

LECTURE -1

DISTRIBUTED SYSTEMS

26 AUGUST 2019

Prof. D. S. Yadav
Department of Computer Science
IET Lucknow

DISTRIBUTED SYSTEMS

1

BOOKS

- **Advanced Concepts in Operating Systems**, Mukesh Singhal and Niranjana G. Shivaratri McGraw Hill
- **Distributed Systems: Concepts & Design**, Coulouris, Dollimore, Kindberg, Pearson Education.
- **Distributed Computing**, Singhal & Khemkalyani, Cambridge University Press.
- **Distributed Systems: Principles and Paradigms**, Tanenbaum & van Steen. Prentice Hall.
- **Introduction to Distributed Algorithms**, Gerard Tel, Cambridge University Press
- **Distributed Algorithms**, Nancy Lynch, Morgan Kaufman
- **Transaction Processing**, Jim Grey & Andreas Reuter, Morgan Kaufman
- **Database Replication**, Bettina Kemme, Morgan Kaufman

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

WHAT IS A DISTRIBUTED SYSTEM?

A very broad definition:

–A set of autonomous processes communicating among themselves to perform a task.

–Collection of autonomous computer system cooperating to each other by exchange of messages.

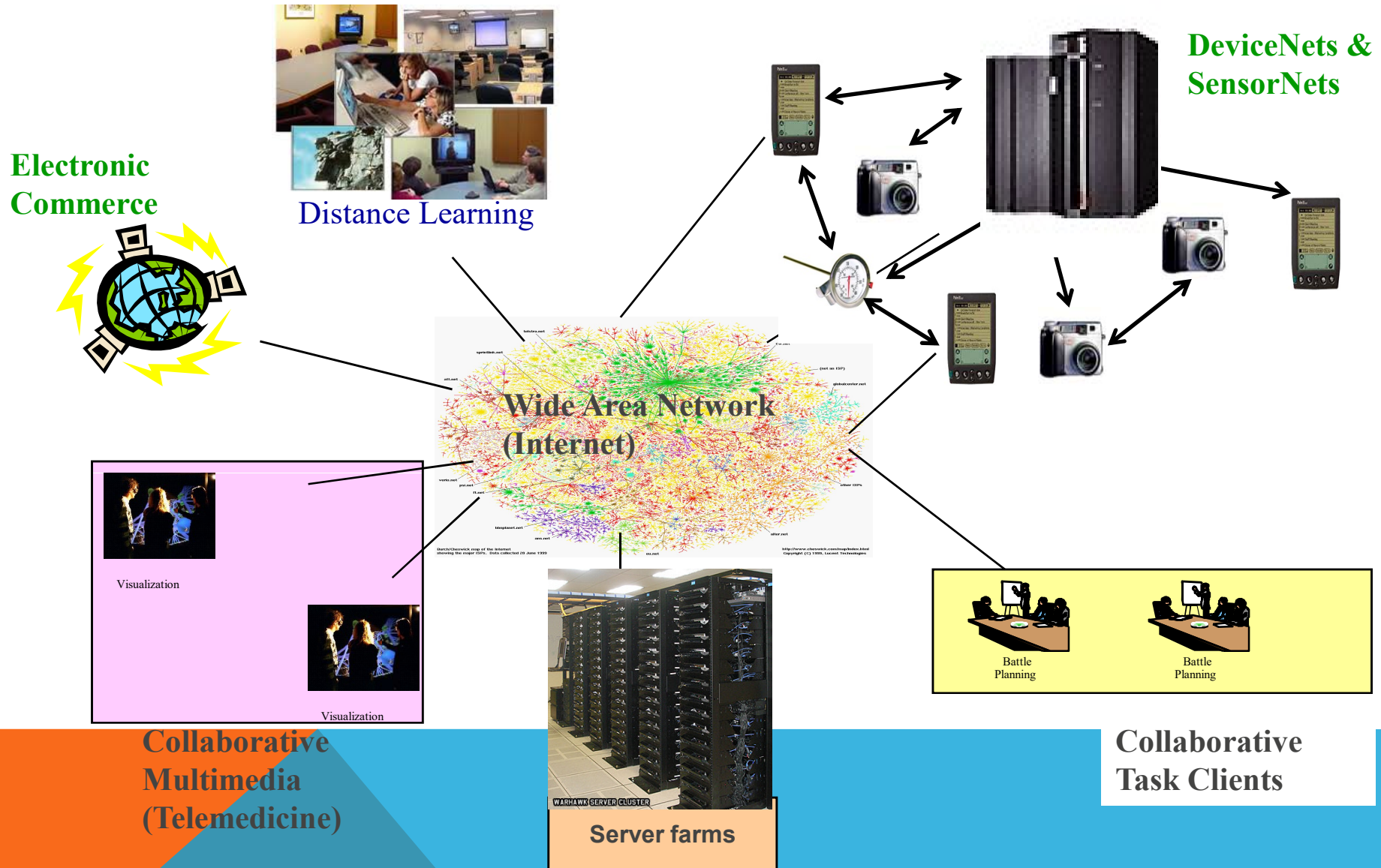
- **Other Definitions :**

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

Issues:

- Un-reliability of communication**
- Lack of global knowledge**
- Lack of synchronization and causal ordering**
- Concurrency control**
- Failure and recovery**

Next Generation Information Infrastructure



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

DISTRIBUTED SYSTEMS

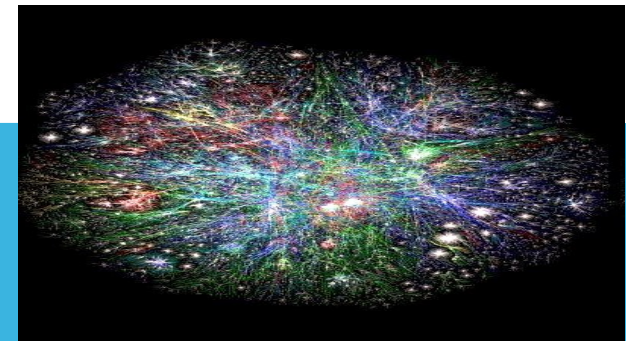
5

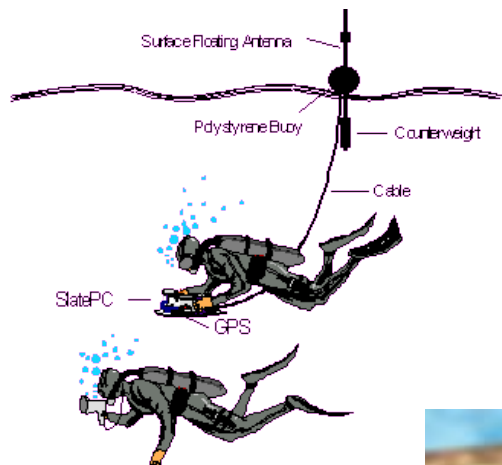
CHARACTERIZING DISTRIBUTED SYSTEMS

- **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.

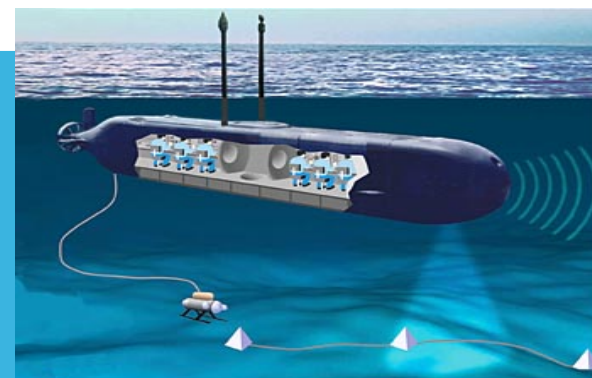
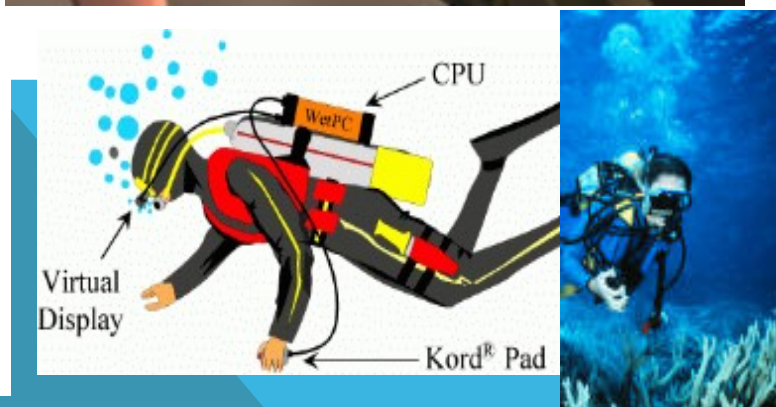
EXAMPLES OF DISTRIBUTED SYSTEMS

- **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





Mobile & ubiquitous distributed systems



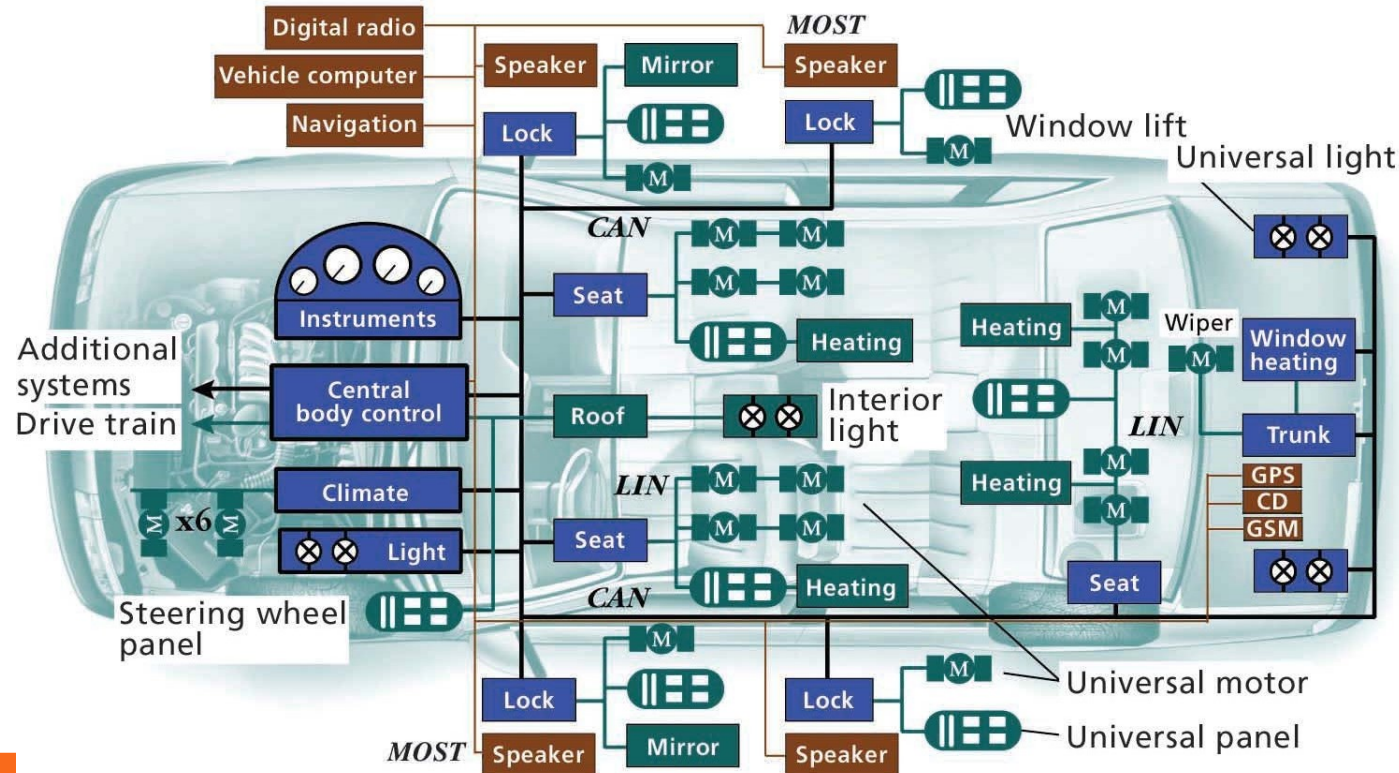
ADVANTAGES

- Resource Sharing
- Higher Performance
- Fault Tolerance
- Scalability

CHALLENGES

- Reliable communication – *Theoretically impossible?*
- Concurrency problems / Synchronization
- Lack of Shared Memory
- Lack of Common Clock

EXAMPLE: AUTOMOTIVE CONTROL



CAN Controller area network
 GPS Global Positioning System
 GSM Global System for Mobile Communications
 LIN Local interconnect network
 MOST Media-oriented systems transport

Source: Leen and Heffernan,
 IEEE Computer, Jan 2002

WHY IS IT HARD TO DESIGN DISTRIBUTED SYSTEMS?

- **The usual problem of concurrent systems:**
 - **Arbitrary interleaving of actions makes the system hard to verify**
- **Design of Distributed System/Algorithms/Protocols**
 - **Distributed Systems Design may be highly deceptive**
 - **Model checking**
 - **Theorem Proving**
- +
- **No globally shared memory (therefore hard to collect global state)**
- **No global clock**
- **Unpredictable communication delays**

MODELS FOR DISTRIBUTED ALGORITHMS

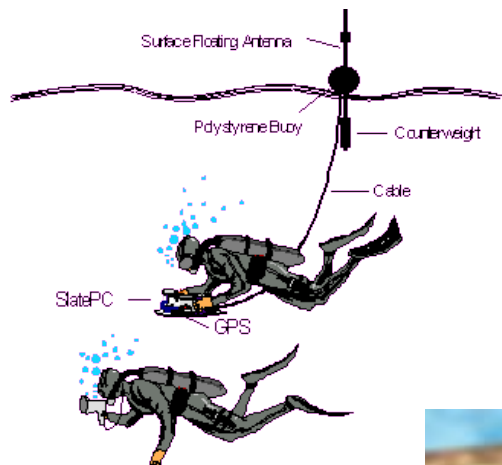
- **Topology:** Completely connected, Ring, Tree etc.
- **Communication:** Shared memory / Message passing
(~~reliable?~~ Delay? FIFO/Causal? Broadcast/multicast?)
- **Synchronous/asynchronous**
- **Failure models:** Fail stop, Crash, Omission,
Byzantine...
- *An algorithm needs to specify the model on which it is supposed to work*

COMPLEXITY MEASURES

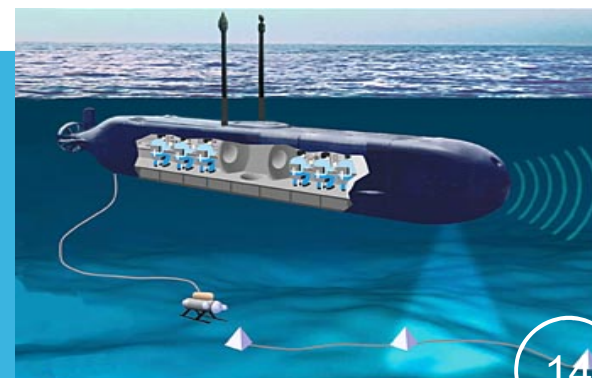
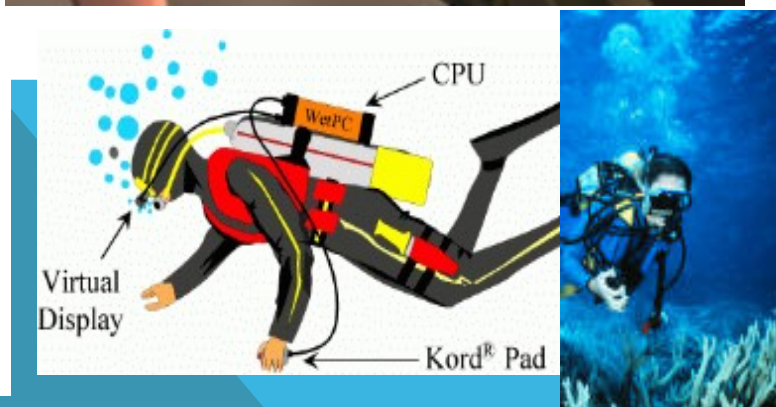
- **Message complexity:** No. of messages
- **Communication complexity / Bit Complexity:** no. of bits
- **Time complexity:**
 - For synchronous systems, no. of rounds
 - For asynchronous systems, different definitions are there.

SOME FUNDAMENTAL PROBLEMS

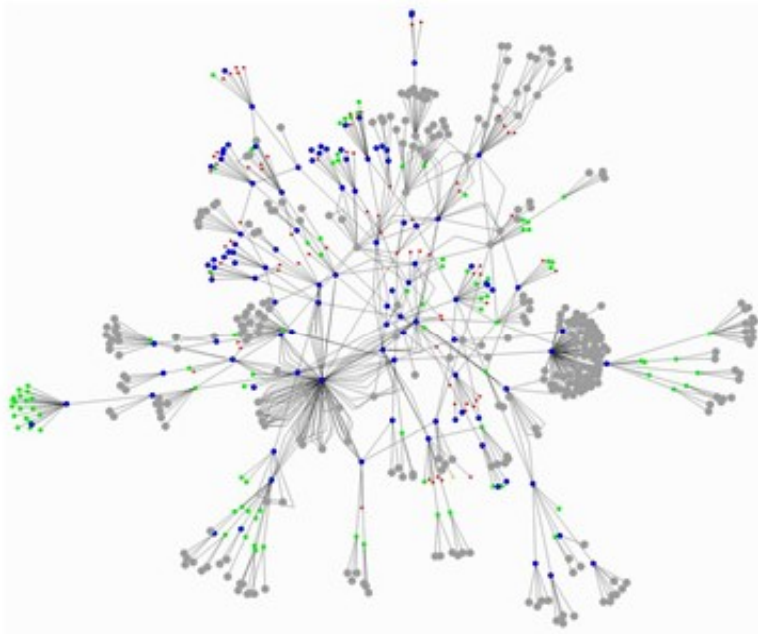
- **Ordering events in the absence of a global clock**
- **Clock synchronization/ Concurrency Control**
- **Capturing the global state**
- **Mutual exclusion**
- **Leader election/ Agreement/Consensus**
- **Termination detection**
- **Agreement protocols**
- **Distributed Database**
- **Distributed Deadlock Detection**
- **Failure Recovery & Fault Tolerance**
- **Data Security**



Mobile & ubiquitous distributed systems



PEER TO PEER SYSTEMS



P2P File Sharing

Napster, Gnutella, Kazaa, eDonkey,
BitTorrent

Chord, CAN, Pastry/Tapestry, Kademlia

P2P Communications

MSN, Skype, Social Networking Apps

P2P Distributed Computing

Seti@home

Use the **vast resources** of machines at **the edge of the Internet** to build a network that allows resource sharing **without any central authority**.

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 TO DESIGN & DEVELOP?

- **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 SYSTEMS

- **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

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
- Approach : Model Checking / Theorem Proving

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

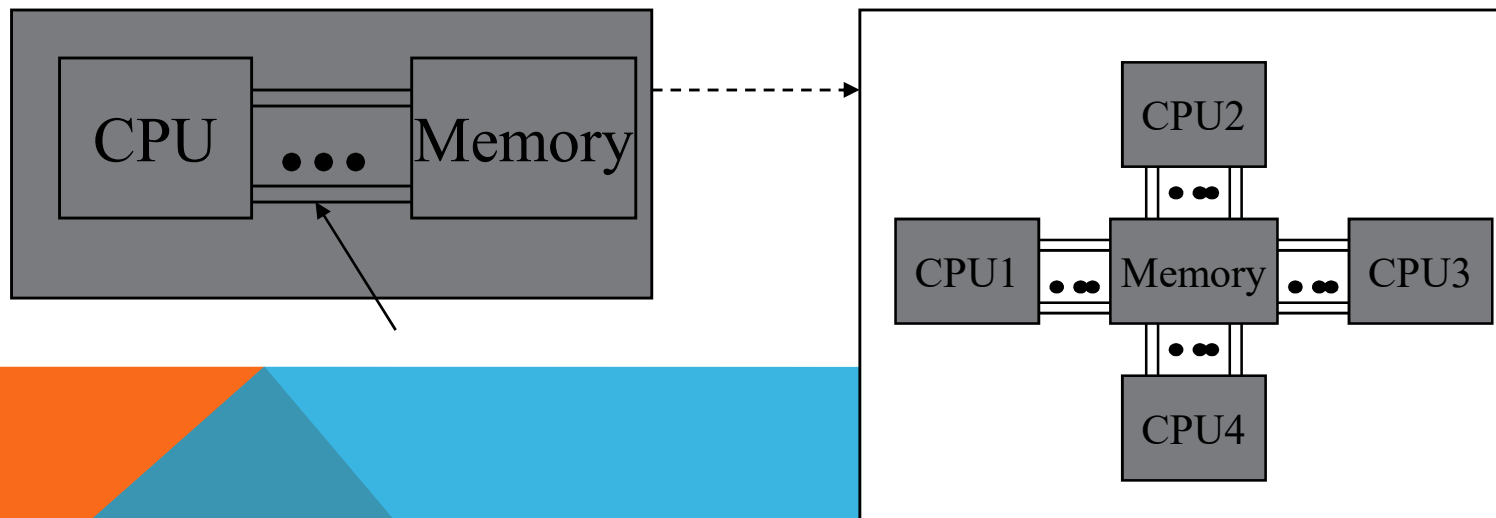
- **Unreliable communication**
 - Best effort, No ACK's or retransmissions
 - Application programmer designs own reliability mechanism
- **Reliable communication / group communication**
 - Different degrees of reliability
 - Processes have some guarantee that messages will be delivered.
 - Reliability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

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”

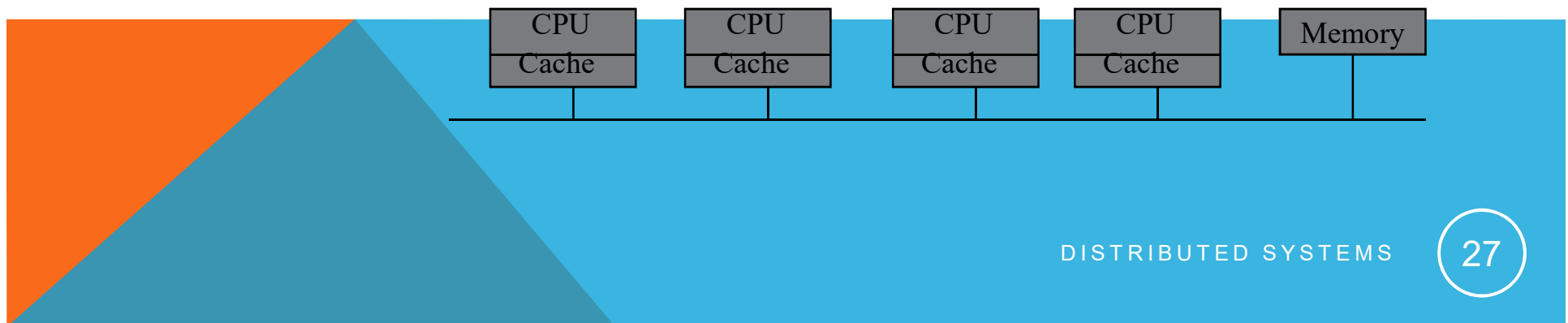
DISTRIBUTED SHARED MEMORY

- **Communication 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



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

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.

DISTRIBUTED SYSTEMS MIDDLEWARE

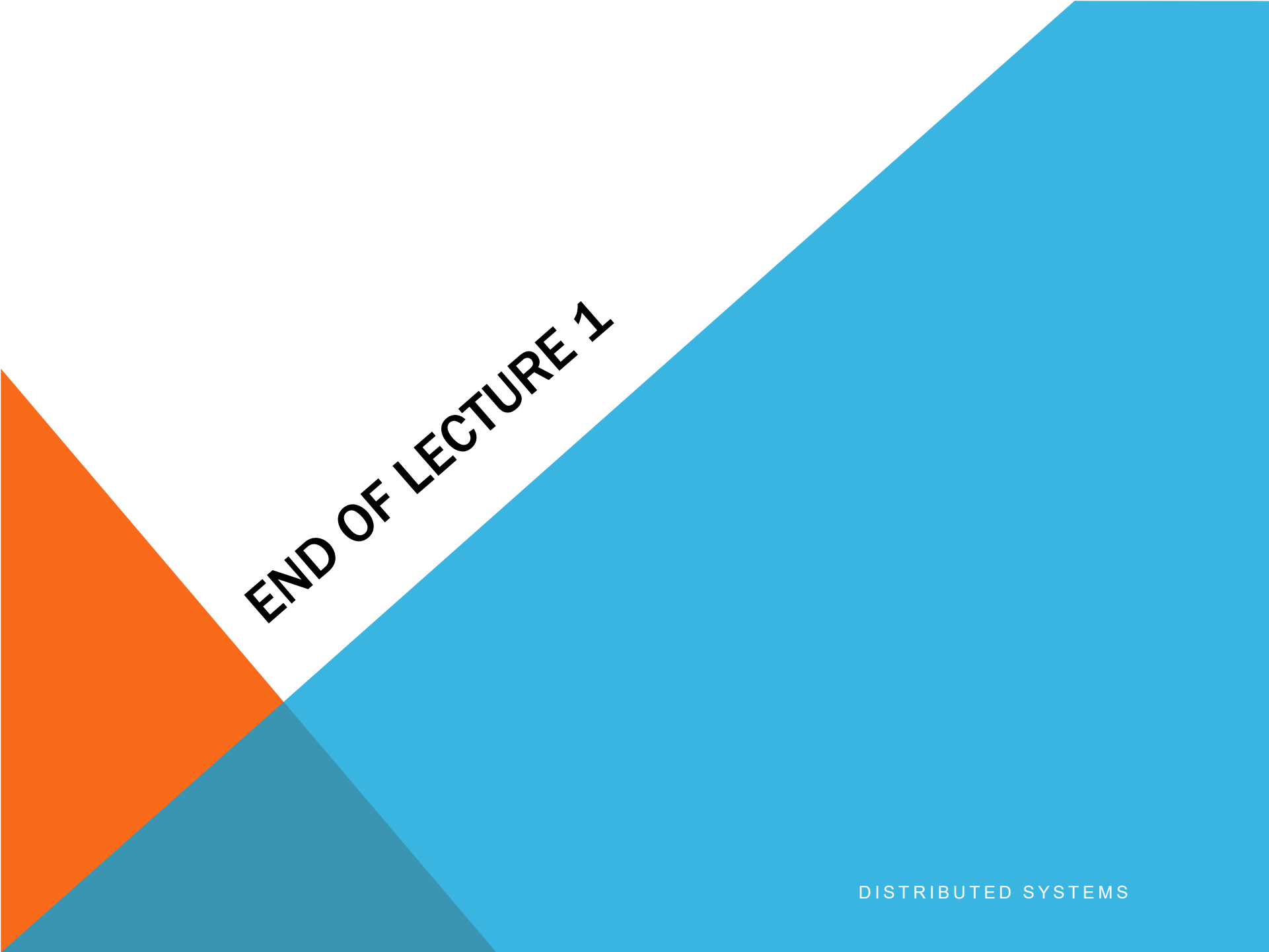
- Middleware is the software between the application programs and the operating System and base networking
- Integration Fabric that knits together applications, devices, systems software, data
- Middleware provides a comprehensive set of higher-level distributed computing capabilities and a set of interfaces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

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.

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



END OF LECTURE 1

DISTRIBUTED SYSTEMS