Synchronous Algorithms I

Barrier Synchronizations and Computing LBTS

Richard M. Fujimoto
Professor

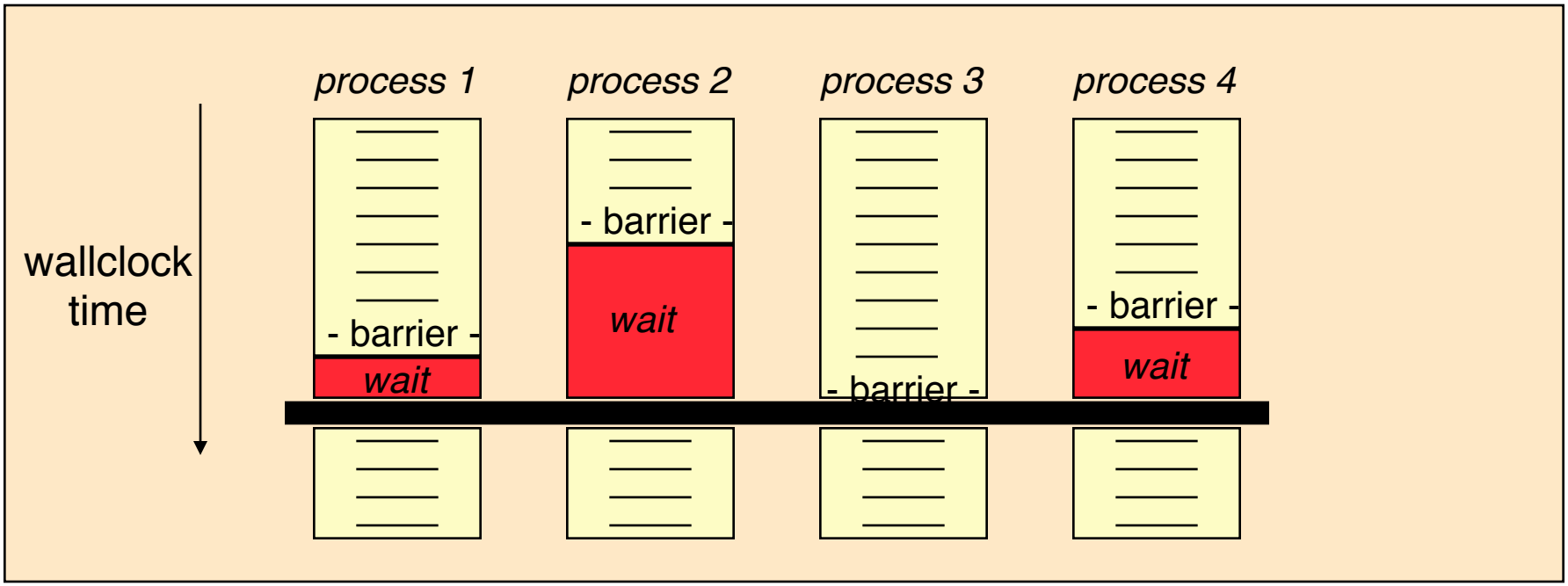
Computational Science and Engineering Division
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0765, USA

http://www.cc.gatech.edu/~fujimoto/

Outline

- Barrier synchronizations and a simple synchronous algorithm
- Implementation of Barrier mechanisms
  - Centralized Barriers
  - Tree Barrier
  - Butterfly Barrier
- Computing LBTS

Barrier Synchronization



Barrier Synchronization: when a process invokes the barrier primitive, it will block until all other processes have also invoked the barrier primitive.

When the last process invokes the barrier, all processes can resume execution

Synchronous Execution

Recall the goal is to ensure each LP processes events in time stamp order

Basic idea: each process cycles through the following steps:

- Determine the events that are safe to process
  - Compute a Lower Bound on the Time Stamp ($LBTS_i$) of events that LP_i might later receive
  - Events with time stamp ≤ LBTS are safe to process
- Process safe events, exchange messages
- Global synchronization (barrier)

Messages generated in one cycle are not eligible for processing until the next cycle

A Simple Synchronous Algorithm

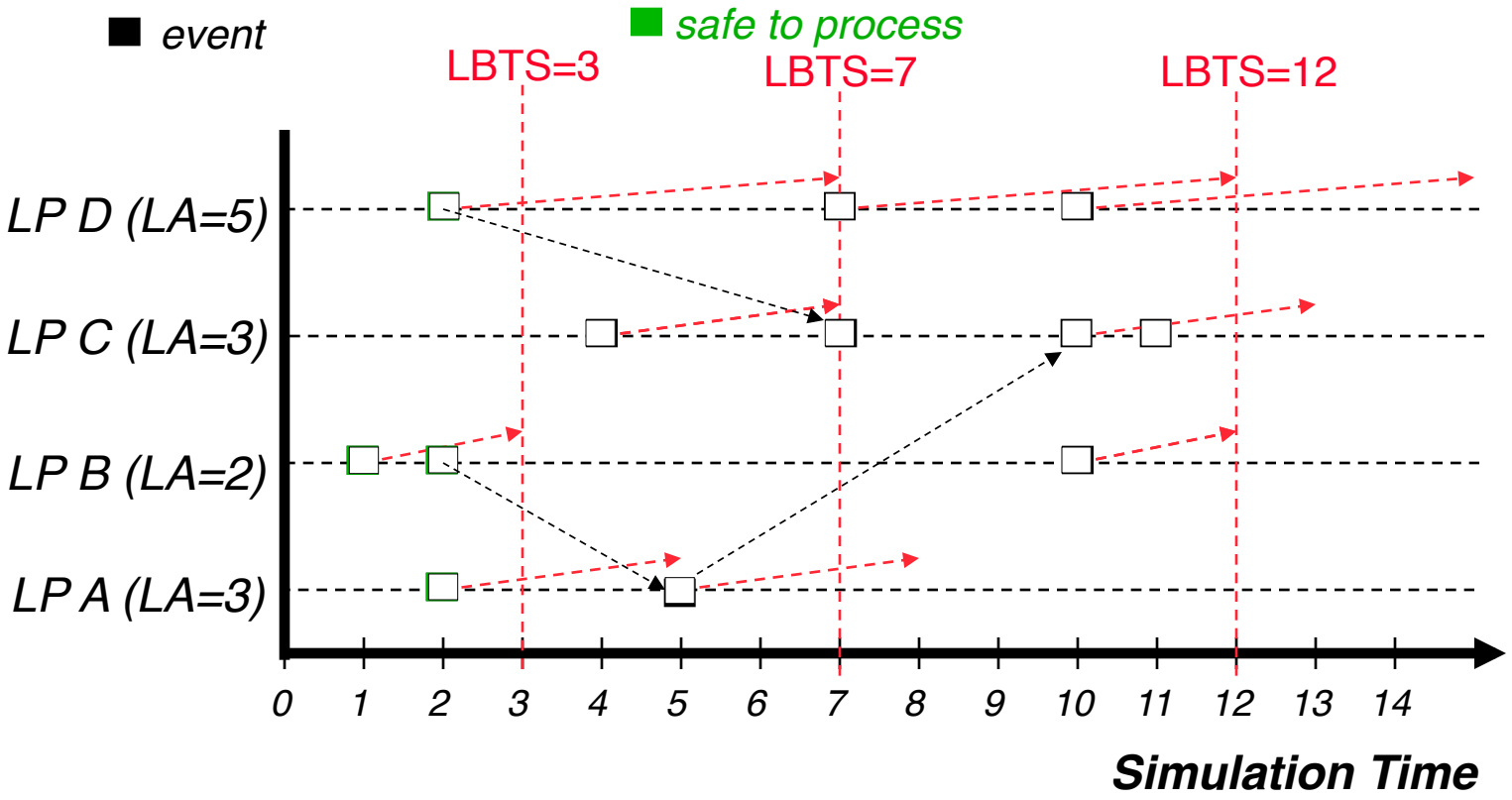
- Assume any LP can communicate with any other LP
- Assume instantaneous message transmission
  - "transient message problem" to be discussed later
- N_i = time of next event in LP_i
- LA_i = lookahead of LP_i

WHILE (unprocessed events remain)
receive messages generated in previous iteration
LBTS = min (N_i + LA_i)
process events with time stamp ≤ LBTS
barrier synchronization
endDO

$LBTS_i$ = a lower bound on the time stamp of messages LP_i might receive in the future

If $LBTS_i$ is the same for all LPs, it is simply called LBTS

Synchronous Execution Example



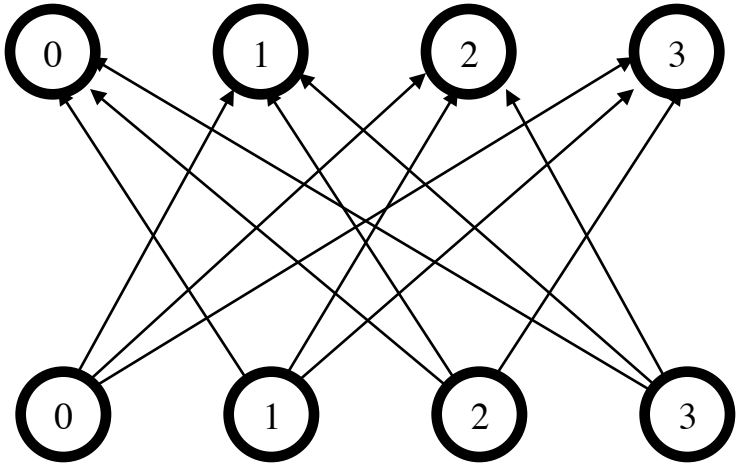
Issues

- Implementing the barrier mechanism
- Computing LBTS (global minimum)
- Transient messages (discussed later)

Barrier Using a Centralized Controller

- Central controller process used to implement barrier
- Overall, a two step process
  - Controller determines when barrier reached
  - Broadcast message to release processes from the barrier
- Barrier primitive for non-controller processes:
  - Send a message to central controller
  - Wait for a reply
- Barrier primitive for controller process
  - Receive barrier messages from other processes
  - When a message is received from each process, broadcast message to release barrier
- Performance
  - Controller must send and receive N-1 messages
  - Potential bottleneck

Broadcast Barrier

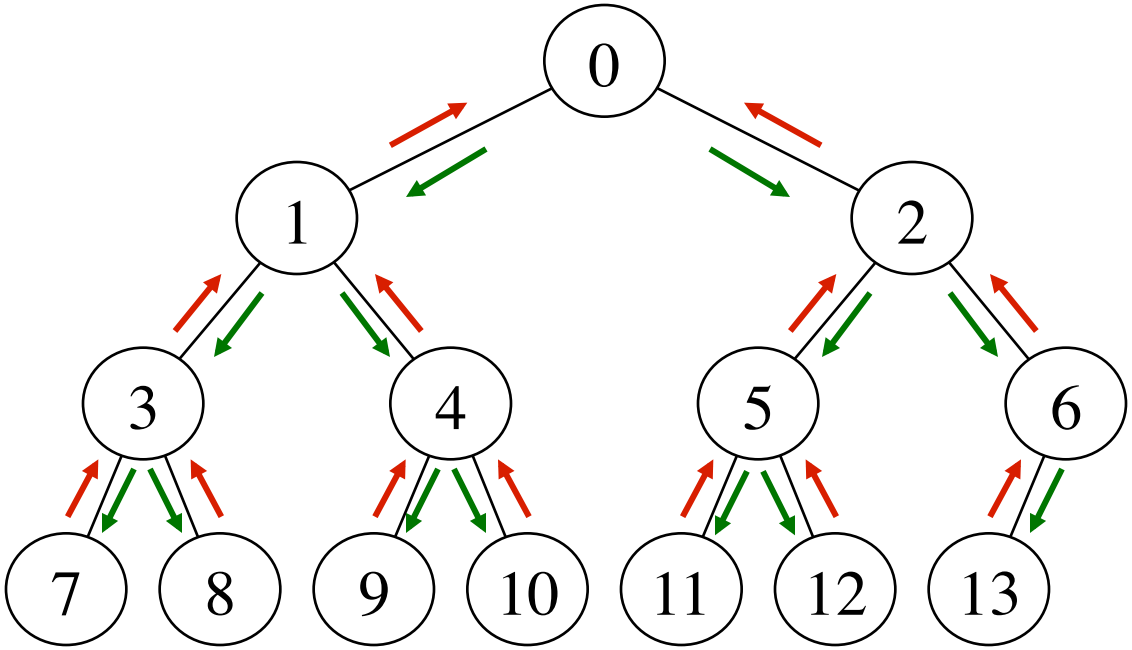


One step approach; no central controller

Each process:

- Broadcast message when barrier primitive is invoked
- Wait until a message is received from each other process
- N (N-1) messages

Tree Barrier

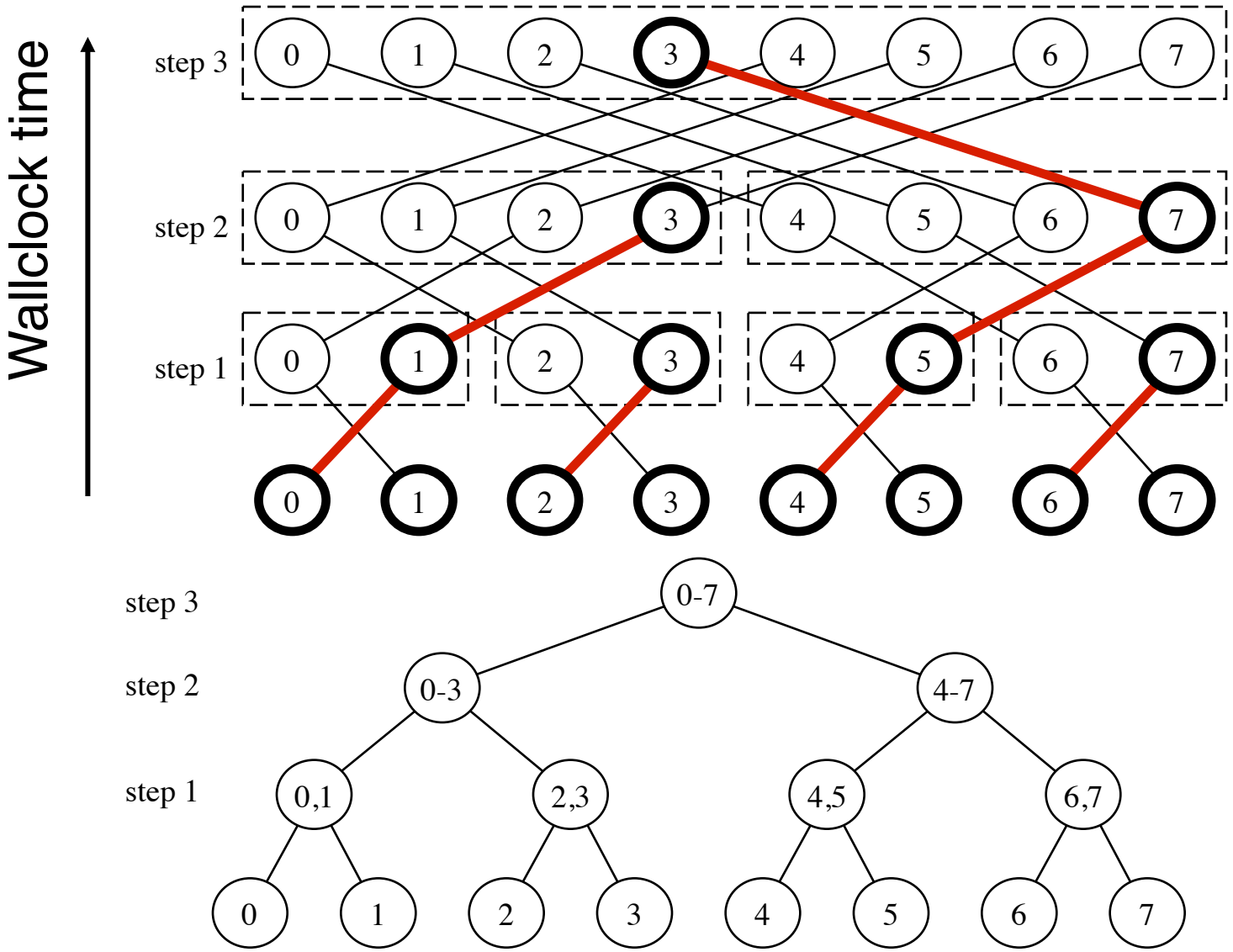


- Organize processes into a tree
- A process sends a message to its parent process when
  - The process has reached the barrier point, and
  - A message has been received from each of its children processes
- Root detects completion of barrier, broadcast message to release processes (e.g., send messages down tree)
- 2 log N time if all processes reach barrier at same time

Butterfly Barrier

- N processes (here, assume N is a power of 2)
- Sequence of $\log_2$ N pairwise barriers (let k = $\log_2$ N)
- Pairwise barrier:
  - Send message to partner process
  - Wait until message is received from that process
- Process p: $b_k b_{k-1} \ldots b_1$ = binary representation of p
- Step i: perform barrier with process $b_k \ldots b_i' \ldots b_1$ (complement ith bit of the binary representation)
- Example: Process 3 (011)
  - Step 1: pairwise barrier with process 2 (010)
  - Step 2: pairwise barrier with process 1 (001)
  - Step 3: pairwise barrier with process 7 (111)

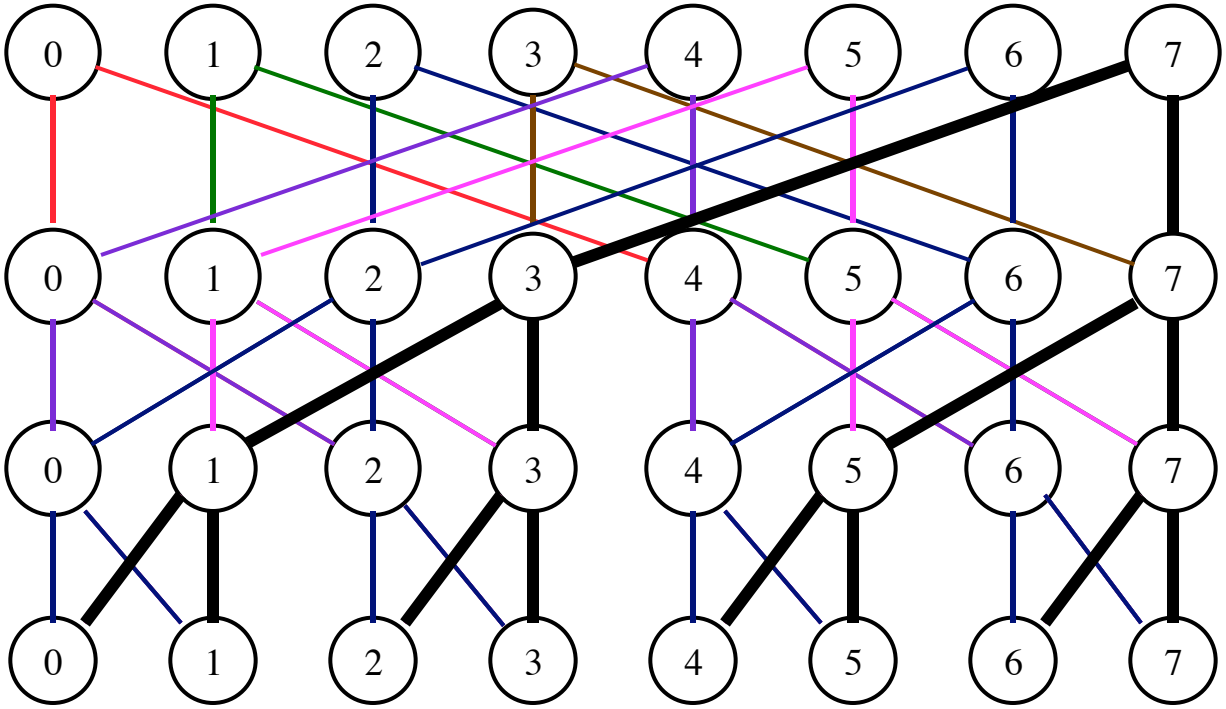
Butterfly Barrier Example



The communication pattern forms a tree from the perspective of any process

Butterfly: Superimpose Trees

An N node butterfly can be viewed as N trees superimposed over each other



After $\log_2$ N steps each process is notified that the barrier operation has completed

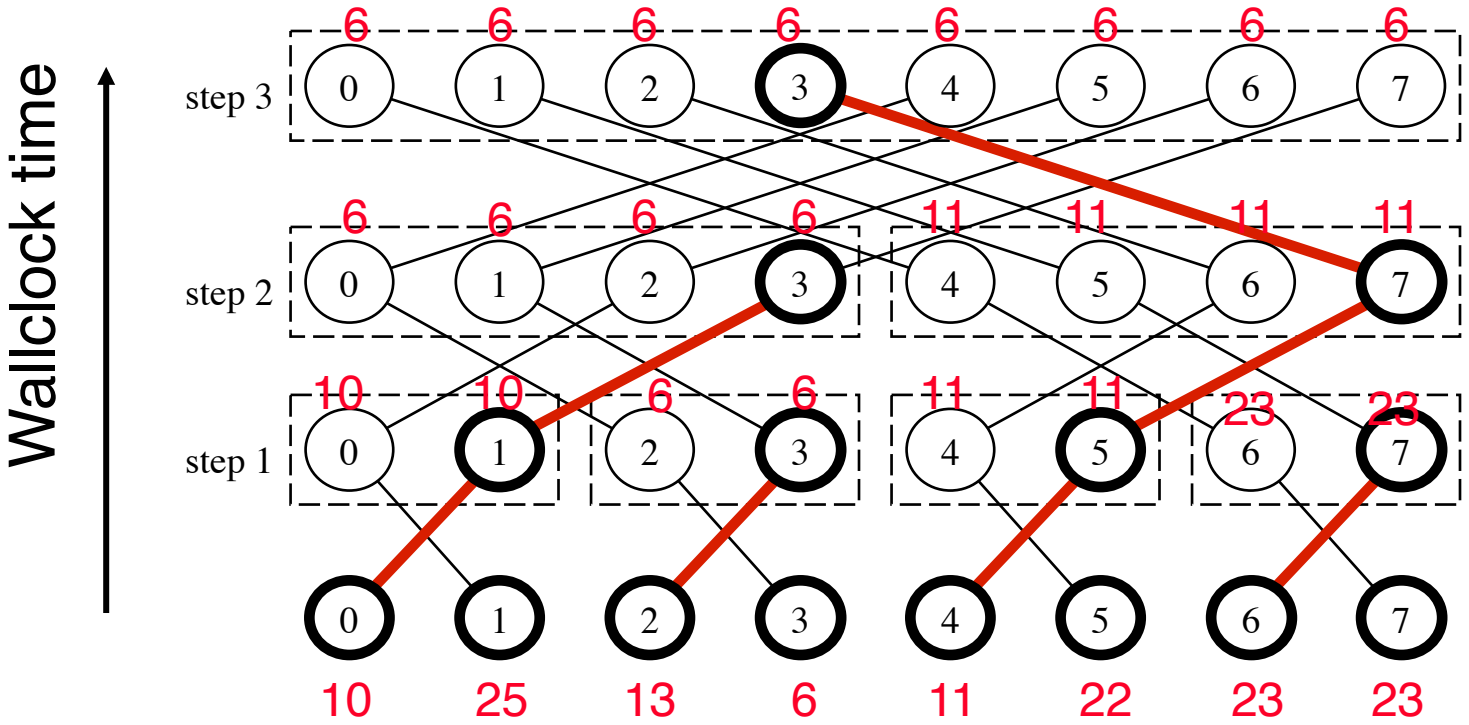
Outline

- Barrier synchronizations and a simple synchronous algorithm
- Implementation of Barrier mechanisms
  - Centralized Barriers
  - Tree Barrier
  - Butterfly Barrier
- Computing LBTS

Computing LBTS

It is trivial to extend any of these barrier algorithms to also compute a global minimum (LBTS)

- Piggyback local time value on each barrier message
- Compute new minimum among local value, incoming message(s)
- Transmit new minimum in next step



After log N steps, (1) LBTS has been computed, (2) each process has the LBTS value

Summary

- Synchronous algorithms use a global barrier to ensure events are processed in time stamp order
  - Requires computation of a Lower Bound on Time Stamp (LBTS) on messages each LP might later receive
- There are several ways to implement barriers
  - Central controller
  - Broadcast
  - Tree
  - Butterfly
- The LBTS computation can be “piggybacked” onto the barrier synchronization algorithm