
Causality and Wave Algorithms

14 April 2010
Lecture 5

Slide credits: R. H. Mak

Topics for Today

1. Causal Ordering and Logical Clocks
2. Wave algorithms
 - Definition, assumptions, general properties
3. Usage
 - Propagation of Information with feedback (PIF)
 - Synchronization (SYN)
 - Distributed function computation (INF)
4. Overview of wave algorithms
 - Ring and polling algorithm
 - Tree algorithm

Source: Tel 2, 6.1-6.2

Causal Order

The *causal (happens before)* relation \prec on the events of an execution E is the smallest relation such that:

1. If e and f are different events of the same process and e happens before f (in E), then $e \prec f$.
2. If s is a send event and r the corresponding receive event (in E), then $s \prec r$.
3. \prec is irreflexive, anti-symmetric, and transitive

Furthermore

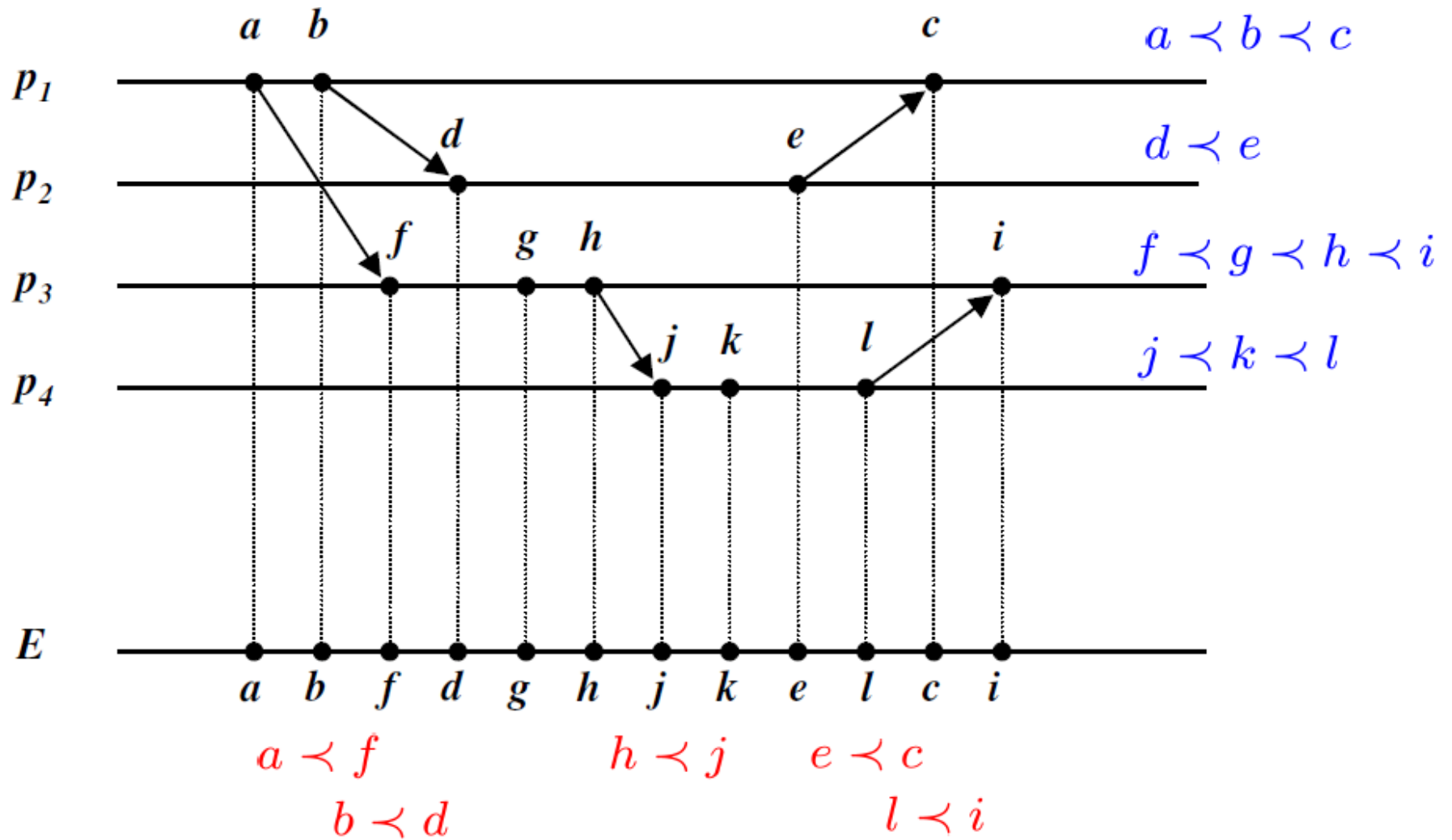
- we write $a \preceq b$ instead of $(a = b \vee a \prec b)$
 - \preceq is a partial order
- We write $a||b$ instead of $\neg(a \prec b \vee b \prec a)$
 - we say that a and b are concurrent

Causality and independence

Thm. Let A be a distributed algorithm with possible events e and f , i.e., there exist executions of A (not necessarily the same) in which e and f are performed.

Then $e // f$ if and only if there exists a reachable configuration γ such that e and f are independent in γ .

Space-time diagram



Consistent Cut

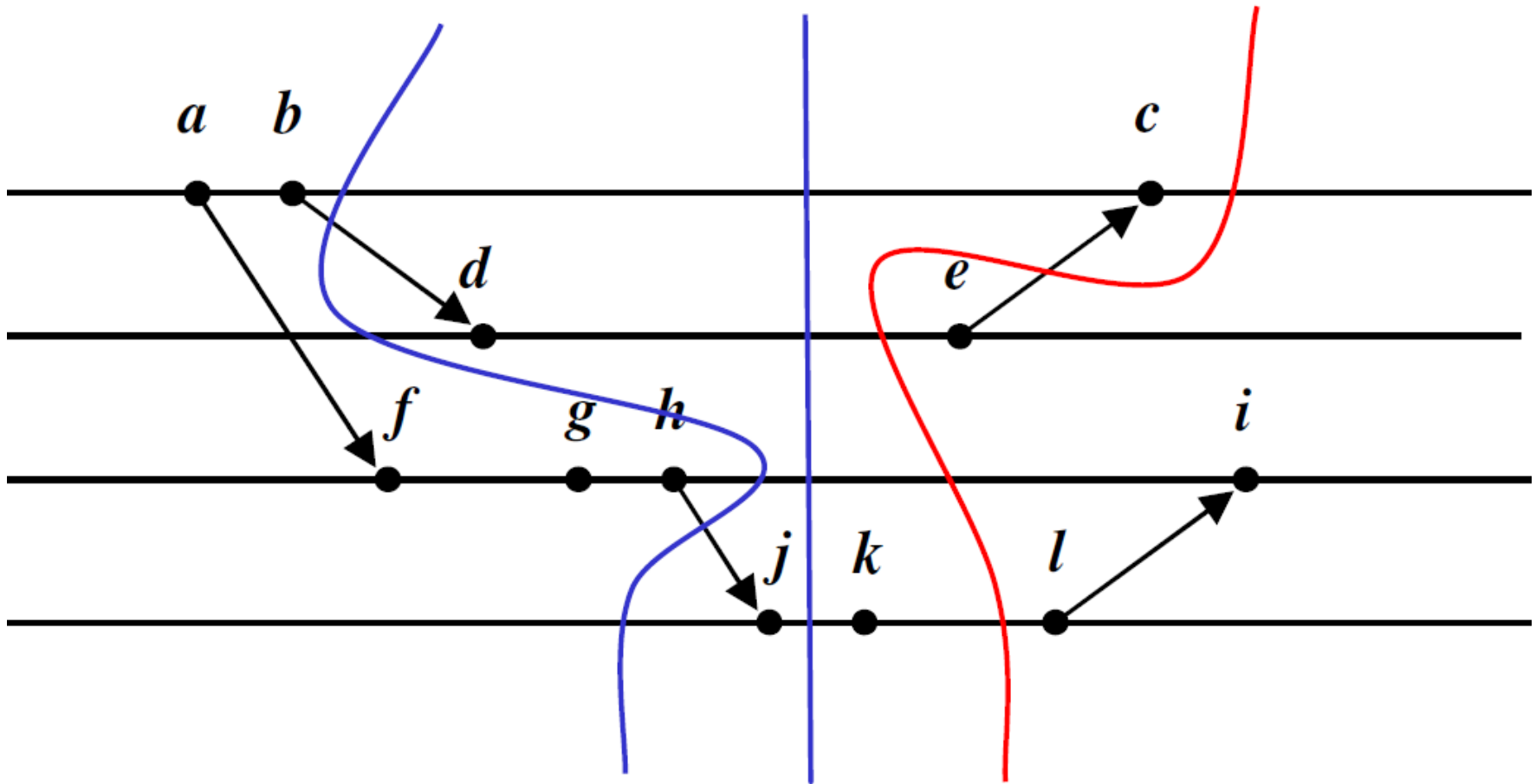
Consider the space-time diagram of an execution E of a distributed algorithm.

A *consistent cut* is a curve that intersects the time axis of all processes (between events) and zero or more communication arrows, each of the latter in such a manner that its send event is to the left of the curve and its receive events to the right.

Prop. Every consistent cut defines configuration that is reachable from the initial configuration of E .

Space-time diagram

consistent



inconsistent

(Logical) clocks

A clock is a function Θ from the set of events to a partially ordered set (X, \leq) that respects the causal order, i.e.,

$$a \prec b \implies \Theta(a) < \Theta(b)$$

1. Order of execution

- For $E = (e_0, e_1, \dots)$ $\Theta_g(e_i) = i$
- Global notion, cannot be computed by any distributed algorithm

2. Real-time clocks

- Induce synchronous mode of cooperation

3. Lamport's logical clock

- Expresses causal order

4. (Mattern's) Vector clock

- Expresses causal order and concurrency
- Concurrency requires X non-total and $a \prec b \iff \Theta(a) < \Theta(b)$

Lamport's logical clock

Lamport's logical clock Θ_L assigns to an event a the length k of the longest sequence

(e_1, \dots, e_k)

of events satisfying

$$e_1 \prec e_2 \prec \dots \prec e_k = a$$

The value $\Theta_L(a)$ of an event a can be computed by a distributed algorithm

```
var  $\Theta_P$ : int  init 0;
```

```
(* an internal event *)
```

```
 $\Theta_P := \Theta_P + 1$ ;
```

```
Change state
```

```
(* A send event *)
```

```
 $\Theta_P := \Theta_P + 1$ ;
```

```
send <msg,  $\Theta_P$ >
```

```
Change state
```

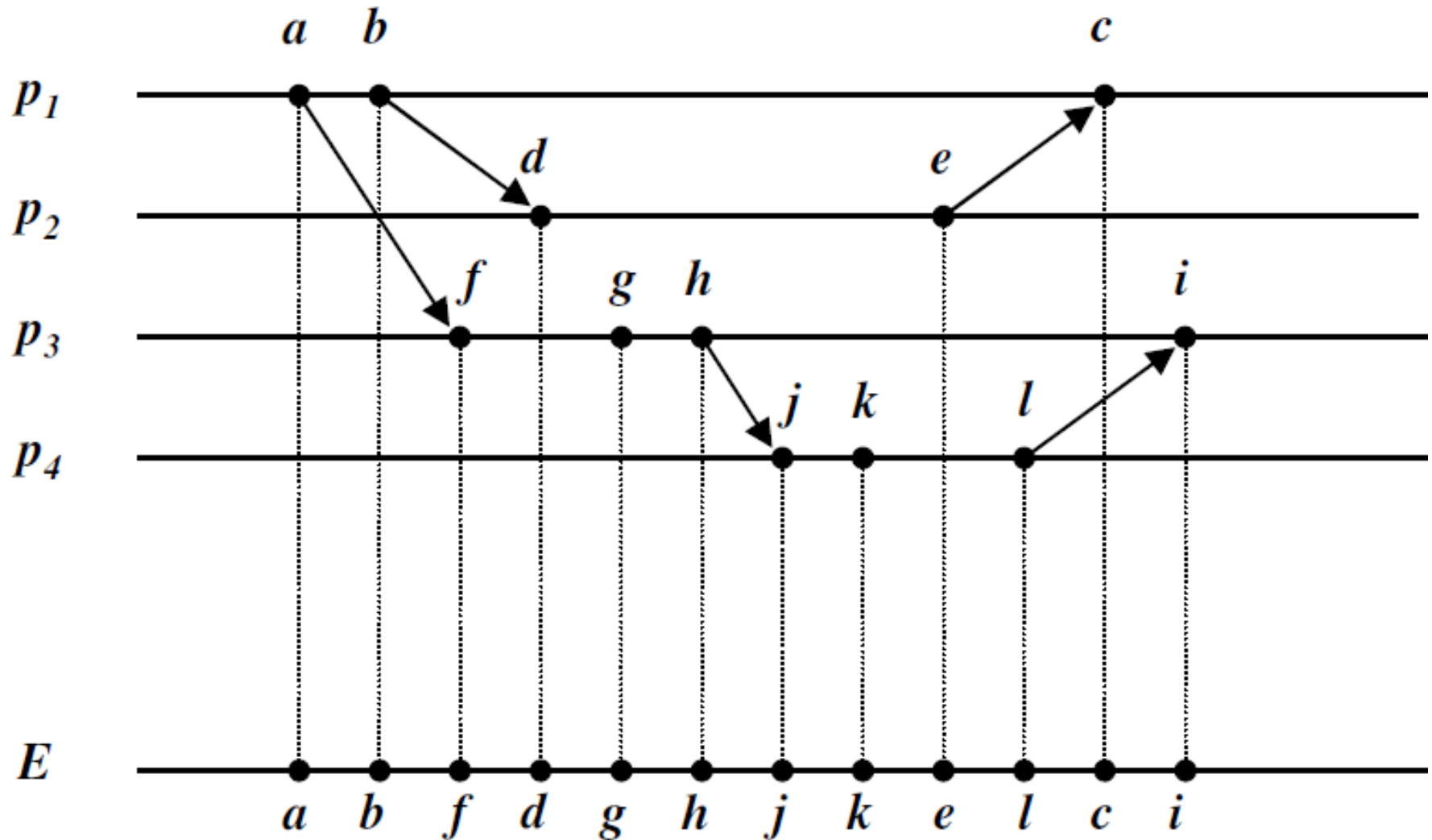
```
(* A receive event *)
```

```
receive <msg,  $\Theta$ >
```

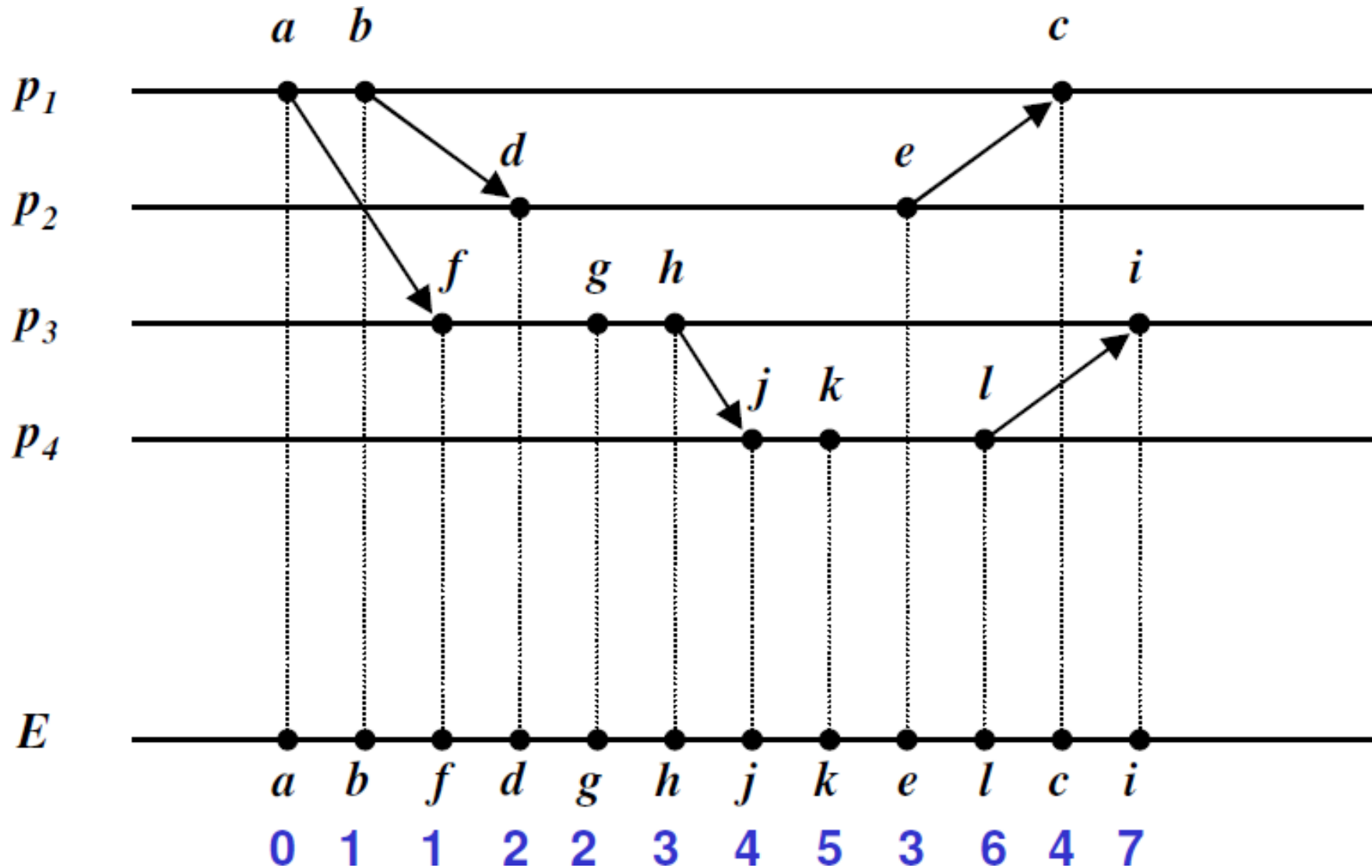
```
 $\Theta_P := \max(\Theta_P, \Theta) + 1$ ;
```

```
Change state
```

Space-time diagram with logical clock



Space-time diagram with logical clock



So Far

1. Causal Ordering and Logical Clocks
2. Wave algorithms
 - Definition, assumptions, general properties
3. Usage
 - Propagation of Information with feedback (PIF)
 - Synchronization (SYN)
 - Distributed function computation (INF)
4. Overview of wave algorithms
 - Ring and polling algorithm
 - Tree algorithm

Wave algorithms (where & why)

Because wave algorithms capture the message passing structure of a distributed algorithm studying them is a form of separation of concerns:

1. **Isolates** network topology
2. **Facilitates the design** of a distributed algorithm
 - **superposition** of computation proper on wave algorithm
 - **generic solution**: the wave algorithm is a subroutine parameter of the distributed algorithm

Wave algorithms (what)

A *wave* algorithm is a distributed algorithm that satisfies the following requirements:

1. Termination

- Each computation is finite

2. Decision

- Each computation contains at least one *decide* event (special internal event, its meaning depends on the application)

3. Dependence

- In each computation every decide event is causally preceded by an event in each process

- A computation of a wave algorithm is called a *wave*.
- An *initiator* (or *starter*) is a process that starts its local algorithm spontaneously, upon an internal trigger.
- A *non-initiator* (or *follower*) starts its local algorithm upon the arrival of a message.

Assumptions of wave algorithms

Unless stated otherwise, the communication network is assumed to be a **static connected** graph, and the channels are assumed to be **bidirectional** and **totally reliable**. Other assumptions are

1. Centralization

- [C | D] centralized, also called *unique initiator*, or decentralized (more than one initiator, possibly all).

2. Network topology

- [A | R | T | C | ...] arbitrary, ring, tree, clique, ...

3. Initial knowledge

- [PI | NI | SoD] process identity, neighbor identity, sense of direction

Computations

- A computation C of a distributed algorithm is a set of events partially ordered by the happens-before \preceq relation.
- The subset of events executed by process $p \in \mathbb{P}$ is denoted by C_p .
- The number of events of computation C is denoted by $|C|$.

Proposition: Assume the asynchronous communication model. In the terminal configuration of a finite computation each process is either properly terminated or blocked on a receive statement.

General Properties

Prop. Each event of a wave algorithm is executed by an initiator or is causally preceded by an event on an initiator.

Prop. Each decide event of a wave algorithm is causally preceded by a send event on all processes other than the one that executes the decide event.

Prop. Let C be a wave algorithm with unique initiator p . For each follower q let $father_q$ be the process from which q received a message in its first event. Then the graph

$$T = (P, \{qr \mid q \neq p \wedge r = father_q\})$$

is a rooted spanning tree directed towards root p .

Usage of wave algorithms

1. Propagation of information with feedback
 - Broadcast or wake-up with termination notification
2. Synchronization
 - E.g. barrier synchronization
3. Distributed function evaluation
 - In particular infimum computations
4. Traversal
 - Applied for access permission to avoid collisions on a shared transmission medium; mutual exclusion
5. Depth-first search

PIF Algorithms

Let Q be a subset of the processes of \mathbb{P} that share a common message M , which must be broadcasted to all processes in \mathbb{P} . Moreover, certain processes of \mathbb{P} must be notified of the termination of the broadcast, *i.e.* they must execute a special notify event with the requirement that a notify event may only be executed when all processes have received the message M .

A distributed algorithm that realizes the above computation, while exchanging only finitely many messages is a PIF algorithm.

Thm. Every PIF algorithm is a wave algorithm.

Thm. Every wave algorithm can be employed as a PIF algorithm

SYN algorithms

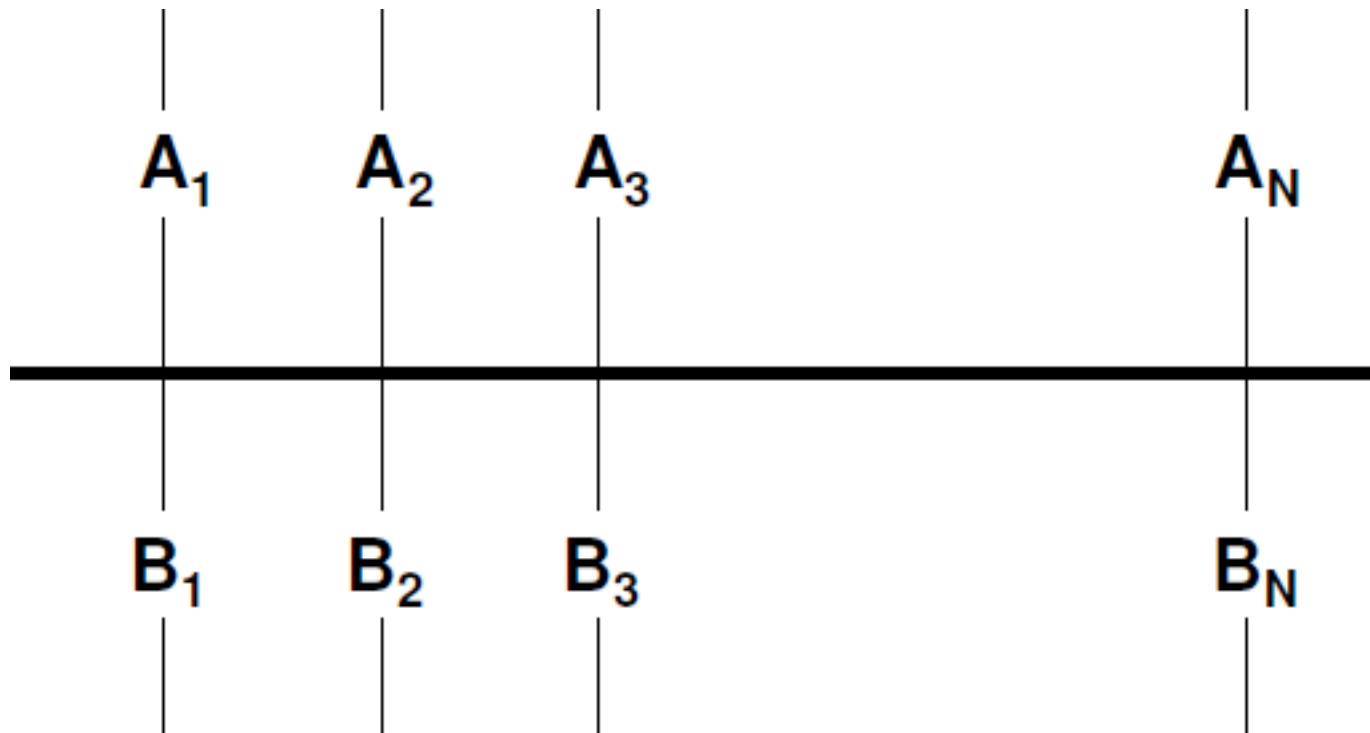
Let \mathbb{P} be a set of processes, where every process $p \in \mathbb{P}$ must execute an internal event a_p and some processes must execute a second event b_p , such that the execution of every a_p event occurs before the execution of any b_p event.

A distributed algorithm that realizes the above computation, while exchanging only finitely many messages is a SYN algorithm.

Thm. Every SYN algorithm is a wave algorithm.

Thm. Every wave algorithm can be employed as a SYN algorithm.

Barrier Synchronization



INF Algorithms

Infimum algorithms are representative of a class of algorithms whose value depends essentially on the input of every process.

Definition Let (X, \leq) be a partial order. Then c is the *infimum* of a and b if:

$$c \leq a, c \leq b, \text{ and } \forall d : (d \leq a \wedge d \leq b \implies d \leq c)$$

We denote the *infimum* c as $a \wedge b$.

Properties \wedge is a binary operator and is:

- commutative (*i.e.* $a \wedge b = b \wedge a$)
- associative (*i.e.* $a \wedge (b \wedge c) = (a \wedge b) \wedge c$).

Application to sets We can generalize \wedge to finite sets as:

$$\inf\{j_1, \dots, j_k\} = j_1 \wedge \dots \wedge j_k$$

Computing infimum In the infimum algorithm (INF), each process q has an input j_q which is an element of a partially ordered set X . Certain processes must compute the value of $\inf\{j_q : q \in \mathbb{P}\}$ and that these processes know when the computation is terminated. They *write* the outcome of the computation as a variable *out* (the *decision*) and are not allowed to change this variable afterwards.

Thm. Every INF algorithm is a wave algorithm.

Thm. Every wave algorithm can be employed as a INF algorithm

Wave algorithms overview

Algorithm	Attributes			Complexity	
	<u>Topology</u>	<u>Centralized?</u>	<u>Knowledge</u>	<u>Message</u>	<u>Time</u>
Ring	R	C	SoD	$ V $	$ V $
Tree	T	D	NI	$ V $	$O(D)$
Echo	A	C	NI	$2 E $	$O(V)$
Polling	C	C	NI	$2 V -2$	2
Phase	A	D	NI	$2D* E $	$2D$
Clique-Phase	C	D	PI, NI	$ V *(V -1)$	2
Finn	A	D	NI	$\leq 4 V * E $	$O(D)$

Ring Algorithm

var $Next_p$: node

{p is an initiator}

begin send <tok> to $Next_p$;

 receive <tok>;

decide

end

{p is not an initiator}

begin receive <tok>;

 send <tok> to $Next_p$;

end

Polling Algorithm

For networks with at least one node v such that $\deg(v) = |V|-1$. One of these nodes must be the unique initiator.

var rec_p : int **init** 0; (* for the initiator only *)

If p is initiator **then**

begin

forall $q \in Neigh_p$ **do** send **<tok>** to q ;

while $rec_p < \#Neigh_p$ **do**

begin receive **<tok>**; $rec_p := rec_p + 1$ **end**;

decide

end

else (* bounce the token *)

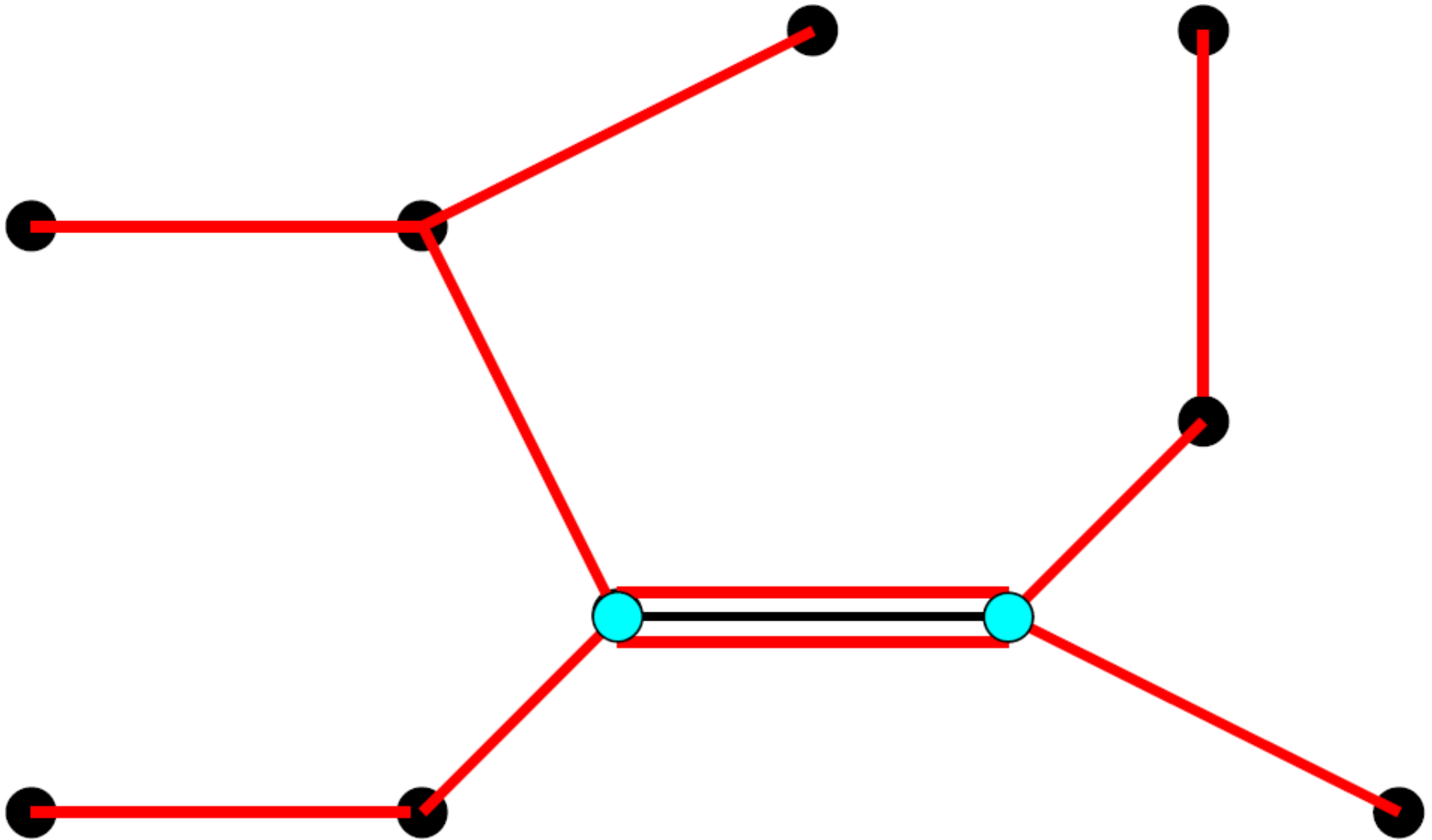
begin receive **<tok>** from q ; send **<tok>** to q **end**

Tree Algorithm

var $rec_p[q]$ for each $q \in Neigh_p$: boolean **init** *false*
(* $rec_p[q]$ is true if p has received a message from q *)

begin while $\#\{q : rec_p[q] = false\} > 1$ **do**
 begin L1: receive **<tok>** from q ;
 $rec_p[q] := true$
 end;
 send **<tok>** to q_0 with $rec_p[q_0] = false$;
 L2: receive **<tok>** from q_0 ; $rec_p[q_0] := true$;
 decide
 (* Inform other processes of decision:
 forall $q \in Neigh_p$ AND $q \neq q_0$ **do** send **<tok>** to q *)
end

Tree Algorithm



Conclusion

1. Causal Ordering and Logical Clocks
2. Wave algorithms
 - Definition, assumptions, general properties
3. Usage
 - Propagation of Information with feedback (PIF)
 - Synchronization (SYN)
 - Distributed function computation (INF)
4. Overview of wave algorithms
 - Ring and polling algorithm
 - Tree algorithm