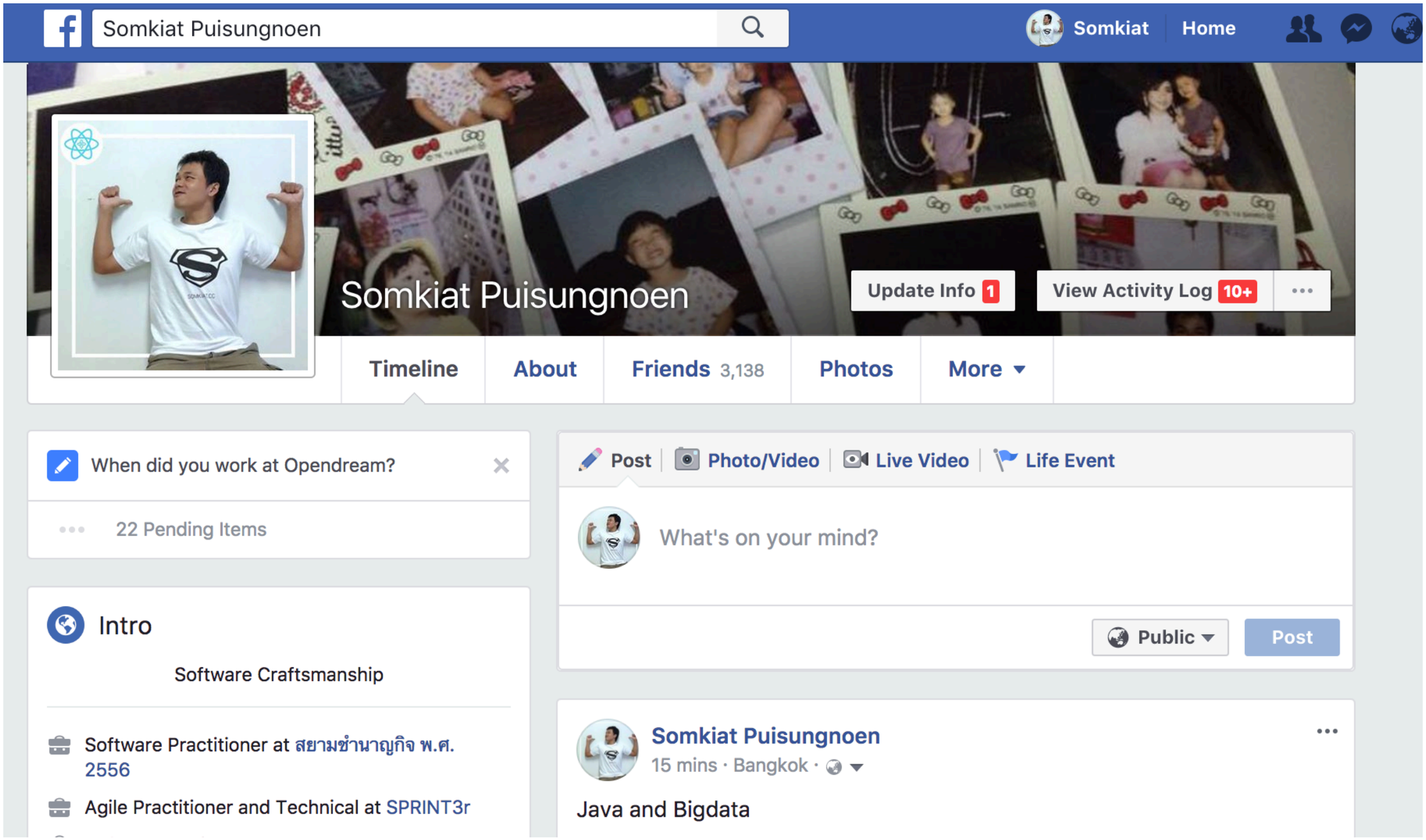
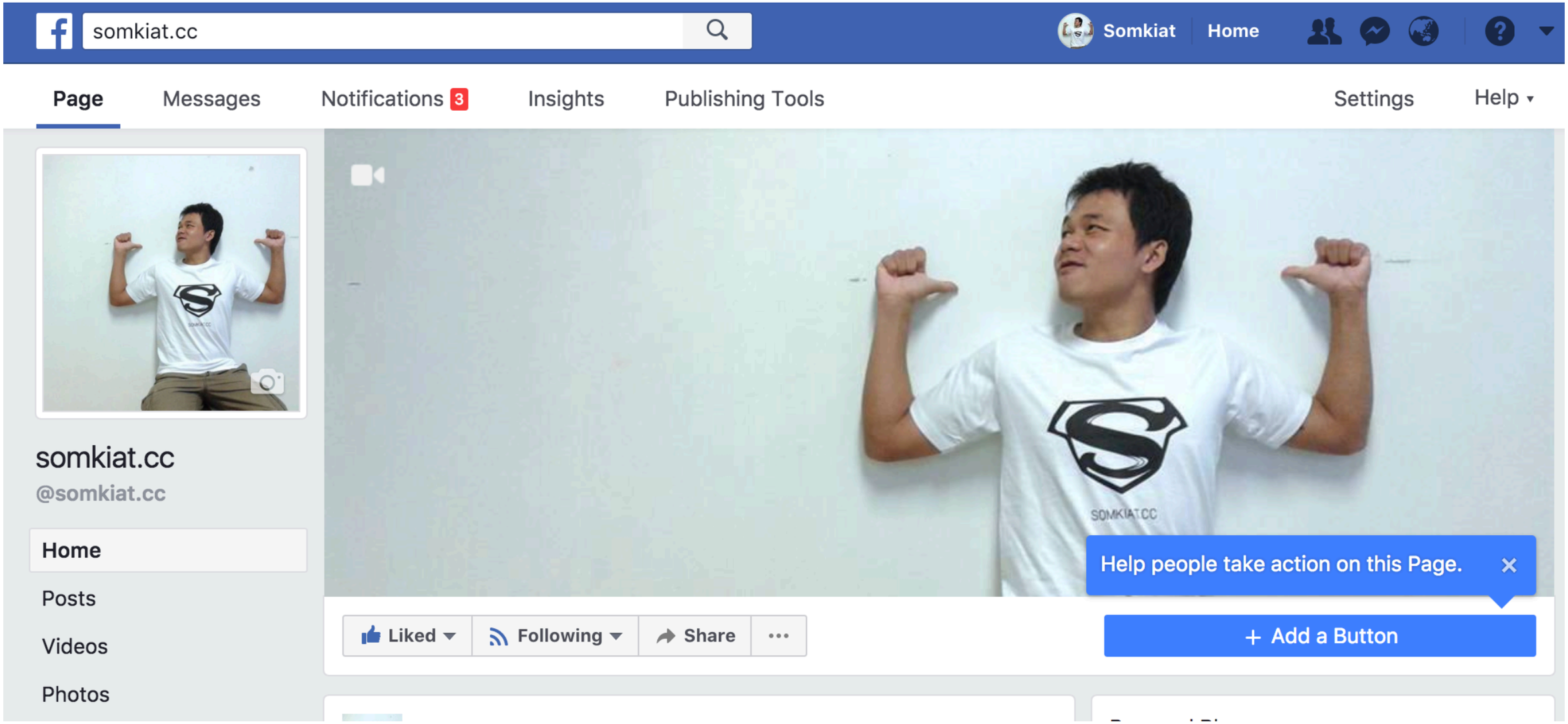
สยามชำนาญกิจ
SIAM CHAMNAN KIT

SQL vs NoSQL





Agenda

- Database model
- RDBMS
- SQL vs. NoSQL
- Workshop
  - Redis
  - PostgreSQL
- SQL tuning with PostgreSQL

Database ?

Database = RDBMS ?

Relation **D**ata**B**ase **M**anagement **S**ystem

RDBMS

Create a protective wrapper around data

Give data a pace to live & control modification

Secure data by enforce access permission

Manage system performance for read/write data

Relational = Database model

Table

customer_id	fname	lname	last_purchase_date	status_level
1234	Alice	Smith	23-Jan-2016	silver
2345	Bob	Washington	14-July-2018	bronze
2352	Charlie	Johnson	20-Dec-2018	gold
2678	Chendong	Mao	19-Dec-2017	gold
3422	Javier	Valdez	23-Dec-2018	gold
3577	Avery	Winston	5-Jan-2019	silver
4240	Charles	Davis	15-Jan-2017	bronze
6235	Vihaan	Patel	12-Nov-2018	gold
2352	Katherine	Coltrane	04-Apr-2016	silver

Database Models

Rank			DBMS	Database Model
Dec 2020	Nov 2020	Dec 2019		
1.	1.	1.	Oracle	Relational, Multi-model
2.	2.	2.	MySQL	Relational, Multi-model
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model
4.	4.	4.	PostgreSQL	Relational, Multi-model
5.	5.	5.	MongoDB	Document, Multi-model
6.	6.	6.	IBM Db2	Relational, Multi-model
7.	7.	↑ 8.	Redis	Key-value, Multi-model
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model
9.	9.	↑ 11.	SQLite	Relational
10.	10.	10.	Cassandra	Wide column
11.	11.	↓ 9.	Microsoft Access	Relational
12.	12.	↑ 13.	MariaDB	Relational, Multi-model
13.	13.	↓ 12.	Splunk	Search engine
14.	14.	↑ 15.	Teradata	Relational, Multi-model
15.	15.	↓ 14.	Hive	Relational

https://db-engines.com/en/ranking

Relational

Rank Dec 2020	Rank Nov 2020	Rank Dec 2019	DBMS	Database Model
1.	1.	1.	Oracle	Relational, Multi-model
2.	2.	2.	MySQL	Relational, Multi-model
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model
4.	4.	4.	PostgreSQL	Relational, Multi-model
5.	5.	5.	MongoDB	Document, Multi-model
6.	6.	6.	IBM Db2	Relational, Multi-model
7.	7.	↑ 8.	Redis	Key-value, Multi-model
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model
9.	9.	↑ 11.	SQLite	Relational
10.	10.	10.	Cassandra	Wide column
11.	11.	↓ 9.	Microsoft Access	Relational
12.	12.	↑ 13.	MariaDB	Relational, Multi-model
13.	13.	↓ 12.	Splunk	Search engine
14.	14.	↑ 15.	Teradata	Relational, Multi-model
15.	15.	↓ 14.	Hive	Relational

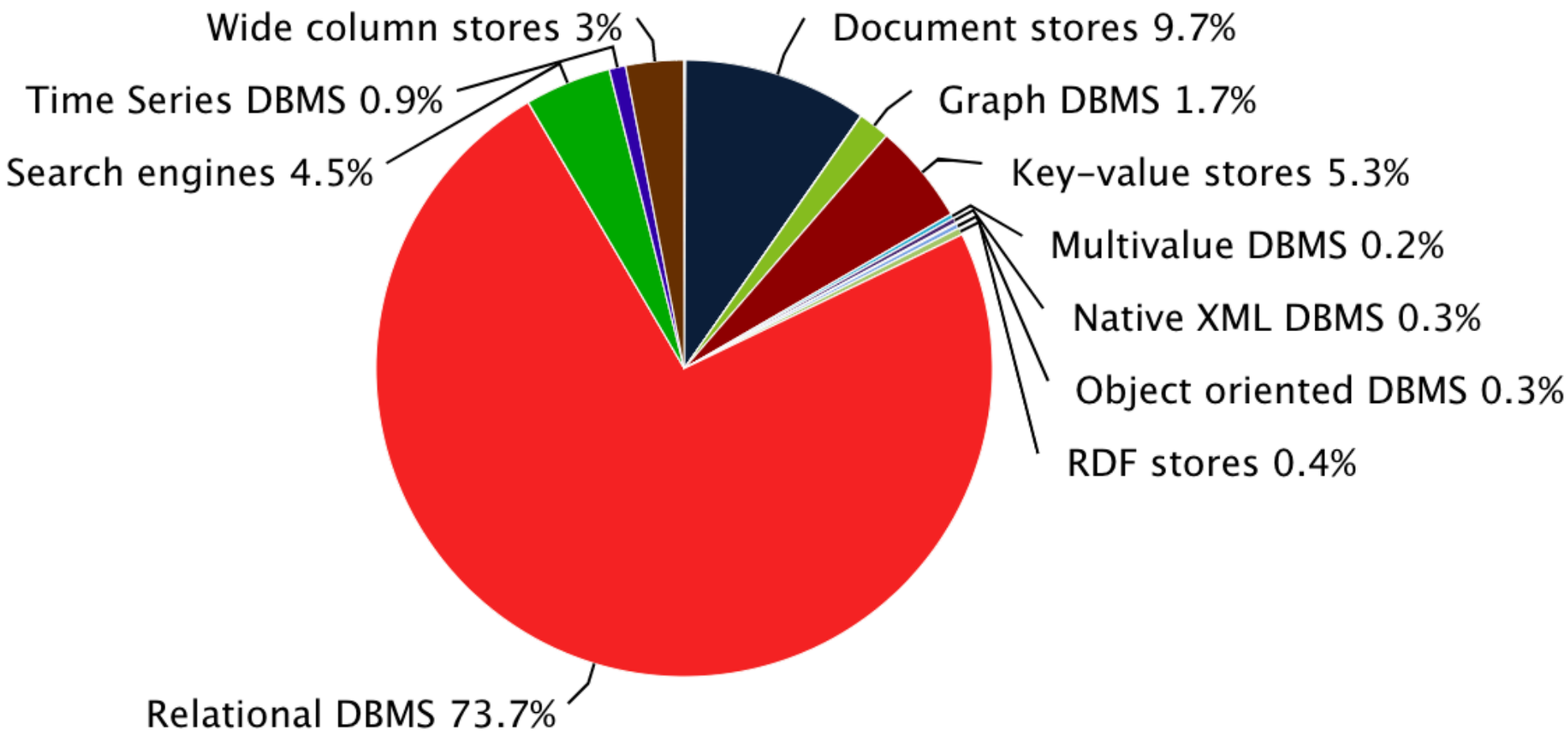
https://db-engines.com/en/ranking

Other database models

Rank Dec 2020	Rank Nov 2020	Rank Dec 2019	DBMS	Database Model
1.	1.	1.	Oracle	Relational, Multi-model
2.	2.	2.	MySQL	Relational, Multi-model
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model
4.	4.	4.	PostgreSQL	Relational, Multi-model
5.	5.	5.	MongoDB	Document, Multi-model
6.	6.	6.	IBM Db2	Relational, Multi-model
7.	7.	↑ 8.	Redis	Key-value, Multi-model
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model
9.	9.	↑ 11.	SQLite	Relational
10.	10.	10.	Cassandra	Wide column
11.	11.	↓ 9.	Microsoft Access	Relational
12.	12.	↑ 13.	MariaDB	Relational, Multi-model
13.	13.	↓ 12.	Splunk	Search engine
14.	14.	↑ 15.	Teradata	Relational, Multi-model
15.	15.	↓ 14.	Hive	Relational

https://db-engines.com/en/ranking

Database models



https://db-engines.com/en/ranking

Why ?

ACID ?

Atomicity
Consistency
Isolation
Durability

SQL ?

Structure Query Language

SQL

SQL uses human-readable syntax

Used to access data in RDBMS

SQL

customer_id	fname	lname	last_purchase_date	status_level
1234	Alice	Smith	23-Jan-2016	silver
2345	Bob	Washington	14-July-2018	bronze
2352	Charlie	Johnson	20-Dec-2018	gold
2678	Chendong	Mao	19-Dec-2017	gold
3422	Javier	Valdez	23-Dec-2018	gold
3577	Avery	Winston	5-Jan-2019	silver
4240	Charles	Davis	15-Jan-2017	bronze
6235	Vihaan	Patel	12-Nov-2018	gold
2352	Katherine	Coltrane	04-Apr-2016	silver

```
SELECT *
FROM SOME_TABLE
WHERE 1=1
HAVING  ...
GROUP BY  ...
```

History

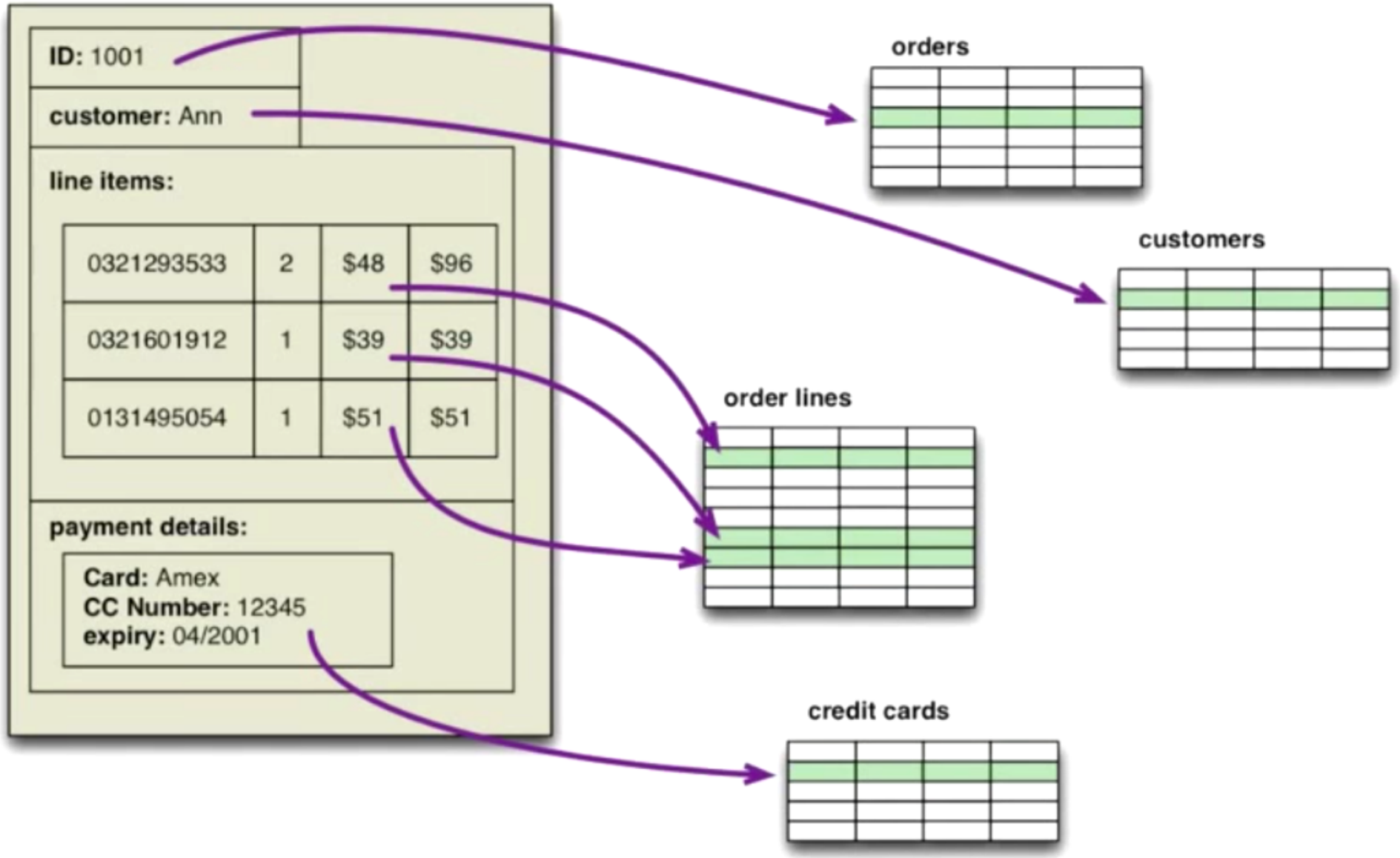
RDBMS

Persistence

Integration

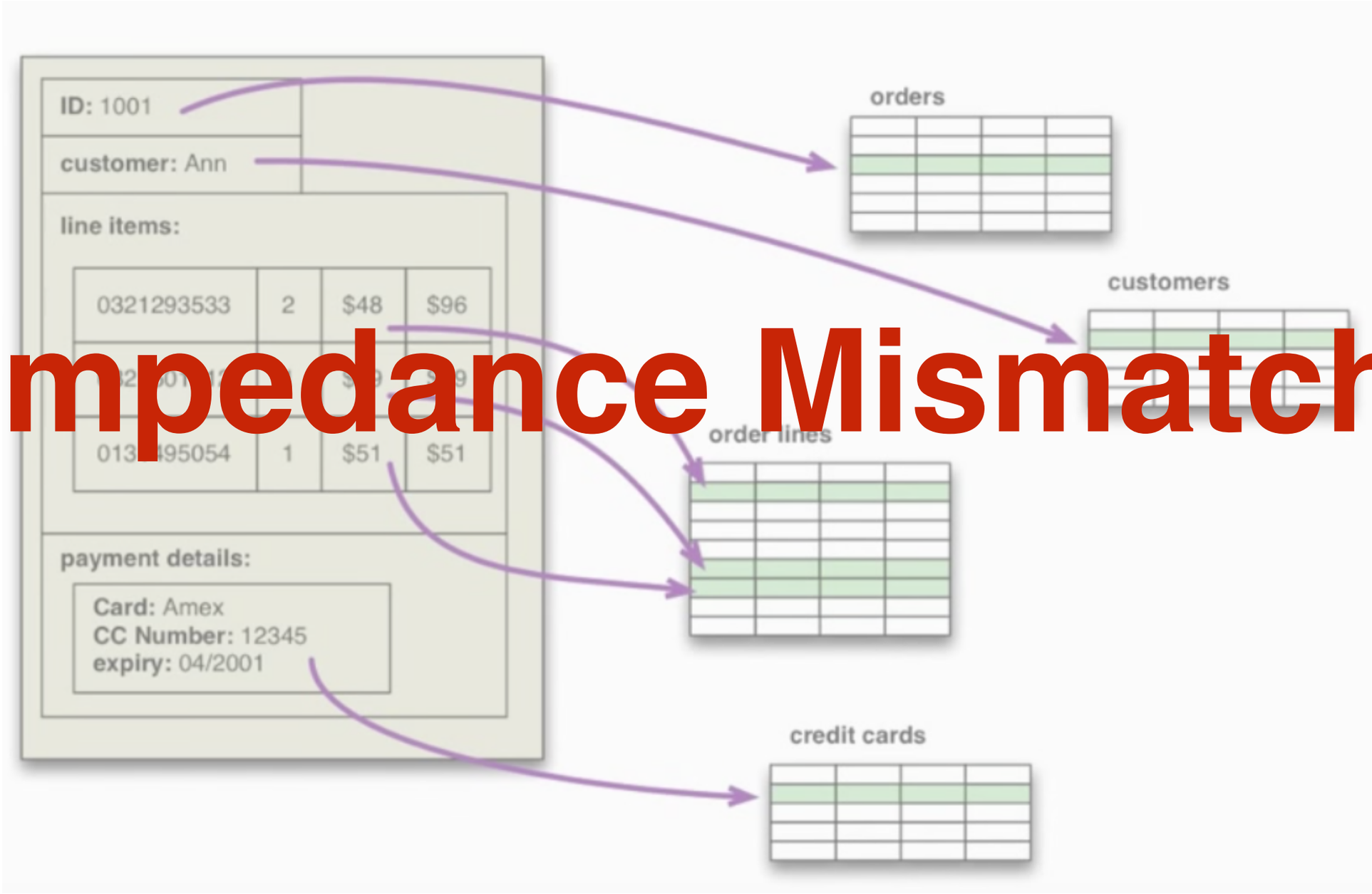
SQL

Transaction

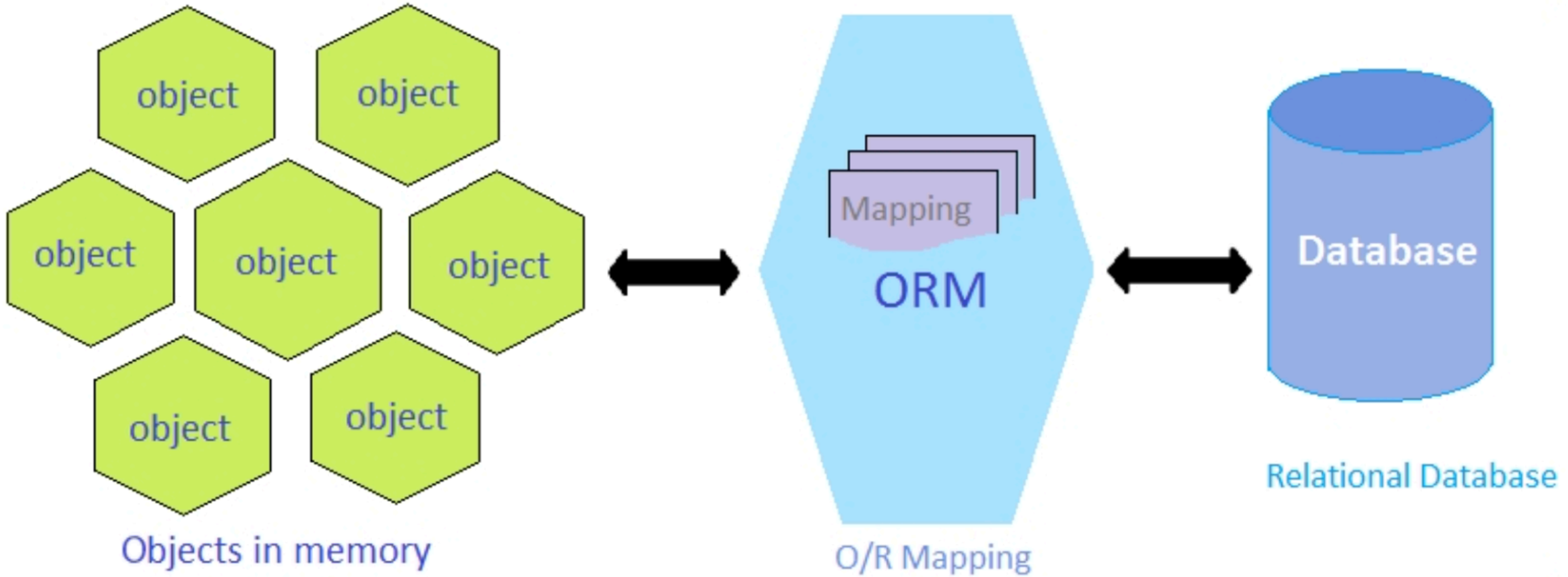


Object Oriented Programming

Impedance Mismatch



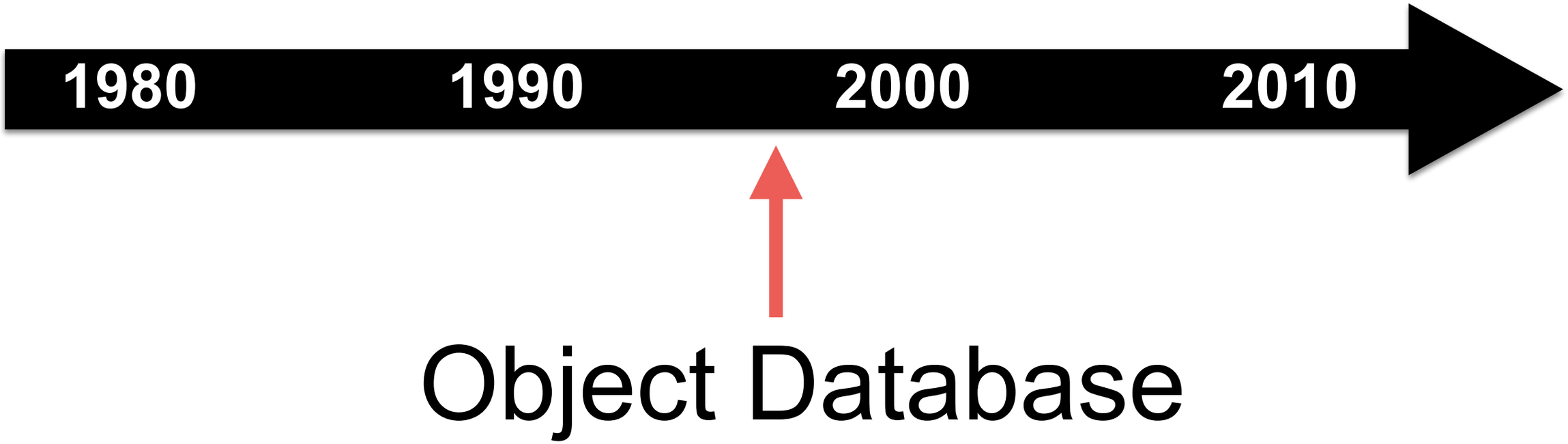
Object **R**elation **M**apping



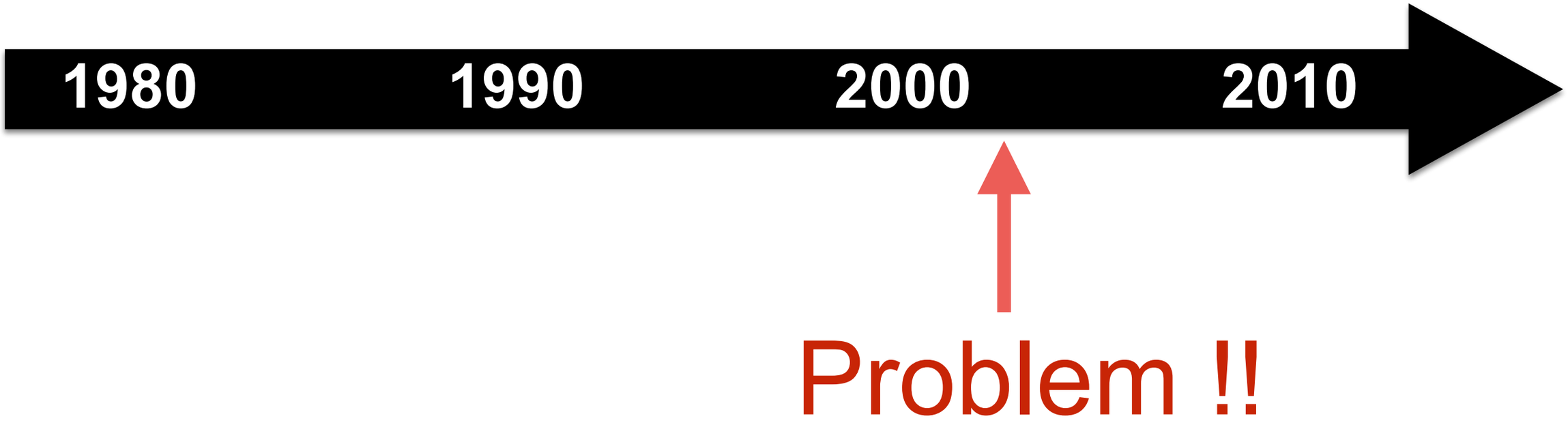
MyBatis

JPA

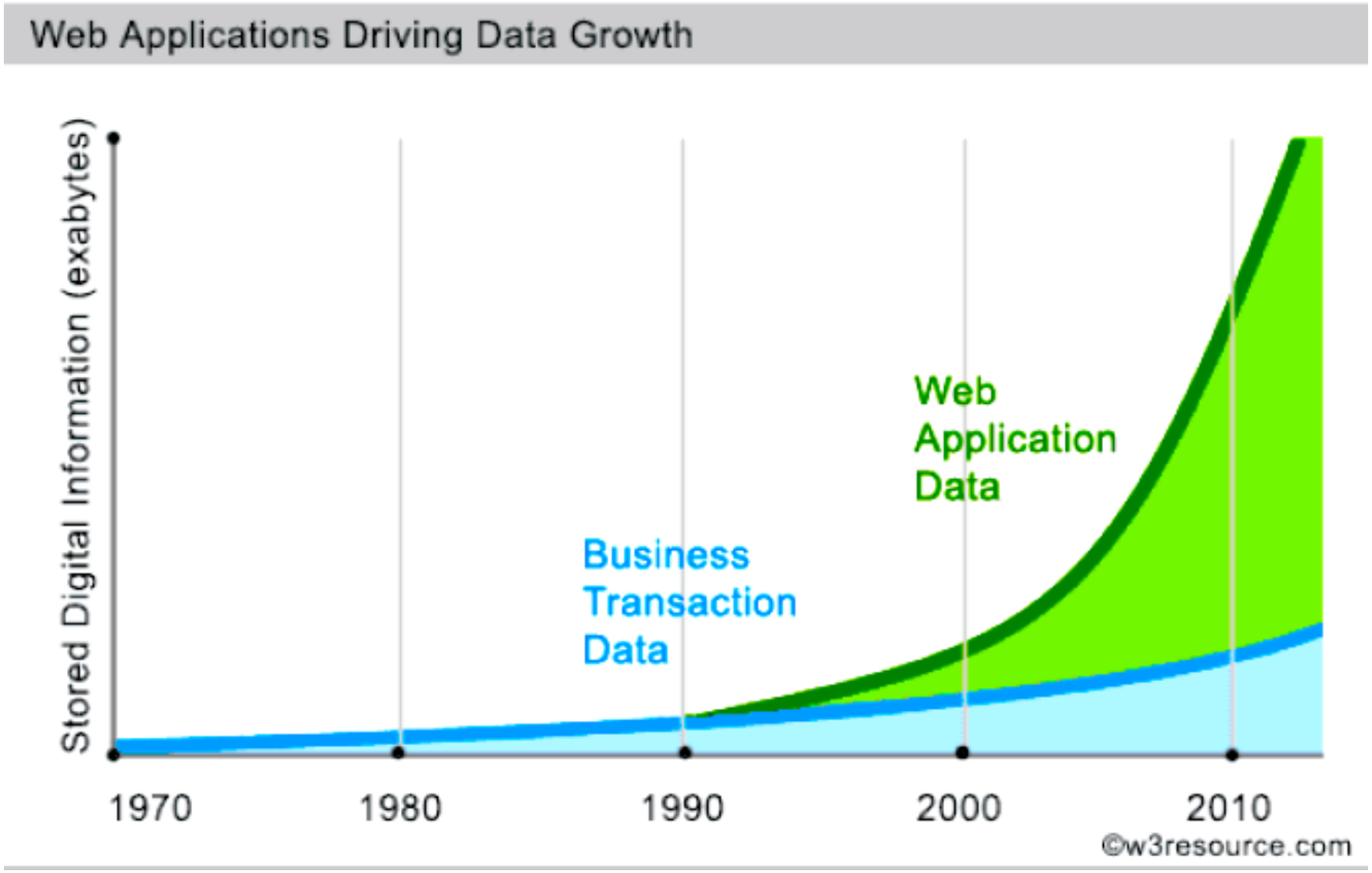
Java Persistence API



Back to RDBMS and ORM



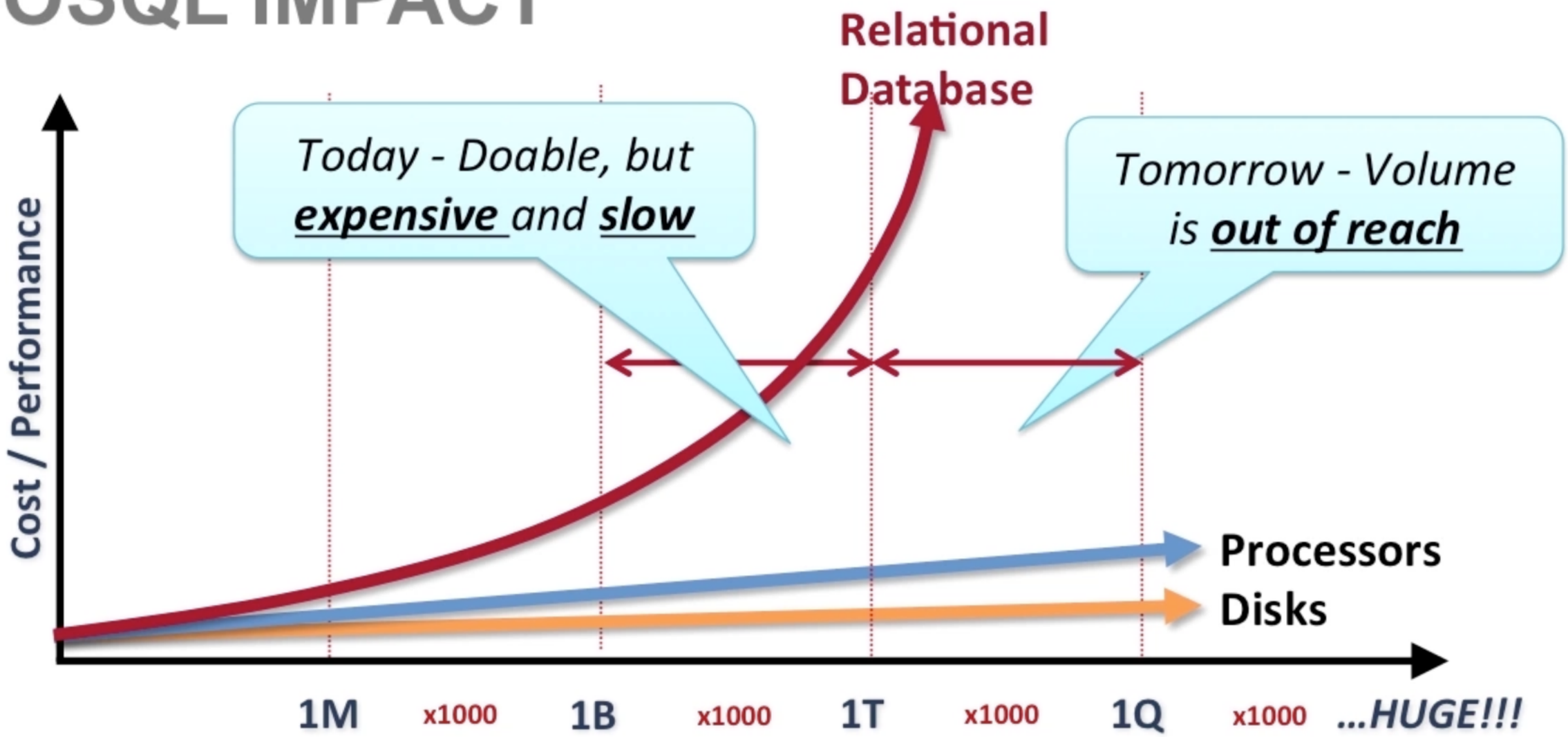
Web Application + Internet



Data challenges

Data volume
Data velocity
Data variety
Data valence

NOSQL IMPACT

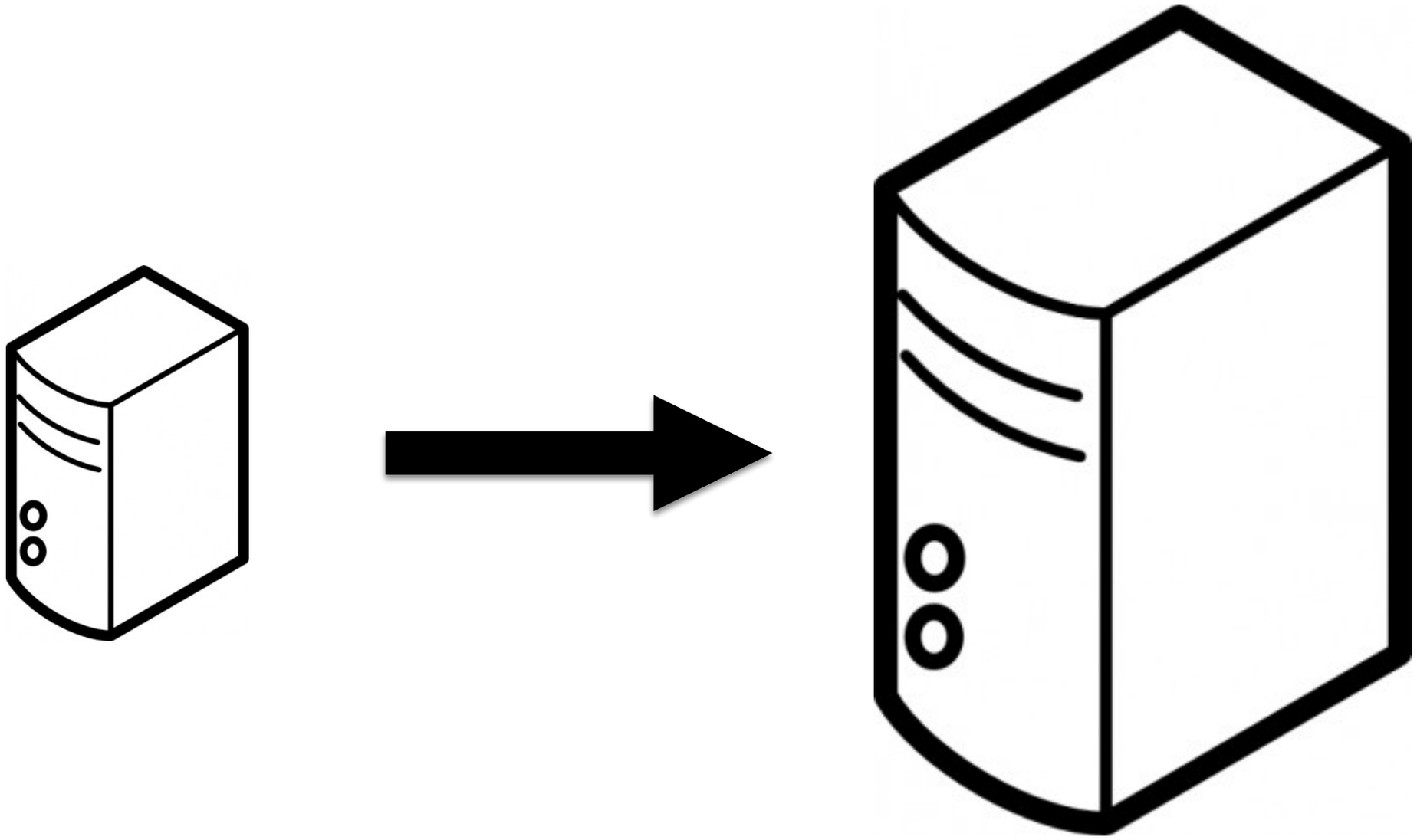


Requirements !!

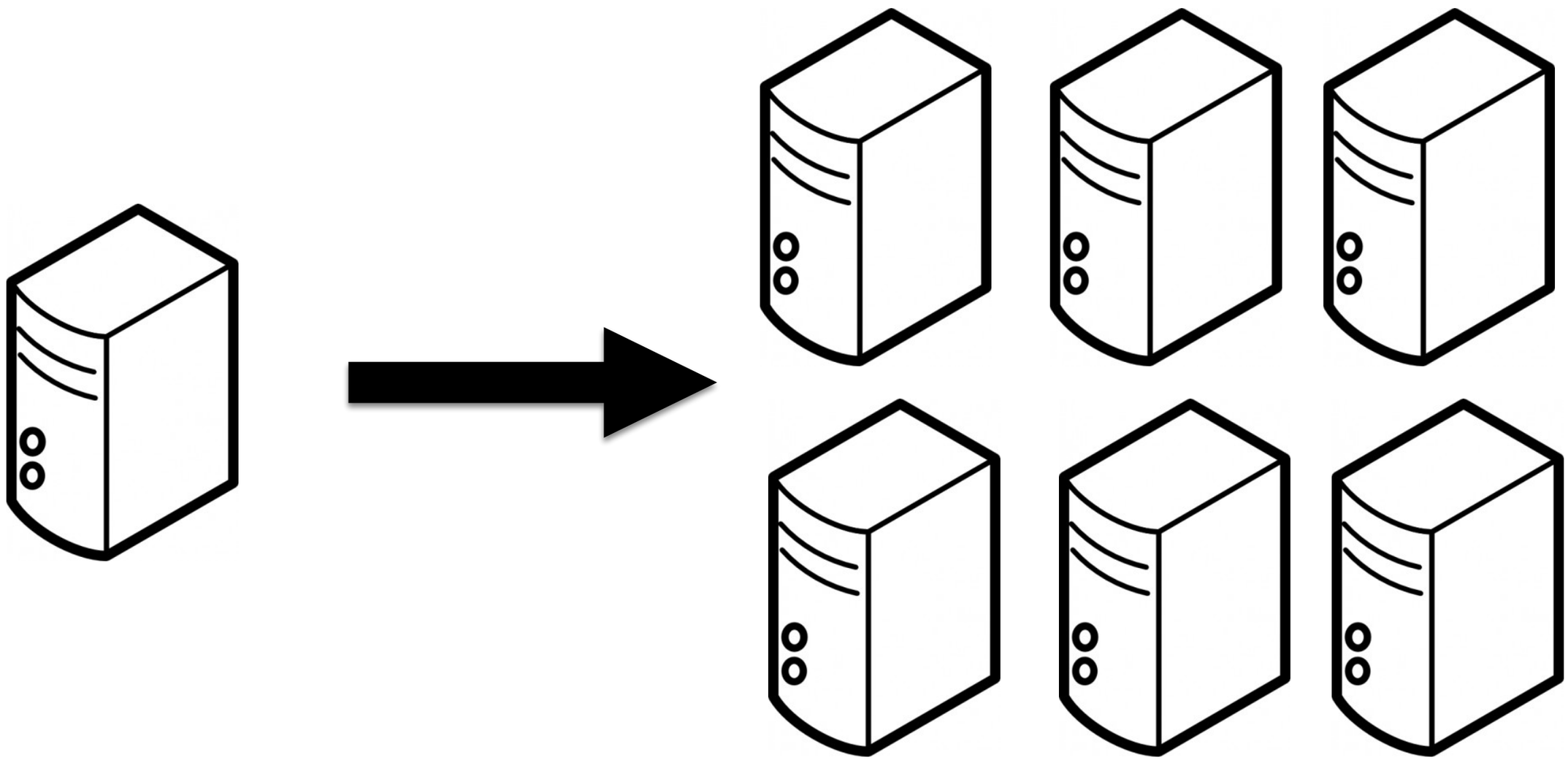
Profile Management	Caching	360 Degree Customer View	Internet of Things	Mobile Applications
Content Management	Catalog	Real Time Big Data	Digital Communication	Fraud Detection

How to scale RDBMS ?

Scale up

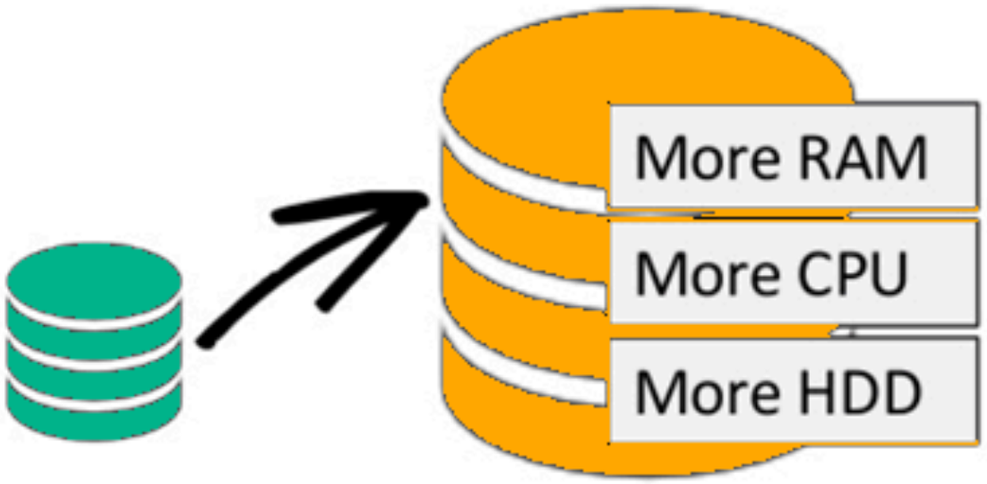


Scale out

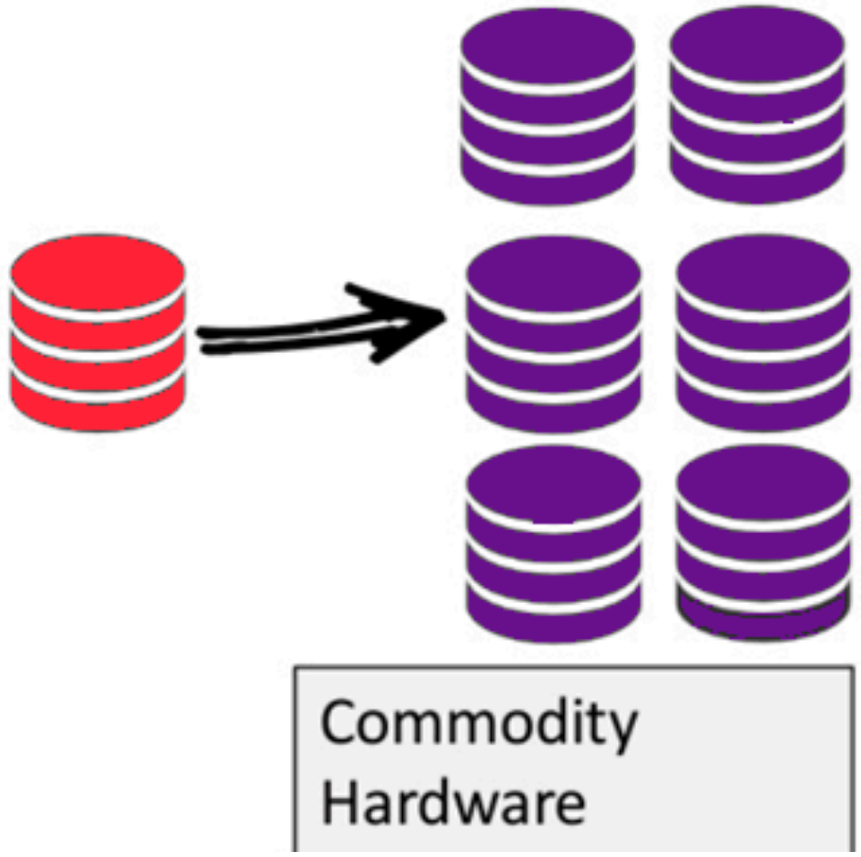


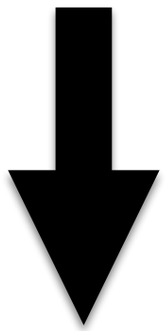
Scale up vs out ?

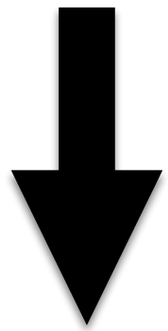
Scale-Up (*vertical* scaling):



Scale-Out (*horizontal* scaling):

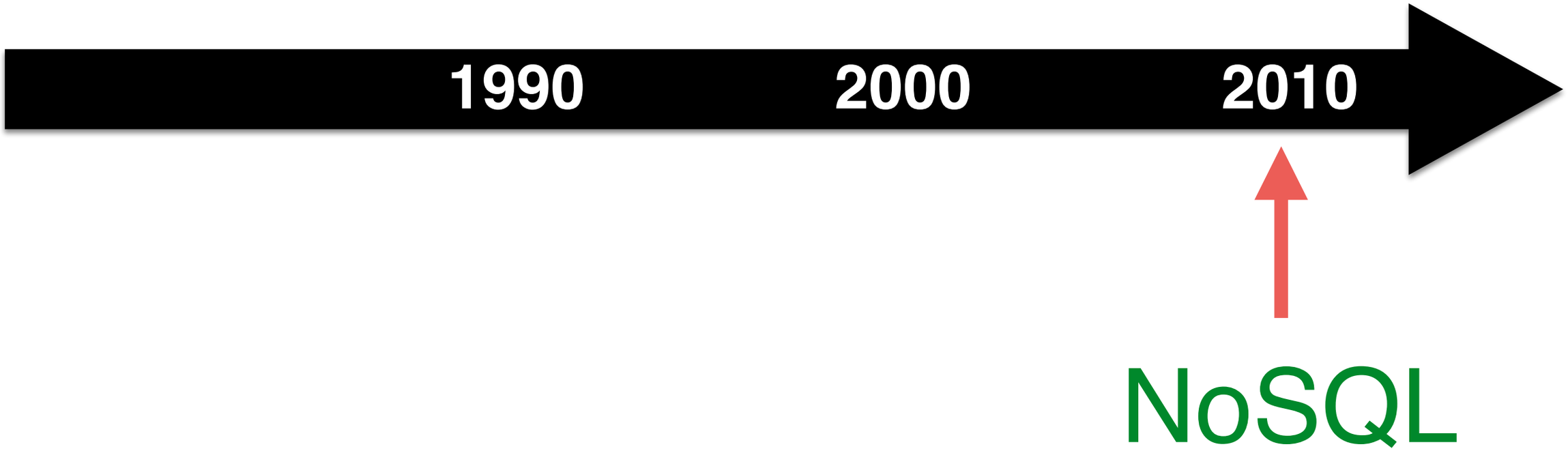






Bigtable

Dynamo



NoSQL

No SQL

No No SQL

Not Only SQL

Non-Relational

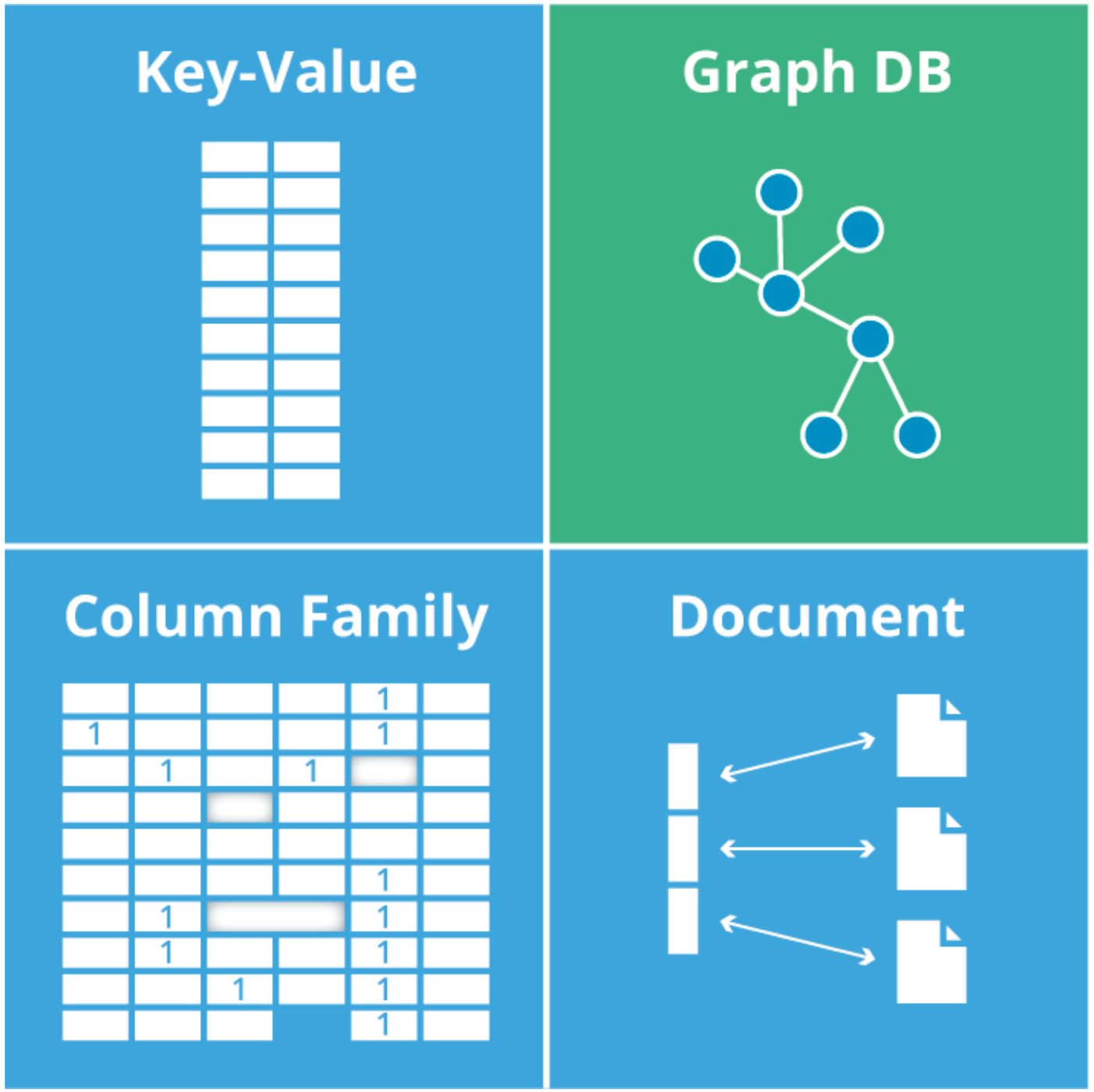
NoSQL properties

Non-relational
Opensource
Cluster friendly
Schema-less

For new web app

Data models

Key-value
Document
Column
Graph
Search
Time-series



Column Family

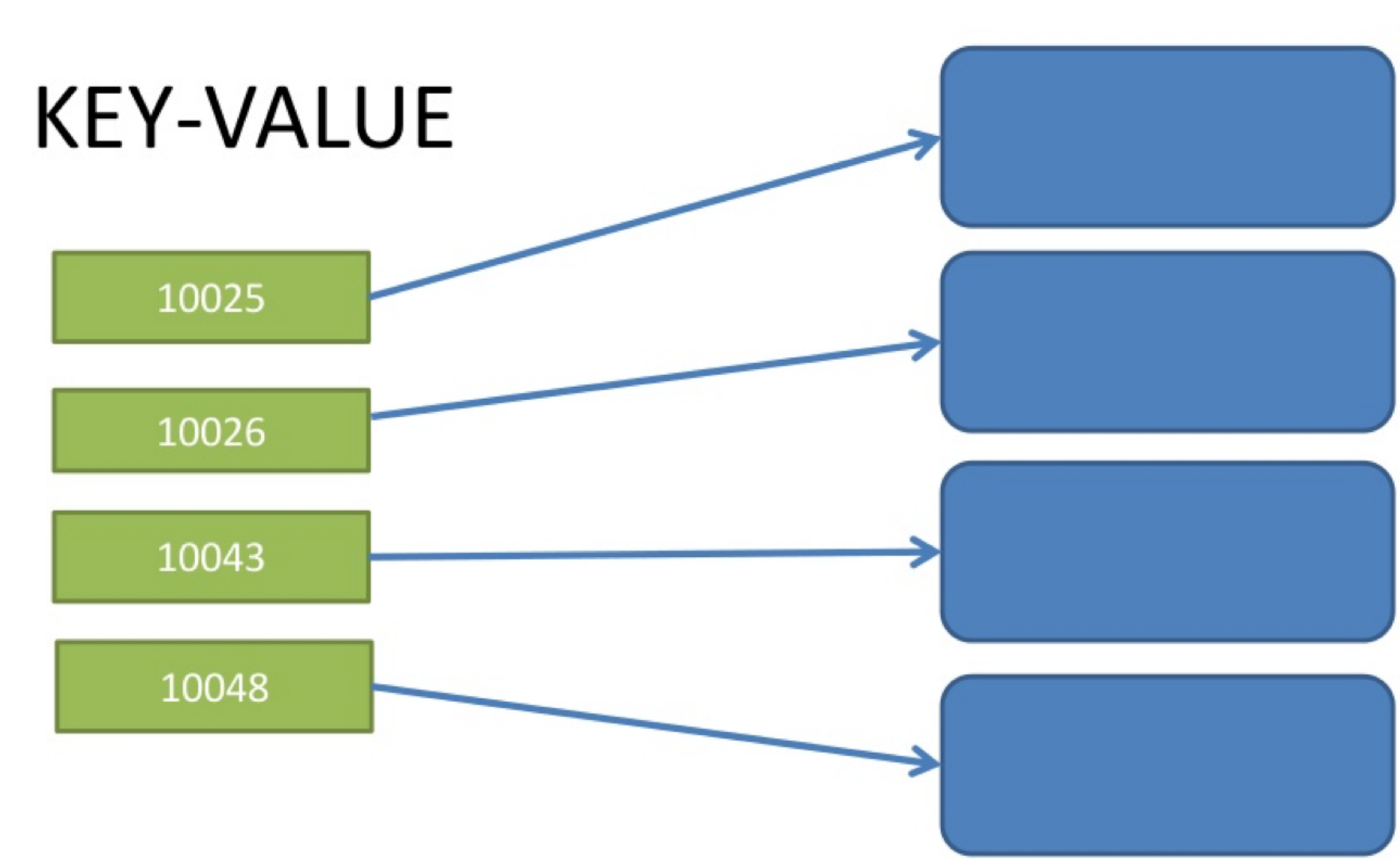
Couchbase

Document

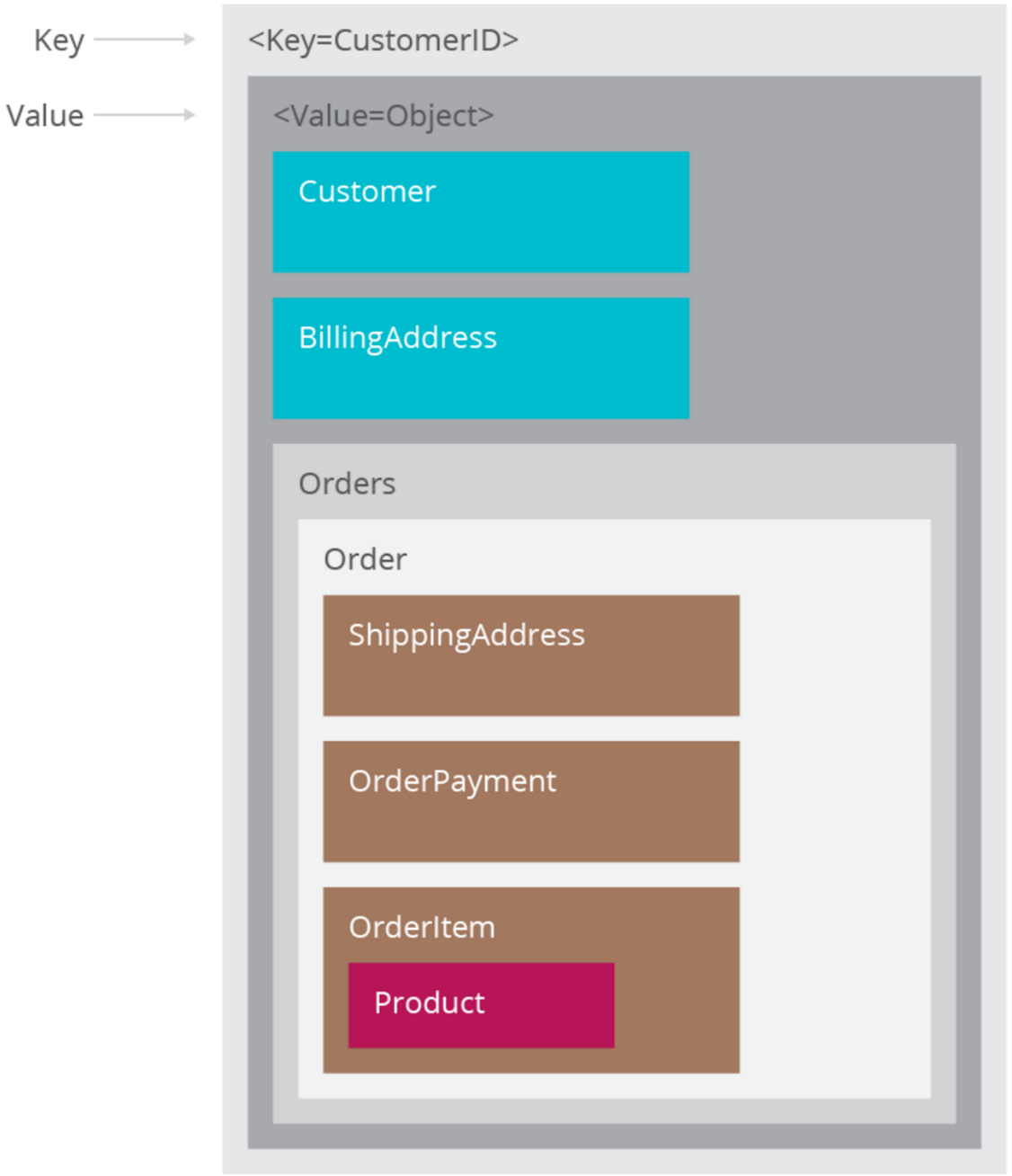
Graph

Key-Value

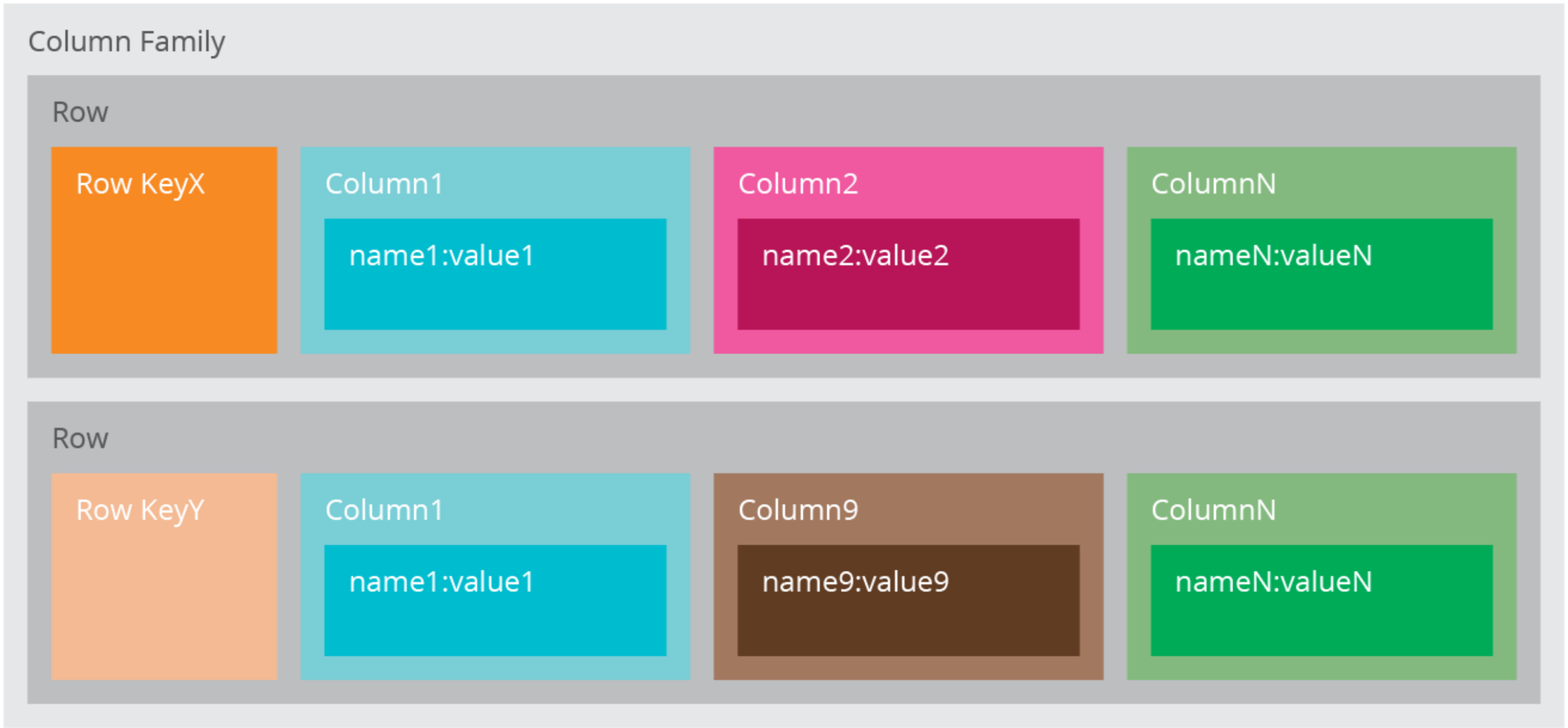
Key-value



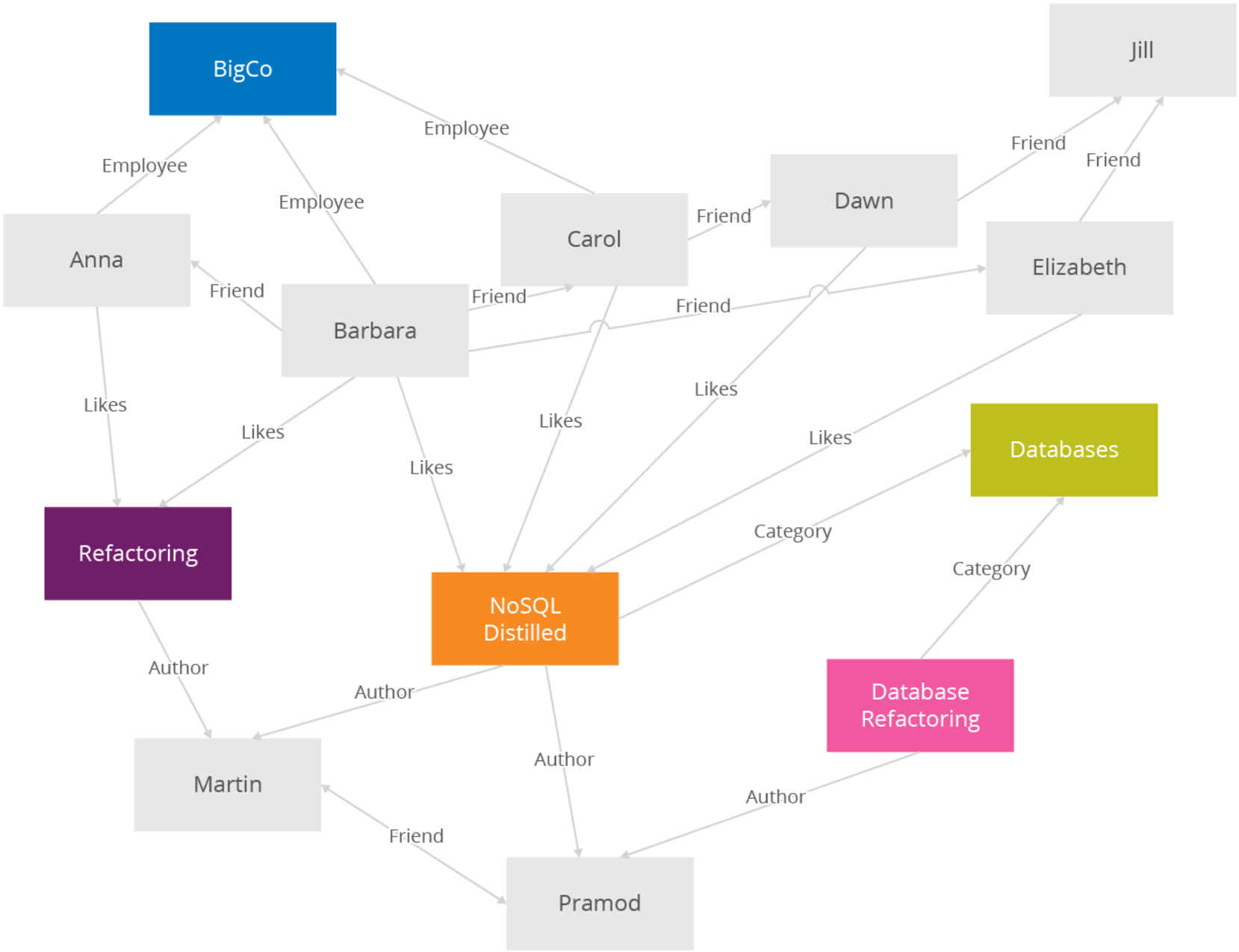
Document



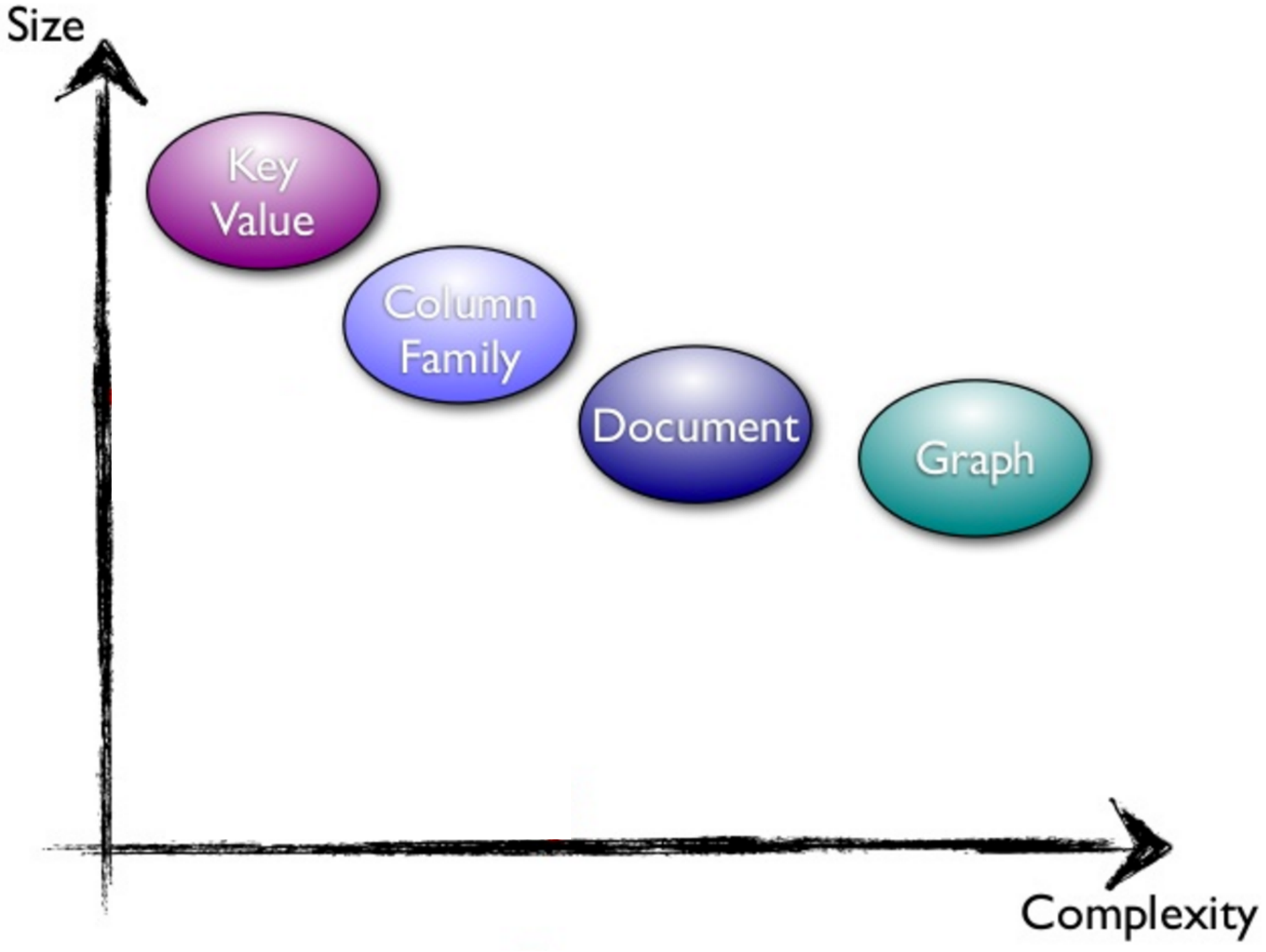
Column



Graph



Data Size vs Complexity



NoSQL and Consistency

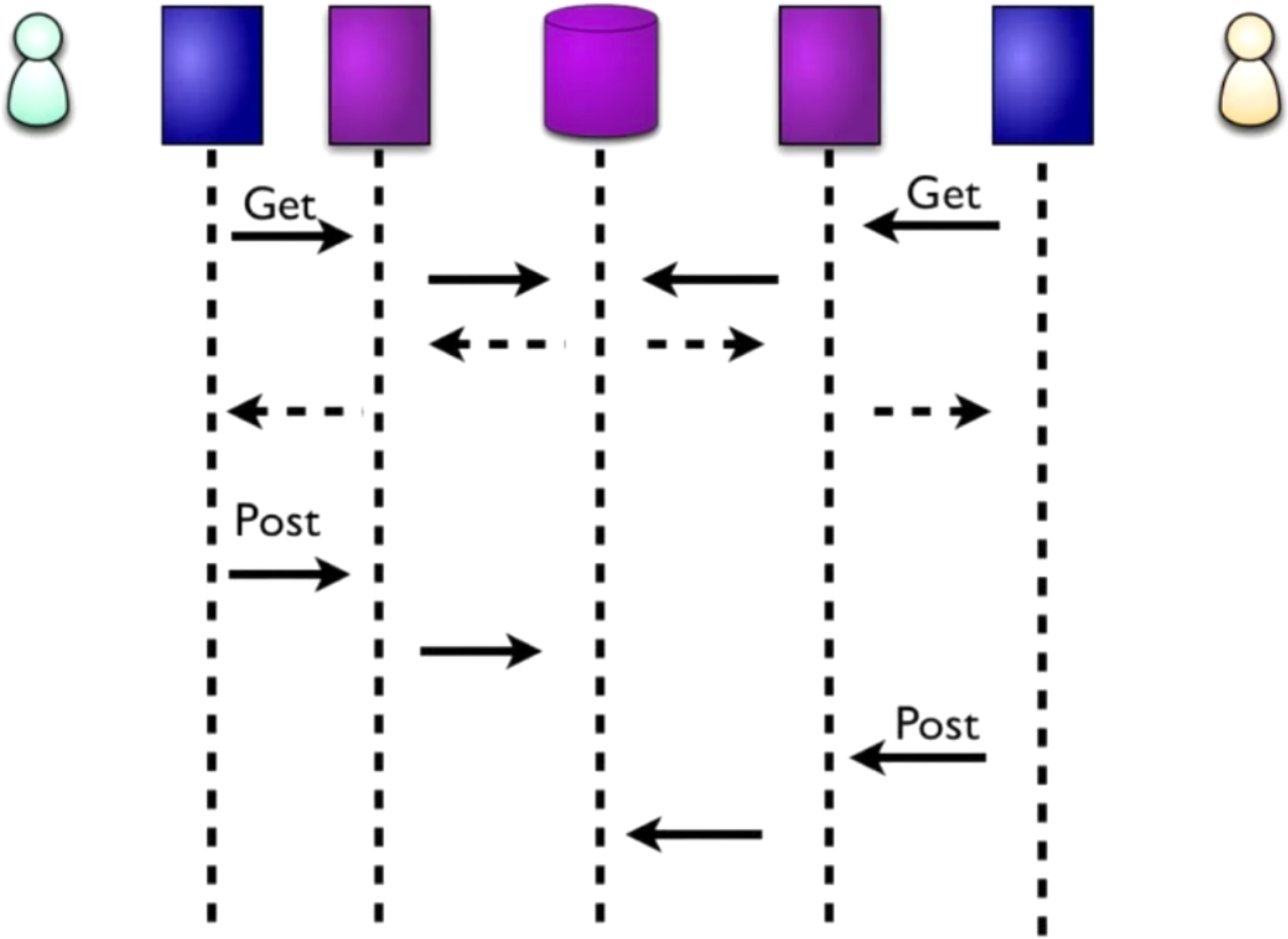
RDBMS => ACID

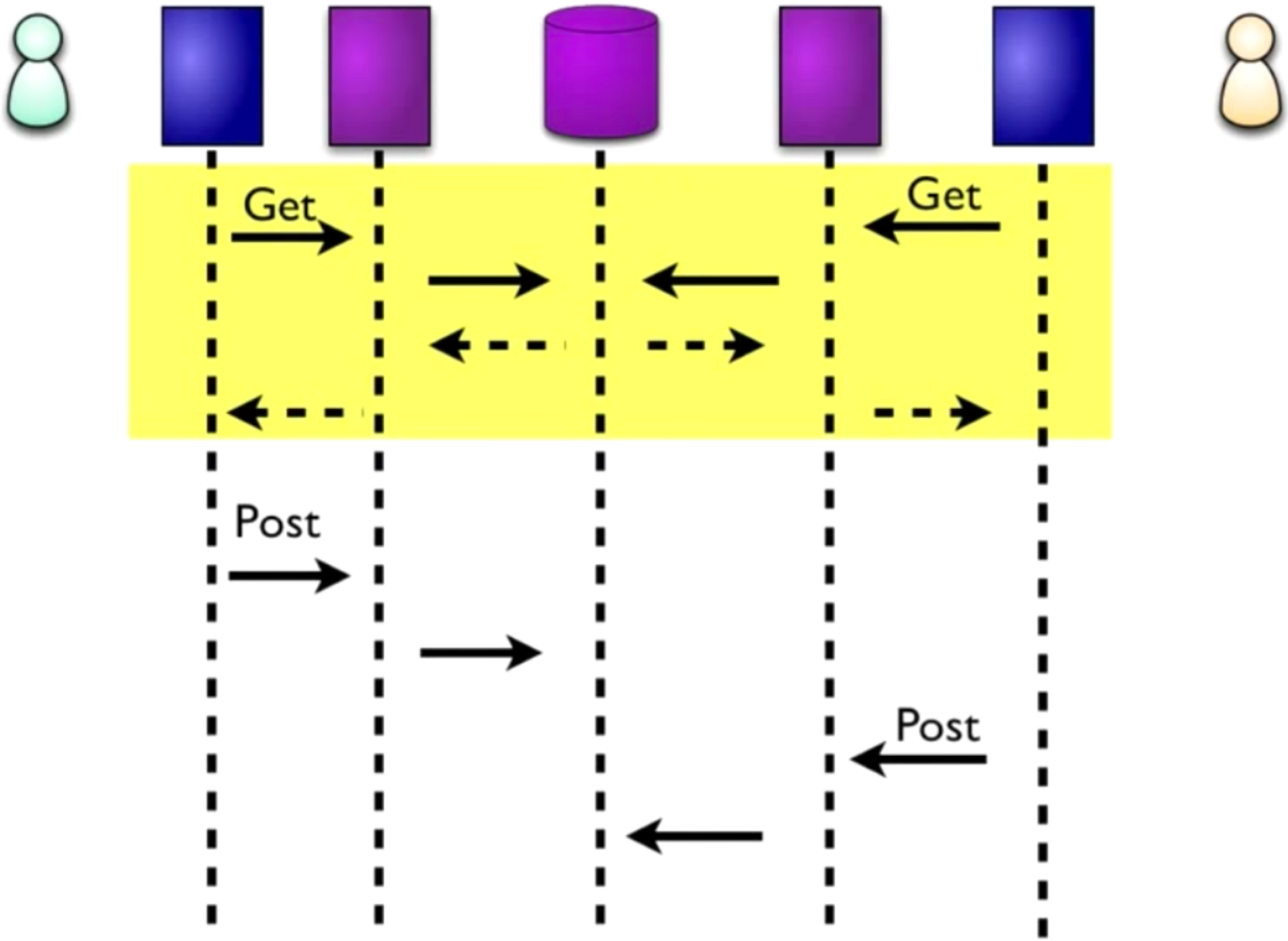
NoSQL => BASE

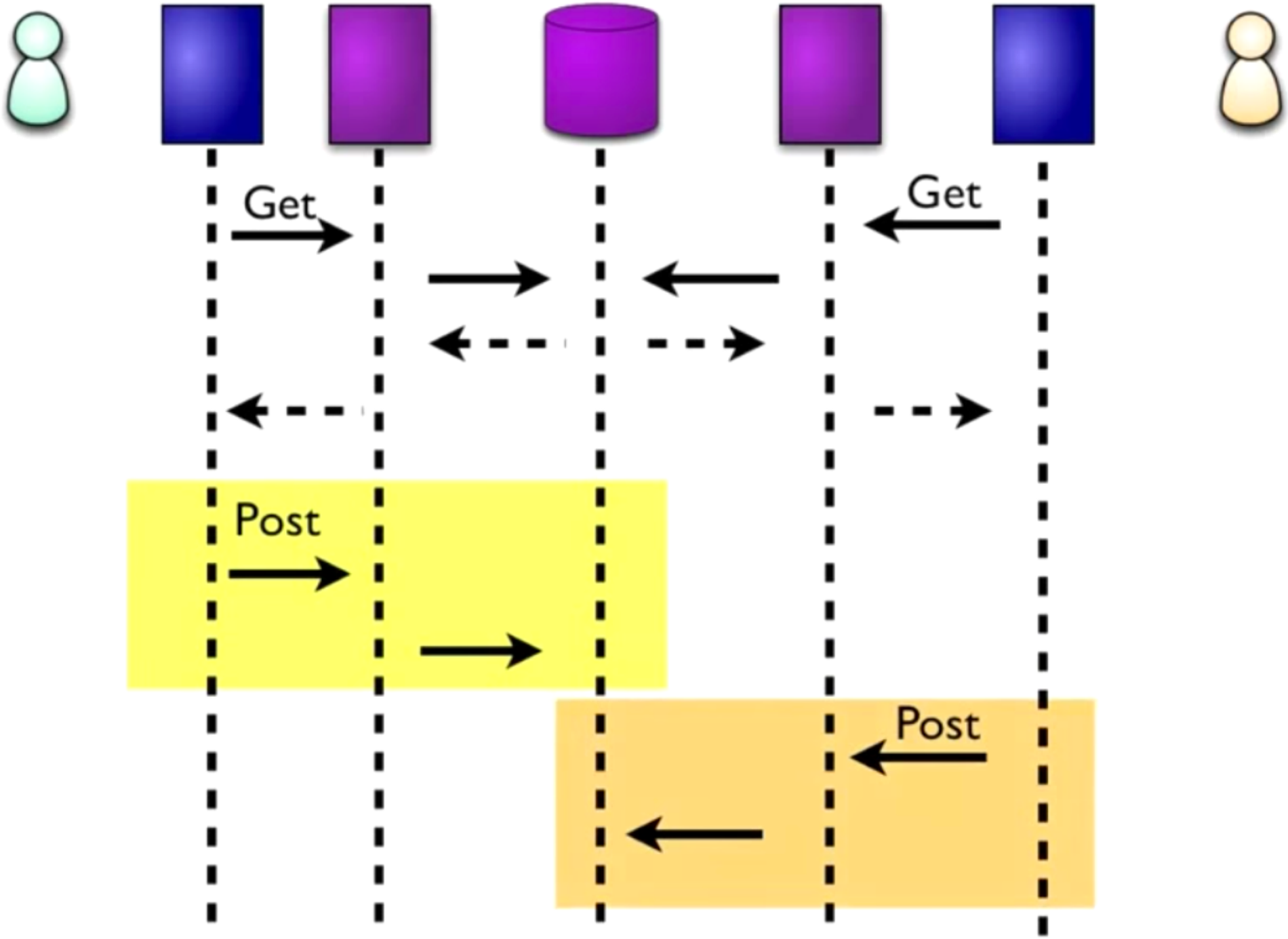
BASE

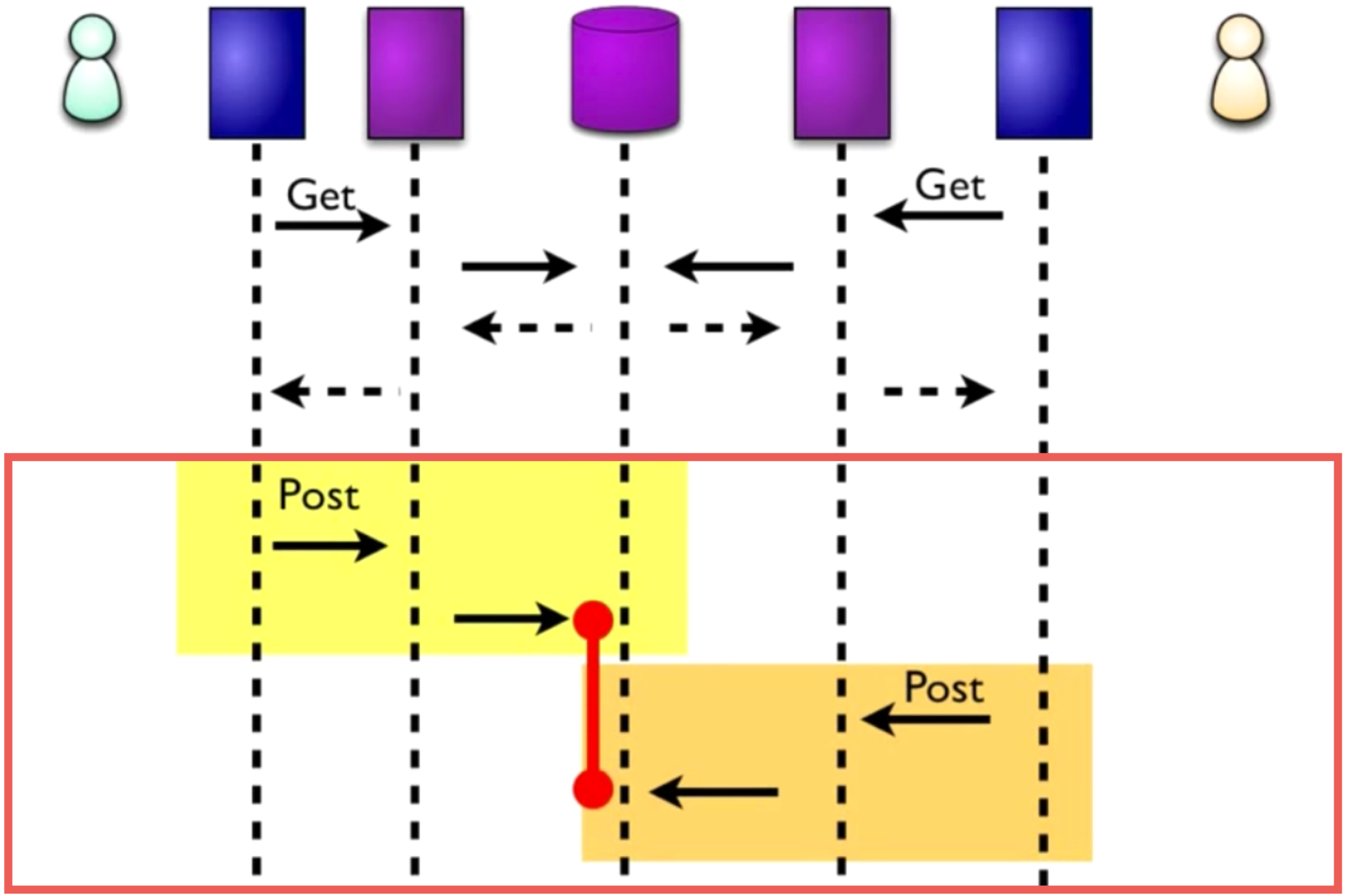
Basic **A**vailability
Soft state
Eventual consistency

Consistency problem





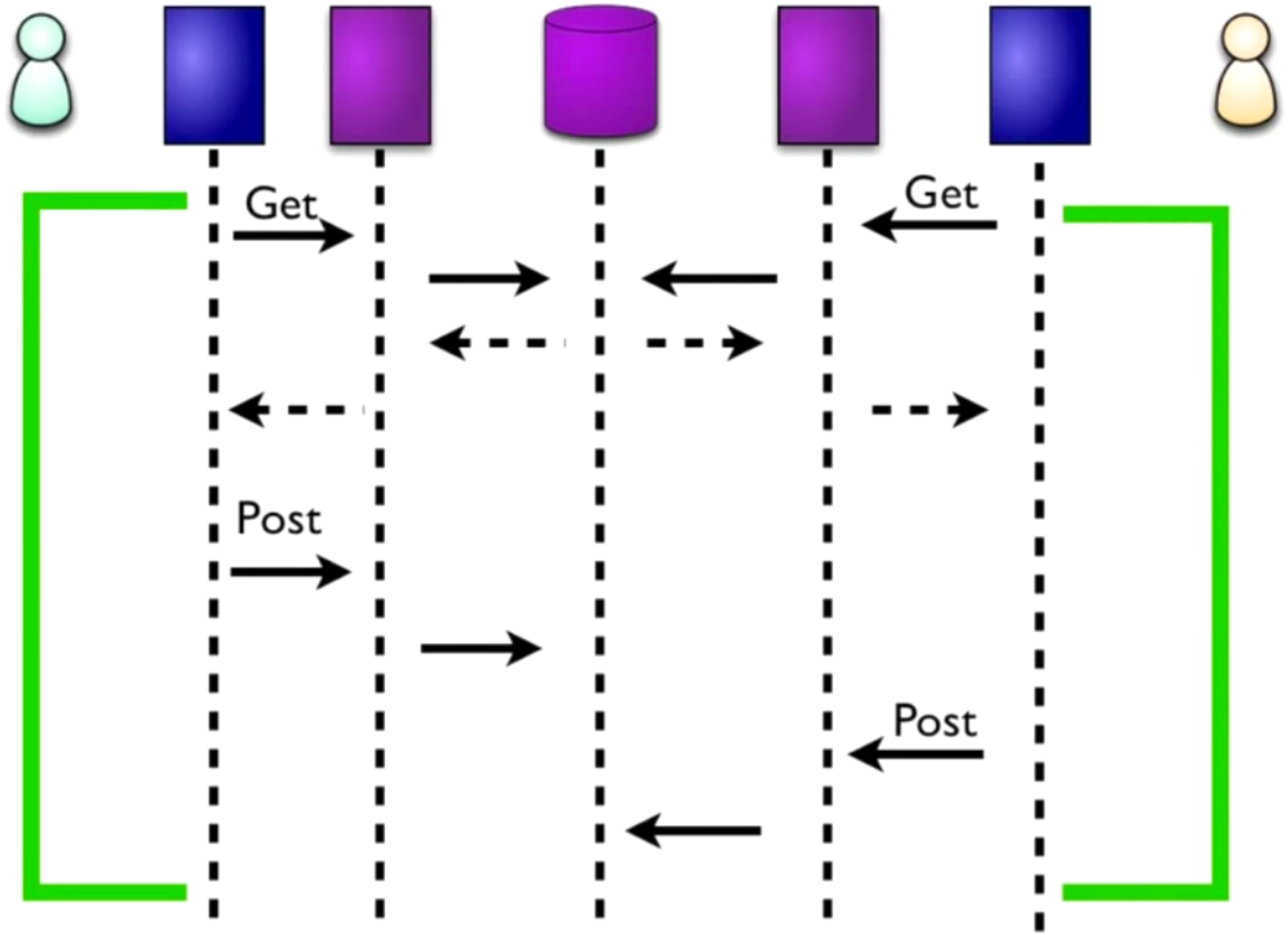




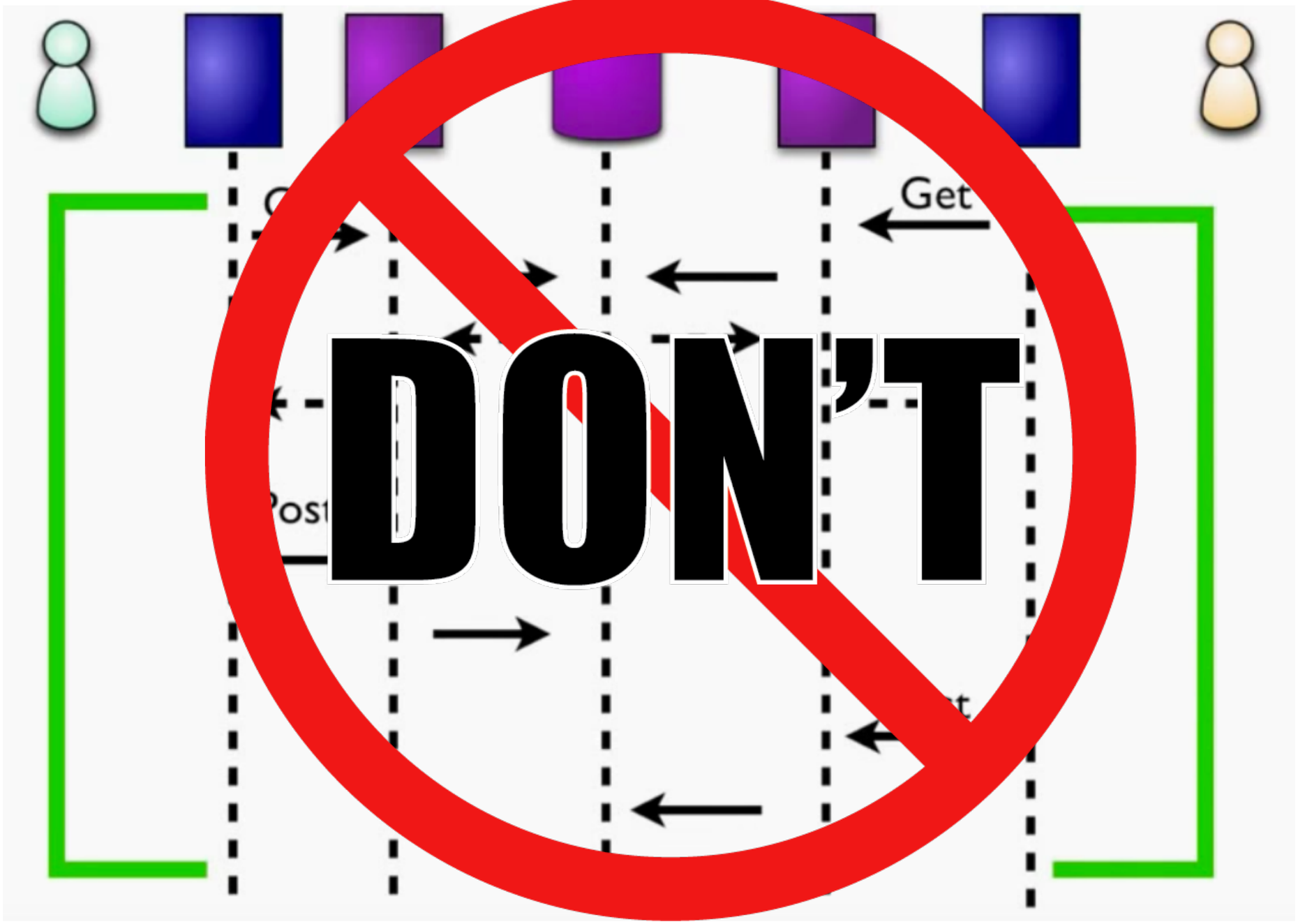
Problem ?

Solution ?

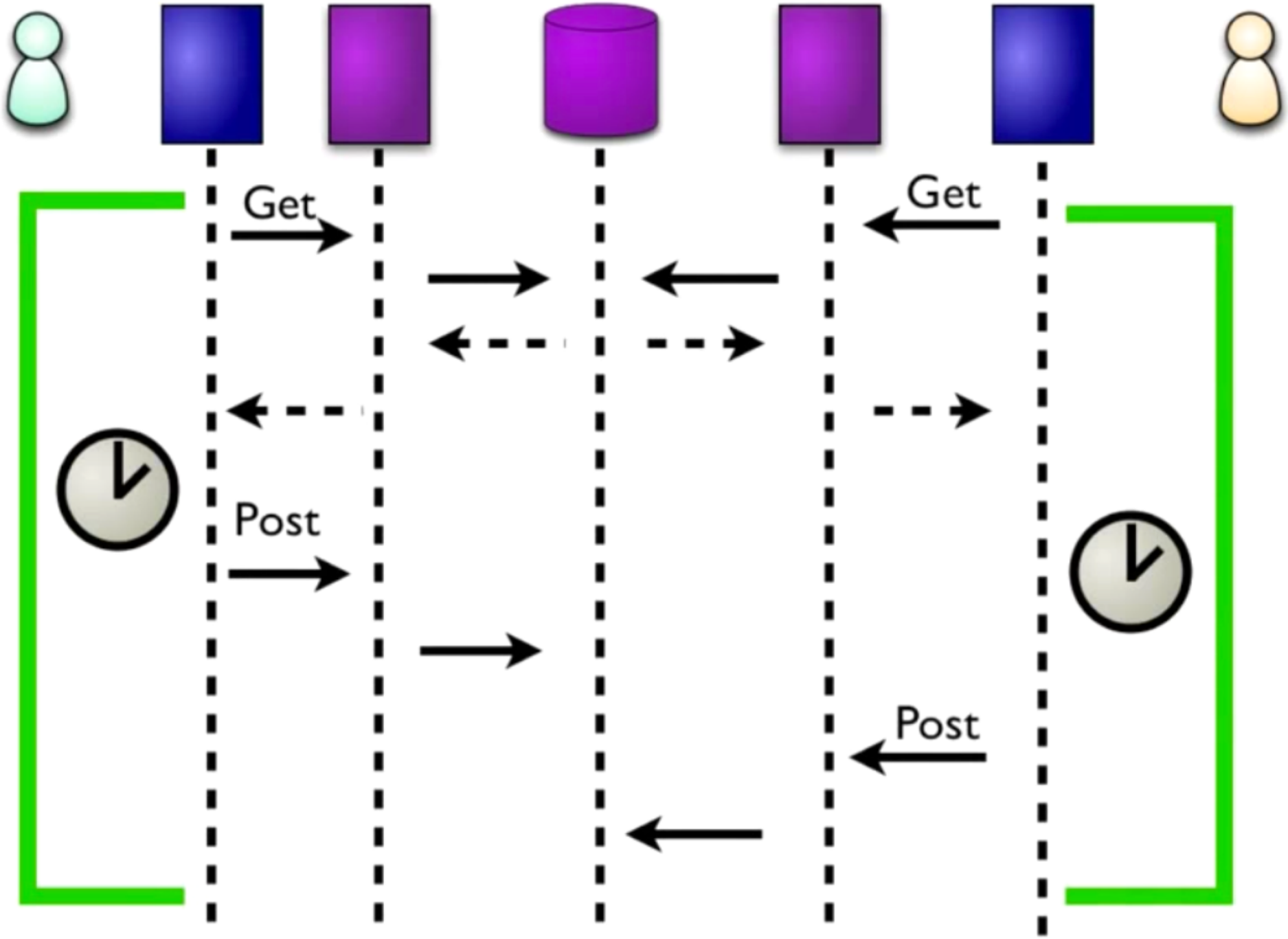
Transaction ?



Transaction ?

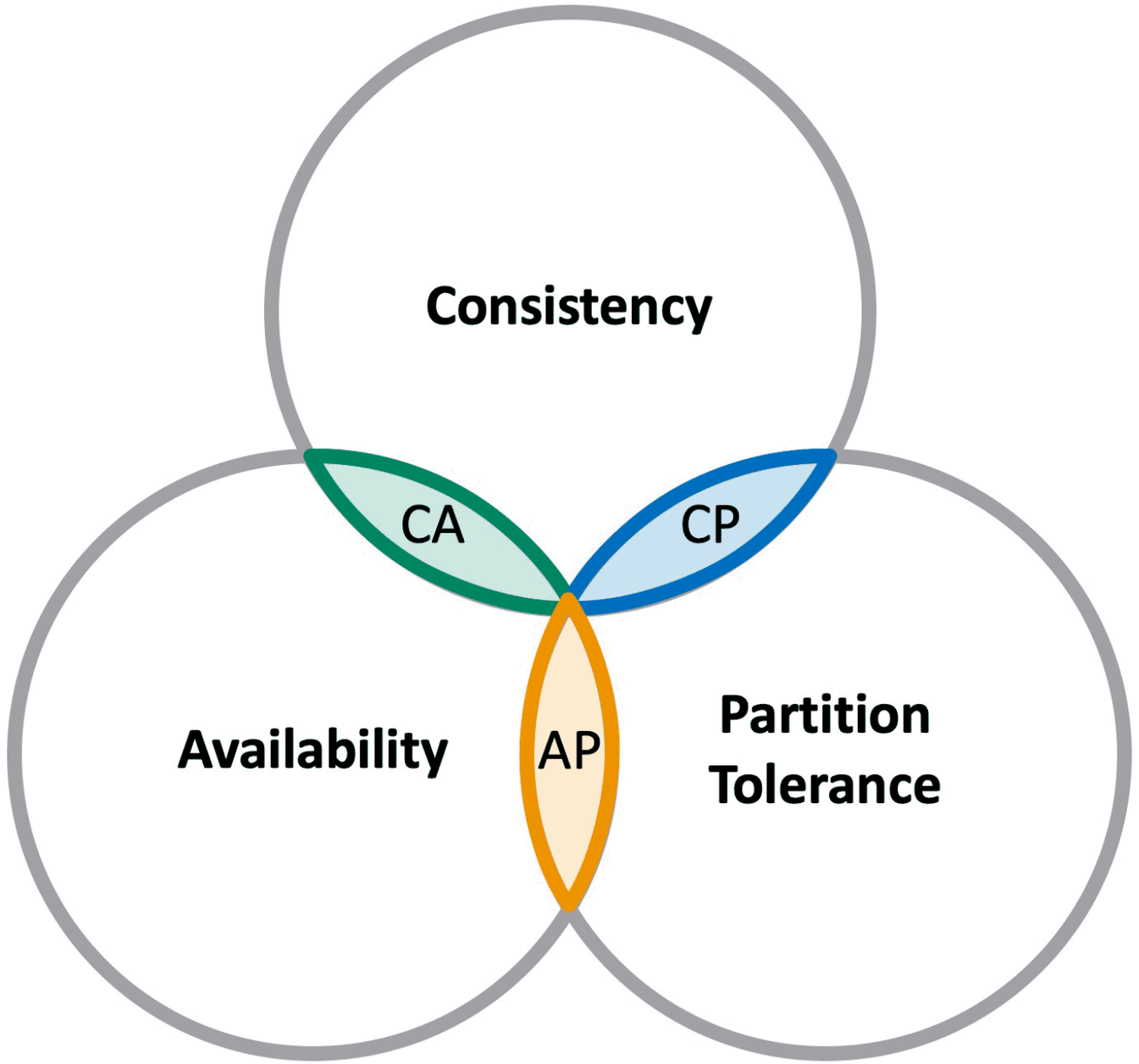


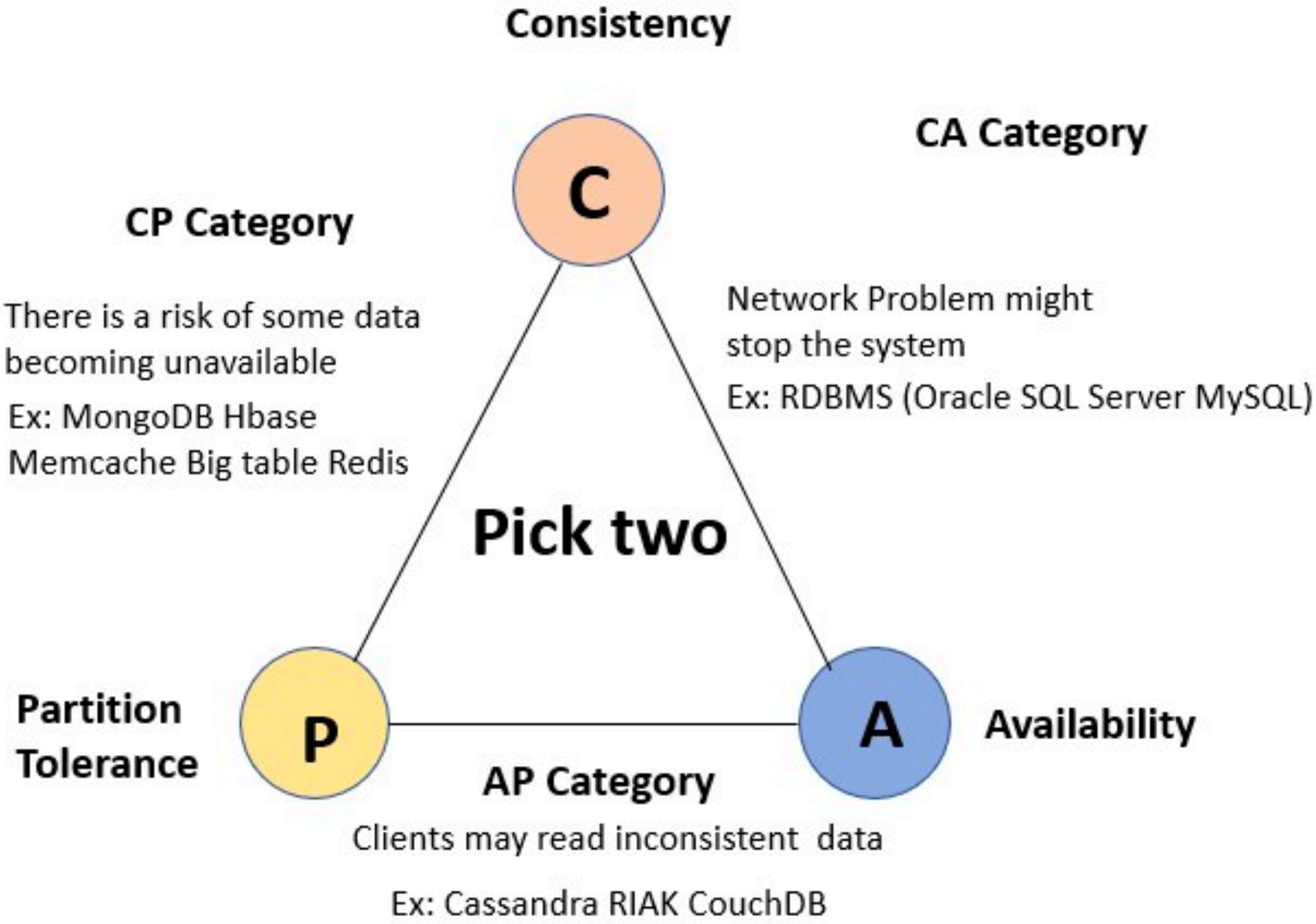
Deadlock

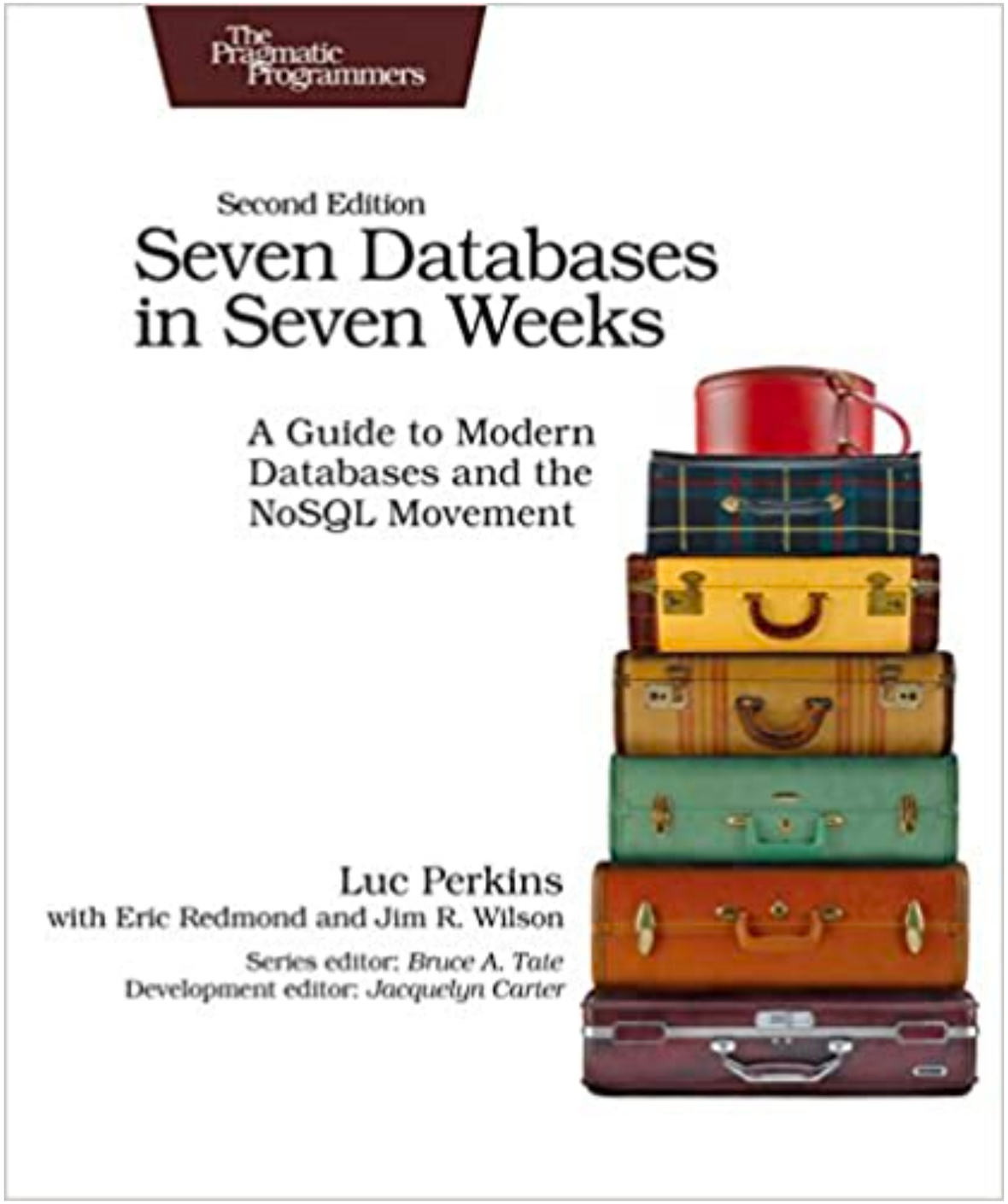


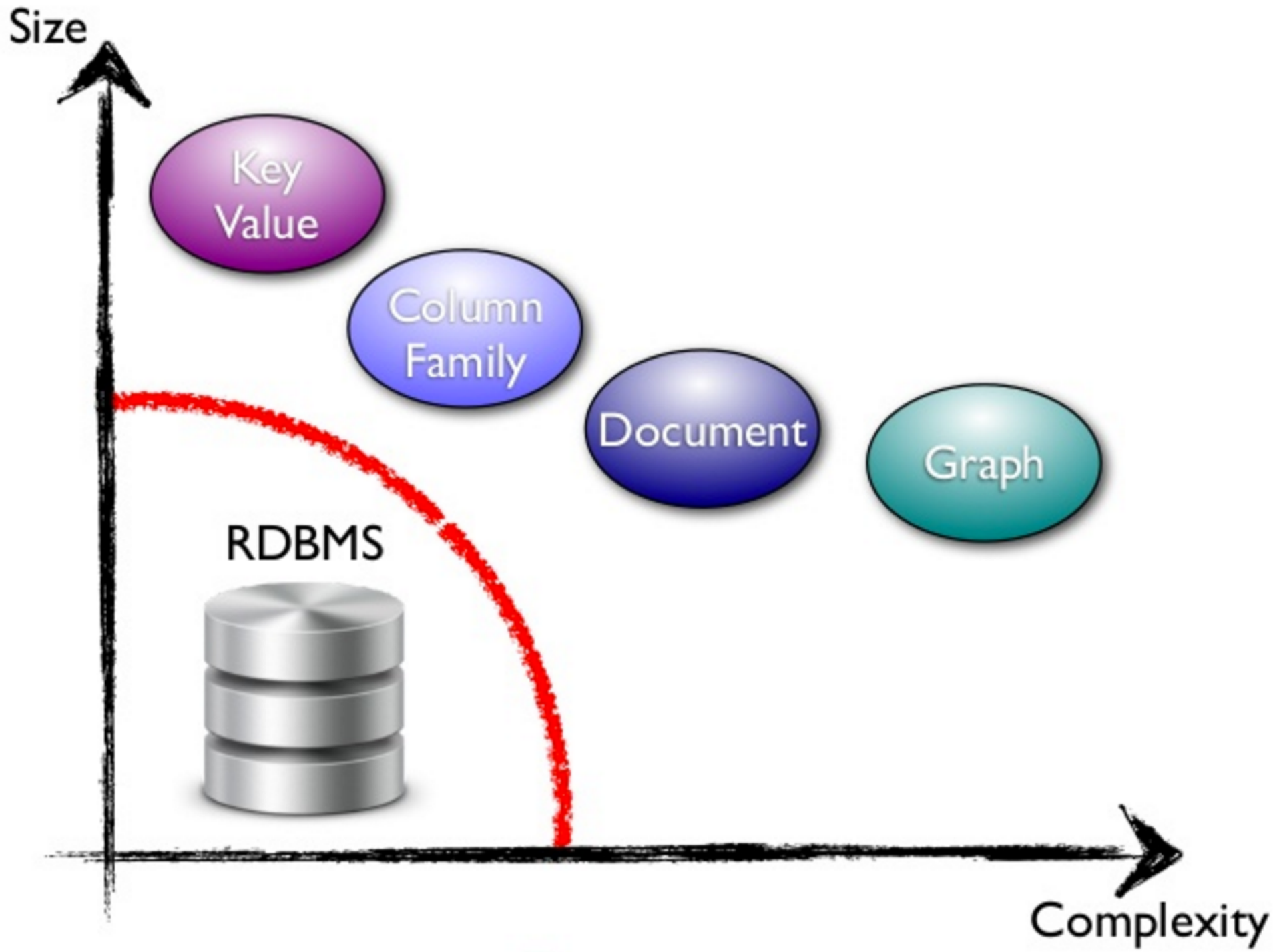
New way ?

CAP Theorem







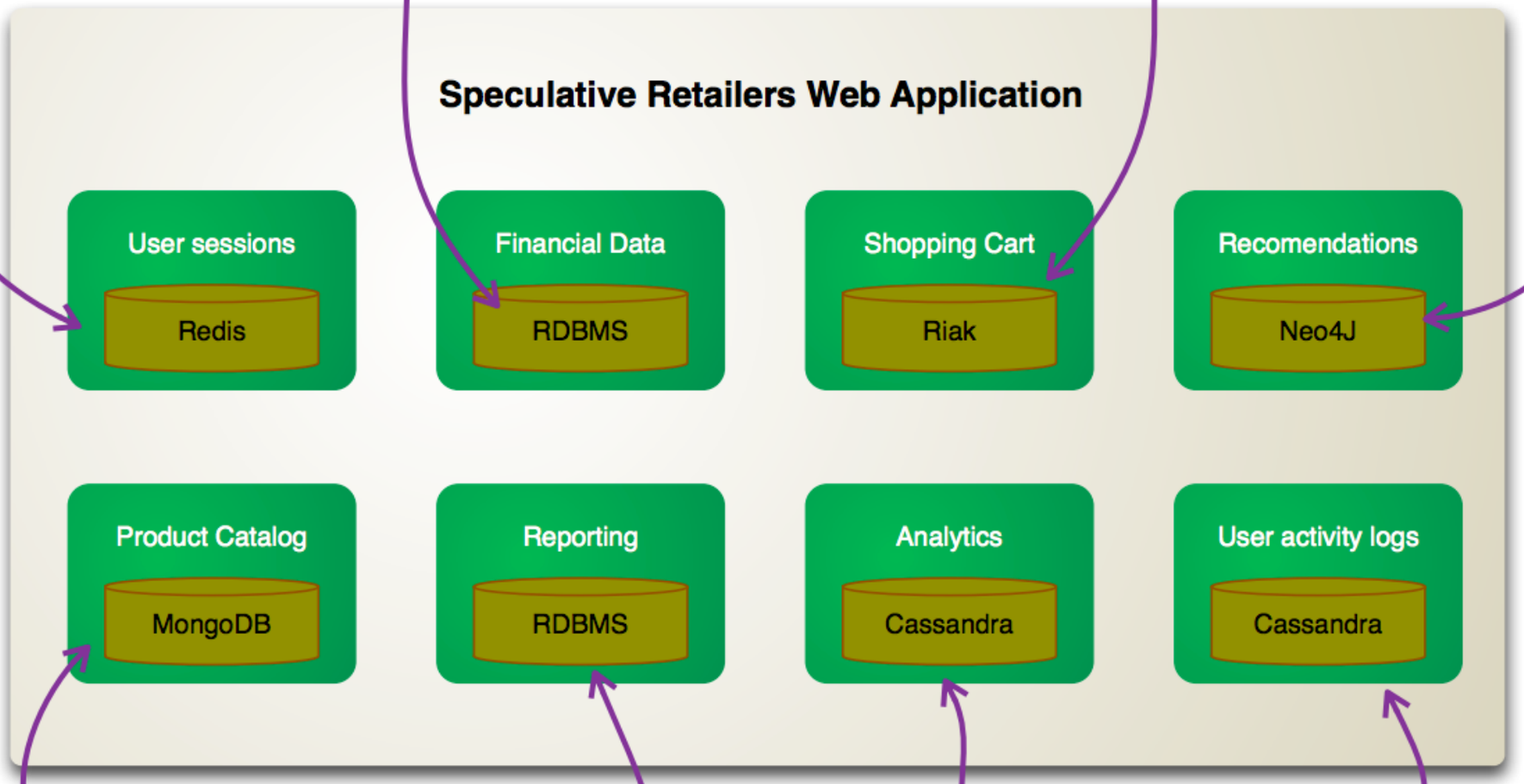


Rapid access for reads and writes. No need to be durable

Needs transactional updates. Tabular structure fits data

Needs high availability across multiple locations. Can merge inconsistent writes

Rapidly traverse links between friends, product purchases, and ratings



This is a very hypothetical example, we would not make technology recommendations without more contextual information

Lots of reads, infrequent writes. Products make natural aggregate

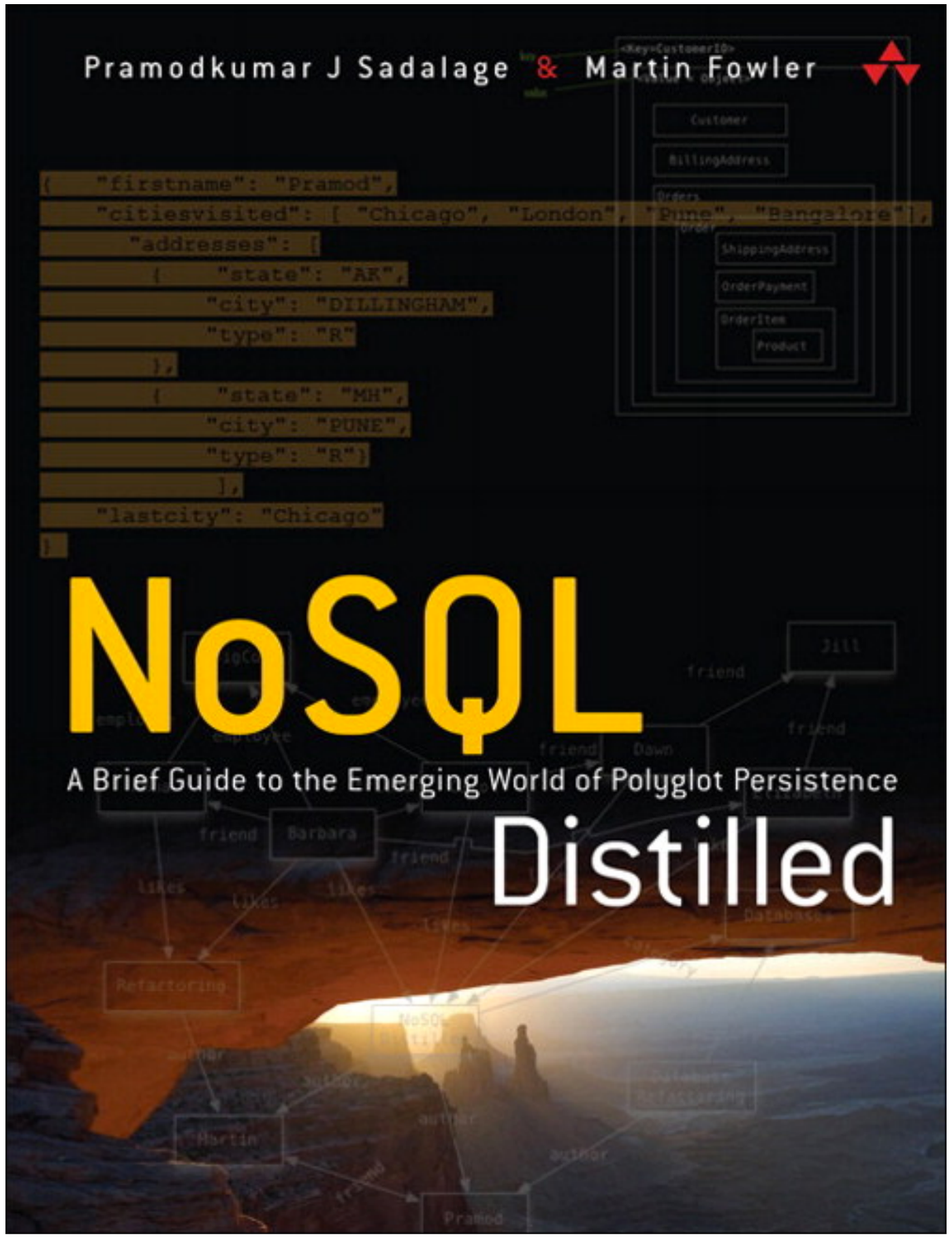
SQL interfaces well with reporting tools

Large-scale analytics on large cluster

High volume of writes on multiple nodes

Problems

Decisions
Organization changes
Immaturity
Dealing with Eventual consistency



NoSQL + RDBMS

SCALEBASE

Tokutek™

dbShards

ScaleArc

AKIBAN

ScaleDB

SCHOONER

NewSQL

JustOneDB

nuoDB

SQLFire

TRANSLATTICE

VoltDB

H-Store

GENIEDB®

xeround

Clustrix

KEEP
CALM

AND

use the right tool
for the right job

Workshop

Workshop

MariaDB
PostgreSQL
Redis
Dealing with Eventual consistency

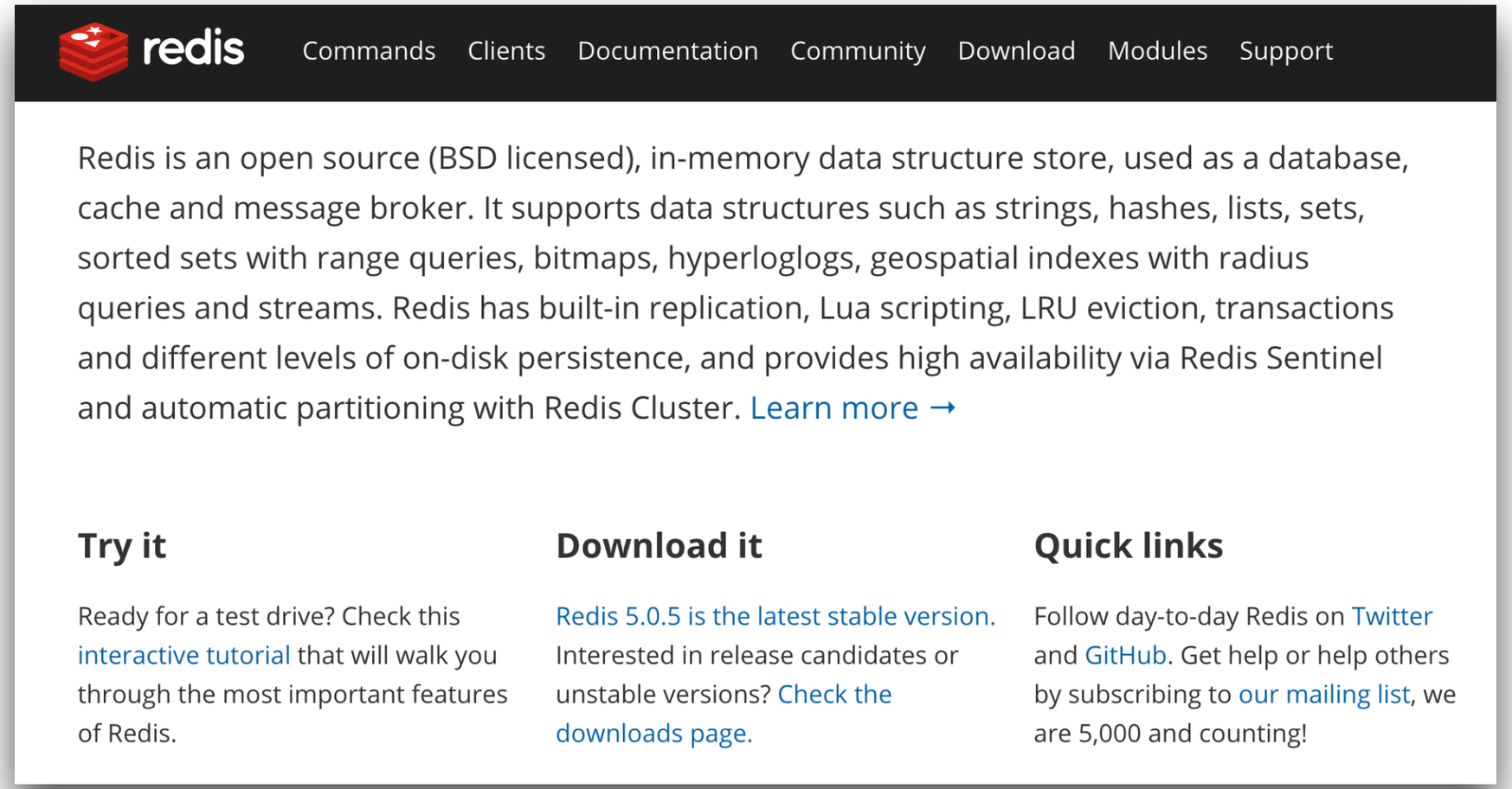
RDBMS and NoSQL

Redis

https://redis.io/

Redis

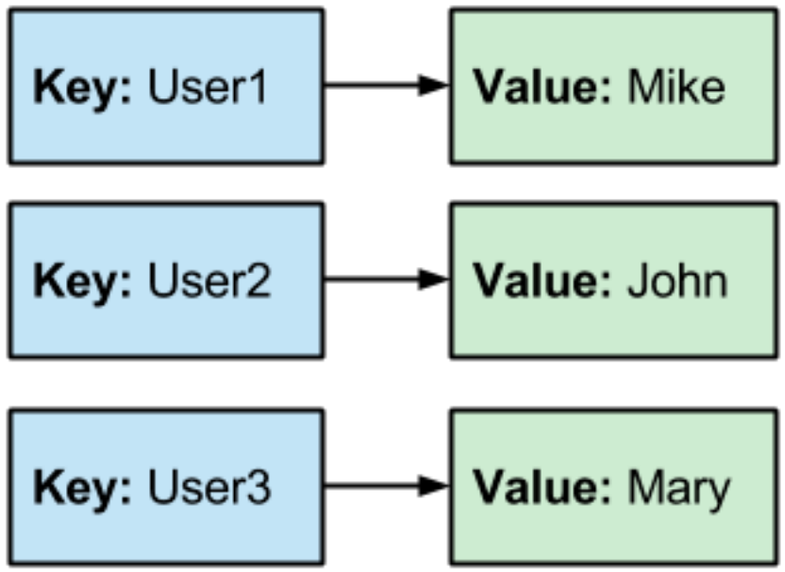
In-memory data structure store
Using for database cache and message broker



https://redis.io/

Redis

In-memory key-value store
Focus on performance and simplicity
All data keep in memory



Redis

REmote **DI**ctionary **S**erver

Data structure server

String, list, set, hash, sorted set

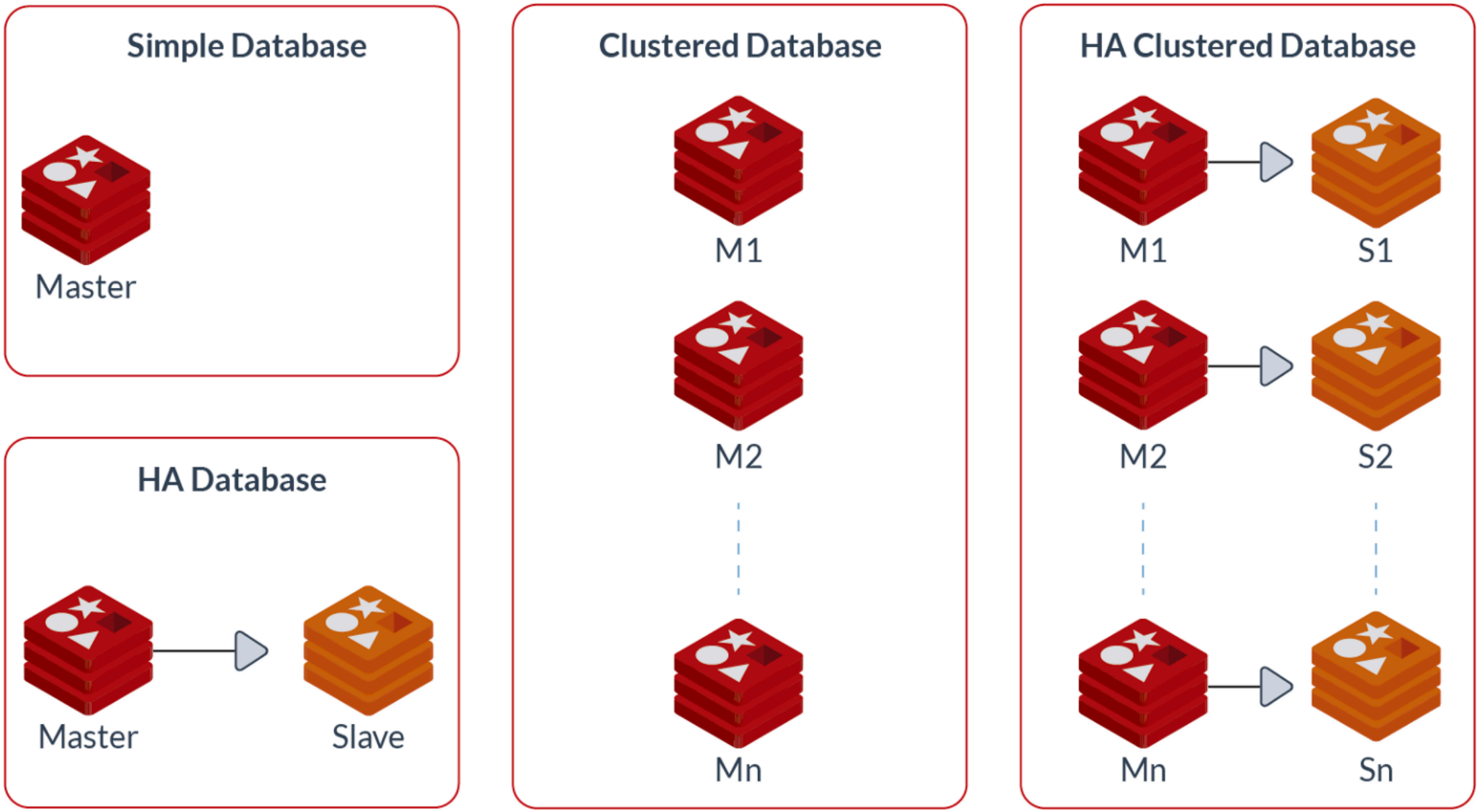
No index

No query language

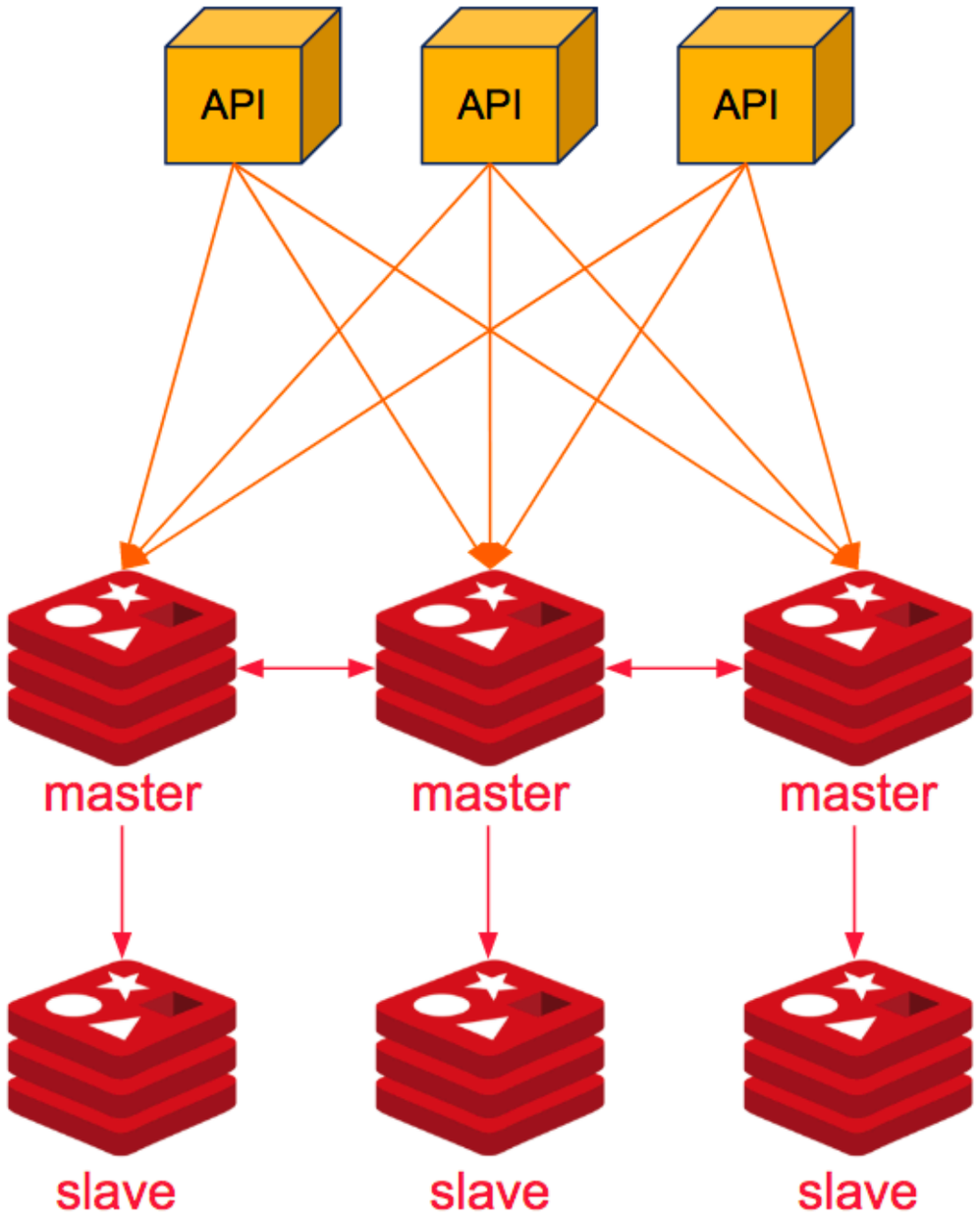
Redis networking

Local or distributed access
Replication
Clustering

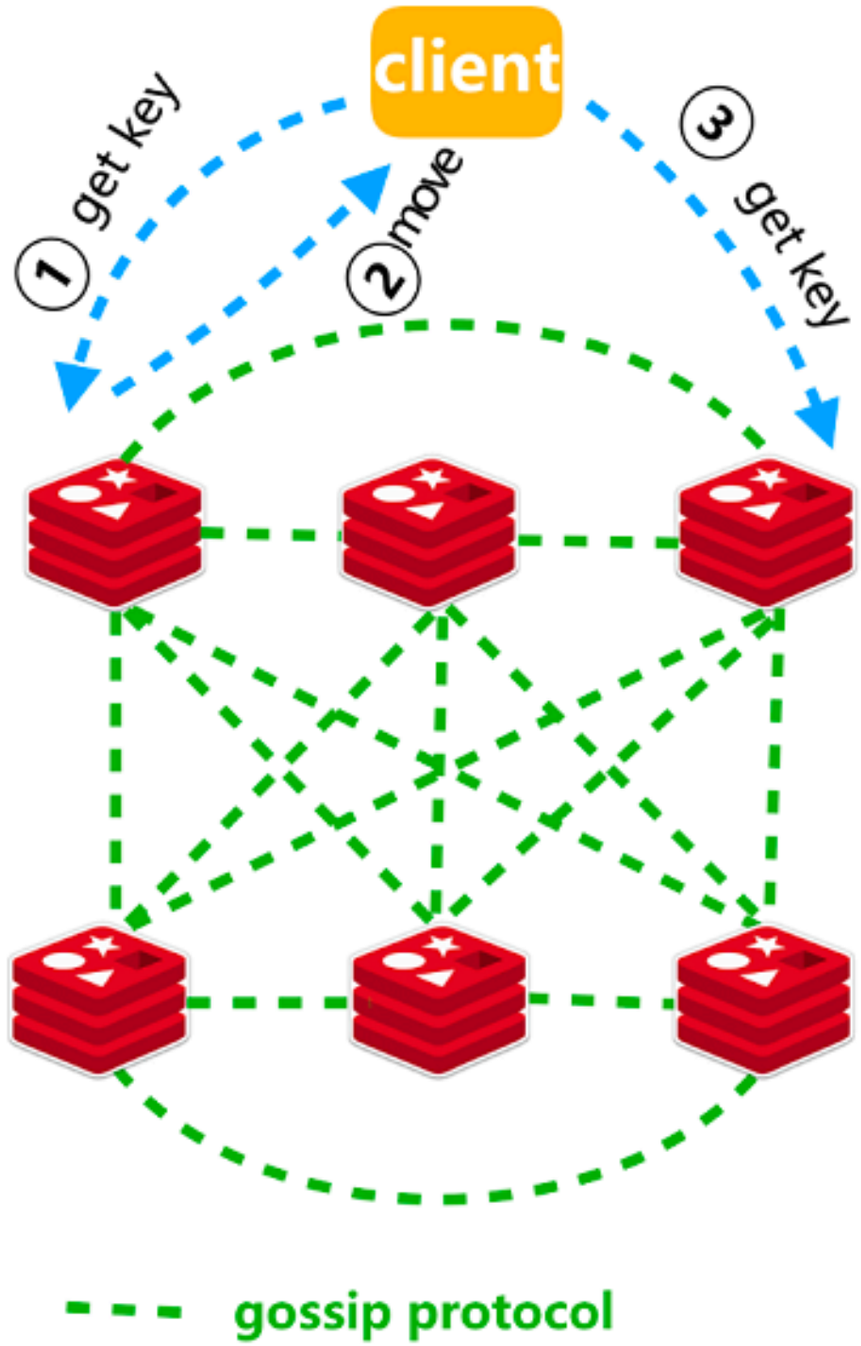
Redis networking



Redis replication



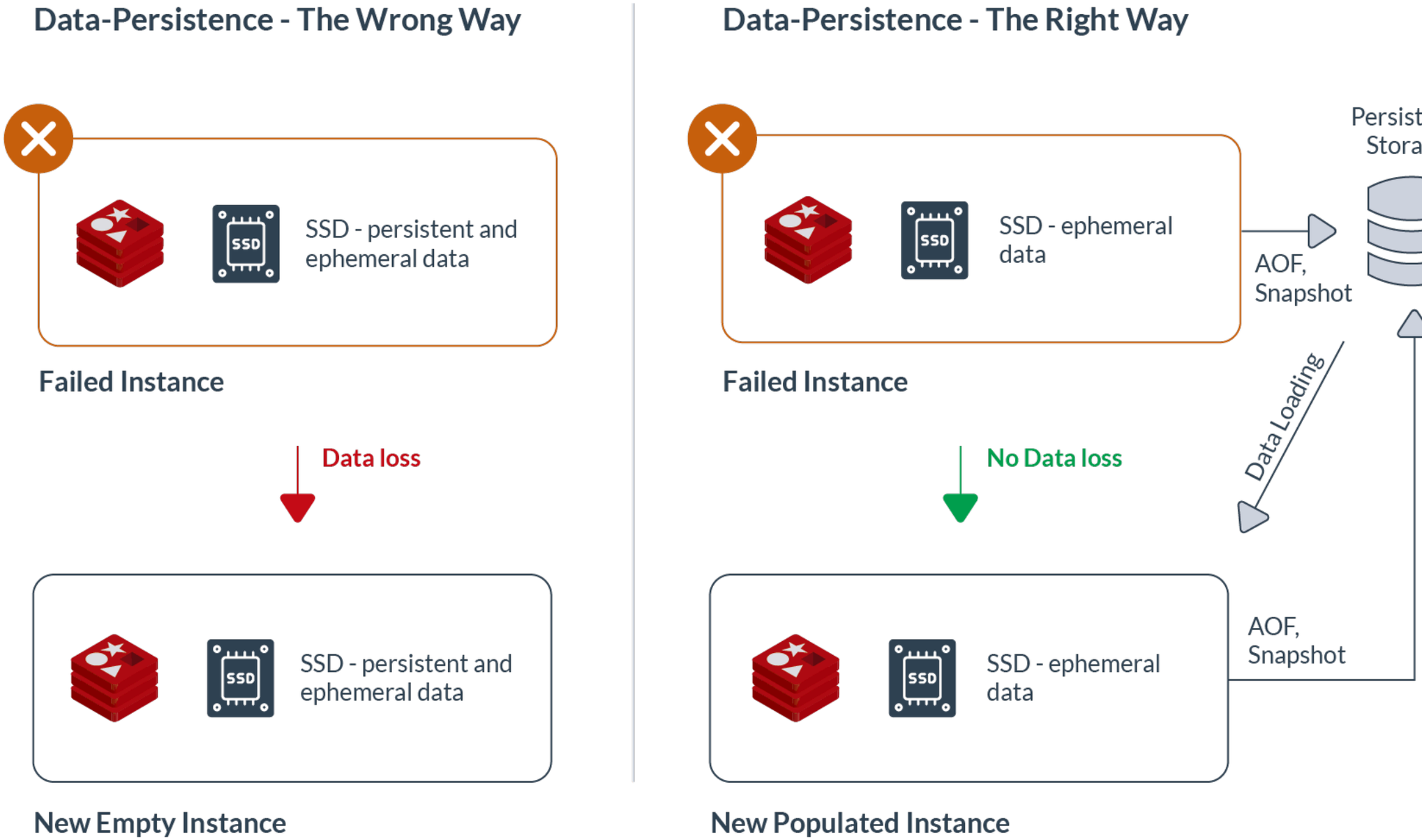
Redis cluster (Sentinel)



Durability options

Memory only
Append-only file (AOF)
Snapshot (RDB)

Memory vs AOF



https://redislabs.com/redis-enterprise/technology/durable-redis-2/

Append-Only-File

Append new data to file system
Every second (fast but less safe)
Every write (safer but slower)

Snapshot

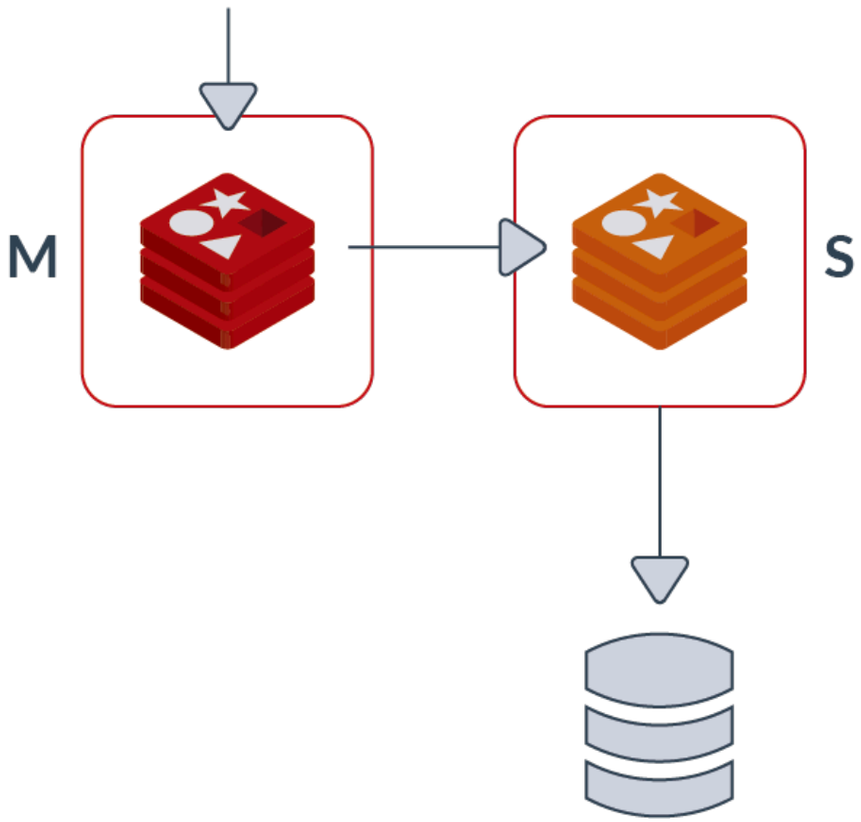
The entire point-in-time view of the dataset is written to persistent storage, across all shards of the database

The snapshot time is **configurable**

Tuning for Master-Slave

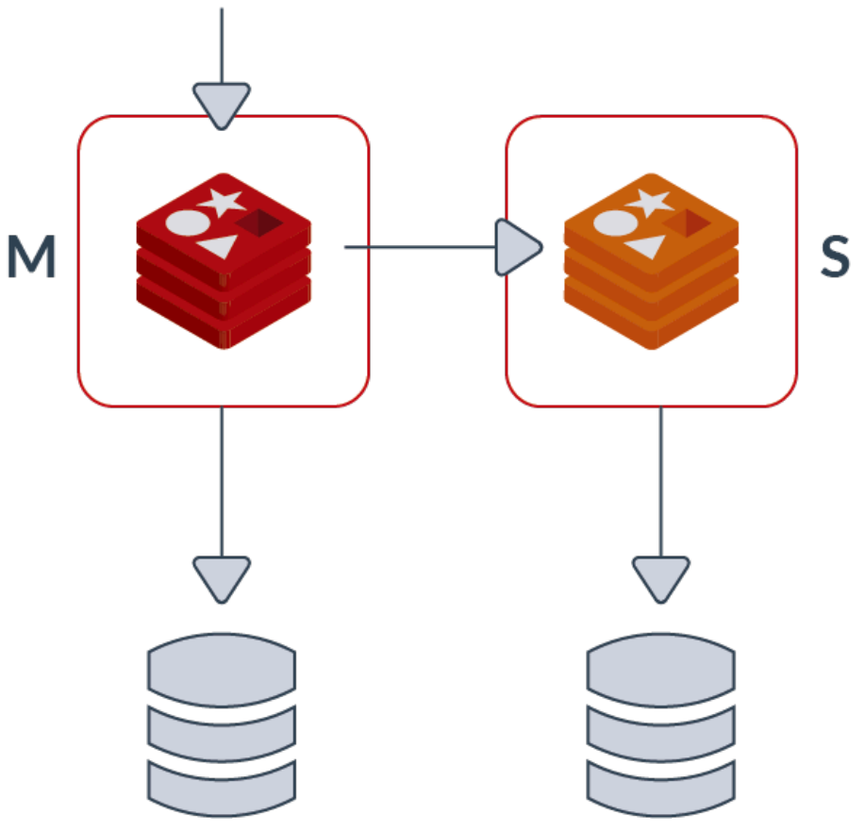
Tuned for Speed

Data-Persistence at the slave



Tuned for Reliability

Data-Persistence at the master & slave



Snapshot vs Backup

Snapshot vs Backup

Snapshot

supports data durability
(i.e. to automatically recover data when there is no copy of the dataset in memory)

Backup

supports disaster recovery
(i.e. when the entire cluster needs to be rebuilt from scratch)

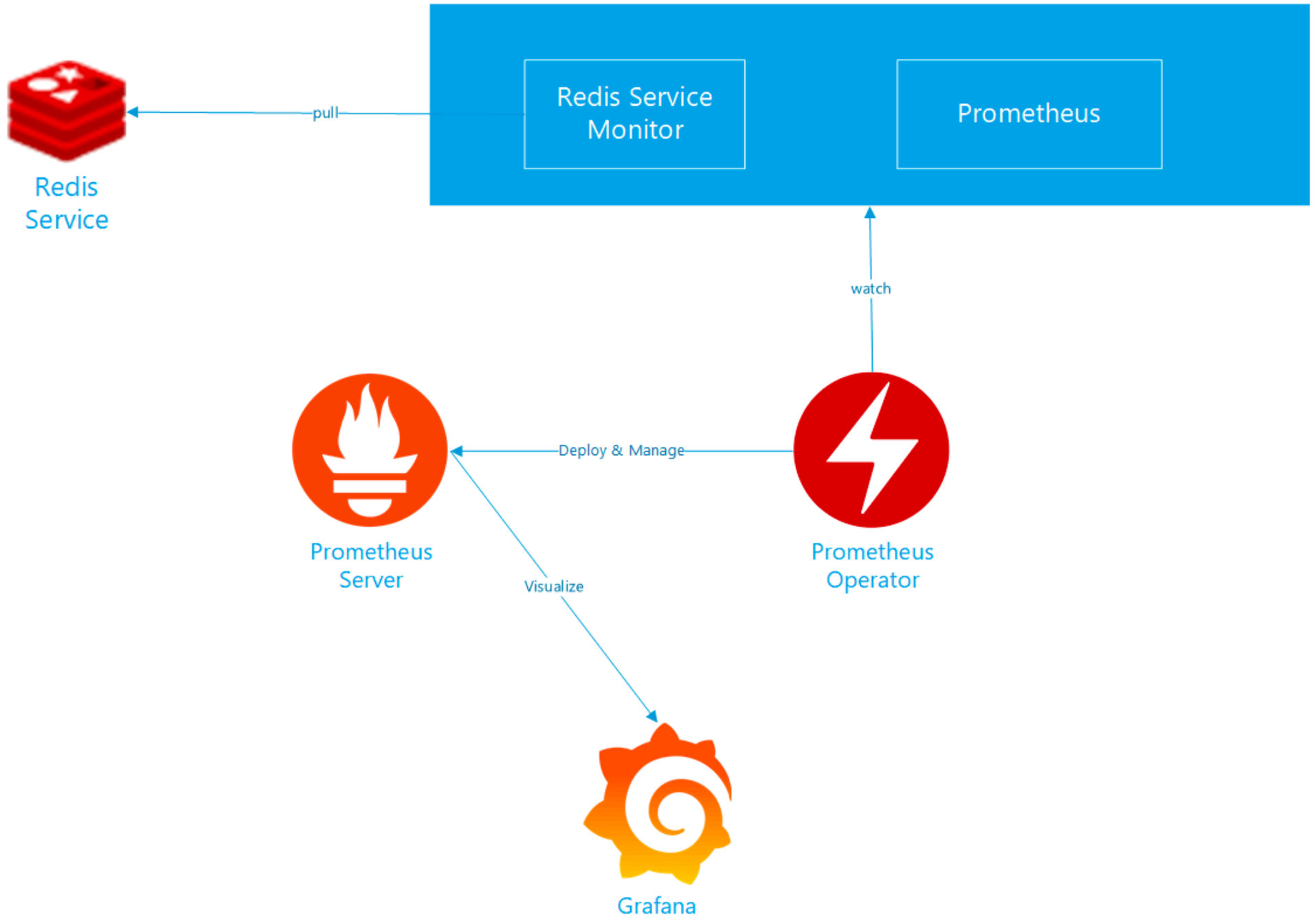
Redis workshop

Basic of Redis operation

Clustering

Monitoring

Redis monitoring



PostgreSQL

https://redis.io/

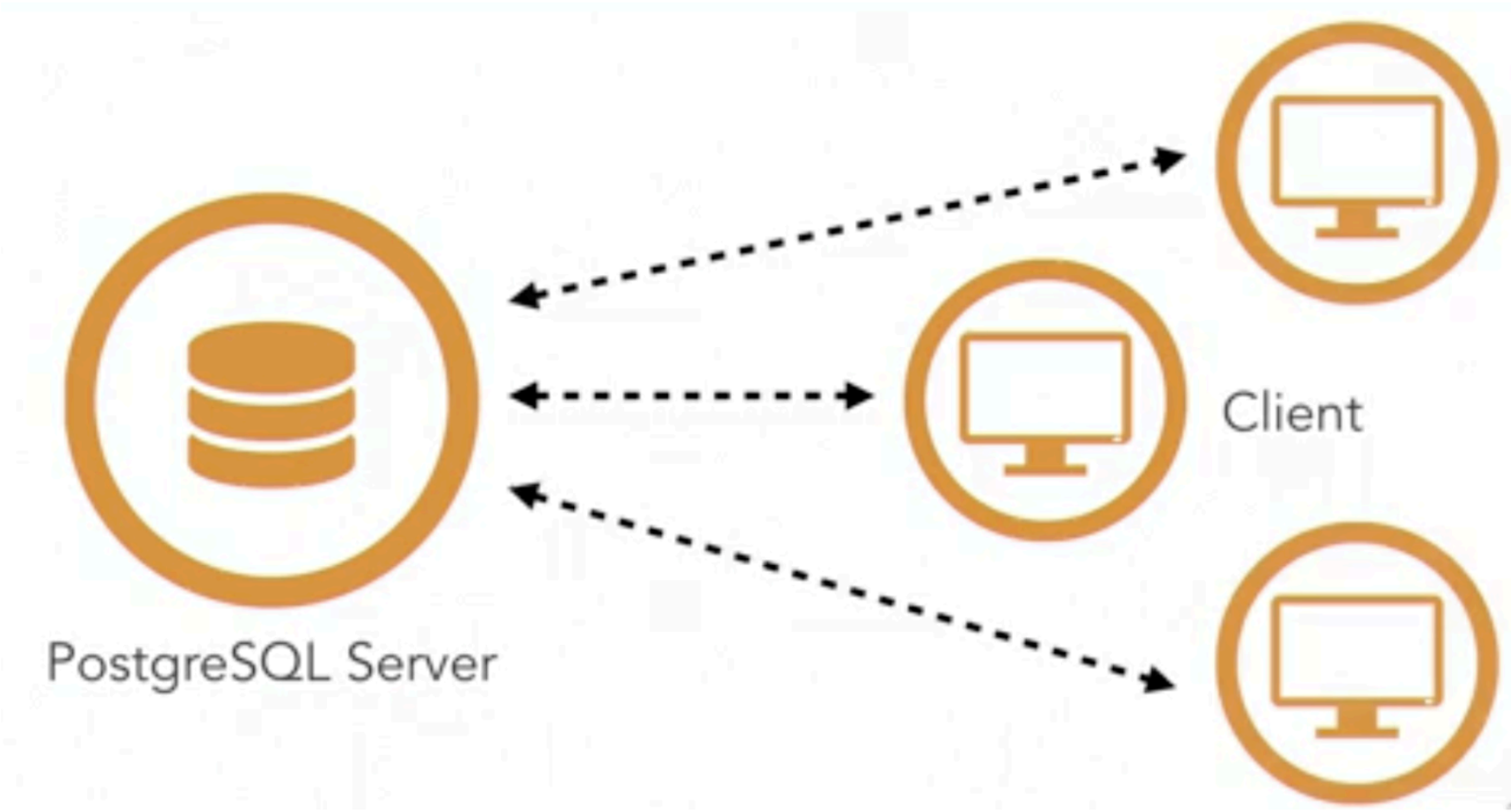
PostgreSQL

RDBMS
Opensource
Active community

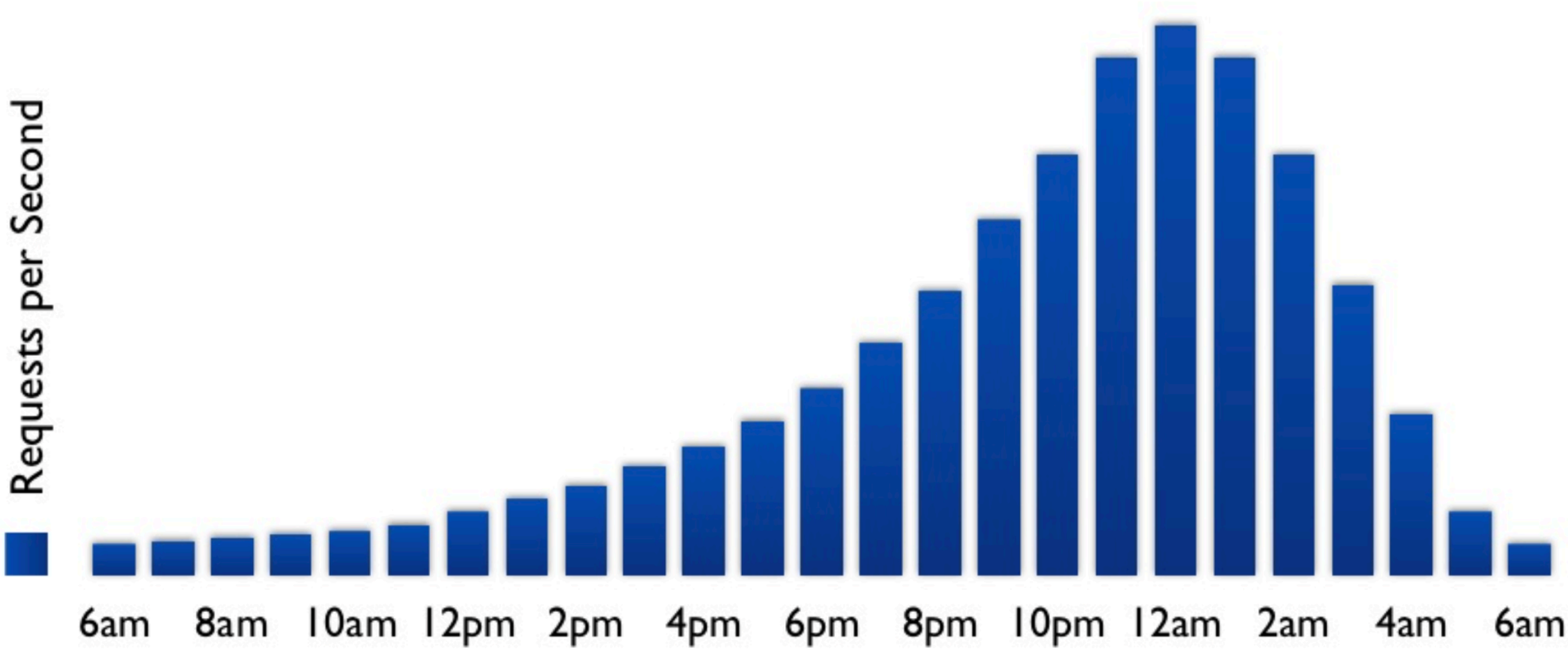
PostgreSQL

Rank			DBMS
Dec 2020	Nov 2020	Dec 2019	
1.	1.	1.	Oracle
2.	2.	2.	MySQL
3.	3.	3.	Microsoft SQL Server
4.	4.	4.	PostgreSQL
5.	5.	5.	MongoDB

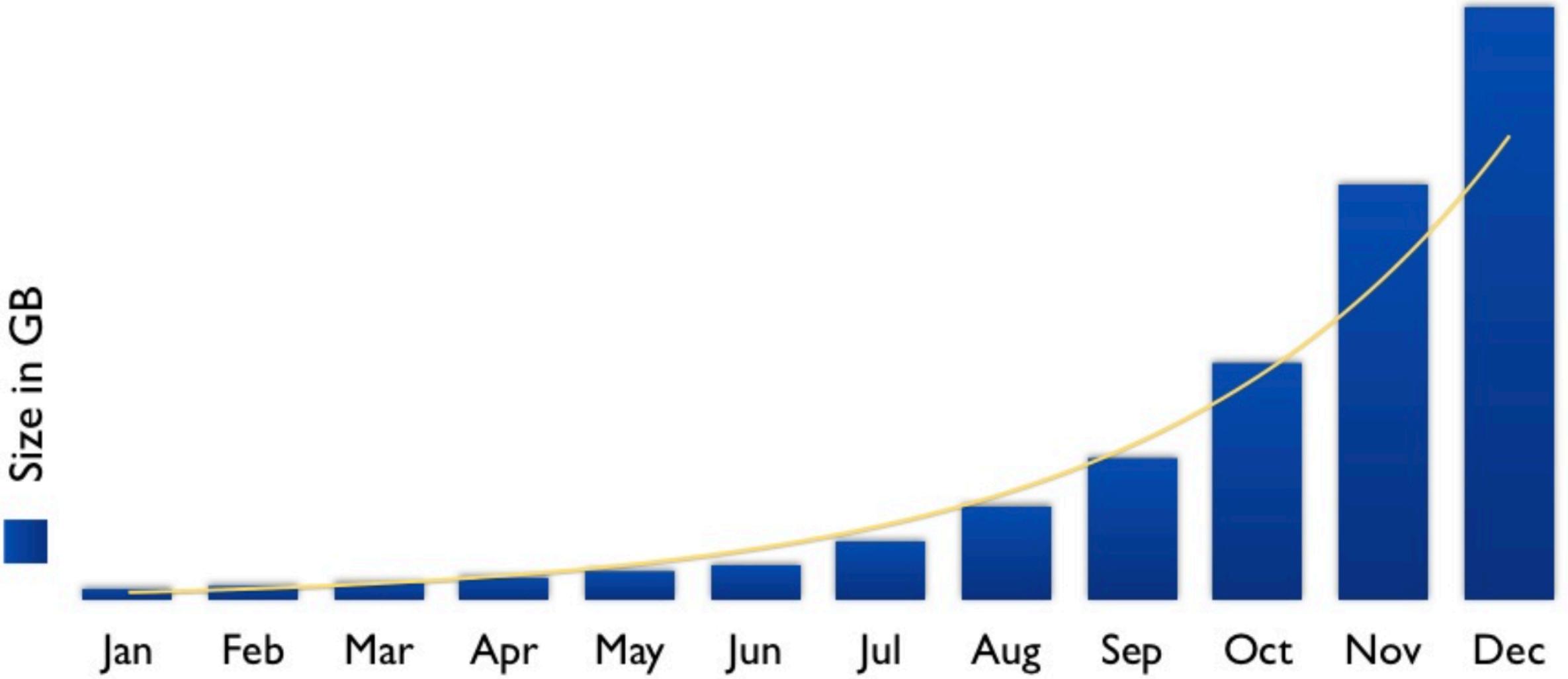
Client/Server model



Concurrency

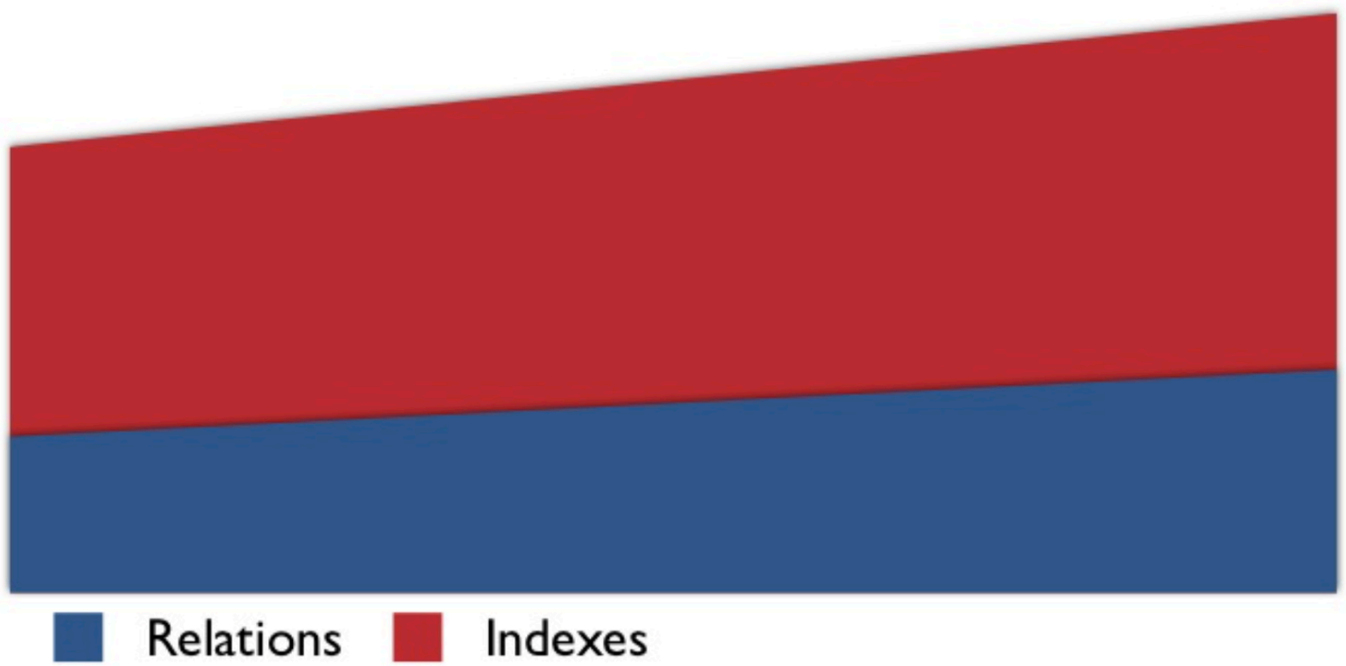


Data on disk



Data on disk

Tables sizes and indexes



Constraints

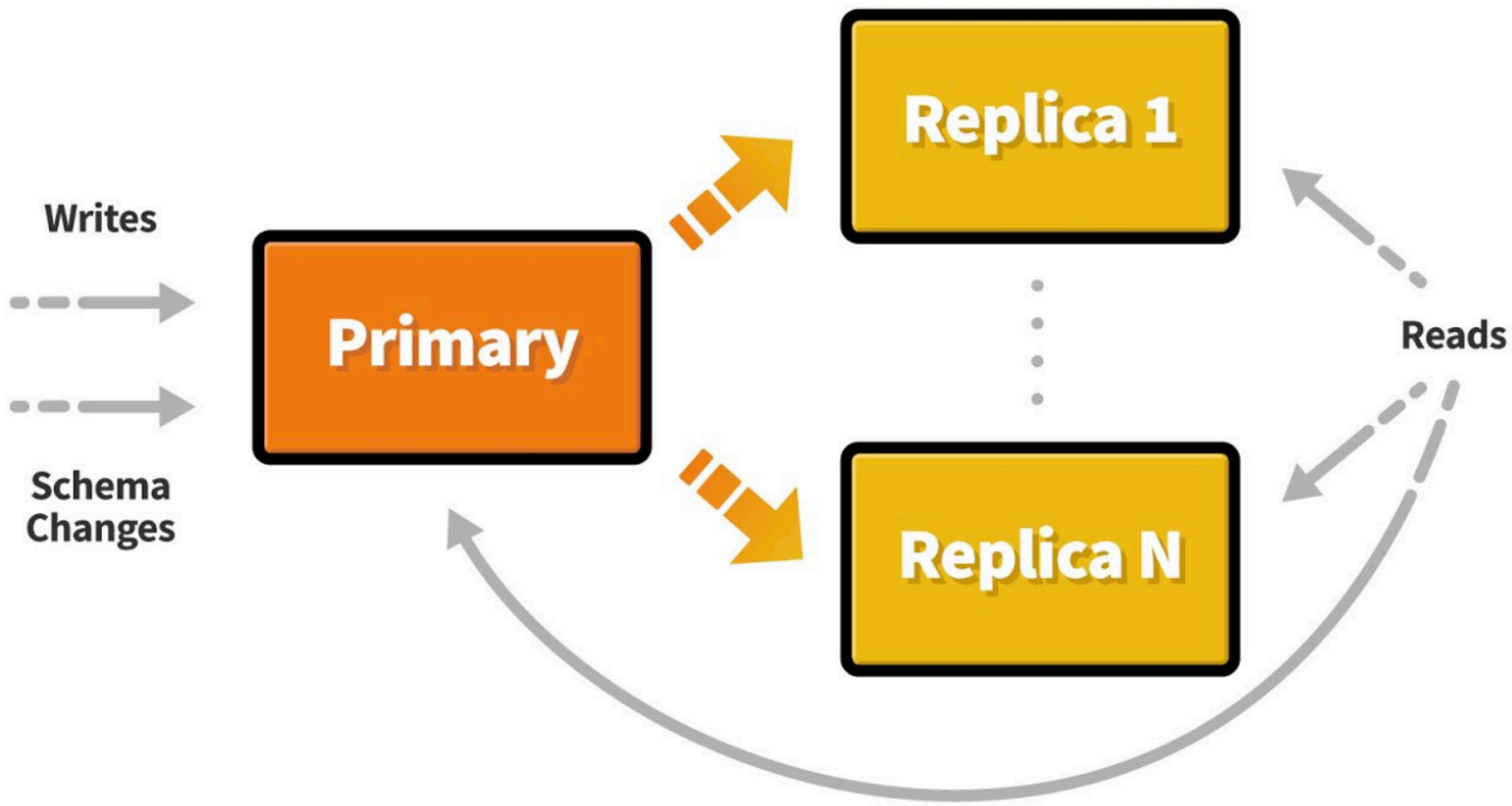
Available memory
Disk speed
I/O speed

Scalability

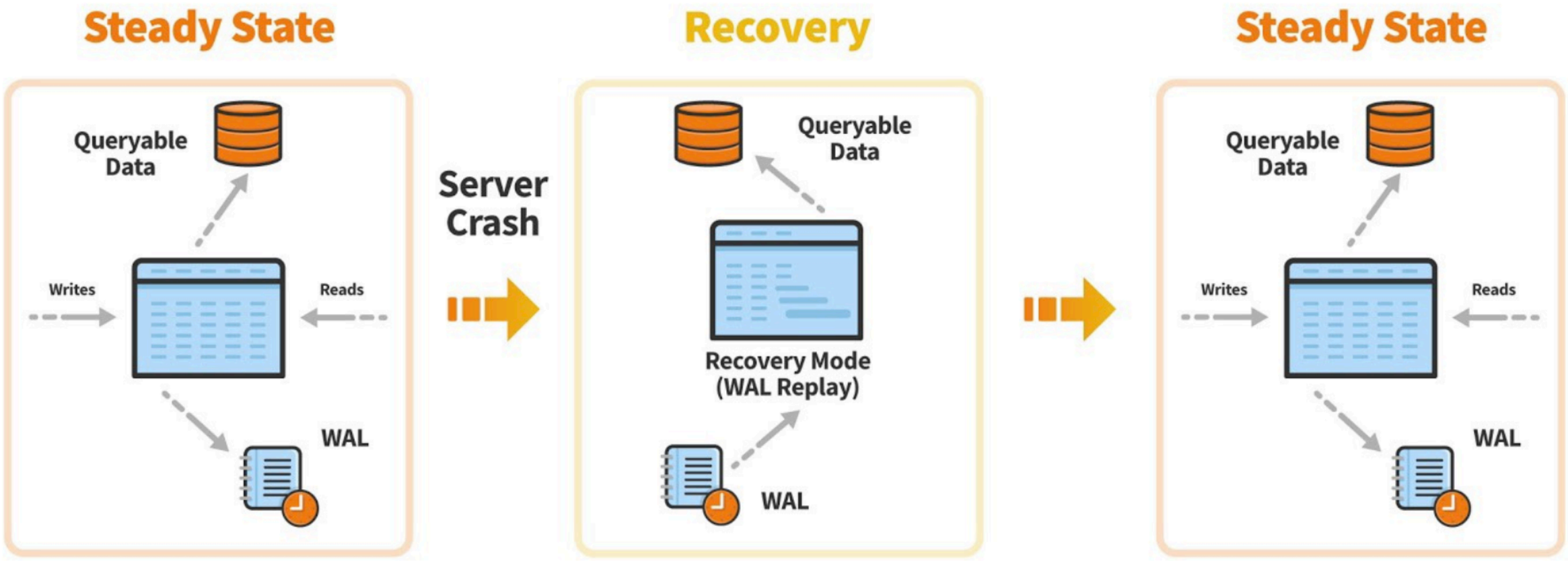
Higher insert performance
Higher read performance
High availability

How an (N+1) node PostgreSQL cluster works ?

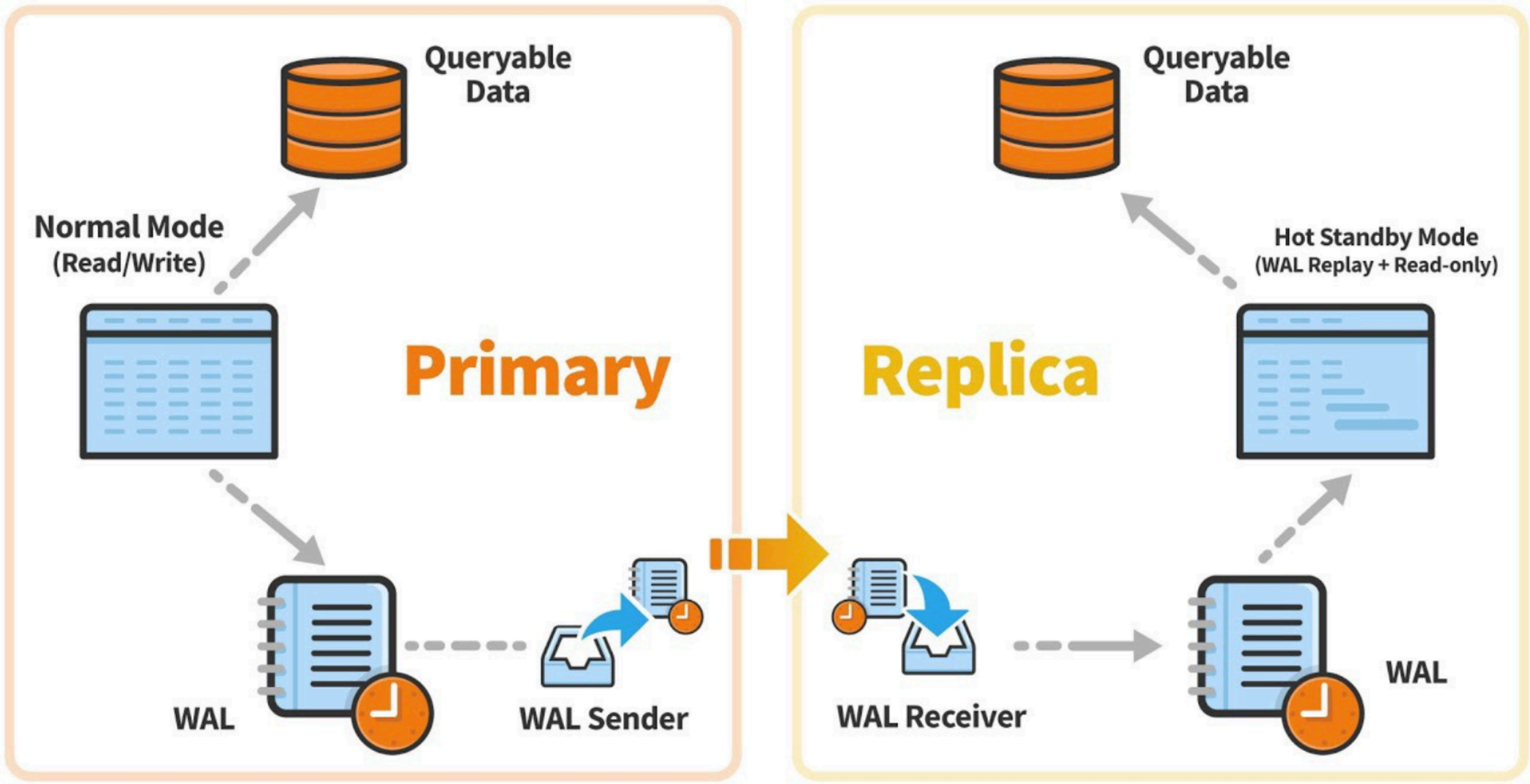
Using streaming replication



WAL (Write Ahead Log)



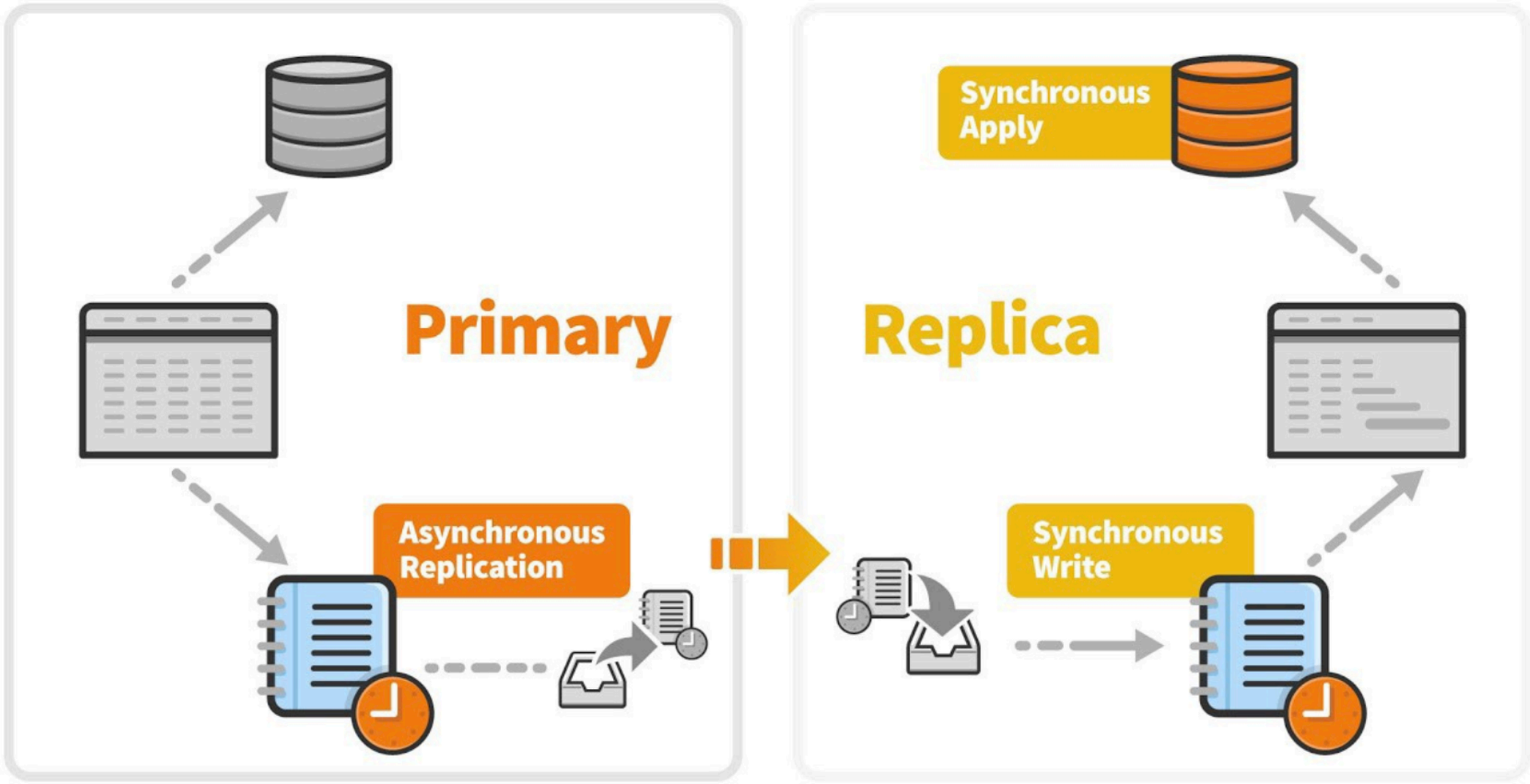
WAL and Replication



Replication modes

Asynchronous replication
Synchronous write replication
Synchronous apply replication

Replication modes



Performance

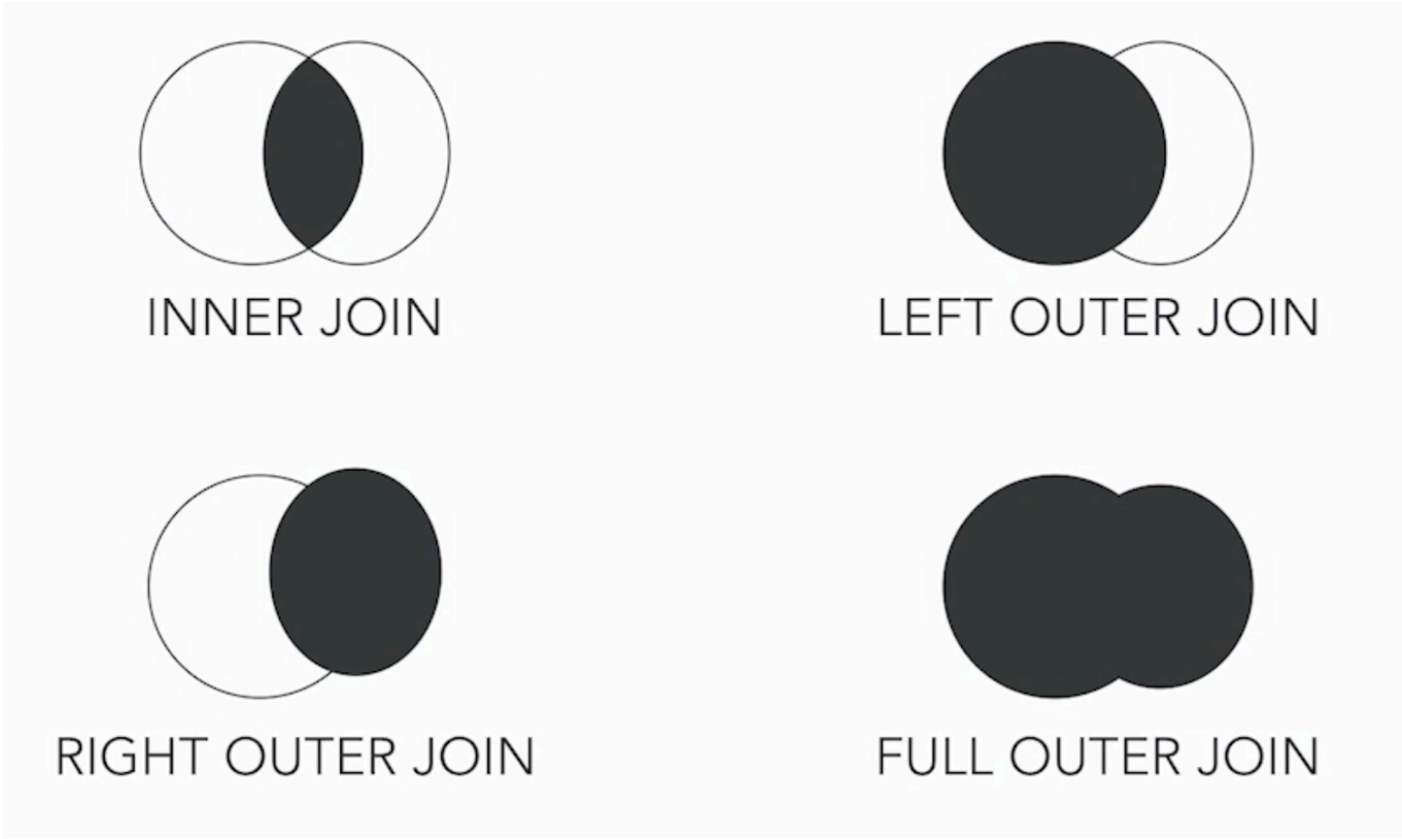
Replication mode	Write performance	Read consistency	Data loss
Async replication	Highest	Weakest, eventually consistent	Highest risk
Sync write replication	Medium	eventually consistent BUT less lag	Low risk
Sync apply replication	Lowest	Strong consistency	Lowest risk

SQL Tuning
with PostgreSQL

Tuning ?

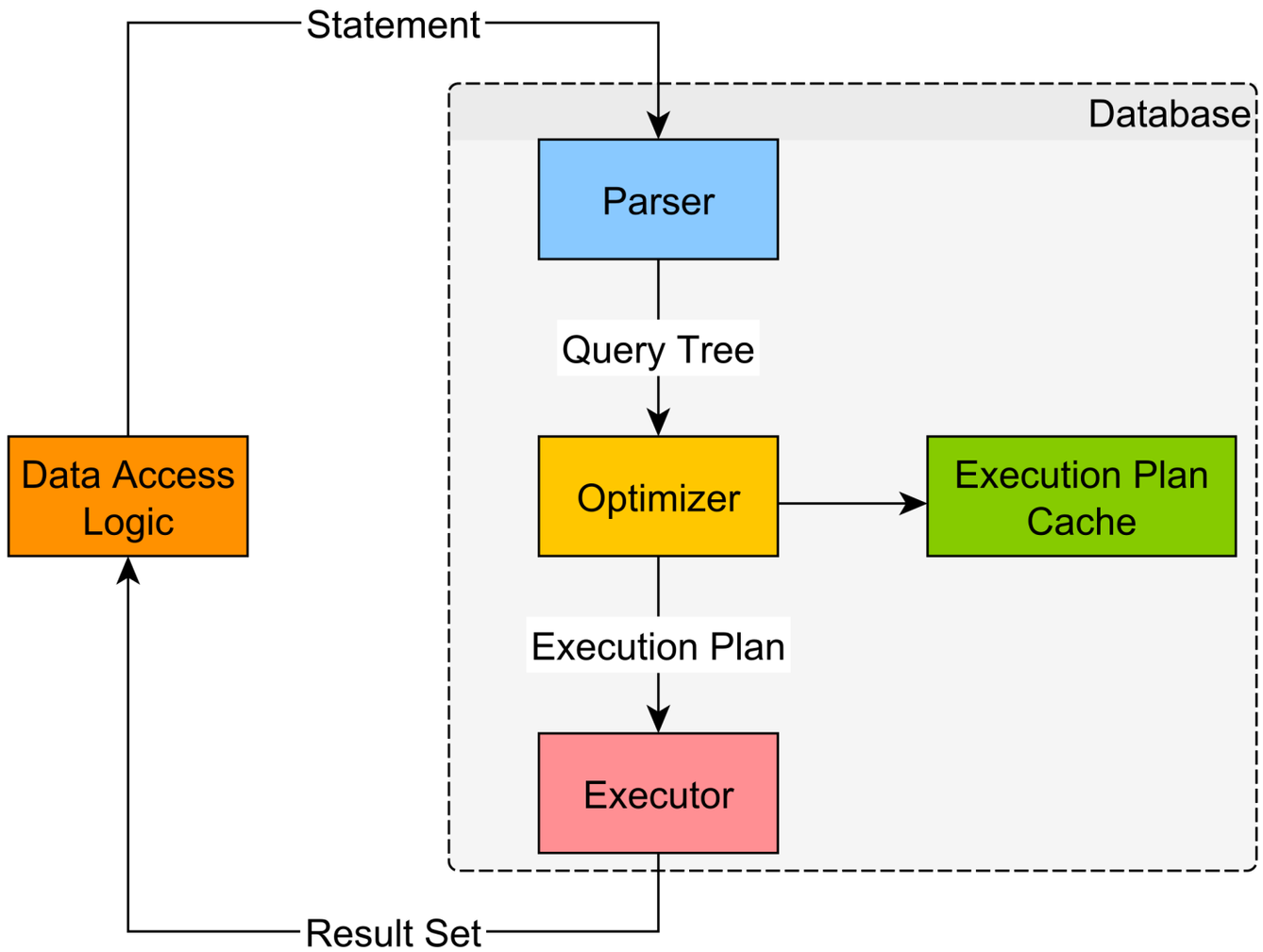
Reduce query response time with query tuning

Explain query ?



SQL use execute plan

Set of step that scan, filter and join data



Efficient execution plan ?

Good or Bad ?

Good

Using index to find data

Bad

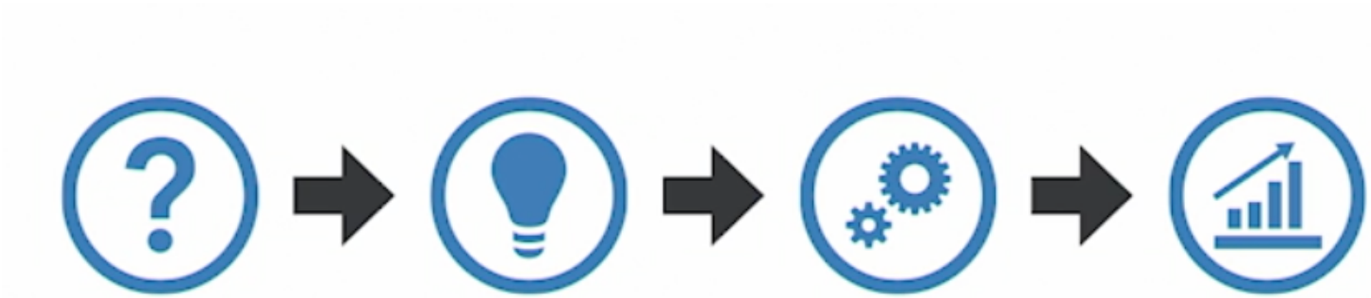
Scan 1M rows to find 100 rows

Step to learn

Understand query plan steps

Understand trade-offs in way to implement query plan

Learn techniques that lead to efficient plans



Basic

Scanning tables and indexes
Joining tables
Partition data

Scanning tables

	customer_id	fname	lname	last_purchase_date	status_level
✔	1234	Alice	Smith	23-Jan-2016	silver
✔	2345	Bob	Washington	14-July-2018	bronze
→	2352	Charlie	Johnson	20-Dec-2018	gold
	2678	Chendong	Mao	19-Dec-2017	gold
	3422	Javier	Valdez	23-Dec-2018	gold
	3577	Avery	Winston	5-Jan-2019	silver
	4240	Charles	Davis	15-Jan-2017	bronze
	6235	Vihaan	Patel	12-Nov-2018	gold
	2352	Katherine	Coltrane	04-Apr-2016	silver

Scanning tables

Scanning looks at each row

Fetch data block containing row

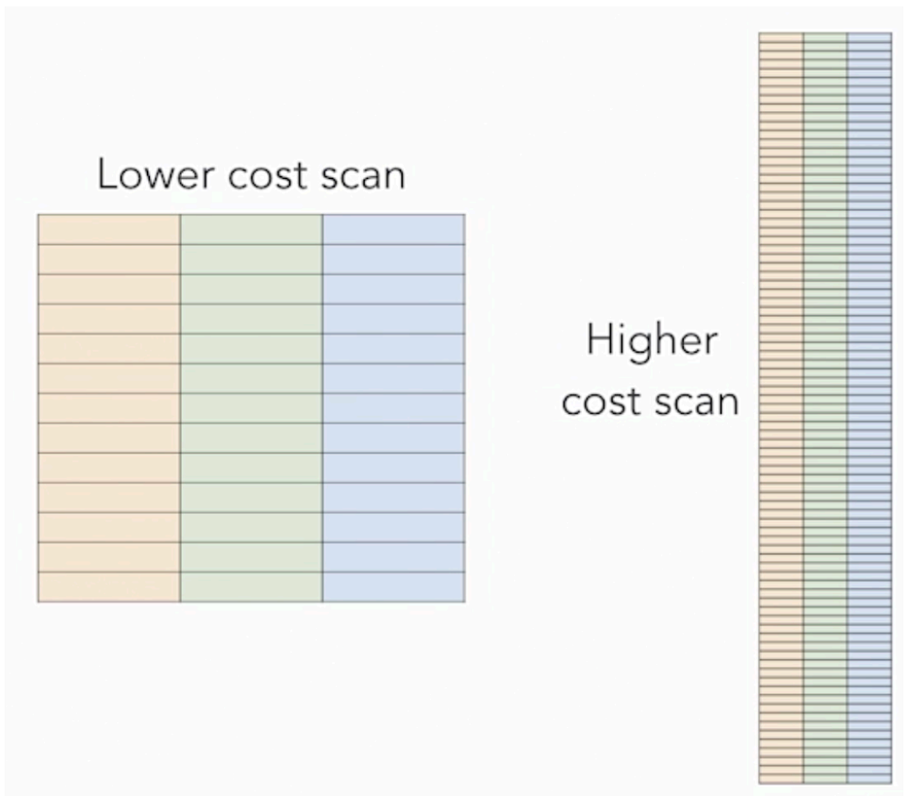
Apply filters or conditions

Cost of query based on number of rows in the table

Cost based on number of rows

Scanning on small tables is efficient

Scanning on large tables can be efficient if have few queries



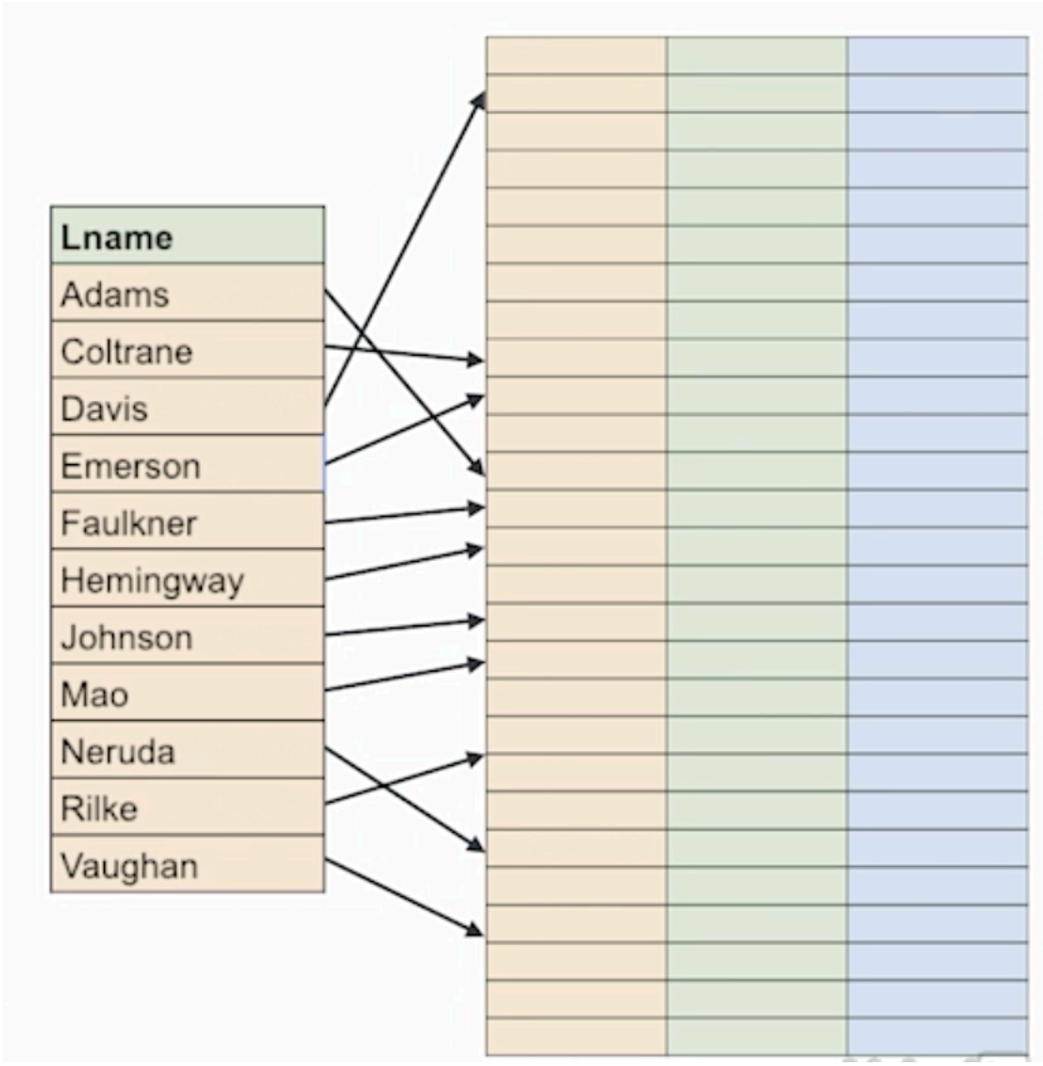
Indexing reduce full table scan

Indexes are ordered

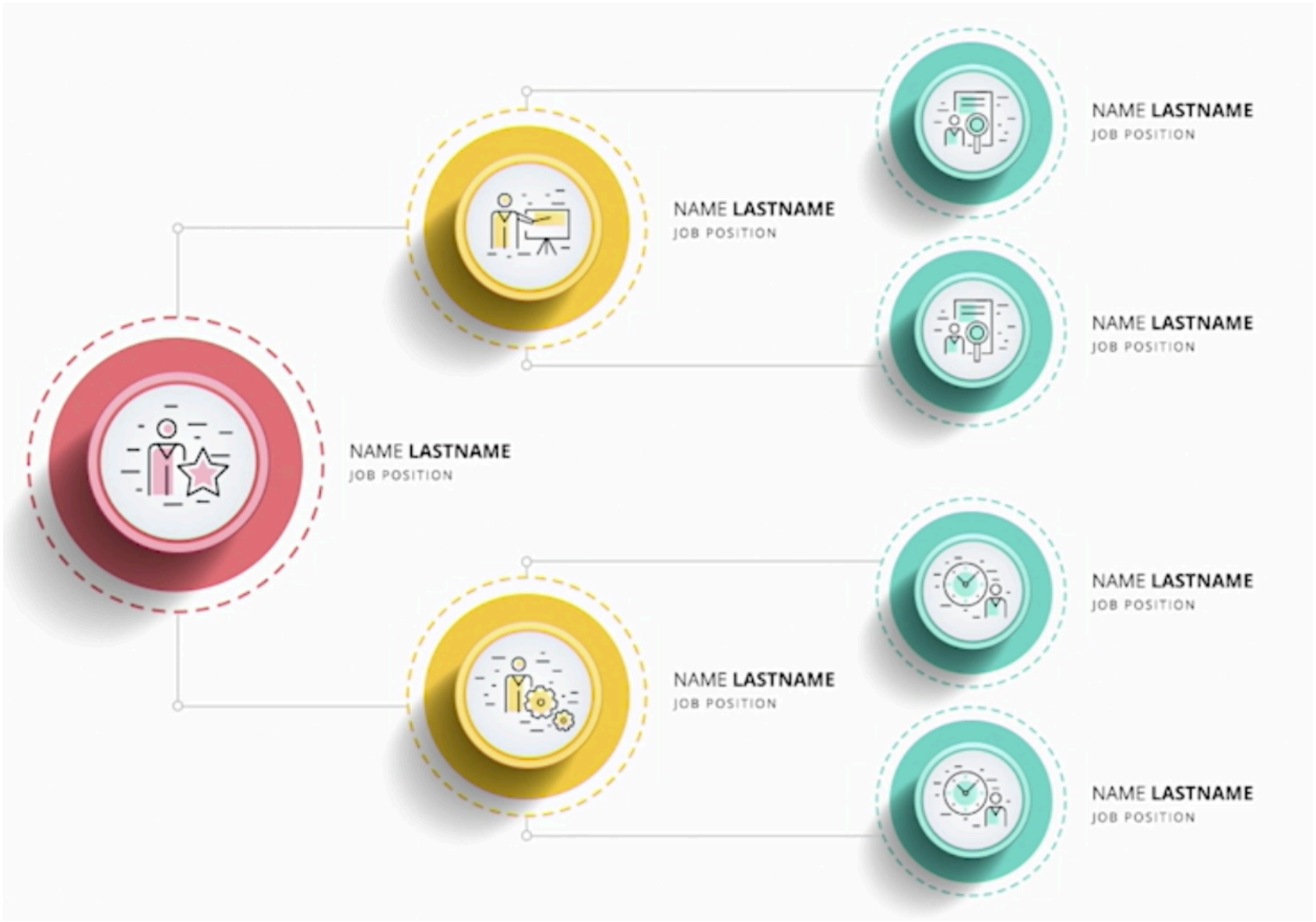
Faster to search index for an attribute value

Point to location of row

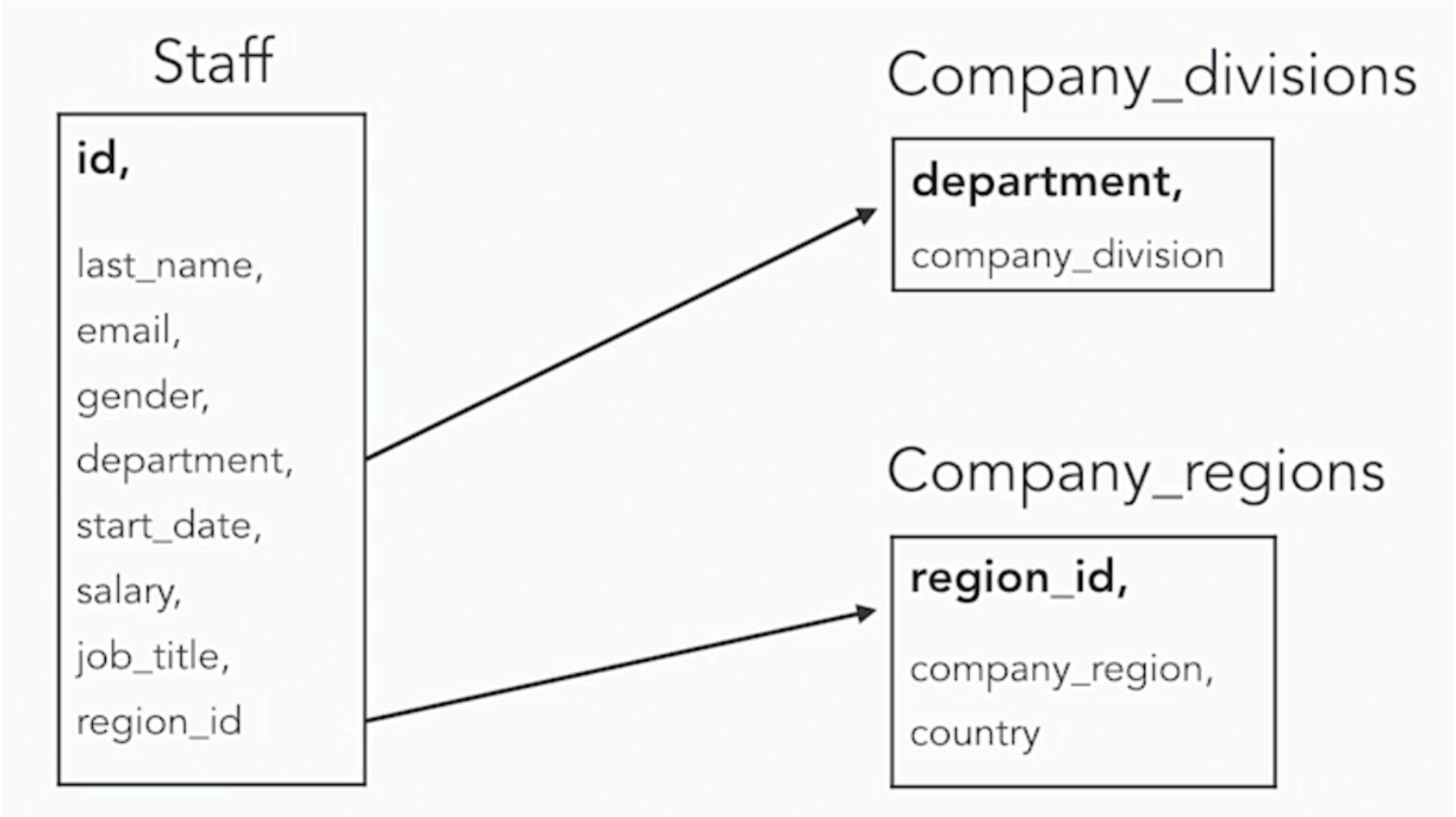
Indexing reduce full table scan



Indexes are optimised for search



Indexes are optimised for search



Purposes of indexes

Speed up access to data
Help enforce constraints
Indexes are ordered
Typically smaller than tables

Speeds

1

Main Memory

100 ns

2

Read 1 MB SSD

1,000,000 ns

(1 ms)

3

Read 1 MB HDD

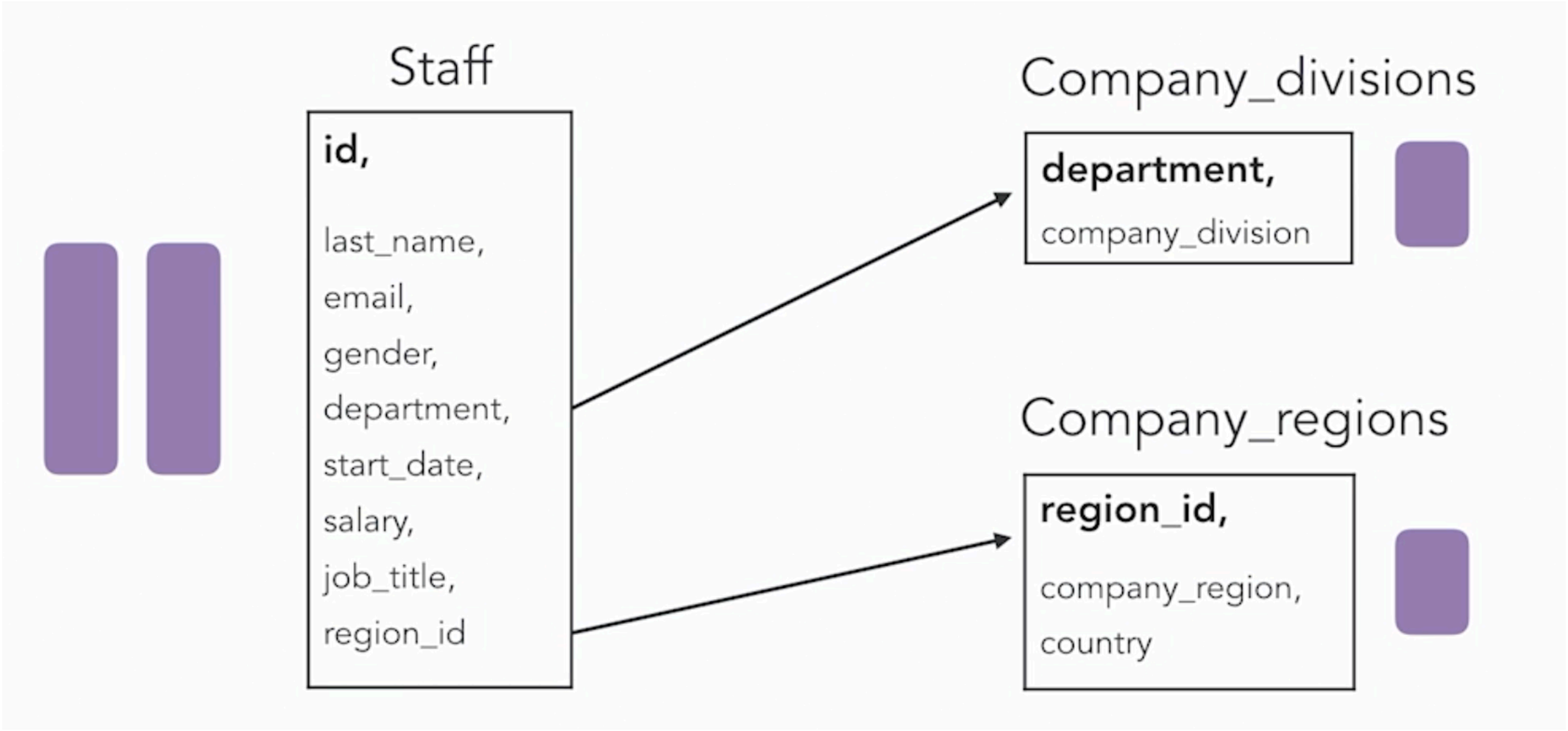
20,000,000 ns

(20 ms)

Implementation of indexes

Data structure separate from table
Sometime duplicate some data (e.g. key)
Organised differently than table data

Implementation of indexes



Types of indexes

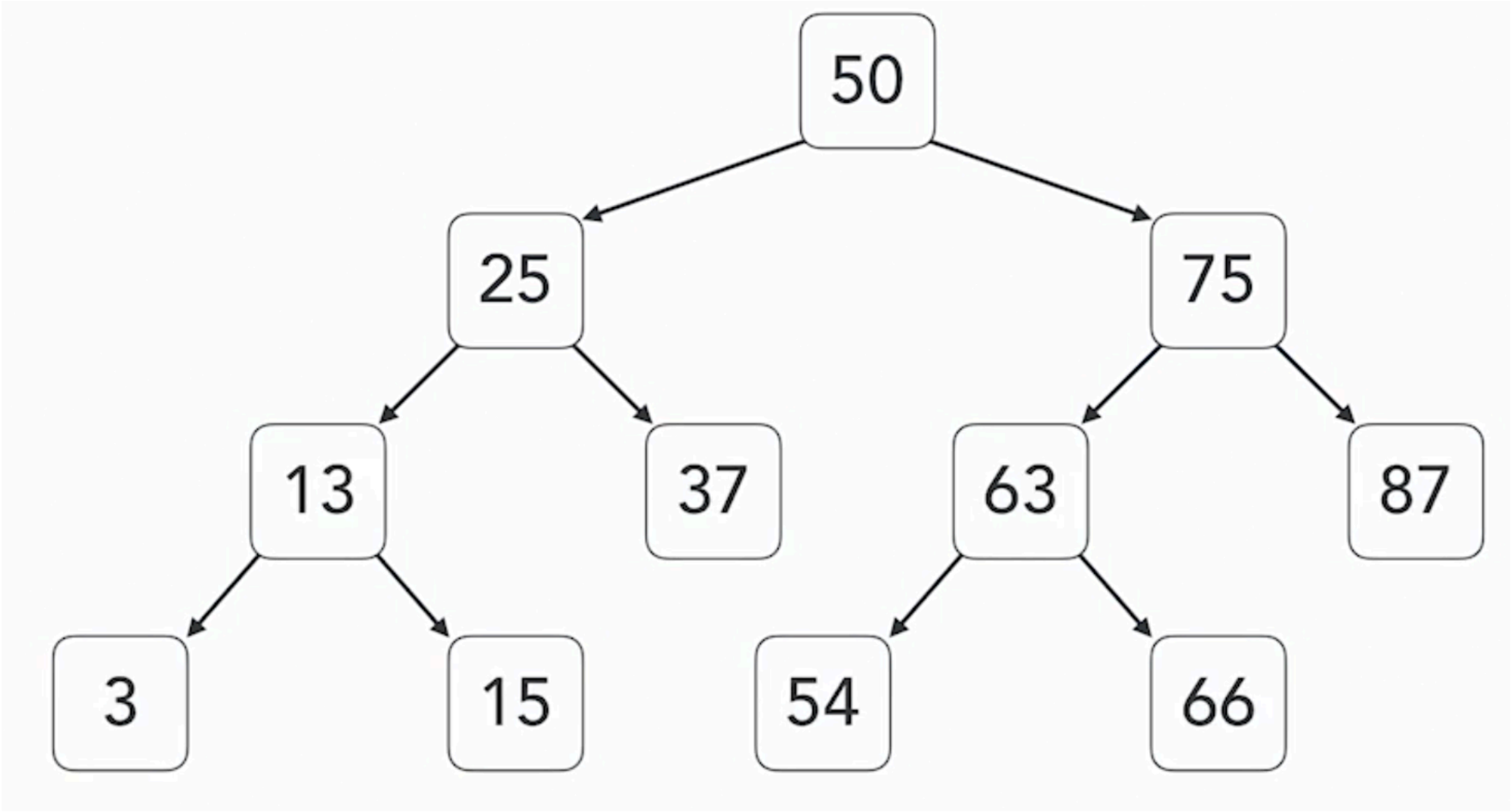
B-tree for equality and range query

Hash for equality query

Bitmap for inclusion

Specialize for geo-spatial or user-defined

B-tree indexes



B-tree indexes

Most common type of index

Use when large number of possible values in a column

Rebalances as need

Time to access is based on depth of tree

Bitmap indexes

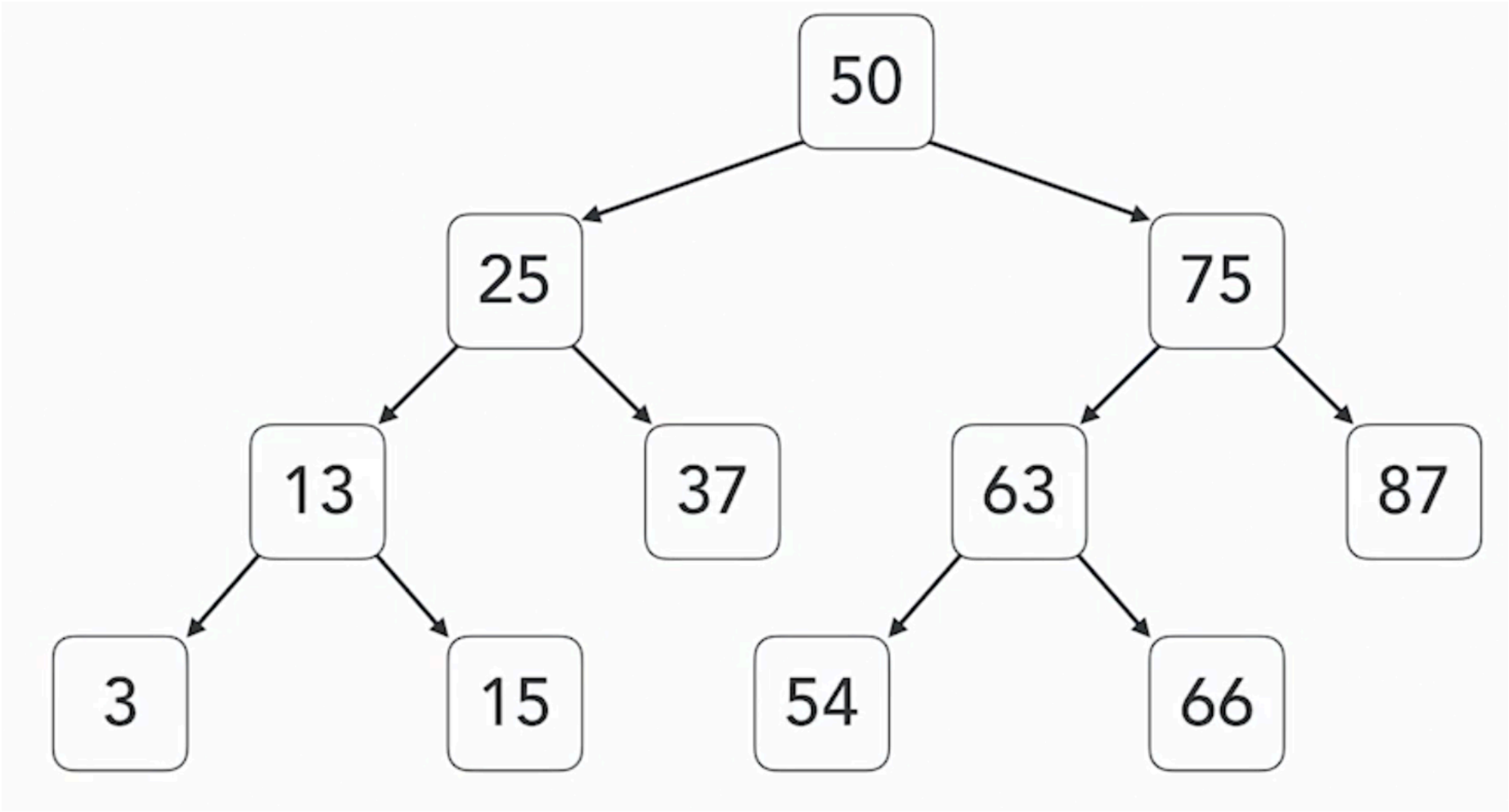
Use when small number of possible values in a column

Filter by bitwise operations (and, or, not)

Time to access is based on time to perform bitwise operation

Use for read intensive, few write

B-tree indexes



Hash indexes

1

Equality Only

Hash indexes used when "=" is used, but not for ranges of values.

2

Smaller Size Than B-Tree

Latest versions of PostgreSQL (10+) have improved hash indexes.

3

As Fast as B-Tree

Builds and lookups are comparable; advantage is size; may fit in memory.

Workshop

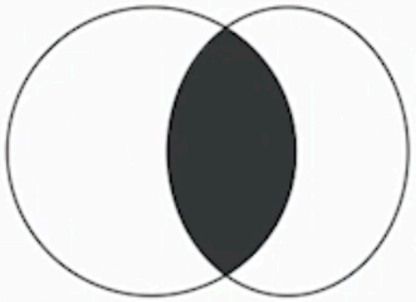
Joining tables

customer_id	fname	lname	last_purchase_date	status_level
1234	Alice	Smith	23-Jan-2016	silver
2345	Bob	Washington	14-July-2018	bronze
2352	Charlie	Johnson	20-Dec-2018	gold
2678	Chendong	Mao	19-Dec-2017	gold
3422	Javier	Valdez	23-Dec-2018	gold
3577	Avery	Winston	5-Jan-2019	silver
4240	Charles	Davis	15-Jan-2017	bronze
6235	Vihaan	Patel	12-Nov-2018	gold
2352	Katherine	Coltrane	04-Apr-2016	silver

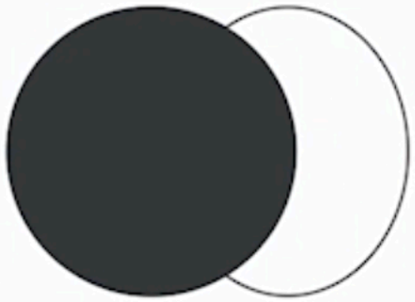
status_level	min_purchase_amt	discount_pct
gold	2500	7
silver	1500	5
bronze	500	3
platnimum	5000	10

customer_id	fname	lname	last_purchase_date	status_level	status_level	min_purchase_amt	discount_pct
2345	Bob	Washington	14-July-2018	bronze	bronze	500	3

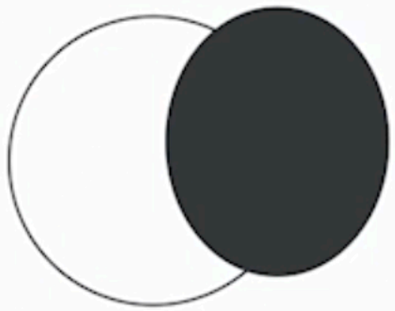
Types of joining



INNER JOIN



LEFT OUTER JOIN



RIGHT OUTER JOIN

FULL OUTER JOIN

Inner join

Most common join

Return rows from both tables that have corresponding row in other table

```
SELECT
    *
FROM
    company_region cr,
INNER JOIN
    staff s
ON
    cr.region_id = s.region_id
```

Left outer join

Return all rows from left table

Return rows from right table that have matching key

```
SELECT
    *
FROM
    company_region cr,
LEFT OUTER JOIN
    staff s
ON
    cr.region_id = s.region_id
```

Right outer join

Return all rows from right table

Return rows from left table that have matching key

```
SELECT
    *
FROM
    company_region cr,
RIGHT OUTER JOIN
    staff s
ON
    cr.region_id = s.region_id
```

Full Outer join

Return all rows from both tables

```
SELECT
      *
FROM
      company_region cr,
FULLER OUTER JOIN
      staff s
ON
      cr.region_id = s.region_id
```

Joining

1

Prefer INNER JOINs

Often fastest

2

OUTER and FULL JOINs

Require additional steps in addition to INNER JOIN

3

When need to NULLs

LEFT, RIGHT, and FULL OUTER JOINS used when need to NULLs

How to match rows ?

Foreign keys in one table
Primary key in another table
How to find matching keys ?

customer_id	fname	lname	last_purchase_date	status_level
1234	Alice	Smith	23-Jan-2016	silver
2345	Bob	Washington	14-July-2018	bronze
2352	Charlie	Johnson	20-Dec-2018	gold
2678	Chendong	Mao	19-Dec-2017	gold
3422	Javier	Valdez	23-Dec-2018	gold
3577	Avery	Winston	5-Jan-2019	silver
4240	Charles	Davis	15-Jan-2017	bronze
6235	Vihaan	Patel	12-Nov-2018	gold
2352	Katherine	Coltrane	04-Apr-2016	silver

status_level	min_purchase_amt	discount_pct
gold	2500	7
silver	1500	5
bronze	500	3
platnimum	5000	10

Joining tables

1

Nested Loop Join

Compare all rows in both tables to each other.

2

Hash Join

Calculate hash value of key and join based on matching hash values.

3

Sort Merge Join

Sort both tables and then join rows while taking advantage of order.

Workshop

Sub-queries vs Join

Sub-queries

Returns a value from a related tables

```
SELECT
  s.id, s.last_name, s.department,
  (SELECT
      company_regions
   FROM
      company_regions cr
   WHERE
      cr.region_id - s.region_id) region_name
FROM
  staff s
```

Sub-queries vs Join

Same logical outcome

SQL queries specify **“what”**

More than one way to express the same thing

Sub-queries vs Join

1

Conventional Wisdom

Always use joins; they are more efficient.

2

Improved Query Plans

Query plan builders are more effective at optimizing subqueries.

3

Maximize Clarity

Both will work well in many cases. Opt for what makes intention clear.

Partition data ?

Storing table data in multiple sub-tables
Called “**Partition**”

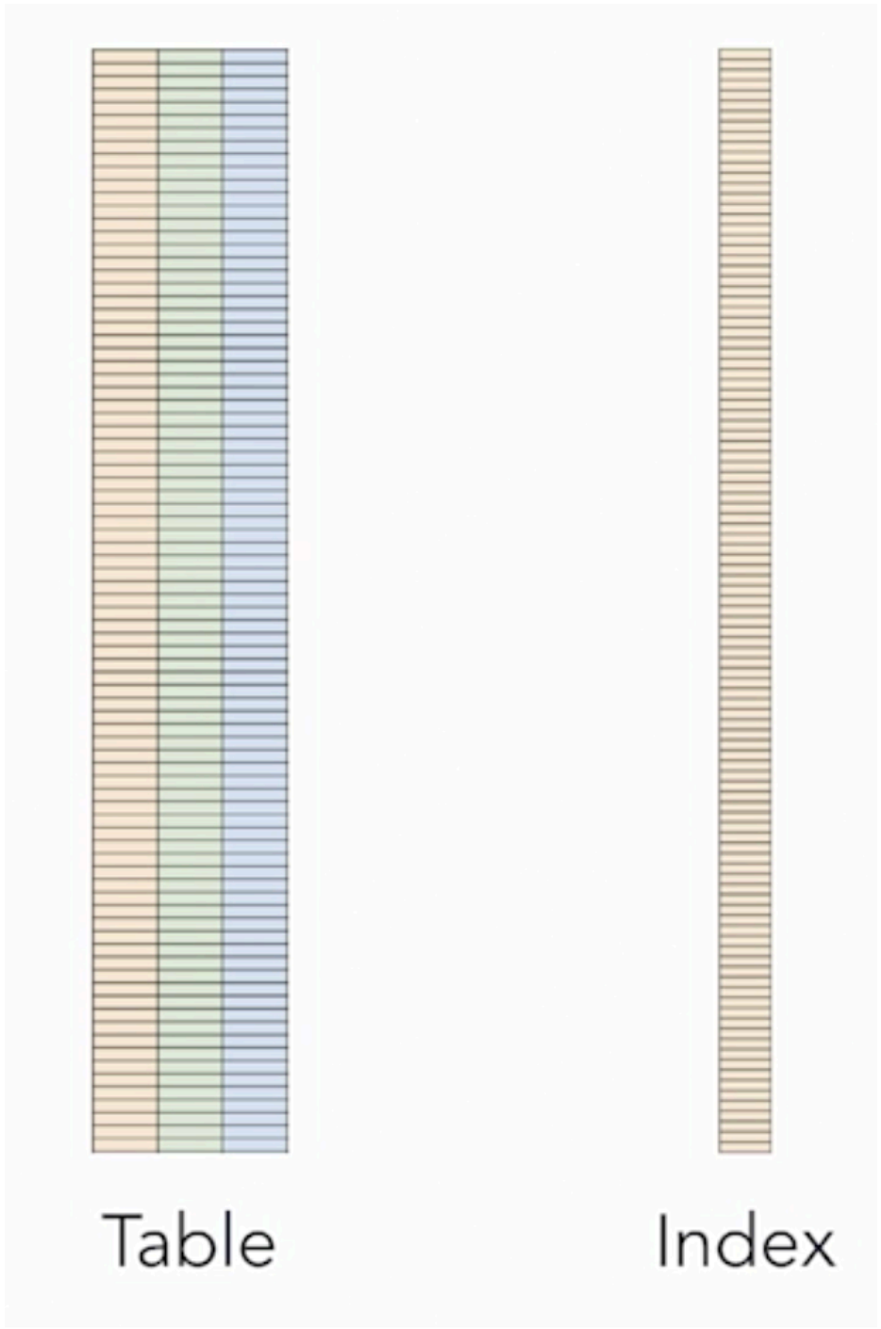
Partition data ?

Used to improve query, load and delete operation

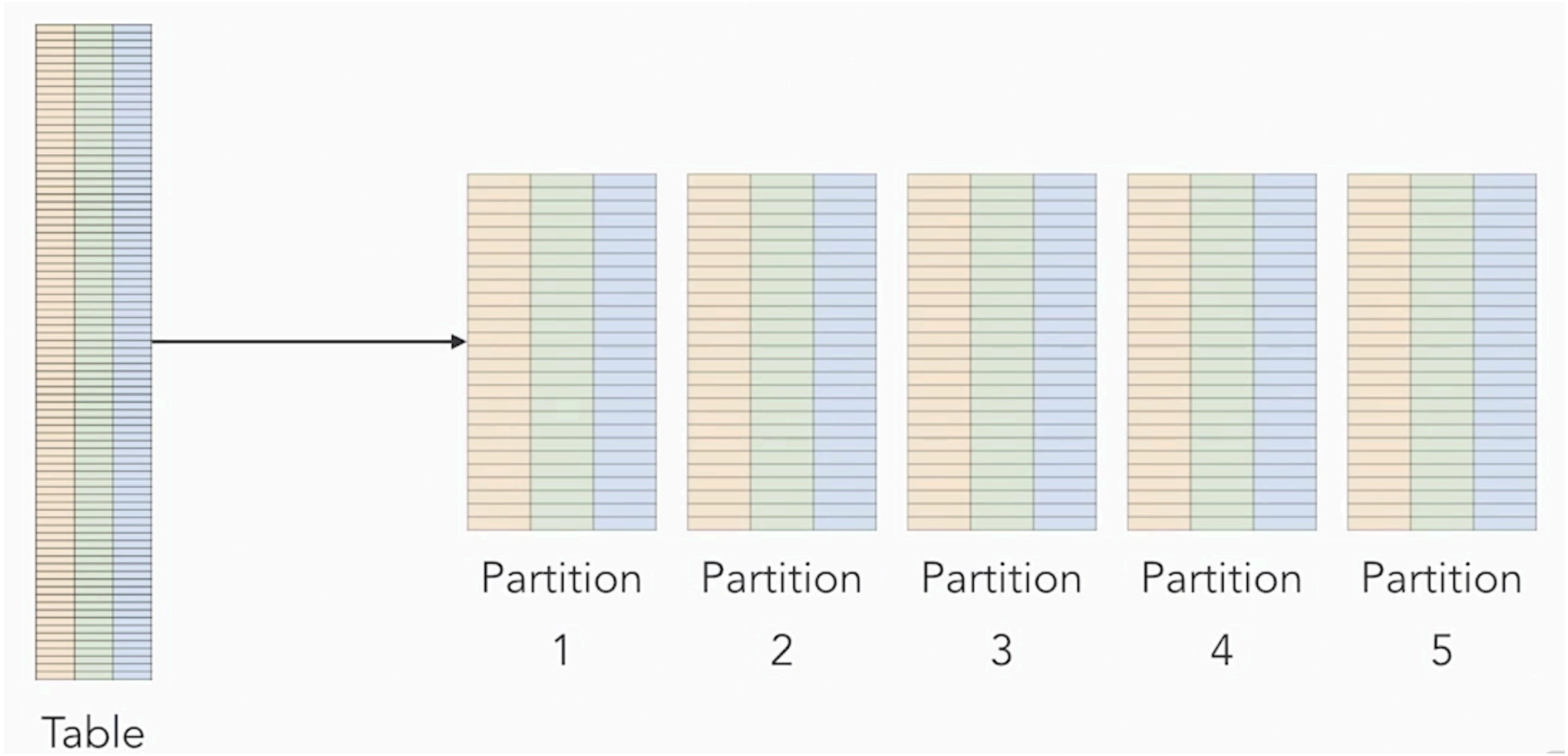
Used for **large tables**

Expensive

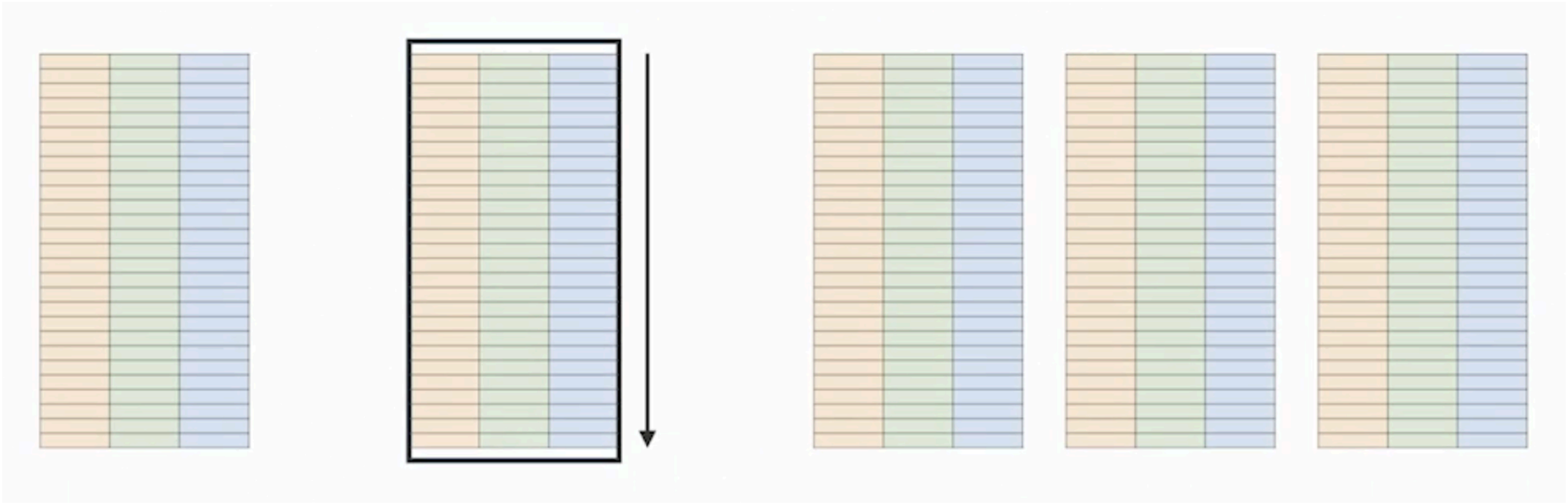
Large table = Large index



Partition



Faster Scan

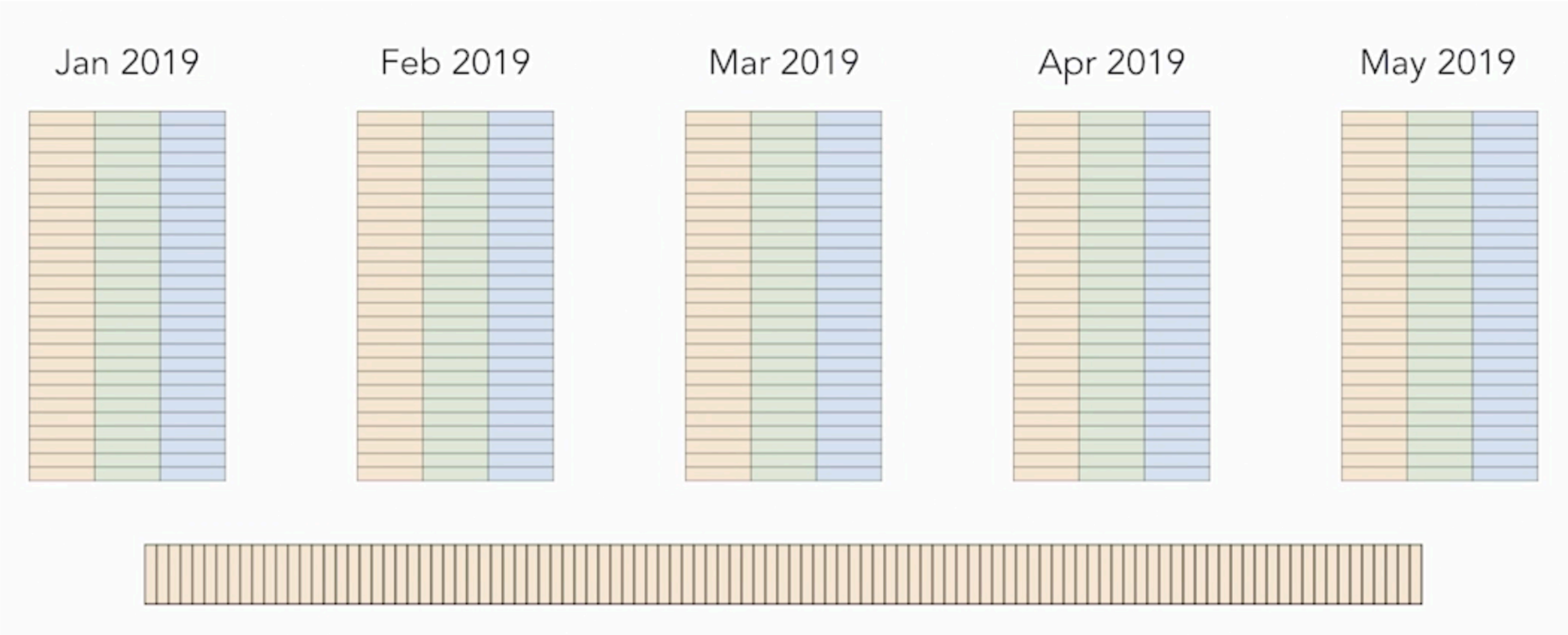


Partition keys

By range
By list
By hash

Local index in each partition

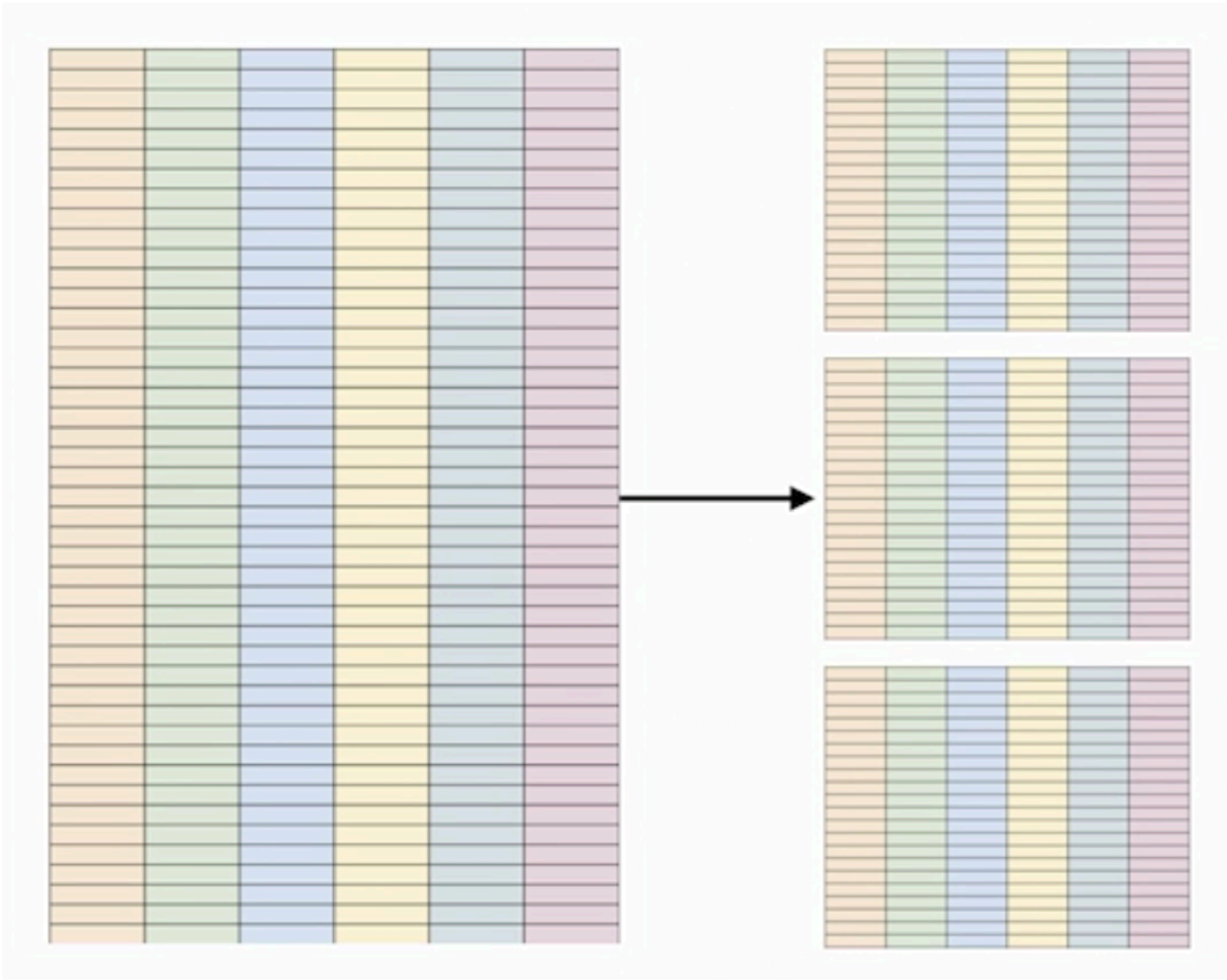
Global index for all partitions



Partition types

Horizontal
Vertical

Horizontal partition



Horizontal partition

Large tables can be difficult to query efficient
Split tables by rows in to partitions
Treat each partition like a table

Types of horizontal partition

Partition by range
Partition by list
Partition by hash

Partition by range

Partition by date
Partition by numeric
Partition by alphabet

By non-overlapping keys

Partition by range

1

Partition Key

Determines which partition is used for data

2

Partition Bounds

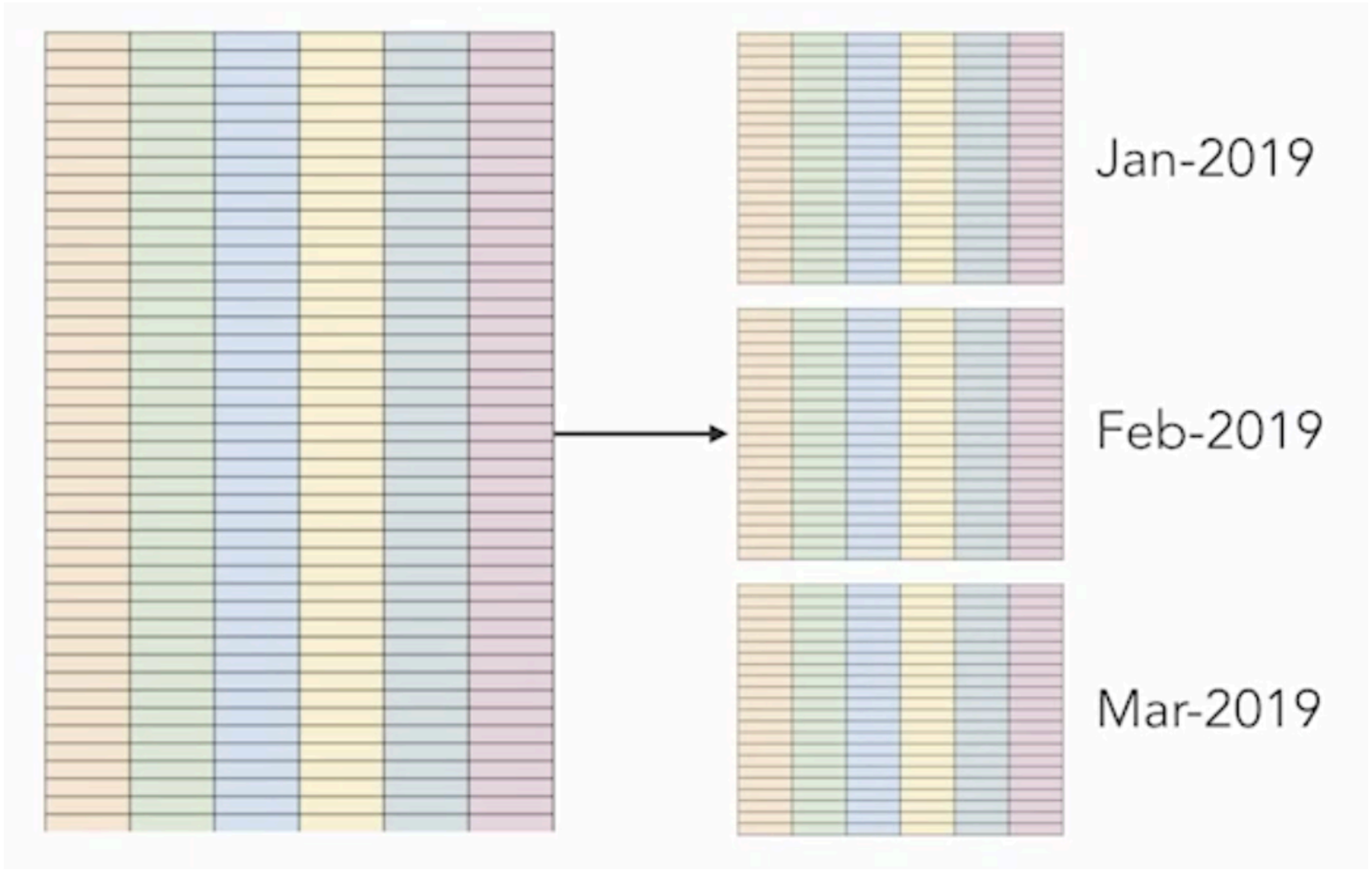
Minimum and maximum values allowed in the partition

3

Constraints

Each partition can have its own indexes, constraints, and defaults

Example partition by date



IoT data

Partition by date

```
CREATE TABLE iot_measurement
  ( location_id int not null,
    measure_date date not null,
    temp_celcius int,
    rel_humidity_pct int)
 PARTITION BY RANGE (measure_date);
```

IoT data

Partition by date

```
CREATE TABLE iot_measurement
  ( location_id int not null,
    measure_date date not null,
    temp_celcius int,
    rel_humidity_pct int)
 PARTITION BY RANGE (measure_date);
```

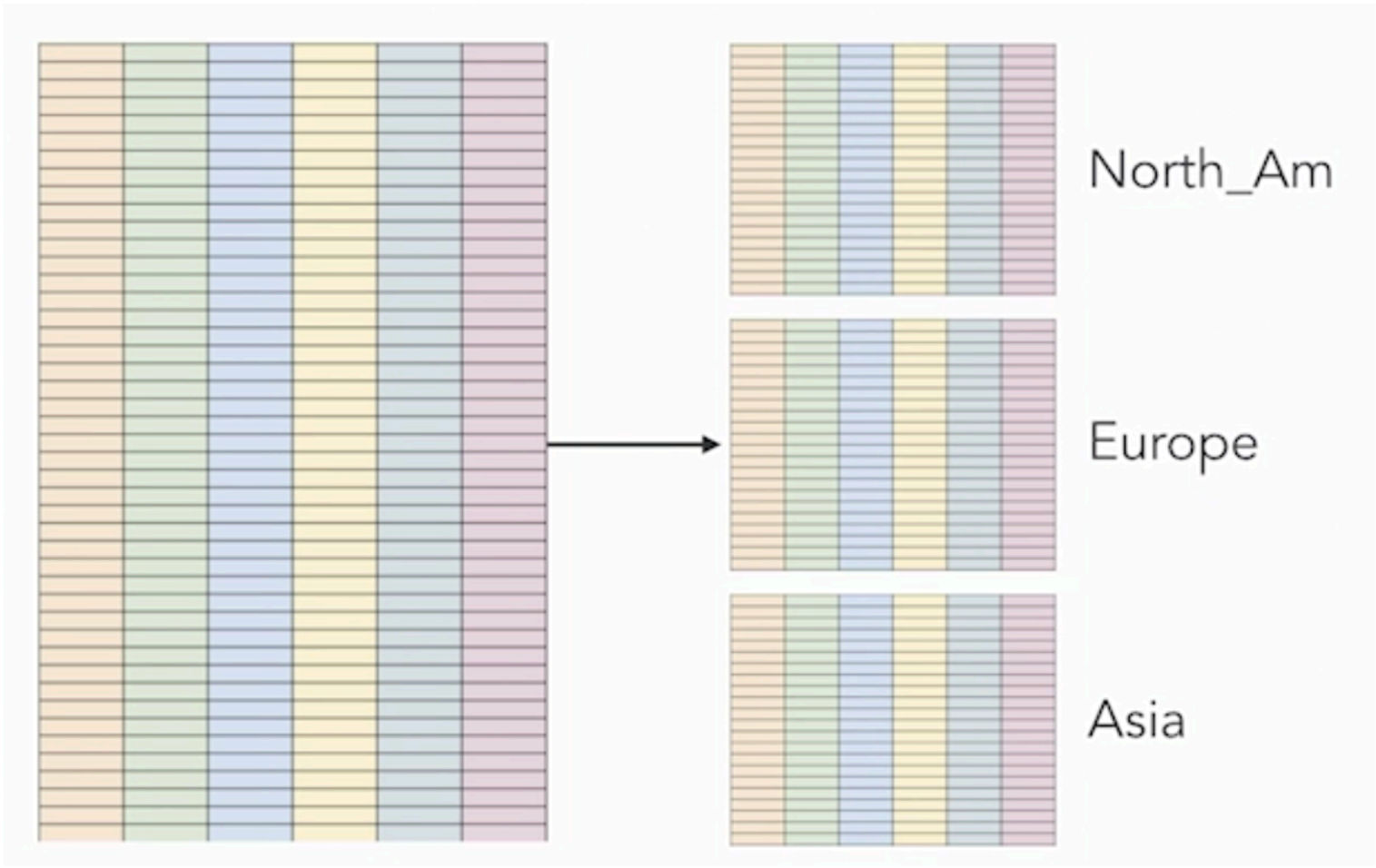
```
CREATE TABLE iot_measurement_wk1_2019 PARTITION OF iot_measurement
   FOR VALUES FROM ('2019-01-01') TO ('2019-01-08');

CREATE TABLE iot_measurement_wk2_2019 PARTITION OF iot_measurement
   FOR VALUES FROM ('2019-01-08') TO ('2019-01-15');

CREATE TABLE iot_measurement_wk3_2019 PARTITION OF iot_measurement
   FOR VALUES FROM ('2019-01-15') TO ('2019-01-22');
```

When to use ?

Query latest data
Comparative queries
Report within range
Drop data after a period of time

Workshop

Partition by list



Partition by list

Partition on value of list of values

By non-overlapping keys

Partition by list

1

Partition Key

Determines which partition is used for data

2

Partition Bounds

List of values for a partition

3

Constraints

Each partition can have its own indexes, constraints, and defaults

Example :: Product catalog

Partition by list of product catalog

```
CREATE TABLE products
    (prod_id    int not null,
     prod_name text not null,
     prod_short_descr text not null,
     prod_long_descr text not null,
     prod_category varchar)
PARTITION BY LIST (prod_category);
```

Example :: Product catalog

Partition by list of product catalog

```
CREATE TABLE products
   (prod_id    int not null,
    prod_name text not null,
    prod_short_descr text not null,
    prod_long_descr text not null,
    prod_category varchar)
PARTITION BY LIST (prod_category);
```

```
CREATE TABLE product_clothing PARTITION OF products
 FOR VALUES IN ('casual_clothing', 'business_attire', 'formal_clothing');

CREATE TABLE product_electronics PARTITION OF products
  FOR VALUES IN ('mobile_phones', 'tablets', 'laptop_computers');

CREATE TABLE product_kitchen PARTITION OF products
  FOR VALUES IN ('food_processor', 'cutlery', 'blenders');
```

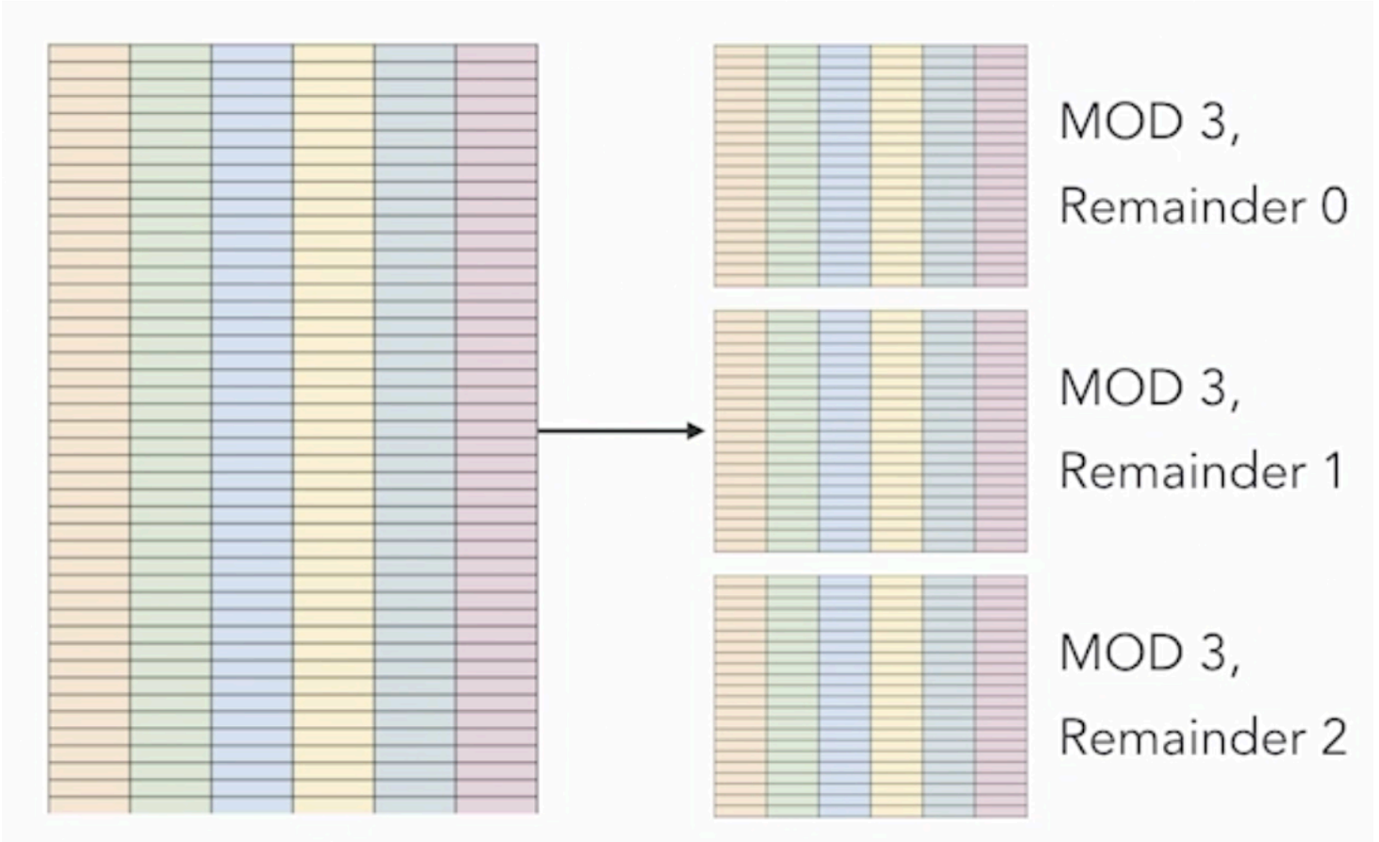
When to use ?

Data logically groups into subgroups
Often query within subgroups
Data not time oriented

Workshop

Partition by hash

Partition on modulus of hash of partition keys



Partition by hash

1

Partition Key

Determines which partition is used for data

2

Modulus

Number of partitions

3

Availability

PostgreSQL 11+, Oracle, MySQL

Example :: Web Path Analytic

Hash function on ci_id

```
CREATE TABLE customer_interaction
    (ci_id int not null,
     ci_url text not null,
     time_at_url int not null,
     click_sequence int not null)
 PARTITION BY HASH (ci_id);
```

Example :: Web Path Analytic

Hash function on ci_id

```
CREATE TABLE customer_interaction
   (ci_id int not null,
    ci_url text not null,
    time_at_url int not null,
    click_sequence int not null)
PARTITION BY HASH (ci_id);
```

```
CREATE TABLE customer_interaction_1 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 0);
CREATE TABLE customer_interaction_2 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 1);
CREATE TABLE customer_interaction_3 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 2);
CREATE TABLE customer_interaction_4 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 3);
CREATE TABLE customer_interaction_5 PARTITION OF customer_interaction
  FOR VALUES WITH (MODULUS 5, REMAINDER 4);
```

When to use ?

Data doesn’t logically group into subgroups
Distribute data across partitions
No need for subgroup-operations

Workshop

Benefits

Limit scans to subset of partitions
Local indexes for each partition
Efficient for add and delete data

Use cases

1

Data Warehouses

- Partition on time
- Query on time
- Delete by time

2

Timeseries

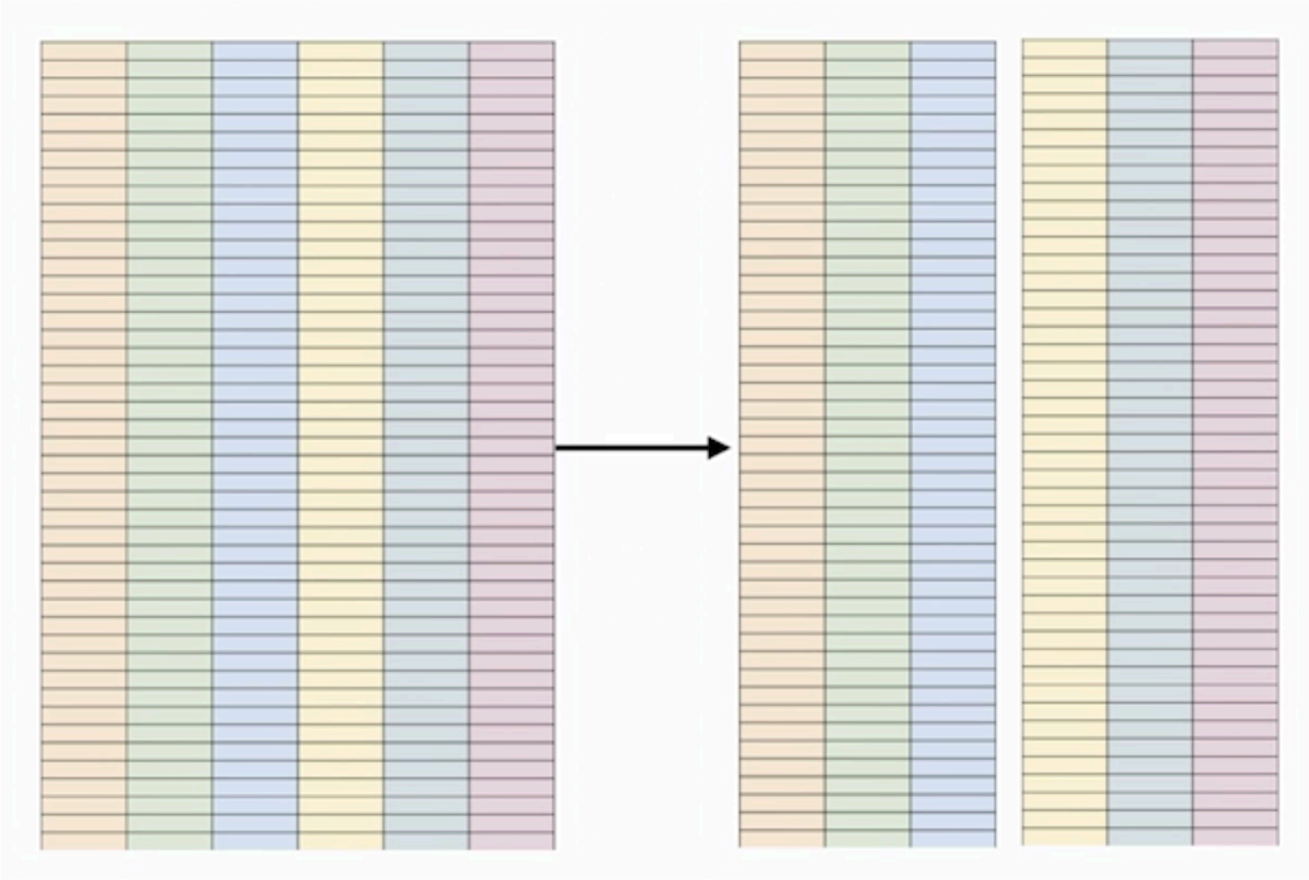
- Most likely query latest data
- Summarize data in older partitions

3

Naturally Partition Data

- Retailer, by geography
- Data science, by product category

Vertical partition



Vertical partition

Implement as separate tables

No partitioning-specific definitions are required

Vertical partition

Separate columns into multiple tables
Keep frequently queried columns together
Used same primary key in all tables

Benefits

Increase number of rows in data block
Global indexes for each partition
Reduce I/O

Use cases

1

Data Warehouses

Partition on groups of attributes

2

Many Attributes

Wide variety of products, each with different attributes

3

Data Analytics

Statistics on subset of attributes; after factor analysis

Workshop