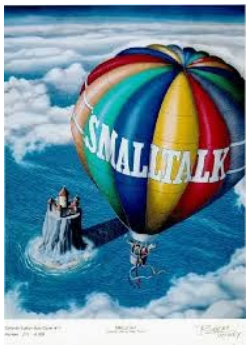
Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue

Programación Concurrente



Instrumentos de la concurrencia

Preguntas

- ¿Quién decide si un thread pasa de runnable a running?

El planificador de threads (scheduler)

- ¿Por qué se puede bloquear un thread?

Cuando intenta acceder a un objeto bloqueado, o cuando se queda durmiendo, o...

- ¿El programador puede influenciar en algo?

Si

- ¿Qué objetos tienen un lock?

Todos

- ¿Qué objetos se pueden sincronizar (syncronized) ?
- ¿Qué clase tiene los métodos wait(), notify() y notifyAll() ?
- ¿Qué objeto los puede invocar?

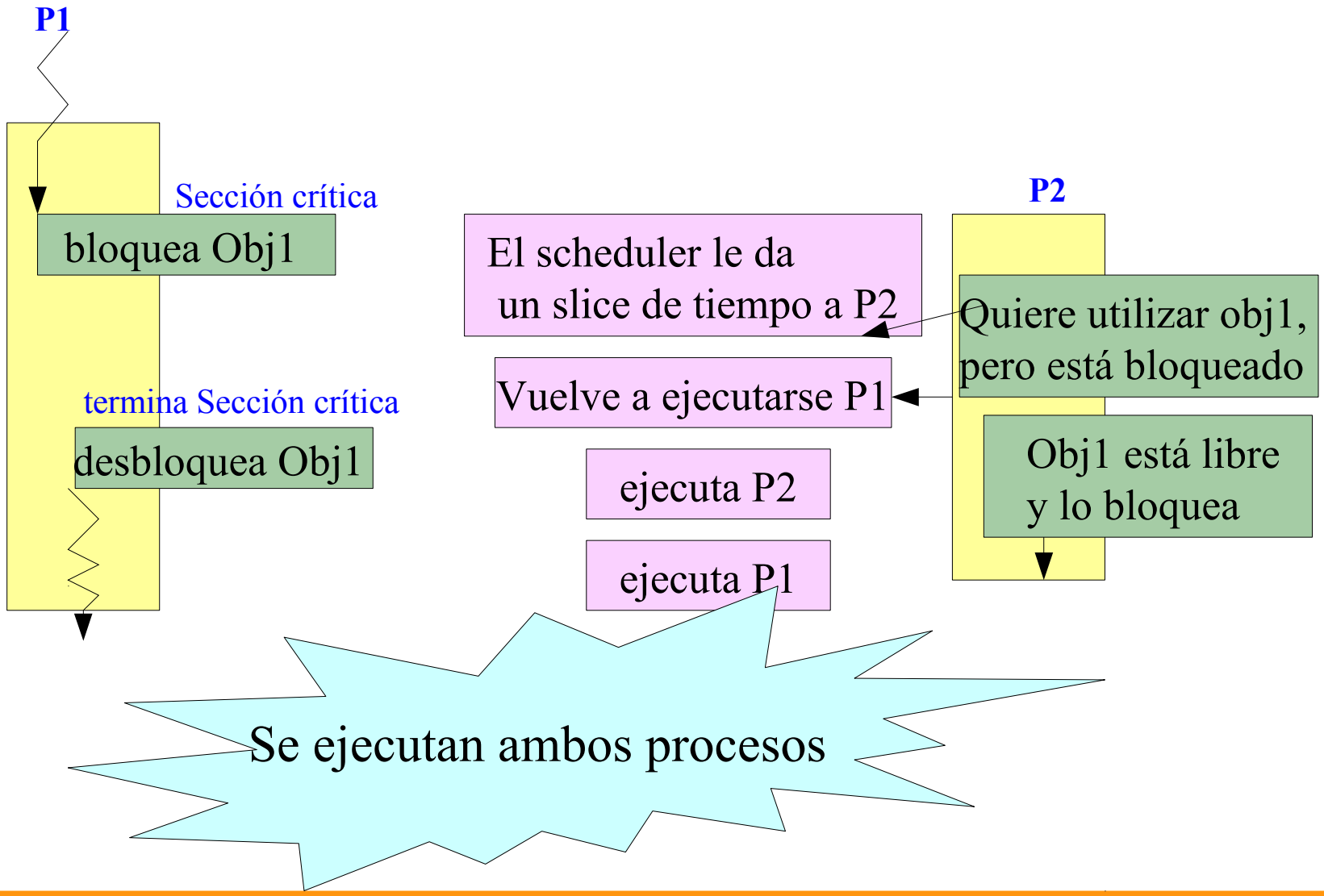
Objeto thread propietario del lock

- ¿Sobre que objeto se los puede invocar?

Sobre el que se sincroniza

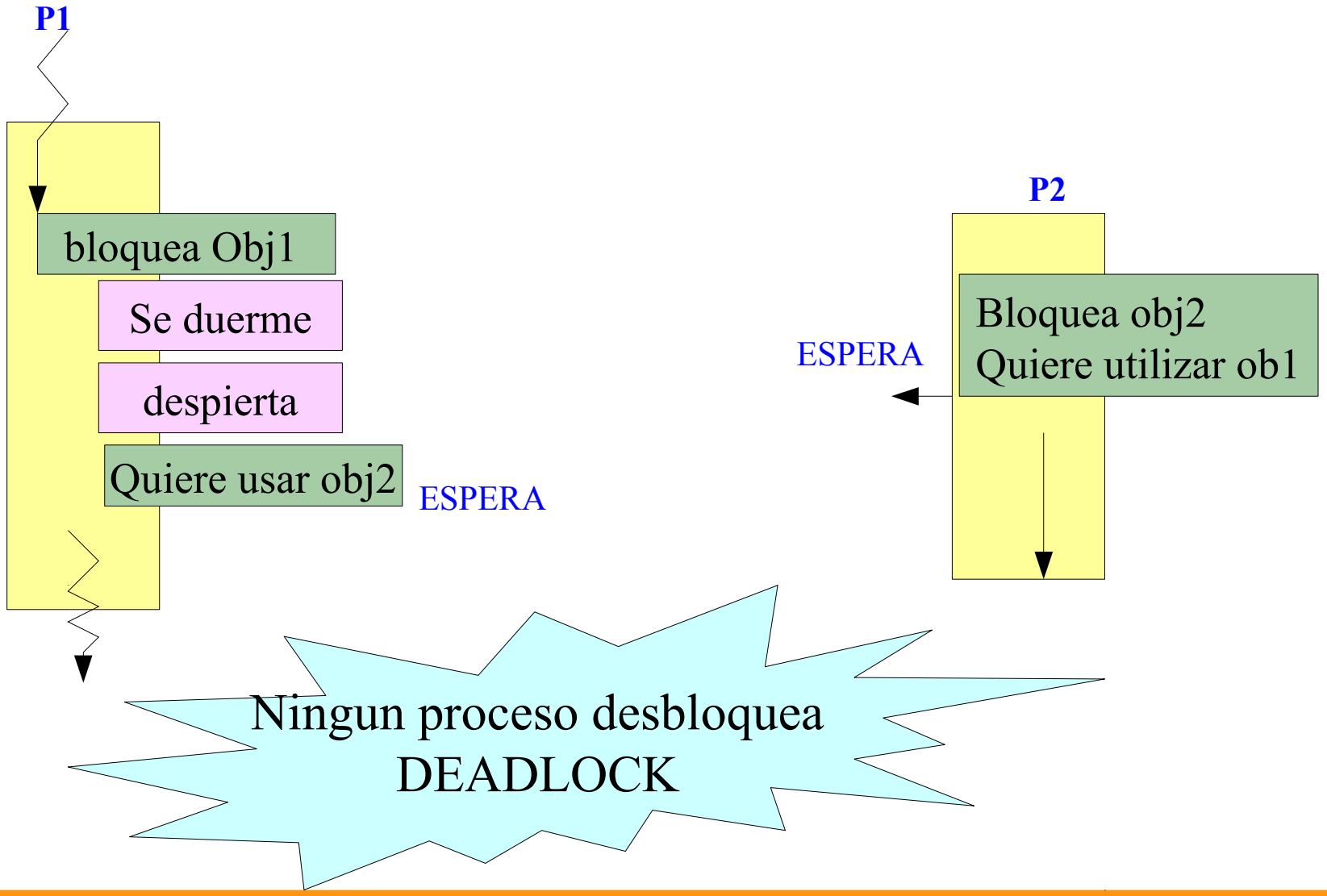
Situaciones

- a)



Situaciones

- b)



Métodos de Object para sincronizar

- ***public final void wait() throws InteruptedException***
  - Espera indefinida hasta que reciba una notificación.
- ***public final void wait(long timeout) throws InteruptedException***
- ***public final wait(long timeout, int nanos)***
  - El thread que ejecuta el método se suspende hasta que recibe una notificación o luego de ese tiempo
- ***public final void notify()***
  - Notifica al objeto un cambio de estado, esta notificación es transferida a solo uno de los threads que esperan
- ***public final void notifyAll()***
  - Notifica a todos los threads que esperan (han ejecutado un wait) sobre el objeto.

Cómo utilizar el método `wait()`

- Utilizar dentro de un método ***synchronized*** y dentro de un ciclo indefinido que verifica la condición:

```
synchronized void hacerCondicion(){
   while (!Condicion)
            this.wait(); //condición cierta
      }
```

- Cuando se suspende el thread en el wait, libera el lock que poseía sobre el objeto.
- La suspensión del thread y la liberación del lock son atómicos (nada puede ocurrir entre ellos).
- La activación del thread y la toma del lock del objeto son también atómicos

Cómo usar el método `notify()`

- Utilizar dentro de un método ***synchronized*** :

```
synchronized void cambiaCondicion(){
    ...//algo cambia y la condición se cumple
    this.notifyAll();
    }
```

- Muchos thread pueden estar esperando sobre el objeto:
  - Con ***notify()*** solo un thread (no se sabe cual) es despertado.
  - Con ***notifyAll()*** todos los thread son despertados y cada uno decide si la notificación le afecta, o si no, vuelve a ejecutar el wait().

- El proceso suspendido debe esperar hasta que el procedimiento que invoca ***notify()*** o ***notifyAll()*** libere el lock del objeto.

Ejemplos de sincronización

- La variable ***sucede*** es la que define que espere un thread y que prosiga el otro

Thread 1

```
public synchronized void esperaEvento() {
  //realiza una vez para c/evento que puede no ser esperado
        while(!sucede) {
                  try { this.wait(); }
                catch (InterruptedException e) {}
        }
        System.out.println("sucede ");
}       //suspende el proceso y lo encola
```

Thread 2

```
public synchronized notificaEvento(){
          sucede=true;
          this.notify();
}
```

Comunicación entre threads

- El thread A está esperando que el thread B le envíe un mensaje

Thread A

```
public synchronized void esperaMensaje(){
        try { this.wait(); }
        catch (InterruptedException e) {}
      }
```

Thread A

```
public void esperaMensaje() {
   while (tieneMensaje == false){
         Thread.sleep(100);
       }
  }
```

Thread B

```
public synchronized void poneMensaje(){
         ...
         this.notify();
}
```

Thread B

```
ublic void poneMensaje()
{
  ….
  tieneMensaje = true)
 }
```

Ejemplo

- En una rotisería

Rotisería

Chef

Comida

Cliente

Ejemplo

Rotisería
- comida Comida - chef Chef - micliente Cliente
Rotiseria (String nombreChef) +getComida() : Comida +getChef( ): Chef +getCliente(): Cliente +setComida(Comida c): void

Chef
- nroPlato int - rotiseria Rotiseria - nombre String
Chef (Rotiseria r, String nomChef) +run() : void

Ciente
- rotiseria Rotiseria - nombre String
Cliente (Rotiseria r, Stringr nomb) +toString() : String +run() : void

Comida
- nroOrden int Comida (int nro)
+toString() : String +getComida() : ordenNro

Como vamos a sincronizar?

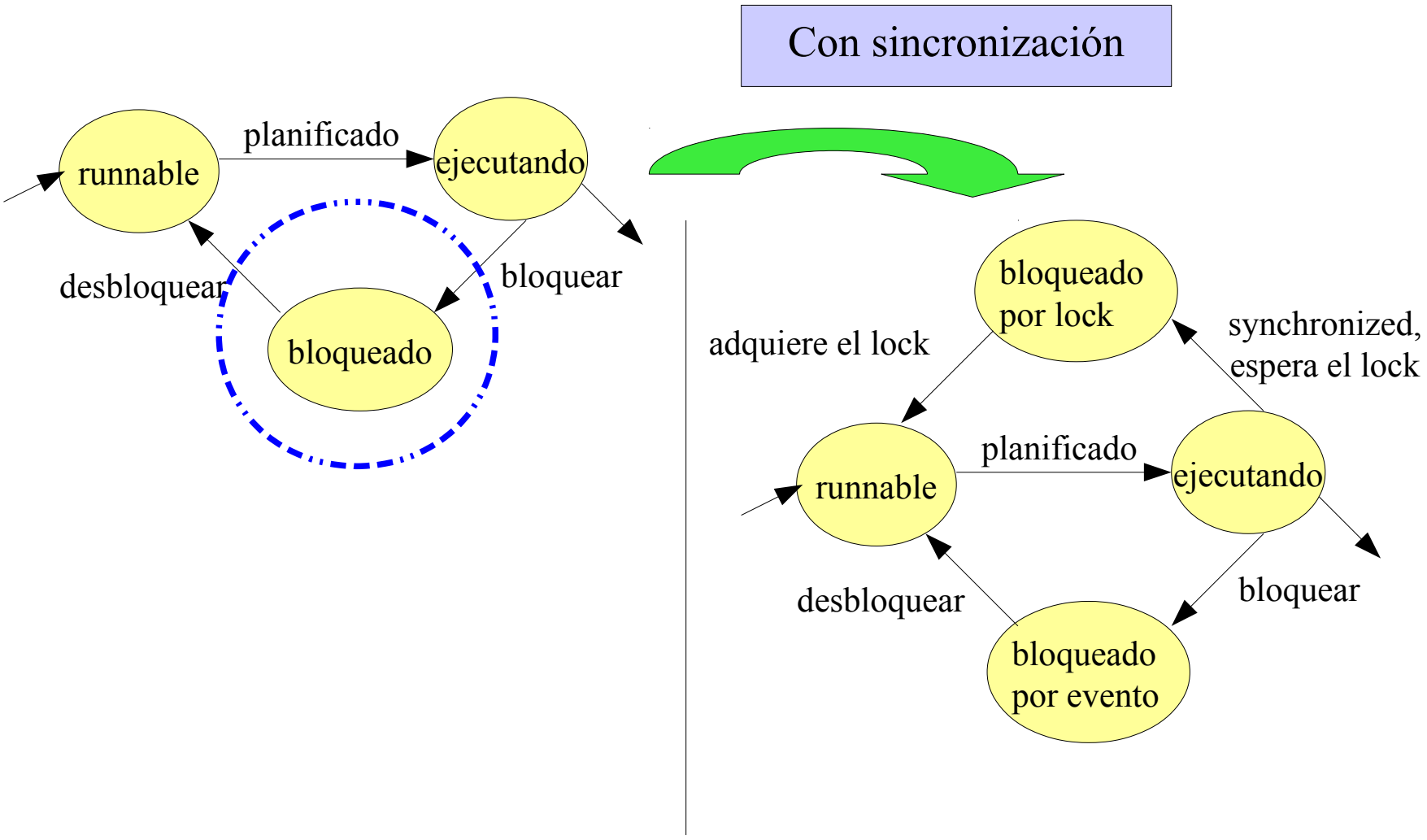
Thread A

```
syncronized (this) {
  while comida != null{
   //hasta que el cliente
   // se lleven la comida
      this.wait();
      }
  }
 …..
syncronized (cliente) {
   // Creo mas comida
   // Le indico a la rotisería
   cliente.notify()
   }
```

Thread B

```
syncronized (this) {
  while comida == null{
      this.wait();
      }
  }
,,,
syncronized (chef) {
      vacio comida = null;
      chef.notify()
      }
```

Entonces …


Con sincronización
runnable
planificado
ejecutando
desbloquear
bloquear
bloqueado
bloqueado por lock
adquiere el lock
synchronized, espera el lock
runnable
planificado
ejecutando
desbloquear
bloquear
bloqueado por evento

Entonces …

Con sincronización e interacción entre hilos