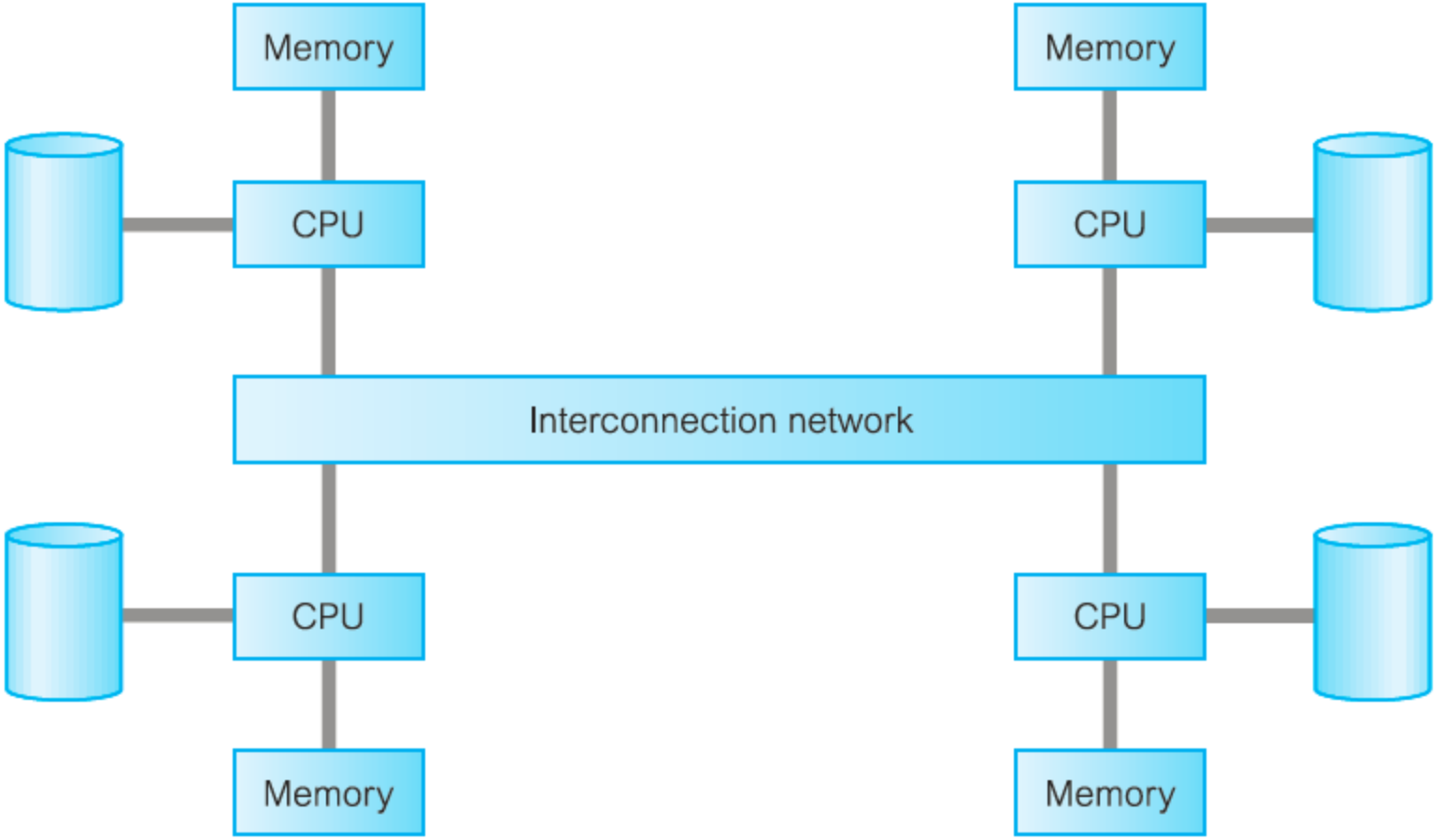
Fragmentation

Distributed Transaction Management (Concurrency & Recovery)

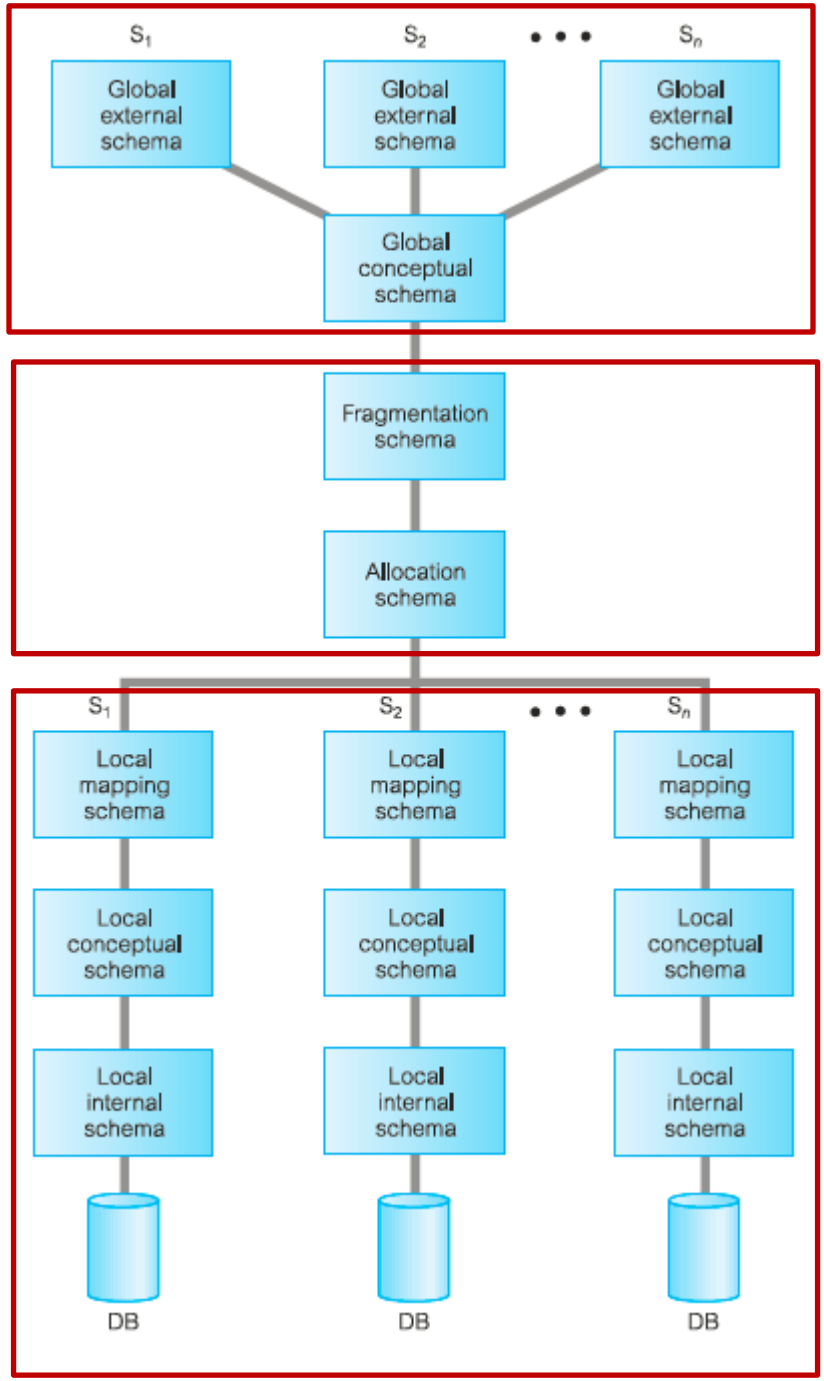
CSC 321/621 – 4/10/2012

Database Architectures: Shared Nothing



Multiple independent computers connected via a network
Data distributed over disks
Standard model for distributed database, model for some parallel databases

Distributed Databases: Architecture



Distributed Databases: Architecture

- Global Schemas:
  - Logical description of the entire database, as if not distributed
    - End-user view of the system
  - Entities, relationships, constraints, security, integrity information
- Fragmentation/allocation schemas:
  - Fragmentation: how the data is to be partitioned (broken up)
  - Allocation: where the data resides, including any replication
- Local schemas:
  - Conceptual/internal schemas: Local databases data representation
  - Mapping schemas: Maps fragments in the allocation schema into objects in the database that have been “exposed”

Fragments

- A fragment is a decomposition of a relation (so at the table level)
  - *Horizontal (subsets of tuples)*
  - *Vertical (subsets of attributes)*
  - *Mixed*
  - *Derived (foreign key based)*
- Fragments are distributed across local databases

Fragment Distribution

- In determining where to allocate and replicate the fragments, need to evaluate transactions:
  - Use qualitative information to decide how to split
    - Relations, attributes, tuples accessed in transactions
    - Types of access (read/write)
    - Predicates employed in SELECTs
  - Use quantitative information to determine where to allocate
    - Frequency of use
    - Organizational patterns of use (who uses it)
    - Performance criteria

Fragment Distribution: Goals

- Goals for good distribution:
  - Locality of reference
    - Put data close to where most heavily used
    - Replicate if multiple heavy users
  - Improved reliability and availability
    - Increase replication when possible to handle failures
  - Balanced storage capacities and costs
    - Exploit cheap mass storage where possible, balance against locality of reference
  - Minimal communication costs
    - Find sweet spot to lower communication costs due to replication (having local copies) while not being so replicated that inundated with updates to keep replicates up to date

Allocation Strategies

Let's examine merit of four different allocation strategies (from one extreme to another)

Centralized: All data in one place

Fragmented: Data spread over network, targeting data to sites use most; no replication

Selective replication: Data spread over network, targeting data to sites use most; replicate frequently used but not commonly updated data; centralize rest

Complete replication: Maintain a complete copy of database at every site

Allocation Strategies

	Locality of reference	Reliability and availability	Performance	Storage costs	Communication costs
Centralized	Lowest	Lowest	Unsatisfactory	Lowest	Highest
Fragmented	High[a]	Low for item; high for system	Satisfactory[a]	Lowest	Low[a]
Complete replication	Highest	Highest	Best for read	Highest	High for update; low for read
Selective replication	High[a]	Low for item; high for system	Satisfactory[a]	Average	Low[a]

[a] Indicates subject to good design.

Fragmentation Questions

- Assume you have the following student table:

Student(StdNo, StdName, StdCampus, StdCity, StdState, StdZip, StdMajor, StdYear)

- Suggest why fragmentation on StdName is probably not a good idea

- For WFU, suggest the fragment definition statements needed to properly horizontally fragment using the StdCampus relation.

Distributed Transaction Types

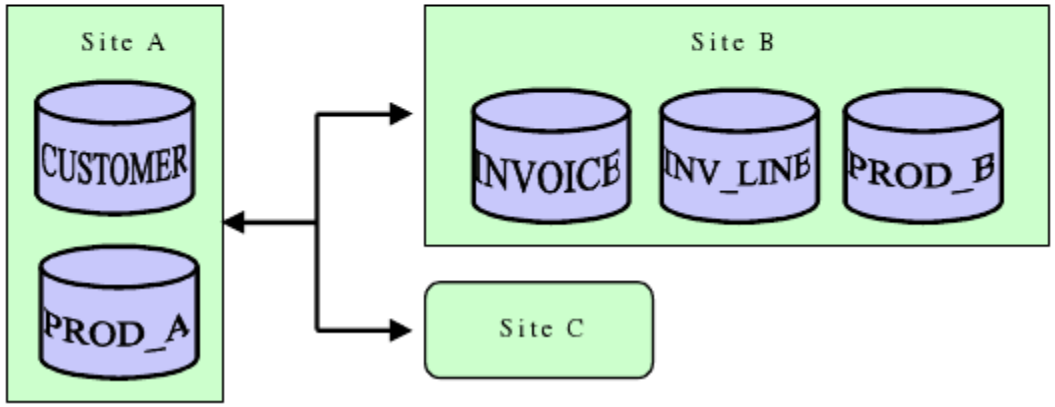
- Transaction classifications, delineated by type:
  - Remote request
    - A (local) site can send a single SQL statement to another (remote) site for execution, with the referenced data and processing only occurring at that site
  - Remote transaction
    - A (local) site can send *multiple SQL statements within a transaction* to another (remote) site for execution, with the referenced data and processing only occurring at that site. The local site can make the final commit/abort decision.
  - Distributed transaction
    - A (local) site can send multiple SQL statement within a transaction to *multiple other (remote) sites for execution*, with the referenced data and processing only occurring at each site proper. The local site can make the final commit/abort decision.
  - Distributed request
    - A (local) site can send multiple SQL statement within a transaction to multiple other (remote) sites for execution. *Data can be accessed from different sites.* The local site can make the final commit/abort decision. The local site can make the final commit/abort decision.

Fragmentation Questions

- Assume you have the following fragmentation and allocation of relations.

Figure P10.1 The DDBMS Scenario for Problem 1

TABLES	FRAGMENTS	LOCATION
CUSTOMER	N/A	A
PRODUCT	PROD_A	A
	PROD_B	B
INVOICE	N/A	B
INV_LINE	N/A	B



What type of transaction is invoked by "SELECT * FROM CUSTOMER;"?

Remote Request

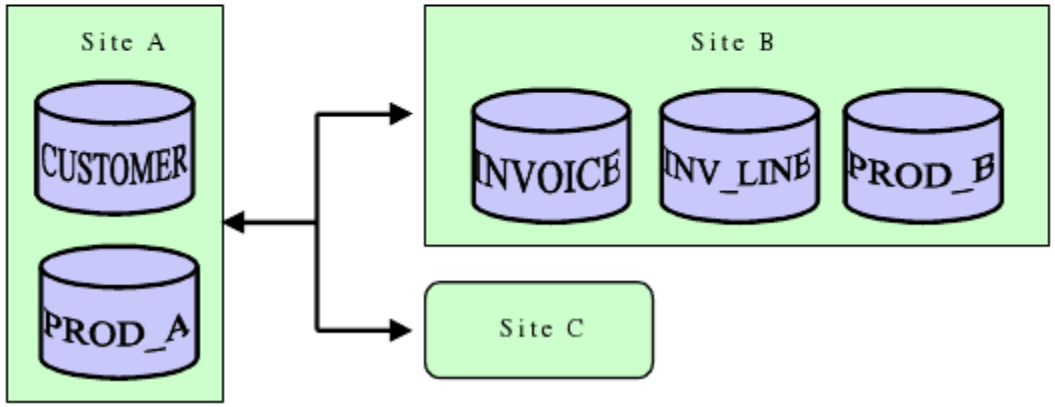
Fragmentation Questions

- Assume you have the following fragmentation and allocation of relations.

Figure P10.1 The DDBMS Scenario for Problem 1

TABLES	FRAGMENTS	LOCATION
CUSTOMER	N/A	A
PRODUCT	PROD_A	A
	PROD_B	B
INVOICE	N/A	B
INV_LINE	N/A	B

What type of transaction is required for "SELECT * FROM PRODUCT WHERE QuantityOn Hand > 10;"

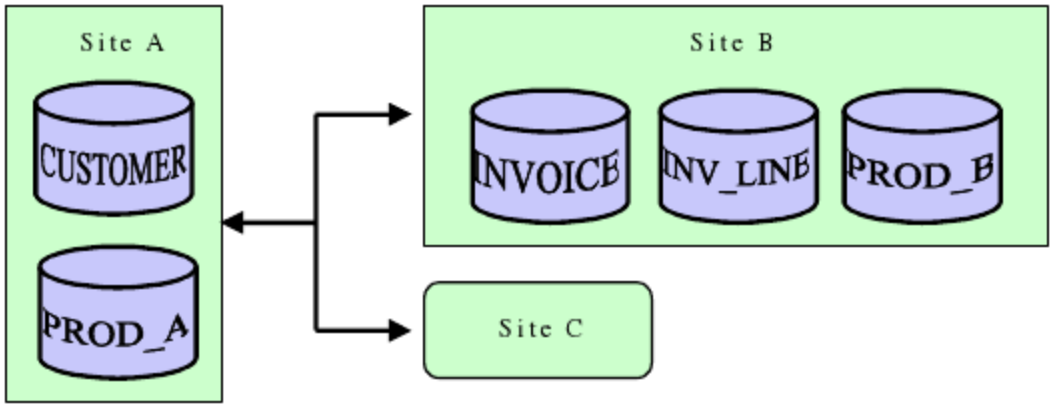


Distributed Request

Fragmentation Questions

Figure P10.1 The DDBMS Scenario for Problem 1

TABLES	FRAGMENTS	LOCATION
CUSTOMER	N/A	A
PRODUCT	PROD_A	A
	PROD_B	B
INVOICE	N/A	B
INV_LINE	N/A	B



Distributed Transaction

```
BEGIN WORK;
INSERT CUSTOMER(CUS_NUM, CUS_NAME, CUS_ADDRESS CUS_BALANCE)
      VALUES ('34210','Victor Ephanor', '123 Main St', 0.00);
INSERT INTO INVOICE(INV_NUM, CUS_NUM, INV_DATE, INV_TOTAL)
      VALUES ('986434', '34210', '10-AUG-1999', 2.00);
COMMIT WORK;
```

Distributed Database Transparency

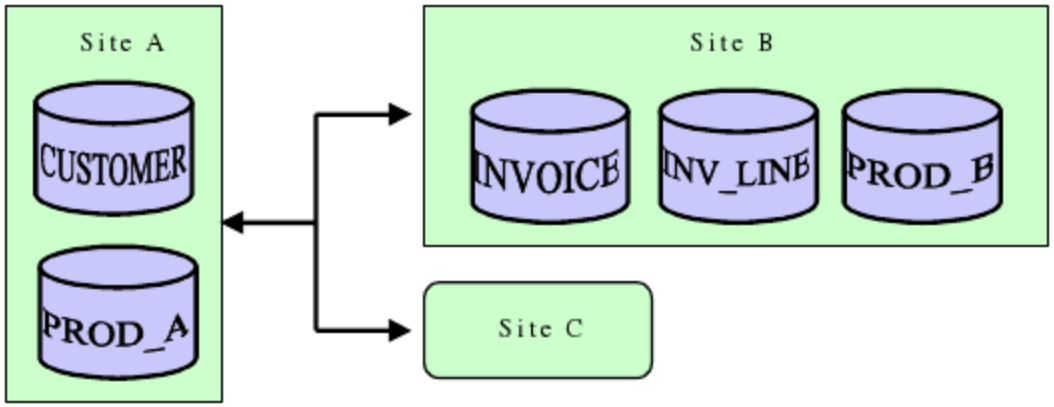
- Should the end-user (employing SQL) be aware of the distribution of data?
  - Fragmentation transparency:
    - The end user is unaware that the data is even distributed; they can access data as if one single database
  - Location transparency:
    - The end user is aware of fragmentation, but is unaware of where the fragments are located; requires queries on fragments, joins & unions to re-merge data as needed
  - Local mapping transparency:
    - Not really transparency at all; user must write queries based on fragments and locations

Fragmentation Questions

Figure P10.1 The DDBMS Scenario for Problem 1

TABLES	FRAGMENTS	LOCATION
CUSTOMER	N/A	A
PRODUCT	PROD_A	A
	PROD_B	B
INVOICE	N/A	B
INV_LINE	N/A	B

If the DBMS maintaining these relations only supports location transparency, queries on which tables will be different from if the database was a standardized (centralized) database?



Product table

What if the DBMS supported fragment transparency?
What if the DBMS supported local mapping transparency?

Distributed Database Transparency

- With locally autonomous sites, how does one ensure that different sites don't create data objects with the same name? (naming transparency)
  - Centralized name server
    - Can lead to availability and performance issues
  - Forced naming conventions
    - As one example, name is a combination of site name, relation name, fragment name, and copy (replicate name): S1.Branch.F3.C2

Distributed Database Transparency

- A transaction that requires access to multiple sites must maintain the distributed database's integrity and consistency
- Can think of a distributed transaction as a set of sub-transactions

- Assume we had a transaction whose job was to print all staff members' names from the realty company database, given the fragmentation we saw previously

Distributed Database Transparency

Fragment S_1

staffNo	position	sex	DOB	salary
SL21	Manager	M	1-Oct-45	30000
SG37	Assistant	F	10-Nov-60	12000
SG14	Supervisor	M	24-Mar-58	18000
SA9	Assistant	F	19-Feb-70	9000
SG5	Manager	F	3-Jun-40	24000
SL41	Assistant	F	13-Jun-65	9000

Fragment S_{21}

staffNo	fName	lName	branchNo
SG37	Ann	Beech	B003
SG14	David	Ford	B003
SG5	Susan	Brand	B003

Fragment S_{22}

staffNo	fName	lName	branchNo
SL21	John	White	B005
SL41	Julie	Lee	B005

Fragment S_{23}

staffNo	fName	lName	branchNo
SA9	Mary	Howe	B007

- Assume that the fragments are distributed over three sites
- There would be three sub-transactions
  - They could run in parallel with each other
  - But, must synchronize with:
    - Local transactions at a given site
    - Other distributed global transactions

Distributed Database Transparency

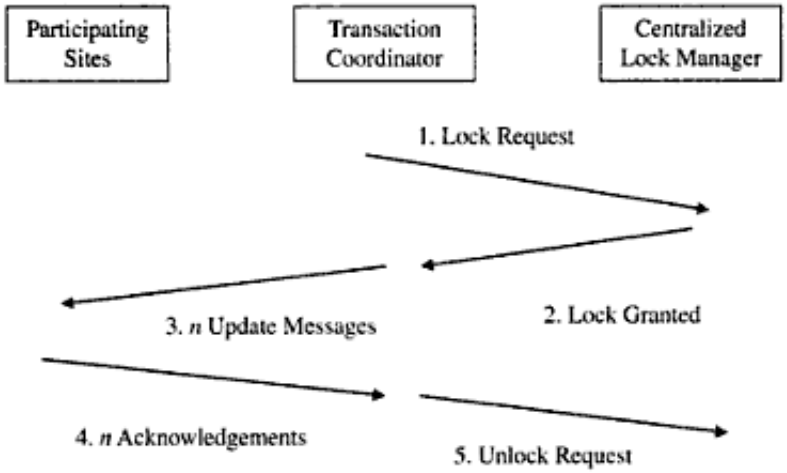
- Concurrency transparency:
  - The results of all independently executed concurrent transactions (distributed & not) are logically consistent with results obtained if executed in some arbitrary serial order
- Failure transparency:
  - Distributed database must ensure transactions are atomic and durable, even when facing new failure possibilities of:
    - Loss of a network message
    - Failure of a network connection
    - Failure of a site
  - Commits at a global level for a transaction must be synchronized with commits at the local sites for the associated sub transactions
  - We can safely assume each site maintains its own *log*

Concurrency Transparency

- The same concurrency problems as before must now be handled within a distributed database
- Also have to deal with replication
  - Assume replicates of a fragment X are updated as part of transaction affecting X
- Need to expand 2-phase locking:
  - A few variations:
    - Centralized 2PL
    - Distributed 2PL
    - Primary Copy 2PL

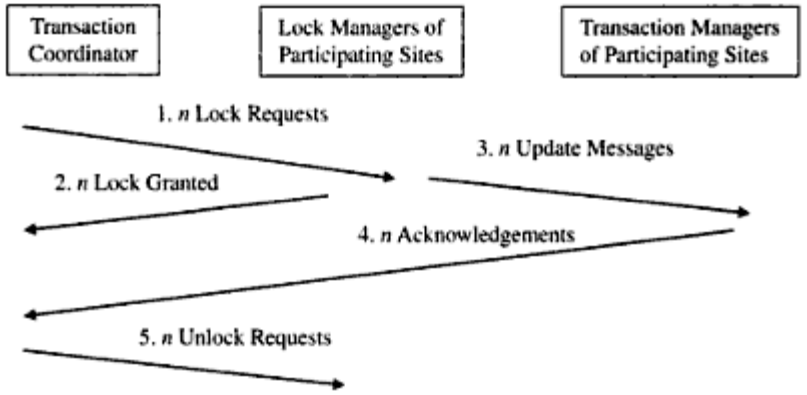
Concurrency Transparency

- Centralized 2PL:
  - Single lock manager at a central site
  - Transaction coordinator at site where transaction started is responsible for locking all required fragments and updating local transaction manager
    - Read locking one copy/fragment
    - Write locking all replicates of fragments so replicates can be updated
  - Easy to manage (lock access, deadlock detection) since data all in one place
  - Low communication overhead: 2n+3 messages where n is number of sites dealing with data
    - 1 lock request, 1 lock grant, n updates messages, n acknowledgements, 1 unlock request
  - Susceptible to bottlenecks and loss of centralized server



Concurrency Transparency

- Distributed 2PL:
  - Have a lock manager at *each* site managing data at that site
  - Without replicates, equivalent to Primary Copy 2PL
  - With replicates, ROWA (read one write all)
    - Allow any copy to be used for reads
    - All copies must be locked for writes
  - High communication costs: 5 * number of sites holding data (each site requires lock request, lock grant, update, acknowledgement, unlock)



Concurrency Transparency

- Primary Copy 2PL:
  - Does not support replicate updates in an atomic fashion
  - Have a set of lock managers, each responsible for managing locks for sets of data – not necessarily at each site
  - Replicates labeled as either primary copy or slave copy
  - Only lock primary copy of data, by requesting appropriate lock manager
  - In theory, should propagate changes to primary copy to replicates ASAP, but not required (no guarantee made) [in theory, everyone should mainly be reading primary copy!]
  - Primarily used when data not significantly replicated, updates are infrequent, and old views of data are OK
  - Tends to be more efficient (in communication, work) because of not having to deal with replicates and lock all copies

Failure Transparency

- Distributed database must ensure transactions are atomic and durable, even when facing new failure possibilities of:
  - Loss of a network message
  - Failure of a network connection
  - Failure of a site
- We will assume that the underlying network protocols resolve issue 1 for us

Failure Transparency

- The DDBMS must ensure that all sub-transactions commit or all rollback

- If a site fails or becomes unavailable, the following need to happen:
  - Globally, transaction manager should:
    - Abort transactions affected by the failure
    - Mark the site as down to prevent further transaction usage
    - Check for updates on restart of failed site
  - Locally, upon restart, failed site:
    - Initiates recovery to abort any partial transactions active at time of failure
    - Update local copy of database to be consistent with rest of system

Let's deal with getting everyone on the same page (all commit or all abort) first.

Failure Transparency: 2-Phase Commit

- Actors:
  - Assume *transaction coordinator* resides at site that initiated transaction
  - *Participants* are responsible for work done at distributed sites
- Need a protocol to support:
  - Determining if any sub-transaction needs to abort
  - Involves *voting* stage and *decision* stage

2PC: Stages

- Voting stage:
  - Coordinator asks all participants if ready to commit
  - If anyone votes “ABORT” or never responds (up to some time period), everyone will be asked to abort
  - If all vote “COMMIT”, everyone will be asked to commit

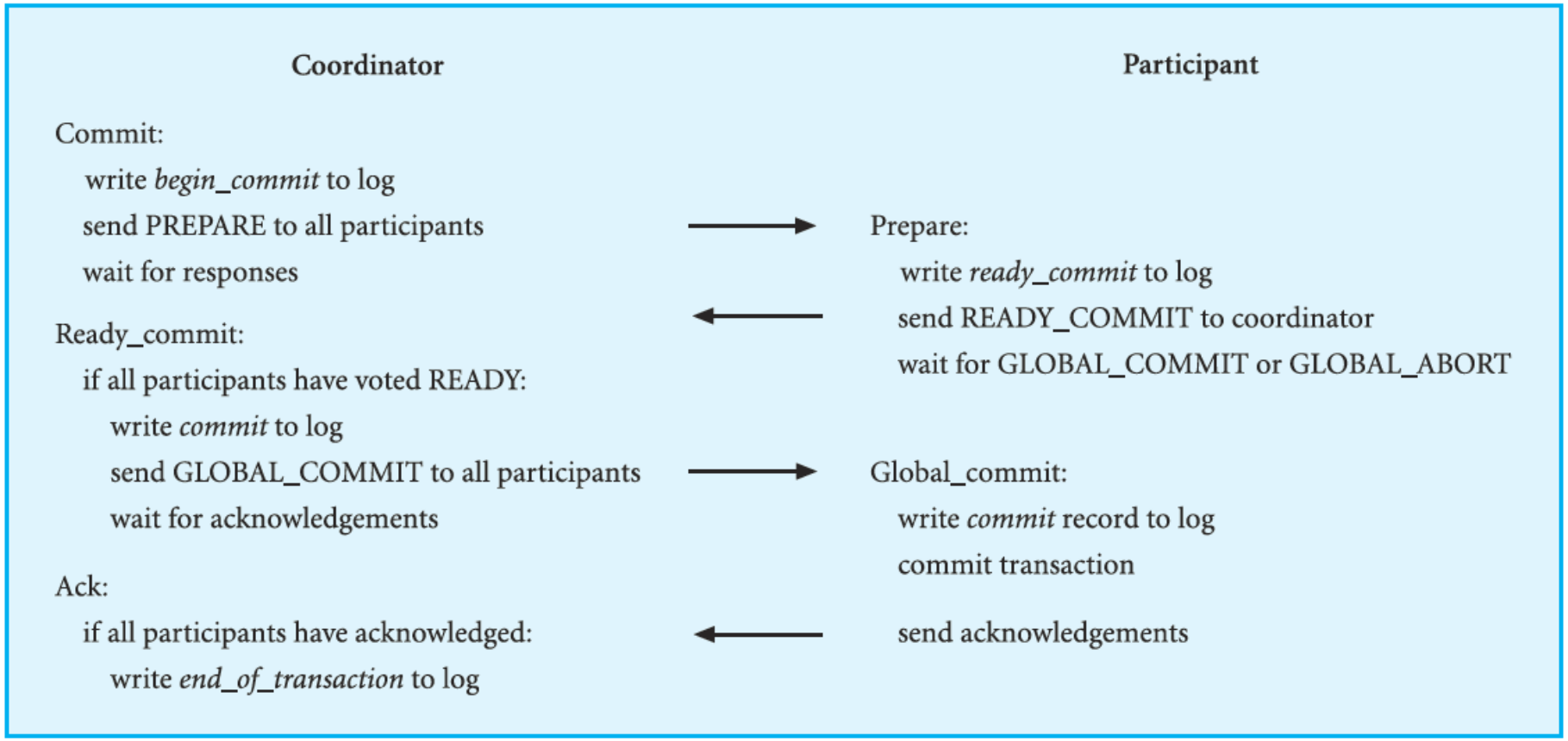
2PC: Stages

- Decision Stage:
  - Assume a vote has been made
  - Anyone that has voted to ABORT can actually precede on their own to abort immediately (as they will eventually be issued a request to abort)
  - Otherwise, wait for decision from coordinator

  - Decision is either GLOBAL COMMIT or GLOBAL ABORT
  - Each local site can use its own log to rollback or commit as necessary

2PC: Stages (Interaction Diagram & Details)

- The coordinator and participant interaction for a participant voting COMMIT



2PC: Stages (Action Diagram &Details)

- The coordinator and participant interaction for a participant voting ABORT