PAYAP UNIVERSITY

Coordination and Agreement

Vaidė Narváez

Computer Information Systems

August 25th, 2010

PAYAP UNIVERSITY

Goal

How a set of processes can coordinate their actions or agree on shared values despite failures?

PAYAP UNIVERSITY

Goal

How a set of processes can coordinate their actions or agree on shared values despite failures?

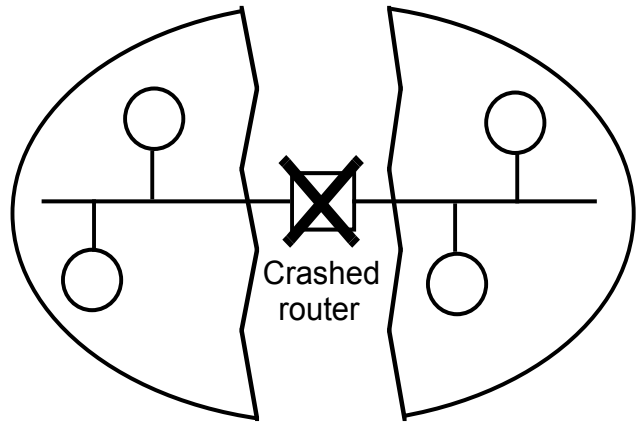
Cockpit of Space Shuttle Atlantis

PAYAP UNIVERSITY

Failures and Failure Detectors

Assumptions:

- Reliable channels
  - Eventual delivery of messages
  - Asymmetric connectivity
  - Intransitive connectivity

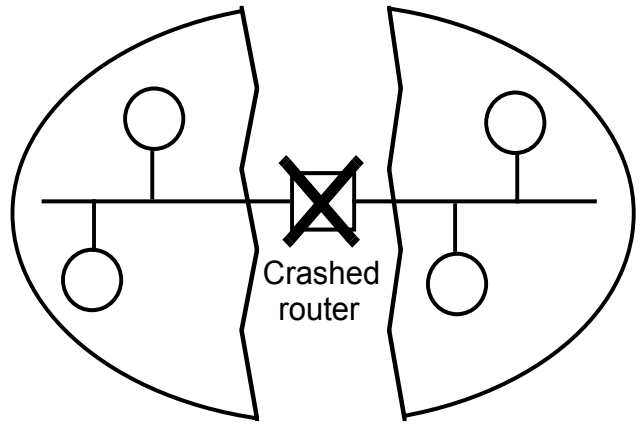


PAYAP UNIVERSITY

Failures and Failure Detectors

Assumptions:

- Reliable channels
  - Eventual delivery of messages
  - Asymmetric connectivity
  - Intransitive connectivity



- Processes fail only by crashing

PAYAP UNIVERSITY

Failures and Failure Detectors

PAYAP UNIVERSITY

Failures and Failure Detectors

A failure detector is a service that processes queries about whether a particular process has failed.

- Unreliable Failure Detectors : Unsuspected or Suspected
- Reliable Failure Detectors : Unsuspected or Failed

PAYAP UNIVERSITY

Failures and Failure Detectors

A failure detector is a service that processes queries about whether a particular process has failed.

- Unreliable Failure Detectors : Unsuspected or Suspected
- Reliable Failure Detectors : Unsuspected or Failed

How can you implement a FD?

PAYAP UNIVERSITY

Failures and Failure Detectors

A failure detector is a service that processes queries about whether a particular process has failed.

- Unreliable Failure Detectors : Unsuspected or Suspected
- Reliable Failure Detectors : Unsuspected or Failed

How can you implement a FD?
Practical use of such a FD?

PAYAP UNIVERSITY

Distributed Mutual Exclusion

- N processes $p_i, i = 1, 2, ..., N$ and no shared variables
- Common resource is accessed in a critical section
- Asynchronous system, no process failures, reliable message delivery

Critical section:

- *enter*() - enter critical section (block if necessary)
- *resourceAccess*() - access shared resources in critical section
- *exit*() - leave critical section (other processes may now enter)

PAYAP UNIVERSITY

Mutual Exclusion: Properties

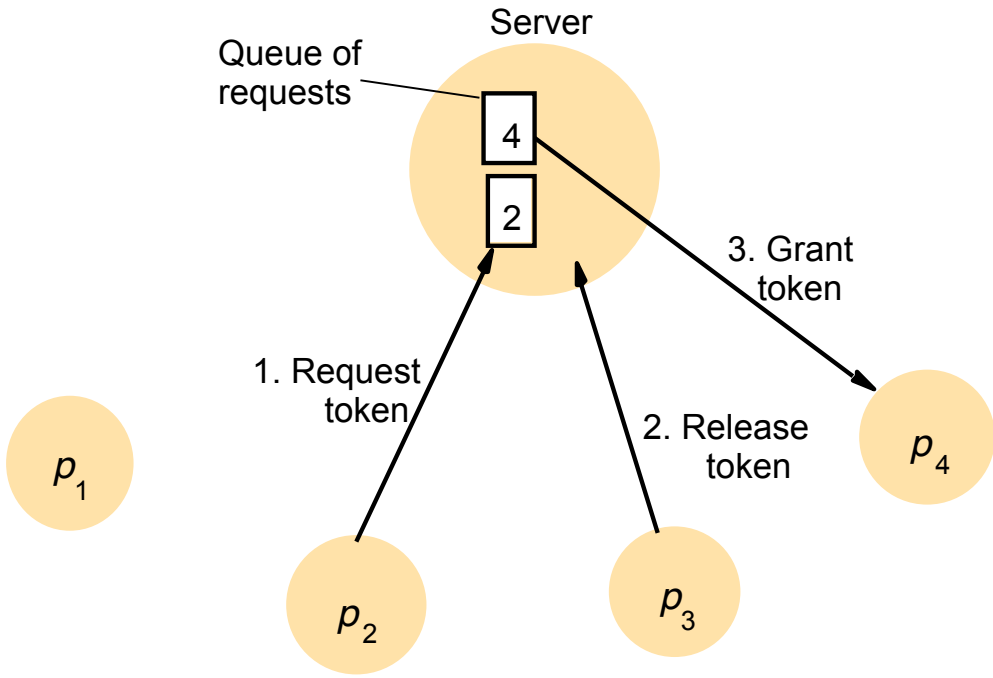
- **ME1: (safety)** At most one process may execute in the CS at a time
- **ME2: (liveness)** Requests to enter and exit the CS eventually succeed
- **ME3: ($\rightarrow$ ordering)** If one request to enter the CS happened-before another, then the entry to the CS is granted in that order

PAYAP UNIVERSITY

Mutual Exclusion: Evaluation

- *bandwidth* consumption: number of messages sent
- *client delay* incurred by a process at entry and exit operation
- *throughput* measured by *synchronization delay*: delay between one's exit and next's entry

The Central Server Algorithm



- server keeps track of a token (permission to enter CS)
- a process requests the server for the token
- the server grants the token if it has the token
- a process can enter if it gets the token, otherwise waits
- when done, a process sends release and exits

PAYAP UNIVERSITY

The Central Server Algorithm

- Safety ?
- Liveness ?
- Ordering ?

PAYAP UNIVERSITY

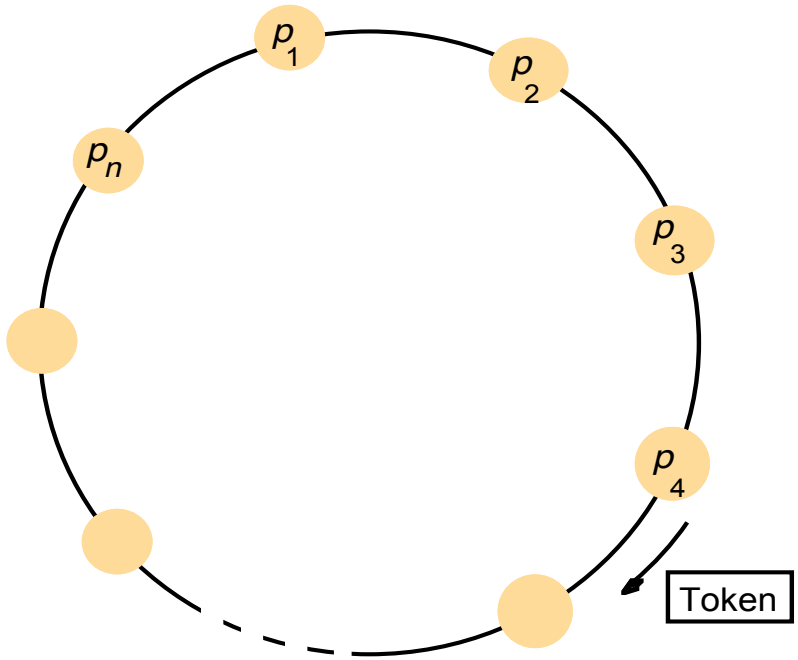
The Central Server Algorithm

- Safety ?
- Liveness ?
- Ordering ?

- Performance: server is a bottleneck for the whole system

PAYAP UNIVERSITY

The Ring-based Algorithm



- logical ring, could be unrelated to the physical configuration
- p_i sends messages to $p_{(i+1) \bmod N}$
- when a process holds the token, it can enter the CS, otherwise waits
- when a process releases the token (exit the CS), it sends to its neighbor

PAYAP UNIVERSITY

The Ring-based Algorithm

- Safety ?
- Liveness ?
- Ordering ?

PAYAP UNIVERSITY

The Ring-based Algorithm

- Safety ?
- Liveness ?
- Ordering ?

- Performance: continuously consumes network bandwidth

PAYAP UNIVERSITY

Multicast and Logical Clocks

- multicast a request message for the token
- enter only if all the other processes reply
- totally-ordered timestamps: $< T, p_i >$
- each process keeps a state: *RELEASED, HELD, WANTED*
- if all have *state* $=$ *RELEASED*, all reply, a process can hold the token and enter
- if a process has *state* $=$ *HELD*, doesn't reply until it exits
- if more than one process has *state* $=$ *WANTED*, process with the lowest timestamp will get all $N - 1$ replies first.

Ricart and Agrawala's algorithm

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

request processing deferred here

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;

PAYAP UNIVERSITY

Multicast synchronization