General Comments

- Information needed by Concurrency Controllers
  - Locks on database objects (System-R, Ingres, Rosenkrantz…)
  - Time stamps on database objects (Thomas, Reed)
  - Time stamps on transactions (Kung, SDD-1, Schlageter, Bhargava…)
- Observations
  - Time stamps mechanisms more fundamental than locking
  - Time stamps carry more information
  - Checking locks costs less than checking time stamps

General Comments (cont.)

- When to synchronize
  - First access to an object (Locking, pessimistic validation)
  - At each access (question of granularity)
  - After all accesses and before commitment (optimistic validation)
- Fundamental notions
  - Rollback
  - Identification of useless transactions
  - Delaying commit point
  - Semantics of transactions

Definition

A dynamic conflict graph (DCG) for a history $H = <D, T, \Sigma, \Pi>$ is a diagraph $<V,E>$ where V is the set of vertices representing T, the set of transactions; E is the set of edges where $<I,J>$ is an edge if and only if there exist conflicting atomic operations σ_j, σ_j for which $\pi(\pi_1) < \pi(\sigma_j)$.

Lemma: The DCG of a serial history is acyclic.

Theorem: A history is in DCP if and only if the DCG of H is acyclic.

Theorem: In a two-step transaction model (all reads for a transaction precede all writes) whenever there is a transaction rollback in the optimistic approach due to failure in validation. There will be a deadlock in the locking approach and will cause a transaction rollback.

Basic Terms

- Database
- Database entity
- Distributed database
- Program
- Transaction, read set, write set
- Actions
- Atomic
- Concurrent processing
- Conflict
- Consistency
- Mutual consistency
- History
- Serializability
- Serial history

Basic Terms (cont.)

- Serializable history
- Concurrency control
- Centralized control
- Distributed control
- Scheduler
- Locking
- Read lock, write lock
- Two phase locking, lock point
- Live lock
- Dead lock
- Conflict graph
- Timestamp
- Version number
- Rollback
- Validation
- Commit

Basic Terms (cont.)

- Optimistic approach
- Majority voting
- Transaction class

- Crash
- Node failure
- Network partition
- Log
- Redo log
- Undo log
- Recovery
- Abort

Concurrency Control

Interleaved execution of a set of transactions that satisfies given consistency constraints.

Concurrency Control Mechanisms:

- Locking (two-phase locking)
- Conflict graphs (SDD-1)
- Knowledge about incoming transactions or transaction typing
- Optimistic
- Requires validation (backout and starvation)

Some Examples:

- Centralized locking
- Distributed locking
- Majority voting
- Local and centralized validation

- **Locking**

  Problem
  - Maintenance
  - Deadlock
  - Pessimistic
  - Necessary in worst case

  Advantage
  - Do not have to worry about type of consistency constraint

- **Centralized Locking**

  Problem
  - Crash of central
  - Node
  - Congestion/less parallelism

  Advantage
  - Simple and requires low overhead

- **Distributed Locking**

  Problem
  - Lock management (not possible in some cases)

  Advantage
  - More concurrency

Locking Protocols

1. Maintenance
2. Deadlock and livelock
3. Congested (often accessed) node
4. Crashes and release of locks
5. Pessimistic
6. Necessary in the worst case

Conflict-Graph Analysis

- Needs knowledge about incoming transactions (access patterns) not possible in many cases.

- Optimistic
  - Back out
  - Validation
  - Track hole lists

Conflict

Two atomic opns σ_i and σ_j conflict if:

1. They belong to different transactions.
2. Both access the same entity.
3. At least one of them is a WRITE OPN.

R-W conflict

W-R conflict

W-W conflict

Conflict preserving exchange in a history

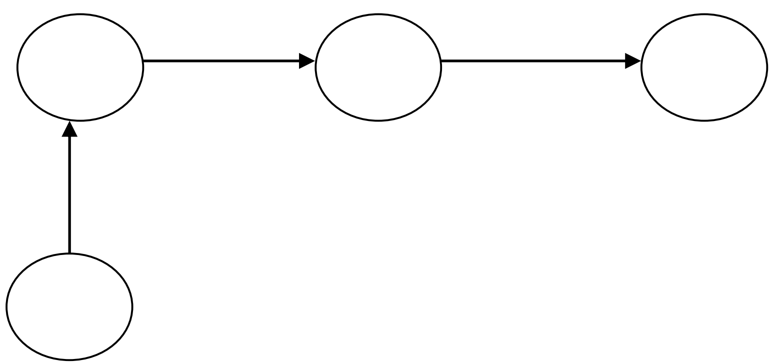
$\theta_1 \; \sigma_i \; \sigma_2 \; \theta_2$

$\equiv \theta_1 \; \sigma_1 \; \sigma_1 \; \theta_2$ (if σ_1, σ_2 do not conflict)

Definition: A Dynamic Conflict Graph (DCG) for a history H = <D,T,Σ,Π> is a diagraph <V,E> where V is the set of vertices representing T, the set of transactions; E is the set of edges where <I,J> is an edge if and only if there exist conflicting atomic operations σ_J, σ_J for which $\Pi(\sigma_I) < \Pi(\sigma_J)$.

Lemma: The DCG of a serial history is acyclic.

Theorem: A history is in DCP if an only if the DCG of H is acyclic.



- Restriction on the Read-Write sets

$S(W_i) \subseteq S(R_i)$ for i = 1….

$$\Rightarrow \quad SR \equiv DSR$$
$$SSR \equiv O$$

- Multi-step transactions
- Interpreted transactions
- Distributed databases

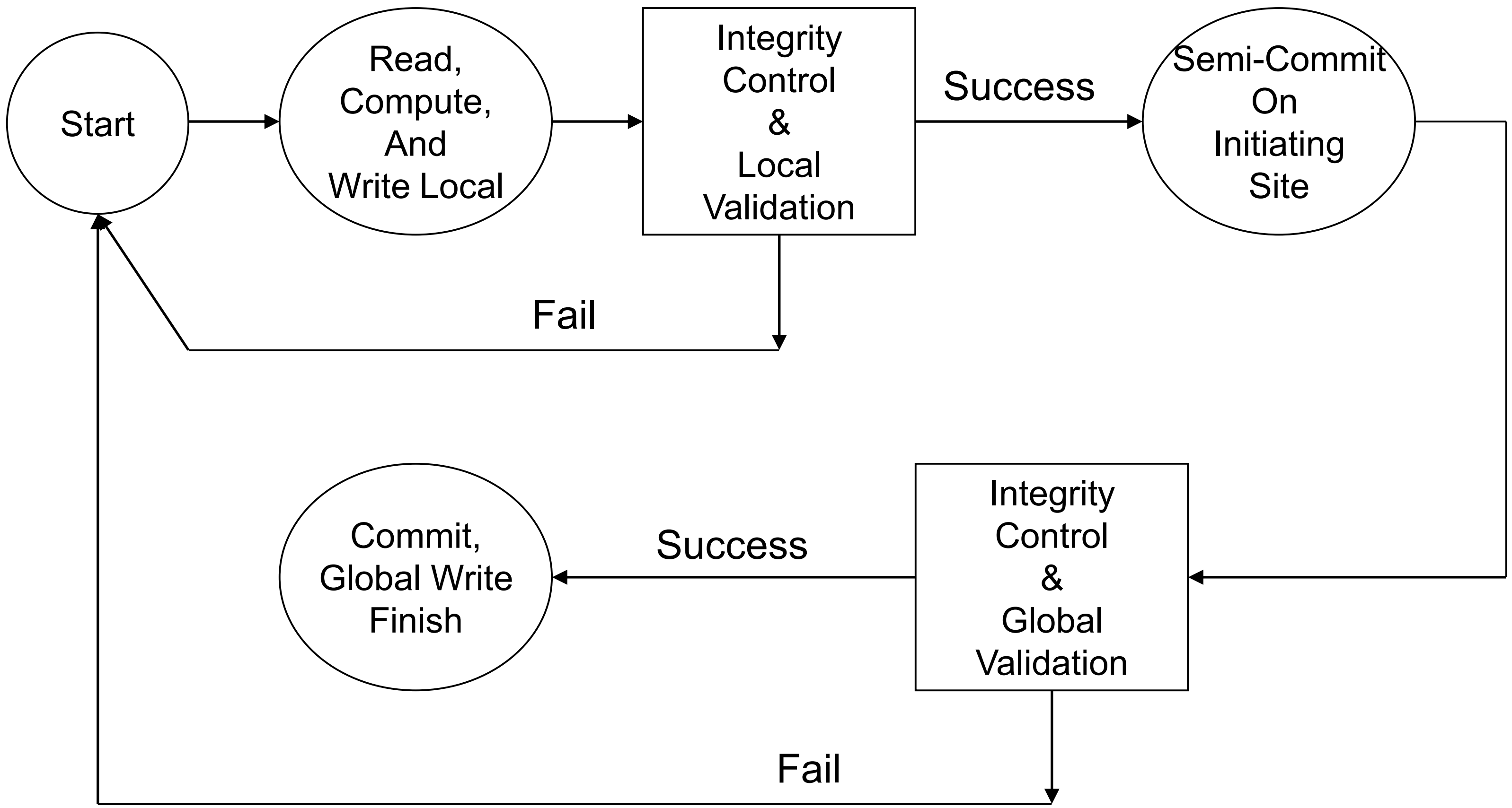


Figure: States of a Transaction

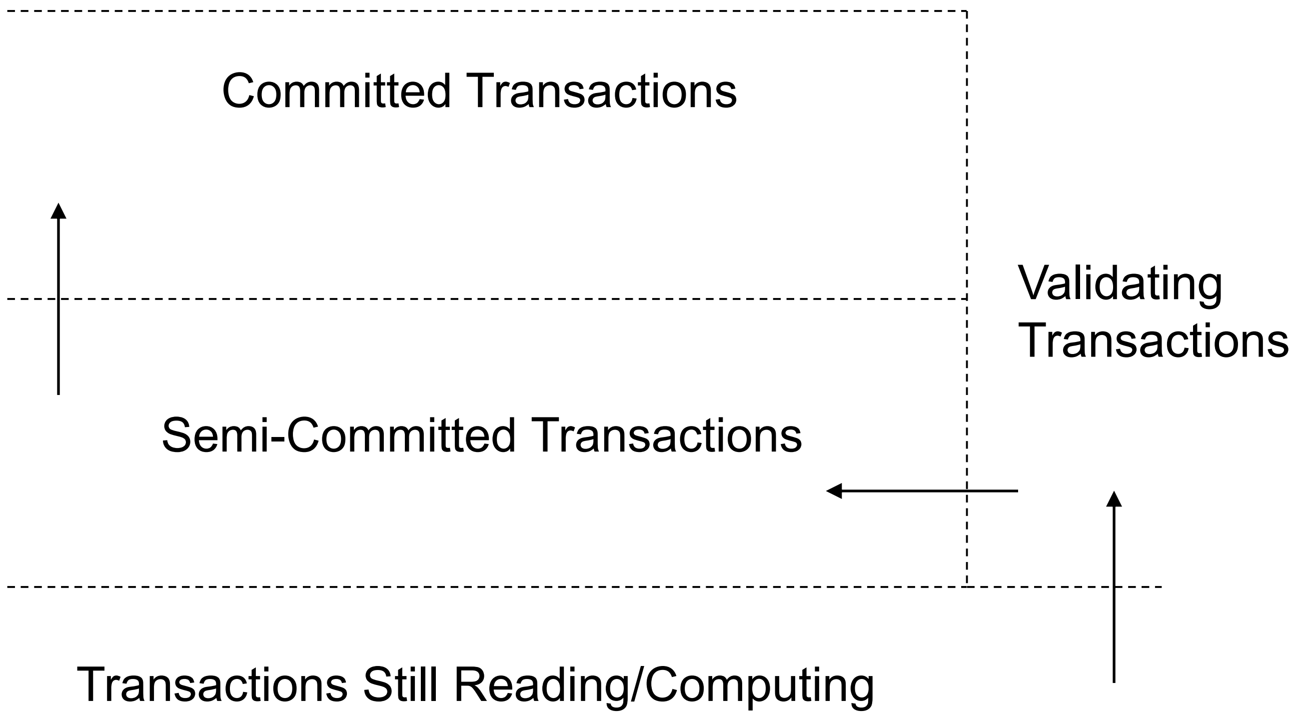
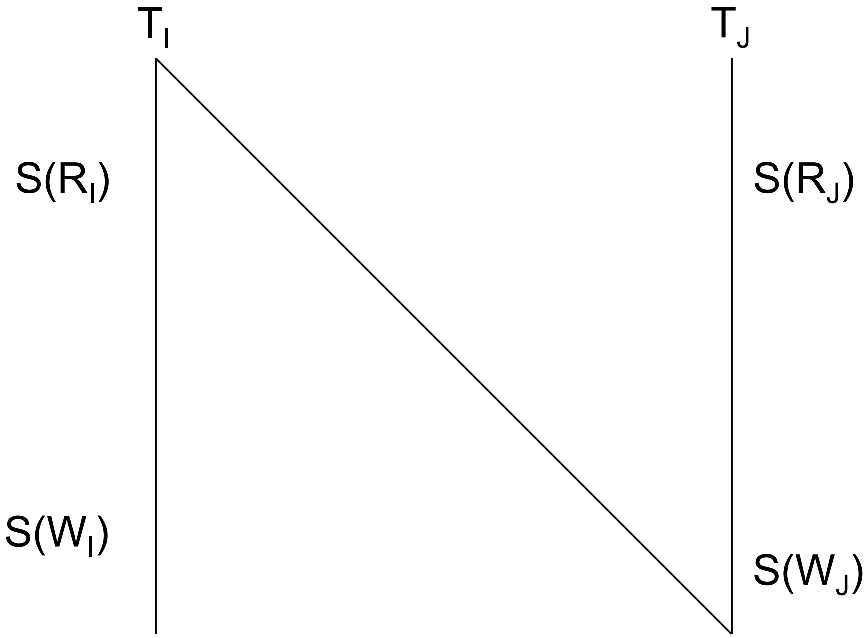


Figure: Transaction Types on a Site



$S(R_I) \cap S(W_J) \neq \emptyset$ AND

$\Pi(R_I) < \Pi(W_J)$

$\Rightarrow T_I \rightarrow T_J$

Locking

$R_I \; R_J \; W_I \; W_J$

Optimistic

$R_I \; R_J \; W_I \; W_J$

$R_I \; R_J \; W_J \; W_I$

Fig. 11.7. Precedence graph for Fig. 11.6.

among the transactions in the cycle. Let the arc $T_{j_{p-1}} \to T_{j_p}$ (take j_{p-1} to be j_k if $p = 1$) be in G because of item A. Then in R, since T_{j_p} appears before $T_{j_{p-1}}$, the final formula for A applies a function f associated with some LOCK A—UNLOCK A pair in T_{j_p} before applying some function g associated with a LOCK A—UNLOCK A pair in $T_{j_{p-1}}$. In S, however, $T_{j_{p-1}}$ precedes T_{j_p}, since there is an arc $T_{j_{p-1}} \to T_{j_p}$. Therefore, in S, g is applied before f. Thus the final value of A differs in R and S, in the sense that the two formulas are not the same, and we conclude that R and S are not equivalent. Thus S is equavalent to no serial schedule. □

A Protocol that Guarantees Serializability

We shall give a simple protocol with the property that any collection of transactions obeying the protocol cannot have a legal, nonserializable schedule. Moreover, this protocol is, in a sense to be discussed subsequently, the best that can be formulated. The protocol is, simply, to require that in any transaction, all locks precede all unlocks.† Transactions obeying this protocol are said to be *two-phase;* the first phase is the locking phase and the second the unlocking phase. For example, in Fig. 11.3, T_1 and T_3 are two-phase; T_2 is not.

Theorem 11.2: If S is any schedule of two-phase transactions, then S is serializable.

Proof: Suppose not. Then by Theorem 11.1, the precedence graph G for S has a cycle, $T_{i_1} \to T_{i_2} \to \cdots \to T_{i_p} \to T_{i_1}$. Then some lock by T_{i_2} follows an unlock by T_{i_1}; some lock by T_{i_3} follows an unlock by T_{i_2}, and so on. Finally, some lock by T_{i_1} follows an unlock by T_{i_p}. Therefore, a lock of T_{i_1} follows an unlock of T_{i_1}, contradicting the assumption that T_{i_1} is two-phase. □

Another way to see why two-phase transactions must be serializable is to imagine that a two-phase transaction occurs instantaneously at the moment it obtains the last of its locks. Then the order in which the transactions reach this point must be a serial schedule equivalent to the given schedule. For if in the given schedule, transaction T_1 locks A before T_2 does, then T_1 surely obtains the last of its locks before T_2 does.

We mentioned that the two-phase protocol in is a sense the best that can be done. Precisely, what we can show is that if T_1 is any transaction that is not two phase, then there is some other transaction T_2 with which T_1 could be

† To avoid deadlock, the locks could be made according to a fixed linear order of the items. However, we do not deal with deadlock here, and some other method could also be used to avoid deadlock.

⋮	⋮
LOCK A	
⋮	
UNLOCK A	
	LOCK A
	LOCK B
	UNLOCK A
	UNLOCK B
LOCK B	
⋮	
UNLOCK B	
⋮	⋮
T_1	T_2

Fig. 11.8. A nonserializable schedule.

run in a nonserializable schedule. Suppose T_1 is not two phase. Then there is some step UNLOCK A of T_1 that precedes a step LOCK B. Let T_2 be:

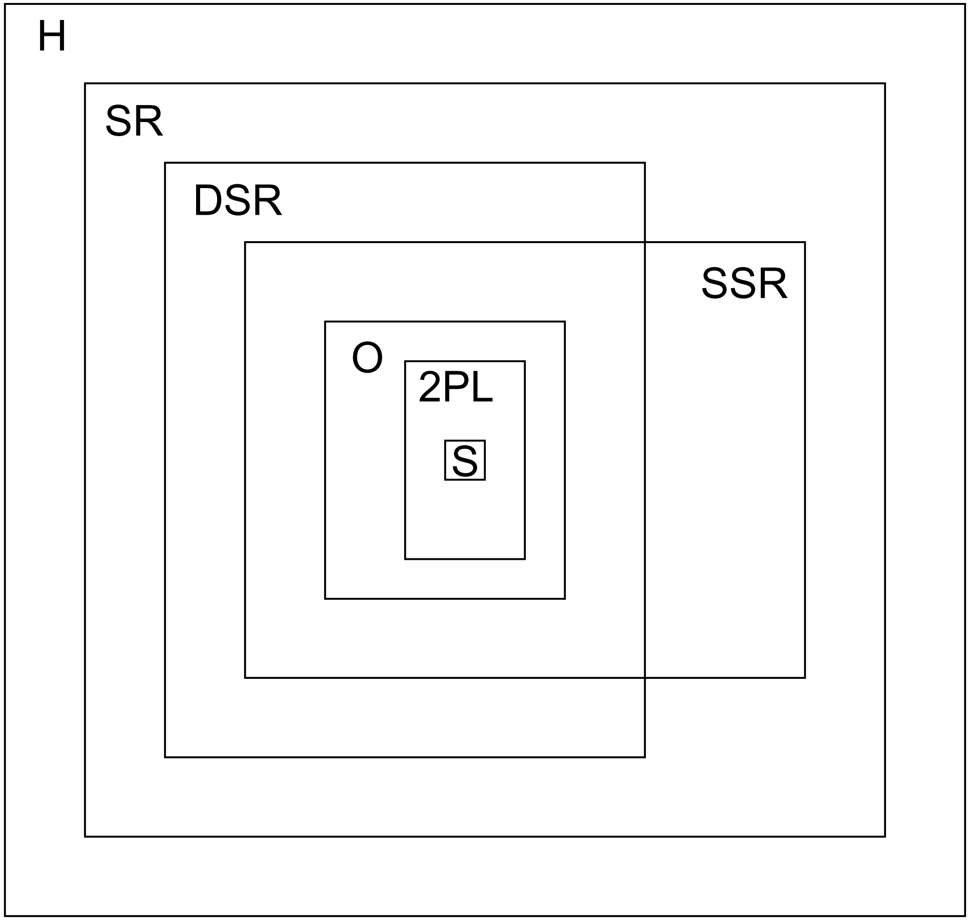
T_2: LOCK A; LOCK B; UNLOCK A; UNLOCK B

Then the schedule of Fig. 11.8 is easily seen to be nonserializable, since the treatment of A requires that T_1 precede T_2, while the treatment of B requires the opposite.

Note that there are particular collections of transactions, not all two-phase, that yield only serial schedules. We shall consider an important example of such a collection in Section 11.5. However, since it is normal not to know the set of all transactions that could ever be executed concurrently with a given transaction, we are usually forced to require all transactions to be two-phase.

11.3 A MODEL WITH READ- AND WRITE-LOCKS

In Section 11.2 we assumed that every time a transaction locked an item it changed that item. In practice, many times a transaction needs only to obtain the value of the item and is guaranteed not to change that value. If we distinguish between a read-only access and a read-write access, we can develop a



Degree of concurrency provided by different classes of histories

Distributed Database Systems

- Computer network (communication system)
- Database systems
- Users (programs, transactions)

Examples:	**Issues**:
Distributed INGRES	Correct processing (serializability)
SDD-1	Consistency of databases (integrity, commitment)
System R*	Resiliency to failures
SIRIUS – DELTA	Performance (response time, throughput)
RAID	Communication delay

Computer Networks:

- Ethernet
- ATM
- FDDI
- ARPANET
- BITNET
- NSF NET
- …

Communications:

- UDP/IP
- TCP/IP
- ISO

Database Systems:

- INGRES
- DB2
- RAID

User Interaction:

- SOL
- Transaction

Definition 1: A history is a quadruple $h = (n, \Pi, M, S)$ where

n is a positive integer,

Π is a permutation of the set

$\Sigma_n = \{R_1, W_1, R_2, W_2, \ldots, R_\eta, W_\eta\}$

equivalently a one-to-one function

$\Pi: \Sigma_\eta \text{ -> } \{1,2,\text{-----},2n\}$

that $\Pi(R_i) < \Pi(W_i)$ for i = 1,2,--n,

M is a finite set of variables representing physical data items,

S is a function mapping Σ_n to 2^M

Set of all histories is denoted by M.

Definition 2: A transaction Ti is a pair (Ri, Wi). A transaction is a single execution of a program. This program may be a simple query statement expressed in a query language.

Definition 3: Read set of Ti is denoted by S (Ri) and Write set of Ti is denoted by S(Wi).

Definition 4: A history $h = (n, \Pi, M, S)$ is serial if $\Pi(Wi) = \Pi(Ri) + 1$ for all $i = 1,2,---n$. In other words, a history is serial if Ri immediately precedes Wi in it for $I = 1,2---n$.

Definition 5: A history is serializable if there is some serial history hs such that the effect of the execution of h is equivalent to hs. Note serializability requires only that there exists some serial order equivalent to the actual interleaved execution history. There may in fact be several such equivalent serial orderings.

Definition 6: A history h is strongly serializable if in hs the following conditions hold true:

- $(Wi) = \Pi(Ri) + 1$
- $(Ri + 1) = \Pi(Wi) + 1$

If ti + 1 is the next transaction that arrived and obtained the next time-stamp after Ti. In strongly serializable history, the following constraint must hold "If a transaction Ti is issued before a transaction Tj, then the total effect on the database should be equivalent to the effect that Ti was executed before Tj.

Note if Ti and Tj are independent, e.g., $\{S(Ri) \cup S(Wi)\} \cap \{S(Rj) U S(Wj)\} = ø$ then the effect of execution TiTj or TjTi will be the same.

history $h = (n, \pi, V_1 S)$

$$\bar{h} = (n + 2, \bar{\pi}, V_1 \bar{S})$$

$$h = T_{n+1} \cdot h \cdot T_{n+2}$$

Live transaction (set can be found in O(n · |V|).

Two histories are equivalent (≡) if they have the same set of live transactions.

Equivalence can be determined O(n · |V|).

Theorem: Testing whether a history h is serializable is NP-complete even if h has no dead transactions.

- Polygraph: Pair of arcs between nodes
- Satisfiability: Problem of Boolean formulas in conjuctive normal forms with two-/three literals

  (SAT)

  (Non-circular)

Concentration of histories

$$h_1 = (n_1, \pi_1, V_1, S_1)$$

$$h_2 = (n_2, \pi_2, V_2, S_2)$$

$$h_1 \circ \ldots n_2, \tau, V_1, P)$$

$$\tau(w_i) = \pi_1(w_i) \quad i \leq n$$

$$\tau(w_i) = \pi_2(w_{i-n}) + 2n \quad \text{for} \quad i > n$$

same true for Ri

$$h_1 = R_1 W_1$$

$$h_2 = R_1 W_1$$

$$h_1 \circ \ldots R_2 W_2$$

Two-Phase Locking

$h = (n, \pi, V, S)$ is 2PL

If $\exists$ distinct non-integer real numbers

$l_1, \ldots, l_n$ such that

(a) $\pi(R_i) < l_i < \pi(W_i)$ for i = 1, ..., n

(b) If $S(R_i) \cap S(W_j) \neq \emptyset$, $i \neq j$, and $\pi(R_i) < \pi(W_j)$, then $l_i < l_j$

(c) If $S(W_i) \cap S(W_j) \neq \emptyset$ and $\pi(W_i) < \pi(W_j)$, then $\pi(W_i) < l_j$

Definition G2PL

A history h is in the global two-phase locking (G2PL) class iff there exists a set of global lock points $\{L_i | i \in T\}$ such that for transactions i and j:

i) $\pi(\alpha_i) \leq L_i \leq \pi(\omega_i)\ \forall i \in T$.

ii) If σ_i and σ_j conflict, and $\pi(\sigma_i) < \pi(\sigma_j)$ then

a) $L_i < L_j$, and

b) $\pi(\sigma_i) < L_j$.

Definition L2PL

A history is in the local two-phase locking (L2PL) class iff there exists a set of local lock points $\{L_i^j | i \in T, j \in N\}$ such that for transactions i and j

i) $\forall i \in T\ L_i^k \leq \pi^k(\sigma_i)$ if $\pi(\omega_i) \leq \pi(\sigma_i)$, and

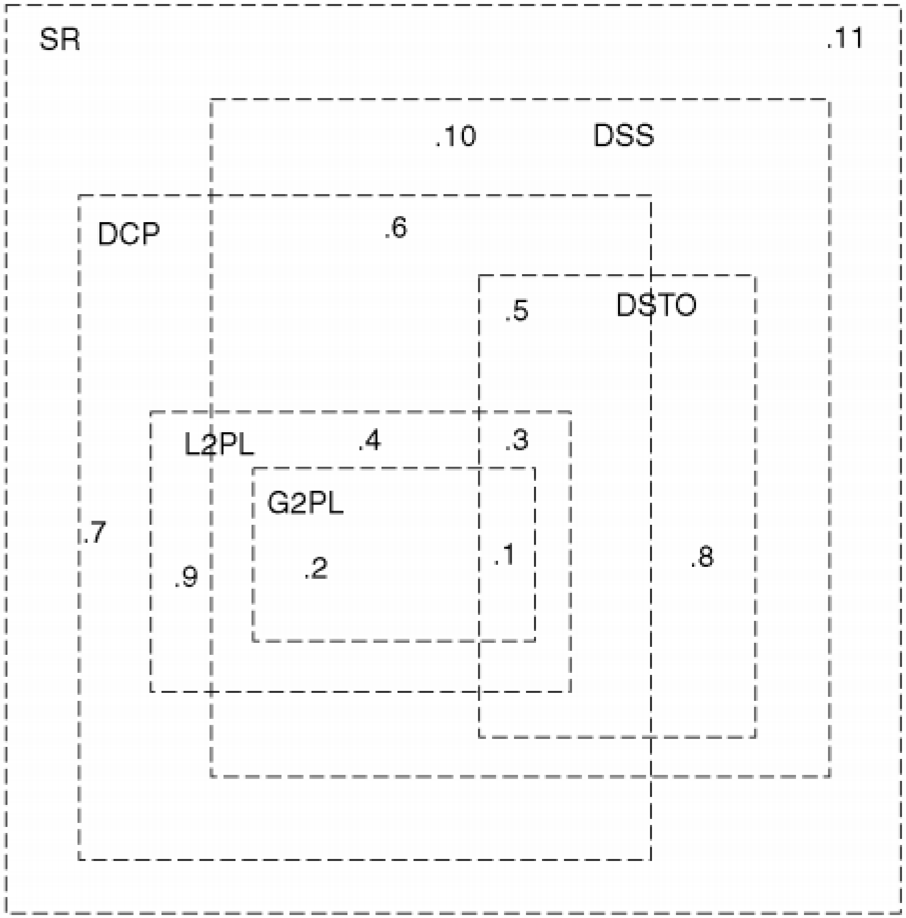
$\pi^k(\alpha_i) \leq L_i^k$ if α_i is on node k.

ii) If σ_i and σ_j conflict on node k, and $\pi^k(\sigma_i) < \pi^k(\sigma_j)$ then

a) $L_i^k < L_j^k$, and

b) $\pi^k(\sigma_i) < L_j^k$,

iii) $L_i^k < L_j^k \Leftrightarrow L_i^m < L_j^m\ \forall k, m \in N$.



All the classes G2PL, L2PL, DCP, DSTO, and DSS are serializable and form a hierarchy based on the degree of concurrency.
SR is the set of all serializable histories.