GFS

Last 2 classes
↓
2 papers
Paper presentation
(attendance in at least 6 mandatory for all)
2 papers/group
mostly 2010+, but some before
Some old

Classical Filesystem:-

1. Basic unix filesystem
   — inodes etc.
2. Basic + some improvement
   (Break disks into block groups & allocate + data block in same block)
   — Performance
3. Journaling — Ext4, NTFS
   — Crash recovery
4. "A log structured file system"

Connection w/c

Flynn's Classification
SISD, SIMD, MISD, MIMD

"Grand Challenge Problem"
www.top500.org/

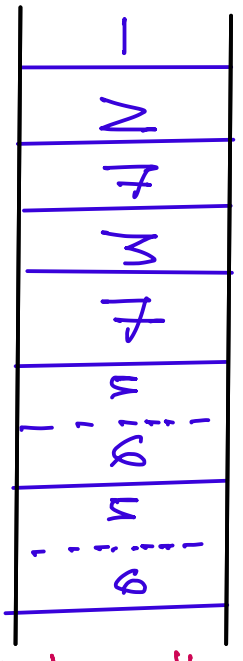
Cluster:-

2003:- Thousands of inexpensive w/c
— failure is the norm

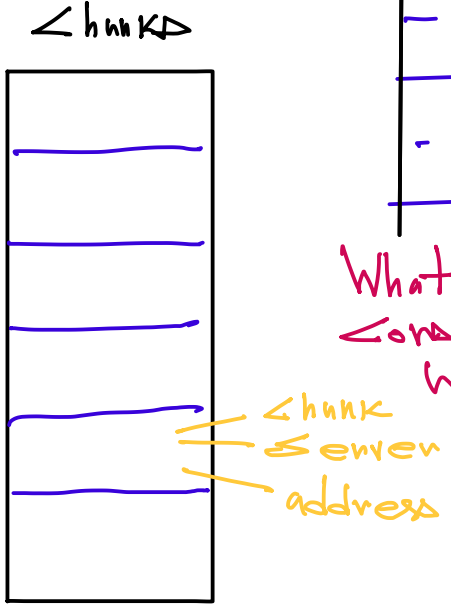
Sequential Reads → Caching is useless

Q) What if master fails?

/usr/test 7th chunk



What is the consistency we want?



Global ID

Q) What happens on a read?

Forward while Receiving ← Client

Location of Chunk Replicas :- Not stored in disk

Assumption :- Idempotent operations (can be done any number times)

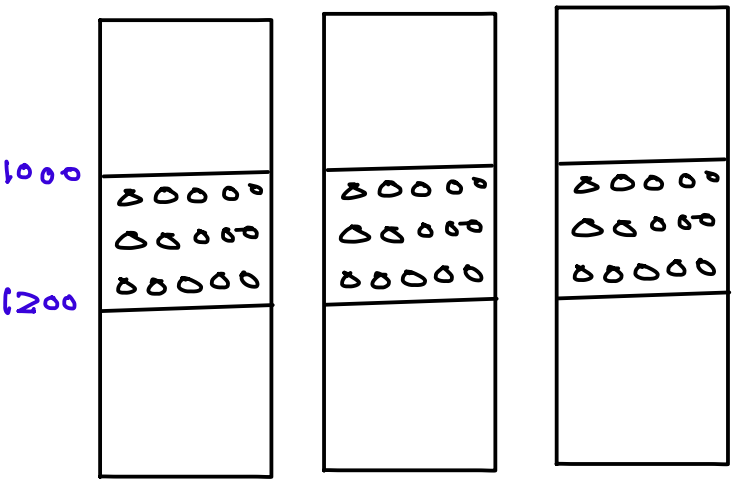
$x = 5$ ← Idempotent

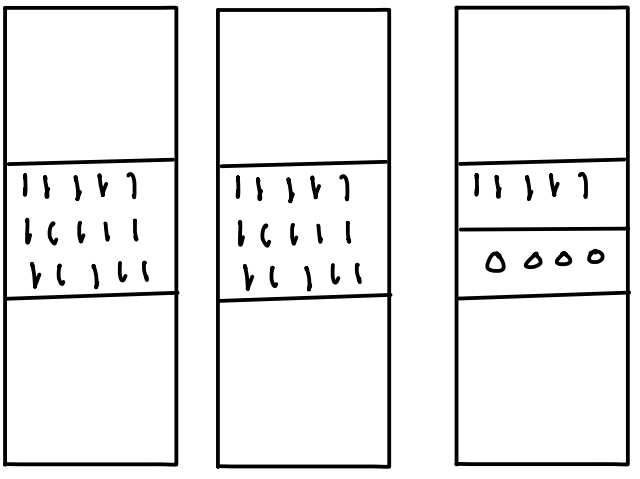
$x += 1$ ← Not idempotent

Region A from byte 1000 to 1200

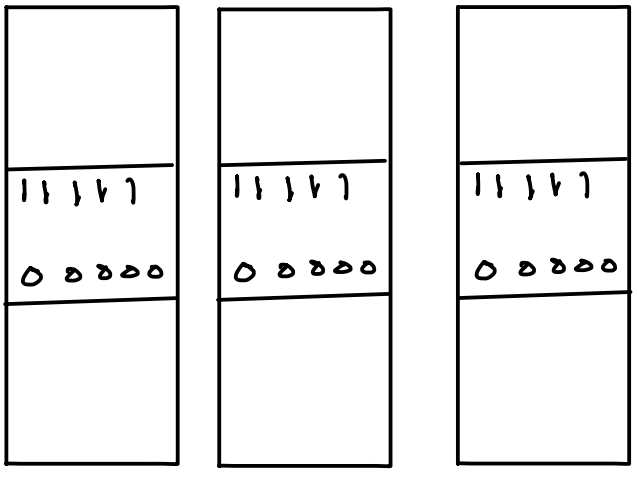
Suppose in all 3 replicas
A bytes are all 0's

Now suppose Client writes all 1's





Inconsistent
Undefined



Consistent
Undefined

Append:- Can be appended anywhere

Write:- At the end of file

			Retry
	1600-2000	All 1's	2000-2400
	1600-2000	"	"
	1600-1800	"	
	1800-2000	All 1's	"

Q) How to Know Duplicate and Undefined?

⇒ Add some ID ⇓ Checksum

Stale Chunk:- Writes from previous master

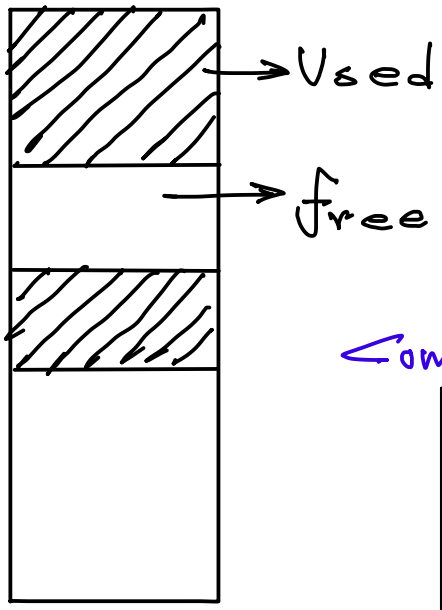
/d1/d2/leaf ⇒ Read Lock on /d1 and /d1/d2 Write Lock on /d1/d2/leaf

ext2 — Block groups — good for performance, bad for crash recovery

ext3 — Journaling — good for crash recovery, bad for performance

ext4 — more support for large files, extents

Traditional (Then)



How to keep track of free spaces?

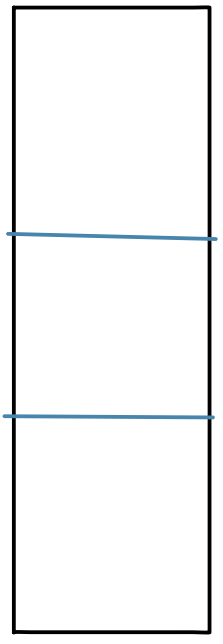
Threading

or

Compact periodically

LFS!

Combines the two.



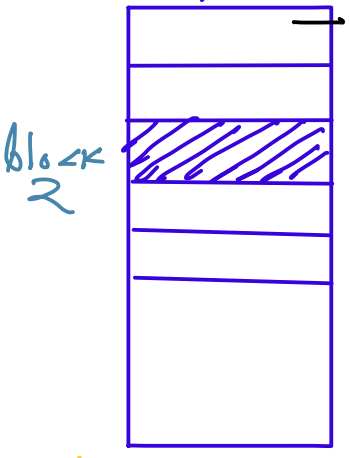
Segments

At any point of time a segment can be fully free, fully occupied, partially free.

Q) How does the GC know what is stale in a segment?

Segment segment summary info

- for all data blocks in the segment, contains its byte offset in the inode where the block address is stored.
- Using this identify, all live data blocks in the segment



for data block 2

Block 2 — inode 3

320

Segment usage info:-

- Amount of live data in the seg
- last mod. time of the seg

Using this, can have your own policy to date at which segments are to be GC'ed. (f. ex say less than 10% live data) & which segments to put together (for ex say 2 segments each with <50% occupancy & not modified last 30 minutes)

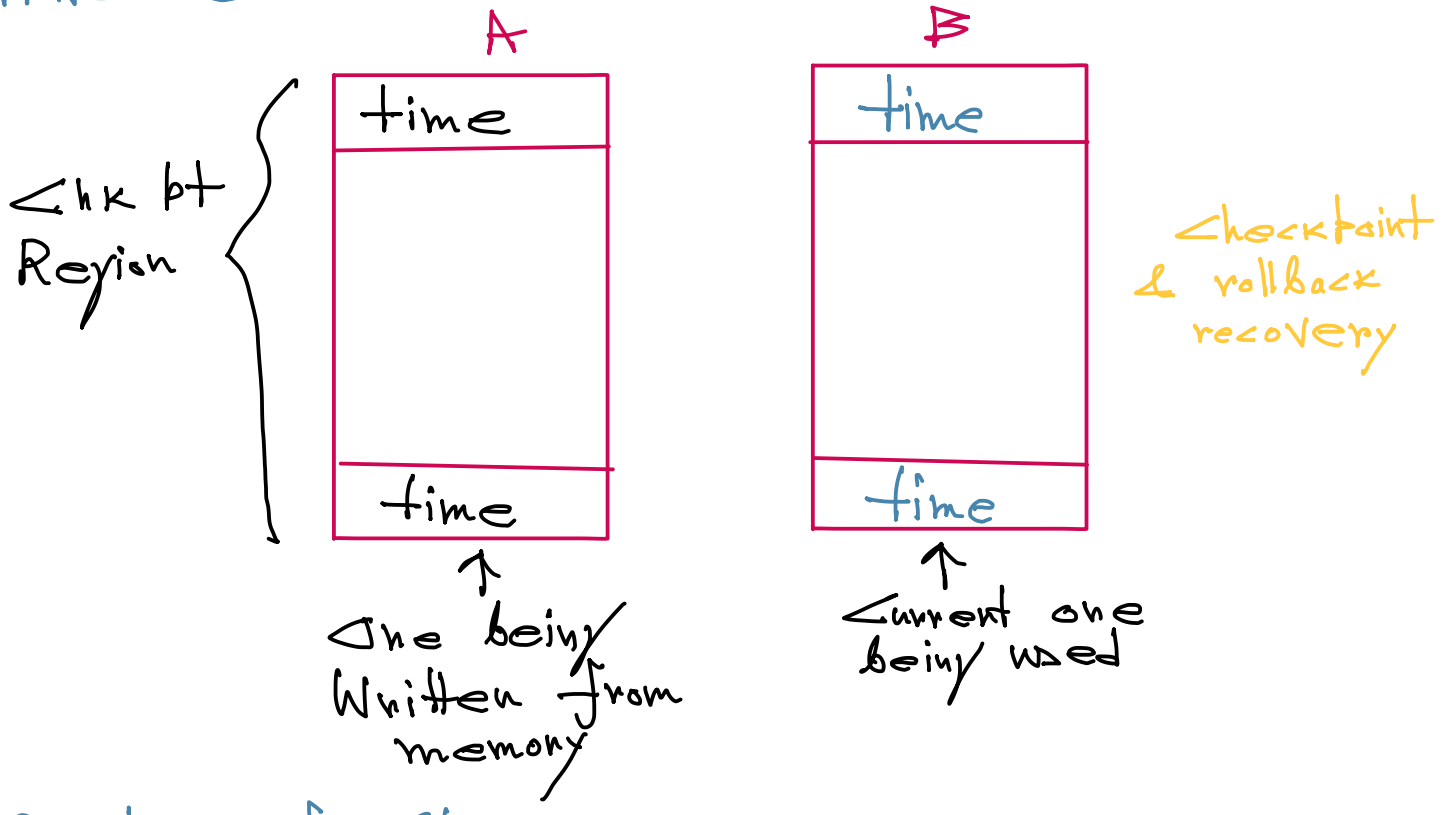
(live data, inode, imap)

Crash Handling:

1. Impossible to not loose any data on crash
2. Ensure that what is being pointed to by a pointer is written first before writing the pointer

Can loss data, but will maintain consistency

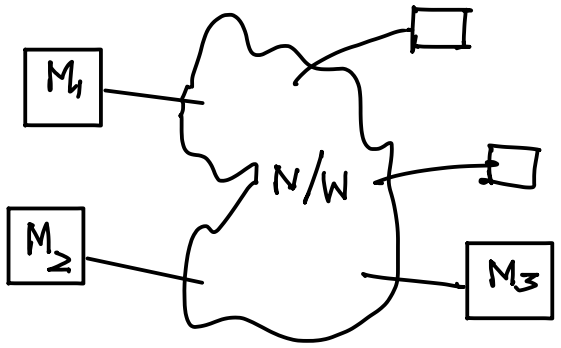
Have 2 copies of the checkpoint on disk



Functions of OS:

1. Scheduling
   - Multiple Processes
     Who runs where & when
2. IPC/Synchronization
3. Memory Management
   - Virtual | Physical
   - Memory allocation/access
4. File System
5. Security

Distributed Systems



Ideally, for a user, it should look like a single system

"Ideal" is not always achived
— Performance.

Scheduling in Dist OS:-

1. Where all can the user submit task?
   — affects other decision, like load balancing, recovery etc
2. Who will do the scheduling?
3. Which task will run where & when?

Memory:-

1. Can we give the user the same view — a single contious address space? — Distributed Shared Memory.
   — Can implement, but costly

Distributed OS

Evaluate any design on:-

(i) Correctness
(ii) Scalability
(iii) Fault behaviour/Tolerance
(iv) Types of user environment/task pattern it is good/bad for

Scheduling:-

(i) Where are tasks submitted?
- One specific m/c Pros/Cons?
- Any m/c
- A subset of m/c

(ii) Who makes the scheduling decision?
- One fixed "master" node
- Any Node — A subset of nodes

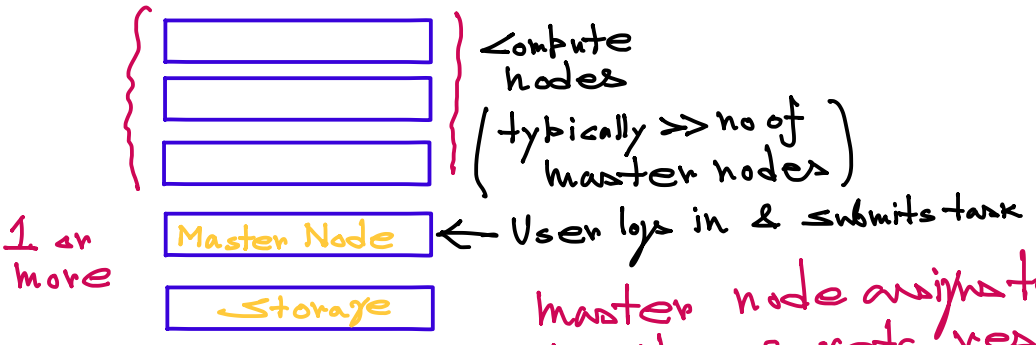
(iii) When should a task run?

Based On:- → Resource Monitoring

- Round-Robin
- based on "local"
  - What is local?
  - How is load info collected?
    - Who collects?
    - How often?
- Round-Robin
- N/W other things

Run 3 copies of the task (SLA) → Fault Tolerance

Ex:- Cluster

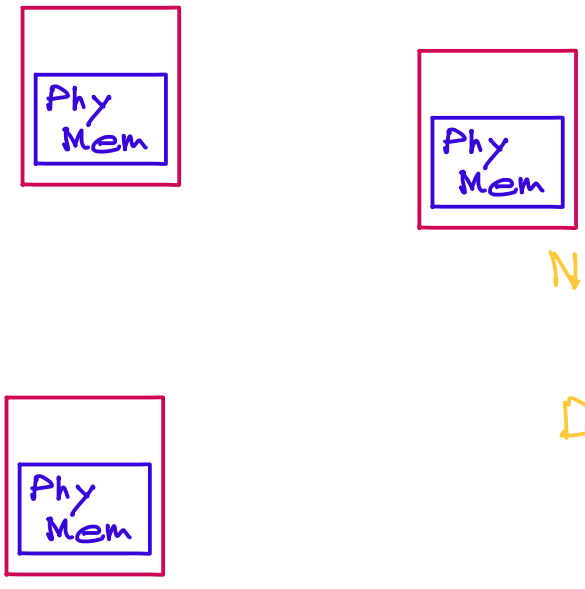


master node assigns to compute nodes & monitors & gets result back to user

Many diff open source & commercial "Cluster middleware/s/w"

Memory:- Distributed Shared Memory

Put a s/w on each m/c so that a user on a m/c can use the memory on any of the m/c



Normal OS:-
Virtual addr:- Phy addr on same m/c

DSM:-
Virtual addr:-
<m/c id> <Phys mem in that m/c>

Process Synchronization:-

Needs:- (i) Mutual Exclusion
(ii) Ordering the execution of process

How to do (ii) in DOS when 2 or more process are running in diff m/c
— Use msg passing

How to do (i)?
(i) Permission based Protocols
(i) Who to ask for permission?
— All
— Subset (with some constraint)

Ask permission for all:-

Process i wants to access a shared resource
i send a REQ to all other process
i accesses the resource once it gets a REPLY from all
i sends a RELEASE to all once finished

Any other Process:-
On getting a REQ, send a REPLY

Pr:- No mutual excl (So, safety is not satisfied)

Add mutual exclusion:-

Allow a process to send only one reply at a time (until RELEASE is received)

Pr:- Deadlock (So, liveness is not satisfied)

Solve Ver 1:-

Add a priority (unique) to each process
Do not reply to lower priority process if REPLY is already sent to a higher priority process

Suppose There is a global clock

⇒ Let the priority be the request assigned

⇒ Let all nodes reply to all request, & put in the request recvd in a queue in ascending order of time

A node enters CS, if it gets all replies & its request is on the top (first) of its own queue

On RELEASE, remove the entry from the queue.

Works, but require FIFO

⇒ Don't reply when you have the lock (Improvement)

Distr Mutex

Permission based algo:-

(i) Permission from all:-

(a) Naive one that doesn't work → $\Sigma(n-1)$

(b) Naive one + request queue based on time (assumption) → $\Sigma(n-1)$

b.1 → required FIFO (send REPLY immediately)

b.2 → doesn't require FIFO (Hold reply if earlier req known)

Metrics:-

(i) # of msgs/CS entry

(ii) Synchronization Delay - Time gap after 1 process has finished CS & the 2nd one has entered CS.

(ii) Quorum based protocols / Voting protocols

Each process asks for a subset of other processes

R_i = Subset of process that process i will ask permission from

Safety (Mutex) Requirement:- $|R_i \cap R_j| \neq \phi$ if $i \neq j$

Desirable:- $|R_i| \approx |R_j|$ for all $i \neq j$

Naive protocol:-

Can get into deadlock

Solution:- Simple: Based on priority
Ask others to relinquish vote

5x(Request Size −1)

On getting a request from i:-

(1) If vote not given to anyone, give to requestor

(2) If given to j already

(a) Priority(j) > Priority(i)

Send FAILED to i

What does i do on receiving it? Queue up req of i

(b) Priority(j) < Priority(i)

Send Enquire to it if it has succeded in getting all votes

What does j do on receiving it?

⟹ (i) You can relinquish

⟹ (ii) You can hold till all votes or atleast 1 FAIL

Issues to think about:-

(1) How do we build the quorum satisfying constraint?

(a) Choose one process to all (extreme; central auditor)

(b) Choose all processes for all (other extreme)

(c) Others (actually done)

(2) Should the quorum depend on operation?

- Read (try making less effort) &
Write (more effort)

Read Quorum
Write Quorum } Separate, as reads are usually much more than others

Ex:- (Extreme) Read - don't ask anyone
(ask only yourself)

Write - Ask all

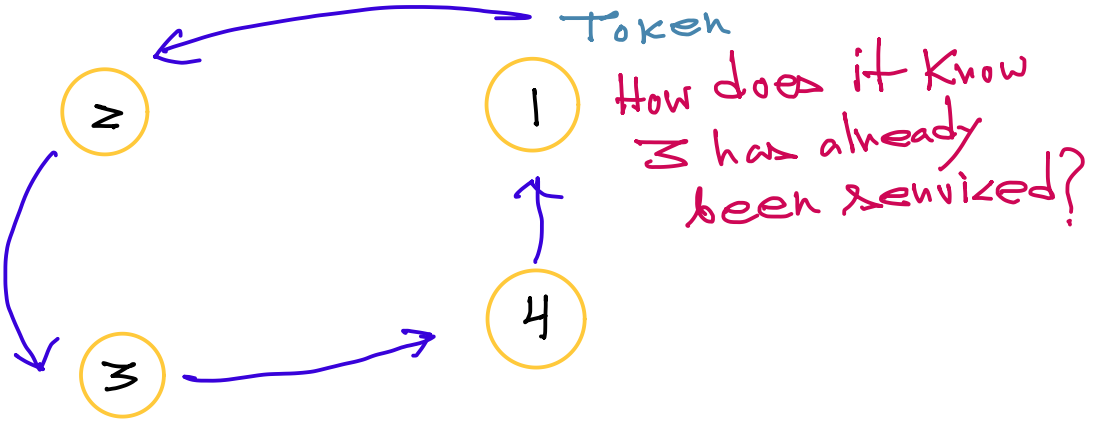
(iii) - Should everyone get 1 vote (equal vote)?
- What about heterogeneous system?

Token based algorithm:-

- One "token" in the system
- Anyone who has the token can enter CS
- Mutex is trivial

Issues:-

- (i) If a process wants to enter CS, how does it know who has the token?
  - does it need to know?? (depends)
  - Req to one process which always knows
  - Search for it
- (ii) What happens to the token after a process exit its CS?
  - just holds it
  - Send it to next process with request (how to know?)
  - Circulate in a specific pattern



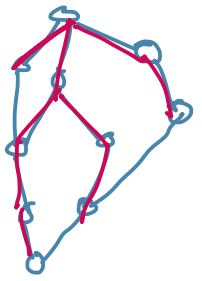
Token based Mutex

Approach 1

1. Token circulates continuously among all nodes
2. If a node needs to use it, it does, else forward it to next node
3. If a node wants to enter CS & does not have the token it waits

How to circulate?

① Build a ring
② " a sp tree
③ Is the algo adaptive (to load)?

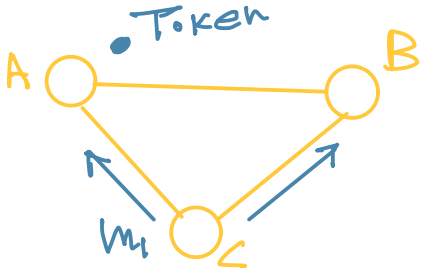


Good at high load

Approach 2 :-

1. Search for the token (seen a bit earlier)

Pr.



C reqs A, B
m is delayed
B req A, C
A give token to B
B sends it to C after use
B req again
Token given back to B after use by C.

A req,
Token given to A after use by B

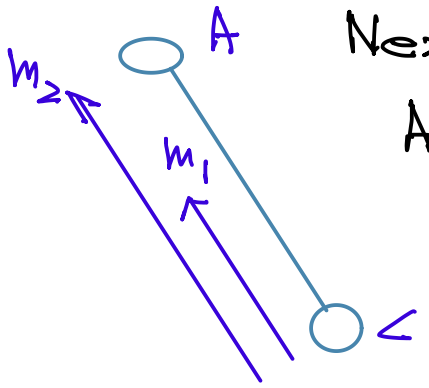
Now m_1 reaches A
⟹ Absolete request

1. How to detect absolete req?
2. Broadcasting is inefficent?

Q) How to solve 1?

(i) How to detect a req is absolete?

One approach – put id with req
– a node can only have one request pending at a time.



Next req of C, m_2, reaches A
A store the req id m_1 reaches
now. if $m_1 < m_2$, ignore

Ok, but what if no m_2?
- Someone would tell A that m_1 is served already
- So who always knows which req are solved?

When a token reaches a node, it tells the node (A) so but in token.

Q) If A does not have token & receives a req, can it just ignore it?

No!!

1. A has req for token from B
2. A gets m_1, drops it
3. A gets token from B, use
4. No one else (B) wants to enter CS
5. So B will drop C's req

Sol:- Remember m_1 (at A)

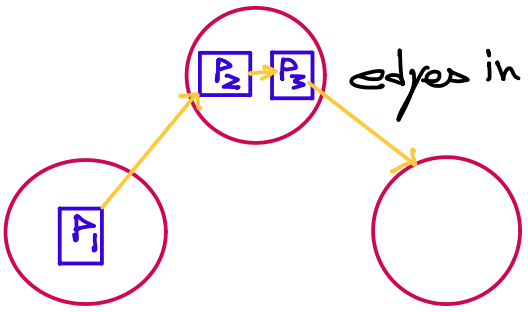
Pr:- What if (in another scenario) m_1 is an obsolete req, and all other req after it to A are even more delayed?

Data Structure??

Deadlock Detection:-

Sequential:- Detect cycle in WFG

Distributed:- Same.
Only process. Cycle has processes spread over multiple nodes.

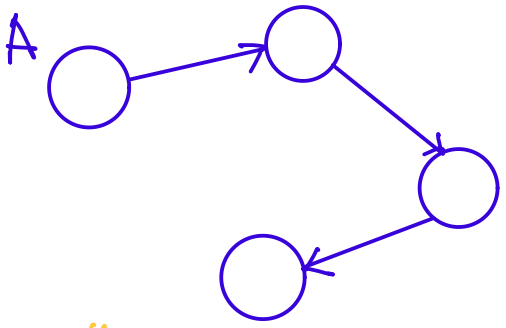


edges in WFC

Sol:- (Say A wants to detect)

Send a msg to nodes A is waiting for. Nodes forward & do the same.

If A gets back its msg. There is a deadlock cycle it with



What if a req is granted just after forwarding the msg over that link

"Phantom Deadlock"