Capítulo 2: Modelo de Computação distribuída

Ajay Kshemkalyani e Mukesh Singhel

Distributed Computing: Principles, Algorithms, and Systems

Cambridge University Press

Programa Distribuído

Um programa distribuído é composto de n processos assíncronos, **p1, p2, p3, ..., pn**.

Os processos não compartilham memória global e comunicam-se unicamente por troca de mensagens.

Os processos não compartilham um relógio global.

Execução de processos e transferência de mensagens é feita de forma assíncrona.

Sem perda de generalidade, assumimos que cada processo executa em um processador distinto.

$\mathbf{C_{ij}}$ será o canal de comunicação entre os processos $\mathbf{p_i}$ e $\mathbf{p_j}$ e $\mathbf{m_{ij}}$ a mensagem trocada entre $\mathbf{p_i}$ e $\mathbf{p_j}$.

A demora na entrega de uma mensagem é finito, porem imprevisível.

Modelo de Execução Distribuída

A execução de um processo consiste na execução seqüencial de ações de um evento.

Os eventos são atômicas e são classificadas em três tipos: evento interna, evento de recebimento de mensagem e evento de envio de mensagem.

Seja $\mathbf{e}_{xj}$ o evento de número **x** no processador $\mathbf{p}_j$.

send(m) e **rec(m)** são as ações de enviar e receber a mensagem **m** respectivamente.

A ocorrência de eventos modifica o estado dos processos e dos canais entre processos.

Eventos internos modificam o estado do processo no qual ocorrem.

Um evento de envio de mensagem modifica o estado do processo que a enviou e o canal no qual a mensagem foi enviada.

Um evento de recebimento de mensagem modifica o estado do processo que a recebeu e o canal no qual a mensagem foi recebida.

Modelo de Execução Distribuída

Os eventos em um processo são ordenados pela ordem de ocorrência.

A execução de um processo $\mathbf{p_i}$ produz uma seqüência de eventos $\mathbf{e_{1i}}$, $\mathbf{e_{2i}}$, $\mathbf{e_3}$, ..., $\mathbf{e_{(x+1)i}}$ e e denotado por $\mathbf{H_i}$ ond:

$\mathbf{H_i = (hi \rightarrow i)}$

$\mathbf{H_i}$ é o conjunto de eventos produzidos por $\mathbf{p_i}$ e a relação binária $\rightarrow$ i define a ordem destes eventos.

A relação binária $\rightarrow$ i define uma dependência causal entre os eventos de $\mathbf{p_i}$.

Modelo de Execução Distribuída

Os eventos de receber e enviar mensagens definem o fluxo de informação e estabelecem uma relação de causalidade entre remetente e destinatário de mensagem.

A relação $\rightarrow$ **msg** captura uma relação causal de dependência devido a envio de mensagem e definida como: Para cada mensagem **m** que é trocada entre dois processos temos:

send(m) $\rightarrow$ msg Rec(m)

A relação $\rightarrow$ **msg** define uma dependência causal entre o par de processos que correspondem ao evento de envio e recebimento da mensagem.

Modelo de Execução Distribuída

A evolução da execução de uma aplicação distribuído pode ser representado como um diagrama espaço-temporal.

A linha horizontal representa o progresso de um processo. Um ponto indica um evento e uma seta indica transferência de uma mensagem.

A linha horizontal não tem relação direta com o tempo físico. É a representação de ordenação lógica dos processos.

Como cada evento é tido como atômico (logo indivisível), justifica-se a utilização da representação pontual na linha de progresso.

.Na figura 2.1 para o processo **p1**, o segundo evento é um evento de envio de mensagem, o terceiro é um evento interno e o quarto um evento de recebimento de mensagem.

Modelo de Execução Distribuída

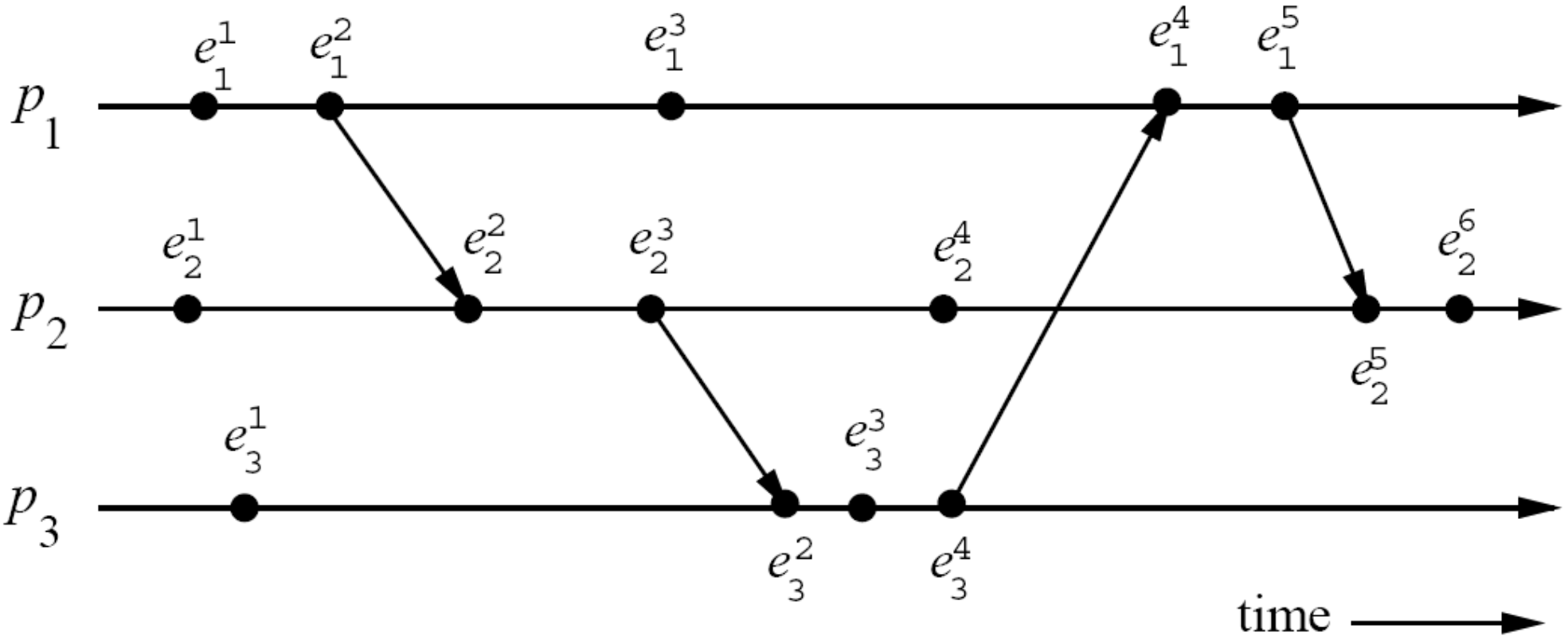

Figura 2.1: Diagrama espaço-temporal de execução distribuída.

Modelo de Execução Distribuída

Relação de causalidade

A execução de uma aplicação distribuída resulta em um conjunto de eventos produzidos pelos eventos.

Como anteriormente definido **H** é o conjunto de todos os eventos.

A relação binária → que expressa causalidade no conjunto H será definida como:

$$\forall e_i^x, \forall e_j^y \in H, e_i^x \rightarrow e_j^y \Leftrightarrow \begin{cases} e_i^x \rightarrow e_j^y \text{ se, } (i = j) \wedge (x < y) \\ \text{Ou} \\ e_i^x \rightarrow \text{msg } e_j^y \\ \text{Ou} \\ \exists e_k^z \in H: e_i^x \rightarrow e_k^z \wedge e_k^z \rightarrow e_j^y \end{cases}$$

A relação de causalidade induz uma ordenação parcial entre os eventos denominada ´**H = (H, →)**

Modelo de Execução Distribuída

Relação de causalidade

A relação → é a relação "Acontece antes" *(happens before)* de Lamport.

Para dois eventos $\mathbf{e_i}$ e $\mathbf{e_j}$, se $\mathbf{e_i} \rightarrow \mathbf{e_j}$, então a ocorrência de $\mathbf{e_j}$ é diretamente dependente da ocorrência do evento $\mathbf{e_i}$. Graficamente significa que existe um caminho de de linhas de progresso de processo e envio de mensagens no diagrama espaço-temporal conectando estes eventos.

Por exemplo na figura 2.1, $e_1^1 \rightarrow e_3^3$ e $e_3^3 \rightarrow e_2^6$.

A relação → denota o fluxo de informação no sistema distribuído e $\mathbf{e_i} \rightarrow \mathbf{e_j}$ indica que potencialmente toda a informação contida em e_i está disponível em $\mathbf{e_j}$.

Por exemplo na figura 2.1 o evento tem conhecimento de todos os outros eventos da figura.

e_2^6

Modelo de Execução Distribuída

Relação de causalidade

Para dois eventos $\mathbf{e_i}$ e $\mathbf{e_j}$, $\mathbf{e_i} \not\rightarrow \mathbf{e_j}$, denota o fato que o evento $\mathbf{e_j}$, não depende do evento $\mathbf{e_i}$. Ou seja o evento $\mathbf{e_i}$ não afeta de forma causal o evento $\mathbf{e_j}$.

Nessa caso o evento $\mathbf{e_j}$ não está ciente do momento de execução do evento $\mathbf{e_i}$. $\mathbf{e_i}$ pode ocorrer antes ou depois de $\mathbf{e_j}$.

Por exemplo, na figura 2.1

Note as duas regras abaixo: $e_1^3 \not\rightarrow e_3^3$ and $e_2^4 \not\rightarrow e_3^1$.

Para dois eventos e_i e e_j,

Para dois eventos e_i e e_j, $e_i \not\rightarrow e_j \not\Rightarrow e_j \not\rightarrow e_i$.

$e_i \rightarrow e_j \Rightarrow e_j \not\rightarrow e_i$.

Modelo de Execução Distribuída

Eventos concorrentes

Para quaisquer dois eventos $\mathbf{e_i}$ e $\mathbf{e_j}$, se $e_i \not\rightarrow e_j$ and $e_j \not\rightarrow e_i$, então os eventos $\mathbf{e_i}$ e $\mathbf{e_j}$ são ditos concorrentes (denotado por $\mathbf{e_i \parallel e_j}$).

Na figura 2.1 $e_1^3 \parallel e_3^3$ and $e_2^4 \parallel e_3^1$.

A relação || não é transitiva, isto é: $(e_i \parallel e_j) \wedge (e_j \parallel e_k) \not\Rightarrow e_i \parallel e_k$.

Por exemplo na figura 2.1 $e_3^3 \parallel e_2^4$ and $e_2^4 \parallel e_1^5$. Entretanto, $e_3^3 \not\parallel e_1^5$.

Para quaisquer dois eventos $\mathbf{e_i}$ e $\mathbf{e_j}$ em uma execução distribuída,

$$e_i \rightarrow e_j \text{ or } e_j \rightarrow e_i, \text{ or } e_i \parallel e_j.$$

Modelo de Execução Distribuída

Concorrência Lógica vs. Física

Em um sistema distribuído dois eventos são logicamente concorrentes se, e apenas se, eles não exercem relação de causalidade um no outro.

Concorrência física por outro lado, indica que os processos executam no mesmo instante físico.

Dois processos podem ser logicamente concorrentes e nunca ocorrer em um mesmo tempo físico.

Entretanto se a velocidade de processamento e atraso na entrega de mensagens esse dois processos poderiam muito bem ocorrer no mesmo tempo físico.

Se um conjunto de eventos concorrente coincide ou não em um tempo físico a saída da aplicação distribuída não deve se alterar.

Sendo assim, sem perda de generalidade, podemos assumir que dois processos concorrentes ocorrem em um mesmo instante físico.

Modelos de Redes de Comunicação

Há diferentes tipos de serviços oferecidos pelas redes de comunicação, podemos citar: **FIFO**, **Não-FIFO** e **Ordenação Causal.**

FIFO (First In First Out) -Cada canal funciona como uma fila “primeiro a chegar primeiro a sair”. Desta forma a ordem de chega de mensagens aos processos é preservada.

Não-FIFO – Cada remetente pode receber as mensagem de forma randômica Não há nenhum tipo de garantia quanto a ordenação de chegada das mensagens. Apesar de todas as mensagens chegarem em um tempo finito.

Modelos de Redes de Comunicação

O modelo “Ordenação Causal” é baseado na relação de Lamport “happens before”..

O sistema que suporta ordenação causal satisfaz a seguinte propriedade:

OC: Para cada duas mensagens $\mathbf{M_{ij}}$ e $\mathbf{M_{kj}}$ se $\mathbf{send(M_{ij}) \rightarrow send(M_{kj})}$ então $\mathbf{rec(M_{ij}) \rightarrow rec(M_{kj})}$.

Essa propriedade garante que a ordem de chegada e envio entre mensagens respeita a causalidade dos eventos.

Ordenação causal implica em canais **FIFO**. Pois $\mathbf{OC \subset FIFO \subset Não\text{-}FIFO}$

Ordenação causal simplifica o design de algoritmos distribuídos, pois prove sincronização pela rede

Estado Global de um sistema distribuído

“Um conjunto do estado de todos os processos e todos os canais”

O estado de um processo é definido pelo conteúdo dos registradores do processo: Pilha memória local, etc... E depende do contexto da aplicação distribuída.

O estado do canal é dado pelo conjunto de mensagens em transito neste canal.

A ocorrência de eventos modifica o estado do processo no qual este ocorre e de seus canais de comunicação.

Um evento interno modifica o estado do processo no qual este ocorre.

Eventos de envio de mensagens modifica o estado do processo que a enviou e o estado do canal no qual foi enviada a mensagem.

Eventos de recebimento de mensagens modifica o estado do processo que a recebeu e o estado do canal no qual a mensagem foi recebida.

Estado Global

Notação

Ls_{ix} denota o estado do processo **P_i** depois da ocorrência do evento **e_{ix}** e antes do evento **$e_{i(x+1)}$**.

Lsi0 denota o estado inicial do processo **p_i**.

Ls_{ix} é o resultado da execução de todos os eventos do processo **p_i** até **e_{ix}**.

Sendo **send(m)≤ Ls_{ix}** denota o fato de **∃y:1≤y≤x:: e_{iy} = send(m)**.

Sendo **rec(m)≤ Ls_{ix}** denota o fato de **∃y:1≤y≤x:: e_{iy} ≠ rec(m)**.

Estado Global

Estado do Canal

O estado do canal depende do estado dos processos que ele conecta.

Seja $SC_{ij}^{x,y}$ O estado do canal C_{ij} O estado do canal é definido como:

$$SC_{ij}^{x,y} = \{ m_{ij} \mid send(m_{ij}) \in e_i^x \wedge rec(m_{ij}) \notin e_j^y \}$$

Logo o estado do canal $SC_{ij}^{x,y}$ denota todas as mensagens que o processo pi enviou no evento $\mathbf{e_{ix}}$ que o processo p_j não recebeu até o processo $\mathbf{e_{jy}}$.

Estado Global

Estado Global

O estado global é o conjunto dos estados locais dos processos e estados dos canais.

Logo o estado global **GS** é definido como:

$$GS = \left\{ \bigcup_{i} LS_i^x, \bigcup_{j,k} SC_{jk}^{yj,zk} \right\}$$

Para um estado global ter sentido, o estado de cada um dos processos e canais deve ser gravado em um mesmo instante lógico.

Isso é facilmente obtido se todos os relógios físicos estiverem perfeitamente sincronizados, ou se houver um relógio físico global. (Na prática ambos são impossíveis)

Estado Global

Estado Global consistente

Mesmo que o estado de todos os componentes não seja gravado no mesmo instante o estado global pode ser consistente se todas as mensagens gravadas como enviadas forem gravadas como recebidas.

A idéia básica é que o estado global não deve violar a causalidade dos eventos – Um efeito não deve estar presente sem sua causa. Uma mensagem não pode recebida se não tiver sido enviada e vice e versa.

Este é um estado global consistente.

Estados globais inconsistentes não tem sentido para o sistema, visto que nunca ocorrem em um sistema.

O estado Global $GS = \{ \bigcup_i LS_i^x, \bigcup_{j,k} SC_{jk}^{yj,zk} \}$ é consistente se

$$\forall m_{ij} : \text{send}(m_{ij}) \in LS_i^x \Leftrightarrow m_{ij} \in SC_{jk}^{yj,zk} \wedge \text{rec}(m_{ij}) \in LS_i^x$$

Isto é: o estado $SC_{jk}^{yj,zk}$ e estado do processo LS_i^x não podem incluir nenhuma mensagem que o processo pi enviou após a execução do evento $\mathbf{e_{ix}}$.

Estado Global

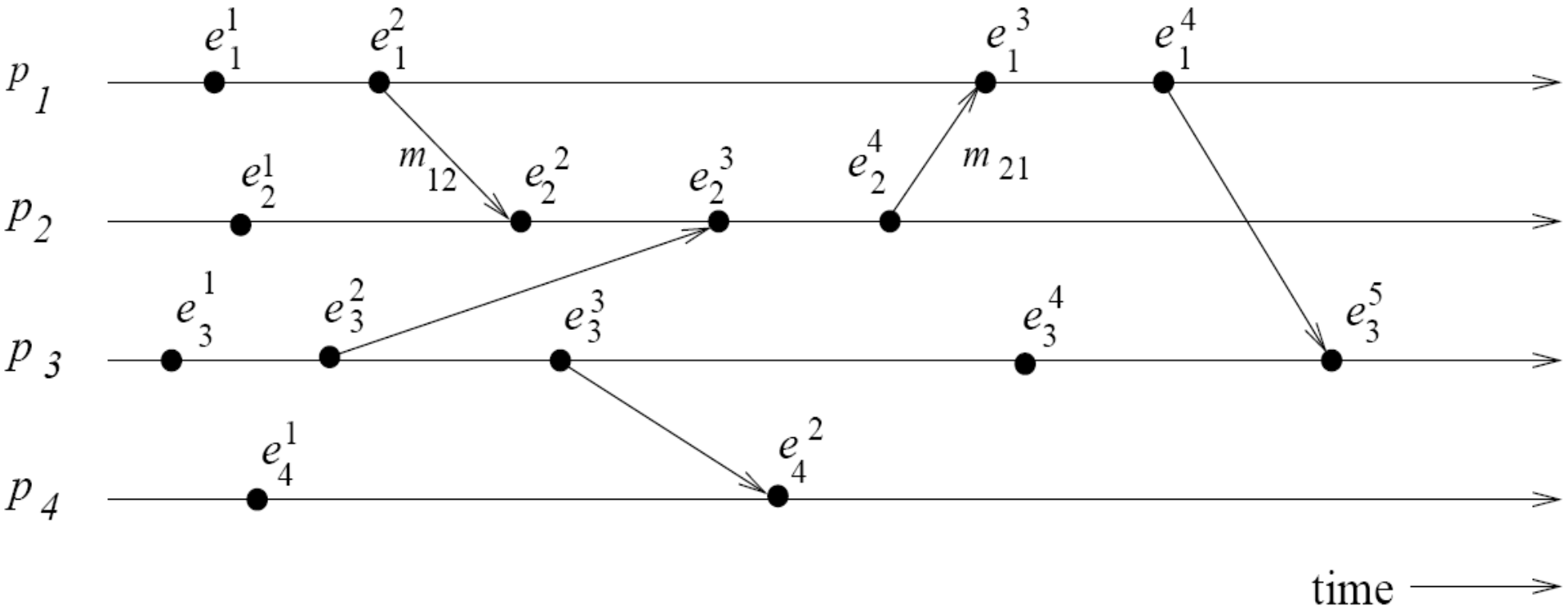

Figura 2.2: Diagrama espaço-temporal de execução distribuída.

Estado Global

Na figura 2.2

O estado Global $GS = \left\{LS_1^1, LS_2^3, LS_3^3, LS_4^2\right\}$ é inconsistente porque o estado de p_2 gravou o recebimento de mensagem $\mathbf{m}_{1,2}$, entretanto o estado de p_1 não gravou o envio.

O estado Global $GS = \left\{LS_1^2, LS_2^4, LS_3^4, LS_4^2\right\}$ é consistente, todos os canais estão vazios com exceção do canal $\mathbf{C}_{1,2}$ que contém a mensagem $\mathbf{m}_{2,1}$

Cortes

“No diagrama de espaço-tempo de uma aplicação distribuida, um corte é uma linha zigzag unindo pontos arbitrarios em cada linha de processo”

Um corte divide o diagrama espaço-temporal e criando dois conjuntos de eventos **Passado** e **Futuro.**

Passado contém todos os eventos a esquerda do corte e **Futuro** todos os eventos a direita.

Para um corte C, seja **Passado(C)** e **Futuro(C)** os eventos em **Passado** e **Futuro** de C respectivamente.

Todo Corte representa um estado global e todo o estado global pode ser representado como um corte no diagrama.

Cortes são uma ferramente gráfica poderosa na representação e entendimento do estado global da aplicação.

Estado Global e Cortes

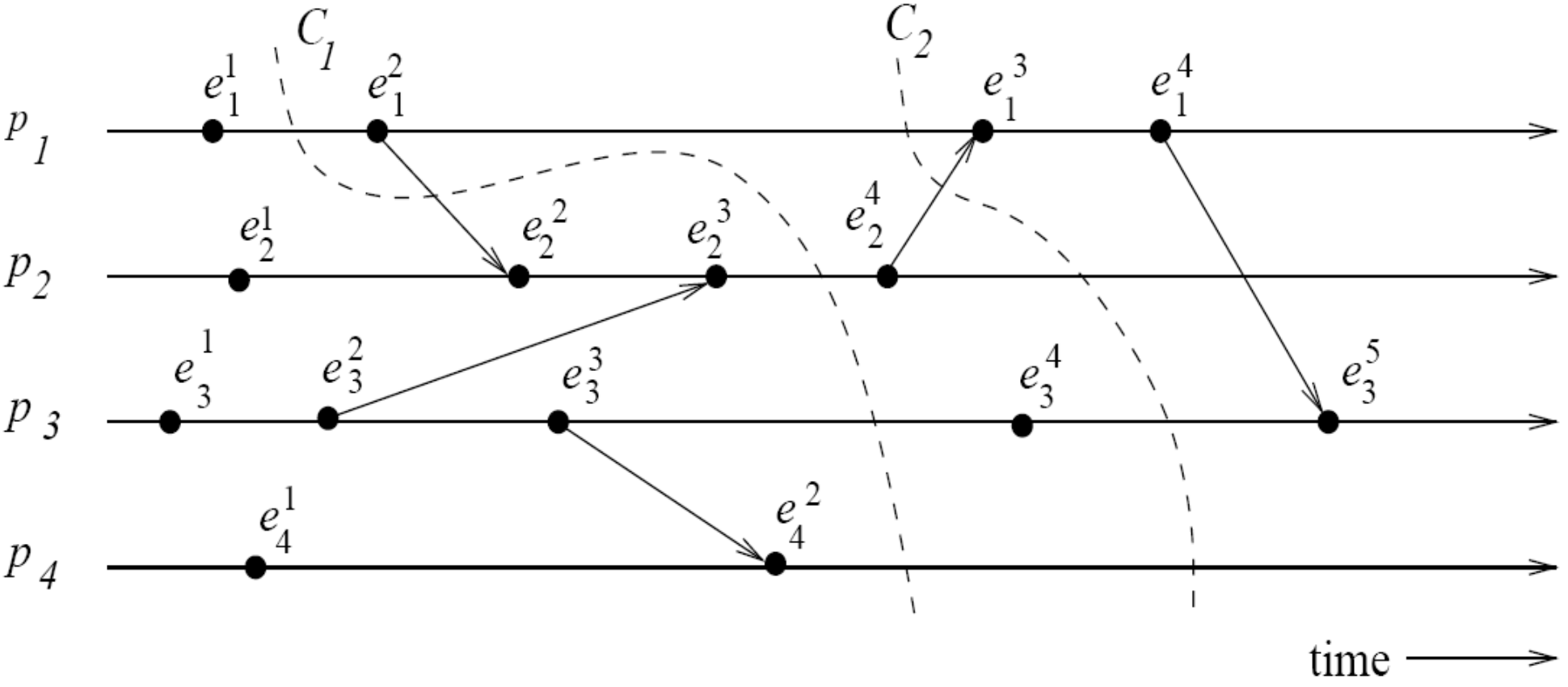

Figura 2.3 Ilustração do Corte em uma execução distribuída

Cortes

Em um corte consistente cada mensagem no **Passado** do corte foi enviada no **Passado** (na figura 2.3 o Corte C2 é consistente).

Em um corte **inconsistente** mensagens são enviadas do **Futuro** e recebidas no **Passado** (na figura 2.3 o Corte C1 é inconsistente).

Todas as mensagens que cruzam a fronteira **Passado Futuro** estão em transito no estado global definido por este corte.

Cones de tempo

Cone de tempo passado de um evento

Um evento $\mathbf{e_j}$ pode ser afetado apenas por um evento $\mathbf{e_i}$ tal que $\mathbf{e_i} \rightarrow \mathbf{e_j}$.

Nessa situação toda a informação de e_i pode estar disponível a $\mathbf{e_j}$.

Todos os eventos ei que satisfazem essa condição pertencem ao passado de $\mathbf{e_j}$.

Seja o **Passado($\mathbf{e_j}$)** todos os eventos de e_j na computação **(H,→)** então,

Passado(ej) = {ei |∀ei ∈ H, ei → ej }.

A figura 2.4 mostra o passado de $\mathbf{e_j}$.

Cones de Tempo

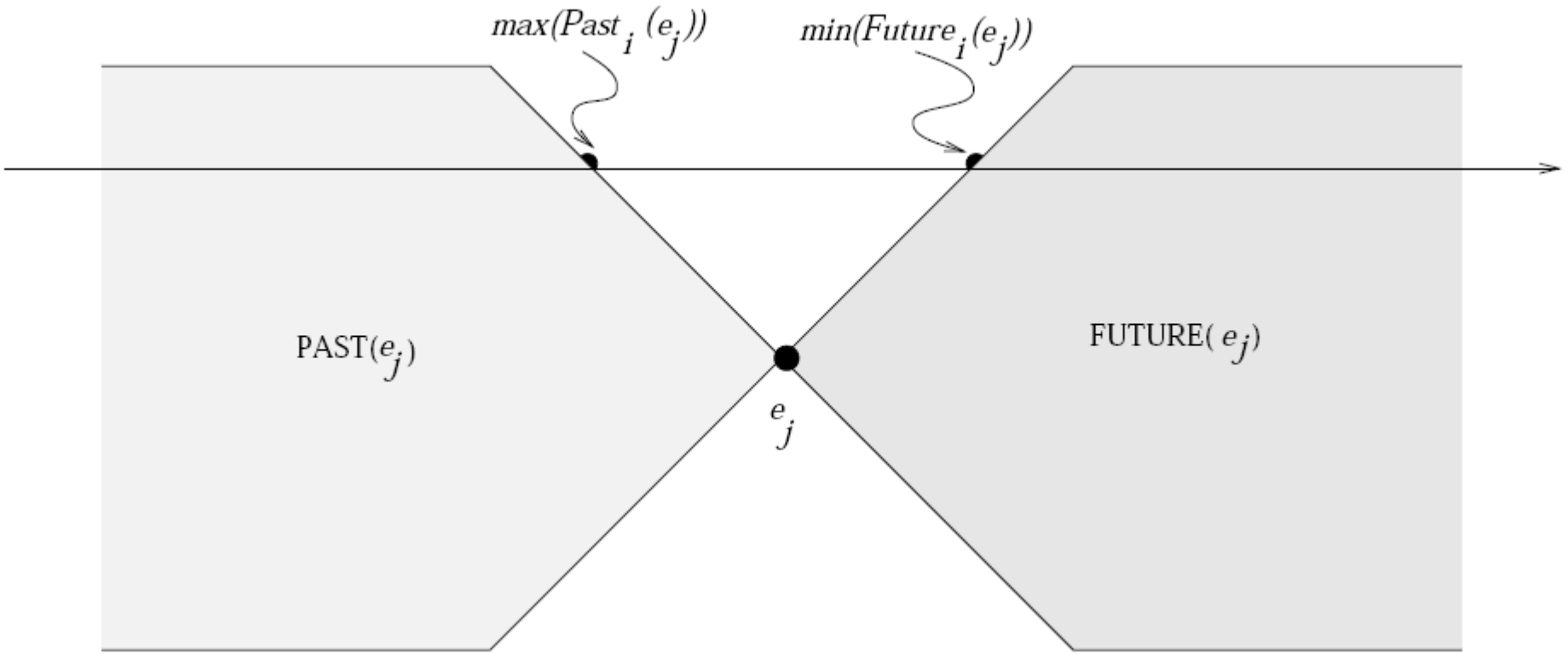

Figura 2.4: Ilustra cones de Passado e Futuro

Cones de tempo

Seja **Passado$_i$(e_j)** o conjunto de todos os eventos do Passado de e_j "**Passado(e_j)**" que estão no processo p_i.

Passado$_i$(e_j) é um conjunto totalmente ordenado por causalidade (relação → i) cujo elemento máximo é denotado por **max(Passado$_i$(e_j)).**

max(Passado$_i$(e_j)) é o ultimo evento no processo p_i que afeta o evento e_j (figura 2.4)

Cones de tempo

Seja **Max_Passado(e_j)** = $\cup_{(\forall i)}$**{max(Passado$_i$(e_j))}.**

Max_Passado(e_j) consiste no conjunto dos últimos eventos em cada processo que afetou o evento **e_j** e é denotado por **superfície do cone passado e_j**

Passado(e_j) representa todos os eventos no cone passado que afetaram **e_j.**

Cone Futuro

O **Futuro** de um evento **e_j.** Denotado por **Futuro(e_j)** contém todos os eventos que são causalmente afetados por **e_j.** (figura 2.4).

Na computação **(H,→),** **Futuro(e_j)** é definido como

Futuro(ej) = {ei |∀ei ∈ H, ej → ei}.

Cones de tempo

Define-se **Futuro$_i$(e_j)** como o conjunto de todos os eventos de **Futuro(e_j)** que estão no processo **p_i**.

Define-se **Min(Futuro$_i$(e_j))** como o primeiro evento no processo pi afetado por e_j.

Define-se **Min_Futuro (e_j)** como $\cup_{(\forall i)}${**min(Futuro$_i$(e_j))**}, Que consiste no conjunto dos primeiros eventos afetados causalmente pelo evento **e_j** em cada processo **p_i**.

Min_Futuro (e_j) é denotado por **superfície do cone Futuro de e_j**

Todos os eventos do processos p que ocorrem depois de **max(Passado$_i$(e_j)),** mas antes de **min(Futuro$_i$(e_j)) são concorrentes a ej.**

Sendo assim, todos, e apenas estes eventos da computação H que pertencem ao conjunto **H – Passado(e_j) – Futuro (e_j)** são concorrentes com o evento **e_j**

Modelos de Comunicação

Há dois modelos básicos de comunicação – síncrono e assincronia

O modelo de comunicação síncrona é do tipo bloqueante em uma mensagem enviada. O processo remetente bloqueia até que a mensagem seja recebida no destinatário.

O processo desbloqueia apenas depois saber que o destinatário recebeu a mensagem.

Logo os processos remetente e destinatário sincronizam-se para receber mensagens.

O modelo de comunicação assíncrona é do tipo não-bloqueante em uma mensagem enviada.

Depois de enviar uma mensagem o processo remetente não se bloqueia para esperar a mensagem chegar ao destino.

A mensagem é colocada em buffer até o destinatário a retirar de lá.

Modelos de Comunicação

Nenhum dos modelos é superior ao outro

Modelos assíncronos provêem maior paralelismo, pois a computação pode continuar enquanto as mensagens estão em transito.

Entretanto um estouro de buffer pode ocorrer se o remetente enviar um grande número de mensagens.

Implementações assíncronas devem ter gerencias mais complexas de buffer.

Devido ao alto grau de paralelismo e do caráter não determinístico dos sistemas assíncronos, é mais difícil implementar algoritmos distribuídos para tais modelos de comunicação.

Comunicação síncrona é mais facilidade lidar e implementar.

Devido a toda a sincronização embutida, é mais provável ocorrer queda de performance e deadlocks.