Large Scale Systems
CS 410 / 510

Lecture 2:

Time and Clock

Suyash Gupta

Assistant Professor

Distopia Labs and ORNG

Dept. of Computer Science

(E) suyash@uoregon.edu

(W) gupta-suyash.github.io

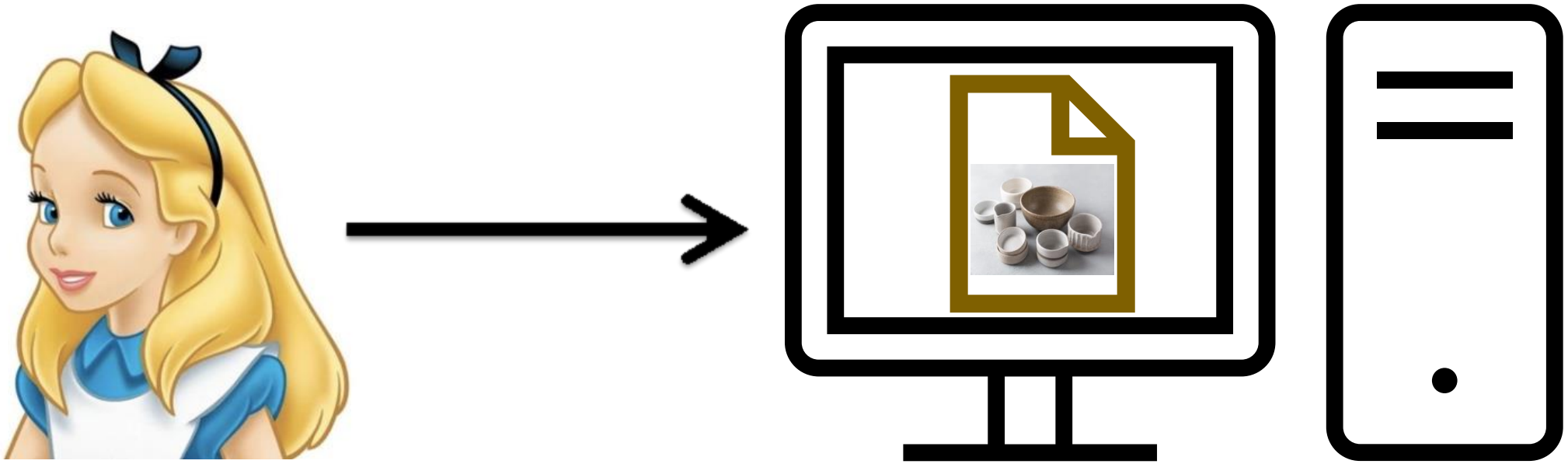
UNIVERSITY OF OREGON

Assignment 1 is Out!

- Assignment 1 is out!
- Form your groups asap and if you are unable to form, send me a message by **5pm PST today**!
- We will discuss Assignment 1 towards the end of the class!

Clocks and Time

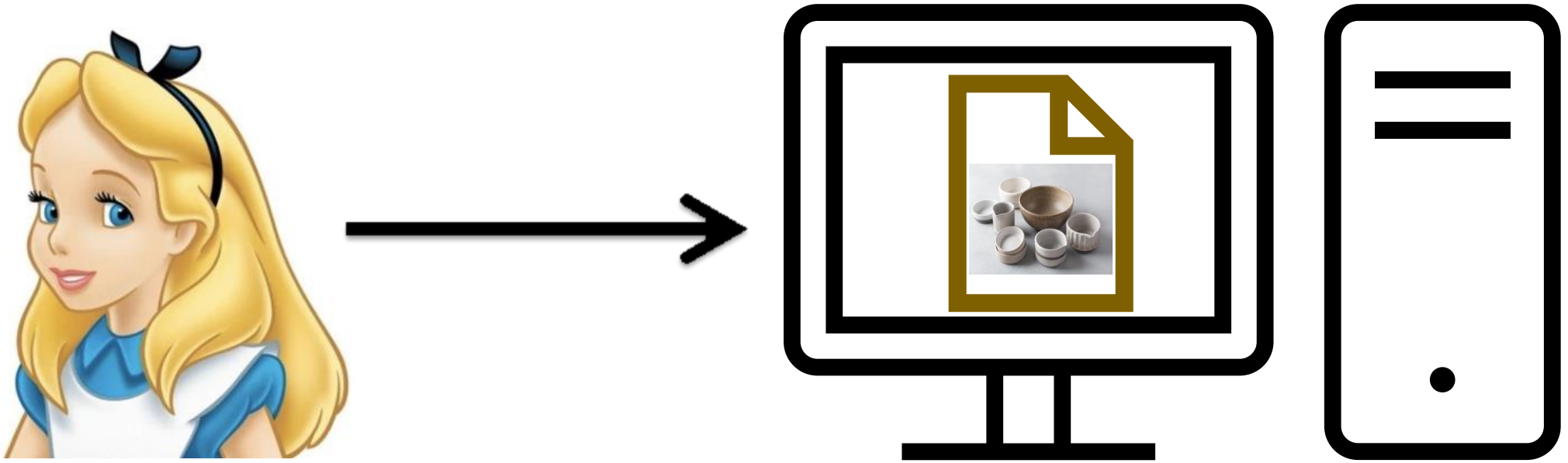
What is the Current time?



Say Alice asks her computer what is the time now.

How does computer know the correct time?

What is the Current time?

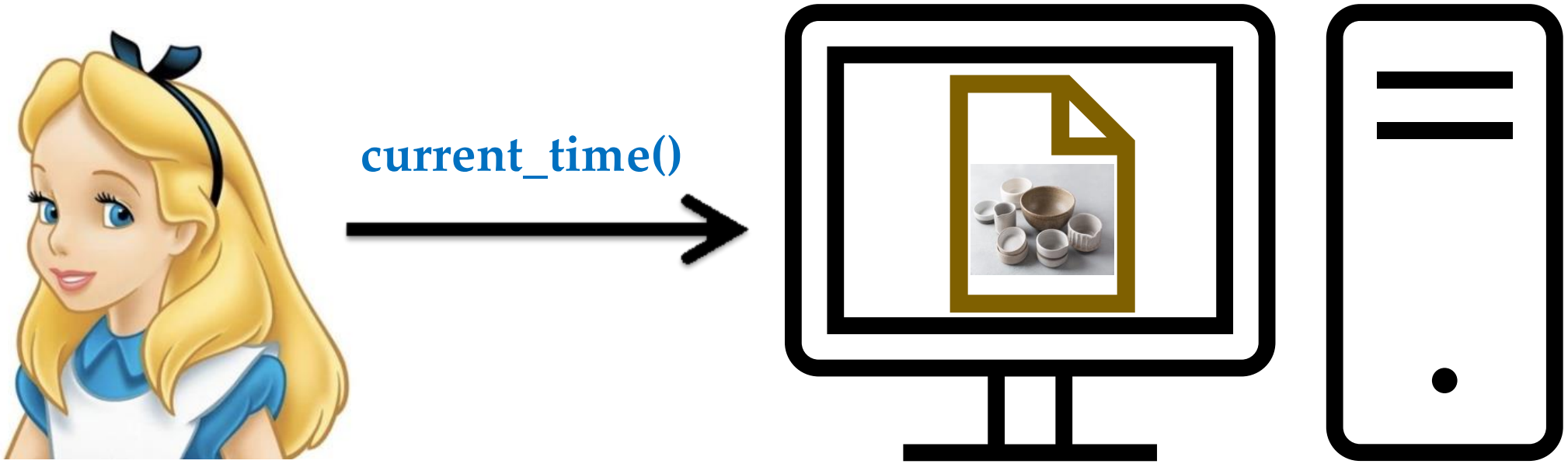


Computer does not understand the notion of time.

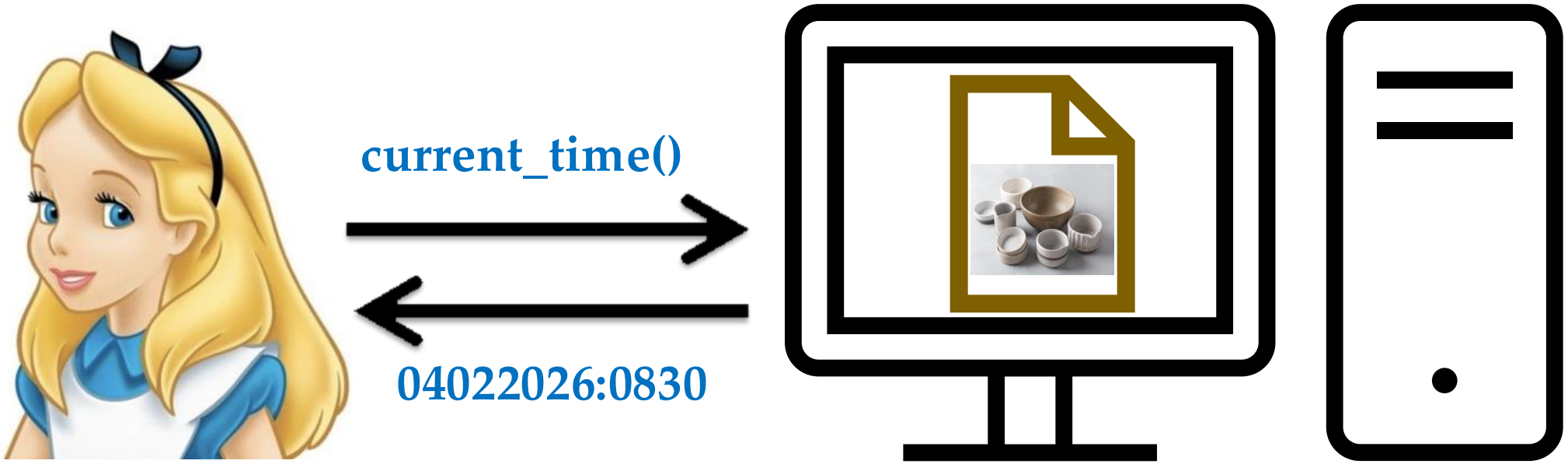
It has access to a physical clock, specifically, a quartz crystal that helps track the time.

At the start, the user sets the date and time for the computer.

What is the Current time?



What is the Current time?



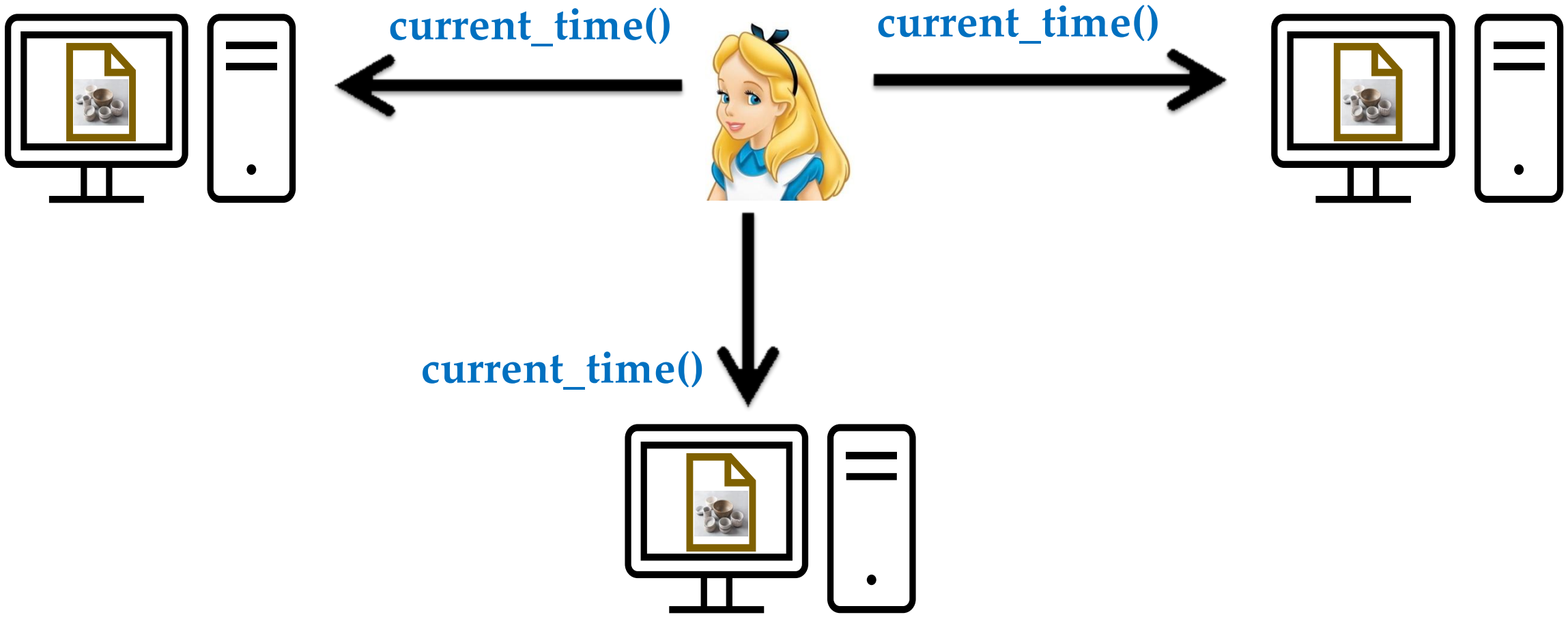
How to Measure Time in a Distributed Setup?

Asking time from a single computer is easy?

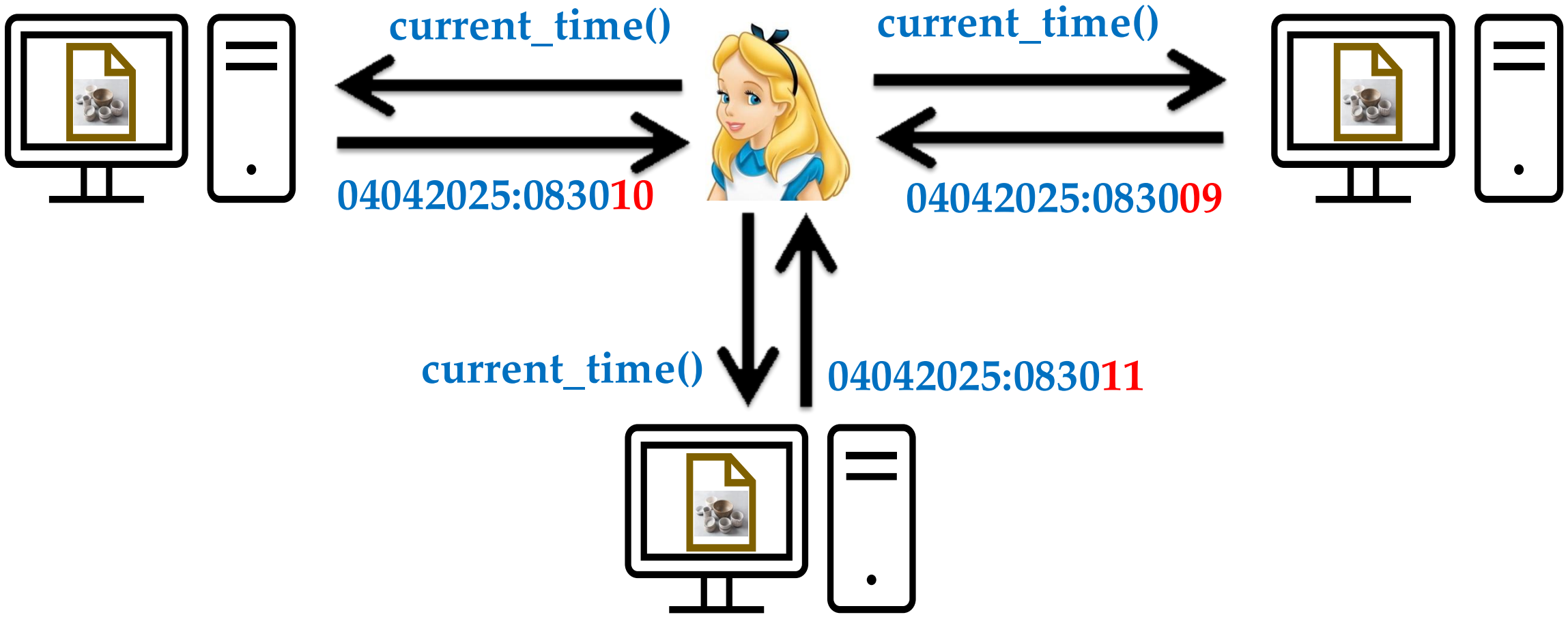
What happens in a replicated system?

Will all the replicas output same time?

How to Measure Time in a Distributed Setup?



How to Measure Time in a Distributed Setup?



As each replica answers using its own physical clock, outputs may differ.

Clock Synchronization

- To match the clocks of all the distributed entities (for example replicas)
  - Employ a clock synchronization protocol
- Many **flavors** of clock synchronization protocol.
- For example: **Network Time protocol**.
  - Assume there is a centralized time server.
  - Time Server is the gold standard for all the time.
  - Every entity (like replica) asks the time server its current time.
  - Once the server sends a response, the receiver subtracts the latency from the response and uses the result to set its clock.

Why do we need clocks?

Why do we need clocks?

- Let's you know the current time.
- Helps to order client requests/transactions/events.
- Remember, a system may receive transactions from multiple clients, so it needs to decide which to process first.
- Similarly, if two events take place, we may need to decide which event took place first.

Events

- An event is an action that changes state of the system/process/thread.

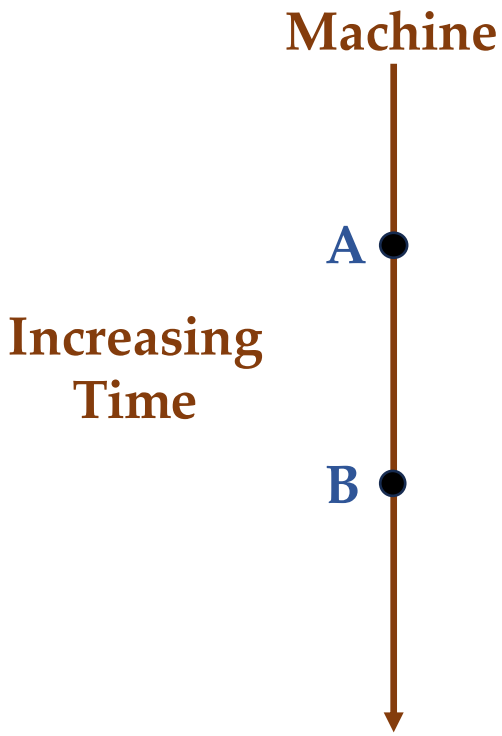
- For example:
  - Sending a message.
  - Receiving a message.
  - Reading/Writing a variable.

Happens Before Relationship

- Happens Before Relationship defines the order between two events.
  - Which event happened before which event?
  - Happens Before notation: →

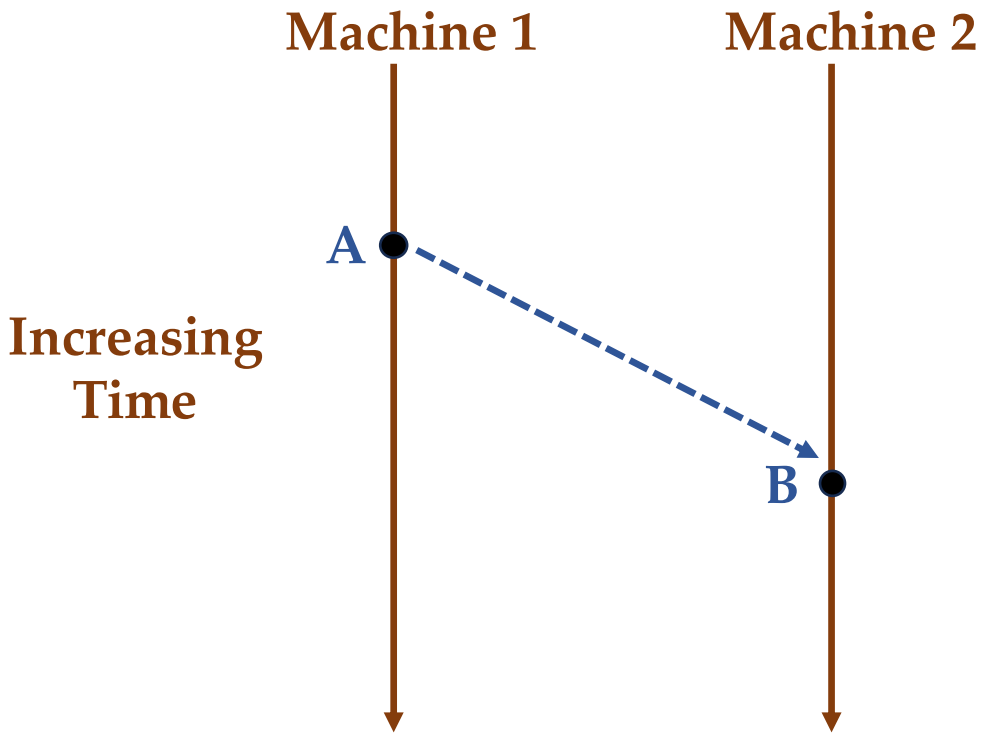
- Given two events **A** and **B**, when can you say that **A happened before B**?

Happens Before Relationship



Given two events A and B occurring at a machine
if A occurs before B, we say that A → B

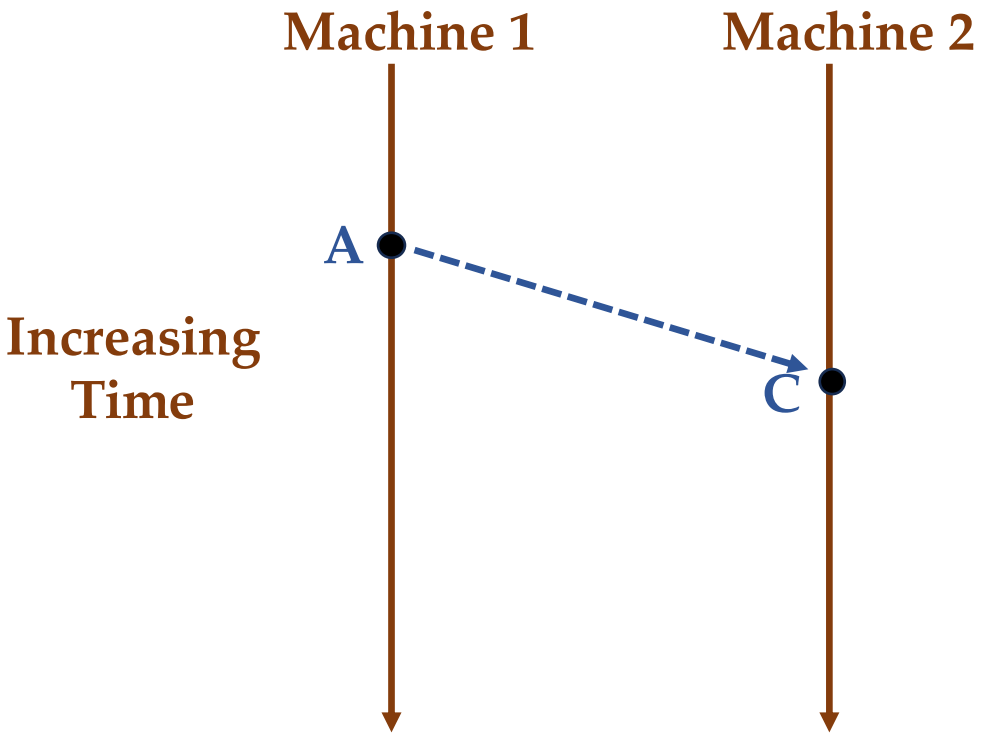
Happens Before Relationship



Given two events A and B, such that A occurs at Machine 1 and causes an event B to occur on Machine 2, we say that A → B.

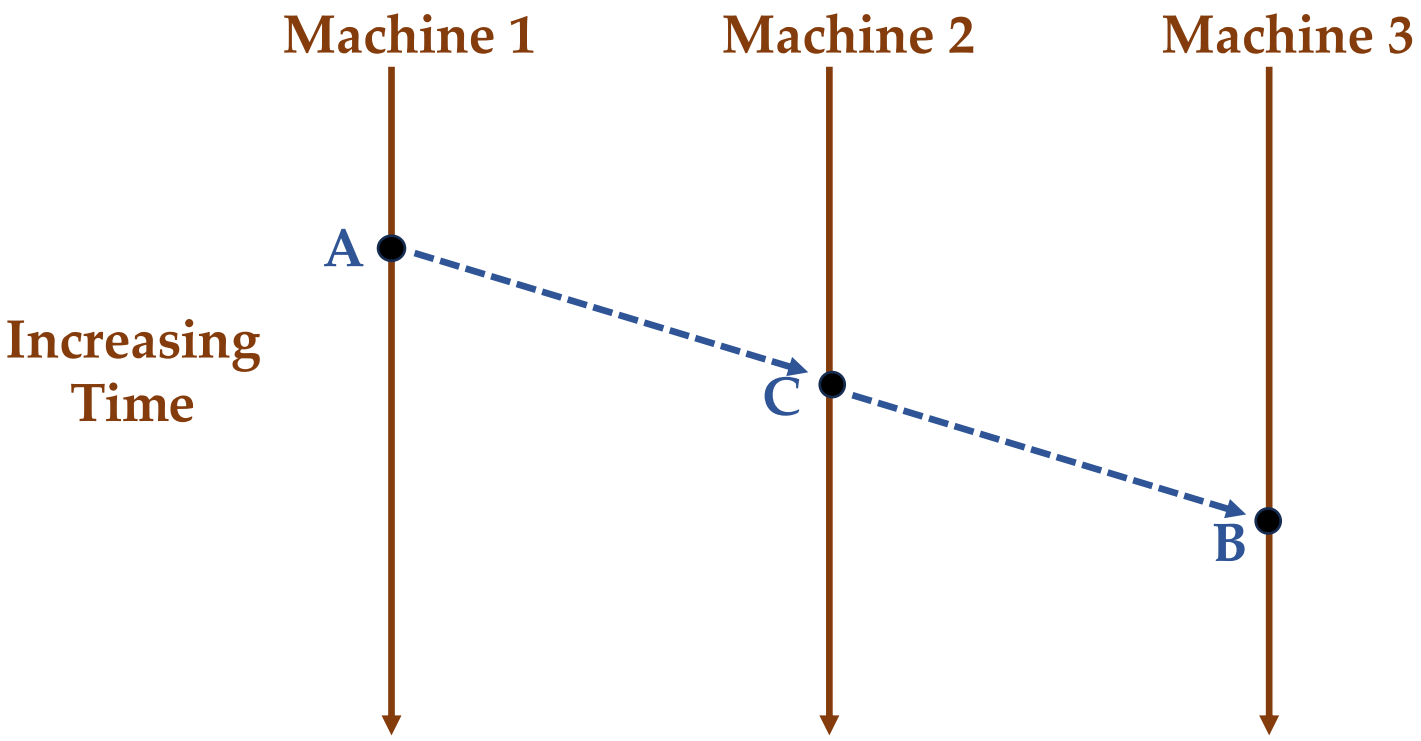
Example: A is "sending a message" and B is "receiving a message"

Happens Before Relationship



A → C

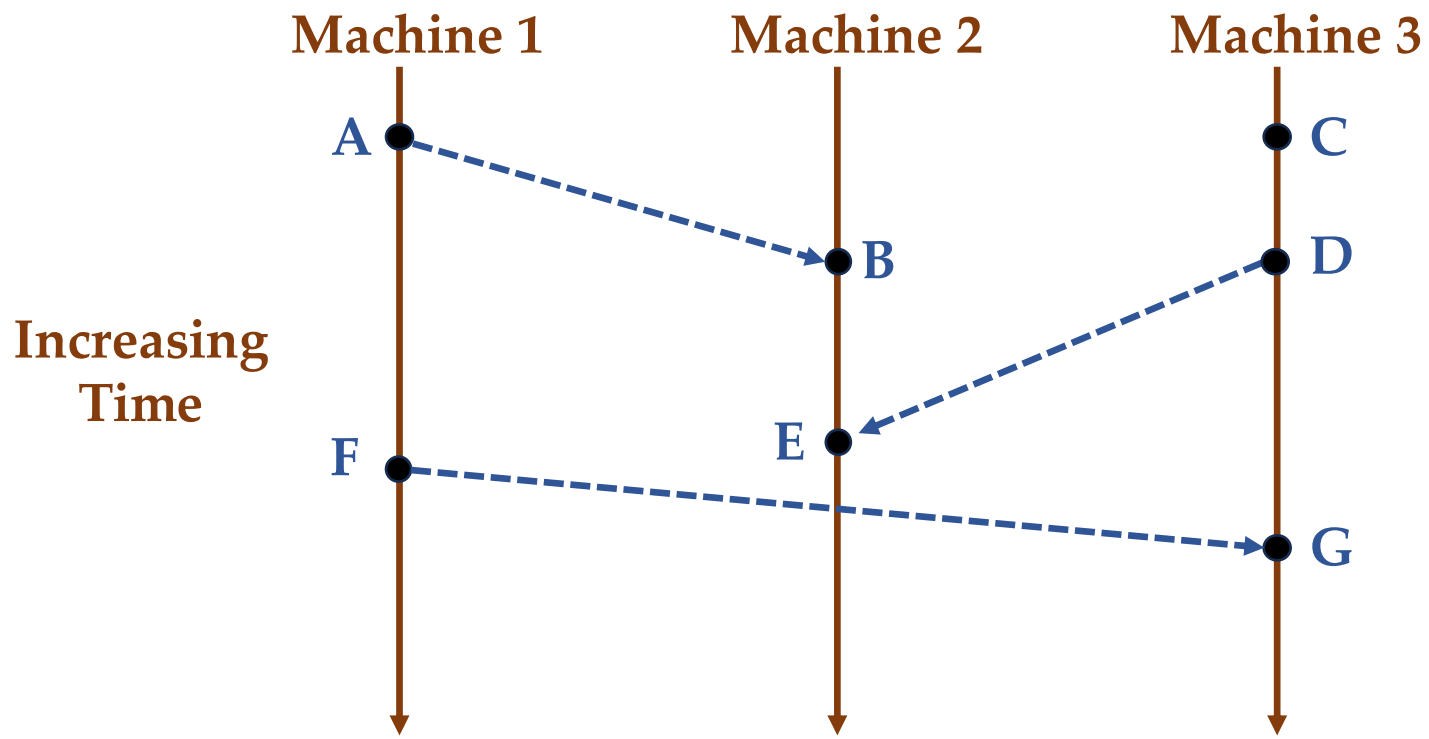
Happens Before Relationship



As A → C and C → B, so A → B.

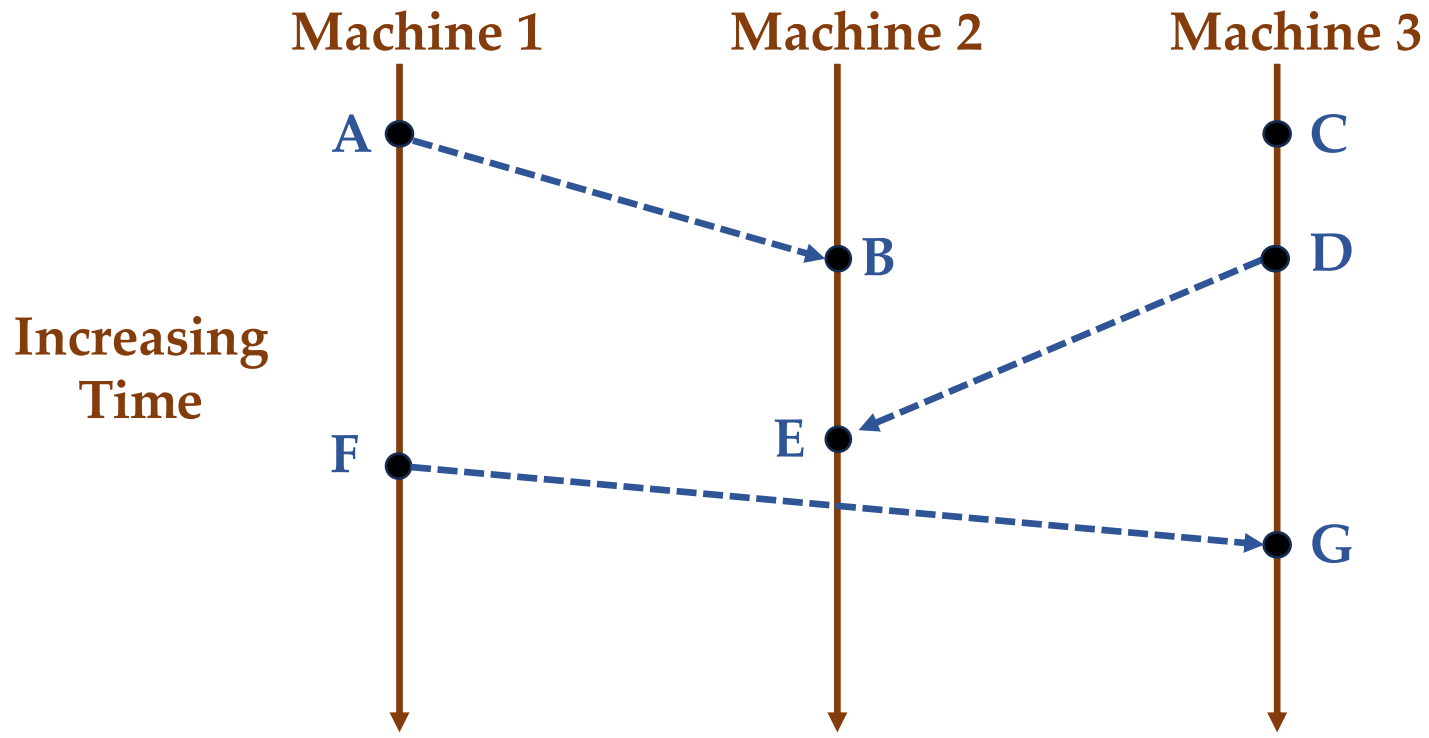
Transitivity holds.

Happens Before Relationship



- Another example!
- We will consider a set of events taking place at each machine.
- Again, we consider both “send” and “receive” operations as separate events.

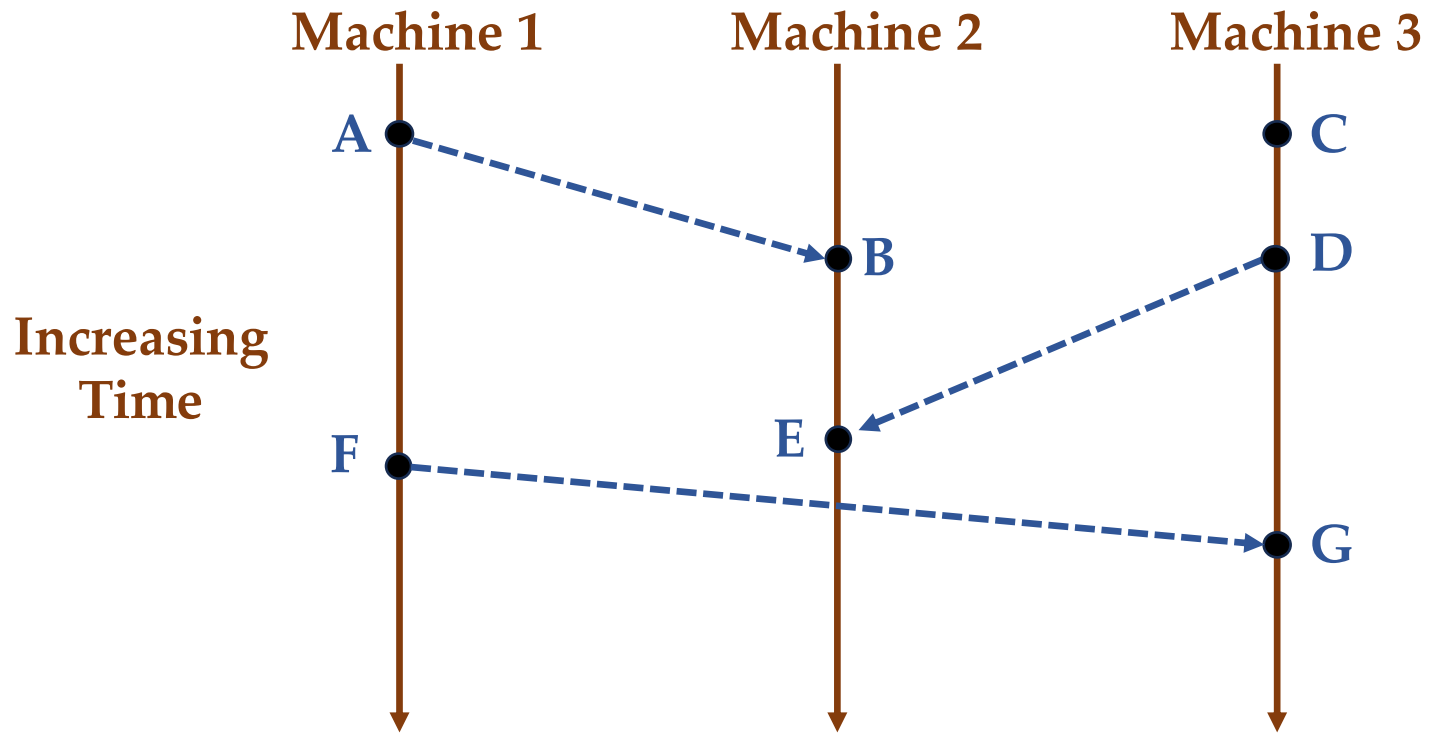
Happens Before Relationship



- Happen Before relationship easily visible:
  - A → B, A → F, B → E, C → D, D → G, D → E, F → G

- Does A → D and D → F ?

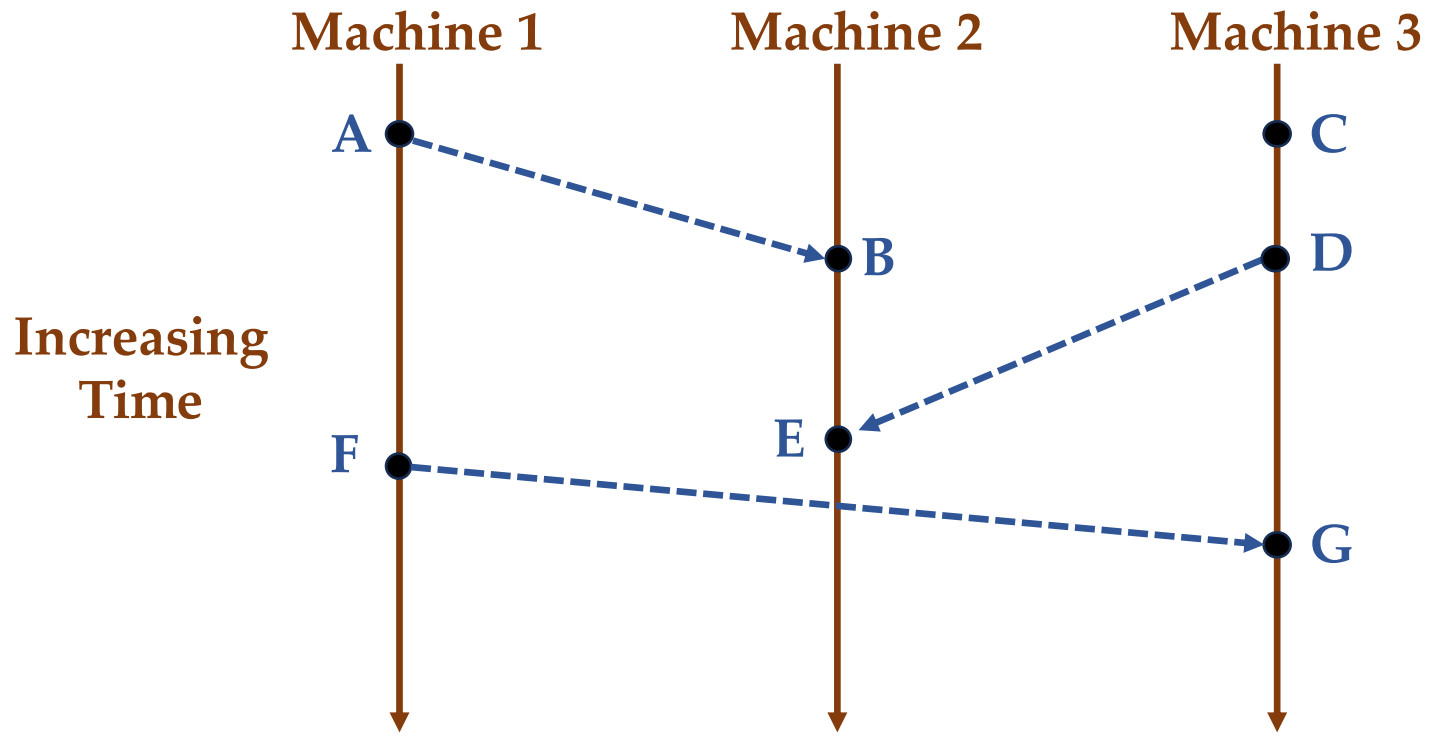
Happens Before Relationship



- Happen Before relationship easily visible:
  - A → B, A → F, B → E, C → D, D → G, D → E, F → G

- Does A → D and D → F ? We cannot say!

Happens Before Relationship



- The trick is to see if there is a reachability path.
  - If not, then these are **concurrent events**.

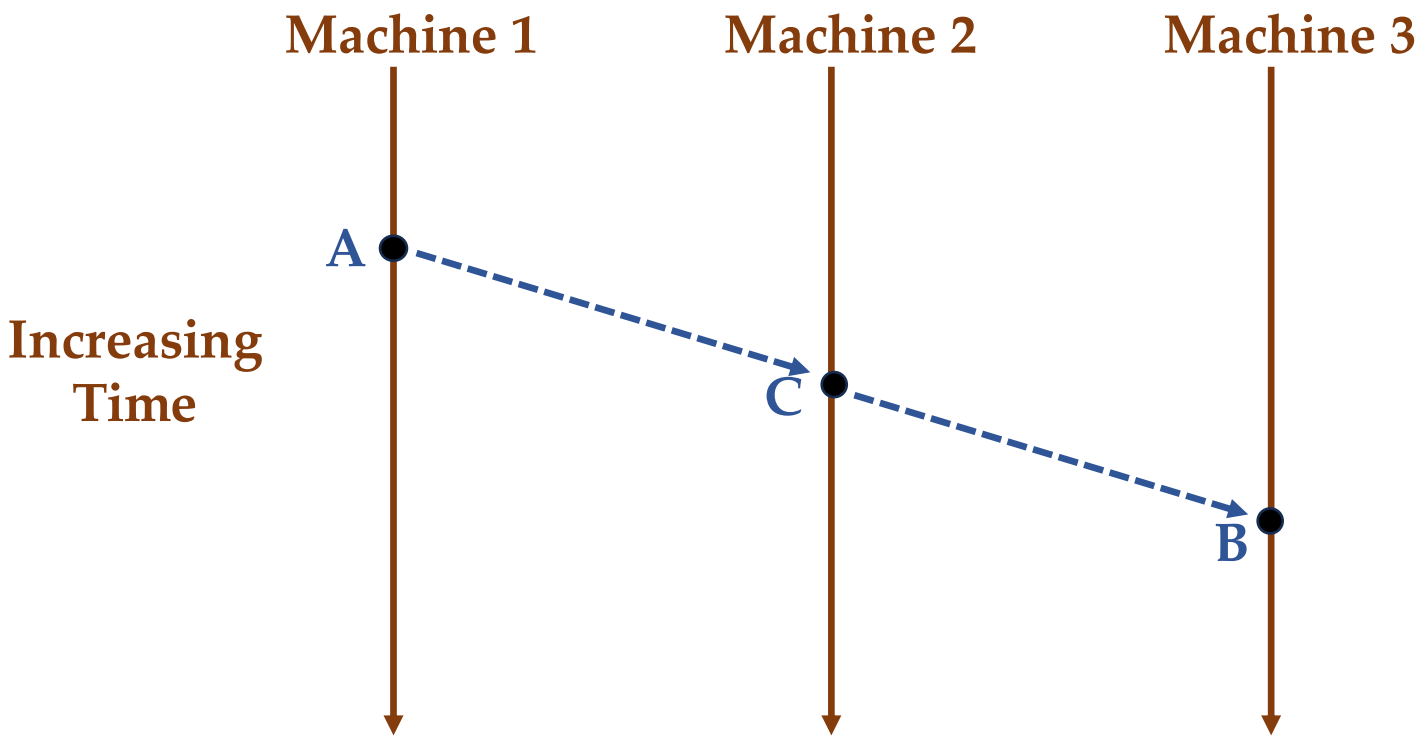
- No path from A to D. Similarly, no path from D to F.

How to use Happens Before

How to use Happens Before

- Using Happens Before Relationship, each system tries to create a **total order** of all the events, and if not possible, then a **partial order**.

Total Order of All Events



As A → C and C → B, so A → B.

So, Total order = A → C → B.

Note: Every element is comparable to each other

Total Order

- Another example of total order?

- Set of all the Natural Numbers under less than (<) operation.

Here,

- $1 < 2$
- $2 < 3$

And so on.

Partial Order

- A set S of elements together with a binary operation is termed as a partially ordered set if it satisfies the following properties.
  - Let the binary operation notation be: $\leqslant$

Partial Order

- A set S of elements together with a binary operation is termed as a partially ordered set if it satisfies the following properties.
  - Let the binary operation notation be: $\preccurlyeq$
- Reflexivity: For all $\mathbf{a \in S, a \preccurlyeq a}$

Partial Order

- A set S of elements together with a binary operation is termed as a partially ordered set if it satisfies the following properties.
  - Let the binary operation notation be: $\leqslant$
- Reflexivity: For all $\mathbf{a \in S, a \leqslant a}$
- Anti-symmetry: For all $\mathbf{a, b \in S}$, **if** $\mathbf{a \leqslant b}$ **and** $\mathbf{b \leqslant a}$, **then** $\mathbf{a = b}$

Partial Order

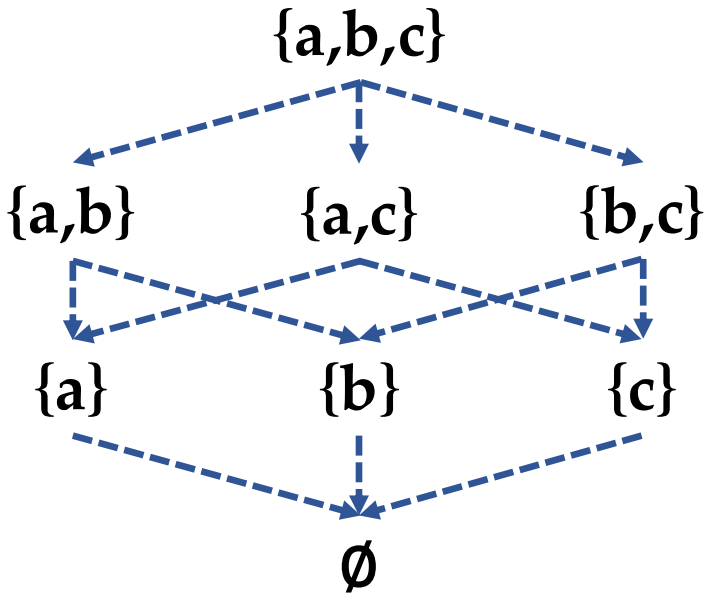
- A set S of elements together with a binary operation is termed as a partially ordered set if it satisfies the following properties.
  - Let the binary operation notation be: $\leqslant$
- Reflexivity: For all $\mathbf{a \in S, a \leqslant a}$
- Anti-symmetry: For all $\mathbf{a, b \in S,}$ **if** $\mathbf{a \leqslant b}$ **and** $\mathbf{b \leqslant a,}$ **then** $\mathbf{a = b}$
- Transitivity: For all $\mathbf{a, b, c \in S,}$ **if** $\mathbf{a \leqslant b}$ **and** $\mathbf{b \leqslant c,}$ **then** $\mathbf{a \leqslant c}$

Partial Order

- Consider the Set of three elements: {a, b, c}.
- Next, consider the set of all subsets of the above set.
  - S = { ∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c} }
- Is this a partially ordered set (aka poset) on operation subset (⊆) ?

Partial Order

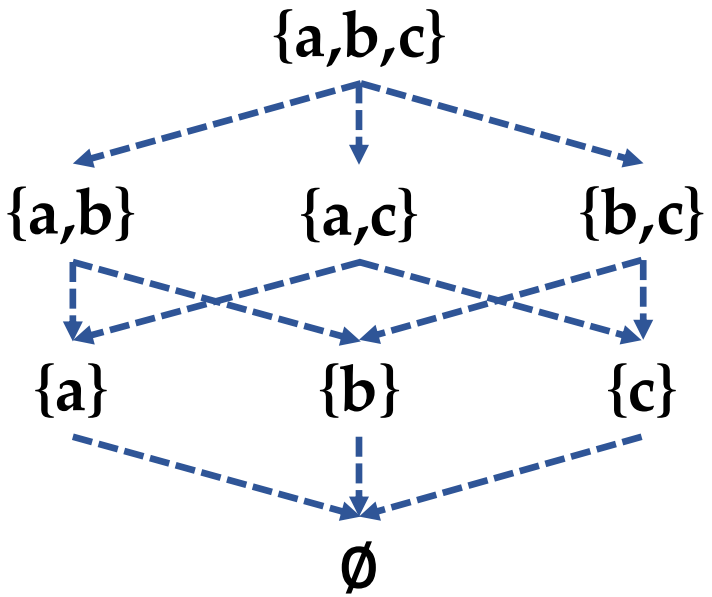
S = { ∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c} }



Poset ordering on ⊆.

Partial Order

S = { ∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c} }



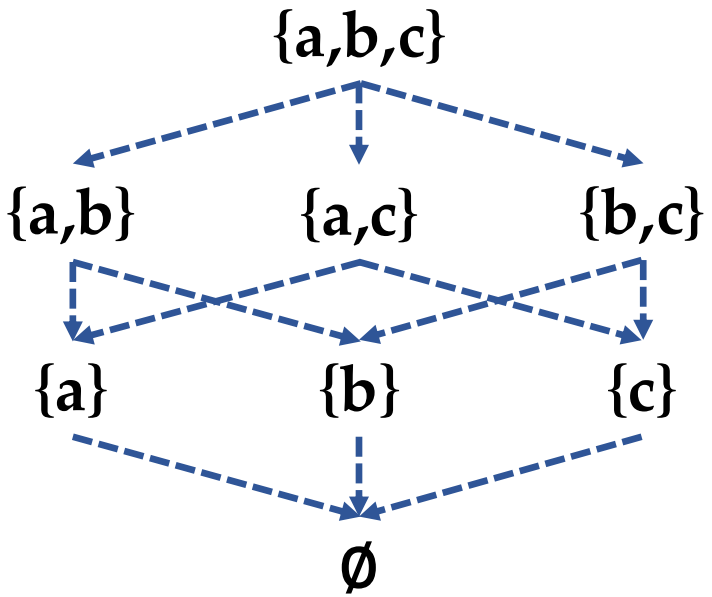
Poset ordering on ⊆.

Let's pick two elements from S: {a} and {a,b}.

- Reflexivity: Is {a} ⊆ {a} ?

Partial Order

S = { ∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c} }



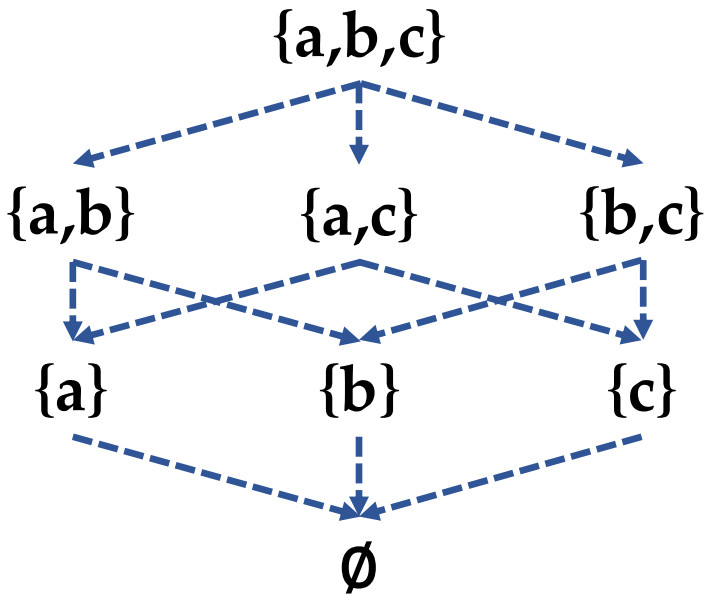
Poset ordering on ⊆.

Let's pick two elements from S: {a} and {a,b}.

- Reflexivity: Is {a} ⊆ {a} ? Yes, so satisfy reflexivity.

Partial Order

S = { ∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c} }



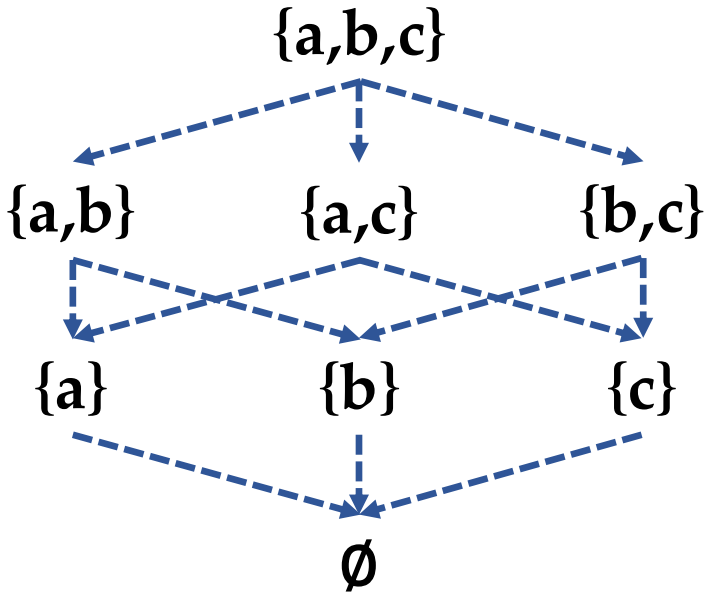
Poset ordering on ⊆.

Let's pick two elements from S: {a} and {a,b}.

- Reflexivity: Is {a} ⊆ {a} ? Yes, so satisfy reflexivity.
- Anti-symmetry: Satisfies implicitly.

Partial Order

S = { ∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c} }



Poset ordering on ⊆.

Let's pick two elements from S: {a} and {a,b}.

- Reflexivity: Is {a} ⊆ {a} ? Yes, so satisfy reflexivity.
- Anti-symmetry: Satisfies implicitly.
- Transitivity: {a} ⊆ {a,b} and {a,b} ⊆ {a,b,c}, so {a} ⊆ {a,b,c}; satisfies!

What do you need to create a Partial Order?

What do you need to create a Partial Order?

- Information about events.
- Information about Happens before relationships.
- Time at which an event occurred!

What do you need to create a Partial Order?

- Information about events.
- Information about Happens before relationships.
- Time at which an event occurred!
- But Physical clocks need a lot synchronization!
  - Is there any other option?

Logical Clocks

- Logical clocks have nothing to do with actual clocks.
  - In fact the term clock is a misnomer!
- Logical clocks help to order events.
- We will study two types of logical clocks:
  - Lamport Clocks
  - Vector Clocks (more practical)

Lamport Clocks

- Created by Leslie Lamport!
- So, what are Lamport Clocks?

Lamport Clocks

- Created by Leslie Lamport!
- So, what are Lamport Clocks?
- They are integer counters.
  - Lamport Clock of Event A denoted as LC(A) will be an integer.
- Clock Condition:
  - **If A → B, then LC(A) < LC(B).**

Lamport Clock Algorithm

- The following algorithm can help to assign each event a Lamport clock.

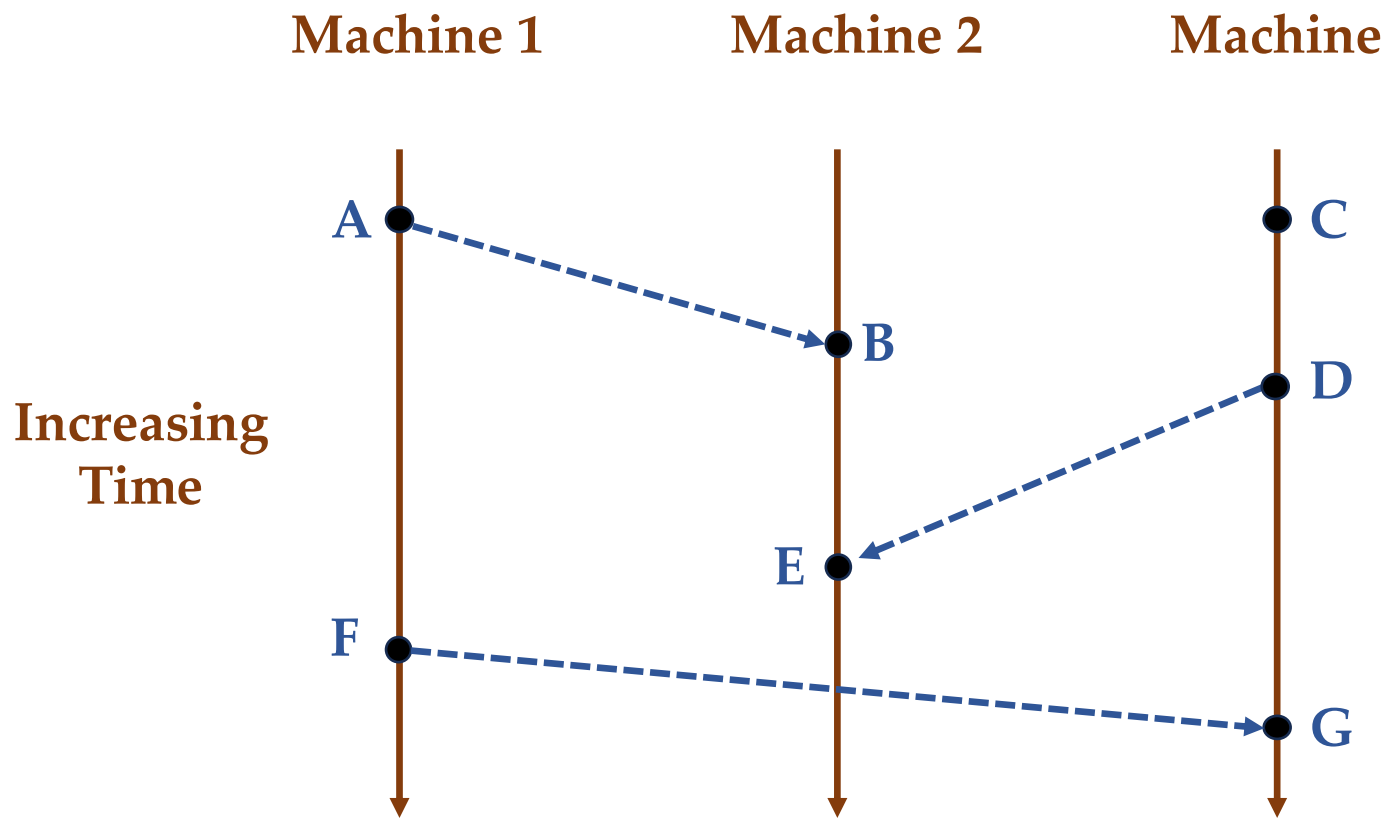
1. Each process/machine keeps a counter, initialized to 0.

2. When an event occurs on a process, it increments its counter by 1 and assigns that value to the event.

3. At the time of sending a message, the process includes the current value of the counter.

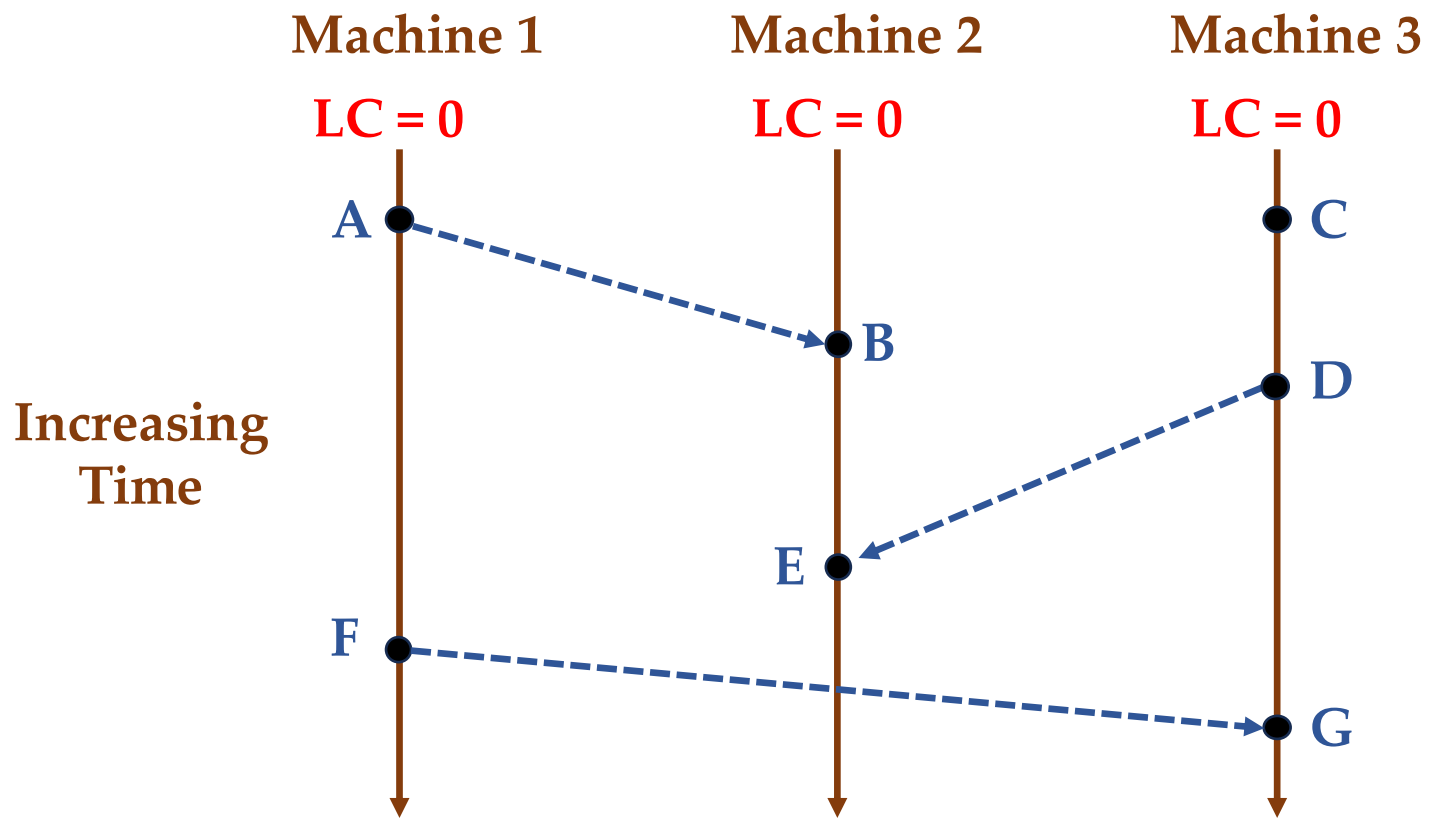
4. On receiving a message, a process sets its counter value as Maximum(local, received) + 1.

Lamport Clock Algorithm



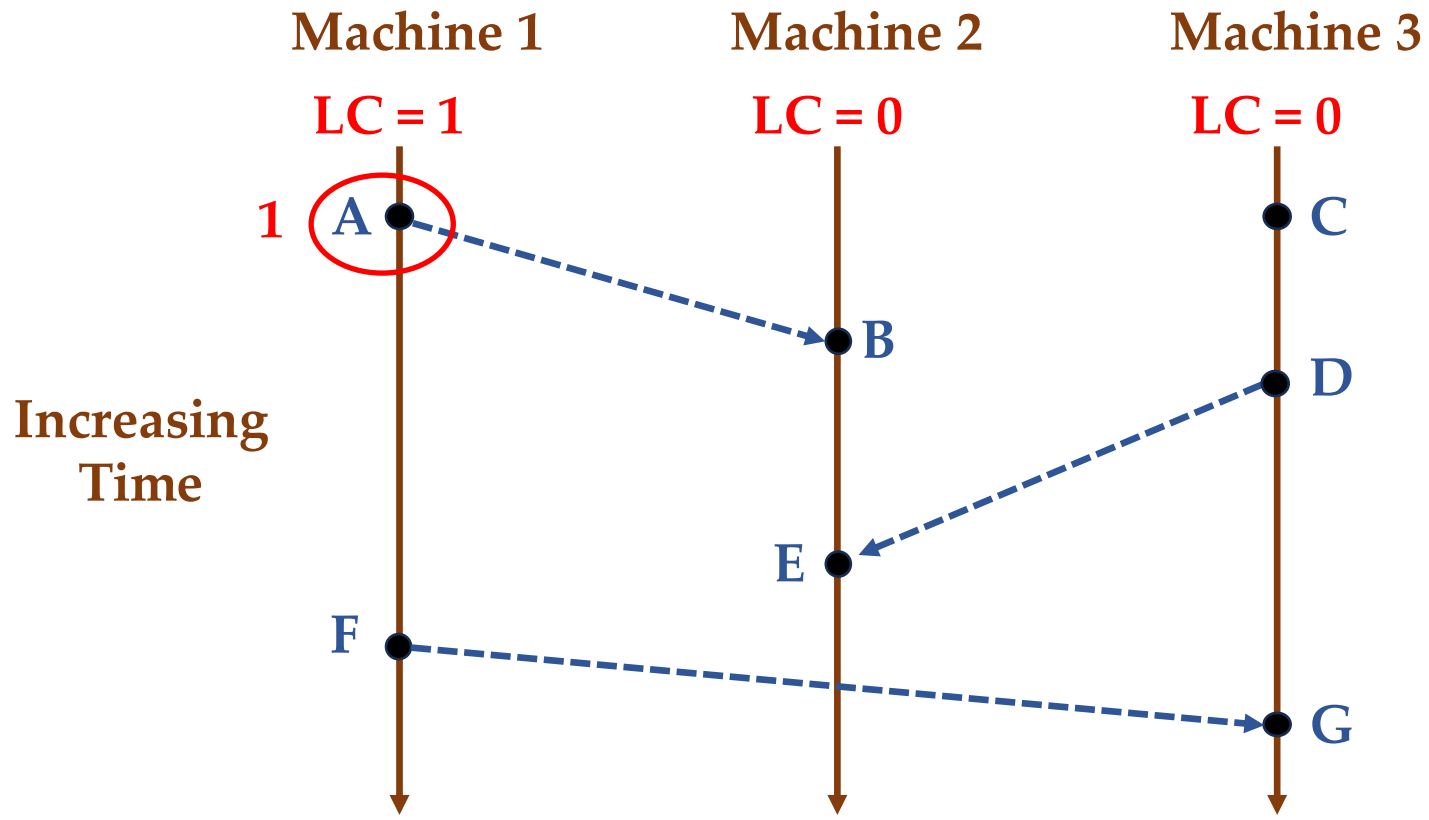
Let's apply apply Lamport Clock Algorithm on this example.

Lamport Clock Algorithm



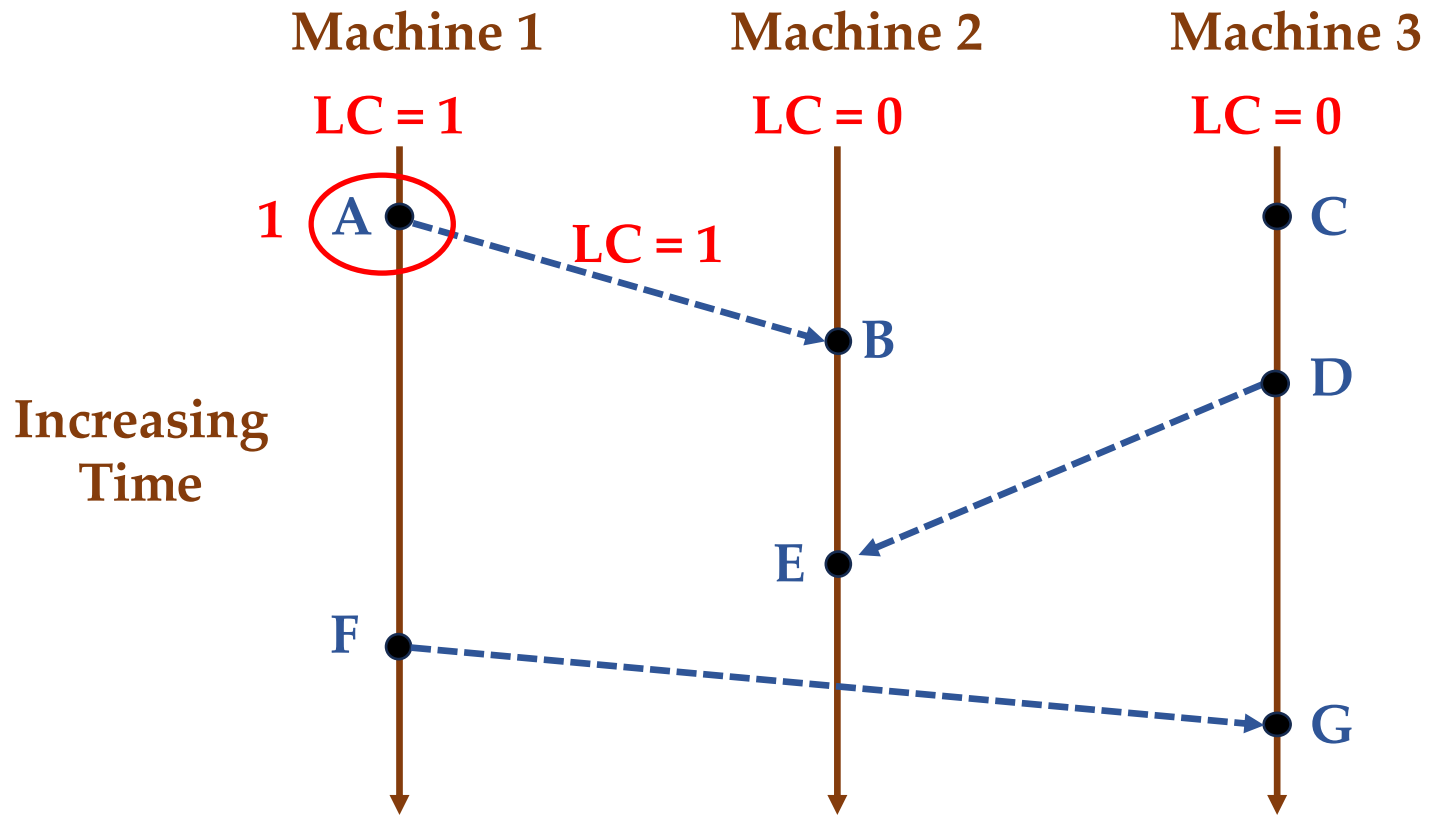
Initialize Lamport Clock at each machine.

Lamport Clock Algorithm



On Machine 1, the first event is A, so we increment its counter and assign it a clock 1.

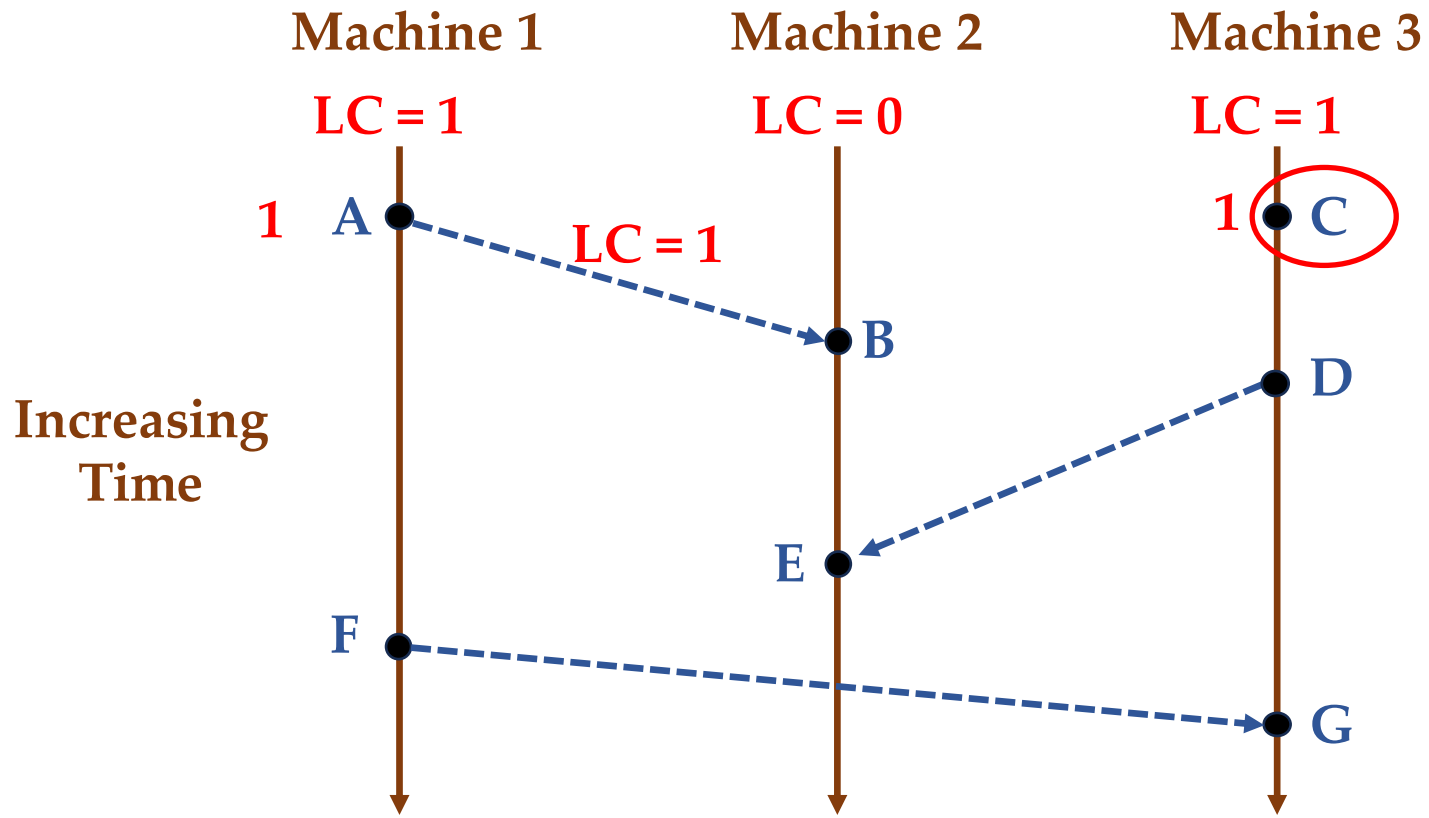
Lamport Clock Algorithm



On Machine 1, the first event is A, so we increment its counter and assign it a clock 1.

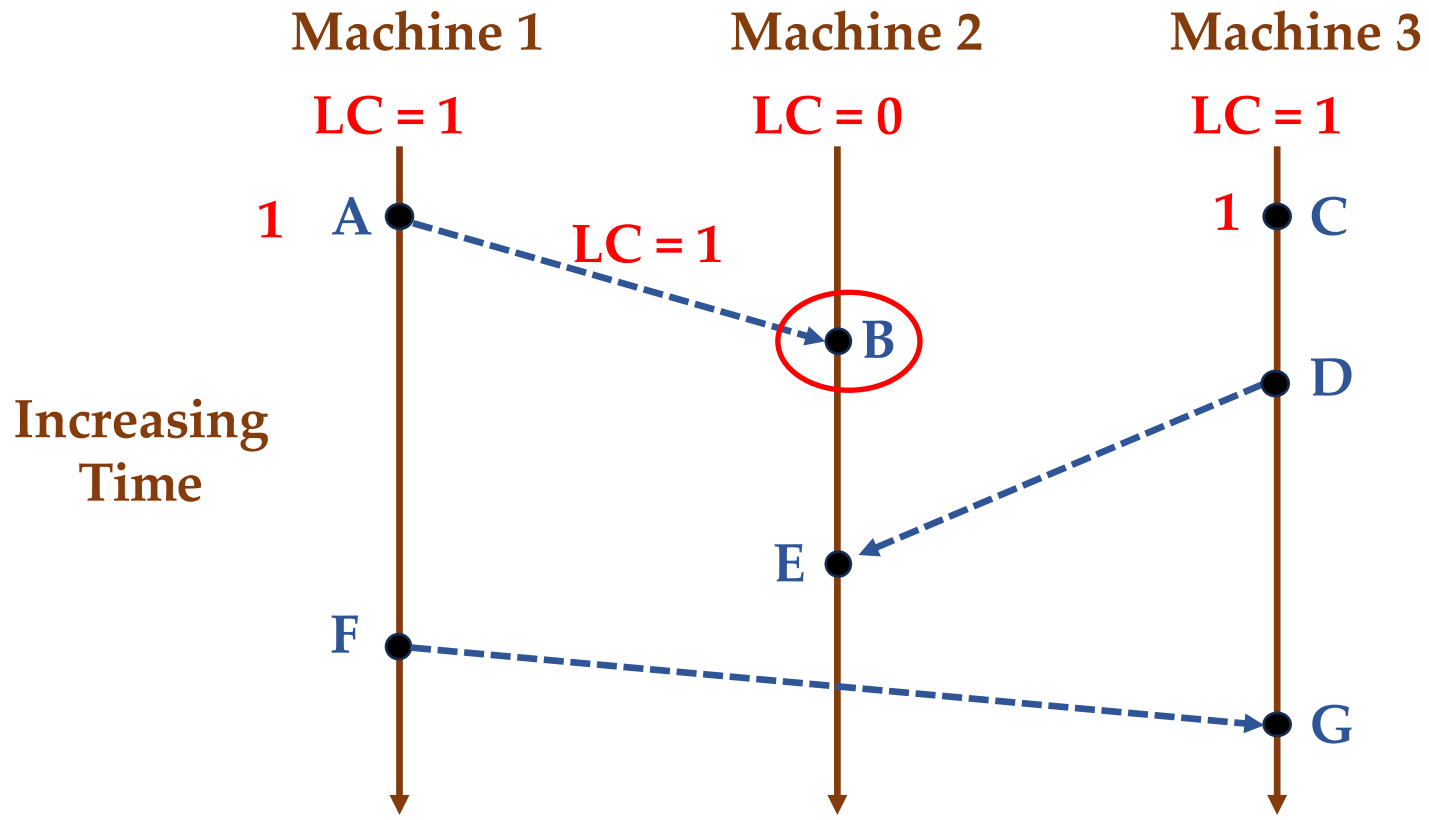
Further, this is a send event, so we send its Lamport Clock to Machine 2.

Lamport Clock Algorithm



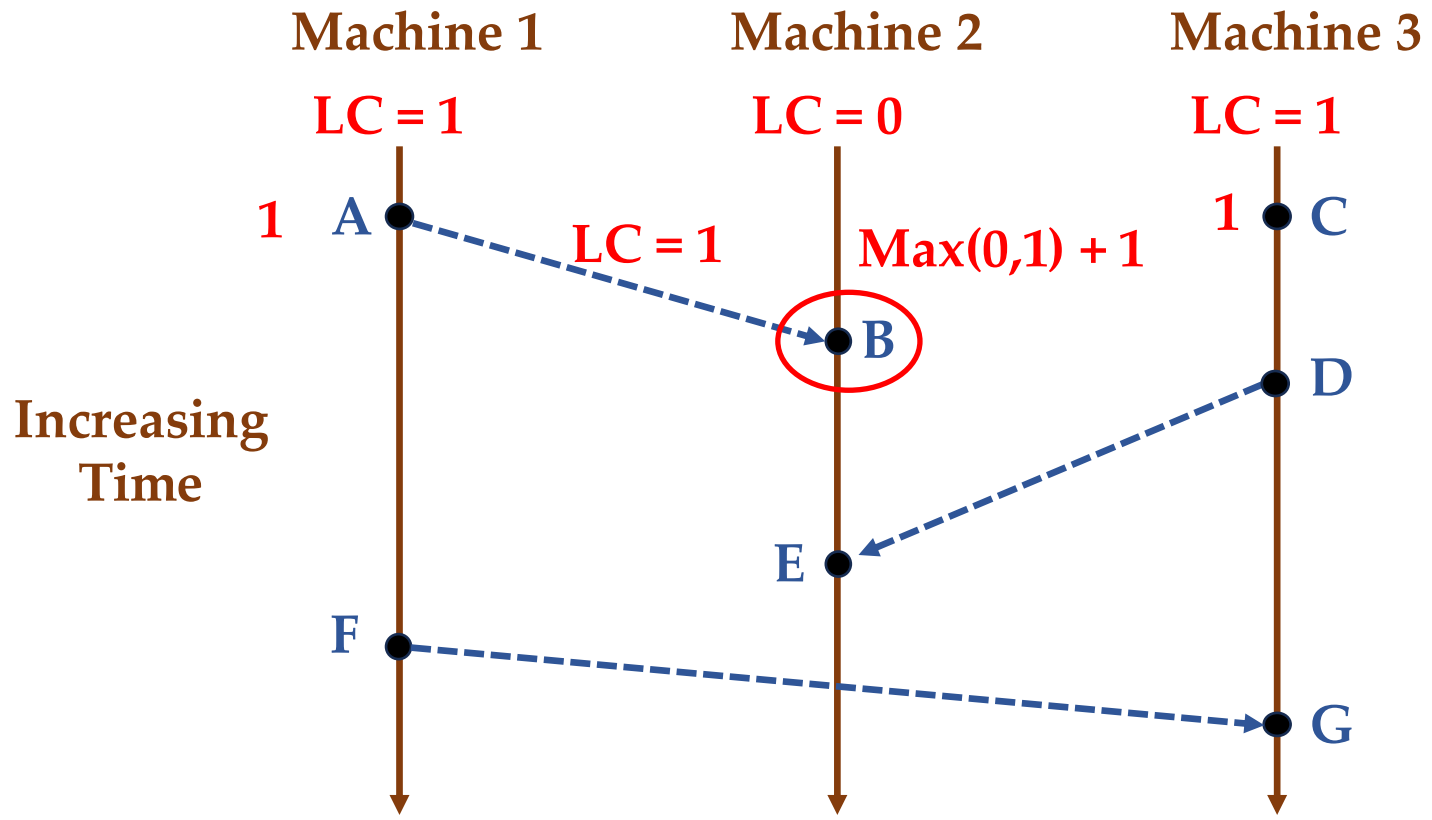
Similarly, on Machine 3, event C is its first event, so we increment its counter and assign it a clock 1.

Lamport Clock Algorithm



On Machine 2, Event B is its first event, and it is a receive event!

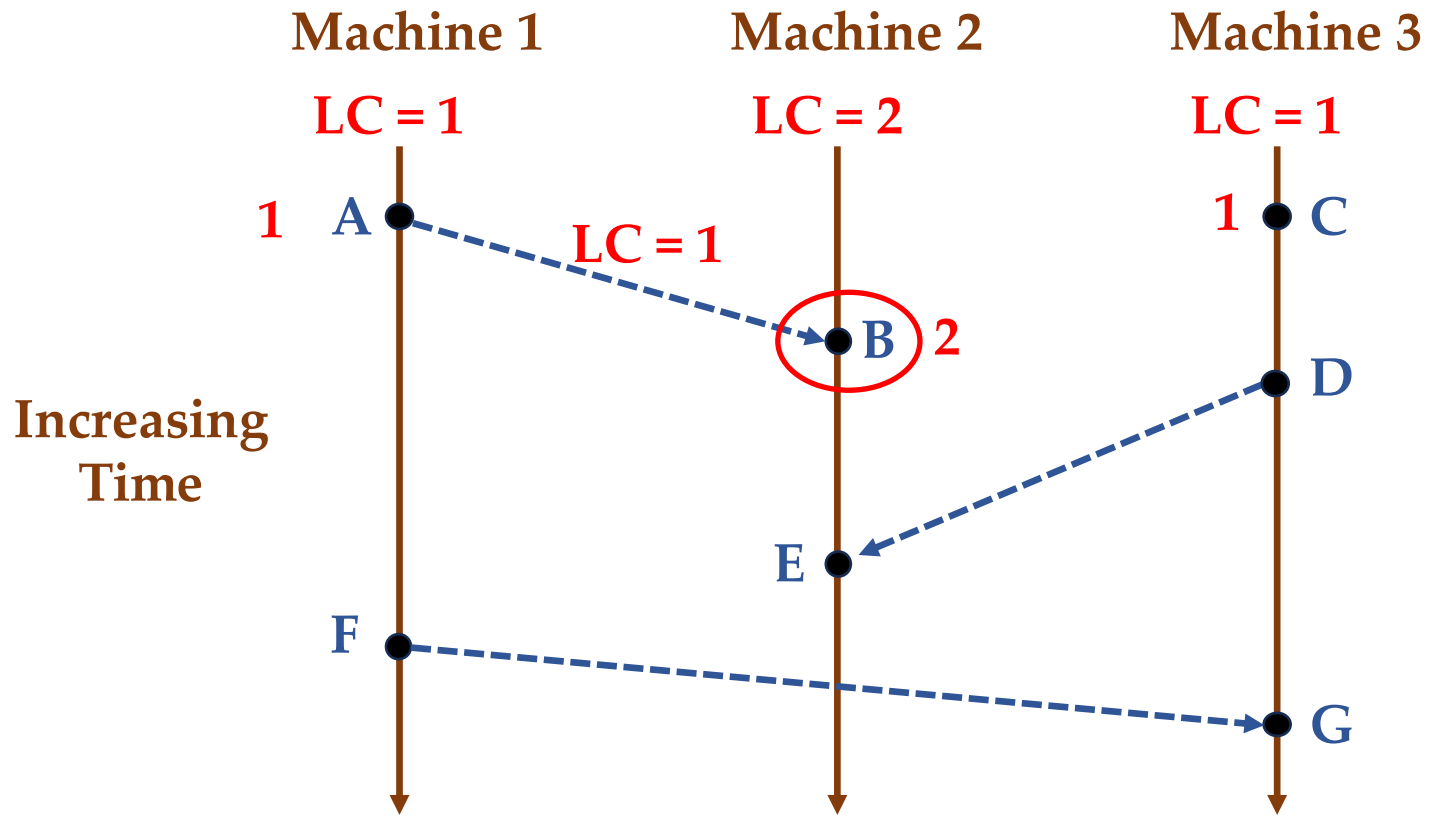
Lamport Clock Algorithm



On Machine 2, Event B is its first event, and it is a receive event!

So, we need to do Max(local, receive) + 1.

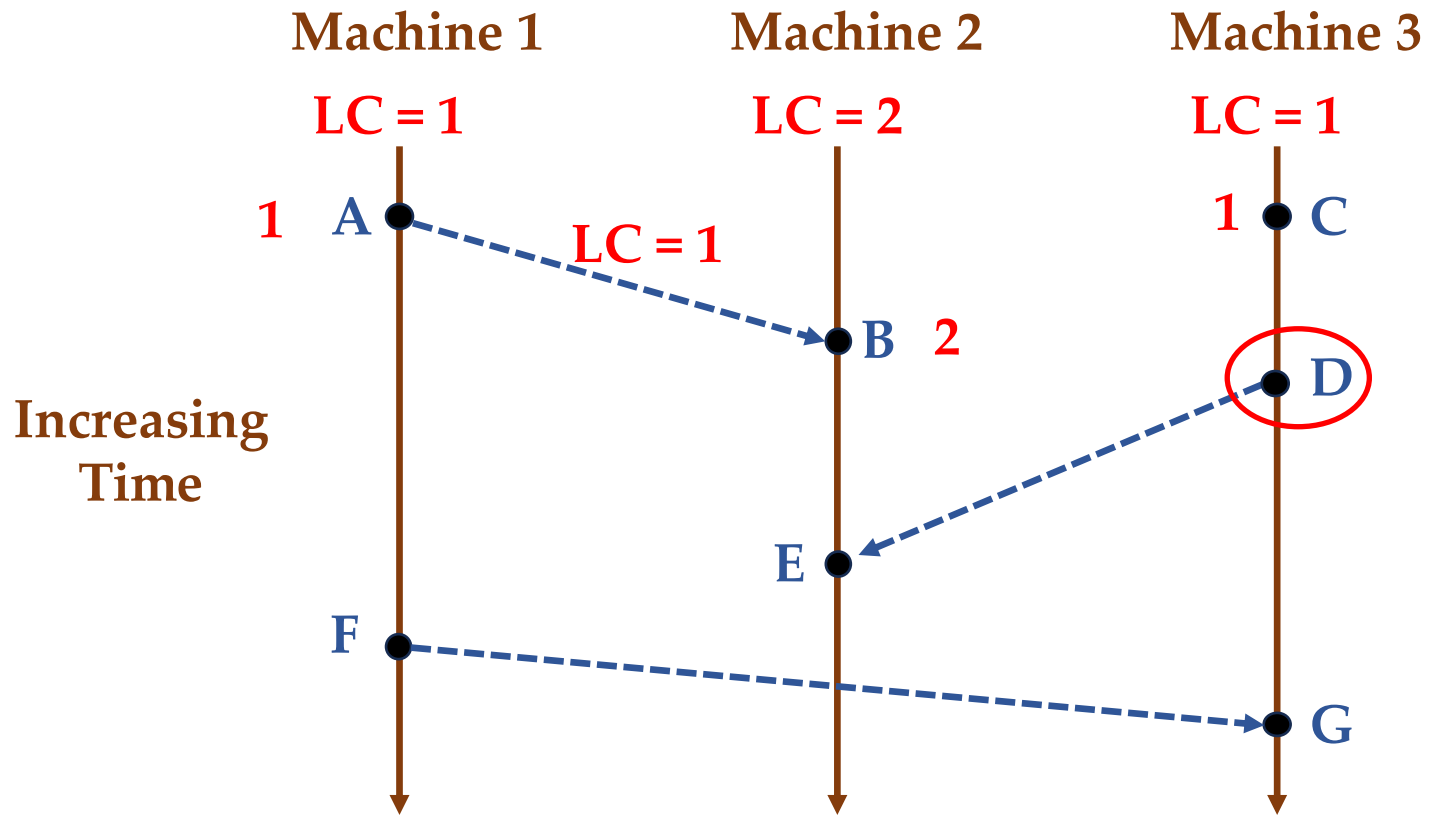
Lamport Clock Algorithm



On Machine 2, Event B is its first event, and it is a receive event!

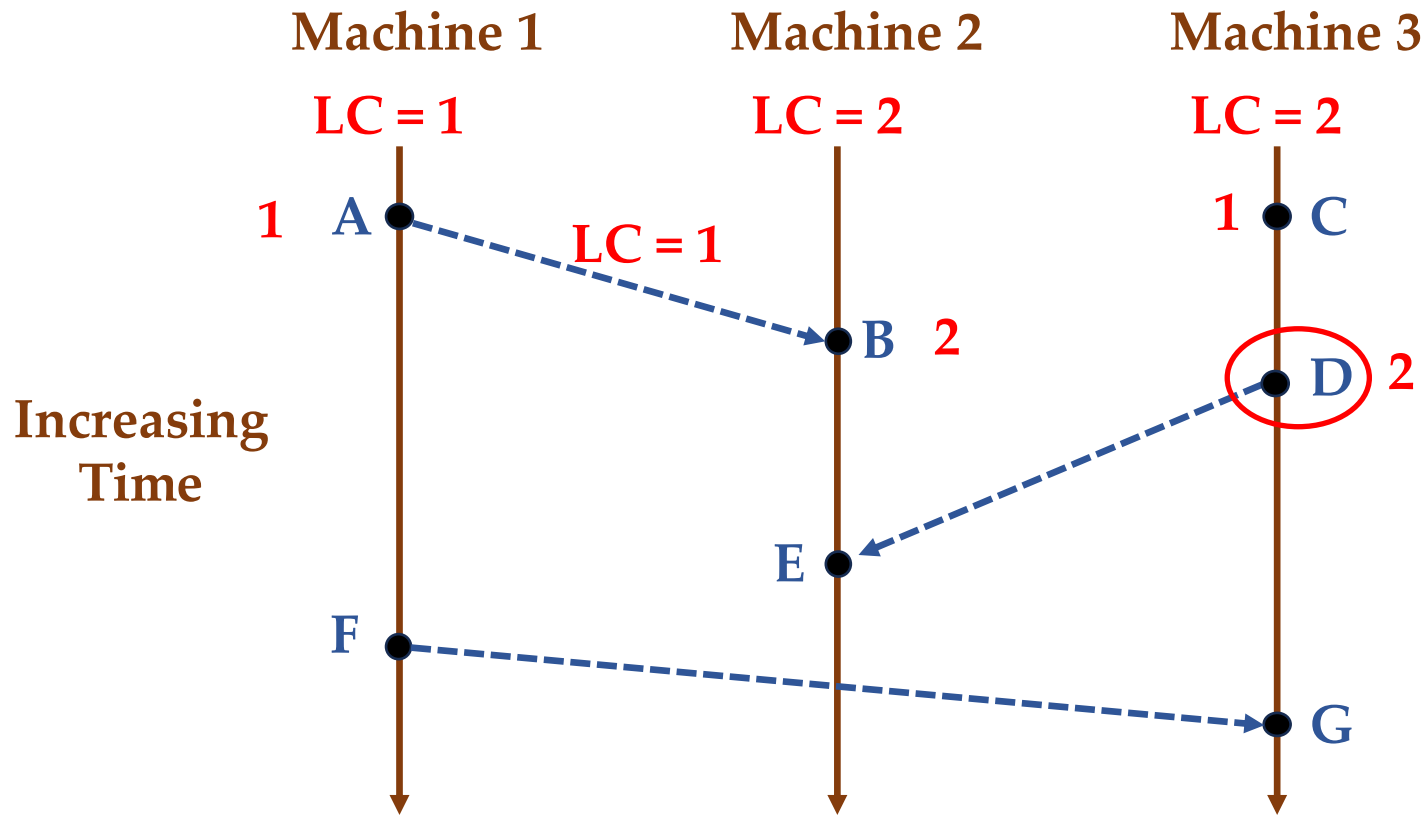
So, we need to do Max(local, receive) + 1.

Lamport Clock Algorithm



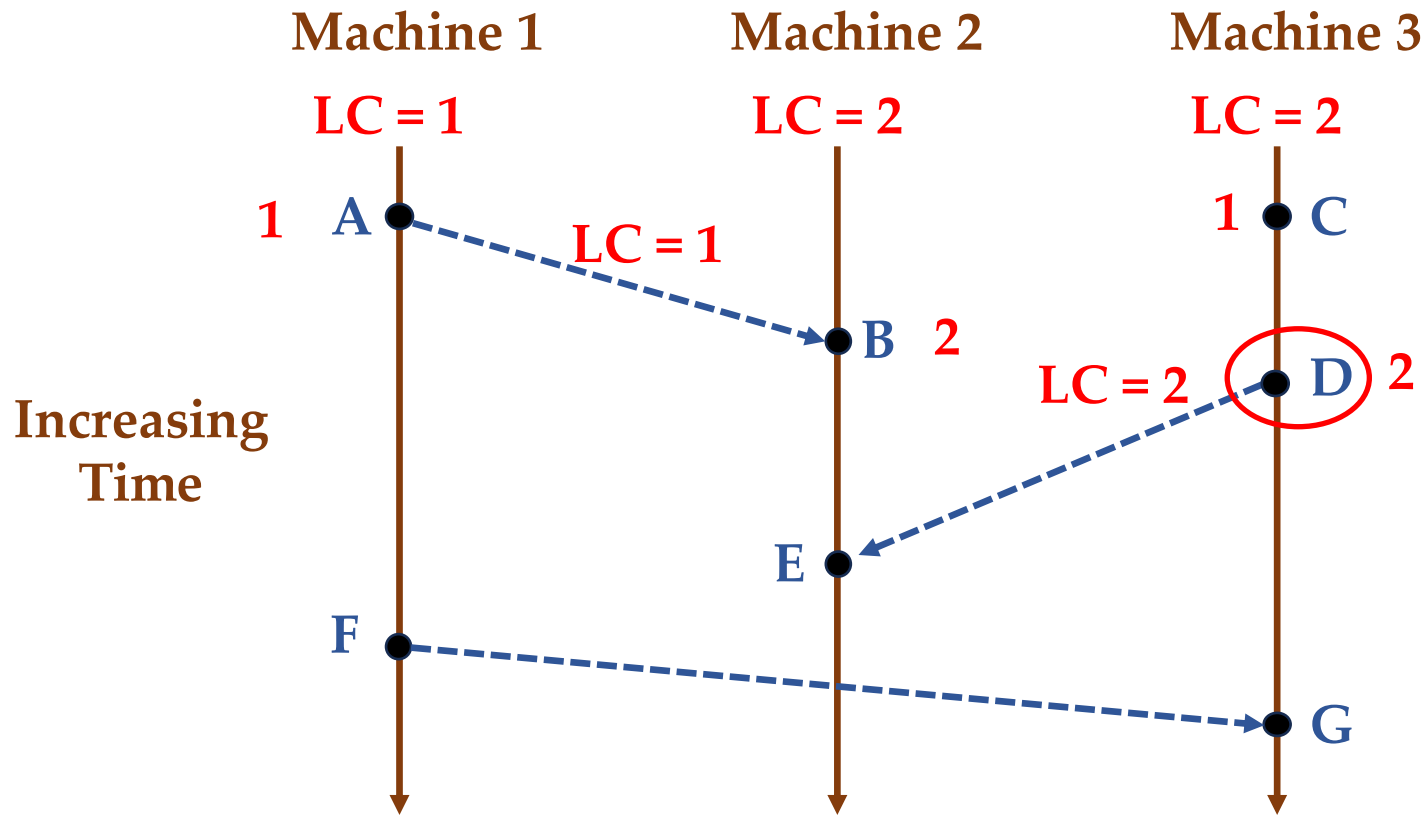
Let's look at Event D next.

Lamport Clock Algorithm



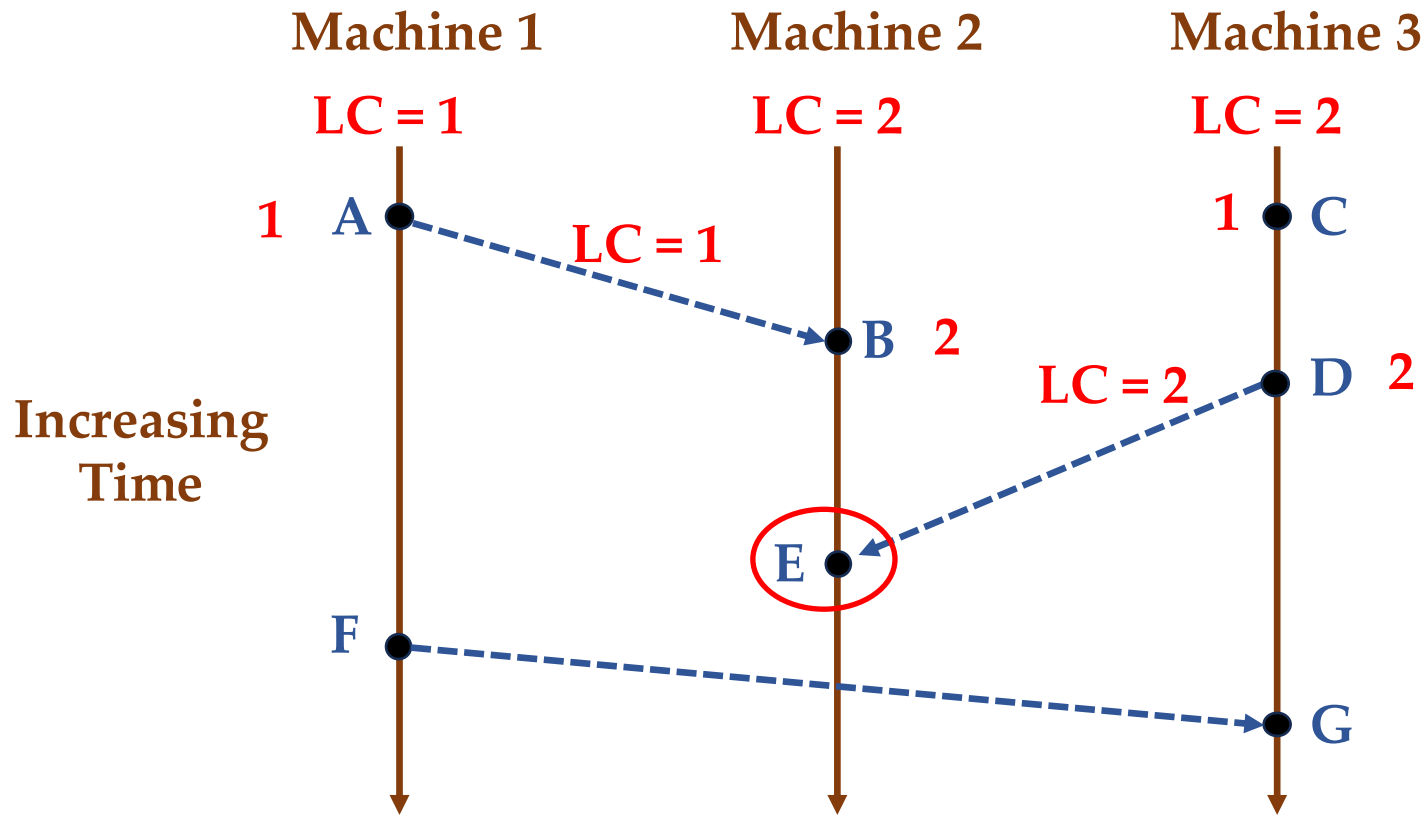
Let's look at Event D next.

Lamport Clock Algorithm



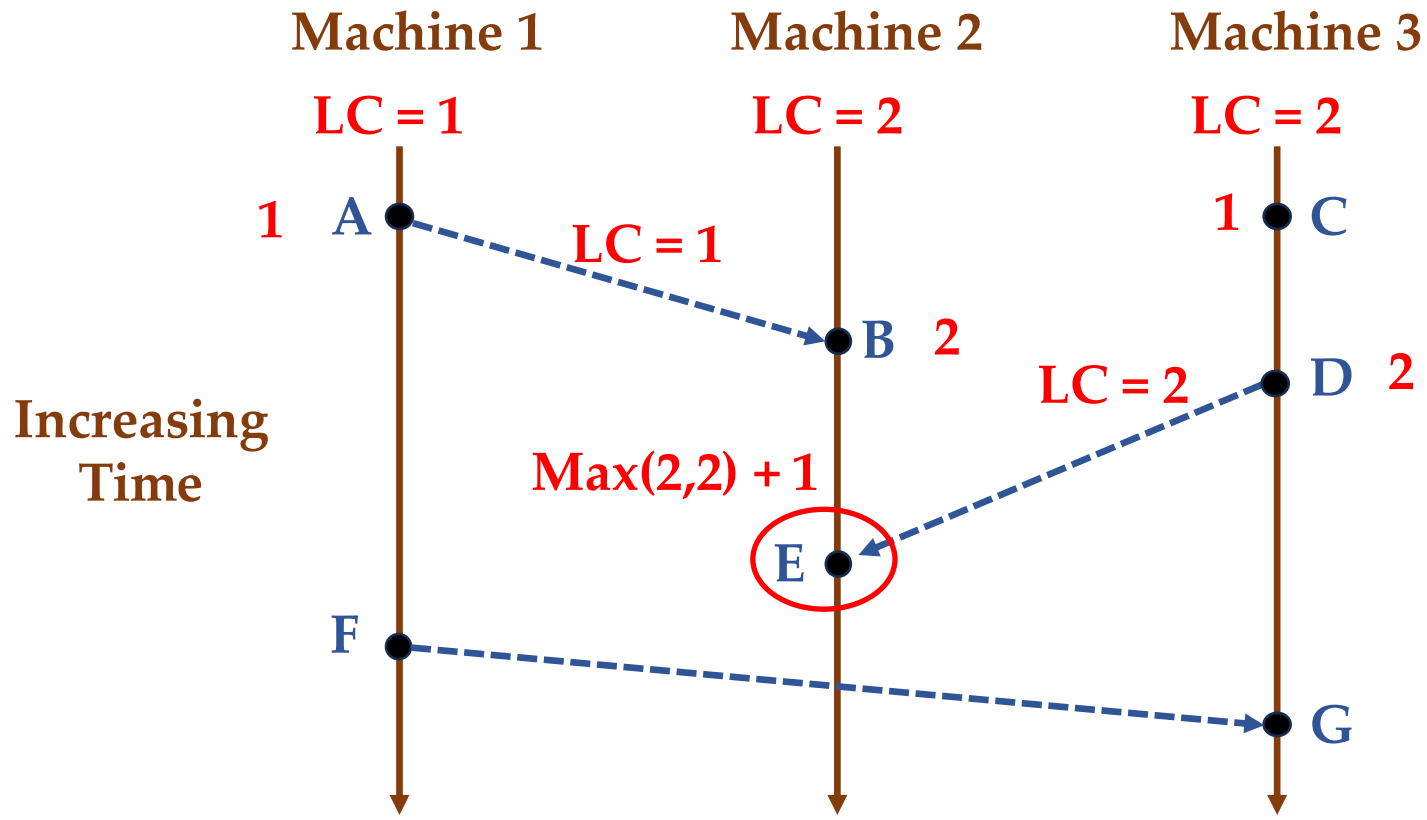
Let's look at Event D next.

Lamport Clock Algorithm



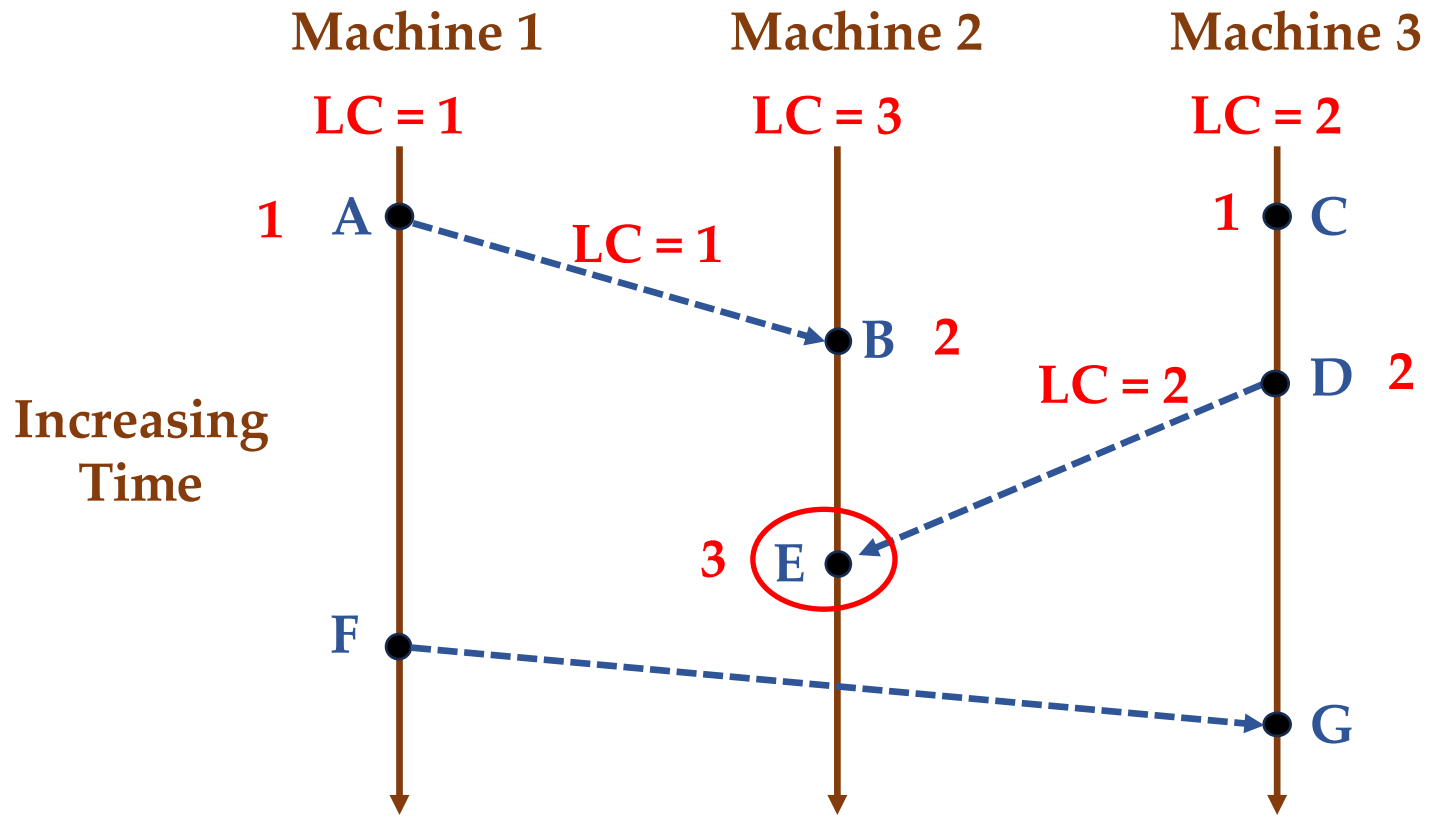
Let's look at Event E next.

Lamport Clock Algorithm



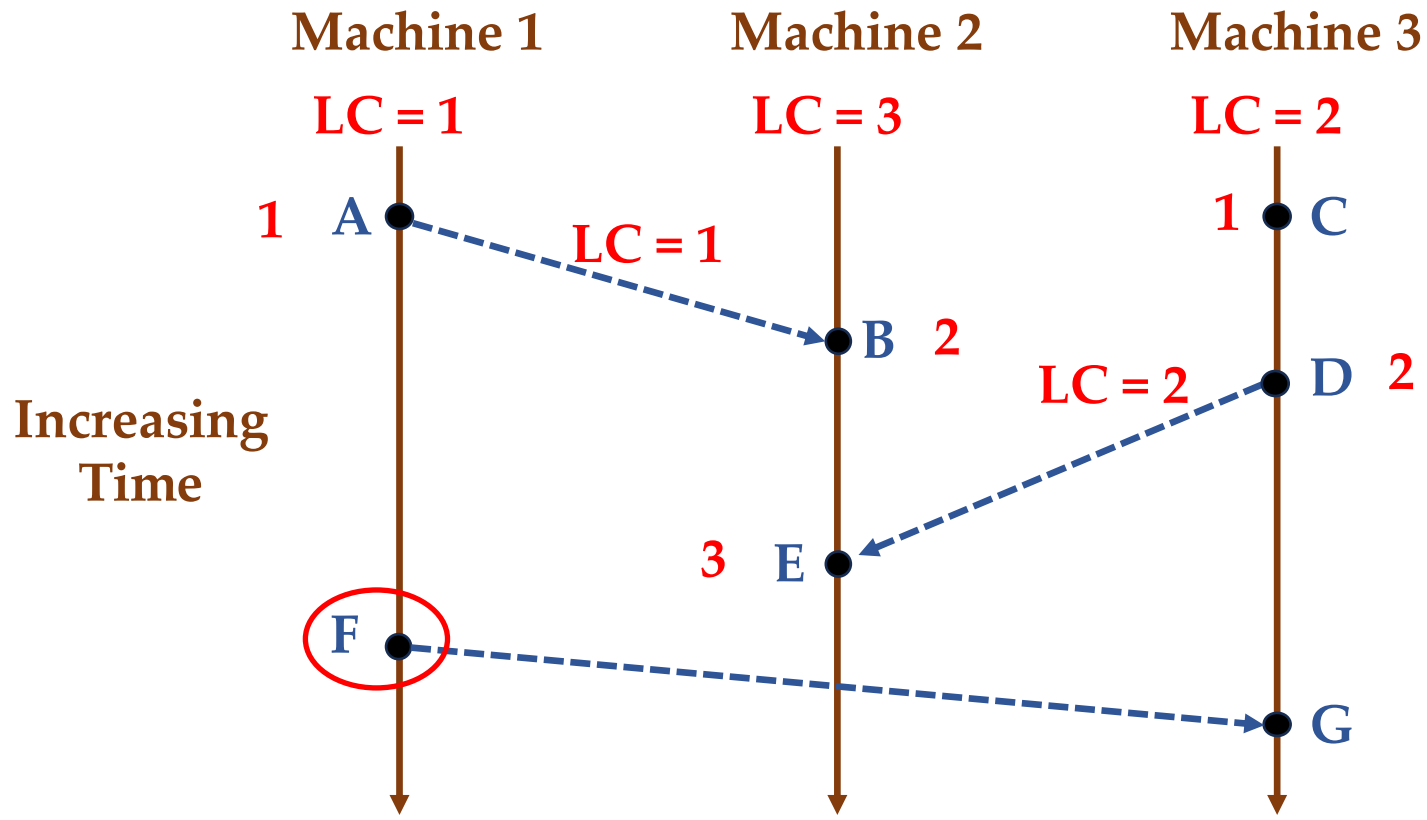
Let's look at Event E next.

Lamport Clock Algorithm



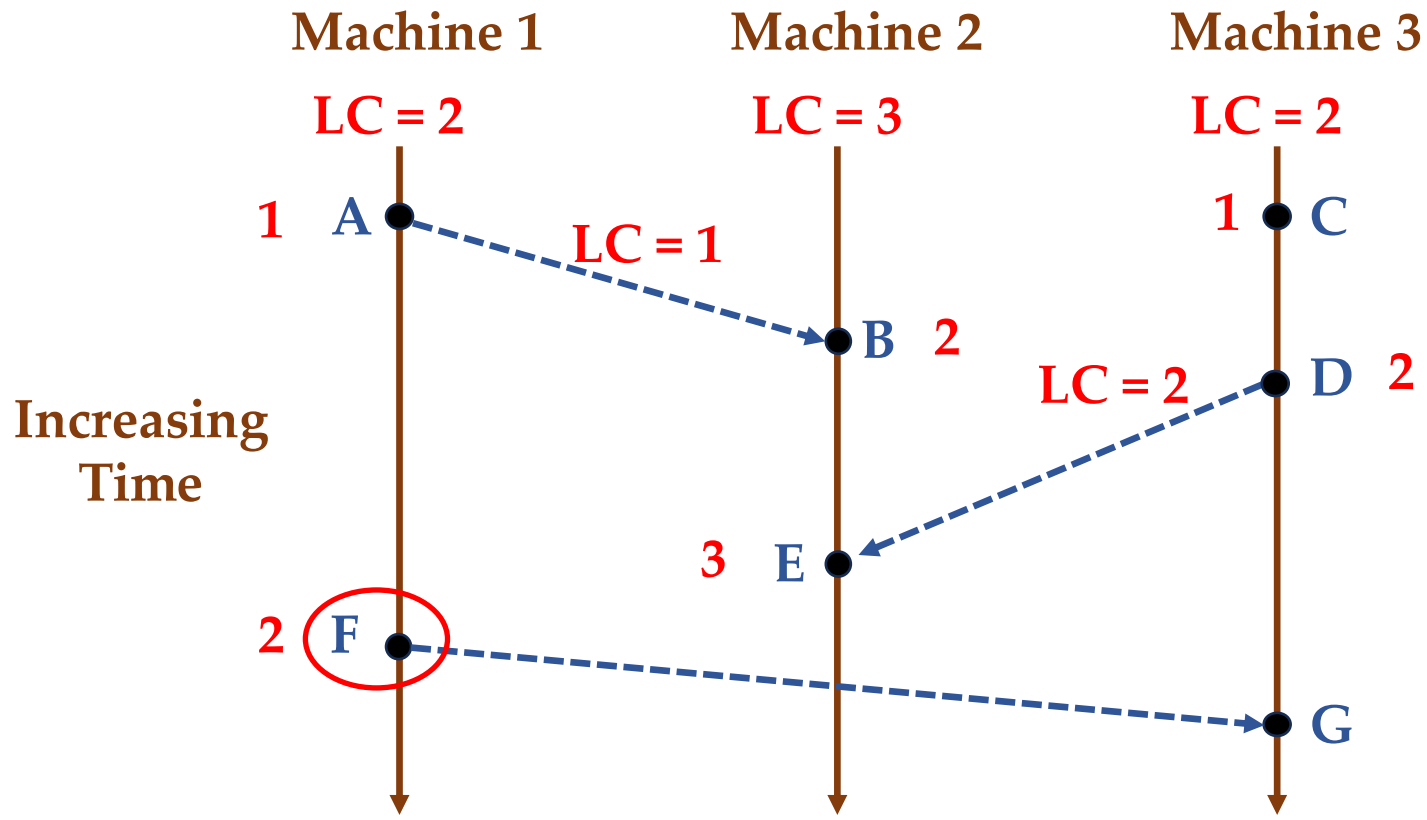
Let's look at Event E next.

Lamport Clock Algorithm



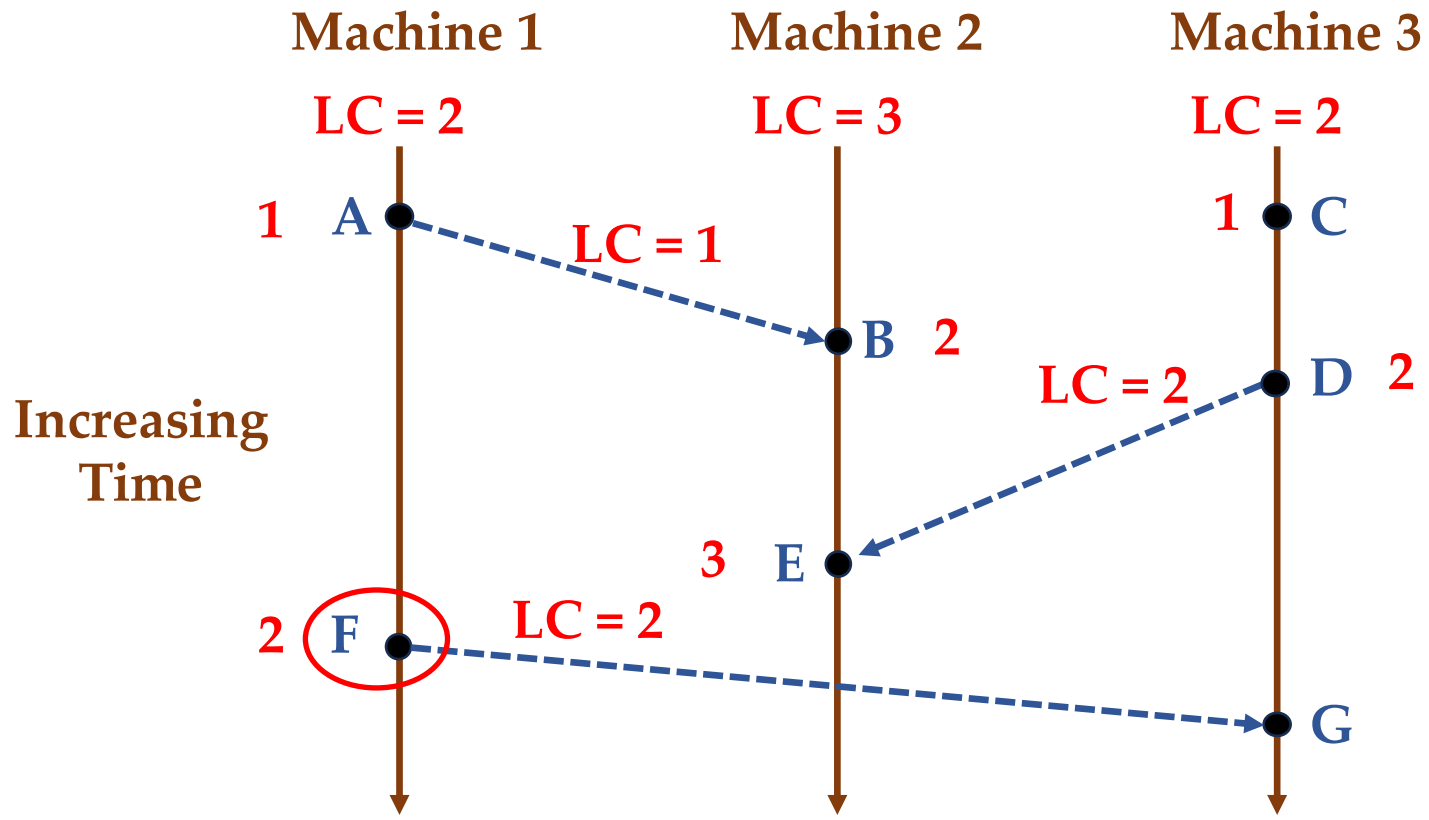
Let's look at Event F next.

Lamport Clock Algorithm



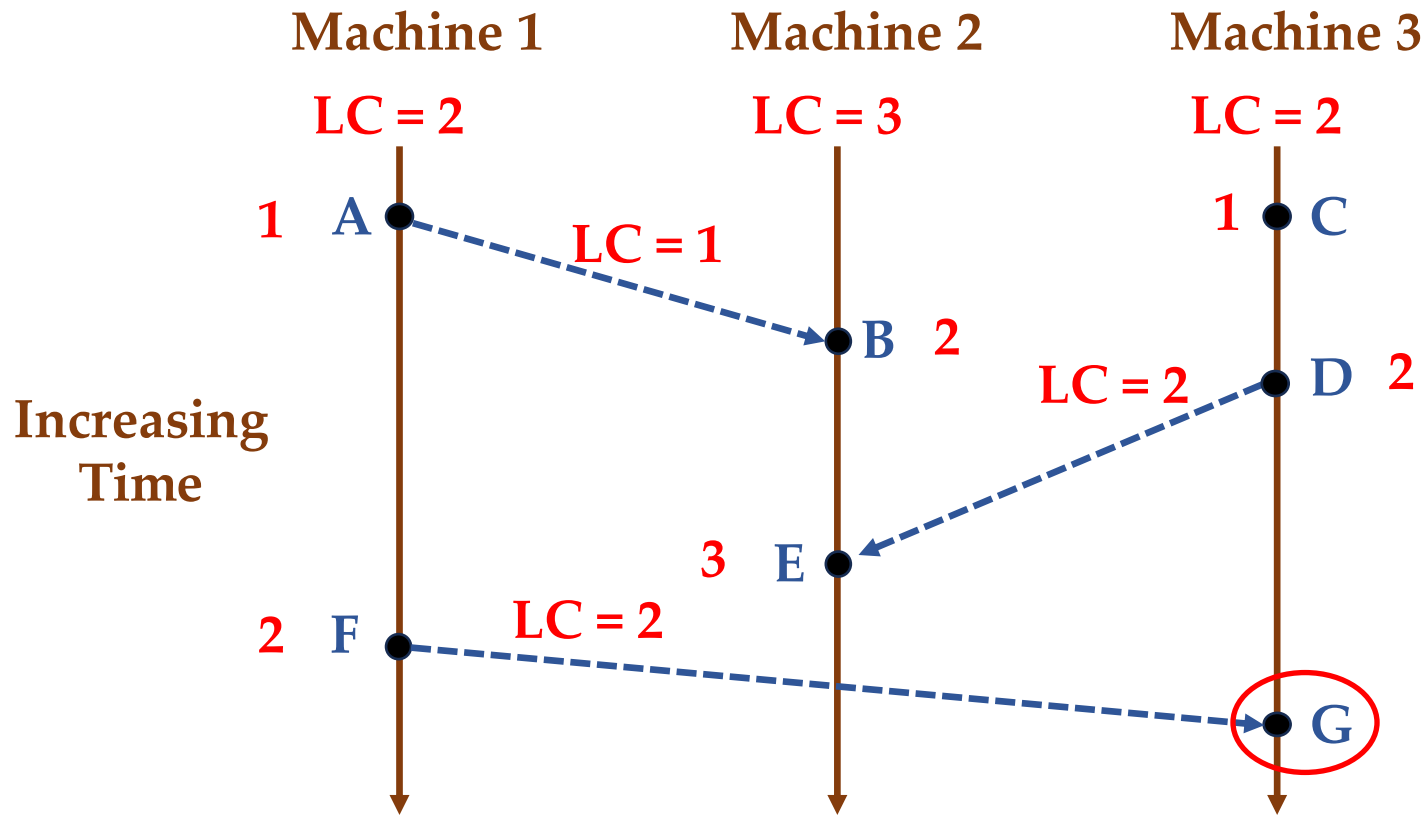
Let's look at Event F next.

Lamport Clock Algorithm



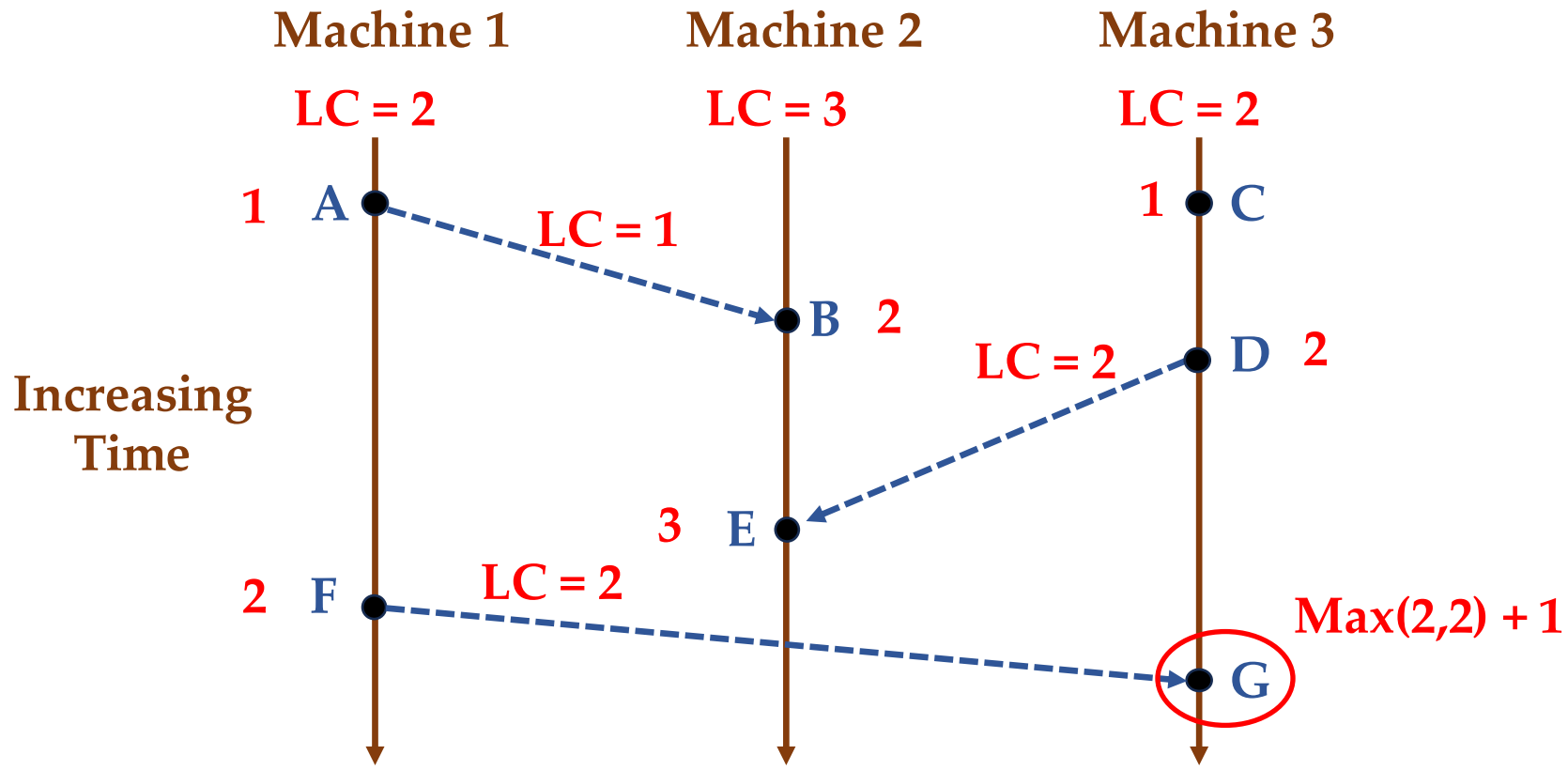
Let's look at Event F next.

Lamport Clock Algorithm



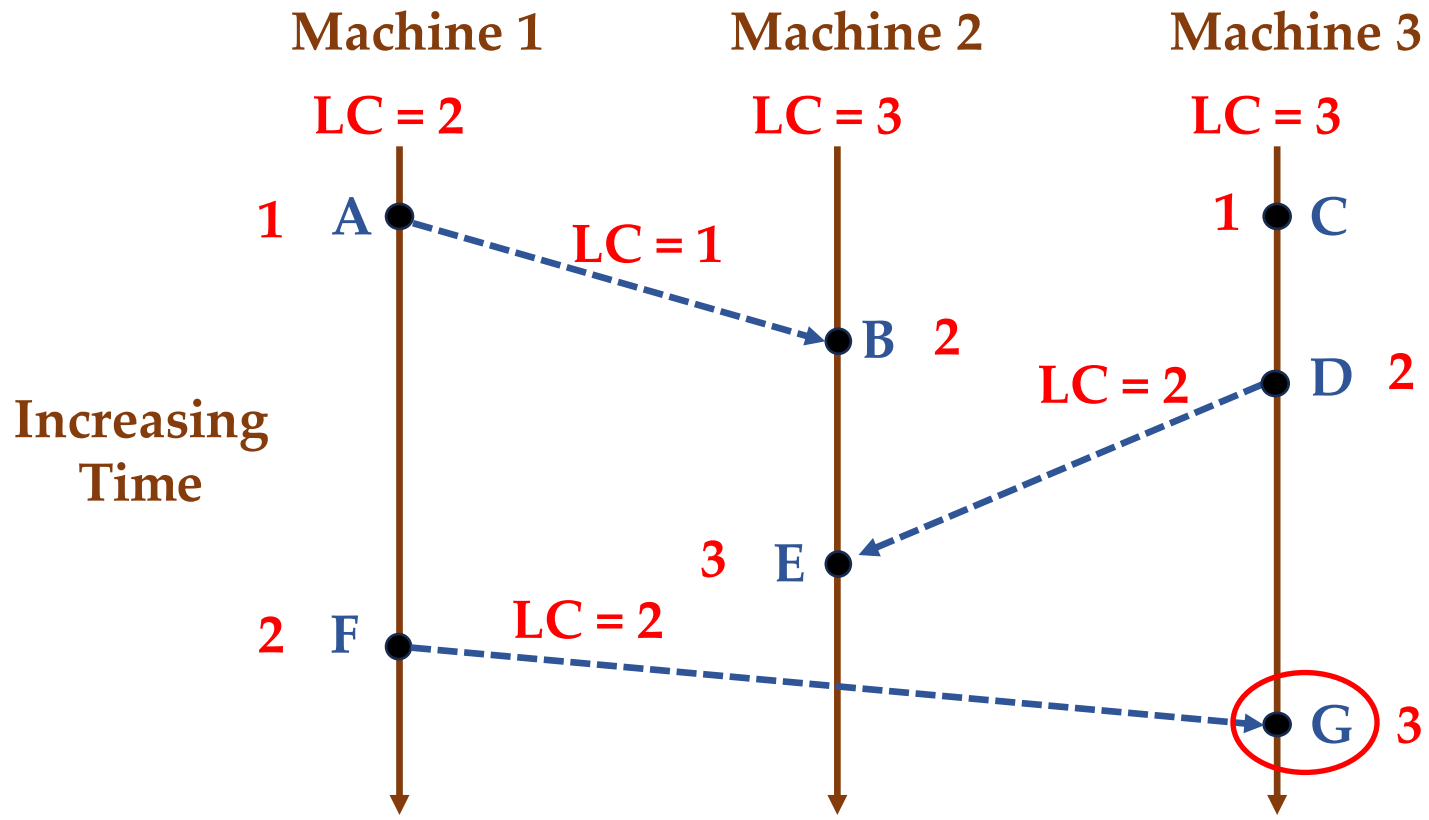
Let's look at Event G next.

Lamport Clock Algorithm



Let's look at Event G next.

Lamport Clock Algorithm

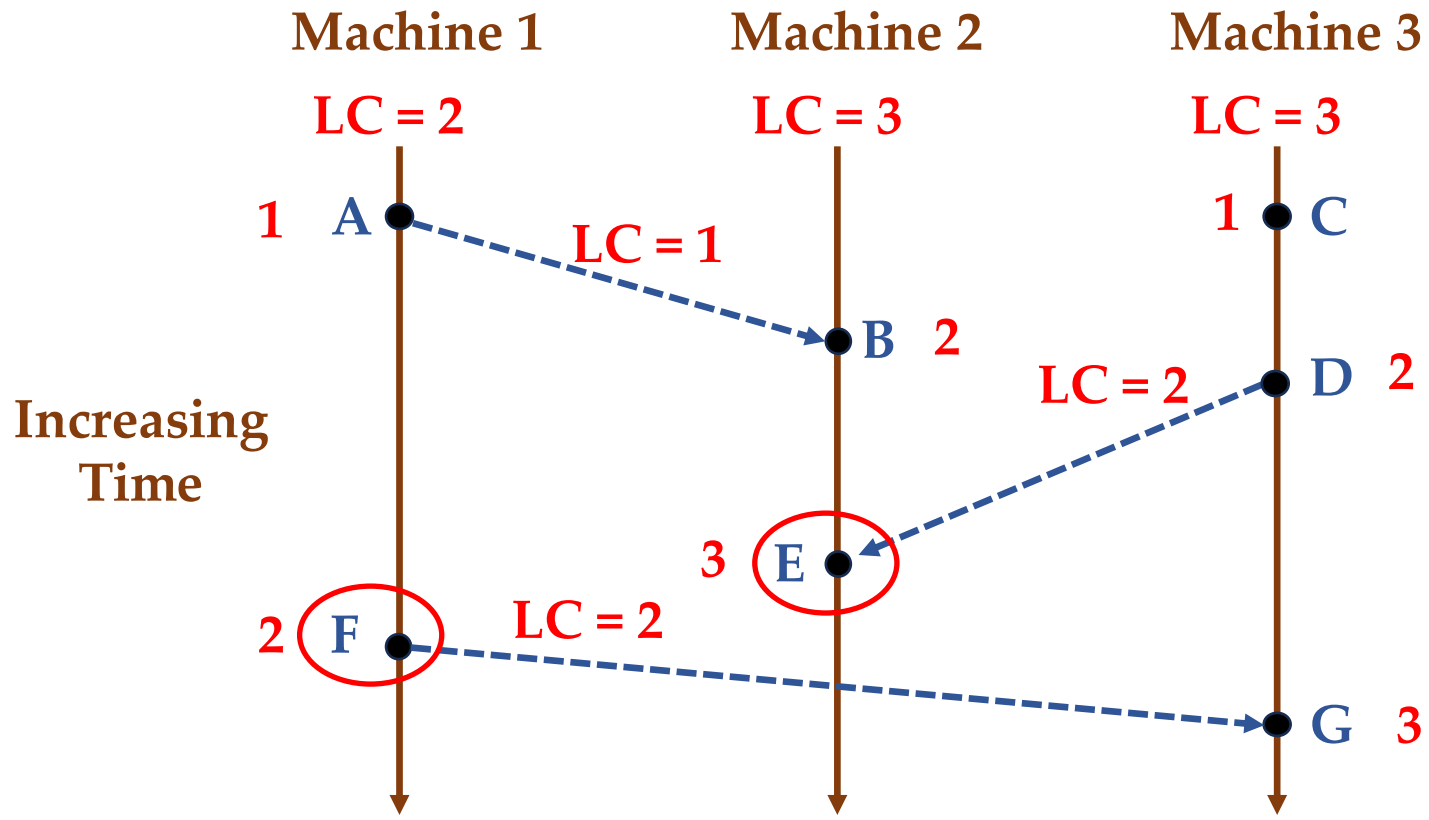


Let's look at Event G next.

Lamport Clocks

- Observe that for the previous diagram, Lamport Clock Condition holds:
  - **If A → B, then LC(A) < LC(B).**
- We say that Lamport Clocks are **consistent with causality**.
- However, does the following holds:
  - **If LC(A) < LC(B), then A → B.**

Lamport Clock Algorithm



For example, events F and E.

LC(F) < LC(E), but F $\not\rightarrow$ E.

Lamport Clocks

- Hence, Lamport Clocks are **consistent with causality,** but

- Lamport clocks **do not characterize causality.**
  - If logical clock of an event is less than the logical clock of another event, then the event with lower Lamport clock happened before the other event.

Are Lamport Clocks Still useful?

Are Lamport Clocks Still useful?

- Yes, you can use implication to deduce about the events.
- Recall that if $\mathbf{A} \Longrightarrow \mathbf{B}$ holds, then $\neg\, \mathbf{B} \Longrightarrow \neg\, \mathbf{A}$ also holds.

Are Lamport Clocks Still useful?

- Yes, you can use implication to deduce about the events.
- Recall that you can also test for the contrapositive of $\mathbf{A \Longrightarrow B.}$
  - i.e., $\mathbf{\neg B \Longrightarrow \neg A.}$
- For example, $\mathbf{LC(A) \not< LC(B) \Longrightarrow A \not\rightarrow B.}$
- This is a particularly useful property as if Lamport clock of event A is not less that another event B, then A did not happen before B.

Vector Clocks

- Vector Clocks are consistent with causality and they characterize causality.

- As the name suggests, they are vectors of integers.
  - Vector Clock of Event A denoted as VC(A).
  - The size of the vector is equal to number of machines/processes.
  - One integer per machine in the vector.

- Clock Condition:
  - **A → B ⇔ VC(A) < VC(B).**
  - Notice the **if and only if** condition.

Vector Clocks

- Vector Clocks are consistent with causality and they characterize causality.

- As the name suggests, they are vectors of integers.
  - Vector Clock of Event A denoted as VC(A).
  - The size of the vector is equal to number of machines/processes.
  - One integer per machine in the vector.

- Clock Condition:
  - **A → B ⇔ VC(A) < VC(B).**
  - Notice the **if and only if** condition.

Vector Clock Algorithm

- The following algorithm can help to assign each event a Lamport clock.

1. Every process/machine keeps a vector of integers (one for each process), initialized to 0.

2. When an event occurs on a process, it increments its own position in the vector by 1 and assigns that value to the event.

3. At the time of sending a message, the process sends its complete vector.

4. On receiving a message, a process sets its vector as Maximum(local_vector, received_vector) + 1.

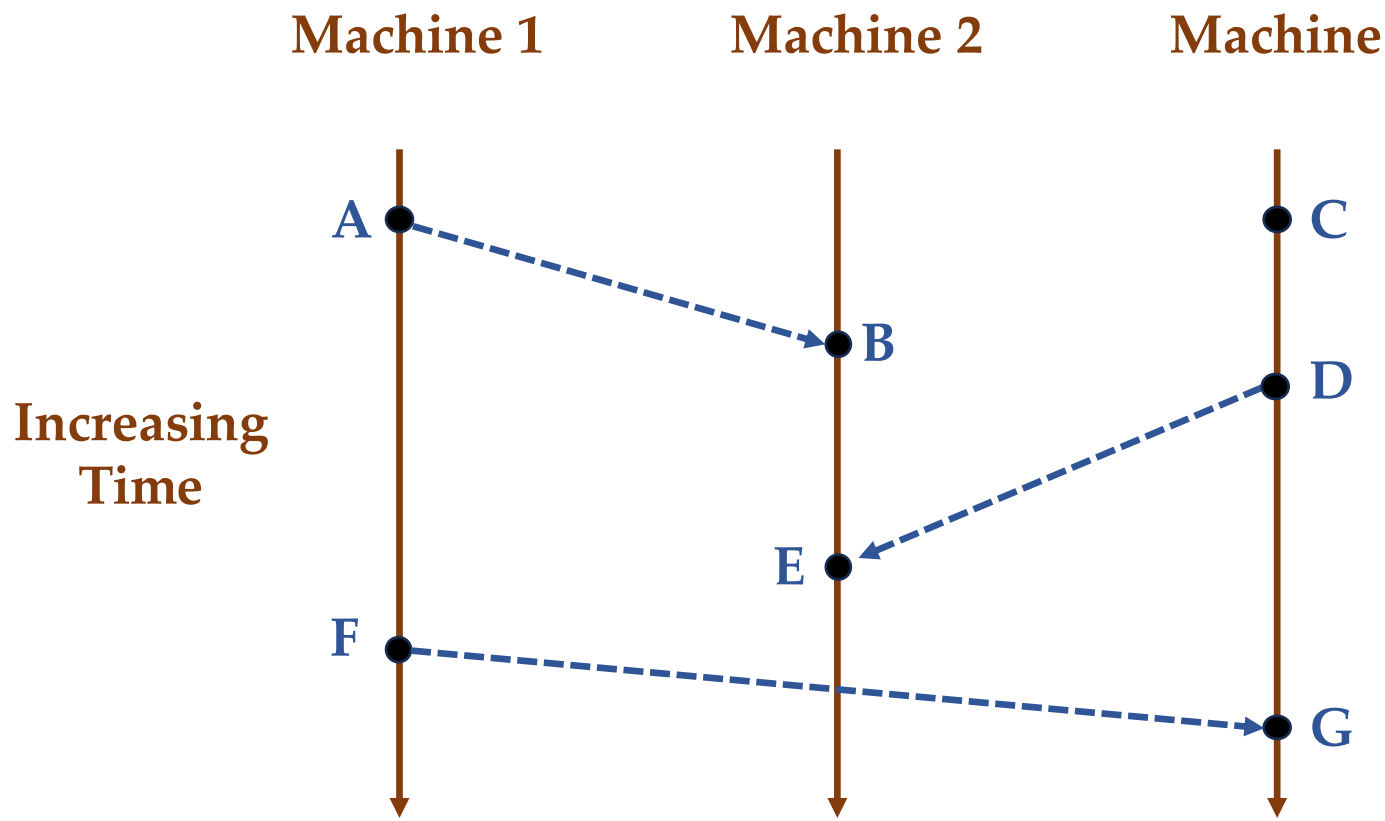
Vector Clock Algorithm

- How to find maximum of two events?
- Say vectors: [1, 5, 3] and [7, 3, 8].

Vector Clock Algorithm

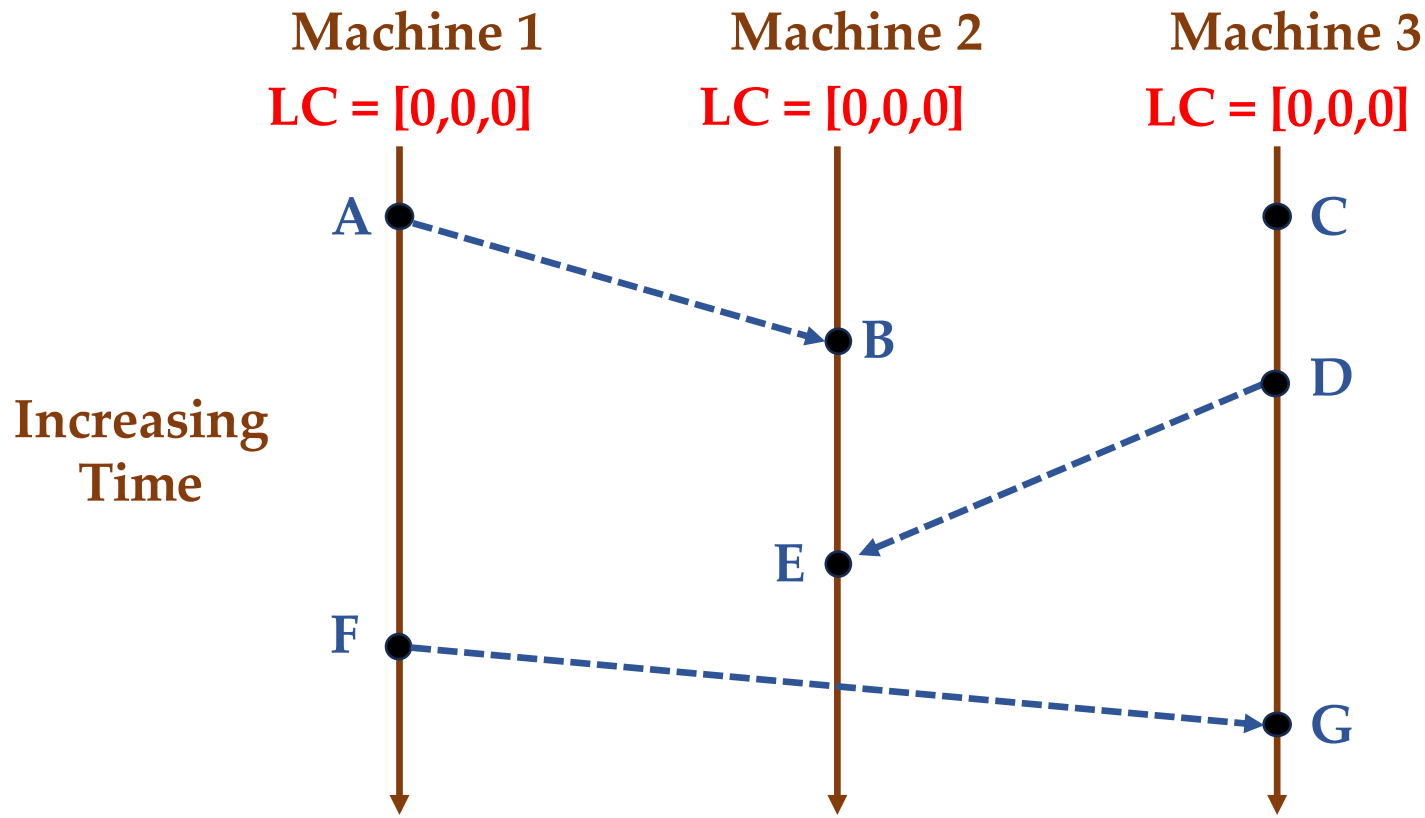
- How to find maximum of two events?
- Say vectors: [1, 5, 3] and [7, 3, 8].
- Do position wise maximum → for each position find the maximum.
- So for above → [7,5,8].

Vector Clock Algorithm



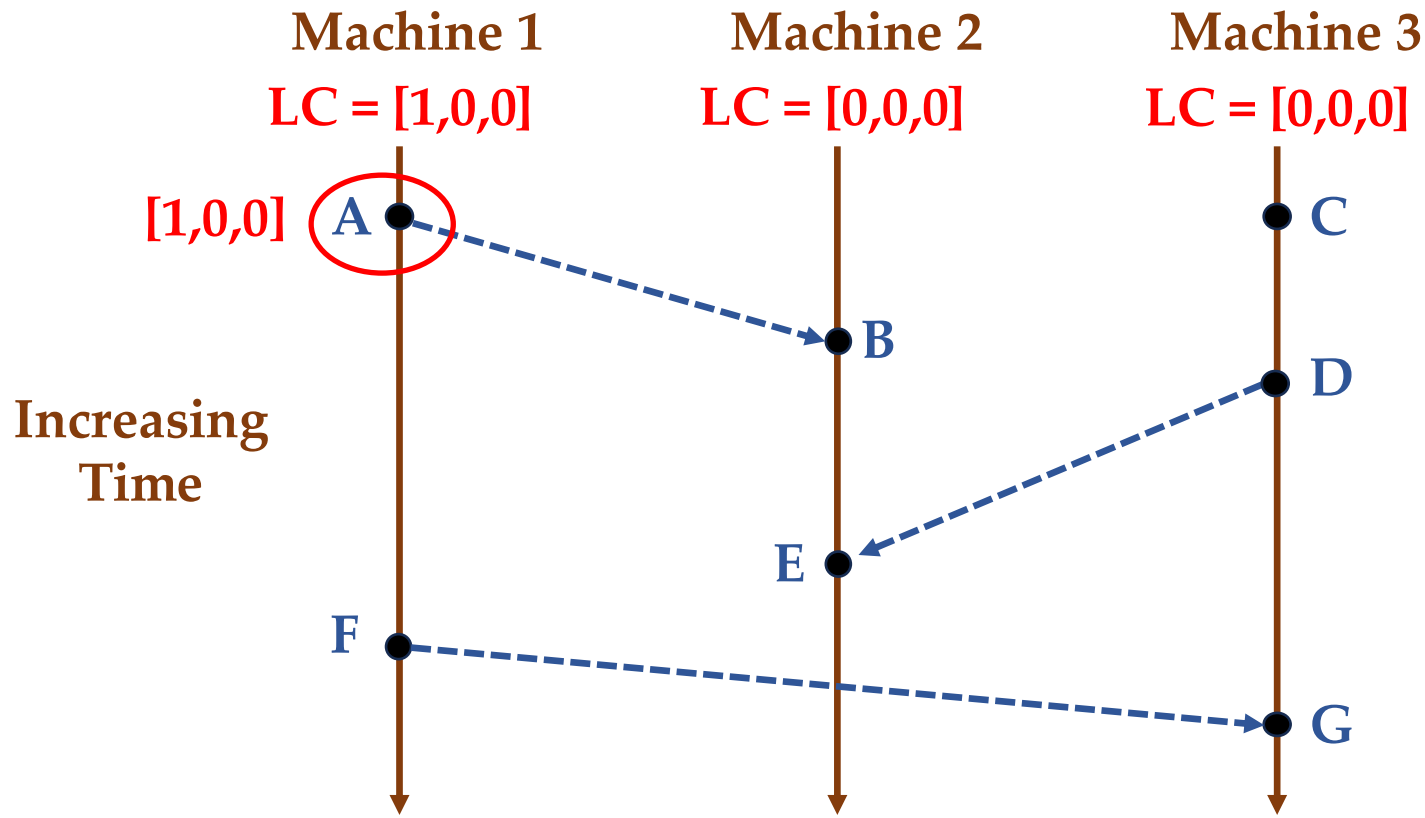
Let's apply apply Vector Clock Algorithm on this example.

Vector Clock Algorithm



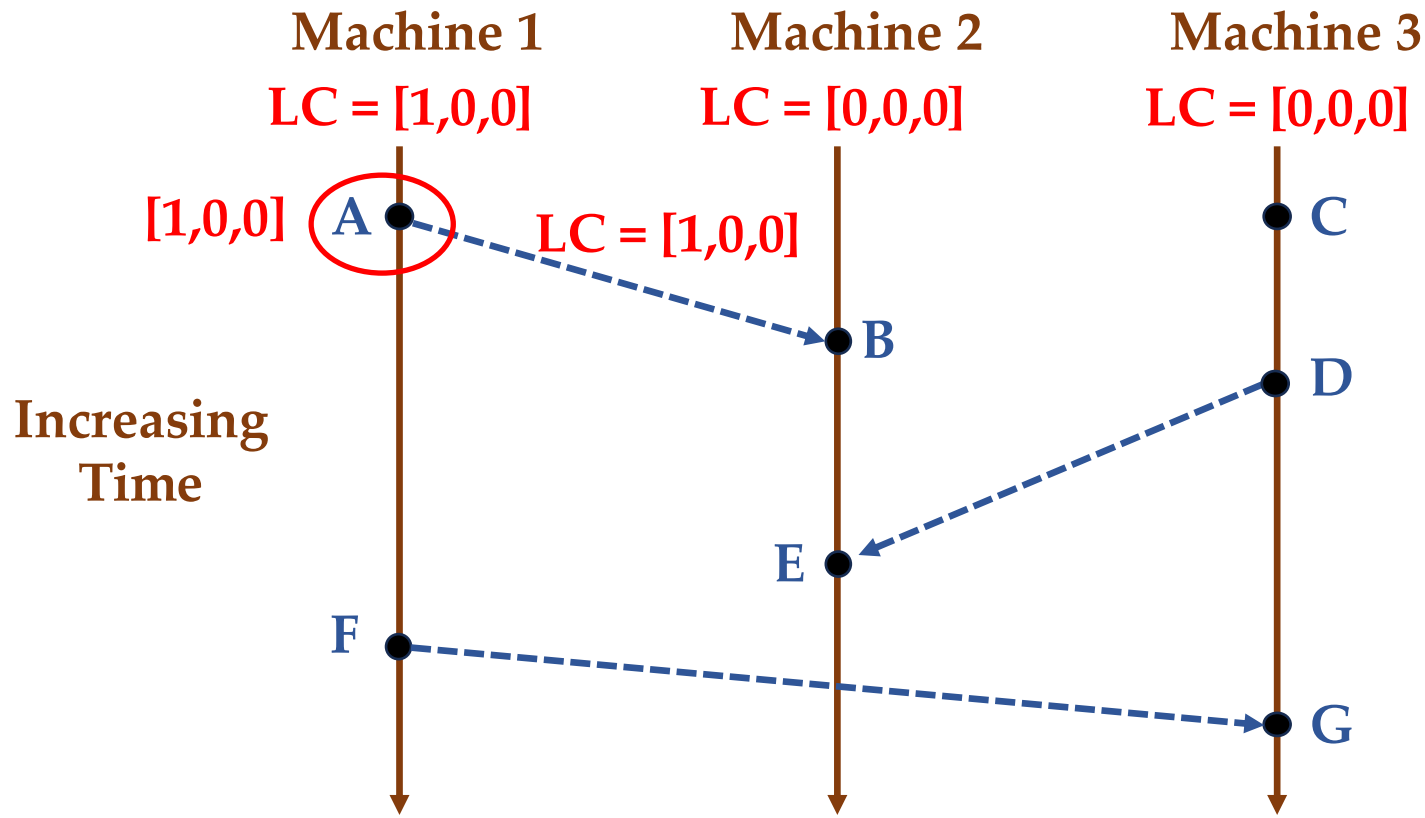
Initialize Vector Clock at each machine.

Vector Clock Algorithm



On Machine 1, the first event is A, so we increment Machine 1's position in the vector clock and assign A a clock [1,0,0].

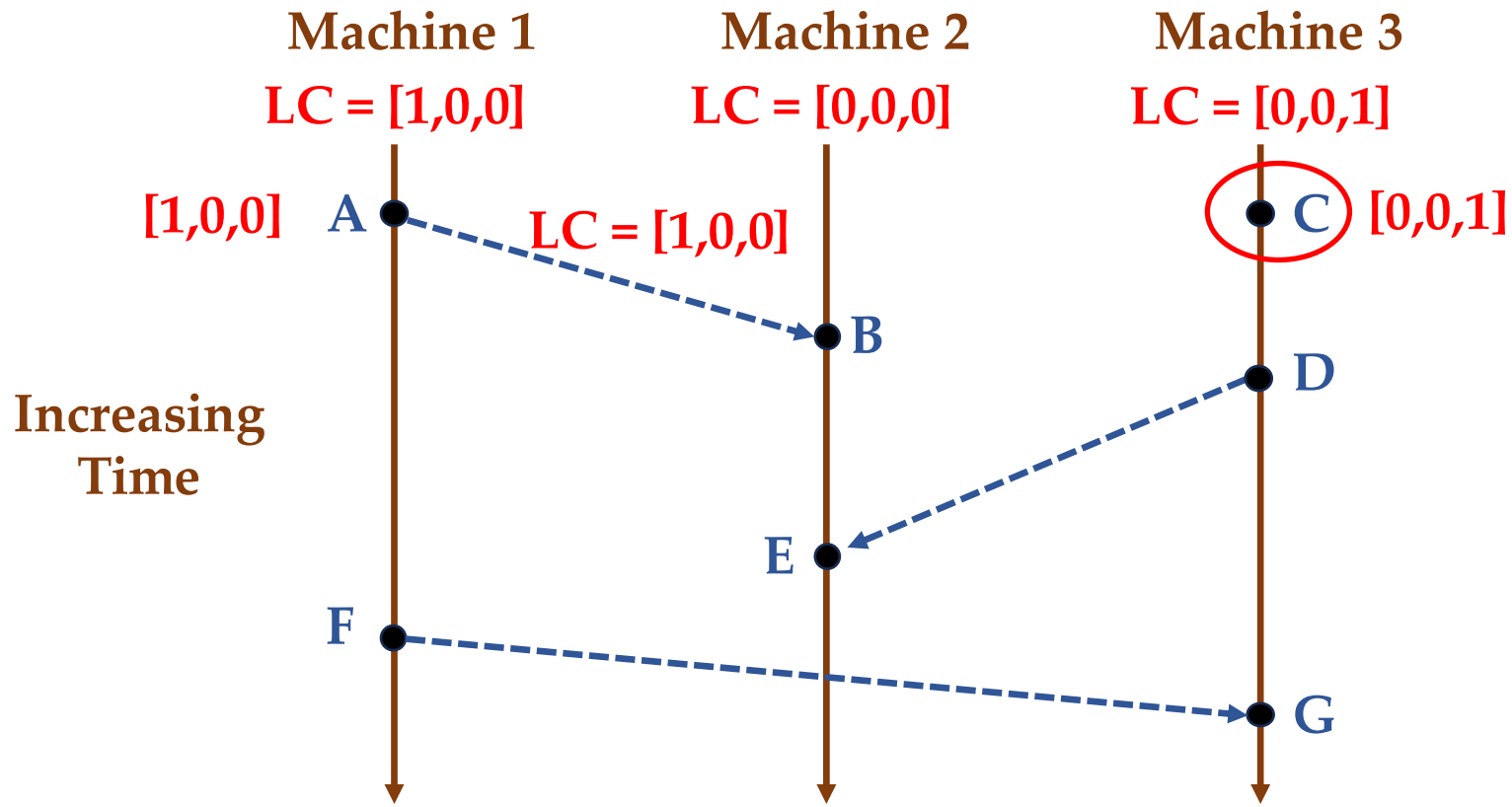
Vector Clock Algorithm



On Machine 1, the first event is A, so we increment Machine 1's position in the vector clock and assign A a clock [1,0,0].

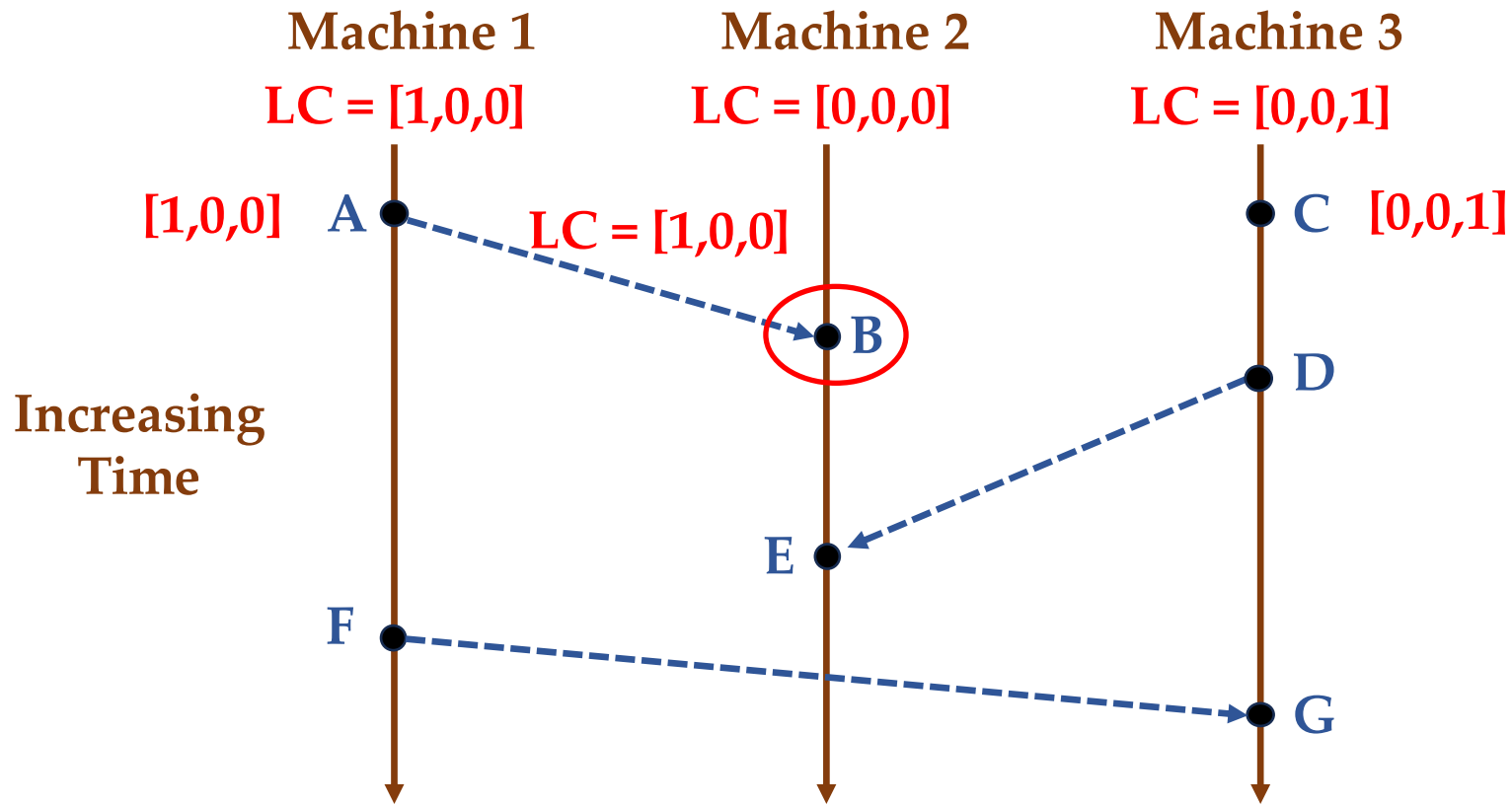
Further, this is a send event, so we send its Lamport Clock to Machine 2.

Vector Clock Algorithm



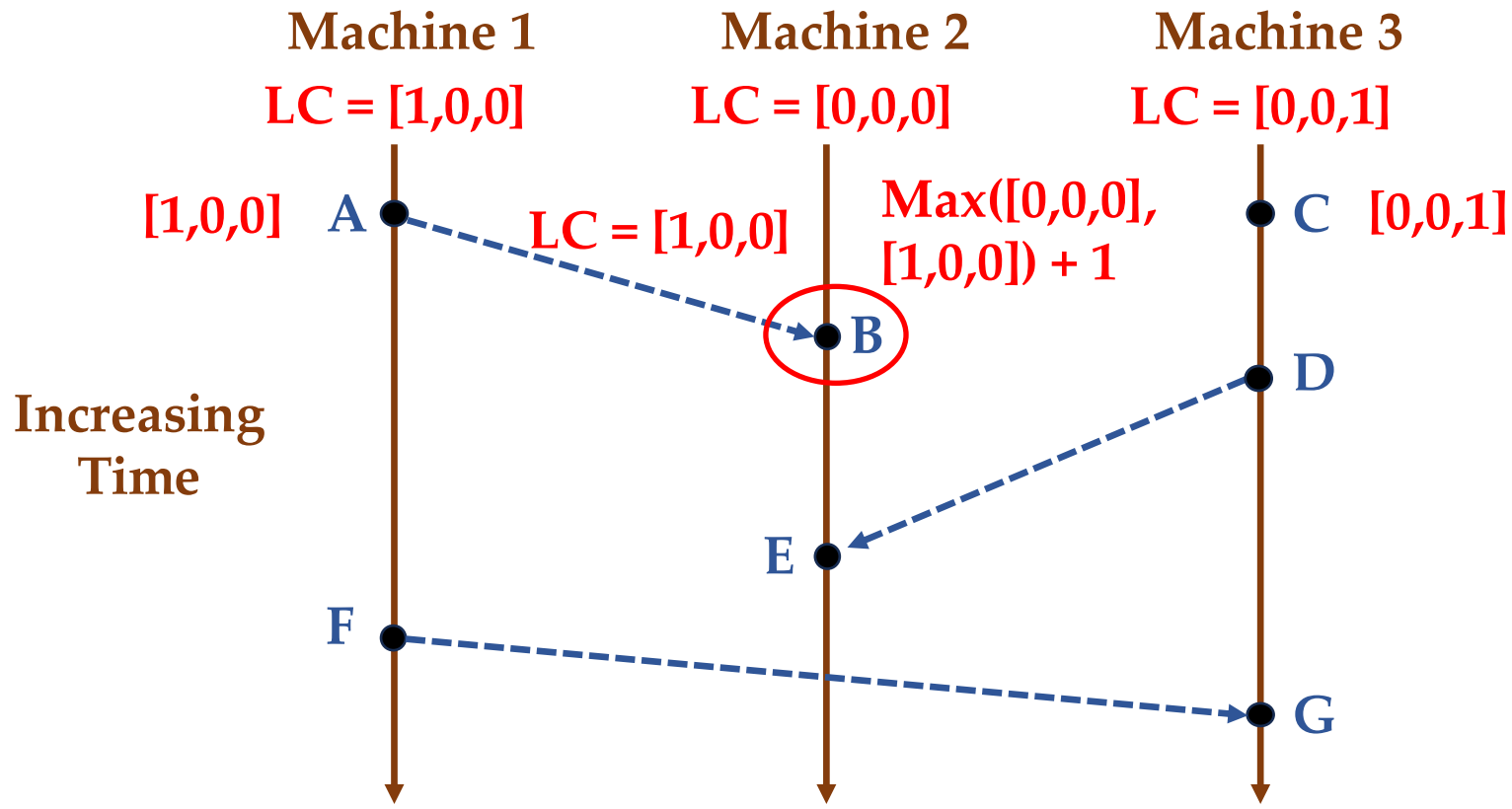
Similarly, on Machine 3, the first event is C, so we increment Machine 3's position in the vector clock and assign C a clock [0,0,1].

Vector Clock Algorithm



On Machine 2, Event B is its first event, and it is a receive event!

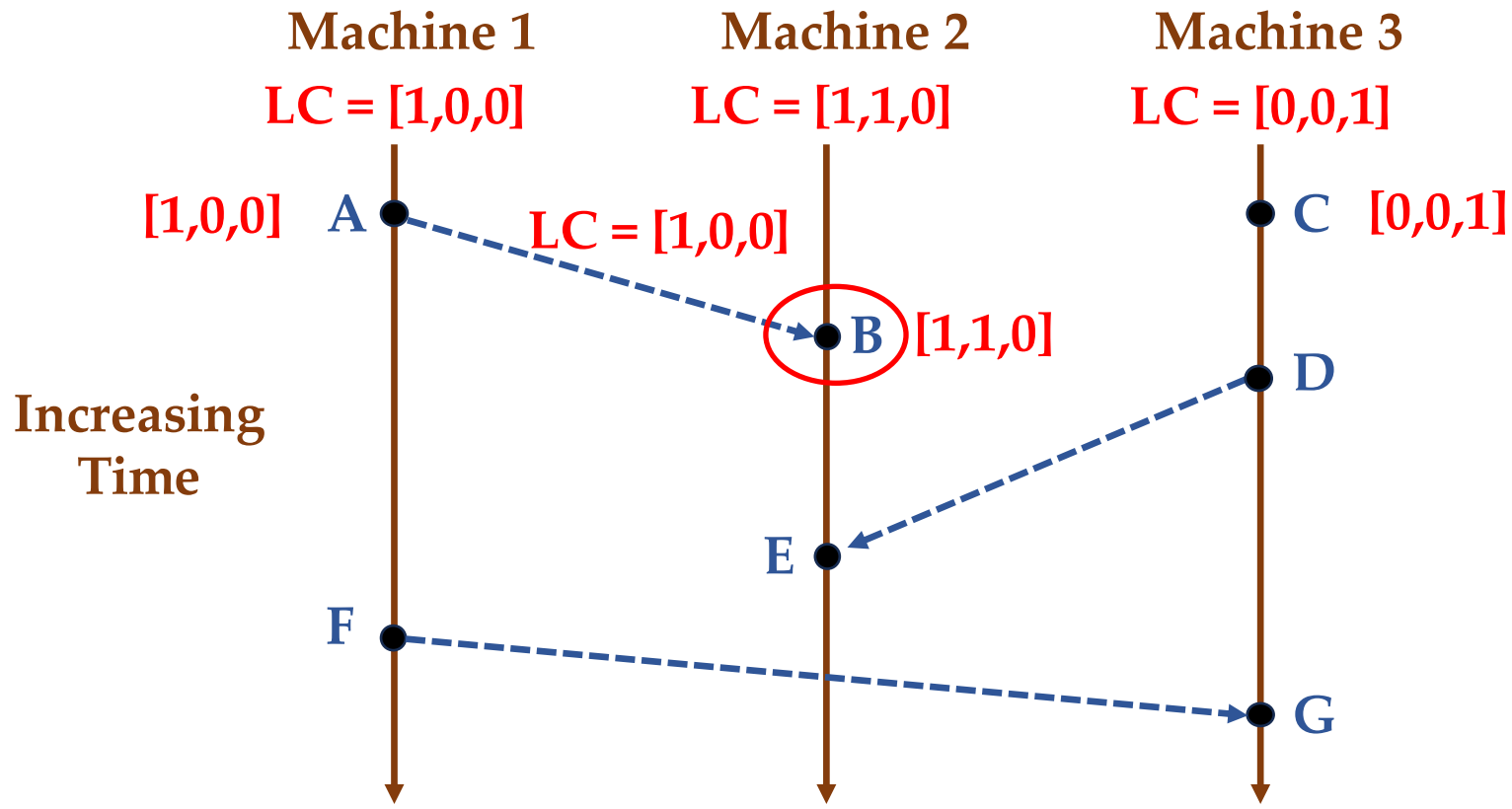
Vector Clock Algorithm



On Machine 2, Event B is its first event, and it is a receive event!

So, we need to do Max(local, receive) + 1.

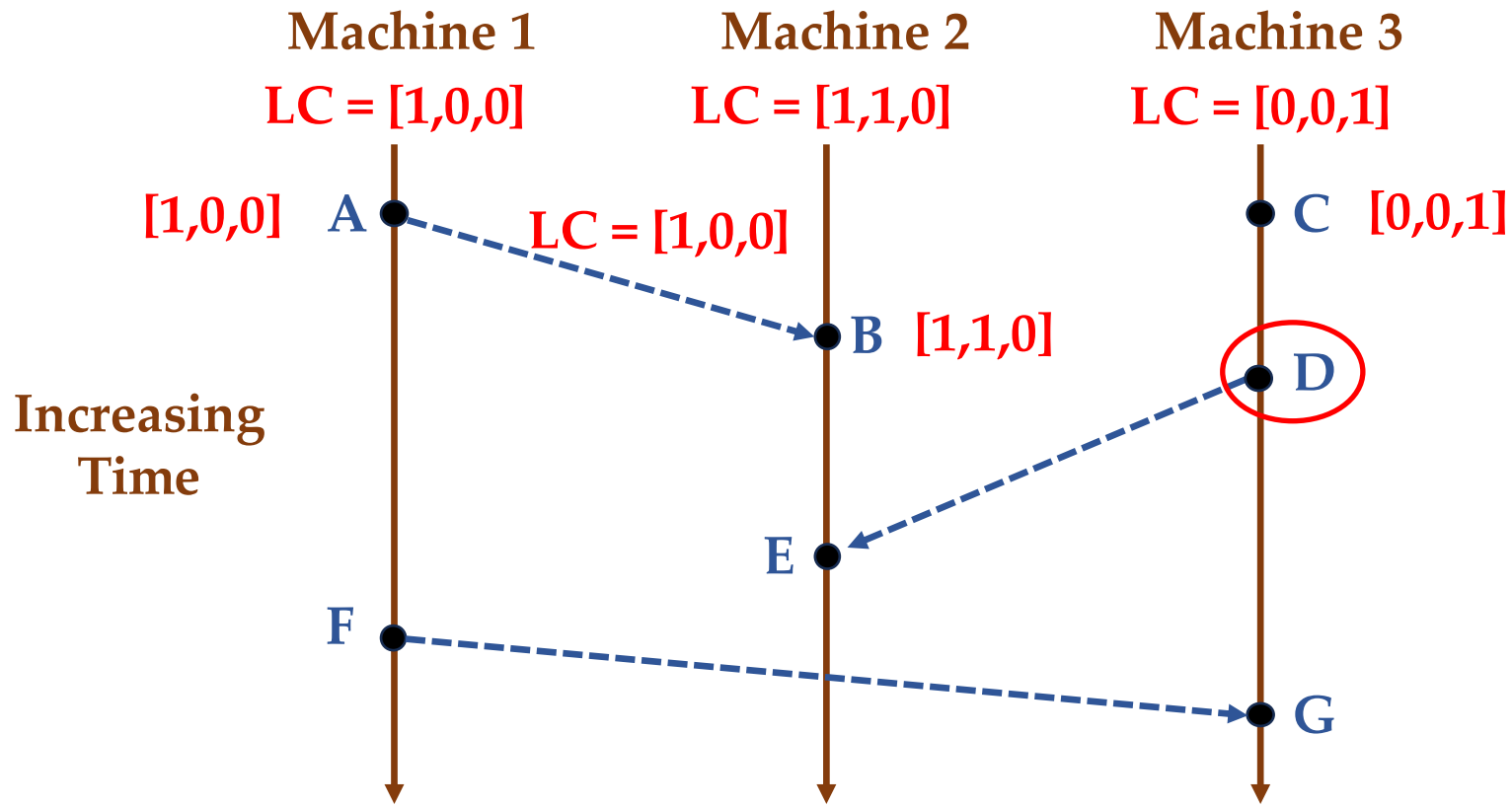
Vector Clock Algorithm



On Machine 2, Event B is its first event, and it is a receive event!

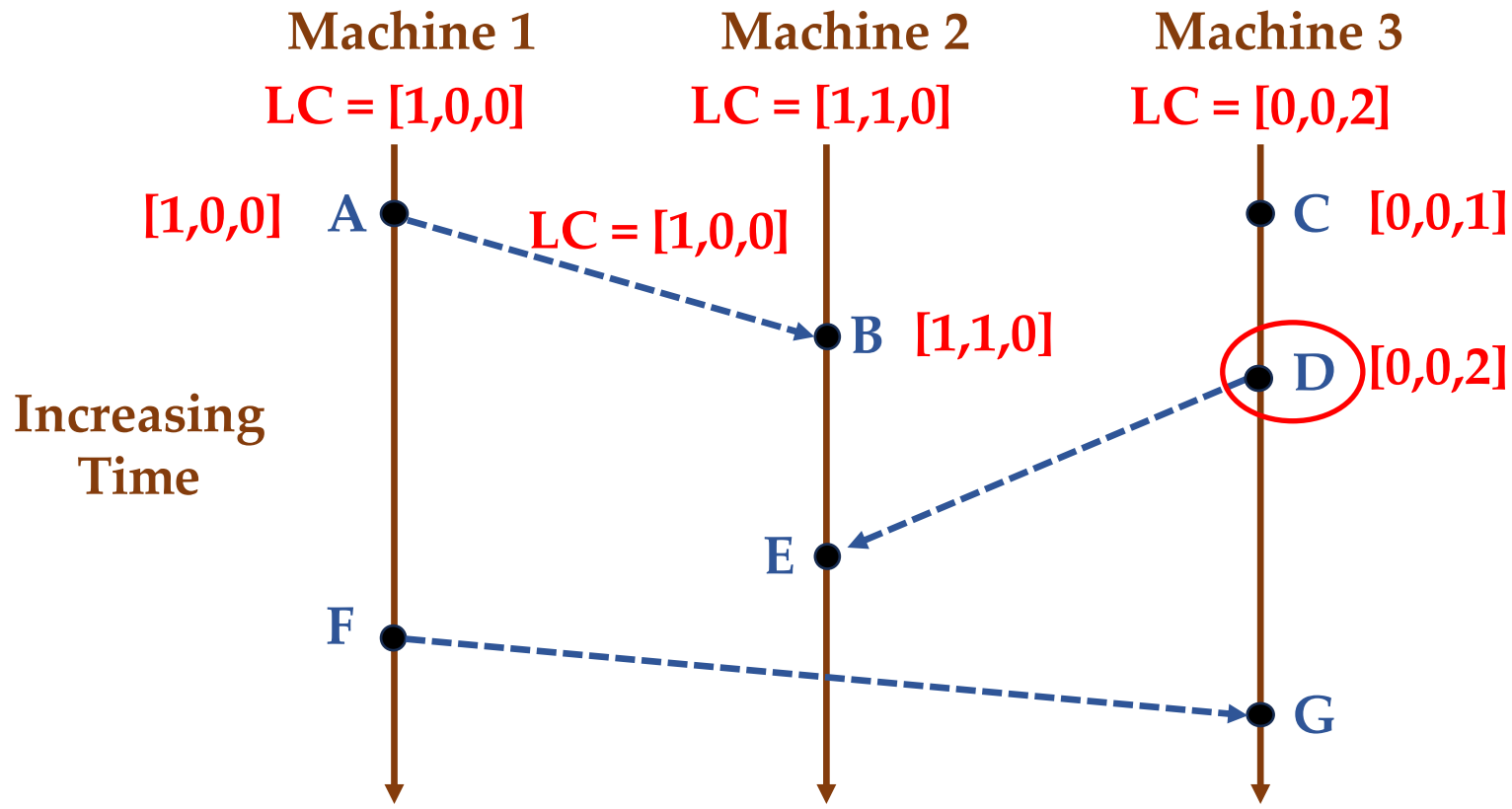
So, we need to do Max(local, receive) + 1.

Vector Clock Algorithm



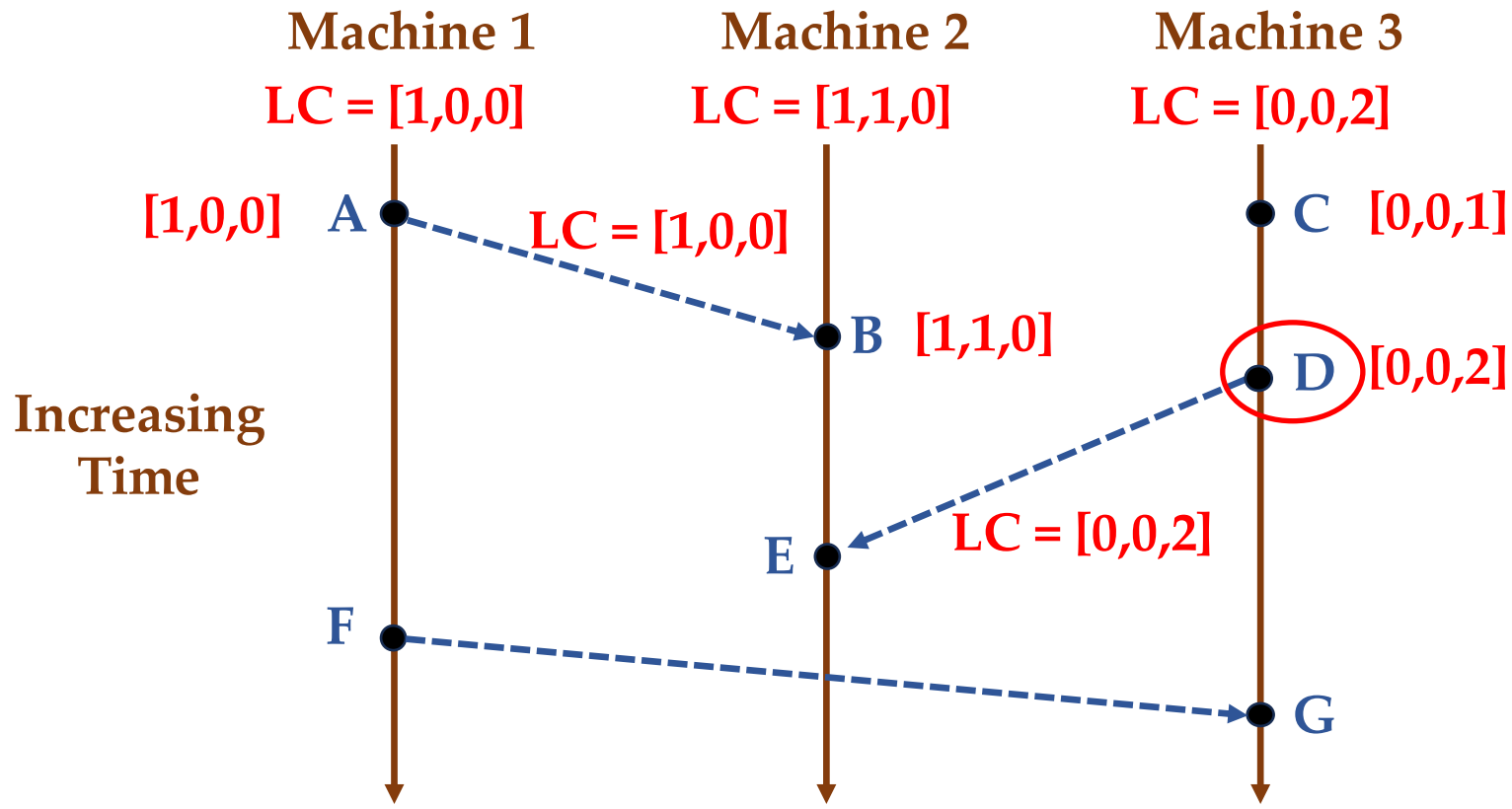
Let's look at Event D next.

Vector Clock Algorithm



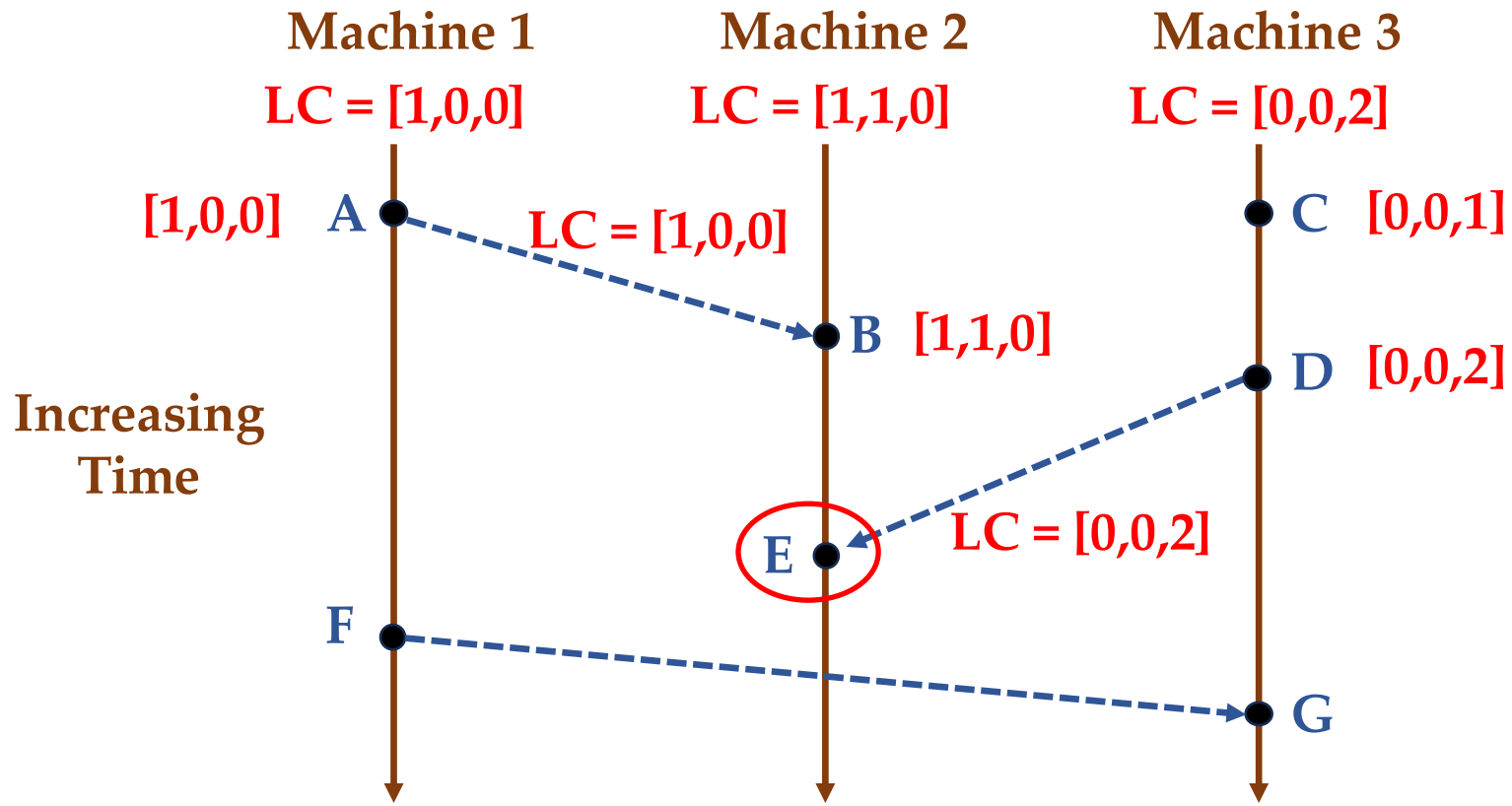
Let's look at Event D next.

Vector Clock Algorithm



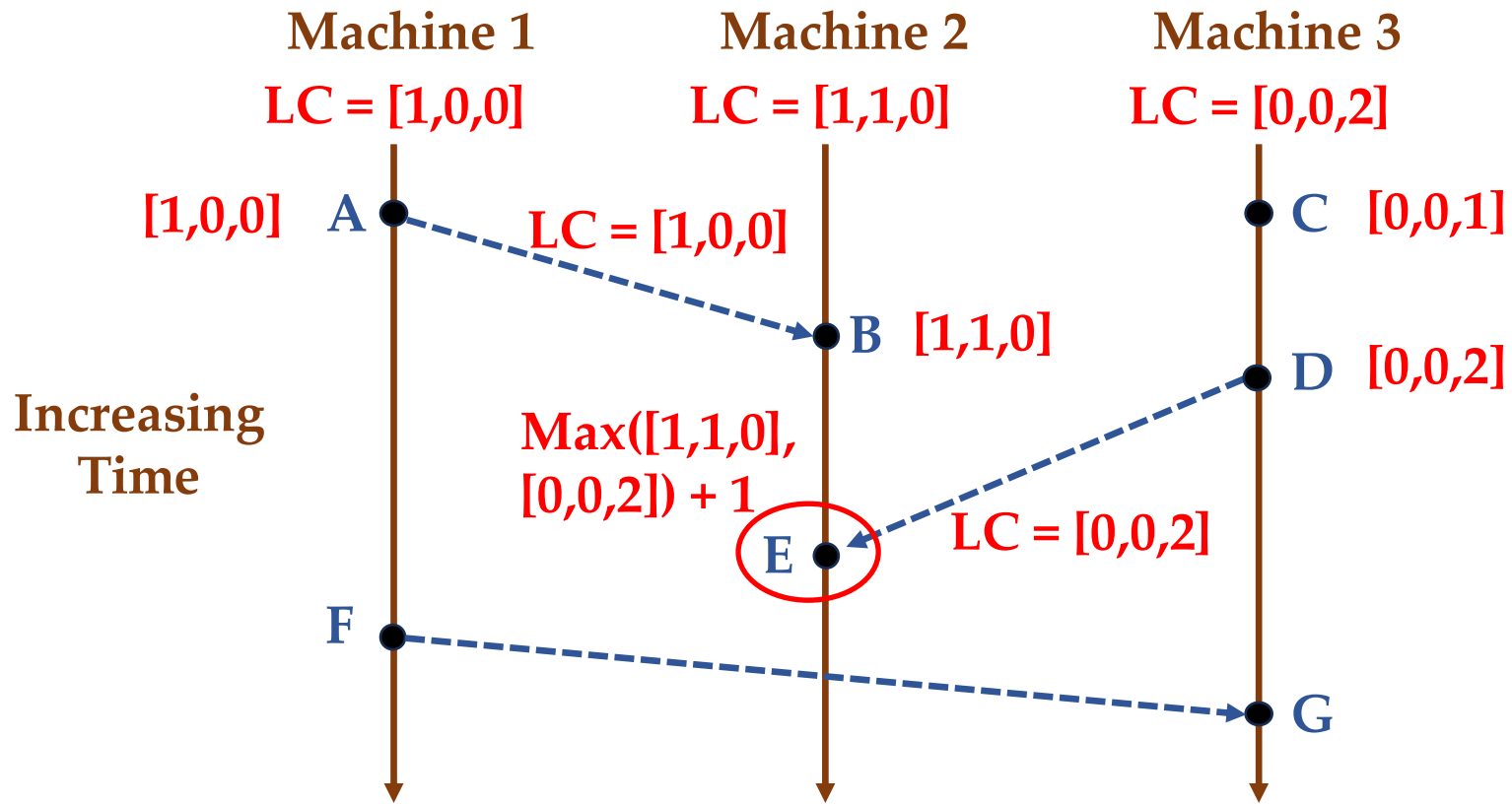
Let's look at Event D next.

Vector Clock Algorithm



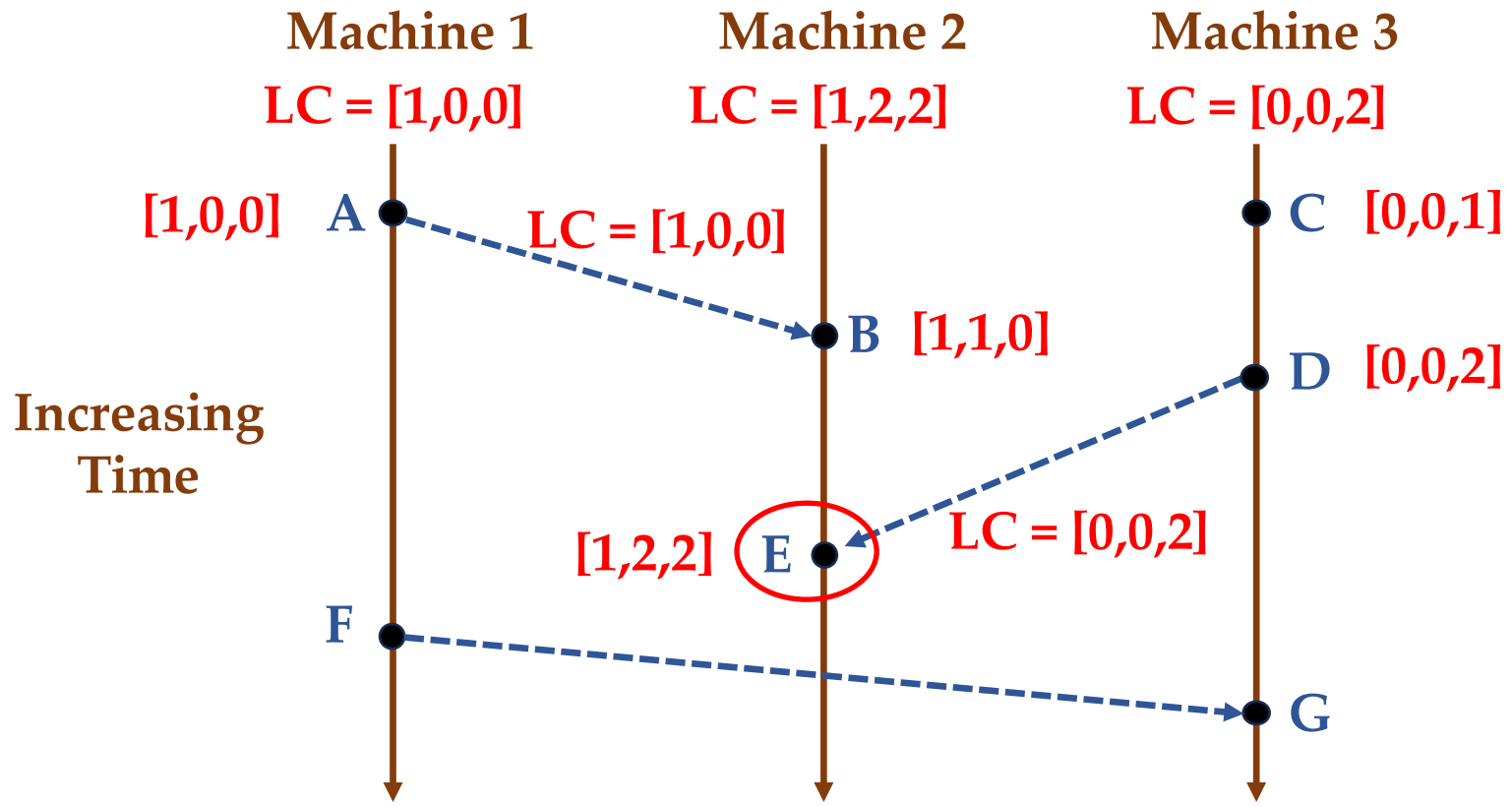
Let's look at Event E next.

Vector Clock Algorithm



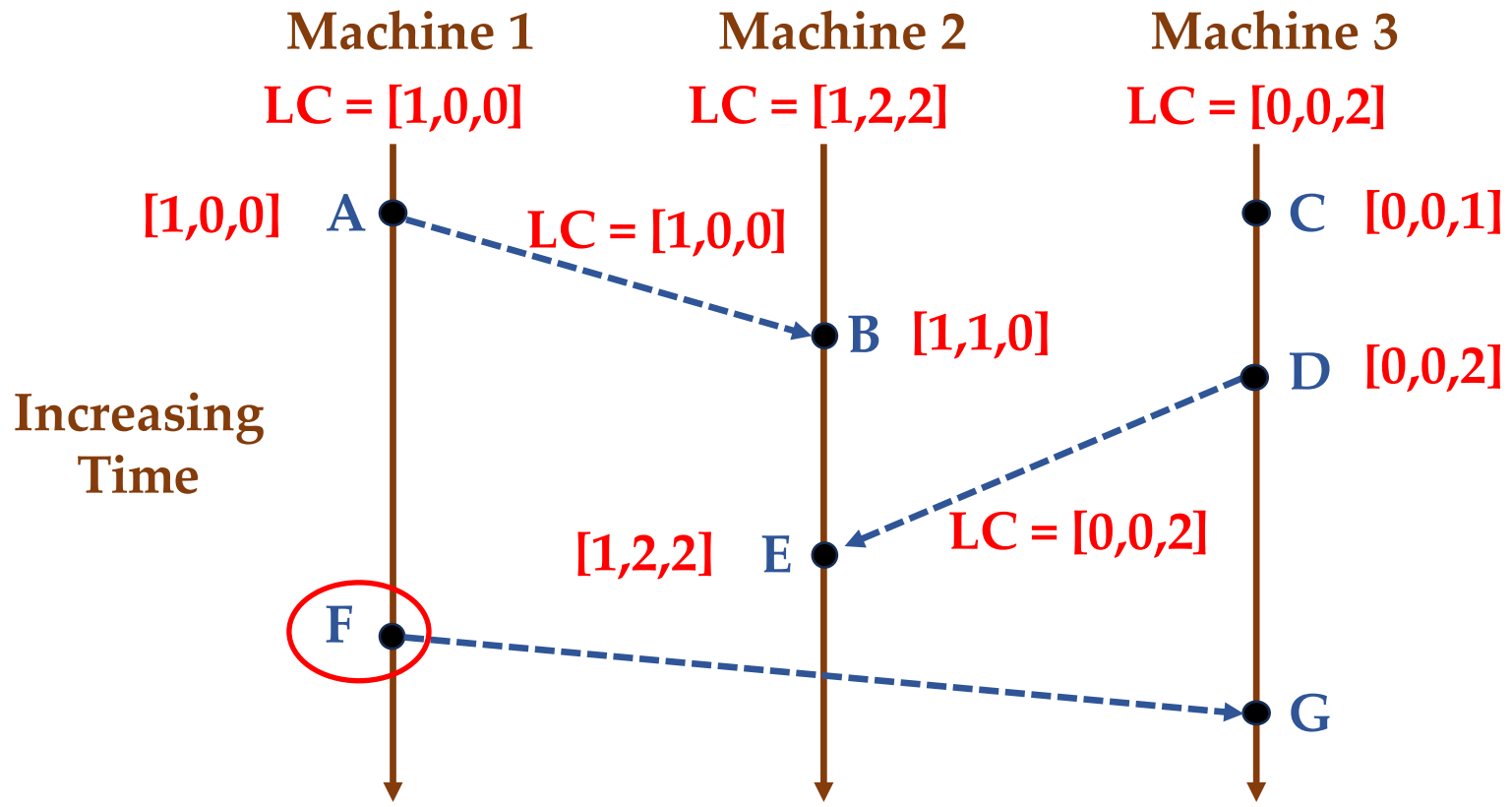
Let's look at Event E next.

Vector Clock Algorithm



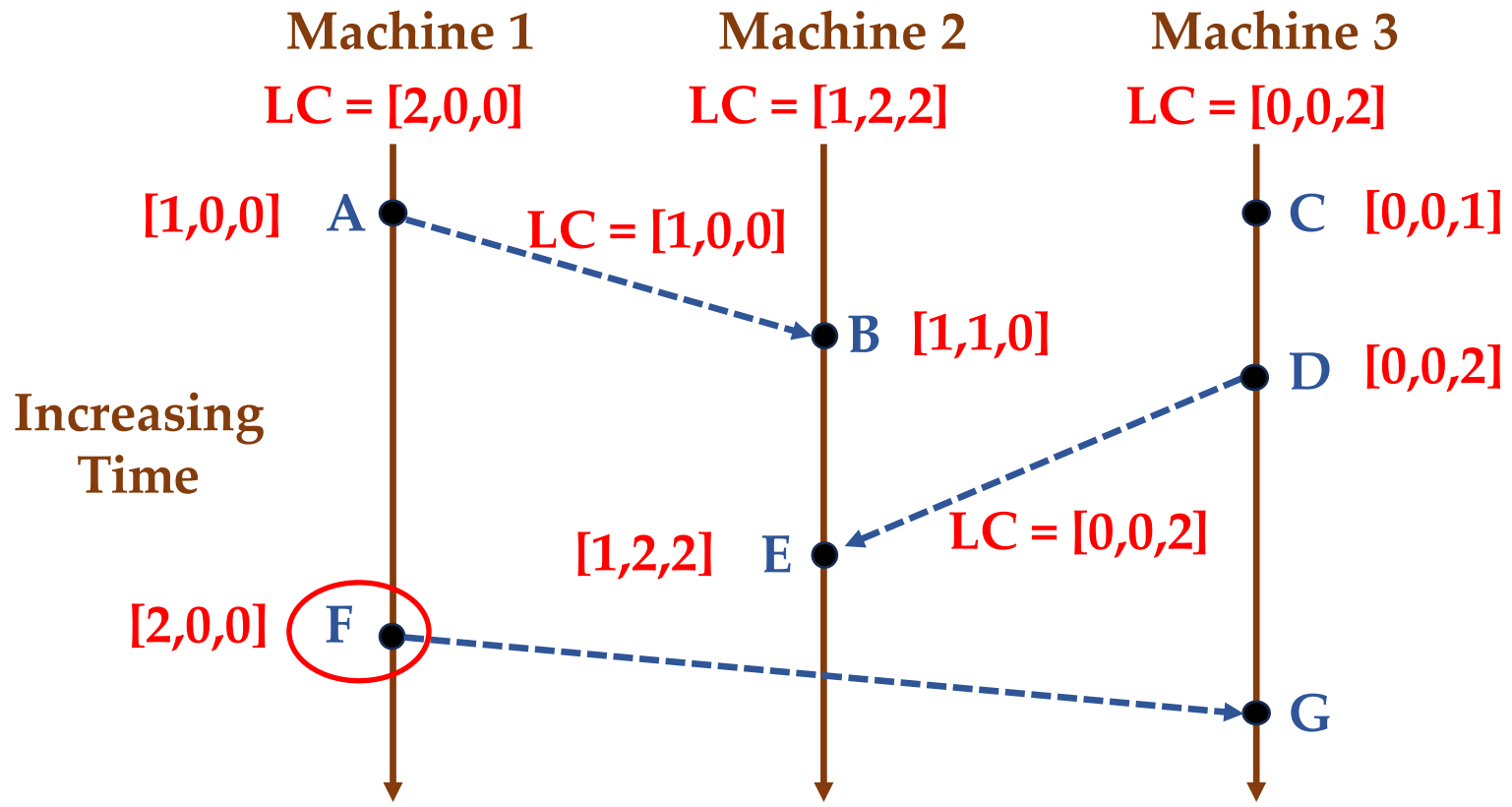
Let's look at Event E next.

Vector Clock Algorithm



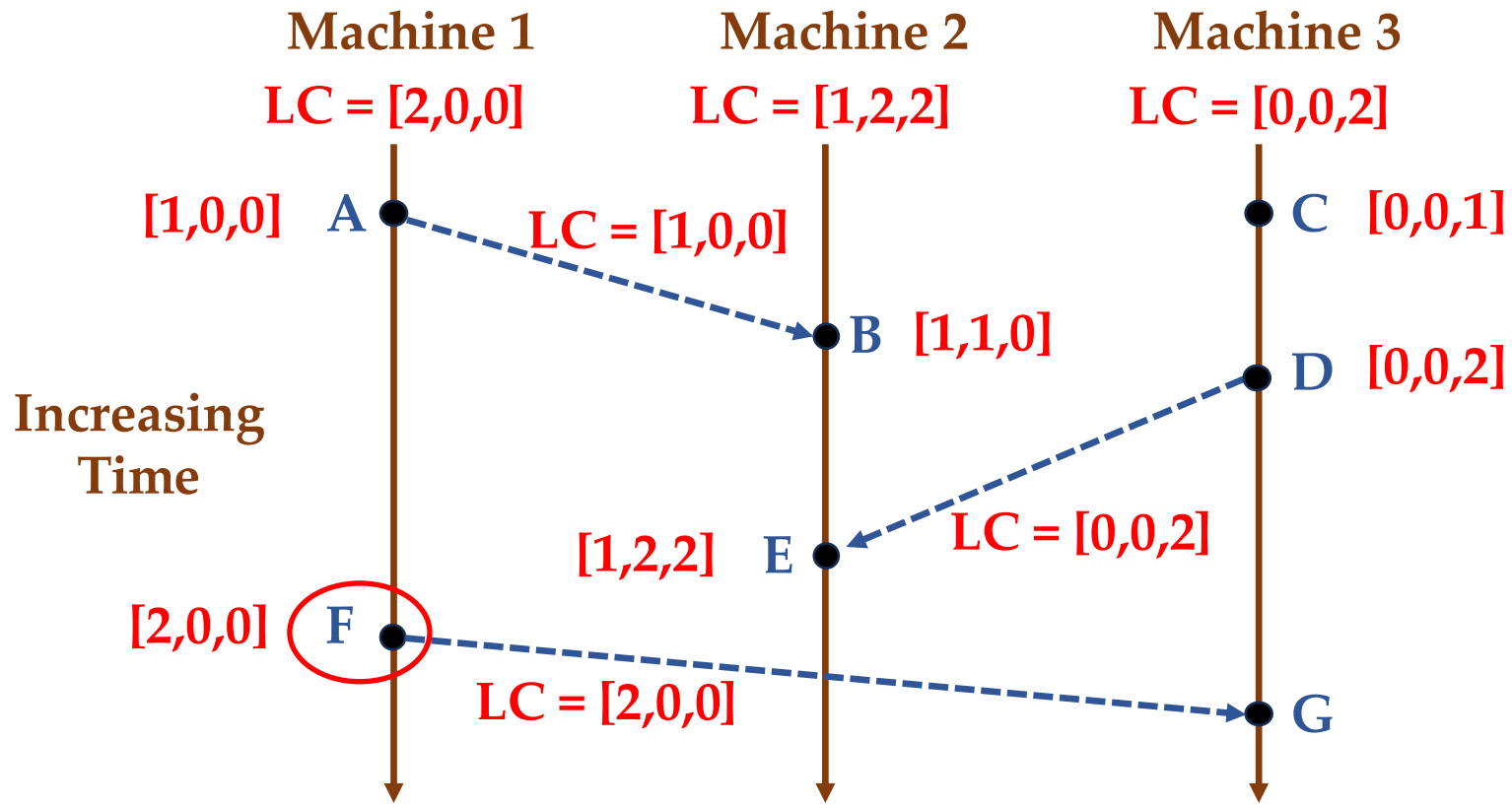
Let's look at Event F next.

Vector Clock Algorithm



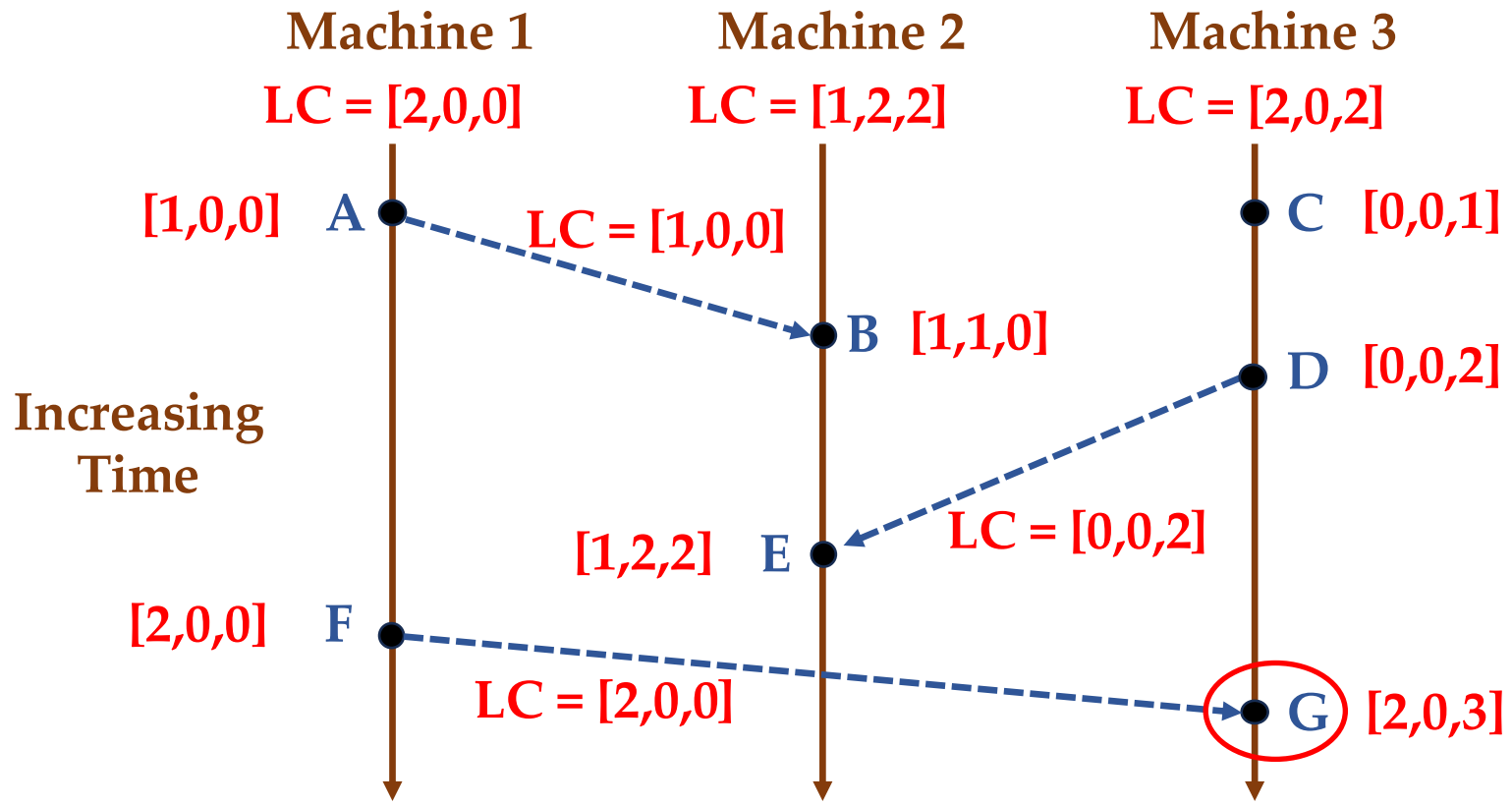
Let's look at Event F next.

Vector Clock Algorithm



Let's look at Event F next.

Vector Clock Algorithm



Let's look at Event G next.