Tolerância a falhas em sistemas distribuídos

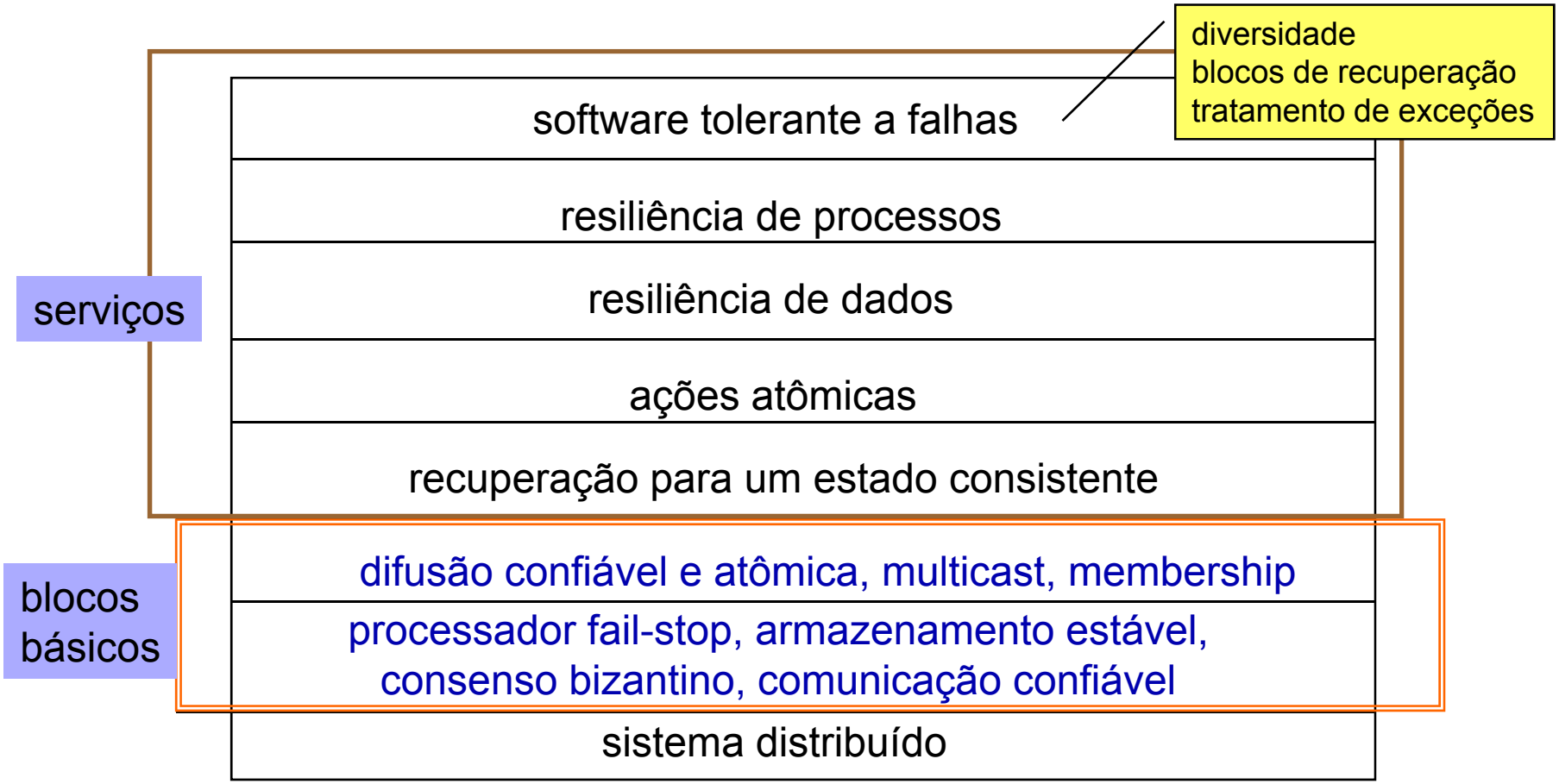
UFRGS
Taisy Silva Weber

Redundância implícita

- sistemas distribuídos são naturalmente redundantes
  - mas redundância sozinha não aumenta a confiabilidade do sistema

Níveis

JALOTE, P. Fault tolerance in distributed systems. Prentice Hall, Englewood Cliffs, New Jersey, 1994



Nesta disciplina

- ✓ sistemas distribuídos
- ✓ processador fail-stop
- ✓ armazenamento estável
- ✓ consenso bizantino
- ✓ comunicação confiável
- ✓ difusão confiável

blocos básicos

- ✓ recuperação para um estado consistente
- ✓ replicação de dados

serviços

Exemplos de sistemas distribuídos

- sistemas de pequena escala
  - LANs
  - clusters
- sistemas de grande escala (ou larga escala)
  - Internet
  - Grids
  - nuvens
- propriedades fortes de tolerância a falhas não escalam adequadamente
  - custo muito alto para manter consenso, consistência de réplicas e estado global distribuído em sistemas de larga escala sujeitos a falhas

clusters, grids, clouds

	Clusters	**Grids**	**Clouds**
Size/ scalability	centenas	milhares	centenas a milhares
OS	Linux, Windows	Unix	VM e múltiplas OSs
Ownership	único	múltiplo	único
Inter-connection	Dedicated, high-end with low latency and high bandwidth	Mostly Internet with high latency and low bandwidth	Dedicated, high-end with low latency and high bandwidth
Security/ privacy	login/password Medium level of privacy.	Public/private key pair based authentication and mapping a user to an account. Limited support for privacy.	Each user/application is provided with a virtual machine. High security/privacy.
Discovery	Membership services	Centralised indexing and decentralised info services	Membership services

clusters, grids, clouds

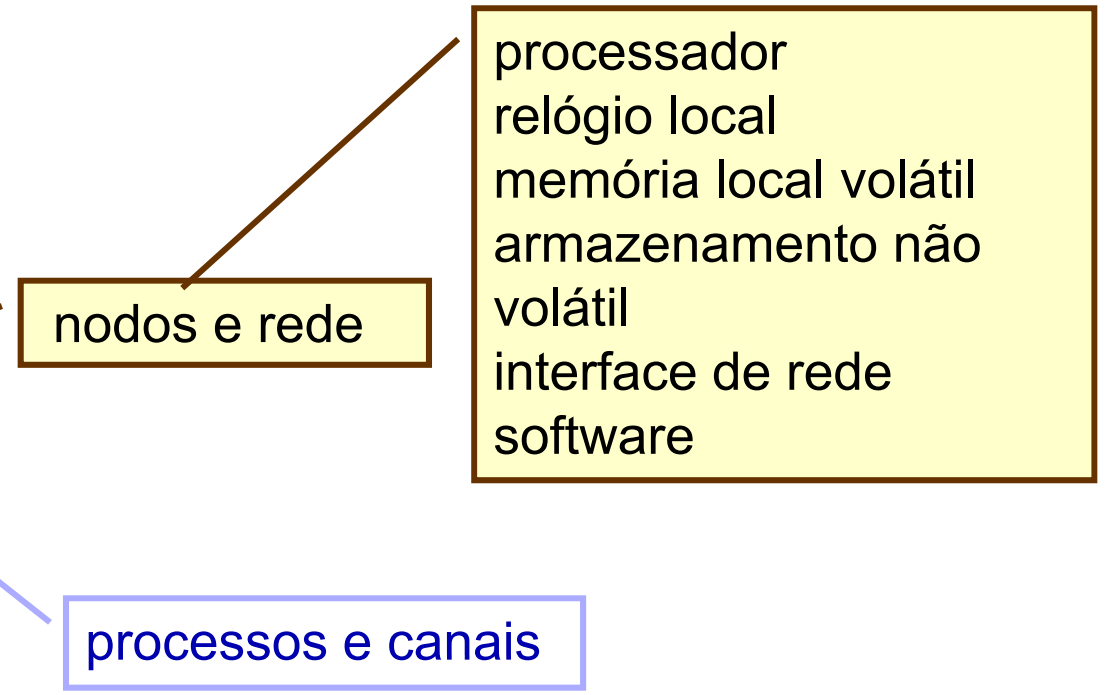
	Clusters	**Grids**	**Clouds**
Resource management	Centralized	Distributed	Centralized/ Distributed
Standards/ inter-operability	Virtual Interface Architecture	Some Open Grid Forum standards	Web Services (SOAP and REST)
Capacity	Stable and guaranteed	Varies, but high	Provisioned on demand
Failure management (Self-healing)	Limited (often failed tasks/ applications are restarted).	Limited (often failed tasks/ applications are restarted).	Strong support for failover and content replication. VMs can be migrated from one node to other

Buyya, Rajkumar, Chee Shin Yeo, Srikumar Venugopal, James Broberg, e Ivona Brandic. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility". *Future Generation Computer Systems* 25, nº. 6 (junho 2009): 599-616.

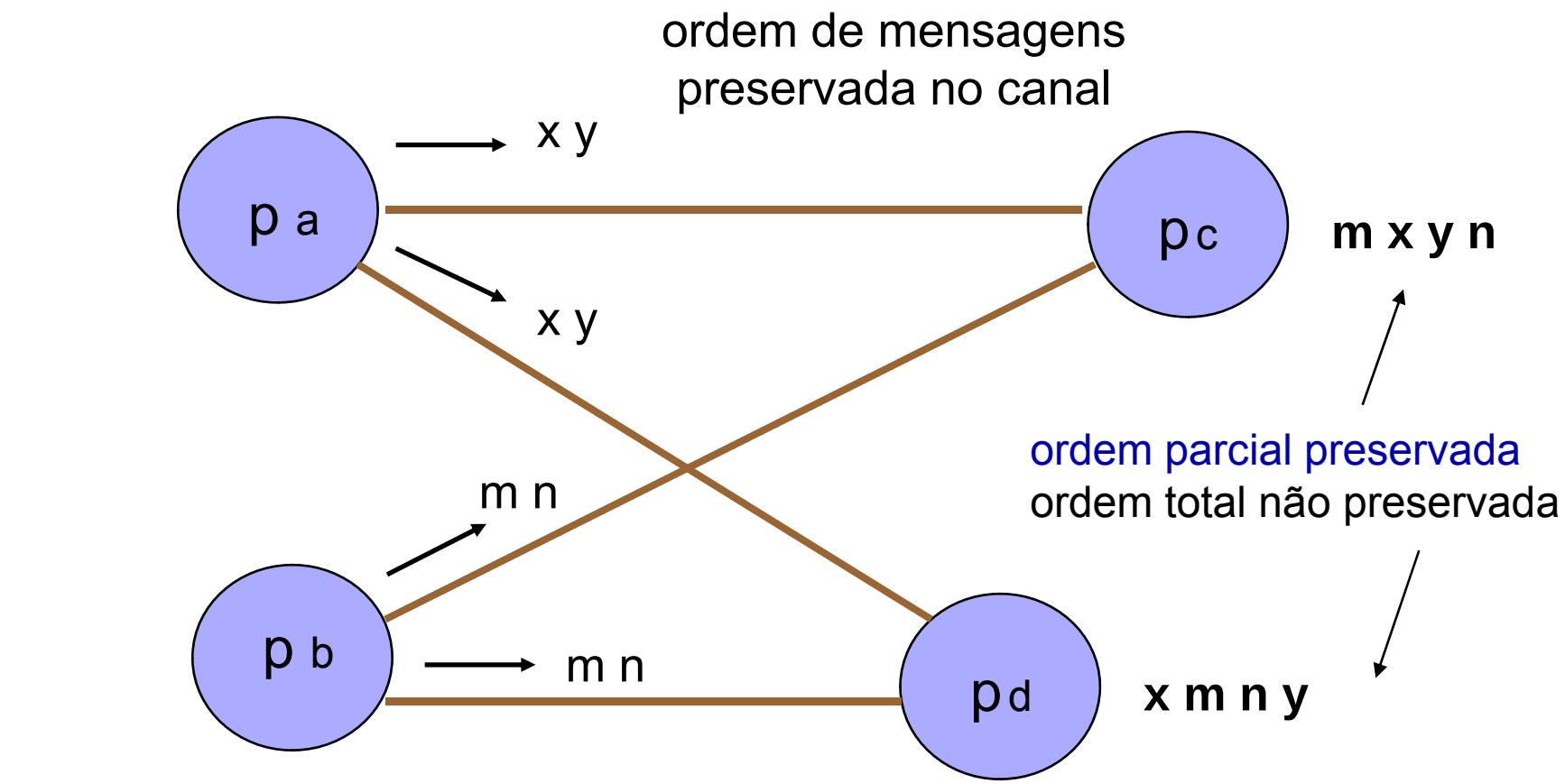
Sistemas distribuídos

- ✓ sem memória compartilhada
- ✓ sem relógio global

- ✓ modelos
  - ✓ modelo físico
  - ✓ modelo lógico



Canal



Modelo lógico

Modelos de tempo

- ✓ sistema assíncrono
  - ✓ não existem limites de tempo

- ✓ sistema síncrono
  - ✓ existe um limite de tempo finito e conhecido

sistema correto opera dentro desse limite

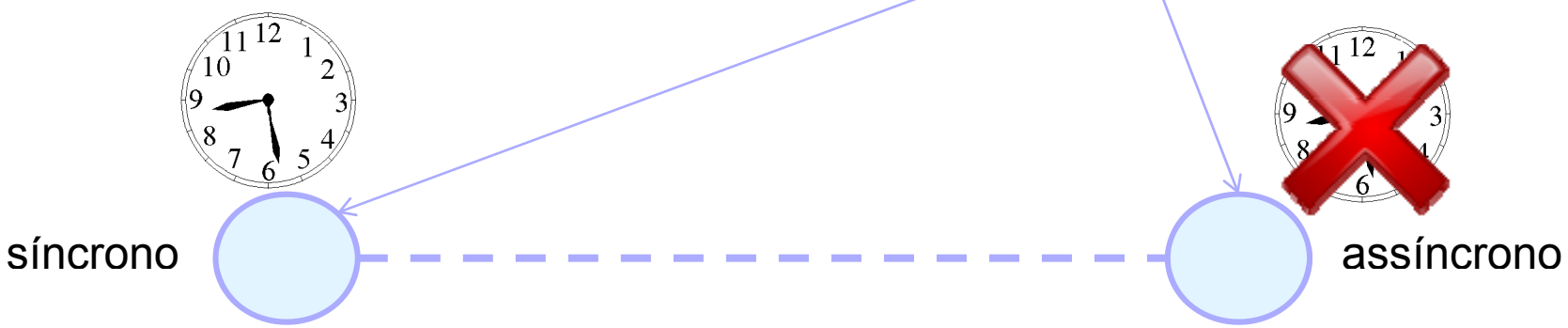
  - ✓ falha de componente pode ser deduzida pela ausência de resposta
  - ✓ timeout

detecção de defeitos em nodos e perdas de mensagens

Modelos de tempo

- ✓ sistema assíncrono
- ✓ sistema síncrono

modelos síncrono e assíncrono podem ser considerados extremos teóricos



outros modelos misturam características desses extremos

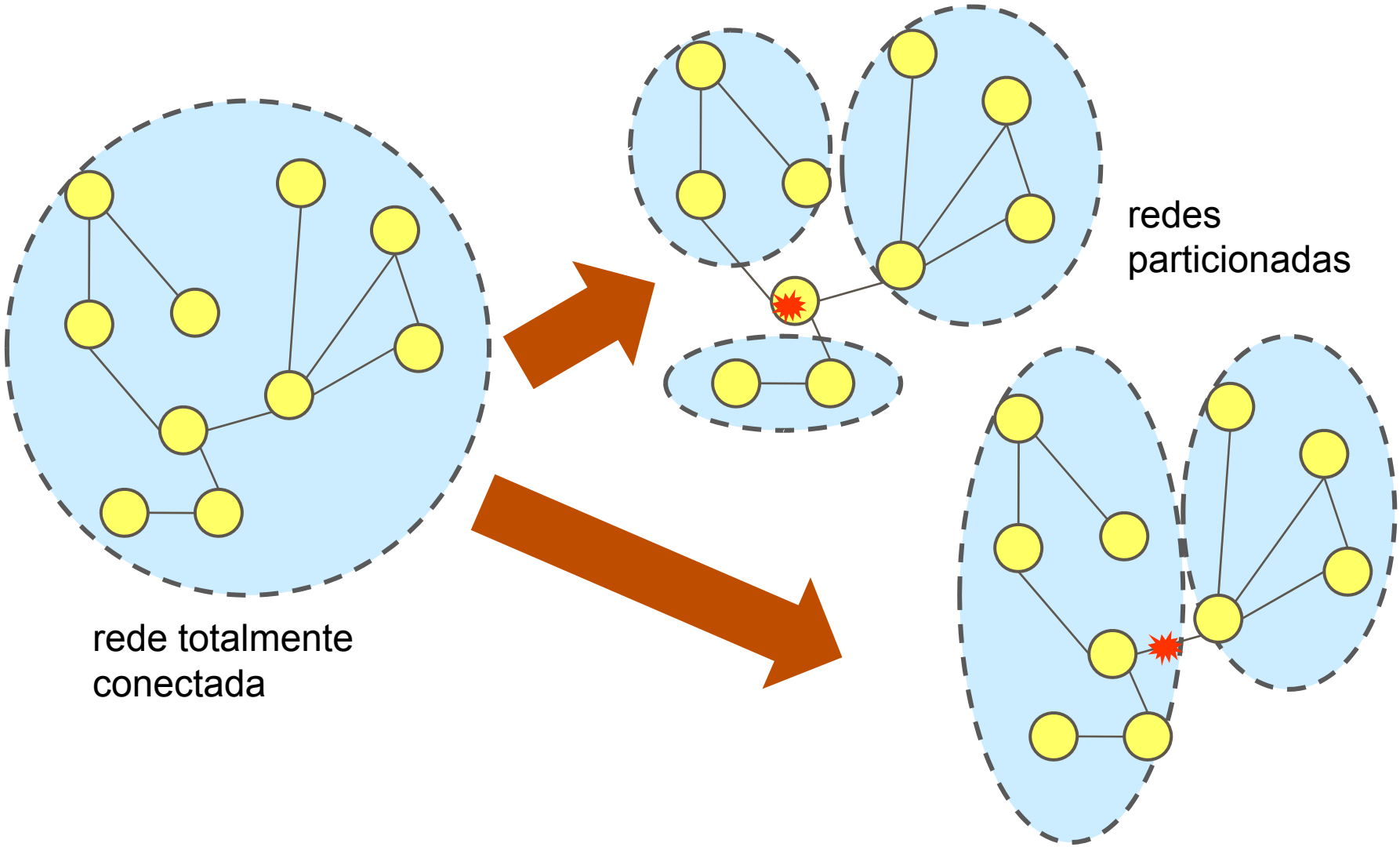
Classificação de falhas (Cristian)

Exemplos de falhas

- processador:
  - crash ou bizantinas
- rede de comunicação:
  - todos os tipos
- clock:
  - temporização ou bizantinas
- meio de armazenamento
  - temporização, omissão ou resposta
- software:
  - resposta

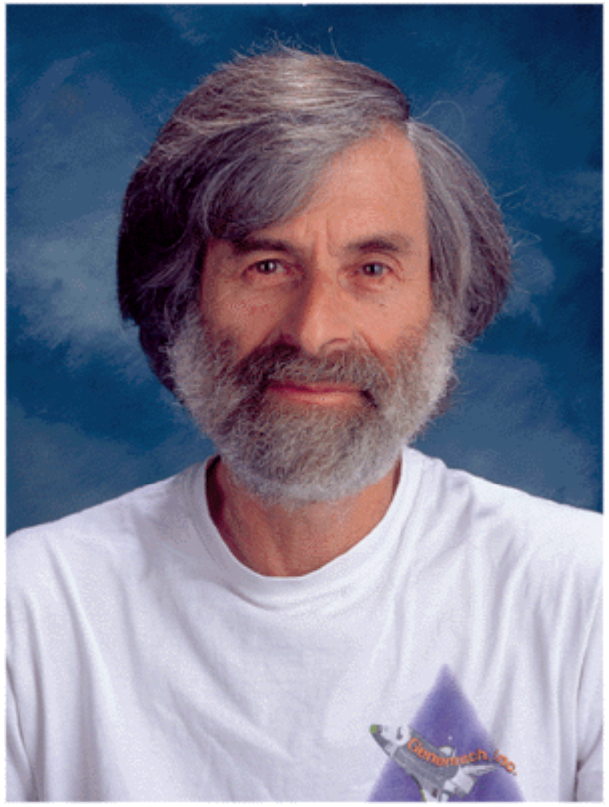
O modelo de falhas é uma simplificação da realidade.Na literatura aparecem vários outros modelos de falhas para sistemas distribuídos. Alguns incluem particionamento de rede.

Particionamento



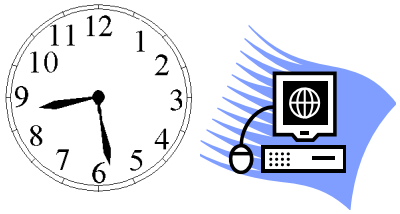
Lamport

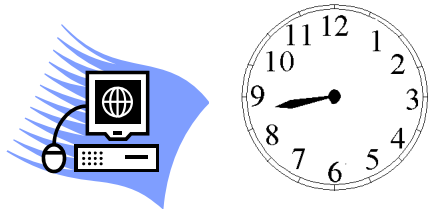
- ✓ clocks lógicos
- ✓ generais bizantinos
- ✓ LaTeX
- ✓ Paxos (consenso)



Ordenação de eventos

- determinar a ordem de eventos que ocorrem em nodos diferentes, medidos por relógios diferentes



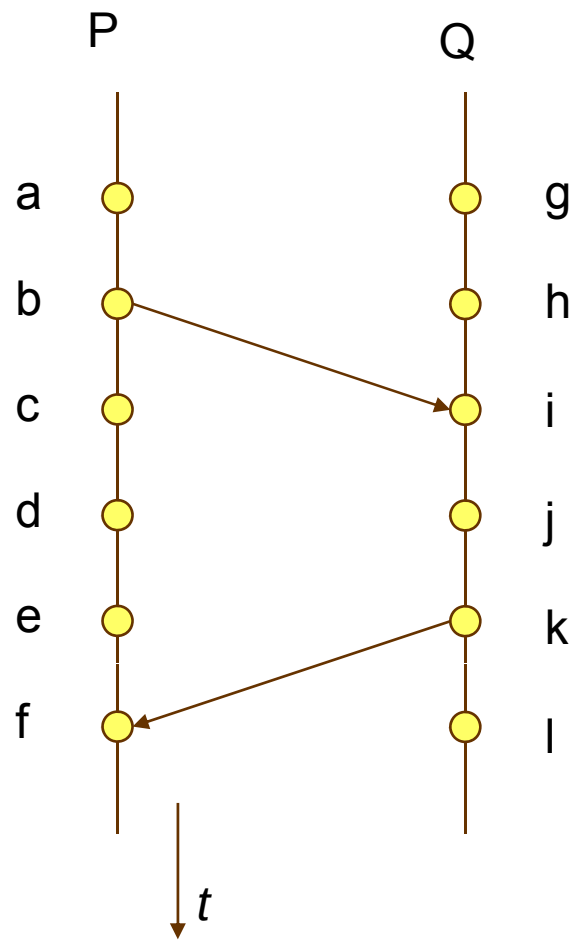


- relação:
  - a "aconteceu antes de" b : $a \rightarrow b$

ordem de ordem parcial

nem todos os eventos podem ser ordenados

Ordenação de eventos



$a \rightarrow b$
$b \rightarrow i$
$h \rightarrow i$
$a \rightarrow b$ e $b \rightarrow i$ então $a \rightarrow i$
nem $a \rightarrow h$, nem $h \rightarrow a$

- ✓ se **a** e **b** são eventos do mesmo processo e **a** é executado antes de **b** então **a** $\rightarrow$ **b**
- ✓ se a é **send** e b é **receive** da mesma msg então **a** $\rightarrow$ **b**
- ✓ $a \rightarrow b$ e $b \rightarrow c$ então $a \rightarrow c$
- ✓ eventos concorrentes: nem $a \rightarrow b$, nem $b \rightarrow a$

Ordenação de eventos

- sistema de clock lógico
  - um sistema de clock lógico é correto se é consistente com a relação $\rightarrow$

exemplo: timestamp T

  - clock lógico carimba um evento de forma que a relação de ordem parcial é mantida

é possível estabelecer uma ordem total não temporal com relógios lógicos

Concordância bizantina

- consenso
  - problema recorrente em sistemas distribuídos
    - solução trivial para sistemas livres de falhas (ou seja perfeitos)
    - não trivial para sistemas com falhas arbitrárias e assíncronos

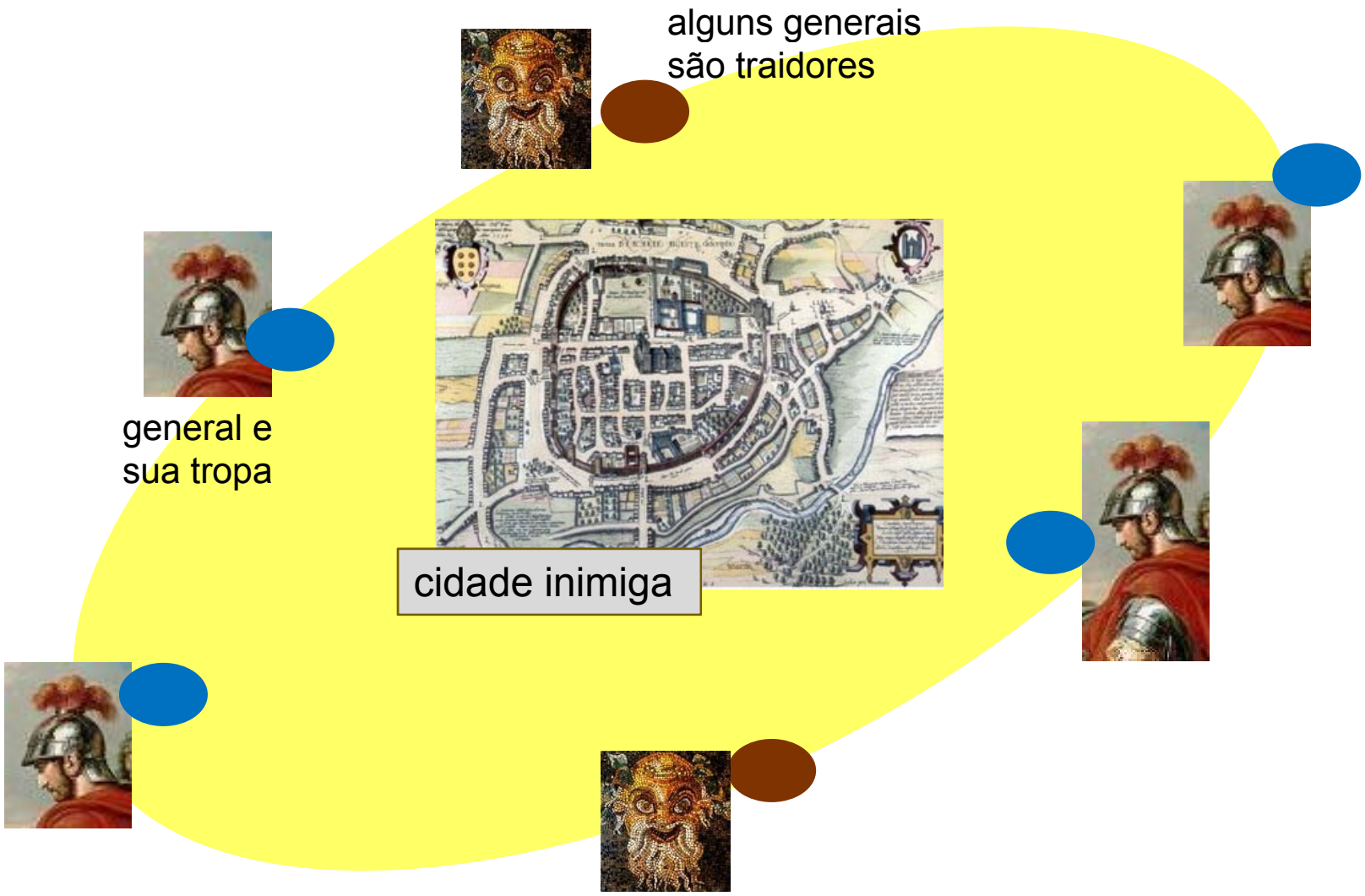
Israel Koren, C. Mani Krishna - *Fault-Tolerant Systems* - Morgan Kaufmann, 2007
pg 42

Concordância bizantina

- alcançar **consenso** na presença de traidores
  - defeitos bizantinos
    - comportamento arbitrário
  - nodo pode enviar informações diferentes para os diferentes componentes com quem se comunica
- problema dos generais bizantinos

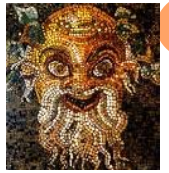
Lamport, Shostak e Pease, 82

Generais bizantinos



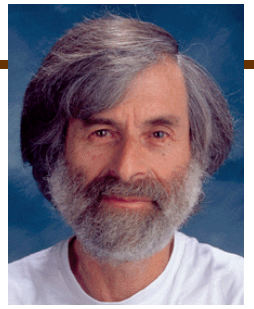
Generais bizantinos

traidores não podem provocar divisão (alguns atacam e outros recuam)



não traidores devem chegar a consenso sobre atacar ou recuar

Algoritmos de Lamport



- ✓ Lamport 82 (Lamport, Shostak e Pease)
  - ✓ solução para:
    - ✓ sistemas síncronos
    - ✓ totalmente conectado
    - ✓ dois algoritmos :
      - ✓ msgs orais
      - ✓ msg assinadas

Orais: comuns, sem assinatura (o emissor é conhecido, impossível verificar a integridade do conteúdo, um nodo traidor pode alterar o conteúdo)

  - ✓ relação entre traidores (m) e não traidores (n-m)

mensagens orais: $n \geq 3m + 1$
mensagens assinadas: $n \geq m + 2$

    - ✓ grande número de rodadas: $m + 1$
    - ✓ grande número de mensagens: $O(n^m)$

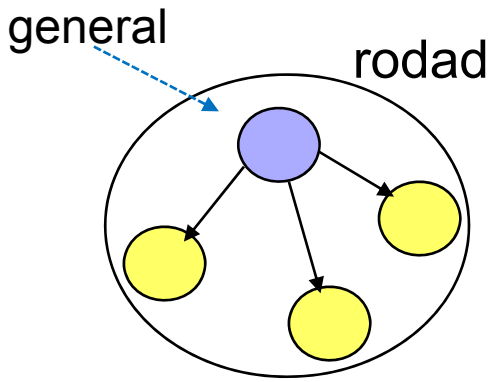
Algoritmo para mensagens orais

consistência interativa (ICA)

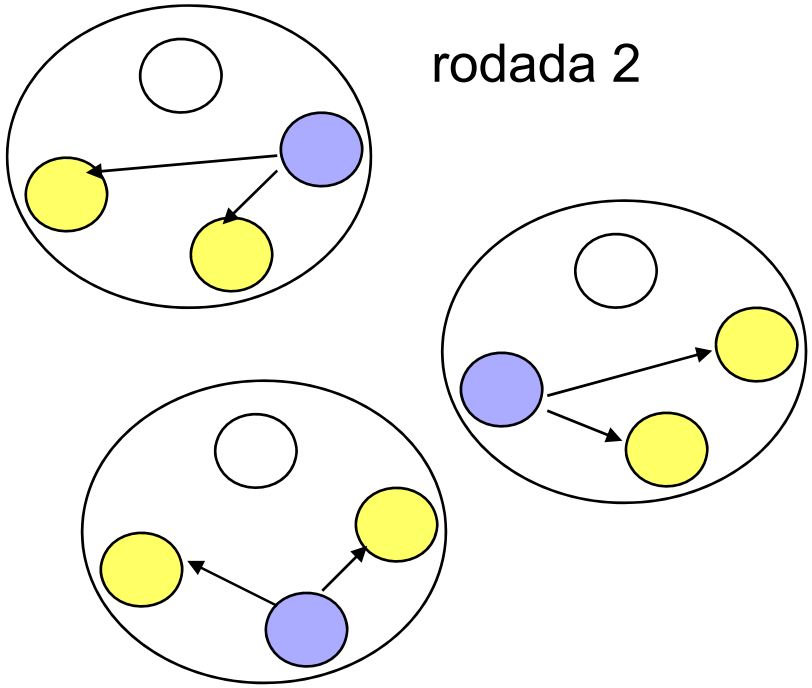
- ICA(0)
  - o general envia o seu valor para os outros n -1 nodos
  - cada nodo usa valor recebido, ou default (se nada recebeu)
- (fim da recursão)
- ICA(m), m>0
  - o general envia o seu valor para os outros n -1 nodos
  - nodo i: v(i) (valor recebido pelo nodo i ou default),
    - nodo i atua como general em ICA(m-1) enviando v(i) para os demais n -2 nodos (confirmação do valor)
  - para cada nodo i:
    - v(j) valor recebido do nodo j (j≠i)
    - nodo i usa valor *maioria*(v(1), ..., v(n-1))

ICA(m)

ICA(m) deve ser usado por todos os nodos para alcançar consenso



usando ICA(m) em cada nodo, cada nodo do sistema terá o mesmo valor assumido para todos os outros nodos



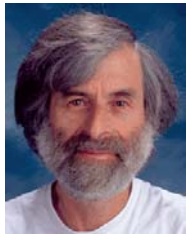
ICA(1)

no máximo
um traidor

toda mensagem enviada é recebida corretamente;
o receptor sabe quem enviou a mensagem
a ausência de uma mensagem pode ser detectada (time-out)

Concordância bizantina

- algoritmos de Lamport
  - ótimos em relação ao número de rodadas
  - mas exigem número exponencial de mensagens



- outros algoritmos foram propostos
  - alguns mais eficientes em relação ao número de mensagens
  - alguns suportam sistemas não totalmente conectados
  - alguns suportam sistemas assíncronos através de procedimentos randômicos
    - comunicação epidêmica:
      - garante certa probabilidade de todos os nodos recebem uma mensagem difundida

Impossibilidade de consenso

- consenso é impossível em sistemas assíncronos (Fischer, 1985) - conhecido como **problema FLP**
  - mesmo com um único traidor
  - mesmo quando só ocorrem falhas de crash

> Lamport não chegou a provar para qualquer falha, apenas para falhas bizantinas

  - entretanto o problema pode ser solúvel se alguma forma fraca de sincronismo for introduzido (Dolev 1987; Dwork 1988; Chandra e Toueg, 1996)
  - esforços foram direcionados para detectores de falhas não confiáveis (Chandra e Toueg)

Paxos

- Lamport 1988
  - ilha grega Paxos
    - sistema de consenso ficcional
- consenso
  - sistemas assíncronos
  - não garante progresso (FLP)
  - mas garante *safety* (livre de inconsistências)
- usado em sistemas de réplicas de arquivos e banco de dados

usado pelo google, Microsoft, IBM e outras empresas

Armazenamento estável

essencial em vários esquemas de suporte a tolerância a falhas

- parte do estado do sistema permanece disponível mesmo após defeito do sistema
- conteúdo é preservado apesar de falhas
  - um disco magnético **não** é armazenamento estável
- exemplos de implementação:
  - sombreamento de disco
    - imagens idênticas em dispositivos separados
    - 2 discos - espelhamento

Tandem

  - RAID: **R**edundant **A**rray of **I**nexpensive **D**isks

I pode ser também Independent

Exemplo de implementação

- **R**edundant **A**rray of **I**nexpensive **D**isks — 1987
  - propostos inicialmente para diminuir **custos** de armazenamento e prover alta **velocidade**
    - **atualmente:** confiabilidade e desempenho

- bit interleaving: entrelaçamento de bits
  - perda de um disco pode comprometer todo o array
  - colocando mais discos ou mais código de recuperação de erros esse problema pode ser contornado

http://en.wikipedia.org/wiki/RAID

RAID

- RAID 0: sem redundância
- RAID 1
  - dois discos idênticos espelhados
  - método mais caro (100% redundância)
- RAID 2
  - bits entrelaçados + palavra código
    - número de discos depende do algoritmo de correção de erros
- RAID 3
  - bytes entrelaçados + disco extra para paridade
  - cada byte sequencial está num disco diferente
- RAID 4
  - como RAID 3, mas com **setores** entrelaçados
  - vantagem para o SO
  - paridade em disco extra

RAID 5 e 6

existem combinações

- RAID 5 — o mais popular
  - como RAID 3 mas sem disco de paridade
    - **paridade** é **distribuída** pelos discos do sistema
  - pode ser implementado a partir de dois discos

além da paridade distribuída

- RAID 6
  - como RAID 5, mas com **mais um** disco de paridade
    - dois discos podem falhar sem perda de dados
    - pode trocar drive defeituoso com sistema em operação
  - degrada para RAID 5 quando 1 disco está defeituoso

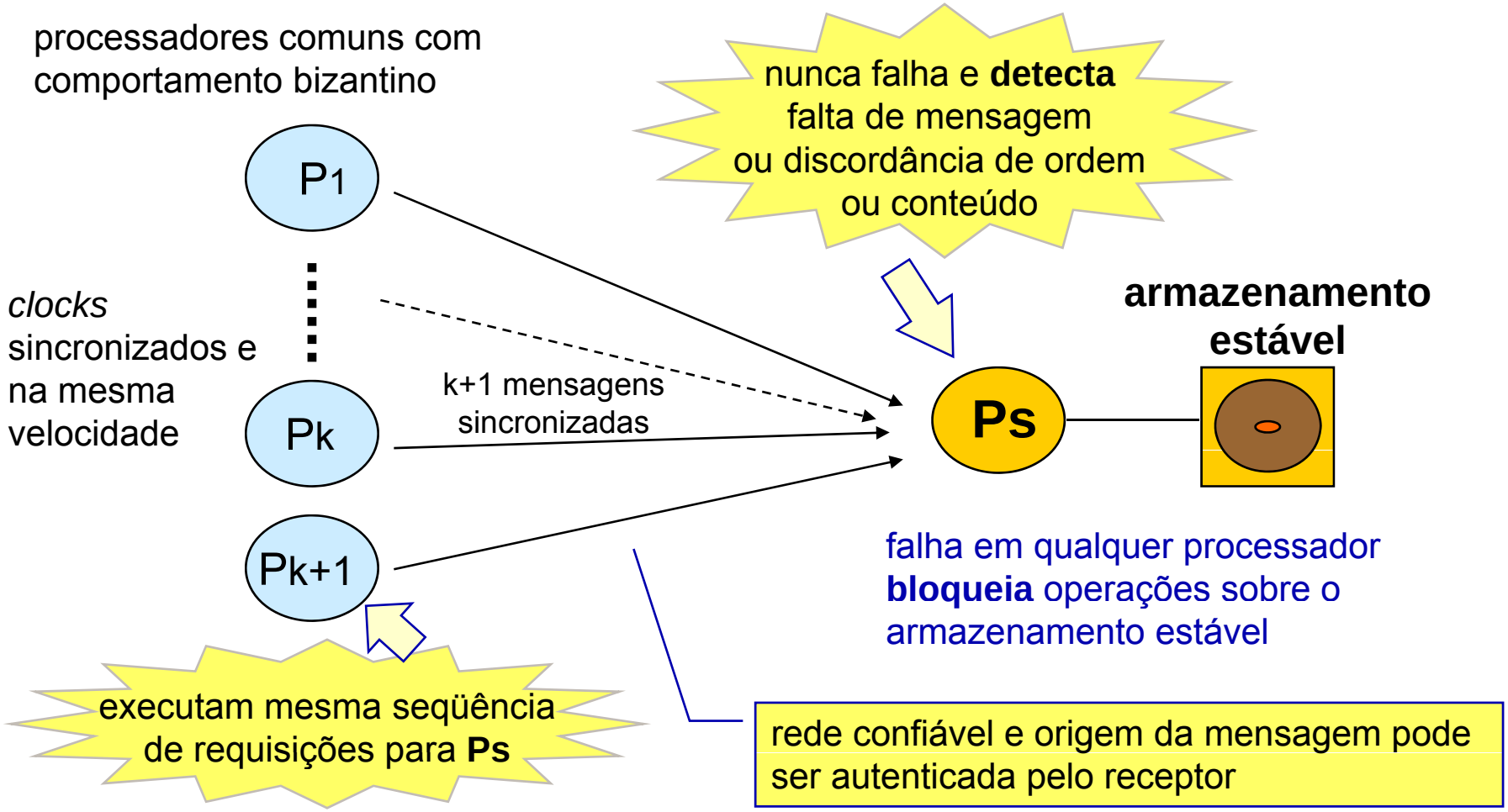
Processadores fail-stop

- em caso de defeito, nodo cessa operação sem realizar qualquer ação incorreta
  - comportamento **fail-stop** assumido por grande parte dos esquemas de TF
- processadores reais não são por natureza **fail-stop**
  - processadores reais com defeito se comportam de maneira **arbitrária**
  - nodos aproximadamente **fail-stop** podem ser construídos a partir de processadores reais (**k fail-stop**)

exemplo: grande parte dos servidores tolerantes a falhas

k fail-stop

comporta-se como um processador fail-stop a menos que **k+1 ou mais** componentes falhem



Tipos de difusão

- broadcast
  - envio de mensagens a **todos** os nodos do sistema
- multicast
  - envio de mensagens a **alguns** nodos do sistema
- sensível a falhas de nodo e comunicação

multicast envolve o conceito de grupo

em broadcast ou multicast sobre comunicação **ponto a ponto**: um nodo pode falhar após ter iniciado difusão, assim alguns nodos podem ter recebido a mensagem e outros não

também existem problemas com redes de broadcast e multicast não confiável

Propriedades na difusão

- ✓ confiabilidade

  valem tanto para broadcast como multicast

  - ✓ mensagem deve ser recebida por todos os nodos operacionais

- ✓ ordenamento consistente
  - ✓ diferentes mensagens enviadas para nodos diferentes são entregues na mesma ordem em todos os nodos

  ordenamento consistente é diferente de ordenamento temporal

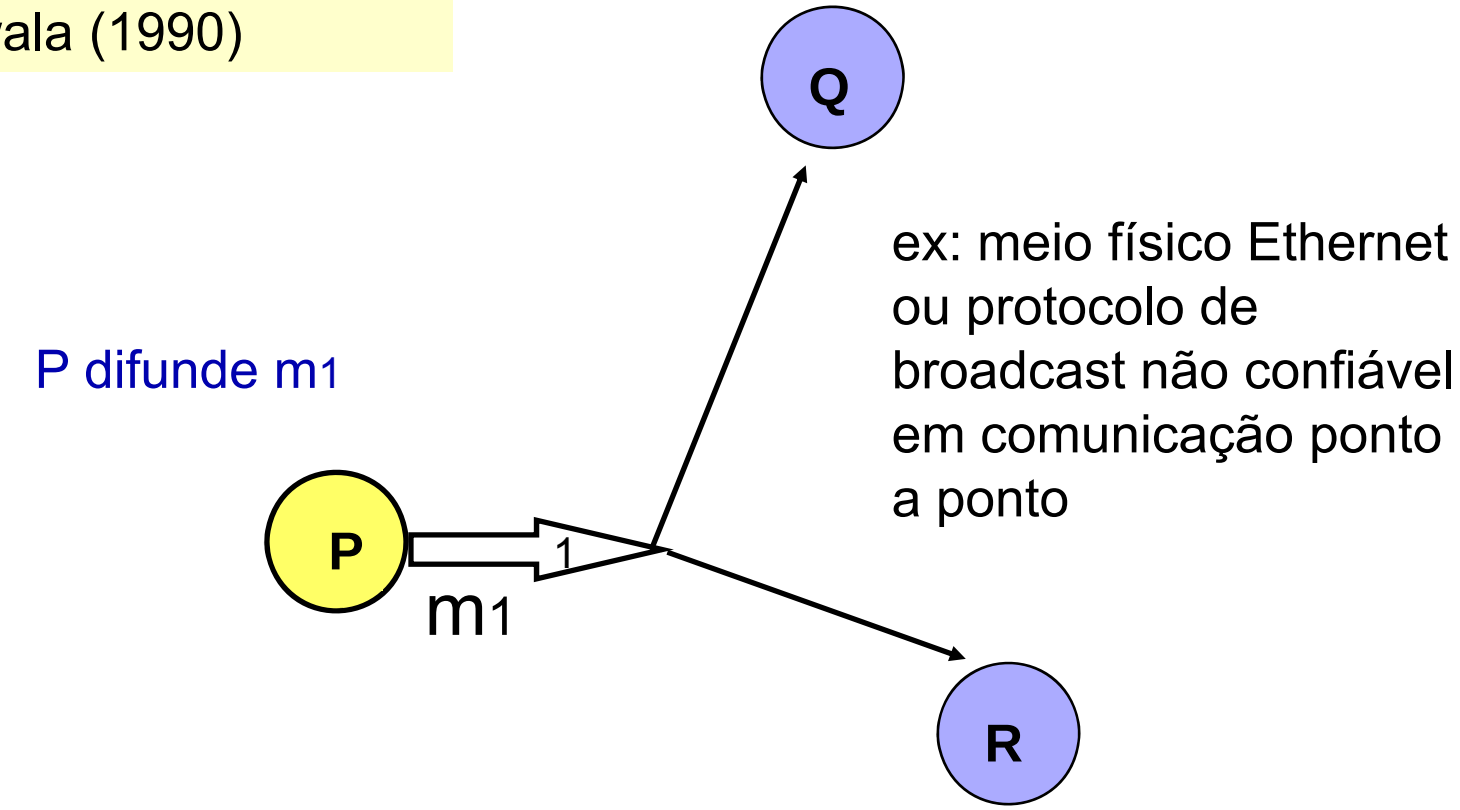
- ✓ preservação de causalidade
  - ✓ a ordem na qual mensagens são entregues é consistente com a relação causal de envio das mensagens

  mensagens sem relação causal poderiam ser entregues em qualquer ordem

Trans

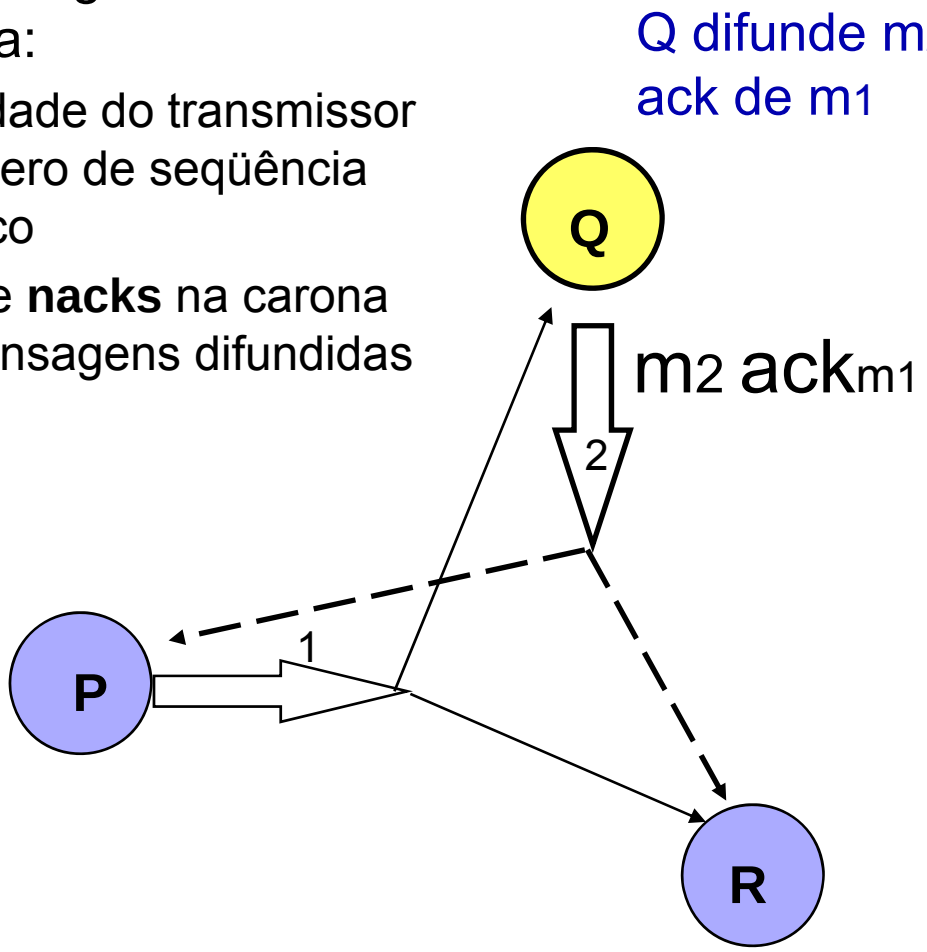
- ✓ primitiva confiável baseada em broadcast não confiável

Melliar-Smith, Moser e Agrawala (1990)



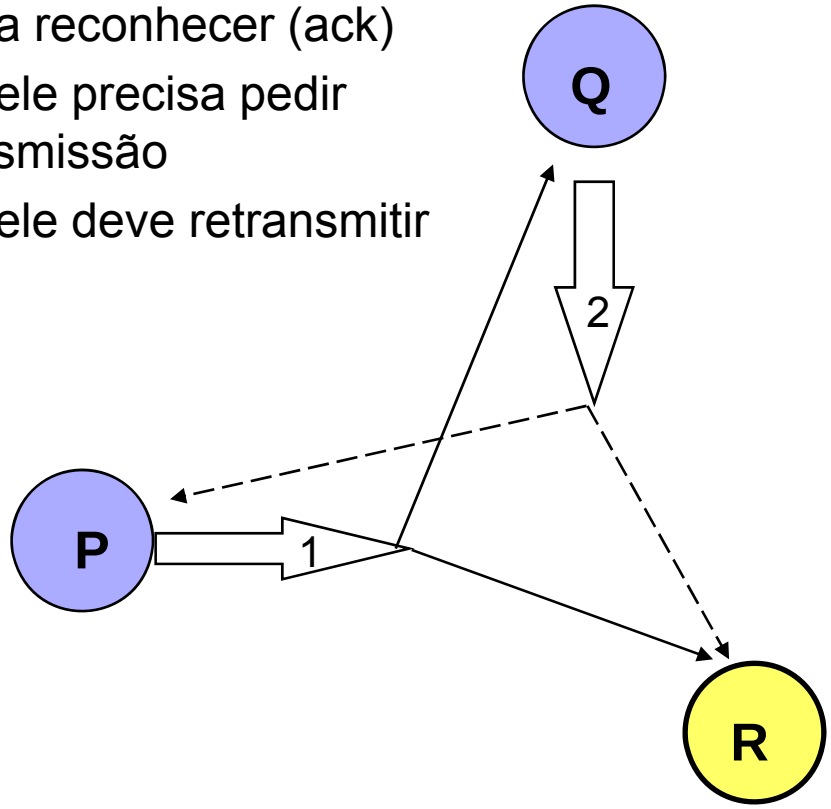
Trans

- cada mensagem transporta:
  - identidade do transmissor e número de seqüência unívoco
  - **acks** e **nacks** na carona de mensagens difundidas



Trans

- o receptor, a partir de acks e nacks, determina
  - que mensagens ele não precisa reconhecer (ack)
  - quais ele precisa pedir retransmissão
  - quais ele deve retransmitir

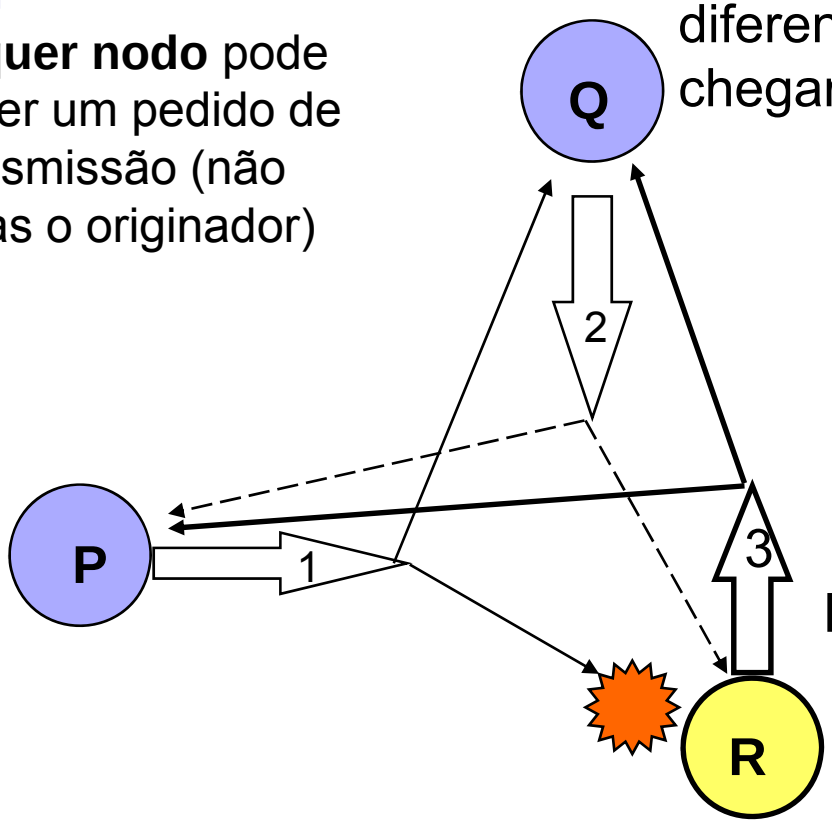


R **recebeu** m1 e m2
R **não** envia ack$_{m1}$
pois Q já enviou

Trans

- se o receptor R determina que não recebeu m1
  - deve pedir retransmissão
  - **qualquer nodo** pode atender um pedido de retransmissão (não apenas o originador)

sem ordenação: mensagens podem ser recebidas em cada nodo em uma ordem diferente (no exemplo m1 chegará em R após m2)



R **não** recebeu m1
R envia nack$_{m1}$ pedindo retransmissão

m3 ack$_{m2}$ nack$_{m1}$

ExemploTrans

- ✓ A, B, C, D = **mens** a, b, c, d = **acks**, a, b, c, d = **nacks**
- ✓ A
- ✓ A Ba — trans. de B reconhece A
- ✓ A Ba Cb — trans. de C reconhece B, não precisa rec. A
- ✓ A Ba Cb Dc
- ✓ A Ba Cb Dc Ecd
- ✓ A Ba Cb Dc Ecd Cb — trans. de E viu por Dc que não recebeu C
- ✓ A Ba Cb Dc Ecd Cb Fec — algum nodo retransmite C(sem novos acks)

Bibliografia

- JALOTE, P. Fault tolerance in distributed systems. Prentice Hall, Englewood Cliffs, New Jersey, 1994
- GÄRTNER, F. C. Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments. ACM Computing Surveys, Vol. 31, No. 1, March 1999.
- DEFAGO, X.; SCHIPER, A.; URBAN, P. Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. ACM Computing Surveys, Vol. 36, No. 4, December 2004, pp. 372–421
- FREILING, GUERRAOUI, KUZNETSOV, The Failure Detector Abstraction . ACM Computing Surveys. Vol 43 Issue 2, Jan 2011

OBS: nas notas de aulas da disciplina, um roteiro básico pode ser encontrado com um resumo dos tópicos discutidos neste item.