

EE/PEP 345

Modeling and Simulation

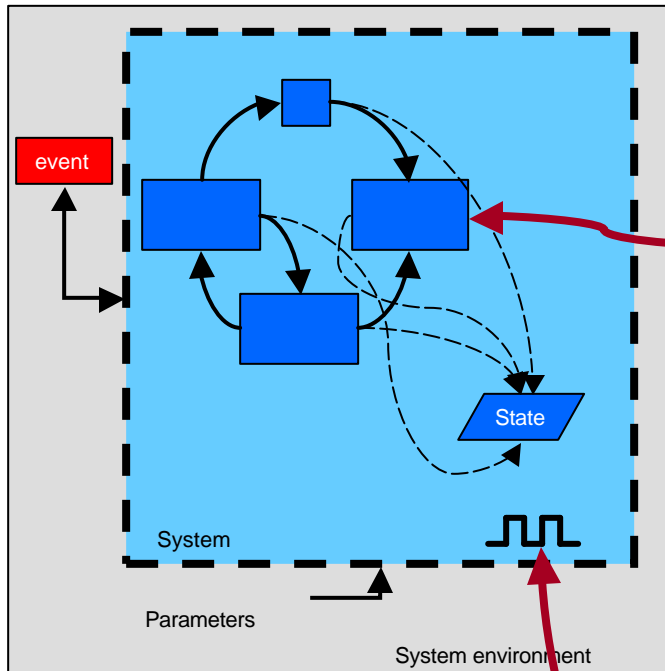
Spring 2004

Class 3

Today's topics

- Text – Chapter 3 – General Principles

Concepts in Discrete-Event Simulation



System: set of objects, joined to accomplish some purpose

Model: Abstraction of real system, defines relationships between entities, system parameters, system state, etc.

State of system: collection of variables necessary to describe system at any time

Entity: object of interest in system

Attribute: property of an entity

Event: Instantaneous occurrence that may be associated with change of system state

Activity: a predefined set of actions by system objects, usually in a specified time period

List: a collection of associated entities, ordered in some logical fashion

Event notice: Record of an event to occur at some present or future time, along with associated data

Event list: List of event notices (Future Event List)

Delay: Duration of time of unspecified indefinite length, not known until it ends

Clock: A system variable representing simulated time

Able and Baker, in our current context

- System state:

$L_Q(t)$ – the number of customers waiting to be served at time, t

$L_A(t) = (0,1)$, the idle/busy status of Able at time, t

$L_B(t) = (0,1)$, the idle/busy status of Baker at time, t

- Entities:

Able and Baker are the only entities we tracked last time

- Activities:

Inter-arrival time

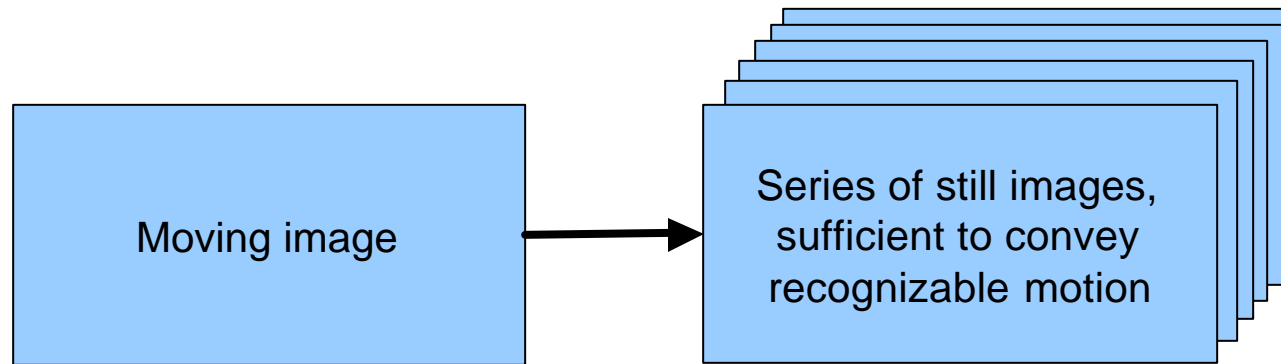
Able's service time

Baker's service time

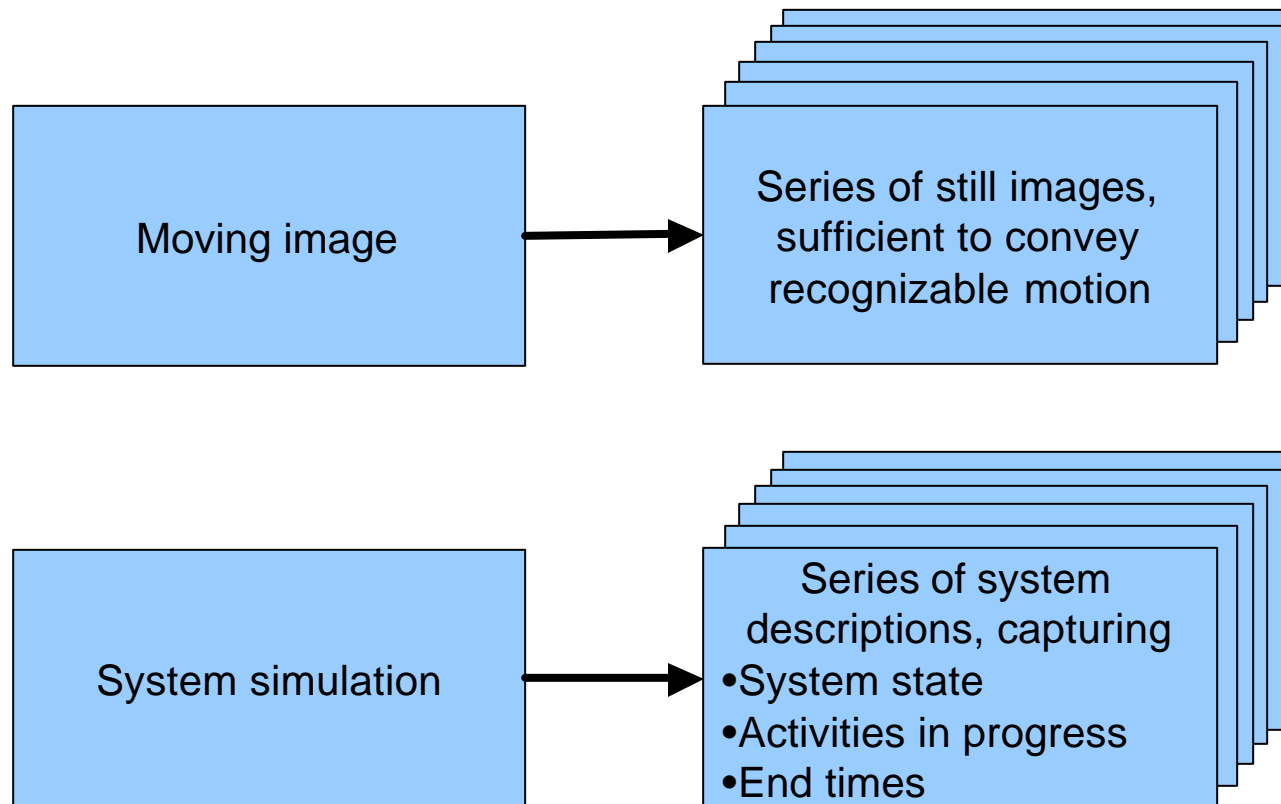
- Delay

Customer waiting time until serviced by Able or Baker

System Snapshots



System Snapshots



Example system snapshot at time t

Clock	System State	Entities and Attributes	Set 1	Set 2	...	Future Event List (FEL)	Cumulative Statistics and Counters
t	(x,y,z,...)		Queue memberships			(3, t ₁) (1, t ₂)	
t ₁	(x-1, y, z', ...)					(1, t ₂)	
...							

System state = (x, y, z, ...) means:

state variable 1 = x, (e.g., customers waiting = x)

state variable 2 = y, (e.g., Able is busy)

state variable 3 = z, ... (e.g., Baker is idle)


Future Event List = {(3, t₁), (1, t₂)} means:

A type 3 event will occur at t=t₁ (e.g., Baker service completion at t₁)

A type 1 event will occur at t=t₂ (e.g., customer arrival at t₂)

Event-scheduling/time-advance algorithm

Clock	System State	...	Future Event List	...
t	(5,1,6)		(3,t ₁) (1,t ₂) (1,t ₃) ... (2,t _n)	

- 
1. Remove event notice for imminent event (at $t=t_1$)
 2. Advance CLOCK to imminent event time
 3. Execute imminent event
 Update system state, change entity attributes, set membership, as needed
 4. Generate future events, if needed, and place them on FEL, ranked by time of occurrence
 5. Update cumulative statistics and counters

Clock	System State	...	Future Event List	...
t ₁	(5,1,5)		(1,t ₂) (4,t [*]) (1,t ₃) ... (2,t _n)	

Actions occurring on the FEL

- Deleting the imminent event (e.g., servicing a customer in queue)
- Removing events that have already been scheduled (e.g., if customer can leave a queue)
- Adding future events as they get scheduled
- Maintaining time-ordering of events

Length and contents of FEL changes dynamically during simulation

How can you efficiently (in terms of storage and processing efficiency) maintain information about the FEL in a simulation?

Considerations in running an event-scheduling/time-advance simulation

- System snapshot at $t=0$
 - Preset internal values for simulation – initial conditions
 - External controls – “exogenous” event – from the system environment
- Bootstrapping
 - Used for creating an external event stream, e.g., customer arrivals.
 - Events are placed on FEL by calculating offset from present simulation time.
- Simulation stopping conditions
 - Run for a preset time (e.g., 1000 seconds of simulator time)
 - Run for a preset number of events (e.g., 1000 customers have been served)
 - Run for a preset real-world time (e.g., until the conference paper is due to be sent)
 - Run until an internal event occurs (e.g., the receiver synchronizes)
 - Run until an external event occurs (e.g., user decides to modify experiment)

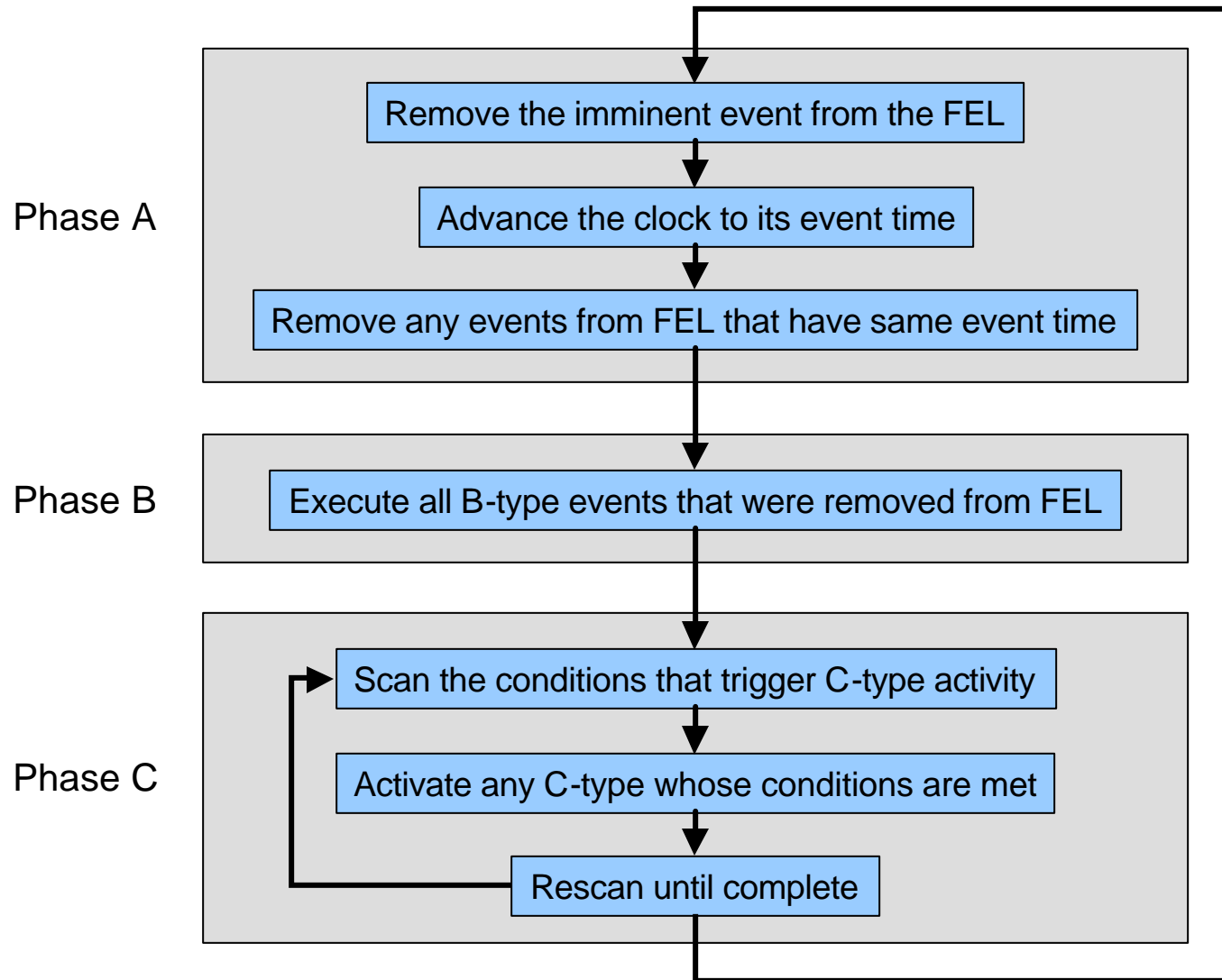
World Views

- System model is developed differently, based on “world view” employed:
 - Event scheduling
 - Events (the only things that can affect system state) are the prime focus of simulation
 - Process-interaction
 - A process is the life cycle of one entity in simulation
 - Entities:
 - request resources = *events* and
 - use them = *activities*
 - Well suited to object oriented languages (e.g., C++)
 - Entities and processes = objects
 - Activity-scanning
 - Simulation runs in fixed time increments
 - Rule-based approach to decide if an activity can begin at any given point in simulated time
 - Events are quantized to the cycle time of the simulation

World Views - continued

- Activity-scanning costs considerable simulation overhead to decide if an activity can be started
- Hybrid activity-scanning/event-scheduling approach:
 - Two types of activities:
 - B activities:
 - Must occur.
 - Primary events (completion of activity) and unconditional activities
 - Can be scheduled in advance
 - C activities:
 - Conditional events or activities
 - Depend on conditions being true
 - Scan for them at the end of a time advance

Three Phase Approach to Improved Activity-Scanning

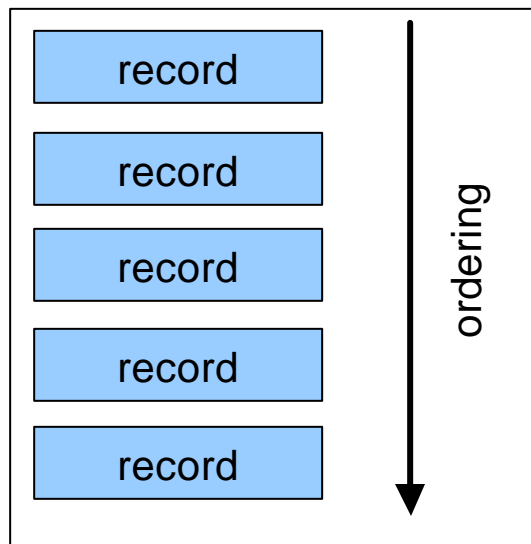


List Processing

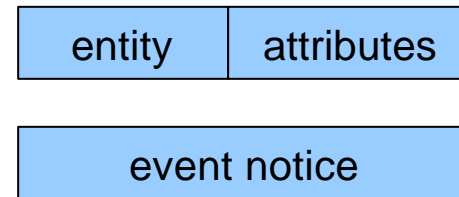
- Methods to handle lists of entities and the FEL
- Properties and operations on lists:
 - A list is a set of records with an ordering (ranking) operation, accessible in order
 - One item in a list is a “record” – used to store one entity or one event notice.
 - The head

List processing

A generic list

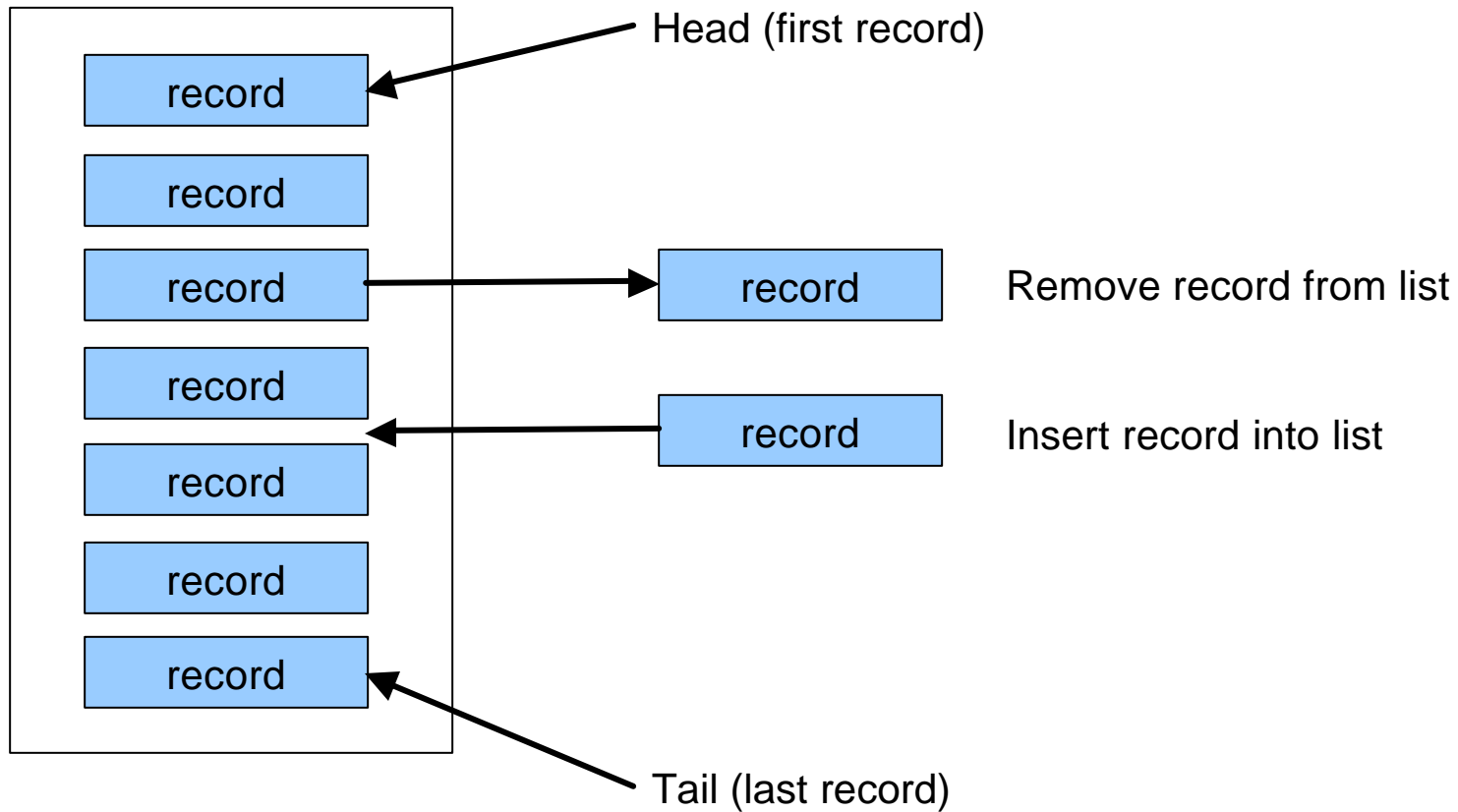


Sample list records for simulation

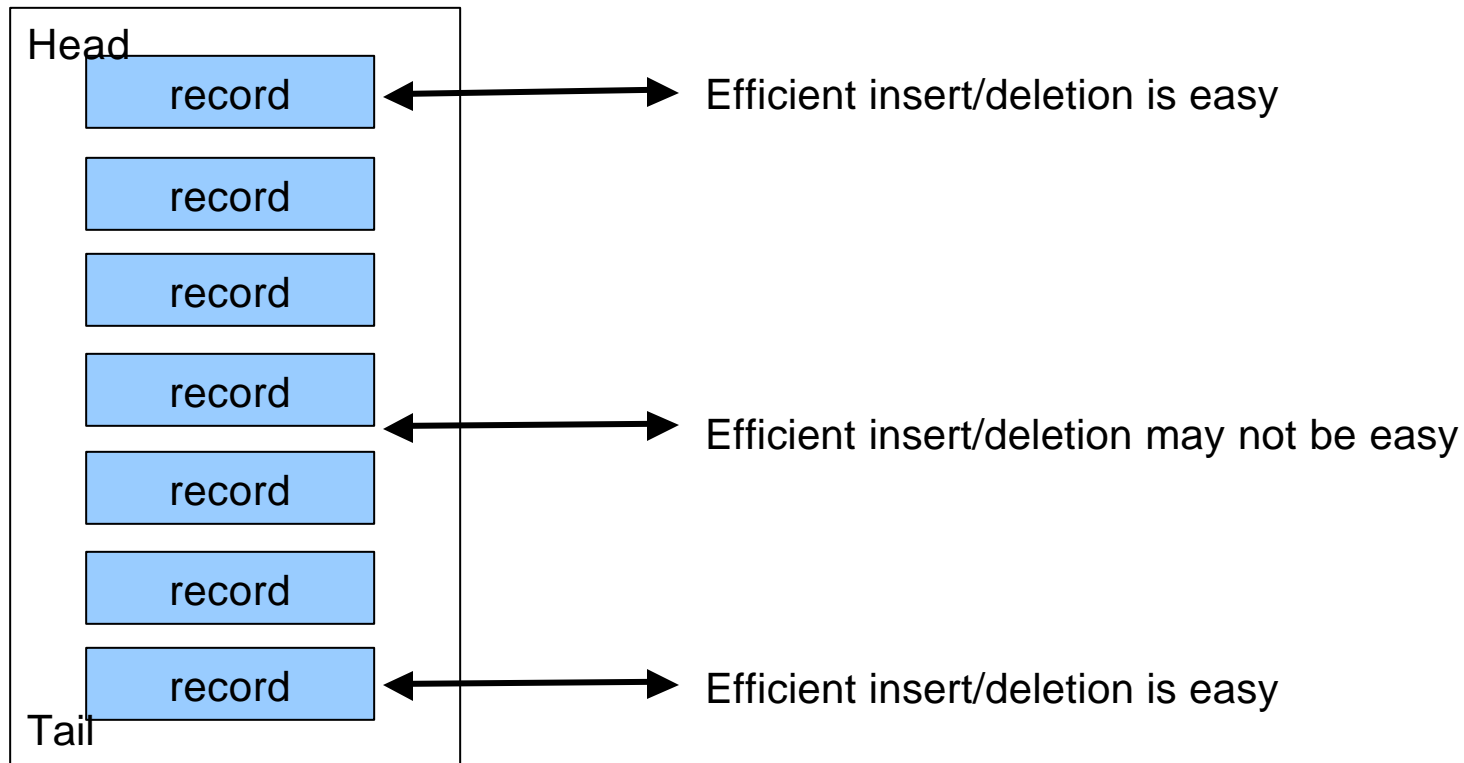


List processing

List actions

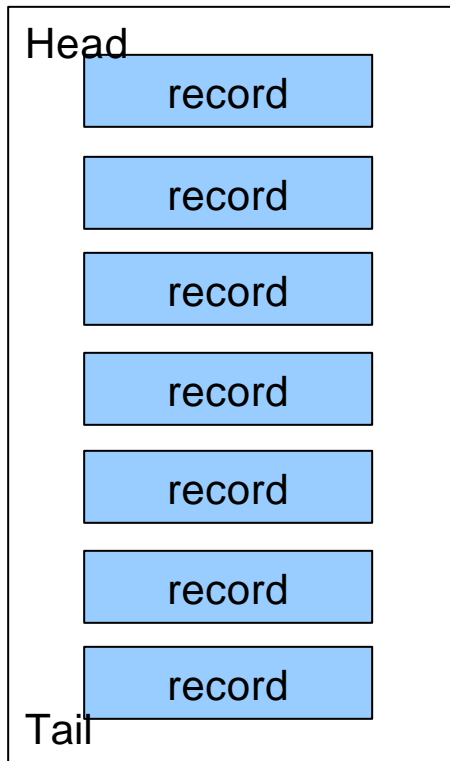


List processing



List processing

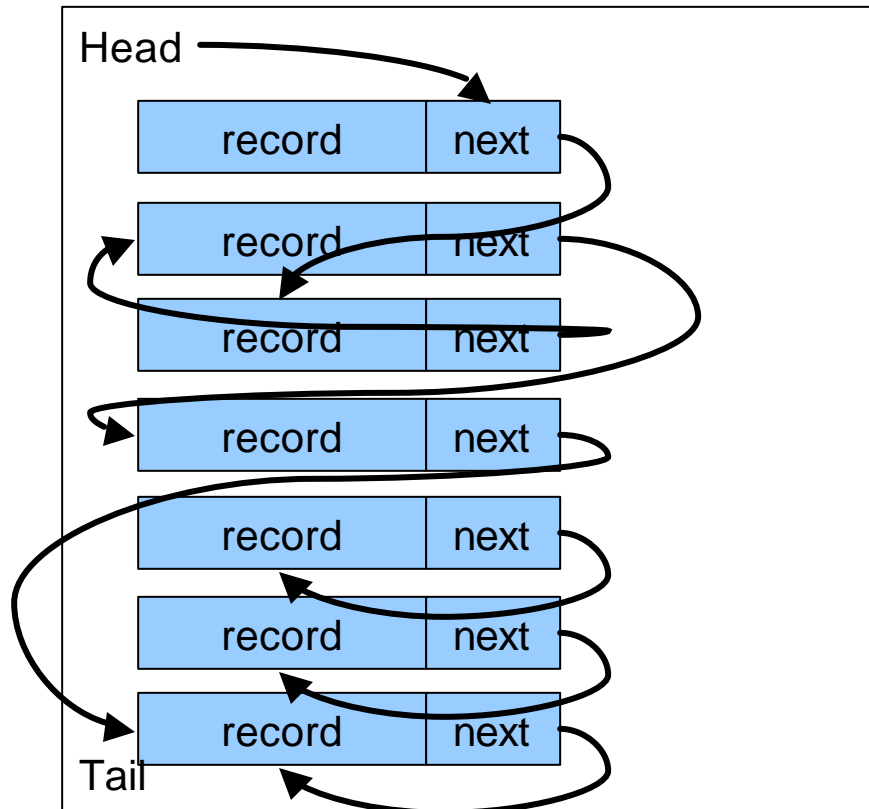
- data structures



- FORTRAN:
 - A list is an array. Successive records are in contiguous locations in memory
 - Shrinking/expanding lists may not make efficient use of memory
- C:
 - Use a structure for a linked list
 - Use malloc() for variable size structures
- C++
 - Use classes for lists
 - Allocate memory as needed
- LISP, SNOBOL, others
 - List processing is an inherent feature of language
- MatLab
 - wasn't designed for list processing, but structured data types simplify list processing
 - memory allocation is automatic

List processing

- data structures, implementation with linked lists



Insertion of record M after record L, before N:
Set M's "next" to record where L pointed
Redirect "next" of record L to M

Deletion of record V, after U, before W:
Change V to point to W, not U
U's "next" is arbitrary. Set to "null", U,
or any other useful value

Given R, finding Q, S, such that $Q < R < S$
may require search of all records up to S

List processing with structures in Matlab

```
FEL(1).event='arrival'  
FEL(1).time=5  
FEL(1).next=13
```

```
FEL(13).event='arrival'  
FEL(13).time=8  
FEL(13).next=5
```

```
FEL(5).event='departure'  
FEL(5).time=9  
FEL(5).next=5
```

List processing enhancements

- A linked list must be searched from the beginning. The average search time to find where to put a record is on the order of $N/2$ for an N record list.
- A doubly linked list includes “previous” record pointers, as well as “next” record pointers.
- Adding a “middle” pointer, besides the “head” and “tail” pointers allows faster traversal of a doubly linked list from the middle element. However, there is additional bookkeeping as records get added/deleted and the list center changes.
- “Hash tables” allow fast indexing into a list with minimal searching.

Dynamic versus Static Memory-Management Tradeoffs for Simulation

- Static:
 - Allocate a fixed maximum amount of storage for simulation data
 - E.g., fixed array
- Dynamic
 - Request memory as needed for simulation data
 - E.g., malloc() or language that dynamically allocates from a “heap”

	Advantages	Disadvantages
Static memory allocation	<ul style="list-style-type: none">• Faster startup• More consistent simulation speed	<ul style="list-style-type: none">• Predetermined maximum list sizes• May require extra time during simulation to reorder records
Dynamic memory allocation	<ul style="list-style-type: none">• No fixed simulation size•	<ul style="list-style-type: none">• Slower startups• Memory fragmentation may require slow “garbage collection”

Homework 3

- Chapter 3, page 94, exercise 4.
- Continue your project started last week. Set objectives and define overall approach.