UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Lecture 5: Causality and Logical Clocks

CS 539 / ECE 526

Distributed Algorithms

Announcements

- PS1 due Sunday
  - Further clarifications on models: adopt the models in the lectures by default (no change to problems)

Operating Systems

R. Stockton Gaines
Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocess systems

CR Categories: 4.32, 5.29

Operating Systems

R. Stockton Gaines Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed

- Widely considered the most important paper in the history of distributed computing
- "Jim Gray once told me that he had heard two different opinions of this paper: that it's trivial and that it's brilliant. I can't argue with the former, and I am disinclined to argue with the latter."

Operating Systems — R. Stockton Gaines, Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed

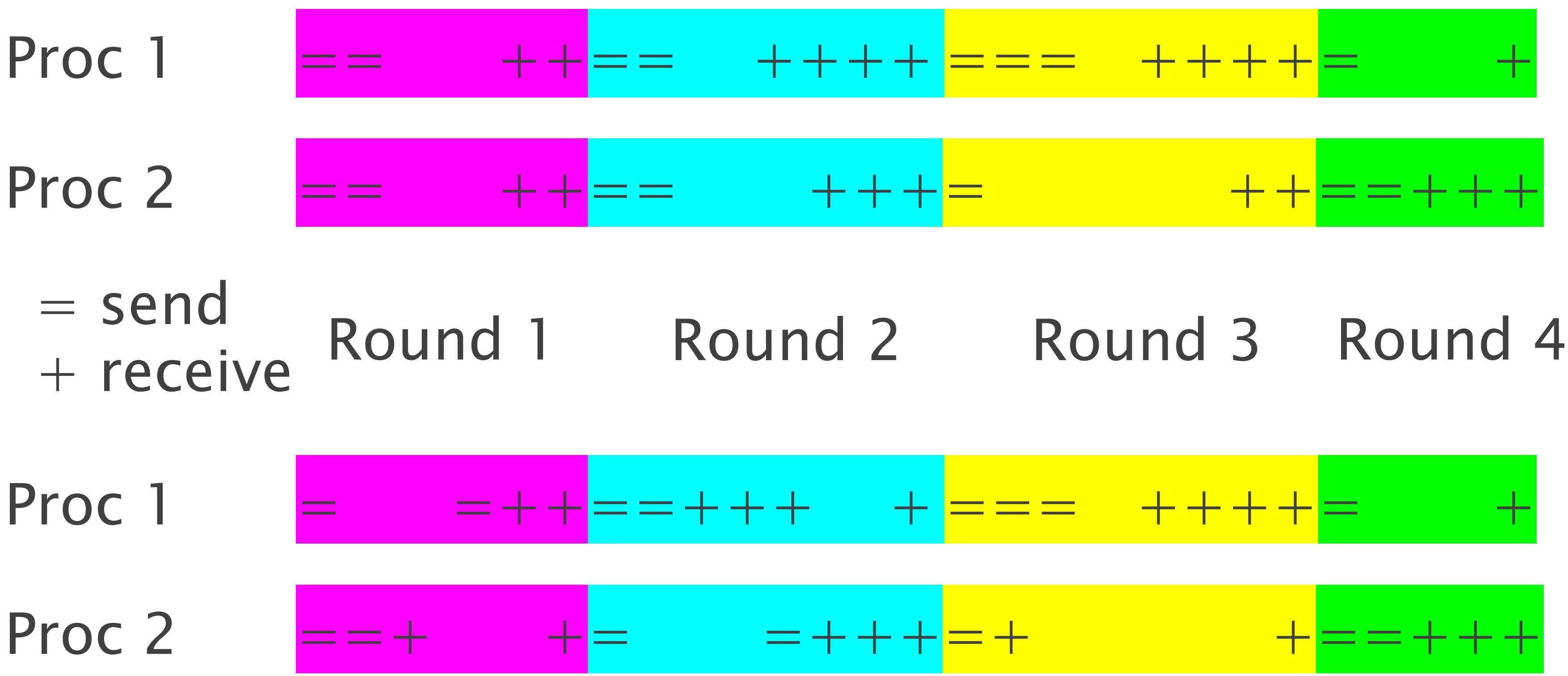
- In a nutshell, the paper says:
  - X happens before Y if, X and Y occur on the same process and X precedes Y
  - A message must be sent before it can be received
- Trivial or brilliant?

Outline

- Equivalent schedules in asynchrony
- Causality: happens-before relation
- Logical clocks
  - Lamport clock
  - Vector clock

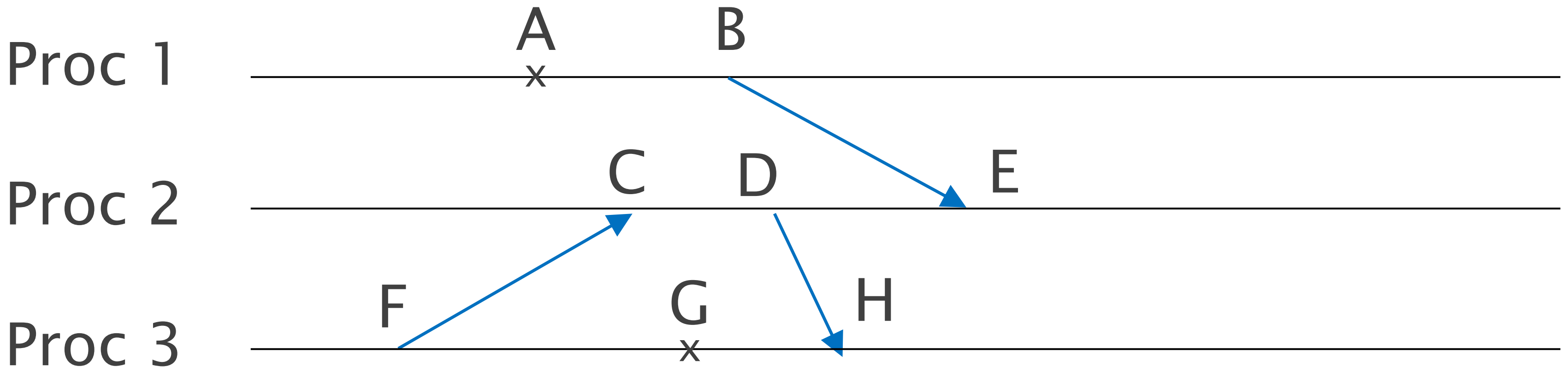
Equivalence of Executions

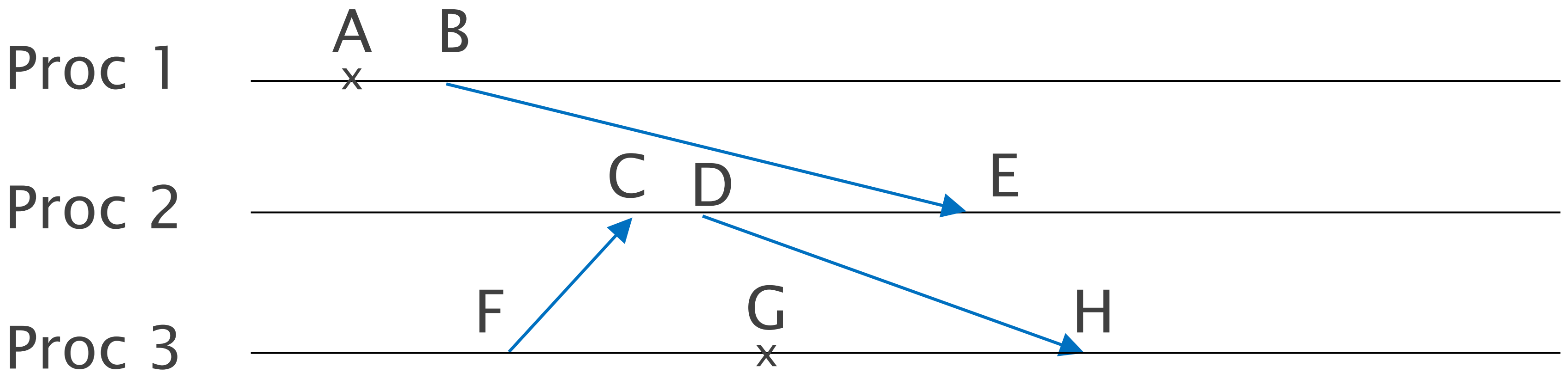
- Are the following executions equivalent?



Equivalence of Executions

- Are the following executions equivalent?





Equivalence of Executions

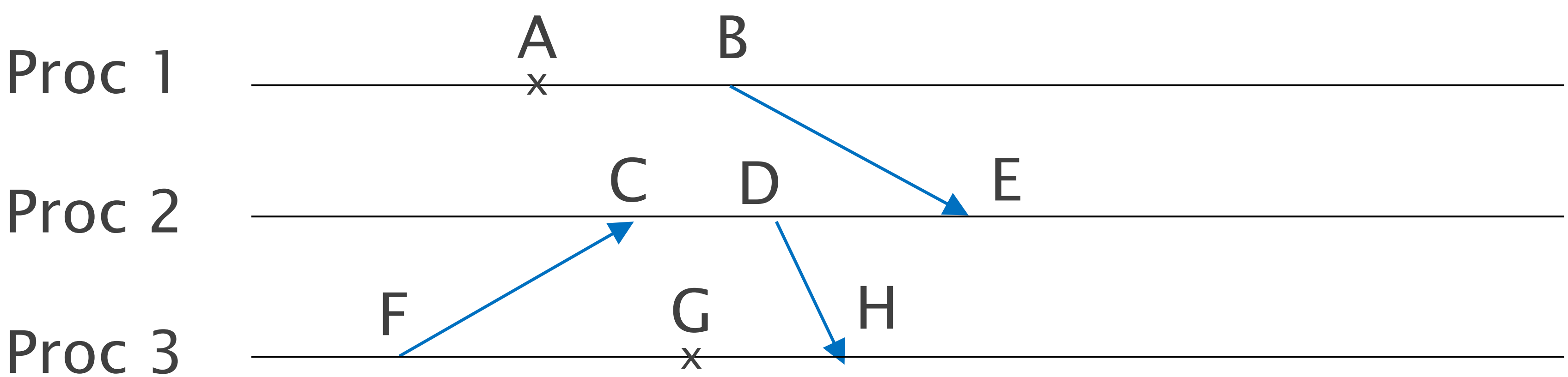
- How do we define equivalence formally?
- Two executions are equivalent if every process finds the two executions **indistinguishable**
  - Again, may not be equivalent for external observers!
- Implies identical actions by each process
- But how do we define indistinguishable?

Equivalence of Executions

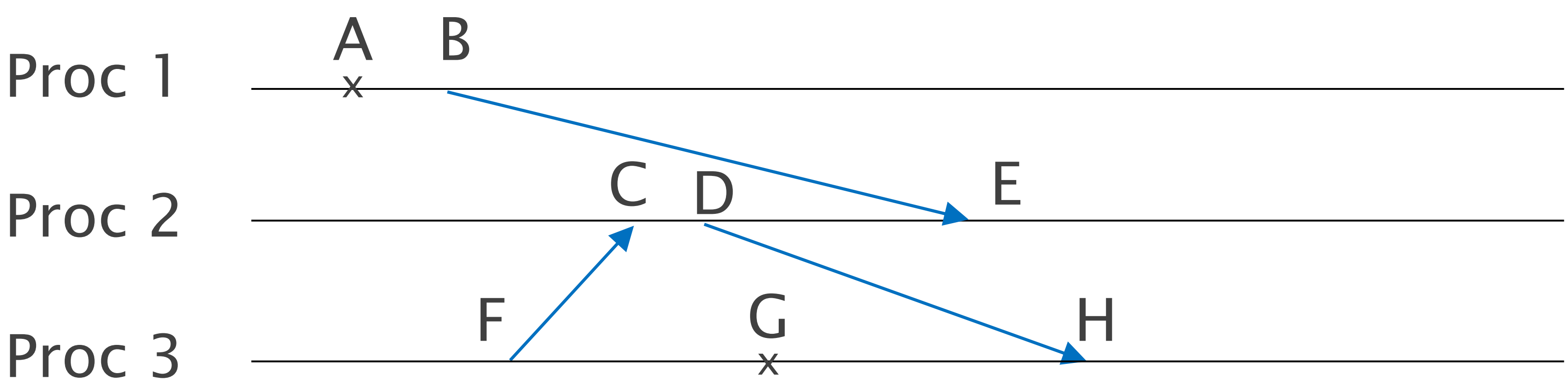
- In asynchrony, process cannot rely on time
- So indistinguishable to a process if consisting of the same **sequence** of events at that proc
  - Same set of events in the same order
- Each event can be
  - Local compute step
  - Sending a message
  - Receiving a message

Equivalence of Executions

- Are the following executions equivalent?



(3, F) (1, A) (2, C) (3, G) (1, B) (2, D) (3, H) (2, E)



(1, A) (1, B) (3, F) (2, C) (2, D) (3, G) (2, E) (3, H)

Equivalence of Schedules

- Given a global schedule of events, check each process' local schedule
- Anything else to check?

(1, A) (1, B) (2, C) (2, D) (2, E) (3, F) (3, G) (3, H)?

(3, F) (1, A) (2, C) (3, G) (1, B) (2, D) (3, H) (2, E)

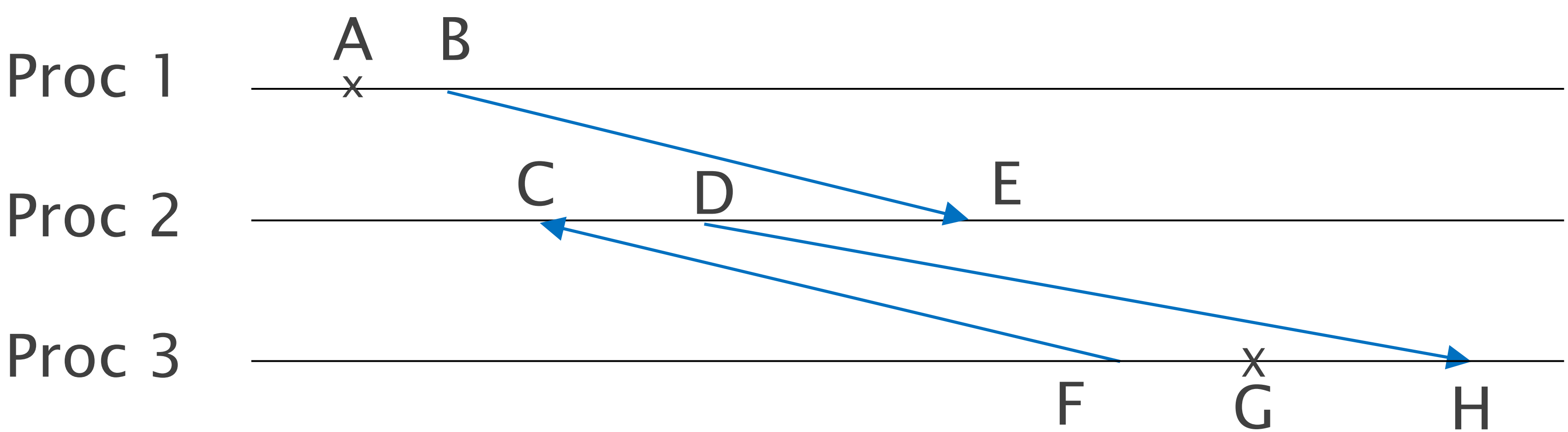
Proc 1	(1, A)	(1, B)	
Proc 2	(2, C)	(2, D)	(2, E)
Proc 3	(3, F)	(3, G)	(3, H)

(1, A) (1, B) (3, F) (2, C) (2, D) (3, G) (2, E) (3, H)

Equivalence of Schedules

- Given a global schedule of events, check each process' local schedule
- Anything else to check?

(1, A) (1, B) (2, C) (2, D) (2, E) (3, F) (3, G) (3, H)?



Equivalence

- Two **executions** are equivalent if they consist of the same sequence of events at each proc
  - Assuming both are legal executions

- Given a shuffled **schedule** of events, need to check if it is a legal / causal execution
  - Every msg is sent before received

Outline

- Equivalent schedules in asynchrony
- Causality: happens-before relation
- Logical clocks
  - Lamport clock
  - Vector clock

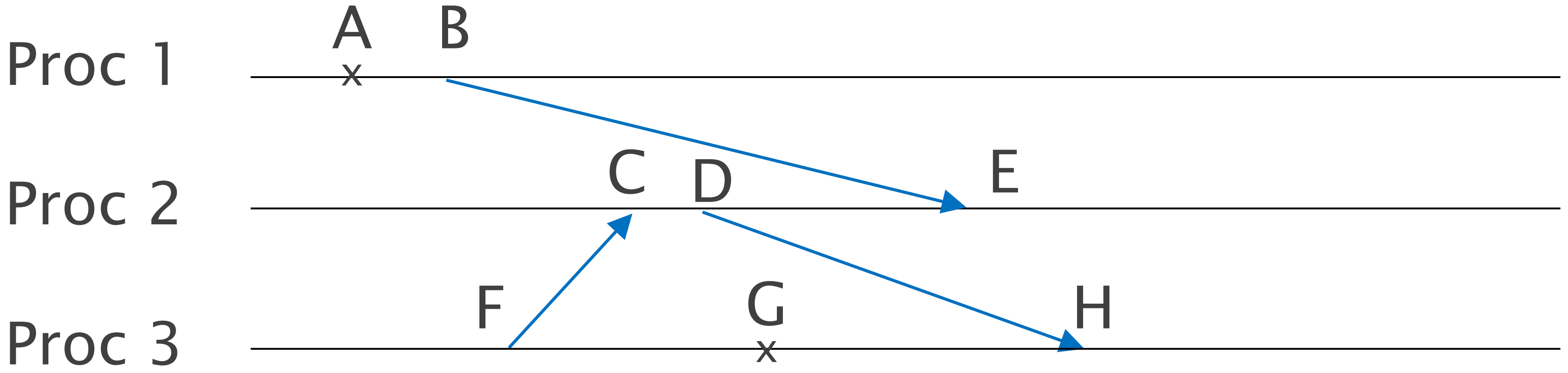
Happens Before

- Event X **happens before** event Y (X → Y) if
  1. X, Y occur at the same process, and X precedes Y;
  2. X is the send event of a message and Y is the receive event of that message; or
  3. there exists event Z such that X → Z and Z → Y.

     (transitive closure of rules 1 and 2)

Happens Before

- What happens-before relations exist below?
  - Rule 1: A → B, C → D, D → E, F → G, G → H
  - Rule 2: B → E, F → C, D → H
  - Rule 3: A → E, F → D, F → E, C → H



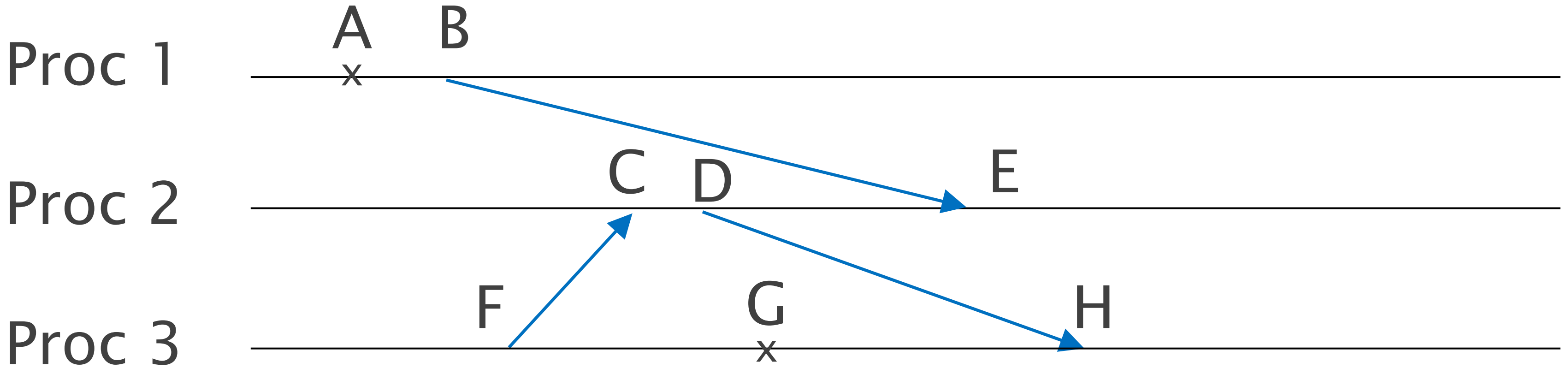
Concurrency

- Event X **happens before** event Y (X → Y) if
  1. X, Y occur at the same process, and X precedes Y;
  2. X is the send event of a message and Y is the receive event of that message; or
  3. there exists event Z such that X → Z and Z → Y.

     (transitive closure of rules 1 and 2)
- If X ↛ Y and Y ↛ X, then events X and Y are **concurrent** (X || Y)

Concurrency

- What events are concurrent below?
  - A || C, A || D, B || C, B || D
  - D || G, E || G
  - A || F, B || F, A || G, B || G, A || H, B || H



Importance of Happens-Before

- Captures causality
  - If X → Y, if X has the *potential* to influence Y
    - Y certainly does not influence X
    - Concurrent events cannot influence one another

- Fully categorize asynchronous execution

Importance of Happens-Before

- Theorem: Let S be a schedule of events in an execution. Let S’ be a permutation of S. S’ is an equivalent execution to S, if and only if S’ has identical happens-before relations as S.

Importance of Happens-Before

- S ~ S' iff identical happens-before relations.
- Proof: "If" direction
  - Identical happens-before → identical ordering at each process (Rule 1) → equivalent (That's it?)
  - S' is valid, i.e., send before receive (Rule 2)

  "Only if" direction
  - Equivalence → same ordering at each proc → Rule 1
  - S' is an execution → Rule 2 preserved

Outline

- Equivalent schedules in asynchrony
- Causality: happens-before relation
- Logical clocks
  - Lamport clock
  - Vector clock

Operating Systems

R. Stockton Gaines
Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed

- In a nutshell, the paper says:
  - X happens before Y if, X and Y occur on the same process and X precedes Y
  - A message must be sent before it can be received
- Trivial or brilliant?

Logical Clock

- Can we assign labels to events to help keep track of their happens-before relations?
- These labels are called **logical clocks**
  - Also refer to the algorithms to assign them
- Ideally, X → Y if and only if Label(X) < Label(Y)

Lamport Clock

- Each proc keeps a local counter (initially 0)
- Increment upon and assign to each local event
  - Sufficient to capture Rule 1 happens-before
- Each msg carries the LC of its send event
- Upon receiving a msg

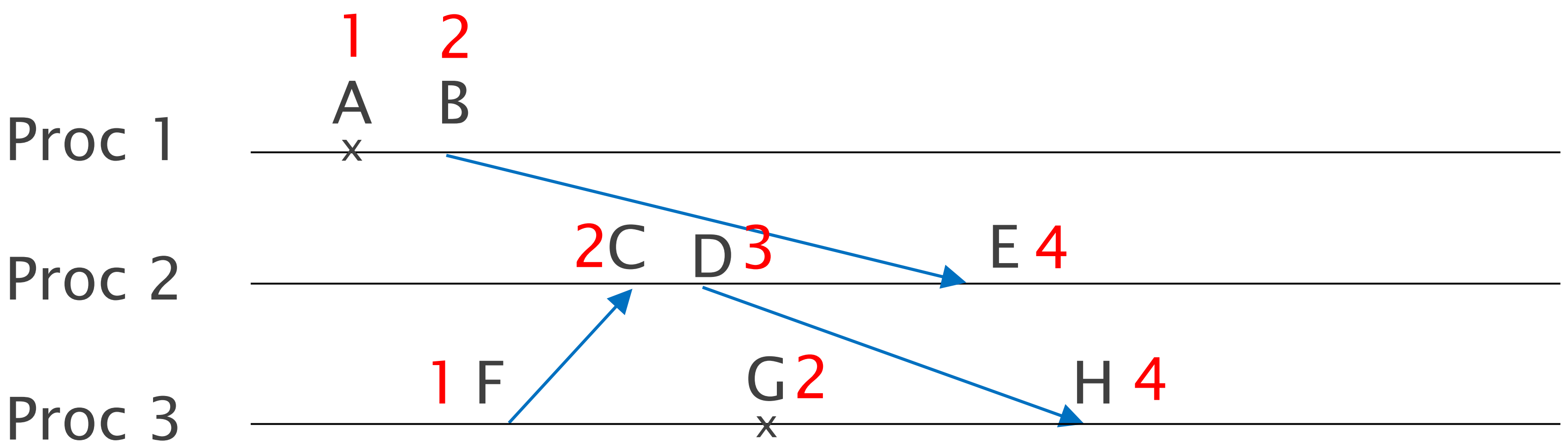
  counter = max(counter, msg.counter) + 1

Lamport Clock

- Receive events:

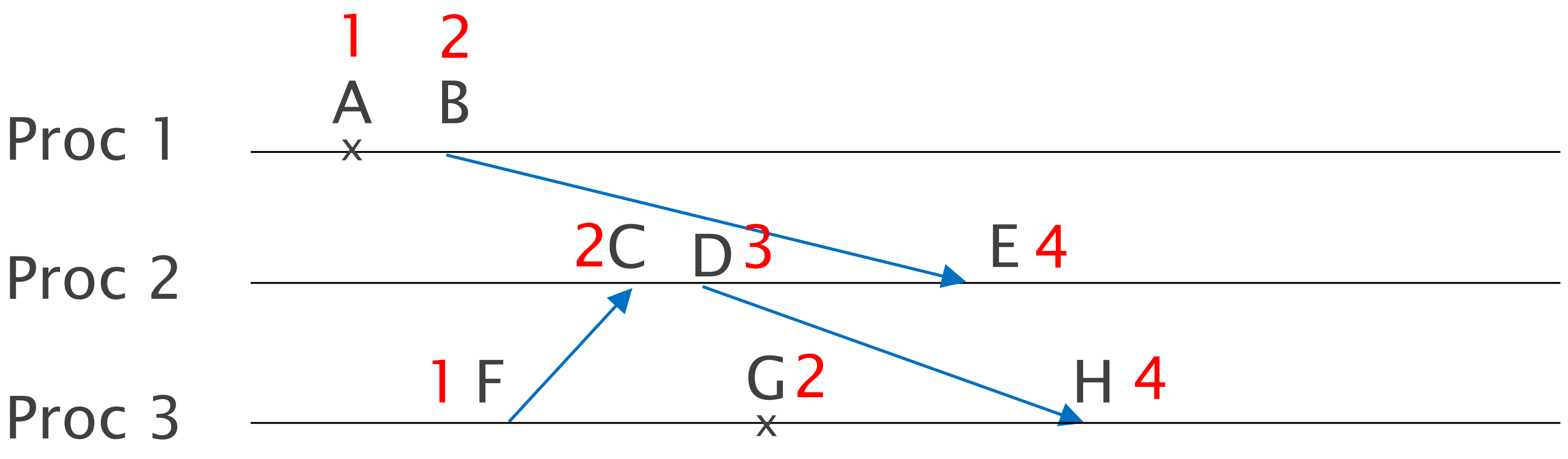
  counter = max(counter, msg.counter) + 1

- Other events: counter += 1



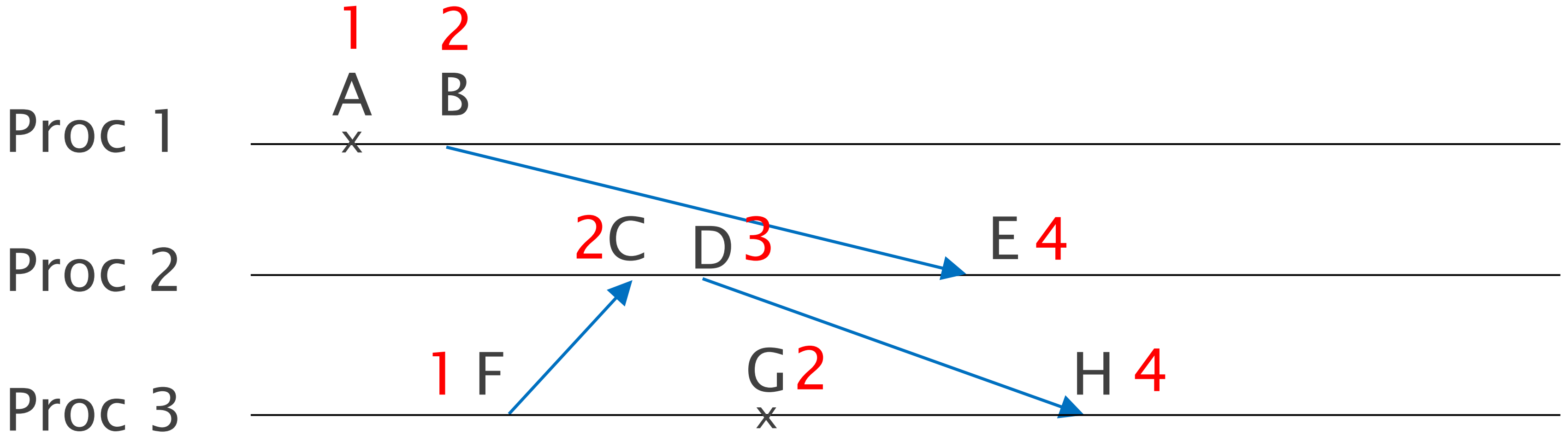
Lamport Clock Property

- If X → Y, then LC(X) < LC(Y)
- Proof:
  - Rule 1: due to counter increment
  - Rule 2: due to max
  - Rule 3: there exist a chain of Rule 1 and Rule 2



Lamport Clock Property

- If LC(X) < LC(Y), then X → Y?
- No!



Lamport Clock Property

- If X $\rightarrow$ Y, then LC(X) < LC(Y). Converse not true.
- Implication?
  - Lamport clocks capture all happens-before relations
    - Replaying events in the order of Lamport clocks yield an equivalent execution
  - But may have "false positives (causality)"
    - May place unnecessary constraints on concurrency

Outline

- Equivalent schedules in asynchrony
- Causality: happens-before relation
- Logical clocks
  - Lamport clock
  - Vector clock

Vector Clock

- Want: If X → Y if and only if VC(X) < VC(Y).

- Each clock value is a vector of integers
  - Initially, all 0
- When process i executes an event, VC[i] += 1
- Each msg carries VC of send
- When process i receives a msg

  VC[j] = max(VC[j], msg.VC[j]) for all j

  and then VC[i] += 1

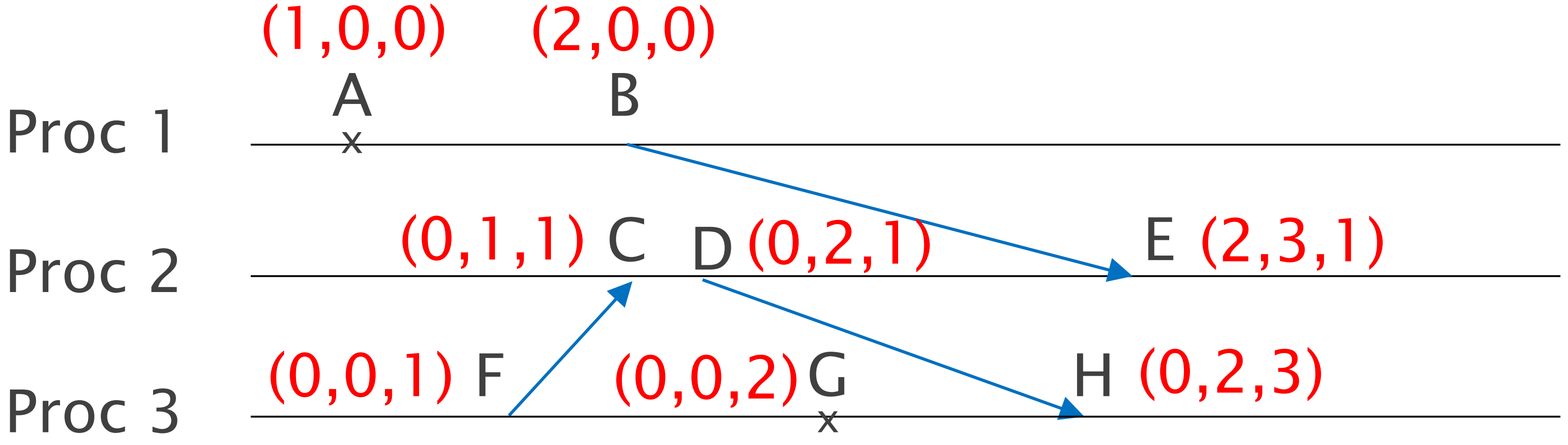
Vector Clock

- Receive events:

  VC[j] = max(VC[j], msg.VC[j]) for all j

  VC[i] += 1

- All events: VC[i] += 1



Vector Clock Comparison

- If X → Y if and only if VC(X) < VC(Y).

- Defn: For two vectors V and W of equal length
  - $V \leq W$ if $V[j] \leq W[j]$ for all j
  - $V < W$ if $V \leq W$ and $V \neq W$

Vector Clock Property

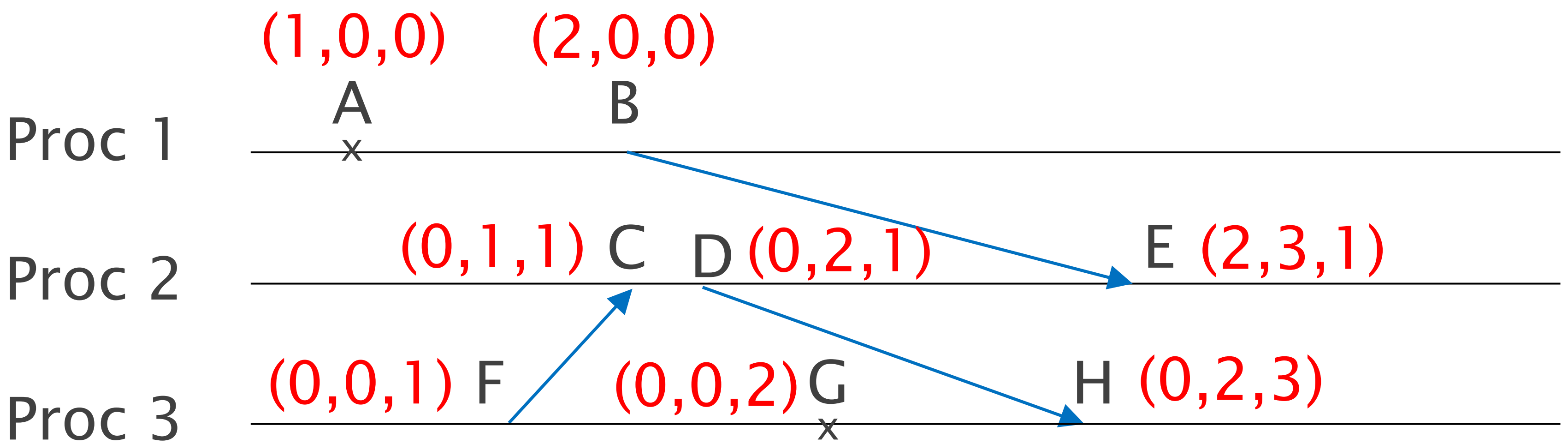
- If X → Y if and only if VC(X) < VC(Y).
- Proof: left to right direction
  - Similar to before
  - Rule 1 due to increment
  - Rule 2 due to max then increment
  - Rule 3 due to transitive closure (chain of 1 and 2)

Vector Clock Property

- If X → Y if and only if VC(X) < VC(Y).
- Proof: right to left direction
  - First, Y → X impossible
  - So just need to rule out Y || X
  - If X and Y occur at same process, not Y || X
  - Can focus on X at proc i and Y at proc j ≠ i

Vector Clock Property

- Lemma: a process knows the latest value of its own vector component, while others may not
  - Proc i is the only one incrementing i^{th} component



Vector Clock Property

- If X → Y if and only if VC(X) < VC(Y).
- Proof: right to left direction
  - Just need to focus on X at proc i and Y at proc j
  - How come VC[X][i] <= VC[Y][i]?
    - Proc j learnt the i^{th} component of X when Y occurs
  - Must exist a chain of msgs X → … → Y that propagates i^{th} component of X proc j learn
  - X and Y not concurrent, QED

Summary

- Happens-before relations
  - capture causality
  - fully categorize asynchronous executions
- Lamport clocks capture happens-before but may also impose false causality
- Vector clocks exactly capture happens-before