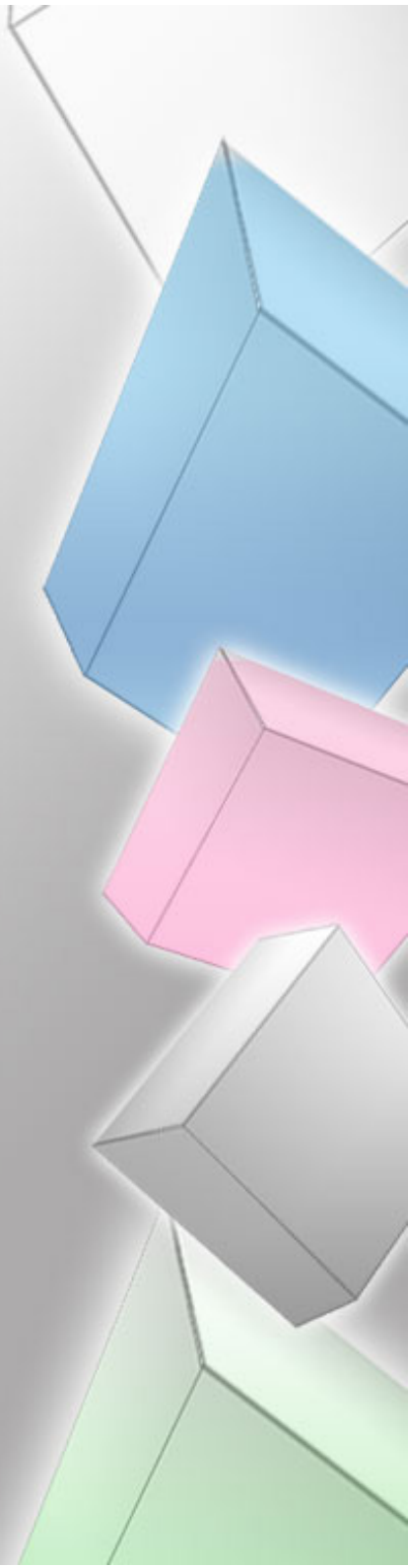
Advanced Topics in Operating Systems

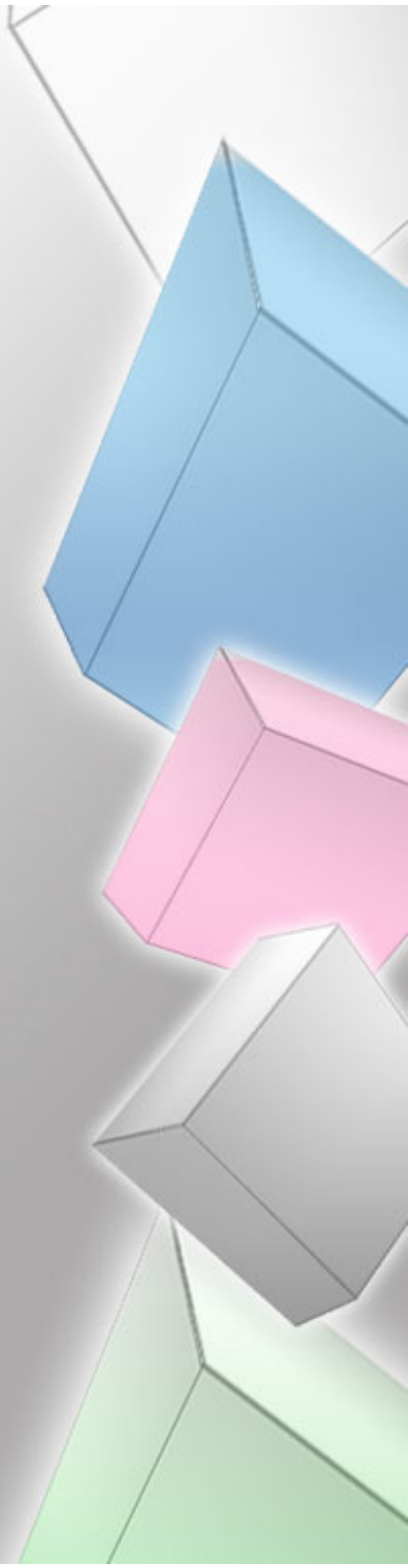
MSc in Computer Science
UNYT-UoG

Dr. Marenglen Biba
8-9-10 January 2010



Lesson 13

01: Introduction
02: Architectures
03: Processes
04: Communication
05: Naming
06: Synchronization
07: Consistency & Replication
08: Fault Tolerance
09: Security
10: Distributed Object-Based Systems
11: Distributed File Systems
12: Distributed Web-Based Systems
13: Distributed Coordination-Based Systems



Introduction To Coordination Models

Referential	Temporal: Coupled	Temporal: Decoupled
Coupled	Direct	Mailbox
Decoupled	Meeting oriented	Generative communication

Figure 13-1. A taxonomy of coordination models (adapted from Cabri et al., 2000).

Meeting-based systems

- Meeting-based systems are often implemented by means of events, like those supported by object-based systems.
- Another mechanism for implementing meetings is publish/subscribe systems.
  - In these systems, processes subscribe to messages containing information about specific subjects, while other process produce (publish) such messages.
    - Temporal coupling, but processes remain anonymous.

Generative Communication

- The most widely-known coordination model.
- The key idea is that a collection of independent processes make use of a shared persistent dataspace of **tuples**.
  - **Tuples** are tagged data records consisting of a number of typed fields.
- Processes can put any type of record into the shared dataspace (i.e., they generate communication records).
- An interesting feature of shared dataspaces is that they implement an associative search mechanism for tuples.
  - When a process wants to extract a tuple from the dataspace, it essentially specifies (some of) the values of the fields it is interested in. Any tuple that matches that specification is then removed from the dataspace and passed to the process.

Overall Approach

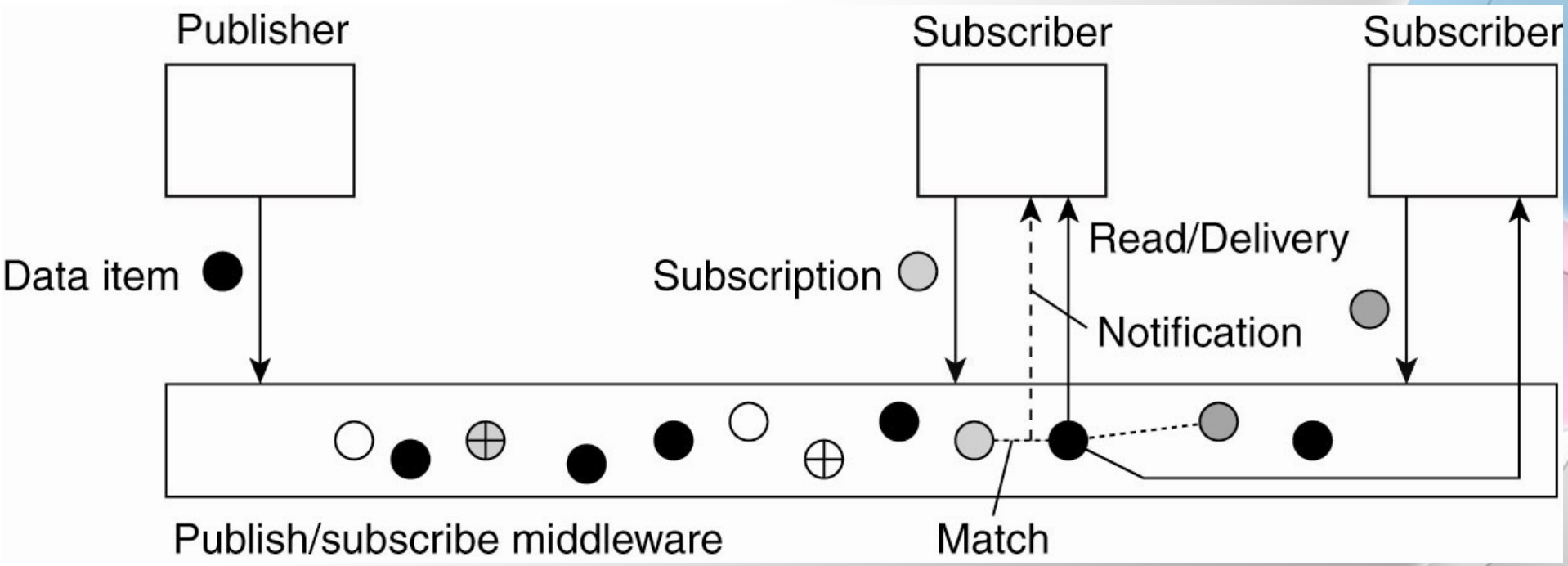


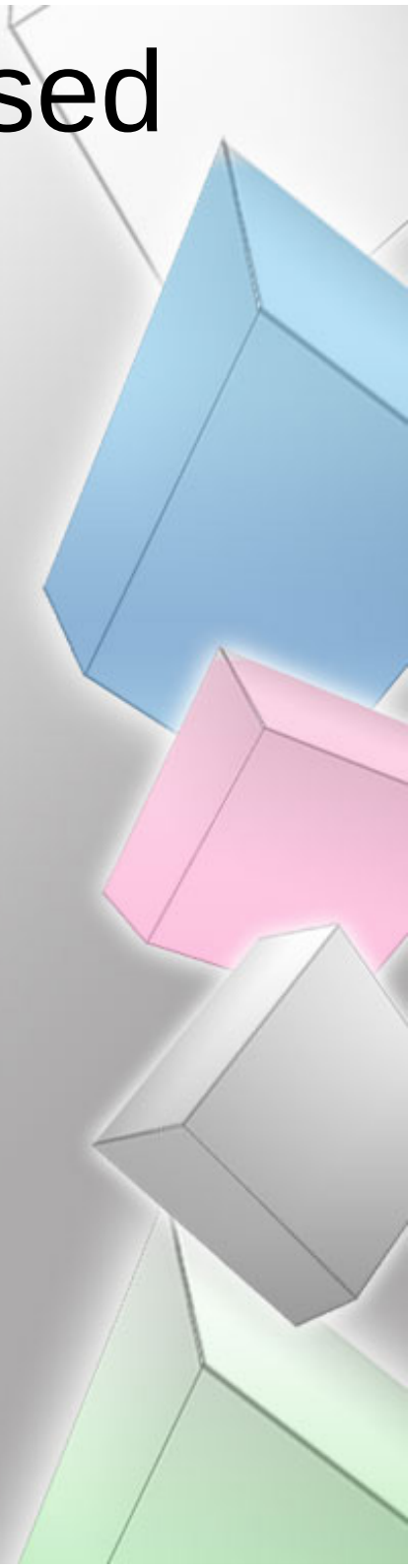
Figure 13-2. The principle of exchanging data items between publishers and subscribers.

Events

- Events complicate the processing of subscriptions.
- Consider the subscription:
  - “Notify when room R4.20 is unoccupied and the door is unlocked”.
  - A distributed system supporting these subscriptions could be implemented by placing sensors.
- We would need to compose such primitive events into a publishable data item to which processes can then subscribe.
  - Event composition is hard!

Distributed Coordination-based Systems

- Coordination Models
- Architectures
- Communication
- Naming
- Consistency and Replication
- Fault Tolerance
- Security



Example: Jini and JavaSpaces

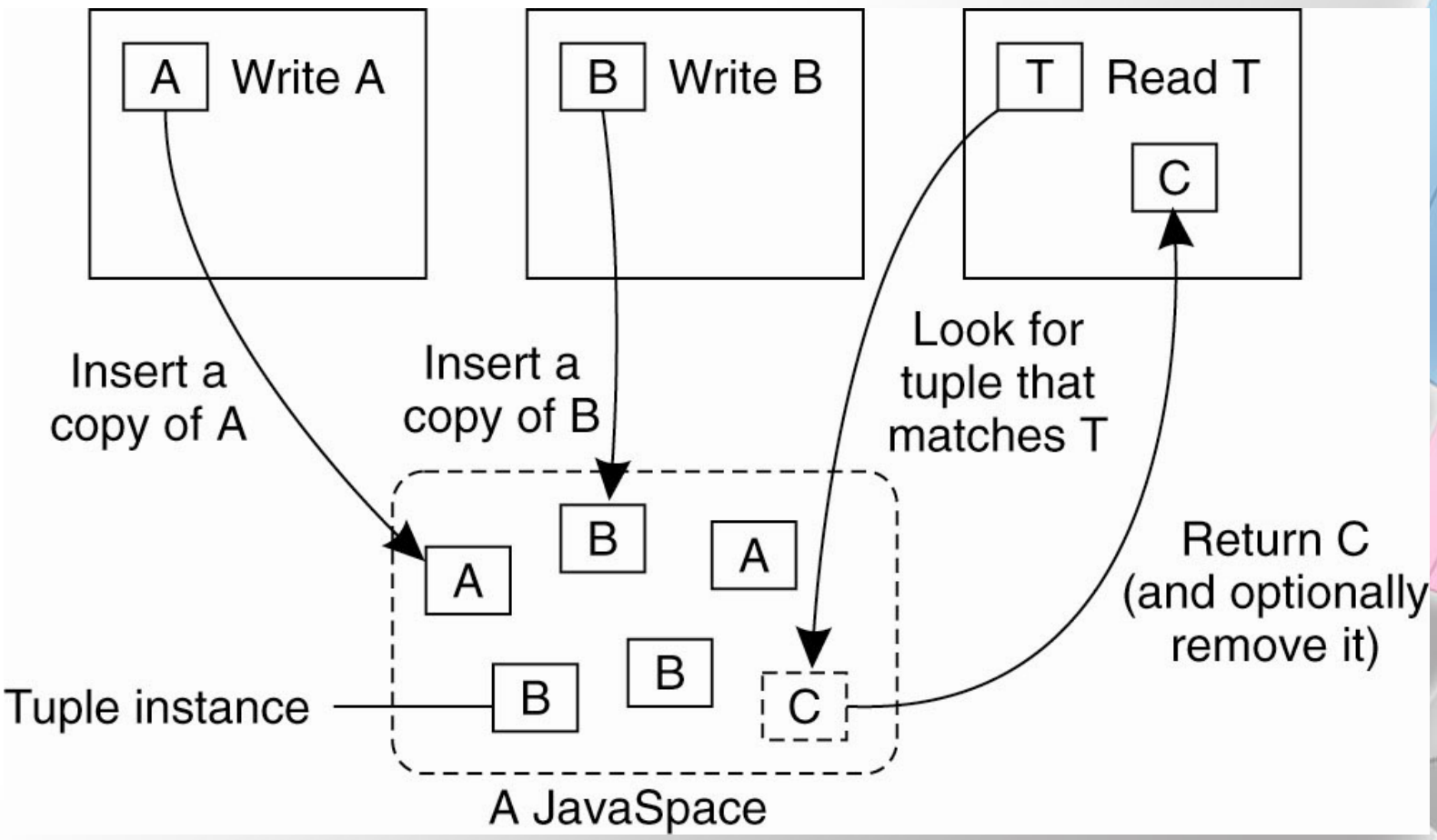


Figure 13-3. The general organization of a JavaSpace in Jini.

Example: TIB/Rendezvous

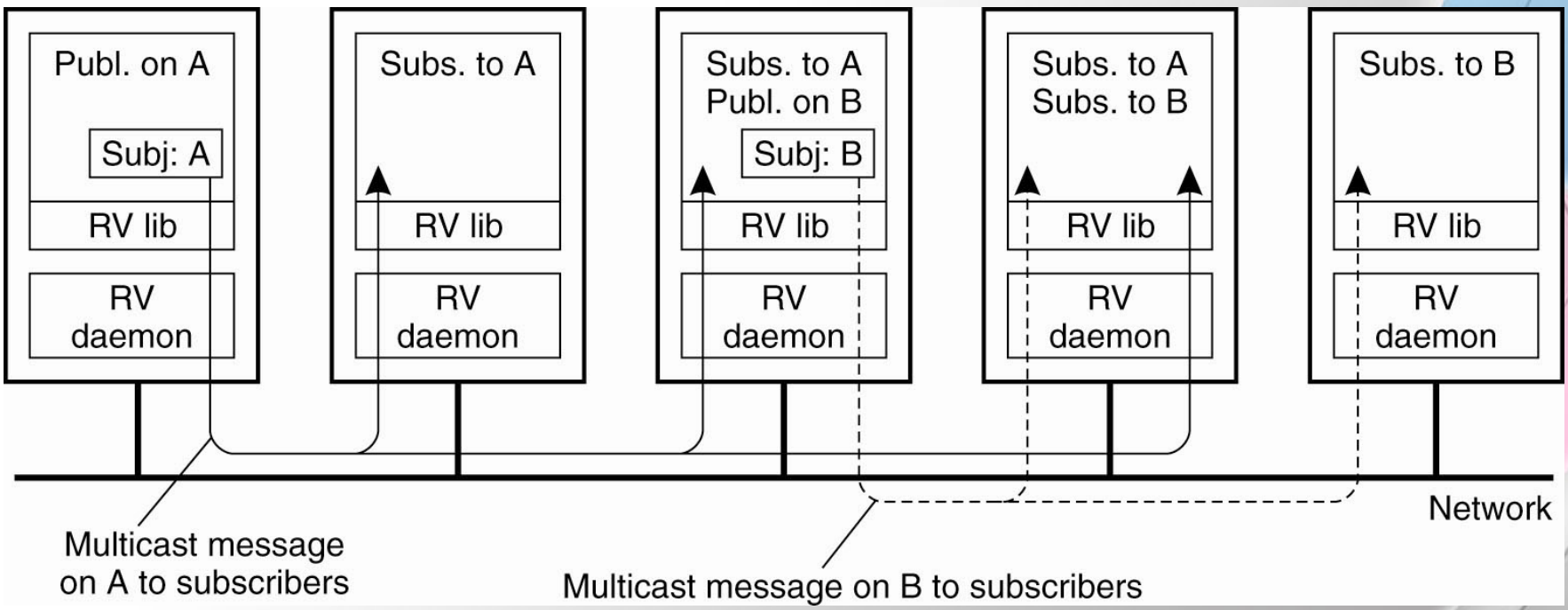


Figure 13-4. The principle of a publish/subscribe system as implemented in TIB/Rendezvous.

Mobility and Coordination

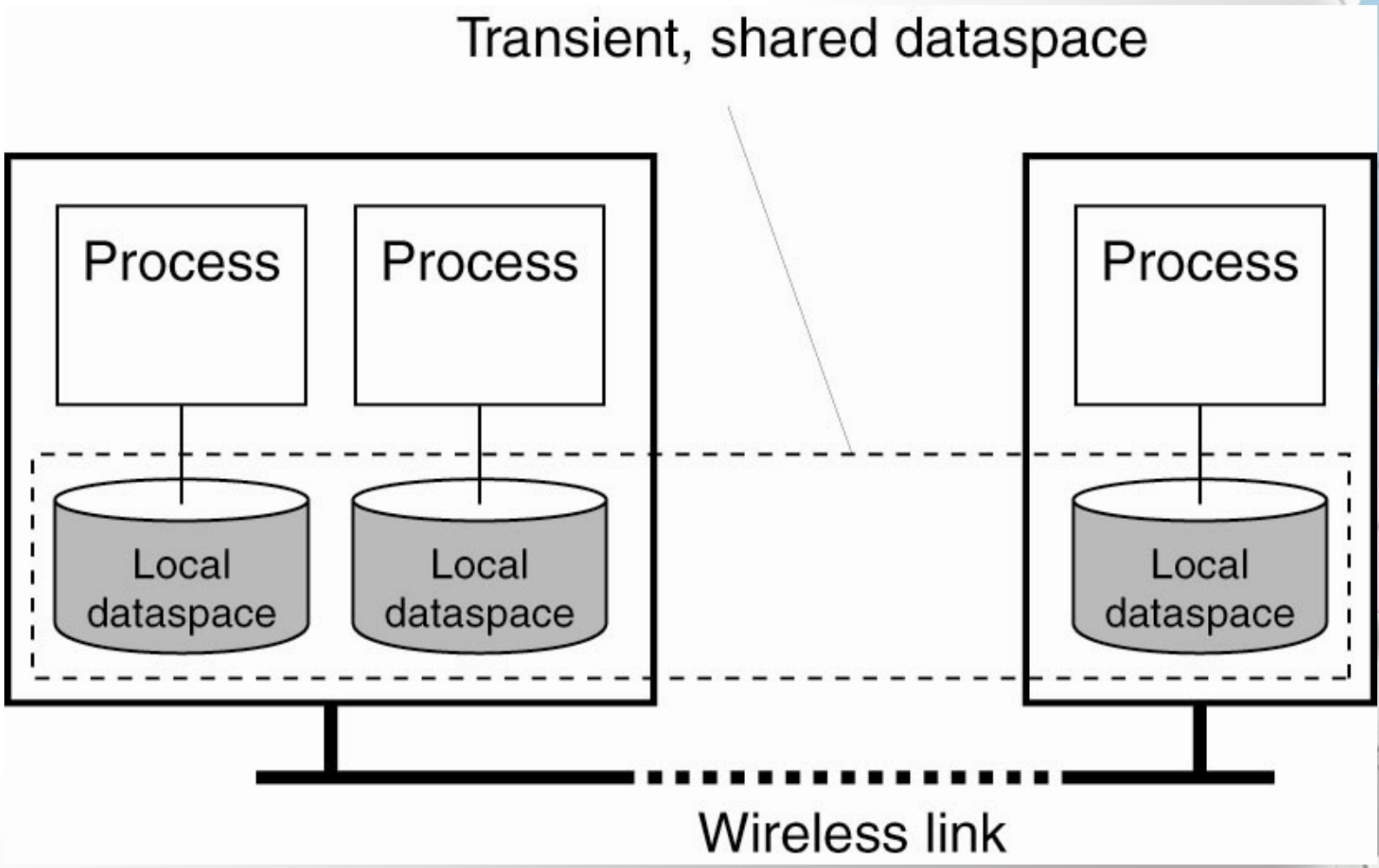
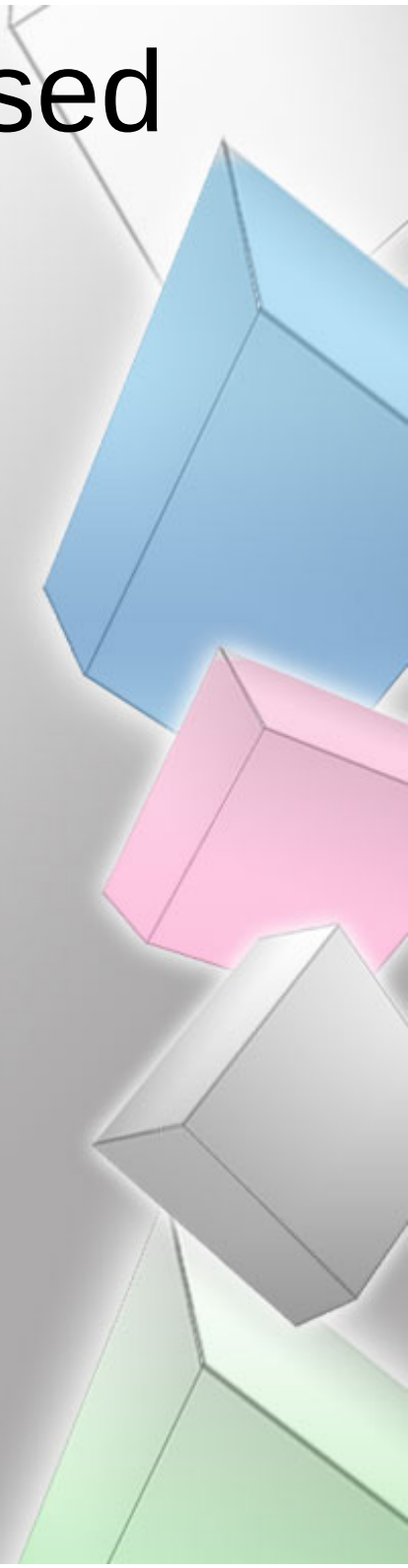


Figure 13-6. Transient sharing of local dataspaces in Lime.

Distributed Coordination-based Systems

- Coordination Models
- Architectures
- Communication
- Naming
- Consistency and Replication
- Fault Tolerance
- Security



Content-Based Routing (1)

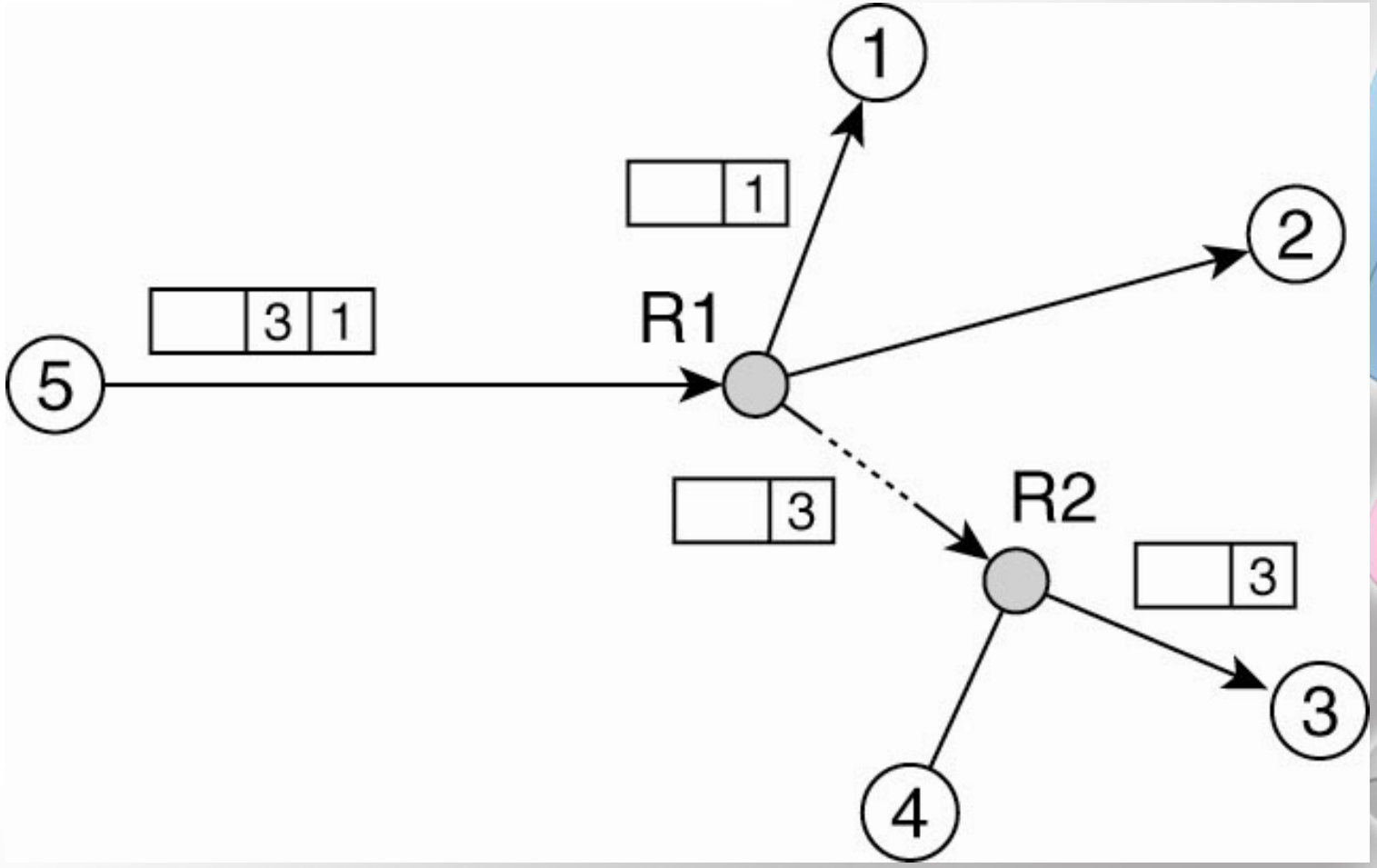


Figure 13-7. Naive content-based routing.

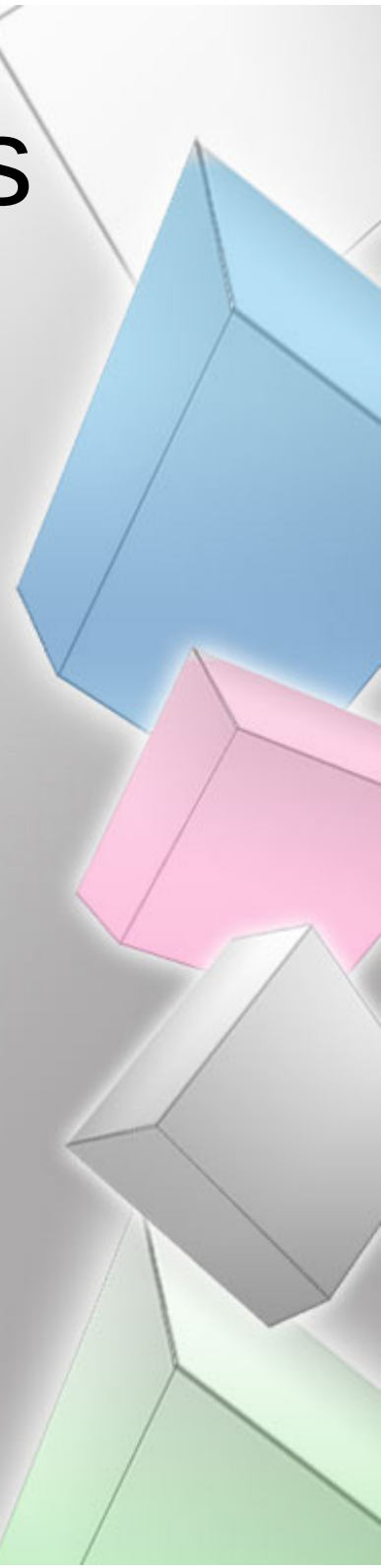
Content-Based Routing (1)

Interface	Filter
To node 3	$a \in [0,3]$
To node 4	$a \in [2,5]$
Toward router R_1	(unspecified)

Figure 13-8. A partially filled routing table.

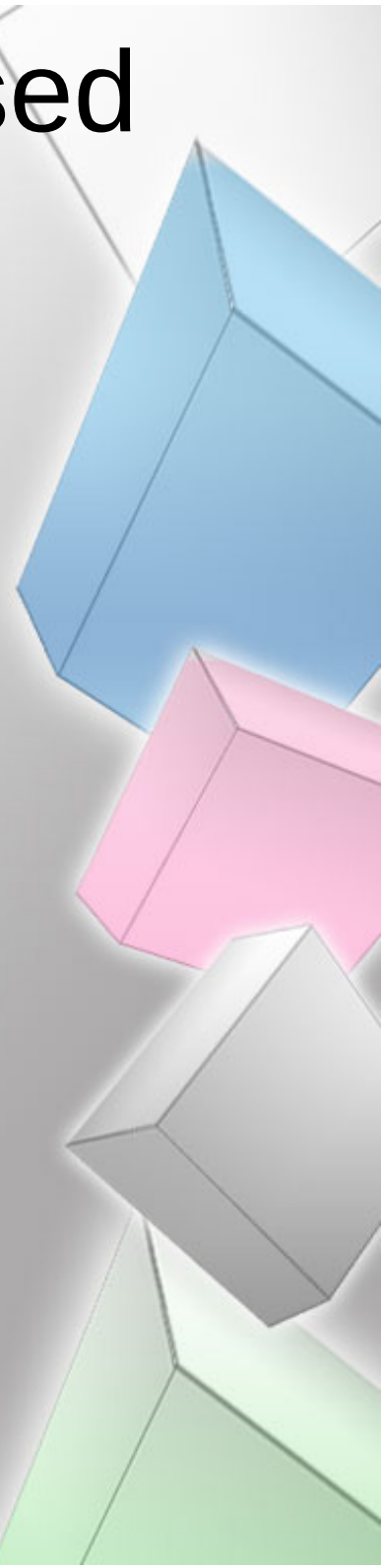
Composite subscriptions

- Example: Reading stocks
  - http://ams.tibco.com



Distributed Coordination-based Systems

- Coordination Models
- Architectures
- Communication
- Naming
- Consistency and Replication
- Fault Tolerance
- Security



Describing Composite Events (1)

Ex.	Description
S1	Notify when room R4.20 is unoccupied
S2	Notify when R4.20 is unoccupied and the door is unlocked
S3	Notify when R4.20 is unoccupied for 10 seconds while the door is unlocked
S4	Notify when the temperature in R4.20 rises more than 1 degree per 30 minutes
S5	Notify when the average temperature in R4.20 is more than 20 degrees in the past 30 minutes

Figure 13-9. Examples of events in a distributed system.

Describing Composite Events (2)

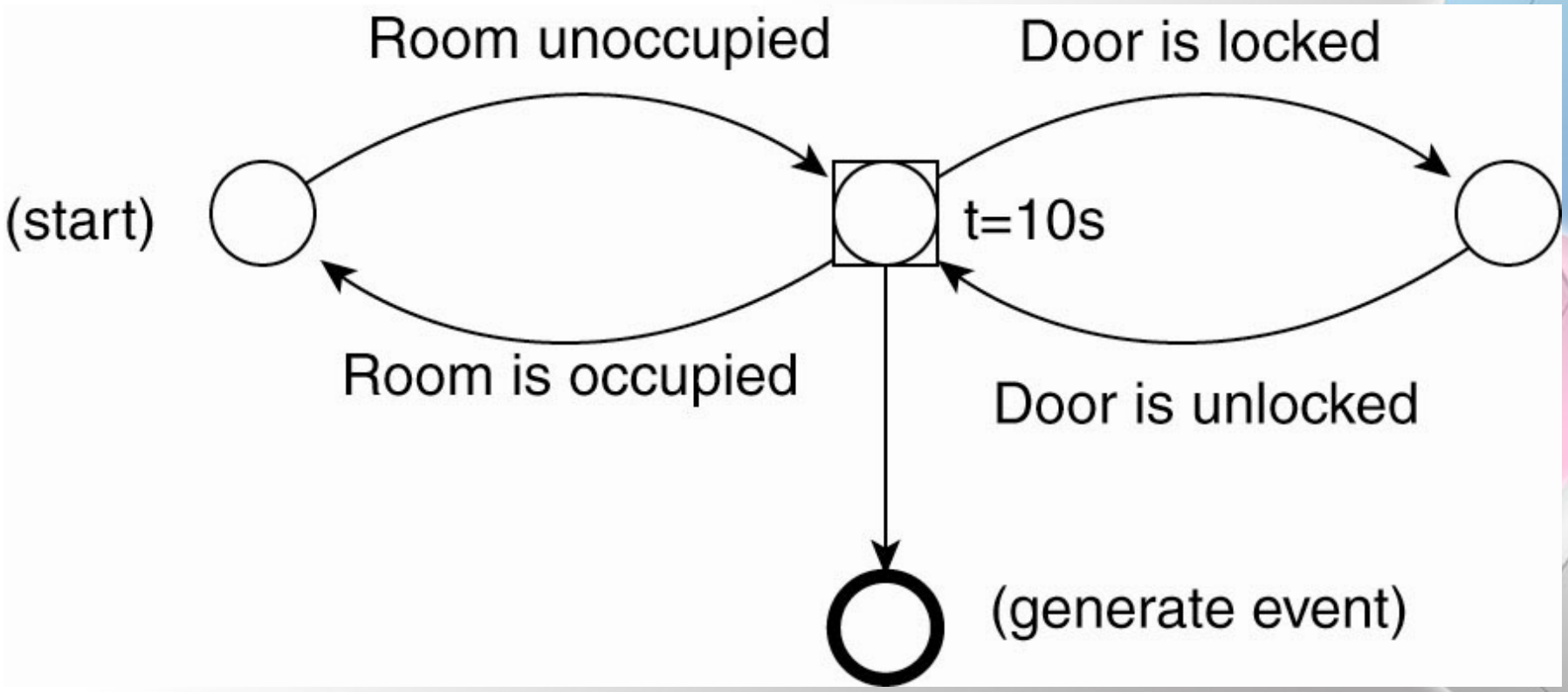


Figure 13-10. The finite state machine for subscription S3

Describing Composite Events (3)

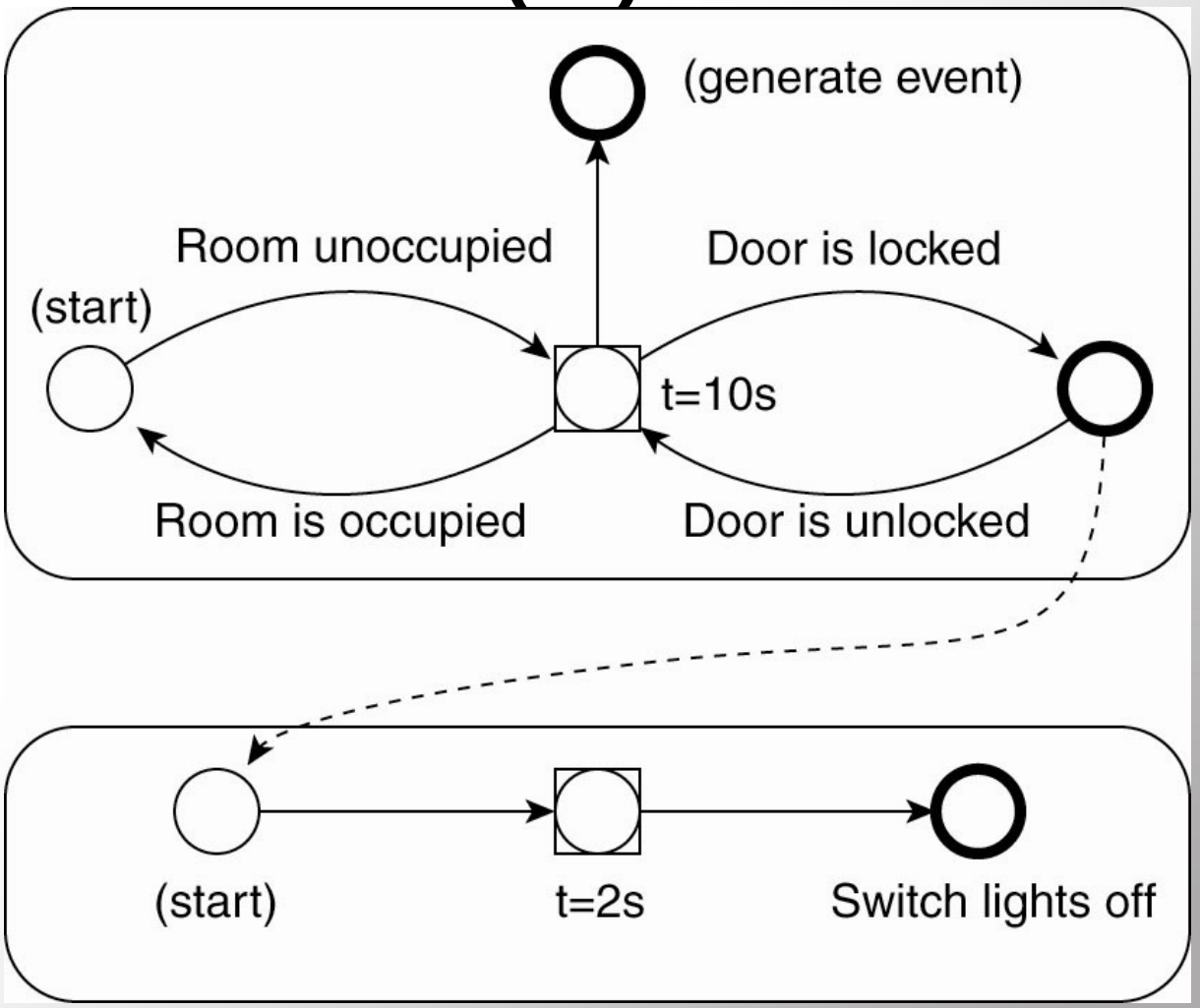
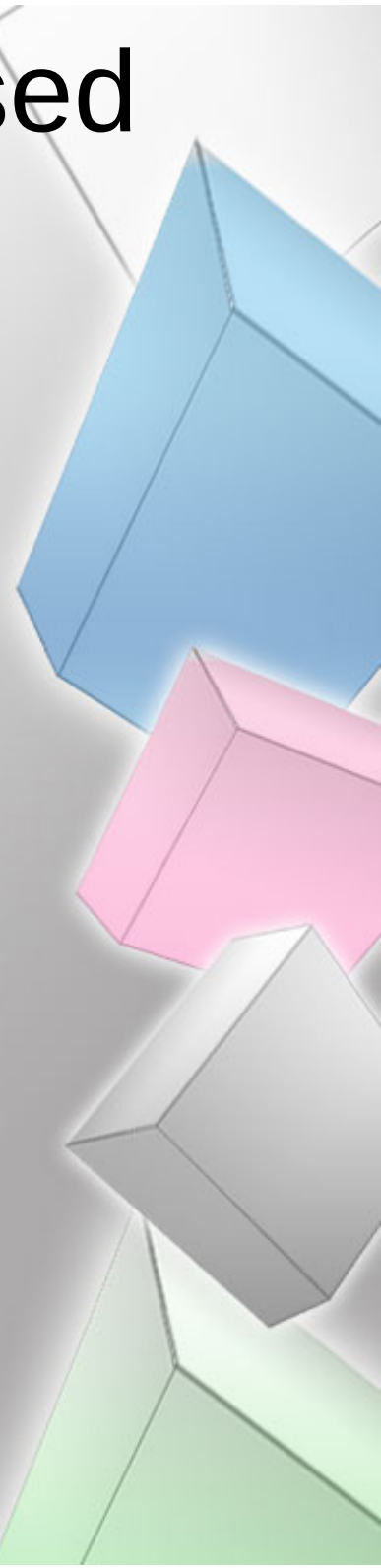


Figure 13-11. Two coupled FSMs.

Distributed Coordination-based Systems

- Coordination Models
- Architectures
- Communication
- Naming
- Consistency and Replication
- Fault Tolerance
- Security



General Considerations to Static Approaches (1)

An efficient distributed implementation of a JavaSpace has to solve two problems:

1. How to simulate associative addressing without massive searching.
2. How to distribute tuple instances among machines and locate them later.

General Considerations to Static Approaches (2)

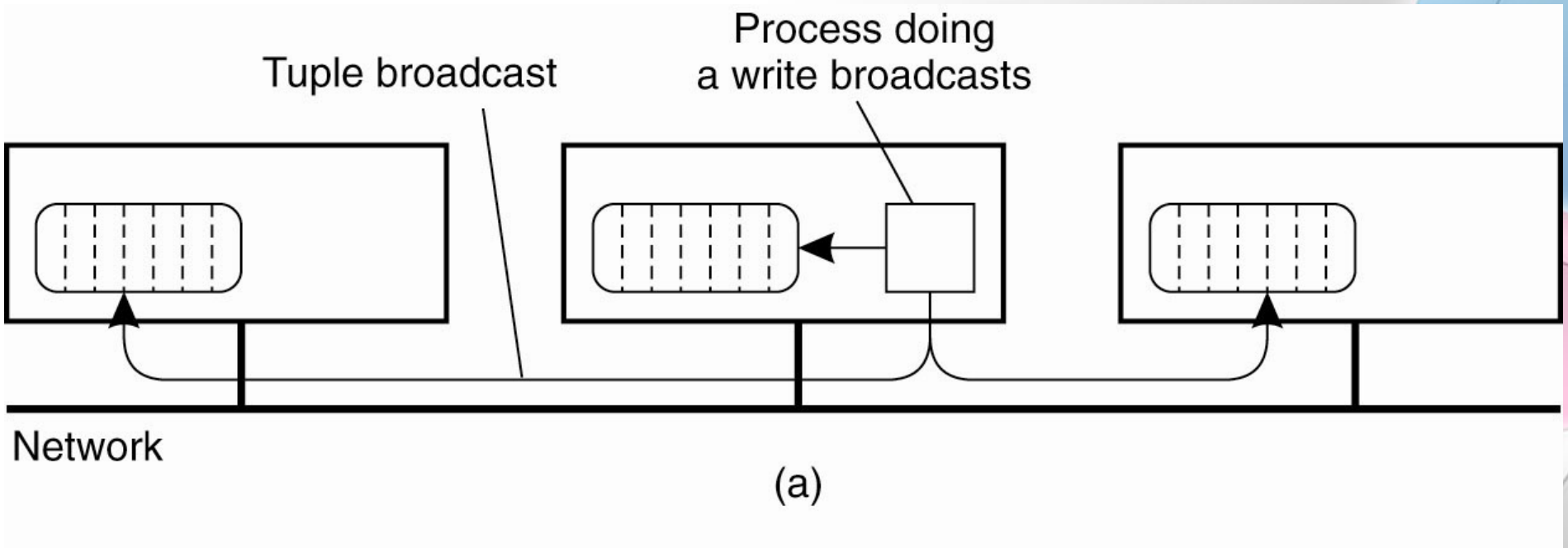


Figure 13-12. A JavaSpace can be replicated on all machines. The dotted lines show the partitioning of the JavaSpace into subspaces. (a) Tuples are broadcast on write.

General Considerations to Static Approaches (3)

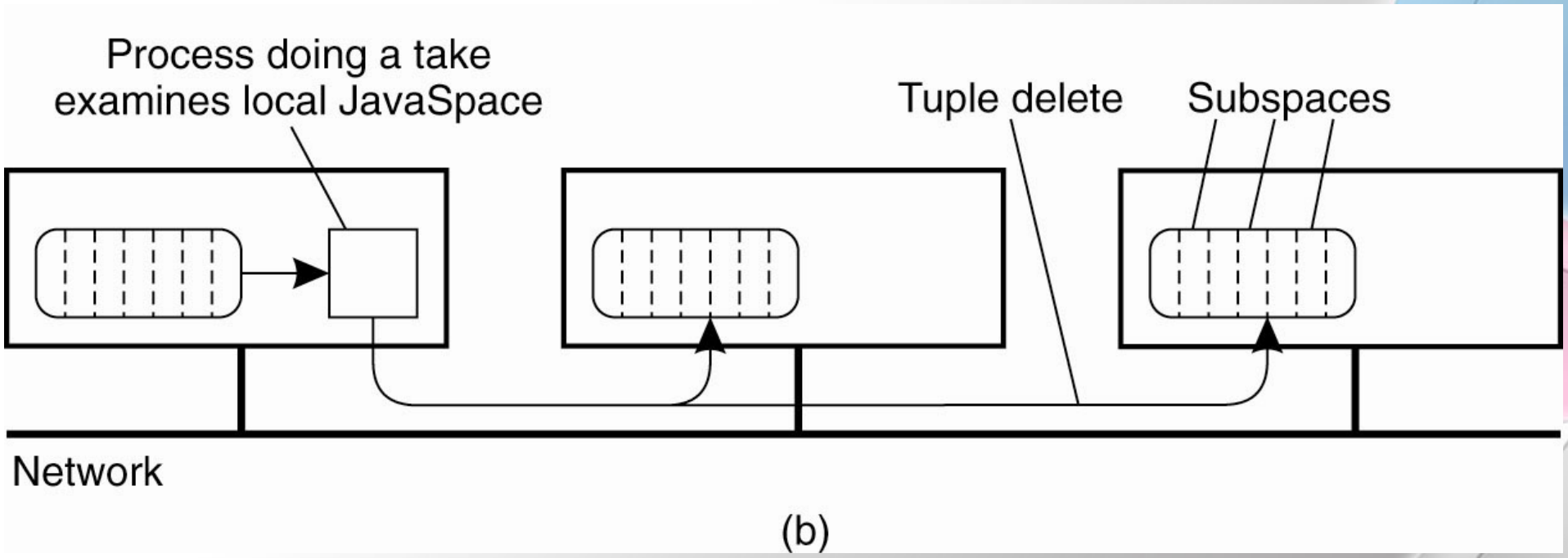


Figure 13-12. A JavaSpace can be replicated on all machines. The dotted lines show the partitioning of the JavaSpace into subspaces. (b) reads are local, but the removing an instance when calling take must be broadcast.

General Considerations to Static Approaches (4)

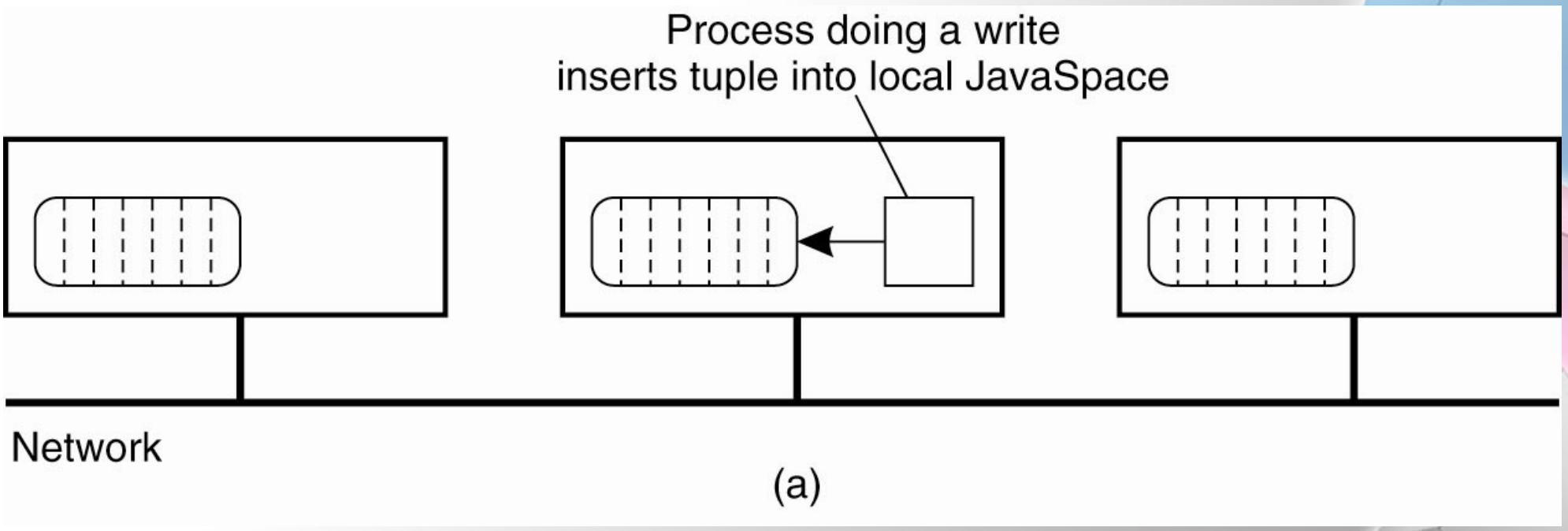


Figure 13-13. Nonreplicated JavaSpace. (a) A write is done locally.

General Considerations to Static Approaches (5)

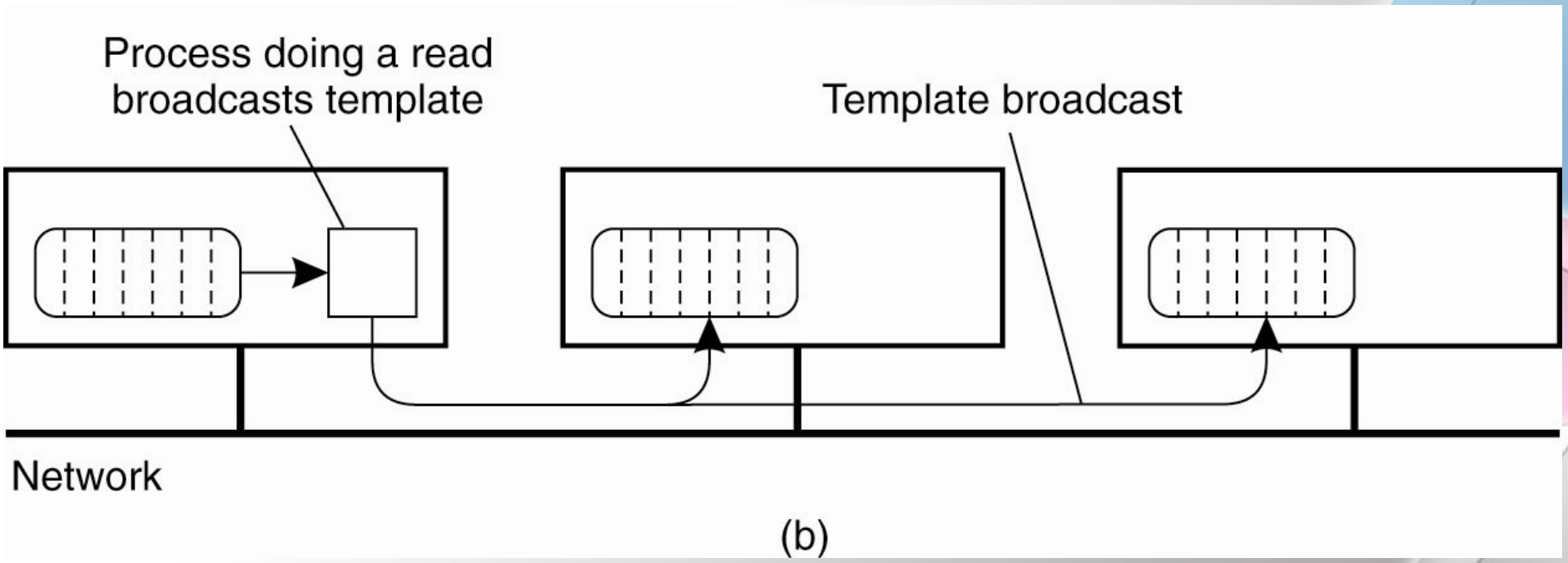


Figure 13-13. Nonreplicated JavaSpace. (b) A read or take requires the template tuple to be broadcast in order to find a tuple instance.

General Considerations to Static Approaches (6)

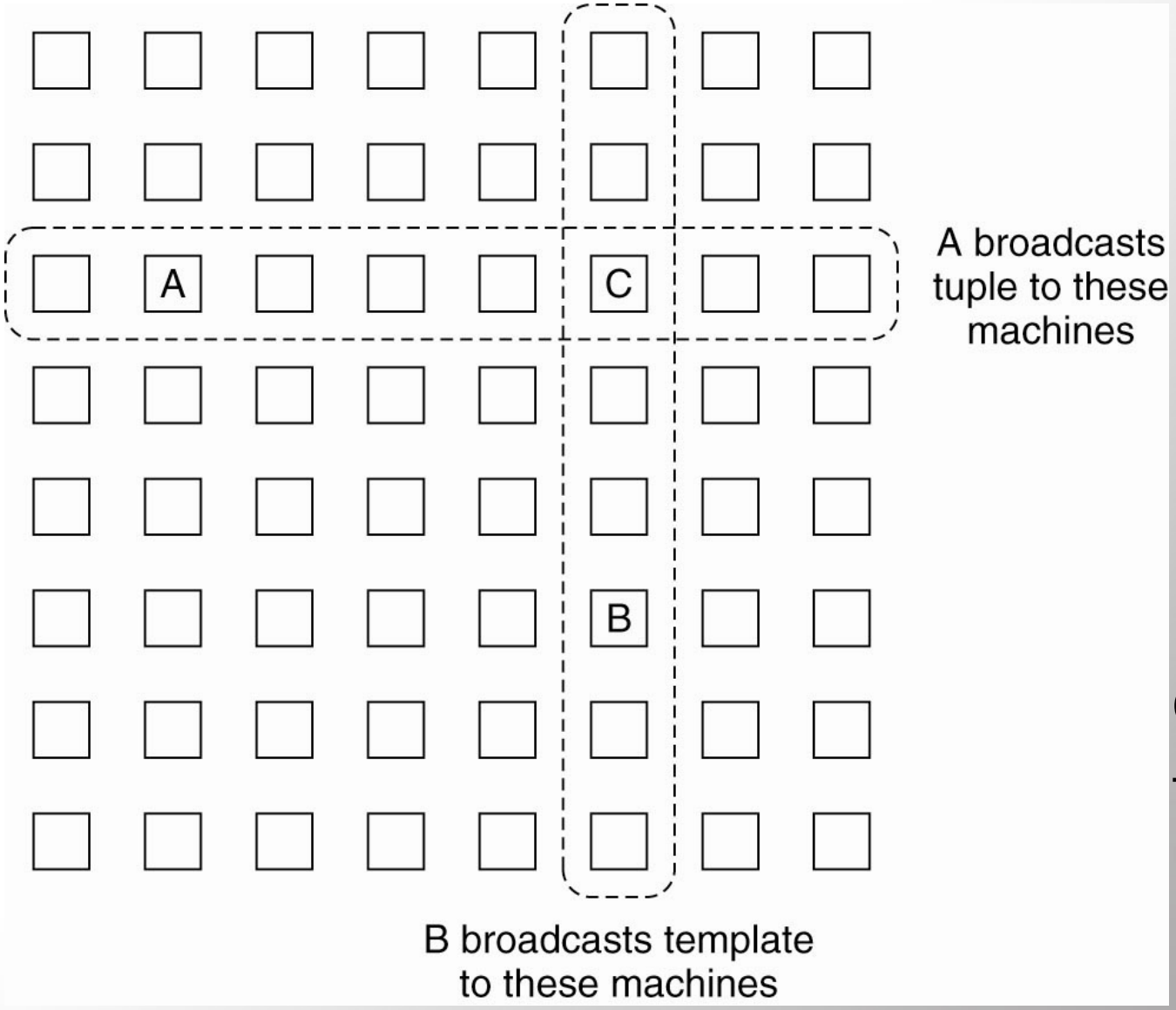
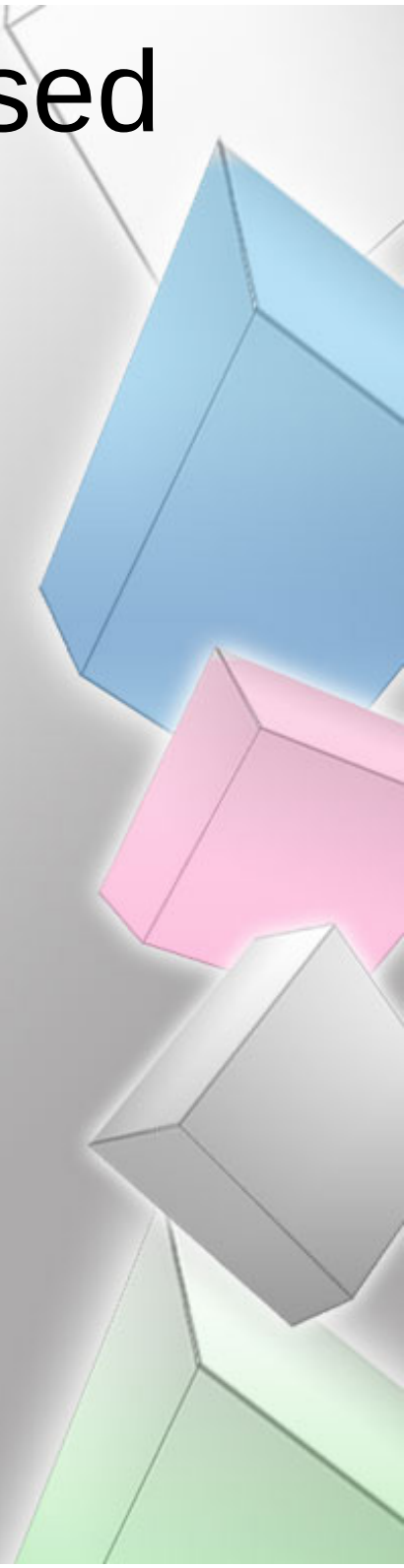


Figure 13-14. Partial broadcasting of tuples and template tuples.

Distributed Coordination-based Systems

- Coordination Models
- Architectures
- Communication
- Naming
- Consistency and Replication
- Fault Tolerance
- Security



Example: Fault Tolerance in TIB/Rendezvous (1)

A daemon attaches a sequence number to the messages.

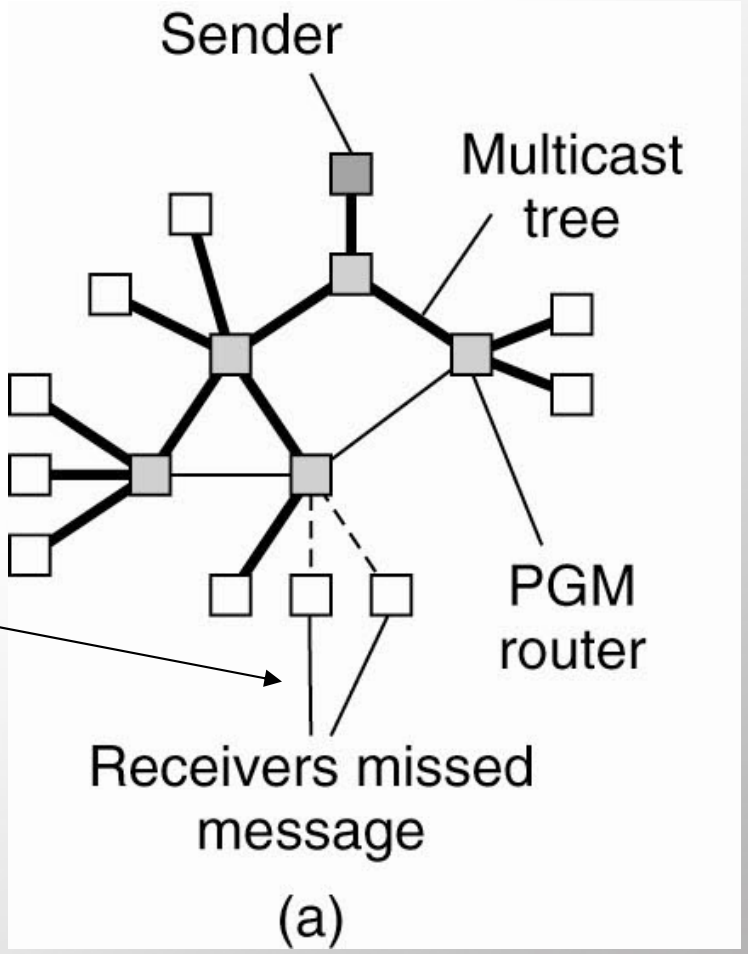
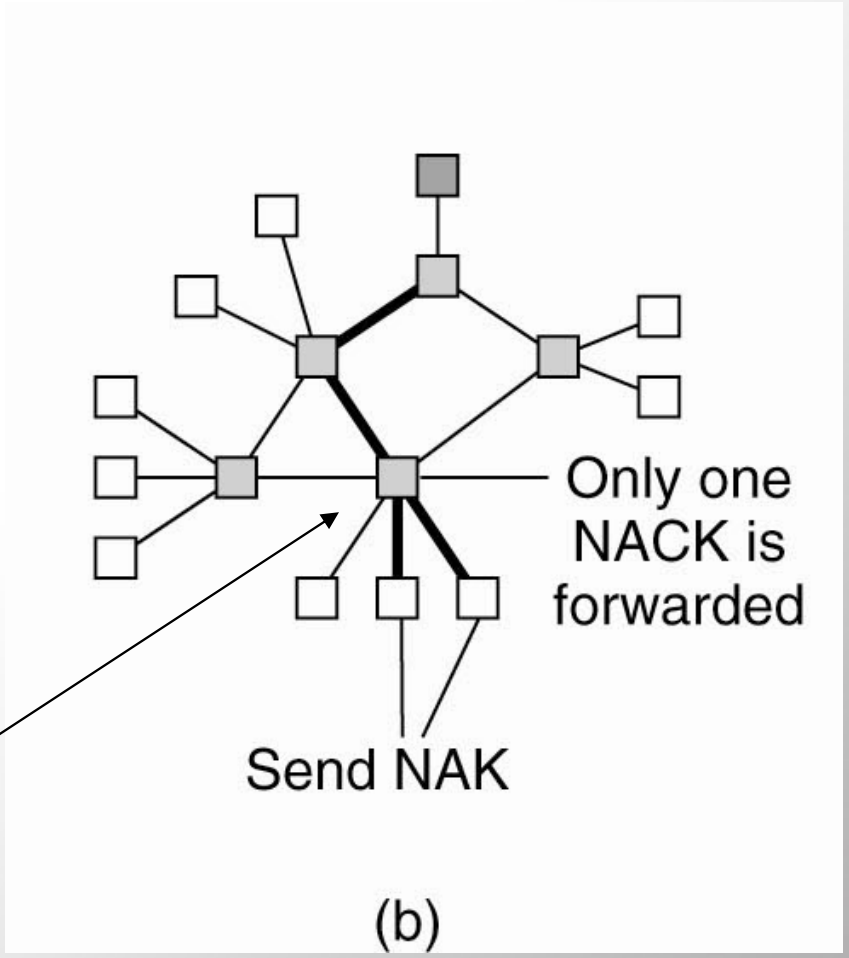


Figure 13-16. The principle of Pragmatic General Multicast (PGM). (a) A message is sent along a multicast tree.

Example: Fault Tolerance in TIB/Rendezvous (2)



Intermediate nodes may have cached the message!

Figure 13-16. The principle of PGM. (b) A router will pass only a single NAK for each message.

Example: Fault Tolerance in TIB/Rendezvous (3)

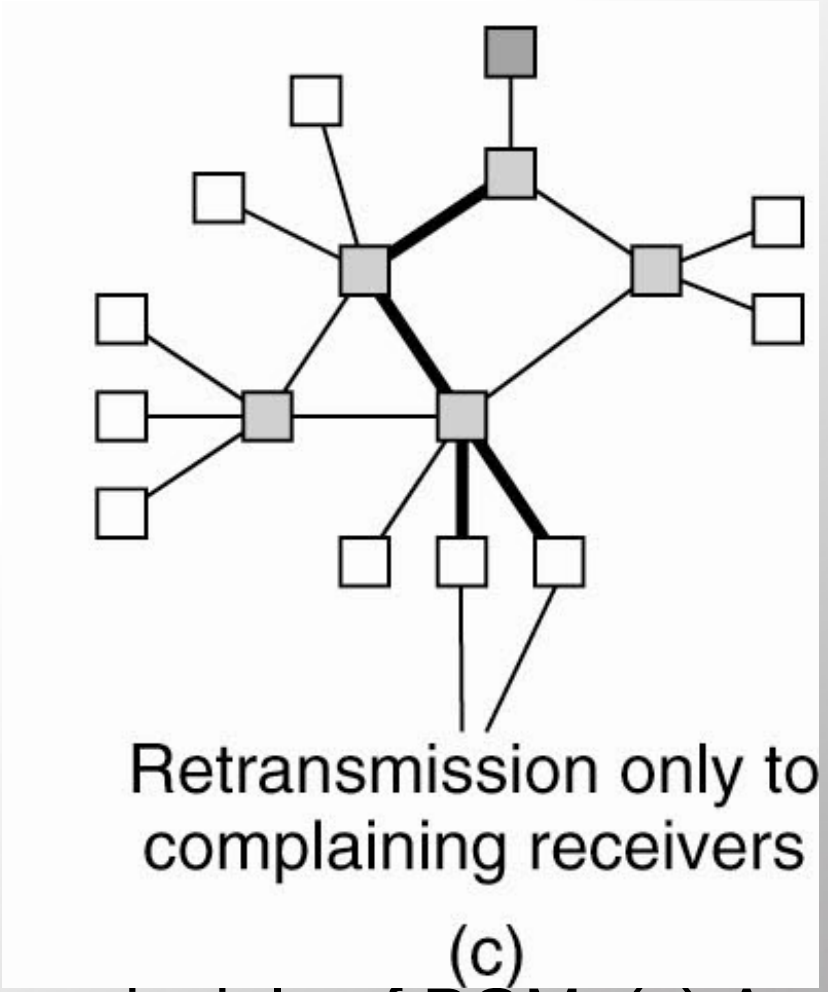
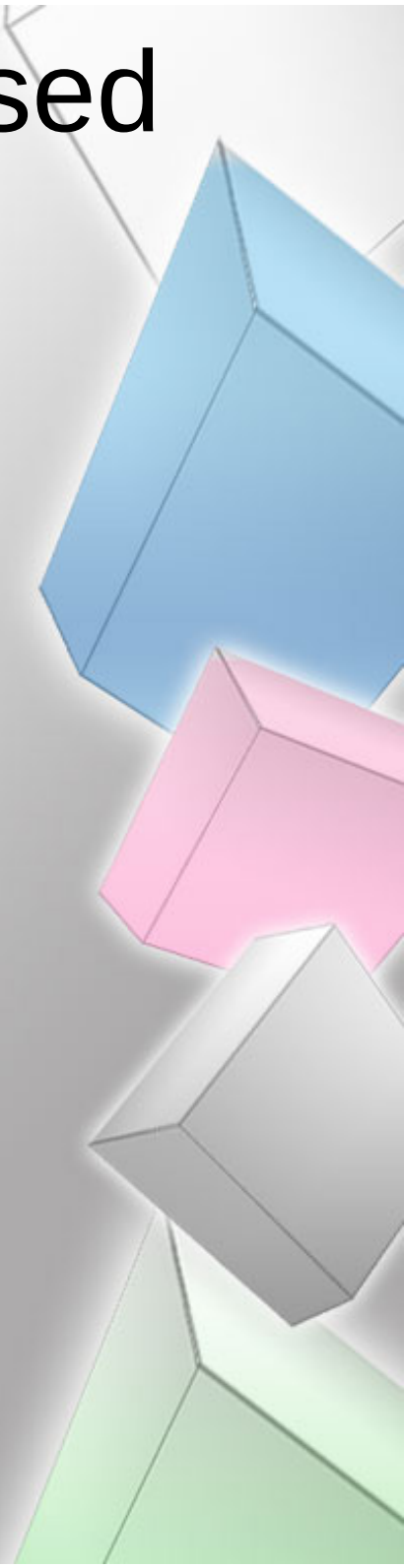


Figure 13-16. The principle of PGM. (c) A message is retransmitted only to receivers that have asked for it.

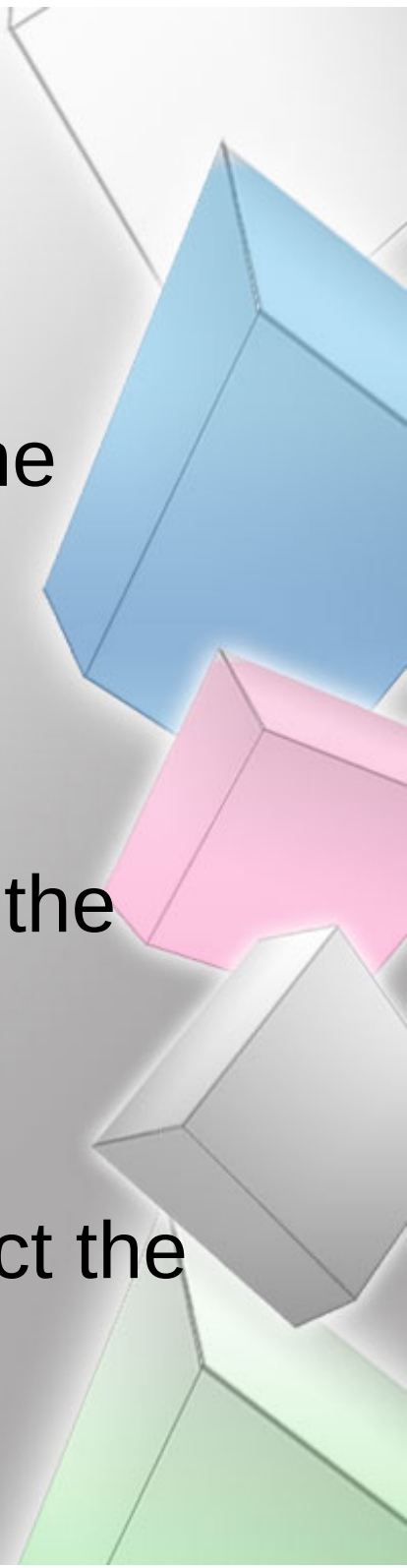
Distributed Coordination-based Systems

- Coordination Models
- Architectures
- Communication
- Naming
- Consistency and Replication
- Fault Tolerance
- Security



Security

- Information confidentiality
  - It is sometimes important to disallow the middleware to inspect published data.
    - Solved through end-to-end encryption
- Subscription confidentiality
  - Subscriptions may not be disclosed to the middleware
- Publication confidentiality
  - Publishers may want to explicitly restrict the group of possible subscribers.



Decoupling Publishers from Subscribers

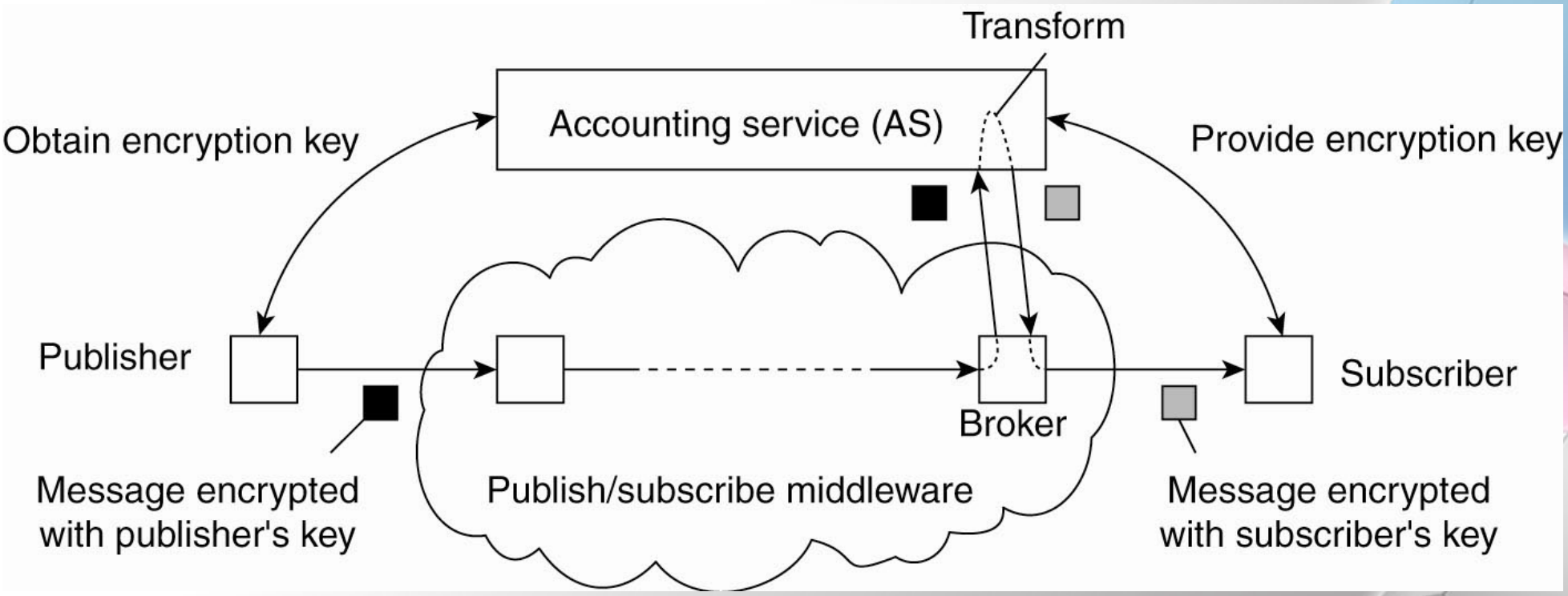


Figure 13-18. Decoupling publishers from subscribers using an additional trusted service.

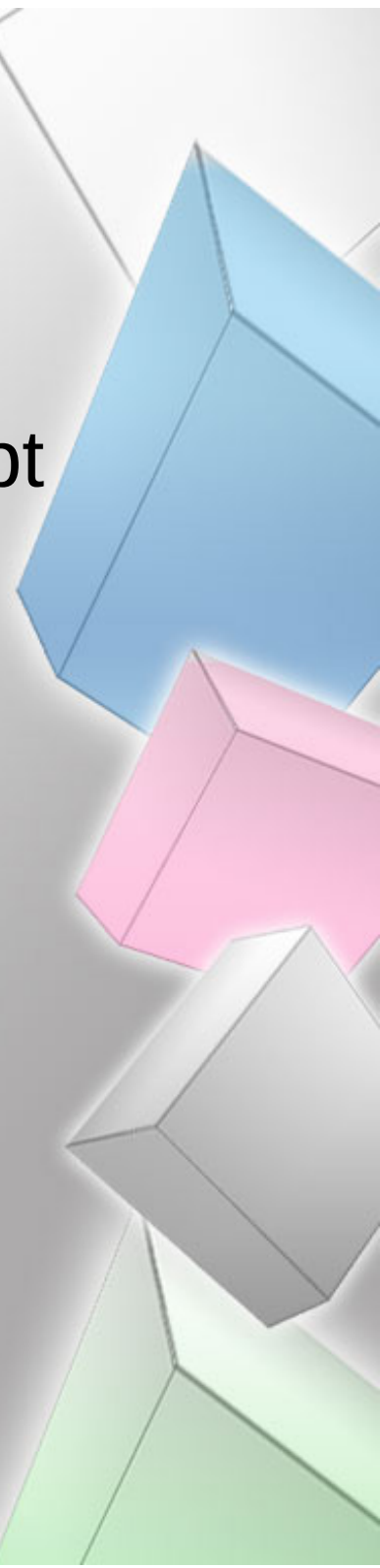
Secure Shared Dataspaces

- A common approach is to encrypt the fields of data items and let matching take place only when decryption succeeds and content matches with a subscription.

- Considering that most implementations make use of only a single server, extending that server with authentication and authorization mechanisms is often the approach followed in practice.

End of Lesson 13

- Readings
  - Distributed Systems, Chapter 13, except 13.2.3 and 13.8.2.



JMS-API

- The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients.
- JMS is a part of the Java Platform, Enterprise Edition.
- Message-oriented technologies attempt to relax tightly coupled communication (such as TCP network sockets, CORBA or RMI) by the introduction of an intermediary component, which in this case would be a queue.

JMS Components

- JMS provider
  - An implementation of the JMS interface for a Message Oriented Middleware (MOM). Providers are implemented as either a Java JMS implementation or an adapter to a non-Java MOM.
- JMS client
  - An application or process that produces and/or receives messages.
- JMS producer
  - A JMS client that creates and sends messages.
- JMS consumer
  - A JMS client that receives messages.
- JMS message
  - An object that contains the data being transferred between JMS clients.
- JMS queue
  - A staging area that contains messages that have been sent and are waiting to be read.
- JMS topic
  - A distribution mechanism for publishing messages that are delivered to multiple subscribers.

JMS API

- The JMS API supports two models:
  - point-to-point or queuing model
  - publish and subscribe model

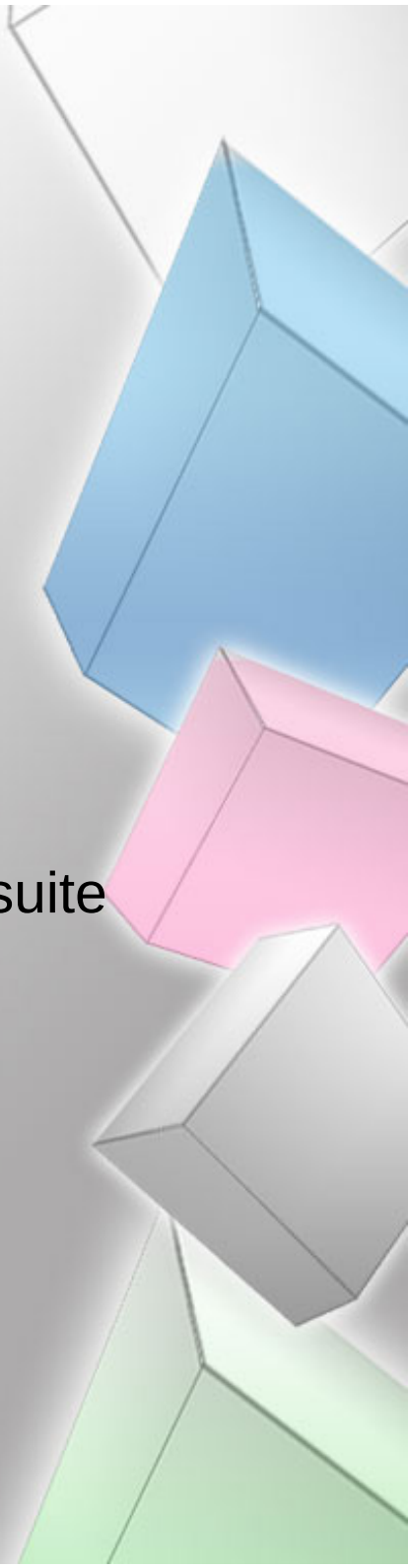
- In the point-to-point or queuing model, a sender posts messages to a particular queue and a receiver reads messages from the queue. Here, the sender knows the destination of the message and posts the message directly to the receiver's queue. It is characterized by the following.

- The publish/subscribe model supports publishing messages to a particular message topic. Subscribers may register interest in receiving messages on a particular message topic. In this model, neither the publisher nor the subscriber know about each other.

Support for JMS

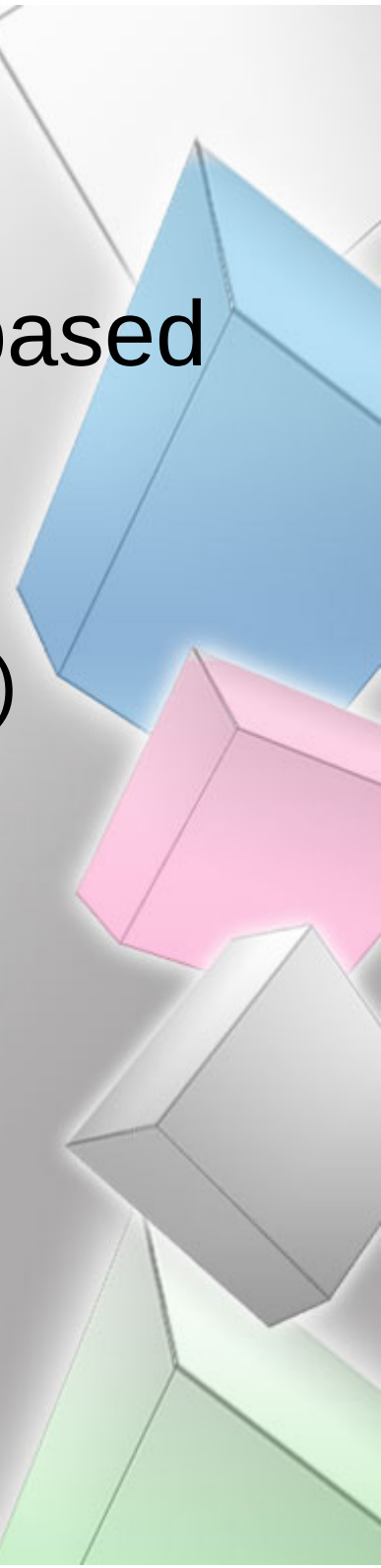
- Open Source
  - OpenJMS, from The OpenJMS Group
  - JBoss Messaging from JBoss
  - Open Message Queue, from Sun Microsystems
  - And others….

- Commercial
  - BEA Weblogic, part of the Oracle Fusion Middleware suite
  - Oracle AQ
  - SAP NetWeaver WebAS Java JMS from SAP AG
  - SonicMQ from Progress Software
  - TIBCO Software
  - WebSphere Application Server from IBM
  - WebSphere MQ from IBM (formerly MQSeries)

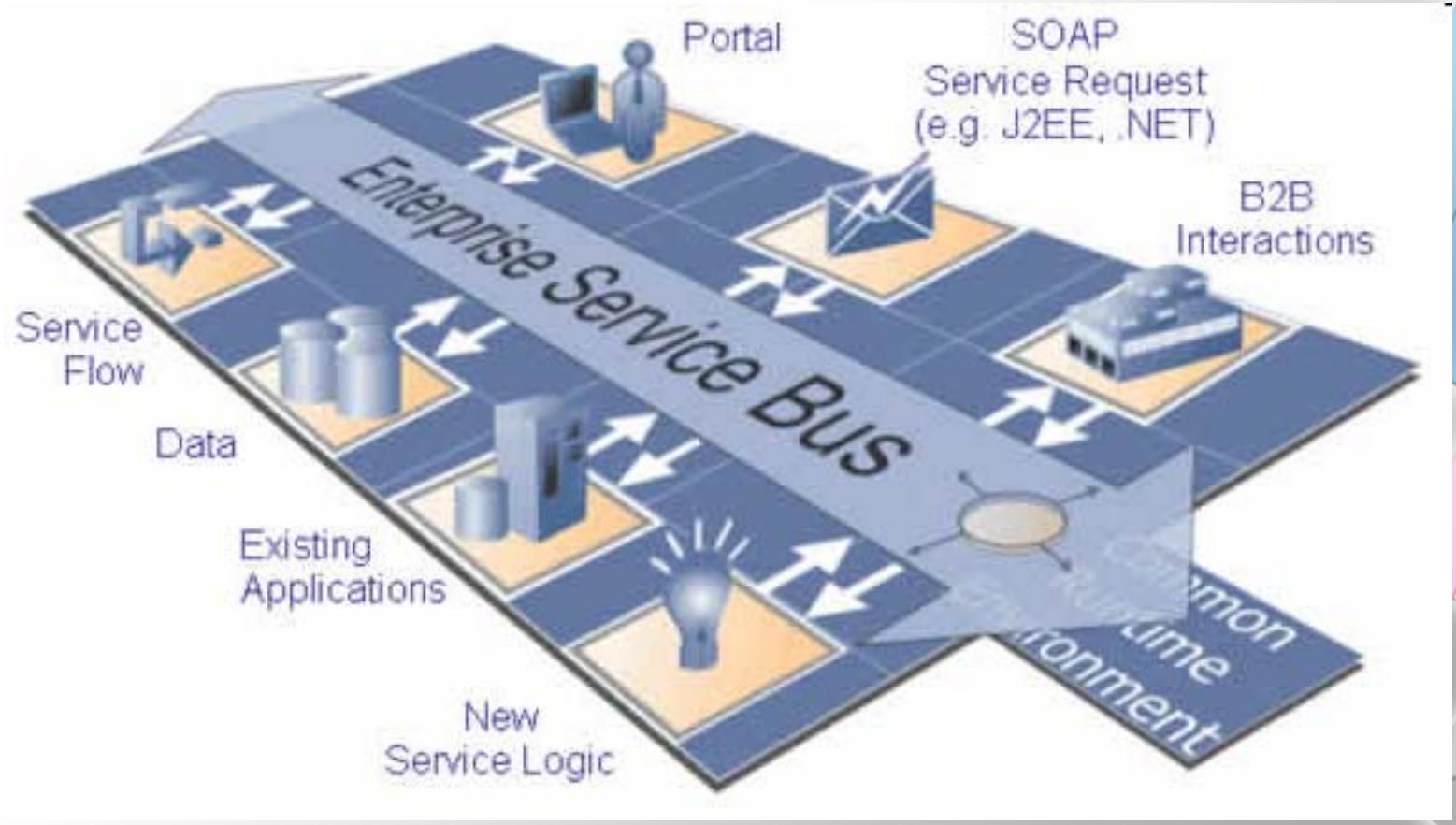


Project Proposal 6

- Implement in Java two simple JMS-based applications
  - Point-to-Point (Hotels Management)
  - Publish/Subscribe (Stock Management)

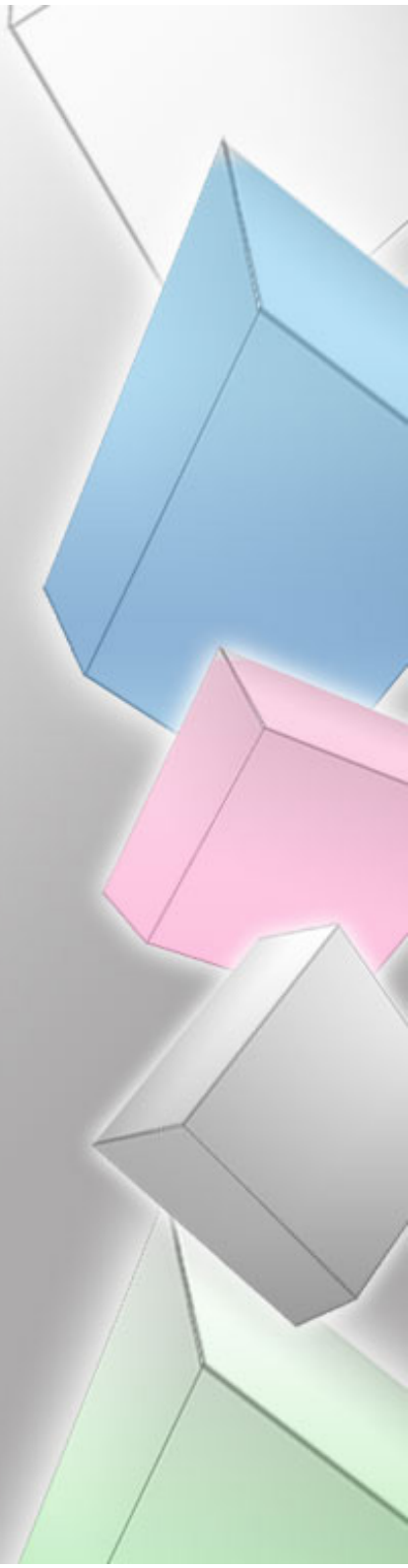


Messaging and Enterprises



Open Source Enterprise Messaging

- Apache ActiveMQ
- Open Enterprise Service Bus



Good Luck