concurrency control property to avoid race conditions

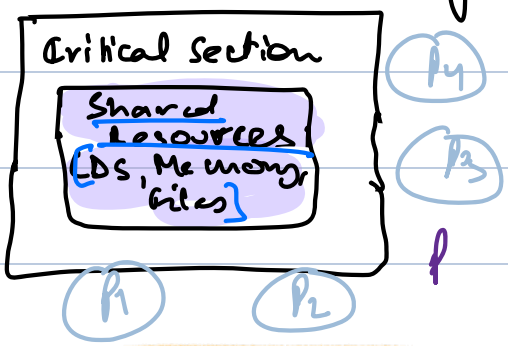
Critical Section
↳ section accessible to only one process at a time

Mutual Exclusion

ME → makes sure process access shared resources or data in serialized way

Single System
↳ shared Memory
easy

Distributed System
↳ no shared Memory
↳ no common physical clock



Requirements

Safety
- Only one process in Critical Section at a time

Liveness

No Deadlock
No process should block the progress of others indefinitely

No Starvation
Every process that requests entry into CS should eventually allowed to enter

fairness
- Every process should have fair chance to access Critical Section

Lamport's Algorithm for Mutual Exclusion

↳ permission based non token algorithm using Timestamp.

Key Components

logical clock

each P maintains logical clock increments with every event S & R.

Request Queue

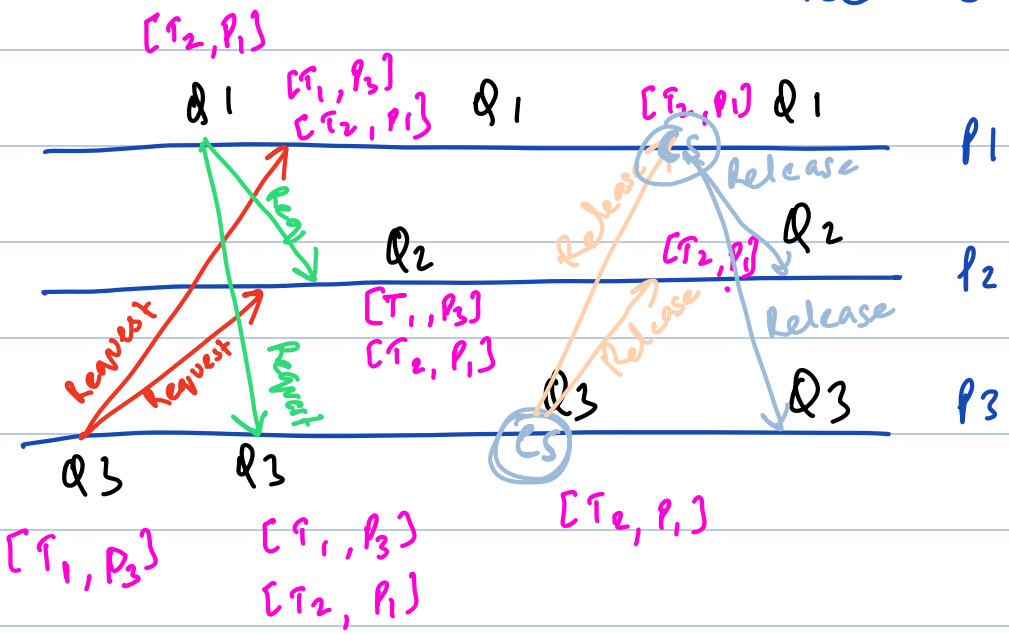
each P keeps queue of requests to enter CS.

3(N-1)
message per critical section execution invocation (N-1) each for 3 (Req, Rep, Rel.)

Algorithm :

↳ 3 messages → Request
Reply
Release

[FIFO] order for Request process



Steps: Request entry → send REQUEST with

Timestamp to all processes

→ Increment LC before sending Request

→ Process added to local Q.

Receive Request → acknowledges with REPLY

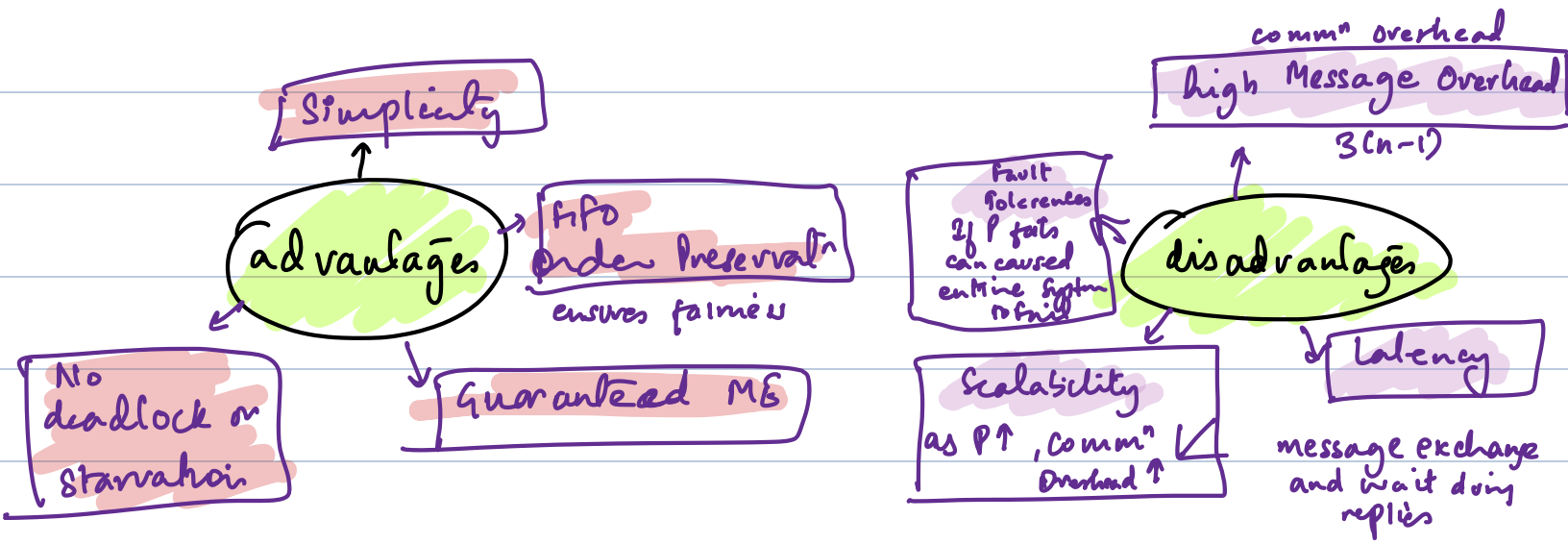
→ adds requests to local Queue

Enter Critical section only when → Process with least Timestamp enters CS

→ It receives reply from all processes

Exit Critical section → process removes its requests from Queue and sends RELEASE message to all processes

→ The requests is removed from local queue after getting RELEASE messa


Simplicity
advantages
FIFO order Preservation
ensures fairness
No deadlock or starvation
Guaranteed ME
comm^n overhead
High Message Overhead
3(n-1)
Fault tolerances
If P fails can caused entire system to fail
disadvantages
Scalability
as P↑, comm^n Overhead↑
Latency
message exchange and wait during replies