Aims:

- **Resource Sharing**: costly resources (high-quality printers and expensive programs) can be shared
- **Computation Speedup**: Algorithms can run concurrently
- **Reliability**: Failure of one site does not imply failure of the whole system

Redundancy prerequisite

Especially last point difficult to fulfill

**Design Issues**

Several Levels of Distribution possible listed from tightly coupled to loosely coupled

- Shared memory
- Shared file system
- Bus Systems
- Switching Systems

Key Problem Areas:

- Transparency: pretend to be one computer
- Reliability: want availability, fault tolerance
- Performance
- Scalability: avoid centralised tables and algorithms

**Communication**

Simplest way of communication:
Client/Server Model

Idea: group of processes (servers) used by clients
Advantage: Simple communication

Simple enough to study several problems
First problem: Addressing! Possible Solutions:

- Hardware address into client code: inflexible
- Broadcasting: works only on local networks
- Name Servers: Ask special host
  Example: Domain Name Service, DNS

Second Problem:
Blocking (synchronous) vs. non-blocking primitives
Conceptual ease vs. performance

Third Problem:
Reliable vs. unreliable primitives
Where does the error correction go?
Kernel (once for all) vs. application (possibly more efficient)
Remote Procedure Call

Very simple idea: execute procedure on different host
Goal: total transparency

Basic Schema:
- Client sends arguments to server
- Server executes call
- Server sends results back

Difference to local call hidden in kernel routines

Details complicated:
- Have to transfer parameters
- deal with failures

Problems with parameter passing
- Different representation on different machines: either common format (inefficient), or store format in message
- What to do with call-by-reference parameters? Can copy arrays, but not arbitrary data structures

Generating Timestamps

Unique timestamps needed for co-ordination
No problem in monoprocessor system: use system clock
Not possible in distributed systems
One way out:
- Each host maintains logical clock which is advanced with each event
- All message from host contain logical clock
- When message with greater logical clock is received, increase own logical clock to this value
- with two identical timestamps, let host number decide

Failure problems more complicated
Have several cases
- Client cannot locate server: Generate exception
  \[\Rightarrow\] transparency lost
- Lost Request Messages: use timer
- Lost Reply Messages: client timer insufficient:
  could execute operation more than once
  Solution: use sequence numbers
- Server Crashes: Sequence numbers not enough: When did crash occur?
  Can guarantee \textit{at least once}, \textit{at most once} semantics, but not \textit{exactly once} semantics
  \[\Rightarrow\] have to have call-specific remedies
- Client Crash: leaves orphans (unwanted computations)
  No general way of getting rid of them
Mutual Exclusion algorithms

A Distributed Version (Ricart and Agrawala)
Assume reliable messages, unique timestamps
Following steps:

- Process trying to enter critical section:
  sends to all other processes name of section and unique timestamp

- Process receiving such a message:
  - Sends back OK if not interested in critical section
  - Queues message if already in critical section

- Receiver wants to enter critical section
  ⇒ enters critical section if its request has lower timestamp and queues message, otherwise sends OK

Grants mutual exclusion without deadlock or starvation

Problems:

- Requires that everyone knows about all other hosts
- Algorithm fails if one host fails
  ⇒ Reconstruction of network necessary in this case

Suitable for networks where configuration is stable

A Token Ring Algorithm

Assumption: Network organised on a (physical or logical) ring, i.e., each node has unique successor in line
Simple algorithm:

- At initialisation, generate token
- Pass token around continuously
- Process wanting to enter a critical region waits for token
- After leaving critical section, process passes token to next neighbour

Properties:

- Detecting lost tokens difficult: Time spent in critical region unbound
- Detecting dead processes easy if sent token is acknowledged

Election Algorithms

Problem: Select new co-ordinator
Assumption:
Know id of every host on the network
First example: Bully algorithm

- P sends message to all hosts with higher number
- No response ⇒ P wins and is co-ordinator
- Answer received ⇒ host with higher number has taken over

Second Example: Ring Algorithm

- any host may send Election message
- passed around the net, which each host id added
- If original host gets message, determines co-ordinator and sends new message around
- After it has gone round, host removes it