CS 230 - Distributed Systems

Lecture 1 - Introduction to Distributed Systems
Mondays, Wednesdays 12:30-1:50p.m.

Prof. Nalini Venkatasubramanian
nalini@ics.uci.edu

Course logistics and details

- Course Web page
  - http://www.ics.uci.edu/~cs230
- Lectures - MW 12:30-1:50p.m
- Must Read: Course Reading List
  - **Collection of Technical papers and reports by topic**
- Reference Books
  - **Distributed Systems: Concepts & Design**, 4th ed. by Coulouris et al. ISBN: 0-321-26354-5.
  - **Distributed Systems: Principles and Paradigms**, 2nd ed. by Tanenbaum & van Steen. ISBN: 0-132-39227-5.
  - **Distributed Computing: Principles, Algorithms, and Systems**, 1st ed. by Kshemkalyani & Singhal. ISBN: 0-521-87634-6

Prerequisite Knowledge

- Necessary – Operating Systems Concepts and Principles, basic computer system architecture
- Highly Desirable – Understanding of Computer Networks, Network Protocols
- Necessary – Basic programming skills in Java, C++,...

Course logistics and details

- Homeworks
  - Paper summaries
- Midterm Examination
- Course Project
  - Maybe done individually or in groups
    - **Project proposal due end of Week 2**
    - **Survey of related research due end of Week 6**
    - **Final Project presentations/demos/reports – Finals week**
  - Potential projects will be available on webpage

CompSci 230 Grading Policy

- Homeworks - 30% of final grade
  - **1 paper summary due every week from Week 2**
- Midterm - 30% of final grade
  - **Tentatively in Week 7**
- Class Project - 40% of the final grade
- Final assignment of grades will be based on a curve.

Lecture Schedule

- Weeks 1,2,3: Distributed Systems Fundamentals
  - **Introduction – Needs/Paradigms**
    - **Basic Concepts and Terminology, Concurrency**
  - **Time and State in Distributed Systems**
  - **Messaging/Communication in Distributed Systems**
    - **Naming, Directory Services**
- Week 4,5,6: Distributed OS and Middleware Issues
  - **Communication**
    - **Remote Procedure Calls, Remote Method Invocation**
  - **Distributed Process and Resource Management**
    - **Task Migration, Load Balancing, SOA**
  - **Distributed Process Synchronization**
    - **Distributed Mutual Exclusion, Distributed Deadlocks**
  - **Distributed I/O and Storage Subsystems**
    - **Distributed FileSystems**

Lecture Schedule

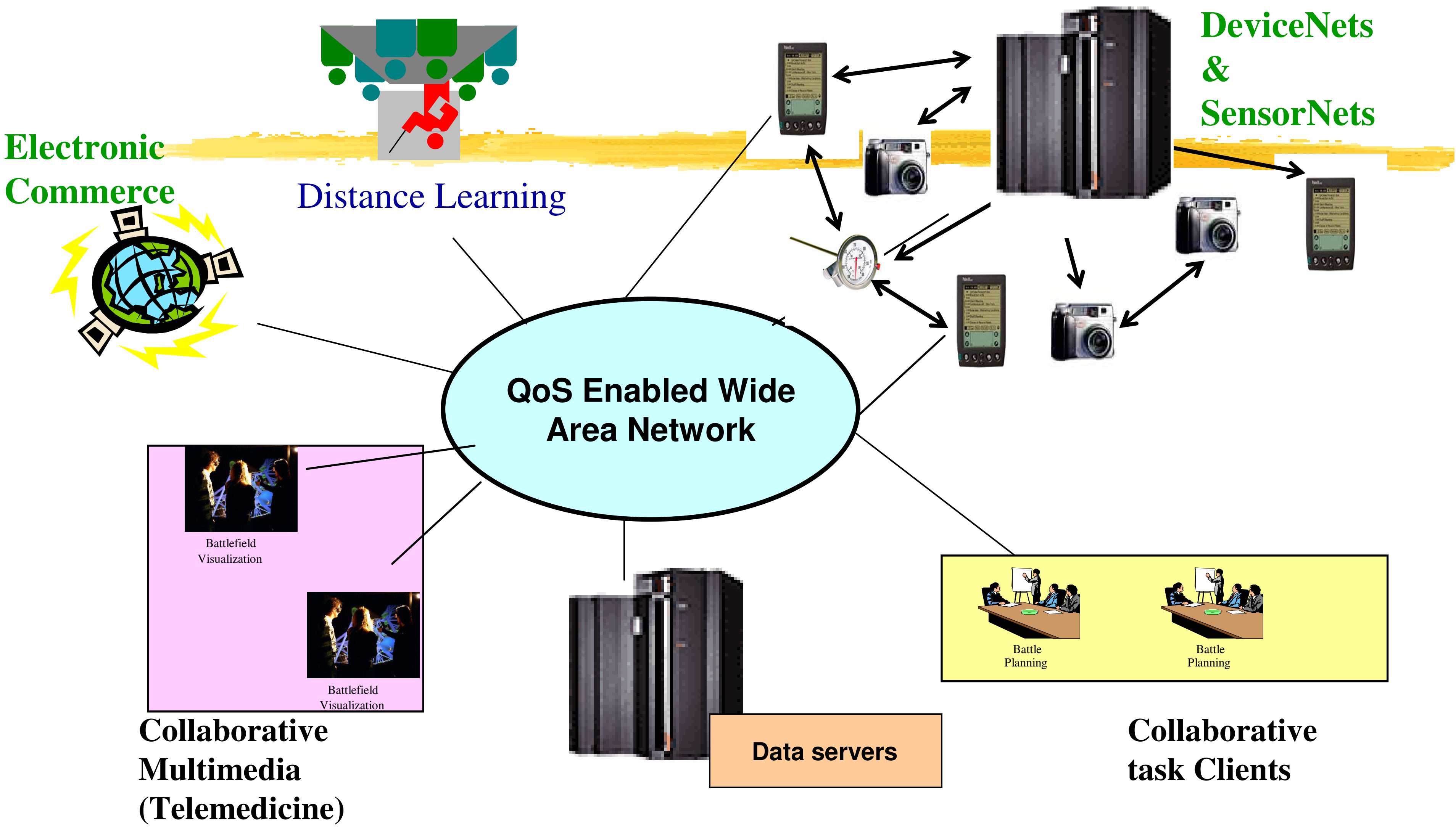
- Weeks 7,8,9: Non-functional "ilities" in distributed systems
  - **Reliability and Fault Tolerance**
  - **Quality of Service and Real-time Needs**
  - **Scalability**
  - **Security and Privacy**

- Week 10: Sample Distributed Systems
  - **P2P, Grid and Cloud Computing**
  - **Mobile and Pervasive Systems/Applications**

Introduction

- Distributed Systems
  - Multiple independent computers that appear as one
  - Lamport's Definition
    - **" You know you have one when the crash of a computer you have never heard of stops you from getting any work done."**
  - "A number of interconnected autonomous computers that provide services to meet the information processing needs of modern enterprises."

Next Generation Information Infrastructure



Requirements - Availability, Reliability, Quality-of-Service, Cost-effectiveness, Security

Characterizing Distributed Systems

- Multiple Autonomous Computers
  - each consisting of CPU's, local memory, stable storage, I/O paths connecting to the environment
  - Geographically Distributed
- Interconnections
  - some I/O paths interconnect computers that talk to each other
- Shared State
  - No shared memory
  - systems cooperate to maintain shared state
  - maintaining global invariants requires correct and coordinated operation of multiple computers.

Examples of Distributed Systems

- Transactional applications - Banking systems
- Manufacturing and process control
- Inventory systems
- General purpose (university, office automation)
- Communication – email, IM, VoIP, social networks
- Distributed information systems
  - WWW
  - Cloud Computing Infrastructures
  - Federated and Distributed Databases

Why Distributed Computing?

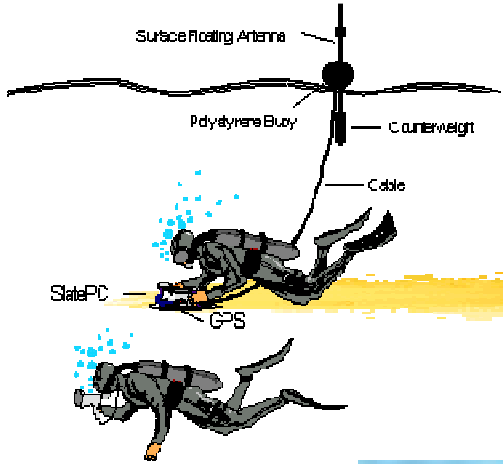
- Inherent distribution
  - Bridge customers, suppliers, and companies at different sites.
- Speedup - improved performance
- Fault tolerance
- Resource Sharing
  - Exploitation of special hardware
- Scalability
- Flexibility

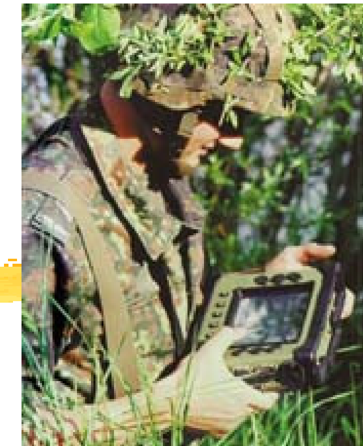
Why are Distributed Systems Hard?

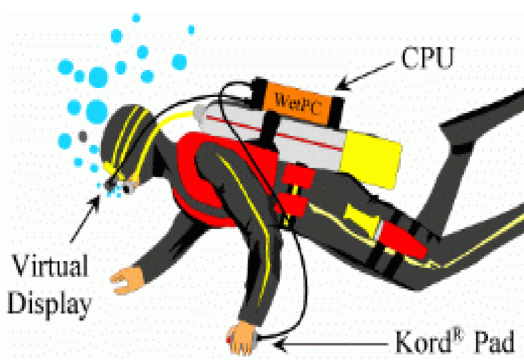
- Scale
  - numeric, geographic, administrative
- Loss of control over parts of the system
- Unreliability of message passing
  - unreliable communication, insecure communication, costly communication
- Failure
  - Parts of the system are down or inaccessible
  - Independent failure is desirable

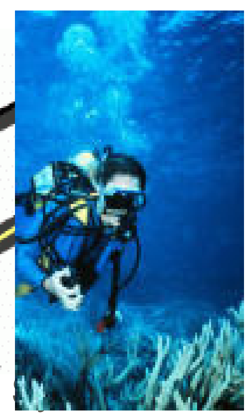
Design goals of a distributed system

- Sharing
  - HW, SW, services, applications
- Openness(extensibility)
  - use of standard interfaces, advertise services, microkernels
- Concurrency
  - compete vs. cooperate
- Scalability
  - avoids centralization
- Fault tolerance/availability
- Transparency
  - location, migration, replication, failure, concurrency

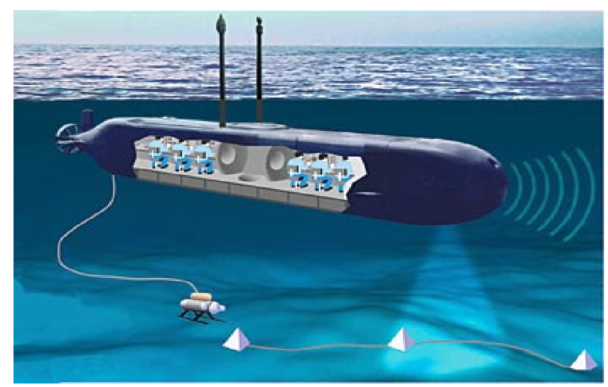




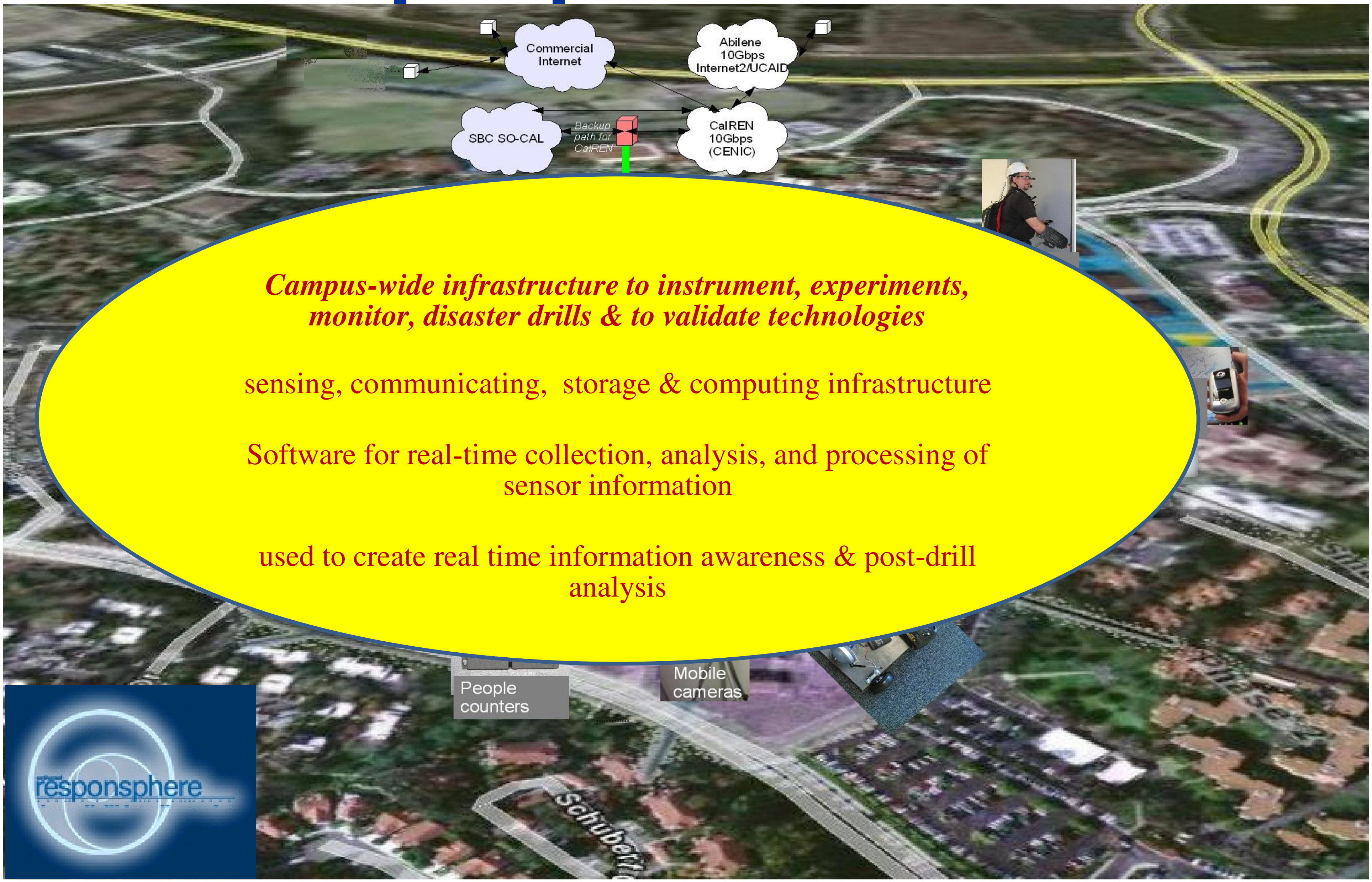




Distributed



Pervasive Sensing and Computing – UCI Responsphere



Campus-wide infrastructure to instrument, experiments, monitor, disaster drills & to validate technologies

sensing, communicating, storage & computing infrastructure

Software for real-time collection, analysis, and processing of sensor information

used to create real time information awareness & post-drill analysis

Classifying Distributed Systems

- Based on degree of synchrony
  - Synchronous
  - Asynchronous
- Based on communication medium
  - Message Passing
  - Shared Memory
- Fault model
  - Crash failures
  - Byzantine failures

Computation in distributed systems

- Asynchronous system
  - no assumptions about process execution speeds and message delivery delays
- Synchronous system
  - make assumptions about relative speeds of processes and delays associated with communication channels
  - constrains implementation of processes and communication
- Models of concurrency
  - Communicating processes
  - Functions, Logical clauses
  - Passive Objects
  - Active objects, Agents

Concurrency issues

- Consider the requirements of transaction based systems
  - Atomicity - either all effects take place or none
  - Consistency - correctness of data
  - Isolated - as if there were one serial database
  - Durable - effects are not lost
- General correctness of distributed computation
  - Safety
  - Liveness

Communication in Distributed Systems

- Provide support for entities to communicate among themselves
  - Centralized (traditional) OS's - local communication support
  - Distributed systems - communication across machine boundaries (WAN, LAN).
- 2 paradigms
  - Message Passing
    - **Processes communicate by sharing messages**
  - Distributed Shared Memory (DSM)
    - **Communication through a virtual shared memory.**

Message Passing

- Basic communication primitives
  - Send message
  - Receive message
- Modes of communication
  - Synchronous
    - atomic action requiring the participation of the sender and receiver.
    - Blocking send: blocks until message is transmitted out of the system send queue
    - Blocking receive: blocks until message arrives in receive queue
  - Asynchronous
    - Non-blocking send:sending process continues after message is sent
    - Blocking or non-blocking receive: Blocking receive implemented by timeout or threads. Non-blocking receive proceeds while waiting for message. Message is queued(BUFFERED) upon arrival.

Reliability issues

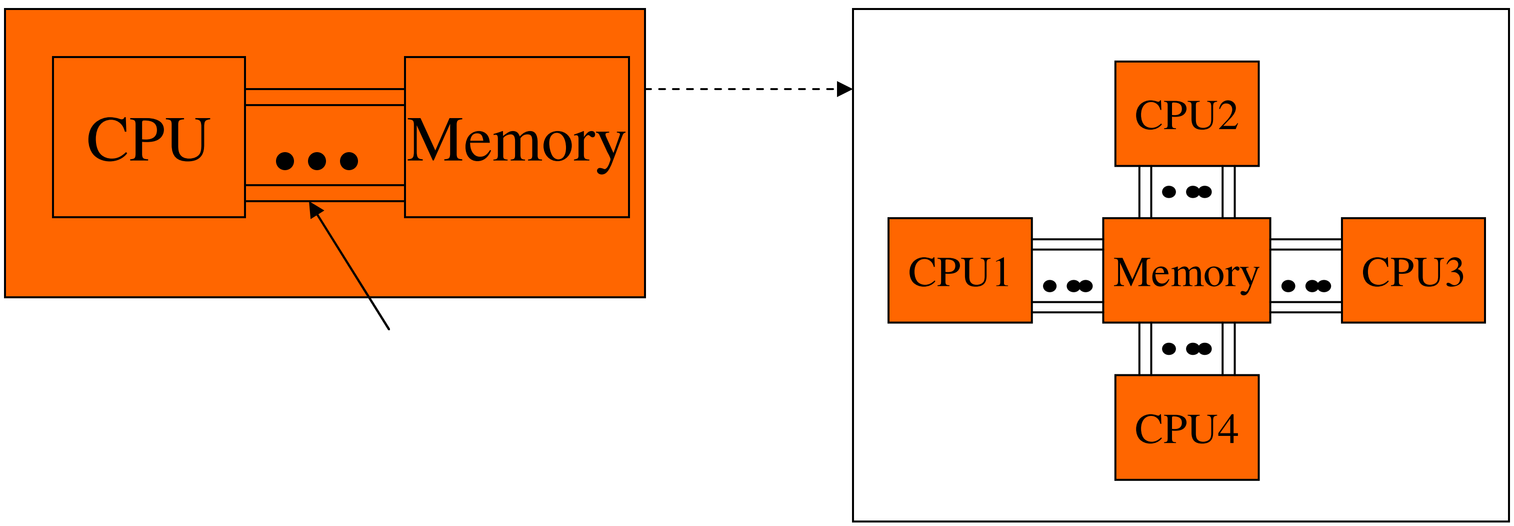
- Unreliable communication
  - Best effort, No ACK's or retransmissions
  - Application programmer designs own reliability mechanism
- Reliable communication
  - Different degrees of reliability
  - Processes have some guarantee that messages will be delivered.
  - Reliability mechanisms - ACKs, NACKs.

Reliability issues

- Unreliable communication
  - Best effort, No ACK's or retransmissions
  - Application programmer designs own reliability mechanism
- Reliable communication
  - Different degrees of reliability
  - Processes have some guarantee that messages will be delivered.
  - Reliability mechanisms - ACKs, NACKs.

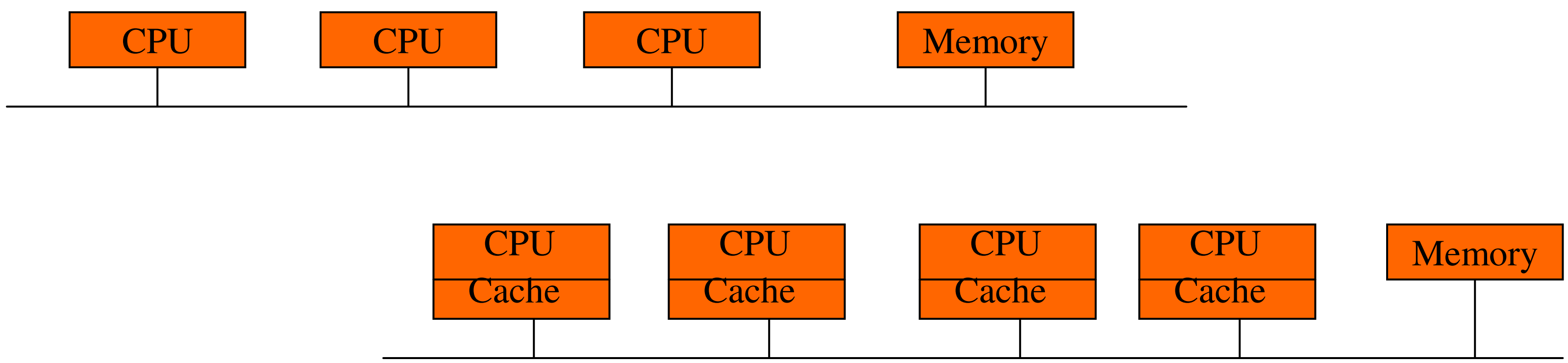
Distributed Shared Memory

- Abstraction used for processes on machines that do not share memory
  - Motivated by shared memory multiprocessors that do share memory



Distributed Shared Memory

- Processes read and write from virtual shared memory.
  - Primitives - read and write
  - OS ensures that all processes see all updates
- Caching on local node for efficiency
  - Issue - cache consistency



Remote Procedure Call

- Builds on message passing
  - extend traditional procedure call to perform transfer of control and data across network
  - Easy to use - fits well with the client/server model.
  - Helps programmer focus on the application instead of the communication protocol.
  - Server is a collection of exported procedures on some shared resource
  - Variety of RPC semantics
    - **"maybe call"**
    - **"at least once call"**
    - **"at most once call"**

Fault Models in Distributed Systems

- Crash failures
  - A processor experiences a crash failure when it ceases to operate at some point without any warning. Failure may not be detectable by other processors.
    - **Failstop - processor fails by halting; detectable by other processors.**
- Byzantine failures
  - completely unconstrained failures
  - conservative, worst-case assumption for behavior of hardware and software
  - covers the possibility of intelligent (human) intrusion.

Other Fault Models in Distributed Systems

- Dealing with message loss
  - Crash + Link
    - **Processor fails by halting. Link fails by losing messages but does not delay, duplicate or corrupt messages.**
  - Receive Omission
    - **processor receives only a subset of messages sent to it.**
  - Send Omission
    - **processor fails by transmitting only a subset of the messages it actually attempts to send.**
  - General Omission
    - **Receive and/or send omission**

Other Distributed System issues

- Concurrency and Synchronization
- Distributed Deadlocks
- Time in distributed systems
- Naming
- Replication
  - improve availability and performance
- Migration
  - of processes and data
- Security
  - eavesdropping, masquerading, message tampering, replaying

Client/Server Computing

- Client/server computing allocates application processing between the client and server processes.
- A typical application has three basic components:
  - Presentation logic
  - Application logic
  - Data management logic

Client/Server Models

- There are at least three different models for distributing these functions:
  - Presentation logic module running on the client system and the other two modules running on one or more servers.
  - Presentation logic and application logic modules running on the client system and the data management logic module running on one or more servers.
  - Presentation logic and a part of application logic module running on the client system and the other part(s) of the application logic module and data management module running on one or more servers

Management and Support

Network Management

Enterprise Systems:
Perform enterprise activities

Application Systems:
support enterprise systems

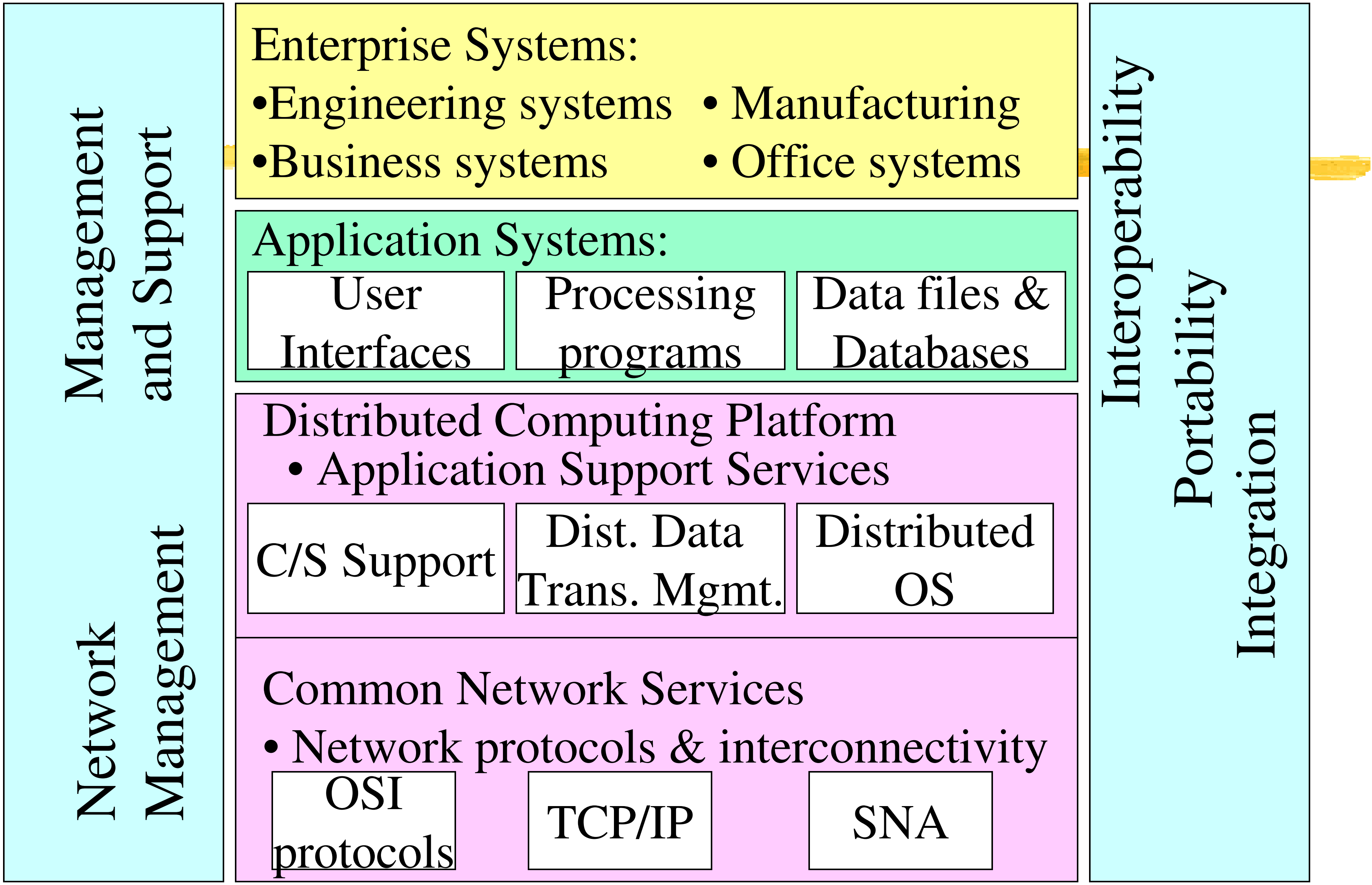
Distributed Computing Platform

- Application Support Services (OS, DB support, Directories, RPC)
- Communication Network Services (Network protocols, Physical devices)
- Hardware

Interoperability

Portability

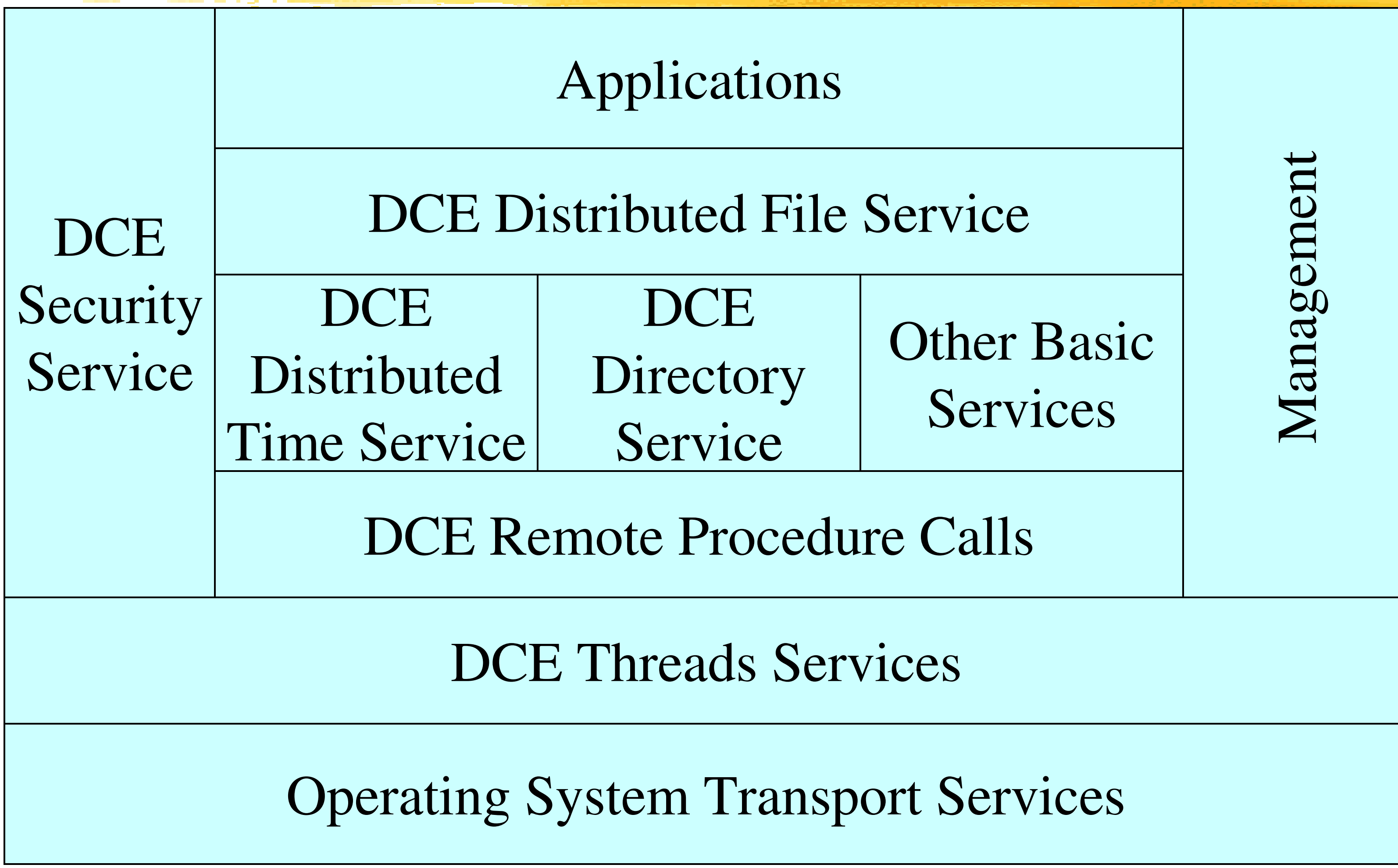
Integration


Management and Support
Network Management
Enterprise Systems:
•Engineering systems
• Manufacturing
•Business systems
• Office systems
Application Systems:
User Interfaces
Processing programs
Data files & Databases
Distributed Computing Platform
• Application Support Services
C/S Support
Dist. Data Trans. Mgmt.
Distributed OS
Common Network Services
• Network protocols & interconnectivity
OSI protocols
TCP/IP
SNA
Interoperability
Portability
Integration

Distributed Computing Environment (DCE)

- DCE is from the Open Software Foundation (OSF), and now X/Open, offers an environment that spans multiple architectures, protocols, and operating systems.
  - DCE supported by major software vendors.
- It provides key distributed technologies, including RPC, a distributed naming service, time synchronization service, a distributed file system, a network security service, and a threads package.

DCE



Applications
DCE Distributed File Service
DCE Security Service
DCE Distributed Time Service
DCE Directory Service
Other Basic Services
Management
DCE Remote Procedure Calls
DCE Threads Services
Operating System Transport Services

Distributed Systems Middleware

- Middleware is the software between the application programs and the operating System and base networking
- Integration Fabric that knits together applications, devices, systems software, data
- Middleware provides a comprehensive set of higher-level distributed computing capabilities and a set of interfaces to access the capabilities of the system.

The Evergrowing Middleware Alphabet Soup

Distributed Computing Environment (DCE)

WS-BPEL

WSIL

Java Transaction API (JTA)

Orbix

WSDL

JNDI

LDAP

JMS

IOP

BPEL

BEA Tuxedo®

IIOP

EAI

RTCORBA

GIOP

Object Request Broker (ORB)

SOAP

Message Queuing (MSMQ)

XQuery

Distributed Component Object Model (DCOM)

opalORB

XPath

IDL

Remote Method Invocation (RMI)

ZEN

Rendezvous

ORBlite

Encina/9000

BEA WebLogic®

Remote Procedure Call (RPC)

Extensible Markup Language (XML)

Distributed Sys

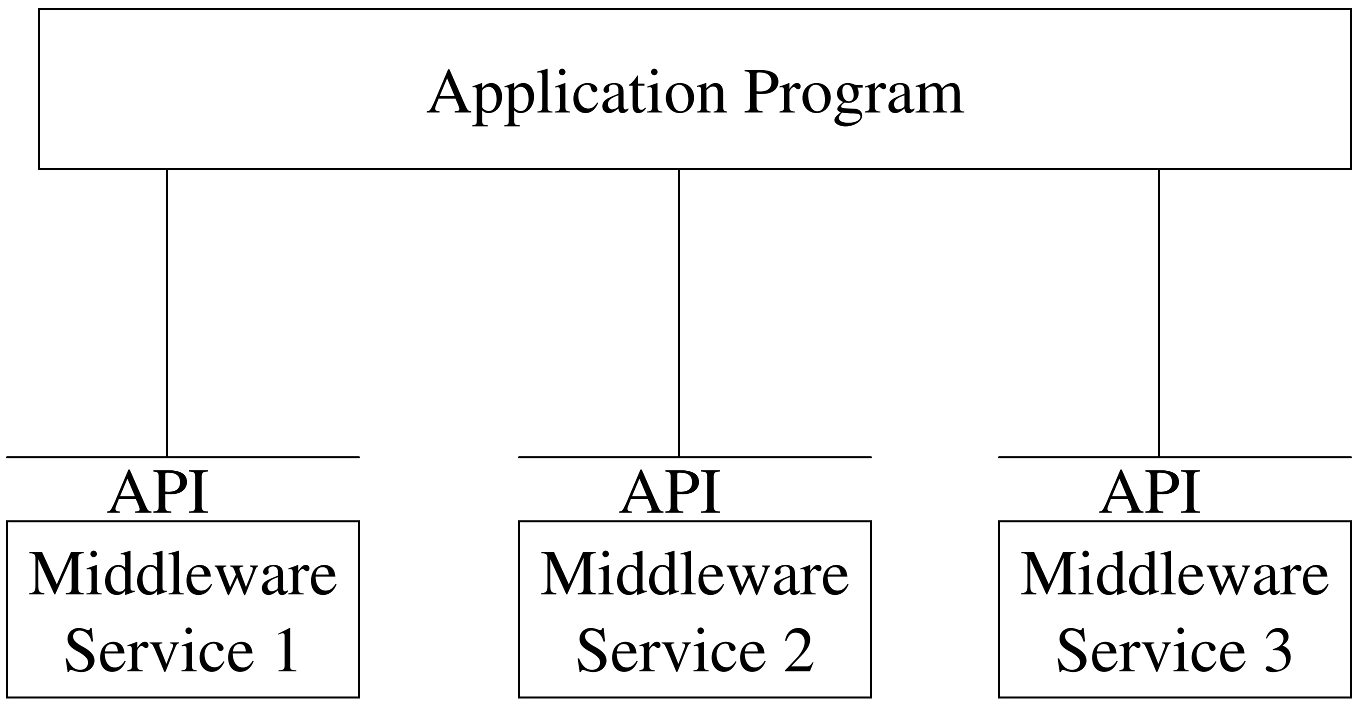
The Evergrowing Alphabet Soup

- CORBA, OMG, CanCORBA, ORBIX, JavaORB, ORBLite, TAO, Zen, RTCORBA, FTCORBA,DCOM, POA,IDL,IOP,IIOP,
- ObjectBroker, Visibroker, Orbix, ObjectBus,ESBs
- MOM – TIBCO TIB/Rendezvous, BEA MessageQ, Microsoft MSMQ, ActiveWorks
- JVM, JINI, RMI, J2EE, EJB,J2ME, JDBC,JTA, JTS,JMS, JNDI,
- Enterprise Middleware Technologies -- BEA WebLogic, IBM WebSphere, TivoliBeans
- ENCINA, Tuxedo, CICS
- XML, XQuery,
- SOAP, Web Services, WSDL, BPEL
- .....

Distributed Systems Middleware

- Enables the modular interconnection of distributed software
  - **abstract over low level mechanisms used to implement resource management services.**
- Computational Model
  - **Support separation of concerns and reuse of services**
- Customizable, Composable Middleware Frameworks
  - **Provide for dynamic network and system customizations, dynamic invocation/revocation/installation of services.**
  - **Concurrent execution of multiple distributed systems policies.**

Modularity in Middleware Services



Useful Middleware Services

- Naming and Directory Service
- State Capture Service
- Event Service
- Transaction Service
- Fault Detection Service
- Trading Service
- Replication Service
- Migration Service

Integration Frameworks Middleware

- Integration frameworks are integration environments that are tailored to the needs of a specific application domain.
- Examples
  - Workgroup framework - for workgroup computing.
  - Transaction Processing monitor frameworks
  - Network management frameworks

Distributed Object Computing

- Combining distributed computing with an object model.
  - Allows software reusability and a more abstract level of programming
  - The use of a broker like entity or bus that keeps track of processes, provides messaging between processes and other higher level services
  - Examples
    - **CORBA**
    - **JINI, EJB, J2EE**
    - **E-SPEAK**
    - **Note: DCE uses a procedure-oriented distributed systems model, not an object model.**

Issues with Distributed Objects

- Abstraction
- Performance
- Latency
- Partial failure
- Synchronization
- Complexity

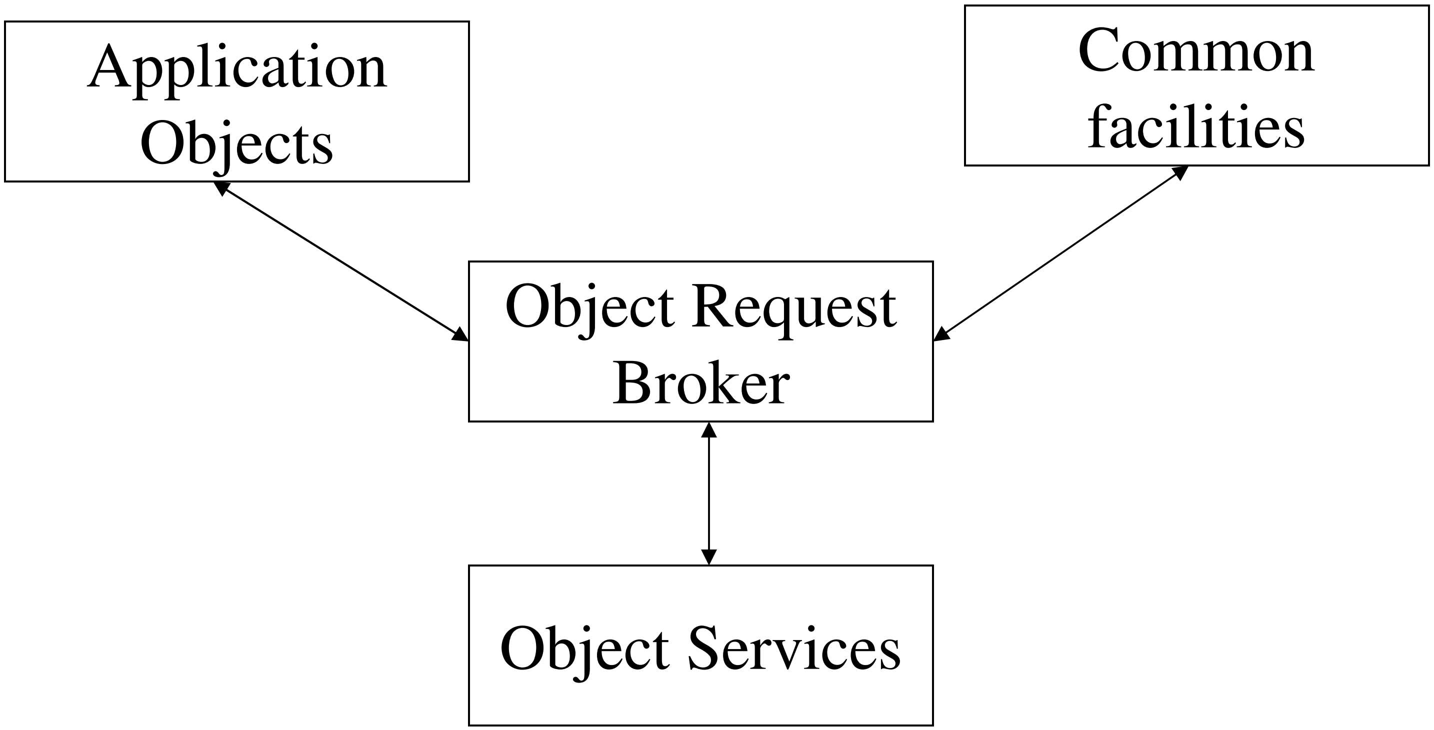
Techniques for object distribution

- Message Passing
  - **Object knows about network; Network data is minimum**
- Argument/Return Passing
  - **Like RPC. Network data = args + return result + names**
- Serializing and Sending Object
  - **Actual object code is sent. Might require synchronization. Network data = object code + object state + sync info**
- Shared Memory
  - **based on DSM implementation**
  - **Network Data = Data touched + synchronization info**

CORBA

- CORBA is a standard specification for developing object-oriented applications.
- CORBA was defined by OMG in 1990.
- OMG is dedicated to popularizing Object-Oriented standards for integrating applications based on existing standards.

The Object Management Architecture (OMA)



OMA

- ORB: the communication hub for all objects in the system
- Object Services: object events, persistent objects, etc.
- Common facilities: accessing databases, printing files, etc.
- Application objects: document handling objects.

Virtual Time & Global States of Distributed Systems

- Asynchronous distributed systems consist of several *processes* without common memory which communicate (solely) via *messages* with unpredictable transmission delays
- Global time & global state are hard to realize in distributed systems
  - We can only *approximate* the global view
    - ***Simulate synchronous*** **distributed system on a given asynchronous systems**
    - ***Simulate*** **a** ***global time***
    - ***Simulate*** **a** ***global state***

Simulate Synchronous Distributed Systems

- *Synchronizers* [Awerbuch 85]
  - Simulate clock pulses in such a way that a message is only generated at a clock pulse and will be received before the next pulse
  - Drawback
    - **Very high message overhead**

Clock Synchronization in Distributed Systems

- Clocks in a distributed system drift:
  - Relative to each other
    - **Logical Clocks are clocks which are synchronized relative to each other.**
  - Relative to a real world clock
    - **Determination of this real world clock may be an issue**
    - **Physical clocks are logical clocks that must not deviate from the real-time by more than a certain amount.**

Synchronizing Logical Clocks

- Need to understand the ordering of events
- Notion of time is critical
- "Happens Before" notion.
  - E.g. Concurrency control using timestamps
- "Happens Before" notion is not straightforward in distributed systems
  - No guarantees of synchronized clocks
  - Communication latency

Event Ordering

- Lamport defined the "happens before" (=>) relation
  - If a and b are events in the same process, and a occurs before b, then a => b.
  - If a is the event of a message being sent by one process and b is the event of the message being received by another process, then a => b.
  - If X =>Y and Y=>Z then X => Z.

If a => b then time (a) => time (b)

Causal Ordering

- "Happened Before" also called causal ordering
- Possible to draw a causality relation between 2 events if
  - They happen in the same process
  - There is a chain of messages between them

Logical Clocks

- Monotonically increasing counter
- No relation with real clock
- Each process keeps its own logical clock Cp used to timestamp events

Causal Ordering and Logical Clocks

- Cp is incremented before each event.
  - Cp = Cp + 1
- When p sends a message m, it piggybacks a logical timestamp t = Cp.
- When q receives (m,t) it computes:
  - Cq = max(Cq,t) before timestamping the message receipt event.
- Results in a partial ordering of events.

Lamport Logical Clock

P1	P2	P3
0	0	0
4	5	3
8	10	6
12	15	9
16	20	12
20	25	15
24	30	18
28	35	21
32	40	24

P1	P2	P3
0	0	0
4	5	3
8	10	6
12	15	**11**
16	20	14
20	25	17
24	30	20
28	35	23
36	40	26

Total Ordering

- Extending partial order to total order

time	Proc_id

- Global timestamps:
  - (T_a, P_a) where T_a is the local timestamp and P_a is the process id.
  - $(T_a,P_a) < (T_b,P_b)$ iff
    - **$(T_a < T_b)$ or $((T_a = T_b)$ and $(P_a < P_b))$**
  - Total order is consistent with partial order.

Problems with Total Ordering

- A linearly ordered structure of time is not always adequate for distributed systems
  - captures dependence of events
  - loses independence of events - artificially enforces an ordering for events that need not be ordered.
  - Mapping partial ordered events onto a linearly ordered set of integers it is *losing information*
    - **Events which may happen simultaneously may get different timestamps as if they happen in some definite order.**
- A partially ordered system of *vectors* forming a *lattice* structure is a natural representation of time in a distributed system
- It resembles Minkowski's relativistic space-time (see Special Theory of Relativity)

Physical Clocks

- How do we measure real time?
  - 17th century - Mechanical clocks based on astronomical measurements
    - **Solar Day - Transit of the sun**
    - **Solar Seconds - Solar Day/(3600*24)**
  - Problem (1940) - Rotation of the earth varies (gets slower)
  - Mean solar second - average over many days

Atomic Clocks

- 1948
  - counting transitions of a crystal (Cesium 133) used as atomic clock
  - TAI - International Atomic Time
    - **9192631779 transitions = 1 mean solar second in 1948**
  - UTC (Universal Coordinated Time)
    - **From time to time, we skip a solar second to stay in phase with the sun (30+ times since 1958)**
    - **UTC is broadcast by several sources (satellites...)**

Accuracy of Computer Clocks

- Modern timer chips have a relative error of 1/100,000 - 0.86 seconds a day
- To maintain synchronized clocks
  - Can use UTC source (time server) to obtain current notion of time
  - Use solutions without UTC.

Berkeley UNIX algorithm

- One daemon without UTC
- Periodically, this daemon polls and asks all the machines for their time
- The machines respond.
- The daemon computes an average time and then broadcasts this average time.

Decentralized Averaging Algorithm

- Each machine has a daemon without UTC
- Periodically, at fixed agreed-upon times, each machine broadcasts its local time.
- Each of them calculates the average time by averaging all the received local times.

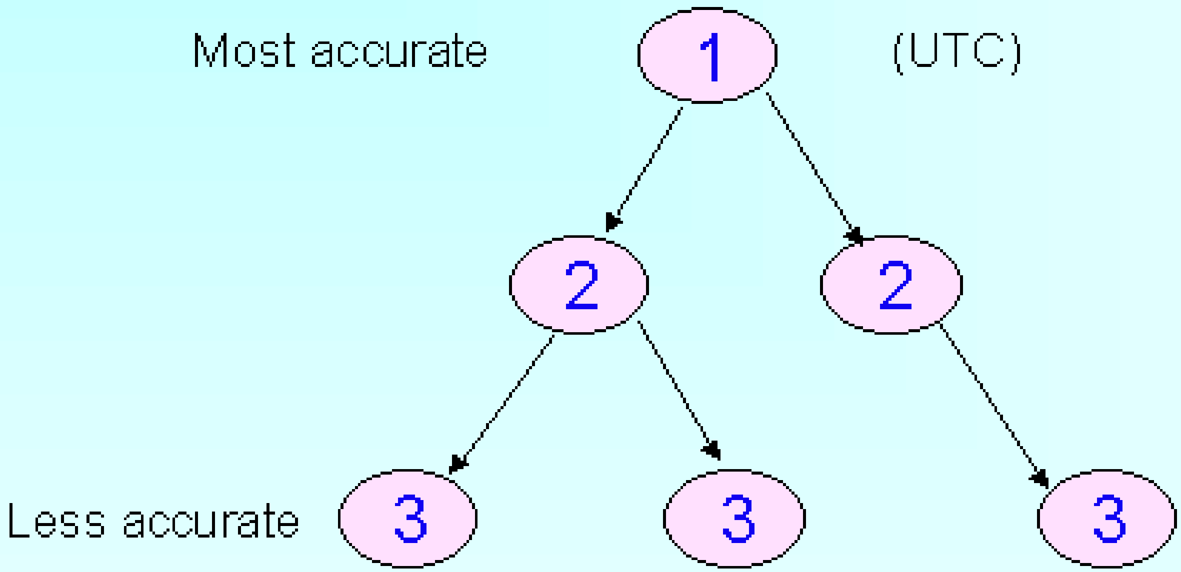
Clock Synchronization in DCE

- DCE's time model is actually in an interval
  - I.e. time in DCE is actually an interval
  - Comparing 2 times may yield 3 answers
    - **t1 < t2**
    - **t2 < t1**
    - **not determined**
  - Each machine is either a time server or a clerk
  - Periodically a clerk contacts all the time servers on its LAN
  - Based on their answers, it computes a new time and gradually converges to it.

The Network Time Protocol

- Enables clients across the Internet to be synchronized accurately to the UTC
  - Overcomes large and variable message delays
  - Statistical techniques for filtering can be applied
    - **based on past behavior of server**
  - Can survive lengthy losses of connectivity
  - Enables frequent synchronization
  - Provides protection against interference
  - Uses a hierarchy of servers located across the Internet (Primary servers connected to a UTC time source).

Hierarchy in NTP



Time Manager Operations

- Logical Clocks
  - C.adjust(L,T)
    - **adjust the local time displayed by clock C to T (can be gradually, immediate, per clock sync period)**
  - C.read
    - **returns the current value of clock C**
- Timers
  - TP.set(T) - reset the timer to timeout in T units
- Messages
  - receive(m,l); broadcast(m); forward(m,l)