Distributed Mutual Exclusion Algorithms

Course: Distributed Computing

Faculty: Dr. Rajendra Prasath

Spring 2019

About this topic

This course covers various concepts in **Mutual Exclusion in Distributed Systems.** We will also focus on different types of distributed mutual exclusion algorithms in distributed contexts and their analysis

RECAP

What did you learn so far?

- Challenges in Message Passing systems
- Distributed Sorting
- Space-Time Diagram
- Partial Ordering / Causal Ordering
- Concurrent Events
- Local Clocks and Vector Clocks
- Distributed Snapshots
- Termination Detection
- Topology Abstraction and Overlays
- Leader Election Problem in Rings
- Message Ordering / Group Communications

Recent Topic …

- Communication Models
- Design Issues
  - Process Failures
- Message Ordering
  - Good / Bad ordering
  - Various Types of Ordering of messages
- Group Communication
  - Causal ordering based approach

→ Many more to come up … stay tuned in !!

Message Ordering (recap)

- How to order messages?
  - Send vs Delivery

  - Global Time Ordering
  - Total Ordering
  - Causal Ordering
  - Sync Ordering
  - FIFO Ordering
  - Unordered multicast

- Good / Bad Ordering

Topics to focus on …

- ➔ Leader Election in Distributed Systems
- ➔ Topology Abstraction and Overlays
- ➔ Message Ordering
- ➔ Group Communication

- ➔ **Distributed Mutual Exclusion**

- ➔ Deadlock Detection
- ➔ Check pointing and rollback recovery

For Mid Semester 2

Mutual Exclusion in Distributed Systems

Let us explore mutex algorithms proposed for various interconnection networks

Why do we need MutEx?

- Mutual Exclusion
  - Operating systems: Semaphores
    - In a single machine, you could use semaphores to implement mutual exclusion
    - How to implement semaphores?
      - Inhibit interrupts
      - Use clever instructions (e.g. test-and-set)
    - On a multiprocessor shared memory machine, only the latter works

Characteristics

- Processes communicate only through messages – no shared memory or no global clocks
- Processes must expect unpredictable message delays
- Processes coordinate access to shared resources (printer, file, etc.) that should only be used in a mutually exclusive manner.

Race Conditions

- Consider Online systems – For example, Airline reservation systems maintain records of available seats
- Suppose two people buy the same seat, because each checks and finds the seat available, then each buys the seat
- Overlapped accesses generate different results than serial accesses – race condition

Distributed Mutual Exclusion

- Needs
  - Only one process should be in critical section at any point of time
  - What about resources?

Distributed Mutual Exclusion

- No Deadlocks – no set of sites should be permanently blocked, waiting for messages from other sites in that set
- No starvation – no site should have to wait indefinitely to enter its critical section, while other sites are executing the CS more than once
- Fairness - requests honored in the order they are made. This means processes have to be able to agree on the order of events. (Fairness prevents starvation.)
- Fault Tolerance – the algorithm is able to survive a failure at one or more sites

Distributed MutEx – An overview

Token-based solution: Processes share a special message known as a token

- Token holder has right to access shared resource
- Wait for/ask for (depending on algorithm) token; enter Critical Section (CS) when it is obtained, pass to another process on exit or hold until requested (depending on algorithm)
- If a process receives the token and doesn't need it, just pass it on

Distributed MutEx – A Few Issues

- Who can access the resource?
- When does a process to be privileged to access the resource?
- How long does a process access the resource? Any finite duration?
- How long can a process wait to be privileged?
- Computation complexity of the solution

Types of Distributed MutEx

- Token-based distributed mutual exclusion algorithms
  - Suzuki – Kasami's Algorithm
- Non-token based distributed mutual exclusion algorithms
  - Lamport's Algorithm
  - Ricart-Agartala's Algorithm

Token Based Methods

Advantages:

- Starvation can be avoided by efficient organization of the processes
- Deadlock is also avoidable

Disadvantage: Token Loss

- Must initiate a cooperative procedure to recreate the token
- Must ensure that only one token is created!

Non-Token Based Methods

- **Permission-based solutions:** a process that wishes to access a shared resource must first get permission from one or more other processes.

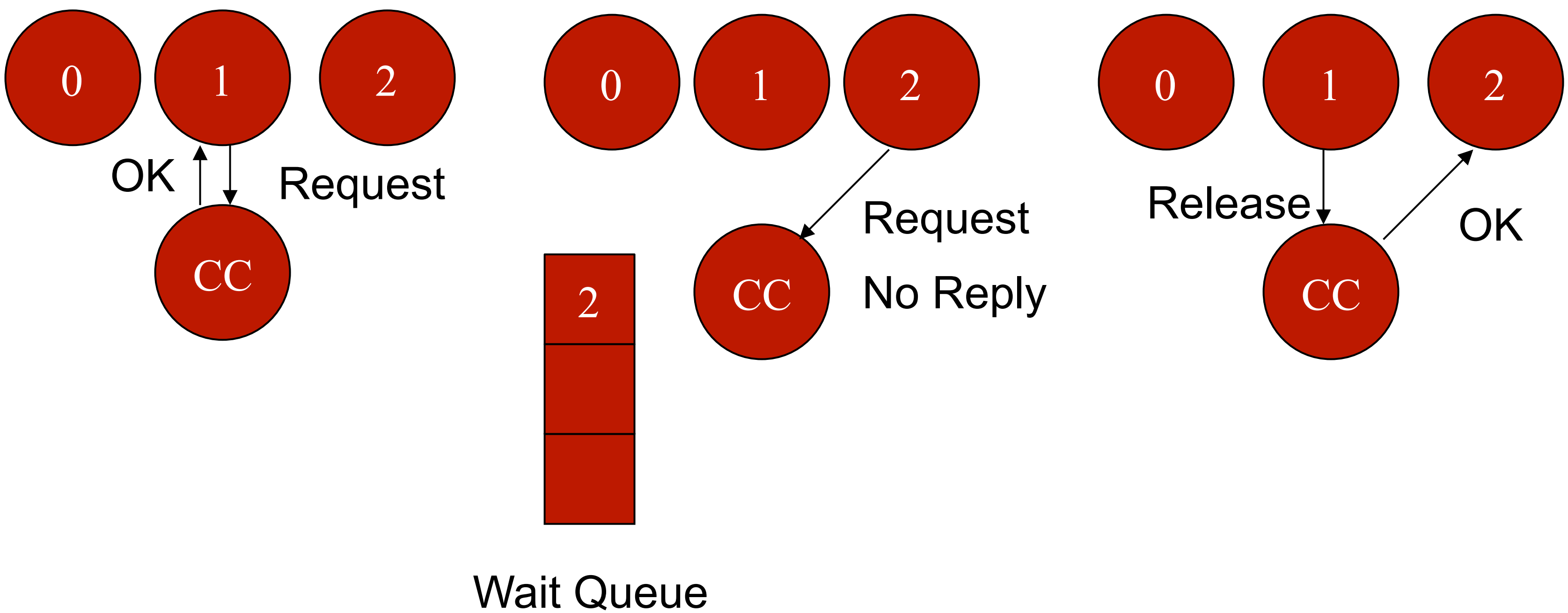
- Avoids the problems of token-based solutions, but is more complicated to implement

Basic Algorithms

- Centralized
- Decentralized
- Distributed
  - Distributed with "voting" – for increased fault tolerance
- Token Ring

Centralized Mutual Exclusion

- Central coordinator manages the FIFO queue of requests to guarantee "no starvation"



Performance Analysis

- Guarantees mutual exclusion
- No starvation: Only if requests served in order
- No deadlock
- Fault tolerant?
  - Single point of failure
  - Blocking requests mean client processes have difficulty distinguishing crashed coordinator from long wait
  - Bottlenecks
- The solution is simple and ease

Centralized Control of MutEx

- A central coordinator (master or leader)
- Is elected (which algorithm?)
- Grants permission to enter CS & keeps a queue of requests to enter the CS.
- Ensures only one process at a time can access the CS
- Has a special token message, which it can give to any process to access CS

Centralized Control - Operations

- To enter a CS, send a request to the coordinator & wait for token.
- On exiting the CS, send a message to the coordinator to release the token.
- Upon receipt of a request, if no other process has the token, the coordinator replies with the token; otherwise, the coordinator queues the request
- Upon receipt of a release message, the coordinator removes the oldest entry in the queue (if any) and replies with a token

Centralized Control - Features

- Safety, Liveness are guaranteed
- Ordering also guaranteed (what kind?)
- Requires 2 messages for entry + 1 messages for exit operation.
- Client delay: one round trip time (request + grant)
- Synchronization delay: 2 message latencies (release + grant)
- The coordinator becomes performance bottleneck and single point of failure

Decentralized MutEx

- More fault-tolerant than centralized approach
- Uses the Distributed Hash Table (DHT) approach to locate objects/replicas
  - Object names are hashed to find the node where they are stored (succ function)
- n replicas of each object are placed on n successive nodes
  - Hash object name to get addresses
- Now every replica has a coordinator that controls access

The Decentralized Algorithm

- Coordinators respond to requests at once:

  Yes OR No

- Majority: To use the resource, a process must receive permission from m > n/2 coordinators
  - If the requester gets fewer than m votes, it will wait for a random time and then ask again
- If a request is denied, or when the CS is completed, notify the coordinators who have sent OK messages, so they can respond again to another request. (Why is this important?)

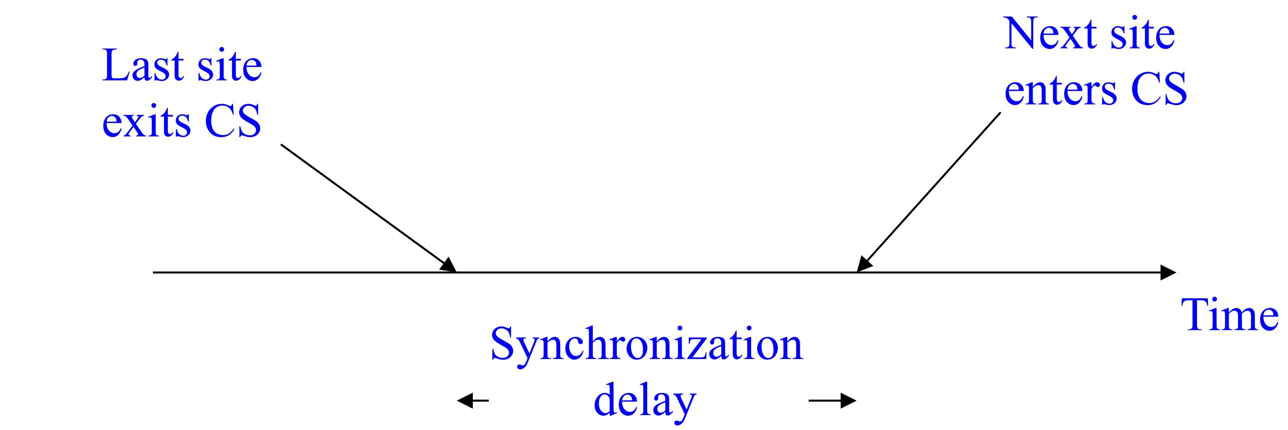
The Decentralized Algo - Analysis

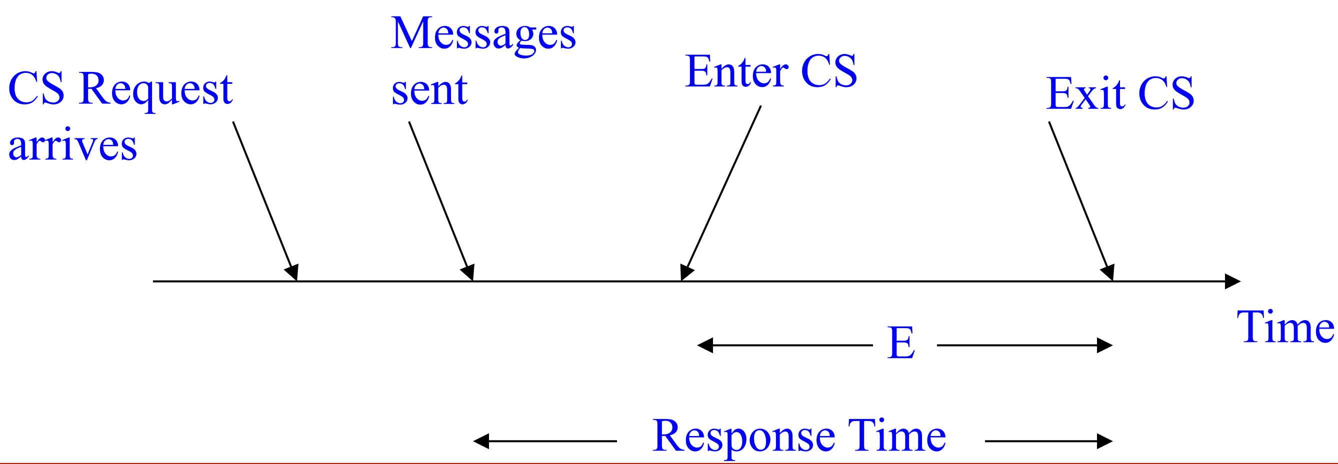
- More robust than the central coordinator approach. If one coordinator goes down others are available.
  - If a coordinator fails and resets then it will not remember having granted access to one requestor, and may then give access to another.
  - It is highly unlikely that this will lead to a violation of mutual exclusion

The Decentralized Algorithm - Issues

- If a resource is in high demand, multiple requests will be generated by different processes.
- High level of contention
- Processes may wait a long time to get permission - Possibility of starvation exists
- Resource usage drops.

Performance Metrics





Performance - Analysis

- Number of messages per CS invocation: should be minimized
- Synchronization delay, i.e., time between the leaving of CS by a site and the entry of CS by the next one: should be minimized
- Response time: time interval between request messages transmissions and the exit of CS
- System throughput, i.e., rate at which system executes the requests for CS: should be maximized
- If d is the synchronization delay, e the average CS execution time:

$$\text{System Throughput} = 1 / (d + e)$$

Performance – Analysis (contd)

- Low and High Load:
  - Low load: No more than one request at a given point in time
  - High load: Always a pending mutual exclusion request at a site
- Best and Worst Case:
  - Best Case (low loads): Round-trip message delay + Execution time - 2T + E
  - Worst case (high loads)
- Message traffic: low at low loads, high at high loads
- Average performance: when load conditions fluctuate widely

Token Ring Approach

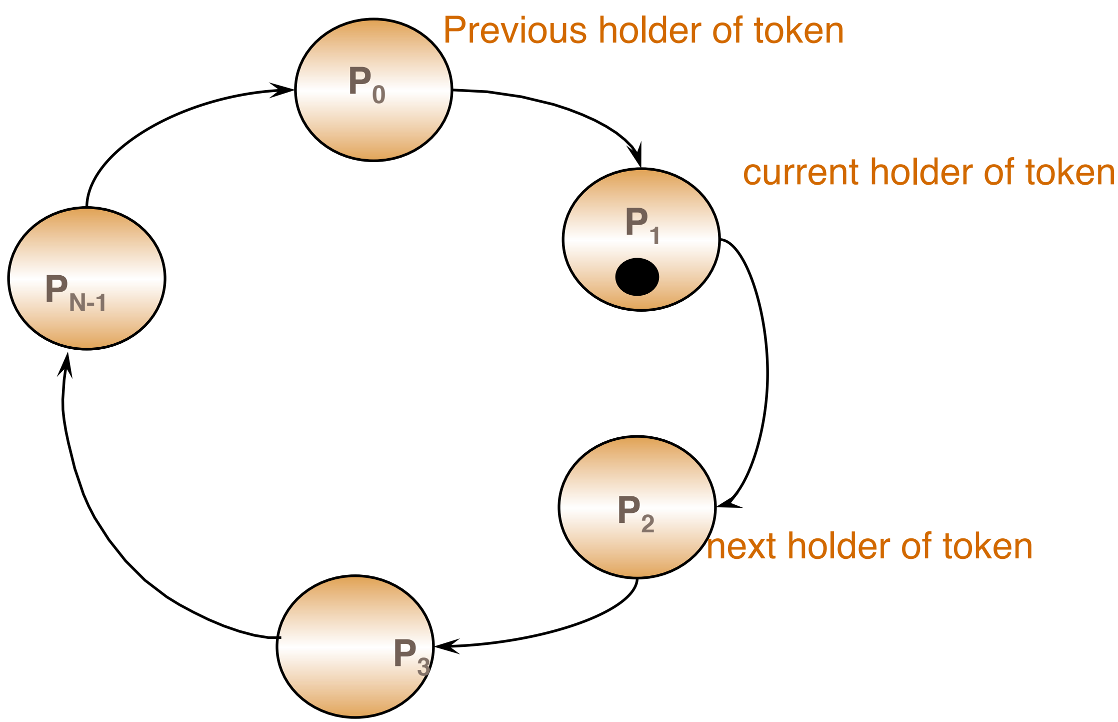
- Processes are organized in a logical ring: P_i has a communication channel to $P_{(i+1)} mod\ N$

Operations:

- Only the process holding the token T can enter the CS
- To enter the critical section, wait passively for T When in CS, hold on to T and don't release it
- To exit the CS, send T onto your neighbor
- If a process does not want to enter the CS when it receives T, it simply forwards T to the next neighbor

Token Rings – Illustration

- Request movements in an unidirectional ring network



Token Rings - Features

- Safety & Liveness are guaranteed
- Ordering is not guaranteed
- Bandwidth: 1 message per exit
- Client delay: 0 to N message transmissions
- Synchronization delay between one process's exit from the CS and the next process's entry is between 1 and N-1 message transmissions

Non-Token Based Algorithms

- Notations:
  - P_i: i th Process
  - R_i: Request set, containing IDs of all P_i s from which permission must be received before accessing CS
  - Non-token based approaches use time stamps to order requests for CS
  - Smaller time stamps get priority over larger ones
- Lamport's Algorithm
  - R_i = {P_1, P_2, ..., P_n}, i.e., all processes.
  - Request queue: maintained at each P_i ordered by time stamps.
  - Assumption: message delivered in FIFO

Lamport's Algorithm

- **Requesting CS:**
  - Send REQUEST(ts_i, i) where (ts_i, i) - Request time stamp; Place REQUEST in $request\_queue_i$
  - On receiving the message; P_j sends time-stamped REPLY message to P_i; Pi 's request placed in $request\_queue_j$
- **Executing CS:**
  - P_i has received a message with time stamp larger than (ts_i, i) from all other sites
  - P_i's request is the top most one in $request\_queue_i$
- **Releasing CS:**
  - Exiting CS: send a time stamped RELEASE message to all sites in its request set
  - Receiving RELEASE message: P_j removes P_i 's request from its queue

Notable Points

- Purpose of REPLY messages from i to j is to ensure that j knows of all requests of i prior to sending the REPLY (possibly any request of i with timestamp lower than j 's request)
- Requires FIFO channels
- 3(n – 1) messages per critical section invocation
- Synchronization delay = max mesg transmission time
- Requests are granted in order of increasing timestamps

Performance Improvements

- 3(n-1) messages per Critical Section invocation
  - (n - 1) REQUEST messages
  - (n - 1) REPLY messages
  - (n - 1) RELEASE messages
- Synchronization delay: T
- Optimization:
  - Suppress reply messages: For example, P_j receives a REQUEST message from P_i after sending its own REQUEST message with time stamp higher than that of P_i's then Do NOT send a REPLY message
  - Messages reduced to between 2(n-1) and 3(n-1)

Ricart & Agrawala's Algorithm

- A time-stamp based approach
- Originally proposed by Lamport using logical clocks
- Modified by Ricart & Agrawala

Ricart & Agrawala's Algorithm

Main Idea:

- Process j need not send a REPLY to Process i if j has a request with timestamp lower than the request of i (since i cannot enter before j here)
- Does not require FIFO
- 2(n – 1) messages per critical section invocation
- Synchronization delay = maximum message transmission time
- Requests granted in order of increasing timestamps

Ricart & Agrawala (contd)

- Processes need entry to critical section multicast a request, and can enter it only when all other processes have replied positively
- Messages requesting entry are of the form $<T, P_i>$
  - T - sender's timestamp (Lamport clock)
  - P_i the sender's identity

Ricart & Agrawala – Algorithm

To enter the Critical Section (CS):

- Set state = wanted
- multicast "request" to all processes (including timestamp)
- wait until all processes send back "reply"
- change state to held and enter the CS

On receipt of a request <T_j, P_j> at P_i:

- if (state == held) or (state == wanted & (T_i, P_i) < (T_j, P_j)) then enqueue the request
- else "reply" to P_j

On exiting the CS:

- change state to release and "reply" to all queued requests

Ricart & Agrawala – Simplified

To request Critical Section:

- ➔ send timestamped REQUEST message (ts_i, i)

On receiving request (ts_i, i) at j:

- ➔ if j is neither requesting nor executing critical section then send REPLY to i
- ➔ if j is requesting and i's request timestamp is smaller than j's request timestamp then
  - ➔ enqueue the request; Otherwise, defer the request

To enter Critical Section:

- ➔ Process i enters critical section on receiving REPLY messages from all processes

To release Critical Section:

- ➔ send REPLY to all deferred requests

Summary

- Mutual Exclusion Problem
- Basics of MutEx algorithms

- Various Types of MutEx algorithms
  - Token-based
    - Token rings
  - Non-Token based algorithm
    - Lamport's Algorithm
    - Ricart – Agrawala's Algorithm
- Performance Metrics

  - Many more to come up … stay tuned in !!

How to reach me?

- ➔ Please leave me an email:

  rajendra [DOT] prasath [AT] iiits [DOT] in

- ➔ Visit my homepage @

  - ➔ http://www.iiits.ac.in/FacPages/index-rajendra.html

  OR

  - ➔ http://rajendra.2power3.com

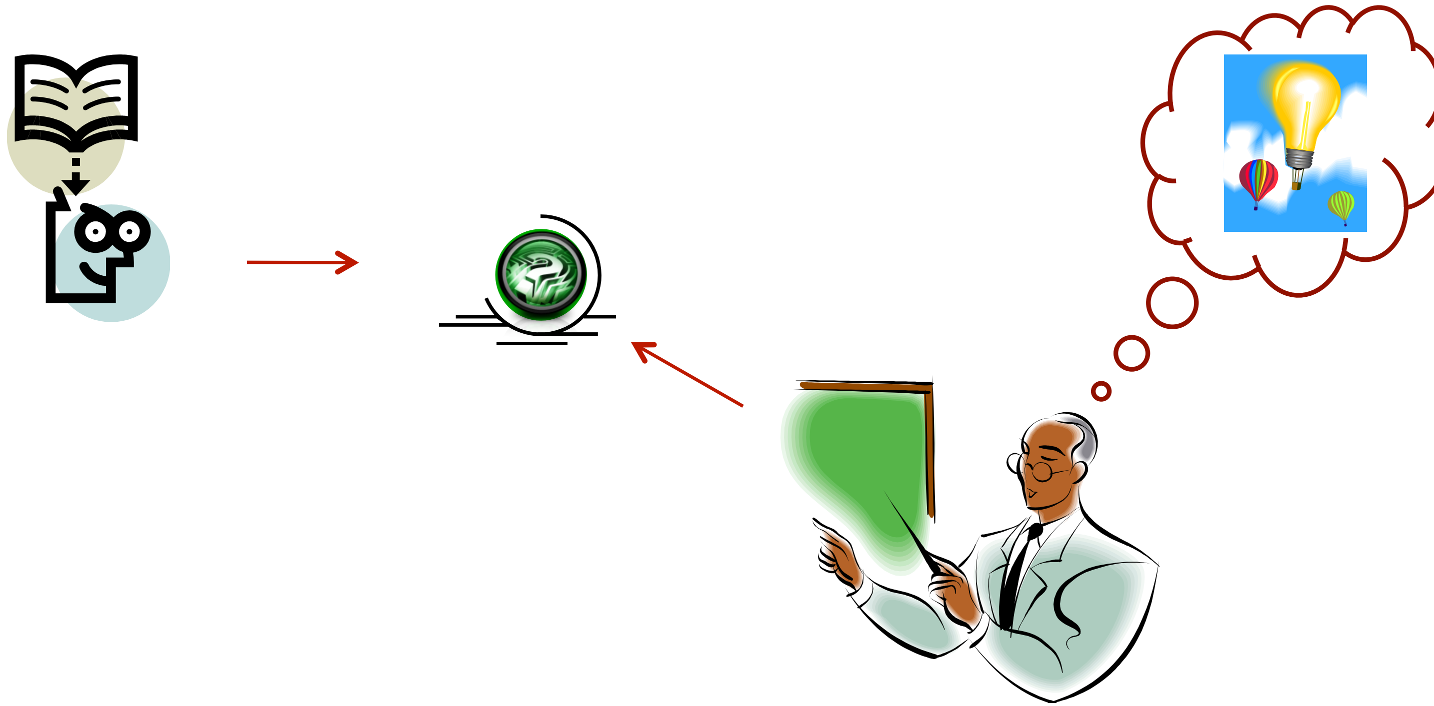
Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Thanks …



… Questions ???