Sistemas Distribuídos Replicação Máquina de Estados & O Algoritmo Paxos

Prof. Elias P. Duarte Jr.
Universidade Federal do Paraná (UFPR)
Departamento de Informática
www.inf.ufpr.br/elias/sisdis

Sumário

- “Máquina de Estados”
- Replicação Máquina de Estados
- O Algoritmo de Consenso Paxos
  - vamos concentrar em 1 instância do Paxos

Máquina de Estados

- Uma representação (modelo) de um processo
- Consiste de:
  - Um conjunto de estados do processo
  - Um conjunto de transições entre estados
  - Uma transição do “estado atual” para o próximo estado ocorre como consequência de um “evento”
    - um evento pode ser, por exemplo, a execução de uma operação pelo processo (digo, máquina de estado)
    - transições de um estado para o próprio são permitidas

Máquina de Estado Determinística

- Uma máquina de estado determinística só permite a ocorrência de 1 evento de cada vez
- A transição de estados causada pelo evento também é única, produz uma saída bem definida
  - logicamente a saída produzida depende do estado atual e da transição que ocorreu

Em Sistemas Assíncronos com Falhas Crash

- A máquina de estados no modelo temporal assíncrono pode demorar um tempo arbitrário para realizar a transição entre estados
  - e para produzir a saída correspondente, se houver
- Considerando que processos (máquinas de estado) podem sofrer falhas crash
- O modelo assíncrono sofre daquele problema crucial: é impossível distinguir um processo falho de um processo lento

Replicação Máquina de Estados

- Aplicação típica: replicação de servidores
- Por exemplo: pense em um grupo de servidores Web replicados
- Na Replicação Máquina de Estados – *State Machine Replication:*

  → *cada réplica é uma máquina de estados*

  → *todas as réplicas iniciam no mesmo estado*

  → *executam a mesma sequência de operações*

  → *desta forma: todas as transições são idênticas!*

Réplicas Consistentes

- A Replicação Máquina de Estados garante que todas as réplicas estarão sempre no mesmo estado
- Desta forma: se uma réplica falha, sem problemas! O serviço continua disponível!
- Até N-1 réplicas podem falhar e o serviço continua disponível
- Além disso: aumenta o desempenho do serviço: ao invés de ter um servidor respondendo clientes, tem N servidores

Operações vão chegando

- Clientes geram requisições de operações que devem ser executadas pelo servidor
- As requisições devem carregar: o id do cliente (lógico), um *timestamp* (vamos explicar direitinho) e a operação a ser executada
- Considere que o servidor é réplicado usando RME
- Note que isso fica totalmente transparente para os clientes!
- Múltiplos clientes podem enviar múltiplas requisições

Múltiplos Clientes & Múltiplas Requisições

- Múltiplos clientes podem enviar múltiplas requisições
- Não podemos deixar que sejam executadas pelas réplicas do servidor em ordens diferentes!
- Deve haver um **consenso** sobre exatamente qual requisição deve ser executada de cada vez
- É justamente isso que o Paxos garante: todas as réplicas executam exatamente a mesma operação

Instâncias do Consenso

- Uma instância do consenso determina qual é a operação a ser executada
- Define portanto a execução de 1 única operação
- Para definir a operação seguinte, outra instância do consenso é executada
- Levando em conta todas as instâncias executadas:

  → o Paxos garante que todos os processos (réplicas, máquinas de estado) executam a mesma sequência de operações, exatamente na mesma ordem

O Algoritmo Paxos

- O Paxos é executado por um sistema distribuído S que consiste de N processos
- O sistema é assíncrono, sujeito a falhas crash
- Os processos recebem múltiplas requisições simultaneamente
- Devem estabelecer uma ordem global, na qual as requisições são atendidas pelos processos
- Cada requisição é identificada de forma única

Identificando Requisições

- Já sabemos como fazer para identificar mensagens de forma única em um sistema distribuído
- Basta a mensagem carregar (id-origem, timestamp)
- O timestamp é um contador de mensagens
- No Paxos precisamos de um timestamp levemente mais elaborado

Timestamps do Paxos

- No Paxos o timestamp das mensagens é usado para estabelecer uma precedência das requisições: devem ser comparáveis entre si
- Assim não podemos ter timestamps iguais e eles devem ser crescentes
- Como fazer?

Timestamps do Paxos

- Simples! Para garantir que os timestamps das mensagens de cada processo sejam únicos e crescentes basta fazer assim:
- O timestamp da 1ª mensagem do processo z é z
- O timestamp da 2ª mensagem do processo z é z+N
- O timestamp da 3ª mensagem do processo z é z+2N
- ....
- O timestamp da iª mensagem do processo z é z +iN
- Veja um exemplo:

Timestamps do Paxos

- Seja N=4
- Timestamps do processo 0: 0, 4, 8, 12, ...
- Timestamps do processo 1: 1, 5, 9, 13, ...
- Timestamps do processo 2: 2, 6, 10, 14, ...
- Timestamps do processo 3: 3, 7, 11, 15, ...

Paxos: Papéis dos Processos

- Os processos do Paxos têm 3 papéis distintos
- Um mesmo processo até pode assumir todos os papéis, mas para compreender o algoritmo é importante ver os papéis separados!

Paxos: Papéis dos Processos

- Os processos do Paxos têm 3 papéis distintos
- Um mesmo processo até pode assumir todos os papéis, mas para compreender o algoritmo é importante ver os papéis separados!
- Os papéis são:

  → *Proposers*: processos que propõem uma próxima requisição para ser atendida

  → *Acceptors*: são os processos que entram em acordo decidindo qual a próxima requisição a ser atendida

  → *Learners*: processos que aprendem uma decisão

Papéis dos Processos no Paxos

- Vamos usar o termo tradicional de consenso e, ao invés de executar o consenso sobre “requisições” vamos executar o consenso para “valores”
- Assim, de novo, um processo pode ser um dos:

  → *Proposers*: processos que propõem valores (por exemplo vindos de clientes do serviço)

  → *Acceptors*: decidem 1 valor

  → *Learners*: aprendem o valor decidido

Proposers Enviam Mensagens com Timestamps

- Um *proposer* propõe um valor
- Um *proposer* envia mensagens com timestamps únicos explicados anteriormente
- Os *acceptors* então executam o consenso: primeiro concordando em aceitar um valor vindo de *proposer*, e depois efetivamente aceitando o valor (2 fases!)
- Um *learner* recebe o valor decidido

O Algoritmo Paxos

Algoritmo Paxos

1ª Fase: → Proposer envia mensagem Prepare com timestamp para uma maioria dos acceptors;

→ Acceptor concorda em aceitar e responde OK se não tiver recebido outro Prepare com timestamp maior // i.e.: responde OK para o maior

→ Se o acceptor já tinha recebido Prepare com timestamp maior: envia o timestamp maior para o proposer; se já tiver aceitado (Fase 2) valor com menor timestamp, envia OK com valor aceito

2ª Fase: → Se o proposer recebeu OK de uma maioria dos processos: fechou! Envia uma mensagem Accept com o valor da requisição com o maior timestamp

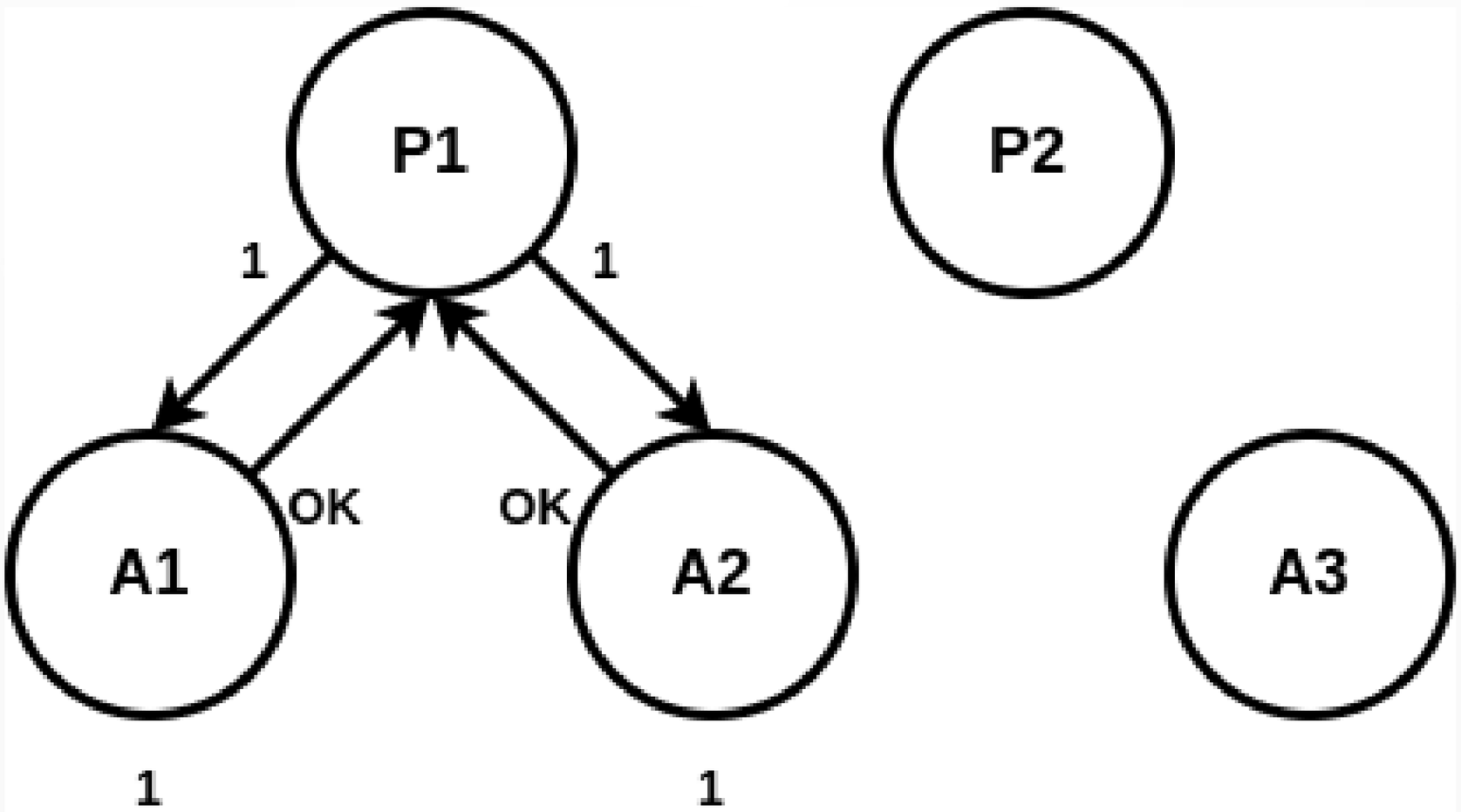
→ Se tiver recebido valor de algum processo: adota o valor!

→ O acceptor só aceita se não tiver recebido outro PrepareRequest com timestamp ainda maior

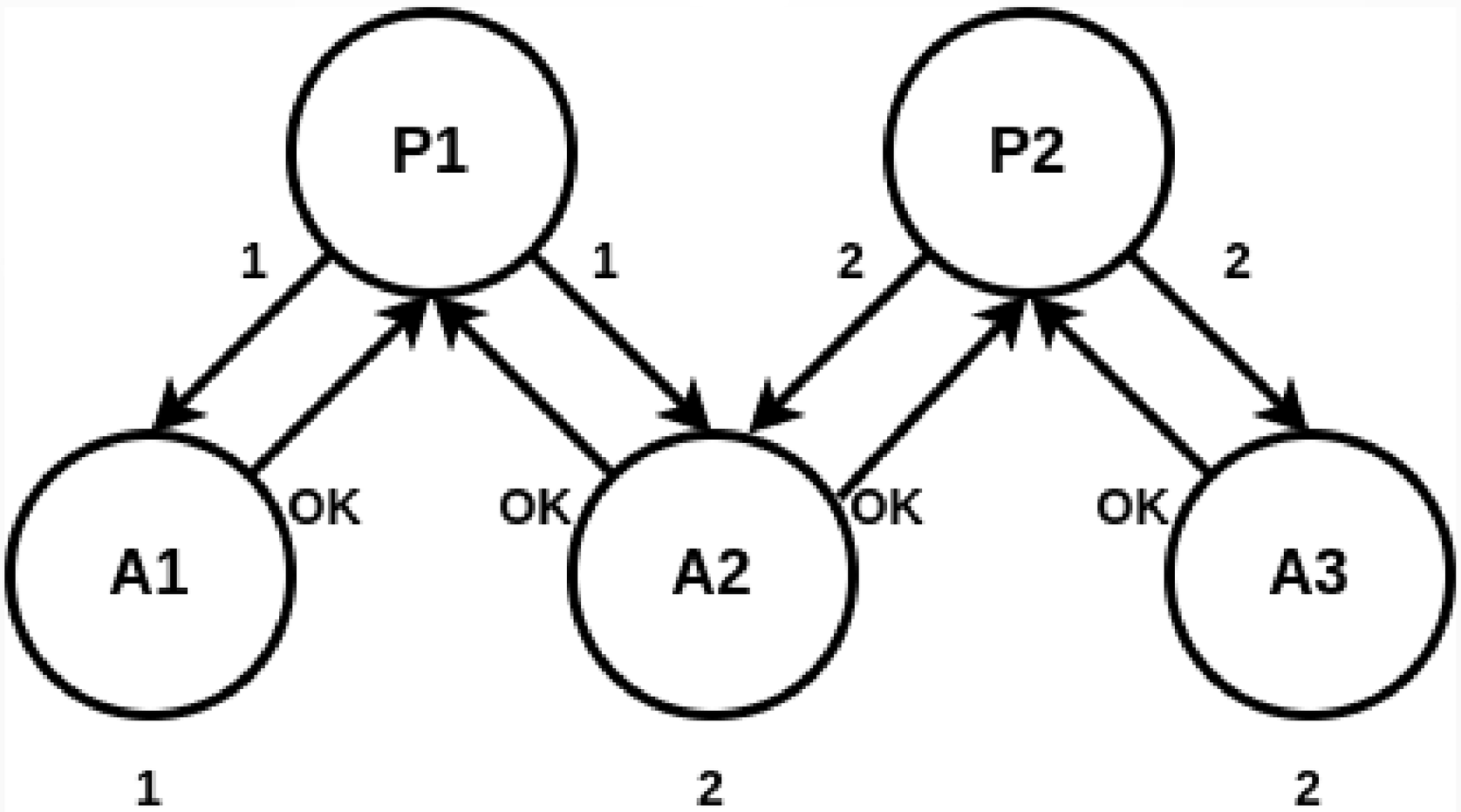
Paxos: Mais Proposer & Learner

- Atenção: se o proposer não recebe OK de uma maioria, então incrementa o timestamp e continua tentando
- No algoritmo original o learner tem que receber o valor decidido de uma maioria de acceptors
- Uma minoria pode ter decidido outro valor :-0

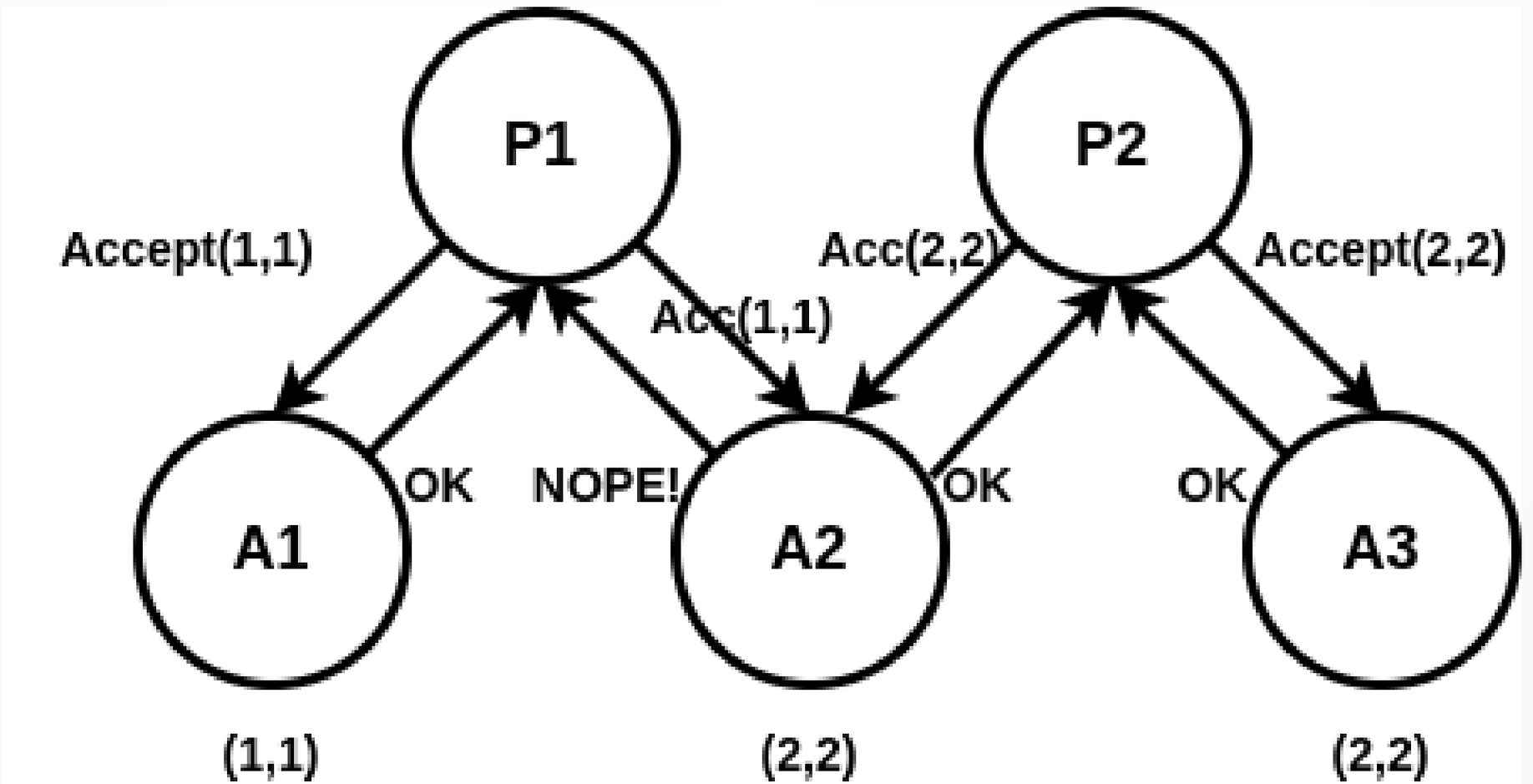
Exemplo Paxos: 3 acceptors, 2 proposers → Proposer1 propõe para maioria de acceptors



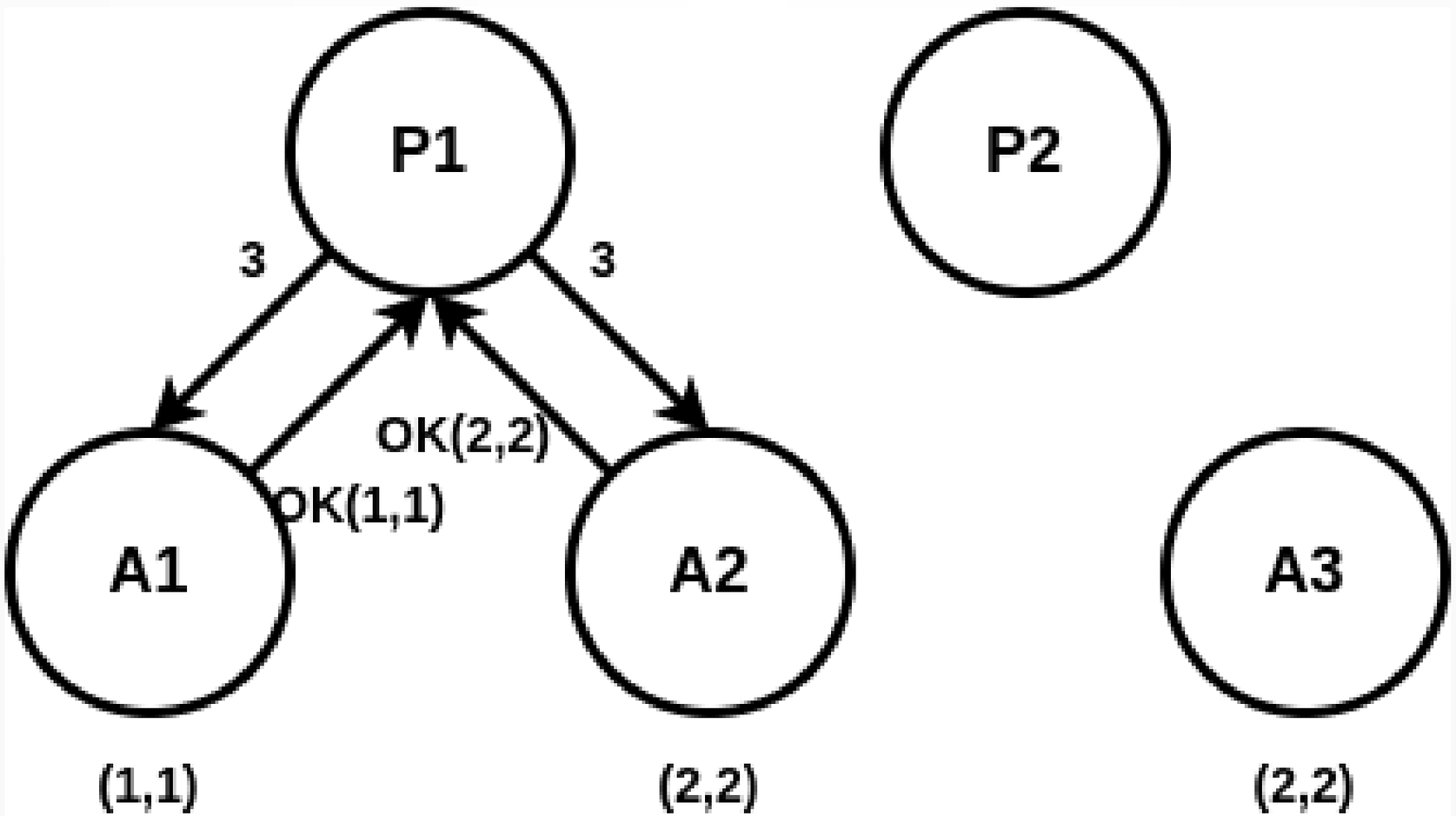
Agora o Proposer 2 Propõe



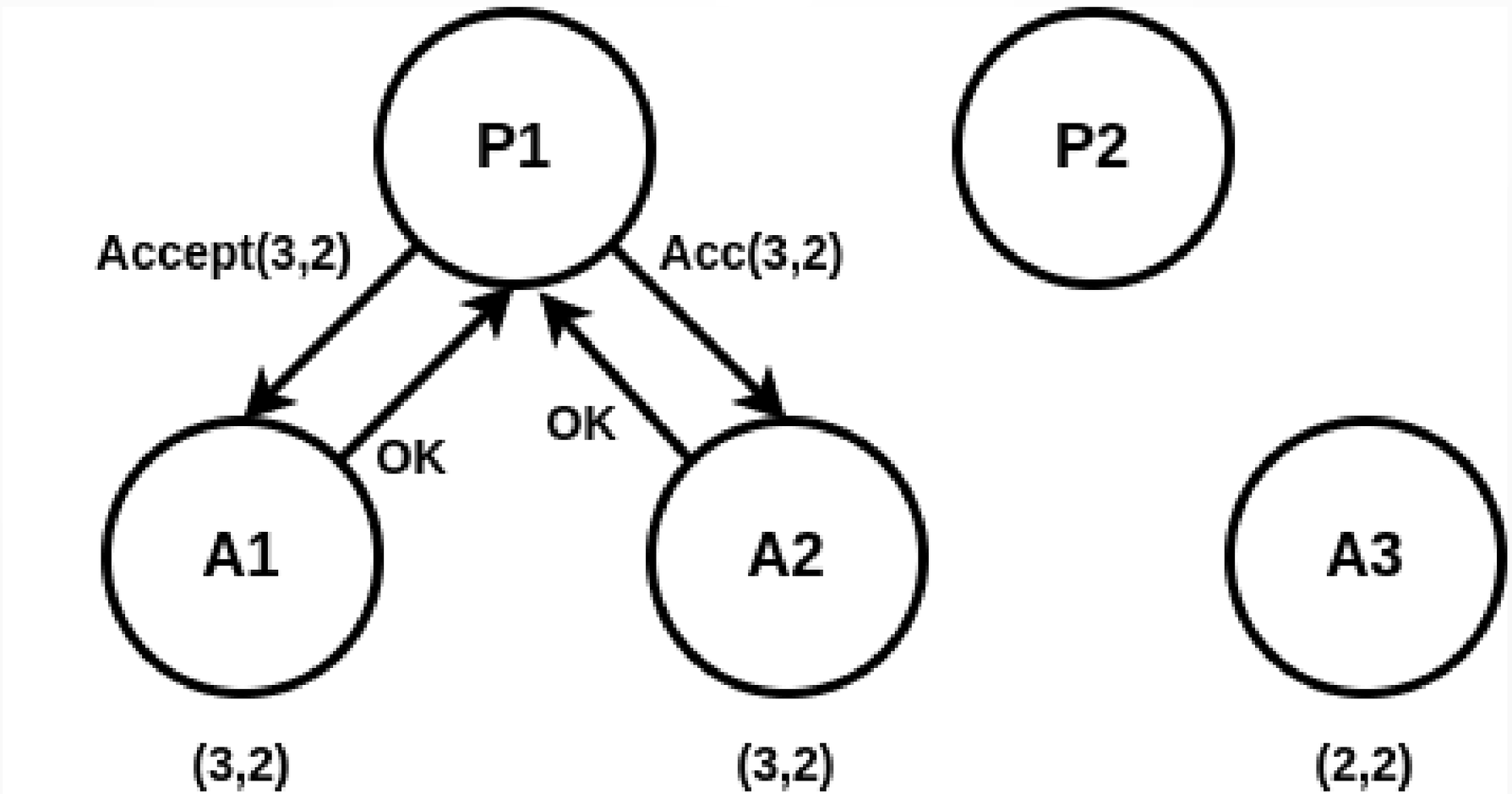
Na Fase 2: P2 tem sucesso



P1 Continua Tentando!
Mas A1, A2 já decidiram, veja o que ocorre



Todos os acceptors decidem!



Paxos: Coordenador

- As implementações do Paxos acabam usando um coordenador que é um processo central que recebe requisições de proposers e repassa aos acceptors
- Motivo: os proposers podem ficar competindo, aumentando os valores dos timestamps rapidamente de forma que nunca conseguem uma maioria

Propriedades do Paxos

- O Paxos garante a segurança (safety): jamais dois valores diferentes vão ser decididos
  - veja que a maioria tem um papel decisivo nesta propriedade
- O Paxos garante o progresso sob condições ditas de “sincronia fraca”
  - se há acceptors muuuuito lentos: nenhum proposer consegue maioria e ficam aguardando...
  - aqui entra um detector de falhas, mas a precisão é importante!

Referências:

A referência clássica de replicação máquina de estados:

F. B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," ACM Computing Surveys, Vol. 22, No. 4, 1990.

Uma referência introdutória para o Paxos:

L. Lamport, “Paxos Made Simple,” ACM SIGACT News (Distributed Computing Column), Vol. 32, No. 4, 2001.

Conclusão

- Definimos a Replicação Máquina de Estados
- Estudamos o Algoritmo Paxos (1 instância)

Obrigado!
Página da Disciplina
Sistemas Distribuídos:
www.inf.ufpr.br/elias/sisdis