10 – Indirect Communication

- Group Communication
- Publish-Subscribe
- Message Queus
- Coulouris 6
-

- Point-to-point communication
  - Participants need to exist at the same time
  - Establish communication
  - Participants need to know address of each other and identities
  - Not a good way to communicate with several participants

- Indirect communication
  - Communication through an intermediary
    - No direct coupling between the sender and the receiver(s
  - Space uncoupling
    - No need to know identity of receiverand viceversa
    - Participants can be replaced, updated, replicated, or migrated
  - Time uncoupling
    - independent lifetimes
    - requires persistence in the communication channel

Indirect Communcation

- scenarios where users connect and disconnect very often
  - Mobile environments, messaging services, forums
- Event dissemination where receivers may be unknown and change often
  - RSS, events feeds in financial services
- Scenarios with very large number of participants
  - Google Ads system, Spotify
- Commonly used in cases when change is anticipated
  - need to provide dependable services

Indirect Communcation

- performance overhead introduced by adding a level of indirection
  - reliable message delivery, ordering affect scalability
- more difficult to manage because lack of direct coupling
- difficult to achieve end-to-end properties
  - real time behavior
  - security

Indirect communcation

- “All problems in computer science can be solved by another level of indirection.”
- Indirect communication
  - communication between entities in a DS through an intermediary with no direct coupling between sender and receiver(s).
- Lots of variations in
  - Intermediary
  - Coupling
  - Implementation details and tradeoffs therein
- “There is no performance problem that cannot be solved by eliminating a level of indirection.”

Indirect communication

	Time-coupled	*Time-uncoupled*
Space coupling	*Properties*: Communication directed towards a given receiver or receivers; receiver(s) must exist at that moment in time *Examples*: Message passing, remote invocation (see Chapters 4 and 5)	*Properties*: Communication directed towards a given receiver or receivers; sender(s) and receiver(s) can have independent lifetimes *Examples*: See Exercise 15.3
Space uncoupling	*Properties*: Sender does not need to know the identity of the receiver(s); receiver(s) must exist at that moment in time *Examples*: IP multicast (see Chapter 4)	*Properties*: Sender does not need to know the identity of the receiver(s); sender(s) and receiver(s) can have independent lifetimes *Examples*: Most indirect communication paradigms covered in this chapter

Group communication

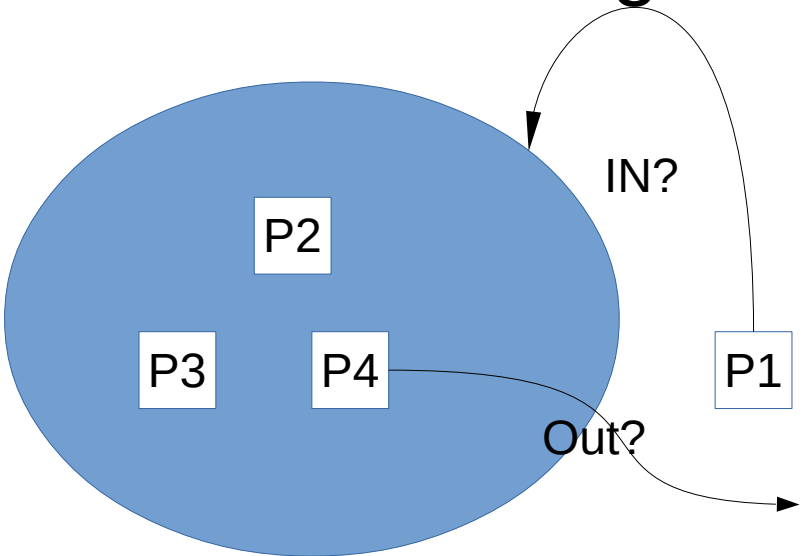
- Group communication offers a service whereby a message is sent to a group and then this message is delivered to all members of the group.
- Characteristics
  - Sender is not aware of the identities of the receivers
  - Represents an abstraction over multicast communication
- Possible implementation over IP multicast (or an equivalent overlay network), adding value in terms of
  - Managing group membership
  - Detecting failures and providing reliability and ordering guarantees

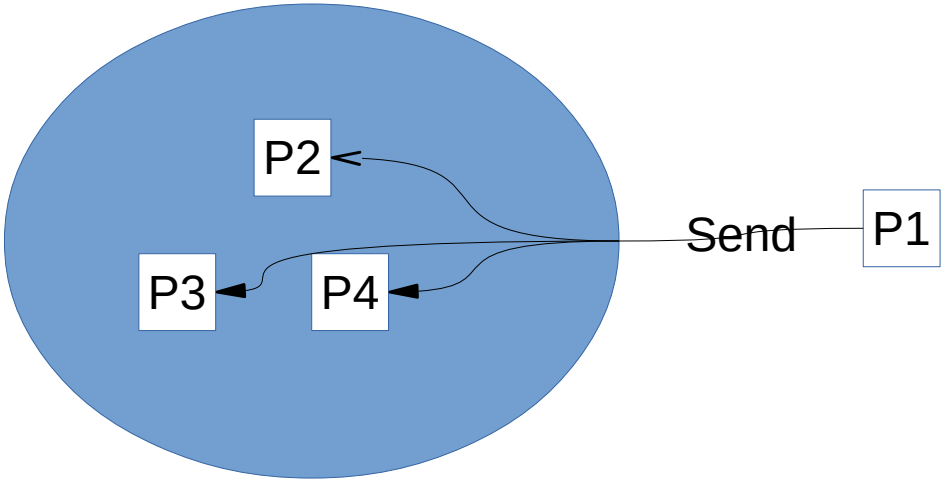
Group communication

- Reliable dissemination of information to potentially large numbers of clients,
  - financial industry, where institutions require accurate and up-to-date access to a wide variety of information sources
- Support for collaborative applications
  - where events must be disseminated to multiple users to preserve a common user view –
  - for example, in multiuser games
- Support for a range of fault-tolerance strategies
  - including the consistent update of replicated data
  - or the implementation of highly available (replicated) servers
- Support for system monitoring and management
  - including for example load balancing strategies

Group Communciation

- Central abstraction:
  - group & associated membership
- Processes join (explicitly) or leave (explicitly or by failure)
- Send single message to the group of N, not N unicast messages



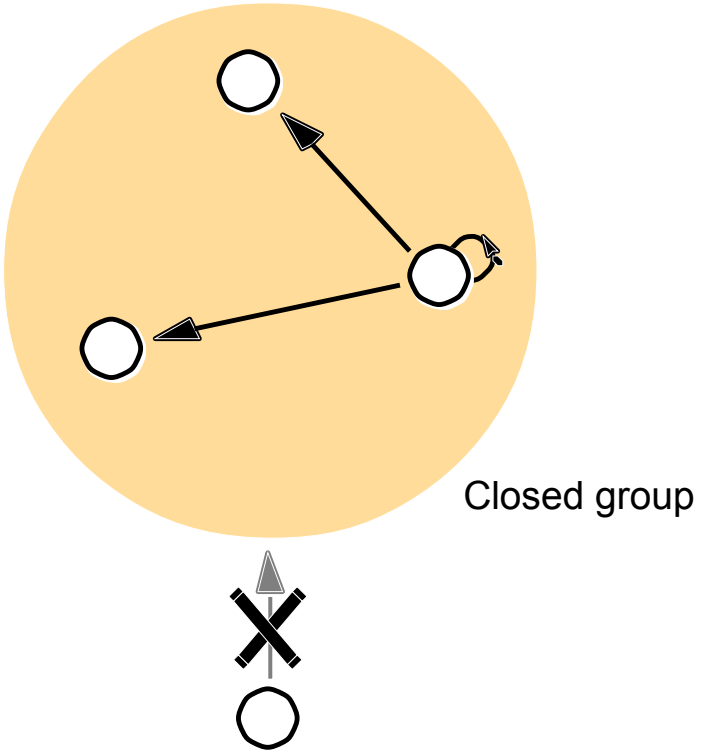


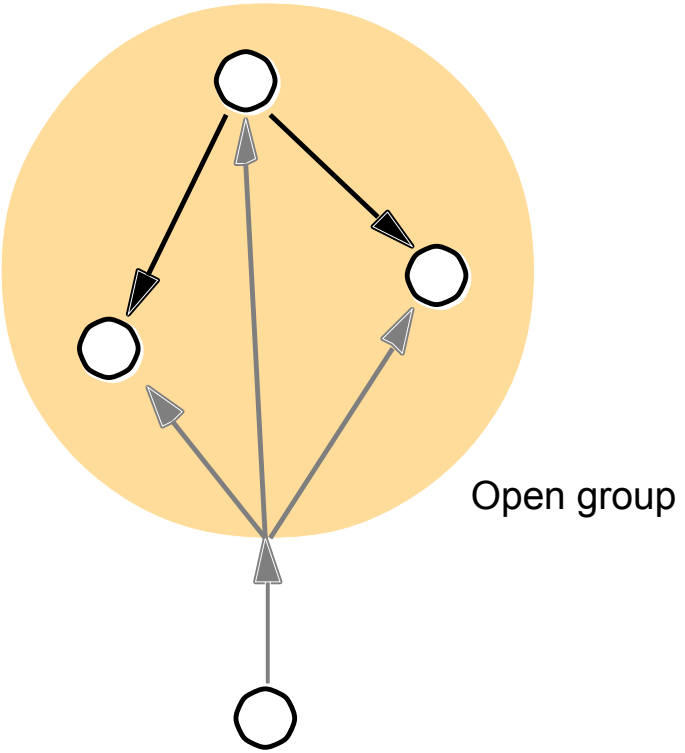
Groups

- Process groups and object groups
  - Most research on process groups
  - Abstraction: resilient process
  - Messages delivered to a process endpoint, no higher
  - Messages typically unstructured byte arrays, no marshalling etc
  - Level of service ≈ socket
- Object group
  - higher level approach
  - Collection of objects (same class!) process same invocations
  - Replication can be transparent to clients
  - Invoke on single object (proxy)
  - Requests sent by group communication
  - Voting in proxy usually
- Process groups still more widely researched & deployed

Groups

- Closed
  - Cooperating servers
  - Internal messages
- Open
  - Notification of services





Implementation Issues

- Reliable delivery
- Unicast delivery reliability properties
  - Delivery integrity
    - message received same as sent, never delivered twice
  - Delivery validity
    - outgoing message eventually delivered
- Group communication reliability properties build on this
  - Delivery integrity
    - Deliver message correctly at most once to group members
    - Note: stronger than RPC delivery guarantees!
  - Delivery validity
    - message sent will be eventually delivered (if not all group members fail)
  - Agreement/consensus
    - Delivered to all or none of the group members
    - Note: also called atomic delivery

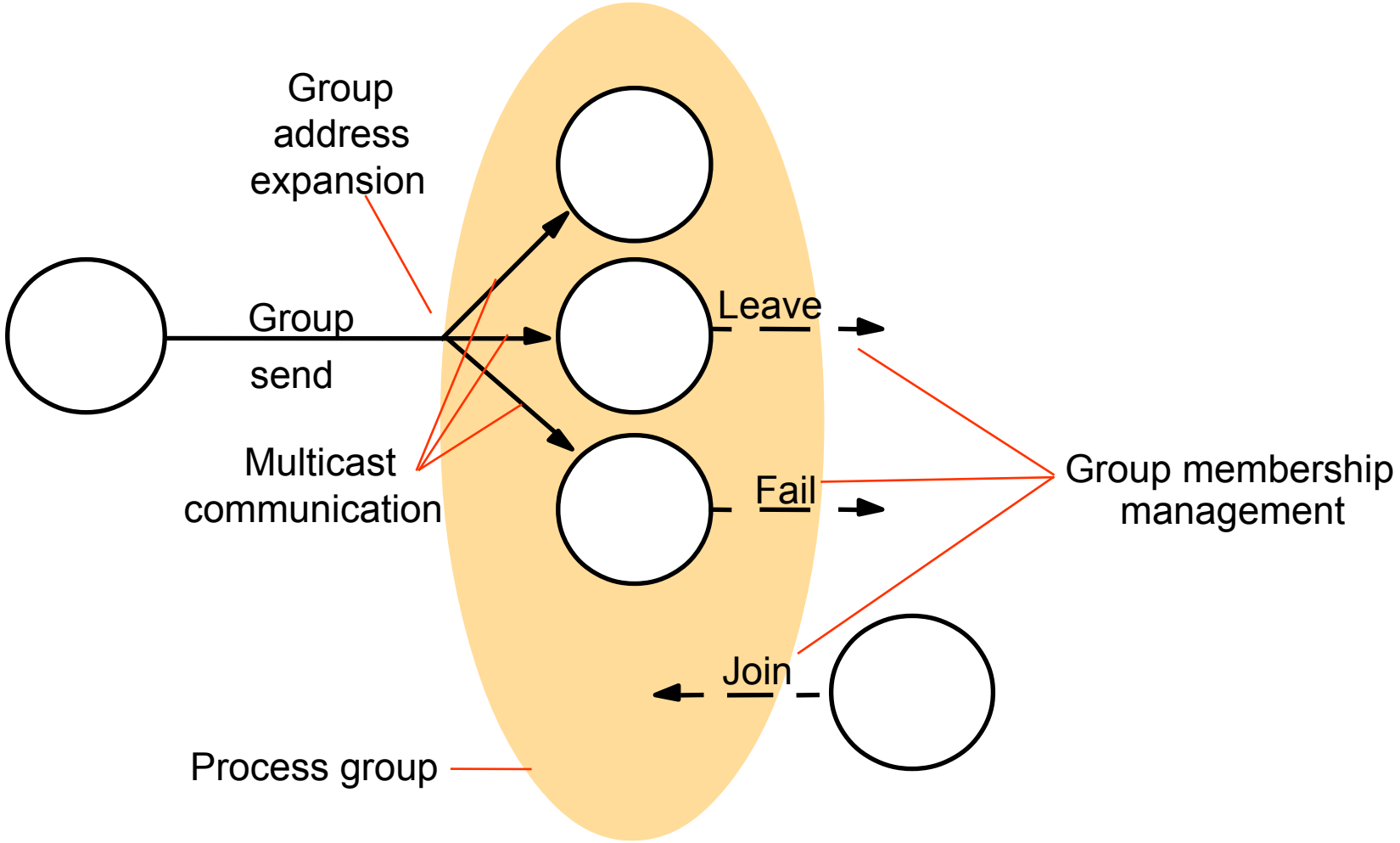
Ordering

- FIFO ordering
  - first-in-first-out from a single sender to the group
- Causal ordering
  - preserves potential causality, happens before
- Total ordering
  - messages delivered in same order to all processes
- Perspective
  - Strong reliability and ordering is expensive: scale limited
  - More probabilistic approaches & weaker delivery possible

Groups membership

- Providing an interface for group membership changes
  - The membership service provides operations to create and destroy process groups and to add or withdraw a process to or from a group
- Failure detection
  - The service monitors the group members not only in case they should crash, but also in case they should become unreachable because of a communication failure.
- Notifying members of group membership changes
  - The service notifies the group’s members when a process is added, or when a process is excluded
- Performing group address expansion
  - When a process multicasts a message, it supplies the group identifier rather than a list of processes in the group.

Groups membership

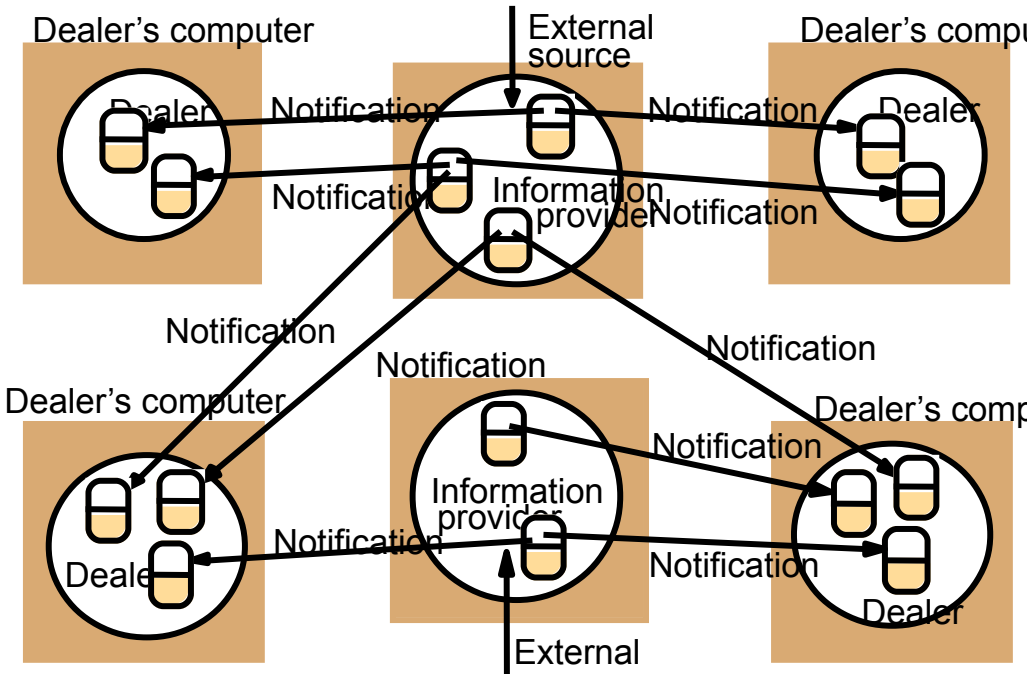


Publish-subscribe

- Pub-sub or distributed event systems
  - Most widely used from this chapter
- Publishers publish structured events to event service (ES)
- Subscribers express interest in particular events
- ES matches published events to subscriptions
- Applications
  - Financial info systems
  - Other live feeds of real-time data (including RSS)
  - Cooperative working (events of shared interest)
  - Ubiquitous computing (location events, ... from infrastructure)
  - Lots of monitoring applications, including internet net.

Example

- Stock trading system
- Let users see latest market prices of stock they care about
- Info for a given stock arrives from multiple sources
- Dealers only care about stocks they own (or might)
- May only care to know above some threshold, in addition
- Two kinds of tasks
  - Info provider receives updates (events) from a single external source
  - Dealer process creates subscription for each stock its user(s) express interest in

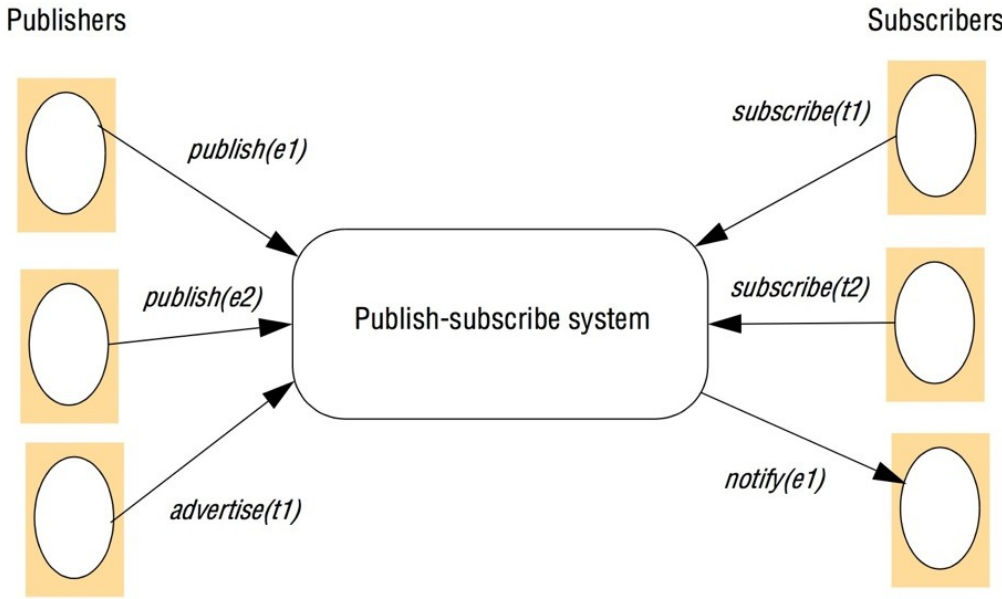


Characteristics

- Heterogeneity
  - Able to glue together systems not designed to work together,
  - Have to come up with an external description of what can be subscribed to: simple flat, rich taxonomy, etc
- Asynchrony
  - Decoupling means you never have to block!
- Possible delivery guarantees
  - All subscribers receive the same events (atomicity)
  - Events correctly delivered to subscribers at most once to subscribers (integrity)
  - message sent will be eventually delivered (validity)
  - Real-time

Programming Model

- Publishers
  - Disseminate event **e** through **publish(e)**
  - Register/advertise via a filter (pattern over all events):
    - **f: advertise (f)**
    - Expressiveness of pattern is the subscription model
  - Can also remove the offer to publish: **unadvertise (f)**
- Subscribers
  - Subscribe via a filter (pattern)
    - f:subscribe(f)
  - Receive event e matching
    - f: notify(f)
  - Cancel their subscription:
    - unsubscribe(f)



Subscription model

- Channel-based
  - Publishers publish to named channels
  - Subscribers get ALL events from channel
  - Very simplistic, no filtering (all other models below do)
  - CORBA Event Services uses this
- Topic-based (AKA subject-based)
  - Each notification expressed in multiple fields, one being topic
  - Subscriptions choose topics
  - Hierarchical topics can help (e.g., old USENET rec.sports.cricket)

Subscription model

- Content-based
  - Generalization of topic based
  - Subscription is expression over range of fields (constraints on values)
  - Far more expressive than channel-based or topic-based
- Type-based
  - Use object-based approaches with object types
  - Subscriptions defined in terms of types of events
  - Matching in terms of types or subtypes of filter
  - Ranges from coarse grained (type names) to fine grained (attributes and methods of object)
  - Advantage: clean integration with object-based programming languages

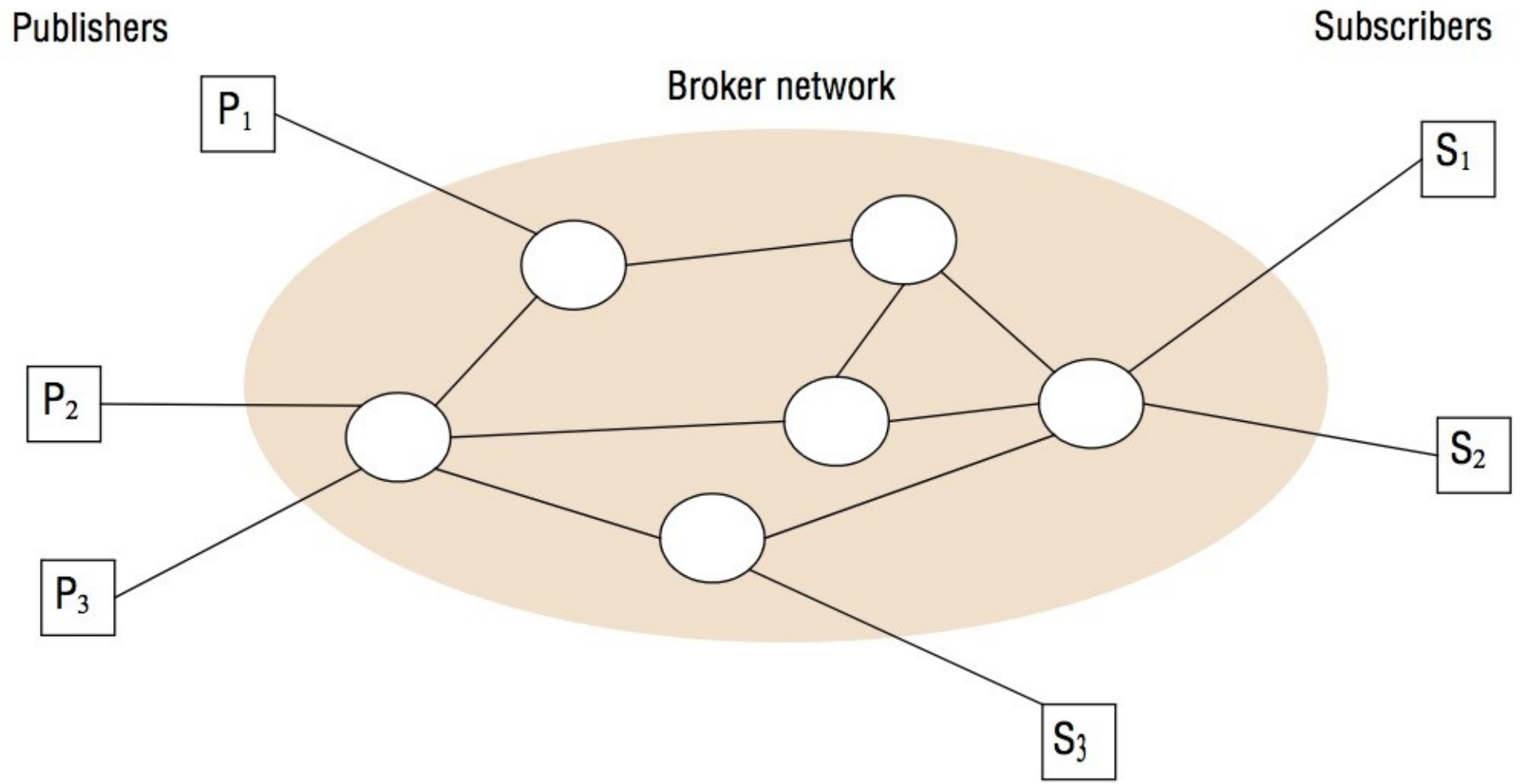
Main concern

- Deliver events efficiently to all subscribers that have filters that match the events
- Security
- Scalability
- Failure handling
- Quality of Service (QoS)
- Tradeoffs:
  - Latency/reliability
  - Ease in implementation / expressive power to specify events of interest

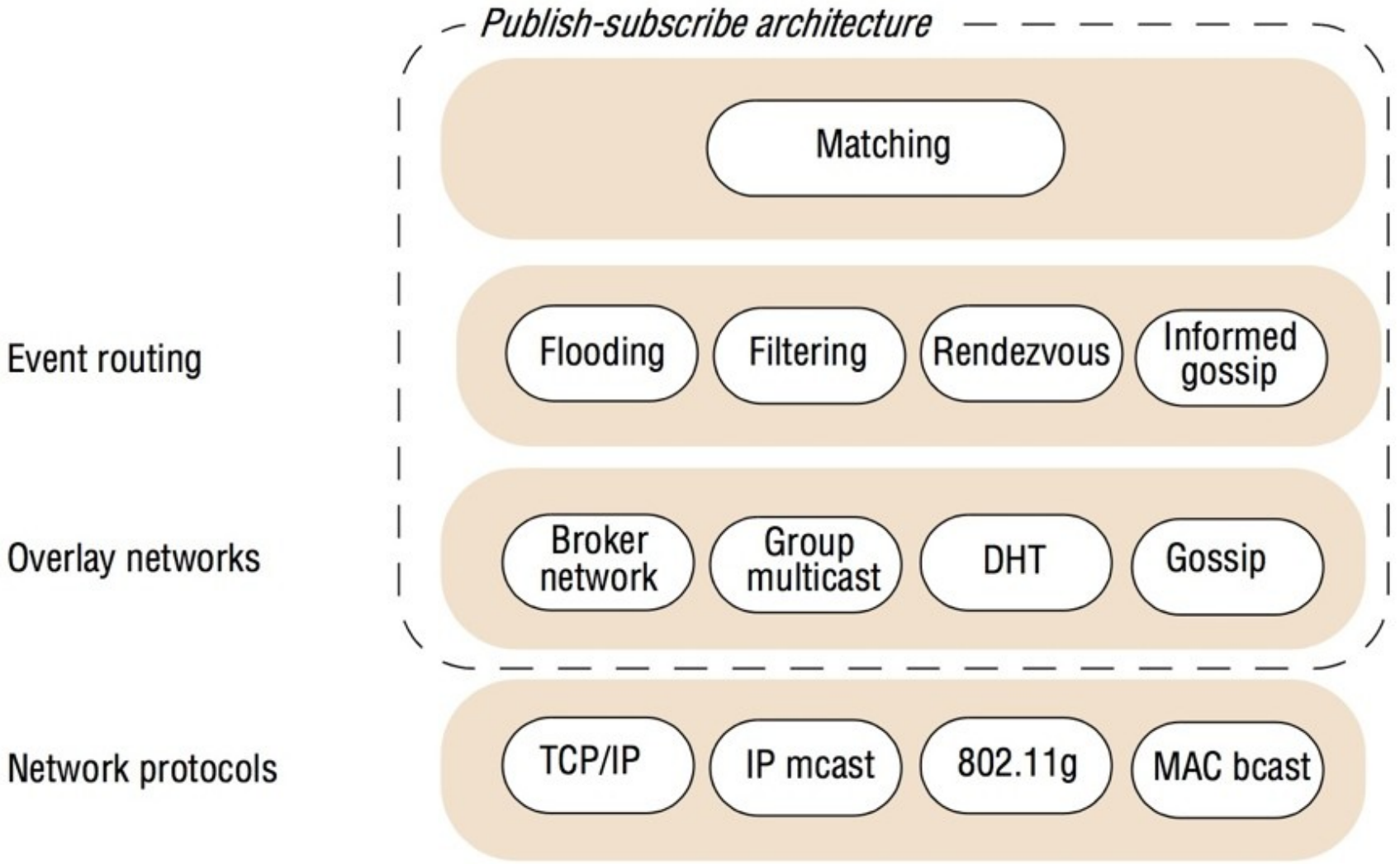
Centralized vs. distributed

- Centralized schemes simple
  - Implementing channel-based or topic-based simple
  - Map channels/topics onto groups
  - Use the group’s multicast (possibly reliable, ordered, ..)
  - Implementation of content/type/ more complicated
- Most implementations are network of brokers
- Some implementations are peer-to-peer (P2P)
  - All publisher and subscriber nodes act as the pub-sub broker

Distributed



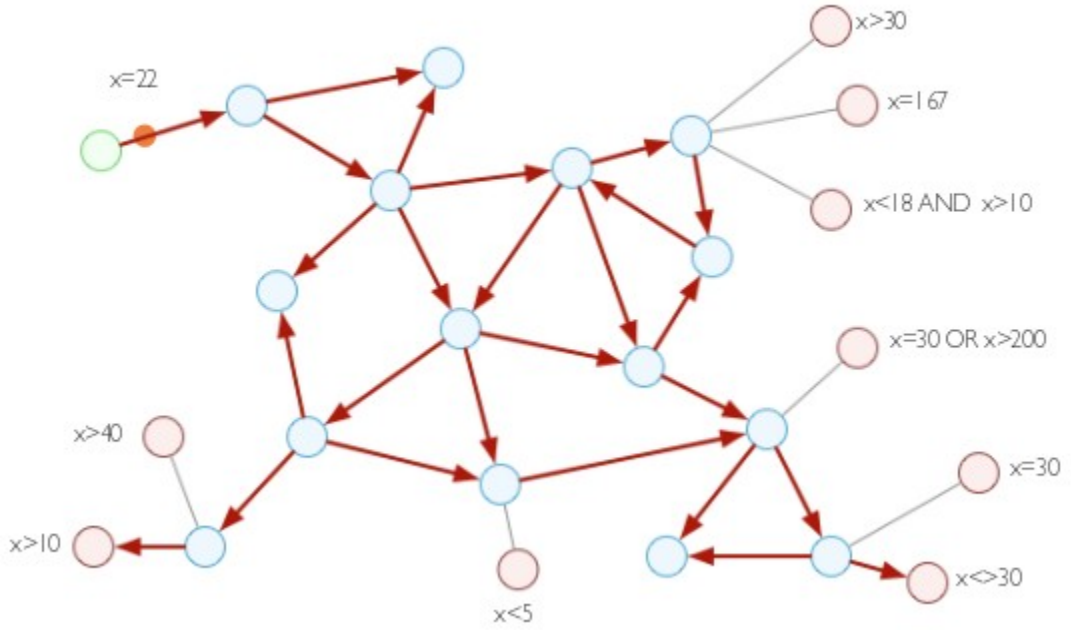
Distributed

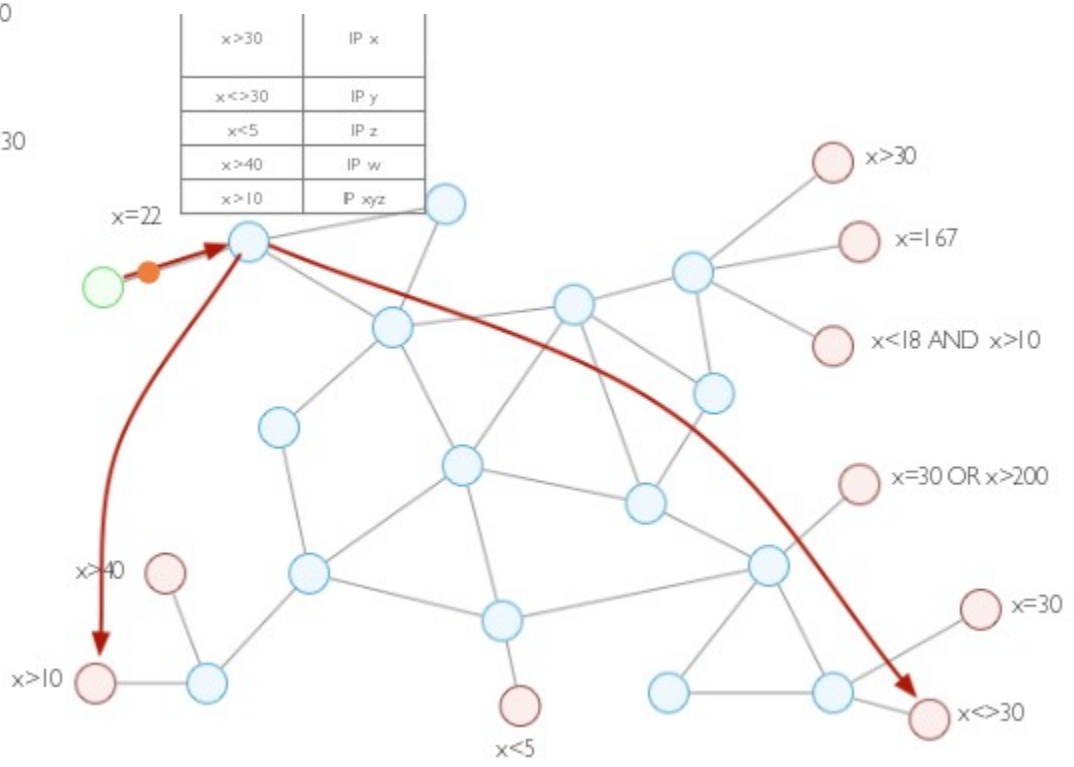


Distributed - content-based routing

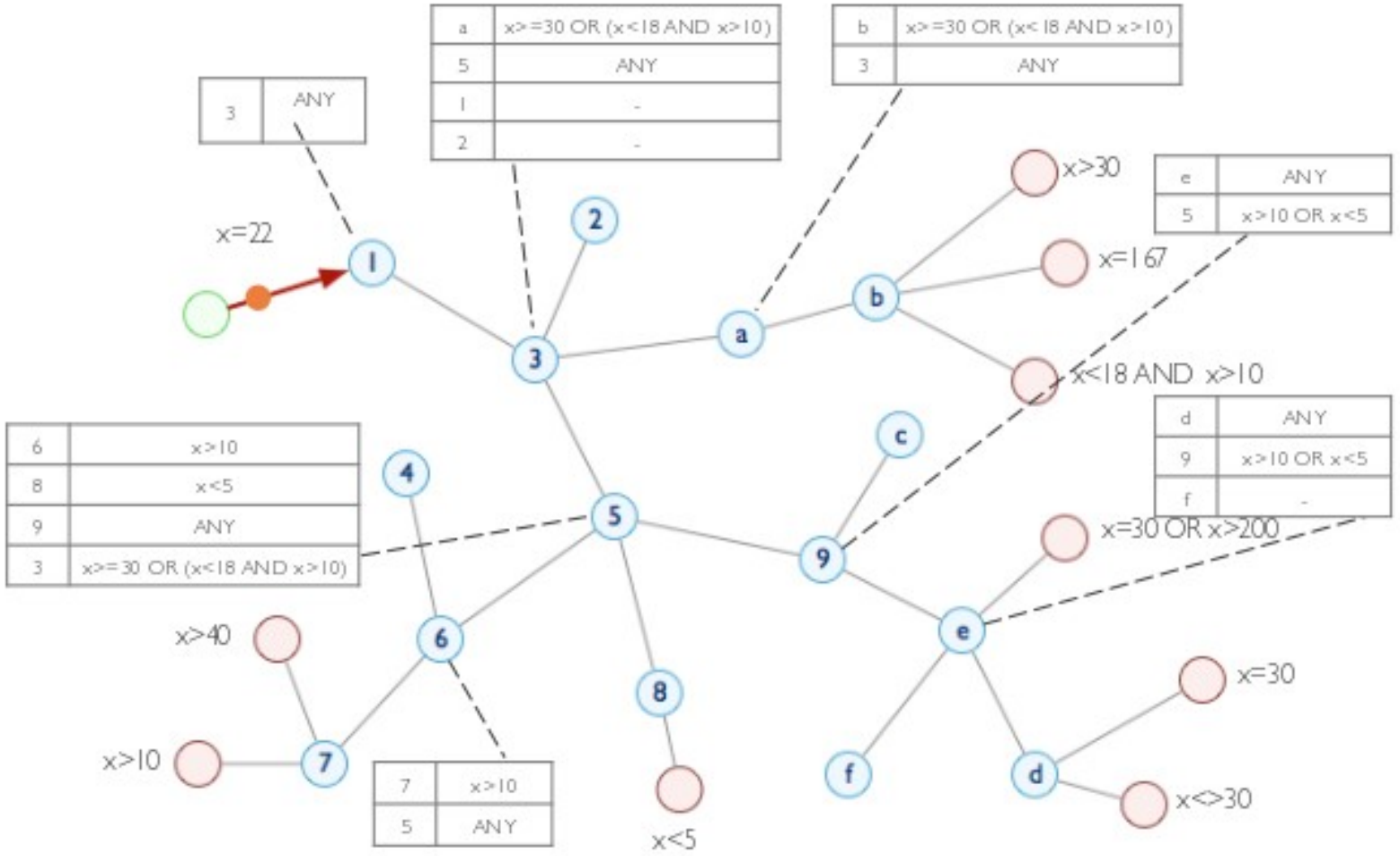
- Flooding (with duplicate suppression)
  - Simplest version
    - Send event to all nodes on a network
    - Can use underlying multicast/broadcast
  - More complicated
    - Brokers arranged in acyclic forwarding graph
    - Each node forwards to all its neighbors (except one that sent it to node)
- Filtering (filter-based routing)
  - Only forward where path to valid subscriber I.e., subscription info propagated through network towards publ's
  - Detail:
    - Each node maintain neighbors list
    - For each neighbor, maintain subscription list/criteria
    - Routing table with list of neighbors and subscribers downstream

Flooding


x=22
x>30
x=167
x<18 AND x>10
x=30 OR x>200
x>40
x=30
x>10
x<>30
x<5


x>30
IP x
x<>30
IP y
x<5
IP z
x>40
IP w
x>10
IP xyz
x=22
x>30
x=167
x<18 AND x>10
x=30 OR x>200
x>40
x=30
x>10
x<>30
x<5

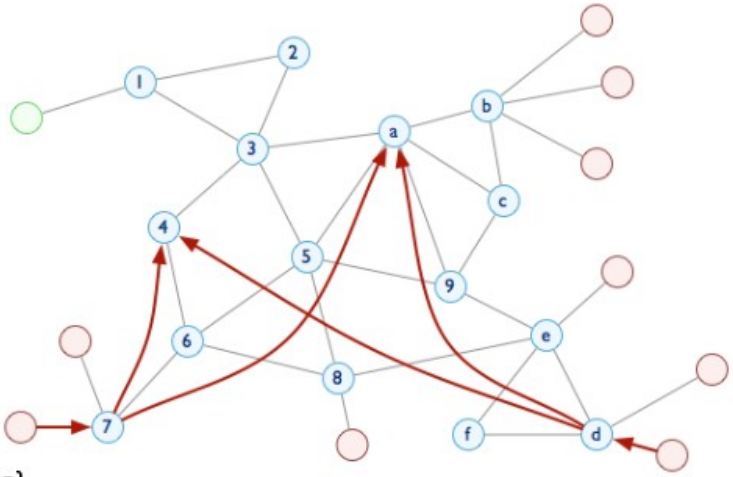
Filtering


3 ANY
a x>=30 OR (x<18 AND x>10)
5 ANY
1 -
2 -
b x>=30 OR (x<18 AND x>10)
3 ANY
e ANY
5 x>10 OR x<5
d ANY
9 x>10 OR x<5
f -
6 x>10
8 x<5
9 ANY
3 x>=30 OR (x<18 AND x>10)
7 x>10
5 ANY
x=22
x>30
x=167
x<18 AND x>10
x=30 OR x>200
x>40
x>10
x<5
x=30
x<>30
1
2
3
a
b
4
5
c
9
6
8
e
7
f
d

Distributed - content-based routing

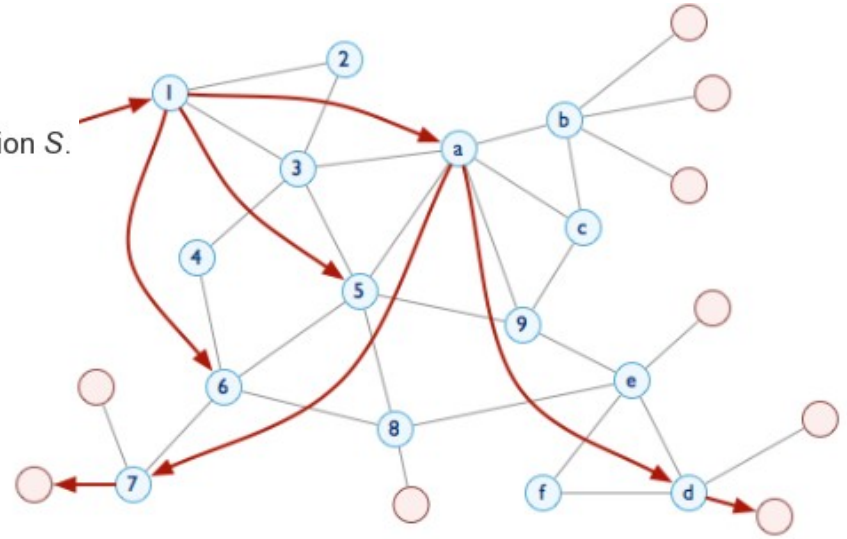
- Rendezvous
  - It is based on two functions,
    - SN and EN, used to associate respectively subscriptions and events to brokers in the system.
  - Given a subscription s,
    - SN(s) returns a set of nodes which are responsible for storing s and forwarding received events matching s to all those subscribers that subscribed it.
  - Given an event e,
    - EN(e) returns a set of nodes which must receive e to match it against the subscriptions they store.
  - Event routing is a two-phases process:
    - first an event e is sent to all brokers returned by EN(e), then those brokers match it against the subscriptions they store and notify the corresponding subscribers.
  - This approach works only if for each subscription s and event e, such that e matches s, the intersection between EN(e) and SN(s) is not empty (mapping intersection rule).

Rendezvous



SN(S) = {4,a}

Phase 1: two nodes issue the same subscription *S*.



EN(e) = {5,6,a}

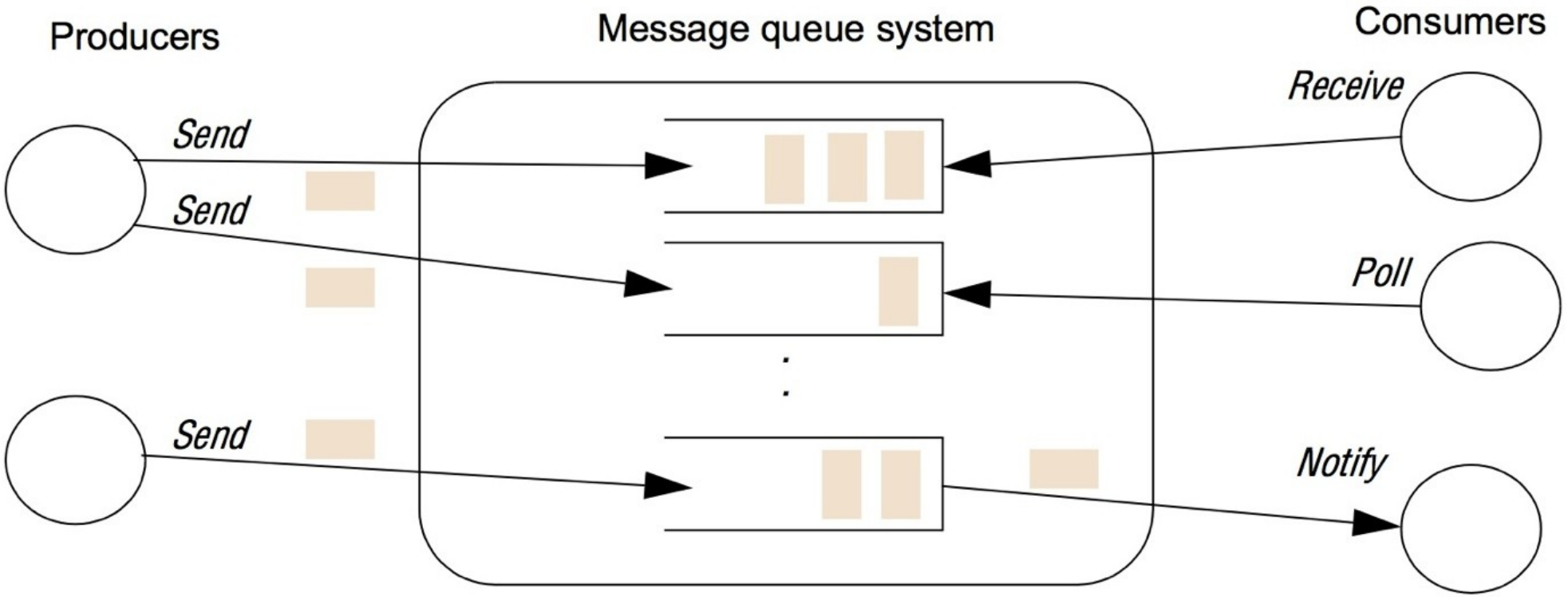
Broker a is the rendez-vous point between event *e* and subscription *S*.

Phase 2: an event *e* matching *S* is routed toward the rendez-vous node where it is matched against *S*.

Message Queues

- intermediary between producers and consumers of data
  - Point-to-Point, not one-to-many
  - Supports time and space uncoupling
- Programming model
  - producer sends message to specific queue
  - consumers can
    - Block
    - Non-block (polling)
    - Notify

Message Queues



Message Queues

- Many processes can send to a queue,
- many can remove from it
- Queuing policy:
  - usually FIFO, but also priority-based
- Consumers can select based on metadata
- Messages are persistent
  - Stored until removed (on disk)
- Transaction support:
  - all-or-none operations
- Automatic message transformation:
  - on arrival, message transforms data from one format to another (data heterogeneity)