Indirect Communication

- **indirect communication:** communication between entities in a distributed system via an **intermediary**, with **no direct coupling** between the sender and the receiver/s
- Remote invocation is based on direct coupling between senders and receivers, making systems rigid and difficult to chane
- indirect communication used when change is anticipated: e.g. mobile environments with users coming and going
- disadvantages:
  - performance overhead due to extra indirection
  - more difficult to manage due to lack of space/time coupling

Space and Time Uncoupling

- **space uncoupling:** sender doesn't know the identity of the receiver/s
  - participants can be replaced, updated, replicated, migrated
- **time uncoupling:** sender and receiver don't need to exist at the same time
  - useful in volatile environments where participants come and go
  - implies persistence in communication channel: messages must be stored
  - NB different to asynchronous communication: asynchronous comms don't imply that the receiver has an independent lifetime

	Time-coupled	Time-uncoupled
Space coupling	message passing, remote invocation	
Space uncoupling	IP multicast	Most indirect communication paradigms

Paradigms

- group communication
- publish subscribe
- message queues
- shared memory

	Groups	*Publish-subscribe systems*	*Message queues*	*DSM*	*Tuple spaces*
Space-uncoupled	Yes	Yes	Yes	Yes	Yes
Time-uncoupled	Possible	Possible	Yes	Yes	Yes
Style of service	Communication-based	Communication-based	Communication-based	State-based	State-based
Communication pattern	1-to-many	1-to-many	1-to-1	1-to-many	1-1 or 1-to-many
Main intent	Reliable distributed computing	Information dissemination or EAI; mobile and ubiquitous systems	Information dissemination or EAI; commercial transaction processing	Parallel and distributed computation	Parallel and distributed computation; mobile and ubiquitous systems
Scalability	Limited	Possible	Possible	Limited	Limited
Associative	No	Content-based publish-subscribe only	No	No	Yes

Figure 1: Summary

Group Communication

- **group communication:** communication via group abstraction
  - space uncoupled service: sender doesn't know receivers identities
  - single message sent by sender to a group gets delivered to all group members
  - single multicast send is defining feature c.f. multiple unicast sends
  - management of group membership
  - more effective use of bandwidth with single multicast to multiple receivers (instead of multiple, independent send operations)
  - detection of failures
  - reliability and ordering guarantees: if a process fails half-way through multiple independent send operations to different recipients, system has no way of guaranteeing whether all recipients received the message or not
- provides more than primitive IP multicast, but may be implemented over IP multicast or an overlay network
- important element when building reliable distributed systems

Applications

- **financial:** reliable dissemination of financial information (e.g. stock tickers) to large number of clients
- institutions need accurate, up-to-date access to large number of information sources
- multiuser game
- fault-tolerance: consistent update of replicated data
- system monitoring/management, load balancing

Primitives

- group
- group membership
- join
- leave
- multicast
- broadcast: communication to all processes in the system

Group Model

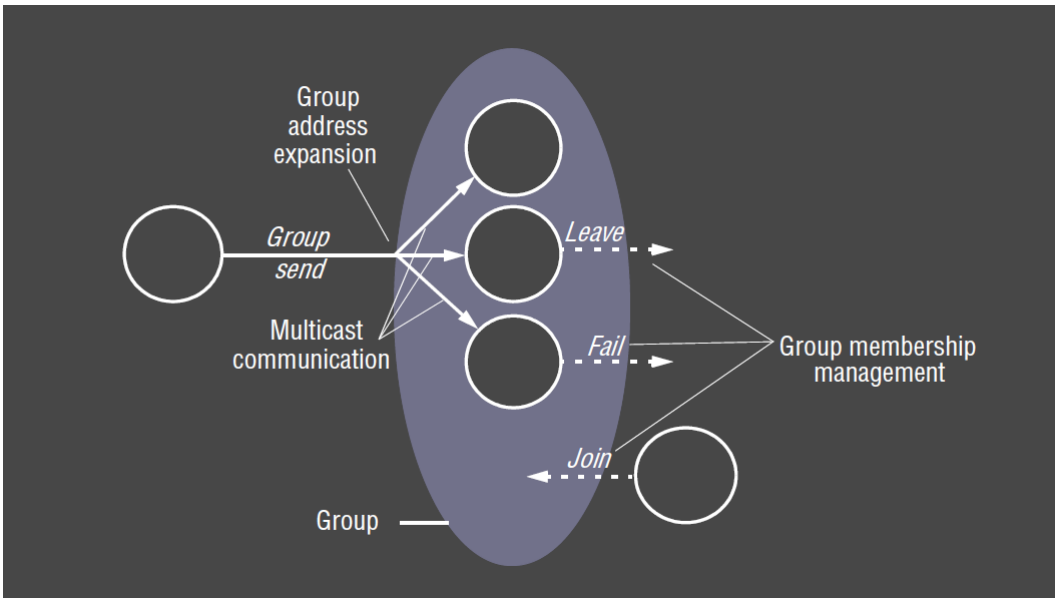


Figure 2: Group Membership

Group Distinctions

These distinctions signficantly impact the underlying multicast algorithms. e.g. some algorithms assume groups are closed

- **process groups:** groups where communicating entities are processes
  - most commonly used, e.g. JGroups
- **object groups:** higher level approach than process groups
  - collection of objects that process the same set of invocations concurrently, each returning responses
- **closed:** only members of the group can multicast to it
- **open:** processes outside the group may send to it
- **overlapping:** entities may be members of multiple groups

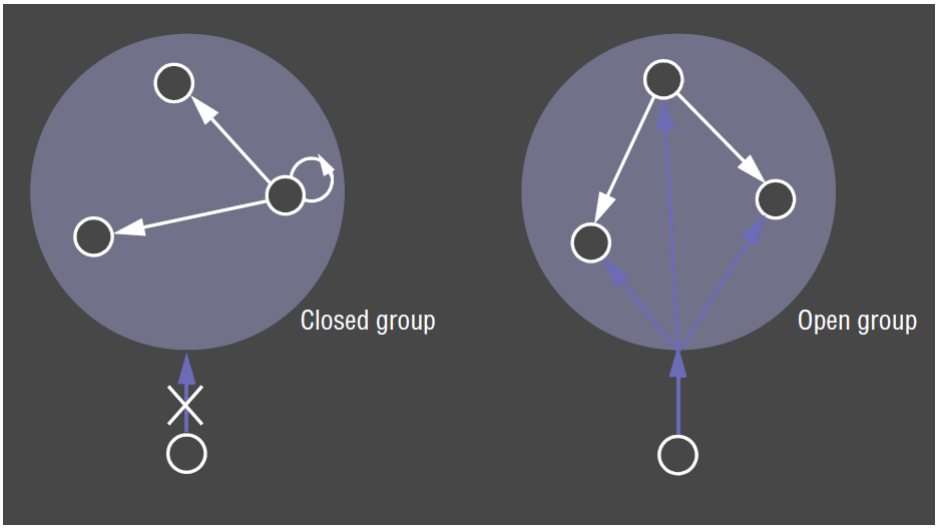


Figure 3: Closed vs Open Groups

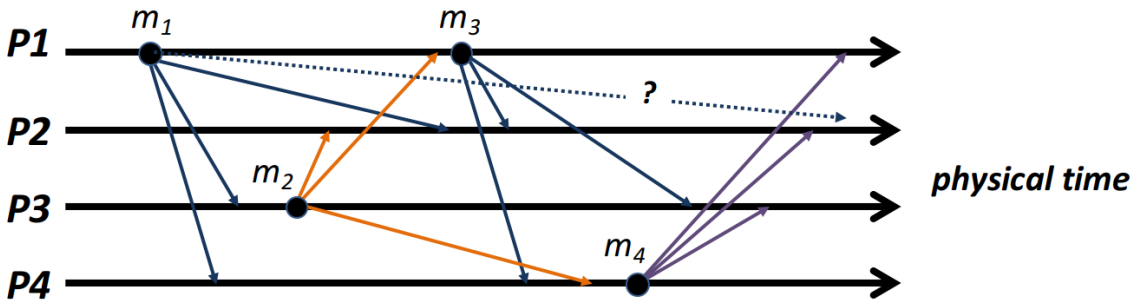
Implementation Issues

Reliability

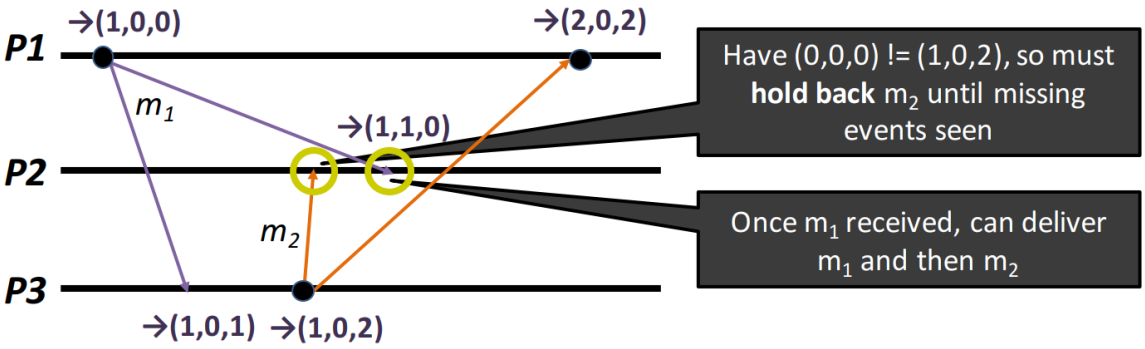
- **reliable multicast:**
  - integrity: deliver message correctly at most once
  - validity: message sent is eventually delivered
  - **agreement:** if the message is delivered to one process, it is delivered to all processes in the group

Ordering

- ordering is not guaranteed by underlying interprocess communication primitives
- Group services offer ordered multicast, which may possess 1+ of the following properties:
- **FIFO ordering:** preserve ordering from sender's perspective
  - if a process sends one message before another, it will be delivered in this order at all processes in the group
- **Causal ordering:** if a message happens before another message, this causal relationship will be preserved in delivery at all processes
- **Total ordering:** if a message is delivered before another message at one process, the same order will be preserved at all processes



FIFO ordering:

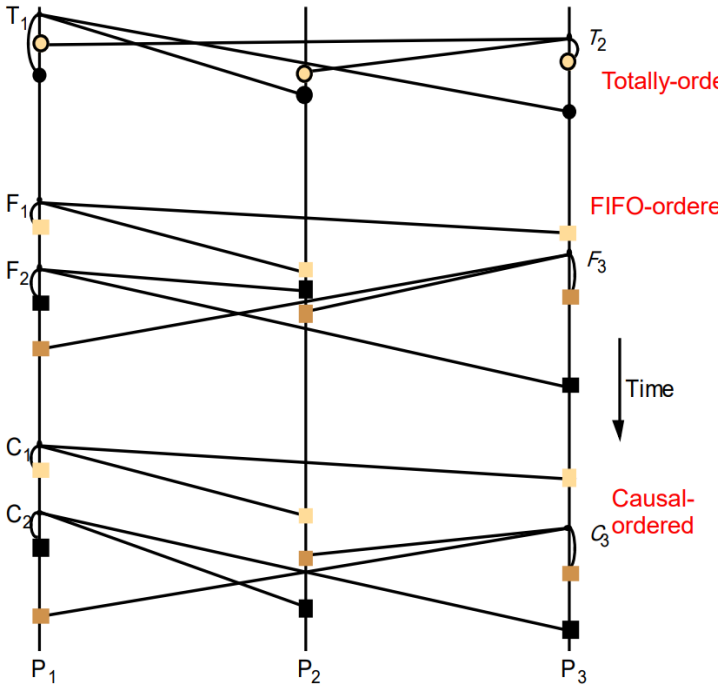


Causal Ordering:

Source

•**Totally ordered** messages T_1 and T_2.
•**FIFO-related** messages F_1 and F_2.
•**Causally-related** messages C_1 and C_3

• Causal ordering implies FIFO ordering
• Total ordering does not imply causal ordering.
• Causal ordering does not imply total ordering.
• Hybrid mode: causal-total ordering, FIFO-total ordering.



Comparison of all 3:

Source

Group membership management

- group members leave and join
- failed members
- notify members of group membership changes
- changes to the group address

Publish-Subscribe

- **publish-subscribe systems:** publisher disseminates events to multiple recipients via an intermediary
  - aka **distributed event-based systems**
  - most widely used paradigm
  - **publishers** publish structure events to an **event service**
  - **subscribers** express interest in events through **subscriptions**, which are arbitrary patterns over the structured events
  - one-to-many: given event eventually delivered to many recipients

Applications

- financial information systems
- live feeds of real-time data
- cooperative working: number of participants notified of events of interest
- ubiquitous computing: management of events from ubiquitous infrastructure (e.g. location events)
- monitoring: e.g. network monitoring
- Google's ad clicks

Dealing Room

- financial information system
- task: allows dealers to see latest market prices of stocks
- market price for a single stock represented by an object
- information providers: processes that collect information arriving in dealing room from a number of external sources
  - each update is an event
  - provider publishes events to pub-sub system for delivery to all dealers subscribed to the corresponding stock
- dealer process subscribes to a named stock
  - it receives notifications and updates the objects representing the stocks
  - update is then displayed to user

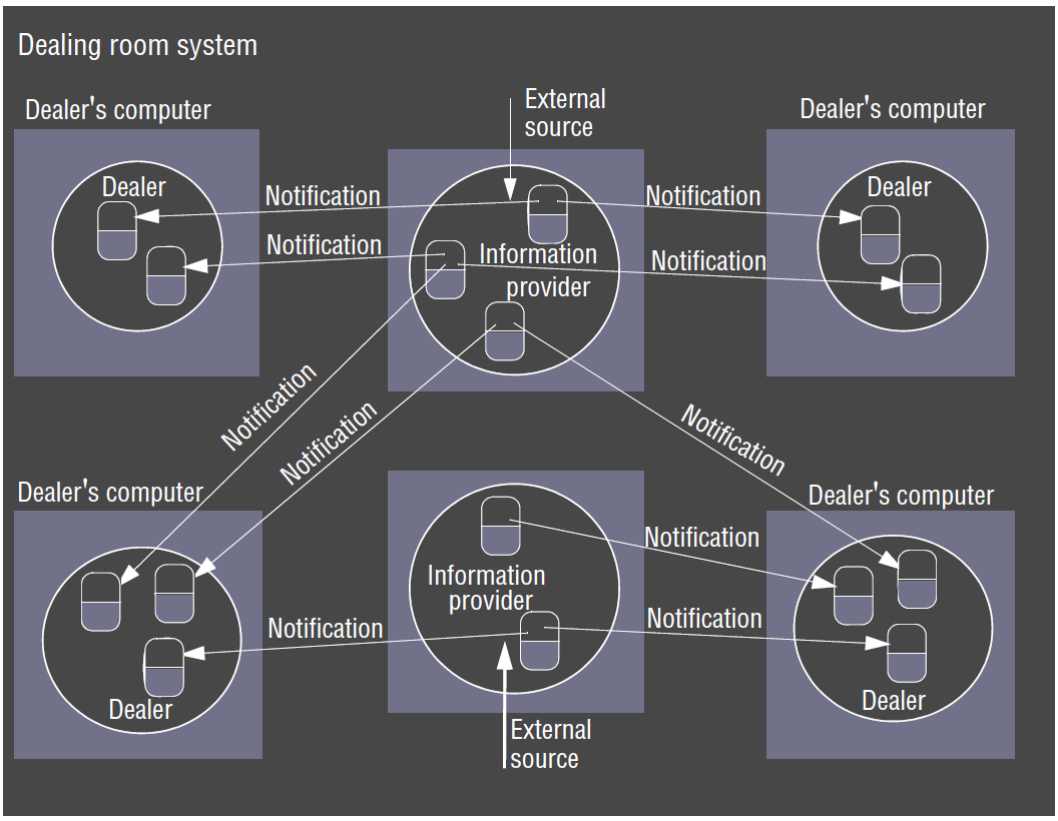


Figure 4: Dealing Room

Events and Notifications

- RMI, RPC support synchronous communication model: client invoking call waits for results to be returned
- events and notifications are associated with **asynchronous communication model**
- event sources can generate different event types
  - attributes contain event information
  - types and attributes are used by subscribers when subscribing to events
  - notifications occur when event types and attributes match that of a subscription

Characteristics

- **heterogeneity:** events allow components that weren't designed for interoperation to work together
  - publisher needs to publish required events

  - subscribers need to subscribe to events of interest
  - interface needs to be provided for receiving/dealing with notifications

- **asynchronous:** communication is asynchronous and event-driven

Model

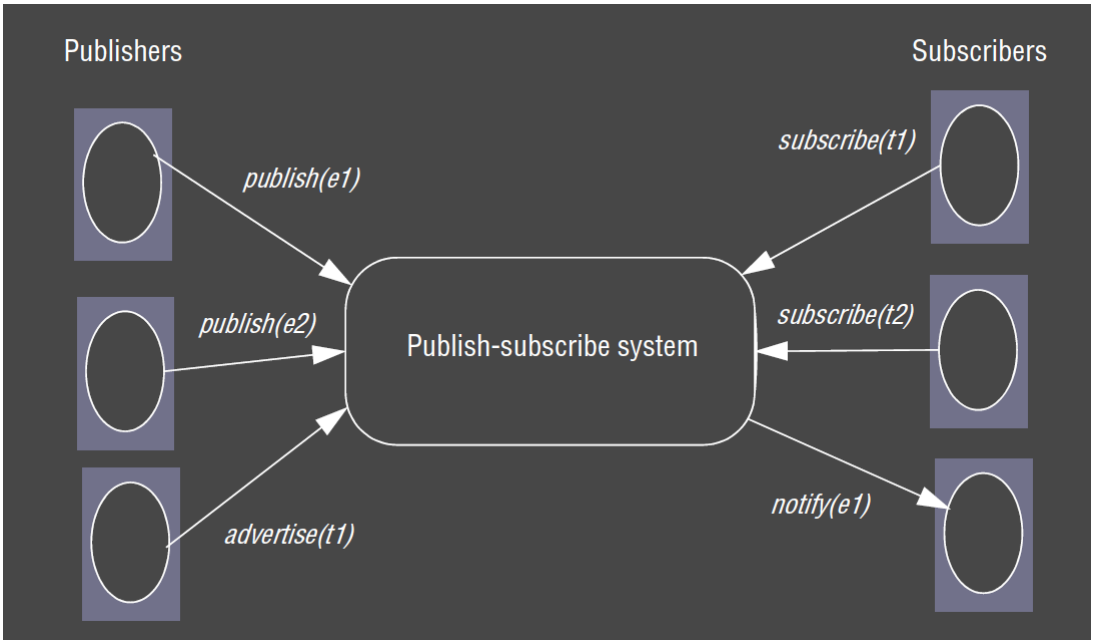


Figure 5: Publish-subscribe

- event `e`
- filter `f`
- `publish(e)`
- `subscribe(f)`
- `unsubscribe(f)`
- `notify(e)`
- `advertise(f)`: publishers can declare the nature of future events in terms of filters
- `unadvertise(f)`

Types

- **channel-based:** publishers publish events to named channels, and subscribers subscribe to one of these channels to receive all events on that channel

  - primitive: only scheme that defines a physical channel
  - more advanced approaches use filtering over event contents

- **topic-based/subject-based:** notification expressed in terms of number of fields, one field denoting the topic
  - subscriptions defined in terms of topics
  - channels are implicitly defined, while topics are explicitly declared
  - permits hierarchical organisation of topics

- **content-based:** generalisation of topic based approach
  - express subscriptions over a particular values for a range of fields in an event notification
  - notifications sent are those matching the attributes specified
  - most flexible

- **type-based:** object-based, with objects having a specific type
  - subscriptions defined in terms of types of events
  - notifications sent are those matching types or subtypes of the given filter
  - similar expressiveness to content-based

Centralised

Architecture

Examples

Message Queue

- **message queue:** messages are placed on a queue, receivers extract messages from the queue

Programming Model

Shared Memory

- **shared memory:** abstraction of global shared memory
  - e.g. distributed shared memory, tuple spaces

Tuple Spaces

York Linda Kernel