Distributed Systems Lecture 17

Professor Louise E. Moser

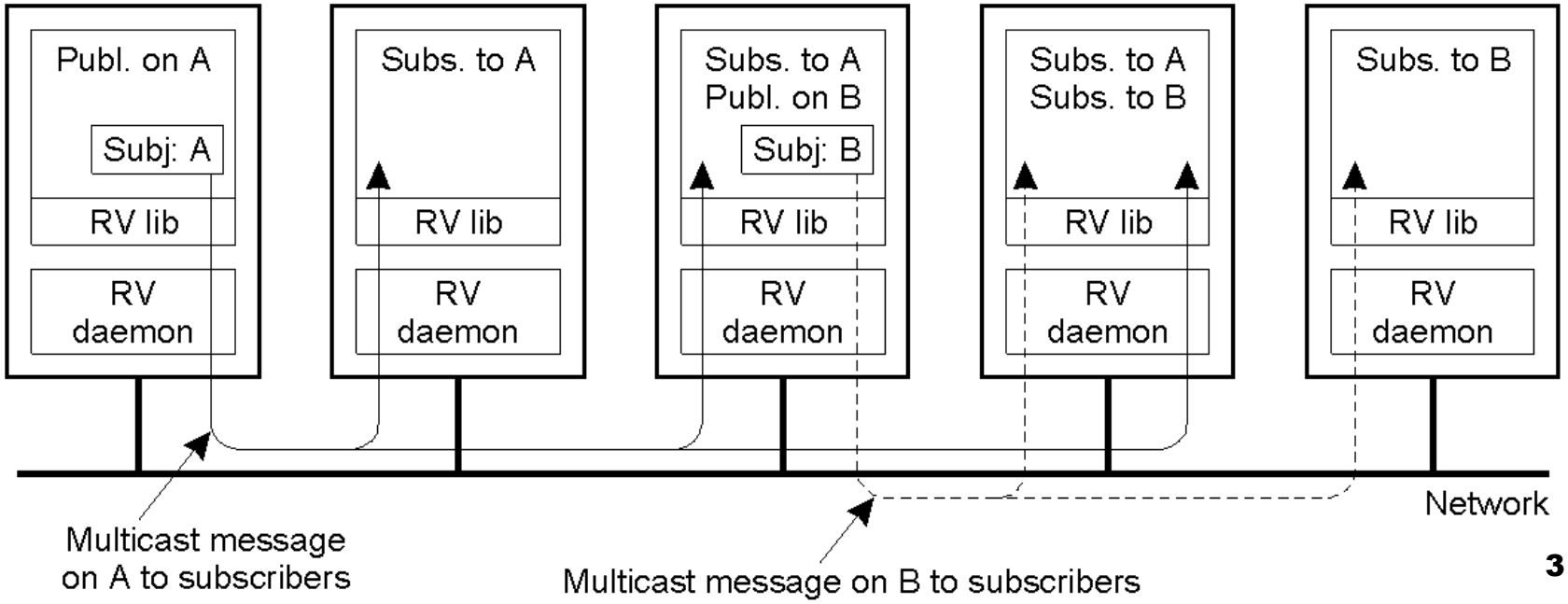
Spring 2013

Coordination-Based Distributed Systems

- **Coordination** deals with all aspects of **communication** and **cooperation** between processes
- Four possible coordination models if we distinguish between:
  - **Temporal coupling: Are communicating / cooperating processes alive at the same time?**
  - **Referential coupling: Do communicating / cooperating processes know each other explicitly?**
- In this lecture, we consider two examples:
  - **TIB / Rendezvous publish / subscribe system based on temporal coupling and referential decoupling**
  - **JavaSpace / Jini tuple space system based on temporal decoupling and referential decoupling**

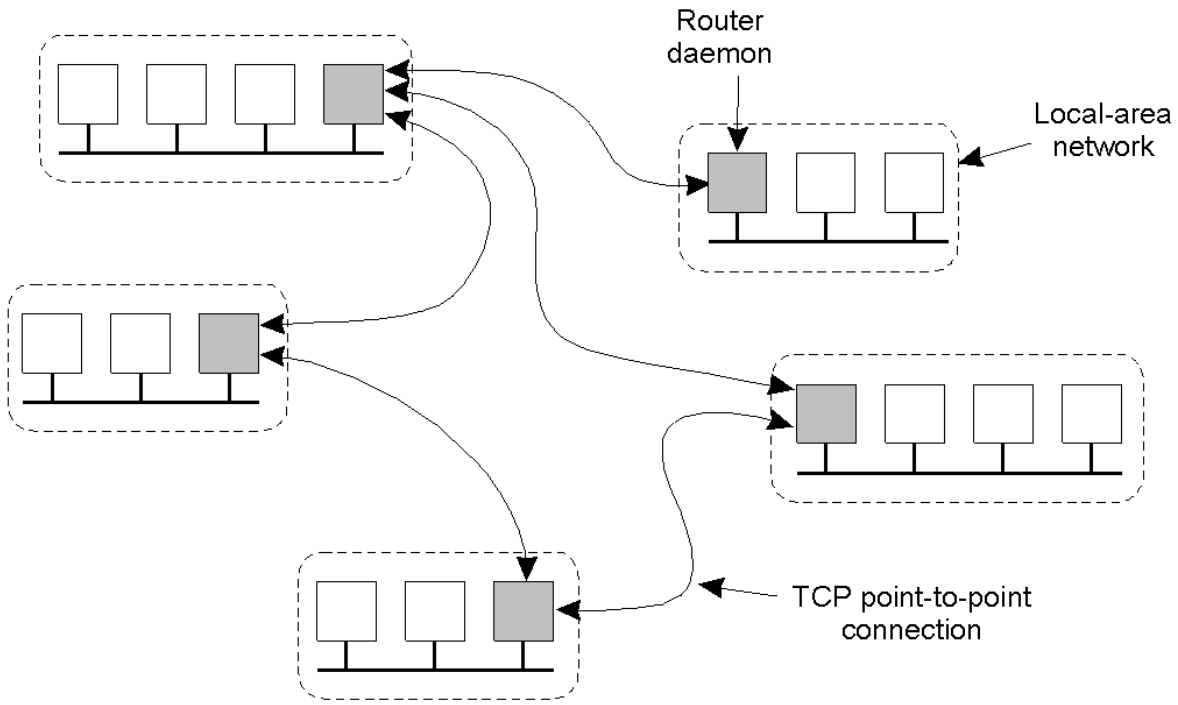
TIB / Rendezvous

- Coordination model: **Temporal coupling** and **referential decoupling**
- **Publish / subscribe system** with **subject-based addressing**
  - **Publishing a message on subject *X* means that the message is sent to all current subscribers to *X***
  - **Receiving a message on subject *X* is possible only if the receiver had subscribed to *X***



TIB / Rendezvous

- TIB / Rendezvous uses **multicasting** to send messages to subscribers
- To span large-scale networks, TIB / Rendezvous effectively builds an **overlay network** with proprietary multicast routers



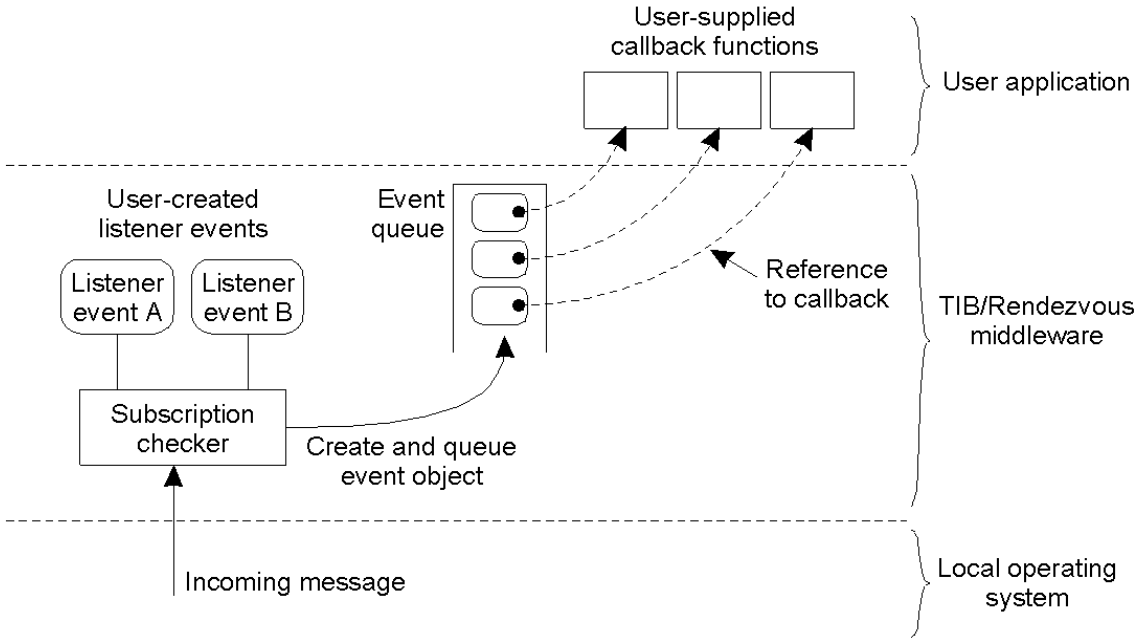
Messages

- TIB / Rendezvous message header fields

Attribute	Type	Description
Name	String	The name of the field, possibly NULL
ID	Integer	A message-unique field identifier
Size	Integer	The size of the field, in bytes
Count	Integer	The number of elements, in the case of an array
Type	Constant	A constant indicating the type of data
Data	Any type	The data stored in a field

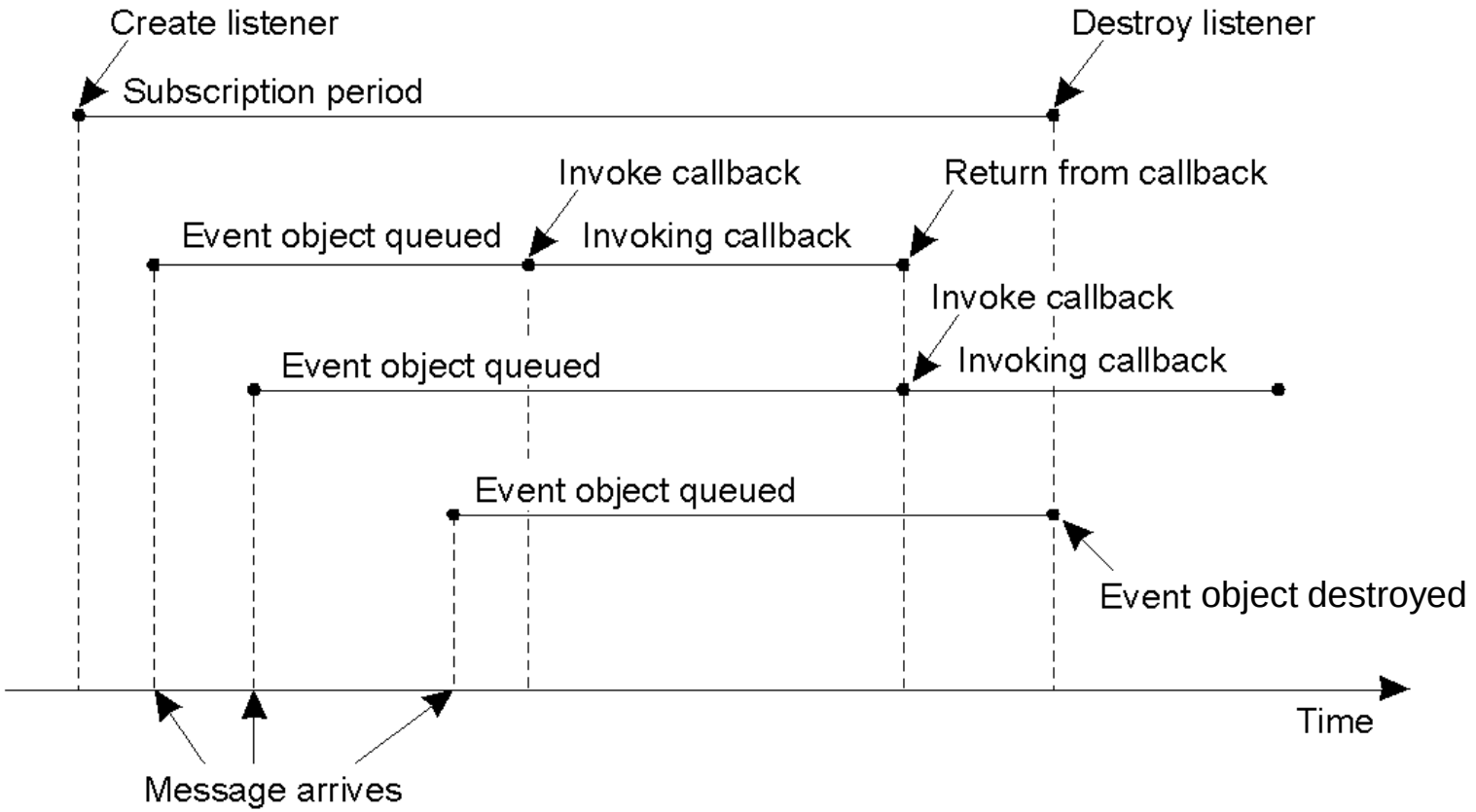
Events

- **Publish-subscribe systems** are typically supported by events
- When a subscriber receives a **notification message** on a subject of interest and to which it subscribed, an **event** occurs
- The subscriber is a **listener** for such kinds of events
  - **A local object registers a callback for the subject of interest**



Events

- **Event scheduling:** Events for the same listener are handled one after the other; they might be lost / ignored if the listener is destroyed too soon

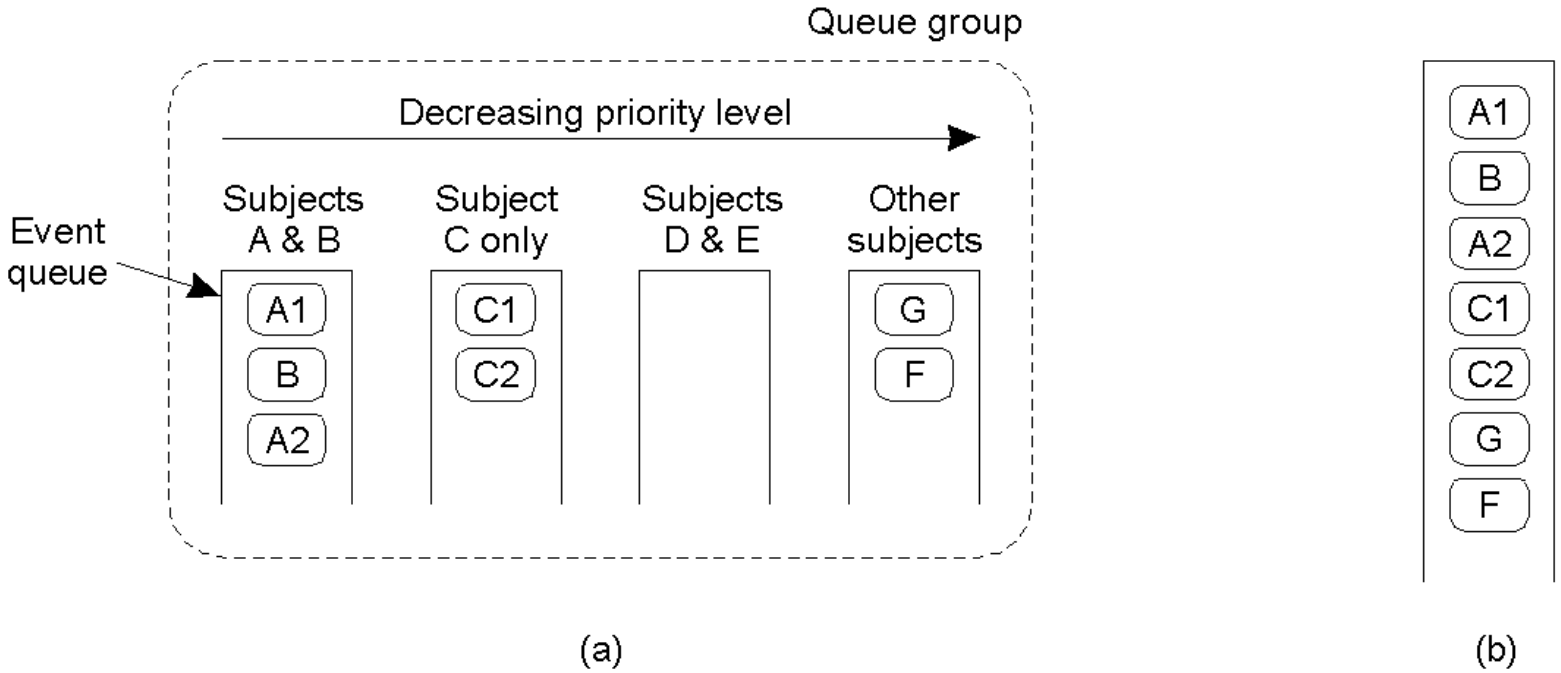


Events

- **Priority scheduling**

  (a) Priority scheduling of events using a queue group

  (b) A queue equivalent to the queues in the queue group in (a)



Naming

- Names are important, because in some sense they provide the “address” of a message

Example	Valid?
Books.Computer_systems.Distributed_Systems	Yes
.ftp.cuss.vu.nil	No (starts with a '.')
ftp.cuss.vu.nil	Yes
NEWS.res.com.so	Yes
Marten..van_Steen	No (empty label)
Marten.R.van_Steen	Yes

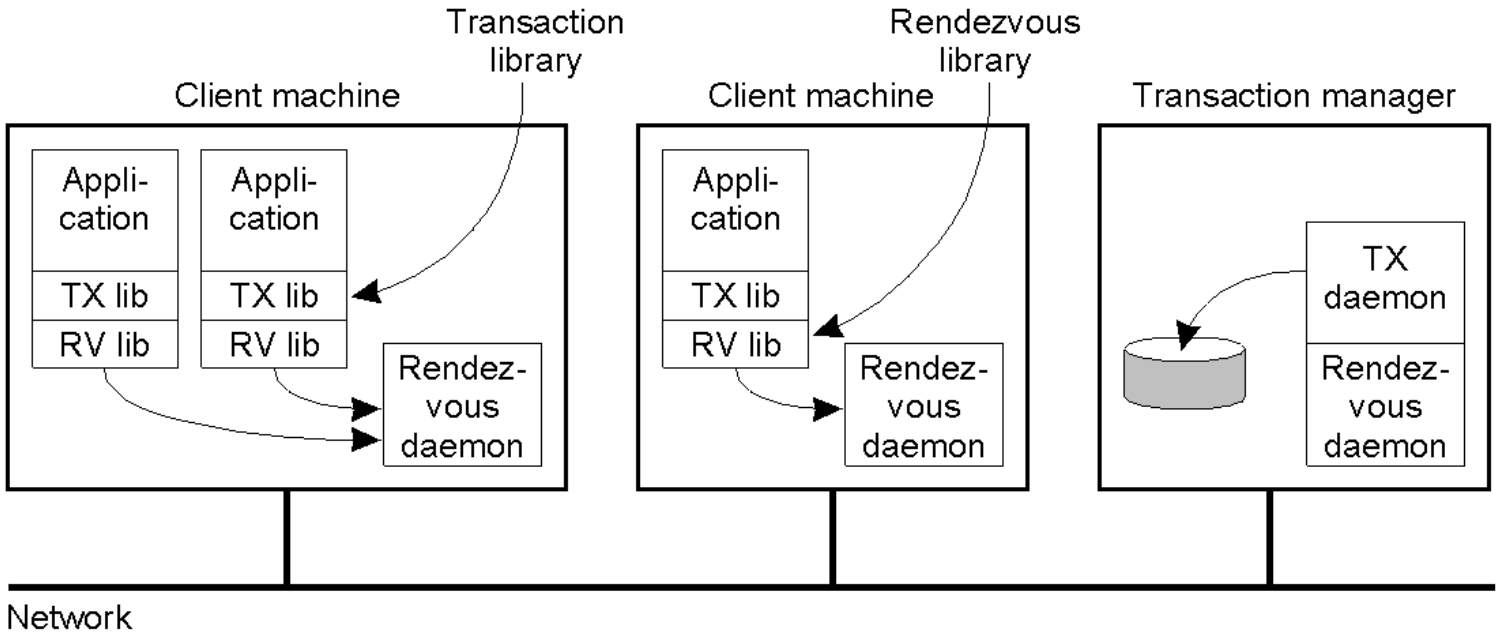
Naming

- **Filtering** ensures that the right messages reach the subscribers
- Examples of filtering using **wildcards *** in subject names

Subject Name	Matches
*.cuss.vu.nil	ftp.cuss.vu.nil www.cuss.vu.nil
nil.vu.*	nil.vu.cuss.ftp nil.vu.cuss.zephyr nil.vu.few.www
NEWS.comp.*.books	NEWS.comp.so.books NEWS.comp.ai.books NEWS.comp.se.books NEWS.comp.theory.books

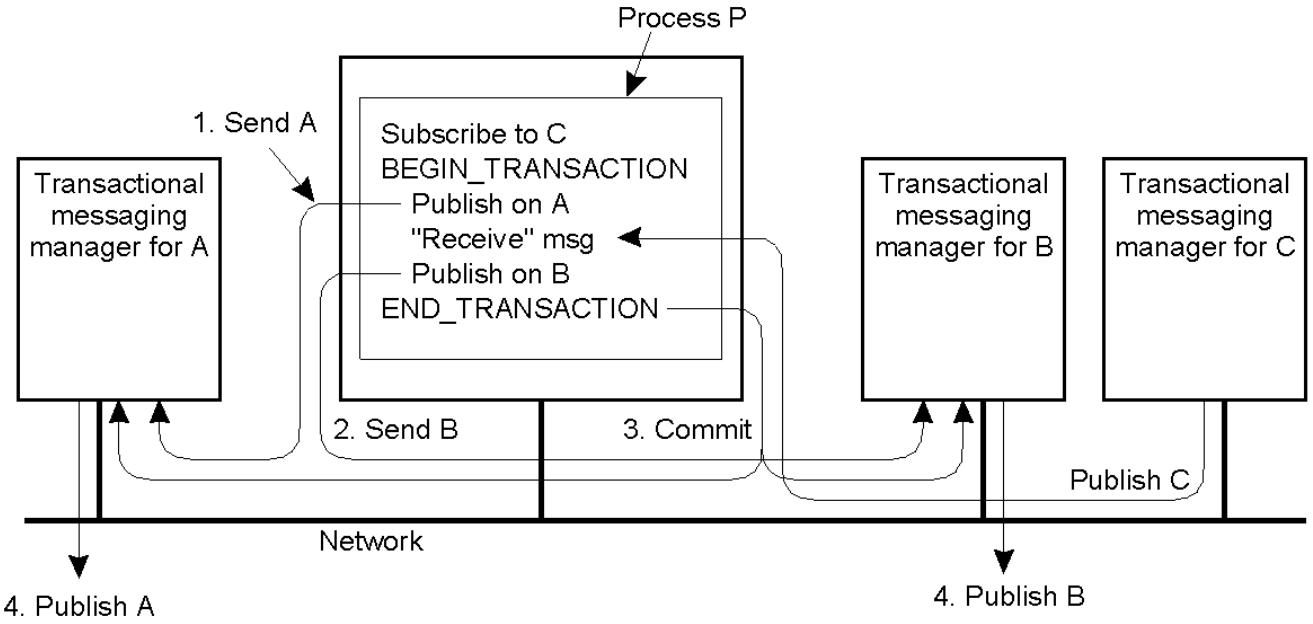
Transactional Messaging

- **Idea:** A (single) sender process must **store published messages** until **commit time**, and only then make the messages available to subscribers
- The layered architecture of TIB / Rendezvous showing the **Transactional Messaging (TX) library** on top of the **Rendezvous (RV) library** and the **Rendezvous daemon**



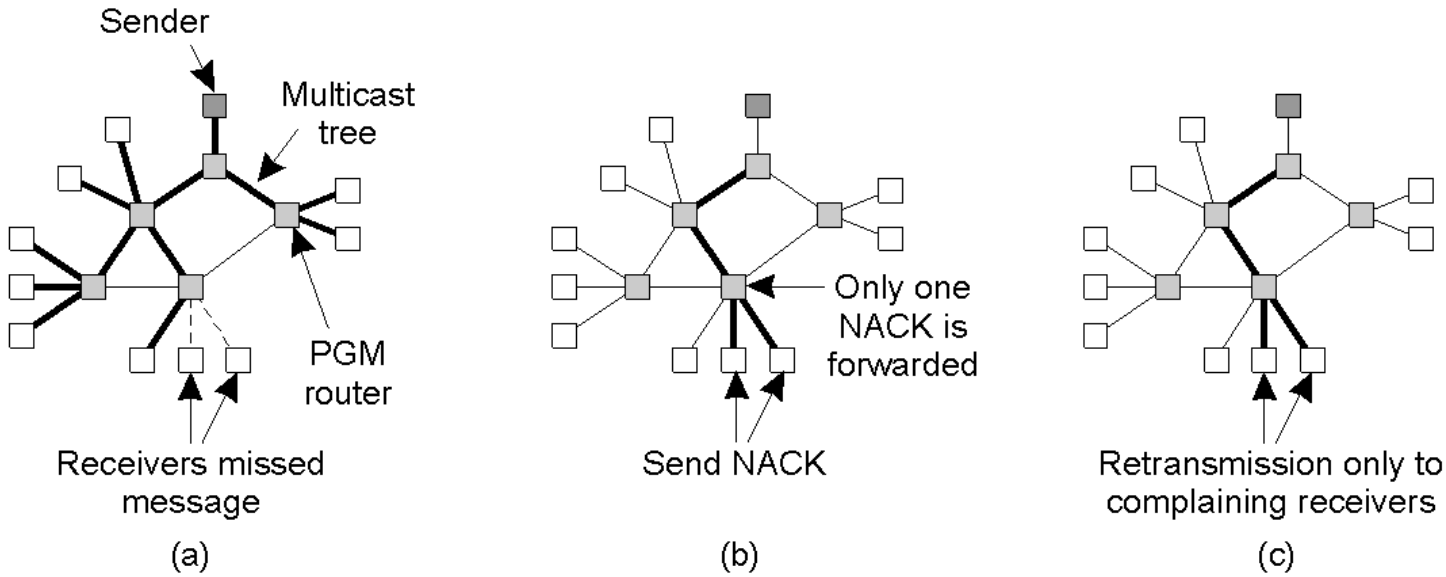
Transactional Messaging

- Transactional messaging is **not** the same as a transaction, because only a **single** process is involved
- Transactional messaging in TIB / Rendezvous showing a **Transactional messaging manager** for each subject and Begin and End of the Transaction



Reliable Multicasting

- TIB / Rendezvous relies on **reliable multicasting** for publishing messages to subscribers
- **Pragmatic General Multicast (PGM):** A NACK-based scheme in which a receiver tells the sender if it is missing a message



(a) A message is sent along a multicast tree

(b) A router passes back only a single NACK for each message missed

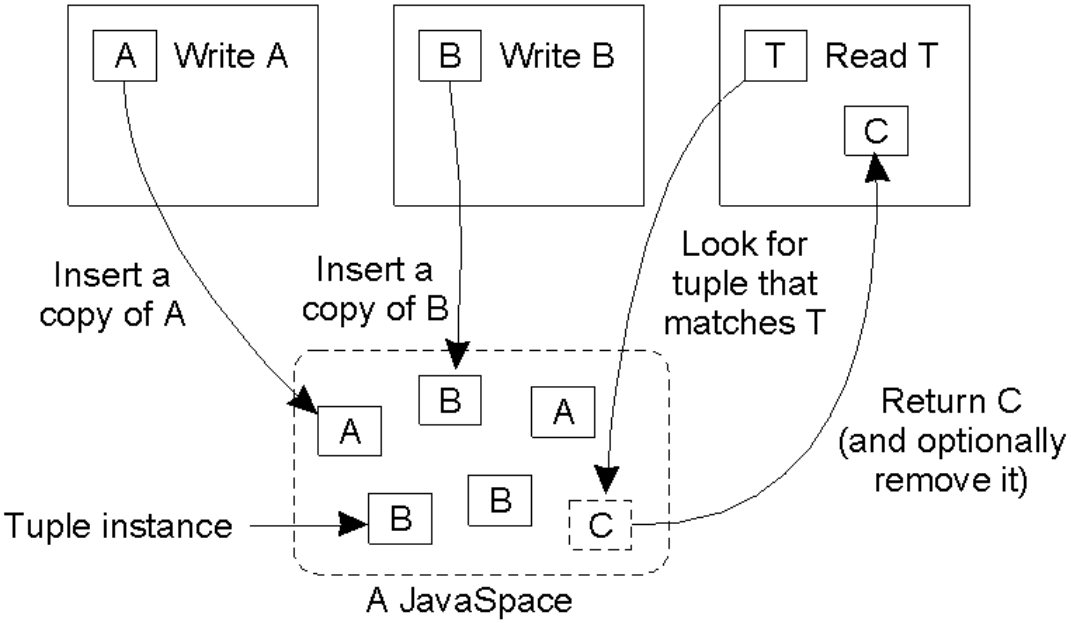
(c) A message is retransmitted only to receivers that requested it

JavaSpaces / Jini

- Coordination model: **Temporal decoupling** and **referential decoupling**
- **JavaSpaces** is a tuple-based storage system
  - A **tuple** is a **set of references to objects**
  - Tuples are stored in **serialized (marshalled)** form in a JavaSpace
- To **read** a tuple:
  - **Construct a template**, typically with some **blank fields (wild cards)**
  - **Match a template against a tuple** using **field-by-field comparison**

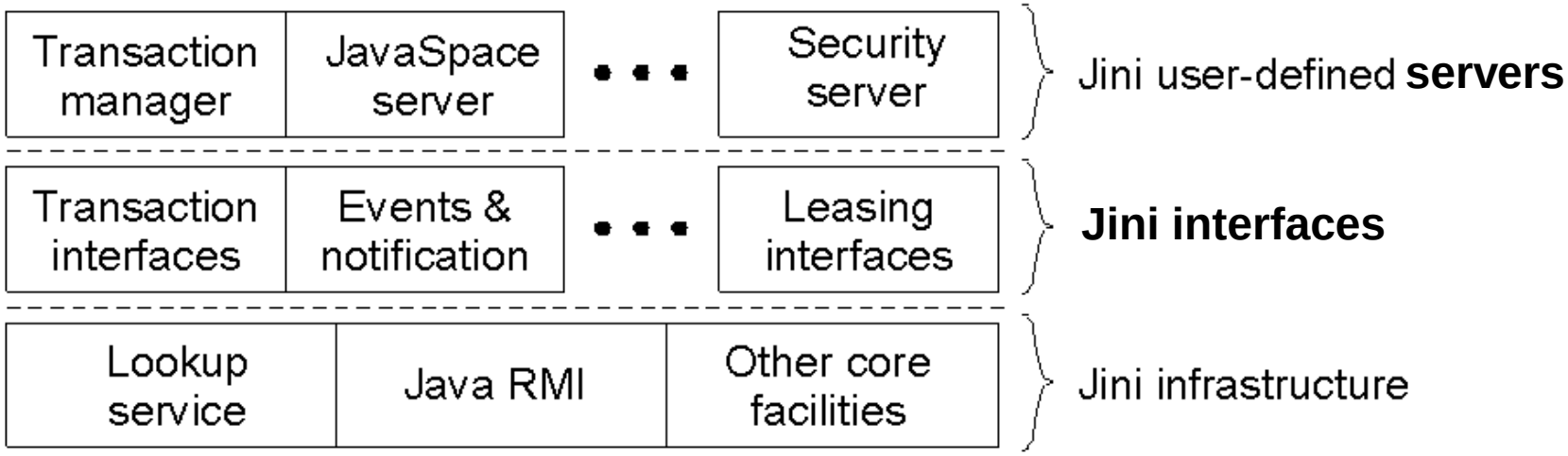
JavaSpaces / Jini

- **Write:** **Store** a **tuple instance** in a **JavaSpace**
- **Read:** **Compare** a **template** to **tuple instances**; the **first match returns a tuple instance**
- **Take:** **Compare** a template to tuple instances; the **first match returns a tuple instance** and **removes the matching instance**



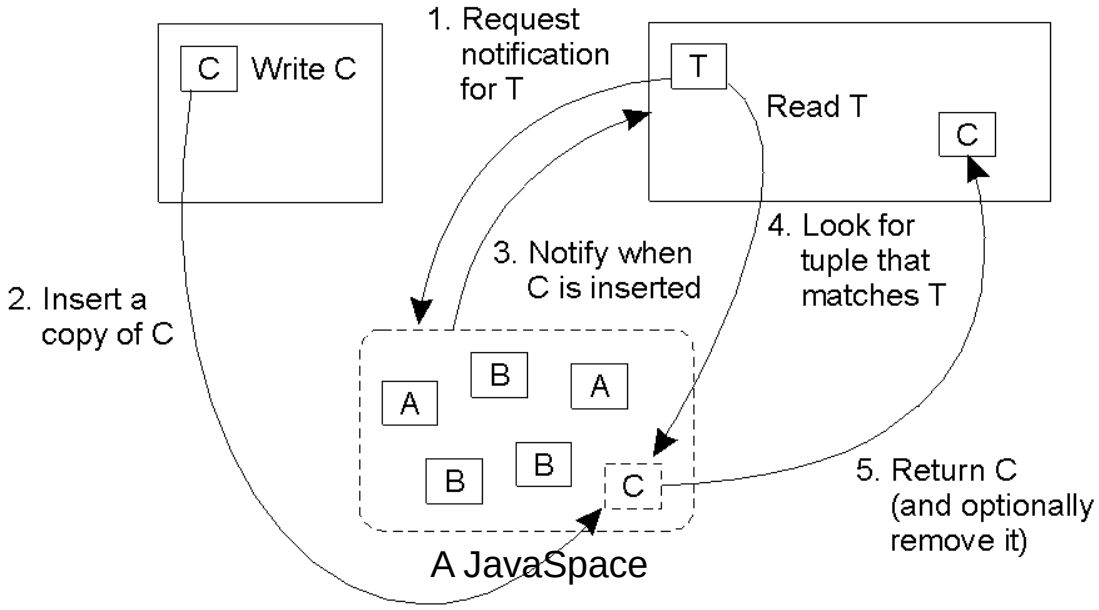
Jini System Architecture

- The layered architecture of Jini



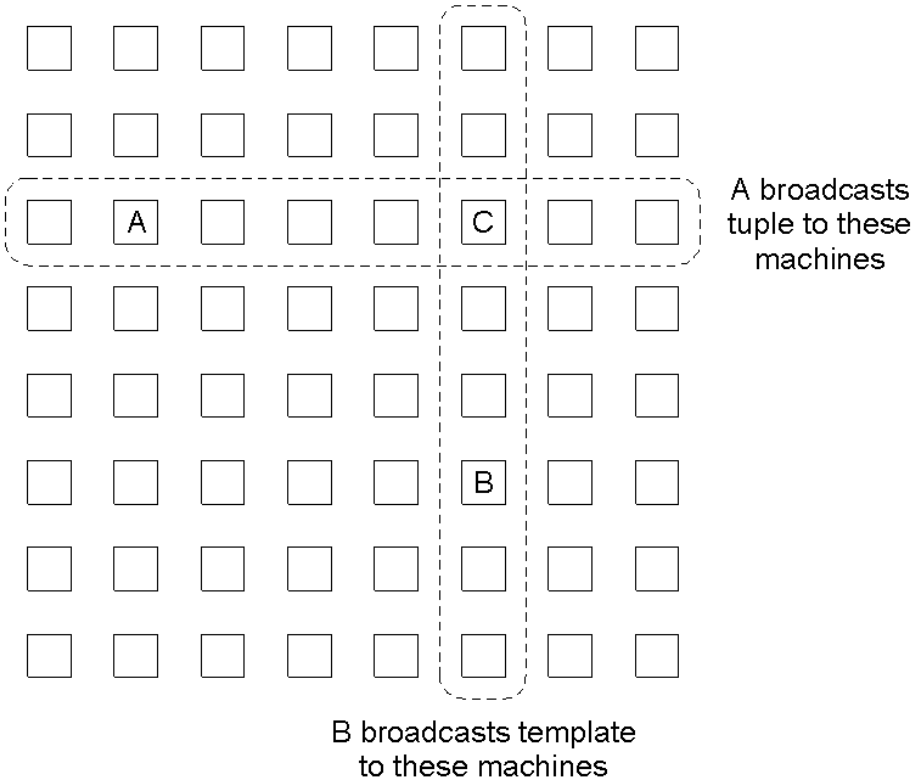
Communication

- A process can register to be **notified** when an **event** (a store or a match) occurs
- Like TIB / Rendezvous, Jini uses a **callback** mechanism and a **listener** object, but is implemented using **Java RMI** rather than the proprietary PGM



Communication

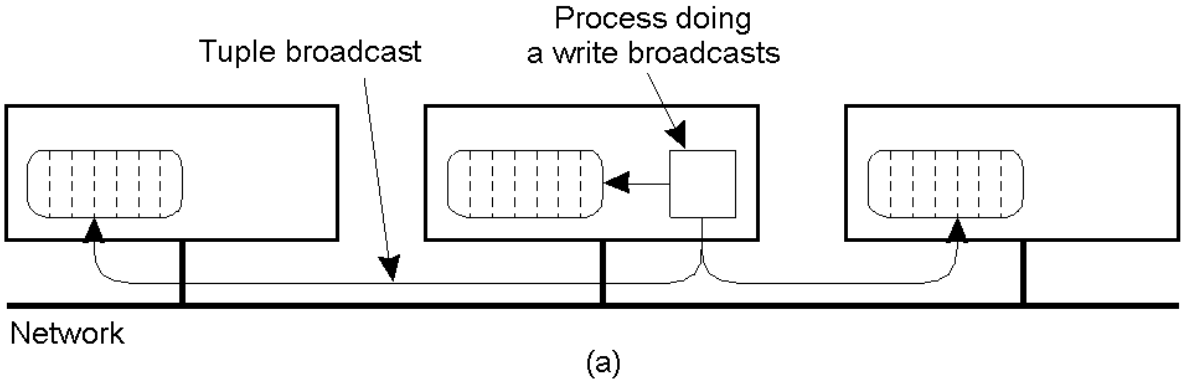
- Partial broadcasting of tuples and tuple templates (similar to the Intersecting Broadcast Machine (IBM) of Melliar-Smith, Moser, Das)



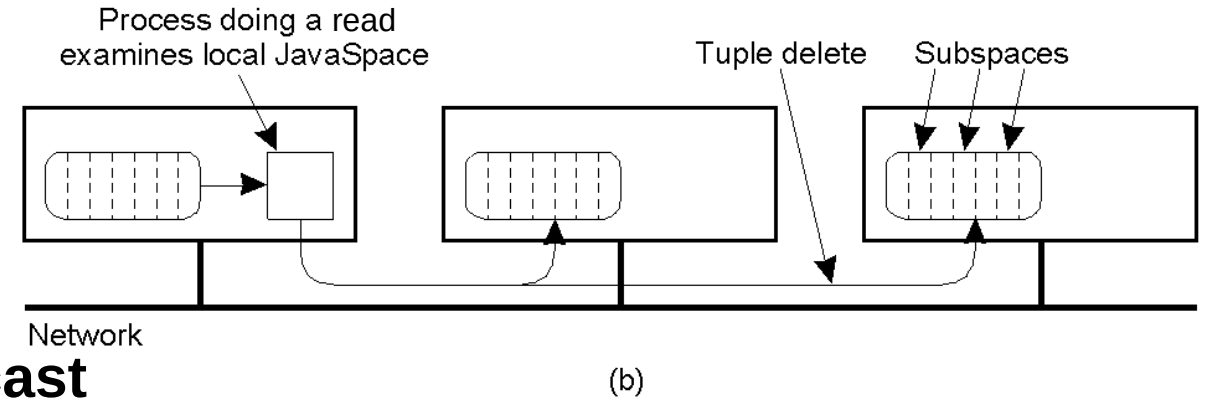
JavaSpace Server

- **Replicated JavaSpace**

(a) Writes of **tuples** are **broadcast**



(b) Reads of **tuple templates** are **local**, but **Takes** of **tuple templates** are **broadcast**



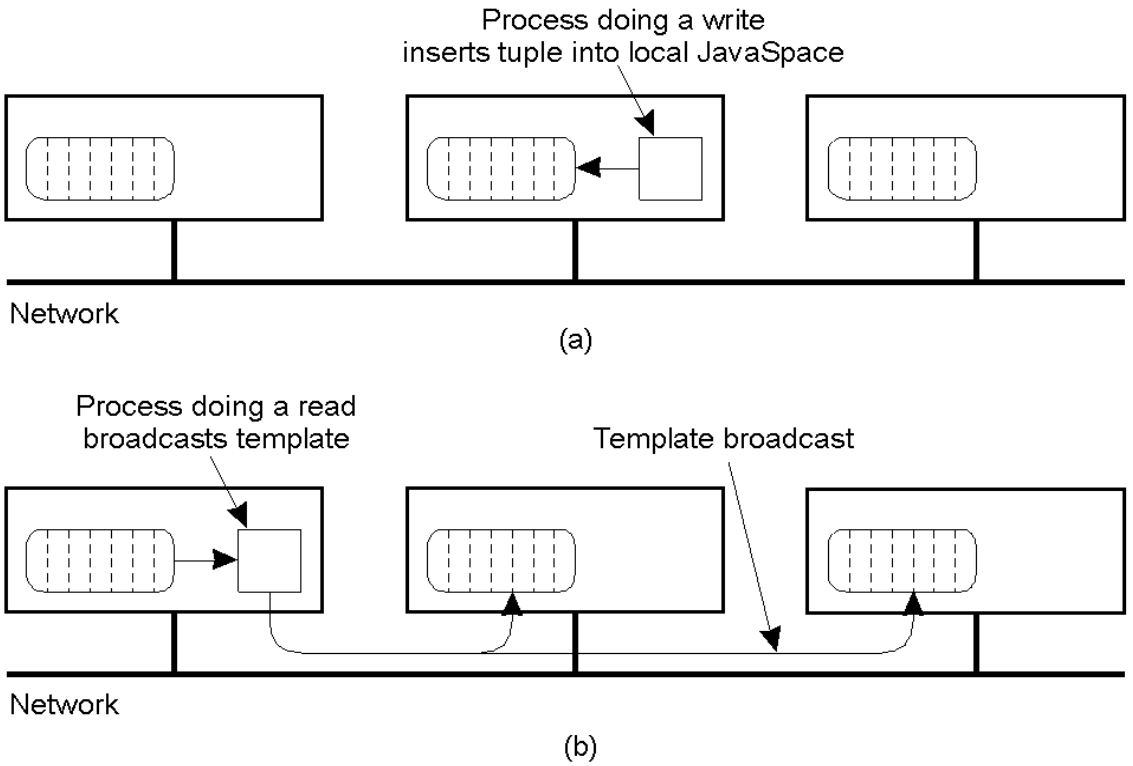
- The dotted lines show subspaces of the JavaSpace

JavaSpace Server

- **Distributed JavaSpace**

(a) Writes of **tuples** are **local**

(b) Reads and **Takes** of **tuple templates** are **broadcast**



- For scalability, do not replicate; instead, use different local JavaSpaces and lose location transparency
- Possibly move a JavaSpace to places that have lots of clients

Jini Lookup Service

- The organization of a **service item**

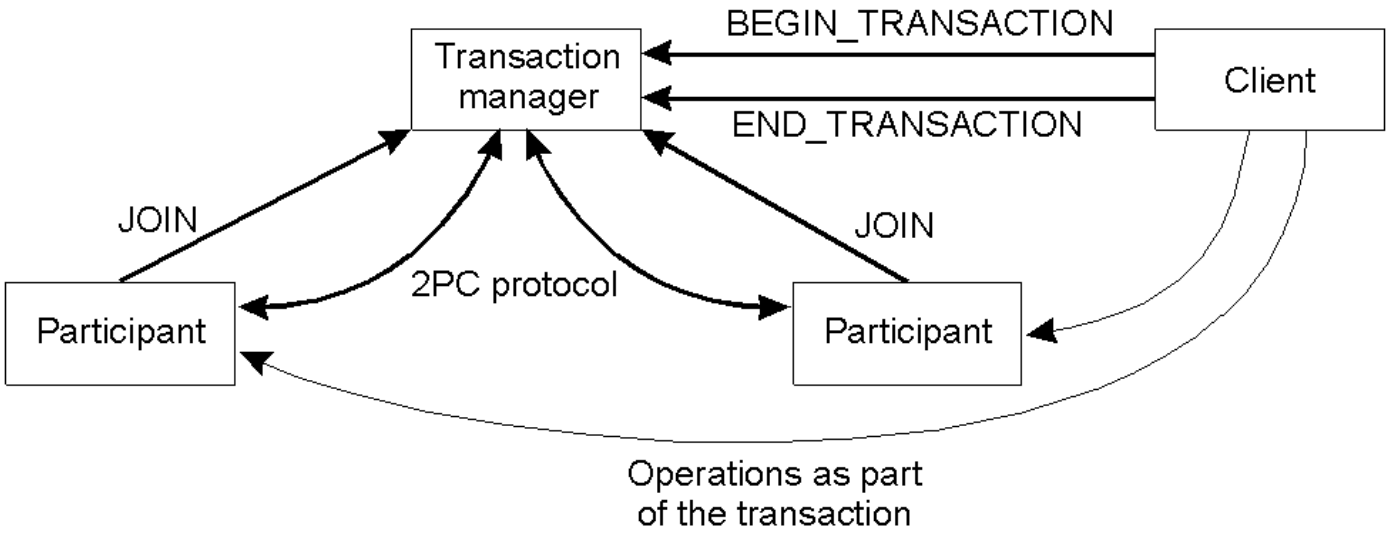
Field	Description
ServiceID	The identifier of the service associated with this item
Service	A (possibly remote) reference to the object implementing the service
AttributeSets	A set of tuples describing the service

- Examples of pre-defined tuples for service items

Tuple Type	Attributes
ServiceInfo	Name, manufacturer, vendor, version, model, serial number
Location	Floor, room, building
Address	Street, organization, organizational unit, state, zip code, country

Synchronization of Transactions

- Jini provides a standard interface for the **2PC protocol**, and offers a default implementation of the 2PC protocol
- Communication required by Jini's transaction protocol is shown by thick lines in the figure



TIB / Rendezvous vs. JavaSpaces / Jini

Issue	TIB / Rendezvous	JavaSpaces / Jini
Major design goal	Uncoupling of processes	Flexible integration
Coordination model	Publish/subscribe	Generative communication
Network communication	Multicasting	Java RMI
Messages	Self-describing	Process specific
Event mechanism	For incoming messages	As a callback service
Processes	General purpose	General purpose
Names	Character strings	Byte strings
Naming services	None	Lookup service
Transactions (operations)	Messages	Method invocations
Transactions (scope)	Single process (see text)	Multiple processes
Locking	No	As JavaSpace operations
Caching and replication	No	No
Reliable communication	Yes	Yes
Process groups	Yes	No
Recovery mechanisms	No explicit support	No explicit support
Security	Secure channels	Based entirely on Java

Wrapping Up

- **Final Exam**
  - Wednesday, June 12, 12:00-3:00pm, this room
  - Similar to Midterm, but 12 questions with more emphasis on the second part of the course
  - Two 8.5”x11” cheat sheets allowed
- **Study hard!**
- **Good luck!**