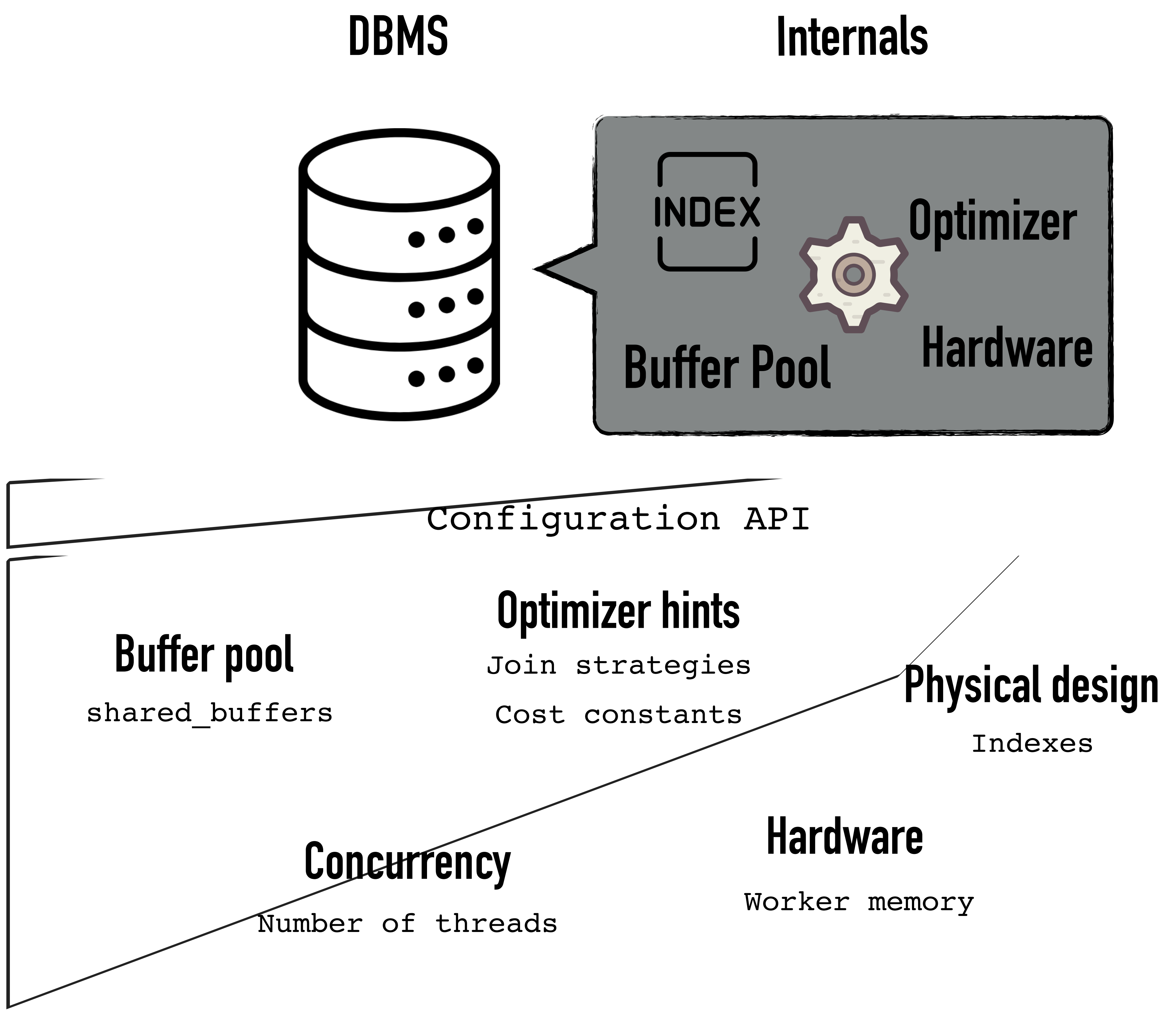
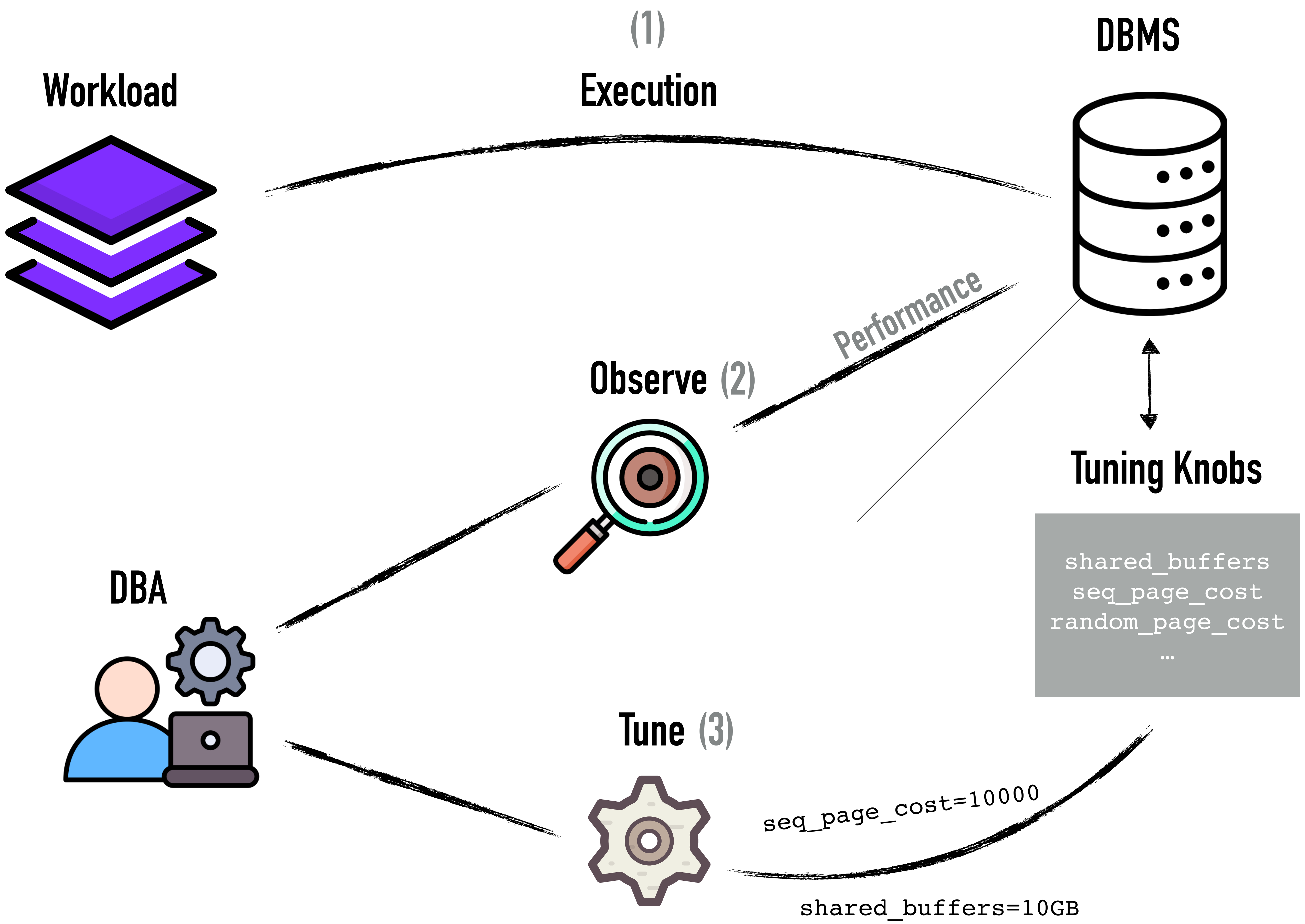
λ-TUNE: HARNESSING LARGE LANGUAGE MODELS FOR AUTOMATED DATABASE SYSTEM TUNING

Victor Giannakouris, Immanuel Trummer
Computer Science Department
Cornell University

HOW TO TUNE A DATABASE SYSTEM



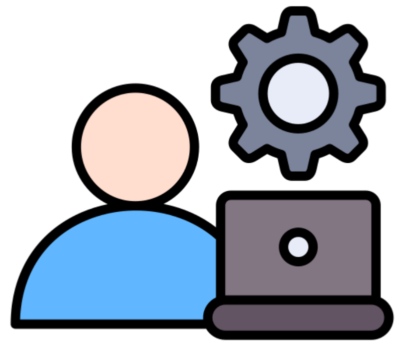
TUNING PIPELINE



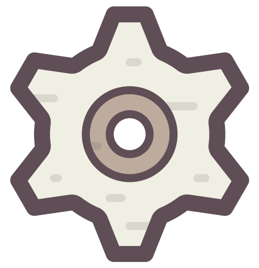
TUNING DATABASE SYSTEMS IS HARD

Combinatorial search space of tuning hints (knobs and values / indexes)

DBA



Tune

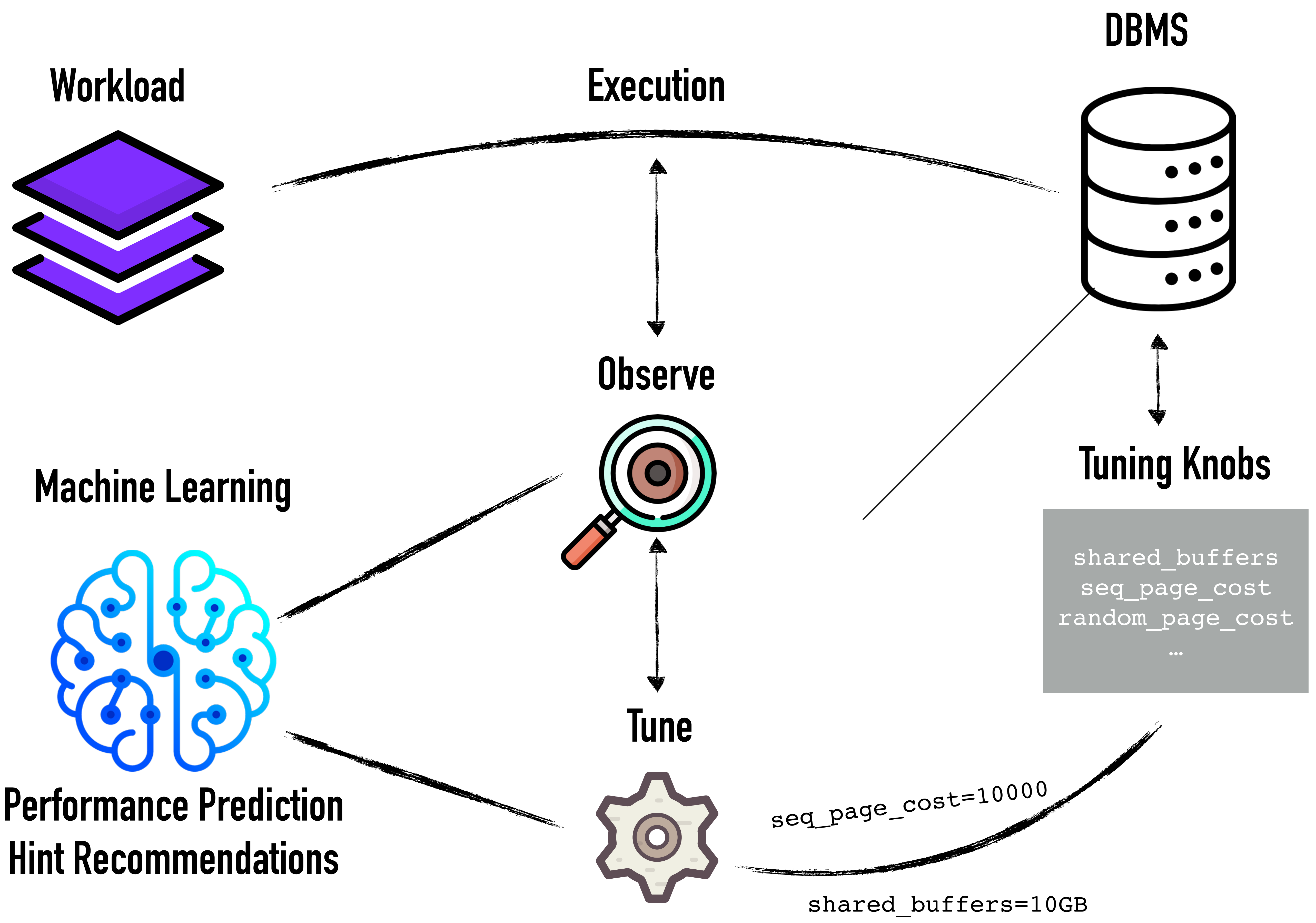


```
seq_page_cost=10000
seq_page_cost=50000

shared_buffers=10GB
shared_buffers=5GB

enable_nestloop=OFF
enable_nestloop=ON

CREATE INDEX ON TBL_1 (COL_1)
CREATE INDEX ON TBL_2 (COL_2)
```

Final hint set

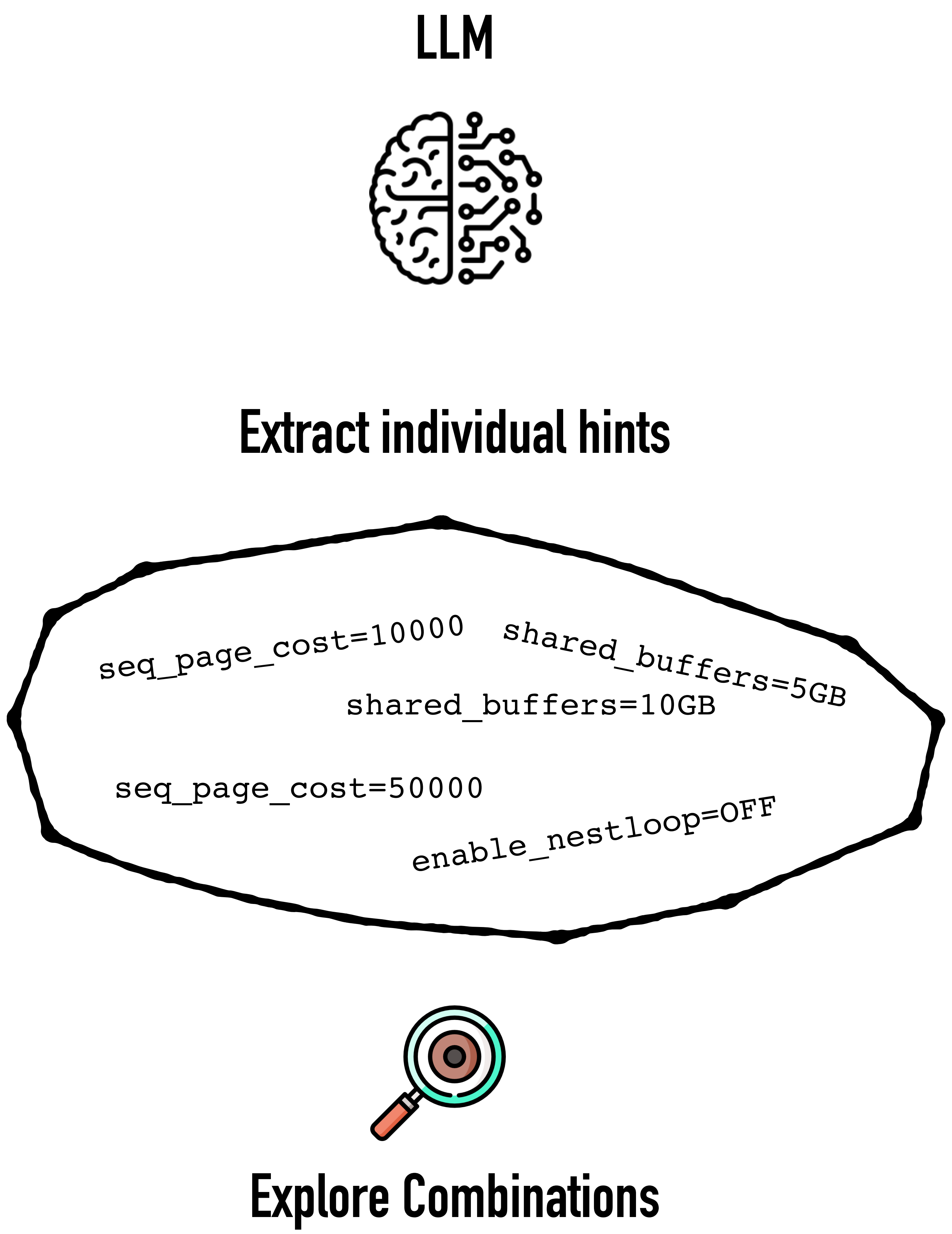
```
shared_buffers
seq_page_cost
random_page_cost
…
```

TUNING USING LEARNED SOLUTIONS



LANGUAGE MODEL BASED SOLUTIONS

- DB-BERT
  - Language model: BERT
  - Approach: Reinforcement Learning
- GPTuner
  - Language model: GPT-4
  - Bayesian Optimization
- LLM Usage
  - Extract ***individual tuning hints*** from documents
  - Still have to deal with a vast search space of ***hint combinations***



EXPLOITING THE LATEST ADVANCES OF LARGE LANGUAGE MODELS

- Latest Language Models support large prompt & response sizes
  - Not possible with BERT
- LLMs can be exploited further
  - Domain-specific tuning practices are *already* *included in their pre-trained weights*
- Instead of extracting single hints
  - Ask the LLM directly for *only a few sets* of full configurations

EXPLOITING THE LATEST ADVANCES OF LARGE LANGUAGE MODELS

BERT

Model

512 Tokens

Input

No generative output

Output

365M Parameters

EXPLOITING THE LATEST ADVANCES OF LARGE LANGUAGE MODELS

BERT

Model

512 Tokens

Input

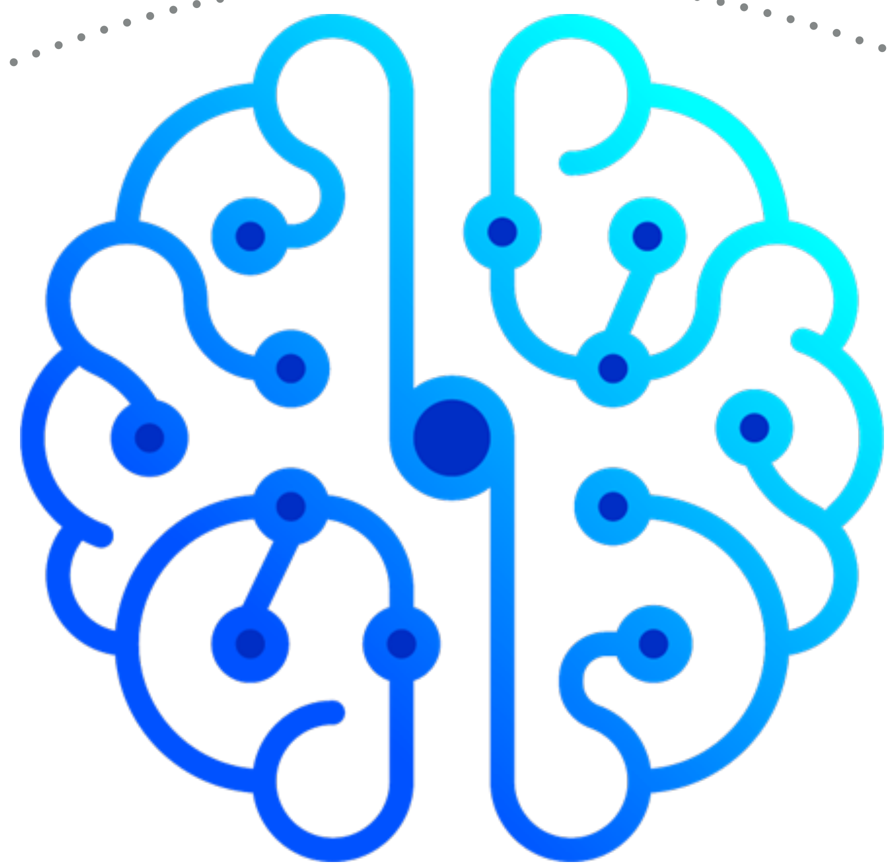
No generative output

Output

365M Parameters

GPT-4

Input



Output

Generative Output

EXPLOITING THE LATEST ADVANCES OF LARGE LANGUAGE MODELS

GPT-4

Input

Output

More information
Tuning context
System
Workload

Generate entire
configurations

Domain-specific knowledge
Tuning practices

EXPLOITING THE LATEST ADVANCES OF LARGE LANGUAGE MODELS

LLM

Extract individual hints

seq_page_cost=10000
shared_buffers=5GB
shared_buffers=10GB
seq_page_cost=50000
enable_nestloop=OFF

Explore Combinations

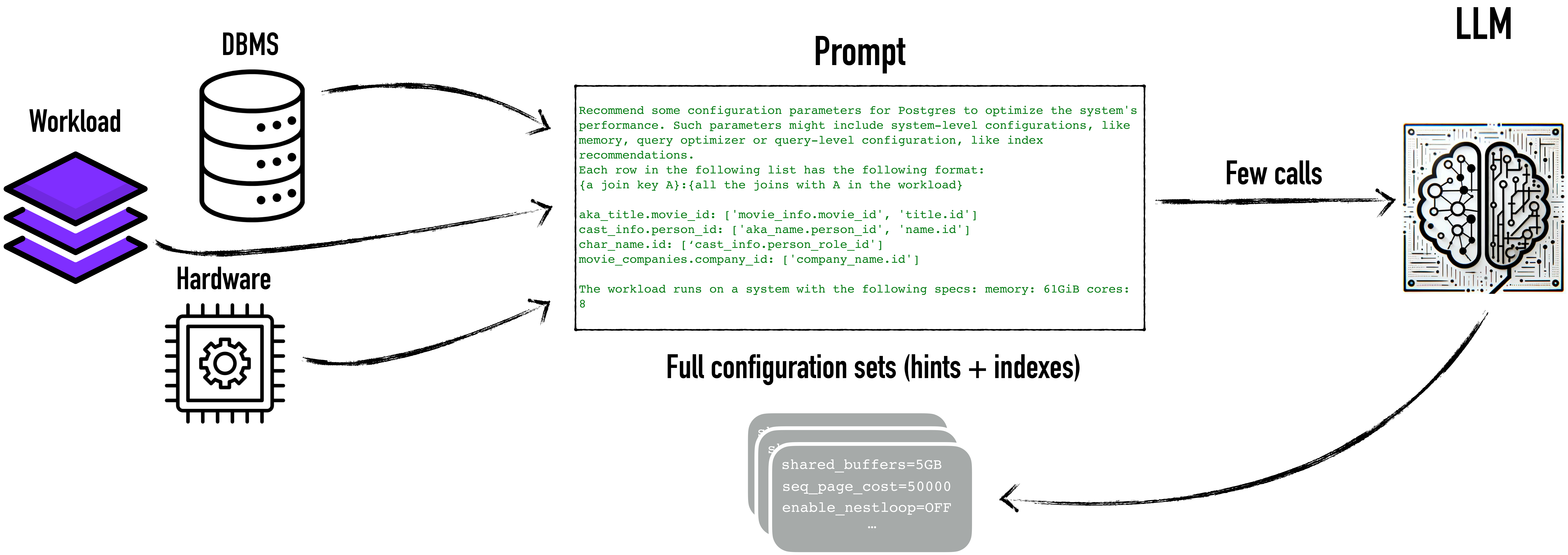
LLM

Generate Full Configurations

shared_buffers=5G seq_page_cost=5000 enable_nestloop=OF	shared_buffers=15 seq_page_cost=1000 enable_nestloop=ON
shared_buffers=15GB seq_page_cost=10000 enable_hashjoin=OFF	work_mem=512MB random_page_cost=1.1 enable_mergejoin=OFF

Find the best one

λ-Tune


DBMS
Workload
Hardware
Prompt
Recommend some configuration parameters for Postgres to optimize the system's performance. Such parameters might include system-level configurations, like memory, query optimizer or query-level configuration, like index recommendations.
Each row in the following list has the following format:
{a join key A}:{all the joins with A in the workload}
aka_title.movie_id: ['movie_info.movie_id', 'title.id']
cast_info.person_id: ['aka_name.person_id', 'name.id']
char_name.id: ['cast_info.person_role_id']
movie_companies.company_id: ['company_name.id']
The workload runs on a system with the following specs: memory: 61GiB cores: 8
Few calls
LLM
Full configuration sets (hints + indexes)
shared_buffers=5GB
seq_page_cost=50000
enable_nestloop=OFF
…

λ-Tune

How to generate a good prompt?

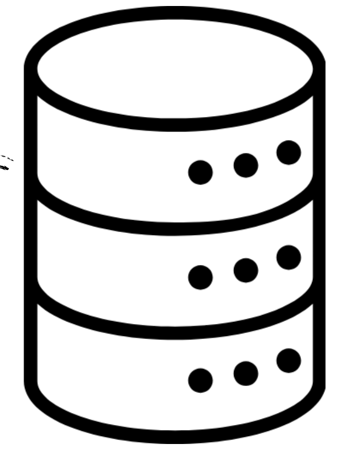
```
Recommend some configuration parameters for Postgres to optimize the system's
performance. Such parameters might include system-level configurations, like
memory, query optimizer or query-level configuration, like index
recommendations.
Each row in the following list has the following format:
{a join key A}:{all the joins with A in the workload}

aka_title.movie_id: ['movie_info.movie_id', 'title.id']
cast_info.person_id: ['aka_name.person_id', 'name.id']
char_name.id: ['cast_info.person_role_id']
movie_companies.company_id: ['company_name.id']

The workload runs on a system with the following specs: memory: 61GiB cores:
8
```

How to select among the retrieved configurations?

```
shared_buffers=5GB
seq_page_cost=50000
enable_nestloop=OFF
…
```

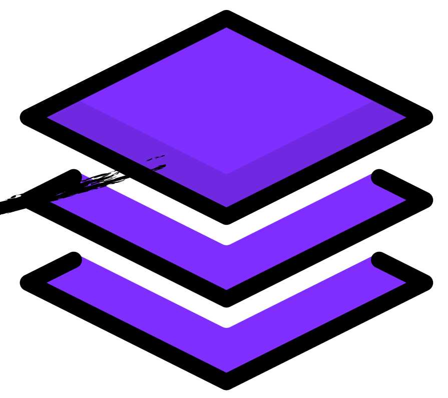
PROMPT GENERATION – PROMPT TEMPLATE

DBMS

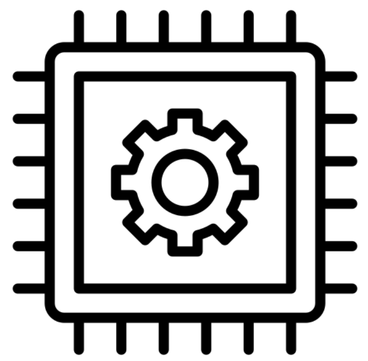


```
Recommend configuration parameters for ${DBMS}$ to
optimize the system's performance. Parameters might
include system-level configurations, like memory,
query optimizer or physical design configurations,
like index recommendations.

Each row in the following list has the following format:

{a join key A}:{all the joins with A in the workload}

${COMPRESSED_WORKLOAD}$

The workload runs on a system with the following specs:

memory: ${MEMORY}$

cores: ${CORES}$
```

Workload



Hardware



PROMPT GENERATION - COMPRESSION

```
Recommend configuration parameters for ${DBMS}$ to
optimize the system's performance. Parameters might
include system-level configurations, like memory,
query optimizer or physical design configurations,
like index recommendations.

Each row in the following list has the following format:

{a join key A}:{all the joins with A in the workload}

${COMPRESSED_WORKLOAD}$

The workload runs on a system with the following specs:

memory: ${MEMORY}$

cores: ${CORES}$
```

PROMPT GENERATION – COMPRESSION

- Naive approach
  - Include all the SQL queries of the workload
- High monetary costs
  - Pay per token to the LLM provider
  - Lengthy queries
  - Redundant information
    - Queries might be similar (e.g. joins)
- Large workloads do not even fit in the prompt
  - Model prompt size limitations

PROMPT GENERATION - COMPRESSION

- Decompose input queries into smaller components
  - e.g. joins
- Select the most important components (operators) with respect to
  - A user-defined token budget
  - A computational cost – how expensive an operator is

PROMPT GENERATION – COMPONENT SELECTION

- Assign each component ***p*** to
  - An execution cost ***V(p)*** (the optimizer's estimated cost)
  - A token size ***H***
- Integer Linear Program (ILP)
  - Maximize the value ***V(p)***
  - With a token budget ***B***

Maximize: $$\sum_{p \in P} V(p) \cdot R_p$$

Subject to: $$\sum_{\langle c_1, c_2 \rangle \in P} H_{c_2} \cdot R_{\langle c_1, c_2 \rangle} + \sum_{c \in C} H_c \cdot L_c \leq B$$

Query execution plan

```
tpch=# explain select * from orders o, customer c where o_custkey = c_custkey;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Hash Join  (cost=69577.00..519864.11 rows=15000000 width=265)
   Hash Cond: (o.o_custkey = c.c_custkey)
   ->  Seq Scan on orders o  (cost=0.00..410912.00 rows=15000000 width=107)
   ->  Hash  (cost=50827.00..50827.00 rows=1500000 width=158)
         ->  Seq Scan on customer c  (cost=0.00..50827.00 rows=1500000 width=158)
 JIT:
   Functions: 10
   Options: Inlining true, Optimization true, Expressions true, Deforming true
(8 rows)
```

(Obtained using the EXPLAIN clause)

CONFIGURATION SELECTION

<u>Evaluate all configurations in a random order</u>

1. Reconfigure the system
2. Run all queries
3. Find the optimal configuration

CONFIG 1

```
shared_buffers=5GB
seq_page_cost=50000
enable_nestloop=OFF
CREATE INDEX ..
```

EXECUTE → TIME

CONFIG 2

```
shared_buffers=5GB
seq_page_cost=10000
enable_hashjoin=OFF
CREATE INDEX ..
```

EXECUTE → TIME

CONFIG 3

```
shared_buffers=15G
seq_page_cost=50000
enable_mergejoin=OFF
…
```

EXECUTE → TIME

CONFIG 4

```
shared_buffers=5GB
seq_page_cost=50000
enable_nestloop=OFF
CREATE INDEX..
```

EXECUTE → TIME

PICK THE CONFIGURATION THAT ACHIEVES THE MINIMUM EXECUTION TIME

CONFIGURATION SELECTION

The execution order matters

CONFIG 1	CONFIG 2	CONFIG 3	CONFIG 4
shared_buffers=5GB seq_page_cost=50000 enable_nestloop=OFF CREATE INDEX ..	shared_buffers=5GB seq_page_cost=10000 enable_hashjoin=OFF CREATE INDEX ..	shared_buffers=15G seq_page_cost=50000 enable_mergejoin=OFF …	shared_buffers=5GB seq_page_cost=50000 enable_nestloop=OFF CREATE INDEX..
EXECUTE	EXECUTE	EXECUTE	EXECUTE
43s	2m	14m	5m

Set 43s as the timeout for the rest

CONFIGURATION SELECTION

The execution order matters

CONFIG 1	CONFIG 2	CONFIG 3	CONFIG 4
shared_buffers=15G seq_page_cost=50000 enable_mergejoin=OFF …	shared_buffers=5GB seq_page_cost=10000 enable_hashjoin=OFF CREATE INDEX ..	shared_buffers=5GB seq_page_cost=50000 enable_nestloop=OFF CREATE INDEX ..	shared_buffers=5GB seq_page_cost=50000 enable_nestloop=OFF CREATE INDEX..
EXECUTE	EXECUTE	EXECUTE	EXECUTE
14m	2m	43s	5m

Need to spend 14 minutes before finding a good configuration

CONFIGURATION SELECTION – EVALUATION ROUNDS

Allocate each configuration equal execution time per round

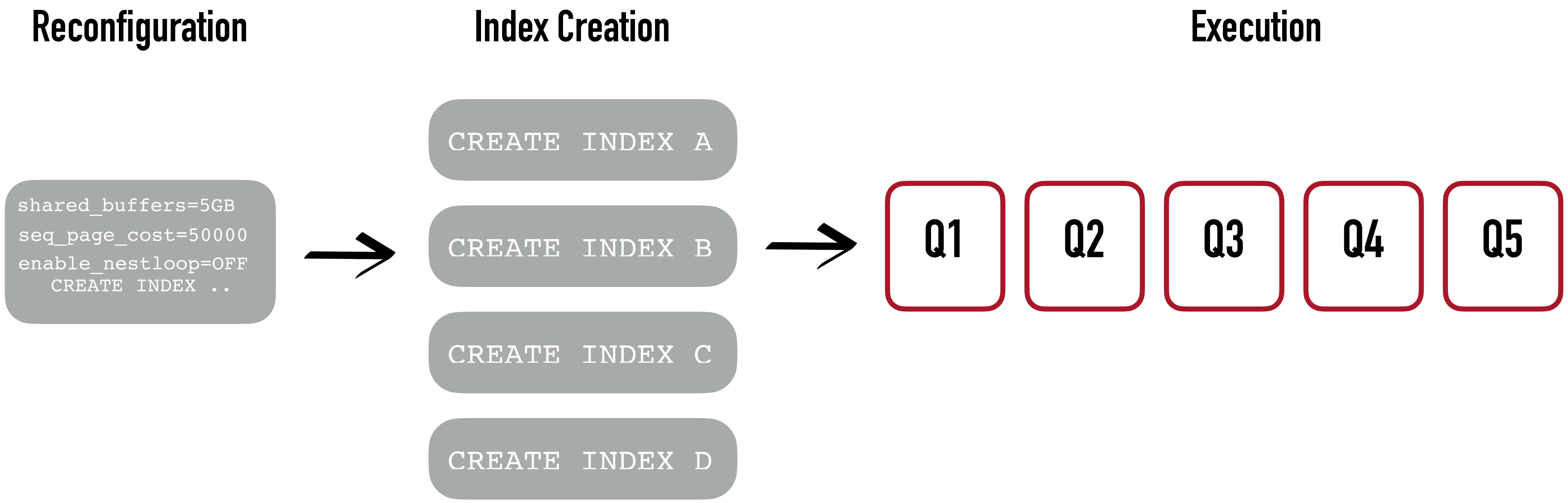
CONFIG 1	CONFIG 2	CONFIG 3	CONFIG 4
shared_buffers=15G seq_page_cost=50000 enable_mergejoin=OFF …	shared_buffers=5GB seq_page_cost=10000 enable_hashjoin=OFF CREATE INDEX ..	shared_buffers=5GB seq_page_cost=50000 enable_nestloop=OFF CREATE INDEX ..	shared_buffers=5GB seq_page_cost=50000 enable_nestloop=OFF CREATE INDEX..
EXECUTE	EXECUTE	EXECUTE	EXECUTE
10s (R1), 20s (R2), 40s (R3)	10s (R1), 20s (R2), 40s (R3)	10s (R1), 20s (R2), 40s (R3)	10s (R1), 20s (R2), 40s (R3)

Execution time follows a geometric progression

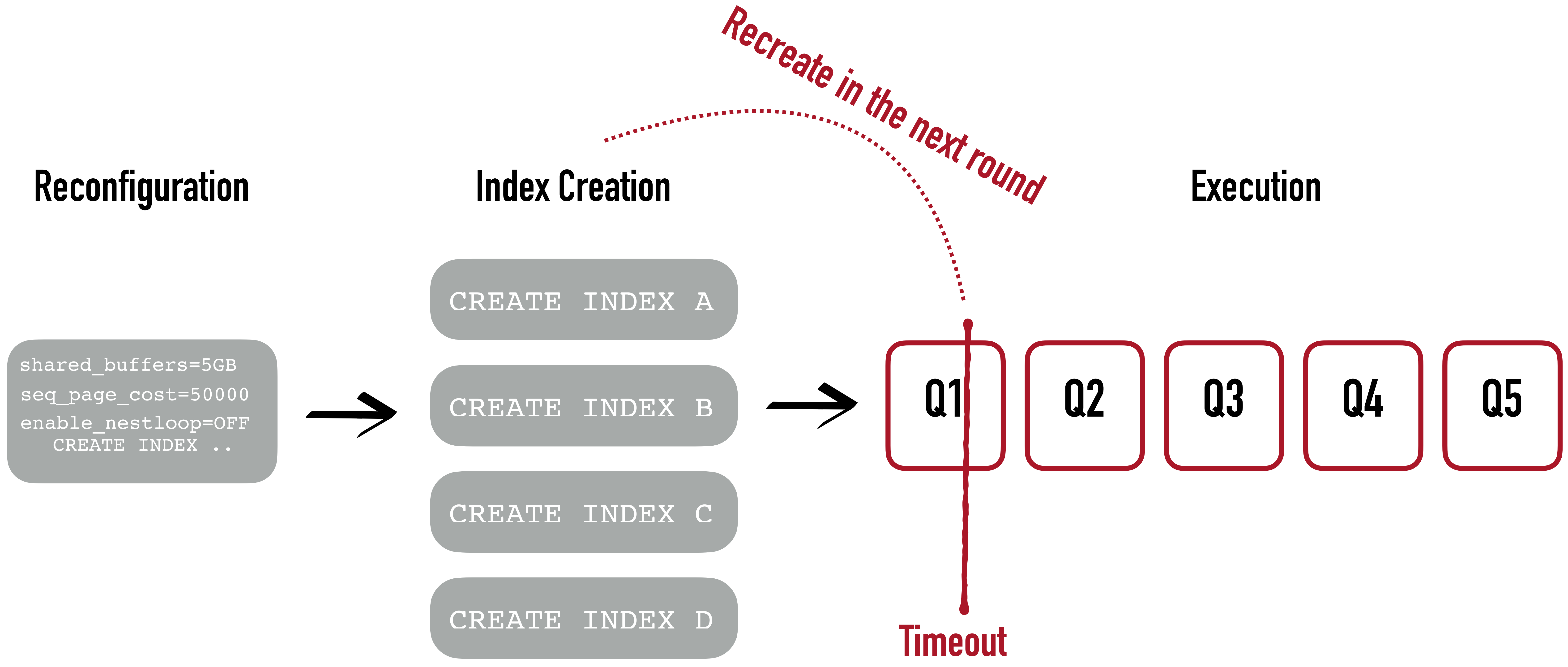
Given an initial timeout r and a factor a

$$R_1 = r \cdot a^0, R_2 = r \cdot a^1, R_3 = r \cdot a^2, \ldots, R_n = r \cdot a^{n-1}$$

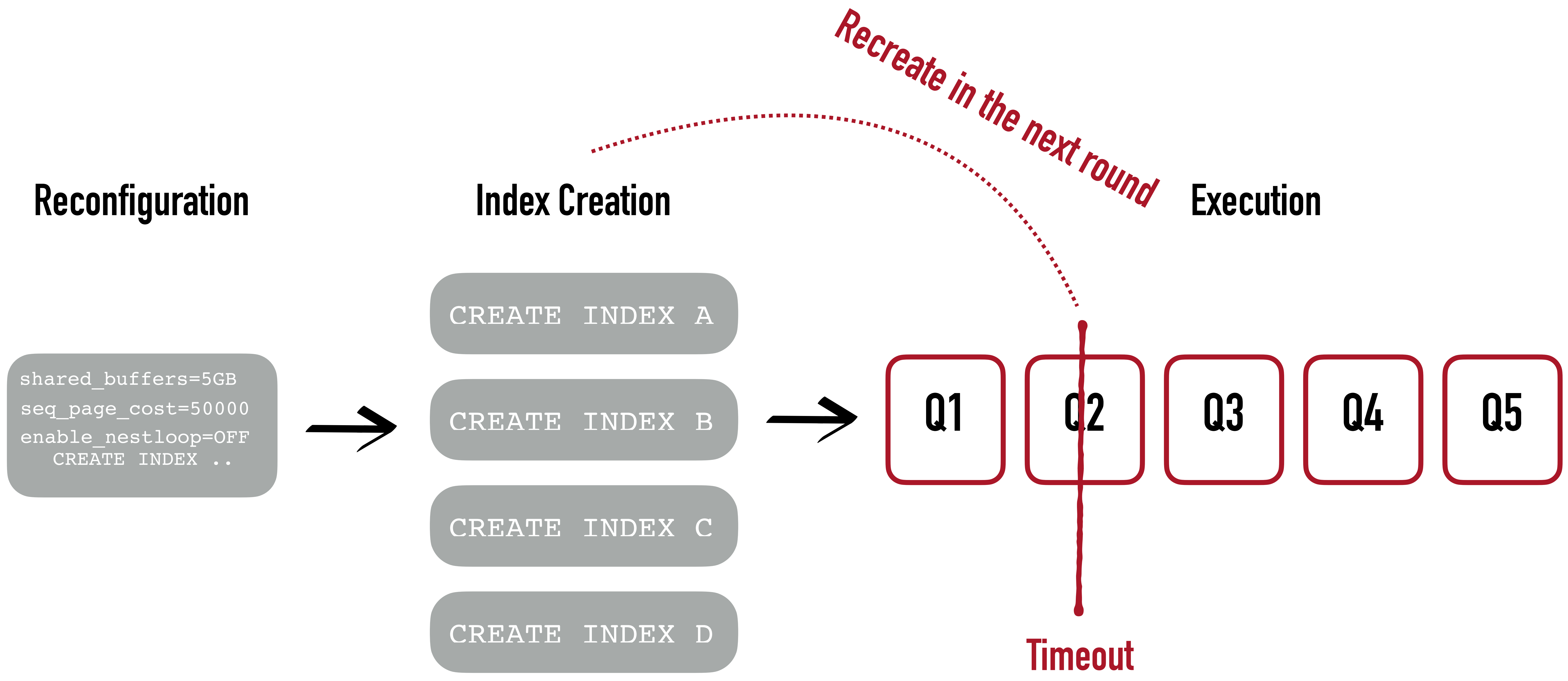
QUERY SCHEDULING



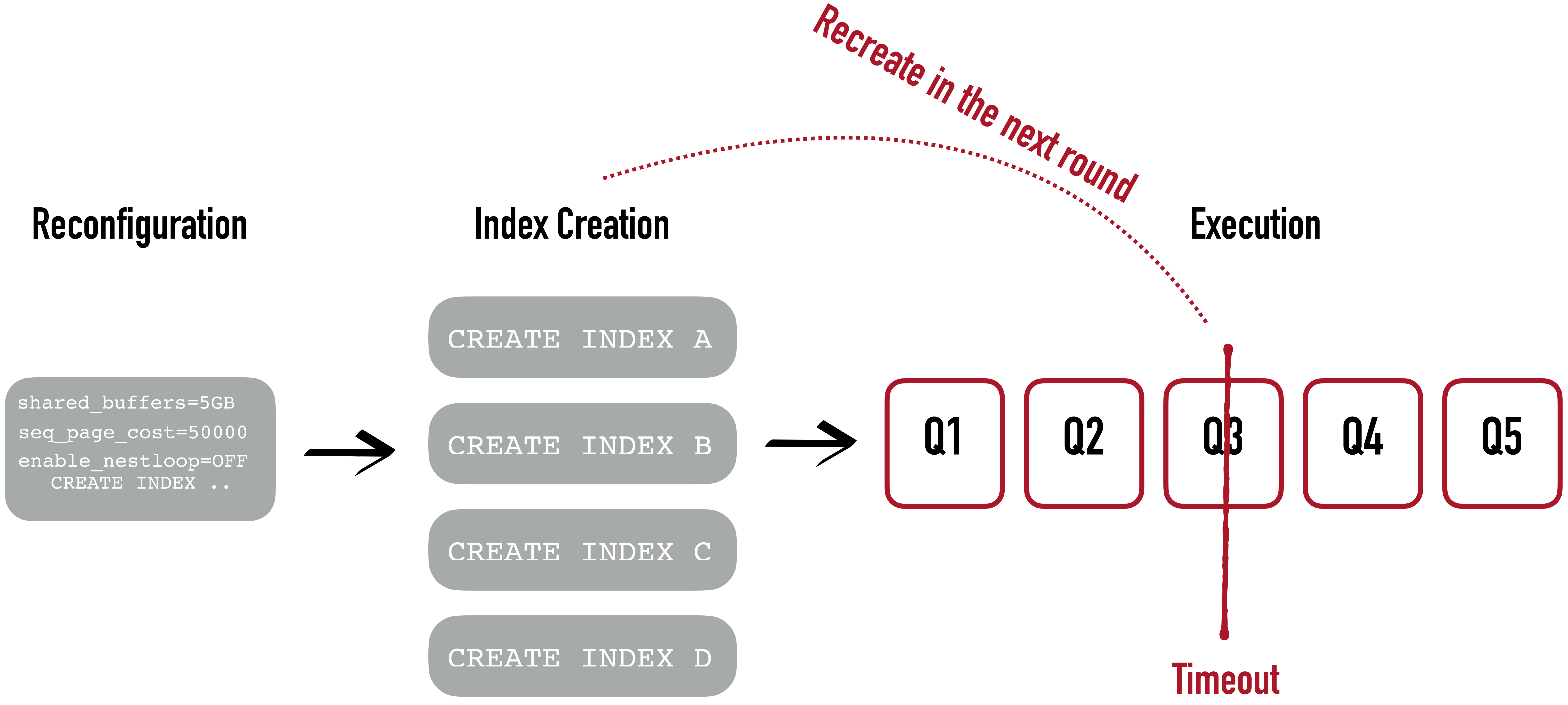
QUERY SCHEDULING



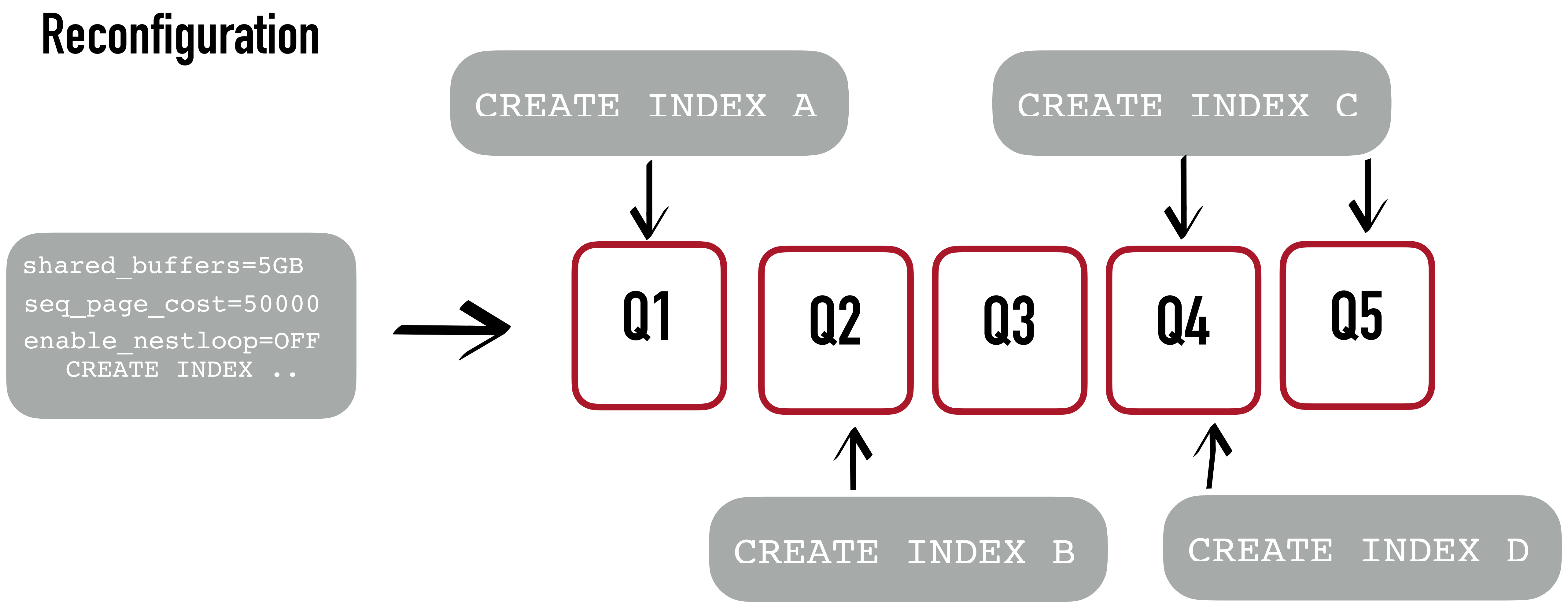
QUERY SCHEDULING



QUERY SCHEDULING



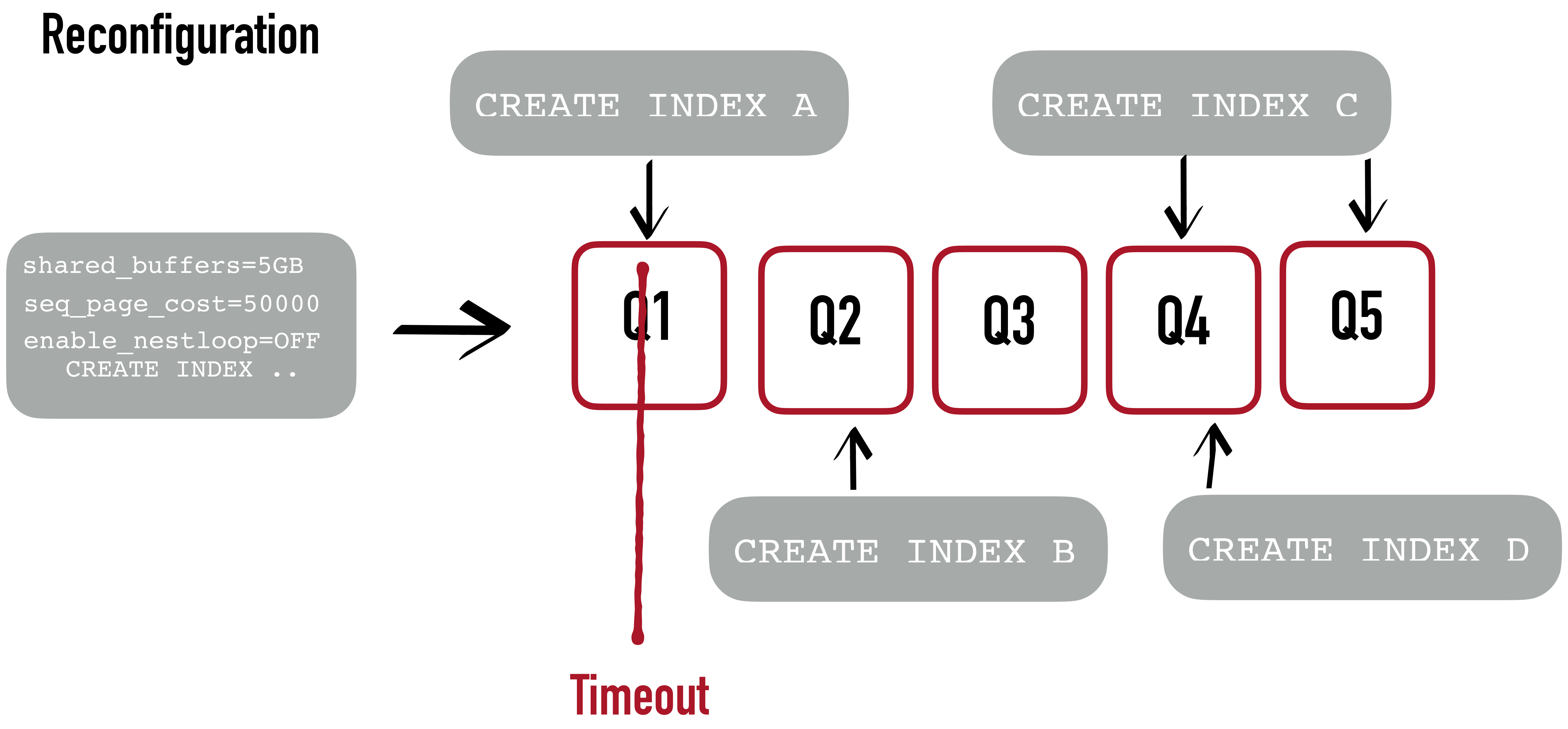
QUERY SCHEDULING



Interleave query execution with index creation

Lazy index creation

QUERY SCHEDULING



If the timeout is exceeded, we only pay the cost of the necessary indexes in each round

According to the index creation cost, we optimally order the queries using a dynamic programming approach

EXPERIMENTAL EVALUATION

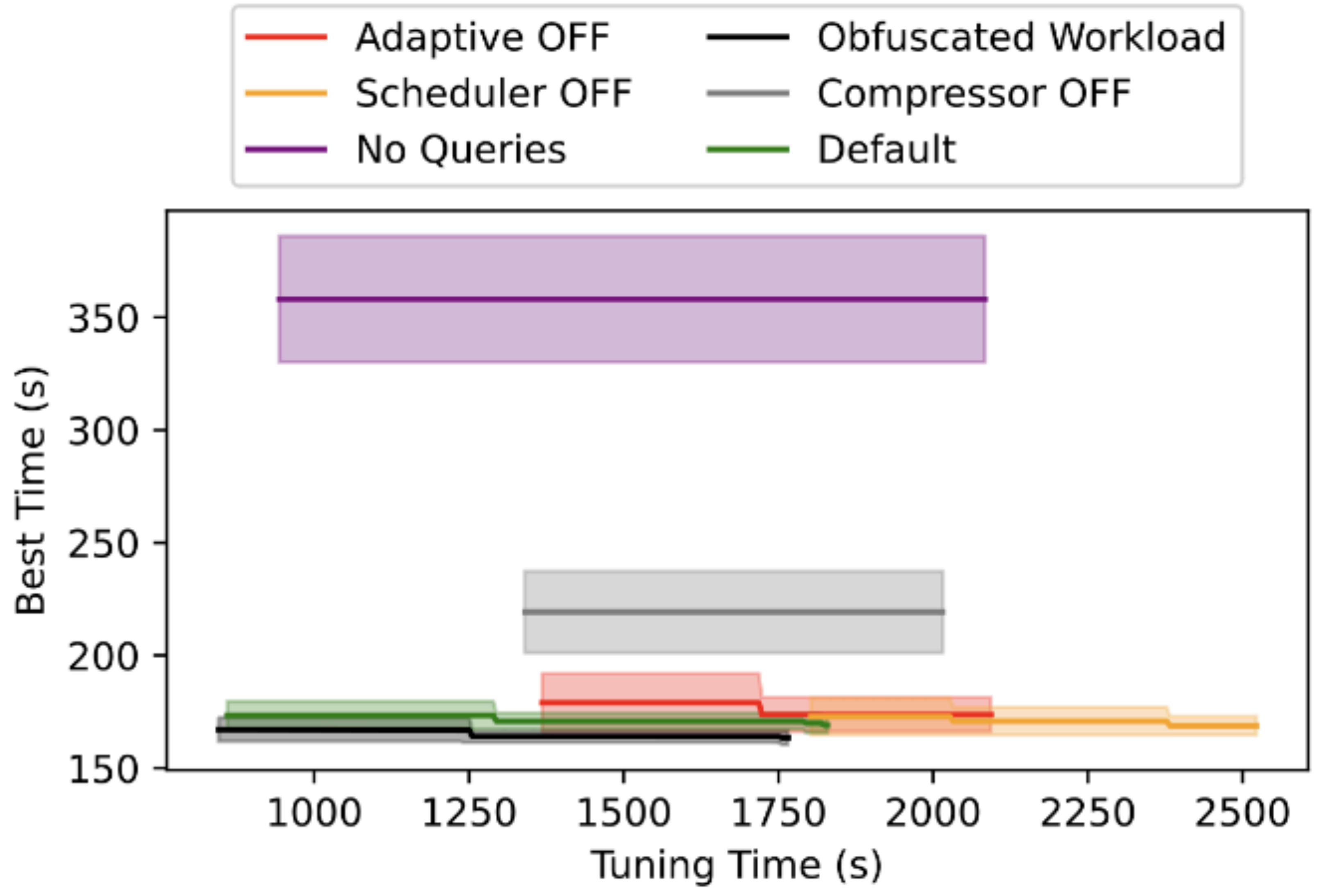
- Setup
  - p3.2xlarge EC2 Instance (AWS) with GPU
    - 8 vCPUs
    - 61 GiB Memory
- Benchmarks
  - Join Order Benchmark (JOB)
  - TPC-H 1GB / 10GB
  - TPC-DS
- Database Systems
  - Postgres
  - MySQL
- Baselines
  - DB-BERT, GPTuner
  - UDO, MLOS (LlamaTune), ParamTree,
  - Indexes
    - Dexter
    - DB2Advisor

ABLATION STUDY

Is the LLM aware of the input workload? (— Obfuscated Workload)

Is the compression important? (— Compressor OFF)

Does the scheduler accelerates the tuning process? (— Scheduler OFF)



Join Order Benchmark (JOB) over Postgres

EXPERIMENTAL EVALUATION – SUMMARY

COST OF THE BEST CONFIGURATION FOUND BY EACH APPROACH, SCALED TO THE BEST CONFIGURATION OVERALL

Benchmark	DBMS	Initial Indexes	λ-Tune	UDO	DB-Bert	GPTuner	LlamaTune	ParamTree
TPC-H 1GB	PG	Yes	1.07	1.96	1.13	1	2.08	3.23
TPC-H 1GB	MS	Yes	1.06	1	1.02	1.73	1.39	3.24
TPC-H 10GB	PG	Yes	1.03	1	1.05	1.04	2.38	3.18
TPC-H 10GB	MS	Yes	4.98	1	5.16	5.84	2.86	15.2
JOB	PG	Yes	1	1.32	1.05	1.1	3.48	3.48
JOB	MS	Yes	1	1.07	3.69	3.69	3.22	3.22
TPC-H 1GB	PG	No	1.05	3.76	1	1.06	1.43	4.24
TPC-H 1GB	MS	No	1.2	2.83	1.02	1	1.61	3.64
TPC-H 10GB	PG	No	1.65	1.54	2.45	2.52	1	1.54
TPC-H 10GB	MS	No	1.04	3.2	1.09	1	1.88	3.2
JOB	PG	No	1	1.69	1.08	1.13	3.09	3.26
JOB	MS	No	1	3.07	3.07	3.07	3.07	3.07
TPC-DS	PG	No	1	1.37	1.67	1.66	3.33	3.33
TPC-DS	MS	No	1.79	3.25	1	1.03	1.05	3.25
Average			1.41	2	1.82	1.91	2.27	4.07

EXPERIMENTAL EVALUATION – SUMMARY

COST OF THE BEST CONFIGURATION FOUND BY EACH APPROACH, SCALED TO THE BEST CONFIGURATION OVERALL

Benchmark	DBMS	Initial Indexes	λ-Tune	UDO	DB-Bert	GPTuner	LlamaTune	ParamTree
TPC-H 1GB	PG	Yes	1.07	1.96	1.13	1	2.08	3.23
TPC-H 1GB	MS	Yes	1.06	1	1.02	1.73	1.39	3.24
TPC-H 10GB	PG	Yes	1.03	1	1.05	1.04	2.38	3.18
TPC-H 10GB	MS	Yes	4.98	1	5.16	5.84	2.86	15.2
JOB	PG	Yes	1	1.32	1.05	1.1	3.48	3.48
JOB	MS	Yes	1	1.07	3.69	3.69	3.22	3.22
TPC-H 1GB	PG	No	1.05	3.76	1	1.06	1.43	4.24
TPC-H 1GB	MS	No	1.2	2.83	1.02	1	1.61	3.64
TPC-H 10GB	PG	No	1.65	1.54	2.45	2.52	1	1.54
TPC-H 10GB	MS	No	1.04	3.2	1.09	1	1.88	3.2
JOB	PG	No	1	1.69	1.08	1.13	3.09	3.26
JOB	MS	No	1	3.07	3.07	3.07	3.07	3.07
TPC-DS	PG	No	1	1.37	1.67	1.66	3.33	3.33
TPC-DS	MS	No	1.79	3.25	1	1.03	1.05	3.25
Average			1.41	2	1.82	1.91	2.27	4.07

EXPERIMENTAL EVALUATION – SUMMARY

COST OF THE BEST CONFIGURATION FOUND BY EACH APPROACH, SCALED TO THE BEST CONFIGURATION OVERALL

Benchmark	DBMS	Initial Indexes	λ-Tune	UDO	DB-Bert	GPTuner	LlamaTune	ParamTree
TPC-H 1GB	PG	Yes	1.07	1.96	1.13	1	2.08	3.23
TPC-H 1GB	MS	Yes	1.06	1	1.02	1.73	1.39	3.24
TPC-H 10GB	PG	Yes	1.03	1	1.05	1.04	2.38	3.18
TPC-H 10GB	MS	Yes	4.98	1	5.16	5.84	2.86	15.2
JOB	PG	Yes	1	1.32	1.05	1.1	3.48	3.48
JOB	MS	Yes	1	1.07	3.69	3.69	3.22	3.22
TPC-H 1GB	PG	No	1.05	3.76	1	1.06	1.43	4.24
TPC-H 1GB	MS	No	1.2	2.83	1.02	1	1.61	3.64
TPC-H 10GB	PG	No	1.65	1.54	2.45	2.52	1	1.54
TPC-H 10GB	MS	No	1.04	3.2	1.09	1	1.88	3.2
JOB	PG	No	1	1.69	1.08	1.13	3.09	3.26
JOB	MS	No	1	3.07	3.07	3.07	3.07	3.07
TPC-DS	PG	No	1	1.37	1.67	1.66	3.33	3.33
TPC-DS	MS	No	1.79	3.25	1	1.03	1.05	3.25
Average			1.41	2	1.82	1.91	2.27	4.07

CONCLUSIONS

- Large Language Models can significantly enhance the tuning process
  - Can speed-up automated tuning
- Latest advancements in LLMs motivate design changes
  - Full configuration generation
- λ-Tune
  - Prompting – workload compression with ILP
  - Efficient evaluation – minimize time spent in inefficient configurations

QUESTIONS?