

Sistemas Distribuídos

Falhas e Consenso

Vladimir Rocha

Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - 3.4. Consenso no modelo parcialmente-síncrono
 4. Conclusão
-

Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - 3.4. Consenso no modelo parcialmente-síncrono
 4. Conclusão
-

1. Contexto

- Concordar o horário de ataque (*Two Generals' Problem*)



1. Contexto

- Concordar o horário de ataque (*Two Generals' Problem*)
 - Os generais 1 e 2 somente vencem a torre se atacarem simultaneamente
 - Os generais 1 e 2 somente se comunicam através de mensagens
 - As mensagens podem ser perdidas no caminho



1. Contexto

- Concordar o horário de ataque (*Two Generals' Problem*)
 - O general 1 (você) envia uma mensagem M1 para atacar a torre às 20.00 de hoje.
 - Você atacaria às 20.00?



1. Contexto

- Concordar o horário de ataque (*Two Generals' Problem*)
 - O general 1 (você) envia uma mensagem M1 para atacar a torre às 20.00 de hoje.
 - Você atacaria às 20.00?
 - Precisa de uma confirmação !



1. Contexto

- Concordar o horário de ataque (*Two Generals' Problem*)
 - O general 2 recebe M1 e responde com a mensagem M2 que aceita atacar às 20.00.
 - O general 2 deveria atacar às 20.00?



1. Contexto

- Concordar o horário de ataque (*Two Generals' Problem*)
 - O general 2 recebe M1 e responde com a mensagem M2 que aceita atacar às 20.00.
 - O general 2 deveria atacar às 20.00?
 - O general 2 precisa de uma confirmação! (pois a M2 pode ter se perdido)



1. Contexto

- Concordar o horário de ataque (*Two Generals' Problem*)
 - O general 1 (você) recebe M2 e responde com a mensagem M3 que confirma o ataque.
 - Você deveria atacar?



1. Contexto

- Concordar o horário de ataque (*Two Generals' Problem*)
 - O general 1 (você) recebe M2 e responde com a mensagem M3 que confirma o ataque.
 - Você deveria atacar?
- Note que sempre haverá a necessidade de confirmar a mensagem do outro



1. Contexto

- Concordar o horário de ataque (*Two Generals' Problem*)
 - O general 1 (você) recebe M2 e responde com a mensagem M3 que confirma o ataque.
 - Você deveria atacar?
- Note que sempre haverá a necessidade de confirmar a mensagem do outro
- **Impossível** de resolver sob certas condições [FLP85].



1. Contexto

- Nos sistemas distribuídos, o problema dos generais pode ser caracterizado como a coordenação entre um grupo de processos para chegarem a um acordo por um valor.
- Esse acordo é denominado de **consenso** [coulouris13].

1. Contexto

- O consenso é um problema comum a diversos serviços especificados para os sistemas distribuídos:
 - Eleição de um líder (*Leader Election*)
 - Acesso exclusivo a um recurso (*Mutual Exclusion*)
 - Transações distribuídas (*Two Phase Commit*)
- Usado em diversos sistemas:
 - Blockchain
 - Replicação
- E em diversas funcionalidades:
 - Criação de um username único
 - Reservas (passagens, itens, cinemas)

1. Contexto

- O que devemos fazer para que todos os processos cheguem a um consenso?
- Essa aula mostrará os mecanismos e condições sob quais é possível resolver esse problema

Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - 3.4. Consenso no modelo parcialmente-síncrono
 4. Conclusão
-

2. Revisão e novos Conceitos

- 2.1. Modelos de sistema [revisão Cap1]
- 2.2. Modelos baseado em falhas
- 2.3. Safety & Liveness
- 2.4. Interfaces de um componente

2. Revisão de Conceitos

2.1. Modelos de sistema [tanenbaum17]

- Síncrono
- Assíncrono
- Parcialmente síncrono

2. Revisão de Conceitos

- Síncrono:

Os limites de tempo para transferir uma mensagem são conhecidos

Os limites de tempo para processar uma ação são conhecidos

LAN/Datacenters

- Assíncrono:

Sem limites de tempo para transferir uma mensagem

Sem limites de tempo para processar uma ação

Internet

2. Revisão de Conceitos

- Parcialmente Síncrono:

Inicialmente o comportamento do sistema é assíncrono

Eventualmente o comportamento é síncrono

a partir do *Global Stabilization Time* (**GST**).

Internet

2. Novos Conceitos

2.2. Modelos baseado em falhas

- Falha do processo
- Falha do algoritmo

2. Novos Conceitos

- Falha do processo

Falha por Parada (*Crash-Failure*)

Omissão (*Omission*)

Recuperação (*Recovery*)

Violação (*Eavesdropping*)

Arbitrária (*Byzantine*)



Do mais simples
ao mais complexo

- Um processo **correto** é aquele que não apresenta falhas

2. Novos Conceitos

- Falha do processo

Falha por Parada (*Crash-Failure*)



Para e permanece parado.
Outros processos podem
detectar esse estado.

Omissão (*Omission*)

Recuperação (*Recovery*)

Violação (*Eavesdropping*)

Arbitrária (*Byzantine*)

- Um processo **correto** é aquele que não apresenta falhas

2. Novos Conceitos

- Falha do processo

Falha por Parada (*Crash-Failure*)

Omissão (*Omission*)

Recuperação (*Recovery*)

Violação (*Eavesdropping*)

Arbitrária (*Byzantine*)



Omite o envio ou recebimento de mensagens.

- Um processo **correto** é aquele que não apresenta falhas

2. Novos Conceitos

- Falha do processo

Falha por Parada (*Crash-Failure*)

Omissão (*Omission*)

Recuperação (*Recovery*)

Violação (*Eavesdropping*)

Arbitrária (*Byzantine*)



Para por um tempo mas volta a funcionar. Pode armazenar informações (log)

- Um processo **correto** é aquele que não apresenta falhas

2. Novos Conceitos

- Falha do processo

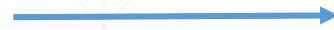
Falha por Parada (*Crash-Failure*)

Omissão (*Omission*)

Recuperação (*Recovery*)

Violação (*Eavesdropping*)

Arbitrária (*Byzantine*)



As informações são observadas por terceiros (problema de confidencialidade).

- Um processo **correto** é aquele que não apresenta falhas

2. Novos Conceitos

- Falha do processo

Falha por Parada (*Crash-Failure*)

Omissão (*Omission*)

Recuperação (*Recovery*)

Violação (*Eavesdropping*)

Arbitrária (*Byzantine*)



Qualquer tipo de comportamento errôneo ou malicioso (não confiável)

- Um processo **correto** é aquele que não apresenta falhas

2. Novos Conceitos

- Falha do algoritmo

Fail-Stop

Fail-Noisy

Fail-Silent

Fail-Safe

Fail-Arbitrary



Do mais simples
ao mais complexo

2. Novos Conceitos

- Falha do algoritmo

Fail-Stop



Se um processo falha será por parada. Os outros processos detectam essa falha de forma precisa.

Fail-Noisy

Fail-Silent

Fail-Safe

Fail-Arbitrary

2. Novos Conceitos

- Falha do algoritmo

Fail-Stop

Fail-Noisy →

Fail-Silent

Fail-Safe

Fail-Arbitrary

Se um processo falha, os outros processos não conseguem detectar durante um tempo essa falha de forma precisa. Porém, depois de um tempo, sim.

2. Novos Conceitos

- Falha do algoritmo

Fail-Stop

Fail-Noisy

Fail-Silent

Fail-Safe

Fail-Arbitrary



Qualquer tipo de
comportamento errôneo ou
malicioso (não confiável)

2. Novos Conceitos

2.3. Safety & Liveness

- Propriedades do algoritmo (ou sistema) distribuído a serem alcançadas.
- Nem sempre é possível obter as duas propriedades.

2. Novos Conceitos

- **Safety property**

Proíbe que “coisas ruins” aconteçam (*bad things don't happen*).

Violação detectada em uma execução finita.

- **Exemplos**

- O anel do Chord não será quebrado.
- Um recurso com exclusão mútua só será acessado por um processo por vez.
- No algoritmo de eleição, somente haverá um líder.
- No multicast com ordem total, se um processo envia m1 e depois m2, estas serão entregues nessa ordem em todos os outros processos.
- Na blockchain Hyperledger, todos enxergam a mesma ordem da cadeia de blocos.

2. Novos Conceitos

- Liveness property

Eventualmente “coisas boas” acontecerão (*good things do happen*)

Violação não pode ser detectada em uma execução finita.

- Exemplos

- Eventualmente os fingers do Chord serão restabelecidos.
- Na exclusão mútua, o processo acessará o recurso.
- No algoritmo de eleição, eventualmente será escolhido um líder.
- No multicast com ordem total, todos os processos em algum momento obterão as mensagens e farão a entrega (delivery) à aplicação.
- Na blockchain Bitcoin, eventualmente a cadeia de blocos convergirá para todos

2. Novos Conceitos

2.4. Interfaces de um componente (software que executa um serviço)

- Solicita: método que permite solicitar um pedido ao componente
- Indicação: método que permite receber uma informação do componente

E.g. Interface para um componente que realiza a escolha de um líder

- Solicita: $\langle \text{Eleição} \mid P_i \rangle$
- Indicação: $\langle \text{Líder} \mid P \rangle$

Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - 3.4. Consenso no modelo parcialmente-síncrono
 4. Conclusão
-

3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

Em outras palavras, cada processo pode propor um valor diferente, porém todos devem concordar com somente um desses valores propostos.

3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

No caso dos generais



3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

No caso dos generais



Consenso

Consenso

3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

No caso dos generais



3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

No caso da blockchain



3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

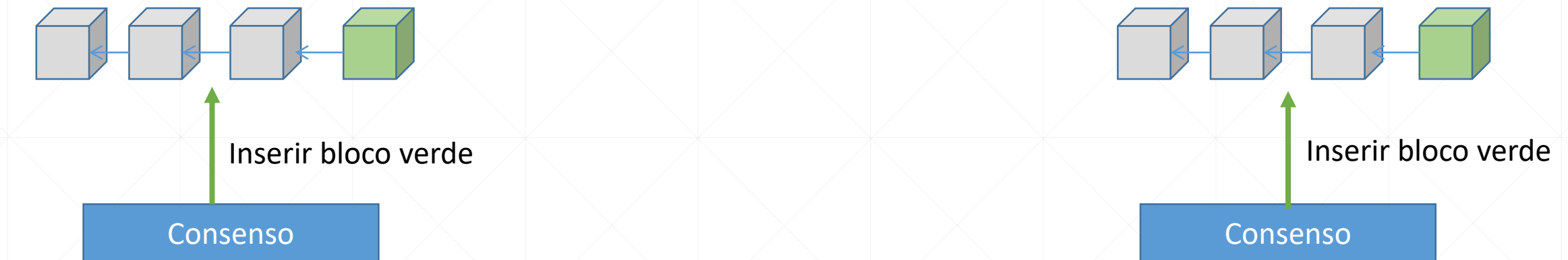
No caso da blockchain



3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

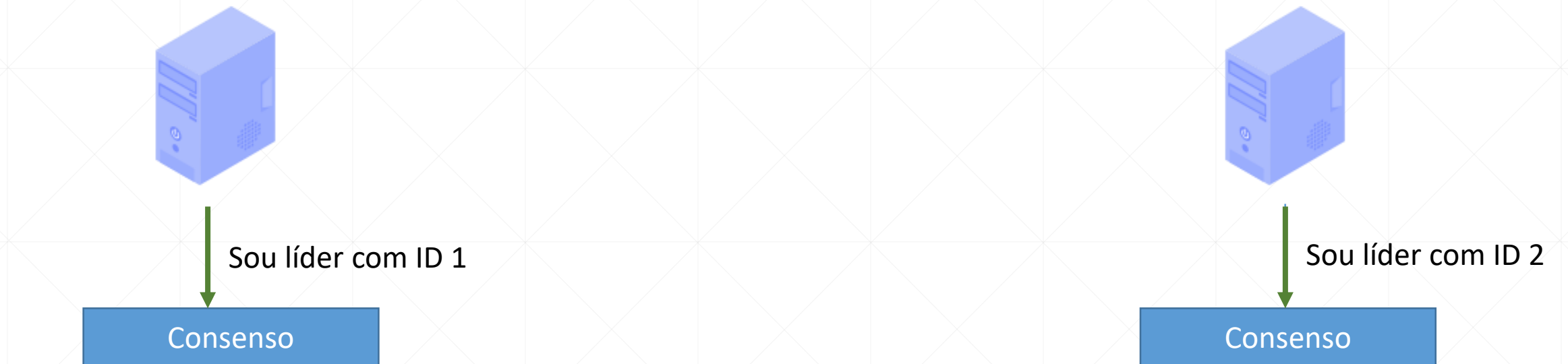
No caso da blockchain



3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

No caso da eleição de um líder



3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

No caso da eleição de um líder



Consenso

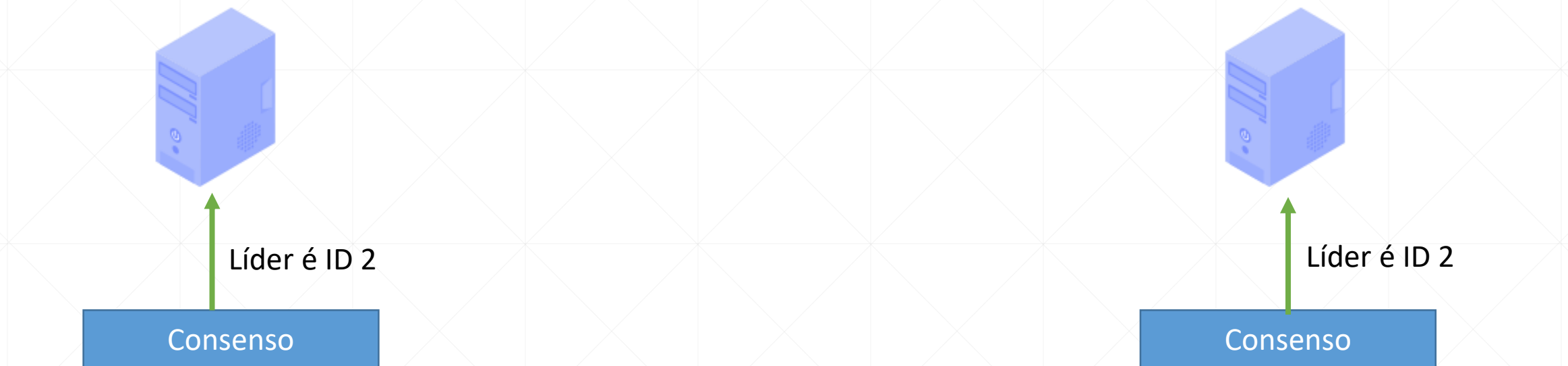
Consenso



3. Consenso

Nos sistemas distribuídos, consenso é o mecanismo que permite a um grupo de processos concordarem com um valor.

No caso da eleição de um líder



Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - 3.4. Consenso no modelo parcialmente-síncrono
 4. Conclusão
-

3.1. Especificação

- Para especificar o consenso devemos considerar:
 - Interfaces do componente
 - Modelo
 - Propriedades de corretude
 - Implementação e desempenho
-
- O que deve realizar o serviço
- Como deve ser realizado

Veremos o consenso **determinístico**: dada uma mesma entrada, o consenso deverá sempre produzir o mesmo resultado


Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - 3.4. Consenso no modelo parcialmente-síncrono
 4. Conclusão
-

3.2. Consenso no Modelo Assíncrono

- Interfaces do componente (instalado em cada processo):
 - Solicita: <Propose | v>
 - Indicação: <Decide | v>
- Modelo
 - Sistema Assíncrono
 - Falha do Processo: falha por parada (*crash-failure*)

3.2. Consenso no Modelo Assíncrono

- Interfaces do componente (instalado em cada processo):
 - Solicita: <Propose | v>
 - Indicação: <Decide | v>
- Modelo
 - Sistema Assíncrono
 - Falha do Processo: falha por parada (*crash-failure*)  Caso mais simples !

3.2. Consenso no Modelo Assíncrono

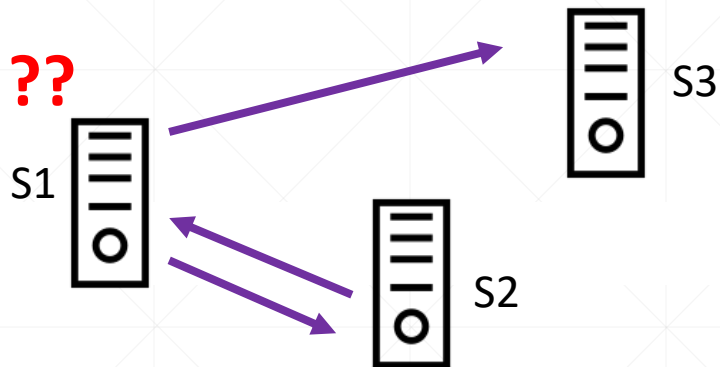
- Interfaces do componente (instalado em cada processo):

- Solicita: $\langle \text{Propose} \mid v \rangle$
- Indicação: $\langle \text{Decide} \mid v \rangle$


- Modelo

- Sistema Assíncrono

- Falha do Processo: falha por parada (*crash-failure*) ← Caso mais simples !



3.2. Consenso no Modelo Assíncrono

- Interfaces do componente (instalado em cada processo):
 - Solicita: <Propose | v>
 - Indicação: <Decide | v>
- Modelo
 - Sistema Assíncrono
 - Falha do Processo: falha por parada (*crash-failure*)  Caso mais simples !
- Conhecido como **resultado FLP**, foi provado que não há solução determinística garantida caso um nó falhe [FLP85].

Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 - 3. Consenso**
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono**
 - 3.4. Consenso no modelo parcialmente-síncrono
 4. Conclusão
-

3.3. Consenso no Modelo Síncrono (Regular)

- Interfaces do componente (instalado em cada processo):
 - Solicita: <Propose | v>
 - Indicação: <Decide | v>

3.3. Consenso no Modelo Síncrono (Regular)

- Interfaces do componente (instalado em cada processo):

- Solicita: <Propose | v>
- Indicação: <Decide | v>

- Modelo:

- Sistema síncrono
- Falha do Processo: falha por parada (*crash-failure*)
- Falha do Algoritmo: *fail-stop*



Casos mais simples !

3.3. Consenso Regular

- Propriedades de Corretude do consenso
 - Validade: somente valores propostos podem ser decididos
 - Término: cada processo correto eventualmente decidirá
 - Integridade: cada processo decide somente uma vez
 - Acordo: todos os processos **corretos** decidirão pelo mesmo valor
 - Nada se assume para os processos que falham.

3.3. Consenso Regular

- Implementação (Visão Geral)

- Supondo N processos (sendo P_i o processo com identificador i)
- Seja *proposal* a variável que armazena o valor proposto
- Seja *lastProp* o valor do identificador de um processo

Para cada rodada (*round*) $i = 1$ a N

Processo com identificador i é o **líder** da rodada

Decide por um valor V

Faz o *broadcast* de V para todos os processos

Outros processos

Adotam o valor V em *proposal*

Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou (passa para próxima rodada)

3.3. Consenso Regular

Para cada rodada (*round*) $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

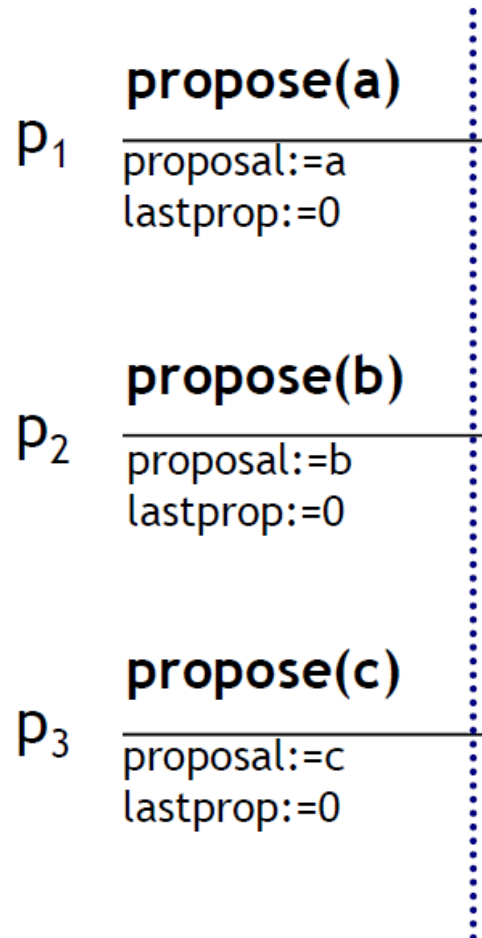
Faz o *broadcast* de V para todos os processos

Outros processos

Adotam o valor V em *proposal*

Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou, passa próxima rodada



3.3. Consenso Regular

Para cada rodada (*round*) $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

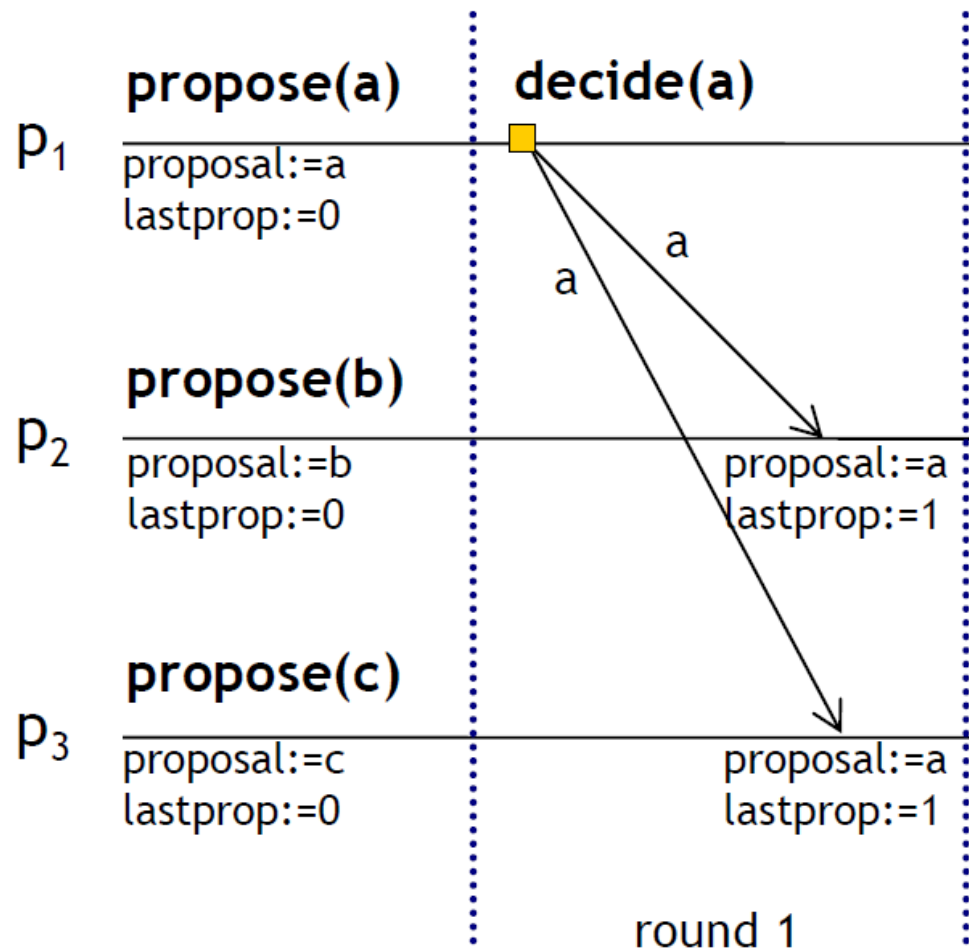
Faz o *broadcast* de V para todos os processos

Outros processos

Adotam o valor V em *proposal*

Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou, passa próxima rodada



3.3. Consenso Regular

Para cada rodada (*round*) $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

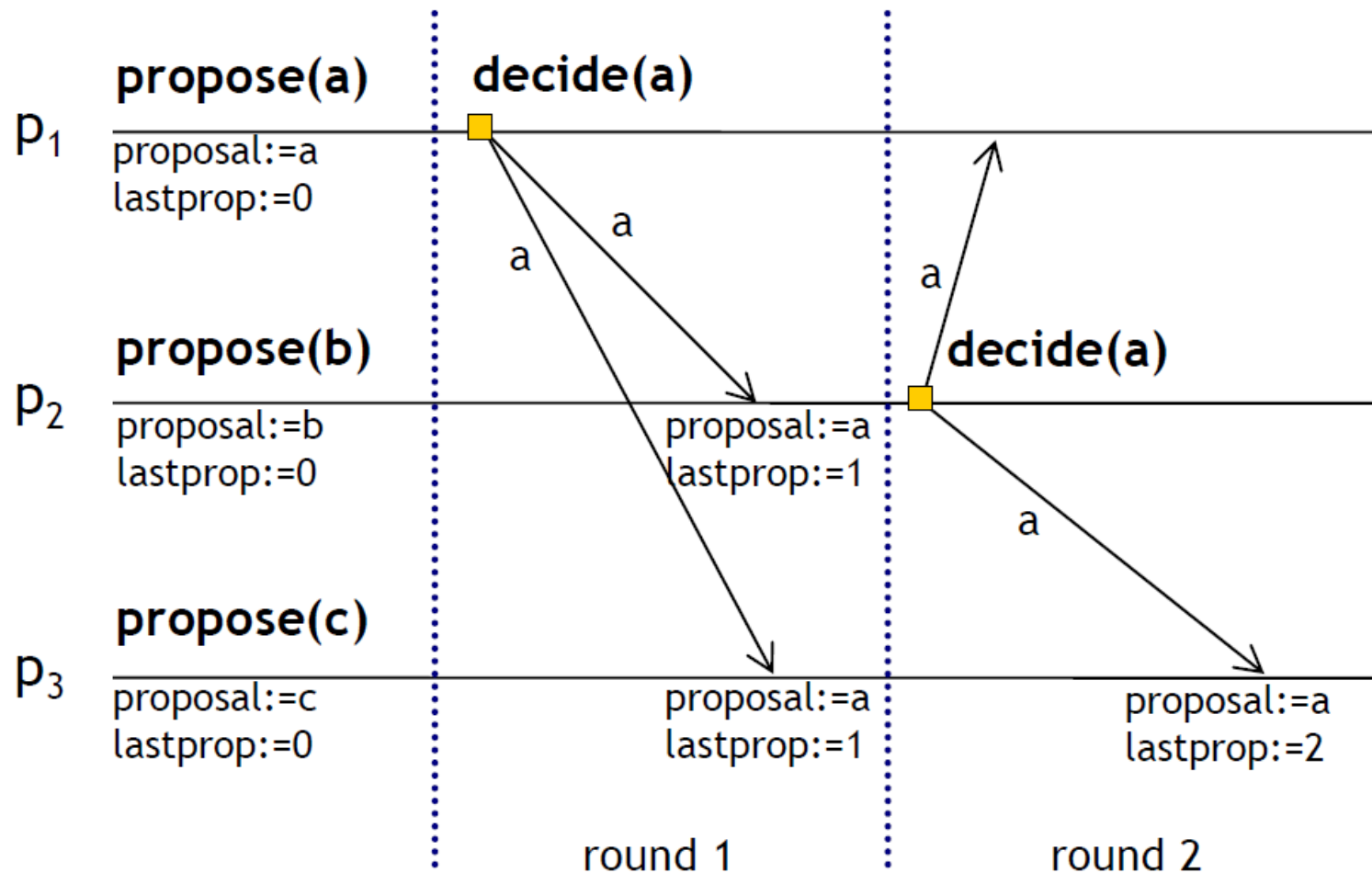
Faz o *broadcast* de V para todos os processos

Outros processos

Adotam o valor V em *proposal*

Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou, passa próxima rodada



3.3. Consenso Regular

Para cada rodada (*round*) $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

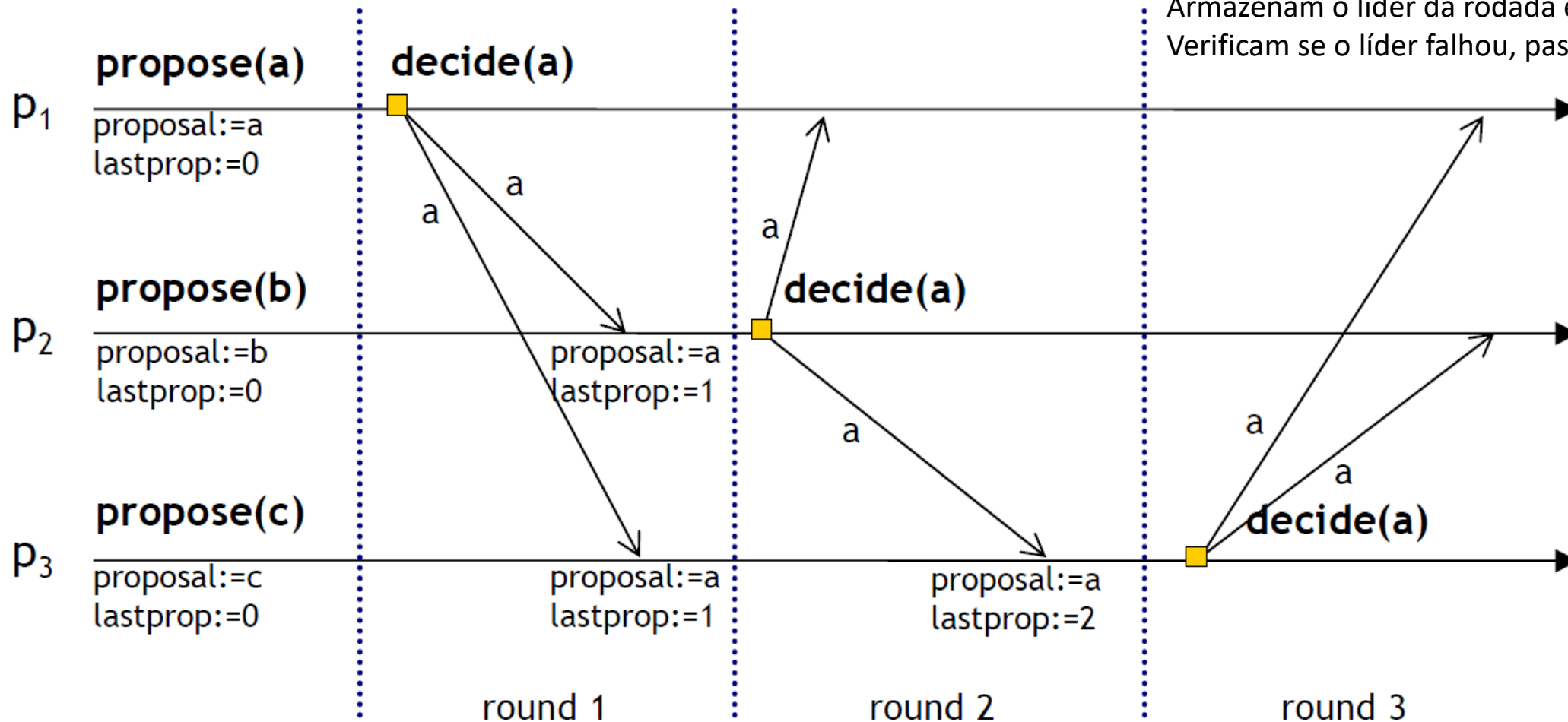
Faz o *broadcast* de V para todos os processos

Outros processos

Adotam o valor V em *proposal*

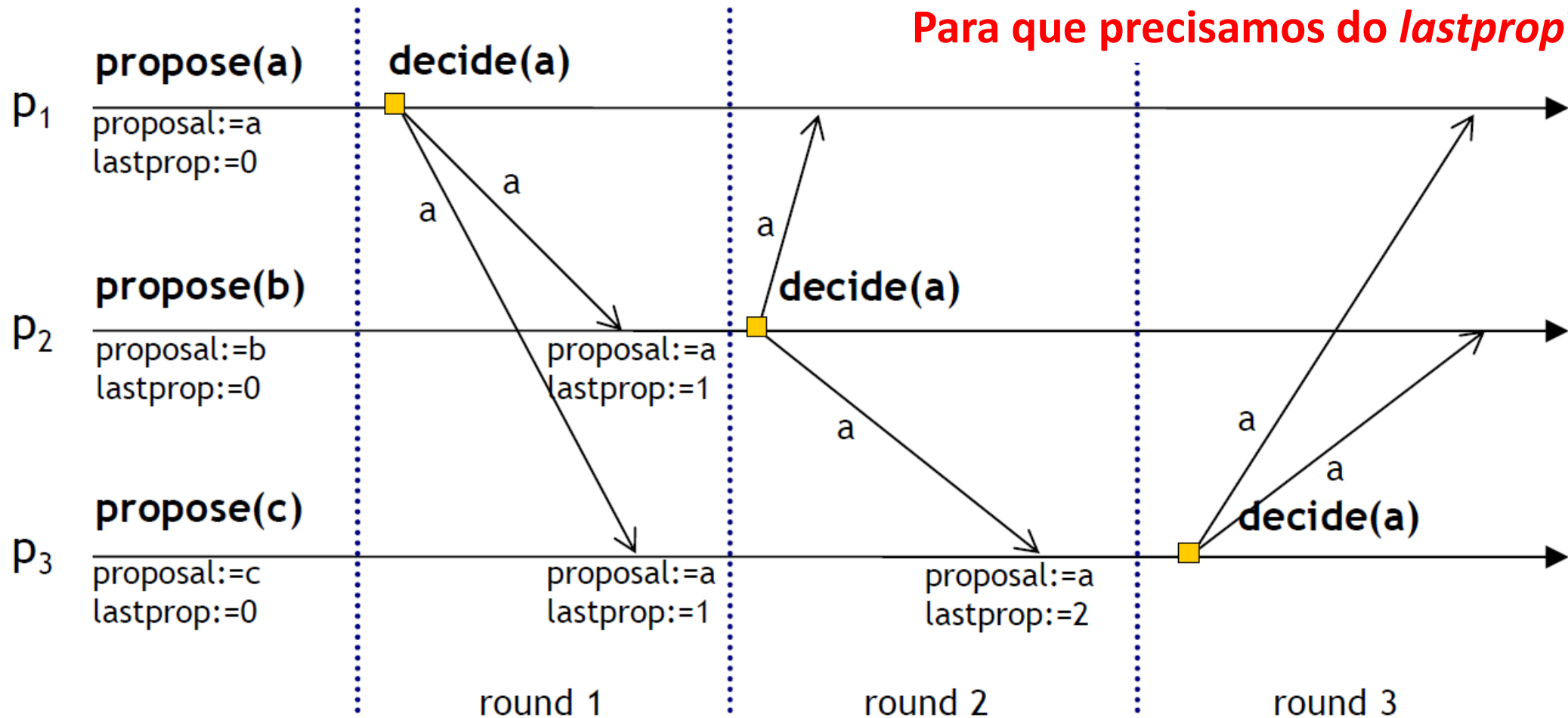
Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou, passa próxima rodada



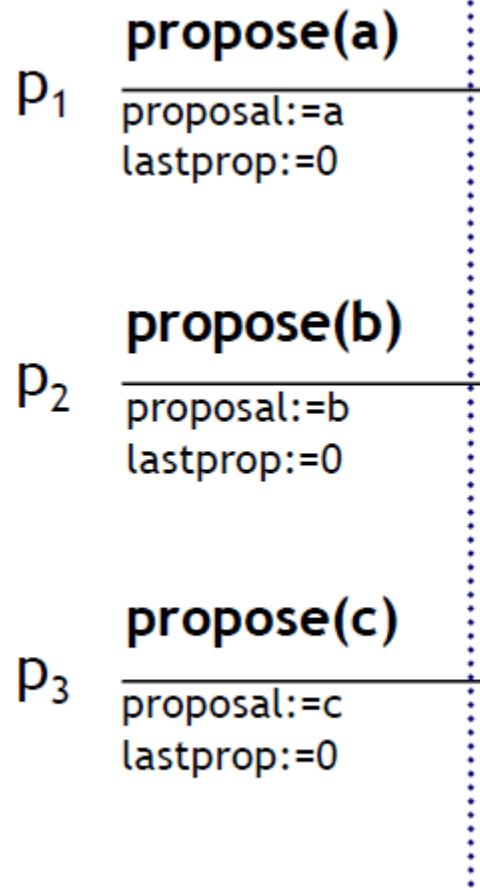
3.3. Consenso Regular

Para que precisamos do *lastprop*?

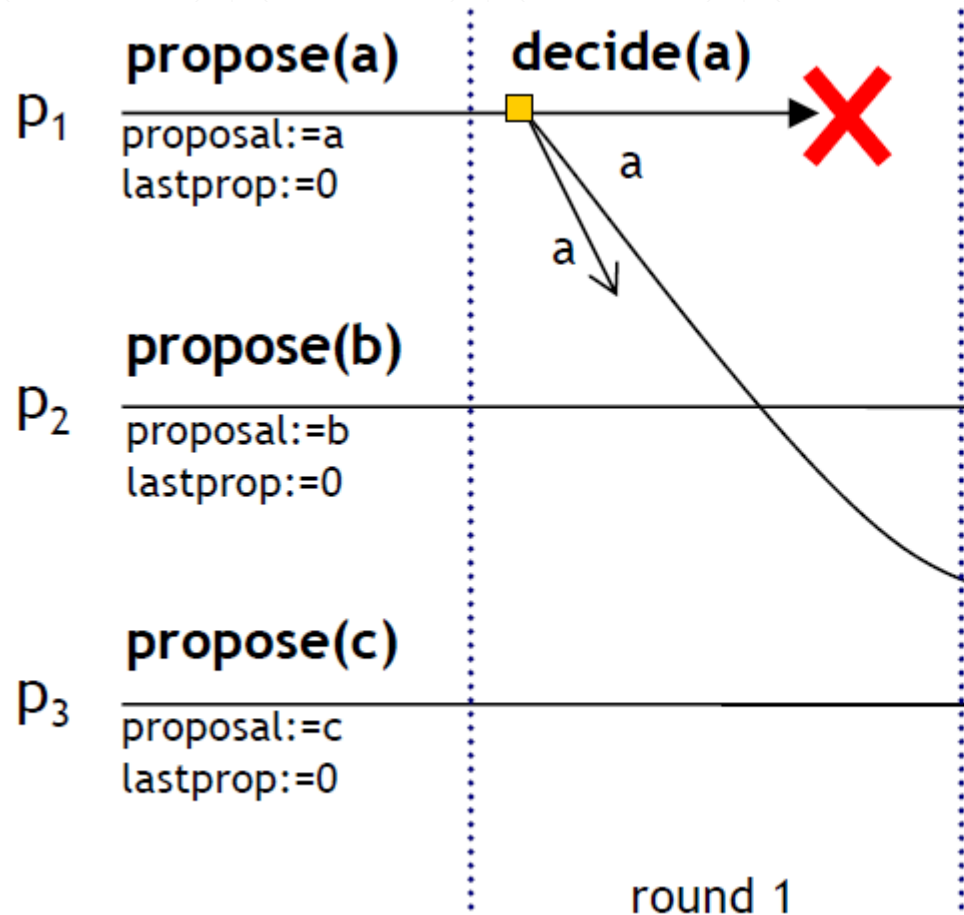


3.3. Consenso Regular

Para que precisamos do *lastprop*?



3.3. Consenso Regular

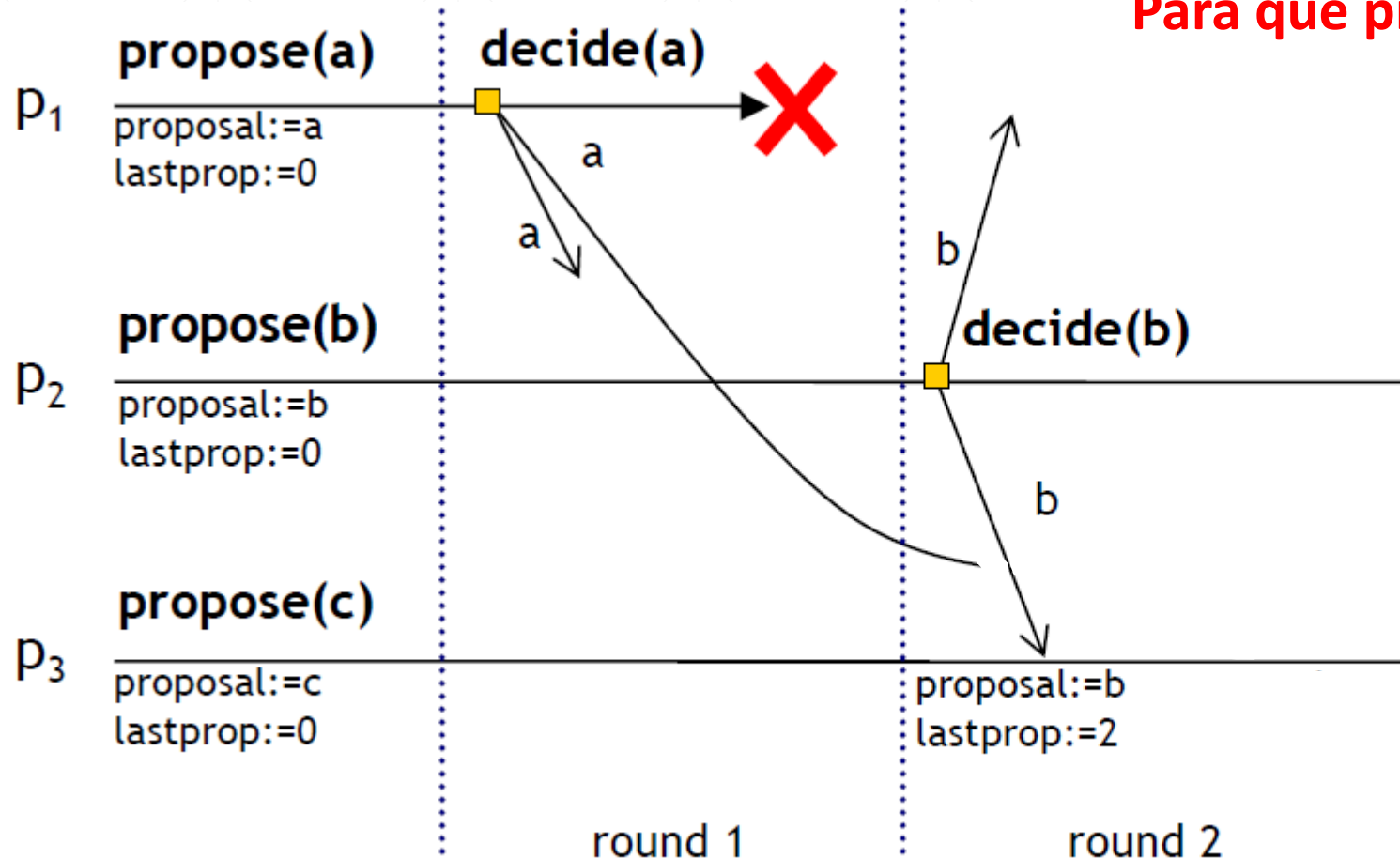


Para que precisamos do *lastprop*?

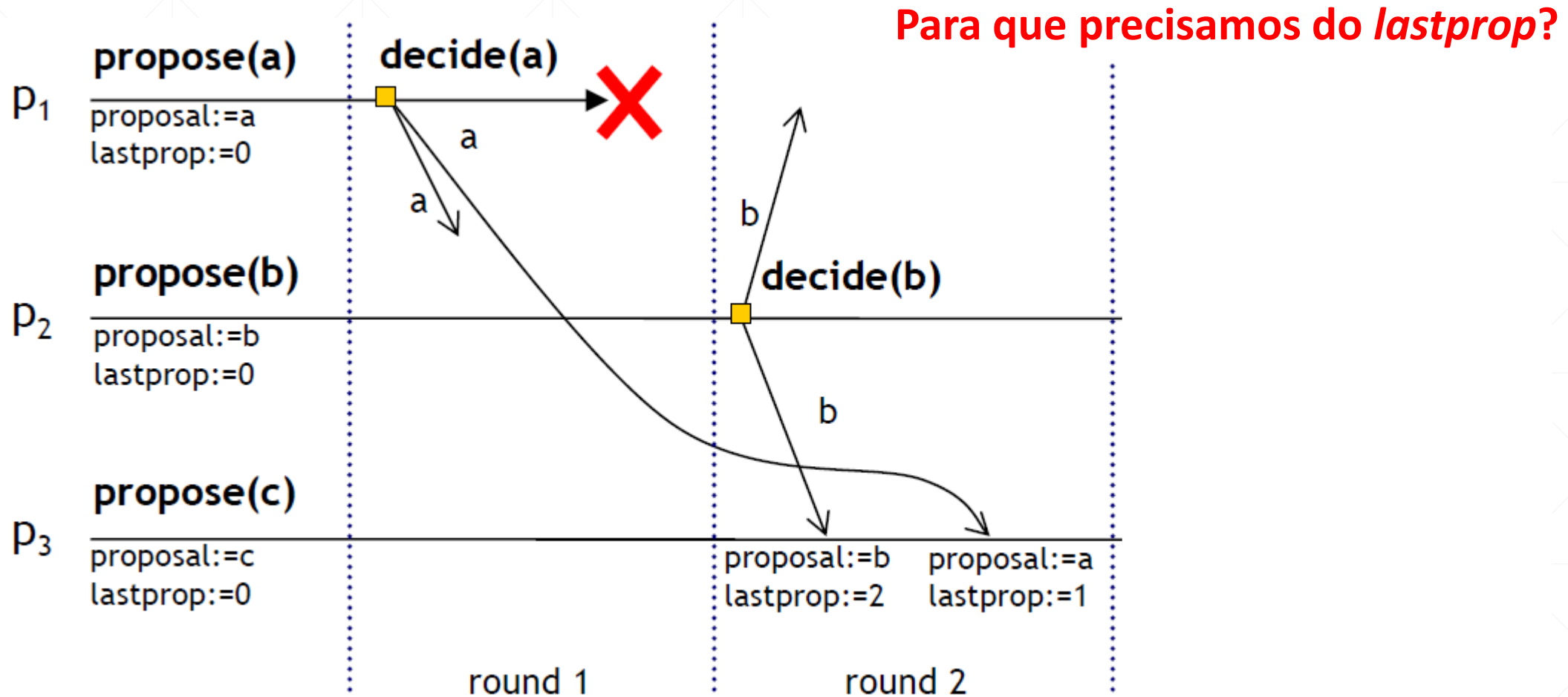
acordo: todos os processos **corretos**
decidirão pelo mesmo valor

3.3. Consenso Regular

Para que precisamos do *lastprop*?

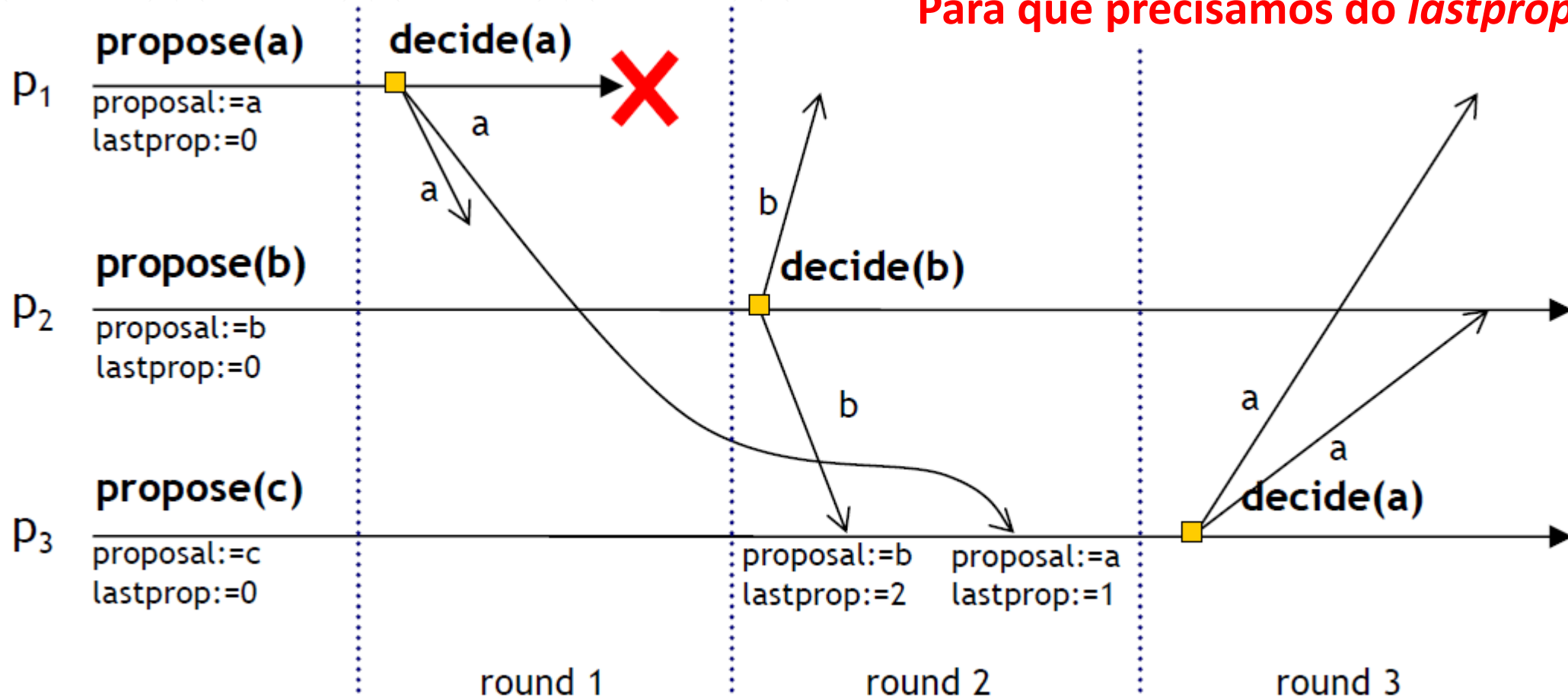


3.3. Consenso Regular



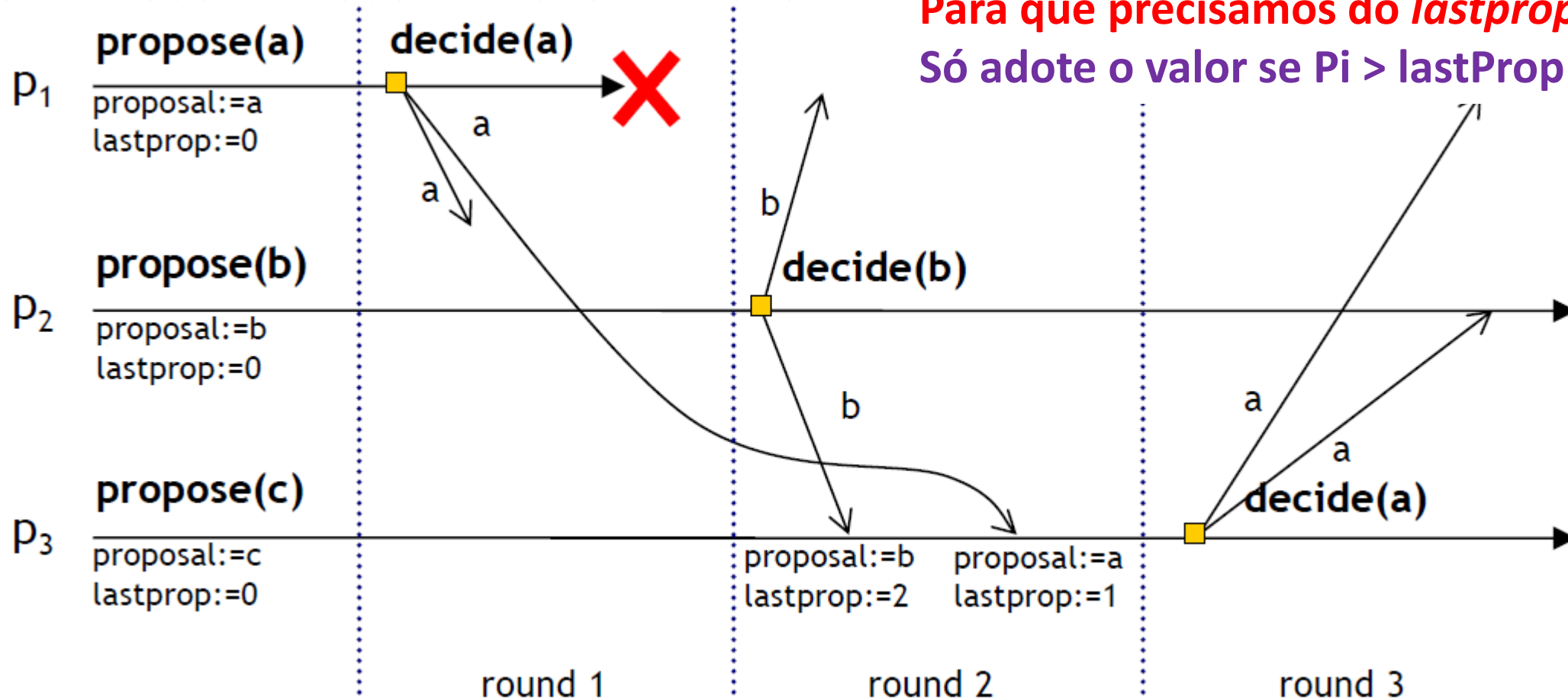
3.3. Consenso Regular

Para que precisamos do *lastprop*?

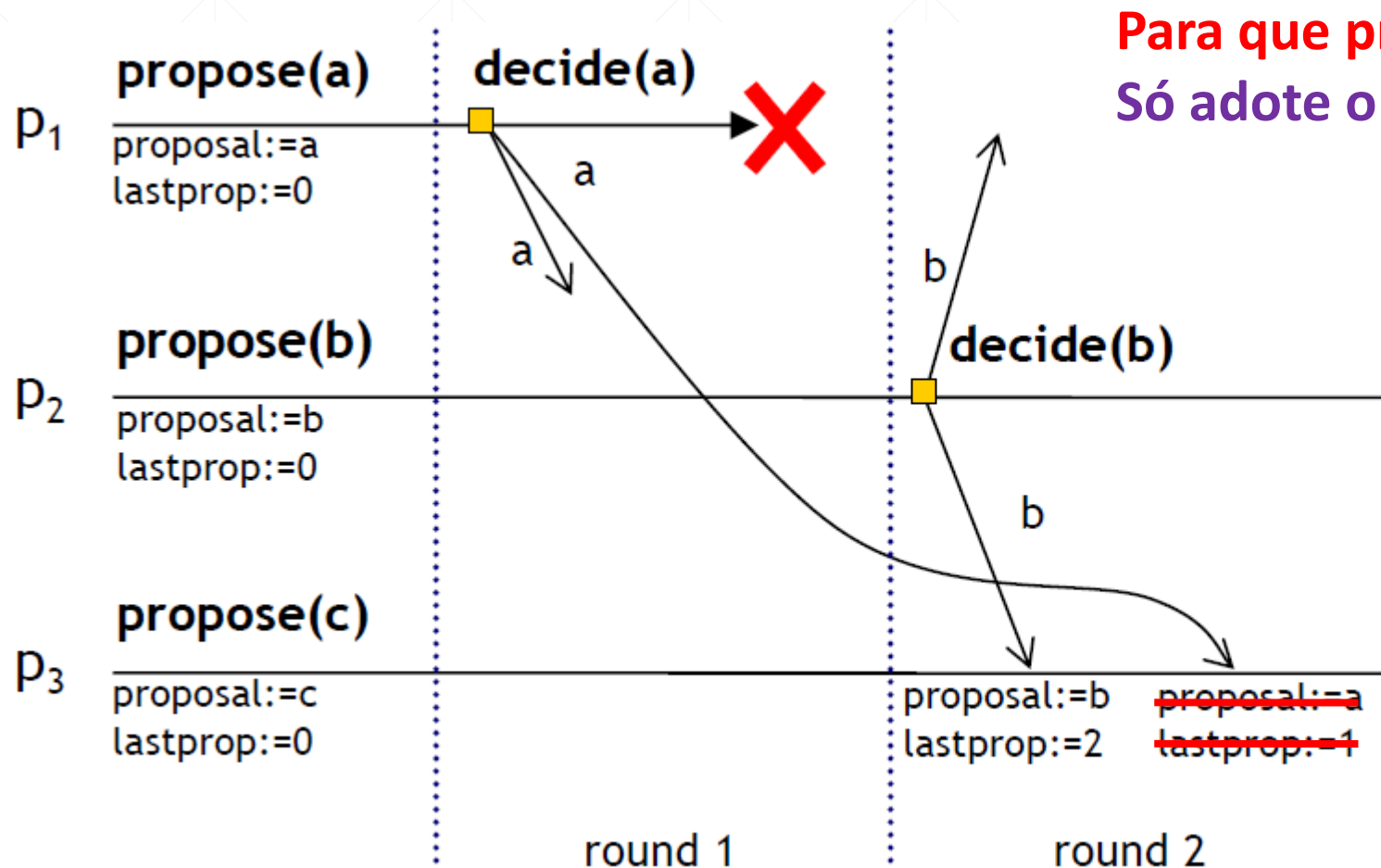


3.3. Consenso Regular

Para que precisamos do *lastprop*?
Só adote o valor se $P_i > \text{lastProp}$



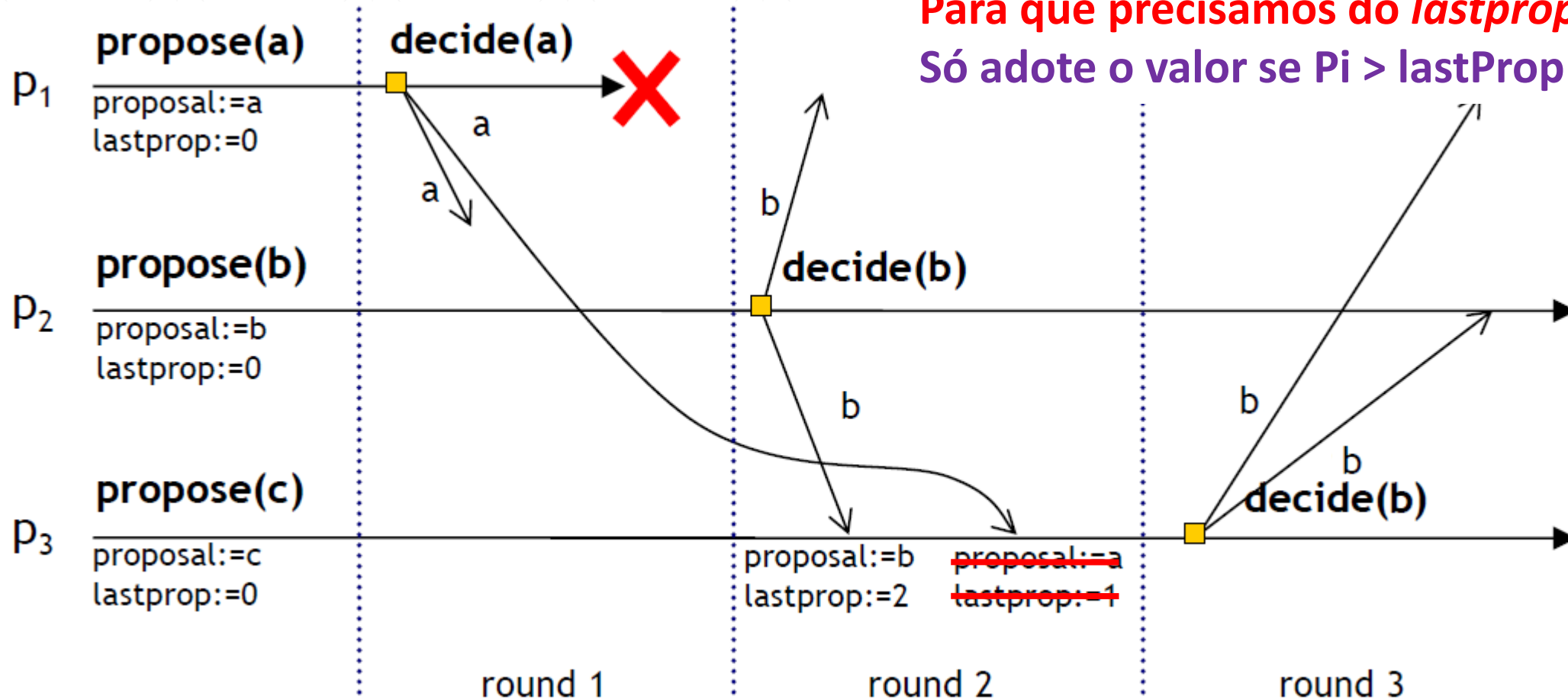
3.3. Consenso Regular



Para que precisamos do *lastprop*?
Só adote o valor se $P_i > \text{lastProp}$

3.3. Consenso Regular

Para que precisamos do *lastprop*?
Só adote o valor se $P_i > \text{lastProp}$



3.3. Consenso Regular

- Atualizando a implementação

Para cada rodada $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

Faz o *broadcast* de V para todos os processos

Outros processos

Adotam o valor V em *proposal*

Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou (passa para a próxima rodada)

3.3. Consenso Regular

- Atualizando a implementação

Para cada rodada $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

Faz o *broadcast* de V para todos os processos

Outros processos

$P_i > \text{lastProp}$ { Adotam o valor V em *proposal*
Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou (passa para a próxima rodada)

3.3. Consenso Regular

- Corretude da Implementação (**Validade, Término, Integridade, Acordo**)

Para cada rodada $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

Faz o *broadcast* de V para todos os processos

Outros processos

$P_i > lastProp$ { Adotam o valor V em *proposal*
Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou (passa para a próxima rodada)

3.3. Consenso Regular

- Corretude da Implementação

Para cada rodada $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

Faz o *broadcast* de V para todos os processos

Validade: só se decide por um
valor proposto ...

Outros processos

$P_i > lastProp$ { Adotam o valor V em *proposal*
Armazenam o líder da rodada em *lastProp*

... ou adotado

Verificam se o líder falhou (passa para a próxima rodada)

3.3. Consenso Regular

- Corretude da Implementação

Para cada rodada $i = 1$ a N



Término: no máximo N rodadas ...

Processo com identificador i é o líder da rodada

Decide por um valor V

Faz o *broadcast* de V para todos os processos

Outros processos

$P_i > \text{lastProp}$ { Adotam o valor V em *proposal*
Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou (passa para a próxima rodada)



... se o líder falha, a verificação
fail-stop permite a progressão

3.3. Consenso Regular

- Corretude da Implementação

Para cada rodada $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

Faz o *broadcast* de V para todos os processos

Outros processos

$P_i > lastProp$ { Adotam o valor V em *proposal*
Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou (passa para a próxima rodada)

Integridade: para cada um das rodadas

o processo decide somente 1 vez
(quando é líder)

3.3. Consenso Regular

- Corretude da Implementação

Para cada rodada $i = 1$ a N

Processo com identificador i é o líder da rodada

Decide por um valor V

Faz o *broadcast* de V para todos os processos

Outros processos

$P_i > \text{lastProp}$ { Adotam o valor V em *proposal*
Armazenam o líder da rodada em *lastProp*

Verificam se o líder falhou (passa para a próxima rodada)

Acordo:

Seja i o leader correto com o mínimo id.

... que decide um valor V

todos os processos $k > i$ também decidem V , pois i é correto e porque para passar à rodada $i+1$ o processo $i+1$ recebeu V pelo *broadcast* (sem sobreescrita).

3.3. Consenso Regular

- Desempenho da implementação

Quantidade de Mensagens:

N rodadas, com N mensagens transferidas por rodada $O(N^2)$.

broadcast

Tolerância a falhas:

Até $N-1$

Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - 3.4. Consenso no modelo parcialmente-síncrono
 4. Conclusão
-

3.4. Consenso no Modelo Parcialmente Síncrono

- Interfaces do componente (instalado em cada processo):
 - Prepare – Promise
 - Accept – Learn
 - Modelo:
 - Sistema parcialmente síncrono
 - Falha do Processo: *crash/omission/recovery failure*
 - Falha do Algoritmo: *fail-noisy*
 - Propriedades:
 - Sim garante safety: todos escolhem o mesmo valor
 - Não garante liveness: pode ficar indefinidamente tentando chegar ao consenso
-

3.4. Consenso Paxos

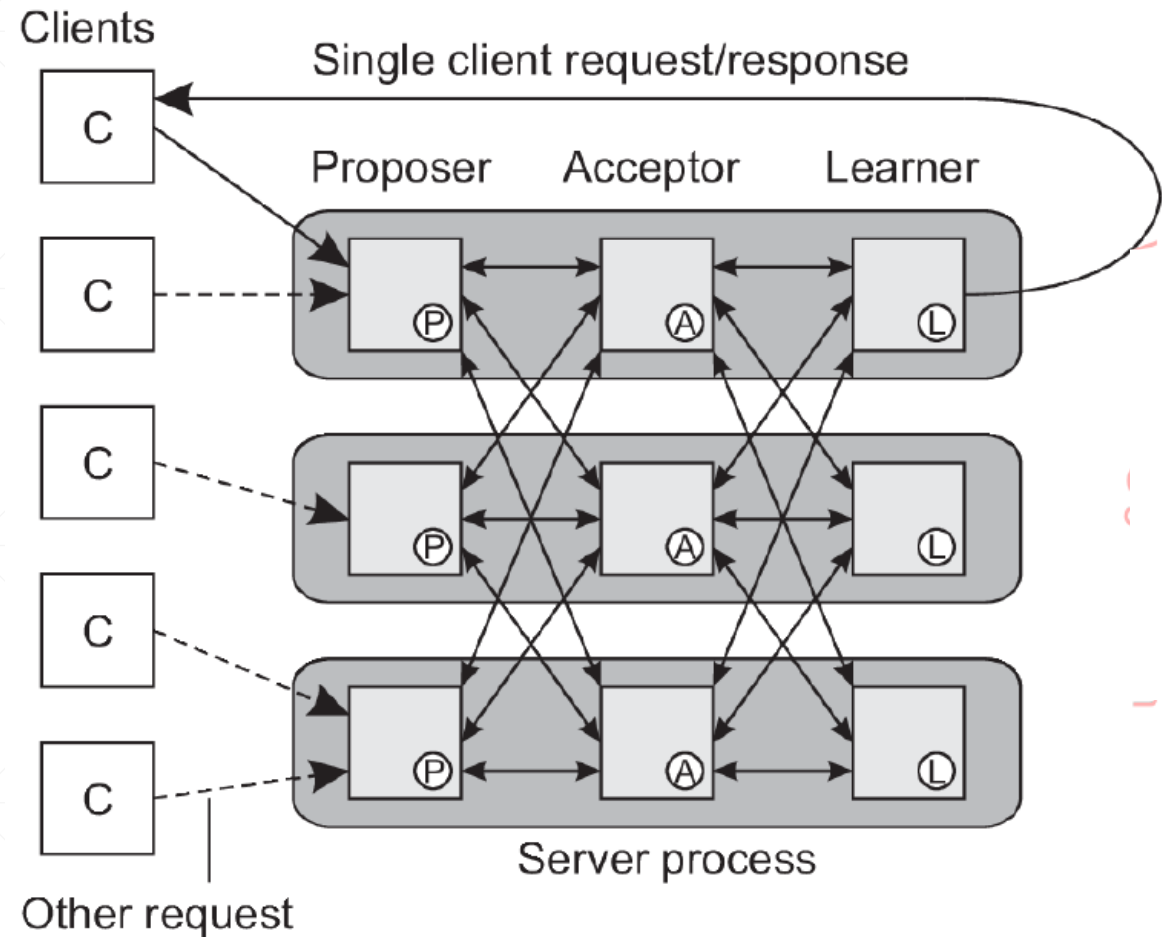
- Paxos [Paxos01] é utilizado como base de diversas tecnologias, como:
 - Zookeeper → Kafka, Hadoop/Spark, Solr e Facebook, Twitter, Yahoo
<https://zookeeper.apache.org/doc/current/zookeeperUseCases.html>
 - Raft → Blockchain Hyperledger, Kubernetes
 - Chubby → Google
- Uso de quórum (especificamente da maioria), pois não é possível esperar pela resposta de todos os processos em um sistema parcialmente síncrono.
- Uso de duas fases, uma para a preparação e outra para a aceitação

3.4. Consenso Paxos

- Organização dos processos:

Se a maioria dos acceptors
aceitam uma proposta v , então
 v deve ser escolhida

Os learners decidem (“decide”)
a proposta escolhida v



3.4. Consenso Paxos

- Paxos usa o conceito de round (ballot)

Se uma proposta com o round n (denominado proposal number) e valor v foi decidido ('decide') então qualquer round maior a n terá valor v .

Isso garante o consenso !

3.4. Consenso Paxos: algoritmo

Proposer

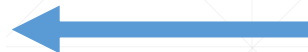
1. Obter um número único (proposal number) n .
2. **Envie prepare(n)** para todos os acceptors



4. Quando recebidas respostas da maioria:
Escolha o valor v do maior proposal number
Envie accept(n, v) para todos os acceptors.



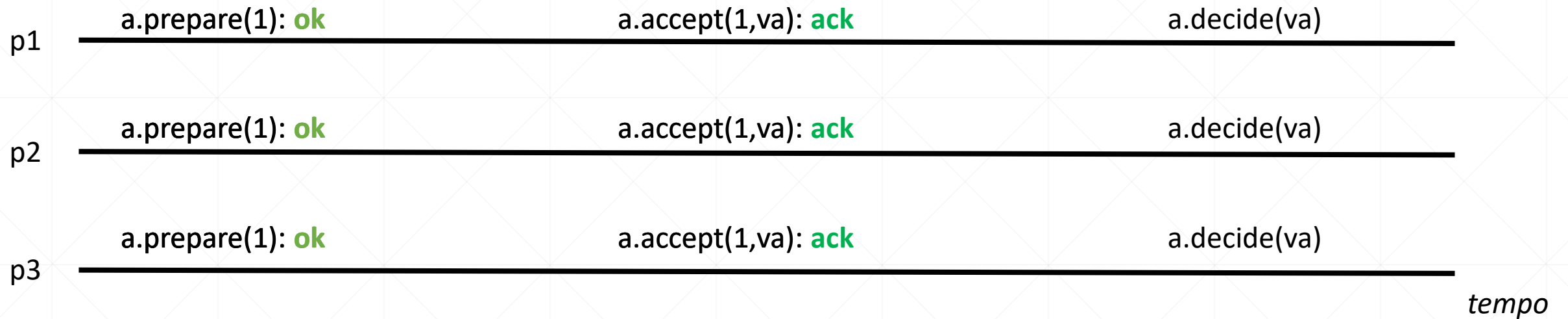
6. Quando recebidas respostas da maioria:
se consegue maioria de acks
 decide(v) & envie para todos os learners
else
 abort



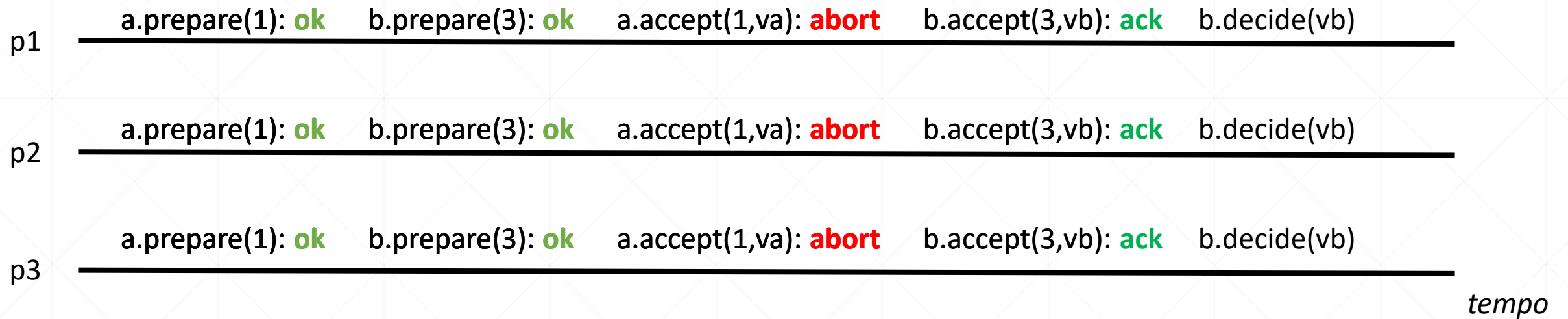
Acceptor

3. Quando receber prepare(n):
Ignore propostas menores a n
return **ok** n_{accepted} v_{accepted}
inicialmente \perp e atualizados
no passo 5. accept
5. Quando recebido accept(n, v):
se não respondeu um prepare $m > n$
 $n_{\text{accepted}} = n$; $v_{\text{accepted}} = v$;
 return **ack**
else return **abort**

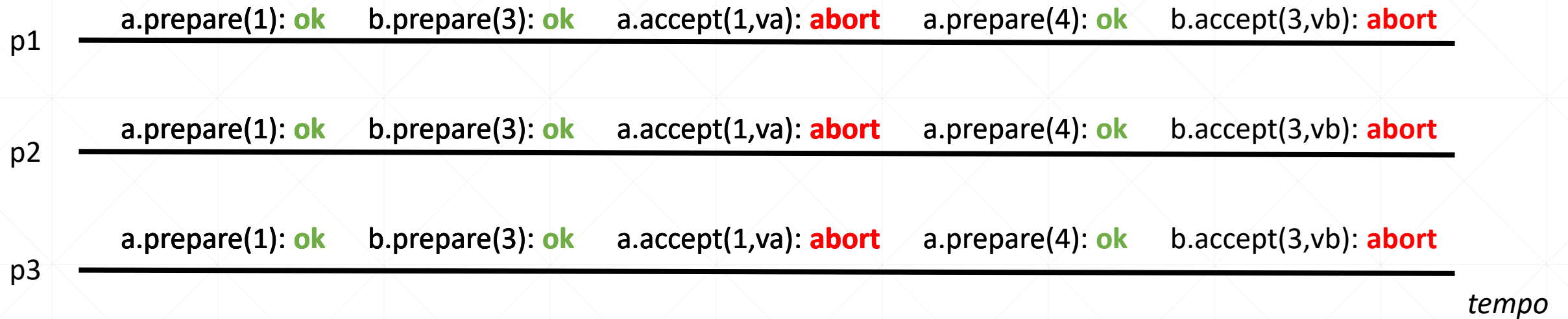
3.4. Consenso Paxos: caso normal



3.4. Consenso Paxos: caso normal com segundo aceito

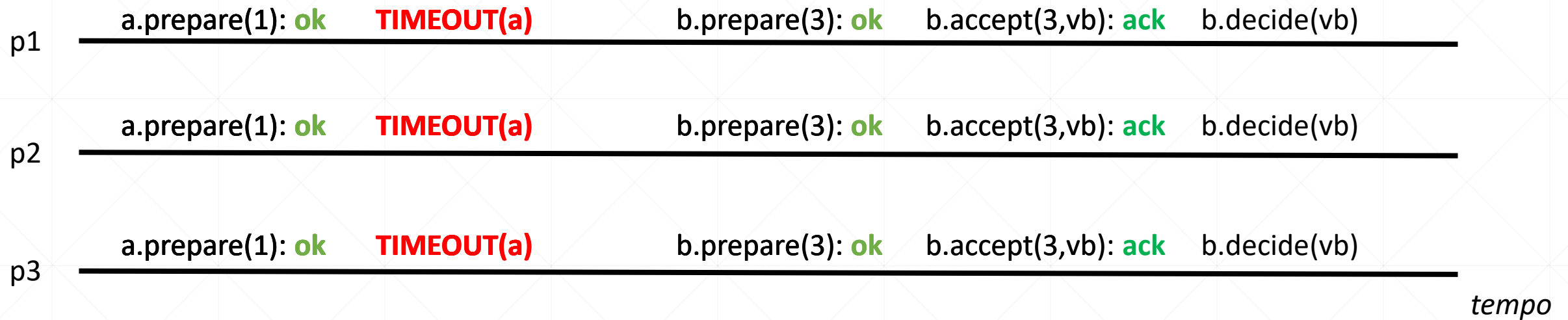


3.4. Consenso Paxos: caso de ciclo infinito



3.4. Consenso Paxos: caso de falha do primeiro proposer

Timeout e volta ao caso normal



Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - * Consenso em Transações Distribuídas
 4. Conclusão
-

Lembrando do Capítulo 1



CockroachDB
[2015 -]



Cloud Spanner
[2012 -]

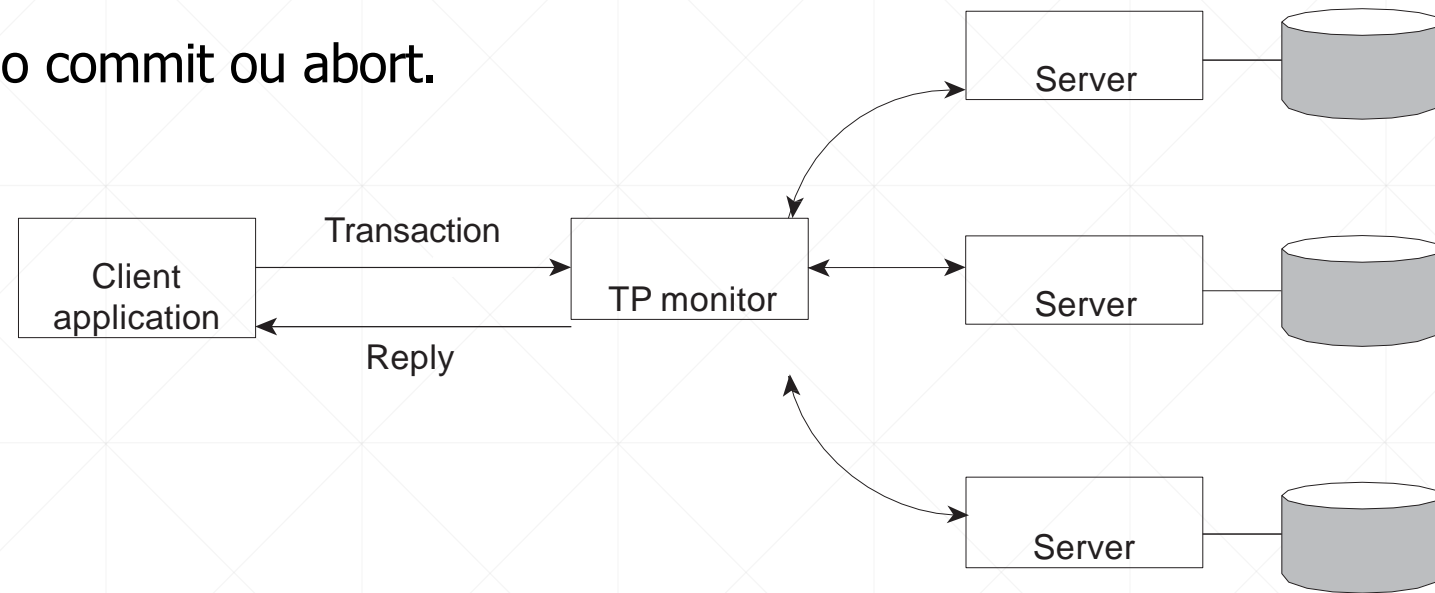
- Monitor de Processamento de Transações

Um **TP Monitor** é responsável por coordenar a execução de uma transação.

- Iniciar/quebrar em subtransações/finalizar

- Onde está o consenso?

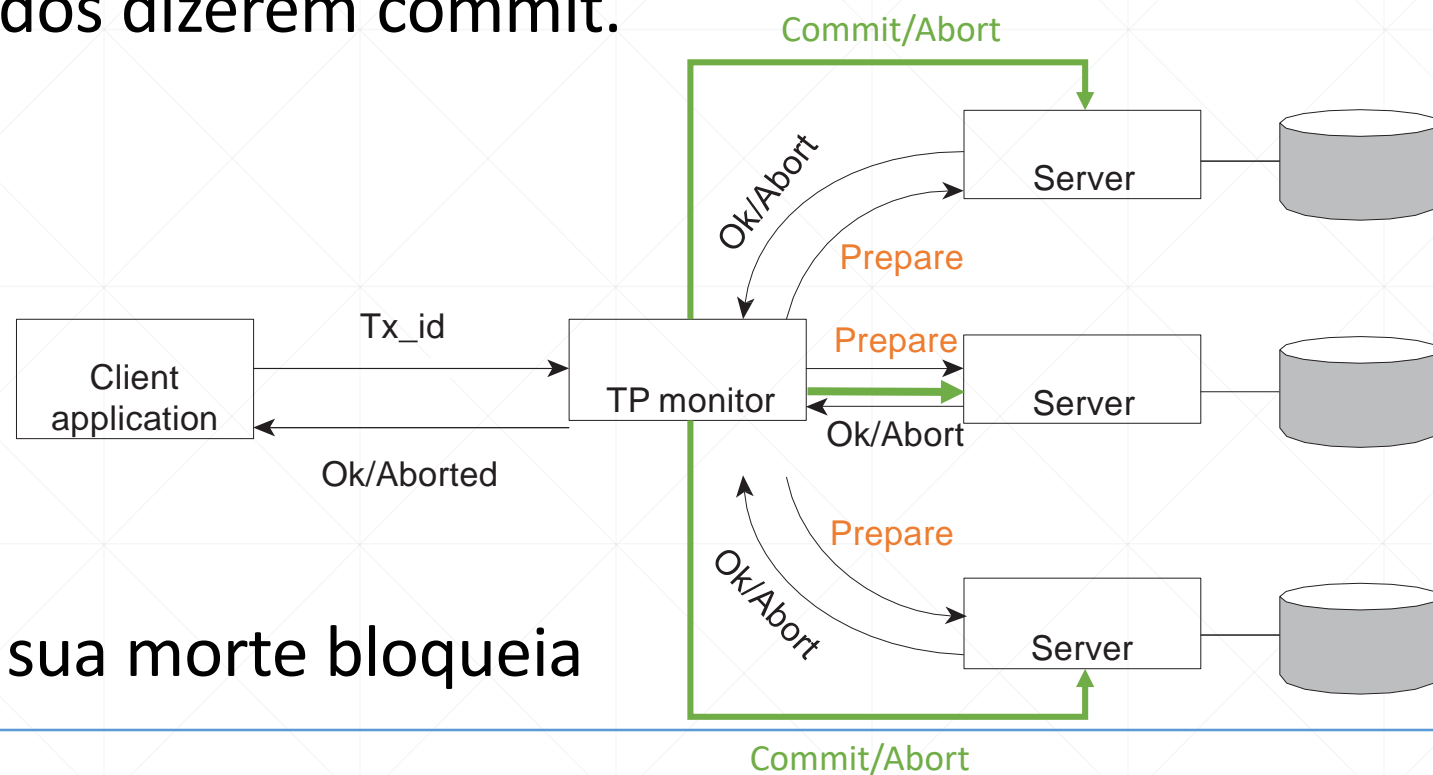
Todos devem concordar no commit ou abort.



D
i
f
f
e
r
e
n
t
e
r
s

Consenso 2PC

- Protocolo Two Phase Commit (2PC) para transações distribuídas
- Coordenador (TP Monitor) realiza duas fases: **prepare** – **commit/abort**.
- Basta 1 server dizer abort que todos devem abortar.
- Só fará commit se todos disserem commit.



- TP Monitor é SPoF e sua morte bloqueia servidores e cliente.

D
i
f
e
r
e
n
t
e
s

Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - * Teorema CAP
 4. Conclusão
-

Teorema CAP

- No ano 2000 Eric Brewer mostrou que, em sistemas distribuídos que replicam dados, é possível ter duas das três propriedades citadas abaixo [Tanenbaum17]:

Consistency (consistência): os dados replicados aparecem como se houvesse somente uma única cópia.

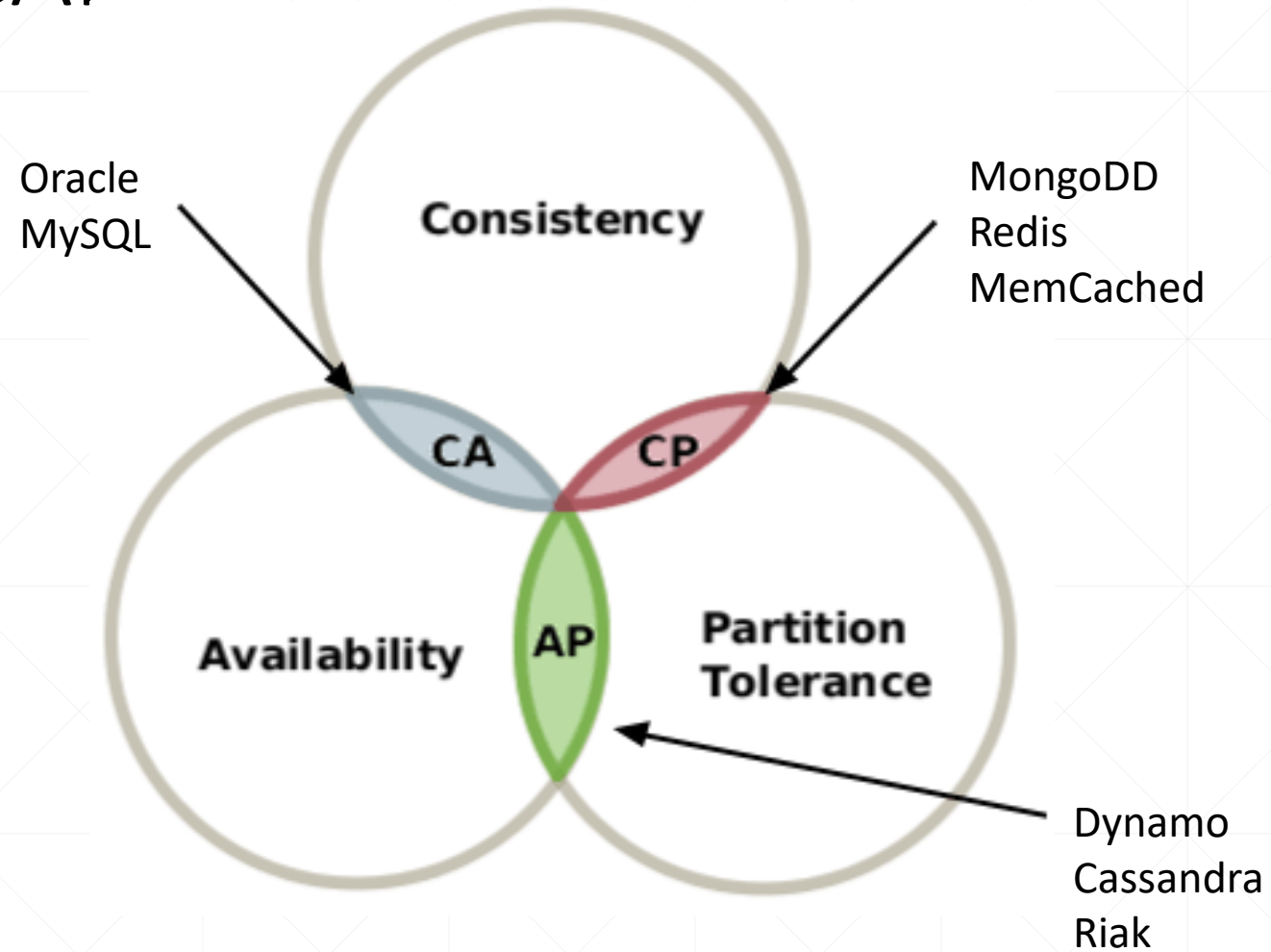
Availability (disponibilidade): todas as atualizações são eventualmente executadas.

Partitioning network (particionamento da rede): tolerante a particionamento em grupos de processos, dado a falhas da rede.

Teorema CAP

- Dado o cenário mais normal na Internet, onde há particionamento, segundo o CAP você precisa escolher entre consistência e disponibilidade:
 1. Exemplo de um banco (transações financeiras), o que você escolheria?
 2. Exemplo de uma reserva de passagens aéreas, o que você escolheria?
 3. No caso do Dynamo, o que foi escolhido?
 4. No caso do Paxos, o que foi escolhido?
 5. No caso da Blockchain proof of work, o que foi escolhido? (Capítulo Blockchain)
-

Teorema CAP

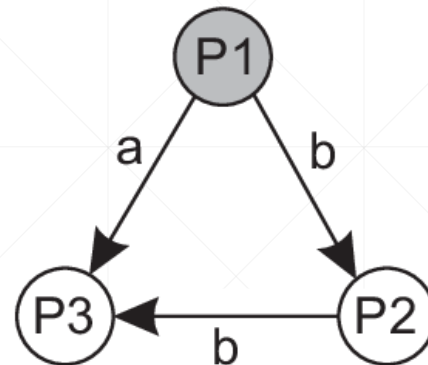
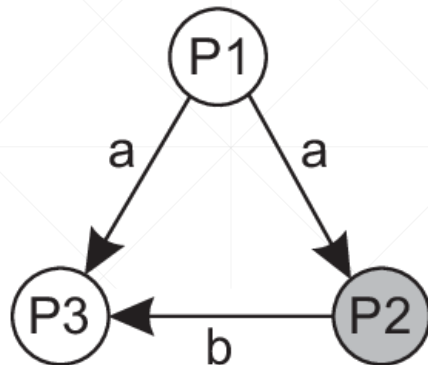


Agenda

1. Contexto do Consenso
 2. Revisão e novos Conceitos
 3. Consenso
 - 3.1. Especificação
 - 3.2. Consenso no modelo assíncrono
 - 3.3. Consenso no modelo síncrono
 - * **Comportamento Bizantino**
 4. Conclusão
-

Comportamento Bizantino

- Um processo com falha bizantina é aquele que apresenta um comportamento errôneo ou malicioso.
 - Encaminha mensagens corruptas.
 - Envia diferentes informações a diferentes processos.
- Seria possível obter o consenso sob falhas bizantinas usando Paxos?
 - Olhemos do ponto de vista de três proposers.



Conceitos adquiridos

- Two generals' problem.
- Consenso e propriedades: Safety & Liveness.
- Modelos de sistema: Síncrono, Assíncrono, Parcialmente síncrono.
 - Global Stabilization Time GST
- Modelos baseado na falha: Crash, Omission, Recovery, Byzantine.
- FLP: impossibilidade do consenso no modelo assíncrono
- Paxos: possibilidade do consenso no modelo parcialmente síncrono e síncrono
- 2PC – Two Phase Commit
- Teorema CAP – Consistency, Availability, network Partitioning.

Referências

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM* 32, 2, 374-382. 1985.

[Coulouris11] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed Systems: Concepts and Design* (5th ed.). Addison-Wesley Publishing Company, USA. 2011.

[Tanenbaum17] A. Tanenbaum and M. Steen. *Distributed Systems: Principles and Paradigms (3rd Edition)*. Person Education, Inc. 2017.

[Paxos01] L. Lamport. Paxos Made Simples. *ACM SIGACT News (Distributed Computing Column)*. Microsoft. 2001.