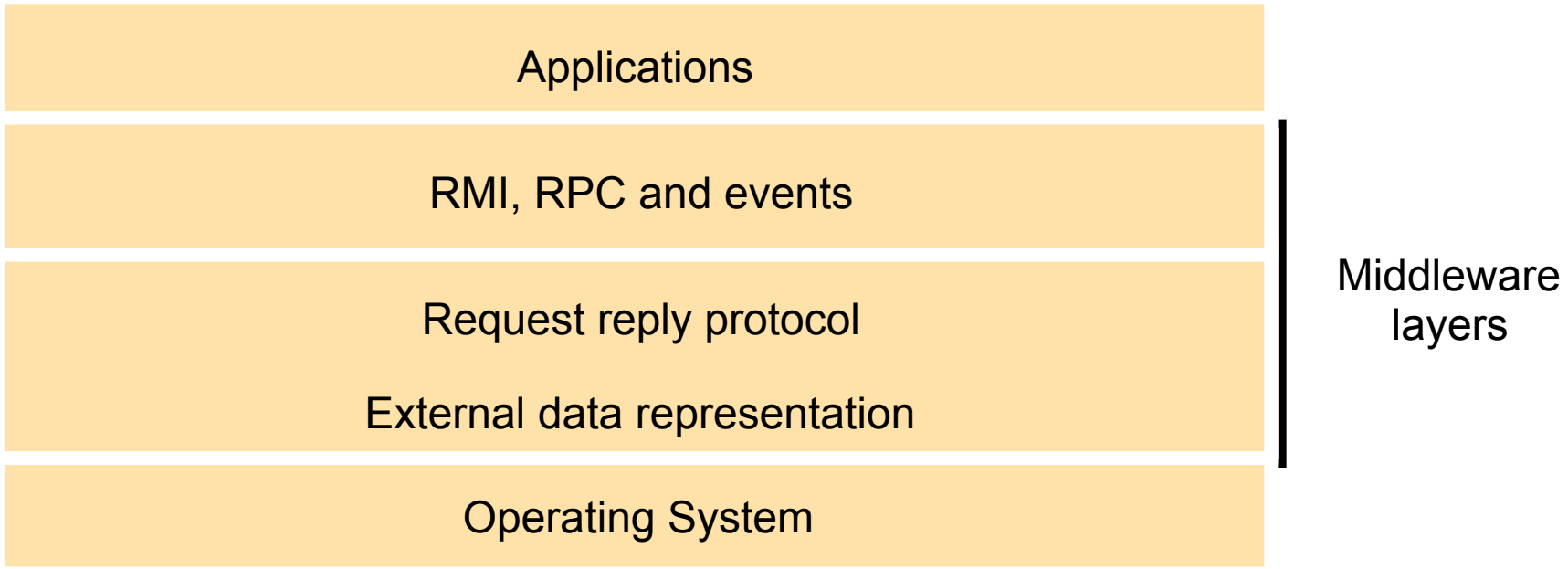
Distributed Systems

Lecture 10: Distributed Objects & Event Notification

Middleware layers



Traditional Interfaces

- Interfaces promote modularity.
- Recall the use of c header files.
- One module may access another module without concern over implementation details.
- Method signatures are specified.
- The compiler need only consider signatures when compiling the caller.

Interface Definition Language

- Definition: An *interface definition language* (IDL) provides a notation for defining interfaces in which each of the parameters of a method may be described as for input or output in addition to having its type specified.
- These may be used to allow objects written in different languages to invoke one another.

Interface Definition Language

- A language independent IDL can be used bridge the gap between programming languages.
- Examples include:
  - Corba IDL (Object-oriented syntax)
  - OSF's Distributed Computing Environment DCE (C like syntax)
  - DCOM IDL based on OSF's DCE and used by Microsoft's DCOM
  - Sun XDR (An IDL for RPC)
  - Web Services WSDL
- In the case of Web Services, how is WSDL different from XSDL?

CORBA IDL example

```
// In file Person.idl
struct Person {
        string name;
        string place;
        long year;
} ;
interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long number();
};
```

How does this compare with WSDL?

File interface in Sun XDR (Originally External Data Representation but now an IDL) for RPC

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
      int length;
      char buffer[MAX];
};
struct writeargs {
      FileIdentifier f;
      FilePointer position;
      Data data;
};
```

```
struct readargs {
      FileIdentifier f;
      FilePointer position;
      Length length;
};

program FILEREADWRITE {
   version VERSION {
      void WRITE(writeargs)=1; // procedure
      Data READ(readargs)=2; // numbers
   }=2;      // version number
} = 9999; // program number
// numbers passed in request message
// rpcgen is the interface compiler
```

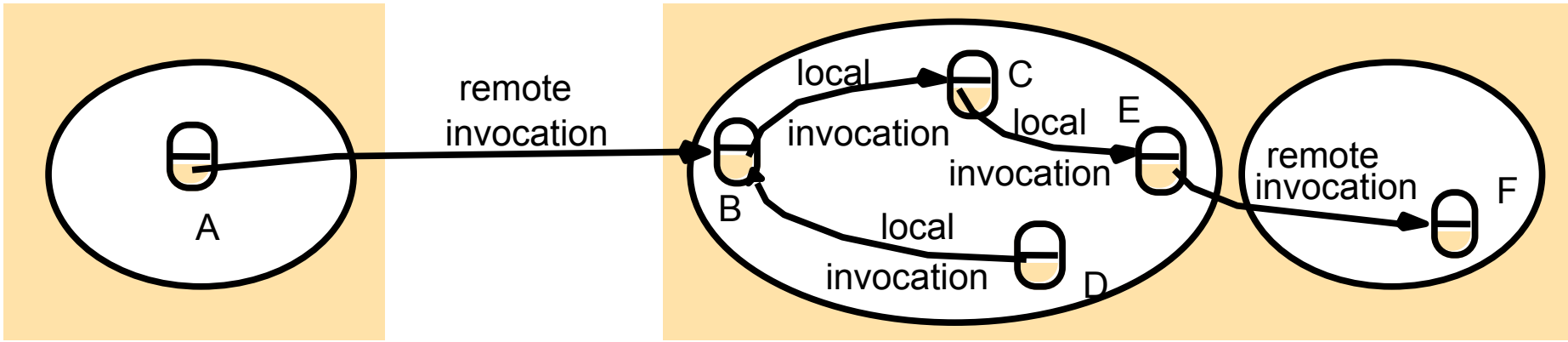
Traditional Object Model

- Each object is a set of data and a set of methods.
- Object references are assigned to variables.
- Interfaces define an object's methods.
- Actions are initiated by invoking methods.
- Exceptions may be thrown for unexpected or illegal conditions.
- Garbage collection may be handled by the developer (C++) or by the runtime (.NET and Java)

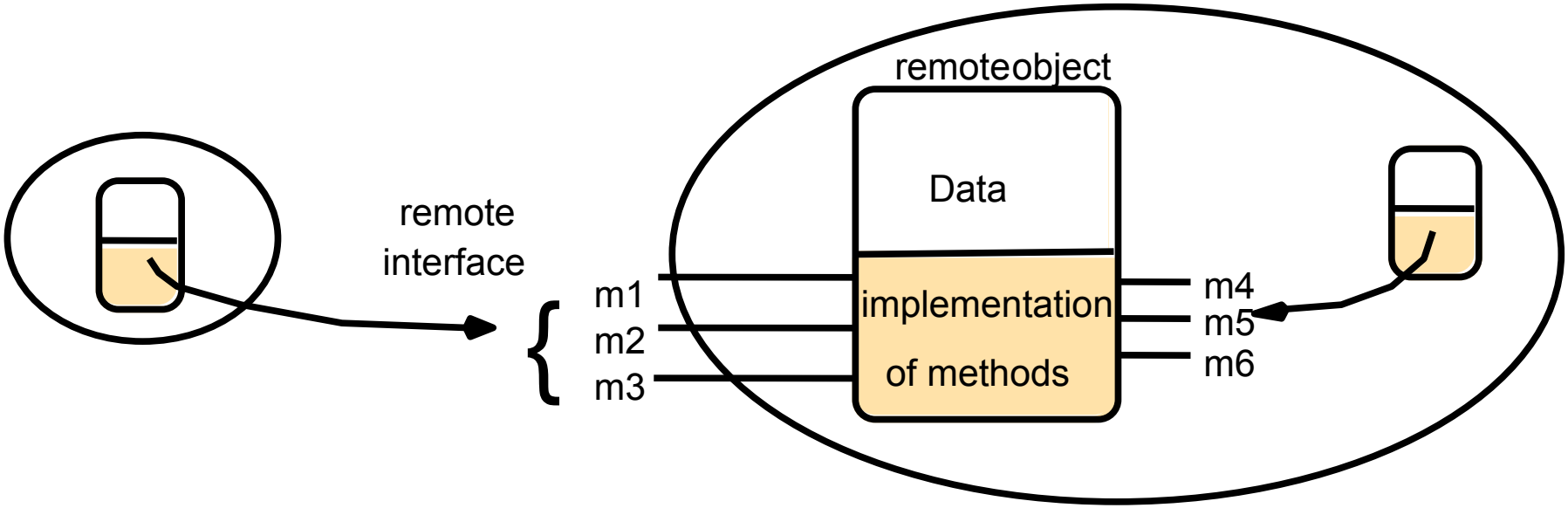
Distributed Object Model

- Having client and server objects in different processes enforces encapsulation. You must call a method to change its state.
- Methods may be synchronized to protect against conflicting access by multiple clients.
- Objects are accessed remotely (by message passing) or objects are copied to the local machine (if the object's class is available locally) and used locally.
- Remote object references are analogous to local ones in that:
  1. The invoker uses the remote object reference to identify the object and
  2. The remote object reference may be passed as an argument to or return value from a local or remote method.

Remote and Local Method Invocations



A Remote Object and its Remote Interface



RMI Design Issues

- RMI Invocation Semantics

  Local calls have Exactly Once semantics. Remote calls have Maybe, At Least Once or at Most Once semantics. Different semantics are due to the fault tolerance measures applied during the request reply protocol.

- Level of Transparency

  Remote calls should have a syntax that is close to local calls.

  But it should probably be clear to the programmer that a remote call is being made.

Invocation Semantics

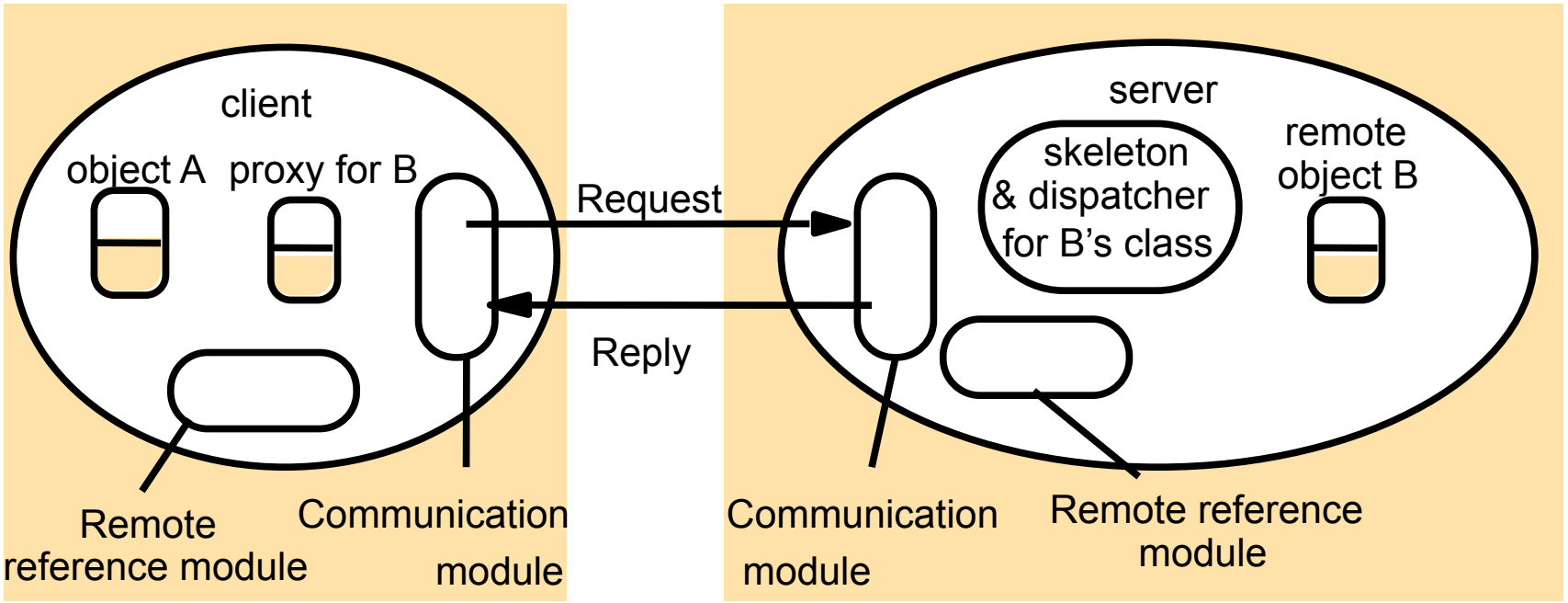
Fault tolerance measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply (history)	*At-most-once*

Duplicate filtering means removing duplicate request at the server.

Invocation Semantics

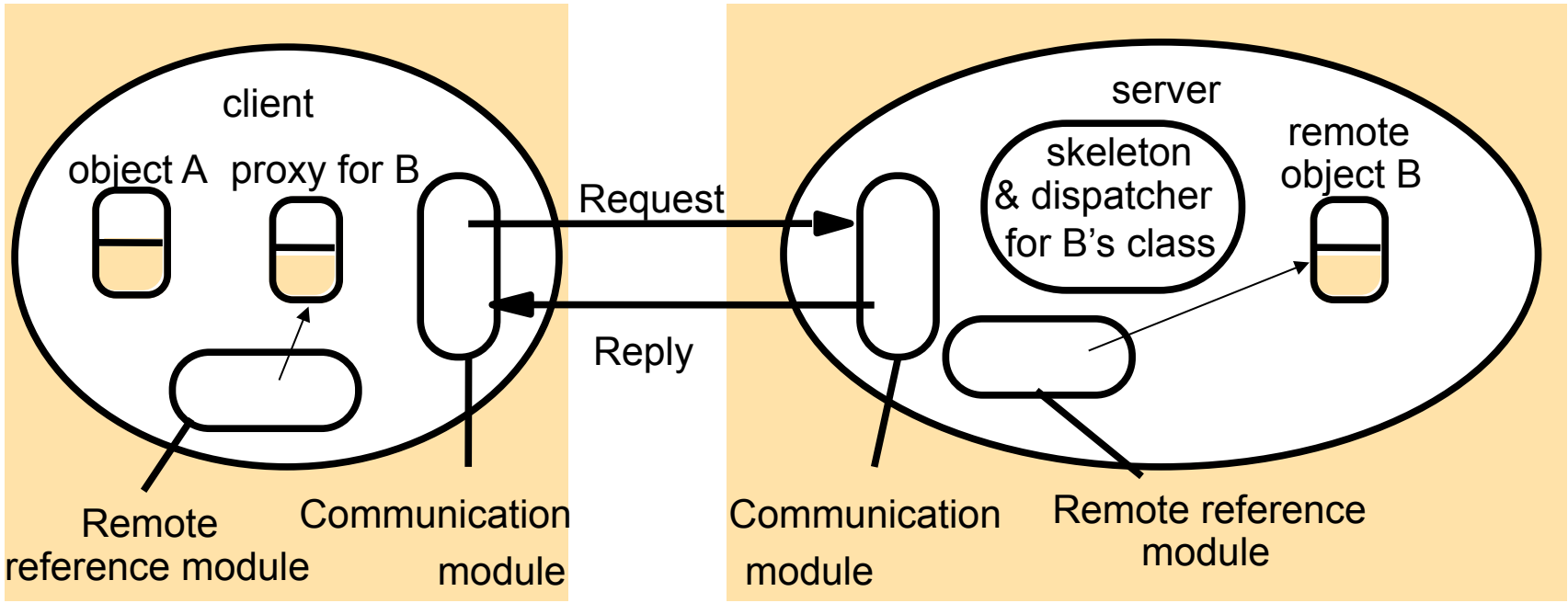
- Maybe semantics is useful only for applications in which occasional failed invocations are acceptable.
- At-Least-Once semantics is appropriate for idempotent operations.
- At-Most-Once semantics is the norm.

Generic RMI Modules



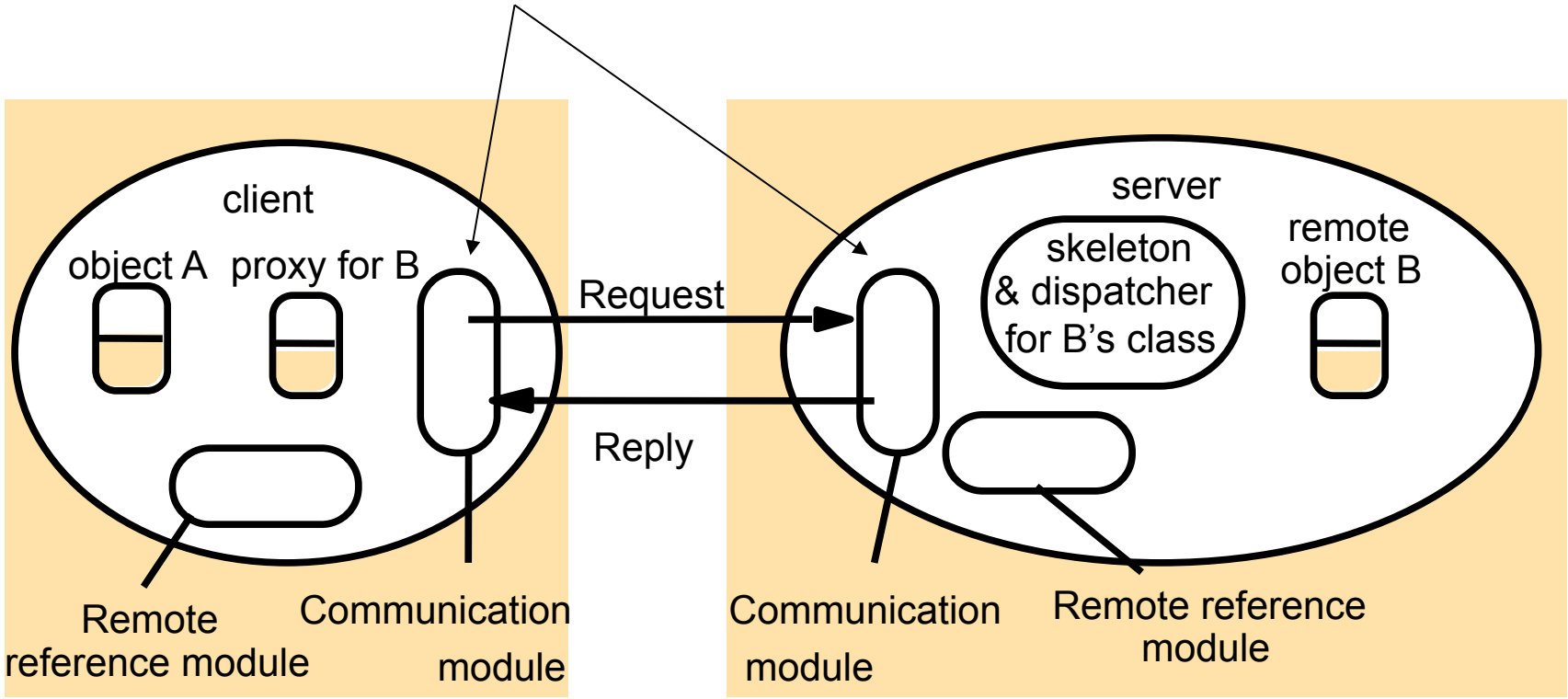
The Remote Reference Module

The remote reference module holds a table that records the correspondence between local object references in that process and remote object references (which are system wide).



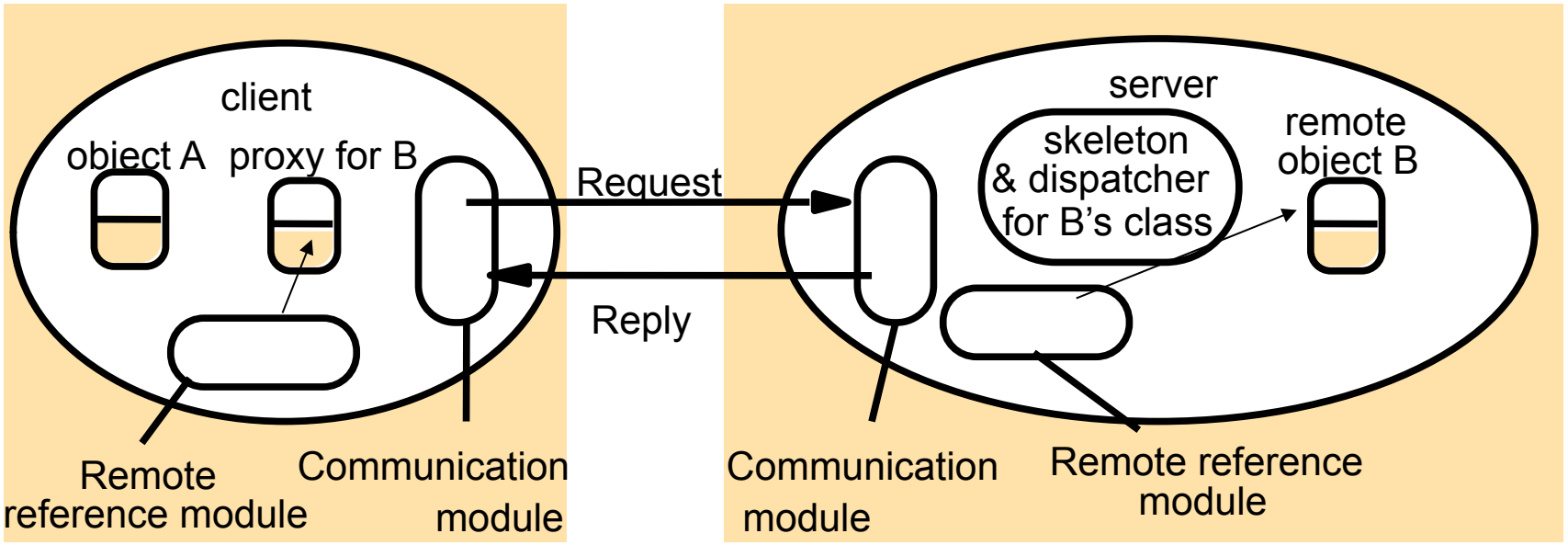
The Communication Module

Coordinate to provide a specified invocation semantics. The communication module selects the dispatcher for the class of the object to be invoked, passing on the remote object's local reference.



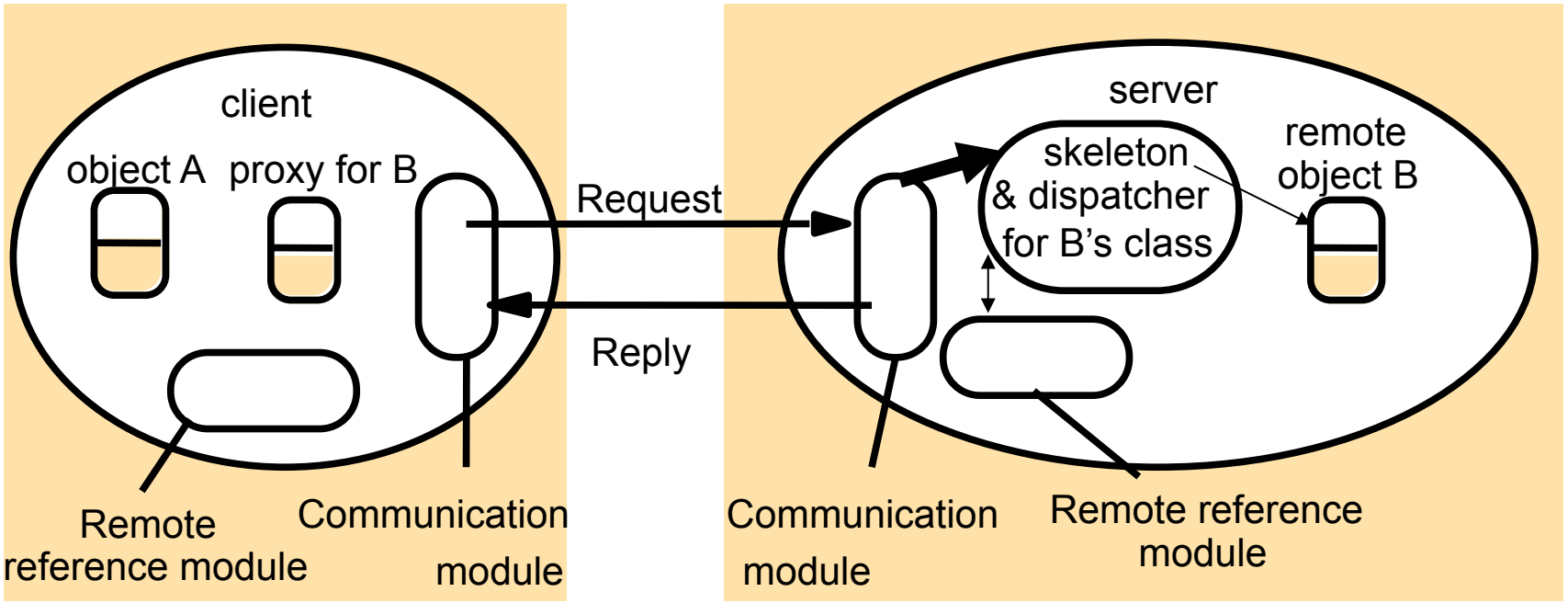
Proxies

The proxy makes the RMI transparent to the caller. It marshals and unmarshals parameters. There is one proxy for each remote object. Proxies hold the remote object reference.



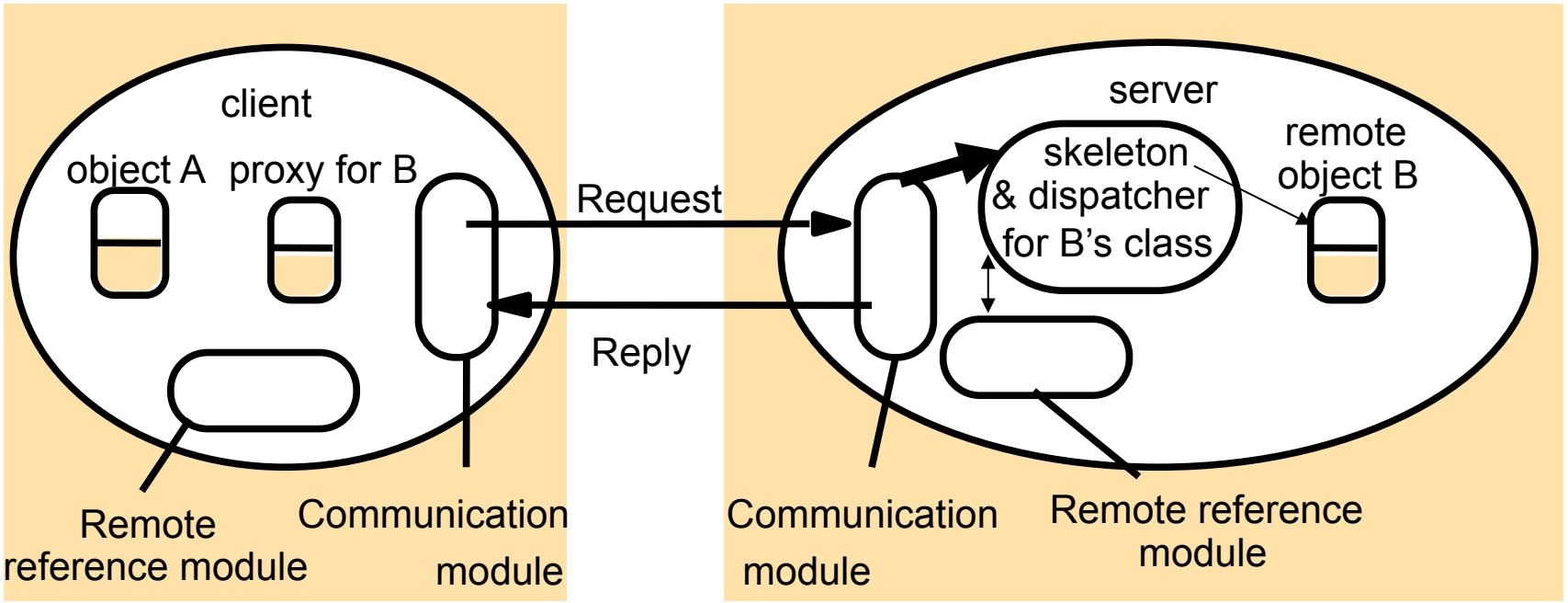
Dispatchers and Skeletons (1)

The server has one dispatcher and skeleton for each class representing a remote object. A request message with a methodID is passed from the communication module. The dispatcher calls the method in the skeleton passing the request message. The skeleton implements the remote object's interface in much the same way that a proxy does. The remote reference module may be asked for the local location associated with the remote reference.

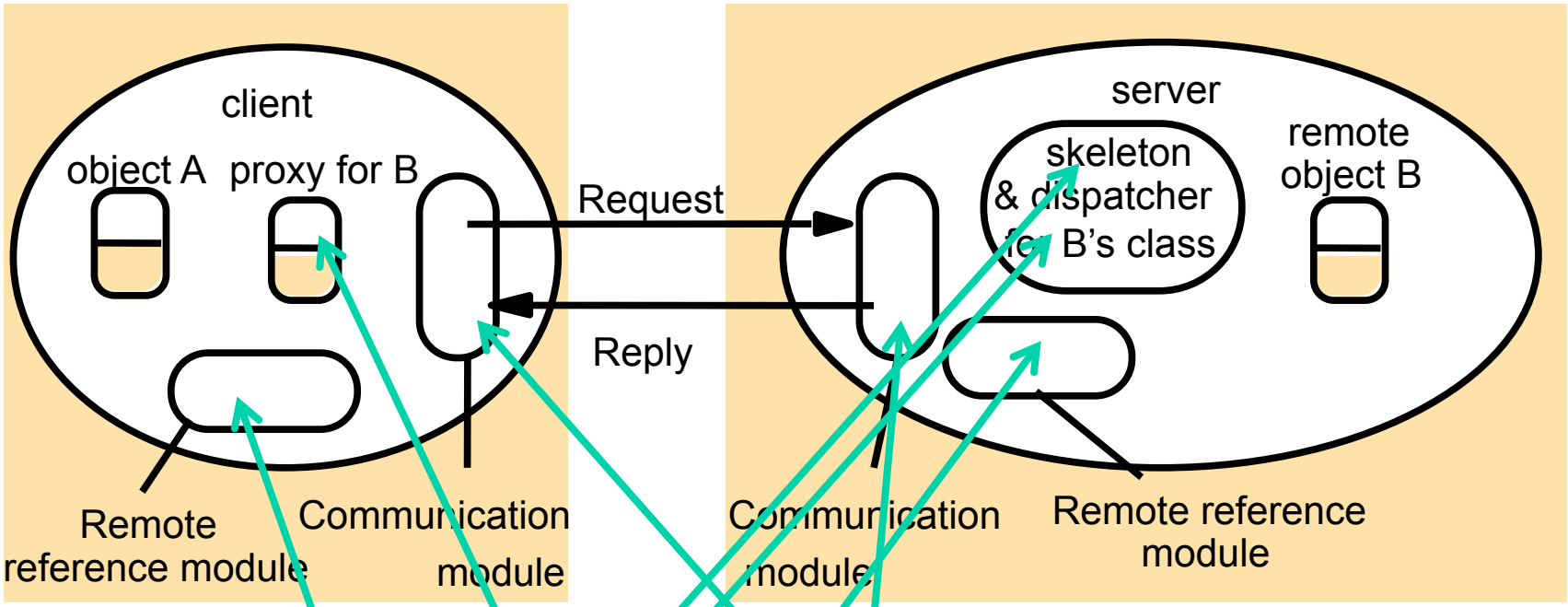


Dispatchers and Skeletons (2)

The communication module selects the dispatcher based upon the remote object reference. The dispatcher selects the method to call in the skeleton. The skeleton unmarshalls parameters and calls the method in the remote object.



Generic RMI Summary



Proxy - makes RMI transparent to
remote interface. Marshals reques
results. Forwards request.

carries out Request-reply protocol

translates between local and remote object references and creates remote object references. Uses remote object table

software - between
lication level objects

terface.
nvokes

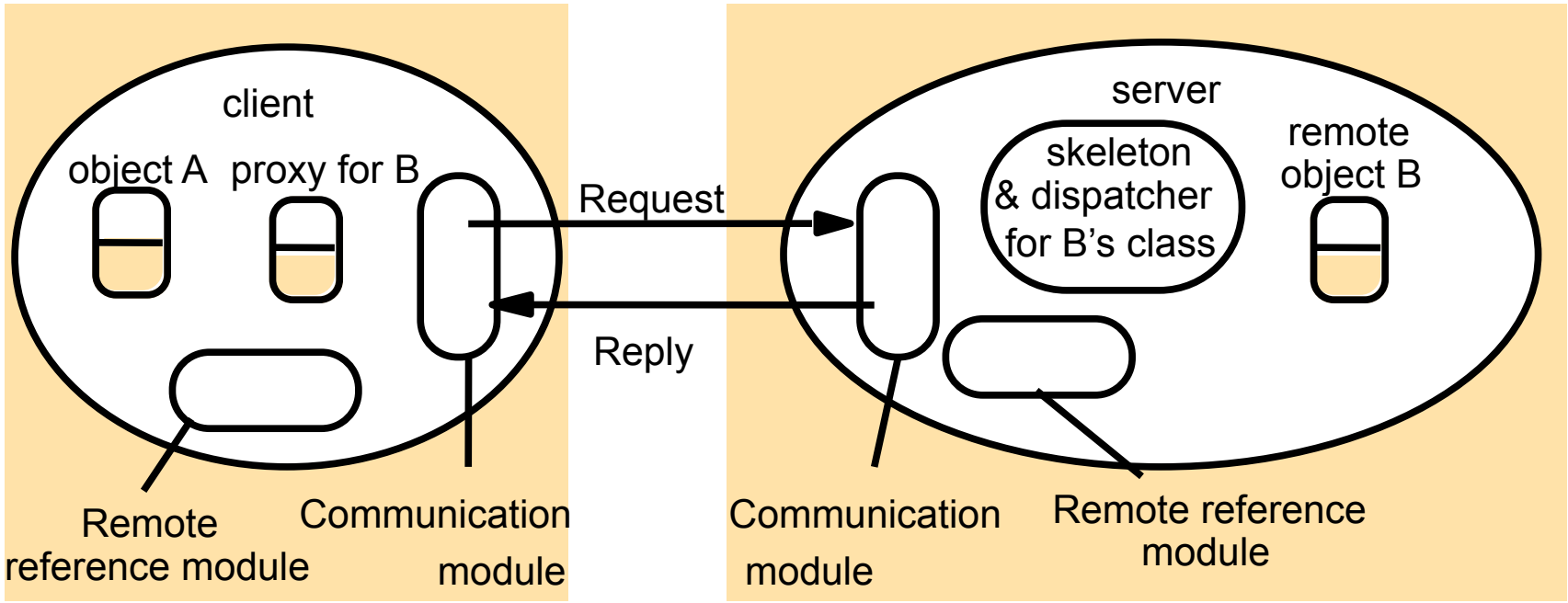
communication and
te reference modules

Binders

Java uses the rmiregistry

CORBA uses the CORBA Naming Service

Binders allow an object to be named and registered.



Local Events and Notifications

- Examples of the local event model:

  (1) A keystroke causes an interrupt handler to execute, storing a key character in the keyboard buffer.

  (2) A mouse click causes an interrupt handler to call a registered listener to handle the mouse event.

Distributed Event Based System

- Suppose a whiteboard server is willing to make calls to all registered clients when the drawing is changed by any one client.
- Clients may subscribe to this service (register interest).
- The whiteboard server publishes the events that it will make available to clients.
- This is the publish-subscribe paradigm

Two Characteristics of Distributed Event Based Systems

(1) Heterogeneous

-- event generators publish the types of events they offer

-- other objects subscribe and provide callable methods

-- components that were not designed to work together may interoperate

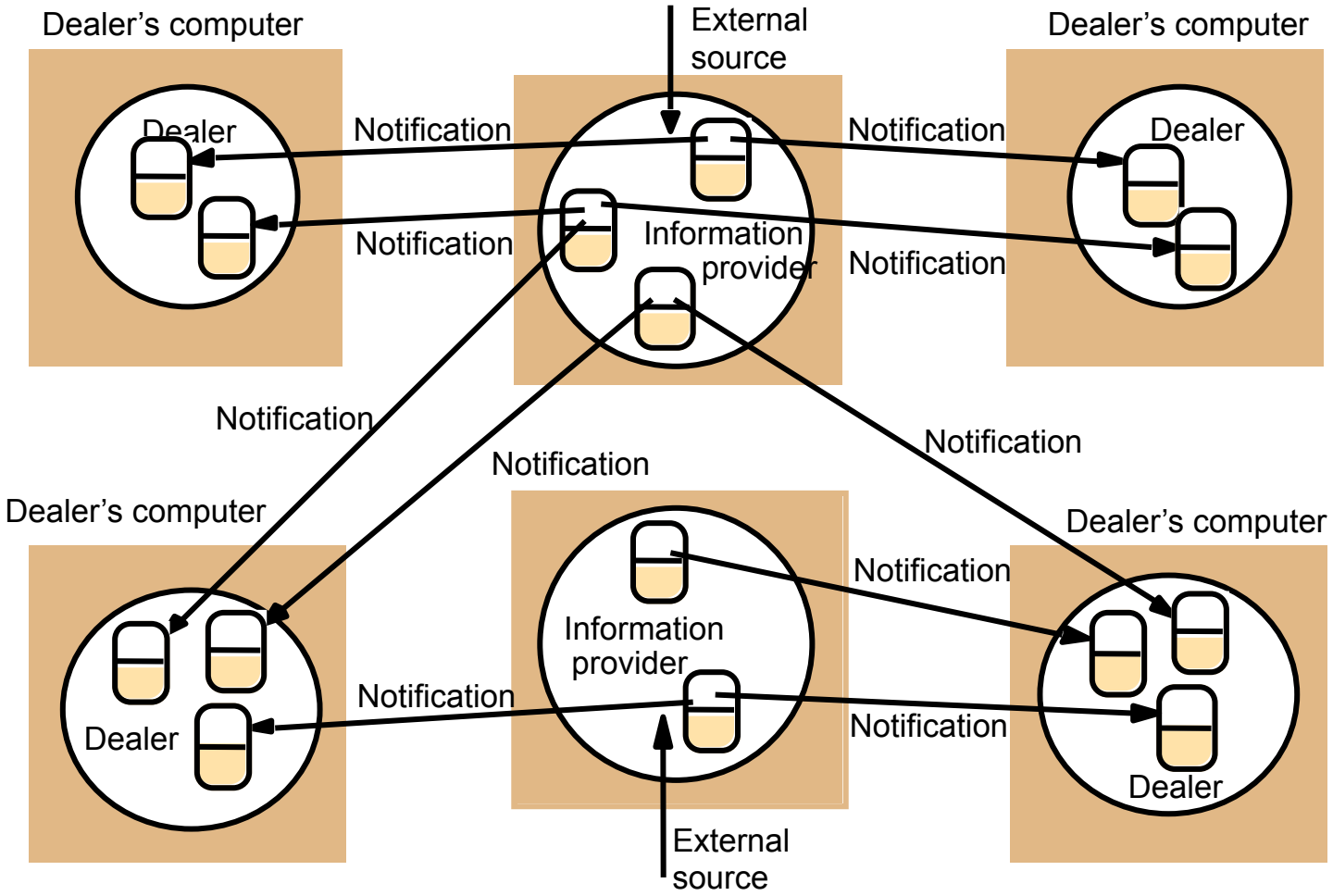
Two Characteristics of Distributed Event Based Systems

(2) Asynchronous

-- Publishers and subscribers are decoupled

-- notifications of events are sent asynchronously to all subscribers

Dealing room system



Architecture for distributed event notification