Distributed Operating Systems: Theoretical Foundations

Neeraj Mittal

Department of Computer Science

The University of Texas at Dallas

neerajm@utdallas.edu

Two Important Characteristics

Absence of Global Clock

there is no common notion of time

Inherent Limitations

Two Important Characteristics
 Absence of Global Clock

Absence of Shared Memory

Ordering of Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Two Important Characteristics

Absence of Global Clock

- there is no common notion of time
- Absence of Shared Memory
 - no process has up-to-date knowledge about the system

Inherent Limitations

Two Important Characteristics
 Absence of Global Clock

Absence of Shared Memory

Ordering of Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Different processes may have different notions of time

Inherent Limitations

Two Important Characteristics

Absence of Global Clock

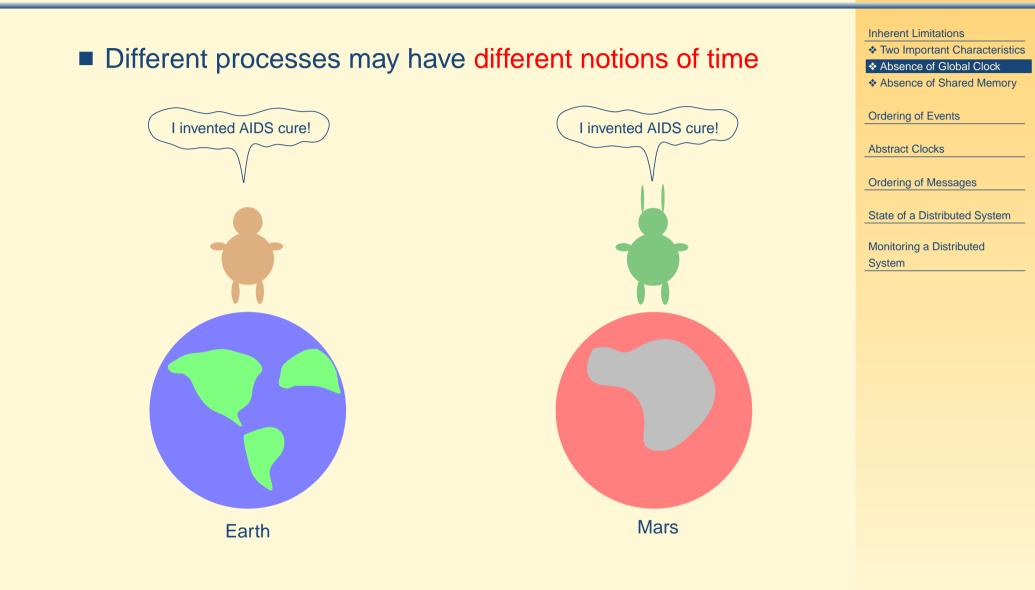
Absence of Shared Memory

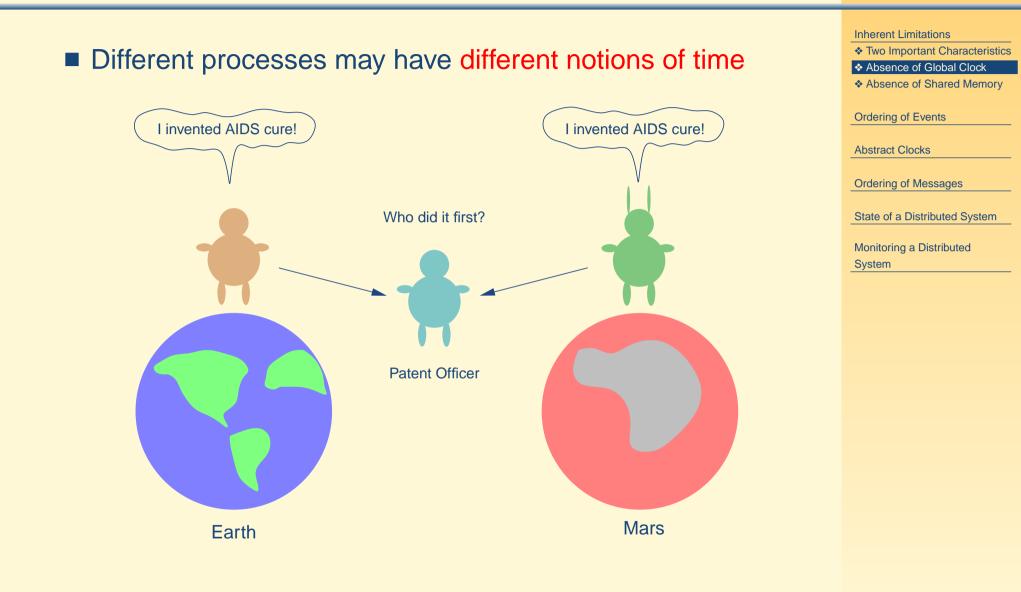
Ordering of Events

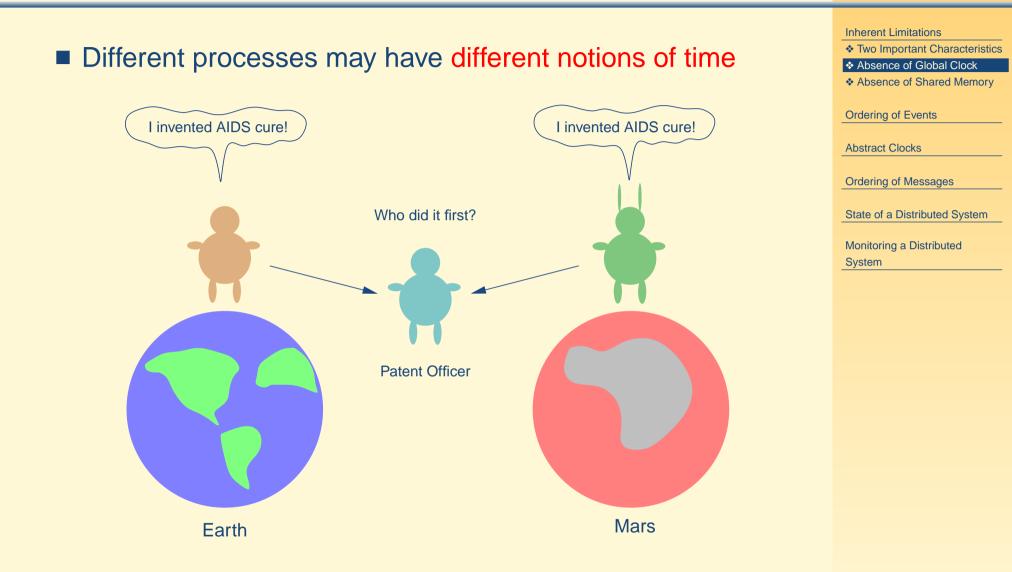
Abstract Clocks

Ordering of Messages

State of a Distributed System







Problem: How do we order events on different processes?

A process does not know current state of other processes

Inherent Limitations

Two Important Characteristics
Absence of Global Clock

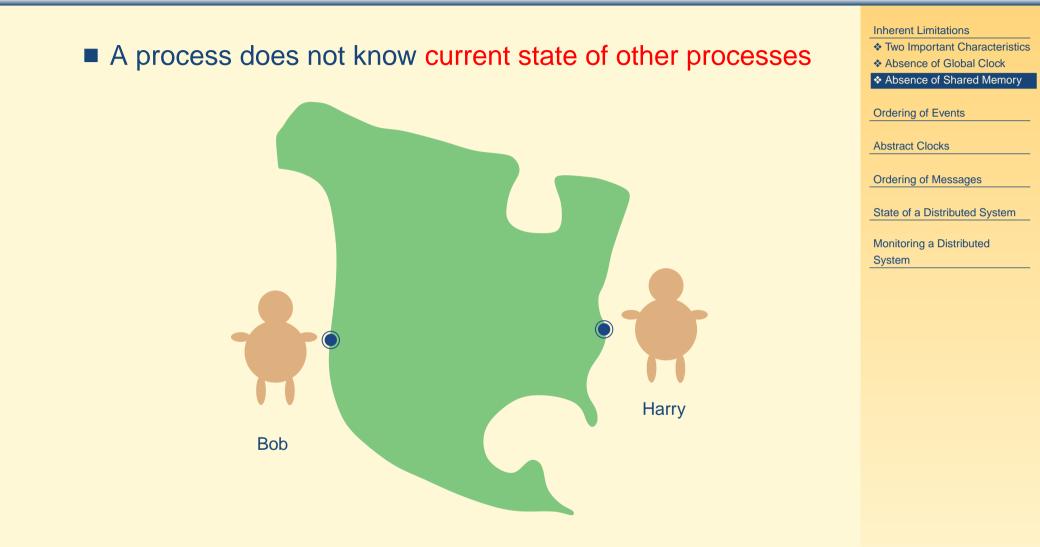
Absence of Shared Memory

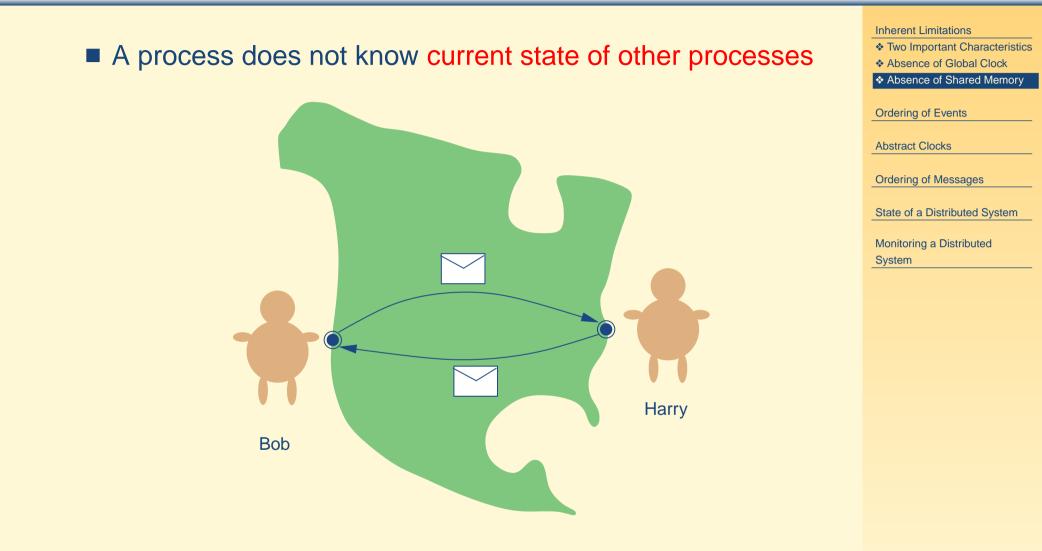
Ordering of Events

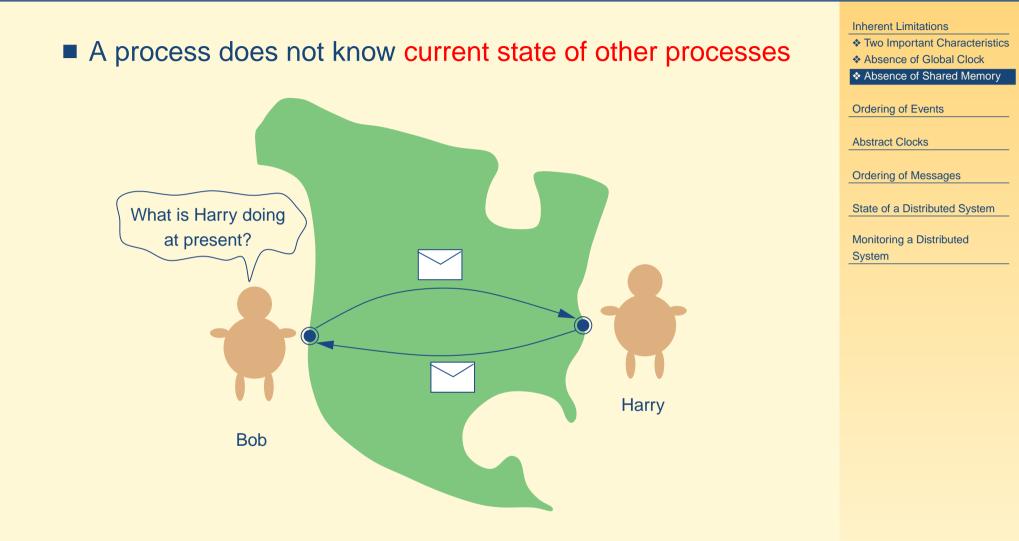
Abstract Clocks

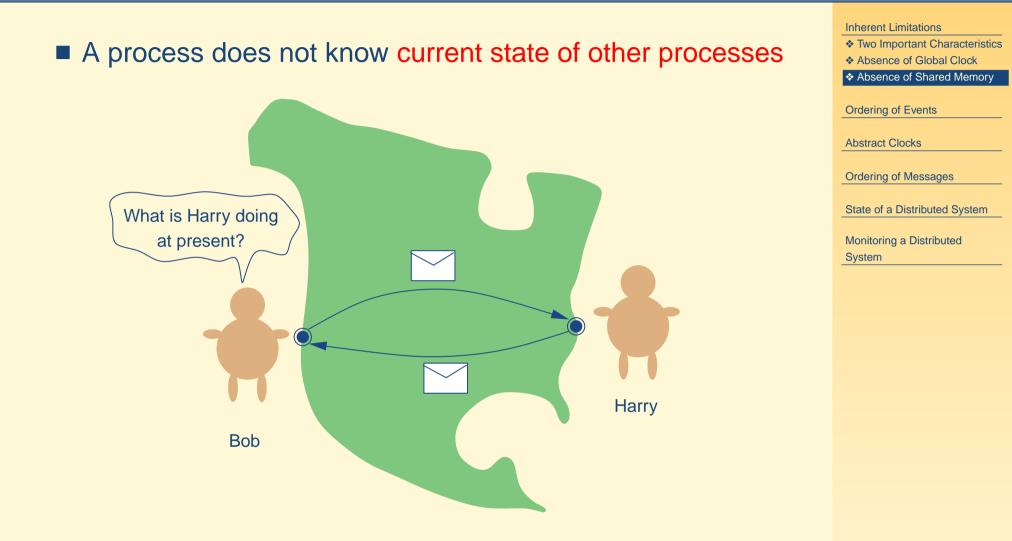
Ordering of Messages

State of a Distributed System









Problem: How do we obtain a coherent view of the system?

Three cases:

1. Events executed on the same process:

Inherent Limitations

Ordering of Events

 When is it possible to order two events?
 Happened-Before Relation

Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Three cases:

- 1. Events executed on the same process:
 - if e and f are events on the same process and e occurred before f, then e happened-before f

Inherent Limitations

Ordering of Events
When is it possible to order two events?
Happened-Before Relation
Concurrent Events

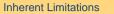
Abstract Clocks

Ordering of Messages

State of a Distributed System

Three cases:

- 1. Events executed on the same process:
 - if e and f are events on the same process and e occurred before f, then e happened-before f
- 2. Communication events of the same message:



Ordering of Events

When is it possible to order two events?
Happened-Before Relation
Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Three cases:

- 1. Events executed on the same process:
 - if e and f are events on the same process and e occurred before f, then e happened-before f
- 2. Communication events of the same message:
 - if e is the send event of a message and f is the receive event of the same message, then e happened-before f

Inherent Limitations

Ordering of Events
When is it possible to order two events?
Happened-Before Relation
Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Three cases:

- 1. Events executed on the same process:
 - if e and f are events on the same process and e occurred before f, then e happened-before f
- 2. Communication events of the same message:
 - if e is the send event of a message and f is the receive event of the same message, then e happened-before f
- 3. Events related by transitivity:

Inherent Limitations

Ordering of Events
♦ When is it possible to order two events?
♦ Happened-Before Relation
♦ Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Three cases:

- 1. Events executed on the same process:
 - if e and f are events on the same process and e occurred before f, then e happened-before f
- 2. Communication events of the same message:
 - if e is the send event of a message and f is the receive event of the same message, then e happened-before f
- 3. Events related by transitivity:
 - if event e happened-before event g and event g happened-before event f, then e happened-before f

Inherent Limitations

Ordering of Events
♦ When is it possible to order two events?
♦ Happened-Before Relation
♦ Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Happened-before relation is denoted by \rightarrow

Inherent Limitations

Ordering of Events When is it possible to order two events? Happened-Before Relation

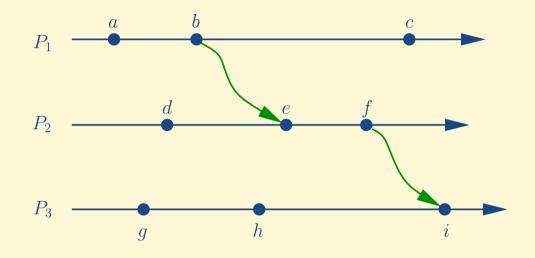
Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

■ Happened-before relation is denoted by →
 ■ Illustration:



Inherent Limitations

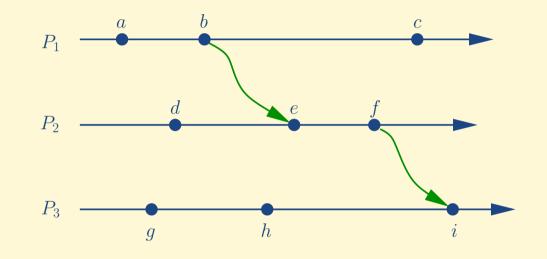
Ordering of Events When is it possible to order two events? Happened-Before Relation Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

■ Happened-before relation is denoted by →
 ■ Illustration:



• Events on the same process: examples: $a \rightarrow b, a \rightarrow c, d \rightarrow f$

Inherent Limitations

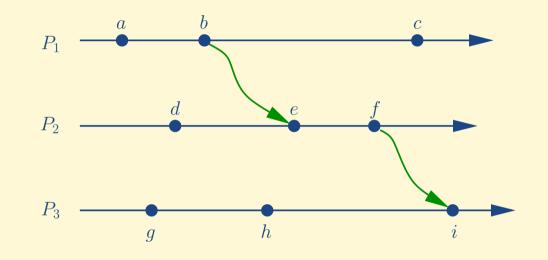
Ordering of Events When is it possible to order two events? Happened-Before Relation Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

■ Happened-before relation is denoted by →
 ■ Illustration:



- Events on the same process: examples: $a \rightarrow b$, $a \rightarrow c$, $d \rightarrow f$
- Events of the same message: examples: $b \rightarrow e, f \rightarrow i$

Inherent Limitations

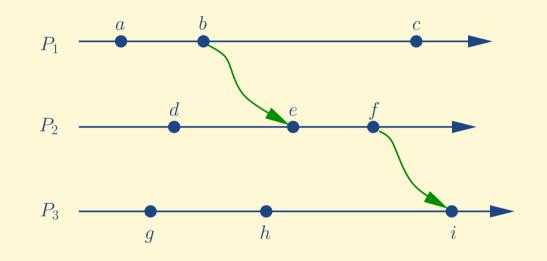
Ordering of Events When is it possible to order two events? Happened-Before Relation Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

■ Happened-before relation is denoted by →
 ■ Illustration:



- Events on the same process: examples: $a \rightarrow b$, $a \rightarrow c$, $d \rightarrow f$
- Events of the same message: examples: $b \rightarrow e, f \rightarrow i$
- Transitivity: examples: $a \rightarrow e, a \rightarrow i, e \rightarrow i$

Inherent Limitations

Ordering of Events When is it possible to order two events? Happened-Before Relation Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Events not related by happened-before relation

Inherent Limitations

Ordering of Events When is it possible to order two events?

Happened-Before Relation

Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Events not related by happened-before relation
Concurrency relation is denoted by ||

Inherent Limitations

Ordering of Events

When is it possible to order two events?
Happened-Before Relation

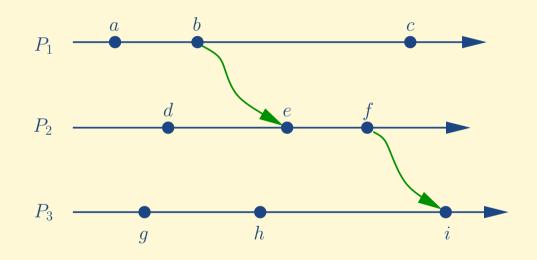
Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

- Events not related by happened-before relation
- Concurrency relation is denoted by
- Illustration:



Inherent Limitations

Ordering of Events When is it possible to order

two events? Happened-Before Relation

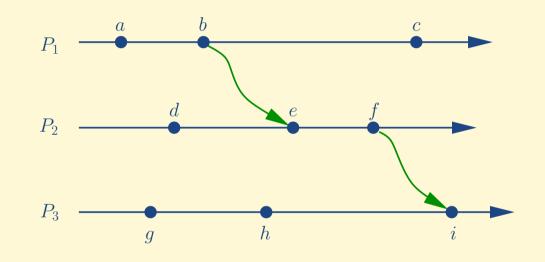
Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

- Events not related by happened-before relation
- Concurrency relation is denoted by
- Illustration:



• Examples: $a \parallel d$, $d \parallel h$, $c \parallel e$

Inherent Limitations

Ordering of Events When is it possible to order

two events? Happened-Before Relation

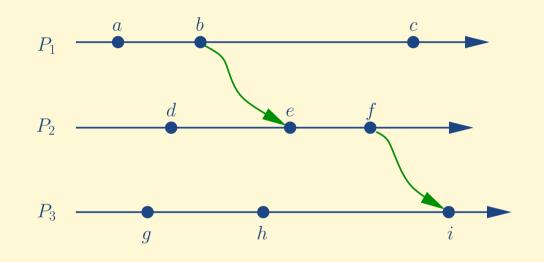
Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

- Events not related by happened-before relation
- Concurrency relation is denoted by
- Illustration:



• Examples: $a \parallel d, d \parallel h, c \parallel e$

Concurrency relation is not transitive: example: $a \parallel d$ and $d \parallel c$ but $a \not\parallel c$

Inherent Limitations

Ordering of Events

When is it possible to order two events?
Happened-Before Relation

Concurrent Events

Abstract Clocks

Ordering of Messages

State of a Distributed System

Logical Clocks

used to totally order all events

Inherent Limitations

Ordering of Events

Abstract Clocks

Different Kinds of Clocks

Logical Clock

- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Logical Clocks

used to totally order all events

Vector Clocks

used to track happened-before relation

Inherent Limitations

Ordering of Events

Abstract Clocks

Different Kinds of Clocks

Logical Clock

- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Logical Clocks

used to totally order all events

Vector Clocks

- used to track happened-before relation
- Matrix Clocks
 - used to track what other processes know about other processes

Inherent Limitations

Ordering of Events

Abstract Clocks

Different Kinds of Clocks

Logical Clock

- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Logical Clocks

used to totally order all events

Vector Clocks

used to track happened-before relation

Matrix Clocks

 used to track what other processes know about other processes

Direct Dependency Clocks

used to track direct causal dependencies

Inherent Limitations

Ordering of Events

Abstract Clocks

Different Kinds of Clocks

Logical Clock

- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

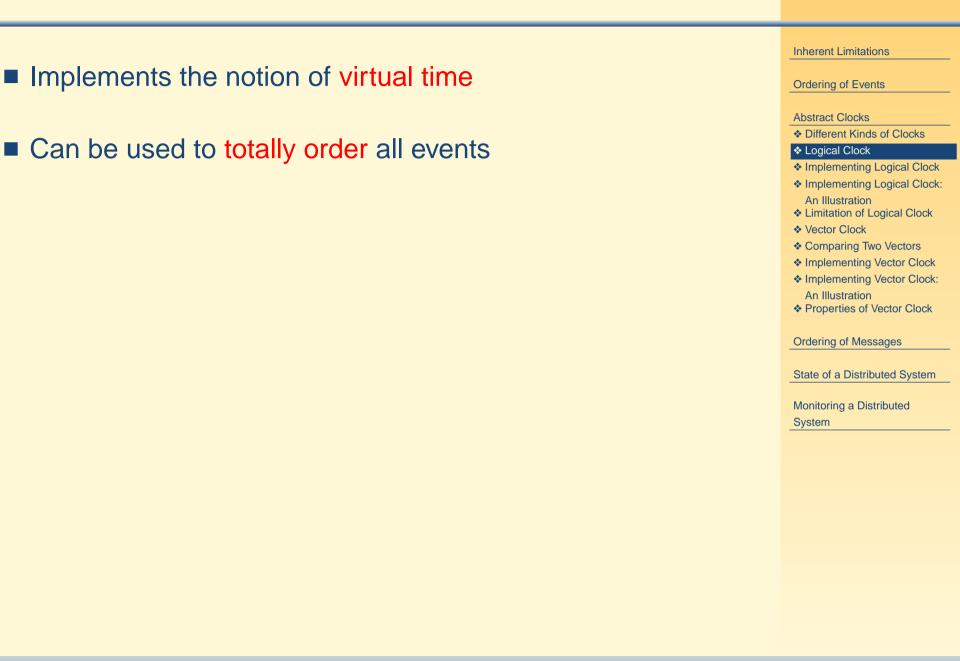
Ordering of Messages

State of a Distributed System

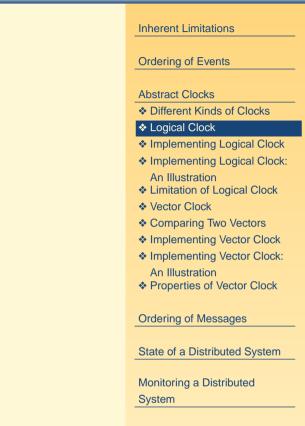
Logical Clock

Inherent Limitations Implements the notion of virtual time Ordering of Events Abstract Clocks Different Kinds of Clocks Logical Clock Implementing Logical Clock Implementing Logical Clock: An Illustration Limitation of Logical Clock Vector Clock Comparing Two Vectors Implementing Vector Clock Implementing Vector Clock: An Illustration Properties of Vector Clock Ordering of Messages State of a Distributed System Monitoring a Distributed System

Logical Clock



Logical Clock



Implements the notion of virtual time

- Can be used to totally order all events
- Assigns timestamp to each event in a way that is consistent with the happened-before relation:

 $e \to f \Rightarrow C(e) < C(f)$

C(e): timestamp for event eC(f): timestamp for event f

Implementing Logical Clock

Each process has a local scalar clock, initialized to zero

• C_i denotes the local clock of process P_i

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock

Implementing Logical Clock

- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- Each process has a local scalar clock, initialized to zero
 - C_i denotes the local clock of process P_i
- Action depends on the type of the event:

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock

Implementing Logical Clock

- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Inherent Limitations Each process has a local scalar clock, initialized to zero Ordering of Events • C_i denotes the local clock of process P_i Abstract Clocks Different Kinds of Clocks Logical Clock Implementing Logical Clock Action depends on the type of the event: Implementing Logical Clock: An Illustration Limitation of Logical Clock Vector Clock Comparing Two Vectors • Protocol for process P_i : Implementing Vector Clock Implementing Vector Clock: An Illustration Properties of Vector Clock Ordering of Messages State of a Distributed System Monitoring a Distributed System

Each process has a local scalar clock, initialized to zero	Inherent Limitations Ordering of Events
• C_i denotes the local clock of process P_i	Abstract Clocks
Action depends on the type of the event:	 Logical Clock Implementing Logical Clock Implementing Logical Clock: An Illustration Limitation of Logical Clock
Protocol for process P_i :	 Vector Clock Comparing Two Vectors Implementing Vector Clock Implementing Vector Clock: An Illustration
 On executing an interval event: 	 Properties of Vector Clock Ordering of Messages
$C_i := C_i + 1$	State of a Distributed System
	Monitoring a Distributed System

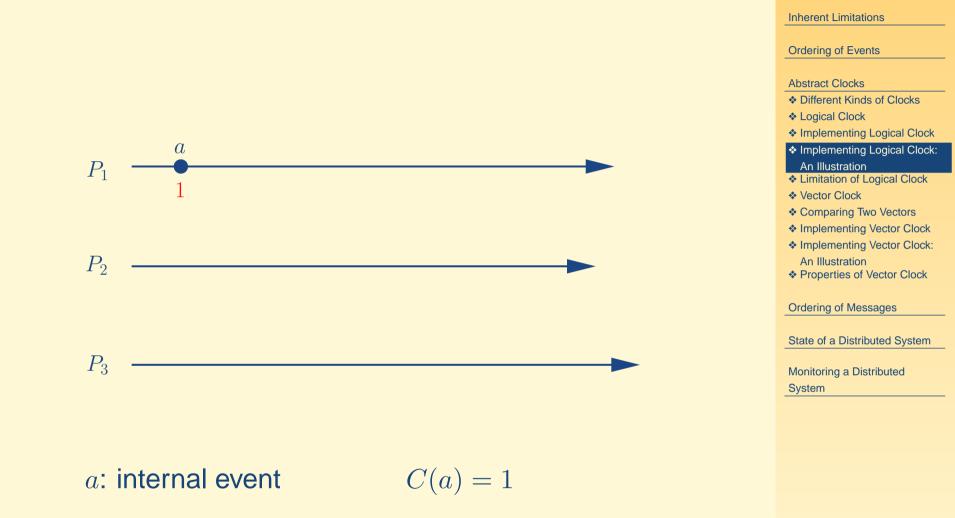
	Inherent Limitations
Each process has a local scalar clock, initialized to zero	Ordering of Events
• C_i denotes the local clock of process P_i	Abstract Clocks
	 Different Kinds of Clocks Logical Clock
Action depende on the type of the events	 Implementing Logical Clock Implementing Logical Clock:
Action depends on the type of the event:	An Illustration
	♦ Vector Clock
• Protocol for process P_i :	 Comparing Two Vectors Implementing Vector Clock
	Implementing Vector Clock: An Illustration
 On executing an interval event: 	Properties of Vector Clock
	Ordering of Messages
$C_i := C_i + 1$	State of a Distributed System
 On sending a message m: 	Monitoring a Distributed
$C_i := C_i + 1$	System
$\cup_i \cdots \cup_i + 1$	

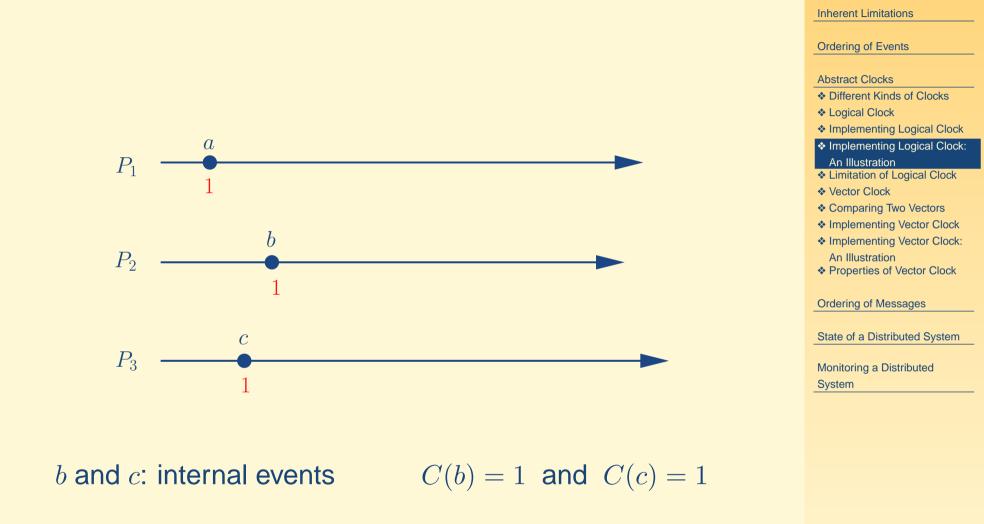
piggyback C_i on m

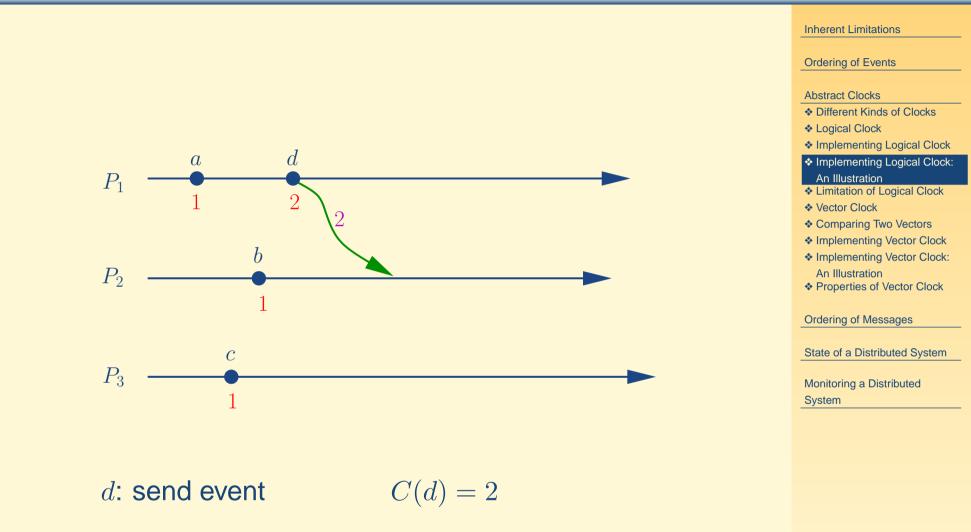
	Inherent Limitations
Each process has a local scalar clock, initialized to zero	Ordering of Events
• C_i denotes the local clock of process P_i	Abstract Clocks
Action depends on the type of the event:	 Logical Clock Implementing Logical Clock Implementing Logical Clock: An Illustration
• Protocol for process P_i :	 Limitation of Logical Clock Vector Clock Comparing Two Vectors Implementing Vector Clock
	 Implementing Vector Clock: An Illustration Properties of Vector Clock
• On executing an interval event: C = C + 1	Ordering of Messages
$C_i := C_i + 1$	State of a Distributed System
 On sending a message m: 	Monitoring a Distributed System
$C_i := C_i + 1$	
piggyback C_i on m	

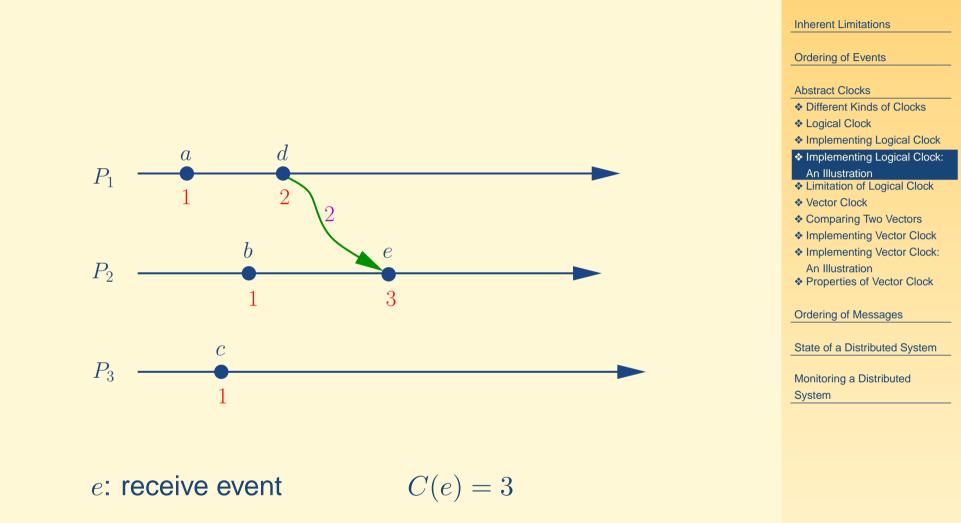
• On receiving a message *m*:

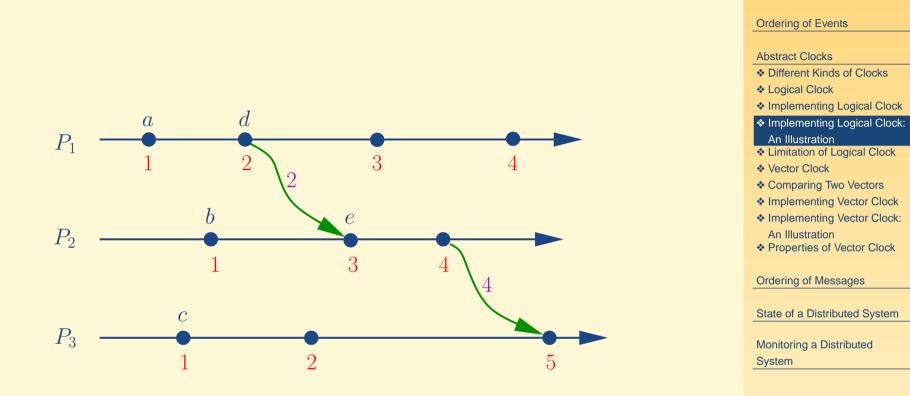
let t_m be the timestamp piggybacked on m $C_i := \max\{C_i, t_m\} + 1$











Inherent Limitations

Limitation of Logical Clock

Logical clock cannot be used to determine whether two events are concurrent

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration

Limitation of Logical Clock

- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Limitation of Logical Clock

Logical clock cannot be used to determine whether two events are concurrent

 $e \parallel f$ does not imply C(e) = C(f)

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration

Limitation of Logical Clock

- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Vector Clock

	Inherent Limitations
Captures the happened-before relation	Ordering of Events
	Abstract Clocks
	Different Kinds of Clocks
	Logical Clock
	Implementing Logical Clock
	Implementing Logical Clock:
	An Illustration
	♦ Vector Clock
	Comparing Two Vectors
	Implementing Vector Clock
	Implementing Vector Clock:
	An Illustration
	Properties of Vector Clock
	Ordering of Messages
	State of a Distributed System
	Monitoring a Distributed
	System

Vector Clock

Captures the happened-before relation

Assigns timestamp to each event such that:

 $e \to f \iff C(e) < C(f)$

C(e): timestamp for event eC(f): timestamp for event f

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock

Vector Clock

- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Vectors are compared component-wise:

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock

Comparing Two Vectors

- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Vectors are compared component-wise:

• Equality:

$$V = V'$$
 iff $\langle \forall i : V[i] = V'[i] \rangle$

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock

Comparing Two Vectors

- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Vectors are compared component-wise:

• Equality:

V = V' iff $\langle \forall i : V[i] = V'[i] \rangle$

Less Than:

 $V < V' \quad \text{iff} \quad \langle \forall i : V[i] \leq V'[i] \rangle \land \langle \exists i : V[i] < V'[i] \rangle$

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock

Comparing Two Vectors

- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Vectors are compared component-wise:

• Equality:

V = V' iff $\langle \forall i : V[i] = V'[i] \rangle$

Less Than:

 $V < V' \quad \text{iff} \quad \langle \forall i : V[i] \leq V'[i] \rangle \land \langle \exists i : V[i] < V'[i] \rangle$

$$\begin{bmatrix} 1\\2\\0 \end{bmatrix} < \begin{bmatrix} 2\\3\\1 \end{bmatrix} \text{ and } \begin{bmatrix} 2\\1\\1 \end{bmatrix} < \begin{bmatrix} 2\\3\\4 \end{bmatrix} \text{ but } \begin{bmatrix} 1\\2\\1 \end{bmatrix} \not < \begin{bmatrix} 2\\1\\3 \end{bmatrix}$$

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock

Comparing Two Vectors

- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- Each process has a local vector clock
 - C_i denotes the local clock of process P_i

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- Each process has a local vector clock
 - C_i denotes the local clock of process P_i
- Action depends on the type of the event:

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- Each process has a local vector clock
 - C_i denotes the local clock of process P_i
- Action depends on the type of the event:
- Protocol for process P_i :

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors

Implementing Vector Clock

- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- Each process has a local vector clock
 - C_i denotes the local clock of process P_i
- Action depends on the type of the event:
- Protocol for process P_i :

• On executing an interval event: $C_i[i] := C_i[i] + 1$

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- Each process has a local vector clock
 - C_i denotes the local clock of process P_i
- Action depends on the type of the event:
- Protocol for process P_i :
 - On executing an interval event: $C_i[i] := C_i[i] + 1$
 - On sending a message *m*:

 $C_i[i] := C_i[i] + 1$ piggyback C_i on m

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock:
 An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

Action depends on the type of the event:
Protocol for process P_i:
On executing an interval event: C_i[i] := C_i[i] + 1
On sending a message m: C_i[i] := C_i[i] + 1 piggyback C_i on m
On receiving a message m:

> let t_m be the timestamp piggybacked on m $\forall k \ C_i[k] := \max\{C_i[k], t_m[k]\}\ C_i[i] := C_i[i] + 1$

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

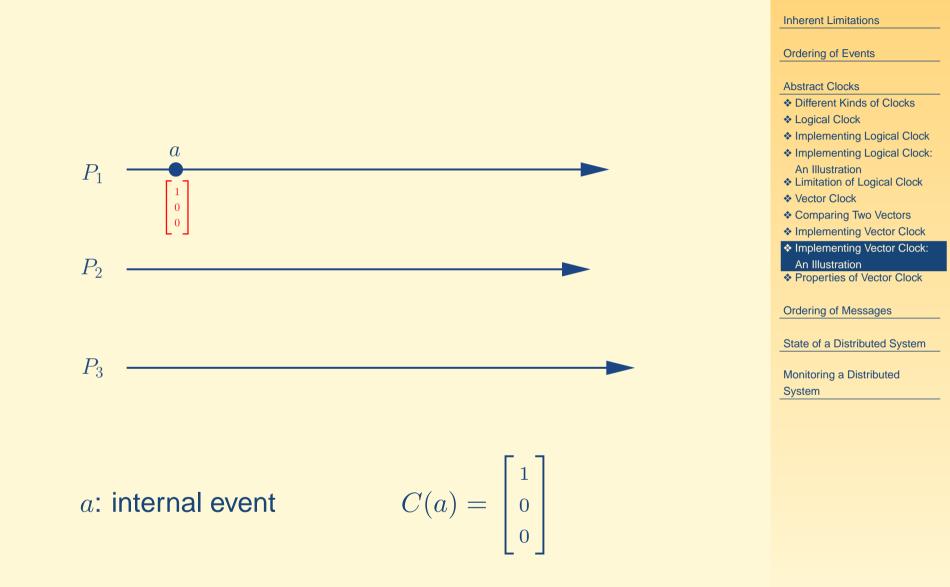
Ordering of Messages

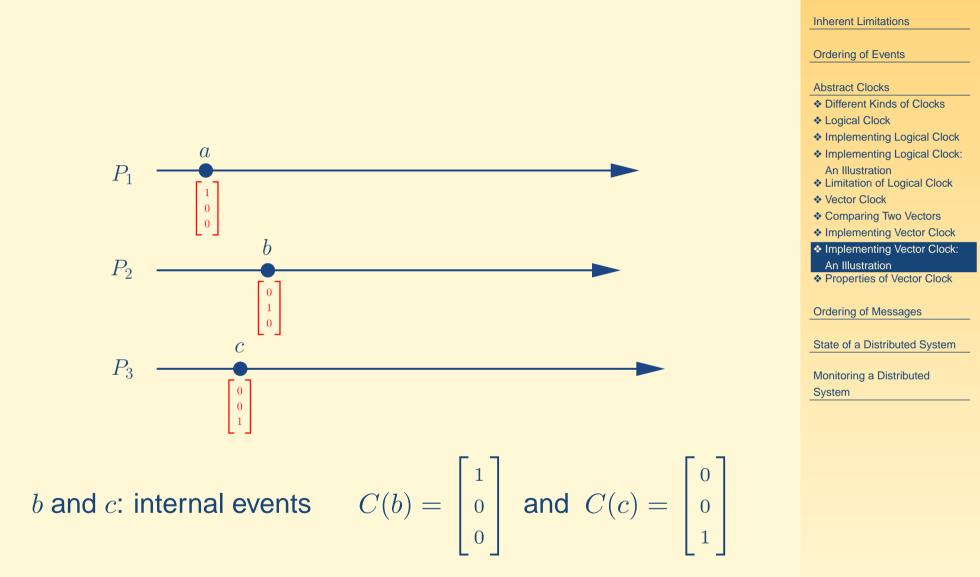
State of a Distributed System

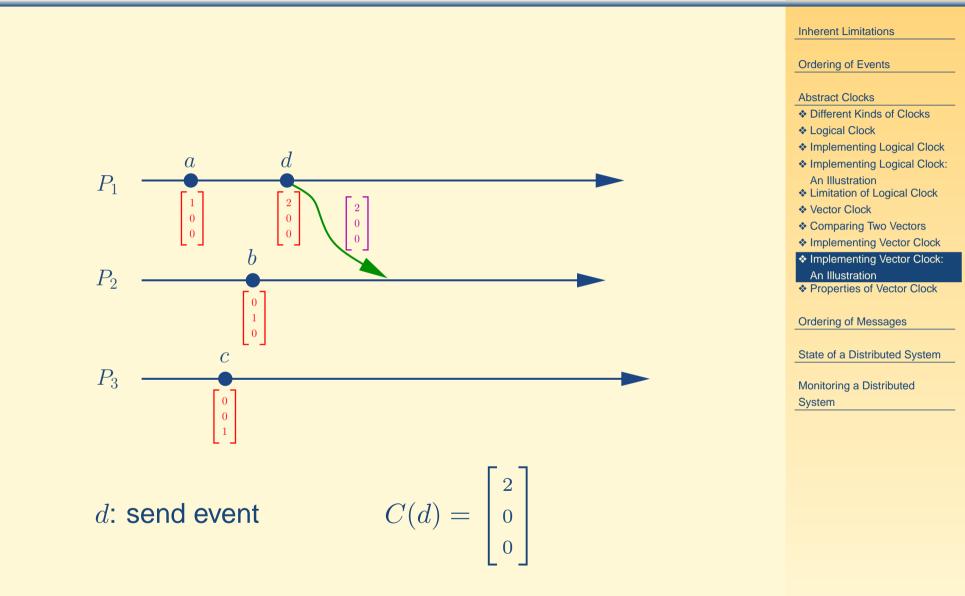
Monitoring a Distributed System

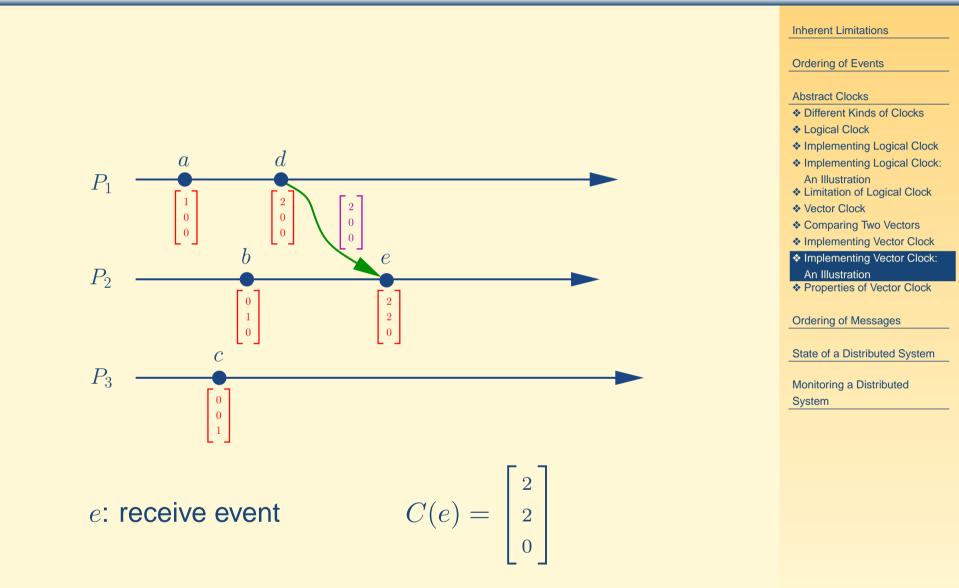
Each process has a local vector clock

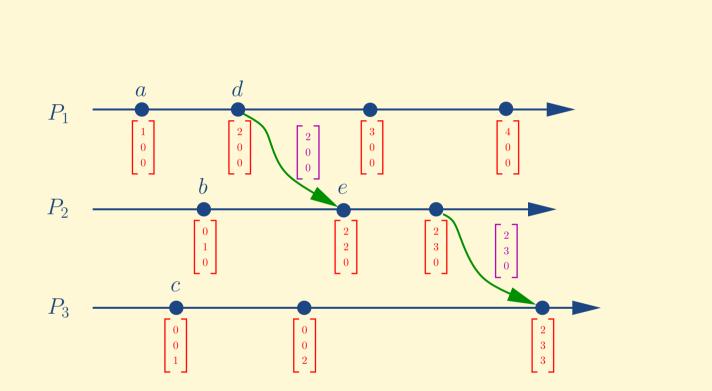
• C_i denotes the local clock of process P_i











Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration
- Properties of Vector Clock

Ordering of Messages

State of a Distributed System

How many comparisons are needed to determine whether an event e happened-before another event f?

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration

Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- How many comparisons are needed to determine whether an event *e* happened-before another event *f*?
 - As many as N integers may need to be compared in the worst case, where N is the number of processes

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration

Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- How many comparisons are needed to determine whether an event e happened-before another event f?
 - As many as N integers may need to be compared in the worst case, where N is the number of processes
 - Suppose e and f occurred on processes P_i and P_j

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration

Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- How many comparisons are needed to determine whether an event *e* happened-before another event *f*?
 - As many as N integers may need to be compared in the worst case, where N is the number of processes
 - Suppose e and f occurred on processes P_i and P_j

 $e \to f$

if and only if

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration

Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- How many comparisons are needed to determine whether an event e happened-before another event f?
 - As many as N integers may need to be compared in the worst case, where N is the number of processes
 - Suppose e and f occurred on processes P_i and P_j

 $e \to f$

if and only if

 $(i = j) \land (C(e)[i] < C(f)[i])$

Inherent Limitations

Ordering of Events

Abstract Clocks

- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration

Properties of Vector Clock

Ordering of Messages

State of a Distributed System

- How many comparisons are needed to determine whether an event e happened-before another event f?
 - As many as N integers may need to be compared in the worst case, where N is the number of processes
 - Suppose e and f occurred on processes P_i and P_j

 $e \to f$

if and only if

 $(i = j) \land (C(e)[i] < C(f)[i]) \bigvee (i \neq j) \land (C(e)[i] \leq C(f)[i])$

Inherent Limitations

Ordering of Events

Abstract Clocks

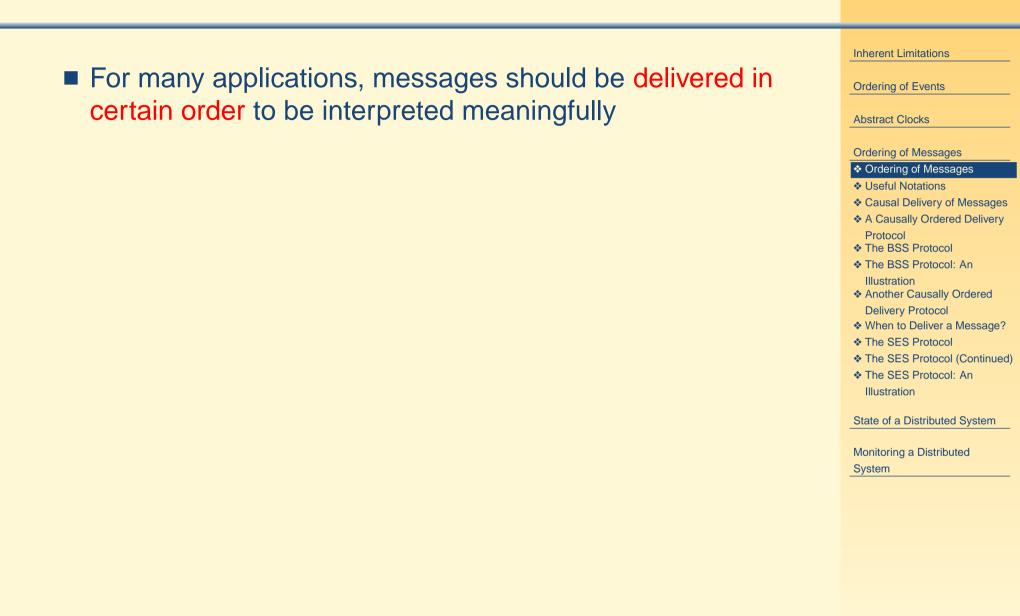
- Different Kinds of Clocks
- Logical Clock
- Implementing Logical Clock
- Implementing Logical Clock: An Illustration
- Limitation of Logical Clock
- Vector Clock
- Comparing Two Vectors
- Implementing Vector Clock
- Implementing Vector Clock: An Illustration

Properties of Vector Clock

Ordering of Messages

State of a Distributed System

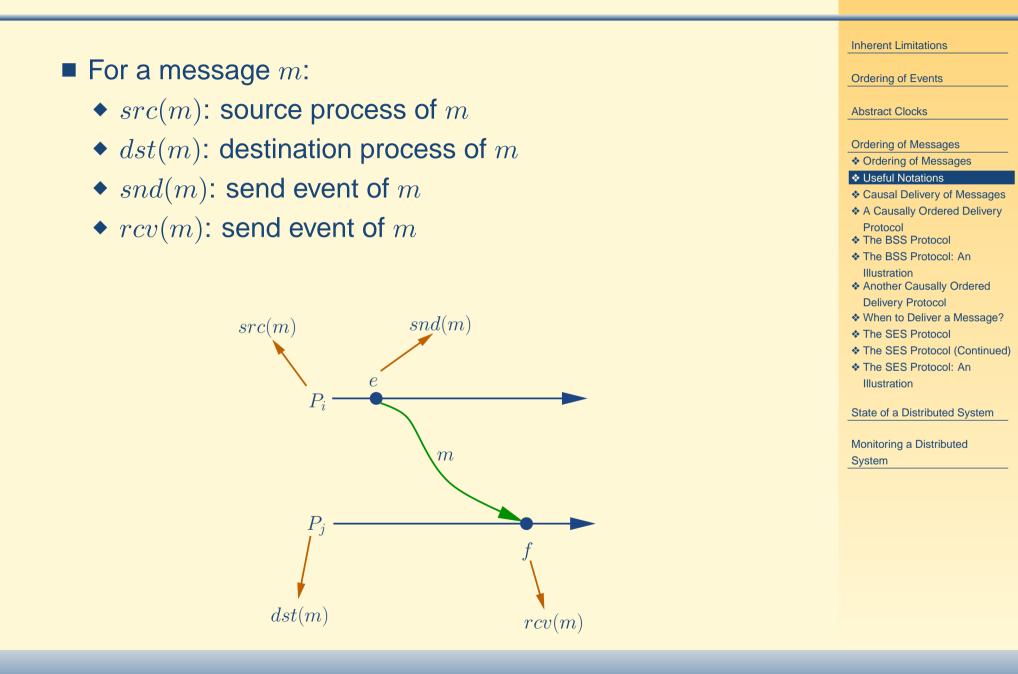
Ordering of Messages



Ordering of Messages

- Inherent Limitations For many applications, messages should be delivered in Ordering of Events certain order to be interpreted meaningfully Abstract Clocks Example: Ordering of Messages Ordering of Messages Useful Notations Causal Delivery of Messages m_1 : Have you seen the movie "Shrek"? A Causally Ordered Delivery Protocol The BSS Protocol Bob The BSS Protocol: An Illustration Another Causally Ordered **Delivery Protocol** When to Deliver a Message? m_2 : Yes I have and I liked it The SES Protocol The SES Protocol (Continued) Alice ✤ The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed Tom System
 - *m*₂ cannot be interpreted until *m*₁ has been received
 Tom receives *m*₂ before *m*₁: an undesriable behavior

Useful Notations



Causal Delivery of Messages

	Inherent Limitations
• A message w causally precedes a message m if	Ordering of Events
$snd(w) \rightarrow snd(m)$	Abstract Clocks
	Ordering of Messages
	 Ordering of Messages
	Useful Notations
	♦ Causal Delivery of Messages
	♦ A Causally Ordered Delivery
	Protocol The BSS Protocol
	♦ The BSS Protocol: An
	Illustration
	Another Causally Ordered
	Delivery Protocol
	 When to Deliver a Message? The SES Protocol
	 The SES Protocol (Continued) The SES Protocol: An
	Illustration
	Indstration
	State of a Distributed System
	Monitoring a Distributed
	System

Causal Delivery of Messages

- A message w causally precedes a message m if $snd(w) \rightarrow snd(m)$
- An execution of a distributed system is said to be causally ordered if the following holds for every message m:

every message that causally precedes m and is destined for the same process as m is delivered before m

	rdering of Events
A	bstract Clocks
0	rdering of Messages
*	Ordering of Messages
*	Useful Notations
*	Causal Delivery of Message
*	A Causally Ordered Delivery
	Protocol
	The BSS Protocol
**	Illustration
*	Another Causally Ordered
	Delivery Protocol
*	When to Deliver a Message?
*	The SES Protocol
*	The SES Protocol (Continue
*	The SES Protocol: An
	Illustration
S	tate of a Distributed System
M	lonitoring a Distributed
	ystem

Causal Delivery of Messages

• A message w causally precedes a message m if $snd(w) \rightarrow snd(m)$

An execution of a distributed system is said to be causally ordered if the following holds for every message m:

every message that causally precedes m and is destined for the same process as m is delivered before m

Mathematically, for every message w:

 $(snd(w) \rightarrow snd(m)) \land (dst(w) = dst(m))$ \Rightarrow $rcv(w) \rightarrow rcv(m)$

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
Ordering of Messages
Useful Notations
Causal Delivery of Messages
A Causally Ordered Delivery
Protocol
The BSS Protocol
The BSS Protocol: An
Illustration Another Causally Ordered
Delivery Protocol
When to Deliver a Message?
♦ The SES Protocol
The SES Protocol (Continued)
✤ The SES Protocol: An
Illustration
State of a Distributed System
Monitoring a Distributed
System

Inherent Limitations Proposed by Birman, Schiper and Stephenson (BSS) Ordering of Events Abstract Clocks Ordering of Messages Ordering of Messages Useful Notations Causal Delivery of Messages ♦ A Causally Ordered Delivery Protocol The BSS Protocol The BSS Protocol: An Illustration Another Causally Ordered **Delivery Protocol** When to Deliver a Message? The SES Protocol The SES Protocol (Continued) ✤ The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed System

- Proposed by Birman, Schiper and Stephenson (BSS)
- Assumption:
 - communication is broadcast based: a process sends a message to every other process

A	Abstract Clocks
C	Ordering of Messages
•	Ordering of Messages
	Useful Notations
•	Causal Delivery of Messages
•	A Causally Ordered Delivery
	Protocol The BSS Protocol
	The BSS Protocol: An
	Illustration
•	Another Causally Ordered
	Delivery Protocol
•	When to Deliver a Message?
•	The SES Protocol
•	The SES Protocol (Continued
•	The SES Protocol: An
	Illustration
S	State of a Distributed System
Ν	Aonitoring a Distributed
6	System

- Proposed by Birman, Schiper and Stephenson (BSS)
- Assumption:
 - communication is broadcast based: a process sends a message to every other process
- Each process maintains a vector with one entry for each process:
 - let V_i denote the vector for process P_i
 - the j^{th} entry of V_i refers to the number of messages that have been broadcast by process P_j that P_i knows of

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
Ordering of Messages
Useful Notations
Causal Delivery of Messages
A Causally Ordered Delivery
Protocol The BSS Protocol
 ♦ The BSS Protocol: An
Withe BSS Protocol. An
 Another Causally Ordered
Delivery Protocol
♦ When to Deliver a Message?
✤ The SES Protocol
The SES Protocol (Continued)
The SES Protocol: An
Illustration
State of a Distributed System
Monitoring a Distributed
System



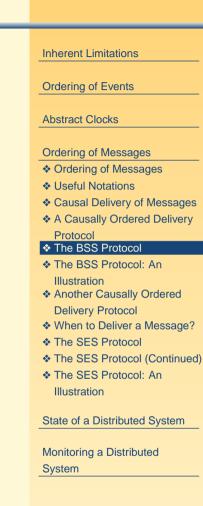
Inherent Limitations • Protocol for process P_i : Ordering of Events Abstract Clocks On broadcasting a message m: Ordering of Messages Ordering of Messages piggyback V_i on mUseful Notations Causal Delivery of Messages $V_i[i] := V_i[i] + 1$ A Causally Ordered Delivery Protocol The BSS Protocol The BSS Protocol: An Illustration Another Causally Ordered **Delivery Protocol** When to Deliver a Message? ✤ The SES Protocol The SES Protocol (Continued) ✤ The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed System

- Protocol for process P_i :
 - On broadcasting a message *m*:

 $\begin{array}{l} \textbf{piggyback } V_i \text{ on } m \\ V_i[i] := V_i[i] + 1 \end{array}$

• On arrival of a message m from process P_j :

let V_m be the vector piggybacked on mdeliver m once $V_i \ge V_m$



- Protocol for process P_i :
 - On broadcasting a message m:

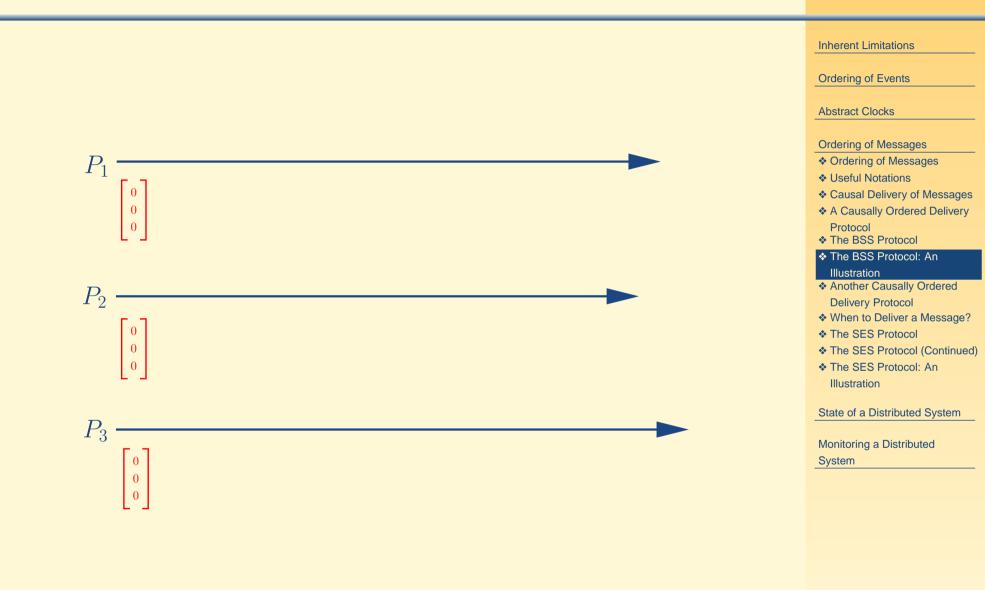
piggyback V_i on m $V_i[i] := V_i[i] + 1$

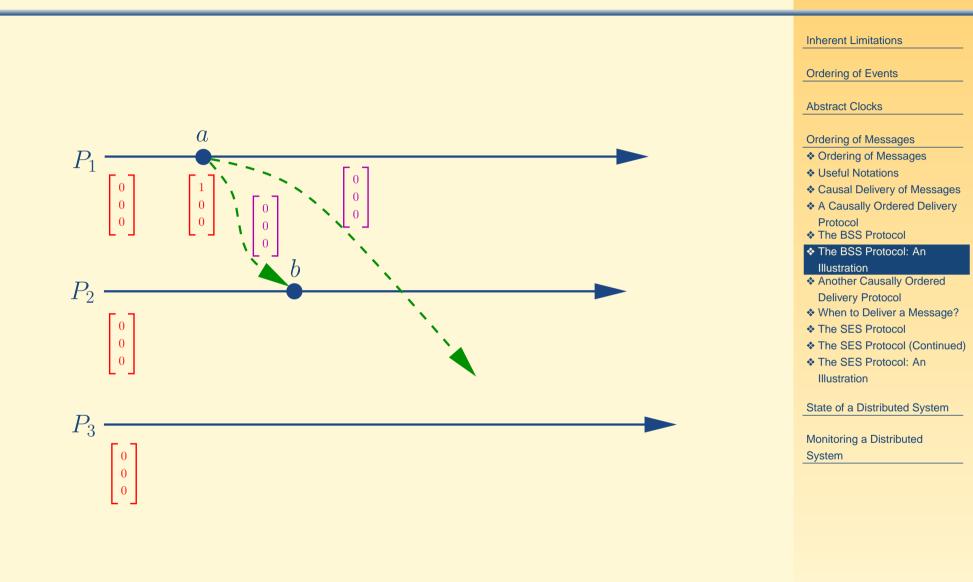
• On arrival of a message m from process P_j :

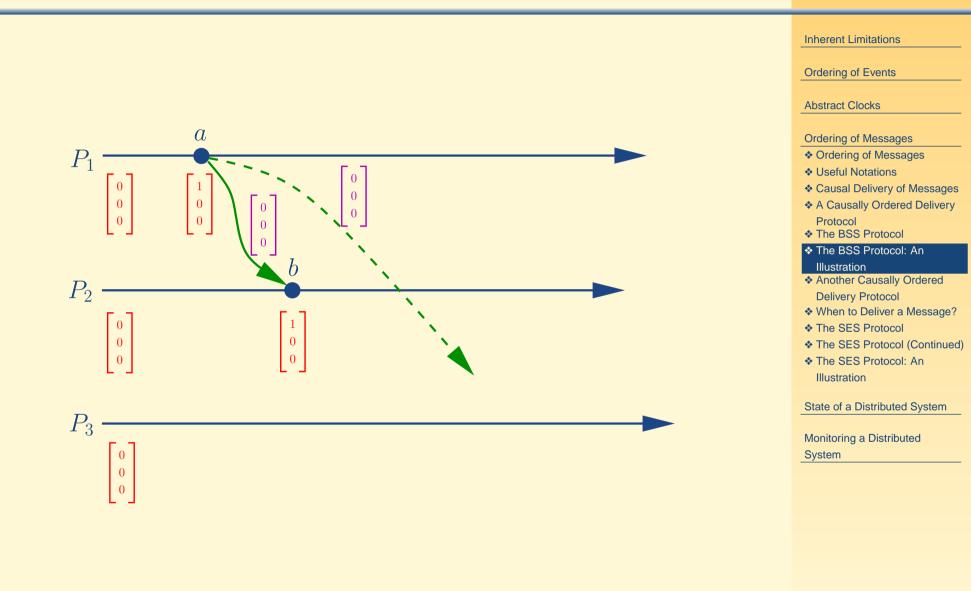
let V_m be the vector piggybacked on mdeliver m once $V_i \ge V_m$

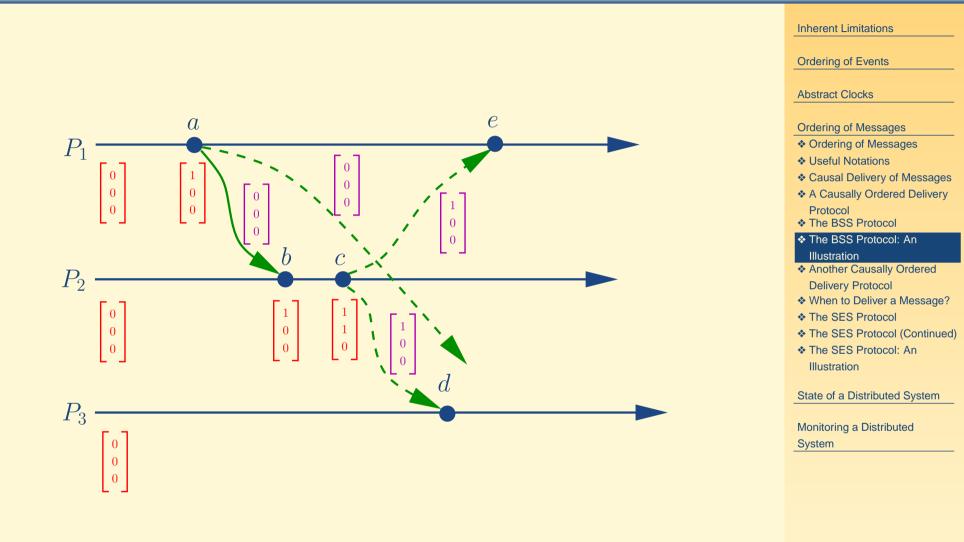
• On delivery of a message m sent by process P_j : $V_i[j] := V_i[j] + 1$

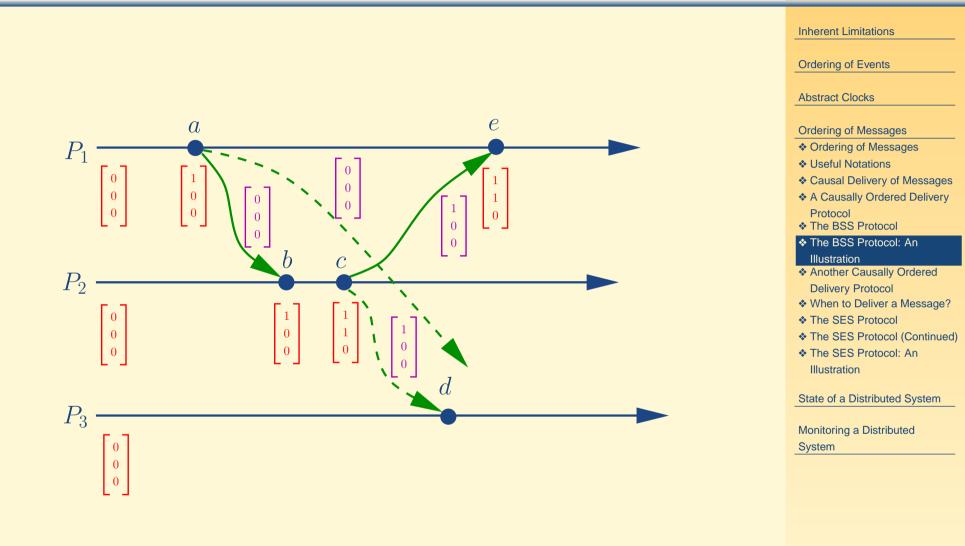


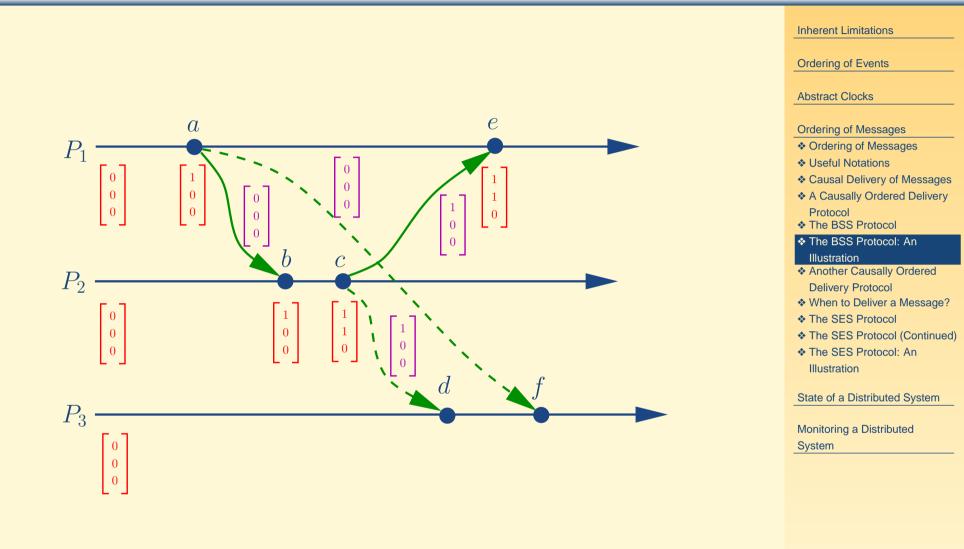


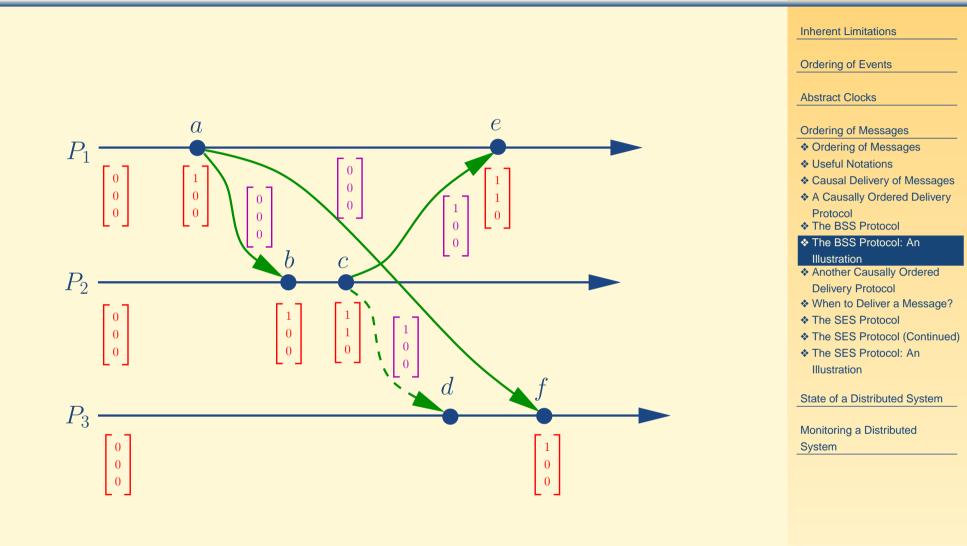


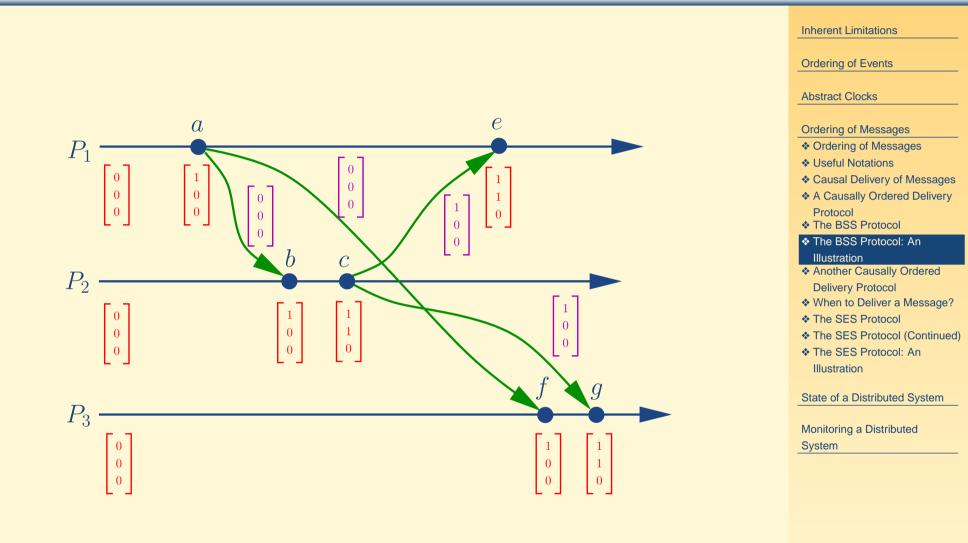












Inherent Limitations Proposed by Schiper, Eggli and Sandoz (SES) Ordering of Events Abstract Clocks Ordering of Messages Ordering of Messages Useful Notations Causal Delivery of Messages A Causally Ordered Delivery Protocol The BSS Protocol The BSS Protocol: An Illustration Another Causally Ordered **Delivery Protocol** When to Deliver a Message? The SES Protocol The SES Protocol (Continued) ✤ The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed System

- Proposed by Schiper, Eggli and Sandoz (SES)
- Assumption:
 - communication is point-to-point
 - processes are using vector clock algorithm

- Proposed by Schiper, Eggli and Sandoz (SES)
- Assumption:
 - communication is point-to-point
 - processes are using vector clock algorithm
- To determine whether a message has arrived out of causal order:

Monitoring a Distributed System

- Proposed by Schiper, Eggli and Sandoz (SES)
- Assumption:
 - communication is point-to-point
 - processes are using vector clock algorithm
- To determine whether a message has arrived out of causal order:
 - let V_m denote the vector timestamp of message m
 - let V_i denote the vector clock of process P_i

_	
	Inherent Limitations
	Ordering of Events
	Abstract Clocks
	Ordering of Messages
	Ordering of Messages
	Useful Notations
	Causal Delivery of Messages
	♦ A Causally Ordered Delivery
	Protocol
	The BSS Protocol
	The BSS Protocol: An
	Illustration
	Another Causally Ordered
	Delivery Protocol
	When to Deliver a Message?
	The SES Protocol
	The SES Protocol (Continued)
	The SES Protocol: An
	Illustration
	State of a Distributed System
	Monitoring a Distributed
	System

- Proposed by Schiper, Eggli and Sandoz (SES)
- Assumption:
 - communication is point-to-point
 - processes are using vector clock algorithm
- To determine whether a message has arrived out of causal order:
 - let V_m denote the vector timestamp of message m
 - let V_i denote the vector clock of process P_i

Does there exist a message w that causally precedes m and destined for P_i such that $V_i \neq V_w$?

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
Ordering of Messages
 Useful Notations
 Causal Delivery of Messages
A Causally Ordered Delivery
Protocol The BSS Protocol
The BSS Protocol: An
Illustration
Another Causally Ordered
Delivery Protocol
Delivery Protocol
Delivery Protocol When to Deliver a Message? The SES Protocol
Delivery Protocol When to Deliver a Message? The SES Protocol The SES Protocol (Continued
Delivery Protocol When to Deliver a Message? The SES Protocol The SES Protocol (Continued The SES Protocol: An
Delivery Protocol When to Deliver a Message? The SES Protocol The SES Protocol (Continued
Delivery Protocol When to Deliver a Message? The SES Protocol The SES Protocol (Continued The SES Protocol: An
Delivery Protocol & When to Deliver a Message? & The SES Protocol & The SES Protocol (Continued & The SES Protocol: An Illustration State of a Distributed System
Delivery Protocol & When to Deliver a Message? & The SES Protocol & The SES Protocol (Continued & The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed
Delivery Protocol & When to Deliver a Message? & The SES Protocol & The SES Protocol (Continued & The SES Protocol: An Illustration State of a Distributed System
Delivery Protocol & When to Deliver a Message? & The SES Protocol & The SES Protocol (Continued & The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed

	Inherent Limitations
A message m destined for process P_i can be delivered if:	Ordering of Events
	Abstract Clocks
	Ordering of Messages
	✤ Ordering of Messages
	Useful Notations
	Causal Delivery of Messages
	A Causally Ordered Delivery
	Protocol The BSS Protocol
	✤ The BSS Protocol: An
	Illustration Another Causally Ordered
	Delivery Protocol
	♦ When to Deliver a Message?
	The SES Protocol
	 The SES Protocol (Continued)
	The SES Protocol: An
	Illustration
	State of a Distributed System
	Manifesian - Distributed
	Monitoring a Distributed
	System

• A message m destined for process P_i can be delivered if:

 for every message w that causally precedes m and destined for P_i:

$$V_i > V_u$$

Ordering of Events
Abstract Clocks
Ordering of Messages
Ordering of Messages
Useful Notations
Causal Delivery of Messages
A Causally Ordered Delivery
Protocol
✤ The BSS Protocol
✤ The BSS Protocol: An
Illustration
Delivery Protocol
♦ When to Deliver a Message?
♦ The SES Protocol
The SES Protocol (Continued)
 The SES Protocol (Continued) The SES Protocol: An
 The SES Protocol (Continued) The SES Protocol: An Illustration
The SES Protocol: An
 The SES Protocol: An Illustration
 The SES Protocol: An Illustration State of a Distributed System
 The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed
 The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed
 The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed
 The SES Protocol: An Illustration State of a Distributed System Monitoring a Distributed

• A message m destined for process P_i can be delivered if:

 for every message w that causally precedes m and destined for P_i:

$$V_i > V_w$$

or, equivalently

Inherent Limitations Ordering of Events
Abstract Clocks
Ordering of Messages
Ordering of Messages
Useful Notations
Causal Delivery of Messages
A Causally Ordered Delivery
Protocol The BSS Protocol
 The BSS Protocol: An
Illustration
Another Causally Ordered
Delivery Protocol
When to Deliver a Message?
The SES Protocol
The SES Protocol (Continued)
The SES Protocol: An
Illustration
State of a Distributed System
Monitoring a Distributed

• A message m destined for process P_i can be delivered if:

 for every message w that causally precedes m and destined for P_i:

$$V_i > V_w$$

or, equivalently

 let past(m, j) denote the set of all messages that causally precede m and are destined for process P_j:

 $V_i > \max_{w \in past(m,i)} \{V_w\}$



Each process maintains a list of tuples with at most one	Inherent Limitations Ordering of Events
tuple for every other process	Abstract Clocks
 ◆ let <i>DL_i</i> denote the list for process <i>P_i</i> 	 Ordering of Messages Ordering of Messages Useful Notations Causal Delivery of Messages A Causally Ordered Delivery Protocol The BSS Protocol The BSS Protocol: An Illustration Another Causally Ordered Delivery Protocol When to Deliver a Message? The SES Protocol (Continued) The SES Protocol: An Illustration
	State of a Distributed System Monitoring a Distributed System

	Inherent Limitations
Each process maintains a list of tuples with at most one	Ordering of Events
tuple for every other process	Abstract Clocks
• let DL_i denote the list for process P_i	Ordering of Messages
	Useful Notations
The list is piggy healed on every measure a process conde	 Causal Delivery of Messages A Causally Ordered Delivery
The list is piggybacked on every message a process sends	Protocol The BSS Protocol
• let DL_m denote the list for message m	The BSS Protocol: An Illustration
	 Another Causally Ordered Delivery Protocol
	 When to Deliver a Message? The SES Protocol
	 The SES Protocol (Continued) The SES Protocol: An
	Illustration
	State of a Distributed System
	Monitoring a Distributed
	System

T

Each process maintains a list of tuples with at most one tuple for every other process	Inherent Limitations Ordering of Events Abstract Clocks
• let DL_i denote the list for process P_i	Ordering of Messages Ordering of Messages Station of Messages Station of Messages Causal Delivery of Messages
 The list is piggybacked on every message a process sends Iet DL_m denote the list for message m 	 Causal Denvery of Messages A Causally Ordered Delivery Protocol The BSS Protocol The BSS Protocol: An Illustration Another Causally Ordered
• if $past(m, j) = \emptyset$, then DL_m does not contain a tuple for process P_j	 Another Causally Ordered Delivery Protocol When to Deliver a Message? The SES Protocol The SES Protocol (Continued) The SES Protocol: An Illustration
	State of a Distributed System Monitoring a Distributed System

	Inherent Limitations
Each process maintains a list of tuples with at most one	Ordering of Events
tuple for every other process	Abstract Clocks
• let DL_i denote the list for process P_i	Ordering of Messages
$\sum L_i \text{ defices the field of process } L_i$	 Ordering of Messages Useful Notations
	Causal Delivery of Messages
The list is piggybacked on every message a process sends	 A Causally Ordered Delivery Protocol The BSS Protocol
• let DL_m denote the list for message m	The BSS Protocol: An Illustration
	Another Causally Ordered
• if $past(m, j) = \emptyset$, then DL_m does not contain a tuple	Delivery Protocol When to Deliver a Message?
for process P_i	 The SES Protocol The SES Protocol
ior process I_j	 The SES Protocol (Continued) The SES Protocol: An
• Otherwise, the tuple for process P_i is given by:	Illustration
	State of a Distributed System
$(j, \max_{w \in past(m,j)} \{V_w\})$	Monitoring a Distributed System

The SES Protocol (Continued)

• Protocol for process P_i :

Ordering of Events

Abstract Clocks

Ordering of Messages

- Ordering of Messages
- Useful Notations
- Causal Delivery of Messages
- A Causally Ordered Delivery Protocol
- The BSS Protocol
- The BSS Protocol: An Illustration
- Another Causally Ordered
 Delivery Protocol
- When to Deliver a Message?
 The SES Protocol

The SES Protocol (Continued)

The SES Protocol: An Illustration

State of a Distributed System

Monitoring a Distributed System

The SES Protocol (Continued)

• Protocol for process P_i :

• On sending a message m to process P_j :

piggyback DL_i on mremove entry for P_j from DL_i , if any add (j, V_m) to DL_i Inherent Limitations

Ordering of Events

Abstract Clocks

Ordering of Messages

- Ordering of Messages
- Useful Notations
- Causal Delivery of Messages
- A Causally Ordered Delivery Protocol
- The BSS Protocol
- The BSS Protocol: An Illustration
- Another Causally Ordered
 Delivery Protocol
- When to Deliver a Message?
 The SES Protocol

The SES Protocol (Continued)

The SES Protocol: An Illustration

State of a Distributed System

Monitoring a Distributed System

The SES Protocol (Continued)

• Protocol for process P_i :

• On sending a message m to process P_j :

piggyback DL_i on mremove entry for P_j from DL_i , if any add (j, V_m) to DL_i

• On arrival of a message m from process P_j :

if DL_m does not contain any tuple for P_i then deliver m

else

let (j, V) be the tuple for P_i in DL_m deliver m once $V_i > V$ endif

Inherent Limitations

Ordering of Events

Abstract Clocks

Ordering of Messages

- Ordering of Messages
- Useful Notations
- Causal Delivery of Messages
- A Causally Ordered Delivery Protocol
- The BSS Protocol
- The BSS Protocol: An Illustration
- Another Causally Ordered
 Delivery Protocol
- When to Deliver a Message?
- The SES Protocol

The SES Protocol (Continued)

The SES Protocol: An Illustration

State of a Distributed System

Monitoring a Distributed System

The SES Protocol (Continued)

• Protocol for process P_i :

• On sending a message m to process P_j :

piggyback DL_i on mremove entry for P_j from DL_i , if any add (j, V_m) to DL_i

• On arrival of a message m from process P_j :

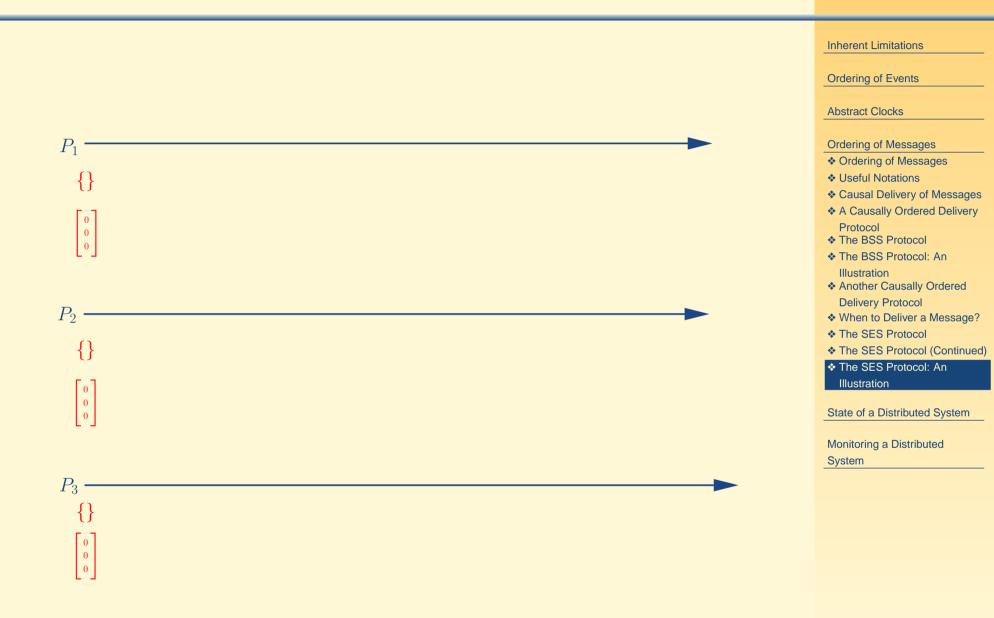
if DL_m does not contain any tuple for P_i then deliver m

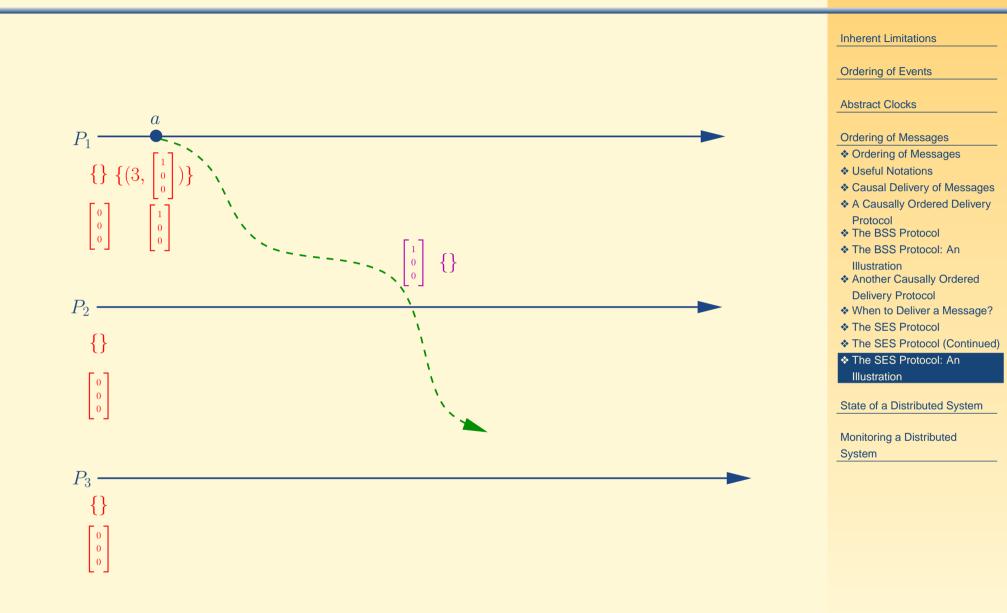
else

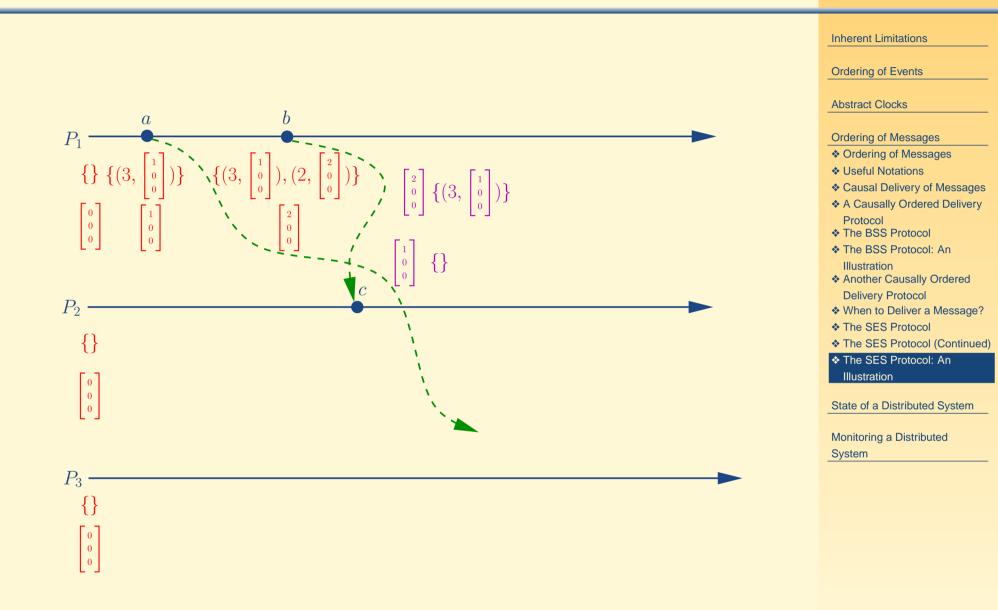
let (j, V) be the tuple for P_i in DL_m deliver m once $V_i > V$ endif

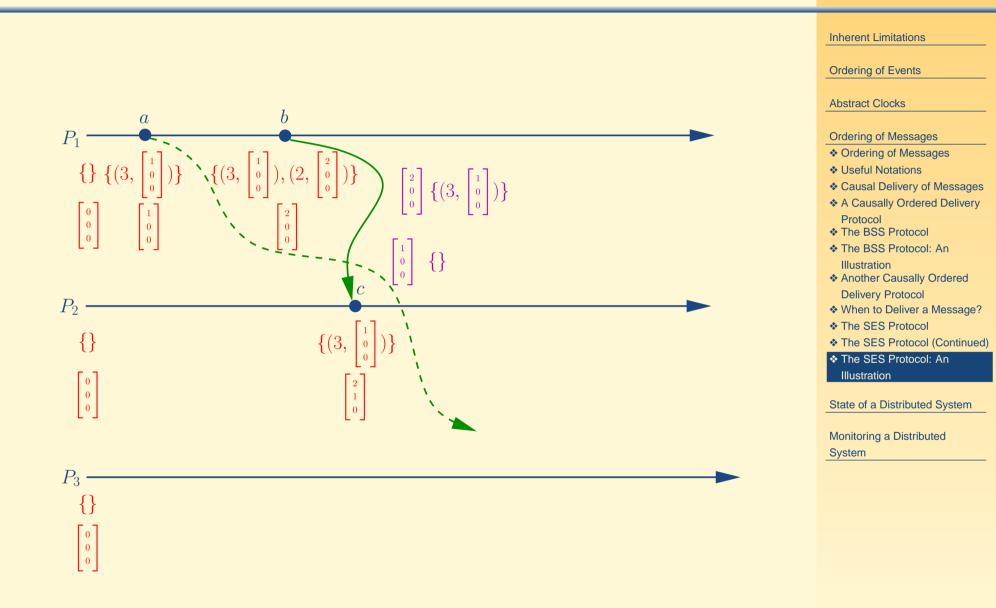
On delivery of a message m:
 merge DL_i and DL_m

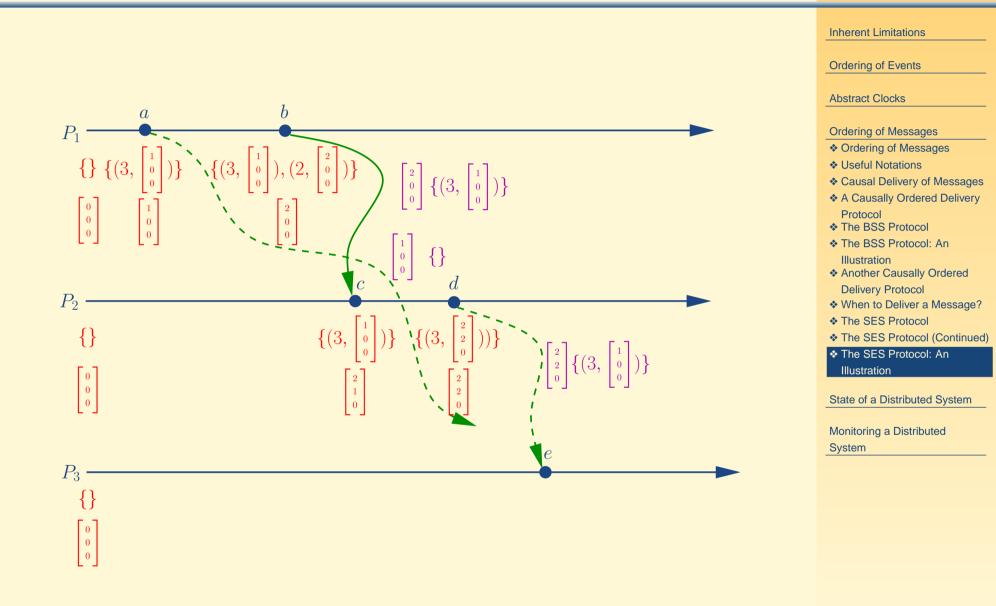
Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
Ordering of Messages
Useful Notations
Causal Delivery of Messages
A Causally Ordered Delivery
Protocol
The BSS Protocol
The BSS Protocol: An
Illustration
Another Causally Ordered
Delivery Protocol When to Deliver a Message?
 The SES Protocol
 ♦ The SES Protocol (Continue)
 The SES Protocol: An
Illustration
indstration
State of a Distributed System
Monitoring a Distributed
System

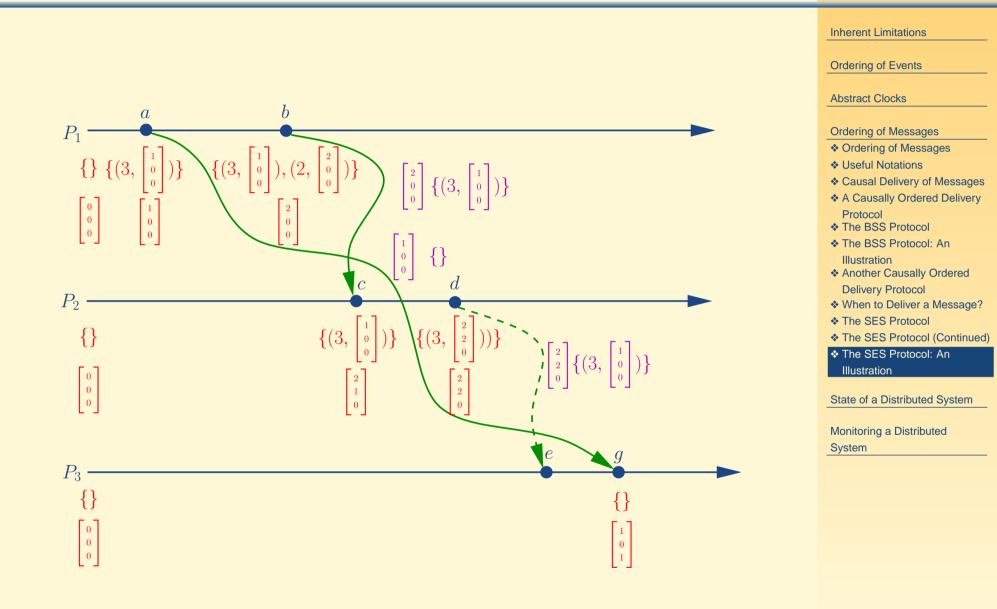


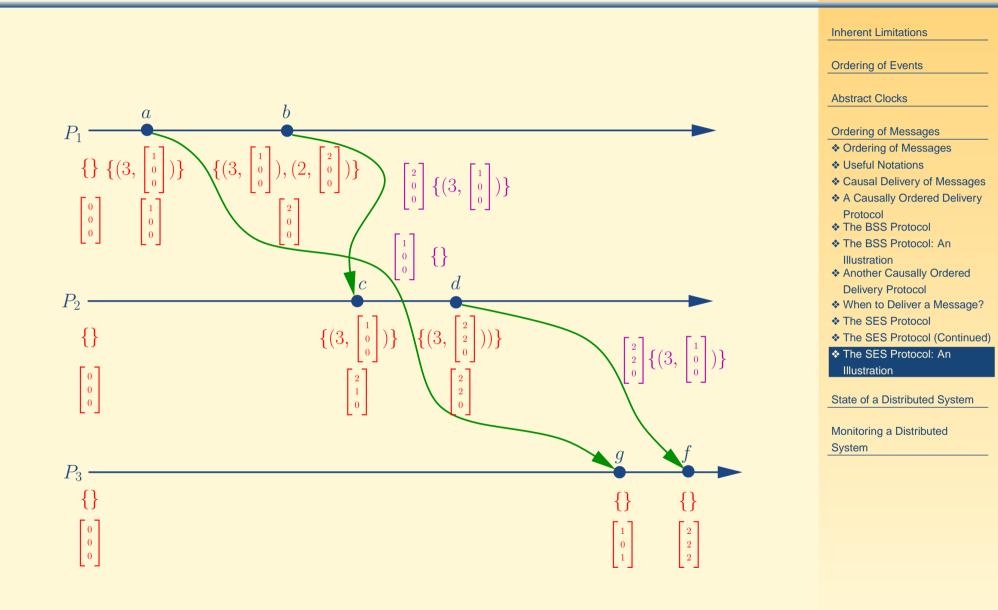












State of a distributed system is a collection of states of all its processes and channels:

/	Abstract Clocks
(Ordering of Messages
_	State of a Distributed System
_	State of a Distributed System
	 Events and Local States When is a Global State
	Meaningful?
	Revisiting Happened-Before
	Relation
	 A Consistent Global State Recording a Consistent
	Global Snapshot
•	The CL Protocol
•	The CL Protocol (Continued)
•	The CL Protocol: An
	Illustration The CL Protocol: An
	Illustration (Continued)
1	Monitoring a Distributed
	System

- State of a distributed system is a collection of states of all its processes and channels:
 - a process state is given by the values of all variables on the process

	Abstract Clocks
	Ordering of Messages
	State of a Distributed System
Ī	State of a Distributed System
	Events and Local States
	When is a Global State
	Meaningful?
	Revisiting Happened-Before
	Relation
	A Consistent Global State
	Recording a Consistent
	Global Snapshot
	 The CL Protocol The CL Protocol
	 The CL Protocol (Continued The CL Protocol: An
	Illustration The CL Protocol: An
	Illustration (Continued)
	indefation (continued)
	Monitoring a Distributed
	System
1	

Inherent Limitations State of a distributed system is a collection of states of all Ordering of Events its processes and channels: Abstract Clocks a process state is given by the values of all variables on Ordering of Messages the process State of a Distributed System State of a Distributed System Events and Local States a channel state is given by the set of messages in transit When is a Global State Meaningful? in the channel Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent **Global Snapshot** The CL Protocol The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

- State of a distributed system is a collection of states of all its processes and channels:
 - a process state is given by the values of all variables on the process
 - a channel state is given by the set of messages in transit in the channel
 - can be determined by examining states of the two processes it connects

Inherent Limitations	
Ordering of Events	
Abstract Clocks	
Ordering of Messages	
State of a Distributed System	ı
State of a Distributed System	err
Events and Local States	
When is a Global State	
Meaningful?	
Revisiting Happened-Before	re
Relation	
A Consistent Global State	
Recording a Consistent	
Global Snapshot	
The CL Protocol	
The CL Protocol (Continue)	ed)
The CL Protocol: An	
Illustration	
The CL Protocol: An	
Illustration (Continued)	
Monitoring a Distributed	
System	

- State of a distributed system is a collection of states of all its processes and channels:
 - a process state is given by the values of all variables on the process
 - a channel state is given by the set of messages in transit in the channel
 - can be determined by examining states of the two processes it connects
- State of a process is called local state or local snapshot
 - the textbook refers to local state as cut event
 - cut event is not same as event

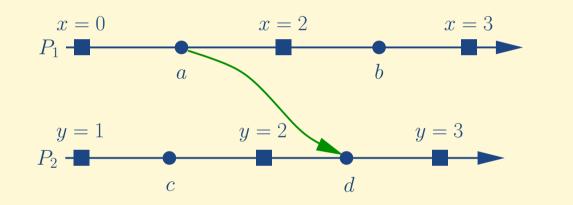
_	ordering of Events
A	bstract Clocks
С	ordering of Messages
S	tate of a Distributed System
_	State of a Distributed System
	Events and Local States
*	When is a Global State
	Meaningful?
*	 Revisiting Happened-Before Relation
*	A Consistent Global State
*	Recording a Consistent
	Global Snapshot
*	The CL Protocol
*	The CL Protocol (Continued
*	The CL Protocol: An
	Illustration
*	The CL Protocol: An
	Illustration (Continued)
	Ionitoring a Distributed
S	ystem

- State of a distributed system is a collection of states of all its processes and channels:
 - a process state is given by the values of all variables on the process
 - a channel state is given by the set of messages in transit in the channel
 - can be determined by examining states of the two processes it connects
- State of a process is called local state or local snapshot
 - the textbook refers to local state as cut event
 - cut event is not same as event
- State of a distributed system is called global state or global snapshot

_	Drdering of Events
^	Abstract Clocks
-	
0	Ordering of Messages
	Note of a Distributed Custom
-	State of a Distributed System State of a Distributed System
_	 State of a Distributed System Events and Local States
	When is a Global State
	Meaningful?
	Revisiting Happened-Before
	Relation
•	A Consistent Global State
•	Recording a Consistent
	Global Snapshot
•	The CL Protocol
•	The CL Protocol (Continued
•	The CL Protocol: An
	Illustration
	The CL Protocol: An
	Illustration (Continued)
Ν	Ionitoring a Distributed
S	System

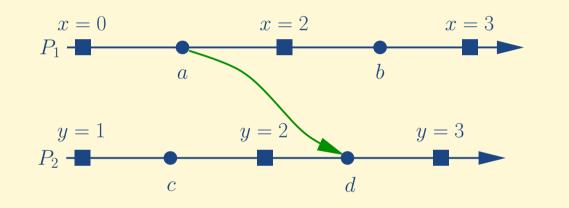
	Inherent Limitations
Processes change their states by executing events	Ordering of Events
	Abstract Clocks
	Ordering of Messages
	State of a Distributed System
	 State of a Distributed System Events and Local States
	When is a Global State Meaningful?
	 Revisiting Happened-Before Relation A Consistent Global State
	Recording a Consistent
	Global Snapshot The CL Protocol
	 The CL Protocol (Continued) The CL Protocol Are
	The CL Protocol: An Illustration
	The CL Protocol: An
	Illustration (Continued)
	Monitoring a Distributed System

- Processes change their states by executing events
- Example:





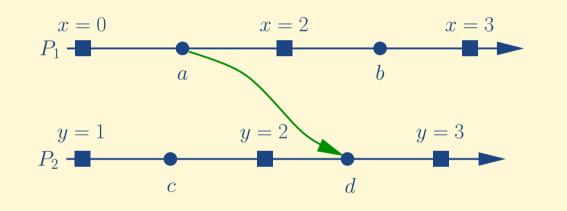
- Processes change their states by executing events
- Example:



• Process P_1 changes its state from x = 0 to x = 2 on executing event a

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
 Events and Local States
✤ When is a Global State
Meaningful?
 Revisiting Happened-Before
Relation
✤ A Consistent Global State
Recording a Consistent
Global Snapshot
The CL Protocol
The CL Protocol (Continued)
The CL Protocol: An
Illustration
Illustration (Continued)
Monitoring a Distributed System

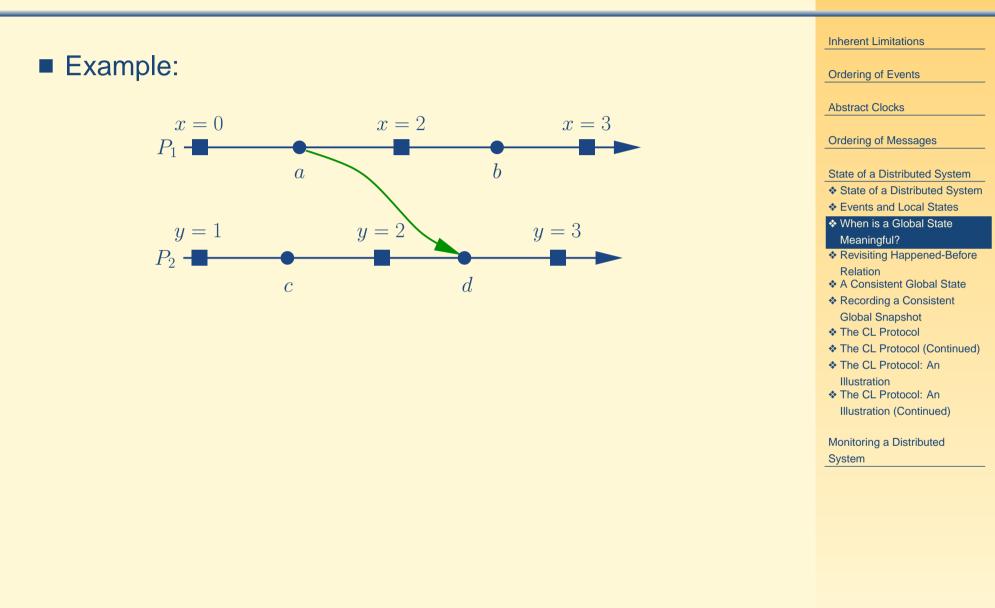
- Processes change their states by executing events
- Example:



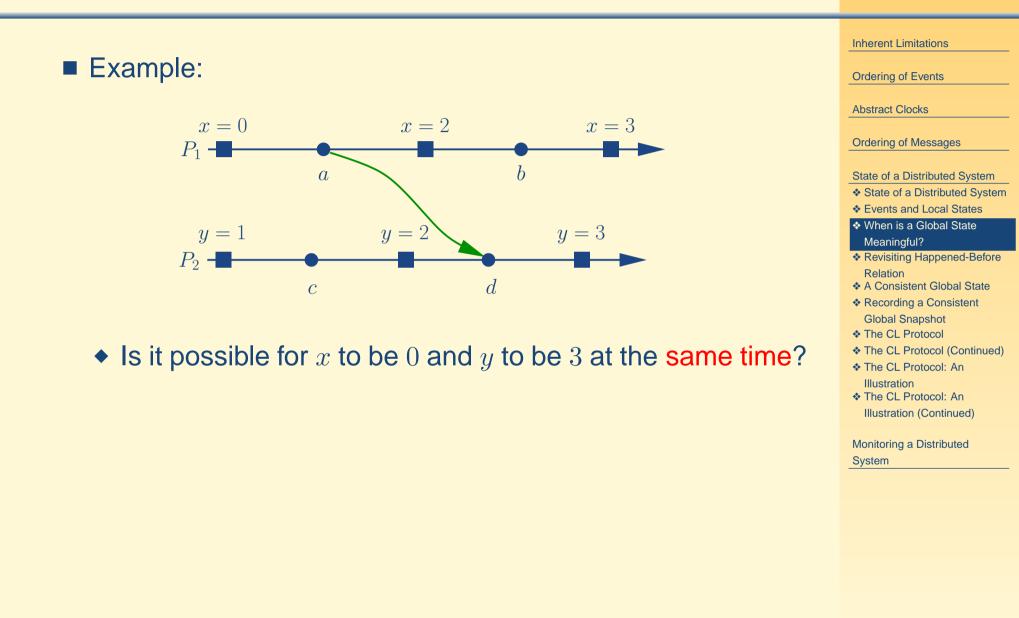
- Process P_1 changes its state from x = 0 to x = 2 on executing event a
- Process P_2 changes its state from y = 2 to y = 3 on executing event d

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
State of a Distributed System
Events and Local States
When is a Global State
Meaningful?
Revisiting Happened-Before
Relation
♦ A Consistent Global State
Recording a Consistent
Global Snapshot
The CL Protocol
The CL Protocol (Continued)
The CL Protocol: An
Illustration
The CL Protocol: An
Illustration (Continued)
Monitoring a Distributed System

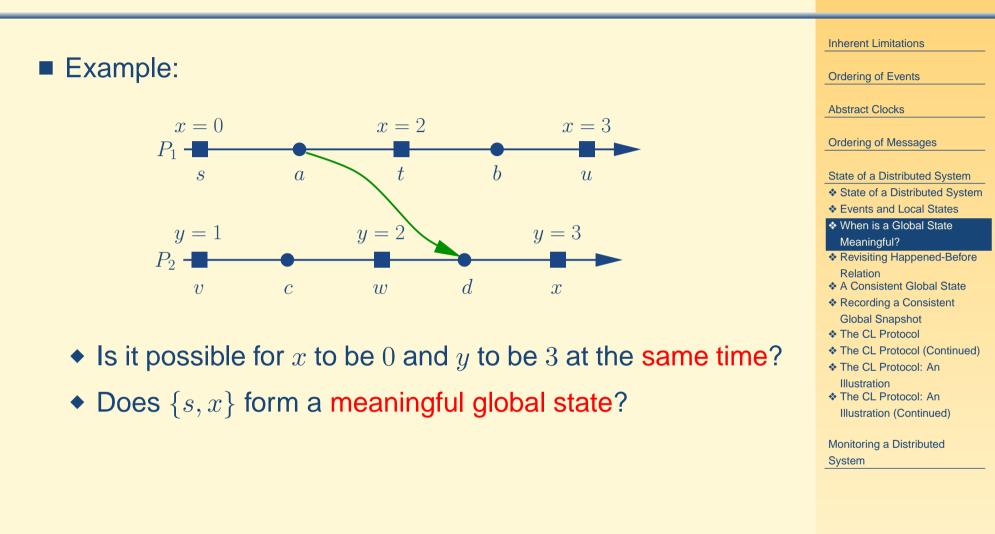
When is a Global State Meaningful?



When is a Global State Meaningful?



When is a Global State Meaningful?



	Inherent Limitations
Typically, happened-before relation is defined on events	Ordering of Events
	Abstract Clocks
	Ordering of Messages
	State of a Distributed System State of a Distributed System
	 State of a Distributed System Events and Local States
	When is a Global State Meaningful?
	 Revisiting Happened-Before Relation
	A Consistent Global State
	Recording a Consistent Global Snapshot
	♦ The CL Protocol
	The CL Protocol (Continued)
	The CL Protocol: An
	Illustration The CL Protocol: An
	Illustration (Continued)
	Monitoring a Distributed
	System

	Inherent Limitations
Typically, happened-before relation is defined on events	Ordering of Events
	Abstract Clocks
The relation can be extended to local states as follows:	Ordering of Messages
	State of a Distributed System
	State of a Distributed System
	Events and Local States
	✤ When is a Global State
	Meaningful?
	Revisiting Happened-Before
	Relation A Consistent Global State
	Recording a Consistent
	Global Snapshot
	The CL Protocol
	The CL Protocol (Continued)
	The CL Protocol: An
	Illustration
	Illustration (Continued)
	Monitoring a Distributed
	System

Inherent Limitations Typically, happened-before relation is defined on events Ordering of Events Abstract Clocks The relation can be extended to local states as follows: Ordering of Messages State of a Distributed System • let s be a local state of process P_i State of a Distributed System Events and Local States When is a Global State Meaningful? Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent **Global Snapshot** The CL Protocol The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

Inherent Limitations Typically, happened-before relation is defined on events Ordering of Events Abstract Clocks The relation can be extended to local states as follows: Ordering of Messages State of a Distributed System • let s be a local state of process P_i State of a Distributed System Events and Local States • let t be a local state of process P_i When is a Global State Meaningful? Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent **Global Snapshot** The CL Protocol The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

Inherent Limitations Typically, happened-before relation is defined on events Ordering of Events Abstract Clocks The relation can be extended to local states as follows: Ordering of Messages State of a Distributed System • let s be a local state of process P_i State of a Distributed System Events and Local States • let t be a local state of process P_i When is a Global State Meaningful? Revisiting Happened-Before • $s \rightarrow t$ if there exist events e and f such that: Relation A Consistent Global State Recording a Consistent **Global Snapshot** The CL Protocol The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

Inherent Limitations Typically, happened-before relation is defined on events Ordering of Events Abstract Clocks The relation can be extended to local states as follows: Ordering of Messages State of a Distributed System • let s be a local state of process P_i State of a Distributed System Events and Local States • let t be a local state of process P_i When is a Global State Meaningful? Revisiting Happened-Before • $s \rightarrow t$ if there exist events e and f such that: Relation A Consistent Global State 1. P_i executed e after s, Recording a Consistent **Global Snapshot** The CL Protocol The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

Inherent Limitations Typically, happened-before relation is defined on events Ordering of Events Abstract Clocks The relation can be extended to local states as follows: Ordering of Messages State of a Distributed System • let s be a local state of process P_i State of a Distributed System Events and Local States • let t be a local state of process P_i When is a Global State Meaningful? Revisiting Happened-Before • $s \rightarrow t$ if there exist events e and f such that: Relation A Consistent Global State 1. P_i executed e after s, Recording a Consistent **Global Snapshot** The CL Protocol 2. P_i executed f before t, and The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

Inherent Limitations Typically, happened-before relation is defined on events Ordering of Events Abstract Clocks The relation can be extended to local states as follows: Ordering of Messages State of a Distributed System • let s be a local state of process P_i State of a Distributed System Events and Local States • let t be a local state of process P_i When is a Global State Meaningful? Revisiting Happened-Before • $s \rightarrow t$ if there exist events e and f such that: Relation A Consistent Global State 1. P_i executed e after s, Recording a Consistent **Global Snapshot** The CL Protocol 2. P_i executed f before t, and The CL Protocol (Continued) ✤ The CL Protocol: An Illustration **3.** $e \rightarrow f$ ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

A Consistent Global State

	Inherent Limitations
For a global state G, let G[i] refer to the local state of	Ordering of Events
process P_i in G	Abstract Clocks
	Ordering of Messages
	State of a Distributed System
	 State of a Distributed System Events and Local States
	 When is a Global State Meaningful?
	 Revisiting Happened-Before Relation
	✤ A Consistent Global State
	Recording a Consistent
	Global Snapshot
	The CL Protocol (Continued)
	✤ The CL Protocol: An
	Illustration
	The CL Protocol: An Illustration (Continued)
	Monitoring a Distributed System

A Consistent Global State

• For a global state G , let $G[i]$ refer to the local state of	Inherent Limitations Ordering of Events
process P_i in G	Abstract Clocks
	Ordering of Messages
A global state G is meaningful or consistent if	State of a Distributed System
$\forall i,j: i \neq j: (G[i] \nrightarrow G[j]) \land (G[j] \nrightarrow G[i])$	 Events and Local States When is a Global State Meaningful?
	 Revisiting Happened-Before Relation A Consistent Global State
	 Recording a Consistent Global Snapshot
	 The CL Protocol The CL Protocol (Continued)
	 The CL Protocol: An Illustration The CL Protocol: An
	 The CL Protocol: An Illustration (Continued)
	Monitoring a Distributed System

A Consistent Global State

$\forall i,j: i \neq j: G[i] \parallel G[j]$	For a global state G , let $G[i]$ refer to the local stat process P_i in G	e of Ordering of Events Abstract Clocks
Revisiting Happened-Befor Relation		State of a Distributed System State of a Distributed System Events and Local States
 Recording a Consistent Global Snapshot The CL Protocol 	$\forall i, j : i \neq j : G[i] \parallel G[j]$	 Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent Global Snapshot
		 The CL Protocol (Continued) The CL Protocol: An Illustration The CL Protocol: An
Monitoring a Distributed System		

Inherent Limitations Proposed by Chandy and Lamport (CL) Ordering of Events Abstract Clocks Ordering of Messages State of a Distributed System State of a Distributed System Events and Local States When is a Global State Meaningful? Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent **Global Snapshot** The CL Protocol The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

Inherent Limitations Proposed by Chandy and Lamport (CL) Ordering of Events Abstract Clocks Assumptions and Features: Ordering of Messages channels satisfy first-in-first-out (FIFO) property State of a Distributed System State of a Distributed System Events and Local States When is a Global State Meaningful? Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent **Global Snapshot** The CL Protocol The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

Inherent Limitations Proposed by Chandy and Lamport (CL) Ordering of Events Abstract Clocks Assumptions and Features: Ordering of Messages channels satisfy first-in-first-out (FIFO) property State of a Distributed System State of a Distributed System channels are not required to be bidirectional Events and Local States When is a Global State Meaningful? Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent **Global Snapshot** The CL Protocol The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

Inherent Limitations Proposed by Chandy and Lamport (CL) Ordering of Events Abstract Clocks Assumptions and Features: Ordering of Messages channels satisfy first-in-first-out (FIFO) property State of a Distributed System State of a Distributed System channels are not required to be bidirectional Events and Local States When is a Global State communication topology may not be fully connected Meaningful? Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent **Global Snapshot** The CL Protocol The CL Protocol (Continued) ✤ The CL Protocol: An Illustration ✤ The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

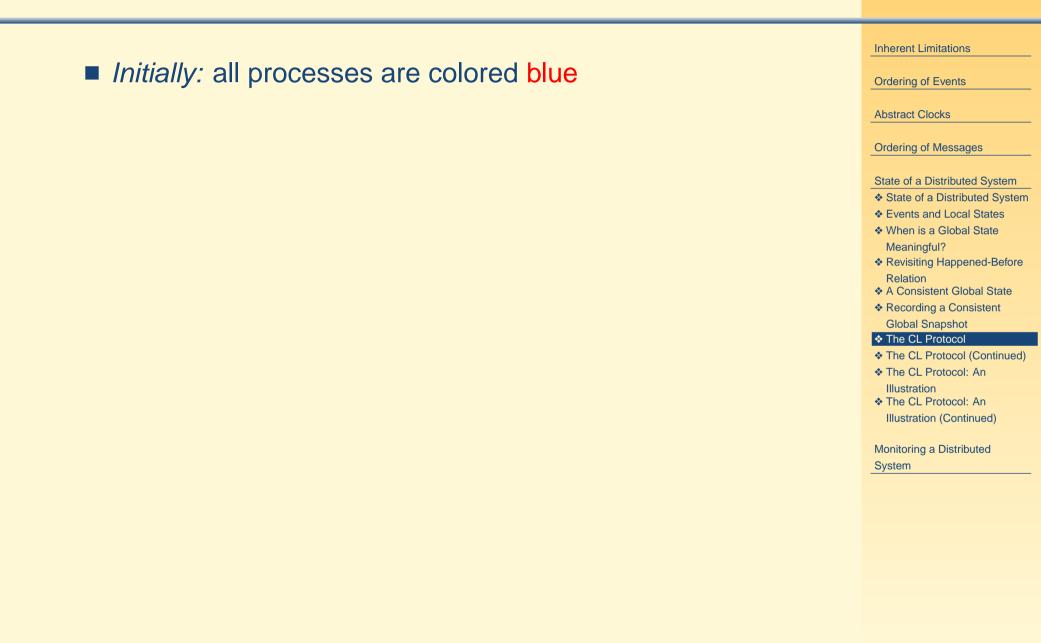
Recording a Consistent Global Snapshot

- Proposed by Chandy and Lamport (CL)
- Assumptions and Features:
 - channels satisfy first-in-first-out (FIFO) property
 - channels are not required to be bidirectional
 - communication topology may not be fully connected
- Messages exchanged by the underlying computation (whose snapshot is being recorded) are called application messages

Recording a Consistent Global Snapshot

- Proposed by Chandy and Lamport (CL)
- Assumptions and Features:
 - channels satisfy first-in-first-out (FIFO) property
 - channels are not required to be bidirectional
 - communication topology may not be fully connected
- Messages exchanged by the underlying computation (whose snapshot is being recorded) are called application messages
- Messages exchanged by the snapshot algorithm are called control messages

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
 State of a Distributed System
 State of a Distributed System Events and Local States
 When is a Global State
Meaningful? Revisiting Happened-Before
Relation
♦ A Consistent Global State
Recording a Consistent
Global Snapshot
The CL Protocol
The CL Protocol (Continued)
The CL Protocol: An
Illustration
The CL Protocol: An
Illustration (Continued)
Monitoring a Distributed
System



	Inherent Limitations
Initially: all processes are colored blue	Ordering of Franks
	Ordering of Events
	Abstract Clocks
Eventually: all processes become red	
	Ordering of Messages
	Otata af a Distributed Ocatara
	State of a Distributed System State of a Distributed System
	 Events and Local States
	 When is a Global State
	Meaningful?
	Revisiting Happened-Before
	Relation
	 Recording a Consistent
	Global Snapshot
	♦ The CL Protocol
	The CL Protocol (Continued)
	The CL Protocol: An
	Illustration
	Illustration (Continued)
	()
	Monitoring a Distributed
	System

	Inherent Limitations
Initially: all processes are colored blue	Ordering of Events
Eventually: all processes become red	Abstract Clocks Ordering of Messages
On changing color from blue to red: record local snapshot send a marker message along all outgoing channels	 State of a Distributed System State of a Distributed System Events and Local States When is a Global State Meaningful? Revisiting Happened-Before Relation A Consistent Global State
	 Recording a Consistent Global Snapshot The CL Protocol The CL Protocol (Continued) The CL Protocol: An Illustration The CL Protocol: An Illustration (Continued) Monitoring a Distributed
	System

	Inherent Limitations
Initially: all processes are colored blue	Ordering of Events
Eventually: all processes become red	Abstract Clocks Ordering of Messages
On changing color from blue to red: record local snapshot cond a marker message along all outgoing channels	State of a Distributed System State of a Distributed System Events and Local States When is a Global State Meaningful?
send a marker message along all outgoing channels On receiving marker message along incoming channel C:	 Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent
if color is blue then	Global Snapshot The CL Protocol The CL Protocol (Continued)
change color from blue to red endif	 The CL Protocol: An Illustration The CL Protocol: An Illustration (Continued)
record state of channel C as application messages received along C since turning red	Monitoring a Distributed System

	Inherent Limitations
Initially: all processes are colored blue	Ordering of Events
	Abstract Clocks
Eventually: all processes become red	Ordering of Messages
On changing color from blue to red:	State of a Distributed System
- on onanging color nom side to red.	State of a Distributed System
record local snapshot	Events and Local States
	When is a Global State
send a marker message along all outgoing channels	Meaningful? Revisiting Happened-Before
On receiving marker message along incoming channel C: if color is blue then change color from blue to red endif record state of channel C as application messages	Relation A Consistent Global State Recording a Consistent Global Snapshot The CL Protocol The CL Protocol (Continued) The CL Protocol: An Illustration The CL Protocol: An Illustration (Continued) Monitoring a Distributed System
received along C since turning red	Oystem
Any process can initiate the snapshot protocol by spontaneously changing its color from blue to red	

	Inherent Limitations
Initially: all processes are colored blue	Ordering of Events
Eventually: all processes become red	Abstract Clocks Ordering of Messages
On changing color from blue to red: record local snapshot send a marker message along all outgoing channels	 State of a Distributed System State of a Distributed System Events and Local States When is a Global State Meaningful? Revisiting Happened-Before
On receiving marker message along incoming channel C: if color is blue then	Relation A Consistent Global State Recording a Consistent Global Snapshot The CL Protocol
change color from blue to red endif	 The CL Protocol (Continued) The CL Protocol: An Illustration The CL Protocol: An Illustration (Continued)
record state of channel C as application messages received along C since turning red	Monitoring a Distributed System
Any process can initiate the snapshot protocol by spontaneously changing its color from blue to red	
 there can be multiple initiators of the snapshot protocol 	

	Inherent Limitations
Global Snapshot: local snapshots of processes just after	Ordering of Events
they turn red	Abstract Clocks
	Ordering of Messages
	State of a Distributed Sustan
	State of a Distributed System
	 State of a Distributed System
	Events and Local States
	When is a Global State Meaningful?
	Revisiting Happened-Before
	Relation
	A Consistent Global State
	Recording a Consistent
	Global Snapshot
	The CL Protocol
	The CL Protocol (Continued)
	✤ The CL Protocol: An
	Illustration
	The CL Protocol: An
	Illustration (Continued)
	Monitoring a Distributed
	System

	Inherent Limitations
Global Snapshot: local snapshots of processes just after	Ordering of Events
they turn red	Abstract Clocks
n-Transit Messages: blue application messages received	Ordering of Messages
by processes after they have turned red	State of a Distributed System
by processes are iney have willed red	 State of a Distributed System Events and Local States
	 When is a Global State
	Meaningful?
	Revisiting Happened-Before
	Relation
	Recording a Consistent
	Global Snapshot
	♦ The CL Protocol
	 ♦ The CL Protocol (Continued) ♦ The CL Protocol: An
	Illustration
	♦ The CL Protocol: An
	Illustration (Continued)
	Monitoring a Distributed
	System

Blobal Snapshot: local snapshots of processes just after hey turn red	Inherent Limitations Ordering of Events Abstract Clocks
n-Transit Messages: blue application messages received y processes after they have turned red Why is the global snapshot consistent?	Ordering of Messages State of a Distributed System State of a Distributed System Events and Local States When is a Global State Meaningful? Revisiting Happened-Before Relation A Consistent Global State Global Snapshot The CL Protocol The CL Protocol: An Illustration The CL Protocol: An Illustration (Continued) Monitoring a Distributed System

l Ir

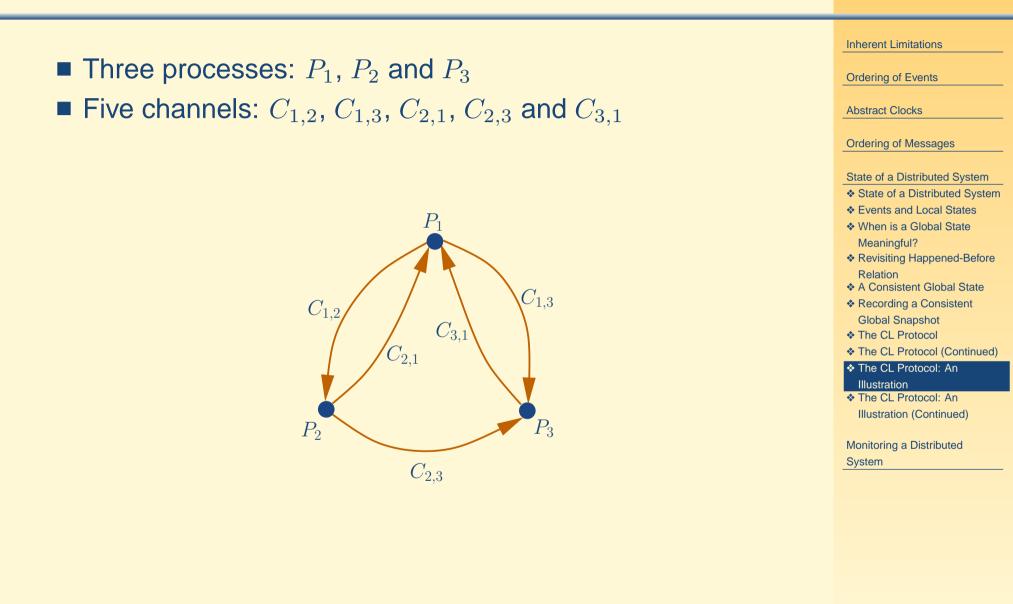
t

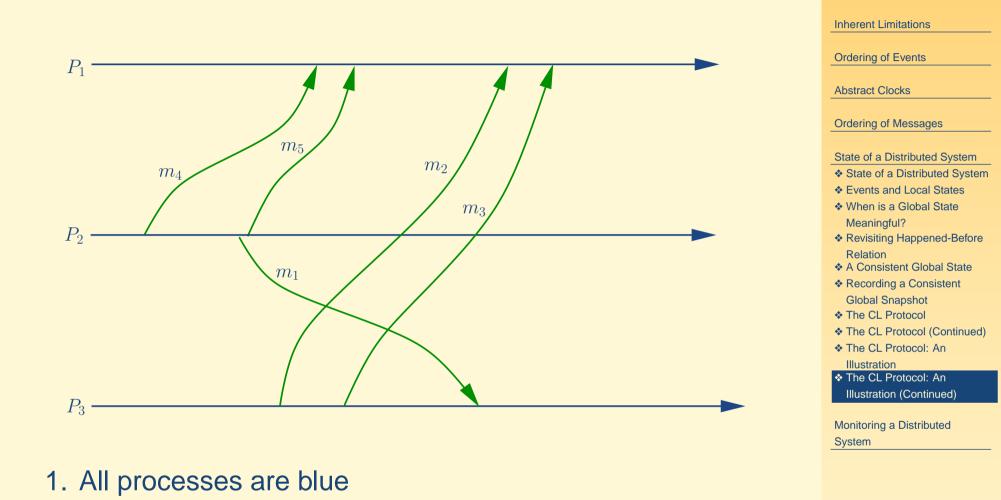
b

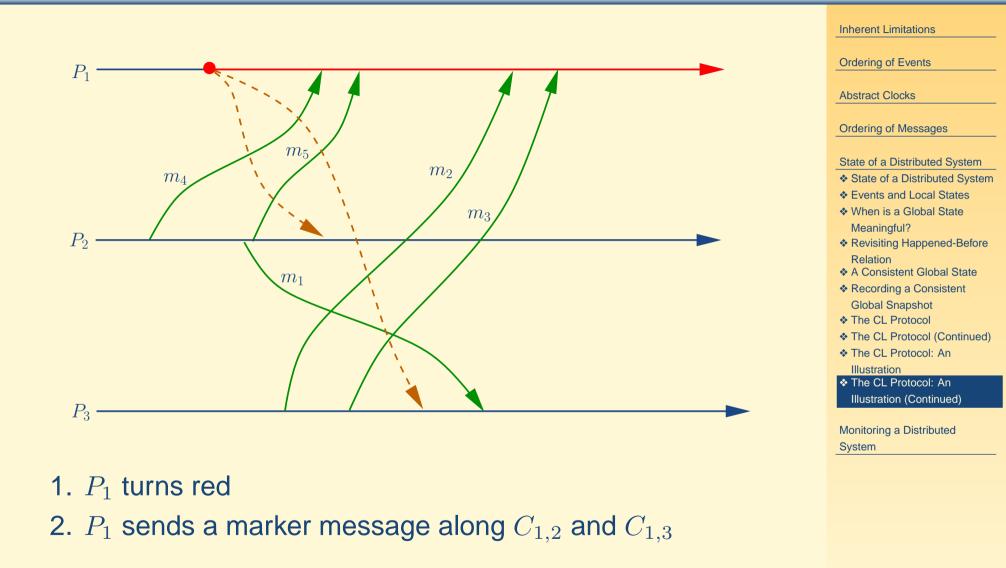
Global Snapshot: local snapshots of processes just after they turn red	Inherent Limitations Ordering of Events Abstract Clocks
In-Transit Messages: blue application messages received by processes after they have turned red	Ordering of Messages State of a Distributed System State of a Distributed System Events and Local States
 Why is the global snapshot consistent? Assume an application message has the same color as its sender 	 When is a Global State Meaningful? Revisiting Happened-Before Relation A Consistent Global State Recording a Consistent Global Snapshot The CL Protocol The CL Protocol (Continued) The CL Protocol: An Illustration The CL Protocol: An Illustration (Continued)
	Monitoring a Distributed System

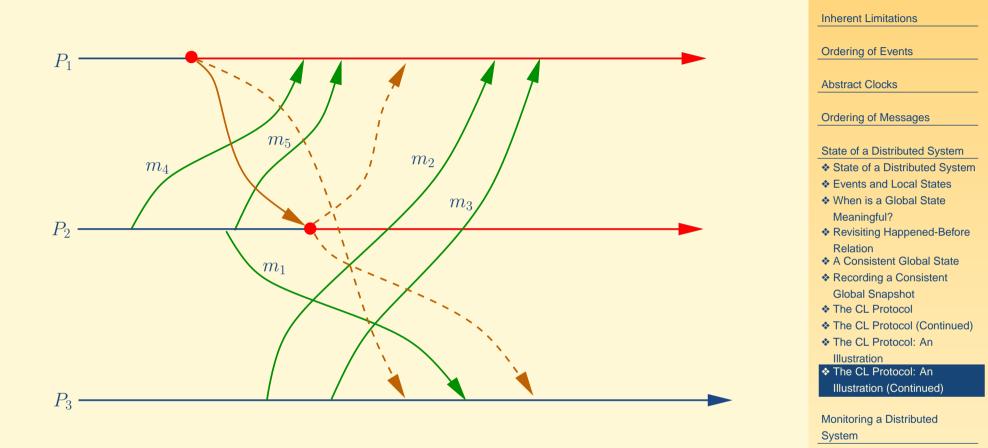
	Inherent Limitations
Global Snapshot: local snapshots of processes just after	Ordering of Events
they turn red	Abstract Clocks
n-Transit Messages: blue application messages received	Ordering of Messages
	State of a Distributed System
by processes after they have turned red	State of a Distributed System
	 Events and Local States When is a Olahal State
Why is the global snapshot consistent?	When is a Global State Meaningful?
why is the global shapshot consistent:	 Revisiting Happened-Before
 Assume an application message has the same color as 	Relation
 Assume an application message has the same color as 	 A Consistent Global State Describer - Operations
its sender	Recording a Consistent Global Snapshot
	♦ The CL Protocol
 Can a blue process receive a red application message? 	The CL Protocol (Continued)
ean a blac process recents a rea application modeage.	The CL Protocol: An
	Illustration The CL Protocol: An
	Illustration (Continued)
	(,
	Monitoring a Distributed
	System

The CL Protocol: An Illustration

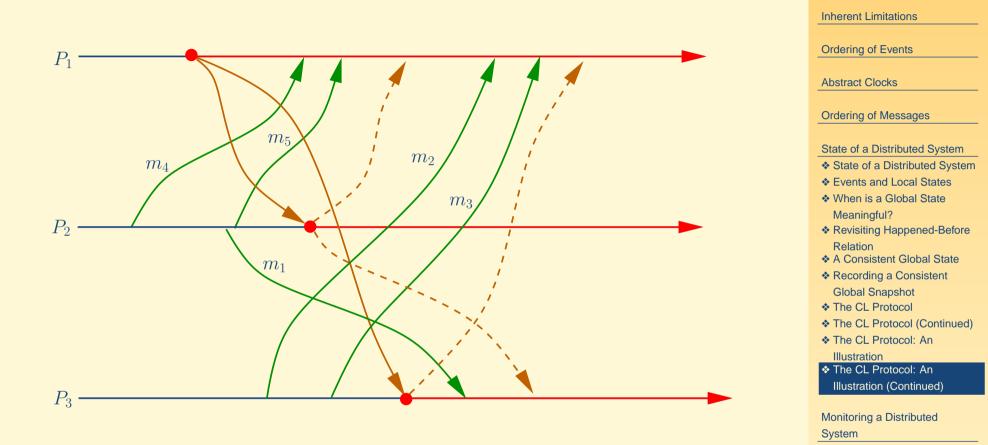




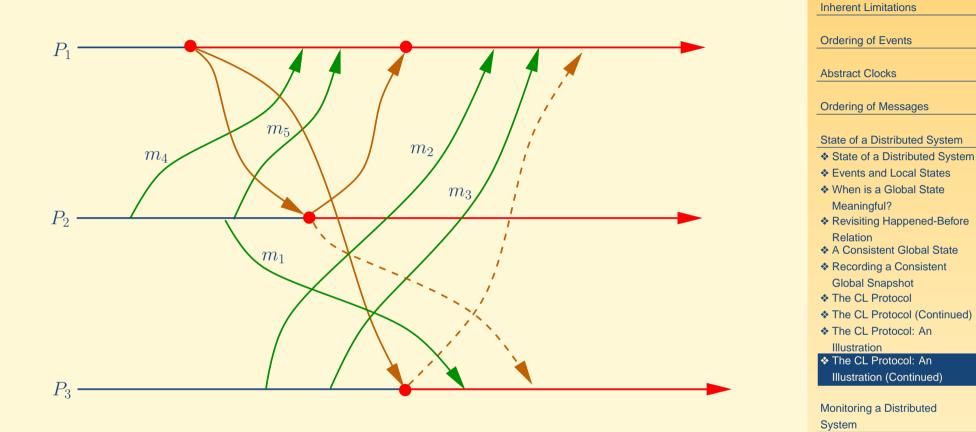




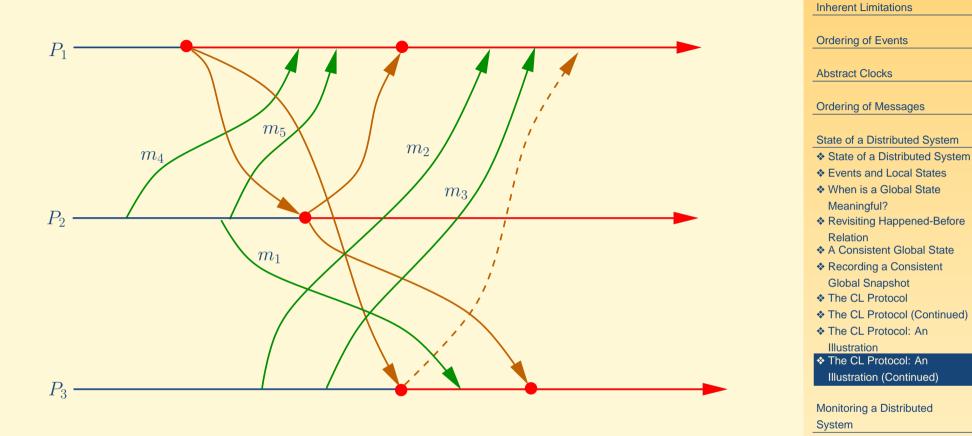
- 1. P_2 receives the marker message along $C_{1,2}$ and turns red
- **2.** P_2 sends a marker message along $C_{2,1}$ and $C_{2,3}$
- **3.** P_2 records the state of $C_{1,2}$ as \emptyset



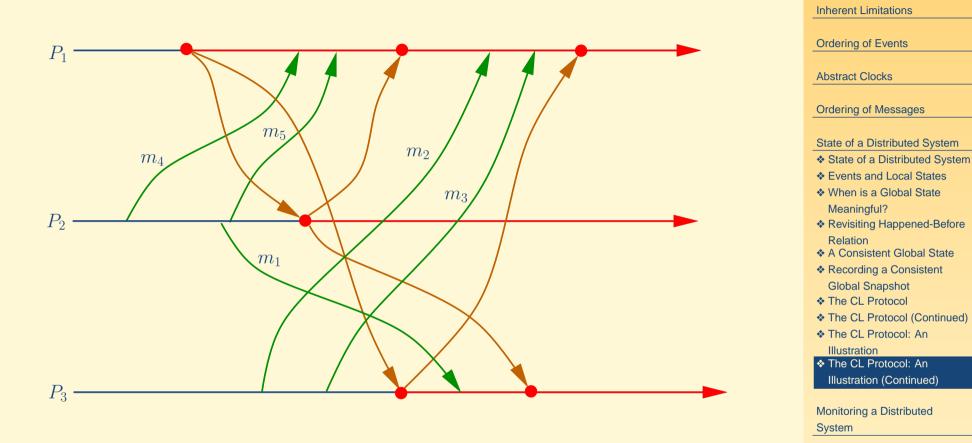
- 1. P_3 receives the marker message along $C_{1,3}$ and turns red
- **2.** P_3 sends a marker message along $C_{3,1}$
- **3.** P_3 records the state of $C_{1,3}$ as \emptyset



1. P_1 receives the marker message along $C_{2,1}$ 2. P_1 records the state of $C_{2,1}$ as $\{m_4, m_5\}$



1. P_3 receives the marker message along $C_{2,3}$ 2. P_3 records the state of $C_{2,3}$ as $\{m_1\}$



1. P_1 receives the marker message along $C_{3,1}$ 2. P_1 records the state of $C_{3,1}$ as $\{m_2, m_3\}$

Many distributed computations obey the following paradigm:

- Many distributed computations obey the following paradigm:
 - A process is either in active state or passive state

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed
System
♦ A Subclass of Distributed
Computation
Illustration
Termination Detection
 Huang's Algorithm Huang's Algorithm
 Huang's Algorithm (Continued)
 Huang's Algorithm: An
Illustration

- Many distributed computations obey the following paradigm:
 - A process is either in active state or passive state
 - A process can send an application message only when it is active

Inherent Limitations

Ordering of Events

Ordering of Messages

Monitoring a Distributed

Computation

Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An

Illustration

A Subclass of Distributed

Distributed Computation: An

State of a Distributed System

Abstract Clocks

System

- Many distributed computations obey the following paradigm:
 - A process is either in active state or passive state
 - A process can send an application message only when it is active
 - An active process can become passive at any time

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed System
♦ A Subclass of Distributed
Computation Distributed Computation: An
Illustration
Illustration
Illustration Termination Detection Huang's Algorithm Huang's Algorithm
Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued)
Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued)
Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An

- Many distributed computations obey the following paradigm:
 - A process is either in active state or passive state
 - A process can send an application message only when it is active
 - An active process can become passive at any time
 - A passive process, on receiving an application message, becomes active

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed
System
 A Subclass of Distributed Computation
 Distributed Computation: An
V DISTINUTED COMPUTATION. AN
Illustration
Illustration ◆ Termination Detection ◆ Huang's Algorithm ◆ Huang's Algorithm (Continued)
Illustration ◆ Termination Detection ◆ Huang's Algorithm ◆ Huang's Algorithm (Continued) ◆ Huang's Algorithm: An
Illustration ◆ Termination Detection ◆ Huang's Algorithm ◆ Huang's Algorithm (Continued) ◆ Huang's Algorithm: An
Illustration ◆ Termination Detection ◆ Huang's Algorithm ◆ Huang's Algorithm (Continued) ◆ Huang's Algorithm: An

- Many distributed computations obey the following paradigm:
 - A process is either in active state or passive state
 - A process can send an application message only when it is active
 - An active process can become passive at any time
 - A passive process, on receiving an application message, becomes active
- Intuitively:

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed
System
♦ A Subclass of Distributed
Computation
Distributed Computation: An
Illustration
Termination Detection
Huang's Algorithm
Huang's Algorithm
(Continued)
Huang's Algorithm: An
Illustration
Illustration
Illustration
Illustration

- Many distributed computations obey the following paradigm:
 - A process is either in active state or passive state
 - A process can send an application message only when it is active
 - An active process can become passive at any time
 - A passive process, on receiving an application message, becomes active

Intuitively:

if a process is active, then it is doing some work

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed
System
♦ A Subclass of Distributed
Computation
Computation
Distributed Computation: An
 Distributed Computation: An Illustration
 Distributed Computation: An Illustration Termination Detection
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued)
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued)
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An

- Many distributed computations obey the following paradigm:
 - A process is either in active state or passive state
 - A process can send an application message only when it is active
 - An active process can become passive at any time
 - A passive process, on receiving an application message, becomes active

Intuitively:

- if a process is active, then it is doing some work
- if process is passive, then it is idle

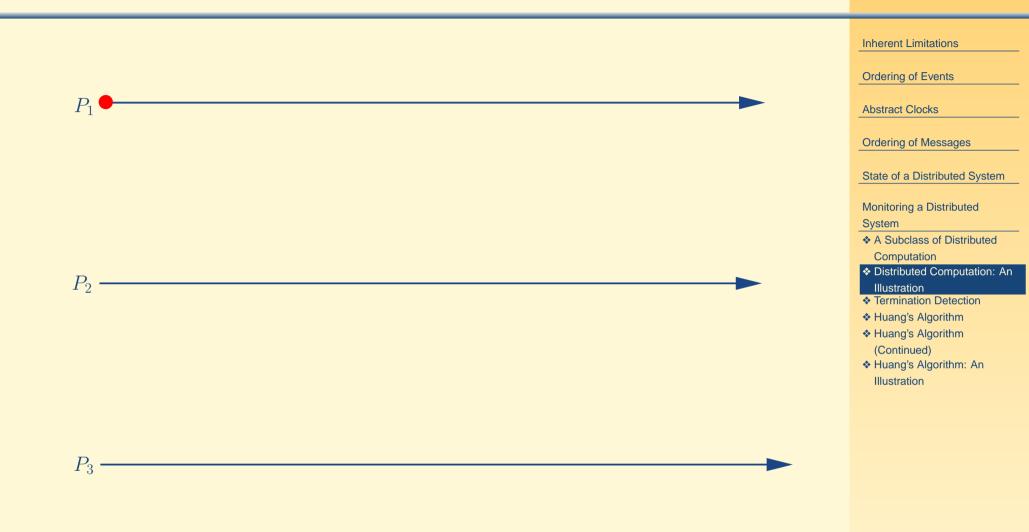
	Inherent Limitations
	Ordering of Events
	Abstract Clocks
	Ordering of Messages
-	
-	State of a Distributed System
	Monitoring a Distributed
	System
	♦ A Subclass of Distributed
	Computation
	 Distributed Computation: An Illustration
	Distributed Computation: An
	 Distributed Computation: An Illustration
	 Distributed Computation: An Illustration Termination Detection
	 Distributed Computation: An Illustration Termination Detection Huang's Algorithm
	 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm
	 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued)
	 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
	 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
	 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
	 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An

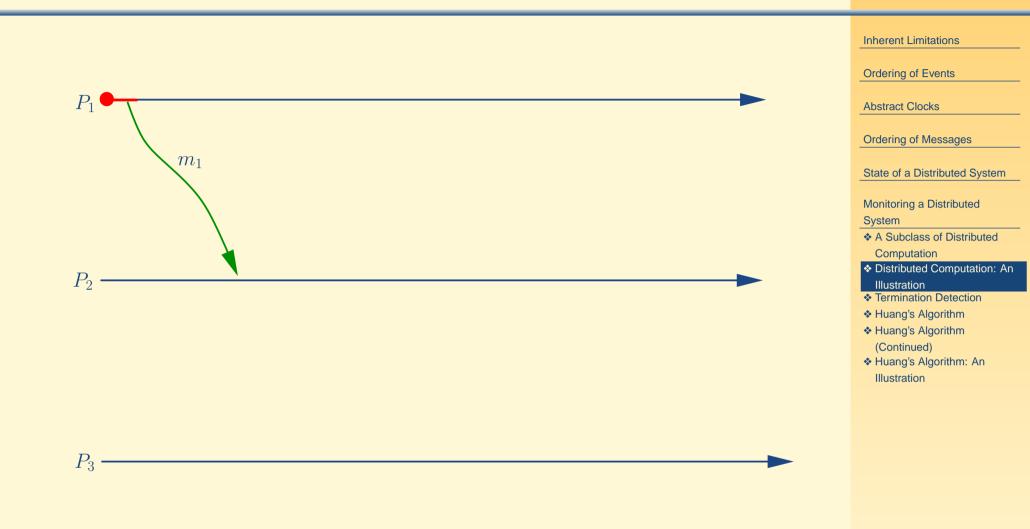
- Many distributed computations obey the following paradigm:
 - A process is either in active state or passive state
 - A process can send an application message only when it is active
 - An active process can become passive at any time
 - A passive process, on receiving an application message, becomes active

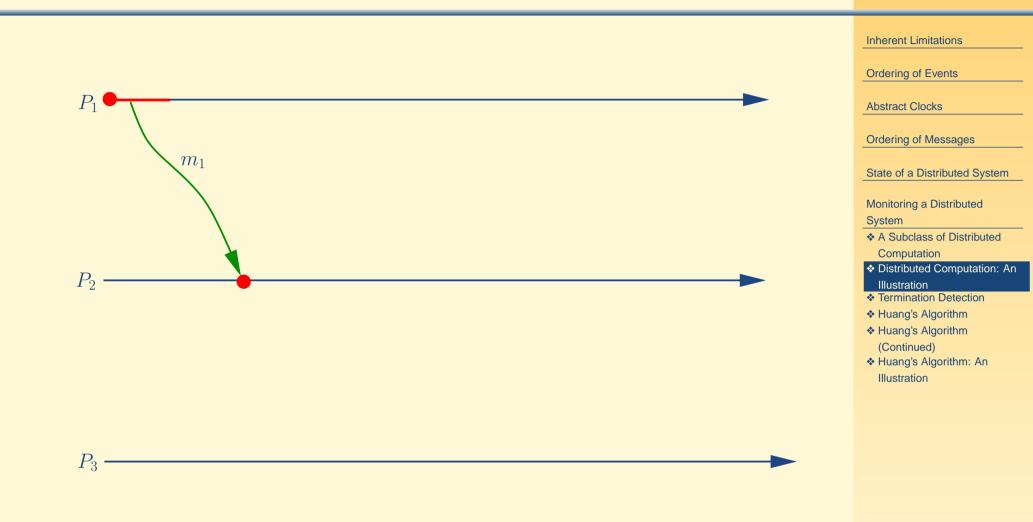
Intuitively:

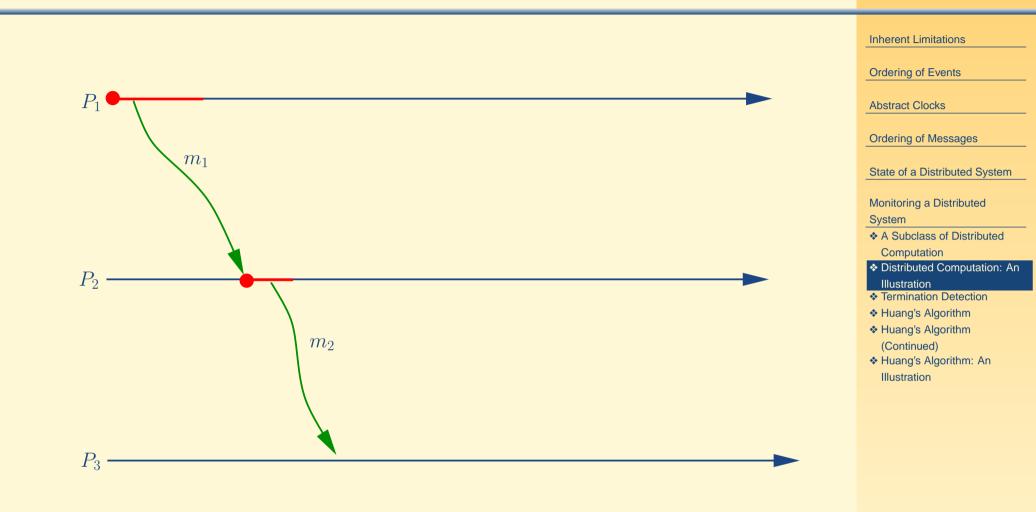
- if a process is active, then it is doing some work
- if process is passive, then it is idle
- an active process uses an application message to send a part of its work to another process

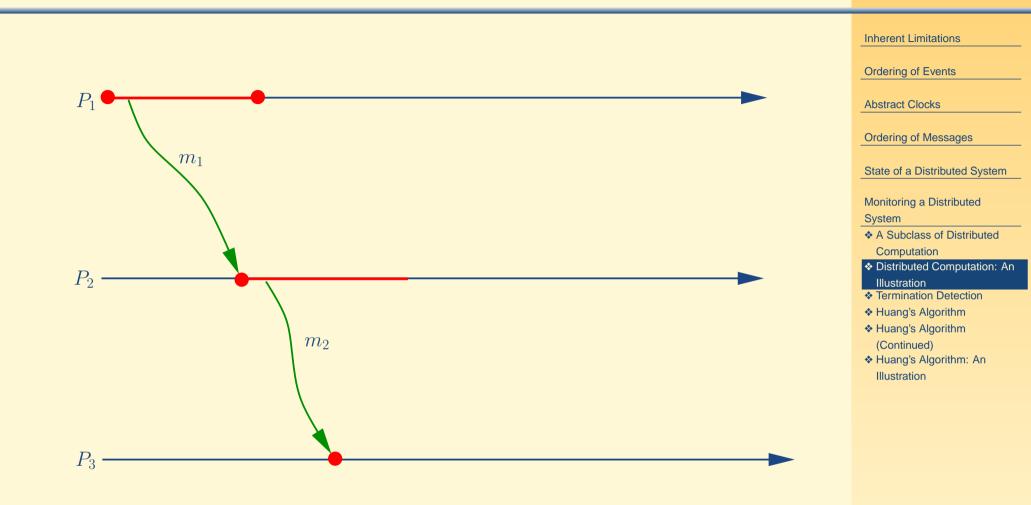
Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed
System
A Subclass of Distributed
Computation
Computation
 Distributed Computation: An Illustration
Distributed Computation: An
 Distributed Computation: An Illustration
 Distributed Computation: An Illustration Termination Detection
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued)
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
 Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An

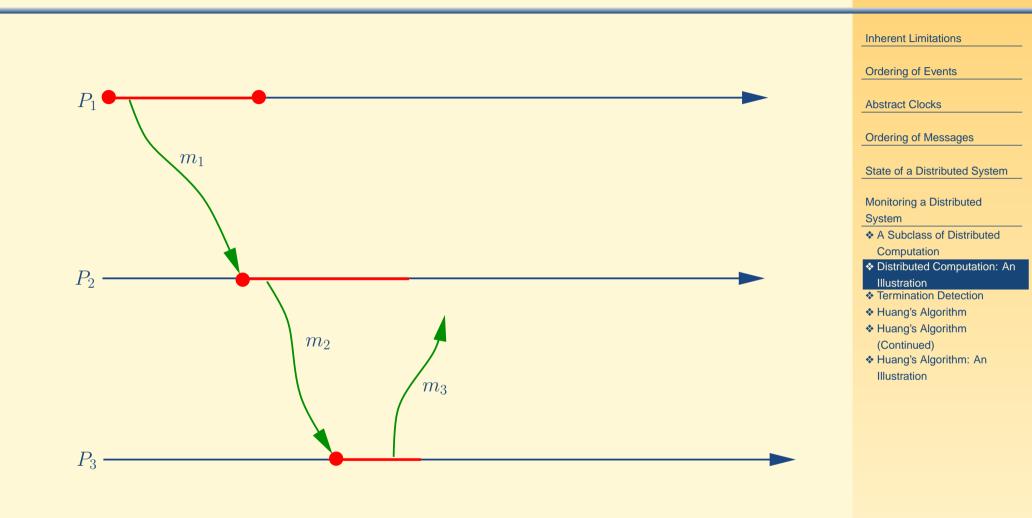


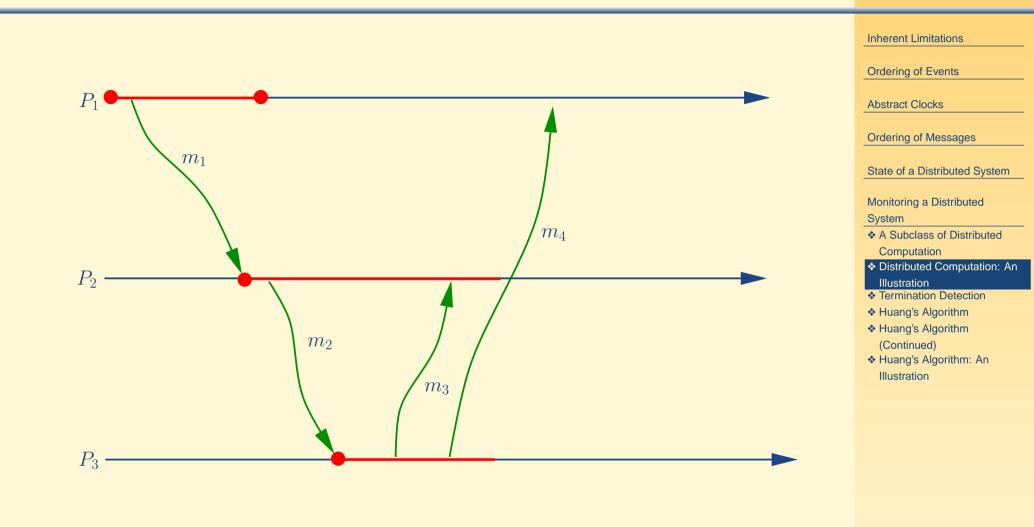


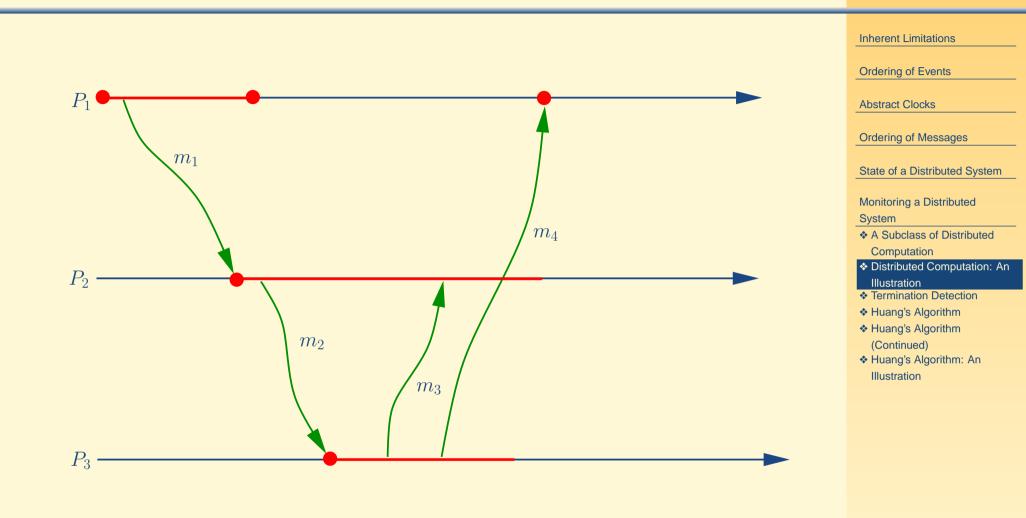


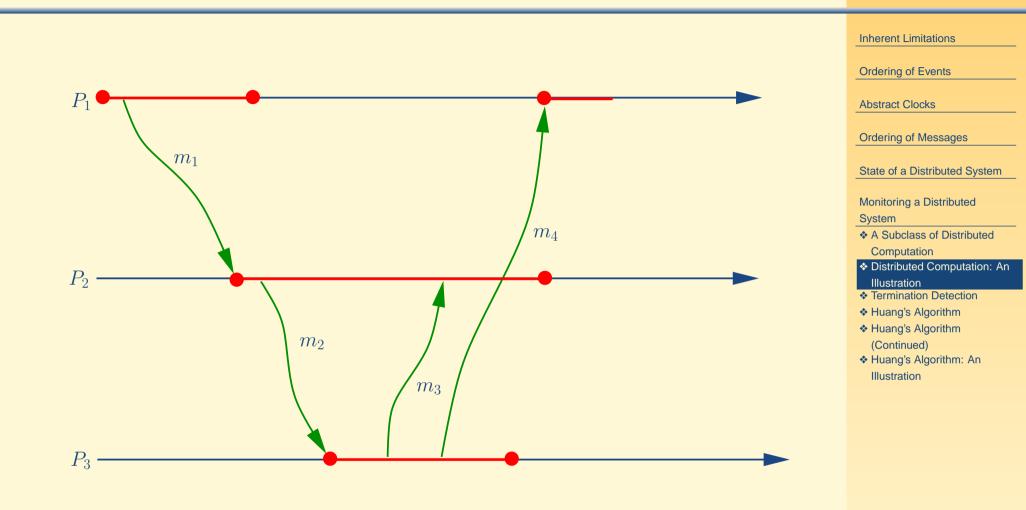


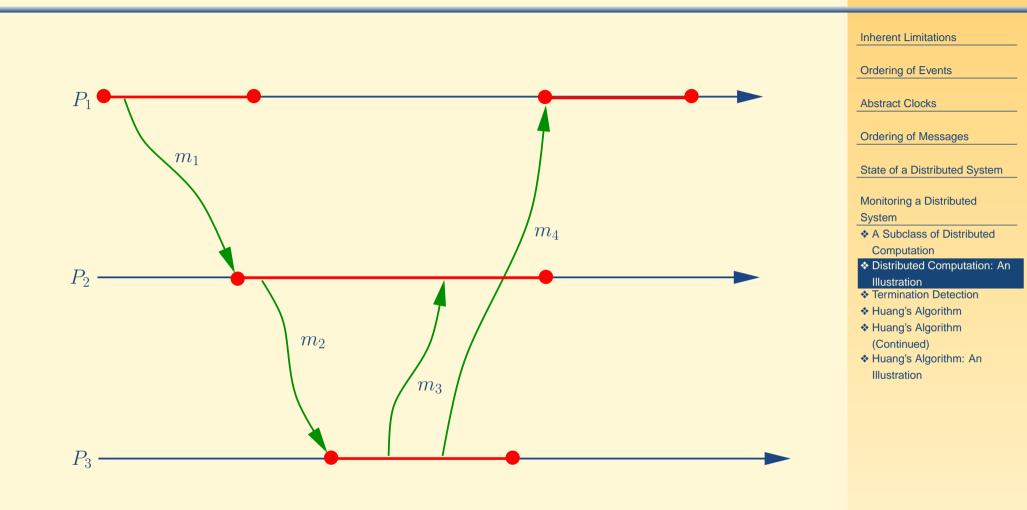












To detect if the computation has finished doing all the work	Inherent Limitations Ordering of Events
	Abstract Clocks
	Ordering of Messages
	State of a Distributed System
	Monitoring a Distributed System * A Subclass of Distributed Computation * Distributed Computation: An Illustration * Termination Detection * Huang's Algorithm (Continued) * Huang's Algorithm: An Illustration

	Inherent Limitations
To detect if the computation has finished doing all the work	Ordering of Events
 all processes have become passive, and 	Abstract Clocks
• all processes have become passive, and	Ordering of Messages
	State of a Distributed System
	Monitoring a Distributed System * A Subclass of Distributed Computation * Distributed Computation: An Illustration * Termination Detection * Huang's Algorithm (Continued) * Huang's Algorithm: An Illustration

	Inherent Limitations
To detect if the computation has finished doing all the work	Ordering of Events
 all processes have become passive, and 	Abstract Clocks
 all channels have become empty 	Ordering of Messages
• all chamers have become empty	State of a Distributed System
	Monitoring a Distributed System A Subclass of Distributed Computation Distributed Computation: An Illustration Termination Detection Huang's Algorithm (Continued) Huang's Algorithm: An Illustration

	Inherent Limitations
To detect if the computation has finished doing all the work	Ordering of Events
 all processes have become passive, and 	Abstract Clocks
 all channels have become empty 	Ordering of Messages
	State of a Distributed System
Different types of computations:	Monitoring a Distributed System A Subclass of Distributed
	 A Subclass of Distributed Computation Distributed Computation: An
	Illustration
	Huang's Algorithm
	 Huang's Algorithm (Continued)
	 Huang's Algorithm: An Illustration

	Inherent Limitations
To detect if the computation has finished doing all the work	Ordering of Events
 all processes have become passive, and 	Abstract Clocks Ordering of Messages
 all channels have become empty 	State of a Distributed System
Different types of computations:	System
 diffusing: only one process is active in the beginning 	Computation Computation: An Illustration Termination Detection
	 Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An Illustration

	Inherent Limitations
To detect if the computation has finished doing all the work	Ordering of Events
 all processes have become passive, and 	Abstract Clocks
 all channels have become empty 	Ordering of Messages
 Different types of computations: diffusing: only one process is active in the beginning non-diffusing: any subset of processes can be active in the beginning 	State of a Distributed System Monitoring a Distributed System A Subclass of Distributed Computation Distributed Computation: An Illustration Termination Detection Huang's Algorithm (Continued) Huang's Algorithm: An Illustration

	Inherent Limitations
To detect if the computation has finished doing all the work	Ordering of Events
 all processes have become passive, and 	Abstract Clocks
 all channels have become empty 	Ordering of Messages State of a Distributed System
Different types of computations:	Monitoring a Distributed System A Subclass of Distributed
 diffusing: only one process is active in the beginning 	Computation Computation: An Illustration
 non-diffusing: any subset of processes can be active in the beginning 	 Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An
no process knows which processes are active and which processes are passive	Illustration

Assumption: computation is diffusing	Inherent Limitations Ordering of Events
	Abstract Clocks
	Ordering of Messages
	State of a Distributed System
	Monitoring a Distributed System * A Subclass of Distributed Computation * Distributed Computation: An Illustration * Termination Detection * Huang's Algorithm (Continued) * Huang's Algorithm: An Illustration

 Assumption: computation is diffusing the initially active process is called the coordinator 	Inherent Limitations Ordering of Events Abstract Clocks Ordering of Messages
	State of a Distributed System Monitoring a Distributed System A Subclass of Distributed Computation Distributed Computation: An Illustration Termination Detection Huang's Algorithm (Continued) Huang's Algorithm: An Illustration

 Assumption: computation is diffusing the initially active process is called the coordinator coordinator is responsible for detecting termination 	Inherent LimitationsOrdering of EventsAbstract ClocksOrdering of MessagesState of a Distributed SystemMonitoring a Distributed System* A Subclass of Distributed Computation* Distributed Computation: An Illustration* Termination Detection* Huang's Algorithm(Continued)* Huang's Algorithm: An Illustration

 Assumption: computation is diffusing the initially active process is called the coordinator coordinator is responsible for detecting termination 	Inherent Limitations Ordering of Events Abstract Clocks Ordering of Messages State of a Distributed System
Initially:	Monitoring a Distributed System A Subclass of Distributed Computation Distributed Computation: An Illustration Huang's Algorithm (Continued) Huang's Algorithm: An Illustration

 Assumption: computation is diffusing the initially active process is called the coordinator coordinator is responsible for detecting termination 	
	: Limitations g of Events : Clocks g of Messages a Distributed System
 Coordinator has a weight of 1 System A Sub Comp Distribullustra Termir Huang (Contined) 	nation Detection g's Algorithm g's Algorithm inued) g's Algorithm: An

 Assumption: computation is diffusing the initially active process is called the coordinator coordinator is responsible for detecting termination 	Inherent Limitations Ordering of Events Abstract Clocks Ordering of Messages State of a Distributed System
 Initially: coordinator has a weight of 1 all other processes have a weight of 0 	Monitoring a Distributed System * A Subclass of Distributed Computation * Distributed Computation: An Illustration * Termination Detection * Huang's Algorithm (Continued) * Huang's Algorithm: An Illustration

	nerent Limitations
Assumption: computation is diffusing order • the initially active process is called the coordinator Abstr • coordinator is responsible for detecting termination Order	dering of Events stract Clocks dering of Messages ate of a Distributed System
 coordinator has a weight of 1 all other processes have a weight of 0 <i>Invariants:</i> 	anitoring a Distributed stem A Subclass of Distributed Computation Distributed Computation: An Illustration Termination Detection Huang's Algorithm (Continued) Huang's Algorithm: An Illustration

 Assumption: computation is diffusing the initially active process is called the coordinator coordinator is responsible for detecting termination 	Inherent Limitations Ordering of Events Abstract Clocks Ordering of Messages State of a Distributed System
 Initially: coordinator has a weight of 1 all other processes have a weight of 0 	Monitoring a Distributed System A Subclass of Distributed Computation Distributed Computation: An Illustration Termination Detection Huang's Algorithm Huang's Algorithm
 Invariants: total amount of weight in the system is 1 	(Continued) ♦ Huang's Algorithm: An Illustration

Inherent Limitations Assumption: computation is diffusing Ordering of Events the initially active process is called the coordinator Abstract Clocks Ordering of Messages coordinator is responsible for detecting termination State of a Distributed System Monitoring a Distributed System A Subclass of Distributed coordinator has a weight of 1 Computation Distributed Computation: An Illustration all other processes have a weight of 0 Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Invariants: Huang's Algorithm: An Illustration total amount of weight in the system is 1 a non-coordinator process has a non-zero weight *if and* only if it is active

Initially:

Inherent Limitations Assumption: computation is diffusing Ordering of Events the initially active process is called the coordinator Abstract Clocks Ordering of Messages coordinator is responsible for detecting termination State of a Distributed System Monitoring a Distributed System A Subclass of Distributed coordinator has a weight of 1 Computation Distributed Computation: An Illustration all other processes have a weight of 0 Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An Illustration total amount of weight in the system is 1 a non-coordinator process has a non-zero weight if and a channel has a non-zero weight if and only if it is

Initially:

Invariants:

only if it is active

non-empty

Inherent Limitations Assumption: computation is diffusing Ordering of Events the initially active process is called the coordinator Abstract Clocks Ordering of Messages coordinator is responsible for detecting termination State of a Distributed System Monitoring a Distributed System A Subclass of Distributed coordinator has a weight of 1 Computation Distributed Computation: An Illustration all other processes have a weight of 0 Termination Detection Huang's Algorithm Huang's Algorithm (Continued) Huang's Algorithm: An Illustration total amount of weight in the system is 1 a non-coordinator process has a non-zero weight if and a channel has a non-zero weight if and only if it is

weight of a channel is the sum of the weight of all its messages

Initially:

Invariants:

only if it is active

non-empty

	Inherent Limitations
Actions:	
	Ordering of Events
	Abstract Clocks
	Ordering of Messages
	State of a Distributed System
	Marchanian - Distributed
	Monitoring a Distributed
	System
	A Subclass of Distributed
	Computation
	Illustration
	✤ Termination Detection
	Huang's Algorithm
	♦ Huang's Algorithm
	(Continued)
	✤ Huang's Algorithm: An
	Illustration

Actions:

On sending an application message:

Inherent Limitations

Ordering of Events

Ordering of Messages

Monitoring a Distributed

Computation

Illustration Termination Detection Huang's Algorithm (Continued) Huang's Algorithm: An

Illustration

A Subclass of Distributed

Distributed Computation: An

State of a Distributed System

Abstract Clocks

System

- On sending an application message:
 - send half of its weight along with the message

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed
System * A Subclass of Distributed
Computation Distributed Computation: An
Illustration Termination Detection Huang's Algorithm
Huang's Algorithm
(Continued) Huang's Algorithm: An
Illustration

- On sending an application message:
 - send half of its weight along with the message
- On receiving an application message:

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed System
A Subclass of Distributed
Computation
Illustration
 Termination Detection
Huang's Algorithm
✤ Huang's Algorithm
(Continued)
Huang's Algorithm: An Illustration
indstration

- On sending an application message:
 - send half of its weight along with the message
- On receiving an application message:
 - add the weight of the message to the current weight

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed
System
Computation
 Distributed Computation: An
Illustration
Termination Detection
Huang's Algorithm
✤ Huang's Algorithm
(Continued) ◆ Huang's Algorithm: An
Illustration

- On sending an application message:
 - send half of its weight along with the message
- On receiving an application message:
 - add the weight of the message to the current weight
- On becoming passive:

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed
System
A Subclass of Distributed
Computation
Distributed Computation: An
Illustration
 Termination Detection
Huang's Algorithm
Huang's Algorithm
(Continued)
Huang's Algorithm: An
Illustration

- On sending an application message:
 - send half of its weight along with the message
- On receiving an application message:
 - add the weight of the message to the current weight
- On becoming passive:
 - send the current weight to the coodrinator

- On sending an application message:
 - send half of its weight along with the message
- On receiving an application message:
 - add the weight of the message to the current weight
- On becoming passive:
 - send the current weight to the coodrinator
- Coordinator announces termination once:

Ir	herent Limitations
~	
0	ordering of Events
A	bstract Clocks
0	rdering of Messages
_	
s	tate of a Distributed System
N	Ionitoring a Distributed
S	ystem
*	A Subclass of Distributed
	Computation
*	Distributed Computation: Ar
	Illustration
	Termination Detection
	Huang's Algorithm
*	Huang's Algorithm
•	(Continued) Huang's Algorithm: An
	Illustration
	indstration

- On sending an application message:
 - send half of its weight along with the message
- On receiving an application message:
 - add the weight of the message to the current weight
- On becoming passive:
 - send the current weight to the coodrinator
- Coordinator announces termination once:
 - it has become passive and

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed
System
Computation
Illustration Termination Detection
 Huang's Algorithm Huang's Algorithm
(Continued)
Huang's Algorithm: An Illustration

- On sending an application message:
 - send half of its weight along with the message
- On receiving an application message:
 - add the weight of the message to the current weight
- On becoming passive:
 - send the current weight to the coodrinator
- Coordinator announces termination once:
 - it has become passive and
 - it has collected all the weight

Inherent Limitations
Ordering of Events
Abstract Clocks
Ordering of Messages
State of a Distributed System
Monitoring a Distributed System
 A Subclass of Distributed Computation
Distributed Computation: An Illustration
 Termination Detection Huang's Algorithm
 Huang's Algorithm (Continued)
Huang's Algorithm: An Illustration
liustration

