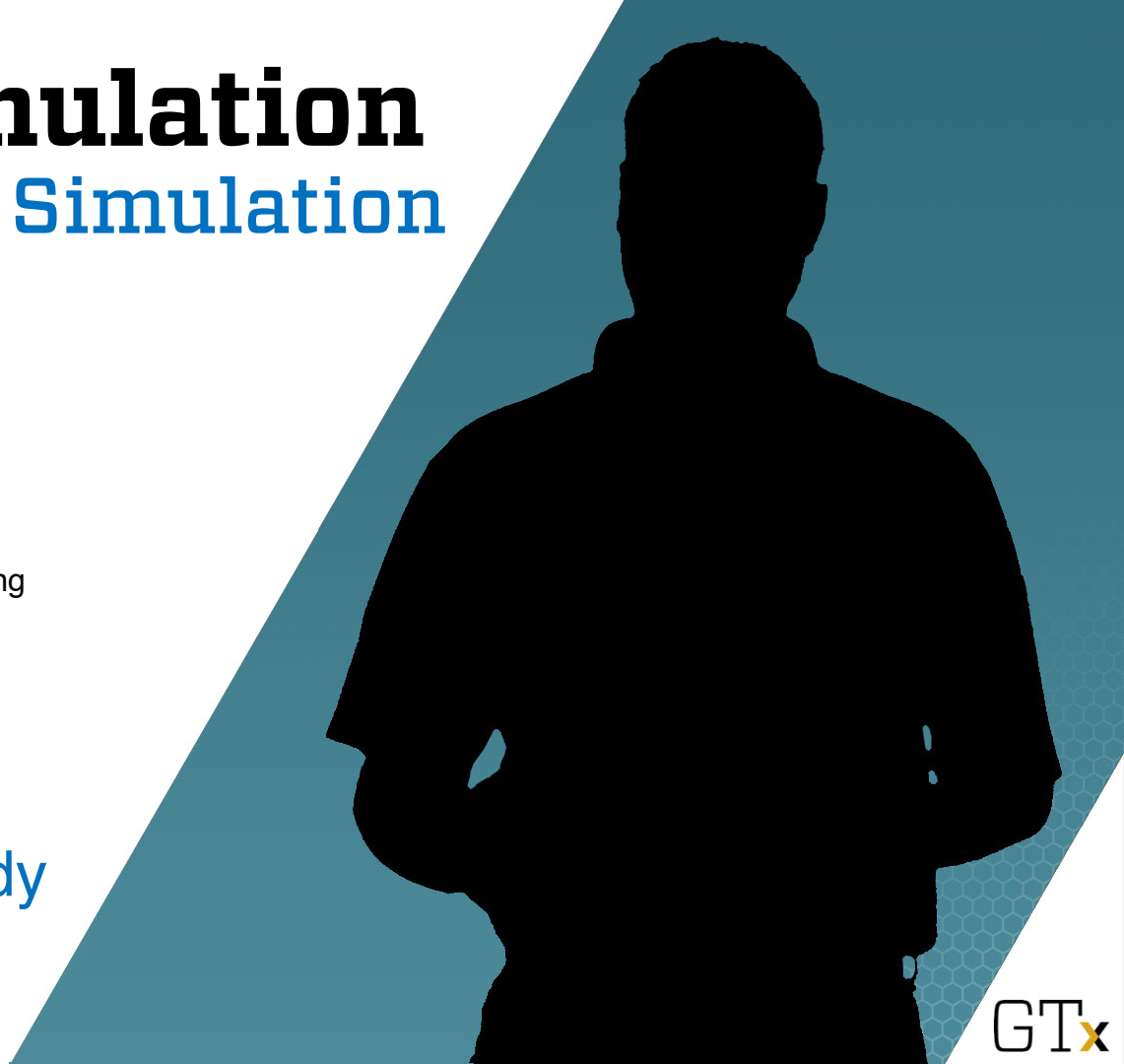
Computer Simulation

Module 4: General Simulation Principles

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Steps in a Simulation Study

GTx

Module Overview

Last Module: We covered hand and spreadsheet simulations.

This Module: We'll step back and look at the big picture – what makes a simulation tick?

Idea: How are simulations carried out and used? What's beneath the surface of a simulation program?

GTx

Module Overview

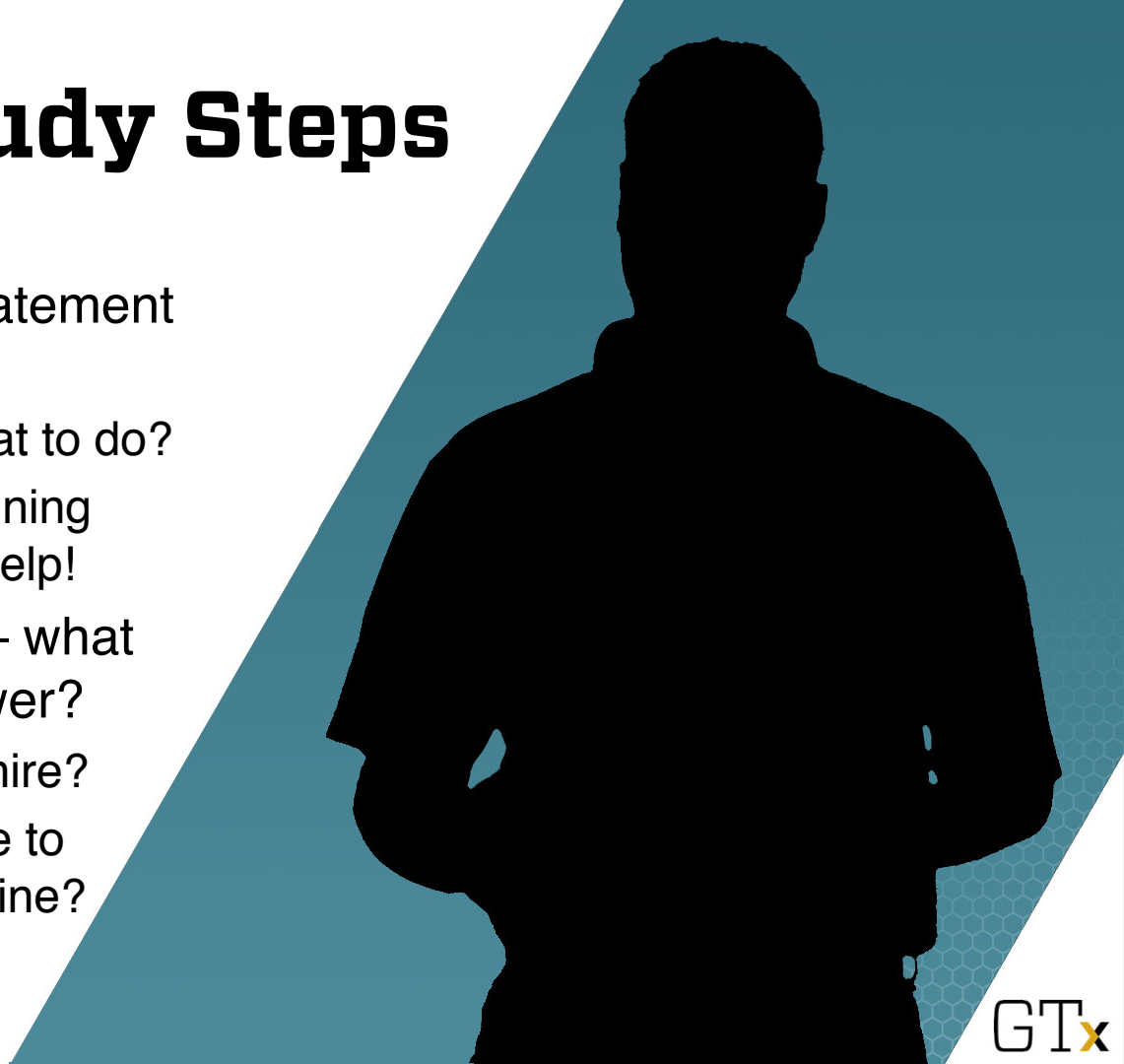
1. Steps in a simulation study
2. Some useful definitions
3. Simulation clock time-advance mechanisms
4. Two modeling approaches for simulations
5. Simulation languages

This Lesson: See what a simulation study involves (shoulda done this already!)

GTx

Simulation Study Steps

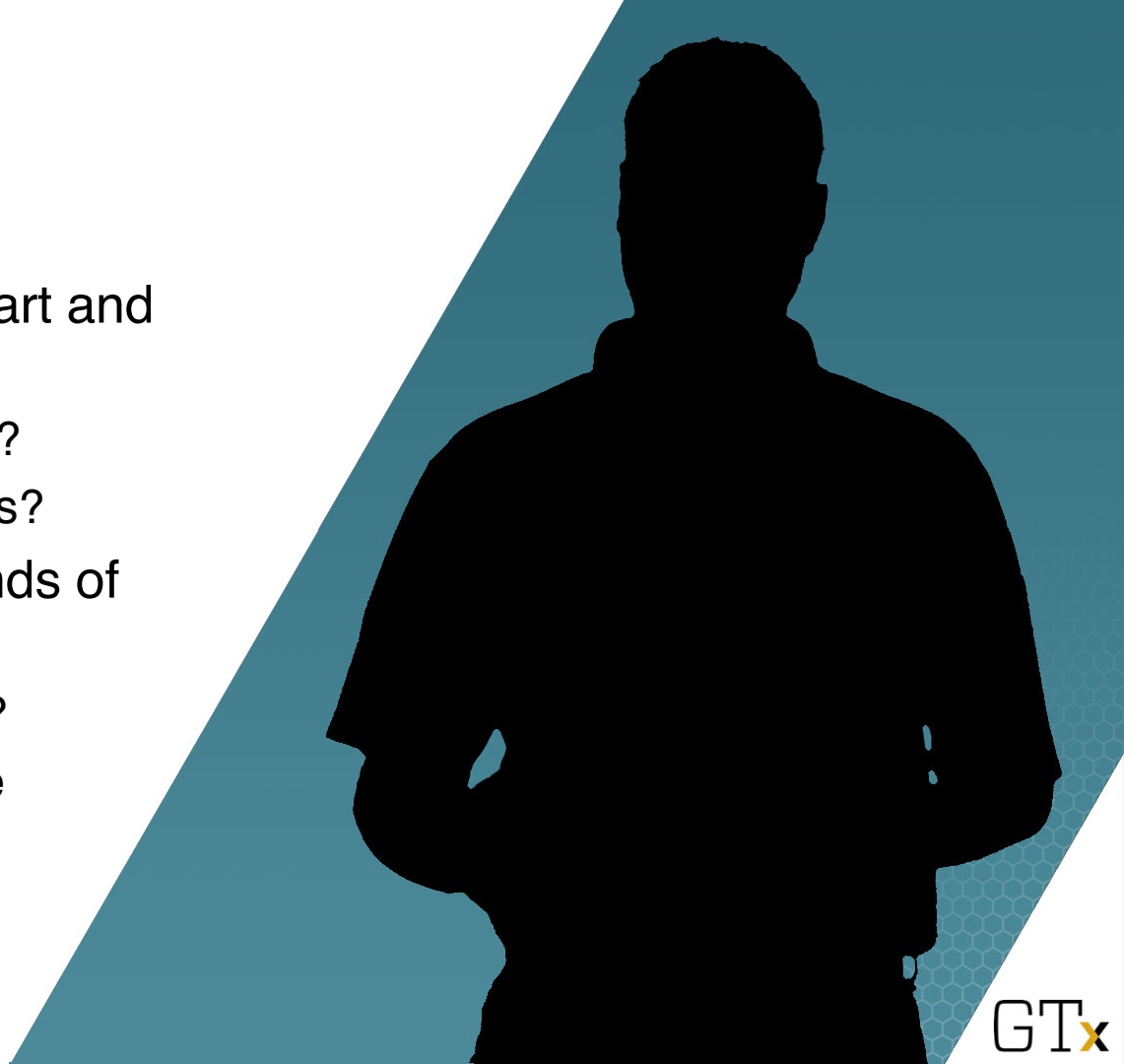
1. Problem Formulation – statement of problem.
    - Profits are too low. What to do?
    - Customers are complaining about the long lines. Help!
2. Objectives and Planning – what specific questions to answer?
    - How many workers to hire?
    - How much buffer space to insert in the assembly line?



GTx

More Steps

3. Model Building – both an art and science.
    - M/M/k queueing model?
    - Need physics equations?
4. Data Collection – what kinds of data, how much?
    - Continuous? Discrete?
    - What exactly should be collected?
    - Budget considerations.



GTx

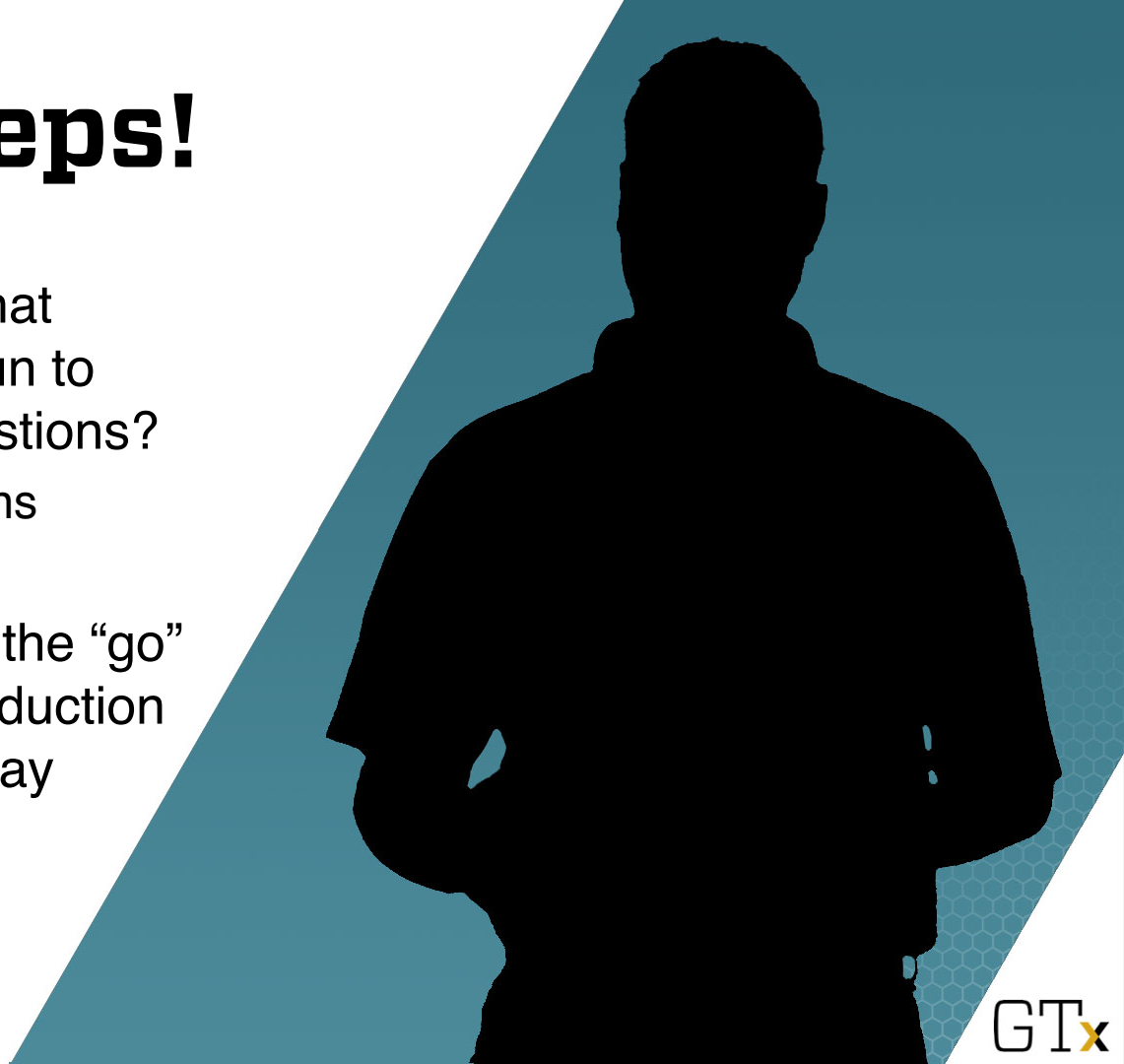
Even More Steps

5. Coding – decide on language, write program
    - Modeling paradigm
        - Event-Scheduling
        - Process-Interaction
    - 100s of languages out there
6. Verification – Is *code* OK? If not, go back to 5 (Coding).
7. Validation – Is *model* OK? If not, go back to 3 (Modeling) and 4 (Data Collection).

GTx

Still More Steps!

8. Experimental Design – what experiments need to be run to efficiently answer our questions?
    - Statistical considerations
    - Time / budget
9. Run Experiments – press the “go” button and make your production runs, often substantial. May require a lot of time.



Yet Still More Steps!

10. Output Analysis – statistical analysis, estimate relevant measures of performance
    - Often iterative with 8 (Experimental Design) and 9 (Production Runs)
    - Almost always need more runs
11. Make Reports, Implement, and Make Management Happy

like like like like like like like like like like like like like like like like

GTx

Summary

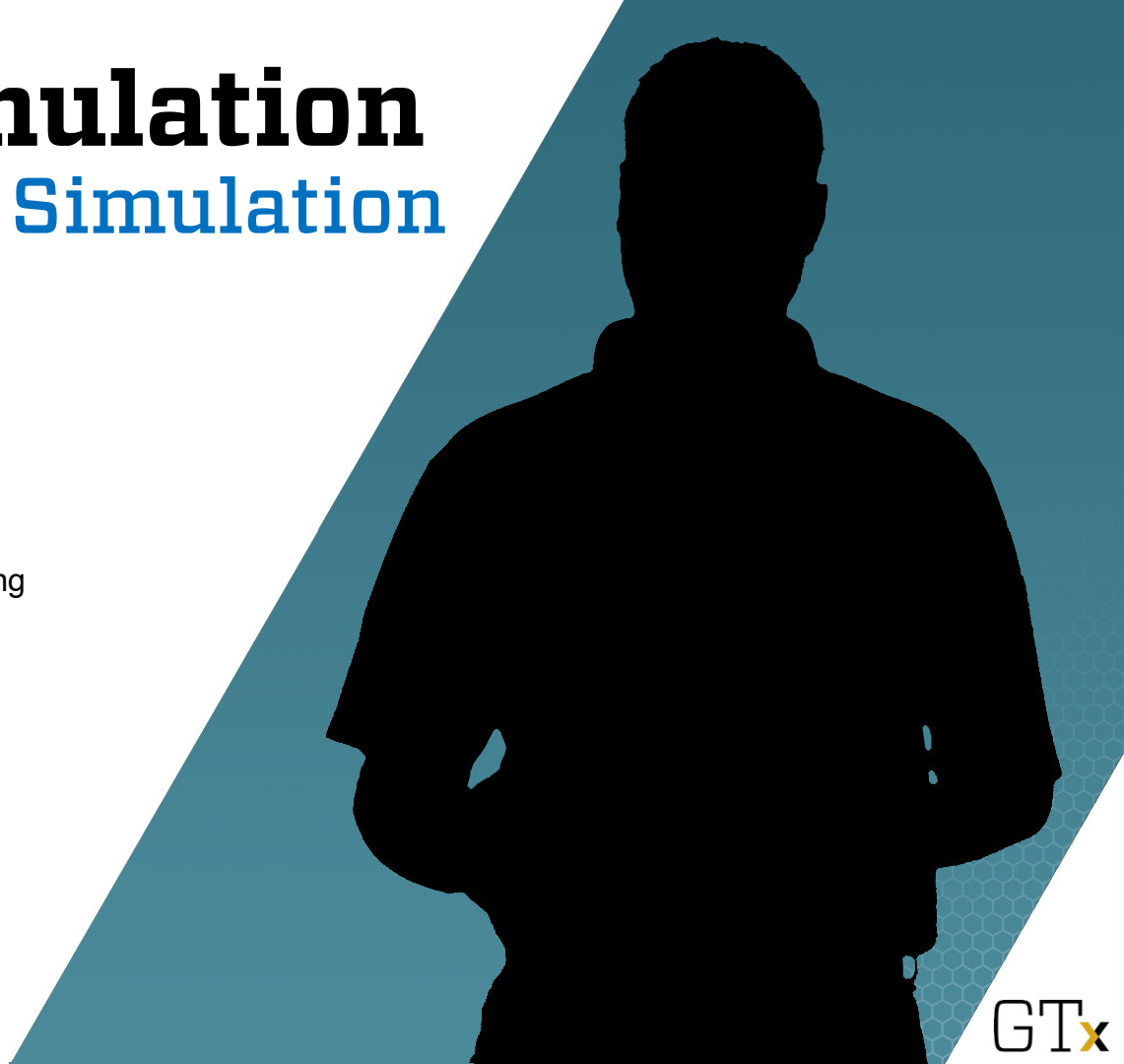
This Time: Discussed all of the steps in a proper simulation study. A bit of art, a bit of science.

Next Time: We'll go through a bunch of easy definitions that are relevant to general simulation models.

Computer Simulation

Module 4: General Simulation Principles

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Some Useful Definitions

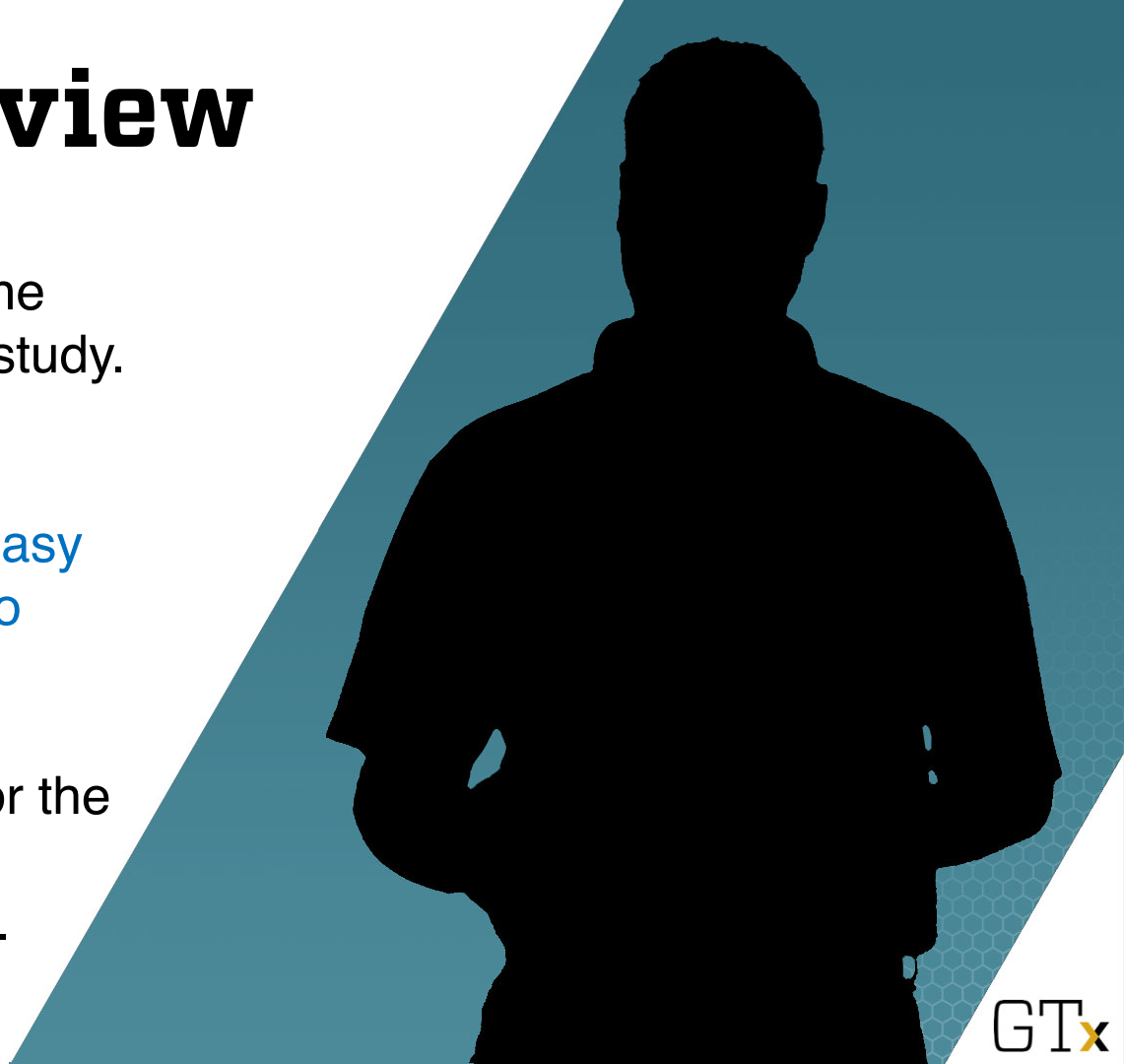
GTx

Lesson Overview

Last Time: Discussed all of the steps in a proper simulation study. An iterative process.

This Time: We'll give some easy definitions that are relevant to general simulation models.

We'll be using these terms for the remainder of the course, especially when we program.

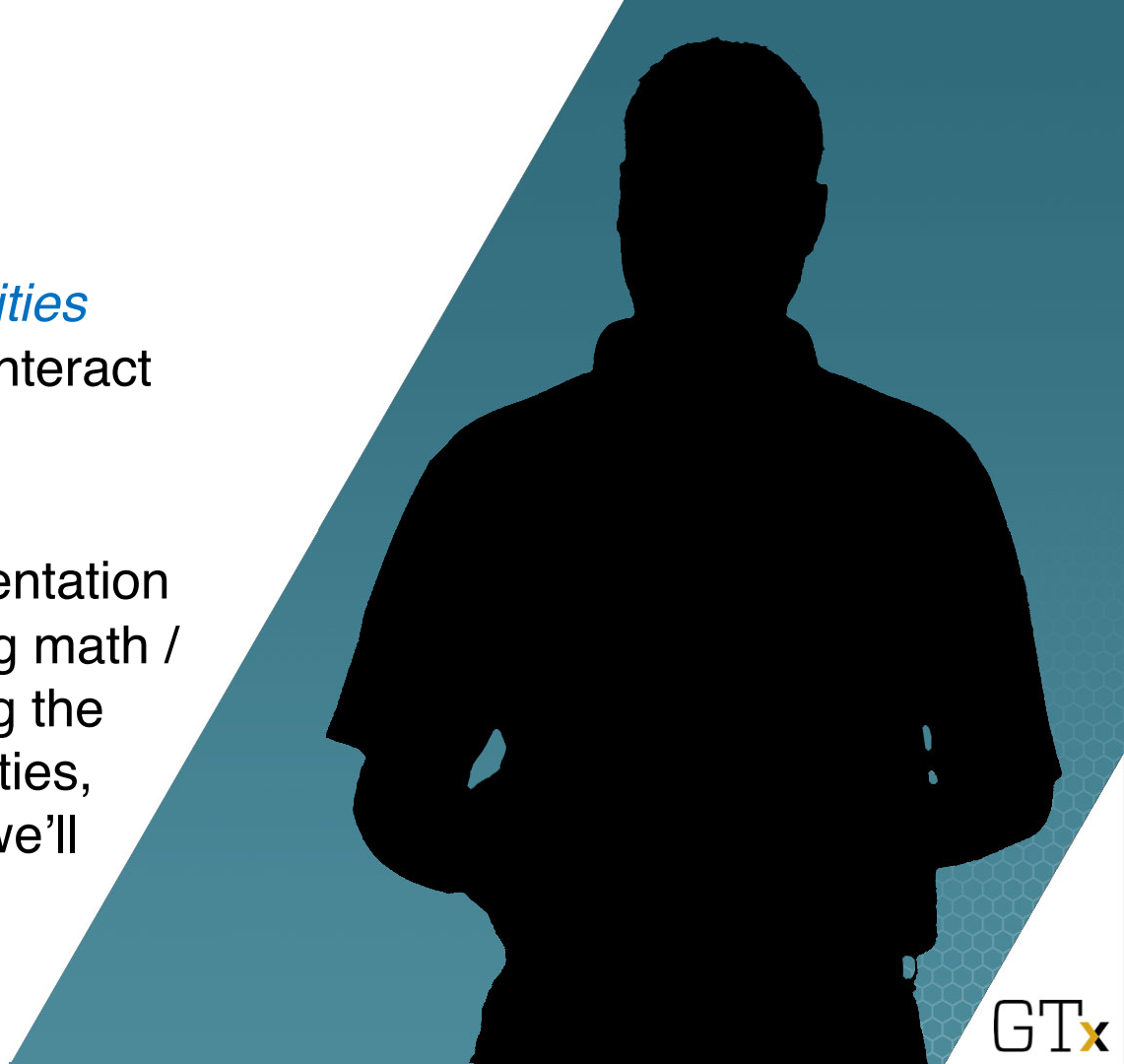


GTx

Definitions

A *system* is a collection of *entities* (people, machines, etc.) that interact together to accomplish a goal.

A *model* is an abstract representation of a system, usually containing math / logical relationships describing the system in terms of states, entities, sets, events, etc. (terms that we'll define below).



GTx

Definitions (cont'd)

System state: A set of variables that contains enough information to describe the system. Think of the state as a "snapshot" of the system.

E.g., in a single-server queue, all you might need to describe the state are:

$L_Q(t)$ = # of people in queue at time t

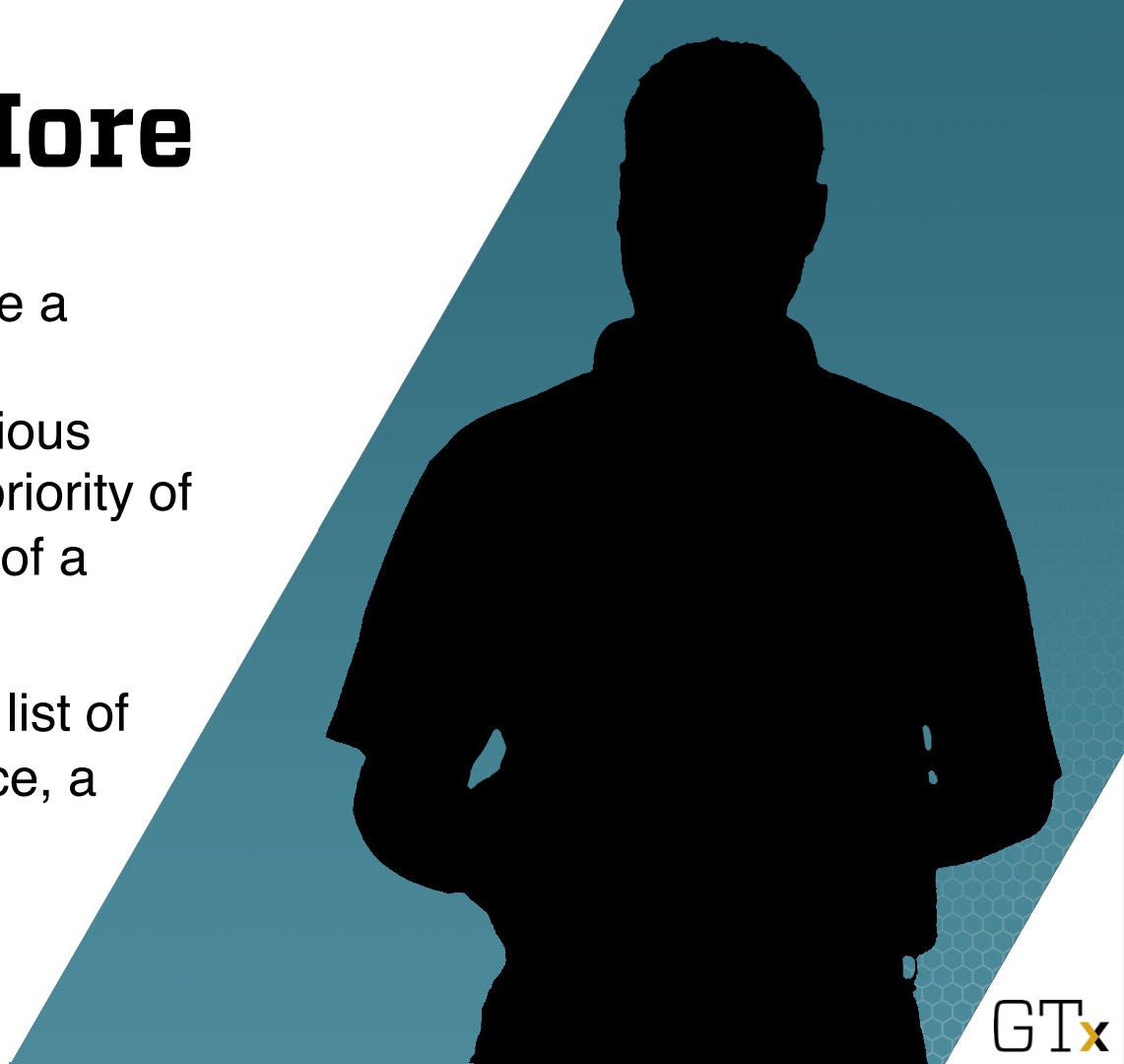
$B(t)$ = 1 [0] if server is busy [idle] at time t.

GTx

More More More

Entities can be permanent (like a machine) or temporary (e.g., customers), and can have various properties or *attributes* (e.g., priority of a customer or average speed of a server).

A *list* (or *queue*) is an ordered list of associated entities (for instance, a linked list, or a line of people).



GTx

I Gotta Have More!

An *event* is a point in time at which the system state changes (and which can't be predicted with certainty beforehand).

Examples: an arrival event, a departure event, a machine breakdown event.

"Event" technically means the time that a thing happens, but loosely refers to "what" happens (an arrival).

GTx

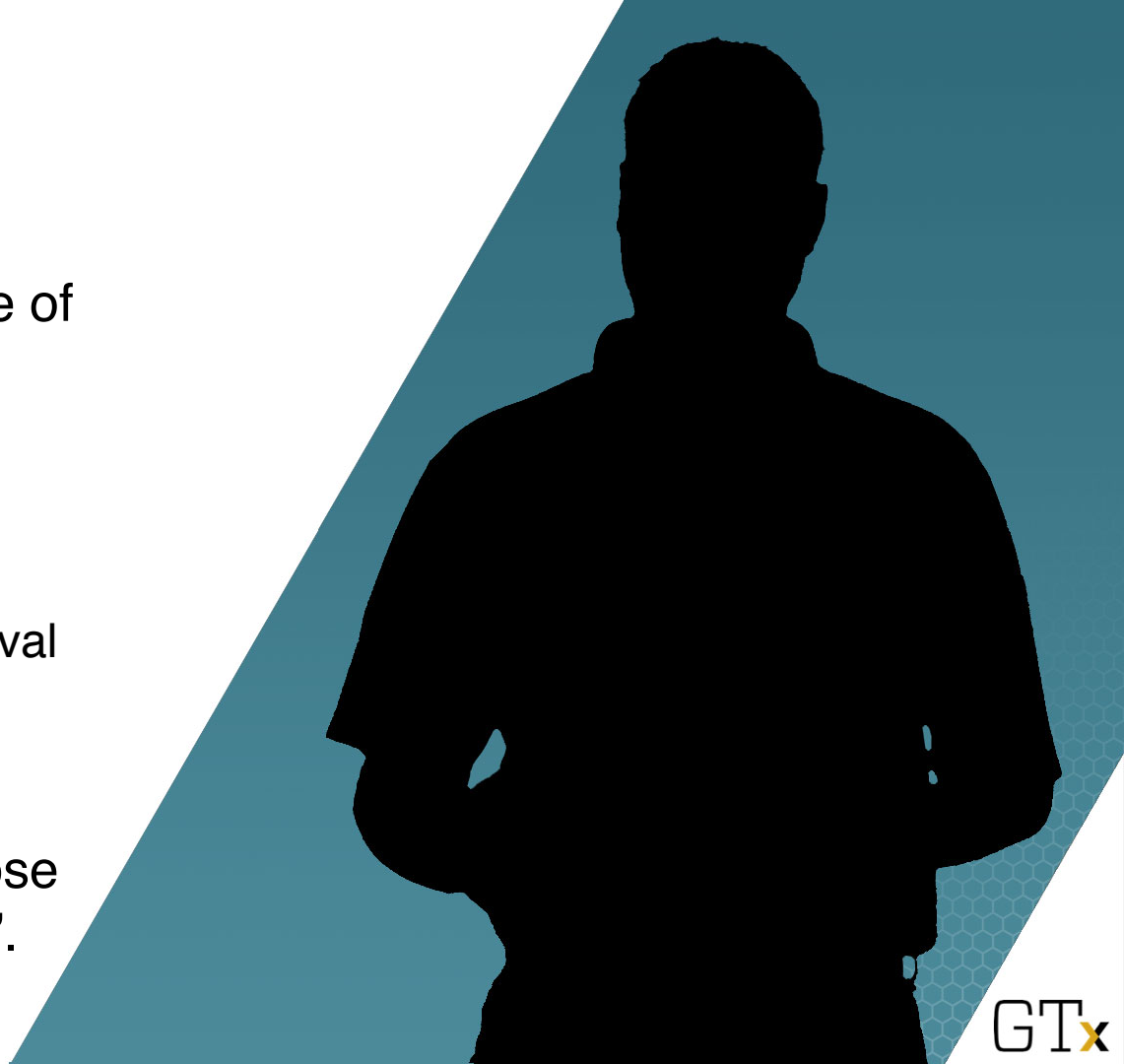
Not Done Yet

An *activity* is a duration of time of *specified length* (aka an *unconditional wait*).

Examples include:

- exponential customer interarrival times
- constant service times

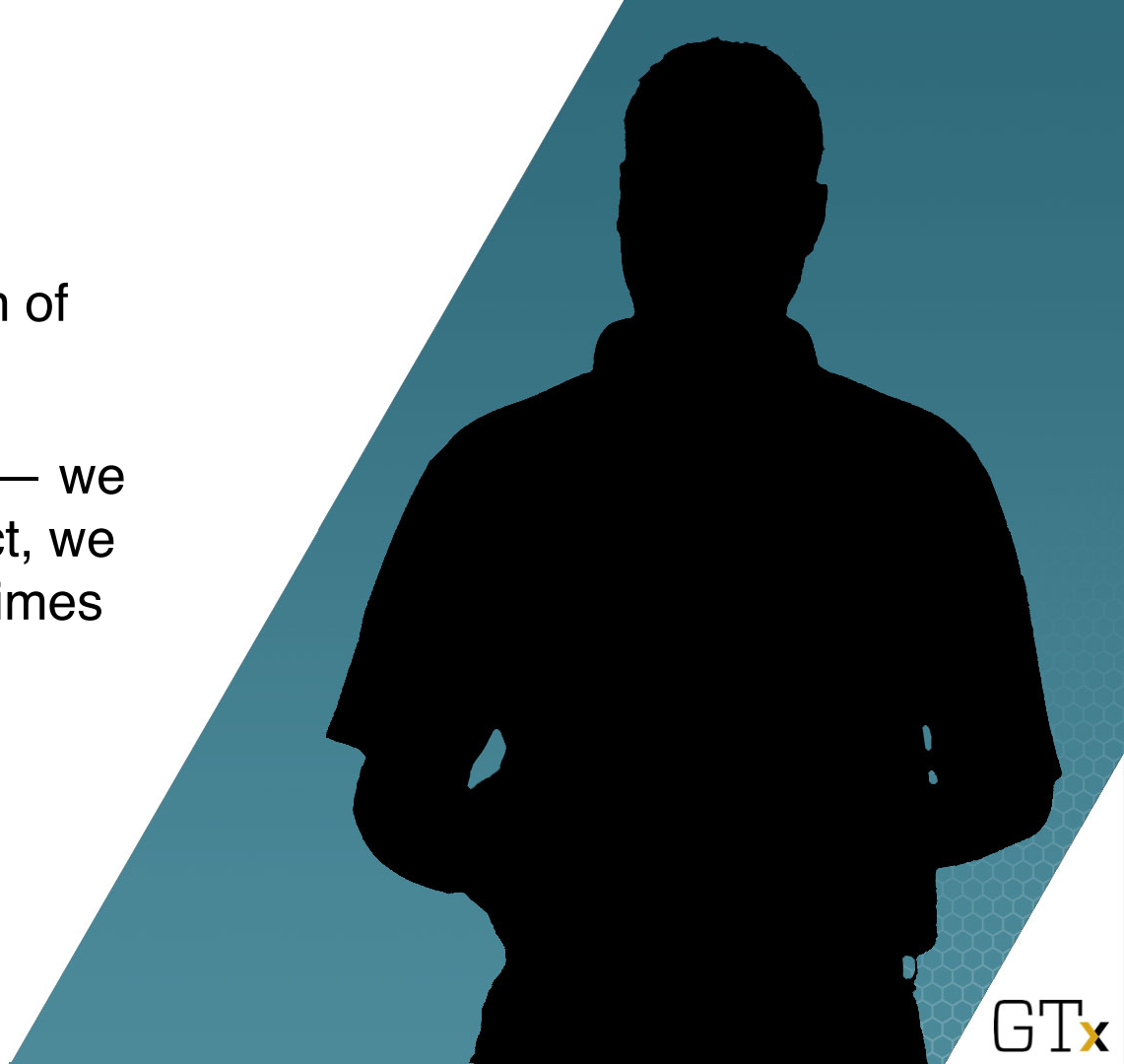
We can explicitly generate those events, so they are “specified”.



And Finally…

A *conditional wait* is a duration of time of *unspecified* length.

E.g., a customer waiting time — we don't know that directly. In fact, we just know arrival and service times and will use those to reverse-engineer the waiting times.

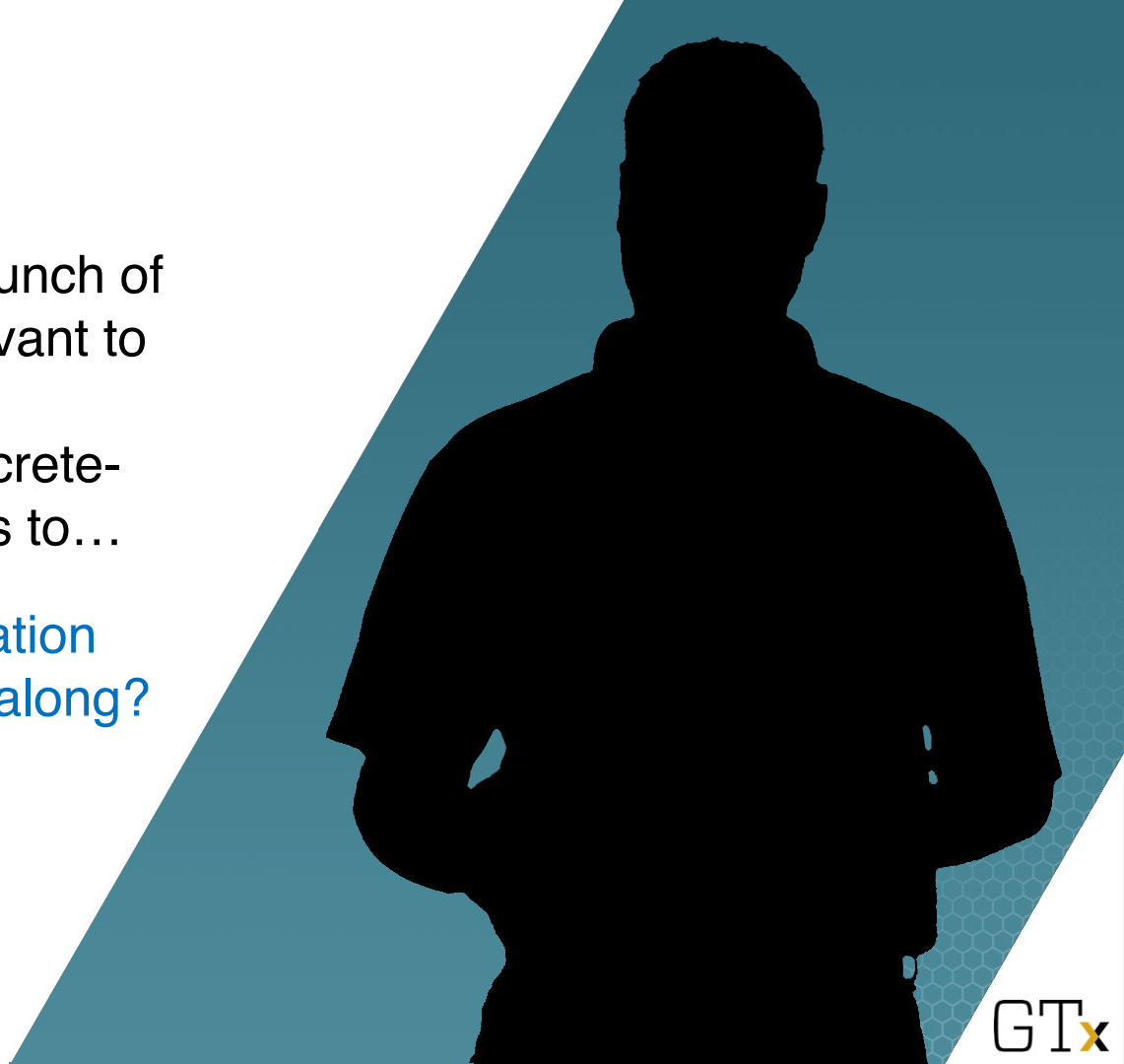


GTx

Summary

This Time: Went through a bunch of easy definitions that are relevant to general simulation models. Especially “event”, as in “discrete-event” simulation. This leads to…

Next Time: What’s the simulation clock and how does it move along?

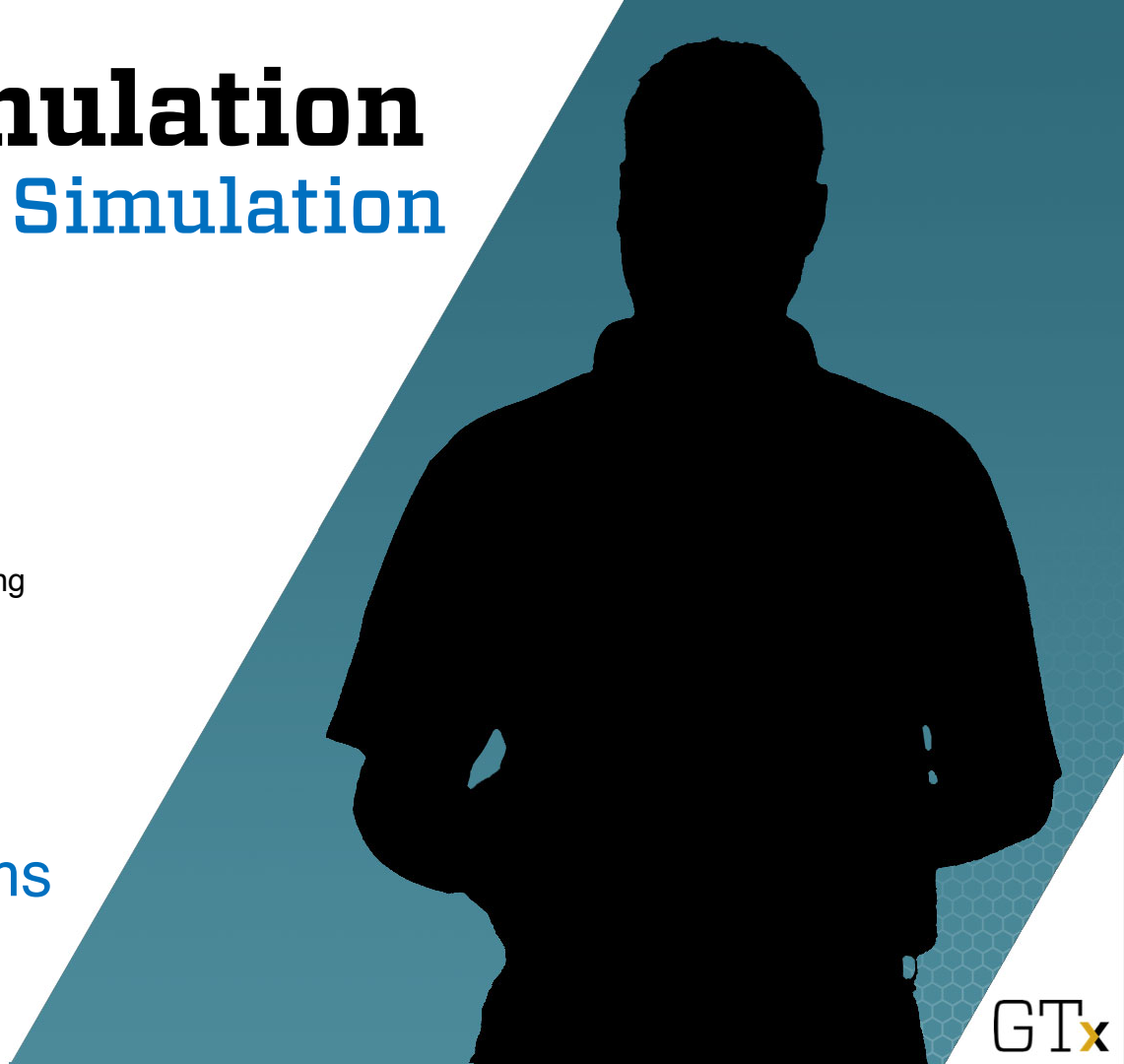


GTx

Computer Simulation

Module 4: General Simulation Principles

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Time-Advance Mechanisms

GTx

Lesson Overview

Last Time: Went through a bunch of easy definitions that are relevant to general simulation models. Especially “event”.

This Time: We’ll discuss the simulation clock and how this is used to move the simulation along as time progresses.

The clock is the “heart” of discrete-event simulation.

GTx

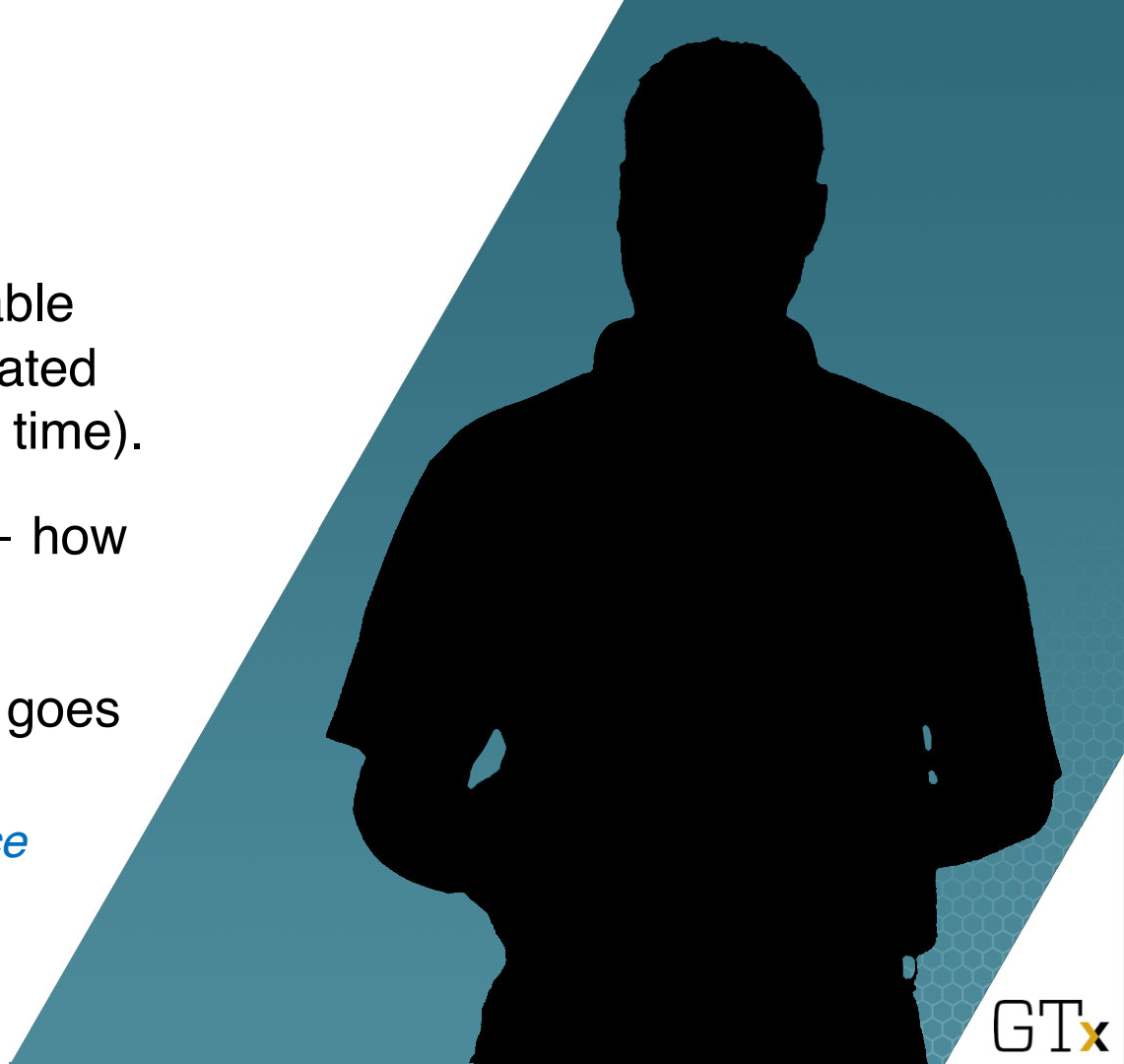
Definitions

The *simulation clock* is a variable whose value represents simulated time (which doesn't equal real time).

Time-Advance Mechanisms — how does the clock move?

Always moves forward (never goes back in time). Two ways:

- *Fixed-Increment Time Advance*
- *Next-Event Time Advance*



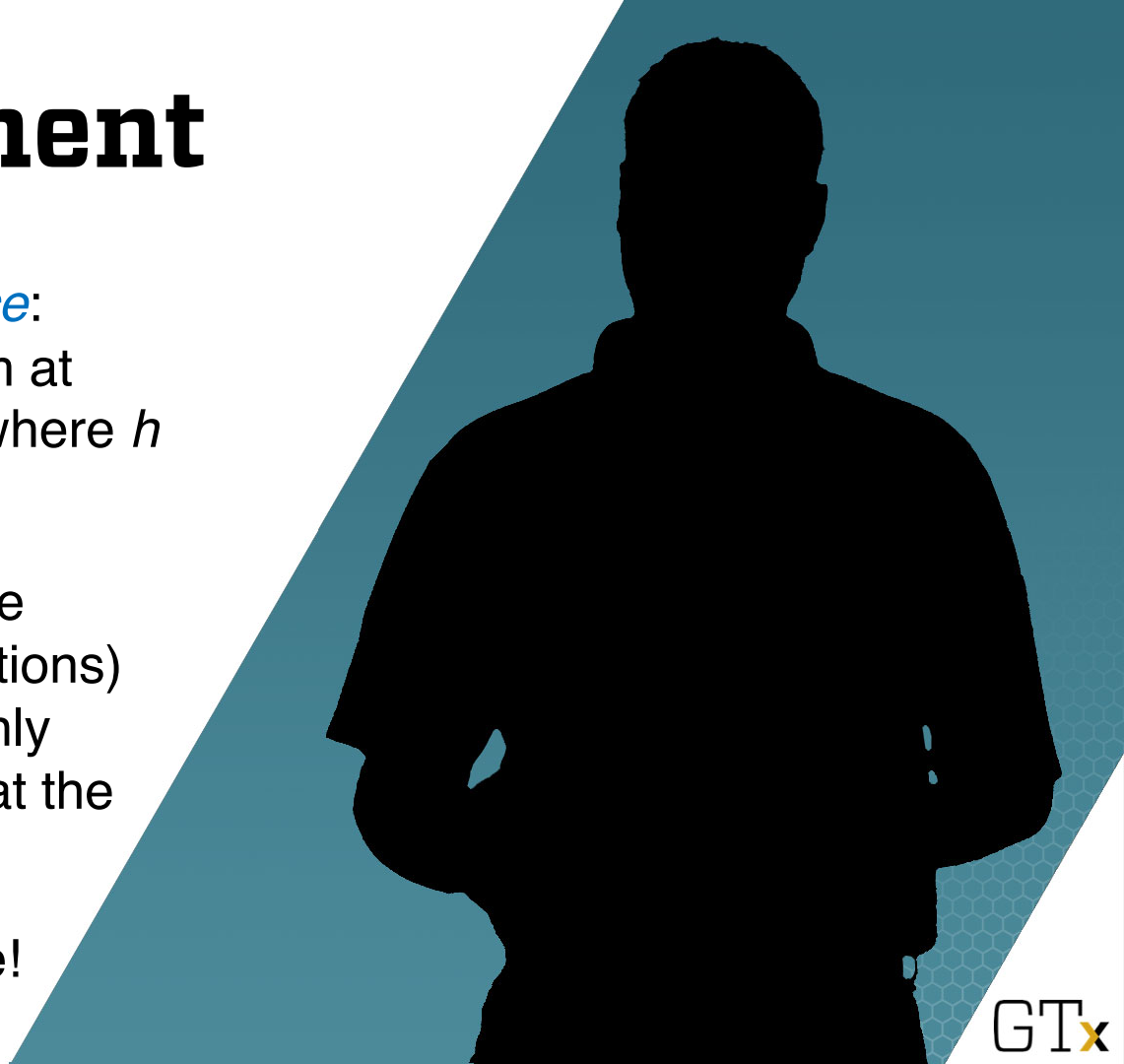
GTx

Clock Movement

Fixed-Increment Time Advance: Update the state of the system at fixed times, nh, $n = 0,1,2,\ldots$, where h is chosen appropriately.

This is used in continuous-time models (e.g., differential equations) and models where data are only available at fixed times (e.g., at the end of every month).

Not emphasized in this course!



GTx

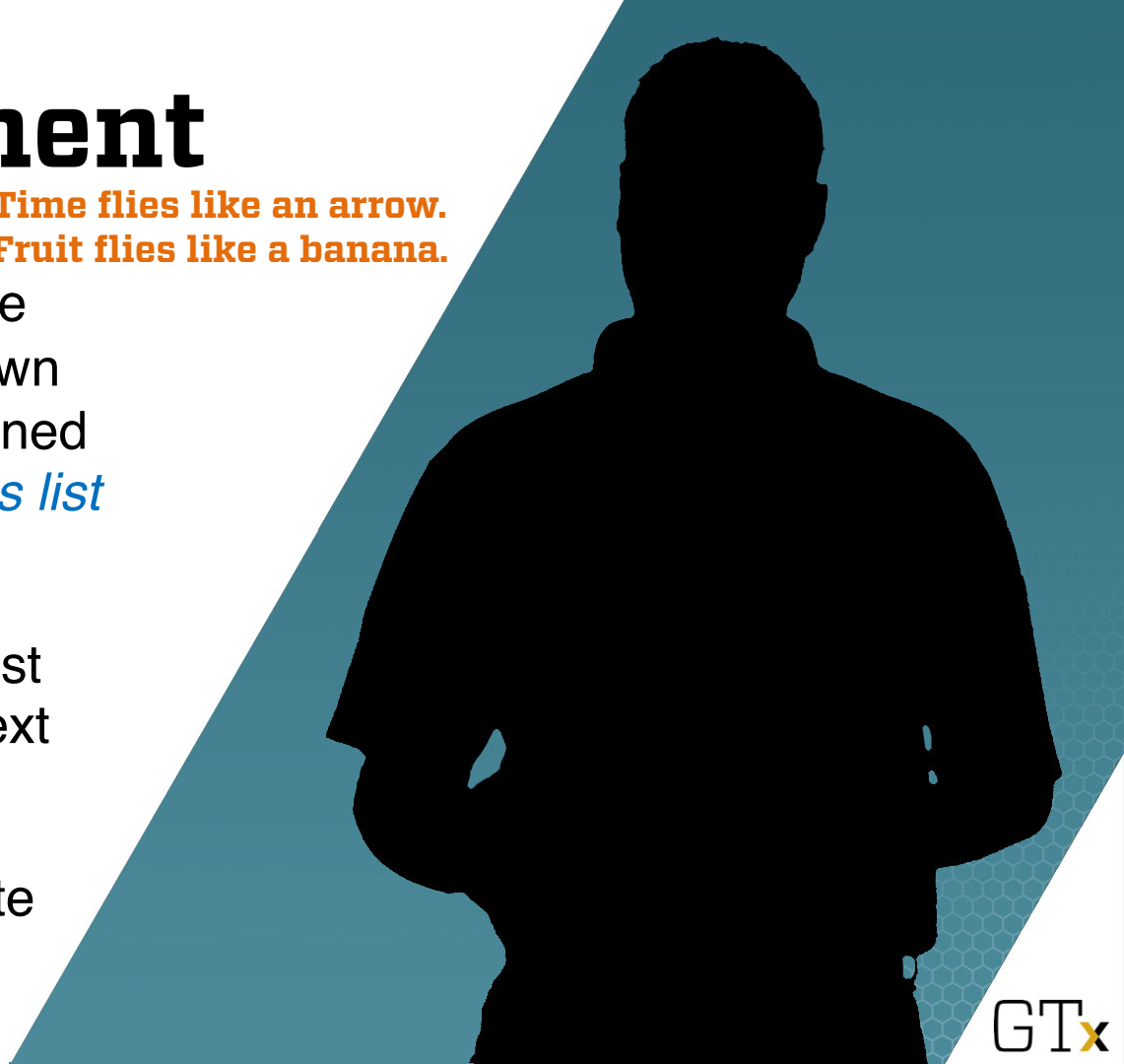
Clock Movement

Time flies like an arrow.
Fruit flies like a banana.

Next-Event Time Advance: The clock is initialized at 0. All known future event times are determined and placed in the *future events list* (FEL), ordered by time.

The clock advances to the most *imminent event*, then to the next most imminent event, etc.

At each event, the system state and FEL are updated.



GTx

Notes on FELs

The system state can only change at event times. Nothing really happens between events.

The simulation progresses by sequentially executing (dealing with) the most imminent event on the FEL.

What do we mean by “dealing with”?

Imminent Events

The clock advances to the most-imminent event.

The system state is updated, depending on what type of event it is (arrival, departure, etc.). For example, if it's an arrival event, you may have to turn on an idle server, or add a new customer to the queue of a busy server.

Update the FEL

GTx

Updating the FEL

What do we mean by “updating the FEL”?

Any time there’s an event, the simulation may update the chronological order of the FEL’s events by

- inserting new events,
- deleting events,
- moving them around, or
- doing nothing.

GTx

Updating the FEL

Example: After guy arrives at a queue, typical simulation programs will immediately spawn the next arrival time. If this arrival time is significantly far enough in the future, we'll simply put it at the "end" of the FEL (in terms of ordered execution times of the events).

Updating the FEL

If that next arrival turns out to happen before another event, e.g., a slow server finishing his current customer, then that next arrival must be inserted in the interior of the FEL (in terms of ordered execution times).

GTx

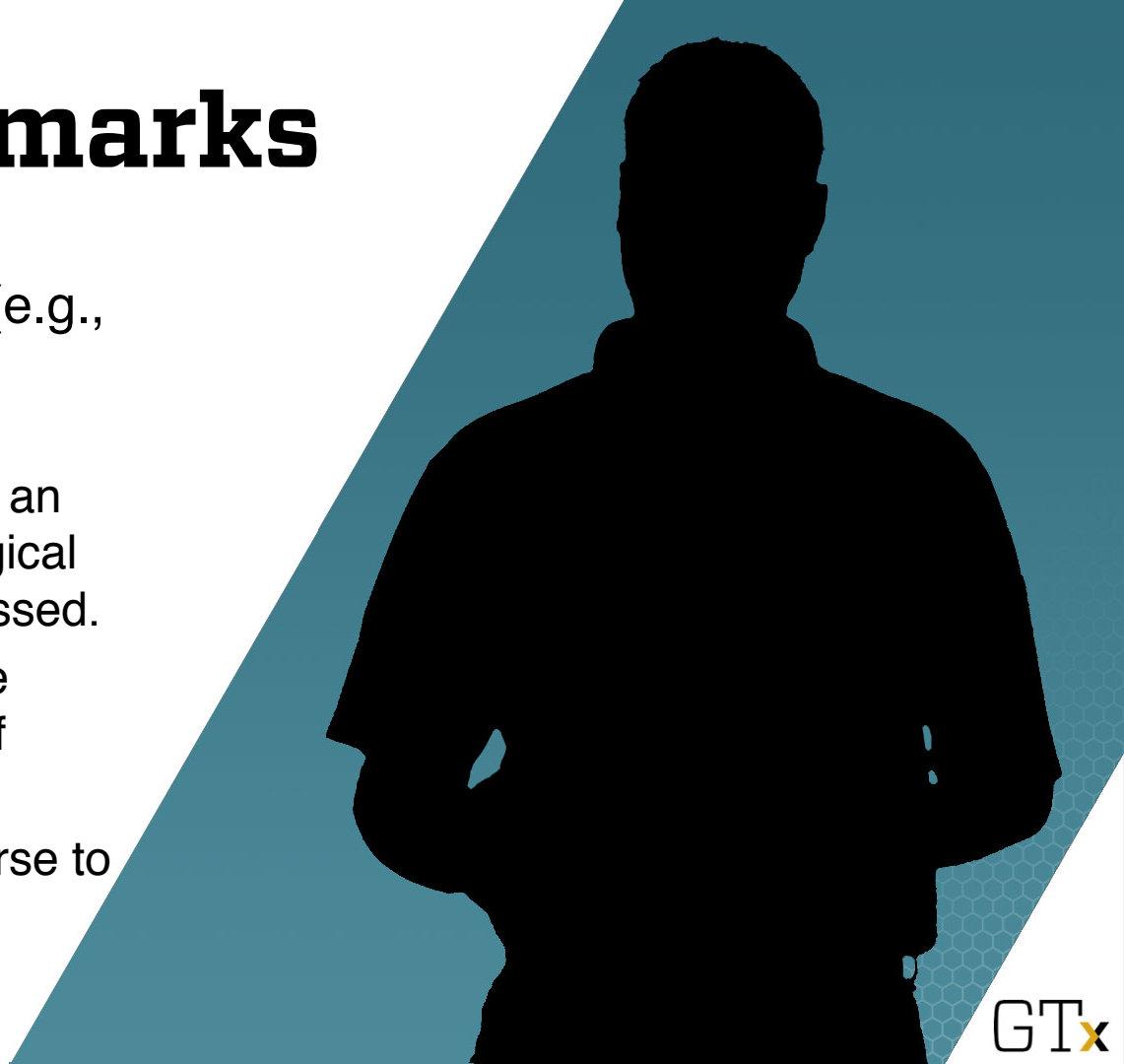
Updating the FEL

What if the arrival is a nasty-looking guy, and a number of customers currently in line get disgusted and leave or switch to other lines? Then you have to delete or move entries in the FEL.

More FEL Remarks

Need efficient list processing (e.g., *linked lists*) for the FEL.

- Singly and doubly linked lists intelligently store the events in an array that allows the chronological order of the events to be accessed.
- Such lists easily accommodate insertion, deletion, switching of events, etc.
- Take a Computer Science course to learn more.

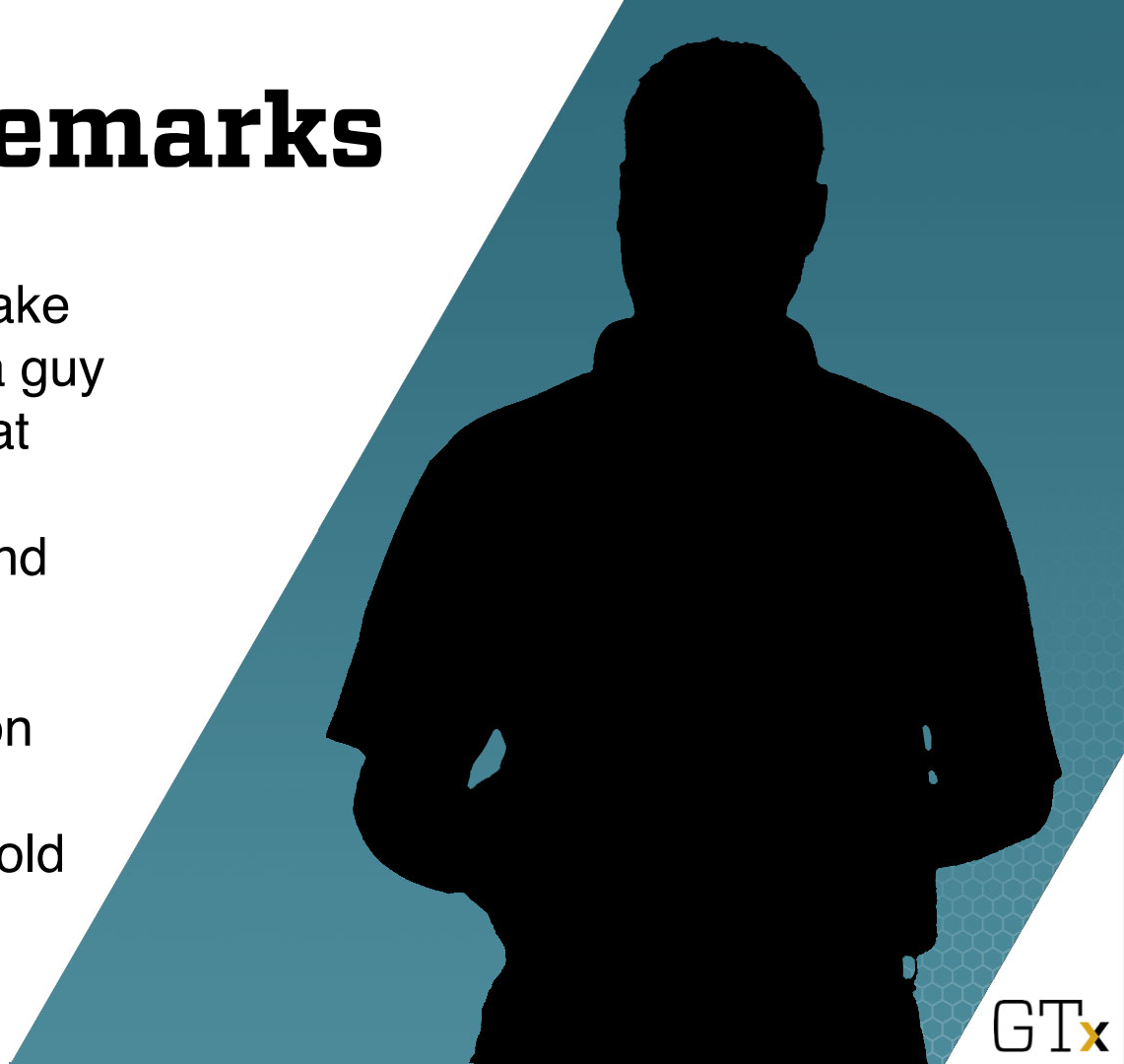


GTx

Even More Remarks

Be careful about events that take place at the same time, e.g., a guy shows up at the same time that another guy finishes getting served. Simply establish ground rules for how to deal with ties.

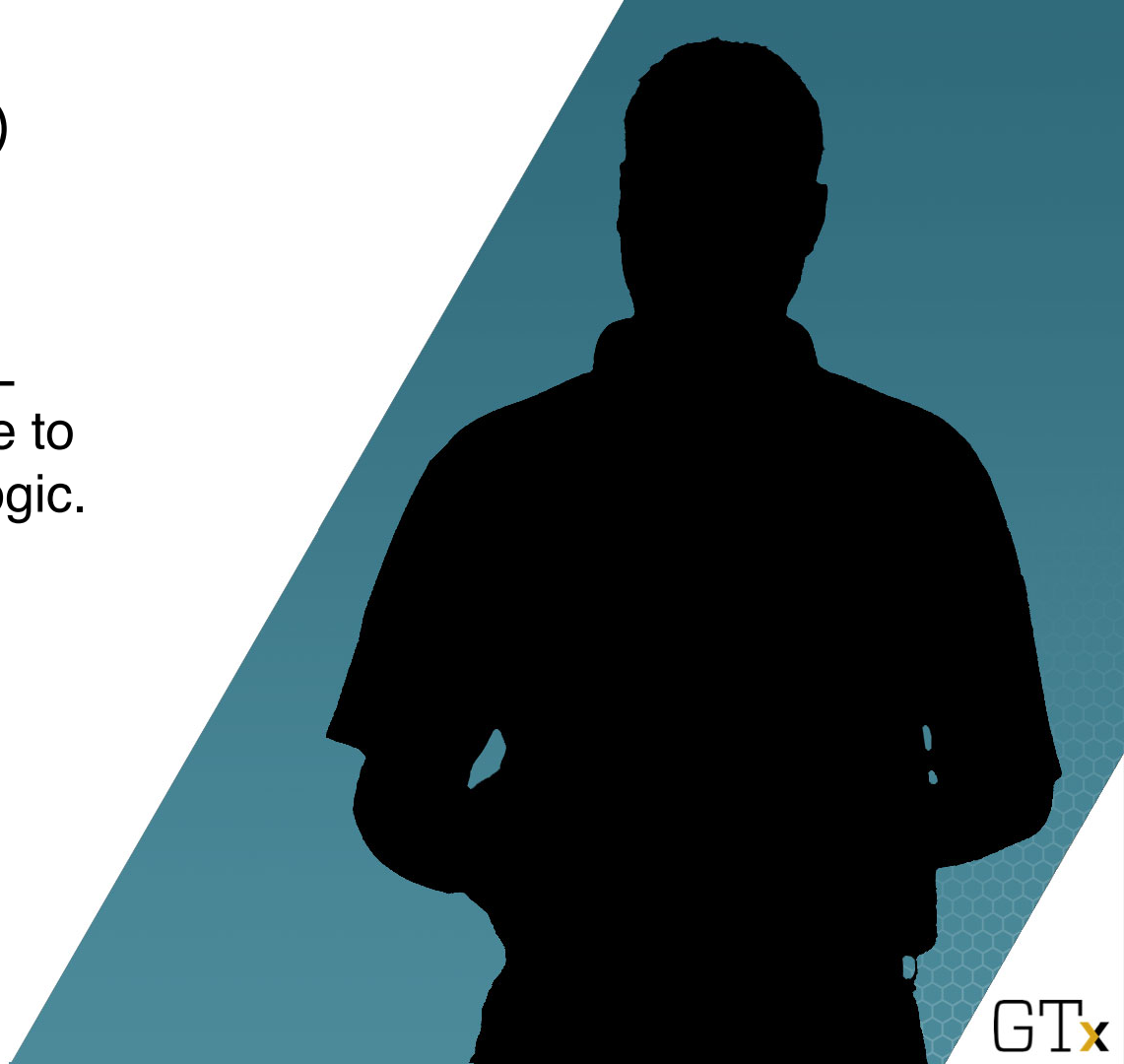
Every discrete-event simulation language maintains a FEL somewhere deep in its cold, cold heart.



Good News ☺

Most commercial simulation packages take care of the FEL stuff for you, so you don't have to mess around at all with FEL logic.

It's completely transparent!



Example

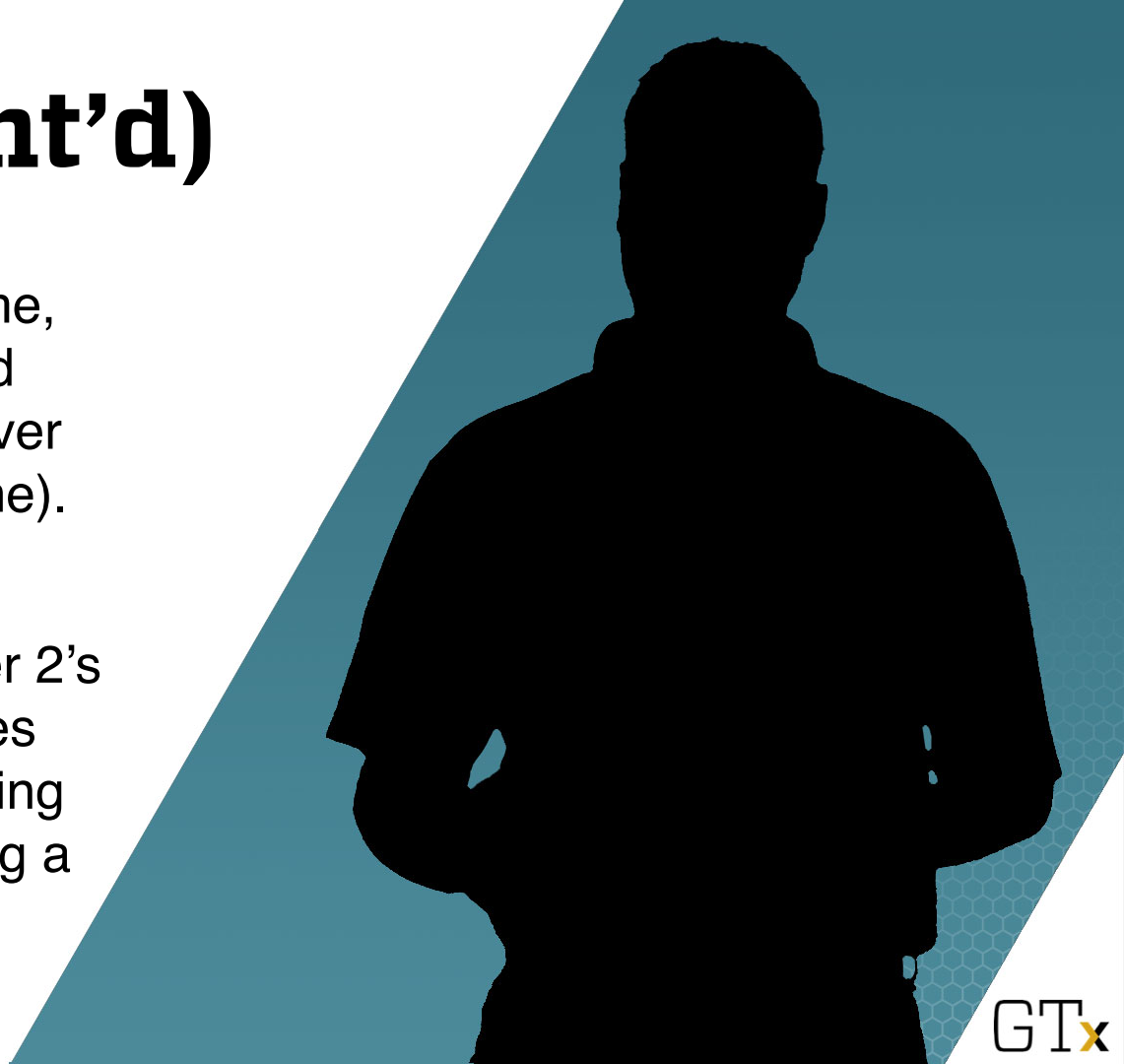
Next-Event Time Advance Example (adapted from BCNN): Consider the usual single-server FIFO queue that will process exactly 10 customers. The arrival times and service times are:

customer	1	2	3	4	5	6	7	8	9	10
arrival time	1	3	4	10	17	18	19	20	27	29
service time	5	4	1	3	2	1	4	7	3	1

Example (cont'd)

Table with entries for event time, system state, queue, FEL, and cumulative statistics (total server busy time and cust system time).

A denotes an arrival and D a departure (e.g., 2A is customer 2's arrival). Services have priorities over arrivals in terms of updating the FEL: break ties by handling a service completion first — get those guys out of the system!



GTx

Example (cont'd)

Whew!

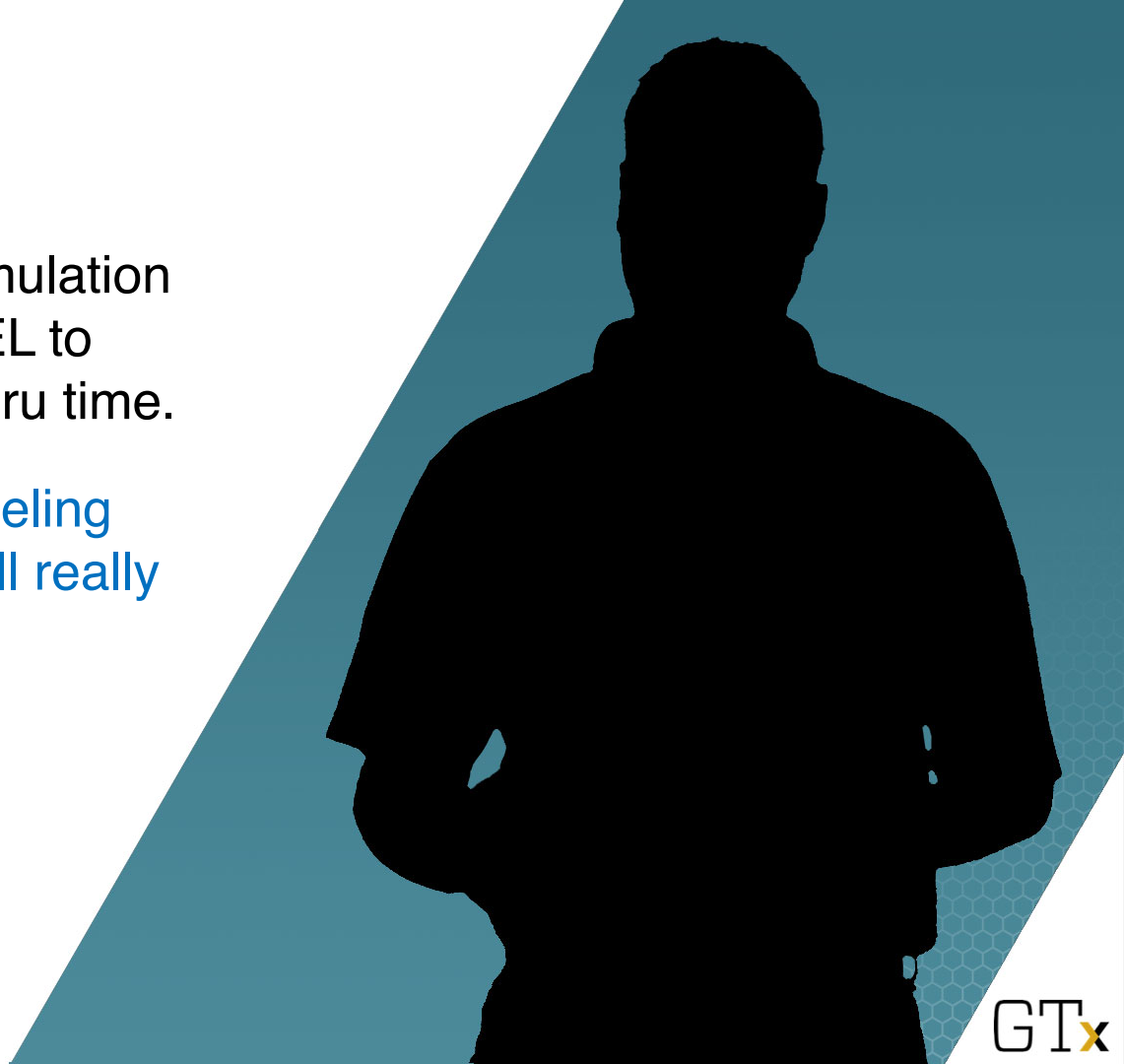
Clock	System State		Queue	FEL	Cumul Stats	
					busy	time in
t	$L_Q(t)$	$B(t)$	(cust, arr time)	(event time, event type)	time	sys
0	0	0	$\emptyset$	(1, 1A)	0	0
1	0	1	$\emptyset$	(3, 2A), (6, 1D)	0	0
3	1	1	(2, 3)	(4, 3A), (6, 1D)	2	2
4	2	1	(2, 3), (3, 4)	(6, 1D), (10, 4A)	3	4
6	1	1	(3, 4)	(10, 2D), (10, 4A)	5	10
10	0	1	$\emptyset$	(10, 4A), (11, 3D)	9	18
$\vdots$						

Tedious as this may appear, the computer handles all of this stuff in a flash.

Summary

This Time: Discussed the simulation clock, and how to use the FEL to move the simulation along thru time.

Next Time: Two general modeling approaches, one of which will really help you to easily simulate complicated systems.

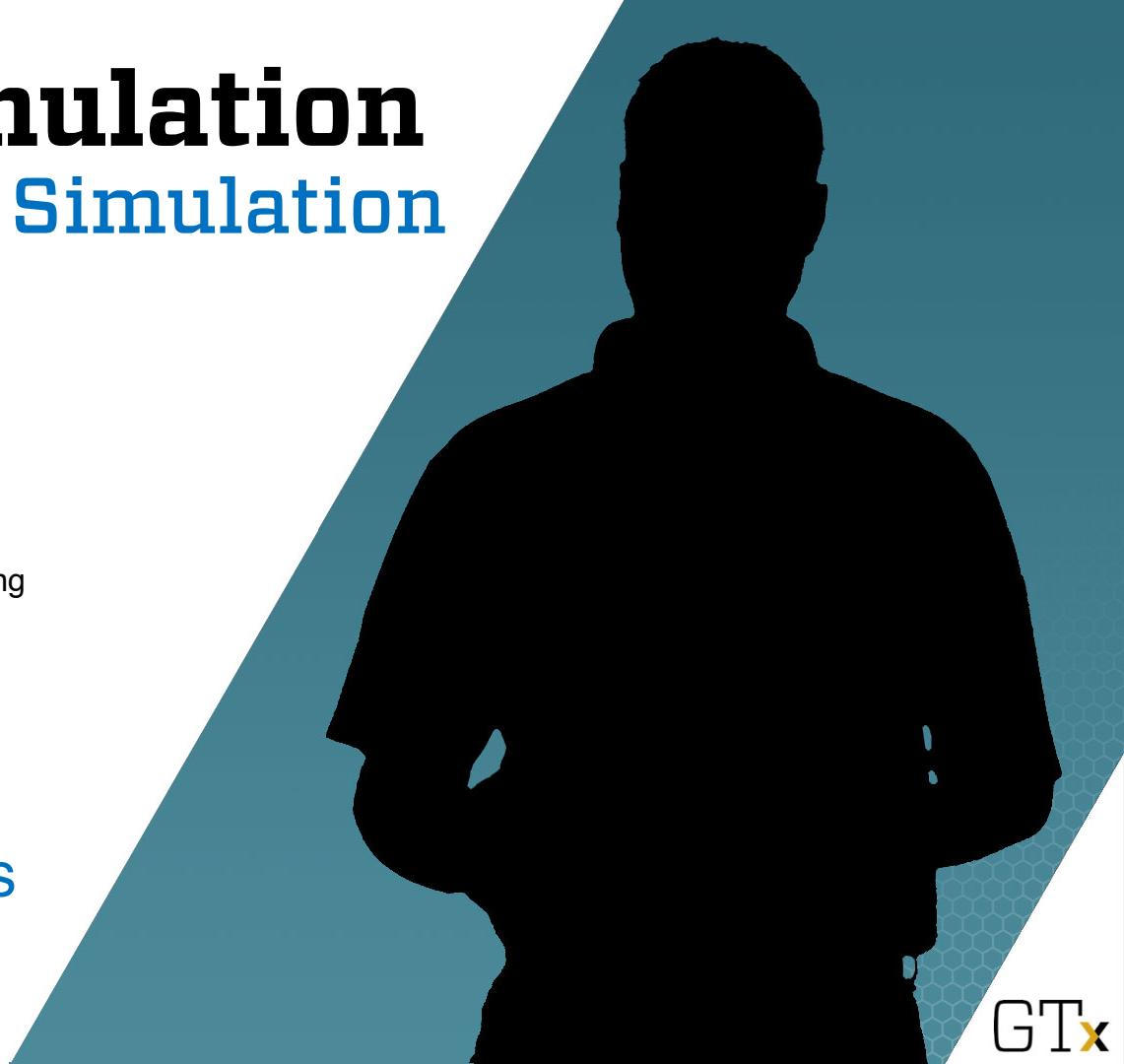


GTx

Computer Simulation

Module 4: General Simulation Principles

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Two Modeling Approaches

GTx

Lesson Overview

Last Time: Discussed the FEL and how time flies in a discrete-event simulation.

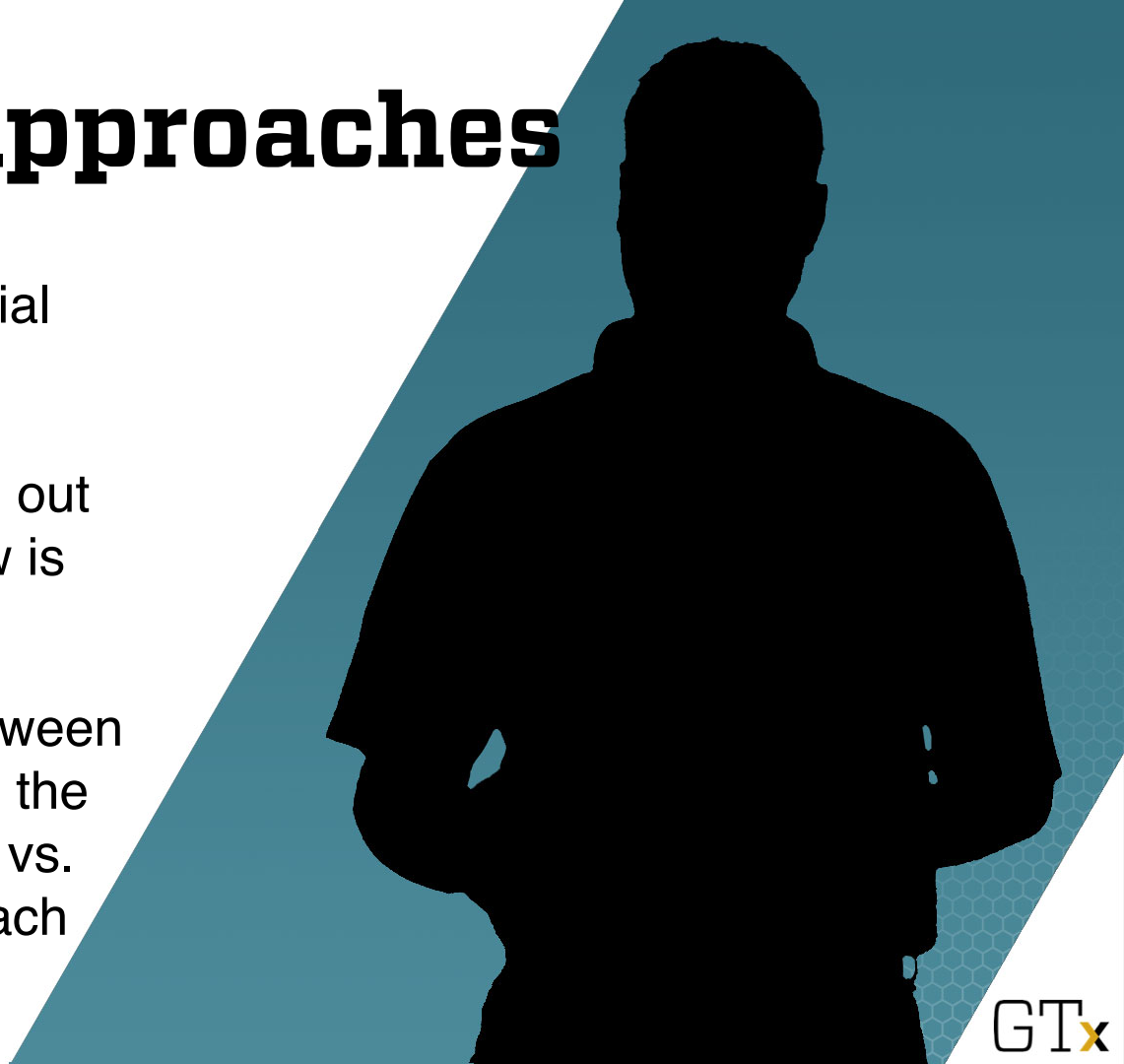
This Time: We'll look at two high-level simulation modeling approaches.

The "process-interaction" approach will be used to model complicated simulation processes.

High-Level Approaches

Hinted last time that commercial simulation packages help you avoid grief by automatically handling the FEL and carrying out the drudgery themselves. How is that possible?

It comes down to a choice between two modeling "world views" — the event-scheduling approach ☹ vs. the process-interaction approach ☺.



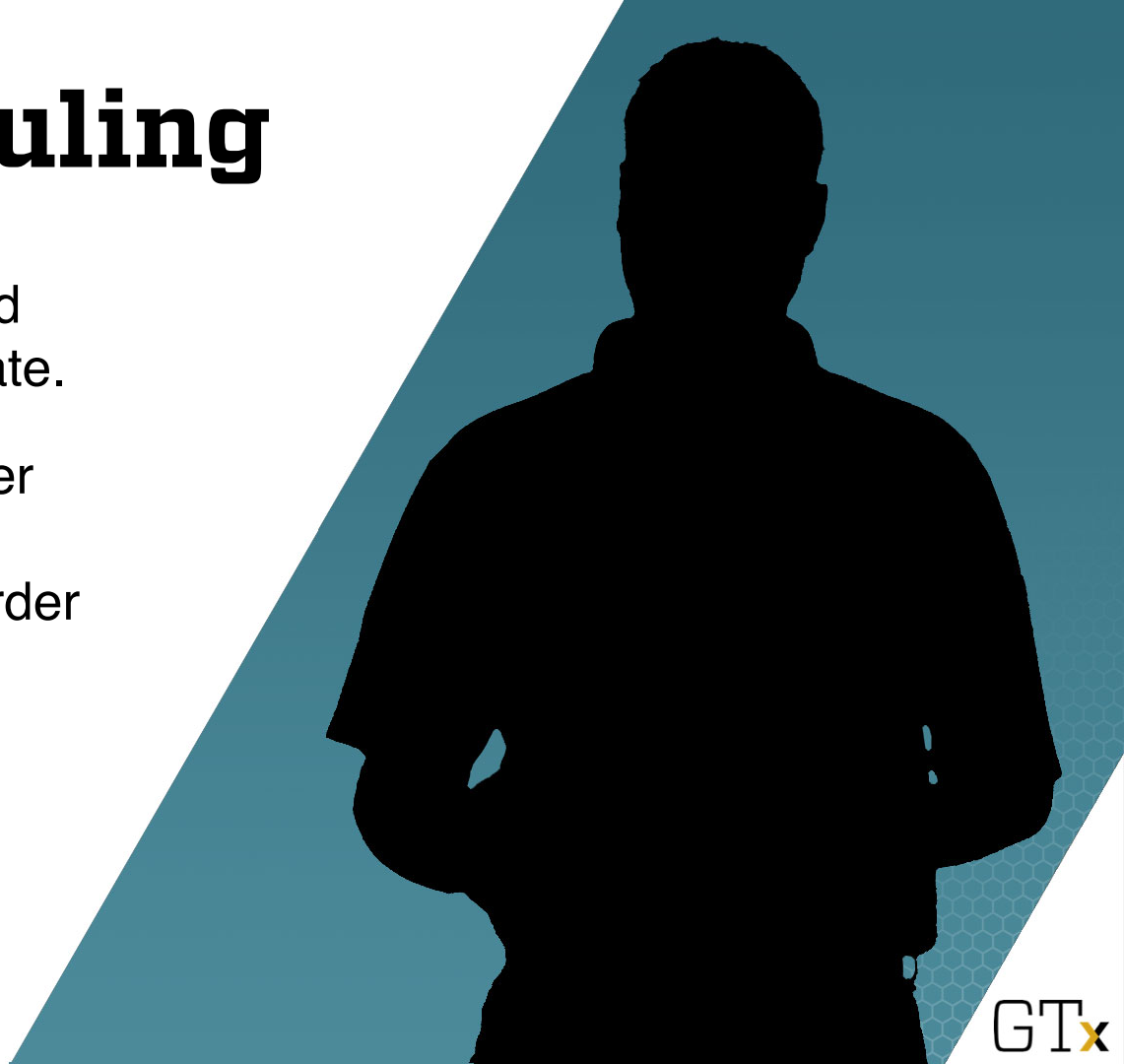
GTx

Event-Scheduling

Concentrate on the events and how they affect the system state.

Help the simulation evolve over time by keeping track of every freaking event in increasing order of time of occurrence.

This is a bookkeeping hassle.



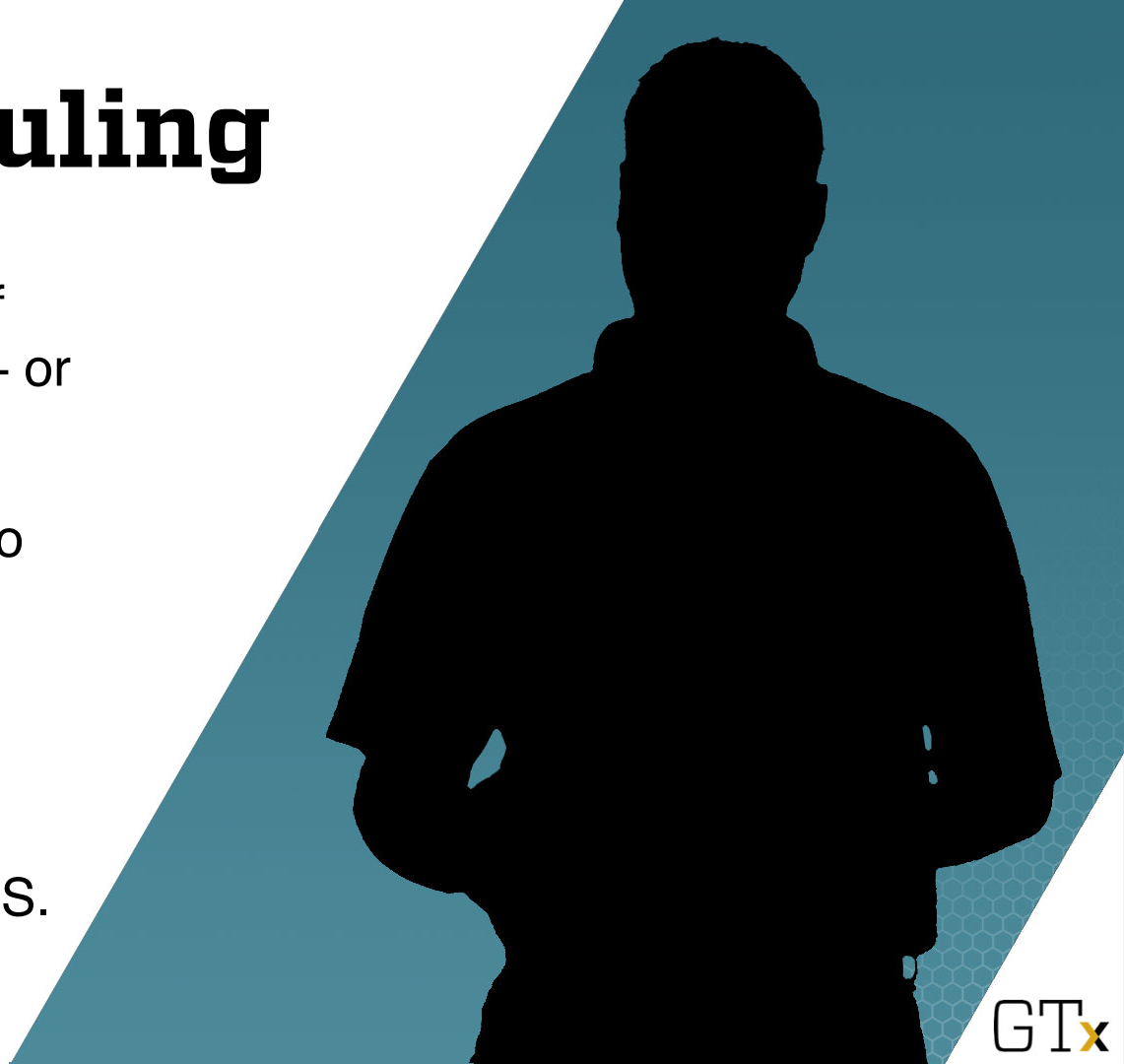
GTx

Event-Scheduling

You might use this approach if you were programming in C++ or Python from scratch.

If you think that you'd like to do so, see the next few slides outlining a generic simulation program. . .

My guess is that you'll get the idea and never want to use E-S.



GTx

Generic ES Flowchart (Law's book)

Main Program

0. Initialization Routine
   - Set clock to 0
   - Initialize system state and statistical counters
   - Initialize FEL

1. Invoke Timing Routine — what/when is the next event?
   - Determine next event type — arrival, departure, end simulation event
   - Advance clock to next time

Main Program (cont'd)

2. Invoke Event Routine — for event type i
   - Update system state
   - Update statistical counters
   - Make any necessary changes to FEL. E.g., if you have an arrival, schedule the next arrival or maybe schedule end of simulation event. If simulation is now over, write report and stop. If simulation isn't over, go back to the Timing Routine.

3. Go back to 1

Arrival Event

- Schedule next arrival
- Is server busy?
  - If not busy,
    - Set delay (wait) for that customer = 0
    - Gather appropriate statistics (e.g., in order to ultimately calculate average customer waiting times)
    - Add one to the number of customers processed
    - Make server busy
    - Return to main program
  - If busy,
    - Add one to number in queue.
    - If queue is full, panic (or do something about it)
    - If queue isn't full, store customer arrival time
    - Return to main program

Departure Event

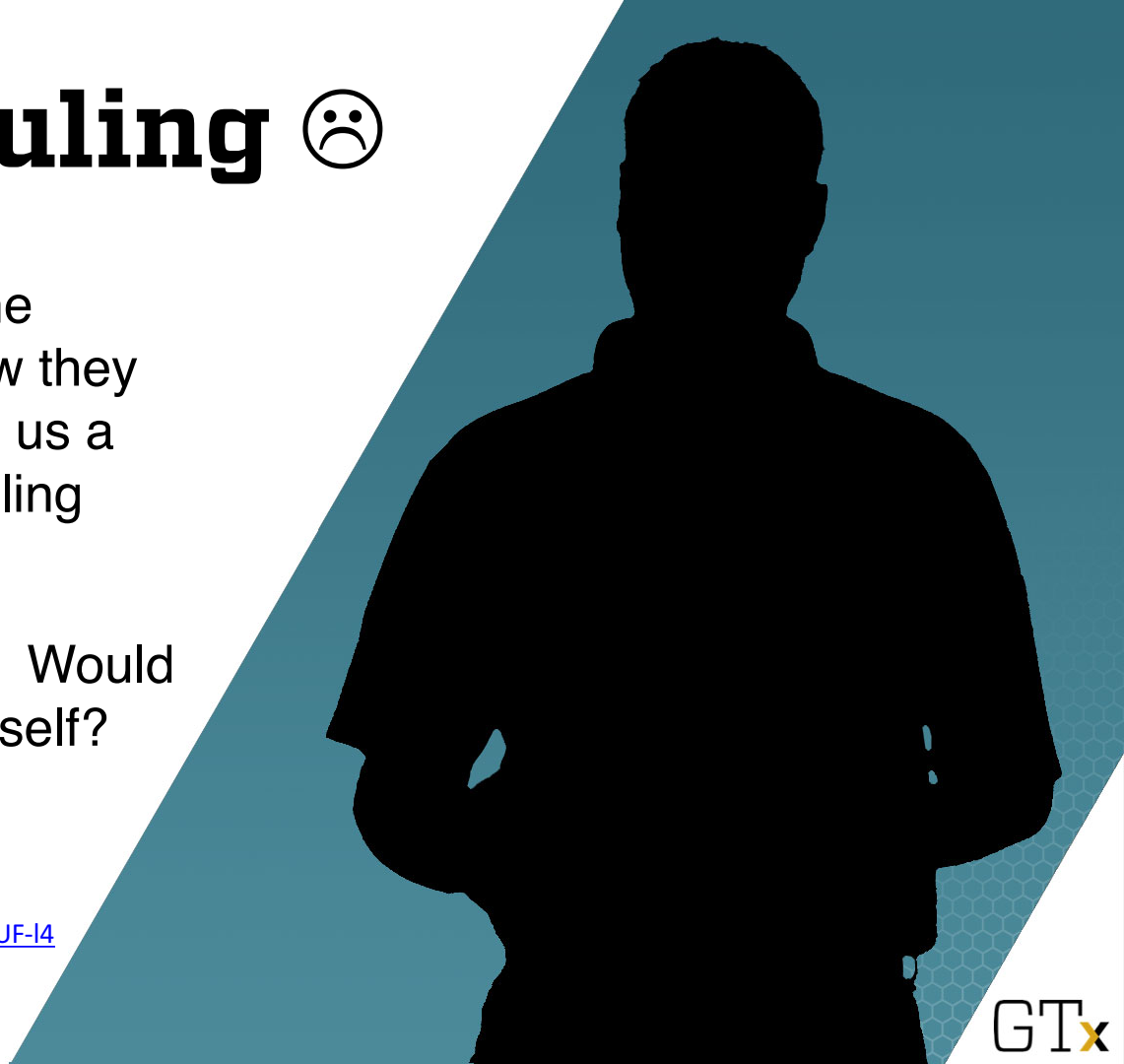
- Is the queue empty?
  - If empty,
    - Make server idle
    - Eliminate departure event from consideration of being the next event
    - Return to main program
  - If not empty,
    - Subtract one from number in queue
    - Compute delay of customer entering service and gather statistics
    - Add one to the number of customers delayed (processed)
    - Schedule departure for this customer
    - Move each remaining customer up in the queue
    - Return to main program

Event-Scheduling ☹

Putting together and coding the previous three charts (and how they interact with each other) gives us a nice but tedious event-scheduling solution.

But this is really quite a mess! Would you really want to do this yourself?

www.youtube.com/watch?v=e4BwG7UF-l4



GTx

Process-Interaction ☺

How would we handle such a generic model in a P-I language (like Arena)? All you do is:

- Create customers every once in a while.
- Process (serve) them, maybe after waiting in line.
- Dispose of the customers after they're done being processed.

This is soooo easy. . .

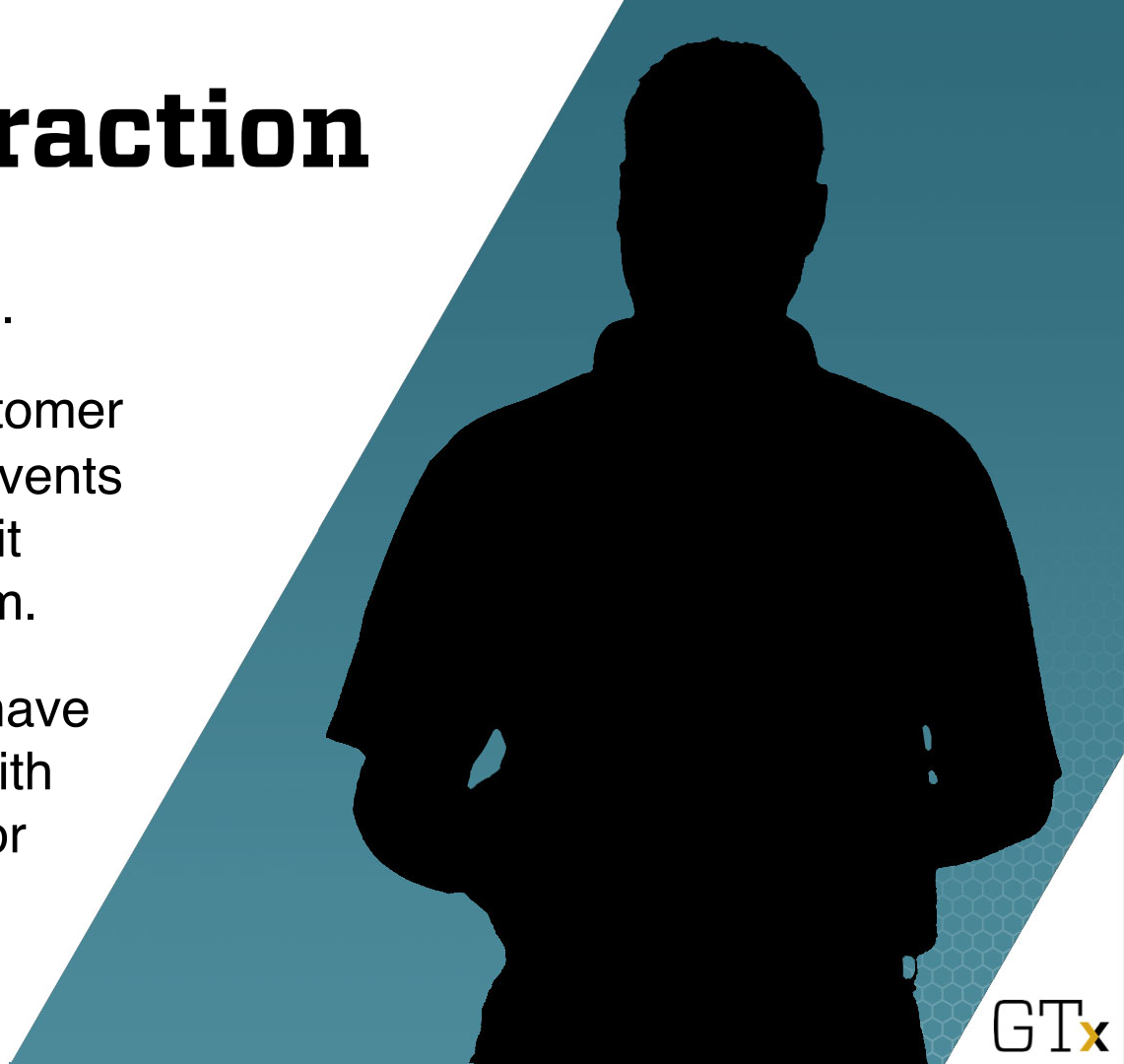
GTx

Process-Interaction

We use this approach in class.

Concentrate on a *generic* customer (entity) and the sequence of events and activities it undergoes as it progresses through the system.

At any time, the system may have many customers interacting with each other as they compete for resources.

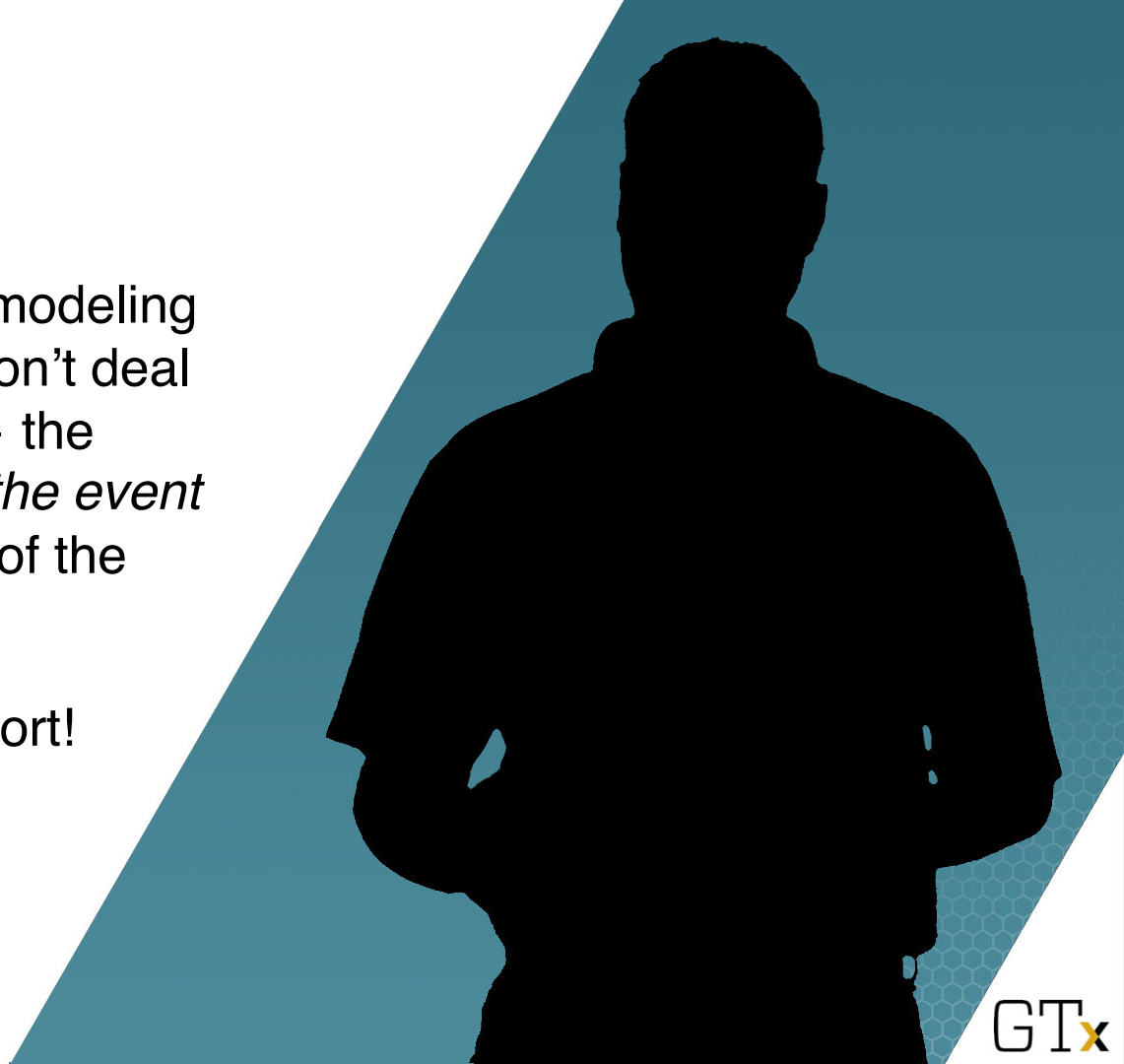


GTx

P-I (cont'd)

You do the generic customer modeling in the P-I approach, but you don't deal with the event bookkeeping — the *simulation language handles the event scheduling* deep inside for all of the generic customers.

Saves lots of programming effort!

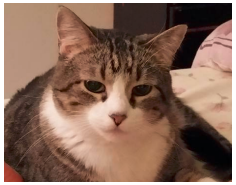


P-I (cont'd)

Example: A customer is generated, eventually gets served, and then leaves. Create – Process – Dispose.

Easy, Easy, Easy! Like Like Like Like!

Arena or any other good simulation language adopts this approach.



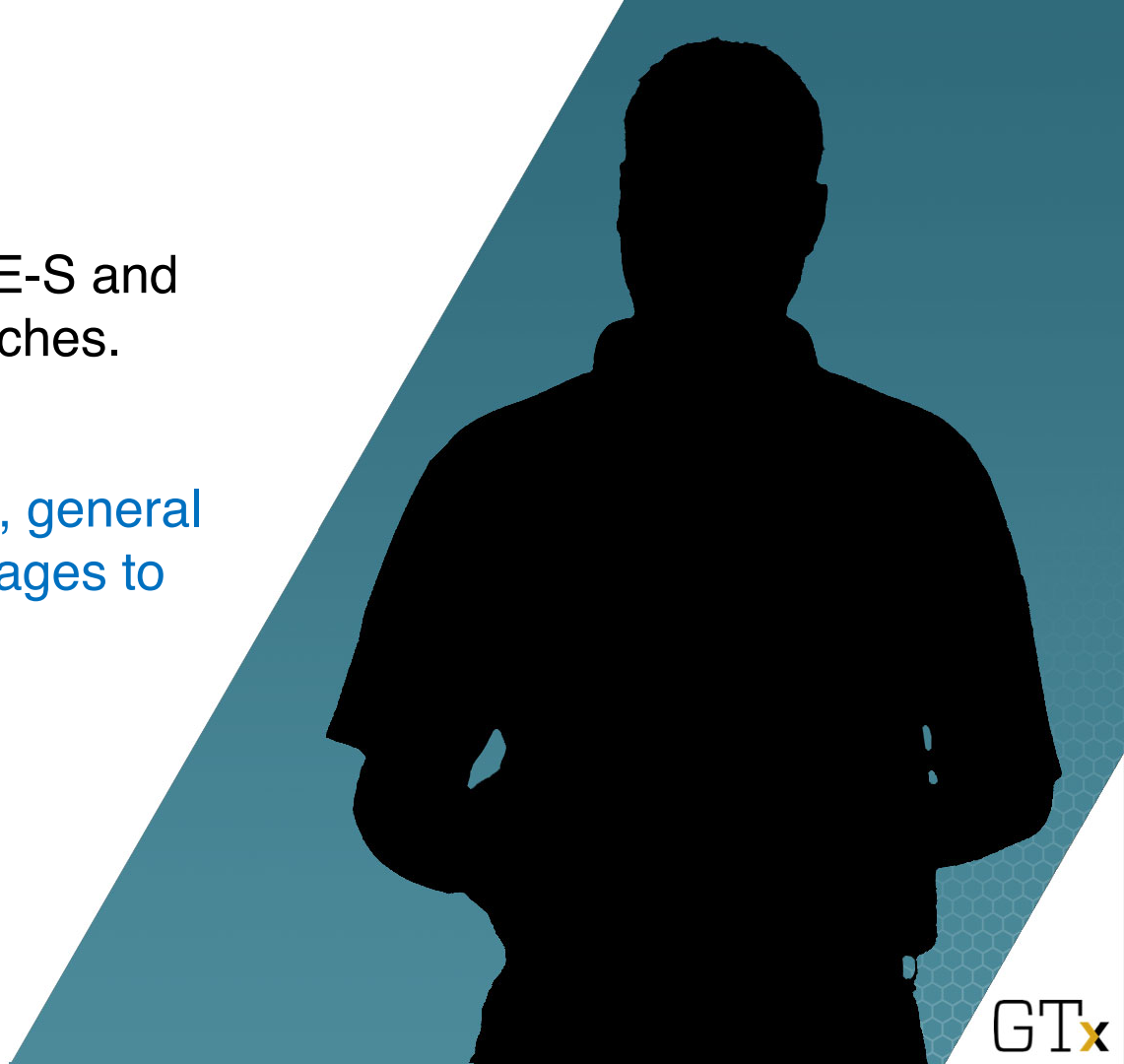
It's purr-fectly easy!

GTx

Summary

This Time: Talked about the E-S and P-I general modeling approaches. We ❤ P-I.

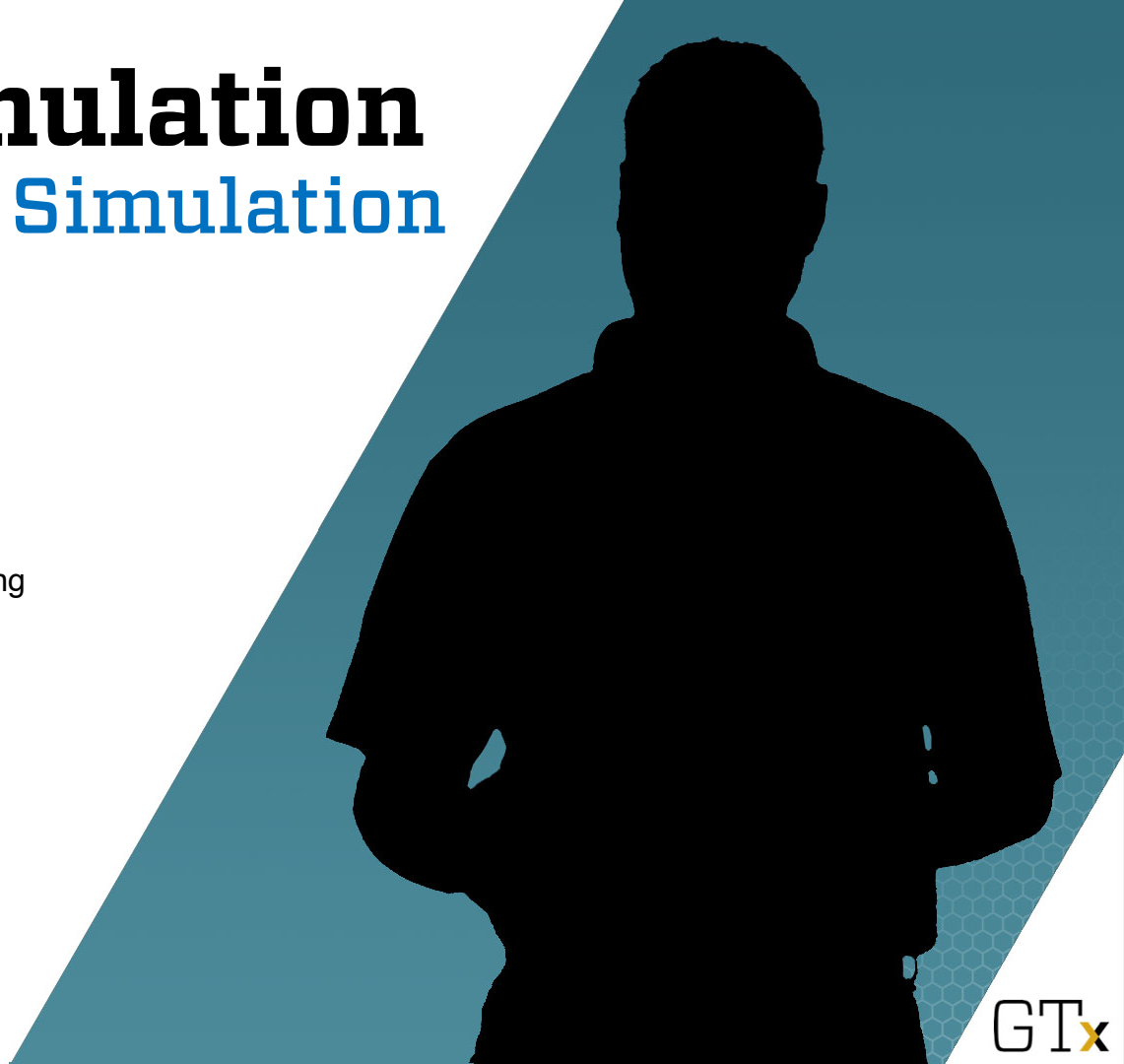
Next Time: We'll give a short, general overview of simulation languages to end the module.



Computer Simulation

Module 4: General Simulation Principles

Dave Goldsman, Ph.D.
Professor
Stewart School of Industrial and Systems Engineering



Simulation Languages

GTx

Lesson Overview

Last Time: Looked at two high-level simulation modeling approaches — E-S and P-I.

This Time: Discuss simulation languages in a very generic way.

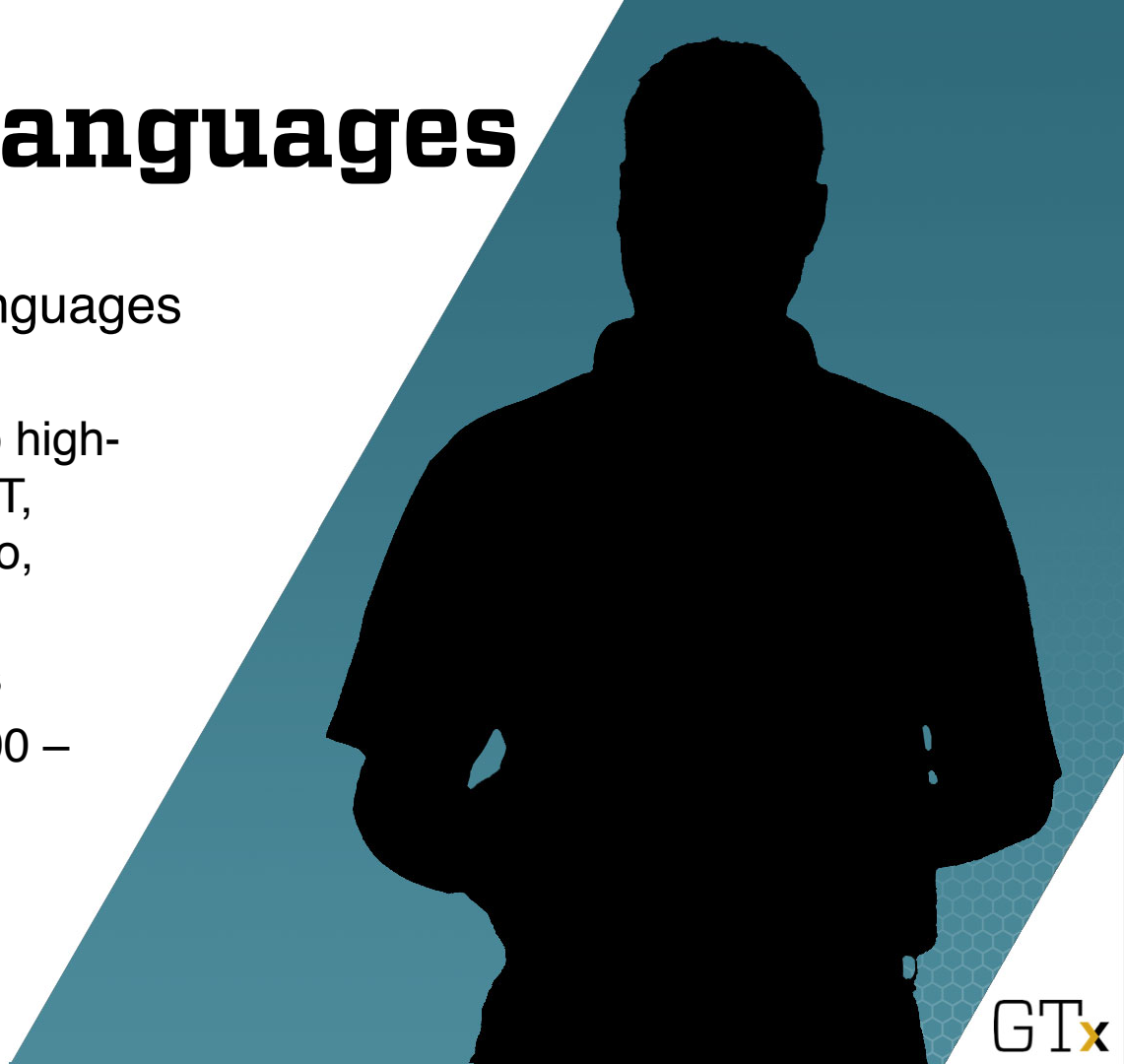
There's a *lot* out there… choose carefully!

GTx

Simulation Languages

More than 100 commercial languages in the ether.

- Examples (sort of from low- to high-level): FORTRAN, SIMSCRIPT, GPSS/H, Extend, Arena, Simio, Automod, AnyLogic
- 5–10 major players at schools
- Huge price range from <$1,000 – $100,000?



GTx

Languages (cont'd)

But there's also some Freeware available!

- Several nice packages in Java, Python (e.g., SimPyl).
- Little bit higher learning curve, but not too bad.

Languages (cont'd)

When selecting a language, you should take into account:

- Cost considerations: $$, learning curve, programming costs, run-time costs, etc.
- Ease of learning: documentation, syntax, flexibility
- World view: E-S, P-I, continuous models, combination
- Features: RV generators, stats collection, debugging aids, graphics, user community

GTx

Languages (cont'd)

Where to learn simulation languages?

- Here (nice to have you here!)
- Textbooks (especially in conjunction with this class)
- Conferences such as the Winter Simulation Conference
- Vendor short courses (at a potentially steep cost)

GTx

Summary

Just went over some high-level considerations in choosing a discrete-event computer simulation language.

This completes Module 4, where we discussed simulation terminology and what's under the simulation hood.

Module 5 will introduce and give tutorials on the Arena simulation language!

GTx