Distributed Transaction Management/ Concurrency Control/Recovery

Replicated Databases - Part 1

Overview

- Distributed Transaction Management/Concurrency Control/Recovery
- 2-Phase Commit (2PC) protocol

- Database Replication – Introduction (continues next week)
  - The benefits of data replication.
  - Basic components of a replication system.
  - Replication model
  - Functional model of replication protocols
  - Replication system consistency

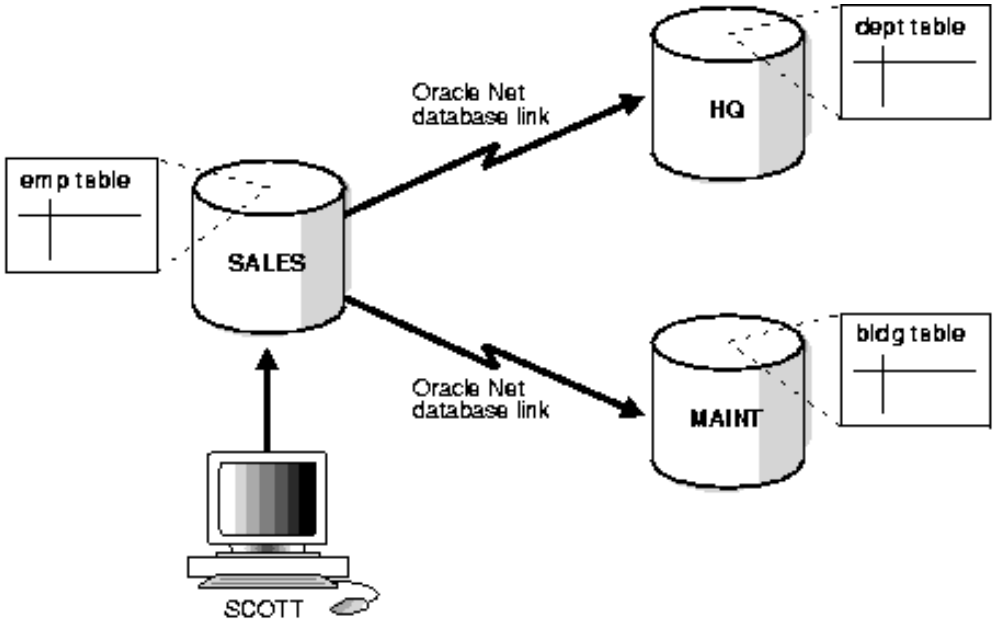
- The lecture slides are based on the ones provided with the course textbook:
  - Connolly. and Begg, C. "*Database Systems - A Practical Approach to Design, Implementation, and Management*.", 6th Ed., Pearson Education Ltd

Distributed Transaction Management

- Distributed transaction accesses data stored at more than one location.
- Divided into a number of *sub-transactions*, one for each site that has to be accessed, represented by an *agent*.
- *Indivisibility* of distributed transaction is still fundamental to transaction concept.
- DDBMS must also ensure indivisibility of each sub-transaction.
- Thus, DDBMS must ensure:
  - synchronization of subtransactions with other local transactions executing concurrently at a site;
  - synchronization of subtransactions with global transactions running simultaneously at same or different sites.
- Transaction manager (transaction coordinator) at each site, to coordinate global and local transactions initiated at that site.
  - Global coordinator vs Local coordinators

A distributed transaction example

- This illustration depicts a distributed database system. There are 3 databases, one local, and two remote.
- A user is shown updating tables on all 3 databases using a distributed transaction.
  - The user Scott updates the local Sales database, the remote HQ database, and the remote Maint database:

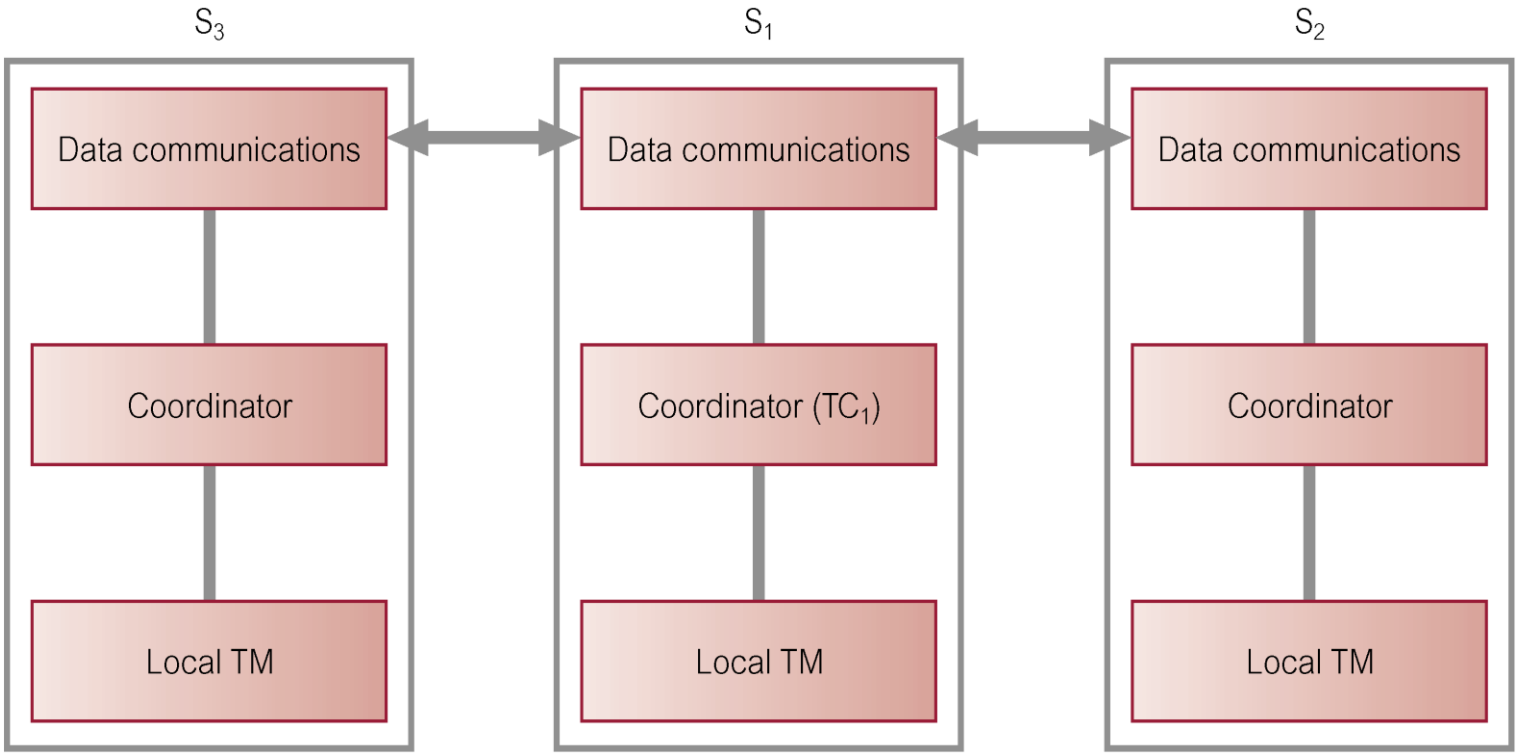



```
UPDATE scott.dept@hq.us.example.com
  SET loc = 'REDWOOD SHORES'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
UPDATE scott.bldg@maint.us.example.com
  SET room = 1225
  WHERE room = 1163;
COMMIT;
```

Coordination of Distributed Transaction

- High-level DBMS modules:
  - *Transaction manager* – coordinates transactions on behalf of applications programs
  - *Scheduler* – implements a particular concurrency control, e.g. locking
  - *Recovery manager* – following a failure, restores DB into a consistent state.
  - *Buffer manager* – responsible for efficient transfer of data from main memory to secondary storage

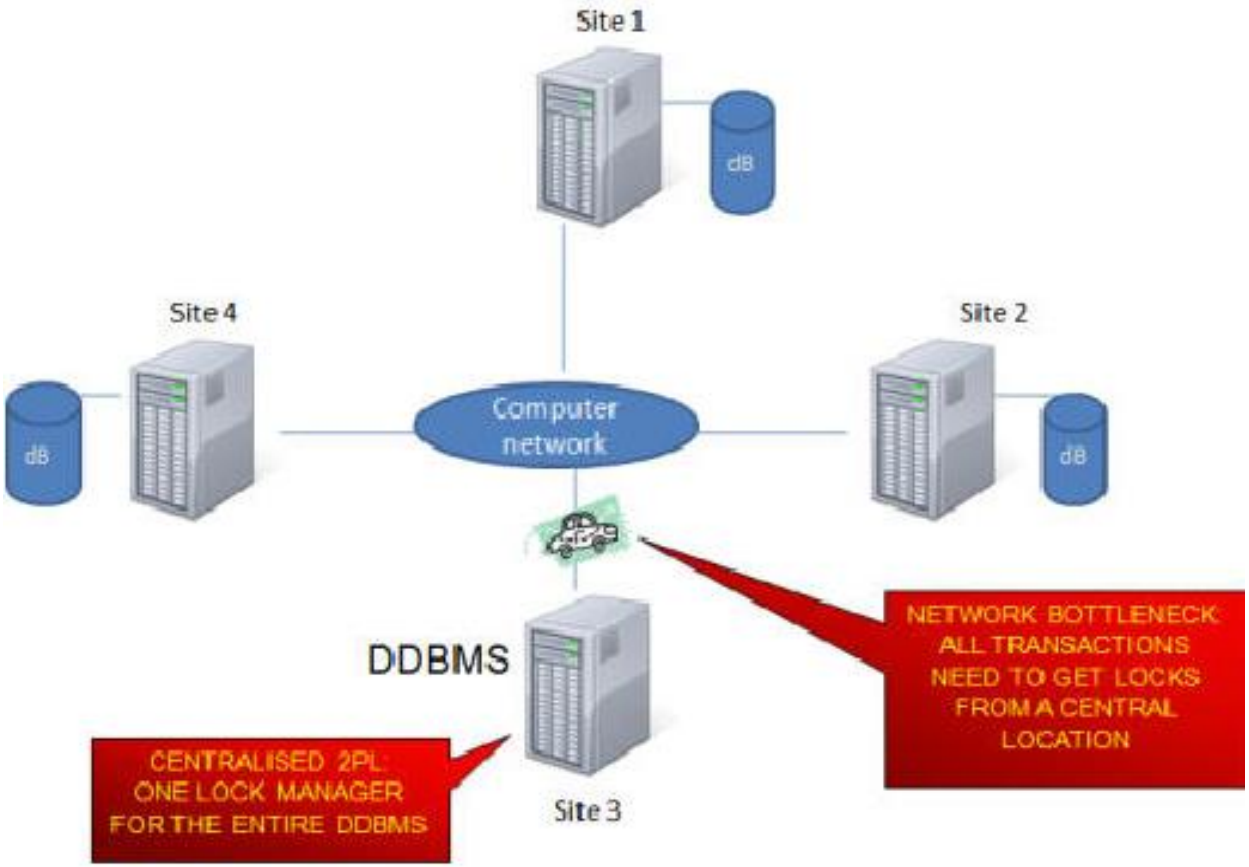
  In DDBMS:
  - **Global** transaction manager/coordinator – coordinates global/local transactions on behalf of applications programs



Distributed Concurrency Control
Distributed Locking

- Data consistency needs to be preserved, in timely manner
- Good CC protocol should:
  - Be resilient to site and communication failures
  - Permit parallelism to satisfy performance needs
  - Incur modest computational and storage overhead
  - Perform satisfactorily in network with significant communication delay
- Multiple-copy inconsistency might arise (if data replicated)!
- Look at four schemes:
  - Centralized Locking.
  - Primary Copy 2PL.
  - Distributed 2PL.
  - Majority Locking.
- Reasoning about these protocols more complex than in centralised systems – parallel transaction executions take place

Centralized Locking



Transaction Management: Centralised 2PL

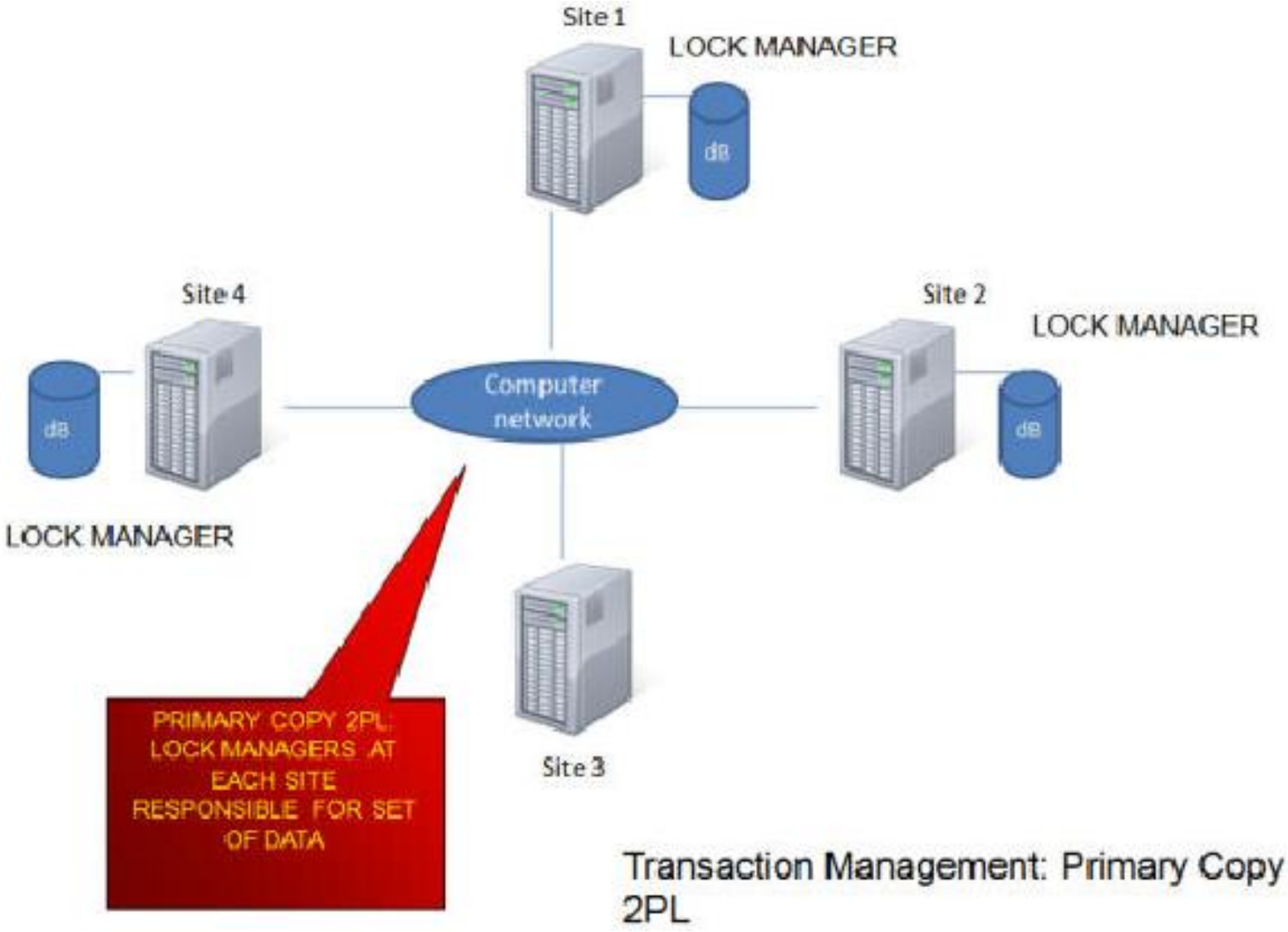
- **Single** site that maintains all locking information.
- ***One* Lock manager** for the whole Distributed DBMS (DDBMS).
- Local transaction managers involved in a global transaction: request and release locks from the Lock manager.
- Or transaction coordinator can make all locking requests on behalf of local transaction managers.

- Advantage – easy to implement.
- Disadvantages – network bottlenecks: all transactions request locks from the central location; and worse reliability: single point of failure.

Primary Copy 2PL

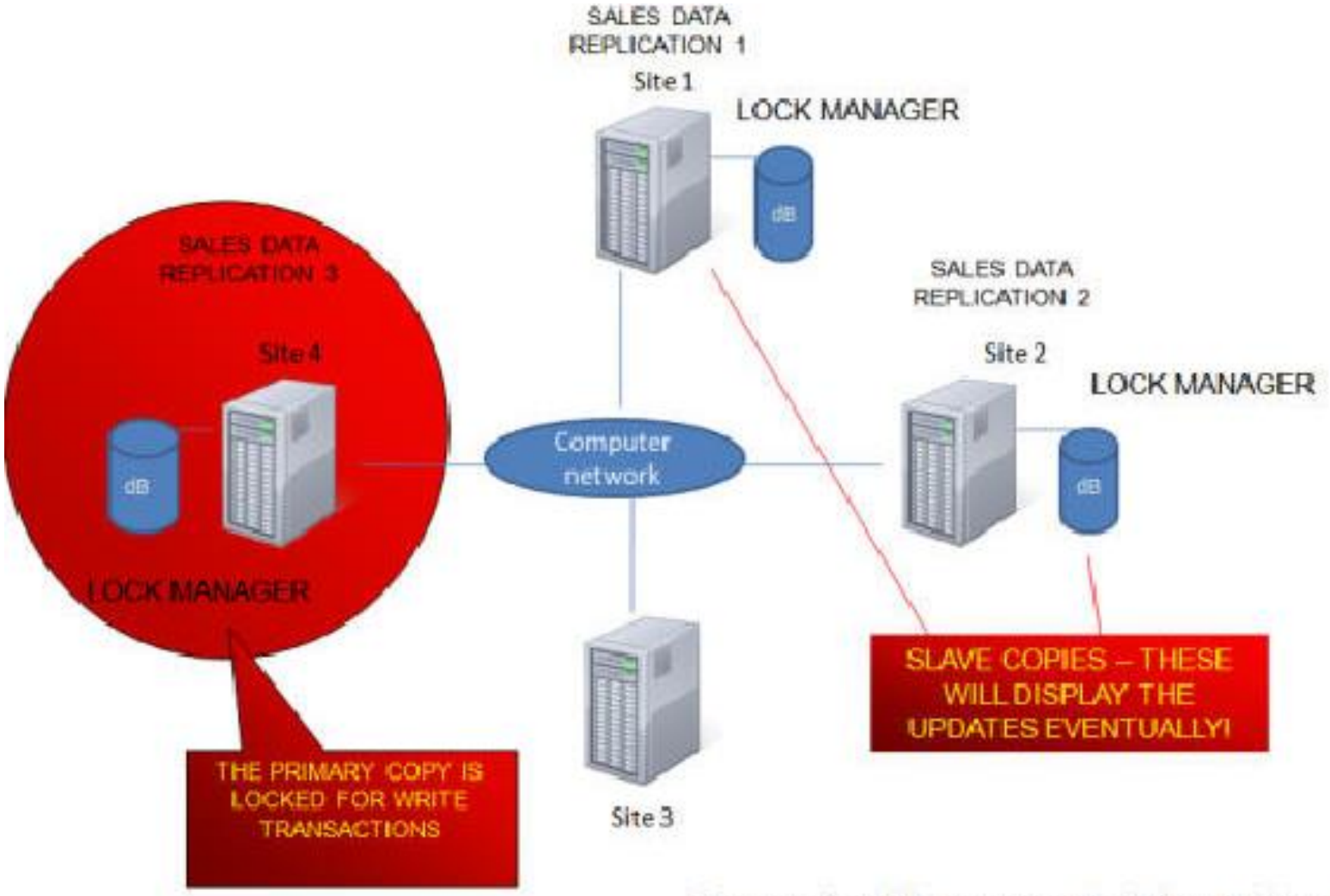
- Lock managers distributed to different sites.
- Each lock manager responsible for managing locks for *a subset of data items*.
- If data item *replicated*, one copy is chosen as the *primary copy*, others are *slave copies*
- Only need to write-lock the primary copy of data item that is to be updated.
- *Once* primary copy has been updated, change can be propagated to slaves.
  - But, *only* primary copy site is guaranteed to hold the current values!
- Advantages – *lower* communication costs and better performance than centralized 2PL.
- Disadvantages –
  - Deadlock handling is more complex, since there's a degree of de-centralization in system (primary copies of data items reside on different sites).
  - Slave copies updated *eventually*.

Primary Copy 2PL (cont.)



Transaction Management: Primary Copy 2PL

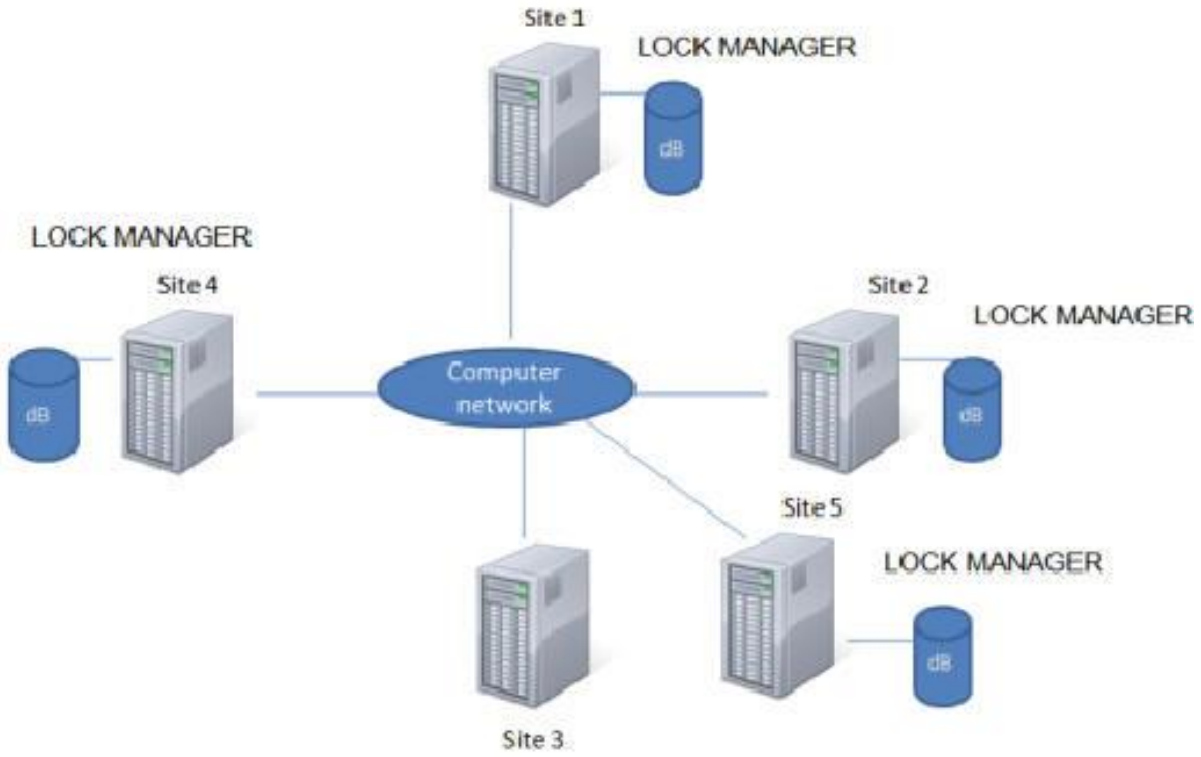
Primary Copy 2PL (cont.)



Transaction Management: Primary Copy 2PL

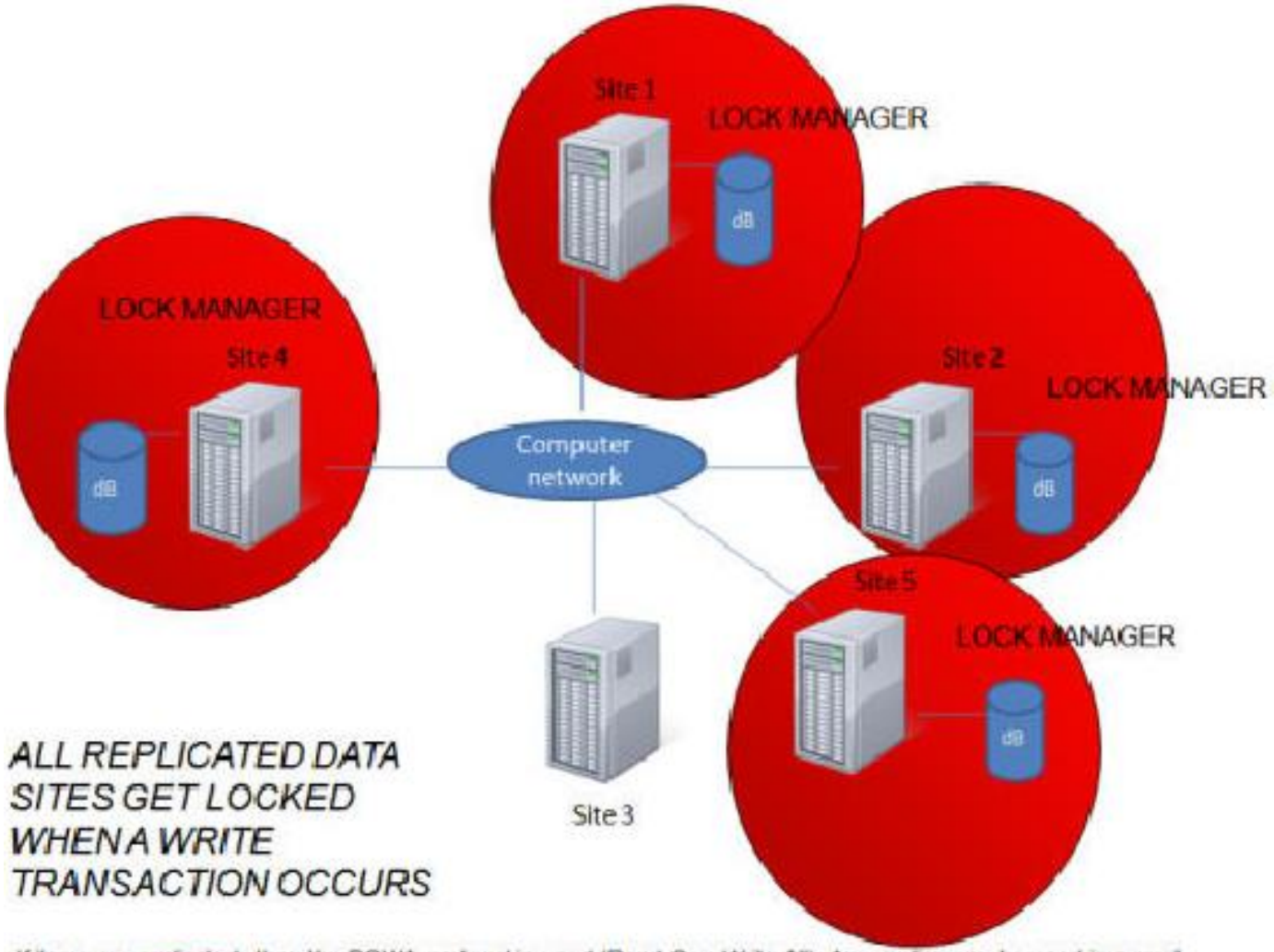
Distributed 2PL

- Lock managers distributed to every site.
- Each lock manager responsible for locks for data at that site.
- If data *not* replicated, equivalent to primary copy 2PL.
- Otherwise, implements ROWA(A) replica control protocol.
  - ROWA(A) stands for Read-One-Write-All-(Available)



Transaction Management: Distributed 2PL

Distributed 2PL (cont.)



If items are replicated, then the ROWA protocol is used (Read One Write All). Any replica can be used in a read operation BUT all copies must be locked for a write operation

Distributed 2PL (cont.)

- Using ROWA(A) protocol:
  - **Any** copy (if not failed) of the replicated item can be used for a read.
  - **All** copies must be “write-locked” before the replicated item can be written.
- Disadvantages
  - Deadlock handling more complex, since there are multiple lock managers
  - Communication costs higher than primary copy 2PL, since all (replicated) items must be locked before update

Majority Locking

- Extension of Distributed 2PL, and ROWA(A).
- To read or write a data item replicated at *n* sites, send a lock request to more than half of the *n* sites where the item is stored.
- Transaction cannot proceed until majority of locks obtained.
- Overly strong in case of read locks, thus:
  - Any number of transactions can simultaneously hold a shared lock on a *majority* of the copies
  - But ONLY 1 transaction can simultaneously hold an exclusive lock on a *majority* of the copies

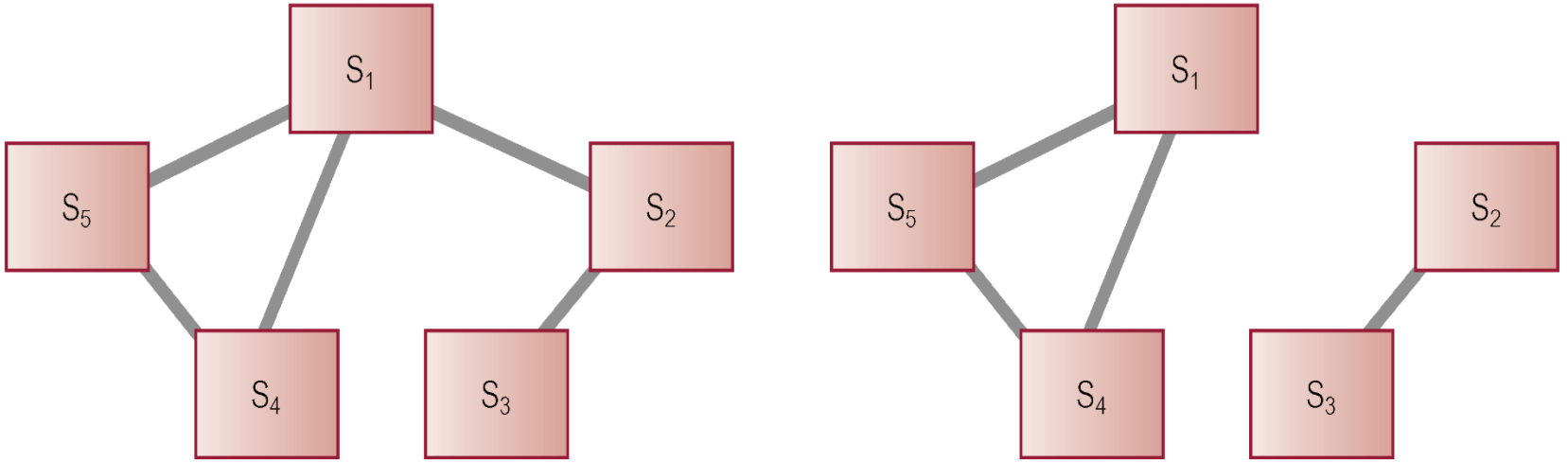
- Related to this are *quorum-based* database replication techniques
  - [*we don't cover those in the module*]

Distributed Serializability

- *Serializability* concept can be extended into distributed environment
- If the schedule of transactions executions at each site is *serlializable*, and these schedules are equivalent, then the **global schedule** is also serializable
  - All (conflicting) subtransactions appear in the **same order** in the **equivalent serial schedule** at all sites (S - site)
  - $T_i^x < T_j^x$ – for all S_x at which T_i and T_j have subtransactions
    - "<" – "*occurs before*" relationship
- Solutions based on the same two approaches as in centralized systems:
  - *Locking*: Concurrent global execution equivalent to some (unpredictable) serial schedule
    - Need to deal with deadlocks – challenging in a distributed system!
  - *Time-stamping*: Concurrent global execution equivalent to a specific serial schedule (corresponding to order of transaction timestamps)
  - If data are **not** replicated, "centralized" versions of the above approaches can be used
    - Need to extend the protocols otherwise!
  - [*We cover only lock-based approaches for Distributed DBMSs*]

Distributed Recovery Control

- DDBMS is highly dependent on ability of all sites to be able to communicate reliably with one another.
- Communication failures can result in network becoming split into two or more partitions.
- May be difficult to distinguish whether communication link or site has failed.
- Examples of a network partitioning



Two-Phase Commit (2PC) Protocol

- 2PC used to guarantee *atomicity* of global transactions
  - A part of the family of *atomic commit* protocols
- A global transaction cannot commit or abort until all its sub-transactions have committed or aborted
- The protocol takes into consideration both *site* and *communication* failures
  - Site failures assumed are *crash* failures; design/software bugs are not taken into account – these are not self-evident!
- For every global transaction there is one site that acts as a **co-ordinator** (transaction manager) - the site where the transaction was initiated, and **participants** sites (resource managers) - the sites accessed by the transaction.
- The co-ordinator knows about all participants, and each participant knows about the co-ordinator

Two-Phase Commit (2PC) Protocol (cont.)

- Two phases: a *voting phase* and a *decision phase*.
- Coordinator asks all participants *whether they are prepared to commit* transaction.
  - If one participant votes abort, or fails to respond within a timeout period, coordinator instructs all participants to abort transaction.
  - If all vote commit, coordinator instructs all participants to commit.
- All participants *must adopt the global decision*.
- If a participant votes abort, *free to abort* transaction immediately
- When a participant votes commit, it *must wait* for coordinator to broadcast global-commit or global-abort message.
- Protocol assumes *each site has its own local log* and can rollback or commit transaction reliably.
- If a participant fails to vote, *abort is assumed*.

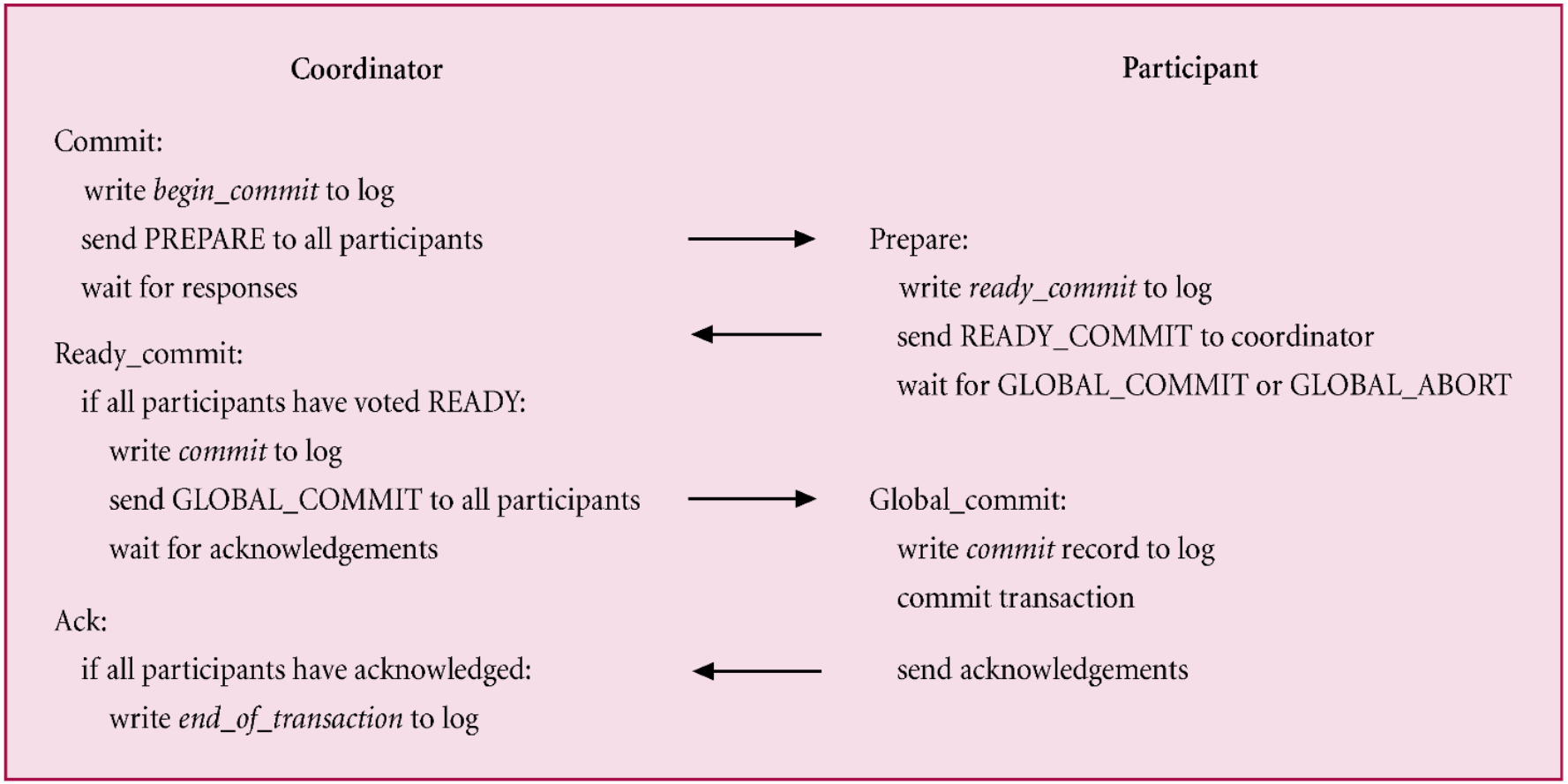
- https://www.coursera.org/lecture/cloud-computing-2/2-2-two-phase-commit-5hKqB
- https://www.coursera.org/lecture/data-manipulation/two-phase-commit-and-consensus-protocols-mXS0H
  – Beware these videos include material we don't cover it in the module too, e.g. Consensus/Paxos etc.

Similarity of 2PC to the concept of marriage

- Decision of two parties is received, and registered by a third party - administrator, who ratifies the marriage
- In order for the marriage to take place, both participants need to confirm they wish to marry
- During the first phase, the administrator receives from the two parties separately the respective confirmation of the desire to marry
- During the second phase, the administrator confirms/serves notice that the marriage has taken place

- Analogy breaks because e.g. marriage is between 2 people, while 2PC coordinates *n* sites; the administrator is not taking part in marriage, while the coordinator *might be* a part of the DDBMS, etc.

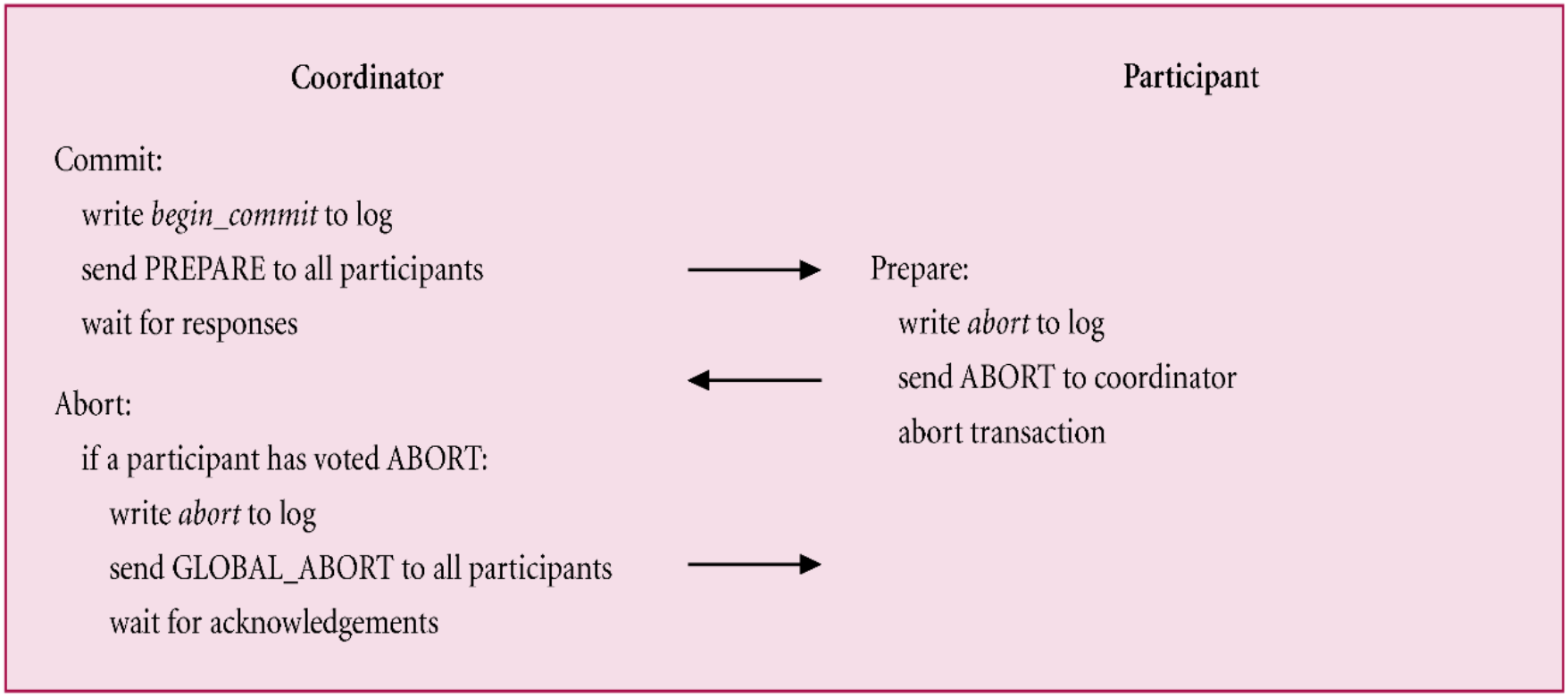
2PC Protocol for Participant Voting Commit



(a)

- Note that writing to log precedes communication messages. Similarly, as part of centralised DB recovery (Lect. 5), the recovery log, e.g. WAL, is first written to.

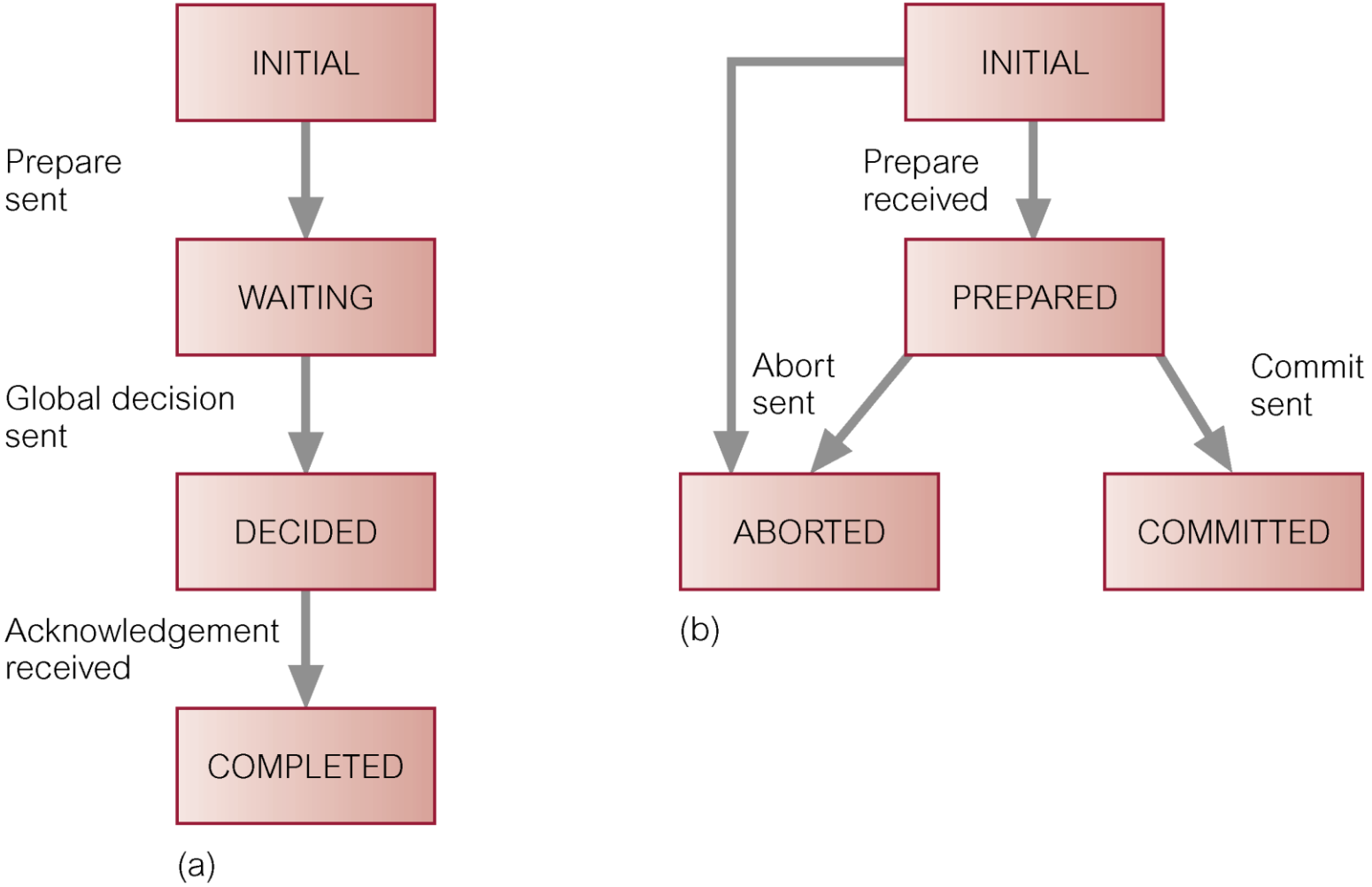
2PC Protocol for Participant Voting Abort



(b)

- This slide does **not** show the case when a participant receives GLOBAL_ABORT message, **and** is **not** the participant who initiated the abort.

State Transition Diagram for 2PC



(a) coordinator; (b) participant

2PC Termination Protocols

- Invoked whenever the *coordinator* or a *participant* fails to receive an expected message and *times out*.
- Coordinator
  - Timeout in WAITING state
    - Globally abort transaction.
  - Timeout in DECIDED state
    - Send global decision again to sites that have not acknowledged.
- Participant
  - Simplest termination protocol is to leave participant blocked until communication with the coordinator is re-established. Alternatively:
  - Timeout in INITIAL state
    - Unilaterally abort transaction.
  - Timeout in PREPARED state
    - Without more information, participant blocked.
    - Could get decision from another participant
      - See later *Cooperative* termination protocol

2PC Recovery Protocols – Coordinator Failure

- Action to be taken by a failed site upon *recovery*. Depends on what stage the *coordinator* or *participant(s)* had reached.

- Coordinator Failure
  - Failure in INITIAL state
    - Recall: the Coordinator has not yet started the commit procedure
    - Recovery starts commit procedure.
  - Failure in WAITING state
    - Recovery restarts commit procedure.
    - The coordinator has not received all responses, but it did not receive an Abort vote either.
      - *How do we know?* If Coordinator received an abort vote, it'd would have progressed to DECIDED state since the only decision in that case can be *Global_abort*.
  - Failure in DECIDED state
    - Recall: The Coordinator has instructed all participants about a Global decision.
    - On restart, if coordinator has received all acknowledgements, it can complete successfully. Otherwise, has to initiate termination protocol discussed above.

2PC Recovery Protocols – Participant Failure

- Action to be taken by a failed site upon *recovery*. Depends on what stage the *coordinator* or *participant(s)* had reached.

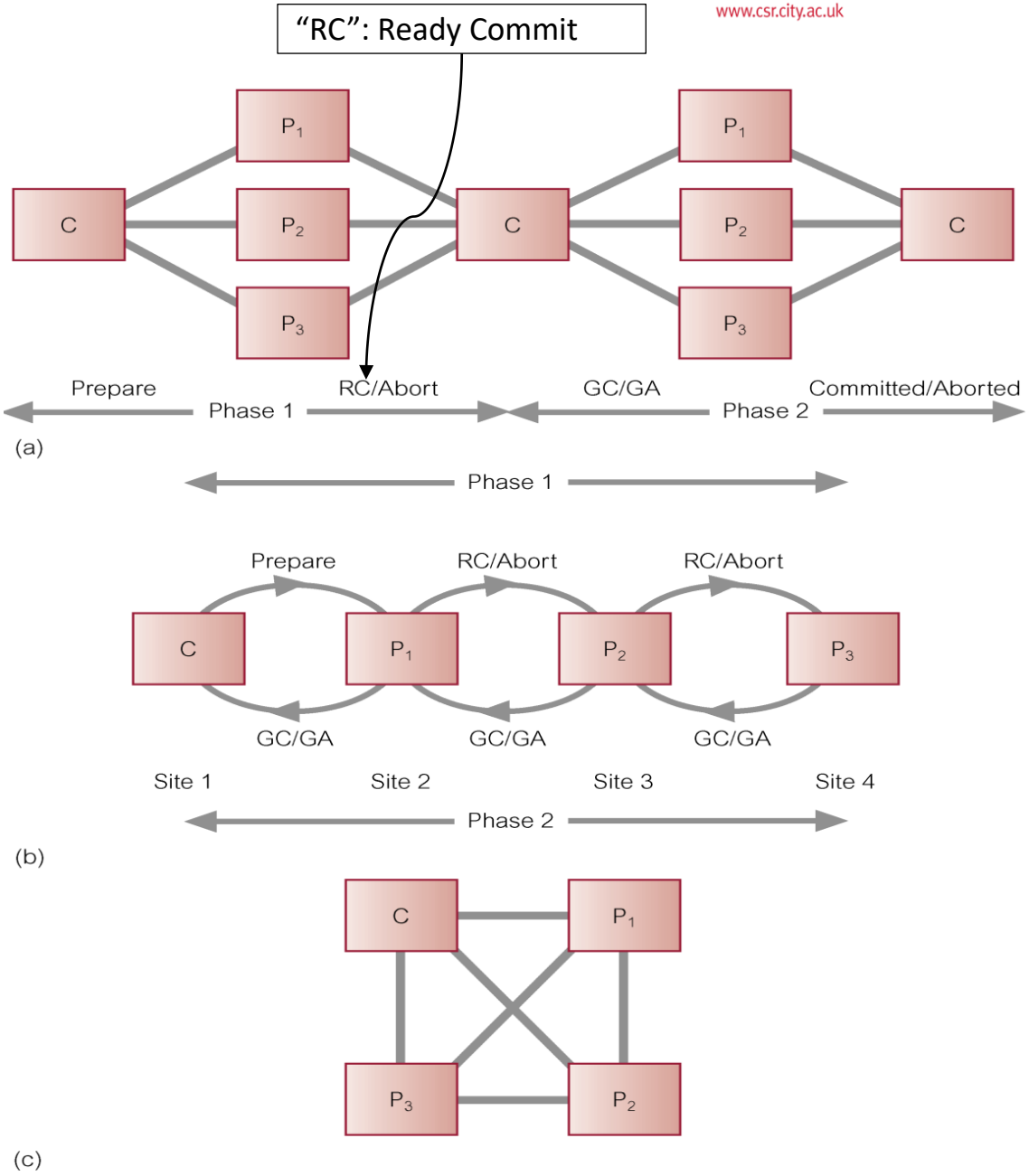
- Participant failure
  - Must ensure that participant on restart performs same action as all other participants, and that this restart can be performed independently.
  - Failure in INITIAL state
    - Upon recovery, unilaterally abort transaction, since it would have been impossible for the Coordinator to commit the transaction without this participant's vote.
  - Failure in PREPARED state
    - Recall: the participant has sent its vote to the Coordinator
    - Recovery via termination protocol above.
  - Failure in ABORTED/COMMITTED states
    - On restart, no further action is necessary, since the participant has completed the transaction.

2PC Topologies

a) Centralised
   - Assumed thus far
   - All comm. funnelled through the Coordinator

b) Linear
   - Fewer messages, but
   - No parallelism

c) Distributed
   - Coordinator sends PREPARE
   - Participants send their decision to **all** sites
   - Eliminates the need for the Decision phase



Blocking nature of 2PC

- A disadvantage of the 2PC is that it is a blocking protocol:
  - after a participant has sent a *ready_to_commit* message to the Coordinator, it will block until a GLOBAL_COMMIT or GLOBAL_ABORT is received.
  - If the Coordinator fails permanently, some participants will never resolve their transactions – participant(s) wait for the GLOBAL_COMMIT or GLOBAL_ABORT message
  - *Cooperative* termination protocols can be used – the blocked participant could contact other participants attempting to find what the global decision is.
    - Likelihood of blocking reduced, but blocking still possible
    - If only the coordinator failed and all participants discover this, they can elect a new coordinator.
    - *Note: Election protocol is a part of 2PC, but we don't cover it!*
      - A simple one is *linear ordering* election protocol

Three-Phase Commit (3PC)
[not covered in detail]

- 2PC is a *blocking* protocol.
- For example, a process that times out after voting commit, but before receiving global instruction, is blocked if it can communicate only with sites that do not know global decision.
- Probability of blocking occurring in practice is sufficiently rare that most existing systems (which rely on an atomic commitment approach) use 2PC.

- An alternative: a non-blocking protocol called *three-phase commit (3PC)* protocol.
- Non-blocking for site failures, except in event of failure of all sites.
- Communication failures can result in different sites reaching different decisions, thereby violating atomicity of global transactions.

Introduction to Database Replication

- Database Replication is the process of copying and maintaining database objects, such as relations, in multiple databases that make up a replicated/distributed database system.

- Every major database vendor has (at least one) replication solution.

Introduction to Database Replication

- A type of Replication can be described using the publishing industry metaphor:
  - *Publisher:* a DBMS that makes data available to other locations through replication, by exposing *publications (*made up of one or many *articles),* each defining a set of objects/data to replicate.
  - *Distributor:* a DBMS that stores replication data and metadata about the publication and in some cases acts as a queue for data moving from the publisher to the subscribers. A DBMS can be both publisher and distributor.
  - *Subscriber:* a DBMS that receives replicated data. A subscriber can receive data from multiple publishers / publications.

- Note: Not all database replication solution conform (fully) to this metaphor.

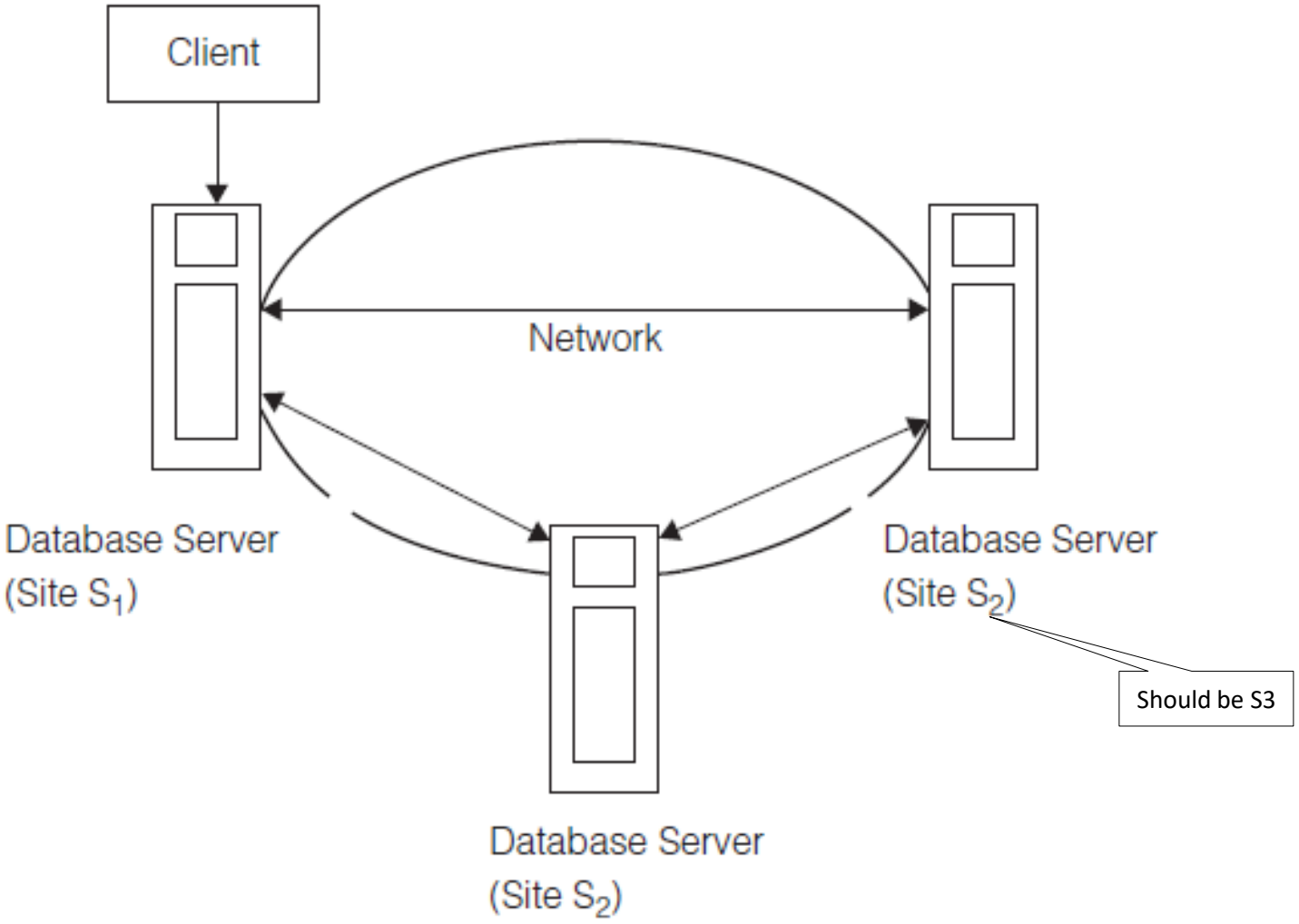
Introduction to Database Replication

- Replication has similar advantages to DDBMS:
  - *Availability (and reliability)*
    - A failure of a site/network link not necessarily makes the whole dataset inaccessible
  - *Improved performance*
    - Read request served locally
    - Better resource balancing than with a centralised system
  - *Supports disconnected computing model*
    - When users become disconnected from their corporate database, but can continue to operate (albeit likely with reduced functionality)
    - Typical for mobile environment

Applications of Replication

- Replication supports a variety of applications that have very different requirements.
- Some applications are supported with only limited synchronization between the copies of the database and the central database system.
  - Remote sales application
  - Consistency synchronization only periodic
  - Autonomy preserved, however
- Other applications demand continuous synchronization between all copies of the database.
  - E.g. financial application for management of shares

Replication Model



Replication Model (cont.)

- Replicated database system consists of several databases, called *replicas* or *copies*.
- At each site there can also exist a backup site and backups are sometimes used interchangeably
  - a backup can also be used in combination with recovery aspects.

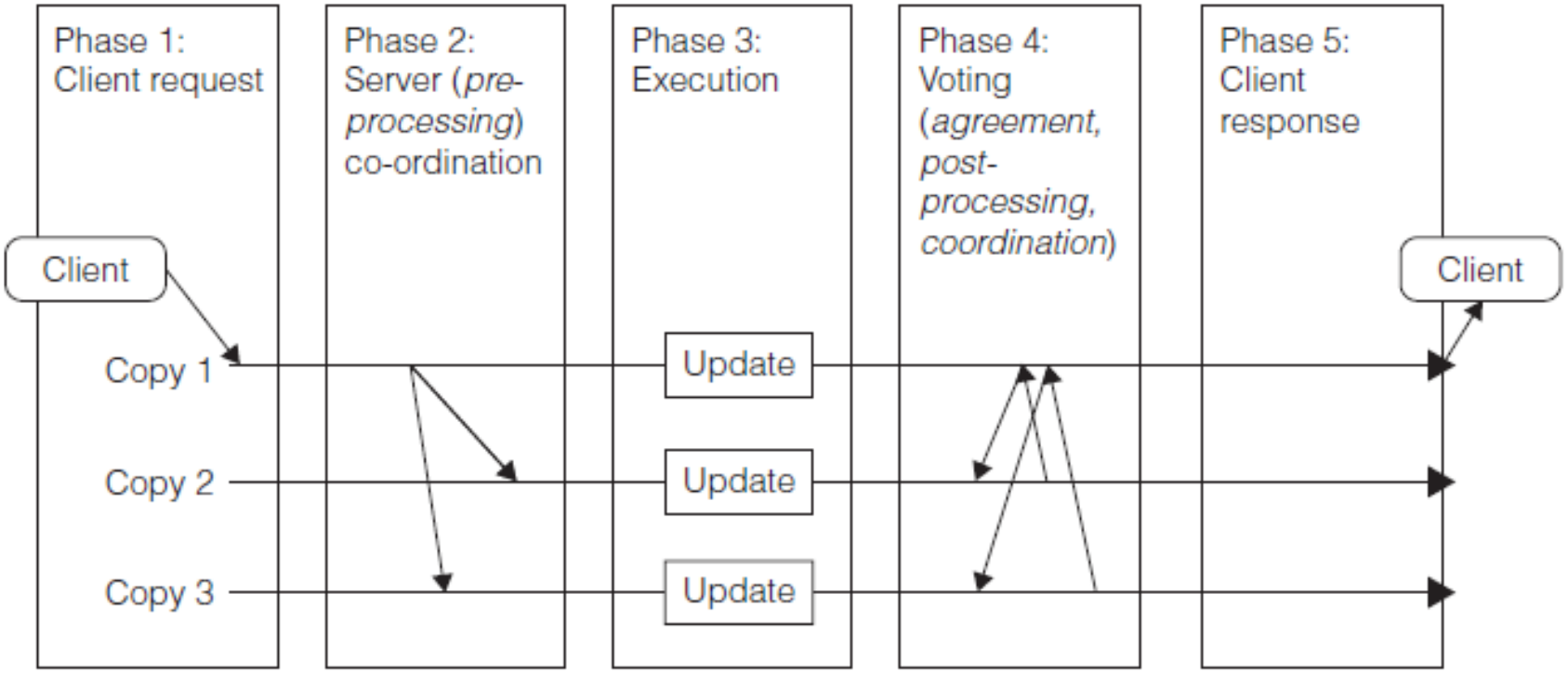
- Replicated database consists of a set of *n* sites S = {S_1, S_2, …, S_n), where *n* >= 2.
- A site hosts a set of copies of data items x_1, x_2, x_3, …; we assume for the remainder of this presentation that each site is a *complete copy* of the database.
  - *Partial replication*, however, is an option.

- To distinguish between physical copies and the logical data item itself, a copy is denoted with the site identifier; e.g. a copy of data item x at site S_1 is denoted as x^1.

Replication Model (cont.)

- Since many transactions might concurrently update copies at different sites, need a consistency criterion to determine whether concurrent execution of transactions accessing copies at different sites is correct.
  - One-copy serializable (**1CSR**) is the main criterion
    - Analogous to Serializable consistency criterion in centralized (non-replicated) DBs
  - A replicated data history is **one-copy serializable (1CSR)** if it is **equivalent** to a serial one-copy history.
    - See slide “Distributed Serializability” above
  - Like in centralized DBMSs, relaxations of this strictest isolation level – 1CSR – exist
    - Many such are based on Snapshot Isolation (SI).
    - In SI read and write requests do NOT conflict (recall that a snapshot is taken when a transaction begins) and thus it’s beneficial for replicated setups too. Still, SI guarantees high level of consistency.
    - NB **1-copy-SI** defined in:
      - Y. Lin, B. Kemme, M. Patino-Martınez, and R. Jimenez-Peris. “*Middleware based data replication providing snapshot isolation”*. ACM SIGMOD Conf., p 419–430, 2005

Functional Model of Replication Protocols



Functional Model of Replication Protocols (cont.)

- **Phase 1:** A client submits its request to one site, called the local site
  - *Active replication* also possible: requests sent to all sites.
- **Phase 2:** Depending on replication scheme, requests are forwarded to the other sites, called the remote sites.
- **Phase 3:** The request is processed.
- **Phase 4:** After all affected sites have processed request, sites communicate again, e.g. to detect inconsistencies, propagate modifications, aggregate results, form a quorum or ensure atomicity of distributed transaction by running a concurrency (i.e. replica) control protocol, such as 2PC.
- **Phase 5:** Result is sent to the client.

Consistency

- Transaction in a replicated database is an ACID unit of work, although, different definitions of consistency exist.
- Strongest form of consistency, 1CSR, degrades performance of a replicated database.
- It has been suggested that a distributed system can only choose two out of the properties: *Consistency*, *Availability*, and *Partition Tolerance* - CAP theorem (Brewer's theorem):
  - In general case, in a distributed system (a collection of interconnected nodes that share data), you can only have 2 out of the following 3 guarantees at a given time across a write/read pair: Consistency, Availability, and Tolerance to Network Partitioning - one of them must be sacrificed
  - More about this in the NoSQL lecture
  - For additional, detailed info on the CAP theorem see e.g.
    - https://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
    - Brewer, E. A. (2000). *Towards robust distributed systems* (abstract). Proceedings of the nineteenth annual ACM Symposium on Principles of distributed computing. Portland, Oregon, USA, ACM**:** 7. https://dl.acm.org/doi/10.1145/343477.343502
    - http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

Consistency Types

- *Strong and Weak Consistency*:
  - Strong – all copies of a data item have same value at end of *update* transaction.
  - Weak (eventual) consistency – values eventually become identical and there is some time where replicas might have different values.

- *Transaction and Mutual Consistency*:
  - Mutual – copies converge to the same value
  - Transaction – global execution history is 1CSR.
  - A system can be mutually consistent but not transactionally consistent, although the opposite is not true.

Consistency Types

- *Session Consistency*, a type of weak consistency:
- A basic property for each replication technique.
  - Guarantees that a client observes its own updates
  - If clients do not observe their own updates a serious race condition arises. A *race condition* is where a transaction writes data item x on S_1 and a subsequent read of x within the same transaction on site S_2 does not reflect the write.
    - Such a condition can exist even *across* sessions - *read-your-own-writes*
      - E.g. consider that a user updates the password on site S_1, logs out and immediately logs in again. Then, a new transaction is started to verify the password, but if this transaction is executed on S_2 and S_2 is not **yet** aware of the new password, an error would occur, and the user might assume the password had not yet been updated.

Required Reading

- Connolly,T. and Begg, C. "*Database Systems - A Practical Approach to Design, Implementation, and Management*.", 6th Ed., Pearson Education Ltd
  - Chapter 25: Sections: 25.1, 25.2, 25.4.1, 25.4.2, 25.4.3
    - Note: 25.4.4 ("3-Phase Commit") - not covered in detail, but need to know of its existence and main characteristics; see the earlier slide.
  - Section 26.1

- This list is only indicative, and the text in these sections should be of course looked at as "additional", but required, reading to the lecture slides.
- This is certainly not an exhaustive list.
- The content of the chapters referred to might NOT be fully relevant to the module. You need to make the final decision about what is relevant and what is not yourselves, given the material that we covered in the lecture.