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

## Lecture 18

## Distributed Objects (II)

**Sections 4.3-4.4, Chapter 5.1-5.4**

**[Indy Gupta]**

# *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*

- **ECE students (and CS) students – if you were refused an Android earlier by Paula Welch, please ask her again (the confusion has been cleared up)**

# *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)**

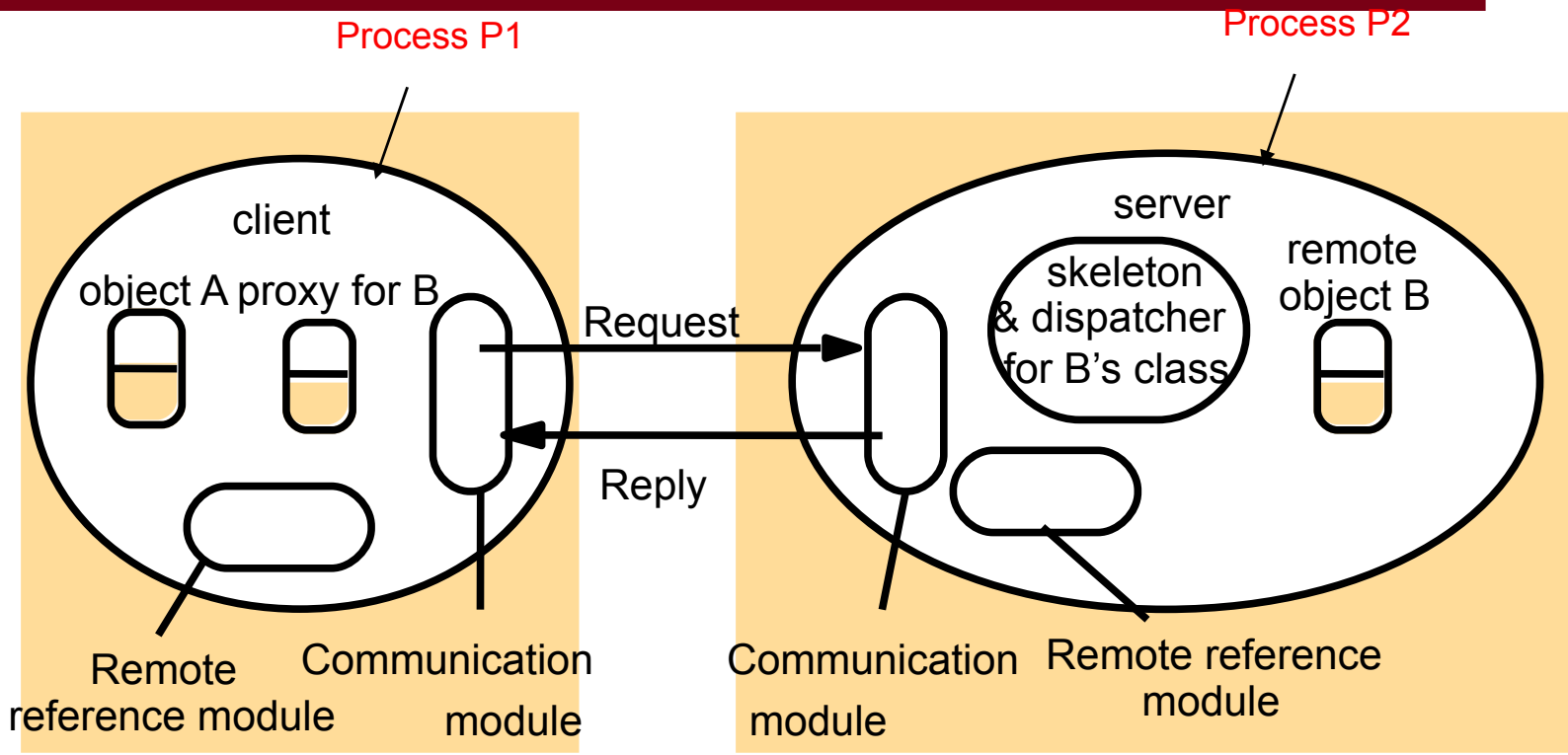
# *Administrative*

- **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*

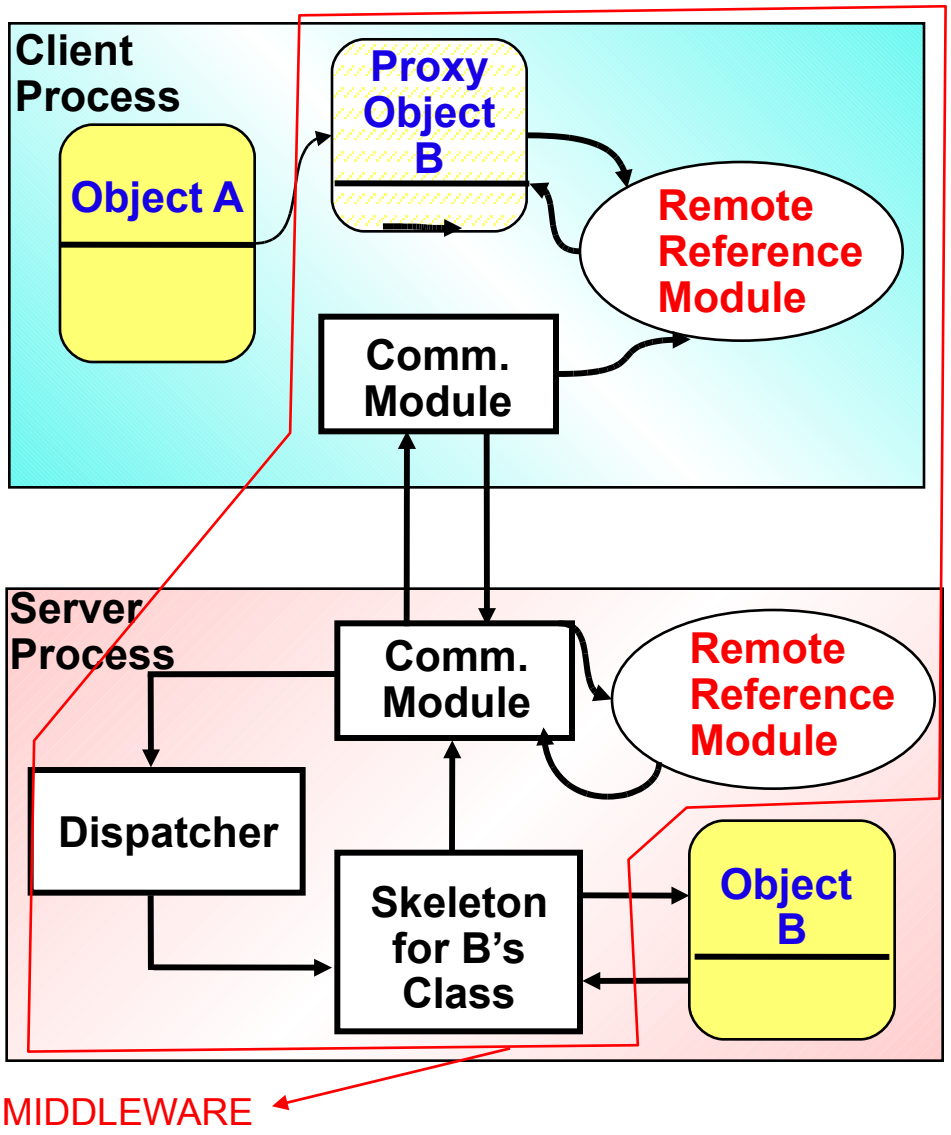
- **Remote Method Invocation (RMI)**
- **Remote procedure call (RPC)**
- **Publish/Subscribe Paradigm**
- **Distributed Event–based Systems**

# Proxy and Skeleton in Remote Method Invocation




Architecture attempts to ensure transparency *when possible*

# Remote Method Invocation (RMI)




Proxy object is a hollow container of Method names.

Remote Reference Module translates between local and remote object references.

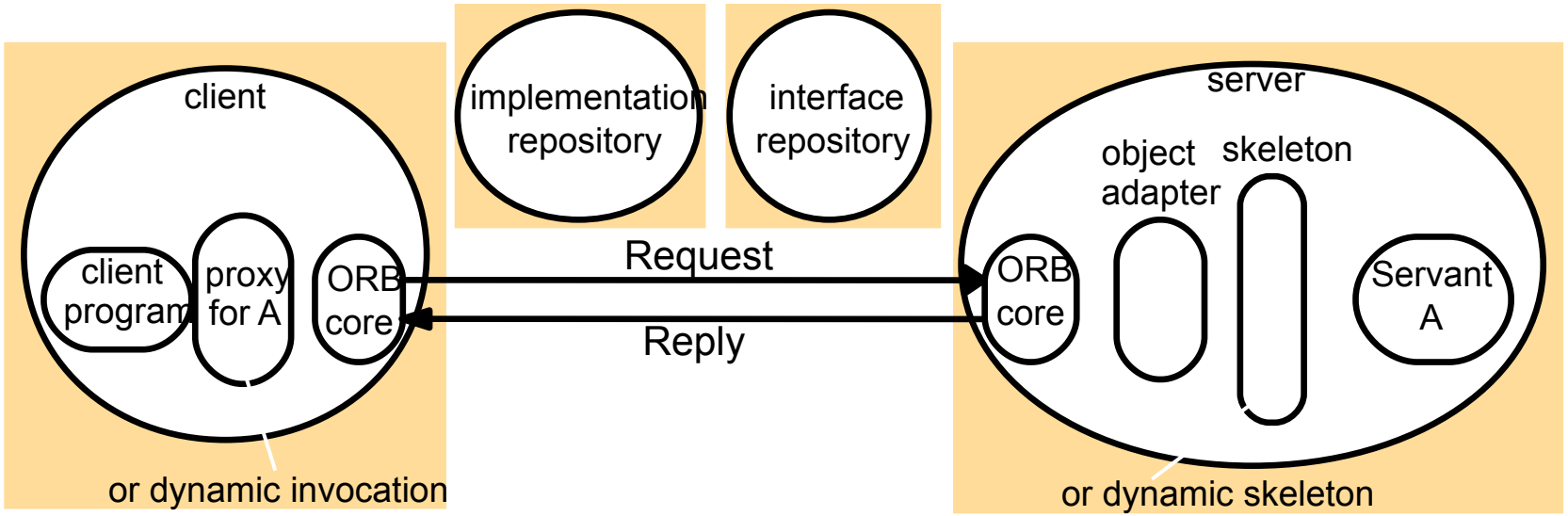
Dispatcher sends the request to Skeleton Object

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

# *Generation of Proxies, Dispatchers and Skeletons*

- **In CORBA, programmer specifies interfaces of remote objects in CORBA IDL**
- **Then, from this IDL, the interface compiler automatically generates code for proxies, dispatchers and skeletons.**
- **For instance, 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.**

# Main components of CORBA architecture



# 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 initiated in an on-demand manner.**
  - **An *activator* is responsible for**
    - » **Registering passive objects, e.g., with the binder (recording the names of the servers against the names of the passive objects)**
    - » **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)**

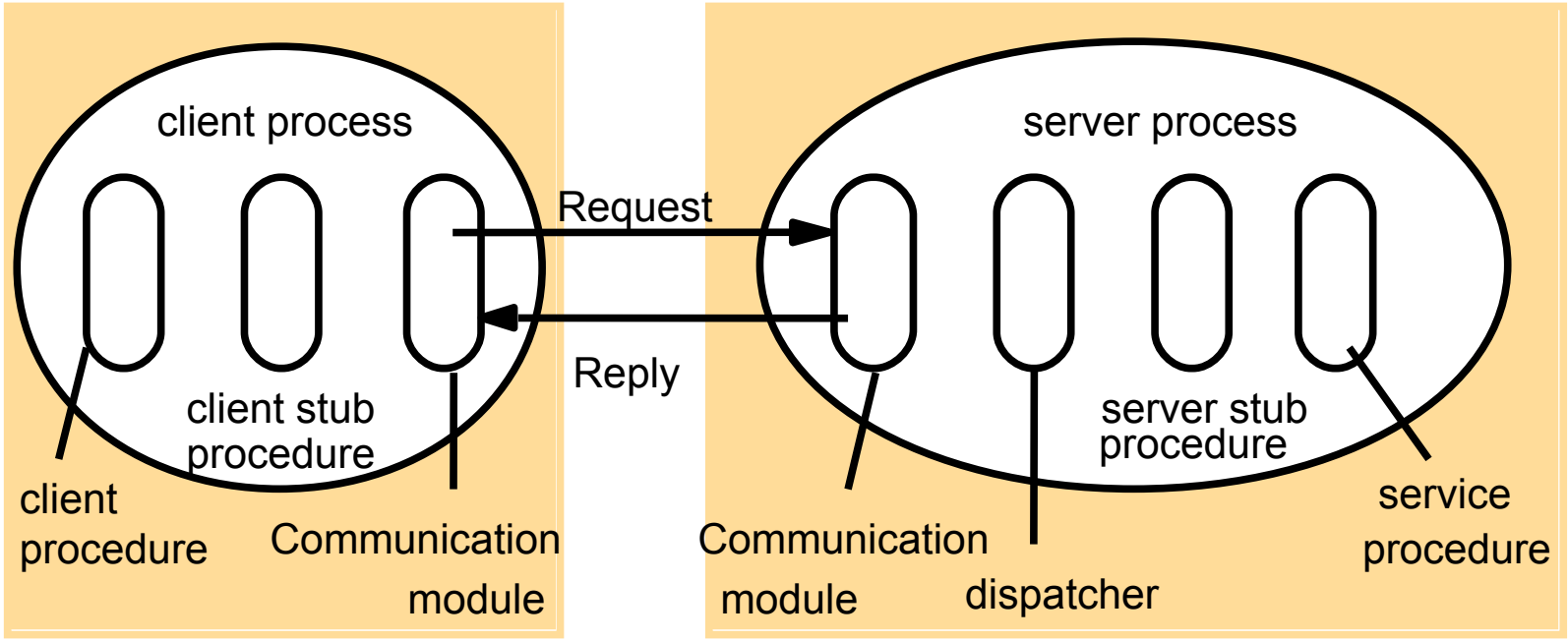
# Etc.

- **Persistent Object = an object that survives between simultaneous invocation of a process. E.g., Persistent Java, PerDIS, Khazana.**
- **If objects are persistent or migrate, may not be a good idea to have remote object reference=IP+port**
  - **Location service= maps a remote object reference to its 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**

# *Remote Procedure Call (RPC)*

- **Uniform, reusable, user-friendly, and action based.**
  - **Provide a familiar interface for the application developer**
  - **Implements the request-reply primitive**
  - **Format of the message is standard**
  - **Supports code reuse**
- **Client process calls for invocation of a procedure at the server process.**
  - **Semantics are similar to RMIs – at least once, at most once, maybe**
  - **Standard interface, independent of applications**
- **A library of reusable procedures, distributed over all sites.**

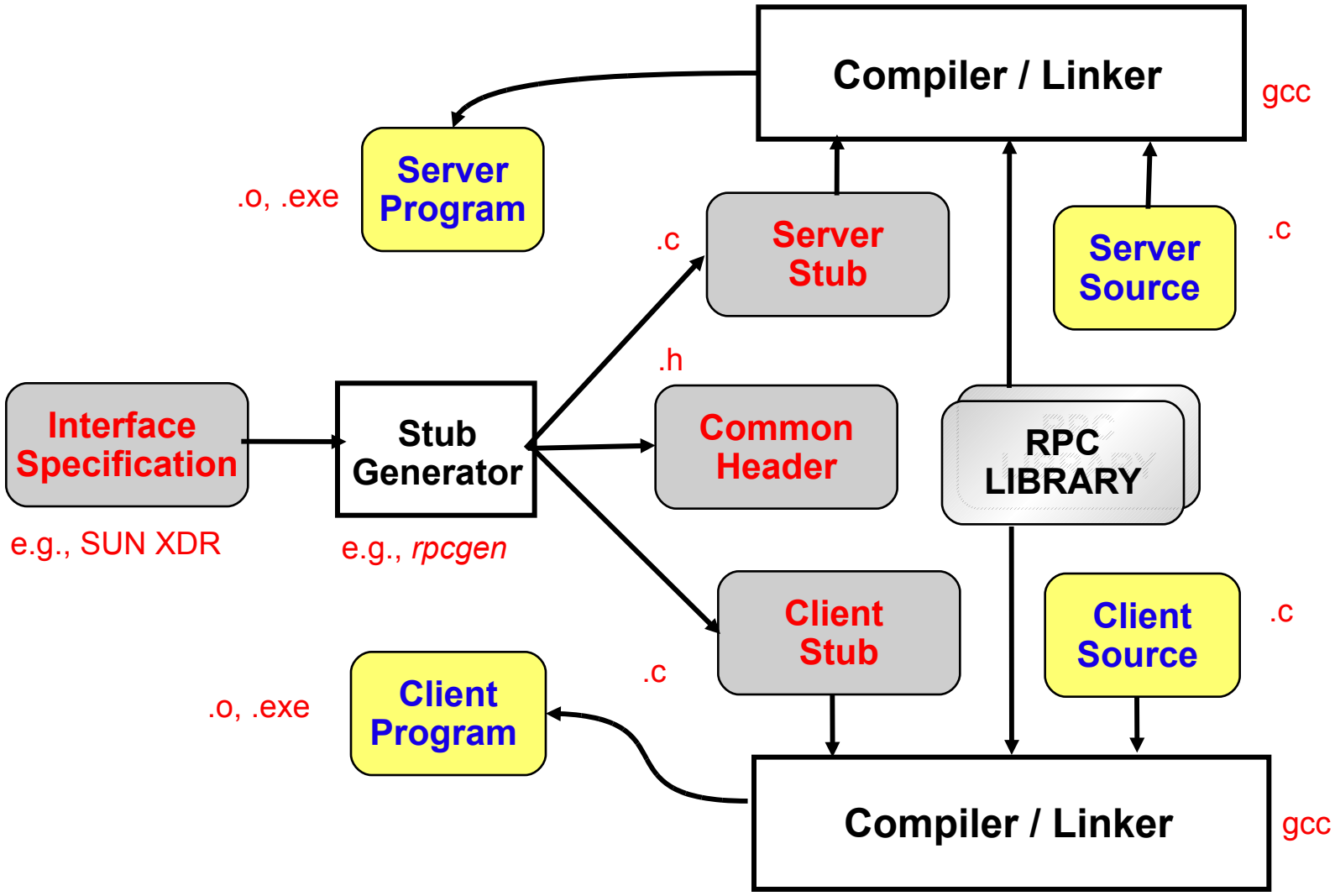
# Client and Server Stub Procedures in RPC



# Stubs

- **Stubs** are generated automatically from interface specifications.
- **Stubs** hide details of (un)marshalling from application programmer & library code developer.
- **Client Stubs** perform marshalling into request messages and unmarshalling from reply messages
- **Server Stubs** perform unmarshalling from request messages and marshalling into reply messages
- **Stubs** also take care of communication & invocation

# The Stub Generation Process



# Files Interface in Sun XDR

Available with most Sun systems, and NFS

```
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;
        Data READ(readargs)=2;
  }=2;
} = 9999;
```

Only one argument allowed
Can specify as struct

Version number

Program number

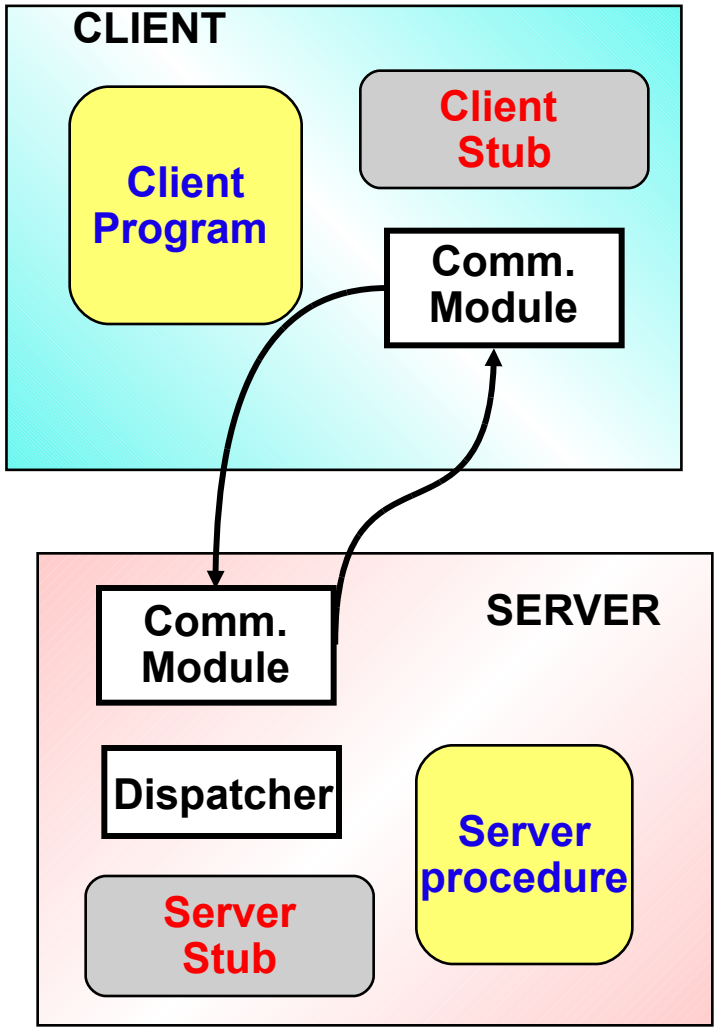
# SUN RPC Client Side

```
/* File : C.c - Simple client of the FileReadWrite service. */
#include <stdio.h>
#include <rpc/rpc.h>
#include "FileReadWrite .h"
main(int argc, char ** argv)
{
        CLIENT *clientHandle;
        char *serverName = "coffee";
        readargs a;
        Data *data;
        clientHandle= clnt_create(serverName, FILEREADWRITE,
                VERSION, "udp"); /* creates socket and a client handle*/
        if (clientHandle==NULL){
                clnt_pcreateerror(serverName); /* unable to contact server */
        exit(1); }
        a.f = 10;   a.position = 100; a.length = 1000;
        data = read_2(&a, clientHandle);/* call to remote read procedure */
                •••
        clnt_destroy(clientHandle); /* closes socket */
}
```

# SUN RPC Server Side

```
/* File S.c - server procedures for the FileReadWrite service */
#include <stdio.h>
#include <rpc/rpc.h>
#include"FileReadWrite.h"
void * write_2(writeargs *a)
{
    /* do the writing to the file */
}
Data * read_2(readargs * a)
{
    static Data result; /* must be static */
    result.buffer = ... /* do the reading from the file */
    result.length = ... /* amount read from the file */
    return &result;
}
```

# Finding RPCs




## Finding An RPC:

RPCs live on specific hosts at specific ports.

Port mapper on the host maps from RPC name to port#

When a server process is initialized, it registers its RPCs (handle) with the port mapper on its host

A client first connects to port mapper (daemon on standard port) to get this handle

The call to RPC is then made by connecting to the corresponding port

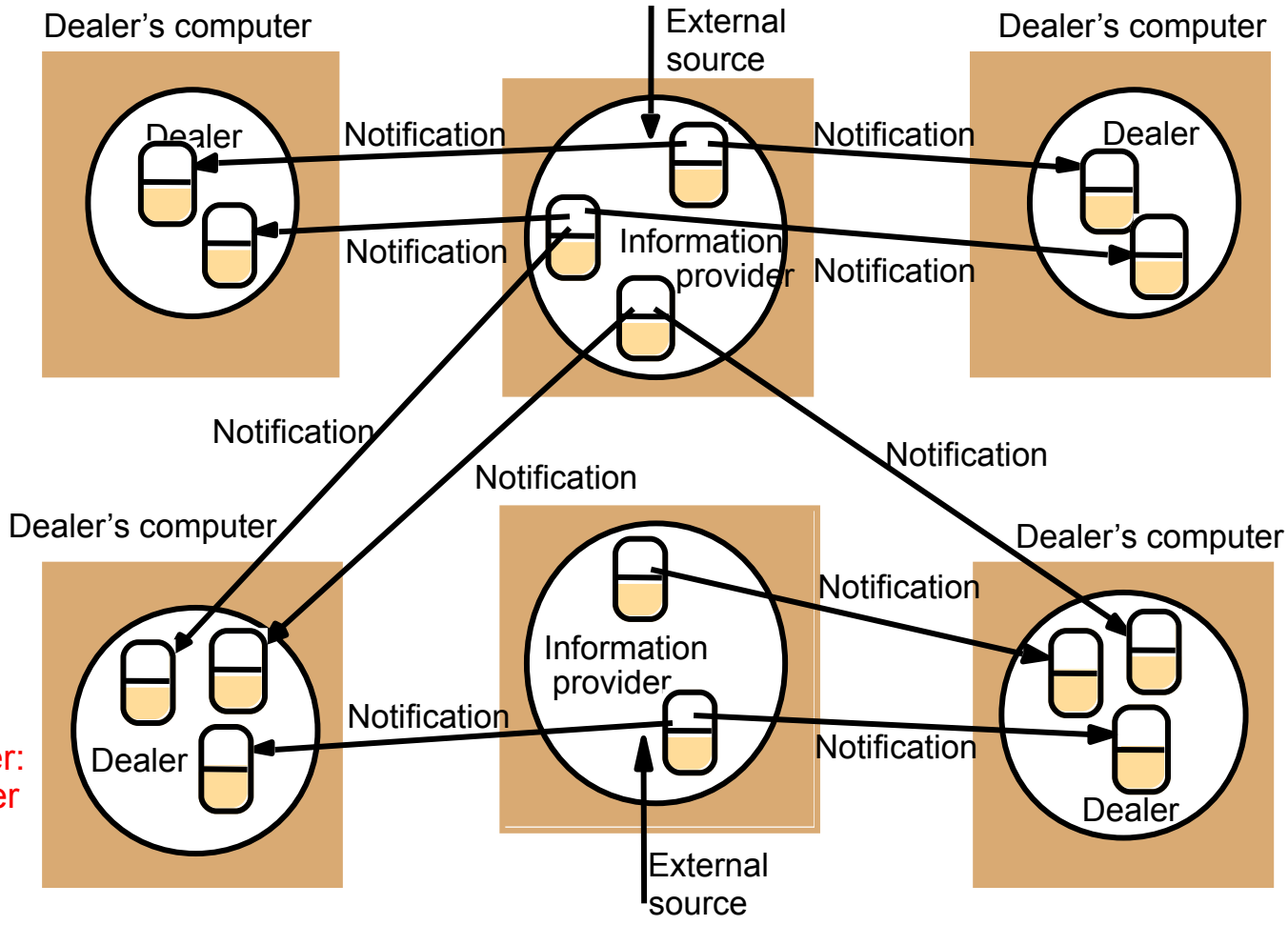
# Publish-Subscribe Paradigm

- **Publisher**
  - Object that declares that it will generate notifications of particular types of events
- **Subscriber**
  - Object that has subscribed to some type of events in another object. It receives notifications about such events.
- **Event**
  - Occurs at an object of interest (may not be same as publisher) as the result of the completion of a method execution
- **Notification**
  - Object that contains information about an event;
  - Typically it contains the type of event and its attributes which generally include identity of the object of interest, method invoked and time of occurrence or sequence number

• Example: Olympics 2008 scores or US Open Tennis scores on your browser. Stock market (next slide).

# Dealing Room System

*[Publish-Subscribe System]*
*e.g, stock market*




At each dealer:
One object per stock-type of interest

# *Distributed Event-based Systems*

- **Based on publish-subscribe paradigm**
- **Two characteristics:**
  - **Heterogeneous:**

Distributed event-based system allows to connect heterogeneous components across the Internet

  - Event-generating objects publish types of events they offer and
  - Other objects subscribe to events and provide an interface for receiving notifications

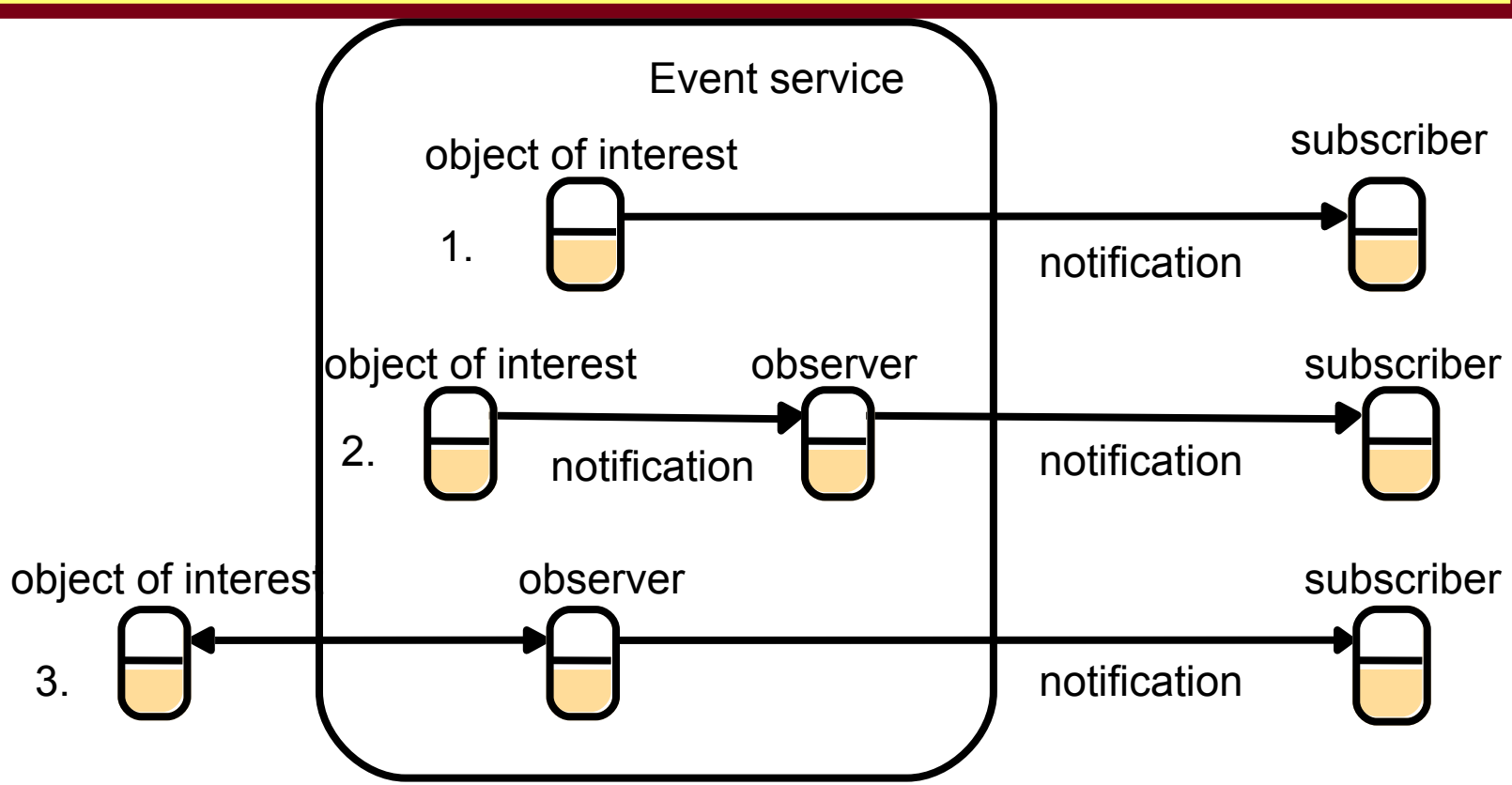
- **Asynchronous:**

Publishers and subscribers are decoupled.

- Notifications are sent *asynchronously* from publisher to subscribers.

- What does “asynchronous” mean? What would “synchronous” mean?

# Architecture for Distributed Event Notification (1)



# Distributed Event Notification (2)

- **Observer (also sometimes called Broker)**
  - Main purpose of an observer is to decouple a publisher/object of interest from its subscribers
  - can have many subscribers
  - = “middleman”
- **Delivery semantics**
  - Variety of delivery guarantees for notifications – application dependent
  - Examples:
    - » Reliable multicast delivery to dealers of stock market
    - » Ordered delivery of notifications
    - » Real-time delivery of notification in hospital patient monitoring
    - » RSS

# Distributed Event Notification (3)

- **Roles of Observers/Brokers**
  - **Forwarding:**
    - » observer sends notifications to subscribers on behalf of one or more publishers
  - **Filtering of notifications:**
    - » filters are applied by observer to reduce the number of notifications received, according to some predicate on the contents of each notification
- **Types of subscriptions/Patterns:** A subscription may be of two main types:
  - **Content-based subscription:** e.g., give me all stocks in price range $100 to $500.
  - **Topic/subject-based subscription:** e.g., give me the latest price of the IBM stock.
  - Can also have more fine-grained types of subscriptions, e.g., notify subscriber only when a certain number of them have generated events.
  - **Notification mailboxes:**
    - » Buffering at observer to wait until subscriber is online

# Summary

- Important Distributed Object Concepts
  - Remote method invocation
  - Remote procedure call
  - Publish/subscribe systems
  - Distributed event systems
- Next lecture – Distributed File Systems
- Reading for next lecture: Distributed File Systems