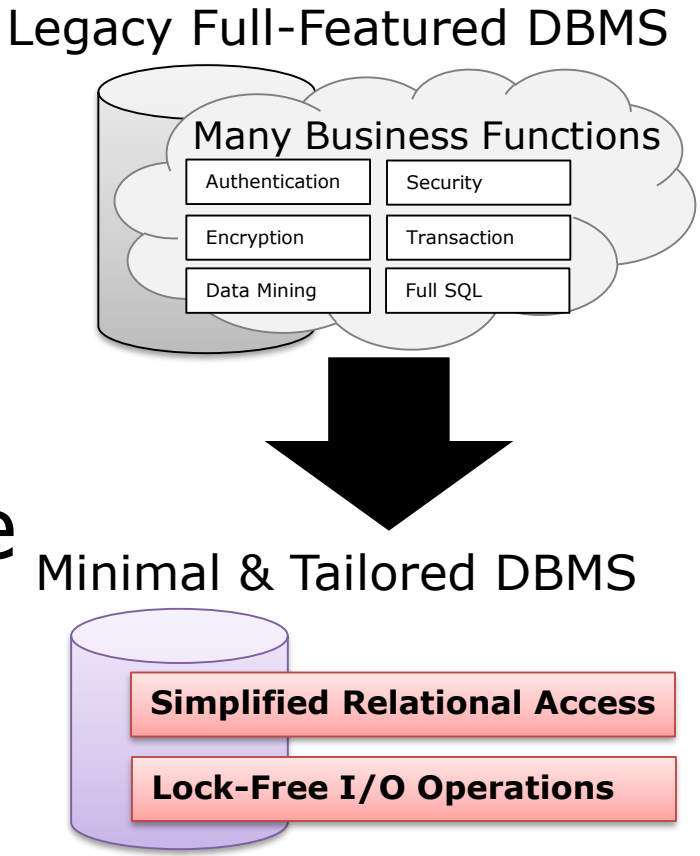
Create a 20 Times Faster Database Engine Optimized to MMOGs

Shuichi Kurabayashi
Technical Advisor/Director of Cygames Research
Cygames, Inc.
kurabayashi_shuichi@cygames.co.jp

GAME DEVELOPERS CONFERENCE EUROPE COLOGNE, GERMANY · 15-16 AUGUST 2016

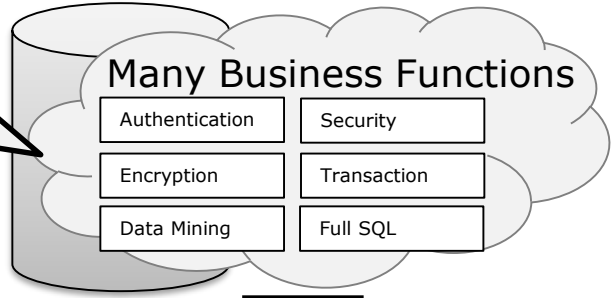
Summary

A **simple in-house database engine tailored to MMOGs** is highly effective to provide mobile MMOGs for multi-millions unique users.



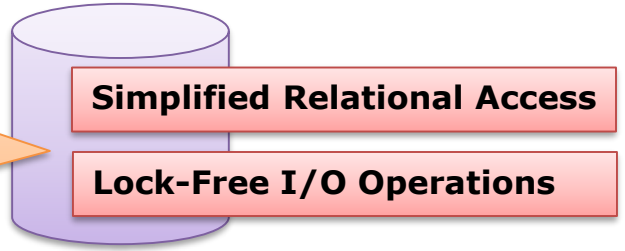
Game developers do not use almost of those business-oriented functions

Legacy Full-Featured DBMS



It is easy to implement the **minimal** database management system.

Minimal & Tailored DBMS



My message is:

Let’s create your own database engine tailored to your titles!

Quick Introduction

Who are you?

What is Cygames?

What is problem?

Introduction: Who I am

Shuichi Kurabayashi, Ph.D.

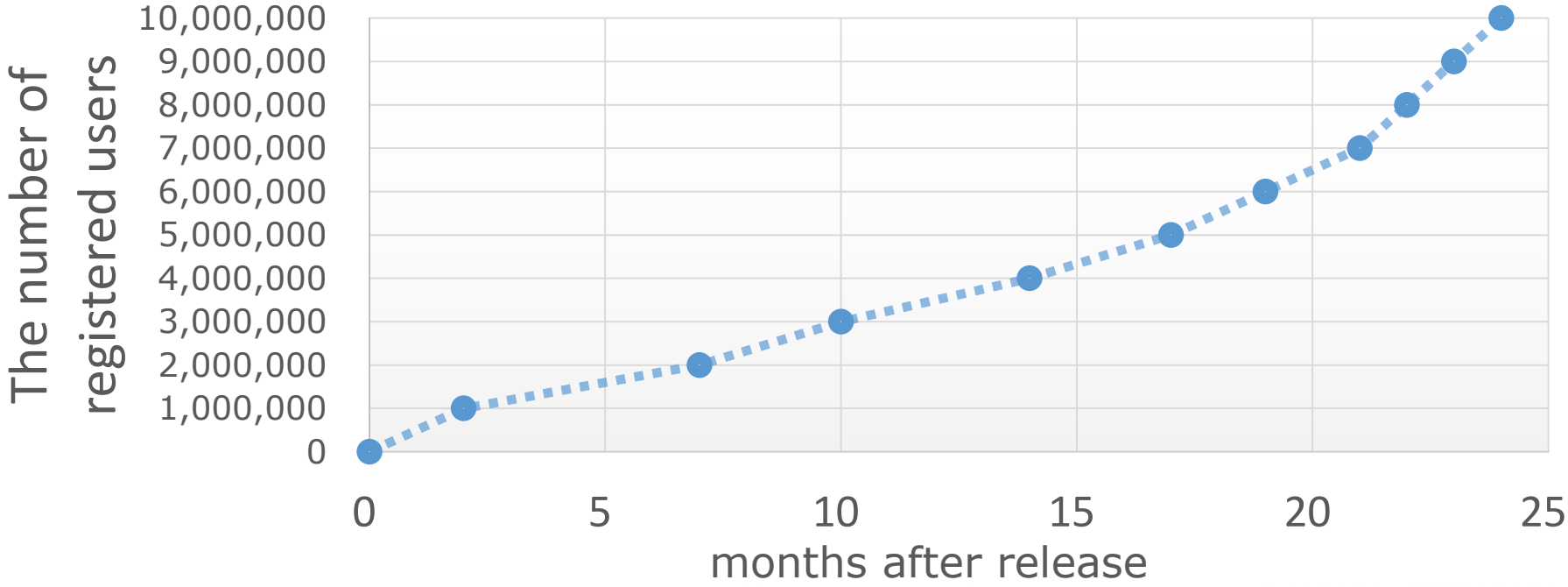
- Technical advisor of Cygames, Inc.
  Also Director of Cygames Research.
- Project Associate Professor at the Graduate School of Keio University

Introduction: Cygames is one of the largest mobile game developers in Japan

Known as the developer of the wildly popular card battle game “Rage of Bahamut”.

Recently released “Shadowverse”.

Background: We have been providing mobile MMOGs for 10 millions users.



Background: Mobile gaming is an important subject of DB Research

Tokyo Stock Exchange

6,700/sec (400,000/min) transactions

Twitter

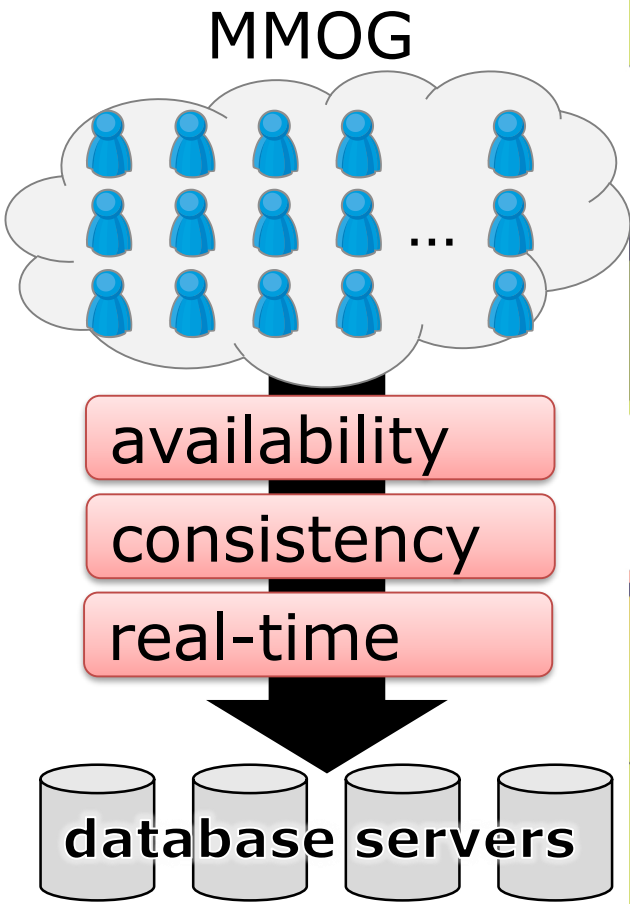
40,000/sec messages at peak.

Mobile game (Japan)

100,000/sec transactions in average.

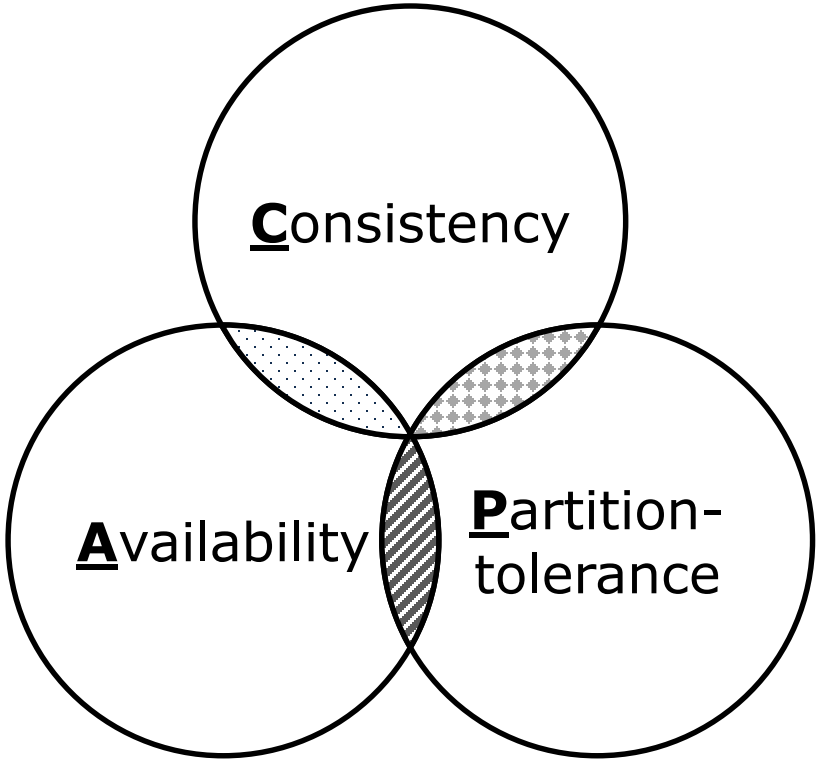
Problem Definition

- Modern MMOGs require databases to support large-scale **availability**, strong **consistency**, and **real-time** response.
- It is difficult to support such capabilities efficiently by using conventional systems.



CAP Theorem

- **C: Consistency of data**
  - After data has been updated, if something else references that data, it will always be guaranteed that the updated data can be referenced.
- **A: Availability of the system**
  - No matter what the current situation, the system will continue to operate.
- **P: Tolerance to network partitions**
  - Data can be distributed.

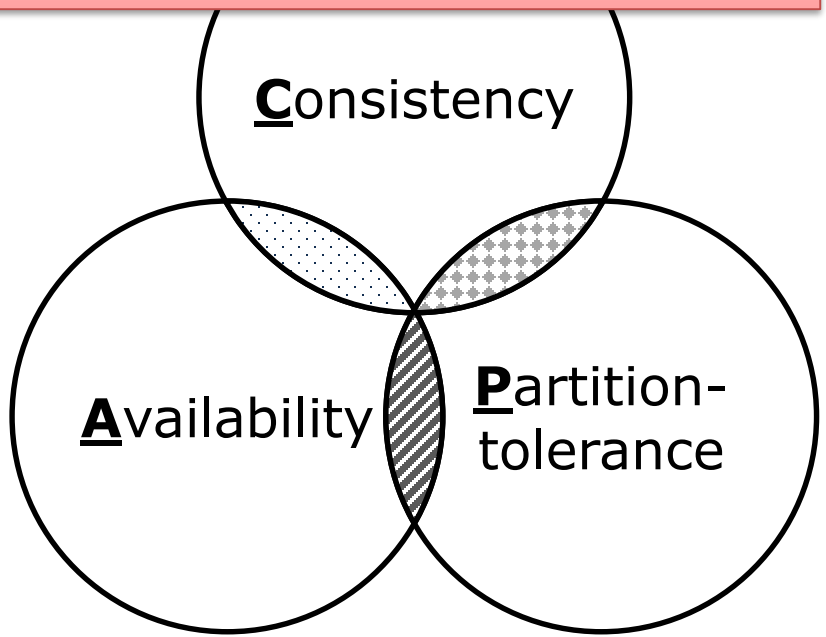


http://berb.github.io/diploma-thesis/original/061_challenge.html

CAP Theorem

- **C: Consistency of data**
  - After data has been updated, if something else references that data, it will always be guaranteed that the updated data can be referenced.
- **A: Availability of the system**
  - No matter what the current situation, the system will continue to operate.
- **P: Tolerance to network partitions**
  - Data can be distributed.

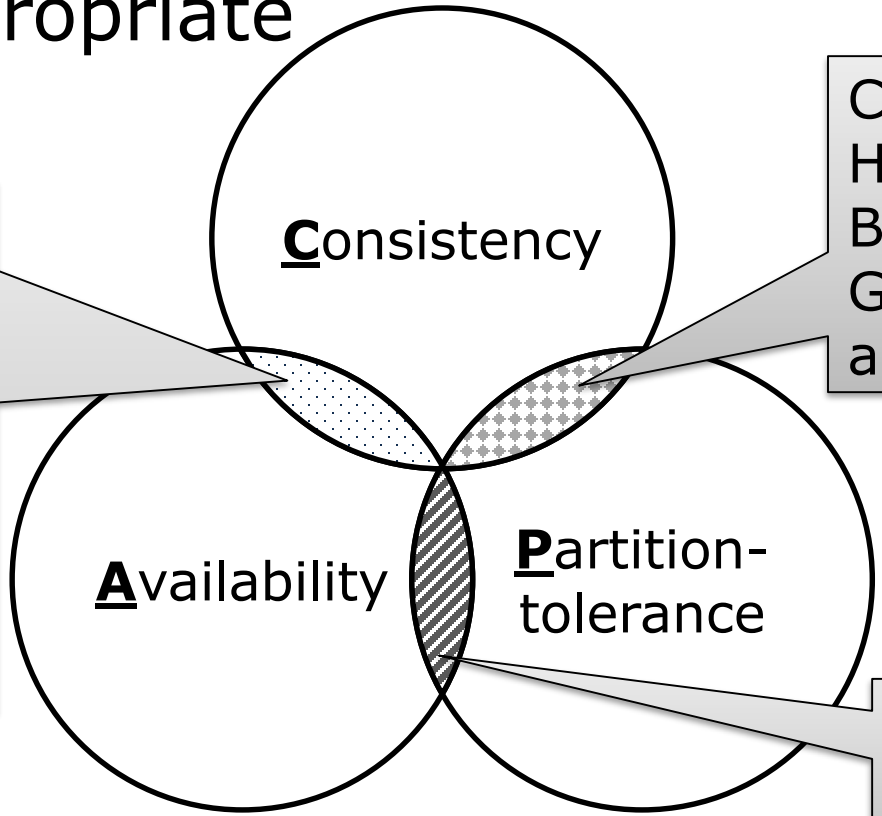
Out of these three guarantees, only two can be fulfilled at a time.



http://berb.github.io/diploma-thesis/original/061_challenge.html

Choosing Appropriate System

CA: conventional relational database systems, such as MySQL, Oracle, and SQL server, strong consistency



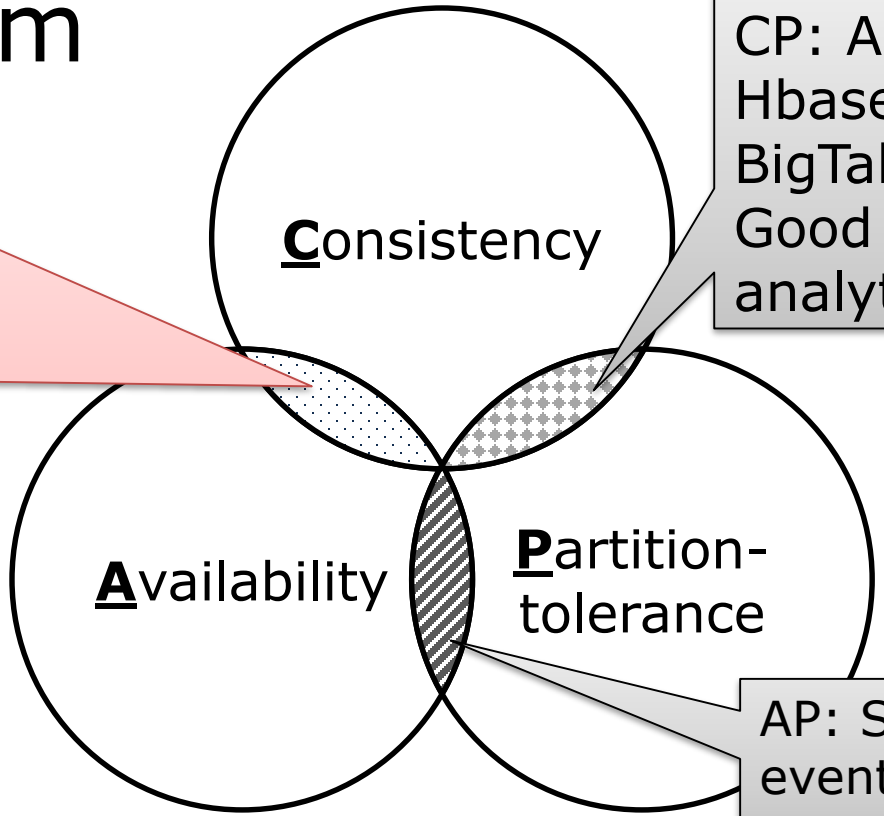
CP: Apache Hbase, Hadoop, BigTable, Good for analytics

AP: SNS,NoSQL, eventual consistency

http://berb.github.io/diploma-thesis/original/061_challenge.html

CAP Theorem

CA is the most important for games, because games require **strong consistency and fast response**

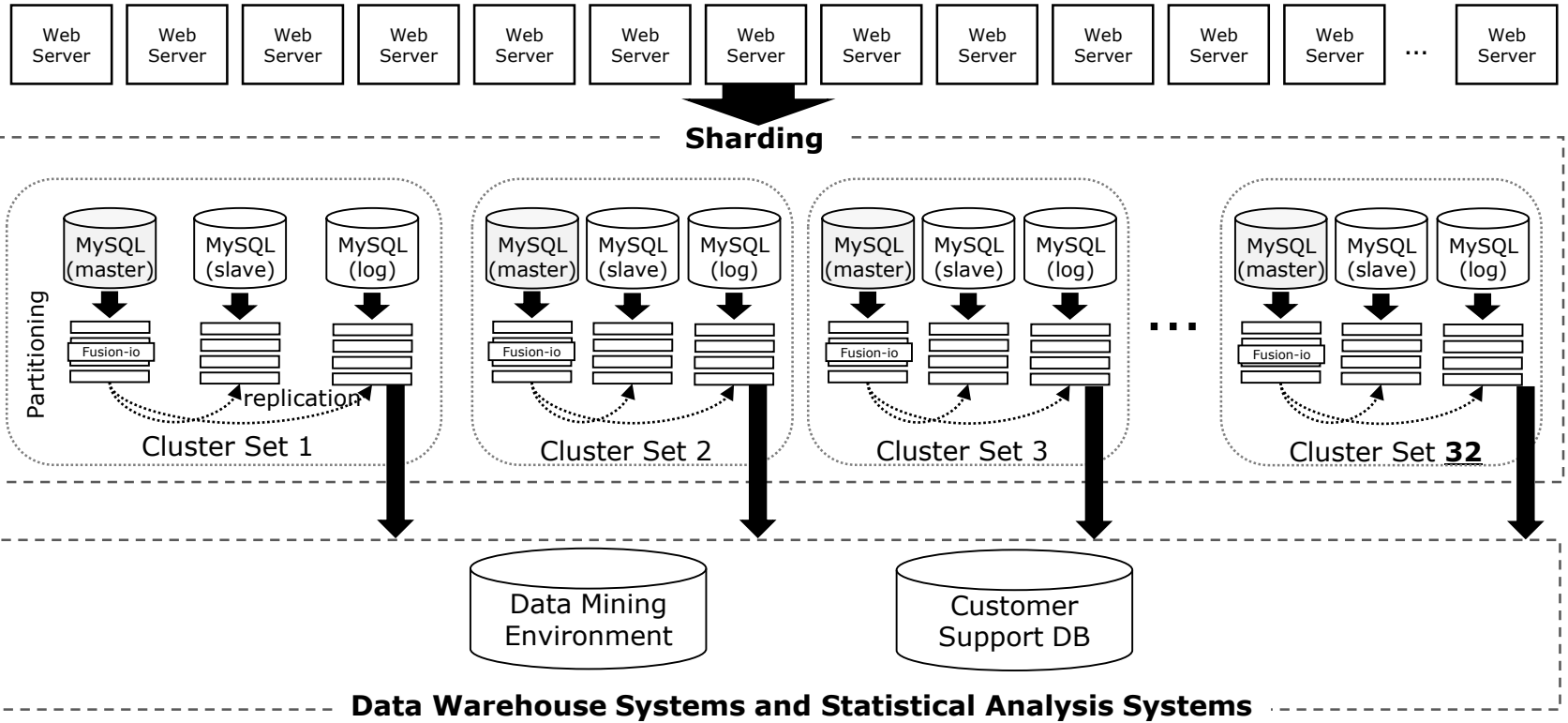


CP: Apache Hbase, Hadoop, BigTable, Good for analytics

AP: SNS,NoSQL, eventually consistency

http://berb.github.io/diploma-thesis/original/061_challenge.html

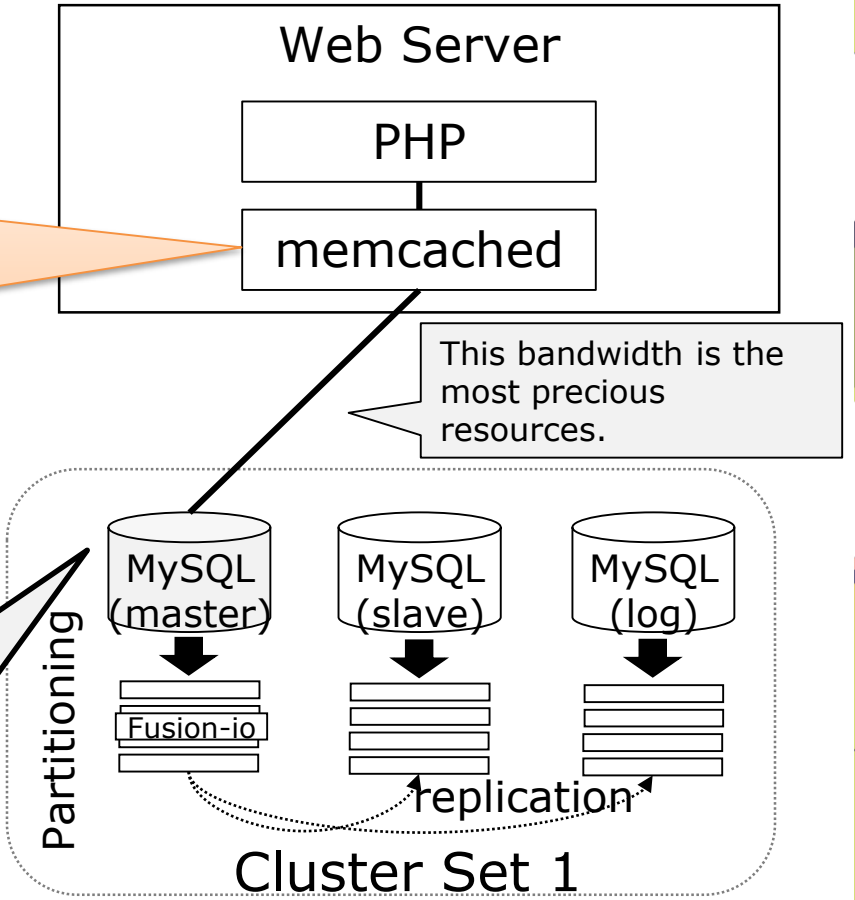
Typical Backend Architecture



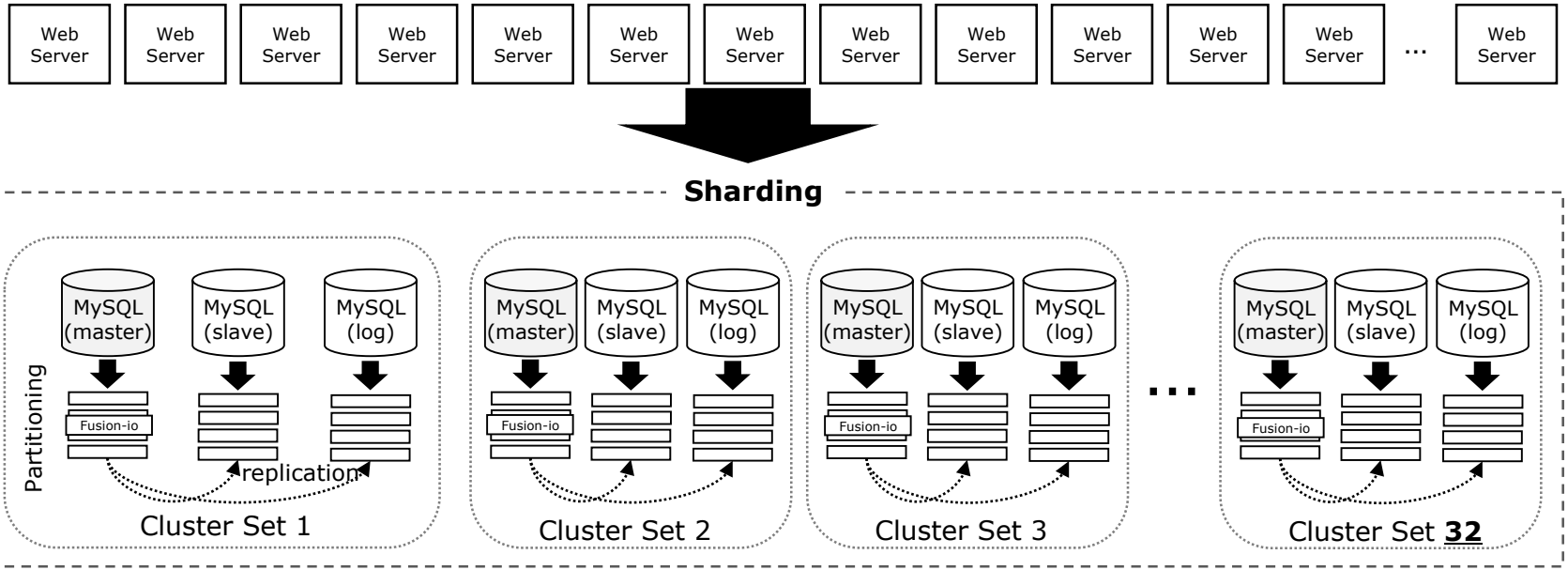
Primitive Unit

Main memory, memcached stores static data shared among all the PHP processes.

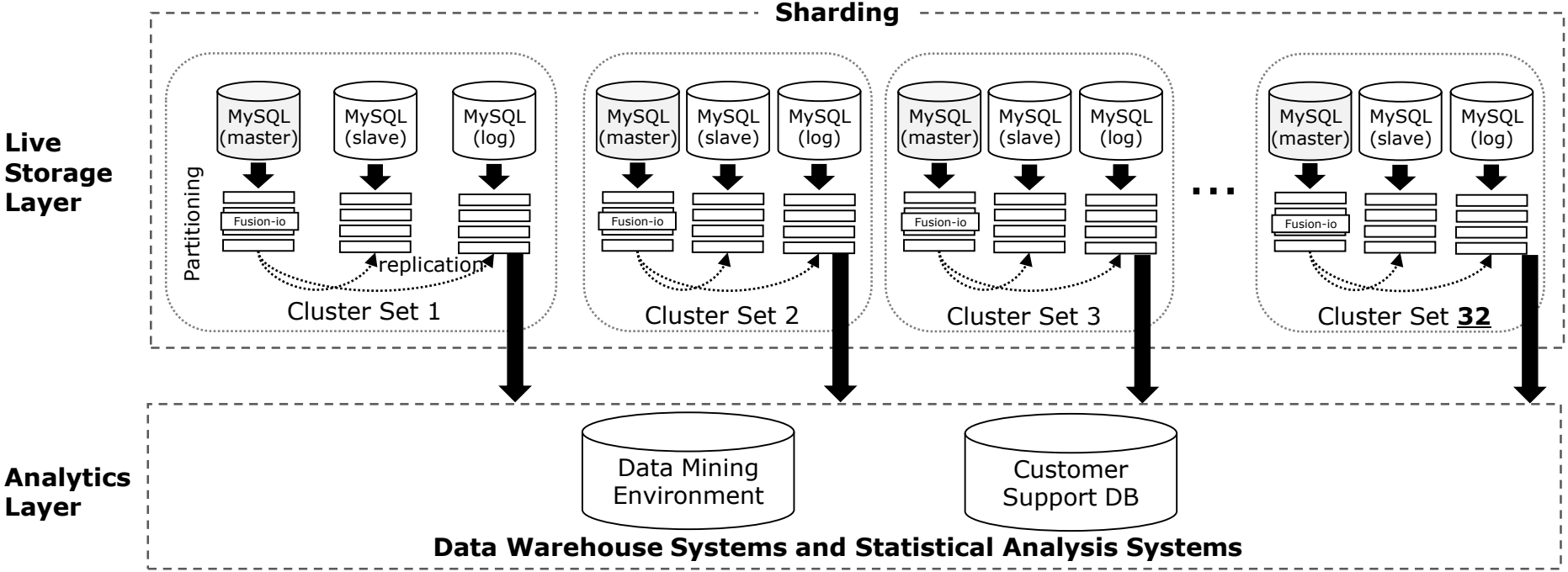
MySQL stores status data, which are updated in a real-time manner.



Combination of **sharding** (horizontal decomposition) and **partitioning** (vertical decomposition) brings high-level parallelism



Data mining process including JOIN operations is carried out by replica DBs that are replicated from the master database asynchronously.



MySQL is working now. But…

Operation can be highly inefficient and unprofitable

Not so scalable	Large maintenance cost	Capacity per machine is not so high

RDB functionalities are appropriate for C-A requirements, but it is too fat.

3 Overheads that decrease RDB's performance

Overhead-1: Too Generic Structure

Overhead-2: Not utilizing data access pattern in games

Overhead-3: Not utilizing modern hardware

Overhead-1: Too Generic Structure!

Data Model supported by Apps

External Schema-1

External Schema-2

External Schema-*n*

Data Model supported by DBMS

Conceptual Schema

```
CREATE TABLE(…);
```

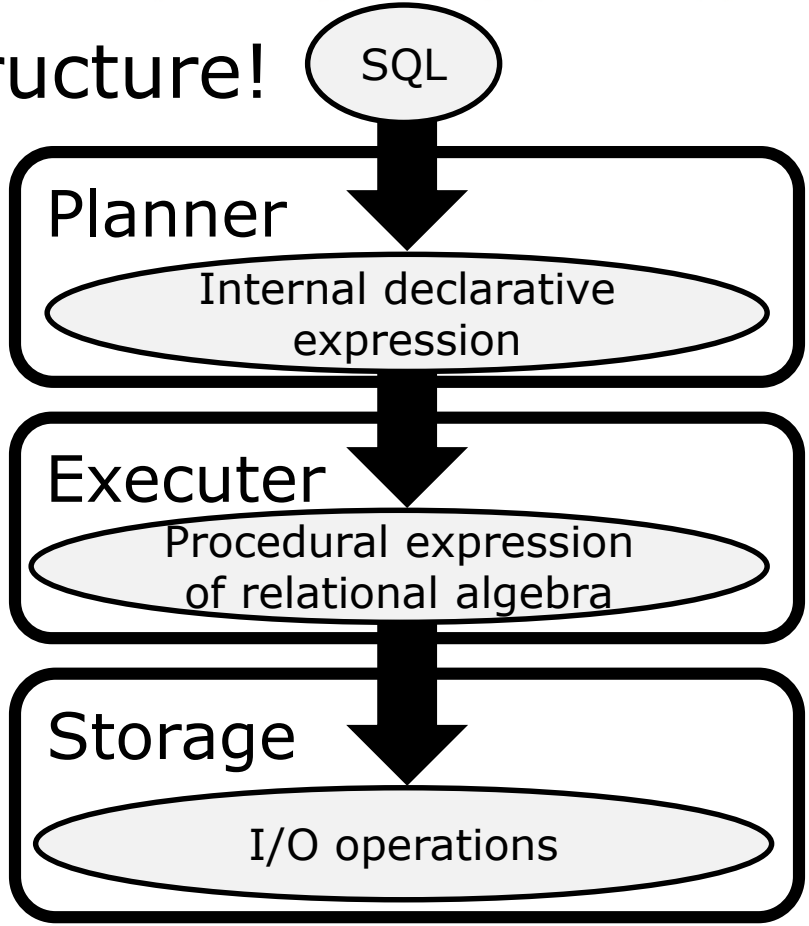
Data Model supported by storage

Internal Schema

```
TupleTableSlot *rec;
```

Overhead-1:Too Generic Structure!

- Planner (logical optimization)
  - Rewrites queries for better performance, by analyzing queries as relational calculus.
- Executor (physical optimization)
  - Executes query primitives such as relational algebras.
- Storage Subsystem
  - Reads and writes disk storages.



Let's resolve Overhead-2: Not utilizing data access pattern in games

	Insert	Update	Select	Real-time	Consistency	Query Type
Legacy Web (e-commerce)	Small	Small	Large	No	Strong Consistency	Dynamic (mutable)
SNS(Large Scale Web Apps)	Large	Small	Massively Large	No	Eventual Consistency	Dynamic (mutable)
IoT (Stream DB)	Massively Large	no	Small	Yes	Application-Dependent	Static (immutable)
Mobile Game	Small	Massively Large	Massively Large	Yes	Strong Consistency	Static (immutable)

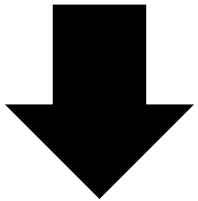
We faces technical requirements that are essentially different from a conventional DBs

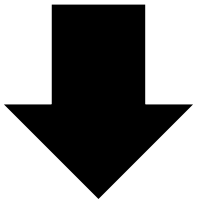
Let’s resolve Overhead-3: Not utilizing modern hardware

The throughput performance becomes from 10 to 20 times faster, due to the multi-core CPU with the high parallelism and a SSD with the highly parallel access.

Solution

Query Pre-Compilation
(Immutable Query)



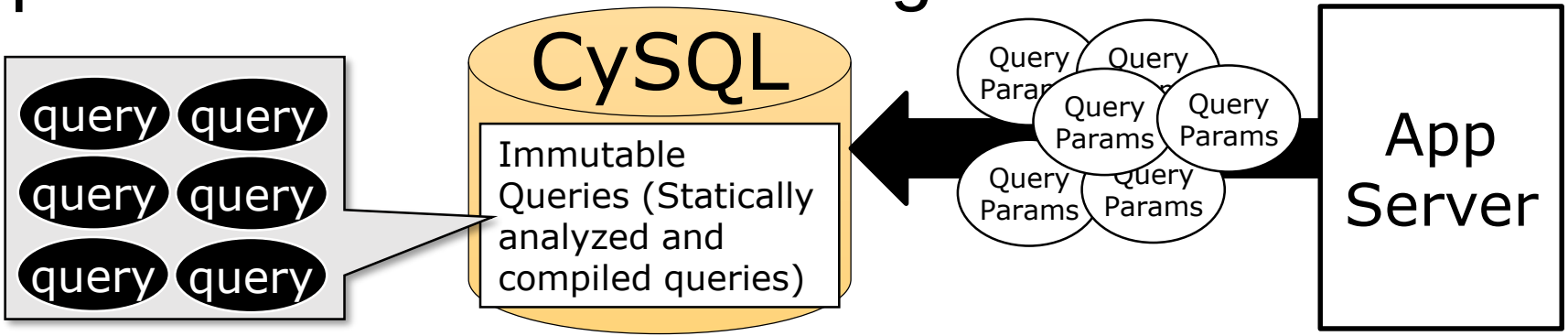


Lower Footprint query processing

Lock-Free Thread Scheduling

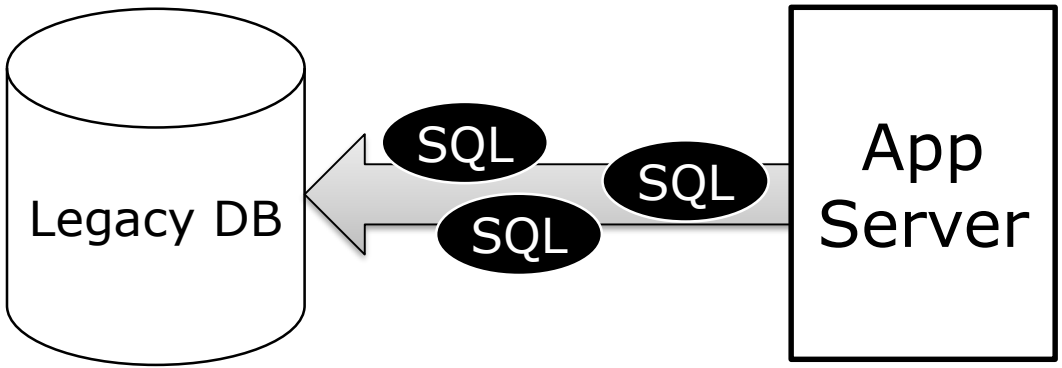
Static Query Analysis is effective

If we can know all the query to be processed, we can apply holistic optimization by analyzing their data access pattern and implicit race condition among them.



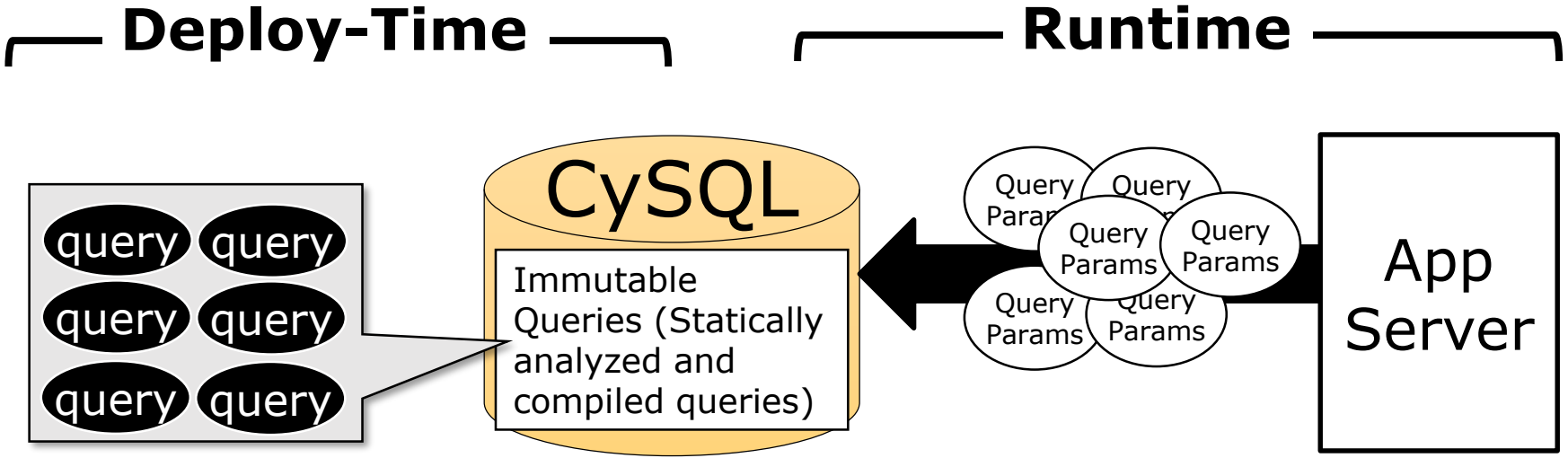
Legacy Query Model

- Legacy query model cannot optimize holistic query, because queries are submitted dynamically.

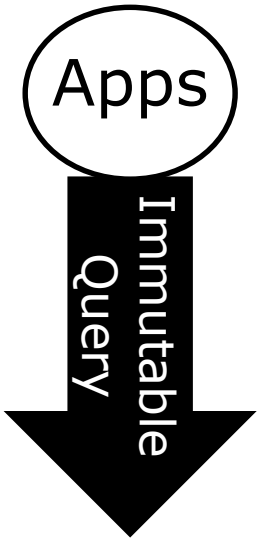


Immutable Query

Immutable query model requires the all queries to be defined at deploy time



SQL parsing and optimizing takes large cost



Controlling transaction manager directly from the application, we can achieve nearly 10 times faster performance than SQL

Planner

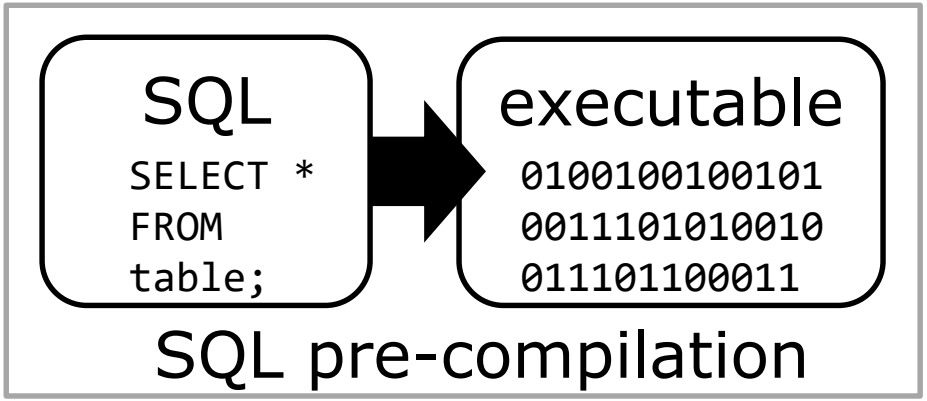
Executer

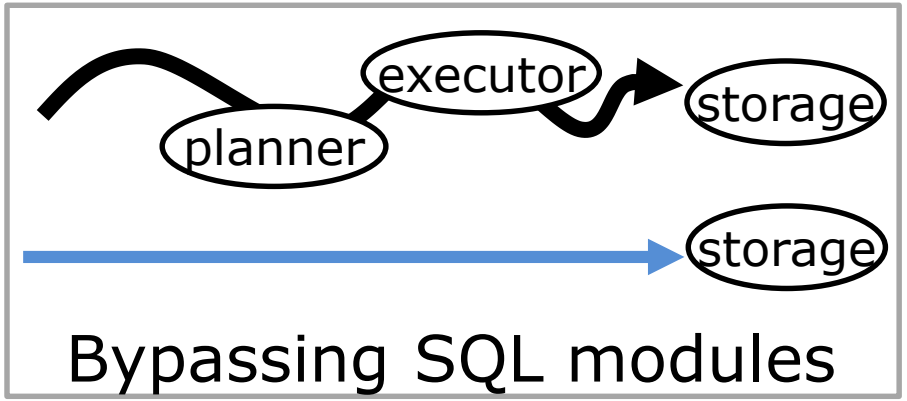
We can bypass the planner and the executor, by compiling SQL into executable machine code.

Storage

Query Processor Bypassing

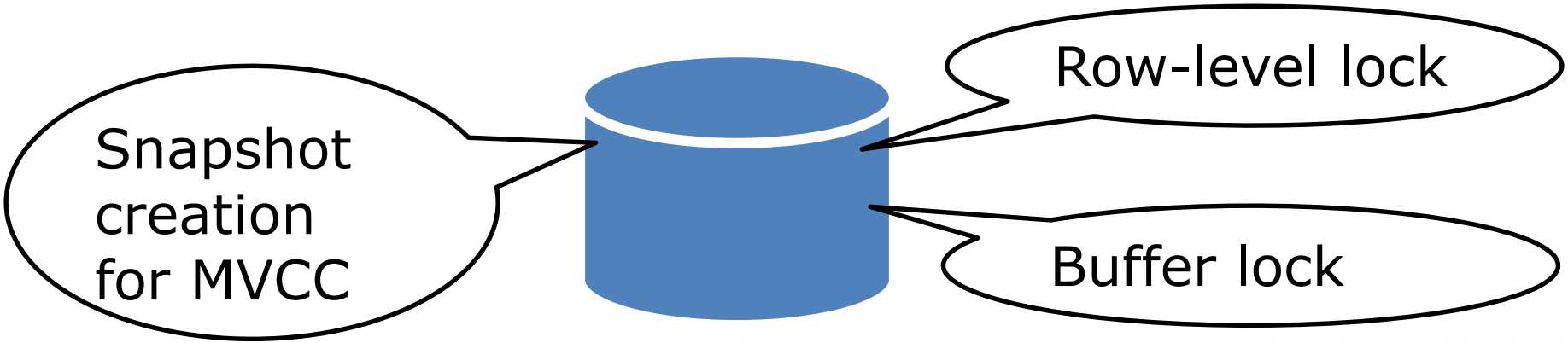
Introducing query pre-compilation and bypassing planner and executor





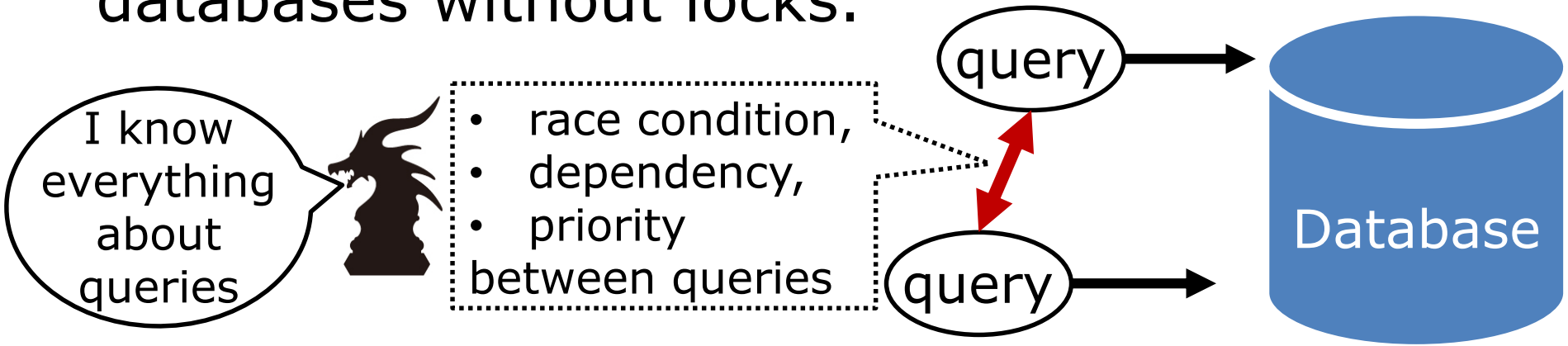
Lock-Free Transaction Processing

Legacy RDB embraces a lot of overhead of sync inside the DB engine for lock mechanism. Each of those overheads is small, but small overhead will be accumulated, so each overhead is the cost which can't be ignored.

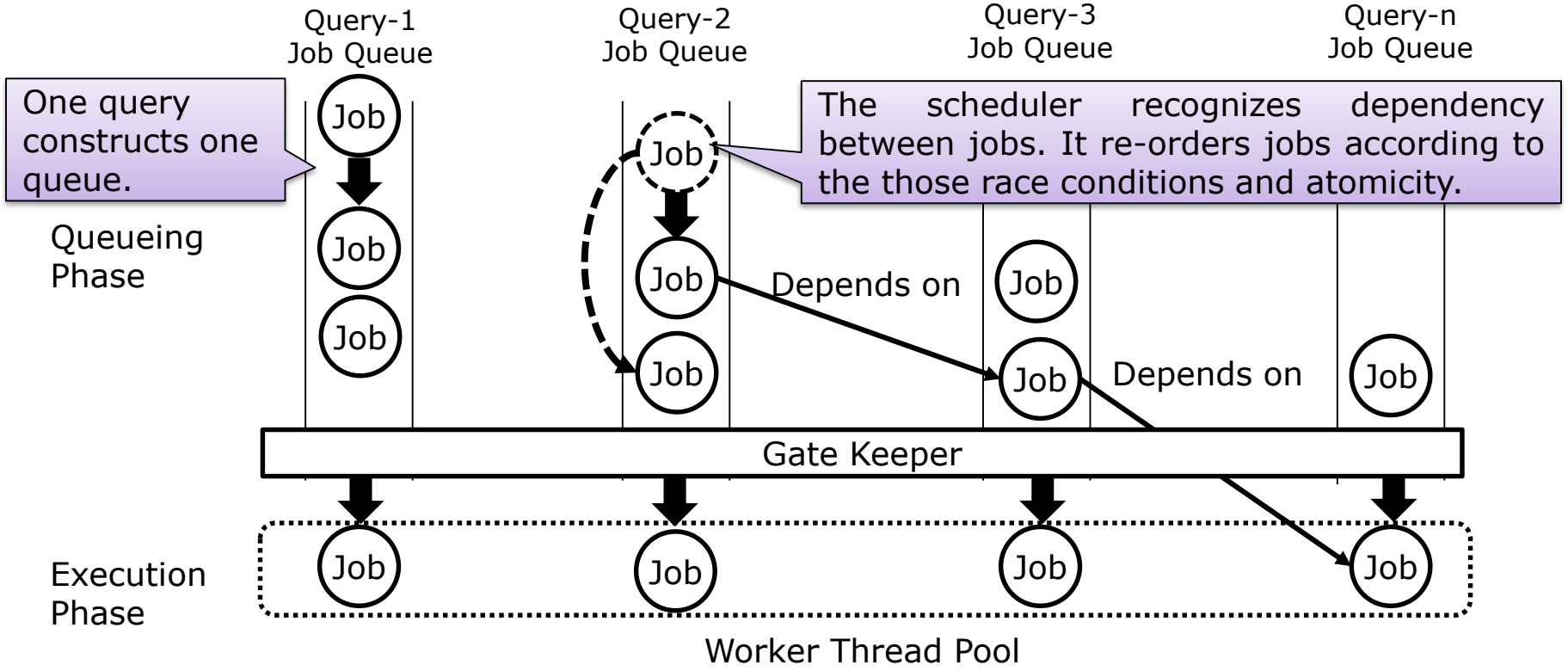


Can we realize Lock-Free?

By using the thread scheduling specialized in the data access pattern specific to the game, multiple threads read/writes databases without locks.



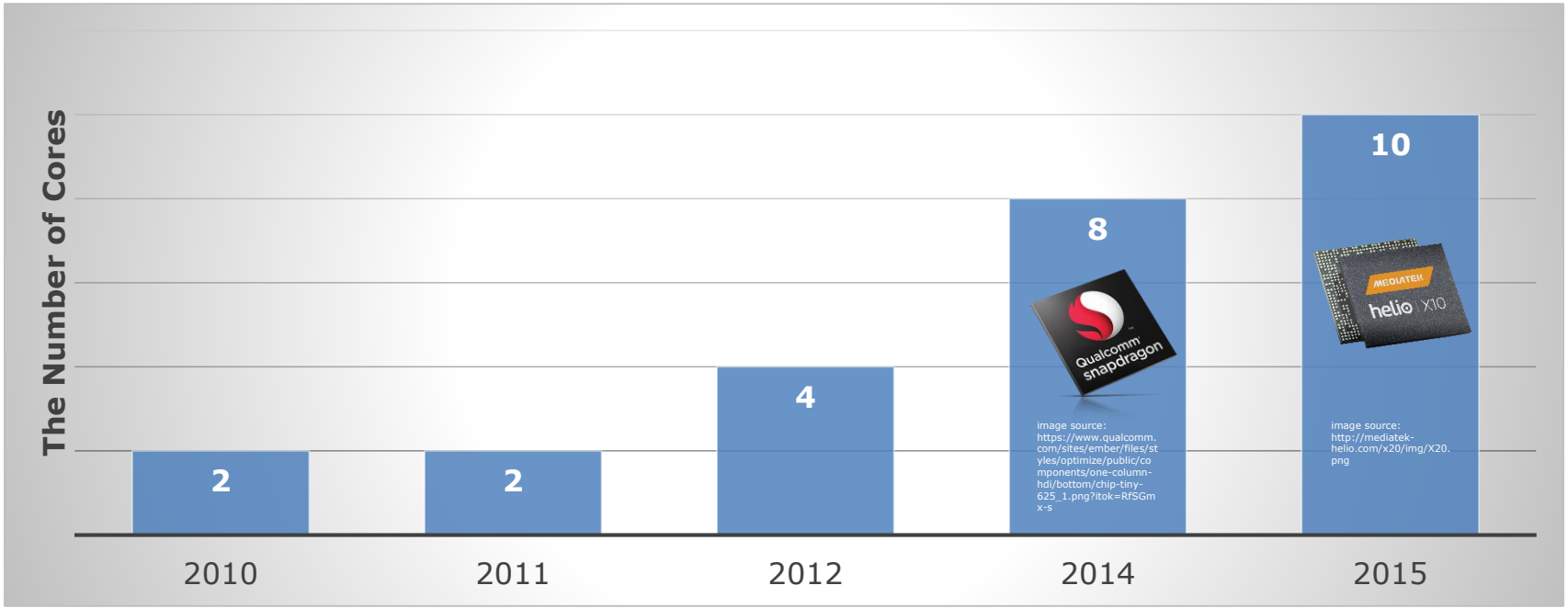
Basic Concept of Lock-Free Thread Scheduling



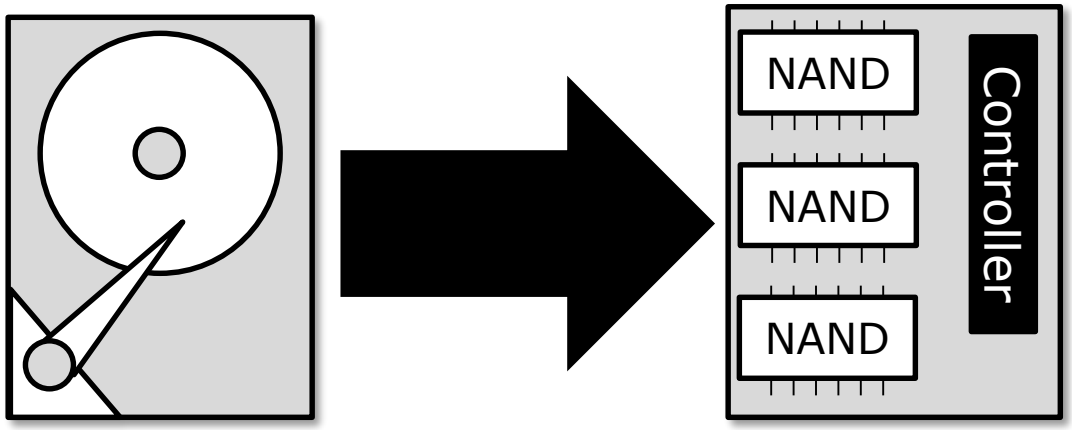
In the immutable query model, DB engine can schedule threads perfectly without synchronization lock.

- Because the DB engine can analyze race condition between queries statically.
- The thread scheduling strategy is easy: “Do not schedule the queries that cannot be executed simultaneously.”

The number of CPU cores are increasing for both energy and performance efficiency.



Parallelism in I/O Devices



HDD

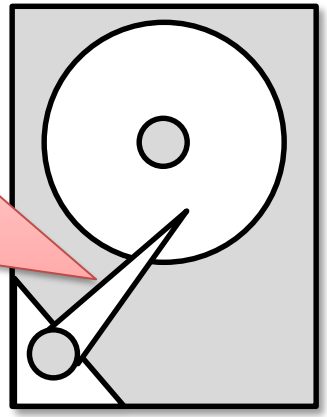
SSD

Not so parallel

Highly Parallel

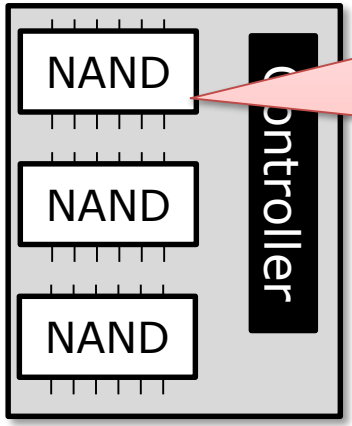
Parallelism in I/O Devices

Only this parts (head) can read/write data



HDD

Not so parallel

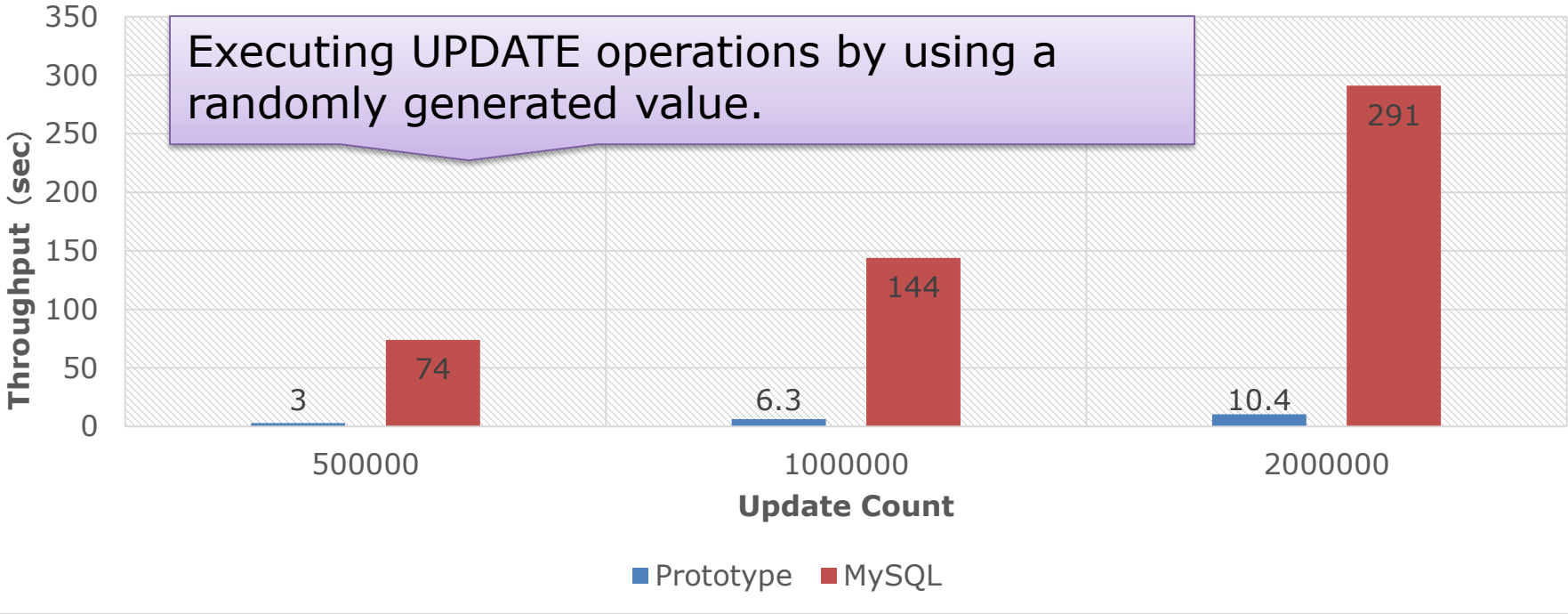


SSD

Highly Parallel

Every chip can read/write data in parallel

Comparison of the prototype and MySQL: UPDATE command



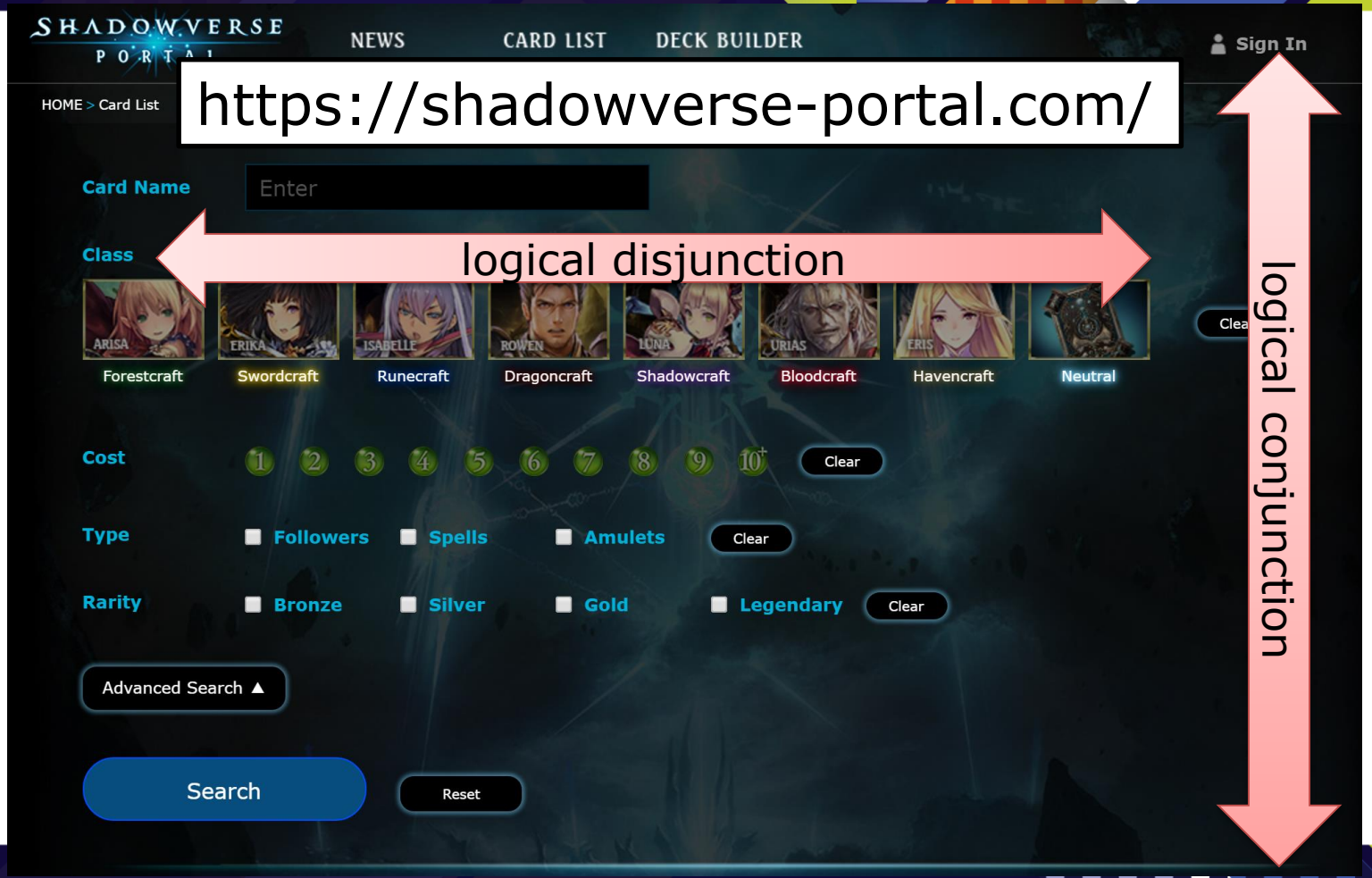
Theoretical Conclusion

Increase the number of threads as many as you can

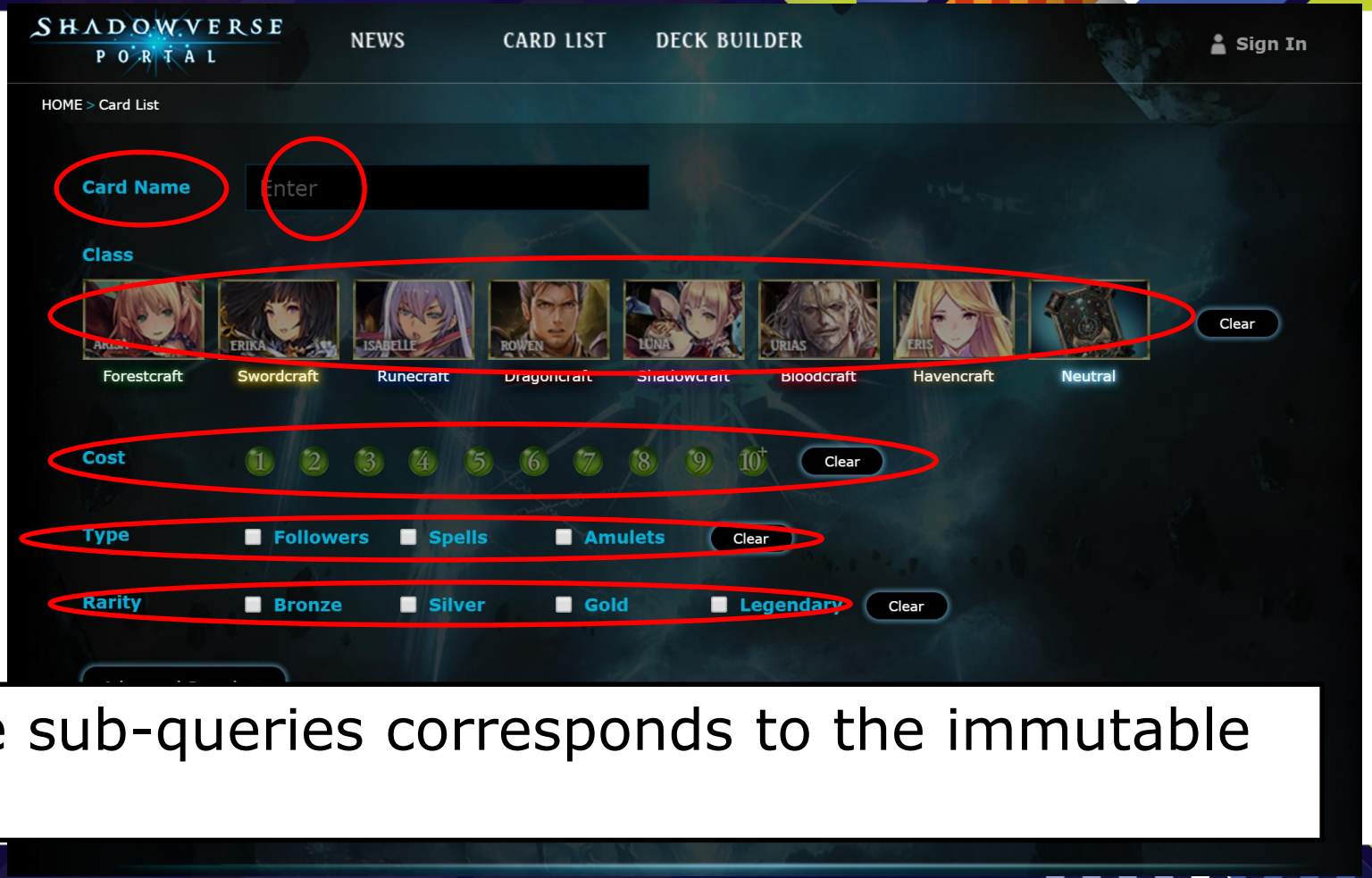
Implementation!

- Here I introduce a quick implementation method of read-only database engine.

Use Case: Fast Search Engine

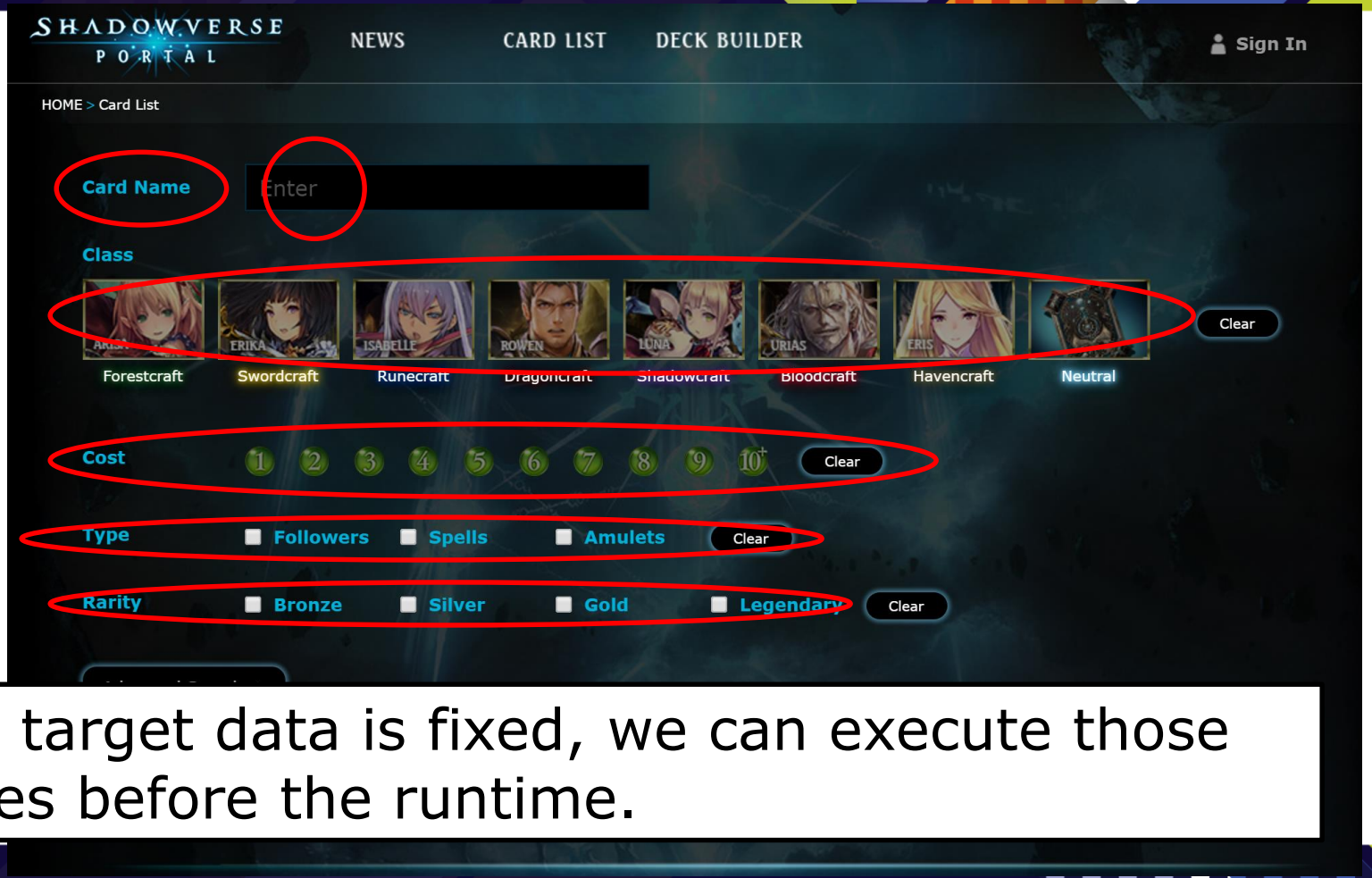


Query type detection



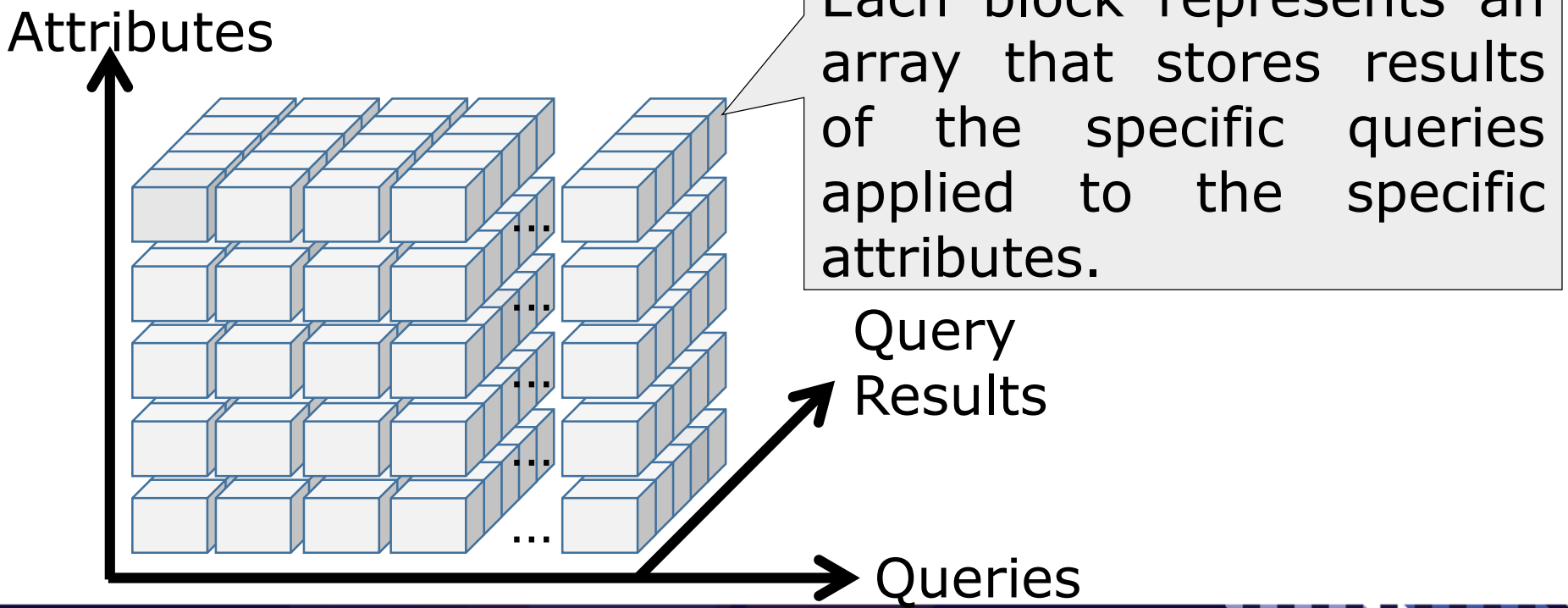
Those are sub-queries corresponds to the immutable queries

Query type detection



When the target data is fixed, we can execute those sub queries before the runtime.

Data Structure



Conclusion

- You can implement tiny RDB within a hour.

- Increase the number of threads as many as you can

Recommended Books

- Database Systems: The Complete Book (2nd Edition) 2nd Edition by Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
- Introduction to Algorithms, 3rd Edition (MIT Press) 3rd Edition by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein