NOSQL DDBBs: Introduction to MongoDB

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Campus d'Excel·lència Internacional

Índex

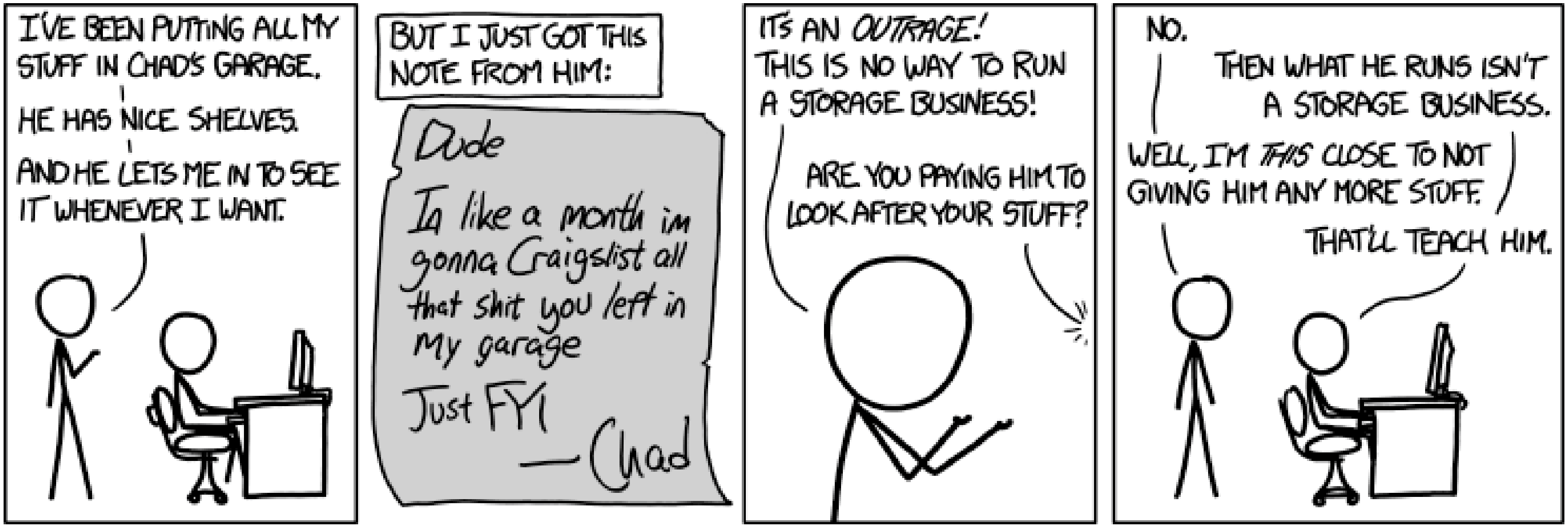
FOUNDATIONS

INTRODUCTION TO BIG DATA AND NOSQL

A New Business Model

- Traditionally databases have been seen as a passive asset
  - OLTP systems: Data gathered is structured to facilitate (automate) daily operations
  - The relational model as *de facto* standard
- Soon, many realized data is a valuable asset for any organization. So, *use* it!
  - Decisional systems: Stored data is analysed to better understand our activity (*I want to know*)
  - Data warehousing as *de facto* standard

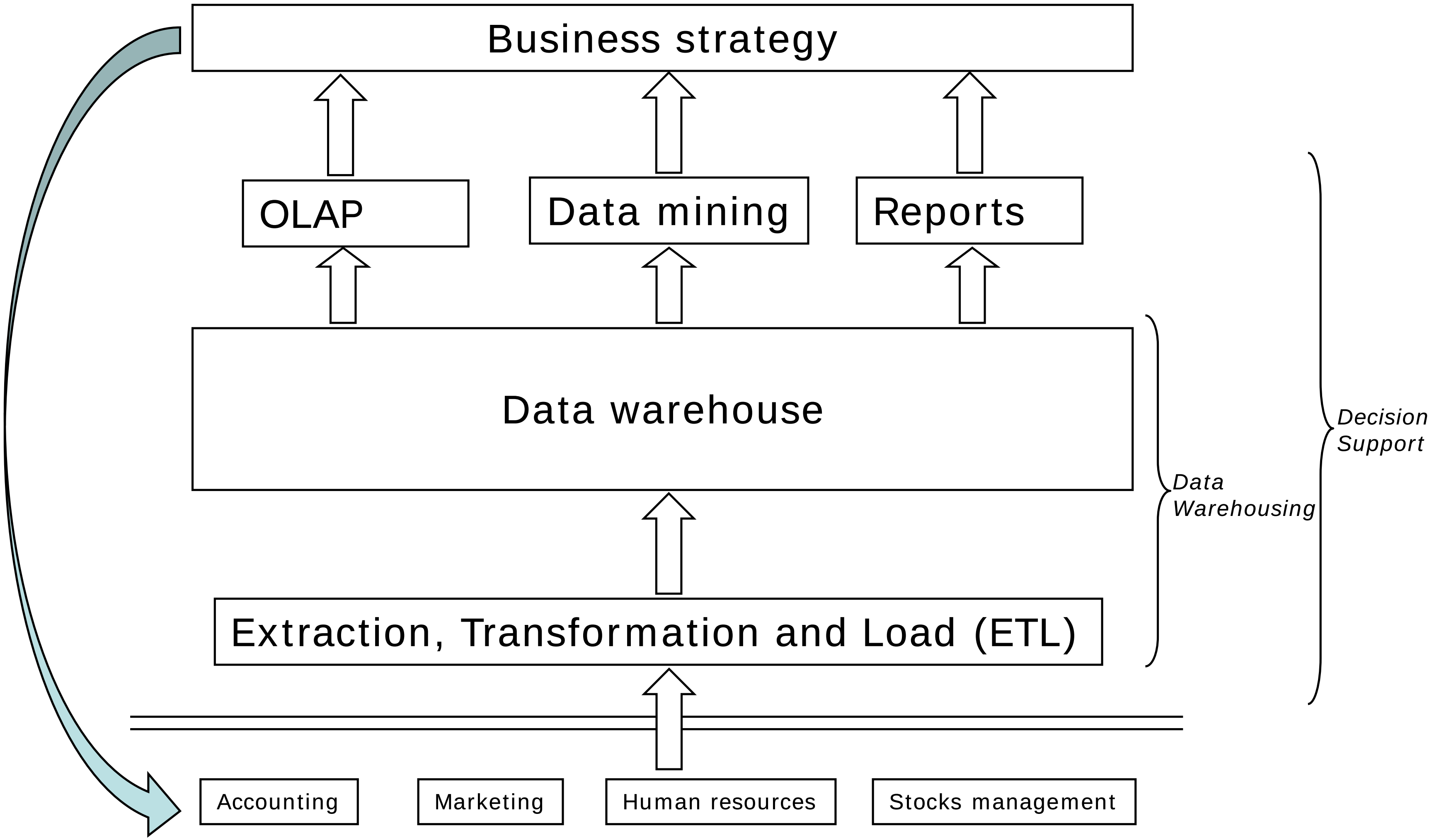
Instagram’s Fable (xkcd.com)



Data As The New Cornerstone

- We have witnessed the bloom of a new business model based on data analytics: ***Data is not a passive but an active asset***
  - «Data is the new oil!» – Clive Humby, 2006
  - «No! Data is the new soil» – David McCandless, 2010
- Organization must adapt their infrastructures to benefit from the data deluge
  - Digital data doubling every 18 months (IDC)
- Innovation and entrepreneurship are mandatory!
- Some numbers:
  - In 2014, the overall Business Intelligence (BI) market value will be $100 billion (the Economist)
  - It is expected to have the largest increment in data management and analytics, at almost 10% each year, roughly twice as fast as the software business as a whole. The market growth is estimated in $14 billion in 2014, up from $8.8 billion in 2008 (Forrester and the Economist)

The Business Intelligence Cycle

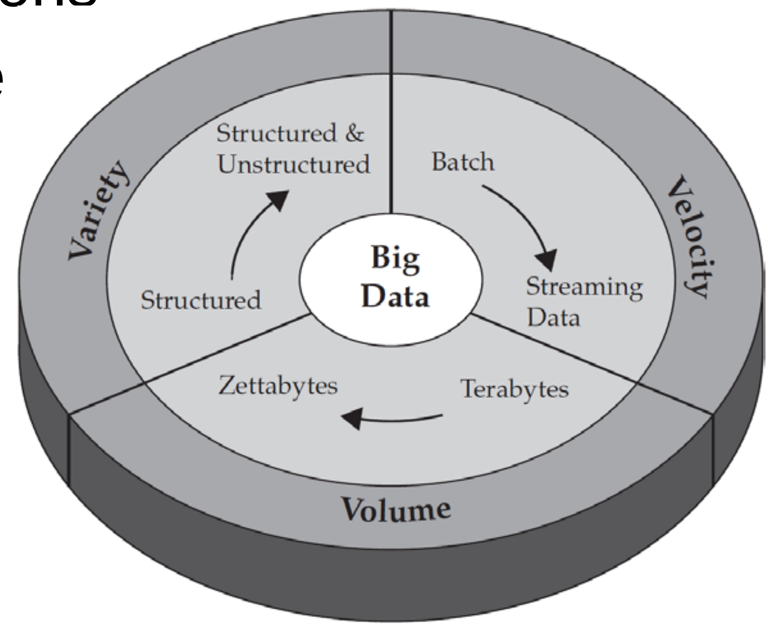


Data Warehousing Vs. Big Data

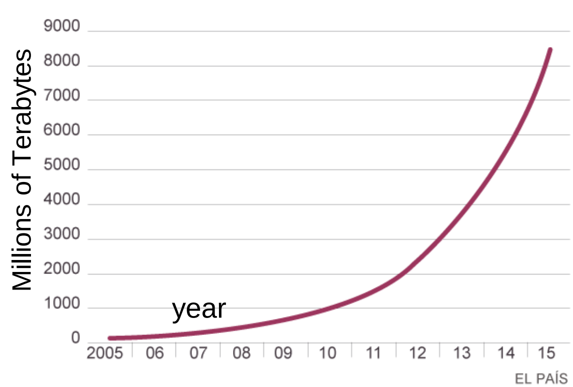
- Both are decision support systems (DSS)
- Big Data can be seen as the evolution of Data Warehousing ecosystems to incorporate external data to the decision making processes of the organization as first class-citizen
  - Open data (external) Vs. organization data (internal)
  - Adding *plug-and-play* data sources Vs. monolithic sources
  - *On-demand* transformations Vs. heavy transformations
  - *On-demand* analysis Vs. traditional analysis
  - And many others consequences…
    - Semantic-aware solutions
    - Privacy
    - Etc.

Big Data Definition

- Definition based on the limitations traditional DSS cannot overcome
  - Velocity
  - Volume
  - Variety
- Then, other traditional V's have been added
- Variability
- Validity/Veracity
- Value



From IBM "Understanding Big Data"

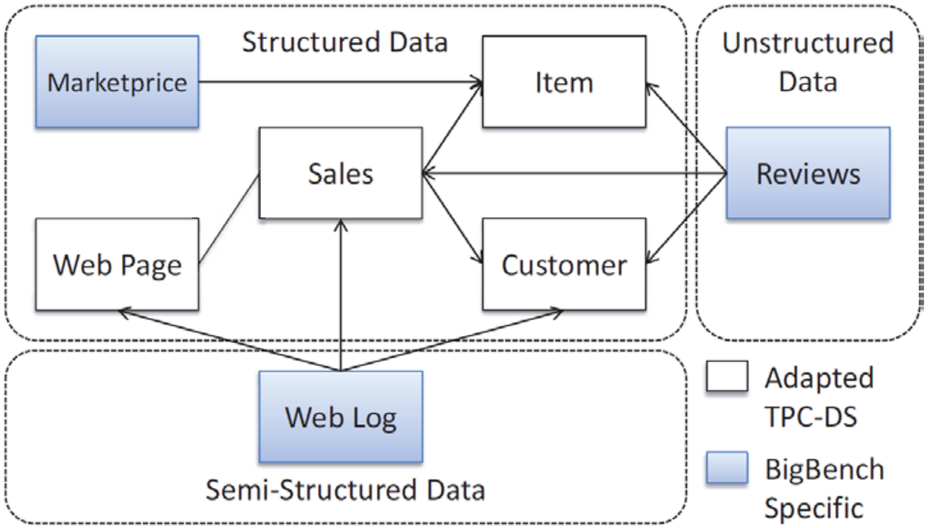


An Example: BigBench

Query processing type	Total	Percentage(%)
Declarative	10	33.3
Procedural	7	23.3
Mix of Declarative and Procedural	13	43.3

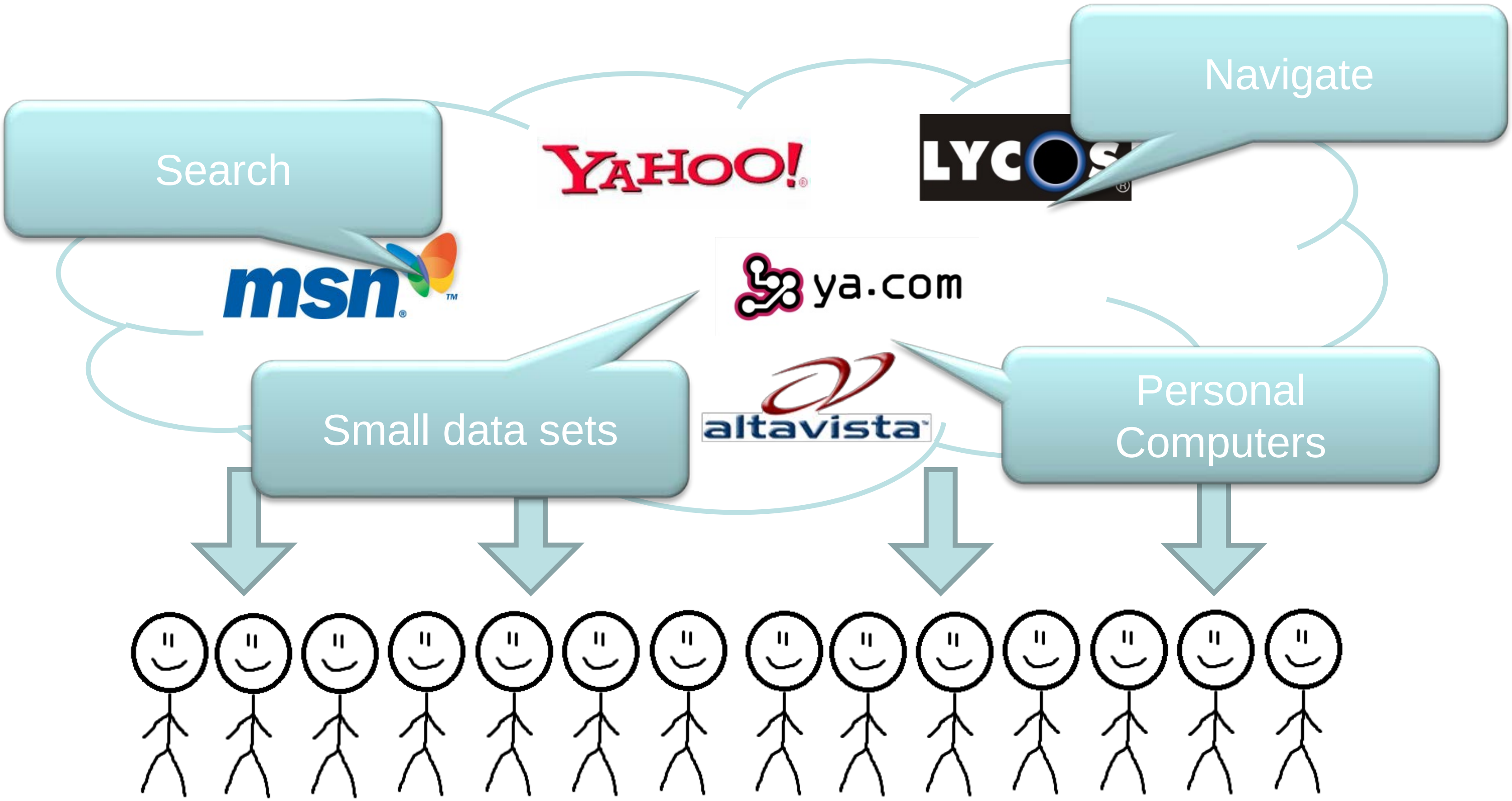
Data sources	Total	Percentage(%)
Structured	18	60.0
Semi-structured	7	23.3
Un-structured	5	16.7

Analytic techniques	Total	Percentage(%)
Statistics analysis	6	20.0
Data mining	17	56.7
Reporting	8	26.7



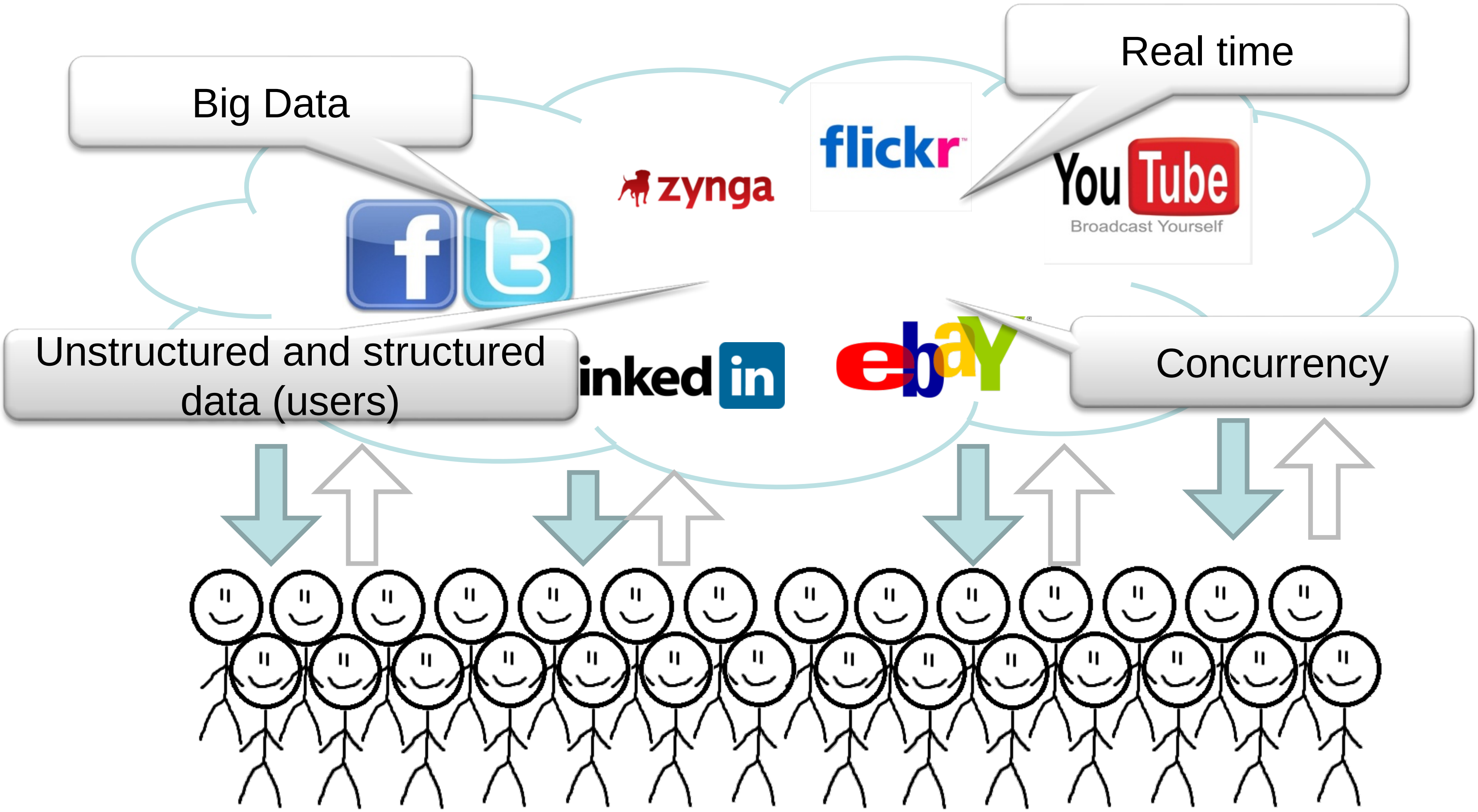
The End of an Architectural Era

WEB 1.0 – Read Era



The End of an Architectural Era

WEB 2.0 – Write Era



RDBMS: One Size Does Fit All

- Mainly write-only Systems (e.g., OLTP)
  - Data storage
    - Normalization
  - Queries
    - Indexes: B+, Hash
    - Joins: BNL, RNL, Hash-Join, Merge Join
- Read-only Systems (e.g., DW)
  - Data Storage
    - Denormalized data
  - Queries
    - Indexes: Bitmaps
    - Joins: Star-join
    - Materialized Views

BUT, WHAT IF I NEED TO DEAL WITH MASSIVE READS AND WRITES AT THE SAME TIME?

RDBMS Approach

Too many reads? → data replication

Too many writes? → data fragmentation

- Distributed RDBMS (DRDBMS) are not flexible enough
  - Logging
    - Persistent redo log
    - Undo log
  - CLI interfaces (JDBC, ODBC, etc.)
  - Concurrency control (locking)
  - Latching associated with multi-thread
  - Two-phase commit (2PC) protocol in distributed transactions
  - Buffers management (cache disk pages)
  - Variable length records management
    - Locate records in a page
    - Locate fields in a record

A**C**I**D** properties

AC**I**D properties

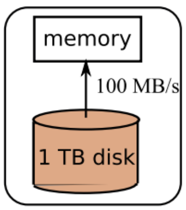
ACID properties

Activity: Why Distribution?

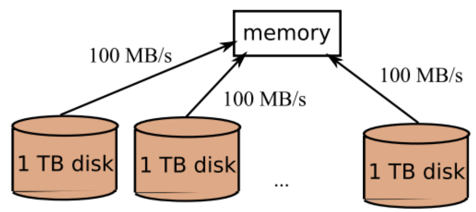
- *Objective: Recognize the benefits of distributing data*
- *Tasks:*
  1. *(15') By pairs, answer the following questions:*
     a) *How long would it take to read 1TB with sequential access (fig. a)? (in secs)*
        – *Can you identify any additional drawback to be considered?*
     b) *How long would it take to read 1TB with parallel access (fig. b)? Assume 100 disks on the same machine with shared-memory and infinite CPU capacity.*
        – *Can you identify any additional drawback to be considered?*
     c) *How long would it take to read 1TB with distributed access (fig. c)? Assume 100 shared-nothing machines in a star-shape LAN in a single rack where all data is sent to the center.*
        – *Can you identify any additional drawback to be considered?*
     d) *Now, repeat the exercise considering a single random access. What changes?*
  2. *(5') Discussion*

Type	**Latency**	**Bandwidth**
Disk	$\approx 5 \times 10^{-3}$s (5 millisec.);	At best 100 MB/s
LAN	$\approx 1 - 2 \times 10^{-3}$s (1-2 millisec.);	$\approx$ 1GB/s (single rack); $\approx$ 100MB/s (switched);
Internet	Highly variable. Typ. 10-100 ms.;	Highly variable. Typ. a few MB/s.;

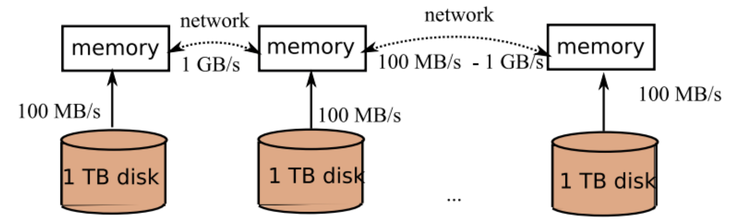
Bottom line (1): it is approx. one order of magnitude faster to exchange main memory data between 2 machines in a data center, that to read on the disk.
Bottom line (2): exchanging through the Internet is slow and unreliable with respect to LANs.



a. Single CPU, single disk



b. Parallel read: single CPU, many disks



c. Distributed reads: an extendible set of servers

NOSQL Goals

- Schemaless: No explicit schema

[data structure]

- Reliability / availability: Keep delivering service even if its software or hardware components fail

[distribution]

- Scalability: Continuously evolve to support a growing amount of tasks [distribution]
- Efficiency: How well the system performs, usually measured in terms of response *time* (latency) and *throughput* (bandwith) [distribution]

The Means to Reach the Goals

- Distributed Databases
  - Divide-and-conquer principle
    - Use (and abuse) of parallelism
- Flexible data models
  - Reduce the *impedance mismatch*
    - Reduce the code overhead to make transformations between different data models
- New architectural solutions
  - Relational database architectures date back to the 70s
    - Some modules may not be needed (incur in a unnecessary performance overhead)

The Problem is **NOT** SQL

- Relational systems are too generic…
  - OLTP: stored procedures and simple queries
  - OLAP: ad-hoc complex queries
  - Documents: large objects
  - Streams: time windows with volatile data
  - Scientific: uncertainty and heterogeneity
- … But the overhead of RDBMS has nothing to do with SQL
  - Low-level, record-at-a-time interface is not the solution

SQL Databases vS. NoSQL Databases
Michael Stonebraker
Communications of the ACM, 53(4), 2010

Distributed Aspects

I. Distributed DB design
- Node distribution
- Data fragments
- Data replication

II. Distributed DB catalog
- Fragmentation trade-off: Where to place the DB catalog
  - Global or local for each node
  - Centralized in a single node or distributed
  - Single-copy vs. Multi-copy

III. Distributed query processing
- Data distribution / replication
- Communication overhead

IV. Distributed transaction management
- How to enforce (or not) the ACID properties
  - Replication trade-off: Queries vs. Data consistency between replicas (updates)
  - Distributed recovery system
  - Distributed concurrency control system

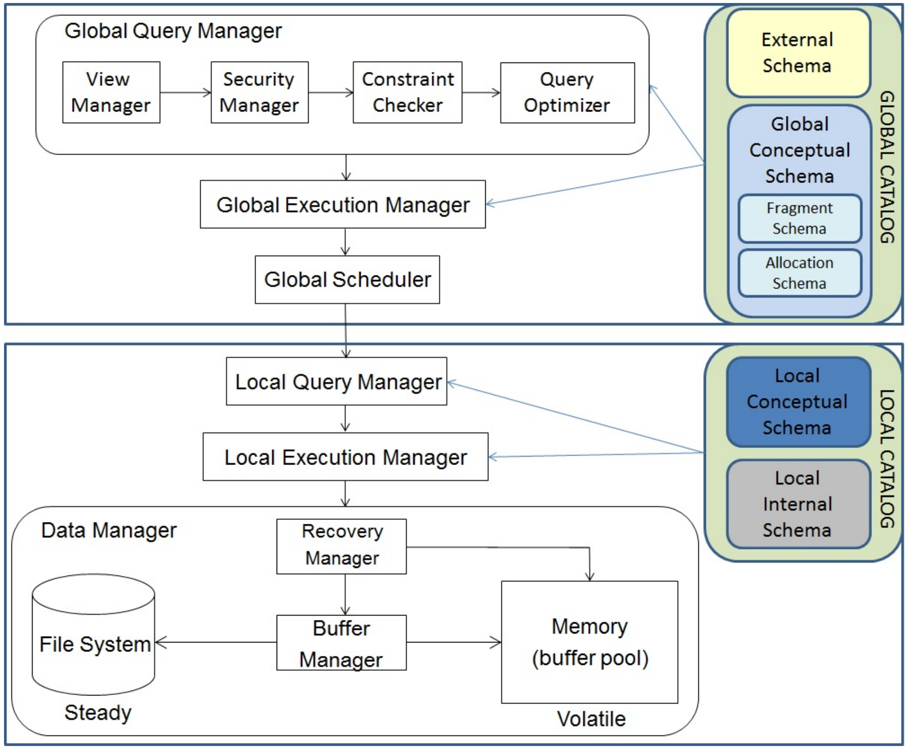
V. Security issues
- Network security

Flexible Data Models

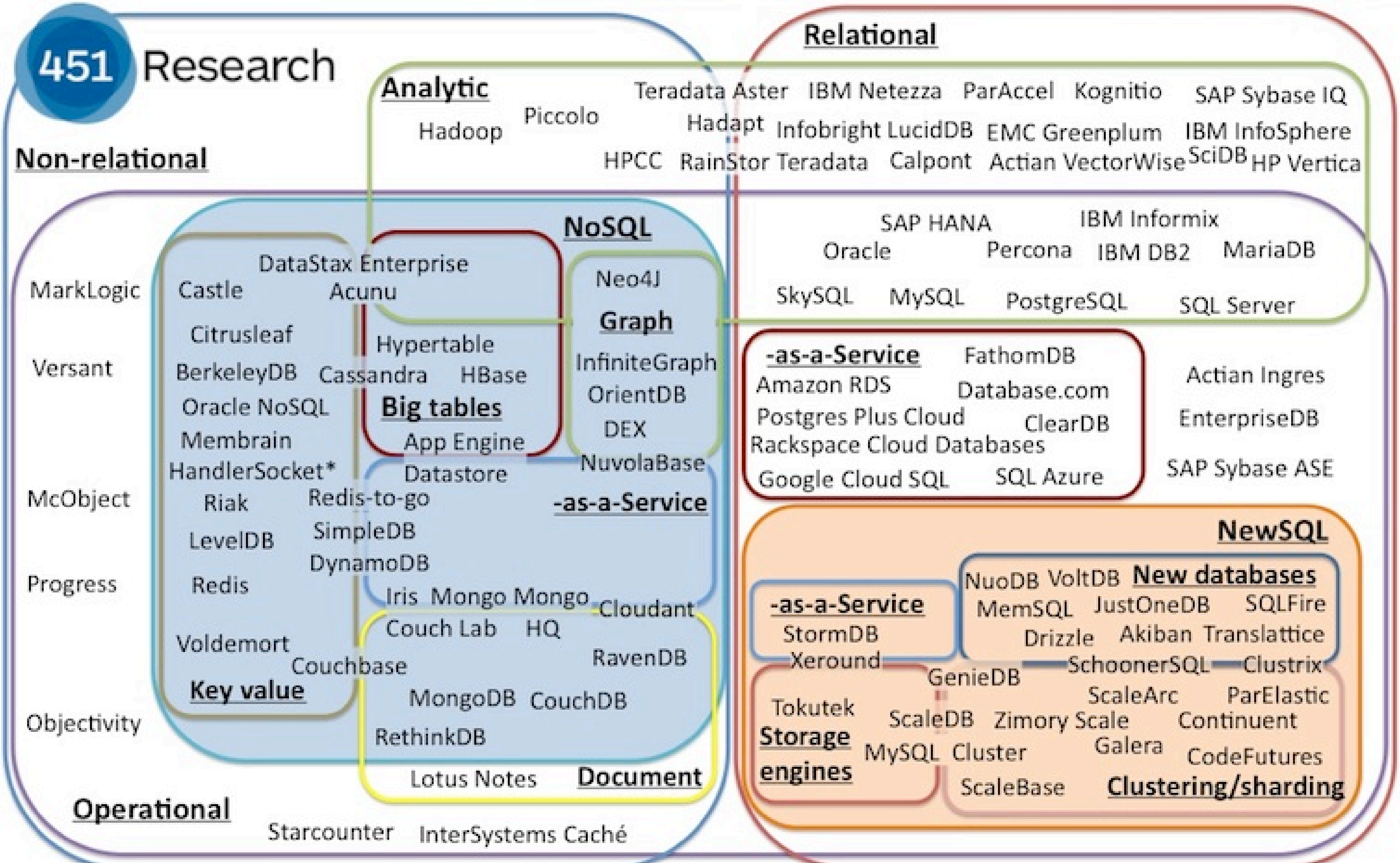
I. Relational databases give for granted the relational model
II. The NOSQL wave presents other data models to boost performance and flexibility
    I. Key-value
    II. Document (kind of key-value)
    III. Graph-based
    IV. Streams
    V. Etc.

New Architectures

- NOSQL introduces a critical reasoning on the reference database architecture
  - ALL relational databases follow System-R architecture (late 70s!)



The evolving database landscape

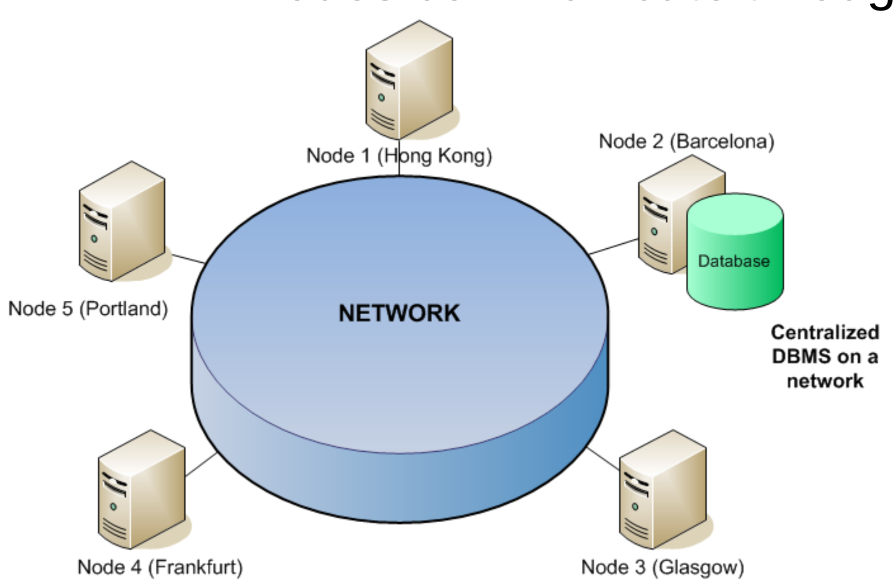


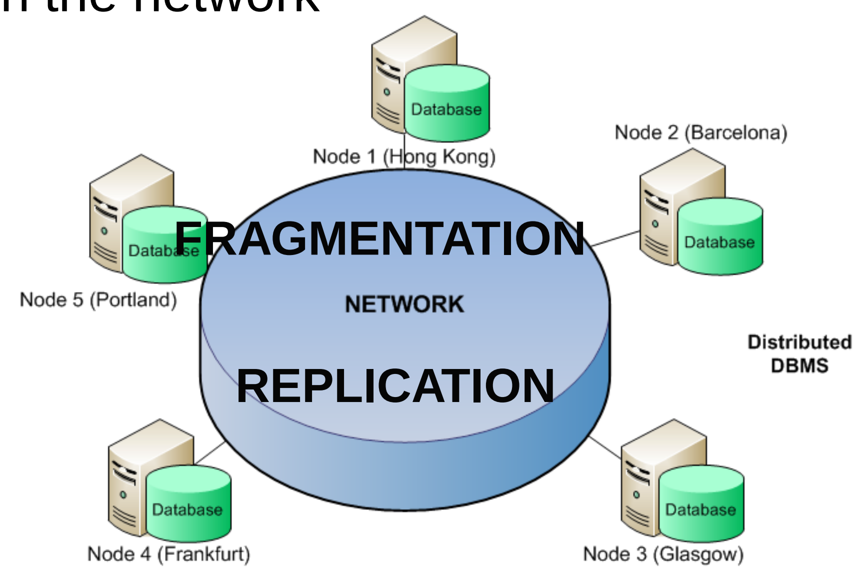
FOUNDATIONS

DISTRIBUTED DATABASES

Distributed Database

- A distributed database (DDB) is a database where <u>data management</u> is distributed over several nodes in a network.
  - Each node is a database itself
    - Potential heterogeneity
  - Nodes communicate through the network





NOSQL and Distributed Databases

I. Distributed DB design
  - Node distribution
  - Data fragments
  - Data replication

II. Distributed DB catalog
  - Fragmentation trade-off: Where to place the DB catalog
    - Global or local for each node
    - Centralized in a single node or distributed
    - Single-copy vs. Multi-copy

III. Distributed query processing
  - Data distribution / replication
  - Communication overhead

IV. Distributed transaction management
  - How to enforce (or not) the ACID properties
    - Replication trade-off: Queries vs. Data consistency between replicas (updates)
    - Distributed recovery system
    - Distributed concurrency control system

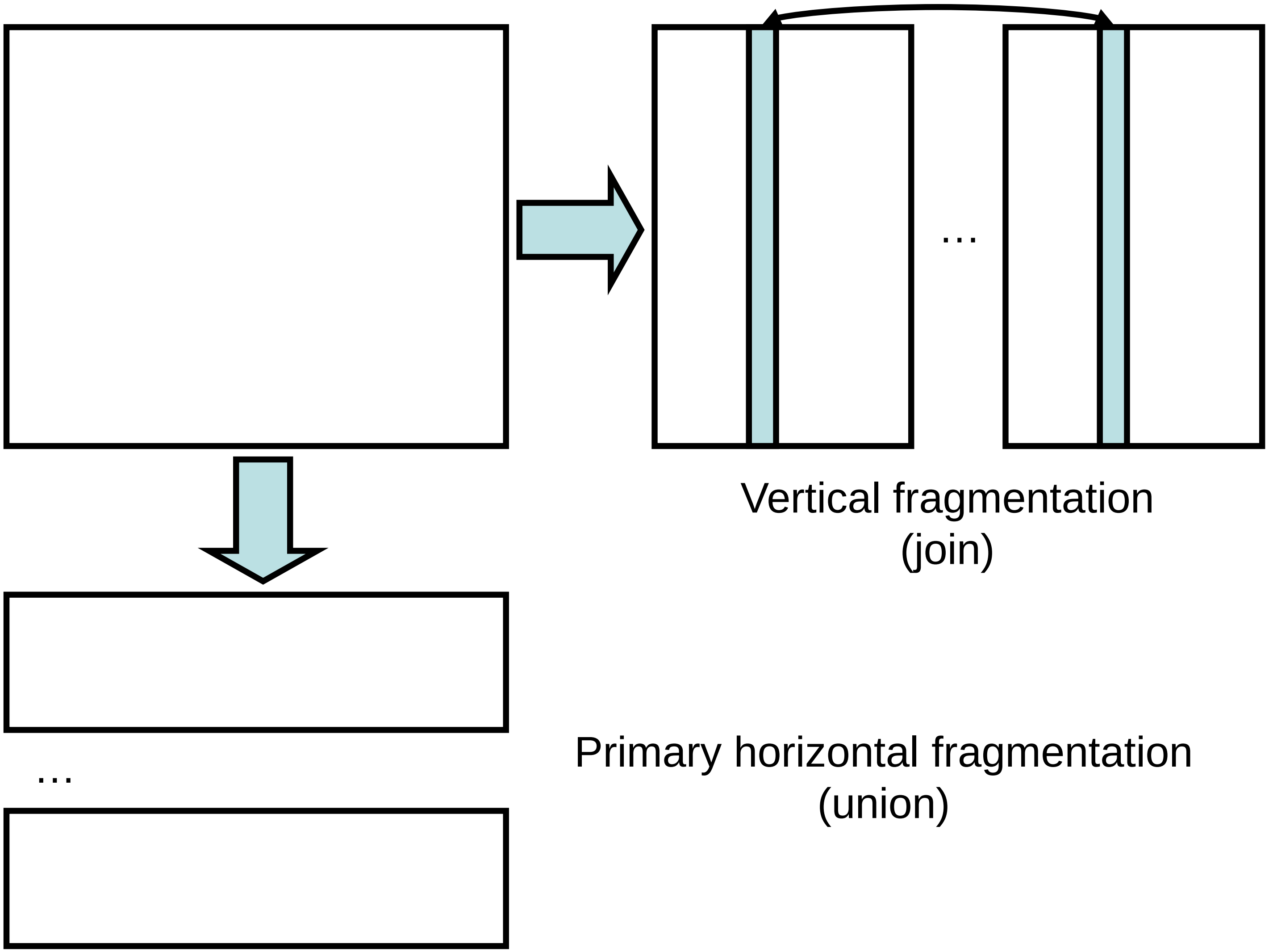
Challenge I: DDB Design

- Given a DB and its workload, how should the DB be split and allocated to sites as to optimize certain objective functions
  - Minimize resource consumption for query processing
- Two main issues:
  - Data fragmentation
  - Data allocation (data replication)

Data Fragmentation

- Fragmentation of a relation is useful for several reasons…
  - An application typically accesses only a subset of relations
  - Different subsets are (naturally) needed at different sites
  - The degree of concurrency is enhanced
    - Facilitates parallelism
  - Fragments likely to be used jointly can be colocated to minimize communication overhead
- However…
  - May lead to poorer performance when multiple fragments need to be joined
  - It becomes more costly to enforce the dependency between attributes in different fragments

Kinds of Fragmentation



Activity: Fragmentation Strategies

- *Objective: Understand the principles behind fragmentation*
- *Tasks:*
  1. *(10') By pairs, answer the following questions:*
     I. *Briefly explain which fragmentation strategy has been applied for the database below.*
     II. *Can you think of the pros and cons you may earn with each solution?*
  2. *(5') Discussion*

Global Relations

```
Kids(kidId, name, address, age)
Toys(toyId, name, price)
Requests(kidId, toyId, willingness)
Note that requests(kidId) is a foreign key to kids(kidId) and
similarly, requests(toyId) refers to toys(toyId).
```

Fragments

```
K1= Kids[kidId, name]
K2= Kids[kidId, address, age]
T1= Toys(price >= 150)
T2= Toys(price < 150)
KT1 = Requests ⋉ T1
KT2 = Requests ⋉ T2
```

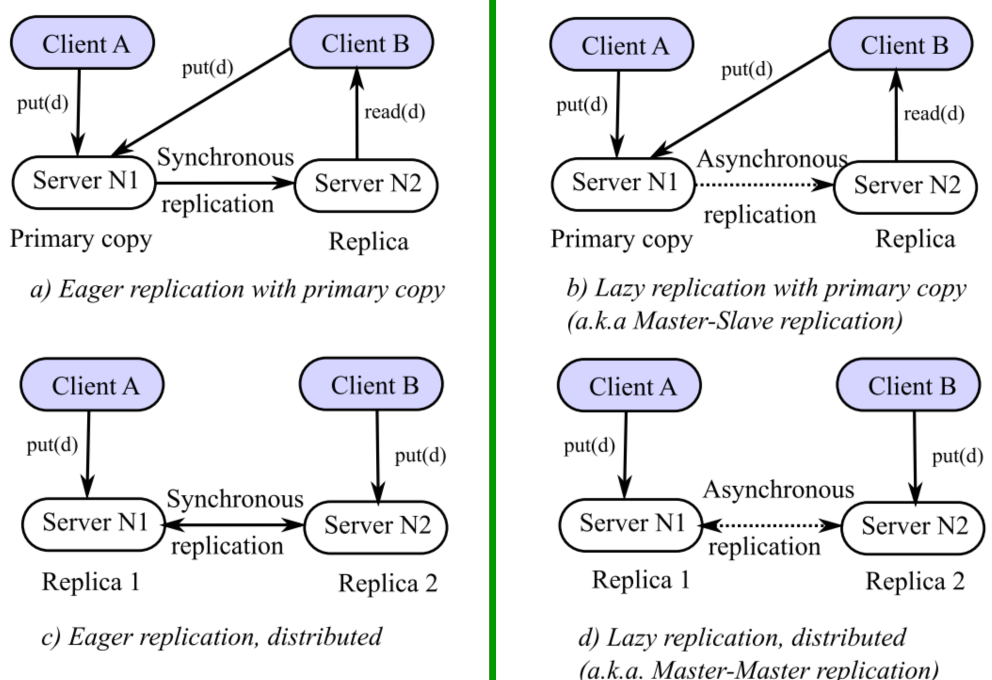
Decide Data Allocation

- Given a set of <u>fragments</u>, a set of <u>sites</u> on which a number of <u>applications</u> are running, **allocate** each fragment such that some <u>optimization criterion</u> is met (subject to certain <u>constraints</u>)
- It is known to be a NP-hard problem
  - The optimal solution depends on many factors
    - Location in which the query originates
    - The query processing strategies (e.g., join methods)
  - Furthermore, in a dynamic environment the workload and access pattern may change
- The problem is typically simplified with certain assumptions (e.g., only communication cost considered)
- Typical approaches build *cost models* and any optimization algorithm can be adapted to solve it
  - Heuristics are also available: (e.g., best-fit for non-replicated fragments)
  - Sub-optimal solutions

Manage Data Replication

- Replicating fragments improves the system throughput but raises some other issues:
  - Consistency
  - Update performance
- Most used replication protocols
  - Eager – Lazy replication
  - Primary – Secondary versioning

Strong Consistency



a) Eager replication with primary copy

b) Lazy replication with primary copy (a.k.a Master-Slave replication)

c) Eager replication, distributed

d) Lazy replication, distributed (a.k.a. Master-Master replication)

Eventually Consistent

Activity: Data Replication Issues

- *Objective: Understand the consequences behind each data replication strategy*
- *Tasks:*
    1. *(10') By pairs, answer the following questions:*
        - *I. Discuss the questions below with your peer*
        - *II. What is the most important feature for each scenario?*
    2. *(5') Discussion*

- You are a customer using an e-commerce based on heavy replication (e.g., Amazon):
    - Show a database replication strategy (e.g., sketch it) where you buy an item, but this item does not appear in your basket.
    - You reload the page: the item appears. What happened?
    - You delete an item from your command, and add another one: the basket shows both items. What happened?
    - Will the situation change if you reload the page?

What About the System Scalability?

- How do we define “Scalability”? And “Elasticity”?
    - 3 minutes to think of it!

The Universal Scalability Law (I)

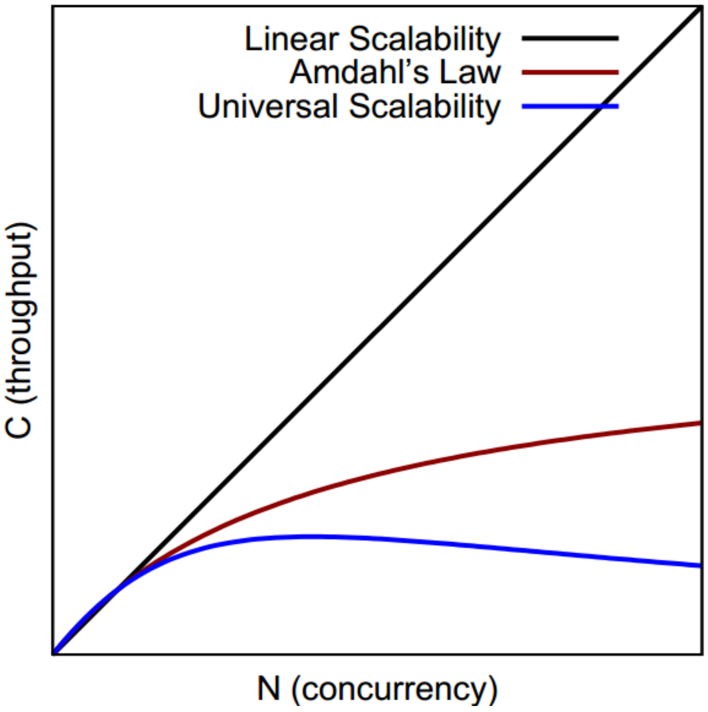
- It can model both Sw or Hw scalability
- The USL is defined as follows:

$$C(N) = \frac{N}{1 + \sigma(N-1) + \kappa N(N-1)}$$

  - C: System's capacity (i.e., throughput) improvement
    - Improvement of queries per second
  - N: System's concurrency
    - (Sw): Number of users / processes active
    - (Hw): Number of CPUs
  - σ: System's contention. Performance degradation due to serial instead of parallel processing
  - κ: System's consistency delay (aka coherency delay). Extra work needed to keep synchronized shared data (i.e., inter-process communication)

The Universal Scalability Law (II)

- If both $\sigma = 0$ and $\kappa = 0$, we obtain linear scalability
- If $\kappa = 0$, it simplifies to Amdahl's law



The USL at Work

- Method:
  - [Step 1] Empirical analysis: Compute C (throughput) for different values of N (concurrency)
  - [Step 2] Perform statistical regression against gathered data (needs some data cooking first)
  - [Step 3] Reverse the transformation to find the σ (contention) and κ (consistency delay) parameters
- How to apply this method step by step:

http://www.percona.com/files/white-papers/forecasting-mysql-scalability.pdf

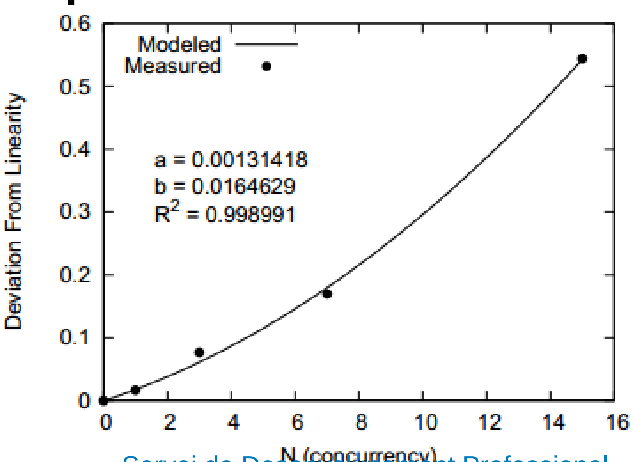
The USL at Work: Example

- System's Setting
  - Percona's MySQL Server with XtraDB
  - Cisco UCS server (2 processors, each with 6 cores and each core can run two threads: 24 threads)
  - 384GB memory

Step 1:

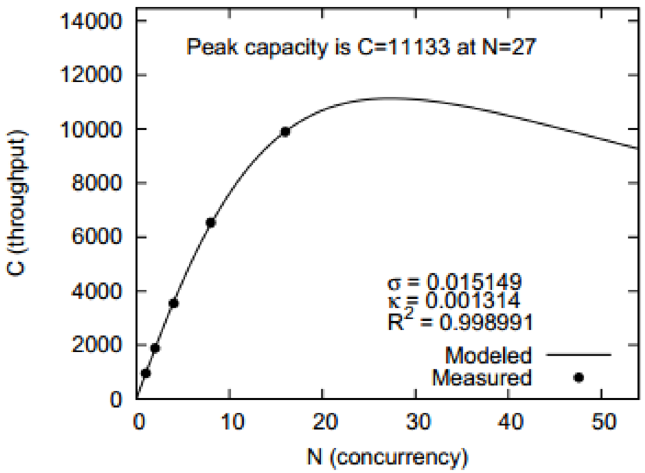
Concurrency (N)	Throughput (C)
1	955.16
2	1878.91
4	3548.68
8	6531.08
16	9897.24

Step 2:



Points are fit in a second-order polynomial: ax^2+bx+0, and a and b are computed

Step 3:



σ and κ are next computed from a and b. Next, we apply the USL formula

Measuring Scalability

- Ideally, scalability should be linear
- Scalability is normally measured in terms of speed-up and scale-up
  - Speed-up: Measures performance when adding Hw for a constant problem size
    - Linear speed-up means that N sites solve in T/N time, a problem solved in T time by 1 site
  - Scale-up: Measures performance when the problem size is altered with resources
    - Linear scale-up means that N sites solve a problem N*T times bigger in the same time 1 site solves the same problem in T time
- The USL shows that linear scalability is hardly achievable
  - σ (contention) could be avoided (i.e., σ = 0) if our code has no serial chunks (everything parallelizable)
  - κ (consistency delay) could be avoided (i.e., κ = 0) if replicas can be synchronized without sending messages

Challenge II: Global Catalog

- Centralized version (@master)
  - Accessing it is a bottleneck
    - Single-point failure
      - May add a mirror
    - Poorer performance
- Distributed version (several *masters*)
  - Replica synchronization
    - Potential inconsistencies

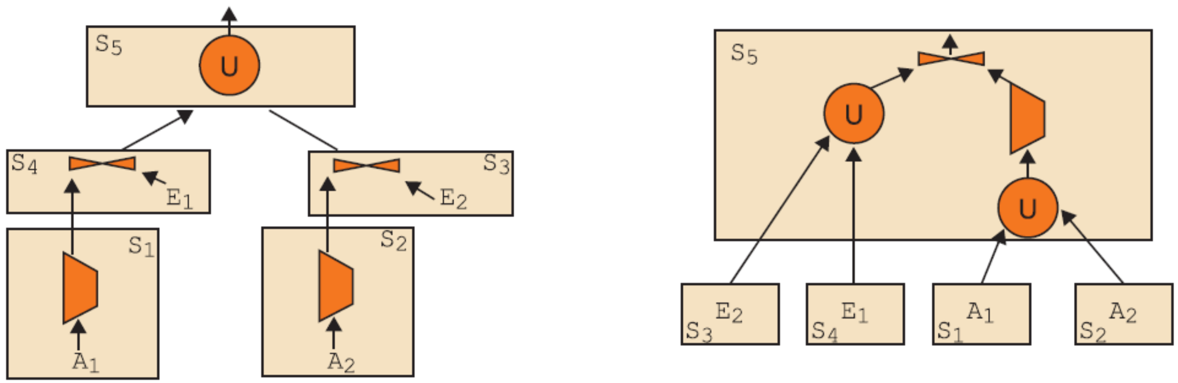
Challenge III: Distributed Query Processing

- Communication cost (data shipping)
  - Not that critical for LAN networks
    - Assuming high enough I/O cost
- Fragmentation / Replication
  - Metadata and statistics about fragments (and replicas) in the global catalog
- Join Optimization
  - Joins order
  - Semijoin strategy
- How to decide the execution plan
  - Who executes what
  - Exploit parallelism (!)

Activity: Distributed Query Processing

- Objective: Recognize the difficulties and opportunities behind distributed query processing
- Tasks:
  1. (10') By pairs, answer the following questions:
     I. What are the main differences between these two distributed access plans?
     II. Under which assumptions is one or the other better?
     III. List the new tasks a distributed query optimizer must consider with regard to a centralized version
  2. (5') Discussion

```
SELECT *
FROM employee e, assignedTo a
WHARE e.#emp=a.#emp AND
a.responsability= 'manager';
```

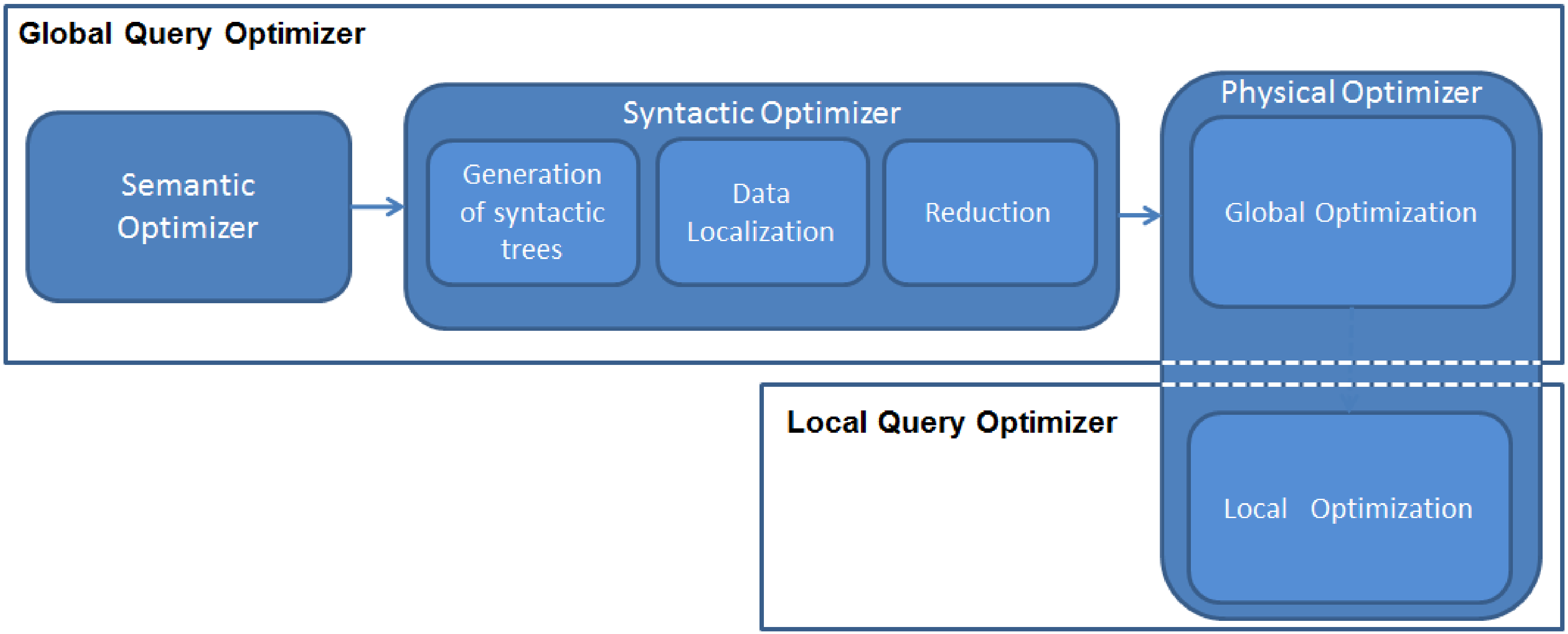


Access Plan A　　　　Acces Plan B

AssignedTo (#emp,#proj, responsibility, fullTime)
- S_1: A_1 = AssignedTo(#emp≤'E3')
- S_2: A_2 = AssignedTo(#emp>'E3')

Employee (#emp, empName, degree)
- S_3: E_2 = Employee(#emp>'E3')
- S_4: E_1 = Employee(#emp≤'E3')

Phases of Distributed Query Processing

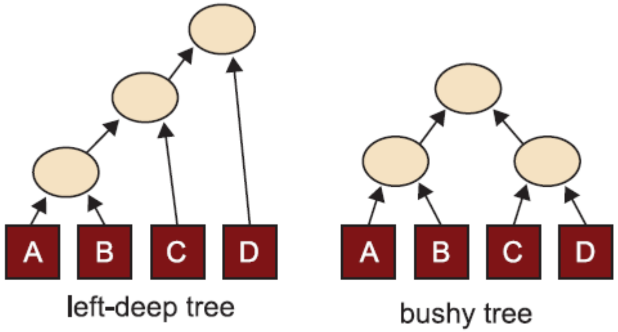


Physical Optimizer

- Transforms an internal query representation into an efficient plan
  - Replaces the logical query operators by specific algorithms (plan operators) and access methods
  - Decides in which order to execute them
- This is done by…
  - Enumerating alternative but equivalent *plans*
    - Dataflow diagram that pipes data through a graph of query operators
  - Estimating their costs
  - Searching for the best solution
    - Using available statistics regarding the physical state of the system

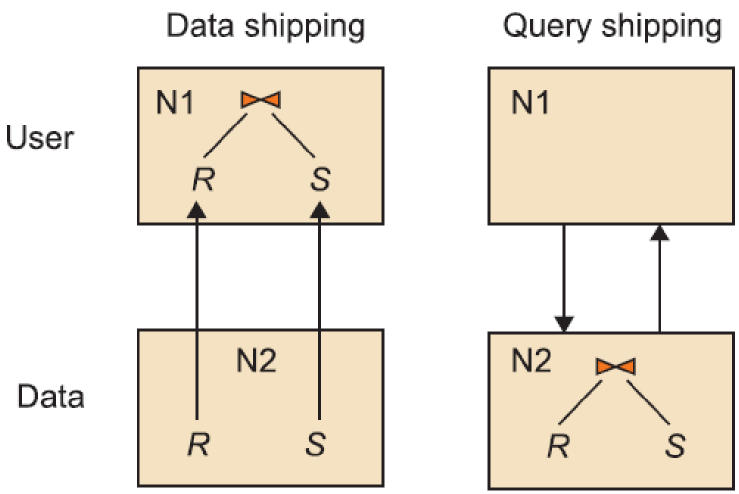
Global Physical Optimizer

- Generation of Execution Alternatives
  - Ordering
    - Left or right deep trees
    - Bushy trees
  - Site selection (exploit **DATA LOCATION**)
    - Comparing size of the relations
    - More difficult for joins (multi-way joins)
    - Size of the intermediate joins must be considered
  - Algorithms to process the query operators
    - **Parallelism (!)**



Site Selection

- Data shipping
  - The data is retrieved from the stored site to the site executing the query
    - Avoid bottlenecks on frequently used data
- Query shipping
  - The evaluation of the query is delegated to the site where it is stored
    - To avoid transferring large amount of data
- Hybrid strategy



The Semi-join Strategy (for joins)

- Different strategies according to the operator
  - Semi-join strategy

- Example:

$$R \bowtie S \quad \begin{cases} (R \ltimes S) \bowtie S & \text{, better if } B_R > B_{S[Jattr]} + B_{R \ltimes S} \\ (S \ltimes R) \bowtie R & \text{, better if } B_S > B_{R[Jattr]} + B_{R \rtimes S} \\ (R \ltimes S) \bowtie (S \ltimes R) & \text{, better if ...} \end{cases}$$

$S_1 \qquad S_2$

- The semi-join strategy vs. Ordering joins
  - Reduces the communication overhead
  - Performs more operations over smaller operators
  - To consider if we have a small join selectivity factor
  - Needed statistics might not be available

Parallel Query Processing

- Employ parallel hardware effectively (i.e., reduce the response time)
  - Process pieces in different processors
  - Serial algorithms adapted to multi-thread environments
  - Divide input data set into disjoint subsets
- May hurt overall execution time (i.e., throughput)
  - Ideally linear speed-up
    - Additional hardware for a constant problem size
      - Addition of computing power should yield proportional increase in performance
        - » N nodes should solve the problem in 1/N time
  - Ideally linear scale-up
    - Problem size is altered with the resources
      - Sustained performance for a linear increase in both size and workload, and number of nodes
        - » N nodes should solve a problem N times bigger in the same time

Kinds of Parallelism

- Inter-query
- Intra-query
  - Intra-operator
    - Unary
      - Static partitioning
    - Binary
      - Static or dynamic partitioning
  - Inter-operator
    - Independent
    - Pipelined
      - Demand driven (pull)
      - Producer driven (push)

Choosing the Best Execution Plan

- Response Time
  - Time needed to execute a query (user's clock)
  - Benefits from parallelism
    - Operations divided into N operations
- Total Cost Model
  - Sum of local cost and communication cost
    - Local cost
      - Cost of central unit processing (#cycles),
      - Unit cost of I/O operation (#I/O ops)
    - Communication cost
      - Commonly assumed it is linear in the number of bytes transmitted
      - Cost of initiating a message and sending a message (#messages)
      - Cost of transmitting one byte (#bytes)
  - Knowledge required
    - Size of elementary data units processed
    - Selectivity of operations to estimate intermediate results
  - Does not account the usage of parallelisms (!)
- Hybrid solutions

An Example of Model Cost

- Parameters:
  - Local processing:
    - Average CPU time to process an instance (T_{cpu})
    - Number of instances processed (#inst)
    - I/O time per operation ($T_{I/O}$)
    - Number of I/O operations (#I/Os)
  - Global processing:
    - Message time (T_{Msg})
    - Number of messages issued (#msgs)
    - Transfer time (send a byte from one site to another) (T_{TR})
    - Number of bytes transferred (#bytes)
- It could also be expressed in terms of packets
- Calculations:

$$\text{Resources} = T_{cpu} * \#inst + T_{I/O} * \#I/Os + T_{Msg} * \#msgs + T_{TR} * \#bytes$$

$$\text{Respose Time} = T_{cpu} * seq_{\#inst} + T_{I/O} * seq_{\#I/Os} + T_{Msg} * seq_{\#msgs} + T_{TR} * seq_{\#bytes}$$

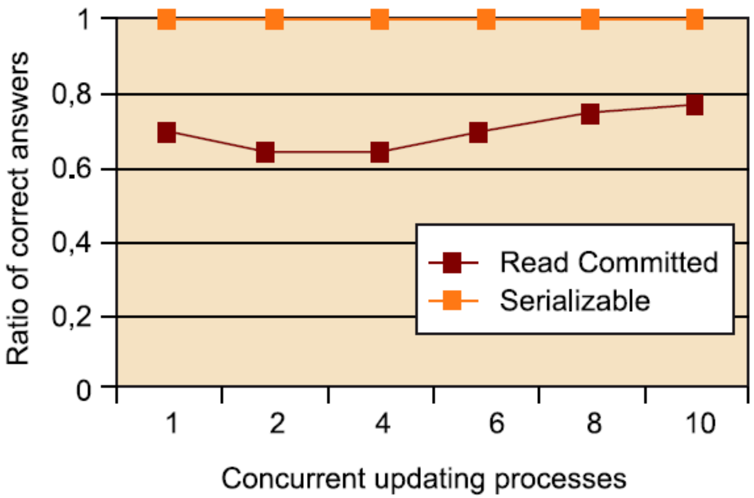
Challenge IV: Distributed Tx Management

- ACID properties are not always necessary
  - All can be relaxed
- Relaxing Consistency and Durability
  - Entails data loss
  - Save synchronization time
- Relaxing Atomicity and Isolation
  - Generate interferences
  - Save locks and contention

Trade-Off: Performance Vs. Consistency

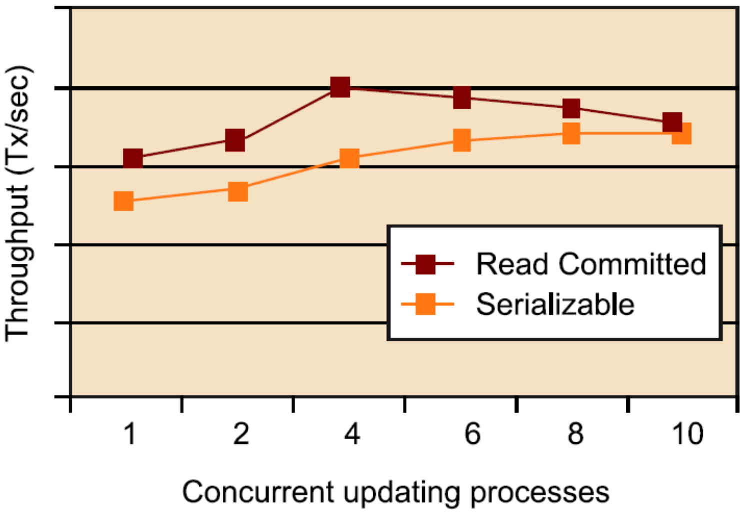
Consistency
(Ratio of correct answers)

Percentage of correct results depending on the number of concurrent transactions



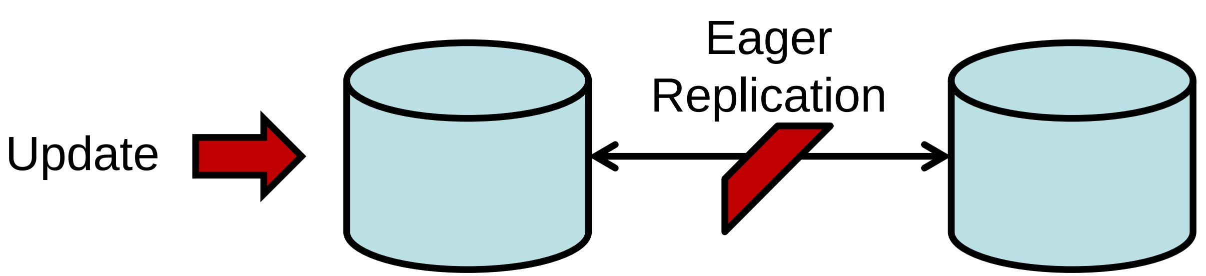
Performance
(System throughput)

Throughput (transactions per second) depending on the number of concurrent transactions.



CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P)
- Example:

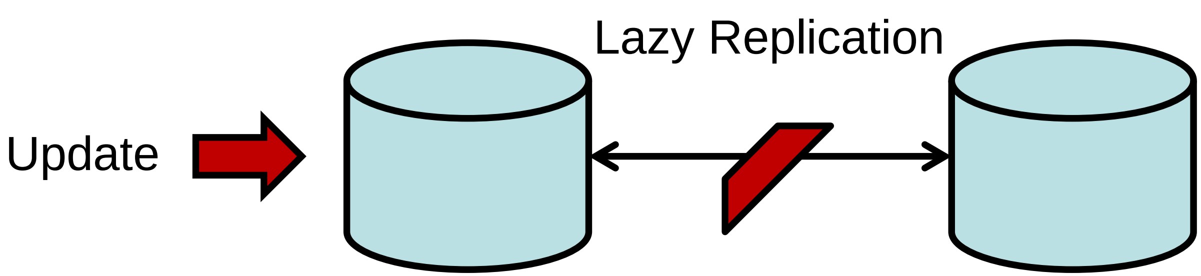


ACID

C -> OK
A -> NO
P -> OK

CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:

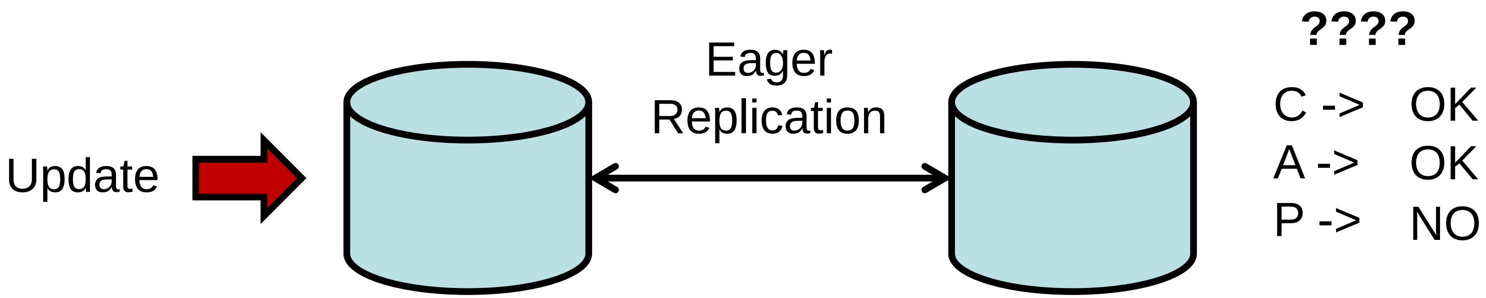


BASE

C -> NO
A -> OK
P -> OK

CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



CAP Theorem Revisited

- The CAP theorem is not about choosing two out of the three **forever and ever**
  - Distributed systems are not always partitioned
- Without partitions: CA
- Otherwise…
  - Detect a partition
    - Normally by means of latency (time-bound connection)
  - Enter an explicit partition mode limiting some operations choosing either:
    - CP (i.e., ACID by means of e.g., 2PCP or PAXOS) or,
      - If a partition is detected, the operation is aborted
    - AP (i.e., BASE)
      - The operation goes on and we will tackle this next
  - If AP was chosen, enter a recovery process commonly known as *partition recovery* (e.g., compensate mistakes and get rid of inconsistencies introduced)
    - Achieve consistency: Roll-back to consistent state and apply ops in a deterministic way (e.g., using time-stamps)
      - Reduce complexity by only allowing certain operations (e.g., Google Docs)
      - Commutative operations (concatenate logs, sort and execute them)
    - Repair mistakes: Restore invariants violated
      - Last writer wins

FOUNDATIONS

ON THE NEED OF A NEW ARCHITECTURE

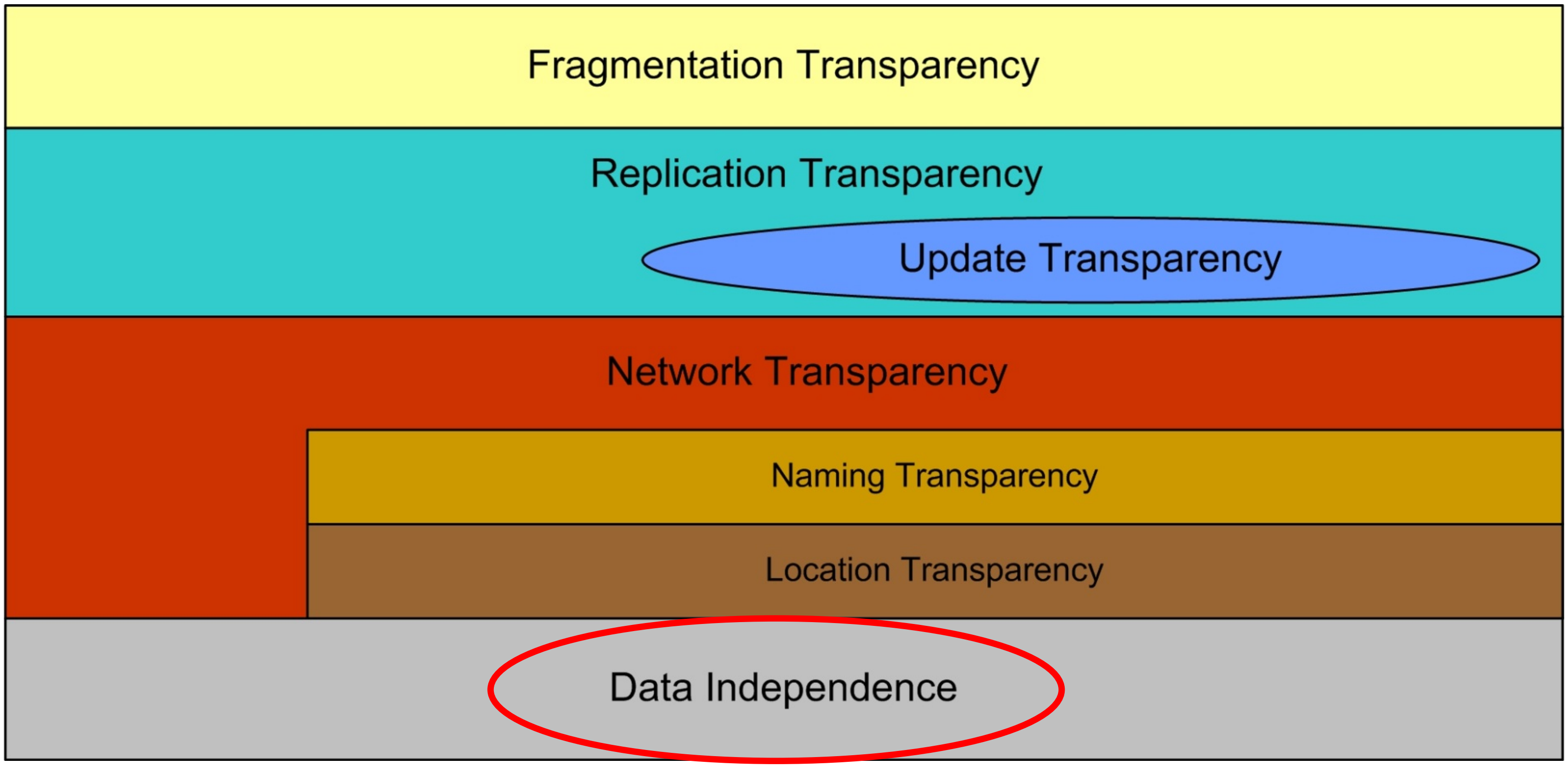
Challenge V: On the Need of a New Architecture

- Distribution is needed to overcome the challenges presented
  - To provide scalability, efficiency (by means of parallelism), reliability / availability
    - But RDDBMS do not meet them (RDDBMS bottlenecks)
- **CAP theorem formulation**: There is a trade-off in distributed systems; either availability or consistency can be always guaranteed (not both)
  - RDDBMS choose consistency
  - NOSQL systems, most of the times, choose availability

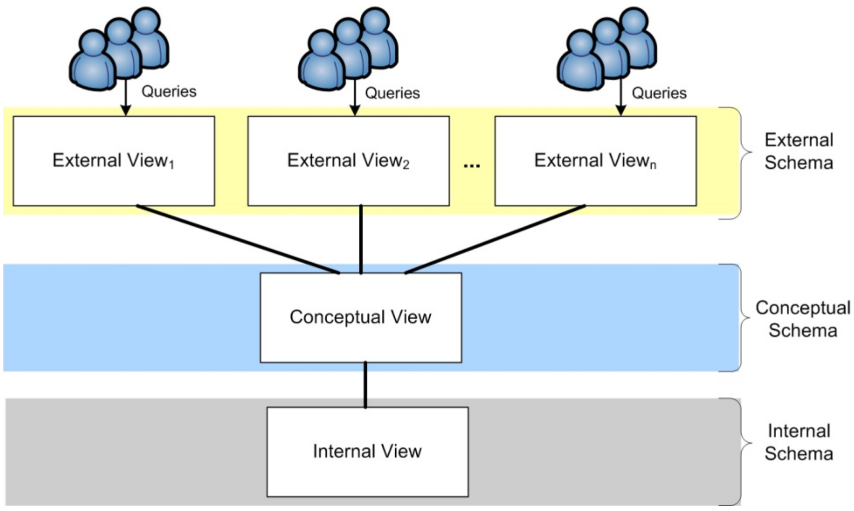
Distributed Architectures

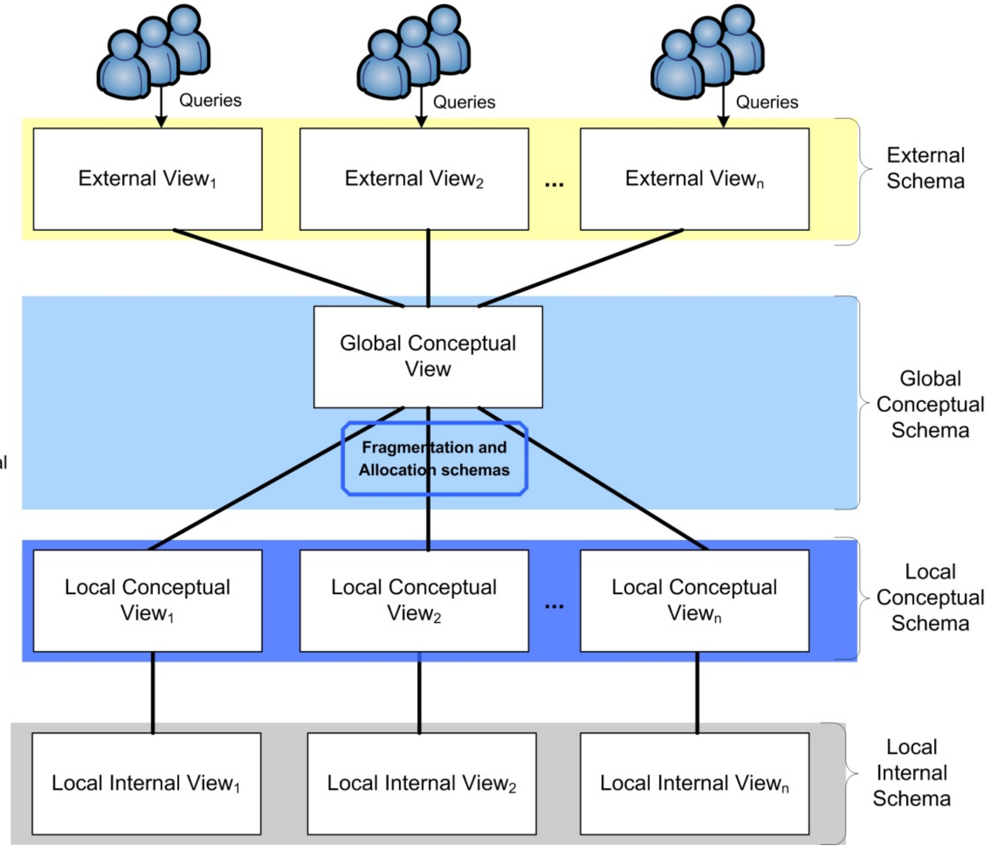
- Main objective: hide implementation (i.e., physical) details to the users
  - Data independency at the logical and physical level must be guaranteed
    - Inherited from centralized DBs (ANSI SPARC)
  - Network transparency
    - Data access must be independent regardless where data is stored
    - Each data object must have a unique name
  - Replication transparency
    - The user must not be aware of the existing replicas
  - Fragmentation transparency
    - The user must not be aware of partitioning

Distributed Architectures



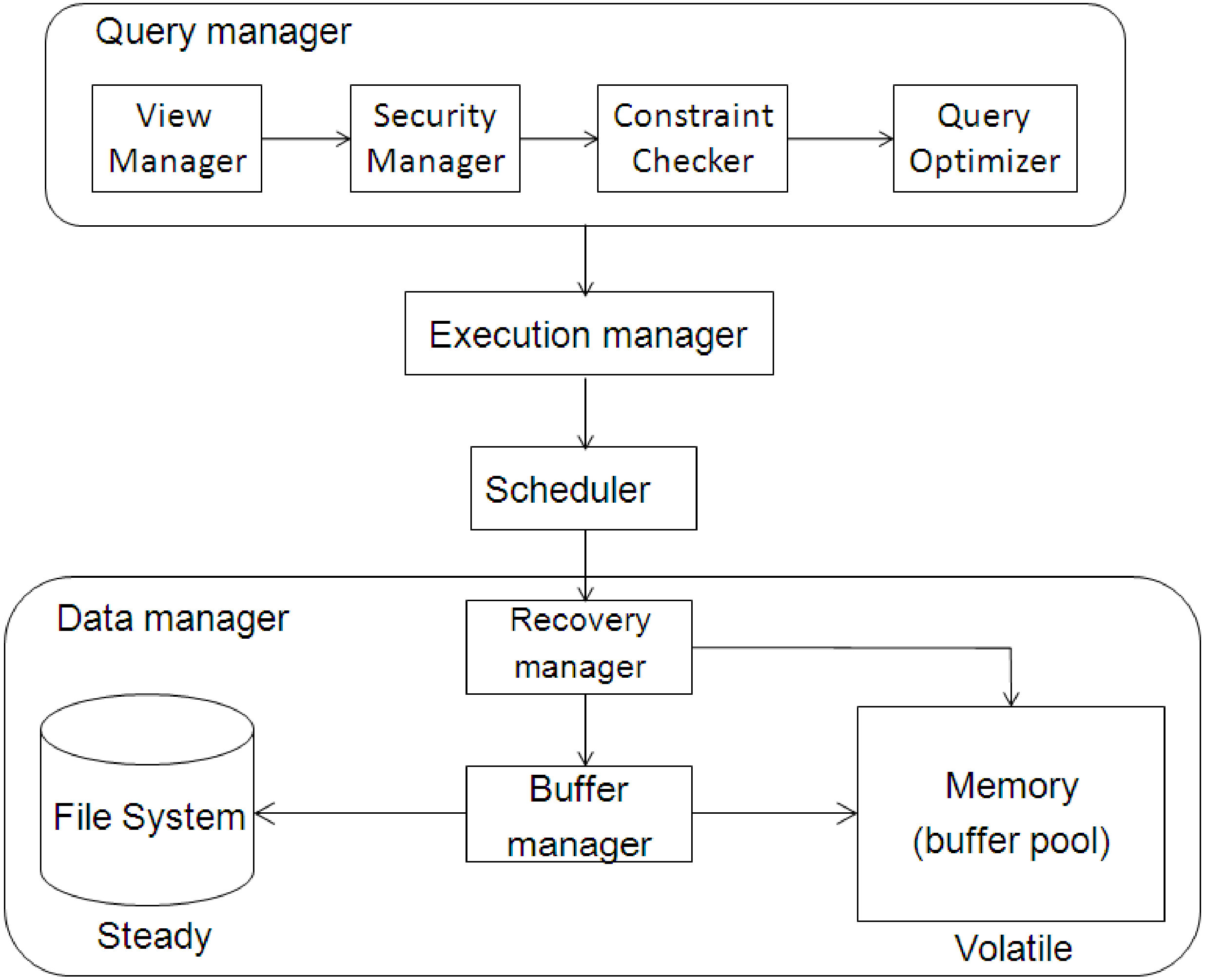
Extended ANSI-SPARC Architecture



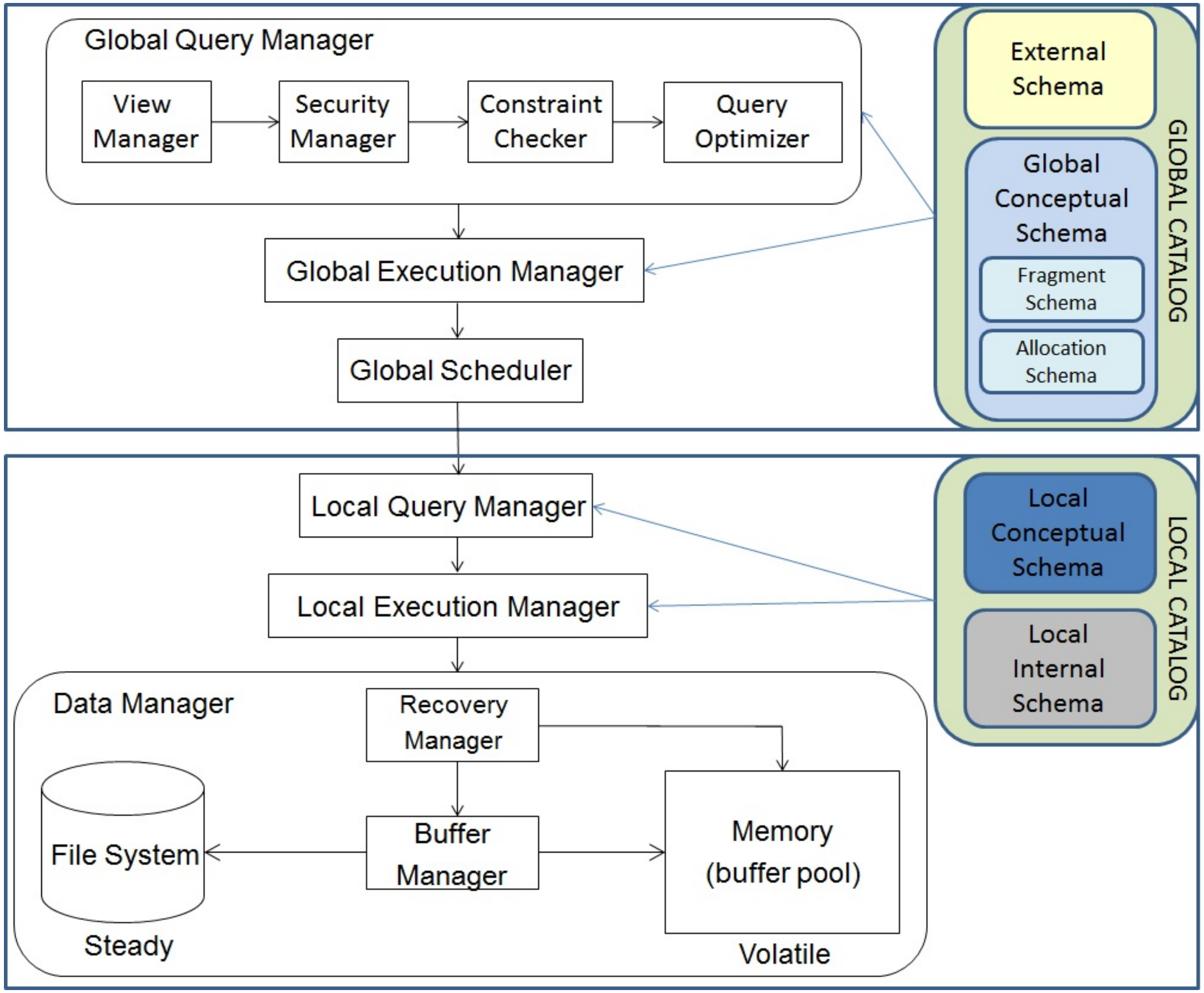


- Global catalog
  - Mappings between ESs – GCS and GCS – LCSs
- Each node has a local catalog
  - Mappings between LCS_i – IS_i

Centralized DBMS Architecture



Distributed DBMS Architecture



NewSQL: A New Architecture for OLTP

But even for distributed, massive OLTP systems RDBMS can be outperformed!

- Main memory DB
  - A DB less than 1Tb fits in memory
    - 20 nodes x 32 Gb (or more) costs less than 50,000US$
  - Undo log is in-memory and discarded on commit
- One thread systems
  - Perform incoming SQL commands to completion, without interruption
    - One transaction takes less than 1ms
  - No isolation needed
- Grid computing
  - Enjoy horizontal partitioning and parallelism
    - Add new nodes to the grid without going down
- High availability
  - Cannot wait for the recovery process
    - Multiple machines in a Peer-To-Peer configuration
- Reduce costs
  - Human costs are higher than Hw and Sw
    - An expert DBA is expensive and rare
  - Alternative is brute force
    - Automatic horizontal partitioning and replication
    - Execute queries at any replica and updates to all of them
- Optimize queries at compile time

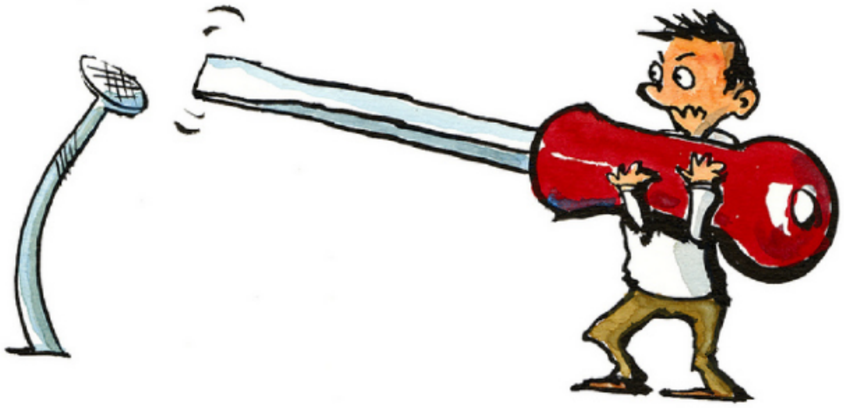
FOUNDATIONS

IMPEDANCE MISMATCH

Challenge VI: Impedance Mismatch (I)

Of hammers and nails...

The Law of the Hammer



If the only tool you have is a hammer,
everything looks like a nail.

Abraham Maslow - The Psychology of Science - 1966

Petra Selmer, Advances in Data Management 2012

Challenge VI: Impedance Mismatch (II)

The Law of the Relational Database

If the only tool you have is a relational database, everything looks like a table.

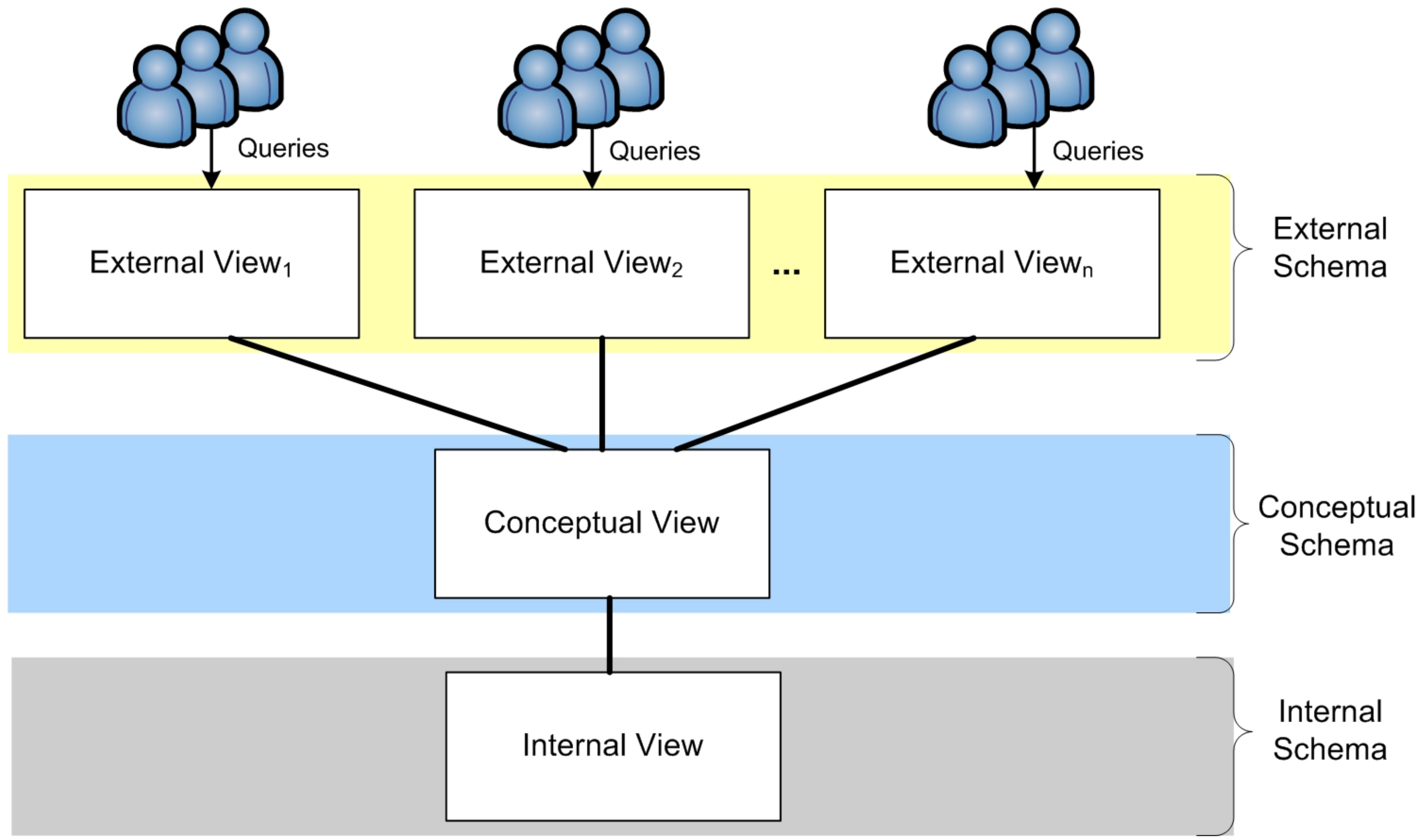
A Walk in Graph Databases - 2012

Petra Selmer, Advances in Data Management 2012

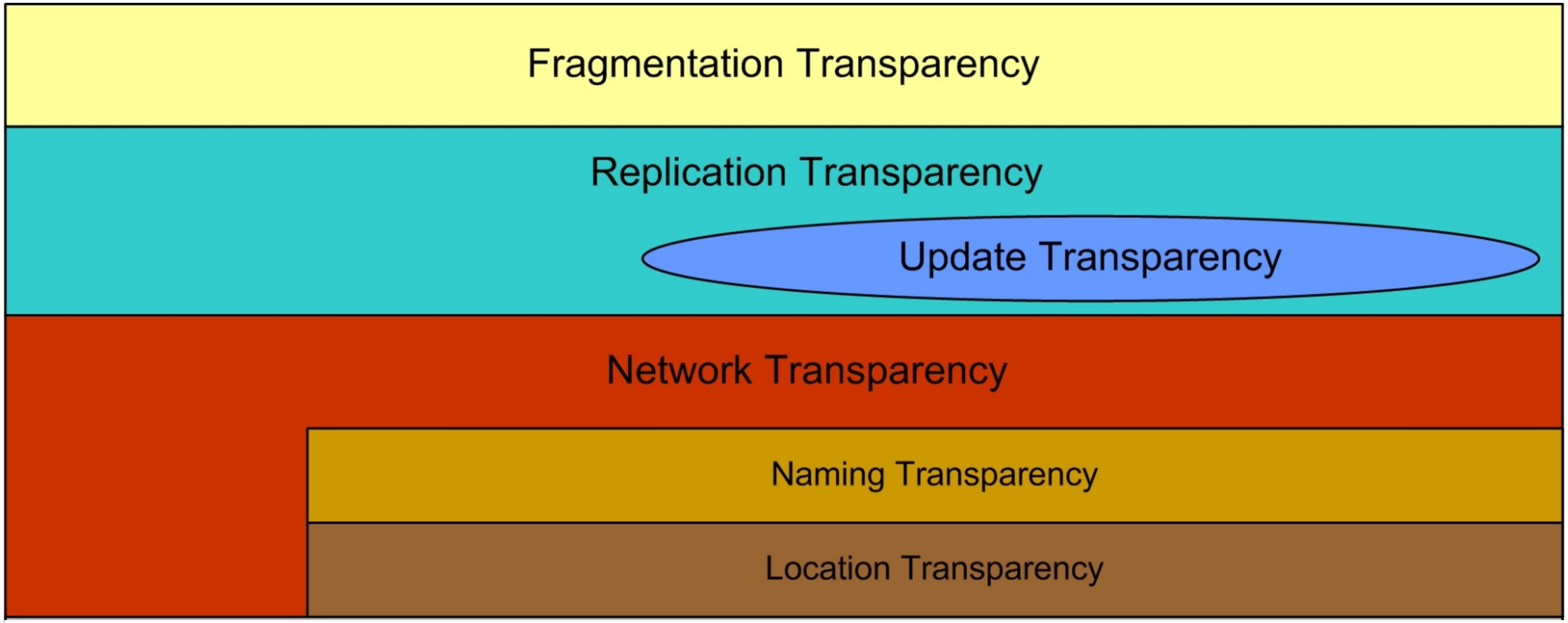
Challenge VI: Impedance Mismatch (III)

- New data models were introduced
  - Graph data model
  - Document-oriented databases
  - Key-value (~ hash tables)
  - Streams (~ vectors and matrixes)
- These *new* models lack of an explicit schema (defined by the user)
  - However, an implicit schema remains

Schemaless Databases



Distributed Architectures



FOUNDATIONS

HOMEWORK

Random Vs. Sequential Reads

- Read the following blog entry before the next lecture
  - http://www.benstopford.com/2015/04/28/elements-of-scale-composing-and-scaling-data-platforms/
- It is extremely important you fully understand the difference between random and sequential reads to fully grasp the next lectura
  - Mandatory, ALL until parallelism
  - If you are interested in fully understand this course, you are advised to read the whole entry (be prepared though, it's a bit long)

Thanks for your attention!