Coordination and Agreement: Elections, Consensus and Related Problems

Reference: Chap. 15 Coulouris et al.

Elections

- To choose a unique process to play a particular role (ex: central server in distributed mutual exclusion)
- Election problem: choose ONE out of N processes as the coordinator process.
- Main issue: each process only call for one election at a time, so N processes can call N concurrent elections.
- Suppose that the coordinator is the process with the largest identifier, where identifiers are unique and totally ordered.

Requirements

- Choice of elected process must be unique
- Each process has a variable ELECTED = ID (id of coordinator) or U (undefined)
- During any particular election:
  - Each participant process must have ELECTED = U or P, which is the non-crashed process at the end of the run with the largest identifier.
  - All processes participate and eventually elect or crash.

Evaluation criteria

- Bandwidth utilization: Total number of messages sent.
- Turnaround time: the number of serialized message transmission times between the initiation and termination of a single run.

Ring-based election algorithm (Chang and Roberts, 1979)

- To elect a single process called coordinator, a process with the largest identifier.
- Processes are arranged in a logical ring.
- Assumptions:
  - each process has a communication channel to the next process
  - messages sent in clockwise direction
  - no failures occur
  - asynchronous system.

Ring-based election -contd

- Initially, every process is a non-participant.
- Process that calls the election:
  - marks itself as participant
  - send it’s ID to the next process with ELECTION message.

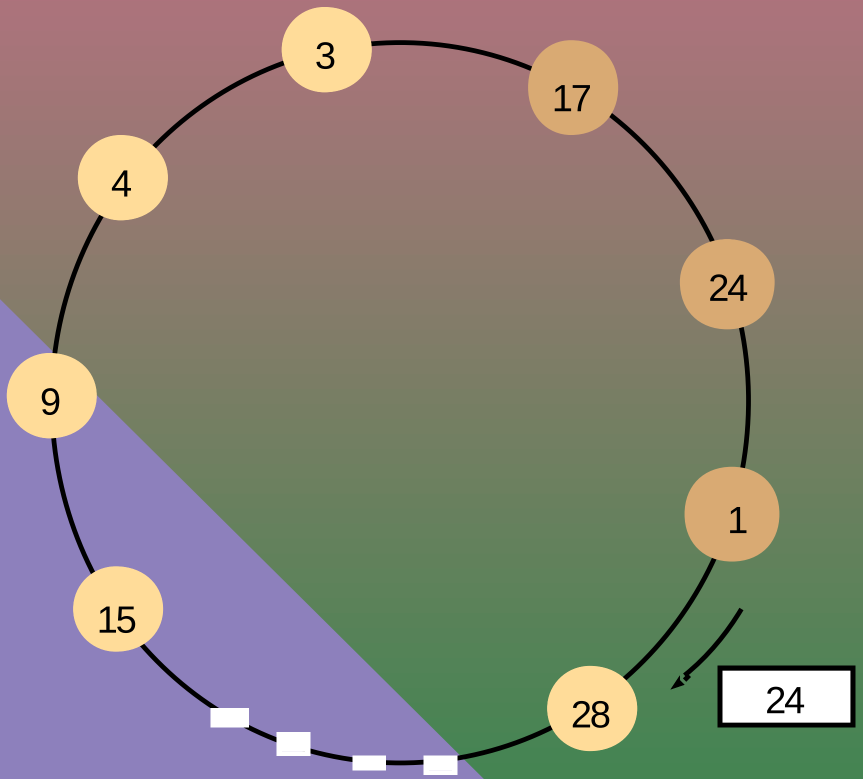
Ring-based election -contd

- Process X receives ELECTION message:
  - If ID of X < ID received, forward message to next process and becomes participant.
  - If ID of X > ID received and X is not participant, X replaces ELECTION message ID with ID of X and becomes participant. If X is a participant, ignore election message.
  - If ID of X = ID received, process X becomes coordinator, marks itself as non-participant and sends ELECTED message to next process, announcing X is coordinator.

Ring-based election -contd

- Process Y receives ELECTED message:
  - Y marks itself as non-participant
  - Sets ELECTED = X
  - (Unless X=Y), it forwards ELECTED message to neighbour.

A ring-based election in progress



Note: The election was started by process 17.
The highest process identifier encountered so far is 24.
Participant processes are shown darkened

Evaluation

- If a single process starts the election, the worst case is when it's anticlockwise neighbour has the highest identifier
  - N-1 messages required to reach the neighbour
  - another N messages before it gets elected
  - another N messages to announce that it is elected
  - Total of 3N-1 messages

Bully Algorithm (Garcia-Molina, 1982)

- Assumptions:
  - reliable message delivery
  - allows processors to crash
  - Synchronous system with timeouts to detect failure
  - Timeout $T=2*T_{trans} + T_{process}$
  - Each process knows ID of other processes and can communicate with them.

Bully Algorithm-contd

- Three message types:
  - ELECTION: to announce an election
  - ANSWER: response to an ELECTION message
  - COORDINATOR: announce elected process
- Process with max ID can elect itself by sending COORDINATOR message to other processes
- A process with a lower ID begins an election by sending ELECTION message to others and awaits ANSWER message as response

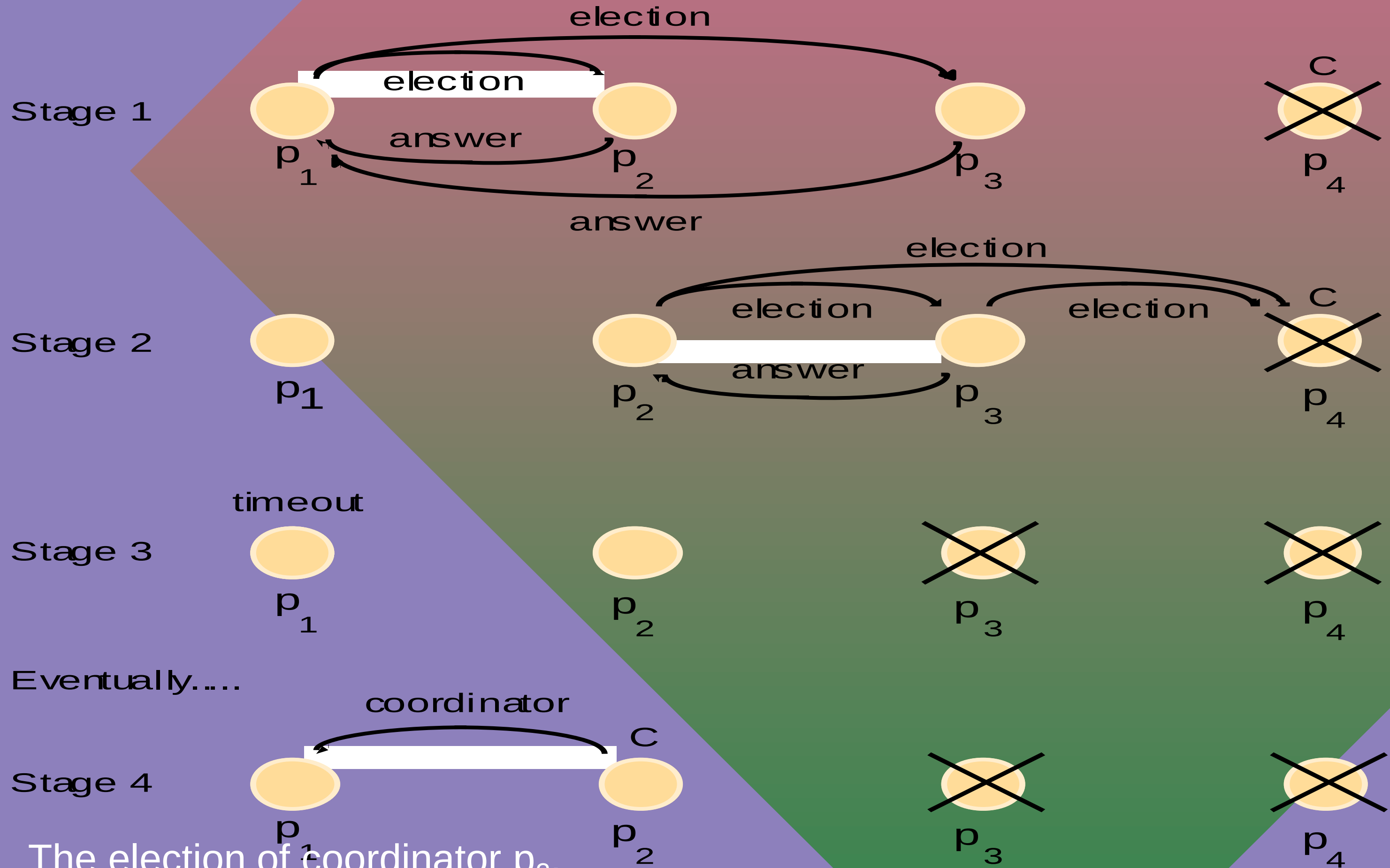
Bully Algorithm-contd

- A process P can send ELECTION message to all processes with ID > ID of P in order to start an election:
  - If no ANSWER arrives within T, P sends COORDINATOR message to all processes with ID<ID of P.
  - If an ANSWER has arrived, wait for COORDINATOR message from elected process. Start new election if COORDINATOR message does not arrive.

Bully Algorithm-contd

- If process P receives ELECTION message:
  - it sends back ANSWER message
  - It starts a new election. WHY????
- Because … P will only receive an ELECTION message from a process whose ID < ID of P
- On the other hand, P would have already started an election since it has a larger ID anyway.
- When a process is restarted it always starts an election.

The bully algorithm



The election of coordinator p_2,

after the failure of p_4 and then p_3

Evaluation

- Best case is when the process with the second highest identifier notices the coordinator's failure and elects itself: N-2 COORDINATOR messages.
  - Turnaround time is 1 message transmission time: COORDINATOR

Evaluation

- Worst case is when process with the lowest ID detects failure and calls for election. N-1 processes will begin election, each sending messages to processes with higher ID, ending up with $O(N^2)$
  - Turnaround time is approx. 5 message transmission times if there are no failures during the run: election, answer, election, answer, coordinator

Agreement/Consensus

- General problem: A set of processes need to agree on a value/decision, after one or more processes have proposed what that value/decision should be.
- General goal of distributed agreement is to have all non-faulty processors reach consensus on some issues within a finite number of steps.
- Several issues:
  - Reliability of message delivery.
  - Can processes crash, and if so, fail silent or Byzantine
  - Is system synchronous or asynchronous?

Two army problem

- Perfect processor but communication lines that lose messages
- Recall the Pepperland Army problem:
  - Apple has 3000 troops, Orange 3000 troops and enemy has 5000 troops
  - If Apple and Orange can coordinate their attack on the enemy, they will win.
  - If either one attacks by itself, it will be slaughtered.
  - The goal of Apple and Orange is to reach an agreement on attacking.
  - They can only communicate using an unreliable channel-sending a messenger who is subject to capture by the enemy.

Typical scenario

Apple: I have a plan, let's attack at dawn

Orange: Splendid, see you at dawn

Apple: Does Orange know that message has reached me?

Typical scenario-contd

Apple: Orange, message received.

Battle is set.

Orange: Does Apple know that message

has arrived?

This is never ending and agreement cannot be reached.

Consensus Problem-definition

- Every process i proposes a value v_i while in the undecided state.
- Process i exchanges messages until it makes a decision d_i and moves to a decided state.
- Requirements:
  - Termination: All correct processes must make a decision.
  - Agreement: same decision for all correct processes.
  - Integrity: if the correct processes all chose the same value, then any correct process in the decided state has chosen that value.

Consensus Problem-algorithm

- Consider a system in which processes cannot fail.
- Collect processes in a group and each process reliably multicast proposed value to the members of the group.
- Each process waits until it has collected N values, evaluates the function majority(v1,v2,…vn) which returns the value that occurs most frequently. If none, return U.
- If values are ordered, min or max function can be used.

Byzantine generals problem

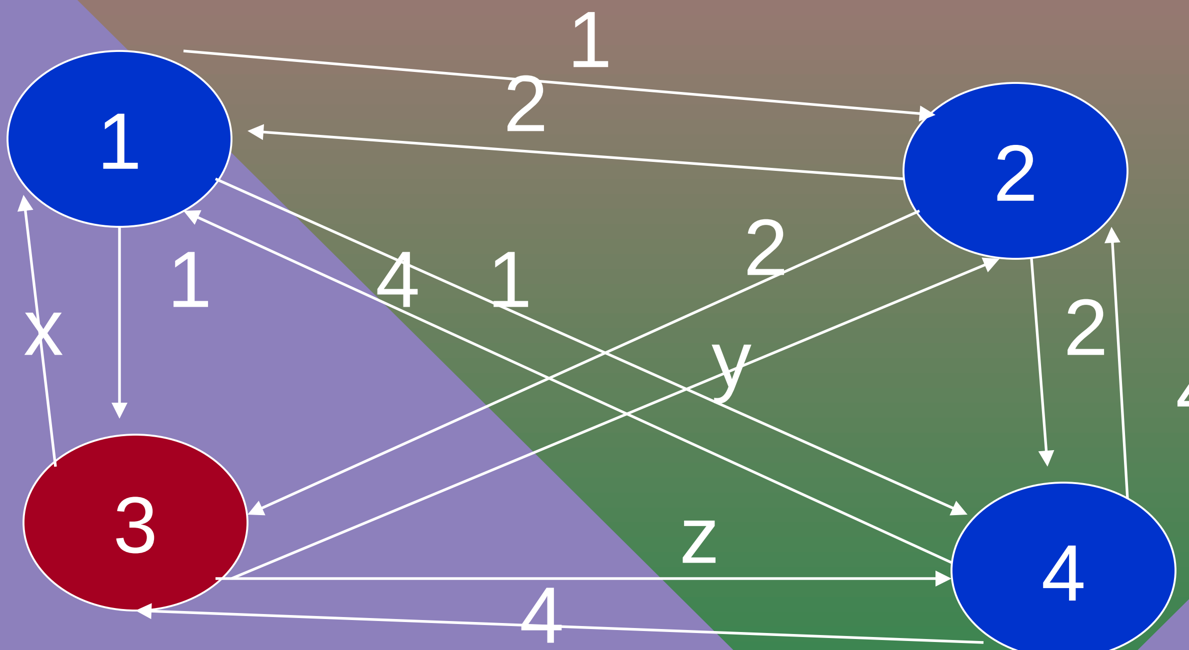
- Perfect communication but faulty processors
- N Pepperland troops each headed by a general
- Communication via phone but m generals are traitors (faulty) who provide incorrect or contradictory info.
- Can the loyal generals still reach an agreement?

Byzantine generals-contd

- Each general is assumed to know how many troops he has.
- Goal of the problem is for the generals to exchange troop strength.
- At the end, each general has a vector of length N of troop strength.
- If general i is loyal, element i is his troop strength. Else, it is undefined.

Byzantine generals-algorithm (Lamport et al., 1982)

- Consider 3 loyal generals and 1 traitor



Byzantine generals-algorithm Step 2

- Results in step 1 are collected as a vector.
- 1 received (1,2,x,4)
- 2 received (1,2,y,4)
- 3 received(1,2,3,4)
- 4 received (1,2,z,4)

Byzantine generals-algorithm Step 3

- Every general pass his vector to every other general. General 3 lies again.
- 1 received: (1,2,y,4),(a,b,c,d),(1,2,z,4)
- 2 received: (1,2,x,4),(e,f,g,h),(1,2,y,4)
- 3 received: (1,2,x,4),(1,2,y,4),(1,2,z,4)
- 4 received: (1,2,x,4),(1,2,y,4),(i,j,k,l)

Byzantine generals-algorithm Step 4

- Each general examines i-th element of each of the newly received vector.
- If any value has a majority, that value is put into the result vector. If no majority, mark as UNKNOWN.
- Generals 1,2,4 agree on (1,2,UNKNOWN,4)