CprE 450/550x
Distributed Systems and Middleware

Synchronization

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu
April 6 & 8, 2004

Readings for Today's Lecture

- References
  - Chapter 5 of "Distributed Systems: Principles and Paradigms"
  - Chapter 14 of Coulouris: "Distributed Systems"

3

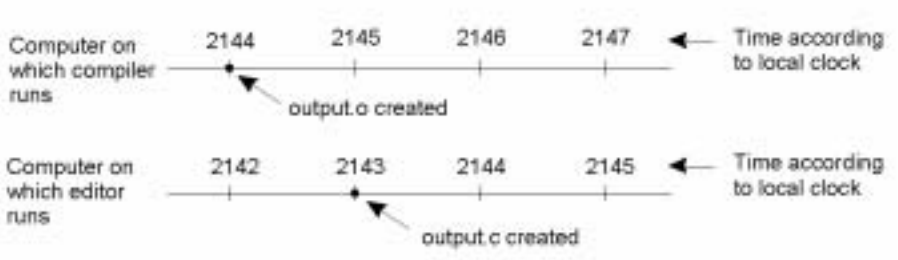
Clock Synchronization

- In a centralized system, time is unambiguous
- In a distributed system, achieving agreement on time is not trivial.
- Example: UNIX makefile

  A change to one source file only requires one file to be recompiled, not all the files
- How make works?
  - Examine the times at which all the source and object files were last modified.
  - In a distributed system in which there is no global agreement on time, how?
- **Is it possible to synchronize all the clocks in a distributed system?**

4

Clock Synchronization



When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

5

Physical Clocks

- Almost all computers have a circuit for keeping track of time.
- Computer Timer is a machined quartz crystal
  - When kept under tension, quartz crystal oscillates at a well-defined frequency, depending on the kind of crystal, how it is cut, and the amount of tension.
  - Two registers: a counter and a holding register
  - Each oscillation decrements the counter by one, when it gets to 0, an interrupt is generated and the counter is reset from the holding register.
  - Each interrupt is called a clock tick.
  - When the system is booted initially, date and time are required to be entered and deposited in CMOS RAM.
  - Each clock tick increases the time stored in CMOS RAM by one such that software clock can be maintained.

6

Physical Clocks

- It doesn't matter if the clock is off by a small of amount for a single computer with a single clock.
- For multiple CPUs with their own clocks, things change:
  - Though the frequency at which a crystal oscillator runs is fairly stable, it is impossible to guarantee the crystals on different computers run at the same frequency.
  - The crystals will run at slightly different rates, which result in the clocks out-of-sync. The time value difference is called clock skew.
  - Programs depending on the time associate with files, objects, messages may fail due to these clock skew.
- **How do we synchronize the clocks with real-world clocks?**

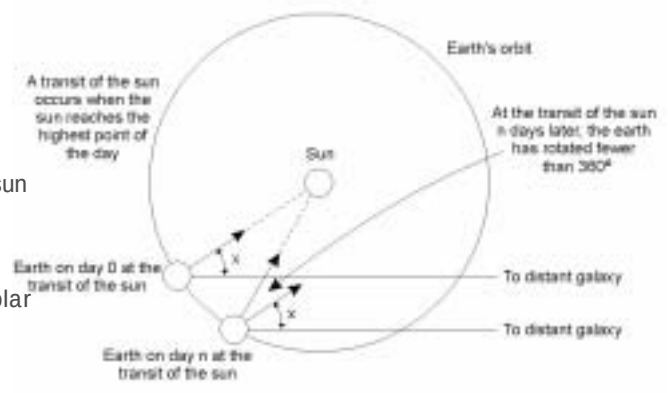
7

Physical Clocks

How time is actually measured?

Time has been measured astronomically.

- Transit of the sun
- Solar day (24h)
- Solar second (1/86400 of a solar day)



Computation of the mean solar day.

8

Physical Clocks

- With the invention of atomic clock in 1948, measuring time becomes more accurately by counting transitions of the cesium 133 atom.
- Physicists took over the job of timekeeping from astronomers
- A second is defined as the time it takes the cesium 133 atom to make exactly 9, 192,631,770 transitions. This number makes an atomic second equal to the mean solar second.
- BIH averages the number of clock ticks from 50 laboratories in the world to produce International Atomic Time (TAI).
- TAI=the mean number of ticks of the cesium 133 clocks since midnight on Jan. 1, 1958 divided by 9, 192,631,770 .

9

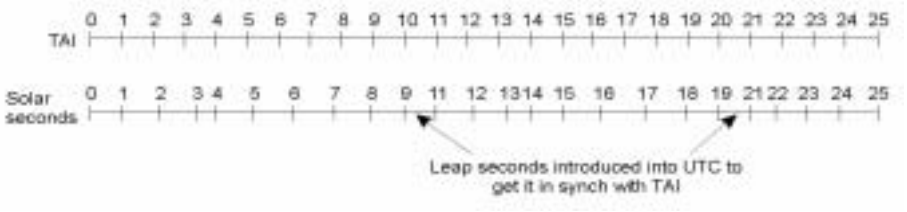
Physical Clocks

- 86,400 TAI seconds is 3 msec less than a mean solar day.
- Over the years, noon would become earlier and earlier.
- BIH introduce leap seconds whenever the difference between TAI and solar time grows to 800 msec.
- Universal Coordinated Time (UTC) (replaced Greenwich Mean Time, which is astronomical time)

- NIST operates a shortwave radio station with call letters WWV from Fort Colins, CO.
- WWV broadcasts a short pulse at the start of each UTC second. +-1msec (+-10msce due to atmosphere fluctuations).
- Similar services, UK's MSF, GEOS (earth satellite), etc.

10

Physical Clocks



TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.

11

Clock Synchronization Algorithms

- Each machine is assumed to have a timer that causes an interrupt *H* time a second. When the timer goes off, the interrupt handler adds one to a software clock.

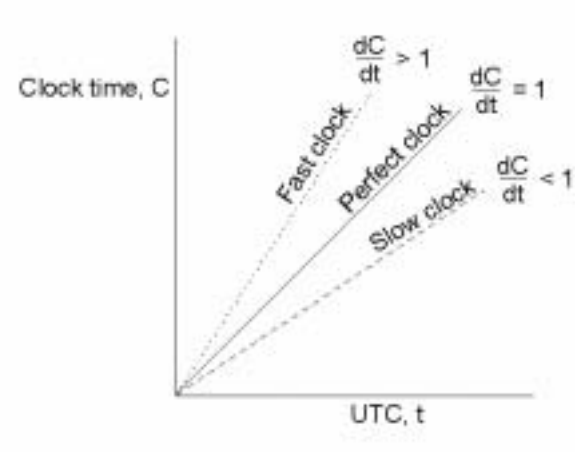
  C: value of the clock

  $C_p(t)$: The value of the clock at machine p at UTC time t.

- Ideally, $C_p(t)=t$ for all p and t. i.e., $dC/dt=1$
- In practice, the relative error obtainable with modern timer chips is 10^{-5}.
- Maximum drift rate r, where $1-r <= dC/dt <= 1+r$.

12

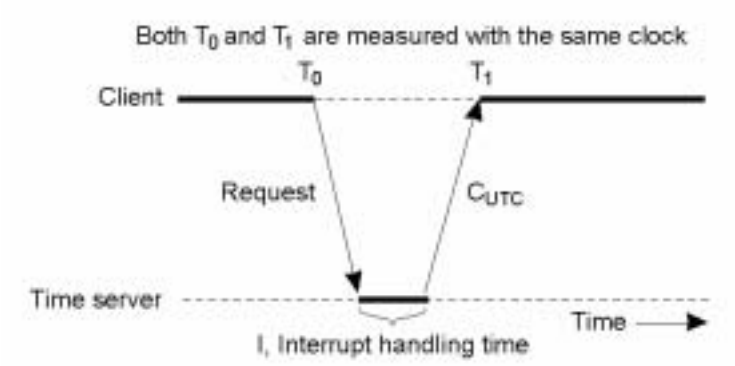
Clock Synchronization Algorithms



The relation between clock time and UTC when clocks tick at different rates.

13

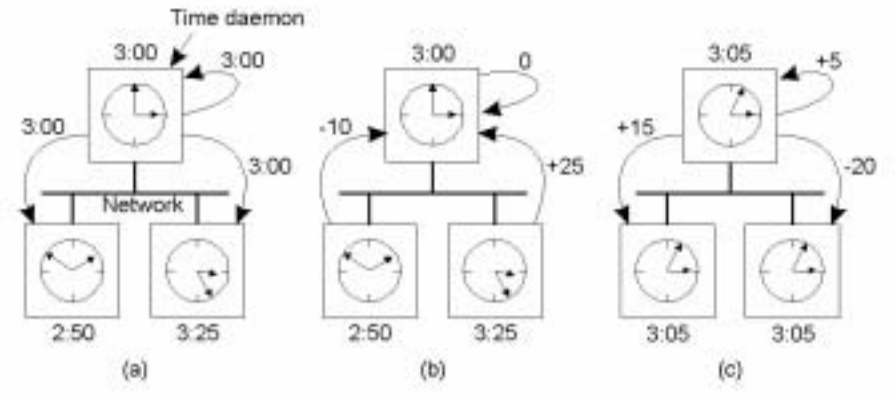
Cristian's Algorithm



Getting the current time from a time server.

14

The Berkeley Algorithm



a) The time daemon asks all the other machines for their clock values
b) The machines answer
c) The time daemon tells everyone how to adjust their clock

15

Averaging algorithm

- Dividing time into fixed-length re-sync intervals.
- At the beginning, each machine broadcasts its own time.
- After a machine broadcasts its time, it starts a local timer to collect all other broadcasts that arrive during some time interval S.
- Then,
  - Average the values from all the other machines
  - Discard the m highest and m lowest values, and average the remaining ones.
  - NTP (Network Time Protocol)
  - Can be further improved

16

Multiple External Time Sources

- WWV, GEOS receivers

17

Use of Synchronized Clocks

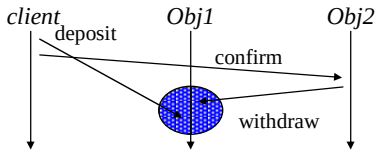
- At-most-once message dilivery
  - Message seq number, what about system crashes and reboots?
  - ConnID+timestamp

18

19

Synchronization: Introduction

- A scary scenario:



- **Synchronization**: temporal ordering of sets of events produced by concurrent processes in time.
  - Synchronization between senders and receivers of messages.
  - Control of joint activity.
  - Serialization of concurrent access to shared objects/resources.
- Why not Semaphores ?!
  - centralized systems: shared memory, central clock
  - distributed system: message passing, no global clock
- Events cannot be totally ordered!

20

A Partial Event Ordering for Distributed Systems (Lamport 1978)

- Absence of central time means: no notion of *happened-when* (no total ordering of events)
- But can generate a *happened-before* notion (partial ordering of events)
- *Happened-Before* relation:

1. P_i — *a* — *b*

Event *a* happened-before Event *b*. (*a -> b)*

2. P_i — *a*; P_j — *b*

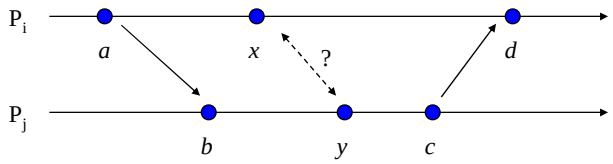
Event *a* happened-before Event *b*. (*a -> b)*

3. P_i — *a*; P_j — *b* — *c*

Event *a* happened-before Event *c*. (*a -> c) (transitivity)*

happened-before Relation

- What when no *happened-before* relation exists between two events?



Events *x* and *y* are concurrent.

- Problem:
  - only approximate knowledge of state of other processes
- Need global time:
  - common clock
  - synchronized clocks

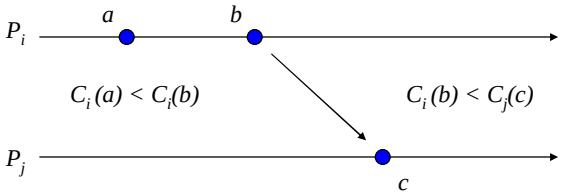
Logical Clocks

- Absolute time?
- Is chronological ordering necessary?
- Logical clock: assigns a number to each local event.

Clock Condition

$$\forall \text{ Events } a,b \;:\; \text{if } a \rightarrow b, \text{ then } C(a) < C(b)$$

- In Other Words:



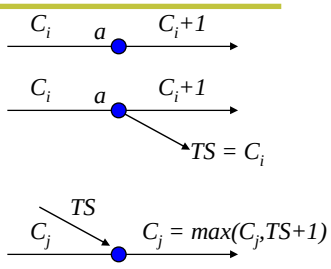
Total Ordering with Logical Clocks

- Rules:

  Rule 1: increment C_i after every local event.

  Rule 2: timestamp outgoing messages with current local clock

  Rule 3: Upon receiving message with timestamp TS, P_j updates local clock C_j to be
  $C_j = max\ (C_j, TS+1)$

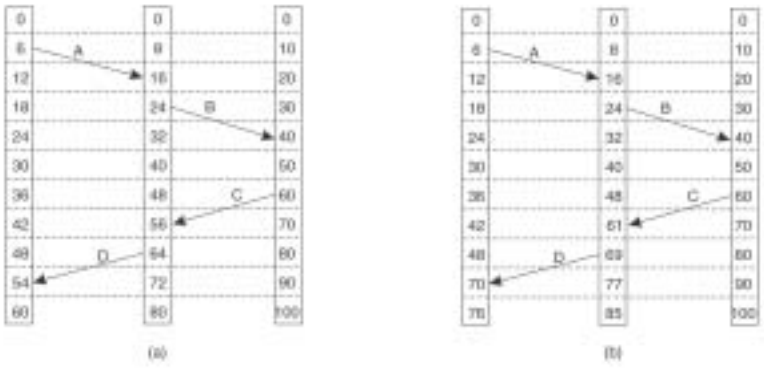


- Total ordering of events: assuming that clocks satisfy Clock Condition, define following relation:

$$a \Rightarrow b \quad \Leftrightarrow \quad C_i(a) < C_j(b) \text{ or } C_i(a) = C_j(b) \text{ and } i < j$$

for events a on P_i and b on P_j.

Lamport Timestamps



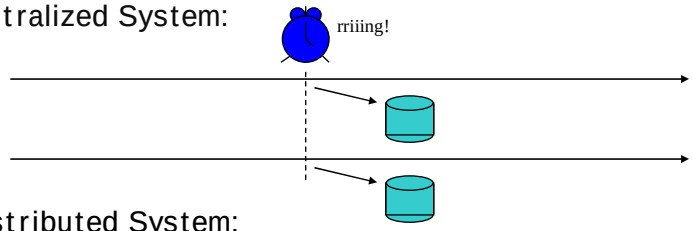
a) Three processes, each with its own clock. The clocks run at different rates.

b) Lamport's algorithm corrects the clocks.

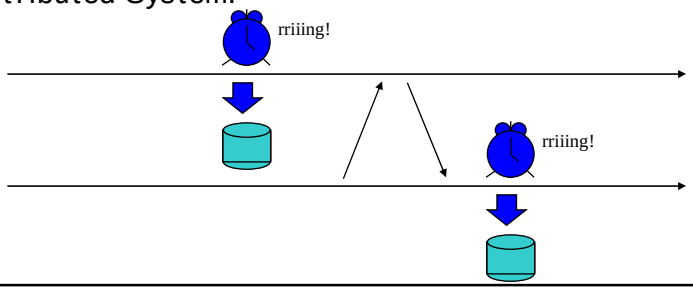
25

Example: Distributed Checkpointing

- "At 5pm everybody writes its state to stable storage!"
- Centralized System:



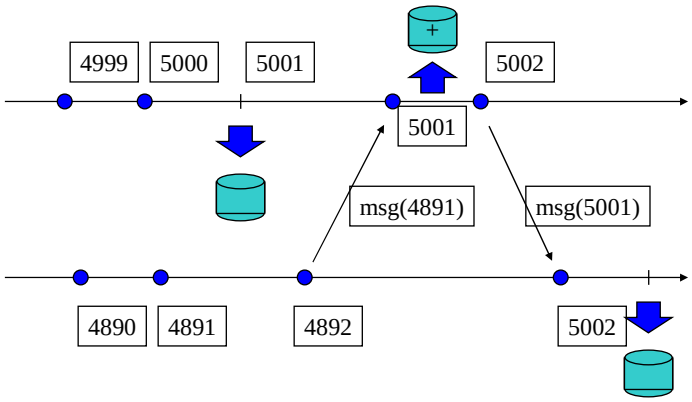
- Distributed System:



26

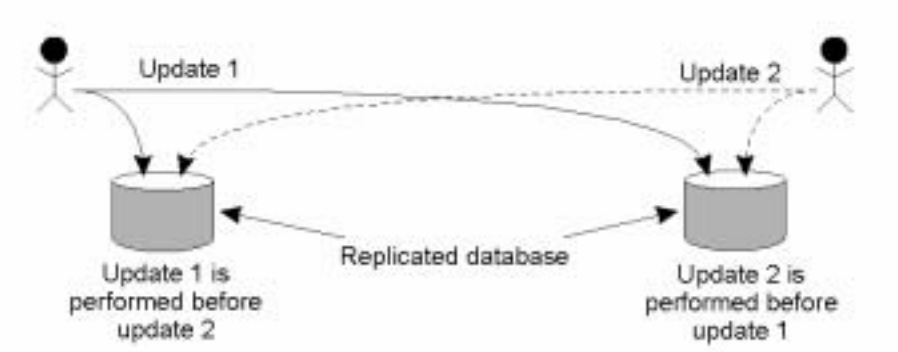
Distributed Checkpointing and Logical Clocks

"At logical-clock time 5000 write state to stable storage!"



27

Another Example: Totally-Ordered Multicasting



Updating a replicated database and leaving it in an inconsistent state.

28

Vector Timestamps

- Lamport timestamps: Can we say sth if C(a)<C(b)?
- Example: BBS message A and B, if totally ordered multicast is used, no way to say whether A is a reaction to B, or A and B are completely independent.

  The problem of Lamport timestamps does not capture causality.
- **Causality can be captured by Vector Timestamps**
  - VT(a): A vector timestamp assigned to event a.
  - If VT(a)<VT(b), then event a is known to causally precede event b.
  - Vector timestamp are contructed by letting each process Pi maintain a vector Vi with the following properties:
    1. $V_i[i]$ is the number of events happened so far at Pi
    2. If $V_i[j]=k$, then Pi knows that k events have occurred at Pj.

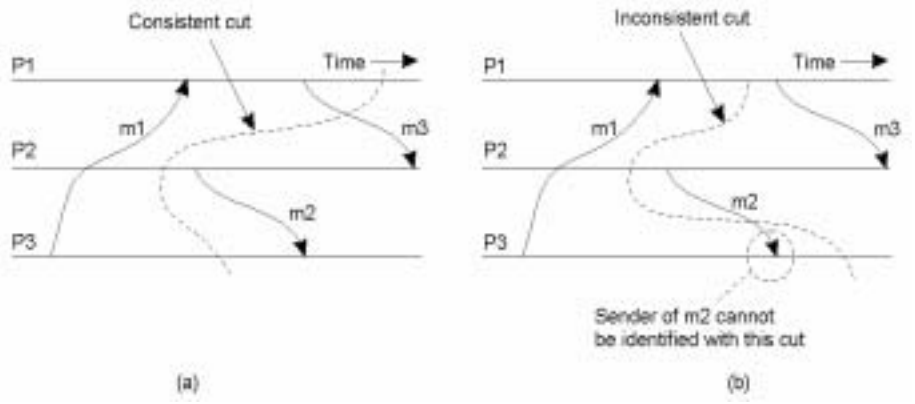
29

30

Global State (1)

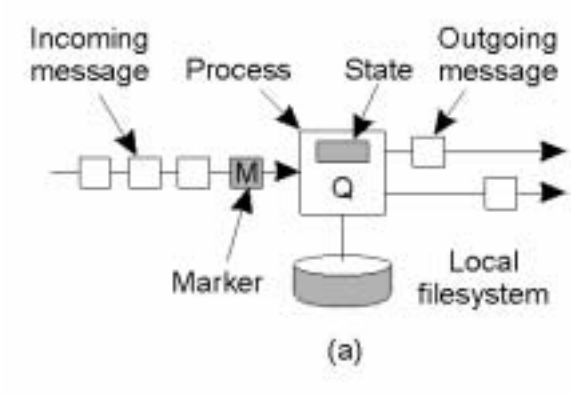
- Knowing the global state in distributed systems is useful on many occasions.
- The global state consists of the local state of each process, together with the messages-in-transit.
- Distributed Snapshot (Chandy and Lamport'85)

Global State (2)



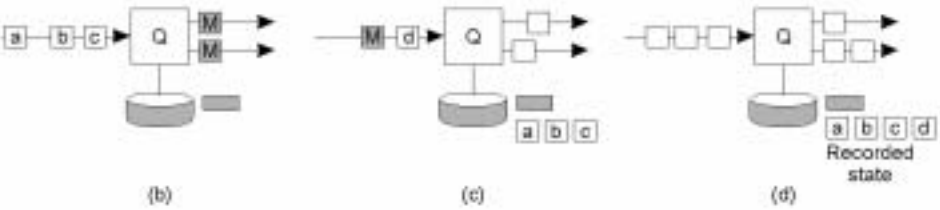
a) A consistent cut
b) An inconsistent cut

Global State (3)



a) Organization of a process and channels for a distributed snapshot

Global State (4)



b) Process Q receives a marker for the first time and records its local state

c) Q records all incoming message

d) *Q* receives a marker for its incoming channel and finishes recording the state of the incoming channel

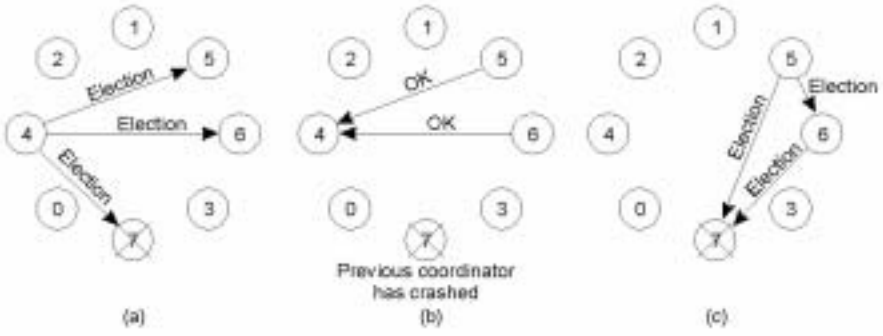
35

Election Algorithms

- Many distributed algorithms requires one process in the system acts as a leader (coordinator, initiator).
- It does not matter which process it is, but one of them has to do it.
- The goal of election algorithm is to ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be.

36

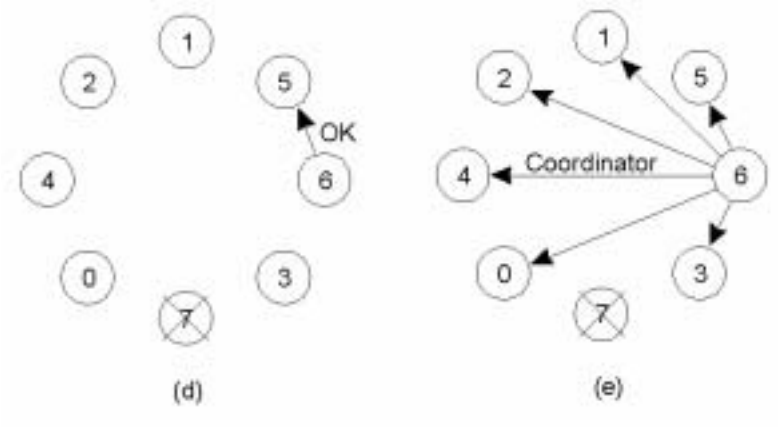
The Bully Algorithm (1)



The bully election algorithm

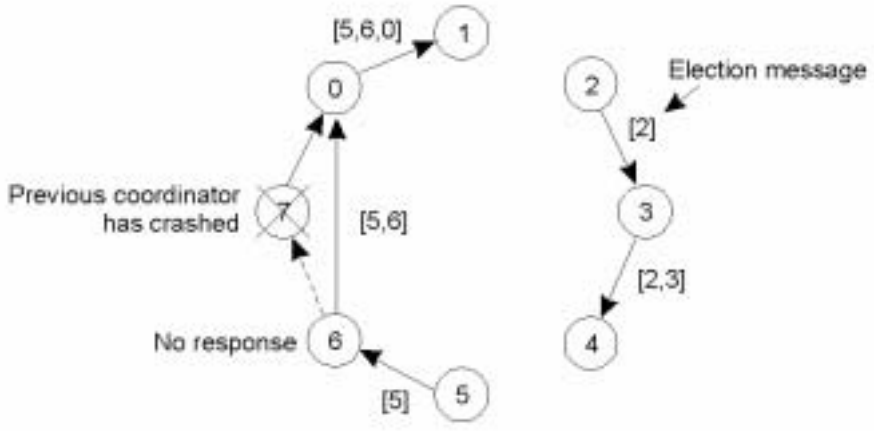
a) Process 4 holds an election
b) Process 5 and 6 respond, telling 4 to stop
c) Now 5 and 6 each hold an election

The Bully Algorithm (2)



d) Process 6 tells 5 to stop
e) Process 6 wins and tells everyone

A Ring Algorithm

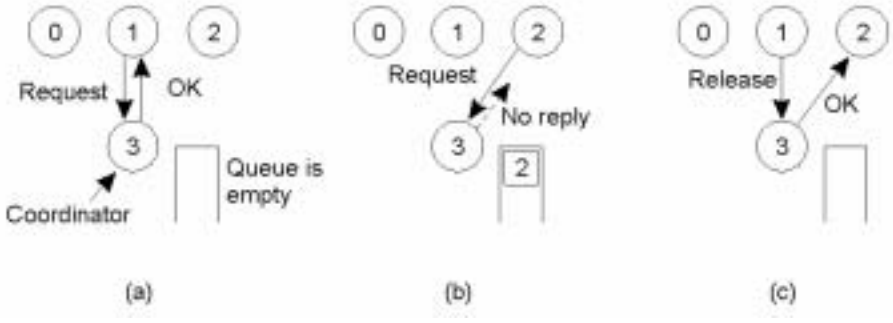


Election algorithm using a ring.

39

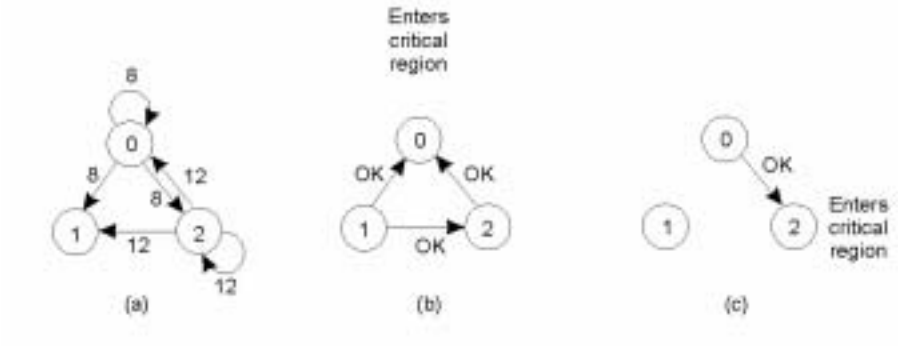
40

Mutual Exclusion: A Centralized Algorithm



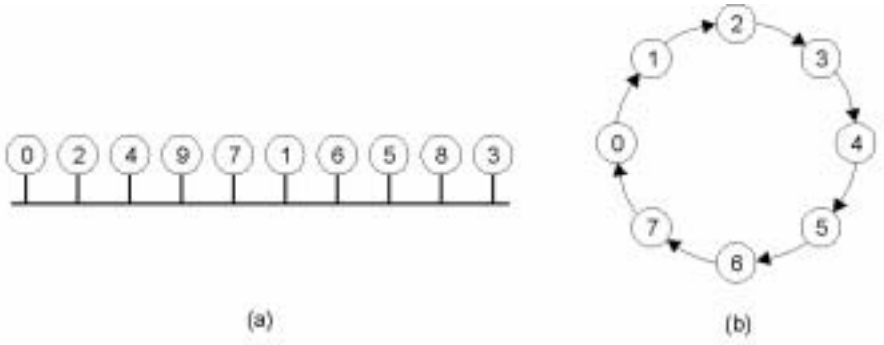
a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted
b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
c) When process 1 exits the critical region, it tells the coordinator, when then replies to 2

A Distributed Algorithm



a) Two processes want to enter the same critical region at the same moment.
b) Process 0 has the lowest timestamp, so it wins.
c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

A Toke Ring Algorithm



a) An unordered group of processes on a network.
b) A logical ring constructed in software.

43

Comparison

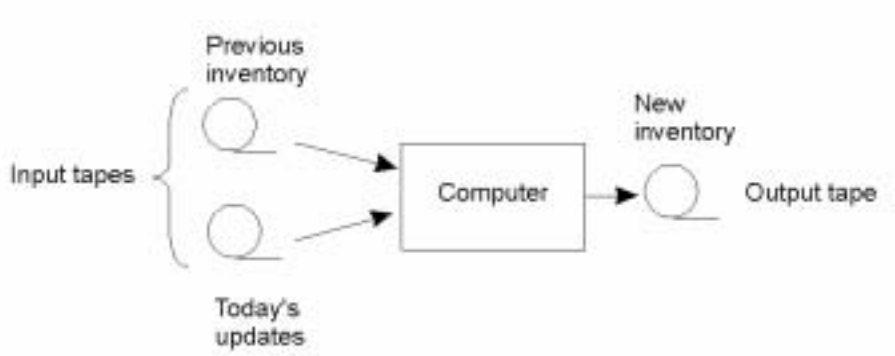
Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

A comparison of three mutual exclusion algorithms.

44

45

The Transaction Model (1)



Updating a master tape is fault tolerant.

46

The Transaction Model (2)

Primitive	Description
BEGIN_TRANSACTION	Make the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Examples of primitives for transactions.

47

The Transaction Model (3)

```
BEGIN_TRANSACTION
  reserve WP -> JFK;
  reserve JFK -> Nairobi;
  reserve Nairobi -> Malindi;
END_TRANSACTION
```

(a)

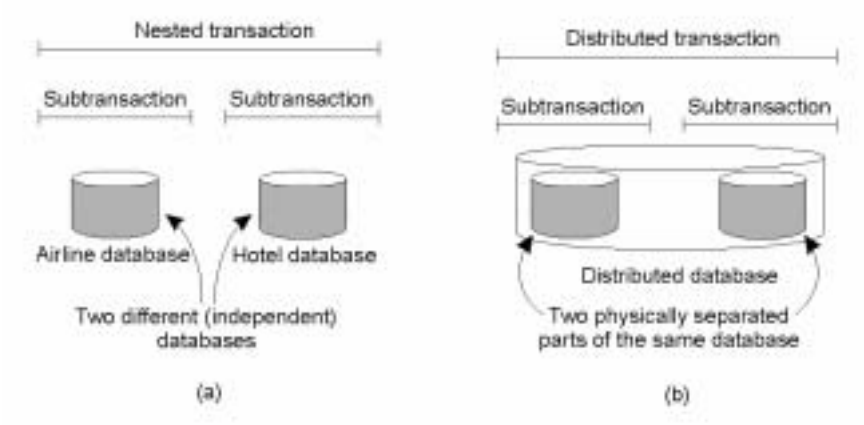
```
BEGIN_TRANSACTION
  reserve WP -> JFK;
  reserve JFK -> Nairobi;
  reserve Nairobi -> Malindi full =>
ABORT_TRANSACTION
```

(b)

a) Transaction to reserve three flights commits
b) Transaction aborts when third flight is unavailable

48

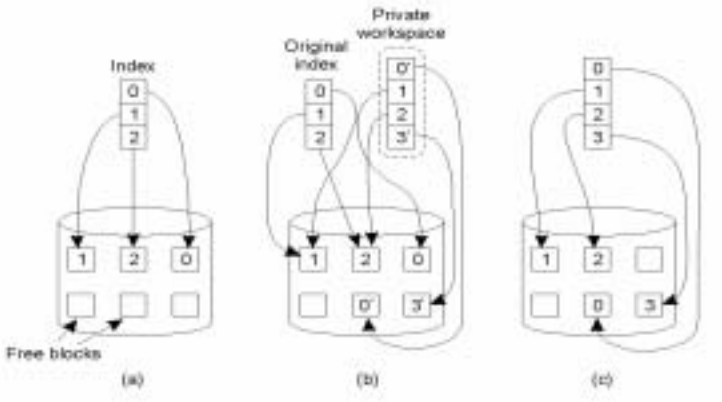
Distributed Transactions



a) A nested transaction
b) A distributed transaction

49

Private Workspace



a) The file index and disk blocks for a three-block file
b) The situation after a transaction has modified block 0 and appended block 3
c) After committing

50

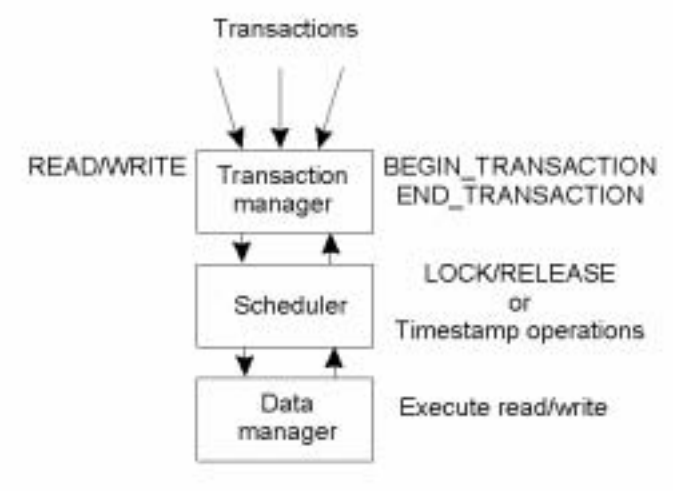
Writeahead Log

x = 0;	Log	Log	Log
y = 0;			
BEGIN_TRANSACTION;			
x = x + 1;	[x = 0 / 1]	[x = 0 / 1]	[x = 0 / 1]
y = y + 2		[y = 0/2]	[y = 0/2]
x = y * y;			[x = 1/4]
END_TRANSACTION;			
(a)	(b)	(c)	(d)

a) A transaction
b) – d) The log before each statement is executed

51

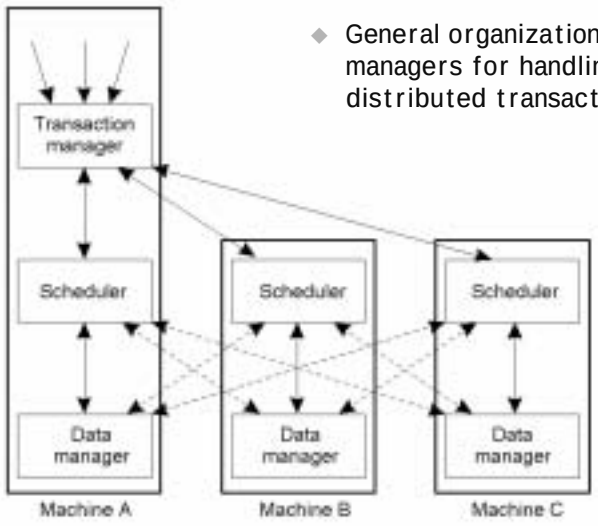
Concurrency Control (1)



- General organization of managers for handling transactions.

52

Concurrency Control (2)



- General organization of managers for handling distributed transactions.

53

54

Atomic Transactions

- Example: online bank transaction:
  ```
  withdraw(amount, account1)
  deposit(amount, account2)
  ```
- What if network fails before deposit?
- Solution: Group operations in an atomic transaction.

- Volatile storage *vs.* stable storage.

- Primitives:
  ```
  BEGIN_TRANSACTION
  END_TRANSACTION
  ABORT_TRANSACTION
  READ
  WRITE
  ```

55

ACID Properties

A atomic: transactions happen indivisibly

C consistent: no violation of system invariants

I isolated: no interference between concurrent transactions

D durable: after transaction commits, changes are permanent

56

Serializability

Schedule is serial if the steps of each transaction occur consecutively.
Schedule is serializable if its effect is "equivalent" to some serial schedule.

```
BEGIN_TRANSACTION
  x = 0;
  x = x + 1;
END_TRANSACTION
```

(a)

```
BEGIN_TRANSACTION
  x = 0;
  x = x + 2;
END_TRANSACTION
```

(b)

```
BEGIN_TRANSACTION
  x = 0;
  x = x + 3;
END_TRANSACTION
```

(c)

Schedule 1	x = 0; x = x + 1; x = 0; x = x + 2; x = 0; x = x + 3	Legal
Schedule 2	x = 0; x = 0; x = x + 1; x = x + 2; x = 0; x = x + 3;	Legal
Schedule 3	x = 0; x = 0; x = x + 1; x = 0; x = x + 2; x = x + 3;	Illegal

(d)

a) – c) Three transactions T_1, T_2, and T_3

d) Possible schedules

57

Testing for Serializability: Serialization Graphs

- Input: Schedule S for set of transactions T_1, T_2, …, T_k.
- Output: Determination whether S is serializable.
- Method:
  - Create *serialization graph G*:
    - » Nodes: correspond to transactions
    - » Arcs: G has an arc from T_i to T_j if there is a T_i:*UNLOCK*(A_m) operation followed by a T_j:*LOCK*(A_m) operation in the schedule.
  - Perform topological sorting of the graph.
    - » If graph has cycles, then S is not serializable.
    - » If graph has no cycles, then topological order is a serial order for transactions.

- Theorem: This algorithm correctly determines if a schedule is serializable.

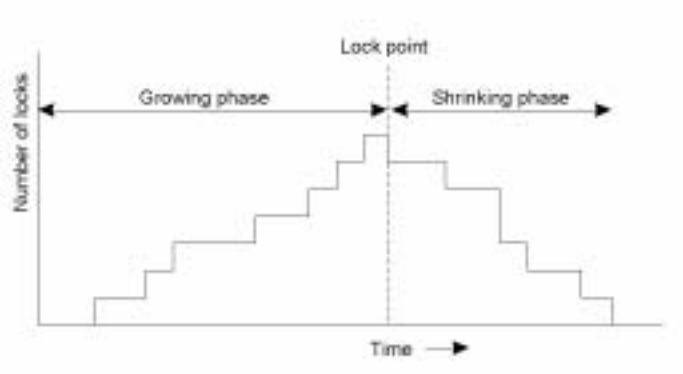
58

Implementation

- How to maintain information from not-yet committed transactions: "Prepare for aborts"
  - **private workspace**
  - **writeahead log / intention lists with rollback**
- Commit protocol
  - **2-phase commit protocol.**
- Concurrency control:
  - **pessimistic -> lock-based: 2-phase locking**
  - **optimistic -> timestamp-based with rollback**

59

Two-Phase Locking

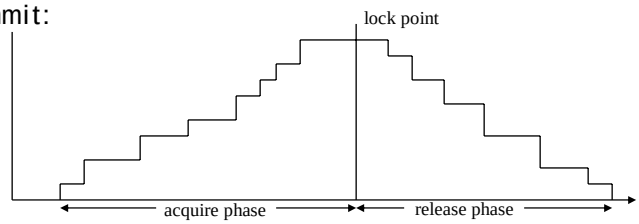


Two-phase locking.

60

Serializability through Two-Phase Locking

- read locks *vs.* write locks
- lock granularity
- arbitrary locking:
  - non-serializable schedules
  - deadlocks!
- Two-Phase Commit:



  - modify data items only after lock point
  - all schedules are serializable

61

Two-Phase Locking (cont)

- Theorem: If S is any schedule of two-phase transactions, then S is serializable.

62

Two-Phase Locking (cont)

- Theorem: If S is any schedule of two-phase transactions, then S is serializable.
- Proof:

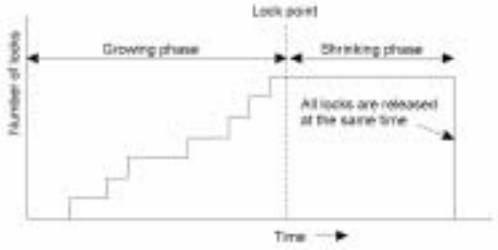
  Suppose not. Then the serialization graph G for S has a cycle,

  $$T_{i1} \rightarrow T_{i2} \rightarrow \ldots \rightarrow T_{ip} \rightarrow T_{i1}$$

  Therefore, a lock by T_{i1} follows an unlock by T_{i1}, contradicting the assumption that T_{i1} is two-phase.

Strict Two-Phase Locking

- Strict two-phase locking:
  - A transaction cannot write into the database until it has reached its commit point.
  - A transaction cannot release any locks until it has finished writing into the database; therefore locks are not released until after the commit point.
- pros:
  - transaction read only values of committed transactions
  - no cascaded aborts
- cons:
  - limited concurrency
  - deadlocks



- Models/protocols can be extended for READ/WRITE locks.

Optimistic Concurrency Control

"Forgiveness is easier to get than permission"

- Basic idea:
  - Process transaction without attention to serializability.
  - Keep track of accessed data items.
  - At commit point, check for conflicts with other transactions.
  - Abort if conflicts occurred.
- Problem:
  - would have to keep track of conflict graph and only allow additional access to take place if it does not cause a cycle in the graph.

Timestamp-based Pessimistic Concurrency Control

- Data items are tagged with read-time and write-time.
- 1. Transaction cannot read value of item if that value has not been written until after the transaction executed.

  Transaction with T.S. t_1 cannot read item with write-time t_2 if $t_2 > t_1$.

  (abort and try with new timestamp)
- 2. Transaction cannot write item if item has value read at later time.

  Transaction with T.S. t_1 cannot write item with read-time t_2 if $t_2 > t_1$.

  (abort and try with new timestamp)
- Other possible conflicts:

  Two transactions can read the same item at different times.

  What about transaction with T.S. t_1 that wants to write to item with write-time t_2 and $t_2 > t_1$?

Timestamp-Based Conc. Control (cont)

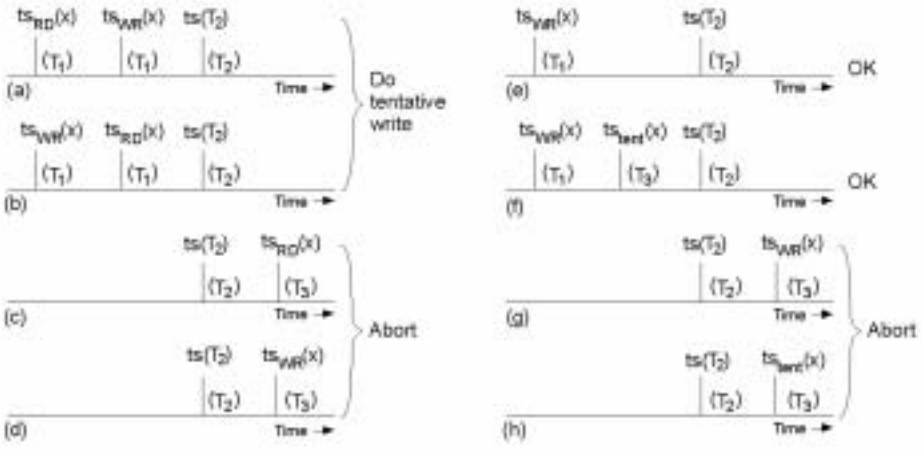
Rules for preserving serial order using timestamps:

a) Perform the operation X if X=READ and $t >= t_w$ or if X=WRITE, $t >= t_r$, and $t >= t_w$.
   X=READ: set read-time to t if $t > t_r$.
   X=WRITE: set write-time to t if $t > t_w$.

b) Do nothing if X=WRITE and $t_r <= t < t_w$.

c) Abort transaction if X=READ and $t < t_w$ or X=WRITE and $t < t_r$.

Pessimistic Timestamp Ordering



Concurrency control using timestamps.

Any Questions?

See you next time.