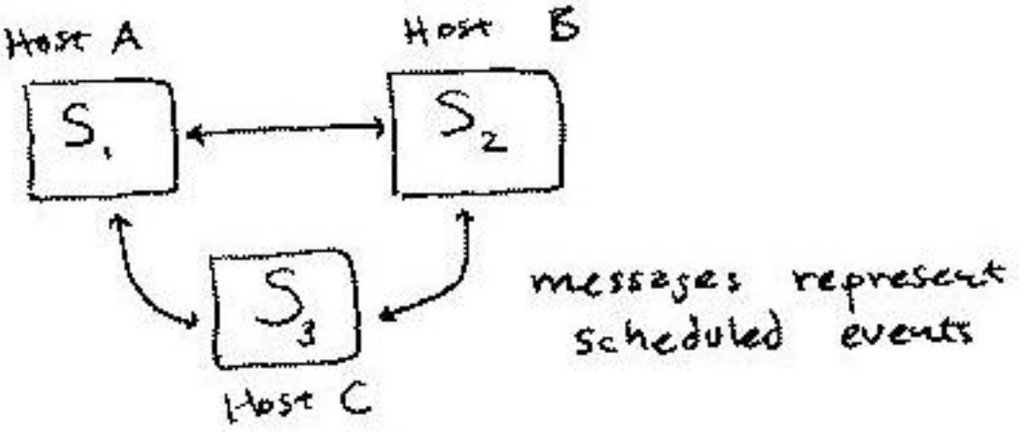
What if simulation is too large for one machine? Need distributed (networked) simulation.

Idea: Split up the simulation into a number of simulators (partition simulated objects)



Observe that we have two choices for implementing a visualizer in a networked simulation:

"pull" - visualizer asks each object for values of state variables — query & response (two messages total per object)

"push" - at certain intervals, each object sends a message containing values of state variables to the visualizer

Distributed Simulation - first attempt

We can classify simulated objects (with respect to one local simulator) as either

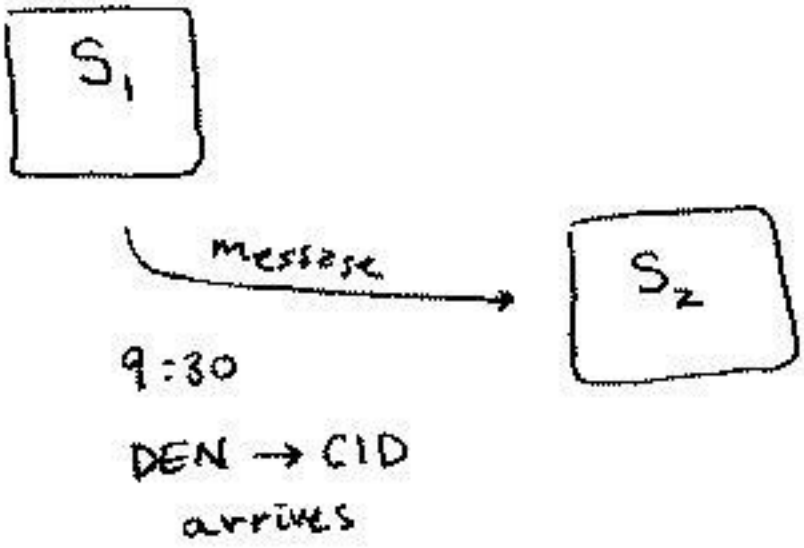
- local = an object that is simulated on the local host (machine)
- remote = an object simulated somewhere else (by another simulator)

Idea: Use the same simulation cycle as before (e.g fire earliest scheduled event in list), but whenever there is an action to schedule an event for a remote object, send a message to the simulator responsible for that object.

When a simulator receives an event message, it adds the event to the scheduled list (each event has a time stamp).

Bug:

Suppose we are simulating an airline, with flights & cities



Here's what can happen at S_2

1. Fire 9 am Arr ORD → CID
2. Fire 10 am Dep CID → MSP
3. Fire 9.30 am Arr DEN → CID

← message arrives to S_2

Notice - time went backward!

can cause problems (example: passengers simulated to connect via CID from DEN to MSP)

Processing events out of order is called a causality error.

The Synchronization Problem:

How to ensure that events are fired in timestamp order?

Theorem: A distributed simulation can only be correct if each component simulator processes events in increasing order.

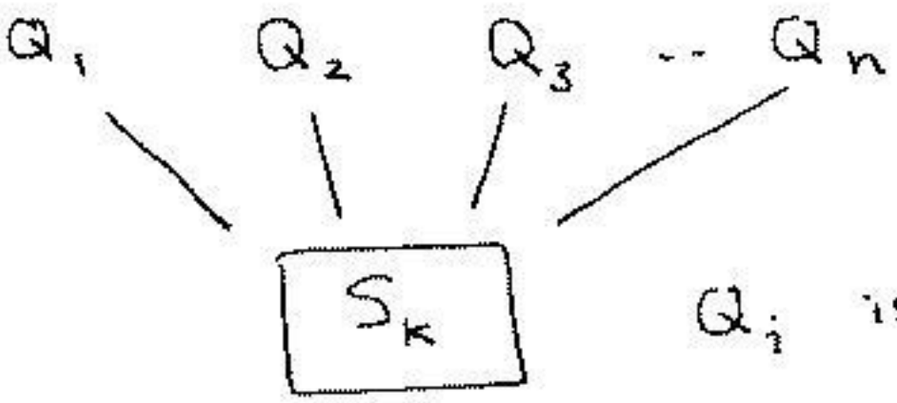
Proof idea: demonstrate that result of simulation could have been obtained by a non-distributed simulation (on a single host).

Solution Ideas

1. A central "timestamp" server could coordinate schedules of all simulators
   - But this adds overhead
   - It isn't "distributed"

2. Queue Invariants.

   Let every scheduled event be a message (many will be to loopback); maintain queues of incoming messages



Q_i is the queue of incoming messages sent from simulator S_i to S_K

Invariant 1

Queues shall be FIFO with timestamps in non decreasing order.

(we need to modify simulators to ensure this is true)

Invariant 2

No simulator fires an event until <u>all</u> its queues are non-empty — then it fires event with minimum timestamp.

Theorem:

A simulation satisfying these two invariants never decreases the simtime at any simulator

(Proof by induction)

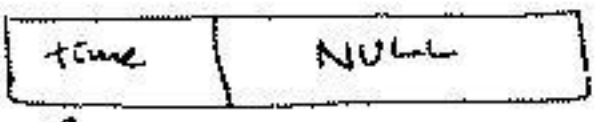
Deadlock

occurs if Q_j at S_i is empty,
Q_i at S_j is empty,
and no messages are "in transit"

How to overcome?

Null Events

From time to time, each simulator S_K sends (to all simulators) a specail "null" event



current simtime of S_K