Lecture Notes

Week 2

INFS3200 Advanced Database Systems

Semester 1, 2021

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

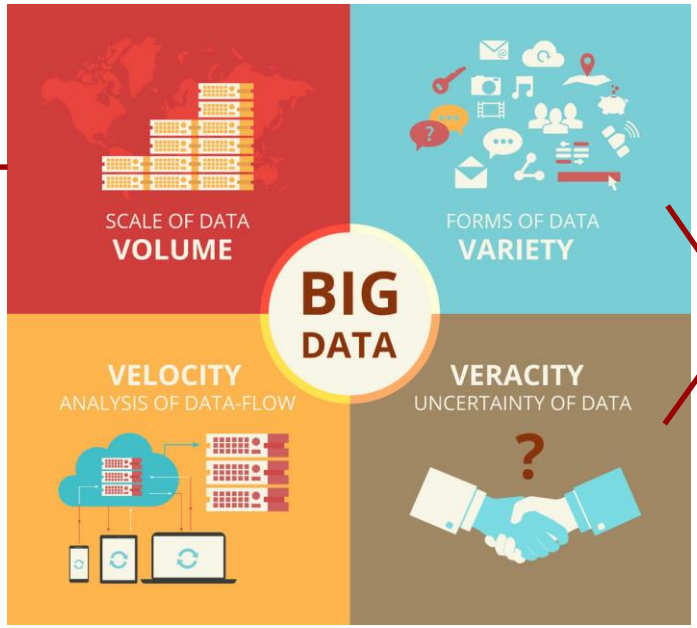
Distributed Database Design

Professor Xue Li

+ Course Structure

Distributed DB

- Design
- Query Processing
- Transaction Management



Data Preprocessing

- Integration
- Quality Management

Fundamental Issues

- Data Privacy
- Data Security

Data Warehouse

- Design
- Implementation

Some Real Systems

- Modern Computing Platforms
- Blockchain

+ Outline

- Distributed Databases Concepts
- Distributed Database Design
- Summary

+ Concepts

- Distributed computing systems
  - A number of autonomous processing elements, not necessarily homogeneous, that are interconnected by a computer network and that cooperate in performing their assigned tasks
- Distributed database
  - A collection of multiple logically interrelated databases, distributed over a computer network
- Distributed database management system
  - A software that manages a distributed database, while making the distribution transparent to the user

+ Why Distributed?

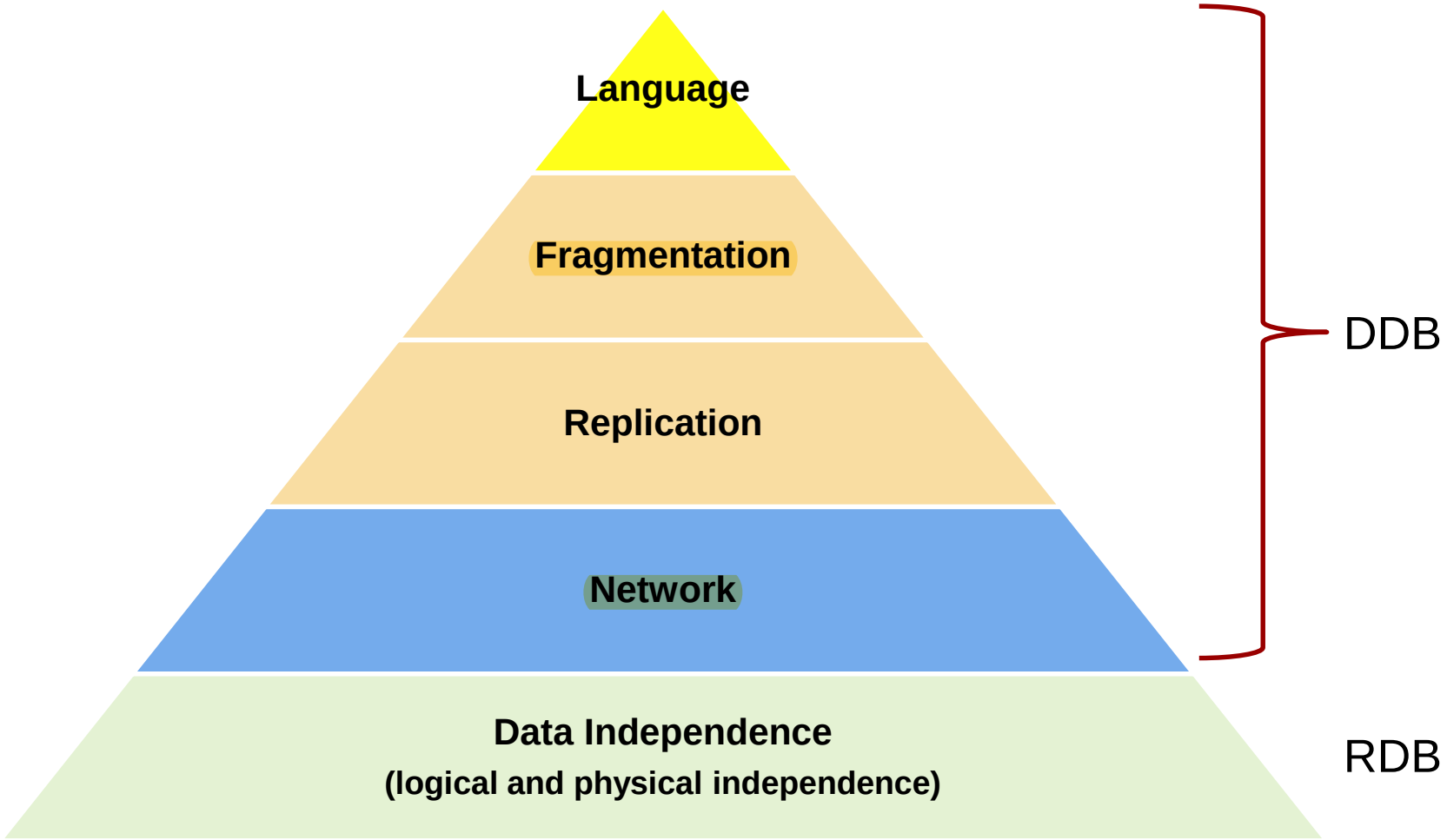
- Corresponds to the organizational structure of distributed enterprises
  - Physically distributed companies
  - E-commerce companies with different geographical centers
  - Supply chain management
- Economic benefits
  - Easy to manage - local autonomy
  - Able to share – global dictionary, data integrity management, transaction management
  - More computing power - processing efficiency and minimization of data transmission costs
- Key characteristic: work towards a common goal
  - Distributed vs parallel systems

+ Benefits of DDBMS

Transparency	• Users don't need to know...
Reliability	• When something fails...
Availability	• When something is needed...
Performance	• Do things faster...
Expansion	• Add more resources later...
Data Integrity	• Is DDB still a DB?

+ Many Types of Transparency

- Users don't need to know about…



+ Reliability and Availability

- Resources are more available in a reliable system
  - Poor reliability and poor availability because of faults, errors and failures, which can be caused by many reasons in a DBMS
- Replicated components (data & software) eliminate single point of failure
  - Failure of a single site or communication link should not bring down the entire system (*fault tolerance*)
  - Managing the 'reachable' data requires distributed transaction support which provides correctness in the presence of failures

...good reliability ⇒ good availability?
...poor reliability ⇒ poor availability?

+ Performance

- Data localization
  - Reduces contention for CPU and I/O services
  - Reduces communication overhead
- Parallel processing
  - Inter-query & intra-query parallelism

+ Expansion and Scalability

- Expansion by adding processing and storage power to the network – new nodes (Scale-Out)
- Replacing by a mainframe vs. adding more cheap computers to the network (Scale-Up)

...elasticity
...scale-up vs. scale-out

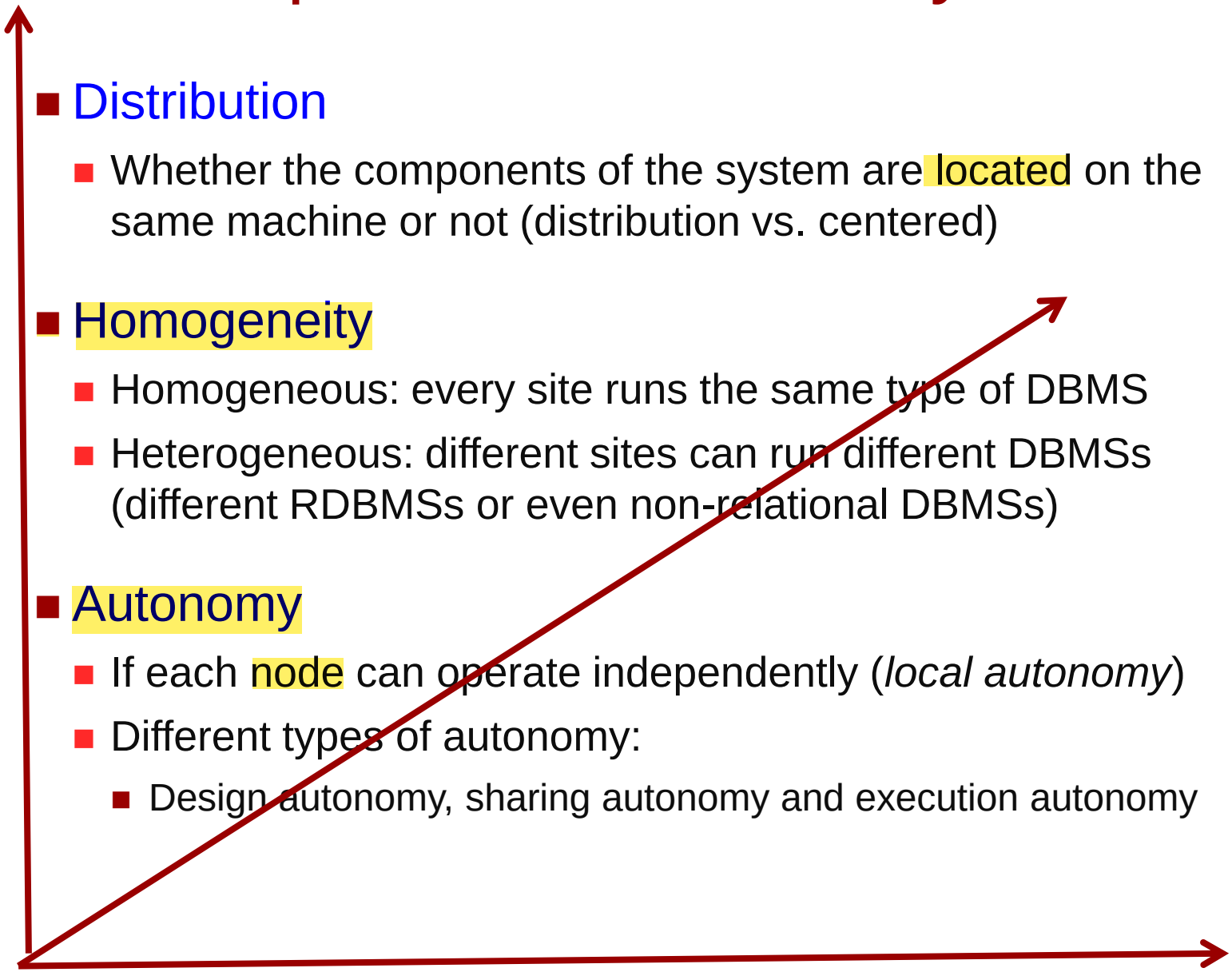
https://insights.illumio.com/post/102foor/scaling-up-vs-scaling-out-your-security-segmentation

When to Distribute?

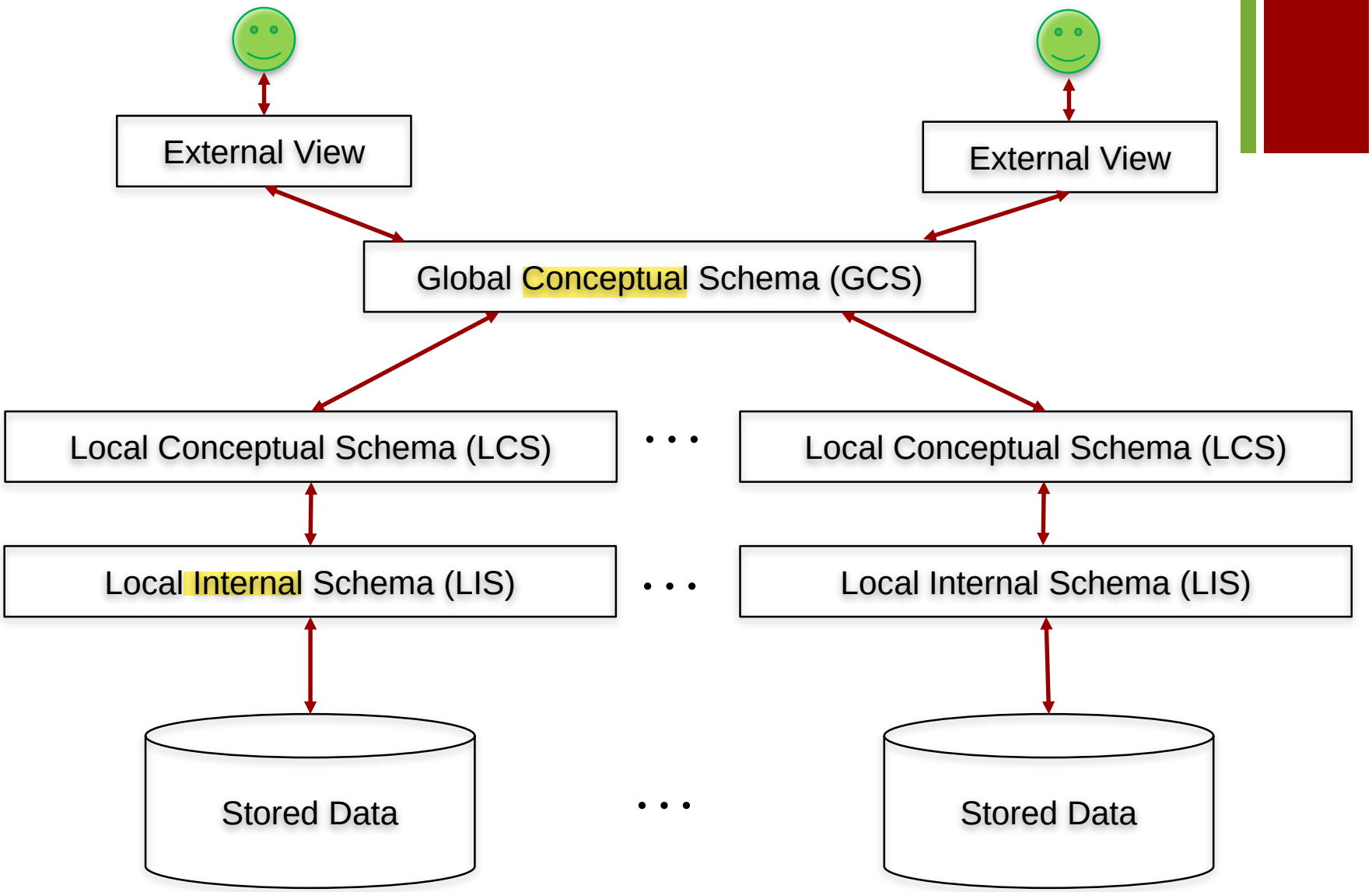
- Distribution is expensive to manage
- If you want to control the whole system, centralization in general is a better idea
  - But distribution is still desirable for handling very high data volumes and distributed data accesses
- Distribute when you don't control whole system
  - Localized data access
  - Mobile computing
  - Systems controlled by organizational units
- DDB theory essential to make an informed decision

+ Concepts in Distributed Systems

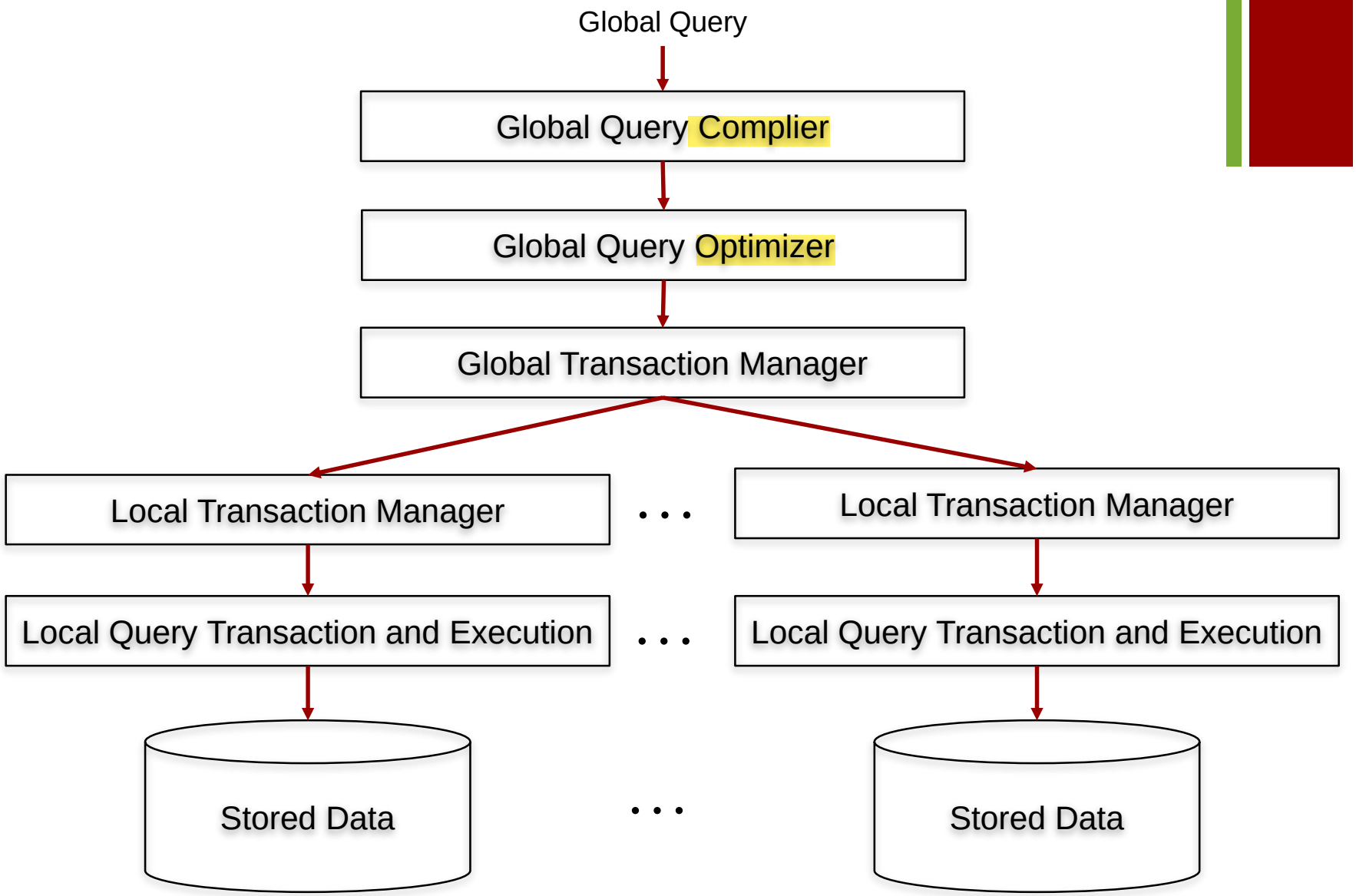
- Distribution
  - Whether the components of the system are located on the same machine or not (distribution vs. centered)
- Homogeneity
  - Homogeneous: every site runs the same type of DBMS
  - Heterogeneous: different sites can run different DBMSs (different RDBMSs or even non-relational DBMSs)
- Autonomy
  - If each node can operate independently (*local autonomy*)
  - Different types of autonomy:
    - Design autonomy, sharing autonomy and execution autonomy



+ DDB Architecture: Schemas



+ DDB Architecture: Components



+ Factors to Consider for DDB Design

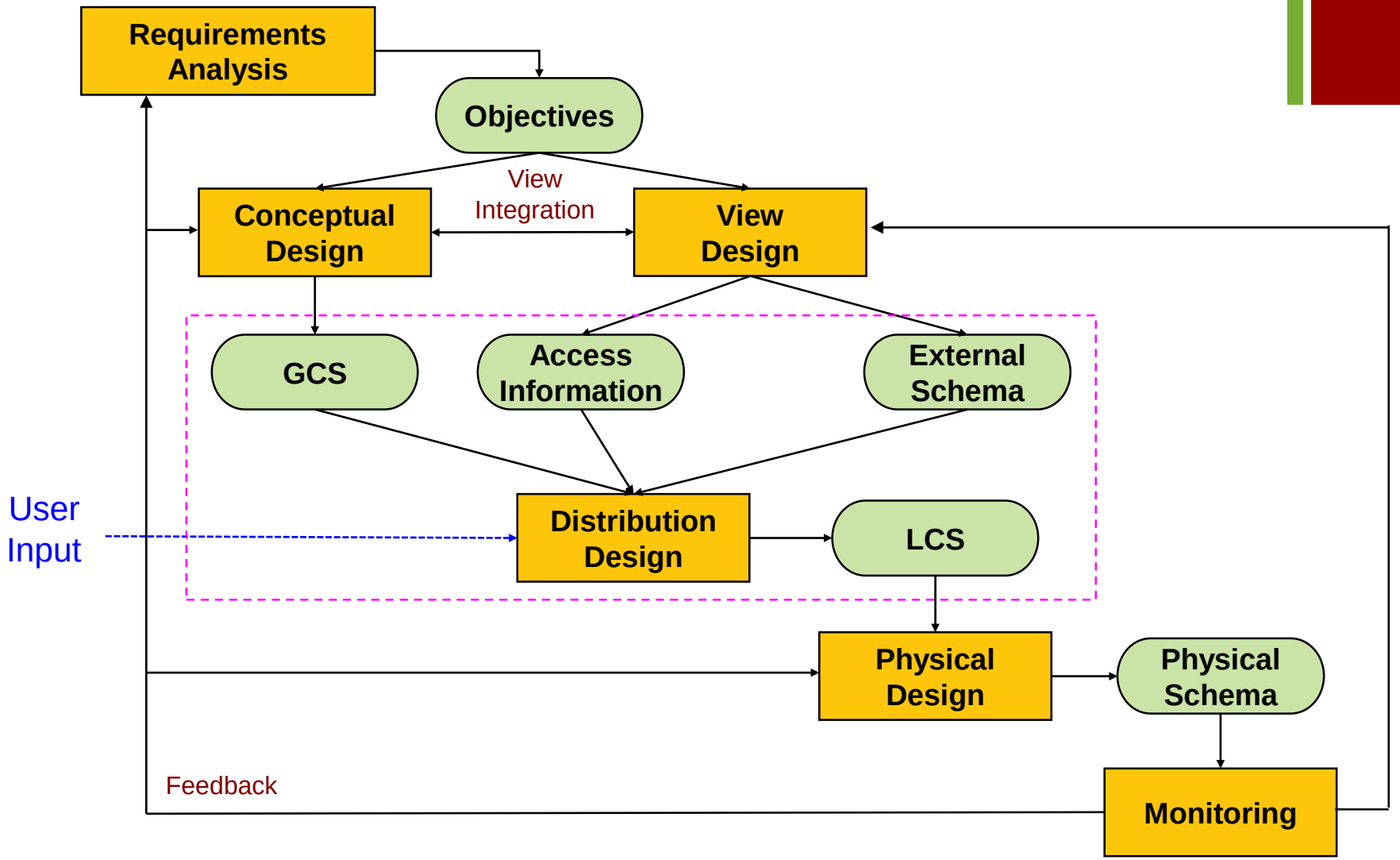
- Design of the network
- Distribution of the nodes
- Distribution of application software
- Distribution of data
  - Level of sharing
  - Access patterns
  - Level of knowledge about site databases…

+ Distributed Database Design

- The Top-down Approach – *a design approach*
  - Starting with the design of GCS (Global Conceptual Schema)
  - Considering data fragmentation and allocation
  - Designing component databases
- The Bottom-up Approach – *an integration approach*
  - Integration of pre-existing LCSs into a GCS
    - Design views and external schemas
    - Local systems will most likely be heterogeneous
  - Many problems with integration and interoperability
    - Often more business problems than technical problems
  - Aka database integration, or federated databases

...more about data integration later

+ Design Process: Top-Down



+ New Issues

- Fragmentation
  - How to split the data?
- Allocation
  - Where should each fragment go?

+ Example

Student(id, name, age, location,…)

Assume 75% of queries are about St Lucia and 25% of queries are about Gatton

```
Q1: select *
    from Student
    where location = "St Lucia" and …
Q2: select *
    from Student
    where location = "Gatton" and …
```

If there are two sites: St Lucia and Gatton
We can have fragmentation F = {F1, F2}

$F1 = \sigma_{location="St\ Lucia"}\ Student$

$F2 = \sigma_{location="Gatton"}\ Student$

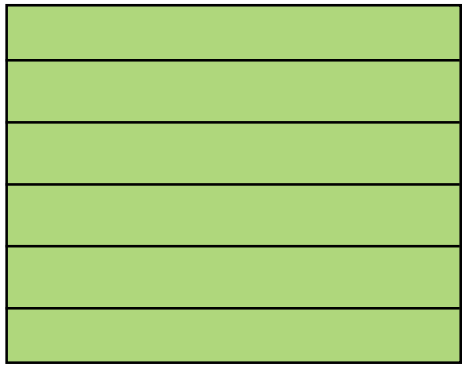
+ Why Fragmentation ?

- Application views are subsets of relations - a fragment is a subset of tuples in a relation
- If there is no fragmentation
  - Option 1: Entire table is at one site - high volume of remote data accesses
  - Option 2: Entire table is replicated - problems in executing updates
- Better performance from parallel execution of query, since sub-queries can operate on fragments, in parallel!
- Sites may own data, so fragmentation is sometimes forced

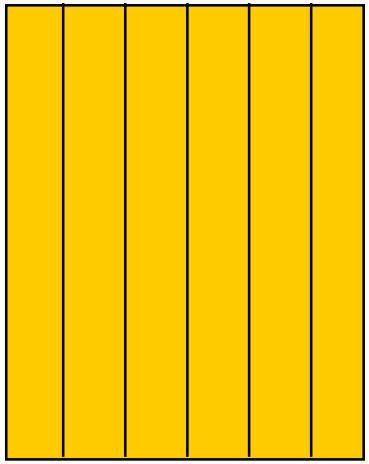
+ Drawbacks of Fragmentation

- Application views do not require mutually exclusive fragments - queries on multiple fragments will **suffer performance degradation** (costly joins and unions)
- Semantic **integrity control** - checking for dependency of attributes that belong to several fragments may require access to multiple sites
- Questions:
  - How much should we fragment?
  - How to resolve conflicting requirements from different applications?

+ Types of Fragmentation



Horizontal



Vertical

Primary: using predicates on this table only
Derived: using foreign relation predicates

...how to derive fragments in SQL?
...it's also possible to have mixed fragmentation

+ Horizontal Fragmentation: Example

PERSON

NAME	ADDRESS	PHONE	SAL
John Smith	44, Here St	3456 7890	34000
Alex Brown	1, High St	3678 1234	48000
Harry Potter	99, Magic St	9976 4321	98000
Jane Morgan	87, Riverview	8765 1237	65800
Peter Jennings	65, Flag Rd	9851 1238	23980

How can this table be partitioned?

PERSON-FRAGMENT1

NAME	ADDRESS	PHONE	SAL
John Smith	44, Here St	3456 7890	34000
Peter Jennings	65, Flag Rd	9851 1238	23980

Can you define a predicate for each of these fragments?

PERSON-FRAGMENT2

NAME	ADDRESS	PHONE	SAL
Alex Brown	1, High St	3678 1234	48000
Harry Potter	99, Magic St	9976 4321	98000
Jane Morgan	87, Riverview	8765 1237	65800

+ Horizontal Partitioning Techniques

- Round robin partitioning
  - Distribute data evenly
  - Good for scanning the whole table 
  - Not good for point or range queries 
- Hash partitioning
  - Distribute data evenly if the hash function is good
  - Good for point queries on key and joins 
  - Not good for range queries and point queries not on key 
- Range partitioning
  - Good for some range queries on partition attribute 
  - Need to select good vector to avoid data/execution skew 

+ Good Fragmentation?

$$F1 = \sigma_{age>35}\ Student$$
$$F2 = \sigma_{age<20}\ Student$$

$$F1 = \sigma_{age>35}\ Student$$
$$F2 = \sigma_{age<40}\ Student$$

+ Fragmentation Properties

- $R \Rightarrow \boldsymbol{F} = \{F_1, F_2, \ldots F_n\}$
- Horizontal fragmentations should not only ensure that the design matches access patterns, but also meet the following three properties:
  - Completeness: $\forall t \in R, \exists F_i \in \boldsymbol{F}: t \in F_i$
  - Disjointness: $\forall F_i, F_i \in \boldsymbol{F}, \quad i \neq j \Rightarrow F_i \cap F_j = \phi$
  - Reconstruction: $R = \bigcup_{i=1,\ldots,n} F_i$

...why disjoint?

...if a fragmentation is complete, is it also reconstructable?

+ Desirable Fragmentation

- Primary horizontal fragmentation is defined using simple conditions
- To ensure completeness and disjointness
  - Approach 1: manually check if the predicates meet the properties

$F1 = \sigma_{age>35}\ Student$
$F2 = \sigma_{age\leq 35}\ Student$

$F1 = \sigma_{location=\text{“St Lucia”}}\ Student$
$F2 = \sigma_{location=\text{“Gatton”}}\ Student$

  - Approach 2: automatically generate predicates with such properties

+ Minterm Predicates

- Simple predicate: $\boldsymbol{P_i = a_j\, \theta\ value}$
- Given a set of simple predicates $P = \{P_1, P_2, \ldots, P_n\}$
- **Minterm predicate** $M_i = P_1^* \wedge P_2^* \wedge \ldots \wedge P_n^*$
  - P_i^* is either P_i or $\neg P_i$
  - Thus, there can be 2^n such predicates in total
- Eliminate useless ones (some are invalid)
  - Outcome: $M = \{M_1, M_2, \ldots M_k\}$
  - Each minterm predicate defines a **fragmentation**
- More discussions at next week's tutorial
  - Examples
  - Why minterm predicates meet the properties of completeness and disjointness?

+ Derived Horizontal Fragmentation

Student(id, name, age, location,…)

$F = \{F1, F2\}$

$F1 = \sigma_{location=\text{“St Lucia”}} Student$

$F2 = \sigma_{location=\text{“Gatton”}} Student$

Study(id, course,…)

Common query for task:

Given student name, list courses she has enrolled in

+ Fragmentation Using Semi-join

- Semi-join operator: $\boldsymbol{R1} \propto \boldsymbol{R2} = \boldsymbol{\pi}_{\boldsymbol{R1}} \boldsymbol{R1} \bowtie \boldsymbol{R2}$
- $R \Rightarrow \boldsymbol{F} = \{F_1, F_2, \ldots\}, S \Rightarrow \boldsymbol{G} = \{G_1, G_2 \ \ldots\}, \ G_i = S \propto F_i$
- R is called the owner of the fragmentation, and S is a member

Why do we need semi-join in data fragmentation?

- Questions:
  - How to ensure completeness?
  - How to ensure disjointness?
  - How to reconstruct the original table?

+ Vertical Fragmentation: Example

PERSON

NAME	ADDRESS	PHONE	SAL
John Smith	44, Here St	3456 7890	34000
Alex Brown	1, High St	3678 1234	48000
Harry Potter	99, Magic St	9976 4321	98000
Jane Morgan	87, Riverview	8765 1237	65800
Peter Jennings	65, Flag Rd	9851 1238	23980

PERSON1

NAME	ADDRESS	PHONE
John Smith	44, Here St	3456 7890
Alex Brown	1, High St	3678 1234
Harry Potter	99, Magic St	9976 4321
Jane Morgan	87, Riverview	8765 1237
Peter Jennings	65, Flag Rd	9851 1238

PERSON2

NAME	SAL
John Smith	34000
Alex Brown	48000
Harry Potter	98000
Jane Morgan	65800
Peter Jennings	23980

...primary Key [NAME] is included in all fragments, why?

+ Vertical Fragmentation Properties

- Given a table $\boldsymbol{R}\,[\,\boldsymbol{A}\,], \boldsymbol{A} = \{\boldsymbol{a_1}, \boldsymbol{a_2}\ \ldots, \boldsymbol{a_n}\}, \boldsymbol{A_i} \subseteq \boldsymbol{A}$
- $\boldsymbol{R}\,[\,\boldsymbol{A}\,] \Rightarrow\ \boldsymbol{R_i}[\boldsymbol{A_i}],\ \ \boldsymbol{i} = \boldsymbol{1}, \ldots, \boldsymbol{m}$

- Completeness: $\bigcup_{i=1,\ldots,m} A_i = A$
- Disjointness: $\forall i, j,\ i \neq j\ \Rightarrow\ A_i \cap A_j = \phi$?
  - This does not guarantee reconstructability!
  - One way is to duplicate key attributes in every fragmentation

How to Group Attributes?

- Affinity matrix

	a1	a2	a3	a4	a5
a1	-	-	-	-	-
a2	50	-	-	-	-
a3	45	48	-	-	-
a4	1	2	0	-	-
a5	0	0	4	75	-

+ Grouping Co-accessed Attributes

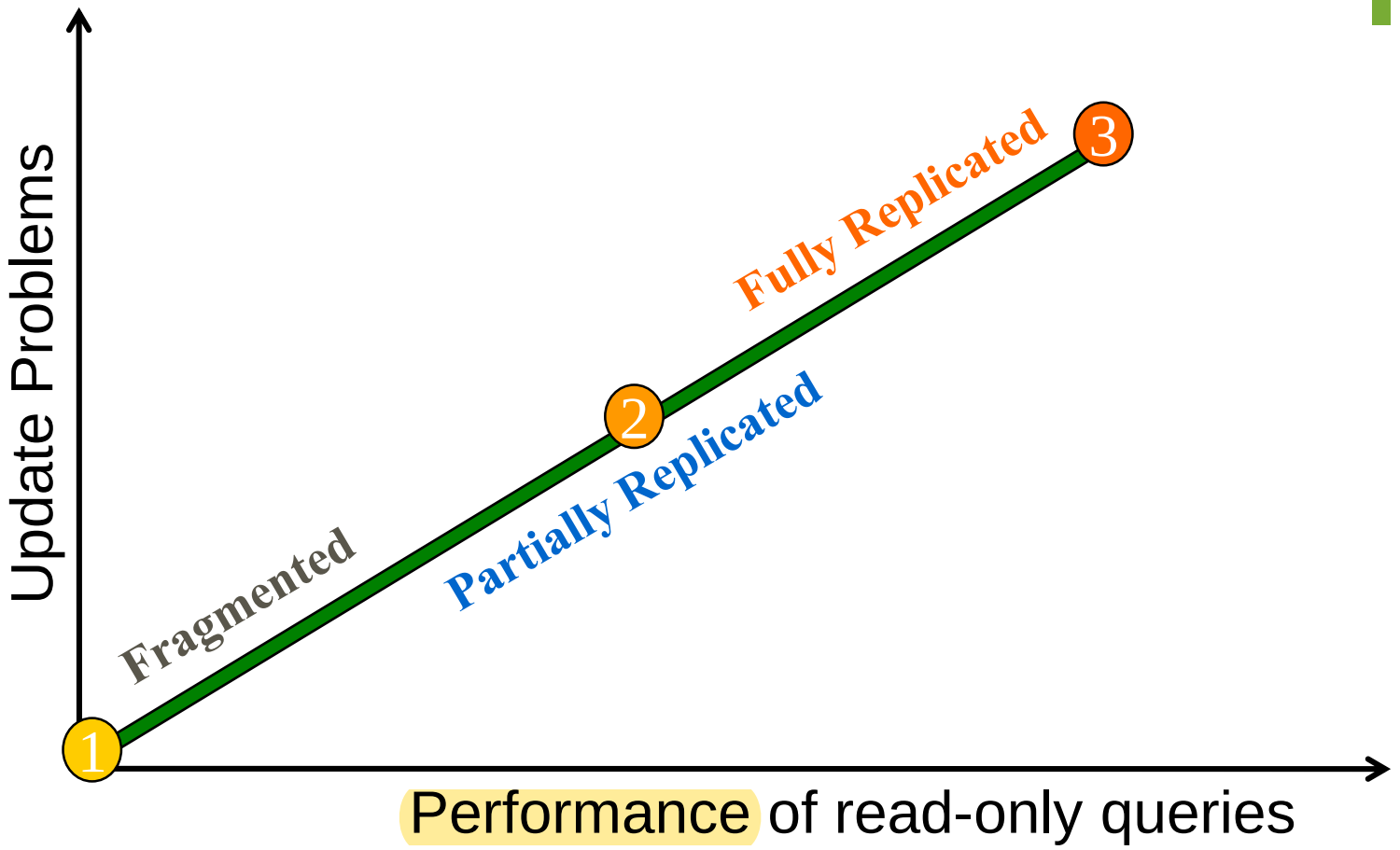
	a1	a2	a3	a4	a5
a1	-	-	-	-	-
a2	50	-	-	-	-
a3	45	48	-	-	-
a4	1	2	0	-	-
a5	0	0	4	75	-

$R_1[k, a_1, a_2, a_3]$ $R_2[k, a_4, a_5]$

+ Replication

- A centralized system has a single copy of all data
- One way to distribute a database is to copy some or all data, possibly several times
  - Each copy stored at a different node in the network
  - Query routed to “nearest” node
- Fragments can be replicated
  - Full replication, and partial replication
- All copies of replicated data must be updated in a single transaction to maintain atomicity and consistency

+ Trade-off


Update Problems
Fragmented
Partially Replicated
Fully Replicated
1
2
3
Performance of read-only queries

+ Allocation

- Now data has been partitioned (and possibly replicated), where should each fragment go?
- Many issues to consider:
  - Where do queries originate?
  - What is the communication cost?
    - Size of answers? Relations?
  - What is the storage capacity and cost at sites?
    - Size of fragments?
  - What is the processing power at sites?
  - What is the query processing strategy?
    - How are joins done?
    - Where are answers collected?

What is fragment allocation?

An Optimisation Problem

- Best placement of fragments and/or best number of copies to a set of sites in order to:
  - Minimise query response time
  - Maximise system throughput
  - Minimise some other costs, such as cost of storing and querying a fragment at a site, cost of updating a fragment at all sites, cost of data communication
- Subject to constraints
  - Available storage
  - Available bandwidth, power, etc.
  - Other, e.g., keep 90th percentile response time < x

…what queries?

+ Inputs to Fragment Allocation

- Database information
  - Fragment Selectivity: #tuples of a fragment F_i that need to be accessed in order to process a query q_j
- Application information
  - Read access: the number of read accesses that a query q_j makes to a fragment F_i during execution
  - Update access: the number of update accesses that a query q_j makes to a fragment F_i during execution
- System information
  - Site information
    - Knowledge of storage and processing capacity
  - Network information
    - Communication cost (channel capacities, distances, protocol overhead)

...how to find optimal allocation?

+ Allocation

- **Input:**
  - **Fragments**: F={F1,F2,…,Fm} **Sites**: S={S1,S2,…,Sn}
  - Typical **queries**: Q={Q1,Q2,…,Qk} detailed with read/write information
- **Output** : an allocation can be represented by a group of mappings:
  - Xij = 1 if Fi is assigned to Sj; otherwise Xij=0

Total_cost = total_local_processing_cost + total_data_exchange_cost + total_stoarge_cost

- The allocation problem is to find an **optimal mapping** to minimize the total cost, with challenges: (1) NP-complete, (2) hard to estimate precise costs, (3) changing costs over time
- **Solutions:** heuristics-based with many simplifications (e.g., communication costs only), supporting dynamic adjustment

What is the definition of cost?

+ Some Questions

- Fragmentation
  - Which tuples to group together in horizontal fragmentation?
  - How to check if a horizontal fragmentation is complete?
  - Which attributes to group together in vertical fragmentation?
  - How do we insert data when a table is fragmented?
  - What about database constraints when a table is fragmented?
- Allocation
  - How to formulate the optimization problem?
  - How to solve the optimization problem?

+ Summary of Week 2

- Centralized vs distributed – understanding of main differences
- Transparency levels and problems associated with them
- Issues with provision of *full* scale DBMS functionality
- Distributed design types (top-down and bottom-up)
- Data fragmentation, replication and allocation
- Distributed databases are much more complex than the centralized counterparts
- Next week: **Distributed Query Processing**

+ Recommended Readings

- Elmasri & Navathe, 6th edition
  - Chapter 26: Introduction to Distributed Databases
- Elmasri & Navathe, 7th edition
  - Chapter 23: Distributed Database Concepts
- Ozsu & Valduriez: *Principles of Distributed Database Systems*, 3rd edition, Springer
  - Chapter 1: Introduction
  - Chapter 3: Distributed Database Design

Textbook (PDF): https://link.springer.com/book/10.1007/978-1-4419-8834-8