Distributed Query Optimization

Knowledge Objetives

1. Explain how the data localization phase of a distributed query processing works
2. Enumerate two strategies used in the global optimization phase
3. Explain the difference between data shipping and query shipping
4. Explain how the semi-join strategy for distributed joins works
5. Name two models used to evaluate the plans proposed by the query optimizer
6. Explain three kinds of parallel query processing and the main approaches to support them
7. Discuss how cost models and query plan evaluation must be extended to support parallel query processing

Understanding Objectives

1. Justify the site selection for joins chosen by a DDBMS
2. Choose between a semi-join or distributed join strategy
3. Compute basic cost models for distributed query processing

DISTRIBUTED QUERY PROCESSING

MOTIVATION & ARCHITECTURE

Efficiency

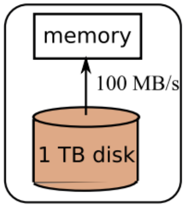
- In NOSQL efficiency is achieved by means of parallelism
  - **Query processing must be able to exploit parallelism**
    - **Divide-and-conquer philosophy**

- Reminder:

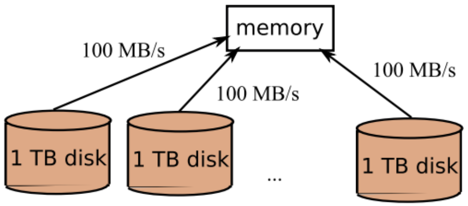
Type	Latency	Bandwidth
Disk	$\approx 5 \times 10^{-3}$s (5 millisec.);	At best 100 MB/s
LAN	$\approx 1 - 2 \times 10^{-3}$s (1-2 millisec.);	$\approx$ 1GB/s (single rack); $\approx$ 100MB/s (switched);
Internet	Highly variable. Typ. 10-100 ms.;	Highly variable. Typ. a few MB/s.;

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.

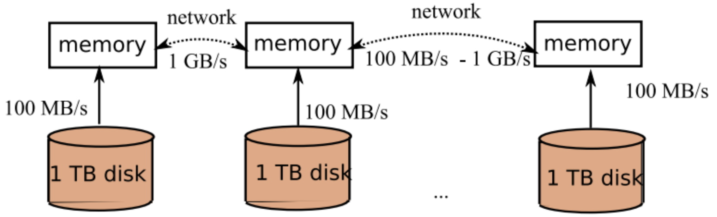
Bottom line (2): exchanging through the Internet is slow and unreliable with respect to LANs.



a. Single CPU, single disk



b. Parallel read: single CPU, many disks



c. Distributed reads: an extendible set of servers

Reminder: 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

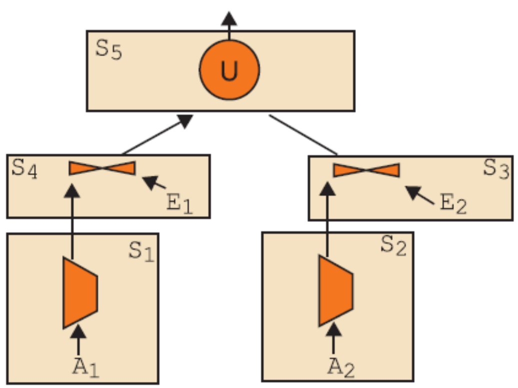
III. Distributed query processing
- Data distribution / replication
- Communication overhead

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

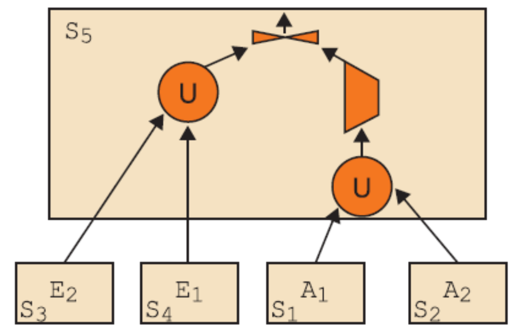
Activity: Distributed Query Processing

- 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?
        I. Under which assumptions is one or the other better?
     II. 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

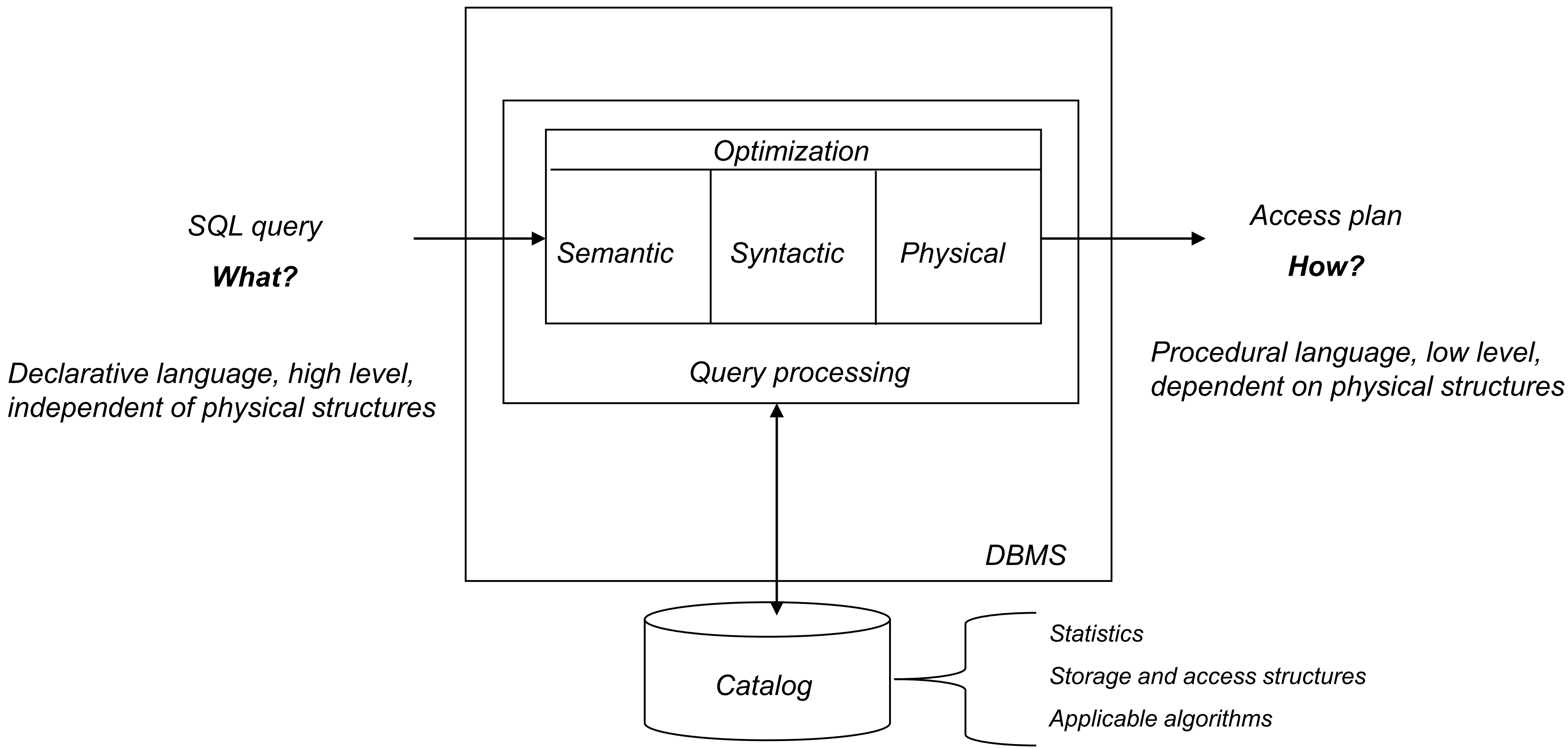


Acces Plan B

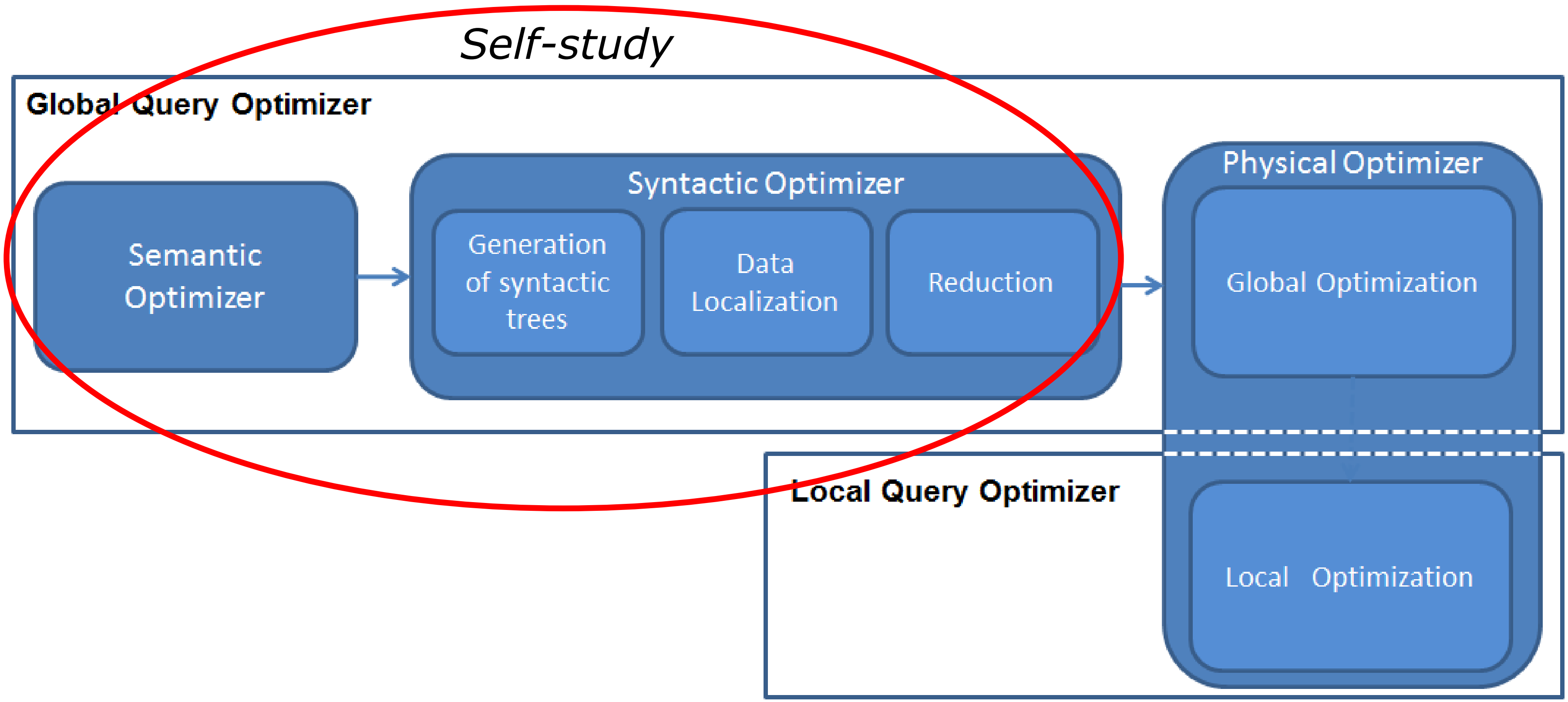
AssignedTo (#emp,#proj, responsibility, fullTime)
- S_1: A_1 = AssignedTo(#emp≤‘E3’)
- S_2: A_2 = AssignedTo(#emp>‘E3’)

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

Architecture of the Query Optimizer



Phases of Distributed Query Processing



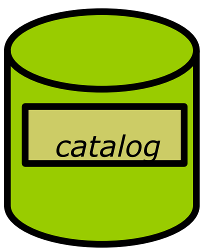
Challenge III: Distributed Query Processing

T

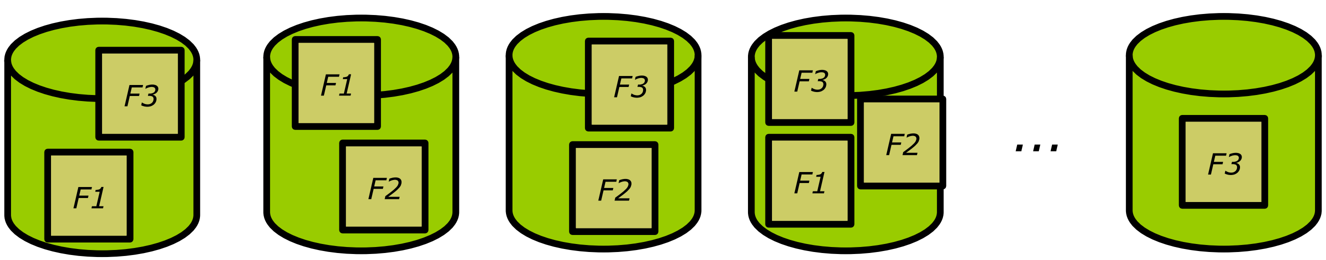
A	B	C	D	E

Conceptual View

Primary server



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



Physical View

Secondary servers

Challenge III: Distributed Query Processing

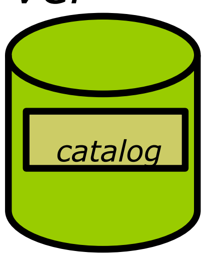
*SELECT **
FROM T
WHERE A > 5

T

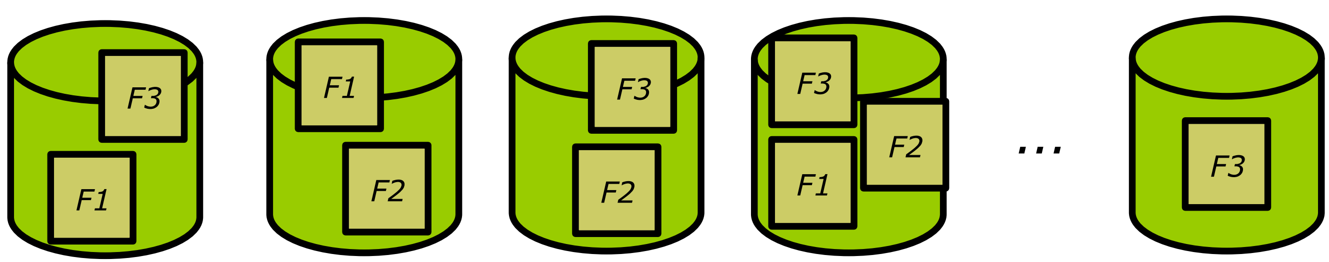
A	B	C	D	E

Conceptual View

Primary server



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



Secondary servers

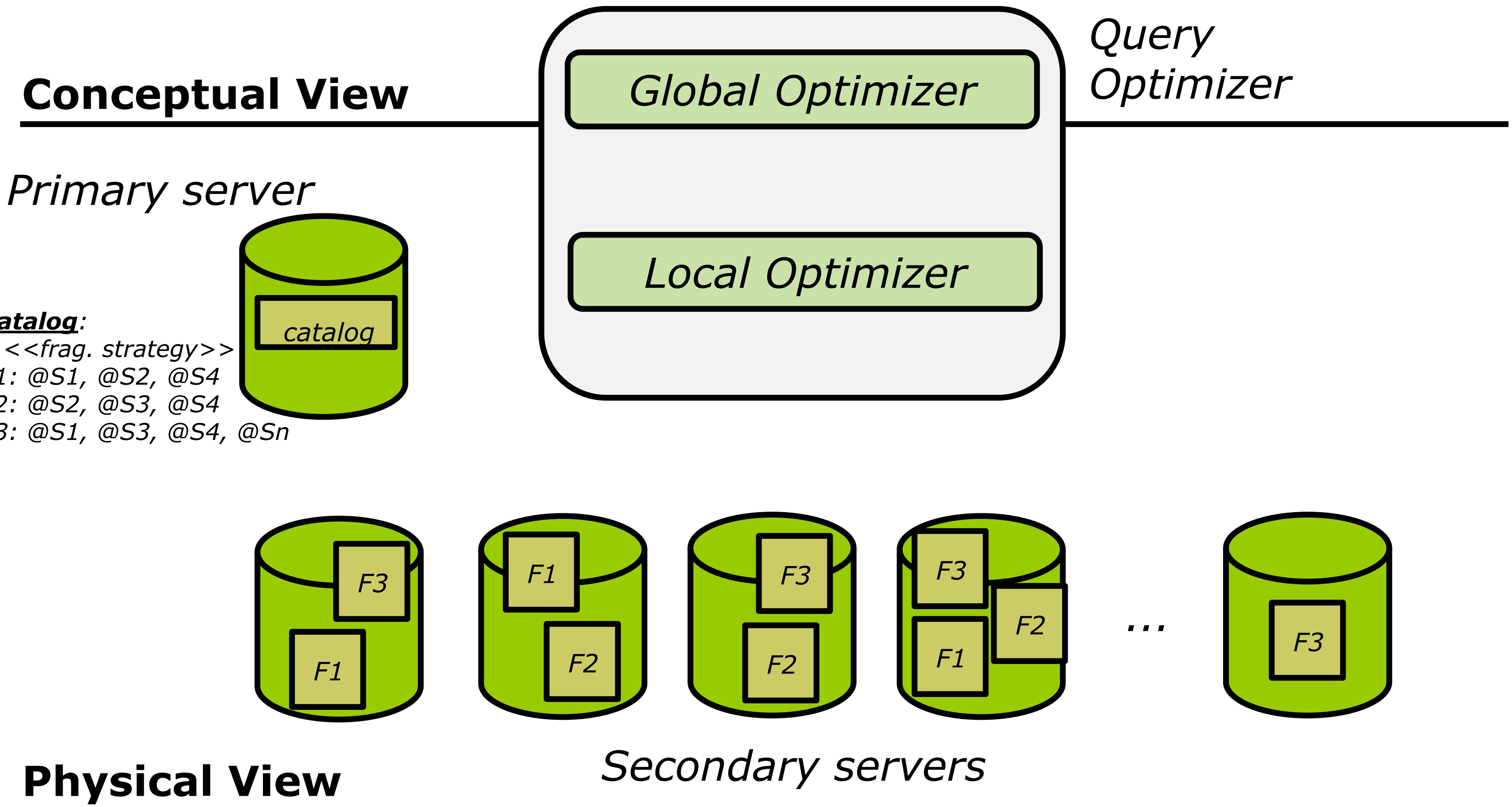
Physical View

Challenge III: Distributed Query Processing

```
SELECT *
FROM T
WHERE A > 5
```

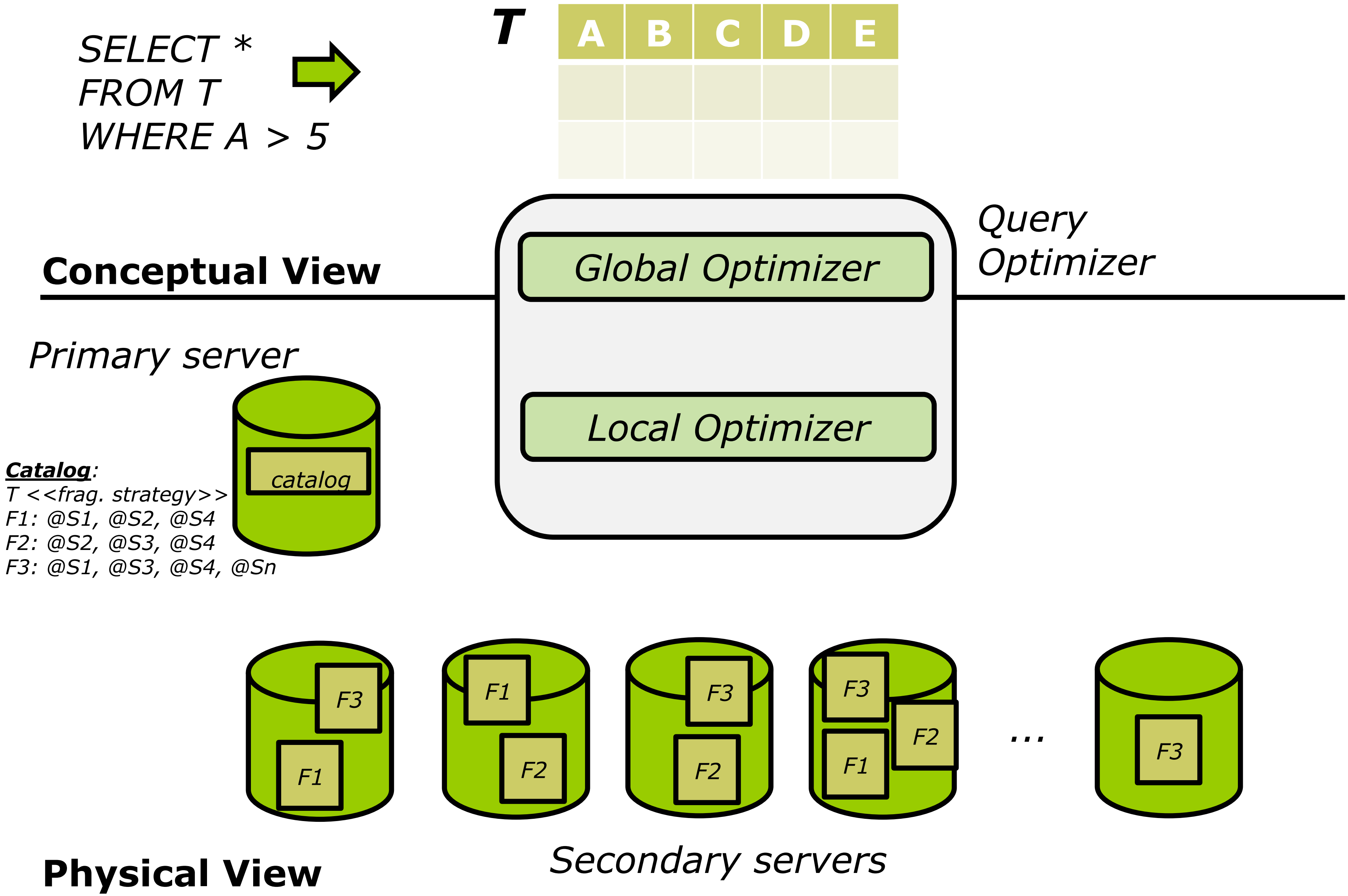
T

A	B	C	D	E



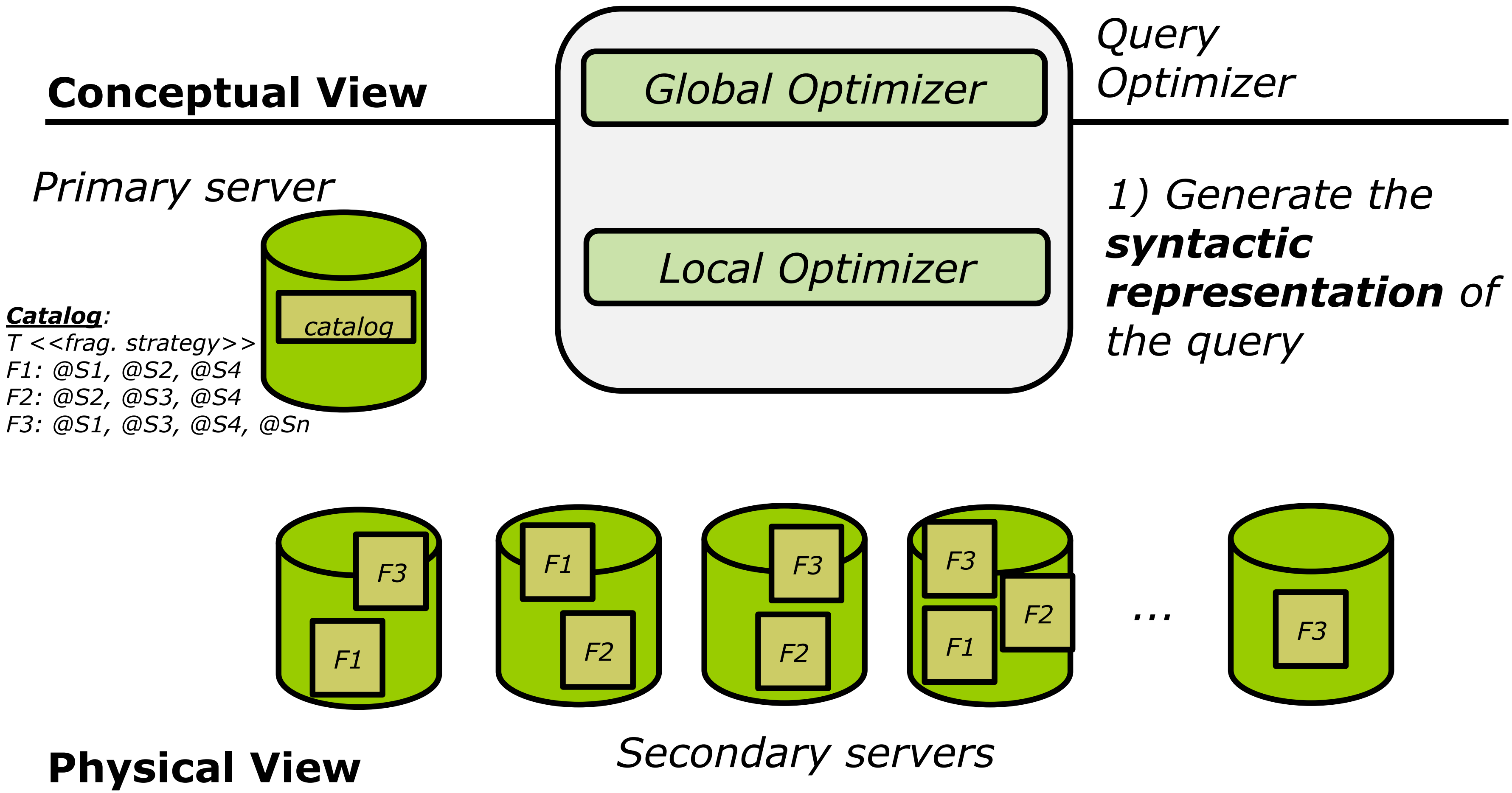
Catalog:
T <<frag. strategy>>
F1: @S1, @S2, @S4
F2: @S2, @S3, @S4
F3: @S1, @S3, @S4, @Sn

Challenge III: Distributed Query Processing

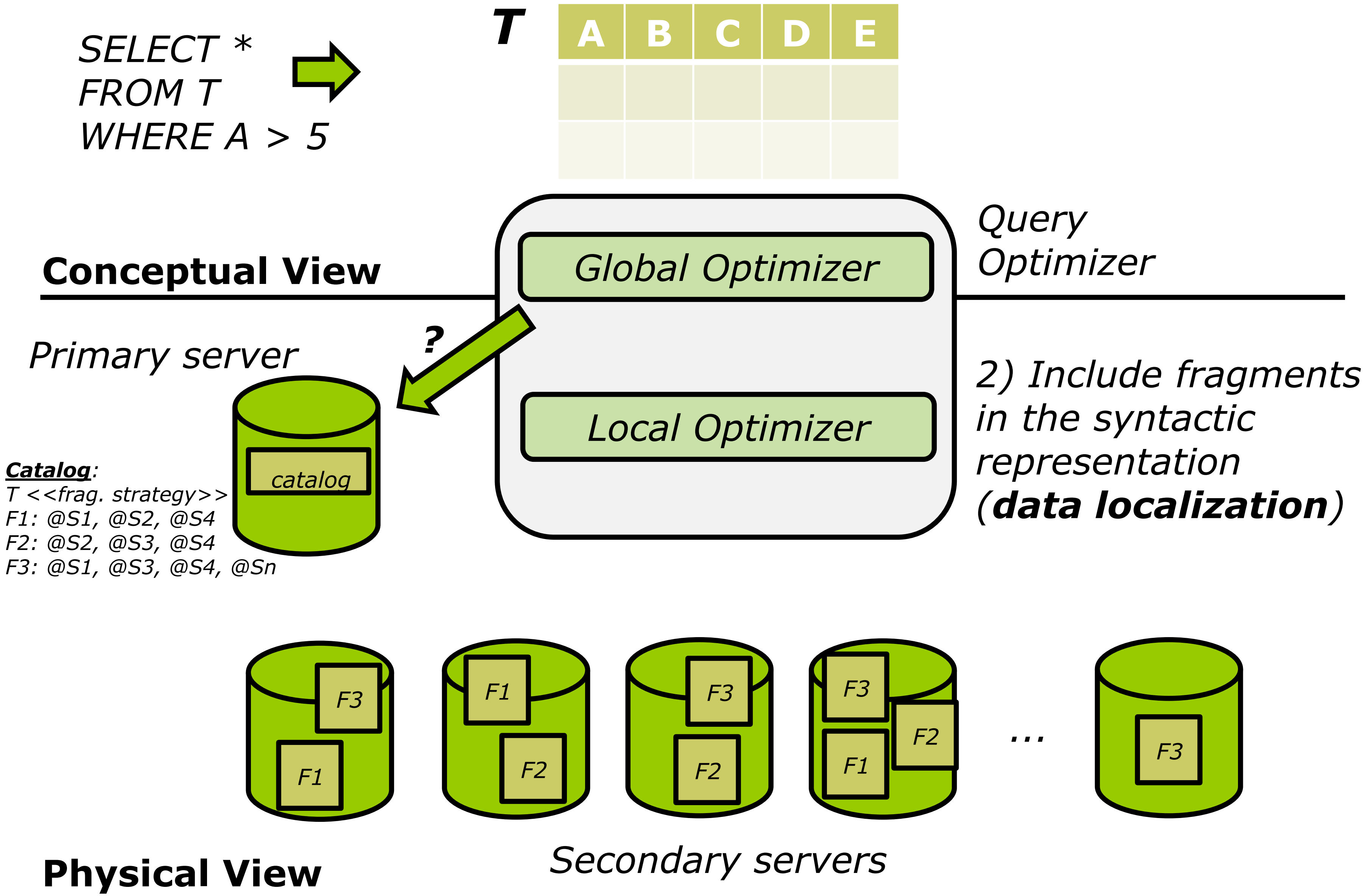


Challenge III: Distributed Query Processing

```
SELECT *
FROM T
WHERE A > 5
```



Challenge III: Distributed Query Processing



Challenge III: Distributed Query Processing

*SELECT **
FROM T
WHERE A > 5

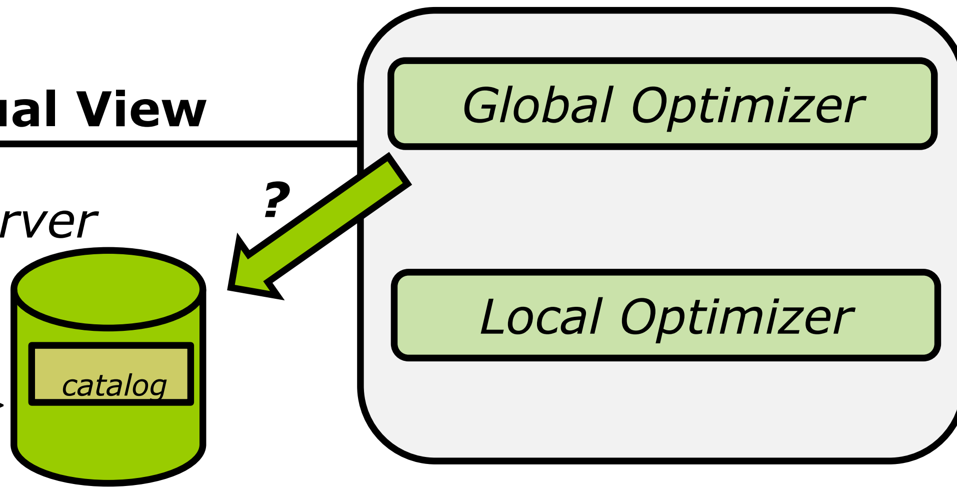
T

A	B	C	D	E

Query Optimizer

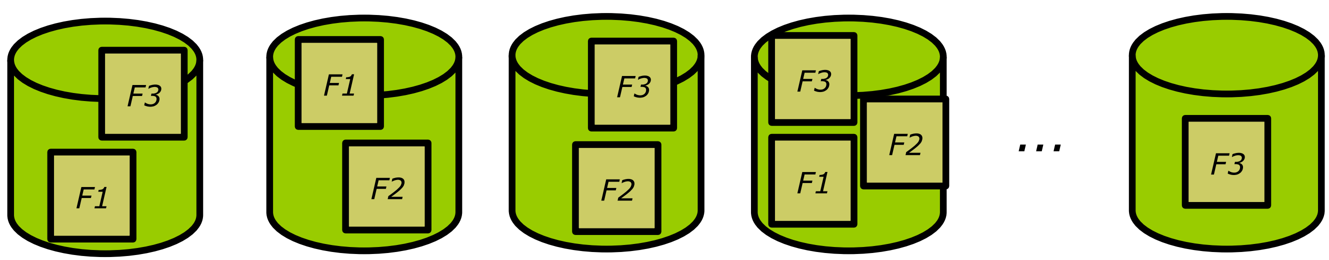
Conceptual View

Primary server



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

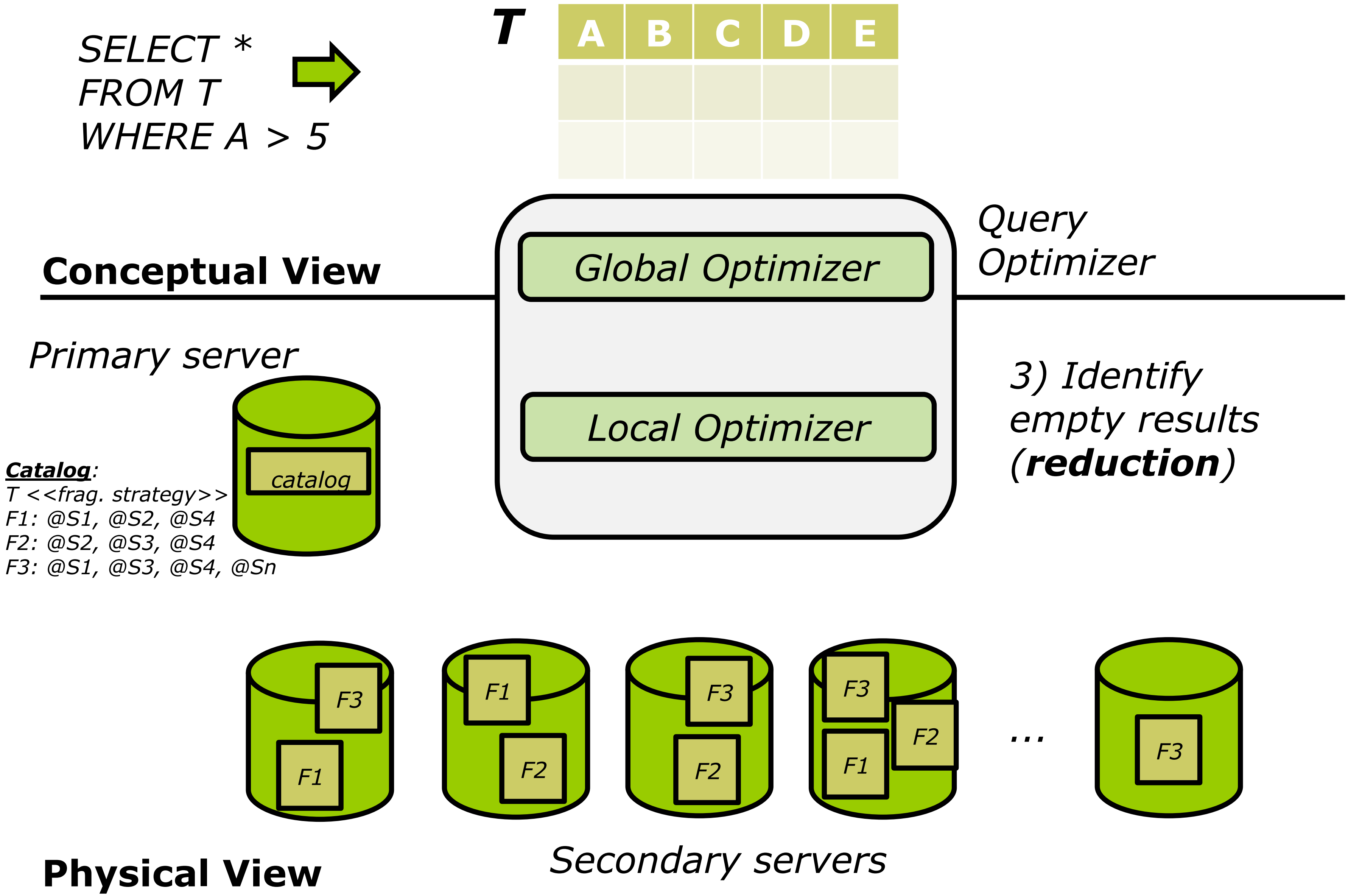
*2) Include fragments in the syntactic representation (**data localization**)*
2.1) Optimize the resulting tree



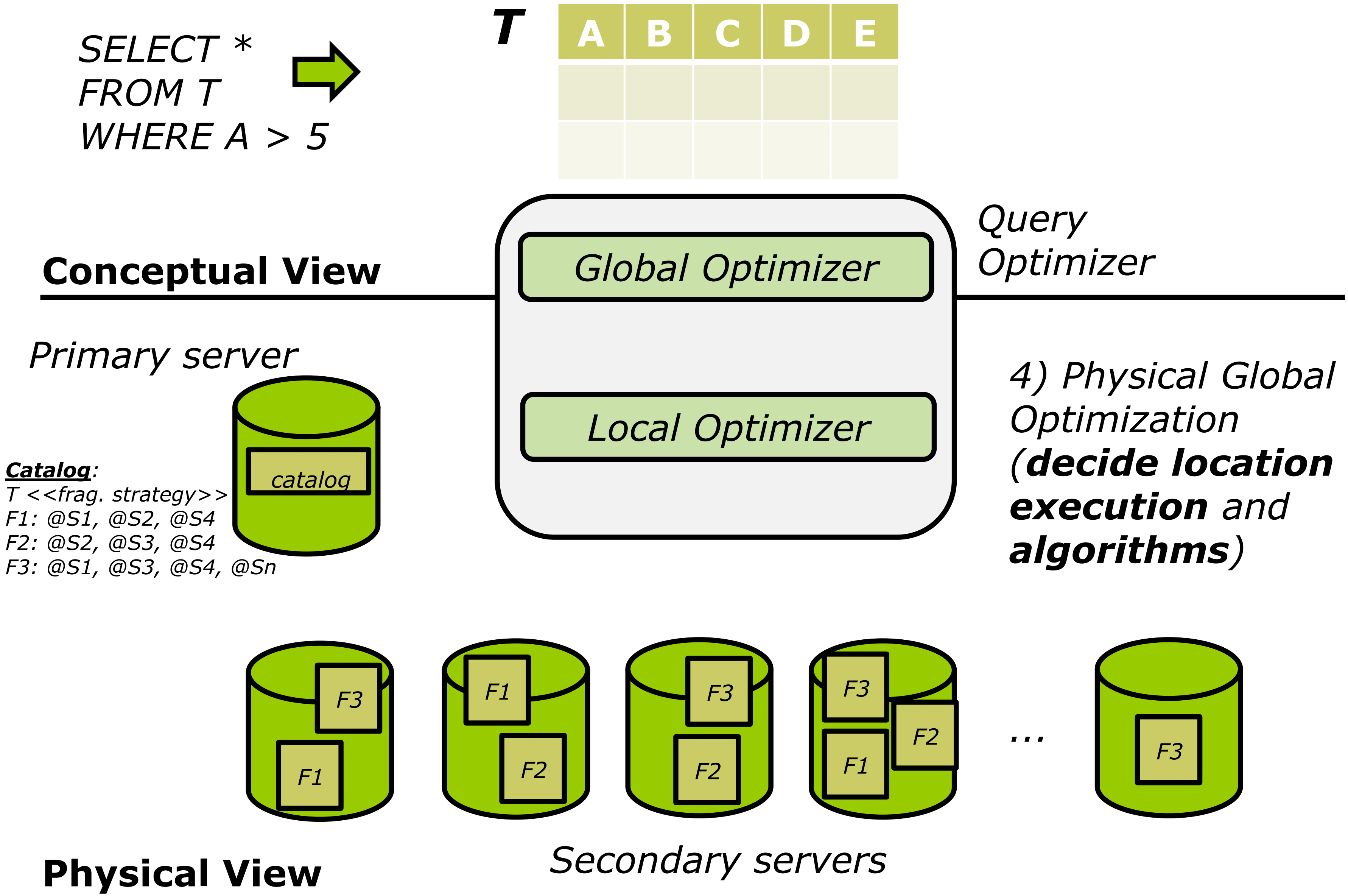
Physical View

Secondary servers

Challenge III: Distributed Query Processing



Challenge III: Distributed Query Processing

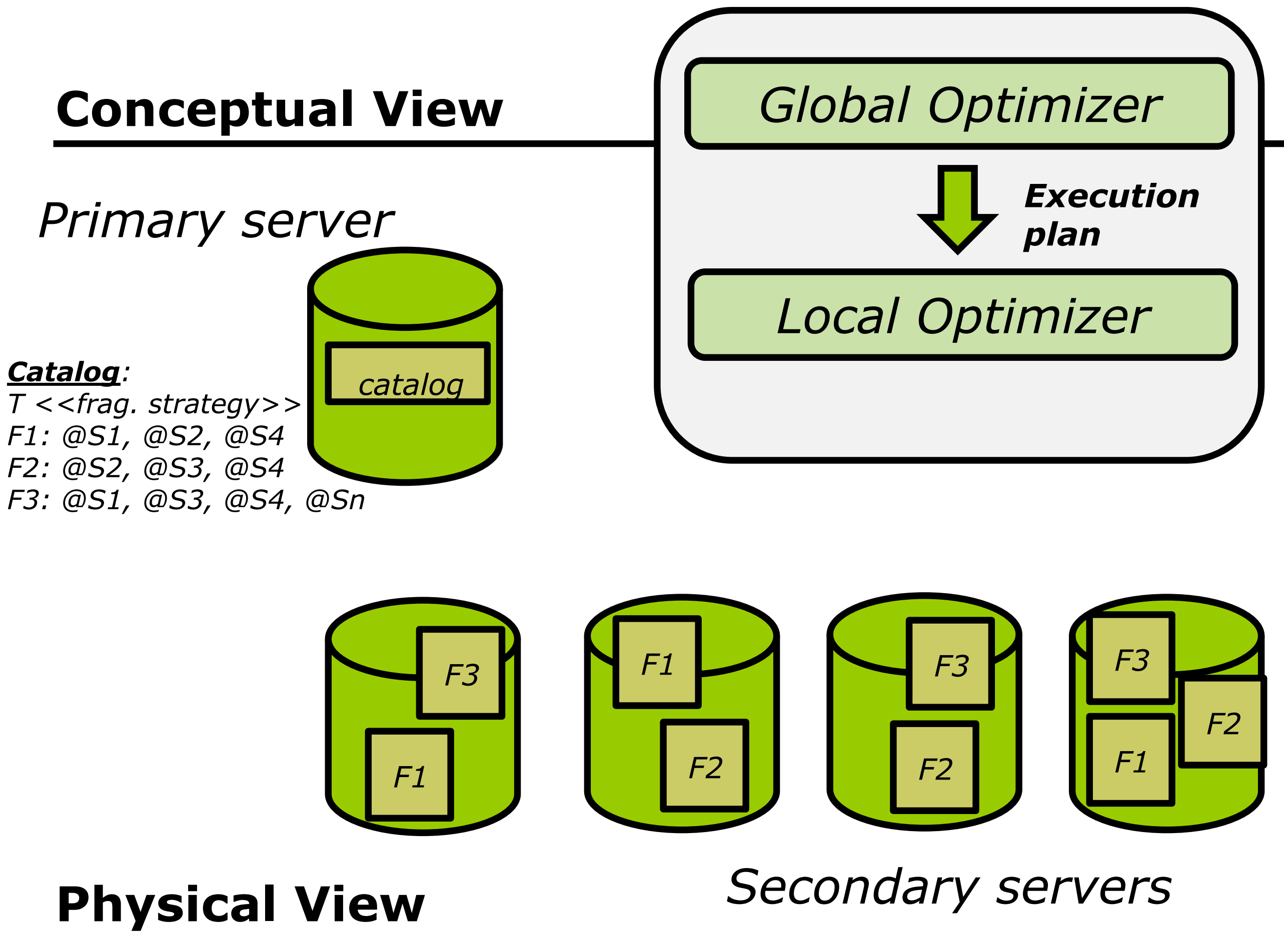


Challenge III: Distributed Query Processing

*SELECT **
FROM T
WHERE A > 5

T

A	B	C	D	E



*4) Physical Global Optimization (**decide location execution** and **algorithms**)*

Challenge III: Distributed Query Processing

Factors to consider:

- Communication cost (*data shipping*)
  - Not that critical for LAN networks if assuming high enough I/O cost
- Fragmentation / Replication
  - Location of the fragments / replicas (global catalog)
  - Statistics about each fragment / replica (required by the cost model)
- Join Optimization
  - Joins order
  - Semi-join strategy
- How to decide the execution plan
  - Who executes what
  - Exploit parallelism (!)

DISTRIBUTED QUERY PROCESSING

PHYSICAL OPTIMIZATION

Global Physical Optimizer

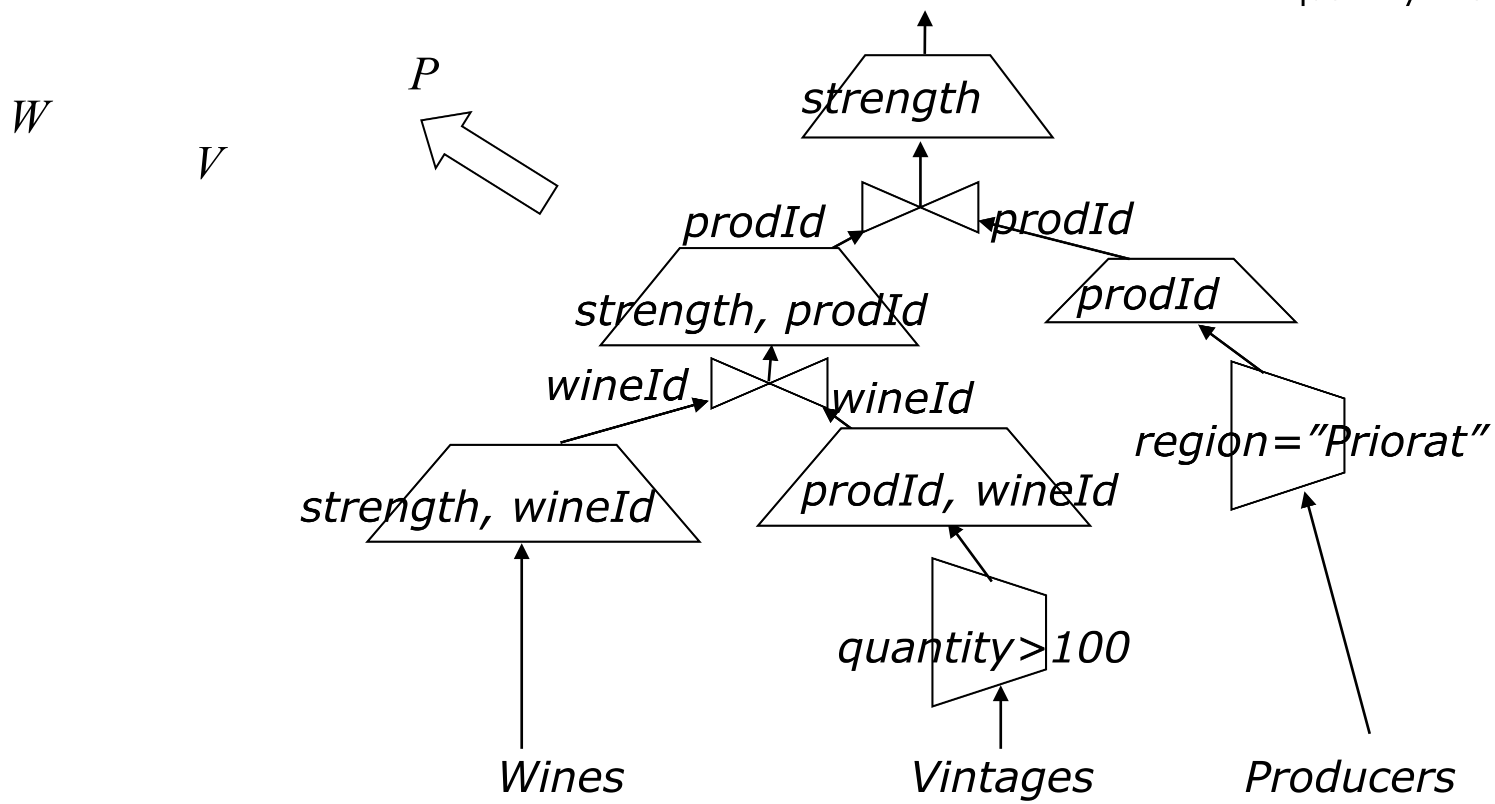
- **Objective**: Transforms an optimize physical representation into an efficient plan
  - Replaces the logical query operators by specific algorithms (plan operators) and access methods
  - Decides in which order execute them
- This is done by…
  - Generating the process tree
  - Enumerating alternative but equivalent *plans*
    - Dataflow diagram (i.e., the **process tree**) 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

- **Objective**: Transforms an optimize physical representation into an efficient plan
  - Replaces the logical query operators by specific algorithms (plan operators) and access methods
  - Decides in which order execute them
- This is done by…
  - Generating the process tree
  - Enumerating alternative but equivalent *plans*
    - Dataflow diagram (i.e., the **process tree**) 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

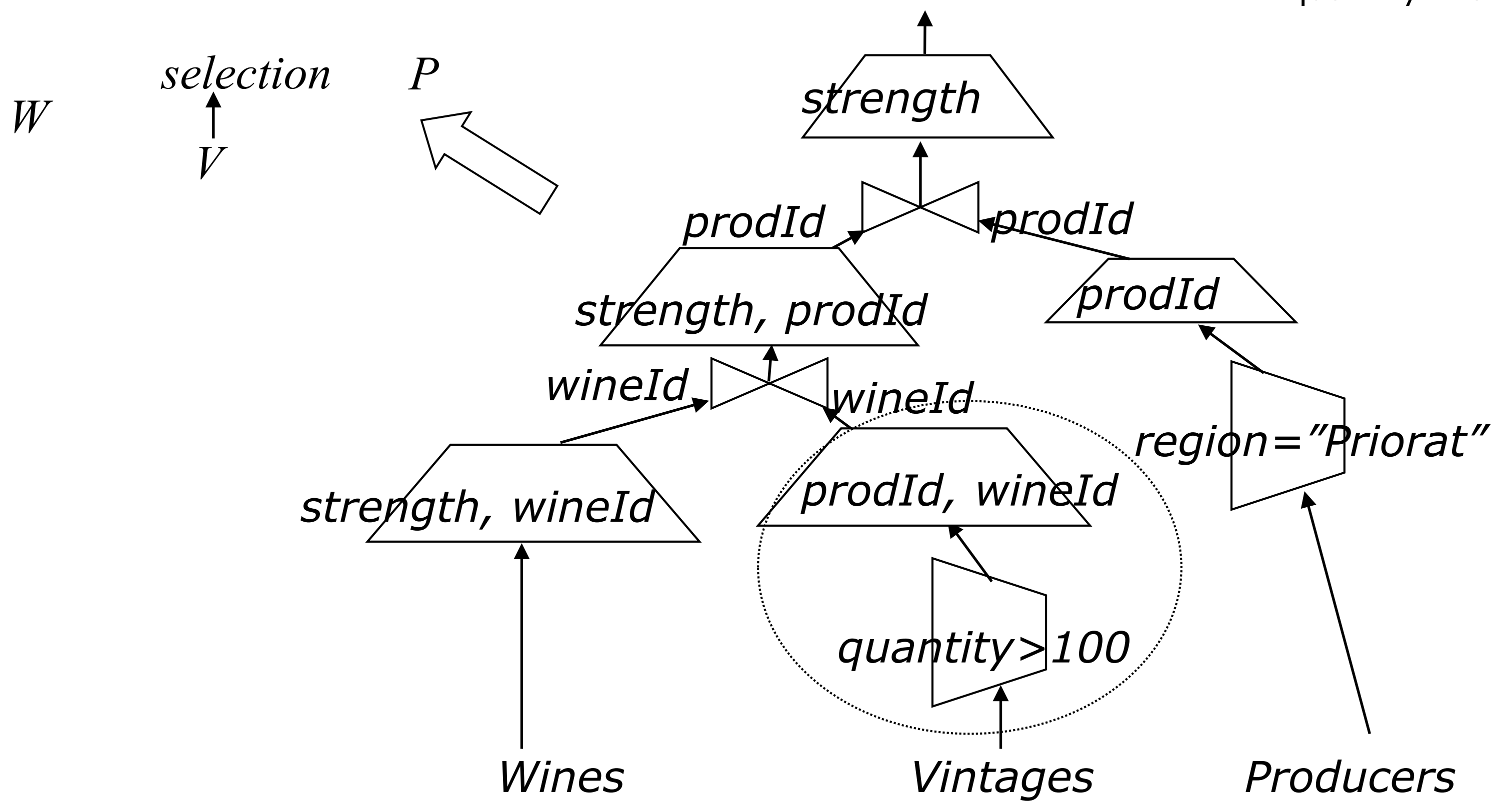
Generating the Process Tree (Recap)

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```



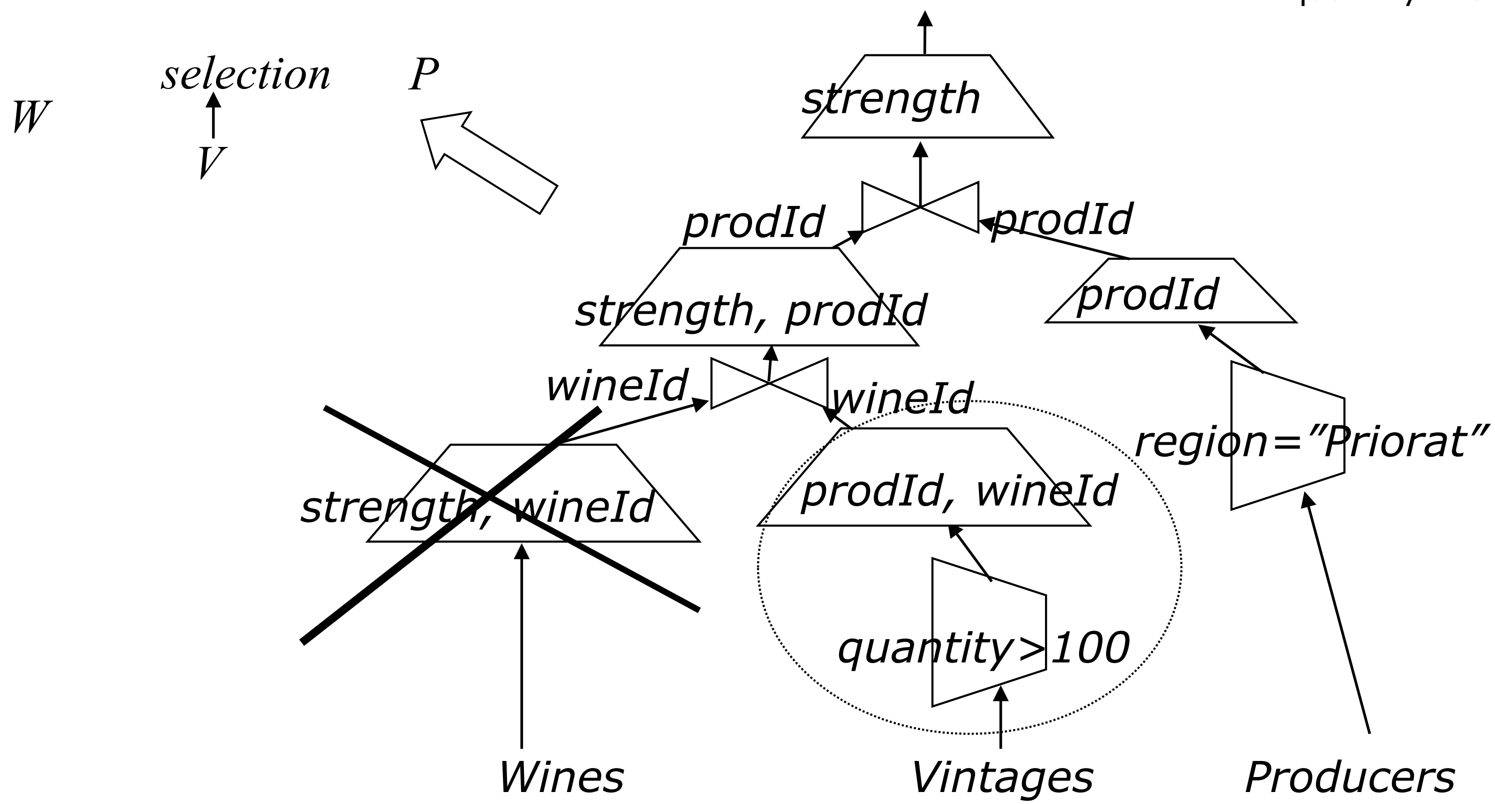
Generating the Process Tree (Recap)

SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
AND p.prodId=v.prodId
AND p.region="Priorat"
AND v.quantity>100;



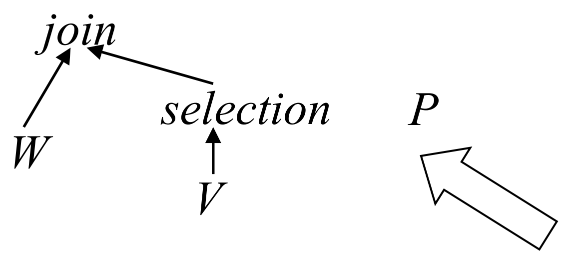
Generating the Process Tree (Recap)

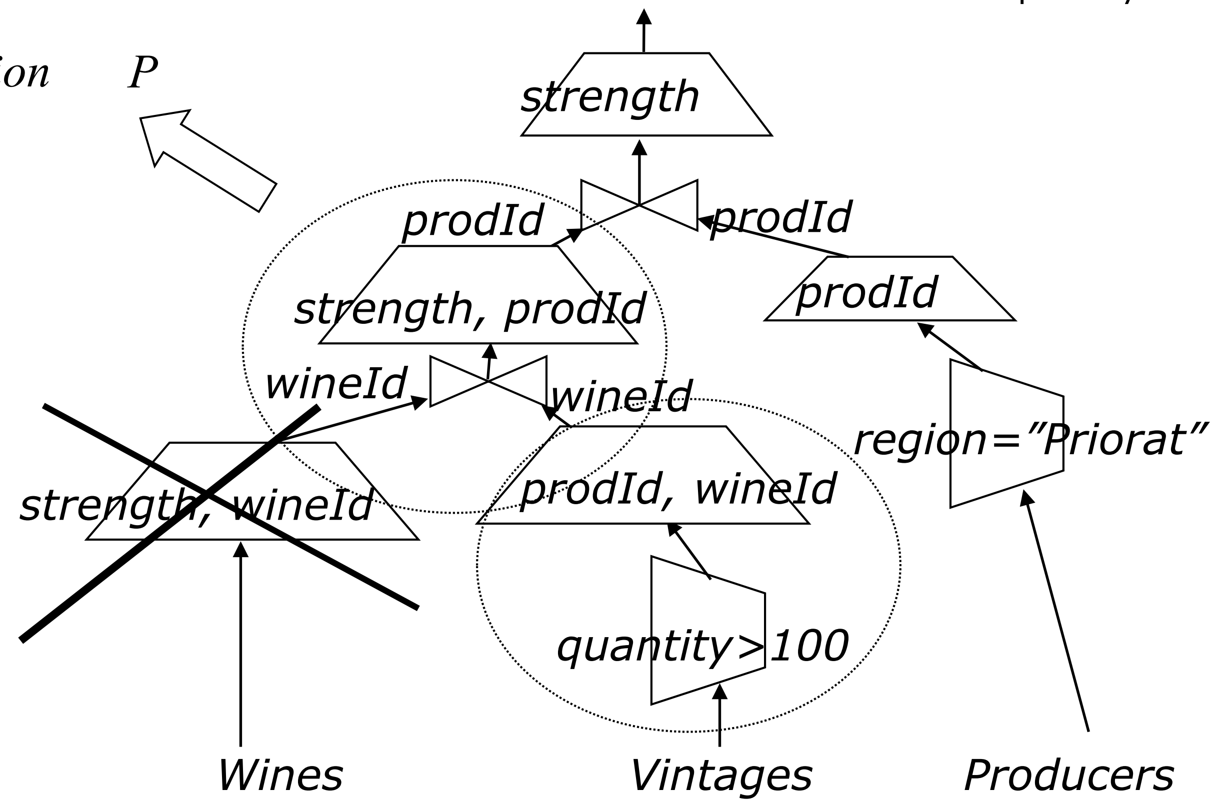
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
AND p.prodId=v.prodId
AND p.region=”Priorat”
AND v.quantity>100;



Generating the Process Tree (Recap)

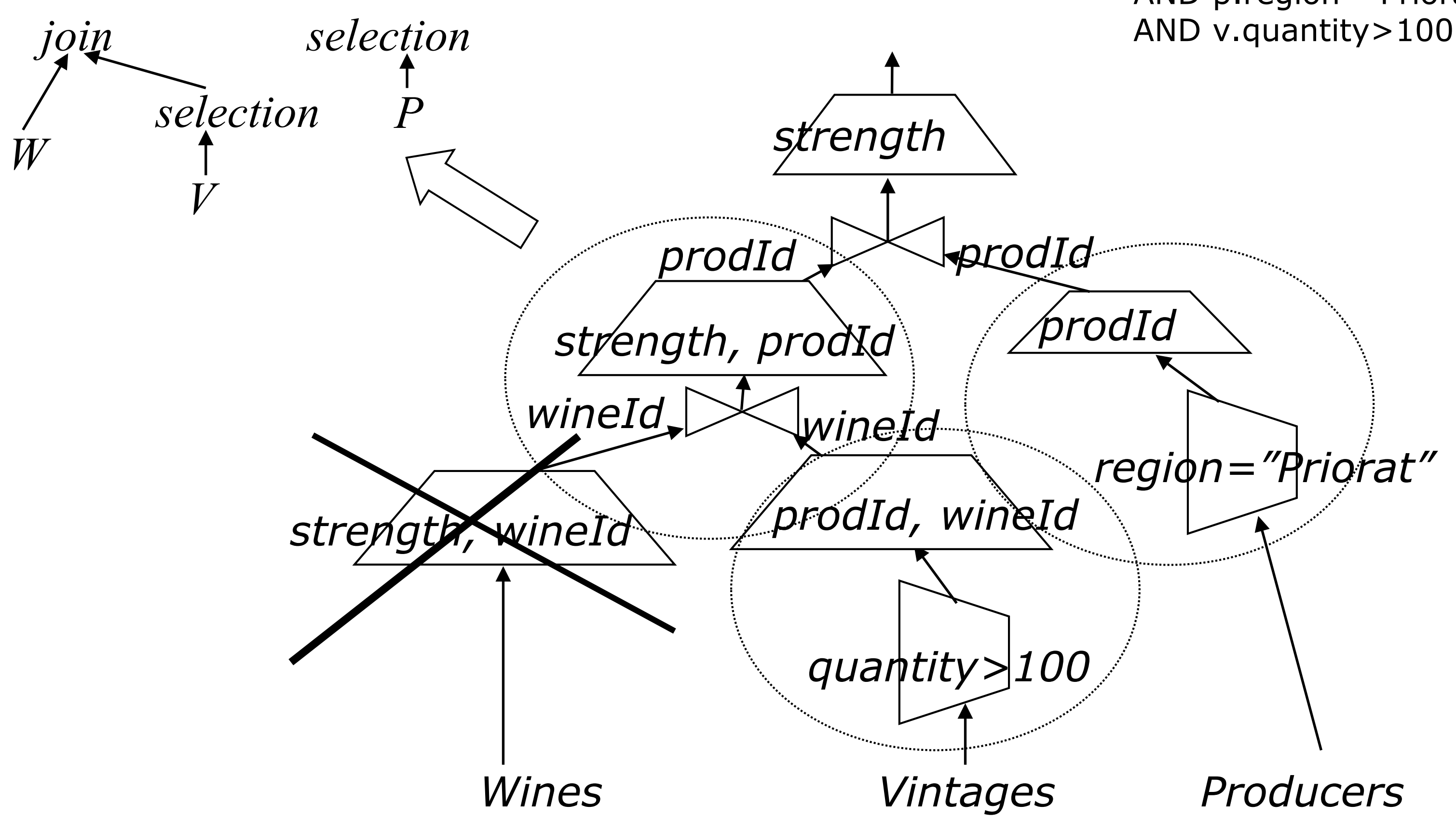
```sql
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region=”Priorat”
        AND v.quantity>100;
```





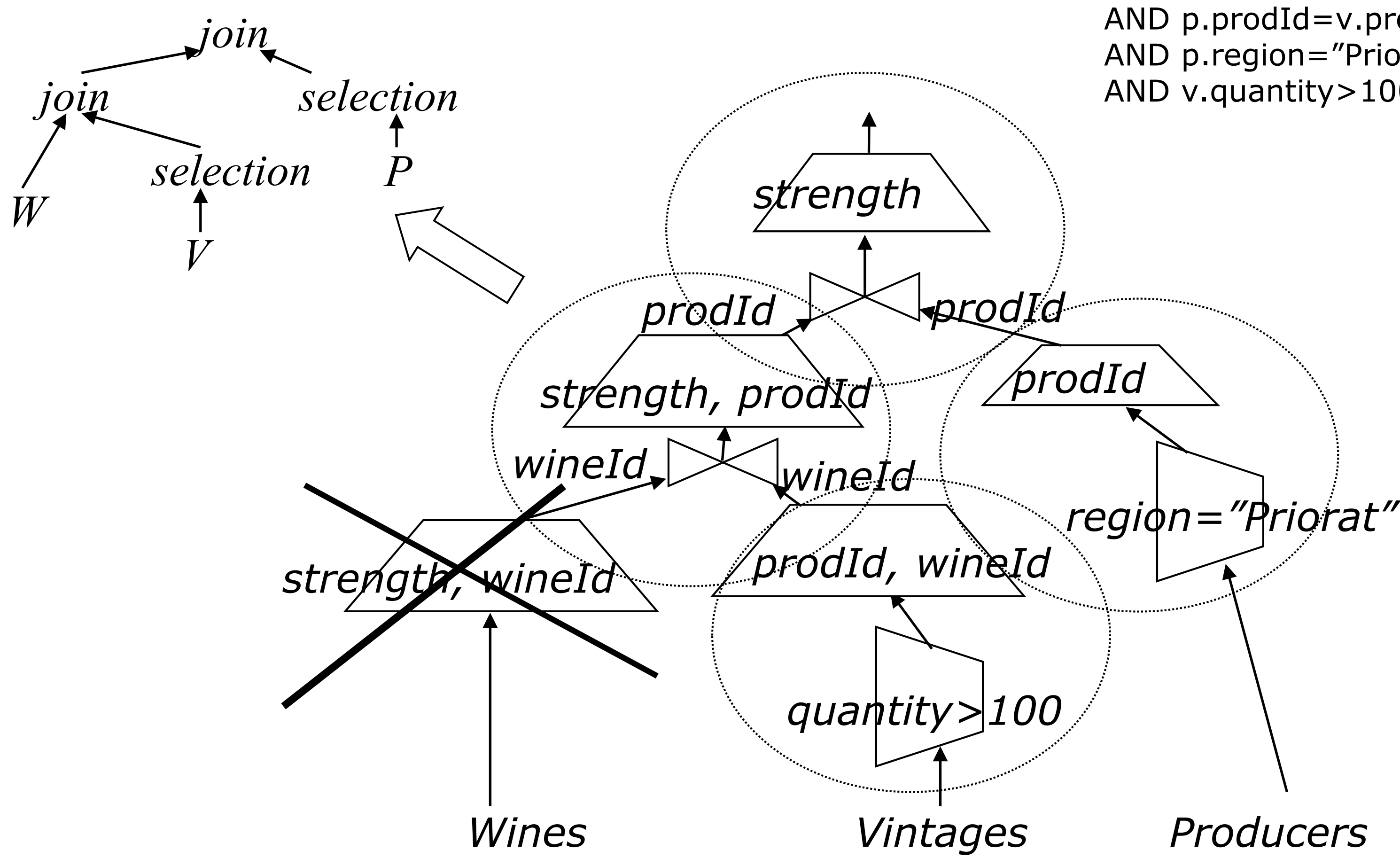
Generating the Process Tree (Recap)

SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
AND p.prodId=v.prodId
AND p.region="Priorat"
AND v.quantity>100;



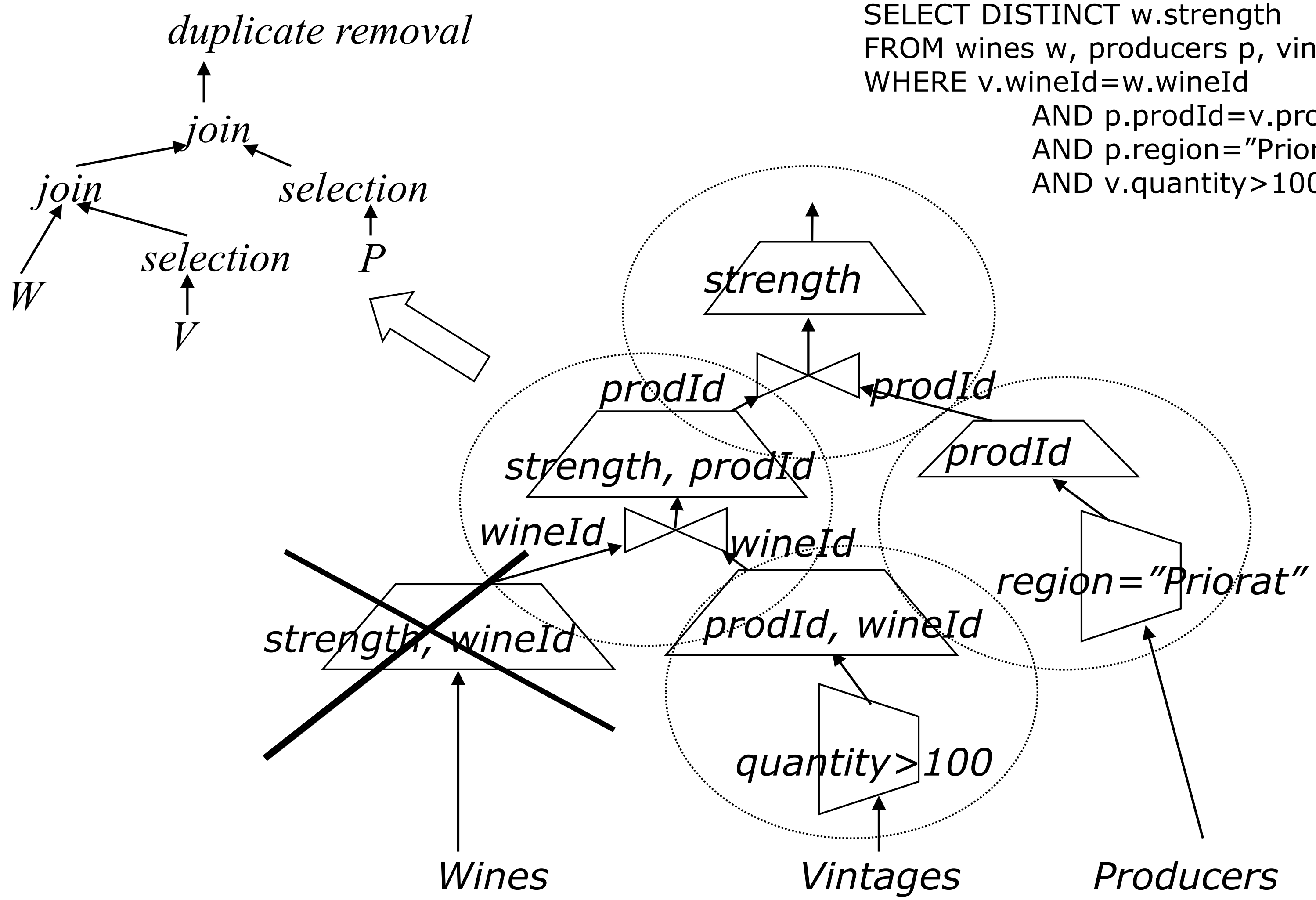
Generating the Process Tree (Recap)

SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
AND p.prodId=v.prodId
AND p.region="Priorat"
AND v.quantity>100;



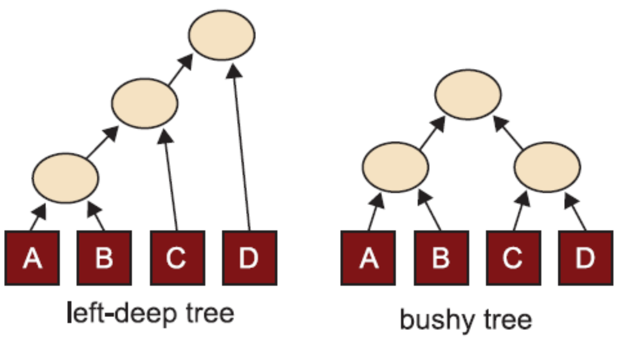
Generating the Process Tree (Recap)

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
      AND p.prodId=v.prodId
      AND p.region="Priorat"
      AND v.quantity>100;
```



Generation of Execution Alternatives

- Execution Order
  - Left or right deep trees
  - Bushy trees
    - Those where both operands can be intermediate results



This typically cannot be chosen and it is embedded in the processing engine

For example:

- Bushy Tree: MapReduce, Spark
- Right-Deep tree: Aggregation Framework (MongoDB)

Generation of Execution Alternatives

- Site selection
  - Unary operators: operations over replicated fragments can be executed in any of the replicas
  - Binary operators: if both fragments are not co-located, one needs to be shipped through the network. Different criteria to choose which one to send:
    - Comparing size of the relations
    - Joins
      - Semi-join strategy
    - In general, it is more difficult for multi-way joins
      - Size of the intermediate joins must be considered

Activity: Distributed Physical Optimization

- Objective: Recognize the computationally expensive nature of distributed physical optimization
- Tasks:
  1. (5') By pairs, answer the following questions:
     I. Compute the fragment query (data location stage) for the database setting and query below
     II. Generate all the alternative process trees (considering site selection) you can figure out
  2. (5') Discussion

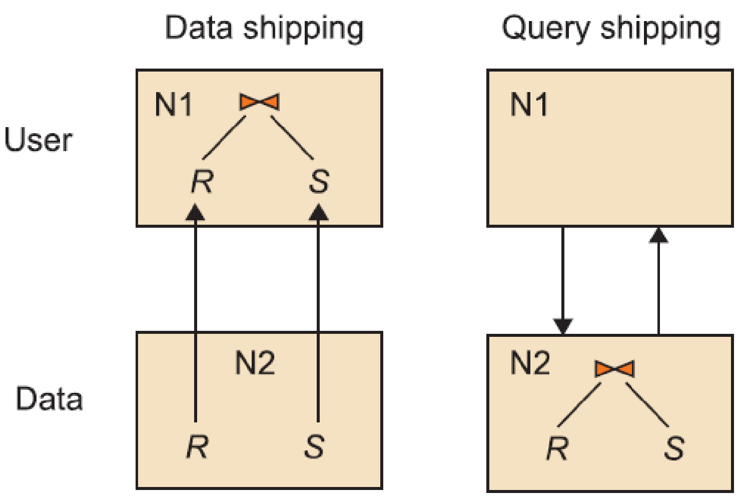
Example:

- A distributed database with 5 sites (i.e., database nodes): S_1, S_2, S_3, S_4 and S_5.
- 3 relations in the database R, S and T.
- Each relation is horizontally fragmented in two fragments (we refer to them by the name of the relation and a subindex, for example: R_1, R_2). You can consider them to be correct (i.e., complete, disjoint and reconstructible).
- Each fragment is replicated at all 5 sites.

Suppose now that the following query is issued in S_3: $Q_1 = \sigma(R) \bowtie \sigma(S) \bowtie T$

Site Selection

- In general, if we decide to send one fragment over the network, we incurr in **data shipping**
  - The data is retrieved from the stored site to the site executing the query
    - Avoid bottlenecks on frequently used data
- But **query shipping** is also an option
  - The evaluation of the query is delegated to the site where it is stored
    - Ideal for unary operators
    - Avoids transferring large amount of data
- Hybrid strategies



The Semi-join Strategy (for joins)

- Different strategies according to the operator
  - Semi-join strategy
- Example:
  - $\underset{S_1}{R} \bowtie \underset{S_2}{S}$

The Semi-join Strategy (for joins)

- Different strategies according to the operator
  - Semi-join strategy
- Example:
  - $\underset{S_1}{R} \bowtie \underset{S_2}{S} \begin{cases} (R \ltimes S) \bowtie S & \textit{, better if } B_R > B_{S[Jattr]} + B_{R \ltimes S} \\ \end{cases}$

The Semi-join Strategy (for joins)

- Different strategies according to the operator
  - Semi-join strategy
- Example:
  - $$\underset{S_1}{R} \bowtie \underset{S_2}{S} \begin{cases} (R \ltimes S) \bowtie S & \text{, better if } B_R > B_{S[Jattr]} + B_{R \ltimes S} \\ (S \ltimes R) \bowtie R & \text{, better if } B_S > B_{R[Jattr]} + B_{R \rtimes S} \end{cases}$$

The Semi-join Strategy (for joins)

- Different strategies according to the operator
  - Semi-join strategy
- Example:
  - $$\underset{S_1}{R} \bowtie \underset{S_2}{S} \begin{cases} (R \ltimes S) \bowtie S & \text{, } \textit{better if } B_R > B_{S[Jattr]} + B_{R \ltimes S} \\ (S \ltimes R) \bowtie R & \text{, } \textit{better if } B_S > B_{R[Jattr]} + B_{R \rtimes S} \\ (R \ltimes S) \bowtie (S \ltimes R) & \text{, } \textit{better if } \ldots \end{cases}$$

The Semi-join Strategy (for joins)

- Different strategies according to the operator
  - Semi-join strategy
- Example:
  - $$\underset{S_1}{R} \bowtie \underset{S_2}{S} \begin{cases} (R \ltimes S) \bowtie S & \text{, } \textit{better if } B_R > B_{S[Jattr]} + B_{R \ltimes S} \\ (S \ltimes R) \bowtie R & \text{, } \textit{better if } B_S > B_{R[Jattr]} + B_{R \rtimes S} \\ (R \ltimes S) \bowtie (S \ltimes R) & \text{, } \textit{better if } \ldots \end{cases}$$
- 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

Algorithms to Process the Query Operators

- The resulting plan must benefit of **parallelism** by exploiting, as much as posible, the **distributed data**
  - Without distribution, no parallelism. But distributed data does not guarantee distribution (!)

Parallel Query Processing

- Employ parallel hardware effectively (i.e., reduce the response time). Strategies:
  - Process pieces in different processors
  - Serial algorithms adapted to multi-thread environments
  - Divide input data set into <u>disjoint</u> 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 (*multi-core server*)
    - Unary
      - Static partitioning
    - Binary
      - Static or dynamic partitioning
  - Inter-operator
    - Independent (*parallel branches of the process tree*)
    - Pipelined (*within the same branch*)
      - Demand driven (pull)
      - Producer driven (push)

Demand-Driven Pipelining

- Each operator supports:
  - Open
  - Next
  - Close
- In principle, not parallel
  - Parent requests activate the execution
  - Nevertheless, a buffer can be used
    - This is similar to producer-driven

Producer-Driven Pipelining

- Generate output tuples eagerly
  - Until the buffers become full
- Pipeline stalls when an operator becomes ready and no new inputs are available
  - It is propagated upwards through the pipeline like a bubble

		Latency	Occupancy
Serial		T	T
Parallel	No stalls	T'(<T)	T/N
	Stalls	T'+k·N	T/N+k

N = operators
T = time units required for the whole query
Latency = time to process the query
Occupancy = time until it can accept more work
k = delay imposed by imbalance and overhead

Estimating each Plan Cost

- Response Time (latency)
  - 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:

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

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

Summary

- Kinds of distributed DBMSs
  - Parallel DB Systems
- Distributed Query Processing
- Global optimization
- Parallel Query Processing
  - Kinds of parallelism
- Evaluating distributed plans
  - Extended cost models
  - Execution plans evaluation

Bibliography

- M.T. Özsu and P. Valduriez. *Principles of distributed database systems*. Second edition. Prentice Hall, 1999
- G. Graefe. *Query Evaluation Techniques*. In ACM Computing Surveys, 25(2), June 1993
- L. Liu, M.T. Özsu (Eds.). Encyclopedia of Database Systems. Springer, 2009