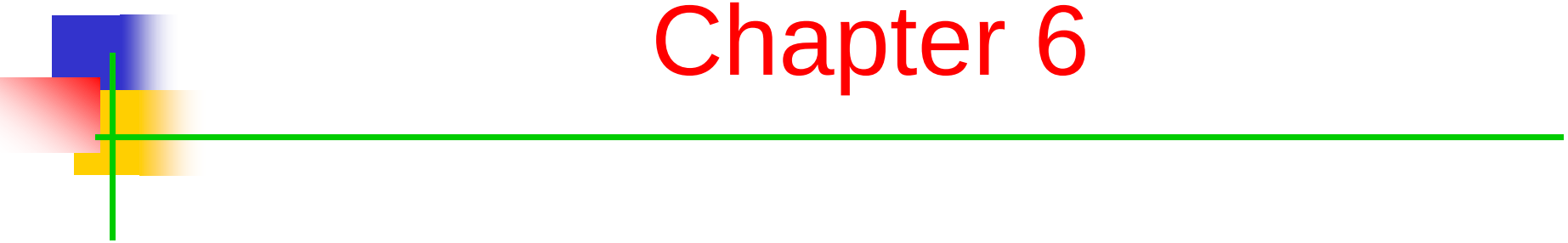
Indirect Communication
Chapter 6



Ali Fanian,
Isfahan University of Technology

Indirect Communication

- The essence of indirect communication is to communicate through an intermediary and hence have no direct coupling between the sender and the one or more receivers.

Direct Coupling

- RPC and RMI are all based on a direct coupling between a sender and a receiver, and this leads to a certain amount of rigidity in the system in terms of dealing with change.
  - In client-server interaction, because of the direct coupling, it is more difficult to replace a server
  - If the server fails, this directly affects the client, which must explicitly deal with the failure

Indirect Coupling

- In contrast, indirect communication avoids this direct coupling
- The literature refers to two key properties stemming from the use of an intermediary:
  - *Space uncoupling, in which the sender does not know or need to know the identity* of the receiver(s), and vice versa.
    - Participants (senders or receivers) can be replaced, updated, replicated or migrated.
  - *Time uncoupling, in which the sender and receiver(s) can have independent* lifetimes.
    - The sender and receiver(s) do not need to exist at the same time to communicate.

Space and time coupling in distributed systems

	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 6.3
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

- The main disadvantage is that there will inevitably be a performance overhead introduced by the added level of indirection

Asynchronous communication vs time uncoupling

- In asynchronous communication, a sender sends a message and then continues
- Time uncoupling adds the extra dimension that the sender and receiver(s) can have independent existences

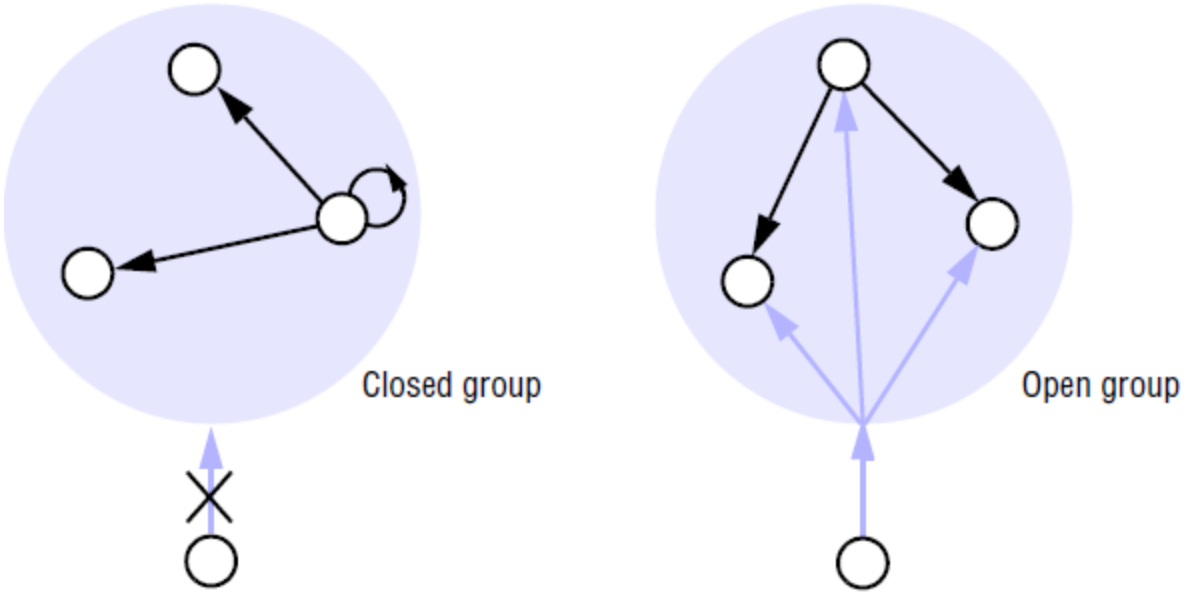
Indirect Communication Tech.

- Group communication
- Publish-subscriber
- Message queue system
- Shared memory

Group Communication

- *Group communication offers a service whereby a message is sent to a group*
  - the sender is not aware of the identities of the receivers

- Open and closed groups
  - A group is said to be *closed if only members of the group* may multicast to it



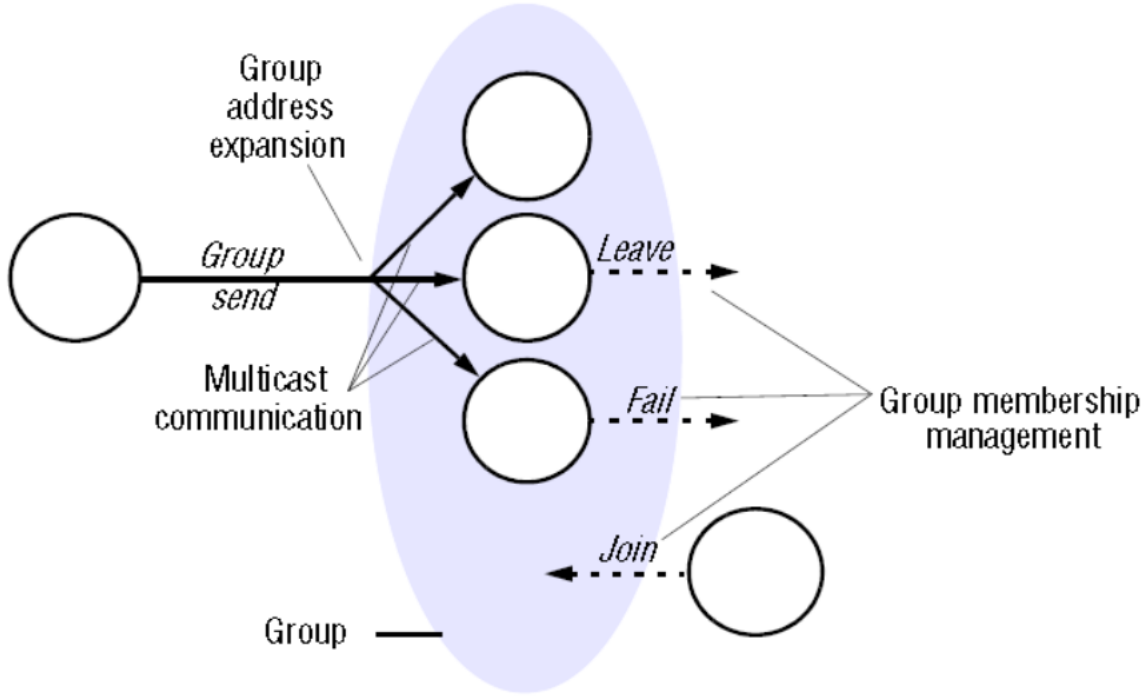
- *Overlapping and non-overlapping groups*
  - In *overlapping groups, entities (processes* or objects) may be members of multiple groups, and non-overlapping groups imply that membership does not overlap

Implementation issues

- **Reliability and ordering in multicast**
  - In group communication, all members of a group must receive copies of the messages sent to the group, generally with delivery guarantees.
  - Group communication services offer *ordered multicast*, with the option of one or more of the following properties (with hybrid solutions also possible)
    - *FIFO ordering*
    - *Causal ordering*
    - *Total ordering*

Implementation issues

- **Group membership management**
  - The key elements of group communication management are summarized in the following



Implementation issues

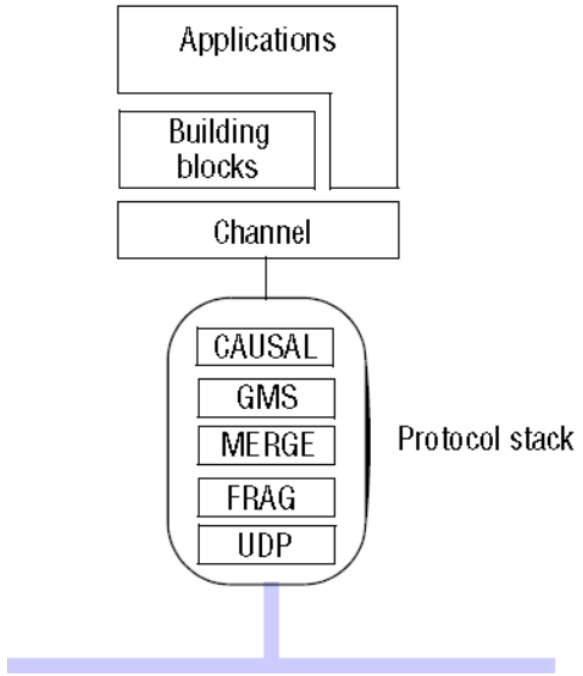
- **Group membership management**
  - a group membership service has four main tasks
    - *Providing an interface for group membership changes*
    - *Failure detection*
    - *Notifying members of group membership changes*
    - *Performing group address expansion*

Implementation issues

- IP multicast is a weak case of a group membership service, with some but not all of these properties.
  - Support join and leave
  - Performs address expansion
  - Doesn’t provide group member information
  - Multicast delivery is not coordinated with membership changes

Case study: the JGroups toolkit

- JGroups is a toolkit for reliable group communication written in Java
- The architecture of JGroups is shown in the following



Case study: the JGroups toolkit

- the main components of the JGroups implementation are
  - *Channels*
    - A process interacts with a group through a *channel* object, which acts as a handle onto a group.
      - *getView, getState*

Case study: the JGroups toolkit

- the main components of the JGroups implementation are
  - *Building blocks*
    - Building blocks are higher-level abstractions on top of the channel
    - Examples of building blocks in JGroups are
      - *MessageDispatcher*
      - *RpcDispatcher*
      - *NotificationBus*

Case study: the JGroups toolkit

- **The protocol stack**
  - The layer referred to as UDP is the most common transport layer in Jgroups
  - FRAG implements message packetization and is configurable in terms of the maximum message size
  - MERGE is a protocol that deals with unexpected network partitioning and the subsequent merging of subgroups after the partition
  - GMS implements a group membership protocol to maintain consistent views of membership across the group
  - CAUSAL implements causal ordering

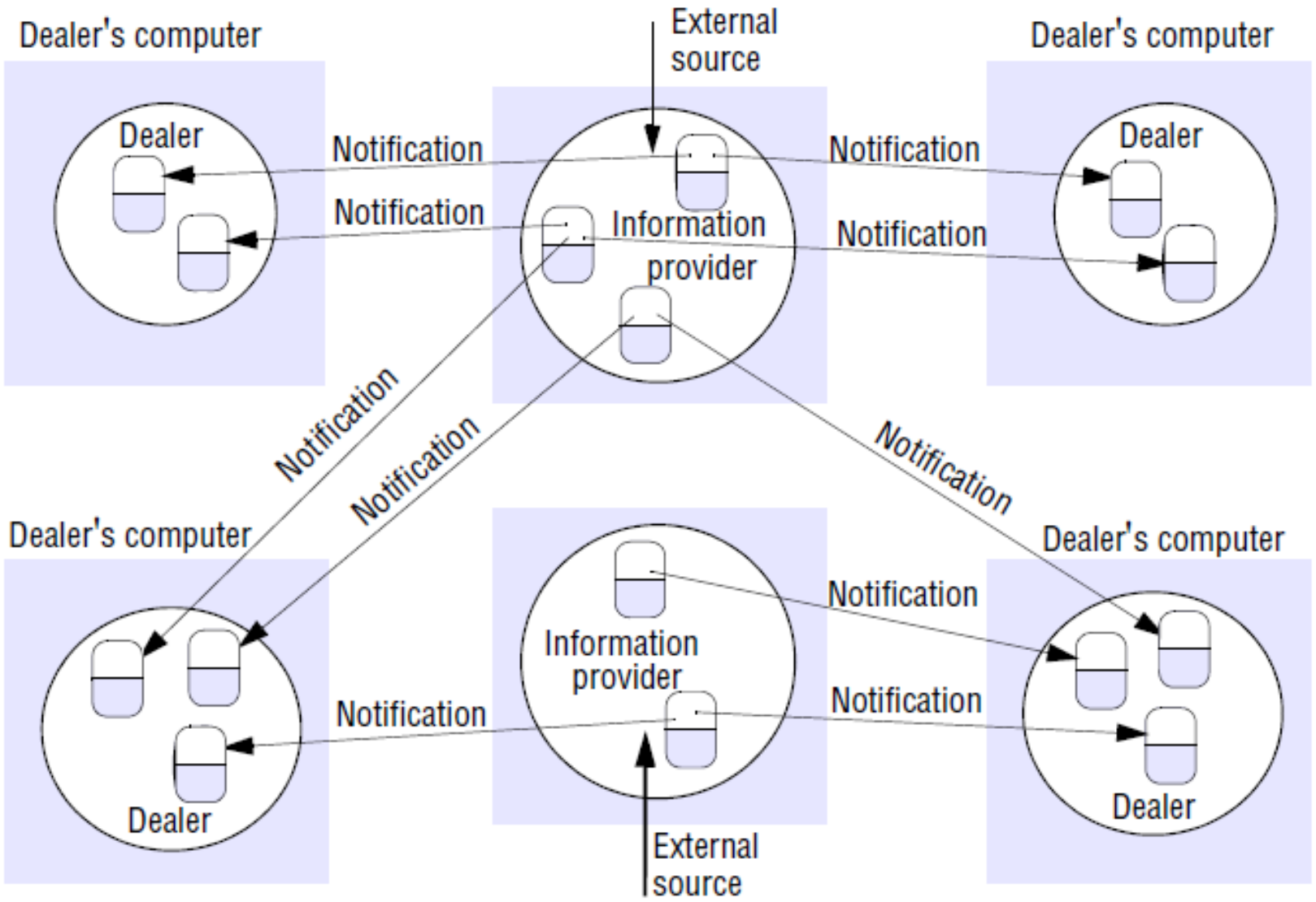
Publish-Subscribe Systems

- A system where *publishers publish structured* events to an event service and *subscribers express interest in particular events through subscriptions which can be arbitrary patterns over the structured events*

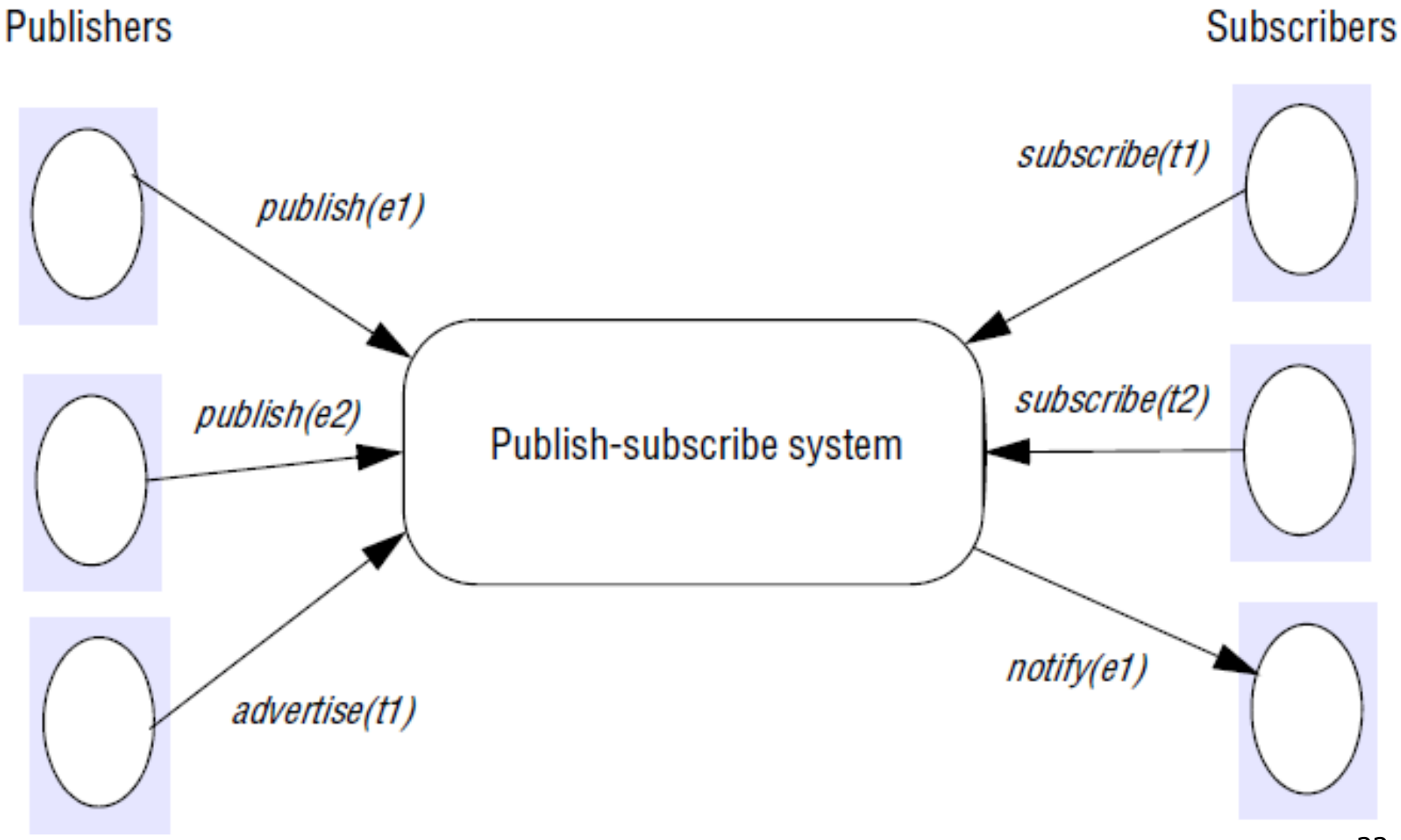
Applications of publish-subscribe systems

- financial information systems
- support for cooperative working, where a number of participants need to be informed of events of shared interest
- a broad set of monitoring applications, including network monitoring in the Internet.
- ….

Dealing room system



The publish-subscribe paradigm



The Subscription (filter) Model

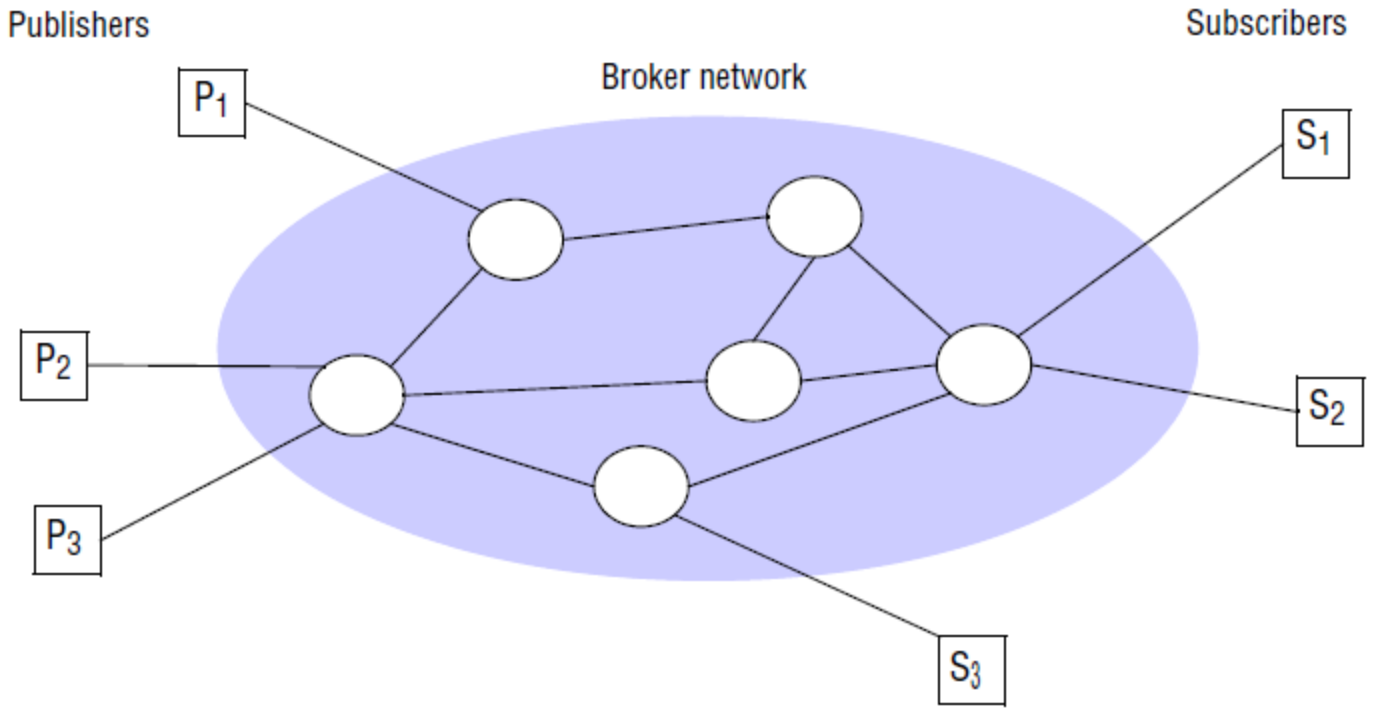
- *Channel-based*
  - publishers publish events to named channels and subscribers then subscribe to one of these named channels to receive all events sent to that channel
- *Topic-based*
  - each notification is expressed in terms of a number of fields, with one field denoting the topic
  - Subscriptions are then defined in terms of the topic of interest
- *Content-based*
  - a content-based filter is a query defined in terms of compositions of constraints over the values of event attributes

Implementation issues

- **Centralized versus distributed implementations**
  - **Centralized model:** implementation in a single node with a server on that node acting as an event broker
    - Interaction with the broker is then through a series of point-to-point messages
    - this can be implemented using message passing

A network of brokers

- Distributed model : In such schemes, the centralized broker is replaced by a *network of brokers that cooperate to offer he desired functionality as* illustrated in Figure



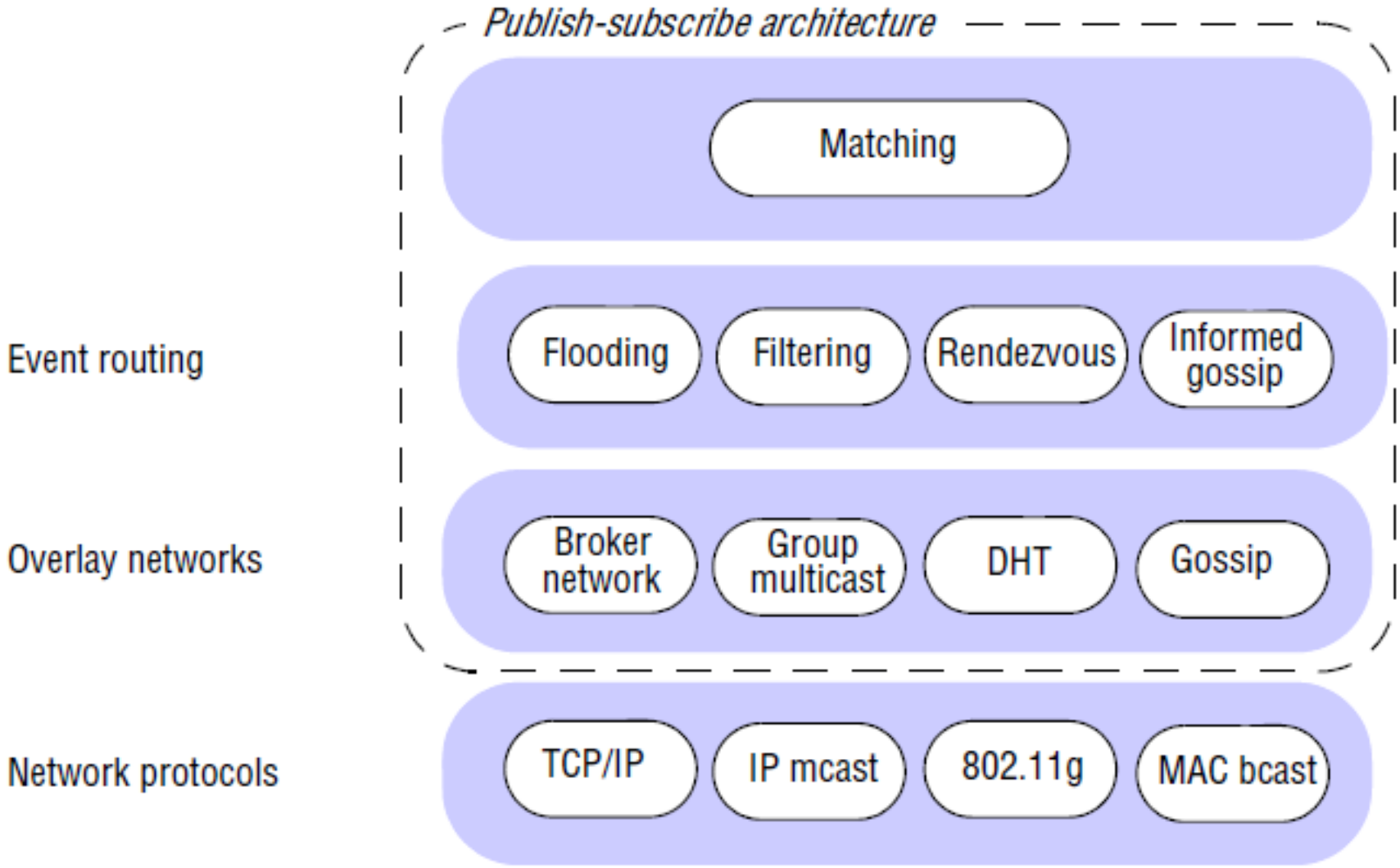
Overall systems architecture

- the implementation of centralized schemes is relatively straightforward
- the implementations of channel-based or topic-based schemes are relatively straightforward
- The distributed implementation of content-based approaches is more complex

Content Based Approch

- For content-based approaches, this problem is referred to as *content-based routing (CBR), with the goal being to exploit content information to efficiently route* events to their required destination

The architecture of publish-subscribe systems in Content Based



- *Flooding*
  - sending an event notification to all nodes in the network and then carrying out the appropriate matching at the subsciber end.
- *Filtering*
  - Brokers forward notifications through the network only where there is a path to a valid subscriber
  - This is achieved by propagating subscription information through the network towards potential publishers and then storing associated state at each broker

- Rendezvous
  - this approach defines rendezvous nodes, which are broker nodes responsible for a given subset of the event space

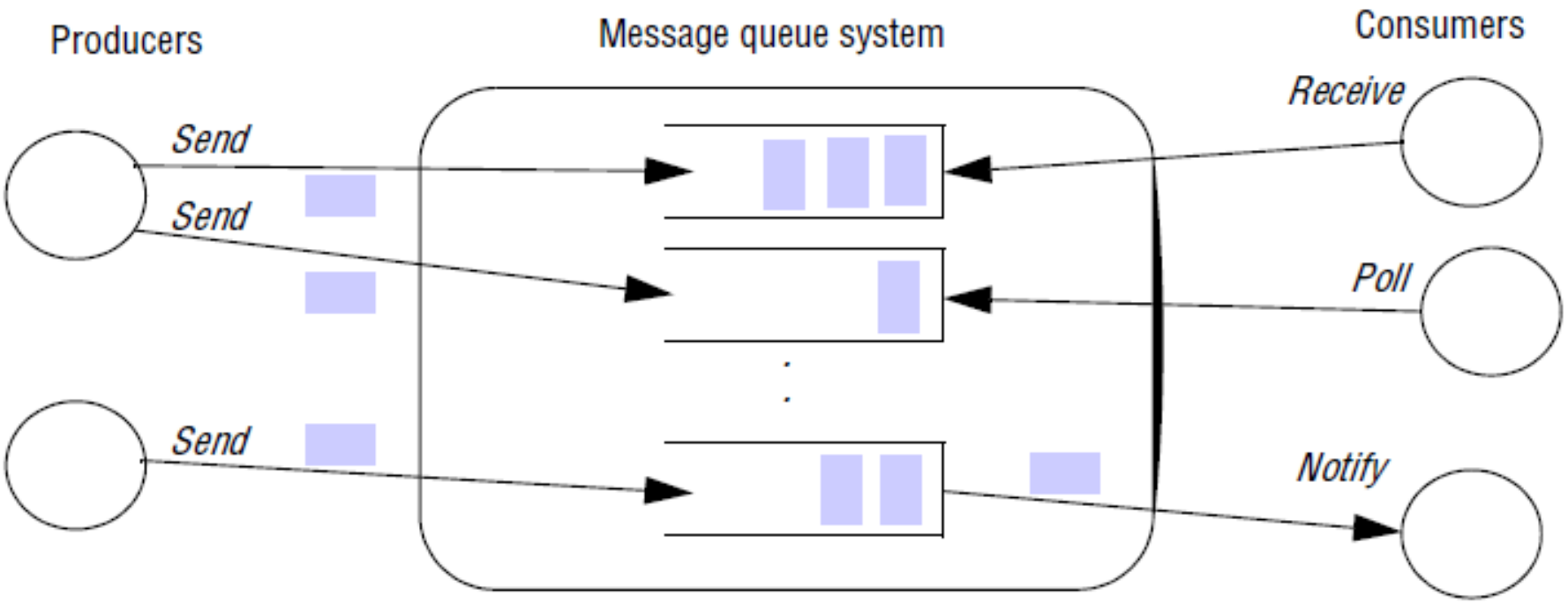
Message queues

- Provide a *point-to-point service using the concept of a message queue as an indirection, thus* achieving the desired properties of space and time uncoupling
- sender places the message into a queue, and it is then removed by a single process

styles of receive

- A blocking receive, which will block until an appropriate message is available;
- A non-blocking receive (a polling operation), which will check the status of the queue and return a message if available, or a not available indication otherwise;
- A notify operation, which will issue an event notification when a message is available in the associated queue.

The message queue paradigm



Distributed Shared memory approaches