COP 6611 Advanced Operating System

Synchronization

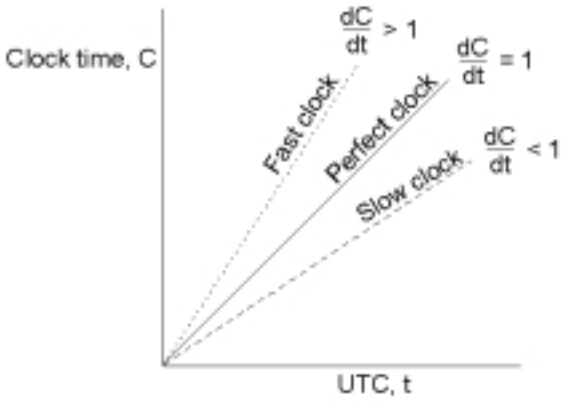
Chi Zhang
czhang@cs.fiu.edu

Outline

- Physical Clock Synchronization
- Logical Clocks
- Global State
- Election Algorithms
- Mutual Exclusion
- Distributed Transactions

2

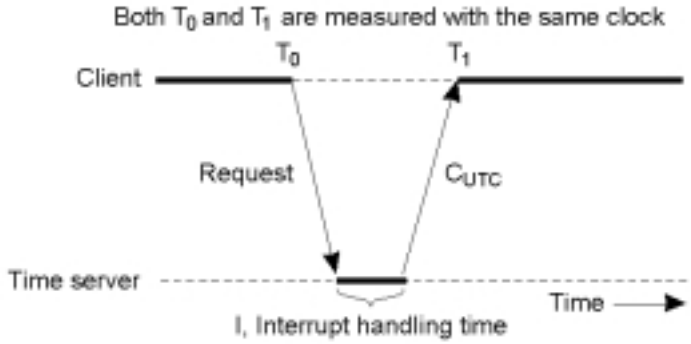
Clock Synchronization Algorithms



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

3

Cristian's Algorithm

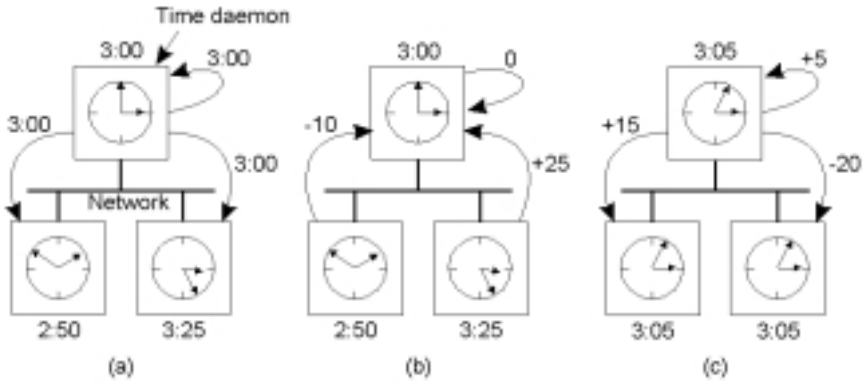


Getting the current time from a time server.

If one machine is synchronized with the standard time

4

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

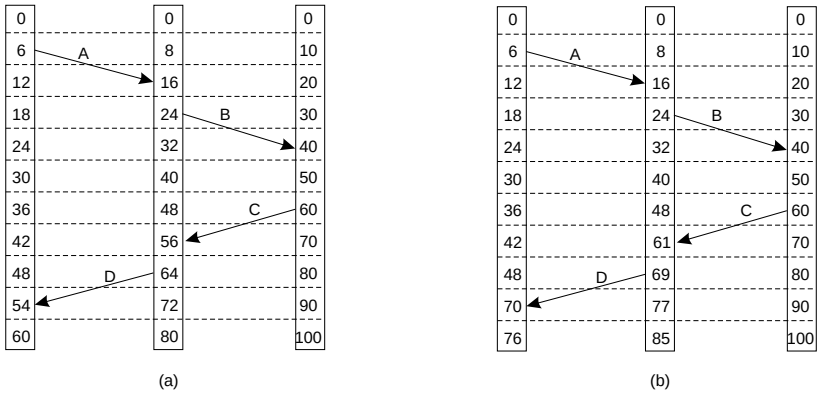
5

At-Most-Once Message Delivary based on Synchronized Clock

- Even in the face of crashes.
- How long to maintain the state?
- Every message carries a connection ID and a time stamp (unique message ID).
  - The same time stamp for retransmitted messages.
- Messages older than G (p.251) is removed.
  - Younger than G? in the table.
- Table entries older than G are removed.
- Every ΔT, CurrentTime is written to disk. After recovery, $G=\max(t_{stored} + \Delta T,\ G)$
  - All states are lost.
  - Before crash, rejects messages with t.s. > G.latest = CurrentTime + ΔT

6

Lamport Timestamps (1)



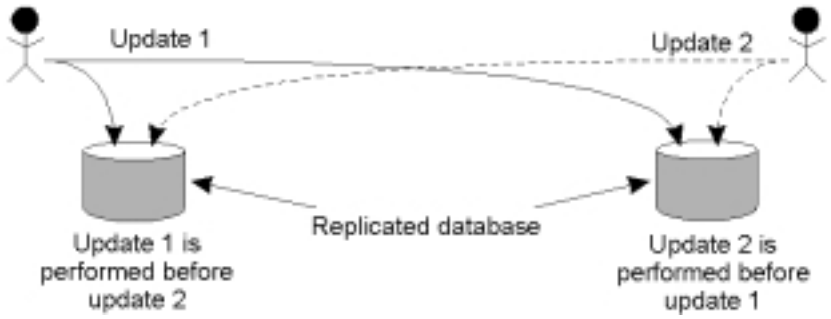
a → b: a happens before b. (transitive relation!), otherwise concurrent.

(i) within a process (ii) between process: Message passing

Lamport's algorithm to assign *C(a)* and *C(b)*

7

Lamport Timestamps (2)

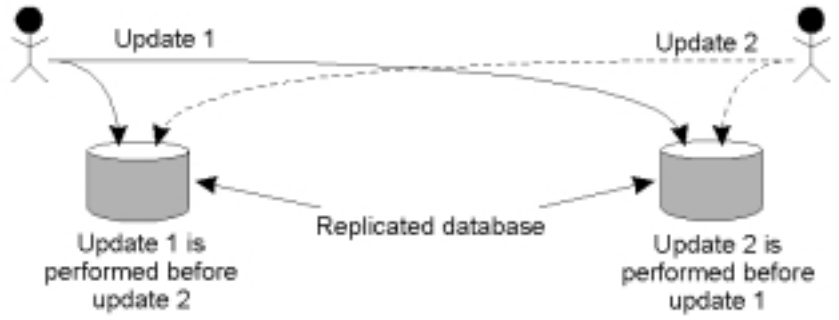


a) Three processes, each with its own clock. The clocks run at different rates.

b) Lamport's algorithm corrects the clocks.

8

Example: Totally-Ordered Multicasting



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

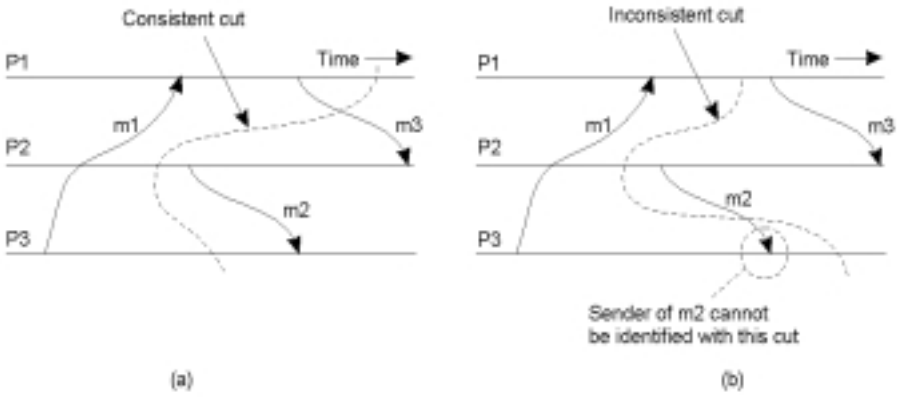
9

Vector Time Stamps

- For each process *i:*
  - $V_i[i]$ is the number messages sent by P_i
  - $V_i[j]=k$: P_i "knows" k messages sent by P_j ($j \neq i$)
- When P_i sends a message *r* to P_j, $V_i[i]$ ++l, and attaches the vector timestamp to *r*
- Message *r* is accepted by P_j *iff*
  - $vt(r)[i] = V_j[i]+1$
  - $vt(r)[m] \leq V_j[m]$ ($m \neq i$)
- Update V_j after *r* is delivered.
  - a happens before b $\Rightarrow$ vt(a) < vt(b)
  - vt(a) < vt(b) $\Rightarrow$ a happens before b

10

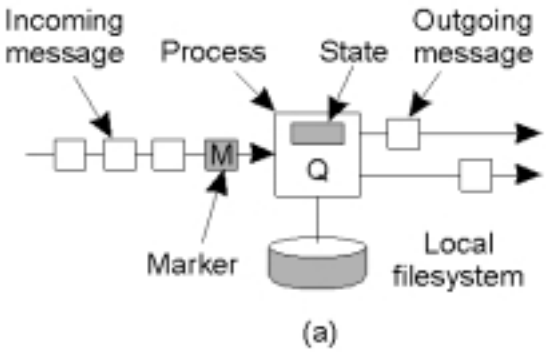
Global State (1)



a) A consistent cut
b) An inconsistent cut

11

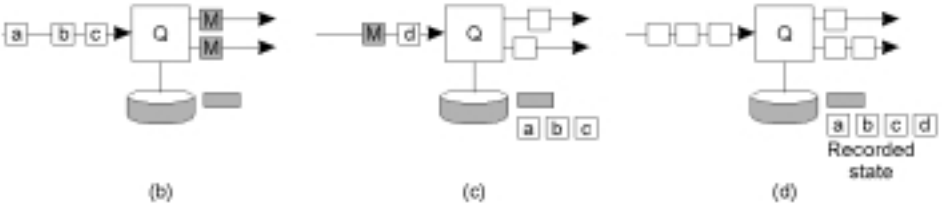
Global State (2)



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

12

Global State (3)



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

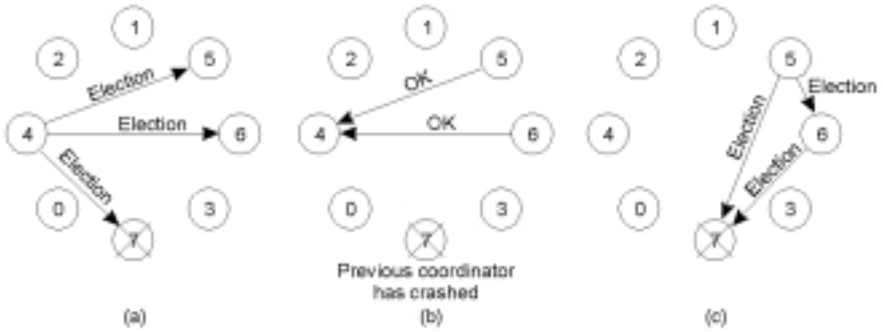
13

Election Algorithms

- Each process has a unique process number.
- The process with the highest number should be elected as the coordinator
- Every process knows the process numbers of all the other processes
- It does not know whether they are currently up or down.

14

The Bully Algorithm (1)

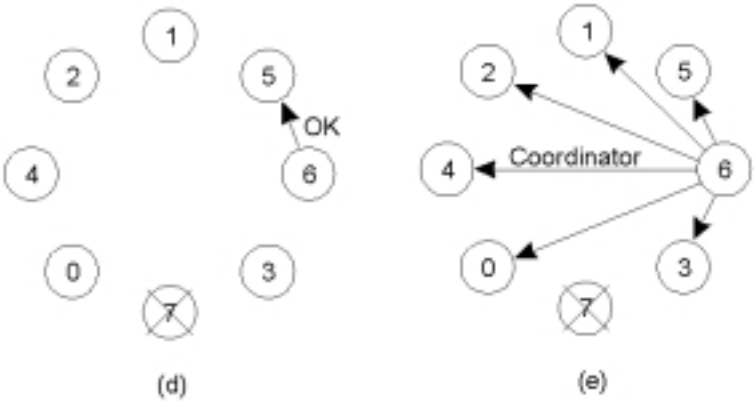


The bully election algorithm

- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election

15

The Bully Algorithm (2)

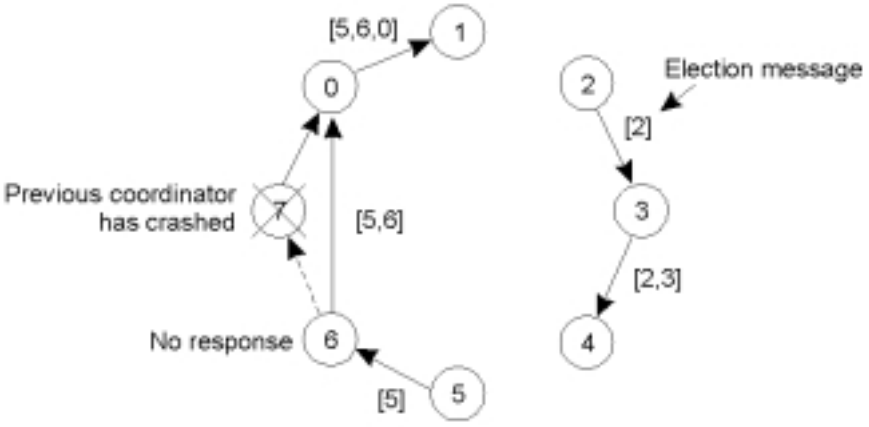


d) Process 6 tells 5 to stop

e) Process 6 wins and tells everyone

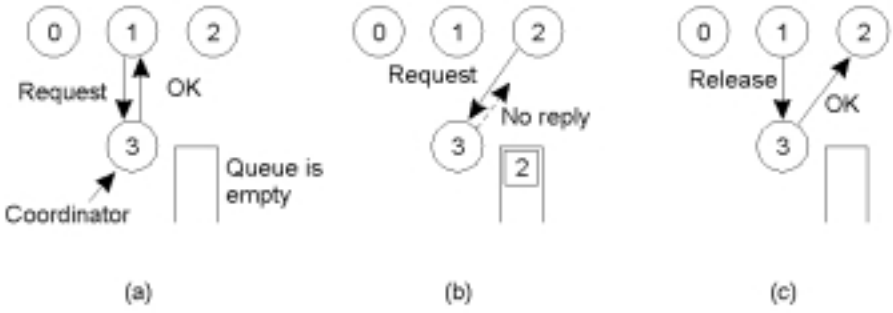
16

A Ring Algorithm



Election algorithm using a ring.

Mutual Exclusion: A Centralized Algorithm

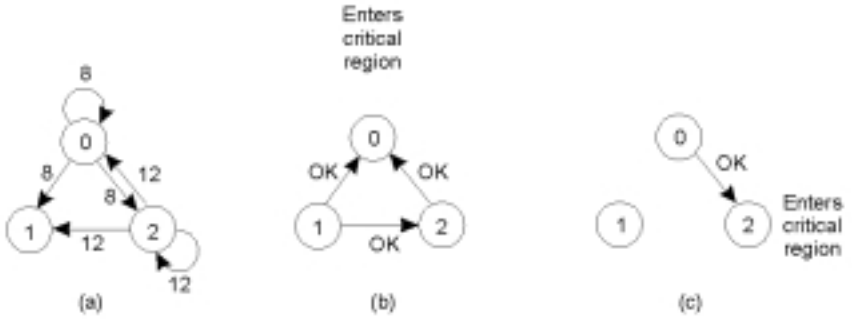


a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted

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, which then replies to 2

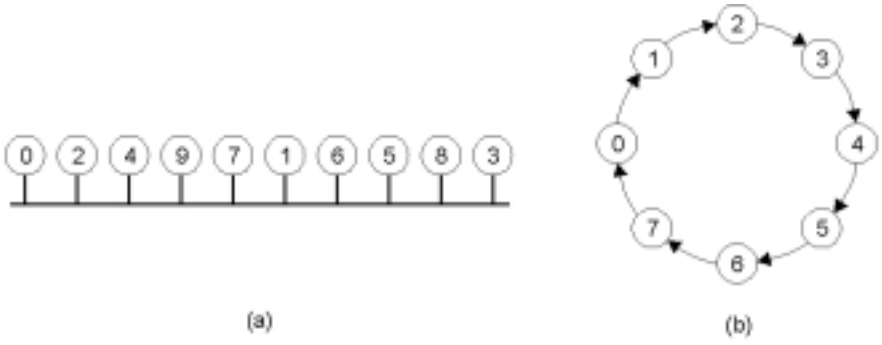
A Distributed Algorithm



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.

19

A Toke Ring Algorithm



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

20

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.

21

The Transaction Model (1)

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.

22

The Transaction Model (2)

```
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

23

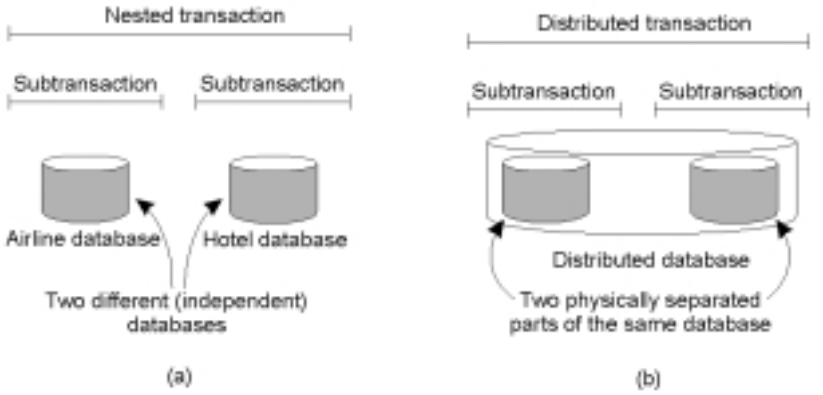
The Transaction Model (3)

ACID

- Atomic
  - Operations in the transaction happens indivisibly
- Consistent
  - E.g. the law of conservation of money
- Isolated (Serializable)
  - Concurrent transactions appear as if one after another.
- Durable
  - Once commits, the data are there forever

24

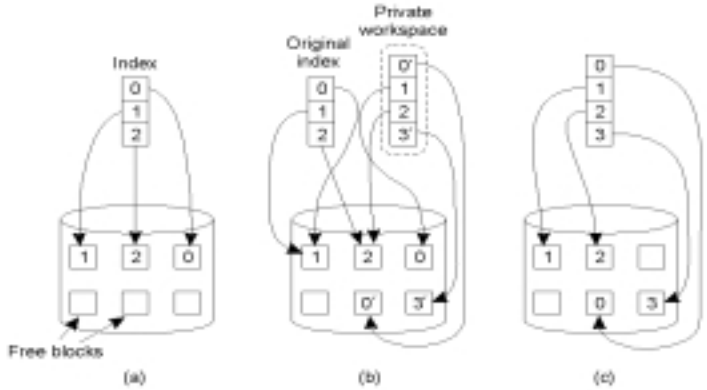
Distributed Transactions



a) A nested transaction
b) A distributed transaction

25

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

26

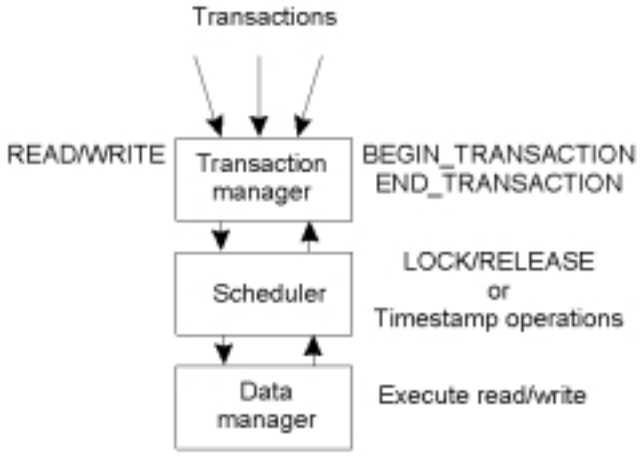
Writeahead Log

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

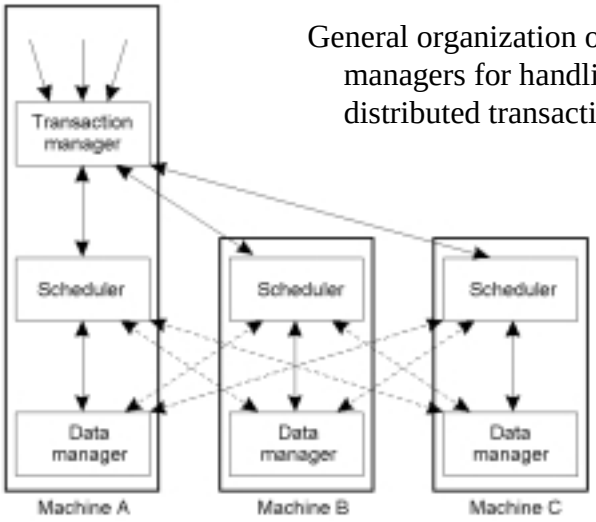
b) – d) The log before each statement is executed

Concurrency Control (1)



General organization of managers for handling transactions.

Concurrency Control (2)



General organization of managers for handling distributed transactions.

29

Serializability

```
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

Read/write conflict; write/write conflict

Pessimistic approaches; Optimistic approaches

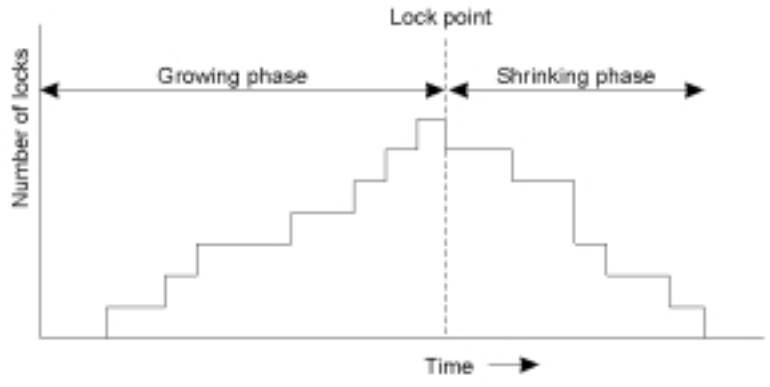
30

Two-Phase Locking (1)

- Request the lock before accessing the data
- Delay the request if already used by another process
- Release the lock if no longer used
- Never grant a lock to a process if it has released another lock.
- Deadlock may occur
  - Request locks in order
  - Detect and kill
  - Timeout and release

31

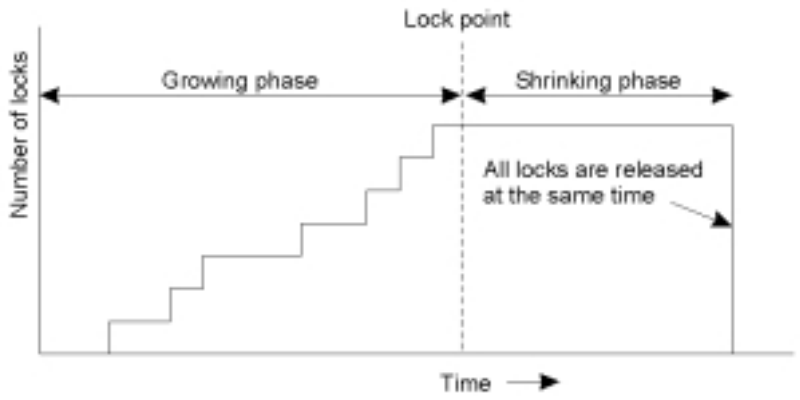
Two-Phase Locking (2)



Two-phase locking.

32

Two-Phase Locking (3)



Strict two-phase locking.

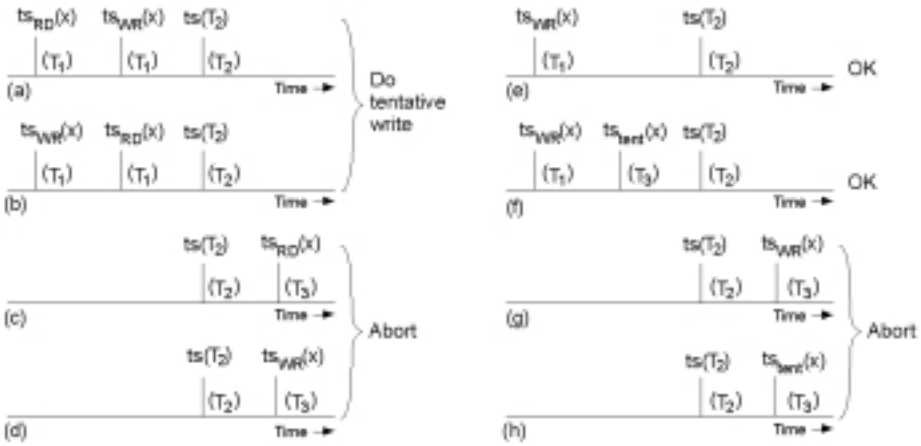
33

Pessimistic Timestamp Ordering (1)

- Assign each transaction T a unique timestamp ts(T). when it starts
  - Serialized as if T commits at ts(T)
- Every data x has a $ts_{RD}(x)$ and a $ts_{WR}(x)$
  - Tentative t.s., becomes permanent after the transaction commits
- For read(T, x) request
  - Abort if $ts(T) < ts_{WR}(x)$
  - If $ts_{WR}(x)$ tentative, wait until it commits
- For write(T, x) request
  - Abort if $ts(T) < ts_{WR}(x)$ or $ts(T) < ts_{RD}(x)$

34

Pessimistic Timestamp Ordering (2)



Concurrency control using timestamps.

Abort rather than wait if requests conflict ⇒Deadlock free!

Optimistic Timestamp Ordering

- Check conflicts at the end of the transaction
  - Check private work space
  - If so, abort
- Allows maximum parallelism if no conflict
- With heavy load and frequent conflicts, a bad choice.