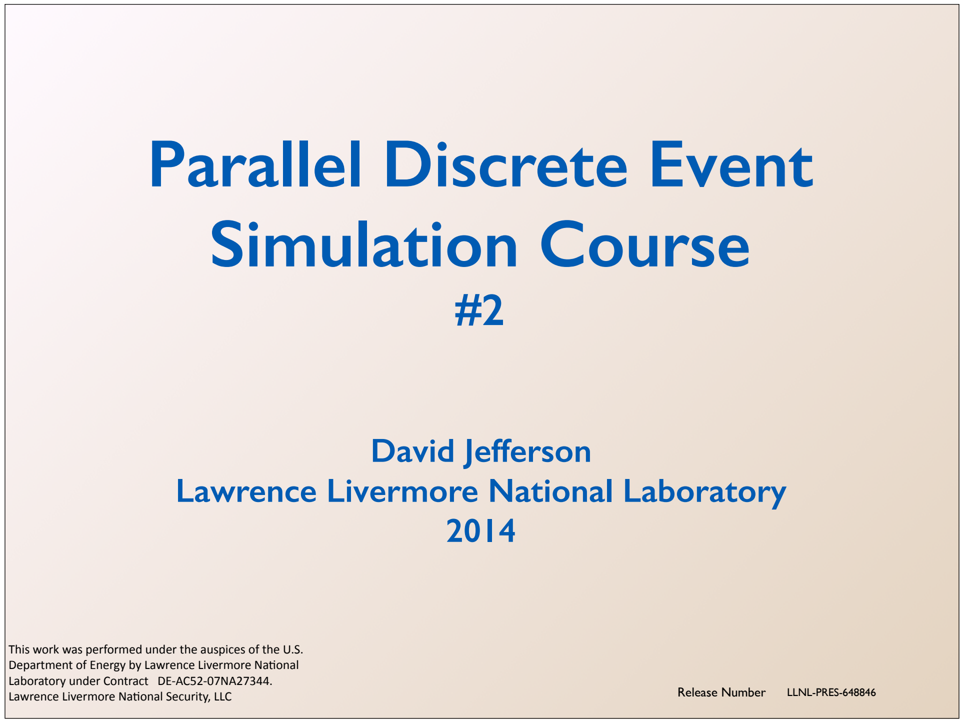
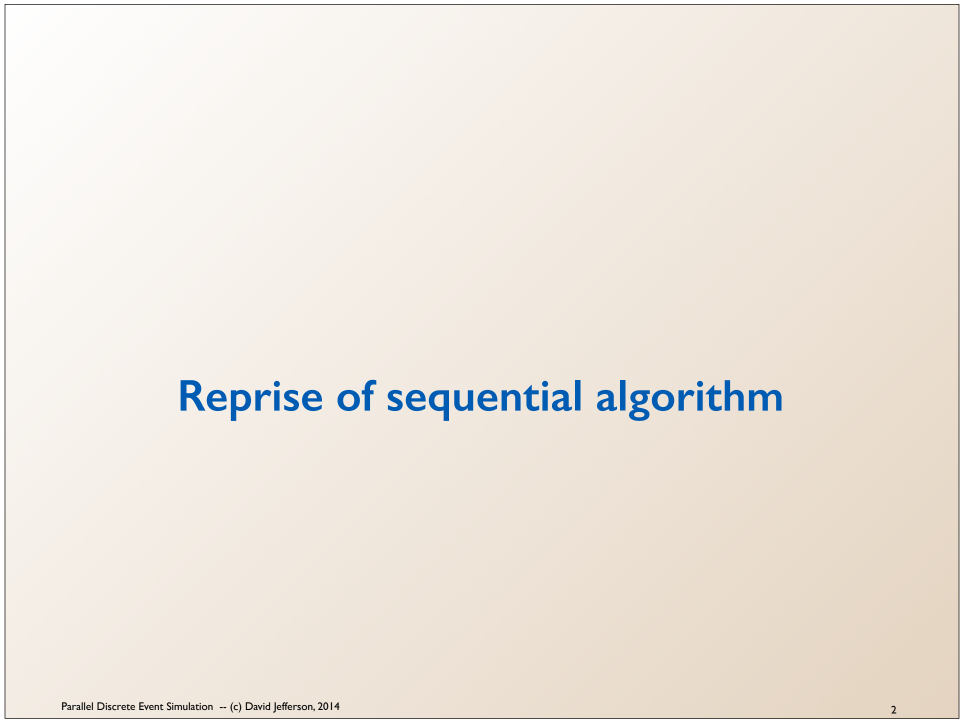


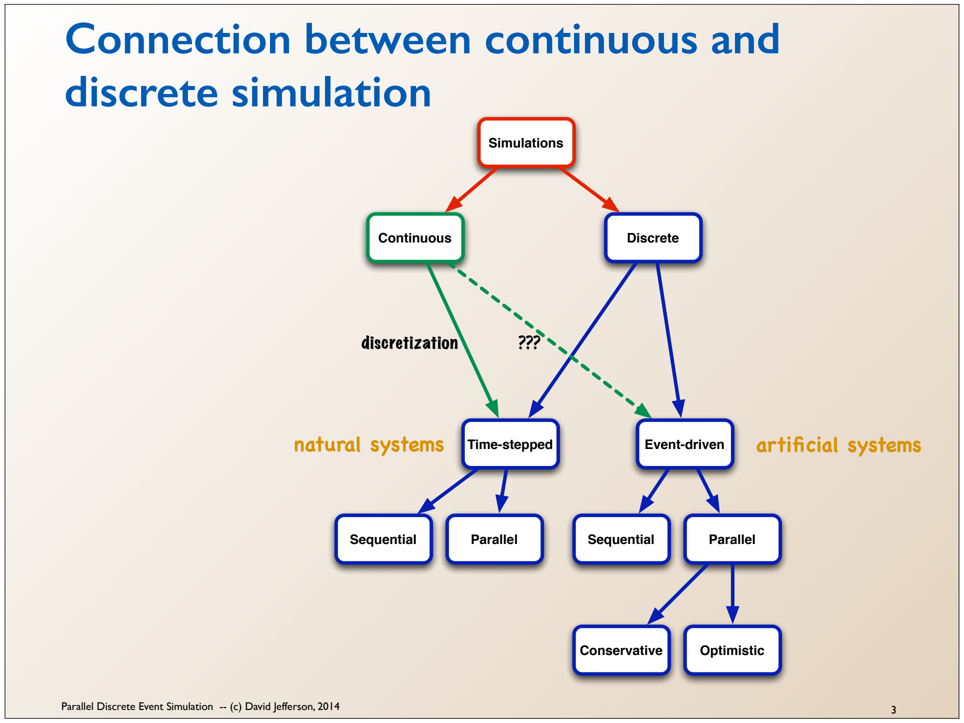
Parallel Discrete Event Simulation Course
#2

David Jefferson
Lawrence Livermore National Laboratory
2014

Release Number LLNL-PRES-648846



Reprise of sequential algorithm



Connection between continuous and discrete simulation

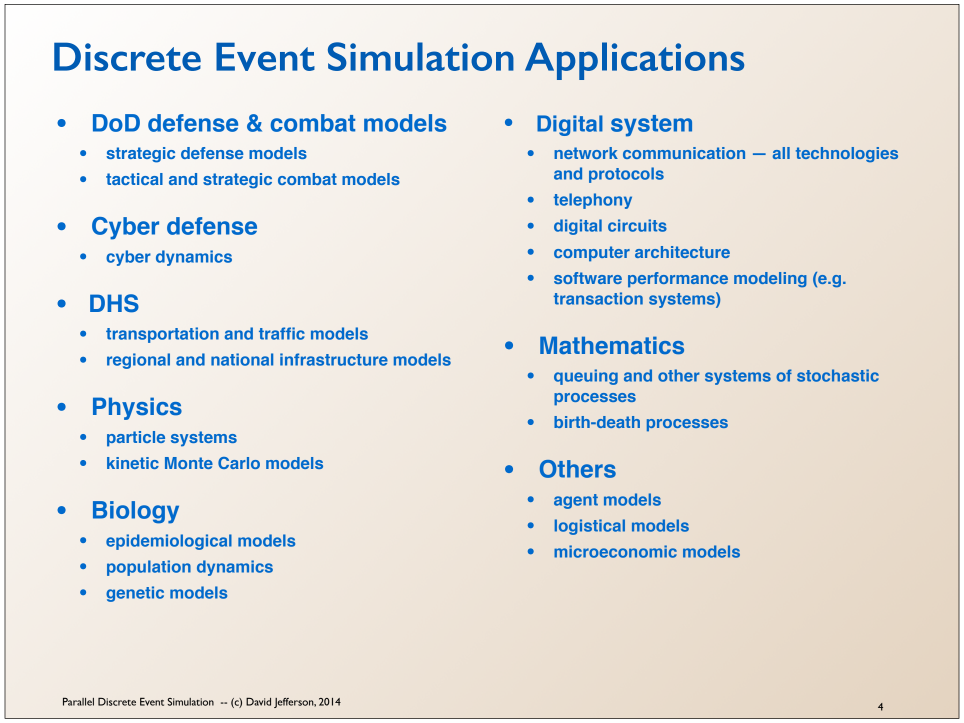
Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

3

Discretization of a continuous model is the transformation from equational form into a discrete model. Virtually always this is a time-stepped model.

Crude generalization: natural systems tend to be most efficiently simulated by time-stepped models, whereas artificial systems are often simulated with event-driven models.

Grand challenge: Develop techniques for transforming continuous models into discrete event models. No one knows how to do that in general, although there are hints coming. Would solve the general problem of multiscale simulation, in time at least.



Discrete Event Simulation Applications

- DoD defense & combat models
  - strategic defense models
  - tactical and strategic combat models
- Cyber defense
  - cyber dynamics
- DHS
  - transportation and traffic models
  - regional and national infrastructure models
- Physics
  - particle systems
  - kinetic Monte Carlo models
- Biology
  - epidemiological models
  - population dynamics
  - genetic models
- Digital system
  - network communication — all technologies and protocols
  - telephony
  - digital circuits
  - computer architecture
  - software performance modeling (e.g. transaction systems)
- Mathematics
  - queuing and other systems of stochastic processes
  - birth-death processes
- Others
  - agent models
  - logistical models
  - microeconomic models

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 4

What are the objects? What are their states? What are the events (state changes)?

MDA and SSA

objects: missiles, sensors, communication relays, debris, etc.

events: launches, maneuvers, observations, communication acts (packet transmission), collision, etc.

Combat models

objects: units, platforms, weapons, communication devices, etc.

events: moving, sensing, firing, exploding, communication, etc.

Transportation:

objects: vehicles

events: start on next segment (of road, or flight plan, etc.), arrive at end of segment, interact with another vehicle, crash, etc.

Biology:

objects: organisms

events: movement/migration, mutation, birth, infections contact, death, etc.

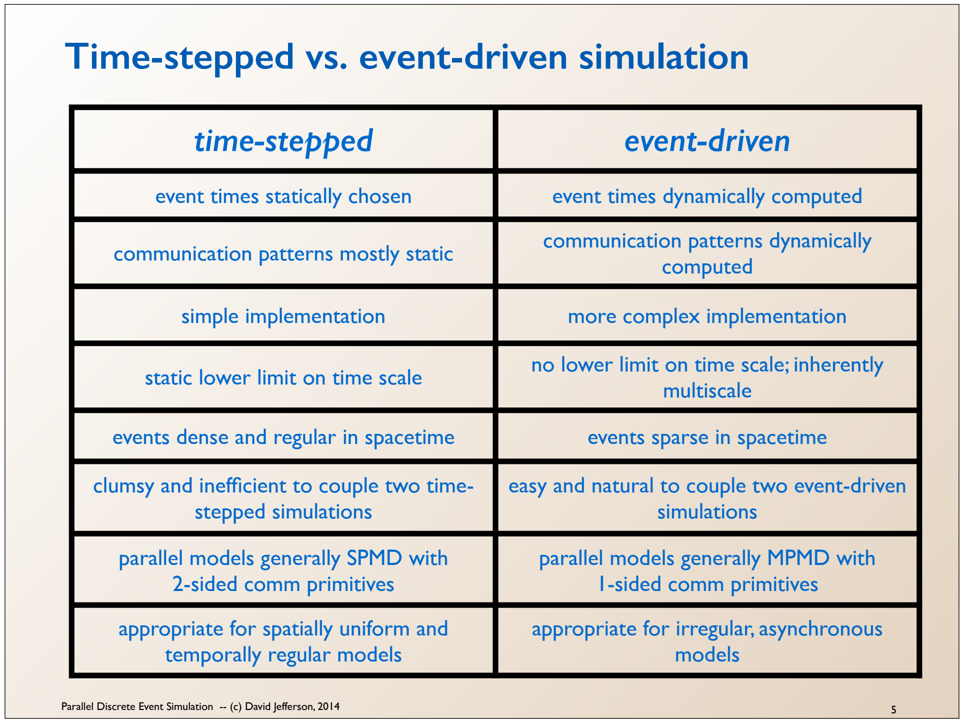
Network models:

objects: nodes, routers, switches

events: packet send, packet arrive, switching decision, routing decision, packet drop, etc.

Economic models:

objects: humans, corporations (producers, consumers, brokers, banks, markets), gov't agencies, etc.



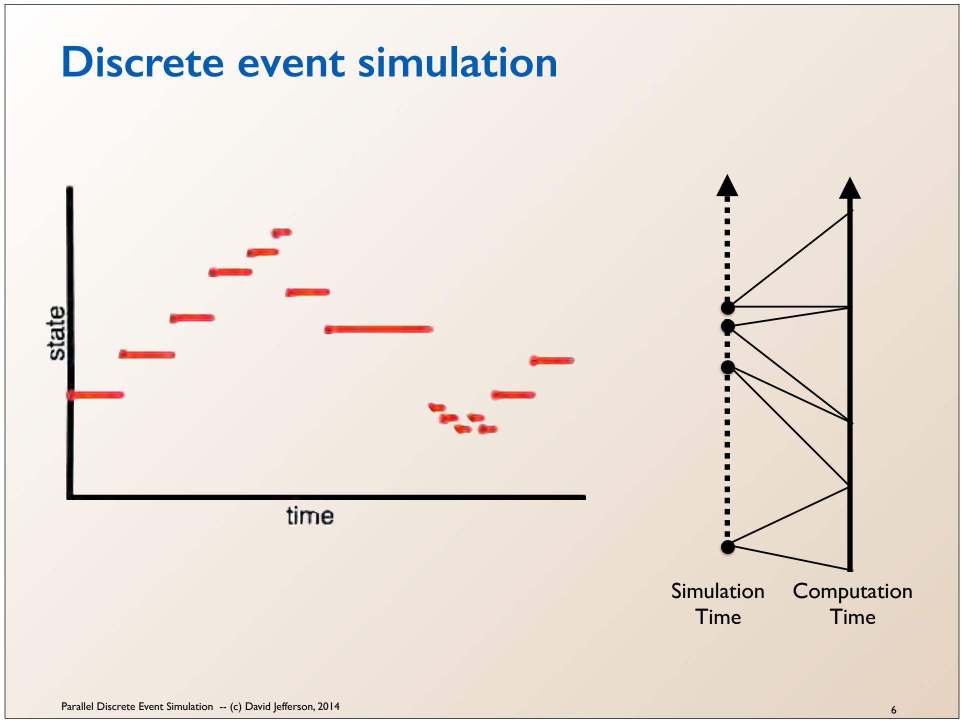
Time-stepped vs. event-driven simulation

time-stepped	*event-driven*
event times statically chosen	event times dynamically computed
communication patterns mostly static	communication patterns dynamically computed
simple implementation	more complex implementation
static lower limit on time scale	no lower limit on time scale; inherently multiscale
events dense and regular in spacetime	events sparse in spacetime
clumsy and inefficient to couple two time-stepped simulations	easy and natural to couple two event-driven simulations
parallel models generally SPMD with 2-sided comm primitives	parallel models generally MPMD with 1-sided comm primitives
appropriate for spatially uniform and temporally regular models	appropriate for irregular, asynchronous models

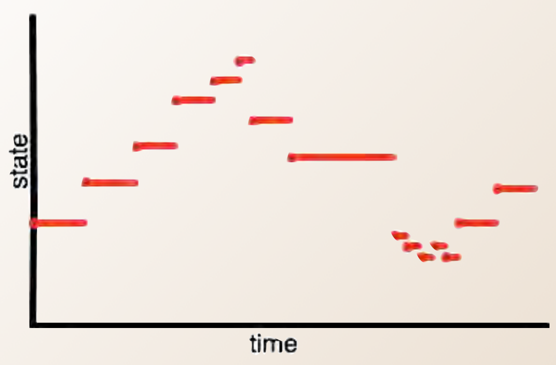
Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 5

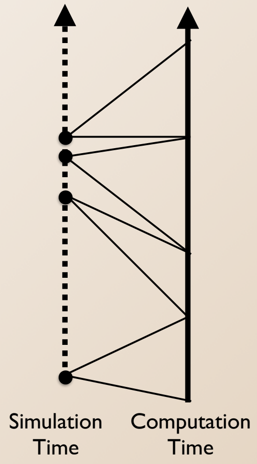
time-stepped
- SPMD
  - timestepped simulations are usually SPMD (single program, multiple data, running the same code in all processes)
- a key consequence is that there is a static limit to how fine a time scale is used in a simulation
- continuous simulations are discretized by transforming the continuum model (equation) into a discrete time-stepped simulation
  - AMR is an attempt to get around the limitations of that approach
  - parallel time-stepped simulations need not, but usually do, run the exact same algorithm at every spatial point and are SPMD

event driven
- MPMD
  - event driven simulations usually have more than one type of object, and are MPMD (multiple program, multiple data, running different code in each process)
- event-driven simulations the simTimes of events are calculated dynamically
  - also generally the event locations are computed dynamically; thus the communication pattern can be arbitraily irregular
  - in such a case there is no limit to how fine a time scale may evolve in a simulation
  - in this respect, event-driven simulation is more general than time-stepped simulation as a paradigm
  - imagine how continuous simulation might change if equations were discretized as event-driven rather than time driven models!
    - AMR is a step in this direction, for the case when there is a spatial continuum as well
    - but no one has a full theory yet about how to do this right



Discrete event simulation



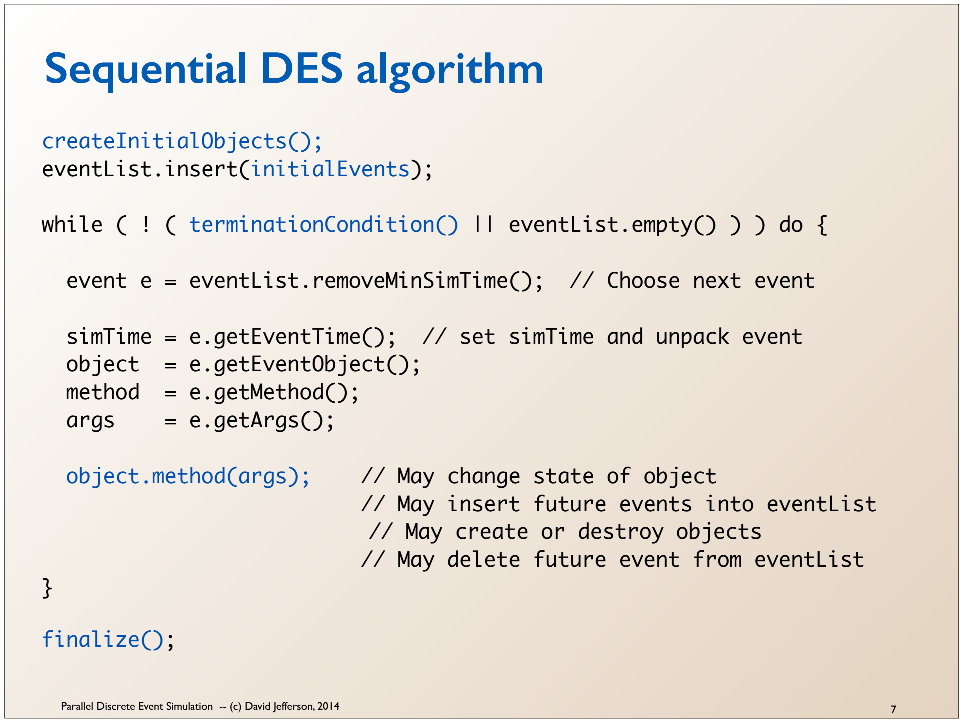


Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 6

The state of the system (or of any object in the system) changes discontinuously at arbitrarily irregular moments in time. The state changes take place instantaneously in simulation time.

The state remains constant between events.

The right hand diagram shows two time lines, one the standard simulation time line with events marked as dots, and the other a computation (wallclock) time line. The diagonal line form triangles that show how much wall clock time is taken by each event*computation* takes place in during the the



Sequential DES algorithm

```
createInitialObjects();
eventList.insert(initialEvents);

while ( ! ( terminationCondition() || eventList.empty() ) ) do {

  event e = eventList.removeMinSimTime();  // Choose next event

  simTime = e.getEventTime();  // set simTime and unpack event
  object  = e.getEventObject();
  method  = e.getMethod();
  args    = e.getArgs();

  object.method(args);    // May change state of object
                          // May insert future events into eventList
                           // May create or destroy objects
                          // May delete future event from eventList
}

finalize();
```

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 7

This is an object-oriented style of DES. We distinguish the simula***tor*** code from the simula**tion** code

Blue text is the simulation code--the model itself
Black text is the simulation algorithm itself--the platform

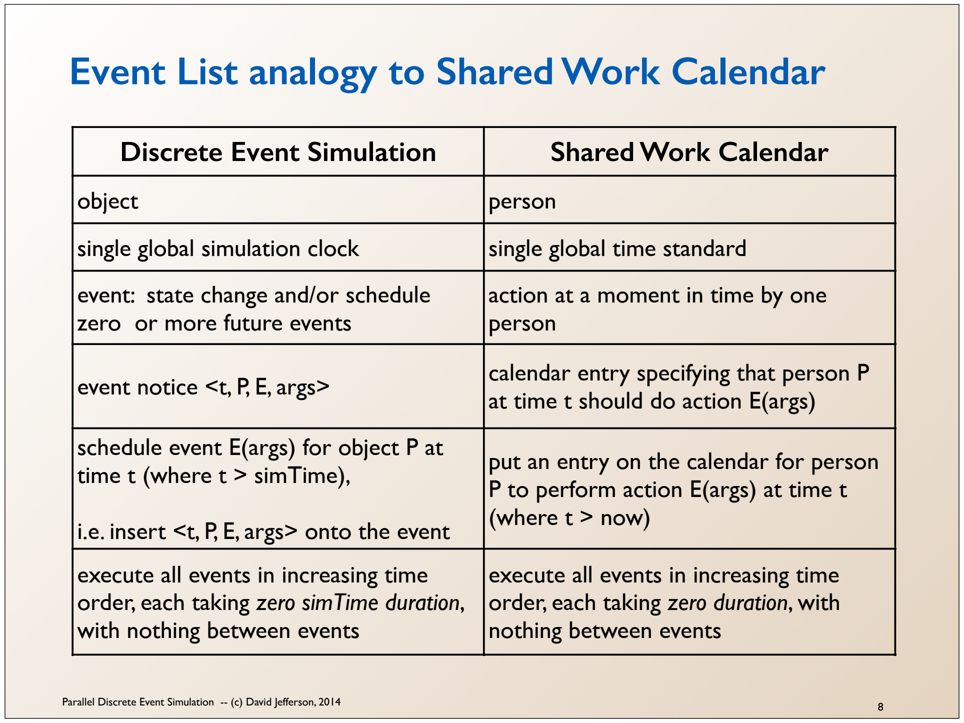
Besides initialization and termination issues, all of the work of the model code is in the object methods

Assume (a) no ties in spacetime (i.e. no two events happen at the same object at the same simTime), and (b) no zero-delay in simTime between the scheduling of an event and the event itself.

- an event at time T can only schedule events at times strictly >T
- no two events for the same object have the same simTime
    - but two events for **different** objects may have the same time
- hence simTime strictly nondecreasing with every event

Reasons for, refinements of, and consequences of these assumptions will come later

- surprisingly complex, especially for parallel sim
- must prepare the ground

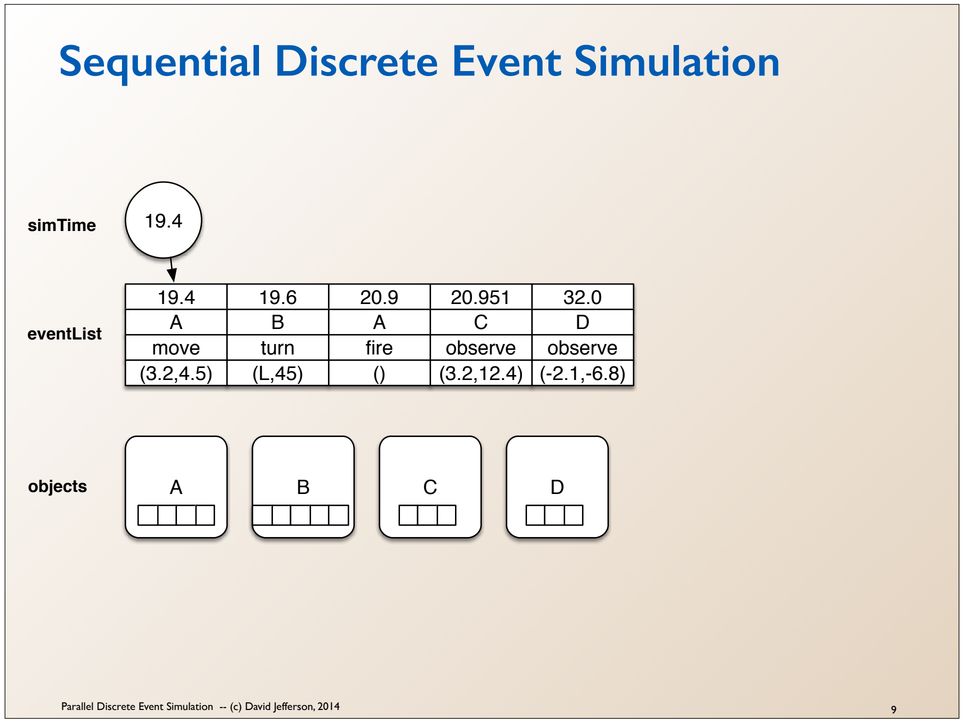


Event List analogy to Shared Work Calendar

Discrete Event Simulation	Shared Work Calendar
object	person
single global simulation clock	single global time standard
event: state change and/or schedule zero or more future events	action at a moment in time by one person
event notice <t, P, E, args>	calendar entry specifying that person P at time t should do action E(args)
schedule event E(args) for object P at time t (where t > simTime), i.e. insert <t, P, E, args> onto the event	put an entry on the calendar for person P to perform action E(args) at time t (where t > now)
execute all events in increasing time order, each taking *zero simTime duration*, with nothing between events	execute all events in increasing time order, each taking *zero duration*, with nothing between events

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

8



Sequential Discrete Event Simulation

19.4	19.6	20.9	20.951	32.0
A	B	A	C	D
move	turn	fire	observe	observe
(3.2,4.5)	(L,45)	()	(3.2,12.4)	(-2.1,-6.8)

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

9

4 objects, A, B, C and D, with their internal states

current simTime is 19.4

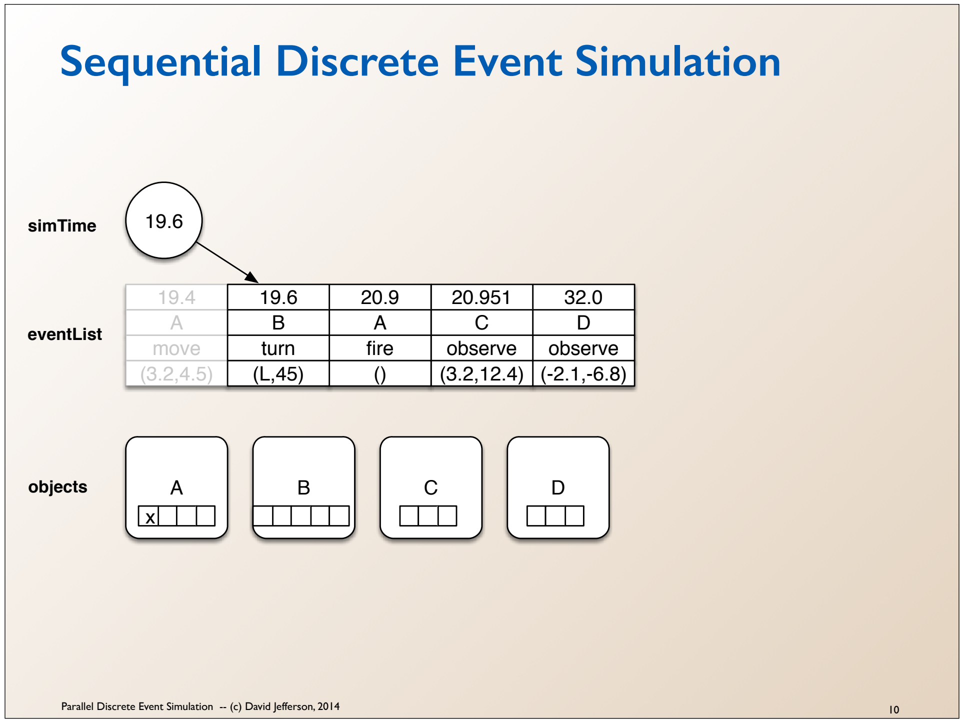
event list has 5 events scheduled in it

simTime is always the same as the lowest simTime event in the event list

event list is not really a linear list--it is generally a tree

heap, red-black tree, splay tree

A is executing the move method



Sequential Discrete Event Simulation

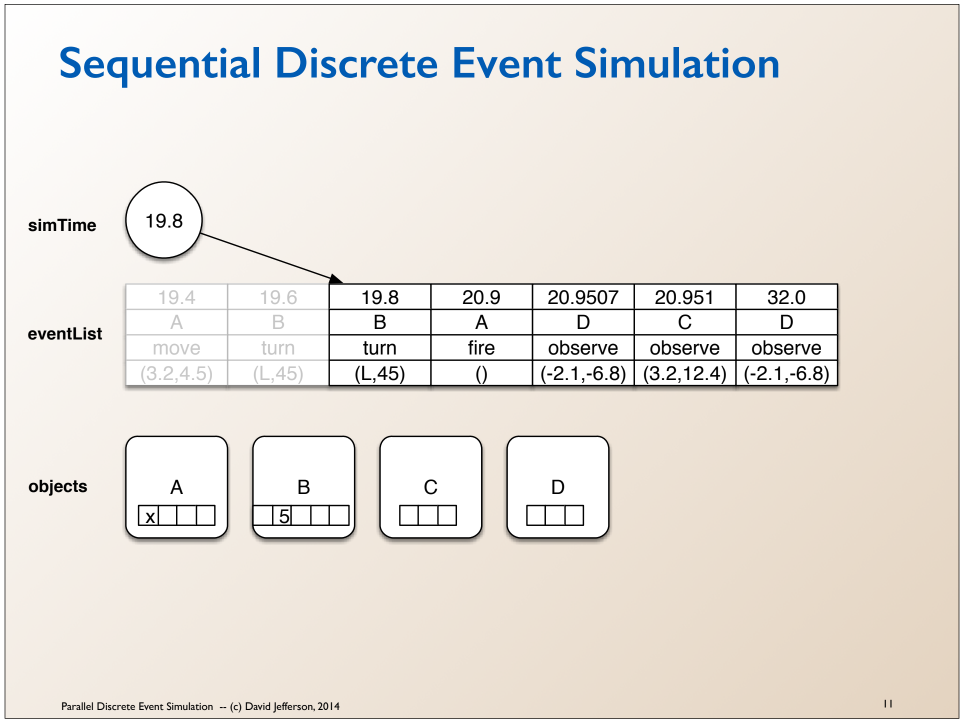
19.4	19.6	20.9	20.951	32.0
A	B	A	C	D
move	turn	fire	observe	observe
(3.2,4.5)	(L,45)	()	(3.2,12.4)	(-2.1,-6.8)

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

10

The move method inserted no events into the eventList, so the next event is at time 19.6 for B

Note A's state has changed



Sequential Discrete Event Simulation

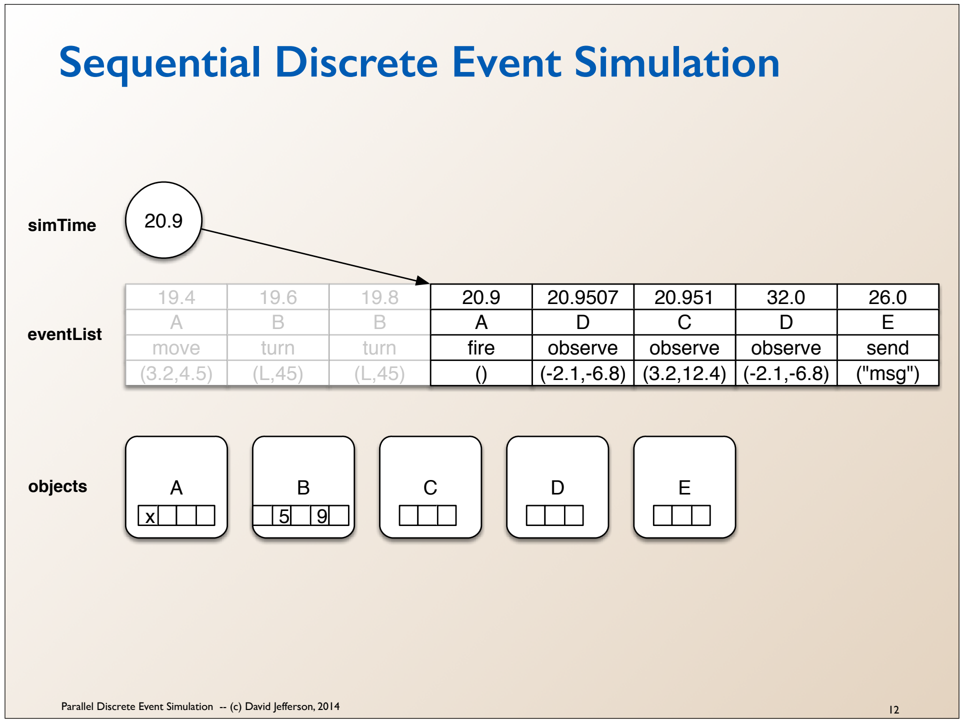
19.4	19.6	19.8	20.9	20.9507	20.951	32.0
A	B	B	A	D	C	D
move	turn	turn	fire	observe	observe	observe
(3.2,4.5)	(L,45)	(L,45)	()	(-2.1,-6.8)	(3.2,12.4)	(-2.1,-6.8)

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

11

B's state has changed, but note that B inserted one event into the event list and it happened to be for itself, and happened to be the very next event.

Hence and event for B runs next also.



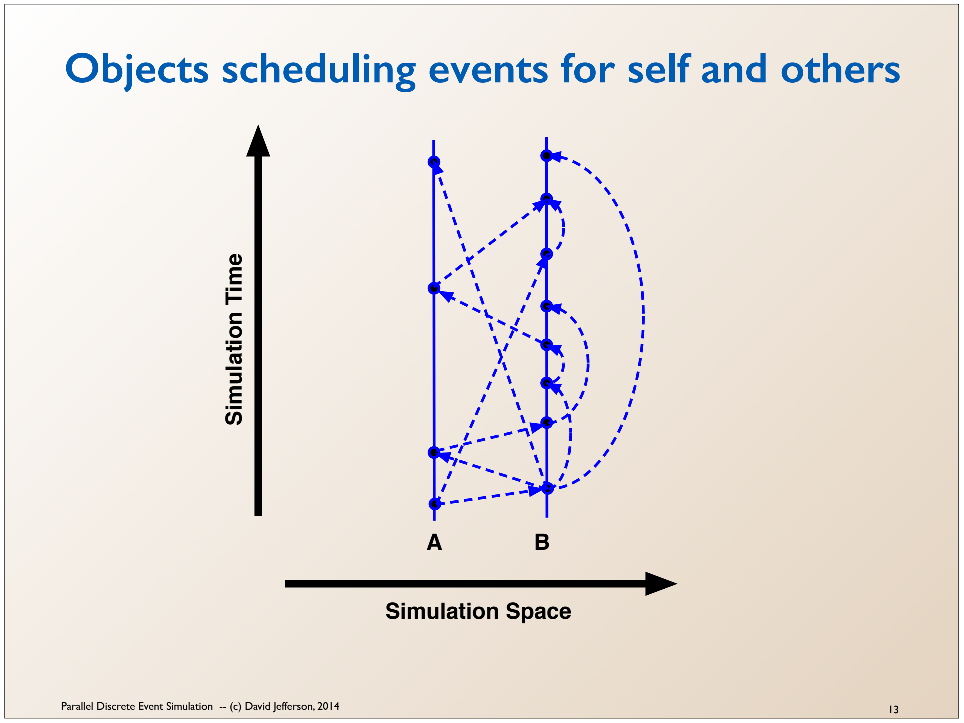
Sequential Discrete Event Simulation

19.4	19.6	19.8	20.9	20.9507	20.951	32.0	26.0
A	B	B	A	D	C	D	E
move	turn	turn	fire	observe	observe	observe	send
(3.2,4.5)	(L,45)	(L,45)	()	(-2.1,-6.8)	(3.2,12.4)	(-2.1,-6.8)	("msg")

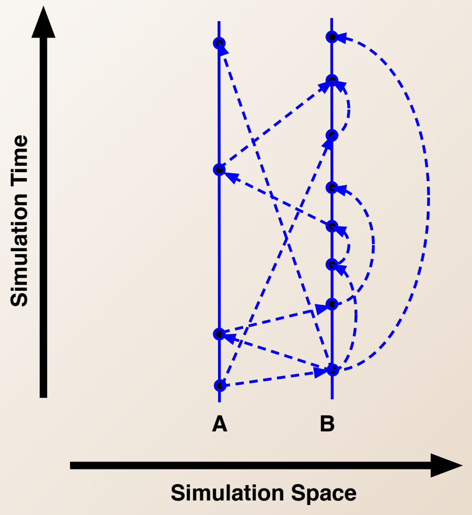
Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

12

B's event caused a new object, E, to be created and also caused and event for E to be scheduled



Objects scheduling events for self and others



Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

13

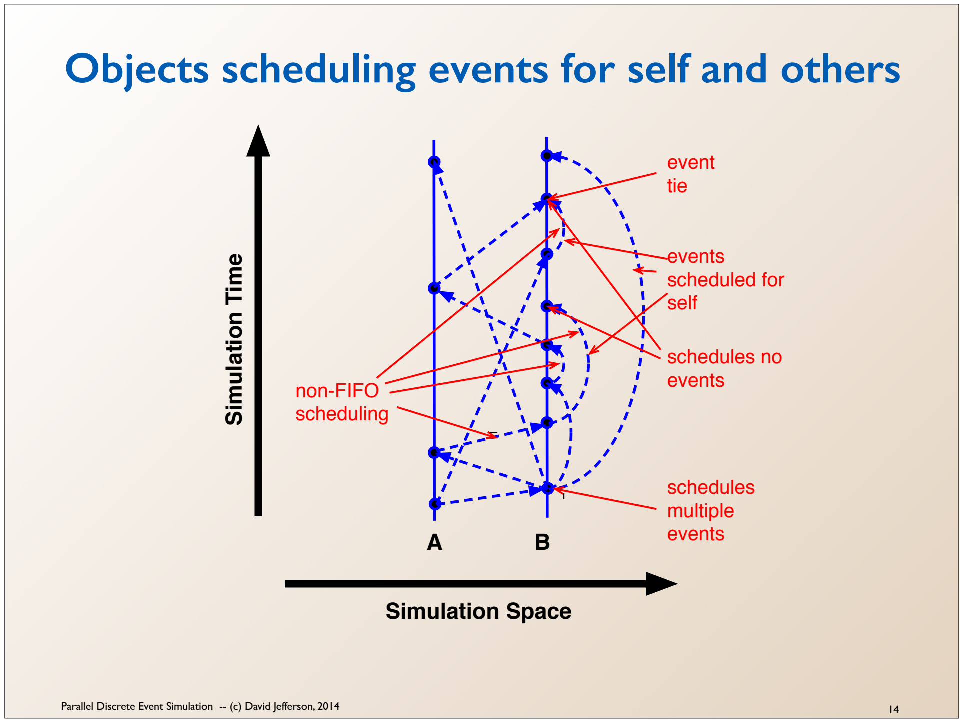
This is a *space-time diagram* representing a portion of the simulation's execution. *Simulation time* increases upward in the diagram.

A and B are objects. *Simulation space* is the namespace of all objects in the model.

The blue vertical lines represent the succession of events in each object.

The blue dots are *events* that take zero simulation time each, but that is where all of the computation takes place. Between events the state of an object is constant.

The diagonal arcs are *event scheduling* actions, or in PDES terminology they are *event messages*. The tail of the arrow is the event that is doing the scheduling of another event, i.e. sending the event message. The head of the arrow is the event being scheduled.



Objects scheduling events for self and others

Simulation Time
Simulation Space
A
B
event tie
events scheduled for self
schedules no events
schedules multiple events
non-FIFO scheduling
Parallel Discrete Event Simulation -- (c) David Jefferson, 2014
14

An event may schedule zero or more other events. We have examples of each here.

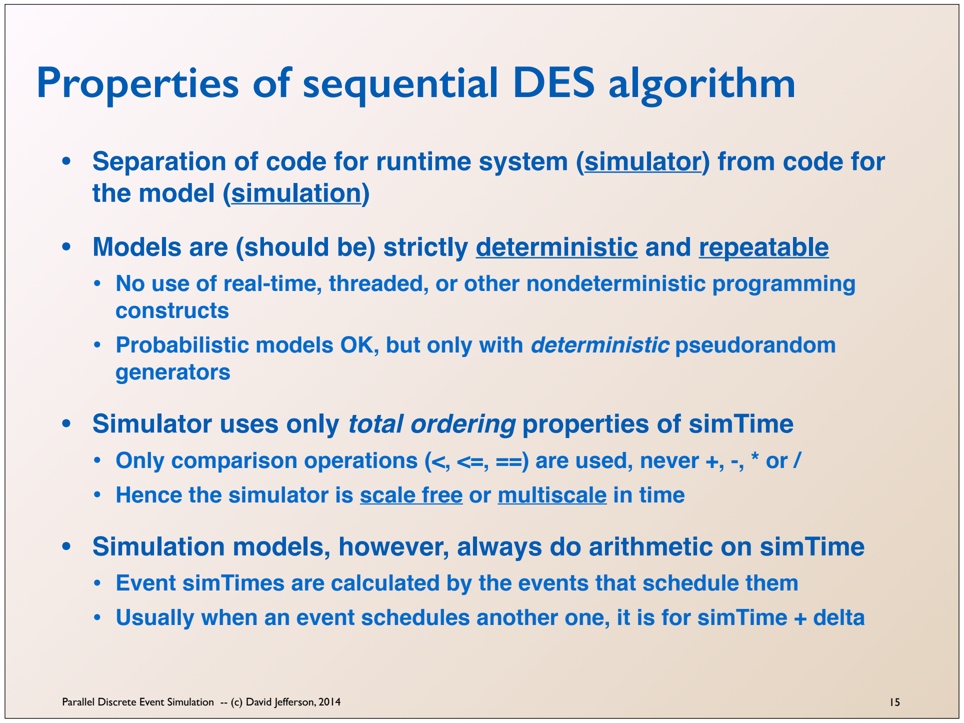
All events are scheduled for the *future* in simulation time, so all of these arrows go upward in the diagram. Events are *never* scheduled in the past, as that would violate causality. The consequences of allowing events to be scheduled in the *present*, with zero simTime delay between the scheduling and scheduled event, are discussed later.

Events may schedule future events for themselves, and that is very common.

Events do not have to scheduled in FIFO order, i.e. it is OK for A to send an event message to B and then at a later simulation time A might send and event message to B for an event to be executed at an *earlier* sim time that the first message. This is not strange if one considers the analogy to calendars. There is no problem with A at time 2 scheduling and event on B's calendar to be executed at time 10 and then for A at time 4 to schedule and event for B to execute at time 7.

Every event (except the initial events) is scheduled by another event, usually exactly one. Hence there is one arrowhead coming into each event.

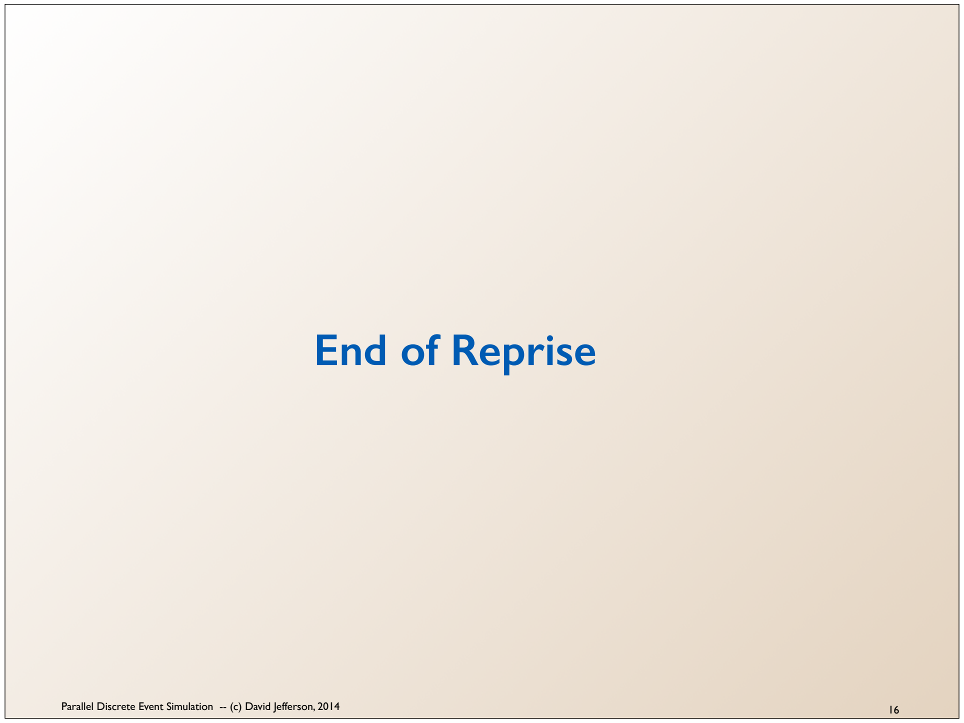
In rare cases two events can be scheduled for the same object at exactly the same time. We call that an event tie. The issues with event ties are complicated, and will be discussed later, but they are semantic, not performance issues. Usually event ties are negligibly rare. If a particular model code results in lots of ties, then one might consider whether a time-stepped simulation is a better fit than an event-driven one.



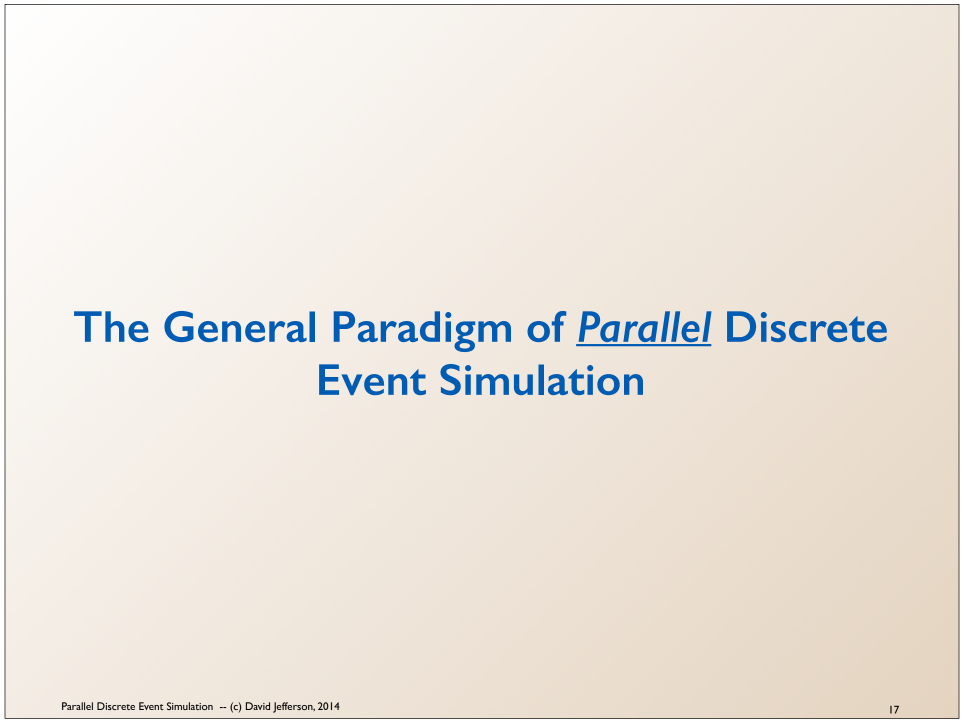
Properties of sequential DES algorithm

- Separation of code for runtime system (simulator) from code for the model (simulation)
- Models are (should be) strictly deterministic and repeatable
  - No use of real-time, threaded, or other nondeterministic programming constructs
  - Probabilistic models OK, but only with *deterministic* pseudorandom generators
- Simulator uses only *total ordering* properties of simTime
  - Only comparison operations (<, <=, ==) are used, never +, -, * or /
  - Hence the simulator is scale free or multiscale in time
- Simulation models, however, always do arithmetic on simTime
  - Event simTimes are calculated by the events that schedule them
  - Usually when an event schedules another one, it is for simTime + delta

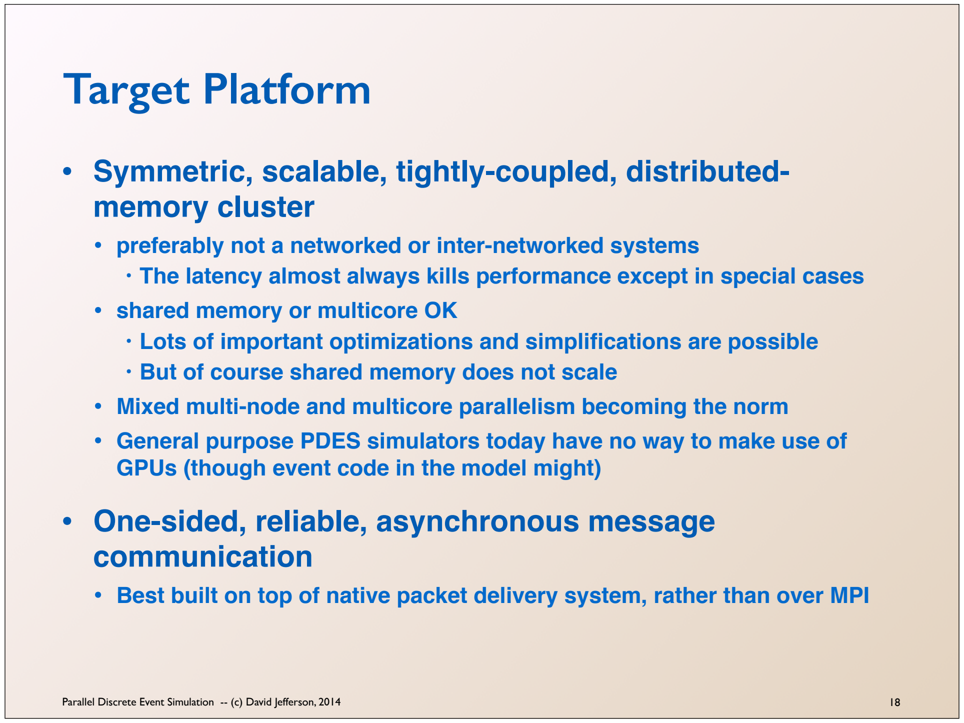
Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 15



End of Reprise



The General Paradigm of ***Parallel*** Discrete Event Simulation



Target Platform

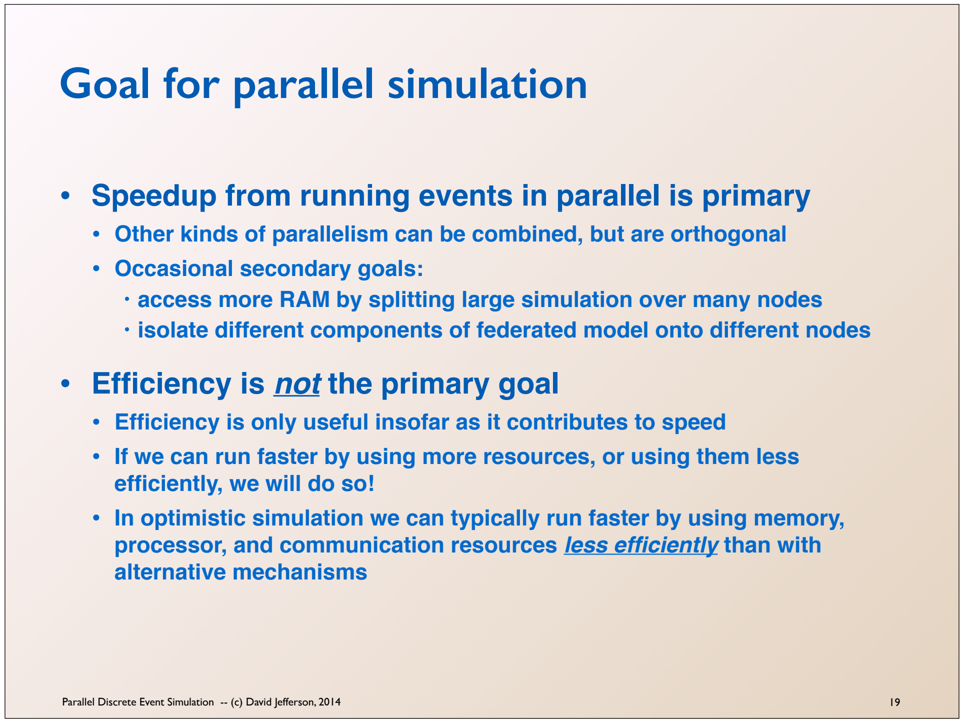
- **Symmetric, scalable, tightly-coupled, distributed-memory cluster**
  - preferably not a networked or inter-networked systems
    - The latency almost always kills performance except in special cases
  - shared memory or multicore OK
    - Lots of important optimizations and simplifications are possible
    - But of course shared memory does not scale
  - Mixed multi-node and multicore parallelism becoming the norm
  - General purpose PDES simulators today have no way to make use of GPUs (though event code in the model might)
- **One-sided, reliable, asynchronous message communication**
  - Best built on top of native packet delivery system, rather than over MPI

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 18

We target our algorithms for the distributed memory architecture because
(a) the problem is more difficult than in shared memory, as several important simplifications logical and performance apply in shared memory, and
(b) we are interested in *scalable* simulation mechanisms

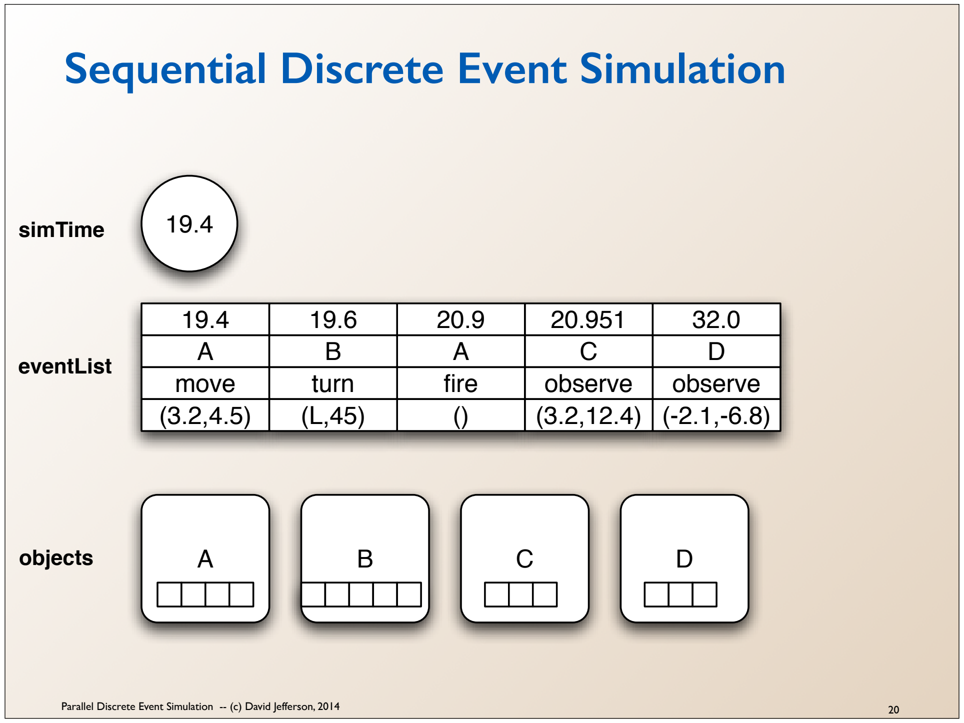
Now that multicore machines are nearly universal, there is a need to allow many objects to *reside* in the memory of the same node, and we will discuss that later in the course.

But even then, we will not allow shared variables between two objects. It is not that it is impossible to define it, but it so greatly complicates synchronization algorithms that it is not worth it, and it risks nondeterminism.



Goal for parallel simulation

- **Speedup from running events in parallel is primary**
  - **Other kinds of parallelism can be combined, but are orthogonal**
  - **Occasional secondary goals:**
    - **access more RAM by splitting large simulation over many nodes**
    - **isolate different components of federated model onto different nodes**
- **Efficiency is *not* the primary goal**
  - **Efficiency is only useful insofar as it contributes to speed**
  - **If we can run faster by using more resources, or using them less efficiently, we will do so!**
  - **In optimistic simulation we can typically run faster by using memory, processor, and communication resources *less efficiently* than with alternative mechanisms**



Sequential Discrete Event Simulation

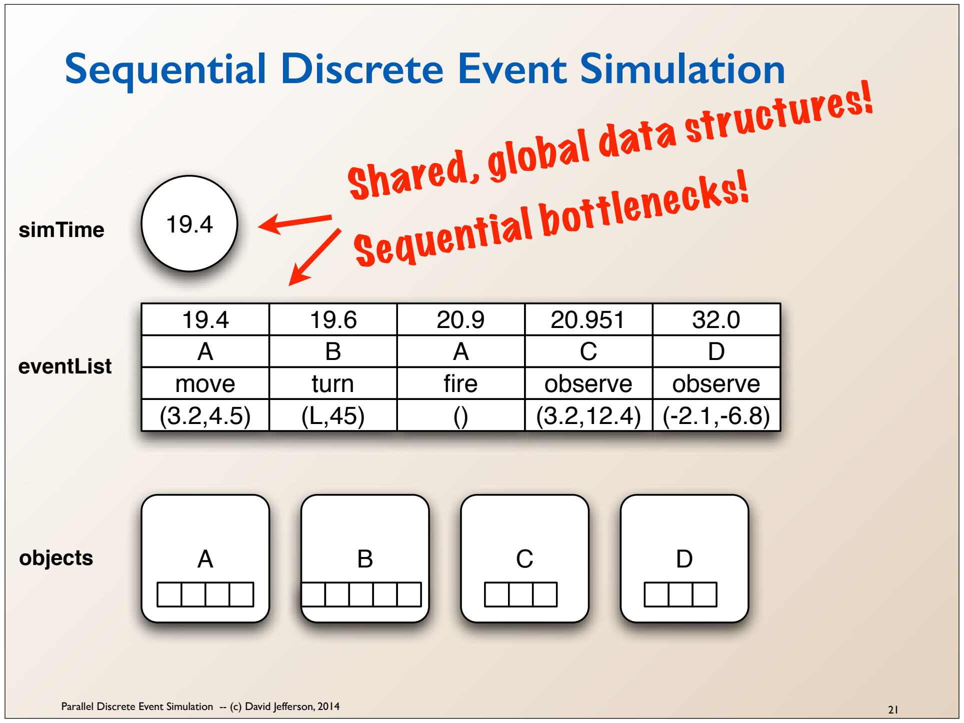
simTime

19.4

eventList

19.4	19.6	20.9	20.951	32.0
A	B	A	C	D
move	turn	fire	observe	observe
(3.2,4.5)	(L,45)	()	(3.2,12.4)	(-2.1,-6.8)

objects



Sequential Discrete Event Simulation

simTime

19.4

Shared, global data structures!

Sequential bottlenecks!

eventList

19.4	19.6	20.9	20.951	32.0
A	B	A	C	D
move	turn	fire	observe	observe
(3.2,4.5)	(L,45)	()	(3.2,12.4)	(-2.1,-6.8)

objects

4 objects, A, B, C and D, with their internal states

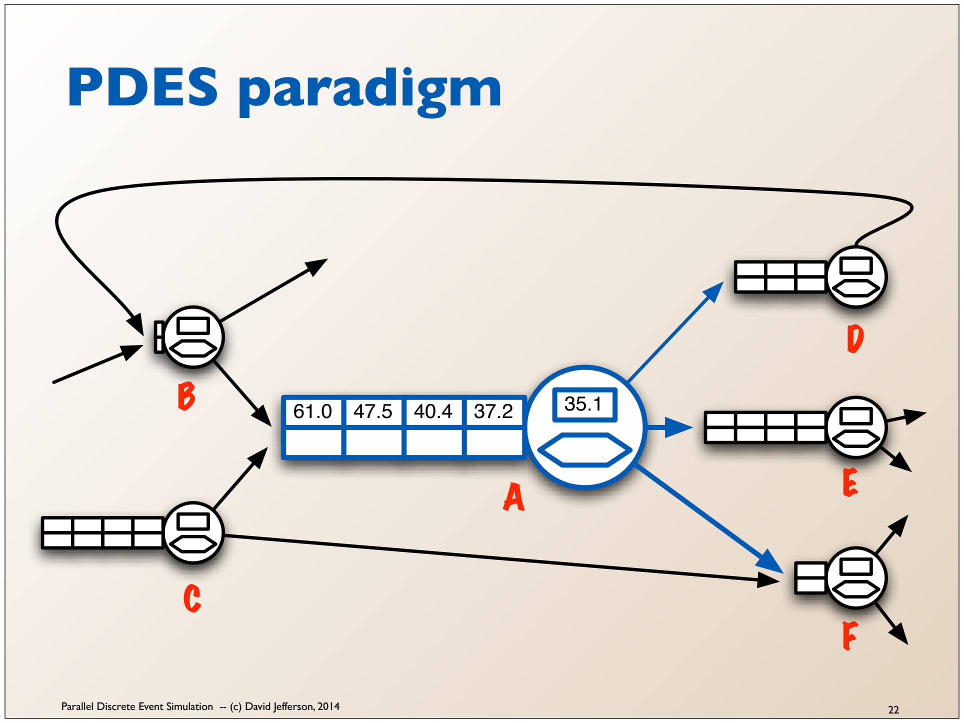
current simTime is 19.4

event list has 5 events scheduled in it

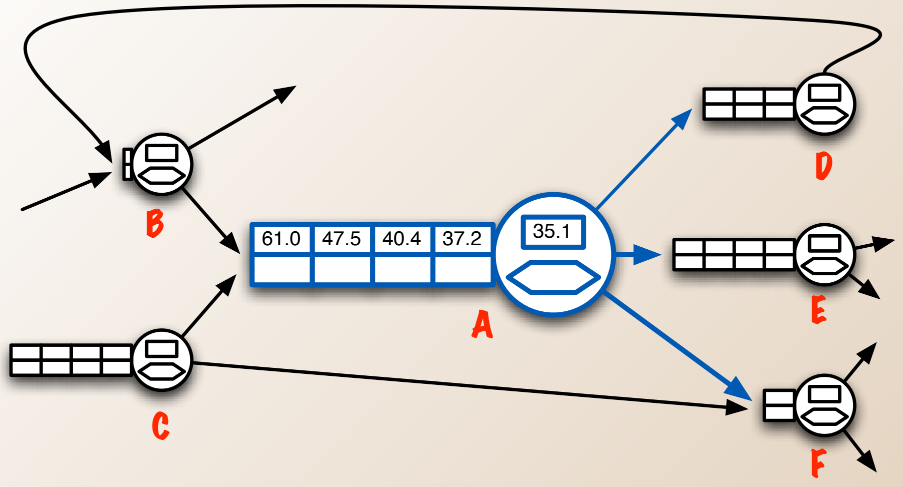
simTime is always the same as the lowest simTime event in the event list

event list is not really a linear list--it is generally a tree
heap, red-black tree, splay tree

A is executing the move method



PDES paradigm



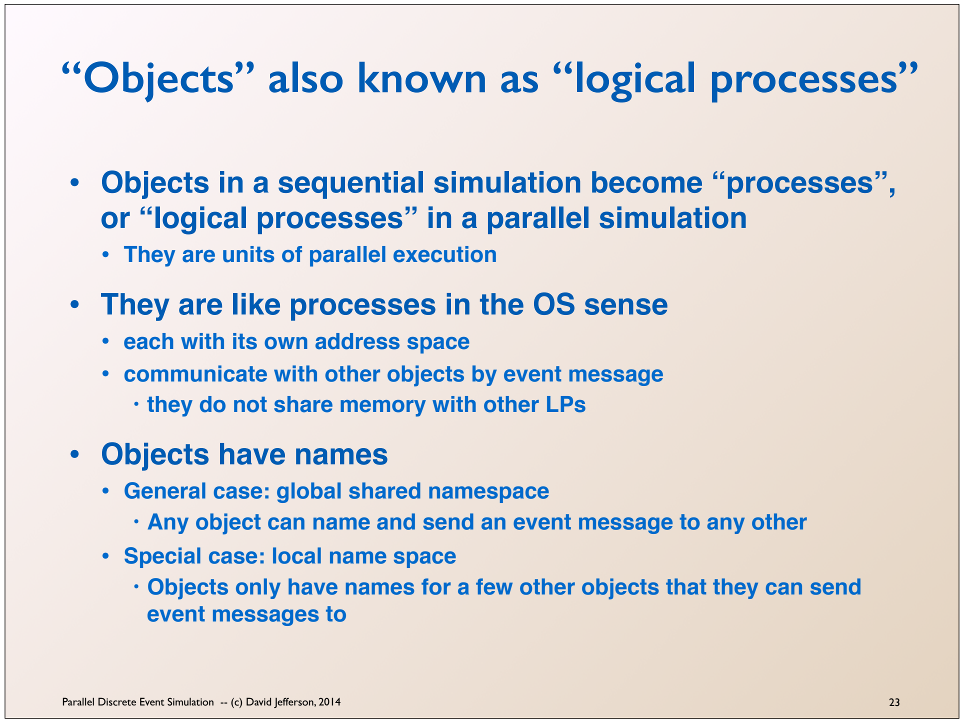
To resolve the issue that the event list simulation clock are sequential bottlenecks, we give each object own simulation clock and its own event list. In order to run on distributed memory machines rather than just shared memory, we schedule events by sending event messages and rename the "event list" as an event message queue or event queue or just input queue. The simulation time for which an event is scheduled is called its timestamp. Event queues are of course priority queues sorted by timestamp. And although that are shown here as linear, in reality it is usually best if they are represented as a red-black tree or a splay tree.

Each circle with its input queue is a simulation object or process or logical process in the simulation. There may be millions of such objects in a big simulation. Each double rectangle is an event message, indicating a method to be called and parameters to be passed to the method, and the simulation time at which it should be done. Hexagons inside the circles represent the state of an object

Each object has its own event message queue, like the event list of the sequential DES algorithm. It is sorted by the timestamp of the message. (Of course it is not usually a linear list—it is usually a balanced tree of some kind.)

Each object has its own simulation clock that tells its simTime (or virtual time). The process in the center of this diagram is processing an event message whose time stamp is 35.1; that process is now simulating at time 35.1. Other clocks need not necessarily agree -- some will be ahead and others will be behind.

Objects all execute mostly asynchronously, and in parallel, but are synchronized with one another to maintain causal consistency. (The reason will be clear later: if you attempt to keep all objects time synchronized, you get no parallelism!)


“Objects” also known as “logical processes”

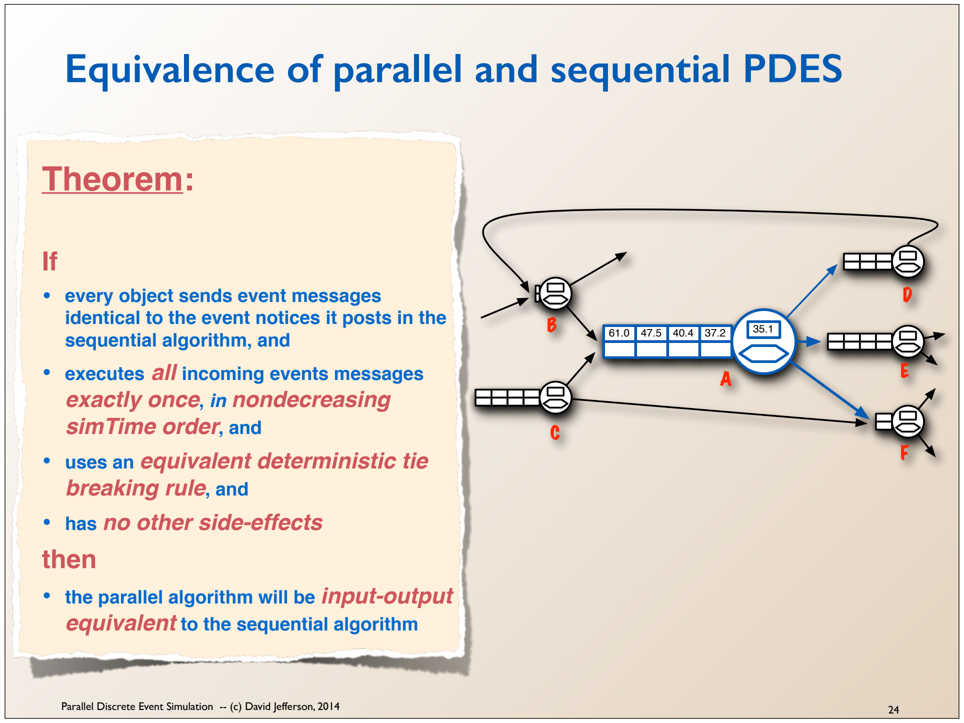
- Objects in a sequential simulation become “processes”, or “logical processes” in a parallel simulation
  - They are units of parallel execution
- They are like processes in the OS sense
  - each with its own address space
  - communicate with other objects by event message
    - they do not share memory with other LPs
- Objects have names
  - General case: global shared namespace
    - Any object can name and send an event message to any other
  - Special case: local name space
    - Objects only have names for a few other objects that they can send event messages to

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 23

The use of the term “process” or “logical process” is an acknowledgement that the object is the unit of parallel execution and synchronization and has its own address space not shared with other logical processes

Global name space is like that for a file system: any code can construct any file name and attempt to open it. The file name space is statically and globally known. What names have files attached to them may not be known to a process, but the space of legal names is.

An example of a local name space would be a PDES paradigm in which the logical process are arranged in a graph and are only permitted to send events to their neighbors in the graph.



Equivalence of parallel and sequential PDES

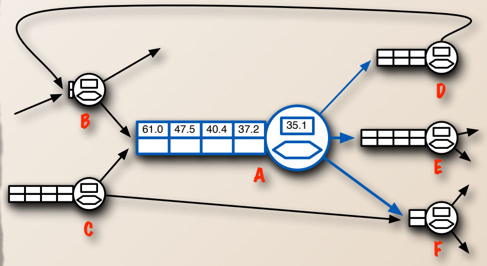
Theorem:

If

- every object sends event messages identical to the event notices it posts in the sequential algorithm, and
- executes *all* incoming events messages *exactly once*, in *nondecreasing simTime order*, and
- uses an *equivalent deterministic tie breaking rule*, and
- has *no other side-effects*

then

- the parallel algorithm will be *input-output equivalent* to the sequential algorithm

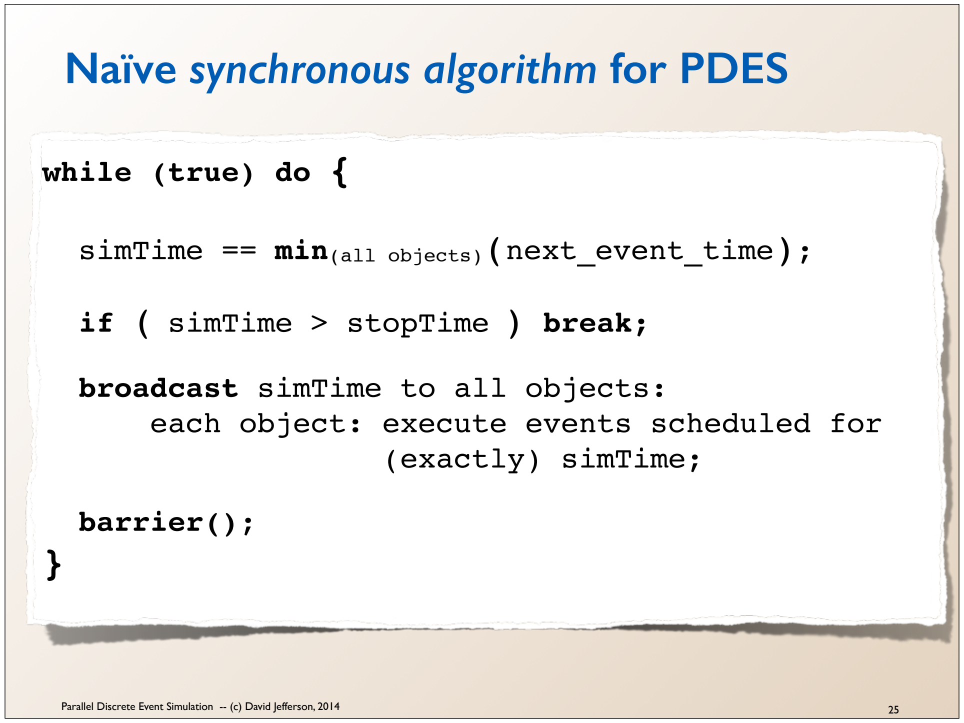


Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

24

This theorem is no-doubt in Euclid somewhere!

Of course assumes that initialization is the same, and events are deterministic, etc.



Naïve *synchronous algorithm* for PDES

```
while (true) do {

  simTime == min(all objects)(next_event_time);

  if ( simTime > stopTime ) break;

  broadcast simTime to all objects:
      each object: execute events scheduled for
                   (exactly) simTime;

  barrier();
}
```

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 25

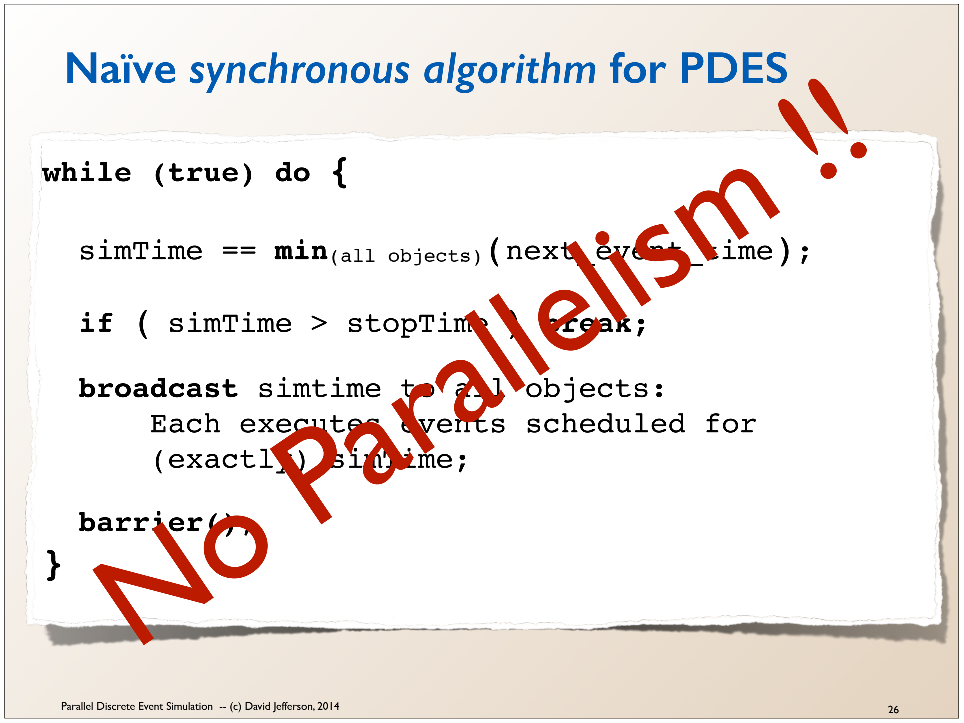
Loop body has 3 global collective operations:

1st line:	Calculation of minTime is a global reduction to find the minimum simulation time of any as-yet-unexecuted event message globally
3rd line:	Involves a broadcast to all objects. Note that more than one object may have event messages tied at the minTime, and each of them may have more than on event message tied at that time. (Obviously must use appropriate tie-breaking rule.)
4th line:	a barrier

This is the kind of algorithm that might occur to a naïve simulation programmer who is familiar with SPMD-style scientific programming

It has the property that no event at simTime == t starts executing before all events with simTime < t are completed.

This algorithm is correct, i.e. equivalent to the sequential algorithm. In fact, it is essentially identical to the sequential algorithm, just distributed across multiple processes.



Naïve *synchronous algorithm* for PDES

```
while (true) do {

  simTime == min(all objects)(next_event_time);

  if ( simTime > stopTime ) break;

  broadcast simtime to all objects:
      Each executes events scheduled for
      (exactly) simtime;

  barrier();
}
```

No Paralelism !!

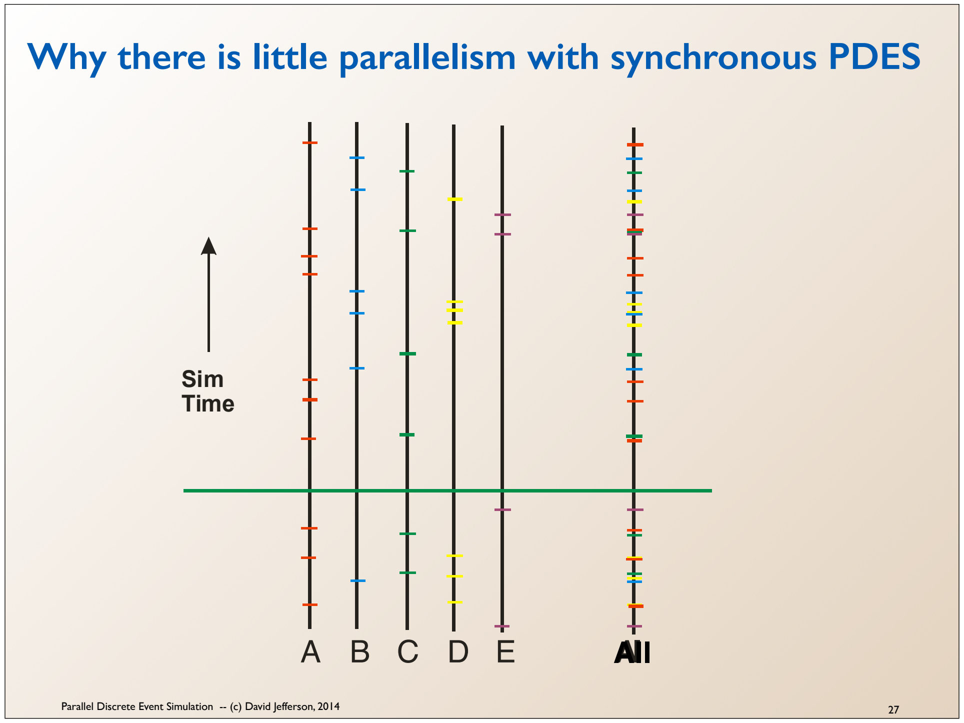
Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

26

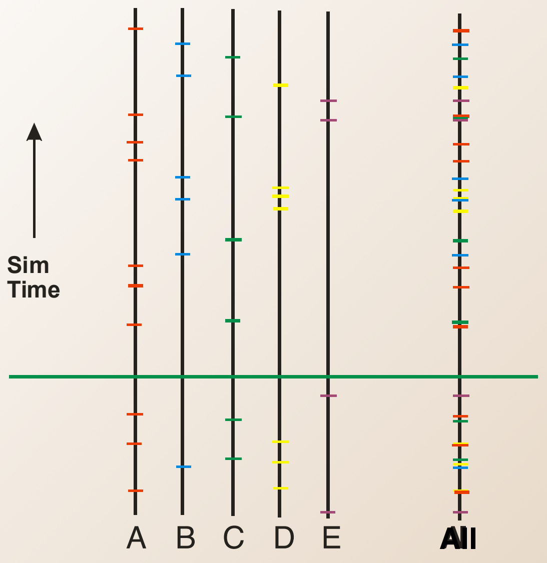
Unfortunately, because ties are so rare, this essentially is a ***sequential*** algorithm. Unless there are many ties at each distinct simTime, and unless those ties are spread across numerous objects / nodes, then essentially no parallelism is achieved. And considering the collectives, it will run much slower than the sequential algorithm would (if there were enough RAM to run the sequential algorithm)!

The problem is the hard synchronization property that no event at simTime == t starts executing before all events with simTime < t are finished. Put another way, it tries to guarantee that the entire parallel simulation stays synchronized in simulation time, so that in any given instantaneous snapshot all objects in the simulation have progressed to the same simulation time, much like a time-stepped simulation. We must give up that synchronous execution style! We must allow some events to execute before others with lower timestamps, as long as the causality constraints are not violated, or else we generally get no parallelism! This observation is part of the reason for the name "Time Warp" in the fundamental optimistic PDES algorithm I will show later. (Another part is that algorithm's ability to go forward and backward in time ... more later.)

This is why the objects simulation clocks and are not kept synchronized. It would force sequentialization.



Why there is little parallelism with synchronous PDES



Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

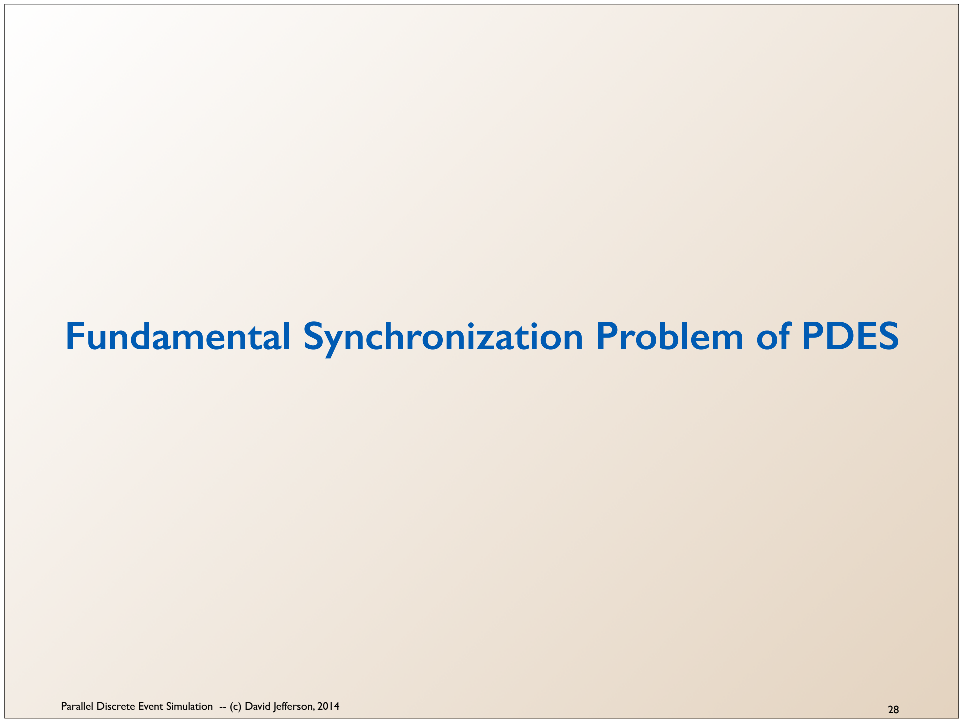
27

The simulation time lines for five object are shown on the vertical lines. The colored hash marks are the times at which events occur.

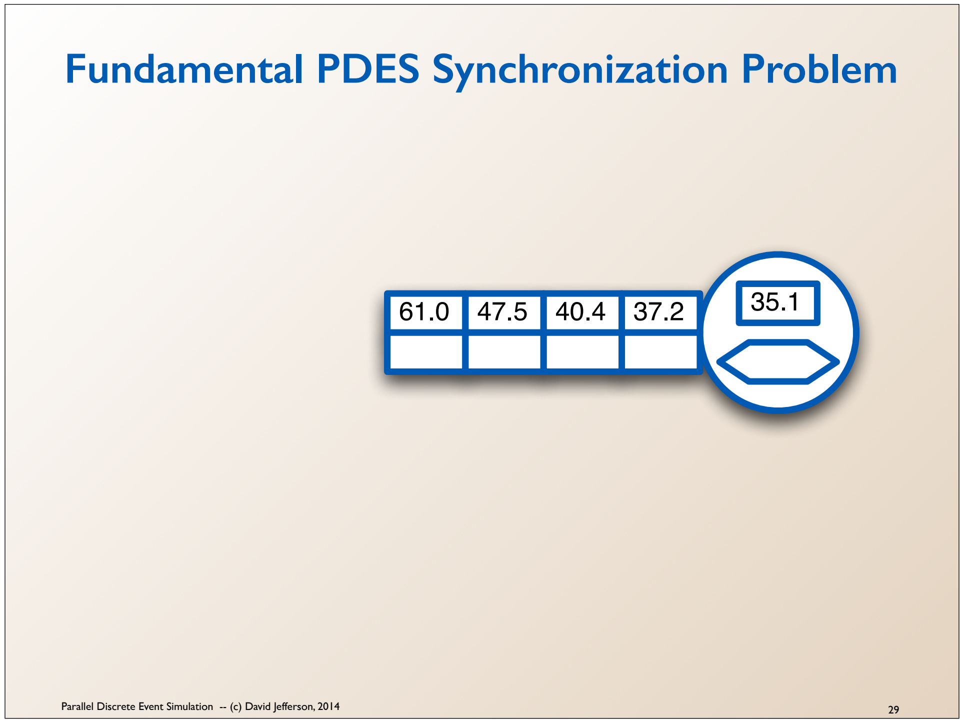
In the far right on the time line labeled ALL is the projection of all of the events from all of the objects. As you can see, there are almost no ties in the simulation. This is typical for discrete event simulations, especially when the times at which events occur are calculated probabilistically.

If the parallel simulation rule is that all events at times < t must finish executing before an event at time t, then this diagram show that there would be essentially no parallelism.

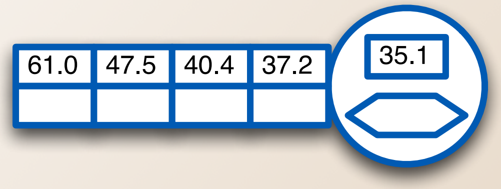
We must give up the idea that with parallel simulation events must be executed in strict increasing time order.



Fundamental Synchronization Problem of PDES



Fundamental PDES Synchronization Problem



Parallel Discrete Event Simulation -- (c) David Jefferson, 2014

29

Look at one single object on its processor node

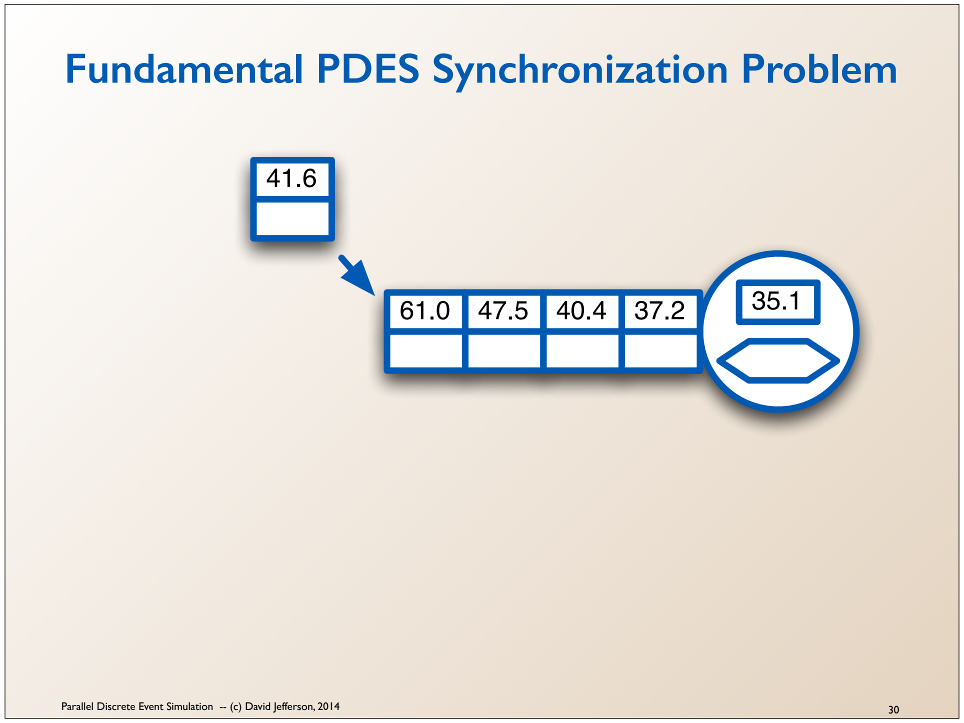
Event messages arrive asynchronously from other objects on other nodes.

Some of those objects are ahead in simulation time and some are behind.

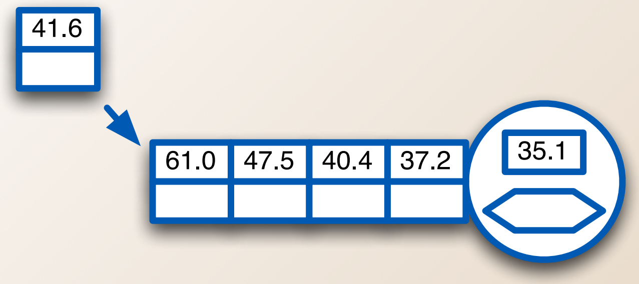
Event messages do not necessarily arrive in increasing timestamp order, but they are inserted in the queue and sorted by timestamp

Of course all events must be executed in strictly increasing order.

Based on local knowledge only, what event should be executed next???



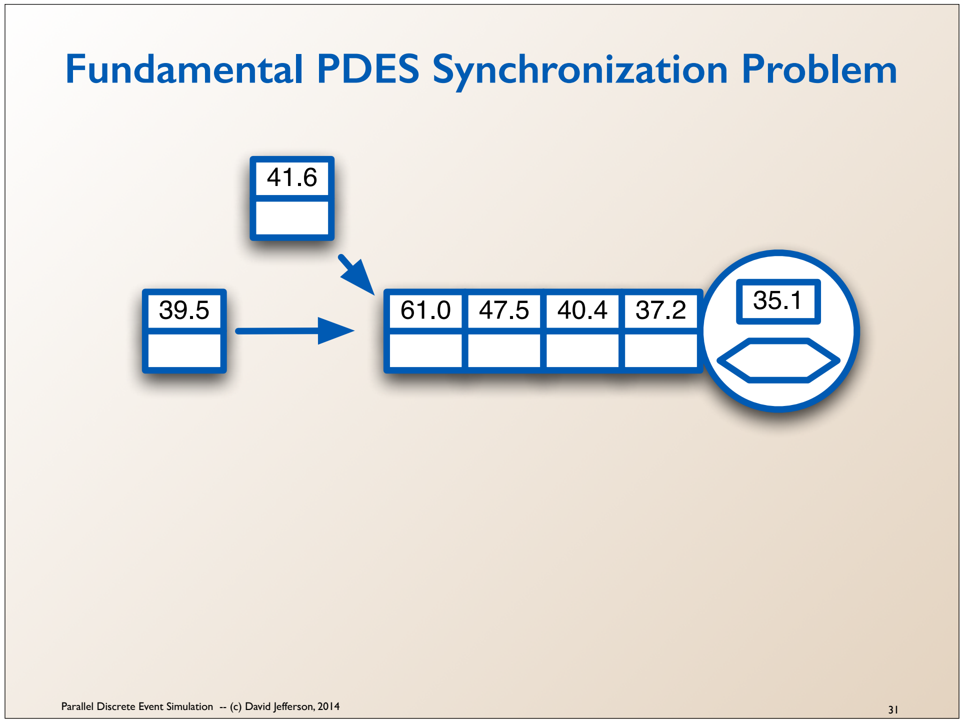
Fundamental PDES Synchronization Problem



This object MUST execute its event messages in strictly increasing time order

So after finishing event 35.2, what is the next event message its should execute?

Considering that 41.6 is on the way, it might appear that it is safe to execute 37.2 and 40.4

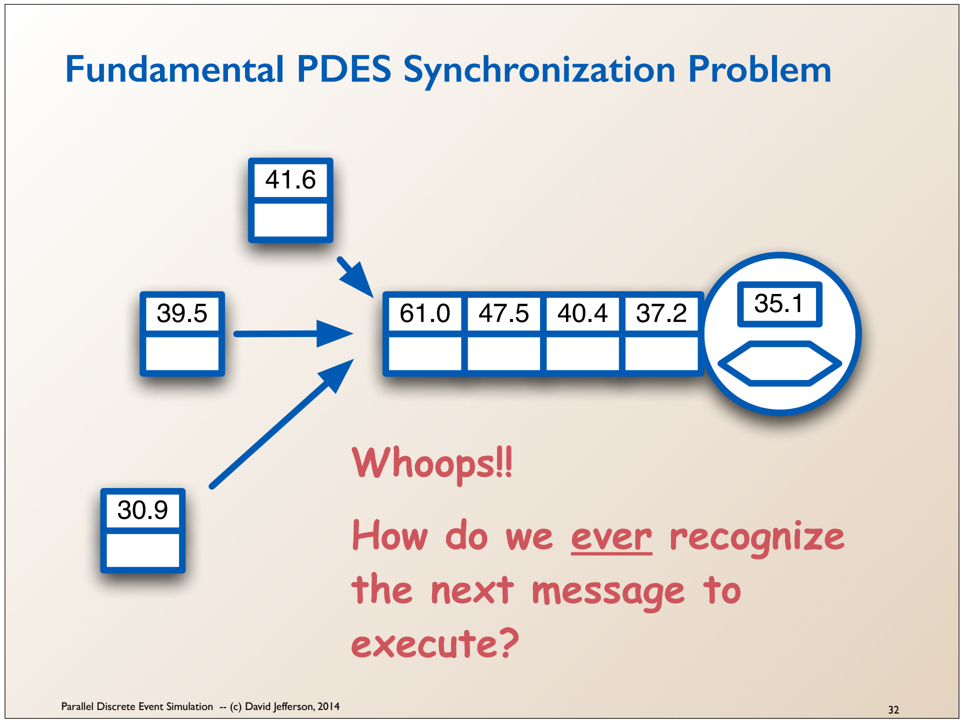


Fundamental PDES Synchronization Problem

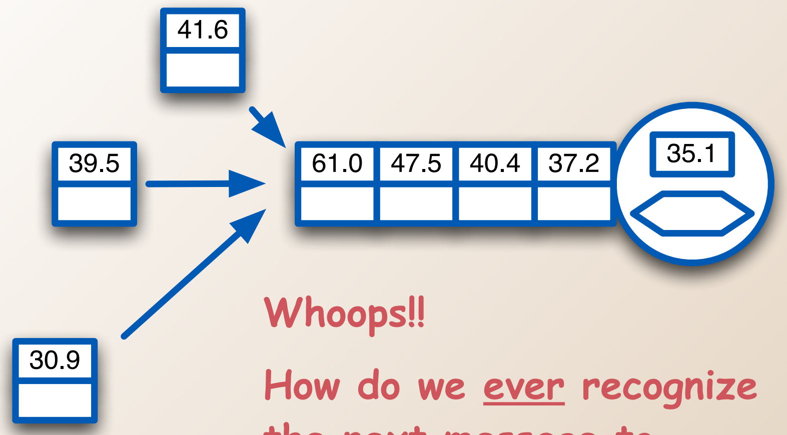
Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 31

But no, it cannot execute 40.4 because unbeknownst to this process, a message with time stamp 39.5 is on the way!

But at least we can execute 37.2 next!



Fundamental PDES Synchronization Problem



Whoops!!

How do we *ever* recognize the next message to execute?

Oh, but wait! We cannot execute 37.2, and in fact, we should not have executed 35.1, because a message with timestamp 30.9 is going to arrive!

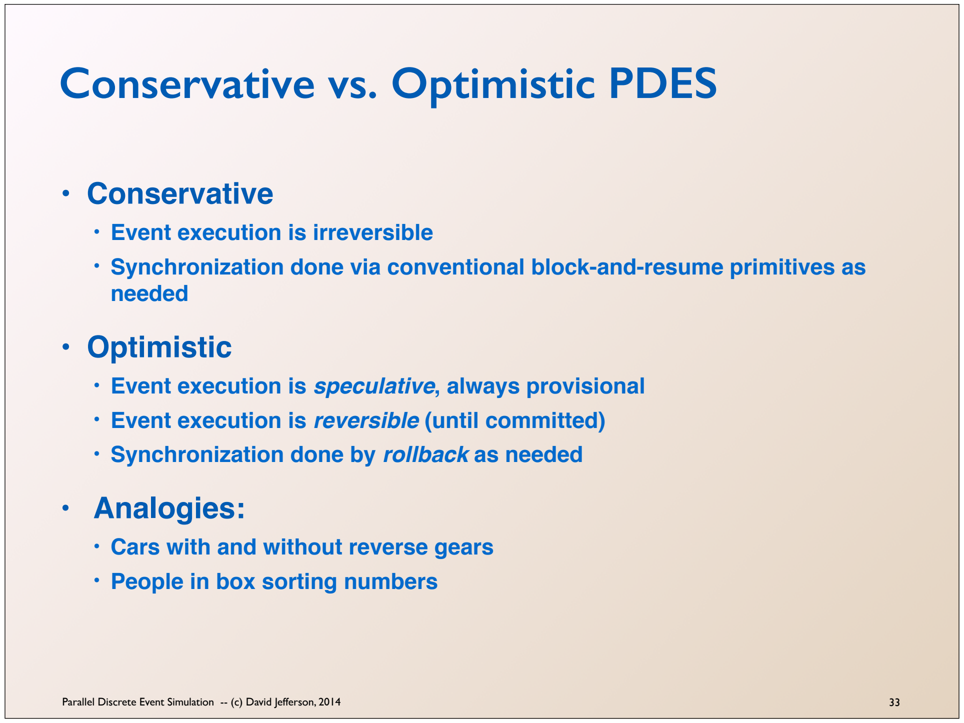
How can you even recognize the next event to be executed????

When can you execute any events at all? And how many? That is the fundamental problem?

Two broad kinds of solutions:

Irreversible events —> conservative synchronization
Never execute events until it is guaranteed "safe" to do so
Requires additional nonlocal information to determine when it is safe too execute events and not risk executing them out of order (causality violation)

Reversible events —> optimistic synchronization
Execute events speculatively; roll back (reverse) the execution of events when necessary to execute a late-arriving event message with an earlier timestamp, the re-execute the events that were rolled back.



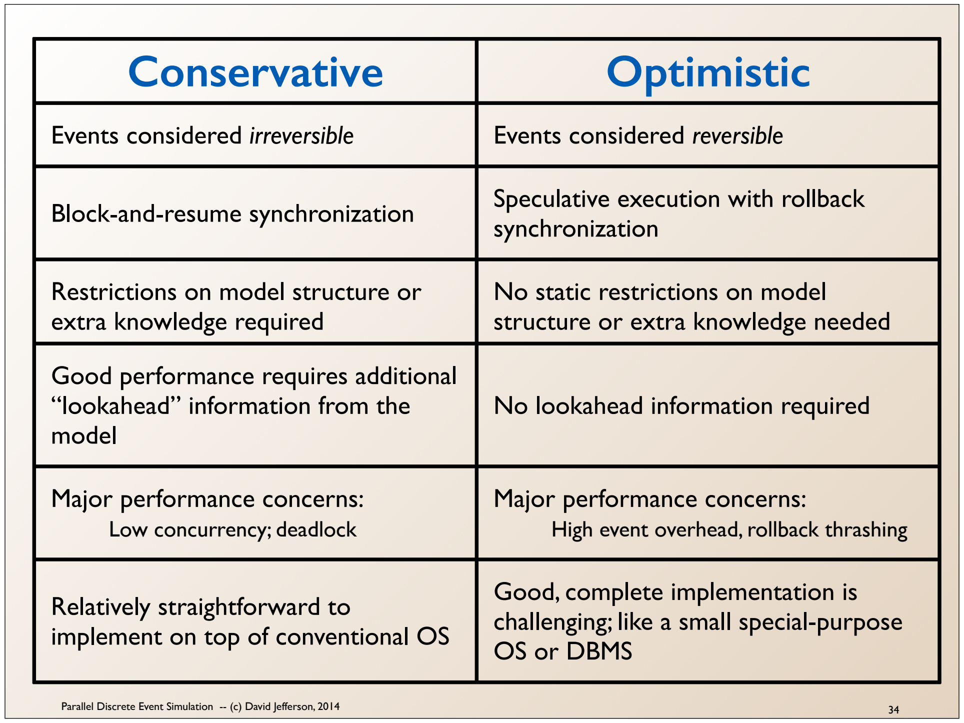
Conservative vs. Optimistic PDES

- Conservative
  - Event execution is irreversible
  - Synchronization done via conventional block-and-resume primitives as needed
- Optimistic
  - Event execution is *speculative*, always provisional
  - Event execution is *reversible* (until committed)
  - Synchronization done by *rollback* as needed
- Analogies:
  - Cars with and without reverse gears
  - People in box sorting numbers

Parallel Discrete Event Simulation -- (c) David Jefferson, 2014 33

Conservative synchronization might be compared to a country in which cars have forward gears and brakes, but no reverse gears. Think of the trouble that causes: head-in parking, home driveways and garages, parallel parking. Think of deadlocks on narrow roads, bridges and tunnels, or Manhattan traffic.

Optimistic synchronization adds that reverse gear. You might not drive 1 mile out of 10,000 in reverse gear, but having it make a huge difference in the flexibility of the car, and the fleet of cars.



Conservative	Optimistic
Events considered *irreversible*	Events considered *reversible*
Block-and-resume synchronization	Speculative execution with rollback synchronization
Restrictions on model structure or extra knowledge required	No static restrictions on model structure or extra knowledge needed
Good performance requires additional “lookahead” information from the model	No lookahead information required
Major performance concerns: Low concurrency; deadlock	Major performance concerns: High event overhead, rollback thrashing
Relatively straightforward to implement on top of conventional OS	Good, complete implementation is challenging; like a small special-purpose OS or DBMS