GBC074 - Sistemas Distribuídos

Consenso

Paulo Coelho
Universidade Federal de Uberlândia

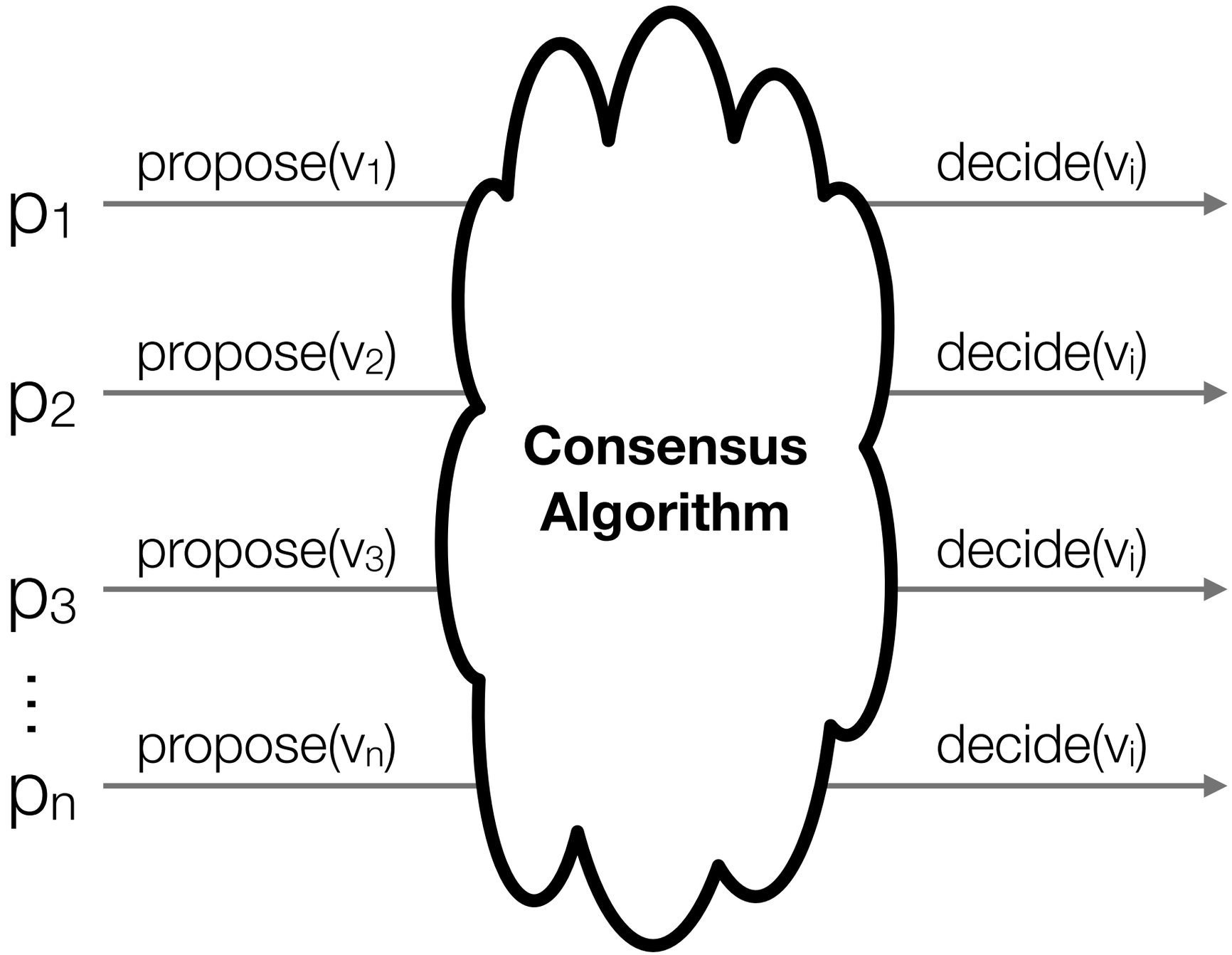
Introdução

- Replicação de processos
  - ✦ Processos devem entrar em acordo quanto a uma sequência de comandos a serem processados por todos
- Problemas de acordo
  - ✦ Processos devem concordar em alguma coisa,
  - ✦ Solução depende do modelo computacional:
    - De triviais a impossíveis.
- Difusão atômica e consenso distribuído são problemas equivalentes

Consenso - modelo

- ✦Considere um conjunto $\Sigma = \{ p_1, \ldots, p_n \}$ de *n* processos
- ✦Processos se comunicam exclusivamente por meio de troca de mensagens
  - ✦ Primitivas **send(m)** and **receive(m)**
- ✦Assume-se o modelo *crash failure*
  - ✦ Um processo *correto* nunca falha
  - ✦ Um processo faltoso eventualmente falha
- Existem *f* processos faltosos

Consenso - especificação



Consenso - especificação

- Propriedades
  - ✦ **Integridade uniforme**: se um processo decide um valor **v**, então **v** foi previamente proposto por algum processo
  - ✦ **Acordo uniforme**: dois processos não podem decidir de modo diverso
  - ✦ **Terminação**: todo processo correto eventualmente decide exatamente um valor

Consenso em sistemas síncronos

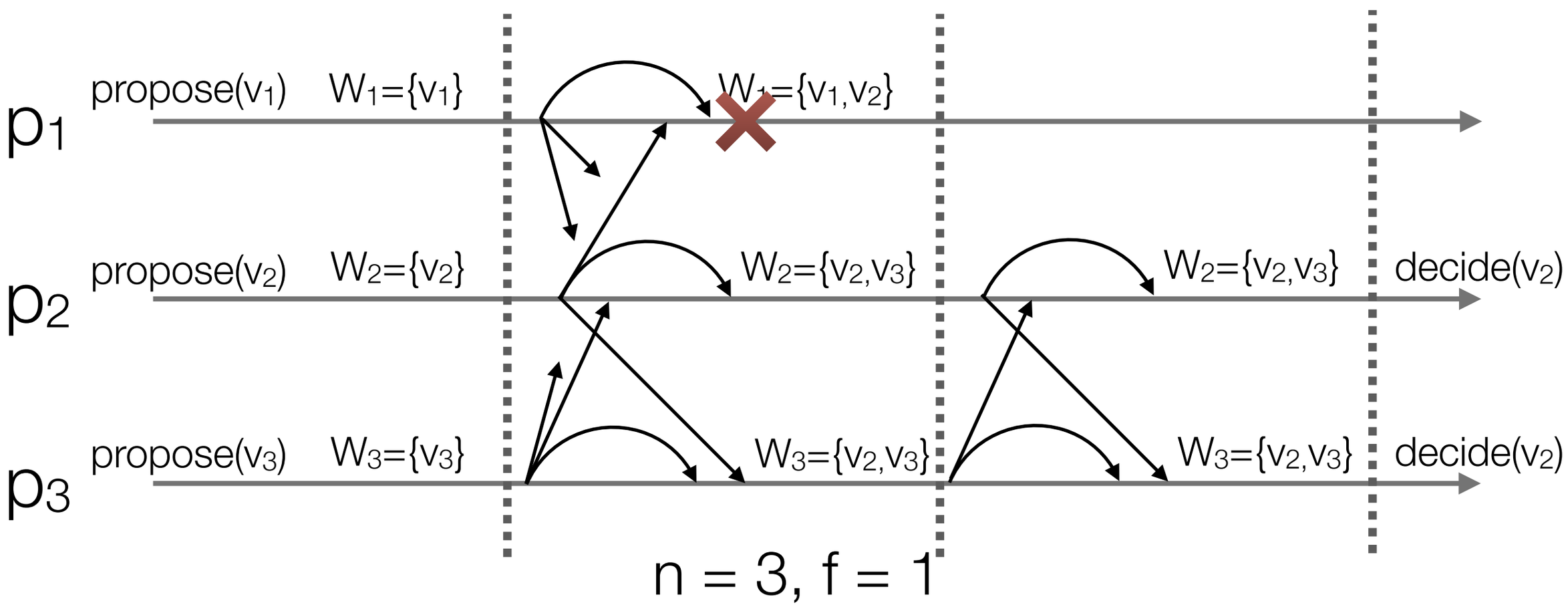
- Velocidade relativa dos processos e atrasos de mensagens são delimitados
- Execução procede em sequência de rodadas
  - ✦ Em cada rodada:
    - Processos geram nova mensagem a partir do estado atual e a enviam a todos os processos
    - Ao final, aplicam função de transição de estado utilizando o estado atual e mensagens obtidas para definir novo estado
    - Se um processo falhar no meio de envio de mensagem, apenas um subconjunto dos processos pode receber a mensagem

Consenso em sistemas síncronos

- Algoritmo 1:
  - ✦ Cada processo mantém variável **W** contendo subconjunto de **V**, o conjunto de todos os valores possíveis
  - ✦ Inicialmente, cada processo p_i possui apenas seu próprio valor inicial em **W**
  - ✦ Em cada rodada, cada processo p_i difunde (*broadcasts*) sua variável **W** e, ao final, adiciona todos os elementos recebidos ao seu próprio **W**
  - ✦ Depois de f+1 rodadas, p_i escolhe um elemento **v** de **W** de maneira determinístico e decide **v**

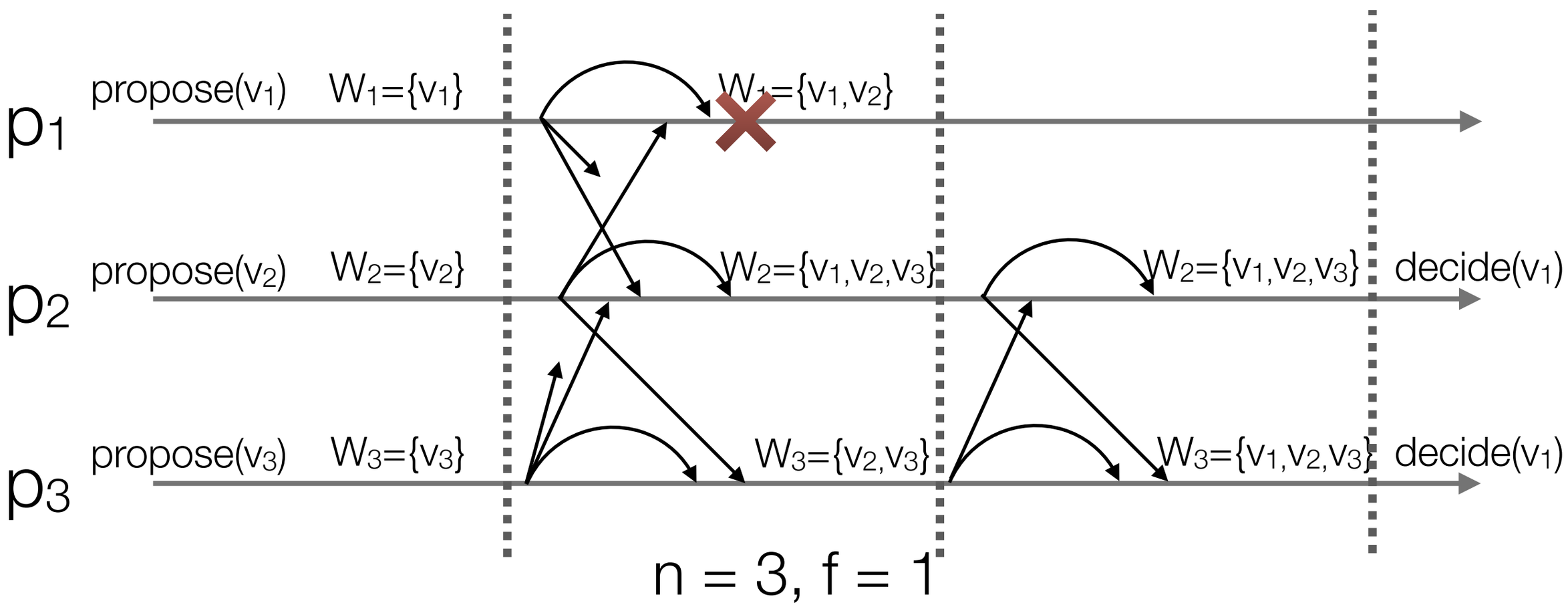
Consenso em sistemas síncronos

- Exemplo de execução
  - ✦ Por que precisamos de **f+1** rodadas?



Consenso em sistemas síncronos

- Exemplo de execução
  - ✦ Por que precisamos de **f+1** rodadas?



Consenso em sistemas síncronos

- Corretude:
  - ✦ Lema 1: Se nenhum processo falha durante uma rodada **r**, **1 ≤ r ≤ f+1**, então $\mathbf{W_i(r) = W_j(r)}$ para todo $\mathbf{p_i}$ e $\mathbf{p_j}$ ativos após **r** rodadas
  - ✦ Lema 2: Suponha que $\mathbf{W_i(r) = W_j(r)}$ para todo $\mathbf{p_i}$ e $\mathbf{p_j}$ ativos após **r** rodadas.
    Neste caso, para toda rodada **r'**, **r ≤ r' ≤ f+1**, o mesmo ocorrerá
  - ✦ Lema 3: Se processos $\mathbf{p_i}$ e $\mathbf{p_j}$ estão ativos após **f+1** rodadas, então $\mathbf{W_i(r) = W_j(r)}$ ao final da rodada **f+1**

Consenso em sistemas assíncronos

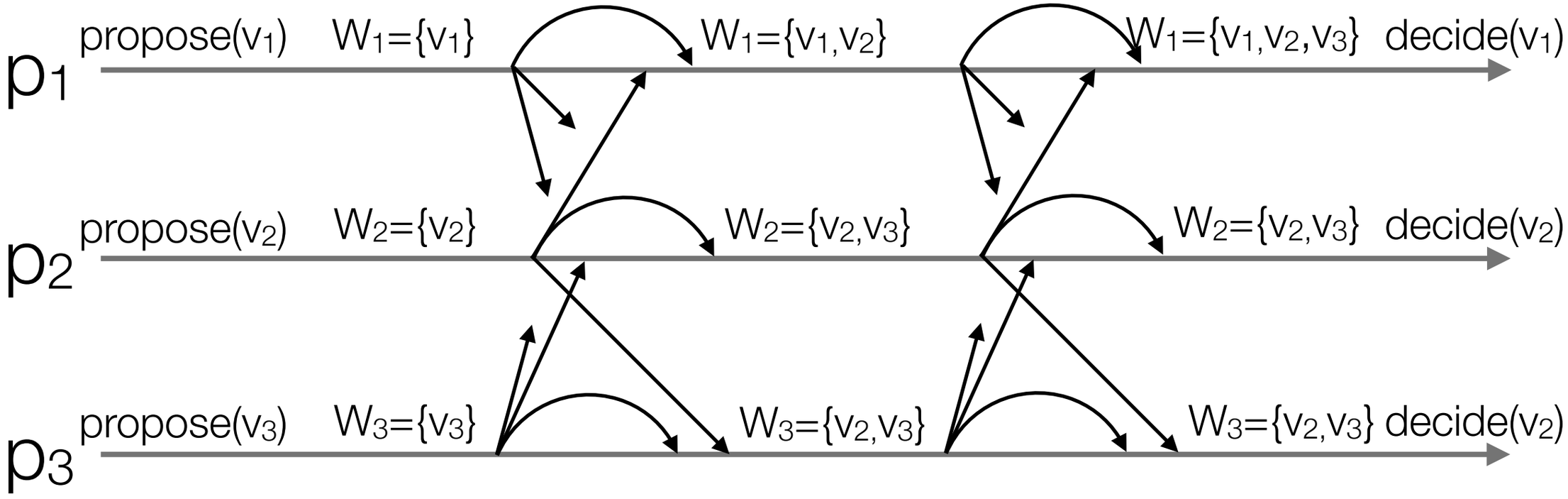
- Sem limite para velocidade relativa dos processos e atrasos na mensagens
- Execução em sequência de rodadas.

✦ Em cada rodada:

- Processos geram nova mensagem a partir do estado atual e a enviam a todos os processos
- Esperam por ***n-f*** respostas e aplicam função de transição de estado utilizando o estado atual e mensagens obtidas para definir novo estado
- Se um processo falhar no meio de envio de mensagem, apenas um subconjunto dos processos pode receber a mensagem

Consenso em sistemas assíncronos

- Algoritmo
  - ✦ Algoritmo anterior ainda funciona?



n = 3, f = 1

Consenso em sistemas assíncronos

- Resultado de impossibilidade de FLP
  - ✦ “Impossibility of distributed consensus with one faulty process”, by Fischer, Lynch, and Paterson, JACM 1985
  - ✦ Nenhum algoritmo consegue resolver consenso em um sistema assíncrono sujeito a falhas
- ✦ Resultado fundamental em computação distribuída
- ✦ Válido para $f \geq 1$, independente do valor de n

Consenso em sistemas assíncronos

- Impossibilidade de FLP: considere o consenso binário
  - ✦ Pela validade:
    - se todos as propostas são 1, a decisão deve ser 1
    - se todos as propostas são 0, a decisão deve ser 0
  - ✦ Se começarmos com todos os valores 1 e formos trocando um-a-um por 0, em algum momento sairemos de uma entrada que necessariamente leva a 1 para uma que pode levar a 0
  - ✦ Uma execução em que o último valor alterado pertence a um processo correto/falho pode ser construída de forma que leve a uma decisão 1/0
  - ✦ Como os processos não tem certeza se ele falhou o não, ambas as decisões deve ser possíveis neste cenário, pois são indistinguíveis
  - ✦ Logo, há um estado bivalente, decidido pela troca de mensagens e não pelos valores iniciais
  - ✦ Dado um estado bivalente, sempre é possível forçar um próximo estado bivalente

Resolvendo consenso

- Sistemas síncronos são muito fortes…
  e assíncronos não possuem uma solução
- Para resolver consenso, pode-se
  - ✦ Fortalecer as considerações sobre o modelo
  - ✦ Enfraquecer a definição do problema
  - ✦ Fazer as 2 coisas acima

Consenso - fortalecendo o modelo

- Sistema parcialmente síncrono
  - ✦ Sistema assíncrono que eventualmente se torna síncrono
  - ✦ Global Stabilization Time (GST)
    - Tempo a partir do qual sistema se torna síncrono
    - Desconhecido para os processos
  - ✦ Questão fundamental:
    - É possível criar um algoritmo de consenso que nunca viola *safety* enquanto o sistema é assíncrono e garante *liveness* quando condições adicionais são satisfeitas?

Consenso com GST

- Dwork, Lynch and Stockmeyer, “Consensus in the presence of partial synchrony”, JACM 1988
  - ✦ Modelo de rodadas sobre sistemas parcialmente síncronos
  - ✦ Em cada rodada: envio, recebimento, transição de estado
  - ✦ **Antes** do GST mensagens pode ser perdidas
  - ✦ **Após** GST mensagens entre processos corretos não se perdem

Consenso com GST

- n = 3*f + 1 processos

Initialization

v ← process p's proposed value

Round r:

send step

send v to all processes

state transition step

if messages received ≥ n - f **then**

v ← most often received value, if not unique take the smallest

if at least n-f values received equal to x and not decided **then**

decide x

Consenso com detectores de falhas

- Consenso também pode ser resolvido em sistemas assíncronos com detectores de falhas
  - ✦ Classes de detectores de falhas

Completeness	**Accuracy**			
	Strong	*Weak*	*Eventual Strong*	*Eventual Weak*
Strong	Perfect, $\mathscr{P}$	Strong, $\mathscr{S}$	Eventually Perfect, $\Diamond\mathscr{P}$	Eventually Strong, $\Diamond\mathscr{S}$
Weak	$\mathscr{Q}$	Weak, $\mathscr{W}$	$\Diamond\mathscr{Q}$	Eventually Weak, $\Diamond\mathscr{W}$

Consenso com detectores de falhas

- Chandra e Toueg: resolvendo consenso com ◇S
  - ✦ n = 2*f + 1 processos

Initialization
v ← proposed value
r ← 0
t ← 0

while undecided **do**
… (next slide) …

upon receiving (decide, w)
p sends (decide, w) to all
p decides on w and halts

Consenso com detectores de falhas

while undecided **do**

phase 1

c ← (r mod N) + 1

r ← r + 1

send (p,r,v,t) to c

phase 2

c waits for first ⌈(N+1)/2⌉ estimates

c chooses the estimate w with the largest timestamp

c proposes value (c,r,w)

phase 3

p waits for a proposal or suspects c

if proposal (c,r,w) is received **then**

v ← w; t ← r

send (r, ACK) to c

else

send (r, NACK) to c

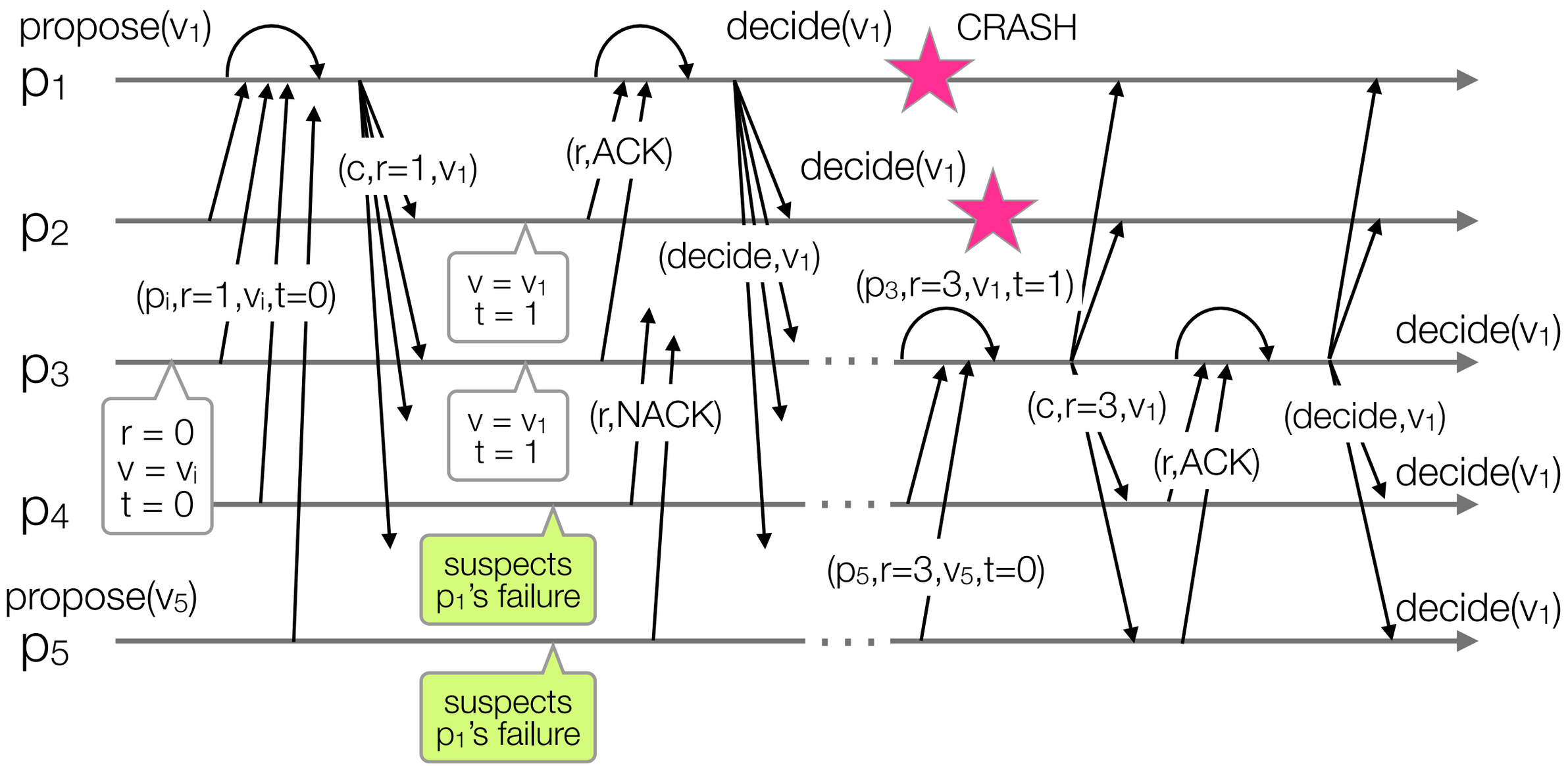
phase 4

c waits for ⌈(N+1)/2⌉ ACK/NACK messages

if there are ⌈(N+1)/2⌉ ACK messages **then**

c sends (decide, w) to all

Consenso com detectores de falhas



Consenso com detectores de falhas

- O algoritmo de consenso para ◇S é ótimo em relação a tolerância a falhas
- Consenso não pode ser resolvido com ◇P em sistemas assíncronos e f > n/2 [CT96]

Consenso com eleição de líderes

- Sistema assíncrono com eleição de líderes
- n = 2*f+1
- Paxos (Lamport)

✦ Quatro papéis:

- *Proposers*
- *Acceptors*, um quorum Q_a de acceptors, onde $|Q_a| > n/2$
- *Learners*
- *Coordinator/Leader*

Consenso com eleição de líderes

- Paxos
  - ✦ Oráculo de eleição de líder
    - Cada processo **p** tem acesso a um oráculo de eleição de líder que retorna um processo denominado $\mathbf{leader_p}$ tal que existe:
    - (a) um processo correto **l**, e
    - (b) um momento a partir do qual, para todo processo **p**, tem-se $\mathbf{leader_p = l}$

Paxos

Variables

c-rnd : highest-numbered round the process has started
c-val : value the process has picked for round c-rnd
rnd : highest-numbered round the acceptor has participated, initially 0
v -rnd : highest-numbered round the acceptor has cast a vote, initially 0
v-val : value voted by the acceptor in round v-rnd, initially null

Phase 1A
Proposer

To propose value v:
    increase c-rnd to an arbitrary unique value
    send (PHASE 1A, c-rnd) to acceptors

Phase 1B
Acceptor

upon receiving (PHASE 1A, c-rnd) from proposer
    if c-rnd > rnd **then**
        rnd ← c-rnd
        send (PHASE 1B, rnd, v-rnd, v-val) to proposer

Paxos

Phase 2A Proposer

upon receiving (PHASE 1B, rnd, v-rnd, v-val) from Q_a such that c-rnd = rnd
    k ← largest v-rnd value received
    V ← set of (v-rnd,v-val) received with v-rnd = k
    if k = 0 **then** let c-val be v
    else c-val ← the only v-val in V
    send (PHASE 2A, c-rnd, c-val) to acceptors

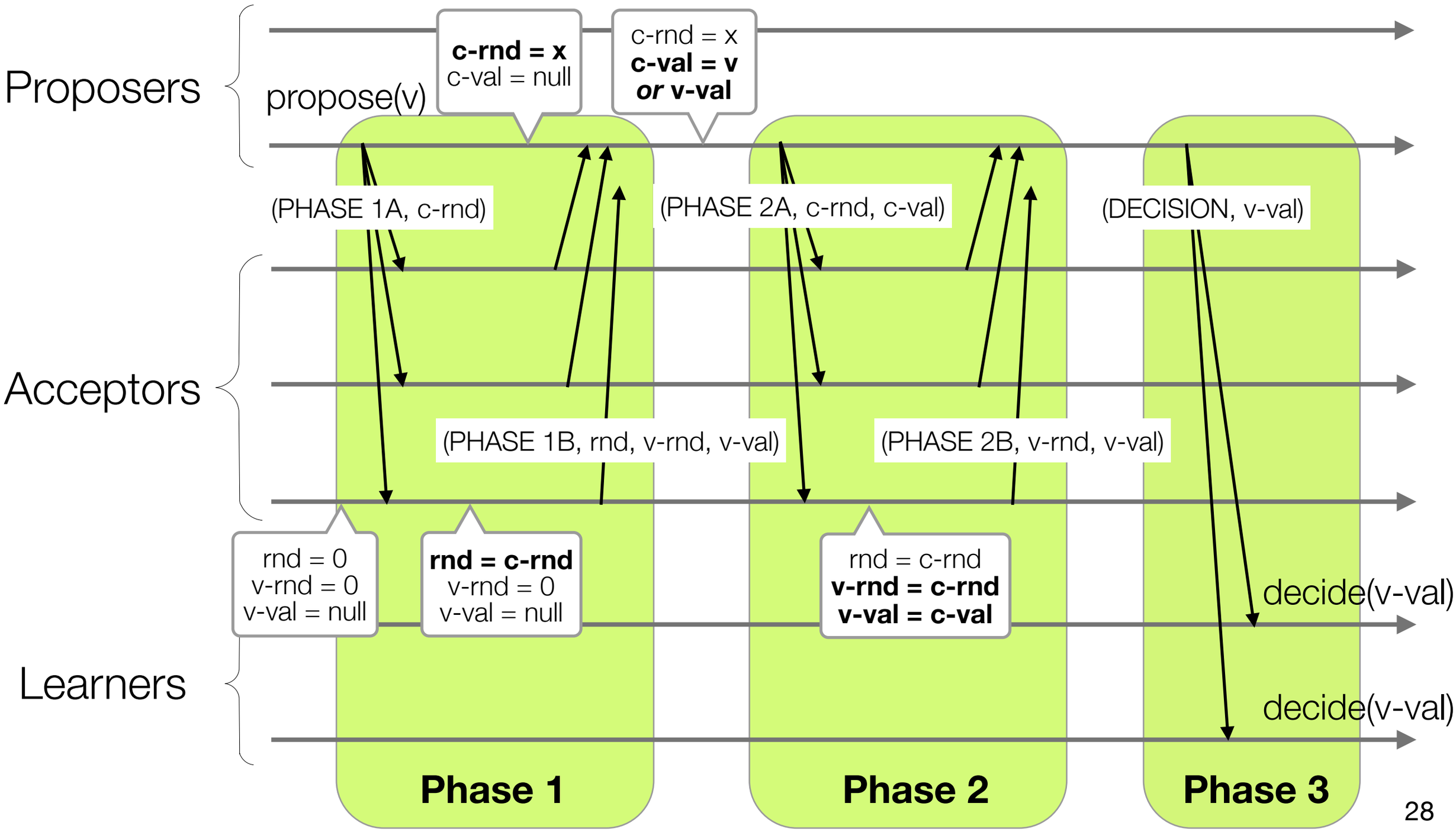
Phase 2B Acceptor

upon receiving (PHASE 2A, c-rnd, c-val) from proposer
    if c-rnd ≥ rnd **then**
        v-rnd ← c-rnd
        v-val ← c-val
        send (PHASE 2B, v-rnd, v-val) to proposer

Phase 3 Proposer

upon receiving (PHASE 2B, v-rnd, v-val) from Q_a
    if for all received messages: v-rnd = c-rnd **then**
        send (DECISION, v-val) to learners

Paxos


Proposers
Acceptors
Learners
propose(v)
c-rnd = x
c-val = null
c-rnd = x
c-val = v
or v-val
(PHASE 1A, c-rnd)
(PHASE 2A, c-rnd, c-val)
(DECISION, v-val)
(PHASE 1B, rnd, v-rnd, v-val)
(PHASE 2B, v-rnd, v-val)
rnd = 0
v-rnd = 0
v-val = null
rnd = c-rnd
v-rnd = 0
v-val = null
rnd = c-rnd
v-rnd = c-rnd
v-val = c-val
decide(v-val)
decide(v-val)
Phase 1
Phase 2
Phase 3

Paxos

- Corretude (intuição para garantia de *safety*)
  - ✦ Se *learner* **L** decide **v**, após receber **v** de *proposer* **P**
  - ✦ Então, **P** recebeu *(PHASE 2B, v-rnd=x, v-val=v)* de $\mathbf{Q_a}$
  - ✦ Seja **P'** outro *proposer* que envia (PHASE 1A, c-rnd') para quórum $\mathbf{Q_a'}$, tal que *c-rnd' > rnd* para todo *acceptor* em $\mathbf{Q_a'}$
  - ✦ Ao menos um *acceptor* **A** está em $\mathbf{Q_a} \cap \mathbf{Q_a'}$, logo **A** envia *(PHASE 1B, rnd, v-rnd, v-val)* para **P'**, onde *v-rnd = x* e *v-val = v*
  - ✦ Pelo algoritmo, **P'** escolhe **v** como seu valor proposto

Paxos

- Extensões e otimizações
  - ✦ Proposers envia propostas ao líder
  - ✦ Líder pode pre-executar Phase 1
  - ✦ Acceptors podem enviar Phase 2B diretamente para Learners
  - ✦ Latência do Paxos (melhor caso)
    - Dois passos de comunicação para líder
    - Três passos de comunicação para demais proposers

Consenso - enfraquecendo a definição

- *k-set agreement*
  - ✦ Acordo: **No máximo** *k* valores diferentes são decididos
  - ✦ Validade: um valor decidido é um valor proposto
  - ✦ Terminação: todos os processos corretos eventualmente decidem

Consenso - enfraquecendo a definição

- *k-set agreement*
  - ✦ Solução trivial se $f < k$
    - Primeiros f+1 processos enviam valores iniciais para todos
    - Um processo decide no primeiro valor que receber

Consenso - enfraquecendo a definição e fortalecendo o modelo

- Algoritmos randomizados
  - ✦ Mais forte que modelo assíncrono
    - Processos podem fazer escolhas aleatórias
  - ✦ Terminação fraca
    - Processos corretos decidem no tempo *t* com probabilidade de pelo menos *p(t)*