Announcements

- HW2 is on Canvas
  - Due date: Friday, Oct. 21

- Exam I
  - Wednesday, Oct. 12
 - In class

Lamport's Algorithm

- Assumption: messages delivered in FIFO order (no loss messages)
- Requesting the CS
  - P_i sends message **REQUEST(t_i, i)** to other processes, then enqueues the request in its own *request_queue$_i$*
  - when P_j receives a request from P_i, it returns a timestamped **REPLY** to P_i and places the request in *request_queue$_j$*
  - *request_queue* is ordered according to **(t_i, i)**
- A process P_i executes the CS only when:
  - P_i has received a message with timestamp larger than t_i from all other processes
  - its own request in the first of the *request_queue$_i$*

Lamport's Algorithm (2)

- Releasing the critical section:
  - when done, a process remove its request from the queue and sends a timestamped **RELEASE** message to all
  - upon receiving a RELEASE message from P_i, a process removes P_i's request from the request queue

Lamport's Algorithm Example
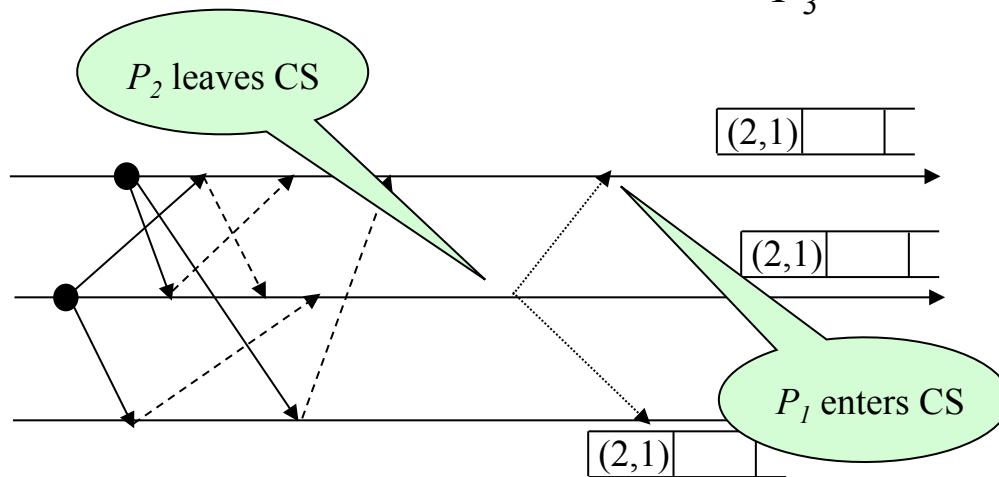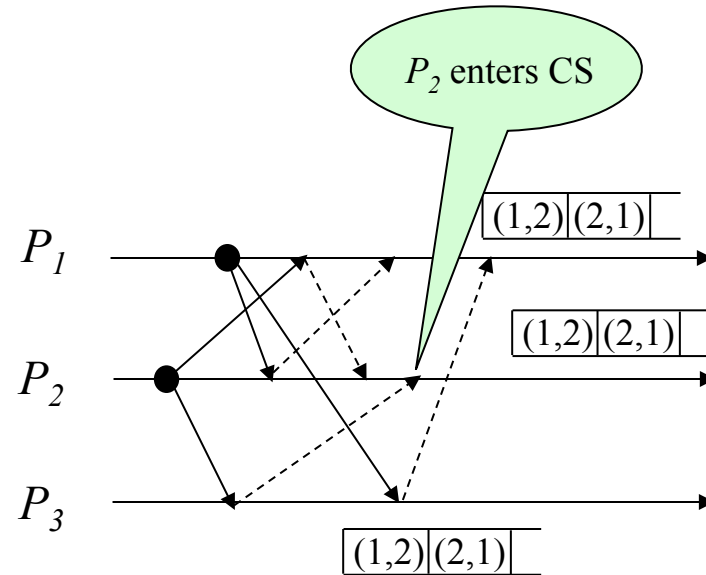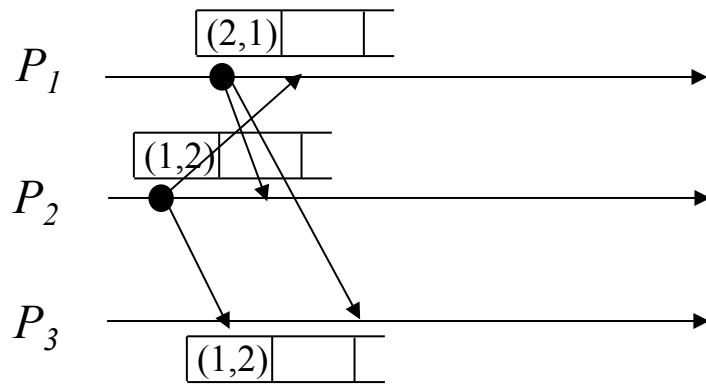
Lamport's: proof of correctness

- Proof by contradiction:
  - assume P_i and P_j are executing the CS at the same time
  - assume request timestamp (t_i, i) of P_i is smaller than that of P_j
  - this means both P_i and P_j have their request at the top of the queue
  - FIFO channels + second assumption + P_j executing => request from P_i must be in $request\_queue_j$
  - contradiction: P_i request in $request\_queue_j$ and not at the top of the queue, however we said timestamp(P_i) < timestamp(P_j) …
- Therefore it cannot be that P_i and P_j are executing the CS at the same time!

About Lamport's Algorithms

- Why FIFO?
- Why Message, not Response?
- Lossy channels work?

- How many messages passed for one CS execution?
  - Can we improve it?

Ricart-Agrawala Algorithm

- ## Optimization of Lamport's algorithm:

Lamport's Algorithm	Ricart-Agrawala Algorithm

Requesting the CS

- P_i sends message **REQUEST(t_i, i)** +
  enqueues the request in *request_queue$_i$*
- when P_i receives a request from P_j, it
  enqueues it and returns a **REPLY** to P_j

Requesting the CS

- P_i sends message **REQUEST(t_i, i)**
- when P_i receives a request from P_j, it returns a **REPLY** to P_j if it is not requesting or executing the CS, or if it made a request but with a larger timestamp. Otherwise, the request is **deferred.**

P_i executes the CS only when:

- has received a msg with timestamp $> t_i$ from everybody
- its own request is the first in the *request_queue$_i$*

P_i executes the CS only when:

- has received a **REPLY** from everybody

Releasing the CS:

- when done, a process remove its request from the queue + sends a timestamped **RELEASE** msg. to everybody else
- upon receiving a RELEASE message from P_i, a process removes P_i's request from its request queue

Releasing the CS:

- when done, a process sends a **REPLY** to all deferred requests

Ricart-Agrawala Algorithm Example
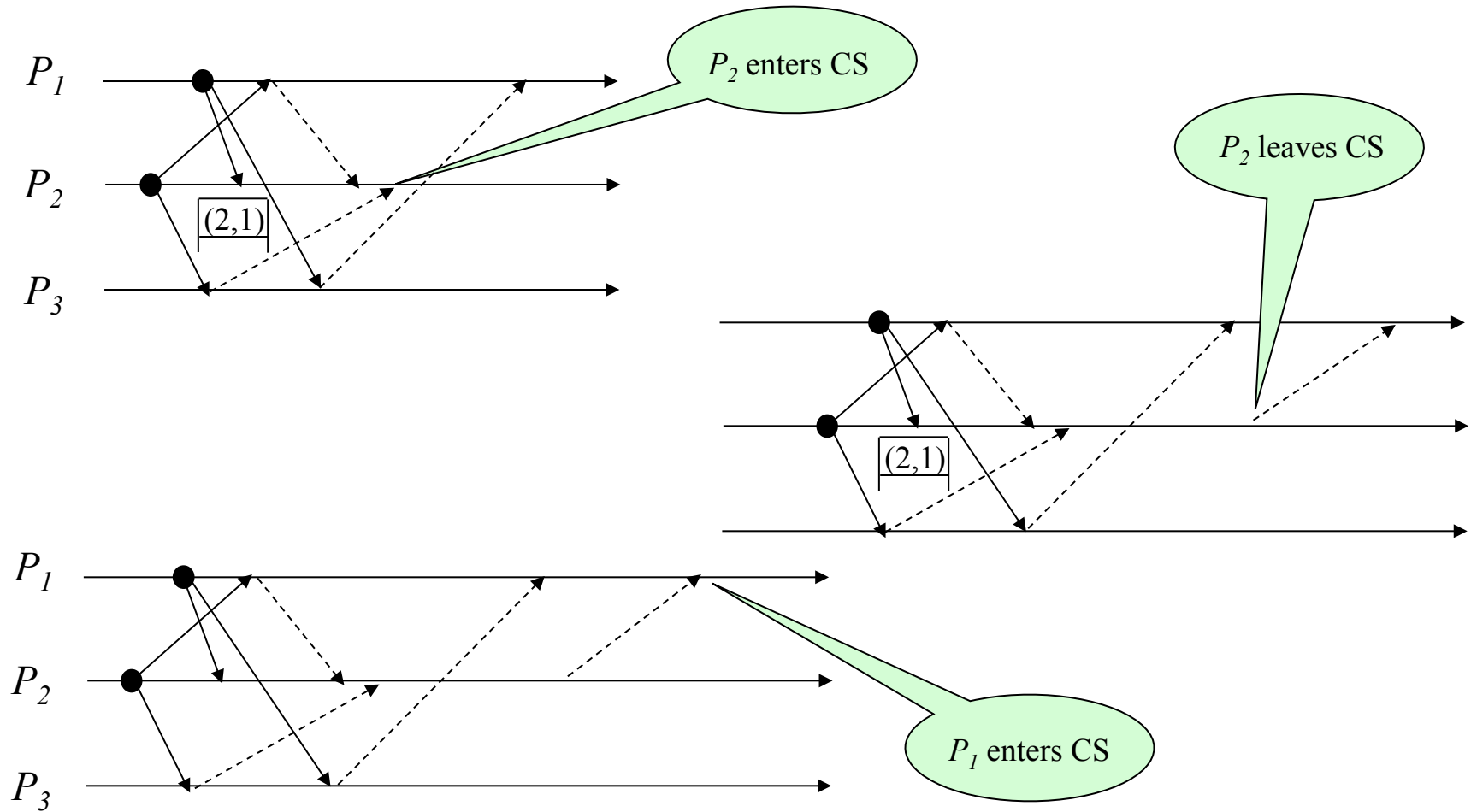
Ricart-Agrawala: proof of correctness

- Proof by contradiction:
  - assume P_i and P_j are executing the CS at the same time
  - assume request timestamp of P_i is smaller than that of P_j
  - this means P_i issued its own request first and then received P_j's request, otherwise P_j request timestamp would be smaller
  - for P_i and P_j to execute the CS concurrently means P_i sent a REPLY to P_j before exiting the CS
  - Contradiction: a process is not allowed to send a REPLY if the timestamp of its request is smaller than the incoming one
- Therefore it cannot be that P_i and P_j are executing the CS at the same time!