Lecture 11: Coordination and Agreement

- Central server for mutual exclusion
- Election – getting a number of processes to agree which is "in charge"

CDK4: Chapter 12 (esp. Section 12.3)

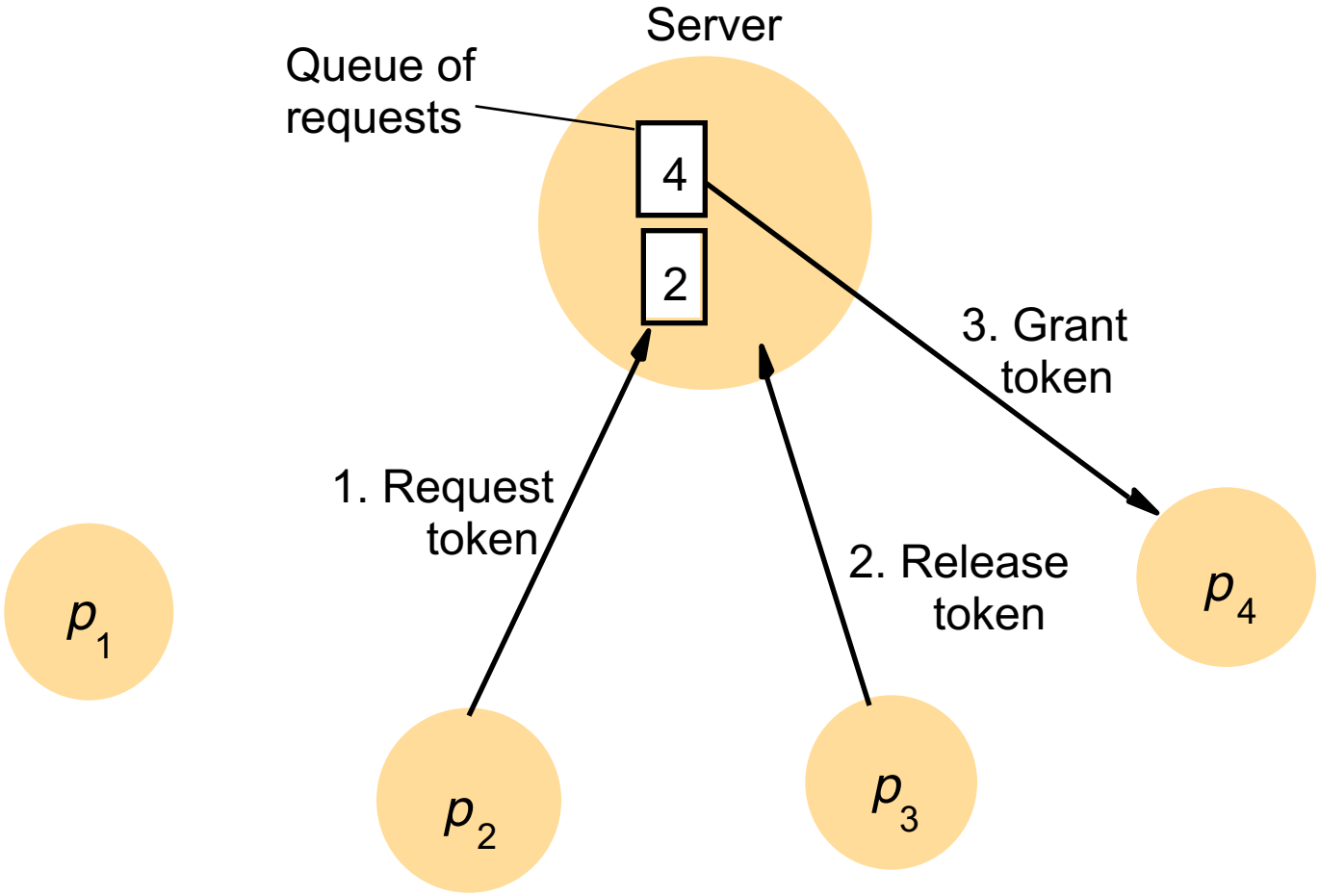
CDK5: Chapter 15 (esp. Section 15.3)

TVS: Chapter 6 (esp. Section 6.5)

(Distributed) Mutual exclusion

- 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
- On machines which don't even share memory, need something different
- The simplest solution is a central server
- Client requests token on entry, and releases it on exit
- Server queues requests when it has no token

Server managing a mutual exclusion token for processes (CDK Fig 12.2)



Alternatives …?

- The server is a single point of failure
- Also possibly a bottleneck for the system
- However many very clever, more distributed algorithms are generally worse in terms of number of messages, and have other problems!

Election – why?

Need a distinguished process:

- To implement locking/mutual exclusion
- Or to coordinate distributed transactions

Election is also a good introduction to what is involved in getting processes in a distributed system to cooperate.

Two Algorithms

- A ring-based election algorithm
- The bully algorithm

Same scenario:

- A fixed set of processes: p1 … pn
- Want to elect the process with the largest *identifier* – where identifiers are totally ordered, e.g. <1/(load+1), i>

Requirements

- Want all processes to agree who is elected i.e. they need to learn the result of the election
- Any process can initiate an election – so there may be more than one happening at any one time ….

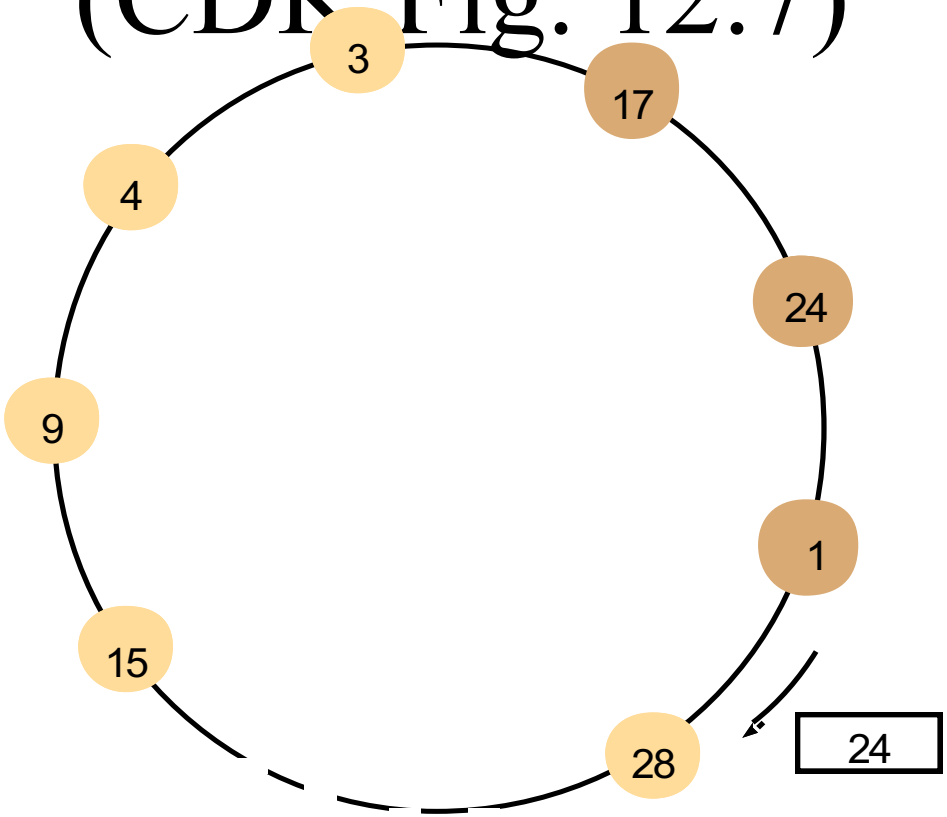
Ring-based algorithm

- Arrange the processes in a logical ring – so each knows which is next in order
- Assume no failures and system is asynchronous
- Initially every process is a *non-participant*
- Initiating process changes itself to be a participant and sends its identifier in an election message to its neighbour

Action on receiving election message

- Receiver compares its own identifier with that in the message
- If the identifier in the message is larger it forwards the election message unchanged
- Otherwise if receiver is already a participant, it discards the message
- Otherwise it forwards the message, but with its own identifier
- It is now a participant

A ring-based election in progress (CDK Fig. 12.7)



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

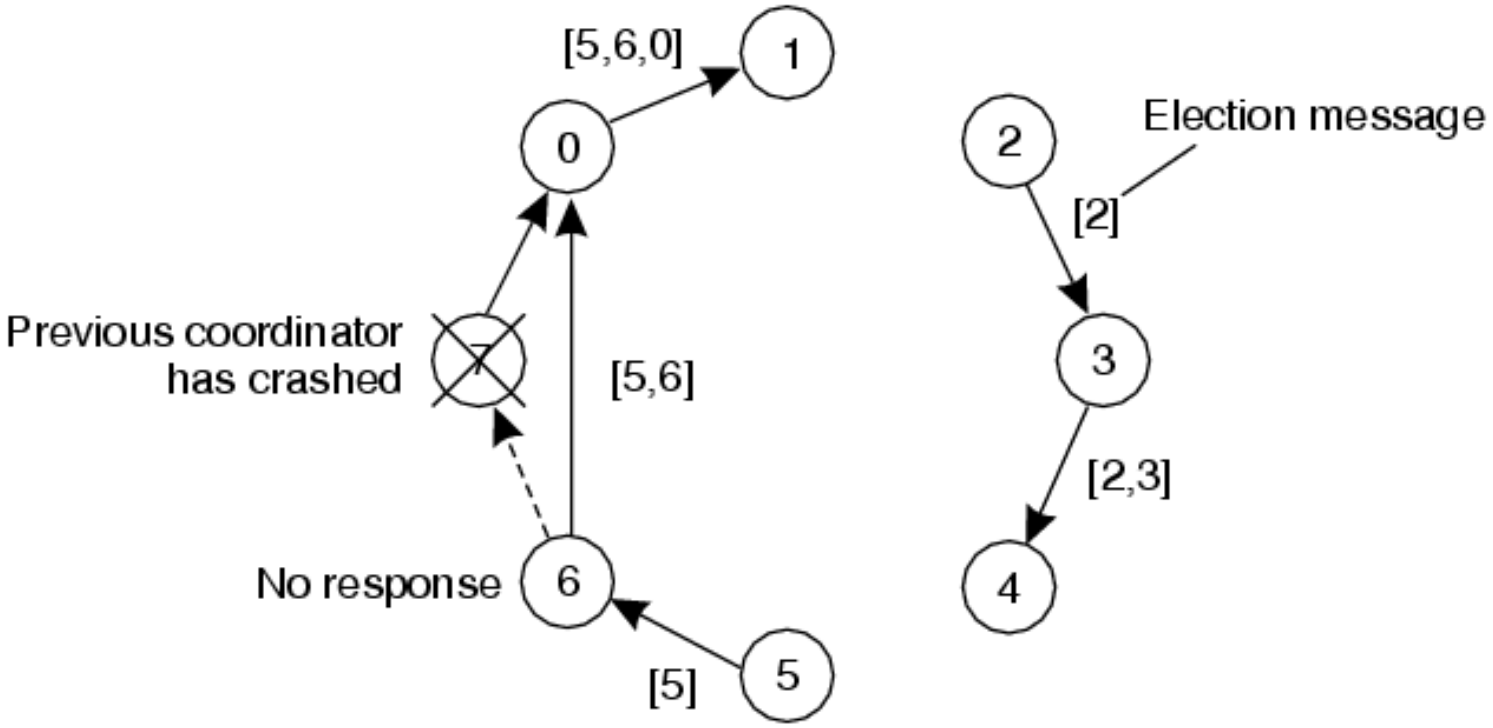
Until …

- If the identifier in the received election message is that of the receiver, that process has just been elected!
- It now sends an elected message round the ring – to tell everyone
- Now each receiver notes the result, passes it on, and resets to non-participating

Evaluation

- How many messages are sent?
- Worst case is when process elected is just before one which initiated election
- Then there are 3n–1 messages
- Toleration of failures: very limited – in principle can rebuild ring, but ….
- Difficult to add new processes to ring too

TVS, Figure 6.21 – two processes initiate election



Bully algorithm
(Garcia-Molina 1982)

- Designed for slightly different situation
- Processes may crash (though messages are assumed reliable)
- A process knows about the identifiers of other processes and can communicate with them
- System is synchronous – it uses timeouts to detect process failure

Three types of message

- An election message – sent to initiate an election
- An answer message – sent in response to an election message
- A coordinator message – sent to announce the identity of the winner of the election

Two timeouts

- If no reply is received in time T, assume that the reply sender has crashed
- If the initiator of an election doesn't learn who won in time T + T' it should begin another election – a process must have crashed in such a way as to mess up this one

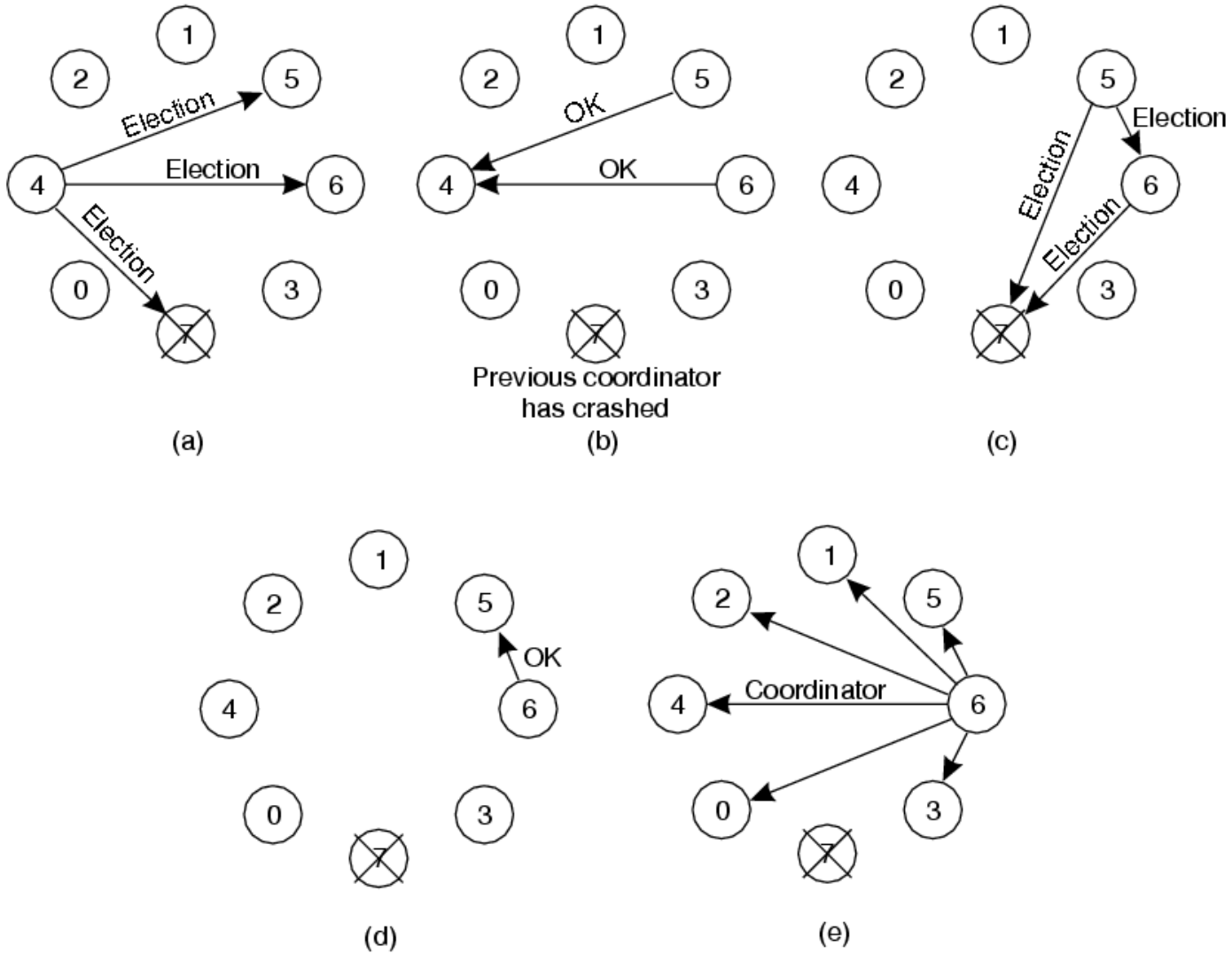
Initiating election

- Process sends an election message to all processes with higher identifiers than itself
- Those that have not crashed will respond with an answer message (within the timeout period) – and the initiating process can then sit back and wait for a coordinator message

Receiving an election message

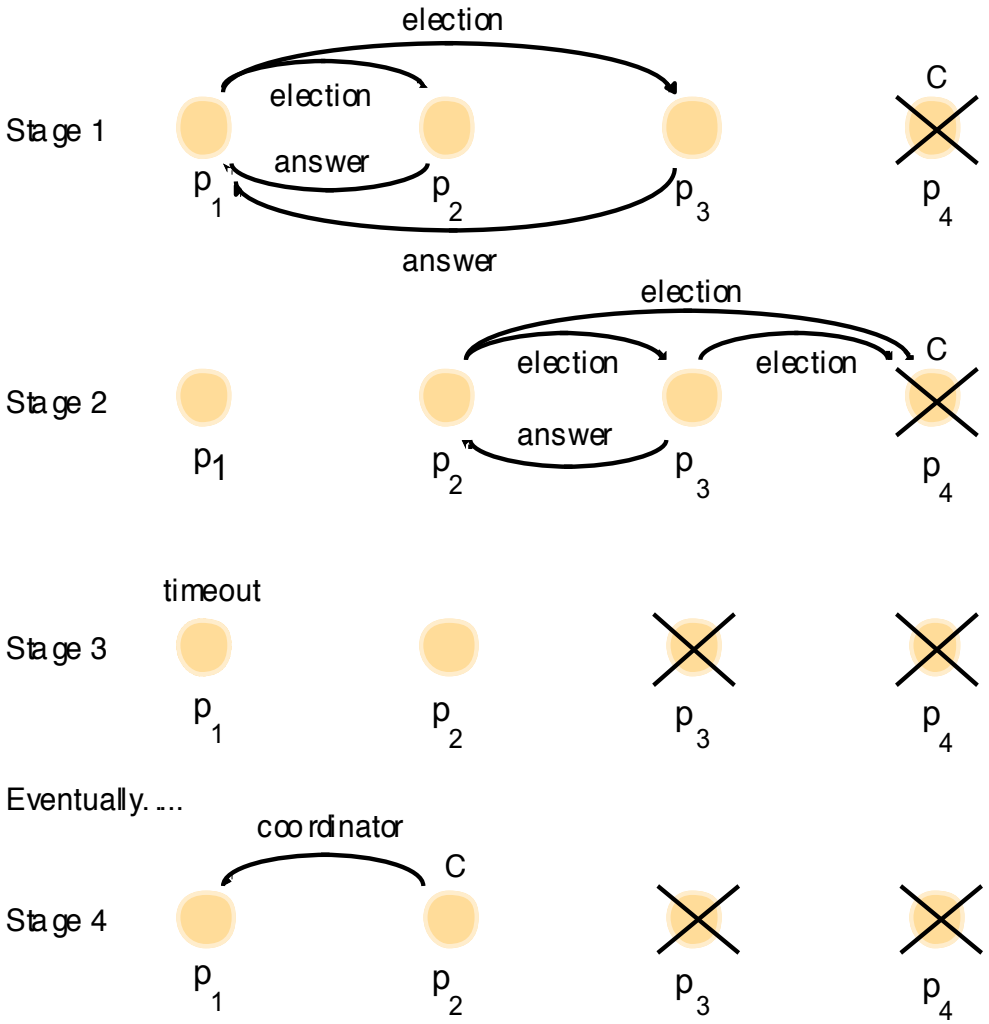
- Any process receiving an election message should send an answer back to the sender
- It should then initiate its own election – by sending an election message to each process with a larger identifier than itself
- Unless it has recently done that, and had neither a coordination reply nor a timeout (T').
- A process with no answering process above it should consider itself elected
- It should then send a coordinator message to all lower processes ….

TVS, Figure 6.20



The bully algorithm (CDK Fig. 12.8)

The election of coordinator p_2, after the failure of p_4 and then p_3



Evaluation

- Need timeouts to be realistic
- Problem if process restarts and elects itself ….
- No of messages depends on which process initiates – the higher the process the smaller the number of messages! If the lowest process initiates, O(n^2) ….
- Exercise: 2015 exam, question 3d

Next: Dealing with failures…