CS425 /CSE424/ECE428 – Distributed Systems – Fall 2011

Remote Procedure Calls & Distributed Objects

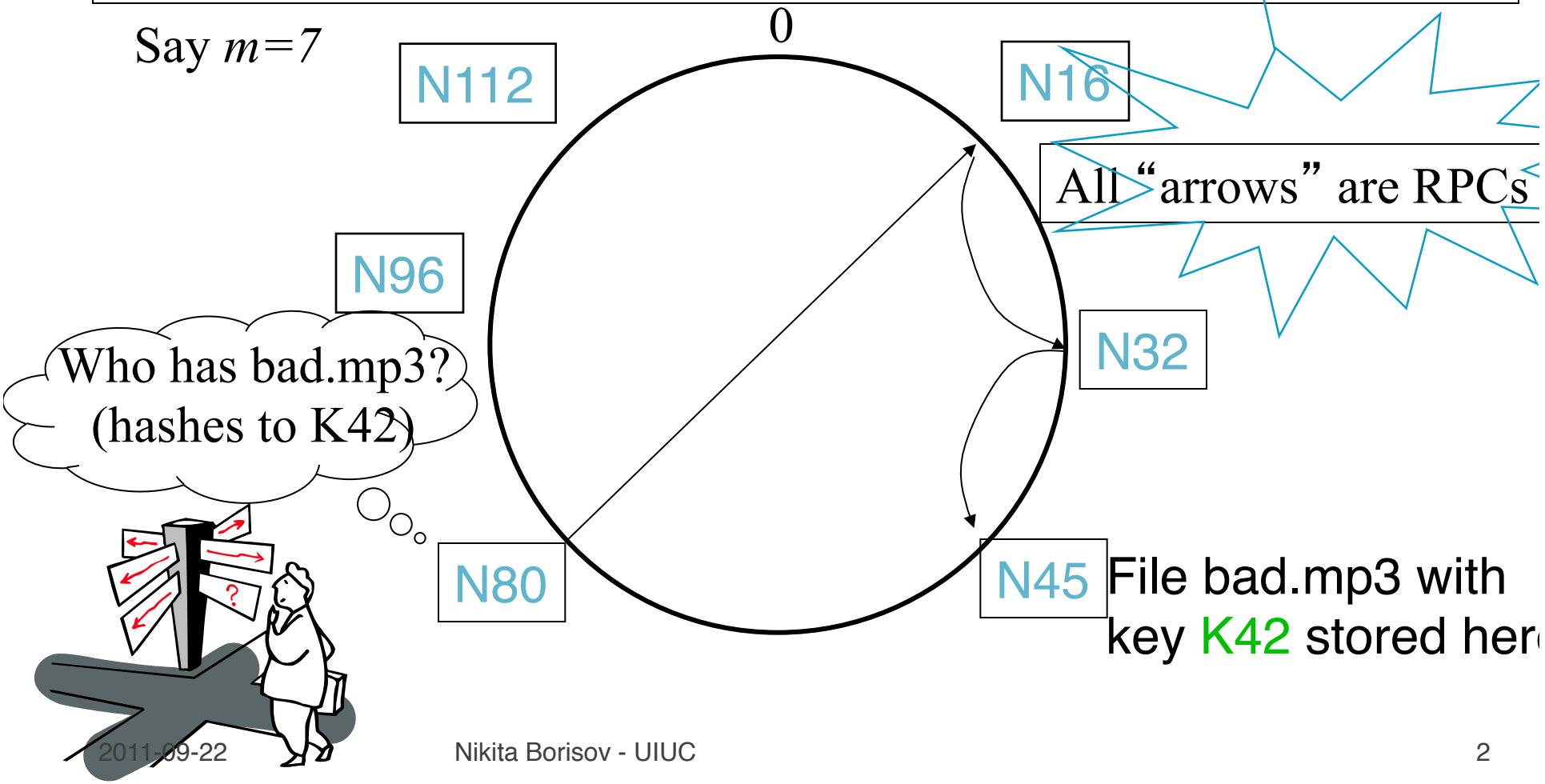
Material derived from slides by I. Gupta, M. Harandi, J. Hou, S. Mitra, K. Nahrstedt, N. Vaidya

Search in Chord

What are "*RPCs*"?

At node *n*, send query for key *k* to largest successor/finger entry < *k*
if none exist, return *successor(n)* to requestor

Say *m*=7



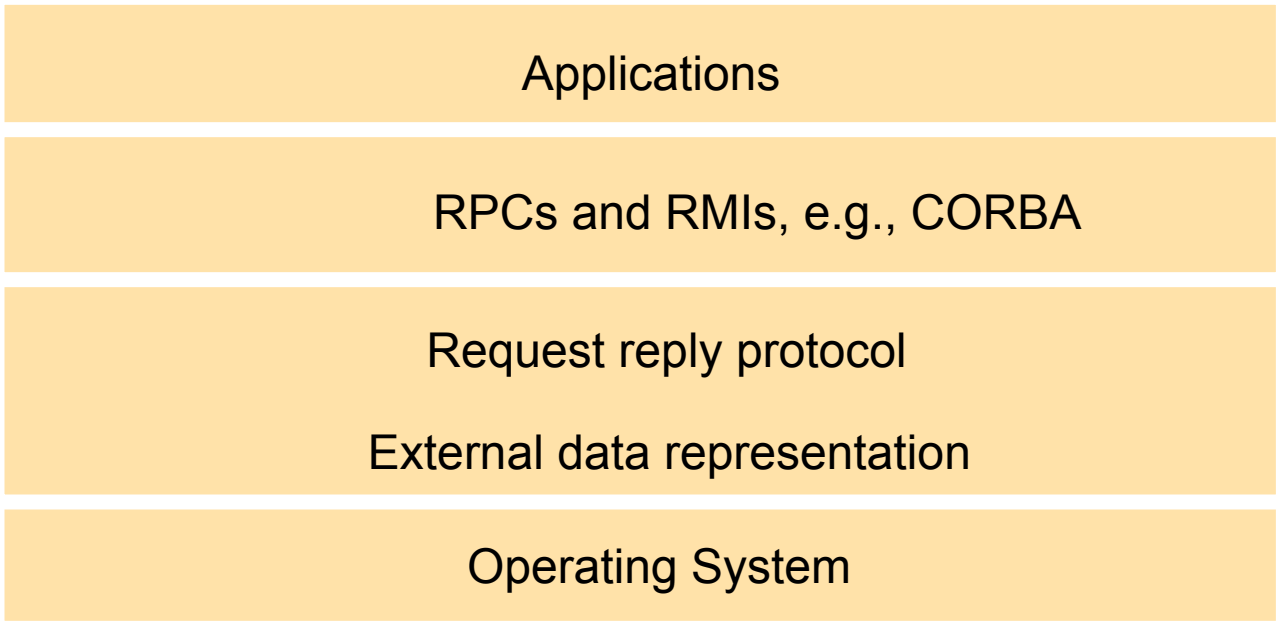
Bank Database Example

How are "*transactions*" executed between a client ATM and a bank server?

- Bank Database: Think of two simultaneous deposits of $10,000 into your bank account, each from one ATM.
  - Both ATMs read initial amount of $1000 concurrently from the bank server
  - Both ATMs add $10,000 to this amount (locally at the ATM)
  - Both write the final amount to the server
  - What's wrong?

- The ATMs need mutually exclusive access to your account entry at the server

Middleware Layers



Middleware layers= *Provide support to the application*\

Run at all servers @user level

RMI=Remote Method Invocation
CORBA=Common Object Request Brokerage Architecture

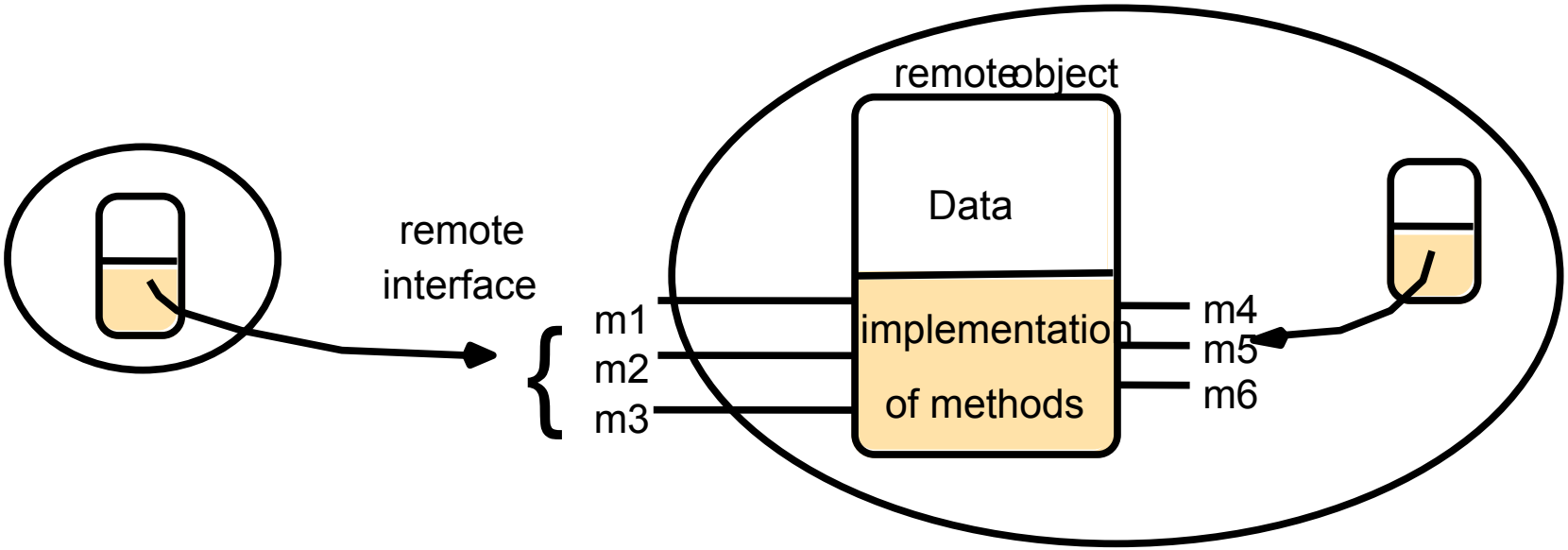
Local Objects

- Within one process's address space
- Object
  - consists of a set of data and a set of methods.
  - E.g., C++/Java object
- Object reference
  - an identifier via which objects can be accessed.
  - i.e., a pointer (C++)
- Interface
  - Signatures of methods
    - Types of arguments, return values, exceptions
  - No implementation
  - E.g., hash table:
    - insert(key, value)
    - value = get(key)
    - remove(key)

Remote Objects

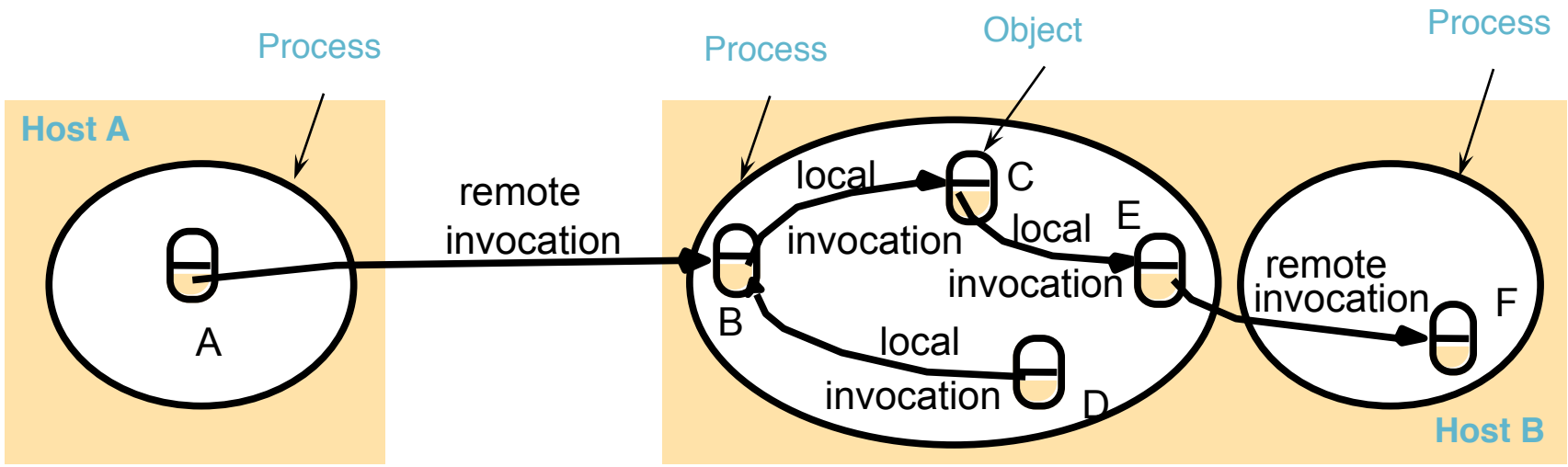
- May cross multiple process's address spaces
- Remote method invocation
  - method invocations between objects in different processes (processes may be on the same or different host).
  - *Remote Procedure Call (RPC): procedure call between functions on different processes in non-object-based system*
- Remote objects
  - objects that can receive remote invocations.
- Remote object reference
  - an identifier that can be used globally throughout a distributed system to refer to a particular unique remote object.
- Remote interface
  - Every remote object has a remote interface that specifies which of its methods can be invoked remotely. E.g., CORBA interface definition language (IDL).

A Remote Object and Its Remote Interface



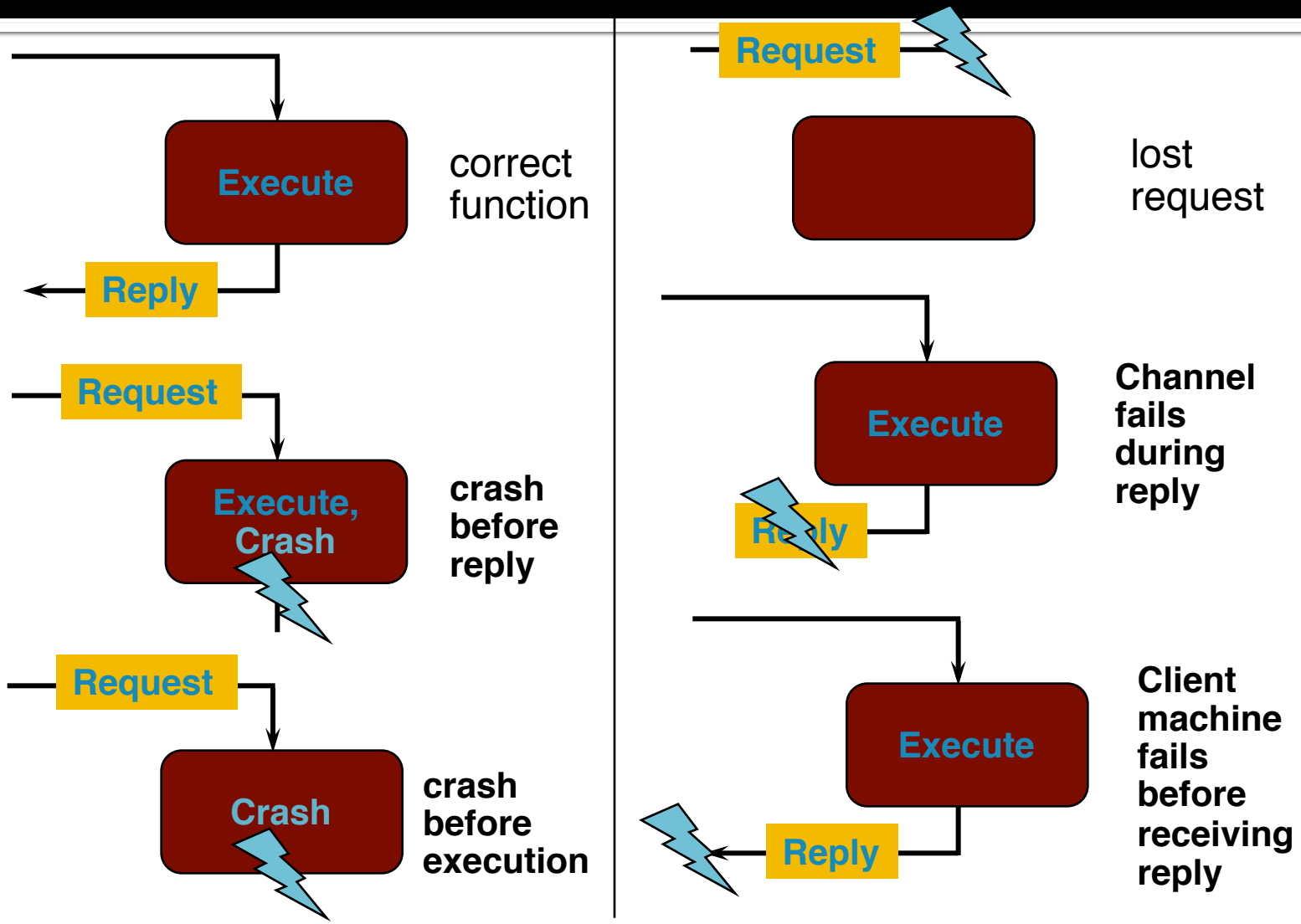
Example Remote Object reference=(IP,port,objectnumber,signature,time)

Remote and Local Method Invocations



Local invocation=between objects on same process.
Has *exactly once* semantics
Remote invocation=between objects on different processes.
Ideally also want *exactly once* semantics for remote invocations
But difficult (why?)

Failure Modes of RMI/RPC



(and if request is received more than once?)

Invocation Semantics

Whether or not to retransmit the request message until either a reply is received or the server is assumed to be failed

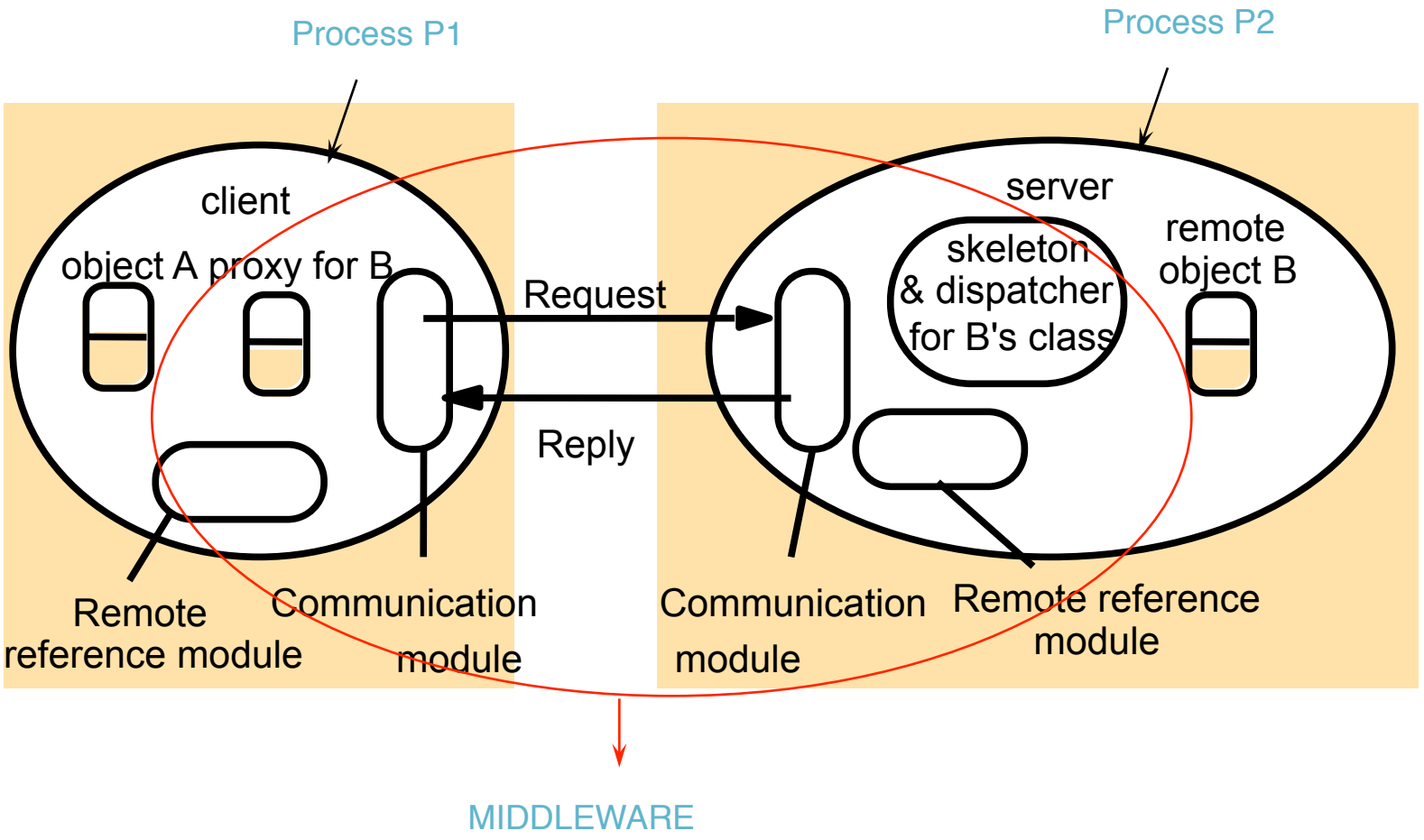
when retransmissions are used, whether to filter out duplicate requests at the server.

whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations

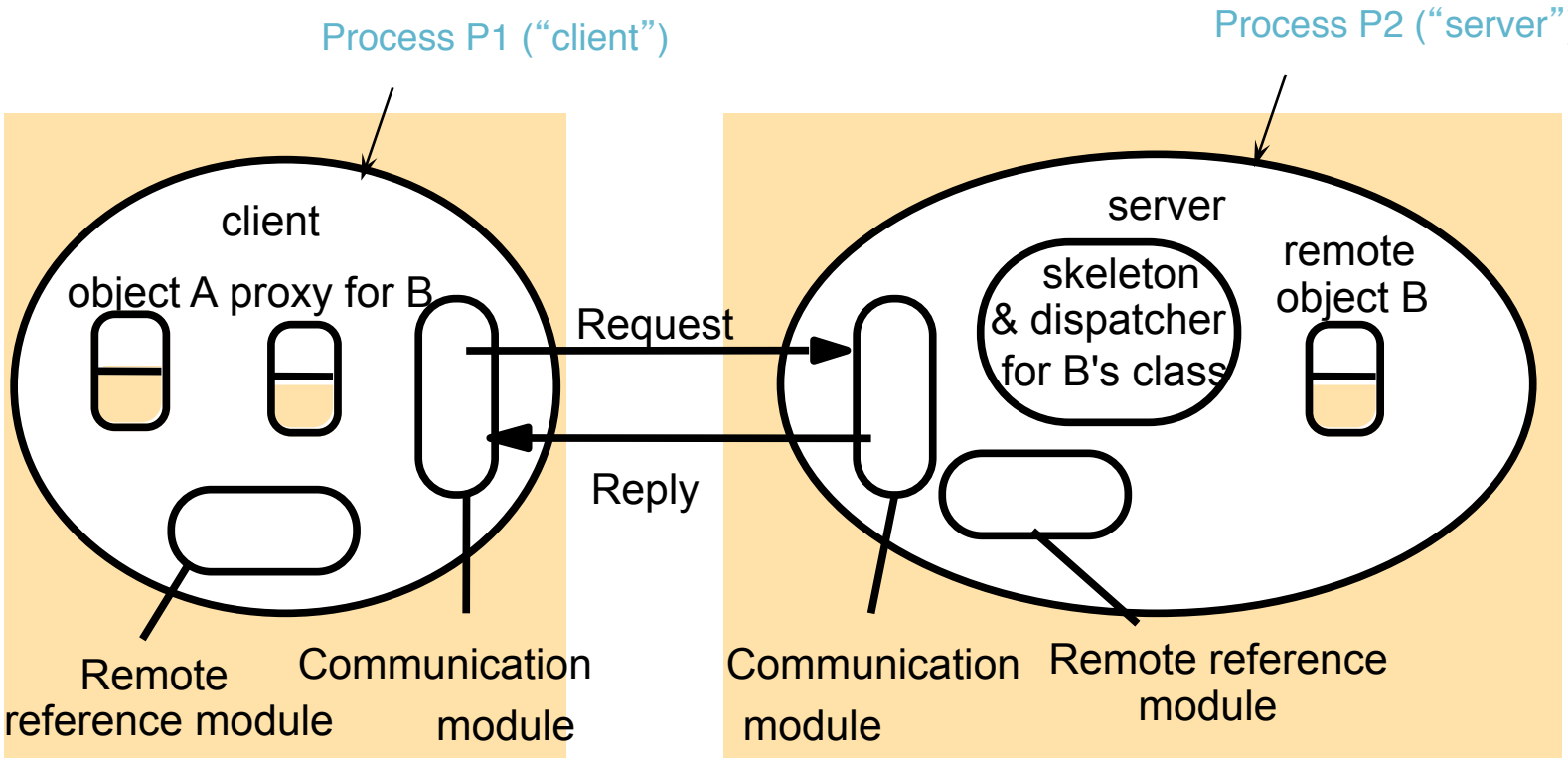
Fault tolera measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
CORBA → No	Not applicable	Not applicable	*Maybe*
Sun RPC → Yes	No	(ok for *idempotent* operations) ↘ Re-execute procedure	*At-least-once*
Java RMI, CORBA → Yes	Yes	Retransmit old reply	*At-most-once*

Idempotent=same result if applied repeatedly, w/o side effects

Proxy and Skeleton in Remote Method Invocation



Proxy and Skeleton in Remote Method Invocation



Proxy

- Is responsible for making RMI transparent to clients by behaving like a local object to the invoker.
  - The proxy implements (Java term, not literally) the methods in the interface of the remote object that it represents. But,…
- Instead of executing an invocation, the proxy forwards it to a remote object
  - Marshals a request message
    - Target object reference
    - Method ID
    - Argument values
  - Sends request message
  - Unmarshals reply and returns to invoker

Marshalling & Unmarshalling

- External data representation: an agreed, platform-independent, standard for the representation of data structures and primitive values.
  - CORBA Common Data Representation (CDR)
  - Sun's XDR
  - Google Protocol Buffers
- Marshalling: the act of taking a collection of data items (platform dependent) and assembling them into the external data representation (platform independent).
- Unmarshalling: the process of disassembling data that is in external data representation form, into a locally interpretable form.

Example: Google Protocol Buffers

```
message Test1 {
  required int32 a = 1;
}
```

```
08 96 01
```

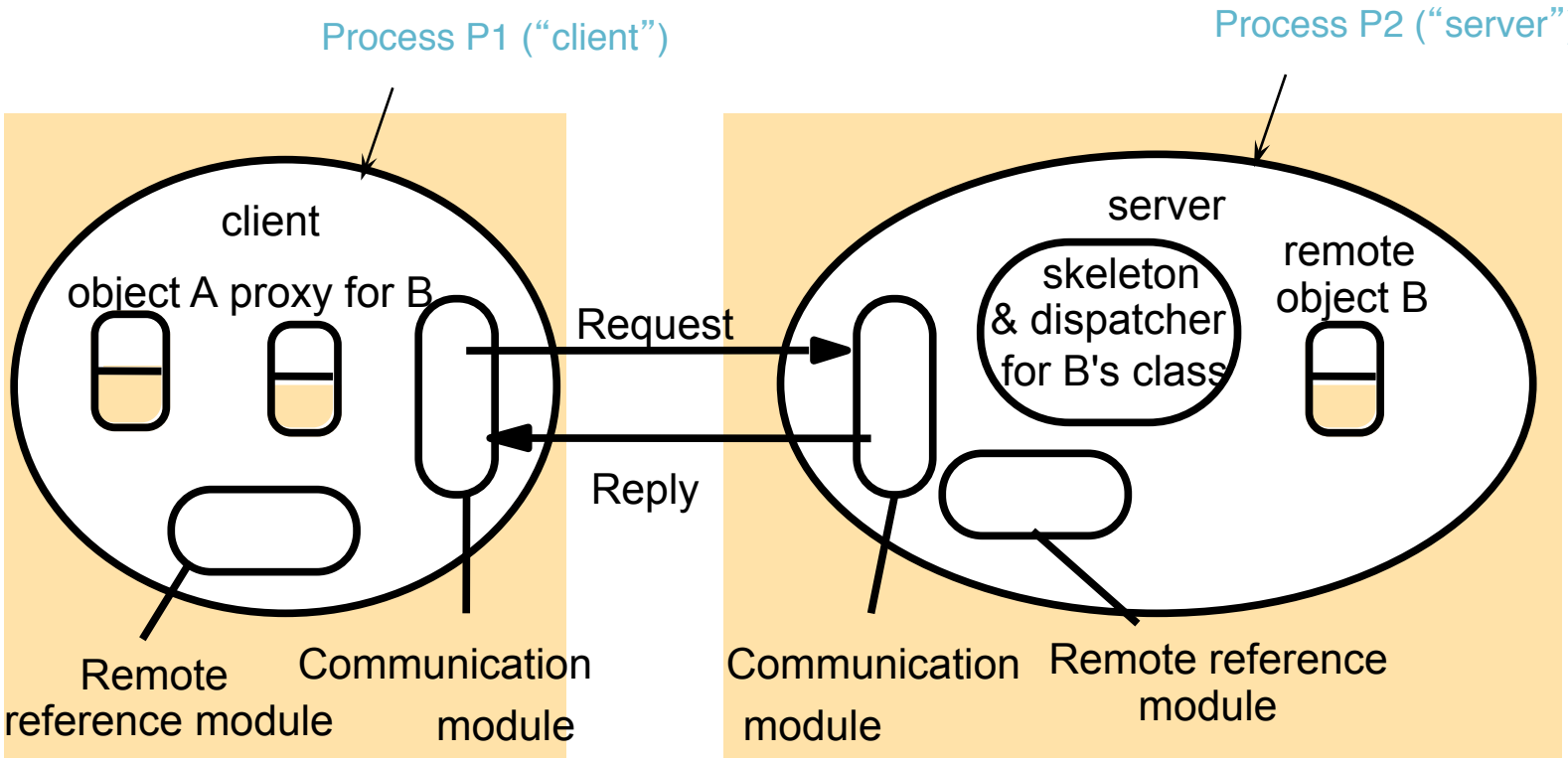
```
message Test2 {
  required string b = 2;
}
```

```
12 07 74 65 73 74 69 6e 67
      t  e  s  t  i  n  g
```

Remote Reference Module

- Is responsible for translating between local and remote object references and for creating remote object references.
- Has a remote object table
  - An entry for each remote object held by any process. E.g., B at P2.
  - An entry for each local proxy. E.g., proxy-B at P1.
- When a new remote object is seen by the remote reference module, it creates a remote object reference and adds it to the table.
- When a remote object reference arrives in a request or reply message, the remote reference module is asked for the corresponding local object reference, which may refer to either a proxy or to a remote object.
- In case the remote object reference is not in the table, the RMI software creates a new proxy and asks the remote reference module to add it to the table.

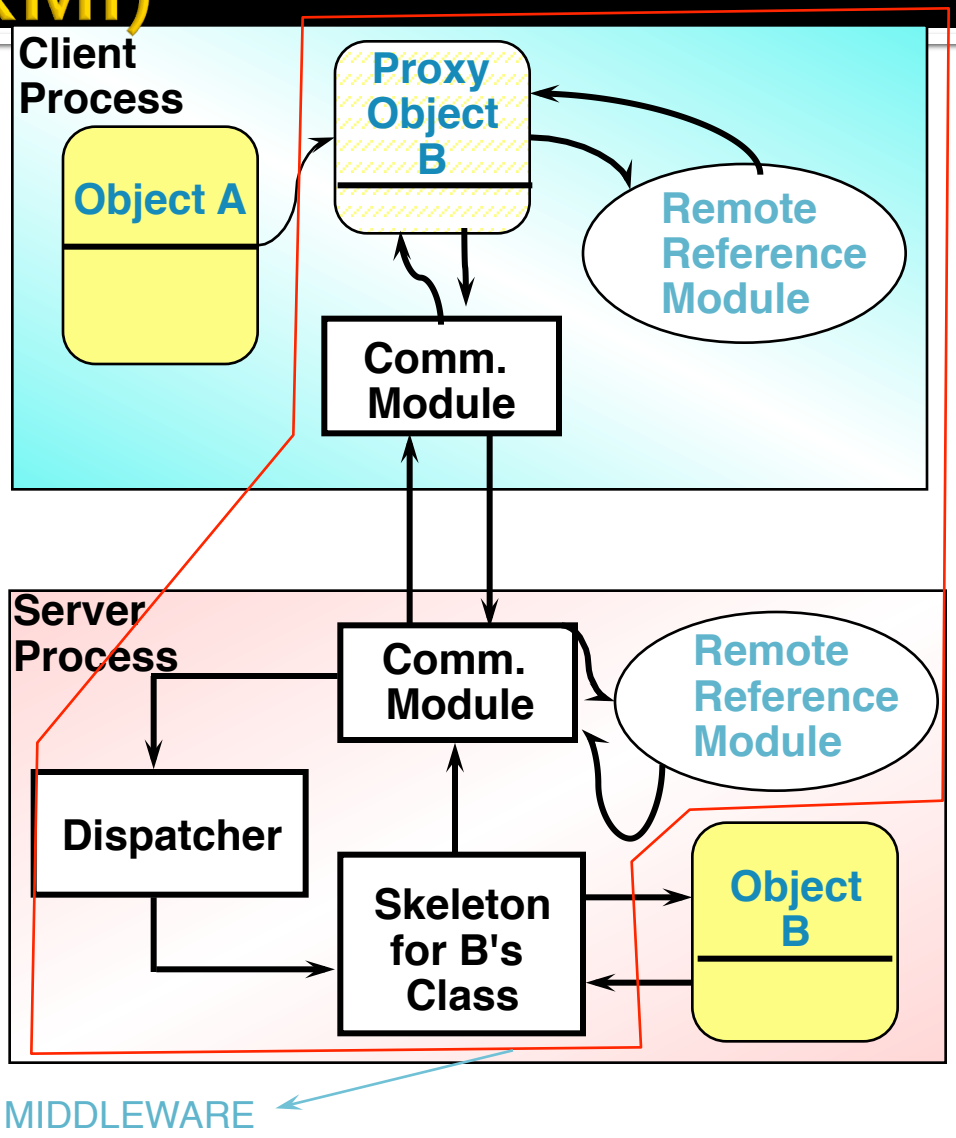
Proxy and Skeleton in Remote Method Invocation



What about Server Side? Dispatcher and Skeleton

- Each process has one dispatcher, and a skeleton for each local object (actually, for the class).
- The dispatcher receives all request messages from the communication module.
  - For the request message, it uses the method id to select the appropriate method in the appropriate skeleton, passing on the request message.
- Skeleton "implements" the methods in the remote interface.
  - A skeleton method un-marshals the arguments in the request message and invokes the corresponding method in the remote object (the actual object).
  - It waits for the invocation to complete and marshals the result, together with any exceptions, into a reply message.

Summary of Remote Method Invocation (RMI)



Proxy object is a hollow container of Method names.

Remote Reference Module translates between local and remote object references.

Dispatcher sends the request to Skeleton Object

Skeleton unmarshals parameters, sends it to the object, & marshals the results for return

Generation of Proxies, Dispatchers and Skeletons

- Programmer only writes object implementations and interfaces
- Proxies Dispatchers and Skeletons generated automatically from the specified interfaces
- In CORBA, programmer specifies interfaces of remote objects in CORBA IDL; then, the interface compiler automatically generates code for proxies, dispatchers and skeletons.
- In Java RMI
  - The programmer defines the set of methods offered by a remote object as a Java interface implemented in the remote object.
  - The Java RMI compiler generates the proxy, dispatcher and skeleton classes from the class of the remote object.

Binder and Activator

- Binder: A separate service that maintains a table containing mappings from textual names to remote object references. (sort of like DNS, but for the specific middleware)
  - Used by servers to register their remote objects by name. Used by clients to look them up. E.g., Java RMI Registry, CORBA Naming Svc.
- Activation of remote objects
  - A remote object is active when it is available for invocation within a running process.
  - A passive object consists of (i) implementation of its methods; and (ii) its state in the marshalled form (a form in which it is shippable).
  - Activation creates a new instance of the class of a passive object and initializes its instance variables. It is called on-demand.
  - An activator is responsible for
    - Registering passive objects at the binder
    - Starting named server processes and activating remote objects in them.
    - Keeping track of the locations of the servers for remote objects it has already activated
  - E.g., Activator=Inetd, Passive Object/service=FTP (invoked on demand)

Etc.

- Persistent Object = an object that survives between simultaneous invocation of a process. E.g., Persistent Java, PerDIS, Khazana.
- If objects migrate, may not be a good idea to have remote object reference=(IP,port,…)
  - Location service= maps a remote object reference to its likely current location
  - Allows the object to migrate from host to host, without changing remote object reference
  - Example: Akamai is a location service for web objects. It "migrates" web objects using the DNS location service

Summary

- Local objects vs. Remote objects
- RPCs and RMIs
- RMI: invocation, proxies, skeletons, dispatchers
- Binder, Activator, Persistent Object, Location Service