# GLOSSARY

**Abort.** The transaction operation that a program uses to indicate that the transaction it is currently executing has terminated abnormally and its effects should be obliterated.

**aborted.** (Informal) The state of a transaction after the DBS has processed the transaction's Abort operation.

**aborted.** (Formal) Transaction $T_i$ is aborted in history $H$ if $a_i \in H$.

**abort list.** A list of the identifiers of the set of committed transactions, stored in stable storage.

**abort record.** A log record that says that a particular transaction has aborted.

**ACA.** Acronym for "avoids cascading aborts."

**ACP.** Acronym for "atomic commitment protocol."

**active.** (Informal) The state of a transaction that has started but has not yet become committed or aborted.

**active.** (Formal) Transaction $T_i$ is active in $H$ if there exists some $p_i \in H$ but $a_i \notin H$ and $c_i \notin H$.

**active list.** A list of the identifiers of the set of active transactions, stored in stable storage.

**after image.** The after image of data item $x$ with respect to transaction $T_i$ is the (last) value written into $x$ by $T_i$.

**aggressive scheduler.** A scheduler that tends to avoid delaying operations by trying to schedule them immediately, possibly at the expense of rejecting other operations later on.

**ancestor.** In a dag, if there is a path from $a$ to $b$ then $a$ is an *ancestor* of $b$.

**archive checkpoint record.** A checkpoint record written by an archive checkpointing procedure.

**archive checkpointing.** Checkpointing performed on the archive database.

**archive database.** The backup copy of a database used for media failure handling.

**archive shadow directory.** A directory that defines the state of the database at the time that checkpointing begins.

**archiving.** The media failure handling technique where values of data items are periodically written to a backup copy of the database, called the archive database. The archive database and log are used to recover the stable database in the event of a media failure.

**atomic commitment protocol (ACP).** A protocol that ensures that a transaction terminates consistently, meaning that it either commits at all sites or aborts at all sites, even if failures occur during the protocol.

**atomic operations.** A database system processes operations atomically if it behaves as if it processes them sequentially, one at a time.

**available.** A copy $x_A$ of a data item or directory at site $A$ is available to site $B$ if $A$ correctly executes each Read and Write on $x_A$ issued by $B$ and $B$ receives $A$'s acknowledgment of that execution. A copy $x_A$ is available if it is available to every other site.

**available copies algorithm.** An enhanced form of the write-all-available approach to replicated data, which guarantees one-copy serializability in the presence of site failures but not communications failures.

**avoids cascading aborts.** (Informal) A DBS avoids cascading aborts if it delays each Read($x$) until all transactions that had previously issued Write($x$) have either committed or aborted.

**avoids cascading aborts.** (Formal) History $H$ avoids cascading aborts (ACA) if, whenever $T_i$ reads $x$ from $T_j$, $c_j < r_i[x]$.

**Basic TO.** The TO scheduler that schedules each operation right away if it can do so without violating the TO Rule. Otherwise, it rejects the operation.

**before image.** The before image of a Write($x$) operation is the value of $x$ just before this operation executed.

**blind write.** A Write on some data item $x$ by a transaction that did not previously read $x$. That is, $w_i[x]$ is a blind write if $r_i[x] \ll w_i[x]$.

**blocking.** The state of an operational process in an atomic commitment protocol when it must await the repair of one or more failures before it can continue processing.

**blocking policy.** A scheduling policy, such as Strict 2PL, which resolves almost all conflicts by blocking one of the conflicting transactions.

**broadcasting.** The activity of sending a message concurrently to many sites.

**B-tree.** A type of tree structured index. See Section 3.13.

**B-tree locking.** Tree locking protocols specialized to B-trees.

**cache.** The area of volatile storage in which the cache manager places copies of recently accessed data items.

**cache consistent checkpointing.** The checkpointing scheme that stops processing new operations (leaving active transactions in a blocked state), waits for in-progress operations to complete, flushes all cache slots, and marks the commit and abort lists to indicate that the Flushes took place.

**cache directory.** A table that gives the name of each data item in the cache and the number of its associated slot.

**cache manager.** The database system module that physically reads and writes the values of data items in the database, and buffers those values in a volatile storage area called the cache.

**cache slot.** An area of cache that can store the value of one data item.

**careful replacement algorithm.** The no-undo/no-redo algorithm described in Section 6.7.

**cascadeless.** A synonym for "avoids cascading aborts."

**certified version.** In two version 2PL, a version written by a committed transaction.

**certifier.** A scheduler that immediately outputs all operations submitted by a transaction except its Commit. It avoids nonserializable executions by rejecting the Commits of certain transactions. Such schedulers are often called "optimistic."

**Checkpoint.** The recovery manager procedure that performs the checkpointing activity.

**checkpointing.** An activity that writes information to stable storage during normal operation in order to reduce the amount of work Restart has to do after a failure.

**checkpoint record.** A log record that documents the completion of a checkpoint.

**child.** If $(a, b)$ is an edge in a dag, then $b$ is called a *child* of $a$.

**CM.** Acronym for "cache manager."

**Commit.** The transaction operation that a program uses to indicate that it has completed executing the current transaction and that the effects of that transaction should be made permanent.

**commit consistent checkpointing.** The checkpointing scheme that stops processing new transactions, waits for all active transactions to commit or abort, flushes all dirty cache slots, and then marks the end of the commit list.

**commit list.** A list of the identifiers of the set of committed transactions, stored in stable storage.

**commit record.** A log record that says that a particular transaction has committed.

**committed.** (Informal) The state of a transaction after the DBS has processed the transaction's Commit operation.

**committed.** (Formal) Transaction $T_i$ is committed in history $H$ if $c_i \in H$.

**committed database state.** The database state in which each data item contains its last committed value with respect to a given execution.

**committed projection.** The committed projection of history $H$, denoted $C(H)$, is the history obtained from $H$ by deleting all operations that do not belong to transactions committed in $H$.

**communication failure.** An event that causes two or more sites to be unable to exchange messages.

**communication topology.** The specification of who sends messages to whom.

**compatibility matrix.** A table each of whose rows and columns corresponds to an operation type and each of whose entries indicates whether or not the corresponding operation types are compatible (i.e., do not conflict).

**compatible.** Histories $H$ and $H'$ are compatible if they have the same operations, and $p <_H q$ implies $p <_{H'} q$.

**complete graph.** A graph that has an edge between every pair of nodes.

**complete history.** A complete history $H$ over a set of transactions $T = \{T_1, \ldots, T_n\}$ is a partial order with ordering relation $<_H$ where

1. $H = \bigcup_{i=1}^{n} T_i$,
2. $<_H \supseteq \bigcup_{i=1}^{n} <_i$, and
3. for any two conflicting operations $p, q \in H$, either $p <_H q$ or $q <_H p$.

**component.** A set of operational sites that, due to a network partition, can communicate with each other, but with no other operational sites.

**conflict.** Two operations conflict if their order of execution affects either the state of the database or the value that one of them returns. In the Read-Write model, two operations conflict if they operate on the same data item and at least one of them is a Write.

**conflict-based.** A scheduler is conflict-based if it bases all of its decisions on ordering conflicting operations in a consistent way. Conflict-based schedulers produce only conflict serializable histories.

**conflict equivalence.** See equivalence.

**conflict serializable.** See serializable.

**connected.** A graph $G = (N, E)$ is *connected* if there is a path connecting every pair of nodes.

**conservative scheduler.** A scheduler that tends to delay operations, in order to avoid rejecting other operations later on.

**conservative SGT.** An SGT scheduler that uses predeclaration to help it determine which operations to delay in order to avoid rejecting other operations later on.

**conservative TO.** A type of TO scheduler that delays operations that could be run without violating the TO Rule, in order to reduce the chance that it will have to reject other conflicting operations with smaller timestamps later on.

**Conservative 2PL.** A two phase locking protocol in which each transaction obtains all of the locks it needs before any of its operations are submitted to the DM.

**consistent state.** A state of the database that satisfies the database's consistency predicates. Intuitively, this means that data item values are internally consistent with each other.

**conversion.** Upgrading a lock to a stronger lock type, such as upgrading a read lock on a data item to a write lock on the same data item.

**cooperative termination protocol.** The termination protocol in which all processes are consulted as to whether the decision is Commit or Abort.

**coordinator.** The process that supervises an atomic commitment protocol.

**copier.** A transaction that initializes a new copy, say $x_A$, by reading an existing copy $x_B$ of $x$ and writing the value of $x_B$ into $x_A$.

**critical section.** A shared program that should be executed by at most one process at a time.

**CSR.** Acronym for "conflict serializable."

**cycle.** A cycle is a simple path where the first and last nodes are identical.

**cyclic restart.** A situation in which a transaction is continually aborted (e.g. selected as a victim of a deadlock) and restarted, but is never given the opportunity to terminate normally.

**dag.** Acronym for "directed acyclic graph."

**dag locking.** A generalization of tree locking to directed acyclic graphs.

**database.** A set of data items.

**database state.** The values of the data items in a database at a particular time.

**data contention.** A situation where transactions are delayed in queues because of lock conflicts.

**data manager.** The data manager (DM) is a composite module of the database system, consisting of a cache manager (CM) and recovery manager (RM).

**database operations.** Operations on data items that are supported by a database system, typically Read and Write.

**database system.** A database system (DBS) is a collection of hardware and software modules that support database operations and transaction operations. In our model, a centralized DBS consists of a cache manager (CM), recovery manager (RM), scheduler, and transaction manager (TM).

**data item.** A named memory area that can contain a value.

**data tree.** A tree structured collection of data items used to direct a tree locking protocol.

**DBS.** Acronym for "database system."

**DC-thrashing.** Thrashing in an idealized system with data contention but no resource contention.

**DC-workload.** $k^2N/D$ where $k$ is the number of locks a transaction requires, $N$ is the multiprogramming level, and $D$ is the number of data items in the database.

**deadlock.** A situation in which each transaction in a set of transactions is blocked waiting for another transaction in the set, and therefore none will become unblocked unless there is external intervention.

**decentralized two phase commit.** The two phase commit protocol in which all processes are in direct communication with each other, and therefore exchange messages directly instead of through a coordinator.

**deferred output.** A transaction $T$'s output statements that the DBS postpones processing until after $T$ commits.

**deferred writing.** In processing Write($x$), the DBS delays distributing Writes to the replicated copies of $x$ until the transaction commits.

**delayed commit.** The heuristic of delaying the commitment of a transaction so that the log buffer occupied by its commit record can fill up before being flushed.

**descendant.** In a dag, if there is a path from $a$ to $b$ then $b$ is a *descendant* of $a$.

**digraph.** A directed graph.

**directed acyclic graph.** A directed acyclic graph is a digraph that contains no cycles.

**directed graph.** A directed graph $G = (N, E)$ consists of a set $N$ of elements called *nodes* and a set $E$ of ordered pairs of nodes, called *edges*.

**directory.** (For stable storage management) A mapping of data items to stable storage locations, used in shadowing.

(For replicated data) A mapping of data items to sites that store copies of those data items.

**directory-oriented available copies.** The available copies algorithm that uses directories to keep track of which copies are operational.

**dirty.** A slot whose dirty bit is set is called dirty.

**dirty bit.** A bit associated with each cache slot that is set iff the value of the data item stored in the cache slot was updated since it was last flushed.

**distributed database system.** A collection of sites connected by a computer network, where each site is a centralized database system that stores a portion of the database.

**distributed deadlock.** A deadlock in which two or more of the waiting situations occur at different sites.

**distributed transaction log.** A stable log in which the coordinator and participants in an atomic commitment protocol record information about distributed transactions.

**DL.** Acronym for "dag locking."

**DM.** Acronym for "data manager."

**down.** The state of a site that has failed (and not yet recovered).

**election protocol.** A protocol by which a set of processes select a unique coordinator.

**EOF.** An end-of-file marker.

**equivalence.** Two histories are (conflict) equivalent, denoted $\equiv$, if they are defined over the same transactions, have the same operations, and order conflicting operations of nonaborted transactions in the same way.

An MV history $H$ is equivalent to an MVor 1V history $H'$ if the operations of $H$ and $H'$ are in one-to-one correspondence, and $H$ and $H'$ have the same reads-from relationships.

Two RD histories over $T$ are equivalent if they are view equivalent, that is, if they have the same reads-from relationships and final writes.

An RD history $H$ over $T$ is equivalent to a 1C history $H_{1C}$ over $T$ if

1. $H$ and $H_{1C}$ have the same reads-from relationships *on data items* (i.e., $T_j$ reads-$x$-from $T_i$ in $H$ iff the same holds in $H_{1C}$), and
2. for final write $w_i[x]$ in $H_{1C}$, $w_i[x_A]$ is a final write in $H$ for some copy $x_A$ of $x$.

**Exclude (EX).** A transaction that updates directories to record the failure of a copy in the directory-oriented available copies algorithm.

**execution.** An informal term denoting the effects of operations issued by programs and performed by a computer.

**fail-stop.** Sites only fail by stopping. That is, they never perform incorrect actions.

**failure-recovery serialization graph.** Given an RD history $H$ over transactions $\{T_0, \ldots, T_n\}$, a *failure-recovery serialization graph* (*FRSG*) for $H$ is a directed graph with nodes $N$ and edges $E$ where:

$$N = \{T_0, \ldots, T_n\} \cup \{\text{create}[x_A] \mid x \text{ is a data item and } x_A \text{ is a copy of } x\} \cup \{\text{fail}[x_A] \mid x \text{ is a data item and } x_A \text{ is a copy of } x\}$$

$$E = \{T_i \rightarrow T_j \mid T_i \rightarrow T_j \text{ is an edge of SG}(H)\} \cup E1 \cup E2 \cup E3$$

where

$$E1 = \{\text{create}[x_A] \rightarrow T_i \mid T_i \text{ reads or writes } x_A\}$$

$$E2 = \{T_i \rightarrow \text{fail}[x_A] \mid T_i \text{ reads } x_A\}$$

$$E3 = \{T_i \rightarrow \text{create}[x_A] \text{ or } \text{fail}[x_A] \rightarrow T_i \mid T_i \text{ writes some copy of } x, \text{ but not } x_A\}$$

**Fetch.** The cache manager operation that reads a data item from stable storage into cache.

**final write.** A Write operation $w_i[x]$ in history $H$ is a final write if $a_i \notin H$ and for any $w_j[x] \in H$ $(j \neq i)$, either $w_j[x] < w_i[x]$ or $a_j \in H$.

Given RD history $H$, $w_i[x_A]$ is a final write for $x_A$ in $H$ if $a_i \notin H$ and for all $w_j[x_A]$ in $H$ $(j \neq i)$, either $w_j[x_A] < w_i[x_A]$ or $a_j \in H$.

**Flush.** The cache manager operation that writes a data item from a (dirty) cache slot to stable storage.

**FRSG.** Acronym for "failure-recovery serialization graph."

**fuzzy checkpointing.** The checkpointing scheme that stops processing new operations (leaving active transactions in a blocked state), waits for in-progress operations to complete, and appends to the commit and abort lists a list of the data items stored in dirty cache slots.

**Garbage Collection Rule.** The Garbage Collection Rule states that an entry $[T_i, x, v]$ can be removed from the log iff (1) $T_i$ has aborted or (2) $T_i$ has committed but some other committed transaction wrote into $x$ after $T_i$ did (hence $v$ is not the last committed value of $x$).

**global deadlock detector.** In a distributed database system, a single process that detects deadlocks by finding cycles in a global waits-for graph.

**global waits-for graph.** In a distributed database system, it consists of the union of the edges in the waits-for graph at every site.

**granularity (of a data item).** The amount of data contained in a data item, e.g., a word of memory, a page of a disk, or a record of a file.

**granularity curve.** Transaction throughput as a function of lock granularity.

**group commit.** See delayed commit.

**growing phase.** In two phase locking, the phase during which a transaction obtains locks.

**handshake.** The sequence of events of passing an operation to a module, waiting for an acknowledgment, and passing another operation. A handshake is used to control the order in which a module executes operations.

**history.** A prefix of a complete history.

**home site.** The site where a distributed transaction $T$ originated and that is therefore the coordinator of $T$'s atomic commitment protocol.

**hot spot.** A portion of the database that is accessed especially frequently.

**idempotent.** The property of Restart that any sequence of incomplete executions of Restart followed by a complete execution of Restart has the same effect as just one complete execution of Restart.

**iff.** Abbreviation for "if and only if."

**immediate writing.** The DBS processes Write($x$) by distributing Writes to replicated copies of $x$ at the moment it receives Write($x$) from the transaction.

**Include (IN).** A transaction that updates directories to record the creation or recovery of a copy in the directory-oriented available copies algorithm.

**incomparable.** In a partial order, if neither of two distinct elements precedes the other, the two elements are incomparable.

**inconsistent retrieval.** A situation where a retrieval program reads some data item $x$ before an update program writes $x$, but reads another data item $y$ after that same update program writes $y$.

**independent failure modes.** Two or more storage devices have independent failure modes if no single failure event can destroy data on more than one device.

**independent recovery.** In an atomic commitment protocol, the ability of a recovering process to reach a decision without communicating with other processes.

**index.** A data structure consisting of a set of index entries that map field values into pointers to records with those field values.

**index entry.** A field value and a list of pointers to records with that field value.

**index locking.** A locking method for avoiding phantoms that sets locks on index entries to prevent accesses to records with the field values indicated by those entries.

**in-place updating.** The DM maintains exactly one copy of each data item in stable storage and that copy is overwritten by each Write.

**integrated scheduler.** A scheduler obtained by integrating an rw synchronizer with a ww synchronizer.

**intentions list.** The list of update records for a transaction, which is applied after a transaction commits, using a no-undo/redo recovery manager algorithm.

**intention lock.** A coarse granularity lock that indicates the owner of the lock has a certain type of lock for a finer granularity data item. For example, an intention read lock on a file indicates that the owner may have a read lock on one or more records of that file.

**interfere.** Loosely speaking, two concurrently executing programs interfere if they interact in undesirable ways. We deliberately leave the definition of this term imprecise.

**interleaved.** The operations of two programs (or transactions) are interleaved if an operation of one program executes in between two operations of the other program.

**invalidated.** In multiversion TO, if the scheduler rejects $w_i[x]$ because it already processed some $r_j[x_k]$ where $ts(T_k) < ts(T_i) < ts(T_j)$, then we say that $w_i[x]$ would have invalidated $r_j[x_k]$.

**last committed value.** The last committed value of a data item $x$ in some execution is the value last written into $x$ in that execution by a committed transaction.

**least recently used.** The replacement strategy that replaces the cache slot least recently accessed.

**linear two phase commit.** The two phase commit protocol in which processes are linearly ordered and can only communicate with their left and right neighbors.

**LM.** Acronym for "Lock Manager."

**lock.** (Noun) A reservation that prevents other transactions from obtaining certain other (conflicting) locks.
(Verb) The operation of setting or obtaining a lock.

**lock coupling.** The tree locking technique whereby a transaction obtains locks on a node $N$'s children before releasing its lock on $N$.

**lock escalation.** A locking method used in conjunction with multigranularity locking whereby if a transaction obtains too many locks at one granularity, it increases the granularity of its subsequent lock requests.

**lock instance graph.** A set of data items structured according to a lock type graph.

**lock manager.** The software module that services the Lock and Unlock operations.

**lock type graph.** A directed acyclic graph that specifies the relative coarseness of granularity of locks in multigranularity locking.

**locked point.** In two phase locking, any moment at which a transaction owns all of its locks.

**log.** A representation of the history of execution, stored in stable storage and used by the recovery manager to restore the last committed values of data items.

**log sequence number.** The address of a log entry in the log.

**logical log.** A log whose entries describe higher level operations than Write.

**lost update.** An update that is overwritten before being read, thereby producing a nonserializable result. The canonical example of a lost update is when two transactions read the old value of a data item and then subsequently both write a new value for that data item.

**LSN.** Acronym for "log sequence number."

**LSN-based logging algorithm.** The partial data item logging algorithm where each data item contains the LSN of the update record whose corresponding operation was the last operation to write into that data item.

**majority.** More than half of a set.

**master record.** In the no-undo/no-redo algorithm, the master record indicates which of two copies of the database in stable storage contains the committed state.

**media failure.** A failure in which a portion of the contents of stable storage is lost.

**message.** A block of data transferred from one transaction to another. We assume that messages are exchanged by being written to and read from the database.

**MGL.** Acronym for "multigranularity locking."

**minimal cycle.** A cycle is *minimal* if, for every two nodes $n_i$ and $n_j$ in the cycle, $(n_i, n_j) \in E$ implies $(n_i, n_j)$ is in the cycle.

**mirroring.** The technique of storing the same data in the same locations of two storage devices to protect against the failure of one of the two devices.

**missing writes.** In the available copies algorithm, a transaction $T_i$'s Write($x_A$) is missing if $T_i$ wrote $x$ but not $x_A$.

**missing writes algorithm.** The algorithm for replicated data in which the DBS uses the write-all approach in the absence of failures, and uses the quorum consensus algorithm in the presence of failures.

**mixed scheduler.** An integrated scheduler where different scheduling rules (i.e. 2PL, TO, SGT) are used for rw and ww synchronization.

**MPL.** Acronym for "multiprogramming level."

**multigranularity locking.** A locking method whereby different transactions can lock different granularity data items.

**multiprogramming level.** The number of active transactions.

**multiversion concurrency control.** A concurrency control algorithm in which each Write on $x$ produces a new version, and when processing a Read, the scheduler must select which version to read.

**multiversion history.** A partial order $H <$ over $T$ with ordering relation $<$ such that

1. $H = h(\cup_{i=1}^{n} T_i)$ for some translation function $h$,
2. for each $T_i$ and all operations $p_i$, $q_i$ in $T_i$, if $p_i <_i q_i$ then $h(p_i) < h(q_i)$, and
3. if $h(r_j[x]) = r_j[x_i]$, then $w_i[x_i] < r_j[x_i]$;
4. if $w_i[x] <_i r_i[x]$ then $h(r_i[x]) = r_i[x_i]$.

**multiversion serialization graph.** The multiversion serialization graph for MV history $H$ and version order $\ll$, MVSG($H$, $\ll$), is SG($H$) with the following *version order edges*

added: for each $r_k[x_j]$ and $w_i[x_i]$ in $H$ where $i$, $j$, and $k$ are distinct, if $x_i \ll x_j$ then include $T_i \rightarrow T_j$; otherwise, include $T_k \rightarrow T_i$.

**multiversion TO.** A TO scheduler that processes operations first-come-first-served. It translates $r_i[x]$ into $r_i[x_k]$, where $x_k$ is the version of $x$ with largest timestamp $\leq$ ts($T_i$). It rejects $w_i[x]$ if it has already processed some $r_j[x_k]$ where ts($T_k$) < ts($T_i$) < ts($T_j$); otherwise, it translates $w_i[x]$ into $w_i[x_i]$.

**mutual exclusion.** Ensuring that at most one process executes a particular shared program, called a critical section.

**MV.** Acronym for "multiversion."

**MVSG.** Acronym for "multiversion serialization graph."

**MVTO.** Acronym for "multiversion TO."

**network partition.** A combination of site and communication failures that divides up the operational sites into two or more components, where every two sites within a component can communicate, but sites in different components cannot.

**one-copy history.** See history.

**one-copy serial.** A serial MV history $H$ is one-copy serial (1-serial) if for all $i, j$, and $x$, if $T_i$ reads $x$ from $T_j$, then $i = j$ or $T_j$ is the last transaction preceding $T_i$ that writes into any version of $x$.

**one-copy serializable.** An MV history is one-copy serializable (1SR) if it is equivalent to a one-serial MV history. An RD history is one-copy serializable if it is equivalent to a serial one-copy history.

**one version history.** See history.

**operational.** The state of a site that is functioning correctly.

**optimistic scheduler.** See certifier.

**page.** The fixed-sized unit of data that can be atomically written to disk storage.

**parent.** If $(a, b)$ is an edge in a dag, $a$ is called a *parent* of $b$.

**partial data item logging.** A physical logging algorithm that logs the before and/or after images of just those portions of data items that were updated.

**partial data item logging algorithm.** The recovery manager algorithm in Section 6.4 that logs partial data items, uses fuzzy checkpointing, and uses a Restart algorithm that recovers from a system failure by doing a backward scan of the log for undo followed by a forward scan for redo.

**partial failure.** In a distributed system, the situation in which some sites are operational while others are down.

**partial order.** A partial order $L = (\Sigma, <)$ consists of a set $\Sigma$ called the *domain* of the partial order and an irreflexive, transitive binary relation $<$ on $\Sigma$.

**participant.** A process that participates in an atomic commitment protocol, but does not supervise it.

**partition.** A partition of a graph $G$ is a collection $G_1 = (N_1, E_1), \ldots, G_k = (N_k, E_k)$ of subgraphs of $G$ such that each $G_i$ is connected and the node sets of these subgraphs are pairwise disjoint; i.e., $N_i \cap N_j = \{\}$ for $1 \leq i \neq j \leq k$.

**path.** A path in a (directed or undirected) graph $G = (N, E)$ is a sequence of nodes $v_1, v_2, \ldots, v_k$ such that $[v_i, v_{i+1}] \in E$ for $1 \leq i < k$.

**path pushing.** A distributed deadlock detection algorithm where each site exchanges lists of paths in waits-for graphs with other sites in order to find a cycle in the union of those graphs.

**penultimate.** Second to last.

**phantom deadlock.** A situation in which the deadlock detector believes there is a deadlock, but the deadlock doesn't really exist.

**phantom problem.** The concurrency control problem for dynamic databases, that is, where transactions can insert and delete data items.

**physical log.** A type of log that contains information about the values of data items written by transactions.

**piggybacking.** The communication technique whereby two or more messages are packaged as one large message, in order to reduce message transmission cost.

**Pin**($c$). The cache manager operation that makes a cache slot $c$ unavailable for flushing.

**predeclaration.** Using predeclaration, transactions "predeclare" their readsets and writesets, meaning that the scheduler learns (a superset of) the readset and writeset of each transaction before processing any of the transaction's operations.

**predicate locking.** A locking method for avoiding phantoms that sets locks on predicates to prevent accesses to records that satisfy those predicates.

**prefix commit-closed.** A property of histories that, whenever it is true of a history $H$, is also true of the committed projection $C(H')$ of every prefix $H'$ of $H$. Recoverability, cascadelessness, strictness, and serializability (see Theorem 2.3) are prefix commit-closed.

**prefix of a partial order.** $L'$ is a *prefix* of a partial order $L = (\Sigma, <)$, written $L' \leq L$, if $L'$ is a restriction of $L$ and for each $a \in \Sigma'$, all predecessors of $a$ in $L$ are also in $\Sigma'$.

**preserves reflexive reads-from.** A multiversion history preserves reflexive reads-froms if whenever $w_i[x] <_i r_i[x]$, then $h(r_i[x]) = r_i[x_i]$.

**primary copy.** The variation of deferred writing in which all transactions use the same copy of each data item while they are executing.

**process.** The operating system abstraction that corresponds to the independent execution of a sequential program.

**proper ancestor.** In a dag, $a$ is a *proper ancestor* of $b$ if it is an ancestor of $b$ and $a \neq b$.

**pure restart policy.** A scheduling policy in which a transaction is aborted whenever it encounters a lock already held by another transaction.

**pure scheduler.** An integrated scheduler where possibly different versions of the same scheduling rule (i.e., 2PL, TO, or SGT) are used for both rw and ww synchronization.

**Purge.** The operation that deletes obsolete entries from a timestamp table. Used by TO schedulers.

**QC.** Acronym for "quorum consensus."

**query.** A transaction that reads one or more data items but does not write any data items.

**quorum.** Each object $o$ (typically a site or copy) in a set $O$ is assigned a non-negative weight. A quorum of $O$ is a subset of $O$ that has more than half the total weight.

**quorum consensus (QC).** The replicated data algorithm in which the DBS processes a Read($x$) by reading a read quorum of copies of $x$ and selecting the most up-to-date copy, and processes a Write($x$) by writing a write quorum of copies of $x$.

RC. Acronym for "recoverable."

**RC-thrashing.** Thrashing in an idealized system with resource contention but no data contention.

RD. Acronym for "replicated data."

RDSG. Acronym for "replicated data serialization graph."

Read($x$). The database operation that returns the current value of data item $x$.

**read lock.** A reservation to read a data item. Ordinarily (for rw synchronization), if transaction $T_i$ owns a read lock on $x$, denoted $rl_i[x]$, then no other transaction can obtain a write lock on $x$.

**read order.** See replicated data serialization graph.

**readset.** The set of data items a transaction reads.

**reads-from.** (Informal) Transaction $T_j$ reads data item $x$ from transaction $T_i$ in an execution if

1. $T_j$ reads $x$ after $T_i$ has written it,
2. $T_i$ does not abort before $T_j$ reads $x$, and
3. every transaction (if any) that writes $x$ between the time $T_i$ writes it and $T_j$ reads it aborts before $T_j$ reads it.

**reads-from.** (Formal) Transaction $T_i$ reads $x$ from transaction $T_j$ in history $H$ if $w_j[x] < r_i[x]$, $a_j \not< r_i[x]$, and if there is some $w_k[x]$ such that $w_j[x] < w_k[x] < r_i[x]$, then $a_k < r_i[x]$. $T_i$ reads from $T_j$ in $H$ if it reads some data item from $T_j$ in $H$.

**recoverable.** (Informal) An execution is recoverable if, for every transaction $T$ that commits, $T$'s Commit follows the Commit of every transaction from which $T$ read.

**recoverable.** (Formal) History $H$ is recoverable (RC) if, whenever $T_i$ reads from $T_j$ in $H$ and $c_i \in H$, $c_j < c_i$.

**recovery manager.** The database system module that is responsible for the commitment and abortion of transactions. It processes the operations Read, Write, Commit, Abort, and Restart.

**recovery procedure.** A procedure that a site (i.e., a DBS) executes after recovering from failure to bring itself to a consistent state so it can resume normal processing.

**Redo Rule.** The Redo Rule states that before a transaction can commit, the value it produced for each data item it wrote must be in stable storage (e.g. in the stable database or the log).

**reflexive reads-from.** A reads-from relationship where a transaction reads the value it previously wrote.

**replacement strategy.** The criterion according to which the cache manager chooses a slot to flush to make room for a data item being fetched.

**replicated data (RD) history.** A *replicated data* (*RD*) history $H$ over $T = \{T_0, \ldots, T_n\}$ is a partial order with ordering relation $<$ where

1. $H = h(\cup_{i=0}^{n} T_i)$ for some translation function $h$;
2. for each $T_i$ and all operations $p_i$, $q_i$ in $T_i$, if $p_i <_i q_i$, then every operation in $h(p_i)$ is related by $<$ to every operation in $h(q_i)$;
3. for every $r_j[x_A]$, there is at least one $w_i[x_A]$ such that $w_i[x_A] < r_j[x_A]$;
4. all pairs of conflicting operations are related by $<$, where two operations *conflict* if they operate on the same *copy* and at least one of them is a Write; and
5. if $w_i[x] <_i r_i[x]$ and $h(r_i[x]) = r_i[x_A]$, then $w_i[x_A] \in h(w_i[x])$.

**replicated data serialization graph (RDSG).** Given a history $H$, an RDSG $G$ for $H$ is a graph containing SG($H$) and including enough edges such that

1. if $T_i$ and $T_k$ write $x$, then either $T_i \lll T_k$ or $T_k \lll T_i$, and
2. if $T_j$ reads-$x$-from $T_i$, $T_k$ writes some copy of $x$ ($k \neq i$, $k \neq j$), and $T_i \lll T_k$, then $T_j \lll T_k$,

where $n_i \lll n_j$ means there is a path from $n_i$ to $n_j$. If $G$ satisfies (1), it induces a *write order*. If it satisfies (2), it induces a *read order*.

**requires redo.** A recovery manager requires redo if it allows a transaction to commit before all the values it wrote are recorded in the stable database.

**requires undo.** A recovery manager requires undo if it allows an uncommitted transaction to record in the stable database values it wrote.

**resource contention.** A situation where transactions are delayed in queues while trying to obtain the use of certain resources (such as processor, memory, or I/O) because other transactions are presently using those resources.

**Restart.** The RM operation that performs a DBS's recovery procedure.

**restriction of a partial order.** A partial order $L' = (\Sigma', <')$ is a restriction of $L = (\Sigma, <)$ on domain $\Sigma'$ if $\Sigma' \subseteq \Sigma$ and for all $a, b \in \Sigma'$, $a <' b$ iff $a < b$.

**RM.** Acronym for "recovery manager."

**rooted dag.** A dag with a unique source, called its root.

**round.** In an atomic commitment protocol, the maximum time for a message to reach its destination.

**rw serialization graph.** A serialization graph that only contains an edge $T_i \rightarrow T_j$ if $r_i[x] < w_j[x]$ or $w_i[x] < r_j[x]$ for some data item $x$.

**rw synchronization.** Controlling the order in which Reads execute with respect to conflicting Writes.

**rw synchronizer.** A scheduler that only performs rw synchronization.

**scheduler.** The database system module that controls the relative order in which database operations and transaction operations execute, by delaying or rejecting some of those operations.

**serial execution.** An execution in which for every pair of transactions, all of the operations of one transaction execute before any of the operations of the other.

**serial history.** A complete (1V, MV, or RD) history $H$ is serial if for every two transactions $T_i$ and $T_j$ that appear in $H$, either all operations of $T_i$ appear before all operations of $T_j$ or vice versa.

**Serializability Theorem.** Theorem 2.1, which says that a history $H$ is serializable iff SG($H$) is acyclic.

**serializable execution.** An execution $E$ that produces the same output and has the same effect on the database as some serial execution of the same transactions that appeared in $E$.

**serializable history.** A history is serializable (SR) if its committed projection is (conflict) equivalent to a serial history.

**serialization graph.** The serialization graph (SG) for history $H$ over transactions $T = \{T_1 \ldots, T_n\}$, denoted SG($H$), is a directed graph whose nodes are the transactions in $T$ that are committed in $H$ and whose edges are all $T_i \rightarrow T_j$ such that one of $T_i$'s operations precedes and conflicts with one of $T_j$'s operations in $H$.

The serialization graph of a multiversion history $H$ over $T$ includes edges $T_i \rightarrow T_j$ $(i \neq j)$ such that for some $x$, $T_j$ reads $x$ from $T_i$.

**serialization graph testing.** The scheduling method that explicitly maintains a serialization graph and avoids nonserializable executions by checking for cycles in that graph.

**SG.** Acronym for "serialization graph."

**SGT.** Acronym for "serialization graph testing."

**SGT certifier.** An SGT scheduler that only checks for serialization graph cycles when it receives a transaction's Commit.

**shadow copy.** An old version of a data item.

**shadowing.** The DM maintains more than one copy of each data item in stable storage, so it may write a data item to stable storage without destroying older (shadow) versions of that data item.

**shadow page algorithm.** The no-undo/no-redo algorithm described in Section 6.7.

**shrinking phase.** In two phase locking, the phase during which a transaction releases locks.

**simple path.** A path in a graph is simple if all nodes, except possibly the first and last in the sequence, are distinct.

**site failure.** The event in which a site stops abruptly and the contents of volatile storage are destroyed.

**site quorums.** The approach to handling communication failures in a replicated database in which the connected component of the network that contains a quorum of sites is the only component that can process transactions that access replicated data.

**slot.** See cache slot.

**source.** In a directed acyclic graph, a source is a node with no incoming edges.

**SR.** Acronym for "serializable."

**SSG.** Acronym for "stored serialization graph."

**ST.** Acronym for "strict."

**stable database.** The state of the database in stable storage.

**stable-LSN.** The stable-LSN of a cache slot storing $x$ marks a point in the log where it is known that the value of the stable database copy of $x$ reflects (at least) all of the log records up to that LSN.

**stable storage.** An area of memory that is resistant to processor and operating system failures. It models secondary storage media, such as disk and tape, on typical computer systems.

**Start.** The transaction operation that a program uses to indicate that it wishes to begin executing a new transaction.

**start-2PC record.** In the two phase commit protocol, the record in the coordinator's distributed transaction log that indicates it has begun the protocol.

**Static 2PL.** See Conservative 2PL.

**stored serialization graph.** The serialization graph maintained by an SGT scheduler.

**strength of locks.** Lock type $p$ is stronger than lock type $q$ if for every lock type $o$, $ol_i[x]$ conflicts with $ql_j[x]$ implies $ol_i[x]$ conflicts with $pl_j[x]$.

**strict.** (Informal) A DBS is strict if it delays each Read($x$) and Write($x$) until all transactions that had previously issued Write($x$) have either committed or aborted.

**strict.** (Formal) History $H$ is strict (ST) if whenever $w_j[x] < o_i[x]$, either $a_j < o_i[x]$ or $c_j < o_i[x]$, where $o_i[x]$ is $r_i[x]$ or $w_i[x]$.

**Strict TO.** The TO scheduler behaves like Basic TO, except that it delays each Write $w_i[x]$ until there is no active transaction that issued a Write on $x$.

**Strict 2PL.** A two phase locking protocol where the scheduler releases all of a transaction's locks together, after the transaction commits or aborts.

**subgraph.** A graph $G' = (N', E')$ is a *subgraph* of $G = (N, E)$ if $N' \subseteq N$ and $E' \subseteq E$.

**system failure.** A failure in which the entire contents of volatile storage is lost.

**termination protocol.** A protocol invoked by a process when it fails to receive an anticipated message while in its uncertainty period.

**Thomas' Write Rule.** A TO ww synchronizer that acknowledges but does not process any Write that arrives too late to be processed in timestamp order.

**thrashing.** A situation where increasing the number of transactions in the system causes the throughput to drop.

**three phase commit.** A nonblocking atomic commitment protocol that can tolerate site failures but not communication failures.

**timeout.** An alarm that indicates to a process that a predefined time interval has elapsed.

**timeout`action.** The action that a process must take if its waiting for an event (typically a message) is interrupted by a timeout.

**timeout failure.** A communication failure in which a site believes it cannot communicate with another site because it is using a timeout period that is too short.

**timestamp-based deadlock prevention.** A deadlock prevention technique where a transaction $T_i$ can wait for another transaction $T_j$ only if $T_i$ has higher priority than $T_j$.

**timestamp ordering.** The scheduling method where each transaction is assigned a unique timestamp and conflicting operations from different transactions are scheduled to execute in timestamp order.

**timestamps.** Values drawn from a totally ordered domain. In concurrency control algorithms, timestamps are usually assigned to transactions such that no two transactions have the same timestamp.

**TL.** Acronym for "tree locking."

**TM.** Acronym for "transaction manager."

**TO.** Acronym for "timestamp ordering."

**topological sort.** In a digraph $G$, a sequence of (all) the nodes of $G$ such that if $a$ appears before $b$ in the sequence, there is no path from $b$ to $a$ in $G$.

**TO Rule.** If $p_i[x]$ and $q_j[x]$ are conflicting operations and $i \neq j$, then the DM processes $p_i[x]$ before $q_j[x]$ iff $T_i$'s timestamp is less than $T_j$'s timestamp.

**total failure.** In a distributed system, the situation in which all sites are down. In replicated databases, total failure of a data item occurs when all copies of the data item are down.

**transaction.** (Informal) The execution of one or more programs that include database and transaction operations, beginning with the operation Start, and ending with operation Commit or Abort.

**transaction.** (Formal) A transaction $T_i$ is a partial order with ordering relation $<_i$ where

1. $T_i \subseteq \{r_i[x], w_i[x] \mid x \text{ is a data item}\} \cup \{a_i, c_i\}$;
2. $a_i \in T_i$ iff $c_i \notin T_i$;
3. if $t$ is $a_i$ or $c_i$ (whichever is in $T_i$), for any $p \in T_i$, $p <_i t$; and
4. if $r_i[x], w_i[x] \in T_i$ then either $r_i[x] <_i w_i[x]$ or $w_i[x] <_i r_i[x]$.

**transaction class.** A class is defined by a readset and writeset. A transaction is in a class if its readset and writeset are in the class's readset and writeset (respectively).

**transaction failure.** The event of a transaction issuing an Abort.

**transaction manager.** The database system module that is the interface between transactions and the rest of the database system. It receives each operation from the transaction, performs any necessary preprocessing of the operation (such as appending a transaction identifier to the operation), and then forwards the operation to the appropriate database system module.

**transaction operations.** The operations Start, Commit, and Abort, which are supported by a database system.

**transaction-oriented shadowing.** The recovery manager scheme for undo/no-redo where a transaction's updated data items are written as new versions in stable storage, and the new versions replace the shadow versions after the transaction commits.

**transitive closure.** The transitive closure of a digraph $G = (N, E)$ is the digraph $G^+ = (N, E^+)$ such that $(a, b) \in E^+$ iff there is a non-trivial path from $a$ to $b$ in $G$.

**transitively closed.** A digraph is *transitively closed* if it is equal to its own transitive closure (i.e., $G = G^+$).

**tree.** A *tree* is a rooted dag with the additional property that there is a unique path from the root to each node.

**tree locking.** The non-two-phase locking protocol in which a transaction can release a lock on a node $N$ of the tree as soon as it has obtained all of the locks it will need on children of $N$.

**two phase commit (2PC).** The atomic commitment protocol that, in the absence of failures, behaves roughly as follows: The coordinator asks all participants to vote; if any participant votes No, the coordinator tells all participants to decide Abort; if all participants vote Yes, then the coordinator tells all participants to decide Commit.

**two phase locking (2PL).** The locking protocol in which each transaction obtains a read (or write) lock on each data item before it reads (or writes) that data item, and does not obtain any locks after it has released some lock.

**two phase rule.** The rule in two phase locking that requires a transaction to set all of its locks before releasing any of them.

**two version 2PL.** A multiversion scheduler that uses 2PL. See Section 5.4 for complete definition.

**TWR.** Acronym for "Thomas' Write Rule."

**uncertain.** In an atomic commitment protocol, the state of a process while it is in its uncertainty period.

**uncertainty period.** In an atomic commitment protocol, the period between the moment a process votes Yes and the moment it has received sufficient information to know what the decision will be.

**undirected graph.** An undirected graph $G = (N, E)$ consists of a set $N$ of elements called *nodes* and a set $E$ of unordered pairs of nodes called *edges*.

**Undo Rule.** The Undo Rule states that if $x$'s location in the stable database presently contains the last committed value of $x$, then that value must be saved in stable storage *before* being overwritten in the stable database by an uncommitted value.

**uninterpreted.** The aspects of an execution that are left unspecified, and can therefore be arbitrary.

**unlock.** The operation of releasing a lock.

**Unpin($c$).** The cache manager operation that makes a previously pinned slot again available for flushing.

**updater.** A transaction that can read and write data items.

**update record.** A log record that documents a Write operation of a transaction.

**validation protocol.** The protocol used by an available copies algorithm to ensure correctness by checking that certain copies are still operational or down just before a transaction commits.

**version number.** In the quorum consensus algorithm, each replicated copy is given a version number, which the DBS uses to select the most up-to-date copy to read.

**version order.** A version order $\ll$ for data item $x$ in MV history $H$ is a total order of versions of $x$ in $H$.

**version order edge.** See multiversion serialization graph.

**victim.** A transaction that is aborted in order to break a deadlock.

**view equivalence.** Two histories $H$ and $H'$ are view equivalent if

1. they are over the same set of transactions and have the same operations;
2. for any $T_i$, $T_j$ such that $a_i$, $a_j \notin H$ (hence $a_i$, $a_j \notin H'$) and any $x$, if $T_i$ reads $x$ from $T_j$ in $H$ then $T_i$ reads $x$ from $T_j$ in $H'$ and
3. for each $x$, if $w_i[x]$ is the final write of $x$ in $H$ then it is also the final write of $x$ in $H'$.

**view serializable.** A history $H$ is view serializable if for any prefix $H'$ of $H$, $C(H')$ is view equivalent to some serial history.

**view update transaction.** In the virtual partition algorithm, a view update transaction updates each site's view of the members of a component (i.e. of a virtual partition).

**virtual partition algorithm.** An algorithm for replicated data. After a communication failure, the DBS can access a data item $x$ only if it is executing in a component that contains a quorum of copies of $x$. Within a component, the DBS uses the write-all approach with respect to the set of copies of $x$ within its component.

**volatile storage.** An area of memory that is vulnerable to hardware and operating system failures. It models main memory on typical computer systems.

**VSR.** Acronym for "view serializable."

**Wait-Die.** A timestamp-based deadlock prevention technique whereby a transaction aborts if it tries to lock a data item currently locked by a higher priority transaction.

**waits-for graph.** A graph whose nodes are labelled by transaction names, and that contains an edge $T_i \rightarrow T_j$ whenever $T_i$ is waiting for $T_j$ to release some lock.

**workspace model.** Transactions write into a workspace, not the database. When they are ready to commit, the Writes are propagated to the database.

**Wound-Wait.** A timestamp-based deadlock prevention technique whereby a transaction $T_i$ is aborted if a higher priority transaction tries to lock a data item currently locked by $T_i$.

**Write($x$, $val$).** The database operation that changes the value of data item $x$ to $val$.

**write-all.** The approach to replicated data where the DBS translates each Read($x$) into a Read of some copy of $x$, and each Write($x$) into Writes on *all* copies of $x$ (not just those copies at operational sites).

**write-all-available.** The approach to replicated data where the DBS translates each Read($x$) into a Read of some available copy of $x$, and each Write($x$) into Writes on all available copies of $x$.

**write lock.** A reservation to write a data item. Ordinarily (for rw and ww synchronization), if transaction $T_i$ owns a write lock on $x$, denoted $wl_i[x]$, then no other transaction can obtain a read lock or write lock on $x$.

**write order.** See replicated data serialization graph.

**writeset.** The set of data items a transaction writes.

**wrt.** Abbreviation for "with respect to."

**ww serialization graph.** A serialization graph that only contains an edge $T_i \rightarrow T_j$ if $w_i[x] < w_j[x]$ for some data item $x$.

**ww synchronization.** Controlling the order in which conflicting Writes execute.

**ww synchronizer.** A scheduler that only performs ww synchronization.

**1-serial.** Acronym for "one-copy serial."

**1SR.** Acronym for "one-copy serializable."

**1V.** Acronym for "one version."

**2PC.** Acronym for "two phase commit."

**2PL.** Acronym for "two phase locking."

**2PL certifier.** A certifier that rejects a transaction's Commit if the transaction issued any operation that conflicts with that of an active transaction.

**2PL history.** A history produced by a 2PL scheduler.

**2V2PL.** Acronym for "two version 2PL."

**3PC.** Acronym for "three phase commit."