Università degli Studi di Bologna
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Scuola di Ingegneria

Corso di **Reti di Calcolatori M**

Comunicazione e Sincronizzazione
Sistemi con gruppi di destinatari

Antonio Corradi

Anno accademico 2014/2015

COMUNICAZIONE

Comunicazione punto-a-punto tra processi

- **Livelli di OSI e di TCP/IP**
- **Correlazione con Sistemi di nomi delle entità**
- **Semantica della comunicazione**

  May-be, At-least-once, At-most-once, Exactly-once

Primitive di comunicazione punto a punto e API

- **Primitive Sincrone/Asincrone - Bloccanti / Non**
- **Simmetriche / Asimmetriche - Bufferizzate / Non**
- **Dirette / Indirette – Affidabili / Non Affidabili**

Standard

COMUNICAZIONE di GRUPPO

Comunicazione tra insiemi di processi

Broadcast e Multicast

come ***invio generalizzato*** di messaggi a tutti i processi del sistema o a un sottoinsieme di processi del sistema (gruppo)

In una località si possono realizzare facilmente (vedi LAN)

Su località diverse non si riescono a ottenere facilmente

per *incapacità espressiva*, *eccesso di overhead*, *mancanza di QoS*, …

PROBLEMI SEMANTICI del multicast e broadcast

- Come trattare le risposte?
- nessuna attesa
- attesa di una sola risposta
- attesa di alcune (quante?) risposte
- attesa di tutte le risposte

COMUNICAZIONE di GRUPPO

IP Broadcast

Broadcast limitato e diretto (in rete locale)

IP Multicast **più costoso**

Multicast per indirizzi di classe D

Multicast locale e …

I protocolli **Internet Group Management IGMP** considerati in Internet da molto tempo (**RFC 1112** e **2236**) **realizzano il multicast locale**

Spesso sono stati operativi spesso sulla sola rete locale e sono realizzati in molte forme diverse e non compatibili

Multicast (più) globale

Ad esempio, **un multicast realizzato attraverso flooding tra reti**

Si fa passare il pacchetto una volta sola (stato sul nodo) e attraverso tutte le code di out eccetto quella da cui è arrivato (per quanto si mantiene stato?)

Modo tradizionale di fare routing con politica semplice e a basso costo(!)

COMUNICAZIONE di GRUPPO

IP Multicast

Ma la **QoS?**

Che garanzie si sono che il messaggio sia veramente arrivato (oltre la semantica best-effort)?

Non ci sono molte garanzie di qualità sulle realizzazioni di IGMP *ossia*
non sappiamo se i messaggi siano stati consegnati tutti e in che ordine

Ad esempio, si usa un dimensionamento a priori del time-to-live (TTL) dei datagrammi (ad indicare la penetrazione e il costo)

TTL=0 invio locale	TTL=1 locale alla connessione
TTL<=32 locale di area	TTL <=64 locale alla regione
TTL<=128 locale al continente	TTL>128 globale

COMUNICAZIONE di GRUPPO

IGMP come esempio di supporto a Multicast locale

(RFC 1112 e 2236)

IP multicast consente di mandare un unico pacchetto a più destinatari, usando *nomi di classe D per identificare il gruppo*, non solo localmente

Necessario un **supporto di router di controllo**

Ogni rete locale prevede almeno un router di IGMP che gestisce il traffico locale in arrivo o in partenza e controlla il gruppo con messaggi IGMP
Si possono anche avere più multicast router

IGMP v1 prevede due soli messaggi semplici con approccio C/S

IGMPQUERY	un router verifica in modo periodico la presenza di host che rispondono ad un indirizzo
IGMPREPORT	un nodo segnala un cambiamento di stato al router nei confronti del gruppo (vedi dopo: **join / leave**)

COMUNICAZIONE di GRUPPO

IGMP v1

La gestione è a carico del router

Il ruolo **attivo** viene sempre dal router che invia query ad intervalli: i nodi rispondono alla query per segnalare presenza o non rispondono (**join con ritardo alla prima query**)

non si prevedono messaggi di **leave** dal gruppo e si comandano operazioni di raggruppamento (un solo report di un nodo per una rete locale)

IGMP v2 (supporto per join / leave)

La gestione può avvenire anche con più router su LAN

Si dirime la *interferenza tra router* con un ordinamento dei numeri di IP

La seconda versione prevede la iniziativa da parte dei nodi e l'invio di un messaggio **esplicito di leave** (ossia sgancio dall'indirizzo di gruppo) da parte dei nodi che devono avvertire il gestore

PRINCIPI di ROUTING MULTICAST

il Multicast deve impegnare meno risorse possibile nella trasmissione dei dati da inviare ai riceventi

Per un buon uso delle risorse ed evitare un impegno eccessivo di banda sono tipiche alcune ipotesi di base

- supporto **per un solo mittente**
- supporto **per un numero variabile n di riceventi**, che si possono anche aggiungere e togliere dinamicamente

La idea è di riuscire ad impegnare al meglio il **principio della condivisione** e di avere la possibilità di **mandare una sola copia**, invece di inviare **N copie del messaggio**

multicast (1 messaggio) **anziché molti unicast** (**N** messaggi)

Date le ipotesi, si prevede di identificare un **albero** che permetta di trovare dei **percorsi condivisi dal mittente ai destinatari** presenti con il **massimo della condivisione dei cammini dalla radice alle foglie**, albero in continua riorganizzazione, vista la dinamicità dei riceventi che si possono aggiungere e togliere

L'albero deve considerare solo le **località dei riceventi attivi** e trascurare quelle in cui non ci sono riceventi correnti:

ROUTING MULTICAST (STABILE)

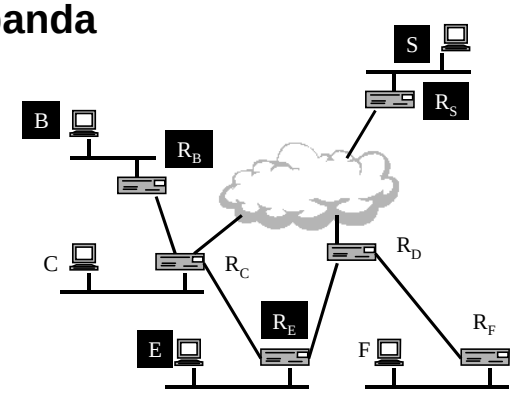
il Multicast prevede la **identificazione** di un **albero** (dinamico) dal **trasmettitore ai riceventi per invii ripetuti**

radice di un albero al trasmettitore, i riceventi finali sono i **nodi foglie**, i router intermedi **nodi intermedi** dell'albero

- il gruppo è aperto con un **solo mittente**
- la appartenenza al gruppo è **dinamica**
- il **join al gruppo** è a carico delle foglie
- **i cammini condivisi ottimizzano la banda**

L'albero è **estremamente dinamico**

Consideriamo il caso di un **host S** che trasmette e **B** ed **E** parte del gruppo riceventi



MULTICAST: SPANNING TREE

Consideriamo solo i router come destinatari e vogliamo costruire un albero sul grafo di connessione

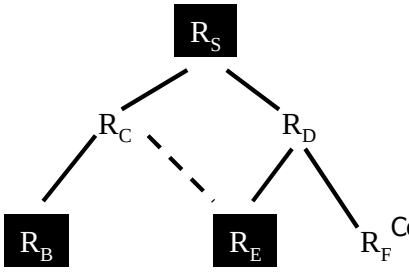
Primo passo (Richiesta identificazione foglie)

Si vuole creare un albero (**spanning tree**) per connettere la radice ai nodi foglia noti, *usando le informazioni del protocollo di routing unicast* e i *singoli cammini*

Si comincia mandando un messaggio in **flooding** verso **tutti** i possibili partecipanti con obiettivo di creare un **bone multicast**

Le risposte che arrivano dai destinatari permettono alla radice di identificare i cammini a distanza minore

alcuni nodi riceventi sono raggiunti da più cammini



MULTICAST: CAMMINI MULTIPLI

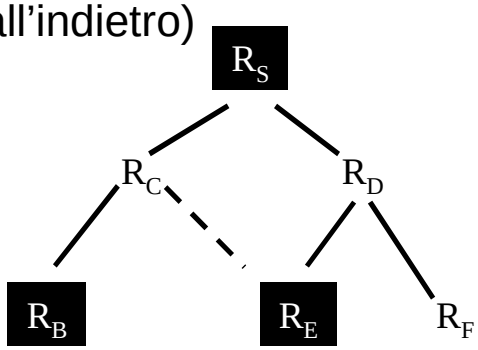
Secondo passo (Percorsi dalle foglie verso la radice)

Ogni foglia segnala i **percorsi** (all'indietro) e può anche **selezionare percorsi nuovi** (anche non minimi) per andare dalla radice alle foglie

- si mandano i messaggi se arrivano con un cammino minore dalla sorgente, e li si invia su tutti i link in uscita eccetto quello di ricezione (per identificare altri percorsi, eventualmente migliori)
- si scelgono alcuni cammini e non si considerano tutti gli altri
- si passano alcuni dei messaggi di ritorno dalle foglie alla radice

Reverse path forwarding (cammino all'indietro)

Per ogni nodo raggiunto da più cammini
la root può selezionare il cammino
Re raggiunto da Rd
in modo da potere scegliere
nuove parti condivise



MULTICAST in AZIONE

Normale routing: la operatività normale richiede di **continuare ad operare** mentre **l'albero** è in **fase di identificazione …**

Distance Vector

Si devono usare informazioni next hop (o usare poisoned reverse) per bloccare cammini troppo lunghi

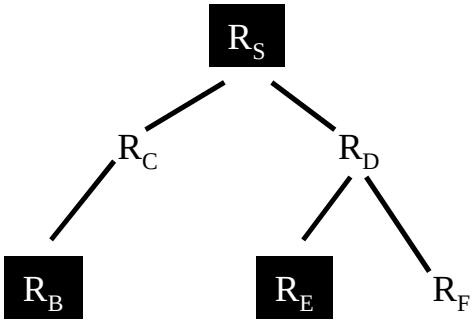
Link State

Si devono costruire tutti gli alberi shortest path per tutti i nodi e usare regole "tie break" per dirimere conflitti

Reverse Path Broadcast (2 passo) per **eliminare Cammini Multipli**

Si fa un broadcast **dalle foglie verso la radice** durante la normale operatività di routing

La radice riceve nuovi cammini e può anche riorganizzare i cammini tentando anche aggregazione di maggiori sottopercorsi dell'albero



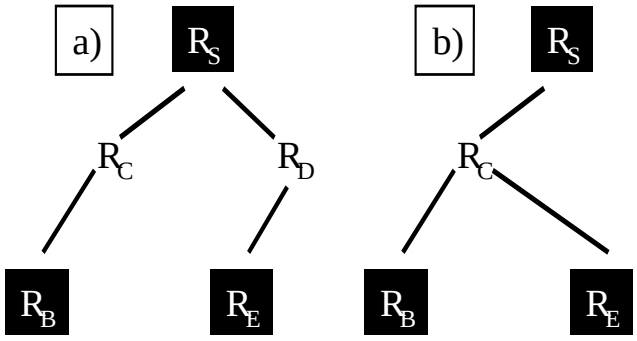
REVERSE PATH BROADCAST

Reverse Path Broadcast consente di scegliere tra cammini diversi per organizzare l'albero al meglio e minimizzare il numero di messaggi scambiati e la banda impegnata

> Con un broadcast dalle foglie (appunto **Reverse Path Multicast**) si trovano **cammini che congiungono una foglia con la radice che possono non essere stati esplorati**

Sta alla radice scegliere la migliore organizzazione

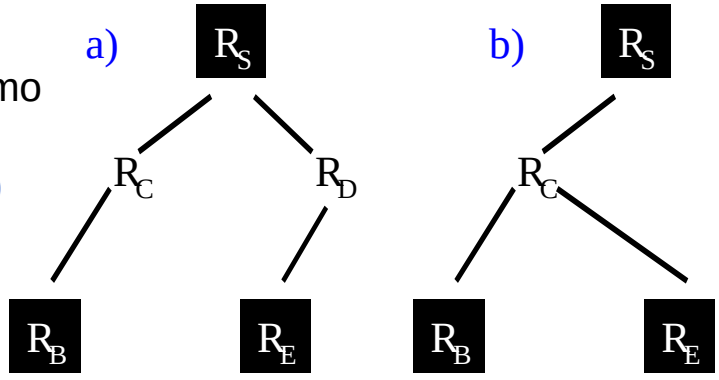
Reverse Path Multicasting (**RPM**) per riorganizzare l'albero (pur con costo elevato)



MULTICAST: PRUNING e GRAFTING

PRUNING (potatura **dinamica**) e **GRAFT** (reinserimento)
si escludono quei router che non hanno al momento alcun ricevente con messaggi di taglio che passano nell'albero
Si deve ricostruire l'albero in caso di variazioni

Reverse Path Multicasting (**RPM**) fatto in modo autonomo dalle foglie per consentire il taglio **PRUNING** - da a) a b) e il reinserimento di parti dell'albero **GRAFT** - da b) ad a)



ROUTING MULTICAST

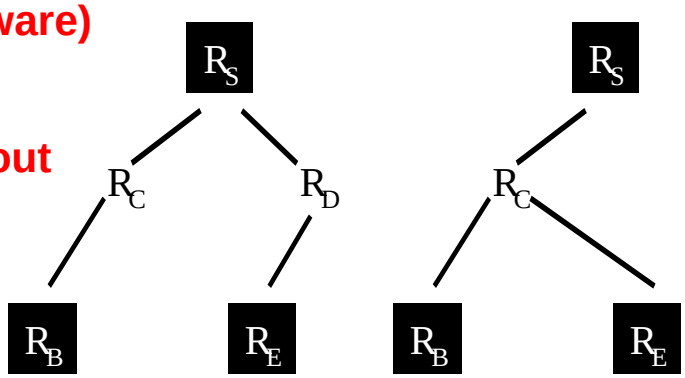
Reverse Path Multicasting dalle foglie alla radice

- usato in molti protocolli multicast
- mantiene lo stato **per-sender**, per **group**

Reti che non hanno membri sono tolte dall'albero e nuove possono rientrare nel gruppo (graft esplicito dal basso) senza riorganizzare ex novo tutto l'albero

Si mantiene lo stato (software) per un intervallo di durata massima stabilita che decade dopo un timeout SOFT-STATE

Critica la determinazione degli intervalli per il RPM



Molti PROTOCOLLI MULTICAST

Molti protocolli diversi di **routing multicast** (**incompatibili**) anche in competizione e supportati da comunità diverse

DVMRP (**RFC 1075**) **D**istance **V**ector **M**ulticast **R**outing **P**rotocol

di tipo RPM basato sul semplice RIP modificato

e molto usato in MBONE (multicast backbone)

Gli aggiornamenti avvengono con messaggi mandati usando percorsi speciali (tunnel) e sfruttando solo alcuni nodi

MOSPF (RFC 1584) **M**ulticast **O**pen **S**hortest **P**ath **F**irst Protocol

Estende il link-state adatto per reti grandi basato su RPM e soft-state

Parte dalla mappa delle reti e la usa per calcolare lo shortest path per tutte le destinazioni

Ottimizza gli alberi ed elimina i cammini non usati

non UNICO STANDARD MULTICAST

PIM (**RFC 2117**) **P**rotocol **I**ndependent **M**ulticast Protocol

che usa qualunque protocollo unicast con distinti modi adatti per sistemi

Sparsi intesi come a bassa probabilità di nodi multipli nella stessa LAN e **Densi** in cui ci sono molti nodi vicini

Densi uso di flooding e prune semplificato rispetto al DVMRP

Sparsi eliminando il maggior numero di router intermedi così da semplificare la struttura dell'albero

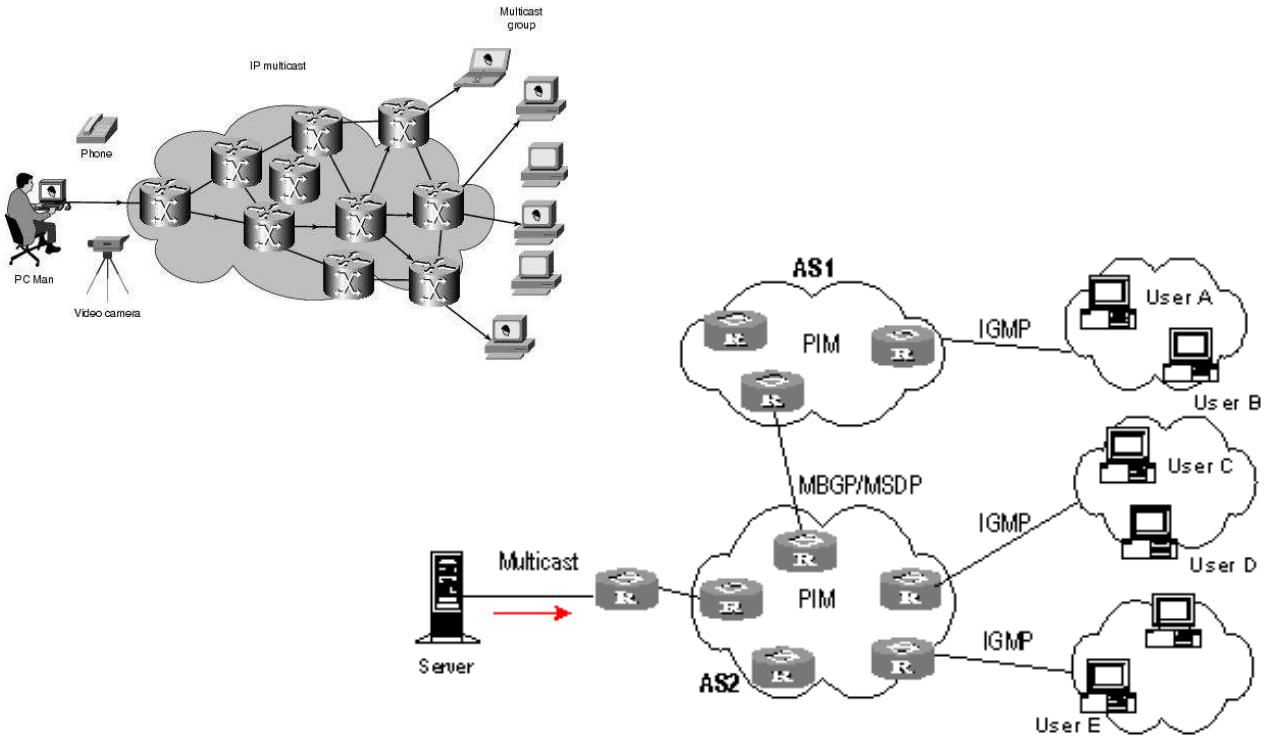
CBT (**RFC 2201**) **C**ore **B**ased **T**rees

adatto per router core da scegliere opportunamente

Si mantengono alcuni nodi core fissi e gli alberi sono unificati senza definire uno stato per ogni sender, e per ogni gruppo

Si arrivano anche ad usare alberi sub-ottimi ma non si paga il costo di organizzazione della struttura di connessione per ogni multicast

PROTOCOLLI MULTICAST



APPLICAZIONI per MULTICAST

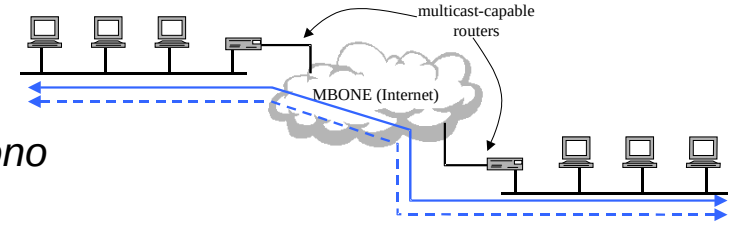
Applicazioni multimediali (integrate o meno con Internet)

- voce a audio **RAT**, RealAudio
- video **VIC**
- testo **NTE**
- whiteboard **WBD**

In generale si usano tutte **sessioni multicast**

Ogni sessione usa protocolli ad hoc per aprire due canali multicast per ogni applicazione

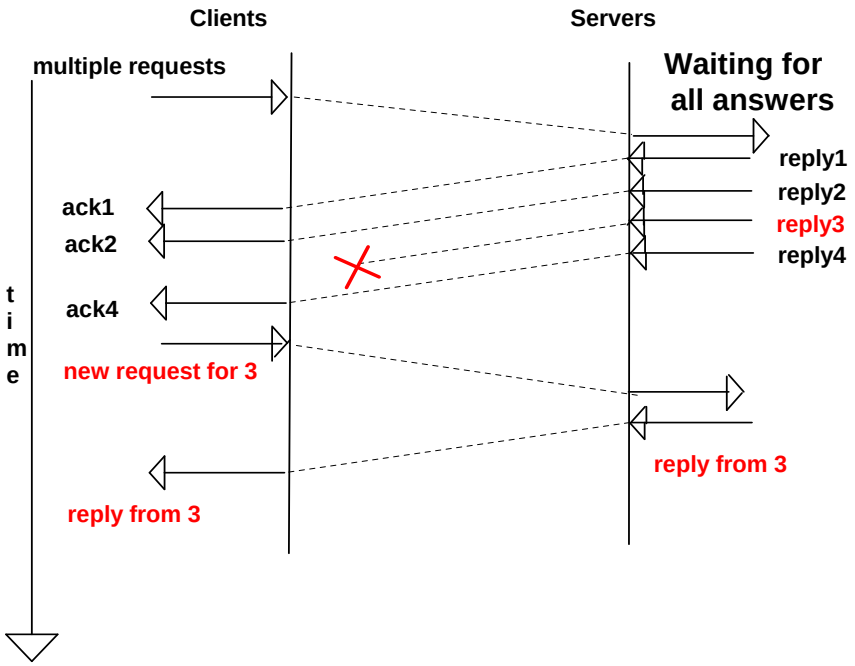
Nel caso si voglia rendere nota la conferenza ci si basa su sistemi di nomi servizio di nomi ***Session Directory RendezVous SDR***
che consente di ottenere informazioni sul flusso



Queste applicazioni richiedono router che siano in grado di garantire una corretta **QoS**

COMUNICAZIONE di GRUPPO

Semantica: Uso di ritrasmissioni selettive? Quante volte?
la semantica della primitiva dipende da queste decisioni



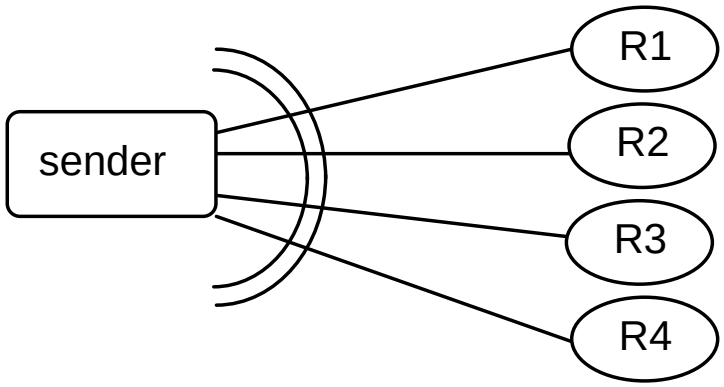
Sollecito **selettivo**
vs.
Sollecito **globale**

Conferma **positiva**
vs.
Conferma **negativa**

COMUNICAZIONE di GRUPPO

SEMANTICA DEL MULTICAST

La azione di **multicast** dovrebbe rendere atomica l'operazione di invio multiplo ad un gruppo o darle un qualche significato



Motivazioni per l'interesse

- *localizzazione di oggetti in un sistema*
- *fault tolerance*
- *uso di dati replicati e streaming*
- *cambiamenti multipli su insiemi di entità*

COMUNICAZIONE di GRUPPO

DUE ASPETTI SEMANTICI DEL MULTICAST

Reliability o affidabilità

ricezione da parte di **tutti i componenti del gruppo**

reliable ⇨ garanzia di consegna

unreliable ⇨ 1 solo tentativo (Chorus)

Atomicità

ricezione da parte di tutti i componenti del gruppo

con possibilità di **ordinamenti diversi** per azioni diverse

I due aspetti possono e devono essere considerati in isolamento

Elemento Fondamentale

dobbiamo pensare non solo alla semantica di una sola azione, ma anche **all'ordinamento** dei **messaggi in caso di più azioni**

RELIABLE MULTICAST

***Affidabilità* si manifesta se ci sono problemi per**

- **omissione di un messaggio**
- **crash del mittente**
- **crash di un ricevente**

Necessità di **identificazione** e **recovery** del guasto attraverso il ***monitoring*** delle azioni di **multicast** e del **gruppo**

- **controllo di ogni comunicazione in atto**
- **eventuale ritrasmissione**
- **rimozione dei componenti falliti**
- **protocollo di rientro nel gruppo**

Si devono considerare **costi aggiuntivi** per la **identificazione** e il **recovery**

da svolgersi in ogni caso (anche a fronte di guasti)

RELIABLE MULTICAST

Implementazione

- Invio di ogni messaggio a tutto il gruppo ed attesa
  timeout e **ritrasmissione** (chi controlla il protocollo?)
- **Quanto** si aspetta? Problemi di efficienza
- E se fallisce il **controllore**?
  Quis custodiet custodes? (Giovenale)

hold-back → il supporto trattiene il messaggio fino a che non si è sicuri che sia quello giusto in ordine

In caso di numerazione densa, il messaggio è ritardato fino alla comparsa di tutti i precedenti (se ha numero 3, ordinato dopo 1 e 2)

negative ack → in piggybacking si invia un contatore usato per indicare eventuali perdite (in modo selettivo)

ORDINAMENTO FIFO

L'altro aspetto ATOMICITÀ è legato alla semantica

con **atomicità** intendiamo la proprietà di **ricezione in un qualche ordine da parte di tutti gli elementi vivi del gruppo**

Ordinamento FIFO → dallo stesso processo a tutti i riceventi per sequenze di messaggi multicast in successione

- In caso di ordinamento FIFO, due messaggi multicast dello stesso mittente arrivano a tutti in ordine
- Ad esempio, m1 e m2 da S1, e m3 e m4 da S2 arrivano a tutti rispettando l'ordine di invio dei due destinatari
- sono compatibili molte sequenze m1 m2 m3 m4, m1 m3 m2 m4, m1 m3 m4 m2, m3 m4 m1 m2, m3 m1 m2 m4, …

Possiamo usare un supporto che garantisce già il FIFO

Altrimenti → dobbiamo garantirlo (numerazione messaggi)

ORDINAMENTO FIFO

Il rispetto dell'ordinamento FIFO garantisce che i messaggi di uno stesso mittente al gruppo (e le sue richieste) siano ricevuti dal gruppo nell'ordine con cui sono stati inviati

Il rispetto dell'ordinamento FIFO non garantisce una proprietà che nel mondo reale tendiamo a considerare in relazione a più mittenti

- A manda una news Na
- B riceve la news e invia una risposta Nb
- C riceve prima Nb poi Na
- D riceve prima Na poi Nb

Dobbiamo considerare la relazione causa/effetto tra i due mittenti

ORDINAMENTO CAUSALE

Ordinamento CAUSA-EFFETTO che può legare eventi di processi mittenti diversi

Ordinamento CAUSALE → eventi che siano correlati da una relazione causa-effetto all'esterno del gruppo devono essere riconosciuti come tali e arrivare in modo congruo a tutti (essere recapitati a tutti)

prima la causa poi l'effetto

In caso di ordinamento CAUSALE, se due messaggi multicast sono legati dalla relazione devono essere considerati da tutti in ordine

Ad esempio, m1 e m2 da S1, e m3 e m4 da S2, e m1 causa m3, arrivano a tutti rispettando l'ordine FIFO e CAUSALE

sono compatibili molte sequenze

m1 m2 m3 m4, m1 m3 m2 m4, m1 m3 m4 m2 NON m3 m1 m4 m2

Non ci sono supporti che garantiscano il CAUSALE

Come possiamo garantirlo?

ORDINAMENTO CAUSALE

Il rispetto dell'ordinamento CAUSALE garantisce che i messaggi in relazione causa-effetto di mittenti diversi siano ricevuti dal gruppo nell'ordine

Il rispetto dell'ordinamento CAUSALE per un unico mittente assomiglia al FIFO ed è facile da realizzare

Il rispetto dell'ordinamento CAUSALE non cattura **situazioni del mondo reale** che tendiamo a dare per scontate in relazione a più operazioni

A richiede una operazione Na; B richiede una operazione Nb

Le operazioni non sono correlate

C riceve prima Nb poi Na, D riceve prima Na poi Nb

ORDINAMENTO ATOMICO

Non ci sono relazioni esterne che ci guidano, ma è bene che il gruppo si comporti in modo ordinato e sensato, con tutti gli elementi che fanno le operazioni nello stesso ordine

Ordinamento ATOMICO garantisce che i messaggi significativi siano recapitati nello stesso ordine a tutti i componenti del gruppo

spesso non importa un ordine prestabilito, **importa che sia lo stesso**

Se una copia C decide di ricevere prima Nb poi Na, tutti devono seguire quella decisione

Nb chieda di applicare un interesse, Na di fare un prelievo

Naturalmente esistono molti **ordinamenti atomici** che si possono considerare sulle operazioni di un gruppo

ORDINAMENTO ATOMICO

In ambito distribuito proporre e rafforzare ordinamenti costa

(il coordinamento tra le entità del gruppo o del supporto che numera)
e tendiamo a imporlo solo quando è il caso
Costo minimo: nessun ordinamento → ognuno lavora in modo libero e indipendente dagli altri

Ordinamenti FIFO e CAUSALE sono ordinamenti che si tendono ad imporre solo per alcuni eventi nel sistema

Ordinamenti parziali

Ordinamento ATOMICO è un ordinamento che si tende ad imporre su tutti gli eventi nel sistema gruppo

Ordinamento totale o globale

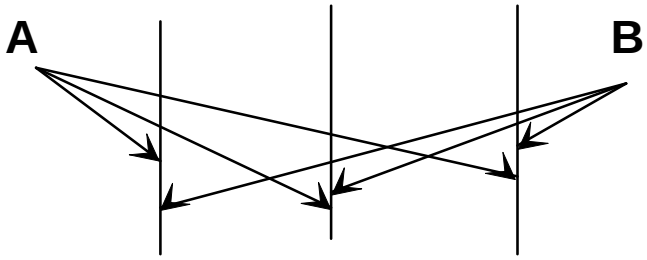
ORDINAMENTI ATOMICI

Ovviamente, dato un gruppo e un insieme di eventi che arrivano dall'esterno, possiamo avere molti ordinamenti atomici differenti
Quanti?

Ordinamenti ATOMICI

Dei molti ordinamenti atomici, alcuni possono rispettare il **CAUSALE** e il **FIFO**, altri solo il **FIFO**, altri solo il **CAUSALE**, altri nessuno dei due

I costi dell'atomico sono molto diversi



ORDINAMENTO dei MULTICAST

Multicast FIFO con ordinamento solo dallo stesso mittente

i messaggi di uno stesso mittente arrivano con mittente e numero d'ordine e sono consegnati in ordine

Multicast CAUSALE con ordinamento causale (logico)

i messaggi arrivano e sono consegnati in un ordine per il gruppo che rispetta la relazione se l'evento **A** viene prima di **B**

ordinamento di Lamport

Multicast ATOMICO richiede lo stesso ordine (**qualunque sia**) per tutti i messaggi per ogni componente del gruppo

multicast ATOMICO impone un ordinamento totale o globale dei messaggi che arrivano e devono essere consegnati nello stesso ordine ad ogni componente del gruppo

Sottolineiamo che un atomico non sussume necessariamente gli altri due o uno dei due

IMPLEMENTAZIONE dei MULTICAST

Multicast ATOMICO richiede lo stesso ordine

Un modo semplice di implementarlo è di avere nel gruppo un **elemento di accesso** che ordina i messaggi man mano che arrivano con numerazione crescente prima di smistarli al gruppo

Ogni componente riceve i messaggi ed è tenuto a proporli al livello applicativo solo nell'ordine prescritto

Gli svantaggi di questa implementazione sono

Problemi in caso di ***guasto del coordinatore*** (che decide la politica)

Unfairness della gestione: alcuni vicini al coordinatore o favoriti dallo stesso possono piazzarsi continuamente prima di altri (che hanno fatto la richiesta prima)

Soluzione a costo basso, ma molto poco giusta

→ *coordinatore* ***mobile*** *(****token circolante****)*

Molte altre soluzioni anche più dinamiche (Lamport, ring, etc.)

SINCRONIZZAZIONE

Sincronizzazione significa imposizione di ordinamenti su eventi

vincoli sull'ordine temporale di alcuni eventi di un sistema distribuito

necessaria per provvedere una **vista consistente** del sistema alla **globalità** dei processi che comunicano

Comunicazione e Sincronizzazione sono spesso correlate, ad esempio:

- *sincronizzazione sender / receiver* di un messaggio
- controllo di attività *cooperanti*
- *serializzazione di accessi* a risorse condivise
- N processi in accesso ad una risorsa (mutua esclusione)

quindi, si devono imporre ***ordinamenti sugli eventi significativi***

SINCRONIZZAZIONE CLOCK

Sincronizzazione via TEMPO FISICO ed OROLOGI FISICI

Unico tempo se assumiamo di avere

- un **unico clock** disponibile a tutti i nodi oppure
- un **clock** per ogni nodo tutti perfettamente sincronizzati

Questo ipotesi di lavoro è perfettamente accettabile in sistemi concentrati o limitati, ma assolutamente non garantita in ambiti distribuiti e globali

Si è definito **Universal Coordinated Time (UTC)** che si basa sulla trasmissione del valore e sulla correzione locale

Alcuni sistemi si basano sul **coordinamento dei clock** di interesse

> Un nodo accerta il tempo da tutti i componenti del gruppo, lo media e lo distribuisce come tempo dell'intero gruppo (Berkeley time)

SINCRONIZZAZIONE CLOCK

NTP - Network Time Protocol propone un protocollo basato su UTC e sulla sincronizzazione con questo per ottenere un **accordo sugli orologi**

Si cerca di superare gli eventuali ritardi di trasmissione del tempo comune attraverso **politiche di filtraggio statistiche** legate al comportamento storico del server

> Si considera una **gerarchia di server**, in cui ogni nodo trasmette il tempo ai **vicini di livello inferiore** (sottoalbero)
>
> i **primari** sono più accurati e mano a mano che ci si allontana dalla **radice** abbiamo una minore accuratezza
>
> Bisogna anche considerare la possibilità di guasto di un **server**

Il problema che si può verificare, basandosi su clock non perfettamente sincroni, è che un evento avvenuto successivo venga etichettato e considerato precedente rispetto ad un evento che lo segue in tempo (sincronizzazione sbagliata)

SINCRONIZZAZIONE

Sincronizzazione via TEMPO FISICO si scontra con le difficoltà di garantire la sincronizzazione di orologi e comporta un notevole overhead e un margine di errore

Ma la precisione richiede di coordinare sempre più frequentemente i diversi clock, in più potremmo avere conflitti e drifting degli orologi

Si considerano ***sincronizzazioni*** non basate su algoritmi complessi di accordo del ***clock fisico*** ma basate su ipotesi che restringono l'interesse a un **sottoinsieme** degli **eventi** del sistema globale

La logica è quella di lavorare ***su una località di eventi*** (considerando un sottoinsieme degli eventi) e di creare un accordo rispettando alcune ipotesi che ***limitino l'overhead*** **e il** ***costo*** **dei protocolli**

STRATEGIE di SINCRONIZZAZIONE

Metodi di Sincronizzazione

Ordinamento dei tempi logici di Lamport

timestamp (indicatori di tempo) per etichettare gli eventi ed ordinarli → clock logici e relazione "happened before"

Token passing

token che viene passato in un anello logico e consente di ordinare gli eventi

Ordinamento degli eventi sulla base della priorità

uso della priorità dei processi per ordinare gli eventi correlati

sistemi real-time

RELAZIONE di LAMPORT

Lamport si propone di **ordinare gli eventi in un sistema distribuito prescindendo dal tempo fisico**

Si cominciano a considerare solo **alcuni eventi** del sistema distribuito in uno scenario costituito da **processi** che hanno una loro storia interna e possono esibire un comportamento:

1) locale: *generando eventi localmente*

2) remoto: *inviando messaggi ad un altro processo*

Ordinamento deve tenere conto di alcuni eventi significativi e permettere un **ordinamento** su cui si possa stabilire una **corretta** sincronizzazione con **costi accettabili** e **non troppo onerosa da realizzare**

sulla base di **comunicazione** e di una **analisi locale delle informazioni**

RELAZIONE HAPPENED-BEFORE →

Ordinamento degli eventi per un **insieme di processi che comunicano attraverso scambio di messaggi** basato sulla causalità introdotta dalle azioni di scambio di messaggi

1) Se a e b sono eventi dello stesso processo ed a è eseguito prima di b , allora a → b (***ordine locale***)

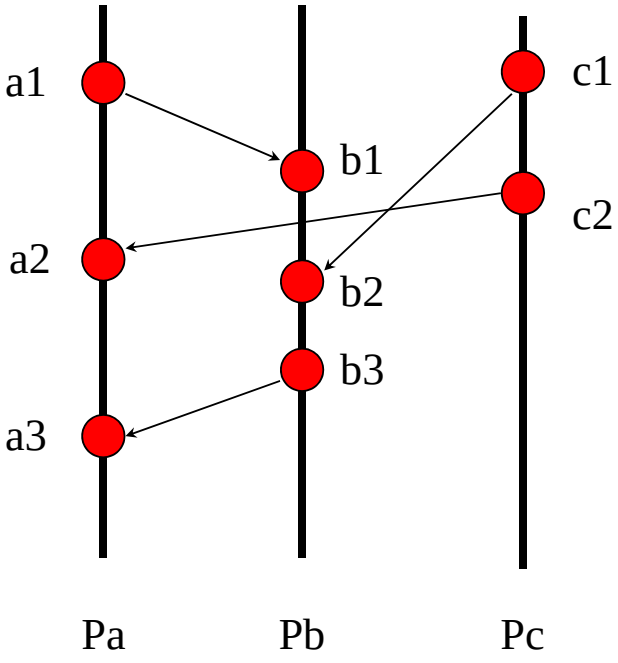
2) Se a è l'evento di invio e b l'evento di ricezione di un messaggio, allora a → b (***ordine in comunicazione***)

3) Se a → b e b → c, allora a → c (***transitività***)

La relazione → introduce un **ordinamento parziale** degli eventi di un sistema e sussiste solo tra alcuni eventi del sistema e non tra tutti (*non è un ordinamento totale*)

Due eventi ***concorrenti*** ↑↑ *se* non a → b *e* non b → a

RELAZIONE HAPPENED-BEFORE →



a1 → a2, a1 → a3
a1 → b1, a1 → b2, a1 → b3

c1 → c2
c1 → b2, c1 → b3, c1 → a3

Eventi concorrenti
a1↑↑ c1, a1↑↑c2, …
a2↑↑ b2, a2 ↑↑ b3, …

RELAZIONE HAPPENED-BEFORE →

La relazione **happened-before** permette di lavorare in un sistema distribuito in cui esista solo → per ordinare

Non esiste un unico orologio globale (**tempo globale**), ma un insieme di clock locali (**tempi locali**)

Si consideri che se lavoriamo in modo asincrono è possibile un qualunque **ritardo di trasmissione** per i messaggi, *variabile* e *maggiore* di ogni intervallo significativo per gli eventi osservabili nel sistema

⇨ Necessità di un **ordinamento (anche) globale o totale** per la sincronizzazione

Vogliamo costruire un sistema di tempo logico basato sulla relazione → ossia su orologi logici e non su orologi fisici

CLOCK LOGICI e TIMESTAMP

Vogliamo costruire un **clock di sistema (sistema di timestamp)** per assegnare un **numero** che ordini gli eventi

*La relazione happened_before è solo **parziale***

La funzione TS(i) di tempo (timestamp) assegna un valore ad ogni evento

Se a → b, allora il timestamp logico degli eventi deve essere tale che TS(a) < TS(b)

Condizione di clock (Logical Clock - LC)

Per a e b, se a → b, allora LC(a) < LC(b)

Si vuole derivare la funzione di clock logico globale **LC** per gli eventi del sistema relativo ai vari processi Pi

NOTA: non è vero che, se LC(a) < LC(b), allora a → b

CLOCK LOGICI e TIMESTAMP

C1. Per ∀ a e b, se a → b nello stesso processo Pi, allora $LC_i(a) < LC_i(b)$

C2. Per ∀a e b, se a è l'invio di un messaggio nel processo P_i e b la ricezione nel processo P_j, allora $LC_i(a) < LC_i(b)$

I1. Ogni processo P_i incrementa LC_i tra due eventi

I2. Per ∀a invio di un messaggio nel processo P_i, il messaggio contiene il clock come timestamp $TS = LC_i(a)$

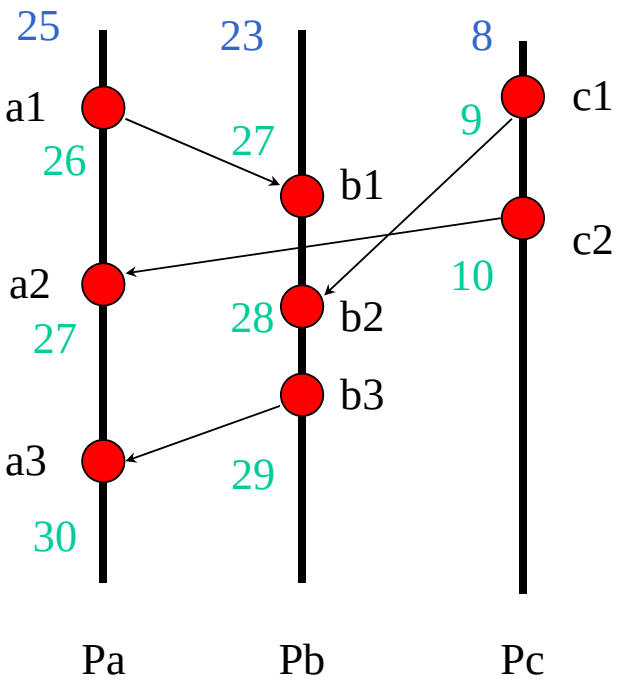
I3. Per ∀b ricezione di un messaggio nel processo P_j, il processo mette il clock logico al valore maggiore del clock corrente e del timestamp

$$LC_j = \max (TS_{ricevuto}, LC_{icorrente}) + 1$$

Le regole introducono una **relazione d'ordine parziale**

Molti eventi concorrenti a ↑↑ b con timestamp uguale

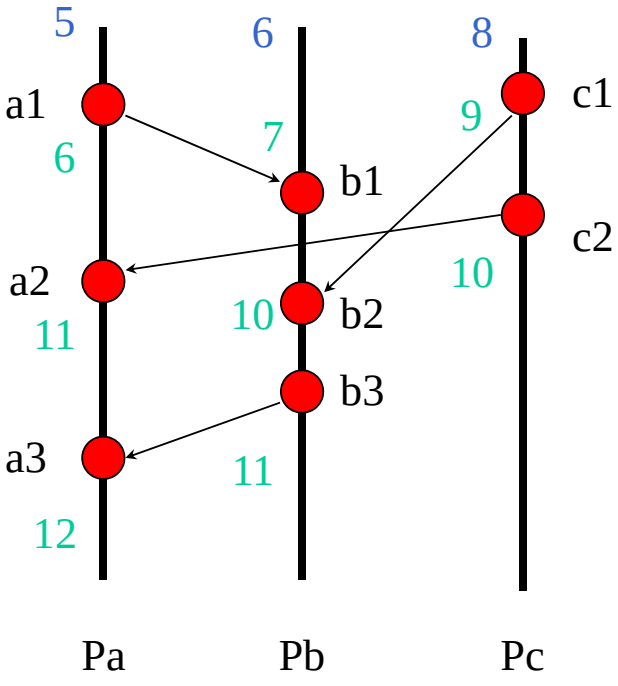
CHI NON RICEVE, NON ADEGUA



Le relazione → permette di ordinare eventi in causa effetto

ma il mittente che trasmette ha un ruolo di iniziativa e adegua di meno il clock logico di chi riceve
(*è il ricevente che deve aggiornare il clock al mittente, se è il caso con una trasmissione*)

HAPPENED-BEFORE → PARZIALE



Le relazione → permette di catturare ordinamento di eventi in causa effetto
ma fa ordinare anche eventi che non lo sono

- Eventi concorrenti nel mondo reale - c1 e b1 che vengono gestiti come in sequenza

Eventi b2 e c2 come sono tra di loro (stesso timestamp)?

ORDINAMENTO TOTALE e ⇨

È necessario introdurre una **relazione di ordine totale** tra tutti gli eventi dei processi del sistema

Le condizioni garantiscono un sistema di clock logici che consente di definire una **relazione di ordine globale** ⇨ tra tutti gli eventi nel sistema, basata sulla parziale →

relazione d'ordine totale ⇨

Se a è un evento in un processo Pi e b un evento in un processo Pj, allora a ⇨ b se e solo se

R1) $LC_i(a) < LC_j(b)$ oppure

R2) $LC_i(a) = LC_j(b)$ e $P_i < P_j$

Ci si può basare su ⇨ per creare della sincronizzazione e stabilire univoci e corretti ordinamenti

ORDINAMENTO TOTALE e REALTÀ

La relazione di Lamport → può anche **perdere di vista** il mondo fisico e le relazioni ragionevoli

In genere **chi riceve messaggi aggiorna il proprio tempo**

Chi non riceve potrebbe avere anche timestamp molto bassi e clock logici non sincronizzati (e che lo favoriscono in ogni sincronizzazione)

Problema del canale nascosto

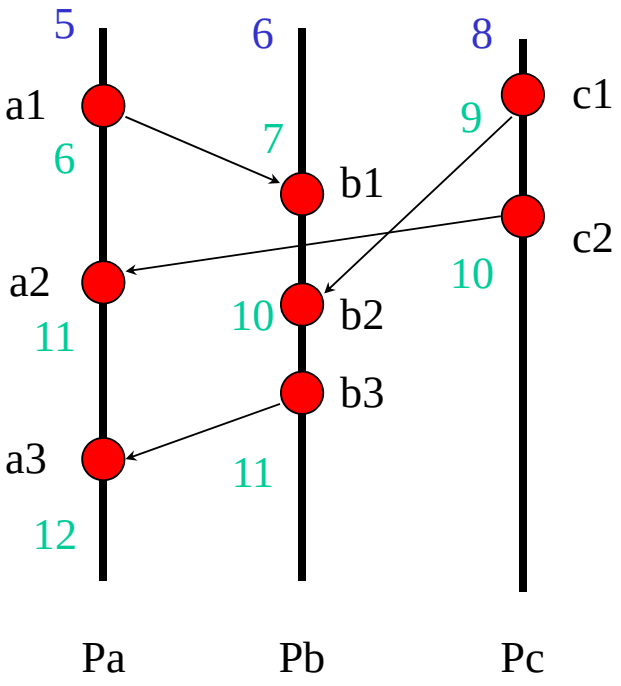
Se un processo usa un canale esterno e non mappato (**hidden channel**) potrebbe portare ad una situazione che non rispetta la relazione causa / effetto

L'**effetto** nel **mondo reale** potrebbe avere un timestamp molto più basso della **causa**

Problema della causalità

Due eventi in relazione potrebbero non essere in rapporto causale

HAPPENED-BEFORE ⇨ TOTALE



Le relazione ⇨ ordina qualunque coppia di eventi

e fa ordinare anche eventi che non sono ordinabili
Eventi concorrenti nel mondo reale - c2 e b2
che vengono gestiti come in sequenza
Prima processo Pb, poi Pc

ORDINAMENTO VECTOR CLOCK

Si possono considerare anche **clock logici vettoriali** o **Vector Clock** per ordinare gli eventi dei processi del sistema

Ogni processo mantiene il proprio timestamp e un vettore $V_i[k]$ di interi di dimensione pari ai processi nel sistema

Un elemento del **vettore dei clock** $V_i[k]$ contiene informazioni su cosa un processo conosce del clock degli altri

Il processo P_i ha nel vettore

1) $V_i[i]$ il proprio timestamp (indice i)

2) $V_i[k]$ il timestamp che conosce per ogni altro processo P_k

La *struttura dati* quindi è più complessa per ogni processo e anche il protocollo da comunicare e seguire per l'aggiornamento

UN PROTOCOLLO VECTOR CLOCK

Il protocollo di aggiornamento dei **Vector clock** prevede

1. Ogni processo P_i incrementa $V_i[i]$ tra due eventi
2. Per ∀invio di un messaggio al processo P_i, il messaggio contiene l'intero vector clock come conoscenza del tempo di P_i dopo avere incrementato il proprio $V_i[i] = V_i[i] + 1$
3. Per ∀ricezione di un messaggio, il processo P_j incrementa il proprio $V_j[j] = V_j[j] + 1$ e aggiorna il proprio vettore secondo

$$V_j[k] = \max (V_j[k], V_i[k])$$

Il ricevente ottiene informazioni sul tempo del processo ricevente e anche sui tempi che questo conosce degli altri

Vettori di clock (anche matriciali) consentono di propagare informazioni e di avere garanzie sulle propagazioni di informazioni

CLOCK LOGICI e VECTOR CLOCK

Il protocollo dei **clock logici**
produce variazioni dei valori del clock in caso di ricezione dei messaggi

In caso di comunicazione ➔ il clock del ricevente si adegua

I successivi eventi sono ordinati con → happened-before

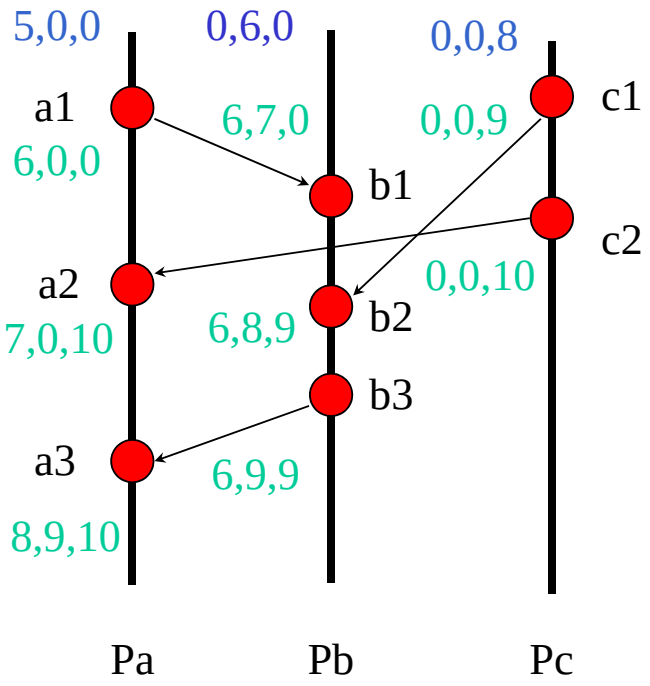
Ma anche eventi non in → possono essere scambiati per tali

Il protocollo dei **vector clock**
la propagazione dell'intero vettore produce adeguamenti dell'intero vector clock del ricevente

In caso di comunicazione ➔ il clock del ricevente si adegua in modo univoco

Eventi in → sono riconosciuti in relazione e eventi non in relazione non sono in relazione neppure per gli algoritmi di vector clock

VECTOR CLOCK



Con i vector clock riusciamo a capire se due eventi sono in reale relazione di causa effetto

Non solo eventi in relazione sono ordinati, ma altri che non sono in relazione non vengono considerati in relazione

Eventi concorrenti nel mondo reale - c1 e b1, a1 e c2,…

Non sono messi in relazione

SINCRONIZZAZIONE

Il caso più **semplice di sincronizzazione** è il caso di un insieme di processi che devono accedere ad **una risorsa in mutua esclusione**

> **GARANZIE** *ogni processo accede alla risorsa per un tempo limitato e la rilascia dopo averla usata*

OBIETTIVI:

Safety - un solo processo alla volta accede alla risorsa

Liveness - ogni processo che ha fatto richiesta riceve l'accesso alla risorsa in un tempo finito

Fairness – richieste diverse devono essere risolte da una politica fair

Ovviamente abbiamo molti modi per realizzarlo

Escludiamo priorità fisse che non sono fair e possono causare starvation

SINCRONIZZAZIONE su RISORSA

Processo coordinatore

Un approccio completamente centralizzato prevede un unico processo coordinatore noto a tutti gli altri (i partecipanti non si devono conoscere tra loro – modello C/S)

Ogni processo che vuole accedere alla risorsa invia al coordinatore la richiesta di accesso e al termine dell'uso notifica il rilascio

Il processo coordinatore decide gli accessi alla risorsa secondo una sua politica per garantire la mutua esclusione

Ovviamente il coordinatore può decidere politiche diverse (gestione FIFO o anche diverse)

Assumiamo che riceva le richieste e che le richieste siano accodate in modo affidabile (ma ritardi qualunque)

COORDINATORE su RISORSA

Protocollo

1) un processo che vuole richiedere la risorsa invia un **messaggio di richiesta** (*request*)

2) Il coordinatore, consultando la propria coda delle richieste e se la risorsa è libera, decide di rispondere ad una richiesta (*reply*) Ovviamente deve mandare **una sola *reply*** ad una sola richiesta per volta

3) al ricevimento del *reply*, il processo usa la risorsa e al termine, la libera mandando un **messaggio di rilascio** al coordinatore (*release*), che decide di fare passare una altra richiesta, ecc.,ecc.

3 messaggi per ogni accesso in sezione critica

SVANTAGGI derivanti dal ruolo centralizzato e unico del coordinatore

Caso di guasto o di non fairness del coordinatore

Ritardo differenziato nell'arrivo al coordinatore

SINCRONIZZAZIONE di LAMPORT

Soluzione decentralizzata e senza punti di singolarità

Un insieme di **N processi** che devono accedere ad una **risorsa singola** in **mutua esclusione** senza alcun ruolo **centralizzato** e cercando di garantire che le richieste siano servite in ordine (in modo fair)

I processi tendono ad esaminare solo la loro coda delle richieste

I processi scambiano messaggi tra di loro per ottenere la sincronizzazione e usano la relazione di Lamport per i clock (fino alla relazione ⇨)

Assunzioni:

- *messaggi da un processo ad un altro devono arrivare in modo FIFO*
- *i messaggi possono essere ritardati ma non persi*
- *la connessione tra i processi è completa e diretta*

SINCRONIZZAZIONE LAMPORT

Uso di clock logici e relazione di Lamport

Ogni processo ha una **coda locale** dei messaggi ricevuti, in cui i messaggi sono accodati in ordine di timestamp

La coda locale contiene inizialmente, per tutti i processi, il messaggio T_0:P_0, inferiore di ogni clock del sistema

Si considera il clock come tempo logico indicato con un intero e con il processo che lo connota

Ogni messaggio ha quindi un timestamp che dipende da entrambi i componenti e che consente l'ordinamento

Un processo che decide di accedere alla risorsa deve eseguire un protocollo di coordinamento globale

Ad ogni processo devono essere noti tutti gli altri e non si ipotizzano guasti
(N processi in ordine di indice e gruppo statico)

PROTOCOLLO Mutua Esclusione

Protocollo

1) Il processo Pi manda il messaggio T_m:P_i ad ogni processo (anche alla propria coda) per segnalare l'intenzione di accedere alla risorsa
2) Alla ricezione del messaggio T_m:P_i il processo P_j (nella propria coda) invia una risposta con il proprio timestamp aggiornato
3) *il processo P_i usa la risorsa se sulla sua coda locale*:

- ha la richiesta T_m:P_i ordinata **prima** di ogni altra (relazione ⇨)
- ha almeno **un** messaggio da **ogni altro processo** con **timestamp superiore** a T_m

4) al rilascio, P_i rimuove il messaggio dalla propria coda e invia un messaggio di rilascio con il proprio timestamp ad ogni processo
5) Ogni processo P_j riceve la richiesta di rilascio e rimuove il messaggio di richiesta dalla propria coda

SINCRONIZZAZIONE su RISORSA

Ogni processo che esegue il protocollo riceve la risorsa in un tempo limitato, che dipende dalla relazione di Lamport, se ogni processo rispetta i vincoli di correttezza

Notiamo che il processo che ha fatto la richiesta attende e forza un coordinamento con ogni altro processo partecipante

Ogni messaggio inviato richiede una risposta da ogni altro

La attesa di questa risposta permette l'arrivo di eventuali altri messaggi di altri che possono precedere questo e che si mettono in coda in ordine di timestamp

Ogni coda del processo richiedente viene ordinata, e quindi ha la stessa giusta conoscenza degli altri

Almeno **(N-1)** messaggi inviati e altrettanti ricevuti

SINCRONIZZAZIONE su RISORSA

Il caso peggiore per la sincronizzazione è il caso di tutti che vogliono entrare contemporaneamente sulla risorsa

In caso di due processi facciano richiesta, i due sono d'accordo sul fatto che entri prima chi ha il timestamp inferiore

non ci possono essere conflitti

L'algoritmo avviene senza *nessuna centralizzazione*, ma in modo completamente distribuito

Per ogni **azione** sulla **sezione critica** il **numero di messaggi scambiato è** (considerando un eventuale broadcast come N-1 messaggi, a meno di costo inferiore)

Numero di messaggi **3 * (N-1)** o N-1 e 2 broadcast

Abbiamo un **costo elevato** dovuto alla decentralizzazione

Ipotesi gravose di **gruppo statico** e **nessun guasto**

ALTRO PROTOCOLLO M.E.

Protocollo Ricart & Agrawala

1) Il processo P_i manda il messaggio T_m:P_i ad ogni processo (anche alla propria coda) per segnalare l'intenzione di accedere alla risorsa
2) Alla ricezione del messaggio T_m:P_i il processo P_j invia

- un reply immediato se non necessita la risorsa o il richiedente ha priorità superiore alla sua
- un reply ritardato se sta usando la risorsa

3) Il processo P_i accede alla risorsa solo se riceve N-1 messaggi di assenso
4) Al rilascio, deve mandare assenso a tutte le richieste arrivate
5) *Le richieste sono cancellate dopo avere dato l'assenso*

Solo un processo può avere N-1 risposte e

solo un processo può accedere alla risorsa

SINCRONIZZAZIONE su RISORSA

Per ogni azione sulla sezione critica il numero di messaggi scambiato (un eventuale broadcast costa come N-1 messaggi)

Numero di messaggi **2 * (N-1)**

Ci sono quindi N-1 messaggi dal richiedente ed N-1 da tutti gli altri

difficile prevedere un coordinamento a costo inferiore

Questi algoritmi basati su **variazioni della relazione** di Lamport sono **completamente distribuiti** (non gestore unico)

fair e *liberi da deadlock* e *da starvation*

hanno **costi** elevati in termini di messaggi scambiati per il coordinamento

hanno un **costo elevato** dovuto alla decentralizzazione

Ipotesi gravose di messaggi non persi e gruppo statico senza guasti

MULTICAST ATOMICO

Una implementazione distribuita del **multicast atomico** può essere meno centralizzata di quella con un unico coordinatore

CATOCS

CAusal & **T**otally **O**rdered **C**ommunication operations **S**upport

basato sul *coordinamento di un insieme di gestori* che decidono sull'ordine delle richieste e le servono coordinandosi

Il gruppo *non* ha un *gestore centrale unico*, ma coordina in modo continuo e crea una visione unica: si può avere un gestore deciso per ogni richiesta che contratta con gli altri e ottiene tutte le risposte per sincronizzarsi con gli altri

Realizzazione non **troppo scalabile** e **implementazione** poco **efficiente (?)** o almeno efficiente solo in casi particolari

*uso di **broadcast a basso livello** per risolvere alcuni problemi applicativi specifici e **migliorare la efficienza** (avendo un supporto che risponde alle ipotesi di non avere messaggi persi, ecc.)*

MULTICAST ATOMICO - ISIS

ISIS

sistema basato su **gruppi** con **replicazione attiva** e con la necessità di una visione con **diversi gradi** di **coordinamento** dei diversi componenti del gruppo

Il sistema ottiene il coordinamento con molte forme distinte di multicast per il gruppo

Si prevedono molte forme di multicast (dette **BCast**)

FBCast (**F**ifo **BCast**)

CBCast (**C**ausal **BCast**)

ABCast (**A**tomic **BCast**) **GBCast** (**G**roup **BCast**)

oltre al caso di nessun garanzia di ordinamento

In genere, non si fanno ipotesi di centralizzazione nel gruppo: per ogni operazione che necessita di un master la decisione su chi esercita il ruolo è dinamica

MULTICAST ATOMICO - ISIS

ISIS ABCast (Atomic BCast)

usa una coda per ogni componente del gruppo e *Lamport*

I messaggi che arrivano ad un qualunque elemento del gruppo sono marcati con il suo ***timestamp iniziale*** e considerati solo se marcati in ***modo finale*** in ordine giusto per la relazione di Lamport

> Ogni messaggio arrivato richiede una fase di coordinamento del gestore **(e hold-back)** per **determinare il timestamp finale**

Il coordinatore riceve il messaggio:

- **lo marca**, e lo **manda agli altri** (con il suo timestamp)
- che attribuiscono **un timestamp** al messaggio in base al proprio tempo e lo rimandano con un loro timestamp
- lo marca come definitivo con il timestamp **più alto** ricevuto (*necessario?*)
- **lo rimanda** a tutti gli altri per comunicare il **timestamp finale**

Problemi: ritardo ed overhead - costo in messaggi 3 * (N-1)

ATOMIC MULTICAST

ISIS ABCast (**A**tomic **BCast**) ordinamento totale per il gruppo

si deve produrre una **visione coerente del gruppo**

Non si richiede una **visione conforme** a **numerazione esterna** necessariamente (che non si deve rispettare)

I componenti del gruppo non possono operare sulle richieste finché non si ha la garanzia che il messaggio:

- è stato **visto anche da tutti gli altri** (arrivo)
- è stato **ordinato rispetto a tutti gli altri messaggi** per i riceventi del gruppo (ordine di arrivo)

Il gruppo è consistente e **l'atomicità garantita**

- *altri messaggi ancora non arrivati, magari anche partiti prima e precedenti, saranno semplicemente trascurati dall'intero gruppo*

E se avessimo dovuto garantire **multicast causale?**

Come faremmo? È più o meno facile?

MULTICAST CAUSALE - ISIS

ISIS CBCast (**C**ausal **BCast**) anche ordinamento parziale

questo multicast tende a considerare solo alcuni eventi che sono etichettati da ordinare; tutti gli altri eventi possono essere ordinati da ogni componente del gruppo in qualunque ordine (limitando così i costi di coordinamento)

ABCast tende a non imporre comportamento (costo) al di fuori del gruppo dei riceventi

CBcast richiede un comportamento corretto al di fuori del gruppo dei riceventi che devono rilevare la relazione **causa effetto** attraverso un rispetto del momento della generazione degli eventi

*il Causal Broadcast richiede un **coordinamento ai mittenti** che devono adeguare il proprio "orologio logico" e fare arrivare la informazione ai riceventi* (le richieste sono accodate al gruppo con i tempi mittenti)
I componenti del gruppo devono solo rispettare l'ordinamento di questi

E se non arrivasse una causa al gruppo prima di processare l'effetto?

Necessità di undo o errore

ALTRO MULTICAST - ISIS

ISIS GBCast (**G**roup **BCast**)

si definiscono gruppi di processi in cui si possono inserire e togliere componenti (creando eventuali inconsistenze)

Per ogni multicast, il messaggio arriva in uno dei due stati:

- a tutti i componenti prima del cambiamento

- a tutti i componenti dopo il cambiamento

ordinamento consistente di tutti gli eventi di BCast o prima o dopo

Il **GBCast** richiede che il **messaggio venga ricevuto solo dopo che lo sono stati tutti i diversi BCast precedenti ancora in atto** (o **prima, in modo consistente**)

Supporto per il monitoring degli eventi di variazione di gruppo (inserimento e di estrazione per guasto)

Uso di una tabella per ogni componente con le appartenenze al gruppo la tabella è aggiornata con un GBCast

JGROUPS - MULTICAST RELIABLE

JGROUPS

Supporto per il multicast affidabile e per il concetto di gruppo

Progettato in Java e con proprietà specificabili dall'utente

Per il livello **trasporto**, sia non connesso, sia connesso, anche con la possibilità di lavorare con JMS (Java Message Service)

Per l'**ordinamento** nel gruppo e la consegna dei messaggi, si ottiene la reliability, intesa come consegna con ritrasmissione dei messaggi, sia varie garanzie di ordinamento: atomico, FIFO, causale, ecc.

Per il **gruppo**, i gruppi sono **dinamici** e viene gestita la appartenenza, l'aggancio e il rilascio al gruppo: ogni elemento del gruppo usufruisce dei messaggi al gruppo, sia dall'esterno, sia intergruppo.

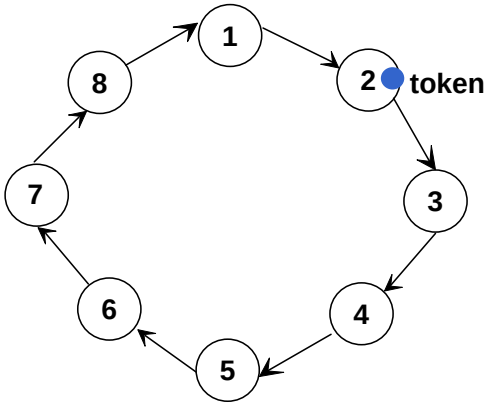
Possibilità di **sicurezz**a, come cifratura e altri protocolli di supporto

SINCRONIZZAZIONE a TOKEN

Per superare il problema del coordinatore centralizzato, la sincronizzazione può avvenire partendo facendo ruotare il ruolo di coordinatore

senza un **ruolo fisso**, ma variando la responsabilità

La sincronizzazione viene **associata** ad un **token** passato dinamicamente tra i diversi N partecipanti



I nodi sono organizzati in un **ring logico**, in cui ogni nodo conosce il successivo

Ogni nodo agisce come gestore quando ha il **token** che deve passare al successivo

Il token circola tra i diversi N partecipanti

SINCRONIZZAZIONE a RING

RING logico che congiunge tutti i partecipanti

Protocollo di accesso alla risorsa: chi riceve il token,

- verifica che sia diretto a lui e
- usa il token per un intervallo di durata massima

(accesso alla risorsa ad intervalli fissi dipendente da **N**)

- lo indirizza al successivo

Un solo processo alla volta accede alla risorsa e non è possibile starvation, se il ring viene percorso in un verso solo

Numero di messaggi **N** per il passaggio completo del token

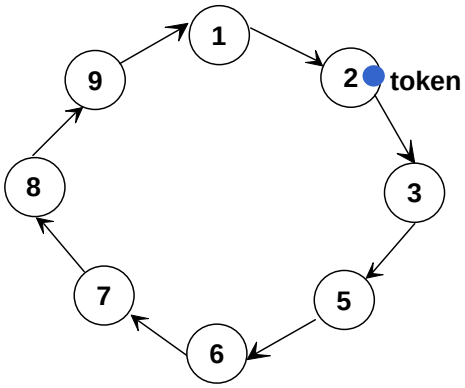
Lo schema di lavoro è tipicamente **proattivo**: il token deve circolare anche senza richieste

Problema se si perde il token (*fallimento del nodo che lo possiede*)

SINCRONIZZAZIONE a RING

Si deve evitare di **non avere token o di averne più di uno**
In caso di *fallimento del nodo che possiede* il token bisogna rigenerarlo

Prevenzione della perdita di token (guasto del nodo gestore)
Ogni nodo che ha inviato il token attiva un intervallo di timeout (dipendentemente da N e dall'intervallo di mantenimento) che viene azzerato al ritorno del token



In caso scatti il **timeout**, il nodo fa partire una procedura di **recovery** per ricreare il token

Si noti che più di un nodo può fare partire la procedura

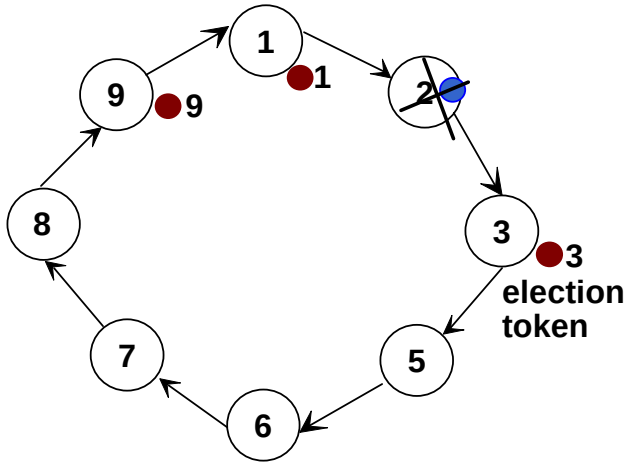
ELEZIONE in RING

Protocollo di elezione per stabilire chi deve diventare gestore
(generare un nuovo token) basato su **priorità statica**

- Al timeout, il processo crea un **token di elezione** (*ET*) con il suo nome ed entra in **stato di elezione** fino al suo ritorno
- Se il processo riceve il normale token prima che l'*ET* generato sia tornato, la elezione è considerata inutile ed è da terminare (***ET distrutto al ritorno***)
- Se un processo riceve un *ET* di un altro, lo registra in una *lista di elezioni* insieme con *l'identità del processo* che l'ha generato, e lo passa nel ring
  se ha già generato un token *ET*, verifica la *priorità statica* e capisce *chi ha* ***priorità o meno*** nella elezione
- Se un processo riceve il suo *ET*, lo rimuove e verifica la lista di registrazione
  Il processo genera il nuovo token, se il nodo stesso è il nodo di ***indice minimo*** (***più prioritario***) nella lista di registrazione

ELEZIONE in un RING

La architettura a RING consente di fare algoritmi di **recovery molto semplici con durata massima** in caso di **guasto singolo**
Ovviamente, ogni nodo deve potere fare delle verifiche locali di correttezza del successivo per saltarlo in caso di suo guasto
E conoscere anche il vicino successivo ulteriore



In questo caso il token potrebbe essere rigenerato dal nodo più prioritario da quelli considerati (qui 1)

Il token di elezione diventa il nuovo token

PROTOCOLLI DI ELEZIONE

Si fanno **protocolli di elezione ogni volta** che si deve trovare un **accordo tra un set di partecipanti senza un ordine predefinito**
Necessari in caso di **fault** e di **recovery** in un **gruppo** per ottenere accordi su una decisione
In molti casi ci si basa su un **accordo statico (ordine) dei partecipanti)**

Algoritmo BULLY

Ogni partecipante P_i che rilevi la necessità di fare una elezione (evento locale a ciascuno) o un recovery

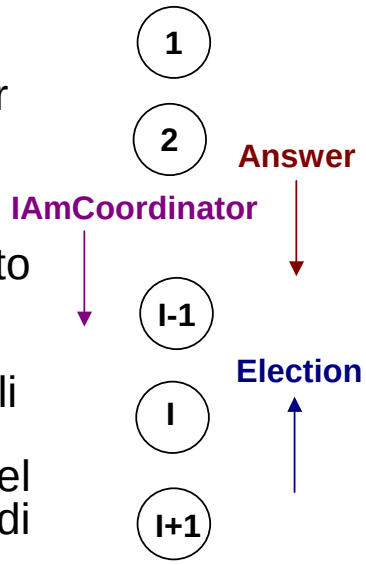
Si considerano tre tipi di messaggi
- elezione **Election**
- risposta **Answer**
- annuncio **IAmCoordinator**

Quante fasi ci sono nei protocolli di elezione?

ELEZIONE BULLY

Ogni **partecipante** può fare partire la **elezione**

1) invia un messaggio di **elezione** ai **processi più prioritari** (**Election**)
1'- in caso di messaggio di elezione da processo meno prioritario, si risponde per bloccarla e si fa **partire una nuova elezione**
2) *dopo un certo tempo*, si aspettano messaggi di coordinamento **Answer** dai nodi superiori
   - se se ne ricevono, ci si ferma
   - se non arriva nessun messaggio da quelli prioritari, diventa il coordinatore e
3) segnala la presenza con l'invio del messaggio **IAmCoordinator** ai nodi di priorità inferiore che vengono avvisati



STATI GLOBALI

In un sistema distribuito si devono a volte considerare e supportare **stati globali associati a tutto il sistema e coordinandone la attuazione nelle singole parti componenti**

checkpoint per recovery, garbage collector distribuito

Assumiamo un **modello asincrono** a scambio di messaggi tra **processi** su nodi diversi (*canali orientati con un solo verso*)

Lo **stato globale** deriva dagli stati di tutti i **processi partecipanti**, ma anche dai **messaggi** scambiati (in atto) tra i diversi processi

Distributed snapshot

Comporre le cose in un unico stato significativo, non potendo garantire una visione globale di tutti gli stati in modo consistente

come fare una composizione dei singoli stati?
componendo stati significativi e potenzialmente consistenti
escludendo stati non significativi (anche non consistenti)

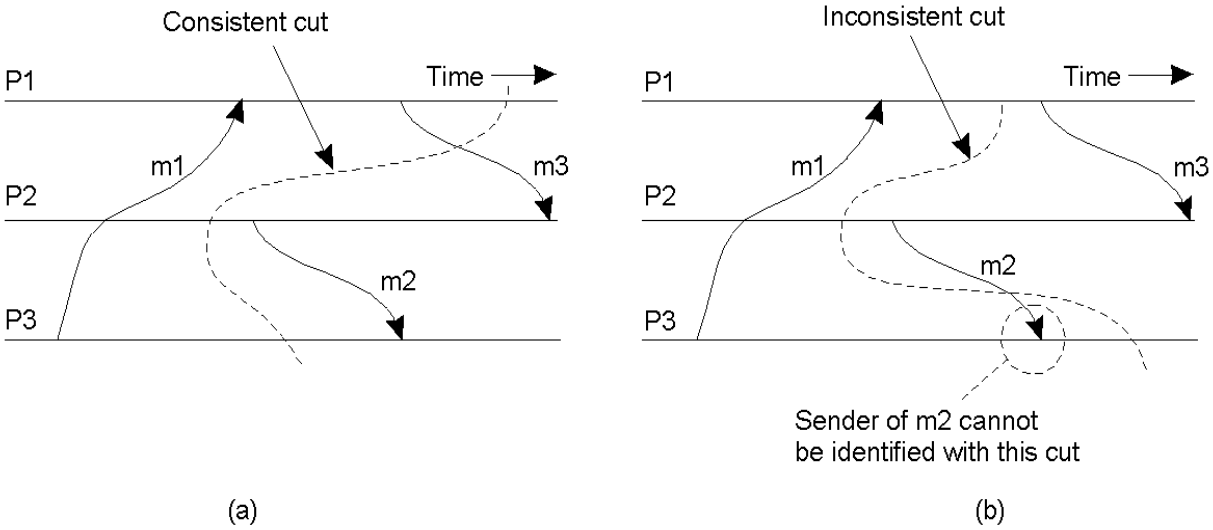
CONSISTENZA dello STATO GLOBALE

Tagli consistenti in un sistema distribuito

Non tutti gli stati di un sistema distribuito sono ammissibili

Tagli consistenti (a) che rappresentano lo stato globale e

Tagli non consistenti (b) che considerano stati parziali non sensati



STATO GLOBALE

I tagli consistenti del sistema distribuito

escludono **situazioni non sensate dal punto di vista operativo**

Taglio o Messaggio consistente

In caso di messaggio di cui si registra la partenza in uno stato di un nodo, e non si registra l'arrivo nello stato del nodo ricevente

È necessario tenere conto del **messaggio nello stato globale** per poterlo ripresentare in caso di replay (cioè **il messaggio è parte dello stato dello snapshot nel processo, parte della coda di ingresso**)

Taglio o Messaggio inconsistente

In caso di messaggio di cui si registra l'arrivo nello stato del nodo ricevente, ma non nello stato del nodo mittente

Questo tipo di registrazione o taglio è **inconsistente**, in quanto incorpora nello stato del ricevente lo stato di un messaggio, senza averlo incorporato nel mittente: in caso di replay il mittente tenderebbe a rimandare il messaggio stesso e a produrre l'effetto di una doppia ricezione nel sistema (si deve evitare la possibilità)

STATI GLOBALI VIA SNAPSHOT

Distributed Global Snapshot (uno alla volta)

Algoritmi locali per mettere insieme un **unico scenario** sensato partendo dagli **stati dei singoli partecipanti** (checkpoint) e dai **messaggi compatibili** in atto scambiati (**stato dei canali**)

OBIETTIVO: fare propagare una **onda di registrazione** di stato da parte dei processi che singolarmente registrano lo stato locale fino a coprire l'intero sistema (ipotesi di **completa raggiungibilità**)

Ogni processo è caratterizzato

- da canali in ingresso e uscita FIFO e connessioni sufficienti
  (*ogni canale bidirezionale → distinto in due canali*)
- da due stati e due **colori** e messaggi di **marker** di gestione stato
  bianco - stato iniziale (*prima dello snapshot*)
  rosso - stato successivo

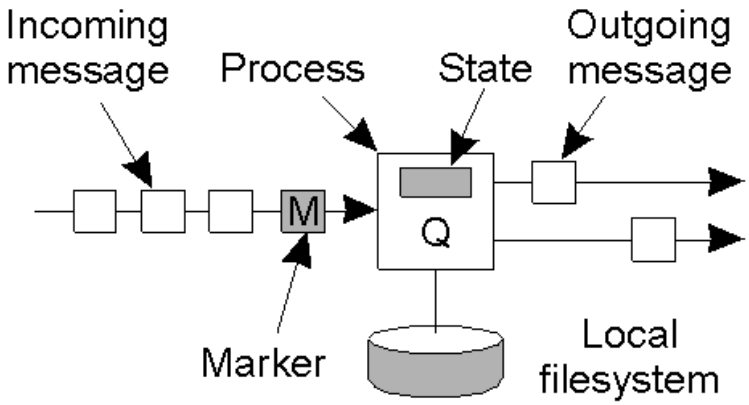
Ogni processo che diventa rosso *fa uno snapshot locale* e *manda un marker* attraverso tutti i suoi canali di uscita
Ogni processo che riceve il marker diventa rosso
Il messaggio marker passa sui canali in ordine insieme con i messaggi applicativi

Distributed Global Snapshot

Ogni processo organizzato su due parti di stato da salvare:
- l'**ambiente locale** (stato) da registrare
- l'insieme di **eventuali messaggi** associati ai ***canali in ingresso*** *da registrare fino al completamento dello stato*

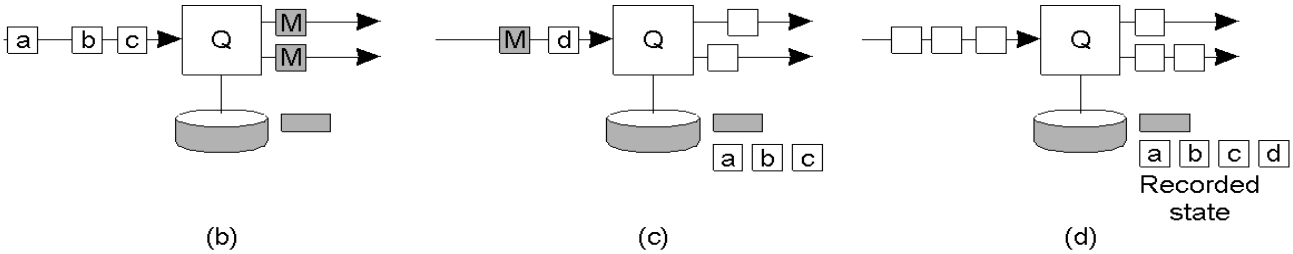
Un processo diventa **rosso** o alla **ricezione di un marker** su un canale di input o se **decide di fare** uno snapshot; dopo rimane rosso
(stabilità dello stato)

Un **processo completa** dopo avere ricevuto un marker su **tutti i suoi canali di input** (e chiude lo snapshot)



(a)

Algoritmo Distributed Global Snapshot



a) Il processo Q riceve un marker e registra lo stato (**checkpoint**)

b) Il processo Q **registra i messaggi** in arrivo dai **canali in input** aperti e manda nuovi marker (eccetto il primo su cui è arrivato il marker). I messaggi sono processati sempre e consumati.

c) Il processo Q riceve un **marker** su uno specifico canale di input (eccetto il primo su cui è arrivato il marker e che è già chiuso)

d) Il processo Q chiude la **registrazione per quel canale (ma i mssaggi continuano a essere serviti**
Ogni processo termina lo snapshotsu tutti i canali, e mantiene lo **stato del nodo** e dei **messaggi salvati dai canali di input**

STATO come insieme di STATI LOCALI

Distributed Global Snapshot
Ogni processo può fare partire lo snapshot (checkpoint dello stato locale) e manda il marker su tutti i suoi canali di out

Per lo **stato dei processi**, si crea una serie di **stati** concorrenti composti dallo **stato locale** più lo stato di **tutti i canali di ingresso**

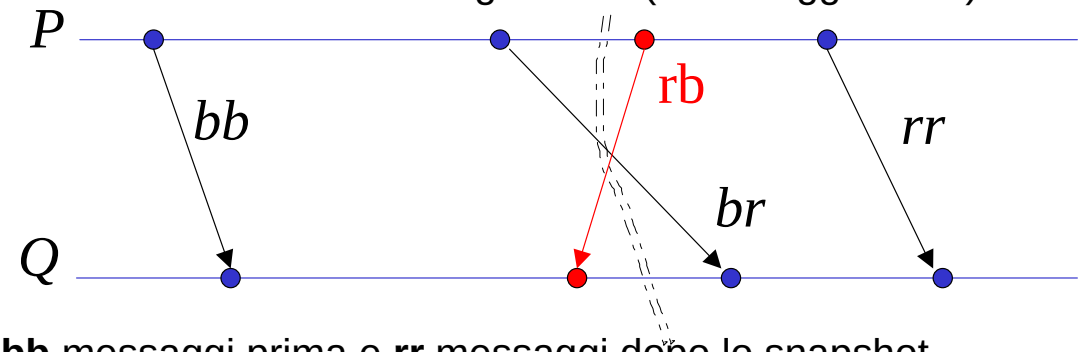
Ogni processo che riceve il marker fa il **checkpoint** del **proprio stato locale** e **manda il marker** a tutti i suoi vicini

Per lo **stato del canale**, si memorizzano tutti i messaggi che arrivano sui **canali di ingresso** fino a che sul canale in input un **marker successivo** segnala che non ci sono altre informazioni significative in ricezione sul canale stesso
La registrazione di quel canale si può chiudere (*checkpoint*)

Lo stato globale dello snapshot risulta composto da:
- gli **stati locali** di tutti i processi
- lo **stato dei canali** di collegamento in ingresso
(i messaggi inviati dai mittenti e *registrati dal ricevente*)

Distributed Global Snapshot

Per lo **stato del canale**, si memorizzano tutti i messaggi che passano al momento dello snapshot
Lo stato globale risulta composto da:
- gli **stati locali** di tutti i processi
- lo **stato dei canali** di collegamento (i messaggi inviati)



- **bb** messaggi prima e **rr** messaggi dopo lo snapshot
- **br** messaggi **da registrare come stato dei canali**
- **rb messaggi non consistenti** (evitati dal procotollo)

Messaggi come rb potrebbero essere considerati due volte nello stato

GESTIONE di SNAPSHOT

Il processo P può cominciare uno snapshot e richiede la collaborazione di tutti gli altri processi che registrano i propri stati di processore e di canale

Come si registra il tutto e dove?

Ogni processo che ha terminato può mandare lo stato al processo che ha iniziato o ad un nodo stabilito P dedicato alla raccolta

Riguardo alla gestione degli snapshot
In prima battuta si pensa a snapshot come eventi rari nel sistema visto il loro costo

Cosa succede se più snapshot attuati insieme?
È possibile fare più snapshot contemporaneamente e distinguerli l'uno dall'altro?