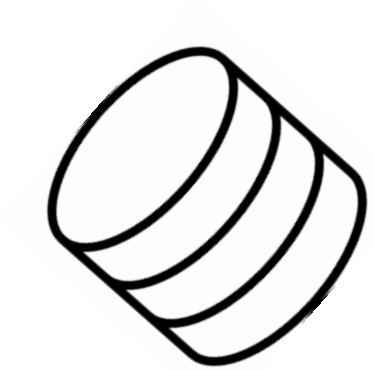
Database stuff

Indexes, Replication, Partition, Sharding



Database indexes

Indexes: what’s the point?

Suppose we have a database table:

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Indexes: what's the point?

Suppose we have a database table:

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Search on **ID** field that contains unique values will require on average $\mathrm{N} / 2$ cell accesses to find a value with Linear Search.

Indexes: what's the point?

Suppose we have a database table:

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Search on **ID** field that contains unique values will require on average $\mathrm{N} / 2$ cell accesses to find a value with Linear Search. Since an **ID** field is also sorted, the Binary Search can be applied, reducing the amount of accesses to $\log_2 \mathrm{N}$.

Indexes: what's the point?

Suppose we have a database table:

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Search on **ID** field that contains unique values will require on average $N / 2$ cell accesses to find a value with Linear Search. Since an **ID** field is also sorted, the Binary Search can be applied, reducing the amount of accesses to $\log_2 N$.

Indexes: what's the point?

Suppose we have a database table:

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Search on **ID** field that contains unique values will require on average $\mathrm{N} / 2$ cell accesses to find a value with Linear Search. Since an **ID** field is also sorted, the Binary Search can be applied, reducing the amount of accesses to $\log_2 \mathrm{N}$.

First Name field is not sorted, so Binary Search is impossible, so $\mathrm{N} / 2$ operations will be required.
First Name is also not unique, so search will need to access *all* cells in order to find a value.

Indexes: what's the point?

Suppose we have a database table:

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Search on **ID** field that contains unique values will require on average $\mathrm{N} / 2$ cell accesses to find a value with Linear Search. Since an **ID** field is also sorted, the Binary Search can be applied, reducing the amount of accesses to $\log_2 \mathrm{N}$.

First Name field is not sorted, so Binary Search is impossible, so $\mathrm{N} / 2$ operations will be required.
First Name is also not unique, so search will need to access *all* cells in order to find a value.

Indexes: what's the point?

Suppose we have a database table:

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Search on **ID** field that contains unique values will require on average $N / 2$ cell accesses to find a value with Linear Search. Since an **ID** field is also sorted, the Binary Search can be applied, reducing the amount of accesses to $\log_2 N$.

First Name field is not sorted, so Binary Search is impossible, so $N / 2$ operations will be required.
First Name is also not unique, so search will need to access *all* cells in order to find a value.

Database table

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Database index

Database table

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Database index

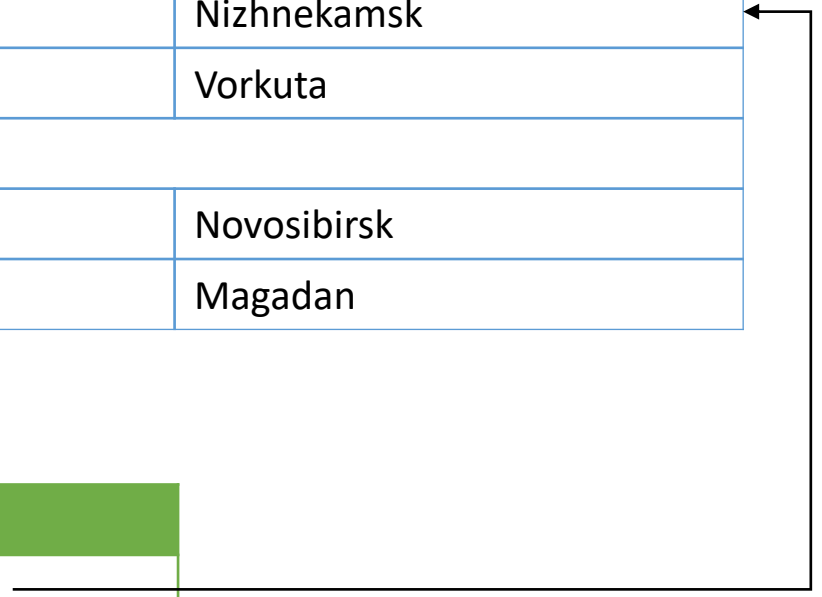
First Name	Record pointer
Vasily	[record 2]
Dmitry	[record 0]
Eugene	[record 9998]
Pavel	[record 1]
Pavel	[record 9999]
…	
Yurii	[record 3]

Database table

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Database index

First Name	Record pointer
Vasily	[record 2]
Dmitry	[record 0]
Eugene	[record 9998]
Pavel	[record 1]
Pavel	[record 9999]
…	
Yurii	[record 3]



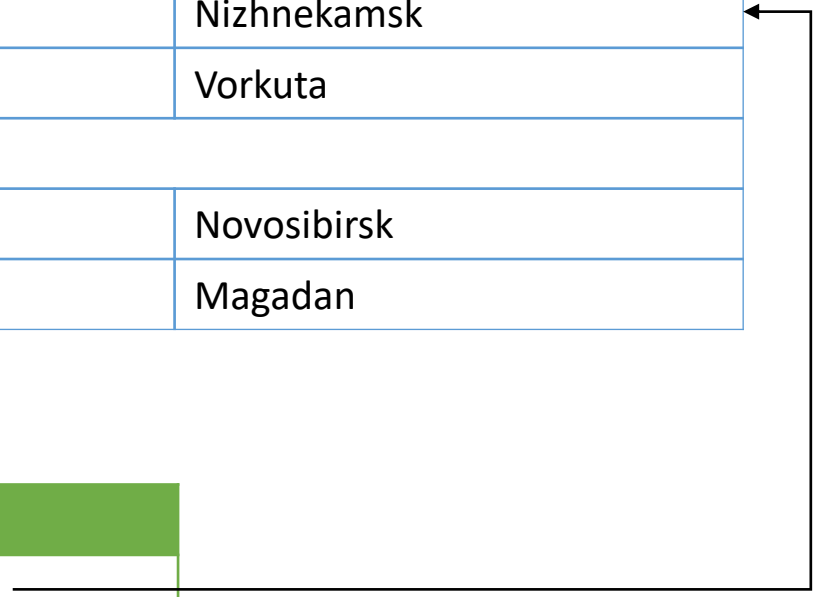
ID	Address
0	ow
1	
2	kamsk
3	a
9998	birsk
9999	an

Database table

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Database index

First Name	Record pointer
Vasily	[record 2]
Dmitry	[record 0]
Eugene	[record 9998]
Pavel	[record 1]
Pavel	[record 9999]
…	
Yurii	[record 3]



Database table

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

Database index

First Name	Record pointer
Vasily	[record 2]
Dmitry	[record 0]
Eugene	[record 9998]
Pavel	[record 1]
Pavel	[record 9999]
…	
Yurii	[record 3]

Database table

ID	First Name	Last Name	Address
0	Dmitry	Aleksandrovich	Moscow
1	Pavel	Ivanovich	Kazan
2	Vasily	Medvedovich	Nizhnekamsk
3	Yurii	Arkadievich	Vorkuta
…			
9998	Eugene	Volfovich	Novosibirsk
9999	Pavel	Dmitriev	Magadan

CLUSTERED

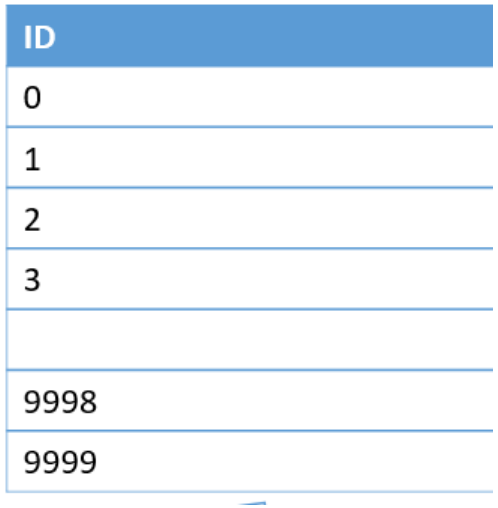
Database index

First Name	Record pointer
Vasily	[record 2]
Dmitry	[record 0]
Eugene	[record 9998]
Pavel	[record 1]
Pavel	[record 9999]
…	
Yurii	[record 3]

NON-CLUSTERED

Index Types

Index Types



ID
0
1
2
3
9998
9999

Clustered

Data points are actually physically sorted.

- Exists in table itself, only one clustered index per table
- Fewer I/O operations when reading the table
- More I/O operations when inserting data in the middle of a table

Index Types

ID
0
1
2
3
9998
9999

Clustered

Data points are actually physically sorted.

- Exists in table itself, only one clustered index per table
- Fewer I/O operations when reading the table
- More I/O operations when inserting data in the middle of a table

First Name	Record pointer
Vasily	[record 2]
Dmitry	[record 0]
Eugene	[record 9998]
Pavel	[record 1]
Pavel	[record 9999]
...	
Yurii	[record 3]

Non-Clustered

A separate table that contains pointers to the table's records.

- Exists separately from table, multiple indexes per table
- Usually represented by columns other than Primary Key

Additional Index Types

On top of being clustered or non-clustered, an index can be of following types

Additional Index Types

On top of being clustered or non-clustered, an index can be of following types

First Name	Last Name
Dmitry	Aleksandrovich
Pavel	Ivanovich
Vasily	Medvedovich
Yurii	Arkadievich
	...
Eugene	Volfovich
Pavel	Dmitriev

Composite

An index that contains more than one column

- Up to 16 fields
- Shouldn't exceed 900-byte limit

Additional Index Types

On top of being clustered or non-clustered, an index can be of following types

First Name	Last Name
Dmitry	Aleksandrovich
Pavel	Ivanovich
Vasily	Medvedovich
Yurii	Arkadievich
	...
Eugene	Volfovich
Pavel	Dmitriev

E-Mail
somemail@google.com
someothermail@yandex.com
yasya95@rambler.ru
test@test.com
...
randommail@mailbox.net
workmail@work.com

Composite

An index that contains more than one column

- Up to 16 fields
- Shouldn't exceed 900-byte limit

Unique

An index that ensures uniqueness of each value in the indexed columns

- Automatically created with the Primary Key
- If combined with Composite, then uniqueness is enforced across all columns

Additional Index Types

On top of being clustered or non-clustered, an index can be of following types

First Name	Last Name
Dmitry	Aleksandrovich
Pavel	Ivanovich
Vasily	Medvedovich
Yurii	Arkadievich
...	
Eugene	Volfovich
Pavel	Dmitriev

E-Mail
somemail@google.com
someothermail@yandex.com
yasya95@rambler.ru
test@test.com
...
randommail@mailbox.net
workmail@work.com

Composite

An index that contains more than one column

- Up to 16 fields
- Shouldn't exceed 900-byte limit

Unique

An index that ensures uniqueness of each value in the indexed columns

- Automatically created with the Primary Key
- If combined with Composite, then uniqueness is enforced across all columns

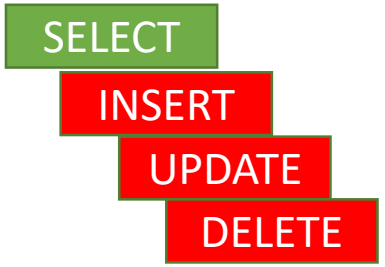
Covering (property of an index)

An index that contains all the data required to process certain query

- Contains all required fields in itself
- Allows a query to be processed without accessing the table

The cost of an Index

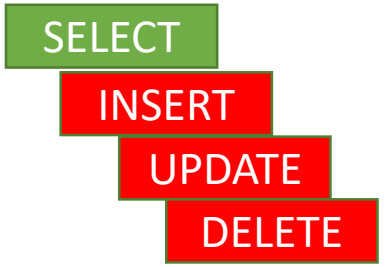
The cost of an Index



Slows down the database when indexed data is altered

- Non-clustered index has to be adjusted appropriately
- Clustered index forces the table itself to adjust

The cost of an Index



Slows down the database when indexed data is altered

- Non-clustered index has to be adjusted appropriately
- Clustered index forces the table itself to adjust

Non-clustered indexes require additional disk space

- As non-clustered index is a table, additional space has to be allocated for it
- Depending on amount of indexing data, can consume significant amount of disk space

The Good and The Bad Index

The Good Index

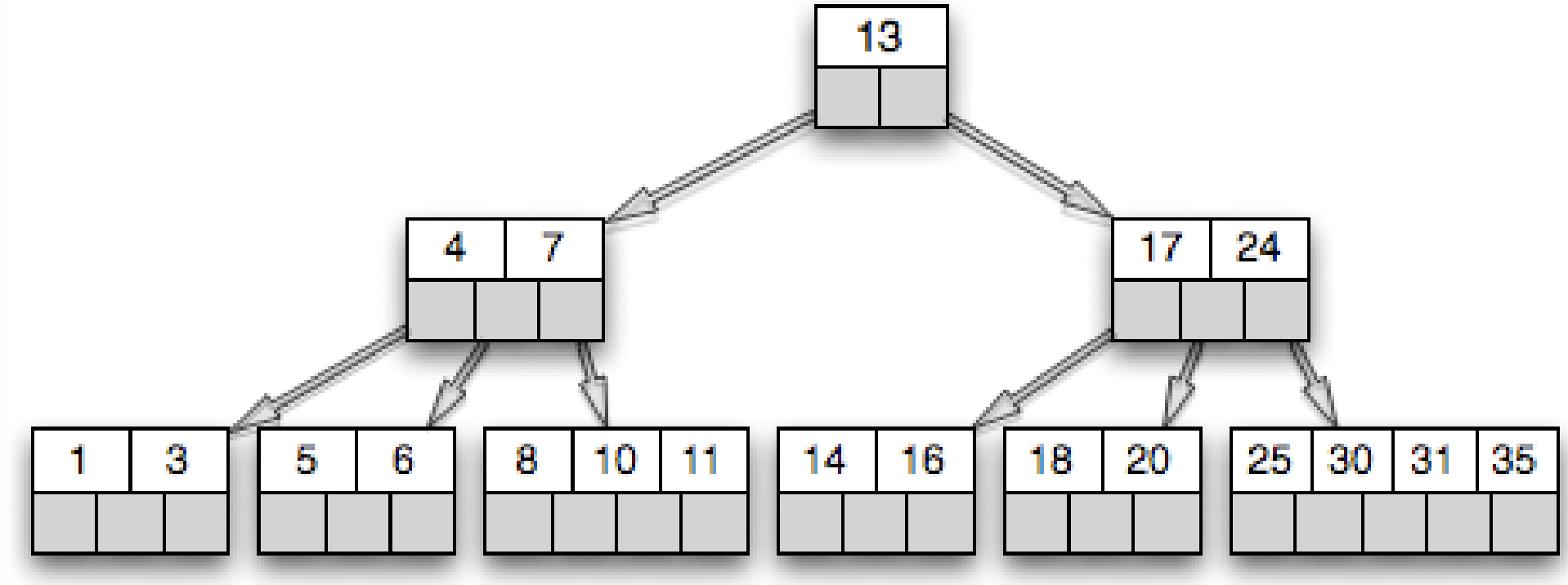
- Rarely-changed columns
- Based on primary or foreign keys
- Indexed columns allow quick identification of data
- Helps to find a range of information
- Keeps data ordered (clustered index)
- (depends on a situation) Covering index

The Bad Index

- Unused / often modified columns
- Columns that contain many duplicates
- Contains too many columns
- Created on small tables

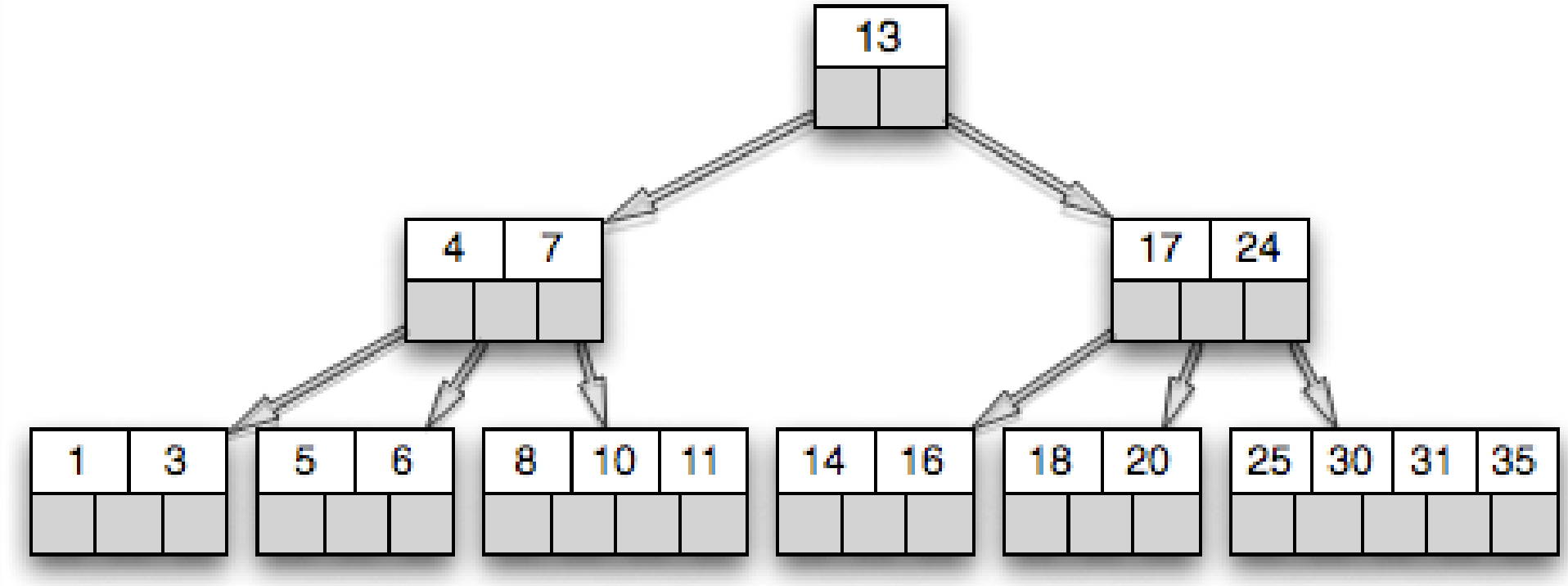
The Ugly Index

Index is actually a B-Tree (as in “balanced”, not “binary”) under the hood



The Ugly Index

Index is actually a B-Tree (as in “balanced”, not “binary”) under the hood



$\log_b N$, where *b* – amount of elements per page

For a table with 1 million elements - $\log_{100} 1000000 = 3$ disk reads

Index operations

Index operations

Creation

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX index_name
ON table_name ( column_name [ASC | DESC] [,…n] )
[WITH {IGNORE_DUP_KEY | DROP_EXISTING | SORT_IN_TEMPDB}]
[ON filegroup_name]
```

```
CREATE NONCLUSTERED INDEX ClientNames ON dbo.Clients (FirstName DESC, LastName
DESC)
```

Index operations

Creation

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX index_name
ON table_name ( column_name [ASC | DESC] [,…n] )
[WITH {IGNORE_DUP_KEY | DROP_EXISTING | SORT_IN_TEMPDB}]
[ON filegroup_name]
```

```
CREATE NONCLUSTERED INDEX ClientNames ON dbo.Clients (FirstName DESC, LastName
DESC)
```

Deletion

```
DROP INDEX index_name ON table_name
```

```
DROP INDEX ClientNames ON dbo.Clients
```

Index operations

Creation

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX index_name
ON table_name ( column_name [ASC | DESC] [,…n] )
[WITH {IGNORE_DUP_KEY | DROP_EXISTING | SORT_IN_TEMPDB}]
[ON filegroup_name]
```

```
CREATE NONCLUSTERED INDEX ClientNames ON dbo.Clients (FirstName DESC, LastName
DESC)
```

Deletion

```
DROP INDEX index_name ON table_name
```

```
DROP INDEX ClientNames ON dbo.Clients
```

Alteration

A database index cannot be altered with ALTER command. To alter an index, it should be dropped and then created anew.

```
CREATE NONCLUSTERED INDEX ClientNames ON dbo.Clients (FirstName DESC, LastName
DESC) WITH DROP_EXISTING
```

Questions

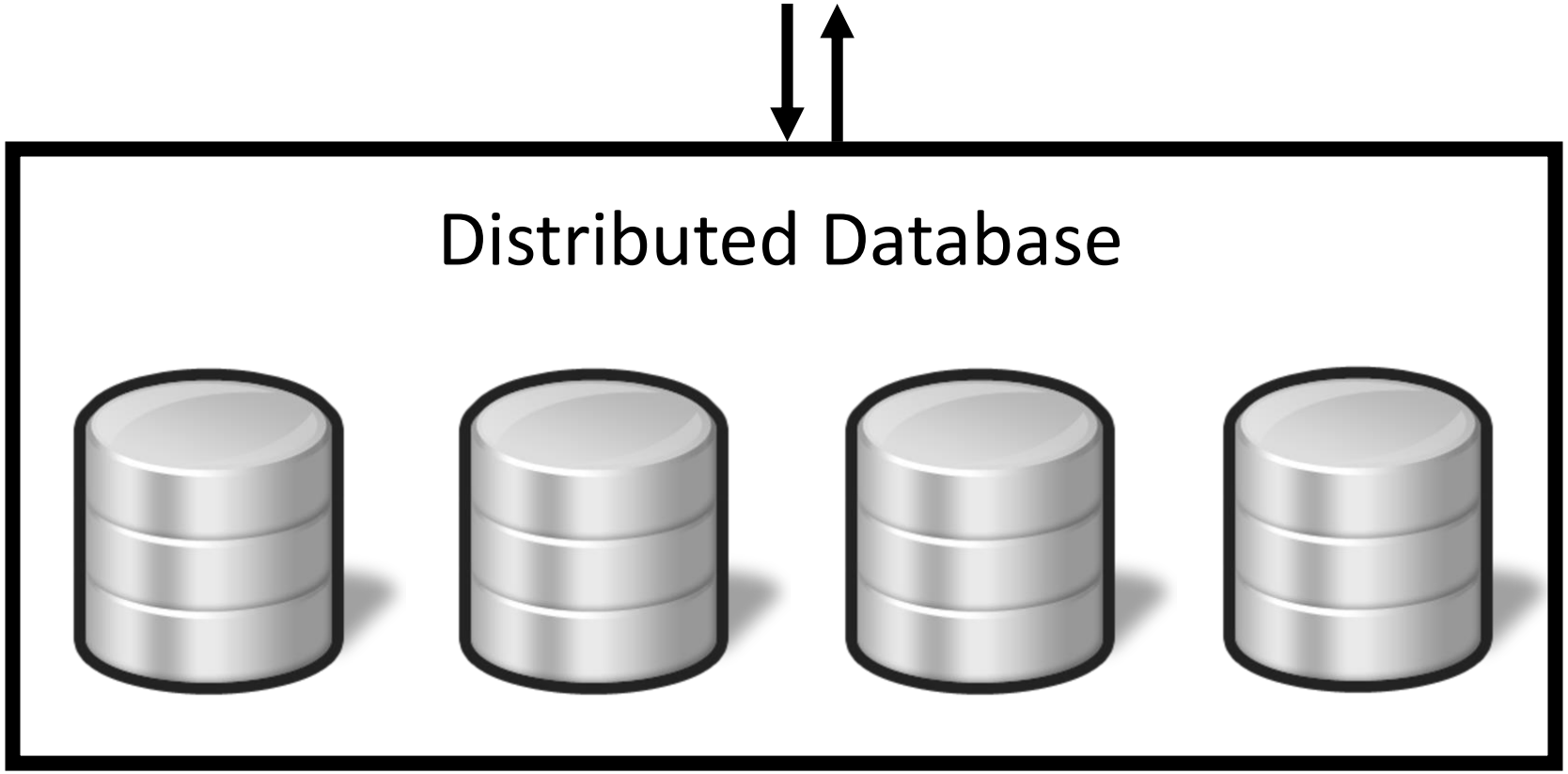
1. What’s the point of an index and what’s the trade-off to achieve it?
2. What makes and index good and what makes it bad?
3. Name two basic index types and explain the difference between them.
4. Describe additional index types.

Database replication

Database Replication

Simply put, replication is keeping the copy of the same data in multiple places.

Not-so-simply-put, replication is a scaling technique. It’s a distribution of database data into multiple databases that altogether make a single distributed database.

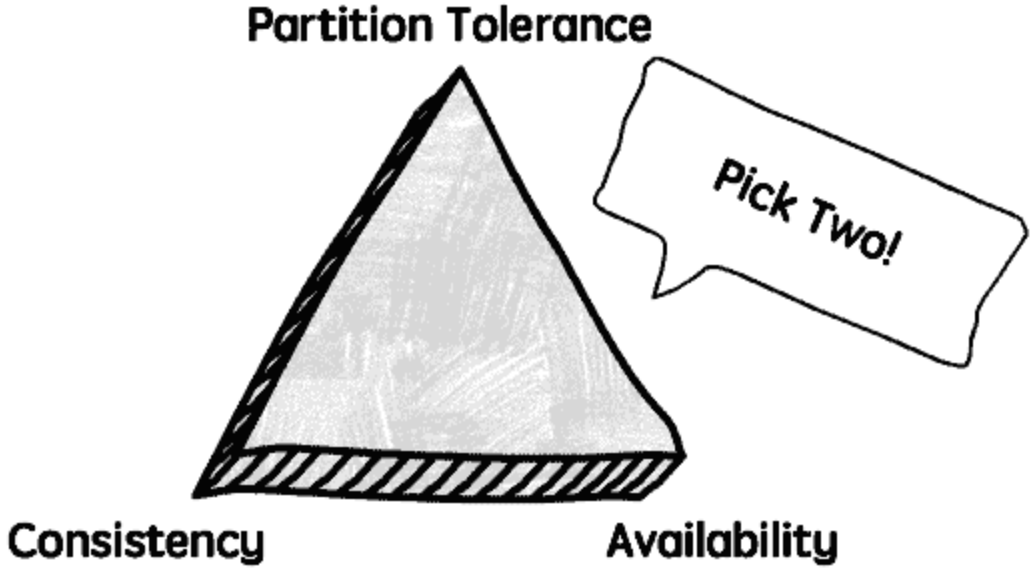


Database Replication: Reasons

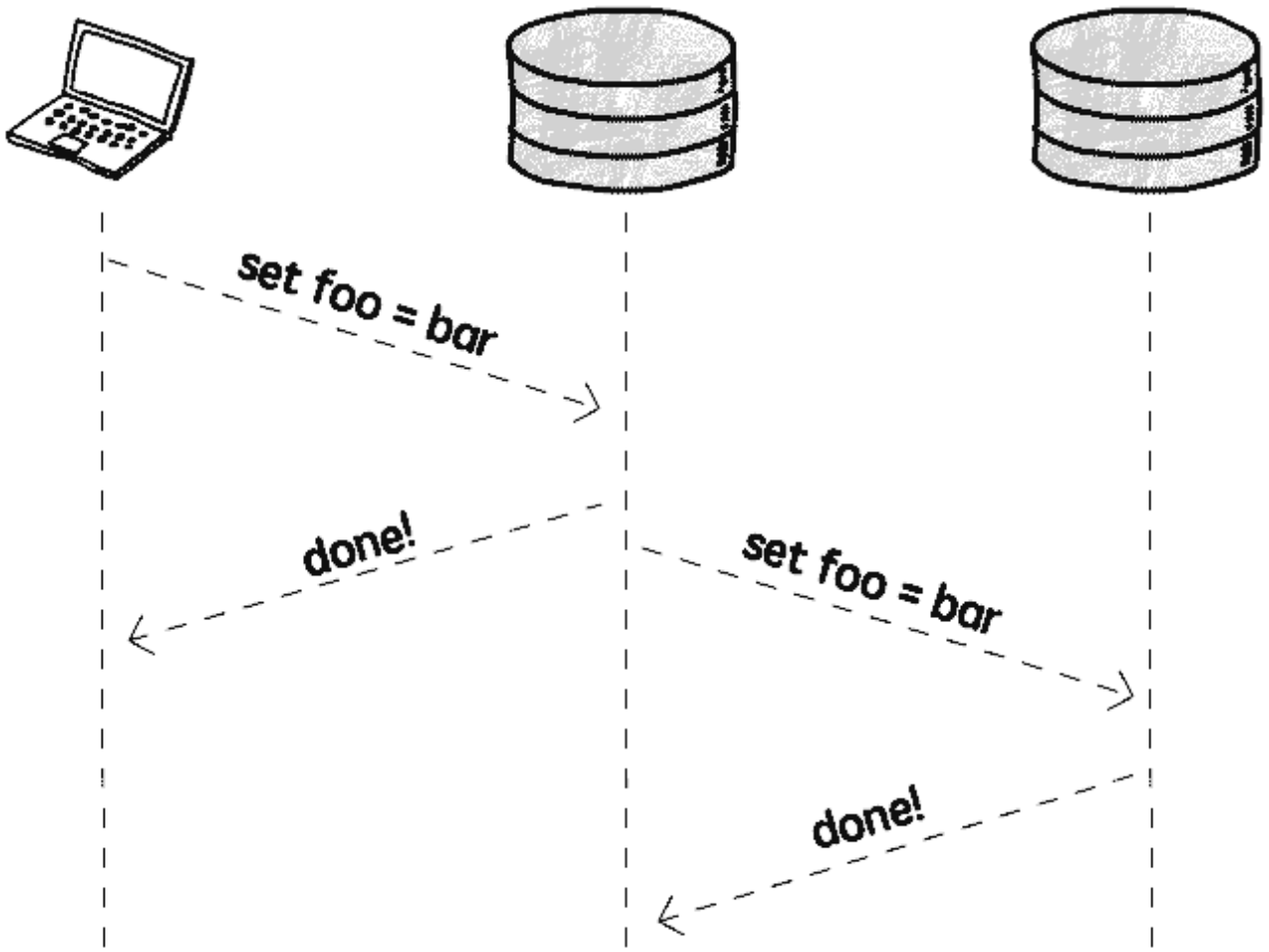
- **Speed:** keeping data close to the users
- **Performance**: having multiple machines handling requests
- **Stability:** switching to the second database in case of masters' failure

Database Replication: Reasons

- **Speed:** keeping data close to the users
- **Performance**: having multiple machines handling requests
- **Stability:** switching to the second database in case of masters' failure



Asynchronous Replication



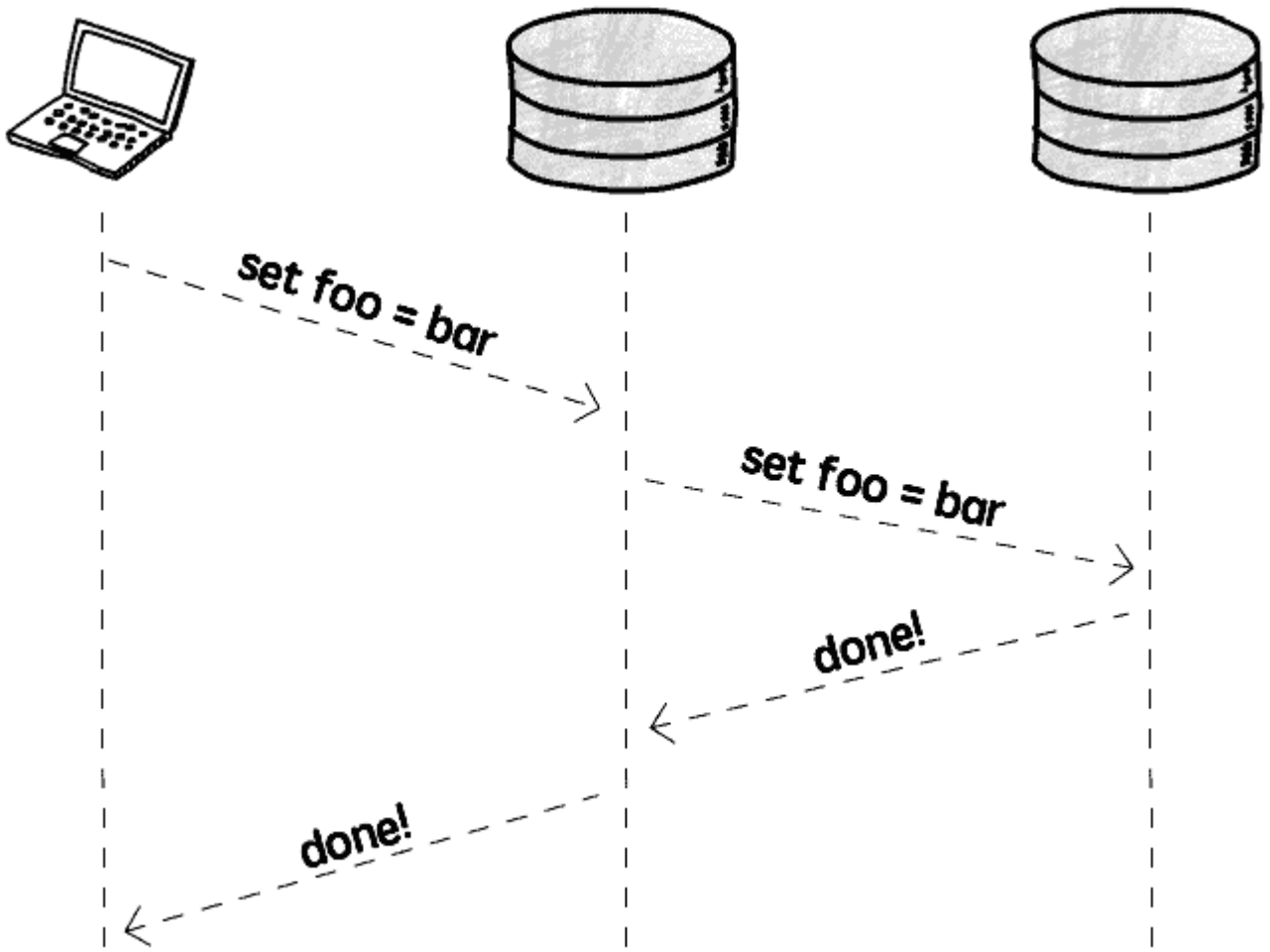
PROs

- No client performance impact
- Replication happens in background

CONs

- Lower durability
- Replication lag

Synchronous Replication



PROs

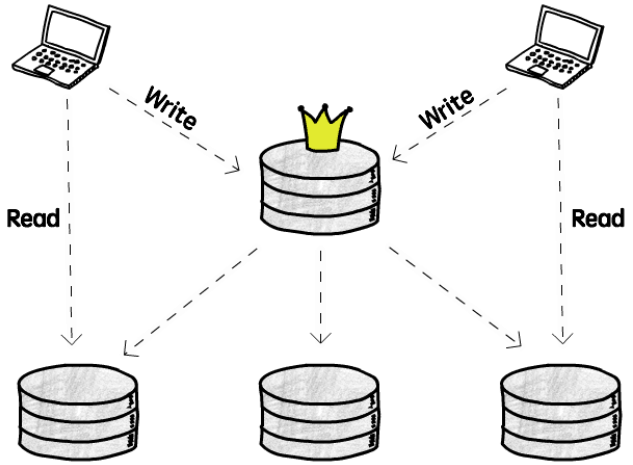
- Higher durability

CONs

- Lower performance
- Lower availability

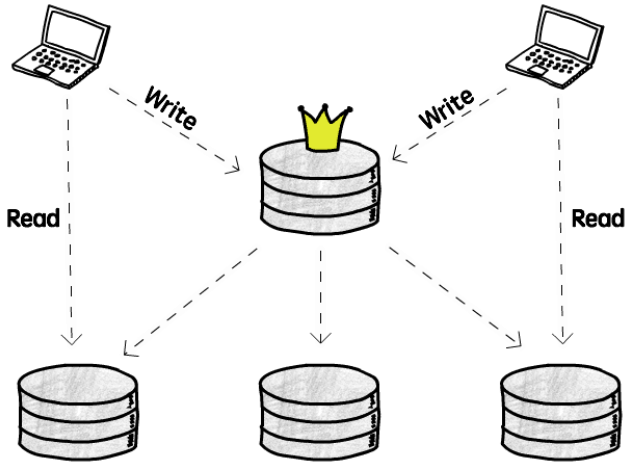
Replication Topologies

Single Leader

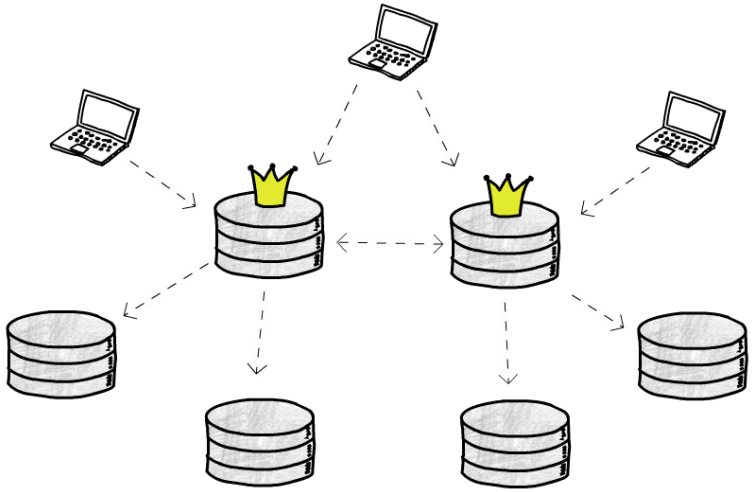


Replication Topologies

Single Leader

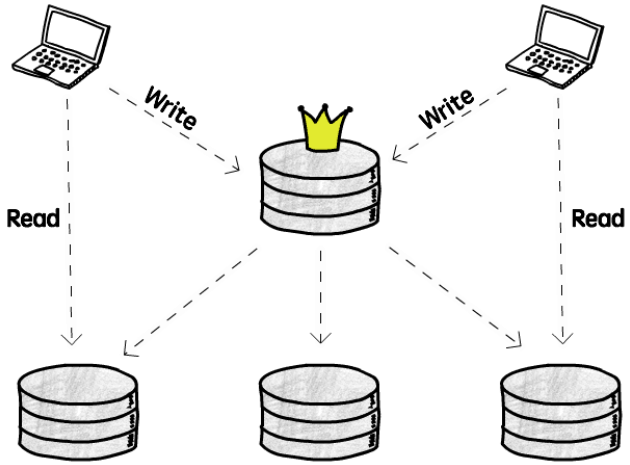


Multi-Leader

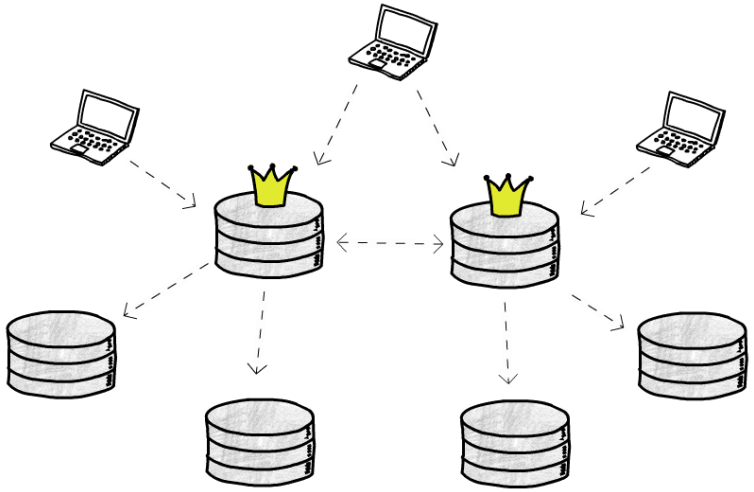


Replication Topologies

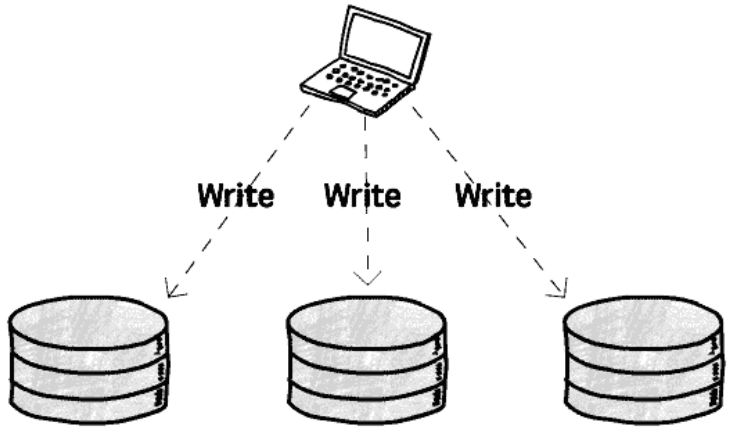
Single Leader



Multi-Leader



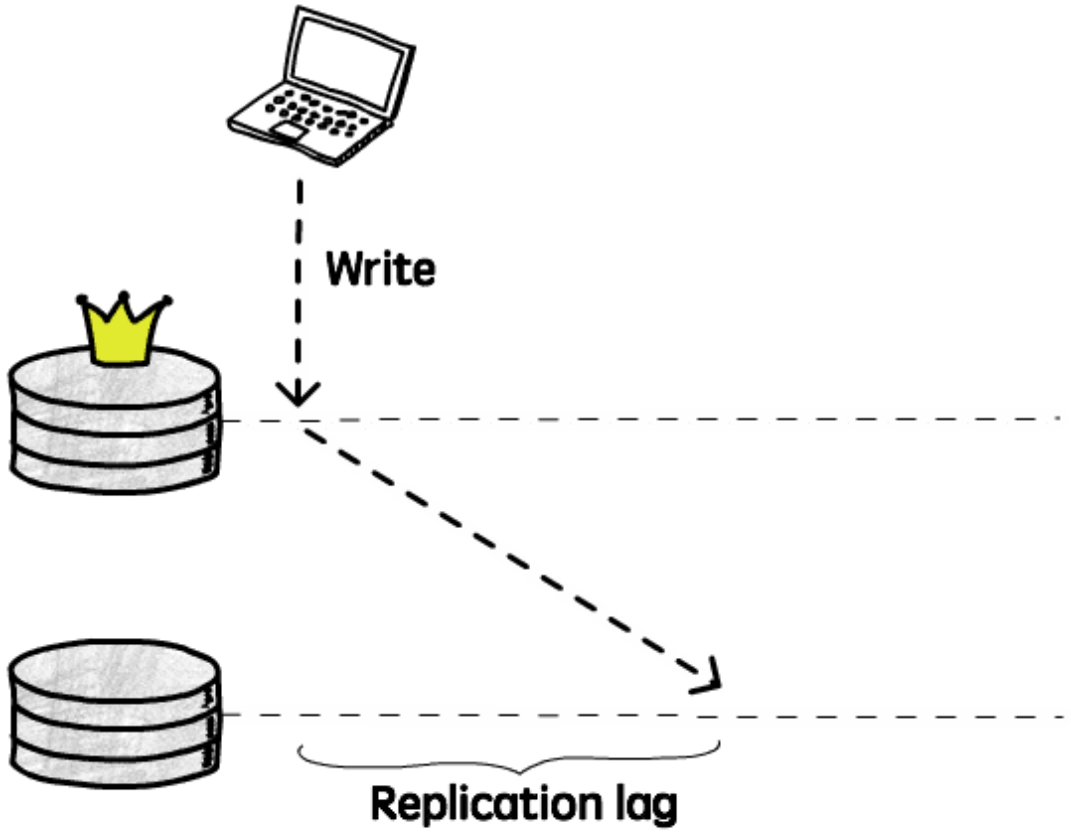
Leaderless



Replication Lag

It’s a delay between data being written into the Leader and being replicated to the Follower.

- Not always a serious problem
- Causes
  1. Read-your-writes consistency failure
  2. Monotonic reads consistency failure
- Used to create “Delayed replicas”



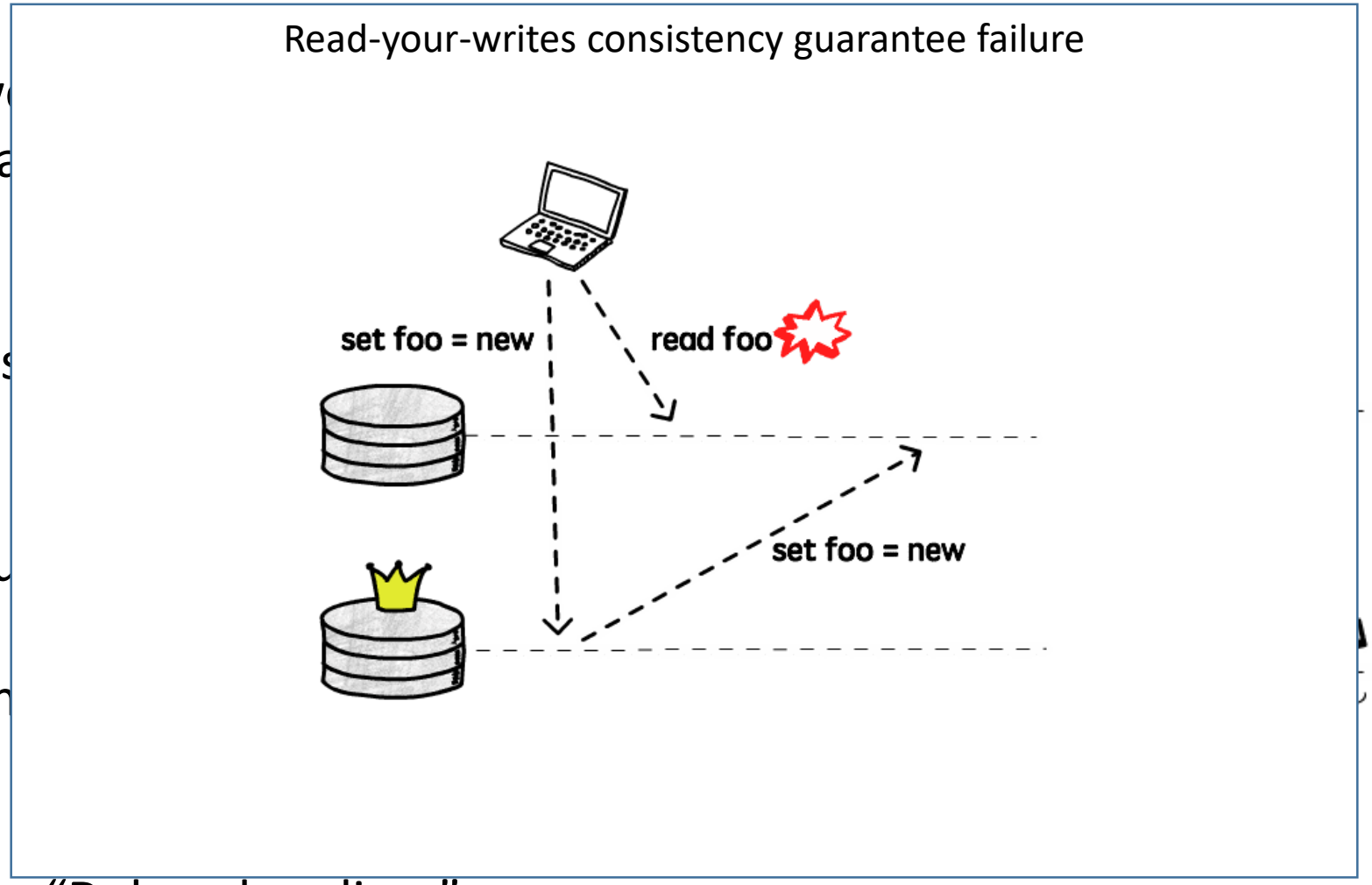
Replication Lag

It's a delay betw
into the Leader a
the Follower.

- Not always a s

- Causes
  1. Read-you
     failure
  2. Monoton
     failure

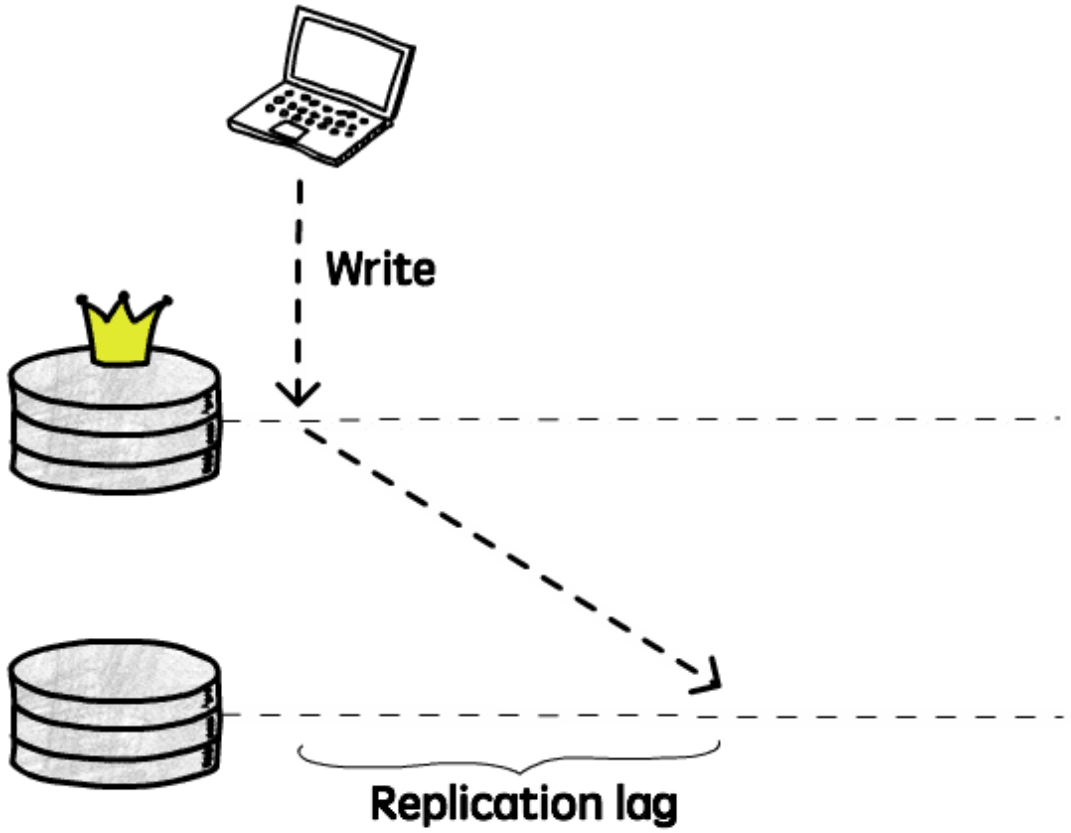
- Used to create "Delayed replicas"



Replication Lag

It’s a delay between data being written into the Leader and being replicated to the Follower.

- Not always a serious problem
- Causes
  1. Read-your-writes consistency failure
  2. Monotonic reads consistency failure
- Used to create “Delayed replicas”



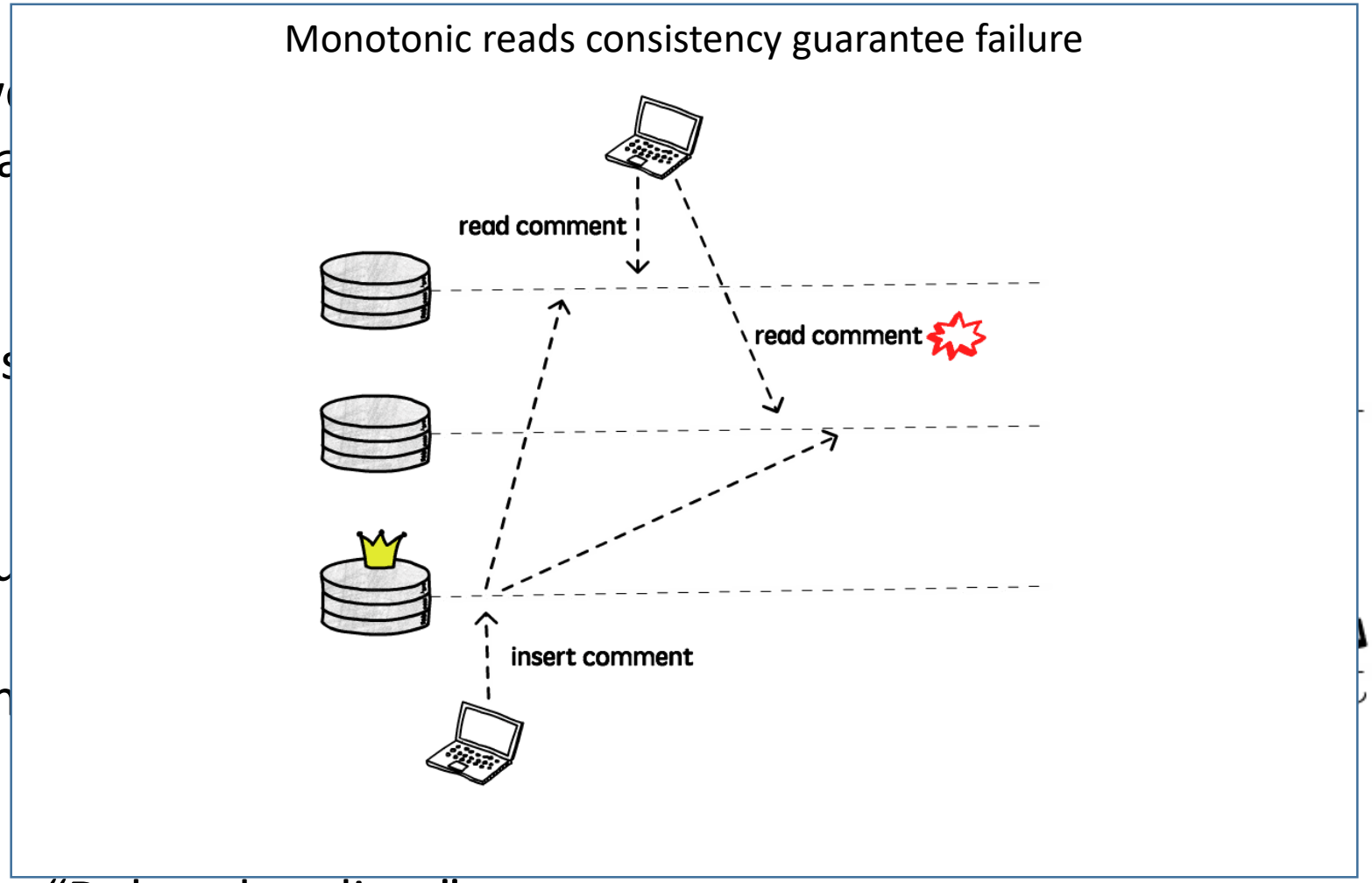
Replication Lag

It’s a delay betw
into the Leader a
the Follower.

- Not always a s

- Causes
  1. Read-you
     failure
  2. Monoton
     failure

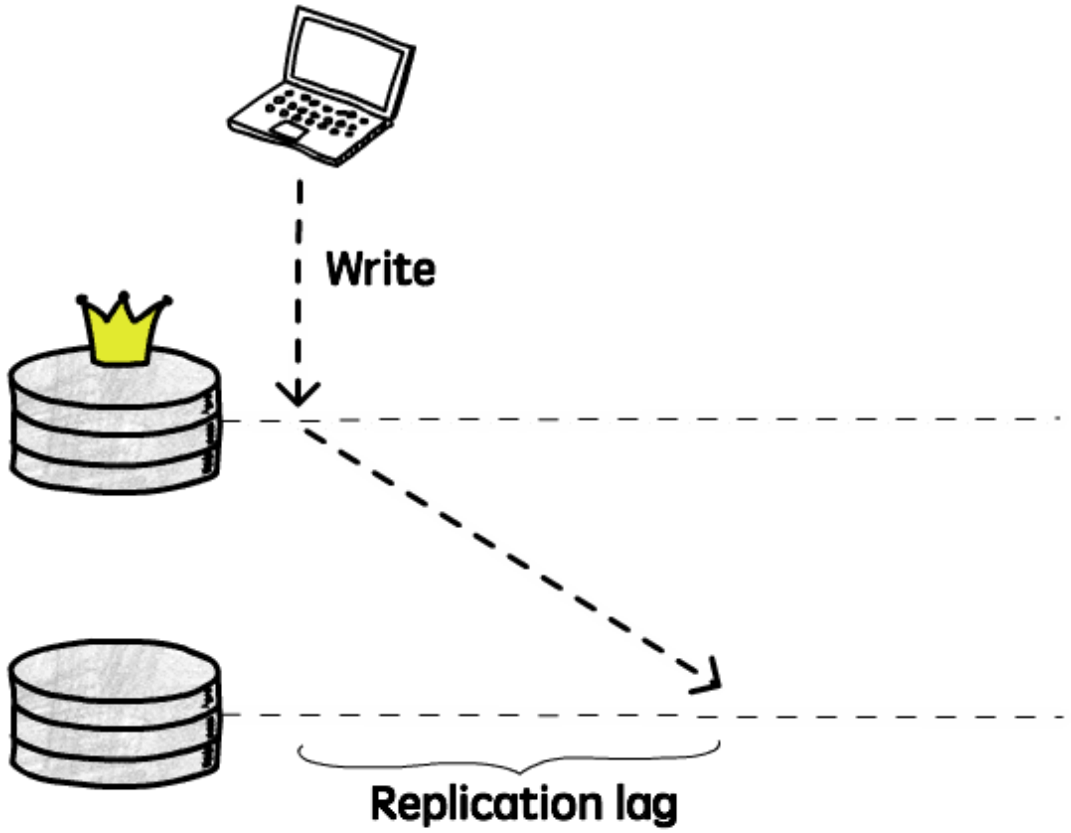
- Used to create “Delayed replicas”



Replication Lag

It’s a delay between data being written into the Leader and being replicated to the Follower.

- Not always a serious problem
- Causes
  1. Read-your-writes consistency failure
  2. Monotonic reads consistency failure
- Used to create “Delayed replicas”



Database Replication: Types

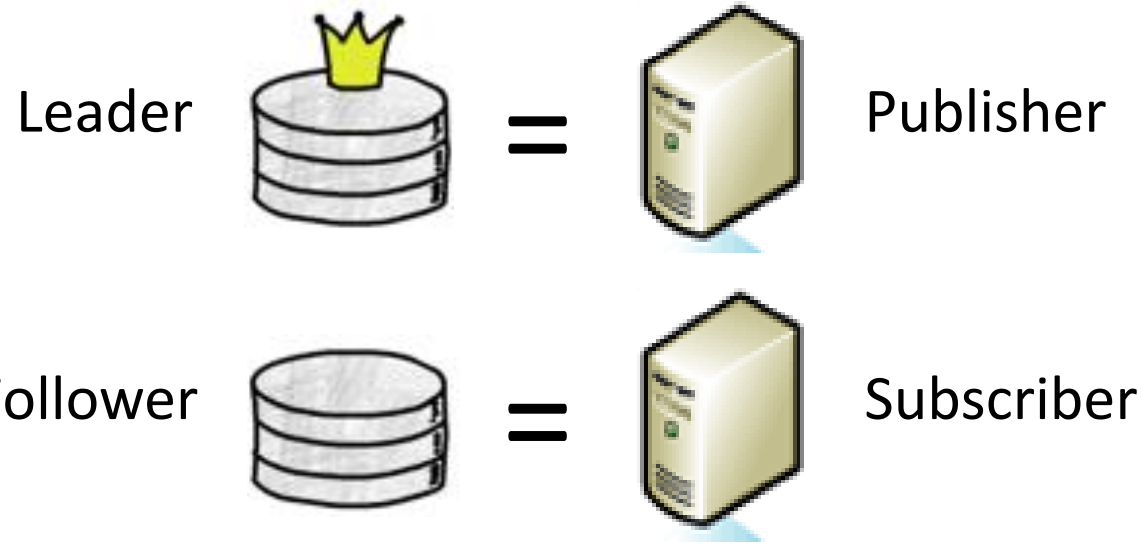
Database Replication: Types

- Snapshot replication
- Transactional replication (Statement-based replication)
- Merge Replication (Row-based replication)

Database Replication: Types

- Snapshot replication
- Transactional replication (Statement-based replication)
- Merge Replication (Row-based replication)

English-Microsoft Dictionary



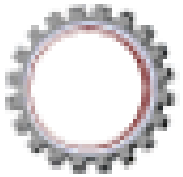
Database Replication: Types

- Snapshot replication
- Transactional replication (Statement-based replication)
- Merge Replication (Row-based replication)

English-Microsoft Dictionary

Leader  =  Publisher

Follower  =  Subscriber



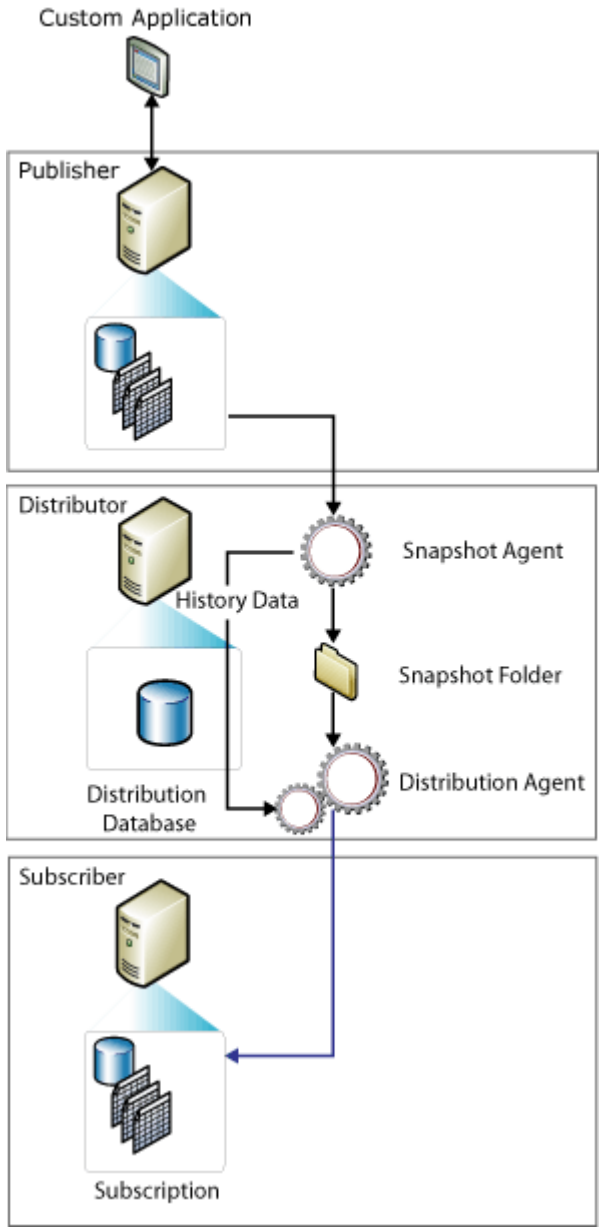
Agent
An application that handles replication operations, a simple executable file, nothing to write home about.

Snapshot replication

Distributes the entire dataset to all Follower DBs in a form of a singe “snapshot”. The snapshot if used to initialize the Subscriber / Follower DBs in other forms of replication.

Appropriate to use when:

- Infrequent changes
- Small data volumes
- Large amount of changes occurred over a short period of time
- It’s acceptable to have copies of data that are out of sync with the Publisher for some time

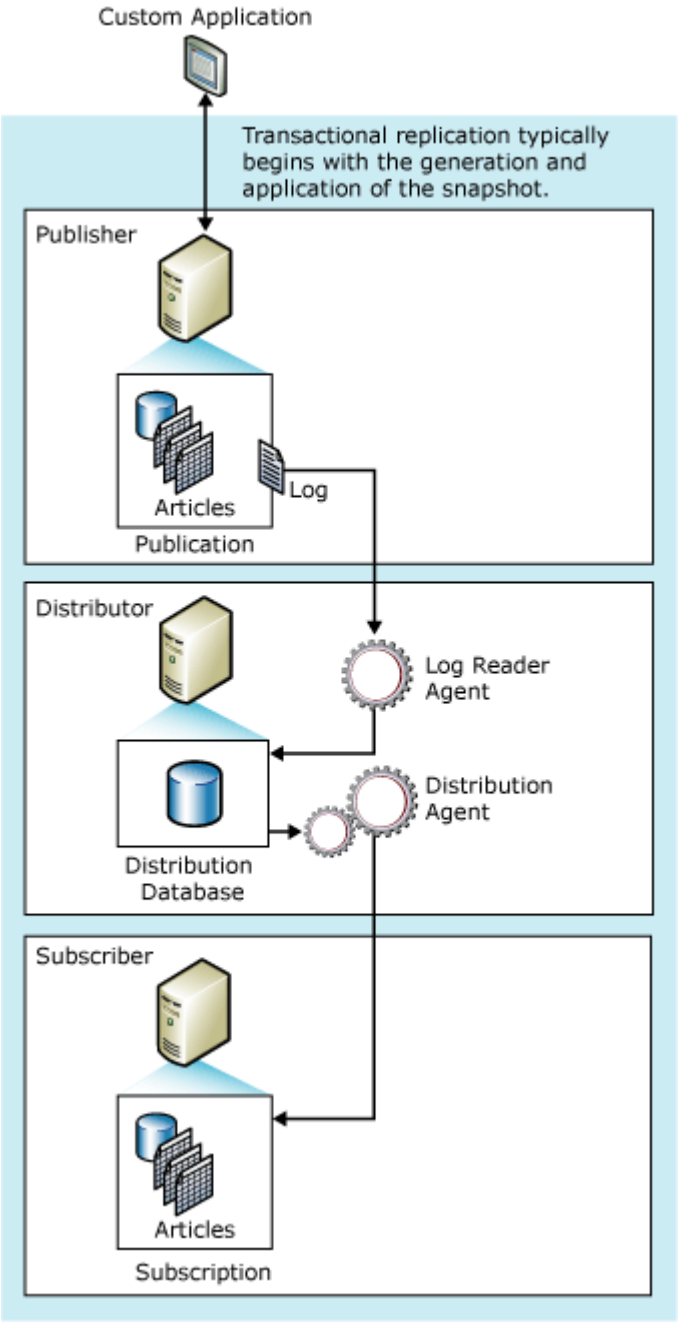


Transactional replication

Generally known as statement-based replication. It applies changes to the Subscriber / Follower in real-time, in the same order and within same transactional boundaries as they occurred at the Publisher / Master.

Appropriate to use when:

- Incremental changes should be immediately propagated
- Application requires low latency of change propagation.
- Application requires access to intermediate data states.
- Publisher has high volume of INSERT, UPDATE and DELETE activity
- Either Publisher or Subscriber is not an MS SQL Server DB

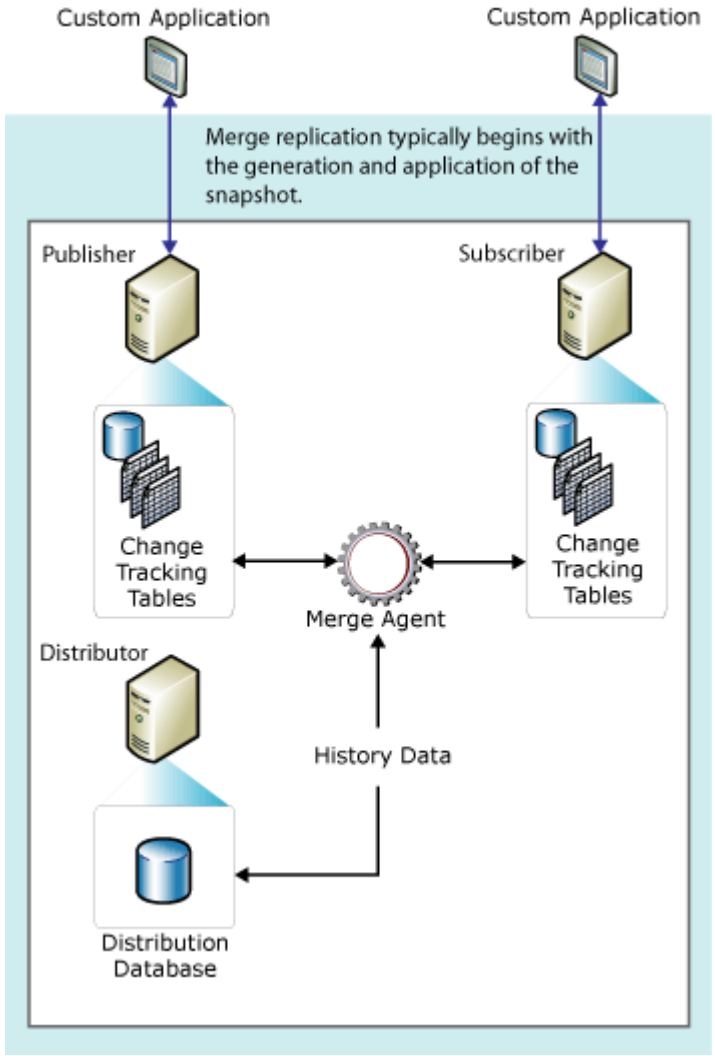


Merge replication

Generally known as row-based replication. Synchronizes rows that were changed on both sides since the last synchronization between Publisher / Master and Subscriber / Follower.

Appropriate to use when:

- Multiple Subscribers might update same data at different time
- Subscribers require offline work with following synchronization
- Each Subscriber requires a different partition of data
- Conflicts might occur and we need an ability to resolve them
- The application requires only final data change instead of intermediate changes



Questions

1. What's the point of replication?
2. What types of replication exist?
3. What replication topologies exist?
4. What's the difference between asynchronous and synchronous replications?

Table partitioning and sharding

Partitioning

Partitioning is breaking up a single database table into multiple entities for the sake of performance and maintainability. Partitioning usually implies **vertical partitioning.**

Users table

ID	First Name	Last Name
0	Dmitry	Aleksandrovich
1	Pavel	Ivanovich
2	Vasily	Medvedovich
3	Yurii	Arkadievich
	...	
9998	Eugene	Volfovich
9999	Pavel	Dmitriev

Partitioning

Partitioning is breaking up a single database table into multiple entities for the sake of performance and maintainability. Partitioning usually implies **vertical partitioning.**

Users table

ID	First Name	Last Name	Addr
0	Dmitry	Aleksandrovich	Mos
1	Pavel	Ivanovich	Kaza
2	Vasily	Medvedovich	Nizh
3	Yurii	Arkadievich	Vork
...			
9998	Eugene	Volfovich	Novc
9999	Pavel	Dmitriev	Mag

Addresses table

ID	
0	
1	
2	
3	
9998	
9999	

Sharding

Sharding is **horizontal partitioning**. When table is sharded, a new table with the same schema is created and populated with original table's data based on a shard key.

Users table

ID	First Name	Last Name
0	Dmitry	Aleksandrovich
1	Pavel	Ivanovich

Sharding

Sharding is **horizontal partitioning**. When table is sharded, a new table with the same schema is created and populated with original table's data based on a shard key.

Users0_1 table

ID	First Name	Last Name
0	Dmitry	Aleksandrovich
1	Pavel	Ivanovich

Users2_3 table

ID	First Name	Last Name

Sharding

Sharding is **horizontal partitioning**. When table is sharded, a new table with the same schema is created and populated with original table's data based on a shard key.

Users0_1 table

ID	First Name	Last Name
0	Dmitry	Aleksandrovich
1	Pavel	Ivanovich
2	Vasily	Medvedovich
3	Yurii	Arkadievich

Users2_3 table

ID	First Name	Last Name

Sharding

Sharding is **horizontal partitioning**. When table is sharded, a new table with the same schema is created and populated with original table's data based on a shard key.

Users0_1 table

ID	First Name	Last Name
0	Dmitry	Aleksandrovich
1	Pavel	Ivanovich
2	Vasily	Medvedovich
3	Yurii	Arkadievich

Users2_3 table

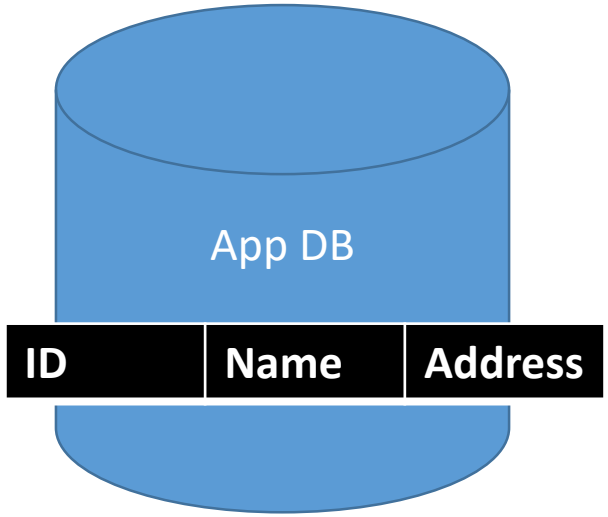
ID	First Name	Last Name

Sharding strategies:

- Lookup
- Range
- Hash

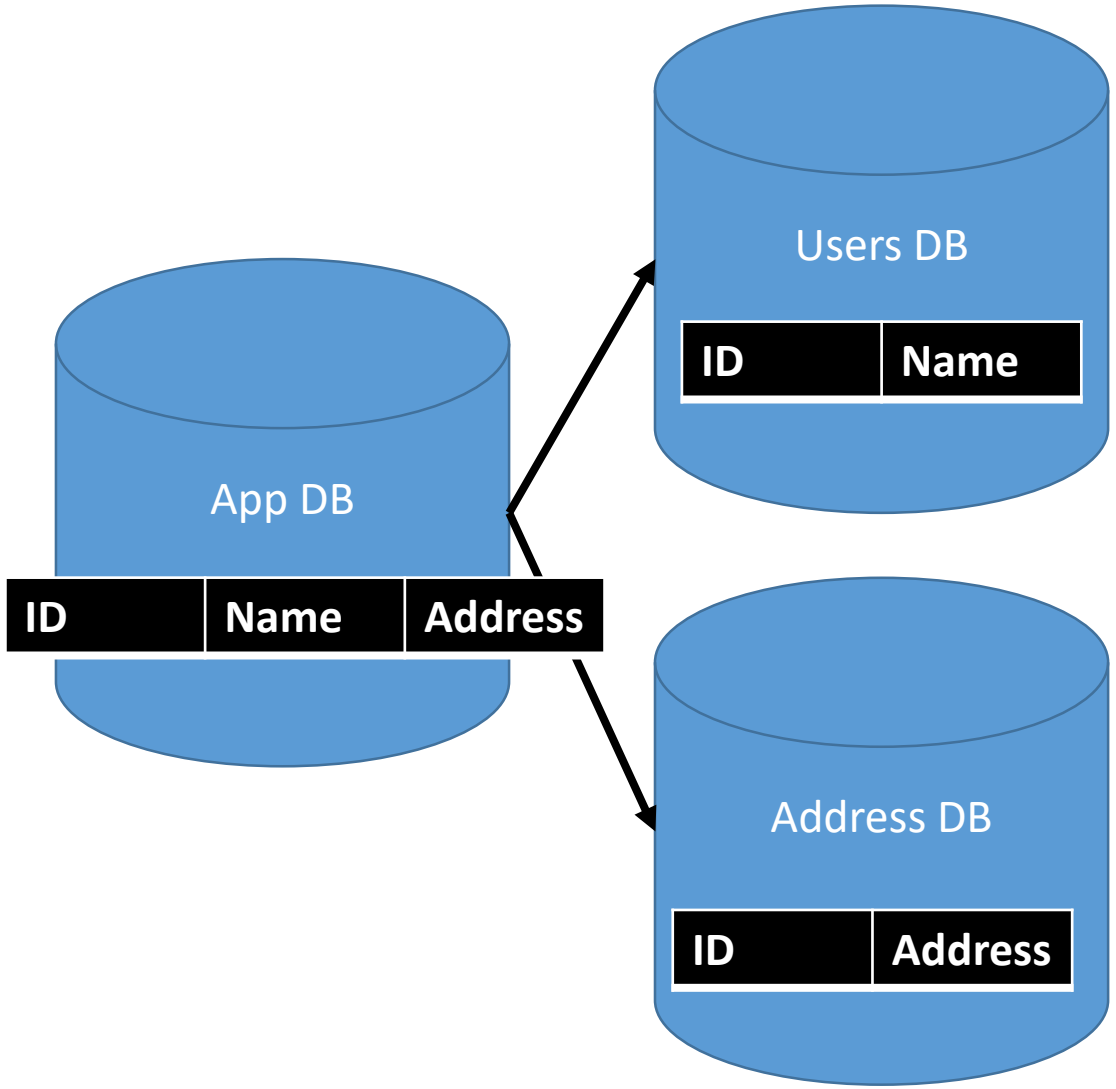
Database partitioning and sharding

Partitioning a sharding can be applied to the entire database and not just tables within.



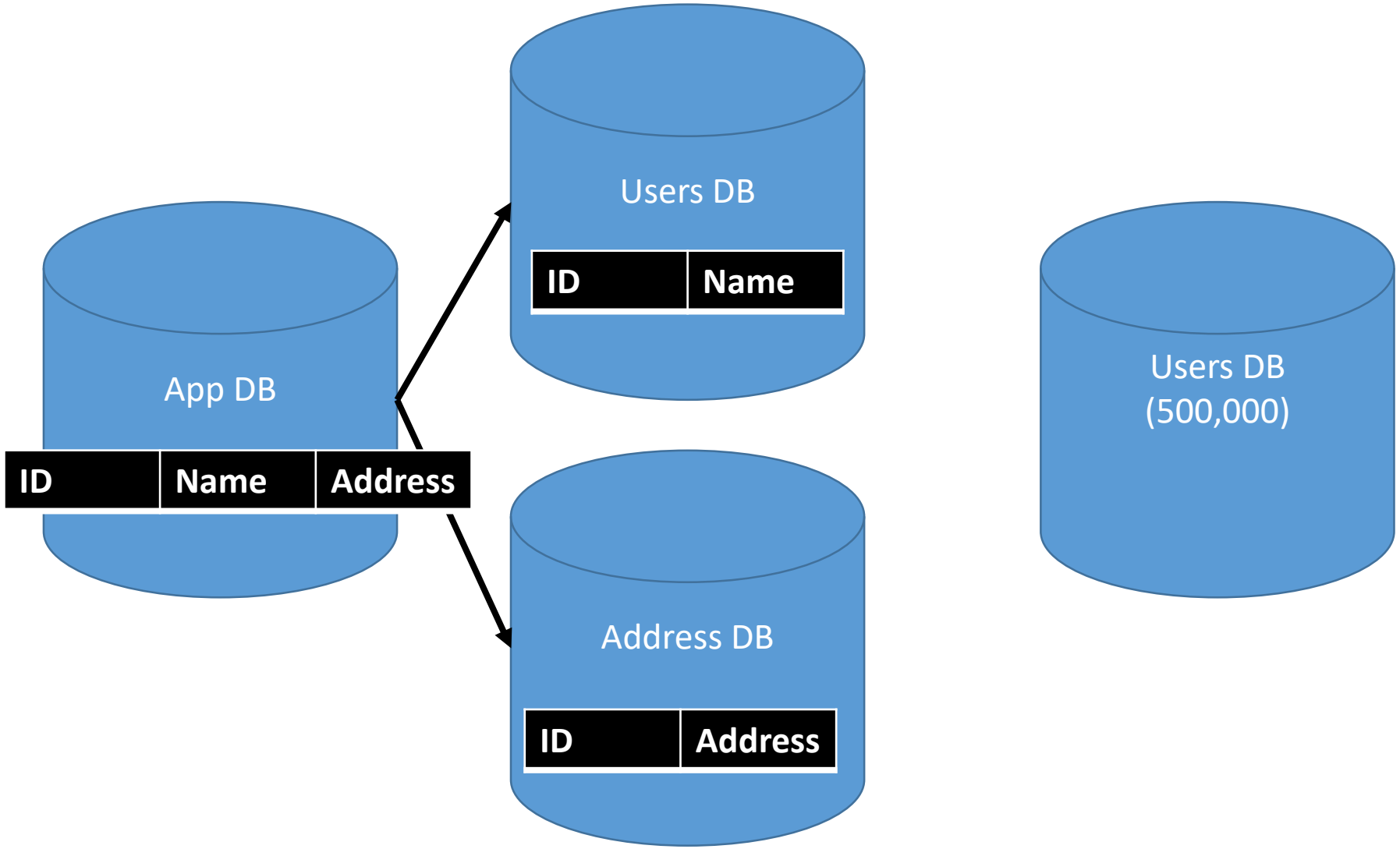
Database partitioning and sharding

Partitioning a sharding can be applied to the entire database and not just tables within.



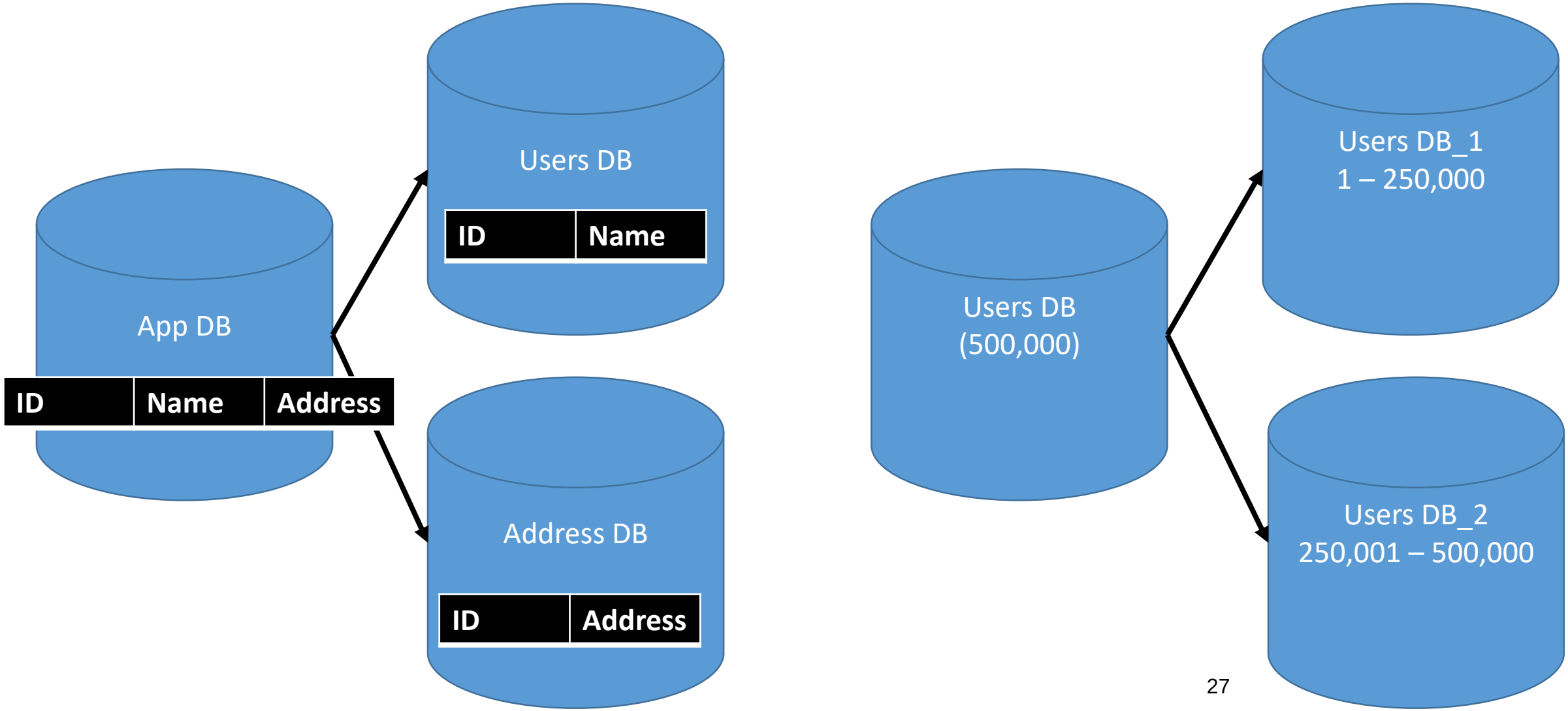
Database partitioning and sharding

Partitioning a sharding can be applied to the entire database and not just tables within.



Database partitioning and sharding

Partitioning a sharding can be applied to the entire database and not just tables within.



Questions

1. What's a partitioning? What's the point of partitioning?
2. What is vertical partitioning?
3. What is horizontal partitioning?
4. What entities can be partitioned?

That’s it.