Coordenação e Acordo

Capítulo 15

De **Coulouris, Dollimore, Kindberg and Blair**

Sistemas Distribuídos: Conceitos e Projeto

Edição 5, © Addison-Wesley 2012

Introdução

Este capítulo apresenta um conjunto de algoritmos cujos objetivos variam, mas os quais tem o mesmo objetivo: o de que um conjunto de processos coordene suas ações ou concorde com um ou mais valores. Por exemplo em uma nave espacial, é fundamental que os computadores que a estão controlando concordem com condições como o fato de a missão da nave espacial estar prosseguindo ou ter sido cancelada. Os computadores também precisam coordenar suas ações corretamente, com relação aos recursos compartilhados (os sensores e controladores da nave espacial). Os computadores devem ser capazes de fazer isso mesmo onde não haja nenhum relacionamento mestre-escravo fixo entre os componentes. Outro objetivo importante do capítulo, ao discutirmos os algoritmos, é considerar as falhas e como lidar com elas ao projetar algoritmos.

Um particionamento de rede

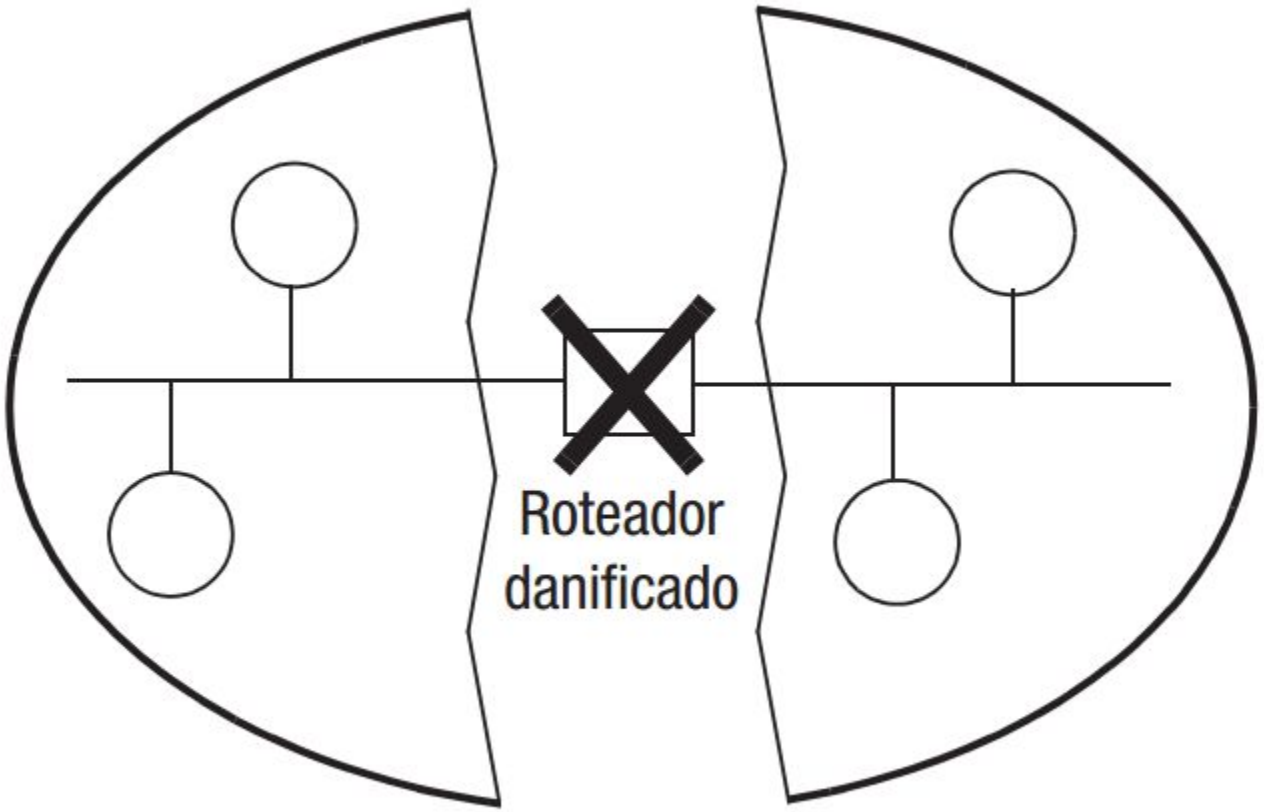


Figura 15.1

Servidor gerenciando uma ficha de exclusão mútua para um conjunto de processos

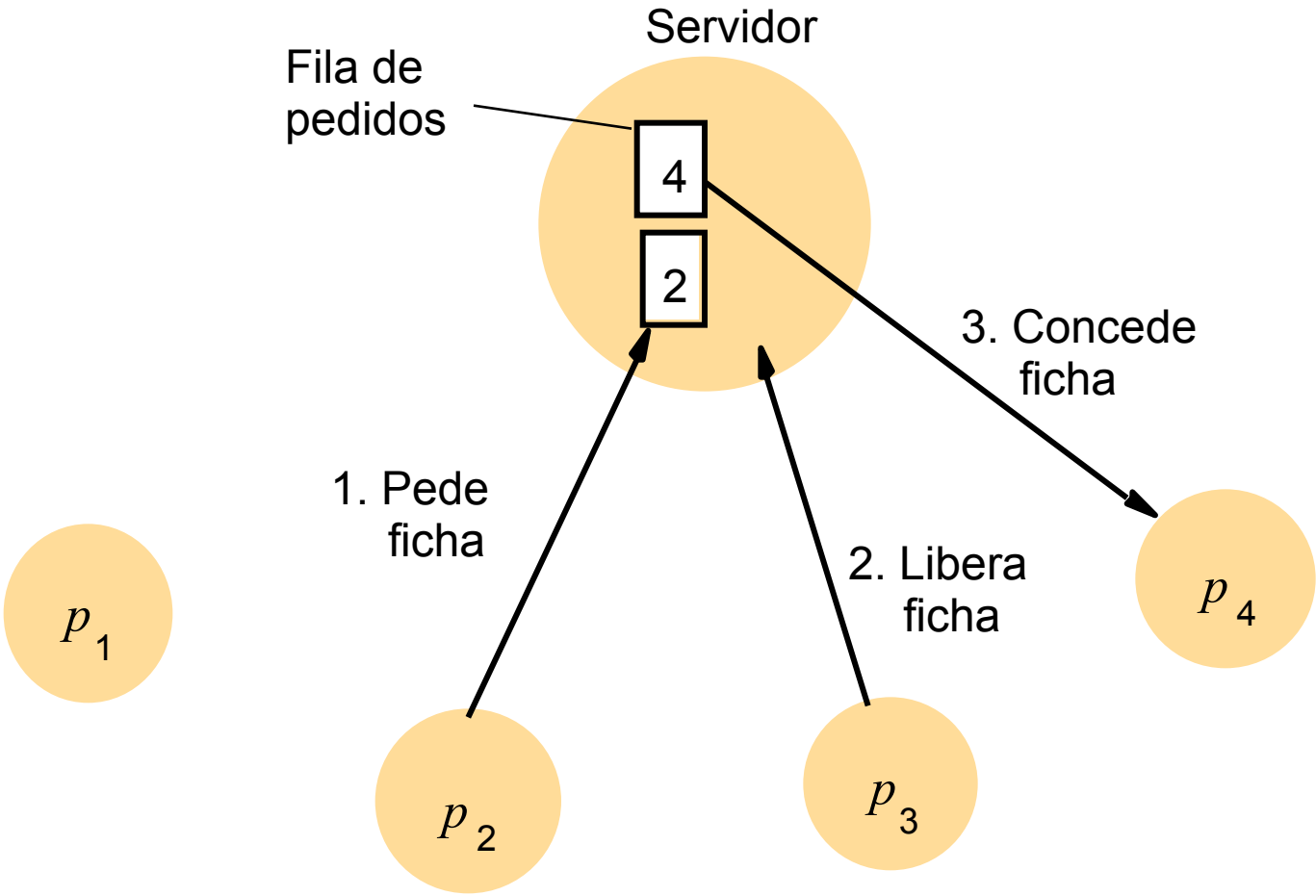


Figura 15.2

Um anel de processos transferindo um *token* de exclusão mútua

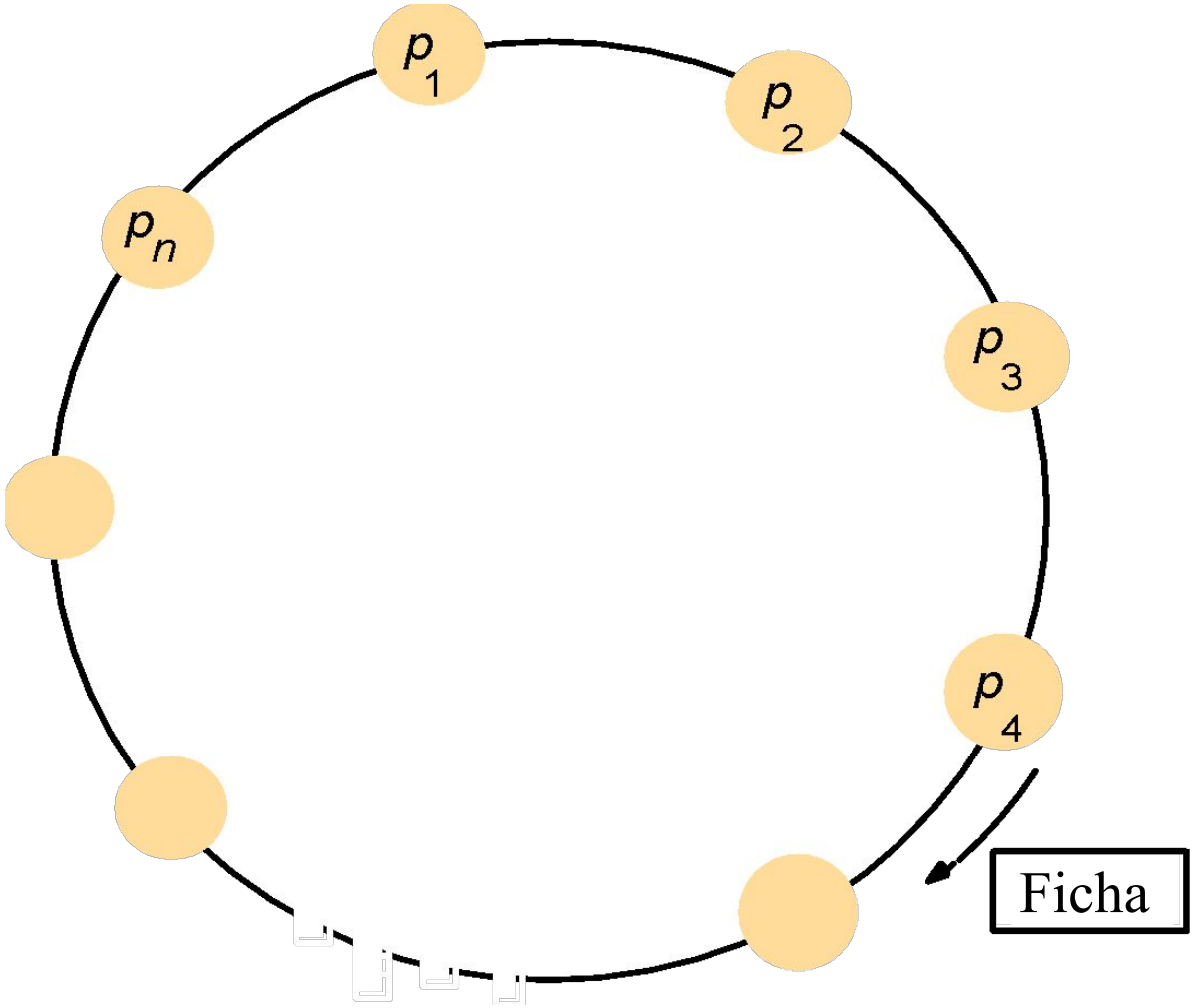


Figura 15.3

Bônus

- 0,5 de bônus na nota da prova 2
- Em duplas, 0,5 para cada um.
- Explicar o algoritmo do próximo slide usando só o seguinte slide e o quadro
- Irei sortear uma dupla
- 30 minutos de preparação

- **Ao final da apresentação farei duas perguntas**
- Cada grupo pode oferecer uma pergunta, se eu usar a pergunta a dupla ganha 0,1 de bônus
- Resposta certa: + 0,1 de bônus
- Resposta errada: - 0,1
- Duas respostas erradas: Farei uma terceira fácil para um dos integrantes, mas se errar a dupla perde todo o bônus e não poderá mais participar

Sincronização por *multicast*

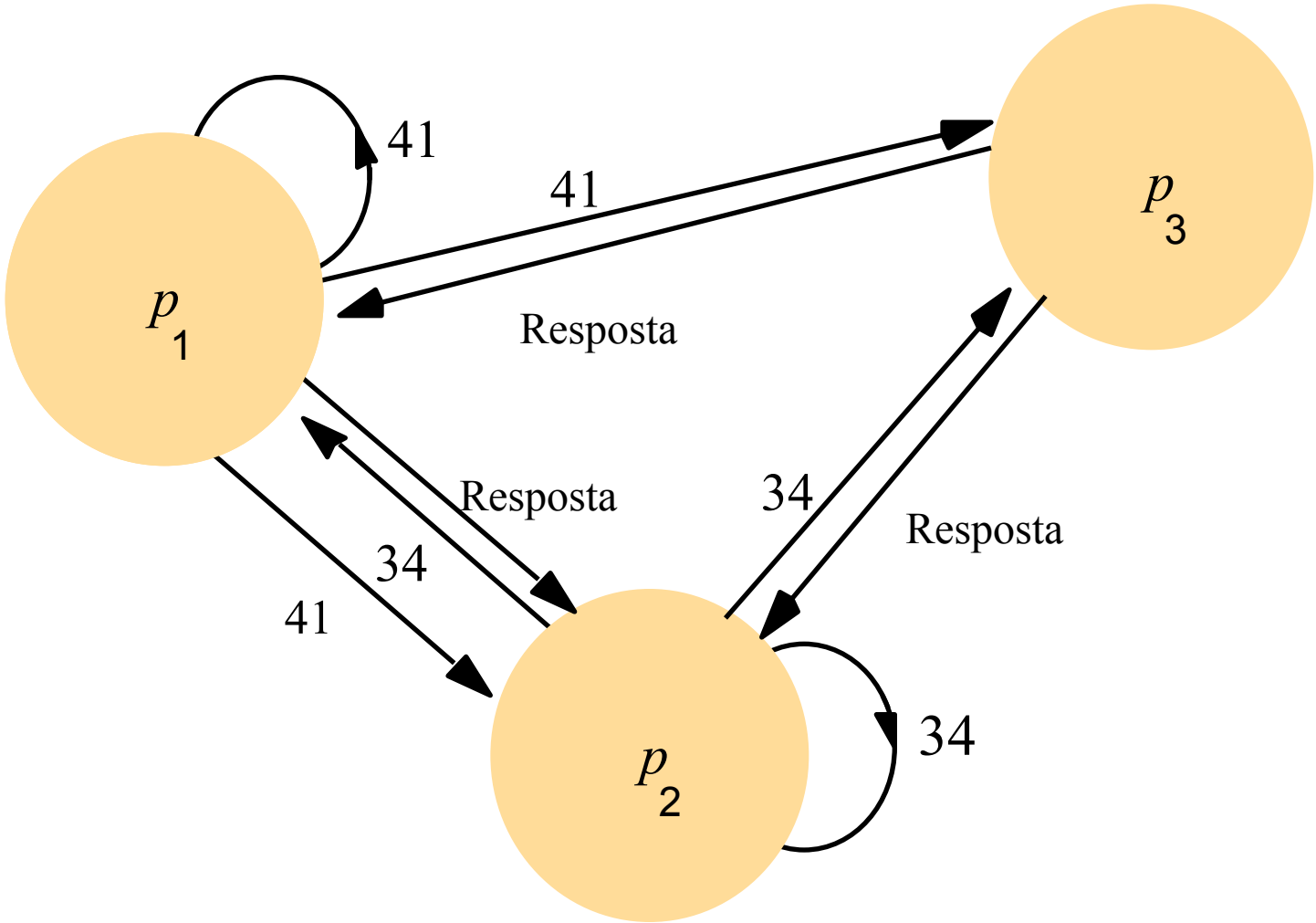


Figura 15.5

Algoritmo de Ricart e Agrawala

Na inicialização
    state := RELEASED;
Para entrar na seção
    state := WANTED;
    Envia a requisição por *multicast* para todos processos;
    T := carimbo de tempo de requisição;
    Espera até (número de respostas recebidas = (N – 1));
    state := HELD;

No recebimento de uma requisição $<T_i, p_i>$ *at* p_j $(i \neq j)$
    if (*state* = HELD or (*state* = WANTED *and* $(T, p_j) < (T_i, p_i)$))
    then
        enfileira requisição de p_i sem responder;
    else
        responde imediatamente para p_i;
    end if
Para sair da seção crítica
    state := RELEASED;
    responde a todas as requisições enfileiradas;

Processamento de requisição

Figura 15.4

Bônus

- 0,5 de bônus na nota da prova 2
- Em duplas, 0,5 para cada um.
- Explicar o algoritmo do próximo slide usando só o seguinte slide e o quadro
- Irei sortear uma dupla
- 30 minutos de preparação

- **Ao final da apresentação farei duas perguntas**
- Cada grupo pode oferecer uma pergunta, se eu usar a pergunta a dupla ganha 0,1 de bônus
- Resposta certa: + 0,1 de bônus
- Resposta errada: - 0,1

Algoritmo de *Maekawa*

Na inicialização

state := RELEASED;

voted := FALSE;

Para p_i *entrar na seção crítica*

state := WANTED;

Envia requisição por *multicast* para todos processos em V_i;

Espera até (número de respostas recebidas = K);

state := HELD;

No recebimento de uma requisição de p_i *em* p_j

if (*state* = HELD *or voted* = TRUE)

then

enfileira a requisição de p_i sem responder;

else

envia repostas para p_i;

voted := TRUE;

end if

Para p_i *sair da seção crítica*

state := RELEASED;

Envia a *liberação* via *multicast* para todos os processo em V_i;

No recebimento de uma liberação de p_i *em* p_j

if (fila de requisições não estiver vazia)

then

remove cabeça da fila – digamos p_k;

envia *resposta* para p_k;

voted := TRUE;

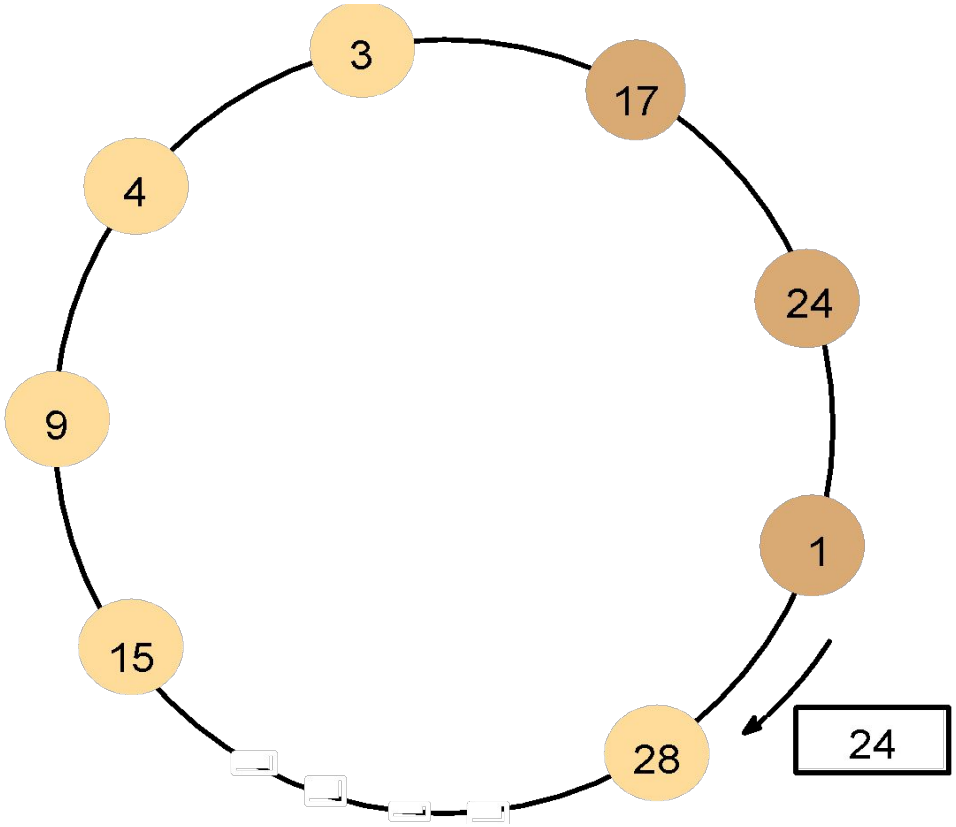
else

voted := FALSE;

end if

Figura 15.6

Uma eleição baseada em anel (em andamento)



Nota: A eleição foi iniciada pelo processo 17.
O identificador de processo mais alto encontrado até aqui é 24. Os processos participantes aparecem com fundo de cor mais escura.

Figura 15.7

O algoritmo valentão

A eleição do coordenador p_2, após a falha de p_4 e depois, de p_3

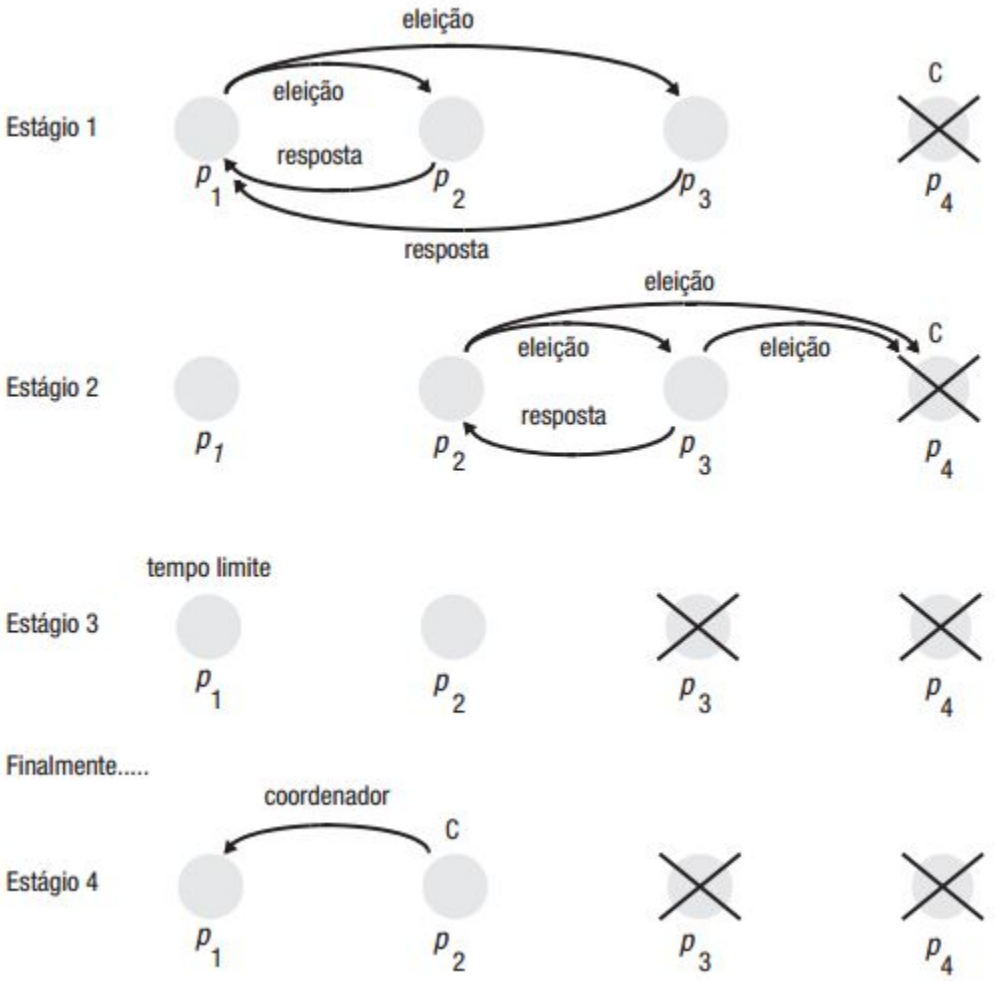


Figura 15.8

Bônus

Pag. 646 até 653

- 0,5 de bônus na nota da prova 2
- Em duplas, 0,5 para cada um.
- Explicar o algoritmo do próximo slide usando só o seguinte slide e o quadro
- Irei sortear uma dupla
- 30 minutos de preparação

- **Ao final da apresentação farei duas perguntas**
- Cada grupo pode oferecer uma pergunta, se eu usar a pergunta a dupla ganha 0,1 de bônus
- Resposta certa: + 0,1 de bônus
- Resposta errada: - 0,1

Algoritmo de *multicast* confiável

Na inicialização

Received := { };

Para o processo p enviar a mensagem m com R-multicast para o grupo g

B-multicast (*g*, *m*); // $p \in g$ é incluído como destino

Em B-deliver(m) no processo q com g = group(m)

if ($m \notin$ *Received*)

then

Received := *Received* $\cup$ { *m* };

if ($q \neq p$) *then B-multicast* (*g*, *m*); *end if*

R-deliver m;

end if

Figura 15.9

A fila de espera para mensagens *multicast* recebidas

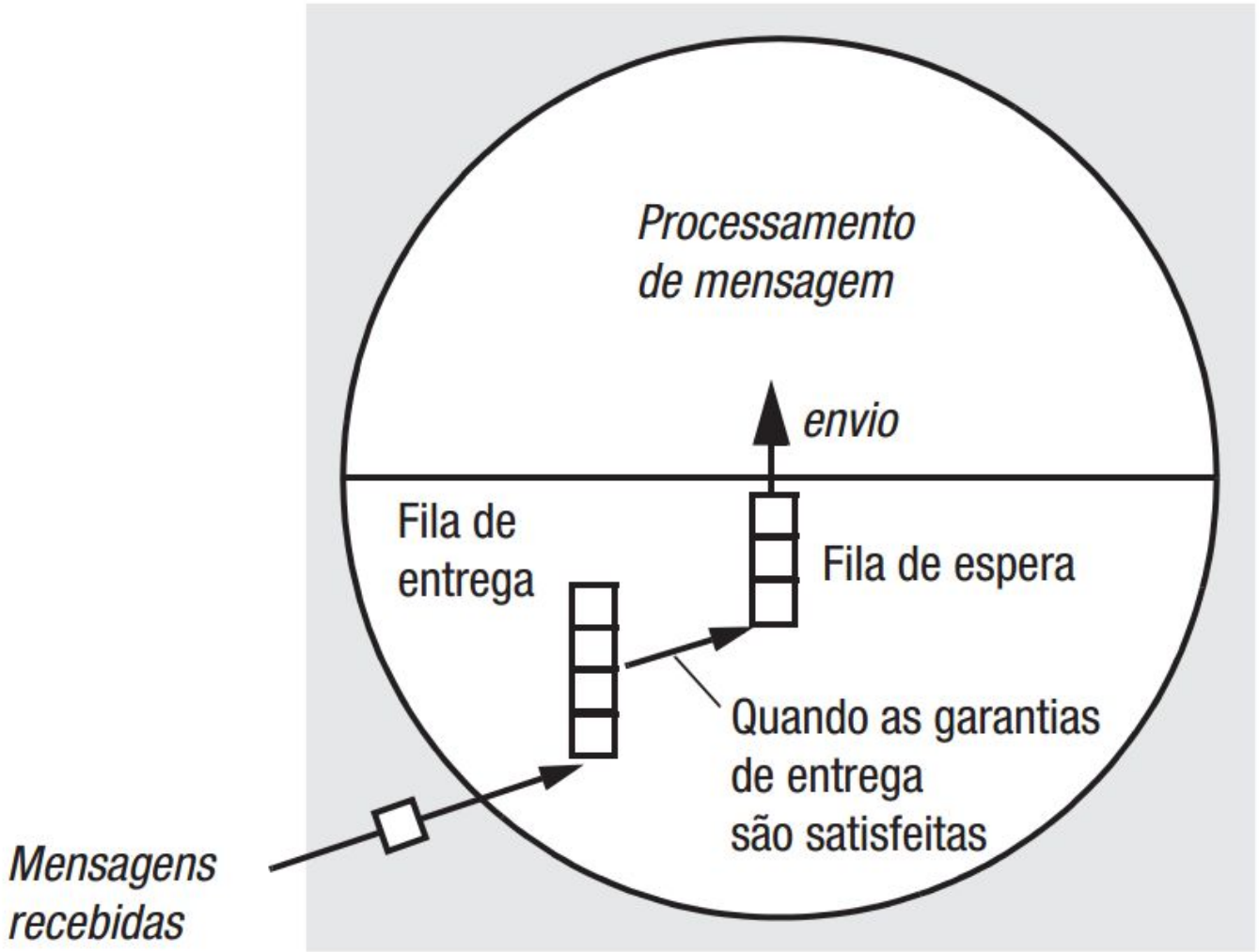


Figura 15.10

Bônus

Pag. 654 até 656
Ordem total
Comparar os 2 métodos

- 0,5 de bônus na nota da prova 2
- Em duplas, 0,5 para cada um.
- Explicar o algoritmo do próximo slide usando só o seguinte slide e o quadro
- Irei sortear uma dupla
- 30 minutos de preparação

- **Ao final da apresentação farei duas perguntas**
- Cada grupo pode oferecer uma pergunta, se eu usar a pergunta a dupla ganha 0,1 de bônus
- Resposta certa: + 0,1 de bônus
- Resposta errada: - 0,1

Ordenação total, FIFO e causal de mensagens de *multicast*

Observe a ordem consistente das mensagens totalmente ordenadas *T1* e *T2*, das mensagens relacionadas com FIFO *F1* e *F2* e das mensagens relacionadas por causalidade *C1* e *C3* – e a ordem de distribuição arbitrária das mensagens.

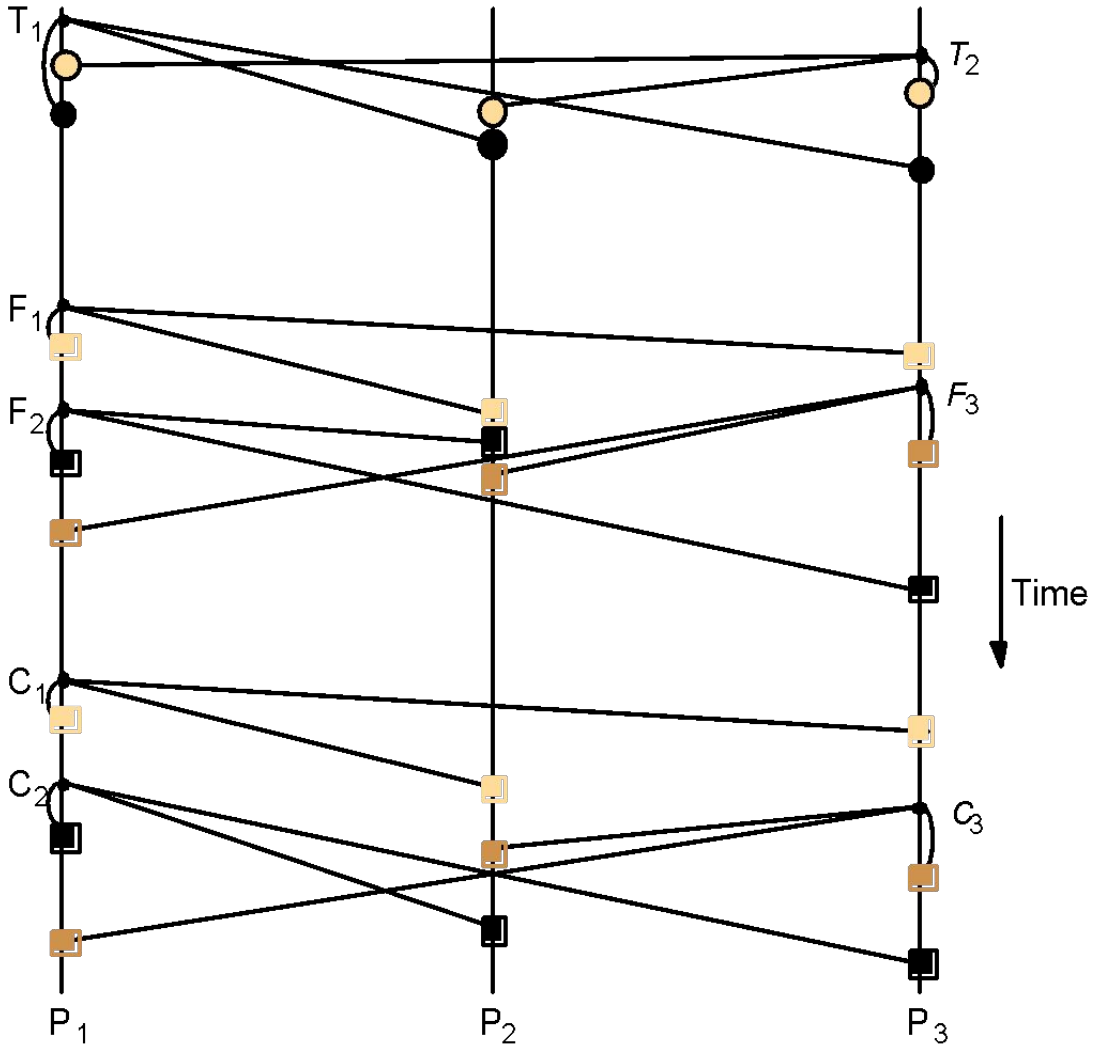


Figura 15.11

Tela do programa de listas de discussão

Lista de discussão: *os.interesting*		
Item	De	Assunto
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
fim		

Figura 15.12

Ordem total usando um sequenciador

1. Algoritmo do membro do grupo *p*

Na inicialização: $r_g := 0;$

Para enviar a mensagem m com TO-multicast para o grupo g

B-multicast ($g \cup \{sequencer(g)\}$, <*m, i*>);

Em B-deliver (< *m, i*>) *com g = group*(*m*)

Coloca <*m, i*> na fila de espera;

Em B-deliver (m_{ordem} = <"ordem",*i, S*>) *com g = group* (m_{ordem})

espera até <*m, i*> na fila de espera e $S = r_g$;

TO-deliver m ; // (após excluí-la da fila de espera)

$r_g := S+1;$

2. Algoritmo do sequenciador de *g*

Na inicialização: $s_g := 0;$

Em B-deliver (<*m, i*>) *com g = group* (*m*)

B-multicast (*g*, < "ordem",*i*, s_g >);

$s_g := s_g + 1;$

Figura 15.13

O algoritmo ISIS para ordenação total

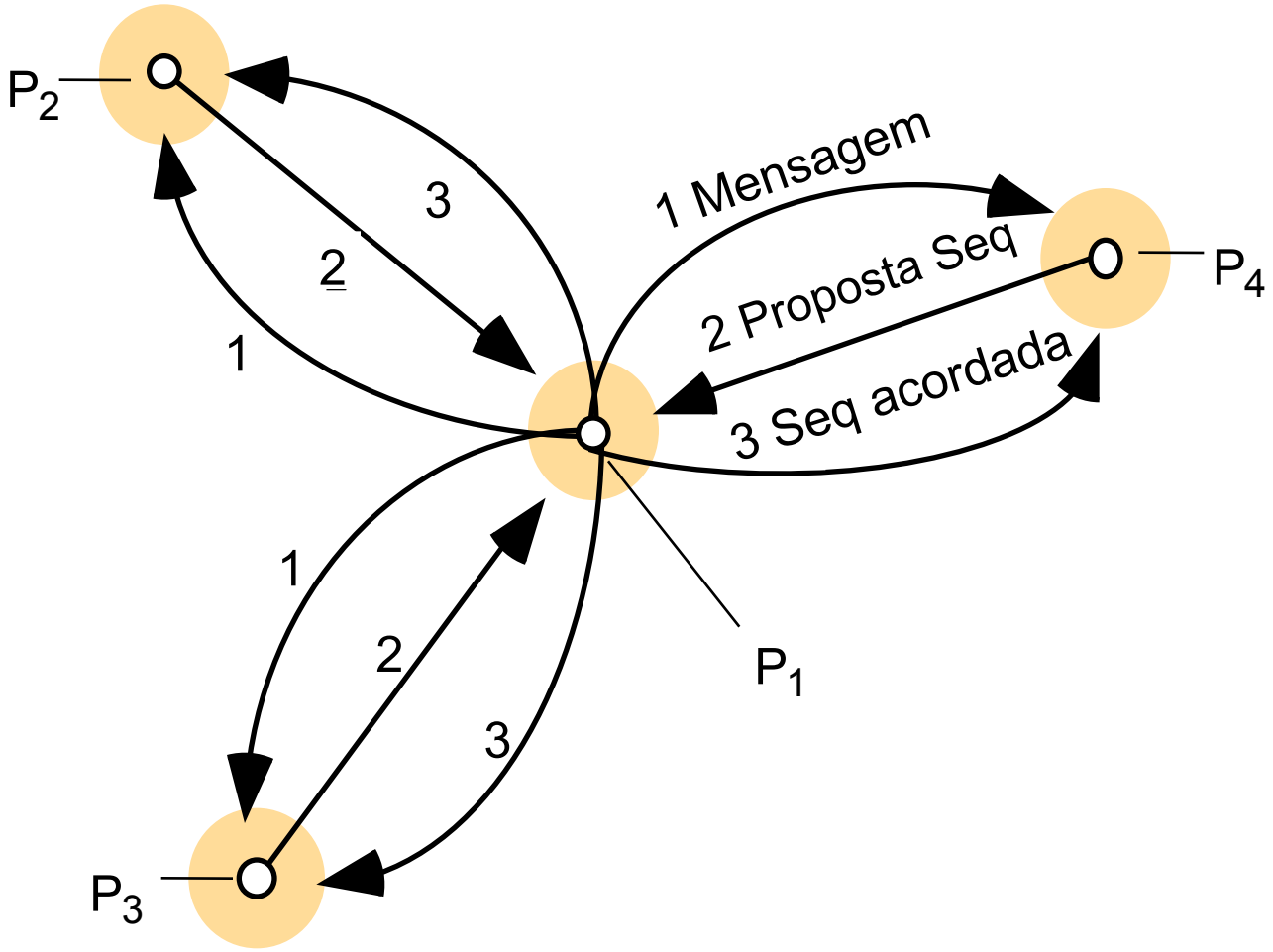


Figura 15.14

Bônus

Pag. 657 até 658

- 0,5 de bônus na nota da prova 2
- Em duplas, 0,5 para cada um.
- Explicar o algoritmo do próximo slide usando só o seguinte slide e o quadro
- Irei sortear uma dupla
- 30 minutos de preparação

- **Ao final da apresentação farei duas perguntas**
- Cada grupo pode oferecer uma pergunta, se eu usar a pergunta a dupla ganha 0,1 de bônus
- Resposta certa: + 0,1 de bônus
- Resposta errada: - 0,1

Ordenação causal usando carimbos de tempo vetoriais

Algoritmo para membro de grupo p_i $(i = 1, 2..., N)$

Na inicialização

$V_i^g[j] := 0\ (j = 1, 2 \ldots, N);$

Para enviar a mensagem m com CO-multicast para o grupo g

$V_i^g[i] := V_i^g[i] + 1;$

B-multicast $(g, < V_i^g, m>);$

Em B-deliver $(< V_j^g, m>)$ *de* p_j $(j \neq i)$, *com* $g = group\ (m)$

coloca $< V_j^g, m>$ na fila de espera;

espera até que $V_j^g[j] = V_i^g[j] + 1$ e $V_j^g[k] \leq V_i^g[k]\ (k \neq j);$

CO-deliver m; // após removê-la da fila de espera

$V_i^g[j] := V_i^g[j] + 1;$

Figura 15.15

Consenso de três processos

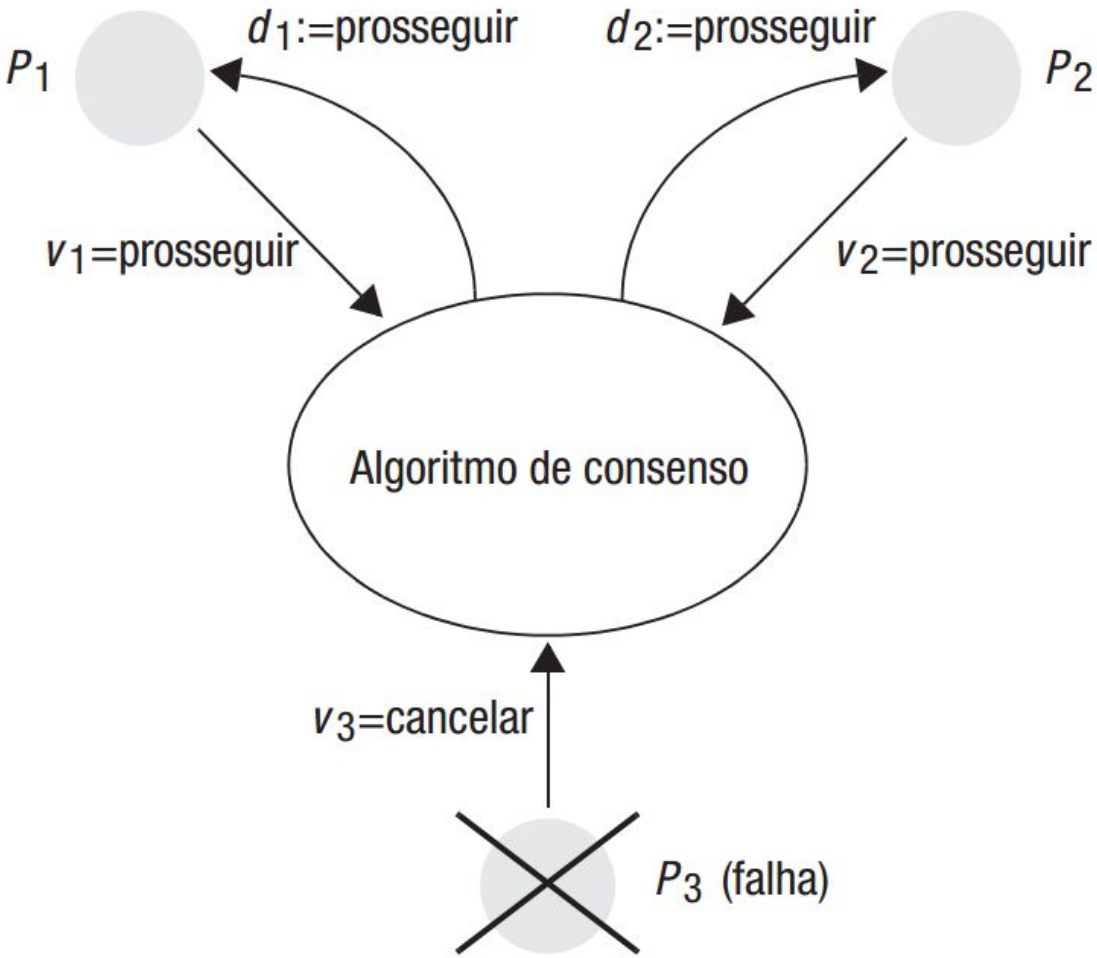


Figura 15.16

Consenso em um sistema síncrono

Algoritmo do processo $p_i \in g$; o algoritmo prossegue em $f + 1$ rodadas

Na inicialização

$Values_i^1 := \{v_i\}$; $Values_i^0 := \{\}$;

Na rodada r $(1 \leq r \leq f + 1)$

B-multicast$(g, Values_i^r - Values_i^{r-1})$; // Envia apenas os valores que não foram enviados

$Values_i^{r+1} := Values_i^r$;

while (na rodada *r*)

{

Em B-deliver (V_j) *de algum processo* p_j

$Values_i^{r+1} := Values_i^{r+1} \cup Vj$;

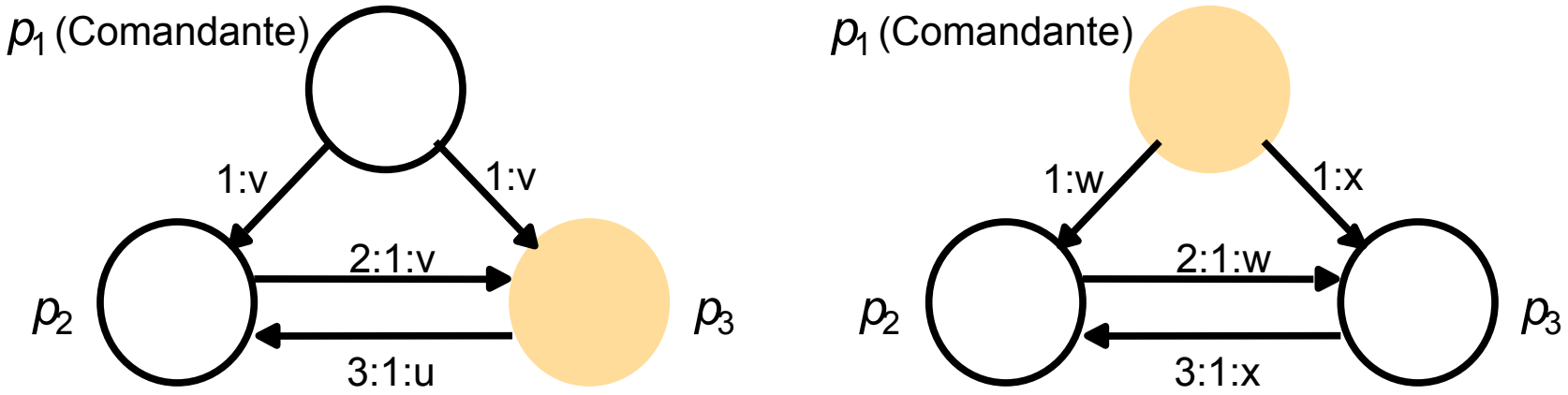
}

Após $(f + 1)$ *rodadas*

Atribui $d_i = minimum\ (Values_i^{f+1})$;

Figura 15.17

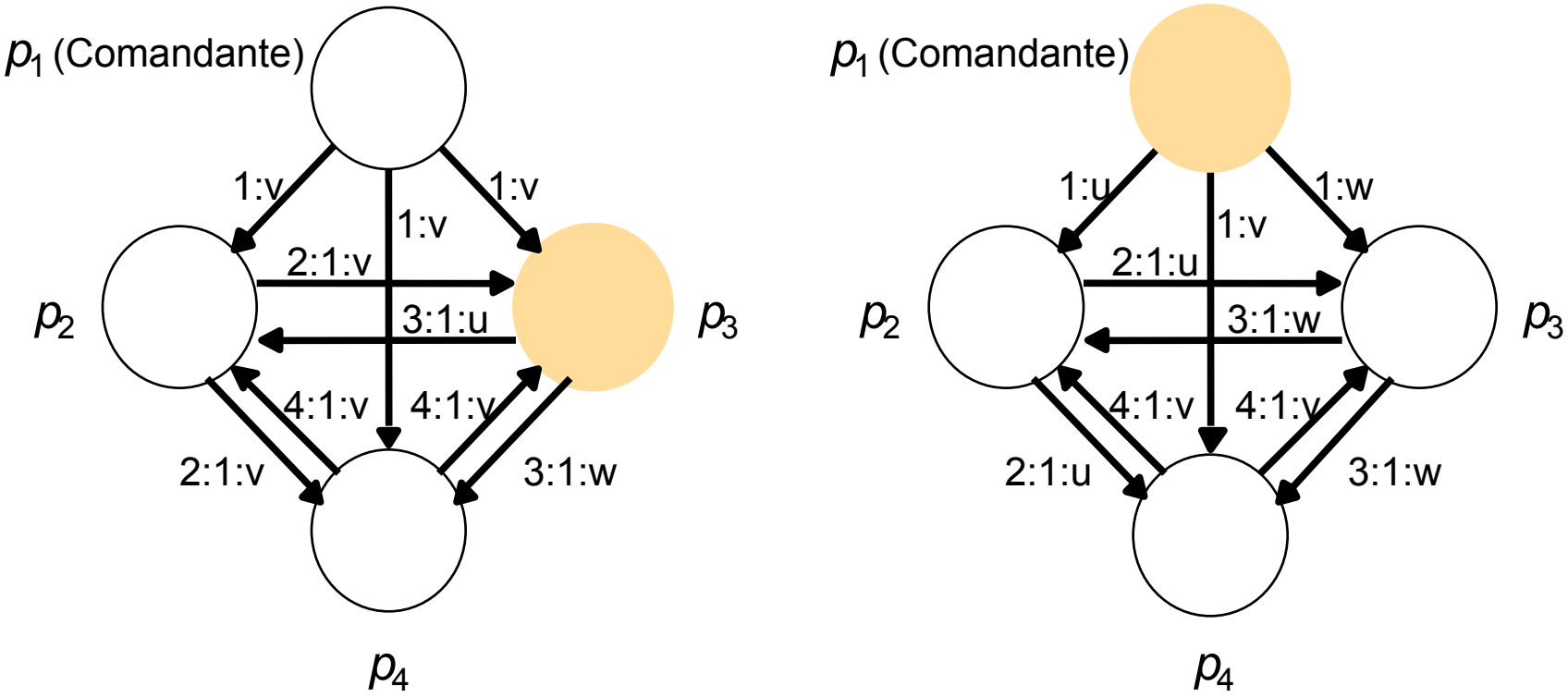
Três generais bizantinos



Os processos falhos aparecem em amarelo

Figura 15.18

Quatro generais bizantinos



Os processos falhos aparecem em amarelo

Figura 15.19

Ler o capítulo 15!