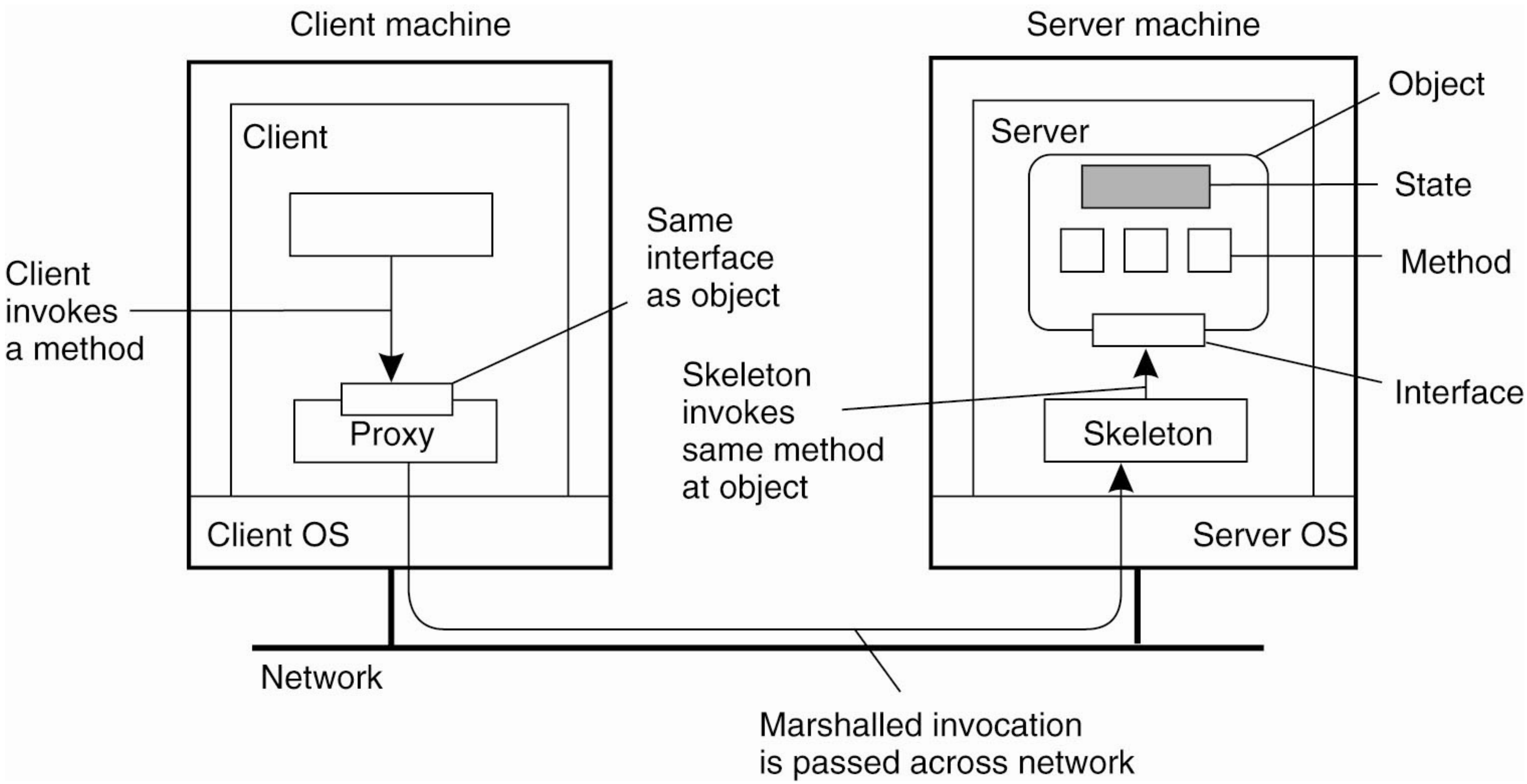
Distributed Middleware

- Distributed objects

- DCOM
- CORBA
- EJBs
- Jini

Distributed Objects



- Figure 10-1. Common organization of a remote object with client-side proxy.

Distributed Objects vs. RPC

RPC : Remote Procedure Call

- Provides argument marshalling / unmarshalling
- Server handles invocation

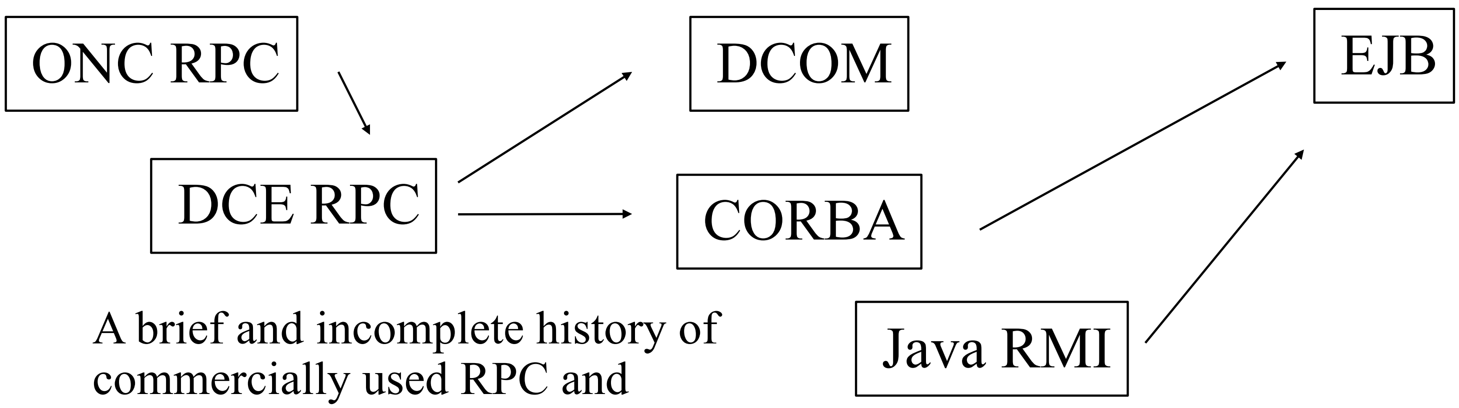
Distributed Objects

- Remote methods on remote objects
- RPC + distributed object references

Distributed object operation:

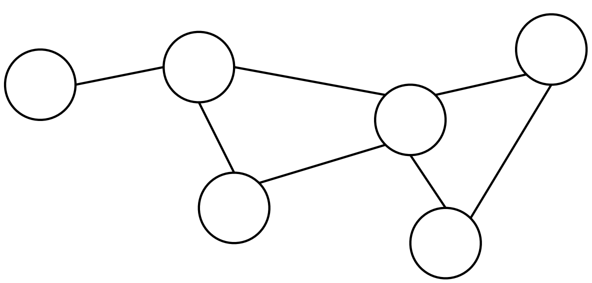
- Server side: create object, register it (register with what?) (always in this order?)
- Client side: get object reference (from where?), invoke method

Distributed Objects through History



A brief and incomplete history of commercially used RPC and distributed object systems

The vision



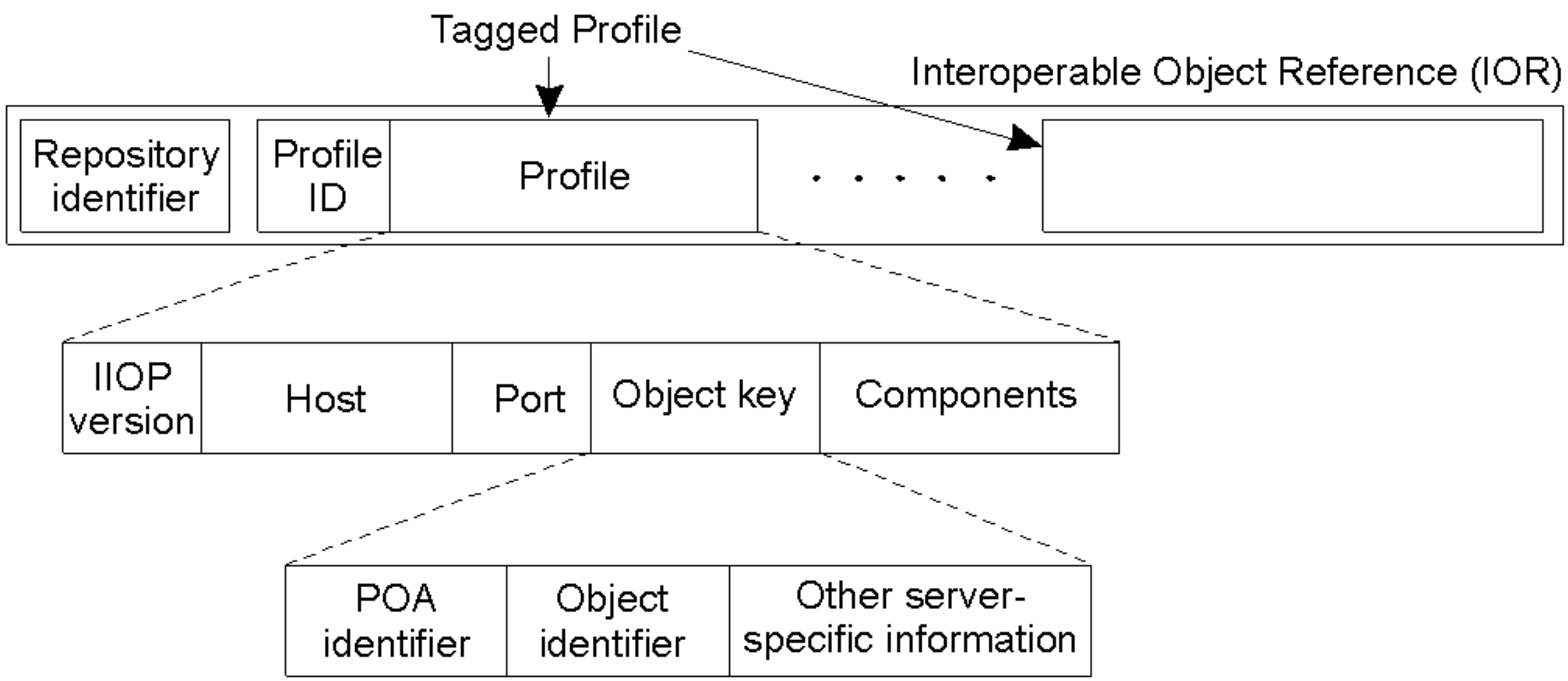
a Grand Distributed System

The reality

Client/Server

Naming: Object References

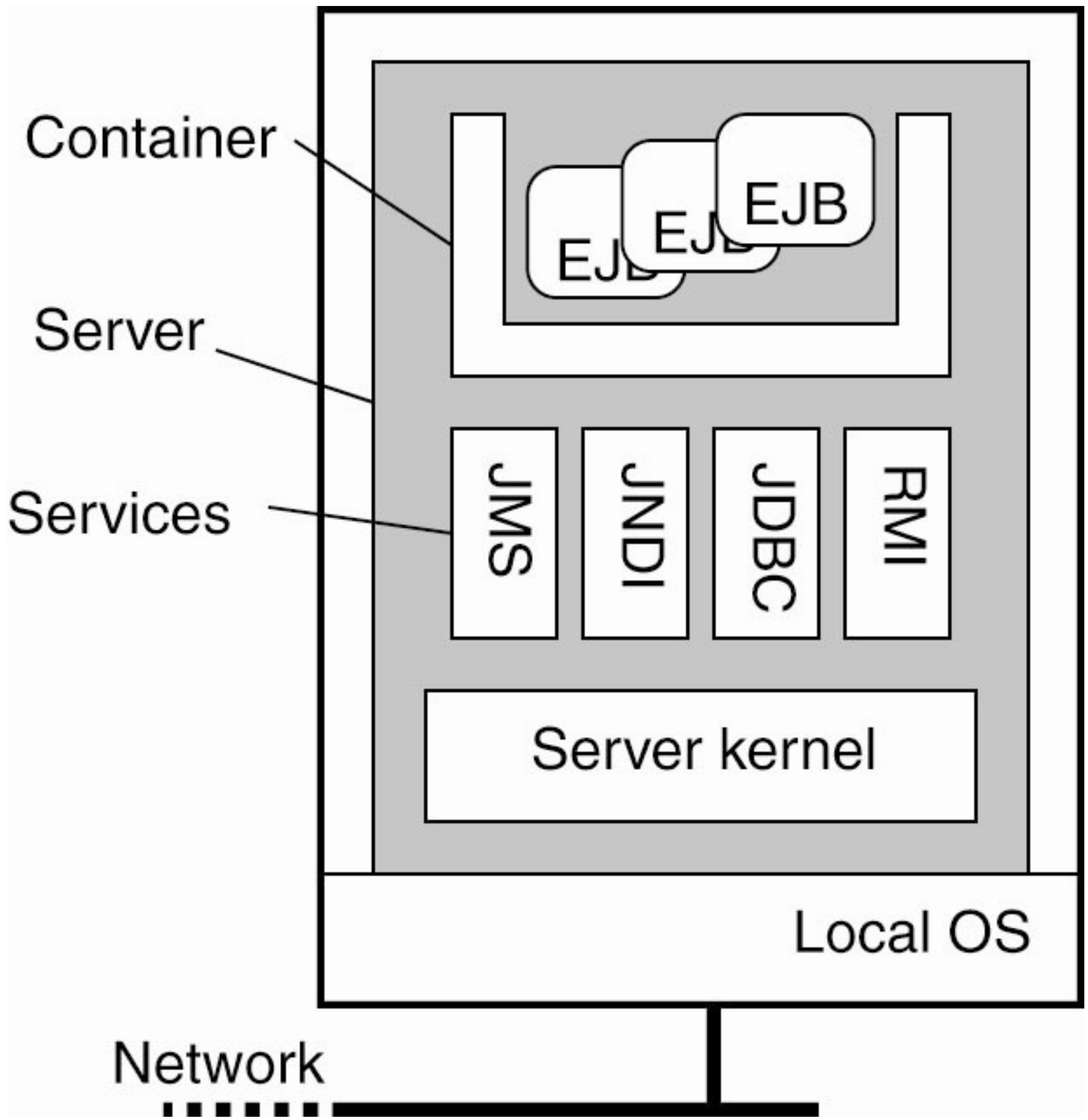
CORBA object reference



- Interoperable object reference: language-independent techniques for referring to objects

Example: Enterprise Java Beans

- Figure 10-2. General architecture of an EJB server.



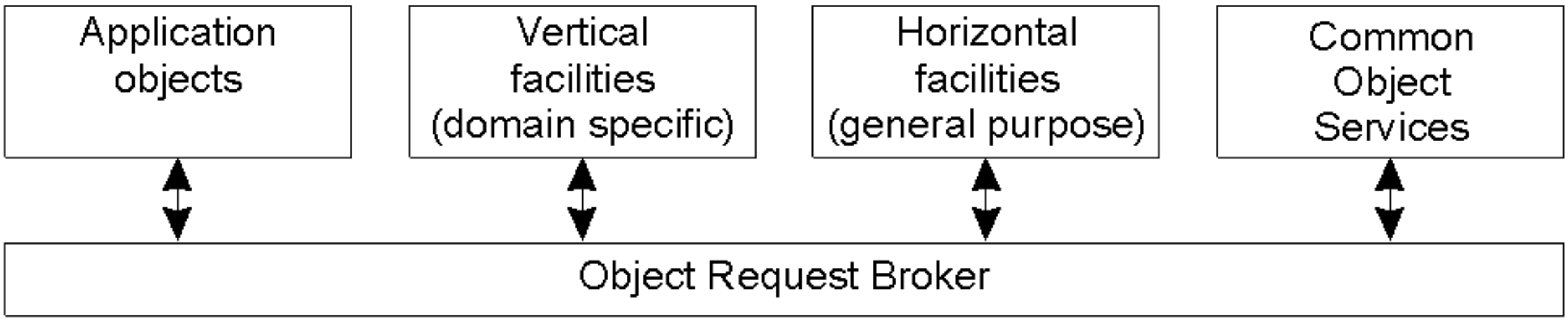
Parts of an EJB

- Home interface:
  - Object creation, deletion
  - Location of persistent objects (entity beans)
  - Object identifier is class-managed
- Remote interface
  - “business logic”
  - i.e. the object itself
- Terminology differences
  - Client/server -> web applications

Four Types of EJBs

- Stateless session beans
- Stateful session beans
- Entity beans
- Message-driven beans

CORBA Overview

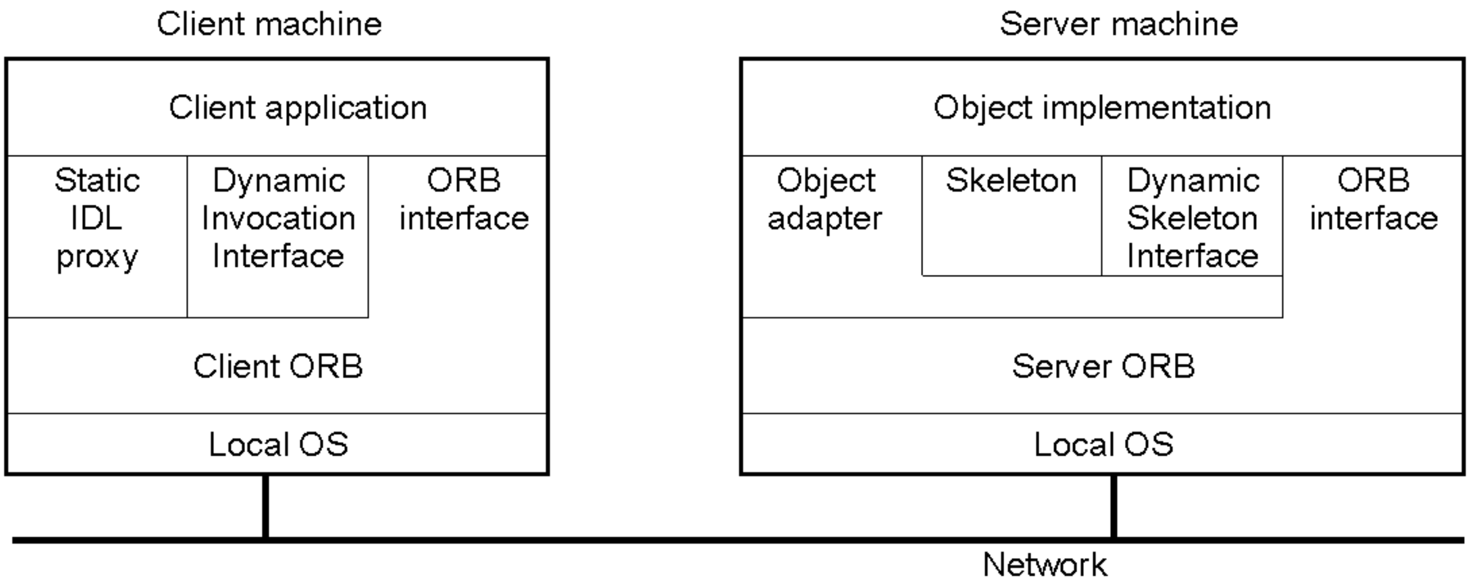


- Object request broker (ORB)
  - Core of the middleware platform
  - Handles communication between objects and clients
  - Handles distribution and heterogeneity issues
  - May be implemented as libraries
- Facilities: composition of CORBA services

Corba Services

Service	Description
Collection	Facilities for grouping objects into lists, queue, sets, etc.
Query	Facilities for querying collections of objects in a declarative manner
Concurrency	Facilities to allow concurrent access to shared objects
Transaction	Flat and nested transactions on method calls over multiple objects
Event	Facilities for asynchronous communication through events
Notification	Advanced facilities for event-based asynchronous communication
Externalization	Facilities for marshaling and unmarshaling of objects
Life cycle	Facilities for creation, deletion, copying, and moving of objects
Licensing	Facilities for attaching a license to an object
Naming	Facilities for systemwide name of objects
Property	Facilities for associating (attribute, value) pairs with objects
Trading	Facilities to publish and find the services on object has to offer
Persistence	Facilities for persistently storing objects
Relationship	Facilities for expressing relationships between objects
Security	Mechanisms for secure channels, authorization, and auditing
Time	Provides the current time within specified error margins

Object Model



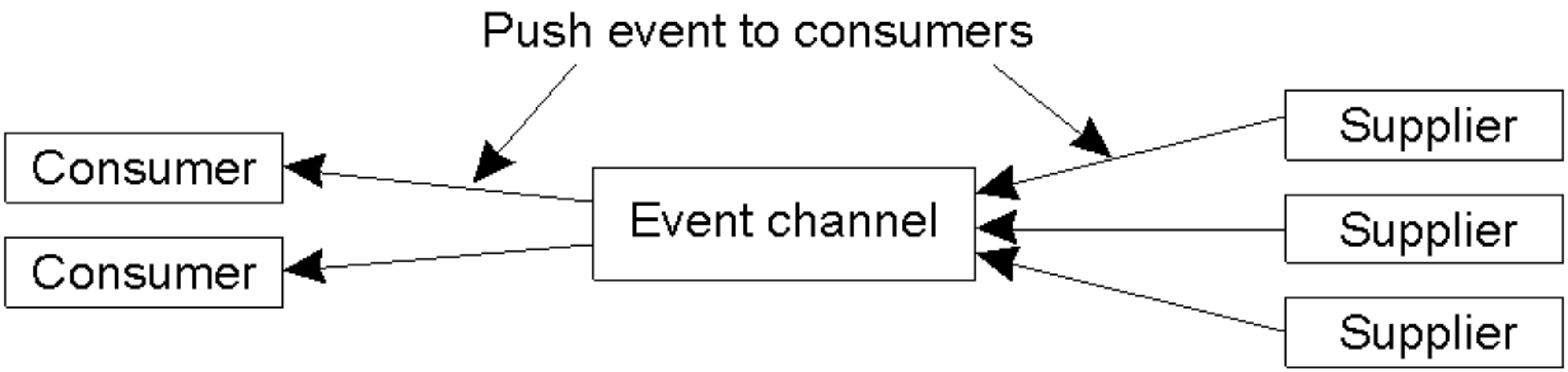
- Objects & services specified using an Interface Definition language (IDL)
  - Used to specify interface of objects and/or services
- ORB: run-time system that handles object-client communication
- Dynamic invocation interface: allows object invocation at run-time
  - Generic *invoke* operation: takes object reference as input
  - Interface repository stores all interface definitions

Object Invocation Models

Request type	Failure semantics	Description
Synchronous	At-most-once	Caller blocks until a response is returned or an exception is raised
One-way	Best effort delivery	Caller continues immediately without waiting for any response from the server
Deferred synchronous	At-most-once	Caller continues immediately and can later block until response is delivered

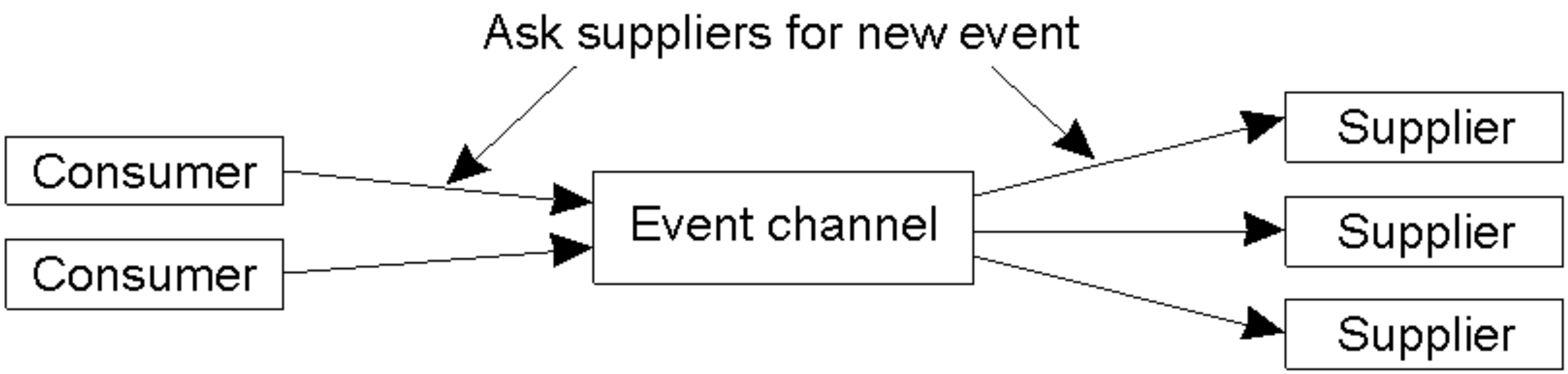
- Invocation models supported in CORBA.
  - Original model was RMI/RPC-like
  - Current CORBA versions support additional semantics

Event and Notification Services (1)



- The logical organization of suppliers and consumers of events, following the push-style model. (**PUB-SUB model**)

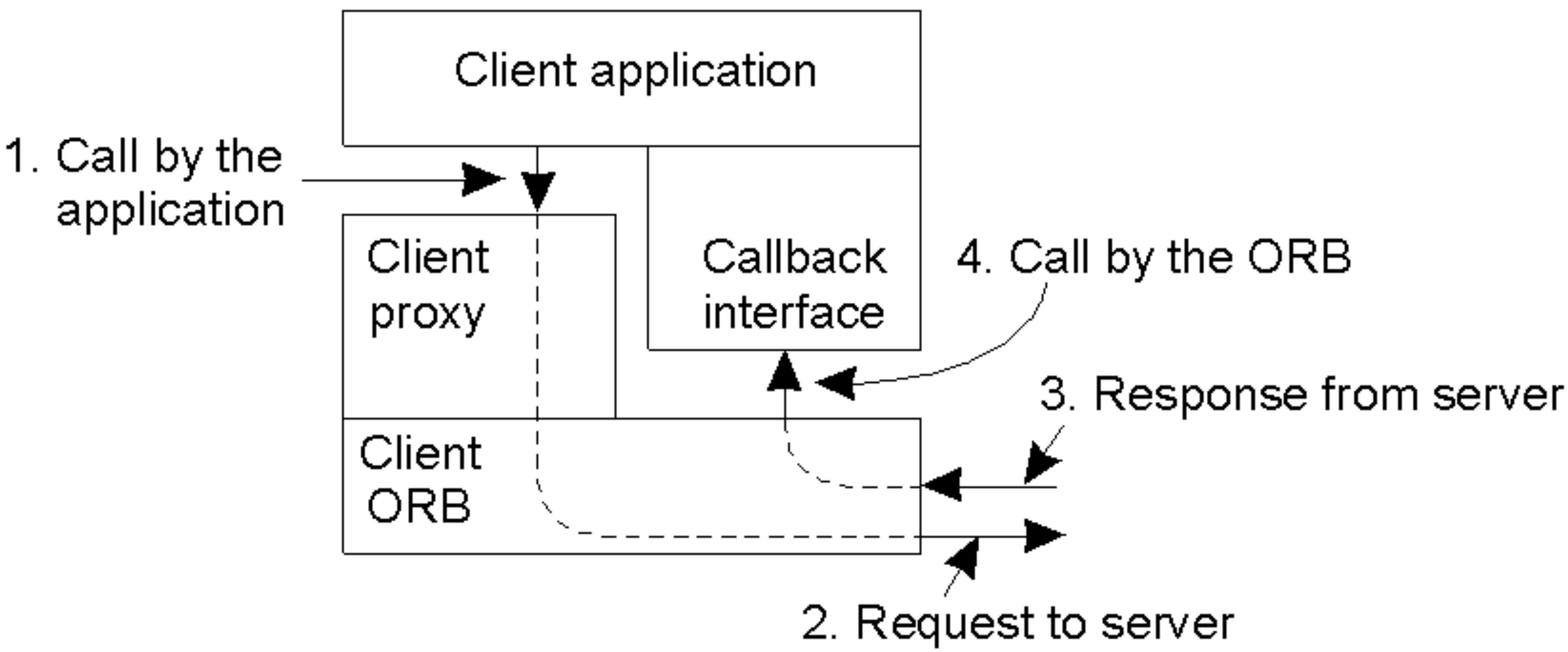
Event and Notification Services (2)



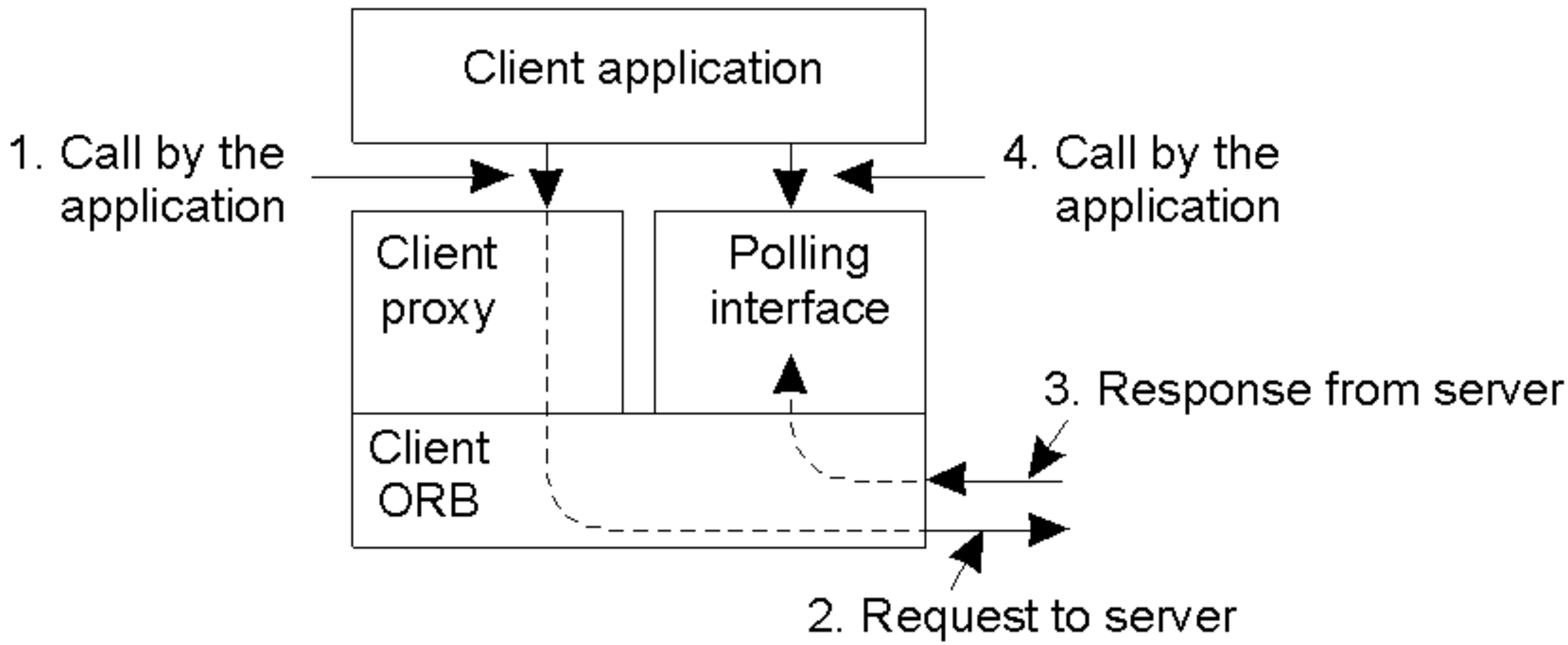
- The pull-style model for event delivery in CORBA.

Messaging: Async. Method Invocation

- CORBA's callback model for asynchronous method invocation.



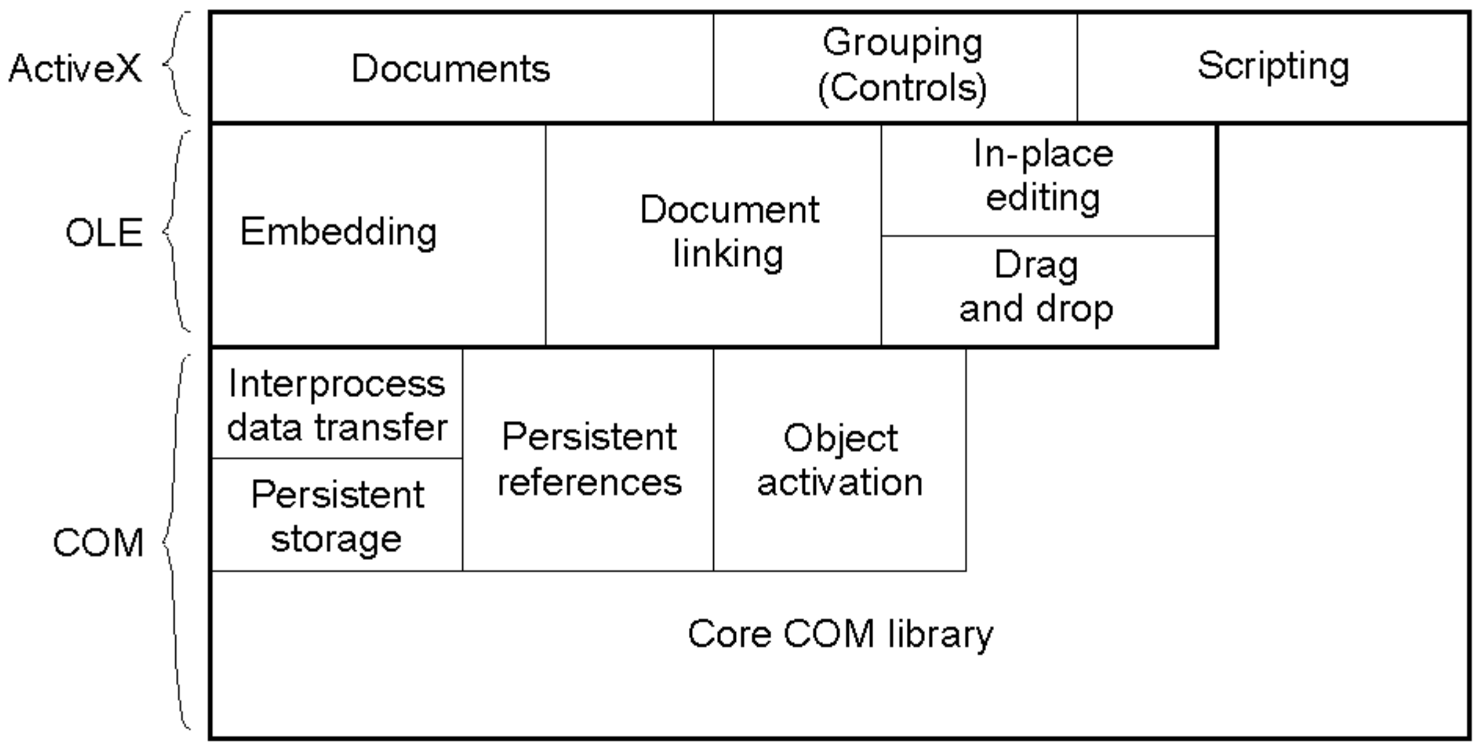
Messaging (2)



- CORBA'S polling model for asynchronous method invocation.

DCOM

- Distributed Component Object Model
  - Microsoft's object model (middleware)
  - Now evolved into .NET

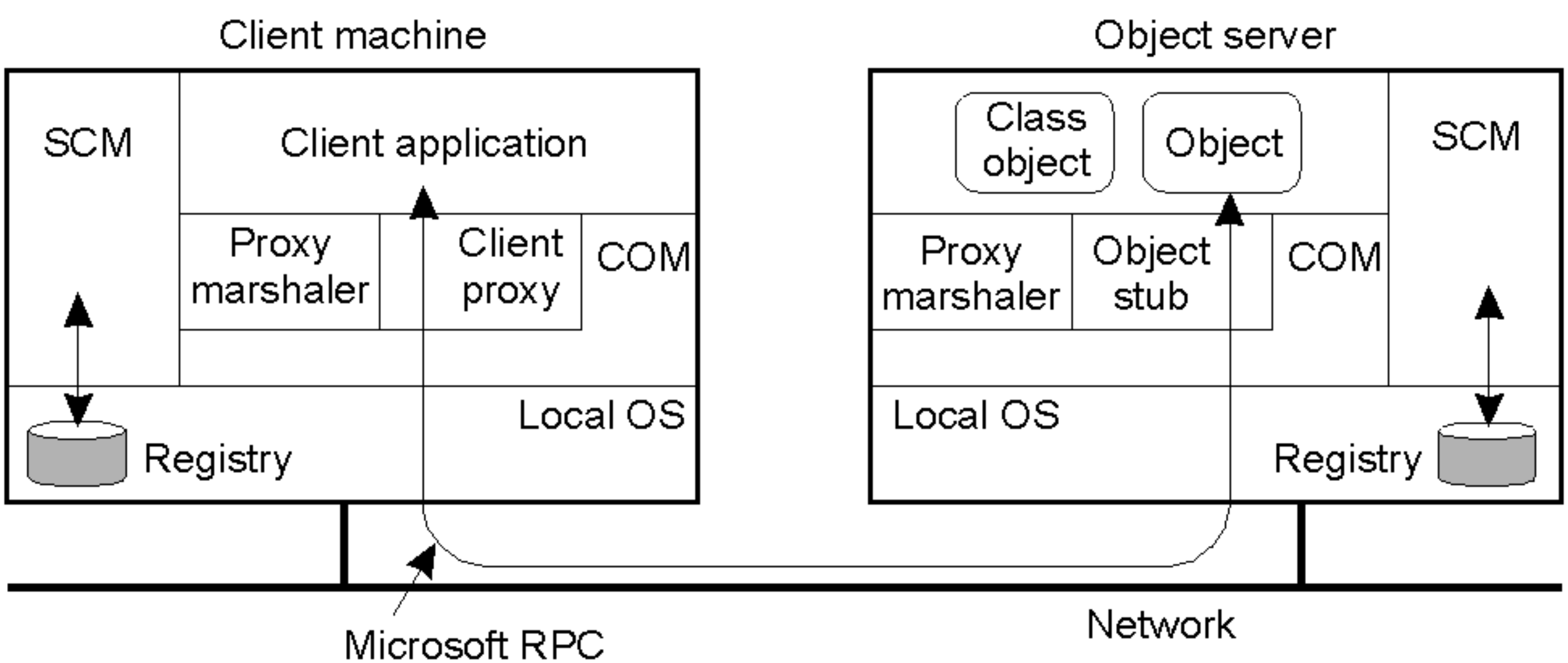


DCOM: History

- Successor to COM
  - Developed to support compound documents
    - Word document with excel spreadsheets and images
- Object linking and embedding (OLE)
  - Initial version: message passing to pass information between parts
  - Soon replaced by a more flexible layer: COM
- ActiveX: OLE plus new features
  - No good consensus on what exactly does ActiveX contain
  - Loosely: groups capabilities within applications to support scripting, grouping of objects.
- DCOM: all of the above, but across machines

Type Library and Registry

- The overall architecture of DCOM.
  - Type library == CORBA interface repository
  - Service control manager == CORBA implementation repository



Monikers: Persistent Objects

Step	Performer	Description
1	Client	Calls BindMoniker at moniker
2	Moniker	Looks up associated CLSID and instructs SCM to create object
3	SCM	Loads class object
4	Class object	Creates object and returns interface pointer to moniker
5	Moniker	Instructs object to load previously stored state
6	Object	Loads its state from file
7	Moniker	Returns interface pointer of object to client

- By default, DCOM objects are transient
- Persistent objects implemented using monikers (reference stored on disk)
  - Has all information to recreate the object at a later time

Monikers (2)

Moniker type	Description
File moniker	Reference to an object constructed from a file
URL moniker	Reference to an object constructed from a URL
Class moniker	Reference to a class object
Composite moniker	Reference to a composition of monikers
Item moniker	Reference to a moniker in a composition
Pointer moniker	Reference to an object in a remote process

- DCOM-defined moniker types.

Distributed Coordination

- Motivation
  - Next generation of systems will be inherently distributed
  - Main problem: techniques to coordinate various components
    - Emphasis on coordination of activities between components

Introduction to Coordination Models

- Key idea: separation of computation from coordination
- A taxonomy of coordination models
  - Direct coordination
  - Mailbox coordination
  - Meeting-oriented coordination (publish/subscribe)
  - Generative (shared tuple space)

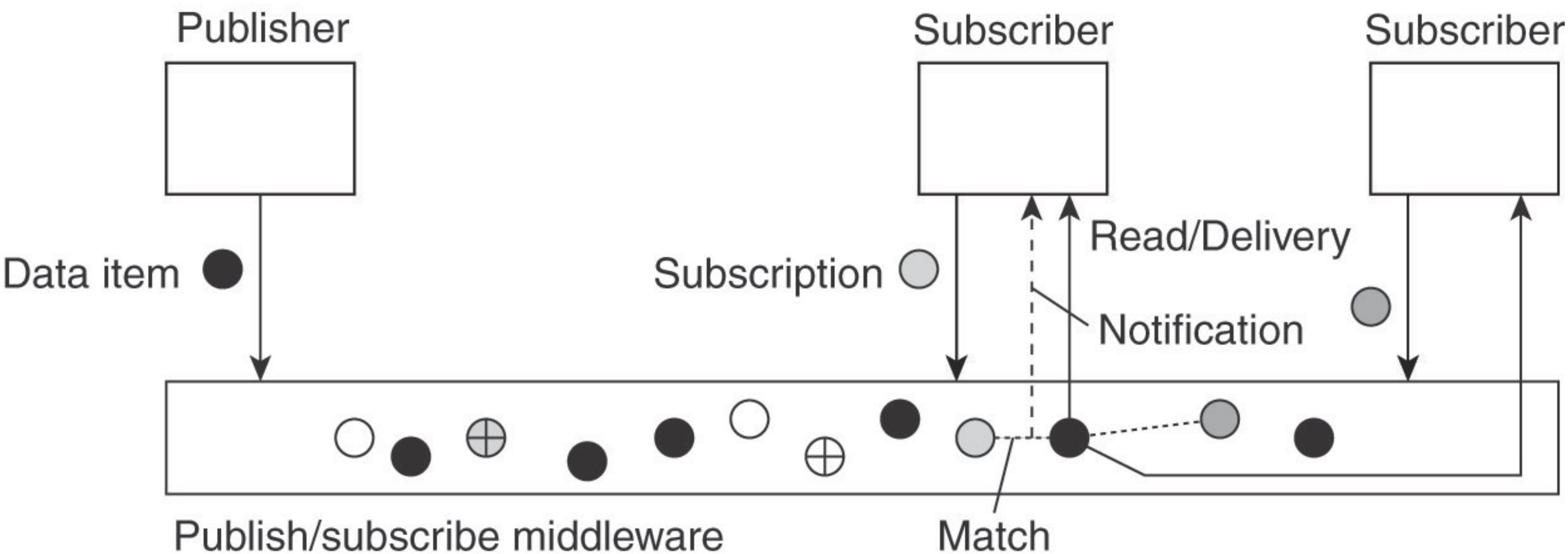
		Temporal: Coupled	Temporal: Uncoupled
Referential	Coupled	Direct	Mailbox
	Uncoupled	Meeting oriented	Generative communication

Jini Case Study

- Coordination system based on Java
  - Clients can *discover* new services as they become available
  - Example: “intelligent toaster”
  - Distributed event and notification system

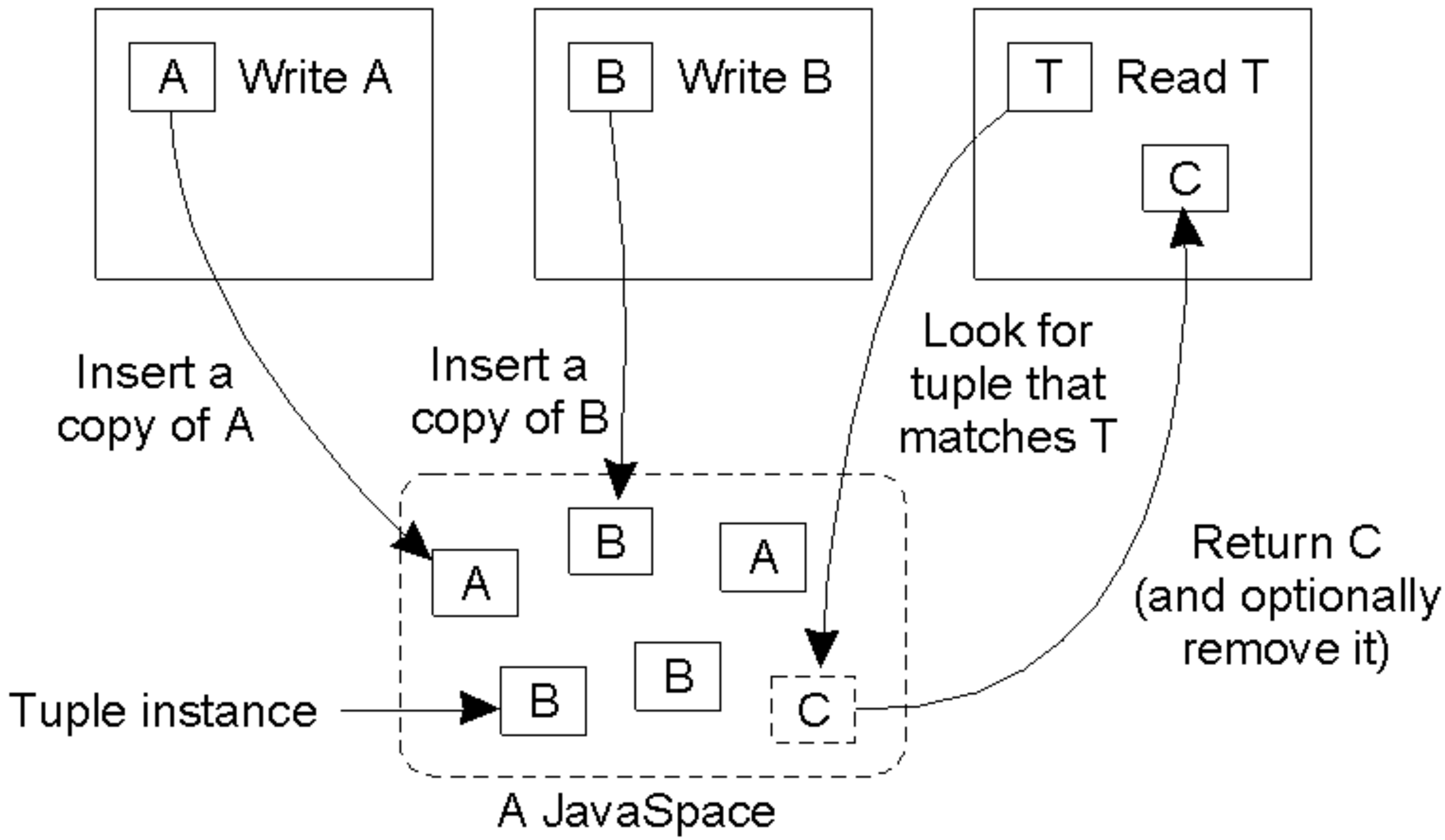
- Coordination model
  - Bulletin board model
  - Uses JavaSpaces: a shared dataspace that stores tuples
    - Each tuple points to a Java object

Overall Approach



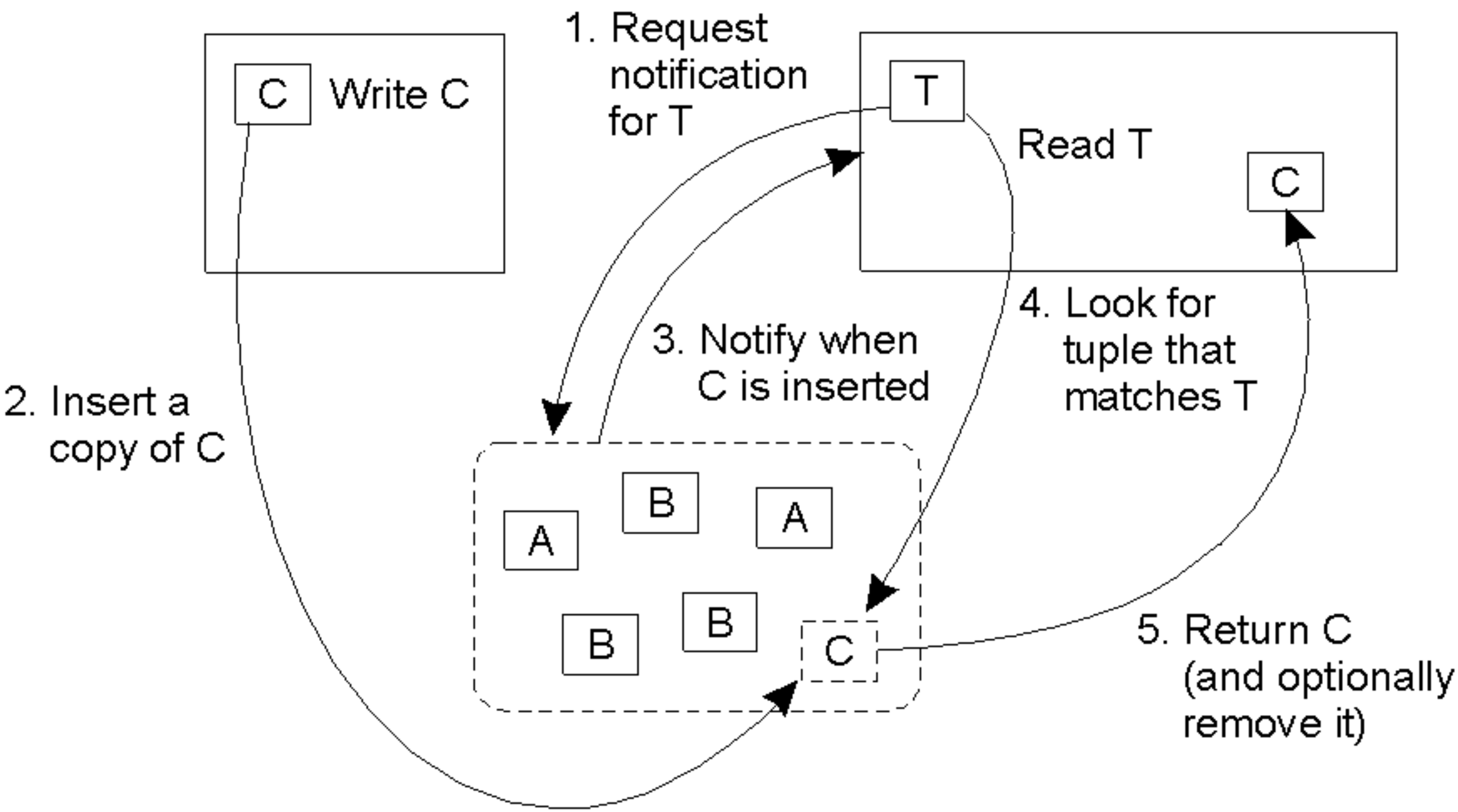
- The principle of exchanging data items between publishers and subscribers.

Overview of Jini



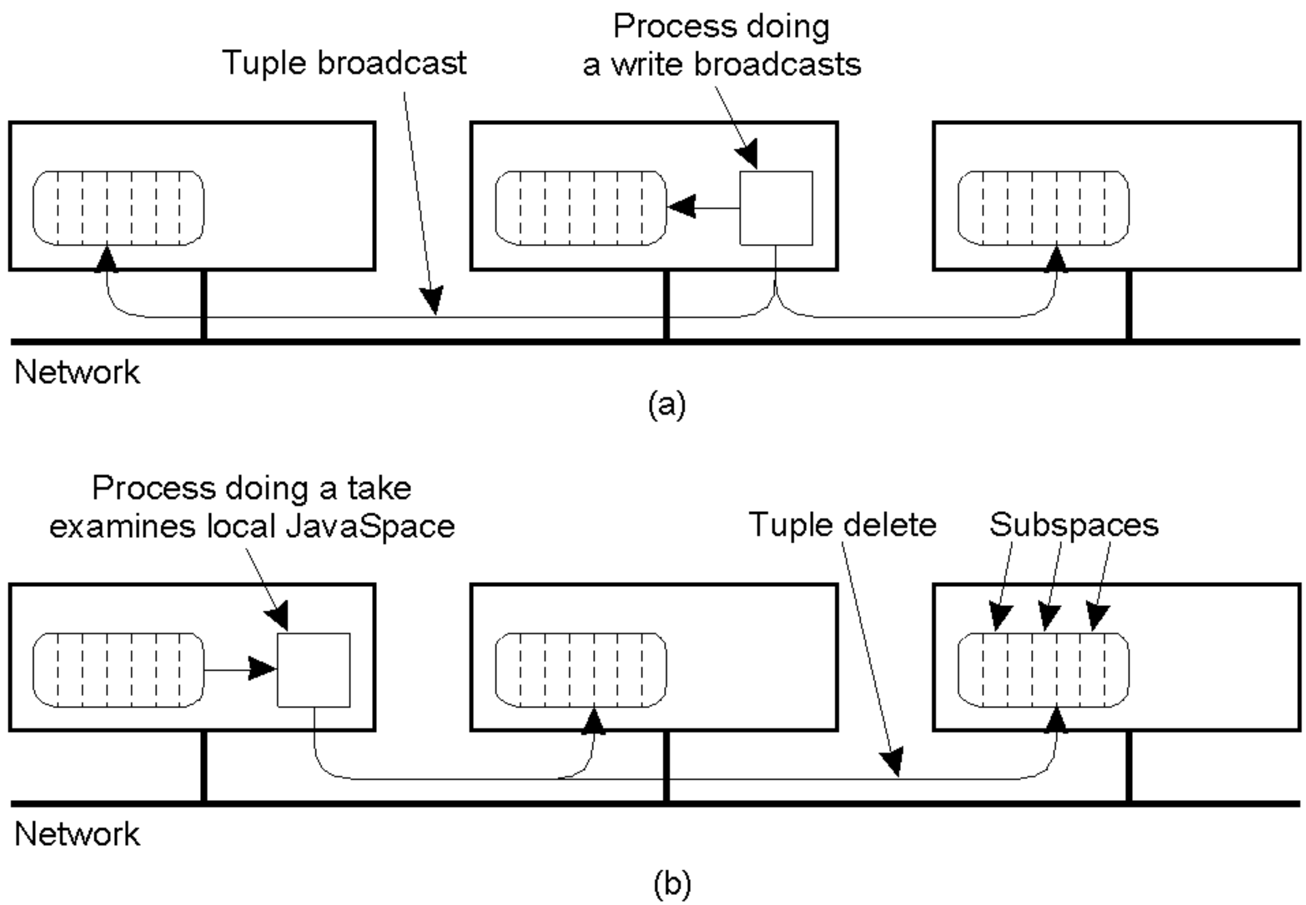
- The general organization of a JavaSpace in Jini.

Communication Events



- Using events in combination with a JavaSpace

Processes (1)

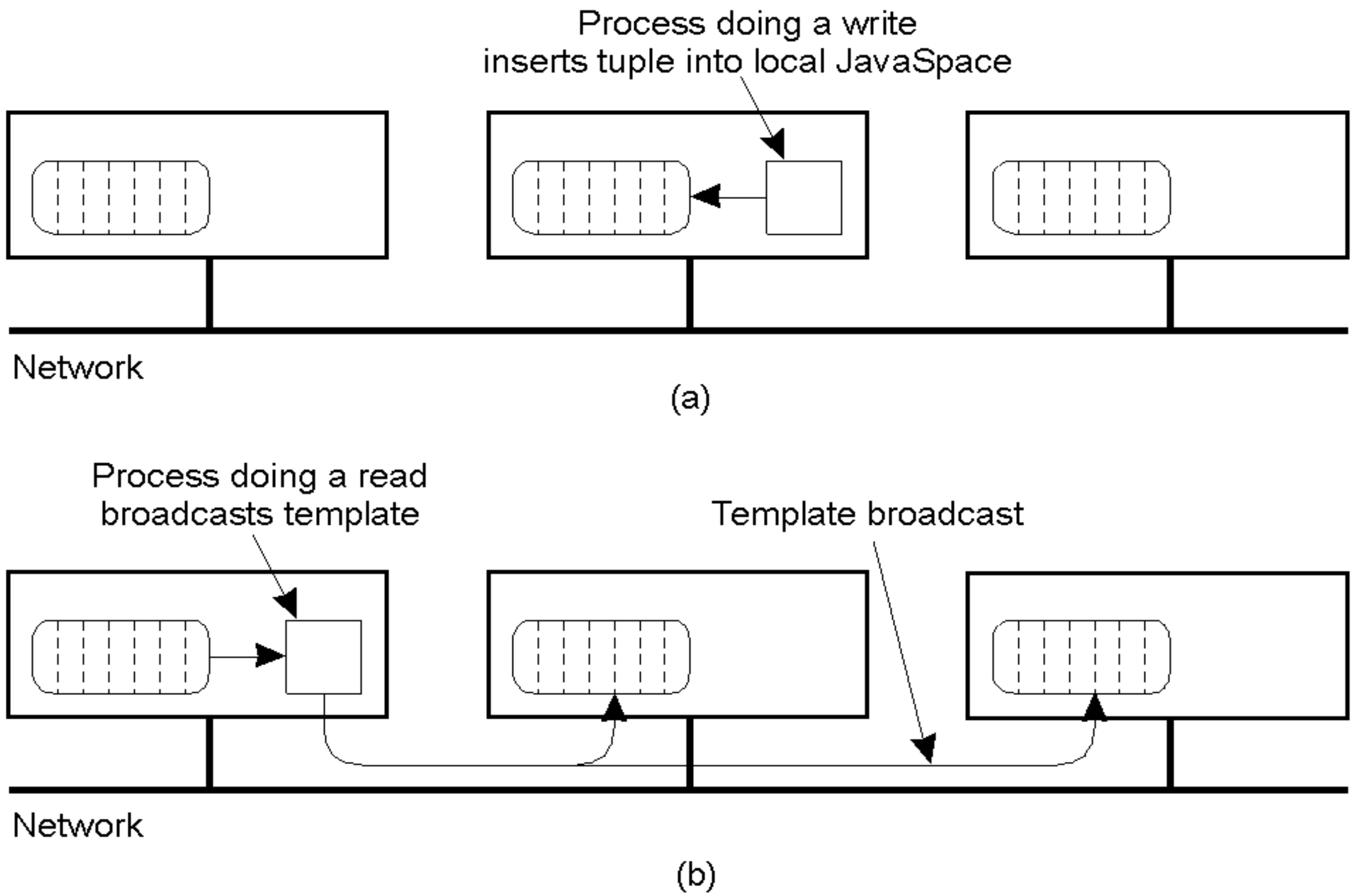


- A JavaSpace can be replicated on all machines. The dotted lines show the partitioning of the JavaSpace into subspaces.

a) Tuples are broadcast on WRITE

b) READs are local, but the removing of an instance when calling TAKE must be broadcast

Processes (2)



- Unreplicated JavaSpace.

a) A WRITE is done locally.

b) A READ or TAKE requires the template tuple to be broadcast in order to find a tuple instance