Algoritmos e Sistemas Distribuídos

Aula teórica 5

Ano lectivo 2014/2015

Mestrado Integrado em Engenharia Informática

Última aula: tempo mínimo para resolver o consenso

- Teorema: Não existe nenhum algoritmo que resolva o problema do consenso num sistema síncrono em menos de f+1 rounds, na presença de f falhas, usando n>f+1 réplicas

- Só demonstrámos para f=1
  - Provámos que é impossível resolver em um round

Similaridade

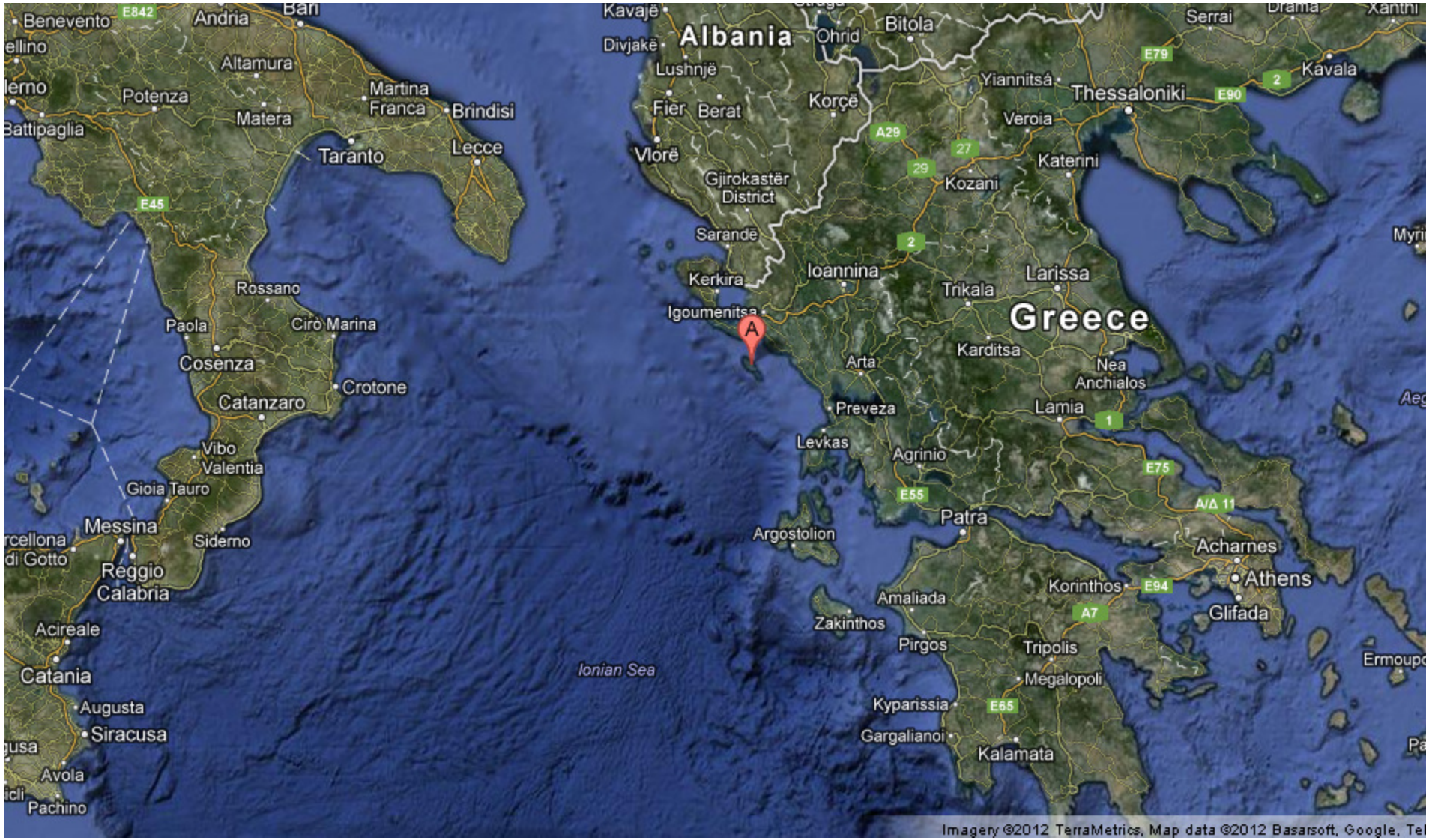
- Dadas duas execuções e1, e2 do mesmo algoritmo, dizemos que elas são similares para o processo i, denominado $e1 \sim_i e2$, se $e1|i=e2|i$
- $e1 \approx e2$ se existirem execuções x0, …, xk tais que: $e1 = x0 \sim_i x2 \sim_j \ldots \sim_k xk = e2$
- Lema: se $e1 \approx e2$ então $dec(e1)=dec(e2)$

Impossibilidade do consenso num sistema assíncrono

- Resultado fundamental (FLP):
- Não existe nenhum protocolo determinístico para resolver o consenso num sistema assíncrono em que um único processo possa ter uma falha por crash
  - Fisher, Lynch, and Paterson. Impossibility of distributed consensus with one faulty process. JACM, Vol. 32, no. 2, April 1985, pp. 374-382

Hoje: Contornar o FLP

- Enfraquecer a propriedade de terminação
  - Apenas garantir terminação em períodos de sincronia: se a rede entregou as mensagens atempadamente durante um certo intervalo
- Algoritmo: Paxos
  - Usado, por exemplo, por Google, Yahoo!, Microsoft, Amazon, etc


Albania
Greece
Ionian Sea
Benevento
Andria
Bari
Altamura
Potenza
Matera
Martina Franca
Brindisi
Taranto
Lecce
Battipaglia
Rossano
Paola
Ciro Marina
Cosenza
Crotone
Catanzaro
Vibo Valentia
Gioia Tauro
Messina
Siderno
Reggio Calabria
Acireale
Catania
Augusta
Siracusa
Avola
Pachino
Kavajë
Divjakë
Lushnjë
Fier
Berat
Vlorë
Korçë
Ohrid
Bitola
Gjirokastër District
Sarandë
Kerkira
Igoumenitsa
Ioannina
Arta
Preveza
Levkas
Agrinio
Patra
Argostolion
Amaliada
Zakinthos
Pirgos
Kyparissia
Gargalianoi
Kalamata
Tripolis
Megalopoli
Yiannitsá
Thessaloniki
Veroia
Katerini
Kozani
Serrai
Kavala
Trikala
Larissa
Karditsa
Nea Anchialos
Lamia
Korinthos
Acharnes
Athens
Glifada
Ermoupo
E842
E45
E55
E65
E75
E79
E90
E94
A29
A7
A/Δ 11
Imagery ©2012 TerraMetrics, Map data ©2012 Basarsoft, Google, Tel

Pressupostos do modelo

- Sistema assíncrono
- Mensagens podem-se perder e ser duplicadas, mas nunca corrompidas
- Processos podem falhar por crash, e depois recuperar
  - Cada processo tem acesso a armazenamento persistente

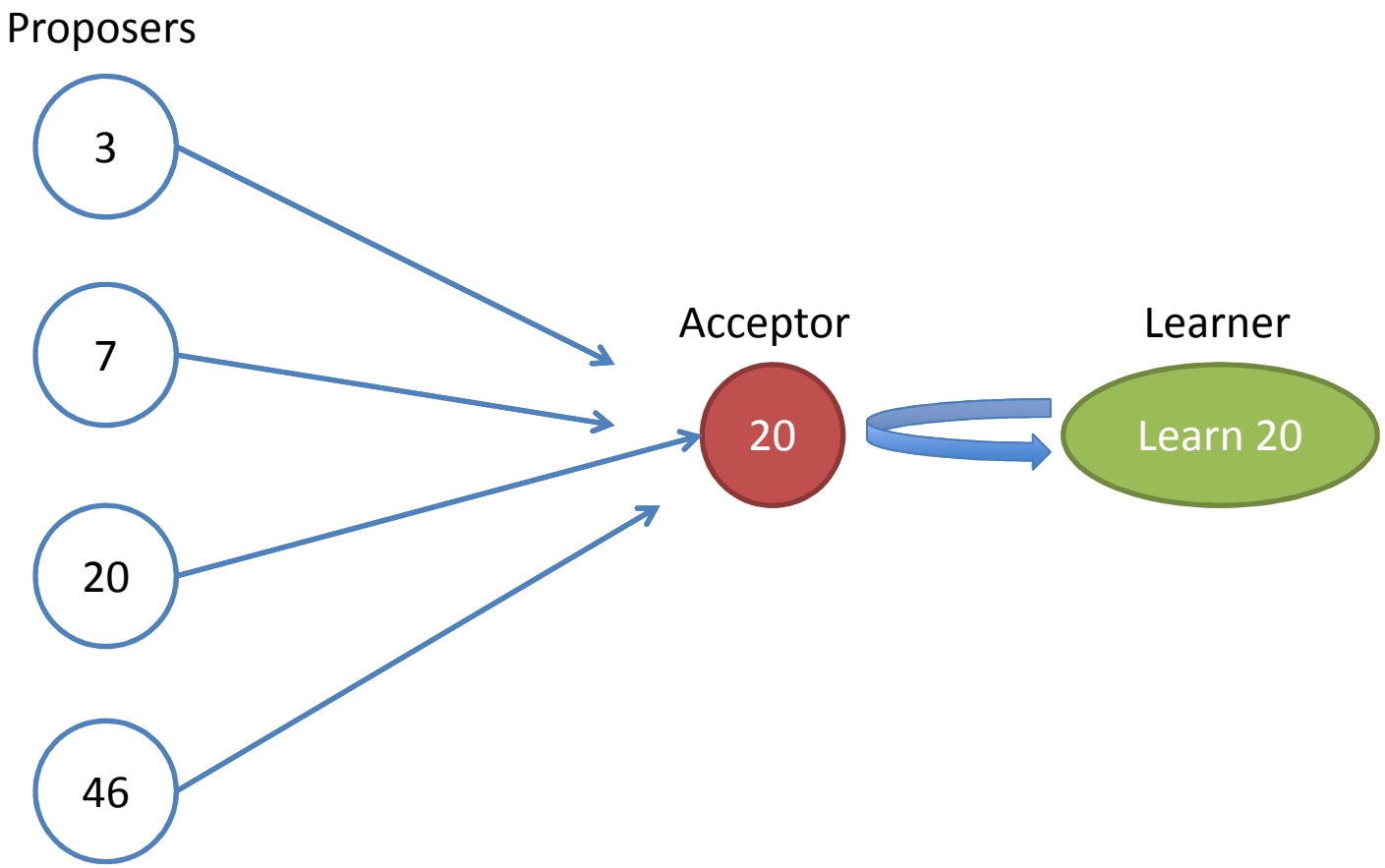
Especificação (safety)

- CS1 [Validade] Se todos os processos tiverem a mesma proposta v, então v é o único valor que pode ser decidido
- CS2 [Acordo] Dois processos correctos não podem decidir valores diferentes
- CS3 [integridade (+forte)] Se um processo correcto decide v, então v era a proposta de algum processo
  - Notar que CS3 => CS1, logo podemos retirar CS1

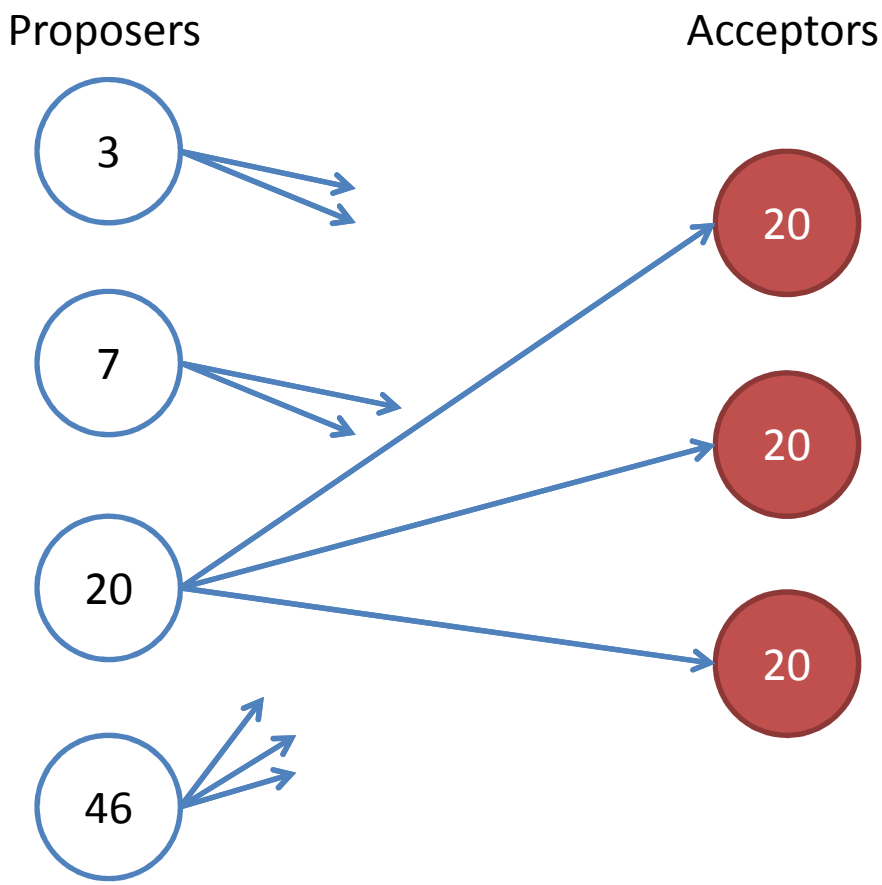
Separação de papéis

- 3 tipos de processos:
  - Proponentes (proposers)
  - Aceitantes (acceptors)
  - “Aprendedores” (learners)
- Na prática podem correr todos na mesma máquina → um processo com as três funções

Solução trivial: um único acceptor



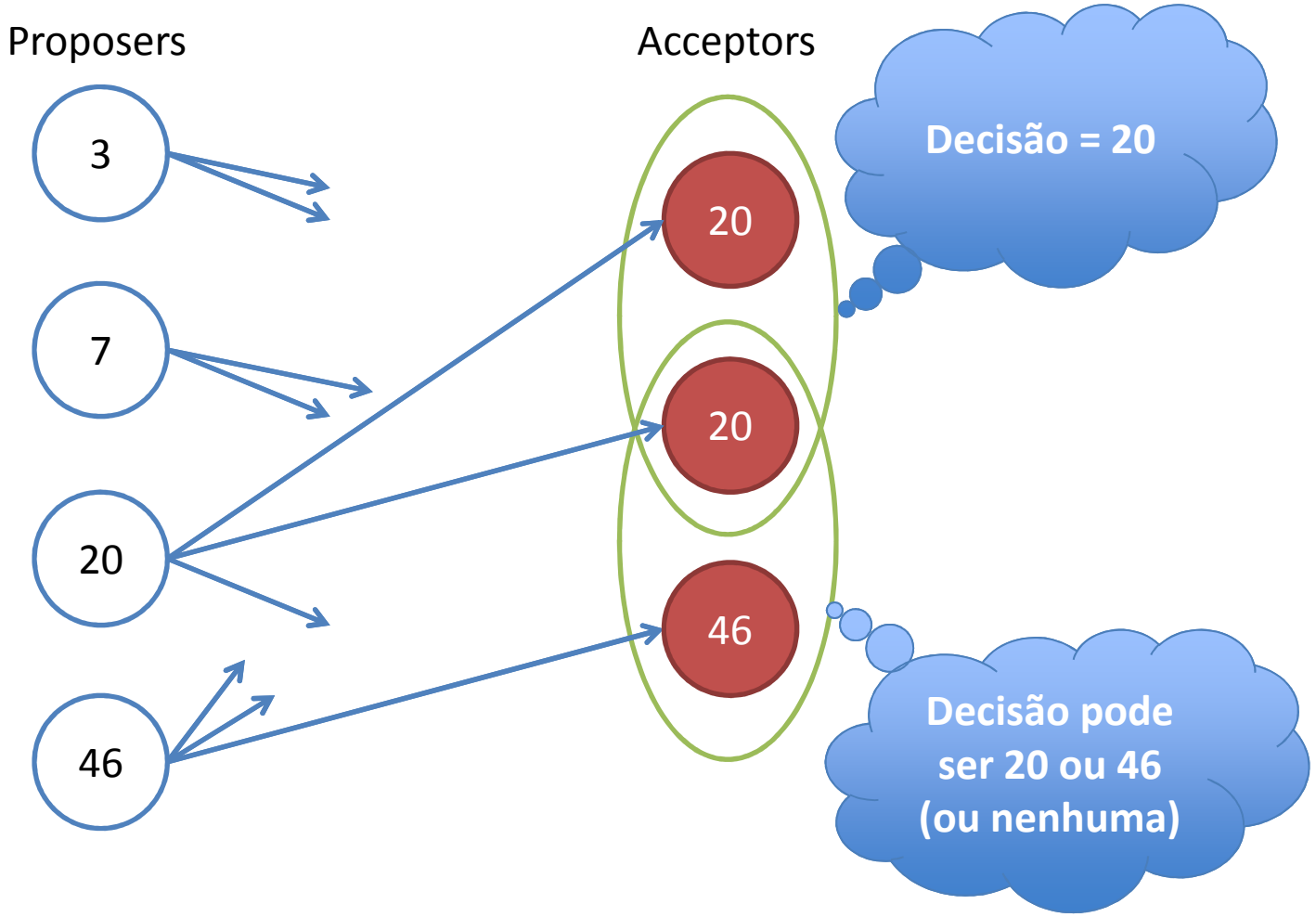
Tolerar falhas dos acceptors



Quóruns

- Problema: diferentes acceptors podem receber diferentes propostas → quando podemos escolher uma decisão final?
- Ideia base: a escolha só é feita quando uma maioria de acceptors aceitar um valor
- Generalização (quórum) – subconjuntos do conjunto de acceptors tais que quaisquer dois subconjuntos se intersectam

Maiorias / quóruns em acção



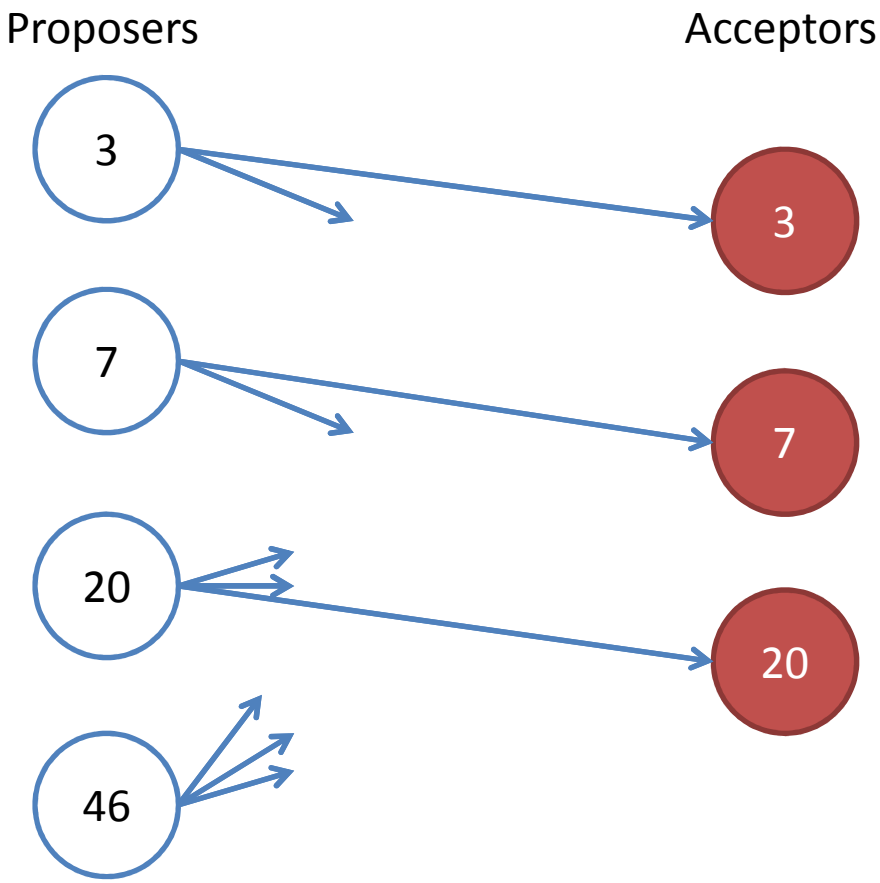
Requisitos da solução

- Supondo que é feita apenas uma proposta e não há falhas
- Temos que garantir que essa proposta é aceite

- P1: Um acceptor tem de aceitar a primeira proposta que recebe

- E se houverem propostas diferentes?

P1 + propostas diferentes



Como lidar com propostas diferentes?

- Acceptors têm de poder aceitar mais do que uma proposta/valor
- Cada proposta tem um número de sequência associado que permite distinguir e ordená-las
  - Proposta = (psn, valor)
  - Propostas diferentes têm psn diferente
  - Definição: uma proposta (ou seja, um par (psn,valor)) foi **escolhida** quando foi aceite por uma maioria de acceptors
    - Nessa altura os learners podem declarar a decisão final

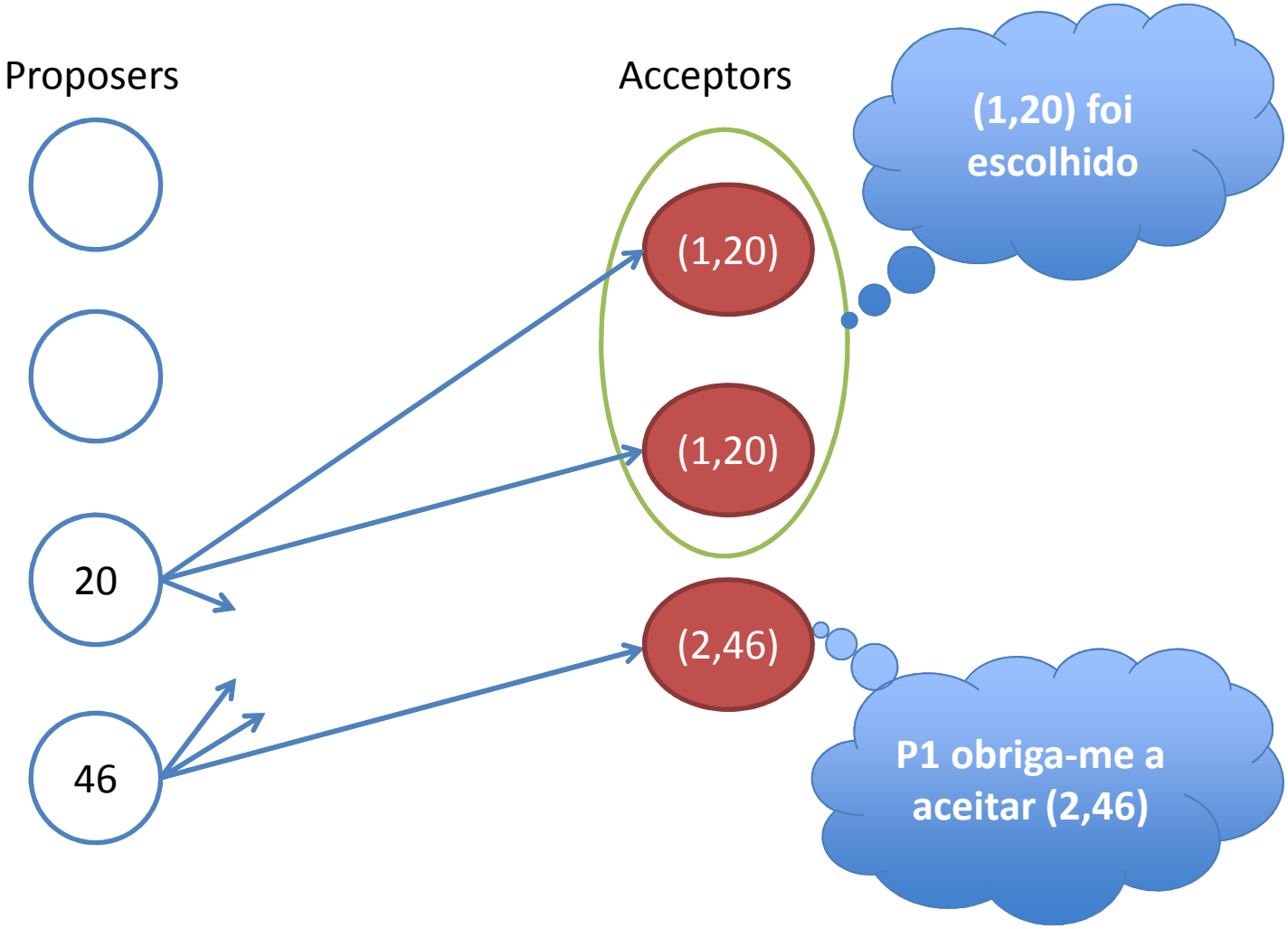
Escolher múltiplas propostas? Yes we can!

- Desde que todas elas tenham o mesmo valor
  - Esse valor é a decisão final do consenso
- Mais um requisito para a solução:
- P2: Se uma proposta com um valor v foi escolhida, então qualquer proposta subsequente (com um psn superior) que seja escolhida tem o valor v

Fortalecer P2

- Vamos definir outro requisito (P2a) que implica P2 e é mais próximo da implementação algorítmica
- P2: Se uma proposta com um valor v foi escolhida, então qualquer proposta subsequente (com um psn superior) que seja escolhida tem o valor v
- P2a: Se uma proposta com um valor v foi escolhida, então qualquer proposta subsequente (com um psn superior) que seja aceite por um acceptor tem o valor v

Dificuldade em conjugar P1 e P2a



Solução: onerar os proposers

- P2b: Se uma proposta com um valor v foi escolhida, então qualquer proposta subsequente (com um psn superior) que seja **emitida por um proposer** tem o valor v
- Notar que P2b implica P2a, que implica P2

Como garantir P2b?

- Supondo que p quer emitir proposta com psn número n
- Se p puder garantir que não existe n'<n que foi escolhida (ou possa vir a ser escolhida) então pode propor a sua proposta inicial
- Caso contrário tem de propor o valor que foi ou possa vir a ser escolhido. Como determinar tal valor?

Como decidir se existe n’<n que foi escolhida (ou possa vir a ser escolhida)

- Se já foi escolhida n’<n, então foi aceite por uma maioria S. Como qualquer outra maioria S’ intersecta S, basta encontrar uma maioria que ainda não aceitou nenhum n’<n
  - Nesse caso proposer usa a sua proposta inicial
- Mas e se n’ ainda vier a ser aceite?

Como decidir que não existe n’<n que foi (ou possa vir a ser) escolhida

- Se já foi escolhida n’<n, então foi aceite por uma maioria S. Como qualquer outra maioria S’ intersecta S, basta encontrar uma maioria que ainda não aceitou e não virá a aceitar nenhum n’<n
  - Nesse caso proposer usa a sua proposta inicial

E se >1 proposta foi aceite?

- Supondo que existem diferentes pares <psn,valor> aceites por diferentes acceptors
- Se algum foi (ou puder vir a ser) escolhido tem de ser o de psn maior
  - Caso contrário, num algoritmo que garanta P2b, a proposta de psn maior teria de ter um valor igual à proposta que foi (ou possa vir a ser) escolhida

Como garantir P2b?

- Prova-se que para garantir P2b, basta garantir:
- P2c: Para quaisquer v, n, se uma proposta com valor v e psn n é emitida, então existe uma maioria de acceptors S tal que:
  - Ou nenhum acceptor em S aceitou nem virá a aceitar nenhuma proposta com psn <n,
  - Ou v é o valor associado à proposta com psn mais alto que foi ou será aceite numa maioria (dentro das propostas com psn<n)

Como é que o algoritmo garante P2c?

- Antes de emitir proposta com psn=n, proposer contacta uma maioria de acceptors, pergunta qual a proposta com psn mais alto (mas inferior a n) que foi ou virá a ser aceite
- Se a nenhum acceptor aceitou (nem virá a aceitar) valor nenhum <n, proposer escolhe a sua proposta inicial
- Caso contrário, propõe o valor associado à proposta de psn mais alto que foi aceite

Dificuldade: prever o futuro

- É difícil prever quais os valores que possam vir a ser aceites
- Ao invés, ao perguntar quais as propostas aceites <n, também pedimos uma promessa que o acceptor não virá a aceitar nenhum valor <n
- Daí garantirmos que o proposer vai saber qual a proposta com psn mais alto (mas inferior a n) que foi **ou virá a ser** aceite

Algoritmo do proposer

```
PROPOSE(v)
  choose unique n, higher than any n seen so far
  send PREPARE(n) to all nodes
  if PREPARE_OK(na, va) from majority then
     va = va with highest na (or choose v otherwise)
  send ACCEPT (n, va) to all
  if ACCEPT_OK(n) from majority then
     send DECIDED(va) to all
```

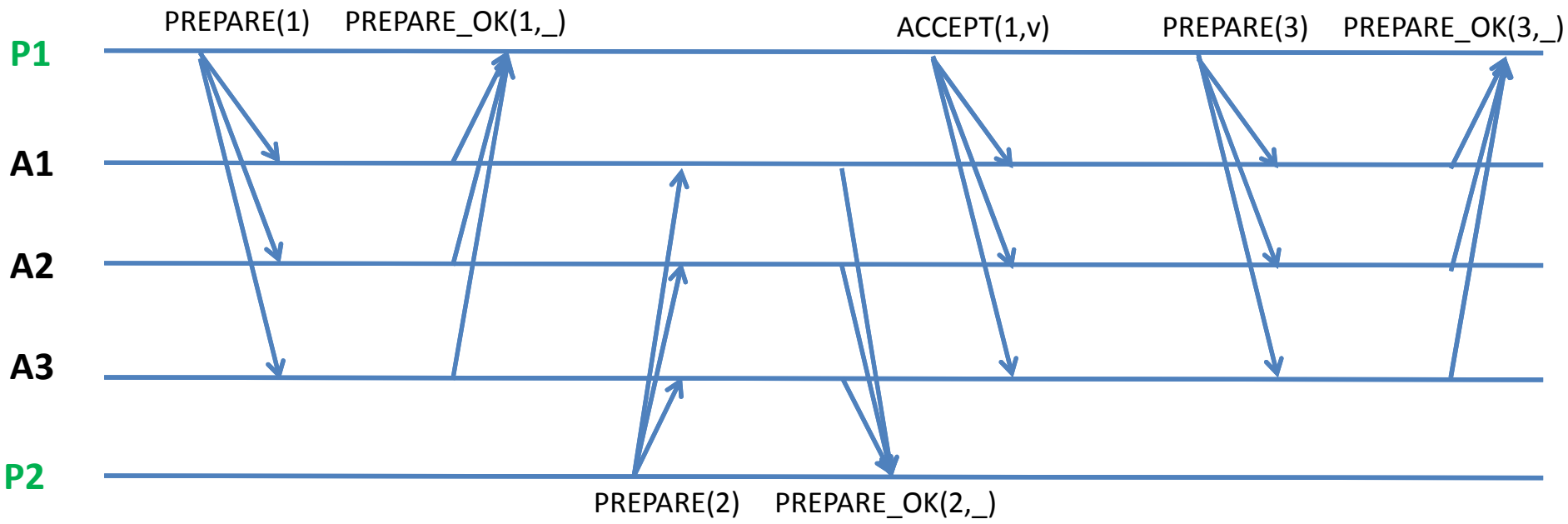
Algoritmo do acceptor

```
State: np (highest prepare), na, va (highest accept)
/* This state is maintained in stable storage */

PREPARE(n)
  if n > np then
     np = n   // will not accept anything <n
     reply <PREPARE_OK,na,va>

ACCEPT(n, v)
  if n >= np then
    na = n
    va = v
    reply with <ACCEPT_OK,n>
```

Aprendizagem

- Os learners têm de contactar os acceptors (ou ser contactados por eles) para saber quando um valor for escolhido

Liveness não está garantida



- Para garantir progresso, é necessário eleger um proposer único (líder) → só é garantidamente possível em períodos de sincronia

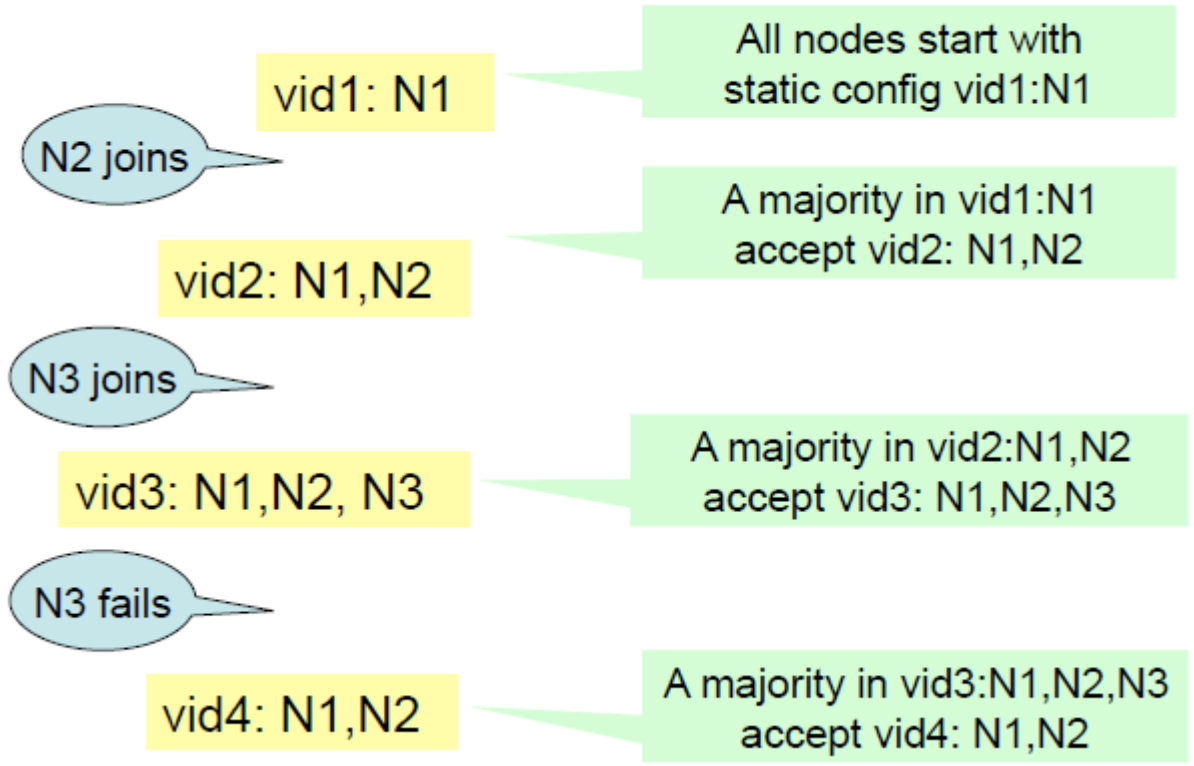
Implementar replicação de máquina de estados (Alternativa #1)

- Correr uma instância do Paxos por cada operação a ser executada
  - instância n usada para determinar a n-ésima operação a ser executada em cada réplica
- Cada réplica corre os 3 papéis do Paxos, e o conjunto de réplicas é fixo
- Clientes enviam operações para o líder, este tenta um número de sequência k e corre a instância k do Paxos
- Eleição de um novo líder traz complicações…

Implementar replicação de máquina de estados (Alternativa #2 – Lab 5)

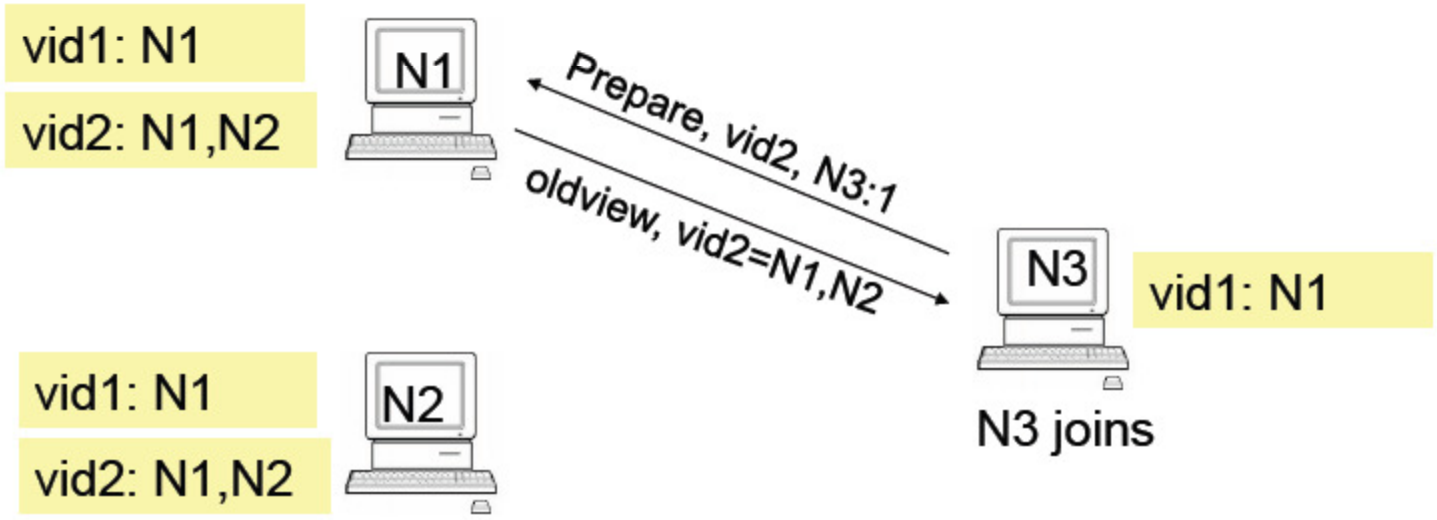
- Sistema replicado evolui numa sequência de vistas:
  - Conjunto actual de réplicas, uma delas é a réplica primária, restantes são backups
  - Identificador de vista (vid)
- Usar o Paxos para chegar a acordo sobre <primário,backups>
  - vid == número da instância do Paxos a correr

Implementar SMR com o Paxos (Lab 5)



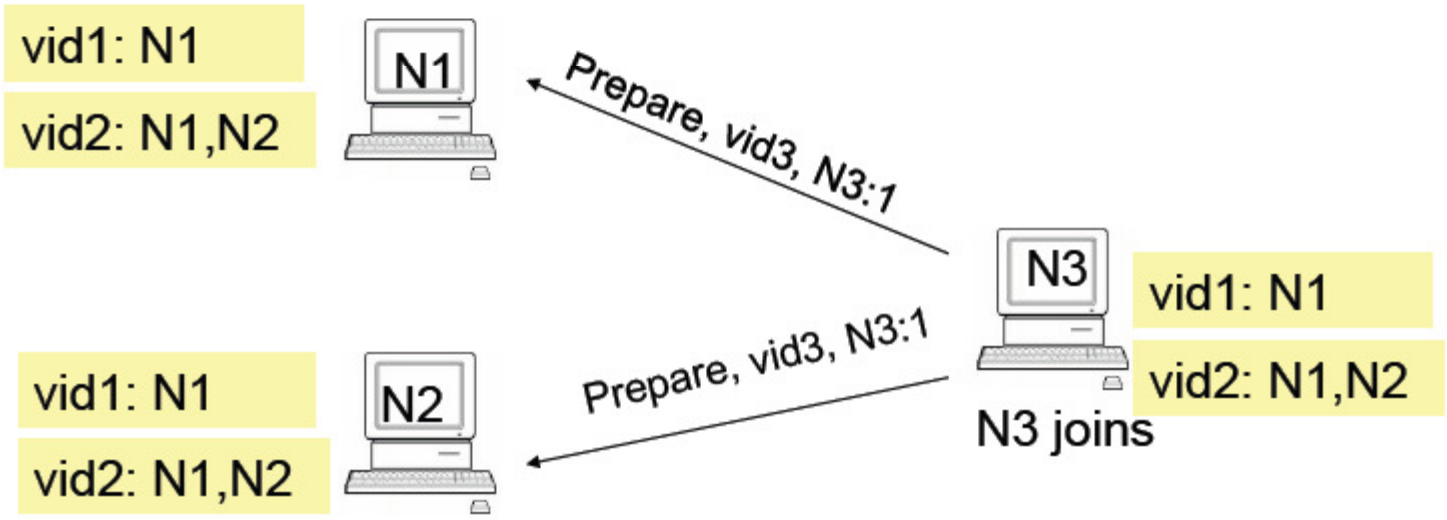
[source: J. Li and M. F. Kaashoek, MIT 6.824 lecture notes]

Implementar SMR com o Paxos (Lab 5)



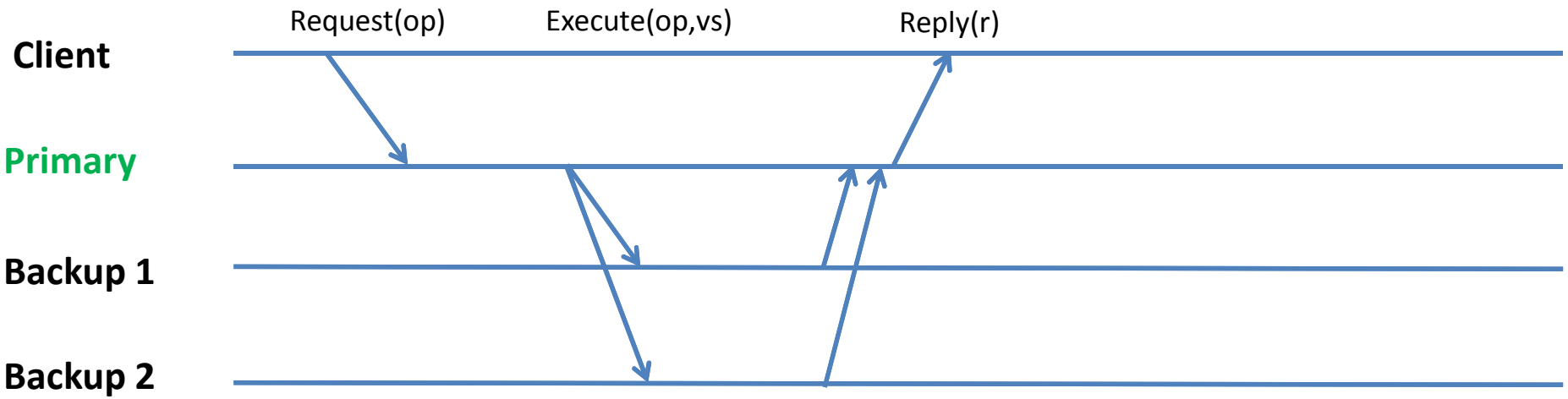
[source: J. Li and M. F. Kaashoek, MIT 6.824 lecture notes]

Implementar SMR com o Paxos (Lab 5)



[source: J. Li and M. F. Kaashoek, MIT 6.824 lecture notes]

Executar operações numa dada vista



Executar operações numa dada vista

- Primário atribui a cada pedido um "viewstamp" → tuplo (vid, num_sequência)
- Pedidos dos vários clientes são executados por ordem do viewstamp, por exemplo:
- (0,0) , (0,1) , (1,0) , (1,1) , (1,2) , (2,0) . . .

Desafio: garantir continuidade

- Qualquer réplica, para executar o pedido com viewstamp v, tem de executar toda a sequência de pedidos com viewstamps inferiores a v
  - Se necessário transfere estes pedidos de outras réplicas (transferência de estado)
- Um dos desafios do Lab 5 é garantir que isto acontece no após mudanças de vista

Exercício

- Desenhe o “timeline” de uma execução do algoritmo (original) do Paxos com três acceptors (A1-A3), dois proposers simultâneos (P1,P2), e em que um dos acceptors aceita duas propostas (ou seja, responde positivamente a dois pedidos de “accept”) com valores diferentes.