**Cristian’s algorithm**

A sends a message to B with its timestamp T1 set to its local clock value

Upon reception, B records the current timeT2 of its local clock

B returns a response message to A with T2 and the current time T3 of its clock

Upon reception, A records the current time T4 of its local time

If both message transmissions take roughly the same time

**Round-trip time** = (T2 – T1) + (T4 – T3)

**Offset of A relative** to B = T4 – T3 – RTT / 2

**A can be re-adjust its time to T3 + RTT / 2**

This results are not accurate but give an acceptable approximation

**Network Time Protocol (NTP)**

Servers are divided into strata

A server having an atomic clock operates at stratum 0

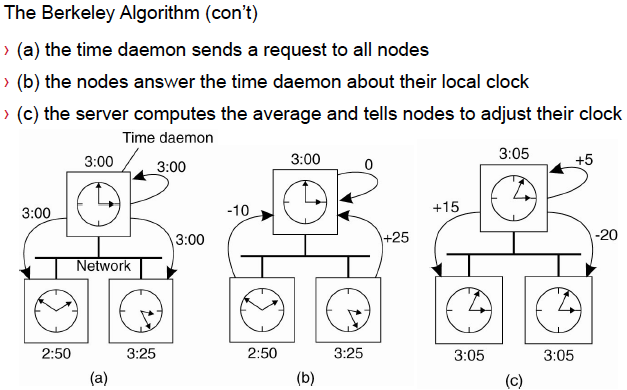
A server synchronized with a server at stratum j operates at stratum j+1

Steps:

Run Cristian’s Algorithm 8 times and records the 8 output pairs <offset, RTT / 2>i (0 <= i < 8)

The pair with the minimum RTT /2 is chosen as the most precise output

The server with the higher stratum adjusts



**Logical Time**

**Happens-before relation** “a happends before b”, denoted by a->b if:

a and b are events from the same process such that a occurs before b

if a is the event of a message being sent by one process and b is the event of the same message being received by another process

or it exists some event c such that a->c and c->b

If it is not possible to determine either a happens before b or b happens before a, we said a and b are concurrent

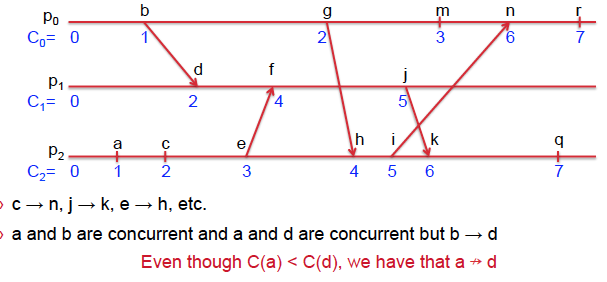
**Goal**: to make the happens-before relation visible to processes

**Steps**: Each process has initial variable Ci to be 0.

Upon sending, increment Ci and sends it in the message

Upon reception of Cj from pj, pi increments Ci such that Ci > Cj

***Guarantee: If a->b then C(a) < c(b)***



**Vector clocks**

**Goal**: to make also the did-not-happen-before relation visible

**Steps**: Each process has initial vector VCi of integer all 0

Upon sending, pi increments VCi[i] and then send a message m with timestamp ts(m) = VCi

Upon reception of ts(m) from pj, accept the max for all other processes and increment its own by 1.

**Result:**

One is **lower** than another, if **each** of its coordinates is **lower** than the corresponding coordinate of the other

VC(a) ≤ VC(b) if for all i (0 ≤ i < n) VC(a)[i] ≤ VC(b)[i]

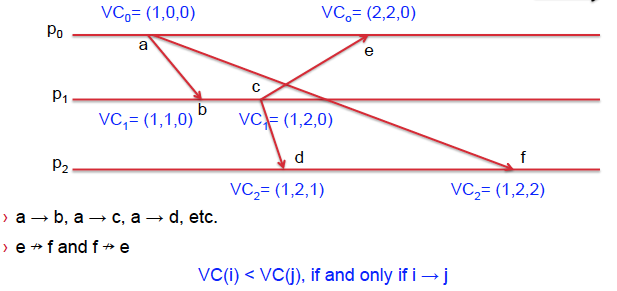
- One is strictly lower than another, if it is lower and different from the other

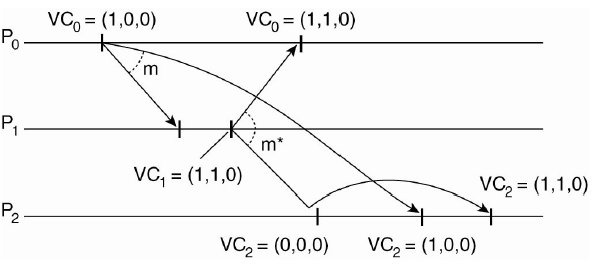
VC(a) < VC(b) if VC(a) ≤ VC(b) and VC(a) ≠ VC(b)

**If VC(a) < VC(b) then a → b**

**- If a → b then VC(a) < VC(b)**

**VC(a) < VC(b) if and only if a → b**





**Eventual consistency**: eventually all replicas are consistent

Assume: no concurrent updates (i.e. the only type of conflict is read-write conflict)

**Primary-backup protocol**

All writes on x are forwarded to a primary server for x

The primary does the update and forwards the request to backup servers

Each backup updates and acknowledges the primary

The primary sends a response to the client

Read operations can be done locally without the above process

**Pro**: guarantees sequential consistency

**Cons**: delayed answer to client due to blocking execution

Local-write primary-backup

When a process wants to update a data it locates it and moves the item to its location

Become the new primary and updates it locally.

It tells the backup to execute the update locally and they will acknowledge the new primary

**Name**

**What is it for**

To act on a particular resource out of many (URL: Uniform Resource Locator)

To share a resource with other processors (printer address)

To allow user communication (e.g. email address)

**Name service:**

Provides a client with data about named objects in distributed systems

Remote object references

NFS file handles (just identifiers)

A name is resolved when it is translated into data about the named resource of object, often to invoke an action upon it

Given an object, one of its attribute is a value of a property associated with it

A binding is the association between a name and an object.

**Two kinds of URI**

Uniform Resource Locator (URL):

Uniform Resource Name (URN) : ISBN, used to name a resource

**Multicast**: sending messages to many

Each group has a multicast address

Message to this address are delivered to all group members

Broadcast, and multicast are costly because a lot of messages are wasted.

**Solution:**

using a **name server**, it maintains the name-address mapping

**Name resolution**: clients contact such server to get the information

**Name space**: organization of all the names

The node at the top with no incoming edges i the root

Each leaf is a file and each inner node is a directory

Each directory maintains a directory table with entries <label: “node identifier”>

The concatenated label of a graph path is a path name

A path name starting from the root is an absolute path name

**Alias**: another name for the same entity

**Symbolic links**

A leaf node stores only an absolute path name; resolving a path name leading to this node returns another path name, that is resolved in turn

**Caching**

Exploiting locality by caching data in memory

Accessing a remote entity can take time

Network access

Disk access

Caching consists in keeping a copy locally (in memory)

Accessing the remote entity creates a copy in cache

Re-accessing the same entity accesses the cached copy instead of remote entity

**Cache-hit:** if the copy is in the cache, the access returns successfully

**Cache-miss:** if the copy is not in the cache, the remote entity must be accessed

Policy

Tells which part should be overridden by new copies when the cache gets full

**Least recently used (LRU) policy**

When the cache is full, and another access occurs, the least-recently used block or path entry is overridden by the new access entry

Well suited for name translation, and for disk blocks as long as |cache| > |file|

**Domain Name System**

DNS

Global layer: highest level nodes, root and its children, nodes that do not change often

Administrational layer: organizations, nodes, change sometimes(server more responsive and non-faulty)

Managerial layer: hosts of local network, change often

**Time**

**Why is time important?**

The updates happening in two different places can be inconsistent if the time is not handled in a distributed system

**Physical Time**

Machine quartz crystal

For one computer, it works fine. The time is shared among different programs

Multiple CPUs imply multiple clocks

The difference returned by two clock values is the clock skew

**International Atomic Time (IAT)**

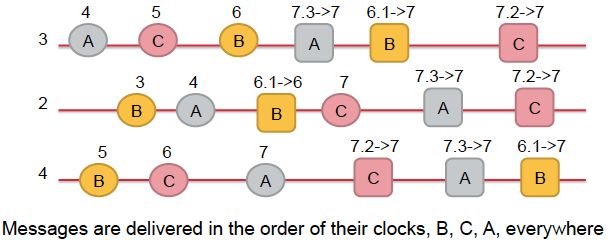
**Mean solar second**

**Multicast**

Ordering of messages:

Totally ordered: for all messages m1 and m2 and all processors pi and pj, if pi delivers m1 before it delivers m2, then m2 is not delivered at pj before m1 is.

Causally ordered: for all messages m1 and m2 and every processor pi, if m1 happens before m2, then m2 is not delivered at pi before m1 is.



Linearizability:

Real-time precedence:

If an operation O1 returns before another operation O2 is invoked, then o1 precedes o2 with respect to real-time.

Definition:

The result of each execution is equivalent to a sequential execution that respects the real-time precedence

Is the same as if the operations by all processes on the same data store were executed in some sequential order and the operations of each individual process appear in the sequence in its program order

Causal consistency < sequential consistency

Serial -> sequential consistent ->casually consistent

Serializable != linearizable

**Solving the consensus problem aims at guaranteeing that:**

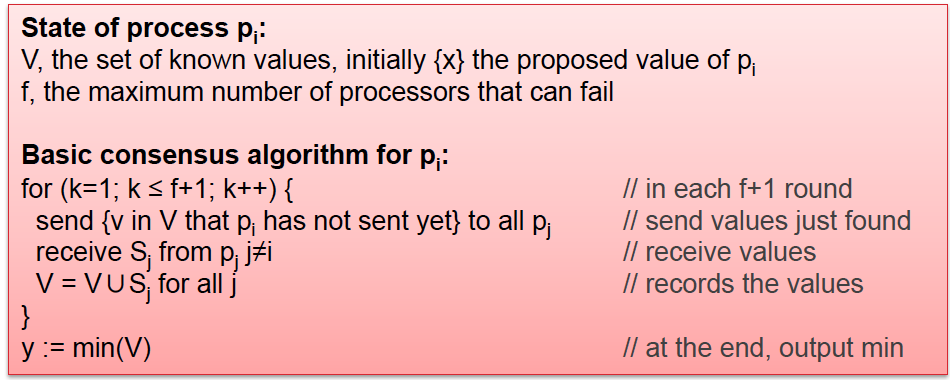
**Termination**: in every admissible execution, yi is eventually assigned a value for every non-faulty processor pi.

**Agreement**: in every execution, if yi and yj are assigned, the yi=yj, for all non-faulty processors pi and pj.->they agree on the same value for y.

**Validity**: in every execution, if for some value v, xi=v for all processes pi, and if yi is assigned for some non-faulty process pi then yi = v. The value decided has been proposed

**Crash-tolerant synchronous solution**

Unicast, crash failures, a max of f processes can fail

****

**Message Oriented Middleware**

**Characteristic:**

Do not require sender and receiver to be active during communication (!= RMI)

Offering intermediate storage capacity for messages

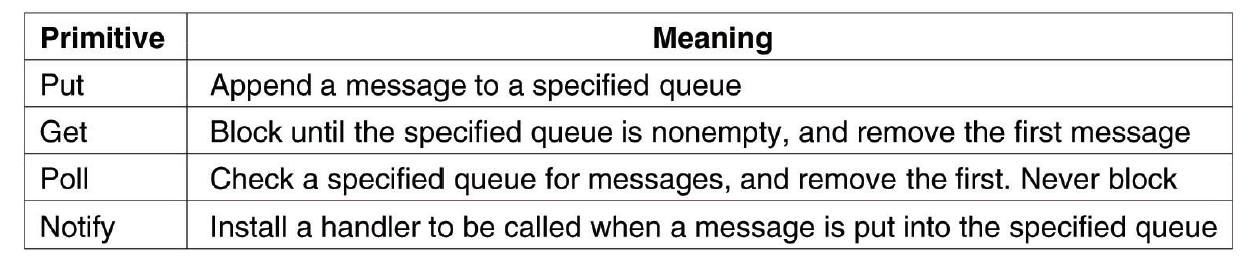
Typically target long (> minutes) message transfers

**Principles**

Each application maintains its local queue

Sends messages to other applications’ queues

Unaware of the reading of its messages (recipient can be down at sending time, because of the queue)



**Message Loss**

Cause:

Network are in general unreliable

Messages can be lost

Example:

A server receives too many requests and cannot treat all

A router drops the message because the queue is full

**TCP / IP**

Connection oriented: end-to-end agreement, reliable data transmission

Before transmission, both sides cooperate through communication

segments numbered

divides into segments and transmits them as IP packets

A check sum is attached

not match its checksum, the segment is dropped

re-ordered by the sequence numbers before sending to higher layers

send acknowledgements to the sender

If the sender receives an acknowledgment, it deletes the appropriate numbers. not acknowledged within a timeout, the sender will resend it.

**Conventional Procedure Call**

Call-by-value

Calling the procedure doesn’t affect the value of the argument variable

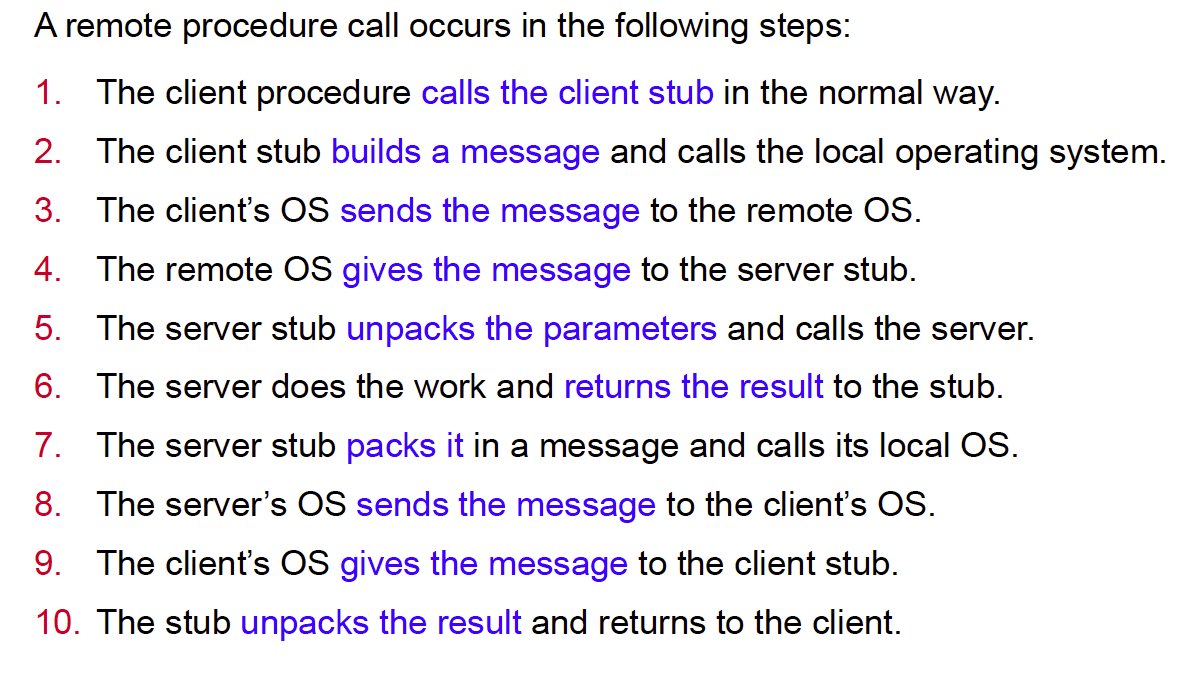
Call-by-reference

Calling a procedure will change the value because it is passed in with a pointer

Call-by-copy / restore

When the procedure returns, the variable will be changed by not during the execution

When doing the remote procedure call (RPC),



The type (int, string) needs to be sent because different architectures have different data representations. Big endian or little endian. Missing this type information may lead to misinterpretation.

**Asynchronous RPC**

The client sends the requests and continue doing its own stuff before the server responds to the request

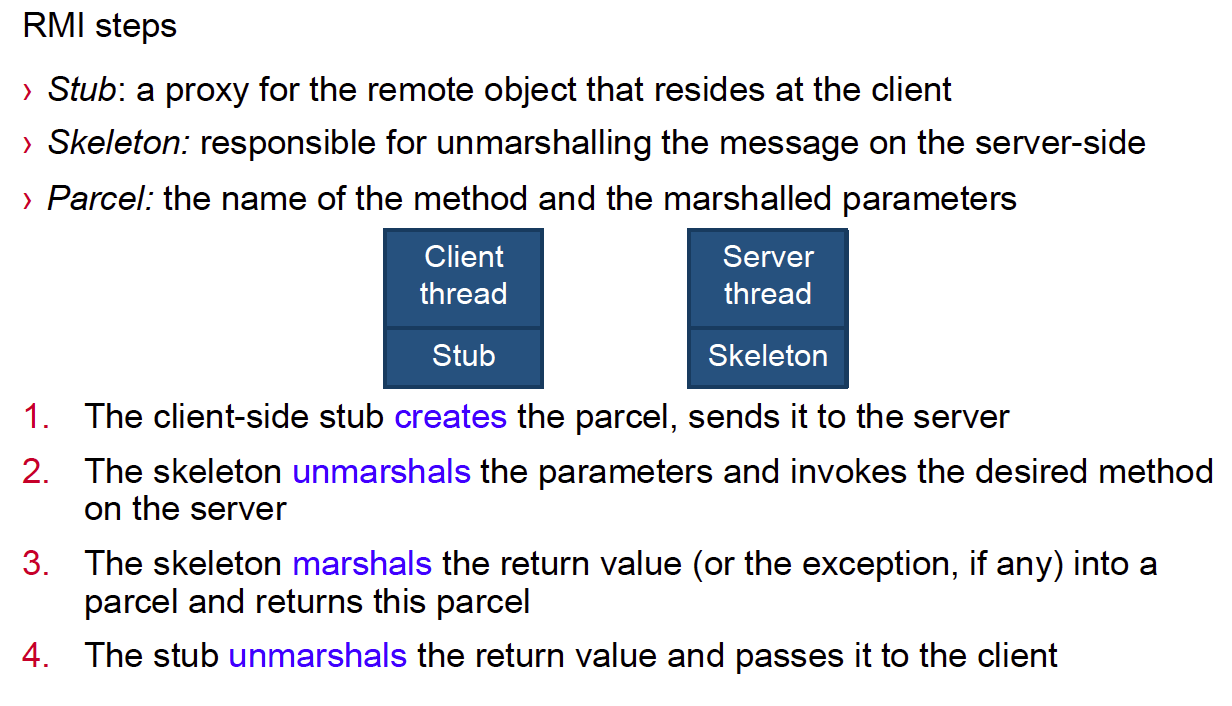
**RMI and RPC**

RPC is procedural: only functions or procedures may be called

RMI is objected-oriented: invocation of methods on remote objects

RPC parameters are generally data structures

RMI parameters can be primitive data types as well as objects



**Quorum system**

A set S of quorums that are mutually intersecting sets

For any pair of quorums Q1, Q2 in the quorum system S

Q1 intersect Q2 must not be empty set

**Write**:

forward the update request along with a version that is a pair of timestamp and node identifier to all replicas of a quorum

Once acknowledged by a quorum of replicas, respond to client

**Read**:

when a read on x is requested, forward the request to all replicas belonging to a quorum and picks the response with the maximum timestamp

Return the chosen copy to the client

Gifford’s approach

N replicas

Upon read request, forward the request to at least Nr servers (a read quorum)

Upon writing, forward the request to at least Nw servers (a write quorum) such that

Nr+Nw > N and Nw>N/2

**Property of transactions**

Atomic, Consistent, Isolated, Durable

**Serializability**: exists an equivalent sequential execution

No cycle in the precedence graph, R to the next W, and W to the next R

Casually related: one happens before the other, Ws that are in the same process, the results should be viewed in the same order by all other processes.

**Component organization**

1, **Layer-based architecture**: requests go downward and responses go upward

2, **Object-based architecture**: objects communicate through Remote Procedure Calls(RPC)

**Communication organization**

1, **Event-based architecture**: communication through events that optionally carry data (subscribers get their desired events delivered)

2**, Date-centred architecture**: through a shared repository, that contains data(e.g. files in a distributed file system)

**Client and server model**

1. The client requests the service whereas the server provides the service
2. The client and the server can be hosted on different machines
3. The communication follows a request-reply model

**Stateless vs stateful server**

**Stateless server**: does not record the state of its clients

i.e. I don’t know your name, tell me, and what do you want?

**Stateful server**: maintains persistent information about its clients

i.e. Yeah, I am waiting for you. I know who you are, tell me what you want

In a stateless server, the message itself should contain the information about the client. In the stateful server, the message can be purely what the client want, which can be processed faster

No recovery needed for stateless server. But it is necessary to recover the state of the client for a stateful server.

Example : NFSv3 (stateless server), Andrew file system (stateful server)

**Application layering**

**Traditional three-layered view:**

The user interface layer: contains the feature to control the application

The processing layer: contains the function of the application

The data layer: contains the data of the application

**The peer-to-peer model**

Every machine acts similarity

Each machine is both a client and a server

No centralized control: the responsibility is distributed evenly

Even the program executing on each machine is similar

**Concurrency**

Threads:

**POSIX threads:**

Current activity: program counter

Stack: temporary data vary by threads

No data, code, files -> all threads share the same piece of code, data and files

**By using threads:**

We don’t need to communicate through interprocess communication, which will cost a lot of time doing the context switch

**Only** do the communication **through the memory**

**No context switches**

User-level threads library vs. kernel-scheduled thread

***Kernel-scheduled thread***

***Advantages***

Kernel can simultaneously schedule multiple threads from the same process on multiple processes.

If one thread in a process is blocked, the Kernel can schedule another thread of the same process.

Kernel routines themselves can be multithreaded.

***Disadvantages***

Kernel threads are generally slower to create and manage than the user threads.

Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

***User-level threads***

***Advantages***

Thread switching does not require Kernel mode privileges.

User level thread can run on any operating system.

Scheduling can be application specific in the user level thread.

User level threads are fast to create and manage.

***Disadvantages***

In a typical operating system, most system calls are blocking.

Multithreaded application cannot take advantage of multiprocessing.

**Process vs. threads**

**Processes**

Isolated: prevents one process from interfering with another

Inefficient: starting / terminating a process and context switches are costly

**Threads**

Non-isolated: avoiding incorrect interferences makes programming harder

Efficient: a thread is a lightweight version of a process

**Mutex**

**Centralized algorithm**: a coordinator grants permissions(coordi crashes)

**Decebtralized** **algorithm**: (starvation, low efficiency)

n replicates, each has a coordinator. Access requires a majority vote from more than n/2 coordinators

(Will not remember that he has grants the permission if crashed)

**Ricart and Agrawala Algorithm: (crash of processes)**

The replies saying Ok go ahead are only sent when the process has no interest in the shared resource or it has lower priority. U go first.

After the access to the resource, the process will reply to others saying I am done, no more interest, u guys can go ahead.

**Token Ring:**

Processes organized in a logical ring, the one holding the token can access the resource.

**Leader election:**

**The Bully algorithm**

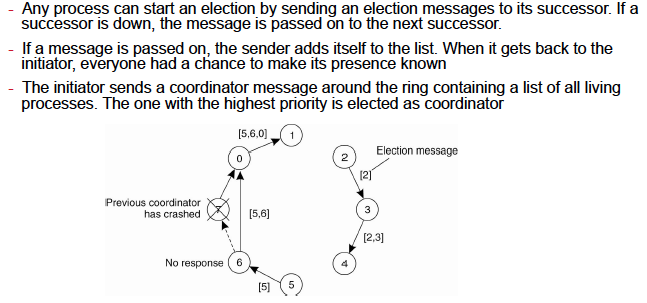
P sends an ELECTION message to all processes with higher numbers

If no one responds, p wins the election and becomes coordinator

If one of the higher-up’s answers, it takes over.

The process with the highest number will be the coordinator and tell everyone.

**Election in a ring**



**Conflict**

A conflict is a relation between two operations that are concurrent, access the same data and at least one of which is an update

**Consistency**

A consistency model is a contract between the processes and the data store such that if processes obey certain rules, the store promises to work correctly.

Coherence

Coherence concerns only one memory location

A memory system is coherent if for each location, all operations are performed in program order.

If the result of an execution is sequentially consistent, then it is also causally consistent

**Advantage of IPv6**

**Scalability in the number of addresses**

**Scalability of multicast addresses**

**Efficient forwarding**

**Routing**

Routing is necessary in all networks except Local Area Networks (LANs) where Ethernet provides direct communication between all pairs of attached hosts

Routing protocol is implemented in **the network layer** of **each router** to determine the route for the transmission of packets to their destination

Adaptive routing is employed sometimes, the best route is re-evaluated periodically, taking into account the current traffic in the network

**Distance-vector routing algorithm**

Maintain a table representing the direction for each destination

Periodically sending the local routing table to all neighbours if any changes happened

When receiving a new table with new destination or a lower-cost route to an existing destination, update the local table

If a link failure is detected, set infinity and send the local table to neighbours

Each router will know the direction leading to the each distination which has a minimal cost

Pros and cons

Simple

Efficient in small networks

Didn’t consider other factors but just consider the number of hops

Inefficient in large network as loops may occur before the convergence state is reached

**Link-state routing protocol**

Used in IS-IS(IP) AND OSPF protocols

Dijkstra’s Algorithm

**Socket**

A communication end-point to which an application can write data that is to be sent out over the underlying network, and from which incoming data can be read

A pair of processes communicating over a network employ a pair of sockets one for each process

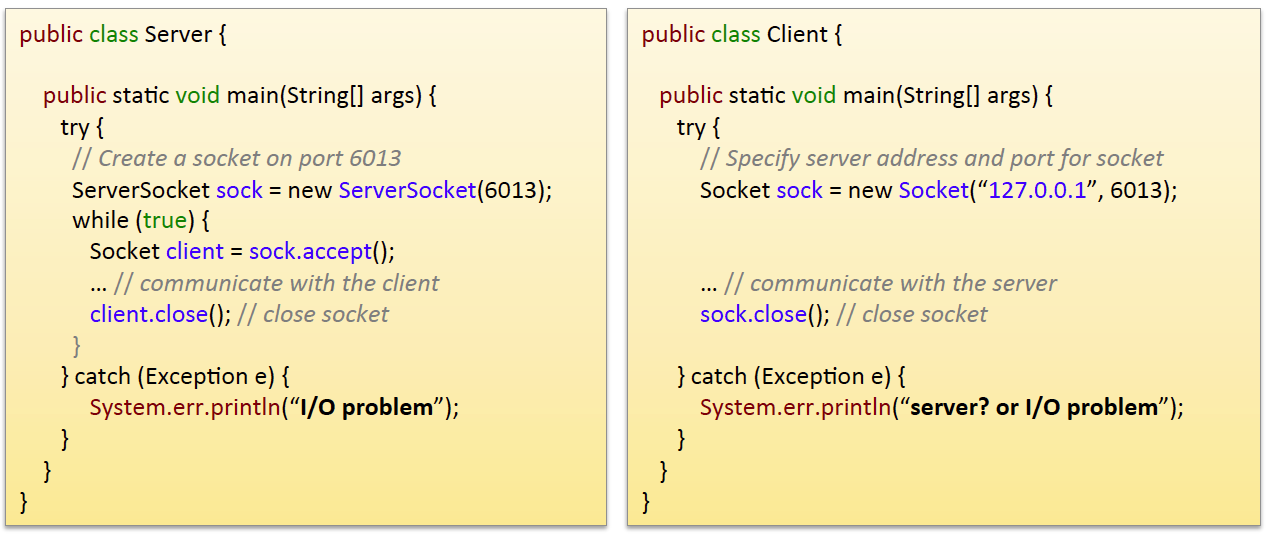
A socket is identified by **an IP address** concatenated **with a port number**

**In client-server model:**

The server waits for incoming client requests by listening to a specified port

Once a request is received, the server accepts a connection from the client socket to complete the connection

Port numbers < 1024 are for specific service protocols HTTP:80, SSH:22, FTP:21,SMTP:25



**MPI VS SOCKETS**

**Sockets:**

Sockets provide only send and receive for communication

Exploits general purpose communication layered protocols (e.g. TCP/IP, UDP)

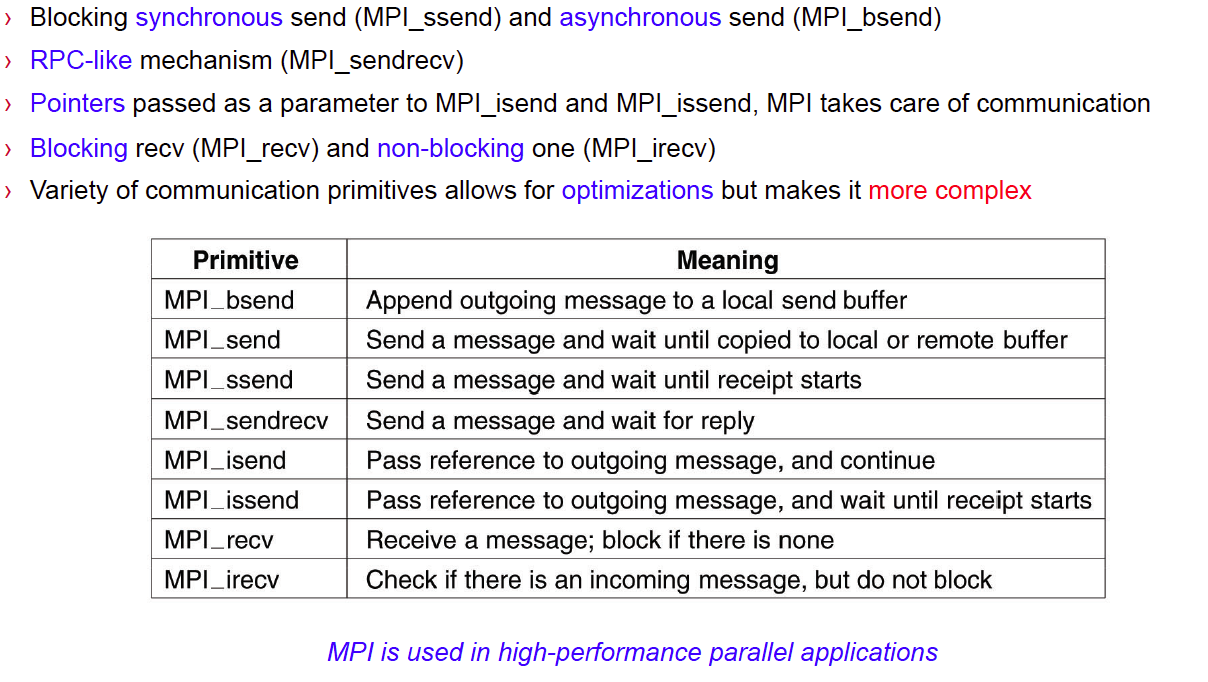
TCP/IP aims at being tolerant to message losses

**MPI(Message-Passing Interface)**

Hardware and platform independent

Direct use of underlying network

Assume no failures, communication involves a group of processors



**Blockchain**

Implements a distributed ledger that is

immutable: transactions can only be appended to the ledger

Pseudonymous: participants are identified by their address

It uses: Gossip-based protocol, P2P network, public key cryptosystem, consensus

**Genesis block** is a special block know initially by all participants

The **pointer** is a representation of a hash of the destination block that the source blocks contains

**Why we need consensus in block chain?**

We want to maintain the chain.

**Solution for Byzantine consensus:**

Limit the number of failure to below n / 3

**Proof of work:**

Motivation: The byzantine node may forge identities to increase the number of faulty nodes to more than or equal to n/3

**Miners:**

Specialised peers, receives a reward for verifying transactions, solve a cryptopuzzle to append a new transaction block to the block chain

Given a block and a threshold, a miner repeatedly:

Select a nonce and applies a pseudo-random function to this block and the selected nonce until it obtains a result lower than the threshold.

The nonce is included in the block, validating that nonce is correct is easy, but solving the puzzle is time-consuming.

Everyone can verify that someone lied about having solved the puzzle i.e. having verified the transaction.

**Resolving a fork**: Pick the deepest branch

**When committed**: given a block chain with parameter k, a block at index i is decided when chain depth reaches i + k

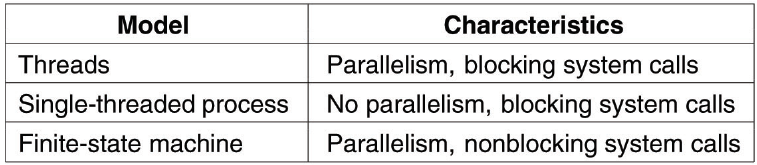
**Ethereum and GHOST**: Pick the heaviest subtree

**Multithreaded Server**

A multithreaded server organized in a dispatcher/worker model

One thread, the dispatcher, reads incoming clients requests

The server chooses an idle worker thread and hands it the request



**Two types of virtual machines**

**Process virtual machine**: A runtime system providing an abstract instruction set used by applications e.g. JRE

**Virtual Machine Monitor**: Provides a system implemented as a layer

**Open standards**

**International Standards Organization**

The Open Systems Interconnection (OSI) reference model names communication levels, and assigns roles to each level

**Internet Engineering Task Force (IETF)**

The Request For Comments (RFC) are a public description of Internet communication protocols (e.g. TCP, UDP, SMTP, ICMP)

**Closed protocols**

Skype: people did reverse engineering to discover the protocol, find security issues, or to implement IM clients

**Mode of communication**

**Connection oriented**

Establish explicitly a connection with a partner before exchanging data

Protocol example: Transmission Control Protocol (TCP)

Application usage: file transfer, web browsing, email

**Connectionless**

No setup in advance is needed

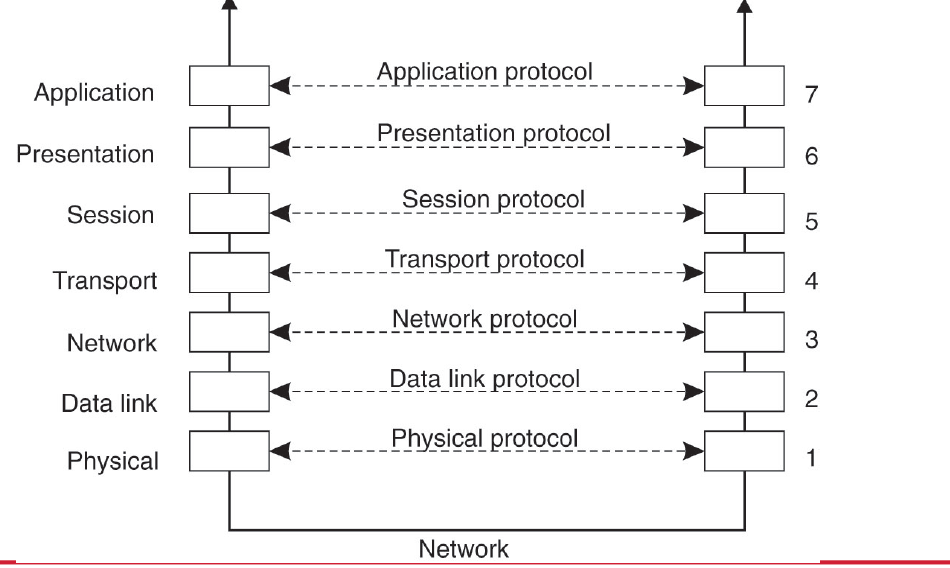
Protocol example: User Datagram Packet (UDP, IP)

Application usage: VolP (Skype, IPTV)

**OSI layers**

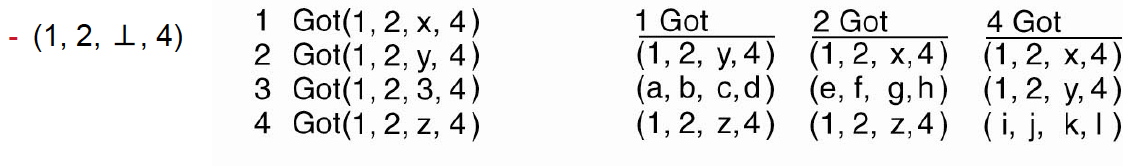
Each layer deals with one specific aspect of the communication

The problem is divided into sub-parts that can be implemented individually



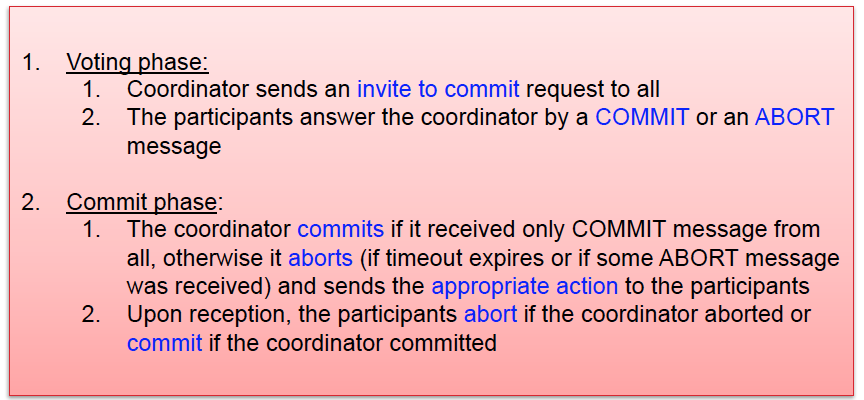
**Byzantine-tolerant synchronous solution**

Unicast, byzantine failures, a max of f < n/3 processes can fail

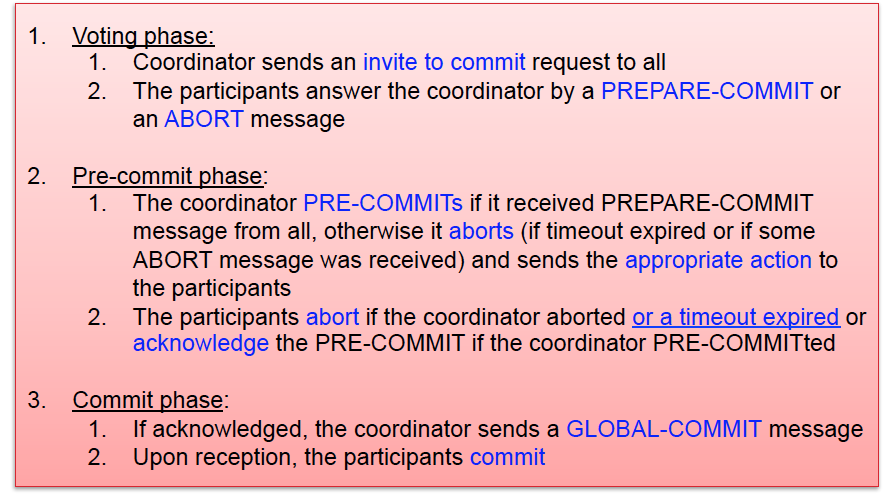


No solution to the consensus problem in an asynchronous system

2PC



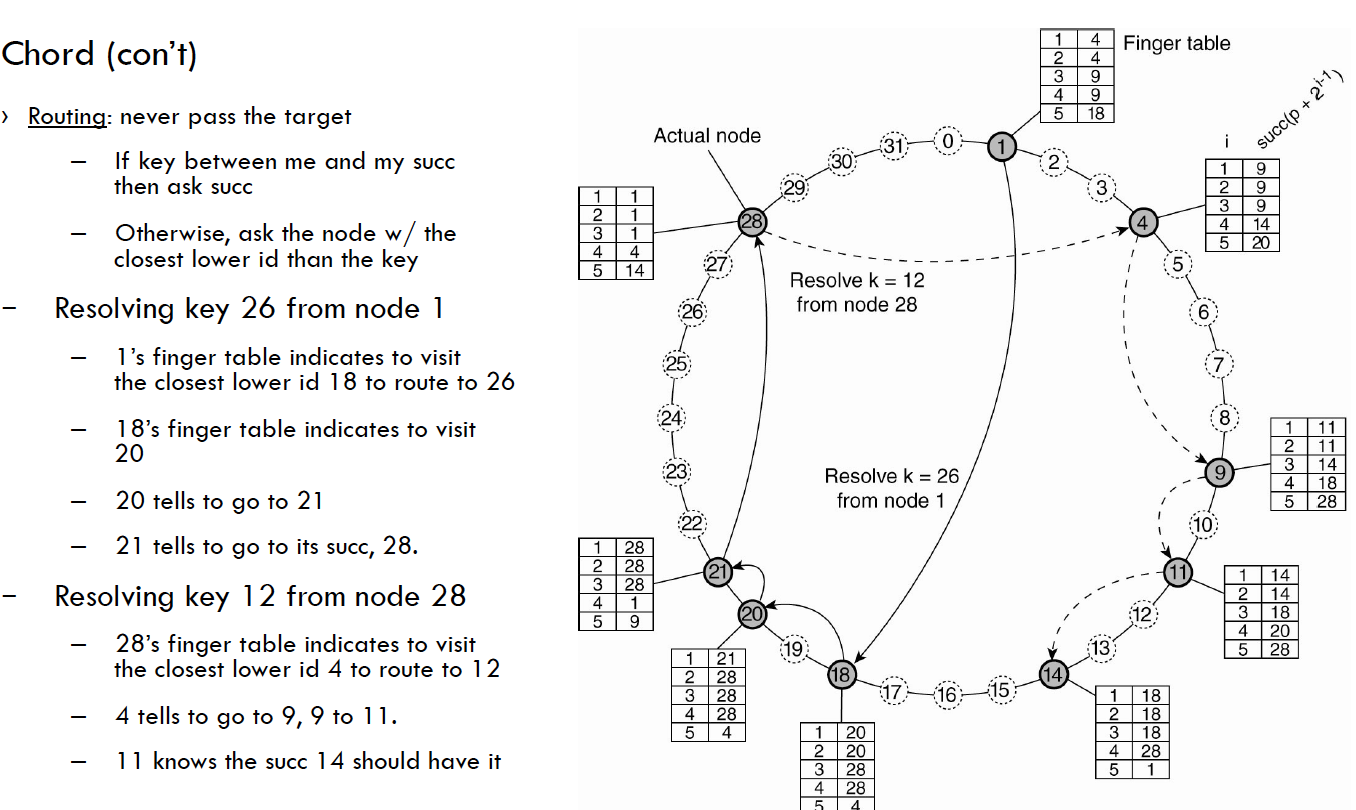
3PC



**Distributed lookup**

Iterative lookup: a node requested to look up a key returns to the requester the address of the next node found

Recurive lookup: a node requested to look up a key forward the lookup



O(logn) lookup time O(logn) information per node

CAN: O(d \* n^(1/d)) lookup time, O(d) space for all

Gossip-based protocols

Anti-Entropy: pi picks pj at random and exchanges updates with pj

Three approaches:

Pi pushes its own update to pj

Pi only pulls in new updates from pj

Pi and pj send updates to each other(push-pull approach)

Two methods for gossip-based protocol

Void activeExec should be an infinite loop sending message periodically

Void passiveExec is triggered upon reception of a new message