Lecture 07 · SENG 41283 · Distributed and Cloud Computing

Indirect Communication

Space and Time Uncoupling · Group Communication · Publish–Subscribe · Message Queues · DSM · Tuple Spaces

1. Introduction

Indirect communication takes place between entities through an intermediary with no direct coupling. Two key properties: **space uncoupling** (sender need not know the identity of the receiver) and **time uncoupling** (sender and receiver need not exist at the same time).

2. Group Communication

A message sent to a group is delivered to all members. The sender is not aware of individual receiver identities. Implemented over IP multicast or overlay networks, with added value: group membership management, failure detection, reliability, and ordering guarantees.

Key Areas of Use

- Reliable dissemination of information to large numbers of clients (financial industry).
- Collaborative applications (multiuser games).
- Fault-tolerance strategies (consistent update of replicated data).
- System monitoring and management (load balancing).

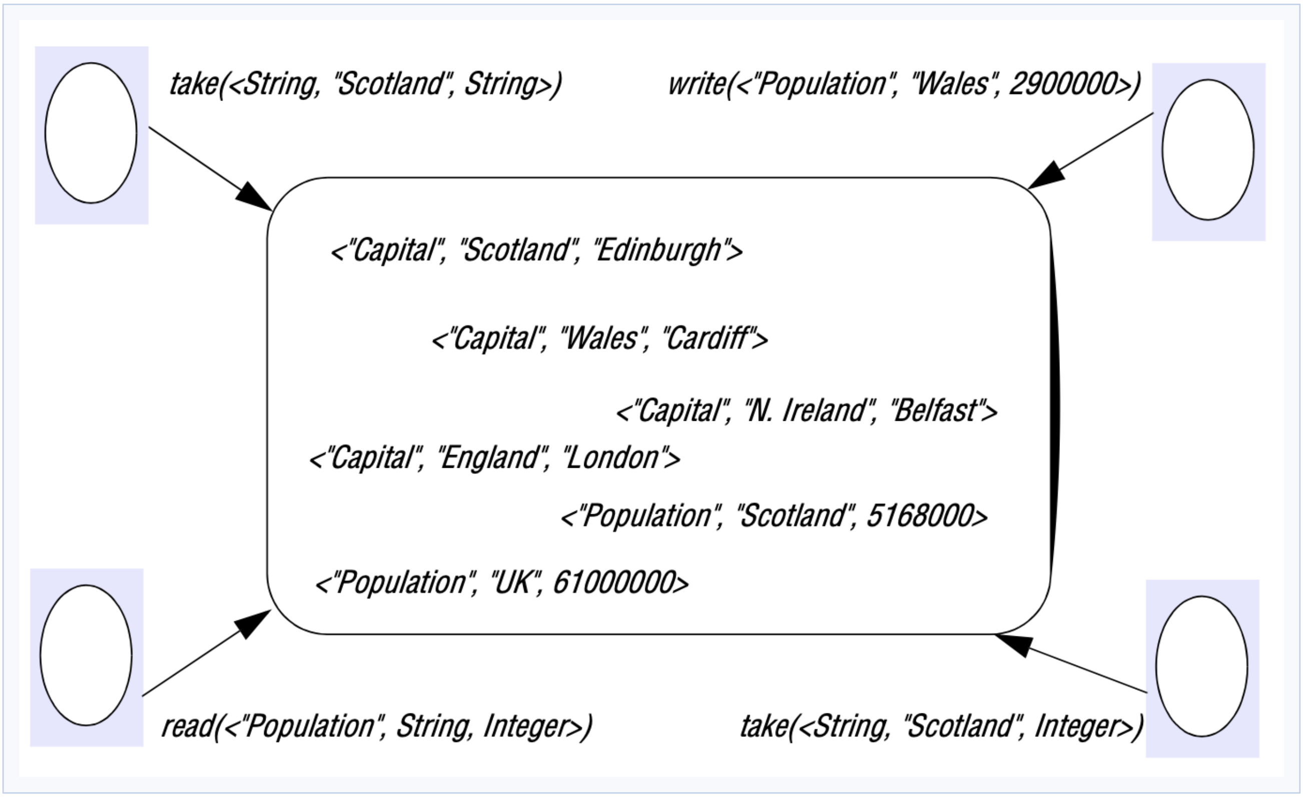

take(<String, "Scotland", String>)
write(<"Population", "Wales", 2900000>)
<"Capital", "Scotland", "Edinburgh">
<"Capital", "Wales", "Cardiff">
<"Capital", "N. Ireland", "Belfast">
<"Capital", "England", "London">
<"Population", "Scotland", 5168000>
<"Population", "UK", 61000000>
read(<"Population", String, Integer>)
take(<String, "Scotland", Integer>)

Figure 2.1 Group communication: a message sent to the group is delivered to all members.

2.1 Programming Model

Central concept: a group with associated membership; processes may join or leave. A single aGroup.send(aMessage) call multicasts to all members.

- **Closed group:** only members may multicast; useful for cooperating servers.
- **Open group:** processes outside the group may send to it.
- **Overlapping groups:** entities may be members of multiple groups (realistic in practice).

2.2 Reliability and Ordering

In addition to one-to-one reliability (integrity and validity), group communication requires **agreement**: if a message is delivered to one process, it is delivered to all processes in the group.

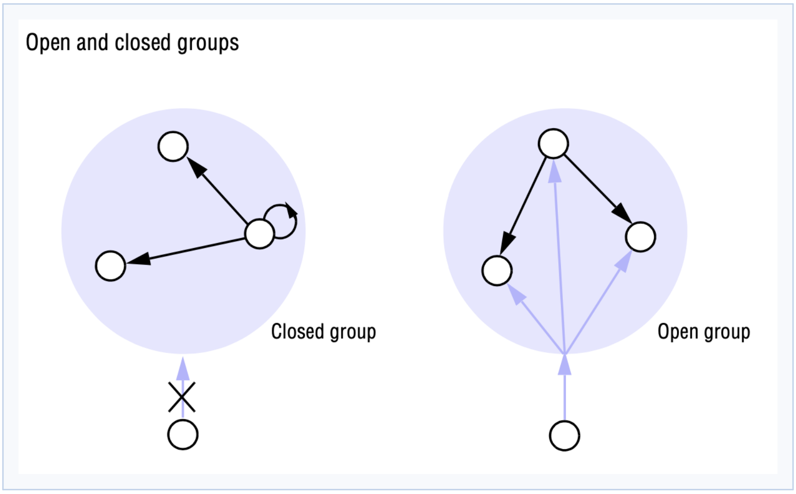


Figure 2.2 FIFO ordering: messages from Pi received at all Pj in the order they were sent.

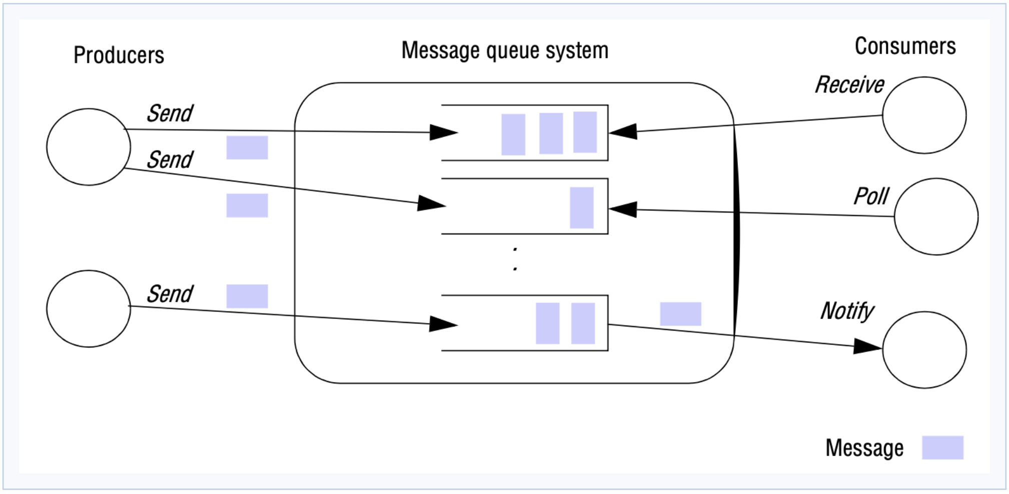
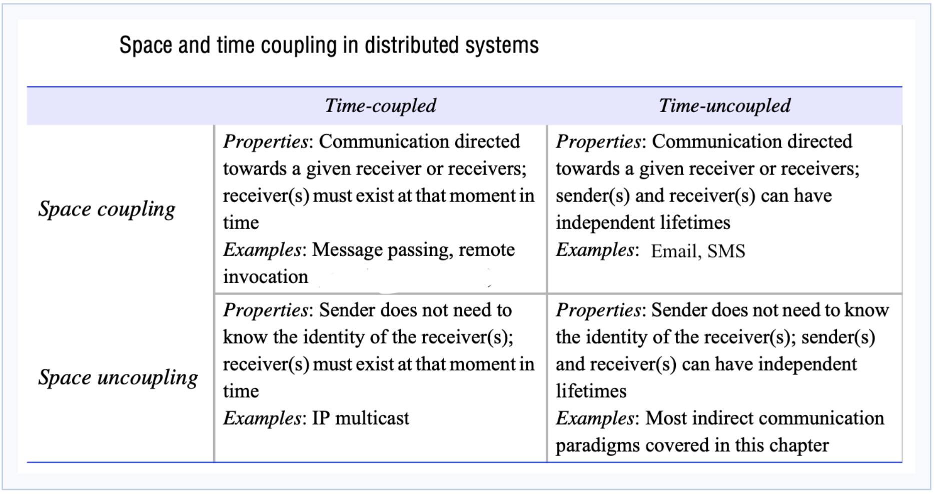


Figure 2.3 Causal ordering: captures causality chains across different senders.



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	*Properties*: Communication directed towards a given receiver or receivers; sender(s) and receiver(s) can have independent lifetimes *Examples*: Email, SMS
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	*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

Figure 2.4 Total ordering: all processes see the same FIFO sequence of messages.

Achieving Total Ordering

Central lock server: one process C is the coordinator; processes send lock messages and wait for grants. When finished, an unlock message releases the next process in the wait queue.

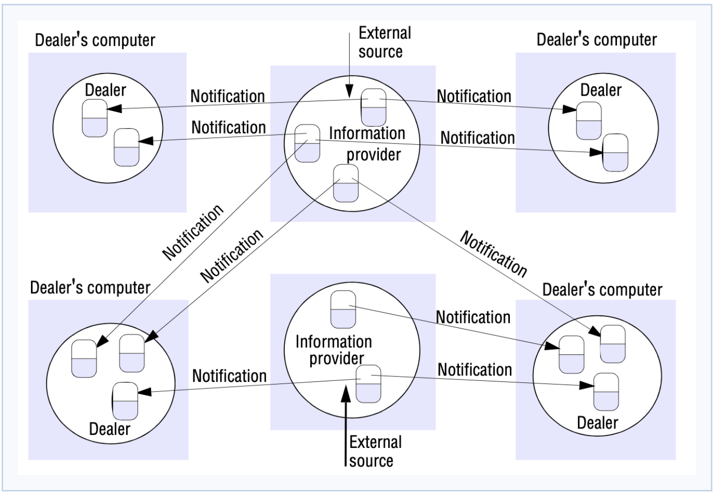


Figure 2.5 Token passing ring: single token circulates; a process may only enter the critical section when holding the token.

2.3 Group Membership Management

- **Interface for membership changes:** create/destroy groups; add/withdraw processes.
- **Failure detection:** monitors members for crashes and unreachability; marks processes Suspected or Unsuspected.
- **Notifying members:** notifies when a process is added or excluded.
- **Group address expansion:** expands group identifier to current membership list for delivery.

3. Publish–Subscribe System

Publishers publish structured events to an event service. Subscribers express interest through subscriptions (arbitrary patterns). The system matches subscriptions against events and ensures correct delivery. Fundamentally a one-to-many communications paradigm.

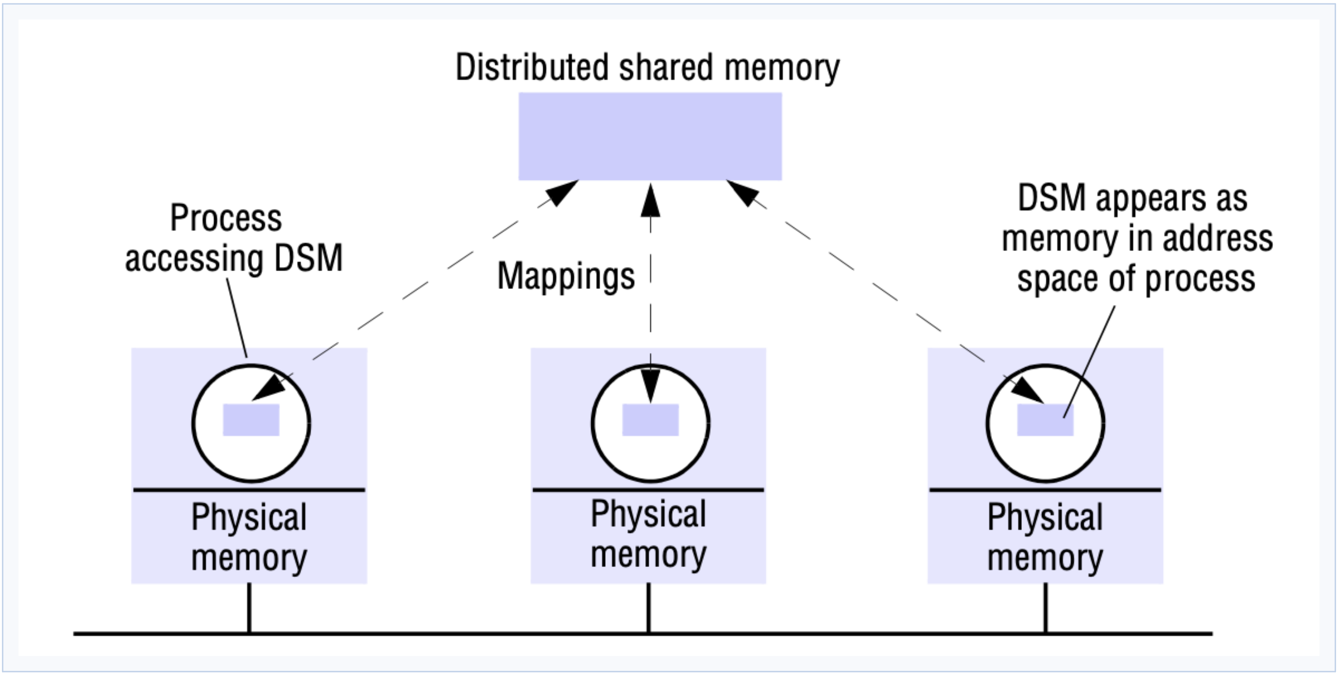

Distributed shared memory
Process accessing DSM
Mappings
DSM appears as memory in address space of process
Physical memory
Physical memory
Physical memory

Figure 3.1 Publish–subscribe system: publishers, event service, and subscribers with filter-based matching.

Subscription Models

- **Channel-based:** subscribe to a named channel; receive all events sent to that channel.
- **Topic-based:** subscriptions defined in terms of the topic of interest.
- **Content-based:** query defined over compositions of constraints on event attribute values.
- **Type-based:** subscriptions defined in terms of types or subtypes of events.

Implementation

Centralised: single event broker; simple but single point of failure and performance bottleneck. **Distributed:** network of cooperating brokers; can survive node failure; scales to Internet.

Routing Strategies

- **Flooding:** send event notification to all nodes; match at subscriber end.
- **Filtering-based routing:** brokers forward only where a path to a valid subscriber exists.
- **Rendezvous:** partition the event space among brokers; rendezvous nodes responsible for a subset.
- **Informed gossip:** nodes periodically and probabilistically exchange events with neighbours.

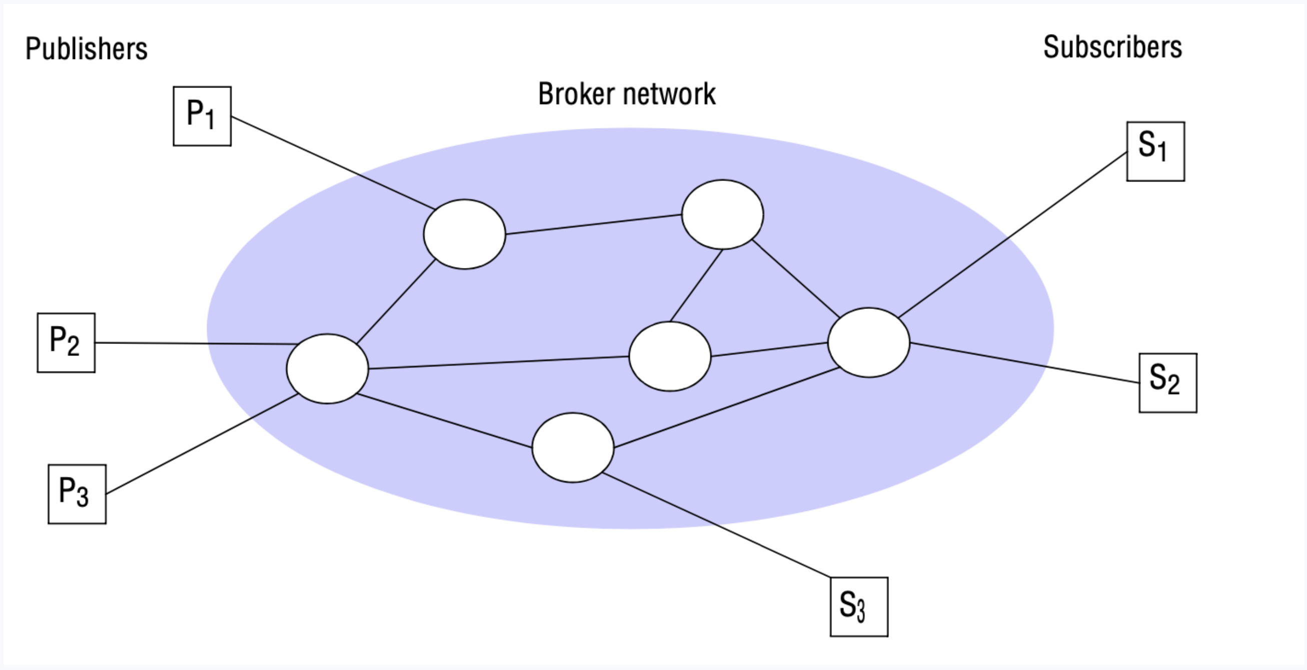


Figure 3.2 Publish–subscribe routing: distributed broker network with filtering-based event routing.

4. Message Queues

Point-to-point service using a queue as indirection, achieving space and time uncoupling. The sender places a message in the queue; a single consumer removes it.

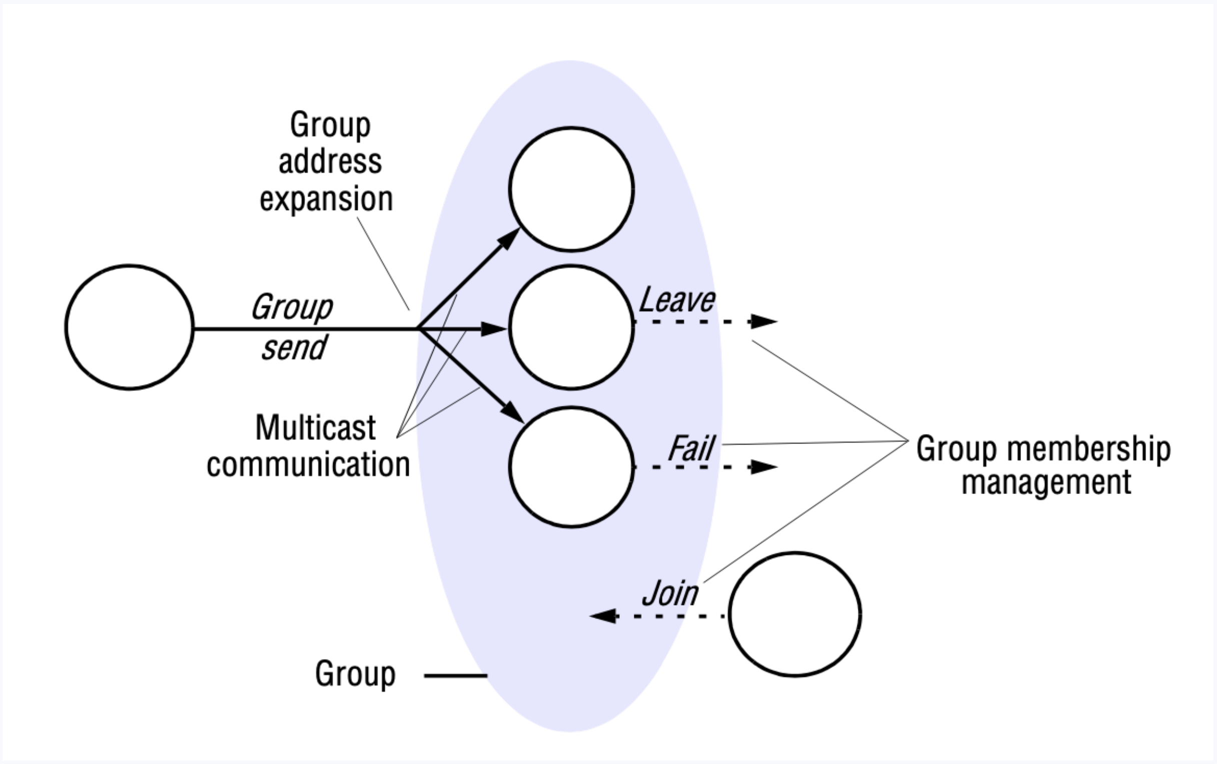


Figure 4.1 Message queue: producer sends to queue; consumer receives with blocking, polling, or notify.

- **Blocking receive:** blocks until a message is available.
- **Non-blocking receive (polling):** checks queue status and returns immediately.
- **Notify operation:** issues an event notification when a message becomes available.

Queuing policy: normally FIFO; most implementations also support priority. Consumers can select messages based on their properties.

5. Distributed Shared Memory (DSM)

An abstraction for sharing data between computers without shared physical memory. Processes access DSM by reads and writes to what appears to be ordinary memory in their address space. The underlying runtime ensures transparently that processes at different computers observe each other's updates.

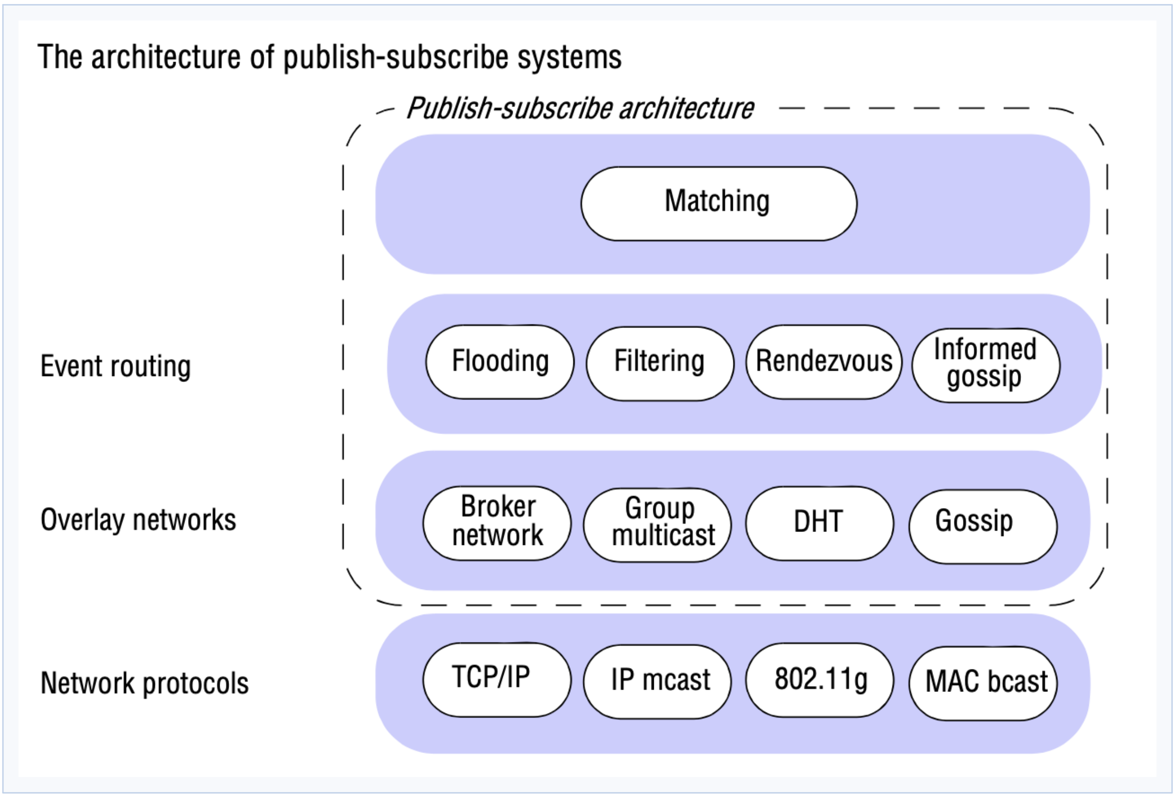


Figure 5.1 Distributed Shared Memory: processes at different computers share a virtual address space.

6. Tuple Space Communication

Processes communicate indirectly by placing tuples in a tuple space; others read or remove them by pattern matching on content.

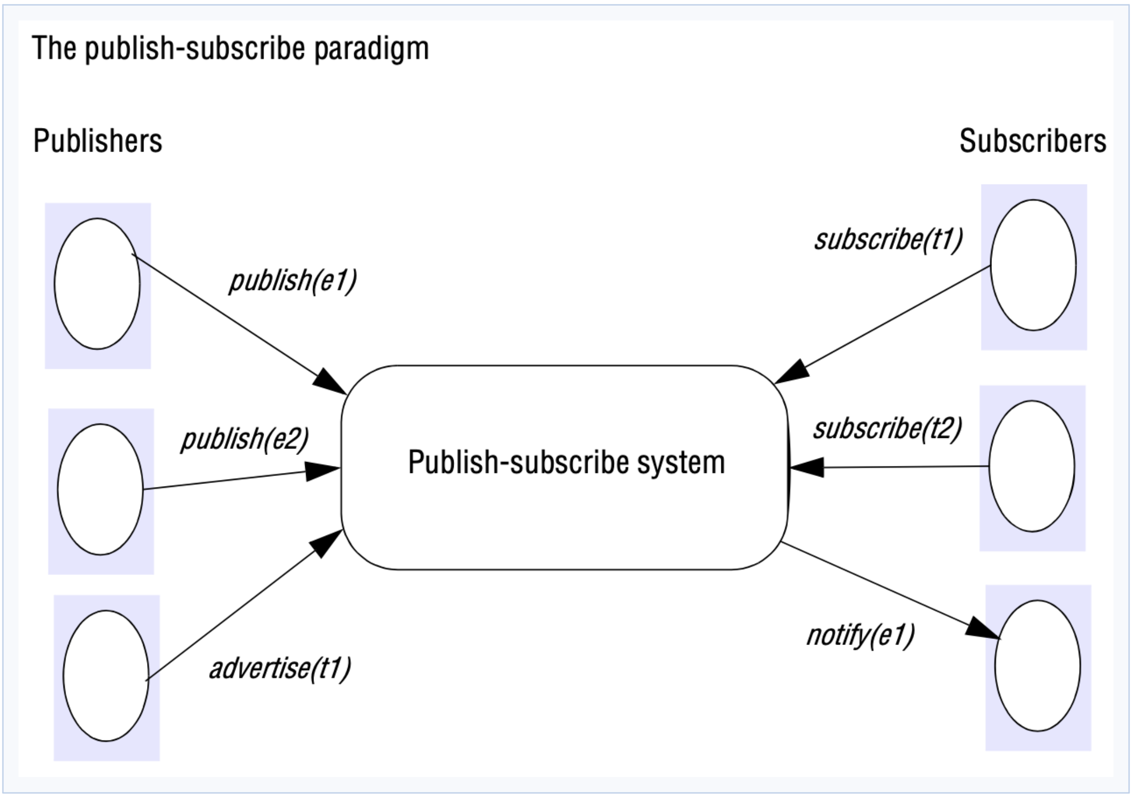


Figure 6.1 Tuple space: processes write, read, and take tuples; accessed by pattern matching.

- **write:** adds a tuple without affecting existing tuples.
- **read:** returns a matching tuple without removing it.
- **take:** returns a matching tuple and removes it from the tuple space.

Assignment

Try out the RabbitMQ Work Queue example in any preferred language and write a Medium blog article explaining the important concepts of message queues. Reference: https://www.rabbitmq.com/getstarted.html