

GBC074 – Sistemas Distribuídos

Coordenação

Introdução

- A partir de agora começamos a ver alguns problemas básicos em sistemas distribuídos e discutir formas de solucioná-los
- Alguns exemplos:
 - Exclusão mútua
 - Eleição de líder
 - Sincronização de relógio
 - Consenso
 - ...

Exclusão Mútua

- Garante acesso exclusivo à uma **região crítica**
- Em um sistema monolítico:
 - uma variável global, um lock, ou outra primitiva de sincronização podem ser usadas na sincronização
- Em um sistema distribuído
 - Como controlar o acesso de múltiplos processos a um recurso compartilhado, garantindo que cada processo controla **exclusivamente** aquele recurso durante seu acesso?

Exemplos com threads

- Dados compartilhados

```
1  #include<pthread.h>
2  #include<stdio.h>
3  #include<stdlib.h>
4
5  #define ROUNDS 10000
6
7  static int global_i;
8
9  void* inc(void* id) {
10     long my_id = (long) id;
11     int i;
12
13     printf("Hello from thread %ld \n", my_id);
14     for(i = 0; i < ROUNDS; ++i) {
15         global_i++;
16     }
17     return NULL;
18 }
19
```

Exemplos com threads

- Dados compartilhados


```
20  int main(int argc, char* argv[]) {
21      long thread, thread_count;
22      pthread_t* thread_handles;
23
24      if(argc < 2) {
25          printf("usage: %s <number of threads>", argv[0]);
26          return 1;
27      }
28
29      global_i = 0;
30      thread_count = strtol(argv[1], NULL, 10);
31      thread_handles = malloc(thread_count*sizeof(pthread_t));
32
33      for (thread = 0; thread < thread_count; thread++)
34          pthread_create(&thread_handles[thread], NULL, inc, (void*) thread);
35
36      for (thread = 0; thread < thread_count; thread++)
37          pthread_join(thread_handles[thread], NULL);
38
39      printf("\n\nFinal value of 'global_ithread' is %d\n", global_i);
40
41      free(thread_handles);
42      return 0;
43  }
```

Exemplos com threads

- Dados compartilhados:
 - Execução

```
material/coding/01-threads via C v13.1.6-clang → make 01-inc
cc      01-inc.c      -o 01-inc
material/coding/01-threads via C v13.1.6-clang → ./01-inc 10
Hello from thread 0
Hello from thread 1
Hello from thread 6
Hello from thread 2
Hello from thread 4
Hello from thread 5
Hello from thread 7
Hello from thread 9
Hello from thread 8
Hello from thread 3

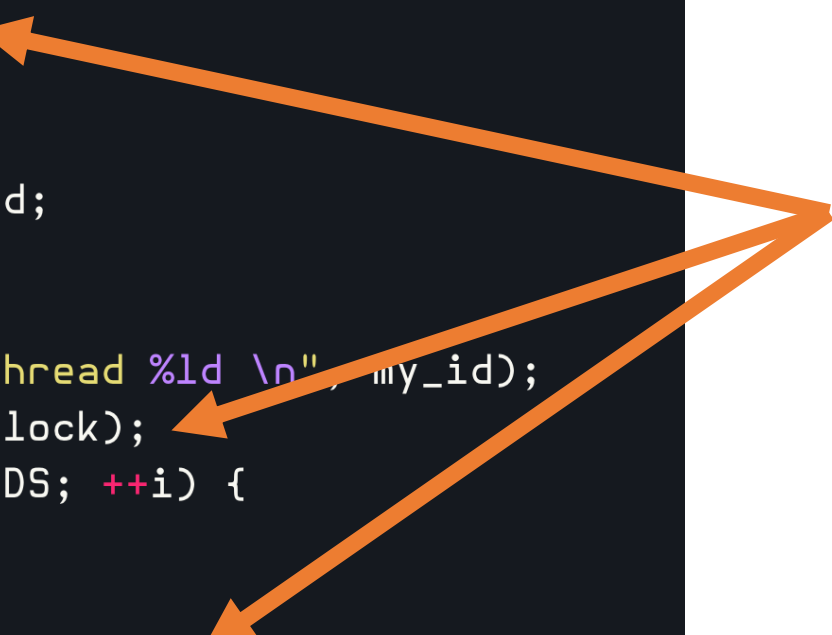
Final value of 'global_ithread' is 36744
material/coding/01-threads via C v13.1.6-clang →
```

A sad face icon (a circle with a downward curve for a mouth) is positioned to the right of the terminal output. A thick orange arrow points from the sad face icon down towards the line "Final value of 'global_ithread' is 36744".

Exemplos com threads – versão 2

- Dados compartilhados:

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #define ROUNDS 10000
6
7  static int global_i;
8  pthread_mutex_t lock;
9
10 void *inc(void *id) {
11     long my_id = (long)id;
12     int i;
13
14     printf("Hello from thread %ld \n", my_id);
15     pthread_mutex_lock(&lock);
16     for (i = 0; i < ROUNDS; ++i) {
17         global_i++;
18     }
19     pthread_mutex_unlock(&lock);
20     return NULL;
21 }
```



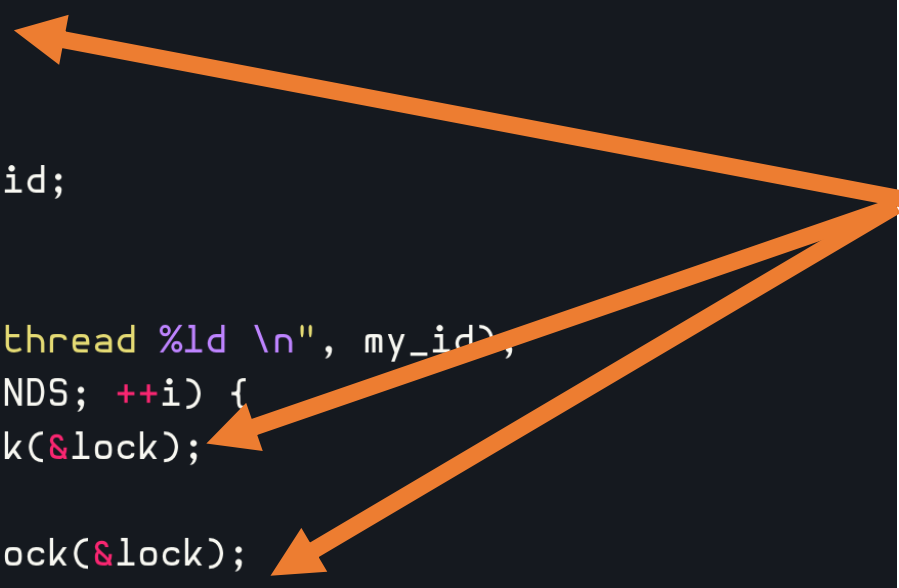
Three orange arrows originate from the right side of the slide and point towards the code. The top arrow points to the declaration of `pthread_mutex_t lock;` on line 8. The middle arrow points to the `pthread_mutex_lock(&lock);` call on line 15. The bottom arrow points to the `pthread_mutex_unlock(&lock);` call on line 19.

Onde colocar o lock?

Exemplos com threads – versão 2

- Dados compartilhados:

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #define ROUNDS 10000
6
7  static int global_i;
8  pthread_mutex_t lock;
9
10 void *inc(void *id) {
11     long my_id = (long)id;
12     int i;
13
14     printf("Hello from thread %ld \n", my_id);
15     for (i = 0; i < ROUNDS; ++i) {
16         pthread_mutex_lock(&lock);
17         global_i++;
18         pthread_mutex_unlock(&lock);
19     }
20     return NULL;
21 }
22
```



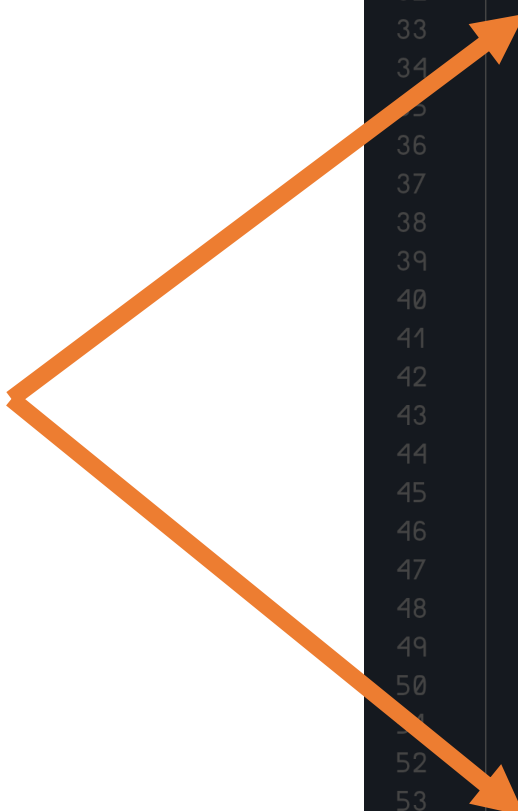
The diagram shows three orange arrows originating from a common point on the right side of the slide. One arrow points to the declaration of the lock variable on line 8: `pthread_mutex_t lock;`. The other two arrows point to the lock acquisition and release calls within the `inc` function: `pthread_mutex_lock(&lock);` on line 16 and `pthread_mutex_unlock(&lock);` on line 18.

Onde colocar o lock?

Exemplos com threads – versão 2

- Dados compartilhados:

```
23 int main(int argc, char *argv[]) {
24     long thread, thread_count;
25     pthread_t *thread_handles;
26
27     if (argc < 2) {
28         printf("usage: %s <number of threads>", argv[0]);
29         return 1;
30     }
31     // initialize the mutex
32     if (pthread_mutex_init(&lock, NULL) != 0) {
33         printf("mutex init failed\n");
34         return 1;
35     }
36
37     global_i = 0;
38     thread_count = strtoul(argv[1], NULL, 10);
39     thread_handles = malloc(thread_count * sizeof(pthread_t));
40
41     // start threads
42     for (thread = 0; thread < thread_count; thread++)
43         pthread_create(&thread_handles[thread], NULL, inc, (void *)thread);
44
45     // wait for threads
46     for (thread = 0; thread < thread_count; thread++)
47         pthread_join(thread_handles[thread], NULL);
48
49     printf("\n\nFinal value of 'global_i' is %d\n", global_i);
50
51     // destroy mutex and thread handles
52     free(thread_handles);
53     pthread_mutex_destroy(&lock);
54
55     return 0;
56 }
```



Exemplos com threads – versão 2

- Dados compartilhados:
 - Execução

```
material/coding/01-threads via C v13.1.6-clang → make 02-inc
cc      02-inc.c      -o 02-inc
material/coding/01-threads via C v13.1.6-clang → ./02-inc 10
Hello from thread 0
Hello from thread 4
Hello from thread 1
Hello from thread 3
Hello from thread 7
Hello from thread 5
Hello from thread 6
Hello from thread 2
Hello from thread 8
Hello from thread 9

Final value of 'global_ithread' is 100000
```



Exclusão Mútua

- Garante acesso exclusivo à uma **região crítica**
- Em um sistema distribuído
 - Como controlar o acesso de múltiplos processos a um recurso compartilhado, garantindo que cada processo controla **exclusivamente** aquele recurso durante seu acesso?
 - Mutex? IPC?

Exclusão Mútua

- Propriedades:

1. **Exclusão mútua:**

- Somente um processo pode estar na **região crítica** em qualquer instante de tempo

2. **Ausência de deadlocks:**

- Se processos estão tentando acessar o recurso, então **algum processo deve conseguir acesso** em algum instante, dado que nenhum processo fique na região crítica indefinidamente

3. **Não-inanição:**

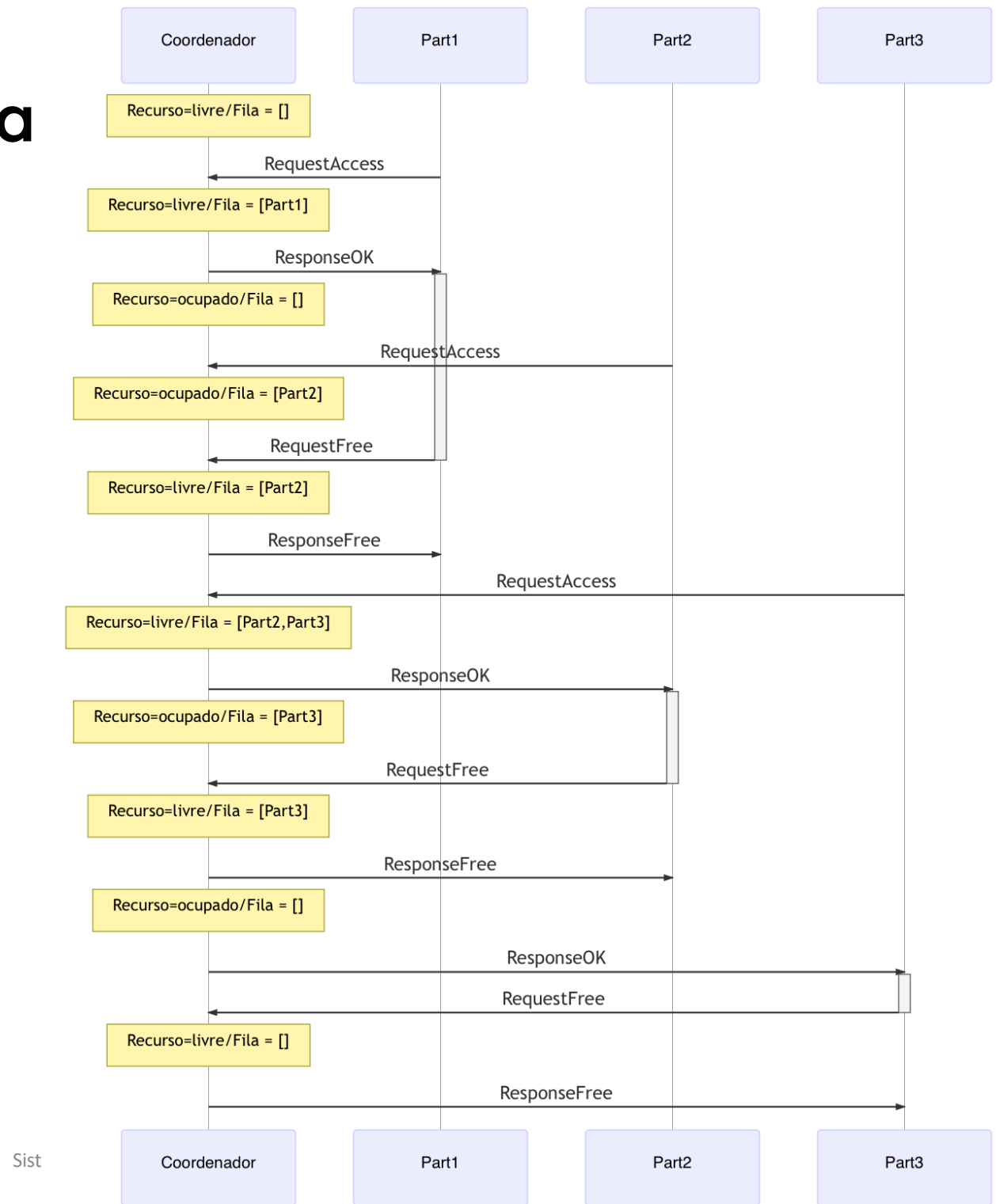
- Todos os processos interessados conseguem, em algum momento, acessar o recurso

4. **Espera limitada:**

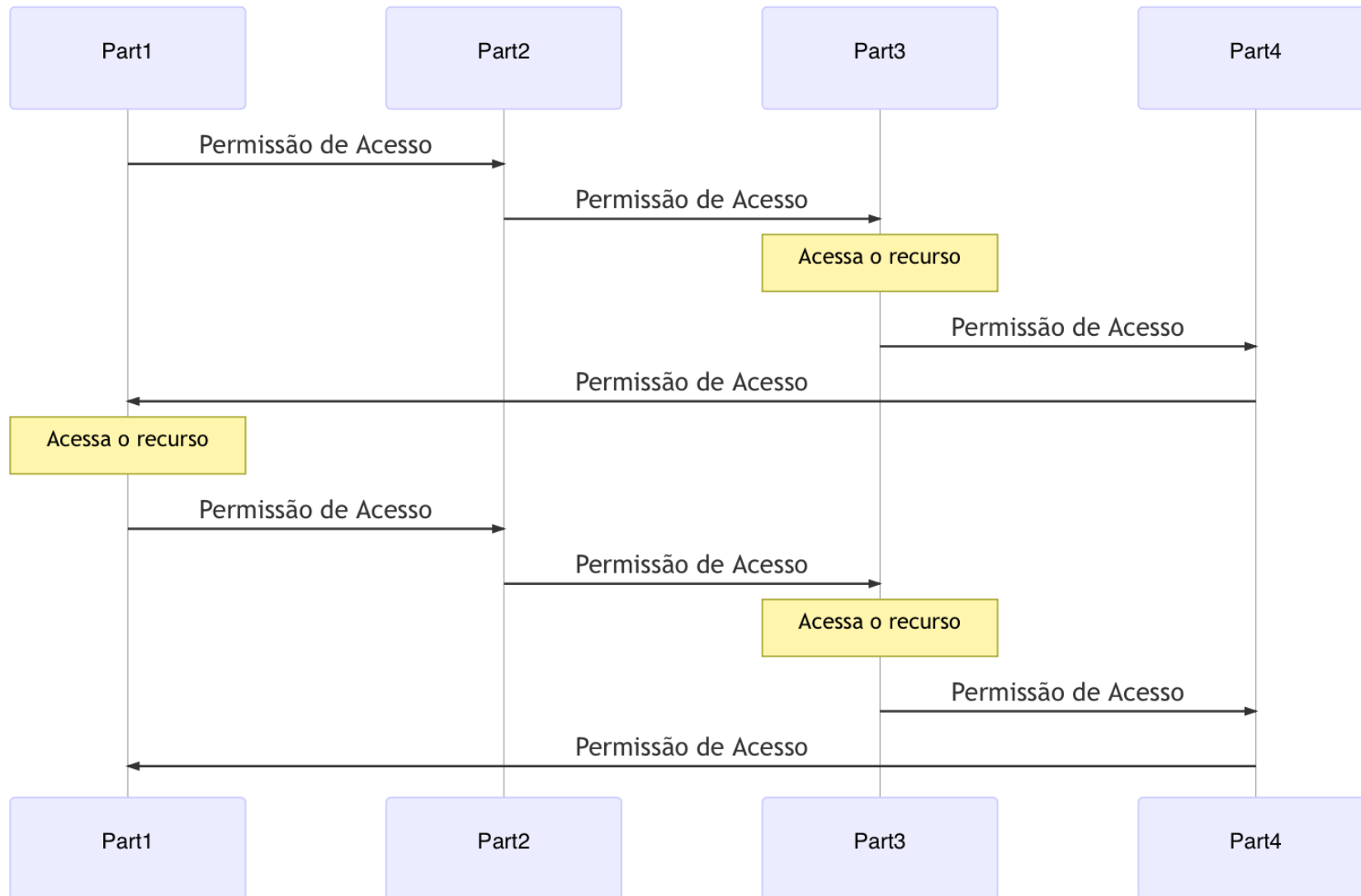
- O tempo de espera pelo recurso é limitado.

Exclusão Mútua

- Coordenador



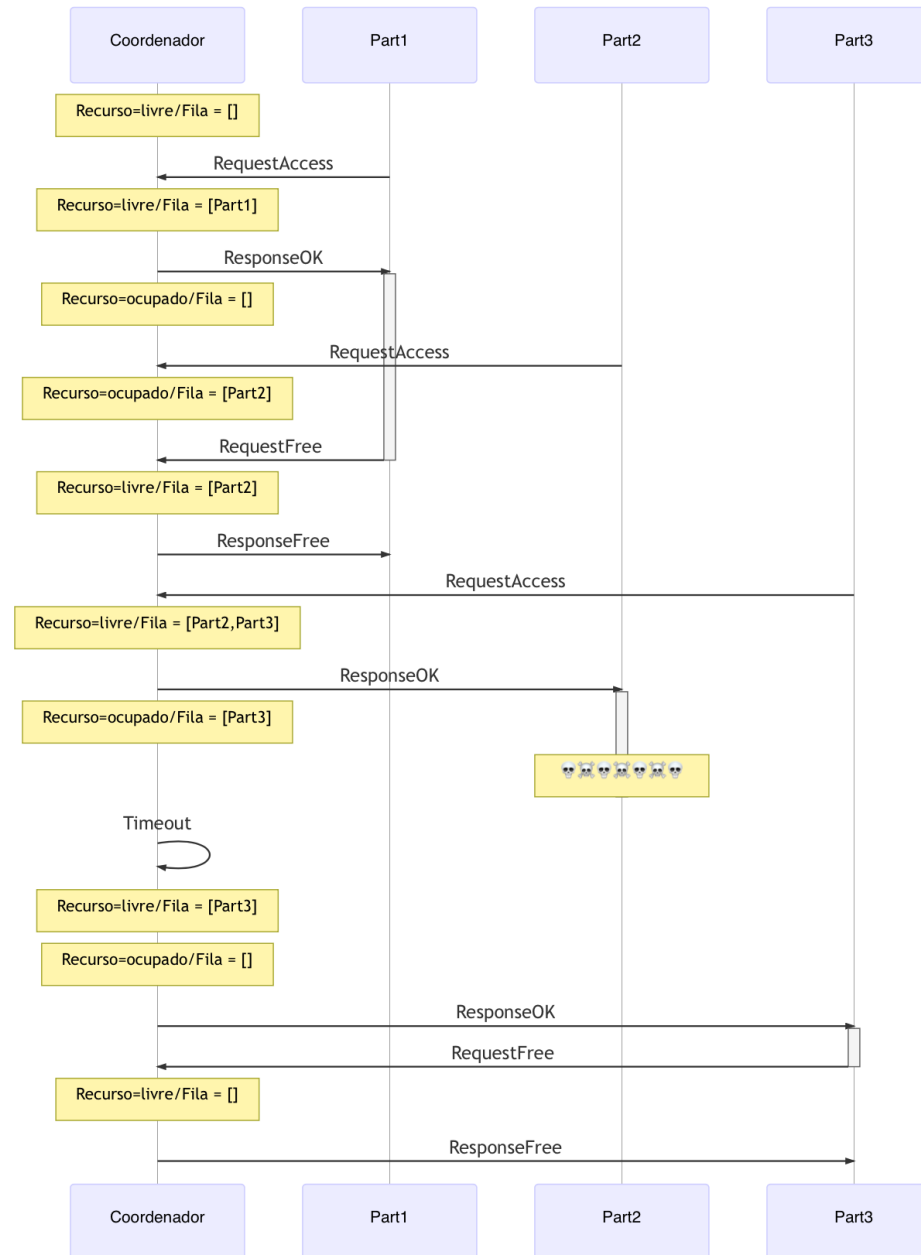
Exclusão Mútua: anel



Exclusão Mútua: lidando com falhas

- Em ambos os algoritmos, centralizado e do anel, se um processo falhar, o algoritmo pode ficar "travado":
 - No algoritmo centralizado, se o coordenador falha antes de liberar o acesso para algum processo, ele leva consigo a permissão.
 - Em ambos os algoritmos, se o processo acessando o recurso falha, a permissão é perdida e os demais processos sofrerão inanição.
 - No algoritmo do anel, se qualquer outro processo falha, o anel é interrompido o anel não conseguirá circular.

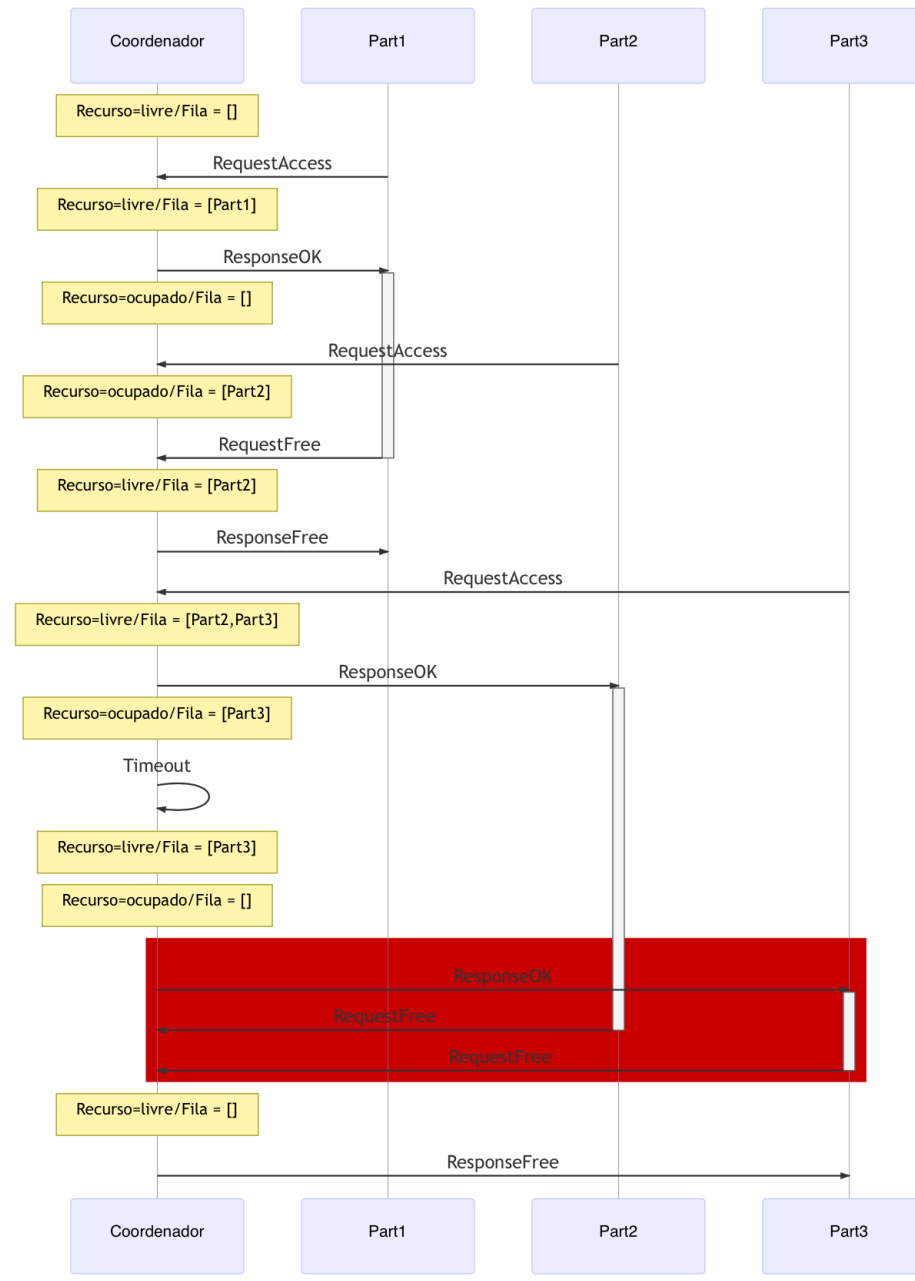
Exclusão Mútua: lidando com falhas - timeout



Exclusão Mútua: lidando com falhas - timeout

- Problema:
 - assumir que o processo parou de funcionar
- Caso isso não seja verdade:
 - Pode-se ter duas autorizações ao mesmo tempo no sistema
 - Possibilidade de violação da propriedade de exclusão mútua.

Exclusão Mútua: lidando com falhas - timeout



Exclusão Mútua: lidando com falhas - timeout

Impossibilidade de detecção de falhas:

Em um sistema distribuído **assíncrono**, é impossível distinguir um processo falho de um processo lento.

Exclusão Mútua: lidando com falhas - timeout

- Qual deve ser um *timeout* **razoável** para o meu sistema?
 - Resposta depende de mais perguntas, como:
 - Qual o custo E de esperar por mais tempo?
 - Qual o custo C de cometer um engano?
 - Qual a probabilidade p de cometer um engano?
- O custo esperado por causa dos erros, isto é, a esperança matemática da variável aleatória custo, é menor que o custo de se esperar por mais tempo, isto é, $C \cdot p < E$?
- Pode-se partir para a análise de algoritmos probabilísticos:

“Se o mundo é probabilístico, porquê meus algoritmos devem ser determinísticos?” - Werner Vogels

Exclusão Mútua: lidando com falhas - quóruns

- **Quórum:**

- Número de pessoas imprescindível para a realização de algo
- **Algo** = liberação de acesso ao recurso

- Abordagem semelhante à coordenada:

- Papel do **coordenador** ainda existe
- No entanto:
 - Participante precisa obter **m** autorizações antes de acessar o recurso
 - **m** é o quórum do sistema

- **Quórum**

- **n** coordenadores
- Participante precisa da autorização de pelo menos **m** coordenadores
- Qual o valor adequado para **m**?

Exclusão Mútua: lidando com falhas - quóruns

- **Quórum**

- n coordenadores
- $m > n/2$ coordenadores

- **Algoritmo: Coordenador**

- Inicializa recurso como livre
- Ao receber uma requisição, a enfileira
- Ao receber uma liberação
 - se do processo a quem autorizou, marca o recurso como livre
 - senão e se de um processo na fila, remove o processo da fila
 - senão, ignore mensagem
- Sempre que recurso estiver marcado como livre **E** a fila não estiver vazia
 - remove primeiro processo da fila
 - envia liberação para processo removido
 - marca o recurso como ocupado

Exclusão Mútua: lidando com falhas - quóruns

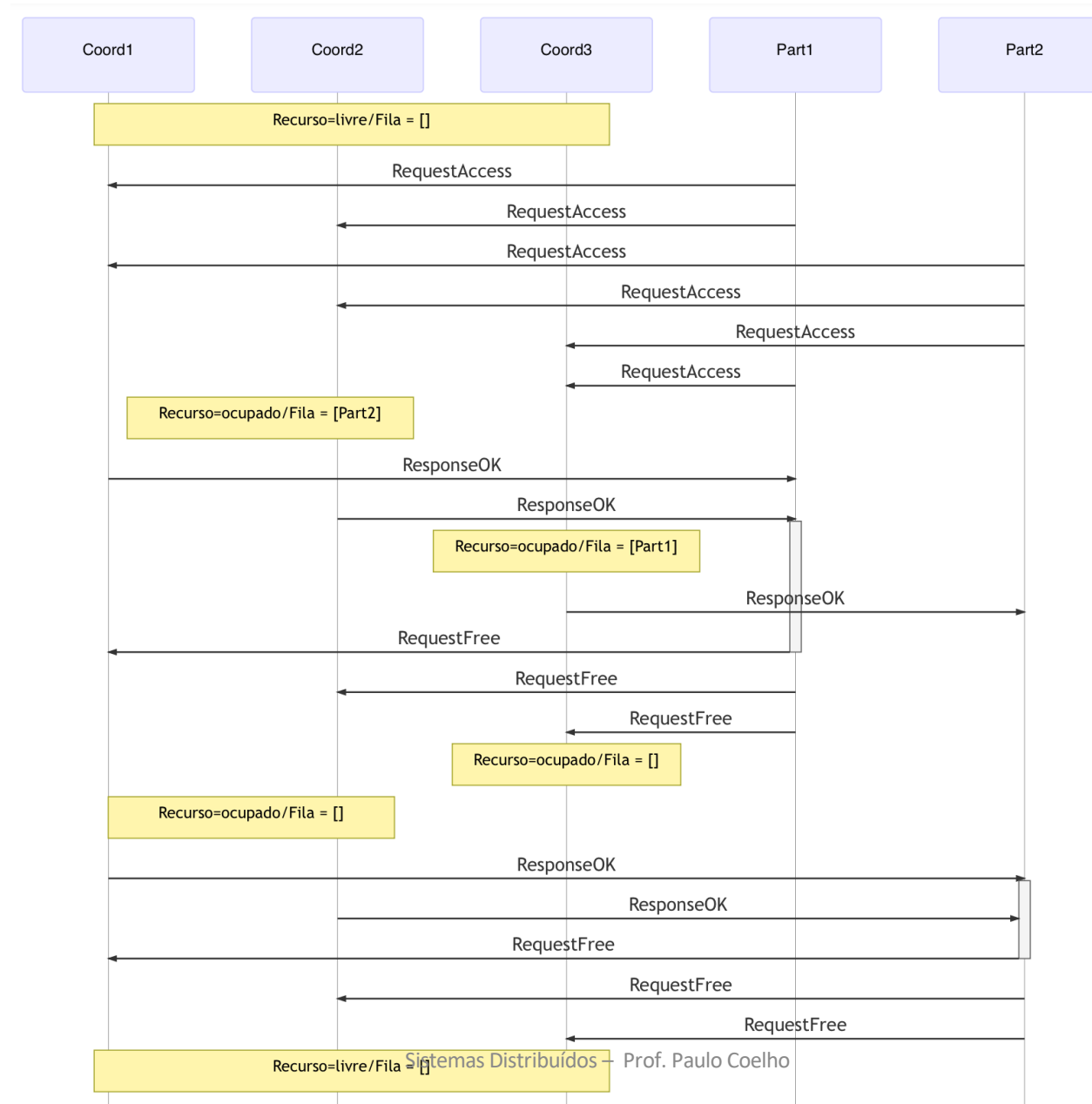
- **Quórum**

- n coordenadores
- $m > n/2$ coordenadores

- **Algoritmo: Participante**

- Envia requisição de acesso aos n coordenadores
- Espera por resposta de m coordenadores
- Acessa o recurso
- Envia liberação do recurso para os n coordenadores

Exclusão Mútua: lidando com falhas - quóruns

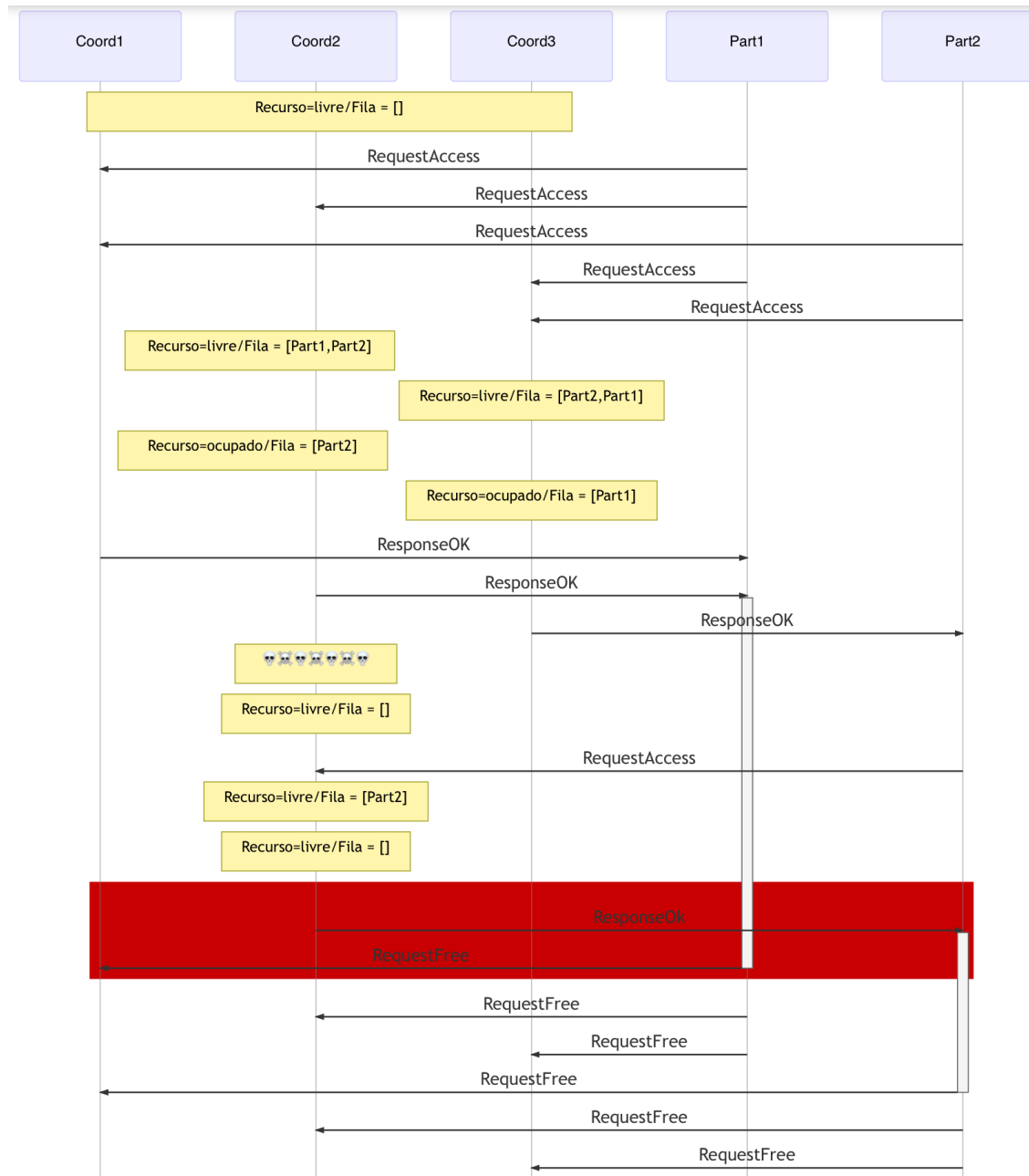


$n=3,$
 $m=2$

Exclusão Mútua: lidando com falhas - quóruns

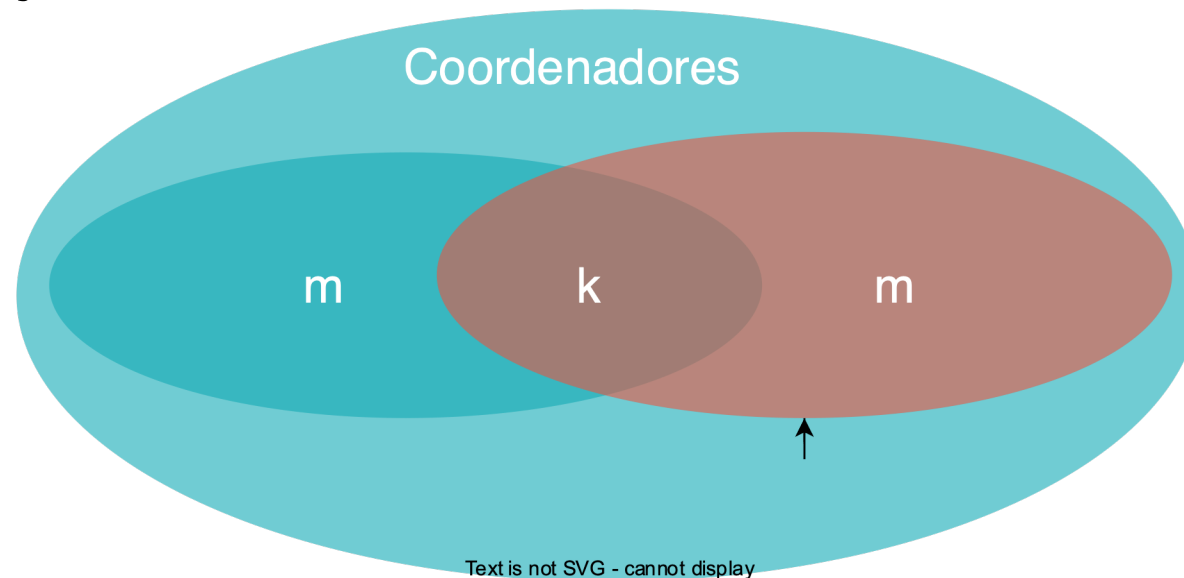
- Deixando o problema mais interessante:
 - Autorizações armazenadas somente em memória
 - Coordenadores, ao falhar e então resumir suas atividades, esquecem das autorizações já atribuídas
- **Perda de memória:**
 - Quando um coordenador falha, esquece que deu ok e reinicia seu estado.
- Este algoritmo é bom?

Exclusão Mútua: lidando com falhas - quóruns


$$n=3,$$
$$m=2$$

Exclusão Mútua: lidando com falhas - quóruns

- Exclusão Mútua é violada
- Qual a probabilidade de isso acontecer?
 - Dados dois quóruns, de tamanho m , que se sobrepõem em k processos, qual a probabilidade P_v de que os k processos na interseção sejam reiniciados e levem à violação?



Exclusão Mútua: lidando com falhas - quóruns

- Seja a **P** a probabilidade de **um coordenador em específico falhar** e se recuperar dentro de uma janela de tempo **δt**
- Temos:
 - Probabilidade de falha de **exatamente 1** coordenador: $P^1(1-P)^{n-1}$
 - Probabilidade de **k coordenadores** falharem: $P^k(1-P)^{n-k}$
 - Probabilidade de quaisquer k em m coordenadores falharem: $\binom{m}{k}P^k(1-P)^{m-k}$
- Mas qual é o tamanho k da interseção?
 - $|A \cup B| = |A| + |B| - |A \cap B| \Rightarrow n = m + m - k$
 - $|A \cap B| = |A| + |B| - |A \cup B| \Rightarrow k = m + m - n = 2m - n$

Exclusão Mútua: lidando com falhas - quórums

- Qualquer falha de **k** ou mais coordenadores é um problema:
 - Probabilidade de quaisquer **k** em **m** coordenadores falharem:

$$P_v = \sum_{k=2m-n}^n \binom{m}{k} P^k (1 - P)^{m-k}$$

- Considere o exemplo:
 - $P = 0.0001$ (1 minuto a cada 10 dias)
 - $n = 32$
 - $m = 0.75n$ (i.e., 24)
 - $P_v < 10^{-40}$

Exclusão Mútua: lidando com falhas - quóruns

- Como ficam as propriedades originais definidas?
 - Exclusão Mútua
 - Ausência de deadlock
 - Não-inanição
 - Espera limitada

Exclusão Mútua: lidando com falhas - quóruns

- Como ficam as propriedades originais definidas?
 - Exclusão Mútua probabilística: $1 - P_v$
 - Ausência de deadlock
 - Não-inanição
 - Espera limitada

Exclusão Mútua: lidando com falhas - quóruns

- Como ficam as propriedades originais definidas?
 - Exclusão Mútua probabilística
 - Ausência de deadlock e
 - Não-inanição:
 - E se cada participante obtiver o ok de um coordenador?
 - Temporizador?
 - Espera limitada

Exclusão Mútua: lidando com falhas - quóruns

- Como ficam as propriedades originais definidas?
 - Exclusão Mútua probabilística
 - Ausência de deadlock
 - Não-inanição
 - Espera limitada:
 - Aborts podem levar a espera infinita

Exclusão Mútua: lidando com falhas - quóruns

- Algoritmo também pode não ser adequado para certas situações
- Uso de um líder para coordenar ações em um SD simplifica o projeto:
 - Mas pode se tornar um ponto único de falha
- E se substituíssemos o coordenador no caso de falhas?
 - Este é o problema conhecido como **eleição de líderes**

Eleição de líder

- Informalmente
 - Procedimento pelo qual **um processo é escolhido** dentre os demais processos E
 - **o processo escolhido é ciente da escolha E**
 - **todos os demais processos o identificam como eleito**
- Uma **nova eleição** deve acontecer sempre que o líder se tornar **indisponível**

Eleição de líder

- Formalmente, um algoritmo de eleição de líderes deve satisfazer as seguintes condições.
 - **Terminação:** algum processo deve se considerar líder em algum momento
 - **Unicidade:** somente um processo se considera líder
 - **Acordo:** todos os outros processos sabem quem foi eleito líder

Eleição de líder

- Exemplo: escolher um líder na turma da disciplina de Sistemas Distribuídos
 - **Candidatos:** todos os membros são elegíveis ou apenas um subconjunto dos mesmos?
 - **Comunicação:** todos se conhecem e se falam diretamente ou há grupos incomunicáveis dentro da turma?
 - **Estabilidade:** de que adianta eleger um dos colegas se frequentemente não está presente quando necessário?
- Em termos computacionais:
 - Processo são **diferentes:** memória, conectividade, ...
- Vamos ignorar estas diferenças por enquanto...

Eleição de líder – Algoritmo do anel

- Processos organizados em um anel lógico
- Troca de mensagens apenas com processos à "esquerda" e à "direita"
- Todos os processos são exatamente idênticos
- Algoritmo de eleição neste anel:
 - processo inicialmente seguidor se torna Candidato,
 - então se declara Eleito,
 - avisa a seus pares e, finalmente,
 - se declara Empossado

Eleição de líder – Algoritmo do anel

Algoritmo do Anel 1

- Na iniciação
 - Organize os nós em um anel lógico
 - $C \leftarrow$ Seguidor
- Quando um processo acha que o líder está morto
 - $C \leftarrow$ Candidato
 - Envia (VoteEmMim) para "a direita" no anel.
- Quando um processo recebe (VoteEmMim)
 - Se $C =$ Seguidor
 - envia (VoteEmMim) para a direita
 - Se $C =$ Candidato
 - $C \leftarrow$ Eleito
 - envia (HabemosLeader) para a direita
- Quando um processo recebe (HabemosLeader)
 - Se $C =$ Seguidor
 - envia (HabemosLeader) para a direita
 - Se $C =$ Eleito
 - $C \leftarrow$ Empossado

Eleição de líder – Algoritmo do anel

- Exemplo de execução 1:
 - Ok!



1	2
$C \leftarrow \text{Seguidor}$	$C \leftarrow \text{Seguidor}$
$C \leftarrow \text{Candidato}$	
$(\text{VoteEmMim}) \rightarrow$	
	$(\text{VoteEmMim}) \leftarrow$
	$(\text{VoteEmMim}) \rightarrow$
$(\text{VoteEmMim}) \leftarrow$	
$C \leftarrow \text{Eleito}$	
$(\text{HabemosLider}) \rightarrow$	
	$(\text{HabemosLider}) \leftarrow$
	$(\text{HabemosLider}) \rightarrow$
$(\text{HabemosLider}) \leftarrow$	
$C \leftarrow \text{Empossado}$	

Eleição de líder – Algoritmo do anel

- Exemplo de execução 2:
 - Problema!



- Qual?
- Como resolver?

1	2
$C \leftarrow \text{Seguidor}$	$C \leftarrow \text{Seguidor}$
$C \leftarrow \text{Candidato}$	$C \leftarrow \text{Candidato}$
$(\text{VoteEmMim}) \rightarrow$	$(\text{VoteEmMim}) \rightarrow$
$(\text{VoteEmMim}) \leftarrow$	$(\text{VoteEmMim}) \leftarrow$
$C \leftarrow \text{Eleito}$	$C \leftarrow \text{Eleito}$
$(\text{HabemosLider}) \rightarrow$	$(\text{HabemosLider}) \rightarrow$
$(\text{HabemosLider}) \leftarrow$	$(\text{HabemosLider}) \leftarrow$
$C \leftarrow \text{Empossado}$	$C \leftarrow \text{Empossado}$

Eleição de líder – Algoritmo do anel

- Identificadores de processo
 - Única máquina: PID (*process id*)
 - Definido pelo SO, tem sentido apenas na própria máquina
 - Válido enquanto o processo estiver executando
 - Pode ser reciclado
 - Reiniciado com o computador
 - Sistema distribuído:
 - Se processo executa em um mesmo *host*:
 - identificador do *host* (e.g., endereço IP)
 - Se mais de um processo executa no mesmo *host*
 - Pode ser apenas um **parâmetro** passado na inicialização
 - Ou a combinação **IP/porta**.
- Vamos assumir que exista um mecanismo de identificação de processos

Eleição de líder – Algoritmo do anel v2

Algoritmo do Anel 2

- Organize os nós em um anel lógico
- Quando p acha que o líder está morto:
 - Envia mensagem $[p]$ "à direita".
- Quando p recebe l
 - Se $p \notin l$
 - Envia $[p:l]$ para a direita.
 - Se $p \in l$
 - Escolhe menor id em l como líder.

- Pior caso para total de mensagens?
 - n processos suspeitam ao mesmo tempo
 - n^2 mensagens

Eleição de líder – Algoritmo do anel v3



Algoritmo de Chang e Robert

- Organize os nós em um anel lógico
- Quando p acha que o líder está morto:
 - Envia mensagem (p) à direita
- Quando p recebe (q)
 - Se $p = q$
 - p se declara líder
 - Senão e se $q > p$
 - Envia (q) para a direita.

- Melhor caso para total de mensagens?
- Pior caso para total de mensagens?
- Como outros ficam sabendo quem se declarou líder?

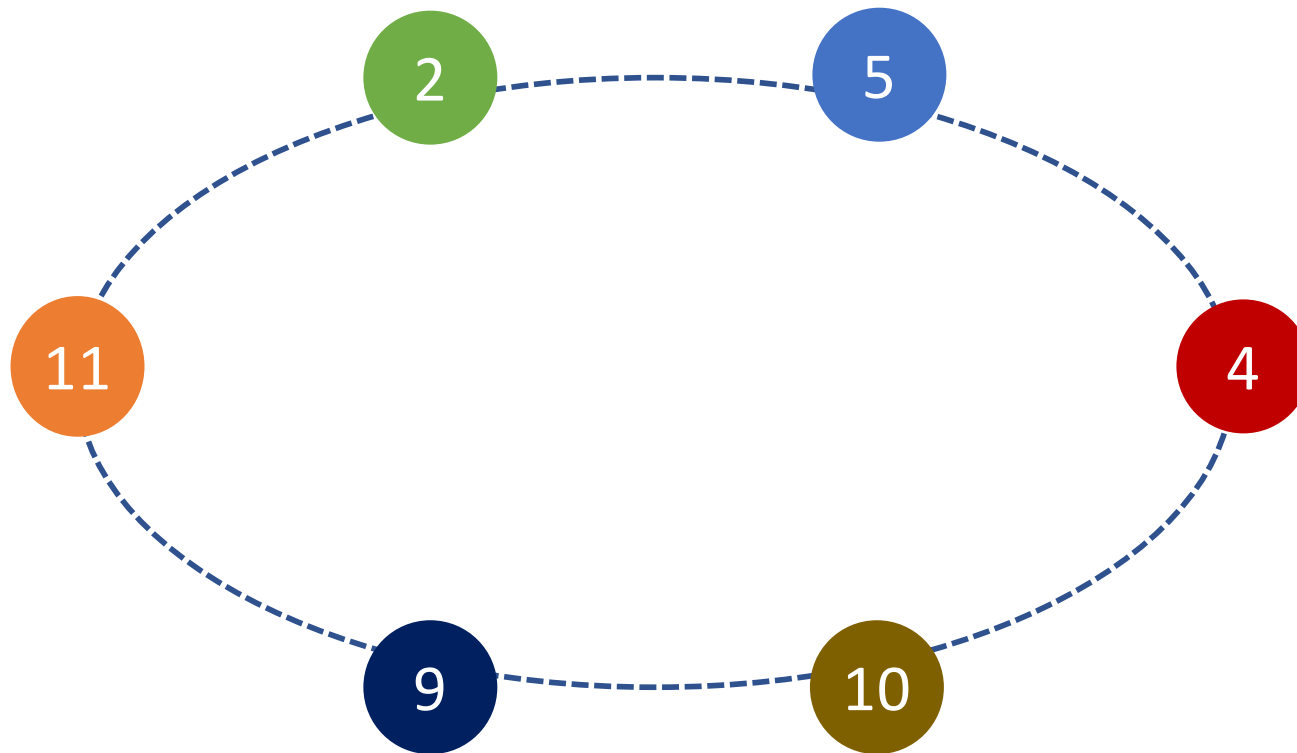
Eleição de líder – Algoritmo do anel v4

Algoritmo de Franklin

- Organize os nós em um anel lógico
- $\text{Ativo} \leftarrow 1$
- Quando p acha que o líder está morto e se $\text{Ativo} = 1$:
 - Envia mensagem (p) à direita e à esquerda
- Quando p recebe e e d , da esquerda e da direita, respectivamente:
 - Se $\text{Ativo} = 1$
 - Se $\max(e, d) < p$
 - Envia mensagem (p) à direita e à esquerda
 - Se $\max(e, d) > p$
 - $\text{Ativo} \leftarrow 0$
 - Envia mensagem $-p$ à direita e à esquerda
 - Se $\max(e, d) = p$
 - p se declara líder.
 - Se $\text{Ativo} = 0$
 - Repassa cada mensagem para o outro lado.

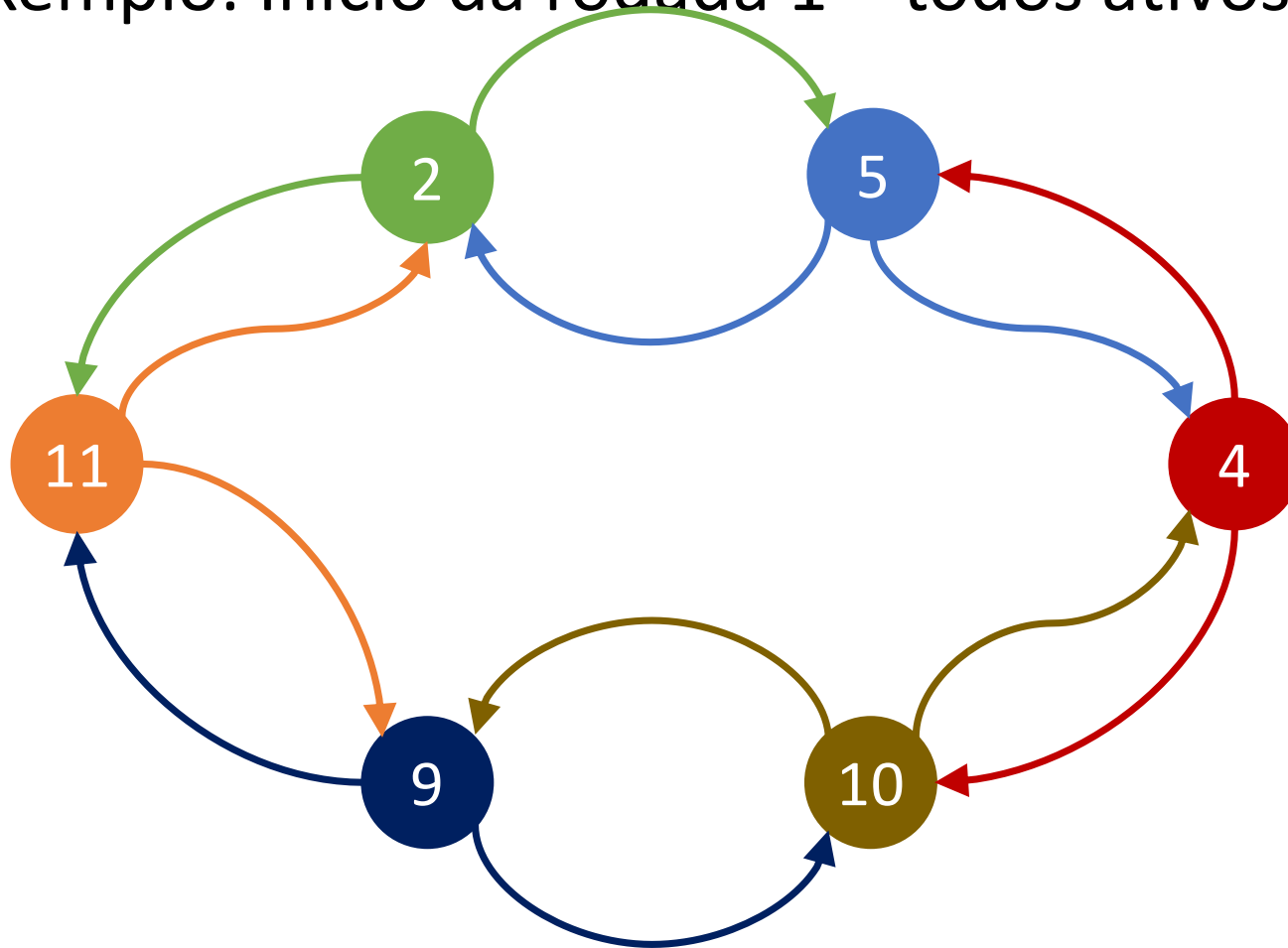
Eleição de líder – Algoritmo do anel v4

- Exemplo: 6 processos



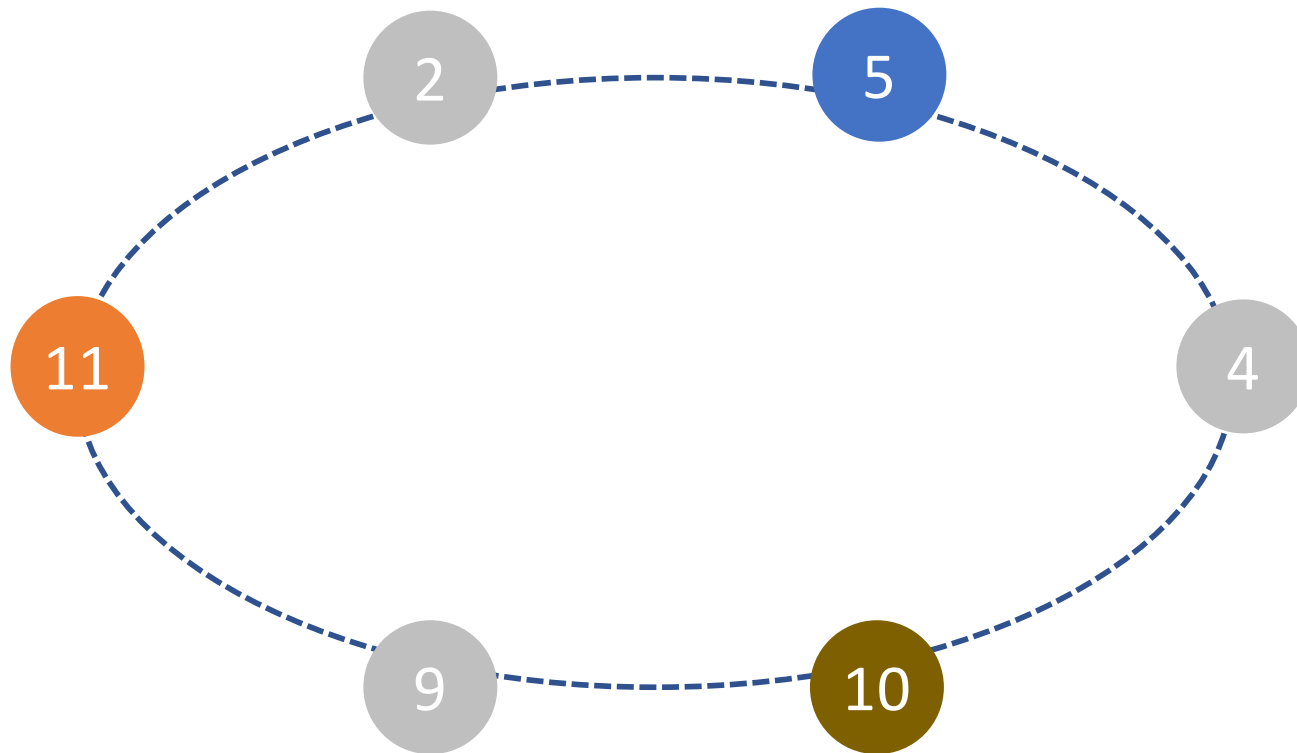
Eleição de líder – Algoritmo do anel v4

- Exemplo: Início da rodada 1 – todos ativos



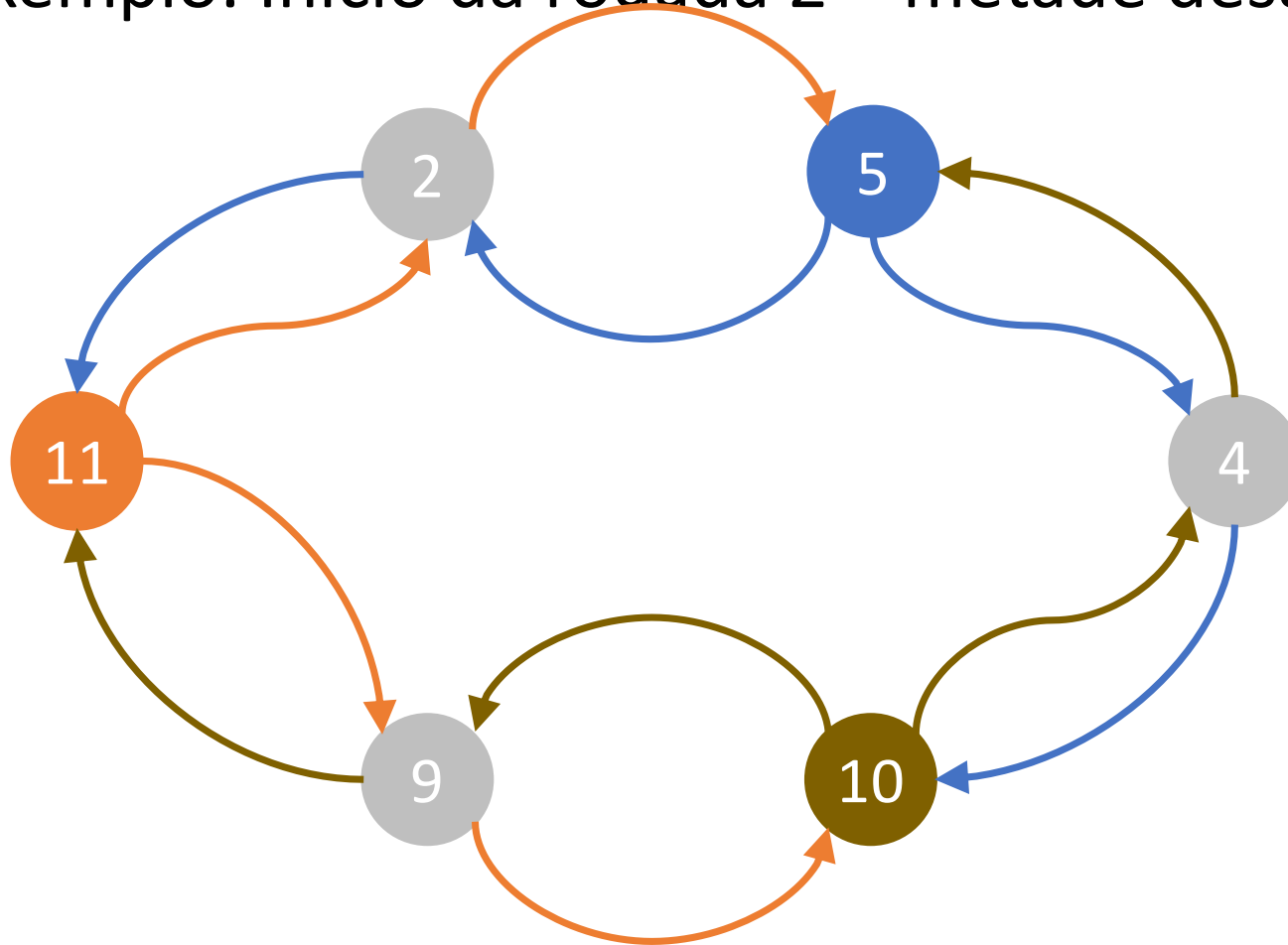
Eleição de líder – Algoritmo do anel v4

- Exemplo: Fim da rodada 1 – metade desativada



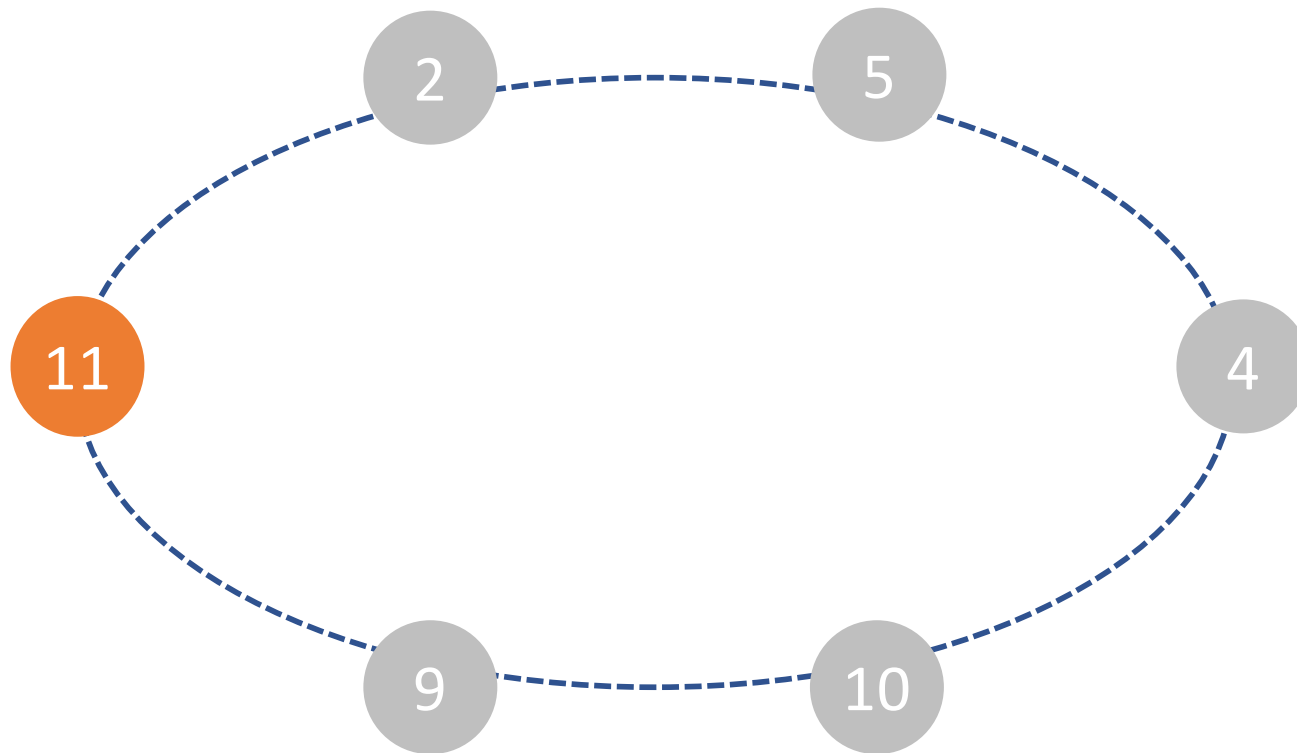
Eleição de líder – Algoritmo do anel v4

- Exemplo: Início da rodada 2 – metade desativada



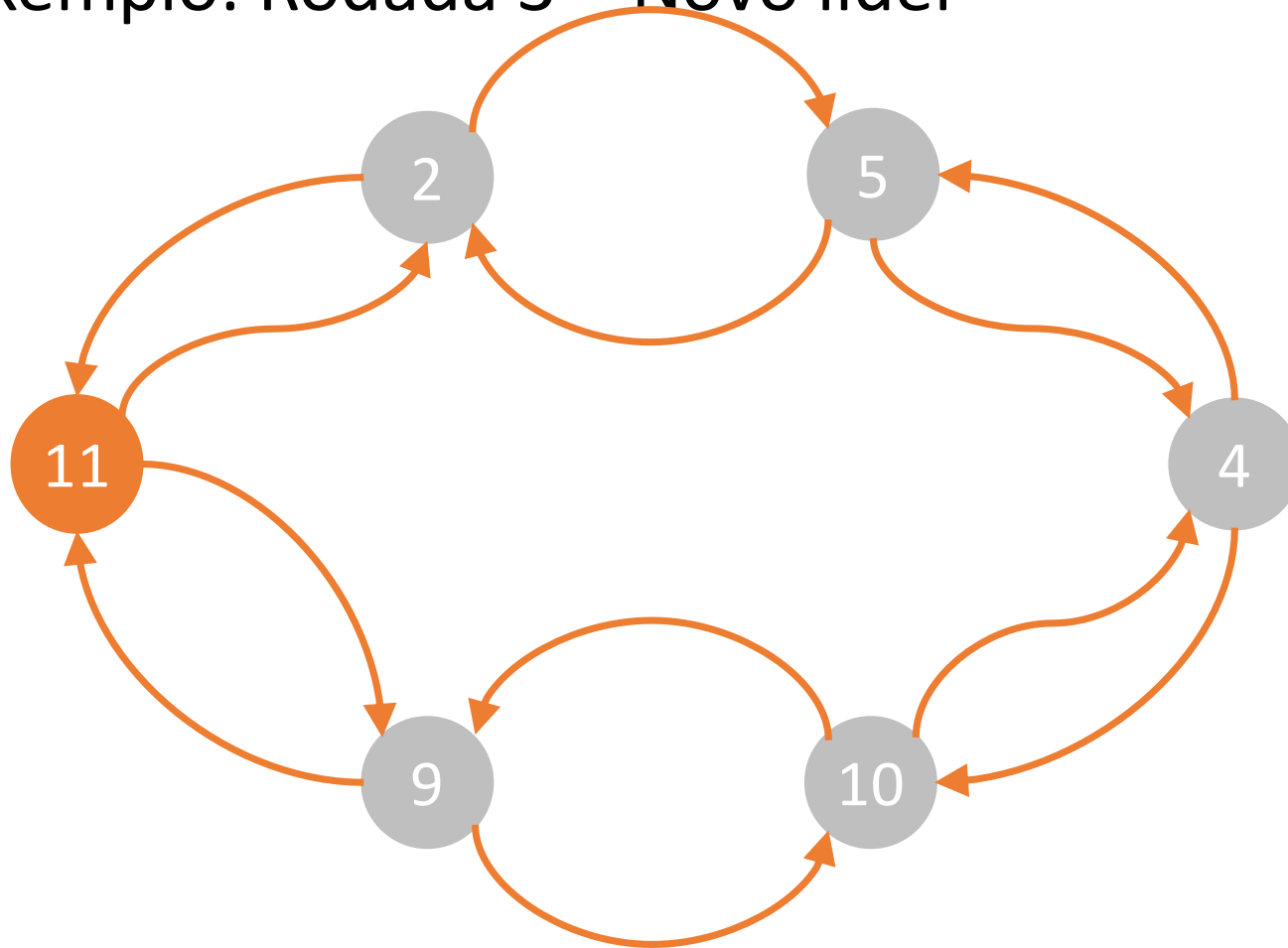
Eleição de líder – Algoritmo do anel v4

- Exemplo: Fim da rodada 2



Eleição de líder – Algoritmo do anel v4

- Exemplo: Rodada 3 – Novo líder



Eleição de líder – Algoritmo do anel v5

- Algoritmo do “brigão” (Bully)
- Alguma **característica comparável** dos processos é escolhida
- Processo com o valor de tal característica mais vantajoso é escolhido
- Para simplificar:
 - Usaremos o id
- Os maiores processos (“brigões”) eliminam os processos menores da competição, sempre que uma eleição acontecer

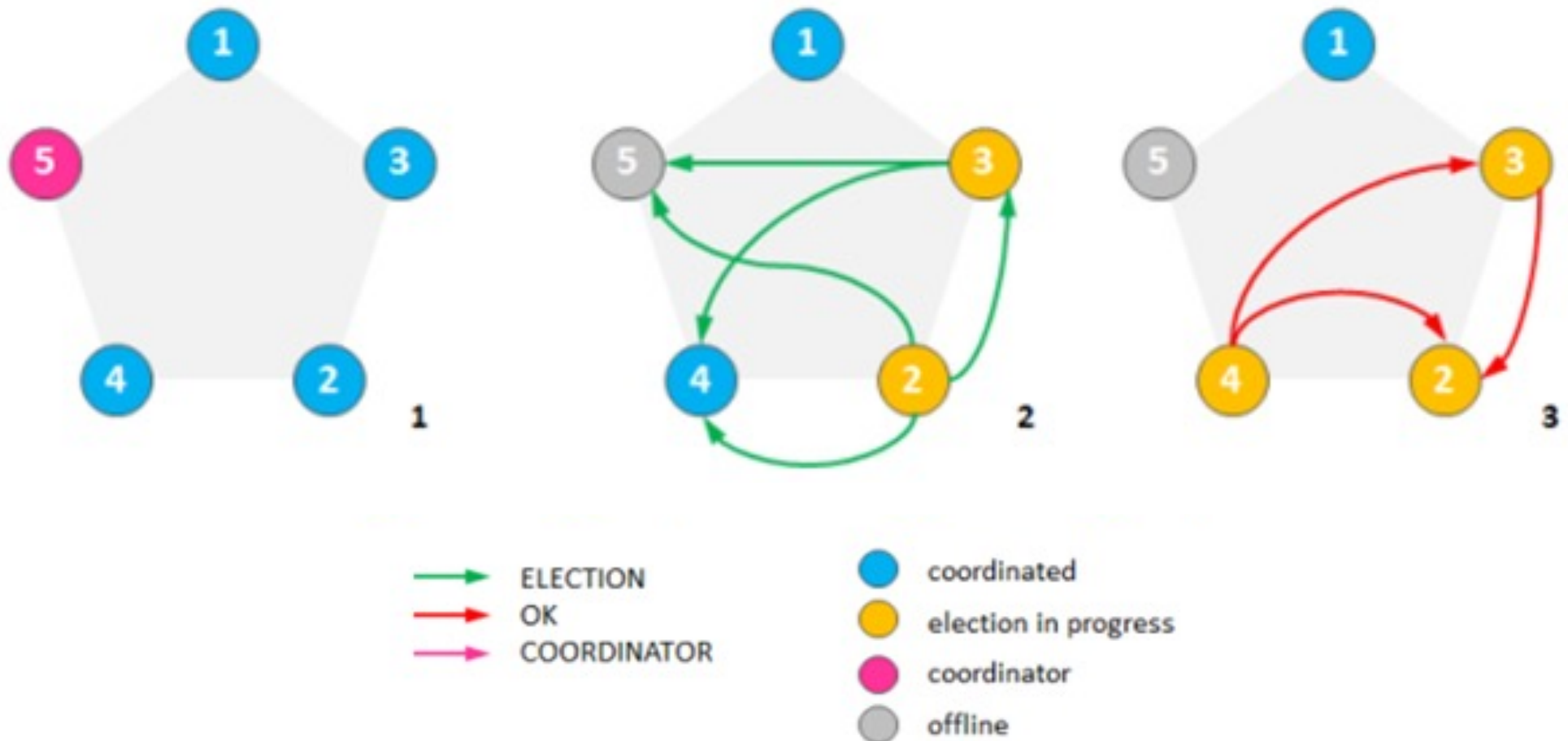
Eleição de líder – Algoritmo do anel v5

Algoritmo do Brigão

- Quando p suspeita que o líder não está presente (muito tempo se receber mensagens do mesmo)
 - p envia mensagem (ELEICA0, p) para todos os processos com identificador maior que p
 - Inicia temporizador de respostas
- Quando temporizador de respostas expira
 - Envia (COORD, p) para todos os processos
- Quando recebe (Ok, p)
 - Para temporizador de resposta
- Quando p recebe (ELEICA0, q), $q < p$
 - Envia (OK, q)
- Quando um processo falho se recupera
 - Inicia uma eleição

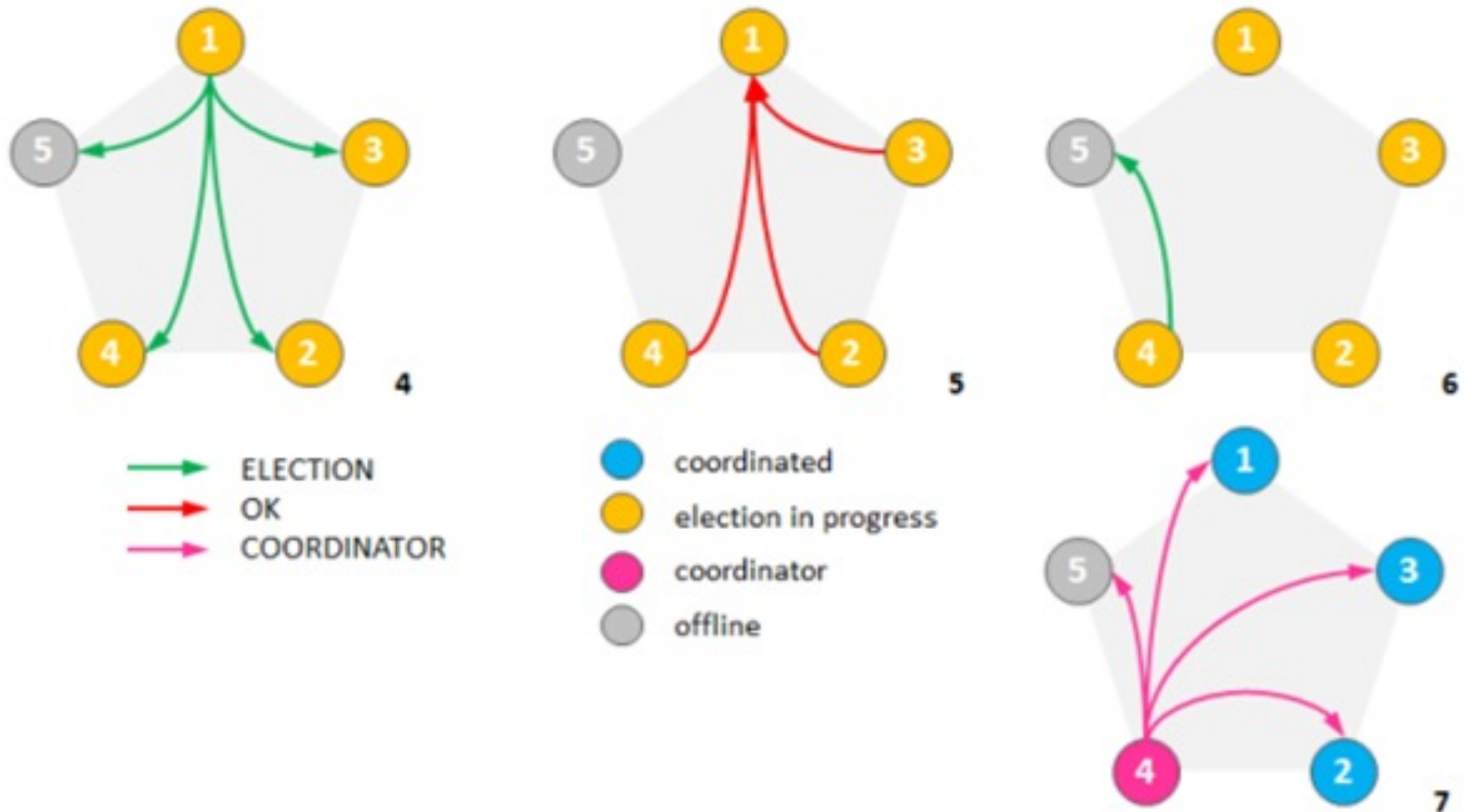
Eleição de líder – Algoritmo do anel v5

- Exemplo:



Eleição de líder – Algoritmo do anel v5

- Exemplo:



Eleição de líder – Algoritmo do anel v5

- A escolha do valor temporizador é fundamental
 - Se o temporizador agressivo:
 - Eleições frequentes desnecessárias
 - Temporizador muito grande:
 - Sistema demorará a eleger um novo líder
 - Tempo muito curto antes de se declarar líder:
 - **Mais de um processo pode se declarar líder** (*split-brain*)
- O algoritmo do brigão não resolve o problema em sistemas totalmente assíncronos