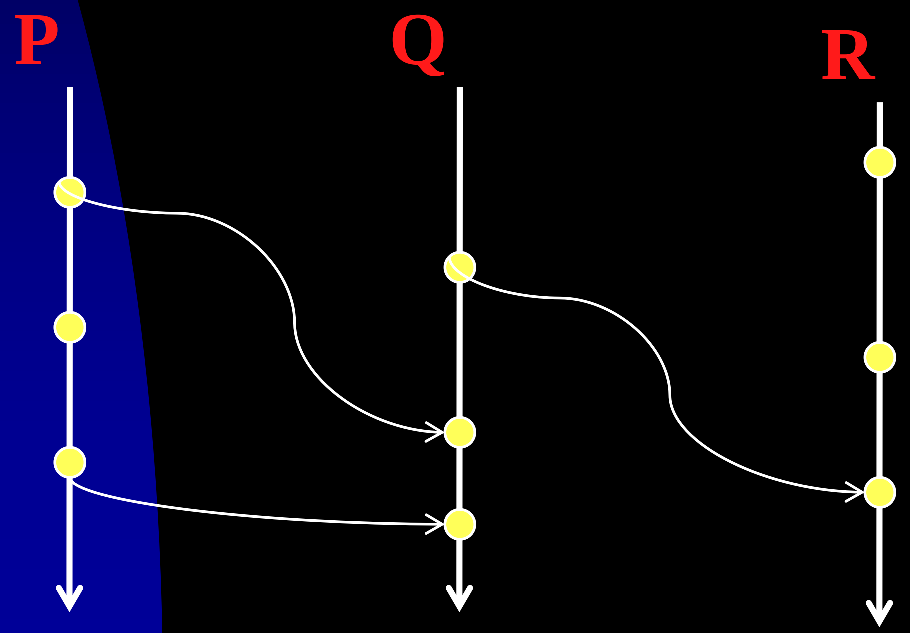
Contents

Distributed Systems

- Definition: A collection of loosely coupled processors interconnected by a communication network.
- Design Issues:
  - A Transparent Interface to Access of Local/Remote Resources
  - Fault Tolerance – Communication, Machines, Services, etc.
  - Scalability

Distributed Systems

- Distributed Synchronization
  - Challenges: No Common Clocks and Memory
  - A Happened-Before Relation – A Partial Order of Events



Distributed Systems

- Why ordering matters?
  - Correctness in concurrent process executions!
- How to achieve the goal?
  - Locking – Two-Phase Locking
  - Timestamp – Synchronization of Logical Counters
    - E.g., A → B → LC(B) = LC(A) + X
- Issues:
  - Atomicity – Two-Phase Commit Protocol
  - Deadlocks

Q & A

Atomic Transactions

- Why Atomic Transactions?
  - Critical sections ensure mutual exclusion in data sharing, but the relationship between critical sections and the logical unit of work might also be important!

  → Atomic Transactions

- Operating systems can be viewed as manipulators of data!

Atomic Transactions – System Model

- Transaction – a logical unit of computation
  - A sequence of read and write operations followed by a commit or an abort.
- Beyond “critical sections”

1. Atomicity: All or Nothing
    - An aborted transaction must be *rolled back*.
    - The effect of a committed transaction must persist and be imposed as a logical unit of operations.

Atomic Transactions – System Model

2. Serializability:

- The order of transaction executions must be equivalent to a serial schedule.

T0	T1
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

Two operations O_i & O_j conflict if

1. Access the same object
2. One of them is write

Atomic Transactions – System Model

- Conflict Serializable:
  - *S* is conflict serializable if *S* can be transformed into a serial schedule by swapping nonconflicting operations.

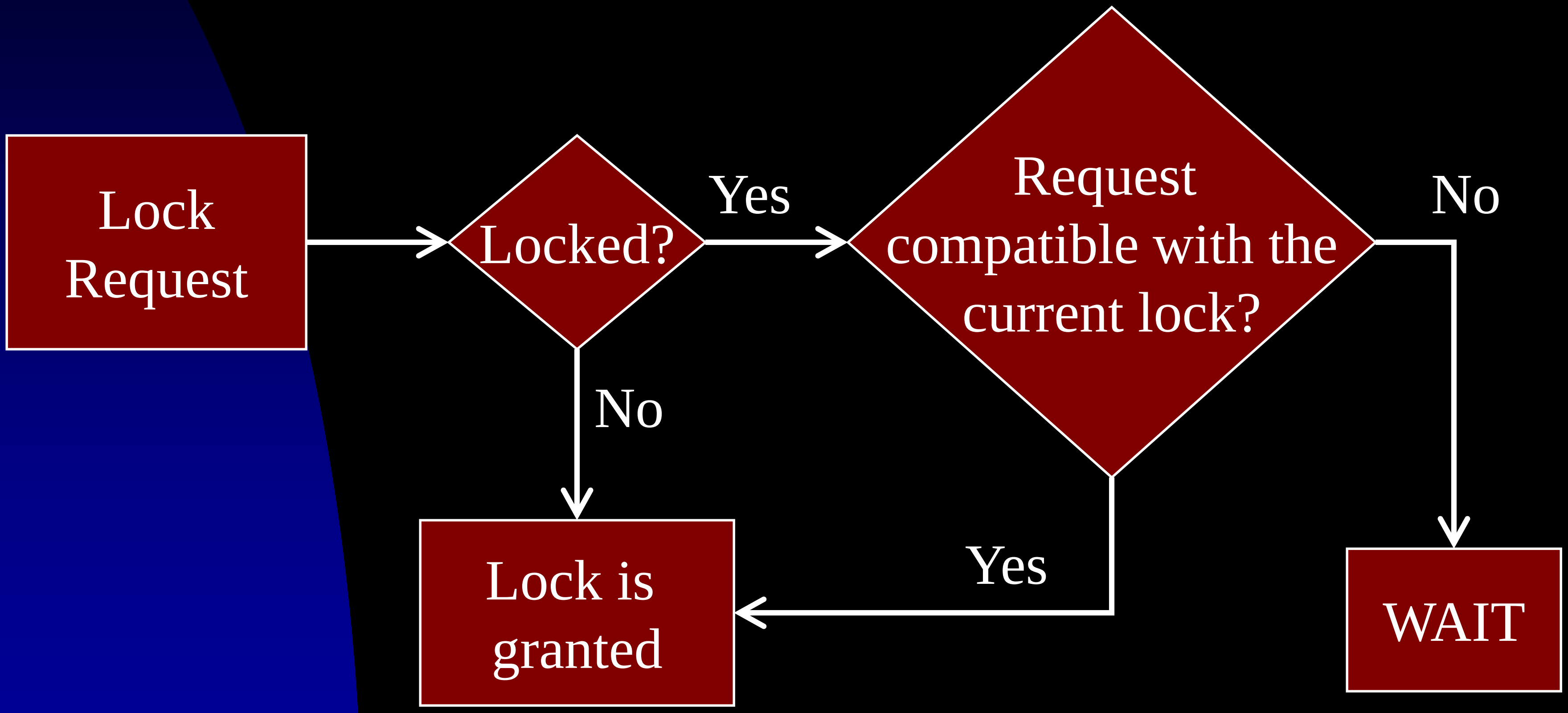
T0	T1
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

T0	T1
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

Atomic Transactions – Concurrency Control

- Locking Protocols
  - Lock modes (A general approach!)
    - 1. Shared-Mode: “Reads”.
    - 2. Exclusive-Mode: “Reads” & “Writes“
  - General Rule
    - A transaction must receive a lock of an appropriate mode of an object before it accesses the object. The lock may not be released until the last access of the object is done.

Atomic Transactions – Concurrency Control



Atomic Transactions – Concurrency Control

- When to release locks w/o violating serializability

  R0(A) W0(A) R1(A) R1(B) R0(B) W0(B)

- Two-Phase Locking Protocol (2PL) – Not Deadlock-Free

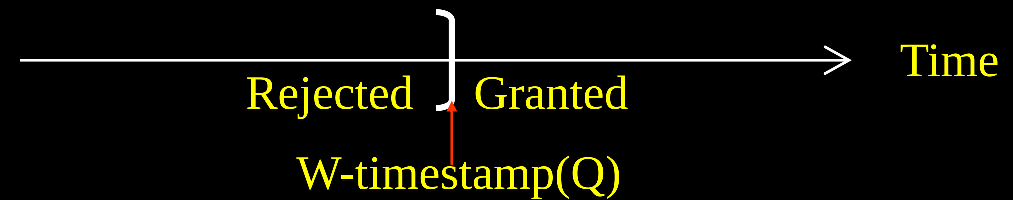
- How to improve 2PL?
  - Semantics, Order of Data, Access Pattern, etc.

Atomic Transactions – Concurrency Control

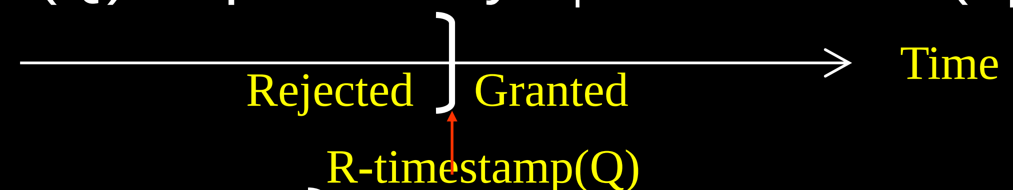
- Timestamp-Based Protocols
  - A time stamp for each transaction $TS(T_i)$
    - Determine transactions' order in a schedule in advance!
  - A General Approach:
    - $TS(T_i)$ – System Clock or Logical Counter
      - Unique?
    - Scheduling Scheme – deadlock-free & serializable

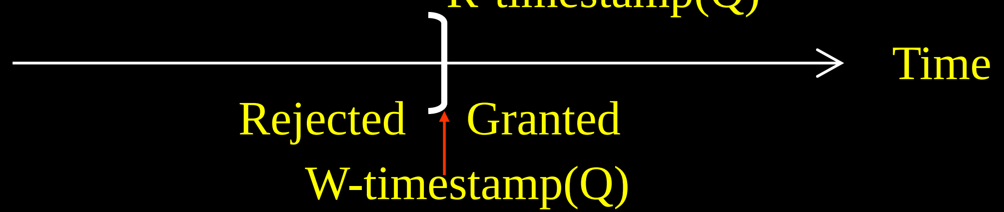
Atomic Transactions – Concurrency Control

- R(Q) requested by T_i → check $TS(T_i)$!



- W(Q) requested by T_i → check $TS(T_i)$!





- Rejected transactions are rolled back and restated with a new time stamp.

Failure Recovery – A Way to Achieve Atomicity

- Failures of Volatile and Nonvolatile Storages!
  - Volatile Storage: Memory and Cache
  - Nonvolatile Storage: Disks, Magnetic Tape, etc.
  - Stable Storage: Storage which never fail.
- Log-Based Recovery
  - Write-Ahead Logging
    - Log Records

< Ti starts >

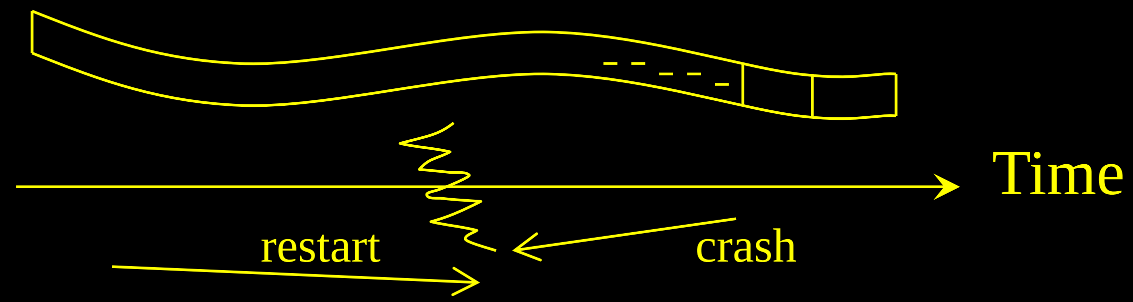
< Ti commits >

< Ti aborts >

< Ti, Data-Item-Name, Old-Value, New-Value>

Failure Recovery

- Two Basic Recovery Procedures:



  - undo(Ti): restore data updated by Ti
  - redo(Ti): reset data updated by Ti
- Operations must be idempotent!
- Recover the system when a failure occurs:
  - "Redo" committed transactions, and "undo" aborted transactions.

Failure Recovery

- Why Checkpointing?
  - The needs to scan and rerun all log entries to redo committed transactions.
- CheckPoint
  - Output all log records, Output DB, and Write <check point> to stable storage!
  - Commit: A Force Write Procedure