CMPU4021 Distributed Systems

INDIRECT COMMUNICATION

Based on Chapter 6: Coulouris, Dollimore and Kindberg,
Distributed Systems: Concepts and Design

Indirect Communication

- The essence of indirect communication is to communicate through an intermediary

- No direct coupling between the sender and the one or more receivers.

Coupling Approaches for Distributed Systems

- Coupling refers to the degree of direct knowledge that one element has of another.

- Direct coupling
  - Interaction through a stable interface
  - API call is hard coded

- Loose coupling
  - Resilient relationship between two or more systems or organizations with some kind of exchange relationship
  - Each end of the transaction makes its requirements explicit, e.g. as an interface description, and makes few assumptions about the other end.

- Decoupled
  - de-coupled in space and time using (event) messages, e.g. via Message-oriented Middleware (MoM), publish-subscribe
  - Often asynchronous stateless indirect communication (e.g. publish-subscribe or complex event processing systems)

Coupling Approaches – Advantages and disadvantages

- Direct coupling
  - Hard to change since subsequent changes in implementation are needed

- Loose coupling
  - Enhanced flexibility; a change in one module will not require a change in the implementation of another module
  - Example:
    - (Web) Services, which are called via interface; service behind interface might be replaced

- Decoupled
  - Asynchronous communication
  - Parallel processing
  - Difficult to ensure transactional integrity
  - Issues in maintaining synchronisation
  - Example:
    - Event-driven Publish/Subscribe; events are received and sent

Key properties

- A simple client-server
  - The direct coupling
    - it is more difficult to replace a server with an alternative one offering equivalent functionality.
    - If the server fails, this directly affects the client, which must explicitly deal with the failure.

- Indirect communication avoids this direct coupling and hence inherits properties, such as:
  - Space uncoupling
  - Time uncoupling

Key properties: Space and time coupling

Space uncoupling

- *T*he sender does not know or need to know the identity of the receiver(s), and vice versa. Because of this space uncoupling, the system developer has many degrees of freedom in dealing with change:
    - participants (senders or receivers) can be replaced, updated, replicated or migrated.
    - E.g. IP multicast

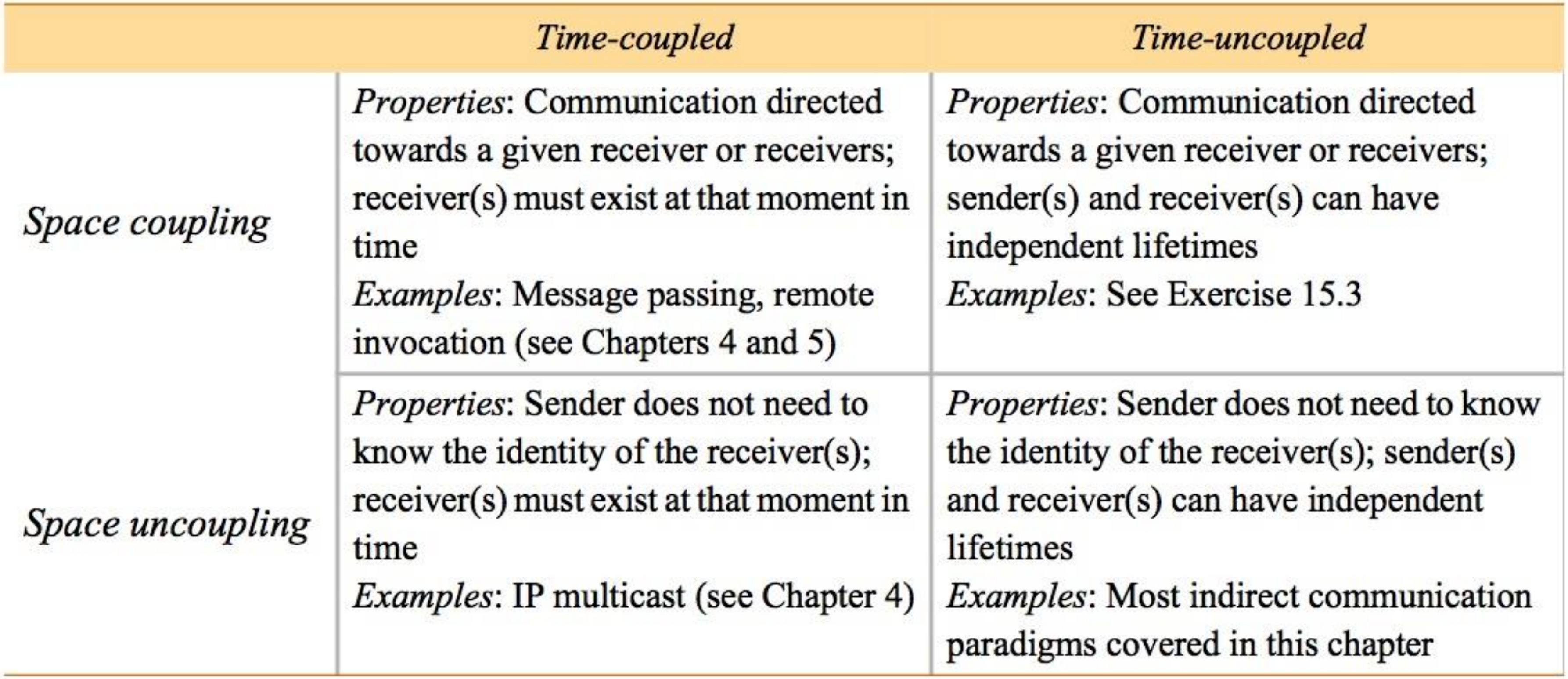
Time uncoupling

- *T*he sender and receiver(s) can have independent lifetimes. In other words, the sender and receiver(s) do not need to *exist* at the same time to communicate.
    - This has important benefits, for example, in more volatile environments where senders and receivers may come and go.

Examples:

- Mobile environments where users may rapidly connect to and disconnect from the network
- Managing event feeds in financial systems

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 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

Indirect Communication Usage

- In distributed systems where change is anticipated
  - in mobile environments where users may rapidly connect to and disconnect from the global network – and must be managed to provide more dependable services.

- For event dissemination in distributed systems where the receivers may be unknown and liable to change
  - E.g. in managing event feeds in financial systems

Relationship with asynchronous communication

- Asynchronous communication
  - A sender sends a message and then continues (without blocking)
  - there is no need to meet in time with the receiver to communicate
- Time uncoupling
  - The sender and the receiver(s) can have independent existence
    - for example, the receiver may not exist at the time communication is initiated

Indirect communication techniques

1. Group communication

   - in which communication is via a group abstraction with the sender unaware of the identity of the recipients

2. Publish-subscribe systems

   - a family of approaches that all share the common characteristic of disseminating events to multiple recipients through an intermediary

3. Message queue systems

   - messages are directed to the familiar abstraction of a queue with receivers extracting messages from such queues

4. Shared memory–based approaches

   - distributed shared memory and tuple space approaches, which present an abstraction of a global shared memory to programmers.

GROUP COMMUNICATION

Group Communication

- A message is sent to a group and then this message is delivered to all members of the group.

- The sender is not aware of the identities of the receivers.

- Represents an abstraction over multicast communication

- May be implemented over
  - IP multicast; or
  - An equivalent *overlay network* adding significant extra value in terms of
    - managing group membership,
    - detecting failures and
    - providing reliability and ordering guarantees.

Overlay networks

Overlay network

- A virtual network consisting of nodes and virtual links, which sits on top of an underlying network (such as an IP network) and offers something that is not otherwise provided:
  - a service that is tailored towards the needs of a class of application or a particular
  - higher-level service – for example, multimedia content distribution;
  - more efficient operation in a given networked environment – for example routing in an ad hoc network;
  - an additional feature – for example, multicast or secure communication.
- Example
  - Skype

Overlay networks - advantages

- They enable new network services to be defined without requiring changes to the underlying network, a crucial point given the level of standardization in this area and the difficulties of amending underlying router functionality.

- They encourage experimentation with network services and the customization of services to particular classes of application.

- Multiple overlays can be defined and can coexist, with the end result being a more open and extensible network architecture.

Overlay networks - disadvantages

- An extra level of indirection
  - May incur a performance penalty
  - They add to the complexity of network services
    - when compared, for example, to the relatively simple architecture of TCP/IP networks.

Group Communication: Key areas of application

- An important building block for reliable distributed systems
- The reliable dissemination of information to potentially large numbers of clients,
  - E.g. in the 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
  - E.g. 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 Communication: Programming Model

- The central concept is that of a group with associated group membership
  - processes may join or leave the group.
- Processes can then send a message to this group and have it propagated to all members of the group with certain guarantees in terms of reliability and ordering.
- Thus, group communication implements multicast communication
  - in which a message is sent to all the members of the group by a single operation.
- A process issues only one multicast operation to send a message to each of a group of processes

Group Communication: Process groups and object groups

- Most group services are using the concept of *process groups*
  - Communicating entities are processes
  - Messages are delivered to processes and no further support for dispatching is provided.
  - Messages are typically unstructured byte arrays with no support for marshalling or complex data types

- *Object groups* provide a higher-level approach to group computing
  - Collection of objects (normally instances of the same class) that process the same set of invocations concurrently, with each returning responses
  - Client objects need to be aware of the replication => they invoke operations on a single, local objects which acts as a proxy for the group
  - Proxy uses a group communication systems to send the invocations to the members of the object group

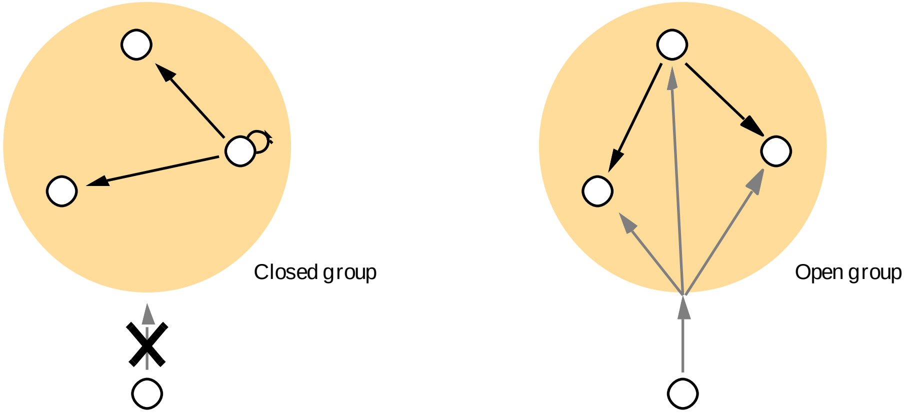
Group Communication: Other Key Distinctions

- A wide range of group communication services has been developed, and they vary in the assumptions they make:
  - Closed and open groups
  - Overlapping and non-overlapping groups
  - Synchronous and asynchronous systems

Closed and open groups

- Closed group
  - if only members of the group may multicast to it.
  - A process in a closed group delivers to itself any message that it multicasts to the group.
  - Useful for delivering events to groups of interested processes.

- Open group
  - if processes outside the group may send to it.
  - The categories ‘open’ and ‘closed’ also apply with analogous meanings to mailing lists.
  - Useful for cooperating servers to send messages to one another that only they should receive.



Overlapping and non-overlapping groups

- In *overlapping groups*
  - entities (processes or objects) may be members of multiple groups

- Non-overlapping groups
  - Imply that membership does not overlap
  - that is, any process belongs to at most one group.

- In real-life systems, it is realistic to expect that group membership will overlap.

Implementation issues: Reliability in multicast

- Reliability in one-to-one communication
  - Integrity
    - the message received is the same as the one sent, and no messages are delivered twice
  - Validity
    - any outgoing message is eventually delivered

- Reliable Multicast
  - Integrity
    - delivering the messages correctly at most once
  - Validity
    - guaranteeing that a message sent will eventually be delivered
  - Agreement
    - Stating that if the message is delivered to one process, then it is delivered to all processes in the group

Implementation issues: Ordering in multicast

FIFO ordering

- First-in-first-out (FIFO) ordering is concerned with preserving the order from the perspective of a sender process, in that if a process sends one message before another, it will be delivered in this order to all processes in the group.

Causal ordering

- Causal ordering takes into account causal relationships between the messages, in that if a message happens before another message in the distributed system this is so-called causal relationship will be preserved in the delivery of the associated message at all processes.

Total ordering

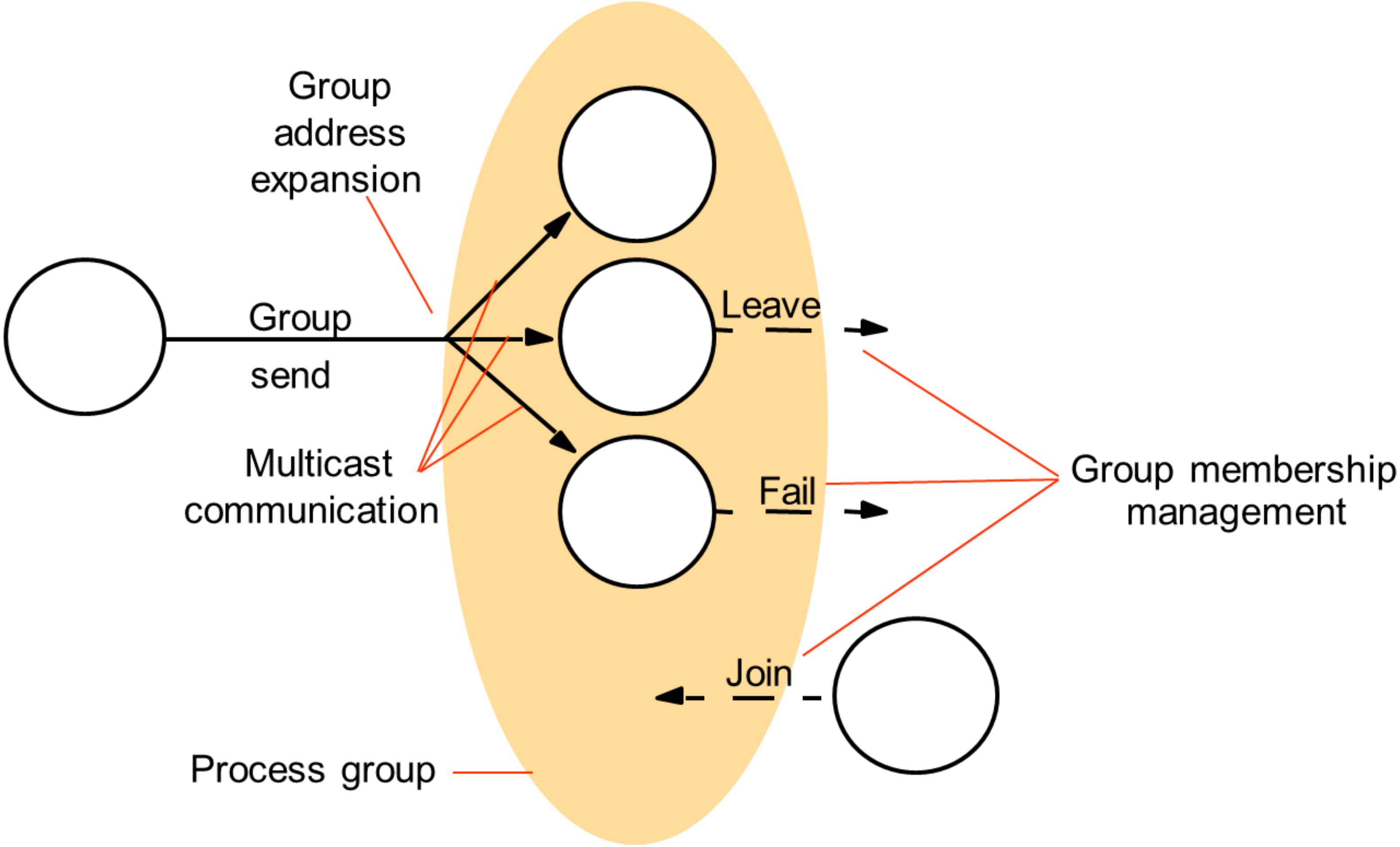
- In total ordering, if a message is delivered before another message at one process, then the same order will be preserved at all processes.

Implementation issues: Group membership management

Group membership management - four main tasks:

- Providing an interface for group membership changes
- Failure detection
- Notifying members of group membership changes
- Performing group address expansion
  - can coordinate multicast delivery with membership changes by controlling address expansion.

The role of group membership management



JGroups toolkit

- JGroups is a toolkit for reliable group communication written in Java
- JGroups supports process groups in which processes are able to
  - join or leave a group,
  - send a message to all members of the group, or indeed to a single member, and
  - receive messages from the group.
- The toolkit supports a variety of reliability and ordering guarantees
- Offers a group membership service.

PUBLISH-SUBSCRIBE SYSTEMS

Publish-Subscribe Systems

- Publish-Subscribe Systems
- Also known as
  - as *distributed event-based systems*
- A publish-subscribe system is 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.
- For example, a subscriber could express an interest in all events related to a book, such as the availability of a new edition or updates to the related web site.
- The task of the publish subscribe system is to match subscriptions against published events and ensure the correct delivery of *event notifications*.
- A given event will be delivered to potentially many subscribers, and hence publish-subscribe is fundamentally a one-to-many communications paradigm.

Publish-subscribe systems

- Used in a wide variety of application domains, particularly those related to the large-scale dissemination of events.

- Examples:
  - financial information systems;
  - other areas with live feeds of real-time data (including RSS feeds);
  - support for cooperative working, where a number of participants need to be informed of events of shared interest;
  - support for ubiquitous computing, including the management of events emanating from the ubiquitous infrastructure (for example, location events)
  - a broad set of monitoring applications, including network monitoring in the Internet.

- Publish-subscribe is also a key component of Google’s infrastructure, including for example the dissemination of events related to advertisements, such as ‘ad clicks’, to interested parties.

Publish-subscribe systems: The programming model

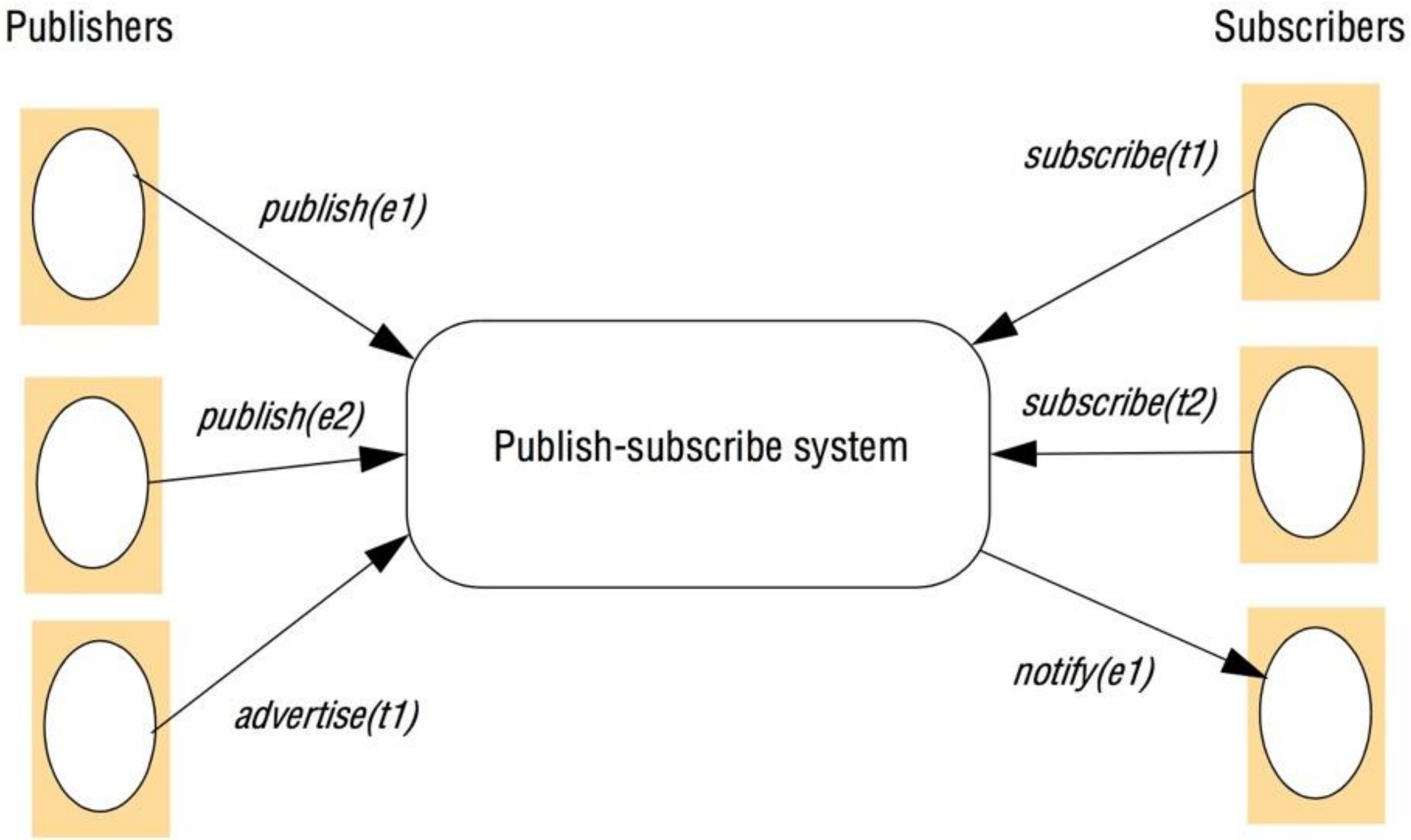
- Publishers disseminate an event *e* through a *publish(e)* operation and subscribers express an interest in a set of events through subscriptions.

- In particular, they achieve this through a *subscribe(f)* operation where *f* refers to a filter
  - that is, a pattern defined over the set of all possible events.
  - The expressiveness of filters (and hence of subscriptions) is determined by the subscription model

- Subscribers can later revoke this interest through a corresponding *unsubscribe(f)* operation.

- When events arrive at a subscriber, the events are delivered using a *notify(e)* operation.

The publish-subscribe paradigm



Events & Notification

- Events use idea:
  - One object can react to a change occurring in another object.
  - Notification of events are asynchronous and determined by their receiver.
- Actions by users are seen as *events* changing the objects that represent the state of the application
- Other objects can react to such events – *notifications* of such events are asynchronous and determined by the receivers
- Distributed event-based systems extend this model to allow objects at different locations to be notified of events

Events & Notification

- Distributed event-based systems:
  - Allow multiple objects at different locations to be notified of events taking place at an object.
  - Use *publish-subscribe* paradigm – an object that generates events *publishes* the type of events that it will make available for observation by other objects;
  - Objects that want to receive notifications from an object that has published its events *subscribe* to the types of events that are of interest to them.
  - Objects that represent events are called *notifications* – may be stored, sent in a message, queried;
  - Subscribing to a particular type of event is also called *registering interest* in that type of event.

Events and notifications

- Main characteristics:
  - *Heterogeneous* – components in a distributed system that were not designed to interoperate can be made to work together (when using event notifications)
  - *Asynchronous* – notifications are sent asynchronously by event-generating objects to all the objects that have subscribed to them to prevent publishers needing to synchronise with subscribers
    - publishers and subscribers need to be decoupled

- Event types:
  - *Attributes:* specify information about an event, such as the name or identifier of the object that generated it, the operation, its parameters and the time (or a sequence number).
  - When subscribing to an event, the type of event is specified, sometimes modified with a criterion as to the value of the attribute.

Dealing room example

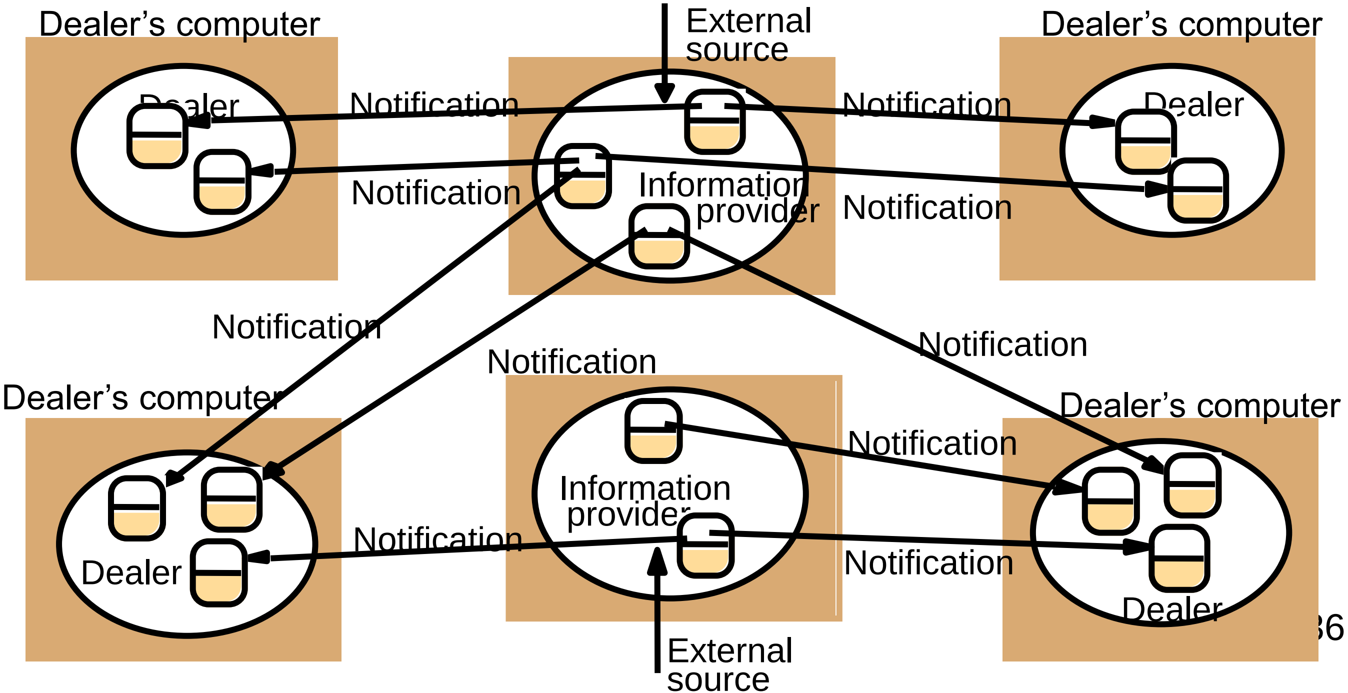
Task:

- *Allow dealers using computers to see the latest information about the market prices of the stock they deal in. The market price for a single named stock is represented by an object with several instance variables. The information arrives in the dealing room from several different external sources in the form of updates to some or all of the instance variables of the object representing the stock and is collected by process we call information providers. Dealers are typically interested only in their specialist stocks.*

Dealing room example (cont)

Could be modelled by processes with two different tasks:

- An information provider process continuously receives new trading information from a single external source and applies it to the appropriate stock objects. Each of the updates to a stock object is regarded as an event. The stock object experiencing such events notifies all of the dealers who have subscribed to the corresponding stock.
- A dealer process creates an object to represent each named stock that the user asks to have displayed. This local object subscribes to the object representing that stock at the relevant information provider. It then receives all the information sent to it in notifications and displays it to the user.



The participants in distributed event notification

- The main component is an event service that maintains a database of published events and of subscribers’ interests.

- Events at an object of interest are published at the event service.

- Subscribers inform the event service about the types of events they are interested in. When an event occurs at an object of interest, a notification is sent to the subscribers to that type of event.

The roles of the participating objects

- *The object of interest*:
  - Experiences changes of state, as a result of its operations being invoked.
    - E.g. events such as a person wearing an active badge entering a room, in which case the room is the object of interest and the operation consists of adding information about the new person to its records of who is in the room.
- *Event*
  - occurs at an object of interest as the result of the completion of a method execution.
- *Notification*:
  - An object that contains information about an event; it contains the type of the event and its attributes such as the identity of the object of interest, the method invoked, the time of occurrence or a sequence number.

The roles of the participating objects (cont)

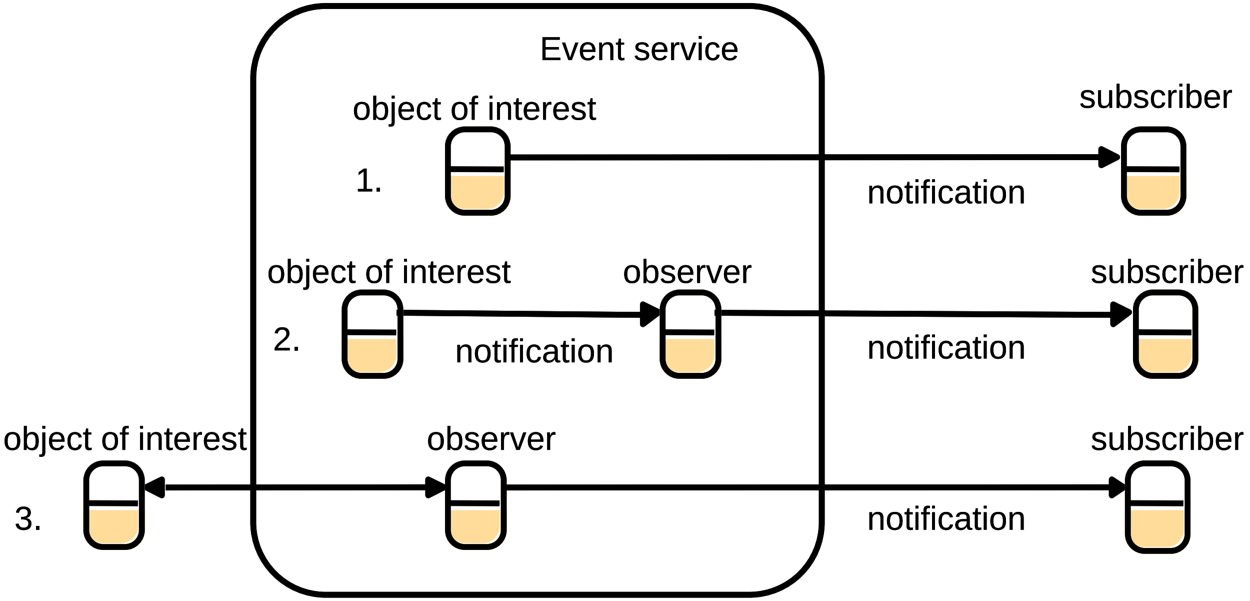
- *Subscriber*:
  - An object that has subscribed to some type of events in another object. It receives notifications about such events.

- *Observer objects*:
  - Purpose – to decouple an object of interest from its subscribers. An object of interest can have many different subscribers with different interests.

- Publisher:
  - An object that declares that it will generate notifications of particular types of event; it may be an object of interest or an observer.

Architecture for distributed event notification

1. An object of interest inside the event service without an observer. It sends notifications directly to the subscribers.
2. An object of interest inside the event service with an observer. The object of interest sends notifications via the observer to the subscribers.
3. An object of interest outside the event service. In this case, an observer queries the object of interest in order to discover when events occur. The observer sends notifications to the subscribers.



Roles for observers

Examples of roles:

- *Forwarding*: A forwarding observer may carry out all the work of sending notifications to subscribers on behalf of one or more objects of interest.

- *Filtering of notifications*: Filters may be applied by an observer so as to reduce the number of notifications received according to some predicate on the contents of each notification.
  - E.g. an event might relate to withdrawals from a bank account, but the recipient is interested only in those greater than €100.

- *Patterns of events*: When an object subscribes to events at an object of interest, they can specify patterns of events that they are interested in.

- *Notification mailboxes*: notification may need to be delayed until a potential subscriber is ready to receive them. The subscriber should be able to turn delivery on and off as required.

Publish-subscribe systems: Centralized versus distributed implementations

Centralized

- The simplest approach is to centralize the implementation in a single node with a server on that node acting as an event broker.

- Publishers then publish events (and optionally send advertisements) to this broker, and subscribers send subscriptions to the broker and receive notifications in return.

- Interaction with the broker is then through a series of point-to-point messages;
  - this can be implemented using message passing or remote invocation.

Publish-subscribe systems: Centralized versus distributed implementations

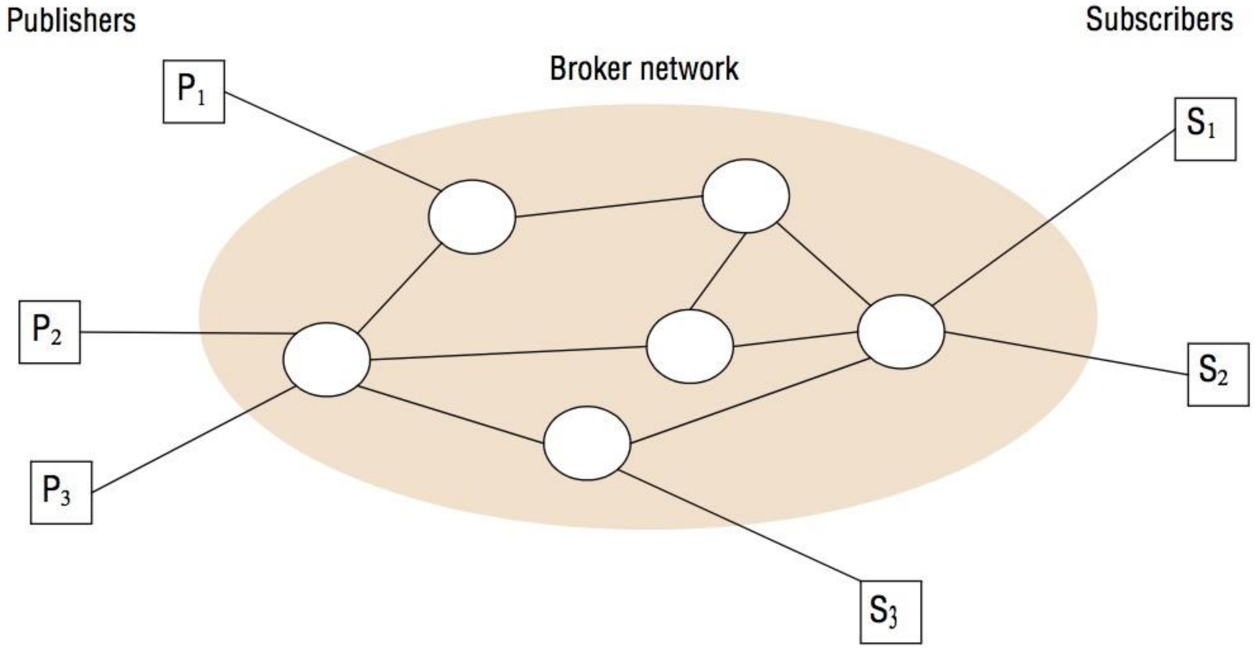
Centralized

- lacks resilience and scalability
- the centralized broker represents a single point for potential system failure and a performance bottleneck.

Publish-subscribe systems: Centralized versus distributed implementations

Distributed implementations

- The centralized broker is replaced by a network of brokers that cooperate to offer the desired.
- Such approaches have the potential to survive node failure and have been shown to be able to operate well in Internet-scale deployments.



Message queue systems

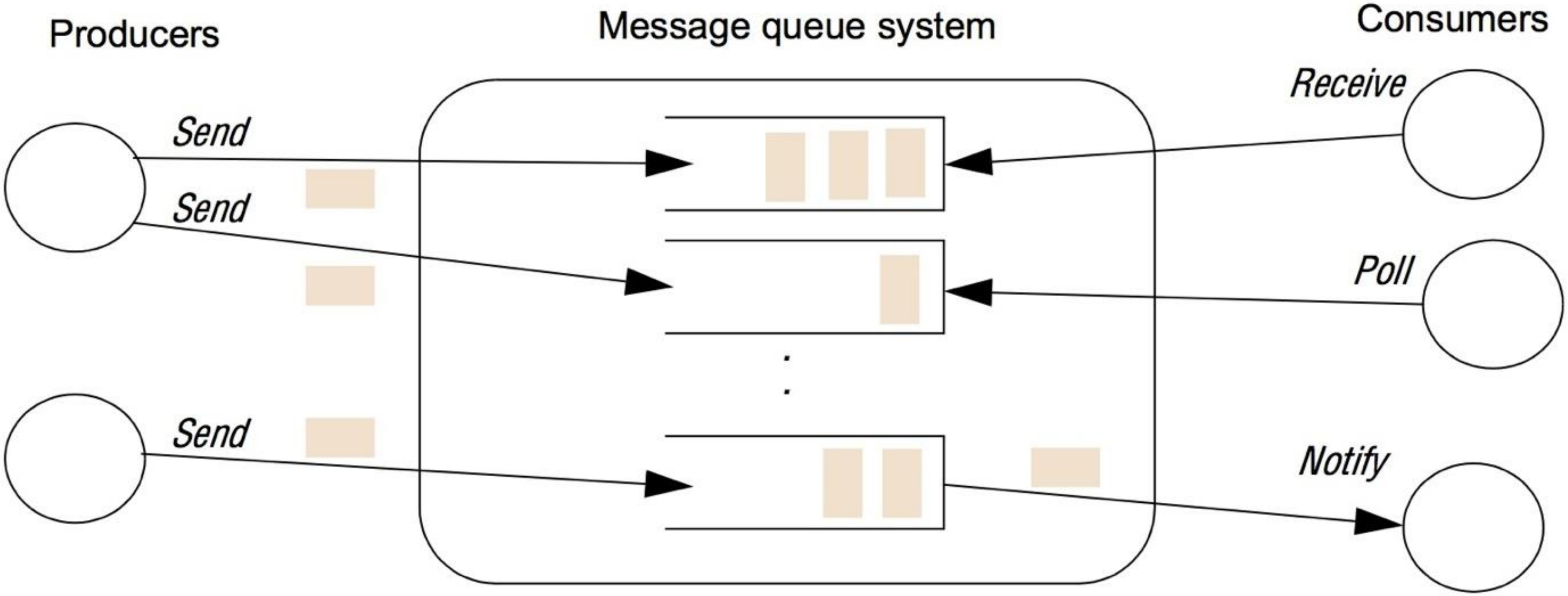
- They are point-to-point
  - the sender places the message into a queue, and it is then removed by a single process. Message queues are also referred to as Message-Oriented Middleware.

- A major class of commercial middleware with key implementations
  - IBM’s WebSphere MQ,
  - Microsoft’s MSMQ
  - Oracle’s Streams Advanced Queuing (AQ).

- The main use of such products is to achieve Enterprise Application Integration (EAI)
  - that is, integration between applications within a given enterprise – a goal that is achieved by the inherent loose coupling of message queues.

- They are also extensively used as the basis for commercial transaction processing systems because of their intrinsic support for transactions.

The message queue paradigm



Shared memory approaches: DSM

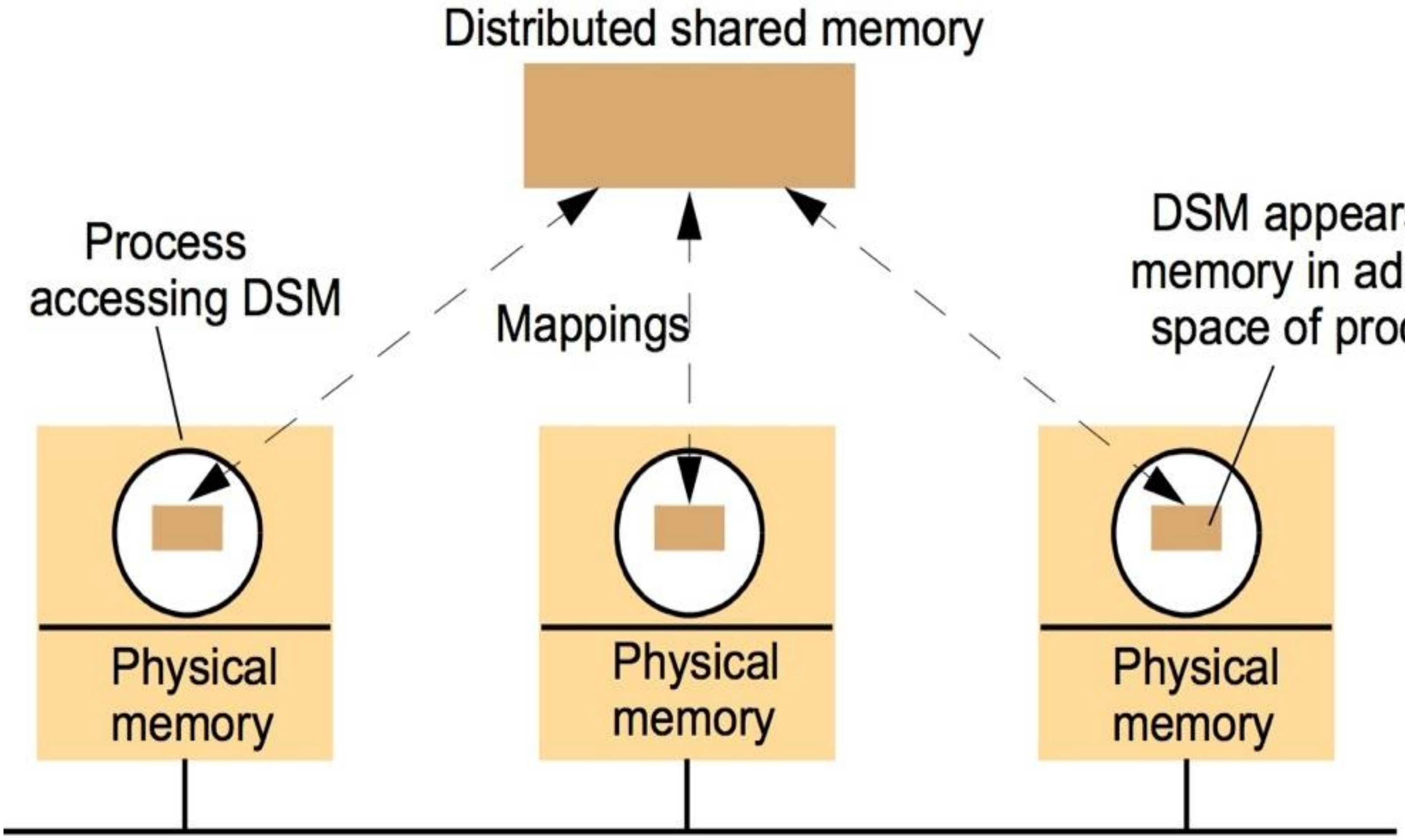
- Distributed shared memory (DSM) is an abstraction used for sharing data between computers that do not share physical memory.

- Processes access DSM by reads and updates to what appears to be ordinary memory within their address space.

- However, an underlying runtime system ensures transparently that processes executing at different computers observe the updates made by one another

- It is as though the processes access a single shared memory, but in fact the physical memory is distributed
- Examples:
  - Apollo Domain file system

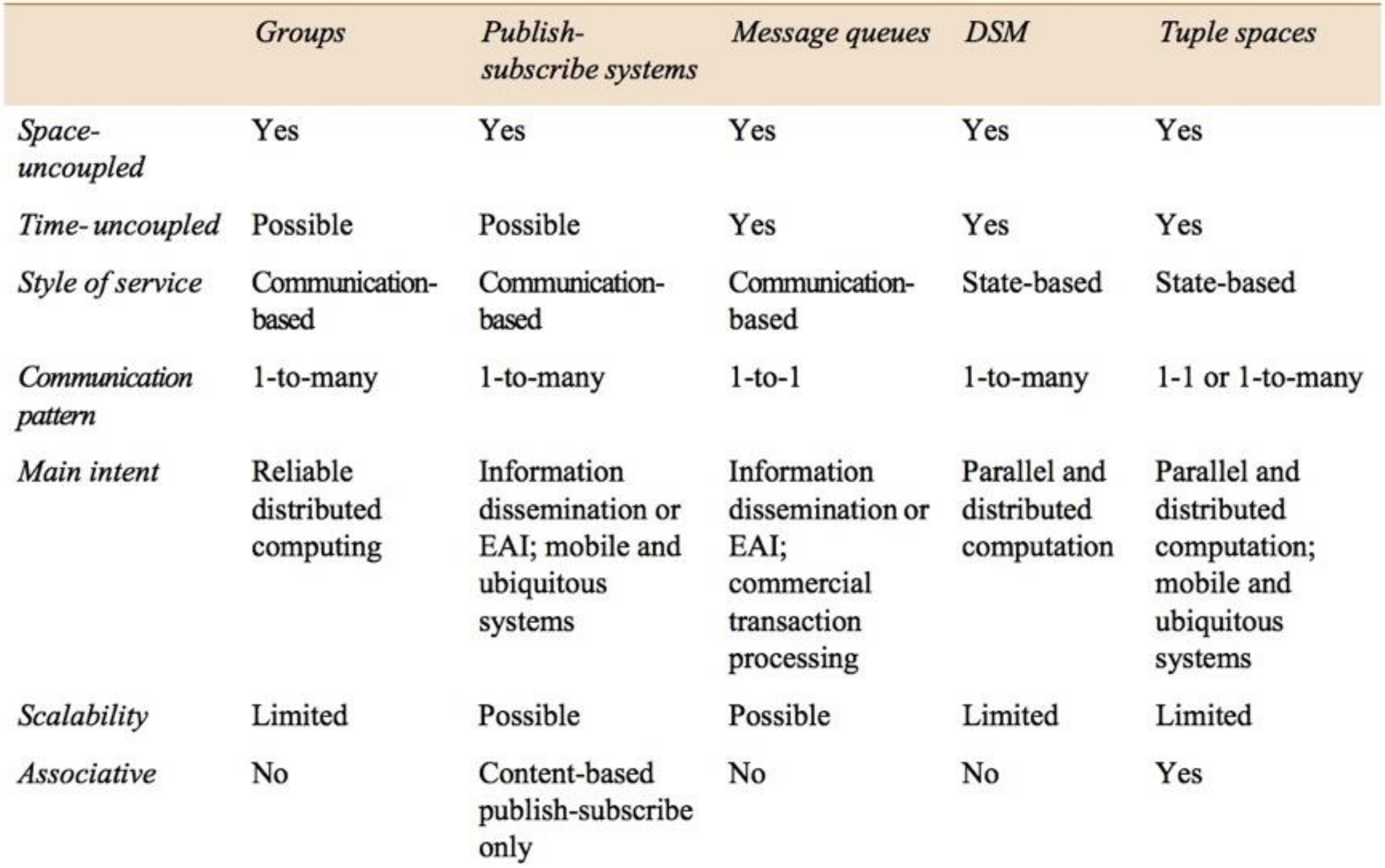
The distributed shared memory abstraction


Distributed shared memory
Process accessing DSM
Mappings
DSM appears as memory in address space of process
Physical memory
Physical memory
Physical memory

Shared memory approaches: Tuple space communication

- Processes communicate indirectly by placing tuples in a tuple space, from which other processes can read or remove them.
- Tuples do not have an address but are accessed by pattern matching on content (content-addressable memory)
- Example:
  - JavaSpaces

Summary of indirect communication styles



	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