Coordination and Agreement: Distributed Mutual Exclusion

Main Reference: Ch. 15 of Coulouris et.al

Supplementary: Ch. 12 of Tanenbaum & Steen

DALE CARNEGIE® TRAINING

Copyright 1996-98 © Dale Carnegie & Associates, Inc.

Aim:

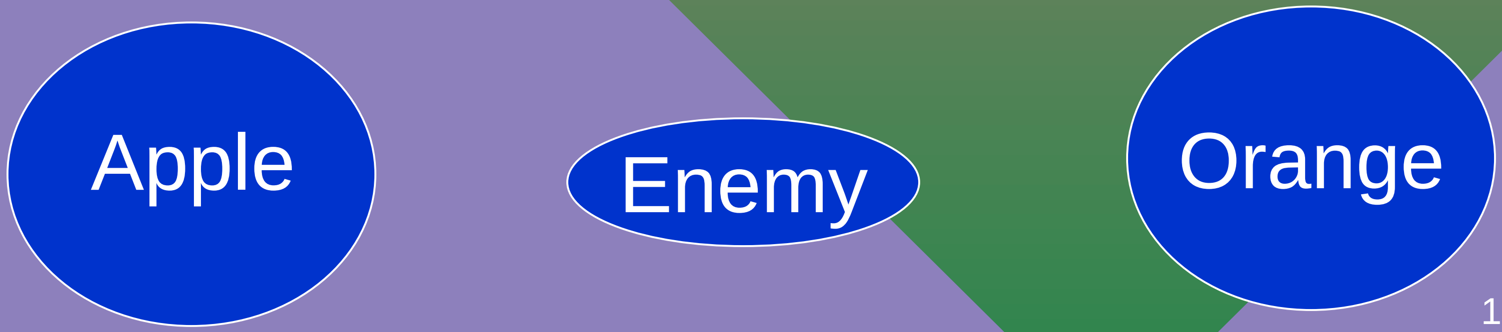
- Understand the problems of coordination and agreement in distributed systems; i.e. how processes coordinate their actions and agree on shared values.
- Study algorithmic techniques for distributed mutual exclusion and election problems, consensus and related problems.

Outline:

- **Distributed mutual exclusion**: the central server algorithm, ring-based algorithm, algorithm using multicast and logical clocks, Maekawa's voting algorithm.
- **Elections**: ring-based election algorithm, bully algorithm.
- **Consensus**: Byzantine agreement, consensus using failure detectors, consensus using randomization.

Agreement in Pepperland

- Apple and Orange (divisions of Pepperland Army) vs. Enemy
- Need to agree on: who will lead the charge and when to charge
- Communicate via messengers
- Asynchronous vs Synchronous Pepperland



Failure detection in Pepperland

- As long as a division is not yet defeated, it will keep sending messengers to the other division.
- In async system, hard to determine if a division has defeated: a messenger may take ages to reach the other dvision.
- In sync system, absence of messenger indicates defeat of the other division.

Impossibility of reaching agreement in the presence of failures

- A messenger might get caught by the enemy on it's way
- Message will not get across (either to charge or surrender), hence unable to reach an agreement.
- No protocol that guarantees agreement between Pepperland divisions can exist if messengers get captured.

Assumptions

- Asynchronous DS (no timing assumptions) vs synchronous DS (there is a bound on the max message transmission delay)
- Begin with algorithms that tolerate no failures BUT consider how to deal with failures.

Failure Assumptions

- Assume that each pair of processes is connected by reliable channels.
- No process failure implies a threat to the other processes' ability to communicate.
- In a sync system, a reliable channel delivers each message within a specified time frame.
- Any failed link/router will eventually be repaired.

Failure detectors

- Knowing when a processor has crashed.
- Two types of detectors: Reliable and unreliable.
- Unreliable: Unsuspected/Suspected states
- Reliable: Unsuspected/failure states
- To cope with failures, we must detect it.

Distributed mutual exclusion

- Motivation – critical sections
- Model and requirements
- Evaluation criteria
- Central server algorithm
- Ring-based algorithm
- Algorithm using multicast & logical clocks
- Maekawa’s voting algorithm
- Consideration for fault tolerance

Motivation-Critical sections

- Collection of processes share resources
- When accessing shared resources (critical section), must ensure consistency and prevent interference.
- Need for distributed mutual exclusion
- Solution must be based solely on message-passing: cannot use shared memory.

Model and Requirements

- A system of N processes p_i, I=1,2,…,N
- Assumptions:
  - Asynchronous system
  - Processes do not fail
  - Message delivery is reliable
- Requirements:
  - Safety: at most one process may execute CS at a time (mutual exclusion)
  - Liveliness: requests to enter/exit CS eventually succeed (no deadlock, no starvation)
  - Ordering: if one request to enter CS happened before another, entry to CS is granted in that order.

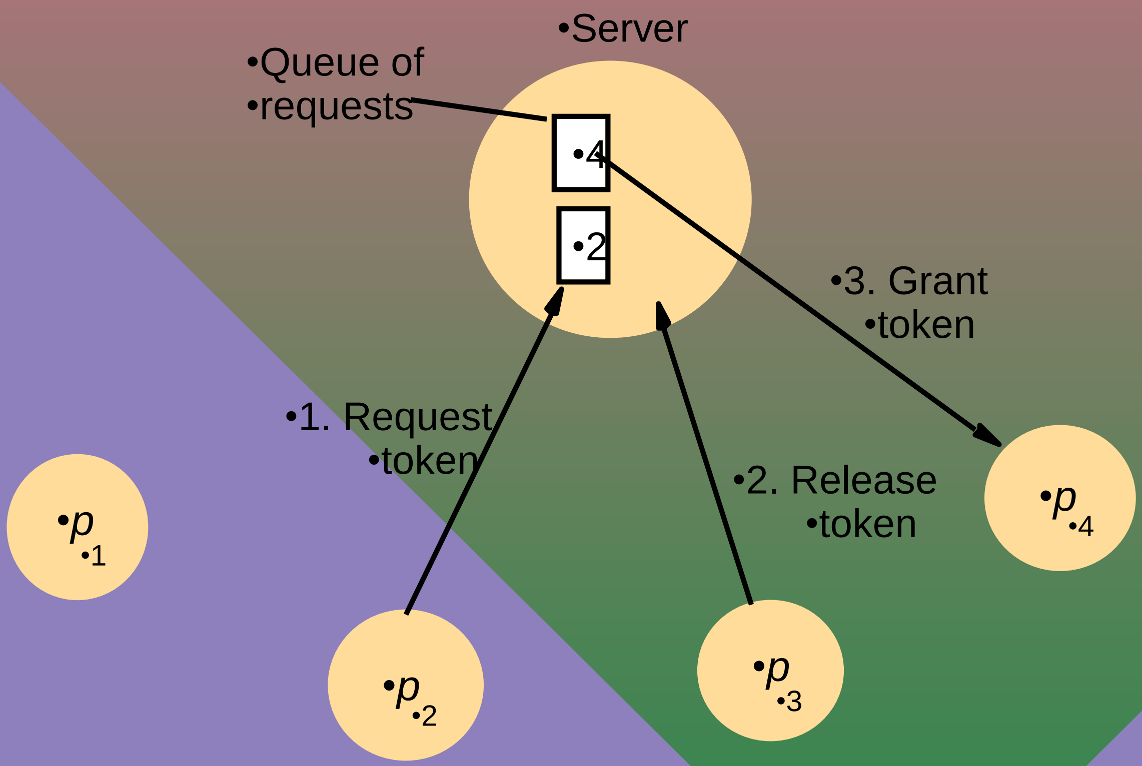
Evaluation Criteria

- Consumed bandwidth: proportional to number of messages sent in each entry-CS and exit-CS operations
- Client delay: time spent at entry-CS and exit-CS operations (worst-case)
- Synchronization delay: time between one process exiting CS and another process entering CS

Central server algorithm

- A server grants permission to enter CS (via token)
- To enter CS: send request to server and wait until it replies with a token.
- To exit CS: send token to server
- Server grants token if no process holds it, else queue the request. A FCFS queue of requests is mantained by server.

Server managing a mutual exclusion token for a set of processes



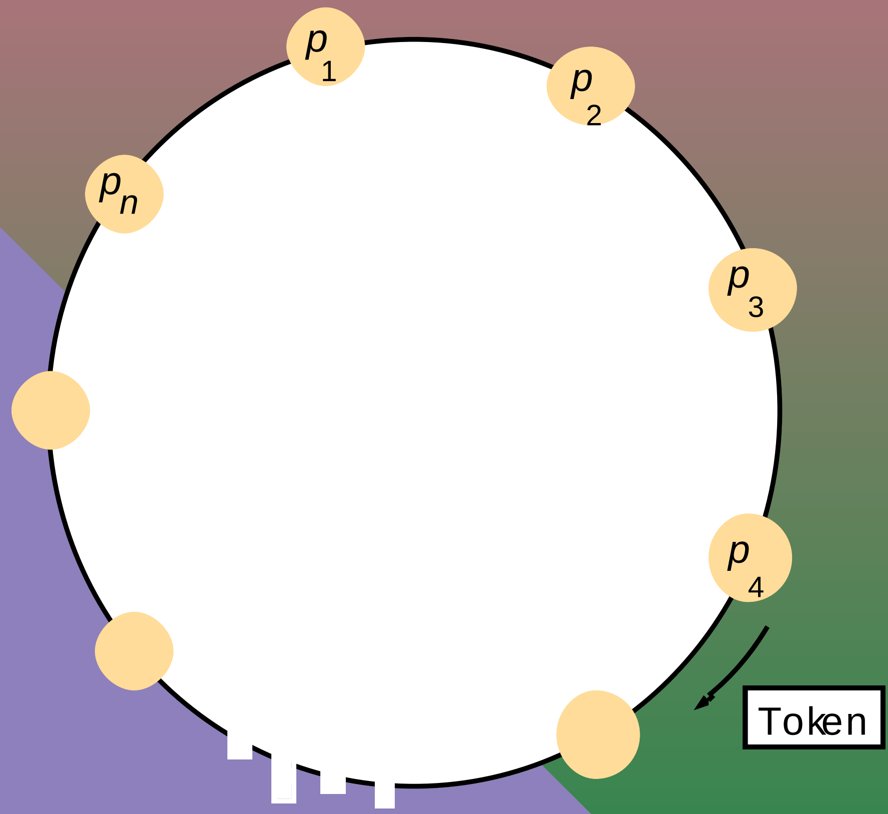
Evaluation of algorithm

- Enter CS takes two messages (a request followed by a grant)
- Exit CS takes 1 release message
- Synchronization delay: time taken for a round trip; I.e. a release message to the server followed by a grant message to the next process to enter CS.

Ring-based algorithm

- Processes arranged in a logical ring
- Each process has a communication channel to the next process in the ring.
- Token is passed from process to process in a single direction
- If process that receives token does not want to enter CS, it passes token to the next process. Otherwise, it retains token until exiting CS.

A ring of processes transferring a mutual exclusion token



Evaluation of algorithm

- Bandwidth: Continuously consumed, except when a process is in CS.
- Requesting process delay between 0 messages (when it has just received the token) and N messages (when it has just passed on the token)
- To exit CS requires 1 message.
- Synchronization delay between exit CS and next enter CS is 1 to N message transmissions.

Algorithm using multicast and logical clocks

- Due to Ricart and Agrawala(1981)
- Multicast a CS request to all N-1 peers, request is granted when all peers reply.
- Each process keeps a Lamport clock; a monotically increasing software counter whose value has no relationship with any physical clock. Each process keeps it’s own logical clock which it uses to apply Lamport timestampp of events.

Algorithm using multicast and logical clocks-contd

- Request message is of the form <T,p_i>, whereT is the sender's timestamp and p_i is the sender's identifier.
- Process states:

RELEASED(outside CS)
WANTED(wanting entry to CS)
HELD(in CS)

Ricart and Agrawala's algorithm

- If process requests entry and state of all other processes is RELEASED, all the process will reply and entry is granted.

On initialization
state := RELEASED;
To enter the section
state := WANTED;
Multicast *request* to all processes; request processing deferred here
T := request's timestamp;
Wait until (number of replies received = (N – 1));
state := HELD;

Ricart and Agrawala’s algorithm

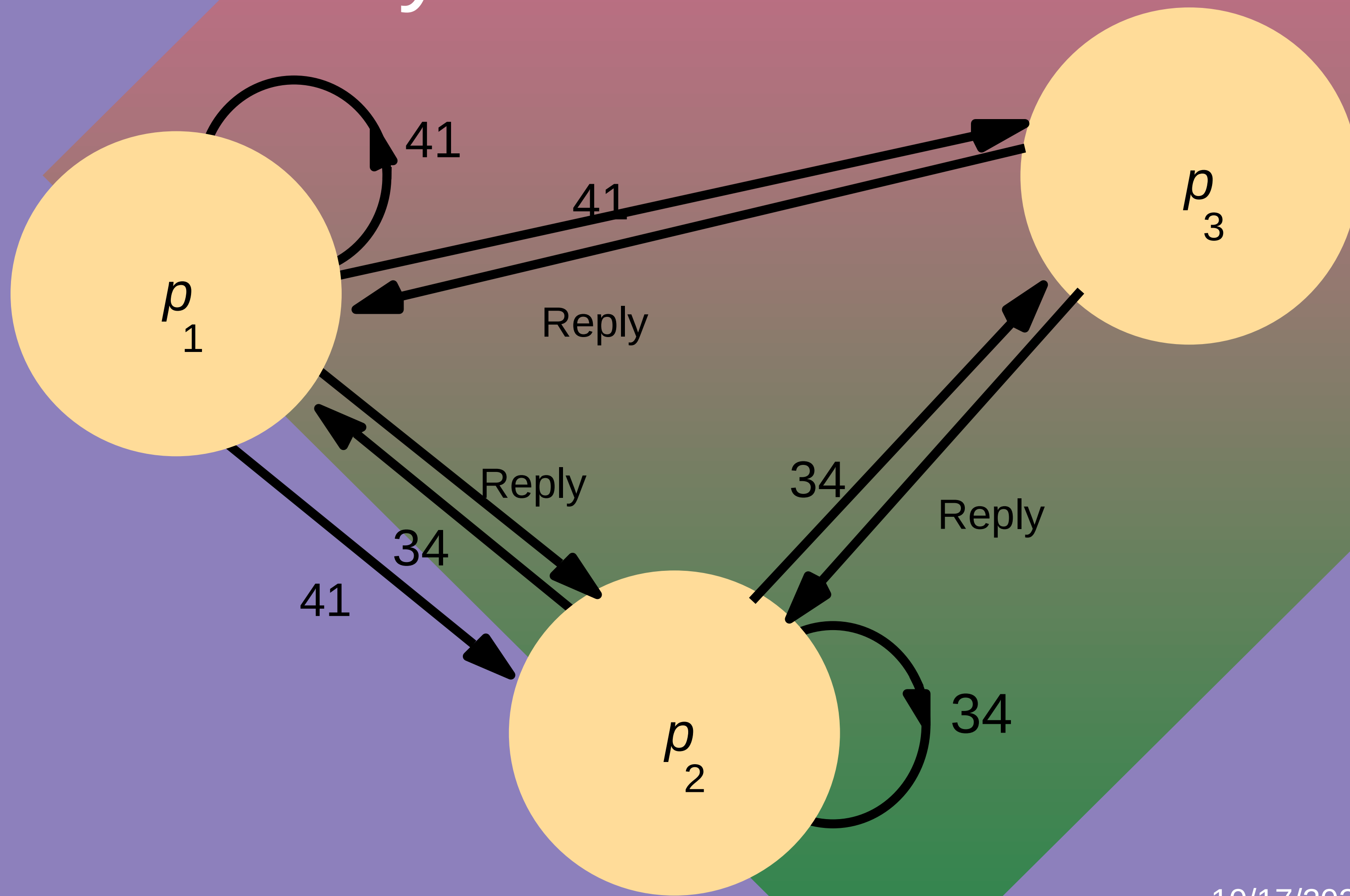
- If some processes is in state HELD, that process will not reply until it exits CS.

On receipt of a request $<T_i, p_i>$ *at* p_j $(i \neq j)$
if (*state* = HELD or (*state* = WANTED *and* $(T, p_j) < (T_i, p_i)$))
then
    queue *request* from p_i without replying;
else
    reply immediately to p_i;
end if
To exit the critical section
state := RELEASED;
reply to any queued requests;

Ricart and Agrawala’s algorithm

- If two or more request entry at the same time, whichever process request bears the lowest timestamp will be the first to collect N-1 replies, granting it next entry.
- If process bear equal timestamps, requests are ordered according to the processes identifier.

Multicast synchronization



Evaluation of algorithm

- To enter CS takes 2(N-1) messages; N-1 to multicast request, followed by N-1 replies.
- Synchronization delay: 1 message transmission.

Maekawa’s voting algorithm

- In order for a process to enter CS, not necessary for all peers to grant it access.
- Sufficient to obtain permission from subsets of peers, as long as the subsets used by any two processes overlap.
- Processes vote for one another to enter CS. A candidate must collect sufficient votes to enter.

Maekawa’s algorithm

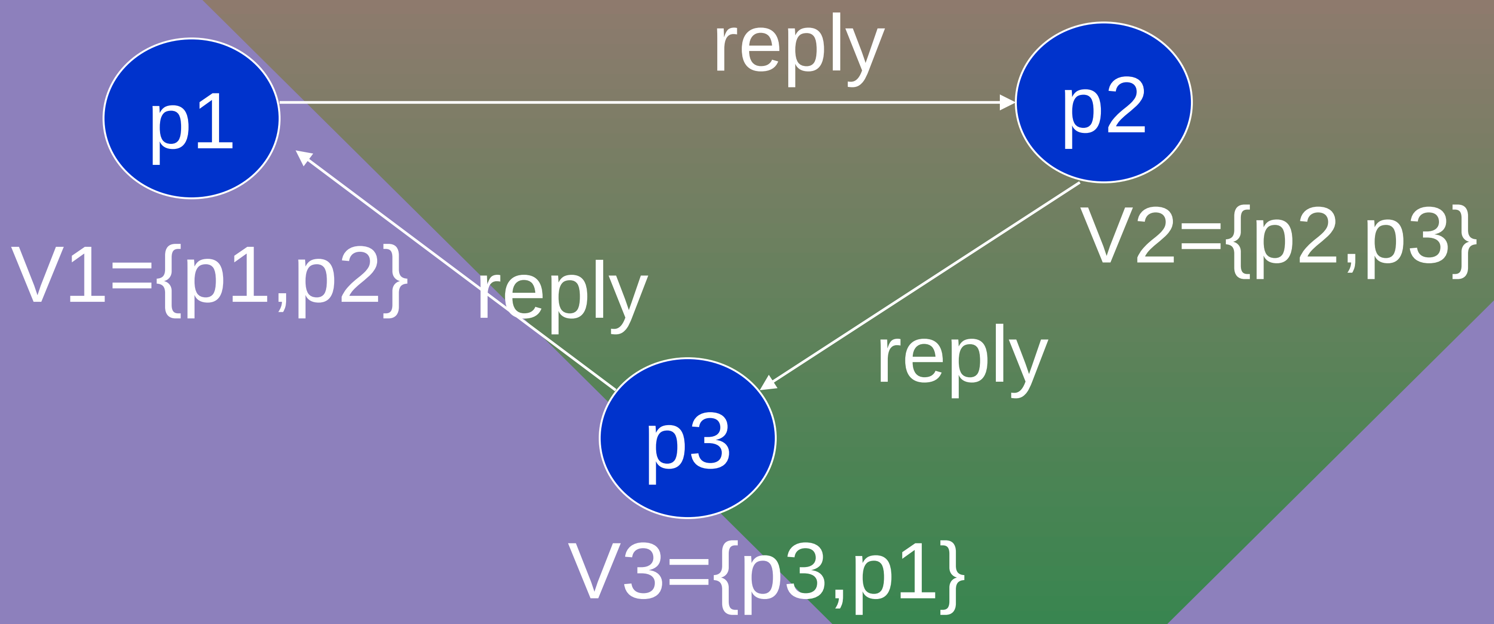
- Each process is associated with a voting set V_i
- To enter CS, process p_i sends request to K-1 members of V_i where $K=| V_i |$
- P_i cannot enter CS until it has received K-1 reply messages.
- When a process p_j in V_i receives p_i’s request, it replies immediately, provided it’s not in HELD state or has replied (voted) since it last received a release message.

Maekawa's algorithm - contd

- Else, it queues request in order of arrival, but does not reply.
- When it receives a release message, it removes the head of the queue and votes.
- To leave CS, p_i sends a release message to all K-1 members of V_i

Maekawa's algorithm-deadlock prone

Each process has received 1 out of 2 votes, so cannot proceed.



Consideration for Fault Tolerance

- Lost messages:None of the algorithms can tolerate loss of messages.
- process crash:
- Ring-based algorithm-NO WAY
- Maekawa’s algorithm – OK, if not in voting set
- Central server, OK if neither holds or requested token.