Ch.2 Discrete Event Simulation Fundamentals

黃俊堯 副教授

台北大學 資訊工程學系

E-mail : jyhuang@mail.ntpu.edu.tw

Contents(1)

2.1 Time

2.2 Real-Time, Scaled Real-Time, and As-Fast-As-Possible Execution

2.3 State Changes and Time Flow Mechanisms

2.3.1 Time-Stepped Execution

2.3.2 Event-Driven Execution

2.4 Discrete-Event Simulation Programs

2.5 An Example Application

2.6 Starting and Stopping the Simulation

Contents

2.7 Parallel/Distributed Simulation Example

2.8 World Views and Object-Oriented Simulation

- 2.8.1 Simulation Processes
- 2.8.2 Object-Based and Object-Oriented Simulation
- 2.8.3 Query Events and Push versus Pull Processing
- 2.8.4 Event Retraction

2.9 Other Approaches to exploiting Concurrent Execution

Simulation Fundamentals

- A computer simulation is a computer program that models the behavior of a physical system over time.
  - Program variables (state variables) represent the current state of the physical system
  - Simulation program modifies state variables to model the evolution of the physical system over time.

What is Simulation?

¢ The simulation must provide :

(1)A representation of the state of the physical system

– Computer simulations define a collection of *state variables*

(2)Some means of changing this representation to model the evolution of the physical system

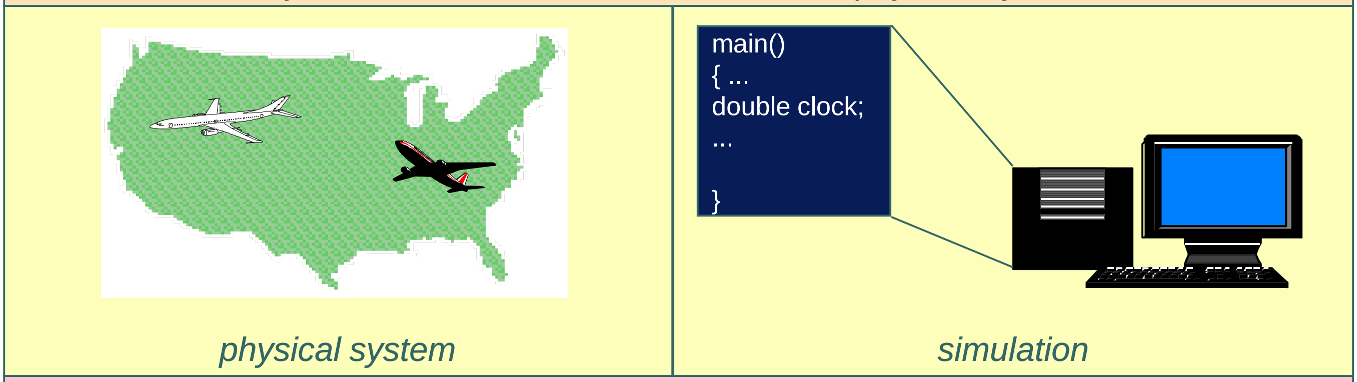
– Changes in the state of physical system are realized by simulation program writing new values into these state variables

(3)Some representation of time

– Time in the physical system is represented through an abstraction called *simulation time*

2.1 Time

- *physical system*: the actual or imagined system being modeled
- *simulation*: a system that emulates the behavior of a physical system



```
main()
{ ...
double clock;
...

}
```

¢ *physical time*: time in the physical system

- Noon, December 31, 1999 to noon January 1, 2000

¢ *simulation time*: representation of physical time within the simulation

- floating point values in interval [0.0, 24.0]

¢ *wallclock time*: time during the execution of the simulation, usually output from a hardware clock

- 9:00 to 9:15 AM on September 10, 1999

2.1 Simulation Time

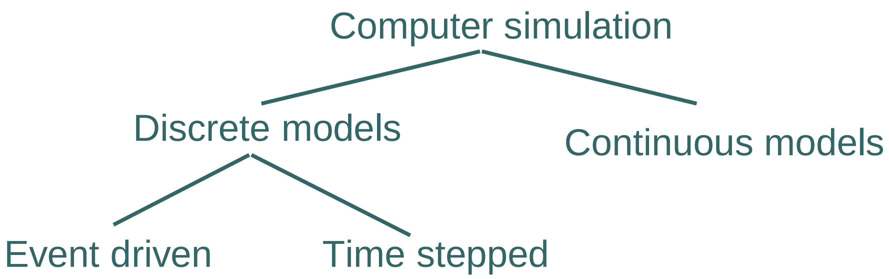
- Simulation time is defined as a totally ordered set of values where each value represents an instant of time in the physical system being modeled.
- For any two values of simulation time T_1 representing instant P_1, and T_2 representing P_2:
  - Correct ordering of time instants
    - If $T_1 < T_2$, then P_1 occurs before P_2
    - 9.0 represents 9 PM, 10.5 represents 10:30 PM
  - Correct representation of time durations
    - $T_2 - T_1 = k (P_2 - P_1)$ for some constant k
    - 1.0 in simulation time represents 1 hour of physical time

2.2 Paced vs. Unpaced Execution

- Mode of Execution
  - *Real-time* execution (paced): each advance in simulation time is paced to occur in synchrony with an equivalent advance in wallclock time
  - *Scaled real-time* execution (paced): each advance in simulation time is paced to occur in synchrony with S * an equivalent advance in wallclock time (e.g., 2x wallclock time)
  - *As-fast-as-possible* execution (unpaced): no fixed relationship necessarily exists between advances in simulation time and advances in wallclock time
- Simulation Time = W2S(W) = $T_0 + S * (W - W_0)$
  - W = wallclock time;   S = scale factor
  - W_0 (T_0) = wallclock (simulation) time at start of simulation

(assume simulation and wallclock time use same time units)

Simulation Taxonomy


Computer simulation

Discrete models

Continuous models

Event driven

Time stepped

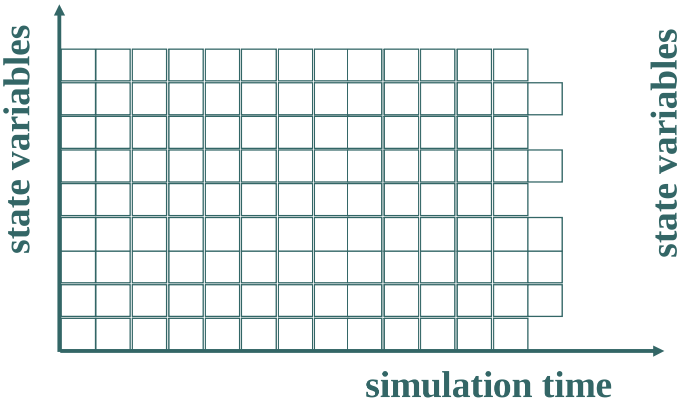
- Continuous time simulation
  - State changes occur continuously across time
  - Typically, behavior described by differential equations
- Discrete time simulation
  - State changes only occur at discrete time instants
  - Time stepped: time advances by fixed time increments
  - Event stepped: time advances occur with irregular increments

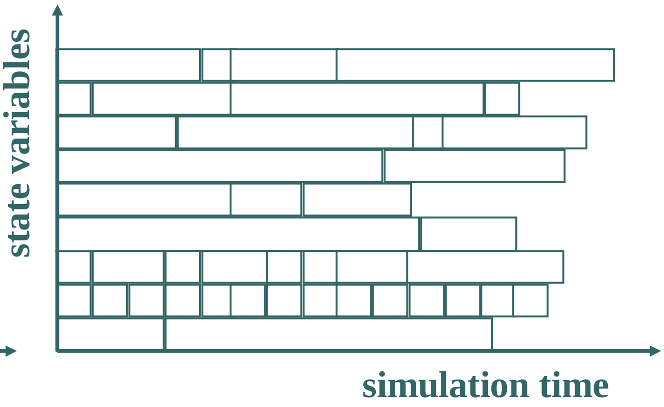
Event-Driven vs. Time Stepped

- Event:
  - State variable is updated when “something interesting”, the “something interesting” occurs is referred to as an *event*
- Event-Driven:
  - Simulation time advance from the time stamp of one event to the next
- Event-Driven vs. Time Stepped
  - An event-driven simulation can emulate a time-stepped simulation by defining events that happen at each time step
  - A time-stepped simulation may emulate an event-driven simulation by defining a time step size that is the greatest common divisor among all the time stamps assigned to events in the simulation

Time-Stepped vs. Event-Driven

Goal: compute state of system over simulation time





Time-stepped execution

Event-driven execution

Time Stepped Execution

- Simulation time is subdivided as a sequence of equal-sized time steps, and the simulation advances from one time step to the next
- Action in the simulation occurring in the same time step are usually considered to be simultaneous, and are often assumed not to have an effect on each other

Time Stepped Execution (Paced)

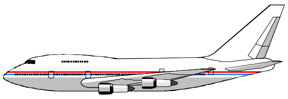
```
While (simulation not completed)
{
    Wait Until (W2S(wallclock time) $\geq$ current simulation
      time)
    Compute state of simulation at end of this time step
    Advance simulation time to next time step
}
```

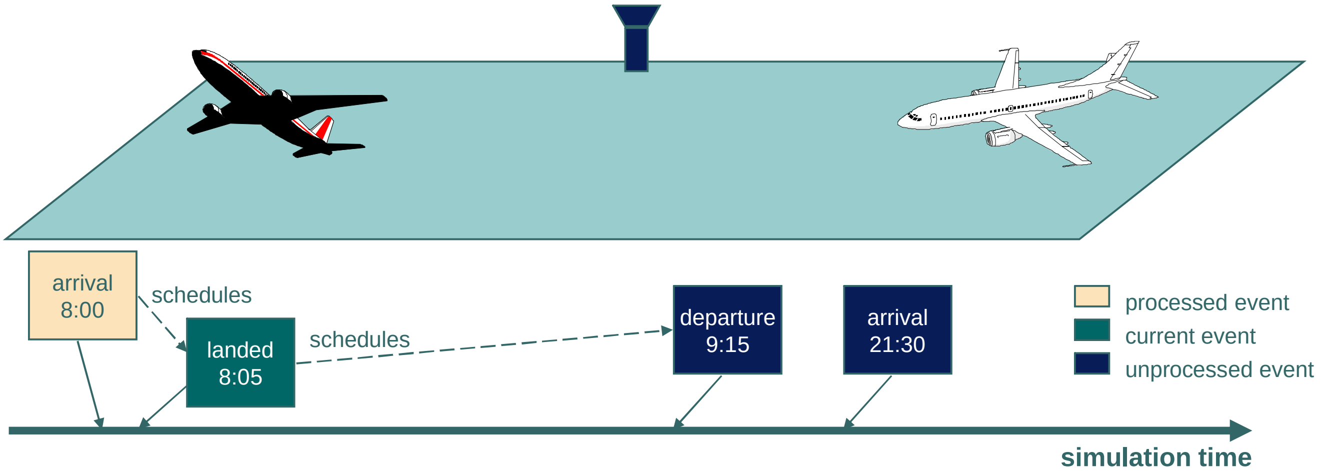
2.4 Discrete Event Simulation

- Discrete event simulation: computer model for a system where changes in the state of the system occur at *discrete* points in simulation time.
- Fundamental concepts:
  - system state (state variables)
  - state transitions (events)
- A DES computation can be viewed as a sequence of event computations, with each event computation is assigned a (simulation time) time stamp
- Each event computation can
  - modify state variables
  - schedule new events

Discrete Event Simulation Computation

example: air traffic at an airport
events: aircraft arrival, landing, departure





arrival 8:00

schedules

landed 8:05

schedules

departure 9:15

arrival 21:30

□ processed event

🟩 current event

■ unprocessed event

simulation time

- Unprocessed events are stored in a pending event list
- Events are processed in time stamp order

Discrete-Event Simulation Program(1)

- A sequential discrete event simulation program typically utilizes three data structure:
  1. The *state variables* that describe the state of the system
  2. An *event list* containing events that are to occur some time in simulated future.
  3. A *global clock* variable to denote the instant on the simulation time axis at which the simulation now reside

Discrete-Event Simulation Program(2)

- All events in the event list must have a *time stamp* greater than or equal to T, which represents time of physical system
- In simulation world, nothing happens unless the simulation computation makes, so require a mechanism to create new events. The mechanism for creating a new event in the simulation is called *scheduling*
- Event scheduling is one way that simulation programs can model causal relationships in the physical system

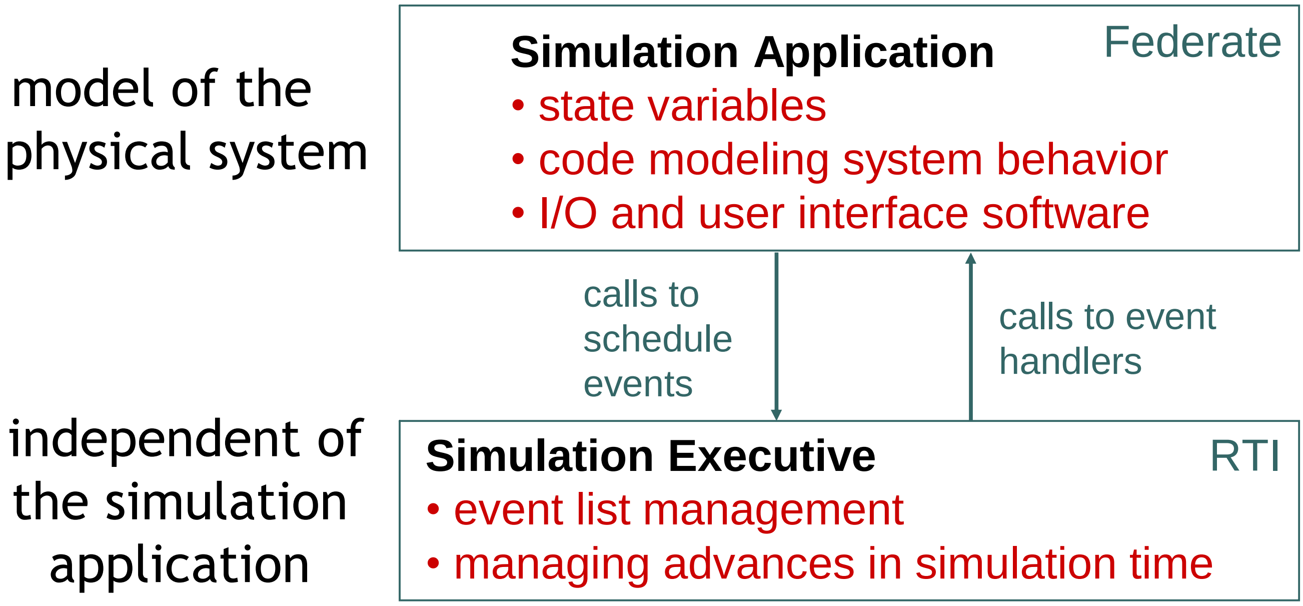
Discrete-Event Simulation Programs(3)

- The procedure for the event may do two things:
  - Modify state variables to model changes in the state of the physical system that result from this event
  - Schedule new events into the simulated future
- Important Point:
  - Only schedule events into the simulation future
  - Always processes the event containing the smallest time stamp next.

Event-Processing Loop

- While (simulation is in progress)
  - Remove smallest time stamped event from event list
  - Set simulation time clock to time stamp of this event
  - Execute event handler in application to process event

Discrete Event Simulation System



model of the physical system

Simulation Application Federate

- state variables
- code modeling system behavior
- I/O and user interface software

calls to schedule events

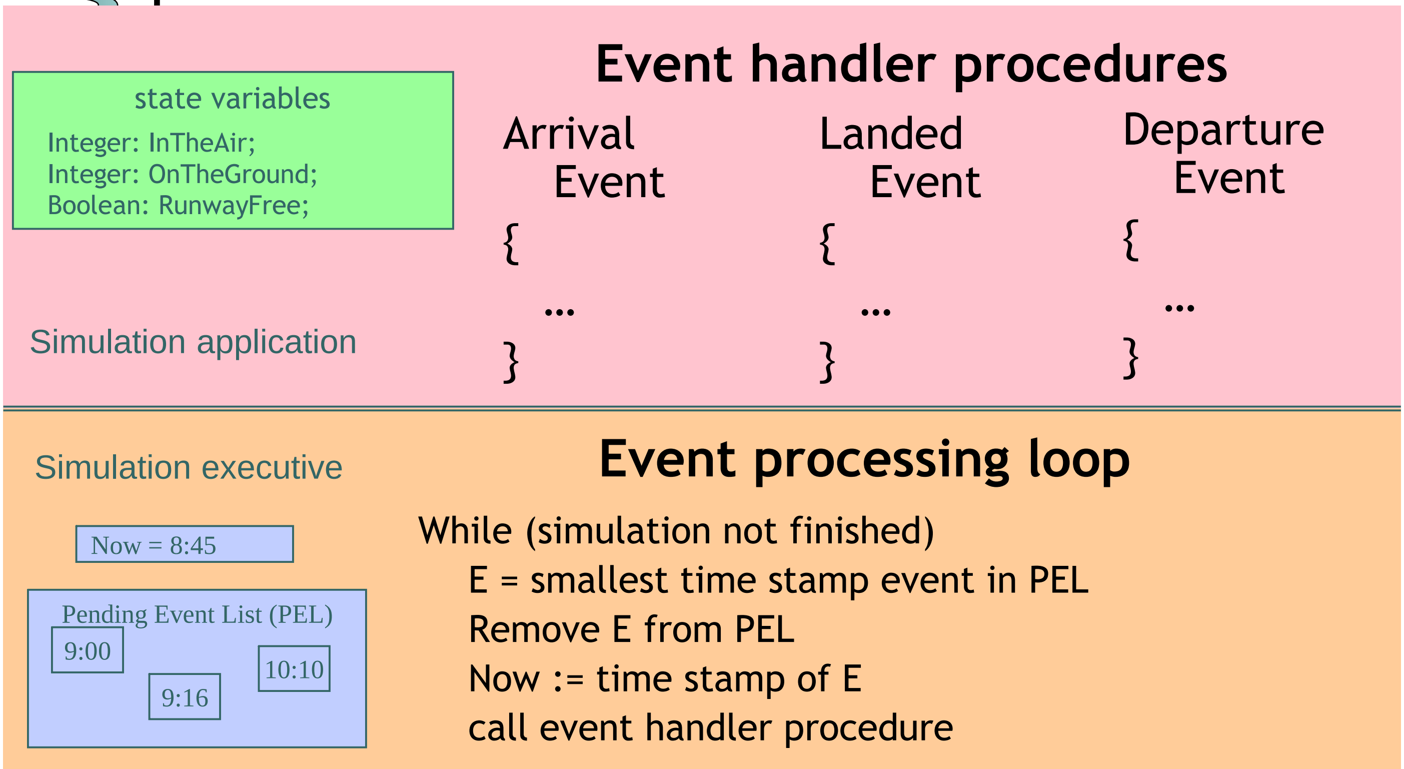
calls to event handlers

independent of the simulation application

Simulation Executive RTI

- event list management
- managing advances in simulation time

Event-Oriented World View



Event handler procedures

state variables

Integer: InTheAir;
Integer: OnTheGround;
Boolean: RunwayFree;

Simulation application

```
Arrival
  Event
{
  …
}
```

```
Landed
  Event
{
  …
}
```

```
Departure
  Event
{
  …
}
```

Simulation executive

Event processing loop

Now = 8:45

Pending Event List (PEL)

9:00

9:16

10:10

```
While (simulation not finished)
    E = smallest time stamp event in PEL
    Remove E from PEL
    Now := time stamp of E
    call event handler procedure
```

2.5 Example: Air traffic at an Airport

Model aircraft arrivals and departures, arrival queueing

<u>Single runway</u> for incoming aircraft, ignore departure queueing

- R = time runway is used for each landing aircraft (constant)
- G = time required on the ground before departing (constant)

State:

- Now: current simulation time
- InTheAir: number of aircraft landing or waiting to land
- OnTheGround: number of landed aircraft
- RunwayFree: Boolean, true if runway available

Events:

- Arrival: denotes aircraft arriving in air space of airport
- Landed: denotes aircraft landing
- Departure: denotes aircraft leaving

Arrival Events

New aircraft arrives at airport. If the runway is free, it will begin to land. Otherwise, the aircraft must circle, and wait to land.

- R = time runway is used for each landing aircraft
- G = time required on the ground before departing
- Now: current simulation time
- InTheAir: number of aircraft landing or waiting to land
- OnTheGround: number of landed aircraft
- RunwayFree: Boolean, true if runway available

Arrival Event:

```
InTheAir := InTheAir+1;
If (RunwayFree)
    RunwayFree:=FALSE;
    Schedule Landed event @ Now + R;
```

Landed Event

An aircraft has completed its landing.

- R = time runway is used for each landing aircraft
- G = time required on the ground before departing
- Now: current simulation time
- InTheAir: number of aircraft landing or waiting to land
- OnTheGround: number of landed aircraft
- RunwayFree: Boolean, true if runway available

Landed Event:

```
InTheAir:=InTheAir-1;
OnTheGround:=OnTheGround+1;
Schedule Departure event @ Now + G;
If (InTheAir>0)
    Schedule Landed event @ Now + R;
Else
    RunwayFree := TRUE;
```

Departure Event

An aircraft now on the ground departs for a new destination.

- R = time runway is used for each landing aircraft
- G = time required on the ground before departing
- Now: current simulation time
- InTheAir: number of aircraft landing or waiting to land
- OnTheGround: number of landed aircraft
- RunwayFree: Boolean, true if runway available

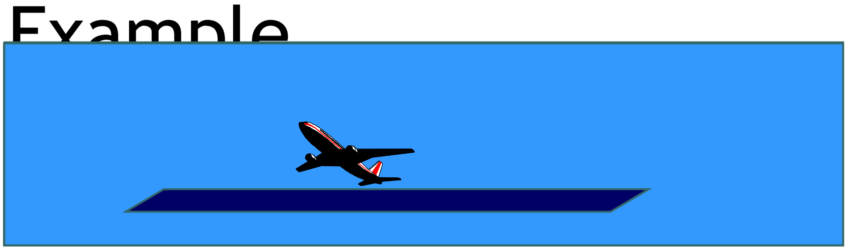
Departure Event:

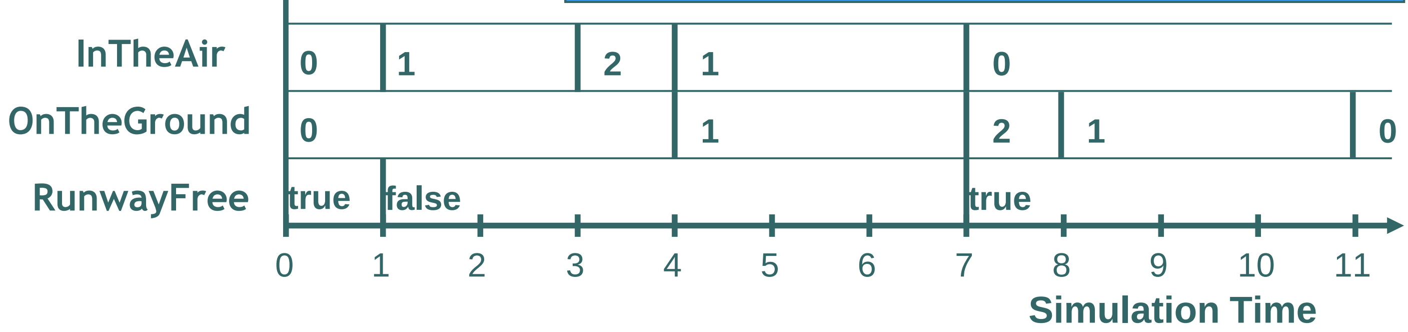
OnTheGround := OnTheGround - 1;

Execution Example

State Variables

R=3
G=4





Processing:

Time	Event
1	Arrival F1
3	Arrival F2

Now=0

Arrival F1

Time	Event
3	Arrival F2
4	Landed F1

Now=1

Arrival F2

Time	Event
4	Landed F1
7	Landed F2

Now=3

Landed F1

Time	Event
7	Landed F2
8	Depart F1

Now=4

Landed F2

Time	Event
8	Depart F1
11	Depart F2

Now=7

Depart F1

Time	Event
11	Depart F2

Now=8

Depart F2

Time	Event

Now=11

2.6 Starting and Stopping the Simulation

- Simulation begins by initializing the state variables, and generating initial event
  - The initial events may be created by defining an “initialization event” with time stamp equal to a simulated time prior to beginning of the actual simulation
- There are several techniques for terminating the execution of the simulation
  - A ”stop simulation” event may be used that is defined to be the last event processed by the simulation
  - An “end simulation time” may be defined that indicates the simulation is terminated when the simulation clock is about to exceed this time

2.7 Parallel/Distributed Simulation Example(1)

- *Logical process – LP*
  - A parallel or a distributed simulation is typically composed of a collection of sequential simulations, each modeling a different part of the physical system and executing on a different processor. It refer to each sequential simulation as a *simulator*
  - Thus the physical system can be viewed as a collection of *physical processes* that interacts in some fashion with each physical process being modeled by a logical process
- When an event that is generated within one LP is relevant to one or more other LPs, a *message* is sent to the other LPs to notify them of the event

2.7 Parallel/Distributed Simulation Example(2)

Modified departure event procedure for distributed simulation.

- Source = ID of this airport
- Dest = ID of destination airport
- Flight_Time[S,D] = time to fly from S to D

Departure Event:

OnTheGround := OnTheGround - 1;

Send message to Dest to schedule an arrival event at time Now+Flight_Time[Source,Dest]

2.8 World Views And Object-Oriented Simulation

- 2.8.1 Simulation Process
- 2.8.2 Object-Based and Object-Oriented Simulations
- 2.8.3 Query Events and Push versus Pull Processing
- 2.8.4 Event Retraction

2.8.1 Simulation Processes(1)

- A simulation process is intended to model a specific entity in the simulation with a well-defined behavior. The behavioral description of the entity is encapsulated by the process
- Simulation Primitives :

  AdvanceTime(*T*): advance *T* units of simulation time
  - Also called "hold" *resource*
  - ex: AdvanceTime(R) to model using runway R units of simulation time

  WaitUntil(*p*): simulation time advances until predicate *p* becomes true
  - Predicate based on simulation variables that can be modified by other simulation processes
  - ex: WaitUntil(RunwayFree) to wait until runway becomes available for landing

2.8.1 Simulation Processes(2)

Identify computation associated with each simulation event

```
/* simulate aircraft arrival, circling, and landing */
InTheAir := InTheAir + 1;
WaitUntil (RunwayFree);      /* circle */
```

Aircraft Arrival

```
RunwayFree := FALSE;         /* land */
AdvanceTime(R);
```

Aircraft Landing

```
RunwayFree := TRUE;

/* simulate aircraft on the ground */
InTheAir := InTheAir - 1;
OnTheGround := OnTheGround + 1;
AdvanceTime(G);
```

Aircraft On The Ground

```
/* simulate aircraft departure */
OnTheGround := OnTheGround - 1;
```

Aircraft Departs

2.8.1 Simulation Processes(3)

Process-oriented simulations are built over event oriented simulation mechanisms (event list, event processing loop)

- Event computation: computation occurring at an instant in simulation time
  - Execution of code section ending with calling a primitive to advance simulation time
- Computation threads
  - Typically implemented with co-routine (threading) mechanism
- Simulation primitives to advance time
  - Schedule events
  - Event handlers resume execution of processes (called “wake up”)

    If there are several that are eligible to execute, the simulation executive must use some prioritization rule to determine which process should be resumed

2.8.1 Simulation Processes(4)

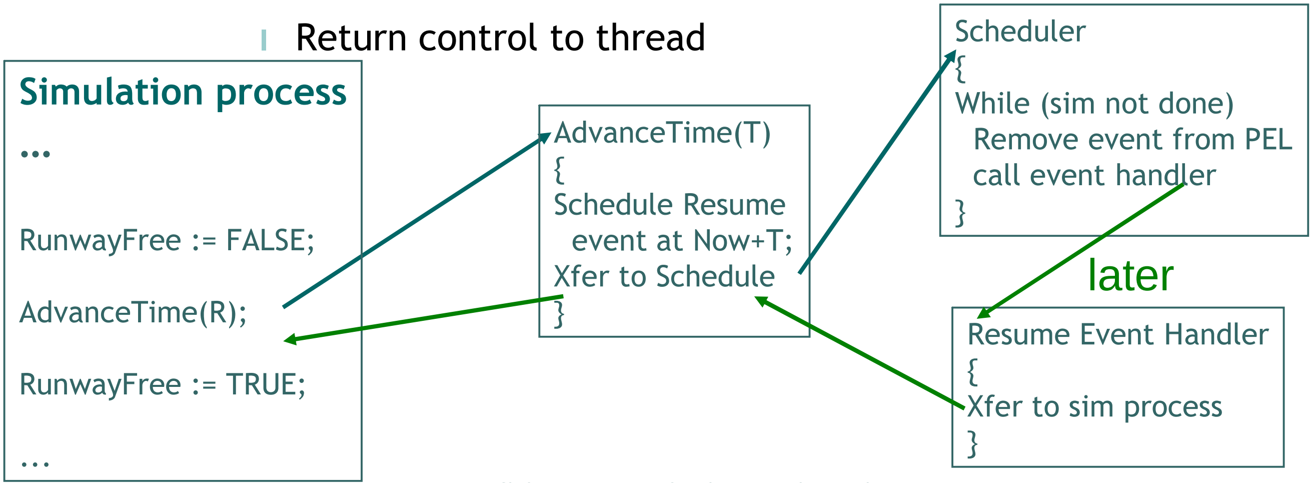
Causes simulation time in the process to advance by T units

Execute AdvanceTime(T):

- Schedule Resume event at time Now+T
- Suspend execution of thread
- Return execution to event scheduler program

Process Resume event:

- Return control to thread



Simulation process
…
RunwayFree := FALSE;
AdvanceTime(R);
RunwayFree := TRUE;
…

AdvanceTime(T)
{
Schedule Resume
event at Now+T;
Xfer to Schedule
}

Scheduler
{
While (sim not done)
Remove event from PEL
call event handler
}

later

Resume Event Handler
{
Xfer to sim process
}

2.8.1 Simulation Processes(5)

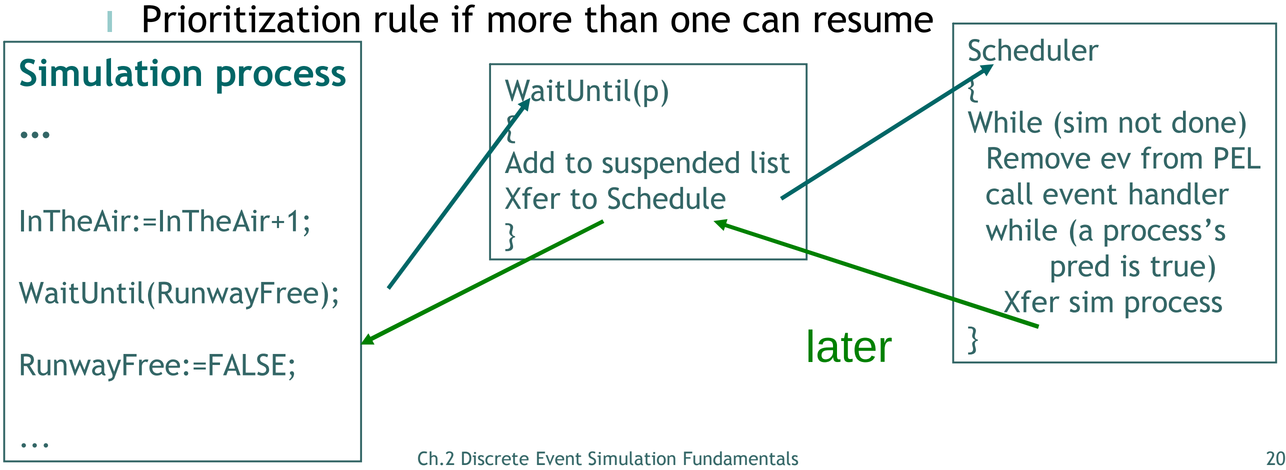
Suspend until predicate p evaluates to true

Execute WaitUntil (p):

- Suspend execution of thread, record waiting for p to become true
- Return execution to event scheduler program

Main scheduler loop

- For each suspended process, check if execution can resume
- Prioritization rule if more than one can resume



Simulation process

```
…

InTheAir:=InTheAir+1;

WaitUntil(RunwayFree);

RunwayFree:=FALSE;

…
```

```
WaitUntil(p)
{
Add to suspended list
Xfer to Schedule
}
```

```
Scheduler
{
While (sim not done)
  Remove ev from PEL
  call event handler
  while (a process's
        pred is true)
    Xfer sim process
}
```

2.8.1 Simulation Processes(6)

- The principal points concerning process-oriented simulations are:
  1. They provide a more convenient paradigm for developing simulation applications for certain types of application
  2. They can be implemented on top of the basic event-oriented style of execution that was described earlier
  3. They incur a certain amount of additional computational overhead to control and manage the execution of simulation process

2.8.2 Object-Based and Object-Oriented Simulation(1)

- An *object* consists of a collection of fields (state variables or attribute) and a set of *methods*, typically implemented as procedures, that model the behavior of component
- Logically, invoking a method can be viewed as sending a message to the object requesting that the method be executed. On parallel or distributed computer, message are used to invoke methods for objects that reside on another computer
- System that require the simulation to be structured as collections of interacting objects are often referred to as *object-based* system. Object-oriented system go a step further by providing a capability called *inheritance* to characterize relationships among collections of similar, but not identical objects

2.8.2 Object-Based and Object-Oriented Simulation(2)

- The new object type is said inherit the properties (fields and methods) of original, called *derived type*, may replace properties with new one, or extend the object type to include entirely new field and methods
- The derived type may replace or *overload* methods in the base object type. Both the derived type and the base type use the same name for the method. The ability to have different methods with the same name is called *polymorphism*
- Methods can be used to implement event procedures. Encapsulation of the state of an object supports parallel and distributed simulations because it discourages the use of global variables that may be difficult to implement on parallel and distributed computers which do not provide shared memory

2.8.3 Query Events and Push versus Pull Processing(1)

- When simulation collections of interacting objects, it is common for object to need to collect state information from other objects
- This might be implemented by invoking an “Ask” method at each of objects requesting the value of variable. In a distributed simulation, message must be sent to retrieve the requested information
- “pull process”
  - Using *query events* to request a state value with time stamp equal to the current simulation time of monitor object
  - The drawback is that two message transmissions are required to collect information from another processor. Further the process requesting the information must usually block until the query has been satisfied

2.8.3 Query Events and Push versus Pull Processing(2)

- “push processing”
  - LP that holds the two variable “pushes” changes to variable to other process
  - This reduces the two messages required for each transmission of state information in the pull approach to only one, and it eliminates forcing the monitor process to block while waiting for the response of its queries
  - The source of the data cannot know if the user of this information requires each new value of the state variable

2.8.4 Event Retraction

- Another commonly used mechanism is to *retract* previously scheduled events
- Simulation program may schedule events that it *believes* will occur when the event is scheduled, but later it will retract the scheduled event should this belief turn out to be incorrect
- Without the ability to retract previously scheduled events, the simulator would need to devise a way to ignore the now invalid departure events when they are processed.

2.9 Other Approaches to Exploiting Concurrent Execution

- To use dedicated functional units to implement specific sequential simulation function
- To execute independent, sequential simulation program on different processors