Parallel Discrete Event Simulation Course

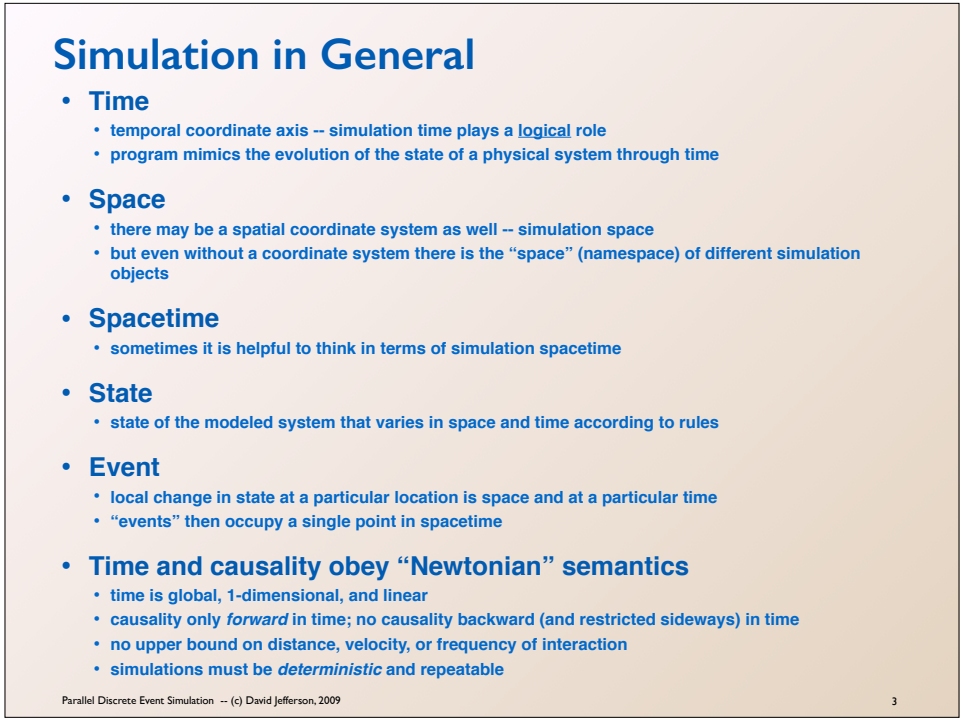
#1

David Jefferson
Lawrence Livermore National Laboratory
2014

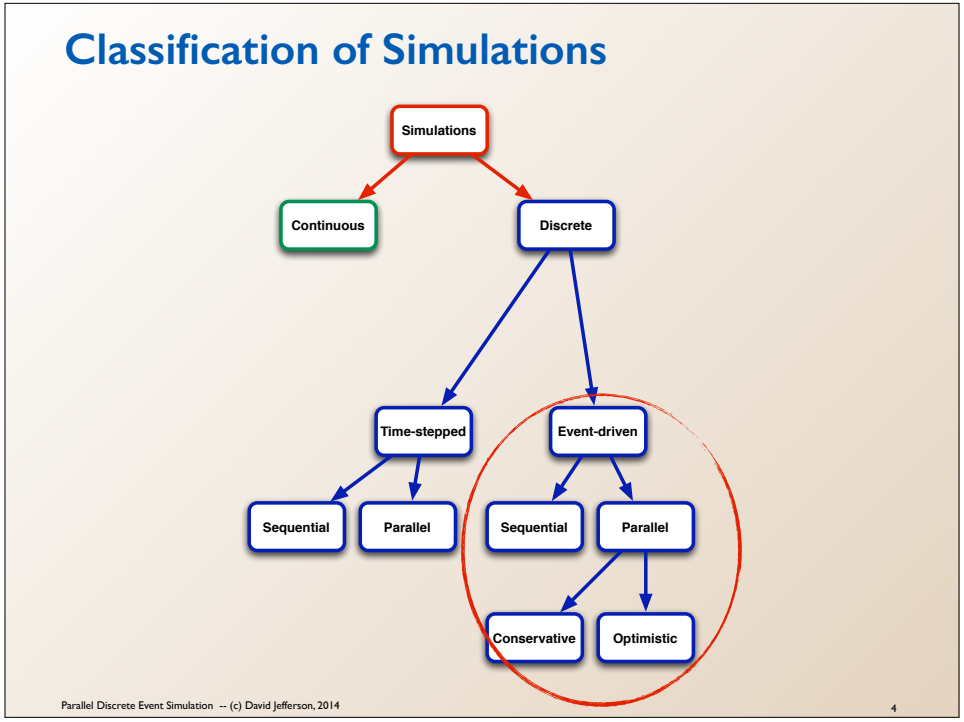
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Release Number: LLNL-PRES-648682

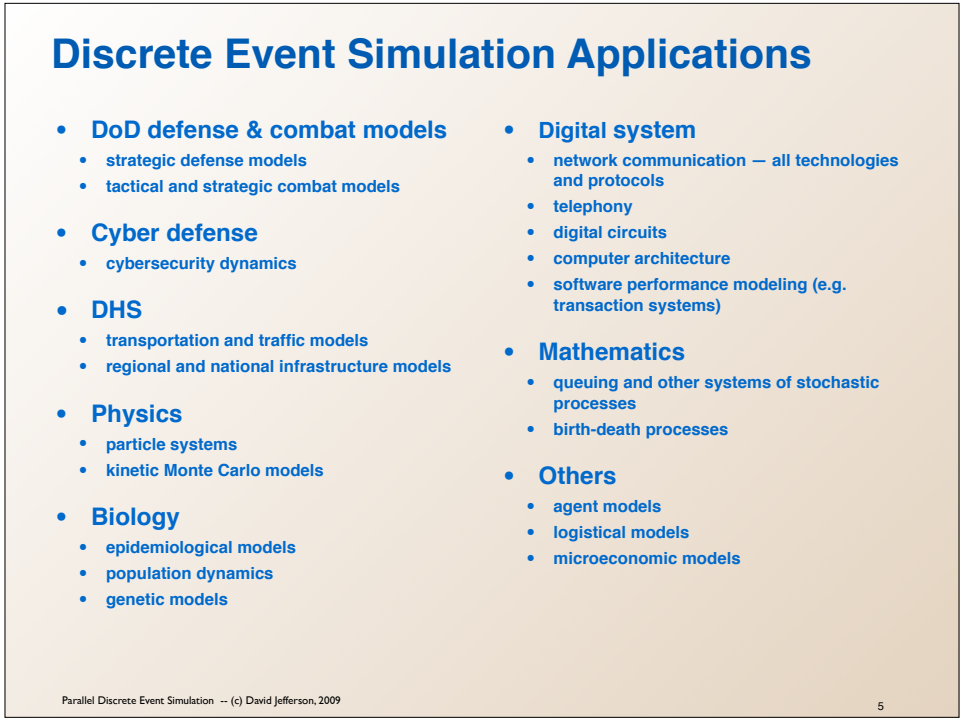
Overview



- **What does it mean for space and time to play a *logical role***
  - It is just as the stack discipline plays a logical role in a procedural program
    - SimTime is global, i.e. Newtonian
    - SimTime is read-only to the model code
    - Like the stack pointer in a procedural language, simulation time is not an ordinary variable
      - Stack pointer is controlled by the compiler or interpreter, and the procedural programmer cannot escape them or violate the discipline
      - SimTime controlled by the simulator runtime system (or compiler, interpreter, etc.) but model cannot escape SimTime discipline
    - Time and space coordinates are not under the complete control of the simulation programmer
      - They are controlled by the *simulator*, the simulation *platform* code
      - The simulation programmer cannot escape or violate their causal discipline
  - time is one dimensional and linearly ordered
    - no branching time or other exotic notion of time
  - sim time never decreases
  - sim time must be consistent with Newtonian causality in the model
    - no upper bound on the "velocity" of causal effects, although always finite
    - no causal behavior backward in time
  - sim time is virtually always numeric, with sum and difference operations defined
    - this is not actually required for simulation algorithms themselves--linear ordering is all that is required
    - but sums and differences are generally required for simulation models that run on them
  - simulation time is ***the*** synchronization standard used in a PDES (later extended to *virtual time*)



The circled categories are what we will study in this course.



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

Combat models

objects: units, ships, aircraft, tanks, weapons, communication devices, etc.

events: moving, sensing, firing, exploding, message transmission, 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, resources, locations

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.

events: financial transactions, demand requests, supply advertisements

History of Discrete Event Simulation

- **Origin of discrete event simulation concept and the sequential algorithm for it is hazy**
  - **Early ideas late 1950s**
  - **No specific person is credited that I know of.**
  - **But then, who is credited with the idea of a runtime stack?**
- **Dozens of programming languages designed specifically for discrete event simulation**
  - **GASP, SimScript, GPSS, Simula I, Simula 67, ModSim, many others**
  - **Nance, Robert -- "History of Discrete Event Simulation Programming Languages", SigPlan Notices, V.28, No.3, March 1993**
- **Contributions of discrete event simulation to computer science:**
  - **coroutines**
  - **object-oriented programming (classes, instances, inheritance)**
  - **reverse computation (!)**

Parallel Discrete Event Simulation -- (c) David Jefferson, 2009 6

For reference on discrete event simulation programming language history see

Nance, Robert -- "History of Discrete Event Simulation Programming Languages", SigPlan Notices, V. 28, No. 3, March 1993

Coroutines

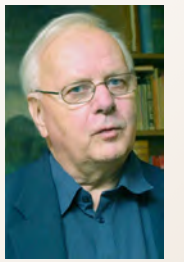
- Besides event lists, a key invention introduced in Simula was the notion of a coroutines, a sequential symmetric context switching mechanism (as opposed to subroutine, which is asymmetric and stack-oriented) that could be compiled at user level with no operating system support
- Coroutines were reminiscent of process switching in time sharing systems, but application directed, not time sliced, and with no change in protection domain.
- They are actually closer to mechanically to user level threads, but the distinction between processes and threads had not been made at the time

  coroutines each have their own stack

  coroutines are not preempted (as threads may be)

  when coroutines give up control, it is not to a scheduler, but to another named coroutine (half as many context switches)

Simula-67 is the origin of the Object-Oriented Programming paradigm



Kristen Nygaard

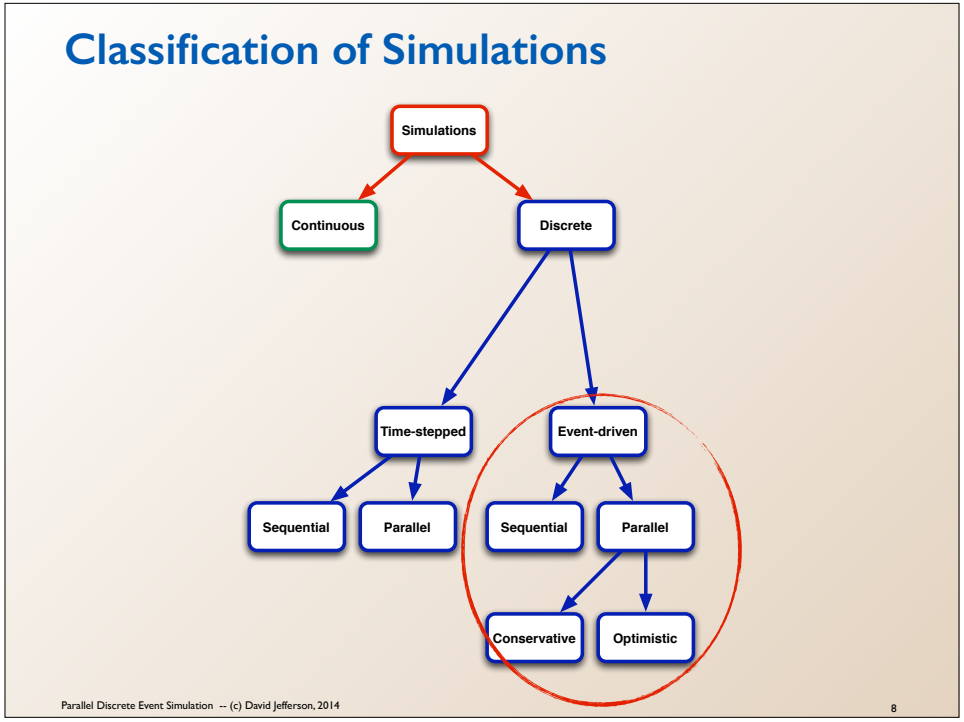
Ole-Johan Dahl

ACM Turing Award, 2001

Commander of the (Norwegian) Order of St. Olav, 2000

It is commonly believed that object-oriented computing originated with Alan Kay and SmallTalk. Not true. While SmallTalk coined the term "object oriented" and made O-O programming popular and more dynamic, the class concept (and the term) originated with Dahl and Nygaard a decade earlier with Simula-67. Furthermore they extended the notion of object-orientedness to include scheduling using simulation time.

Both died in 2002.



The circled categories are what we will study in this course.

Continuous vs. Discrete Simulation

Continuous vs. discrete simulation

Continuous	Discrete
original form: system of ODEs or PDEs	original form: state machines, particle systems, graphs, stochastic processes, etc.
time, space, and state continuous	time, space, and state either continuous or discrete
state varies continuously in time and space; occasional discontinuities	state changes mostly discontinuous; constant or simple analytical changes between
generally not probabilistic; statistics plays no role in a single run	frequently probabilistic (Monte Carlo); statistics is key analytical tool
numerical methods central; numerical analysis is key analytical tool	generally not numerical; numerical analysis plays little role
generally implemented as synchronous, SPMD, and conservative	generally implemented as asynchronous, MPMD, and conservative or optimistic

Continuous and discrete simulation have much in common, so this table summarizes the contrasts. There are VERY FEW who are real experts in both continuous and discrete simulation.

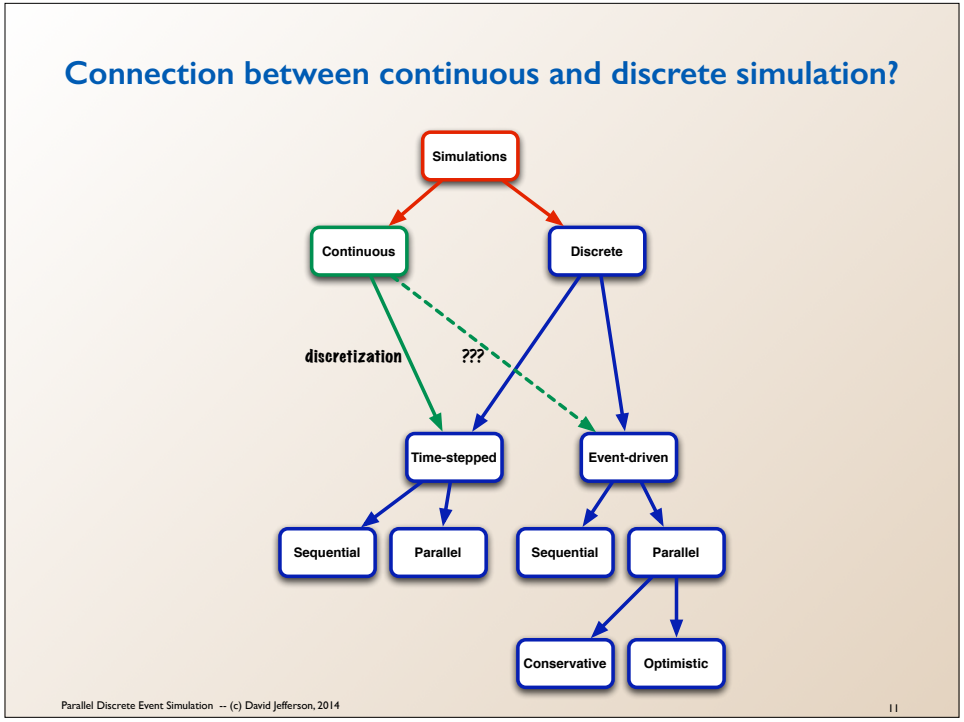
Why should we expect unification? Because either can approximate the other to an arbitrary degree of precision.

Mixed discrete and continuous systems:

fairly straightforward affair for mixed discrete and ODE-derived systems, but no one knows how to unify discrete and PDE-derived systems

Grand challenge: fully unify the computational methods for discrete models and multidimensional continuous models.

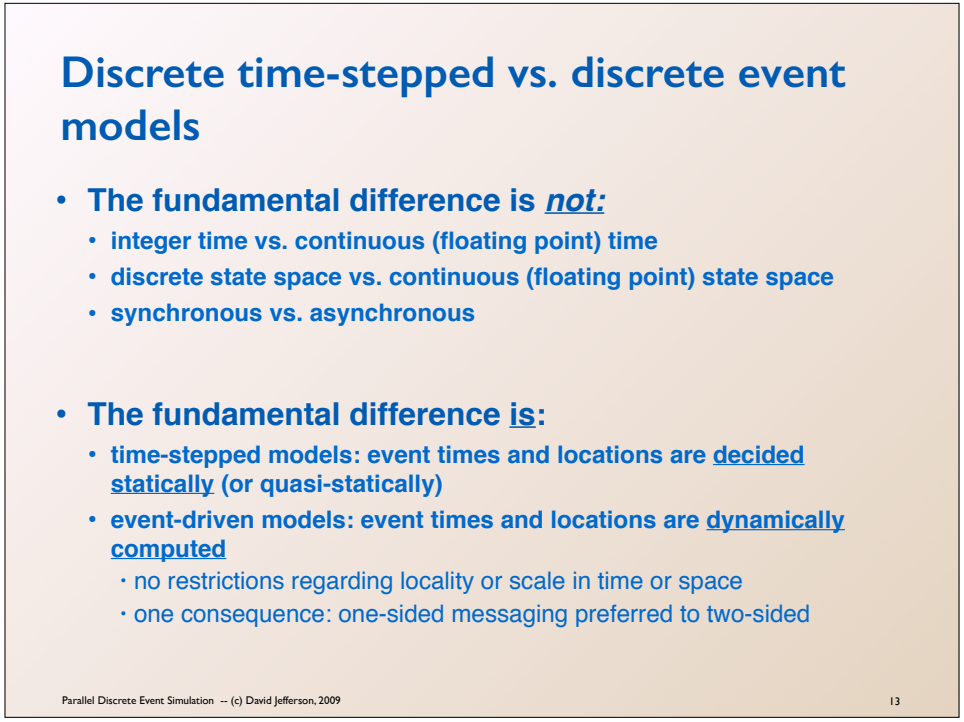
I have long felt that the world could benefit from a unification of the two, but so far no one has stood up to accomplish it. If anyone here is deeply knowledgeable about continuous simulation and wants to collaborate with me in a unification, let me know!



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

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.

Time-stepped vs. Event-driven



difference is analogous to other static vs. dynamic paradigms
    arrays vs. strings or lists
    static storage vs. stack or heap storage

With (quasi)static knowledge of the simtimes and communication patterns built-in to the code, it is possible for the receiving side of the code to know when and how many messages to expect at a given point in the logic. Hence 2-sided messaging with explicit RECEIVE() primitives are appropriate.

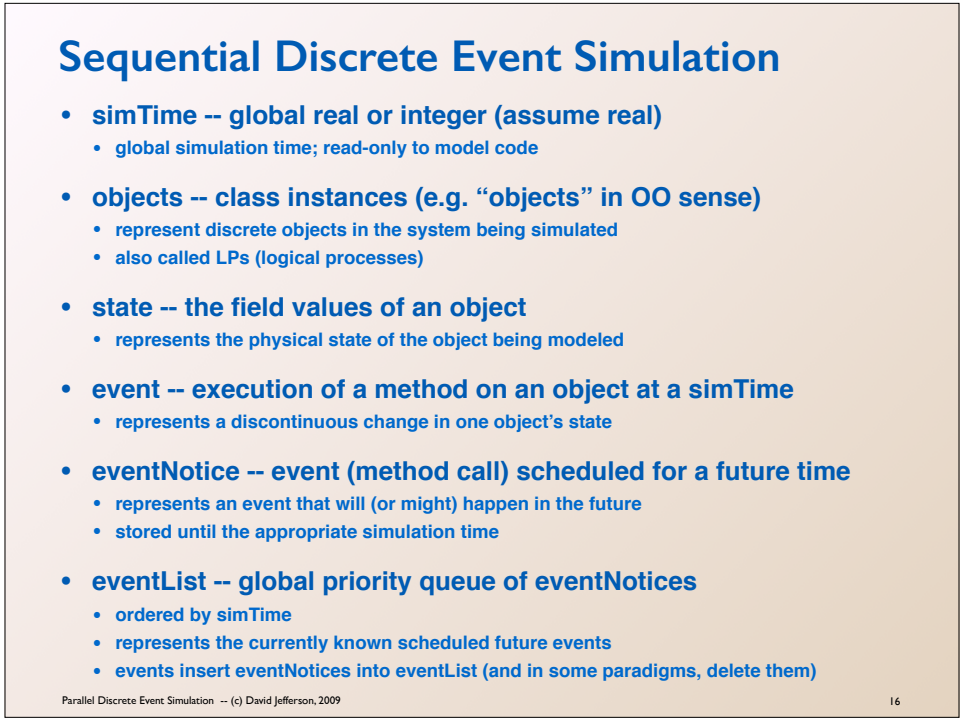
But in PDES the simulator has no static knowledge of the communication pattern or the times at which an object will receive an event message, it is impossible to know if or when a event message will arrive, or how many, so an explicit RECEIVE primitive is and any attempt to use two-sided message primitives requires systematic use of polling.

Time-stepped vs. event-driven simulation

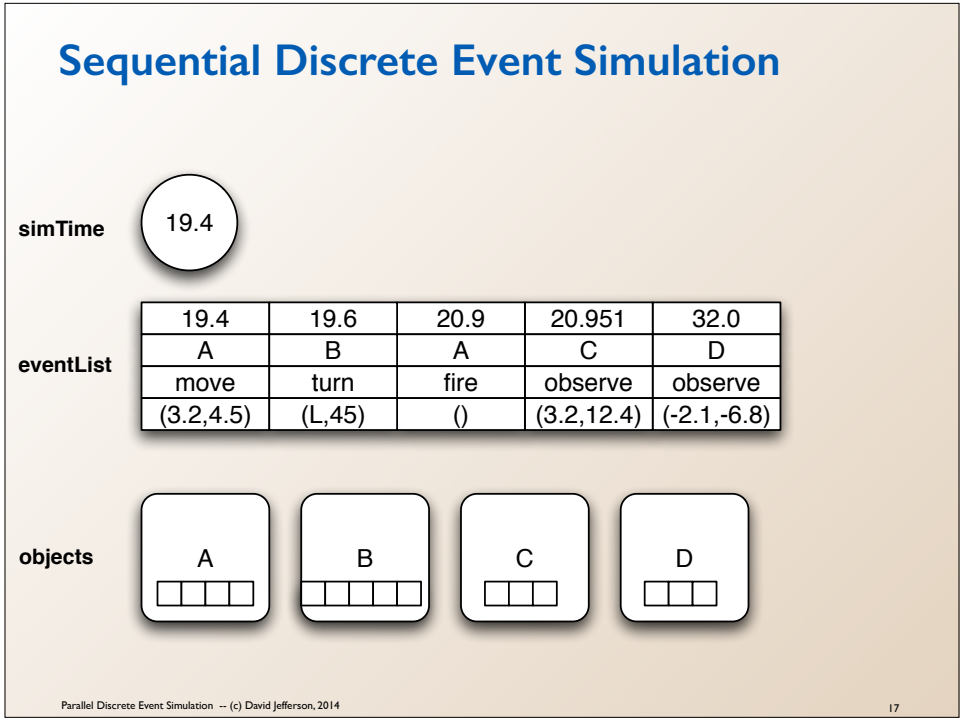
time-stepped	*event-driven*
event times statically chosen	event times dynamically computed
communication patterns mostly static	communication patterns dynamically computed
simple implementation	more complex implementation
static lower limit on time scale	no lower limit on time scale; inherently multiscale
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 and temporally regular models	appropriate for irregular, asynchronous models

- **time-stepped**
  - 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
  - SPMD
    - parallel time-stepped simulations need not, but usually do, run the exact same algorithm at every spatial point and are SPMD
- **event driven**
  - 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
  - MPMD
    - event driven simulations usually have more than one type of object, and are MPMD

Sequential Discrete Event Simulation



The event list is analogous to the runtime stack of procedural programming language. It is a priority queue instead of a stack, but like the stack it organizes the many method/event calls that make up the computation.



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 implemented as a linear list--it is a *priority queue*, generally a tree, i.e.
a heap, red-black tree, or (ideally) a splay tree

A is executing the move method at time 19.4

Sequential DES algorithm

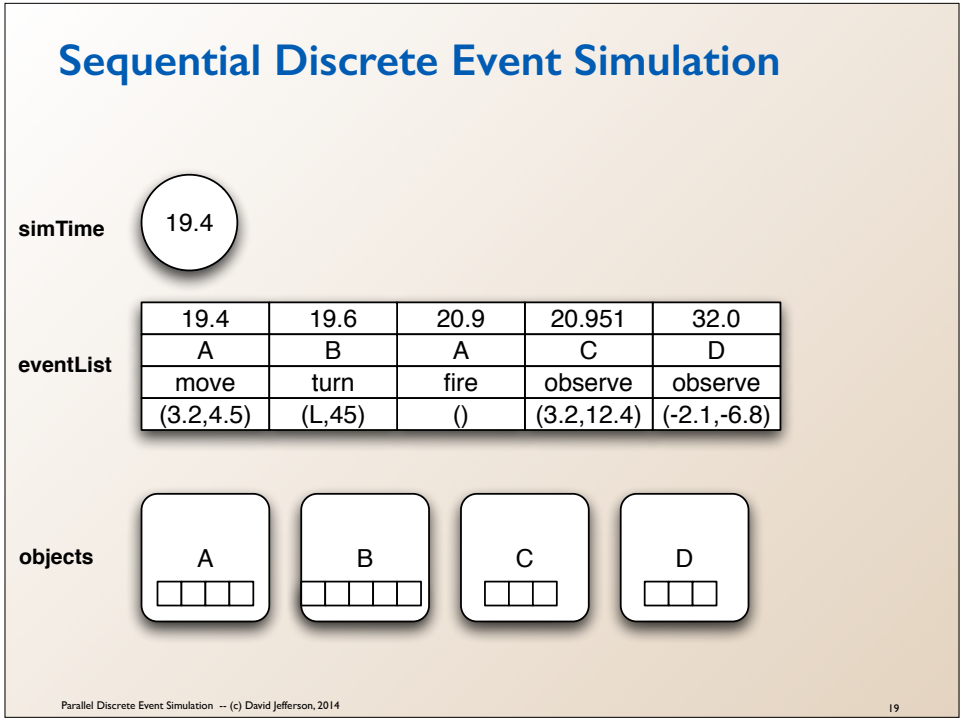
```
createInitialObjects();
eventList.insert(initialEvents);

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();
```


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 no ties and no zero-delay in simTime
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



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

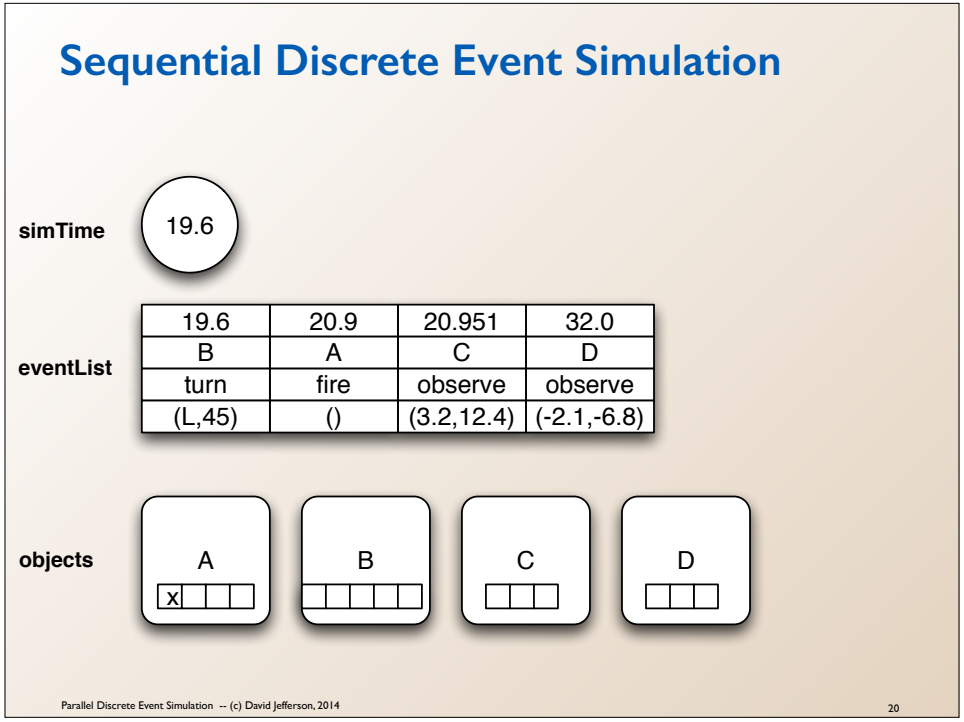
current simTime is 19.4

event list has 5 events scheduled in it

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

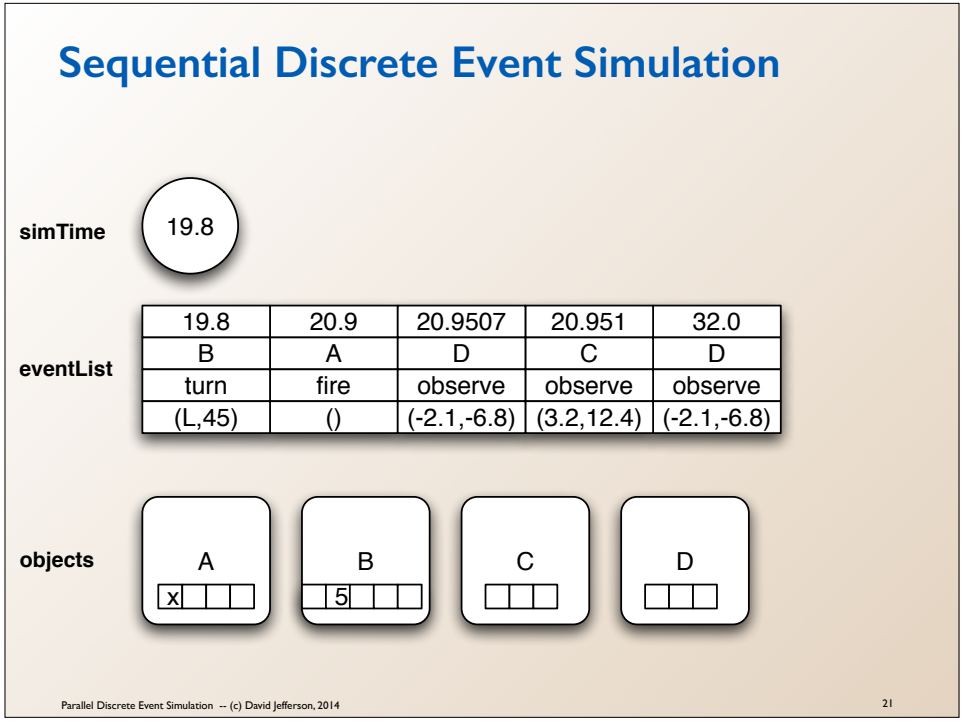
event list is not really implemented as a linear list--it is generally a tree, i.e.
a heap, red-black tree, or (ideally) splay tree

A is executing the move method



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

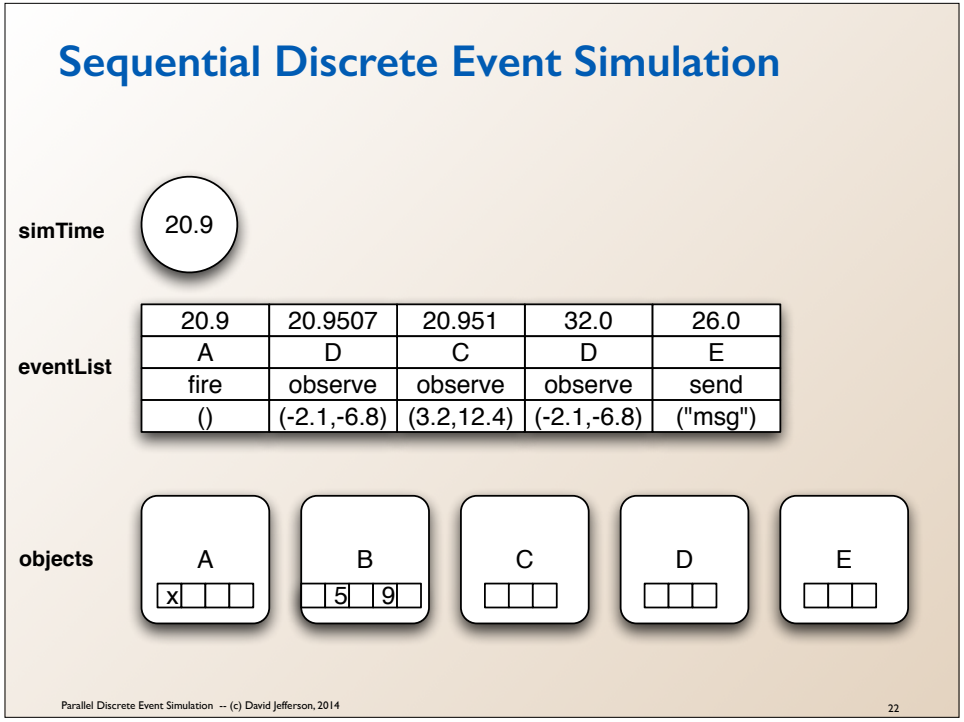


B's state has changed, but note that B inserted two events into the event list.

One it happened to be for itself, and happened to be the very next event.

It also inserted an event for D

Hence and event for B runs next also.



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

That's all there is to the core sequential algorithm! The main variations in this algorithm at the level we are talking about are in the data structures used for the event list!

But efficiently parallelizing this algorithm is a very complex undertaking and requires a lot of parallel computation issues to be rethought from the ground up, as we will see when we get to optimistic parallel algorithms.

That's all there is to the sequential DES algorithm!

- The most significant variations are in the implementation of the priority queue for the event list.
- This algorithm is optimal in both time and space!

Efficient, scalable parallelization of this one simple algorithm is what the rest of the course is about