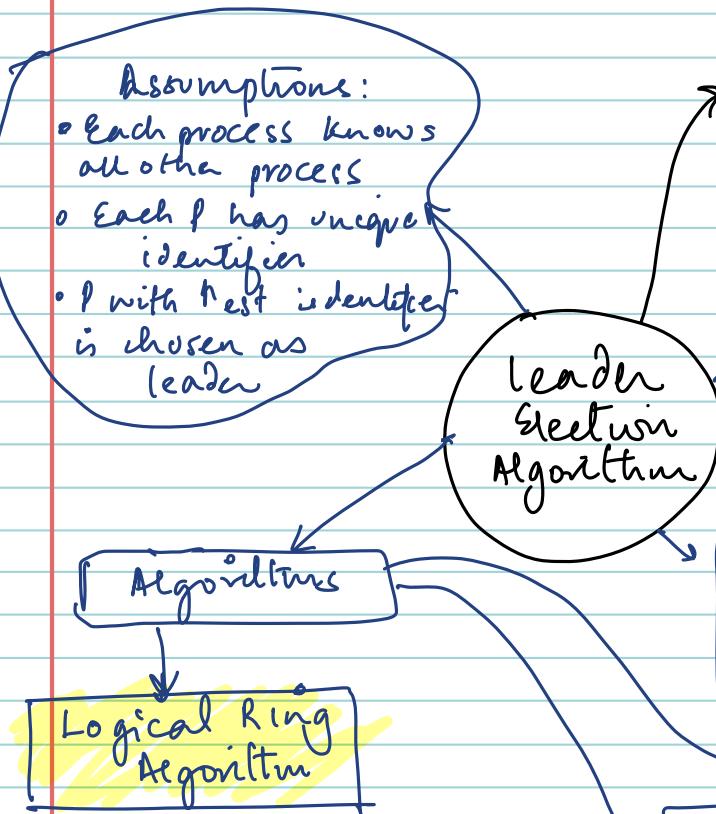


Distributed Systems



definition

process to determine a manager or coordinator among a set of processes on distributed networks

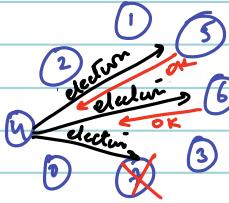
When to elect?

- When system initialises or fails
- process gets no response for some time

Reason to elect leader (Motivation)

- Process Synchronization (who performs what)
- Fault Tolerance (System Recovery)
- Resource Allocation
- Load Balancing (by distributing tasks by leader)

Bully Algorithm



Tree Algorithm

- process arranged in tree structure

Steps: Process P (eg 4) detects failure of leader (eg 7)

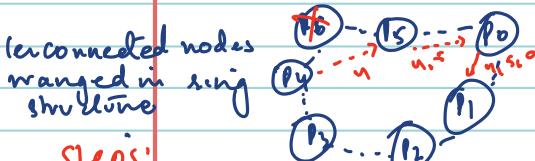
- P_4 sends 'ELECTION' message to all the Process with \uparrow id
 - If no response then P_4 declares itself as leader
 - else, (response from higher ID)
- $P > P_4$ send OK msg & take over election process

Now P who sent OK, sent election msg to \uparrow IDs and this occurs until no \uparrow exists

- Process that receives no OK message is declared new leader
- Sends coordinate msg to all P
- all P recognise the new leader

→ Quicker than Logical Ring ✓

→ High message overhead X



Steps:

- P_6 fails
- P_4 observes failure
- P_4 passes id 4 to 5
- P_5 passes 4, 5 to 6
- P_6 passes 4, 5, 6 to 7
- P_7 passes 4, 5, 6, 7 to 8
- P_8 passes 4, 5, 6, 7, 8 to 9
- P_9 receives message back (4, 5, 6, 7, 8)
- P_9 and knows the ring is traversed
- P_9 picks highest id in the list

Ring msg passing and sends coordinator message to all the process in ring 1 by 1

- ↑ latency, slower election X
- Simple implementation ✓

2

- physical v/s logical
- measure real time
- provides sequence of events in system w/o real clock
- requires periodic sync with time server or GPS to ensure accuracy across DS
- Ideal for real time scenarios; logging timestamp in DB
- Ideal for message passing where event order matters

Time in Distributed Systems

In any system, clock synchronisation is important i.e. (all nodes should have same time), which can be affected by Network delays, varying clock speed, lack of central authority

Problems: Clock sync, Fault tolerance, consistency - Event ordering

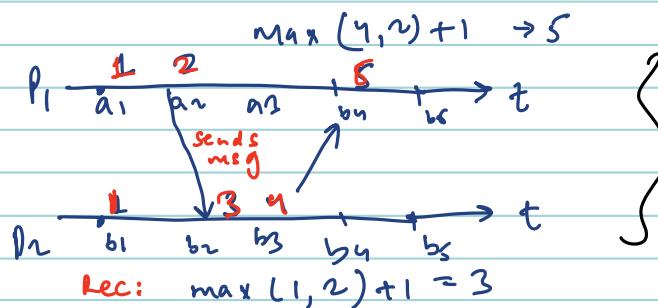
Maintains logical clock.

updates (max(pv; +1) + 1)

Clampart clock

(Capture happens before Relation)

- Each process maintains its local counter (Clampart clock)
- clock updates during internal event, message sending & receiving msg (attaches personal timestamp to msg)

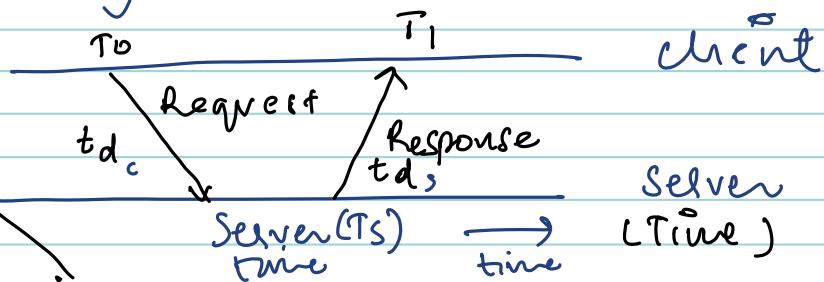


even though P1 & P2 might not have synchronised clock, the logical clock allows them to agree on the order of events.

Cihiha Algorithm

(Synchronising clocks in DS)

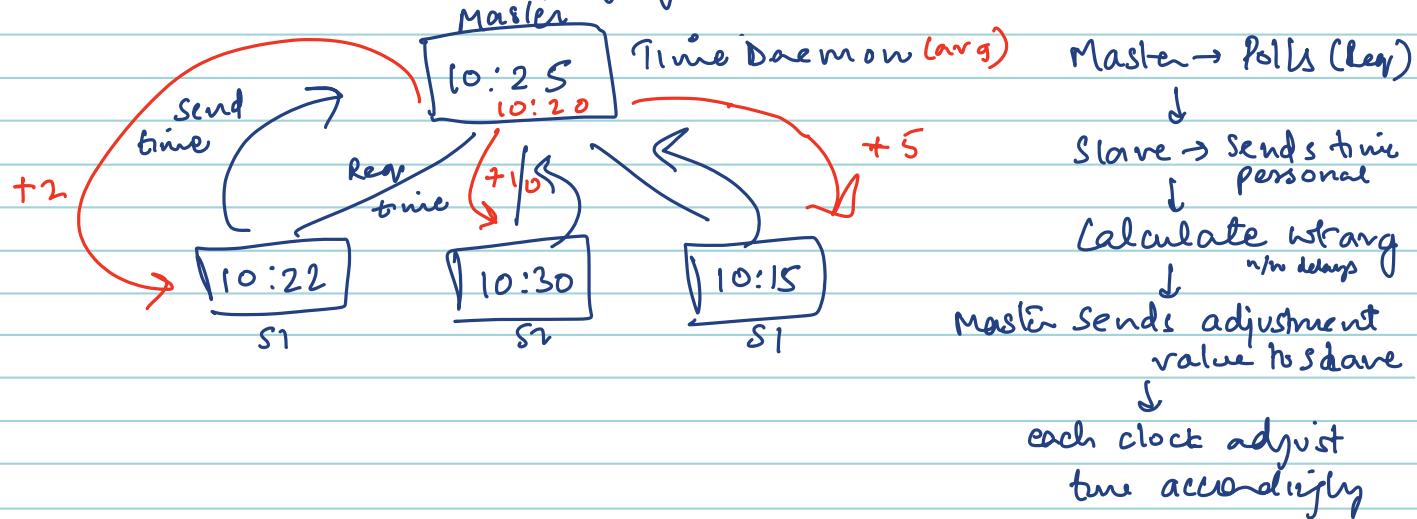
In dist system, clock synchronisation is needed and done using a central time server

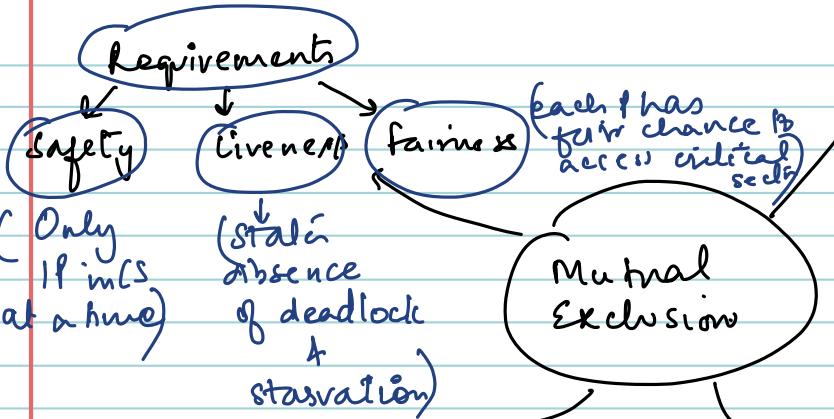


Symm assumption $t_{dc} = t_{ds}$

$$T_{\text{new}} = T_S + \frac{(T_1 - t_{d,c})}{2}$$

Berkeley clock: (Clock synchronisation method in DS to synchronise clocks of all nodes in Network without relying on central time server)





Definition: Concept that ensures that at a time only one process access the critical section of code or shared resource, to avoid race conditions.

Critical section

Shared Resource
file, memory, printer

P access resource in serial way

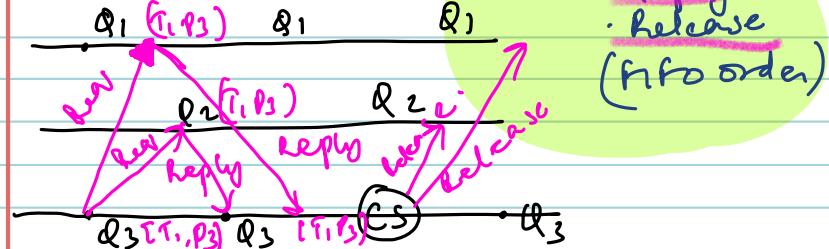
Ricart-Ackermann Algo for Mutual Exclusion

By Glenn Ricart & Ashok Agrawal
extension of Lamport Algo.
follows permission based approach

2 messages → Request
Reply

Lamport Algo for Mutual Exclusion

permission based approach



each process maintains logical clock
each process maintains queue to enter CS

Request → timestamp sent to all P.
increment LC when send
process added to local queue(send)

Reply → Acknowledges with reply (All P)
add req to local queues in FIFO order

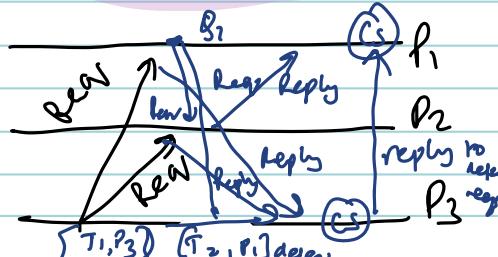
Enter critical section:

- P with least TS enters CS
- After receiving reply from all other P.

Exit CS:

Send release msg to all P
remove req from local min stamps

$3(N-1)$ msg
critical section



Request: Send req to all P to access CS

Reply: Only when 1 or more P is not using CS
+ Not req CS, & has time stamp less than the requested Proc

To use CS: Only when all send reply

To release CS: Send reply to previous def req which requested CS

$2(N-1)$ msg
critical section

a.c. conditions
around
me
for
deadlock

Deadlock Avoiding Condition

1. Mutual Exclusion
2. Hold & wait
3. No preemption
4. Circular wait

COFFMAN conditions

"No progress"

definition: Occurs when set of P each hold some resources & request access to other resources and none of them can proceed.

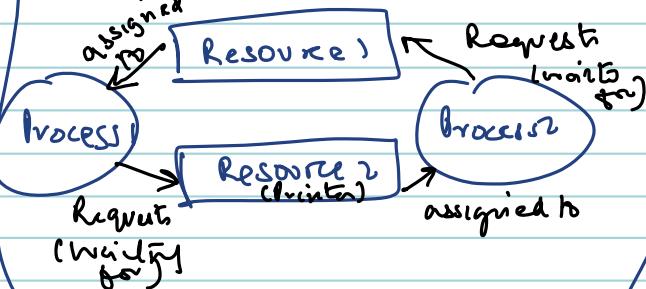
Deadlock
Detection &
Recovery

Involves identifying & resolving situations where multiple processes are waiting indefinitely for resources held by other resulting in deadlock.

Example: • Wait for Graph (WFG)

• Chandy Misra Algo.

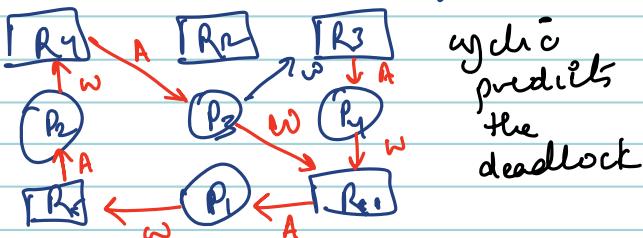
• Resource Allocation Graph (RAG)



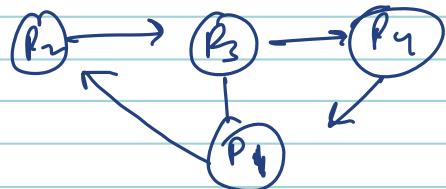
RAG

Resource Allocation Graph (Fredrick)

"Shows state of the system"

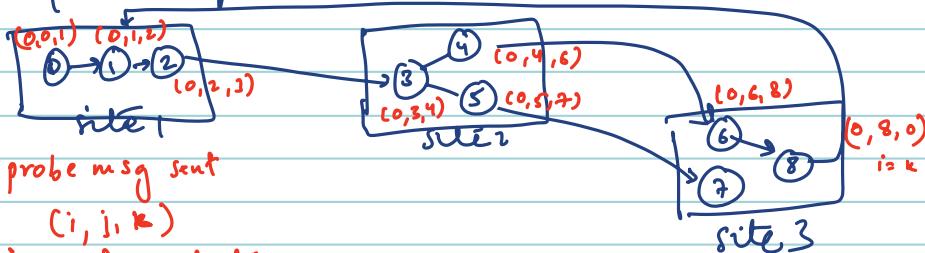


Wait for Graph (Liveness)
Same as RAG, but only processes
not resources
Represents dependencies b/w processes



Chandy Misra Rao Algo: edge chasing probe based PROBE used (for distributed systems).

If process requests for resource and fails or times out, then it sends probe msg to all the processes holding one more of its requested resource



i = sender initial

j = sender

k = receiver

i = k → deadlock,

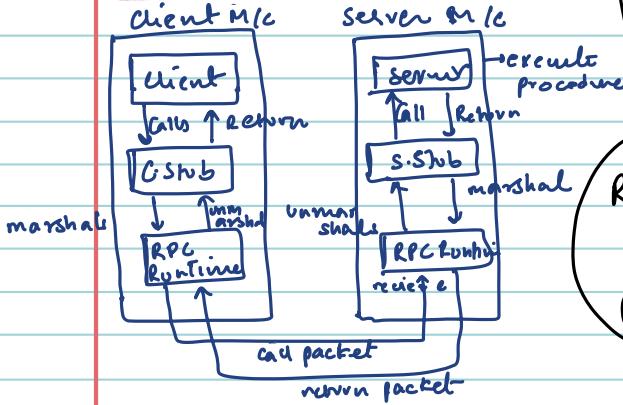
if i = k then deadlock

solution: abort the process to break deadlock

RPC

Remote procedure Call

Principle: protocol enables a program to execute functions on remote server as if it is run on its own.



gRPC

google Remote Procedure Call

Framework for implementing RPC with HTTP/2

Bidirectional Streaming

features

→ high performance
uses HTTP/2
[uses Protobuf]

Cross Language Support

Strongly Typed

[define Services & messages types using Protobuf]
Can be automatically converted to multiple languages ensuring strong typing.

RPC Runtime: manages transmission of msgs b/w n/w across Client & Server.

RPC v/s REST

REST - Representational state Transfer

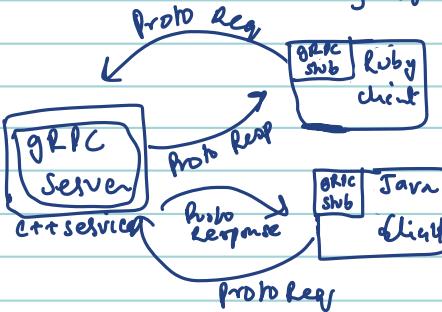
RPC

- Procedure based approach
- Stateful for long lived connections
- over a wide n/w
HTTP, TCP, UDP

REST

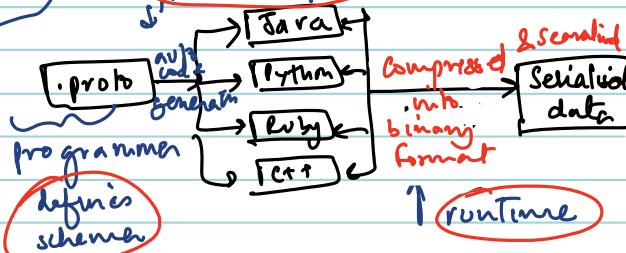
- Resource based Approach using HTTP methods
- stateless
each request has all the necessary info & is not dependent
- concurrent process
- Defined specifically

stob → serialises the request using protobuf.
sends to gRPC servr



- HTTP/2 instead of HTTP/1
- Protobuf instead of XML or JSON

protobuffers: Use Binary data representation enabling data packets to transport faster than other formats like JSON/XML
proto compiler



HTTP 2 : Allows real time communication & improved n/w efficiency

bidirectional streaming
(msg 2 way)
no wait
real time

Multiplexing
(single TCP connection)
instead of opening many connections for each msg.

gRPC Services: Types

- Unary : C1 : S1 (authentication)
- Server Streaming : C1 : Sn (notifications)
- Client Streaming : Cn : S1 (video upload)
- Bidir Streaming : Cn : Sn (chats)

service abc {

 rpc ServerStreaming (reqMsg) returns (stream Response) {

example: code:

```
syntax = "proto3";
message RequestMessage {
    string name = 1;
    int32 id = 2;
}
message ResponseMessage {
    string marks = 3;
}
service State {
    rpc UnaryCall (RequestMessage) returns (ResponseMessage);
}
```

RPC

① Use various protocols (HTTP, TCP)
etc

② → supports unary calls
streaming support

③ For serialisation can use formats
such as - XML, JSON

④ Manual code generation

gRPC

① USES HTTP/2 for communication

② support 4 types of services
unary
server streaming
client streaming
bidirectional

③ For serialisation,
uses proto buf

④ Automatic code gen