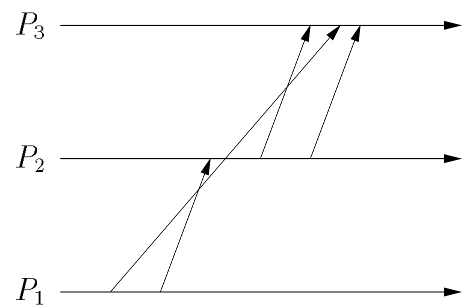
Goals of the lecture

- Hierarchy of communication modes
- Motivation for synchronous ordering
- Crown
- Sufficient conditions for synchronous ordering
- Implementation rules
- Safety theorem
- Liveness

Communication Modes

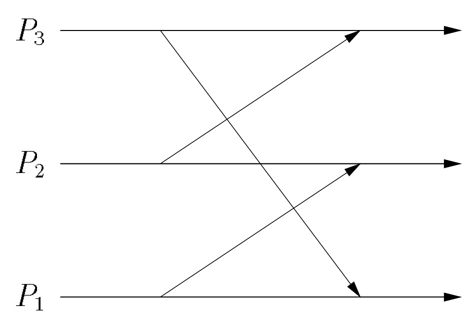
- FIF0



$$s_1 \prec s_2 \implies \neg(r_2 \prec r_1)$$

  – sequence numbers are sufficient to implement FIFO.

- Causally Ordered



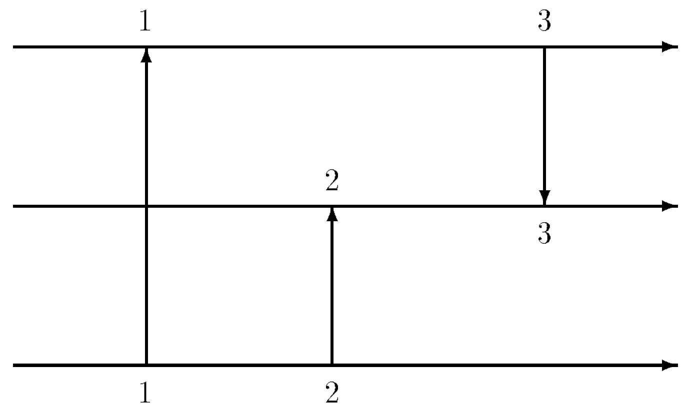
$$s_1 \rightarrow s_2 \implies \neg(r_2 \prec r_1)$$

  – matrix clocks are sufficient to implement Causal ordering.

Communication Modes [Contd.]

- Synchronous Ordering (SYNC)

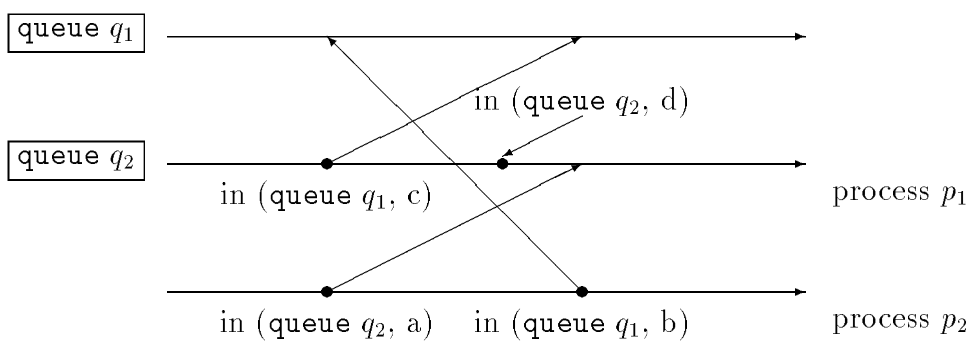
$$\begin{aligned} \exists \mathrm{T} : \mathcal{E} \rightarrow \mathbb{N} \quad &: \quad \forall s, r, e, f \in \mathcal{E} \\ s \leadsto r &\implies \mathrm{T}(s) = \mathrm{T}(r) \\ e \prec f &\implies \mathrm{T}(e) < \mathrm{T}(f) \end{aligned}$$

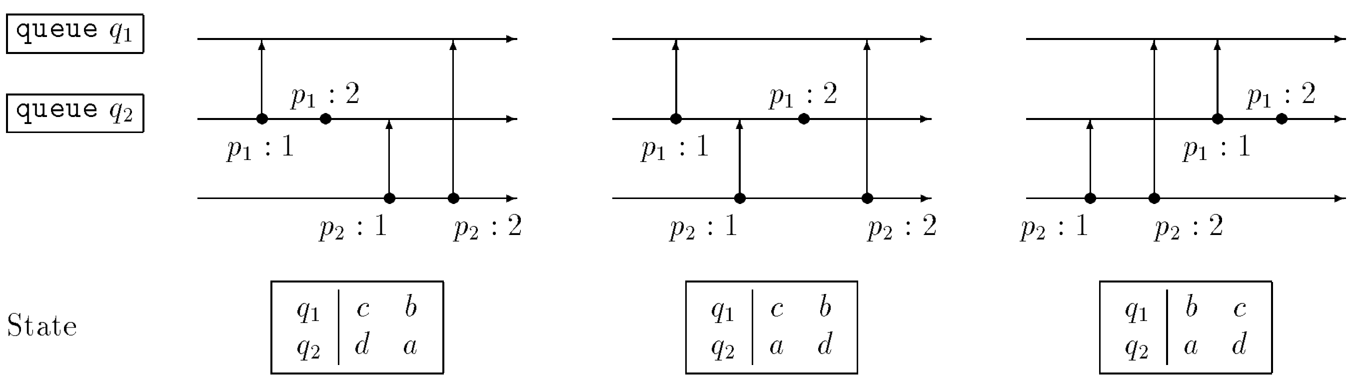


  - time diagram of a synchronous computation can be drawn such that all message arrows are vertical.
  - any order clock is insufficient to implement synchronous ordering.

Motivation for Synch.

p_1	p_2
1: insert (queue q_1, c)	1: insert (queue q_2, a)
2: insert (queue q_2, d)	2: insert (queue q_1, b).





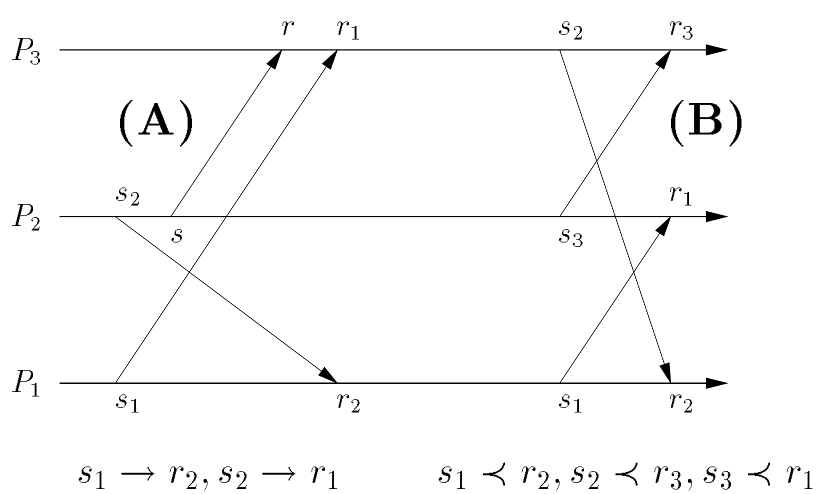
Crowns in Distributed Computation

- A computation is synchronous iff there does not exist a sequence of send and corresponding receive events such that

$$s_0 \rightarrow_{\mathcal{E}} r_1, s_1 \rightarrow_{\mathcal{E}} r_2, \ldots, s_{k-2} \rightarrow_{\mathcal{E}} r_{k-1}, s_{k-1} \rightarrow_{\mathcal{E}} r_0.$$

  – such a structure is called crown.

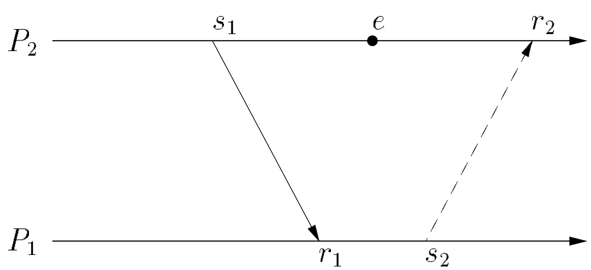
- Example:



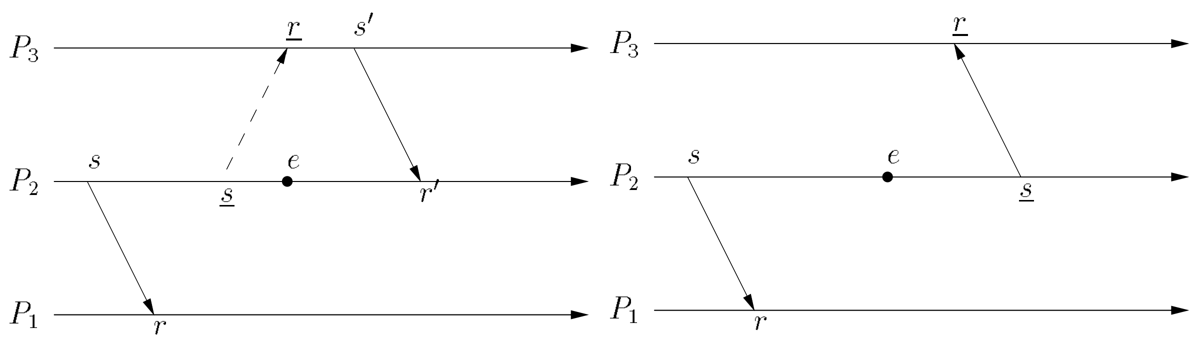
(A) : Crown of size 2
(B) : Strong Crown of size 3

Algorithm

- Commit Point of a Message



- Priority Rule (RP)



(a) The message along with the underlying message (b) The resulting message ordering

Algorithm [Contd.]

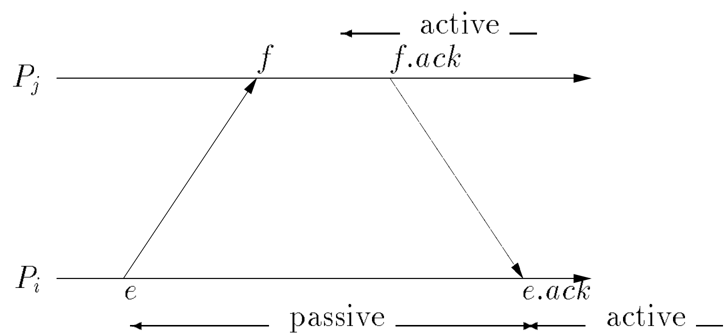
- Send Condition

$$s_1 \prec s_2 \implies r_1 \rightarrow r_2$$

- Receive Condition

$$s_1 \prec s_2 \implies s_1.ack \prec s_2$$

Implementation



- Send Condition

(SC) $s_1 \prec s_2 \implies r_1 \rightarrow r_2$

(SP) $s_1 \prec s_2 \implies s_1.ack \prec s_2$ (Wait for ack)

- Receive Condition

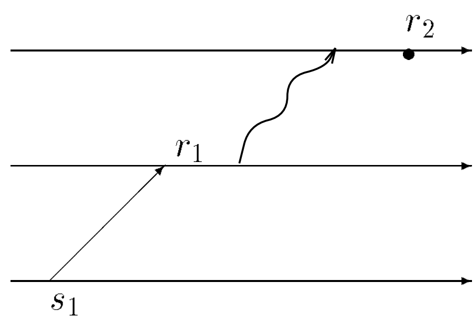
(RC) $s_1 \prec r_2 \implies \neg(r_2 \rightarrow r_1)$

(RP) $s_1 \prec r_2 \implies s_1.ack \prec r_2.ack$ (Send ack if no ack pending)

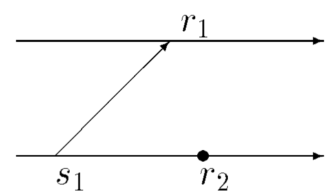
Basic Lemma

Lemma 1 $(s_1 \to r_2)$ *and (SC)* $\Longrightarrow (r_1 \to r_2) \vee (s_1 \prec r_2)$

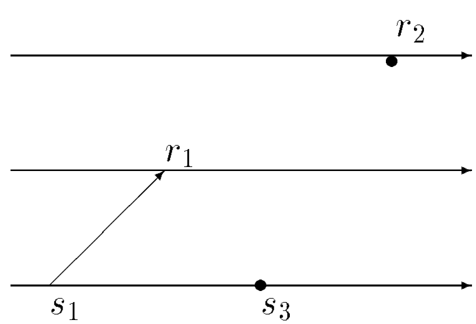
Proof:



Case 1



Case 2



$r_3 \to r_2$

$$s_1 \prec s_3 \Longrightarrow r_1 \to r_3$$
$$\Longrightarrow r_1 \to r_2$$

Case 3

□

Crown and Strong Crown

Lemma 2 *Given SC and RC. 2 CR $\Longrightarrow$ 2 SCR.*

Proof:

$$s_1 \rightarrow r_2, s_2 \rightarrow r_1$$

Let $\neg(s_1 \prec r_2)$

$$\begin{array}{lll} \Longrightarrow & r_1 \rightarrow r_2 & SC \\ \Longrightarrow & \neg(s_2 \prec r_1) & RC \end{array}$$

$$\neg(s_2 \prec r_1) \wedge (s_2 \rightarrow r_1) \Longrightarrow r_2 \rightarrow r_1 \qquad \square$$

Lemma 3 *Given SC, RC. CR(k) $\Longrightarrow$ SCR (k′)*

Proof:

$$s_{i-1} \rightarrow r_i, s_i \rightarrow r_{i+1} \text{ s.t. } \neg(s_i \prec r_{i+1})$$

$$\Longrightarrow \quad r_i \rightarrow r_{i+1}$$

Therefore, $s_{i-1} \rightarrow r_i, s_i \rightarrow r_{i+1}$ reduced to $s_{i-1} \rightarrow r_{i+1}$. $\square$

Safety

$$\neg SYNCH \quad \Leftrightarrow \quad CR \quad \Rightarrow \quad SCR$$

$$s_0 \prec r_1, s_1 \prec r_2, \cdots, s_{k-1} \prec r_0$$

i.e.

$$P(s_i) = P(r_{(i+1) \text{ mod } k})$$

From PR :

$$P(s_i) > P(r_i)$$

we get

$$P(s_0) < P(r_0)$$

Liveness

- If P_k wants to send a message then it can eventually succeed.
- k smallest processes will eventually be in active state.

Overhead

- Priority Rule
  - for every message (s, r) if P$(s) < \mathrm{P}(r)$:
    + one control message and
    + delay of less than $2t$ units of time.
  - for every message (s, r) if P$(s) > \mathrm{P}(r)$:
    + no control message and
    + no delay.

- Send and Receive Protocol
  + one control message and
  + delay is upper bounded by nt, where n is equal to the number of processes.