Discrete Event Simulation

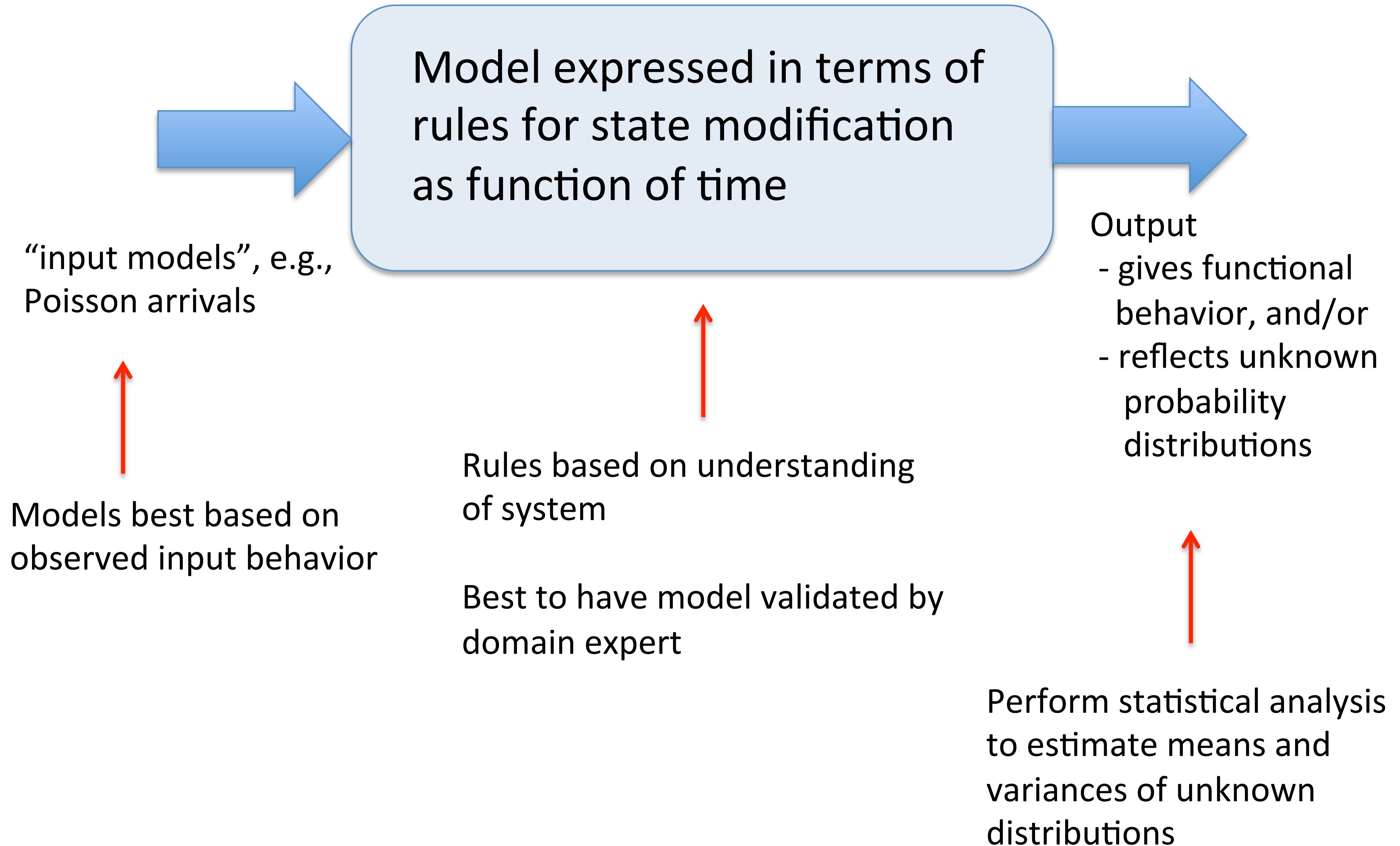
Motivation

- Markovian models can be analyzed and have their state-spaces completely explored when simple/small enough
- When state spaces become too large, discrete event simulation is often a viable alternative.
- Discrete-event simulation can be used to solve models with arbitrarily large state spaces, as long as the desired measure is not based on a "rare event."
- When "rare events" are present, variance reduction techniques can sometimes be used.

Simulation as Model Experimentation

- State-based methods enumerate all possible states a system can be in, and then apply a numerical solution method on the generated state space.
- Simulation generates one or more trajectories (possible behaviors from the high-level model), and collects statistics from these trajectories to estimate the desired performance/ dependability measures.
- Just how this trajectory is generated depends on the:
  - nature of the notion of state (continuous or discrete)
  - type of stochastic process (e.g., ergodic, reducible)
  - nature of the measure desired (transient or steady-state)
  - types of delay distributions considered (exponential or general)

View of Simulation Model



Types of Simulation

Continuous-state simulation is applicable to systems where the notion of state is continuous and typically involves solving (numerically) systems of differential equations. Circuit-level simulators are an example of continuous-state simulation.

Discrete-event simulation is applicable to systems in which the state of the system changes at discrete instants of time, with a finite number of changes occurring in any finite interval of time.

There are two types of discrete-event simulation execution algorithms:

- Fixed-time-stamp advance
- Variable-time-stamp advance

Fixed-Time-Stamp Advance Simulation

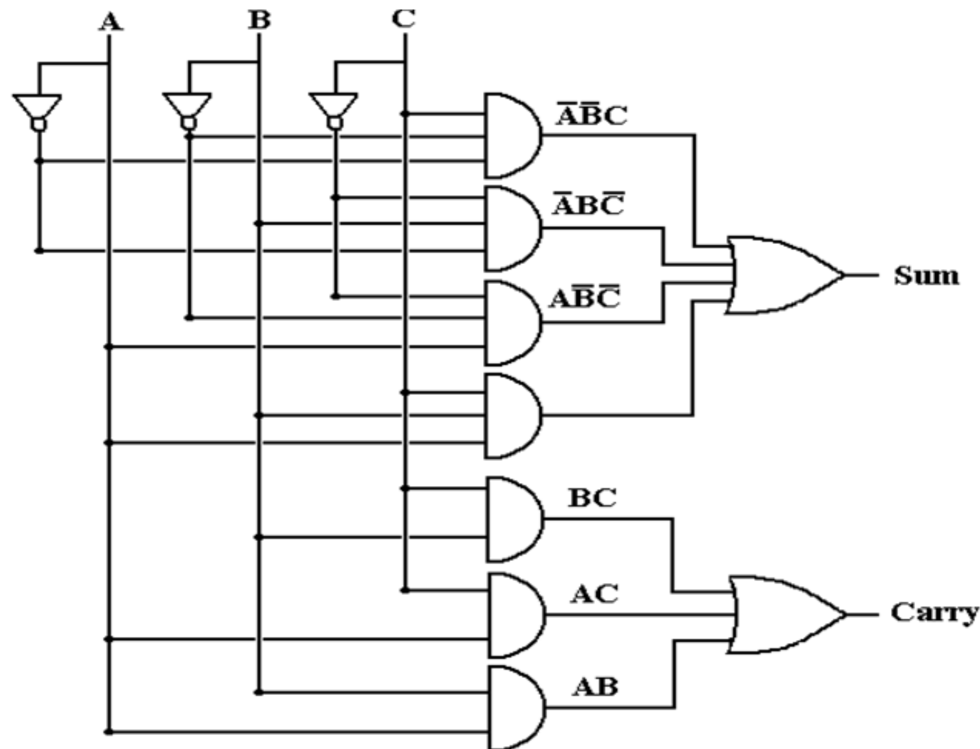
- Simulation clock is incremented a fixed time Δt at each step of the simulation.
- After each time increment, each event type is checked to see if it should have completed during the time of the last increment.
- All event types that should have completed are completed and a new state of the model is generated.
- Rules must be given to determine the ordering of events that occur in each interval of time.
- Example:

- Good for all models where most events happen at fixed increments of time (e.g., gate-level simulations).
- Has the advantage that no "future event list" needs to be maintained.
- Can be inefficient if events occur in a bursty manner, relative to time-step used.

Example of Fixed Time Step Simulation

Gate level simulator

- State updates everywhere each Δ unit of time
  - Simple logic simulator, each Δ a gate delay time
    - Each Δ set gate outputs as function of inputs

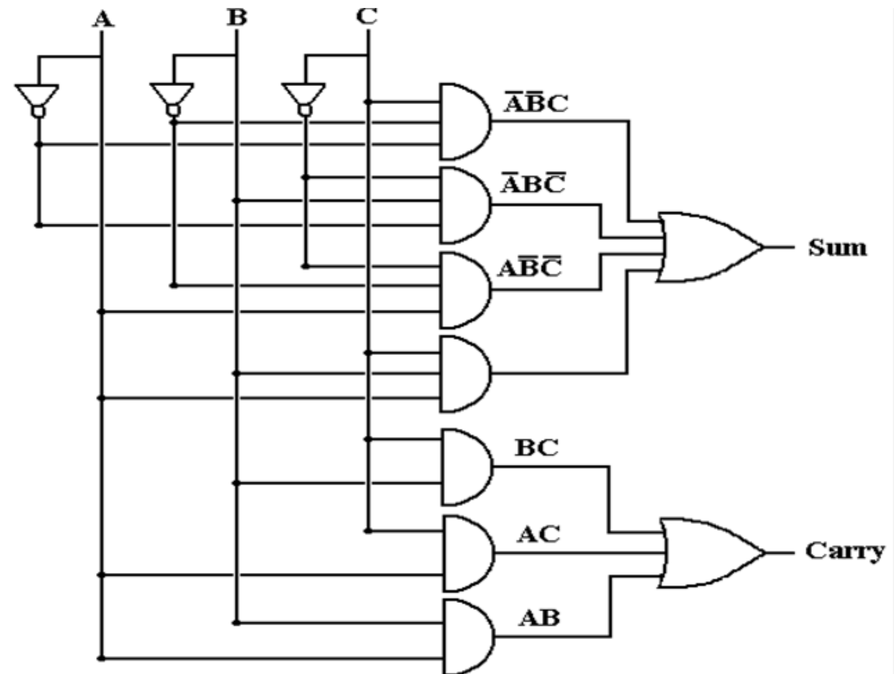


Example of Fixed Time Step Simulation

Focus state changes on instants in time and location in model where state change happens

- Means we *schedule* localized state updates
- This is an "event"
  - Specifies action, location in model, and time

Event = (set value, gate input, t)



Example of Fixed Time Step Simulation

Focus state changes on instants in time and location in model where state change happens

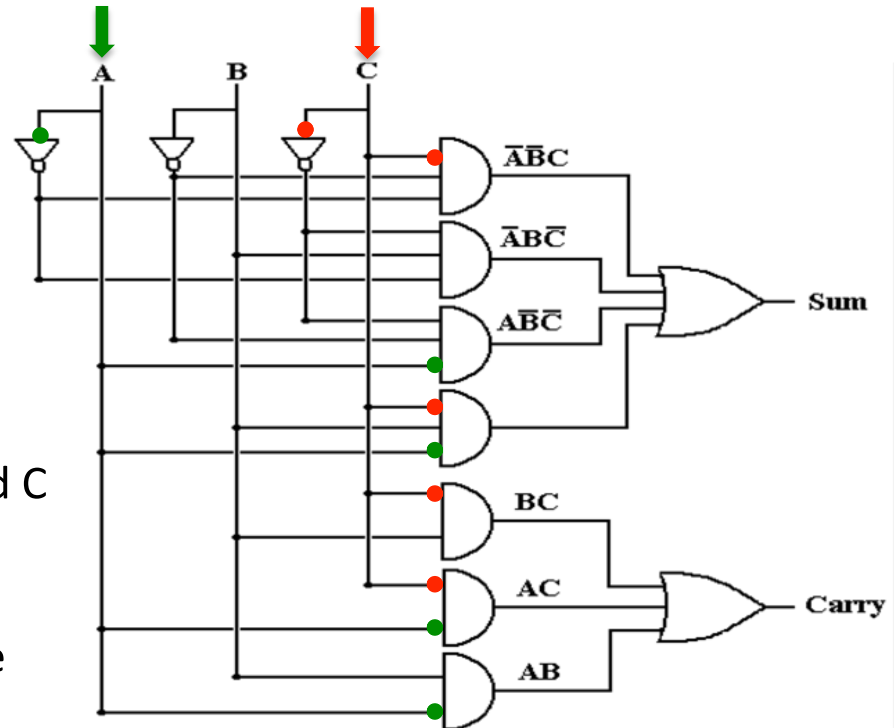
- Means we *schedule* localized state updates
- This is an "event"
  - Specifies action, location in model, and time

Event = (set value, gate input, t)

T = 1000 gate delays

Observe input changes
as result of changes on A and C

Schedule only one gate eval
when multiple inputs change



Example of Fixed Time Step Simulation

Focus state changes on instants in time and location in model where state change happens

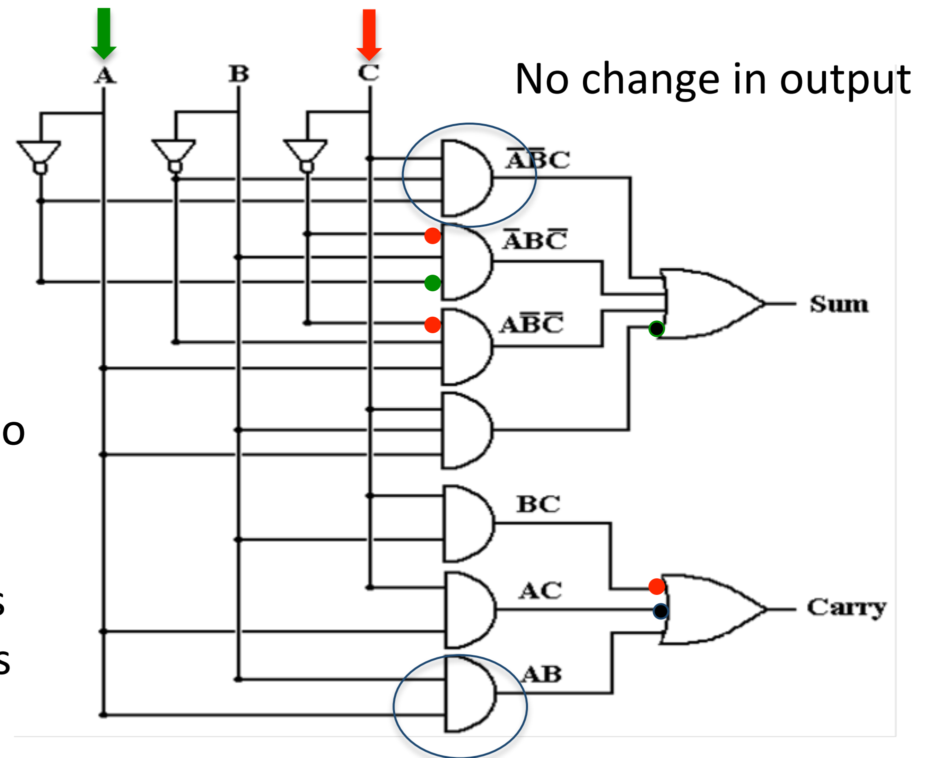
- Means we *schedule* localized state updates
- This is an "event"
  - Specifies action, location in model, and time

Event = (set value, gate input, t)

T = 1001 gate delays

Observe input changes due to Inverter output changes

Observe some input changes Do not cause output changes



Example of Fixed Time Step Simulation

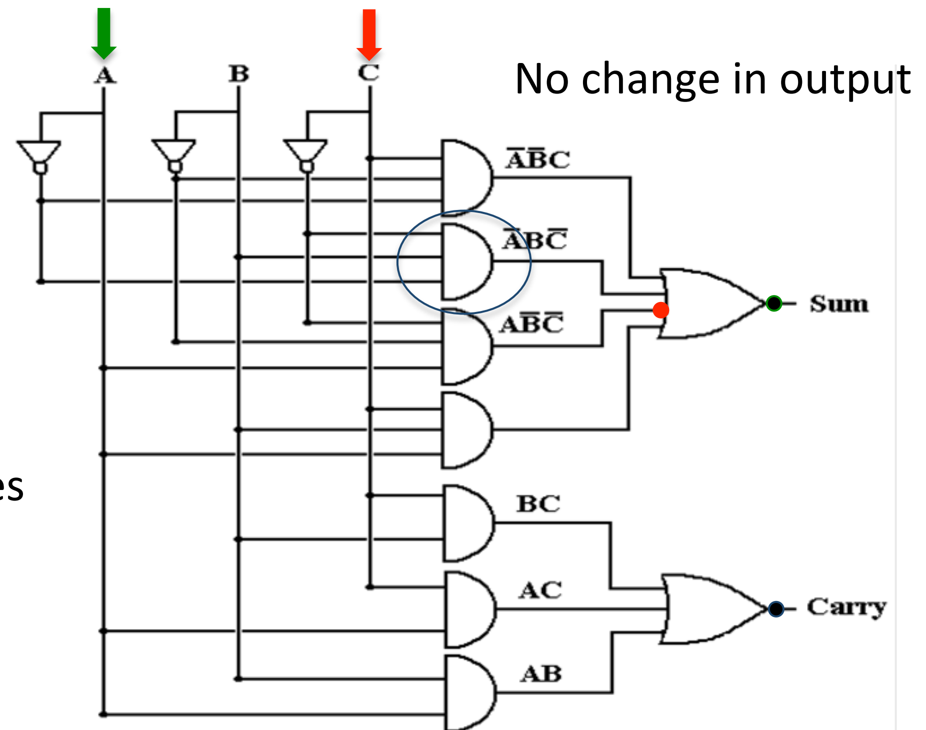
Focus state changes on instants in time and location in model where state change happens

- Means we *schedule* localized state updates
- This is an "event"
  - Specifies action, location in model, and time

Event = (set value, gate input, t)

T = 1002 gate delays

Observe continued selective propagation of signal changes



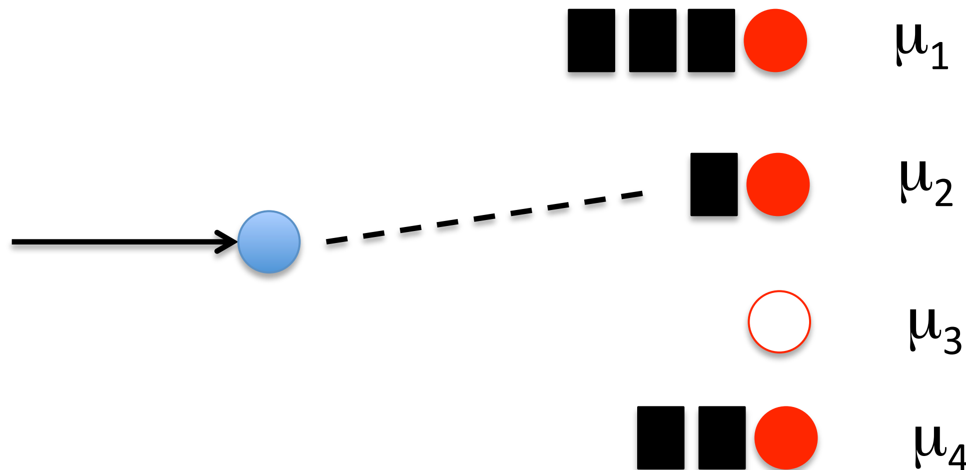
Variable-Time Step Advance Simulation

- Simulation clock advanced a variable amount of time each step of the simulation, to time of next event.
- If event times are general (have memory) then "future event list" is needed.
- Has the advantage (over fixed-time-stamp increment) that periods of inactivity are skipped over, and models with a bursty occurrence of events are not inefficient.

Example of Variable Time Step Simulation

Random inter-arrivals to multi-queue system

- Arrival sent to queue with shortest expected finishing time (number in system*(mean service)
- Service distributions different queues vary



Queueing Example #1

1. First schedule an arrival to the system
   – (arrival, system, t)
2. Choose "next event" in simulation time
   – Implicitly, model state remains untouched in time since most recent event

Processing of Arrival to System (arrival, system, t)

a) Find queue Q with least **expected** finishing time if job->Q
   (assuming FCFS queue management, service not yet known)
   If Q not empty then add job to tail of Q
   else
      sample service time s0
      schedule departure event (departure, Q, t+s0)

b) schedule next arrival to system
   sample inter-arrival time a0
   schedule (arrival, system, t+a0)

Queueing Example #1

3. Next event is either (departure, Q, t+s0) or (arrival,system,t+a0)

Suppose t+s0 < t+a0

Processing of (departure, Q, t')

- compute statistics associated with departing job
- remove job from Q
- if Q not empty
  - schedule next departure
    - sample service time s1
    - schedule (departure, Q, t'+s1)

Event View of DES

Key concepts

- Event specifying what/where/when to do
- Objects where activities occur
- Event processing logic
  - Changes state
  - Schedules future events
- Event list
  - Data structure supporting efficient
    - » Insertion of new events
    - » Removal of event with least time-stamp
    - » Deletion of events

Queueing Example #2

Same as previous example, except

- Jobs arrive to queue with pre-declared service requirements
- Queueing policy at each queue is to give service to the job with least remaining service time (shortest-job-first preemptive-resume)
  - Implies that a job in service may be suspended

Changes to previous logic

- Event data structure should identify job with known (residual) service time

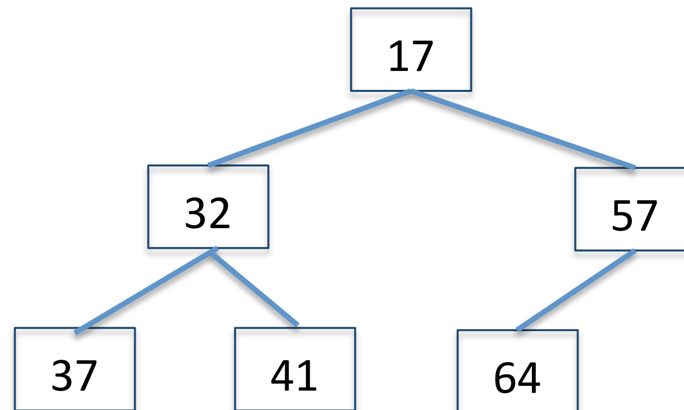
  e.g., (arrival, system, job0, t), (departure, Q, jobk, t')
- Computation of expected finishing time (and selection of Q) based on queueing policy
- Processing of job arrival to system needs to allow for interrupting a job in service
  - Means **cancelling** a scheduled departure event (departure, Q, jobk, T)

Event List data structures

- Sorted list of events
  - Get minimum event in O(1) time, insertion is O(N)
  - Splay-tree
    - All operations are O(log N)
  - Calendar queue
    - With right parameters, O(1) operations
      - Degrades with wrong parameters
  - Binary heap

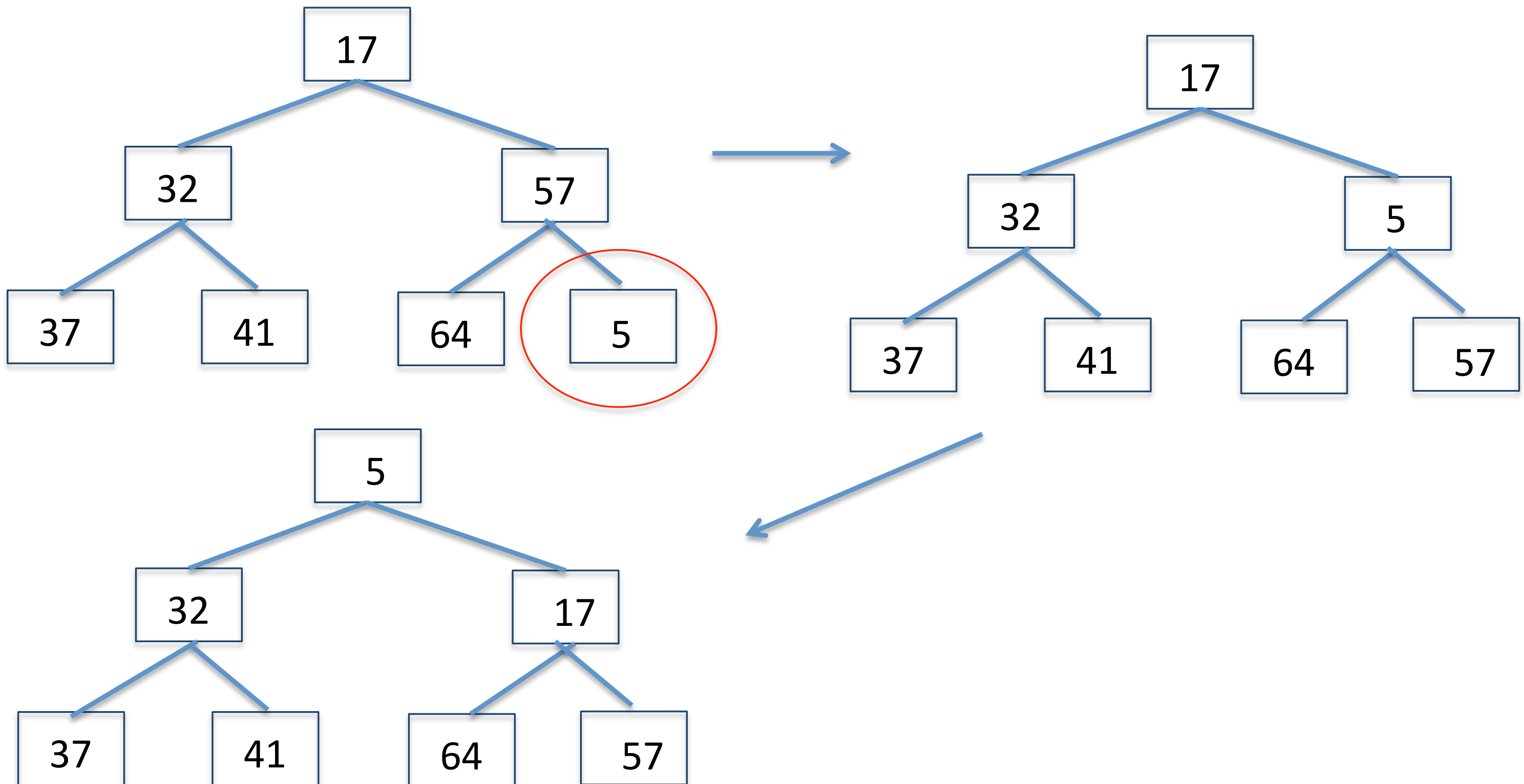
Event List data structure in python

- Basic idea is binary min-heap, implemented in python with heapq
- Constant time identification of event with least time-stamp
- Logarithmic time cost for inserting event
- Maintaining 'heap property'
  - Key of parent node always no smaller than key of child node



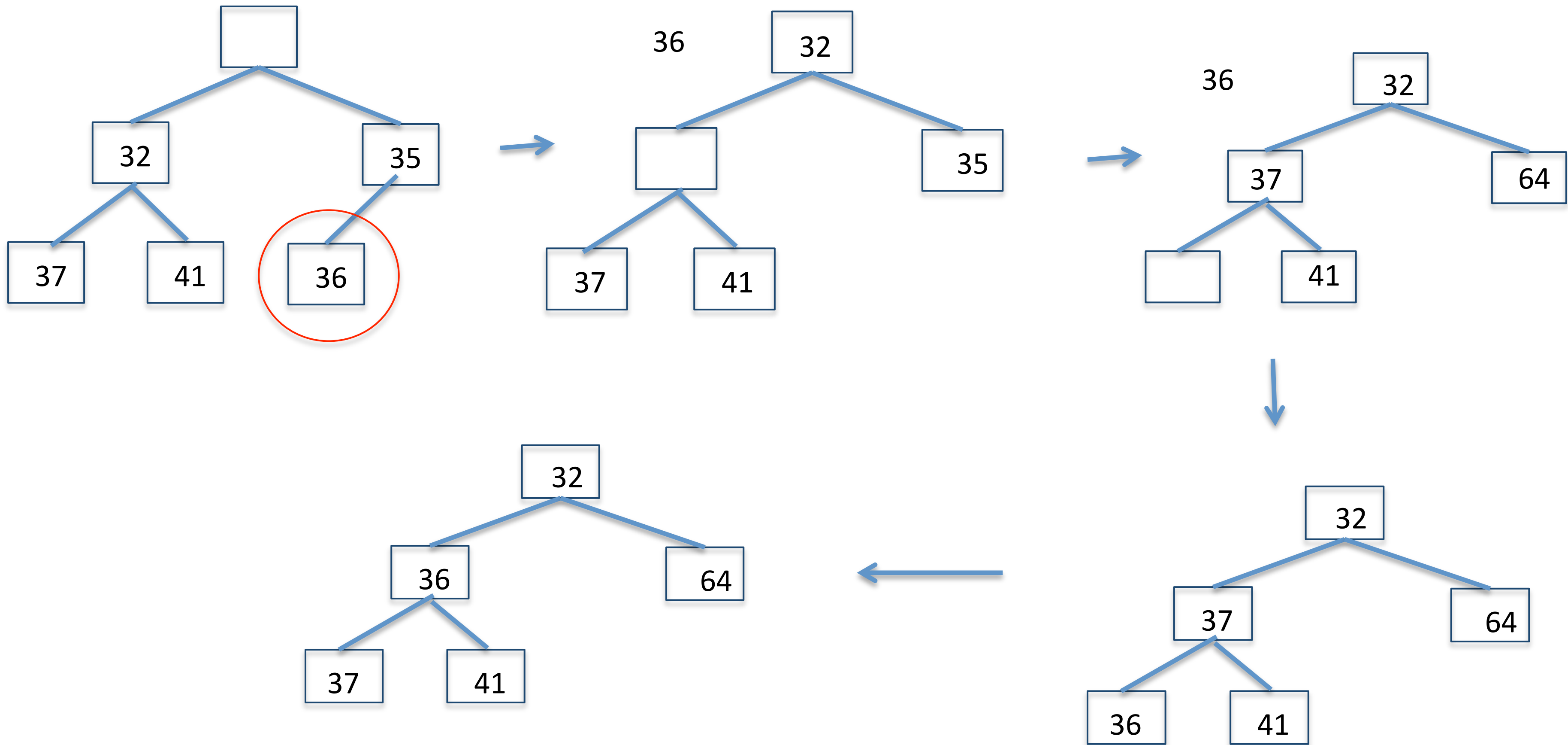
Event List data structure in python

- Adding element
  - Put new element in ‘last place’, then swap child and parent towards root to maintain heap property



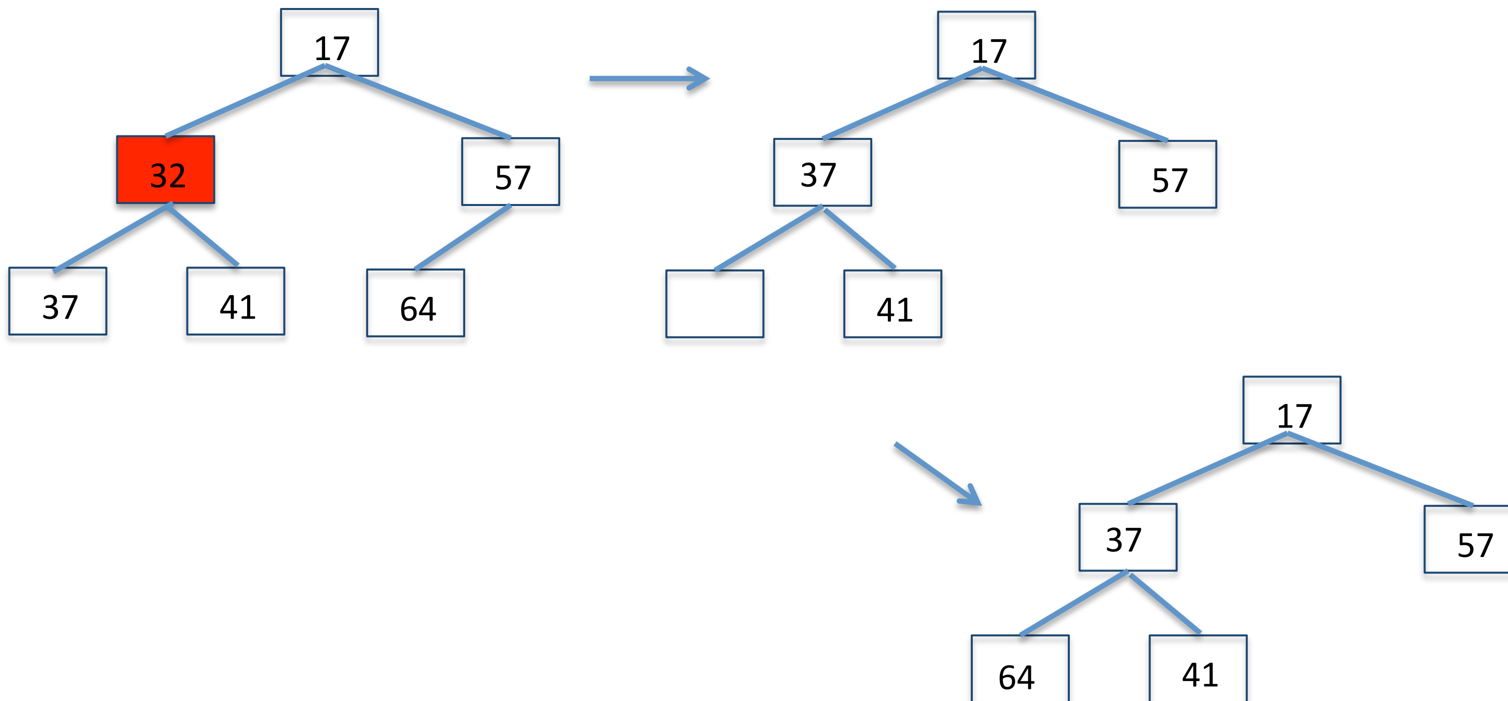
Event List data structure in python

- Removing front element
  - Take last item from heap, move smaller children up to maintain heap property
  - Insert formerly last item from heap



Event List data structure in python

- Removing interior element
  - Same operation as removing root, but removing root of subtree whose root is the deleted location



Implementing Event Cancellation

An event in a min-heap will move position as the simulation progresses

- Need a means of quickly identifying where it is in the heap

sim_evt.py has code for

- Creating an event list
- Adding an event to the event list
- Removing least-time event from the event list
- Cancelling a previously added event from the event list
- Modified heapq to support cancellation

For event cancellation the API returns a code for a scheduled event, which is offered when that event is to be cancelled

In the weeds

Go through sim_evt.py

Application to a model

Go through balanced_queue.py