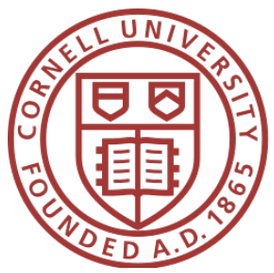
Query Optimization for Data Analysis

Immanuel Trummer



Cornell University

Data Analysis

Astronomy

Biology

Physics

Retail

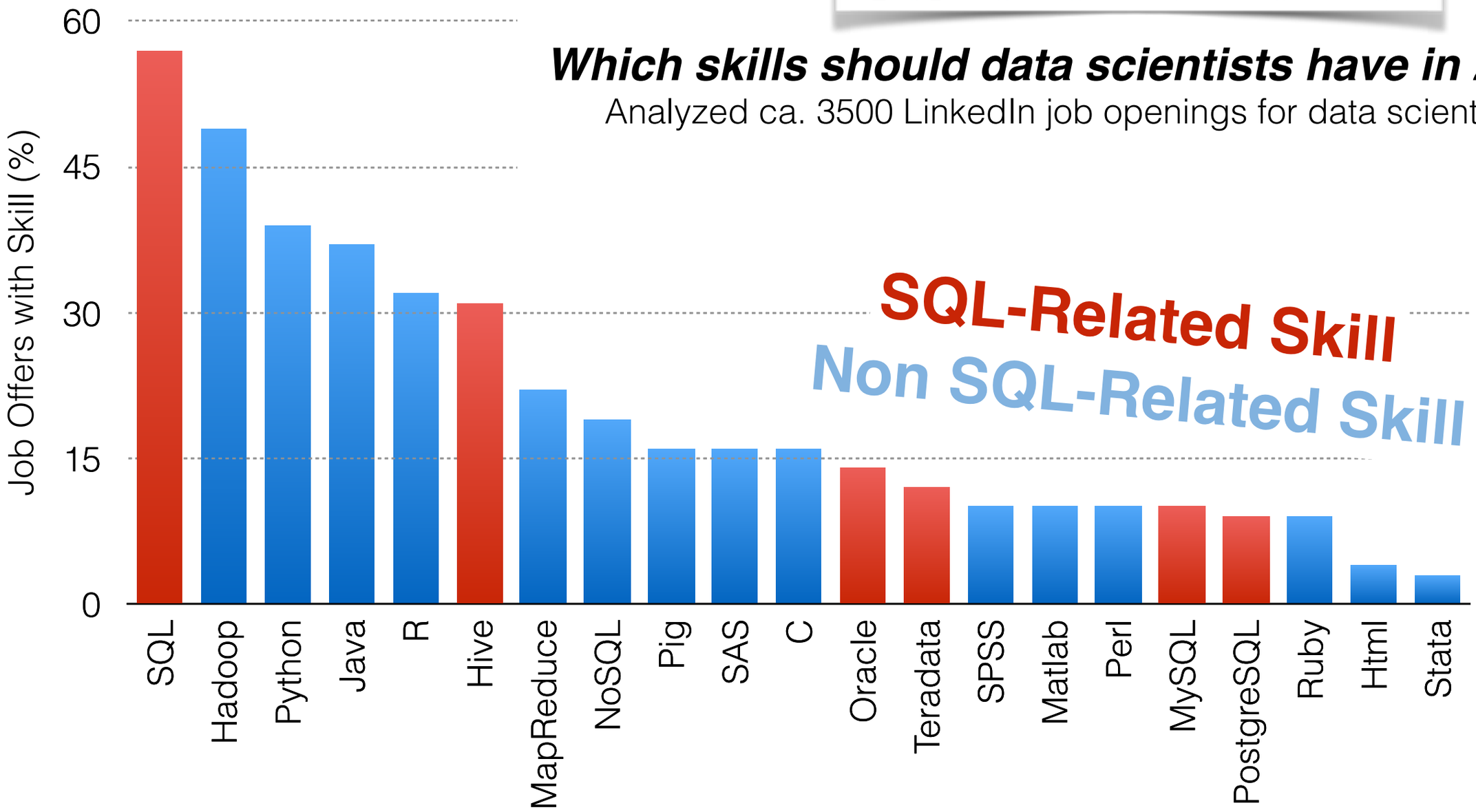
Politics

Popular Data Analysis Skills

CrowdFlower

Which skills should data scientists have in 2016?

Analyzed ca. 3500 LinkedIn job openings for data scientists.



http://www.crowdflower.com/blog/what-skills-should-data-scientists-have-in-2016

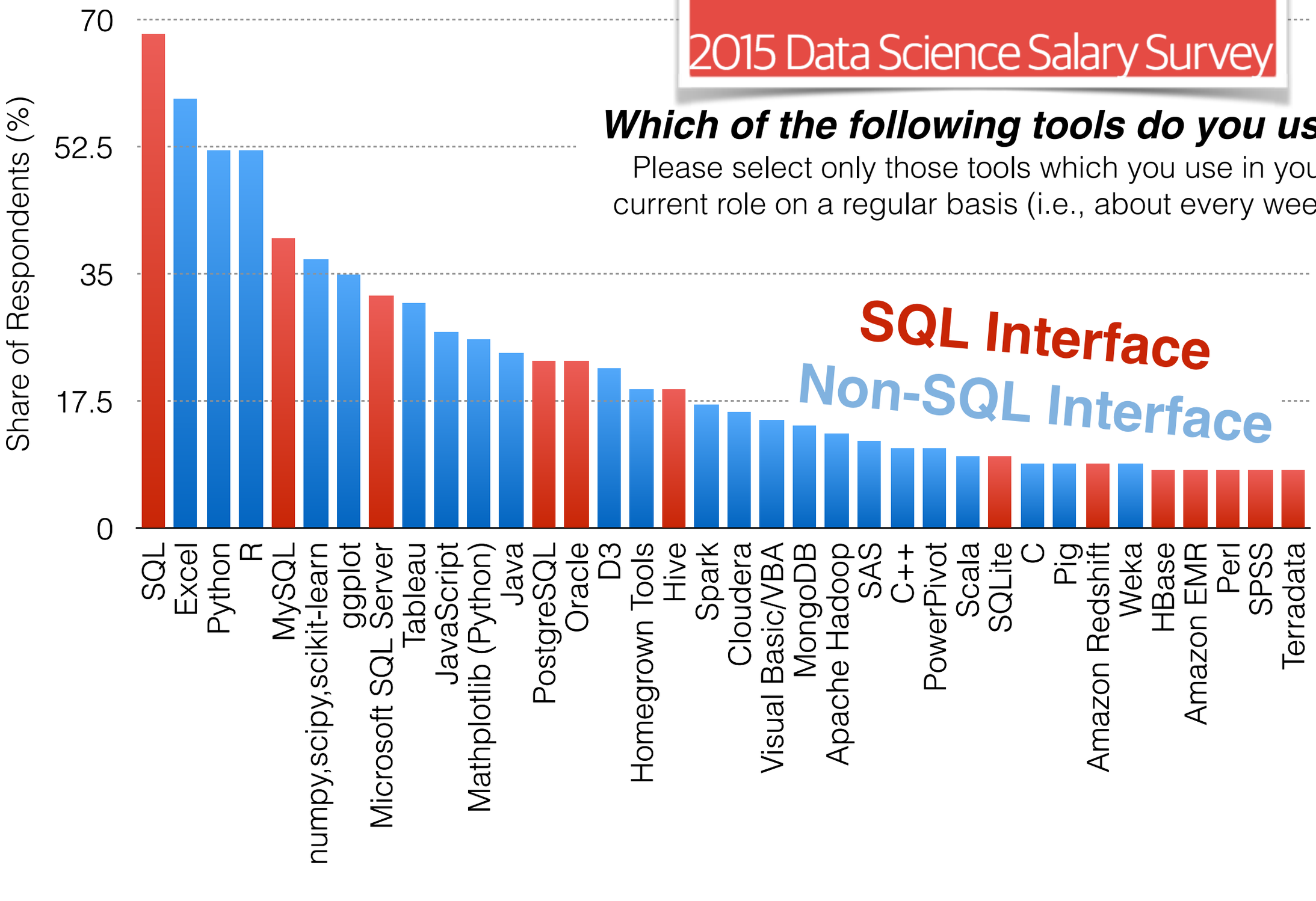
Popular Data Analysis Tools

O'REILLY®

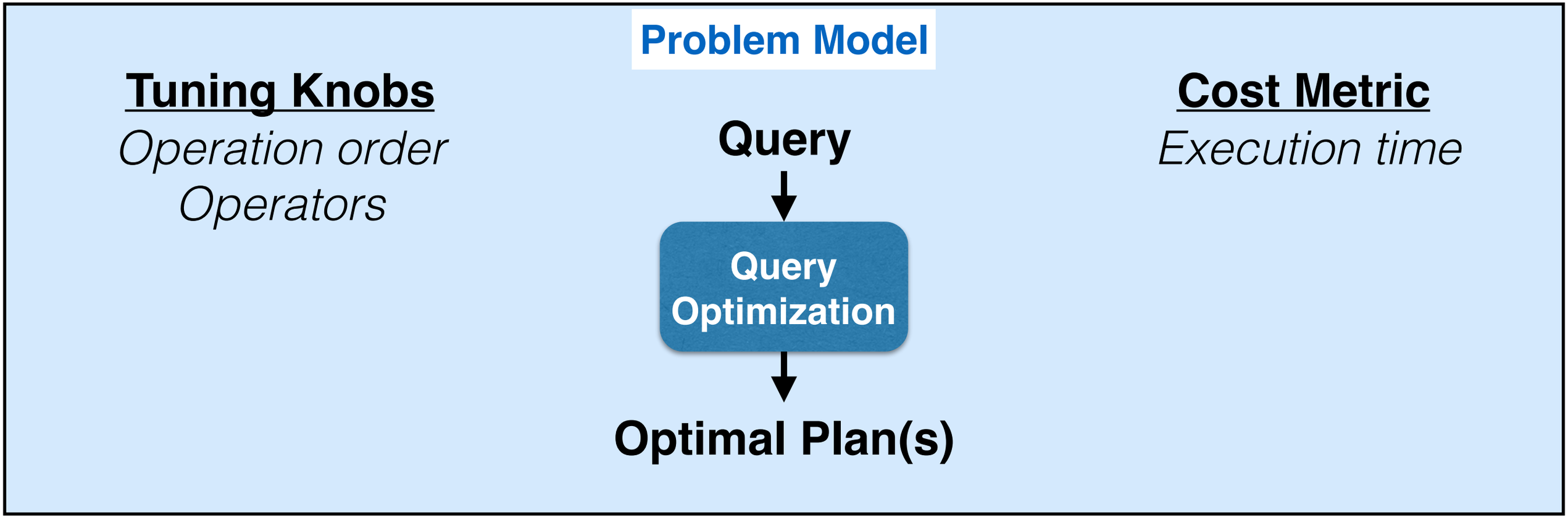
2015 Data Science Salary Survey

Which of the following tools do you use?

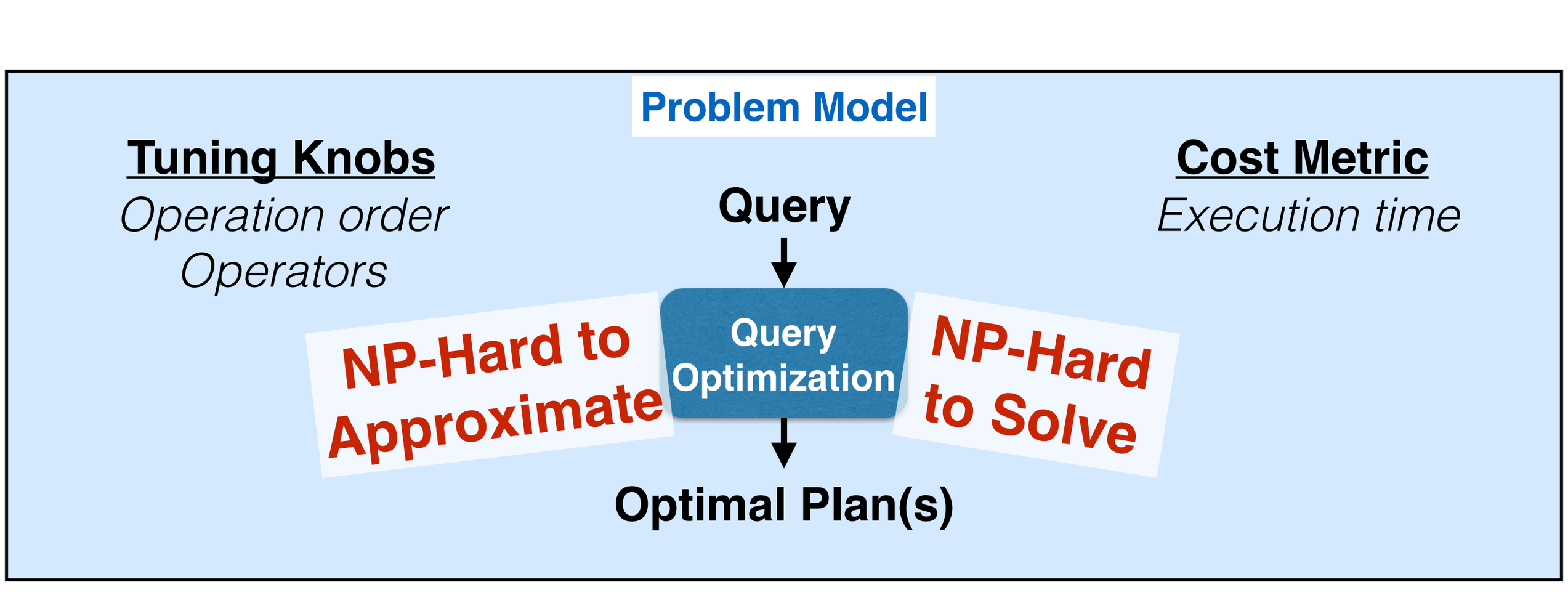
Please select only those tools which you use in your current role on a regular basis (i.e., about every week).



Classical Query Optimization



Classical Query Optimization



Classical Query Optimization

Context in *1979*

Fixed processing platform, few operators available

Optimization Context

Problem Model

Tuning Knobs

Operation order
Operators

Cost Metric

Execution time

Query

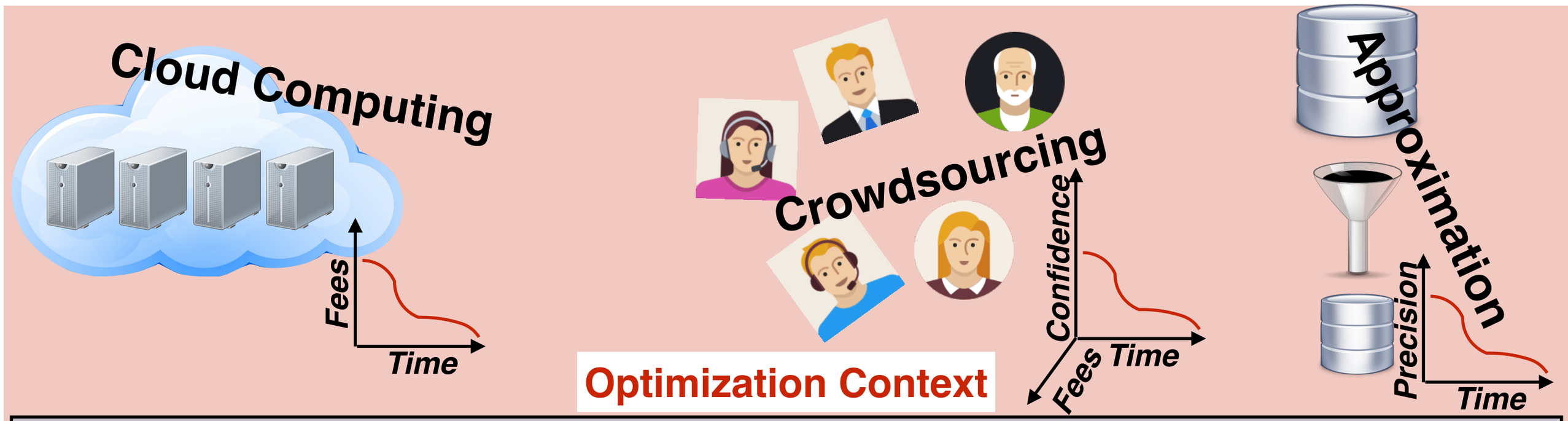
Query Optimization

NP-Hard to Approximate

NP-Hard to Solve

Optimal Plan(s)

Modern Query Optimization



Problem Model

Tuning Knobs

Operation order

Operators

Cost Metric

Execution time

Query

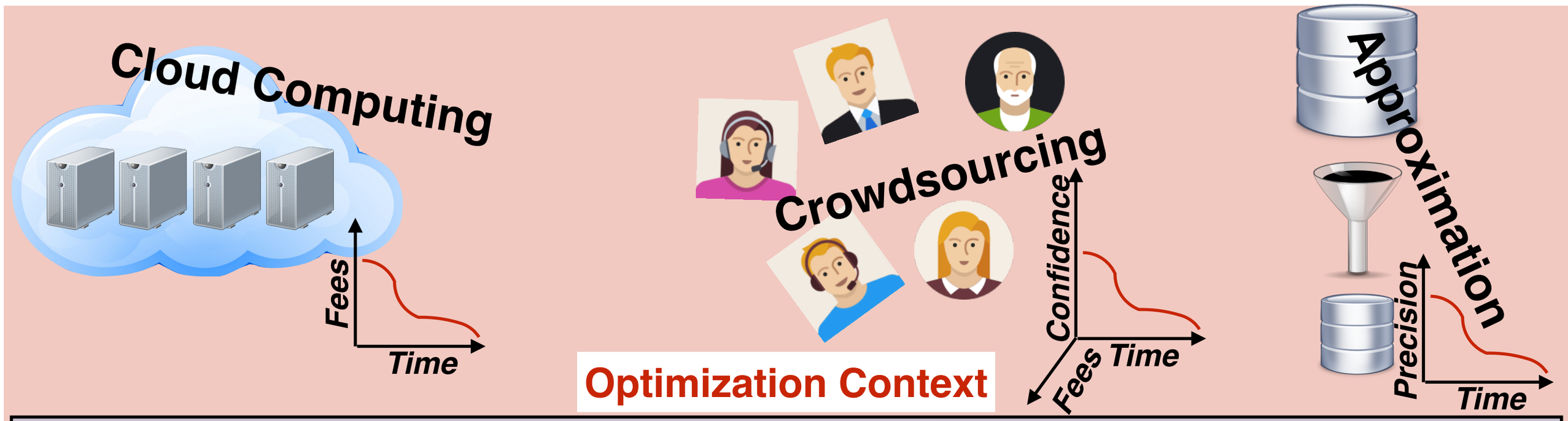
Query Optimization

NP-Hard to Approximate

NP-Hard to Solve

Optimal Plan(s)

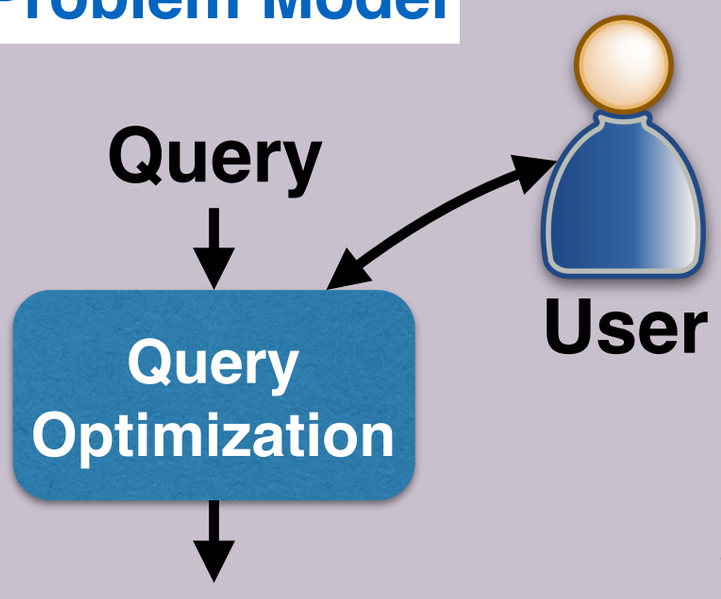
Modern Query Optimization



Problem Model

Tuning Knobs

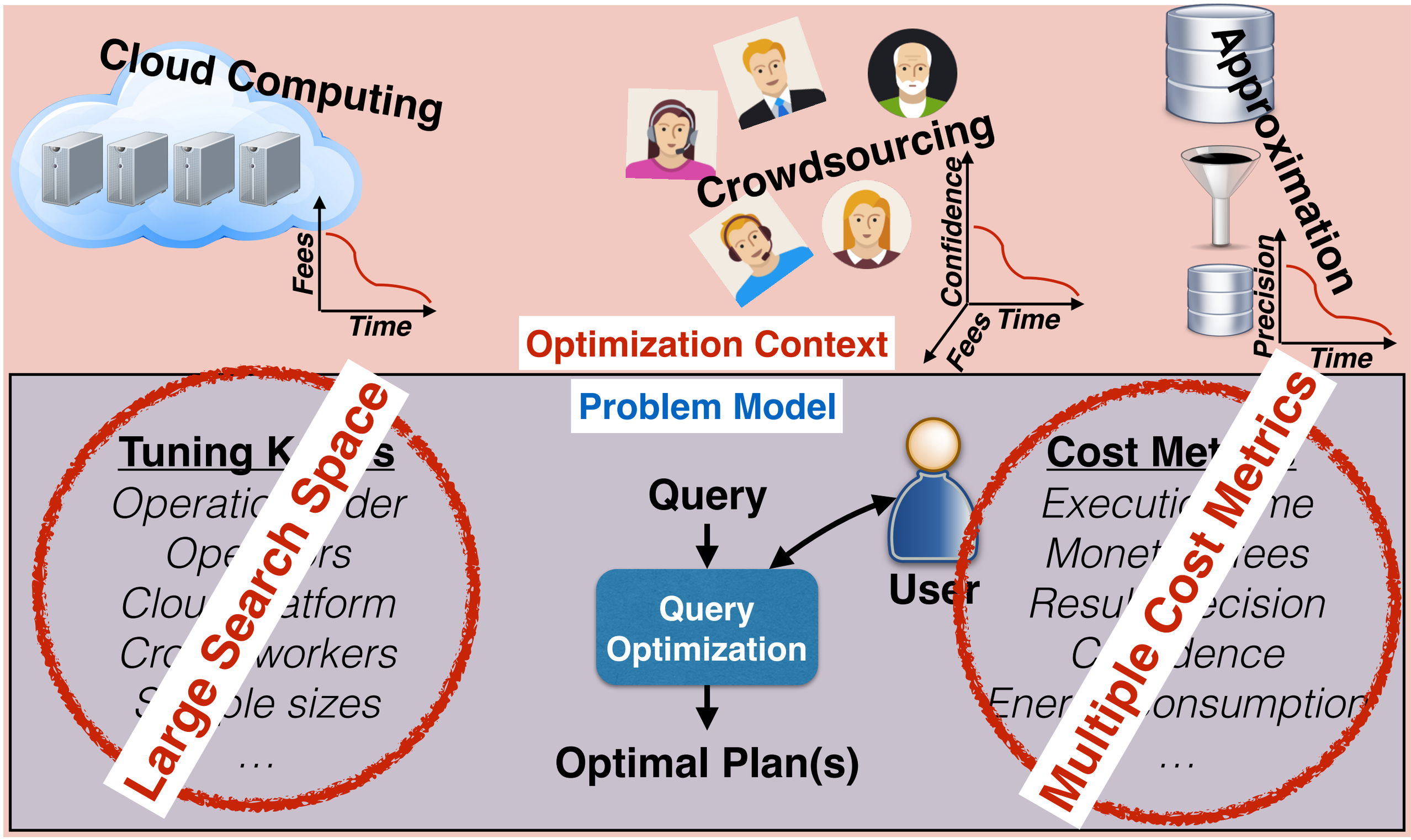
Operation order
Operators
Cloud platform
Crowd workers
Sample sizes
…



Cost Metrics

Execution time
Monetary fees
Result precision
Confidence
Energy consumption
…

Modern Query Optimization



Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

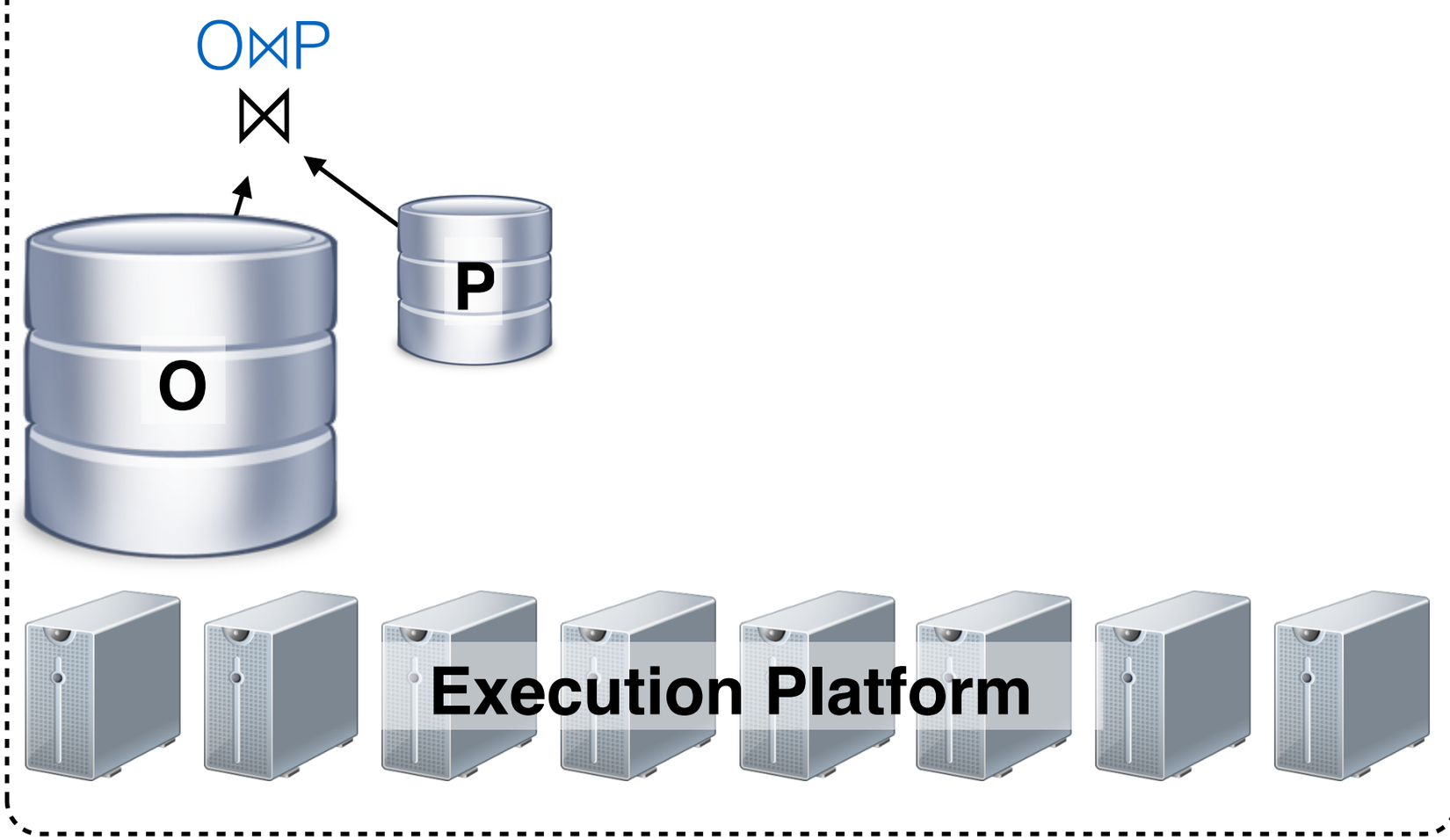
Query Plan

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan

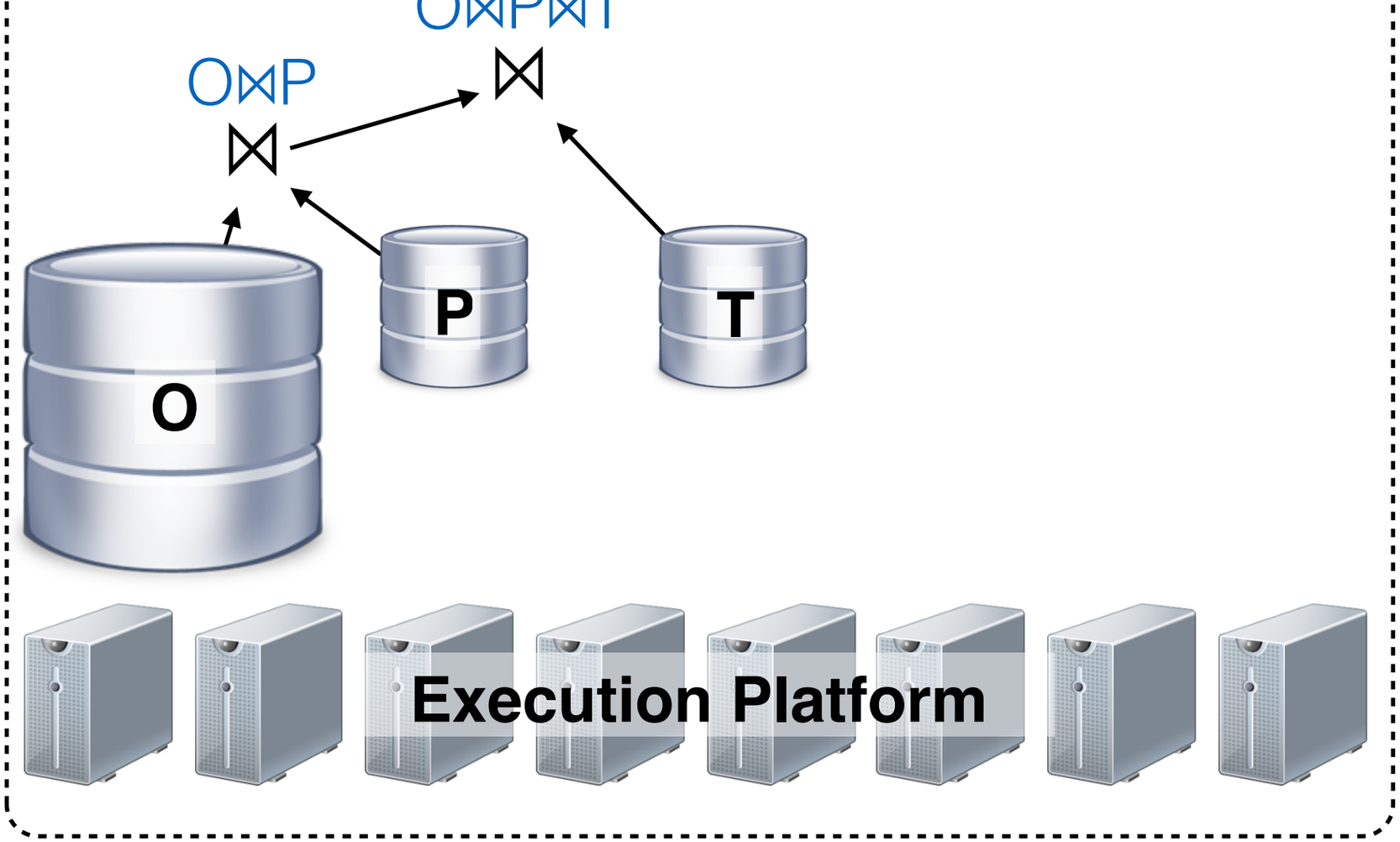


Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan

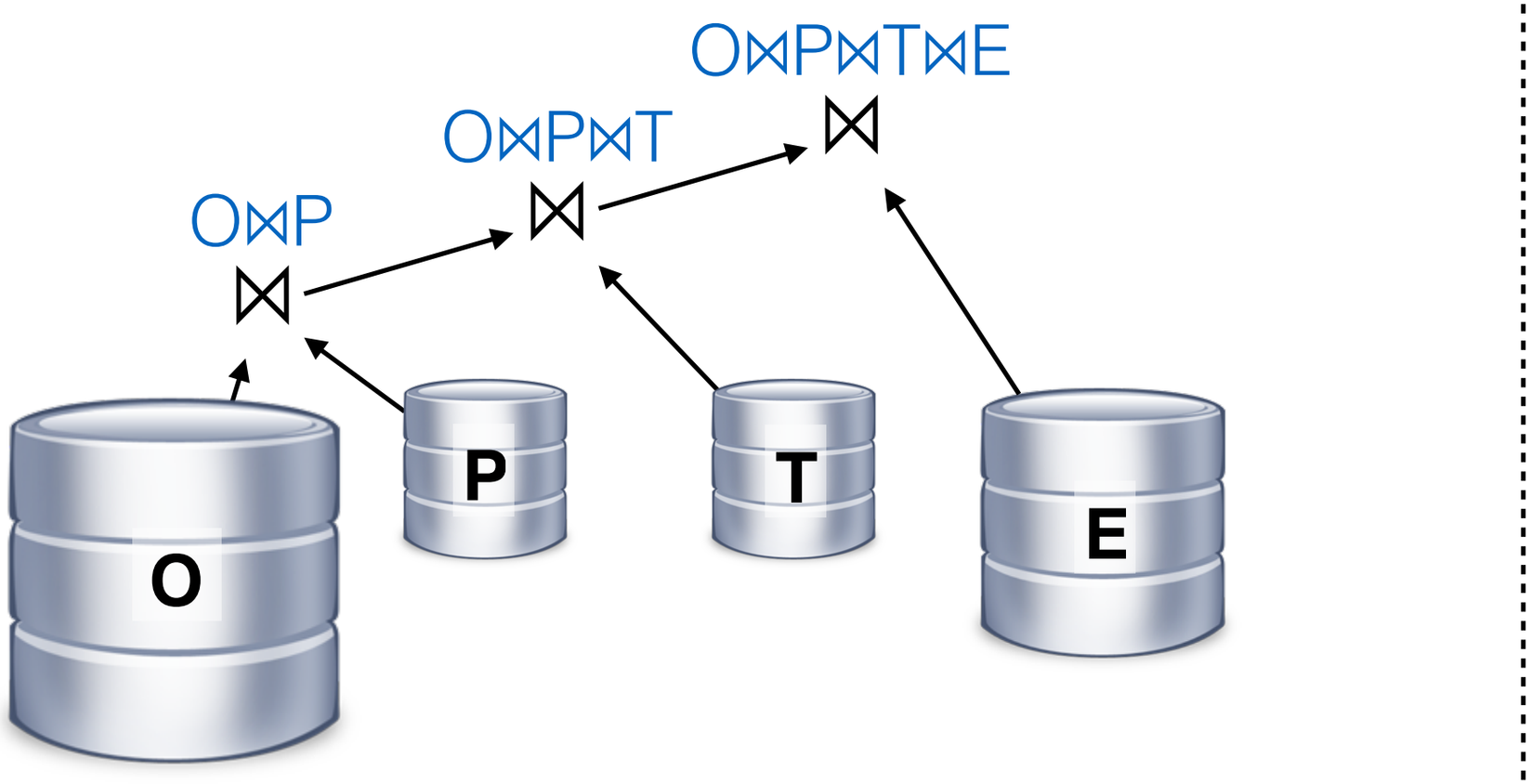


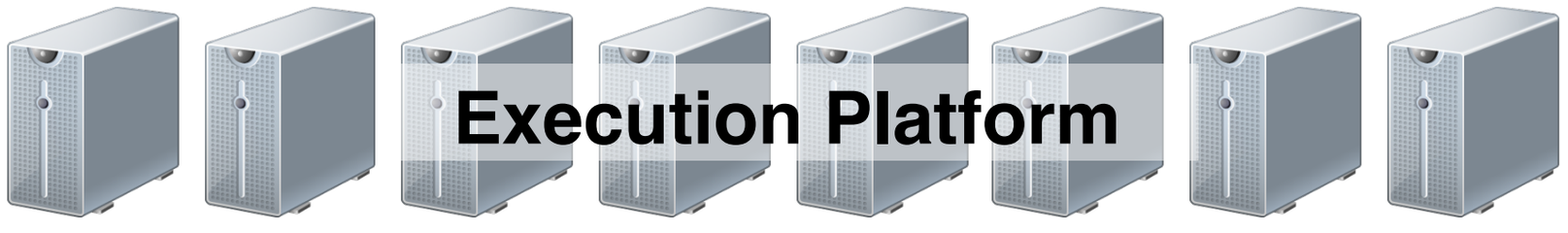
Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



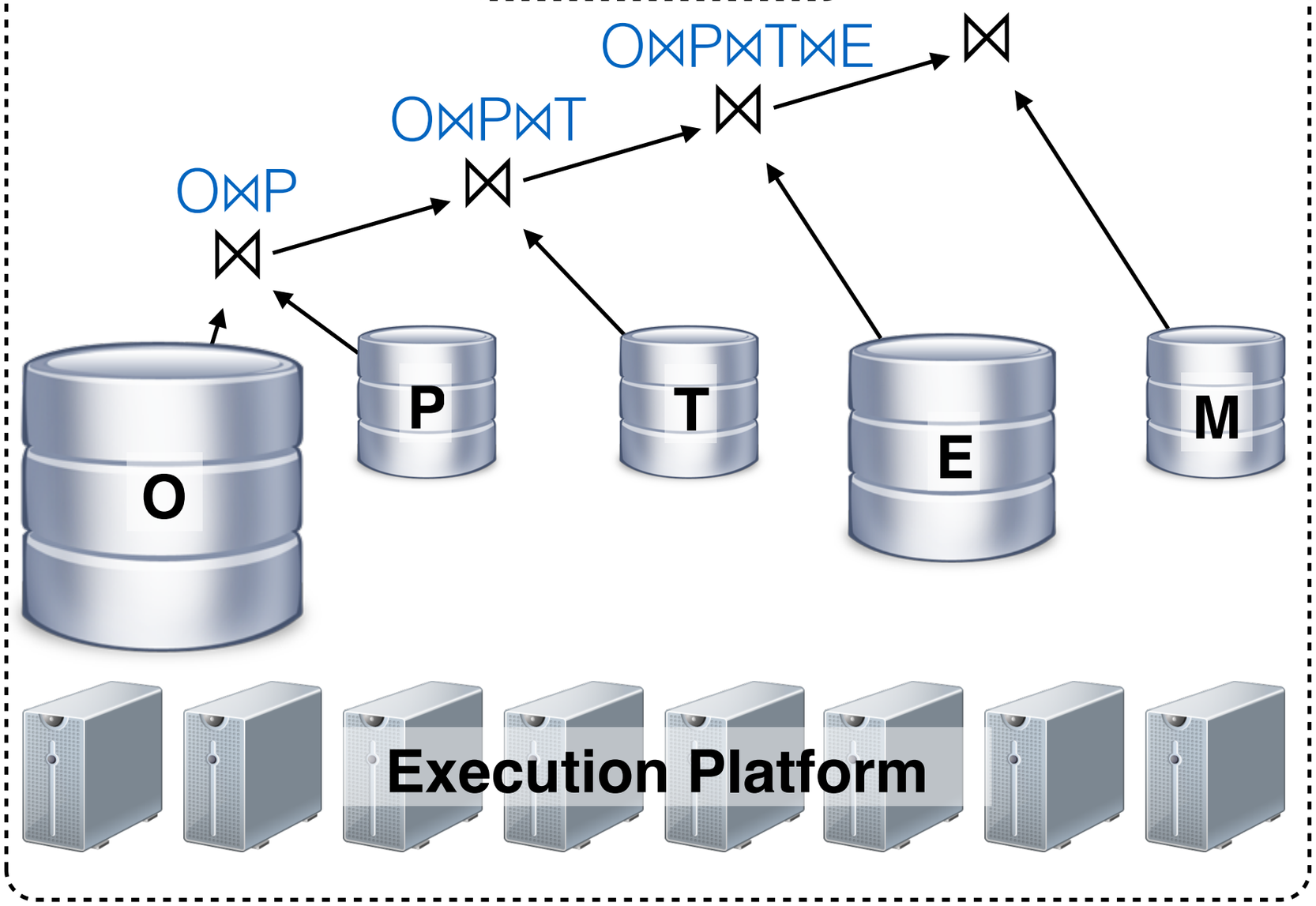


Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan

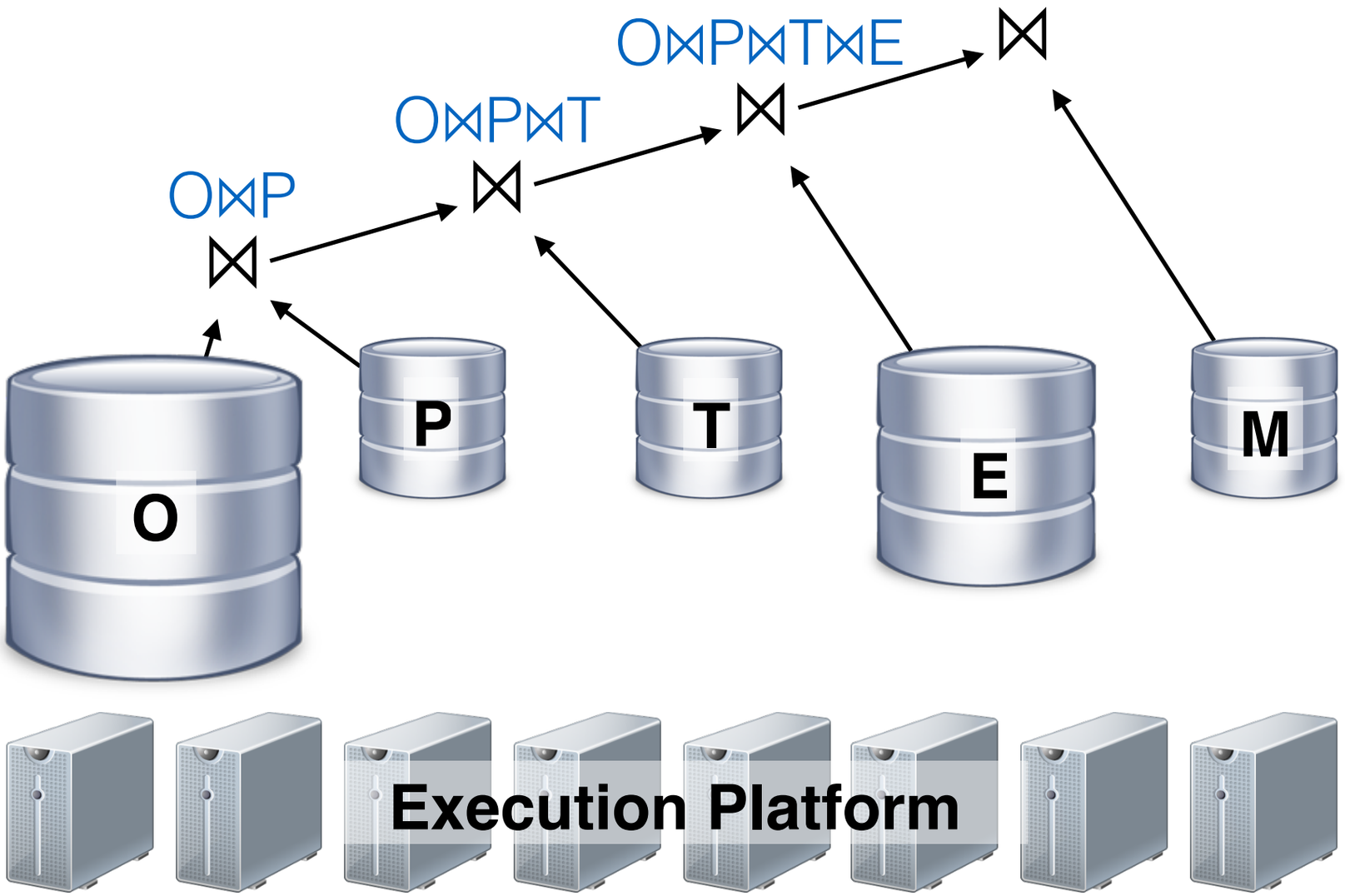


Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Cost

Time:

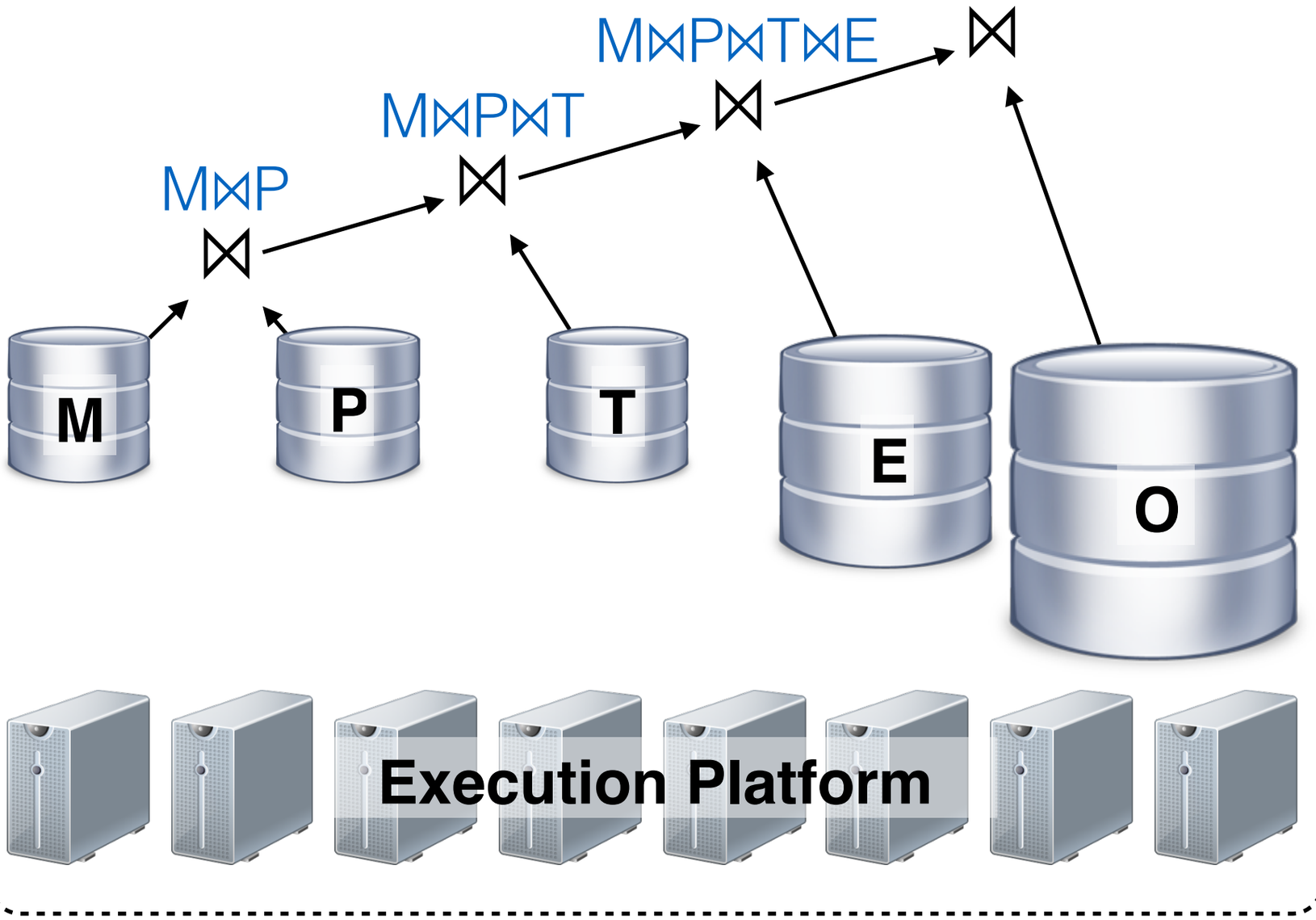
1 day

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Cost

Time:

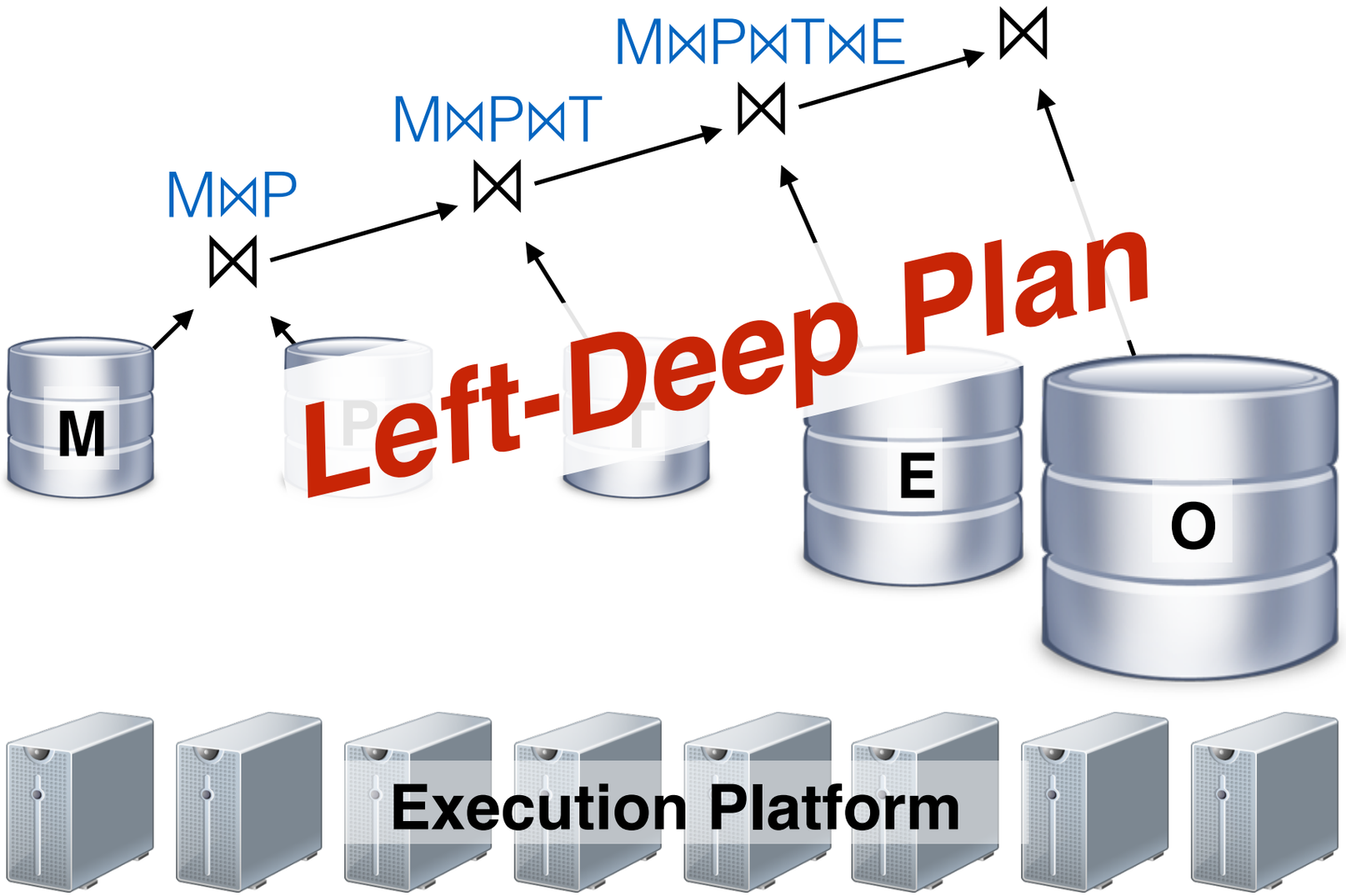
4 hours

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Cost

Time:

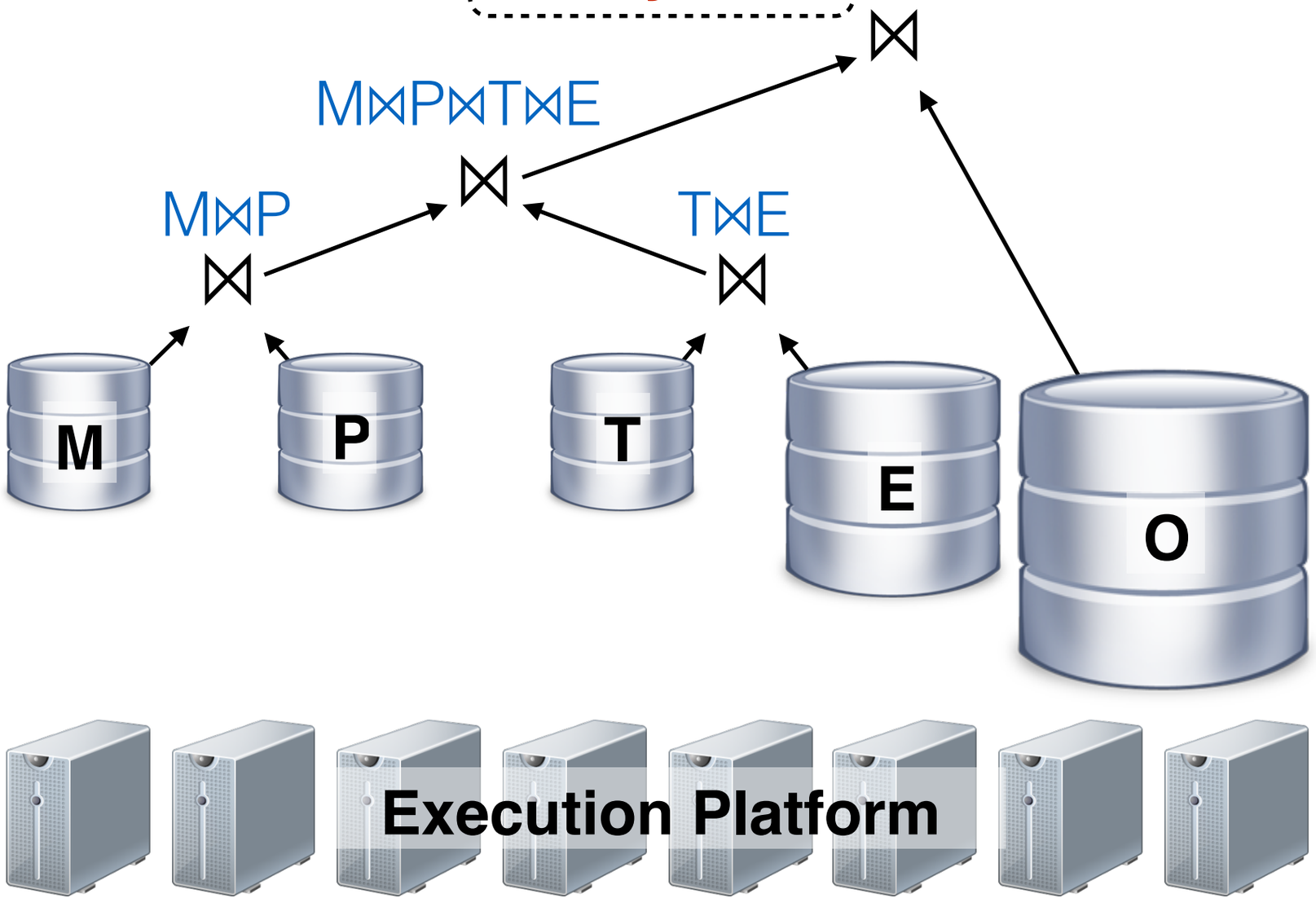
4 hours

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Cost

Time:

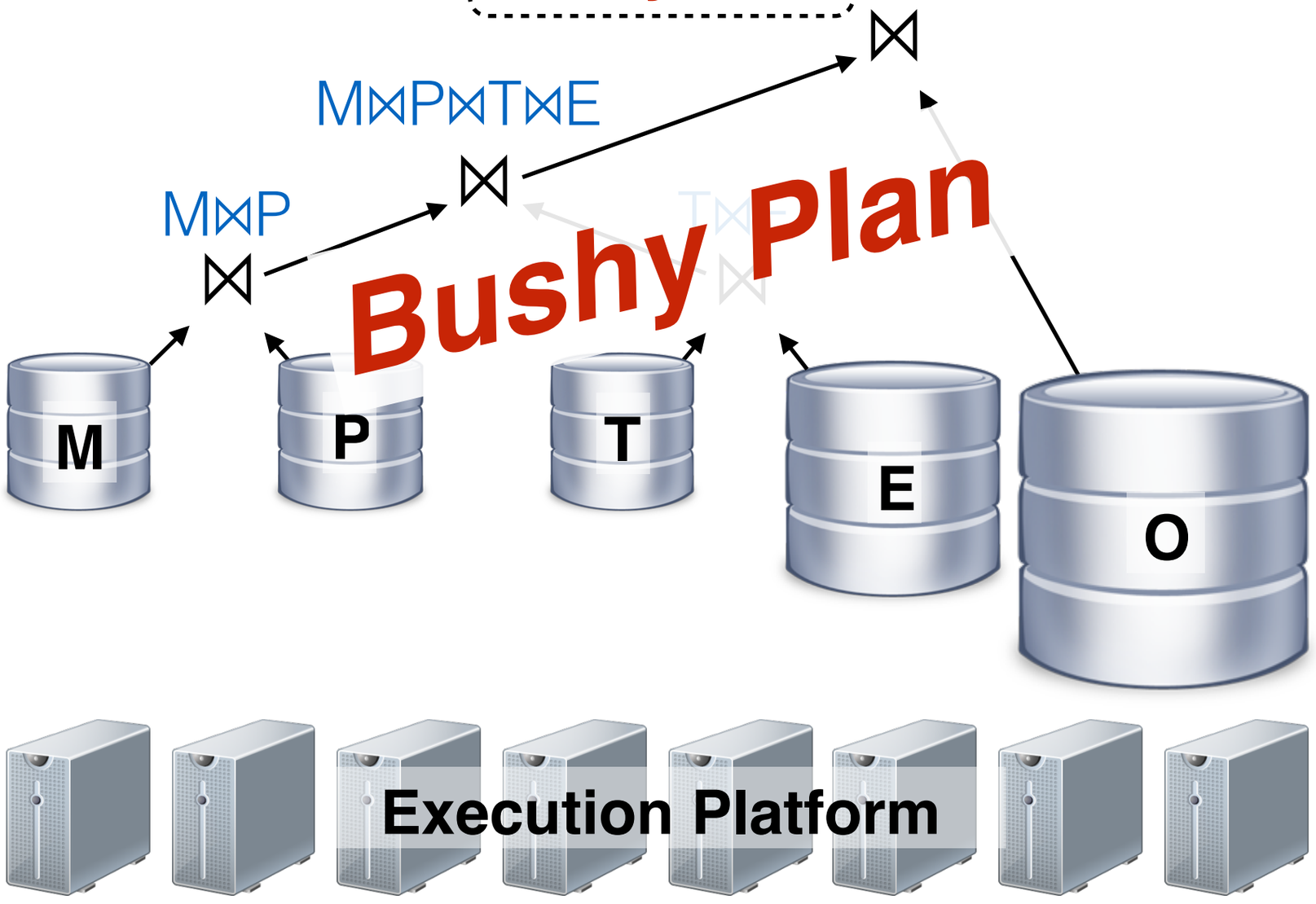
3 hours

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Cost

Time:

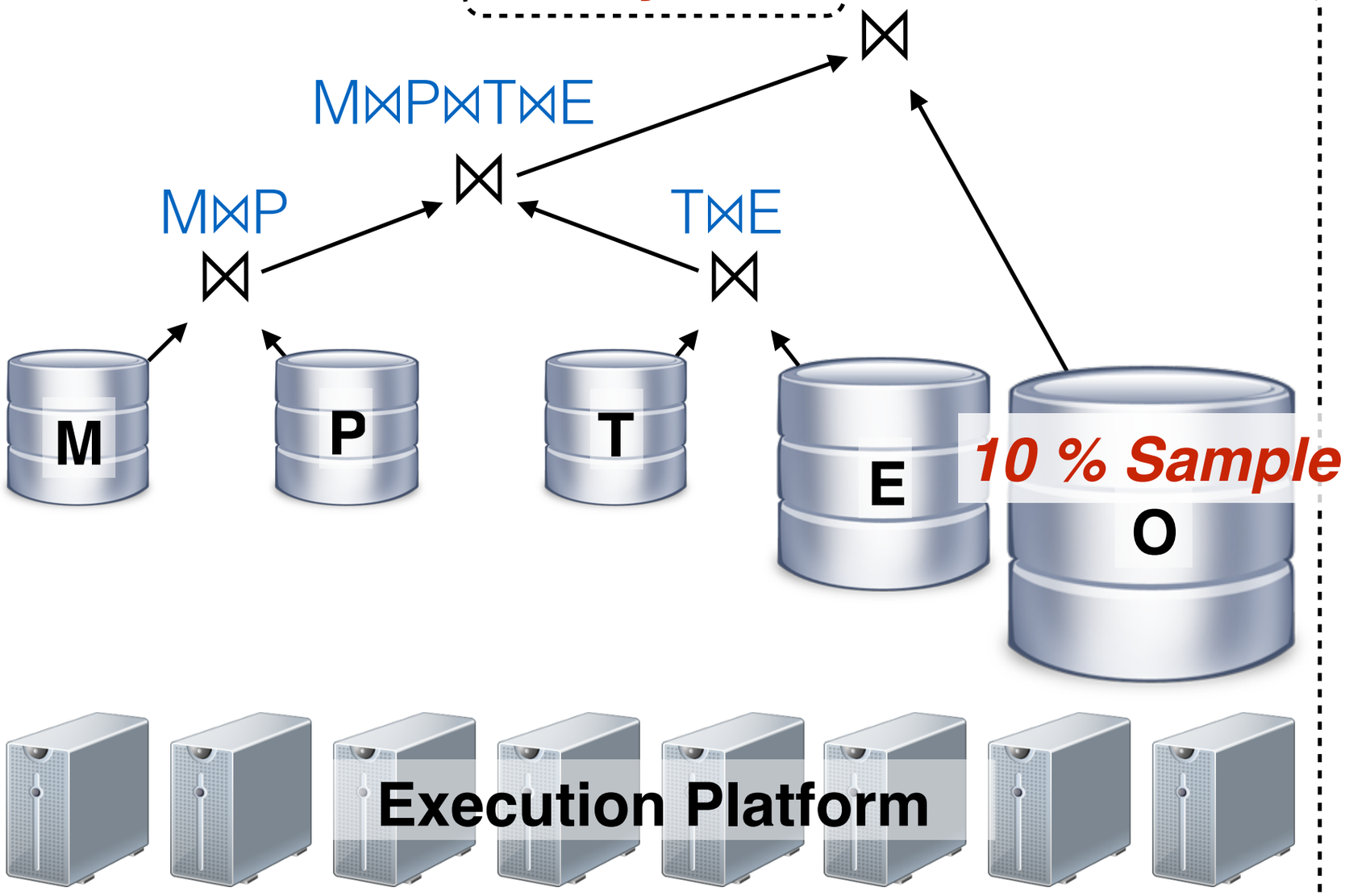
3 hours

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Cost

Time:

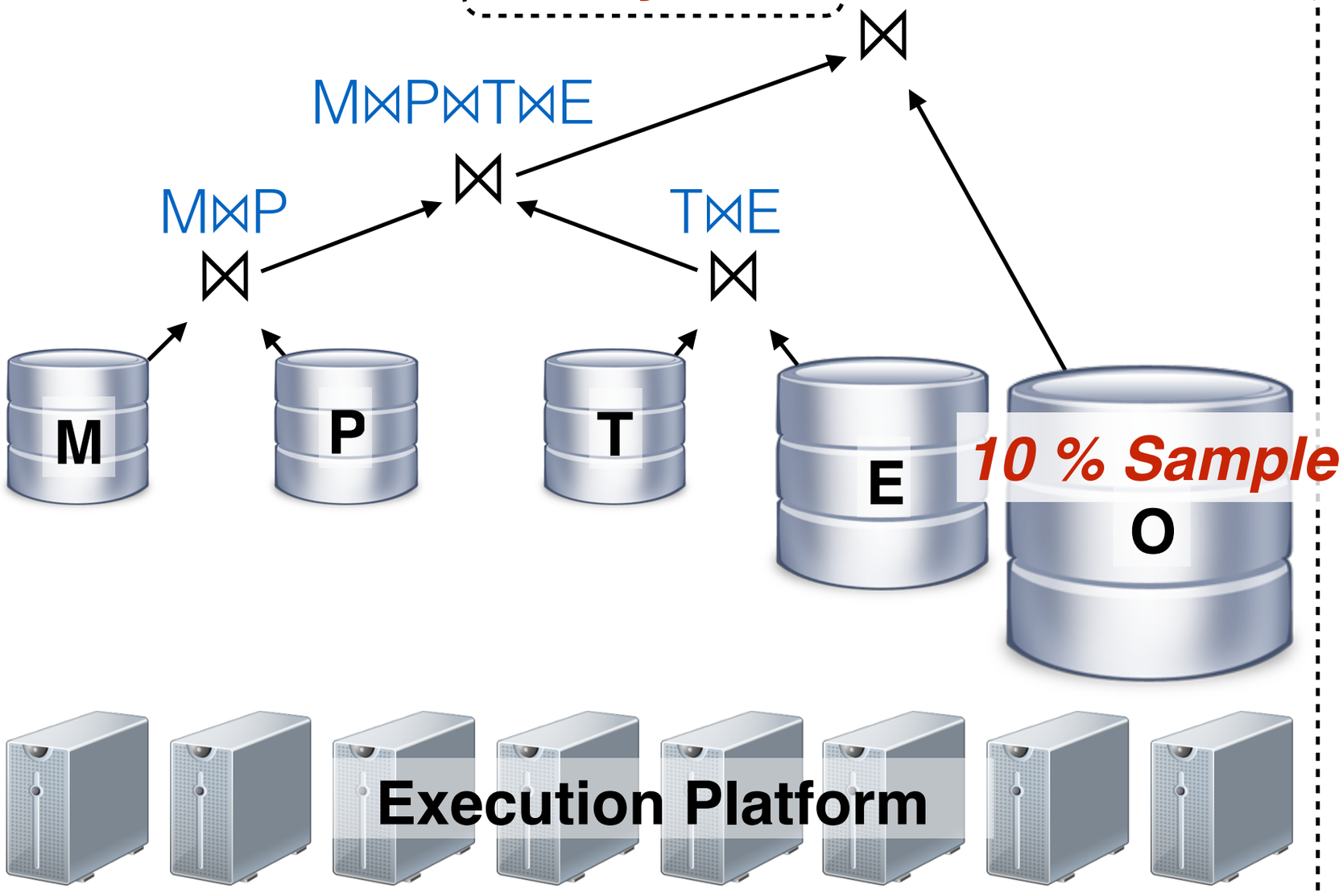
3 hours

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Cost

Time:

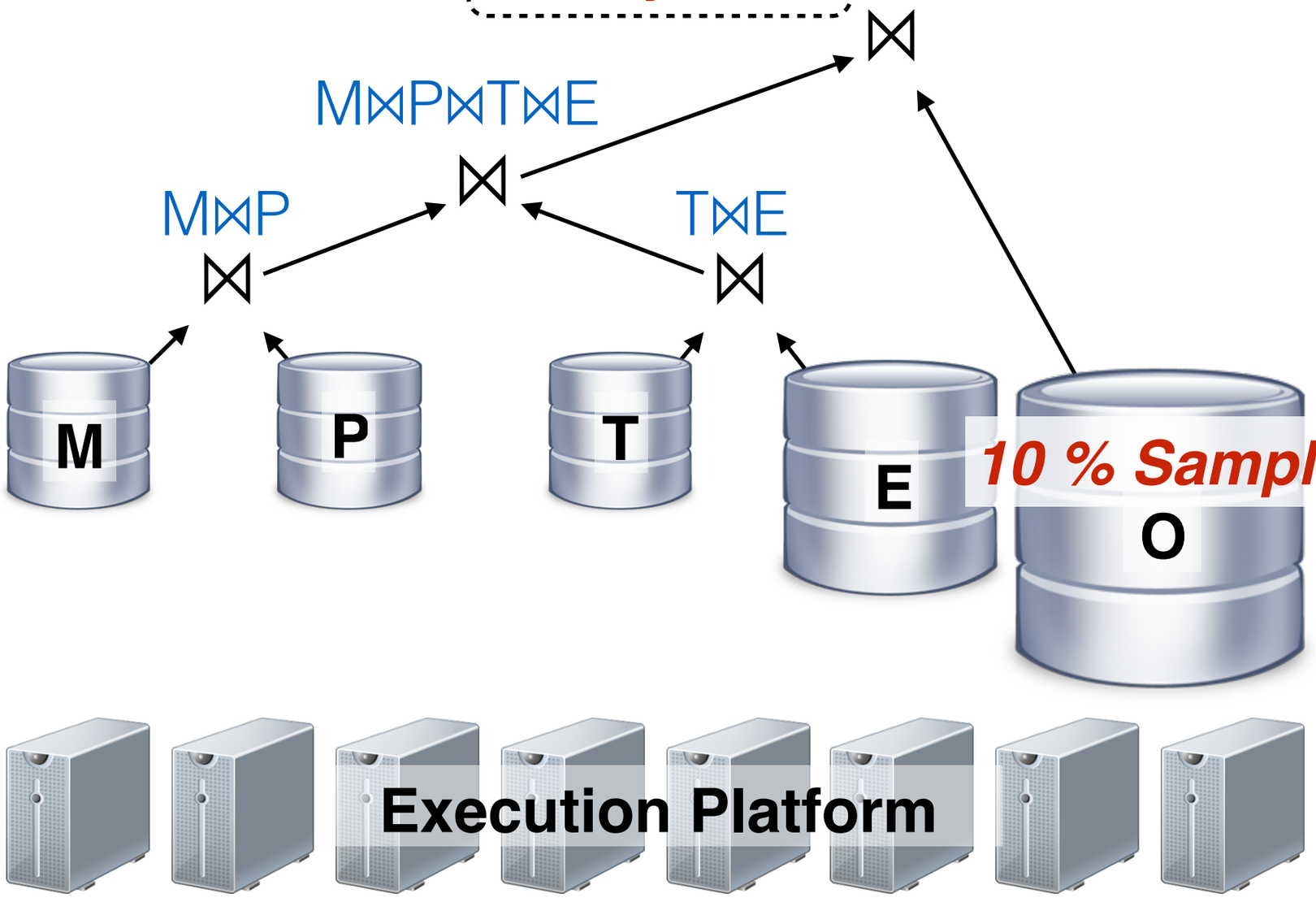
1 hour

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Quality

Time:

1 hour

Completeness:

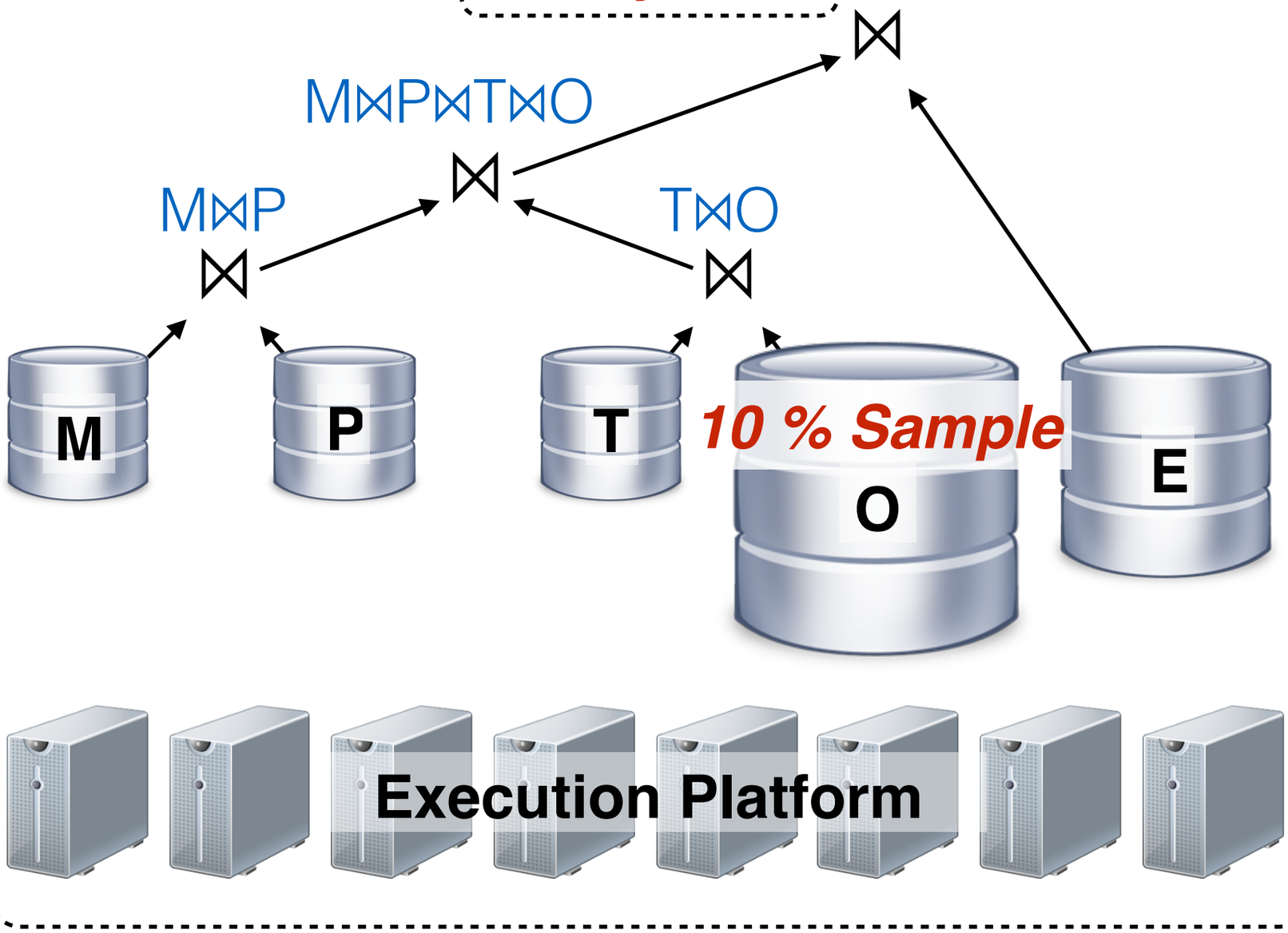
10 %

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Quality

Time:

30 minutes

Completeness:

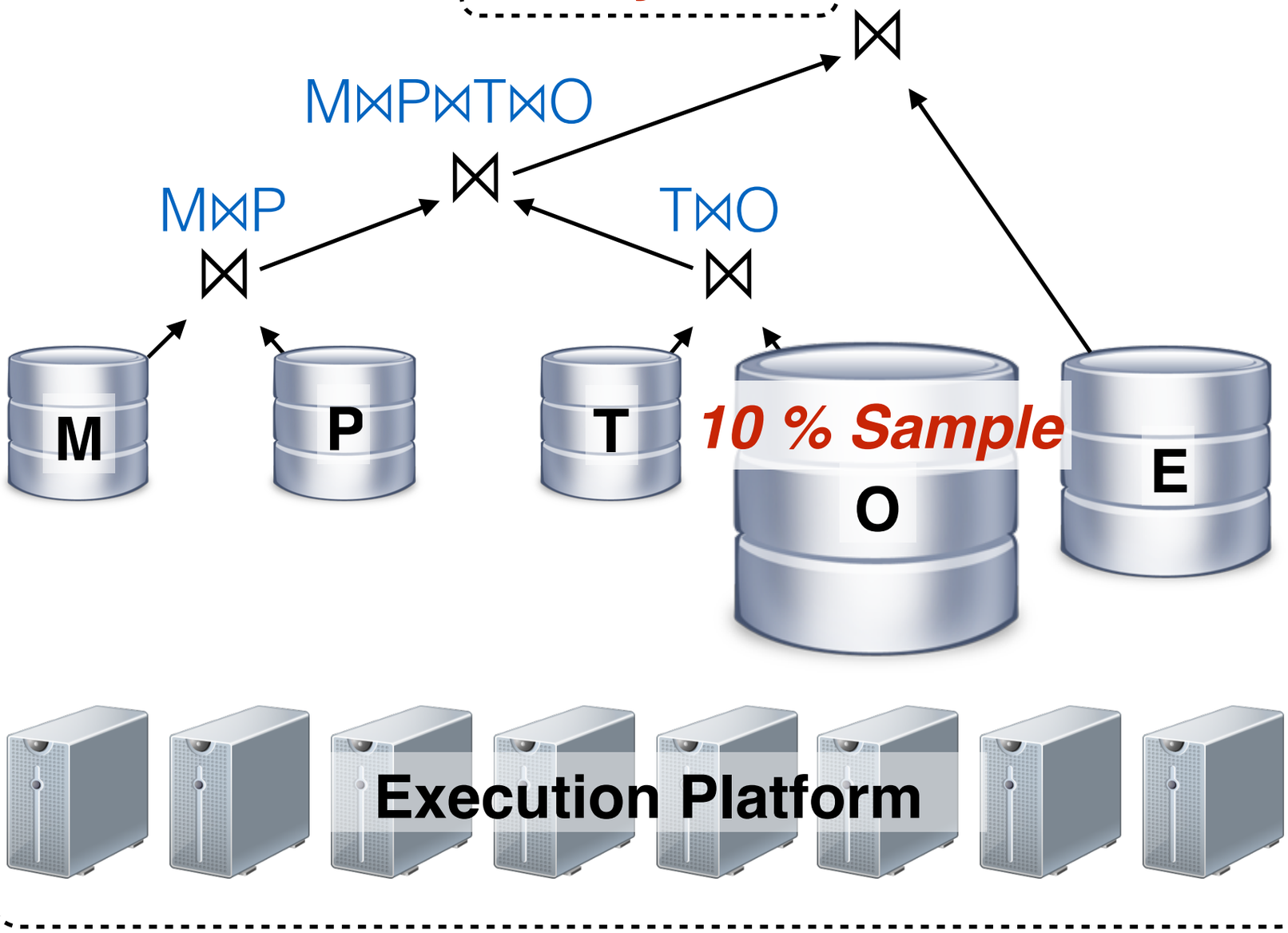
10 %

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Quality

Time:

30 minutes

Completeness:

10 %

Resources:

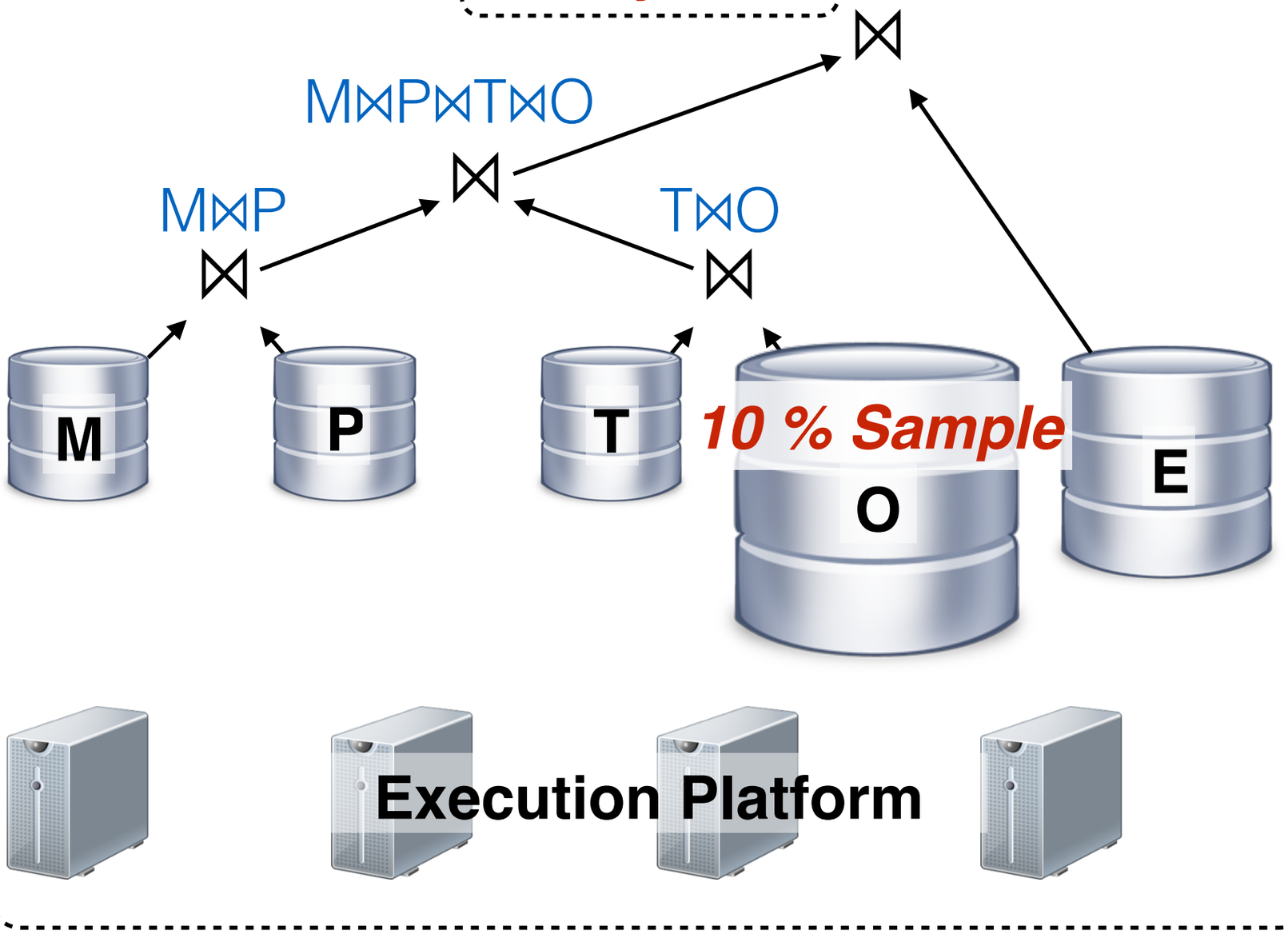
500 CHrs

Example: Google

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

Query Plan



Quality

Time:

34 minutes

Completeness:

10 %

Resources:

283 CHrs

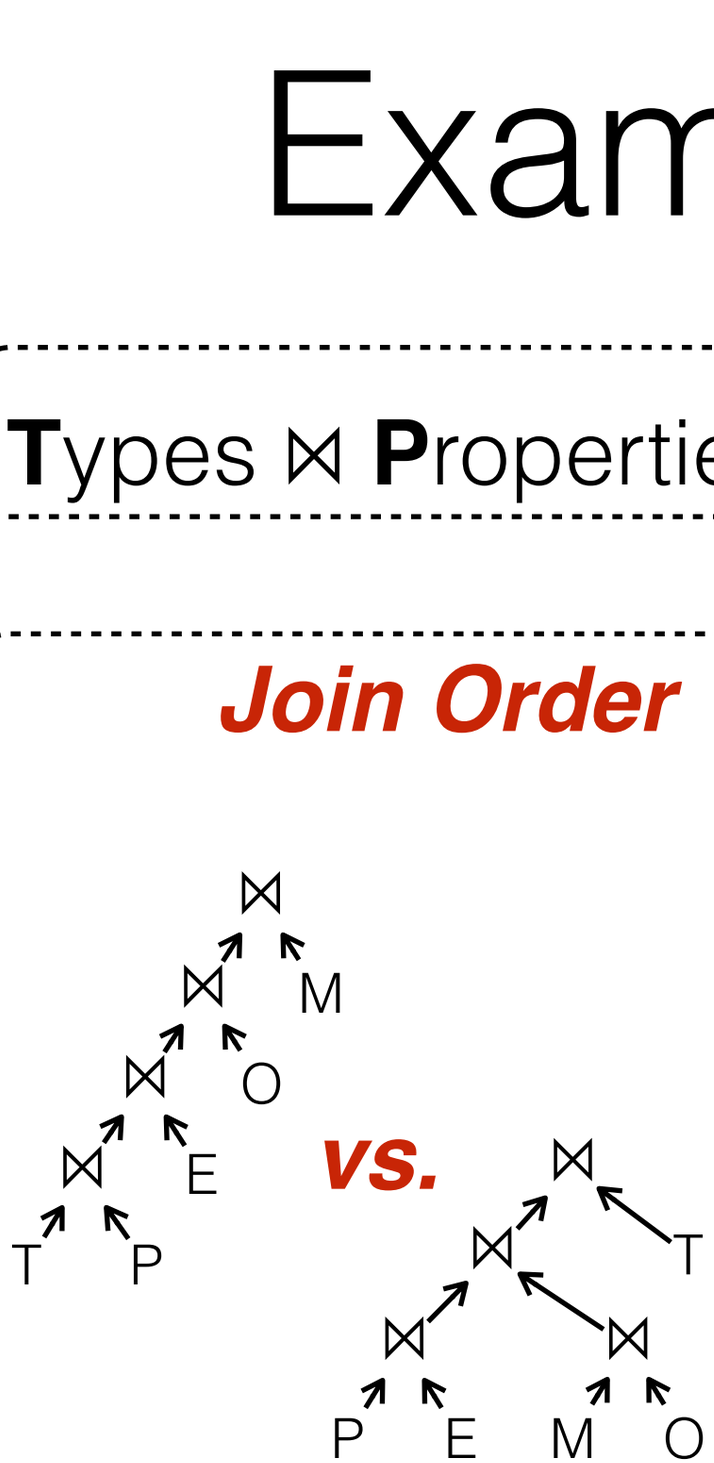
Example Summary

Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

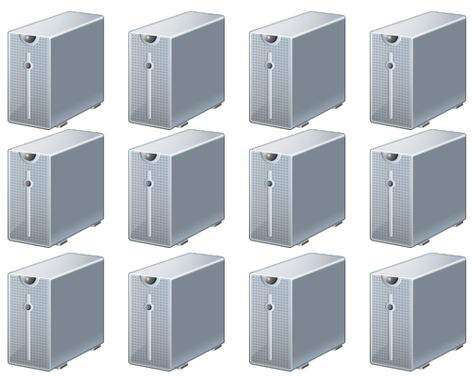
Decisions

Join Order



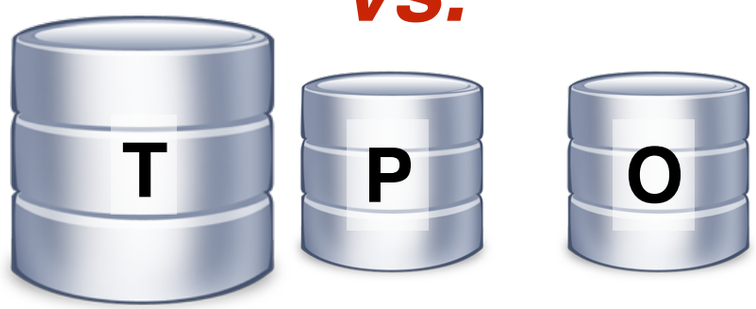
Platform

vs.



Sample Sizes

vs.



Metrics

Execution Time

Resource Consumption

Result Completeness

Example Summary

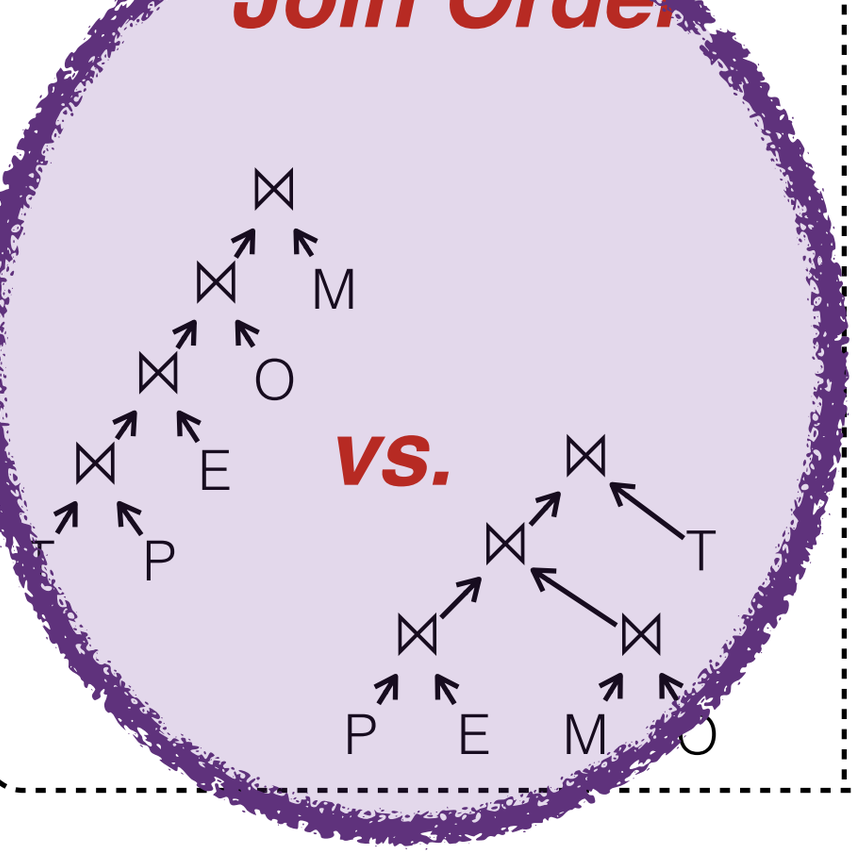
Query

Types ⋈ **P**roperties ⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels …

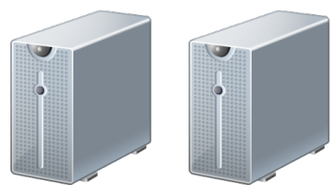
Decisions

Classical Query Optimization

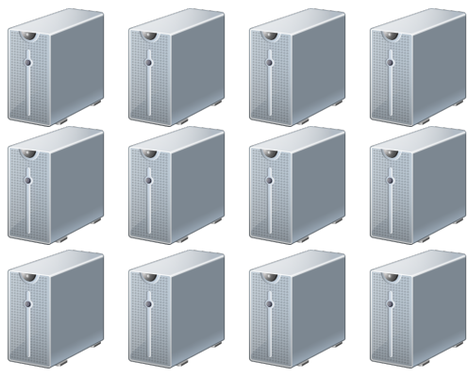
Join Order



Platform

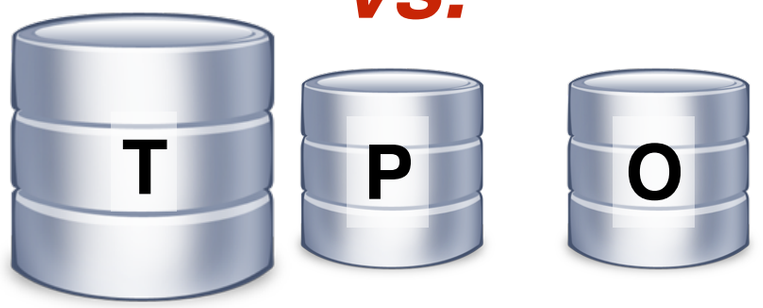


vs.



Sample Sizes

vs.



Metrics

Execution Time

Resource Consumption

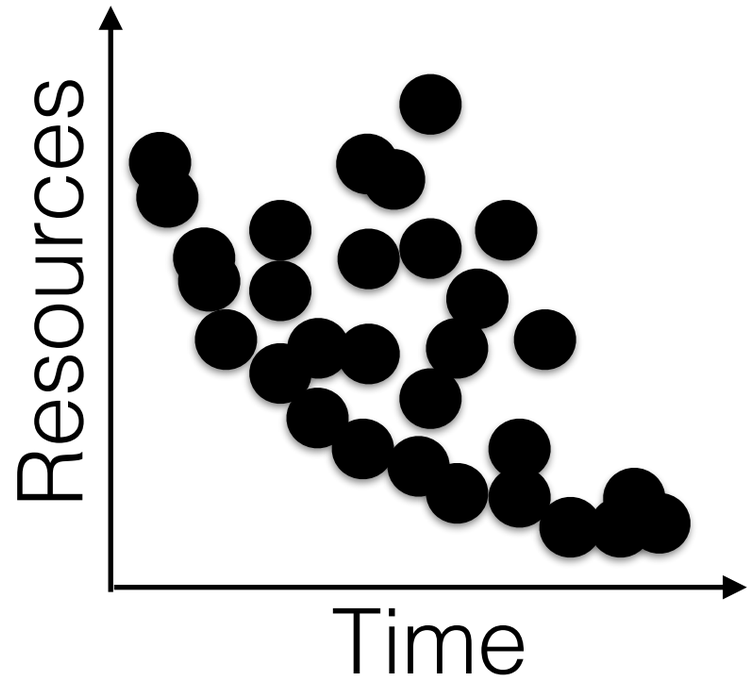
Result Completeness

Example: Google

Types ⋈ **P**roperties
⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels

Query Optimizer

Cost Tradeoffs

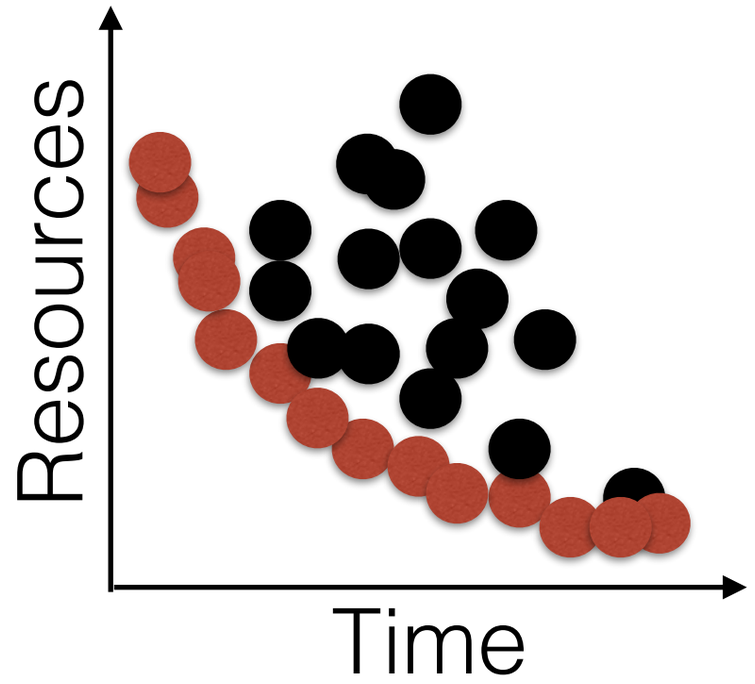


Example: Google

Types ⋈ **P**roperties
⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels

Query Optimizer

Cost Tradeoffs

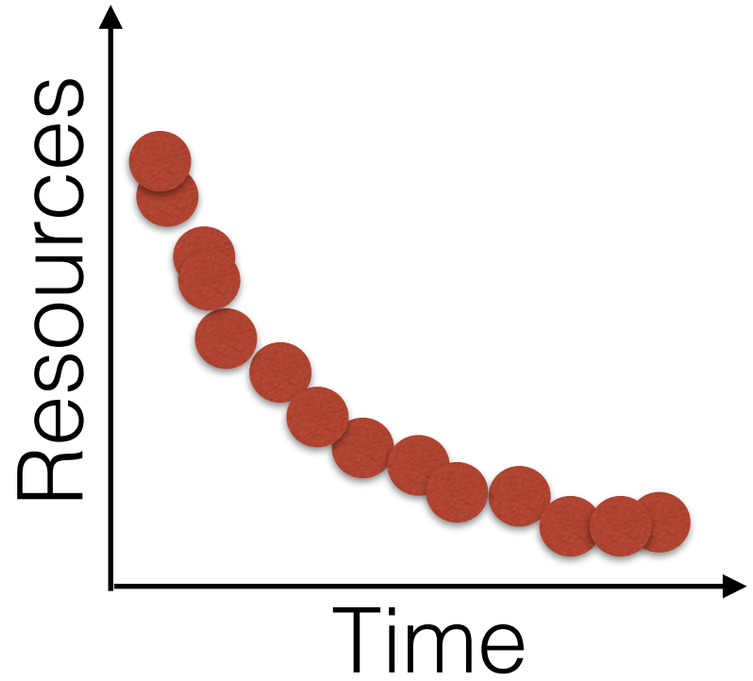


Example: Google

Types ⋈ **P**roperties
⋈ **E**ntities ⋈ **O**pinions ⋈ **M**odels

Query Optimizer

Optimal Tradeoffs



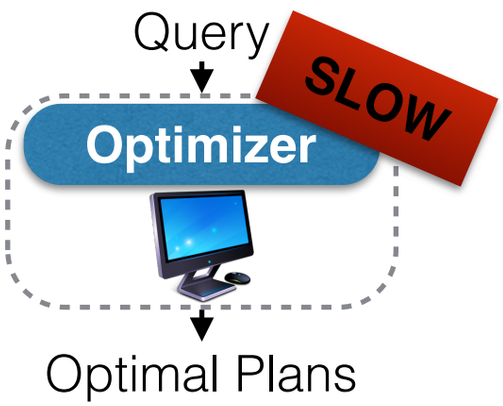
Example: Google

Optimal Tradeoffs

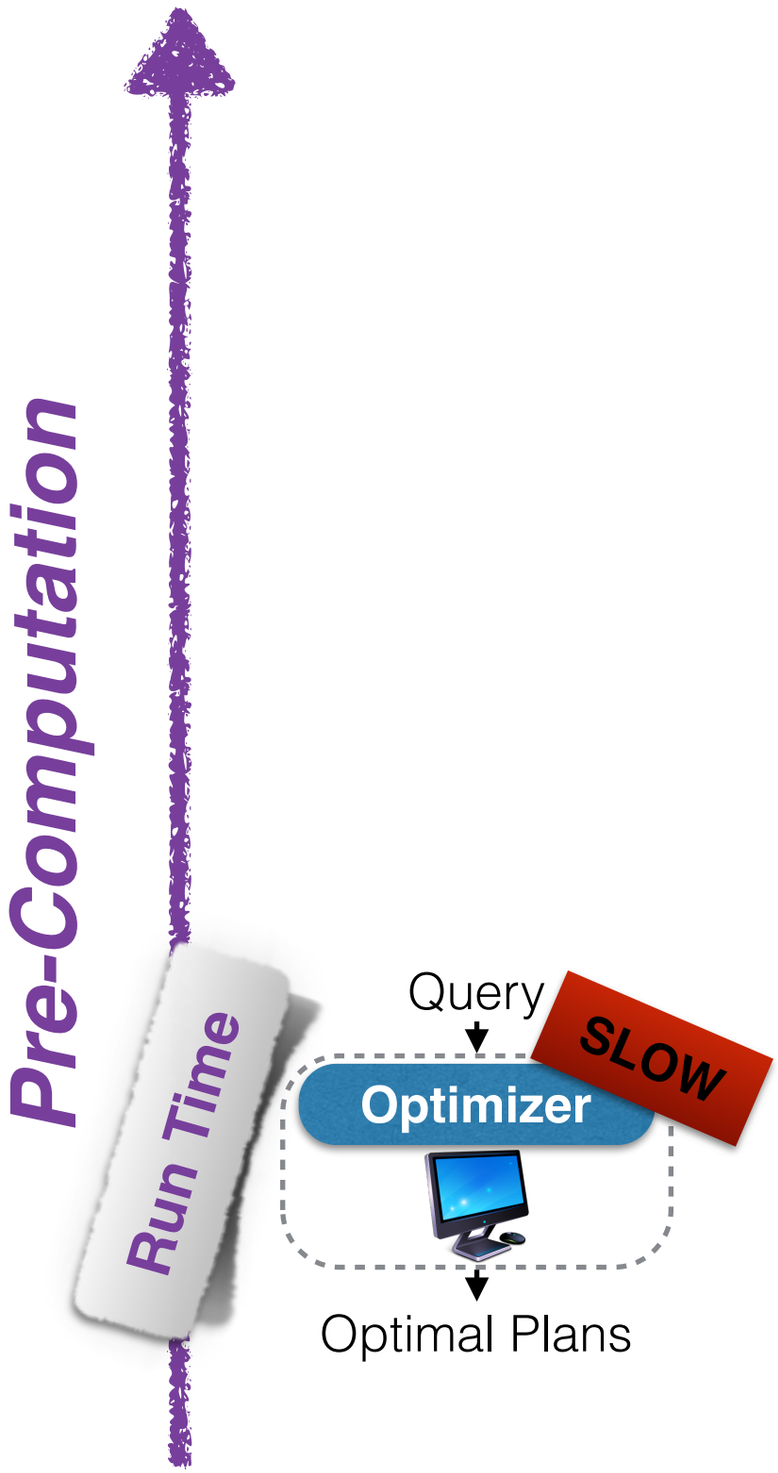
Prior Work:

Takes Hours!!...

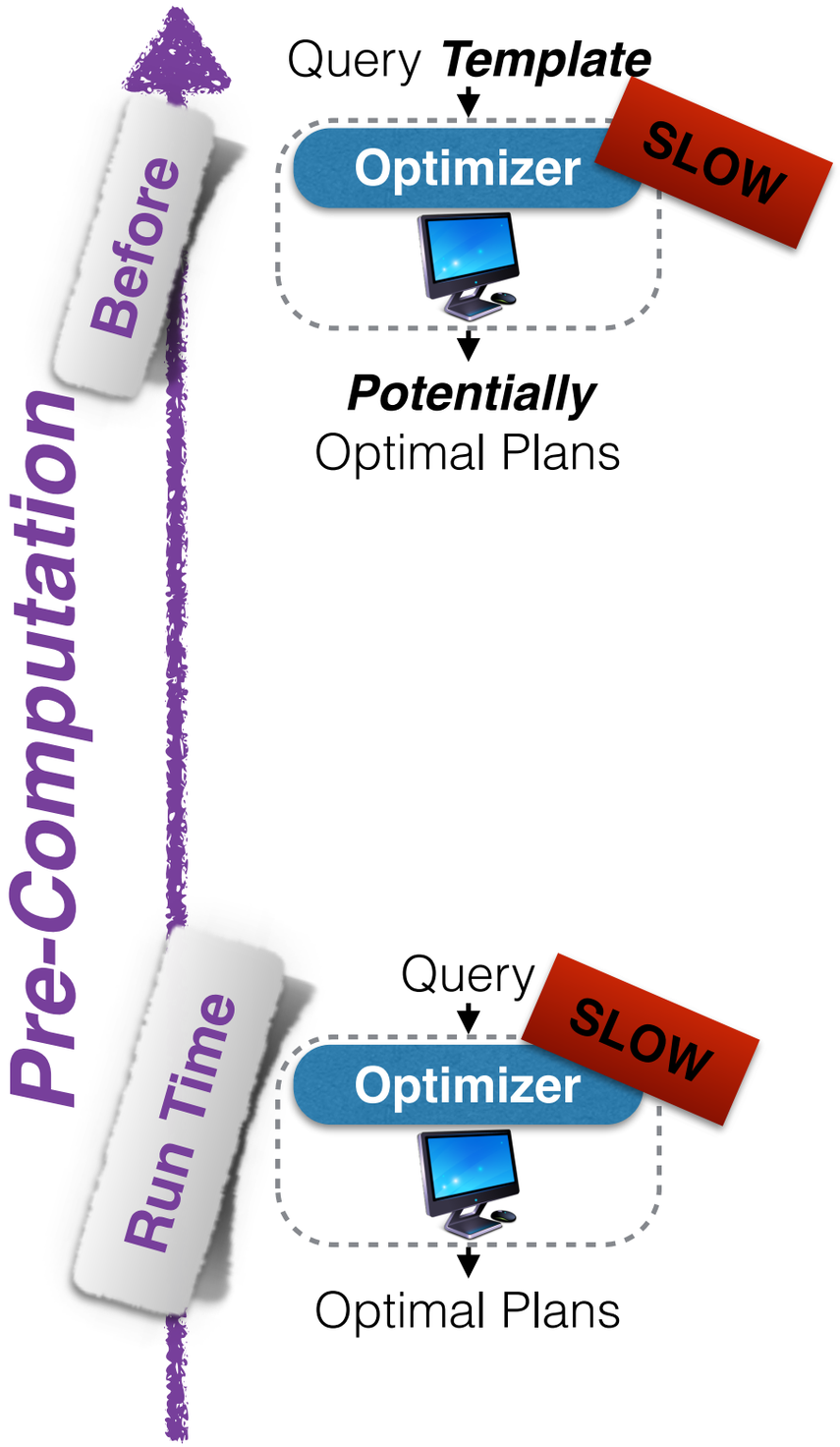
Dissertation Overview



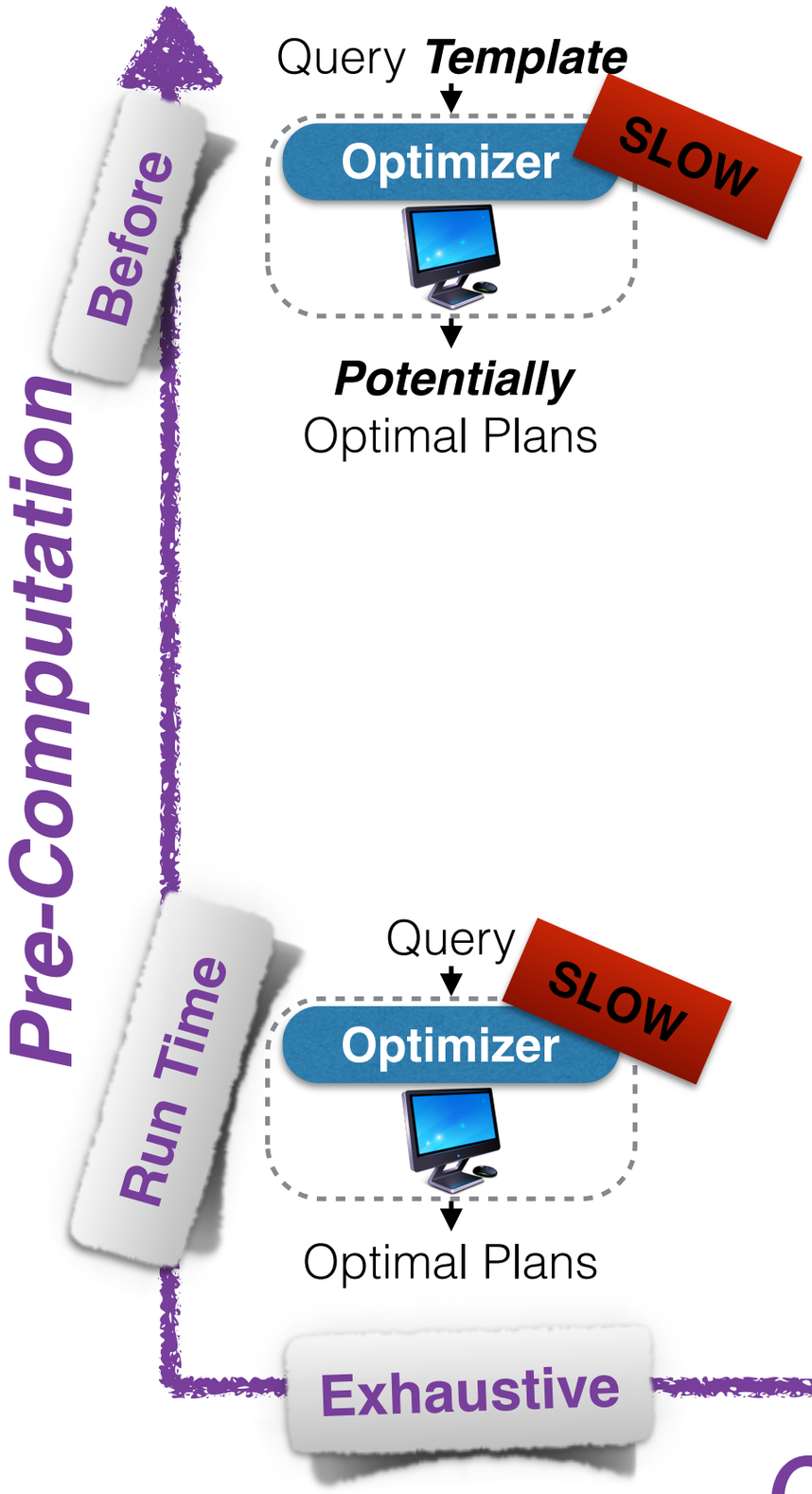
Dissertation Overview



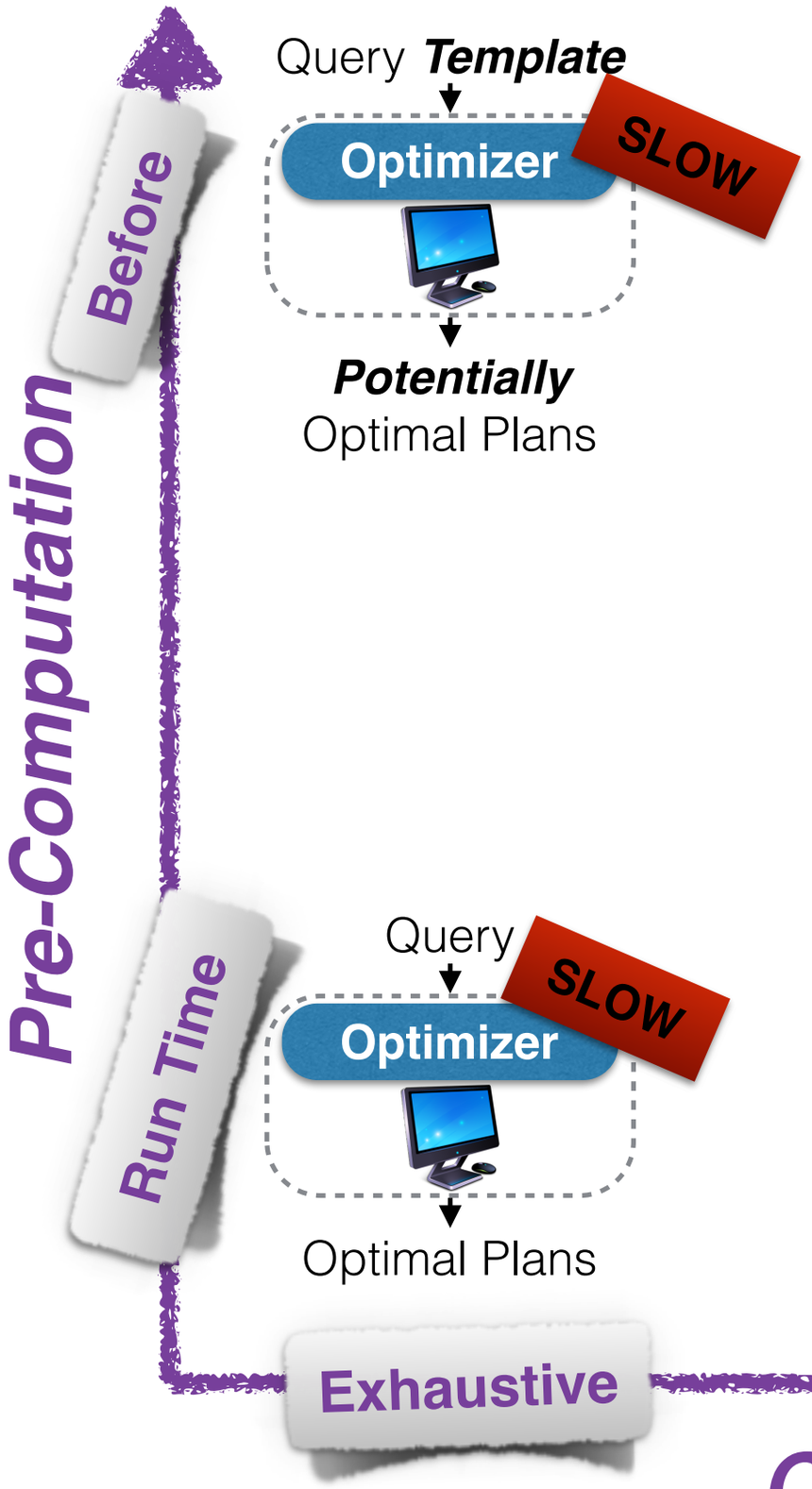
Dissertation Overview



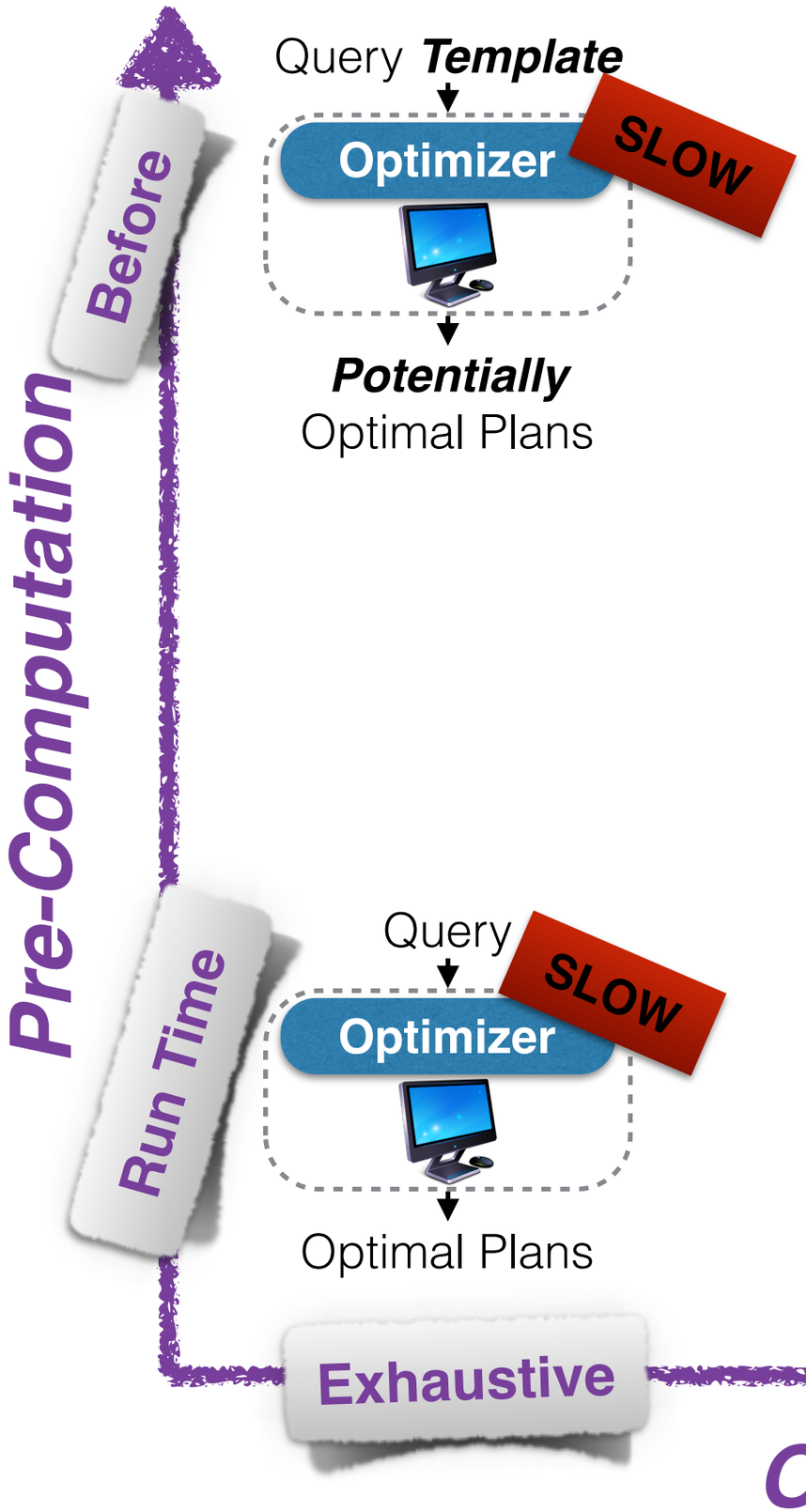
Dissertation Overview



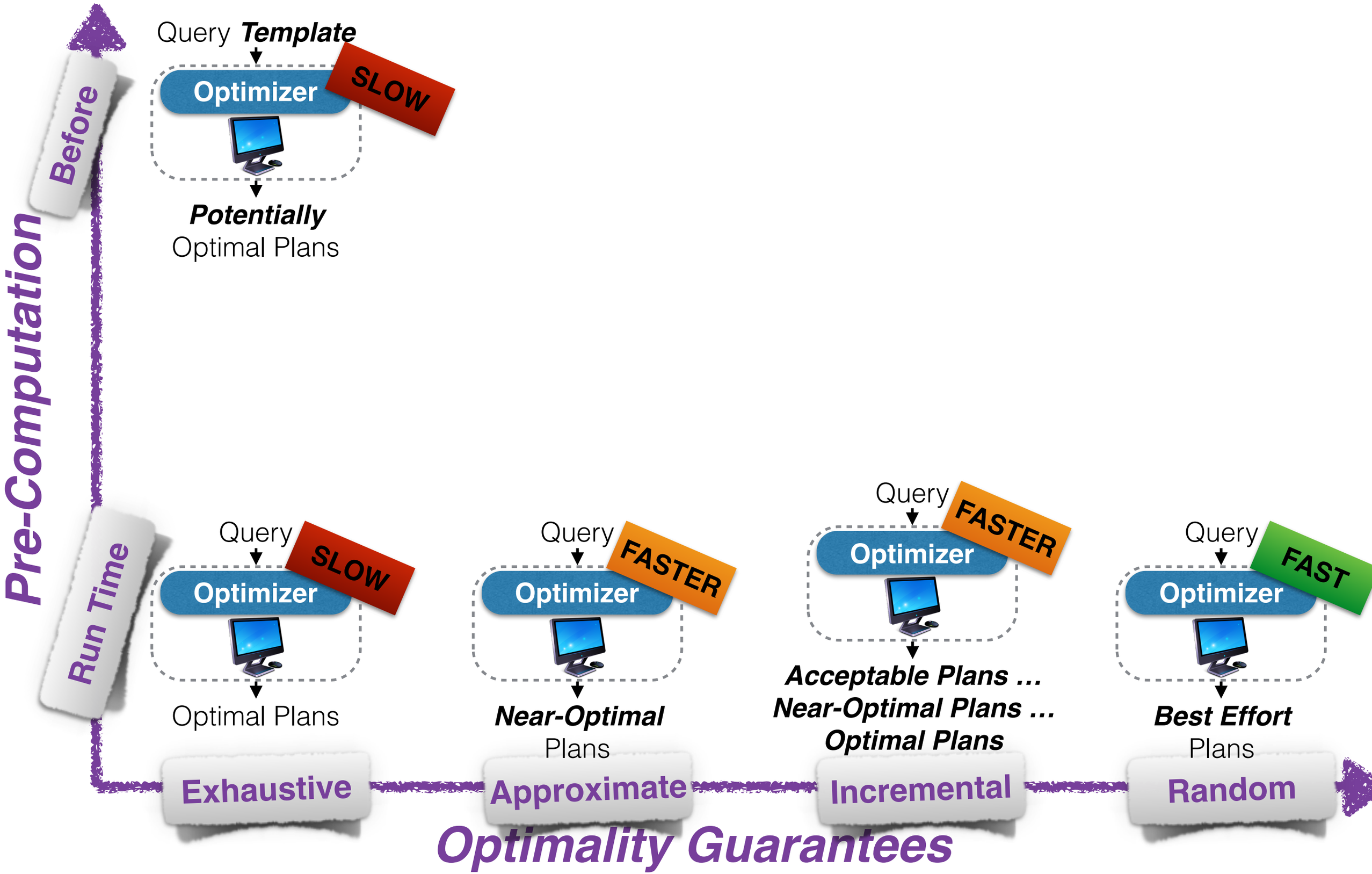
Dissertation Overview



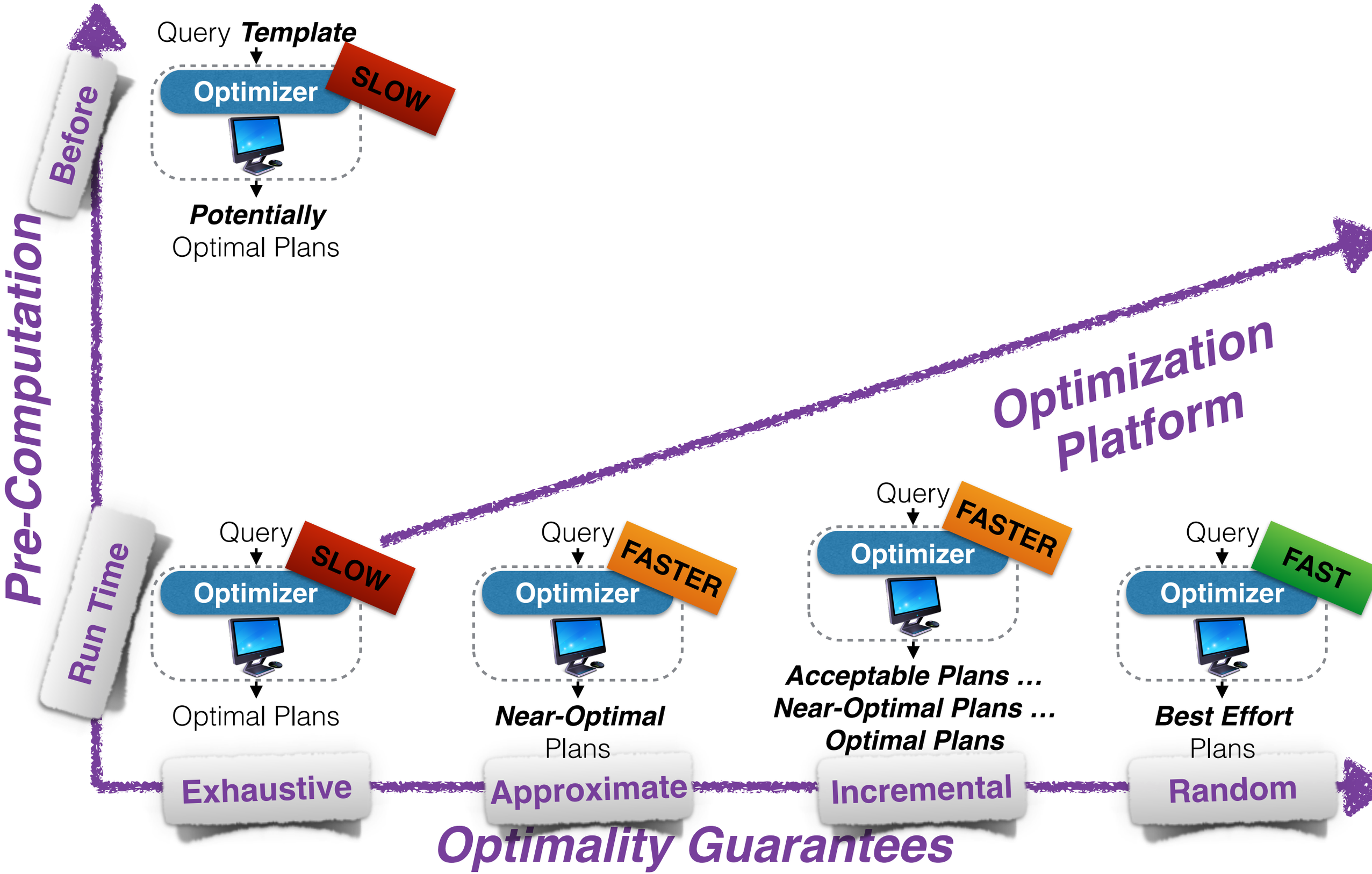
Dissertation Overview



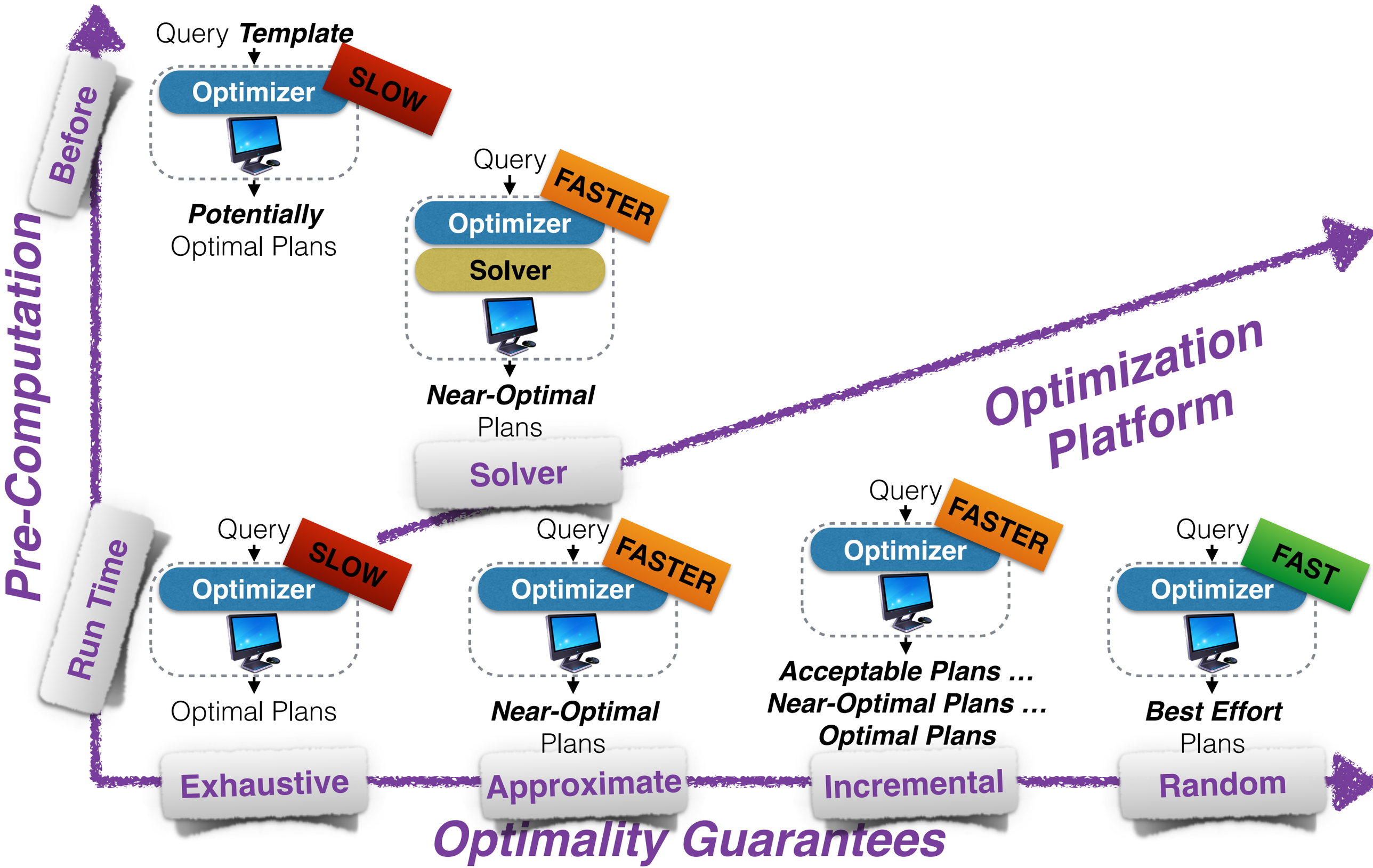
Dissertation Overview



Dissertation Overview



Dissertation Overview



Dissertation Overview

Pre-Computation

Before

Query ***Template***

Optimizer — SLOW

Potentially Optimal Plans

Query — FASTER

Optimizer

Solver

Near-Optimal Plans

Query — FASTER

Optimizer

* Plans

Parallel

Solver

Optimization Platform

Run Time

Query — SLOW

Optimizer

Optimal Plans

Exhaustive

Query — FASTER

Optimizer

Near-Optimal Plans

Approximate

Query — FASTER

Optimizer

Acceptable Plans …
Near-Optimal Plans …
Optimal Plans

Incremental

Query — FAST

Optimizer

Best Effort Plans

Random

Optimality Guarantees

Dissertation Overview

Pre-Computation

Before

Query ***Template***

Optimizer — SLOW

Potentially
Optimal Plans

Query — FASTER

Optimizer

Solver

Near-Optimal
Plans

Solver

Query — FASTER

Optimizer

* Plans

Parallel

Query

Optimizer

FAST

Best Effort
Plans

Quantum

Optimization Platform

Run Time

Query — SLOW

Optimizer

Optimal Plans

Exhaustive

Query — FASTER

Optimizer

Near-Optimal
Plans

Approximate

Query — FASTER

Optimizer

Acceptable Plans …
Near-Optimal Plans …
Optimal Plans

Incremental

Query — FAST

Optimizer

Best Effort
Plans

Random

Optimality Guarantees

Dissertation Overview

Pre-Computation

Before

Query *Template*

Optimizer

VLDB'15

VLDBJ'16

SLOW

Potentially Optimal Plans

Best of VLDB

SIGMOD Research Highlight

CACM Research Highlight

Query

Optimizer

Solver

ArXiV

FASTER

Near-Optimal Plans

Solver

Query

Optimizer

VLDB'16

FASTER

* Plans

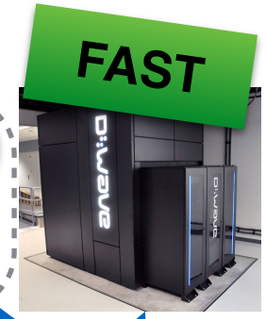
Parallel

Query

Optimizer

VLDB'16

FAST



Best Effort Plans

Quantum

Optimization Platform

Run Time

Query

Optimizer

SLOW

Optimal Plans

Exhaustive

Query

Optimizer

SIGMOD'14

FASTER

Near-Optimal Plans

Approximate

Query

Optimizer

SIGMOD'15

FASTER

Acceptable Plans …
Near-Optimal Plans …
Optimal Plans

Incremental

Query

Optimizer

SIGMOD'16

FAST

Best Effort Plans

Random

Optimality Guarantees

Dissertation Overview

Pre-Computation

Before

Query *Template*

Optimizer

SLOW

VLDB'15

VLDBJ'16

Potentially Optimal Plans

Best of VLDB

SIGMOD Research Highlight

CACM Research Highlight

ArXiV

Query

FASTER

Optimizer

Solver

Near-Optimal Plans

Solver

VLDB'16

Query

FASTER

Optimizer

* Plans

Parallel

VLDB'16

Query

Optimizer

FAST

Best Effort Plans

Quantum

Optimization Platform

Run Time

Query

SLOW

Optimizer

Optimal Plans

Exhaustive

SIGMOD'14

Query

FASTER

Optimizer

Near-Optimal Plans

Approximate

SIGMOD'15

Query

FASTER

Optimizer

Acceptable Plans …
Near-Optimal Plans …
Optimal Plans

Incremental

SIGMOD'16

Query

FAST

Optimizer

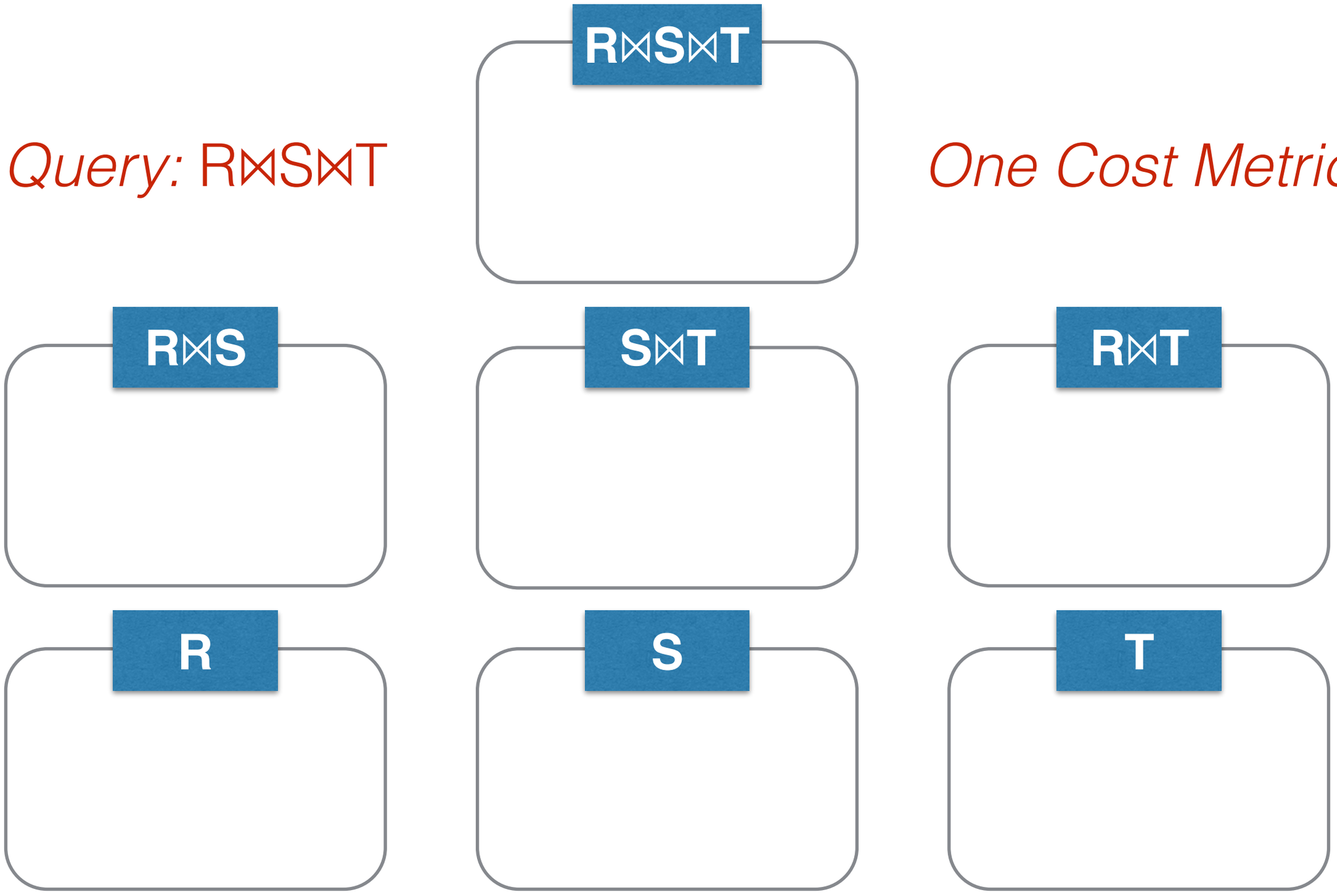
Best Effort Plans

Random

Optimality Guarantees

Classical Query Optimization

Query: R⋈S⋈T

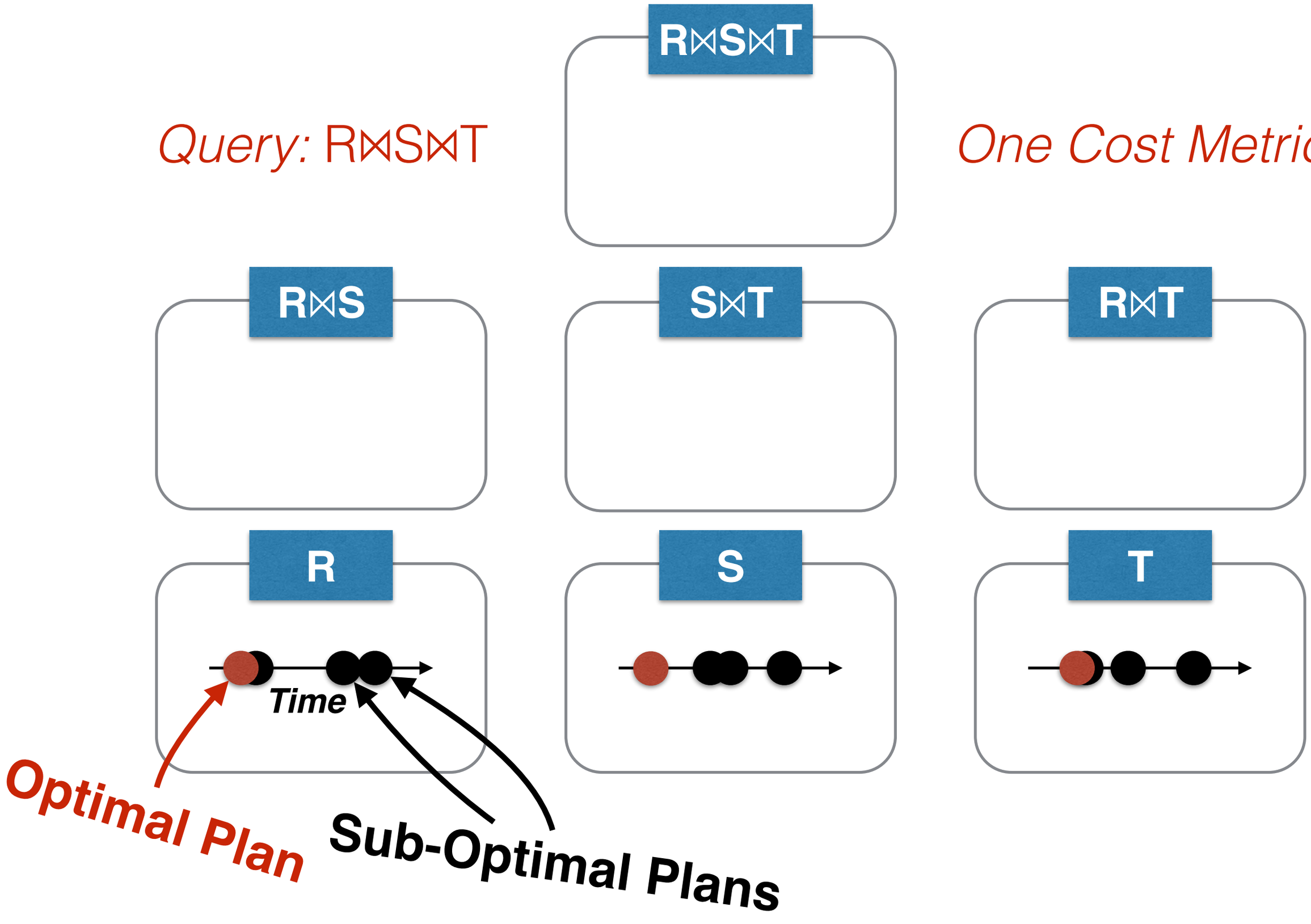


One Cost Metric

Classical Query Optimization

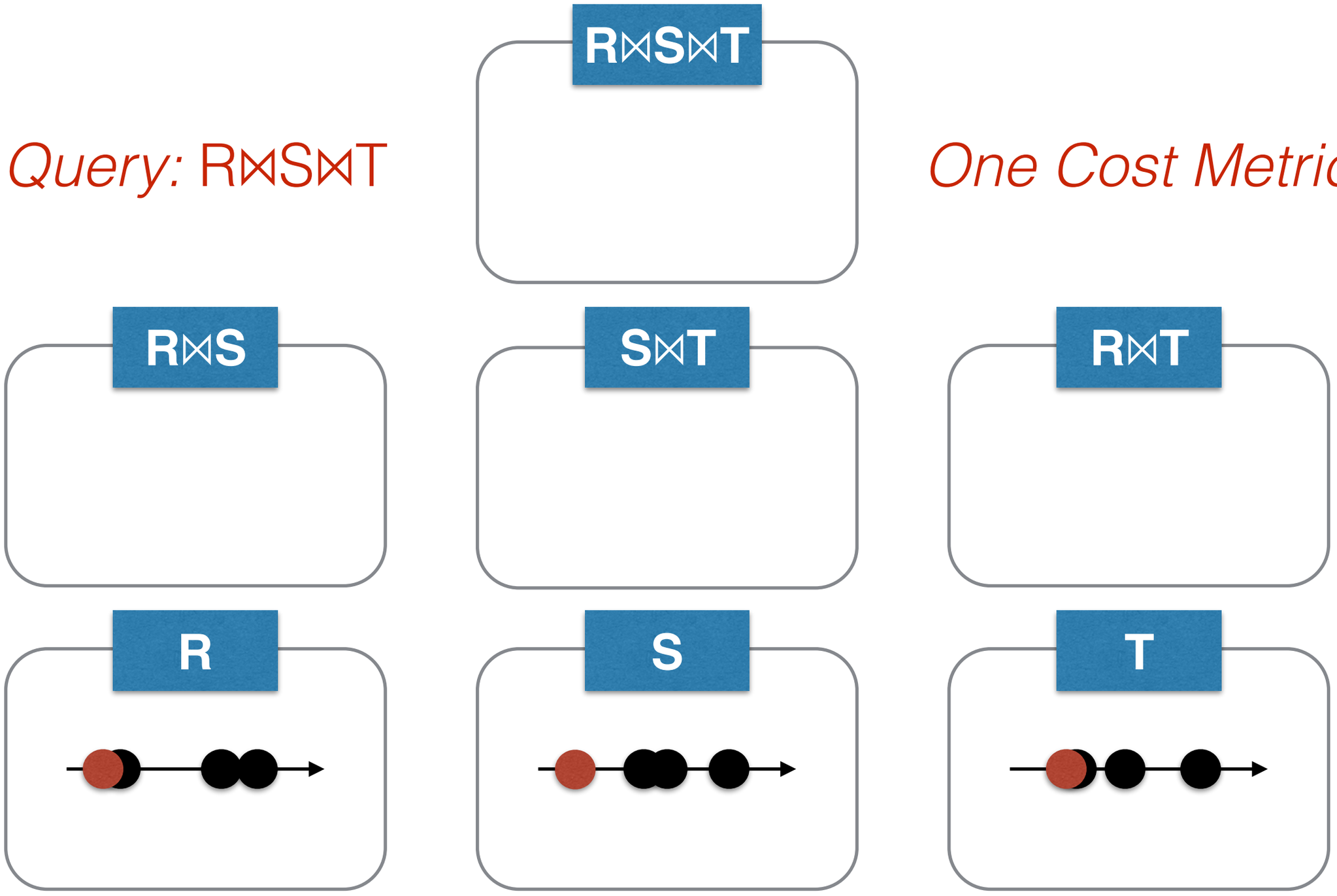
Query: R⋈S⋈T

One Cost Metric



Classical Query Optimization

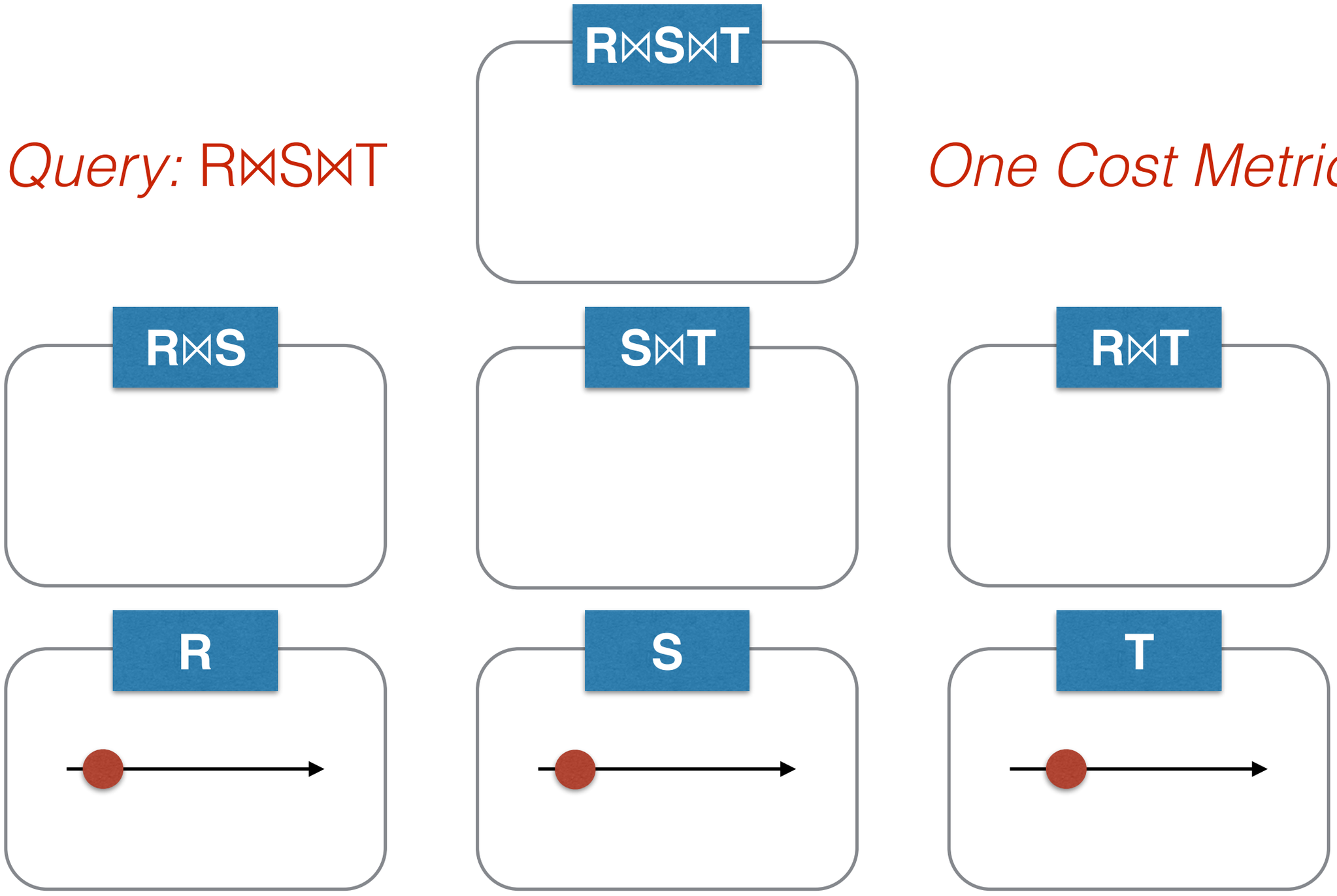
Query: R⋈S⋈T



One Cost Metric

Classical Query Optimization

Query: R⋈S⋈T

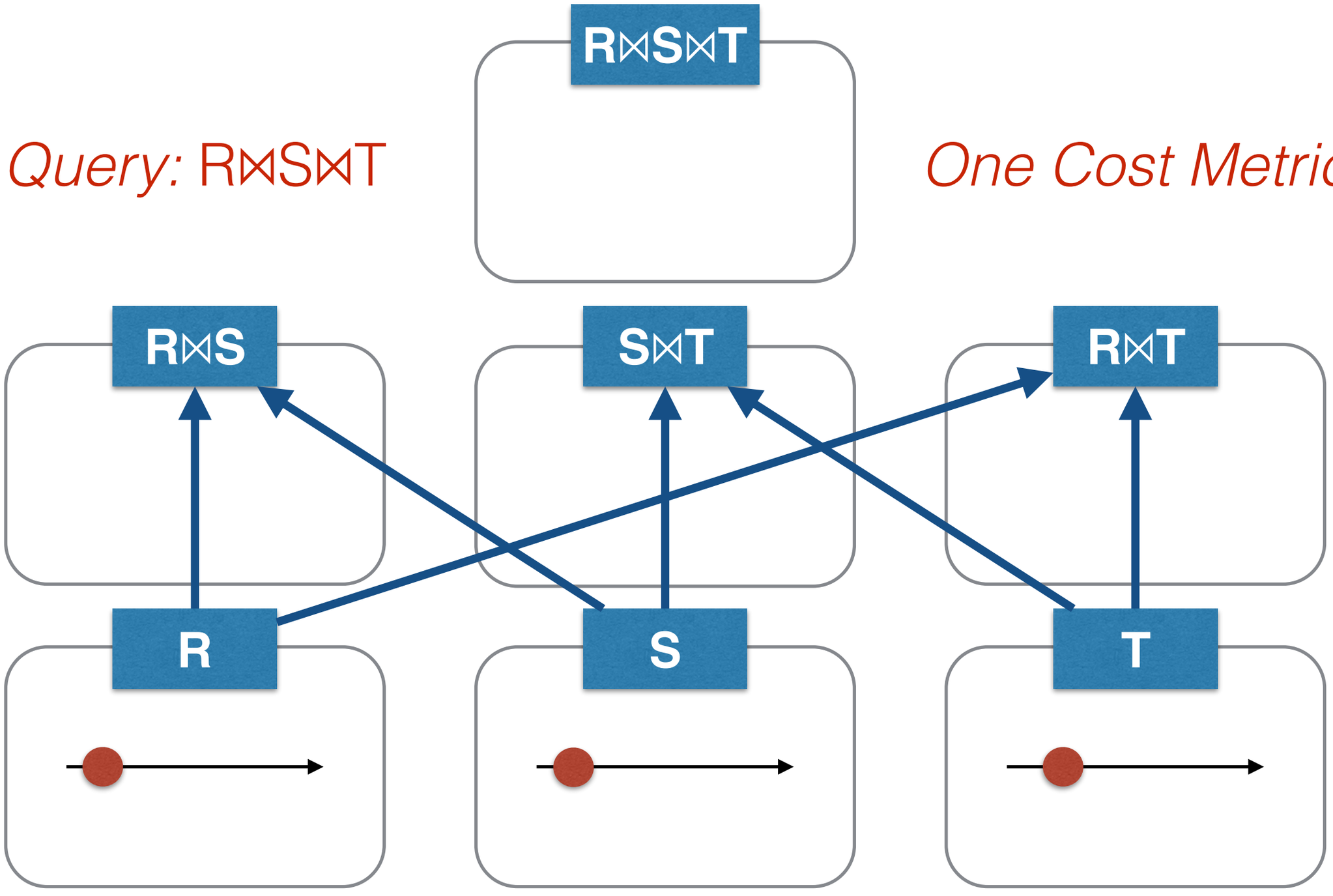


One Cost Metric

Classical Query Optimization

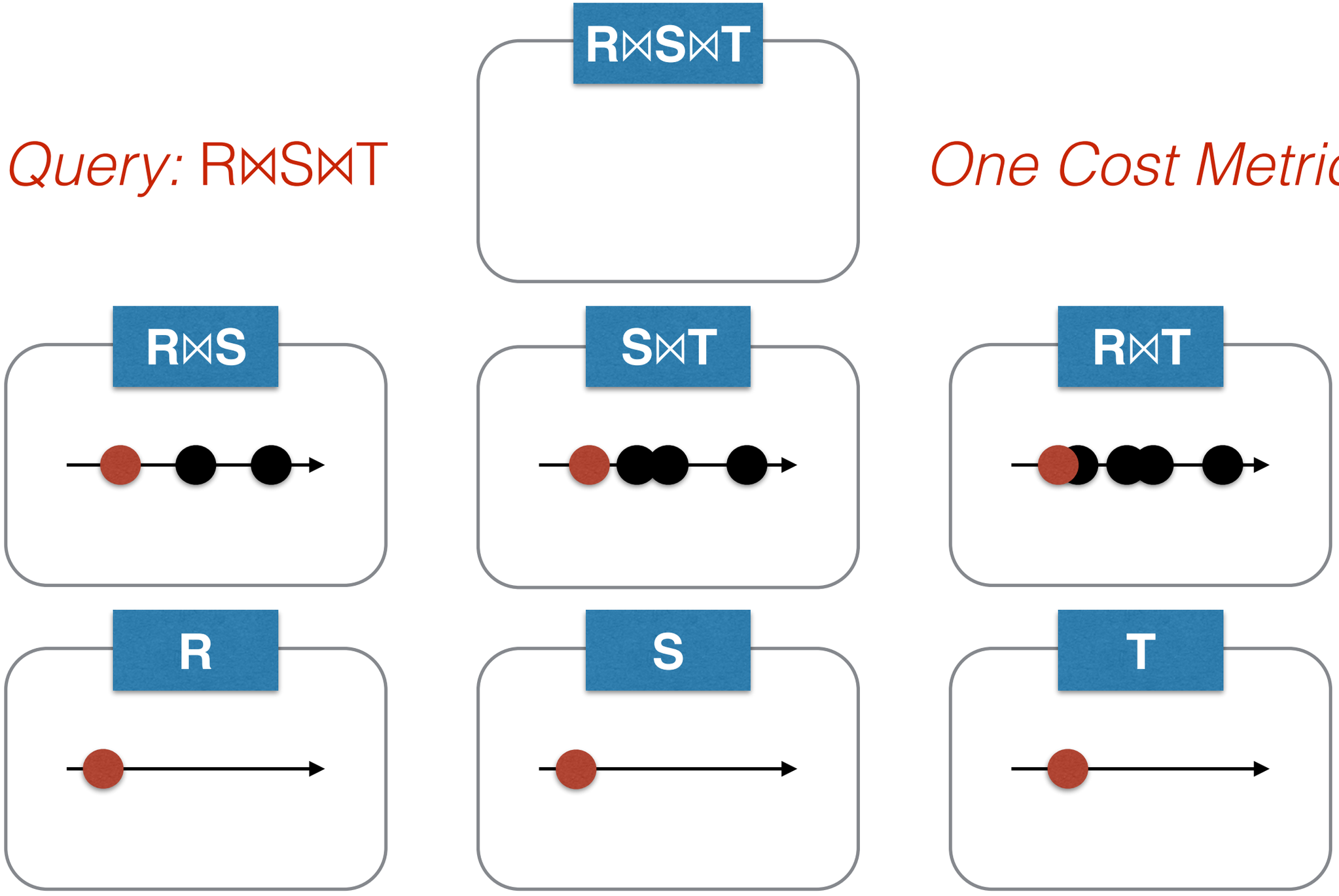
Query: R⋈S⋈T

One Cost Metric



Classical Query Optimization

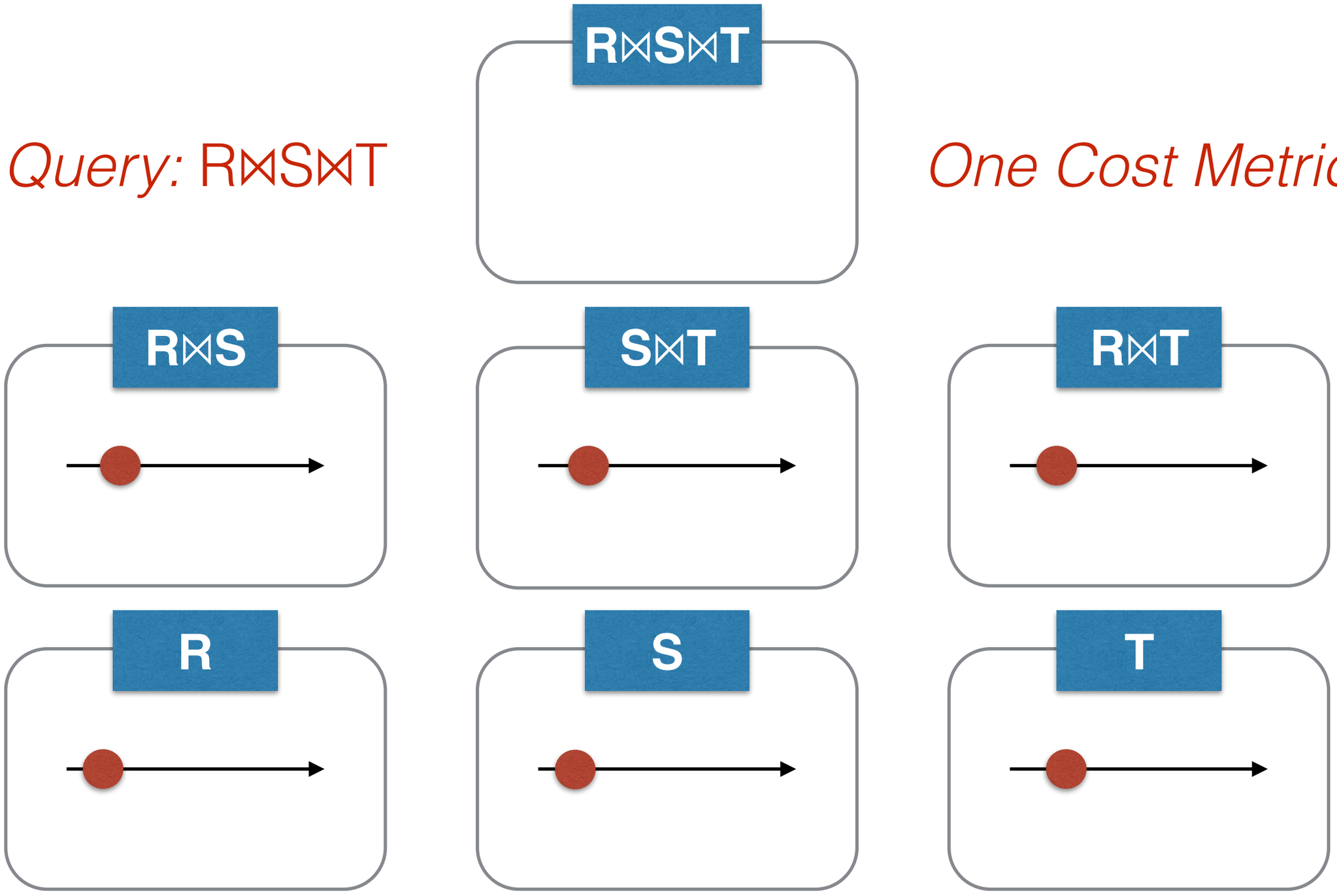
Query: R⋈S⋈T



One Cost Metric

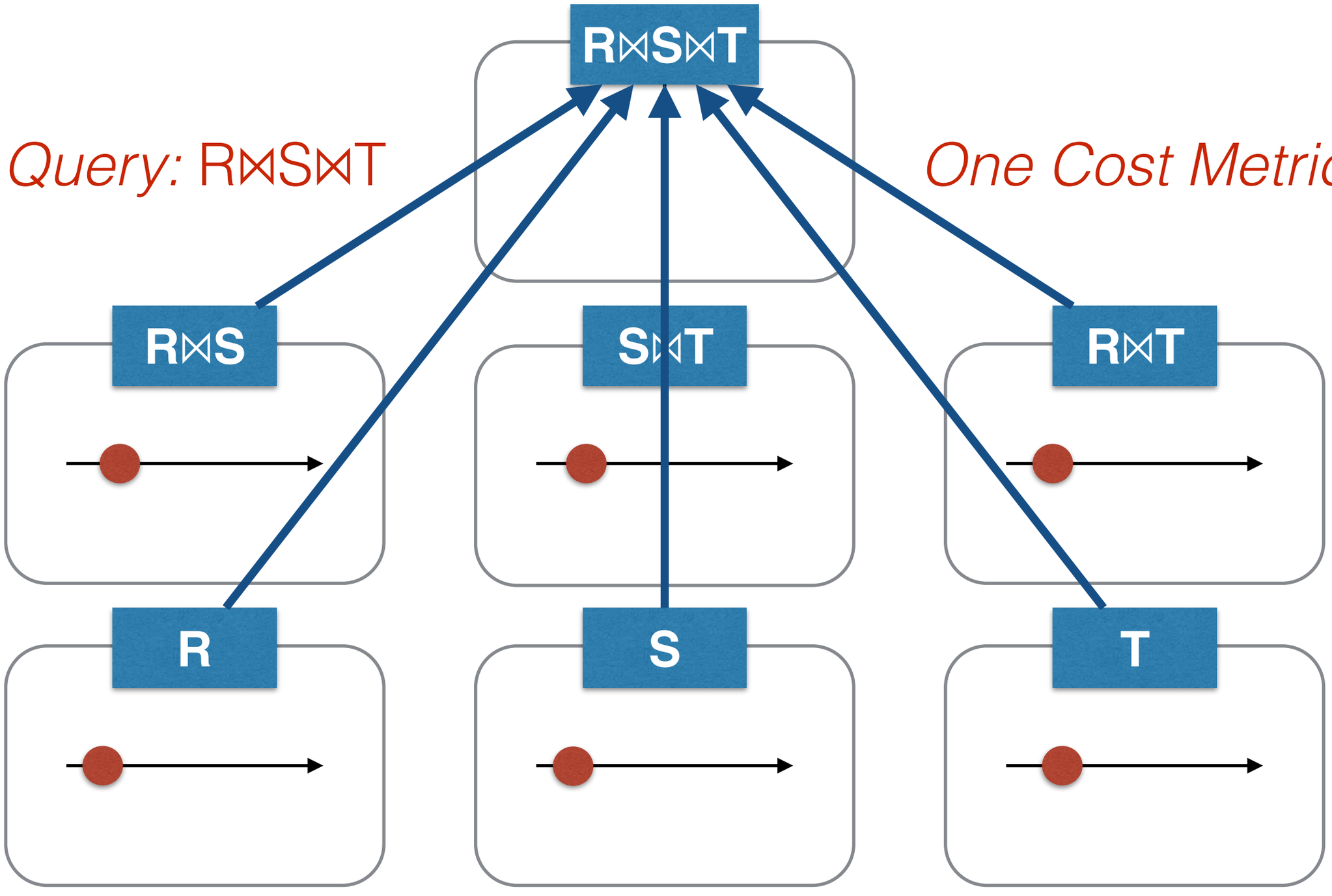
Classical Query Optimization

Query: R⋈S⋈T



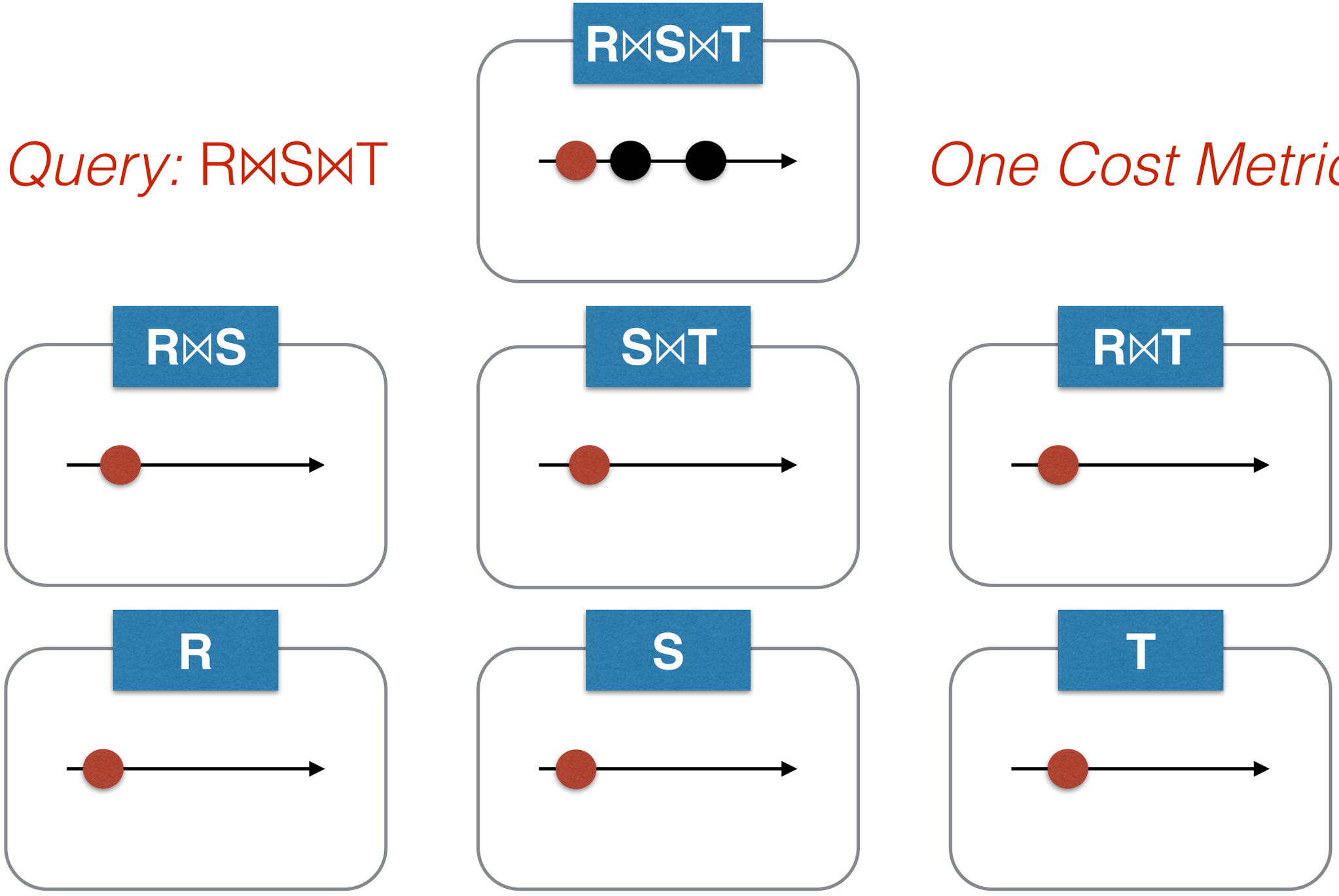
One Cost Metric

Classical Query Optimization



Classical Query Optimization

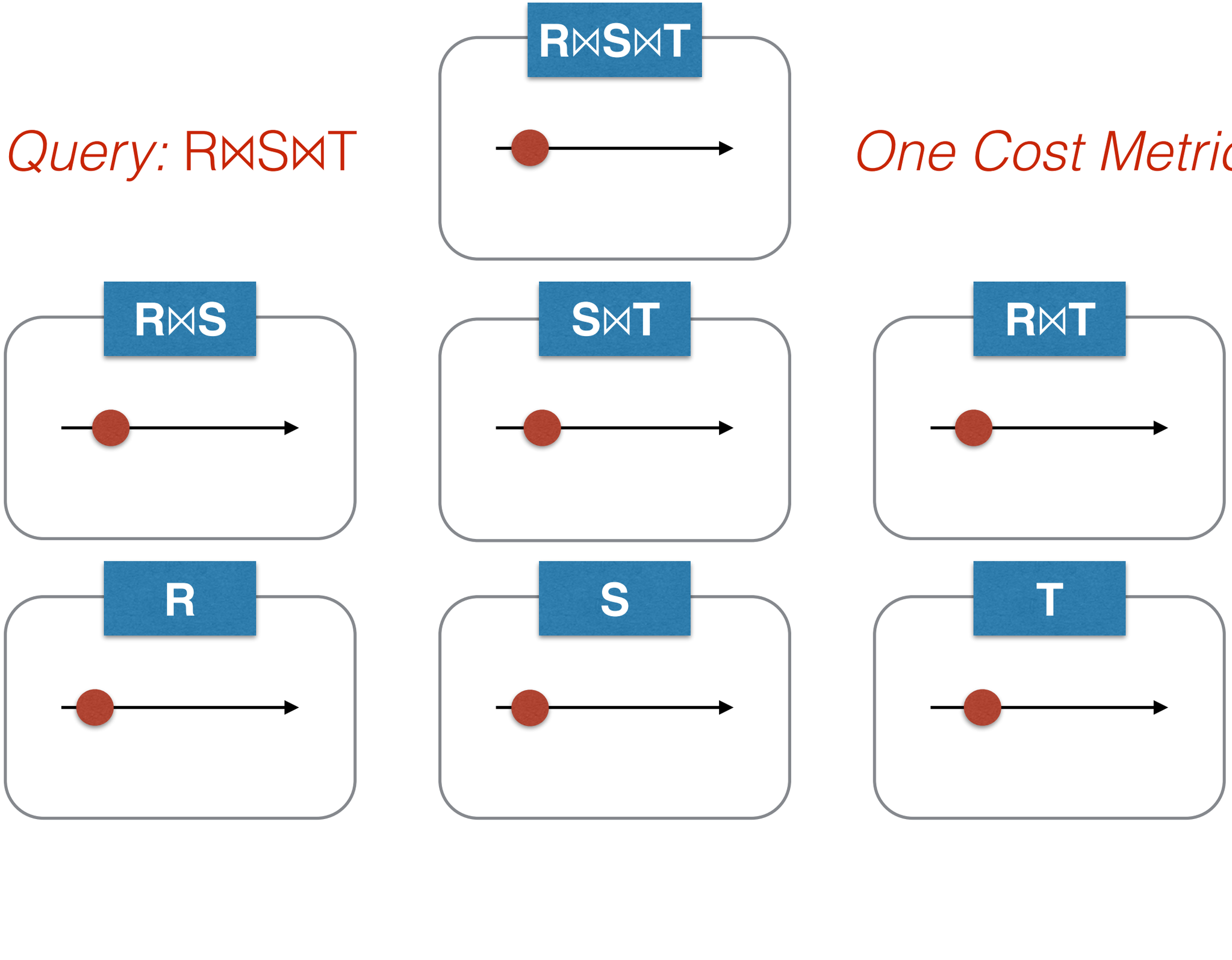
Query: R⋈S⋈T



One Cost Metric

Classical Query Optimization

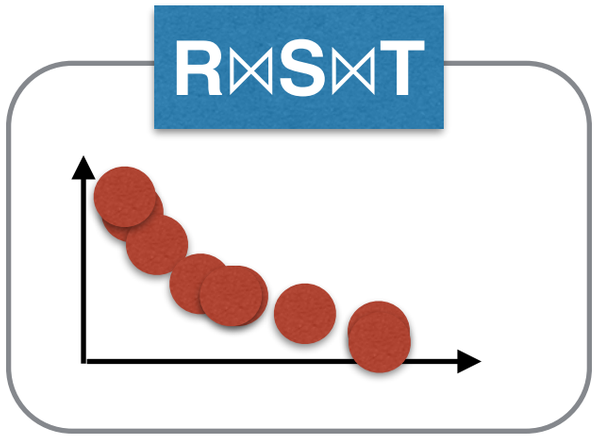
Query: R⋈S⋈T



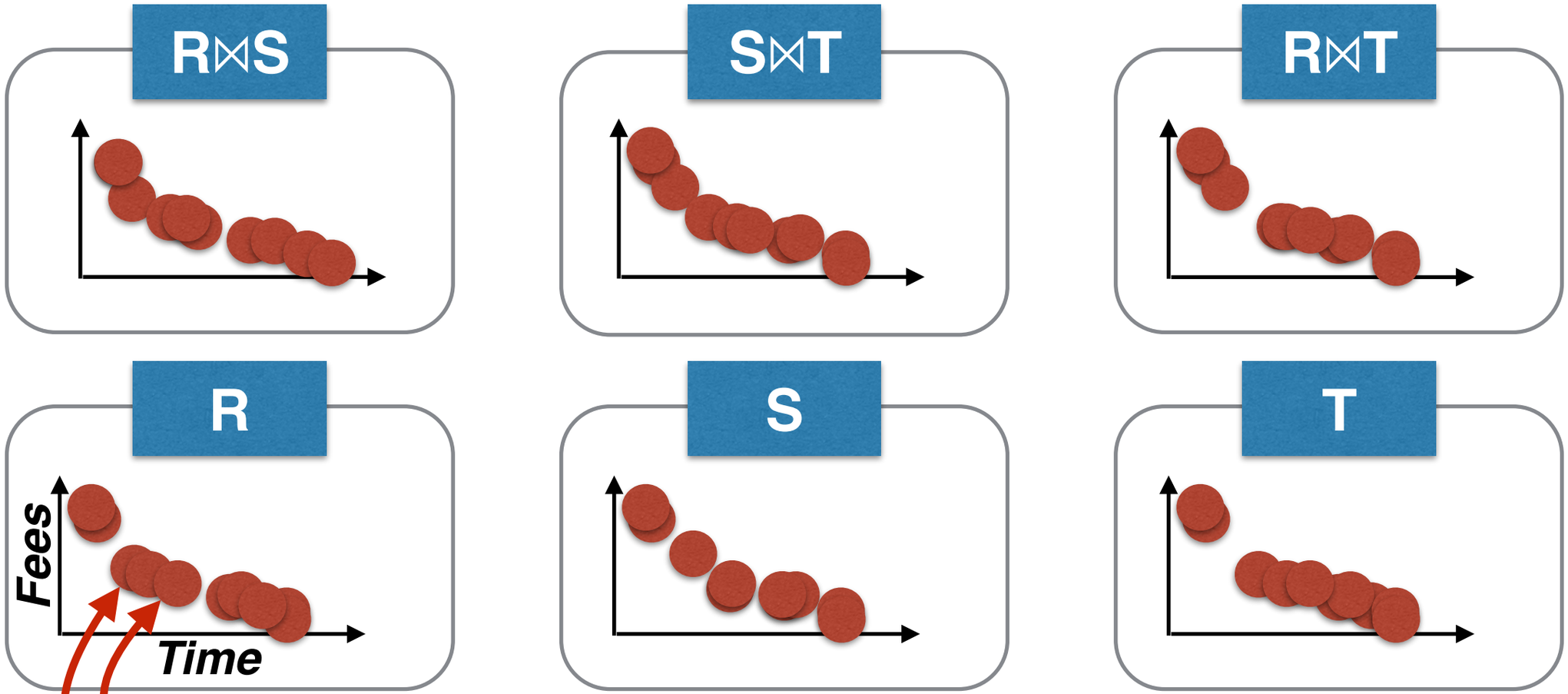
One Cost Metric

Multi-Objective Query Optimization

Query: R⋈S⋈T

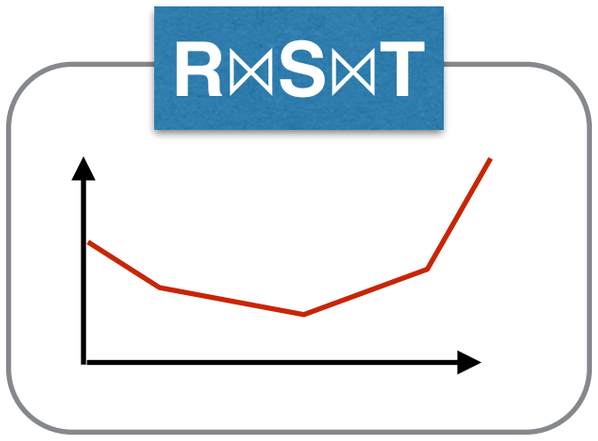


Two *Cost Metrics*



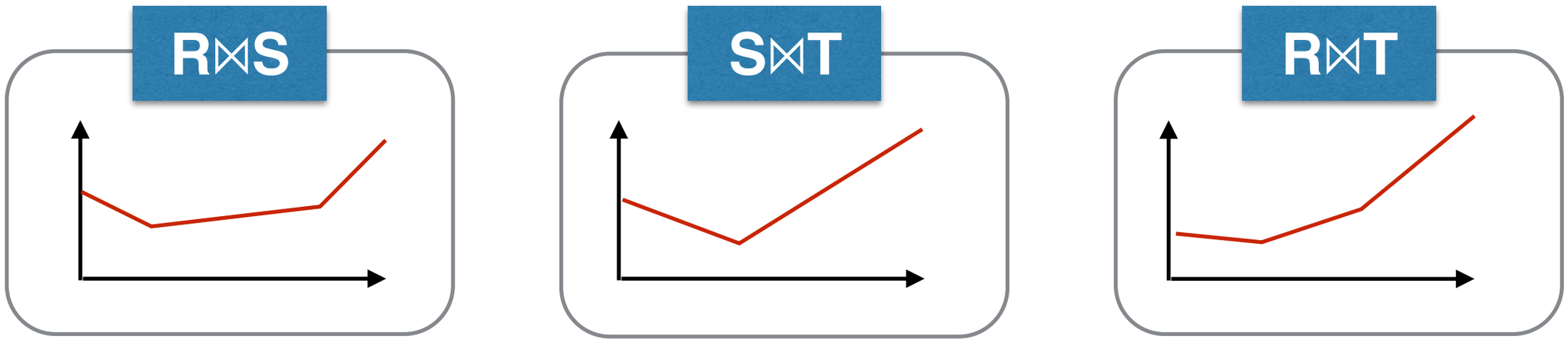
Parametric Query Optimization

Query: R⋈S⋈T



One Cost Metric

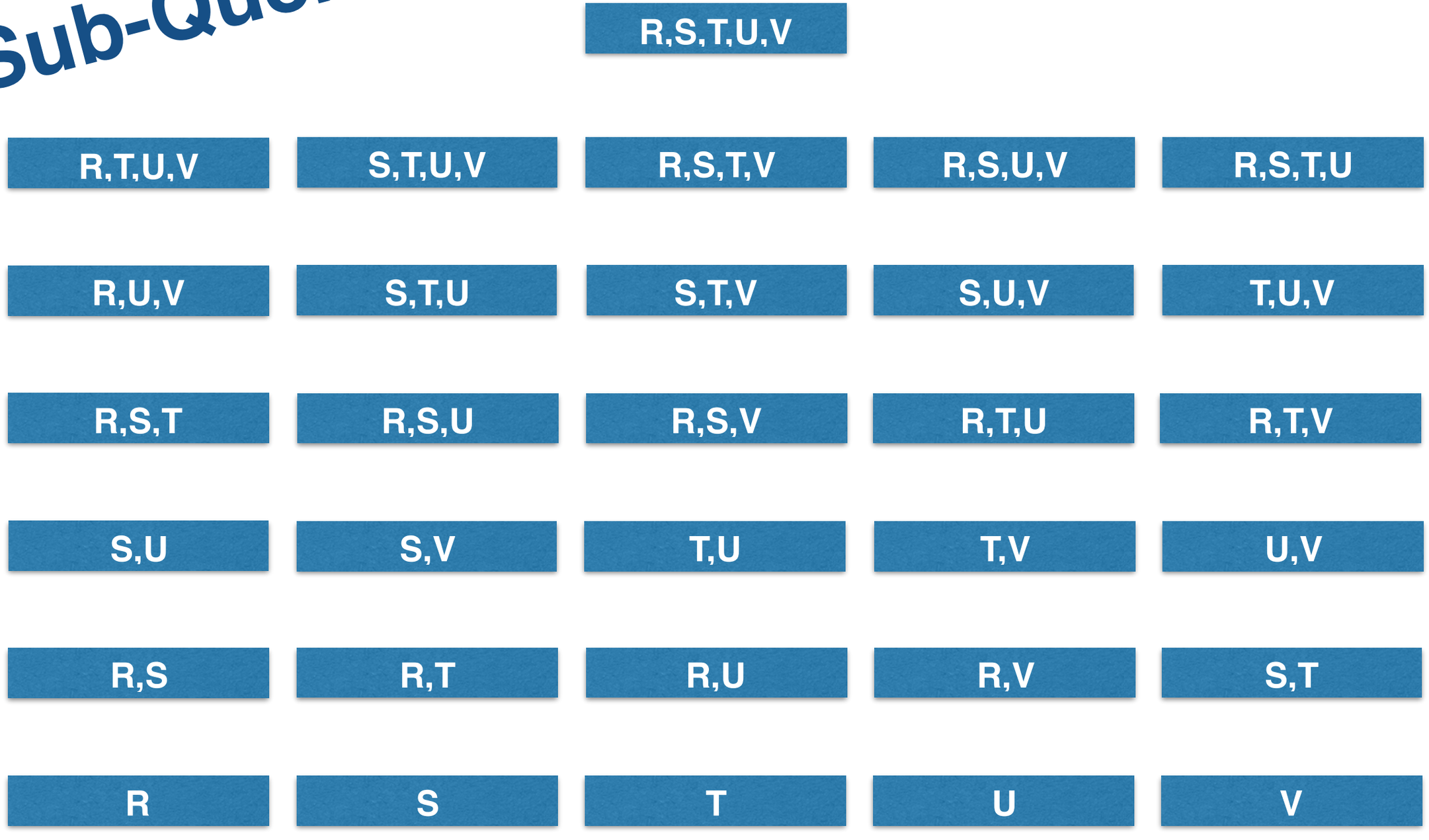
One Unknown



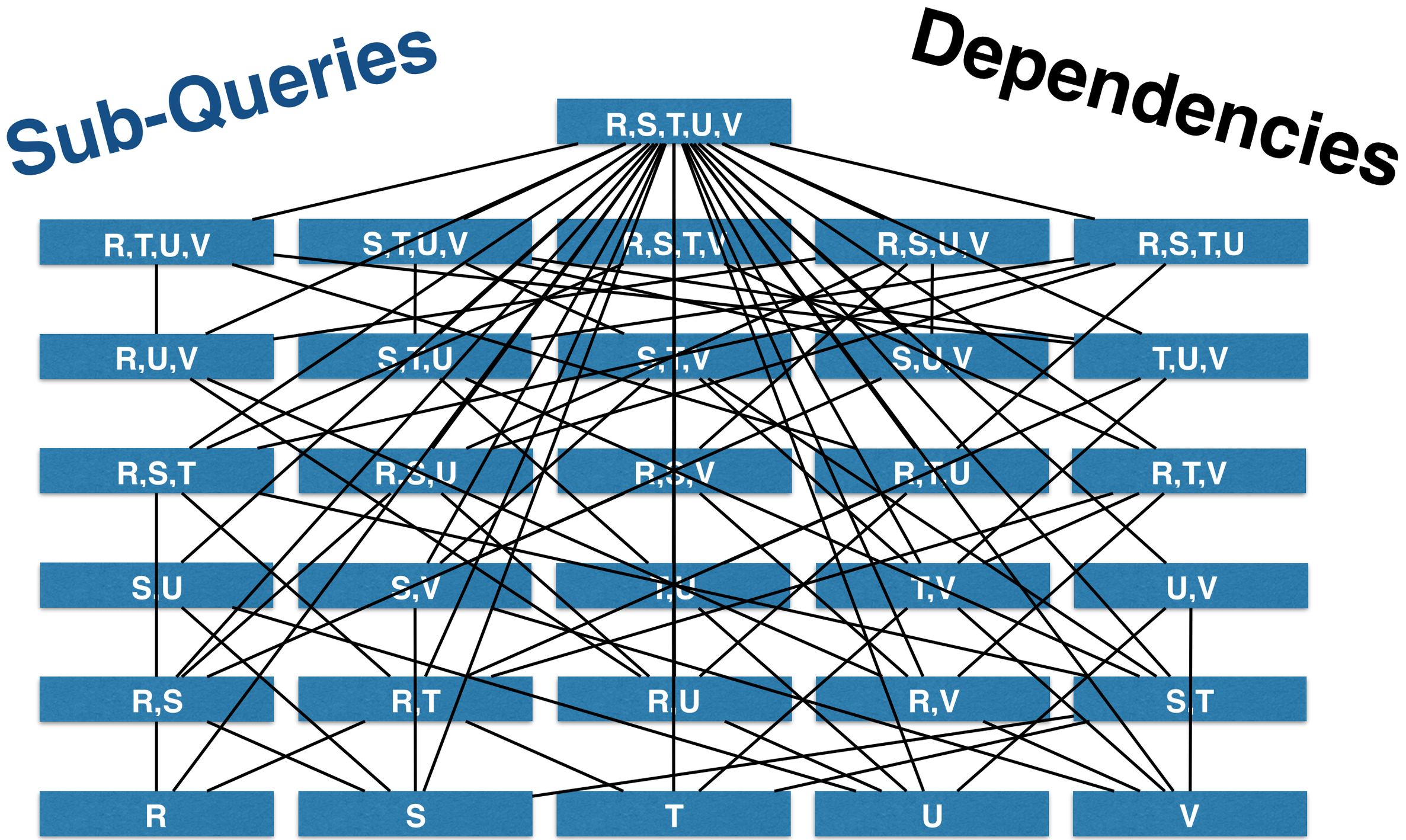
Cost Function

Parallelization Challenges

Sub-Queries



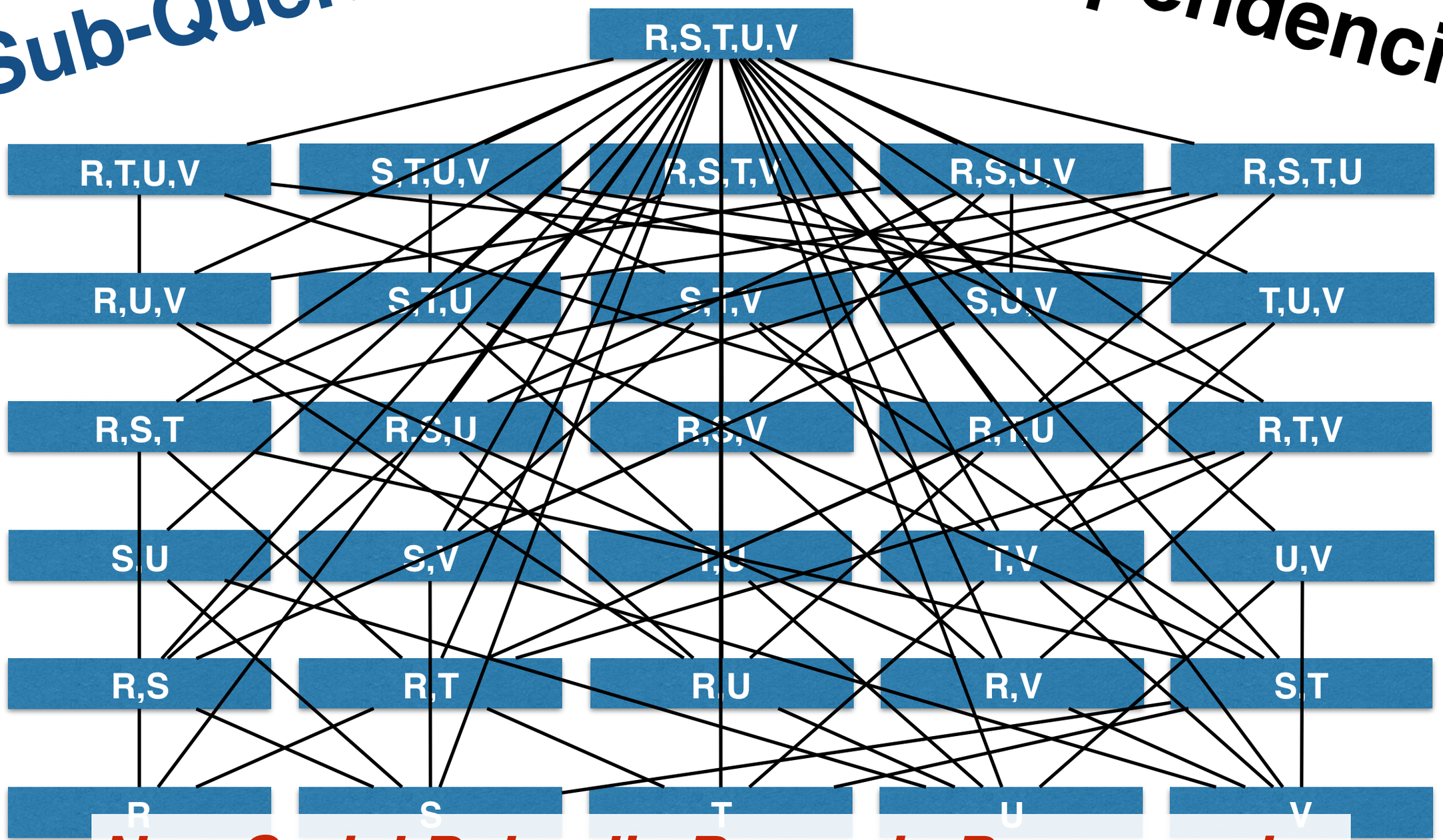
Parallelization Challenges



Parallelization Challenges

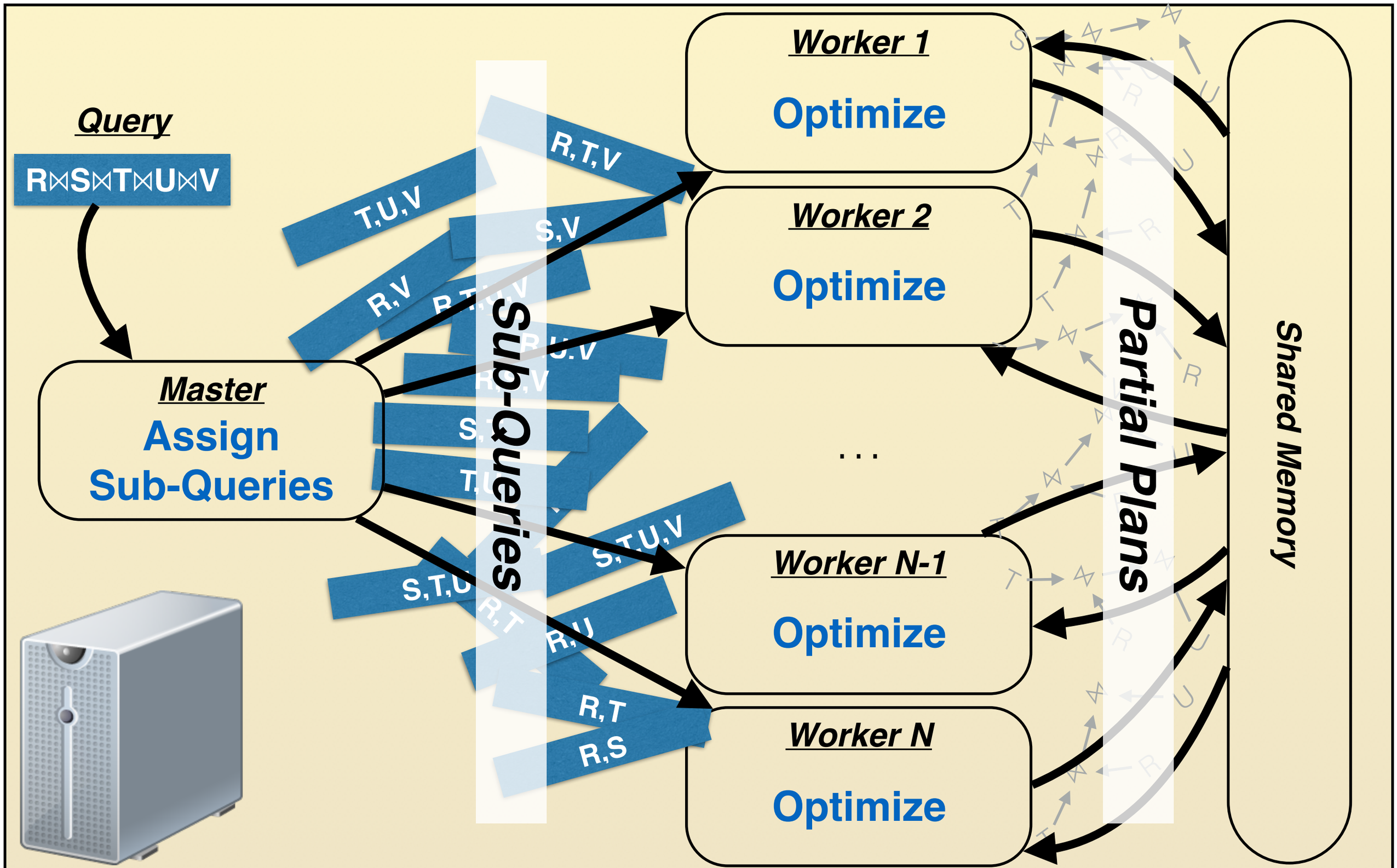
Sub-Queries

Dependencies

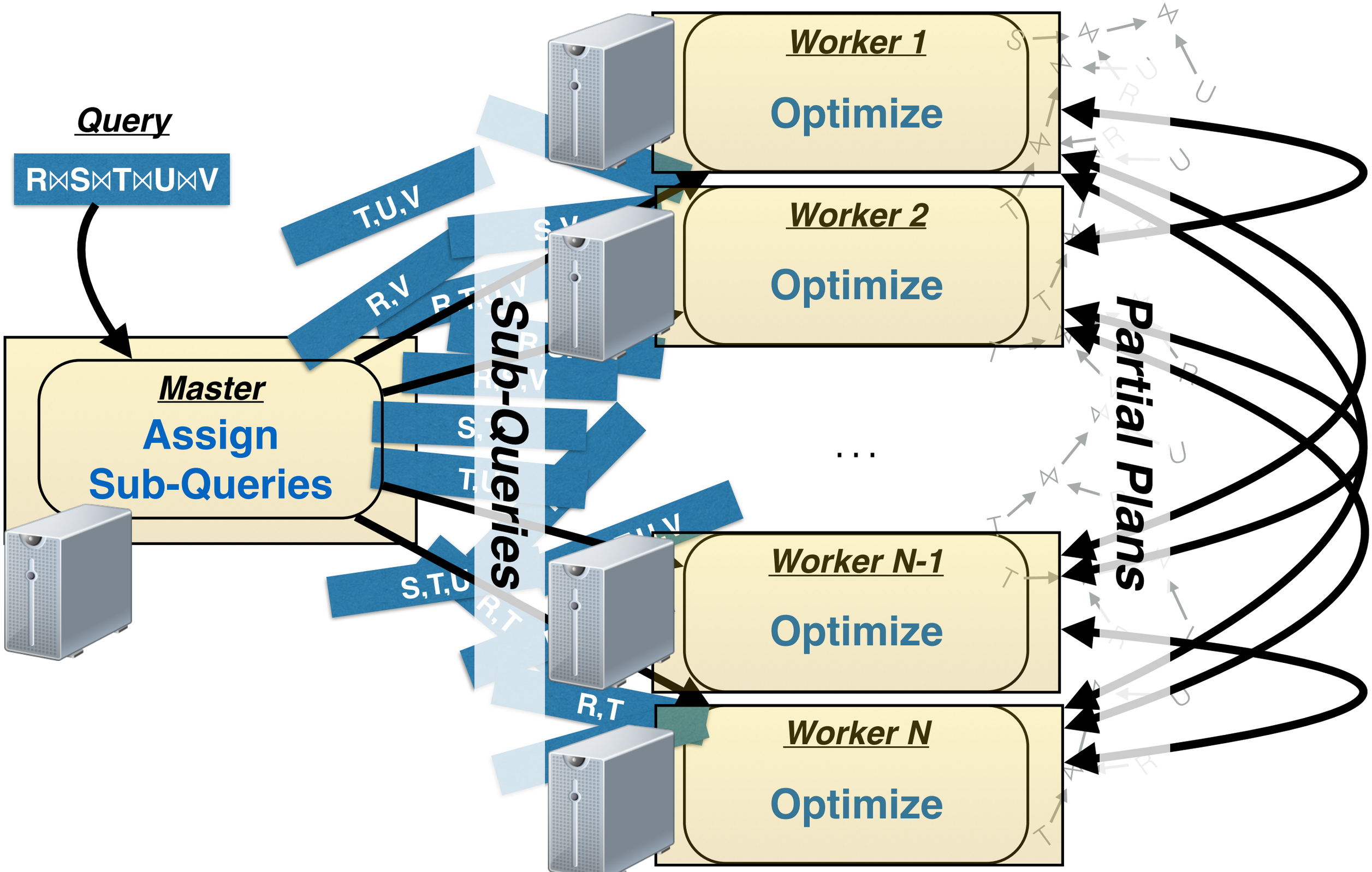


Non-Serial Polyadic Dynamic Programming

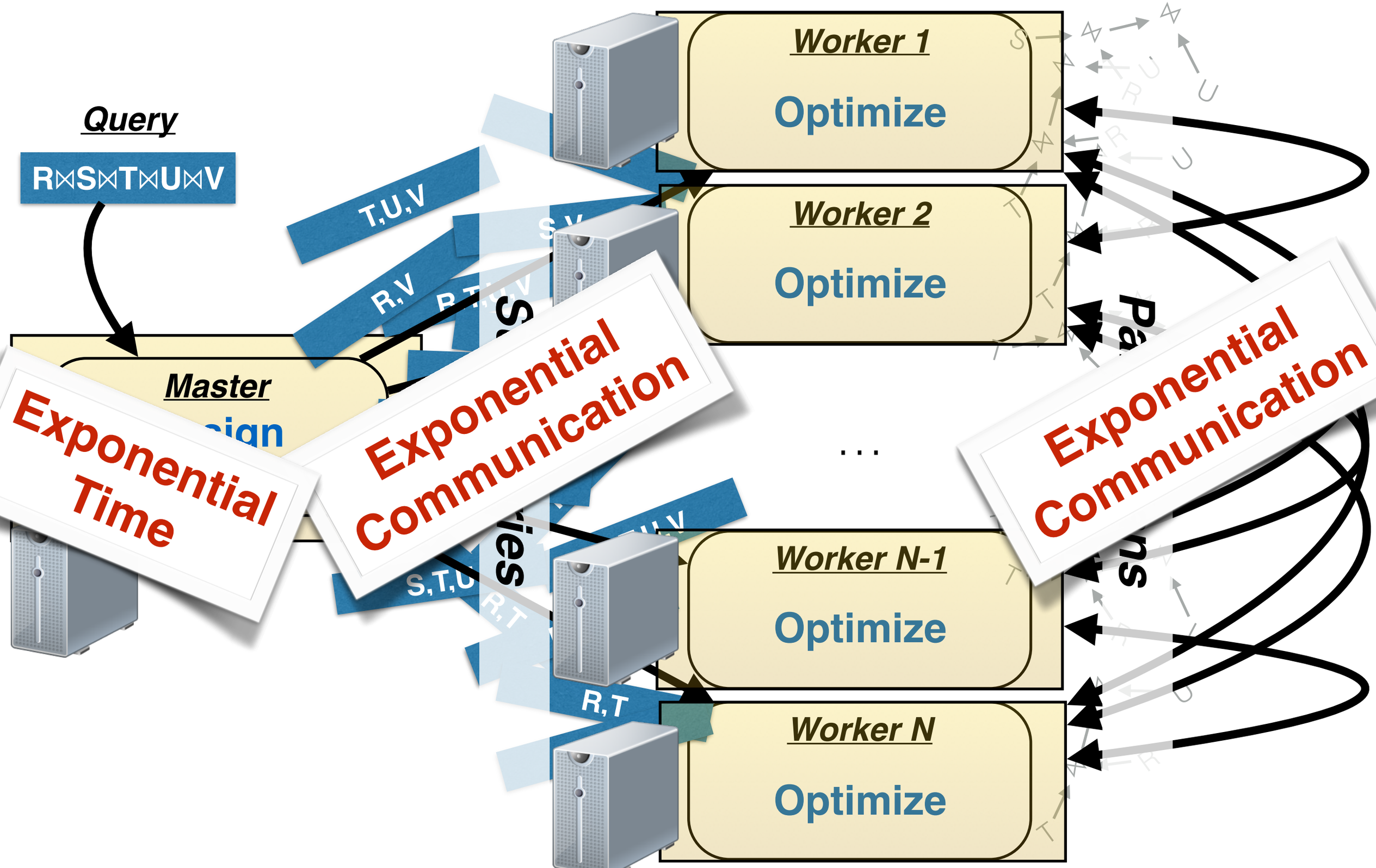
Prior Art


Query
R⋈S⋈T⋈U⋈V
Master
Assign
Sub-Queries
R,T,V
T,U,V
S,V
R,V
R,U,V
S,T,U,V
R,T
R,U
R,T
R,S
Sub-Queries
Worker 1
Optimize
Worker 2
Optimize
...
Worker N-1
Optimize
Worker N
Optimize
Partial Plans
Shared Memory

Prior Art


Query
R⋈S⋈T⋈U⋈V
Master
Assign
Sub-Queries
Sub-Queries
T,U,V
R,V
S,T,U
R,T
Worker 1
Optimize
Worker 2
Optimize
Worker N-1
Optimize
Worker N
Optimize
...
Partial Plans

Prior Art

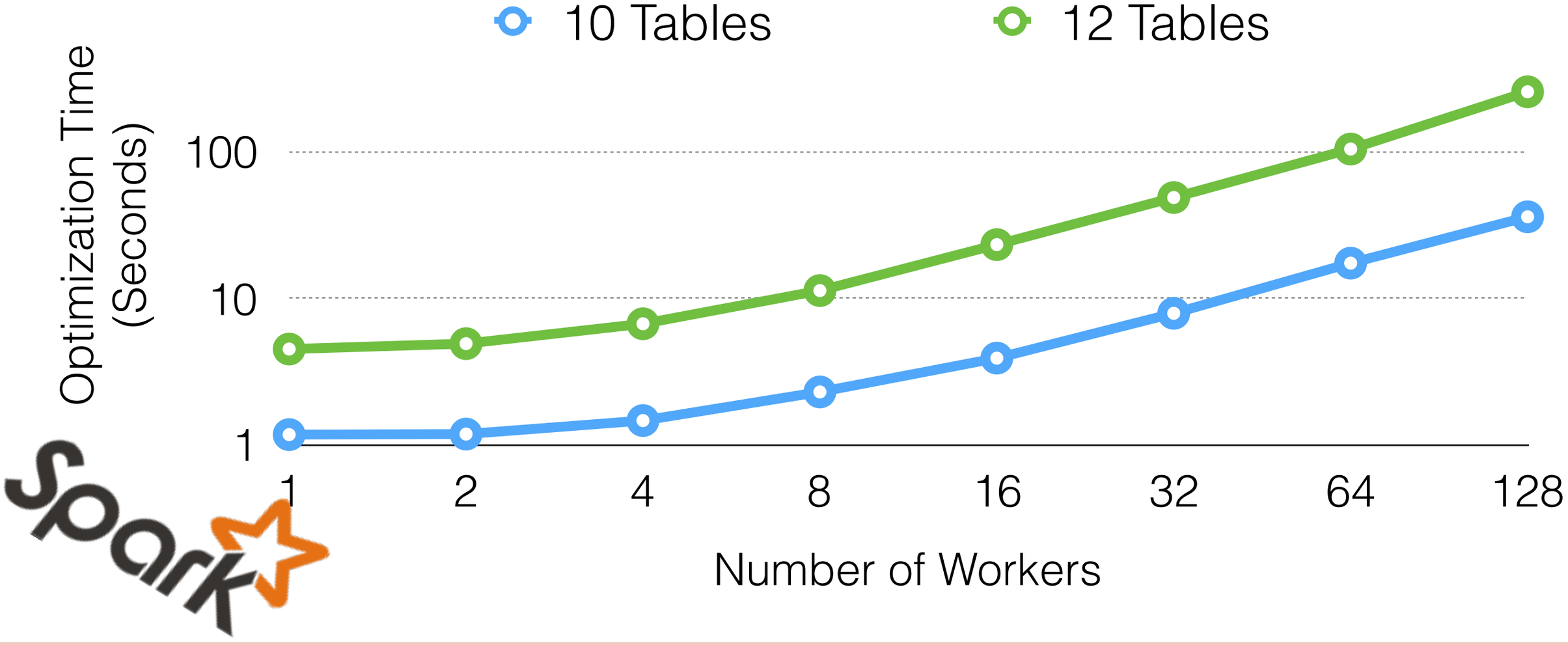

Query
R⋈S⋈T⋈U⋈V
Master
Optimize
Worker 1
Optimize
Worker 2
Optimize
Worker N-1
Optimize
Worker N
Optimize
T,U,V
R,V
S,T,U
R,T
R,T
Exponential Time
Exponential Communication
Exponential Communication
...

Experimental Results

Experimental Setup

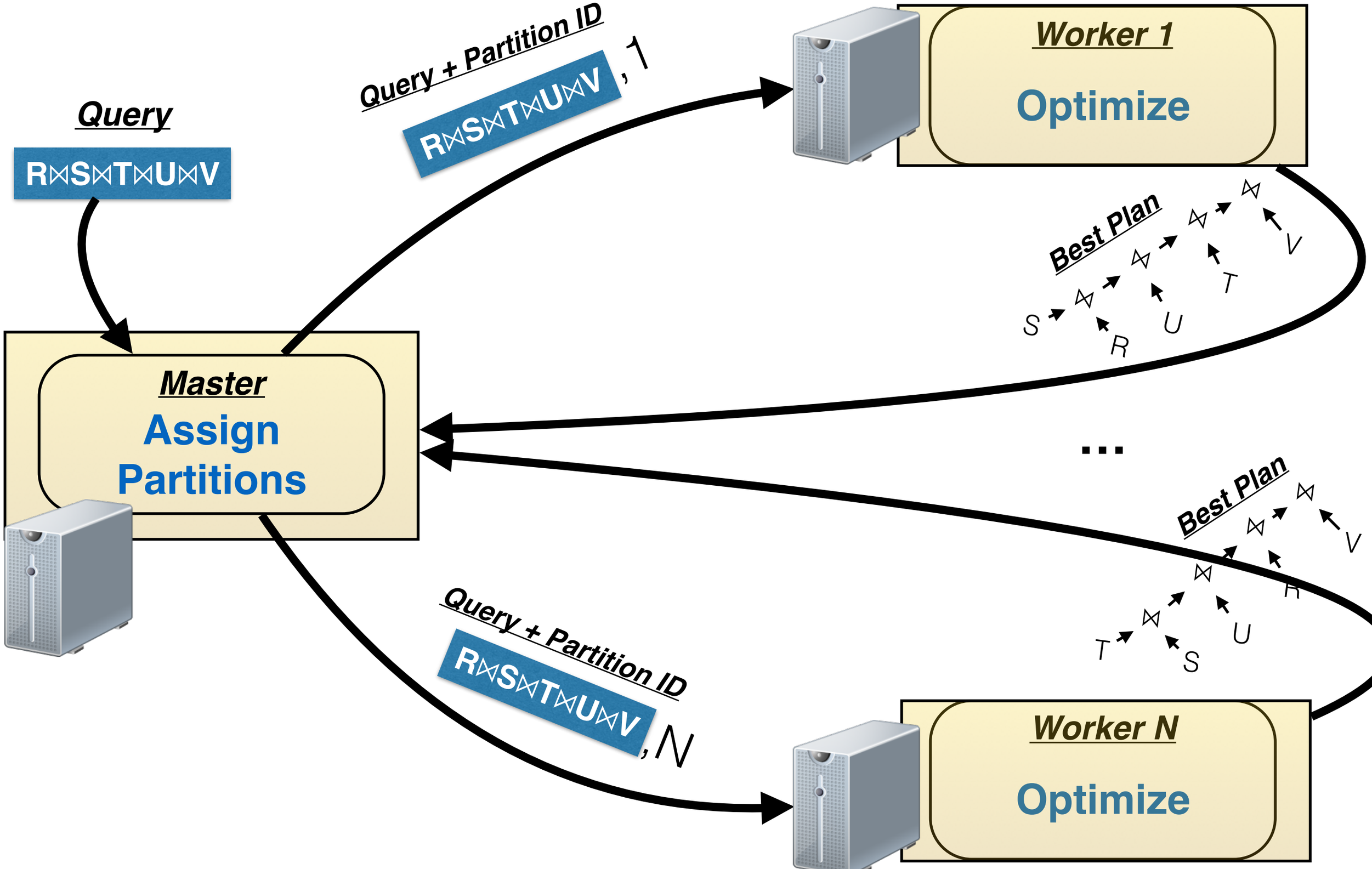
Randomly generated queries; two execution cost metrics; approximated optimization

Spark 1.5 on Yarn 2.7; 100 nodes with 2x Intel Xeon 2.6 GHz, 128 GB memory

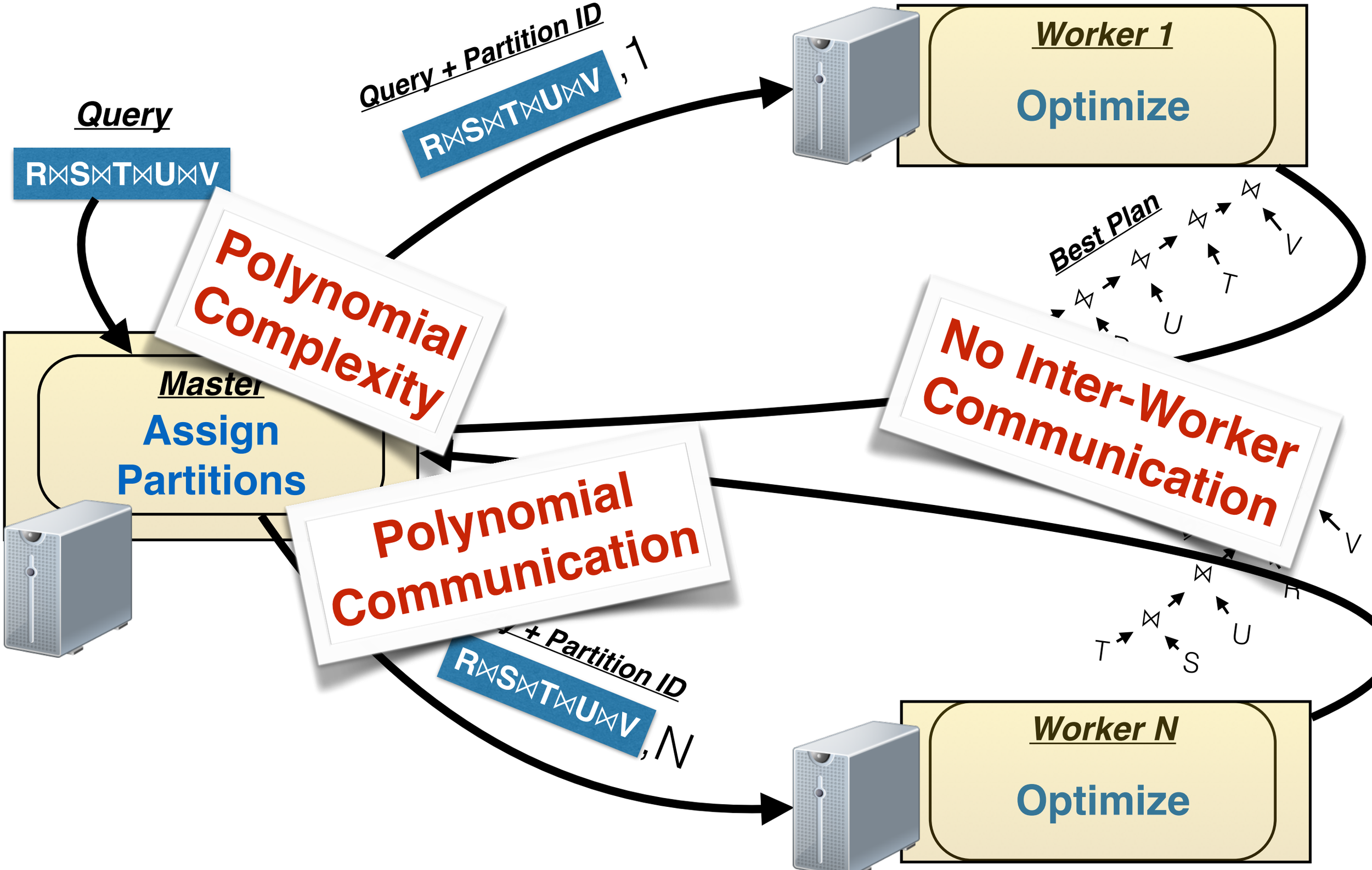


Scalability of previous parallel query optimization algorithms

New Approach



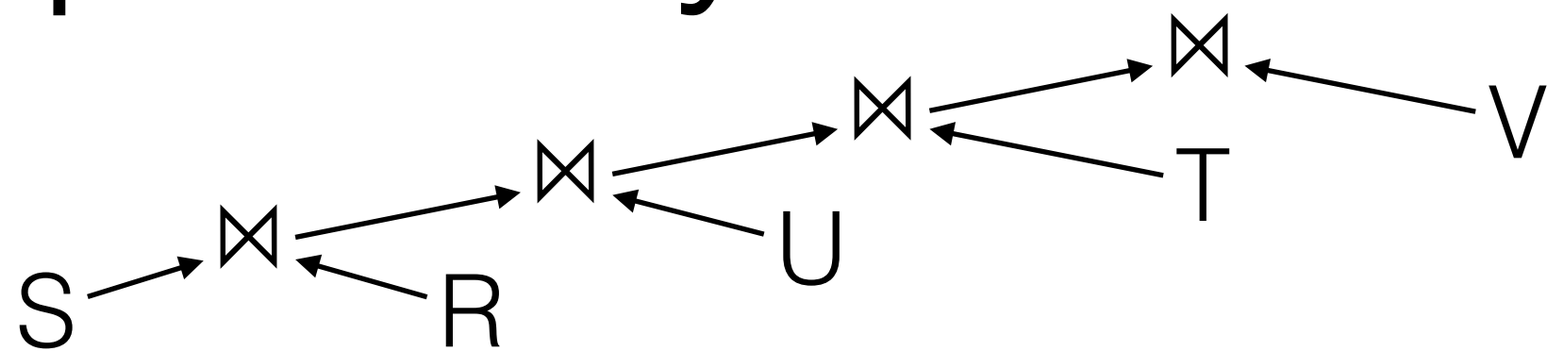
New Approach



Plan Space

Left-Deep Query Plans

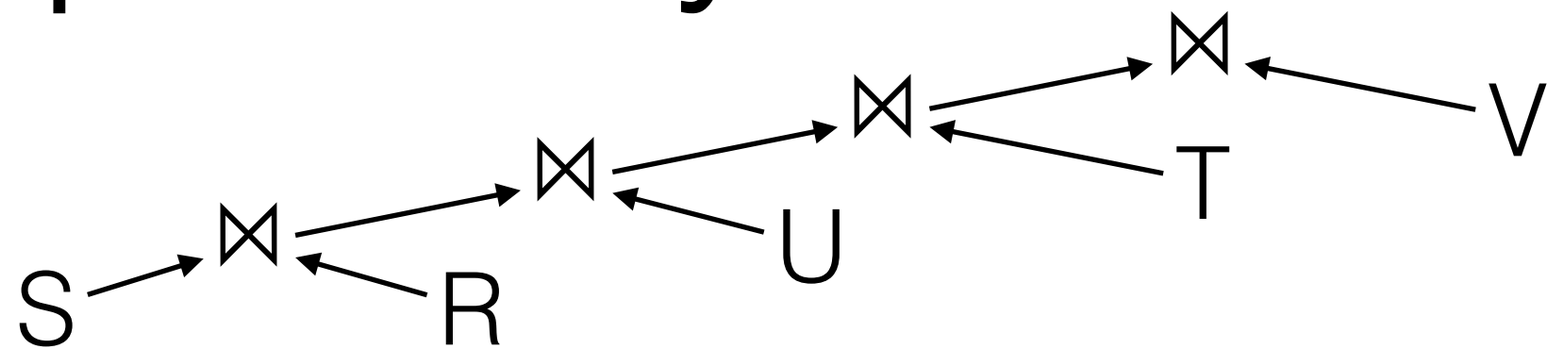
Left-Deep Plan ~ Table Order



Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

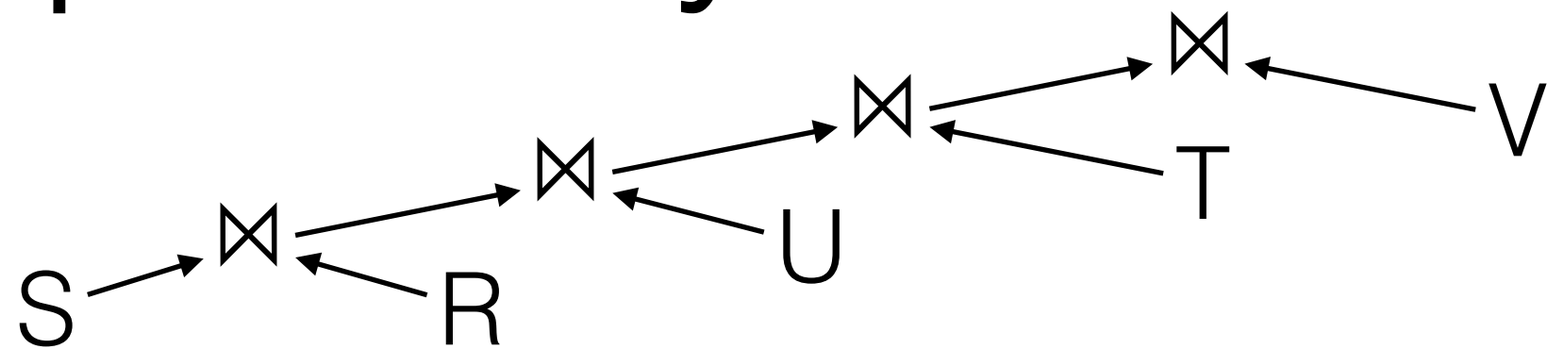
Partitioning via Unary Constraints

First: **R**	First: **S**	First: **T**	First: **U**	First: **V**

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

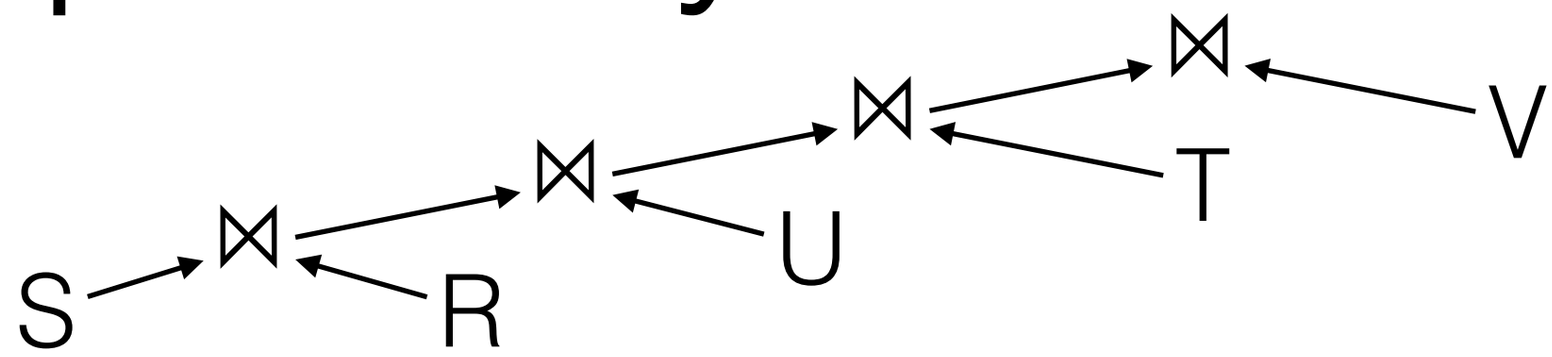
Partitioning via Unary Constraints

First: **R**	First: **S**	First: **T**	First: **U**	First: **V**

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Unary Constraints

First: **R**	First: **S**	First: **T**	First: **U**	First: **V**

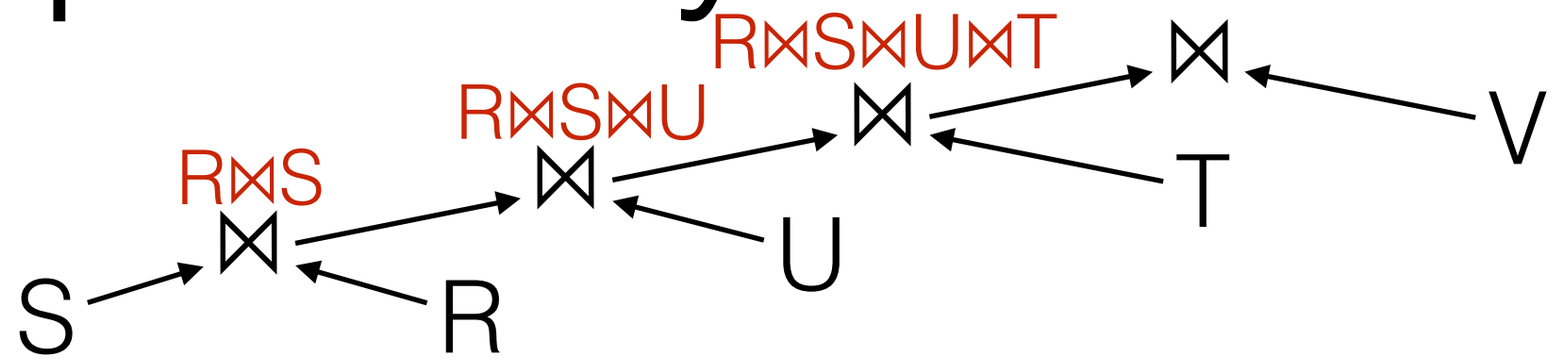
Sub-Queries

	RS	**R**S	R**S**	**RS**
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TU**V**	T,V	R,T,V	S,T,V	R,S,T,V
T**UV**	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
TU**V**		R,V	S,V	R,S,V
T**U**V		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Unary Constraints

First: **R**	First: **S**	First: **T**	First: **U**	First: **V**

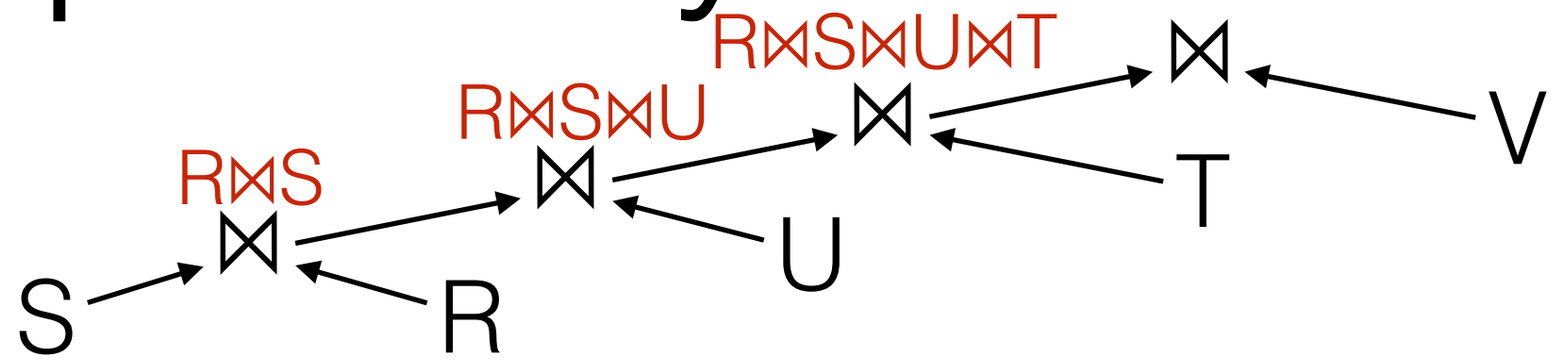
Sub-Queries

	RS	**R**S	R**S**	**RS**
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TU**V**	T,V	R,T,V	S,T,V	R,S,T,V
T**UV**	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
TU**V**		R,V	S,V	R,S,V
T**U**V		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Unary Constraints

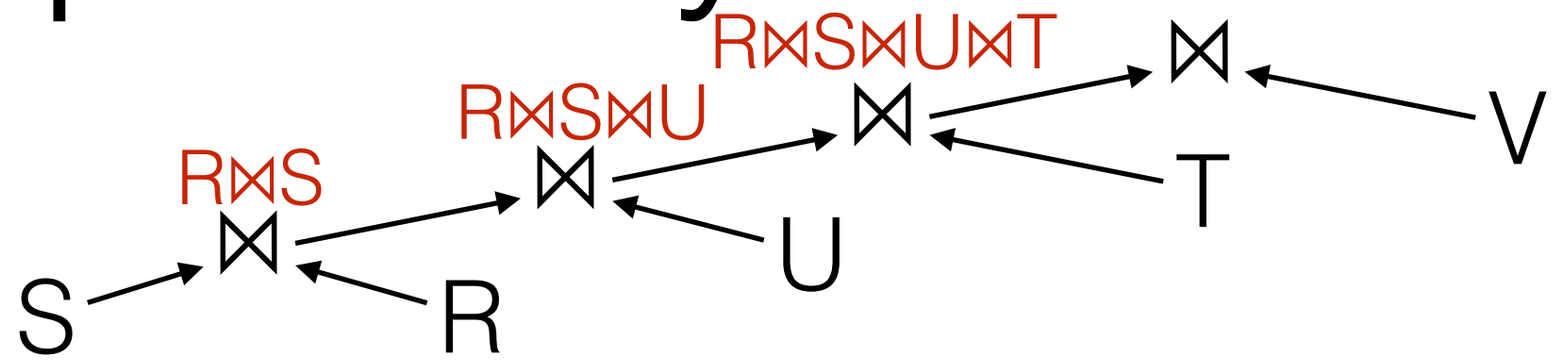
First: **R**	First: **S**	First: **T**	First: **U**	First: **V**

Sub-Queries

	RS	RS	RS	RS
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TUV	T,V	R,T,V	S,T,V	R,S,T,V
TUV	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,U	S,T,U	R,S,T,U
TUV		R,V	S,V	R,S,V
TUV		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Left-Deep Query Plans

Left-Deep Plan ~ Table Order



Plan Space

Sub-Que

Partitioning via Unary Constraints

Parallelism ~ Nr. Tables

Time / 2

TUV
TUV
TUV
TUV
TUV
TUV

T,U

R,V
R,U
R,T

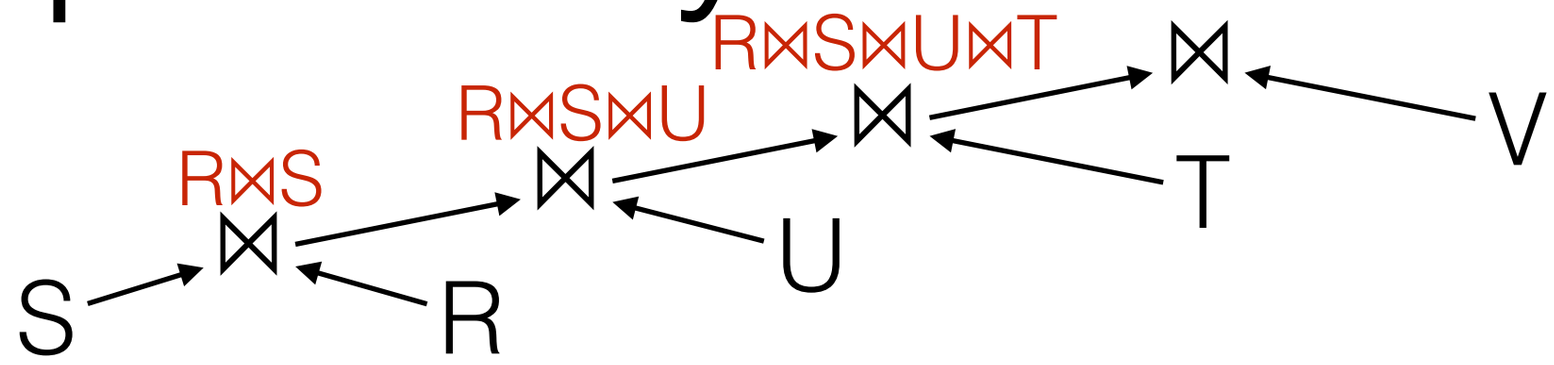
S,T,U
S,V
S,U
S,T

R,S,T,U
R,S,V
R,S,U
R,S,T
R,S

Left-Deep Query Plans

Plan Space

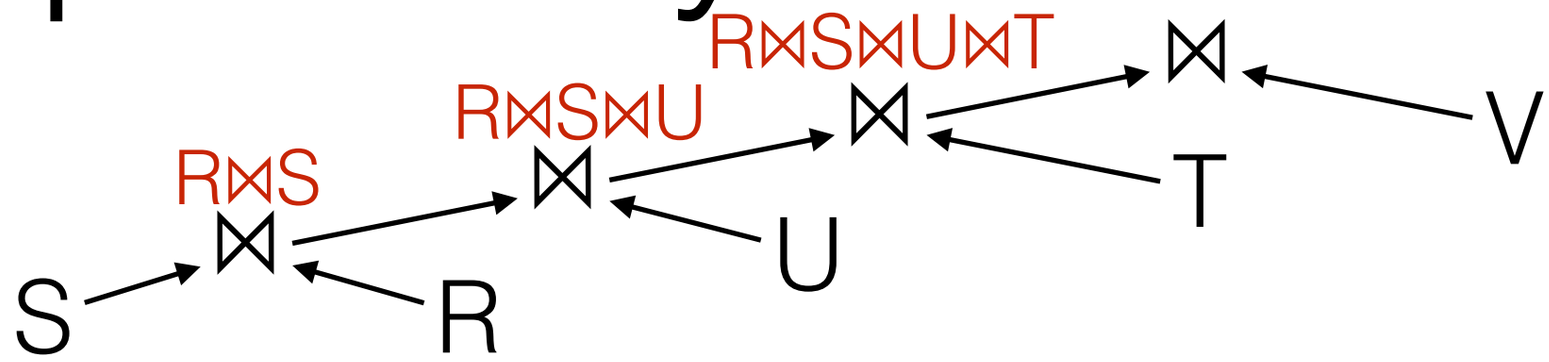
Left-Deep Plan ~ Table Order



Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Binary Constraints

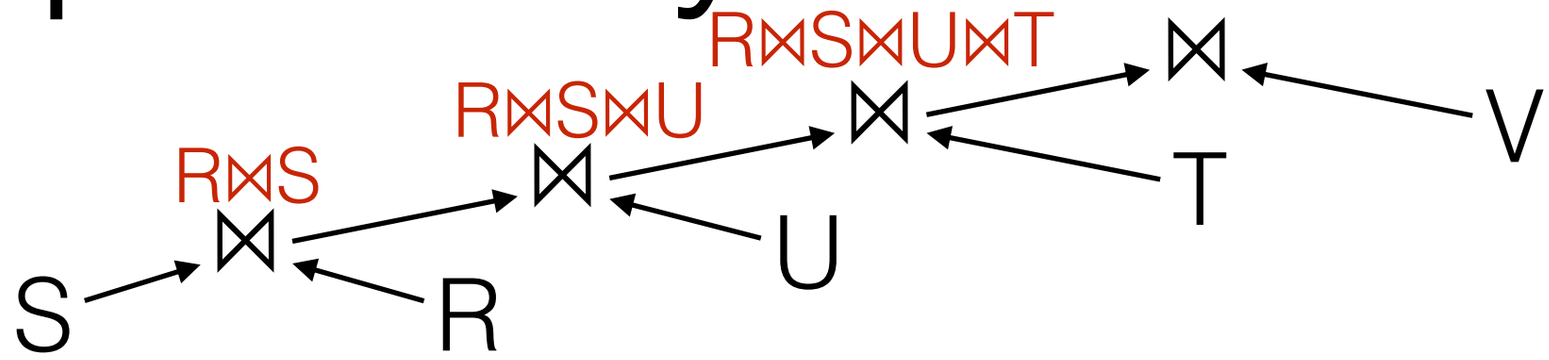
R Before **S**

S Before **R**

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Binary Constraints

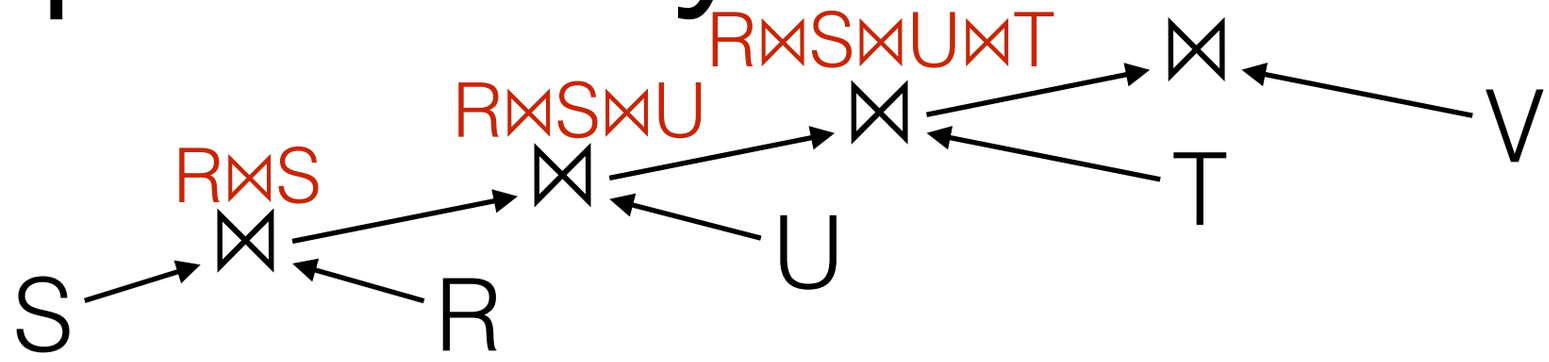
R Before **S**

S Before **R**

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Binary Constraints

R Before **S**	**S** Before **R**

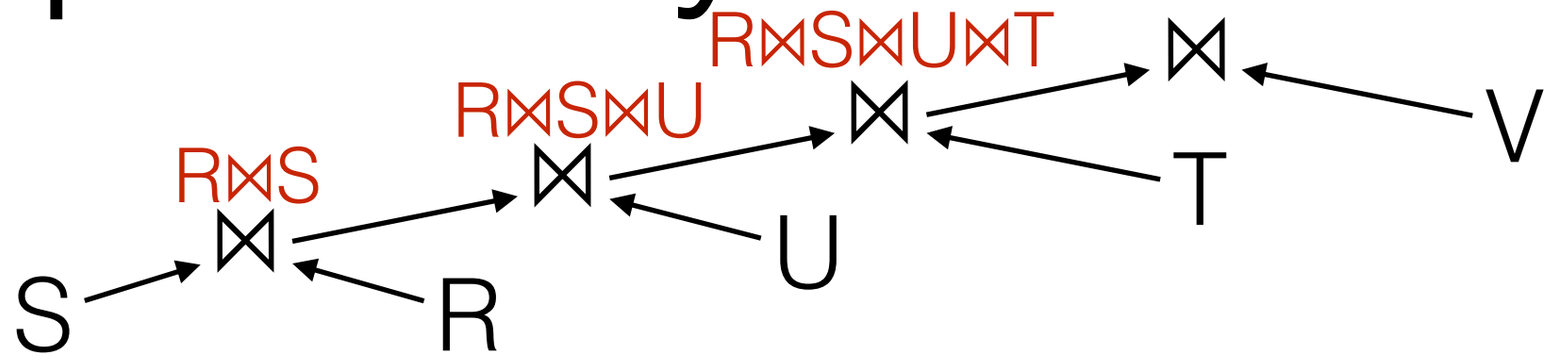
Sub-Queries

	RS	**R**S	R**S**	**RS**
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TU**V**	T,V	R,T,V	S,T,V	R,S,T,V
T**UV**	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
TU**V**		R,V	S,V	R,S,V
T**U**V		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Binary Constraints

R Before **S**	**S** Before **R**

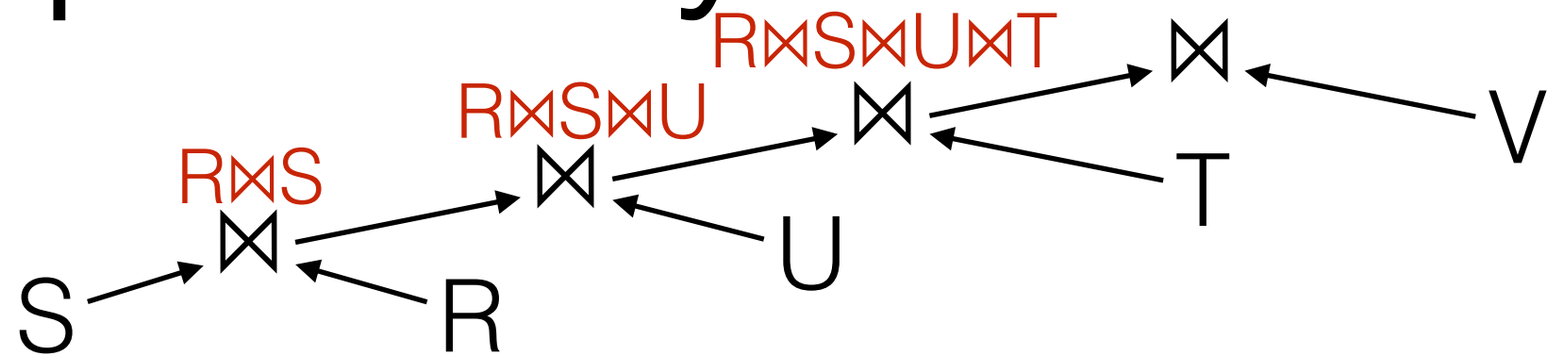
Sub-Queries

	RS	RS	RS	RS
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TUV	T,V	R,T,V	S,T,V	R,S,T,V
TUV	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,U	S,T,U	R,S,T,U
TUV		R,V	S,V	R,S,V
TUV		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Binary Constraints

R Before **S**	
T Before **U**	**U** Before **T**

S Before **R**	
T Before **U**	**U** Before **T**

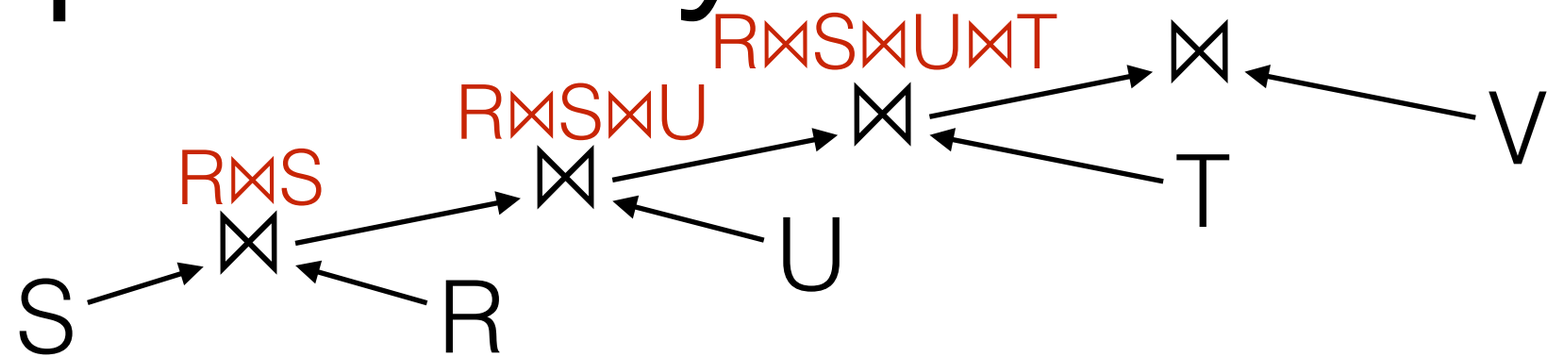
Sub-Queries

	RS	RS	RS	RS
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TU**V**	T,V	R,T,V	S,T,V	R,S,T,V
T**UV**	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
TU**V**		R,V	S,V	R,S,V
T**U**V		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Binary Constraints

R Before **S**

- **T** Before **U**
- **U** Before **T**

S Before **R**

- **T** Before **U**
- **U** Before **T**

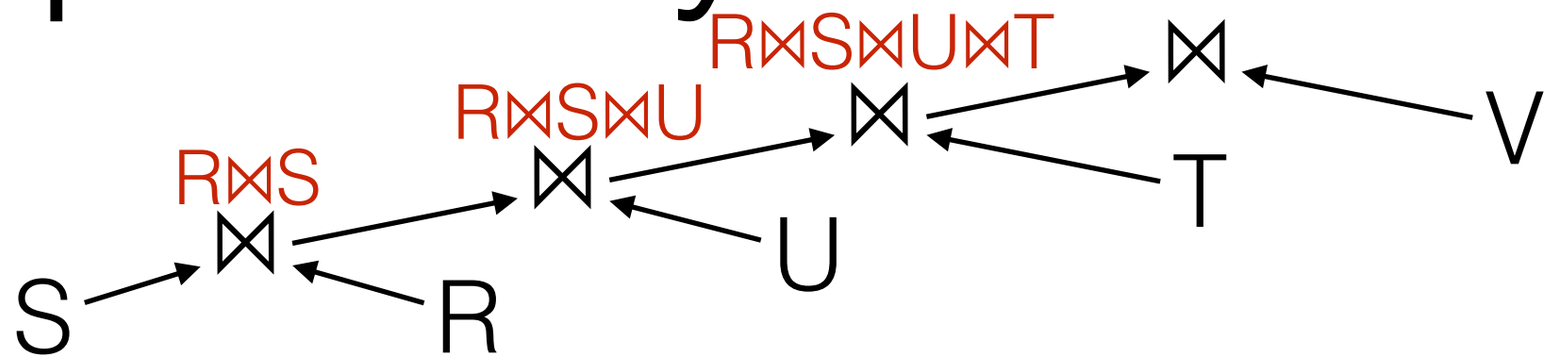
Sub-Queries

	RS	RS	RS	RS
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TU**V**	T,V	R,T,V	S,T,V	R,S,T,V
T**UV**	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,U	S,T,U	R,S,T,U
TU**V**		R,V	S,V	R,S,V
T**U**V		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



Partitioning

Partitioning via Binary Constraints

R Before **S**

- **T** Before **U**
- **U** Before **T**

S Before **R**

- **T** Before **U**
- **U** Before **T**

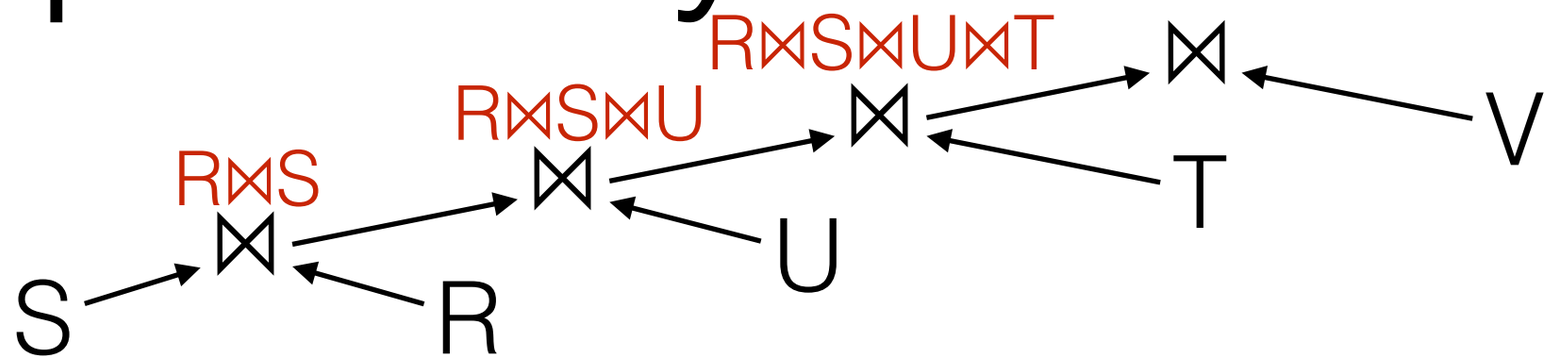
Sub-Queries

	RS	RS	RS	RS
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TU**V**	T,V	R,T,V	S,T,V	R,S,T,V
T**UV**	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
TU**V**		R,V	S,V	R,S,V
T**U**V		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Left-Deep Query Plans

Plan Space

Left-Deep Plan ~ Table Order



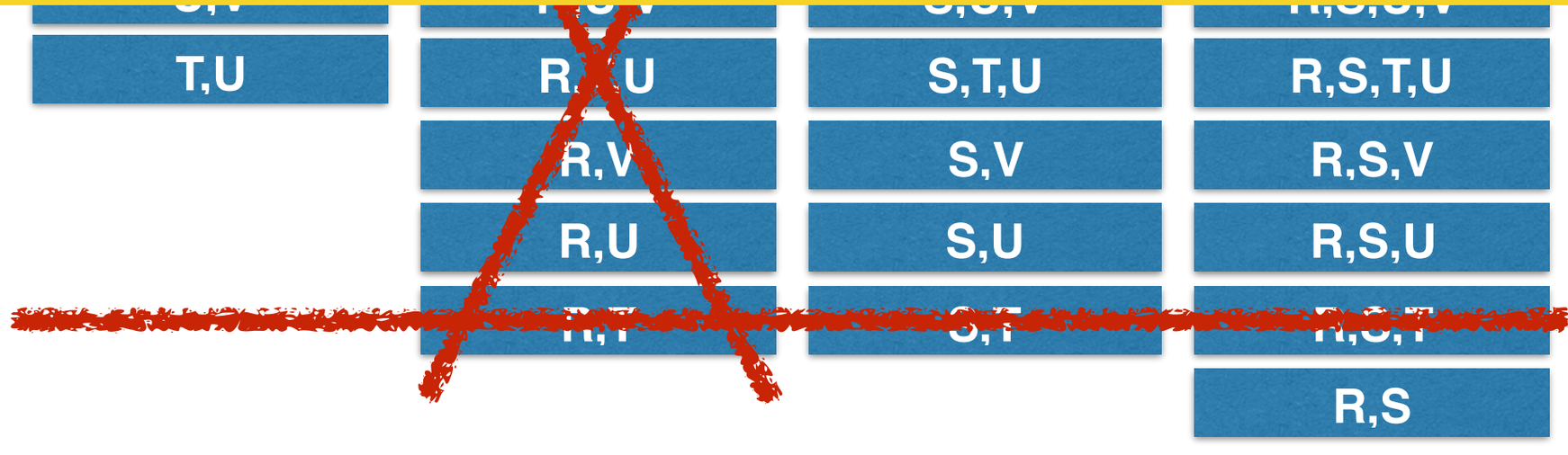
Partitioning

Partitioning via Binary Constraints

Parallelism* *2^nrPartitionLevels

***Time* *(3/4)^nrPartitionLevels**

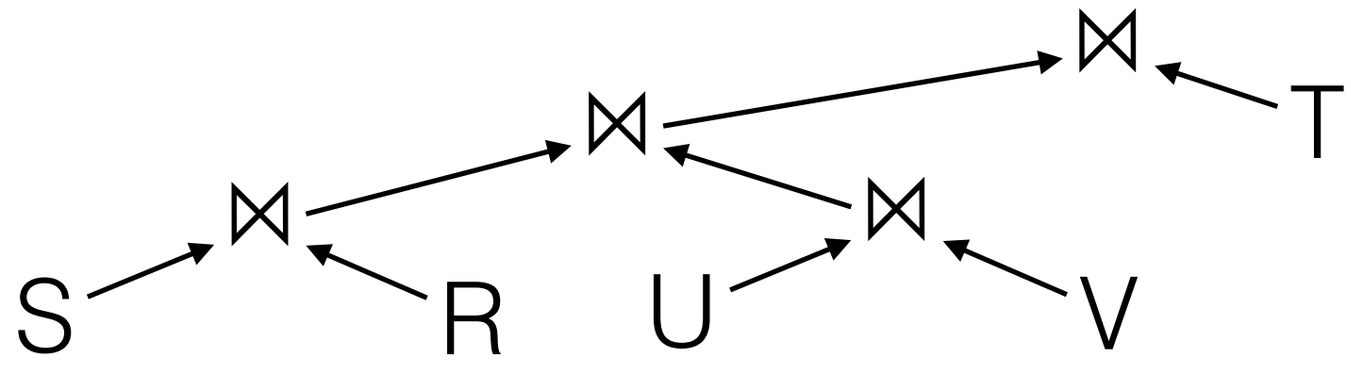
Sub-Queries



Bushy Query Plans

Plan Space

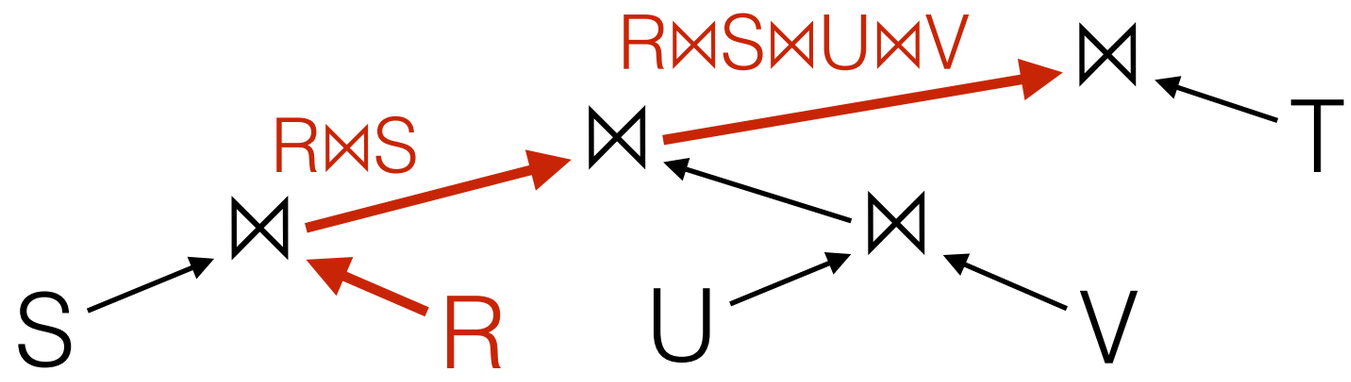
Bushy Plan ~ Binary Tree



Bushy Query Plans

Plan Space

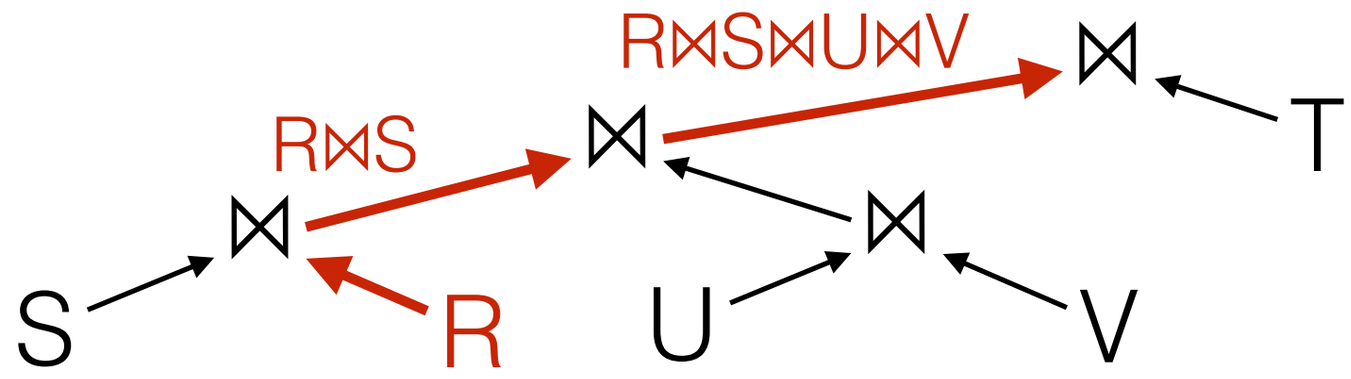
Bushy Plan ~ Binary Tree



Bushy Query Plans

Plan Space

Bushy Plan ~ Binary Tree



Partitioning

Partitioning via Ternary Constraints

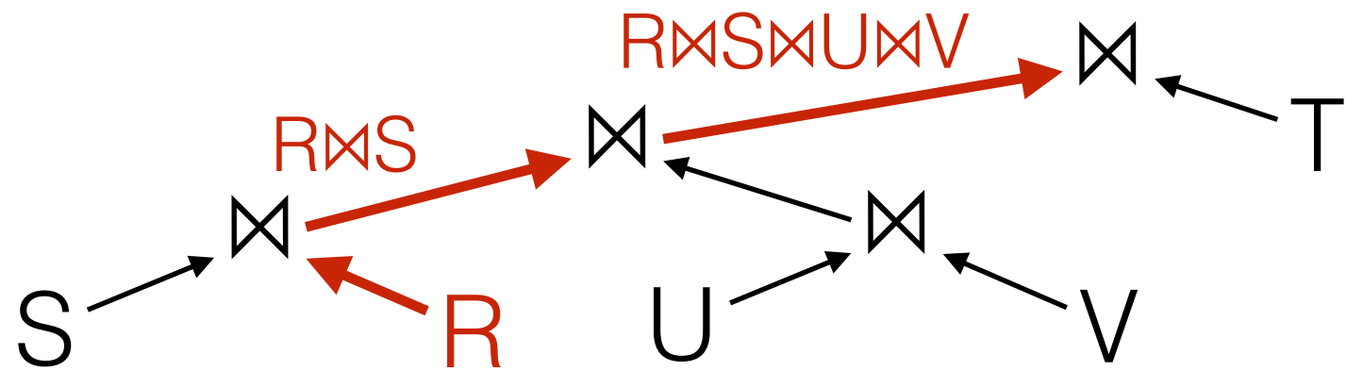
Path **R→Root: T** after **U**

Path **R→Root: T** not after **U**

Bushy Query Plans

Plan Space

Bushy Plan ~ Binary Tree



Partitioning

Partitioning via Ternary Constraints

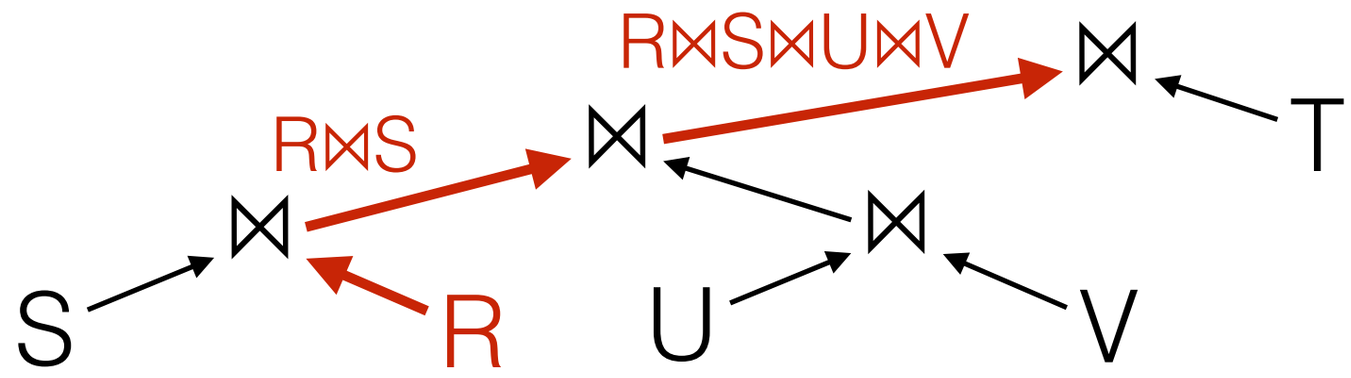
Path **R→Root: T** after **U**

Path **R→Root: T** not after **U**

Bushy Query Plans

Plan Space

Bushy Plan ~ Binary Tree



Partitioning

Partitioning via Ternary Constraints

Path **R→Root: T** after **U**

Path **R→Root: T** not after **U**

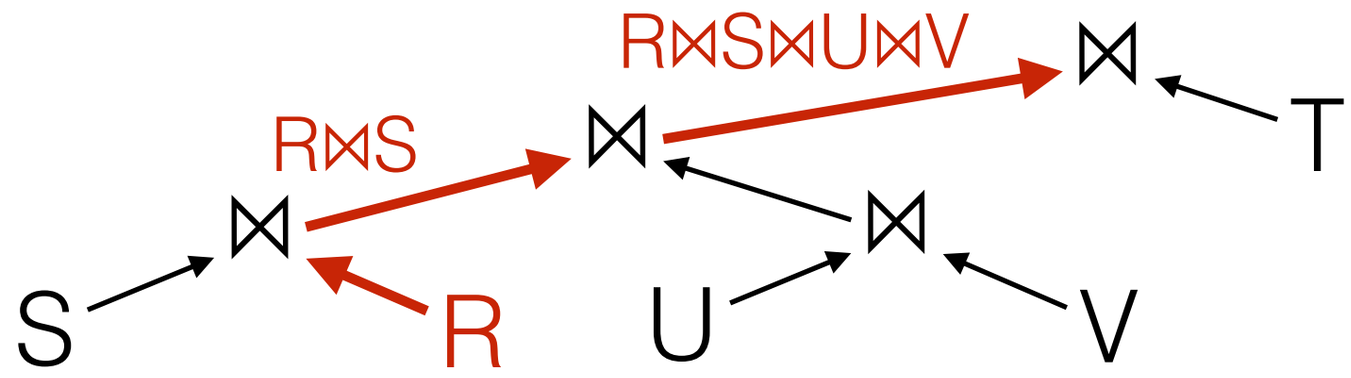
Sub-Queries

	RS	**R**S	R**S**	**RS**
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TU**V**	T,V	R,T,V	S,T,V	R,S,T,V
T**UV**	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
T**U**V		R,V	S,V	R,S,V
TU**V**		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Bushy Query Plans

Plan Space

Bushy Plan ~ Binary Tree



Partitioning

Partitioning via Ternary Constraints

Path **R→Root: T** after **U**

Path **R→Root: T** not after **U**

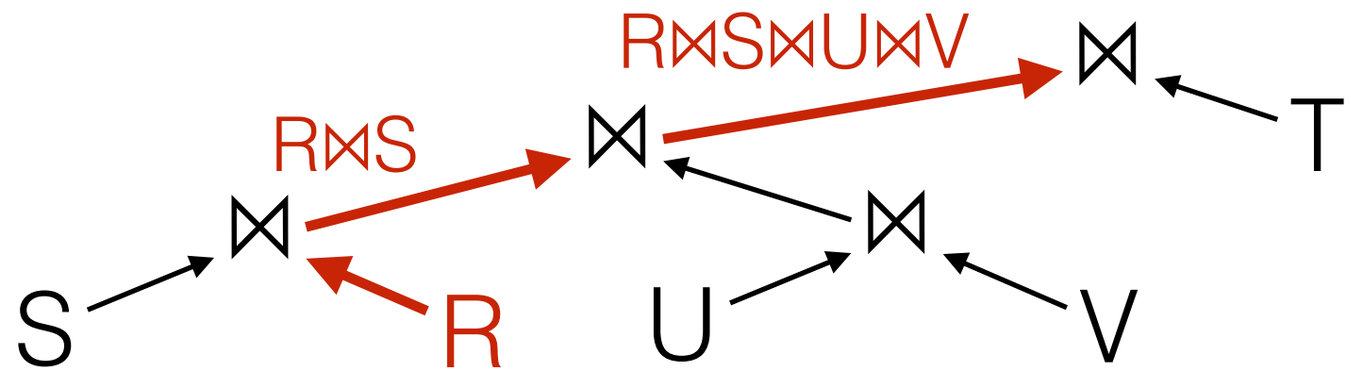
Sub-Queries

	RS	**R**S	R**S**	**RS**
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
T**U**V	T,V	R,T,V	S,T,V	R,S,T,V
T**UV**	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
TU**V**		R,V	S,V	R,S,V
T**U**V		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Bushy Query Plans

Plan Space

Bushy Plan ~ Binary Tree



Partitioning

Partitioning via Ternary Constraints

Path **R→Root: T** after **U**

Path **R→Root: T** not after **U**

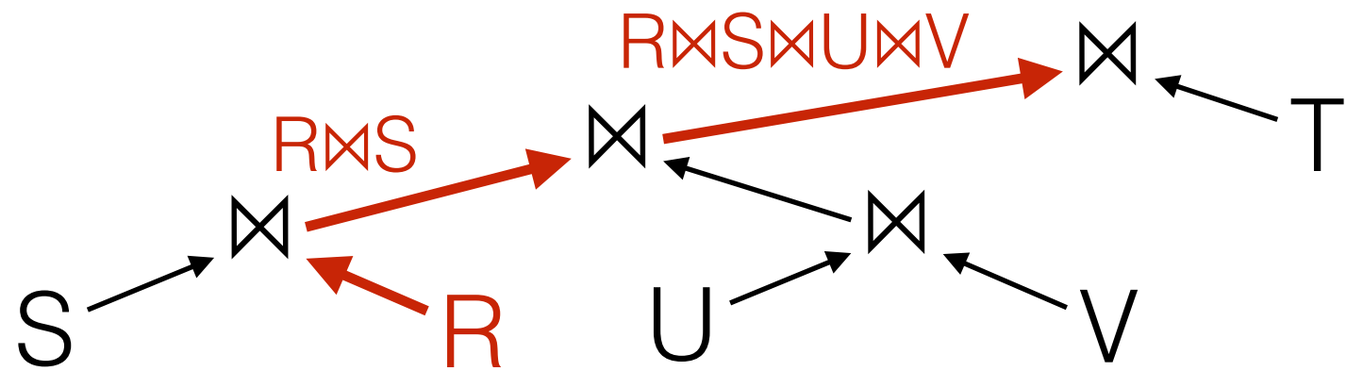
Sub-Queries

	RS	**R**S	R**S**	**RS**
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
T**U**V	T,V	R,T,V	S,T,V	R,S,T,V
T**UV**	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
T**UV**		R,V	S,V	R,S,V
T**U**V		R,U	S,U	R,S,U
T**UV**		R,T	S,T	R,S,T
TUV				R,S

Bushy Query Plans

Plan Space

Bushy Plan ~ Binary Tree



Partitioning

Partitioning via Ternary Constraints

Path **R→Root: T** after **U**

Path **R→Root: T** not after **U**

Sub-Queries

	RS	RS	RS	RS
TUV	T,U,V	R,T,U,V	S,T,U,V	R,S,T,U,V
TUV	T,V	R,T,V	S,T,V	R,S,T,V
TUV	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
TUV		R,V	S,V	R,S,V
TUV		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Bushy Query Plans

Space

Bushy Plan ~

Parallelism ****2***^{nrPartitionLevels}

Sub-Queries ****(7/8)***^{nrPartitionLevels}

Time ****(21/27)***^{nrPartitionLevels}

Sub-Queries

TUV	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
TUV		R,V	S,V	R,S,V
TUV		R,U	S,U	R,S,U
TUV		R,T	S,T	R,S,T
TUV				R,S

Bushy Query Plans

Space

Bushy Plan ~

Parallelism ****2*** ***nrPartitionLevels***

Sub-Queries *(7/8) ***nrPartitionLevels***

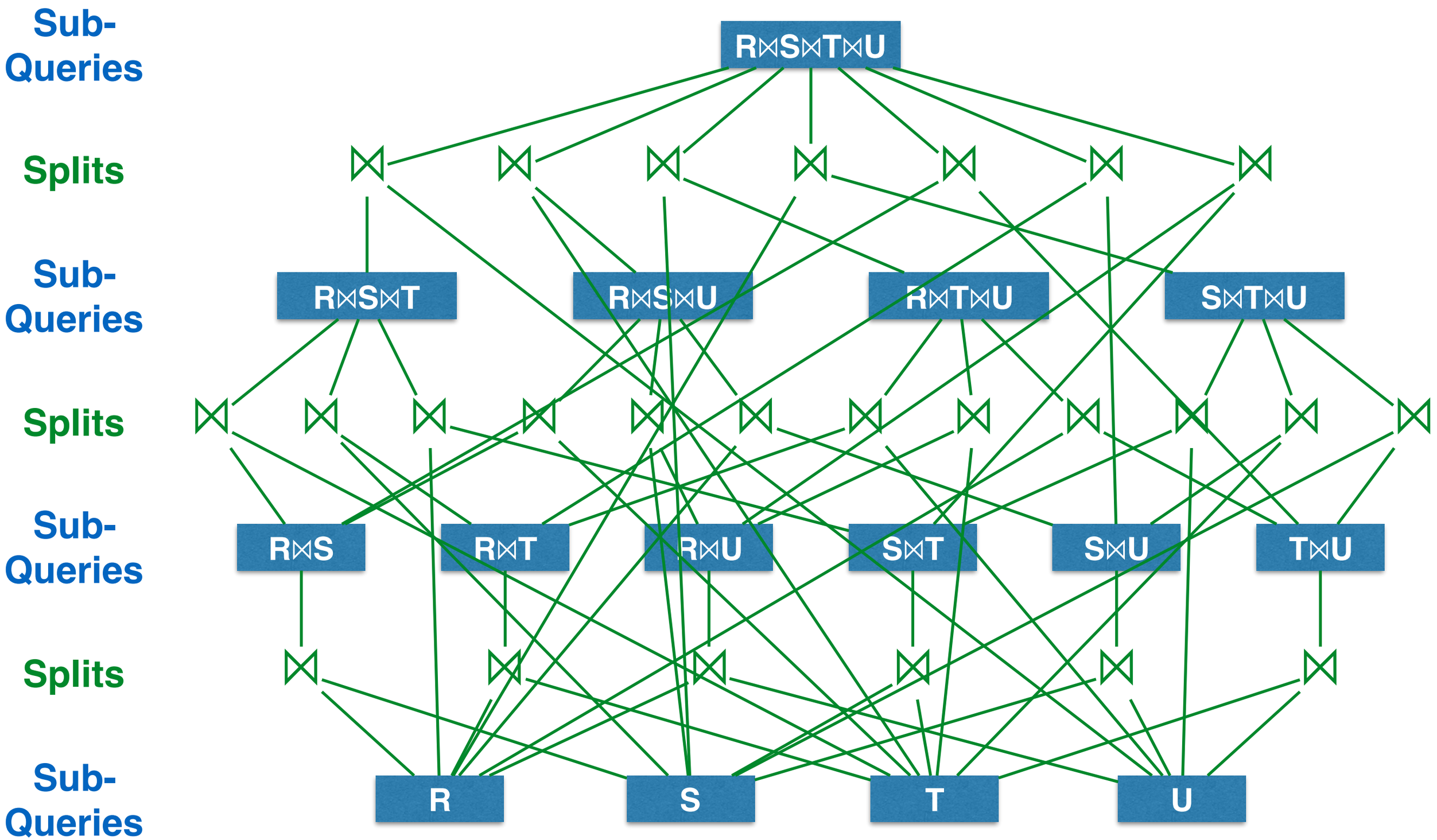
Time ****(21/27)*** ***ionLevels***

?!?

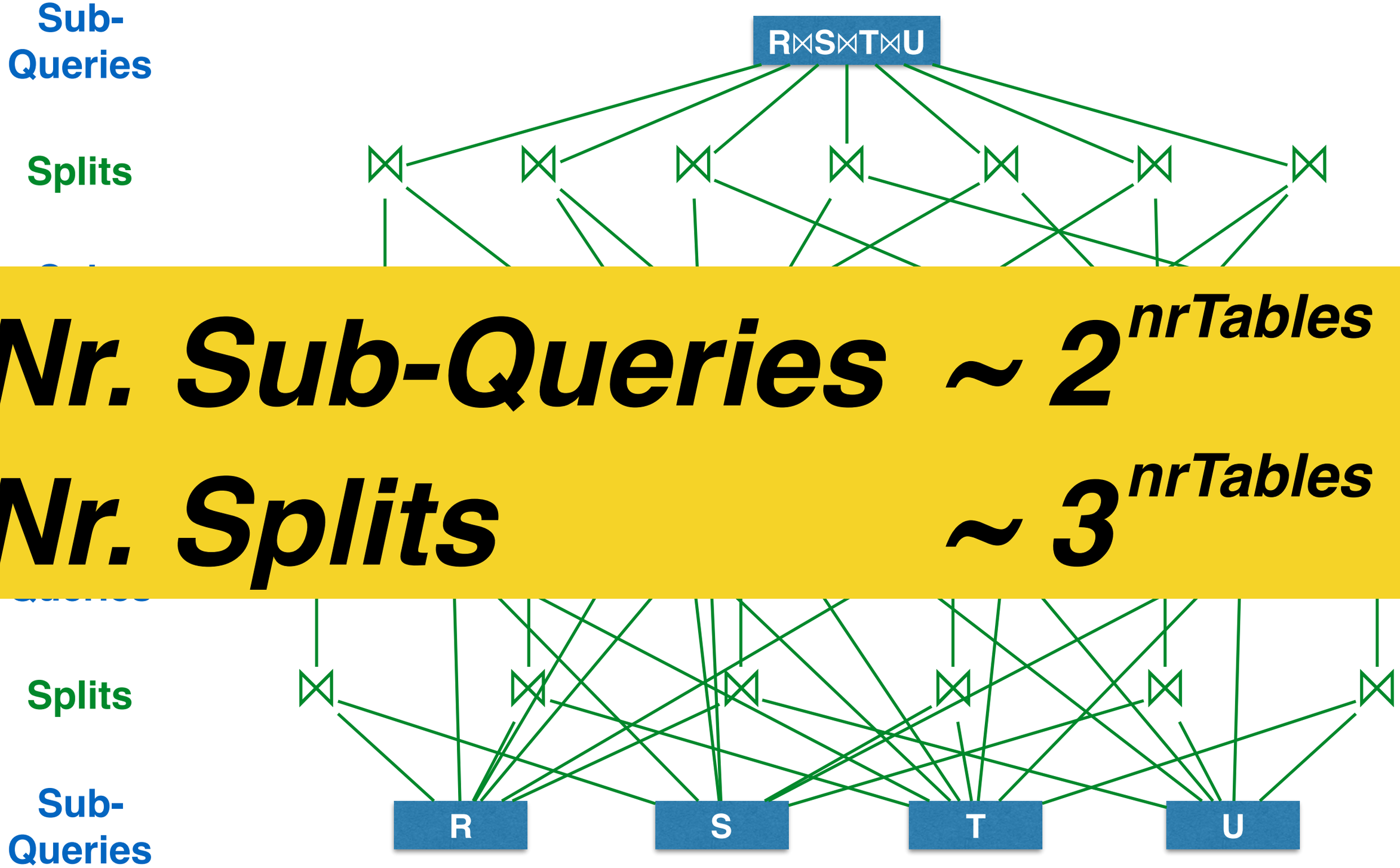
Sub-Queries

TUV	U,V	R,U,V	S,U,V	R,S,U,V
TUV	T,U	R,T,U	S,T,U	R,S,T,U
TUV		R,V	S,V	R,S,V
TUV		R,U	S,U	R,S,U
TUV		T,T	S,T	R,S,T
TUV				R,S

Complexity for Bushy Plans



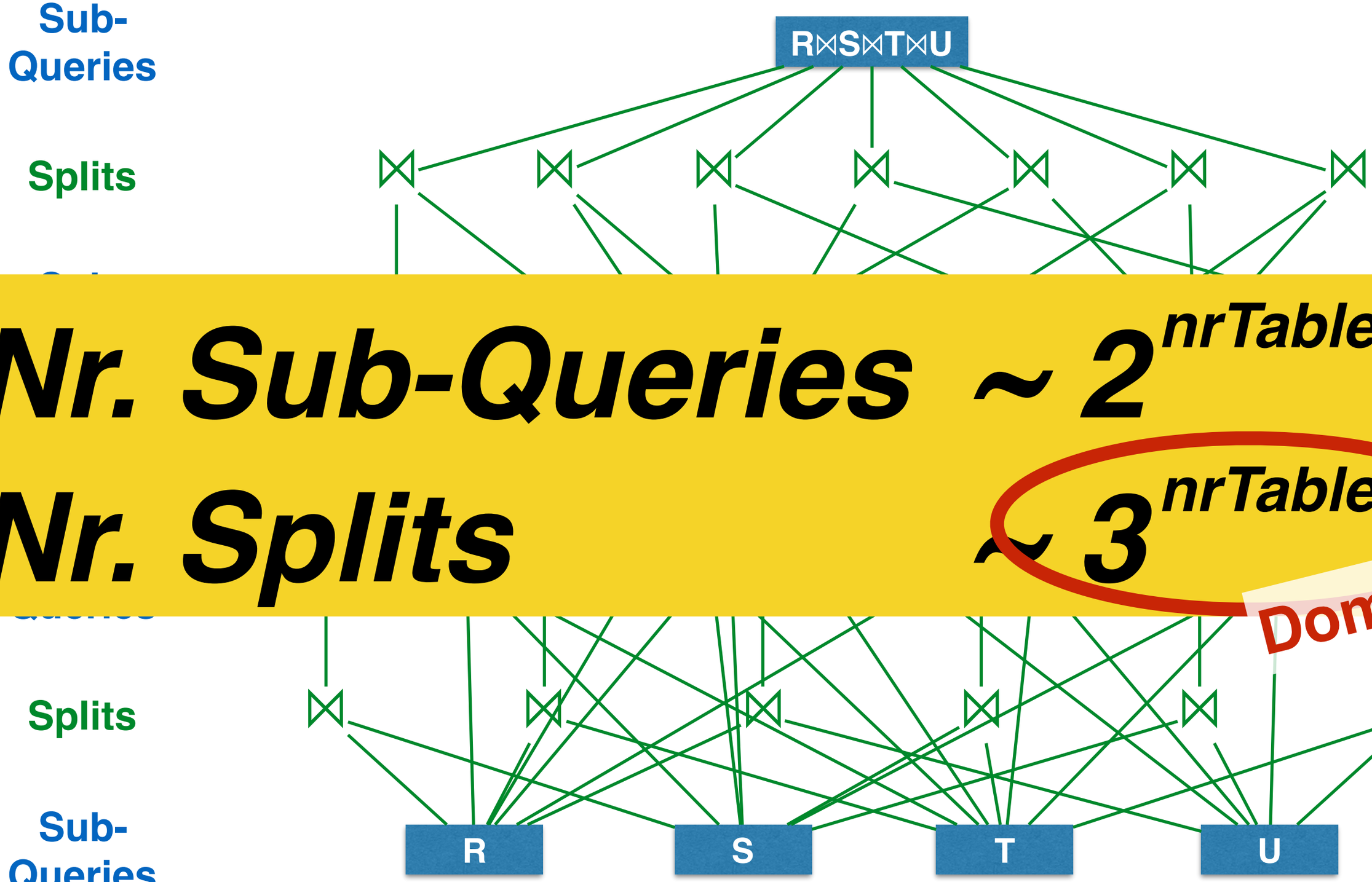
Complexity for Bushy Plans



Nr. Sub-Queries ~ $2^{nrTables}$

Nr. Splits ~ $3^{nrTables}$

Complexity for Bushy Plans

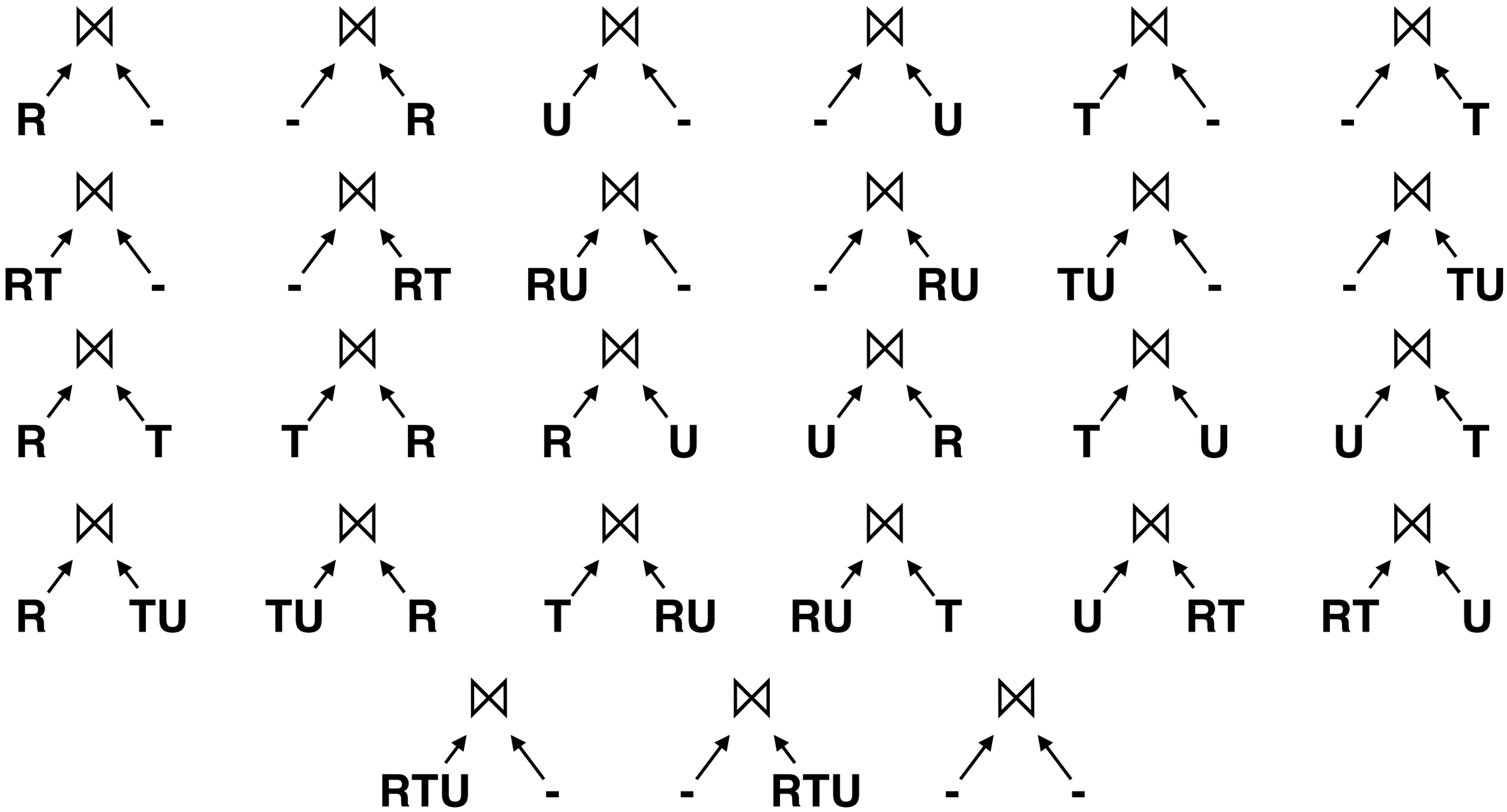


Nr. Sub-Queries ~ $2^{nrTables}$

Nr. Splits ~ $3^{nrTables}$

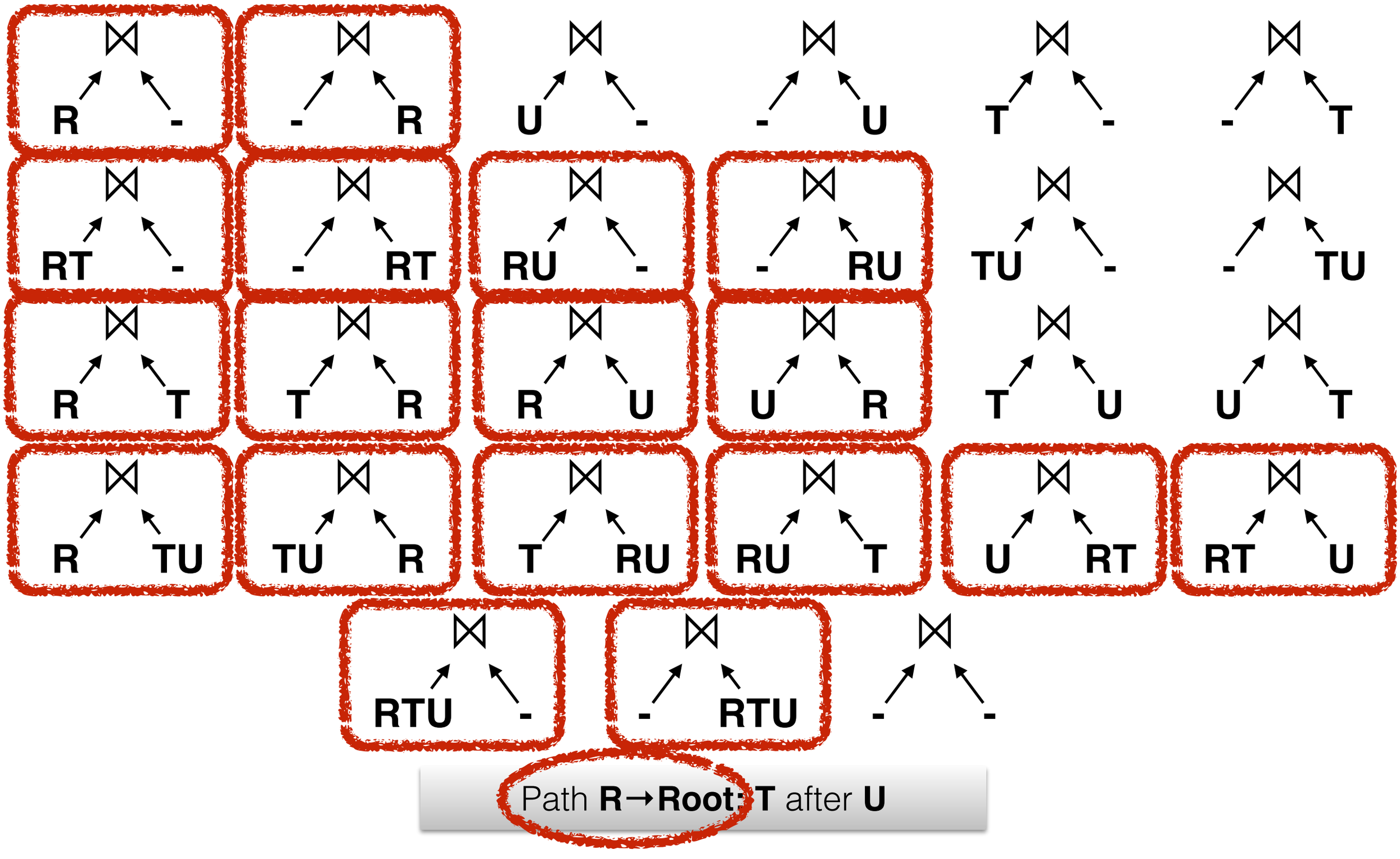
Dominant

Calculating Bushy Speedup

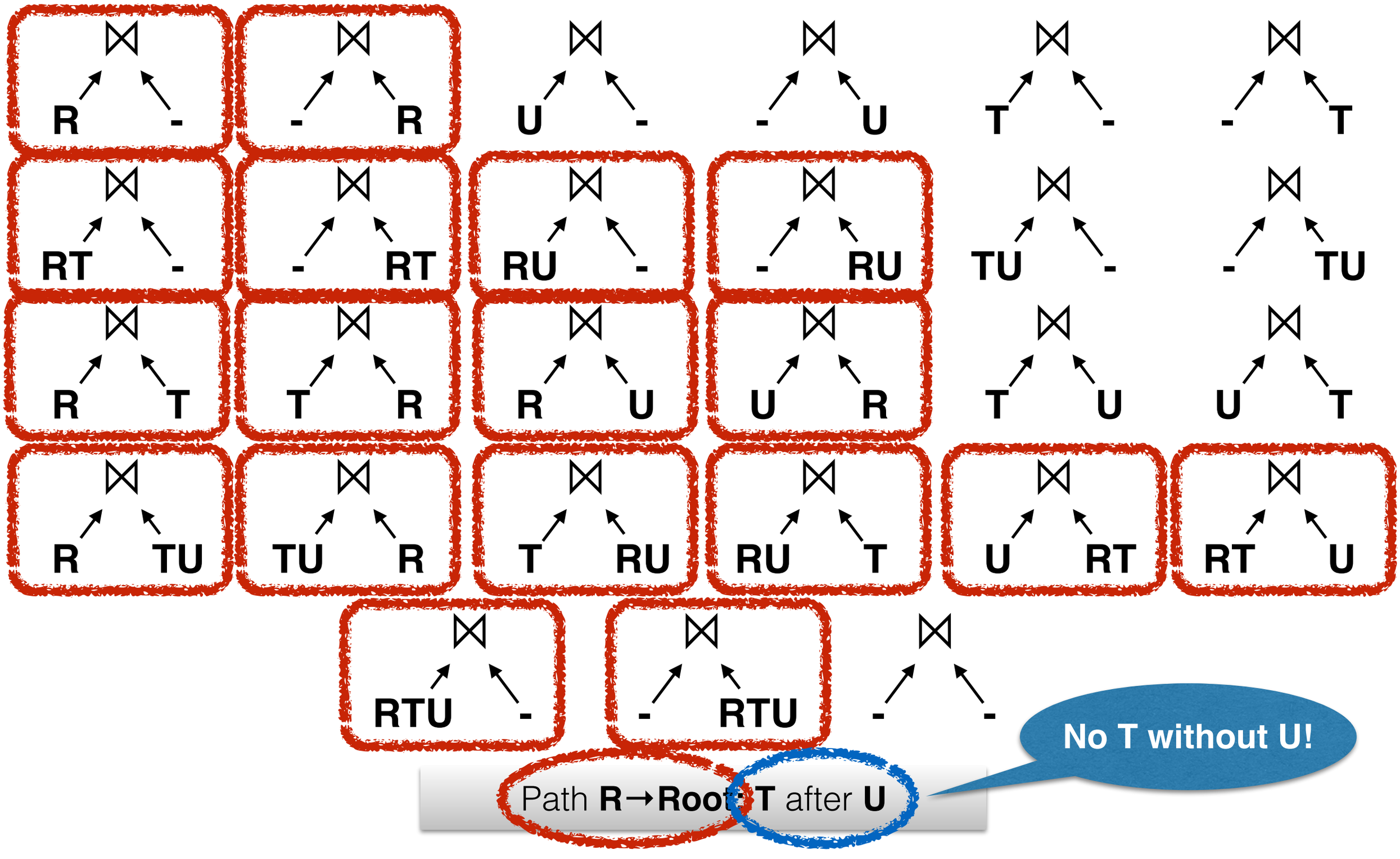


Path **R→Root: T** after **U**

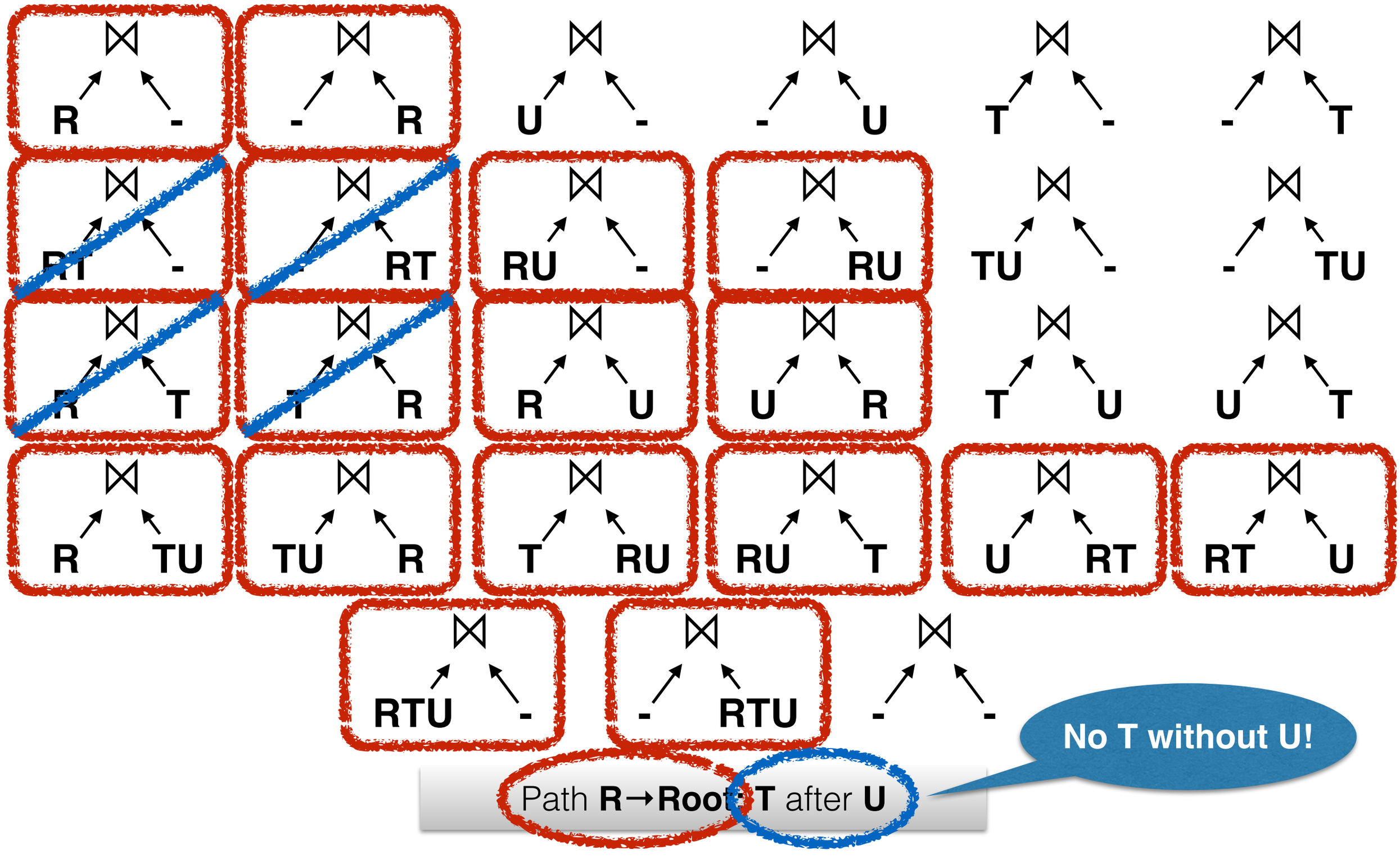
Calculating Bushy Speedup



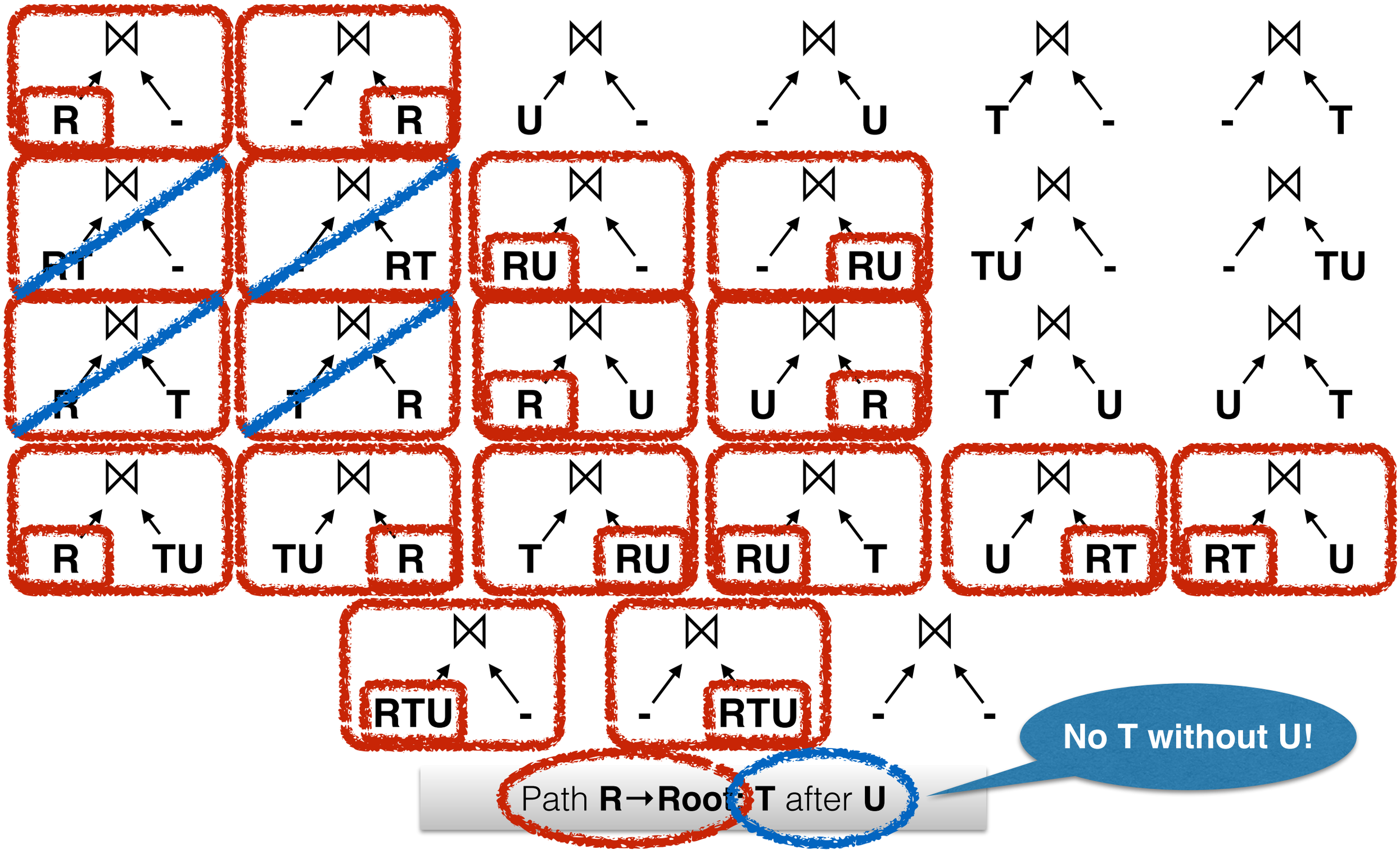
Calculating Bushy Speedup



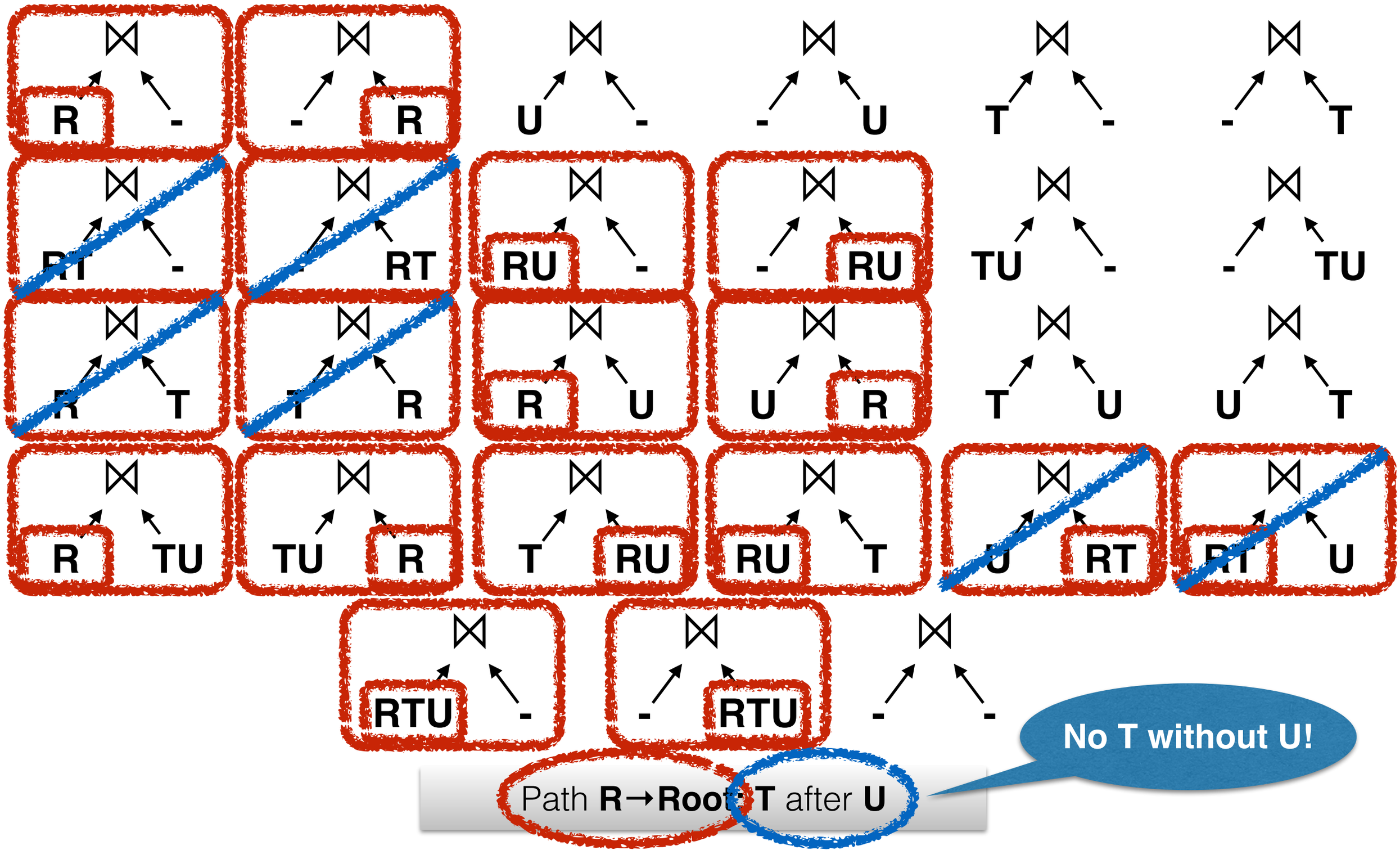
Calculating Bushy Speedup



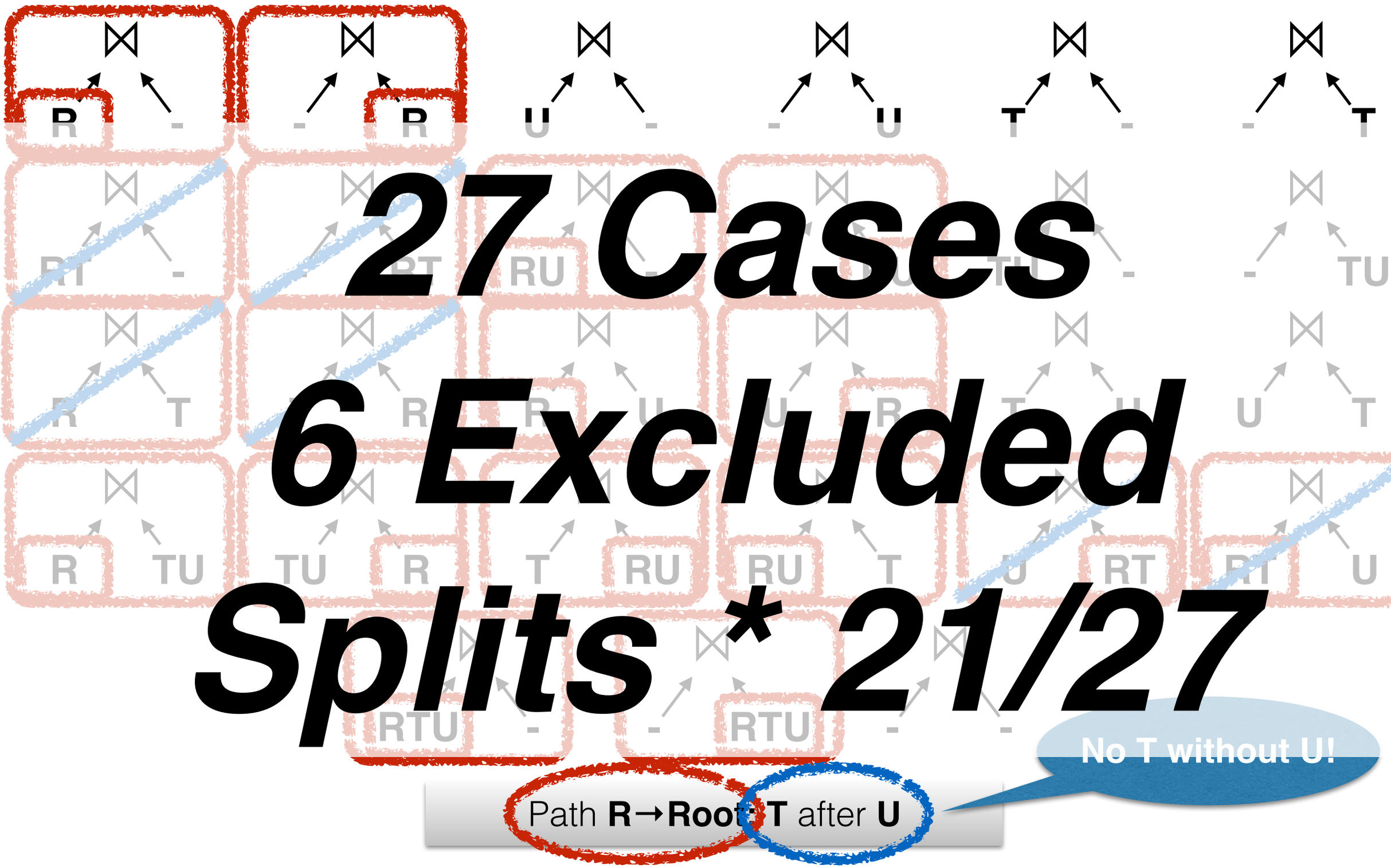
Calculating Bushy Speedup



Calculating Bushy Speedup



Calculating Bushy Speedup



27 Cases

6 Excluded Splits * 21/27

Path **R→Root**: **T** after **U**

No T without U!

Complexity Results

Results for one cost metric:

Metric \ Plan Space	Left-Deep	Bushy
Bytes Sent	$nrWorkers * (querySize + planSize)$	
Time - Master	$nrWorkers^2$	
Memory - Worker	$2^{querySize}*(3/4)^{nrWorkers}$	$2^{querySize}*(7/8)^{nrWorkers}$
Time - Worker	$2^{querySize}*(3/4)^{nrWorkers}*querySize$	$3^{querySize}*(21/27)^{nrWorkers}$

(Considering multiple metrics multiplies by polynomial factor)

Complexity Results

Results for one cost metric:

Metric \ Plan Space	Left-Deep	Bushy
Bytes Sent	$nrWorkers * (querySize + planSize)$	
Time - Master	$nrWorkers^2$	
Memory - Worker	$2^{querySize} * (3/4) * nrWorkers$	$2^{querySize} * (7/8) * nrWorkers$
Time - Worker	$2^{querySize} * (3/4) * nrWorkers * querySize$	$3^{querySize} * (21/27) * nrWorkers$

(Considering multiple metrics multiplies by polynomial factor)

Complexity Results

Results for one cost metric:

Metric \ Plan Space	Left-Deep	Bushy
Bytes Sent — Polynomial	$nrWorkers * (querySize + planSize)$	
Time - Master — Polynomial	$nrWorkers^2$	
Memory - Worker — Exponential	$2^{querySize} * (3/4) * nrWorkers$	$2^{querySize} * (7/8) * nrWorkers$
Time - Worker — Exponential	$2^{querySize} * (3/4) * nrWorkers * querySize$	$3^{querySize} * (21/27) * nrWorkers$

(Considering multiple metrics multiplies by polynomial factor)

Complexity Results

Results for one cost metric:

Metric \ Plan Space	Left-Deep	Bushy
Bytes Sent (Polynomial)	*nrWorkers * (querySize + planSize)*	
Time - Master (Polynomial)	*nrWorkers²* Optimal!	Optimal!
Memory - Worker (Exponential)	$2^{querySize} * (3/4)^{nrWorkers}$	$2^{querySize} * (7/8)^{nrWorkers}$
Time - Worker (Exponential)	$2^{querySize} * (3/4)^{nrWorkers} * querySize$	$3^{querySize} * (21/27)^{nrWorkers}$

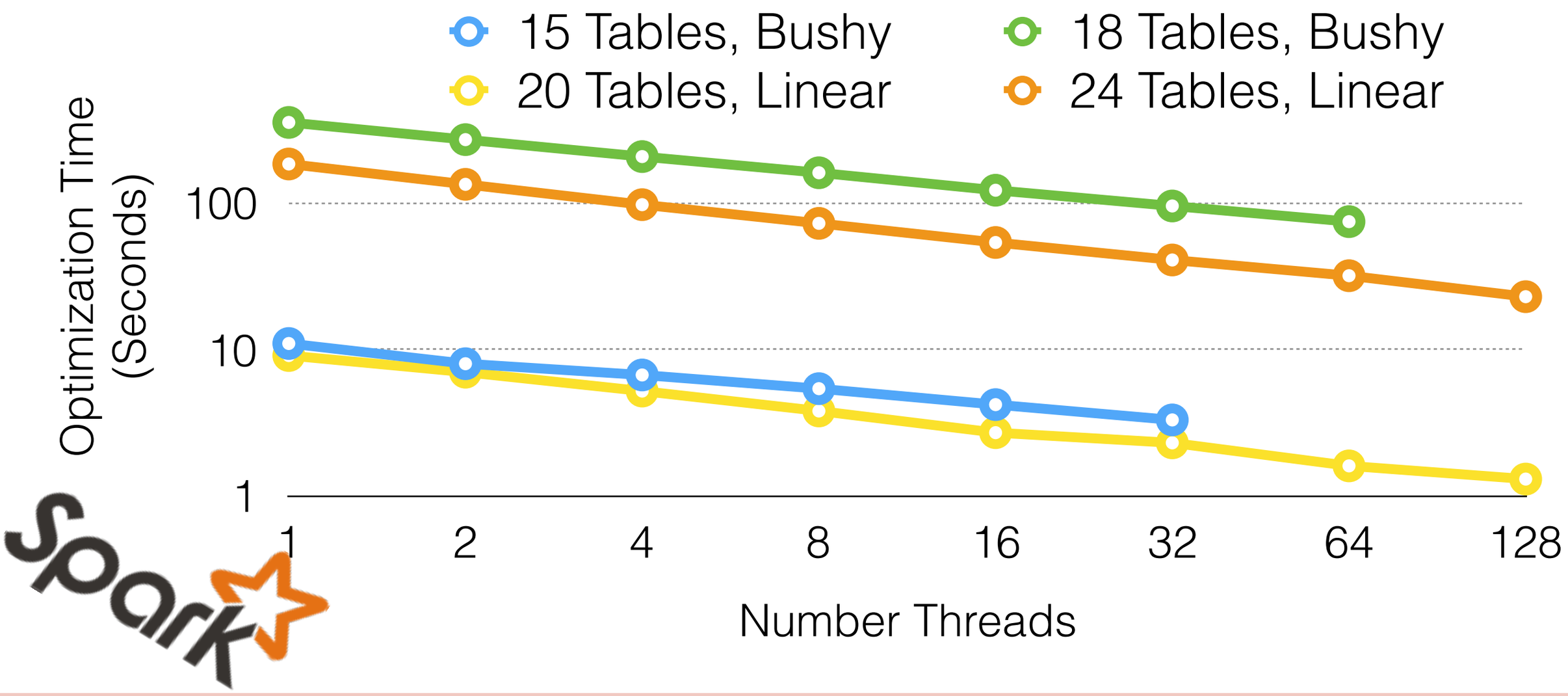
(Considering multiple metrics multiplies by polynomial factor)

Experimental Results

Experimental Setup

Randomly generated queries; one execution cost metric; approximated optimization

Spark 1.5 on Yarn 2.7; 100 nodes with 2x Intel Xeon 2.6 GHz, 128 GB memory



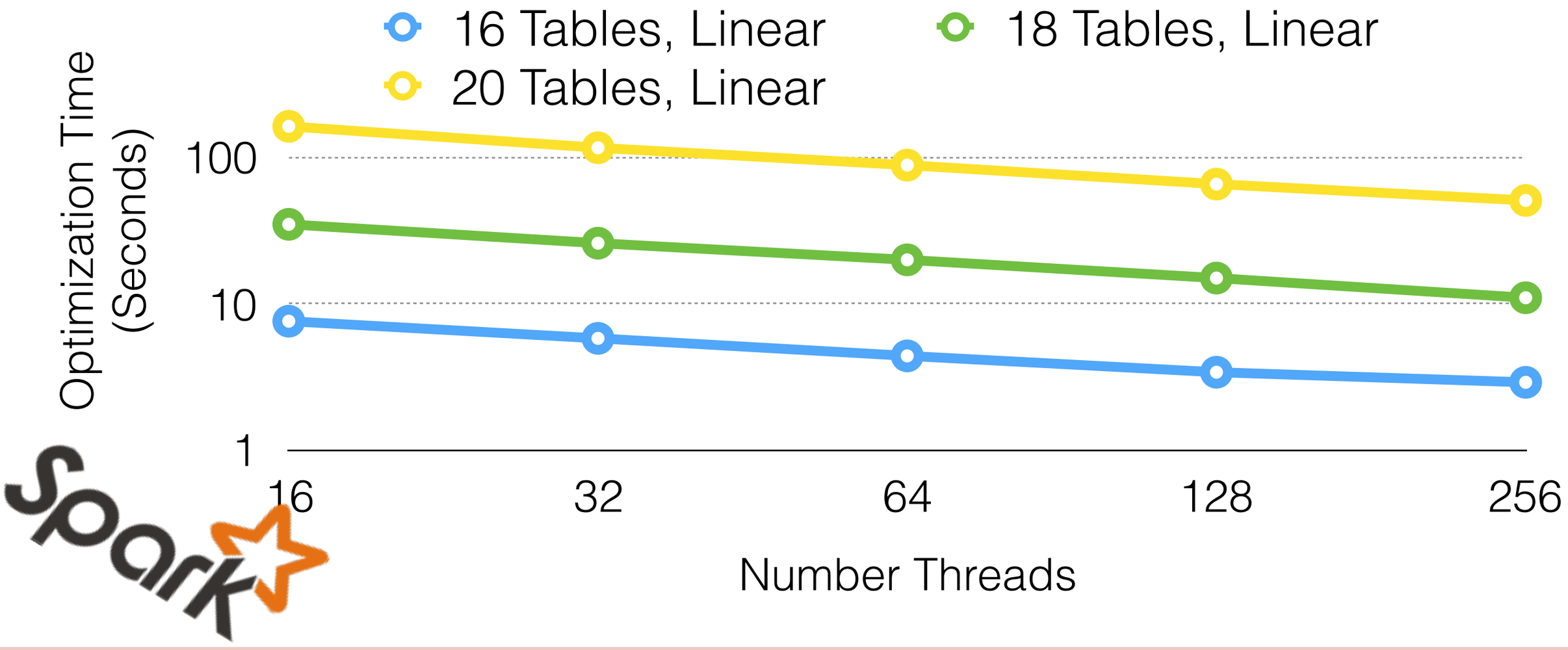
Scalability of new parallel query optimization algorithm

Experimental Results

Experimental Setup

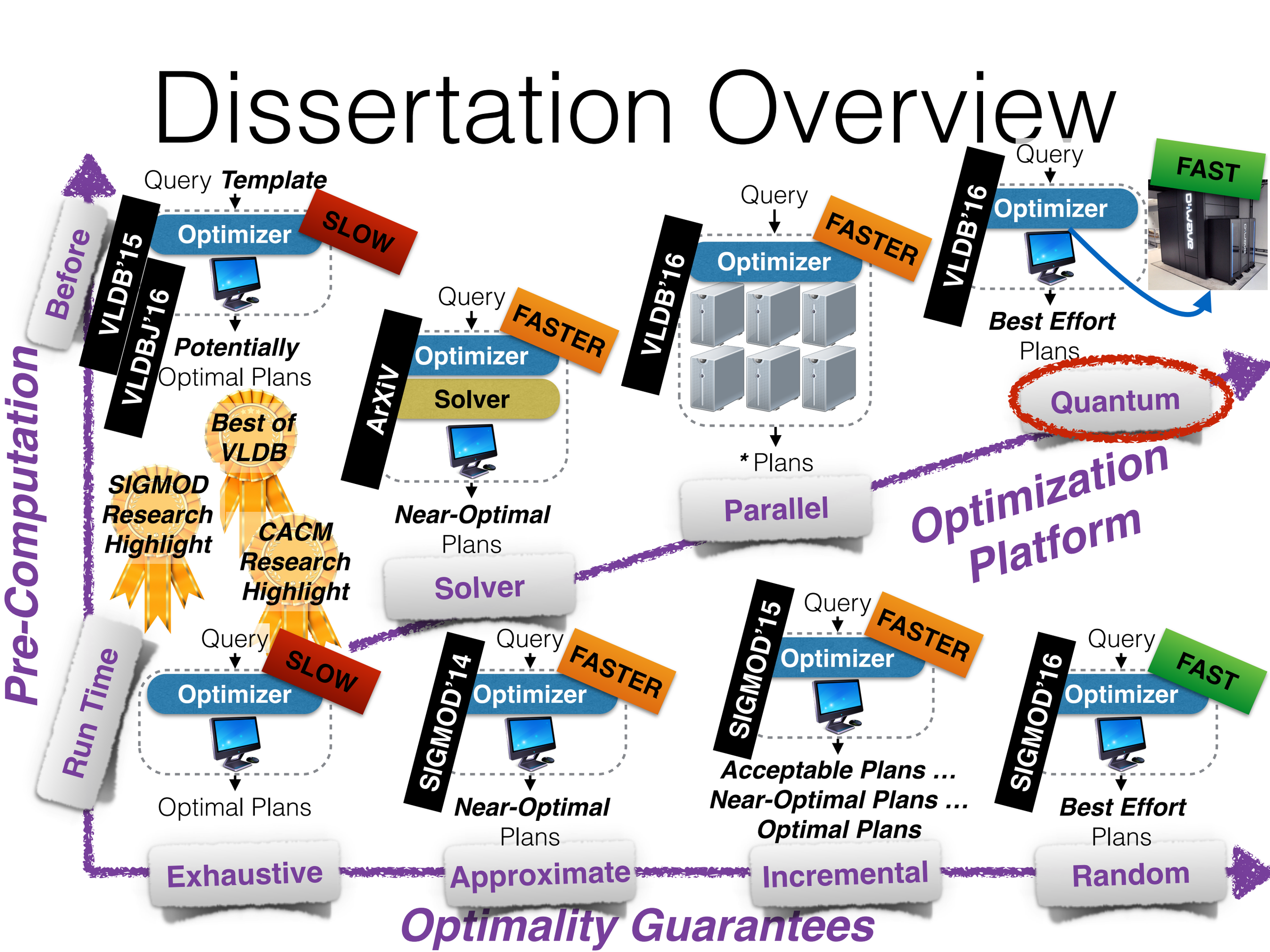
Randomly generated queries; two execution cost metrics; approximated optimization

Spark 1.5 on Yarn 2.7; 100 nodes with 2x Intel Xeon 2.6 GHz, 128 GB memory



Scalability of new parallel query optimization algorithm

Dissertation Overview



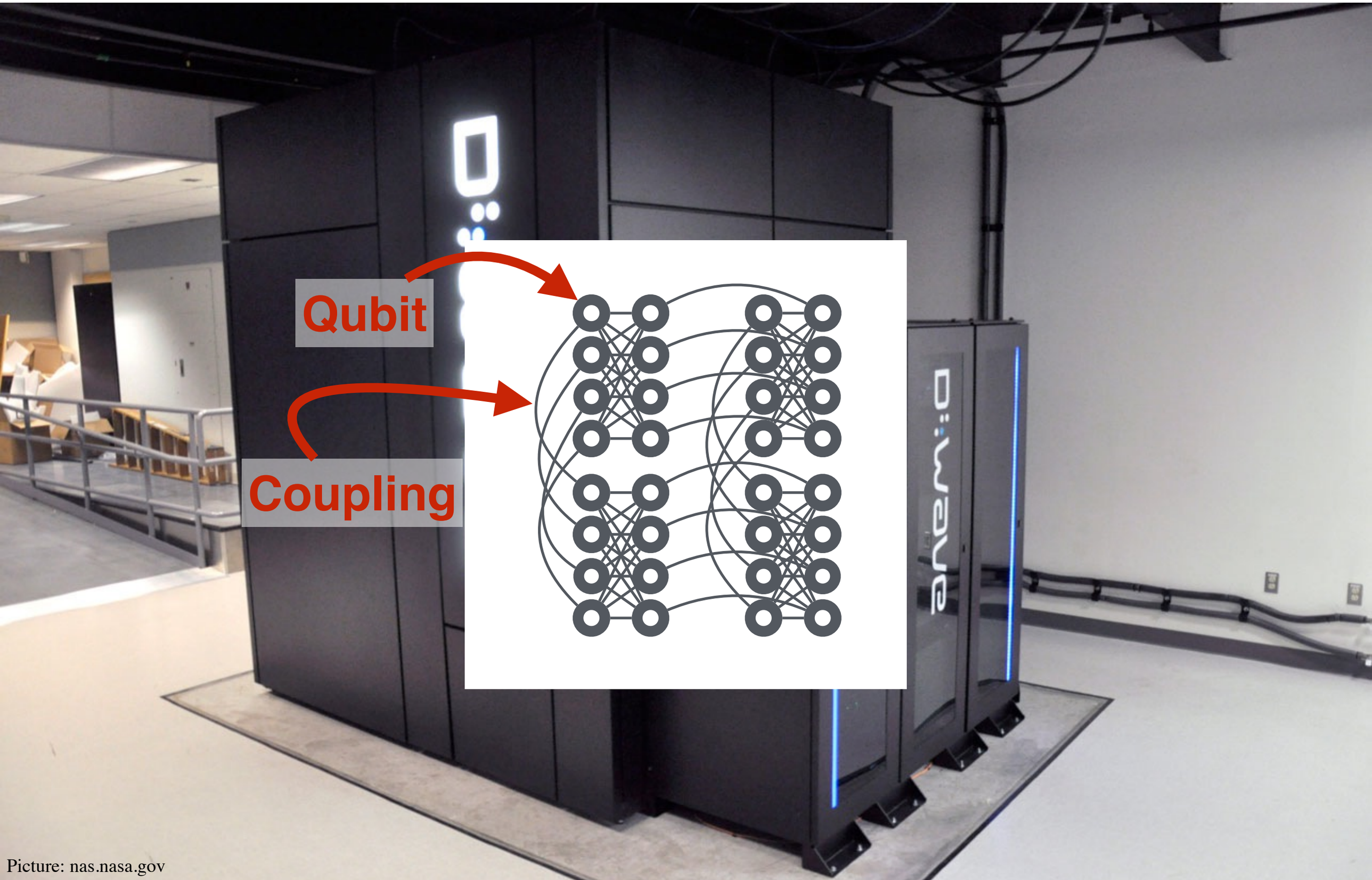
Quantum Computing

Classical Computer	**Quantum Computer**
Bits	Qubits
1 *or* 0	1 *and* 0

D-Wave Quantum Annealer

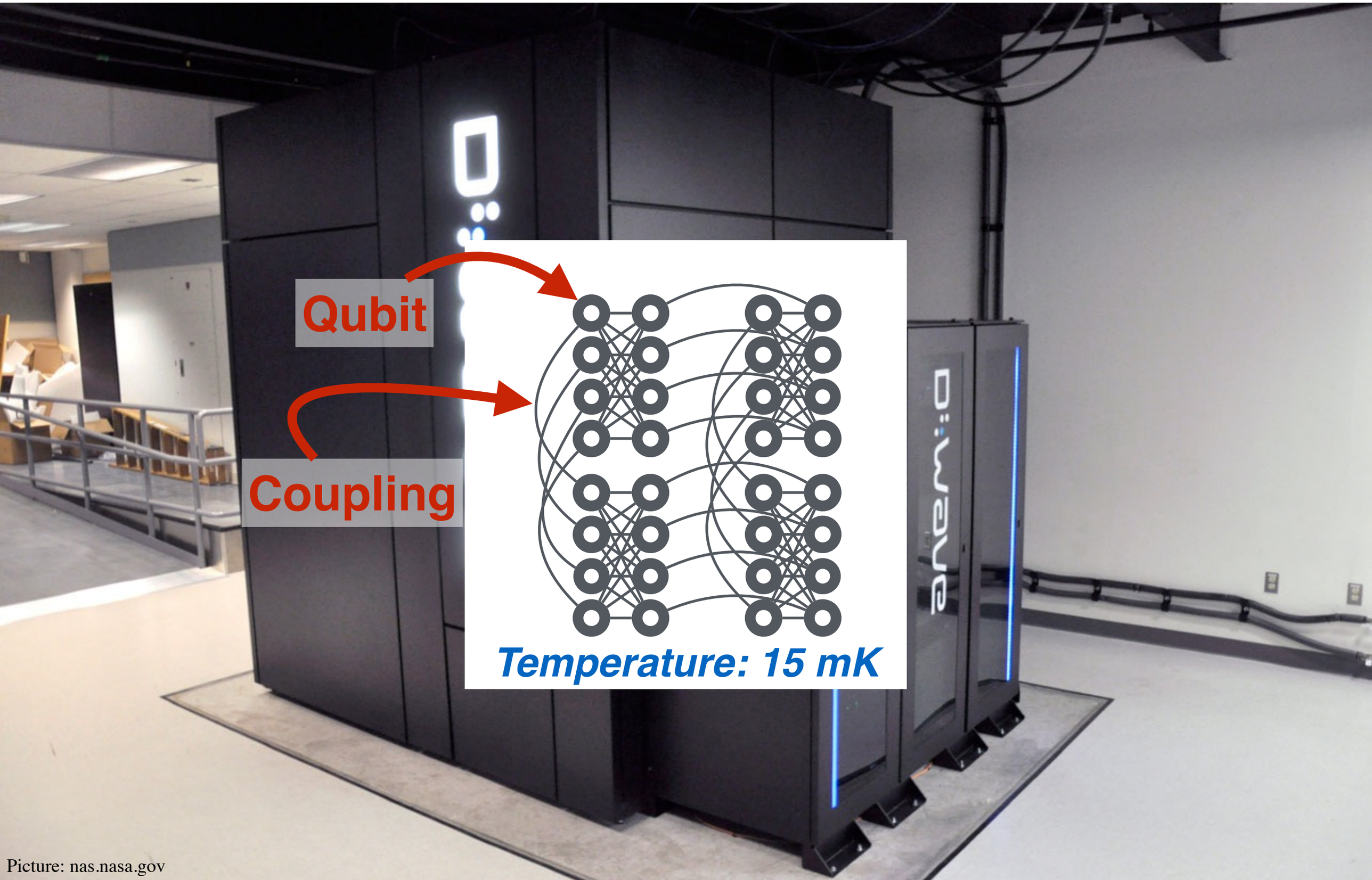
Picture: nas.nasa.gov

D-Wave Quantum Annealer



Picture: nas.nasa.gov

D-Wave Quantum Annealer



Picture: nas.nasa.gov

Comments on D-Wave

The general consensus now is that the D-Wave computers do exhibit some quantum behavior. *5.2014*

As far as I'm concerned, this completely nails down the case for computationally-relevant collective quantum tunneling in the D-Wave machine, at least within the 8-qubit clusters *12.2015*

Experiments on the D-Wave devices over the past years have indicated that the devices use quantum effects. *12.2015*

10,000,000 times faster than simulated annealing running on a single core. *12.2015*

John Martinis
Google/UCSB

Scott Aaronson
MIT

Matthias Troyer
ETH

The sweet spot for quantum annealers is thus quickly finding approximate solutions. *12.2015*

it remains uncertain whether D-Wave's approach will lead to speedups over the best known classical algorithms. *12.2015*

D-Wave has performed a careful study showing that their latest device is 15x faster than the best optimized classical codes on a single core of an Intel CPU (and thus still comparable to a high-end multi-core CPU) *12.2015*

Another challenge is the embedding of problems into the native hardware architecture *12.2015*

more efficient classical optimization algorithms exist for these problems, which outperform the current D-Wave 2X device for most problem instances. *12.2015*

if you wanted to solve a practical optimization problem, you'd first need to encode it into the Chimera graph, and that reduction entails a loss *12.2015*

Comments on D-Wave

The general consensus now is that the D-Wave computers do exhibit some quantum behavior. *5.2014*

As far as I'm concerned, this completely nails down the case for computationally-relevant collective quantum tunneling in the D-Wave machine, at least within the 8-qubit clusters *12.2015*

Experiments on the D-Wave devices over the past years have indicated that the devices use quantum effects. *12.2015*

John Martinis
Google/UCSB

10,000,000 times faster than simulated annealing running on a single core. *12.2015*

Scott Aaronson
MIT

Matthias Troyer
ETH

The sweet spot for quantum annealers is thus quickly finding approximate solutions. *12.2015*

it remains uncertain whether D-Wave's approach will lead to speedups over the best known classical algorithms. *12.2015*

Another challenge is the embedding of problems into the native hardware architecture *12.2015*

more efficient classical optimization algorithms exist for these problems, which outperform the current D-Wave 2X device for most problem instances. *12.2015*

D-Wave has performed a careful study showing that their latest device is 15x faster than the best optimized classical codes on a single core of an Intel CPU (and thus still comparable to a high-end multi-core CPU) *12.2015*

if you wanted to solve a practical optimization problem, you'd first need to encode it into the Chimera graph, and that reduction entails a loss *12.2015*

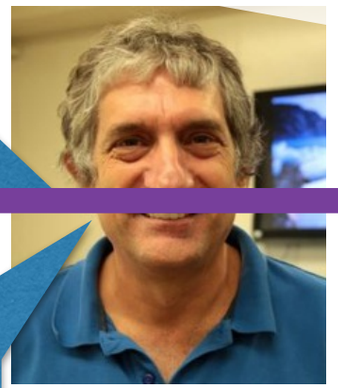
Comments on D-Wave

Quantum Effects

The general consensus now is that the D-Wave computers do exhibit some quantum behavior. *5.2014*

As far as I'm concerned, this completely nails down the case for computationally-relevant collective quantum tunneling in the D-Wave machine, at least within the 8-qubit cluster. *12.2015*

Experiments on the D-Wave devices over the past years have indicated that the devices use quantum effects. *12.2015*



John Martinis
Google/UCSB

10,000,000 times faster than simulated annealing running on a single core. *12.2015*

it remains uncertain whether D-Wave's approach will lead to speedups over the best known classical algorithms. *12.2015*

Another challenge is the embedding of problems into the native hardware architecture *12.2015*

more efficient classical optimization algorithms exist for these problems, which outperform the current D-Wave 2X device for most problem instances. *12.2015*

Scott Aaronson
MIT

Matthias Troyer
ETH

The sweet spot for quantum annealers is thus quickly finding approximate solutions. *12.2015*

D-Wave has performed a careful study showing that their latest device is 15x faster than the best optimized classical codes on a single core of an Intel CPU (and thus still comparable to a high-end multi-core CPU) *12.2015*

if you wanted to solve a practical optimization problem, you'd first need to encode it into the Chimera graph, and that reduction entails a loss *12.2015*

Comments on D-Wave

Quantum Effects

The general consensus now is that the D-Wave computers do exhibit some quantum behavior. *5.2014*

As far as I'm concerned, this completely nails down the case for computationally-relevant collective quantum tunneling in the D-Wave machine, at least within the 8-qubit cluster. *12.2015*

Experiments on the D-Wave devices over the past years have indicated that the devices use quantum effects. *12.2015*

10,000,000 times faster than simulated annealing running on a single core. *12.2015*

John Martinis (Google/UCSB)

Scott Aaronson
MIT

Matthias Troyer
ETH

The sweet spot for quantum annealers is thus quickly finding approximate solutions. *12.2015*

it remains uncertain whether D-Wave's approach will lead to speedups over the best known classical algorithms. *12.2015*

Another challenge is the embedding of problems into the native hardware architecture *12.2015*

D-Wave has performed a careful study showing that their latest device is 15x faster than the best optimized classical code on a single core of an Intel CPU (and thus still comparable to a high-end multi-core CPU) *12.2015*

many efficient classical optimization algorithms exist for these problems, which outperform the current D-Wave 2X device for most problem instances. *12.2015*

if you wanted to solve a practical optimization problem, you'd first need to encode it into the Chimera graph, and that reduction entails a loss *12.2015*

Often Beaten at Finding Optimum

Faster than Single-Core at Approximation

Comments on D-Wave

The general consensus now is that the D-Wave computers do exhibit some quantum behavior. *5.2014*

As far as I'm concerned, this completely nails down the case for computationally-relevant collective quantum tunneling in the D-Wave machine, at least within the 8-qubit cluster. *12.2015*

Experiments on the D-Wave devices over the past years have indicated that the devices use quantum effects. *12.2015*

10,000,000 times faster than simulated annealing running on a single core. *12.2015*

Matthias Troyer
ETH

The sweet spot for quantum annealers thus quickly [illegible] approximate optimizations. *12.2015*

it remains uncertain whether D-Wave's approach will lead to speedups over the best known classical algorithm *12.2015*

D-Wave has performed careful study showing that their latest device is 15x faster than the best optimized classical algorithm on a single core of an Intel CPU (and thus still comparable to a high-end multi-core CPU) *12.2015*

Another [illegible] embedding of problems into the D-Wave hardware architecture *12.2015*

many efficient classical optimization algorithms exist for these problems, which outperform the current D-Wave 2X device for most problem instances. *12.2015*

if you wanted to solve a practical optimization problem, you'd first need to encode it into the Chimera graph, and that reduction entails a loss *12.2015*

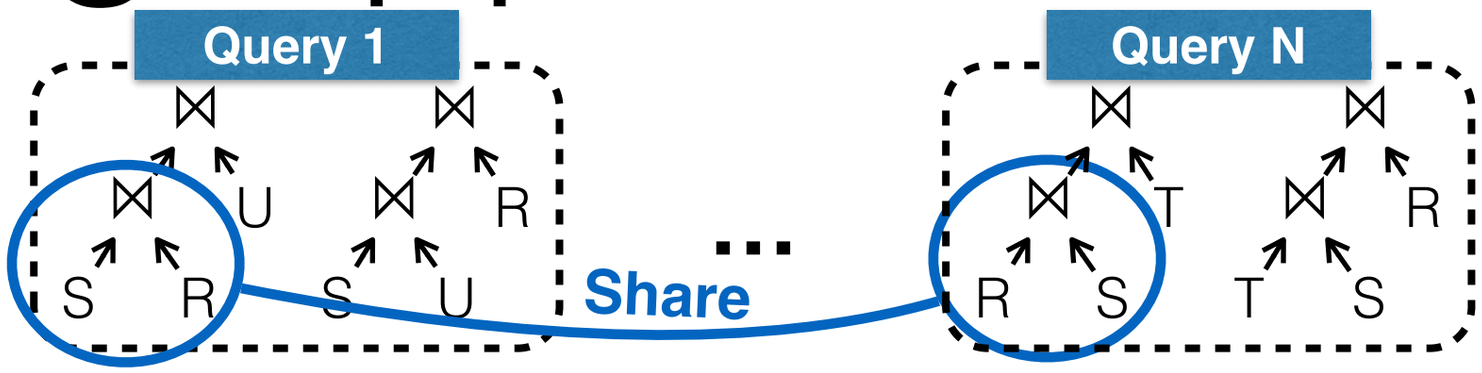
Quantum Effects Often Beaten at Finding Optimum

How About Relevant Problems??

Faster Than Simulated Annealing at Approximation

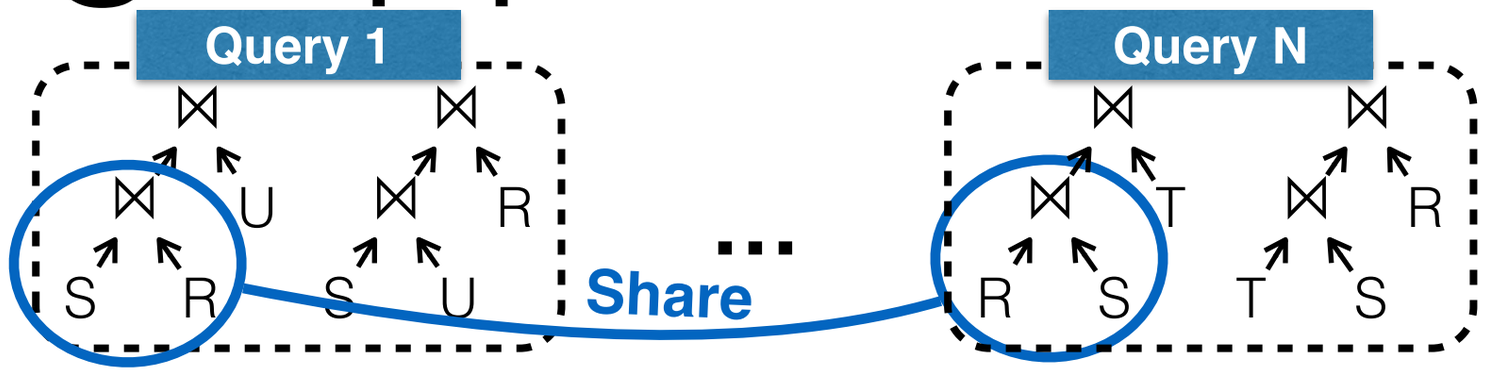
Mapping Approach

Find Optimal Plan Combination



Mapping Approach

Find Optimal Plan Combination



1 *Transform → Quadratic Formula*

Binary Variables:

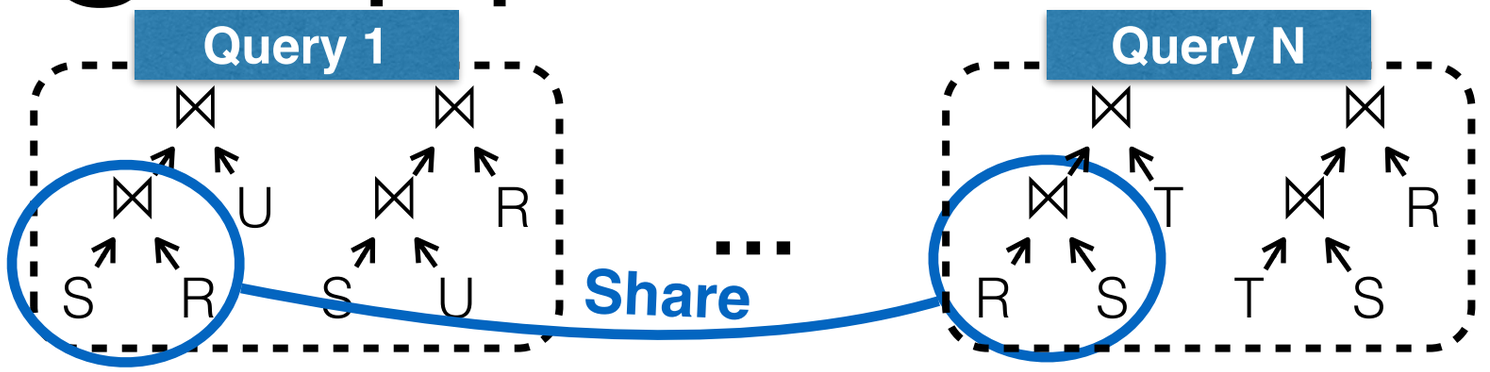
selectPlan2 … *selectPlan1*

Cost Formula:

costTerms *savingsTerms* *constraintTerms*

Mapping Approach

Find Optimal Plan Combination



1 ***Transform → Quadratic Formula***

Binary Variables:

... selectPlan1

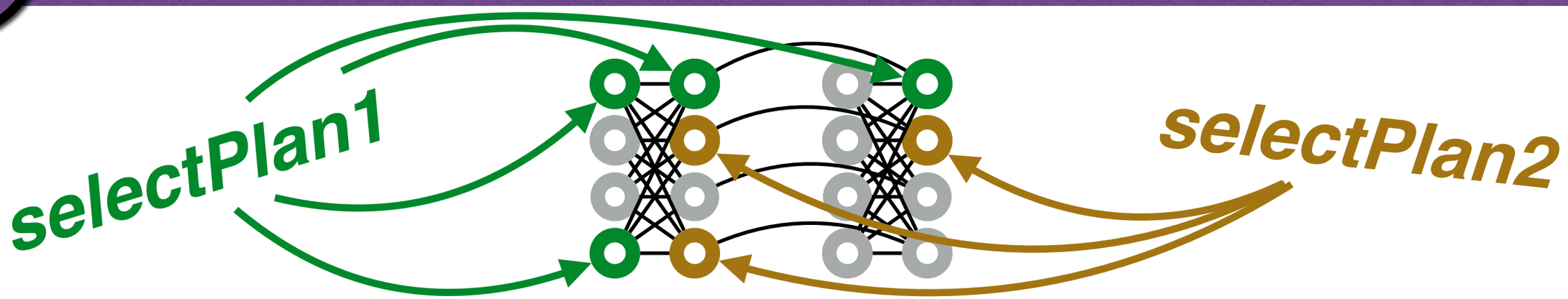
selectPlan2

Cost Formula:

costTerms savingsTerms

constraintTerms

2 ***Transform → Weights on Qubits***

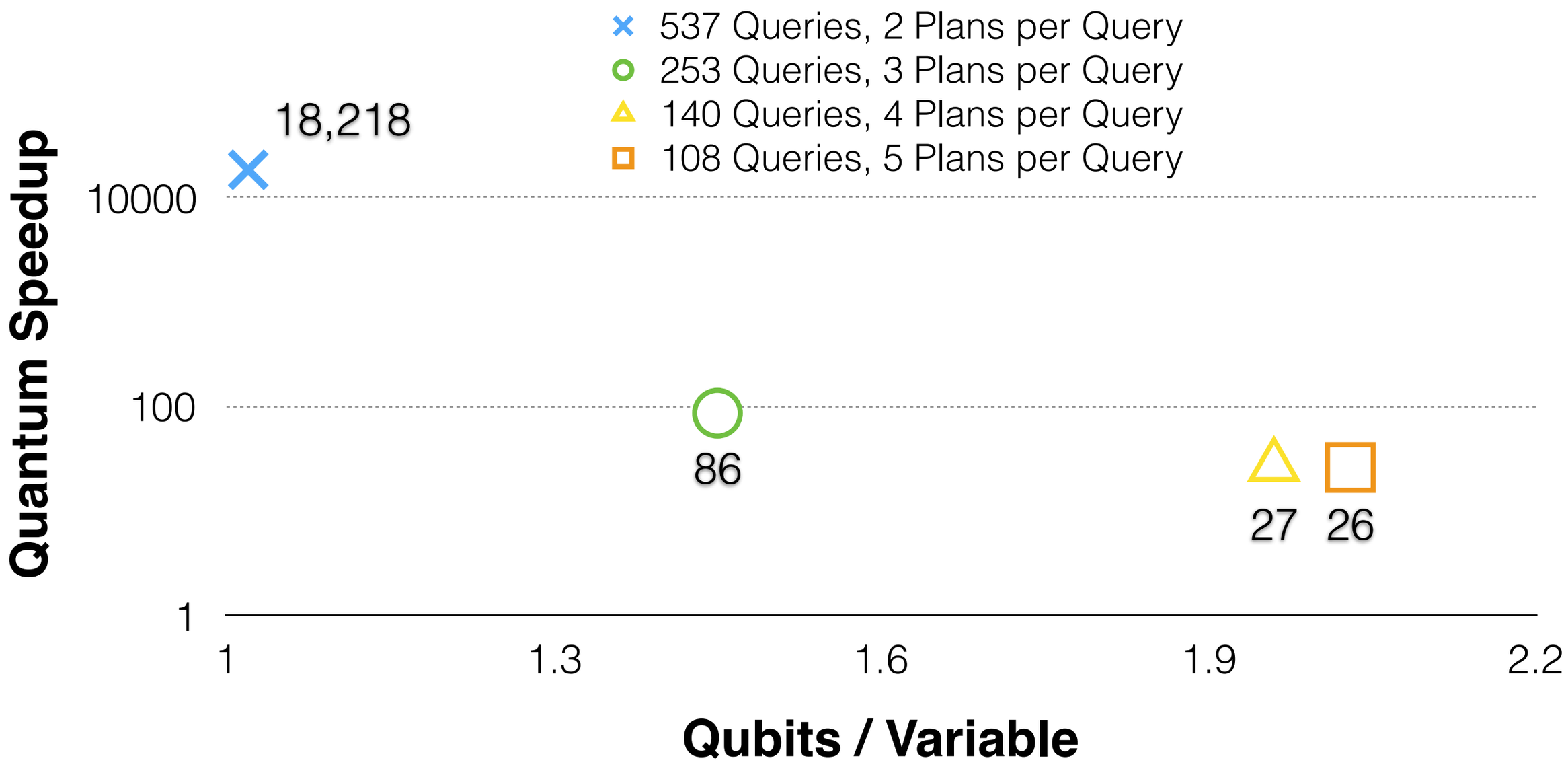


Experimental Results

Experimental Setup

Compared D-Wave 2X vs. Integer Programming, Genetic Algorithms, Heuristics

Compared approximation time on randomly generated problem instances



More Details

Multiple Query Optimization on the D-Wave 2X Adiabatic Quantum Computer

VLDB '16

Immanuel Trummer and Christoph Koch
{firstname}.{lastname}@epfl.ch
École Polytechnique Fédérale de Lausanne

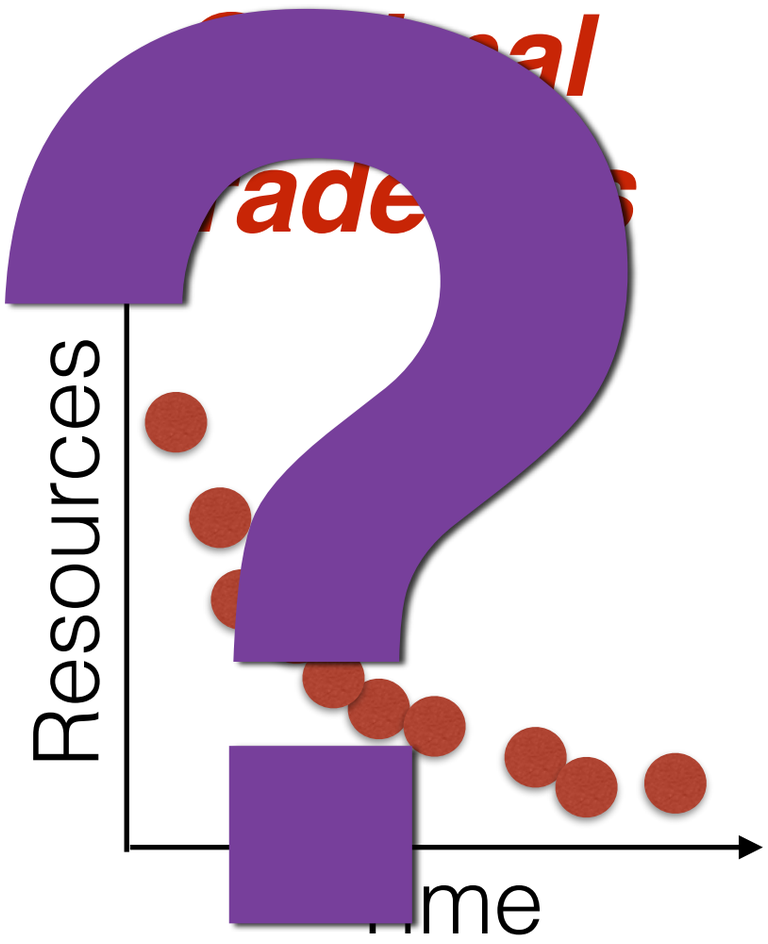
ABSTRACT

The D-Wave adiabatic quantum annealer solves hard combinatorial optimization problems leveraging quantum physics. The newest version features over 1000 qubits and was released in August 2015. We were given access to such a machine, currently hosted at NASA Ames Research Center in California, to explore the potential for hard optimization problems that arise in the context of databases.

In this paper, we tackle the problem of multiple query optimization (MQO). We show how an MQO problem instance can be transformed into a mathematical formula that in California. This device is claimed to exploit the laws of quantum physics [6] in the hope to solve NP-hard optimization problems faster than traditional approaches. The machine supports a very restrictive class of optimization problems while it is for instance not capable of running Shor's algorithm [40] for factoring large numbers[1]. We will show how instances of the multiple query optimization problem can be brought into a representation that is suitable as input to the quantum annealer. We also report results of an experimental evaluation that compares the time it takes to solve MQO problems on the quantum annealer to the time

Example

Optimization Time:



Mining Subjective Associations

Mining Subjective Associations

Mining Subjective Associations

I find kittens cute

Kittens-Cute

kittens are cute

Palo Alto is not big

Palo Alto is a big city

Palo Alto-Big

Palo Alto is a small city

I don't find jogging boring

Jogging-Boring

Jogging is so boring…

1 Extract Opinion Statements

Mining Subjective Associations

Cute Animals

Majority Opinion → *Pr(…)=0.9* → User Opinion → *Pr(…)=0.8* → User Post

Big Cities

Majority Opinion → *Pr(…)=0.85* → User Opinion → *Pr(…)=0.98* → User Post

Boring Sports

Majority Opinion → *Pr(…)=0.8* → User Opinion → *Pr(…)=0.95* → User Post

2 ***Learn User Behavior***

I find kittens cute

Kittens-Cute

kittens are cute

Palo Alto is not big

Palo Alto is a big city

Palo Alto-Big

Palo Alto is a small city

I don't find jogging boring

Jogging-Boring

Jogging is so boring…

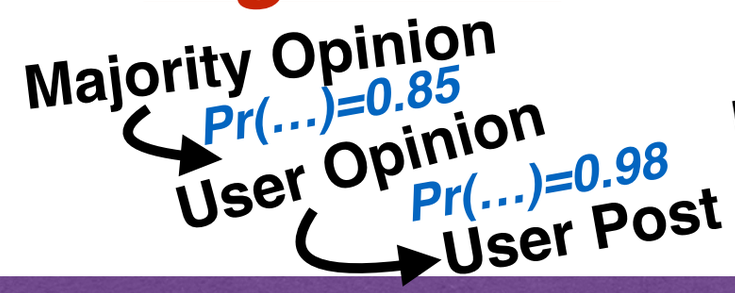
1 ***Extract Opinion Statements***

Mining Subjective Associations

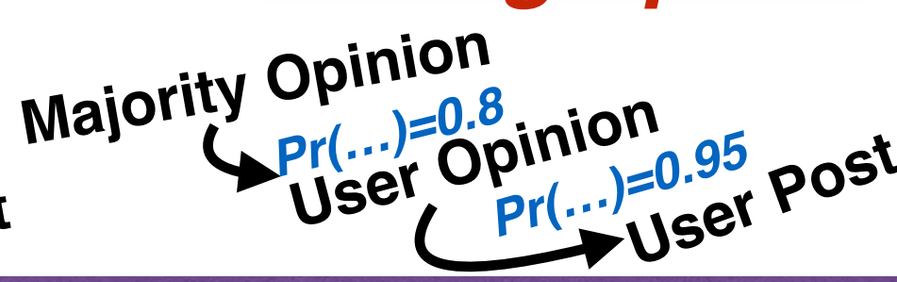
3 Infer Majority Opinion

Cute Animals

Big Cities



Boring Sports



2 Learn User Behavior

I find kittens cute

Kittens-Cute

kittens are cute

Palo Alto is not big

Palo Alto is a big city

Palo Alto-Big

Palo Alto is a small city

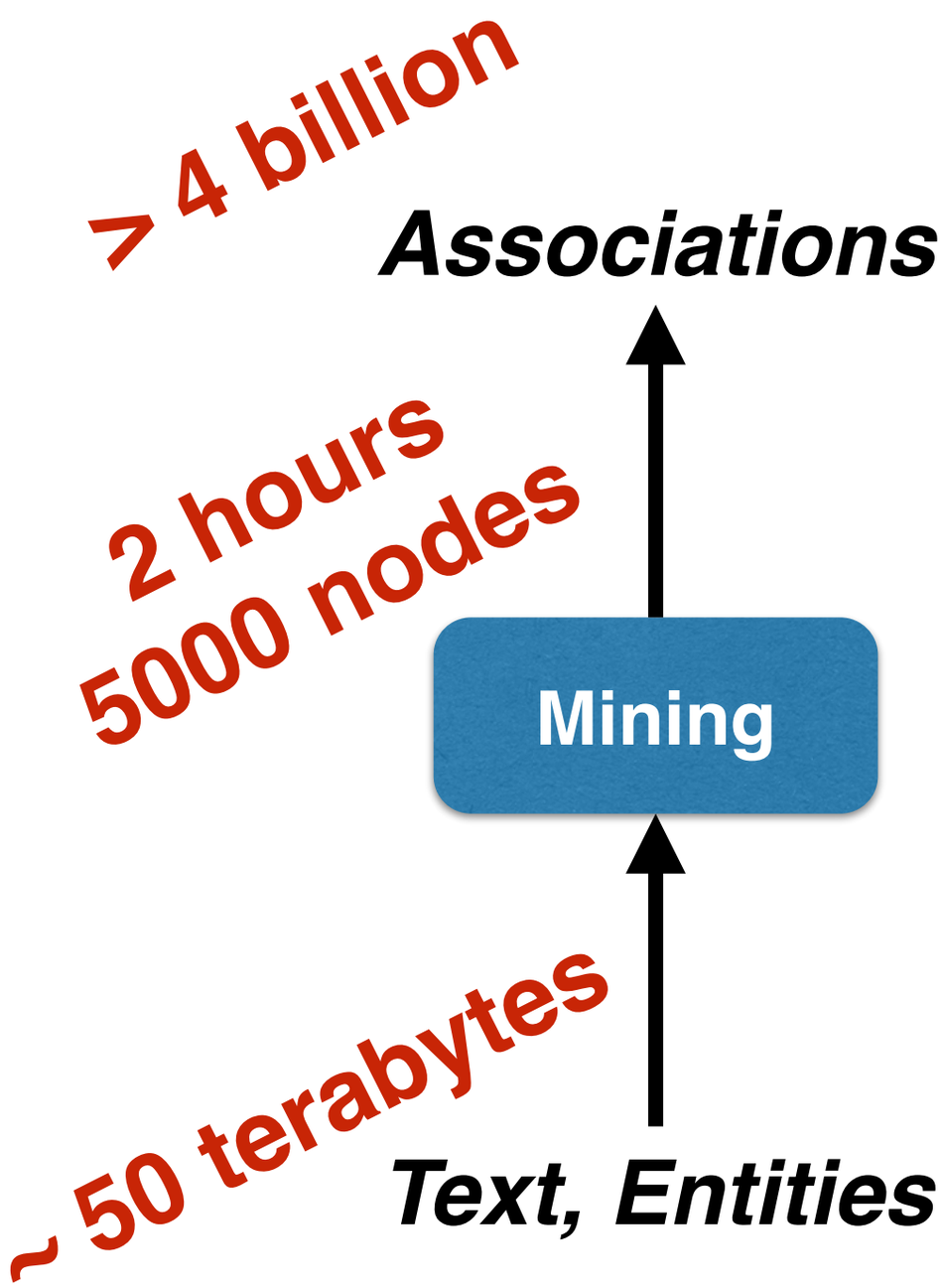
I don't find jogging boring

Jogging-Boring

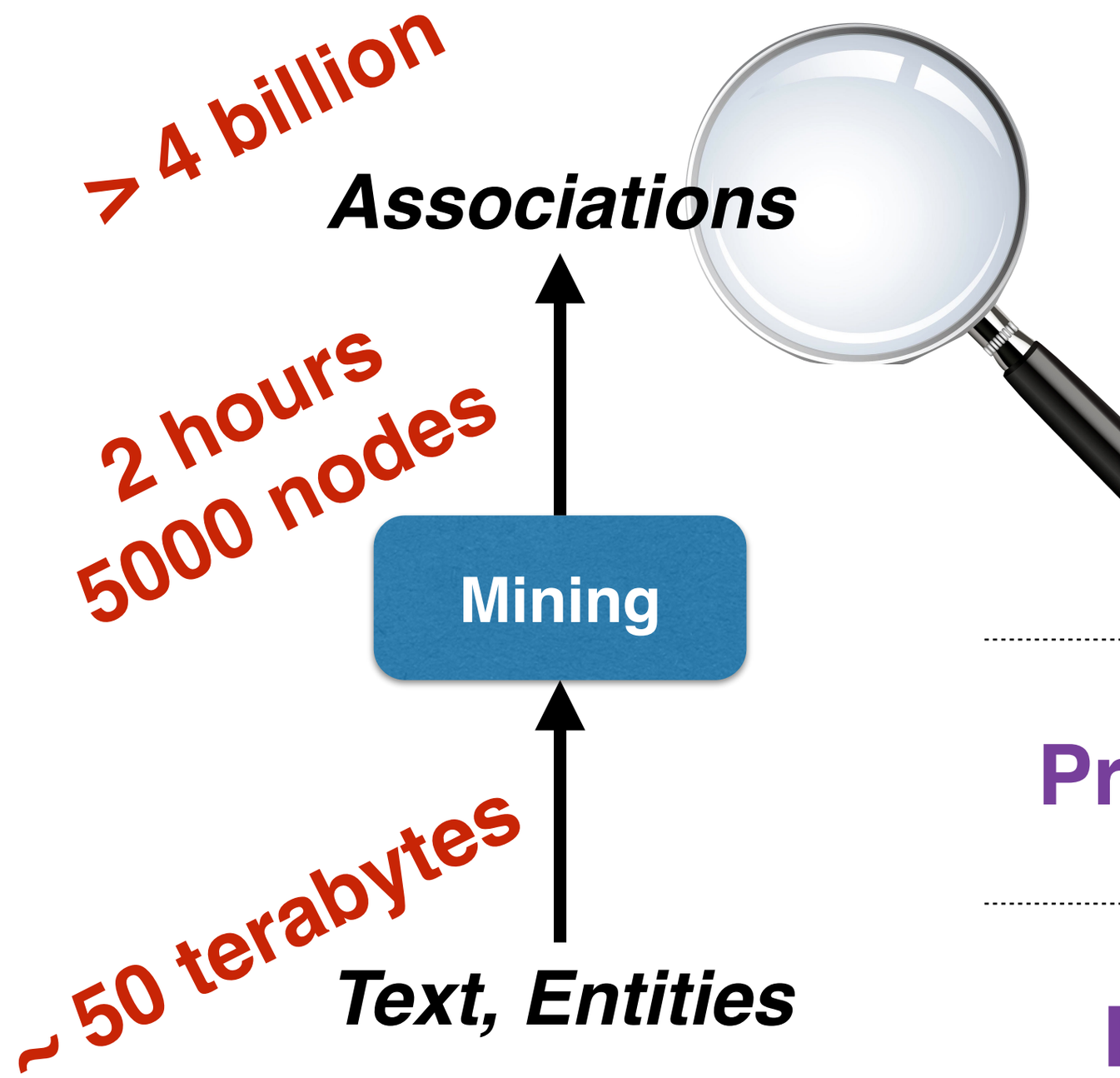
Jogging is so boring…

1 Extract Opinion Statements

Statistics and Results



Statistics and Results



	Prior	This
Precision	54%	**77%**
Recall	48%	**97%**

More Details

SIGMOD '15

Mining Subjective Properties on the Web

Immanuel Trummer*
EPFL
Lausanne, Switzerland
immanuel.trummer@epfl.ch

Alon Halevy
Google, Inc.
Mountain View, USA
halevy@google.com

Hongrae Lee
Google, Inc.
Mountain View, USA
hrlee@google.com

Sunita Sarawagi
Google, Inc. and IIT Bombay
Mountain View, USA/Bombay,
India
sarawagi@google.com

Rahul Gupta
Google, Inc.
Mountain View, USA
grahul@google.com

ABSTRACT

Even with the recent developments in Web search of answering queries from structured data, search engines are still limited to queries with an objective answer, such as EUROPEAN CAPITALS or WOODY ALLEN MOVIES. However, many queries are subjective, such as SAFE CITIES, or CUTE ANIMALS. The underlying knowledge bases of search engines do not contain answers to these queries because they do not have a ground truth. We describe the SURVEYOR system that mines the dominant opinion held by authors of Web content about whether a subjective property applies to a given entity. The evidence on which SURVEYOR relies are statements extracted from Web text that either support the property or claim its

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

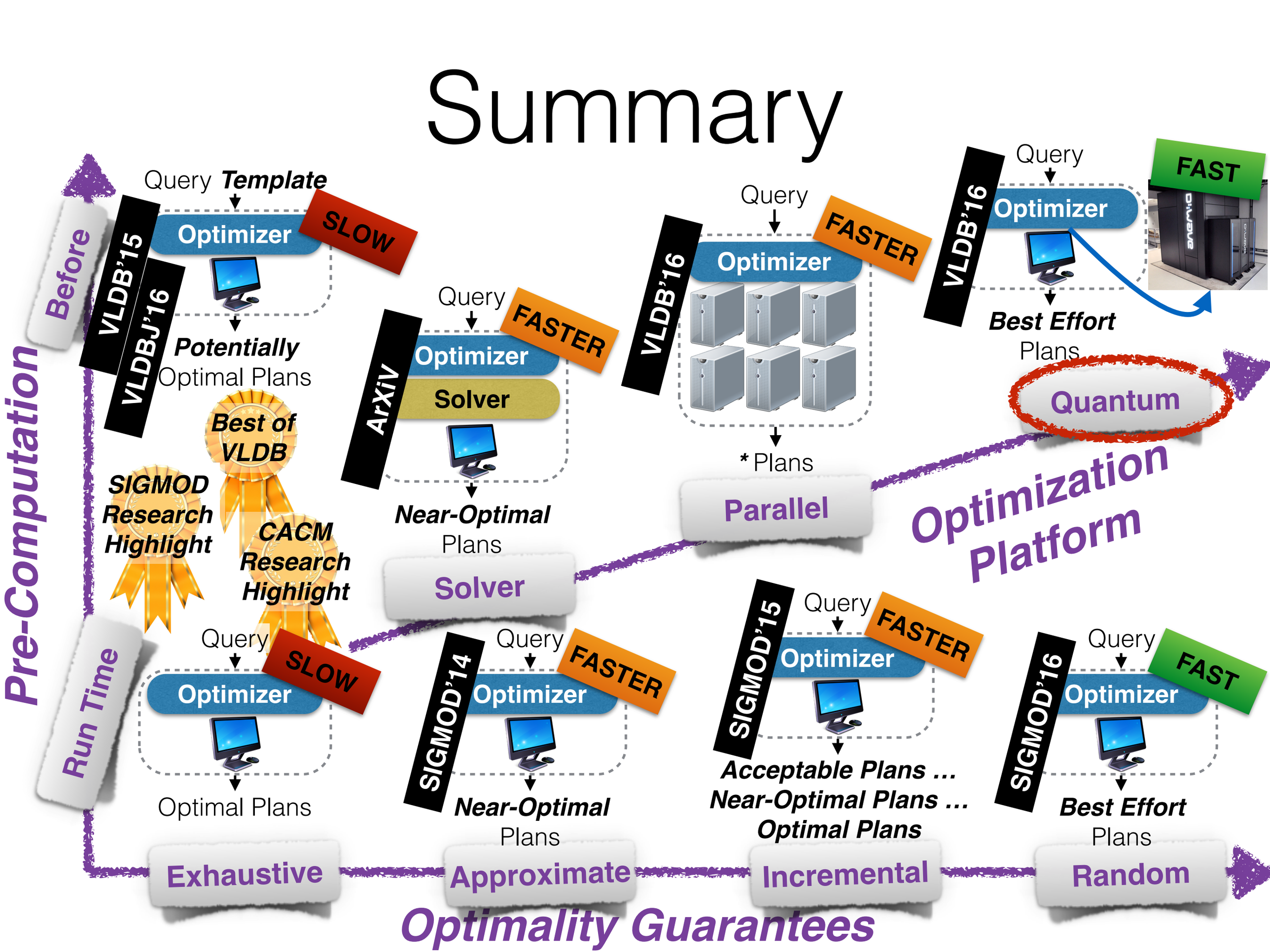
Keywords

Text mining; subjective properties; user behavior model

1. INTRODUCTION

In recent years, Web search engines have invested heavily in answering queries with structured data. For example, queries such as WOODY ALLEN MOVIES or AMERICAN PRESIDENTS will yield a display of the appropriate entities. These

Summary


Pre-Computation
Before
Query Template
Optimizer
SLOW
VLDB'15
VLDBJ'16
Potentially
Optimal Plans
Best of VLDB
SIGMOD Research Highlight
CACM Research Highlight
Run Time
Query
SLOW
Optimizer
Optimal Plans
Exhaustive
Query
FASTER
Optimizer
Solver
ArXiV
Near-Optimal
Plans
Solver
Query
FASTER
Optimizer
SIGMOD'14
Near-Optimal
Plans
Approximate
Query
FASTER
Optimizer
VLDB'16
* Plans
Parallel
Query
FASTER
Optimizer
SIGMOD'15
Acceptable Plans …
Near-Optimal Plans …
Optimal Plans
Incremental
Query
Optimizer
VLDB'16
Best Effort
Plans
FAST
Quantum
Optimization
Platform
Query
FAST
Optimizer
SIGMOD'16
Best Effort
Plans
Random
Optimality Guarantees