico

- Bloqueante vs No Bloqueante

# SISTEMAS OPERATIVOS DISTRIBUIDOS



- *No existe memoria compartida* → Mensajes
- *No existe un 'reloj global'*

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo ordenar los eventos de los procesos?

### ❑ Sincronización de la Hora:

- Protocolo de Tiempo de Red ( NTP )
  - Permite sincronizar las computadoras con hora UTC de un Reloj Atómico.
  - Aplicable a redes locales o amplias por utilizar niveles jerárquicos.
  - Mecanismos:
    - a) Con cierta frecuencia, un servidor de tiempo manda la hora a todas las computadoras para que se sincronizan (redes LAN).
    - b) Cuando se quiere sincronizar, el servidor de tiempo le manda a otra computadora la hora a utilizar que considera el tiempo de latencia del canal de comunicación.

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo ordenar los eventos de los procesos?

### □ Relojes Virtuales:

- Permiten ordenar eventos en un sistema distribuido donde no hay una hora global confiable.
- Sólo permiten conocer que evento se efectuó antes o después de otro.

#### ▪ Mecanismo:

Registrar la cantidad de eventos que sucedieron e incluirlo como un *timestamp* en cada mensaje enviado.

- Relojes Virtuales de Lamport
- Relojes Vectoriales

# SISTEMAS OPERATIVOS DISTRIBUIDOS

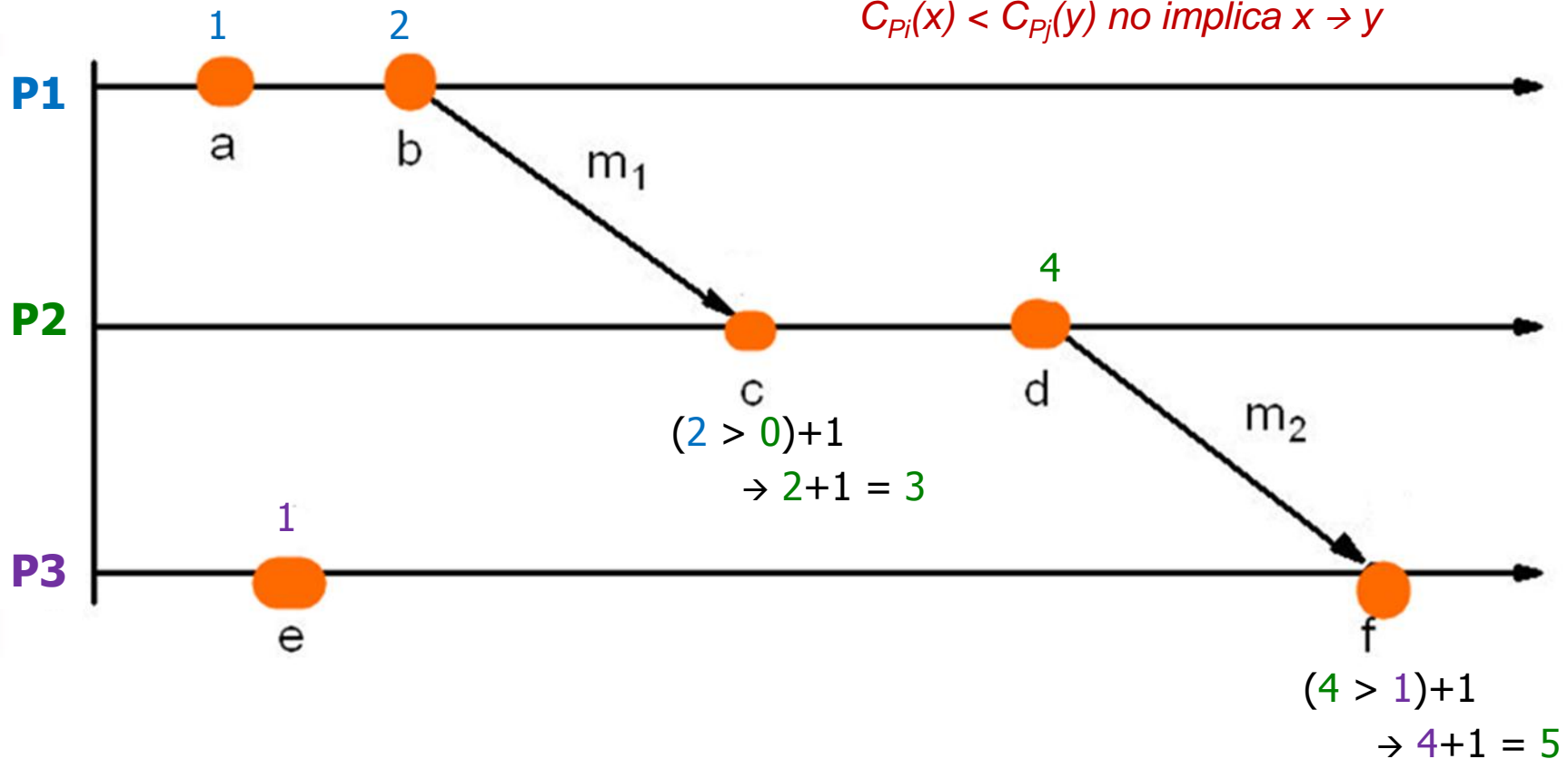
## ➤ ¿Cómo ordenar los eventos de los procesos?

### □ Relojes Virtuales:

- Relojes Virtuales de Lamport

Problema:

$C_{Pi}(x) < C_{Pj}(y)$  no implica  $x \rightarrow y$



# SISTEMAS OPERATIVOS DISTRIBUIDOS

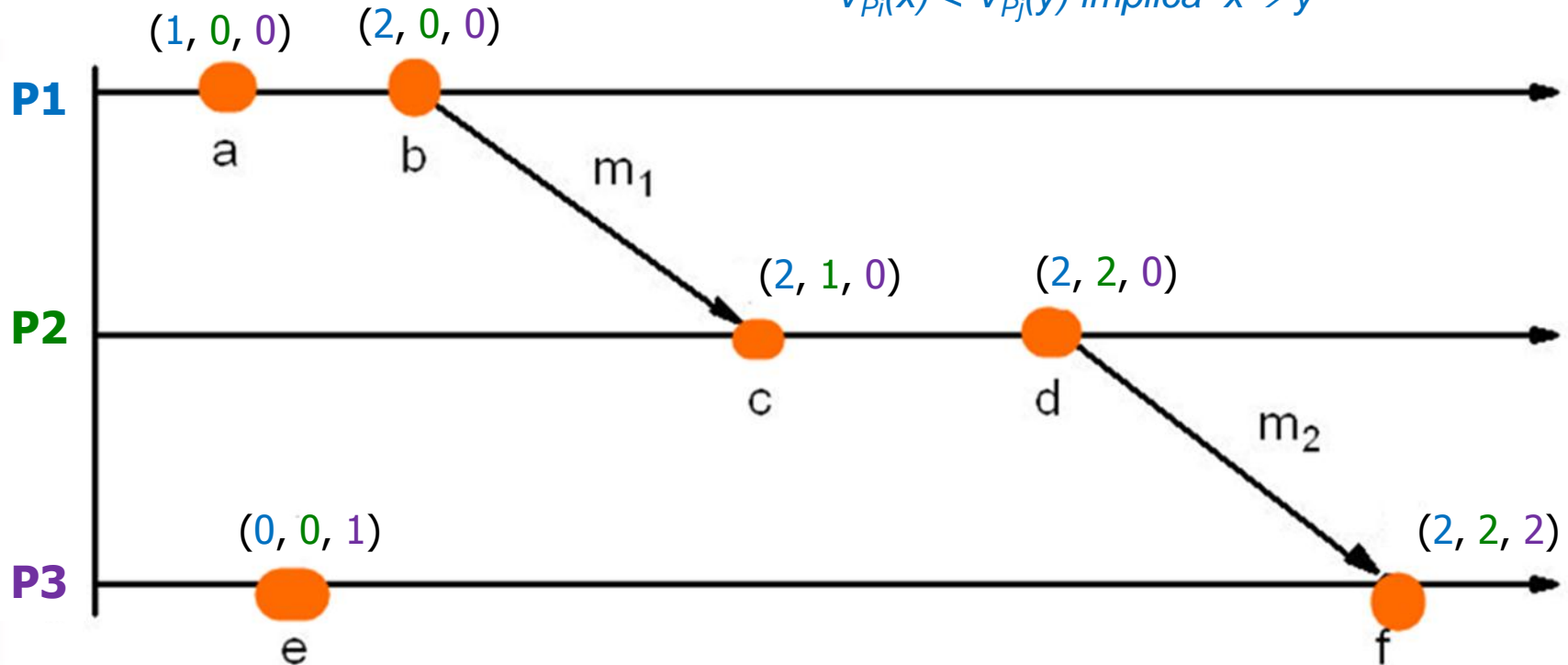
➤ ¿Cómo ordenar los eventos de los procesos?

□ Relojes Virtuales:

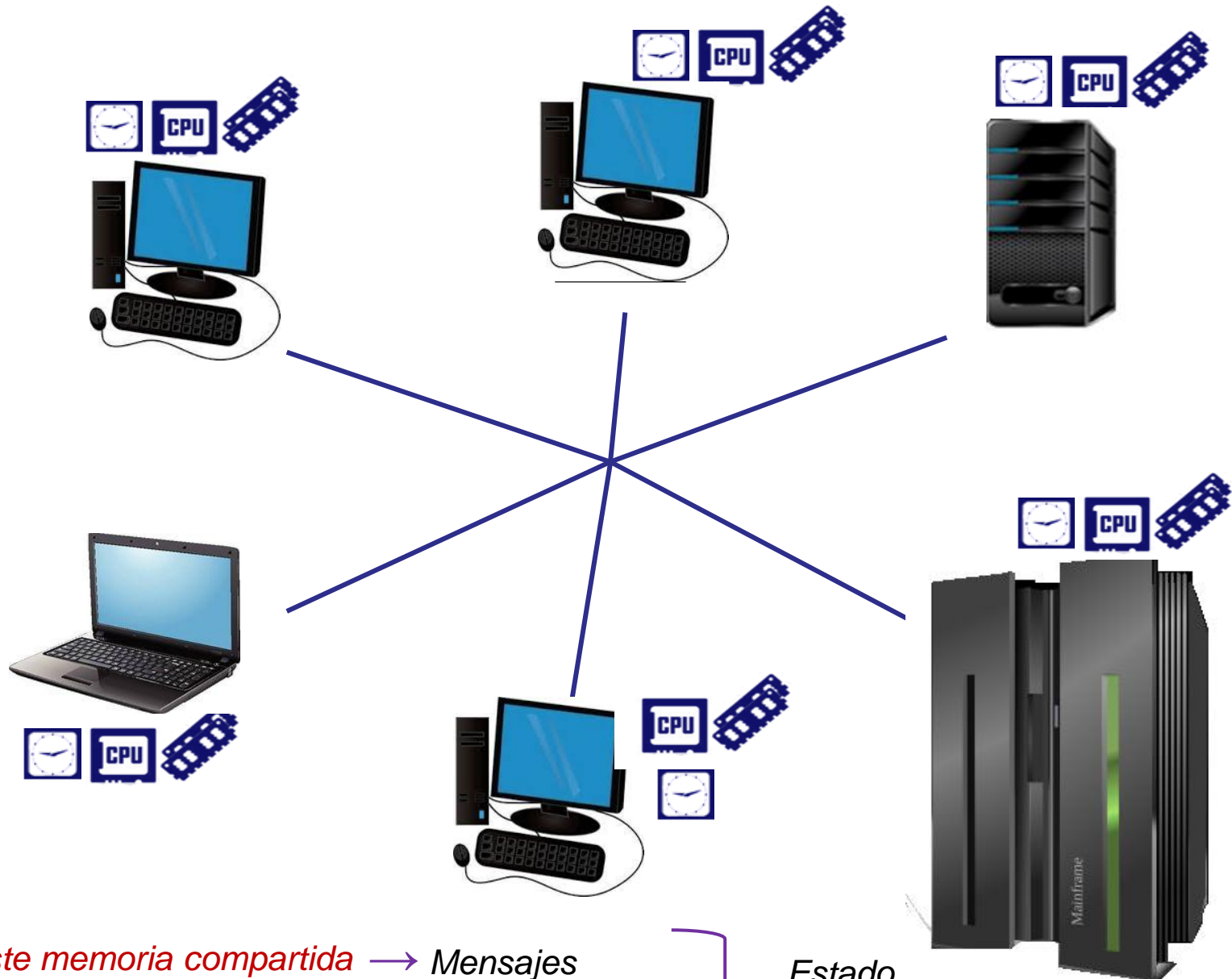
- Relojes Vectoriales

Garantiza:

$V_{P_i}(x) < V_{P_j}(y)$  implica  $x \rightarrow y$



# SISTEMAS OPERATIVOS DISTRIBUIDOS



- *No existe memoria compartida* → Mensajes
  - *No existe un 'reloj global'* → Relojes Virtuales
- Estado Global

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo obtener un Estado Global Consistente?

### □ Estado Global:

- Está definido en un Sistema Distribuido por:
  - el estado interno de cada computadora ( *memoria* )
  - el estado de los canales de comunicación ( *mensajes encolados* )
- Se utiliza para :
  - ✓ *detección del deadlock.*
  - ✓ *establecimiento de puntos de recuperación.*
  - ✓ *detección de objetos que no se encuentren referenciados o utilizados por los procesos.*
  - ✓ *detección de procesos finalizados ( correctamente o por error ).*

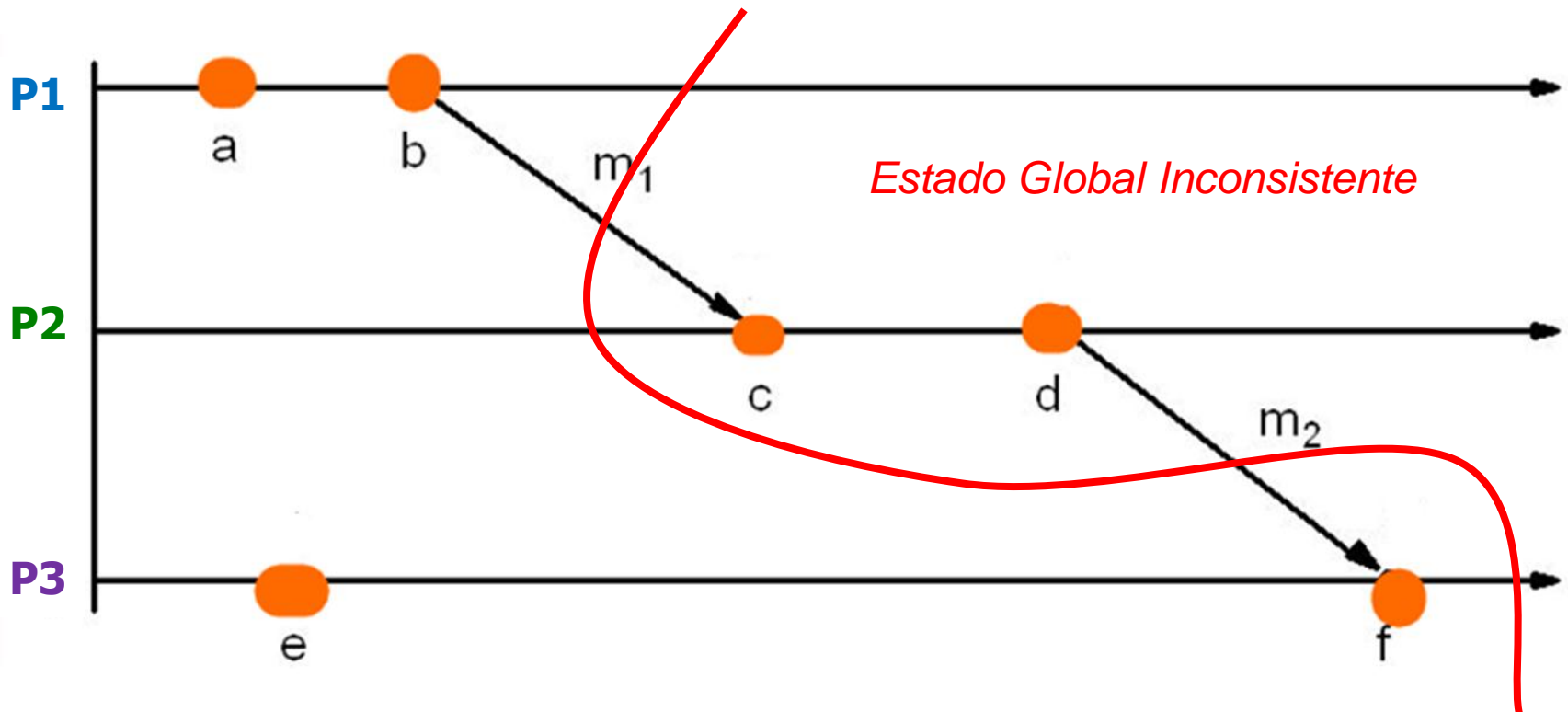


# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo obtener un Estado Global Consistente?

### □ Estado Global:

- Un Estado Global se considera *consistente* si por cada evento recolectado también se tienen recolectados todos que sucedieron anteriormente.

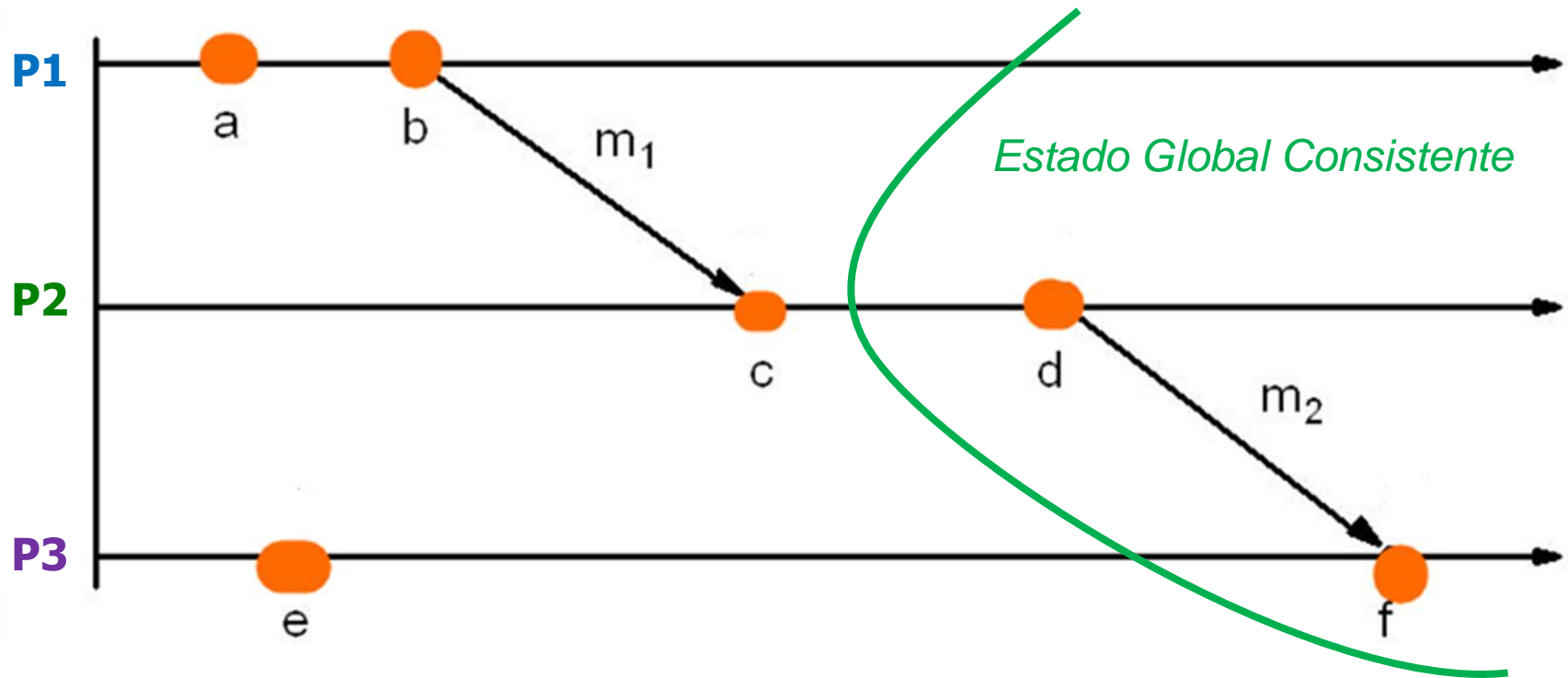


# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo obtener un Estado Global Consistente?

### □ Estado Global:

- Un Estado Global se considera *consistente* si por cada evento recolectado también se tienen recolectados todos que sucedieron anteriormente.

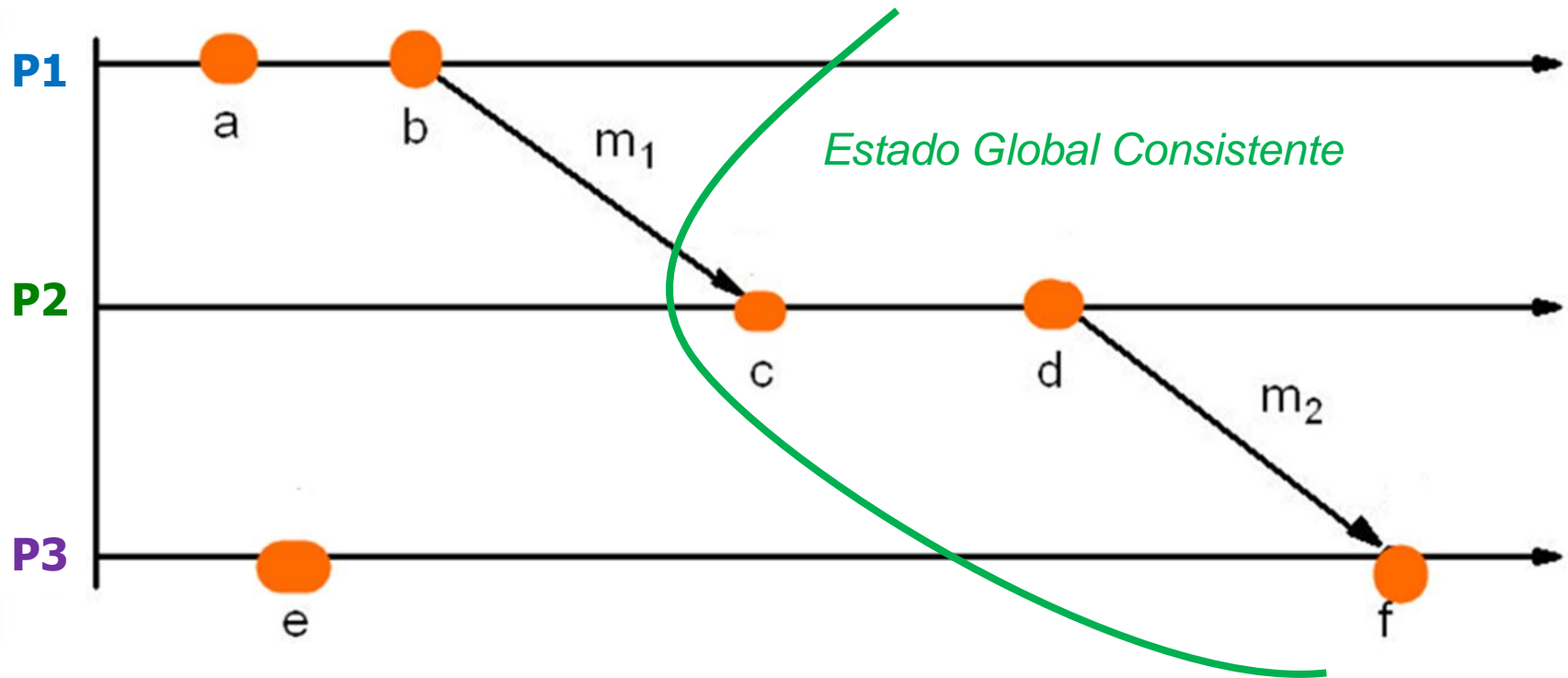


# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo obtener un Estado Global Consistente?

### □ Estado Global:

- Un Estado Global se considera *consistente* si por cada evento recolectado también se tienen recolectados todos que sucedieron anteriormente.

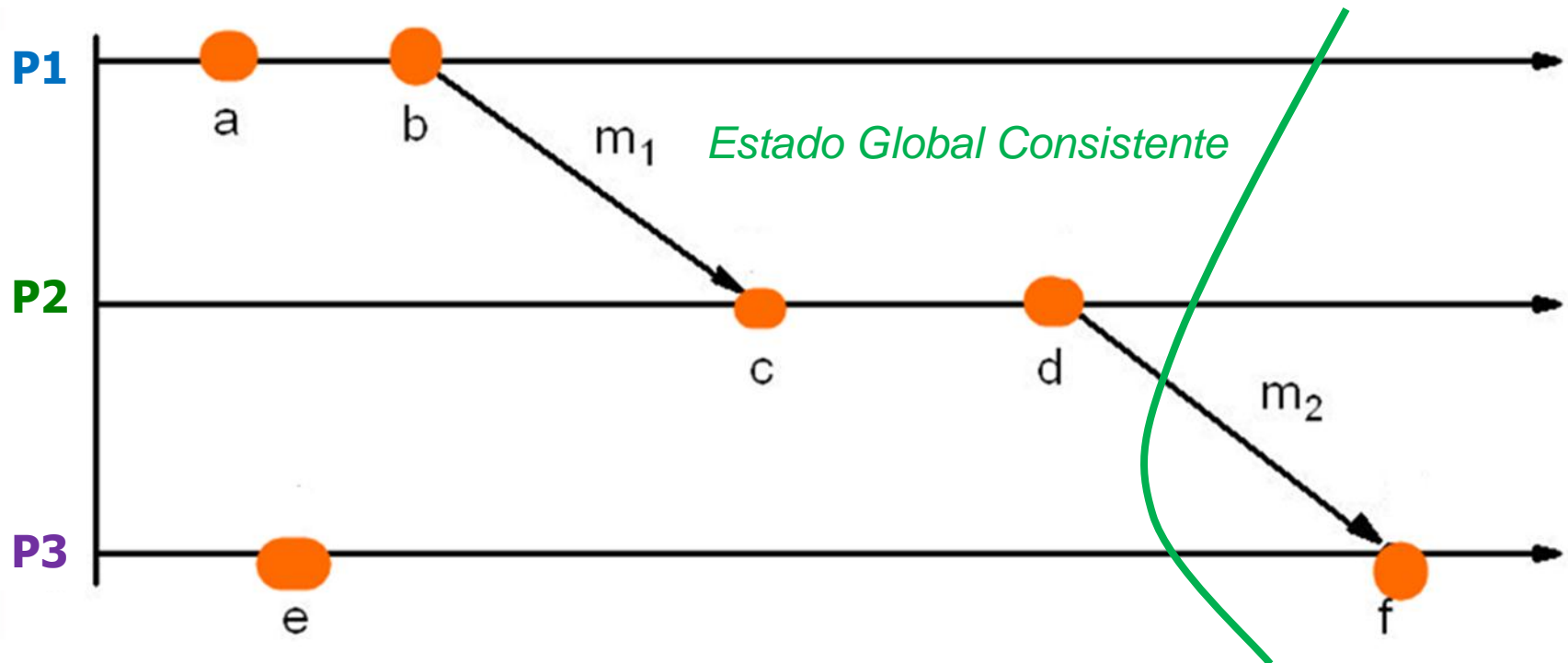


# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo obtener un Estado Global Consistente?

### □ Estado Global:

- Un Estado Global se considera *consistente* si por cada evento recolectado también se tienen recolectados todos que sucedieron anteriormente.



# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ ¿Cómo obtener un Estado Global Consistente?


### □ Algoritmo de la *Instantánea* o *Snapshot* [Chandy & Lamport]:

- 1) El proceso que inicializa el algoritmo guarda su estado local y envía el mensaje *snapshot* al resto de los procesos.
- 2) Cuando un proceso recibe el mensaje *snapshot* desde el proceso P:
  - Si es la primera vez que lo recibe, guarda su estado y reenvía el mensaje a todos los otros procesos. Además comienza a registrar todos los mensajes recibidos de otros procesos.
  - Si no es la primera vez que lo recibe, se almacenan los mensajes anteriores recibidos del proceso P como el estado de ese canal. Además deja de registrar los nuevos mensajes que lleguen de P.

Cuando el proceso haya recibido un mensaje *snapshot* de todos los otros procesos, el algoritmo finaliza enviándose el estado local y de los canales al proceso que lo solicitó.

# SISTEMAS OPERATIVOS DISTRIBUIDOS

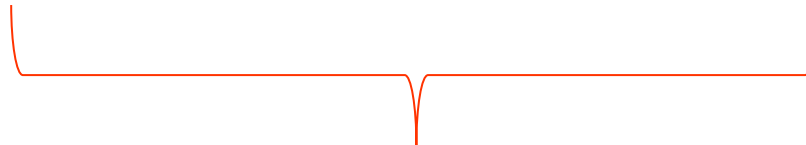
## ➤ Cuestiones para implementar un Sistema Distribuido:

- ❖ *¿Cómo distribuir la Carga de Trabajo?*
- ❖ *¿Cómo administrar los Recursos Compartidos?*
- ❖ *¿Cómo lograr la Sincronización de Procesos?*
- ❖ *¿Cómo manejar el Deadlock?*
- ❖ *¿Cómo lograr un 'Estado Consistente'?* 
- ❖ *¿Cómo asegurar la Confiabilidad y Fiabilidad?*

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Sincronización de Procesos:

- Busca que los Procesos no se interfieran entre sí al ejecutarse en forma concurrente.
- Controla *el acceso de los Recursos Compartidos* de los Procesos



***Región Crítica***



buscando garantizar la ***Exclusión Mutua***.

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Sincronización de Procesos:

### ❑ Implementación de Exclusión Mutua mediante **Semáforos**:

- Se basa en la utilización de un mecanismo especial provistos por el Sistema Operativo para controlar el acceso a la región crítica: los *Semáforos*

- Un *Semáforo* es una clase formada por:

- ✓ *Contador Entero*
- ✓ *Cola de Espera de Procesos (FIFO)*
- ✓ *Funciones Atómicas: Up & Down*

#### Requiere:

- *memoria compartida*
- *colas de espera globales*
- *Eventos globales*

función Down( semáforo S )

```
{  
    S.valor = S.valor - 1;  
    si ( S.valor < 0 ) {  
        Bloquear( procesoActual );  
        AgregarCola( S.colas, procesoActual );  
    }  
}
```

función Up( semáforo S )

```
{  
    S.valor = S.valor + 1;  
    si ( S.valor ≤ 0 ) {  
        p = SacarCola( S.colas );  
        Desbloquear( p );  
    }  
}
```



# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Sincronización de Procesos:

### ❑ Implementación de Exclusión Mutua mediante **Mensajes**:

- Se basa en la utilización de las primitivas de comunicación ( sockets + mensajes ) para controlar el acceso a la región crítica

- Funciones:

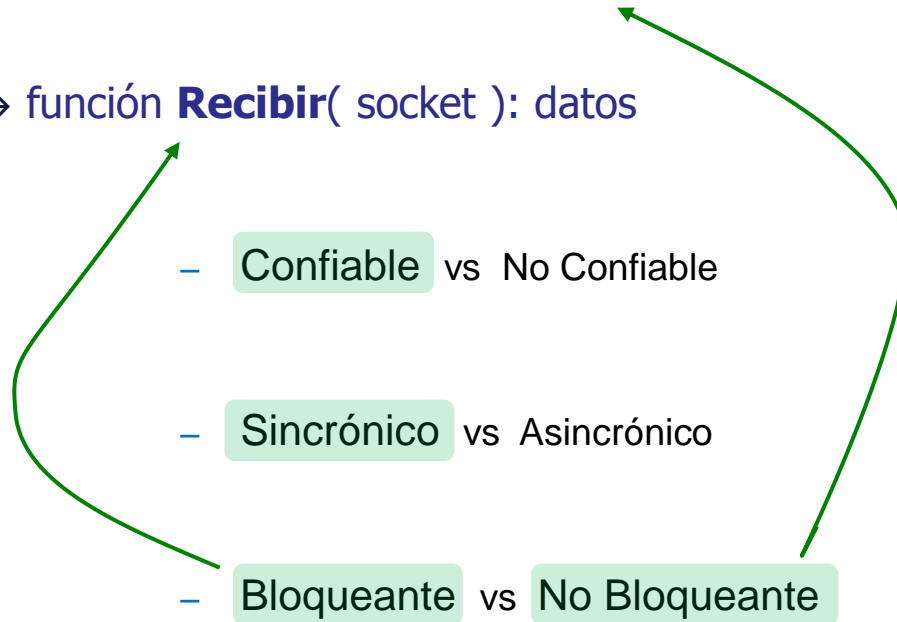
→ función **Enviar**( socket, datos )

→ función **Recibir**( socket ): datos

– **Confiable** vs No Confiable

– **Sincrónico** vs Asincrónico

– **Bloqueante** vs **No Bloqueante**



# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Sincronización de Procesos:

- Ejemplo: Productor / Consumidor con *semáforos*

```
Vector vec[];  
Int pos = 0;  
Semáforo S = 1;  
Semáforo N = 0;
```

función **Productor()**

```
{  
    mientras (verdadero) {  
        Item i = producir();  
  
        Down( S );  
        pos = pos + 1;  
        vec[pos] = i;  
        Up( S );  
        Up( N );  
    }  
}
```

función **Consumidor()**

```
{  
    mientras (verdadero) {  
        Down( N );  
        Down( S );  
        Item j = vec[pos];  
        pos = pos - 1;  
        Up( S );  
  
        consumir( j );  
    }  
}
```

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Sincronización de Procesos:

- Ejemplo: Productor / Consumidor con *mensajes*

función **Productor()**

{

<creación socket S>

mientras (verdadero) {

Item i = producir();

Enviar( S, i );

}

}

función **Consumidor()**

{

<creación socket S>

mientras (verdadero) {

Item j = Recibiri( S );

consumir( j );

}

}

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ **Sincronización de Procesos:**

- ¿Cómo manejar la sincronización con mensajes cuando existen múltiples nodos y procesos?

### □ Estrategias:

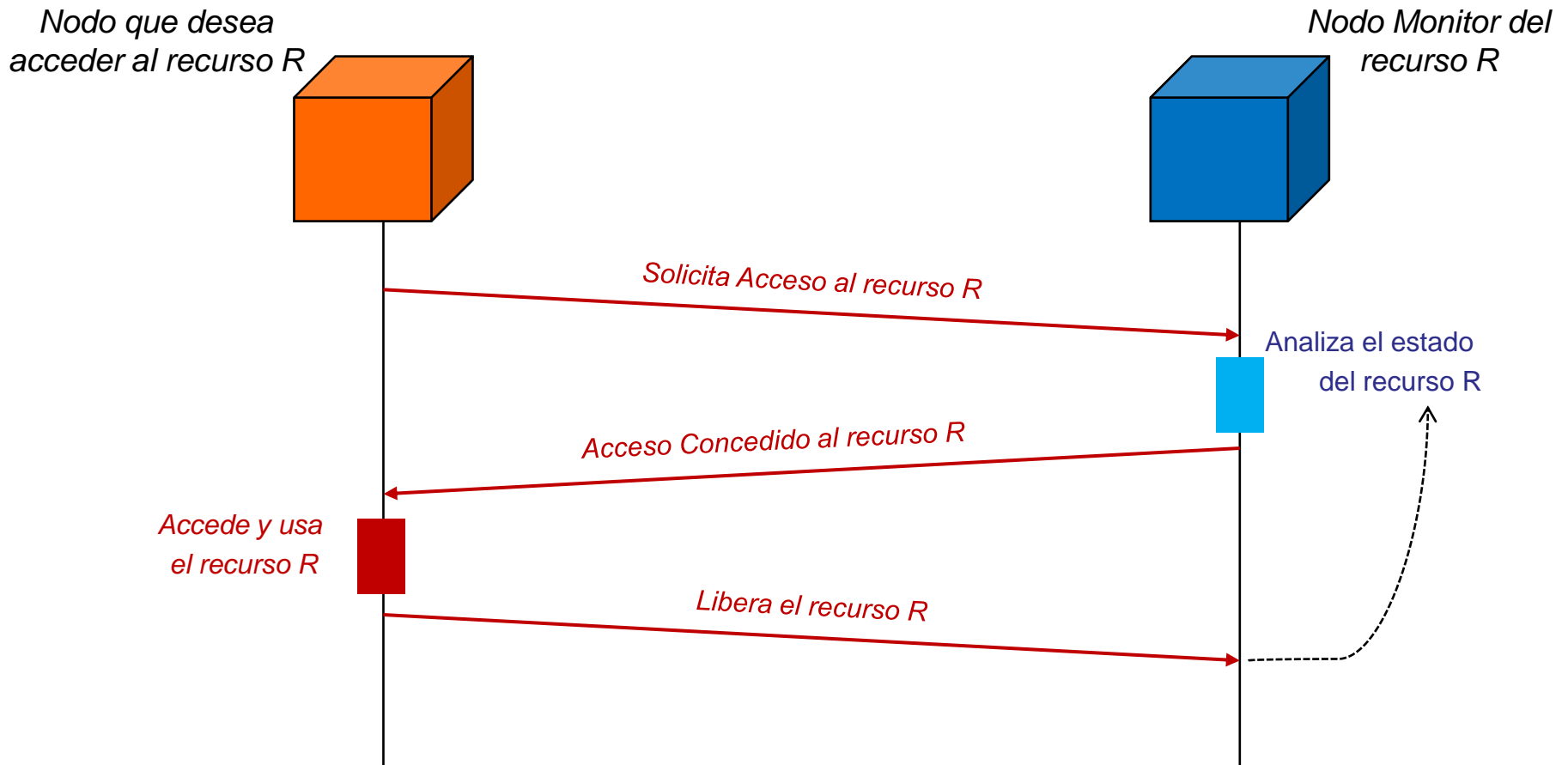
- Centralizada
- Descentralizada:
  - Con Token
  - Sin Token

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Sincronización de Procesos:

- Estrategia Centralizada:

- Existe un nodo de control ( 'Monitor' ) asociado a cada recurso que determina que proceso puede utilizarlo en cada momento.

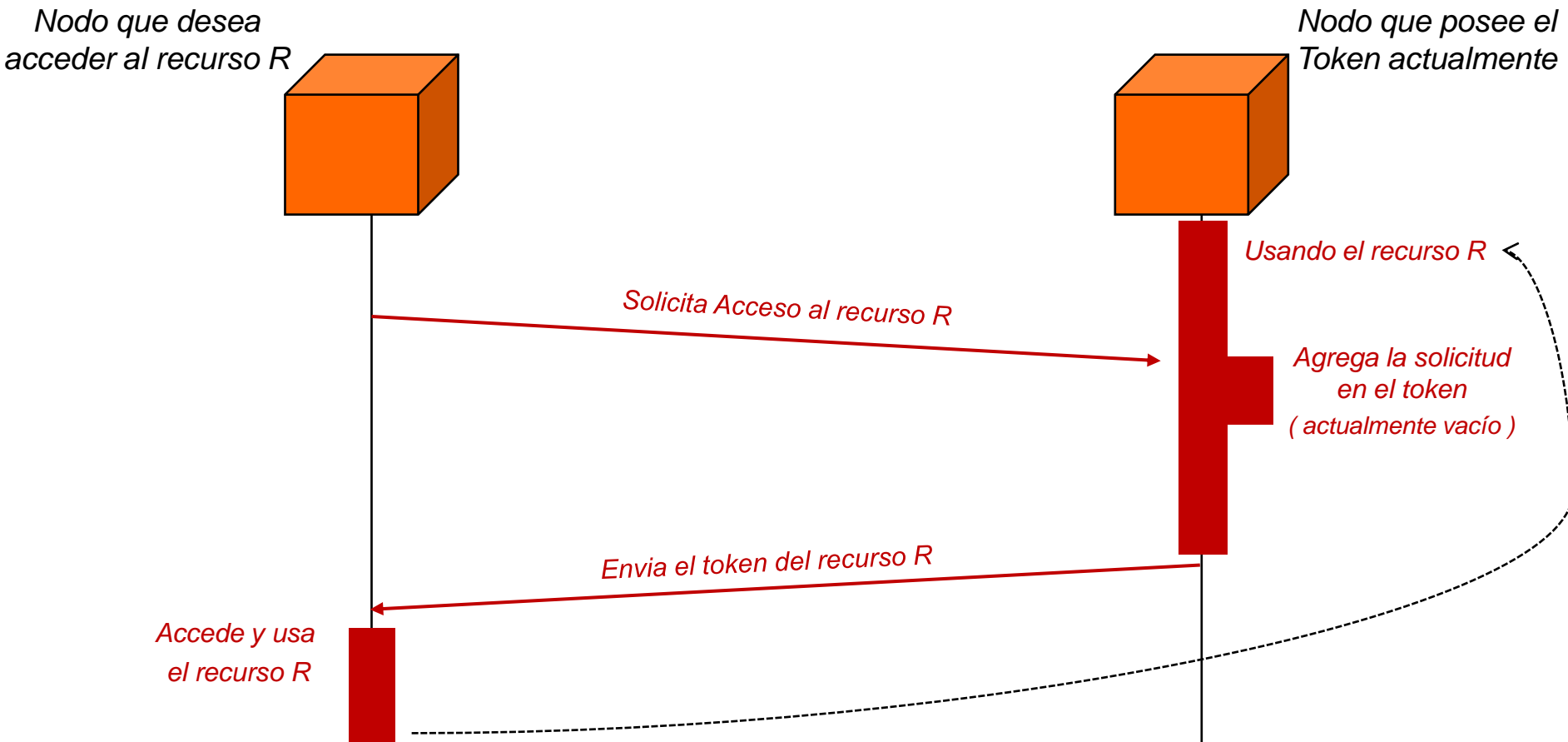


# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Sincronización de Procesos:

- Estrategia Descentralizada con Token:

- No existe un nodo de control pero sí un mensaje especial ( 'token' ) que le permite al que lo posee acceder al recurso.

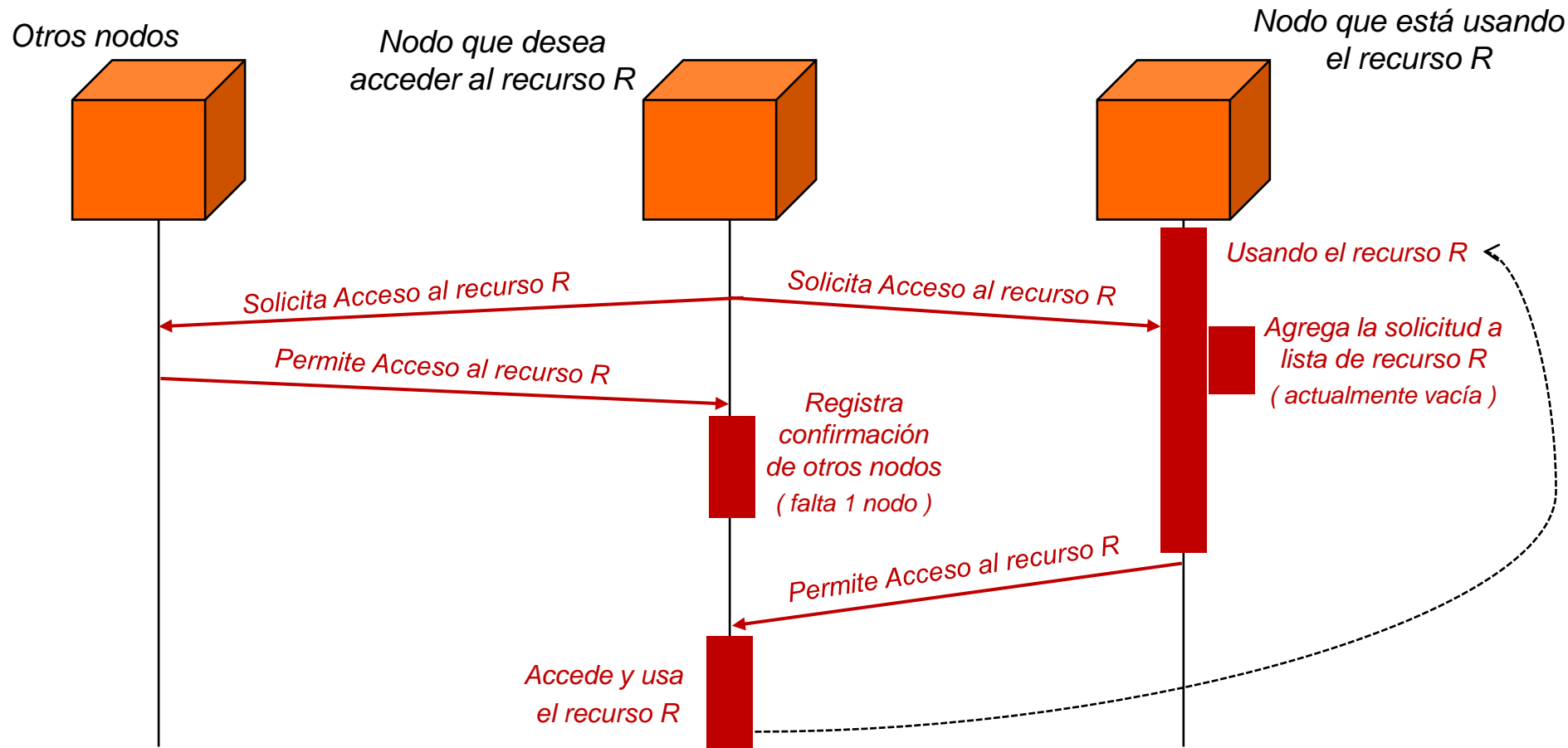


# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Sincronización de Procesos:

- Estrategia Descentralizada sin Token:

- No existe un nodo de control ni un mensaje especial. Se utiliza el orden de las solicitudes para permitir el acceso sincronizado al recurso.



# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Cuestiones para implementar un Sistema Distribuido:

❖ *¿Cómo distribuir la Carga de Trabajo?*

❖ *¿Cómo administrar los Recursos Compartidos?*

❖ *¿Cómo lograr la Sincronización de Procesos?* 

❖ *¿Cómo manejar el Deadlock?*

❖ *¿Cómo lograr un 'Estado Consistente'?* 

❖ *¿Cómo asegurar la Confiabilidad y Fiabilidad?*



# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ DeadLock:

- Condiciones:

1) *Mutua Exclusión*

2) *Tomar y Esperar*

3) *Recursos No Apropiativos*

4) *Espera Circular*

*Condiciones  
Necesarias*

*Condición  
Suficiente*

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ **DeadLock:**

- Mecanismos para resolver DeadLock:
  - ☐ ~~Estrategia del Avestruz~~
  - ☐ ~~Prevenir~~
  - ☐ ~~Evitar~~
  - ☐ Detectar & Eliminar

# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ DeadLock:

### ❑ Detectar y Eliminar el DeadLock:

En forma *recurrente* se analiza si hay *Espera/s Circular/es* entre los procesos  
( se simula la ejecución de los procesos para determinar si terminan o no )

→ si no hay, no hace nada

→ si hay, se elimina

- liberar recurso/s tomado/s
- rollback de proceso/s
- matar proceso/s

### Problema:

*los procesos de la espera circular pueden estar ejecutando en nodos diferentes*

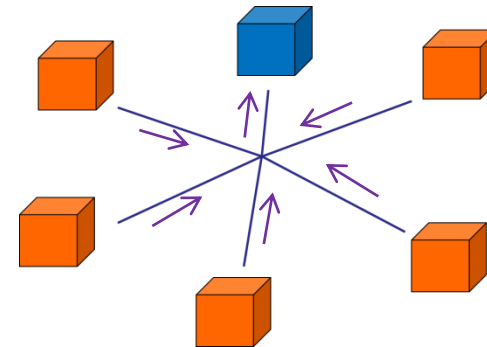
# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ DeadLock:

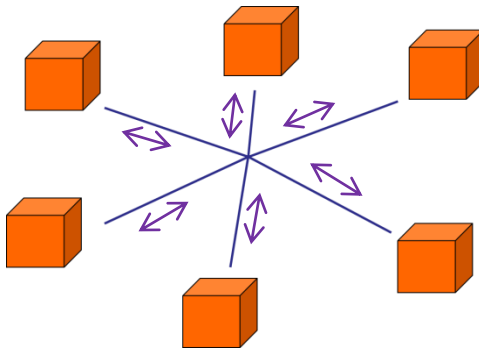
- ¿Cómo detectar el Deadlock entre múltiples procesos ejecutando en distintos nodos?

### □ Estrategias:

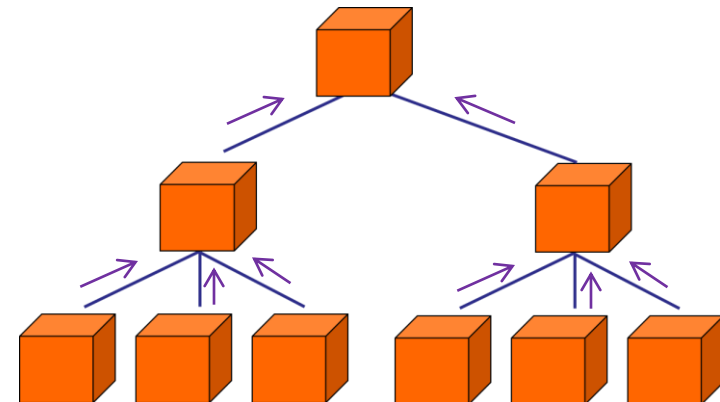
- Centralizada



- Descentralizada



- Jerárquica



# SISTEMAS OPERATIVOS DISTRIBUIDOS

## ➤ Cuestiones para implementar un Sistema Distribuido:

❖ *¿Cómo distribuir la Carga de Trabajo?*

❖ *¿Cómo administrar los Recursos Compartidos?*

❖ *¿Cómo lograr la Sincronización de Procesos?* 

❖ *¿Cómo manejar el Deadlock?* 

❖ *¿Cómo lograr un 'Estado Consistente'?* 

❖ *¿Cómo asegurar la Confiabilidad y Fiabilidad?*

- Guía de Estudio N° 4: *Sincronización en Sistemas Operativos Distribuidos* <http://sistemas.unla.edu.ar/sistemas/sls/lis-4-sistemas-operativos/pdf/SO-GE4-Sincronizacion-en-SODs.pdf>
- Singhal, M., & Shivaratri, N. G. (1994). Advanced concepts in Operating Systems. McGraw-Hill, Inc.. Capítulos 4 a 8.
- Stallings, W. (2005). Sistemas Operativos - Aspectos Internos y Principios de Diseño, 5<sup>ta</sup> Edición Prentice Hall. Capítulos 5 (sección 5.5) y 15 (secciones 15.2 a 15.4).
- Tanenbaum, A.S. (2009). Sistemas Operativos Modernos, 3<sup>ra</sup> Edición Prentice Hall. Capítulo 2 (sección 2.3.8).

# Preguntas



**¡¡GRACIAS!!**

