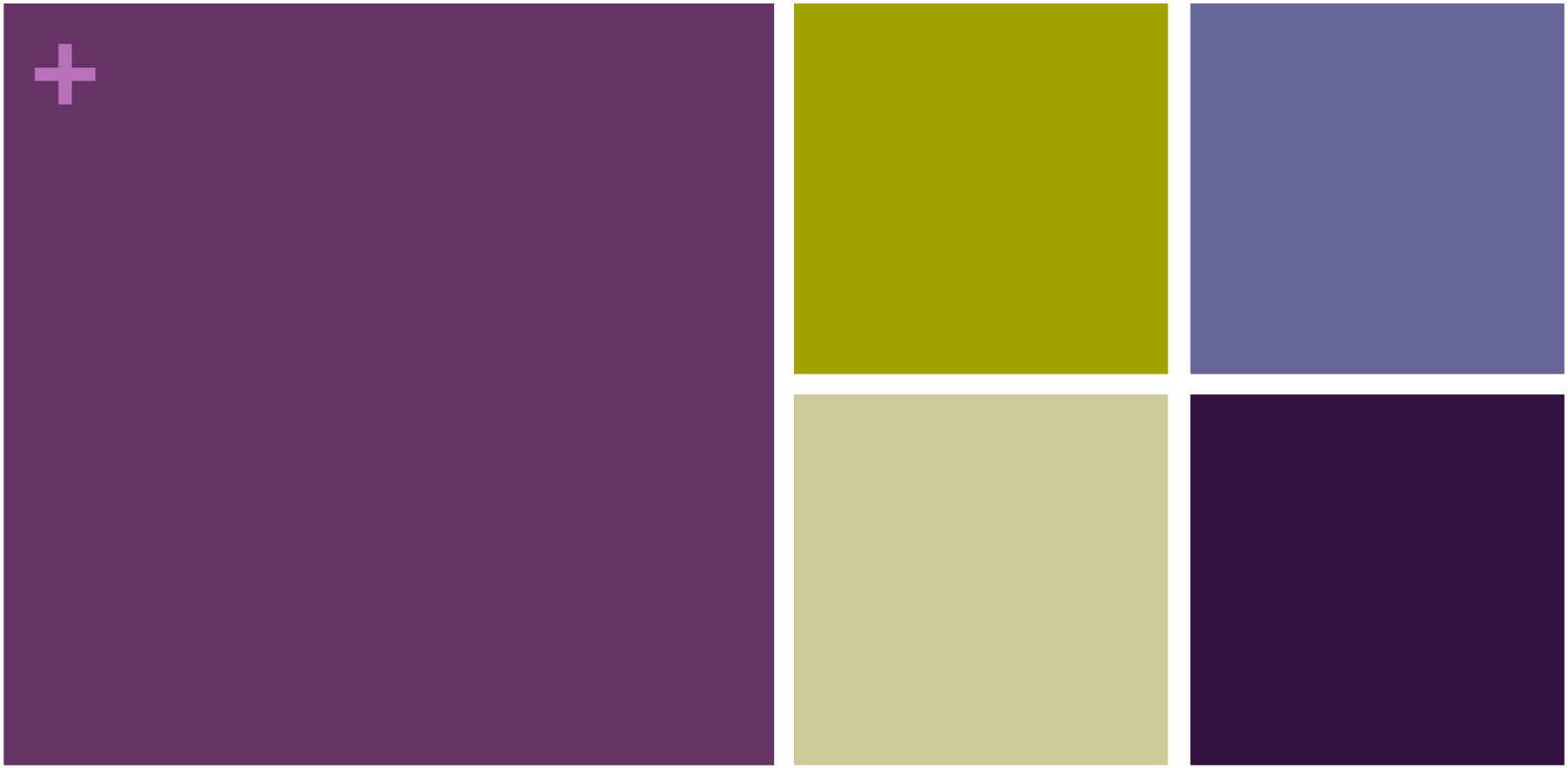


Sistemas Distribuidos

Elección distribuida

Rodrigo Santamaría

Elección distribuida

- Objetivo
- Requisitos básicos
- Requisitos adicionales
- Despliegue
- Entrega y defensa

Elección distribuida

Objetivo

- Implementar un método de elección distribuida basado en el algoritmo *Bully* (tema 5, diap. 33 y sigs.)
  - En el libro, capítulo 4.2.1 (pág. 42)
- El algoritmo se activará
  - Si un proceso detecta que el coordinador actual no responde
  - Cuando un proceso comience/reanude su ejecución

Elección distribuida

- Objetivo
- Requisitos básicos
- Requisitos adicionales
- Despliegue
- Entrega y defensa

Elección distribuida

Requisitos básicos

- E1 (**Seguridad**): al final de la elección todos los procesos tienen el mismo coordinador
- E2 (**Pervivencia**): todos los procesos en ejecución participan de la elección, y al final fijan un coordinador
- No debe ocurrir
  - Espera ocupada
  - Interbloqueos
  - Inanición

Elección distribuida

Proceso

- Clase que extiende de `Thread`
- Información relativa a la elección
  - Atributos:
    - `id`: identificador del proceso
    - `coordinador`: contiene el id acordado como coordinador
    - `estado de la elección`
      - `acuerdo`, `eleccion_activa` o `eleccion_pasiva`
  - Métodos:
    - `ok()`, `eleccion()`, `coordinador()`
      - Implementarán el comportamiento de respuesta a los mensajes homónimos
- Información relativa a la ejecución
  - `estado`: indica si el proceso está corriendo o parado
  - `arrancar()` y `parar()` para cambiar su estado
  - `computar()`: simula la tarea del coordinador
  - `run()`: implementa la tarea que realiza el proceso

Elección distribuida

Proceso: ejecución

- Método ***run()***

```
1     si estado=parado
2         terminar
3     si no
4         esperar entre 0.5 y 1 s
5         valor ← coordinador.computar()
6         si valor < 0 o error
7             elección()
8         volver a 1
```

Elección distribuida

Proceso: arrancar y parar

- `terminar` en el pseudocódigo anterior puede significar:
  - Parar la ejecución del proceso (p. ej. `wait`)
  - Terminar la ejecución del proceso (salir del método `run`)
  - OJO: Terminar nunca significa no hacer nada y volver al paso 1, eso es **espera ocupada**
- Dependiendo de cómo paremos, para volver a arrancar:
  - Sacar de la espera (`notify`, `release`)
  - Volver a arrancar el proceso (`start`)
    - NOTA: Java no permite re-arrancar `Thread` (ver API), es necesario crear de nuevo el proceso y después arrancarlo.

Elección distribuida

Proceso: arranque y elecciones

- Al parar y luego arrancar un proceso, si este es el de id más alto, puede que el resto hayan elegido otro coordinador
  - Tendríamos una situación en la que el coordinador no es el proceso con identificador más alto
  - **Solución**: dentro del proceso de arranque, se inician unas elecciones

Elección distribuida

Proceso: ejecución

- Método ***computar()***

```
si estado=parado
    devolver -1 o error
si no
    esperar entre 0.1 y 0.3 s
    devolver 1
```

Podemos parar de manera 'ficticia' mediante una variable estado (ejemplo de arriba), o finalizar del todo el proceso, y por tanto la llamada no retorna o devuelve un error

NOTA: las **esperas** se pueden realizar mediante `Thread.sleep` pues simulan tiempo de consumo de CPU. Los **timeout** de espera a mensajes de respuesta NO se pueden modelar con `sleep` pues sería una espera ocupada. `wait()` o `Semaphore` tienen versiones que permiten timeouts

Elección distribuida

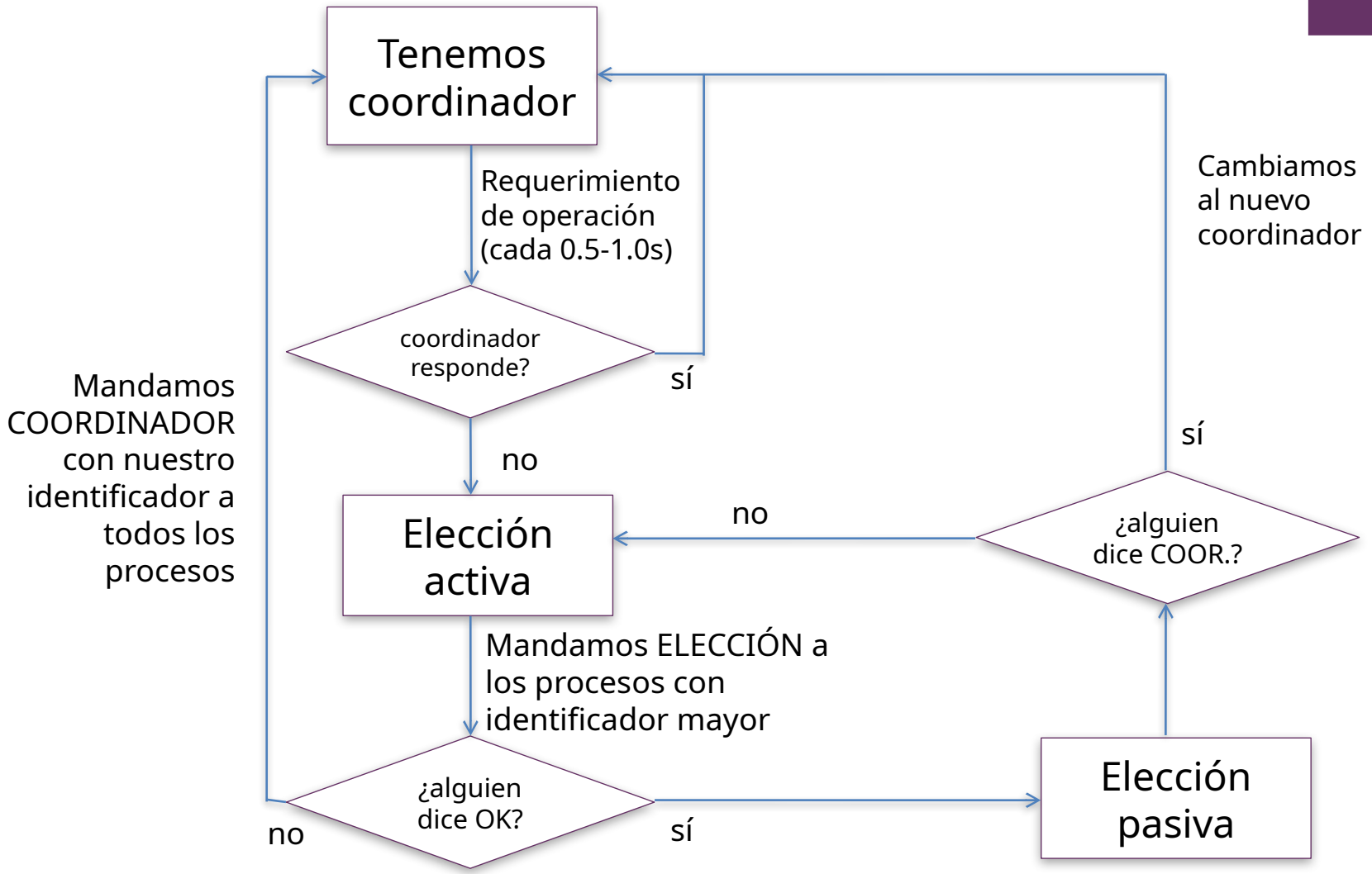
Pseudocódigo: elección del proceso p_i

```
inicio
pj.eleccion() para todo pj con j>i
esperar mensaje respuesta (timeout 1s)
si recibe mensaje respuesta
    esperar mensaje coordinador (timeout 1s)
    si recibe mensaje coordinador(x)
        pi.coordinador ← x
        fin
    si no recibe mensaje coordinador tras timeout
        volver a inicio
si no recibe mensaje respuesta tras timeout
    pi.coordinador ← i //se hace nuevo coordinador
    pj.coordinador(i) para todo pj (j=1…numProcesos)*
    fin
```

- Si un proceso se recupera o se lanza un proceso sustituto con el mismo `id`, éste comienza una nueva elección, aunque el coordinador actual esté acordado y funcionando
- Si *i=numProcesos*, entonces iría directamente al último **si no**
- * Podría enviarse el mensaje *coordinador* sólo a los procesos p_j / j<i (como en diap. 13e)

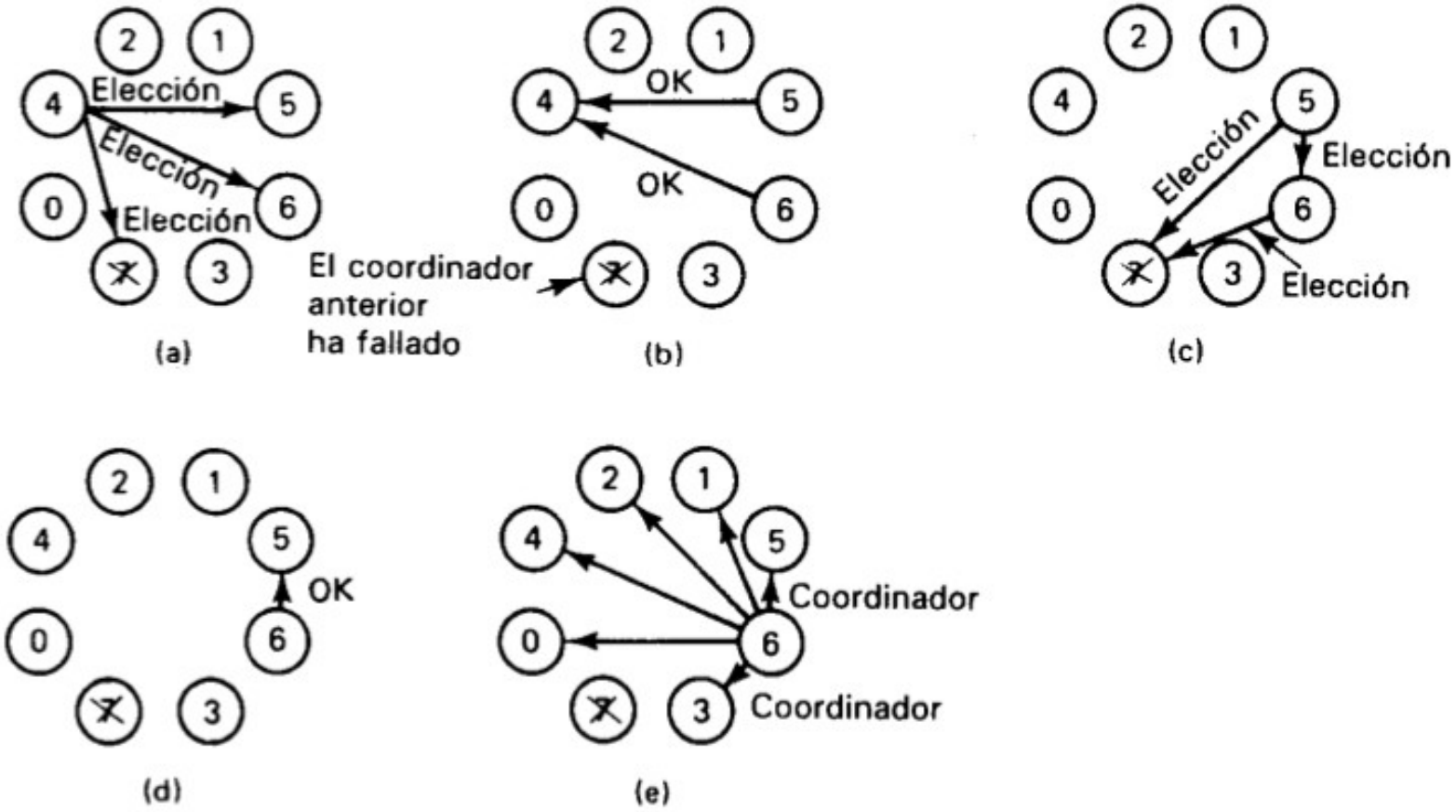
Elección distribuida

Diagrama de flujo



Elección distribuida

Esquema general



Elección distribuida

Entorno de programación

- El lenguaje de programación y bibliotecas asociadas **no es un requisito**
  - Libertad de elección y diseño, mientras se cumplan los requisitos
  - Se dan las pautas y ayuda en Java+Jersey, pero no es obligatorio usar este esquema
- El **contexto sí es un requisito**: la práctica se defenderá en los ordenadores del laboratorio de informática/aula SUN en el sistema operativo Linux instalado.

Elección distribuida

- Objetivo
- Requisitos básicos
- Requisitos adicionales
- Despliegue
- Entrega y defensa

Elección distribuida

Repetición de código = suspenso

- El uso de **repetición de código** es motivo de **suspenso**
- Es una práctica de programación:
  - Deficiente
  - No escalable
  - Fuente de errores
  - Impropia de un ingeniero

```
Proceso p1=new Proceso(1);
Proceso p2=new Proceso(2);
Proceso p3=new Proceso(3);
Proceso p4=new Proceso(4);

p1.start();
p2.start();
p3.start();
p4.start();
```

```
const int maxP=4;
List<Proceso> lista=
       new List<Proceso>();

for(int i=0;i<maxP;i++)
  {
  Proceso p=new Proceso(i);
  p.start();
  lista.add(p);
  }
```

Elección distribuida

Requisitos adicionales (opcionales)

- Interfaz
  - Minimizar métodos y parámetros
  - Claridad, documentación
- Arquitectura
  - Minimizar nº de clases
  - Despliegue sencillo
- Características
  - **Extensibilidad**: facilidad de cambiar a otros nodos/máquinas
  - **Escalabilidad**: facilidad de ampliar a más nodos/máquinas
  - **Rendimiento**: tiempo en llegar a un acuerdo, afinar timeouts
  - **Tolerancia**: fallo de muchos/todos los procesos, recuperación en medio de elección, etc.

Elección distribuida

- Objetivo
- Requisitos básicos
- Requisitos adicionales
- Despliegue
- Entrega y defensa

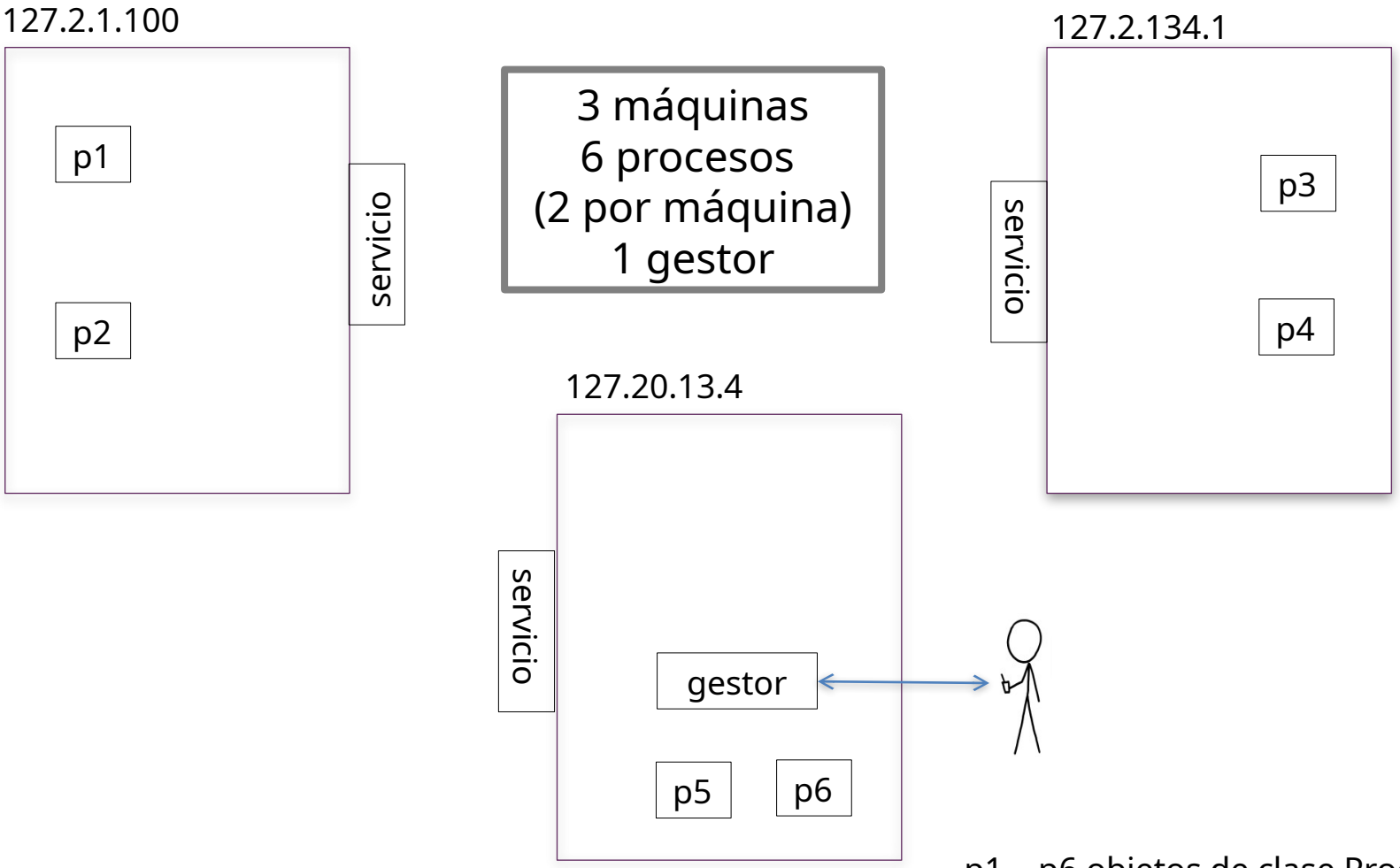
Elección distribuida

Clases

- Vamos a necesitar 3 clases
  - **Proceso**: implementada por cada uno de los hilos que participarán en la elección
  - **Gestor**: ejecutada en una de las tres máquinas, permite parar/arrancar procesos, y consultar el estado de los procesos (coordinador y estado de la elección)
  - **Servicio**: mantiene la lista de procesos de cada máquina e implementa servicios REST para acceder a sus distintos métodos

Elección distribuida

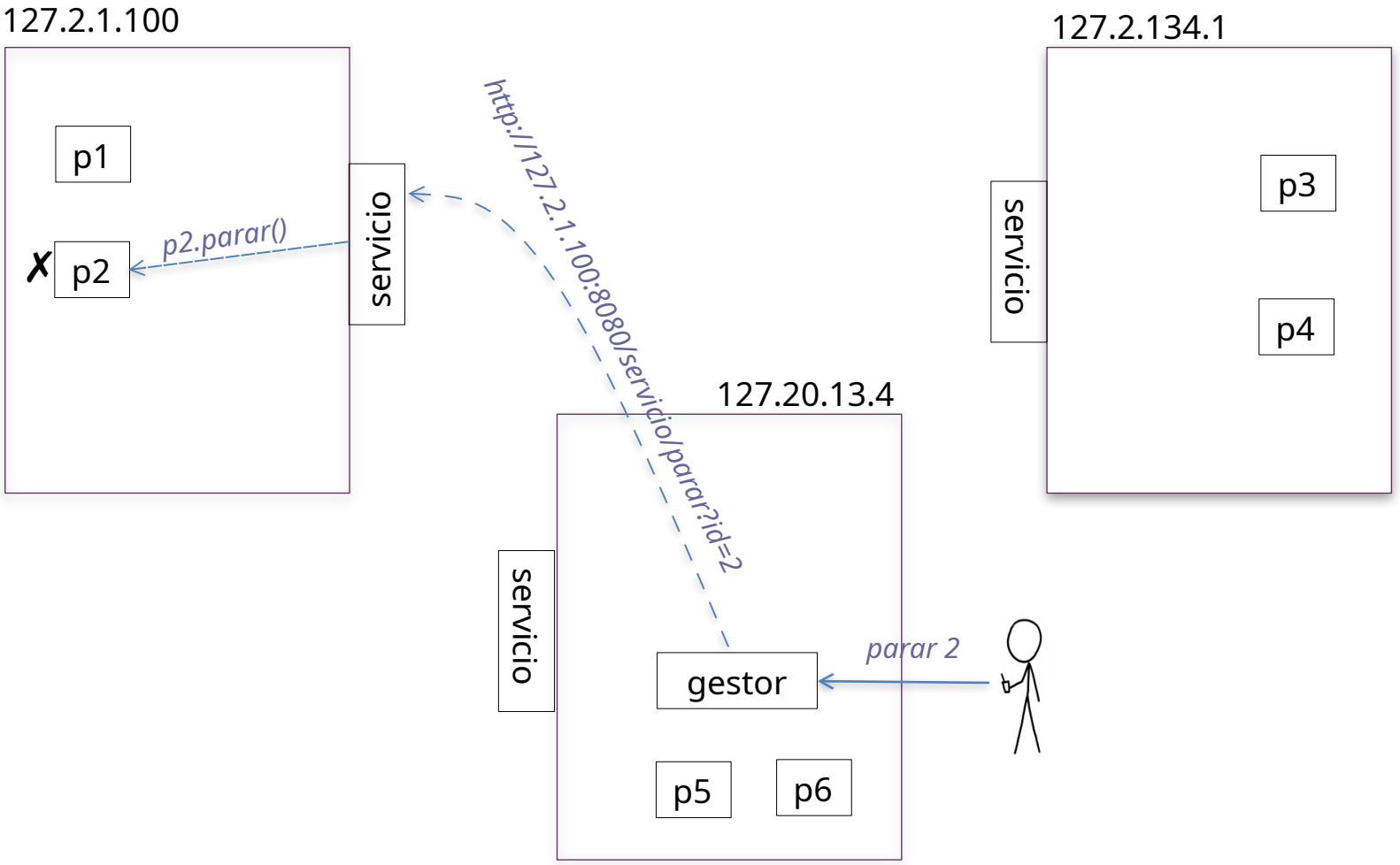
Despliegue



p1… p6 objetos de clase Proceso

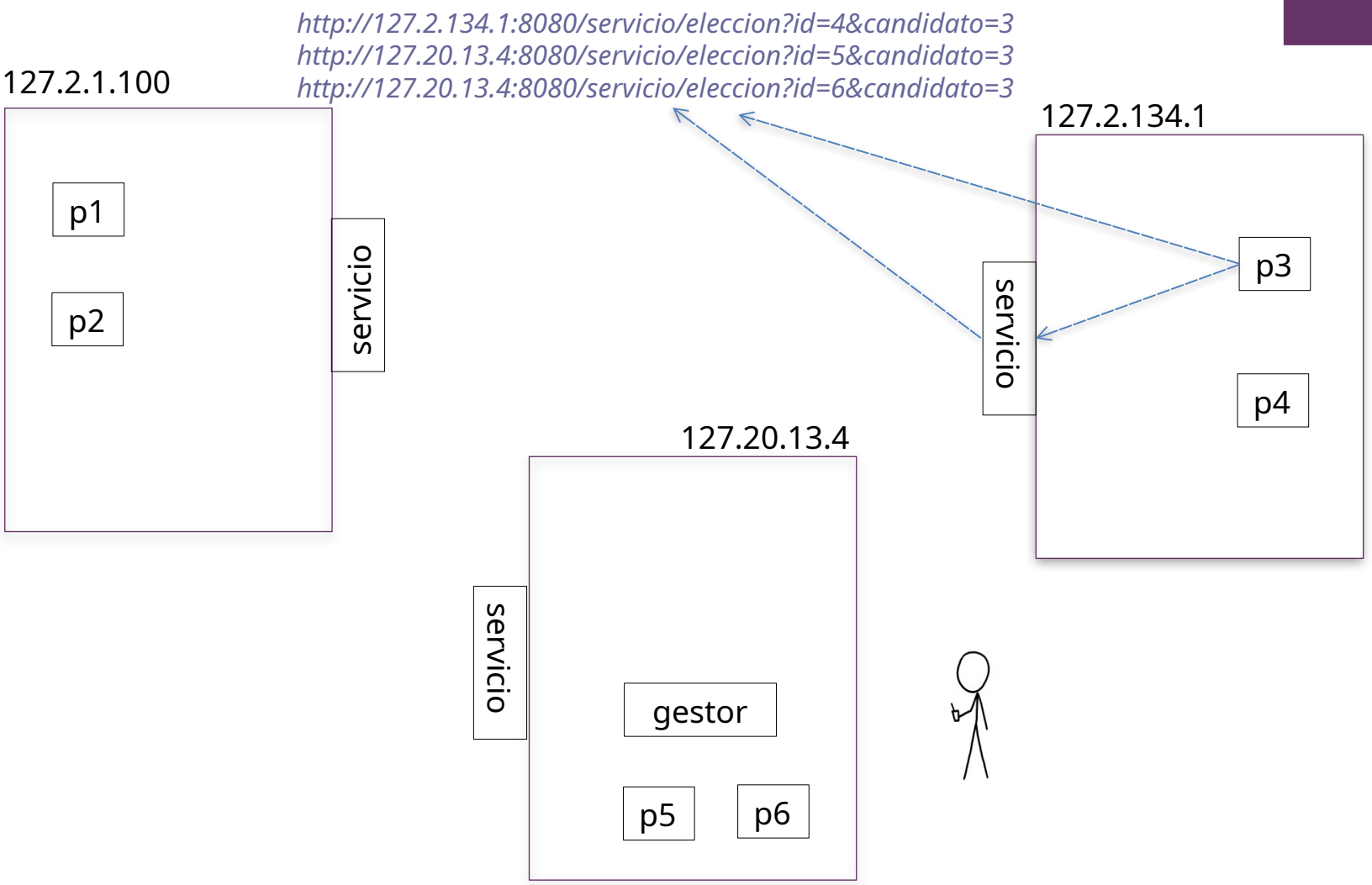
Elección distribuida

Despliegue



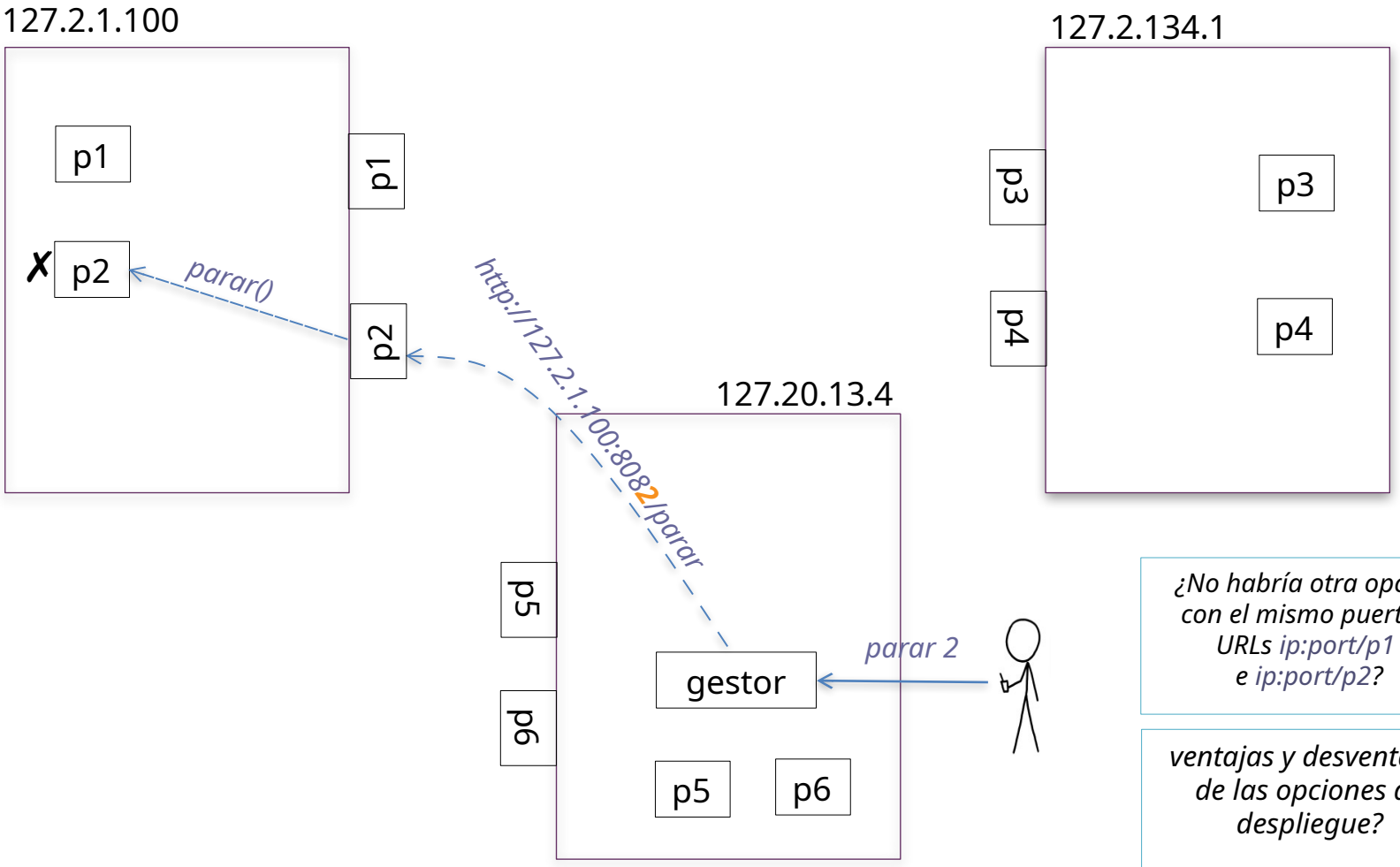
Elección distribuida

Despliegue



Elección distribuida

Despliegue: opción sin servicio



¿No habría otra opción con el mismo puerto y URLs ip:port/p1 e ip:port/p2?

ventajas y desventajas de las opciones de despliegue?

Elección distribuida

- Objetivo
- Requisitos básicos
- Requisitos adicionales
- Despliegue
- Entrega y defensa

Elección distribuida

Entrega

- La entrega se realizará, por parejas, a través de Studium
- Ambos miembros de la pareja entregarán un fichero **tar.gz**
  - Con nombre *Nombre1Apellido1Nombre2Apellido2.tar.gz*
  - Contendrá
    - Las clases `Proceso`, `Gestor` y `Servicio*`
    - Cualquier otra clase, script, etc. necesario para la ejecución
    - No es necesario incluir bibliotecas externas (jersey, etc.)
- Fecha límite de entrega: 8 **de mayo a las 23h**

*Esta última clase puede obviarse dependiendo del tipo de despliegue elegido

Elección distribuida

Defensa

- El horario de defensa se publicará tras la entrega
- Las defensas tendrán lugar la semana siguiente a la entrega
- Se realizarán por parejas, en una de las aulas
  - Tiene replicada toda la infraestructura del laboratorio de informática
  - Cada defensa durará unos 20 minutos, en los que:
    - Se probará exhaustivamente la práctica
    - Se examinará el código fuente
    - Se harán preguntas a ambos miembros de la pareja