CS 425/ECE 428/CSE 424 Distributed Systems (Fall 2009)

Lecture 17

Inter-process Communication and Distributed Objects (I)

Sections 4.3-4.4, Chapter 5

Acknowledgement

- **The slides during this semester are based on ideas and material from the following sources:**
  - **Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.**
  - **Slides from Professor S. Ghosh's course at University of Iowa.**

Administrative

- **MP2 posted October 5, 2009, on the course website,**
  - **Deadline November 6 (Friday)**
  - **Demonstrations , 4-6pm, 11/6/2009**
  - **You will need to lease one Android/Google Developers Phone per person from the CS department (see lease instructions on the web site)!!**
  - **Tutorial for MP2 planned for October 28 evening if students send questions to TA by October 25. Send requests what you would like to hear in the tutorial.**
  - **During October 15-25, Thadpong Pongthawornkamol (tpongth2@illinois.edu) will held office hours and respond to MP2 questions for Ying Huang (Ying is going to the IEEE MASS 2009 conference in China)**

Administrative

- **MP3 proposal instructions**
  - **You will need to submit a proposal for MP3 on top of your MP2 before you start MP3 on November 9, 2009**
  - **Deadline for MP3 proposal: October 25, 2009, email proposal to TA**
  - **At least one representative of each group meets with instructor or TA during October 26-28 during their office hours) watch for extended office hours during these days.**

Plan for Today

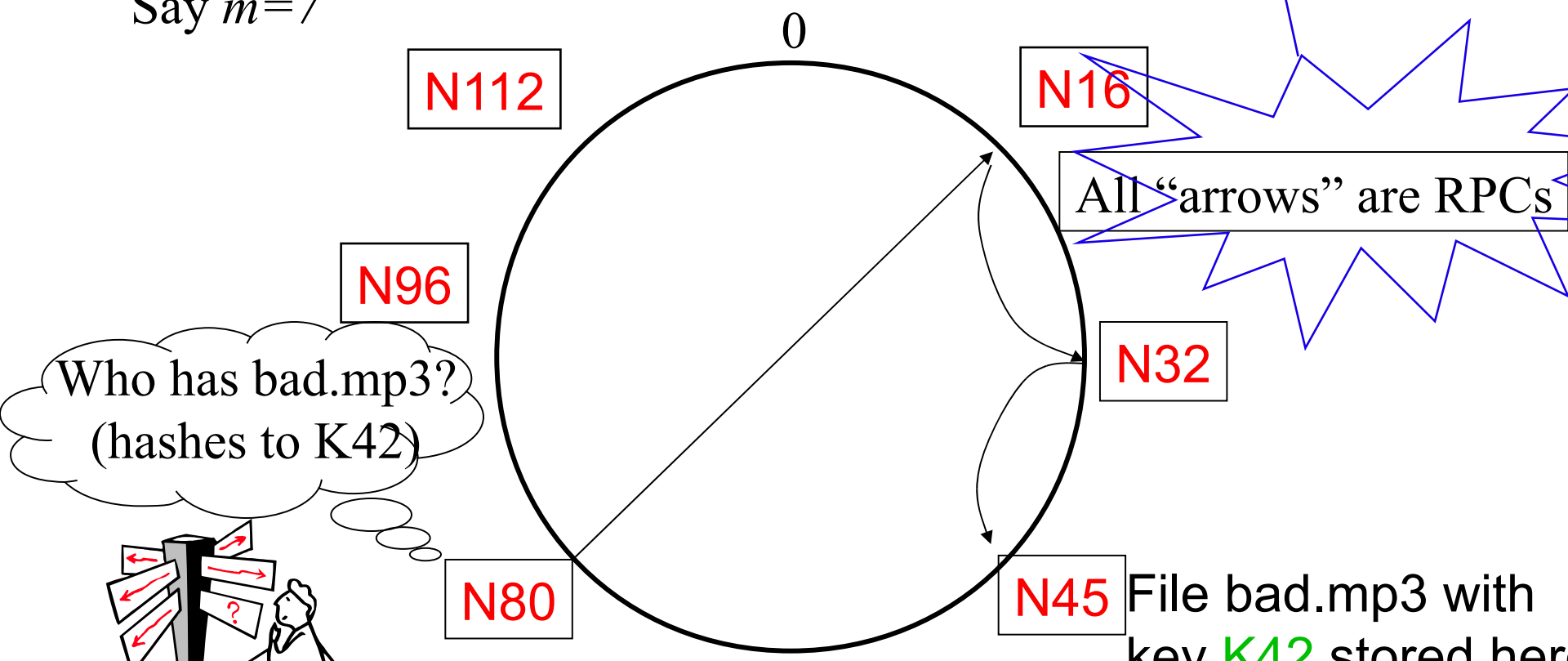
- **Middleware Layer in the protocol stack**
- **Local and remote Objects**
- **Remote method invocation/Remote procedure call paradigms**
- **Failure modes of RMI/RPC**
- **Proxy and skeleton in remote method invocation**
- **Marshaling and de-marshaling**

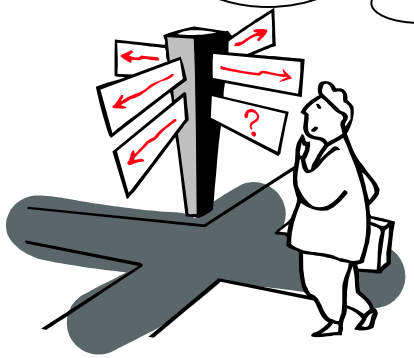
Search in Chord

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



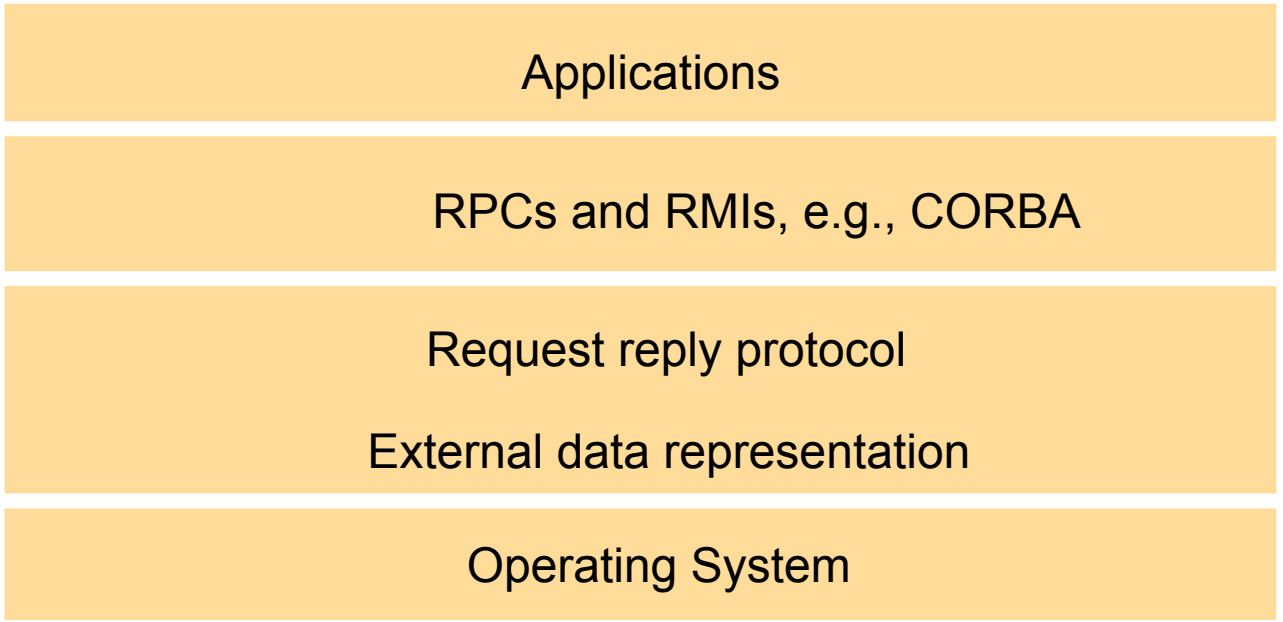


Bank Database Example

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



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

Local Objects

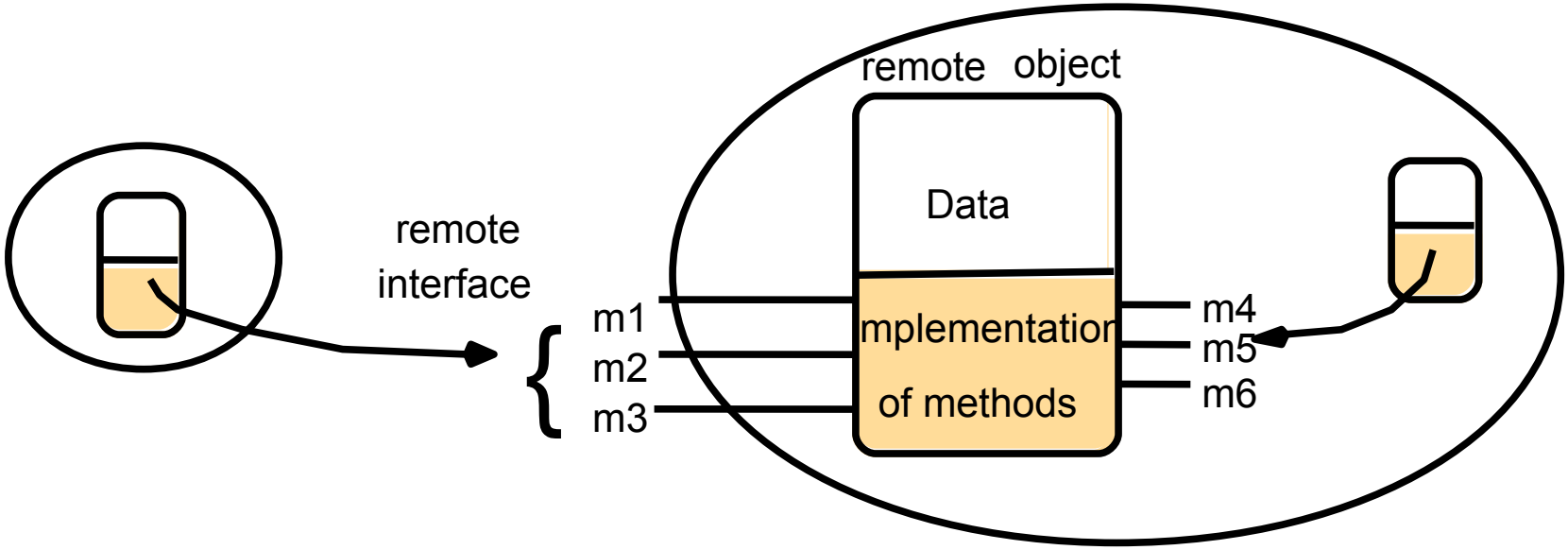
- **Object**
  - consists of a set of data and a set of methods.
  - E.g., C++ object such as the Chord object (Chord data structures + functions at a node).
- **Object reference**
  - an identifier via which objects can be accessed.
  - i.e., a *pointer*
- **Interface**
  - provides a definition of the signatures of a set of methods (i.e., the types of their arguments, return values, and exceptions) without specifying their implementation.
  - E.g., put(objectname), get(objectname) API for Chord object. Same API also applies to other objects such as Gnutella, Kazaa, etc.

Remote Objects

Remote object = object in a different process. This is a relative term. This can receive remote invocations.

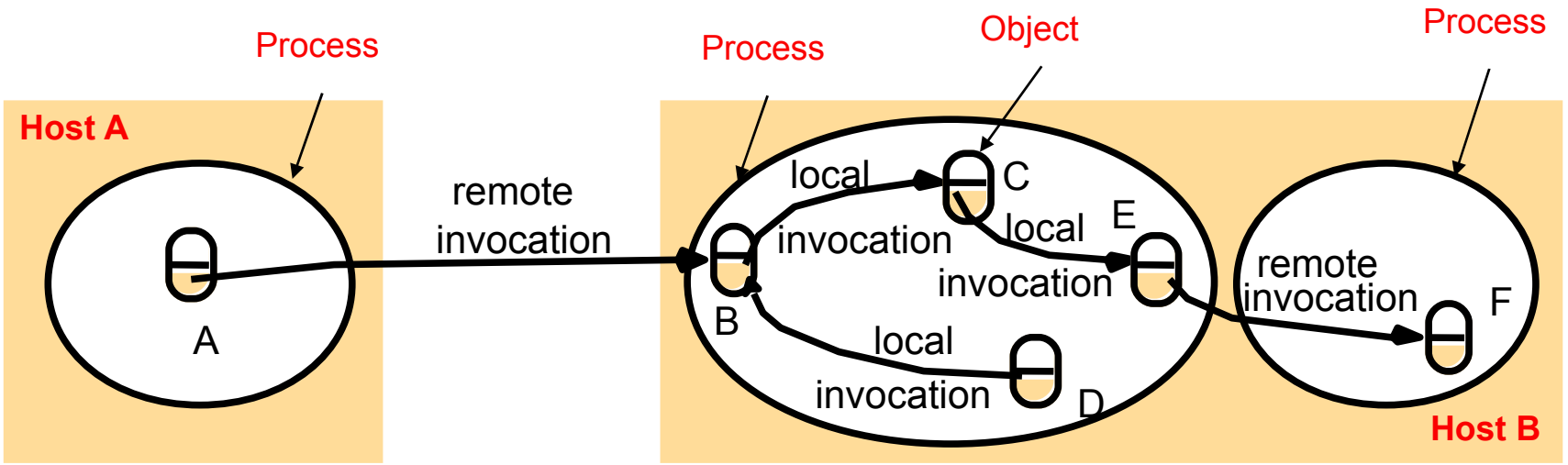
- **Remote method invocation (RMI)**
  - method invocations between objects *in different processes (processes may be on the same or different host)*.
  - Variant of Remote Procedure Call (RPC), which is between different processes (may be on same or different host)
- **Remote object reference**
  - 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)

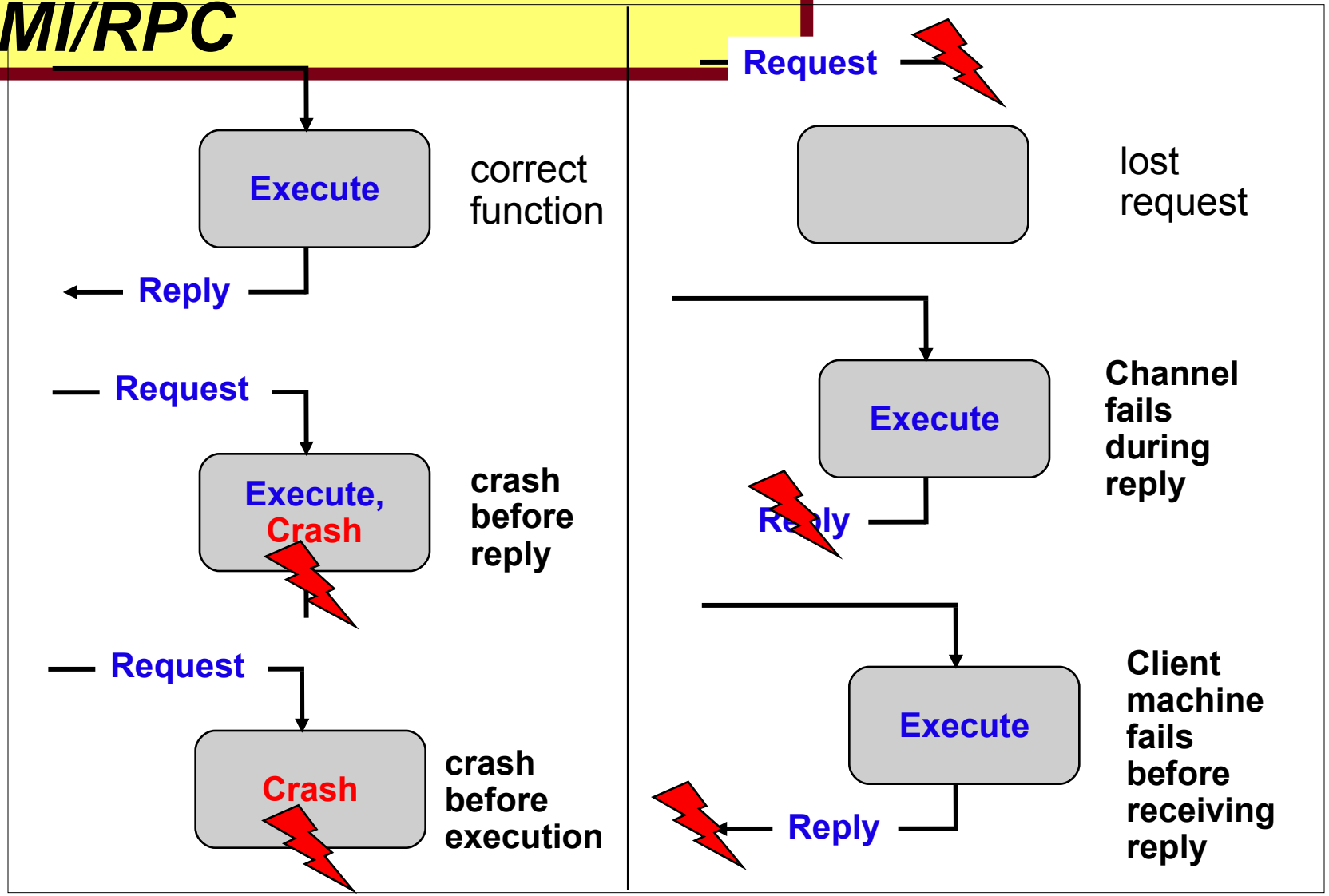
Remote and Local Method Invocations



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?)

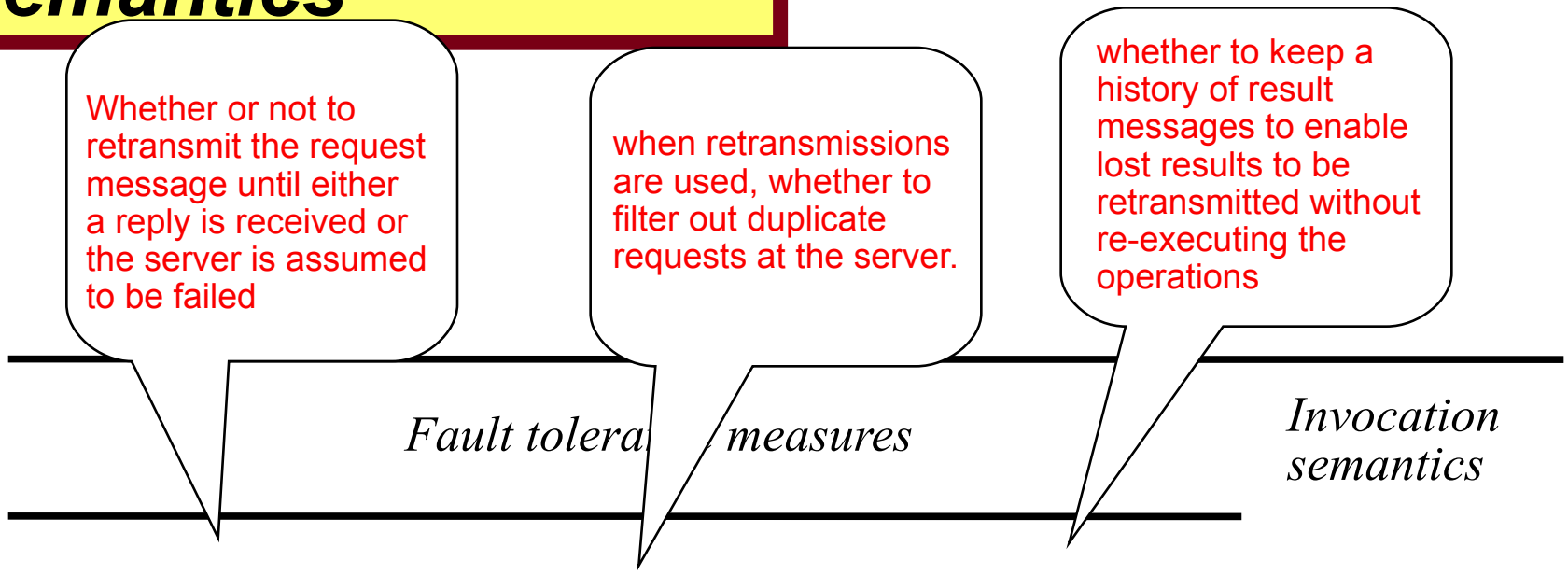
Failure Modes of RMI/RPC

(and if request is received more than once?)



Invocation Semantics

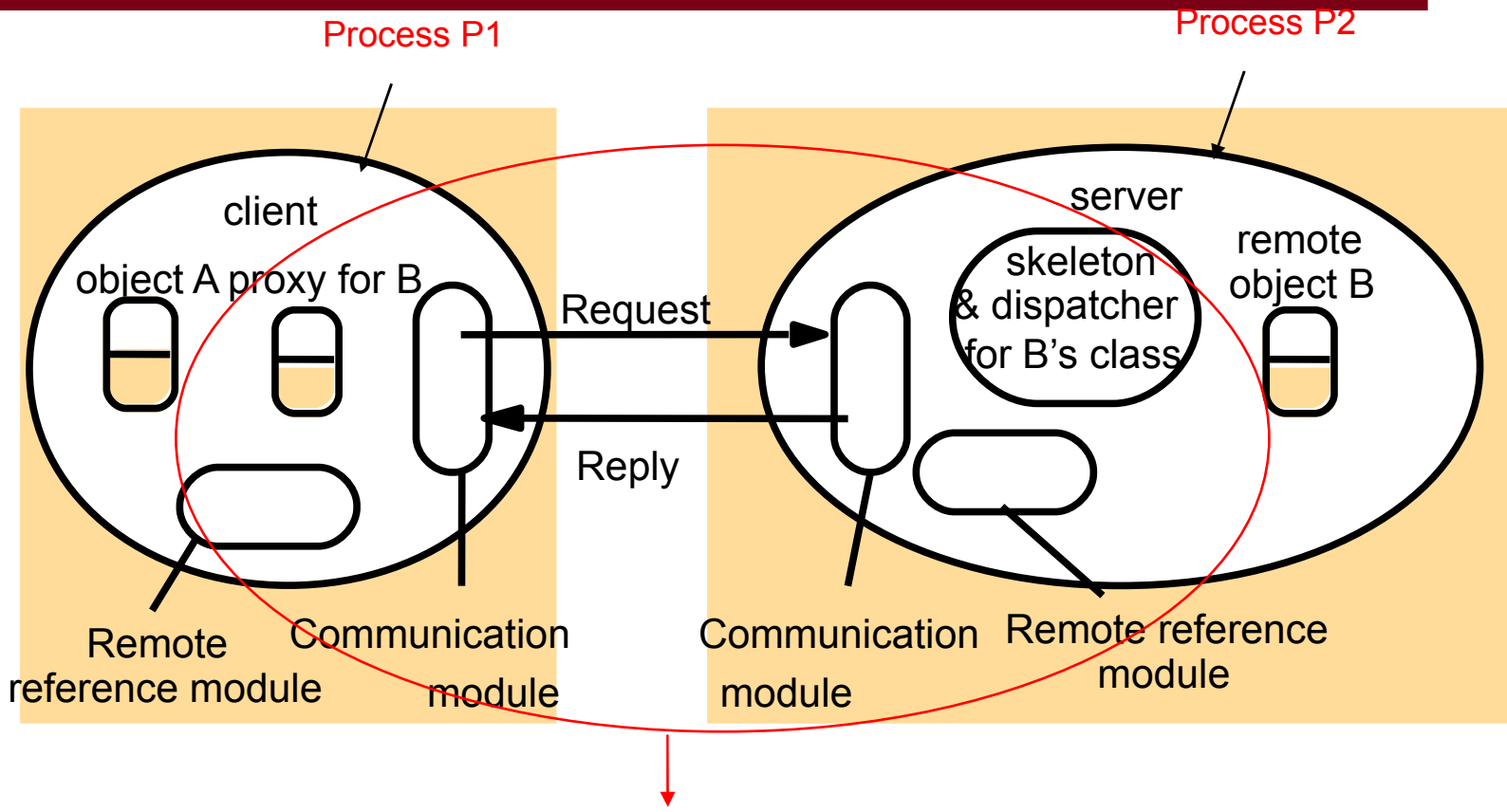
Transparency=remote invocation has same behavior as local invocation
[Birrell and Nelson, inventors of RPC, 1984]
Very difficult to implement in asynchronous network…



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

Idempotent=same result if applied repeatedly

Proxy and Skeleton in Remote Method Invocation



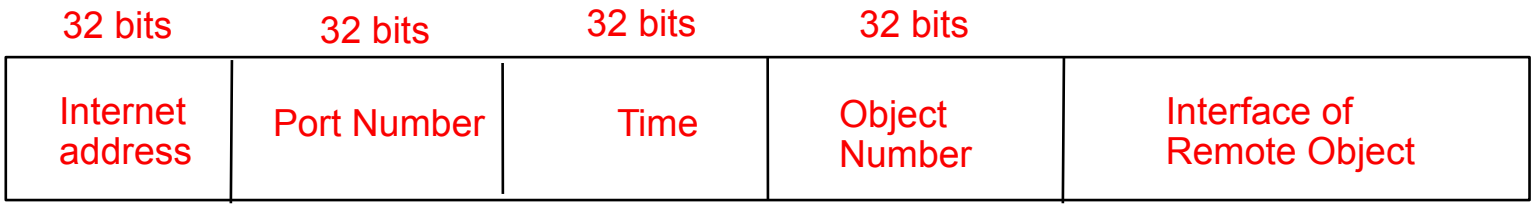
Architecture attempts to ensure transparency *when possible*

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.**

Remote Object References

- **Remote object references must be generated in a manner that ensures uniqueness over space and time**
  - **Even if remote object is deleted, it is important that the remote object reference is not reused**
- **Example of unique remote object reference**
  - **Concatenate Internet address of its computer and the port number of the process that created it with the time of its creation and a local object number**



Proxy

- **Is responsible of 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.**
  - **Each method of the proxy *marshals* the following into a request message: (i) a reference to the target object, (ii) its own method id and (iii) the argument values. Request message is sent to the target, then proxy awaits the reply message, *un-marshals* it and returns the results to the 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)
  - Allows a Windows client (little endian) to interact with a Unix server or Mac server (big endian).

- **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.

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, in a reply message.**

Summary

- Inter-process communication
- Remote method invocation
  - Local and remote objects
  - Distributed objects
- Next lecture – distributed objects and RPC
  - Read Section 5

Backup Slides

Instantiation of remote objects