Modeling and Simulation

PDES Introduction

The Null Message Synchronization Algorithm

Outline

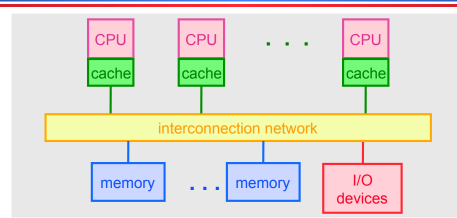
- Parallel / Distributed Computers
- Air Traffic Network Example
- Parallel Discrete Event Simulation
  - » Logical processes & time stamped messages
  - » Local causality constraint and the synchronization problem
- Chandy/Misra/Bryant - Null Message Algorithm
  - » Ground rules
  - » An algorithm that doesn't work
  - » Deadlock avoidance using null messages

Parallel & Distributed Computers

- Parallel computers (tightly coupled processors)
  - » Shared memory multiprocessors
  - » Distributed memory multicomputers
- Distributed computers (loosely coupled processors)
  - » Networked workstations

	Parallel Computers	Distributed Computers
Physical extent	Machine room	Building, city, global
Processors	Homogeneous	Often heterogeneous
Comm. Network	Custom switch	Commercial LAN / WAN
Comm. Latency (small messages)	A few to tens of microseconds	hundreds of microseconds to seconds

Shared Memory Multiprocessors



Examples:
Sun Enterprises
SGI Origin

programming model: shared variables; synchronization via `locks`

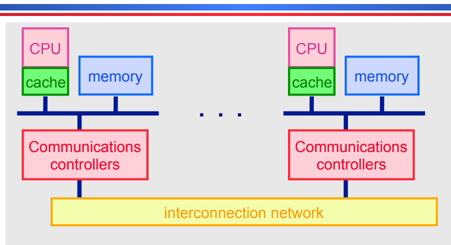
Processor 1

```
{
shared int i; L
…
Lock( L )
i = i + 1;
Unlock( L )
…
}
```

Processor 2

```
{
shared int i; L
…
Lock( L )
i = i + 1;
Unlock( L )
…
}
```


Distributed Memory Multiprocessors



Examples:
IBM SP
Intel Paragon

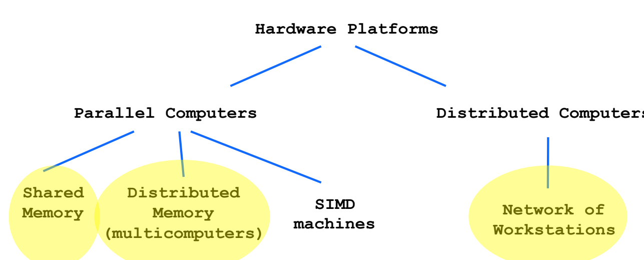
programming model: no shared variables; message passing

Processor 1

```
{
int i; …
Send( 2, &i, sizeof(int))
…
}
```

Processor 2

```
{
int j; …
Receive( &j, sizeof(int))
…
}
```

Event-Oriented World View

Event handler procedures

state variables

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

```
Arrival     Landed     Departure
 Event       Event       Event
{           {           {
  …           …           …
}           }           }
```

Simulation application

Simulation executive

Now = 8:45

Pending Event List (PEL): 9:00, 9:16, 10:10

Event processing loop

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

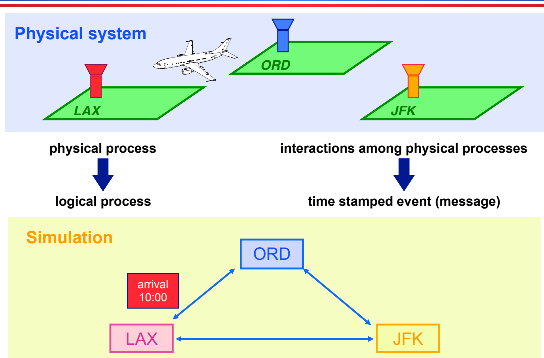

Parallel Discrete Event Simulation

- **Extends** example to model a network of airports
  - » **Encapsulate** each airport simulator in a **logical process**
  - » Logical processes can schedule events (**send messages**) for other logical processes

More generally...

- **Physical system**
  - » Collection of interacting physical processes (airports)
- **Simulation**
  - » Collection of logical processes (LPs)
  - » Each LP models a physical process
  - » **Interactions** between physical processes modeled by scheduling events between LPs

Parallel Discrete Event Simulation: Example



all interactions between LPs must be via messages (no shared state)

LP Simulation Example

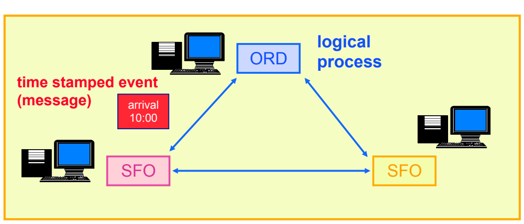
`Now`: current simulation time
`InTheAir`: number of aircraft landing or waiting to land
`OnTheGround`: number of landed aircraft
`RunwayFree`: Boolean, true if runway available

```
Arrival Event:
    InTheAir := InTheAir+1;
    if( RunwayFree )
      RunwayFree:=FALSE;
      Schedule Landed event(local) @ Now + R;

Landed Event:
    InTheAir := InTheAir-1;
    OnTheGround := OnTheGround + 1;
    Schedule Departure event(local) @ Now + G;
    if( InTheAir > 0 ) Schedule Landed event(local) @ Now + R;
    else RunwayFree := True;
Departure Event: (D = Delay to reach another airport)
    OnTheGround := OnTheGround - 1;
    Schedule Arrival Event (remote) @ (Now+D) @ another airport
```


Parallel Discrete Event Simulation: Example

- LP paradigm appears well suited to concurrent execution
- Map LPs to different processors
  - » Multiple LPs per processor OK
- Communication via message passing
  - » All interactions via messages
  - » No shared state variables



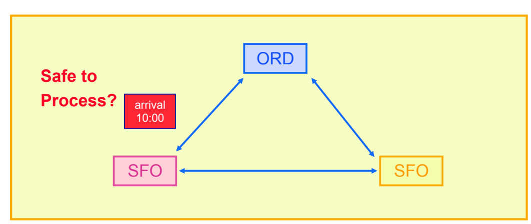
The "Rub"

Golden rule

Golden rule for each process:

"Thou shalt process incoming messages in time stamp order"

local causality constraint

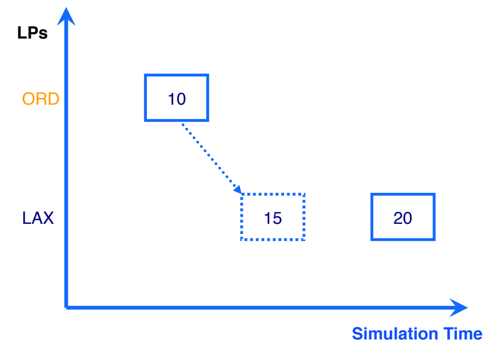


The Synchronization Problem

Synchronization Problem: An algorithm is needed to ensure each LP processes events in time stamp order

Observation: Ignoring events with the same time stamp (for now), adherence to the local causality constraint is sufficient to ensure that the parallel simulation will produce exactly the same results as a sequential execution where all events across all LPs are processed in time stamp order.

The Synchronization Problem



Synchronization Algorithms

- **Conservative synchronization: Avoid violating the local causality constraint (wait until it's safe to process an event)**
  - » deadlock avoidance using null messages (Chandy/Misra/Bryant)
  - » deadlock detection and recovery
  - » synchronous algorithms (e.g., execute in "rounds")
- **Optimistic synchronization: Allow violations of local causality to occur, but detect them at runtime and recover using a rollback mechanism**
  - » Time Warp (Jefferson)
  - » numerous other approaches

Outline

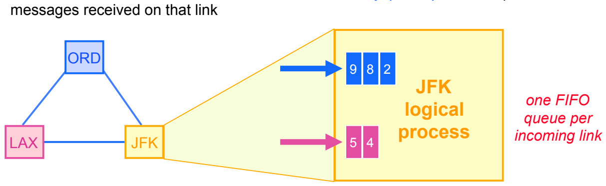
- **Parallel / Distributed Computers**
- **Air Traffic Network Example**
- **Parallel Discrete Event Simulation**
  - » Logical processes
  - » Local causality constraint
- **Chandy/Misra/Bryant Null Message Algorithm**
  - » Ground rules
  - » An algorithm that doesn't work
  - » Deadlock avoidance using null messages

Conservative Algorithms

Assumptions:

- **logical processes (LPs) exchanging time stamped events (messages)**
- **static network topology, no dynamic creation of (and connection of LPs)**
- **messages sent on each link are sent in time stamp order**
- **network provides reliable delivery, preserves order (received in same order that they are sent)**

Observation: The above assumptions imply the time stamp of the last message received on a link is a **lower bound on the time stamp (LBTS)** of subsequent messages received on that link

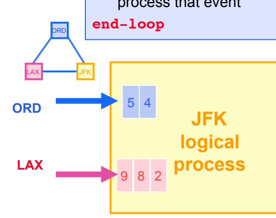


Goal: Ensure LP processes events in time stamp order

A Simple Conservative Algorithm

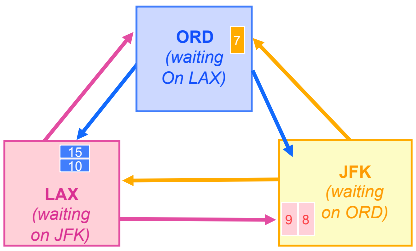
Algorithm A (executed by each LP):
Goal: Ensure events are processed in time stamp order:

```
while( simulation is not over )
    wait until each FIFO contains at least one message
    remove smallest time stamped event from its FIFO
    process that event
end-loop
```



- process time stamp 2 event
- process time stamp 4 event
- process time stamp 5 event
- wait (block) until a message is received from ORD.

Deadlock Example



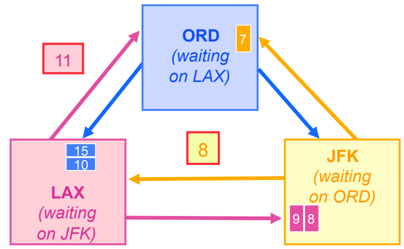
A cycle of LPs forms where each is waiting on the next LP in the cycle.

No LP can advance; the simulation is deadlocked.

***Observation**: Algorithm A is prone to deadlock! (cycle of empty queues…)*

Deadlock Avoidance Using Null Messages

Break deadlock: each LP send "null" messages indicating a lower bound on the time stamp of future messages.



Assume minimum delay (flight time) between airports is 3 units of time

- Recall that JFK is initially at time 5.
- JFK sends null message to LAX (who is waiting for JFK) with time stamp **8 = (5 +3)**
- LAX sends null message to ORD with time stamp **11 = (8+3)**
- ORD may now process message with time stamp **7**

Deadlock Avoidance Using *Null* Messages

Null Message Algorithm (executed by each LP):

Goal: Ensure events are processed in time stamp order and avoid deadlock

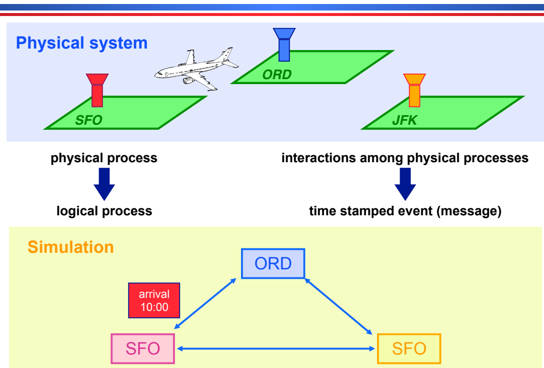
```
while( simulation is not over )
    wait until each FIFO contains at least one message
    remove smallest time stamped event from its FIFO
    process that event
    send null messages to neighboring LPs with time stamp indicating a
          lower bound on future messages sent to that LP (current time plus
          lookahead )
end-loop
```

The null message algorithm relies on a "lookahead" (flight time in the example) ability.

Summary

- **Parallel Discrete Event Simulation**
  - » **Collection of sequential simulators (LPs) possibly running on different processors**
  - » **Logical processes communicating exclusively by exchanging messages**
- **Chandy/Misra/Bryant Null Message Algorithm**
  - » **Null messages: Lower bound on the time stamp of future messages the LP will send**
  - » **Null messages avoid deadlock (non-zero lookahead)**

Parallel Discrete Event Simulation: Example



all interactions between LPs must be via messages (no shared state)