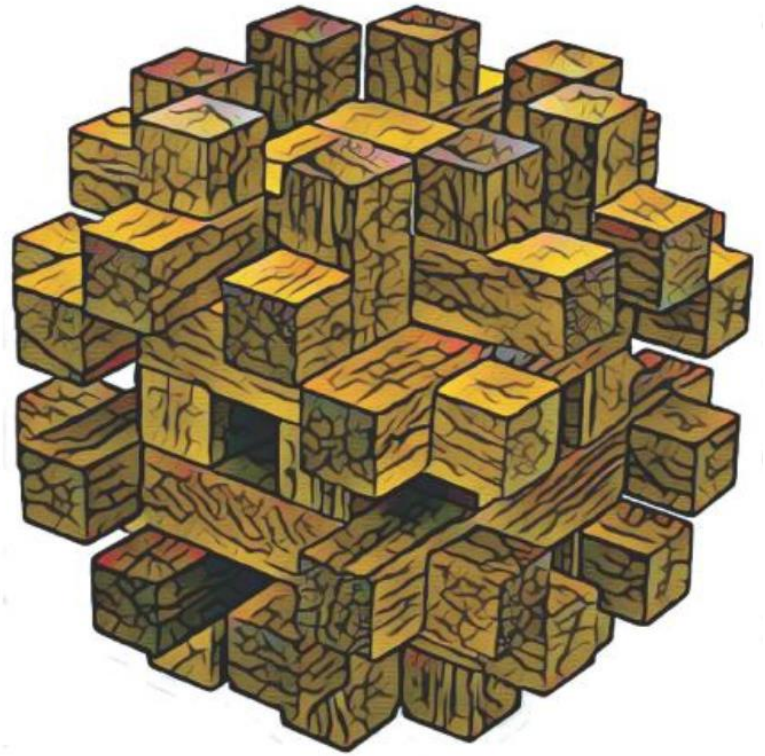
Distributed Systems

Maarten Van Steen & Andrew S. Tanenbaum



3th Edition – Version 3.03 - 2020

Capítulo 6

Coordenação

Quinta-feira, 11 de Agosto de 2022

RELÓGIOS FÍSICOS

PROBLEMA

As vezes precisamos da hora exata e não somente de um enfileiramento

SOLUÇÃO: UTC – UNIVERSAL COORDINATED TIME

- Baseado em um número de transições por segundo do átomo de césio 133
- Até o momento, o tempo real é calculado como a média de 50 relógios atômicos ao redor do mundo
- Introduz um segundo de “leap” de tempos em tempos para compensar o fato que os dias estão de alongando.

NOTA

- UTC é transmitido através de ondas curtas de rádio e satélite. Satélites tem precisão de ±0,5ms

SINCRONIZAÇÃO DE RELÓGIOS

PRECISÃO

O objetivo é manter um desvio entre dois relógios ou duas máquinas quaisquer dentro de um limite específico, conhecido como precisão π:

$$\forall t, \forall p, q : |C_p(t) - C_q(t)| \leq \pi$$

Com $C_p(t)$ o tempo de relógio computado na máquina *p* no tempo *t* hora UTC

ACURÁCIA

- No caso da acurácia, queremos manter o relógio vinculado a um valor α: $\forall t, \forall p : |C_p(t) - t| \leq \alpha$

SINCRONIZAÇÃO

- Sincronização interna: mantém o clock preciso
- Sincronização externa: mantém o relógio acurado

CLOCK DRIFT

ESPECIFICAÇÕES DE RELÓGIO

- Um relógio vem com uma especificação de uma taxa máxima ρ de drift de clock
- F(t) denota a frequência do oscilador do hardware do relógio no tempo t
- F é a frequência ideal (constante) -> atendendo as especificações

$$\forall t : (1-\rho) \leq \frac{F(t)}{F} \leq (1+\rho)$$

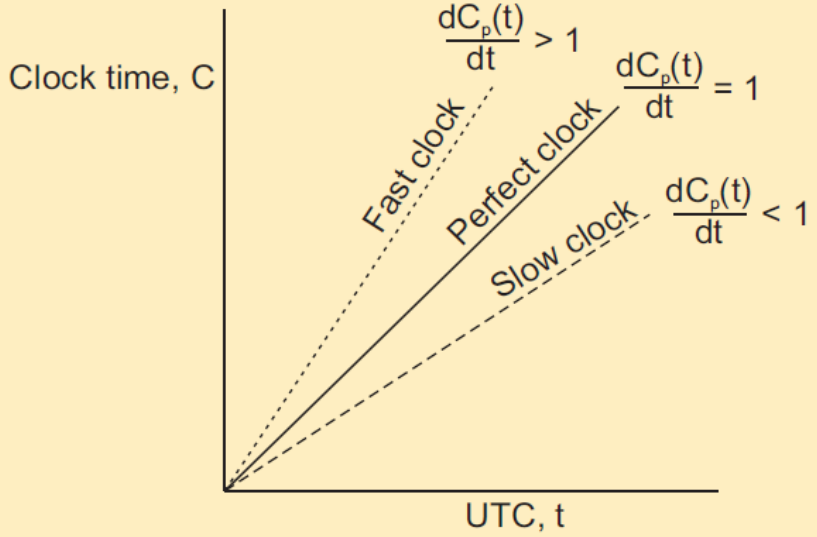
OBSERVAÇÃO

Usando interrupções de hardware, fazemos o casamento de um relógio de software com o relógio de hardware e assim a taxa de drift de relógio fica:

$$C_p(t) = \frac{1}{F}\int_0^t F(t)dt \Rightarrow \frac{dC_p(t)}{dt} = \frac{F(t)}{F}$$

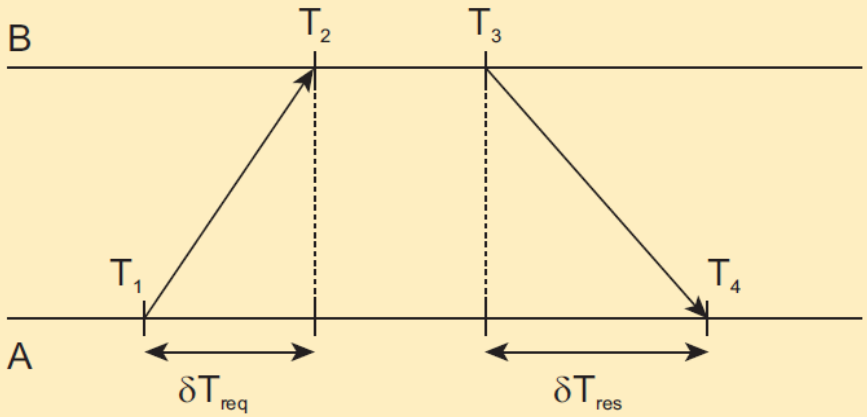
$$\Rightarrow \forall t : 1-\rho \leq \frac{dC_p(t)}{dt} \leq 1+\rho$$

RELÓGIOS:RÁPIDO, PERFEITO, DEVAGAR



DETECTANDO E AJUSTANDO HORAS INCORRETAS

PEGANDO HORÁRIO ATUAL EM SERVIDOR DE HORAS



OBSERVAÇÃO

supondo:

$$\delta T_{req} = T_2 - T_1 \approx T_4 - T_3 = \delta T_{res}$$

$$\theta = T_3 + ((T_2 - T_1) + (T_4 - T_3))/2 - T_4 = ((T_2 - T_1) + (T_3 - T_4))/2$$

$$\delta = ((T_4 - T_1) - (T_3 - T_2))/2$$

NTP – Network Time Protocol

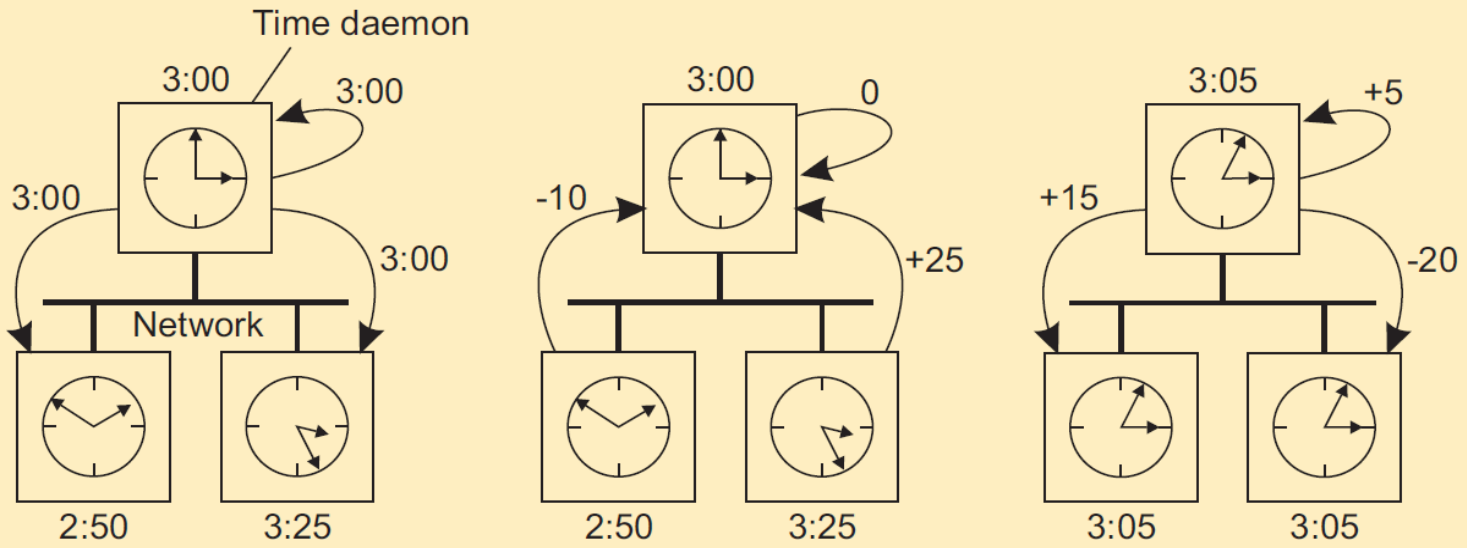
Colete 8 pares (θ, δ) e escolha θ para o qual o atraso associado foi mínimo.

MANTENDO A HORA SEM UTC

PRINCÍPIO

Deixe o servidor de hora escanear todas as máquinas periodicamente, calcule a media, e informe cada máquina como ela deve ajustar seu horário relative ao seu horário atual

USANDO UM SERVIDOR DE HORA



FUNDAMENTAL

É necessário considerer que o ajuste de horário para trás não é permitido => ajustes suaves (i.é , rodar mais rápido ou mais devagar).

SINCRONIZAÇÃO COM BROADCAST DE REFERÊNCIA

ESSÊNCIA

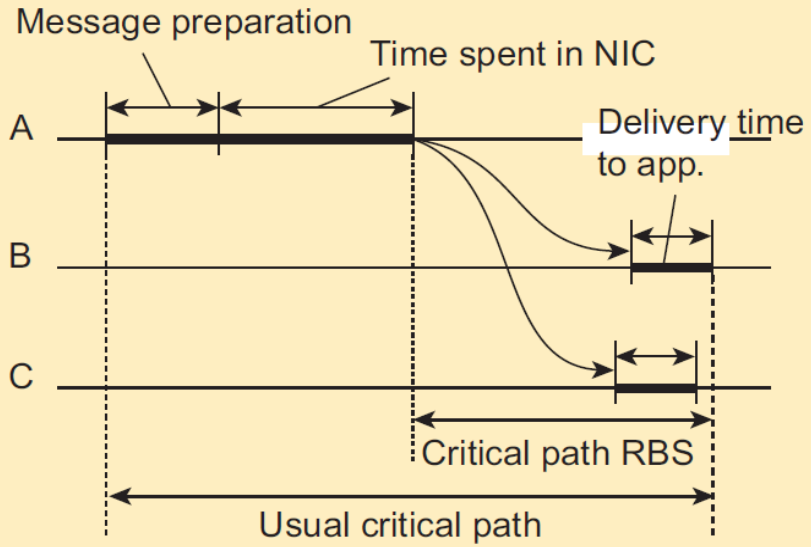
- Um nó transmite (*broadcast*) uma mensagem de referência *m* => cada nó recebedor *p* registra o horário $T_{p,m}$ que recebeu *m*
- Nota: $T_{p,m}$ é lido do relógio local de *p*

PROBLEMA: A CAPTURA DA MÉDIA NÃO CAPTURA DRIFT => USE REGRESSÃO LINEAR

NO: $Offset[p,q](t) = \frac{\sum_{k=1}^{M}(T_{p,k} - T_{q,k})}{M}$

YES: $Offset[p,q](t) = \alpha t + \beta$

RBS MINIMIZA O CAMINHO CRÍTICO



A RELAÇÃO ACONTECEU ANTES

QUESTÃO

O que normalmente importa não é que todos processos concordam exatamente em que hora é, mas que eles concordam em que ordem os eventos aconteceram. Demanda uma noção de ordenamento.

A RELAÇÃO ACONTECEU ANTES (HAPPENED-BEFORE)

- Se *a* e *b* são eventos no mesmo processo e *a* vem antes de *b*, então $a \rightarrow b$.
- Se a é o envio de uma mensagem e *b* é a recepção desta mensagem, então $a \rightarrow b$.
- Se $a \rightarrow b$ *e* $b \rightarrow c$, então $a \rightarrow c$.

NOTA

- Isto introduz um ordenamento parcial de eventos em um sistema com processos operacionais concorrentes.

RELÓGIOS LÓGICOS

PROBLEMA

Como mantemos uma visão global sobre o comportamento do sistema com a relação aconteceu antes ?

ANEXE UM TIMESTAMP *C(e)* EM CADA EVENTO *e*, SATISFAZENDOO AS SEGUINTES PROPRIEDADES

P1: Se *a* e *b* são eventos no mesmo processo, e $a \rightarrow b$, então requisitamos que $C(a) < C(b)$

P2: Se *a* corresponde ao envio de uma mensagem *m*, e *b* ao recebimento desta mensagem, então $C(a) < C(b)$

PROBLEMA

Como anexar um *timestamp* a um evento quando não existe um relógio global => manter um conjunto consistente de relógios lógico, um por processo

RELÓGIOS LÓGICOS: SOLUÇÃO

CADA PROCESSO *Pi* MANTÉM UM CONTADOR LOCAL *Ci* E AJUSTA ESTE CONTADOR

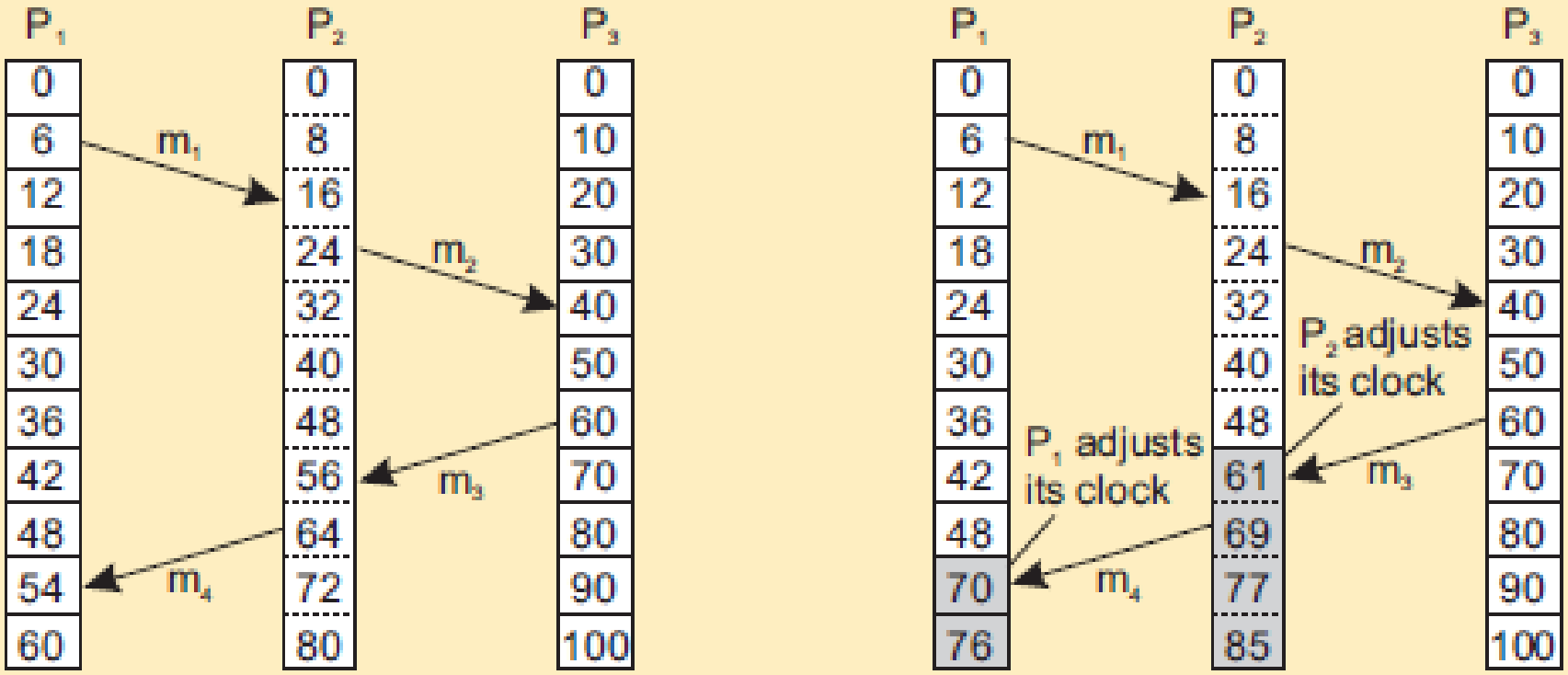
1. Para cada novo evento que acontece dentro de *Pi*, *Ci* é incrementado de 1
2. Cada vez que uma mensagem *m* é enviada por um processo *Pi*, a mensagem recebe um timestamp *ts(m) = Ci.*
3. Toda vez que um mensagem *m* é recebida por um processo *Pj*, *Pj* ajusta seu contador local *Cj* para *max{Cj , ts(m)}*; então executa passo 1 antes de passar *m* para a aplicação.

NOTAS

- Propriedade P1 é satisfeita por (1); Propriedade P2 por (2) e (3).
- Ainda pode ocorrer que dois eventos acontecem ao mesmo tempo. Evite isto quebrando as amarras através de ID´s de processos.

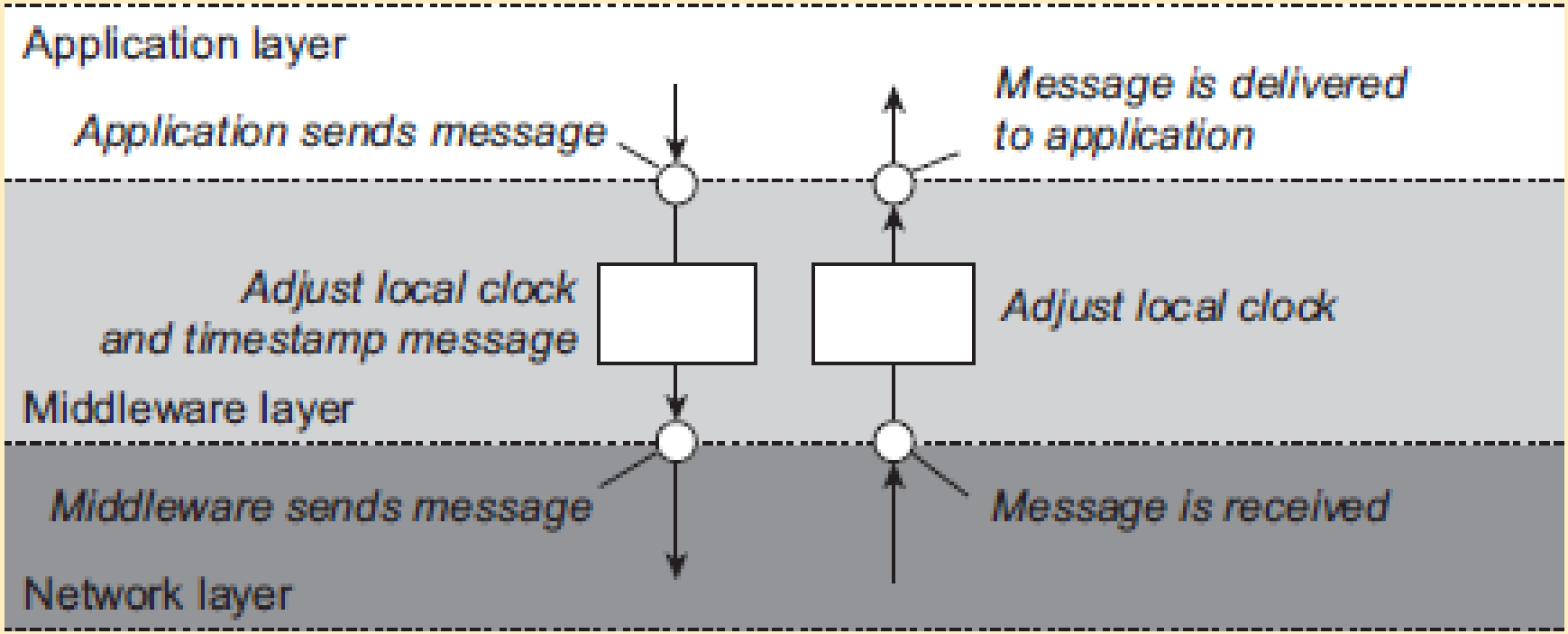
EXEMPLO RELÓGIOS LÓGICOS

CONSIDERE TRÊS PROCESSOS COM CONTADORES DE EVENTOS OPERANDO EM TAXAS DIFERENTES



RELÓGIOS LÓGICOS ONDE IMPLEMENTADOS

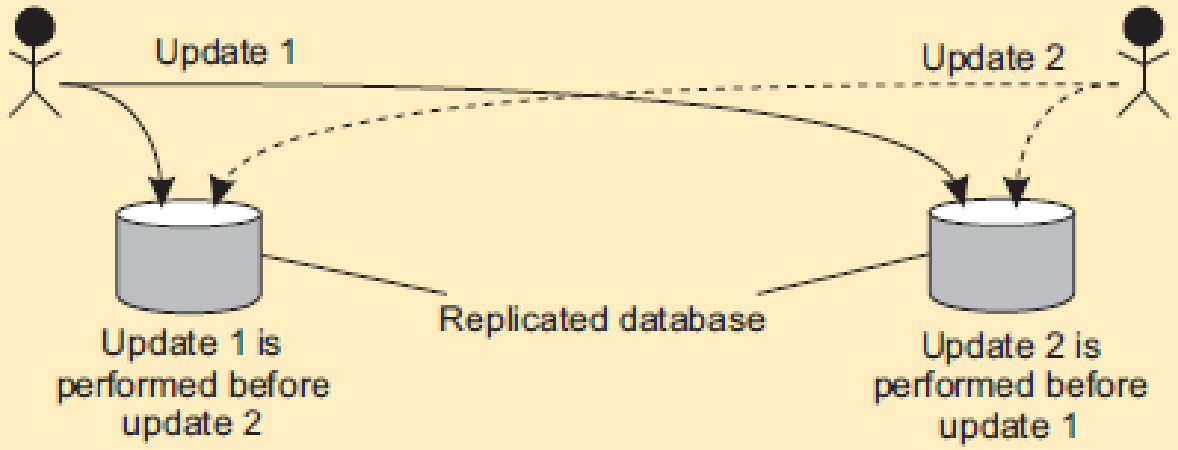
AJUSTES IMPLEMENTADOS EM MIDDLEWARE



EXEMPLO MULTICAST TOTALMENTE ORDENADO

ATUALIZAÇÕES CONCORRENTES EM BASES DE DADOS REPLICADAS SÃO VISTAS NA MESMA ORDEM EM TODOS LUGARES

- P1 adiciona $100 a uma conta (valor inicial: $1000)
- P2 incrementa a conta em 1%
- Existe duas replicas



RESULTADO

- Na falta de sincronização adequada:
- Réplica #1 ← $1111, enquanto réplica #2 ← $1100

EXEMPLO MULTICAST TOTALMENTE ORDENADO

SOLUÇÃO

- Processo P_i envia mensagem com *timestamp* m_i para todos outros processos. A mensagem é colocada em uma fila local $queue_i$
- Qualquer mensagem entrante em P_j é enfileirada em $queue_j$, de acordo com seu *timestamp*, e reconhecida por todos outros processos

Pj PASSA A MENSAGEM m_i PARA SUA APLICAÇÃO SE:

(1) m_i esta na cabeça da fila $queue_j$

(2) Para cada processo P_k, existe uma mensagem m_k na $queue_j$ com um *timestamp* maior

NOTA

- Estamos assumindo que a comunicação é confiável e ordenada por FIFO

RELÓGIOS DE LAMPORT PARA EXCLUSÃO MÚTUA

```
1  class Process:
2    def __init__(self, chan):
3      self.queue      = []                    # The request queue
4      self.clock      = 0                     # The current logical clock
5
6    def requestToEnter(self):
7      self.clock = self.clock + 1                          # Increment clock value
8      self.queue.append((self.clock, self.procID, ENTER)) # Append request to q
9      self.cleanupQ()                                      # Sort the queue
10     self.chan.sendTo(self.otherProcs, (self.clock,self.procID,ENTER)) # Send request
11
12   def allowToEnter(self, requester):
13     self.clock = self.clock + 1                          # Increment clock value
14     self.chan.sendTo([requester], (self.clock,self.procID,ALLOW)) # Permit other
15
16   def release(self):
17     tmp = [r for r in self.queue[1:] if r[2] == ENTER]  # Remove all ALLOWs
18     self.queue = tmp                                     # and copy to new queue
19     self.clock = self.clock + 1                          # Increment clock value
20     self.chan.sendTo(self.otherProcs, (self.clock,self.procID,RELEASE)) # Release
21
22   def allowedToEnter(self):
23     commProcs = set([req[1] for req in self.queue[1:]]) # See who has sent a message
24     return (self.queue[0][1]==self.procID and len(self.otherProcs)==len(commProcs))
```

RELÓGIOS DE LAMPORT PARA EXCLUSÃO MÚTUA

```
1   def receive(self):
2     msg = self.chan.recvFrom(self.otherProcs)[1]        # Pick up any message
3     self.clock = max(self.clock, msg[0])                # Adjust clock value...
4     self.clock = self.clock + 1                         # ...and increment
5     if msg[2] == ENTER:
6       self.queue.append(msg)                            # Append an ENTER request
7       self.allowToEnter(msg[1])                         # and unconditionally allow
8     elif msg[2] == ALLOW:
9       self.queue.append(msg)                            # Append an ALLOW
10    elif msg[2] == RELEASE:
11      del(self.queue[0])                                # Just remove first message
12    self.cleanupQ()                                     # And sort and cleanup
```

RELÓGIOS DE LAMPORT PARA EXCLUSÃO MÚTUA

ANALOGIA COM MULTICAST TOTALMENTE ORDENADO

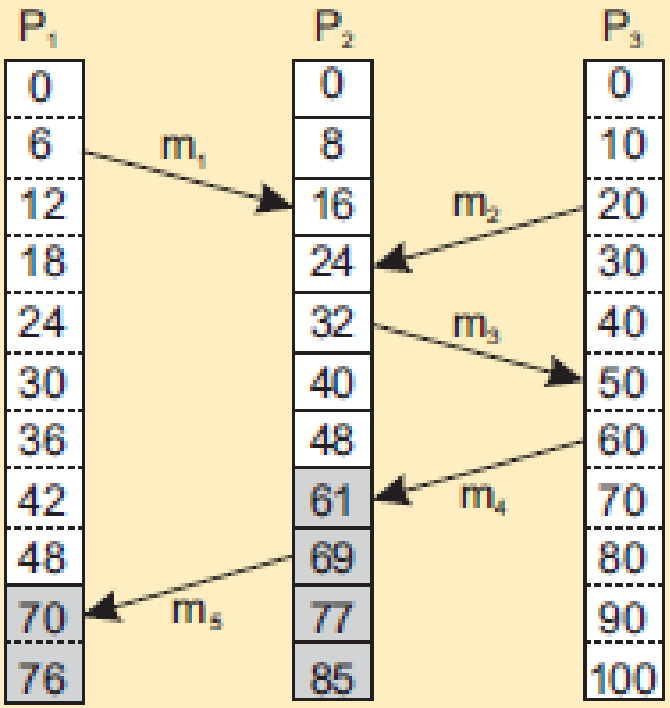
- Com *multicast* totalmente ordenado, todos processos constroem filas idênticas, e entregam mensagens na mesma ordem
- Exclusão mútua é sobre um acordo sobre a ordem que processos poderão entrar em seções críticas

RELÓGIOS VETORES

OBSERVAÇÃO

Relógios de Lamport não garantem que se $C(a) < C(b)$ então b é precedido por a.

TRANSMISSÃO CONCORRENTE DE MENSAGENS USANDO RELÓGIOS LÓGICOS



OBSERVAÇÃO

Evento a: m1 é recebido em T = 16

Evento b: m2 é enviado em T = 20

NOTA

Não podemos concluir que causalmente a precede b

DEPENDÊNCIA CAUSAL

DEFINIÇÃO

Dizemos que *b* pode causalmente depender em *a* se $ts(a) < ts(b)$, com:

- Para todo *k*, $ts(a)[k] \leq ts(b)[k]$ e
- Existe pelo menos um índice *k'* cujo $ts(a)[k'] < ts(b)[k']$

PRECEDÊNCIA VS DEPENDÊNCIA

- Dizemos que a causalmente precede *b*
- *b* pode causalmente depender em *a*, pois pode ter informação de a que é propagada pra *b*

CAPTURANDO CAUSALIDADE

SOLUÇÃO: CADA *Pi* MANTÉM UM VETOR *VCi*

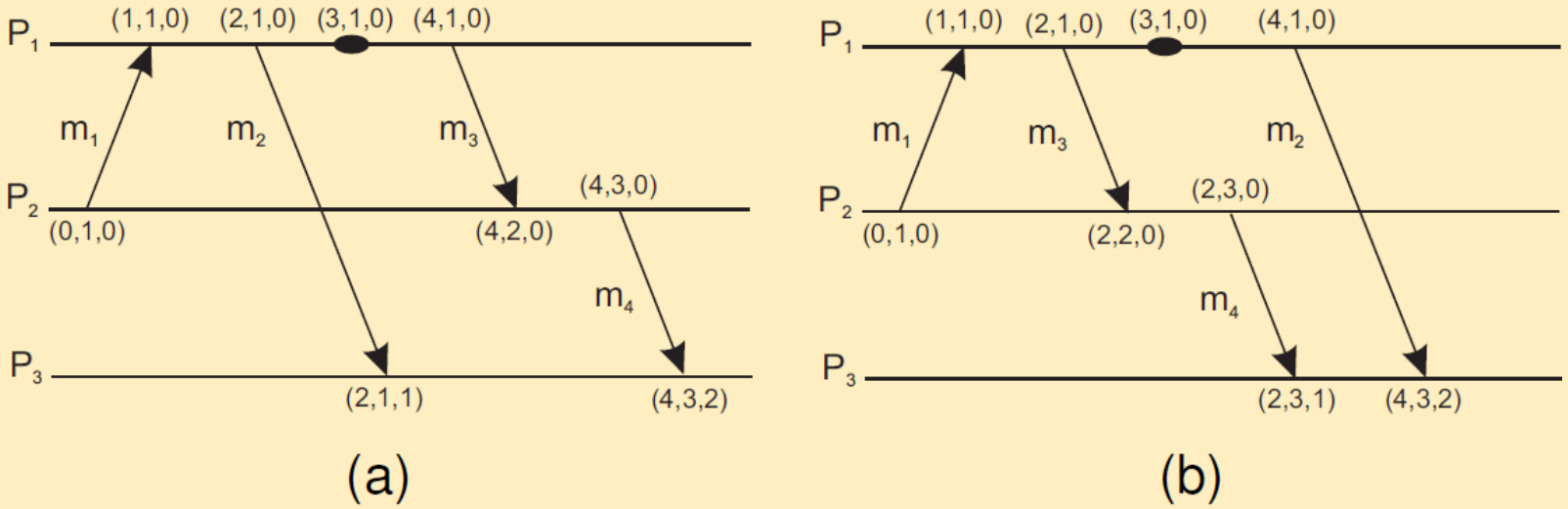
- *VCi[i]* é o relógio local lógico no processo *Pi*
- Se *VCi[j] = k* então *Pi* sabe que *k* eventos ocorreram em *Pj*

MANTENDO RELÓGIOS VETORES (VECTOR CLOCKS)

1. Antes de executar um evento *Pi* execute $VC[i] \leftarrow VC[i] + 1$.
2. Quando processo *Pi* envia uma mensagem *m* para *Pj*, ele seta o vetor *timestamp ts(m)* de *m* igual a *VCi* depois de ter executado passo 1.
3. No recebimento da mensagem *m*, o processo *Pj* seta $VCj[k] \leftarrow max\{VCj[k], ts(m)[k]\}$ para cada *k*, depois de executar passo 1 e então entregar a mensagem para a aplicação

EXEMPLO RELÓGIOS VETOR

CAPTURANDO CAUSALIDADE POTENCIAL NA TROCA DE MENSAGENS



ANÁLISE

Situation	$ts(m_2)$	$ts(m_4)$	$ts(m_2) < ts(m_4)$	$ts(m_2) > ts(m_4)$	Conclusion
(a)	(2,1,0)	(4,3,0)	Yes	No	m_2 may causally precede m_4
(b)	(4,1,0)	(2,3,0)	No	No	m_2 and m_4 may conflict

MULTICAST ORDENADO CAUSALMENTE

OBSERVAÇÃO

Podemos garantir que uma mensagem é entregue somente se todas mensagens causais precedentes foram entregues

AJUSTE

Pi incrementa *VC[i]* somente quando estiver enviando uma mensagem e *Pj* ajusta V*Cj* quando estiver recebendo uma mensagem (i.e., efetivamente não muda *VCj[j]).*

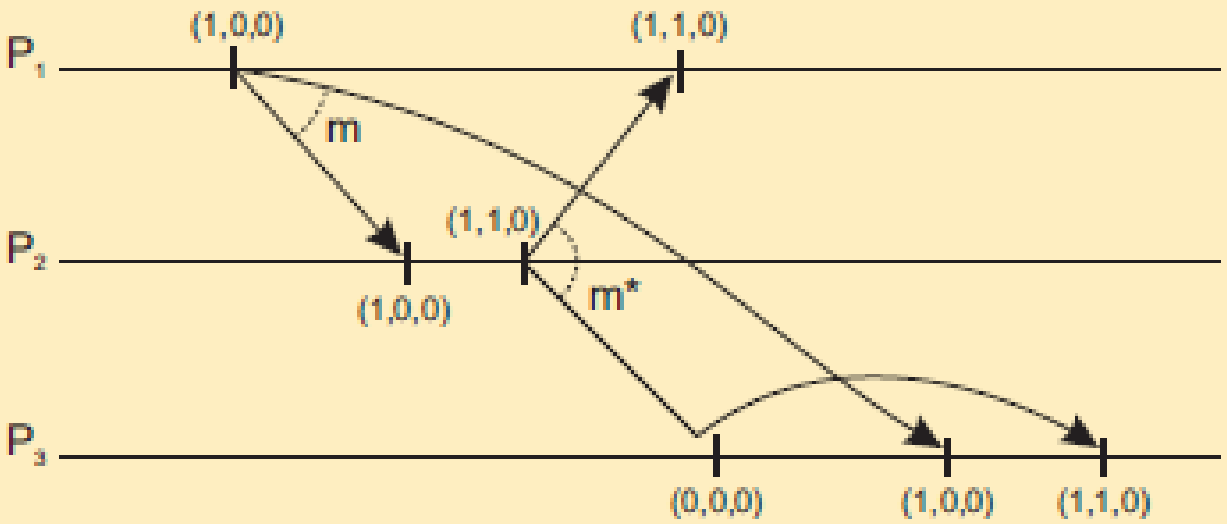
Pj POSTERGA ENTREGA DE m ATÉ:

(1) $ts(m)[i] = VCj[i] + 1$

(2) $ts(m)[k] \leq VCj[k]$ para todo $k \neq i$

MULTICAST ORDENADO CAUSALMENTE

AJUSTE



EXEMPLO

Pegue $VC_3 = [0,2,2]$, $ts(m) = [1,3,0]$ de P_1. Qual informação P_3 tem e qual terá quando estiver recebendo m (de P_1)?

EXCLUSÃO MÚTUA

PROBLEMA

Um número de processos em um sistema distribuído quer acesso exclusivo a um recurso

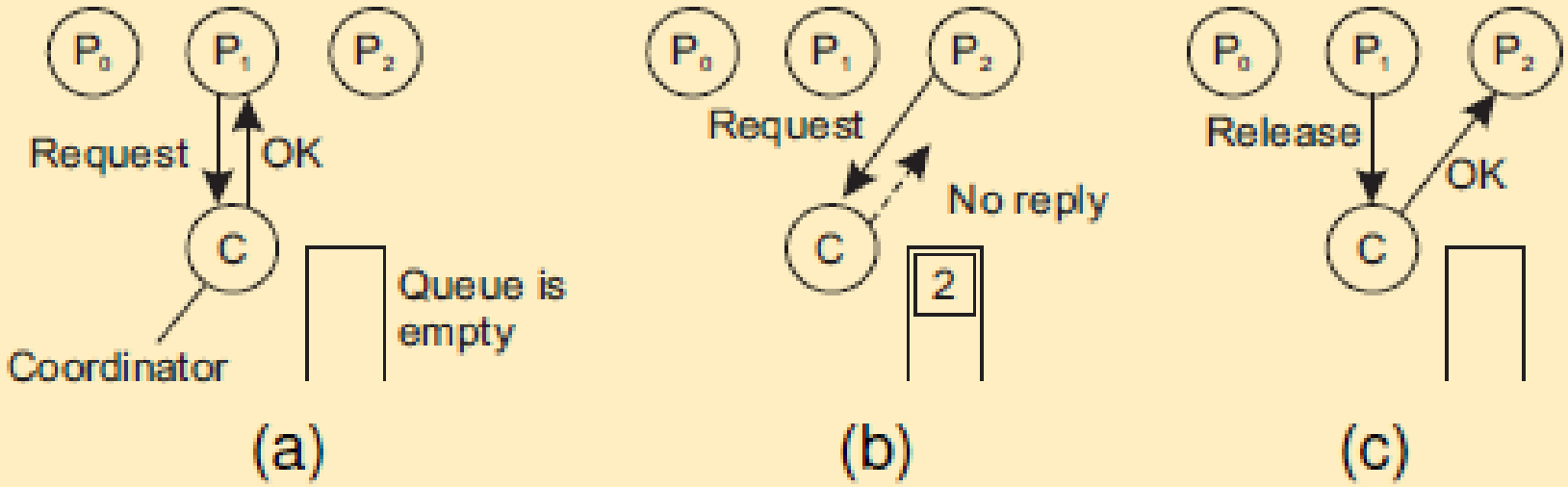
SOLUÇÕES BÁSICAS

Baseado em permissão: Um processo esperando para entrar em uma seção crítica ou acessar um recurso precisa permissão de outros processos

Baseado em token: Um token é passado entre processos. Aquele que tem o token pode entrar na seção crítica ou repassa o token caso não vá entrar.

BASEADO EM PERMISSÃO CENTRALIZADO

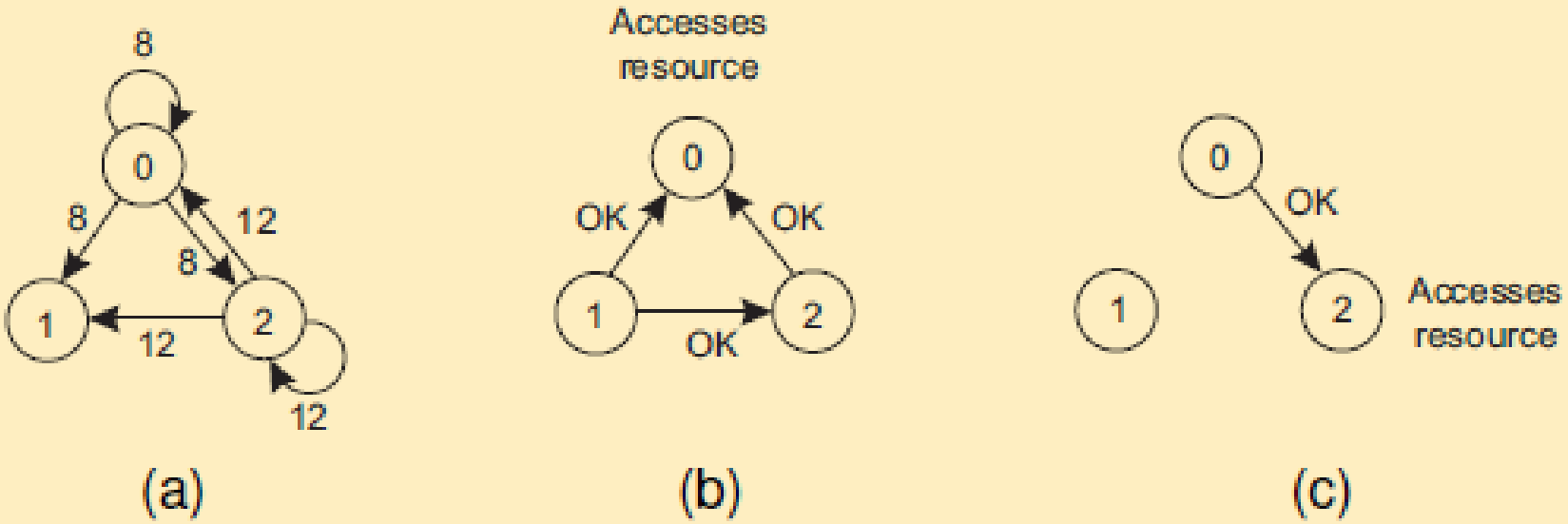
SOLUÇÕES BÁSICAS



a) Processo P1 pede ao coordenador permissão para acessar um recurso compartilhado. Permissão é dada.
b) Processo P2 então pede permissão para acessar o mesmo recurso. O coordenador não responde.
c) Quando P1 libera o recurso, ele avisa o coordenador que então envia resposta a P2

EXCLUSÃO MÚTUA RICART & AGRAWALA

EXEMPLO COM TRÊS PROCESSOS



a) Dois processos querem acessar um recurso compartilhado no mesmo momento

b) P_0 tem o menor *timestamp*, e então vence

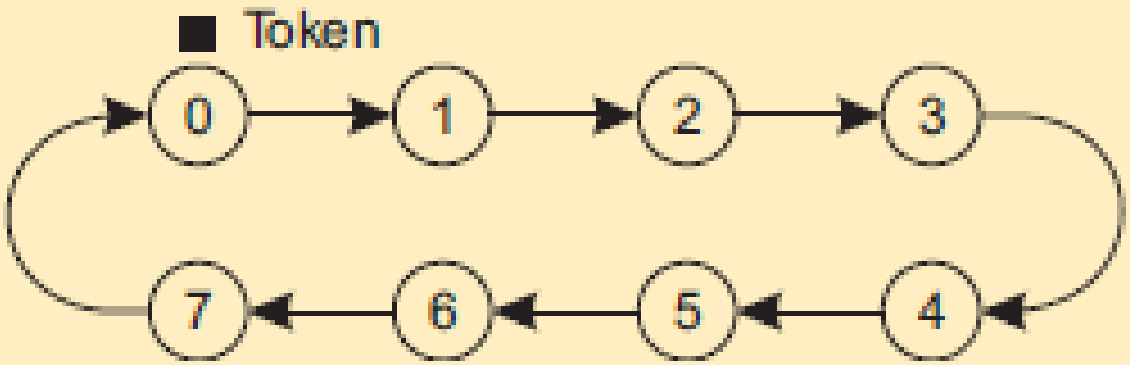
c) Quando processo P_0 terminou, ele envia um OK também, assim P_2 pode agora ir em frente

EXCLUSÃO MÚTUA ALGORITMO TOKEN RING

ESSÊNCIA

Organize um processo em um anel lógico e deixe um token se passado por ele. O que tiver o token terá permissão de entrar na região crítica (se quiser)

UMA REDE OVERLAY CONSTRUÍDA COMO ANEL LÓGICO COM TOKEN CIRCULANDO



EXCLUSÃO MÚTUA DECENTRALIZADA

PRINCÍPIO

Assuma que cada recurso é replicado N vezes, com cada replica tendo seu próprio coordenador => acesso requer uma maioria de votos de m > N/2 coordenadores. Um coordenador sempre responde imediatamente a uma requisição.

SUPOSIÇÃO

Quando um coordenador cai, ele se recupera rapidamente mas terá esquecido as permissões que já deu

EXCLUSÃO MÚTUA DECENTRALIZADA

QUÃO ROBUSTO É O SISTEMA

- Faça $p = \Delta t / T$ ser a probabilidade de um coordenador resetar durante o intervalo Δt, durante o tempo de vida T
- A probabilidade $\mathbb{P}[k]$ de k de m coordenadores resetar durante o mesmo intervalo é:

$$\mathbb{P}[k] = \binom{m}{k} p^k (1-p)^{m-k}$$

- f coordenadores resetam => exatidão é violada quando existe somente uma minoria de coordenadores sem problemas: quando $m - f \leq N/2$, ou $f \geq m - N/2$
- A probabilidade de violação é: $\sum_{k=m-N/2}^{N} \mathbb{P}[k]$

EXCLUSÃO MÚTUA DECENTRALIZADA

VIOLAÇÃO: PROBABILIDADES PARA VÁRIOS VALORES DE PARÂMETROS

N	m	p	Violation
8	5	3 sec/hour	$< 10^{-15}$
8	6	3 sec/hour	$< 10^{-18}$
16	9	3 sec/hour	$< 10^{-27}$
16	12	3 sec/hour	$< 10^{-36}$
32	17	3 sec/hour	$< 10^{-52}$
32	24	3 sec/hour	$< 10^{-73}$

N	m	p	Violation
8	5	30 sec/hour	$< 10^{-10}$
8	6	30 sec/hour	$< 10^{-11}$
16	9	30 sec/hour	$< 10^{-18}$
16	12	30 sec/hour	$< 10^{-24}$
32	17	30 sec/hour	$< 10^{-35}$
32	24	30 sec/hour	$< 10^{-49}$

ASSIM ..

O que podemos concluir ?

EXCLUSÃO MÚTUA COMPARAÇÃO

Algorithm	Messages per entry/exit	Delay before entry (in message times)
Centralized	3	2
Distributed	$2 \cdot (N-1)$	$2 \cdot (N-1)$
Token ring	$1, \ldots, \infty$	$0, \ldots, N-1$
Decentralized	$2 \cdot m \cdot k + m, k = 1, 2, \ldots$	$2 \cdot m \cdot k$

ALGORITMOS DE ELEIÇÃO

PRINCÍPIO

Um algoritmo requer que alguns processos ajam como coordenadores. A questão é como selecionar este processo especial dinamicamente.

NOTA

Em muitos sistemas o coordenador é escolhido manualmente (p.ex. servidores de arquivo). Isto leva a soluções centralizadas = > ponto único de falha

QUESTÕES

1. Se um coordenador é escolhido dinamicamente, até onde podemos falar sobre uma solução centralizada ou distribuída ?
2. Uma solução totalmente distribuída, isto é, sem um coordenador é sempre mais robusta que uma solução centralizada/coordenada ?

SUPOSIÇÕES BÁSICAS

- Todos processos possuem ID´s únicos
- Todos processos conhecem os ID´s de todos processos no sistema (mas não se eles estão ativos ou não ativos)
- Eleição significa a identificação de processos com o maior ID que está ativo

ELEIÇÃO POR BULLYING

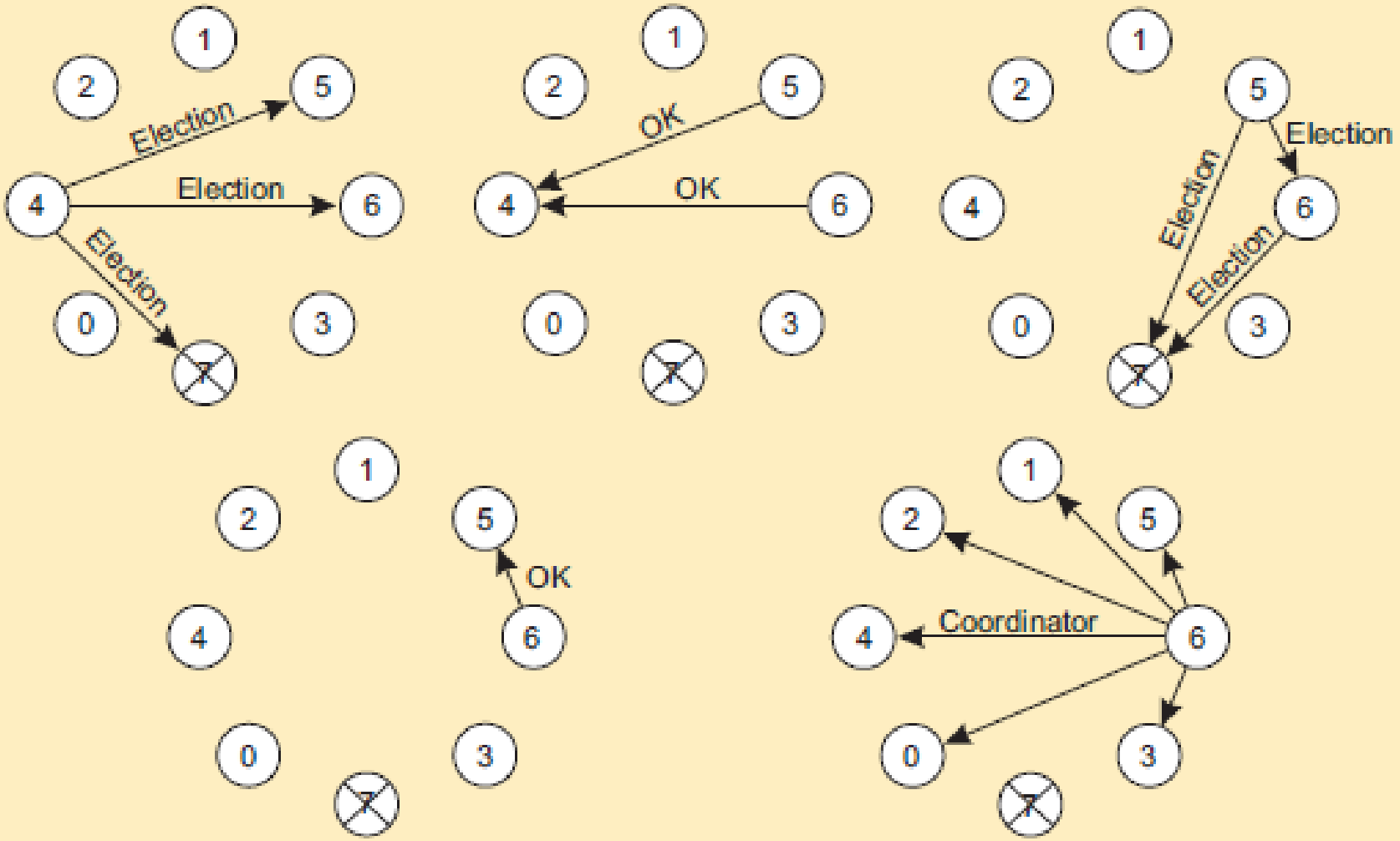
PRINCÍPIO

Considere N processos $\{P_0, ..., P_{N-1}\}$ e $id(P_k) = k$. Quando um processo P_k avisa que o coordenador não está mais respondendo a requisições, ele inicia uma eleição:

1. P_k envia uma mensagem ELEIÇÃO para todos processos com identificadores altos: $P_{k+1}, P_{k+2}, ..., P_{k-1}$.
2. Se não há resposta, P_k vence a eleição e se torna coordenador.
3. Se um dos mais altos responde, ele assume e o trabalho de P_k estará feito.

ELEIÇÃO POR BULLYING

O ALGORITMO DE ELEIÇÃO POR BULLYING



ELEIÇÃO EM UM ANEL

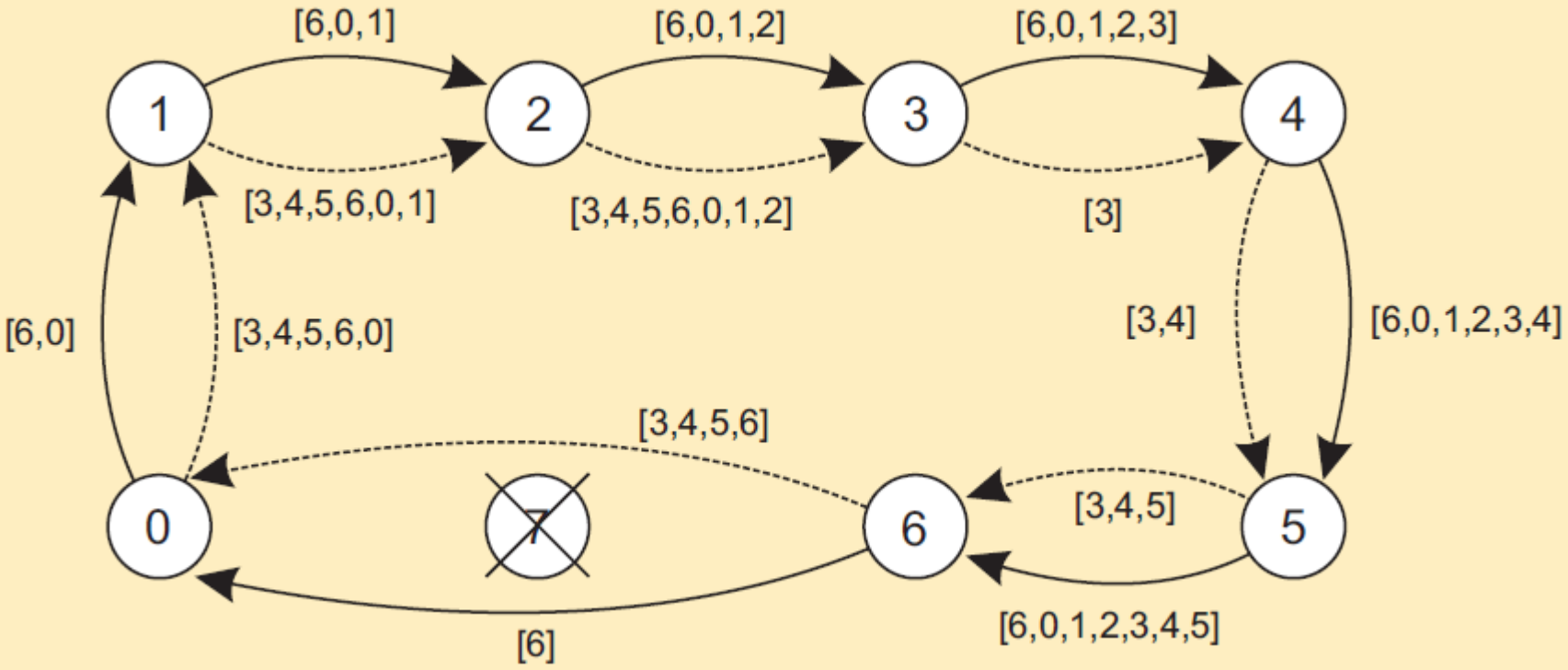
. PRINCÍPIO

Prioridade do processo é obtida através da organização de processos em um anel lógico. Processos com a prioridade mais alta deve ser eleito como coordenador.

- Qualquer processo pode iniciar uma eleição enviando uma mensagem de eleição para seu sucessor. Se o sucessor não está disponível, a mensagem é passada para o próximo sucessor.
- Se uma mensagem é passada a frente, o despachante adiciona ele mesmo a lista. Quando a mensagem volta para o iniciador, todos tiveram a chance de tornarem sua presença conhecida.
- O inicador envia uma mensagem de coordenação ao redor do anel contendo uma lista de processos vivos. O processo com a prioridade mais alta é eleito como coordenador

ELEIÇÃO EM UM ANEL

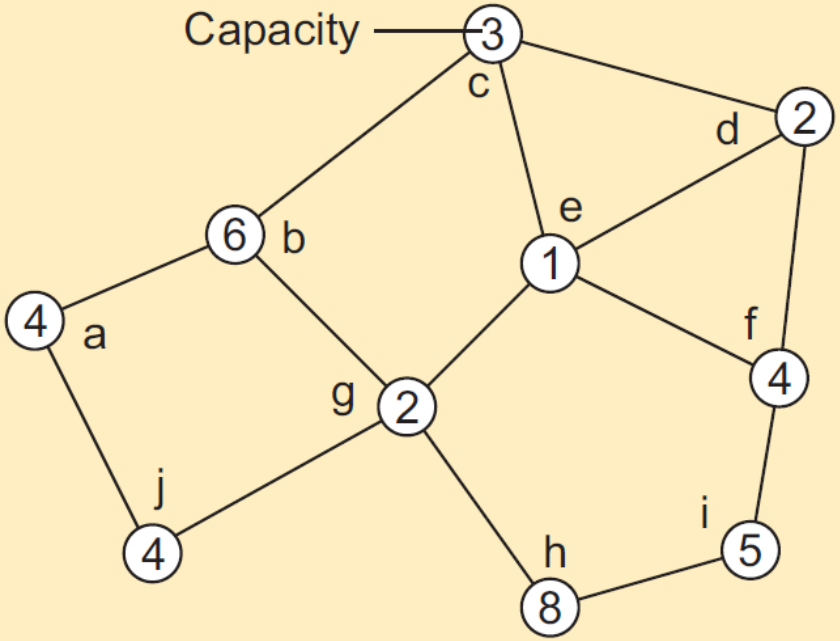
ELEIÇÃO USANDO ALGORITMO DE ANEL

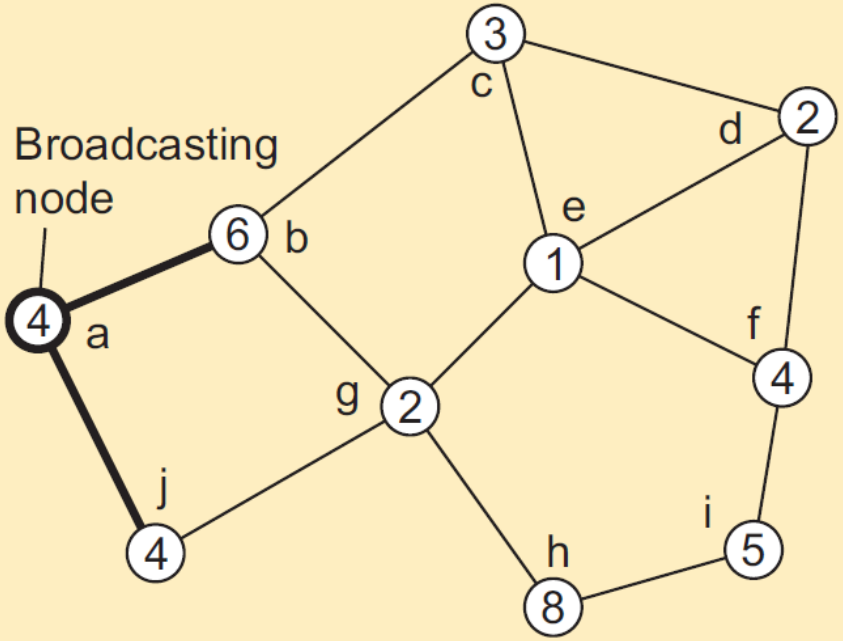


- Linha sólida mostra as mensagens de eleição iniciadas por P6
- A linha pontilhada as mensagens iniciadas por P3

SOLUÇÃO EM REDES SEM FIO

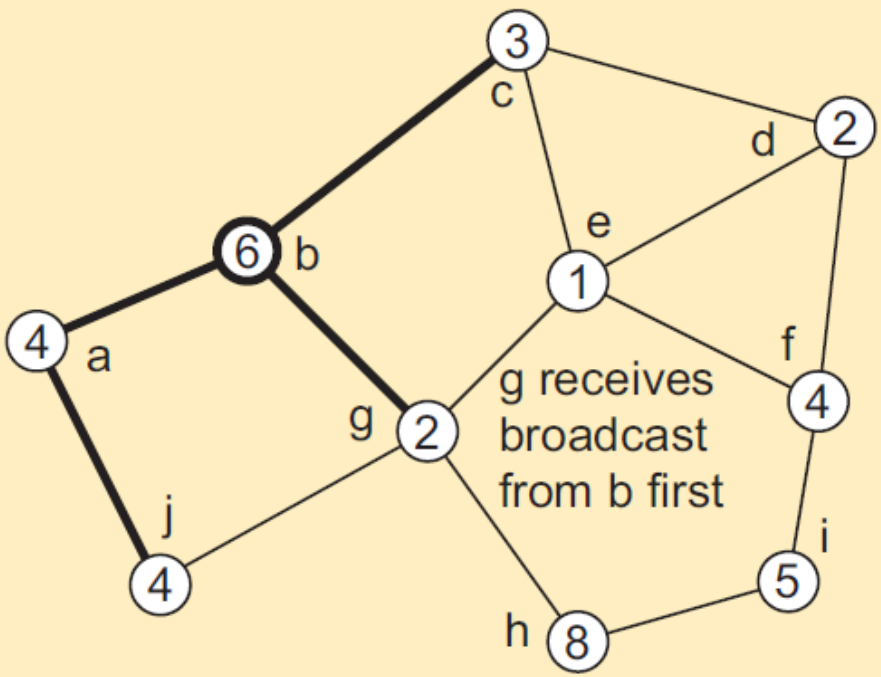
EXEMPLO DE REDE

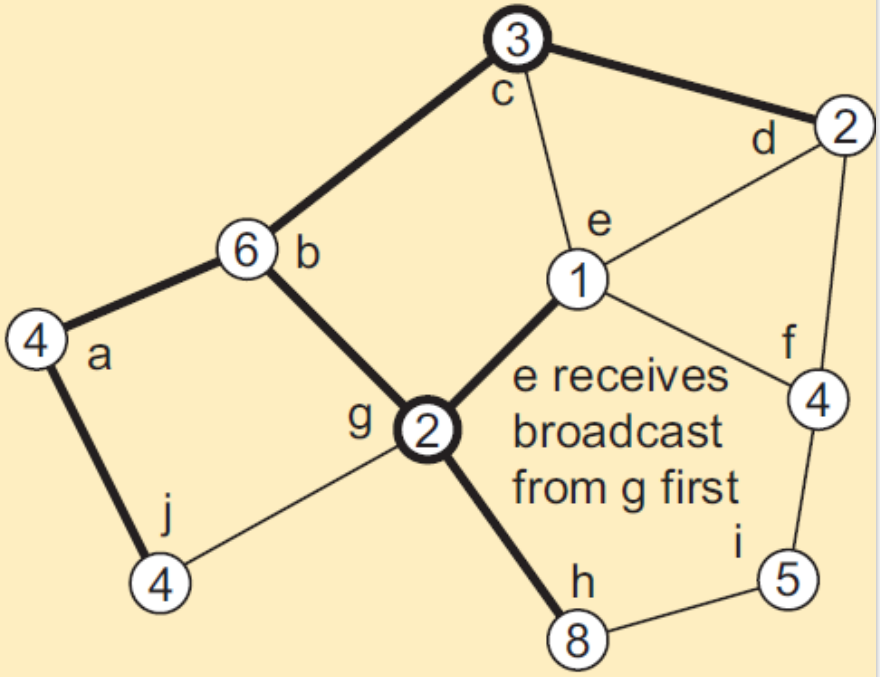




SOLUÇÃO EM REDES SEM FIO

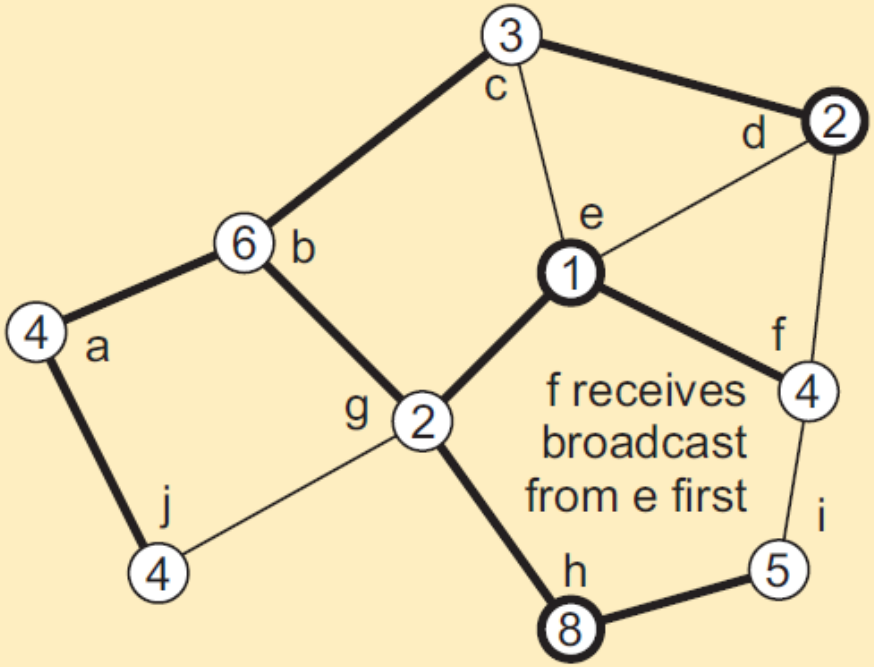
EXEMPLO DE REDE

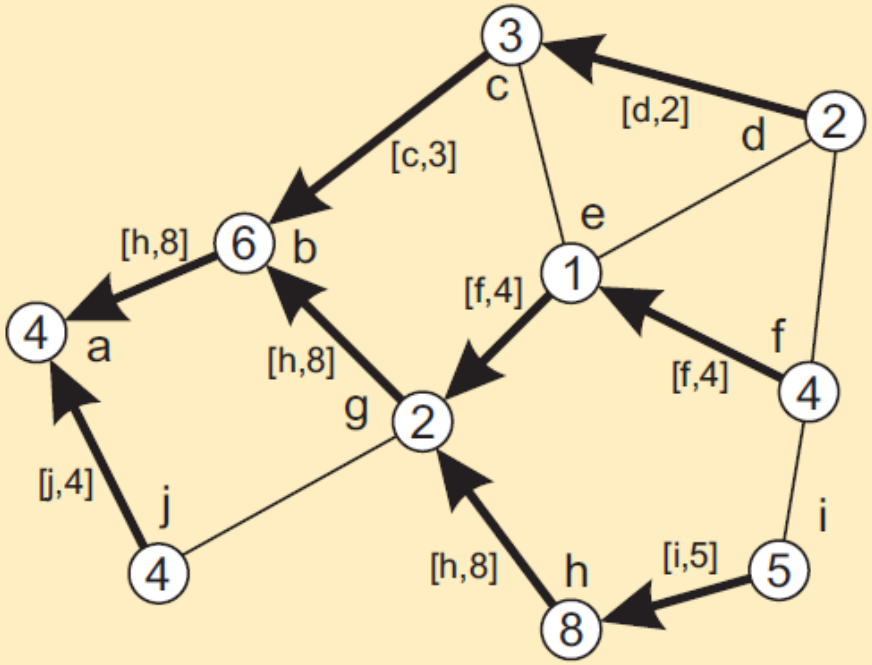




SOLUÇÃO EM REDES SEM FIO

EXEMPLO DE REDE





POSICIONANDO NÓS

QUESTÃO

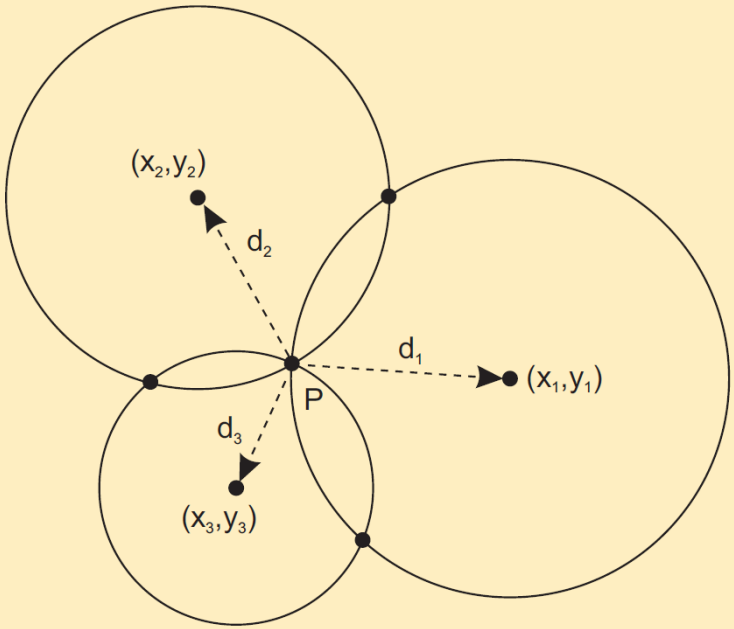
- Em sistemas distribuídos de larga escala no qual nós estão dispersos através de uma rede de larga escala, frequentemente precisamos de uma noção e contabilizar a proximidade ou distância => começa com a determinação (relativa) da localização do nó.

COMPUTANDO POSIÇÃO

OBSERVAÇÃO

- Um nó P precisa de d + 1 posições (*landmarks)* para computar sua própria posição em um espaço d-dimensional. Considere o caso de 2 dimensões

COMPUTANDO POSIÇÃO 2D



SOLUÇÃO

P precisa resolver 3 equações para dois desconhecidos (x_p , y_p)

$$d_i = \sqrt{(x_i - x_P)^2 + (y_i - y_P)^2}$$

GPS GLOBAL POSITIONING SYSTEM

ASSUMINDO QUE OS RELÓGIOS DE SATÉLITES SÃO ACURADOS E SINCRONIZADOS

- Leva um tempo até o sinal alcançar o receptor
- O relógio do receptor está definitivamente for ade sincronismo com o satélite

BÁSICO

- Δr : desvio desconhecido do relógio do receptor
- Xr, Yr, Zr : coordenadas desconhecidas de um receptor
- Ti : timestamp em uma mensagem vinda de um satélite i
- Δi = (Tnow – Ti) + Δr : atraso medido de uma mensagem enviada por um satélite i
- Distância medida até o satélite i: c x Δi (c é a velocidade da luz)
- Distância real: di = cΔi – cΔr = $\sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$

OBSERVAÇÃO

- 4 atélites => 4 equações e 4 desconhecidos (com Δr como um deles)

SERVIÇOS DE LOCALIZAÇÃO BASEADO EM WIFI

IDEIA BÁSICA

- Assuma que temos uma base de dados de pontos de acesso (APs) conhecidos e suas coordenadas
- Assuma que podemos estimar as distâncias para um AP
- Então: com 3 pontos de acesso detetados, podemos computar a posição

COMO DETECTAR PONTOS DE ACESSO ?

- Use um dispositivo com WIFI ativado junto com GPS e ande através de uma área enquanto registra pontos de acesso observados
- Compute a posição CentroID: assuma que um ponto de acesso AP foi detetado em N diferentes localidades $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$, com localização GPS conhecida.
- Compute a localização de AP como sendo $\vec{x}_{AP} = \frac{\sum_{i=1}^{N} \vec{x}_i}{N}$.

PROBLEMAS

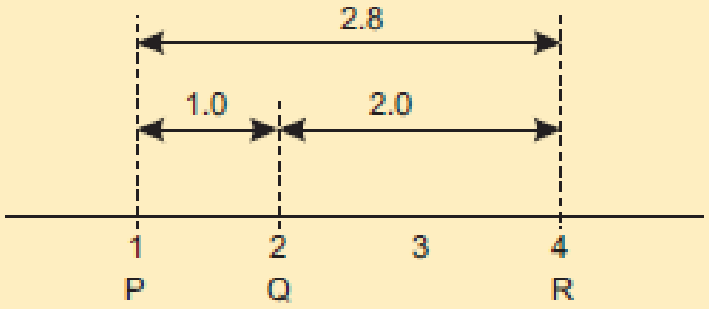
- Acuracidade limitada de cada ponto GPS detectado $\vec{x}_i$
- Um ponto de acesso tem um campo de transmissão não uniforme
- Número de pontos de amostragem detetados N pode ser muito baixo.

COMPUTANDO POSIÇÃO

PROBLEMAS

- Latências medidas para *landmarks* flutuam
- Distâncias computadas não são nada consistentes

DISTÂNCIAS INCOSISTENTES NO ESPAÇO 1D



SOLUÇÃO: MINIMIZAR ERROS

- Use N *landmark* nodes especiais $L_1, \ldots, L_N$
- Landmarks medem suas latências em pares $\tilde{d}(L_i, L_j)$
- Um nó central computa as coordenadas de cada landmark minimizando:

$$\sum_{i=1}^{N} \sum_{j=i+1}^{N} \left(\frac{\tilde{d}(L_i, L_j) - \hat{d}(L_i, L_j)}{\tilde{d}(L_i, L_j)} \right)^2$$

Onde $\hat{d}(L_i, L_j)$ é a distância depois que os nós *Li* e *Lj* foram posicionados

COMPUTANDO POSIÇÃO

ESCOLHENDO A DIMENSÃO m

- O parâmetro escondido na dimensão *m* com $N > m$. Um nó *P* mede sua distância para cada um dos *N landmarks* e computa sua coordenada minimizando

$$\sum_{i=1}^{N} \left(\frac{\tilde{d}(L_i, P) - \hat{d}(L_i, P)}{\tilde{d}(L_i, P)} \right)^2$$

OBSERVAÇÃO

- A prática mostra que *m* pode ser tão pequeno quanto 6 ou 7 para estimativas de latências dentro de um fator 2 do valor atual

VIVAL DI

PRINCÍPIO: REDE DE MOLAS IMPONDO FORÇAS

- Considere uma coleção de *N* nós *P1, ... , PN* com *Pi* tendo coordenadas $\vec{x}_i$. Dois nós exercem força mútua:

$$\vec{F}_{ij} = (\tilde{d}(P_i, P_j) - \hat{d}(P_i, P_j)) \times u(\vec{x}_i - \vec{x}_j)$$

- Com $u(\vec{x}_i - \vec{x}_j)$ é o vetor unidade na direção de $\vec{x}_i - \vec{x}_j$

NÓ Pi REPETIDAMENTE EXECUTA OS PASSOS

1. Measure the latency d_{ij} to node P_j, and also receive P_j's coordinates $\vec{x}_j$.
2. Compute the error $e = \tilde{d}(P_i, P_j) - \hat{d}(P_i, P_j)$
3. Compute the direction $\vec{u} = u(\vec{x}_i - \vec{x}_j)$.
4. Compute the force vector $F_{ij} = e \cdot \vec{u}$
5. Adjust own position by moving along the force vector: $\vec{x}_i \leftarrow \vec{x}_i + \delta \cdot \vec{u}$.

APLICAÇÕES EXEMPLO

APLICAÇÕES TÍPICAS

- Disseminação de dados: talvez a aplicação mais importante. Note que existe muitas variantes de disseminação.
- Agregação: se cada nó Pi mantém uma variável vi. Quando dois nós criam rumor, cada um reseta a variável para:

$$v_i, v_j \leftarrow (v_i + v_j)/2$$

- Resultado: no final, cada nó terá computado a média $\bar{v} = \sum_i v_i/N$.
- O que acontece no caso de inicialmente vi = 1 e vj = 0, j ≠ i ?