Distributed Database Design

Knowledge Objetives

1. Enumerate the three main tasks for distributed database design
2. Briefly define the data allocation problem
3. Outline the four main approaches for replication maintenance
4. Discuss the pros and cons of either centralizing or distributing the catalog
5. Enunciate the Universal Scalability Law
6. Name the two main problems to achieve linear scalability

Understanding Objetives

1. Reconstruct global relations from their fragments (if possible)
2. Given a specific scenario, model simple distributed databases (i.e., decide how to fragment the relations and where to allocate them according to its workload)
3. Use the USL to foresee a systems' scalability

Principles of Distributed Databases

CHALLENGES

DDBs Feats Vs. NOSQL Goals

- Reflect organizational structure
  - Centralized systems are not always possible
  - Facilitates sites autonomy
- Share data
- Expected to offer better performance
  - Data is placed at locations where it is frequently accessed
    - Communication overhead minimized
    - Parallelism can be exploited locally for query processing
- Better availability / reliability
  - In front of site failure
- Scalability
  - The “*divide and conquer*” strategy to provide elasticity and avoid bottlenecks
    - Peaks provisioning
    - Unexpected growth

DDBs Feats Vs. NOSQL Goals

- Reflect organizational structure
  - Centralized systems are not always possible
  - Facilitates sites autonomy
- Share data

$\neq$ Schemaless

- Expected to offer better performance
  - Data is placed at locations where it is frequently accessed
    - Communication overhead minimized
    - Parallelism can be exploited locally for query processing

Efficiency

- Better availability / reliability
  - In front of site failure

Reliability / availability

- Scalability
  - The "*divide and conquer*" strategy to provide elasticity and avoid bottlenecks
    - Peaks provisioning
    - Unexpected growth

Scalability

CLOUD DATABASE $\sqsubseteq$ NOSQL

NOSQL Goals

- Schemaless: No explicit schema [data structure]
- Reliability / availability: Keep delivering service even if its software or hardware components fail [recovery]
- 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]

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]

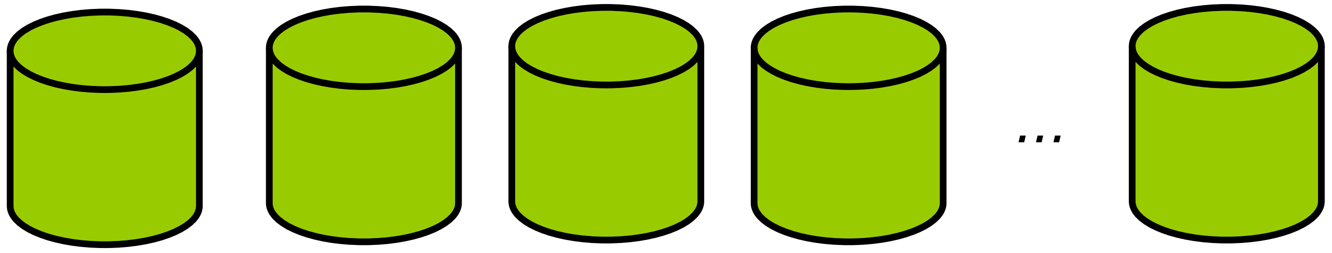
NOSQL Goals

- 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]

Challenges in Data Distribution

A	B	C	D	E

Conceptual View

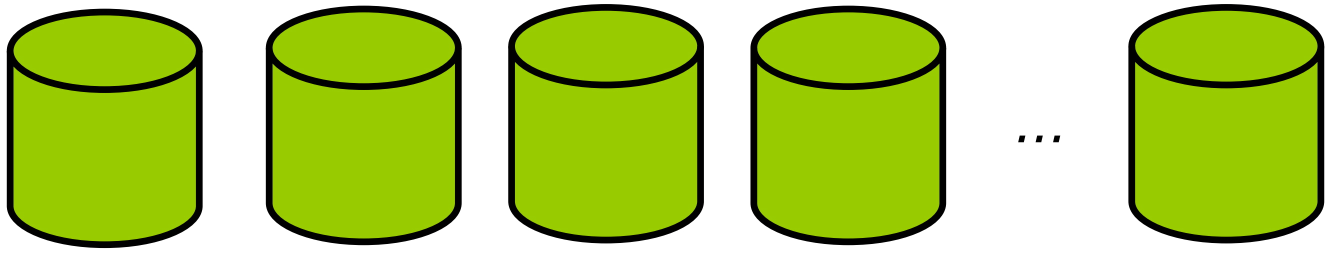


Physical View

Challenges in Data Distribution

Conceptual View

A	B	C	D	E



Physical View

Challenges in Data Distribution

I. Distributed DB design
- Node distribution
- Data fragments
- Data allocation (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 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

Challenges in Data Distribution

I. Distributed DB design
- Node distribution
- Data fragments
- Data allocation (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 the ACID properties
  - Replication trade-off: Queries vs. Data consistency between replicas (updates)
  - Distributed recovery system
  - Distributed concurrency control system

Challenge I: DDB Design

- Suitable for top-down 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)

Distributed Database design

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. *Is this fragmentation strategy complete and disjoint? Can we reconstruct the global relations? Explicit the algebraic operation you would use.*
  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]
```

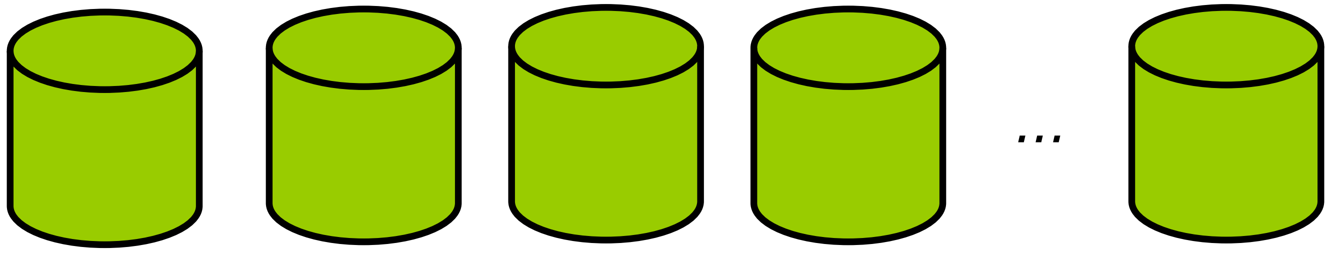
```
T1= Toys(price >= 150)
T2= Toys(price < 150)
```

```
R1 = Requests ⋉ T1
R2 = Requests ⋉ T2
```

Challenge I: Data Fragments

Conceptual View

A	B	C	D	E



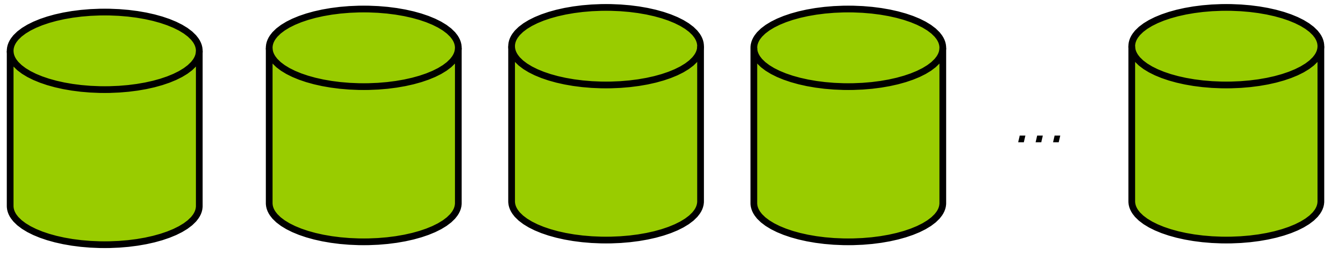
Physical View

Challenge I: Data Fragments

Conceptual View

A	B	C	D	E

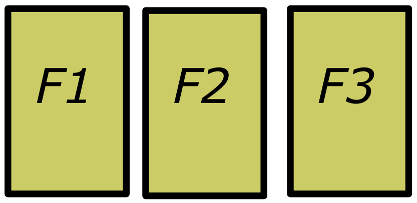
Apply a fragmentation strategy

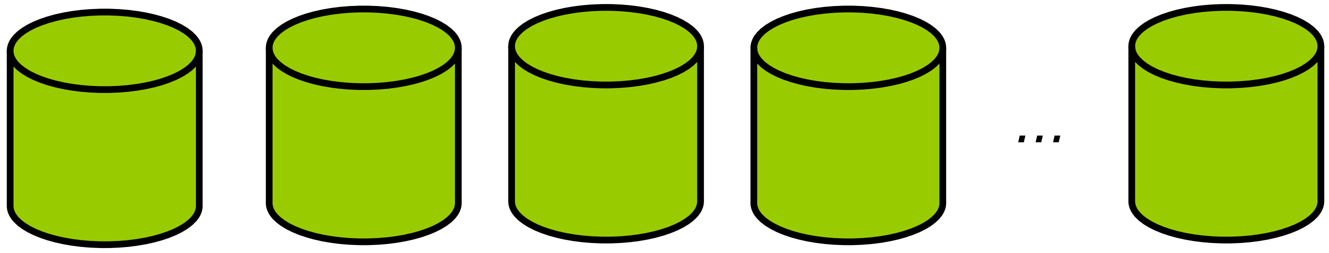


Physical View

Challenge I: Data Replication

Conceptual View





Secondary servers

Physical View

Distributed Database design

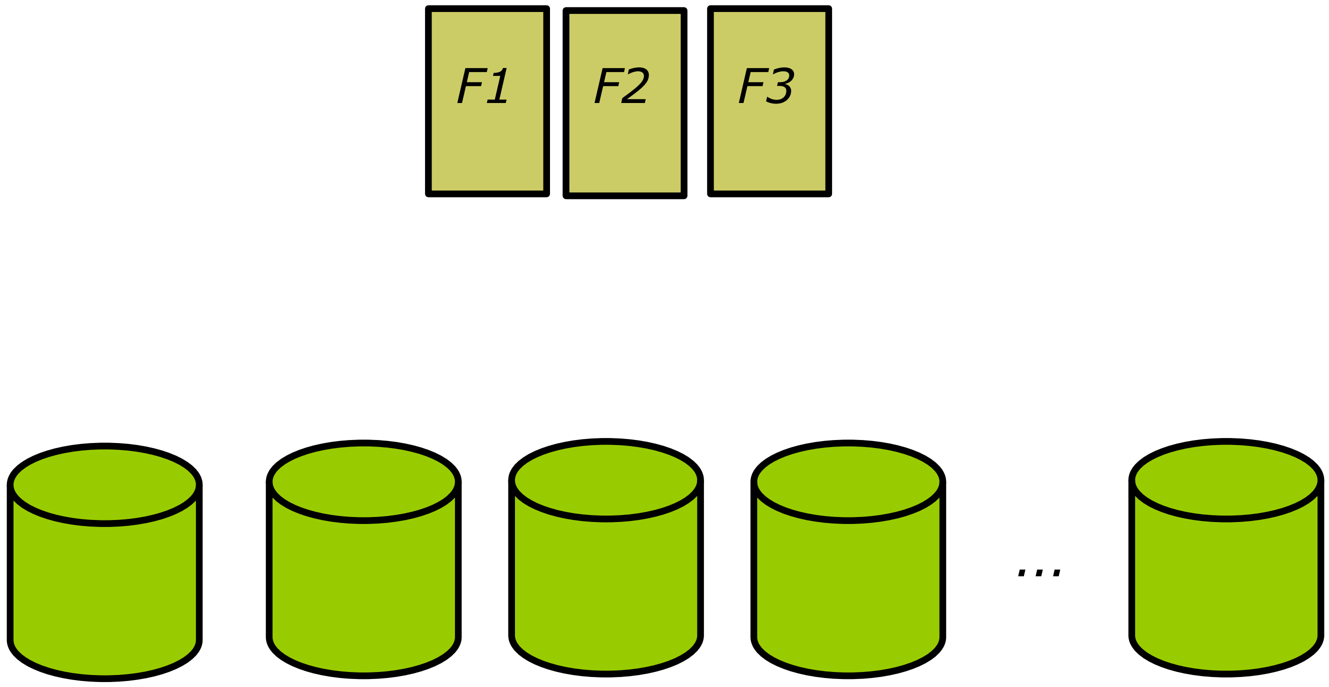
ALLOCATION (AND REPLICATION)

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
- Most advanced approaches build *cost models* and any optimization algorithm can be adapted to solve it
  - Simply the problema with assumptions (e.g., only communication cost considered)
  - Heuristics are also available: (e.g., best-fit for non-replicated fragments)
  - Sub-optimal solutions (i.e., applied per fragment individually)

Challenge I: Data Allocation

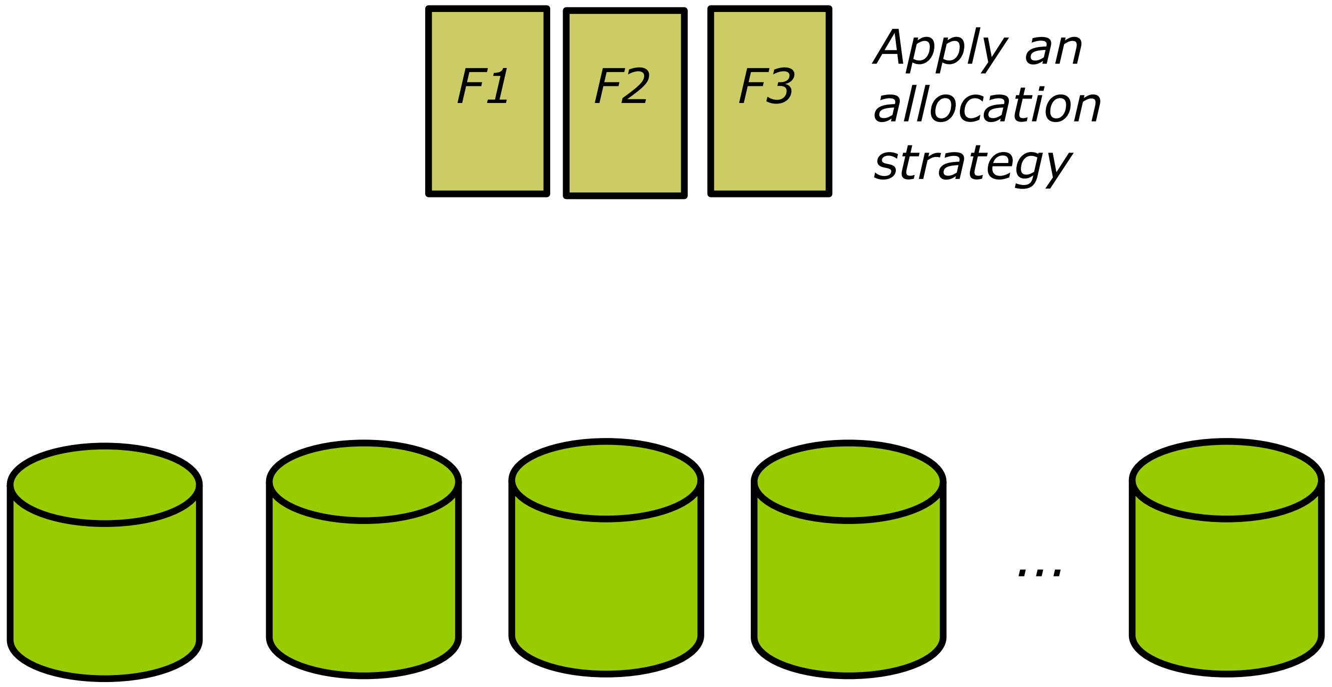
Conceptual View



Physical View

Challenge I: Data Allocation

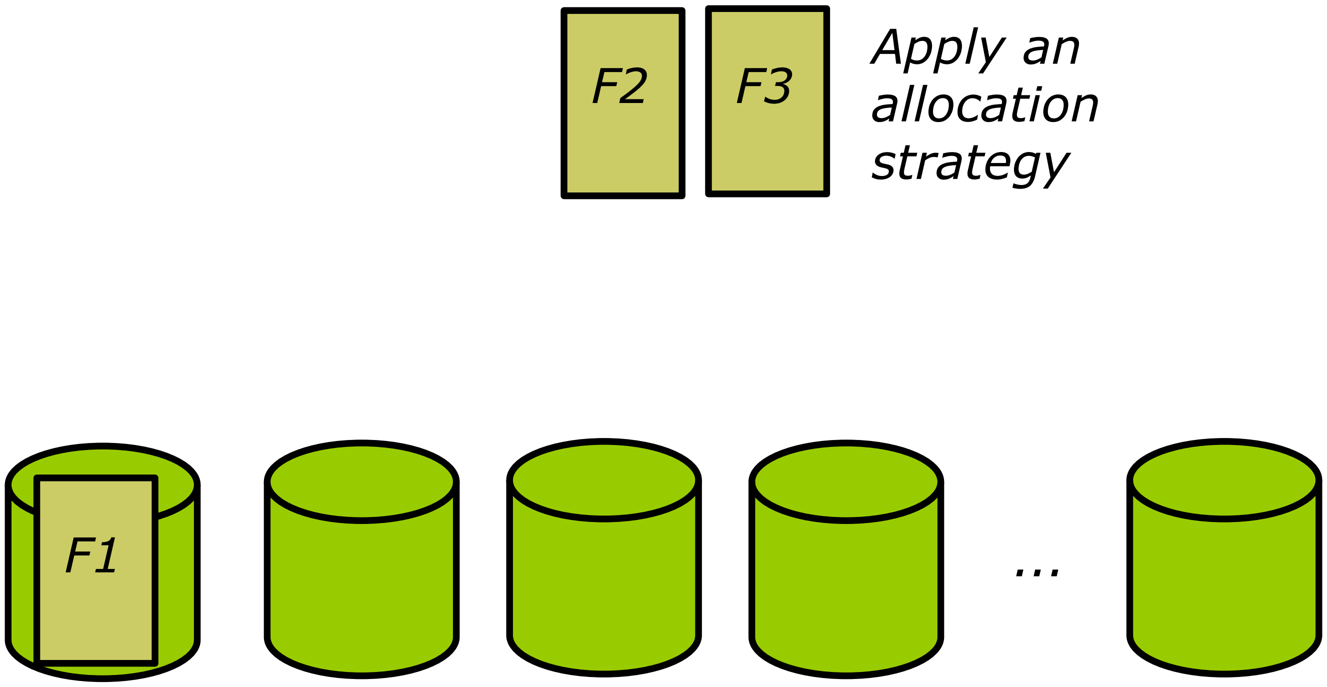
Conceptual View



Physical View

Challenge I: Data Allocation

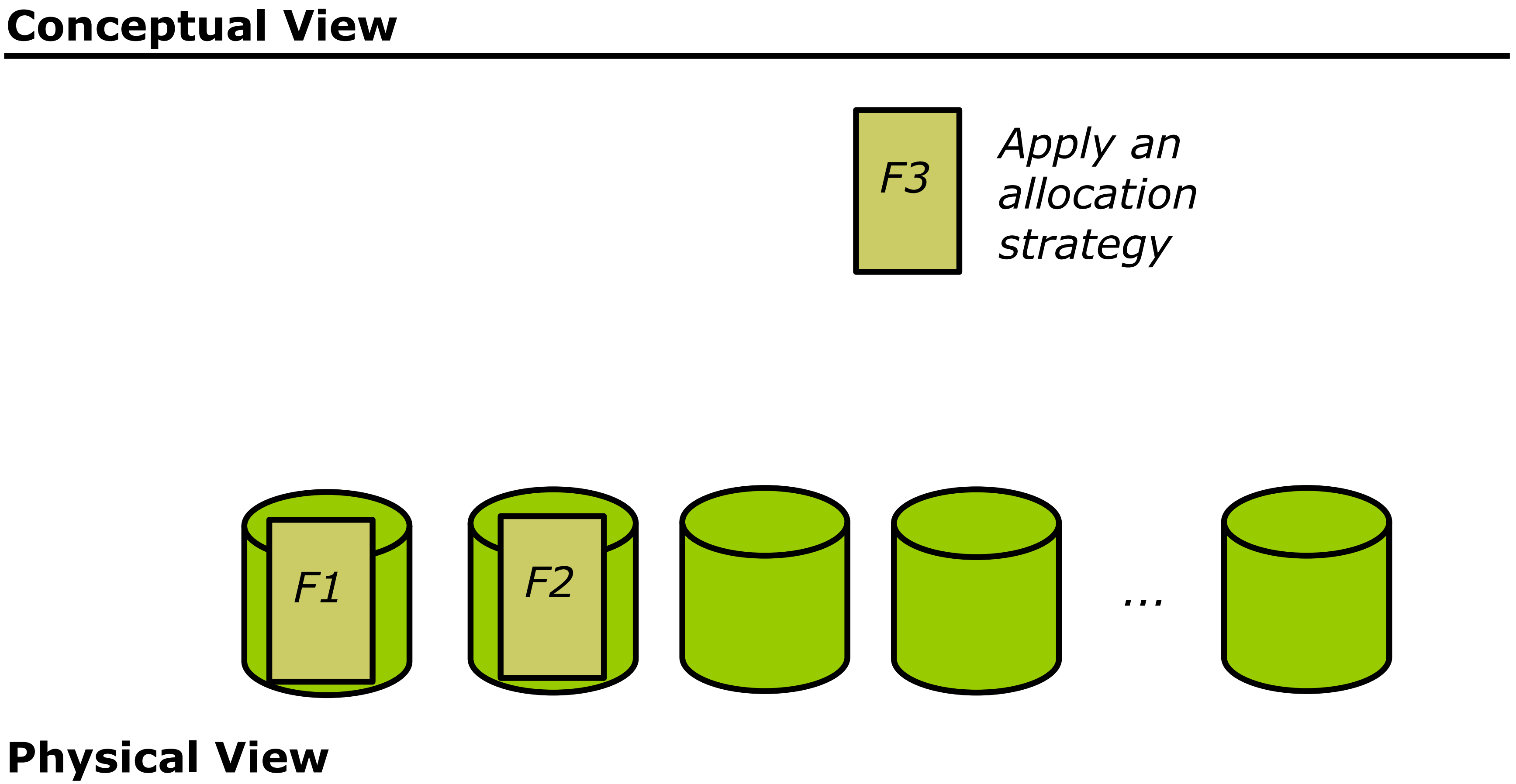
Conceptual View



Apply an allocation strategy

Physical View

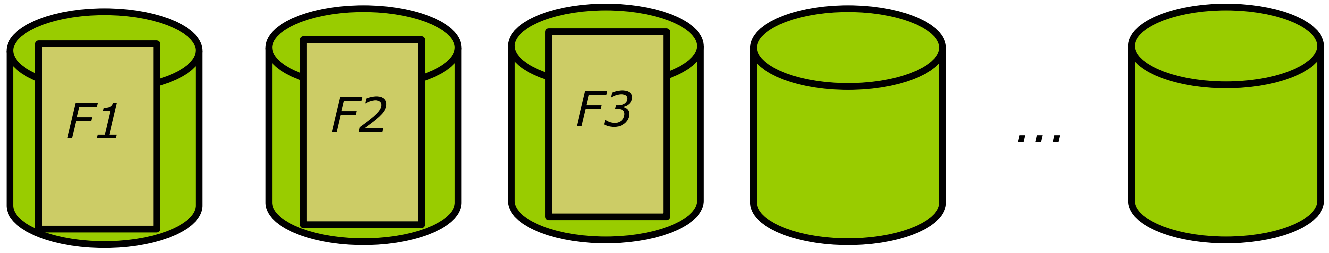
Challenge I: Data Allocation



Challenge I: Data Allocation

Conceptual View

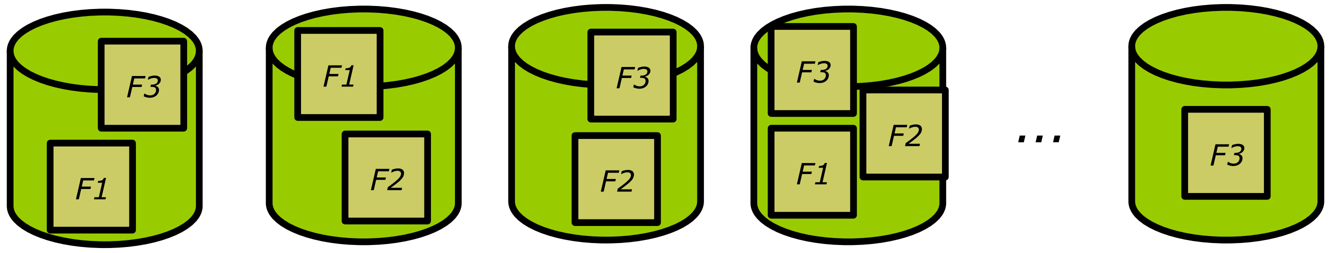
Apply an allocation strategy



Physical View

Challenge I: Data Allocation

Conceptual View

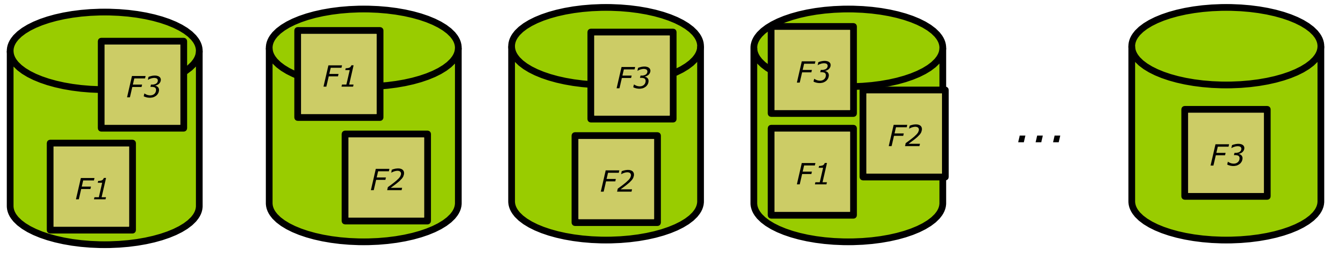


Physical View

Challenge I: Data Allocation

Conceptual View

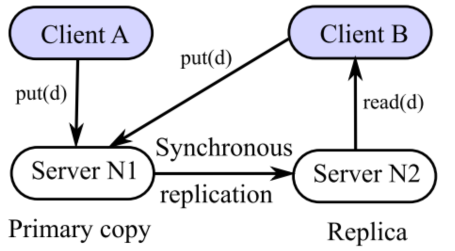
*If a fragment is placed in more than one server, then, we are **replicating** it*



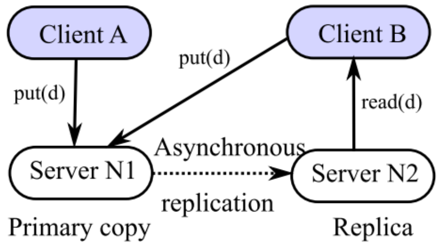
Physical View

Manage Data Replication

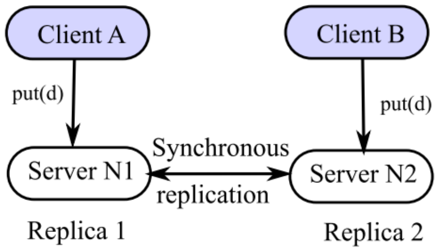
- Replicating fragments improves the system throughput but raises some other issues:
  - Consistency
  - Update performance
- Most used replication protocols
  - Primary – Secondary versioning
  - Eager – Lazy replication



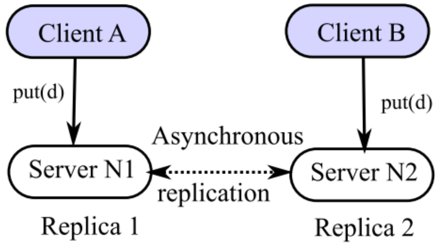
a) Eager replication with primary copy



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



c) Eager replication, distributed

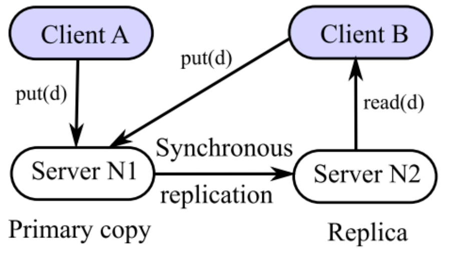


d) Lazy replication, distributed (a.k.a. Master-Master replication)

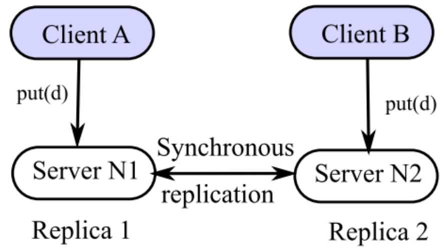
Manage Data Replication

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

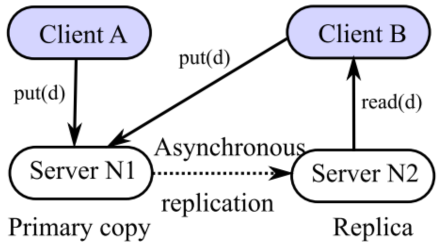
Strong Consistency



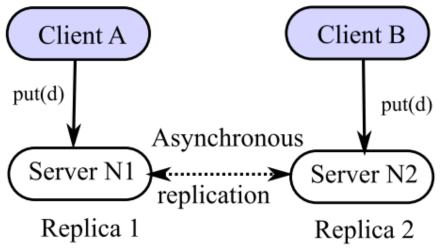
a) Eager replication with primary copy



c) Eager replication, distributed



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



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):
  a) Show a database replication strategy (e.g., sketch it) where:
     1. You buy an item, but this item does not appear in your basket.
     2. You reload the page: the item appears.

     What happened?
  b) Show a database replication strategy (e.g., sketch it) where:
     1. You delete an item from your command, and add another one: the basket shows both items.

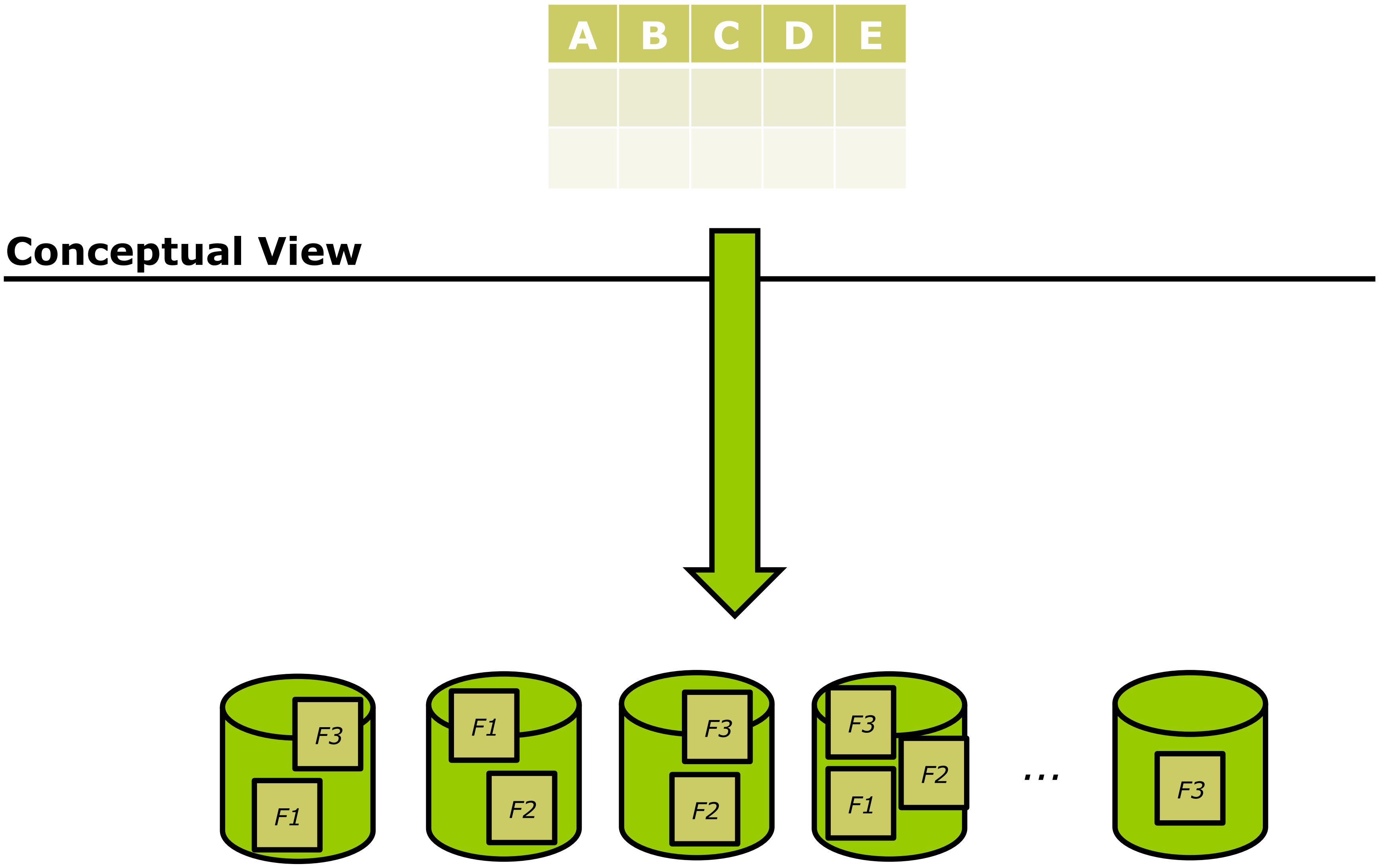
     What happened?

     Will the situation change if you reload the page?

Distributed Database design

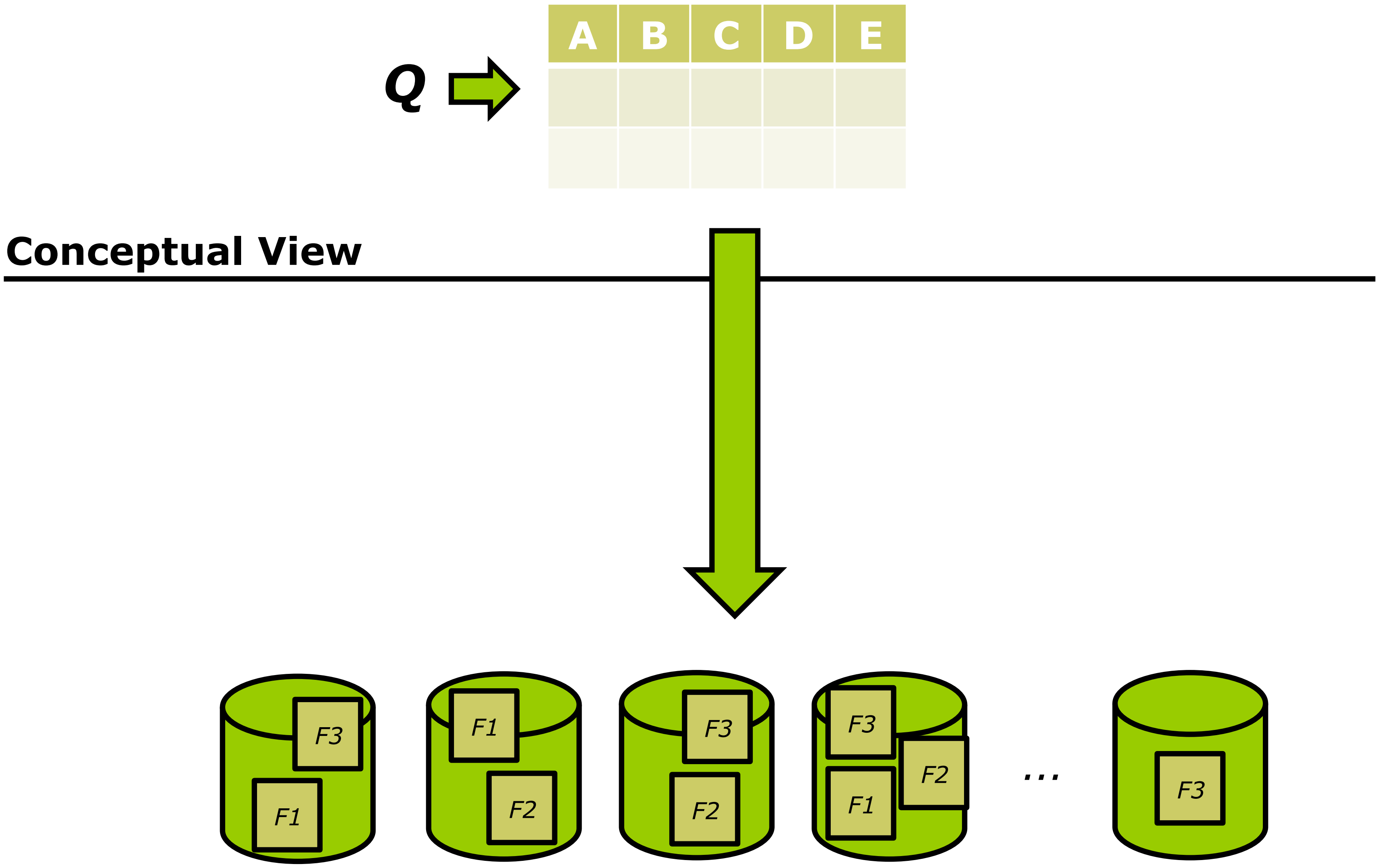
CATALOG

Challenge II: Global Catalog



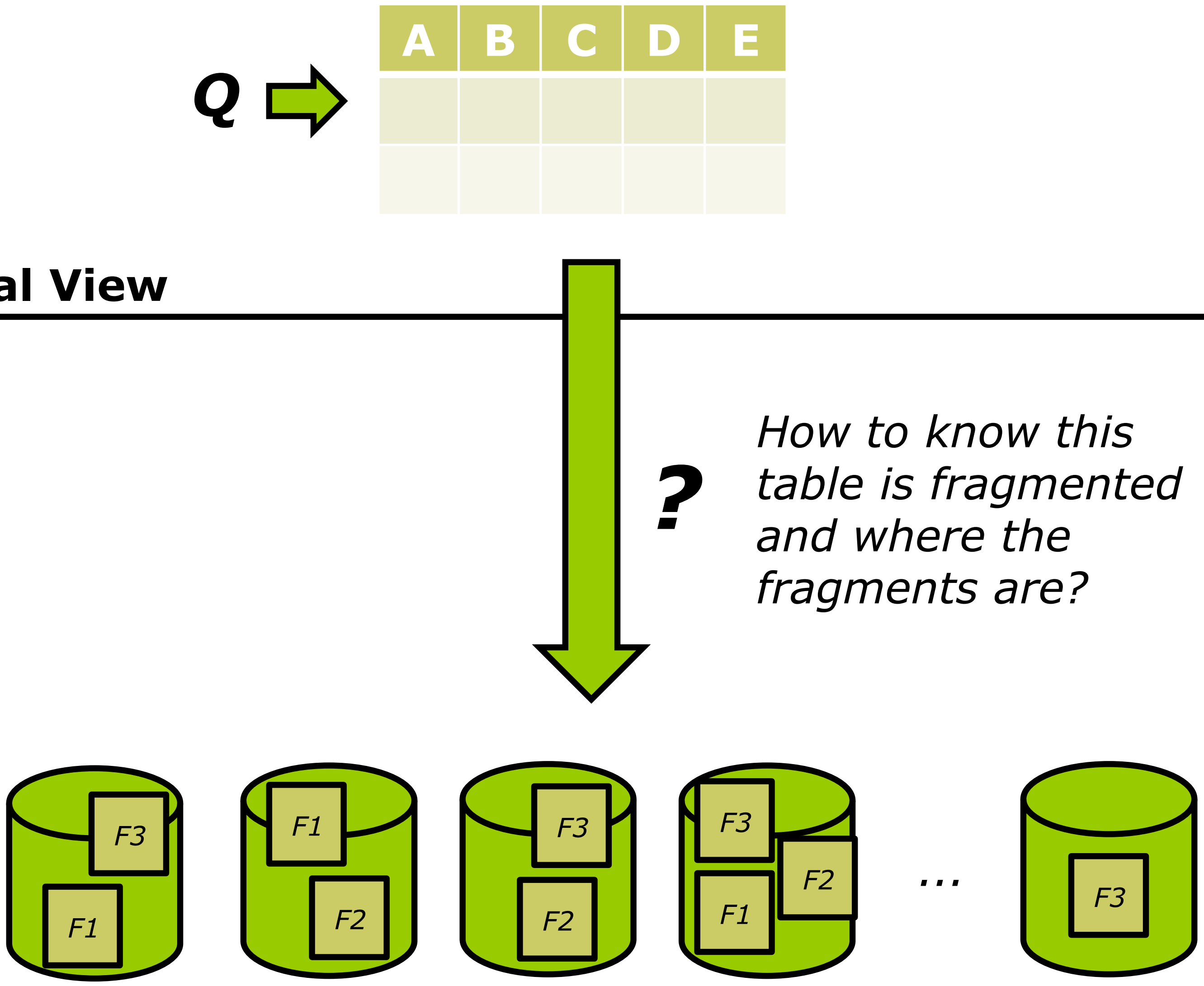
Physical View

Challenge II: Global Catalog



Physical View

Challenge II: Global Catalog



Conceptual View

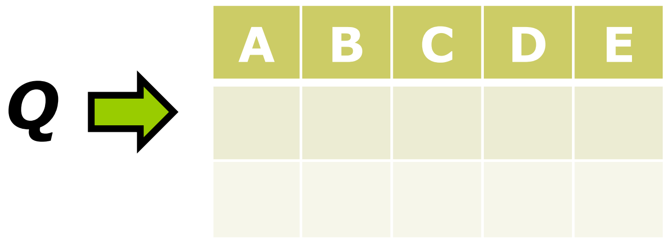
How to know this table is fragmented and where the fragments are?

Physical View

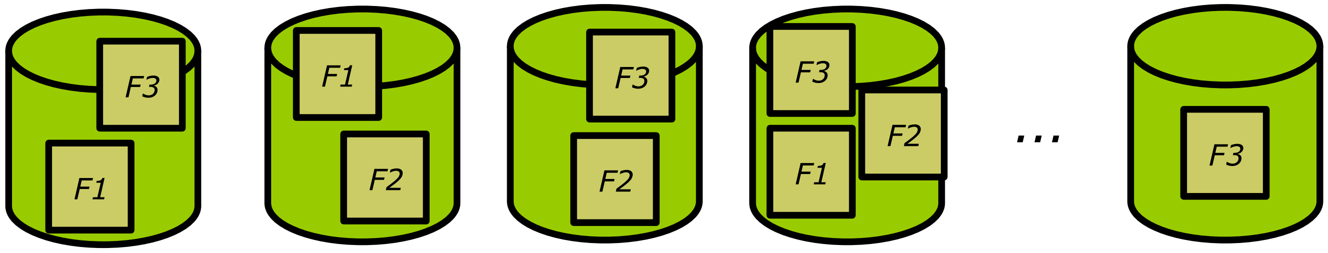
Challenge II: Global Catalog

- 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 II: Global Catalog

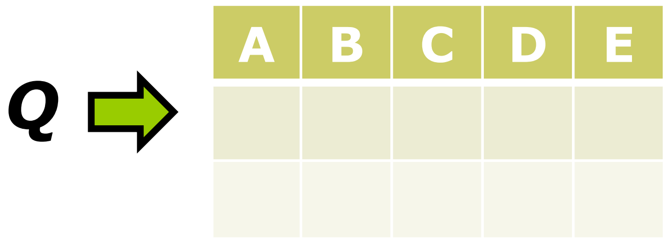


Conceptual View



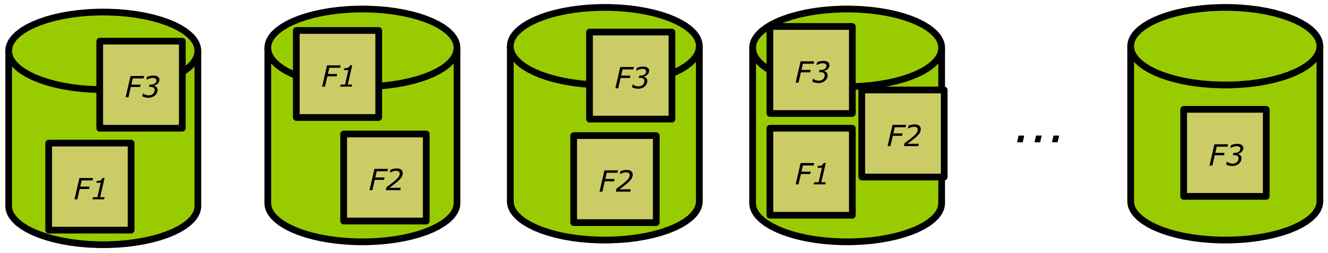
Physical View

Challenge II: Global Catalog



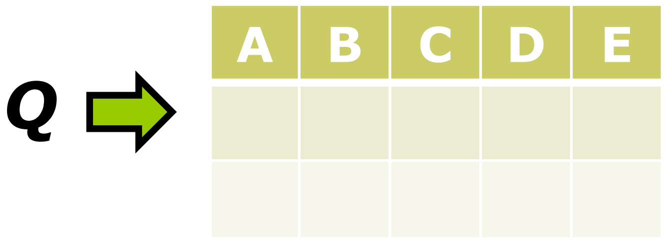
Conceptual View

Catalog:
T <<fragmentation strategy>>
F1: @S1, @S2, @S4
F2: @S2, @S3, @S4
F3: @S1, @S3, @S4, @Sn



Physical View

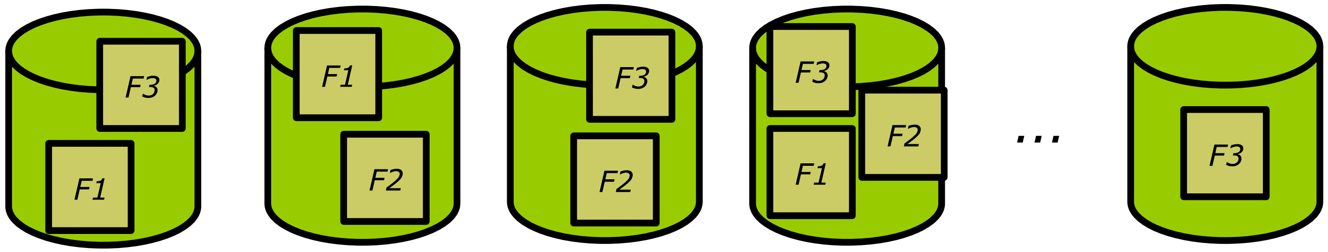
Challenge II: Global Catalog



Conceptual View

Catalog:
T <<fragmentation strategy>>
F1: @S1, @S2, @S4
F2: @S2, @S3, @S4
F3: @S1, @S3, @S4, @Sn

This information is typically stored in a distributed index

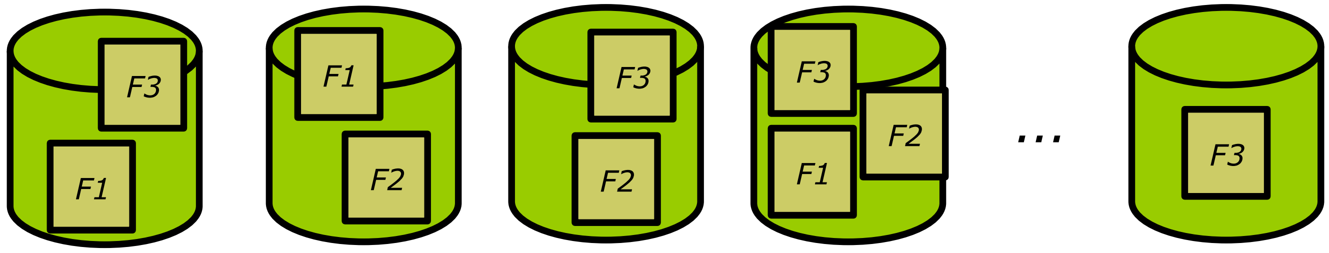


Physical View

Challenge II: Global Catalog

Conceptual View

Centralized Version

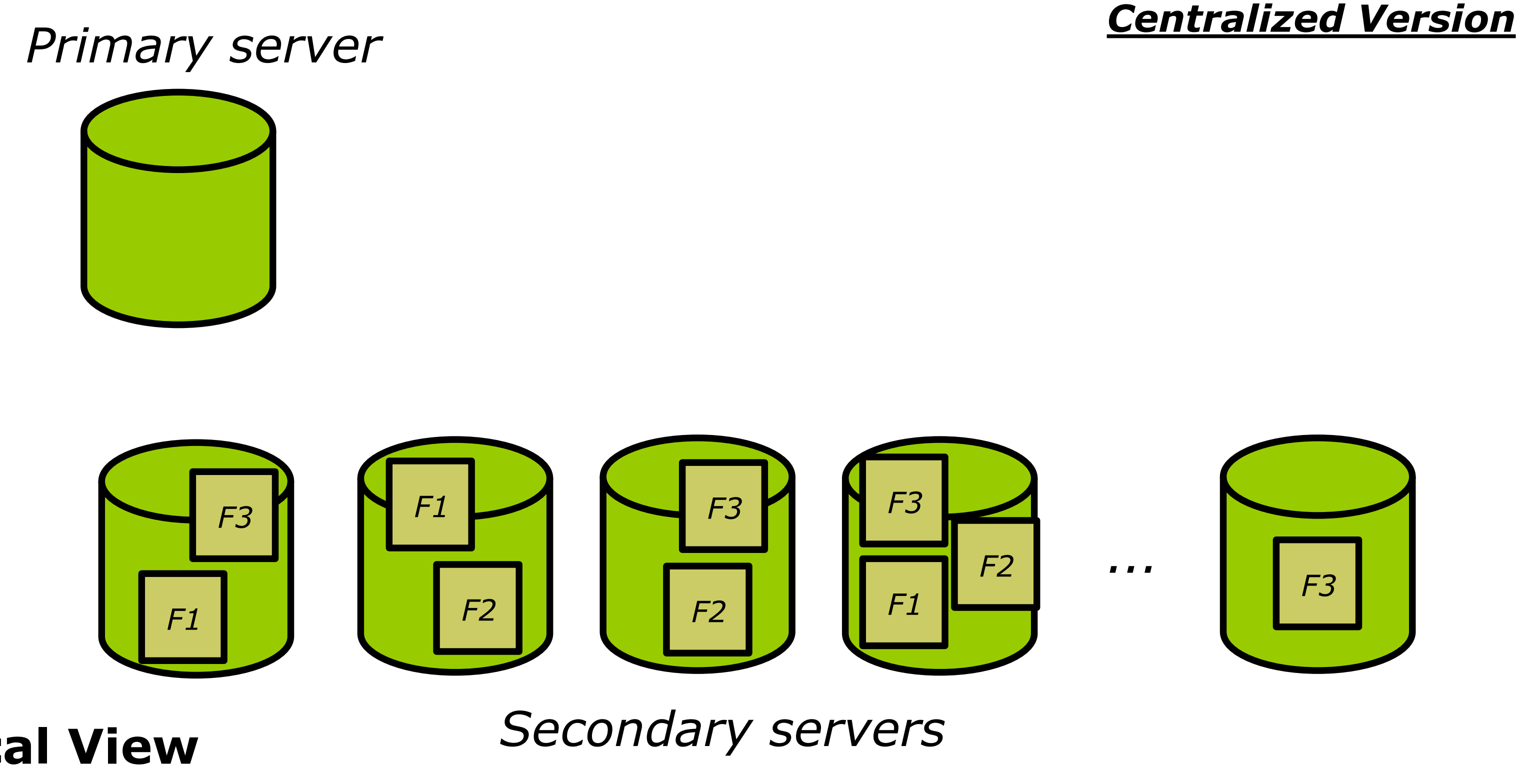


Physical View

Challenge II: Global Catalog

Conceptual View

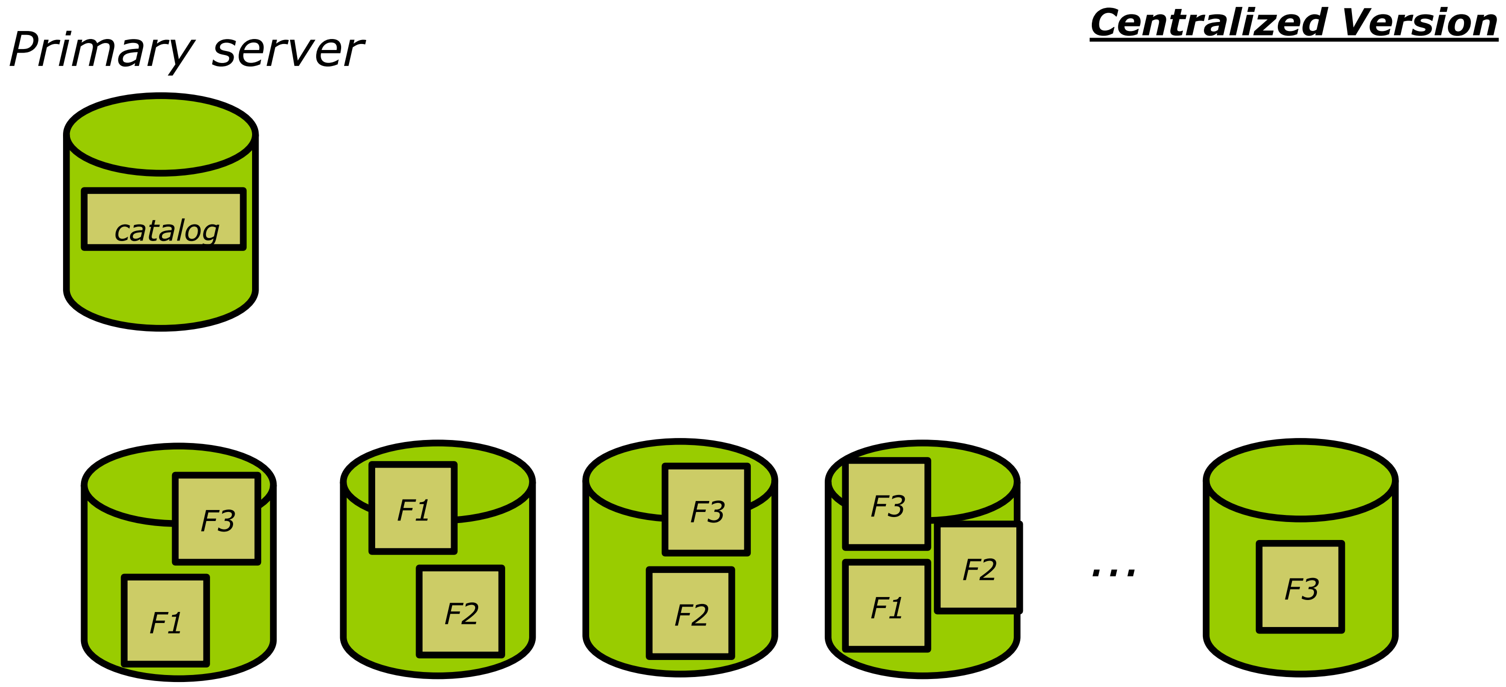
<u>Centralized Version</u>



Physical View

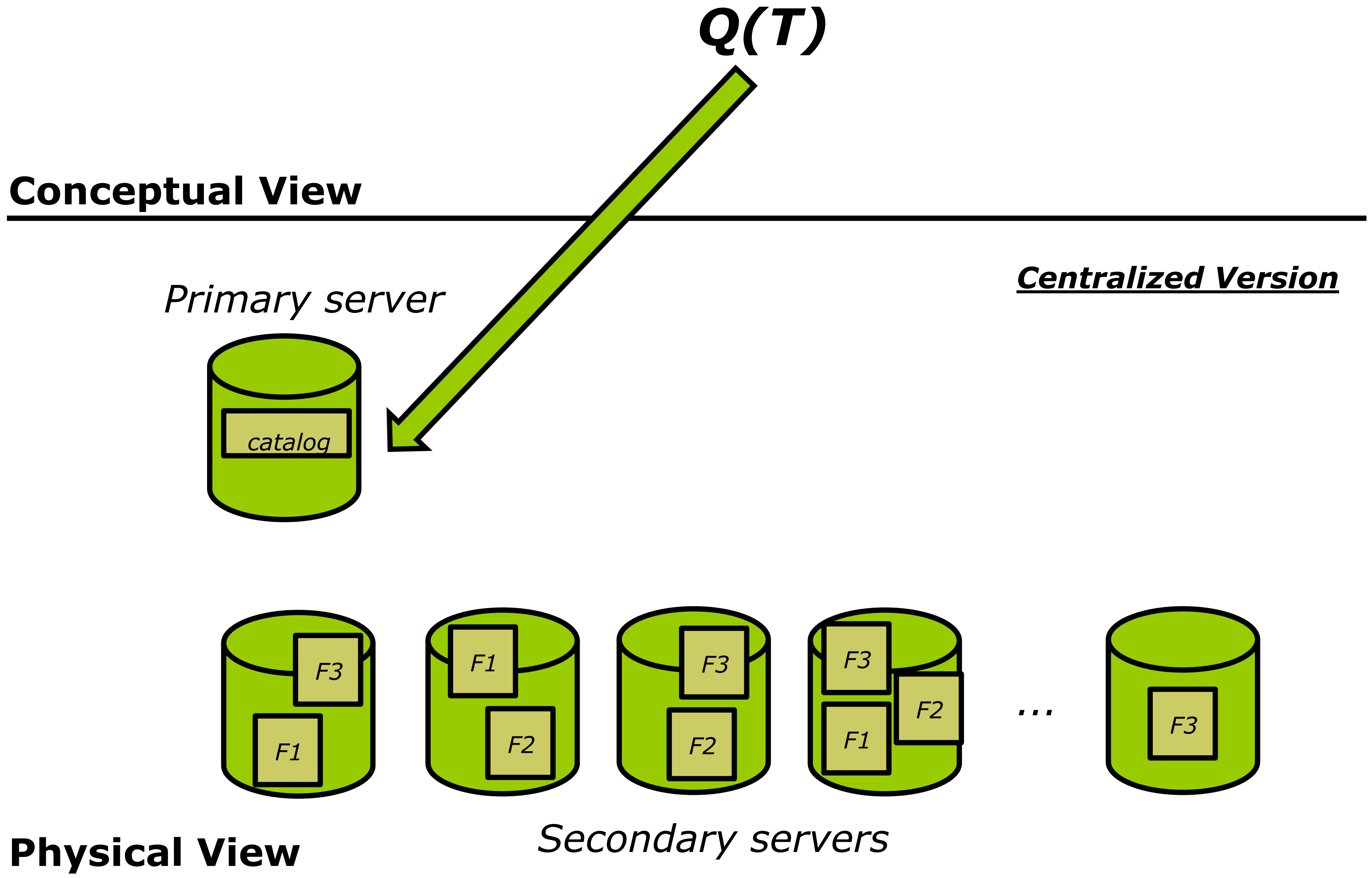
Challenge II: Global Catalog

Conceptual View



Physical View

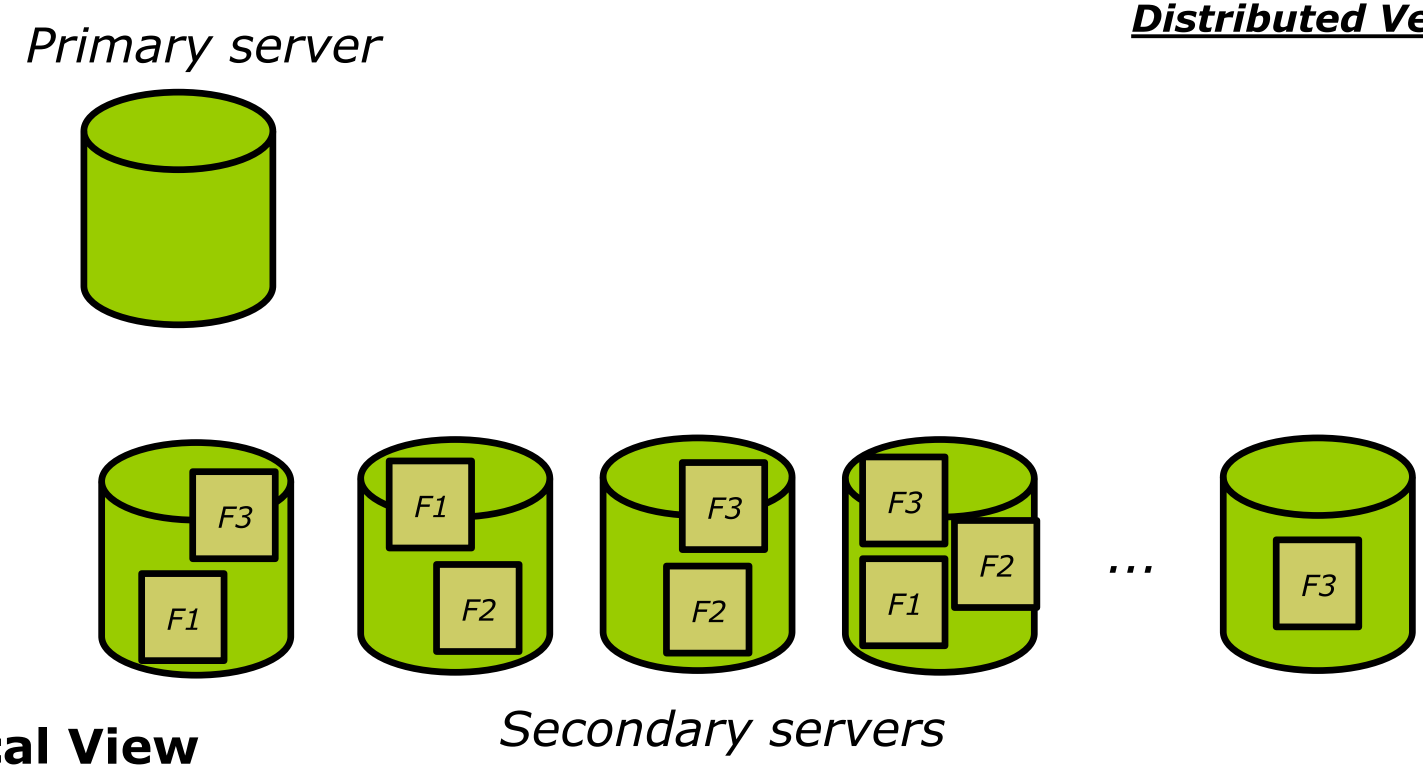
Challenge II: Global Catalog



Challenge II: Global Catalog

Conceptual View

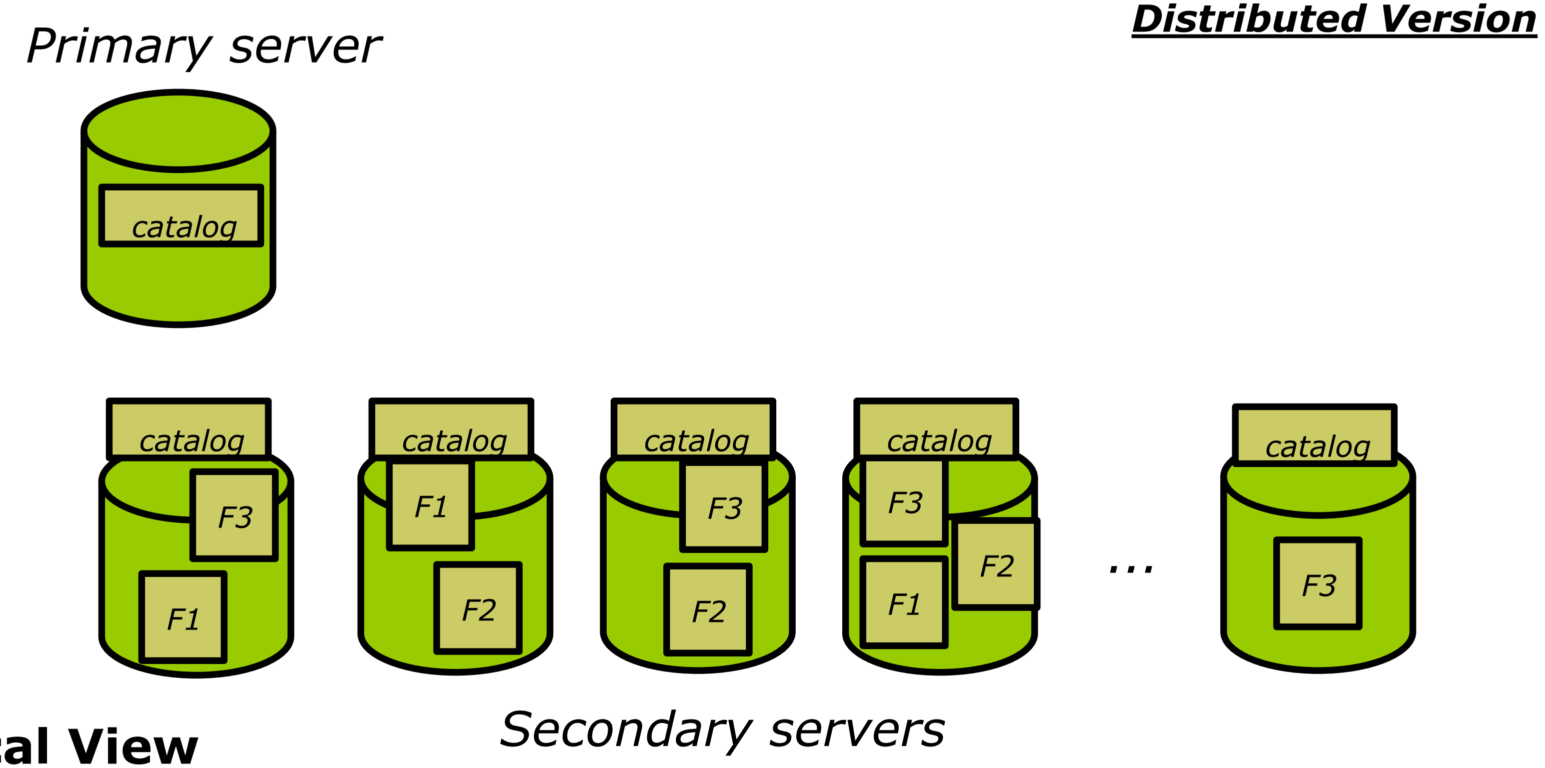
Distributed Version



Physical View

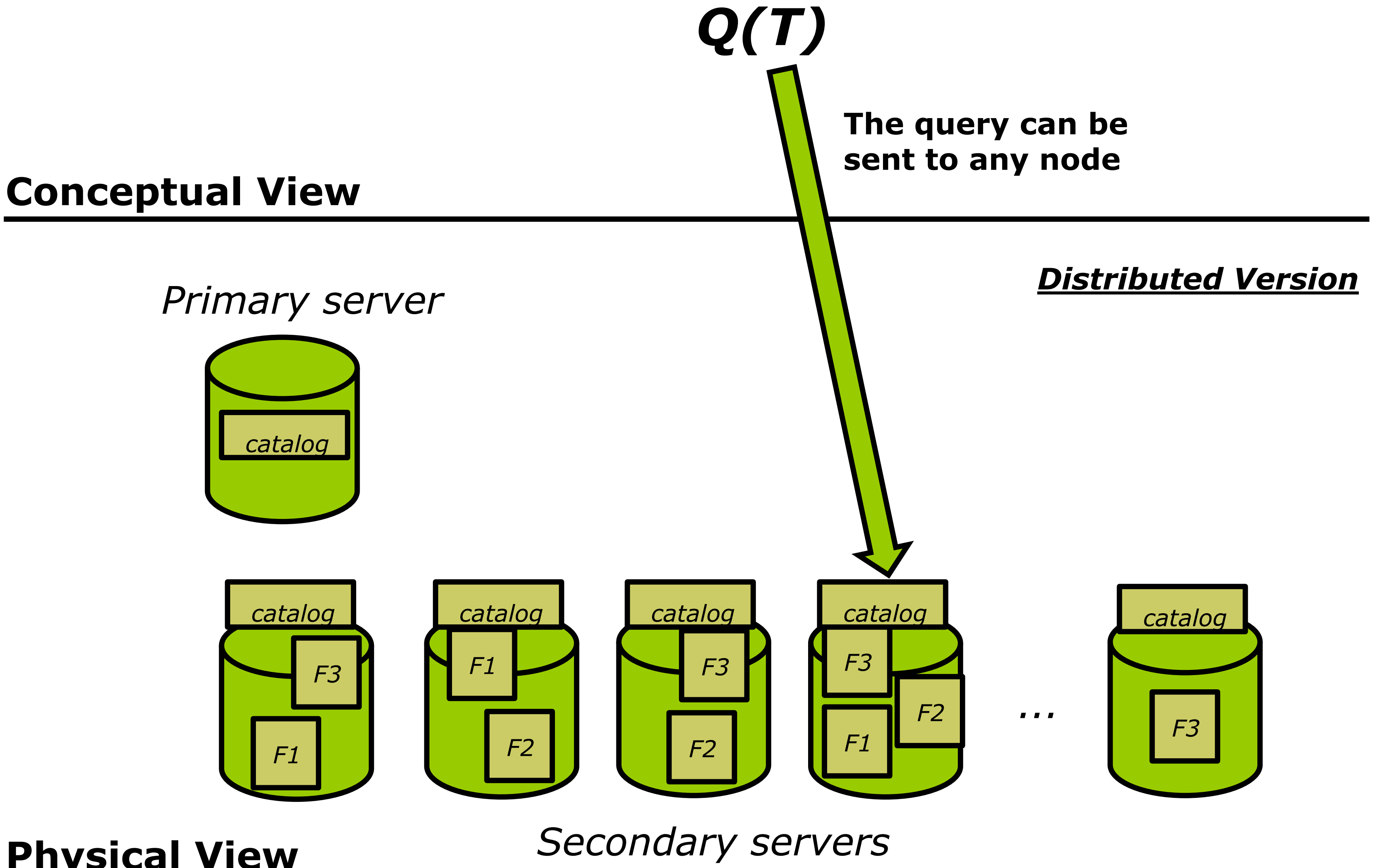
Challenge II: Global Catalog

Conceptual View

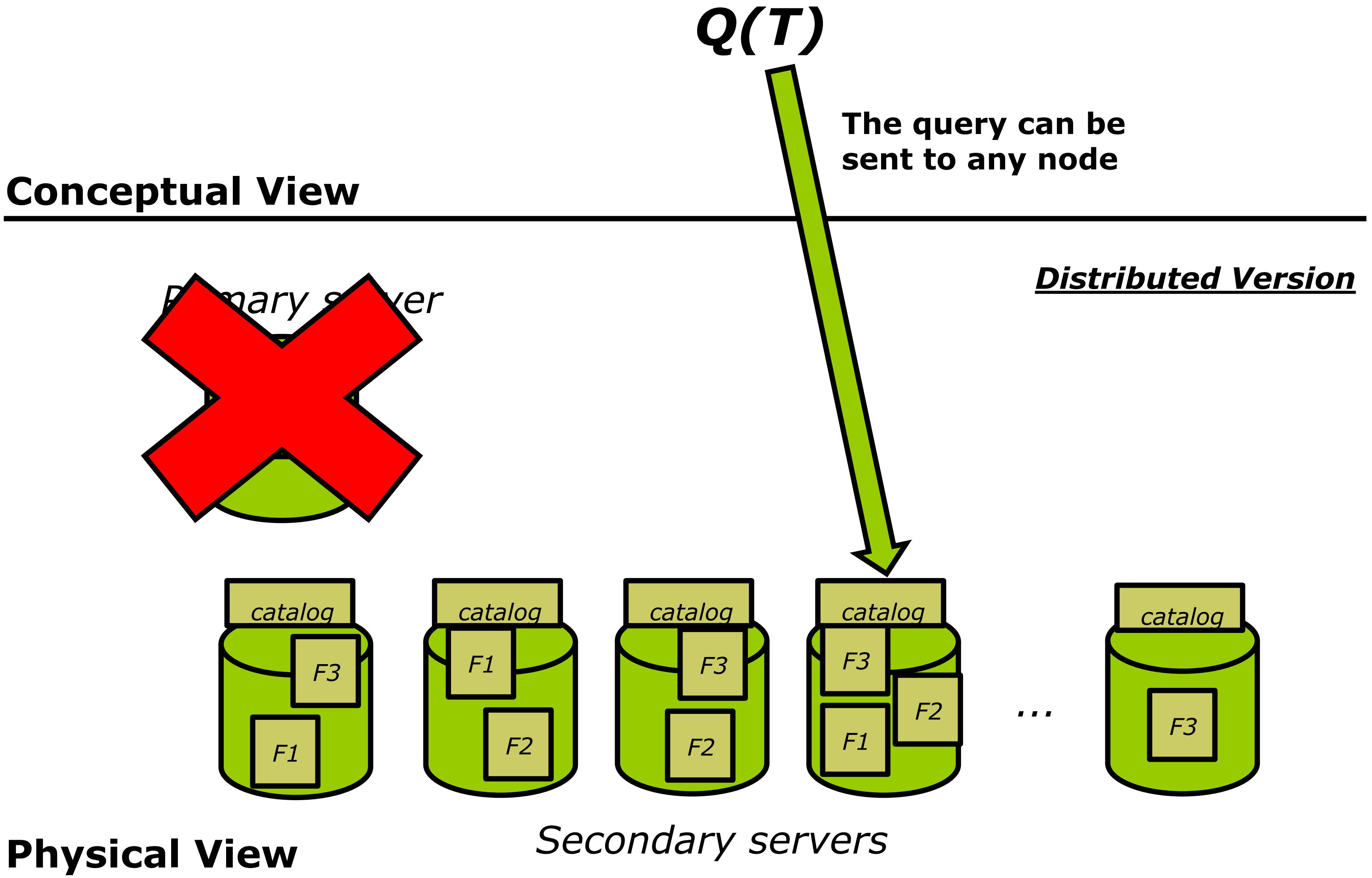


Physical View

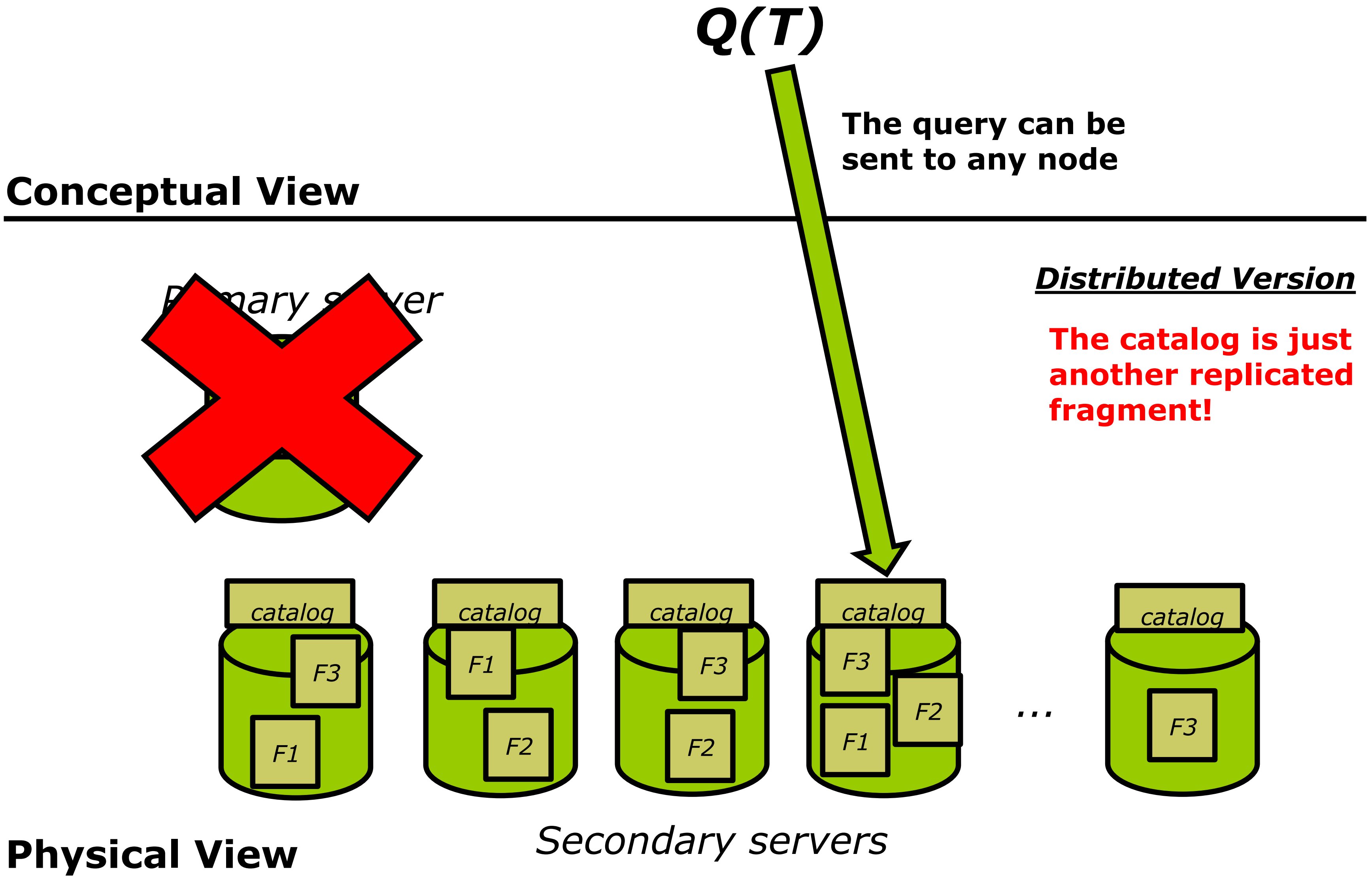
Challenge II: Global Catalog



Challenge II: Global Catalog



Challenge II: Global Catalog



Distributed Database design

SCALABILITY

Definition

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

Definition

- How do we define "Scalability"? And "Elasticity"?
  - 3 minutes to think of it!
- It is a design issue
  - Current systems do not tell you *how many* CPUs or servers you must use
- The Universal Scalability Law (USL) – Gunther 1993
  - It is a mathematical definition of scalability
    - Main idea: You cannot model what you cannot formally define

The Universal Scalability Law (I)

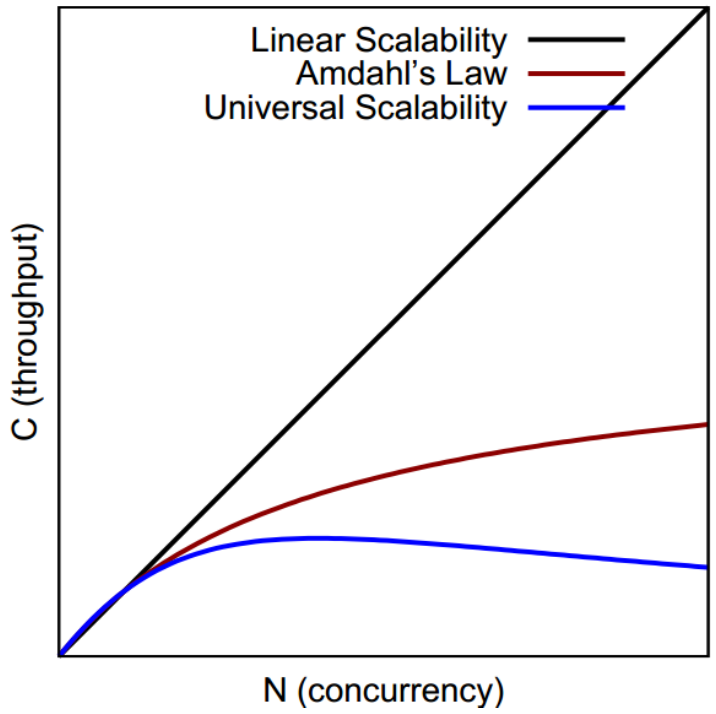
- It can model both Sw or Hw scalability
- The USL is defined as follows:

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

  - 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 σ = 0 and κ = 0, we obtain linear scalability
- If κ = 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

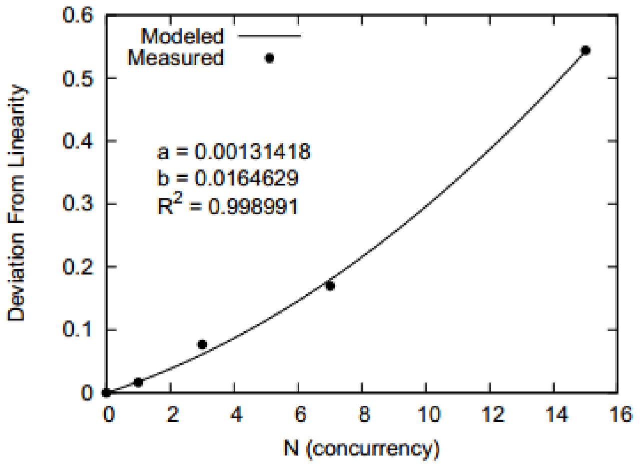
The USL at Work: Example

- 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:

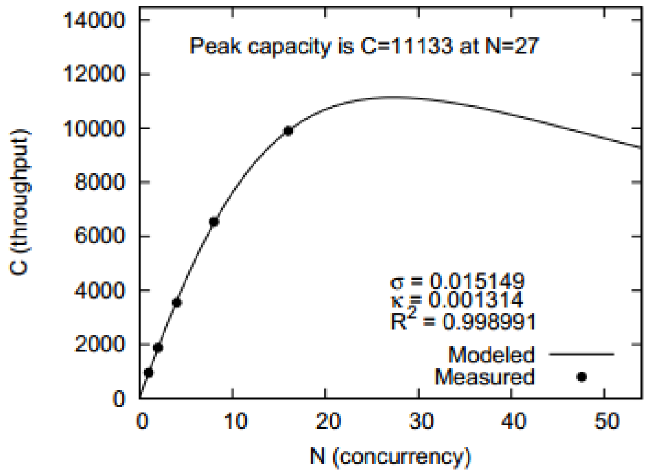
Concurrency (N)	Throughput (C)
1	955.16
2	1878.91
4	3548.68
8	6531.08
16	9897.24

Step 2:



Points are fit in a second-order polynomial: ax²+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

Summary

- Distributed Database design
  - Data fragmentation
  - Data allocation
    - Data replication
- Global catalog management
- Scalability
  - Universal Scalability Law

Bibliography

- M.T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Second edition. Prentice Hall, 1999
- Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart. *Web Data Management*. Cambridge Press, 2011.
- L. Liu, M.T. Özsu (Eds.). *Encyclopedia of Database Systems*. Springer, 2009