

Le comunicazioni ordinate

Comunicazioni Ordinate

- ▶ E' importante (e utile) definire delle primitive di comunicazione che diano qualche garanzia sull'ordine di consegna dei messaggi inviati all'interno di un gruppo di processi
- ▶ Vedremo le seguenti:
 - ▶ Comunicazione che rispetta l'ordine FIFO di invio dei messaggi
 - ▶ Comunicazione che rispetta l'ordine causale
 - ▶ Comunicazione che rispetta un ordine totale

Comunicazione in un gruppo di processi

Comunicazione di gruppo: gruppo definito di processi. Primitive di gruppo che garantiscono vari tipi di reliability:

- ▶ (1) *Best-effort broadcast*
- ▶ (2) *(Regular) reliable broadcast*
- ▶ (3) *Uniform (reliable) broadcast*

▶ Rivediamo la specifica del (Regular) reliable broadcast, (d'ora in poi solo Reliable broadcast) :

▶ ***Safety.***

- ▶ ***Integrity (No Duplication, No Creation):*** per qualsiasi messaggio m , ogni processo corretto consegna m al più una volta, e solo se m è stato precedentemente inviato in broadcast da un processo mittente

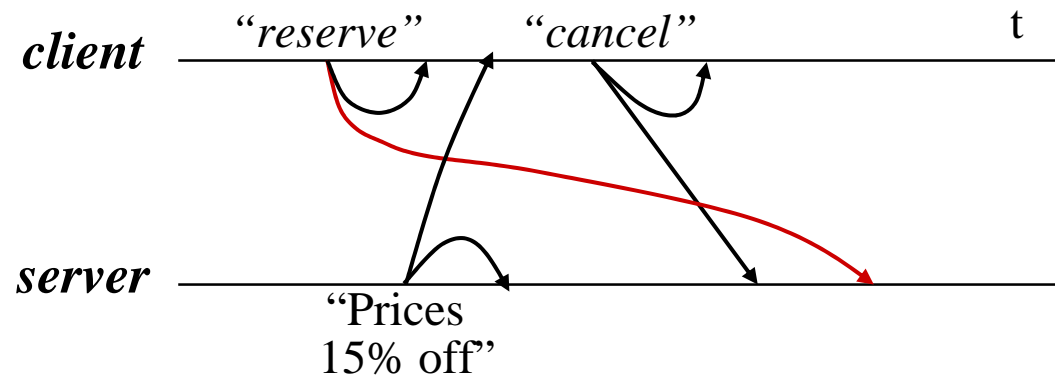
▶ ***Liveness:***

- ▶ ***Validity:*** se un processo corretto invia in broadcast un messaggio m , allora tutti i processi corretti alla fine consegnano m .
- ▶ ***Agreement :*** se un processo corretto consegna un messaggio m , allora tutti i processi corretti alla fine consegnano m .

Utilità della comunicazione ordinata

- ▶ Nel Reliable broadcast non c'è alcun requisito sull'ordine in cui i messaggi sono consegnati
- ▶ Per alcune applicazioni ciò può portare ad "anomalie"...

Esempio: sistema di prenotazione aerea. L'anomalia consiste in una consegna di un msg di cancellazione di una prenotazione che il server non ha ancora registrato!



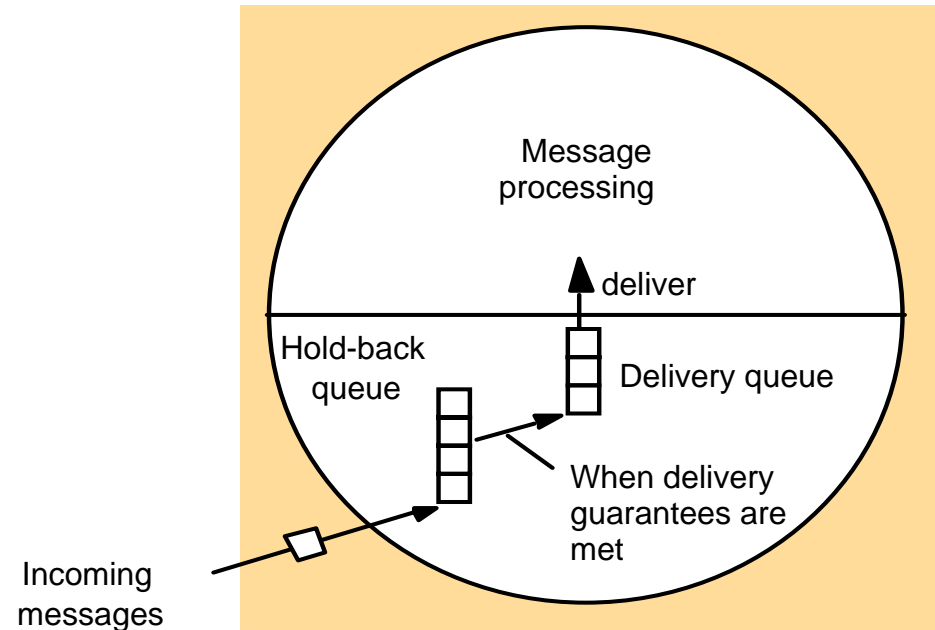
FIFO Broadcast\specifica

- ▶ Qual è la soluzione? Occorre che i messaggi di broadcast inviati dallo stesso mittente vengano consegnati nello stesso ordine in cui i messaggi sono stati inviati.
- ▶ A questo scopo introduciamo una nuova primitiva di comunicazione di gruppo in grado di implementare questa soluzione: FIFO Broadcast
- ▶ La specifica del FIFO broadcast è costituita dalle proprietà viste per il Reliable broadcast alle quali si aggiunge un'altra proprietà di ***Safety*** per catturare la nozione di ordine:
 - ▶ ***FIFO Order***: se un processo invia in broadcast un messaggio m prima di un messaggio m' , allora nessun processo corretto consegna m' a meno che non abbia precedentemente consegnato m .

FIFO Broadcast = Reliable Broadcast + FIFO Order

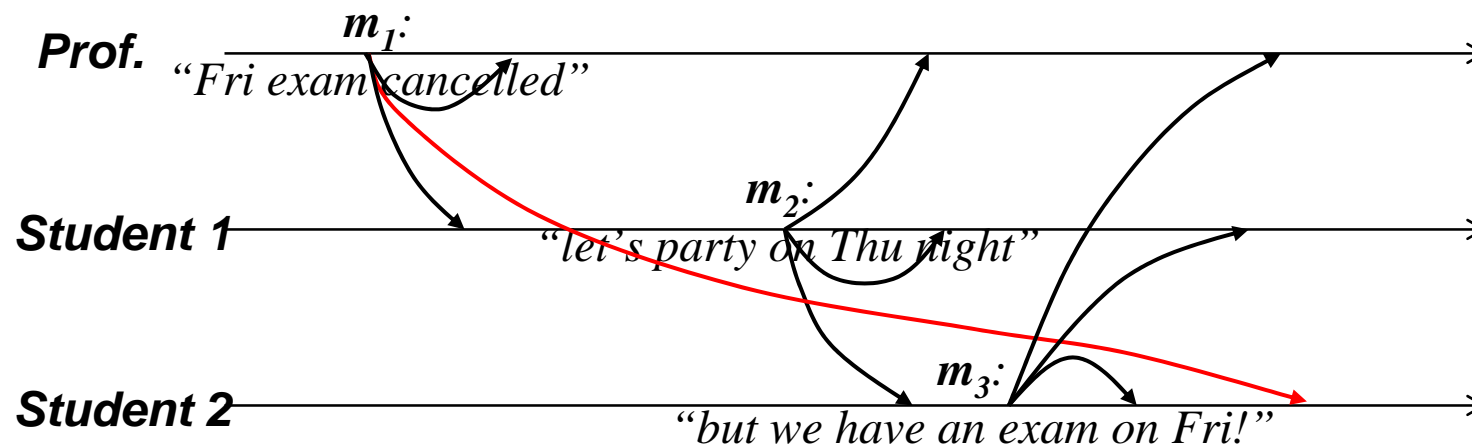
FIFO Broadcast\algoritmo

- Each process p holds:
 - S^p_g a count of messages sent by p to g and
 - R^q_g the sequence number of the latest message to g that p delivered from q
- For p to *FO-multicast* a message to g , it piggybacks S^p_g on the message, *rbBroadcasts* it and increments S^p_g by 1
- On receipt of a message from q with sequence number S , p checks whether $S = R^q_g + 1$. If so, it *FO-delivers* it
- if $S > R^q_g + 1$ then p places message in hold-back queue until intervening messages have been delivered. (note that *rbBroadcast* does eventually deliver messages unless the sender crashes)



Utilità della comunicazione ordinata(2)

- Il FIFO order non preclude tutte le anomalie dovute ad uno strano ordine di consegna...
- Es: Applicazione di tipo newsgroup.



- Anche se il FIFO order è soddisfatto (banalmente), cosa non va? m_2 dipende da m_1 ma Student 2 consegna m_2 prima di m_1
- Qual è la soluzione? Poiché m_1 *precede causalmente* m_2 , allora m_2 non deve essere consegnato finché prima non viene consegnato m_1
- A questo scopo introduciamo una nuova primitiva alle comunicazioni di gruppo in grado di implementare questa soluzione: Causal Broadcast

Causal Broadcast\specifica

- La specifica del Causal broadcast è costituita dalle proprietà viste per il Reliable broadcast alle quali si aggiunge un'altra proprietà di ***Safety*** per catturare la nozione di ordine:
 - Causal Order***: se il broadcast di un messaggio m precede causalmente il broadcast di un messaggio m' , allora nessun processo corretto consegna m' a meno che non abbia precedentemente consegnato m .

Causal Broadcast = Reliable Broadcast+Causal Order

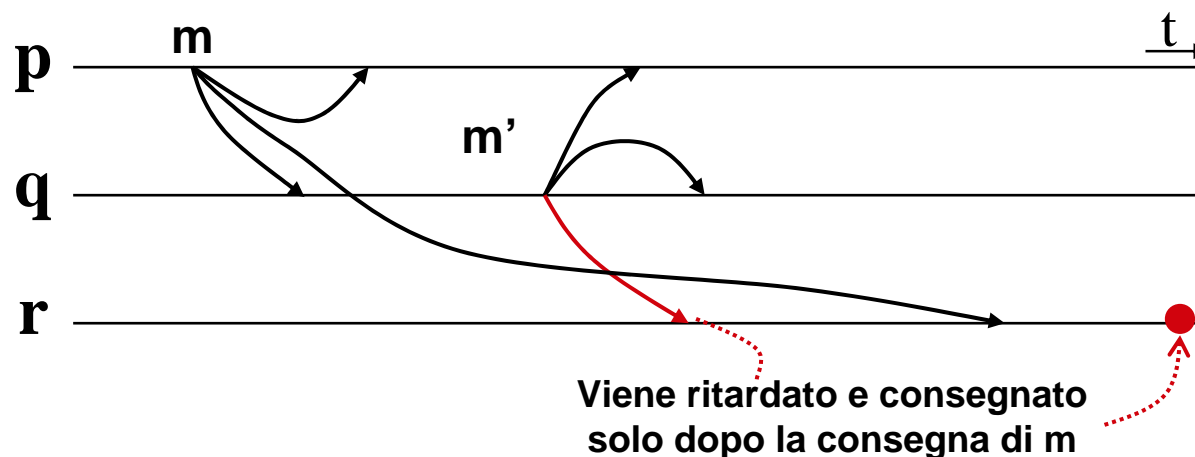
- Causal Order \Rightarrow FIFO Order,
- ma FIFO Order $\not\Rightarrow$ Causal Order
- Quindi, Causal Order = FIFO Order + ?

Causal Broadcast\specifica

▶ **Causal Order = FIFO Order + *Local Order*.**

▶ ***Local Order*:** se un processo consegna un msg **m** prima di inviare in broadcast un msg **m'**, allora nessun processo corretto consegna **m'** a meno che non abbia precedentemente consegnato **m**.

▶ **Esempio:**



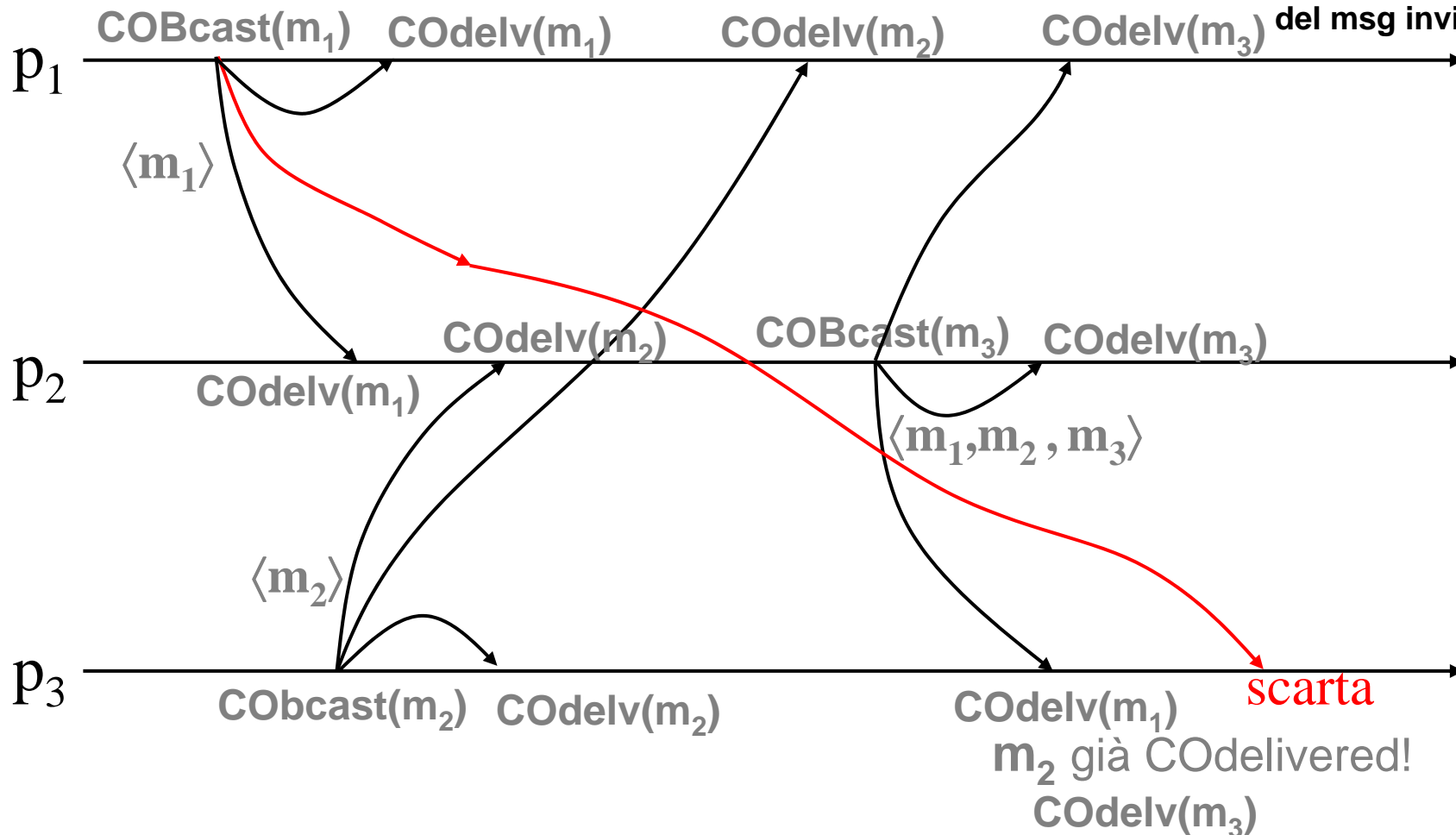
Causal Broadcast\implementazioni

Due implementazioni

Un algoritmo **blocking** che usa **vector clocks**

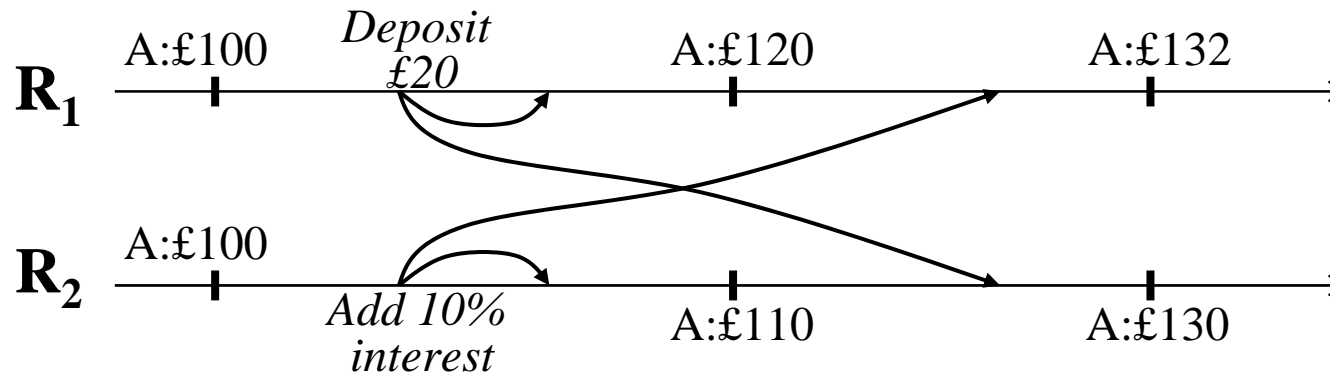
Un algoritmo **non-blocking** che usa il **passato**

IDEA DI BASE:
PIGGYBACKING DEI
MESSAGGI che fanno
parte del PASSATO
del msg inviato



Utilità della comunicazione ordinata(3)

- ▶ Anche il Causal Order non è abbastanza forte per assicurare l'assenza di anomalie
- ▶ Es. Applicazione banking. **Conto bancario replicato su due siti**

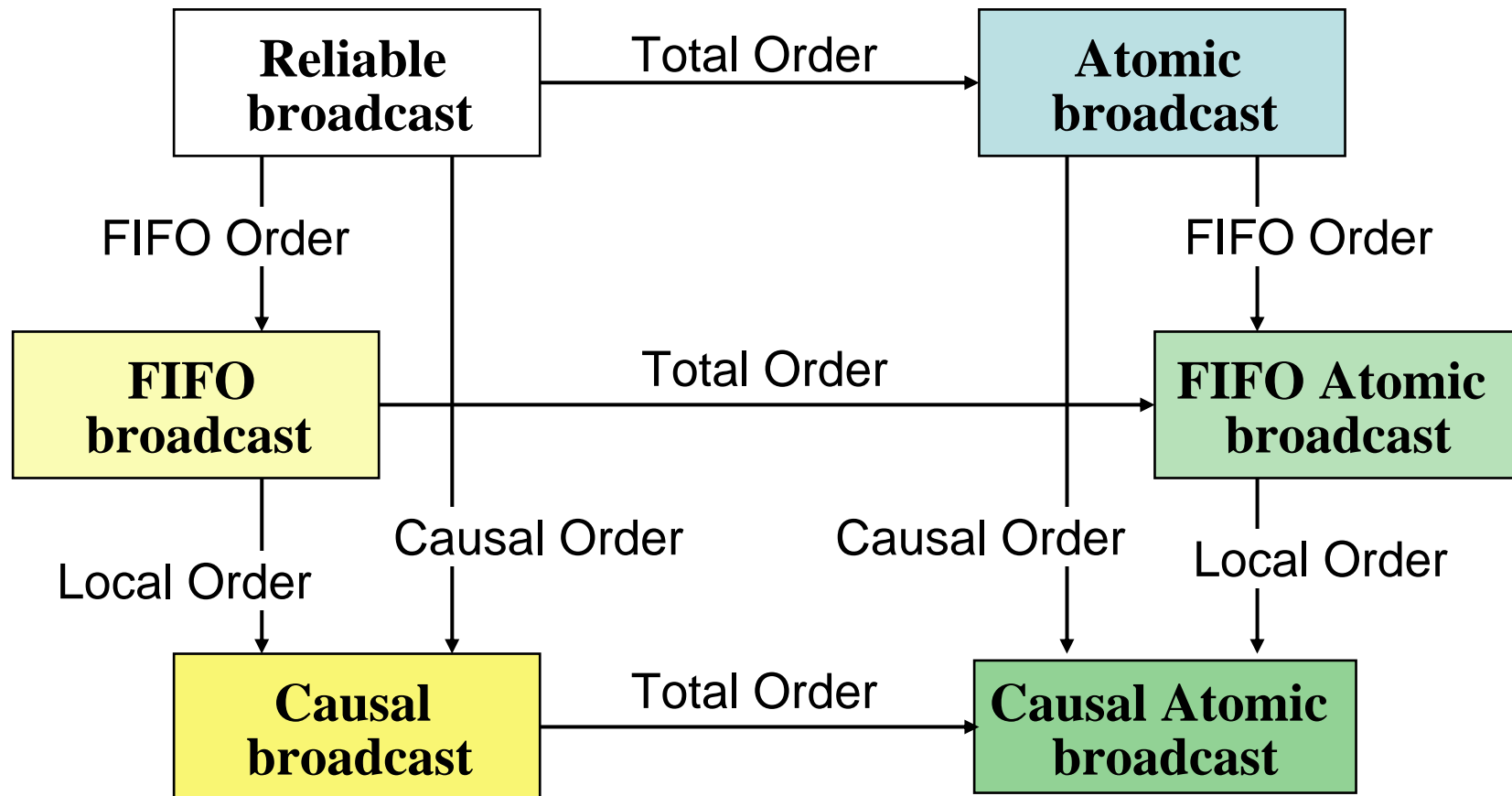


- ▶ Sebbene le repliche siano inizialmente identiche alla fine sono **inconsistenti** anche se il Causal Order è soddisfatto (banalmente)
- ▶ Qual è la soluzione? per garantire che le repliche siano sempre identiche, si deve assicurare che tutti gli update siano consegnati nel **medesimo ordine** anche quando non sono causalmente dipendenti.
- ▶ A questo scopo introduciamo una nuova primitiva comunicazione di gruppo in grado di implementare questa soluzione : *Total Order (Atomic) Broadcast*

Atomic Broadcast\specifica

- ▶ La specifica del Total Order broadcast è costituita dalle proprietà viste per il Reliable broadcast alle quali si aggiunge un'altra proprietà di **Safety** per catturare la nozione di ordine:
 - ▶ **Total Order:** se due processi corretti p e q consegnano entrambi m ed m' , allora p consegna m prima di m' se e solo se q consegna m prima di m'
- ▶ Si noti che la proprietà di total order è una proprietà ortogonale rispetto a FIFO order e causal order. Quindi il total order non è una proprietà più forte rispetto alle altre due. Ad esempio tutti i processi potrebbero consegnare due messaggi in ordine inverso rispetto a quello di invio nel caso in cui il processo inviante non sia corretto.

Gerarchia delle Primitive di Broadcast



Atomic Broadcast e Consenso

- ▶ Si può realizzare il Consenso con l'atomic broadcast
- ▶ Si può realizzare l'atomic broadcast con Consenso e reliable broadcast: il messaggio viene inviato in Reliable Broadcast, i processi riceventi propongono un numero di sequenza per il messaggio (in realtà per tutti quelli in coda non ancora ordinati) facendo partire un Consenso. Alla fine decideranno per la stessa sequenza di consegna per i messaggi.
- ▶ *Quindi si può dimostrare che l'Atomic Broadcast e il Consenso sono problemi equivalenti in un sistema con canali affidabili*
- ▶ Ciò significa che non esiste alcun algoritmo che soddisfa la specifica dell'atomic broadcast in un modello di sistema asincrono con guasti di tipo crash: **FLP per ATOMIC BROADCAST!!**