Sistemas Distribuidos I
(75.74)

Sincronismo, Orden y Cortes de Estado

Sincronización entre Procesos. Orden Local y Global. Cortes de Estado

Docentes

- Pablo D. Roca
- Ezequiel Torres Feyuk
- Guido Albarello
- Ana Czarnitzki
- Cristian Raña

Agenda

- **Sincronismo**
- Orden
- Estado y consistencia

Sincronismo | Definición

Los términos sincrónico / asincrónico **dependen del contexto:**

Ejecución de Eventos

- Sincrónico = bloqueante
- Asincrónico = no bloqueante

Comunicación entre grupos

- Sincrónico = Entidades interactúan entre sí
- Asincrónico = Entidades son independientes

- **Sistemas Digitales**
  - Sincrónico = Entidades coordinadas por el mismo reloj
  - Asincrónico = Entidades coordinadas por diferentes relojes
- **Sistemas Distribuidos**
  - **??**

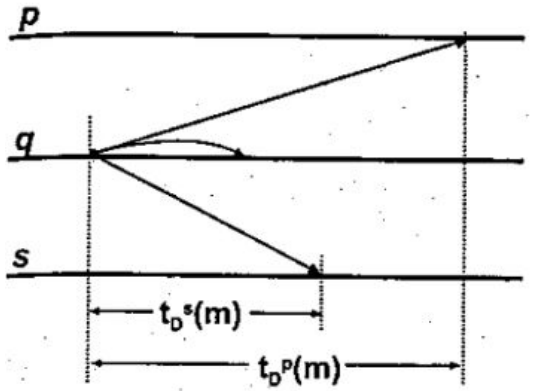
- Un algoritmo / protocolo es **sincrónico** si sus acciones pueden ser delimitadas en el tiempo
  - **Sincrónico**: Entrega de un mensaje posee un *timeout* conocido
  - **Parcialmente sincronico**: Entrega de un mensaje no posee un timeout conocido o bien el mismo es variable
  - **Asincrónico**: Entrega de un mensaje no posee *timeout* asociado
- ¿Cómo se generaliza esto para un sistema?
  - ***Steadiness***
  - ***Tightness***

Tiempo de *Delivery* ($t^P_D(m)$)

- Es el tiempo que tarda un mensaje **m** en ser recibido luego de haber sido enviado hacia **p**

Timeout de *Delivery* (T_{Dmax})

- Todo mensaje envíado va a ser recibido antes de un tiempo T_{Dmax} conocido



Verissimo, L. Rodriguez: Distributed Systems for Systems Architects

Steadiness (σ)

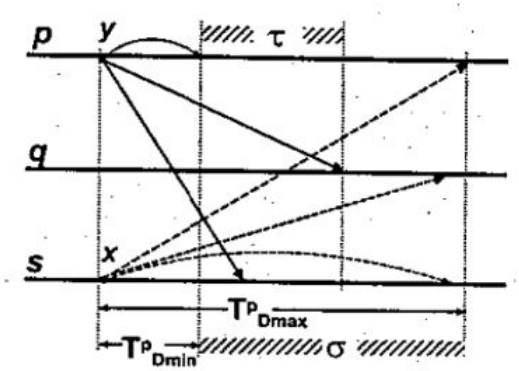
- Máxima diferencia entre el mínimo y máximo tiempo de delivery de cualquier mensaje recibido por un proceso

$$\sigma = \max(T_{Dmax} - T_{Dmin})$$

Tightness (τ)

- Máxima diferencia entre los tiempos de delivery para cualquier mensaje m

$$\forall\ p,q: \tau = \max_{m,p,q}(t^p_D(m) - t^q_D(m))$$



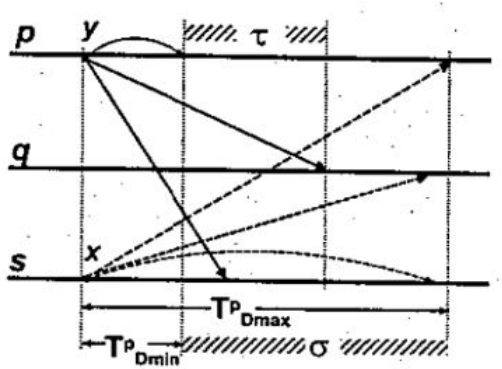
Sincronismo | Propiedades

Steadiness (σ)

- Define la varianza con la cual un proceso observa que recibe los mensajes
- Define que tan constante (*steady*) es la Recepción de mensajes

Tightness (τ)

- Define la simultaneidad con la cual un mensaje es recibido por múltiples procesos



Verissimo, L. Rodriguez: Distributed Systems for Systems Architects

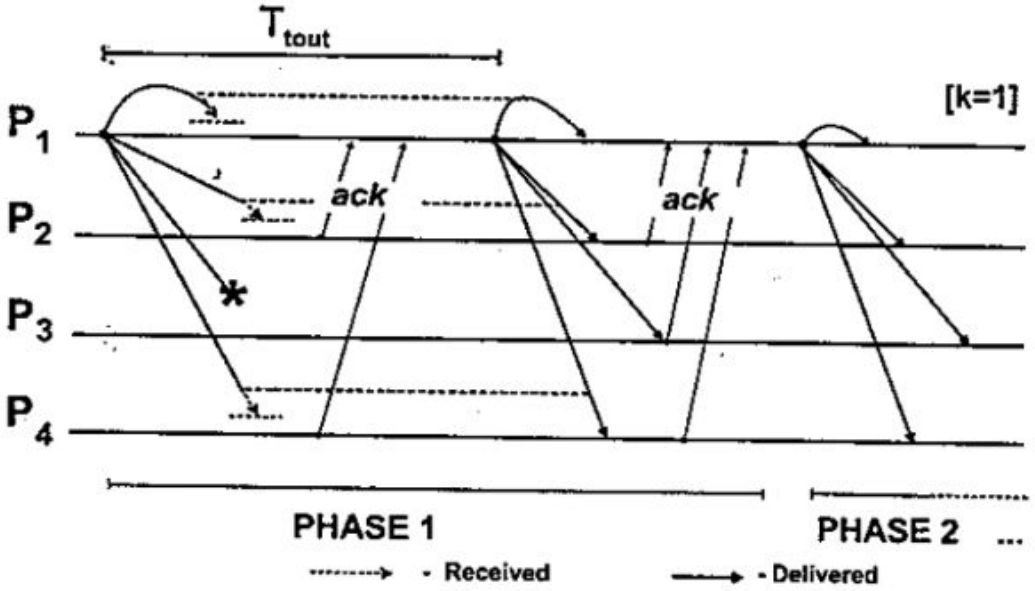


Figure 2.10. Unsteady and Untight Synchronous Protocol

Verissimo, L. Rodriguez: Distributed Systems for Systems Architects

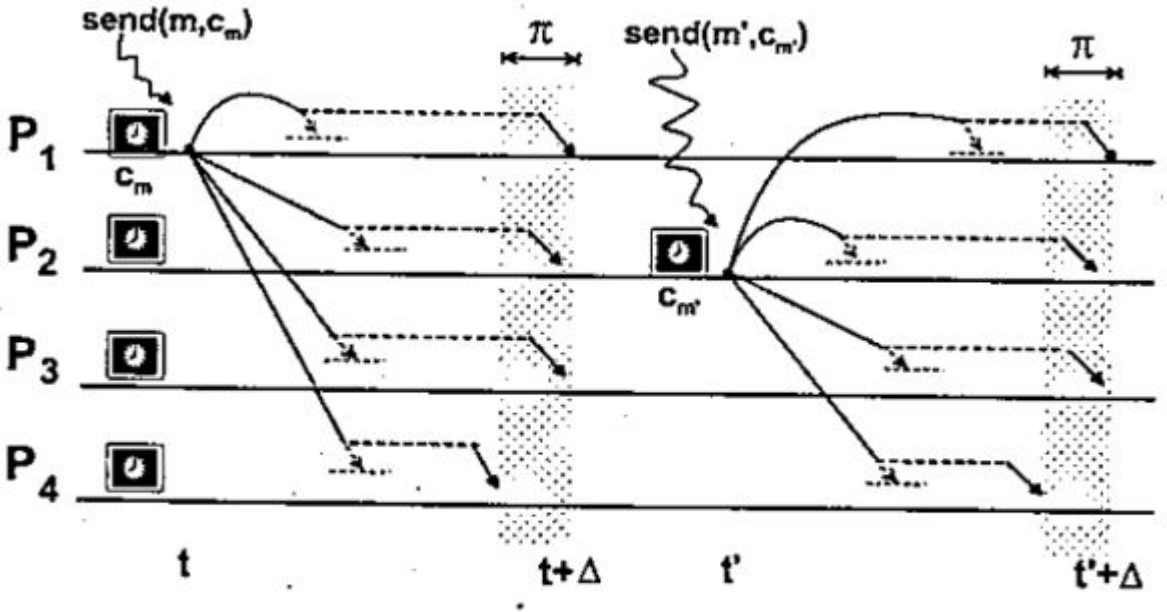


Figure 2.11. Steady and Tight Δ-protocol

Verissimo, L. Rodriguez: Distributed Systems for Systems Architects

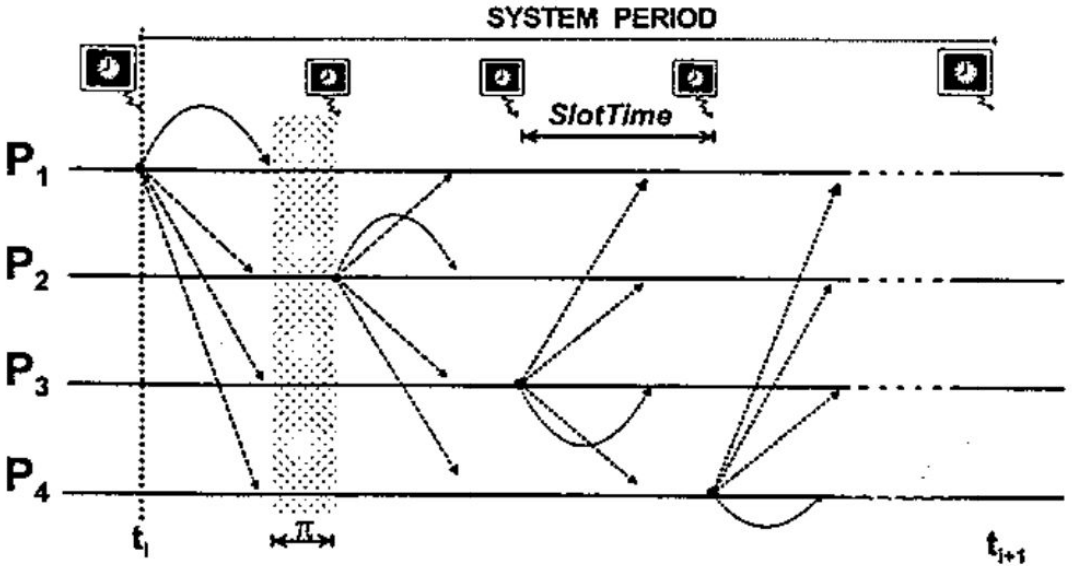


Figure 2.12. Steady and Tight TDMA Protocol

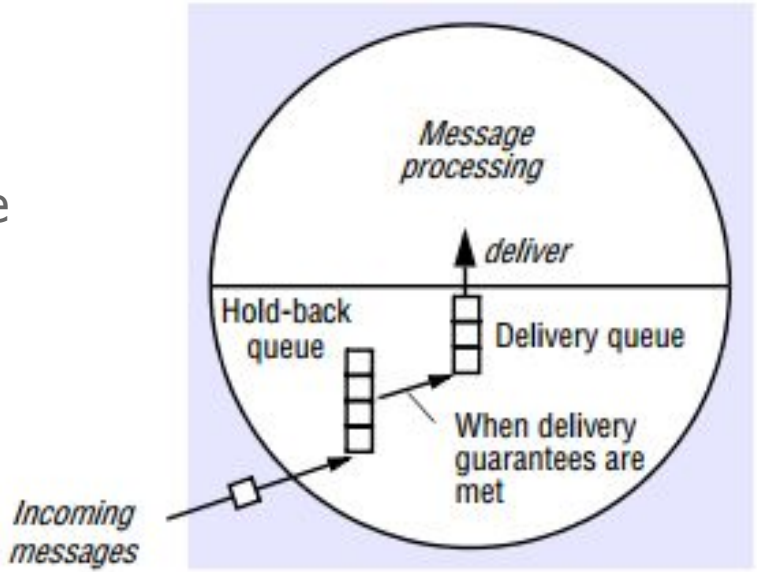
Verissimo, L. Rodriguez: Distributed Systems for Systems Architects

Agenda

- Sincronismo
- **Orden**
- Estado y consistencia

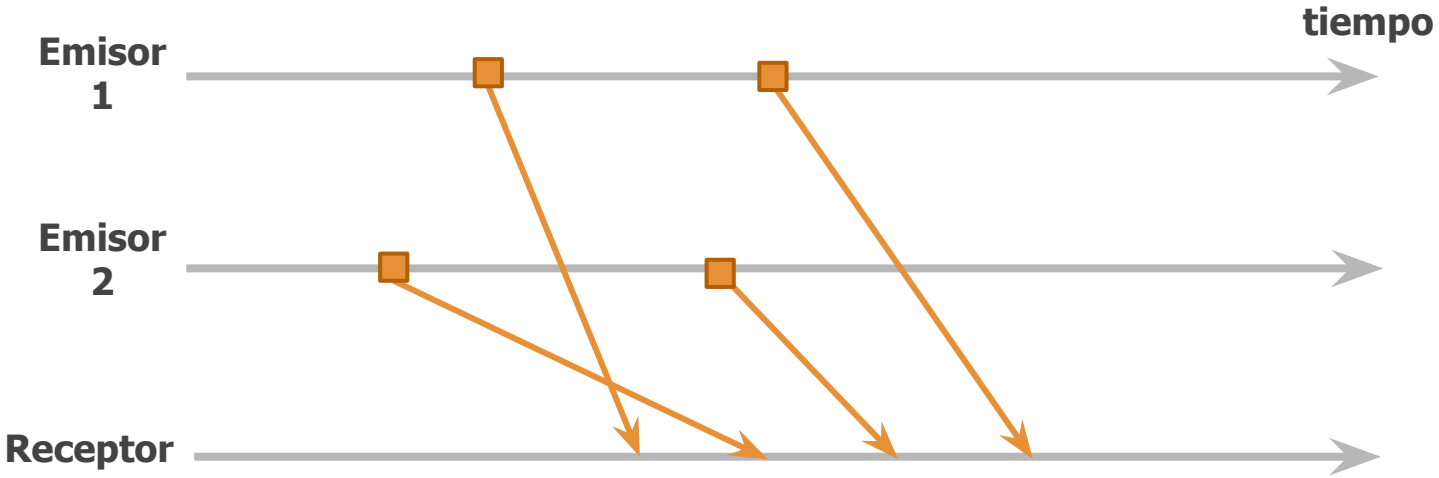
Delivery de Mensajes | *Hold-back queue*

- Envío de mensajes no es lo mismo que *delivery* de los mismos
- El delivery consiste en procesar el mensaje provocando, eventualmente, cambios en el estado del proceso
- Los mensajes se mantienen en una cola que permite controlar el momento en que se lo libera, permitiendo demorar el *delivery*
- También es posible re-ordenar mensajes en esta cola



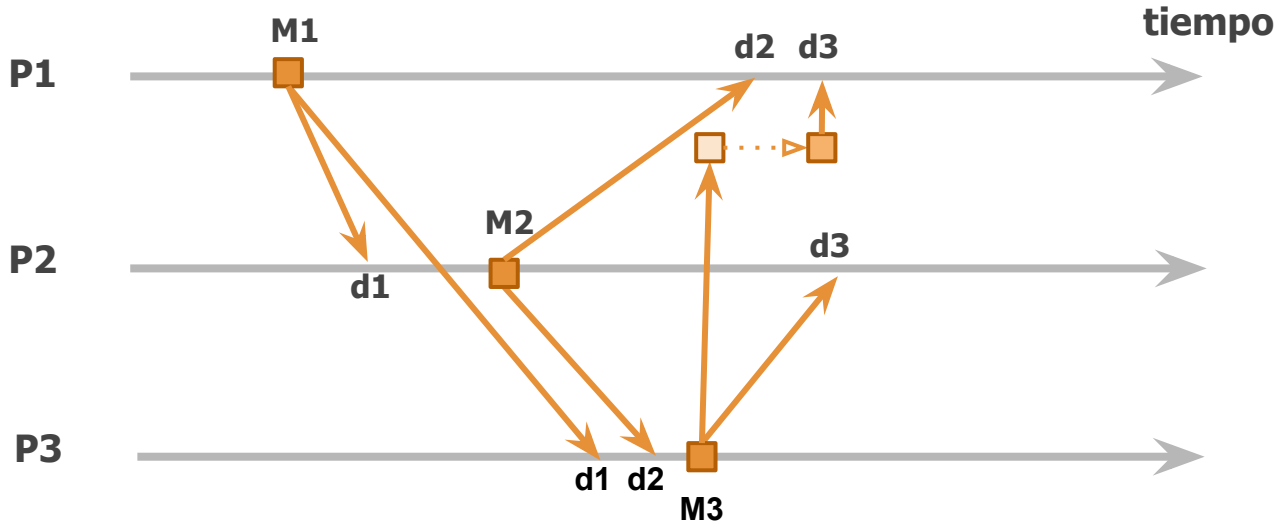
Orden FIFO

Todo par de mensajes desde un mismo emisor a un mismo receptor se entregan en el orden en que fueron enviados.



Orden Causal

Todo mensaje que implique la generación de un nuevo mensaje, es entregado manteniendo esta secuencia de causalidad sin importar el receptor.

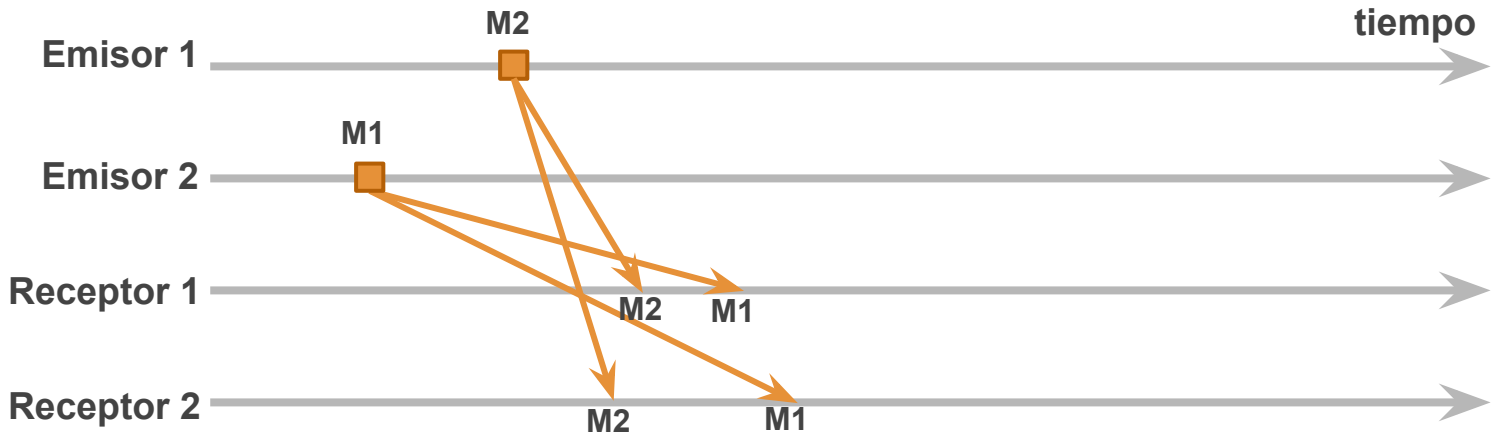


Como:
M1 -> M2 -> M3
Luego:
d1 -> d2 -> d3

(Nota: se asume entrega automática de cada mensaje dentro de su mismo emisor)

Orden Total

Todo par de mensajes entregado a los mismos receptores es recibido en el mismo orden por esos receptores.



Orden | Otras categorizaciones

Orden Lógico

Orden Temporal

Agenda

- Sincronismo
- Orden
- **Estado y consistencia**

Estado Local

Se define $\mathbf{s_j}$, el estado local del proceso j en el instante t, según

$$s_j = (X_0(t), X_1(t), ..., X_n(t))$$

con X_i= variable i del proceso j

Estado Global

Se define **S**, el estado global del sistema en el instante t, según

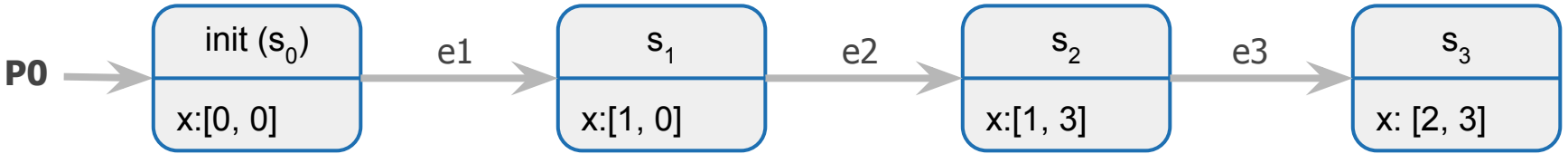
$$S = s_0 \cup s_1 \cup ... \cup s_n$$

con s_i = estado local del proceso j

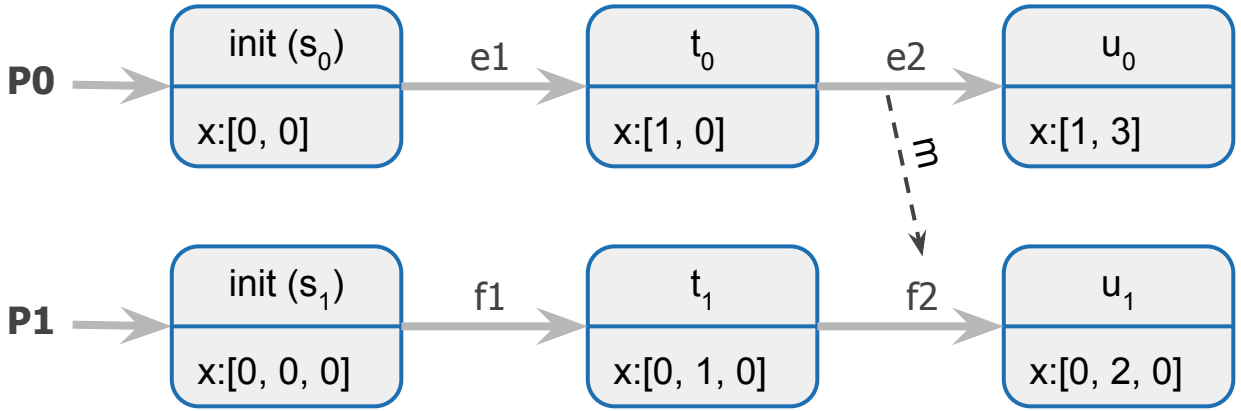
Estado | Sistema como máquina de estados

- Consiste en modelar el sistema como una serie de estados
- Un estado evoluciona al siguiente estado por la ocurrencia de un evento
- Asumiendo instrucciones determinísticas, el procesamiento de cualquier evento bajo el estado actual se puede reproducir

Es más simple de hacer cuando hay un único proceso. Ej:



Con múltiples procesos, es más difícil determinar el estado global S. Ej.:



Estado Globales Válidos: $S = s_0 \cup s_1, t_0 \cup s_1, s_0 \cup t_1$, etc

Estado Globales Inválidos: $S = u_1 \cup t_0$

Historia y Corte de Estados de un Sistema

Historia (o corrida)

Caracteriza al proceso P_i, considerándolo una máquina de estados, mediante la secuencia de todos los eventos procesados:

$$h_i = (e_0 , e_1 , e_2 , ...)$$

con e_j= todo evento aceptado por P_i

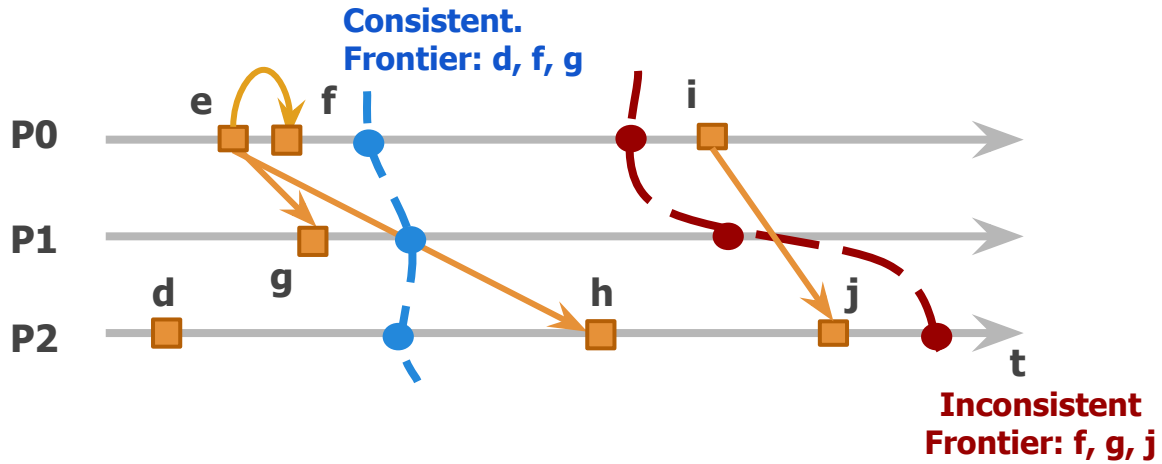
Corte

Unión del subconjunto de historias de todos los procesos del sistema hasta cierto evento k de cada proceso:

$$C = h_0 \cup ... \cup h_n = (e_0 , ... , e_k) \cup ... \cup (e'_0 , ... , e'_{k'})$$

Un **corte** es **consistente** si por cada evento que contiene, también contiene a aquellos que 'ocurren antes' que dicho evento.

$$\forall \text{ evento } e \in C: f \rightarrow e \Rightarrow f \in C$$



Historia y Cortes | Algoritmo de Chandy & Lamport

- Algoritmo que permite obtener **snapshots** de estados globales en sistemas distribuidos
- El objetivo del algoritmo es almacenar estados de un conjunto de procesos y estados de canales (snapshots) de forma que, aunque los estados no hayan ocurrido al mismo tiempo, **el estado global almacenado sea consistente**
- **Hipótesis**
  - Los procesos y los canales de comunicación no fallan
  - Canales son unidireccionales y poseen orden FIFO
  - Grafo fuertemente conexo (caminos de ida y vuelta definidos)
  - Cada proceso puede iniciar un snapshot en cualquier momento

P_i

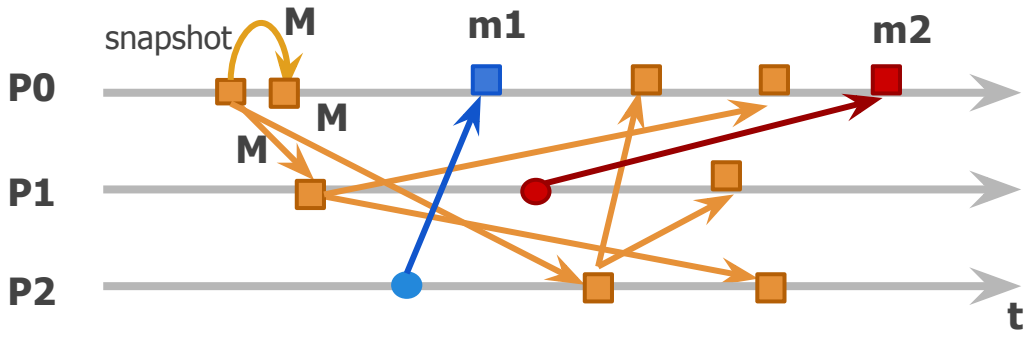
```
canales_out, canales_in
corte_i := None
mensajes := []
```

Regla de envío de Marcador

```
Al almacenar el estado en corte_i
enviar Marcador por canales_out
```

Regla de recepción de Marcador

```
Al recibir un Marcador
if P_i no grabó su estado:
  corte_i := estado de P_i
  activar_log(mensajes, canales_in)
else:
  corte_i += mensajes
  mensajes := []
```



marcadores
mensajes

corte = S0 ∪
S1 ∪
S2 ∪
(P2->P0:m1)

Comunicación *Reliable* | Propiedades y Orden

Comunicación *Reliable* (o Confiable): si se garantiza integridad, validez y atomicidad en el *delivery* de mensajes.

Uno a uno

- Trivial si contamos con protocolos sobre TCP/IP y una red segura

Uno a Muchos

- El grupo debe proveer las 3 propiedades: bajo TCP/IP, atomicidad requiere atención especial
- Definir el orden entre mensajes garantizado: FIFO, causal, total, etc.

Bibliografía

- G. Coulouris, J. Dollimore, t. Kindberg, G. Blair: Distributed Systems. Concepts and Design, 5th Edition, Addison Wesley, 2012.
  - Capítulo 14.5 Global States
  - Capítulo 15.4 Coordination and agreement in group communication
- P. Verissimo, L. Rodriguez: Distributed Systems for Systems Architects, Kluwer Academic Publishers, 2001.
  - Capítulo 2.6 Synchrony
  - Capítulo 2.7 Ordering