Caltech, CMS. CS/IDS 142: Lecture 7.2
Global Snapshots
Mani Chandy
15 November 2019

Take away concepts from the course:

- State transition systems.
- Always nothing bad happens.
- Progress. The system eventually gets closer to its goal.
- Map global view to local agent state
- Data structures, e.g. graphs, that change with computation.

Global Snapshots

What can a process know about the system while the system is changing?

How can a process "know" anything about the state of the system while the system is changing?

Examples of applications of global snapshots:

•OS process must know: Has computation terminated? Are all processes idle and are all channels empty?

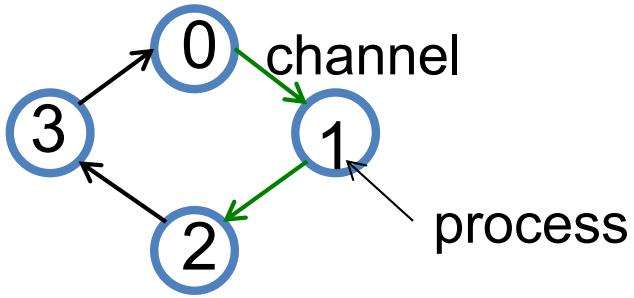
•In a database application: Is the system deadlocked? Is a cycle of transactions waiting for access to tables?

•See *Streaming Systems* book by Tyler Akidau, p. 310: Two reasons for Flink Streaming rise to prominence:

1. Beam programming model
2. Use of the global snapshot algorithm

Simple Example

Cycle of n processes indexed 0, 1, .., n-1.
Channel from process j to process j+1 mod n.
Tokens can be destroyed but not created.



Example with n = 4

Proposed algorithm.

•OS process 0 sends a *count* message containing the number of tokens that it holds.

•When OS process j receives the *count* message it sends the count message adding the number of tokens it holds to the count.

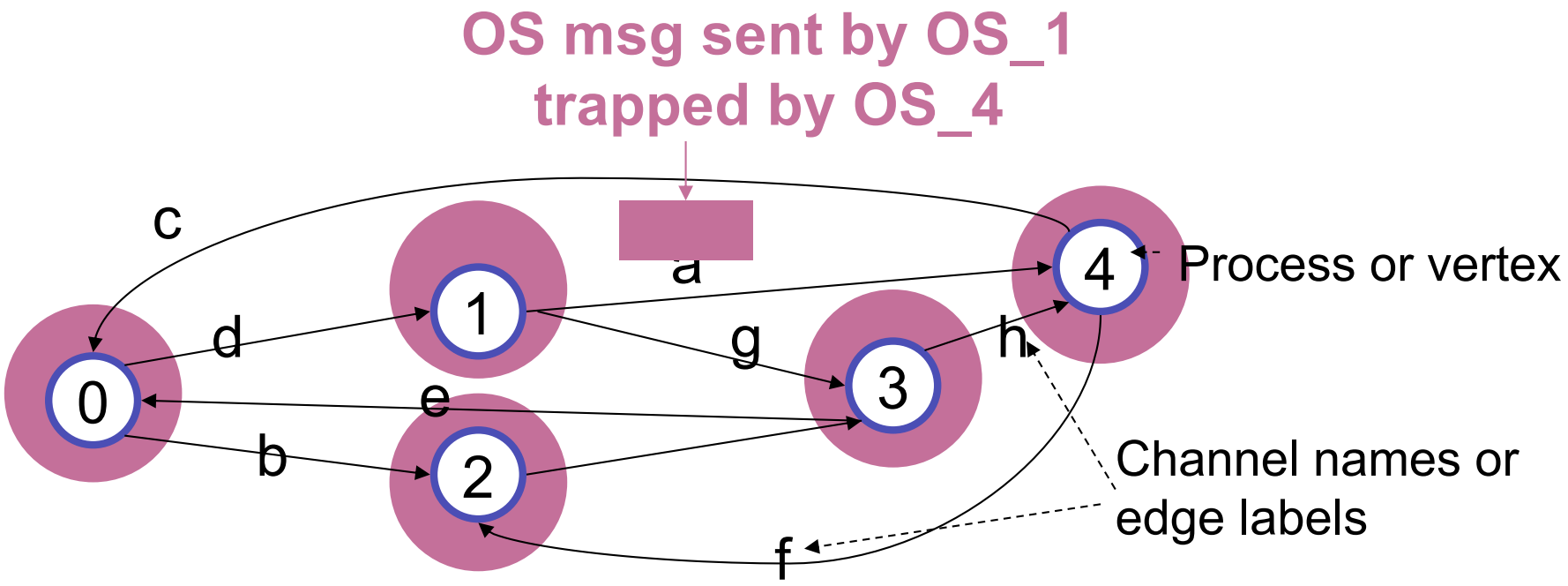
•When OS process 0 receives the *count* message, the count in the message is the number of tokens in the system.

Is the algorithm correct?

NO, because it doesn't count the tokens in transit along the channels.

Distributed Operating System – The Operational Model. *Superposition*

OS processes use same channels:
They can send OS messages that are trapped by receiving OS processes but not sent on to user processes.

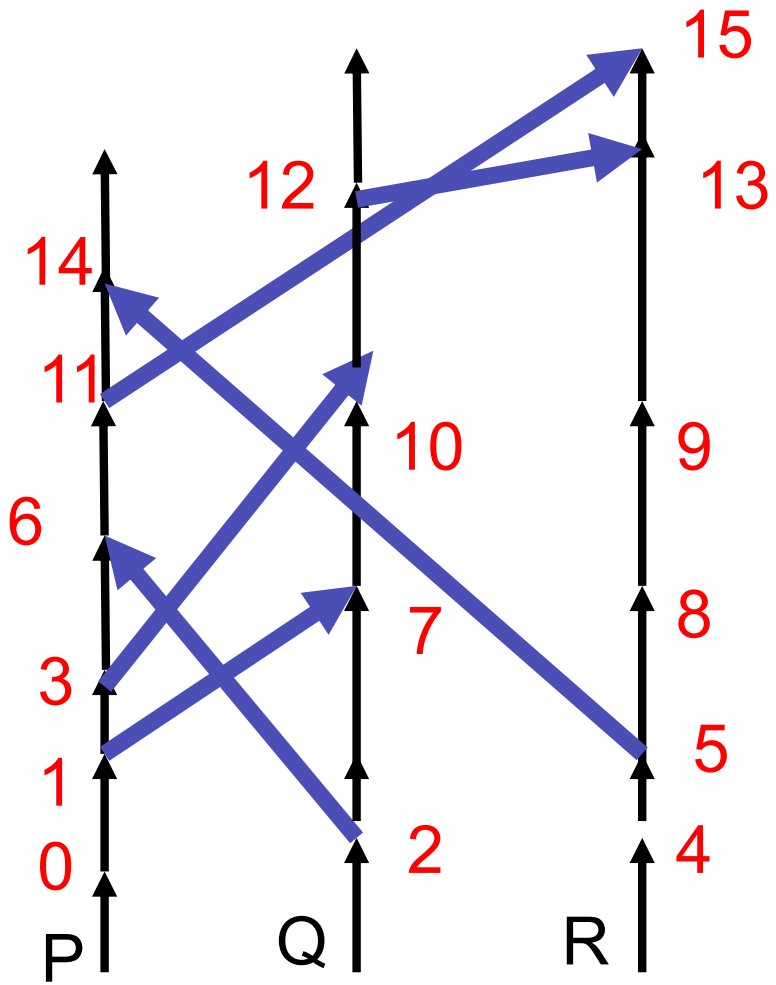


Given computations of each process (traces) what are possible system computations?

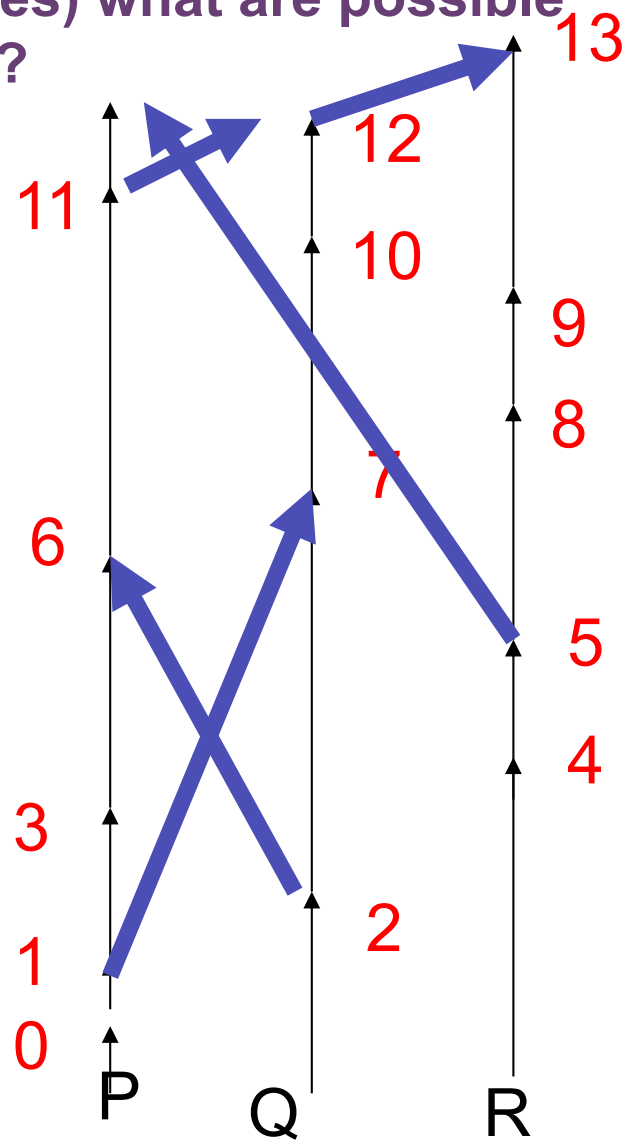
Each vertical line represents a computation of a process.

The blue lines going across vertical lines represent messages in channels.

The time-line graph represents a system computation



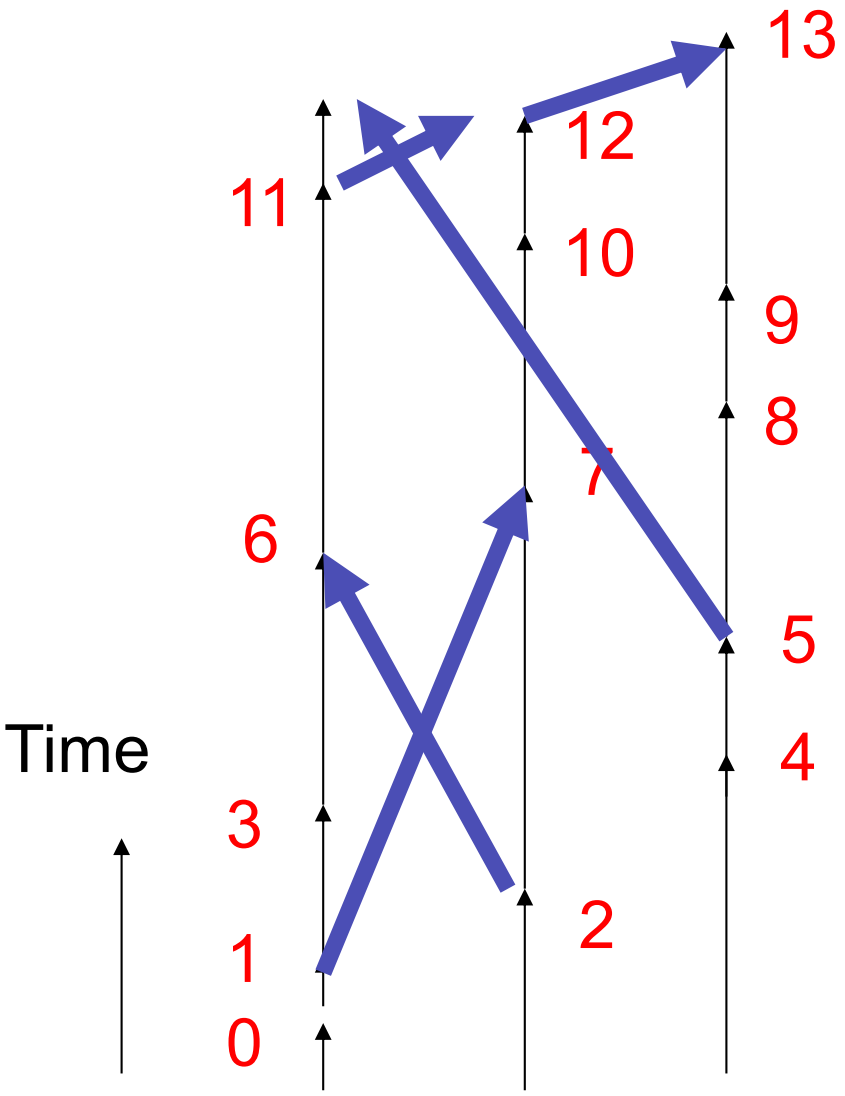
System computation



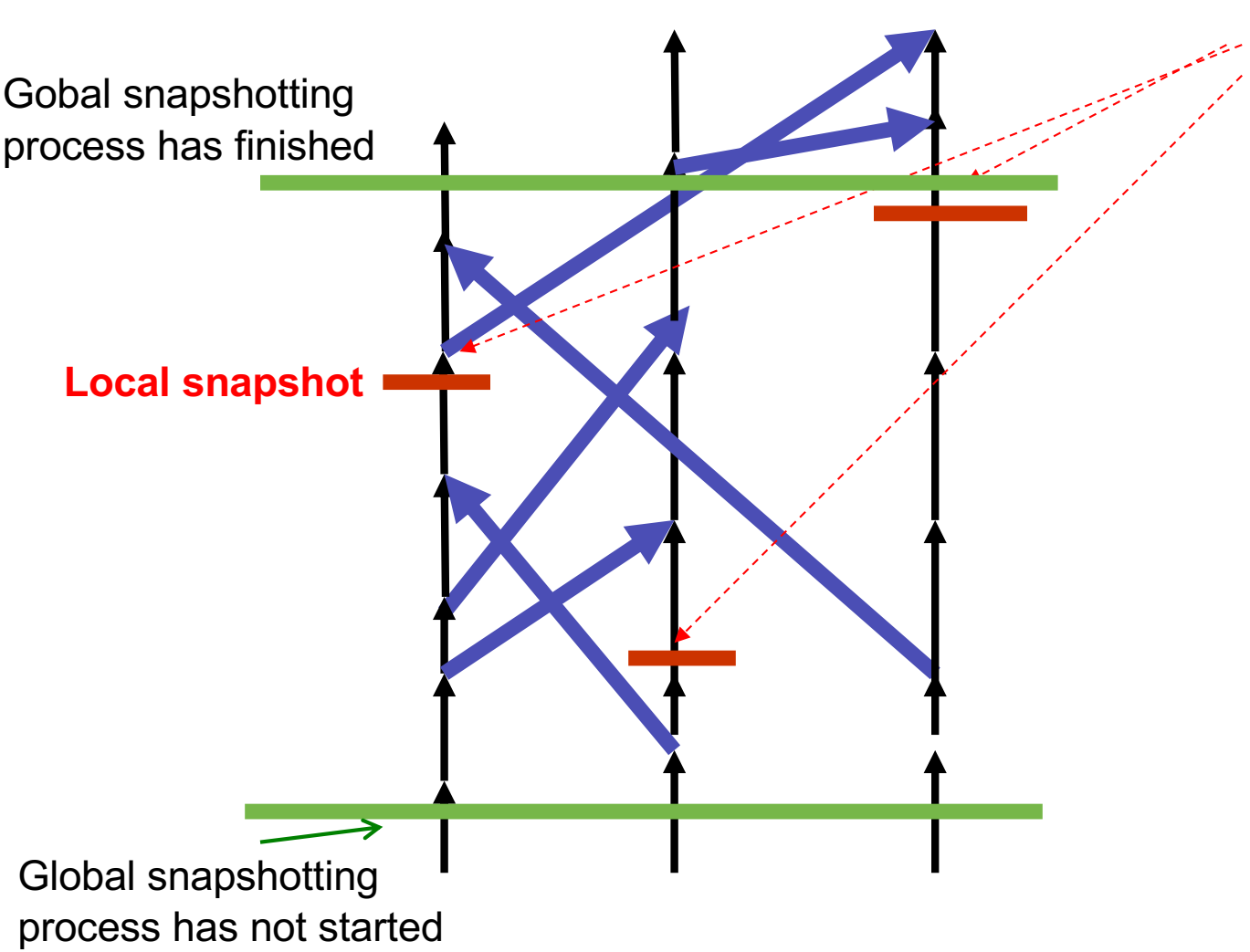
Another valid system computation

Theorem: Any total order consistent with the partial order,
i.e., any numbering of integers 0,1,2,3,
… in which all numbers are from lower-numbered to higher-numbered vertices,
is a valid system computation.

Proof:
•Process computations are valid.
•Channel computations are valid.



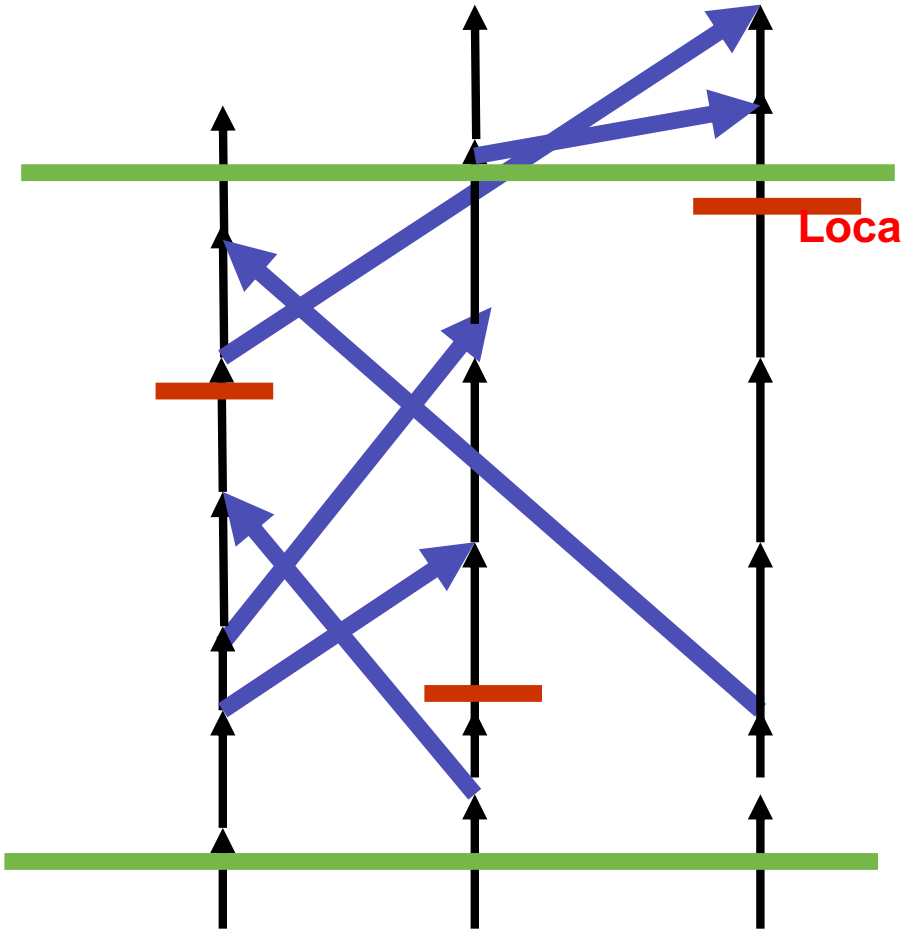
What local snapshots are valid?



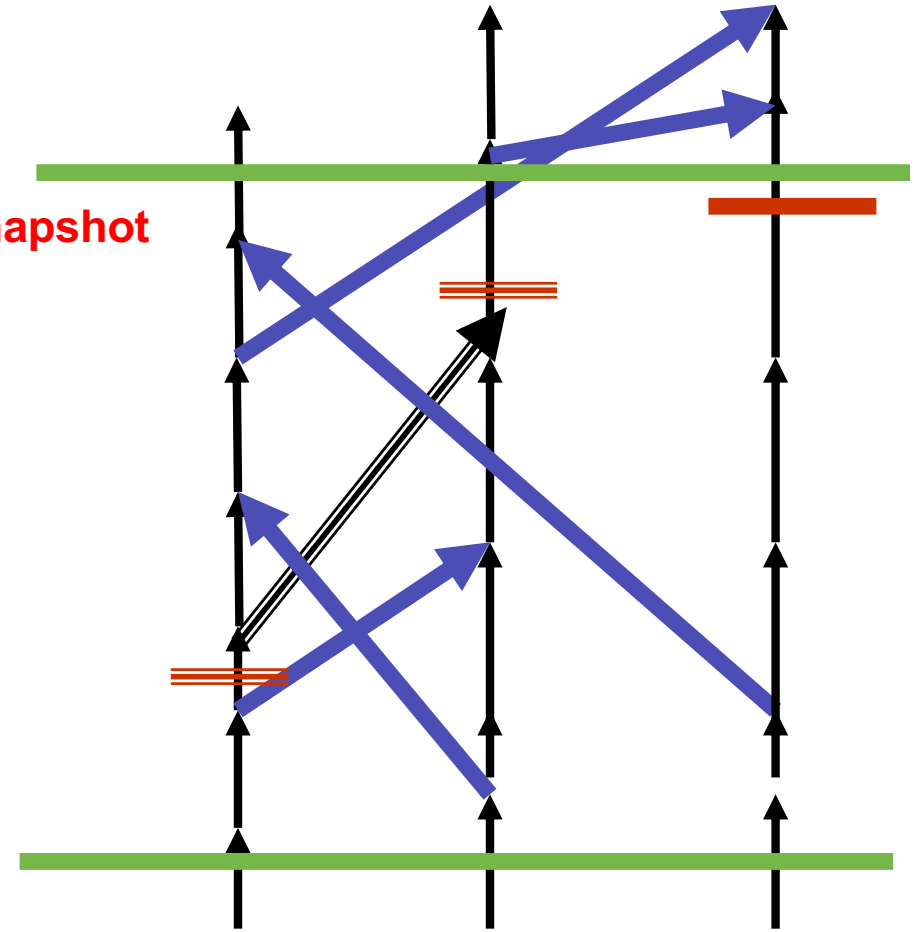
Processes take local snapshots at different times.

Is there a system computation in which these local snapshots are part of a global snapshot?

Is there a system computation which could have produced these local snapshots?



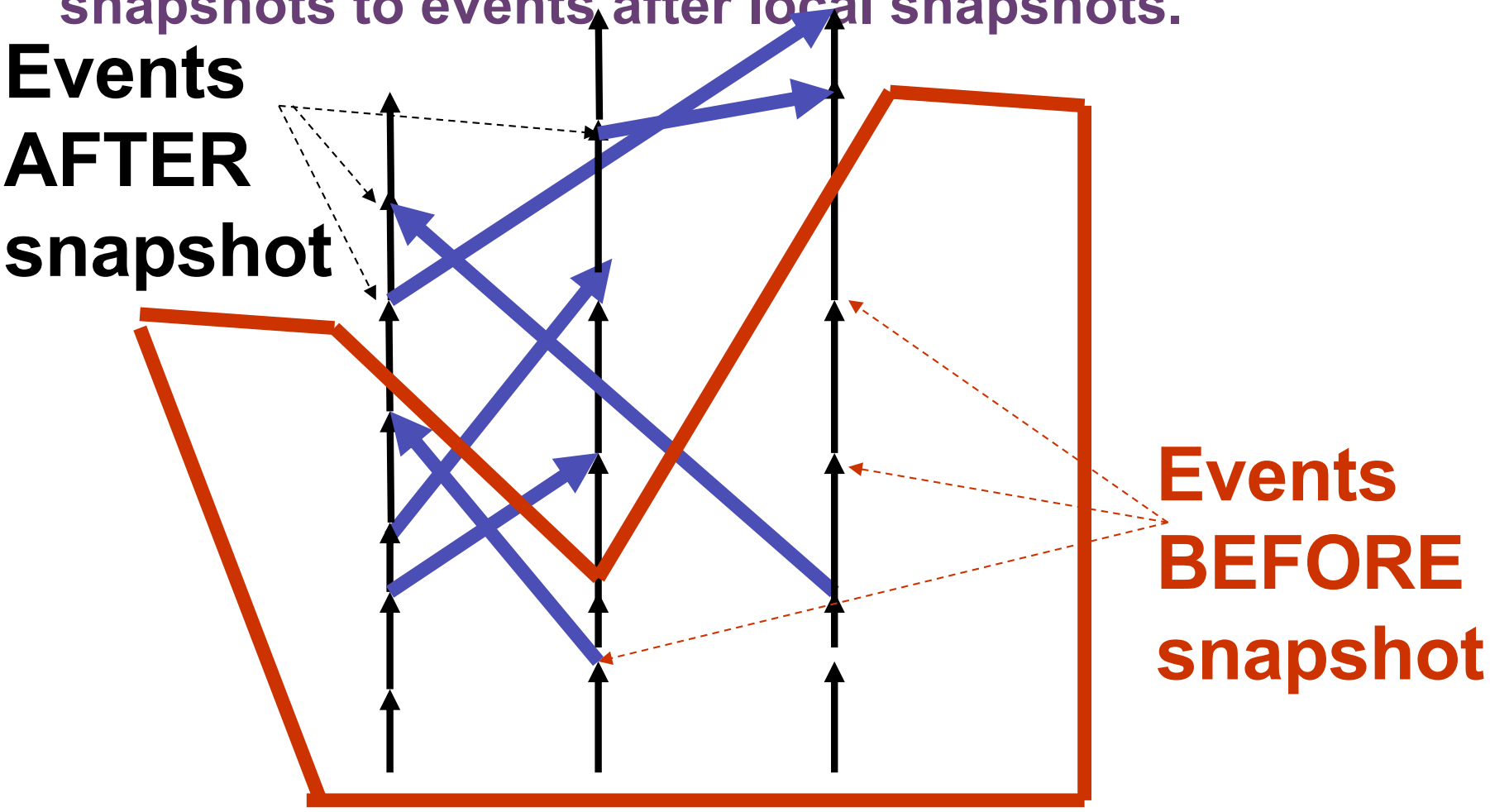
Local snapshot



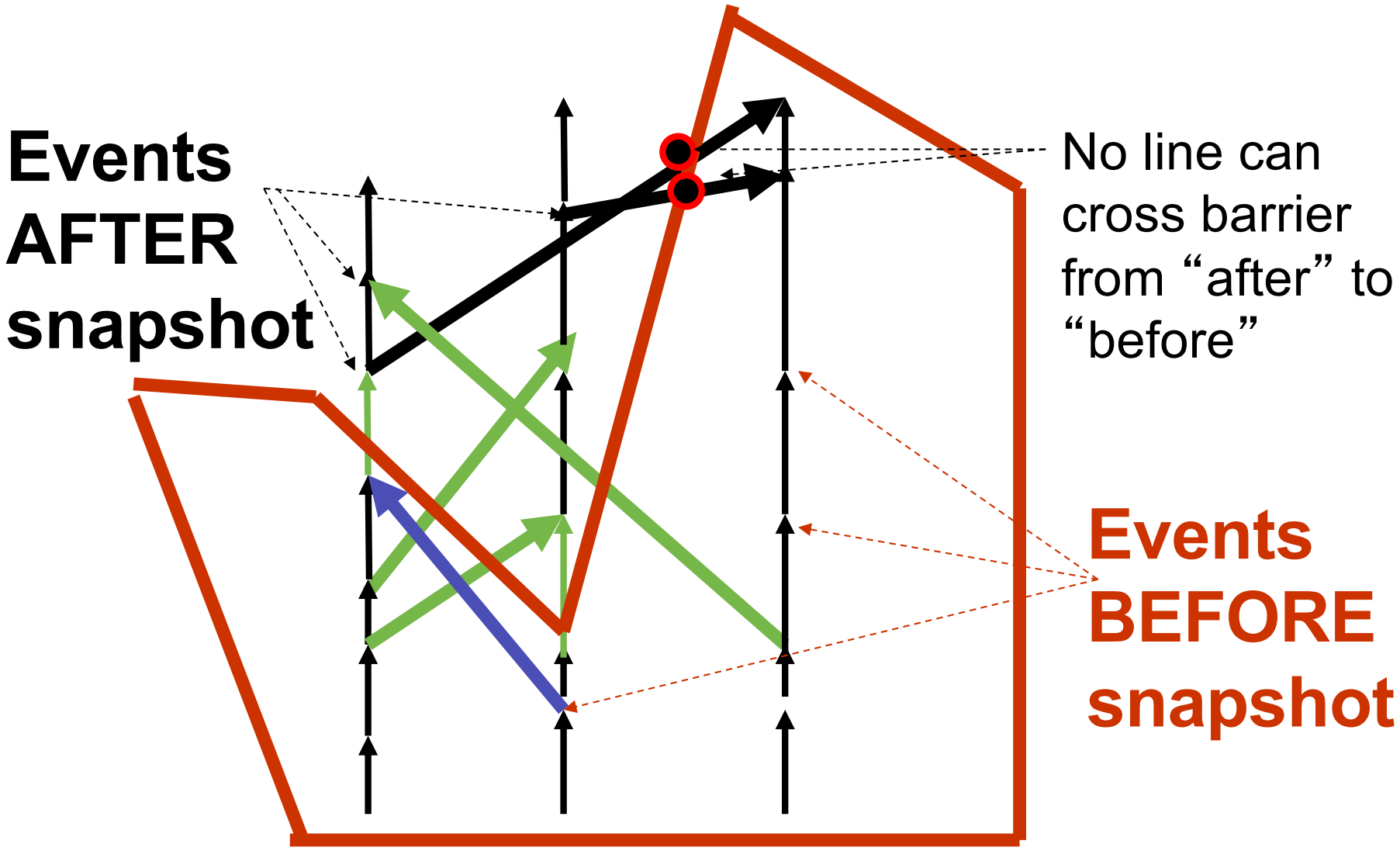
Valid? Yes because message received in a snapshot was sent in an earlier snapshot

Valid? No, because Q received a message that has not been sent!

Consistency of Local Snapshots: All edges in the time-line graph are directed from events before local snapshots to events after local snapshots.



Invalid local snapshots



Theorem

- Let c_numberSent be the number of messages sent by a process p along an outgoing channel c before p takes its (local) snapshot.
- Let c_numberReceived be the number of messages received by a process q along an incoming channel c before q takes its (local) snapshot.
- The local snapshots are consistent if and only if:

For all channels c: c_numberSent >= c.numberReceived.

- Proof: Follows immediately from basic theorem given earlier.

Global Snapshot Algorithm

1. When a process takes its snapshot, it sends a special message (let's call it a *marker* message) along each of its outgoing channels.
2. When a process receives a *marker* message, it takes its snapshot if it has not done so.
3. The state of a channel (p, q) from process p to process q in the global snapshot is the sequence of messages received by q after q takes its snapshot and before q receives the marker along channel (p, q).

Proof of correctness

Claim: Every message sent by a process P after P took its snapshot is received by a process Q after Q took its snapshot.

Proof:

- When P took its snapshot it sent a marker to Q.
- If Q hadn't already taken its snapshot when it received the marker, it takes its snapshot upon receiving the marker.
- All messages sent by P to Q after P takes its snapshot are sent after P sends the marker to Q.