Sistemas Distribuidos

Jueves, 9 de febrero

Coordinación y acuerdo

Coordinación y acuerdo

- Requerimientos:
  - Un grupo de procesos deben coordinar sus acciones
  - Un grupo de procesos deben ponerse de acuerdo en cuanto al valor de una o más variables
- Premisas fundamentales en sistemas distribuidos

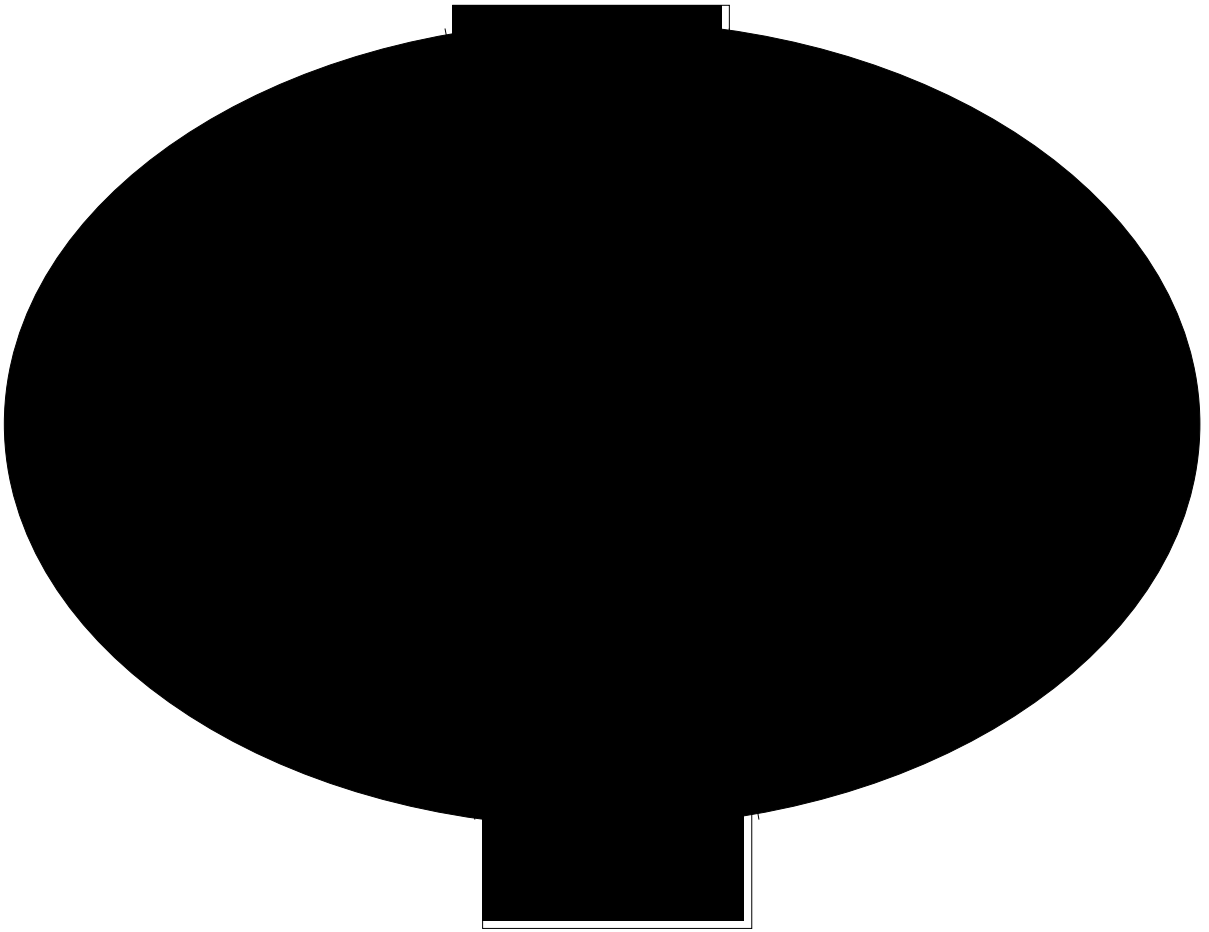
Detección de fallos

- Se asume que cada par de procesos está conectado a través de canales confiables
- i.e., aunque se produzcan fallos en la red, un protocolo de comunicación enmascara esos fallos, por ejemplo, retransmitiendo mensajes perdidos o corruptos
- Ningún proceso depende de otro para el envío de mensajes

Detección de fallos (2)

- En un canal confiable, el destinatario *eventualmente* recibe el mensaje en su búfer de entrada
- En un sistema sincrónico, se asume también que el mensaje va a ser recibido dentro de un lapso de tiempo determinado
- En un intervalo particular, la comunicación entre algunos procesos puede tener éxito mientras que entre otros podría estar demorada

Partición de la red



Conectividad asimétrica

- En redes punto-a-punto como Internet, la comunicación podría ser asimétrica
  - Comunicación es posible del proceso *p* al *q*, pero no viceversa
- La conectividad podría también ser intransitiva
  - Comunicación es posible del proceso *p* al *q*, y del *q* al *r*, pero *p* no podría comunicarse directamente con *r*
- La suposición de confiabilidad asegura que eventualmente los procesos se comunicarán, aunque la comunicación no se dé al mismo tiempo

Caídas

- Un proceso sólo falla si se cae
- Un proceso *correcto* es el que no presenta fallos en ninguna parte de su ejecución
- ¿Cómo decidimos si un proceso se cayó?
  - Detectores de fallos
  - Detectores de fallos locales
  - La mayoría son *no confiables*
    - Regresan 2 posibles valores: *Unsuspected* o *Suspected*
  - Los detectores confiafles
    - Regresan 2 posibles valores: *Unsuspected* o *Failed*

Detector no confiable

- Cada proceso envía un mensaje "*p* is here" a todos los demás procesos cada *T* segundos
- El detector de fallos estima un tiempo máximo de transmisión de mensajes en *D* segundos
- Si el detector de fallos local donde se encuentra el proceso *q* no recibe un "*p* is here" después de *T* + *D* segundos de haber recibido el último, reporta a *q* que *p* está *Suspected*
- Si luego lo recibe, reporta a *q* que *p* está *OK*

Exclusión mutua distribuida

- Los procesos distribuidos necesitan a menudo coordinar sus actividades
- Si se comparten recursos, la exclusión mutua es necesaria para evitar interferencias y asegurar la consistencia
- Problema tradicional de la *sección crítica*
- Sin embargo:
  - No podemos usar variables compartidas
  - No hay facilidades provistas por el kérnel

Solución

- Exclusión mutua basada sólo en el paso de mensajes
- A veces, los recursos son controlados por servidores que proveen *mutex*
- Pero en los demás casos prácticos, es necesario un mecanismo separado para implementar la exclusión mutua

Algoritmos

Asumimos:

- Sistema es asincrónico
- Procesos no fallan
- Envío de mensajes es confiable
- Todos los mensajes son recibidos intactos y ...
- Sólo una vez

Protocolo

- /* enter critical section – block if necessary */
- enter()
- /* access shared resources in critical section */
- resourceAccesses()
- /* leave critical section – other processes may now enter */
- exit()

Requerimientos

1) Cómo máximo, un proceso puede ejecutarse en la sección crítica a la vez

2) Solicitudes de entrada y salida de la sección crítica eventualmente tienen éxito

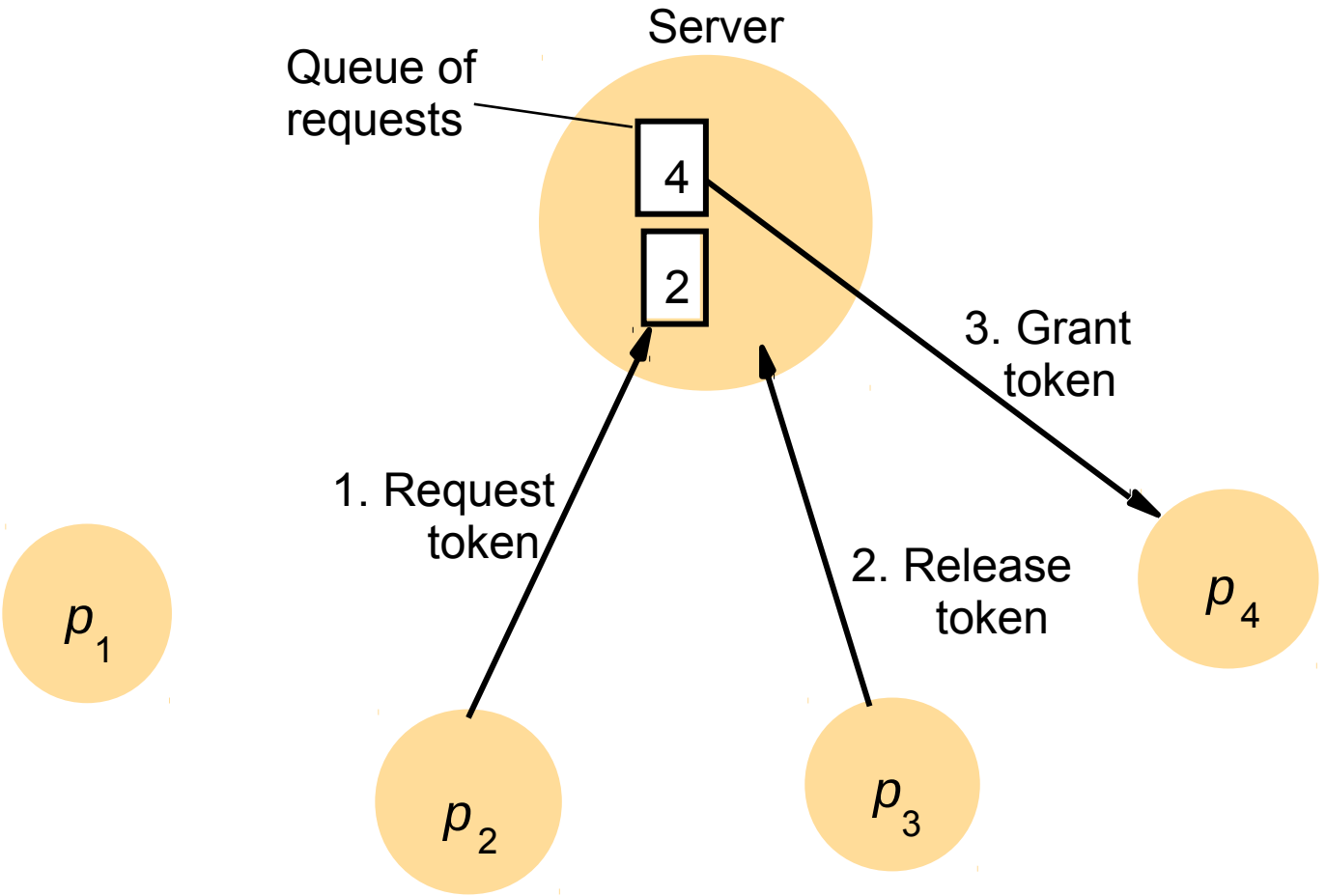
3) Si una solicitud para entrar a la sección crítica *ocurrió-antes* que otra, entonces la entrada se otorga en ese orden

La condición # 2 implica que se evitarán tanto los bloqueos mutuos como la inanición

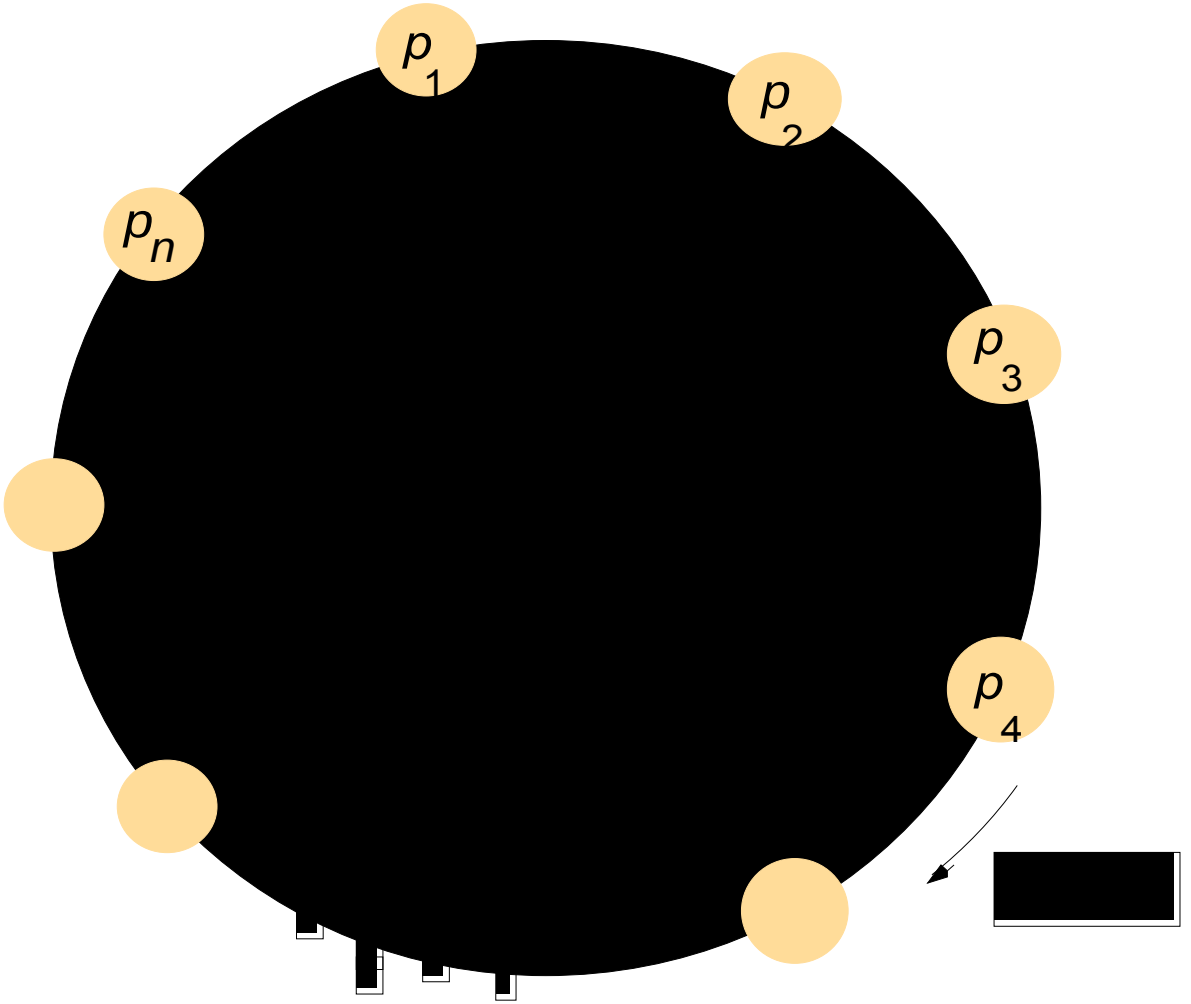
Evaluación

- Ancho de banda
- Retardo
- Efecto sobre el rendimiento del sistema

Servidor central



Anillo



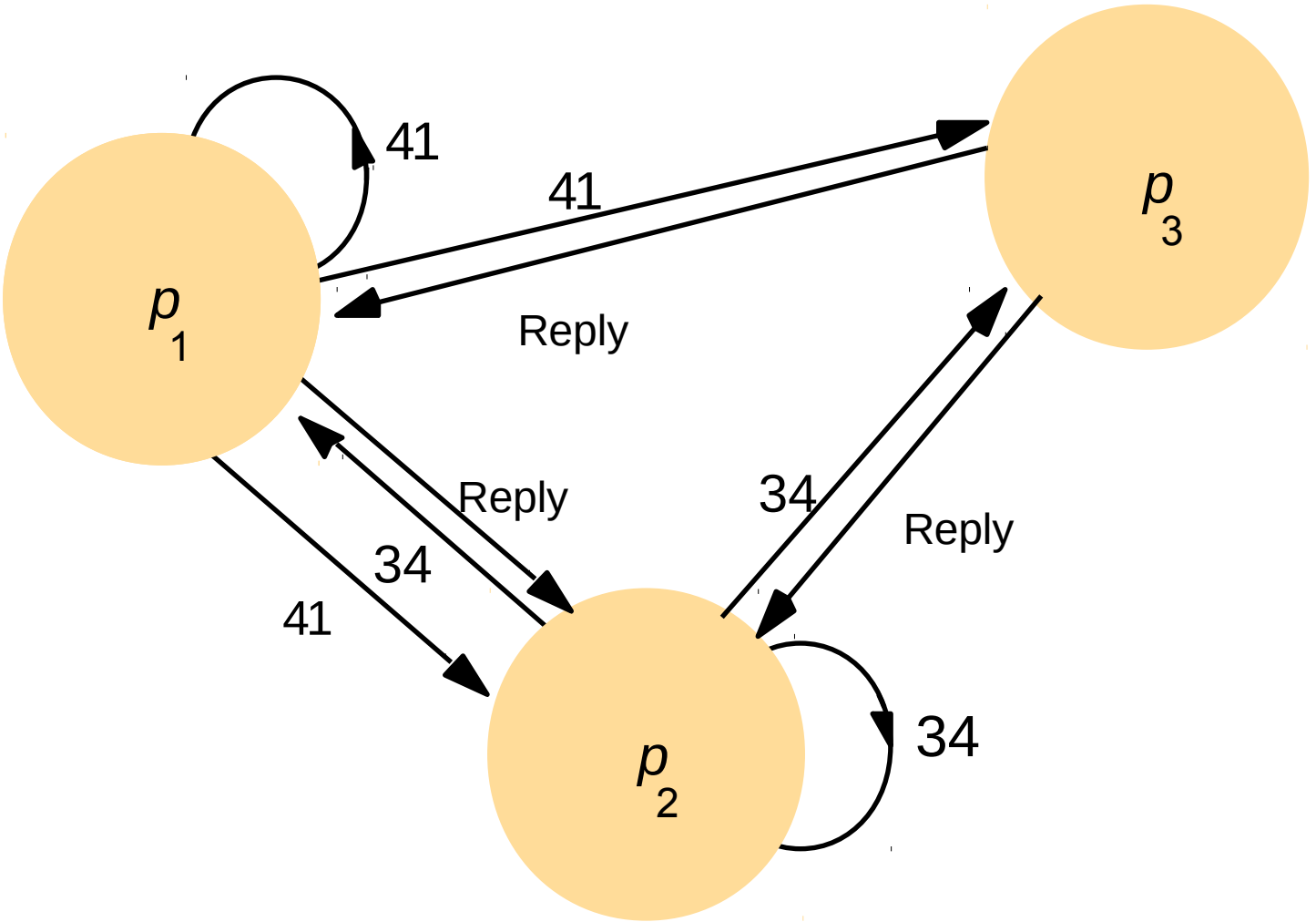
Ricart & Agrawala

On initialization
    state := RELEASED;
To enter the section
    state := WANTED;
    Multicast *request* to all processes;
    T := request's timestamp;
    Wait until (number of replies received = (*N* – 1));
    state := HELD;

request processing deferred here

On receipt of a request $<T_i, p_i>$ *at* p_j $(i \neq j)$
    if (*state* = HELD or (*state* = WANTED *and* $(T, p_j) < (T_i, p_i)$))
    then
        queue *request* from p_i without replying;
    else
        reply immediately to p_i;
    end if
To exit the critical section
    state := RELEASED;
    reply to any queued requests;

Ejemplo



Maekawa's voting

On initialization
  state := RELEASED;
  voted := FALSE;
For p_i to enter the critical section
  state := WANTED;
  Multicast *request* to all processes in V_i;
  Wait until (number of replies received = K);
  state := HELD;
On receipt of a request from p_i at p_j
  if (*state* = HELD *or voted* = TRUE)
  then
    queue *request* from p_i without replying;
  else
    send *reply* to p_i;
    voted := TRUE;
  end if

For p_i to exit the critical section
  state := RELEASED;
  Multicast *release* to all processes in V_i;
On receipt of a release from p_i at p_j
  if (queue of requests is non-empty)
  then
    remove head of queue – from p_k, say;
    send *reply* to p_k;
    voted := TRUE;
  else
    voted := FALSE;
  end if

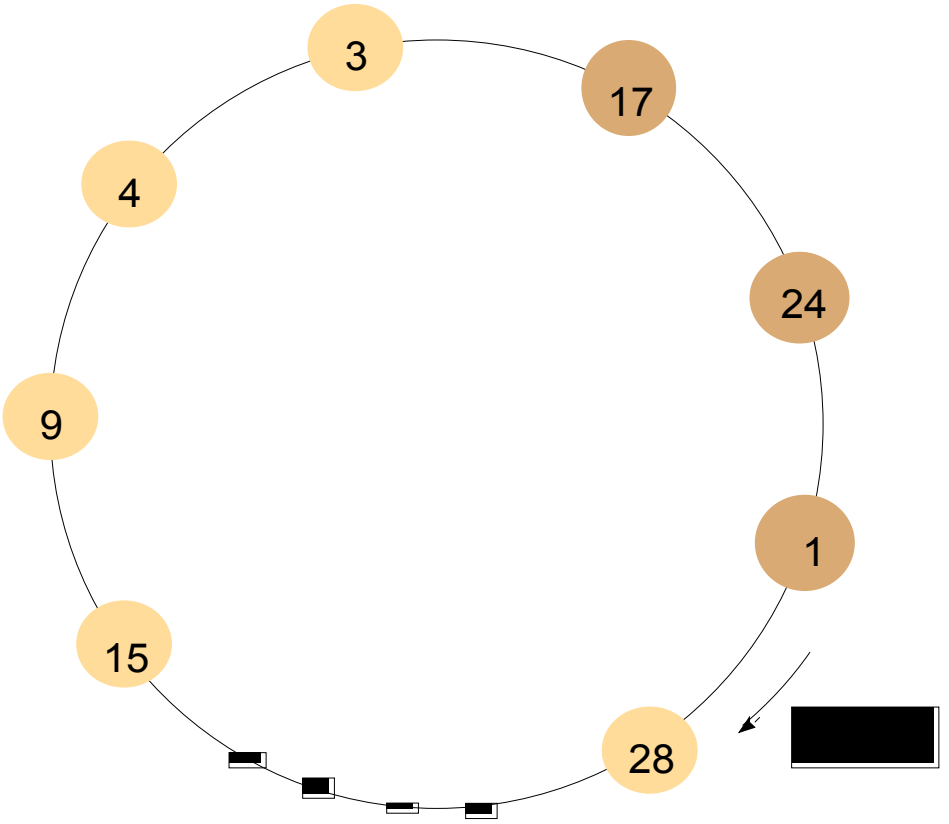
Tolerancia a fallos

- ¿Qué pasa cuando un mensaje se pierde?
- ¿Qué pasa cuando un proceso se cae?

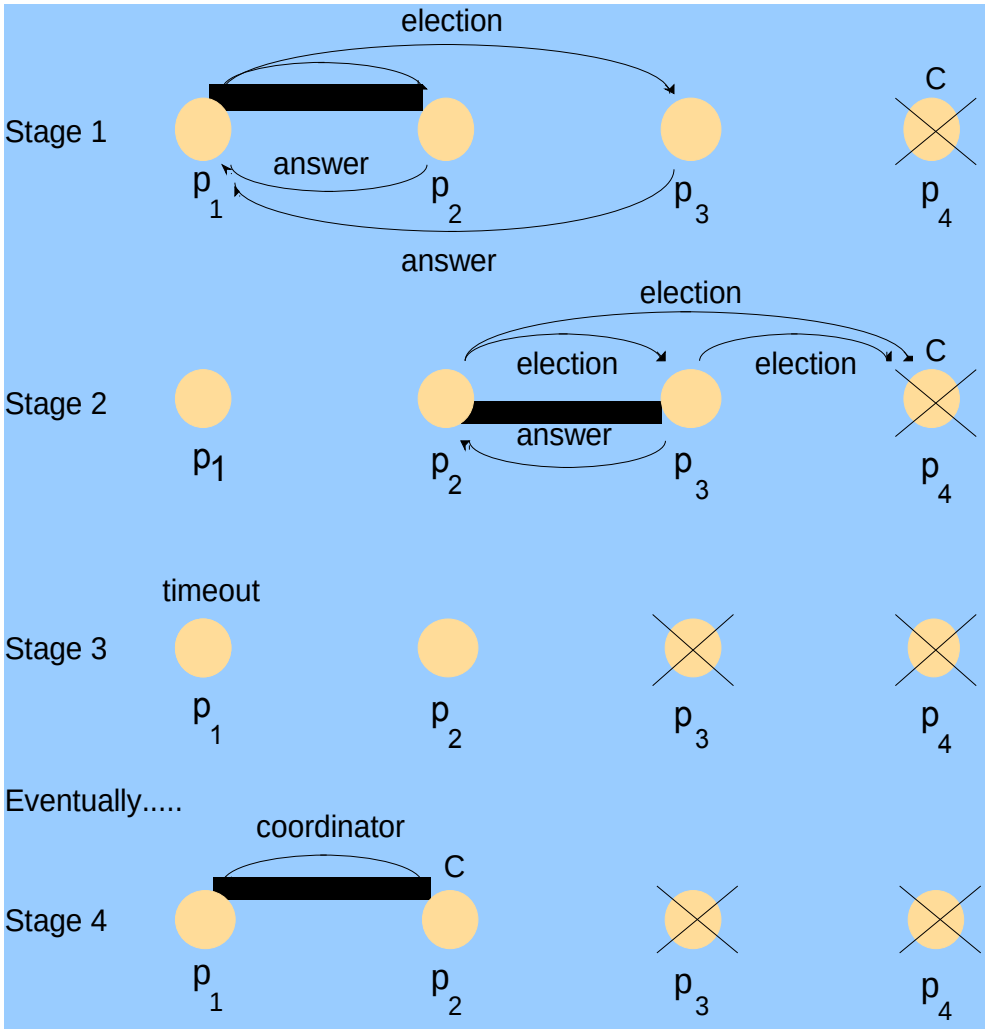
Elecciones

- Un algoritmo de elección permite escoger un único proceso para desempeñar un papel particular
- Por ejemplo, en una variante del algoritmo de servidor central para exclusión mutua, un proceso es escogido para funcionar como servidor
- Un proceso *llama la elección*, es decir, inicia el proceso

Anillo



Bully


election
C
Stage 1
answer
p1
p2
p3
p4
answer
election
election
election
C
Stage 2
answer
p1
p2
p3
p4
timeout
Stage 3
p1
p2
p3
p4
Eventually.....
coordinator
C
Stage 4
p1
p2
p3
p4