# EECS 498 – Lecture Notes #1c
# Overview of Distributed Computing Paradigms

**Farnam Jahanian**
**Department of EECS**
**University of Michigan**

**EECS 498 Lecture Notes**

**http://www.eecs.umich.edu/~farnam**

## Reading List

**This Lecture:**

- Overview of Distributed Computing Paradigms

  *Reading List:*

  Handout in class: Chapter 3 – from "Distributed Computing: Principles and Applications" text by M.L. Liu (recommended text)

**Next Lecture:**

- Distributed computation, logical clocks, global state, consistent cuts & runs

  Handout in class: Chapter 4 (section 4.1-4.10) from text by Mullender "Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanism"

  Tanenbaum Text Sections 5.2 and 5.3

## Distributed Programming Paradigms

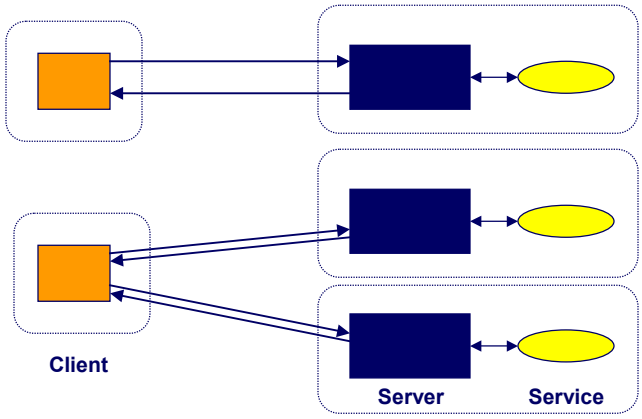
- Message passing model
- Client/server model
- Remote procedure calls
- Distributed objects
- Distributed File Systems
- Distributed transactions
- Process groups and multicast communication
- Publish-subscribe model
- Distributed shared memory
- Peer-to-peer model
- The WWW model and web services

## Paradigms

- **Message Passing Paradigm:**
  - Most basic paradigm: sender, receiver, messages
  - Connection-oriented vs. conectionless communication
  - Socket application programming interface is the most widely-used implementation of this paradigm

- **Client-Server Paradigm: (Tanenbaum 1.5 and 7.3)**
  - Widely-used for networked applications and network services,
  - Simple in concept: structure the system as servers (service providers) and clients (service requestors)
  - Asymmetric roles for two interacting processes
  - Stateless vs. stateful protocols;
    global state information vs. session-state information
  - Many Internet services (FTP, DNS, HTTP) are provided as client-server applications

## The Client-Server Paradigm

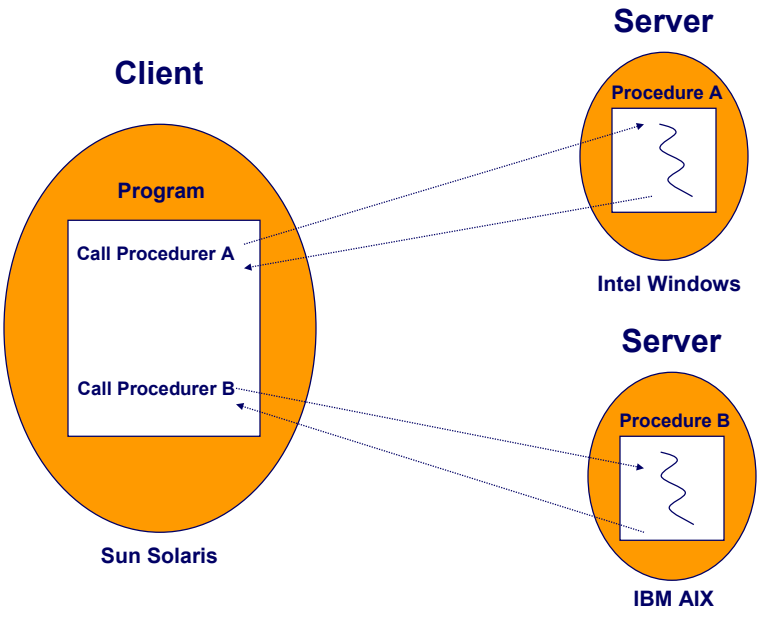



## Remote Procedure Calls

- A paradigm that allows distributed applications to be developed based on the concept of procedural programming --- a convenient abstraction for both communication and synchronization
- Remote operations in the guise of procedural interface
- Perform operations on remote machines asif they were local
- Underlying msg. passing or I/O is hidden from programmer
- Sounds simple but many challenging issues to be addressed (later)???
- Widely used since early 1980s; two APIs prevalent in industry: Open Network Computing RPC evolved from Sun RPC, and Open Group Distributed Computing Environment RPC

See Tanenbaum 2.2

## Logical View of RPC



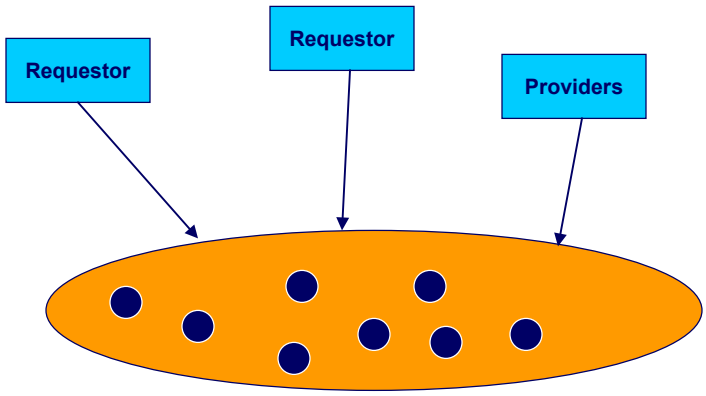

## Distributed Objects Paradigm

- **Natual extension of object-oriented programming**
- **Distributed applications developed (structured) as a collection of interacting objects**
- **Treat everything as objects; services and resources are made available as (remote) objects to clients**
- **Objects provide methods, through the invocation of which an application accesses services**
- **Several implementations of this middleware:**
  - Remote Object Invocation: is essentially an object-oriented implementation of the RPC model (figure 3.8)
  - Corba Object Request Broker (ORB): objects register with ORB which is also called an object bus (figure 3.9); directs requests from requestors to objects that provide services; many Corba implementations including Orbix, Tao and Java IDL
  - Component-based technologies: such MS DCOM, Java Bean; components are specialized packaged objects deigned to interact with others thru standardized interafces

## Object Space

- A variation of distributed object paradigm
- Object space is a virtual space in which objects reside
- Providers put objects in the object space; requestors who subscribe can access objects in the object space
- Object space provide mutual exclusive access to each object
- The abstraction hides the details involved in object lookup and remote vs. local access

## Logical View of Object Space




An Object Space

## Reading List

- **Remote Object Invocation -- Tanenbaum Chapter 2.3**
- **CORBA -- Tanenbaum Chapter 9.1**
- **DCOM -- Tanenbaum 9.2 (optional)**

## Peer-to-Peer Model

- **Unlike client-server model, the participating processes play symmetric roles (with similar capabilities)**
- **Examples of Applications:**
  - Instant messaging, file sharing, virtual rooms, collaborative applications, chat service, video conferencing
- **Overlay networks vs. peer-to-peer applications**
- **Kazaa: is it client server or peer-to-peer?**

## Group Communication Systems (Groupware)

- **Structure a distributed application as a group of cooperating processes (called a process group)**
- **Two basic services:**
  - Group membership service
  - Group communication (multicast) service
- **New groups created and old groups deleted**
- **Dynamic membership: A member can join or leave a group**
- **View collection of processes (members) as a single entity**
- **A process can send a msg. to a group without knowing who all the members are**
- **Various semantics associated with multicast service:**
  - Reliable, FIFO, causal, atomic ordered muticast, … (discussed later)
- **See sections 7.2 and 7.4 in Tanenbaum's text**

## Publish-Subscribe Model

- **Structure a distributed system as a set of publishers and subscribers**
- **Processes subscribe to messages containing information on specific subjects**
- **Other processes produce (or publish) such messages**
- **In most implementations, publishers and subscribers are anonymous**
- **These are also called "subject-based messaging" systems**
- **One can view P/S model as a more abstract variation of group communication model: publisher send multicasts to a group of subscribers, anonymously**

# Distributed Shared Memory

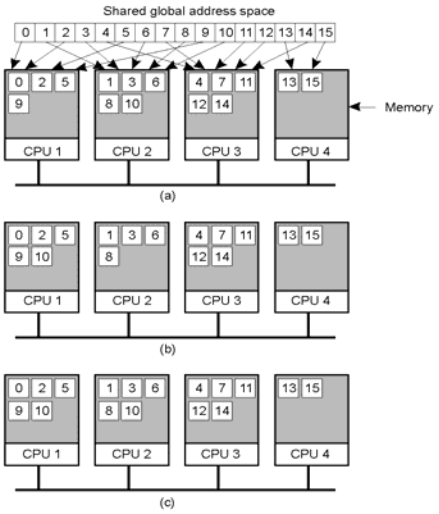
- **Idea: provide the mechanism for a set of networked workstations to share a single, paged virtual address space.**
  - A ref. to local memory is done in hardware → fast
  - Emulate multi-processor caches using MMU and OS: an attempt to ref. an address that is not local causes a page fault, trap to OS, msg. to remote node to fetch the page, and restart faulting instruction
  - Idea is similar to traditional VM systems
- **See Tanenbaum pp. 30-33**
- **Sounds great! What's the problem?**
  - Performance!!!
  - Simple but potential poor performance (similar to thrashing in uni-processors)
- **What to do? Later**

# DSM Example

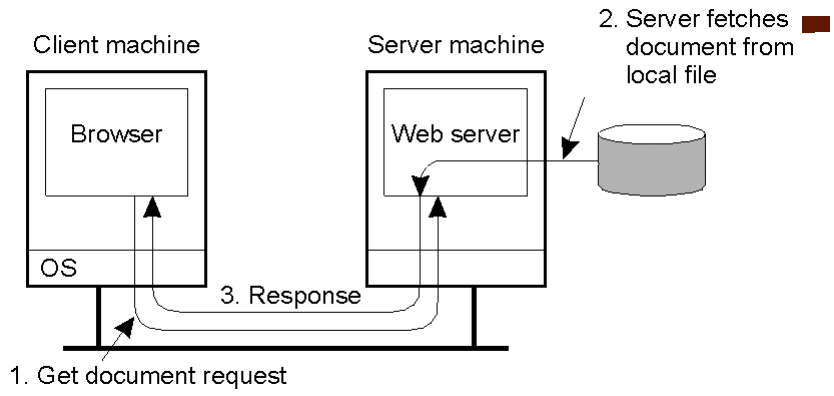
a) **Pages of address space distributed among four machines**

b) **Situation after CPU 1 references page 10**

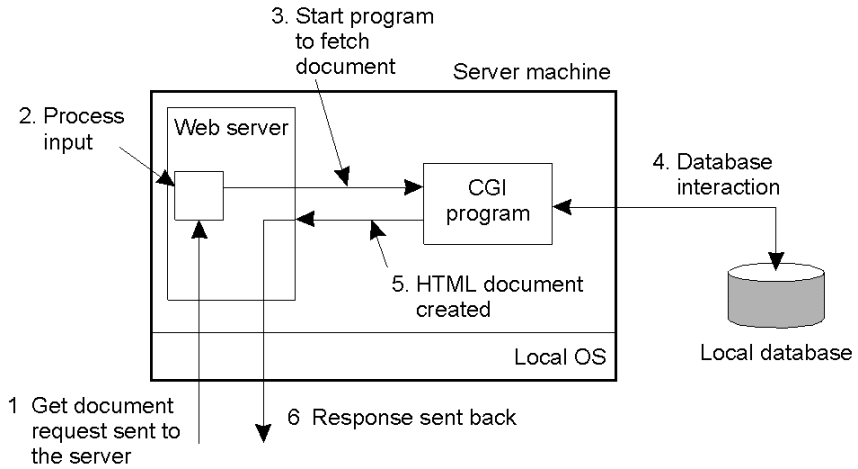
c) **Situation if page 10 is read only and replication is used**



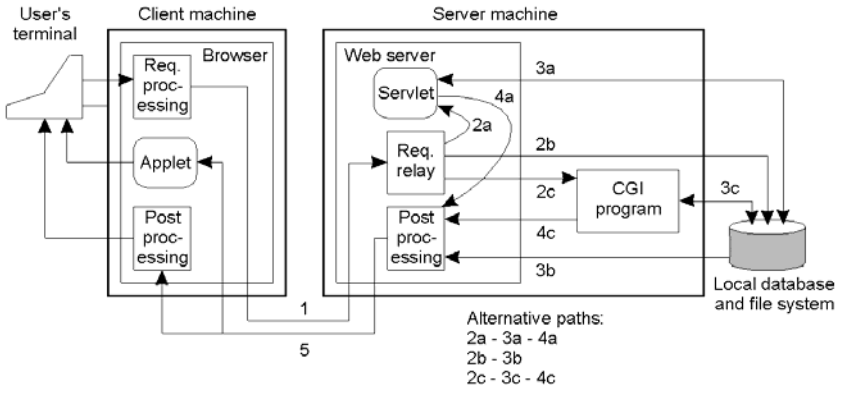
## The World Wide Web




## WWW Architectural Overview (1)



# WWW Architectural Overview (2)




# Web Services

- **The** Web model **has been more rapidly and widely adopted than any other approach to building distributed applications to date. The phenomenal success of the Web model can be attributed to one core characteristic: it is more loosely coupled than traditional distributed programming models like RPC, DCOM and CORBA.**
- **The simplicity of the interactions in the** Web programming model **makes it possible to build systems incrementally. Unlike tightly-coupled RPC and distributed object systems, which require all the pieces of an application be deployed at once, you can add clients and servers to Web-based systems as needed. You can establish connections to new applications fairly easily. And you can do all of this in a decentralized manner, without any central coordination beyond the registration of DNS names, and with a degree of interoperability, scalability and manageability that is remarkably high.**
- **The basic idea behind** Web services **is to adapt the loosely coupled Web programming model for use in applications that are not browser-based. The goal is to provide a platform for building distributed applications using software running on different operating systems and devices, written using different programming languages and tools from multiple vendors, all potentially developed and deployed independently.**

- **Distributed file systems (not covered in 498)**
  - See Tanenbaum Chapter 10
- **Distributed transactions (not covered in 498)**
  - See Tanenbaum Section 5.6