# Tempos e Estados Globais

ECO036 - Sistemas Paralelos e Distribuídos

# Tópicos Abordados

- Tempo
- Relógios e Ordenação de eventos.
  - Relação Happened- Before
  - Relógios Lógicos
  - Vetor de Relógios
  - Relógios Físicos
  - Método de Cristian e Algoritmo de Berkeley
  - Network Time Protocol (NTP)
- Estados Globais
  - Problema reconhecidos
  - Estado global consistente
  - Algoritmo snapshot distribuído

# Porque tempo é interessante?

- Ordenação de eventos
  - Armazenamento de dados em memória, arquivos e banco de dados.
  - Pedidos de acesso exclusivo, como determinar quem pediu primeiro?
  - Trocas interativas

# Relógio e Ordenação de Eventos

# Relógios e Ordenação de Eventos

Para várias aplicações de Sistemas Distribuídos como Agendamento distribuído e exclusão mútua distribuída, é importante determinar a ordem no qual os vários eventos estão sendo executados. Se o sistema tem um relógio global compartilhado, apenas uma marcação de tempo para cada evento com relógio global seria suficiente para determinar a ordem.

# Eventos

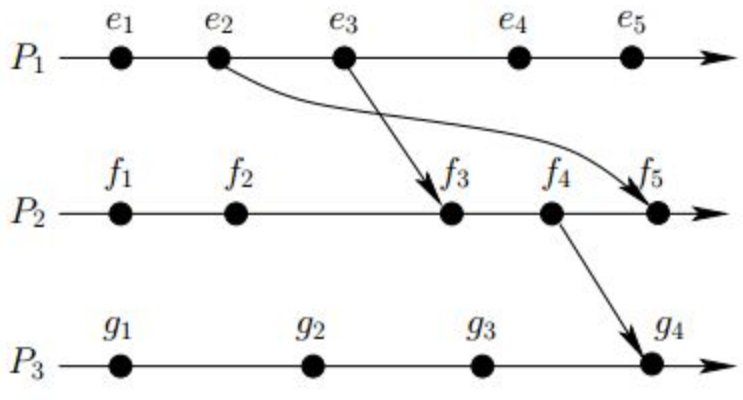
Como ordenar eventos na ausência de um relógio global?

# Relação Happened-before

- Se ***e*** ocorreu antes de ***f*** no mesmo processo, logo $e \rightarrow f$.
- Se ***e*** é o *send* de uma mensagem e ***f*** é o *receive* da mesma mensagem, logo $e \rightarrow f$.

# Exemplo

O conceito dessa relação foi proposto por Lamport. É importante destacar que esta relação consegue dar uma ordem parcial de um conjunto de eventos.
A ordem total pode ou não corresponder a ordem real de execução dos eventos. No entanto todos os processos devem concordar com a mesma ordem total.



# Relógios Lógicos

- Um relógio lógico “marca” cada evento com um valor inteiro de maneira que a resultante da ordem dos eventos seja consistente com a relação *happened-before*

# Relógios Lógicos

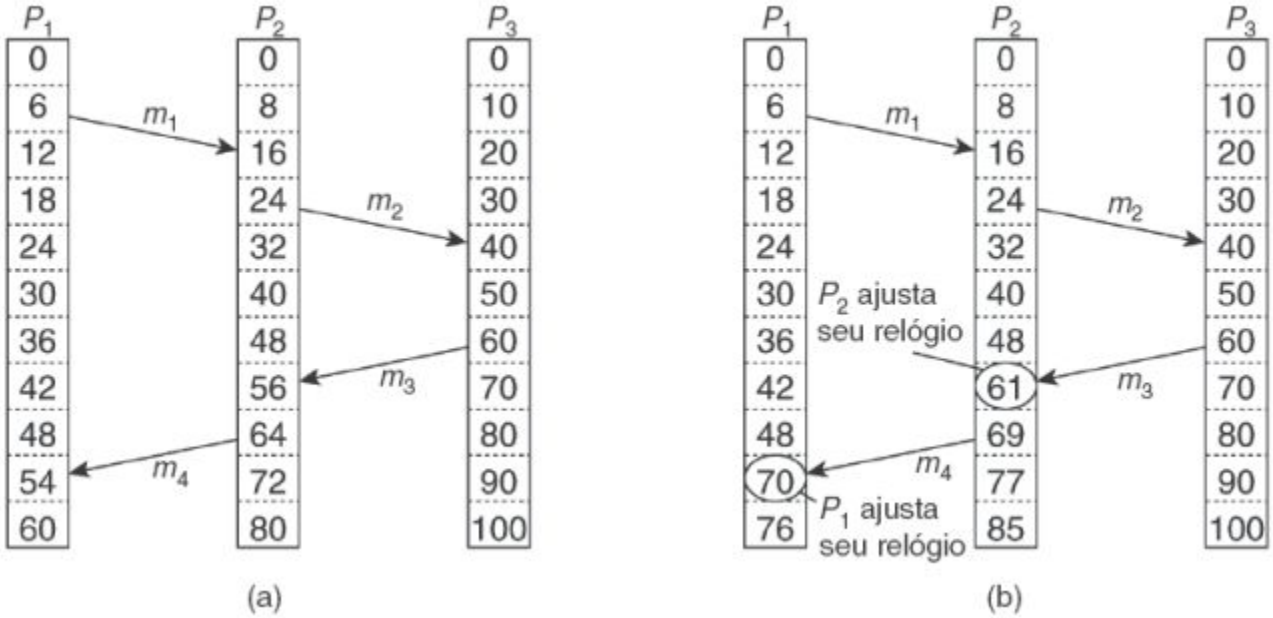
- Inventado por Lamport, o relógio lógico é um contador de software que aumenta a contagem monotonicamente, cujo o valor não precisa ter nenhum relacionamento em particular com nenhum relógio físico.

# Atualização de Relógios Lógicos

- RL1 Li é incrementado antes da ocorrência de um evento no processo pi Li = Li + 1.
- RL2 a) Quando um processo pi envia uma mensagem m, m leva “de carona” ( *piggybacking*) o valor t = Li.

b) Na recepção (m,t), um processo pj calcula Lj: max(Lj,t) e , então, aplica RL1 antes de indicar o tempo do evento no receive.

# Atualização de Relógios Lógicos

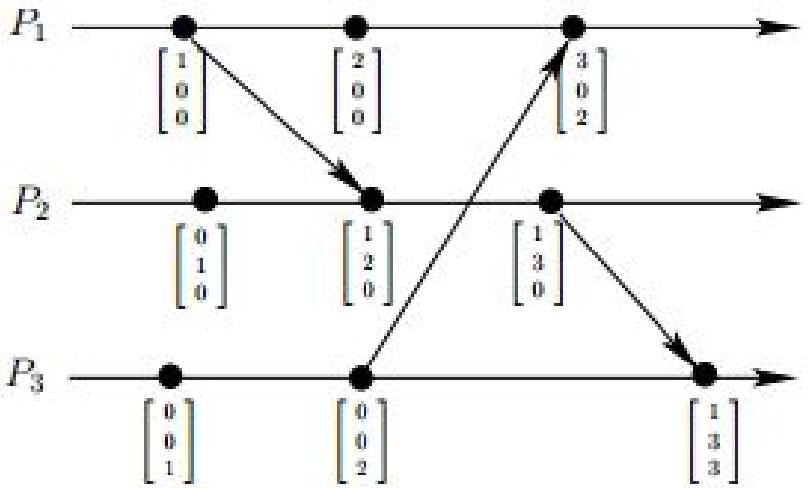


# Vetor de relógios

- Um relógio lógico estabelece uma ordem total de todos os eventos, mesmo quando dois eventos são incomparáveis utilizando a relação *happened-before*. Embora os relógios lógicos não conseguem tratar toda a informação sobre a relação *happened-before.* Vamos descrever um mecanismo chamado de vetor de relógios capaz de abranger toda a informação sobre a relação *happened-before.*

# Vetor de Relógios

Proposto por Mattern and Fidge, cada processo incrementa seu própio relógio vetorial, e após isso envia junto com a mensagem uma cópia de seu relógio vetorial. No recebimento da mensagem, atualiza os valores do relógio vetorial para o máximo de cada um.



# Relógios Físicos

- Embora os Relógios de Lamport dê uma ordem não ambígua aos eventos, o valor do tempo associado aos eventos não é necessariamente próximo do tempo real no qual eles ocorrem. Em alguns sistemas (por exemplo, sistemas de tempo-real), o tempo real é importante. Para amenizar os problemas de sincronização, foram propostos algoritmos como o Algoritmo de Cristian e o algoritmo de Berkeley.

# Curiosidades

**BIH** - *Bureau International de l'Heure* - Paris, França

**TAI** - International Atomic Time

**UTC** - Coordinated Universal Time

# Método de Cristian

- Usa método de sincronização externa
- Periodicamente, cada máquina cliente contata o servidor de tempo e requisita a hora.
- O servidor (WWW), informa a hora “oficial”.
- O cliente assume a hora informada pelo servidor.

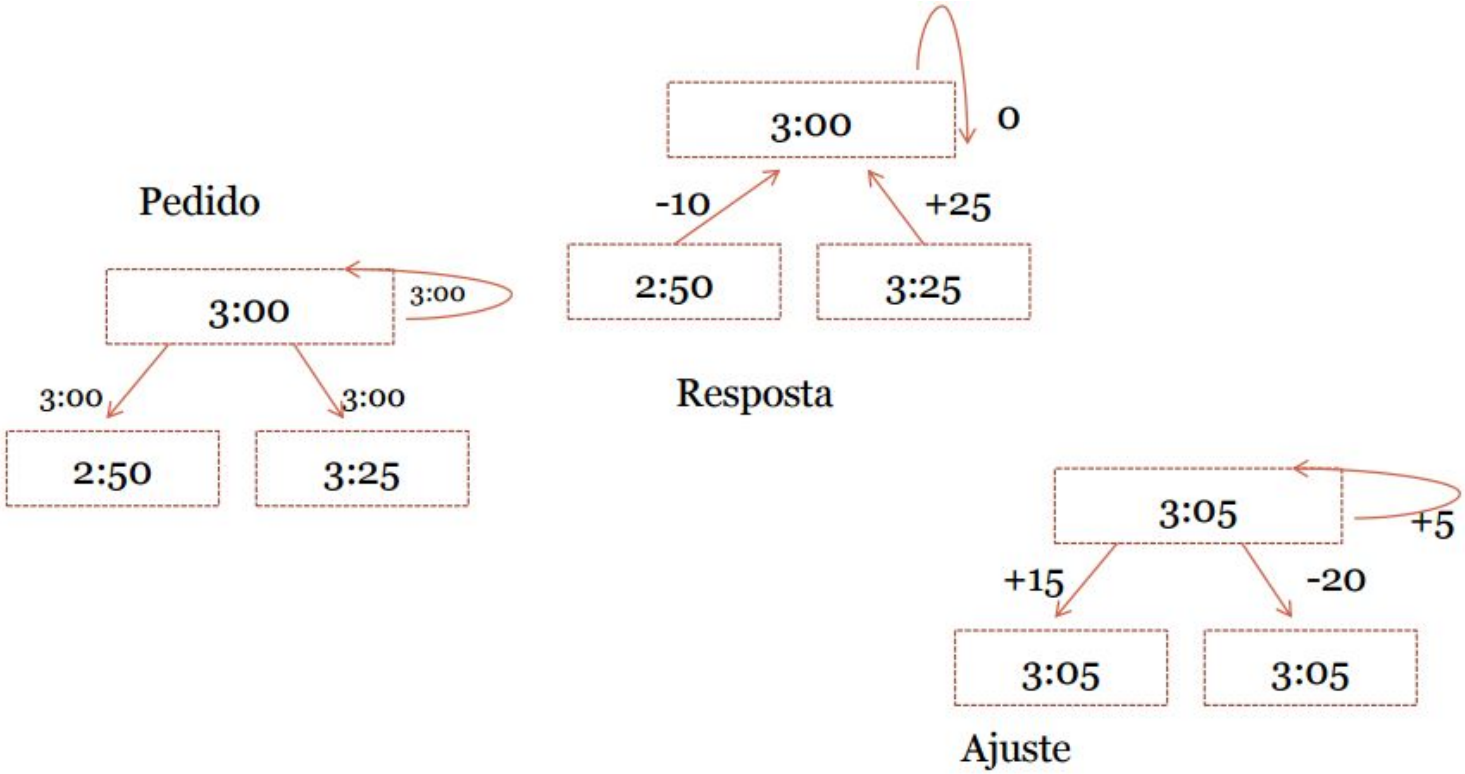
# Método de Cristian

- A hora do cliente não pode “voltar” atrás.
- A solução para este problema é fazer com que o relógio do cliente ande mais lento até sincronizar com o servidor.

# Método de Cristian

- O atraso de transmissão pode ser relevante
  - Contabilizar o RTT(*round-trip time*) e subtrair o tempo de viagem
  - Tentar descobrir tempo de computação do servidor
  - Tentar classificar atrasos curtos e longos.

# Algoritmo de Berkeley



# Algoritmo de Berkeley

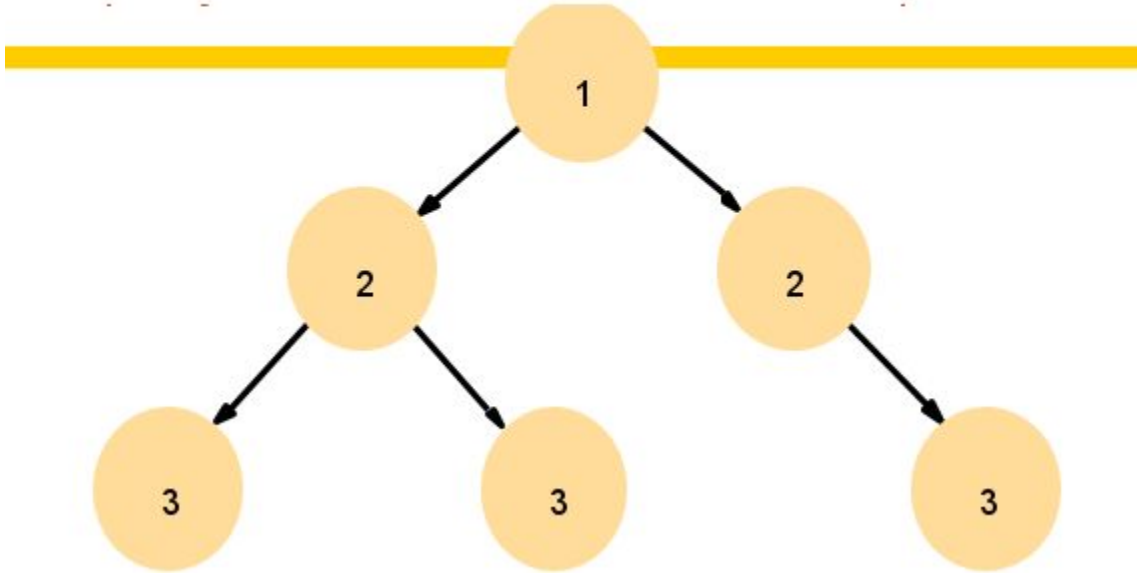
- Algoritmo de Berkeley( Sincronização interna)
  - Precisão: depende do RTT
  - Tolerância a falhas: Calcula a média dos tempos para um subconjunto de computadores que diferem a até um certo valor máximo. Ignora mensagens cujo tempo de transmissão é muito elevado ou muito baixo.
  - O que fazer se o *master* falhar?

    Eleger um novo *master*.

# Network Time Protocol (NTP)

O método de Cristian e o algoritmo de Berkeley se destinam principalmente ao uso dentro de *intranets. O NTP define uma arquitetura para um serviço de tempo e um protocolo para distribuir informações de tempo pela Internet.*

# Network Time Protocol (NTP)



# Network Time Protocol (NTP)

- Servidores NTP sincronizam em um dos três modos:
- *Multicast mode*: para conexões LAN. Um ou mais servers periodicamente lançam o tempo para o servidores via LAN, que fixam seus tempos assumindo um pequeno delay. Esse modo tem baixa precisão.
- *Procedure call*: Um server recebe uma requisição, respondendo com o carimbo de tempo, precisão maior que o anterior.

# Network Time Protocol (NTP)

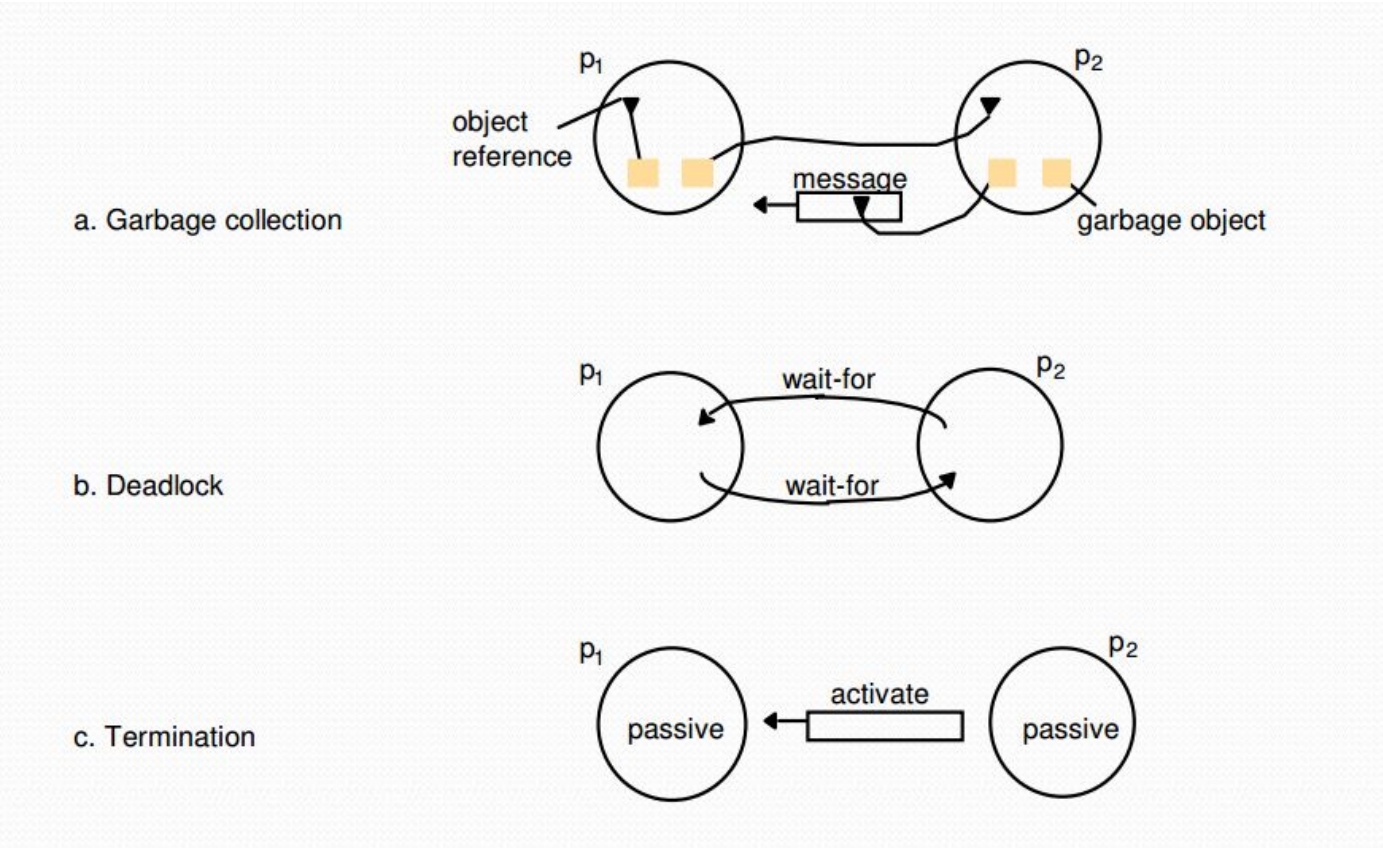
- *Symmetric mode*: usado por servidores que fornecem tempo em LAN, mais preciso dos modos e maior nível de sincronização de subredes. Um par de servidores simétricos trocam mensagens com a informação de tempo

# Estados Globais

- Estado global capta o conjunto de eventos que foram executados até o momento.
- Em sistemas distribuídos é complicado avaliar o estado global, é preciso cuidado ao se estabelecer o que ocorreu durante a execução.
- São utilizados para resolver muitos problemas em sistemas distribuídos.

# Problemas

- Coleta de lixo distribuída: um objeto é considerado lixo se não existem mais referências a ele em nenhuma parte do sistema distribuído.
- Detecção de deadlock distribuída: um deadlock distribuído ocorre quando cada processo de uma coleção de processos espera que outro envie uma mensagem para o outro.
- Detecção de término distribuída: detectar se um algoritmo distribuído terminou.




# Problemas a serem reconhecidos nos estados globais

# Estado Global Consistente

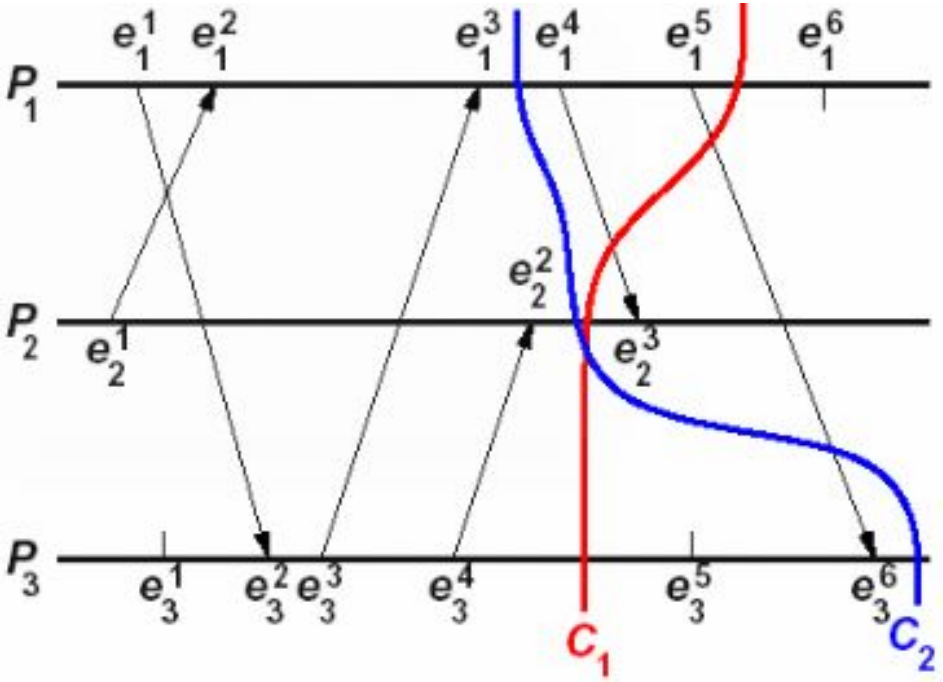
- Histórico local do processo i: $h_i = e_i^1 e_i^2 e_i^3 e_i^4 \ldots$

- Histórico global: $H = \bigcup_{i=1}^{N} h_i$ .

# Estado Global Consistente

- Para achar estados globais consistentes é utilizado uma técnica de corte.
- Este corte é uma divisão na execução do sistema, ou seja é um subconjunto do histórico global.
- Eventos à direita do corte estão fora do estado global e eventos à esquerda do corte estão dentro do estado global.
- Corte podem ser:
  - Consistentes: obedecem causalidade;
  - Insconsistentes: não obedecem causalidade.

# Estado Global Consistente



# Algoritmo snapshot distribuído

- Algorítmo de Chandy e Lamport (1985).
- Determina **estados globais consistentes** em sistemas distribuídos.
- Permite avaliar predicados estáveis.
  - Características instáveis em relação a um objeto: possui lixo, estar em deadlock ou estar terminado.
- Qualquer processo pode inciar o snapshot.
- Vários snapshots podem estar executando em paralelo.

# Condições do algoritmo

- n processos no sistema.
- Cada canal de comunicação é unidirecional com entrega FIFO.
  - FIFO: mensagem enviada primeiro é recebida primeiro.
- Não existe falha no envio de mensagens.
- Todas mensagens chegam intactas (sem perda de informação) e não são duplicadas.
- Processos continuam a executar normalmente durante o snapshot.

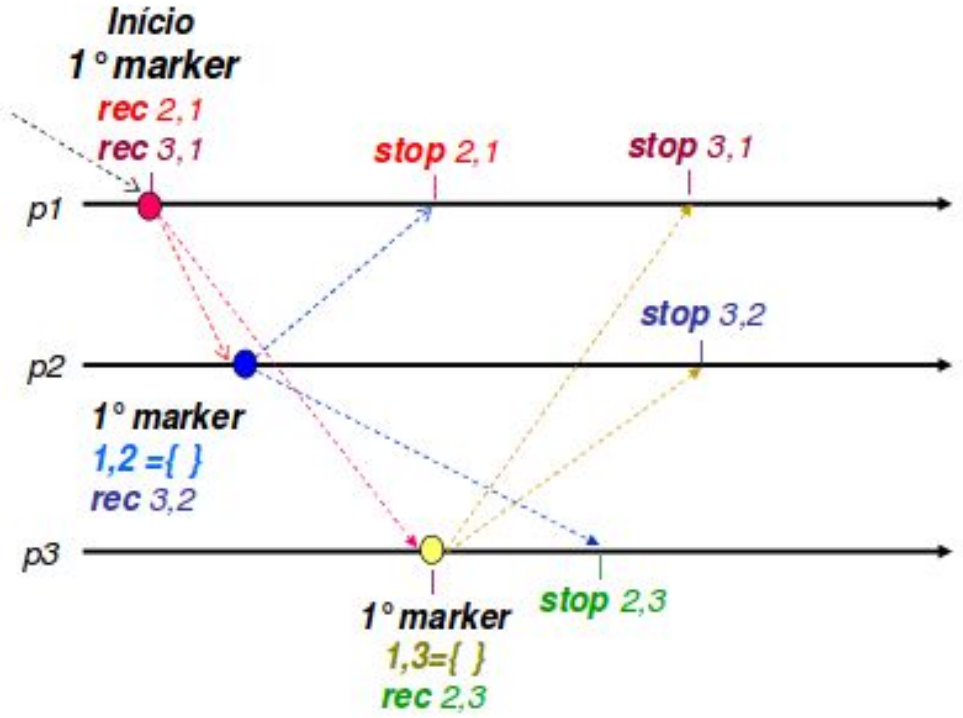
# Algorítmo snapshot distribuído

- Para o processo iniciador $P_i$, ou seja que inicia o snapshot.
  - Salva o seu estado
  - Então cria mensagens do tipo “Marker”
  - Para j = 1 até n, exceto i
    - $P_i$ envia mensagens “Marker” pelo canal $C_{ij}$
  - Grava as mensagens recebidas em cada um dos canais de $P_i$

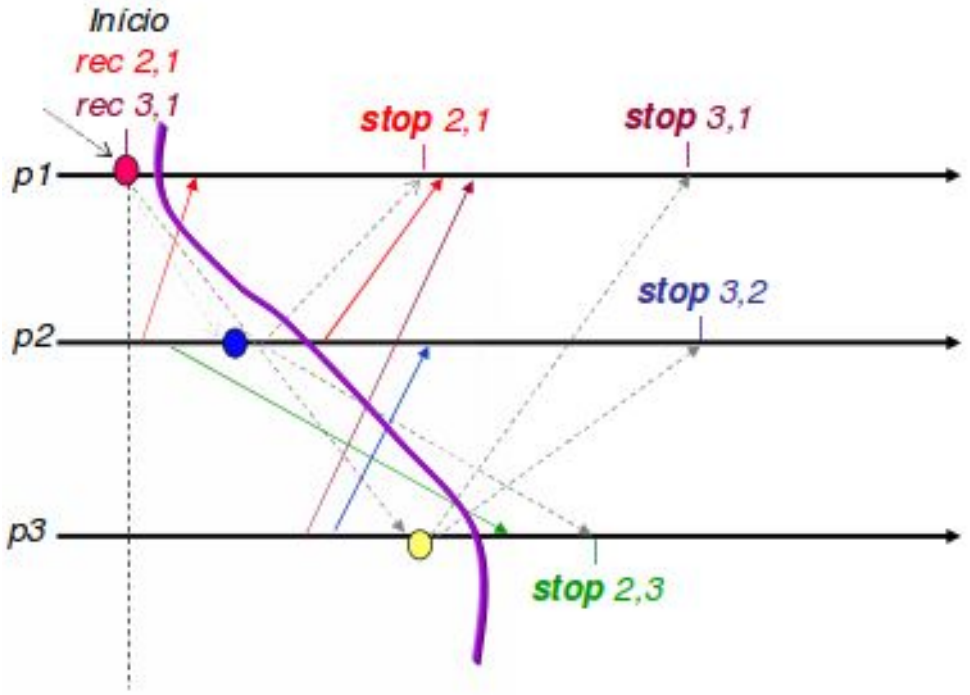
# Algorítmo snapshot distribuído

- Para o processo $P_i$ que recebeu um “Marker”, em um canal $C_{ki}$.
  - se (É o primeiro “Marker” que $P_i$ está recebendo)
    - Salva o seu estado
    - Define o estado do canal $C_{ki}$ como “vazio”
    - Para j = 1 até n, exceto i
      - $P_i$ envia mensagens “Marker” pelo canal $C_{ij}$
    - Grava as mensagens recebidas em cada um dos canais de $P_i$
  - se não
    - Define o estado do canal $C_{ki}$ como todas as mensagens que chegaram desde que a gravação foi ativada para o canal $C_{ki}$

# Exemplo de execução



# Exemplo de execução



# Conclusão

- Na ausência de Tempo Global em Sistemas Distribuídos, o tempo é uma variável importante para determinar a ordem de eventos em certas aplicações.
- Devido a necessidade da utilização de Sistemas Distribuídos os algoritmos aqui apresentados são de extrema importância para o futuro do desenvolvimento de aplicações.

# Referências


- http://users.ece.utexas.edu/~garg/dist/wiley-encyc.pdf (Acesso em Junho de 2016)
- http://www.tlc-networks.polito.it/oldsite/anapaula/Aula_Cap06b.pdf (Acesso em Junho de 2016)
- http://www.di.ubi.pt/~pprata/sdtf/SDTF_10_11_T02_TempoRelogios.pdf (Acesso em Junho de 2016)
- http://www.cin.ufpe.br/~avmm/arquivos/provas%20software/resuminho3.pdf (Acesso em Junho de 2016)
- http://www-usr.inf.ufsm.br/~ceretta/elc1018/sincronizacao.pdf (Acesso em Junho de 2016)
- http://www.dainf.cefetpr.br/~tacla/SDII/Cap10-06-EstadosGlobais.pdf (Acesso em Junho de 2016)