

# Other existing mechanisms to handle concurrency

- Path Expressions
  - abstraction designed to describe the list of all possible legal executions of operations on shared resource
- Communicating Sequential Processes (CSP)
  - the exchange of messages as synchronization points between sequential processes
- ADA tasks
  - language constructs for the message passing
- One thing in common
  - None in practical use currently (though ADA was a popular language in 80s and 90s)

# Multi-threaded programming in Java

- Java allows program to specify multiple threads of execution
- Provides instructions to ensure mutual exclusion, and selective blocking/unblocking of threads

# What is a thread in Java ?

- A thread is a program-counter and a stack
- All threads share the same memory space
- A running thread can
  - Yield
  - Sleep
  - Wait for I/O or notification
  - Be pre-empted
- A key feature: Synchronized methods
  - Allow an exclusive lock, e.g., in an update method

# Basic Syntax

- Build a thread by extending the class `java.lang.Thread`
- Must have a `public void run()` method – it is executed at the start of the thread, and when it finishes, the thread finishes
- Synchronized statement
  - `Synchronized (obj) { block }`
  - Obtains a lock on obj before executing block, releases lock after executing block
- `Wait()` gives up lock and suspends the thread
- `Notifyall()` resumes all threads waiting on object, resumed tasks must reacquire lock before continuing

# Producer Consumer Example

```
Public class ProdCons {  
    private boolean ready ;  
    private Object obj ;  
    public ProducerConsumer() {  
        ready = false ;  
    }  
    public ProducerConsumer (Object o) {  
        obj = o ;  
        ready = true ;  
    }  
}
```

```
Synchronized Object consume() {  
    while (!ready) wait() ;  
    ready = false ;  
    notifyAll() ;  
    return obj ;  
}  
Synchronized void produce (object o)  
{  
    while (ready) wait() ;  
    obj = o ;  
    ready = true ;  
    notifyAll() ;  
}  
}
```

# Introduction to Distributed Systems

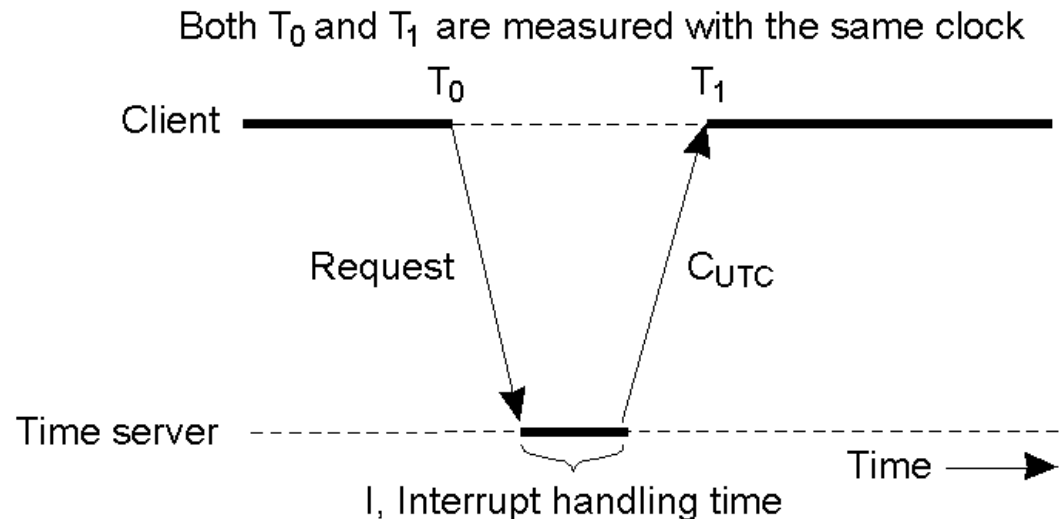
- First two topics
  - Conceptual Issue: Local and Global Clocks
  - Practical Issue: Process Communication
- Distributed Mutual Exclusion

# Absence of Global Clock

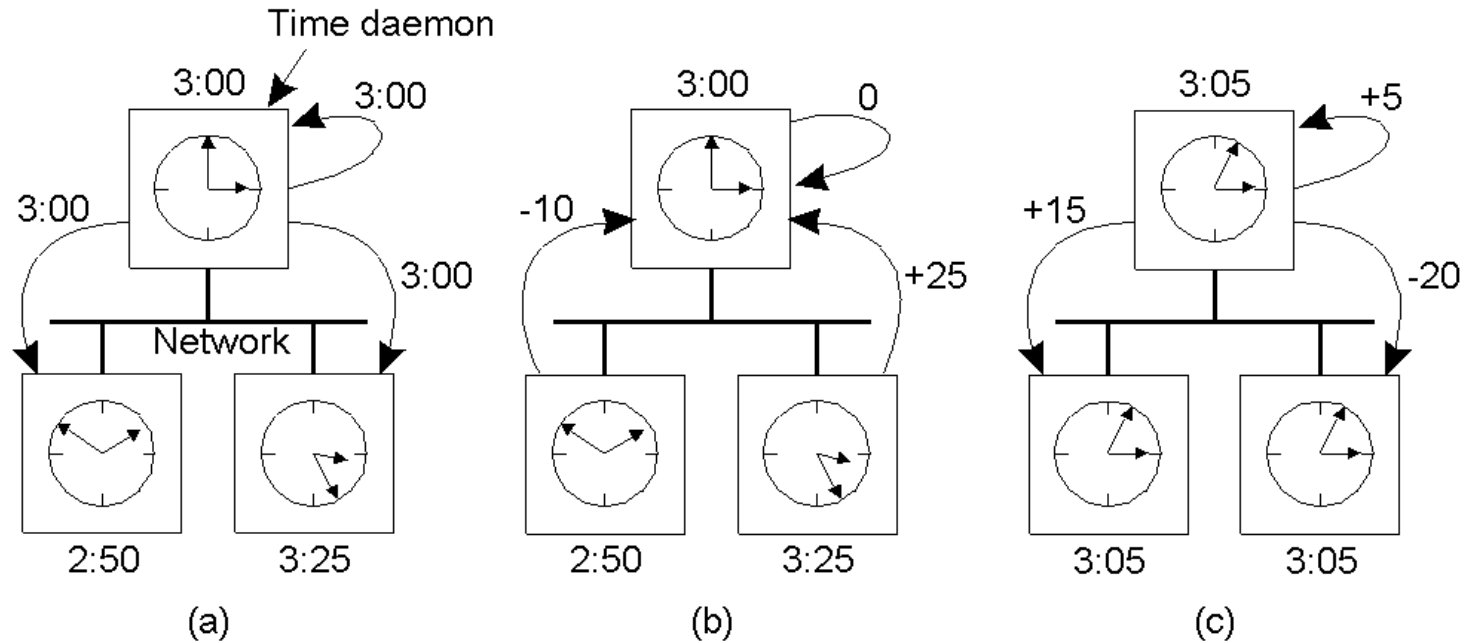
- Problem: synchronizing the activities of different part of the system (e.g. process scheduling)
- What about using a single shared clock?
  - two different processes can see the clock at different times due to unpredictable transmission delays
- What about using radio synchronized clocks?
  - Propagation delays are unpredictable
- Software approaches
  - Clock synchronization algorithms
  - Logical clocks

# Cristian's Algorithm

- Basic idea: get the current time from a time server.
- Issues:
  - Error due to communication delay - can be estimated as  $(T_1 - T_2 - I)/2$
  - Time correction on client must be gradual



# The Berkeley Algorithm



- The time daemon asks all the other machines for their clock values
- The machines answer
- The time daemon tells everyone how to adjust their clock

# Logical clocks

- The need to order events in a distributed system has motivated schemes for “logical clocks”
- These artificial clocks provide some but not all of the functionality of a real global clock
- They build a clock abstraction based on underlying physical events of the system

# “Happened before” relation: definitions

- “Happened before” relation ( $\rightarrow$ ):
  - $a \rightarrow b$  if  $a$  and  $b$  are in the same process and  $a$  occurred before  $b$
  - $a \rightarrow b$  if  $a$  is the event of sending a message and  $b$  is the event of receiving the same message by another process
  - if  $a \rightarrow b$  and  $b \rightarrow c$  then  $a \rightarrow c$ , i.e. the relation “ $\rightarrow$ ” is transitive
- The *happened before* relation is a way of ordering events based on the behavior of the underlying computation

# “Happened before” relation: definitions (2)

- Two distinct events  $a$  and  $b$  are said to be *concurrent* ( $a \parallel b$ ) if and  
and  $a \not\rightarrow b$      $b \not\rightarrow a$
- For any two events in the system, either  $a \rightarrow b$ ,  $b \rightarrow a$  or  $a \parallel b$
- Example:

$$e_{11} \parallel e_{21}$$

$$e_{22} \rightarrow e_{13}, e_{13} \rightarrow e_{14}$$

thus  $e_{22} \rightarrow e_{14}$

