Querying Data in Ignite

Edward Curren
ENTERPRISE ARCHITECT

@EdwardCurren http://www.edwardcurren.com

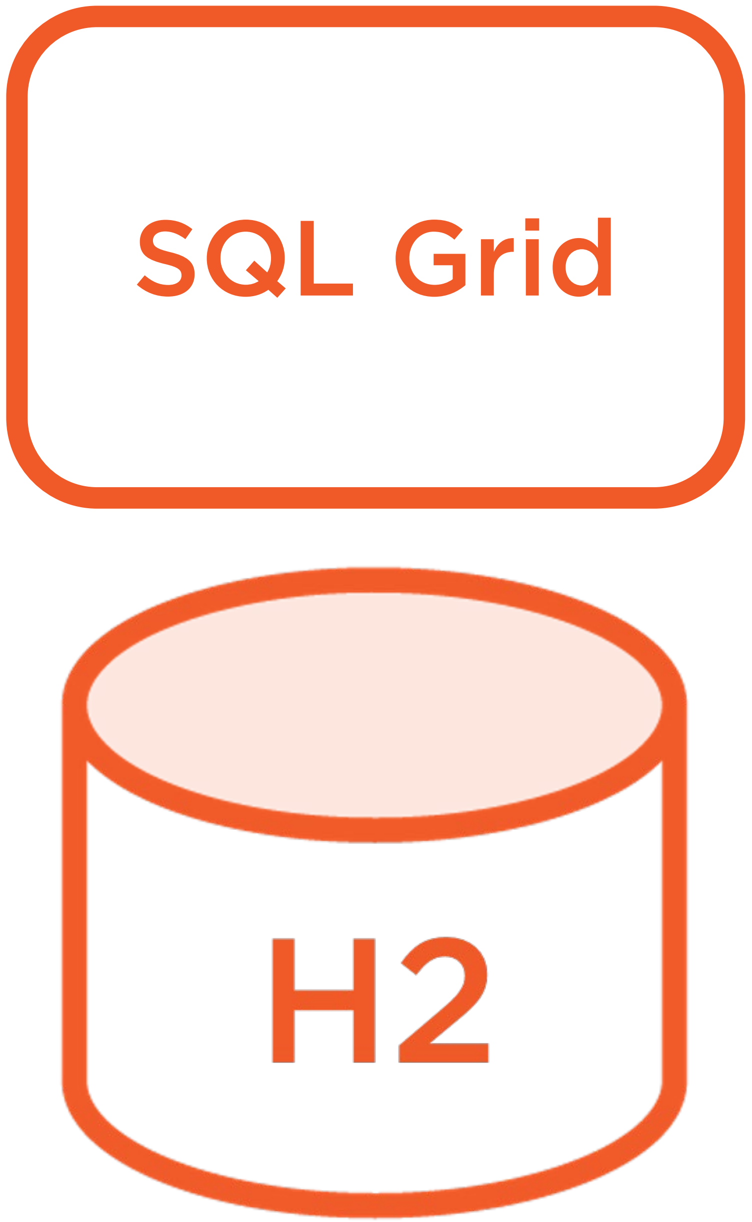
Overview

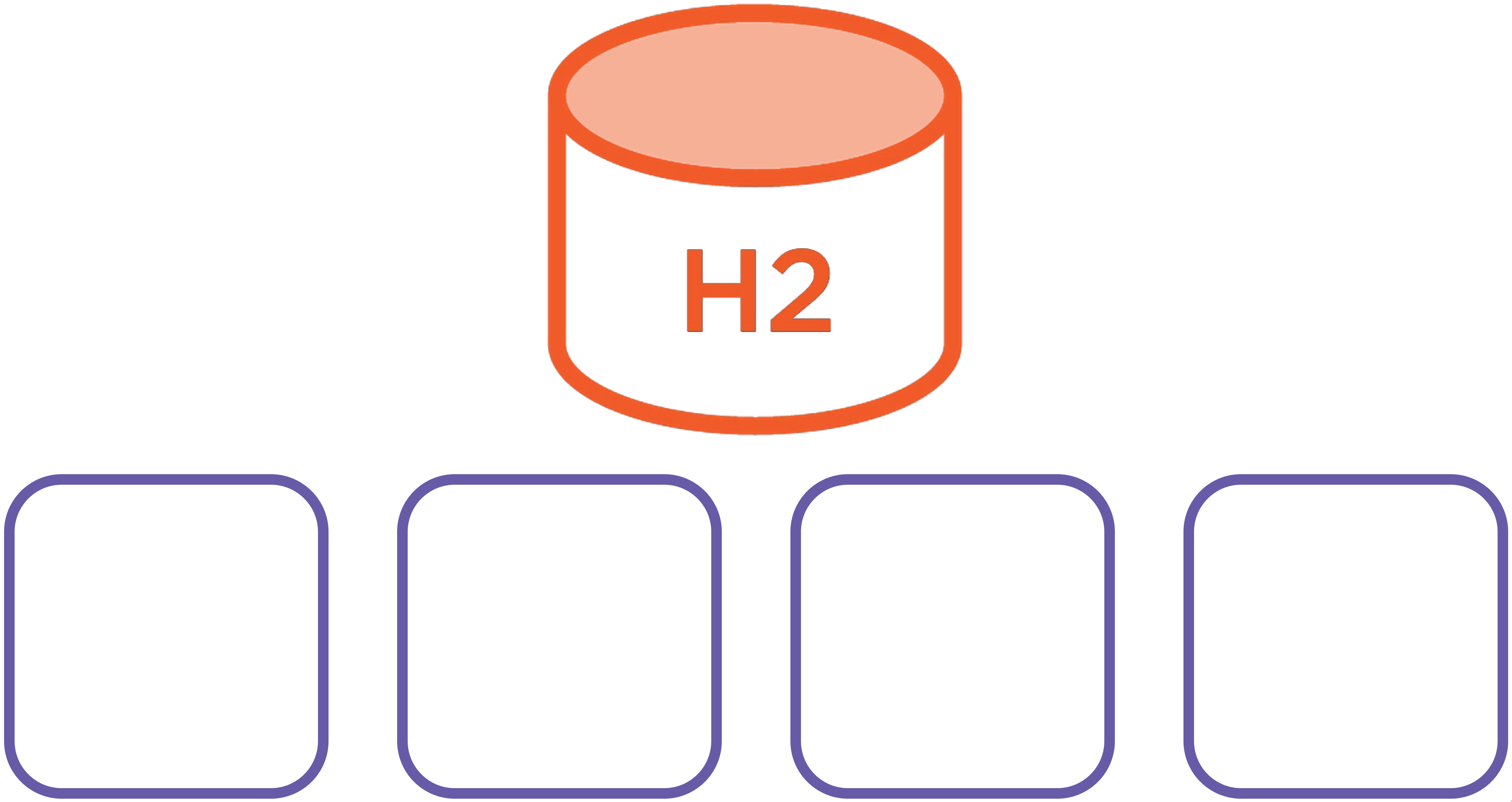
Architecture

Affinity function & affinity collocation

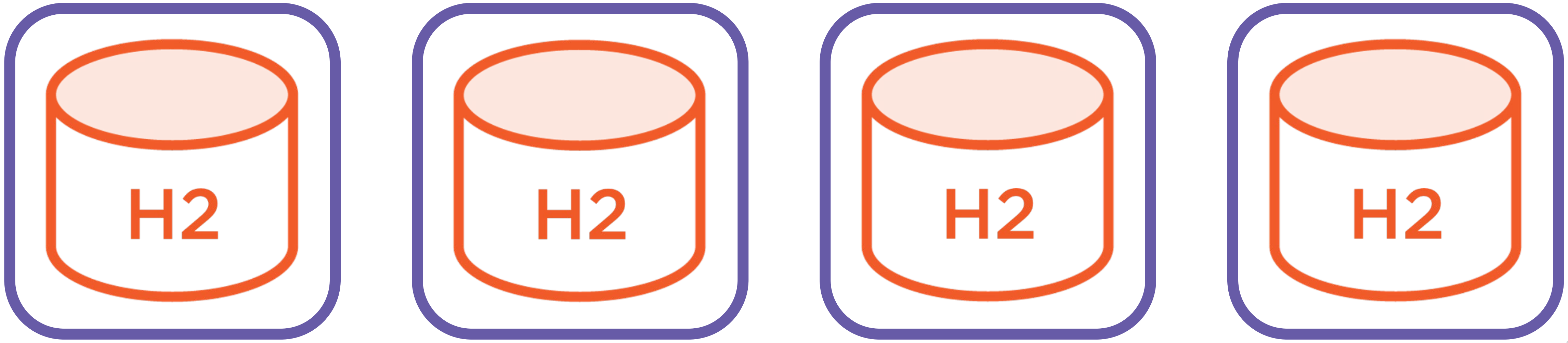
“Fat” keys

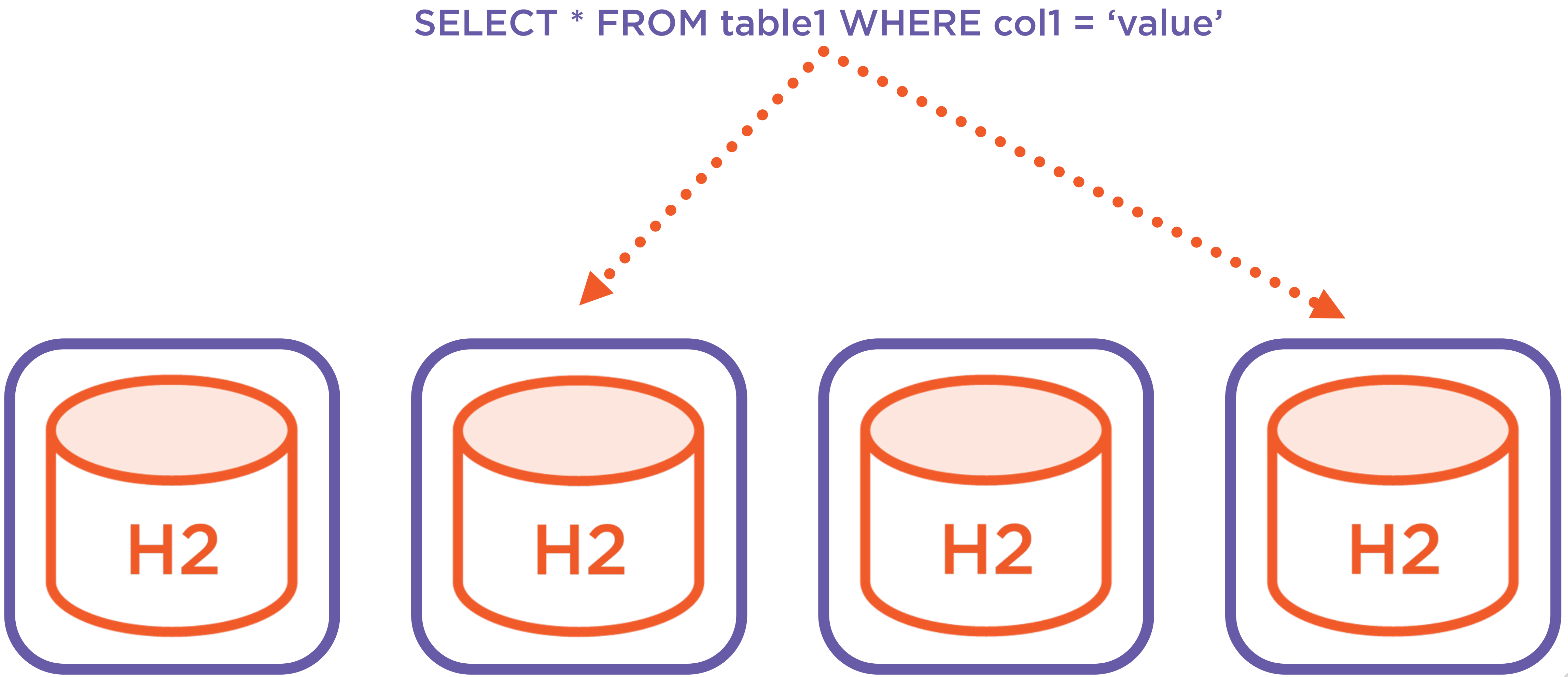
SQL & other types of queries

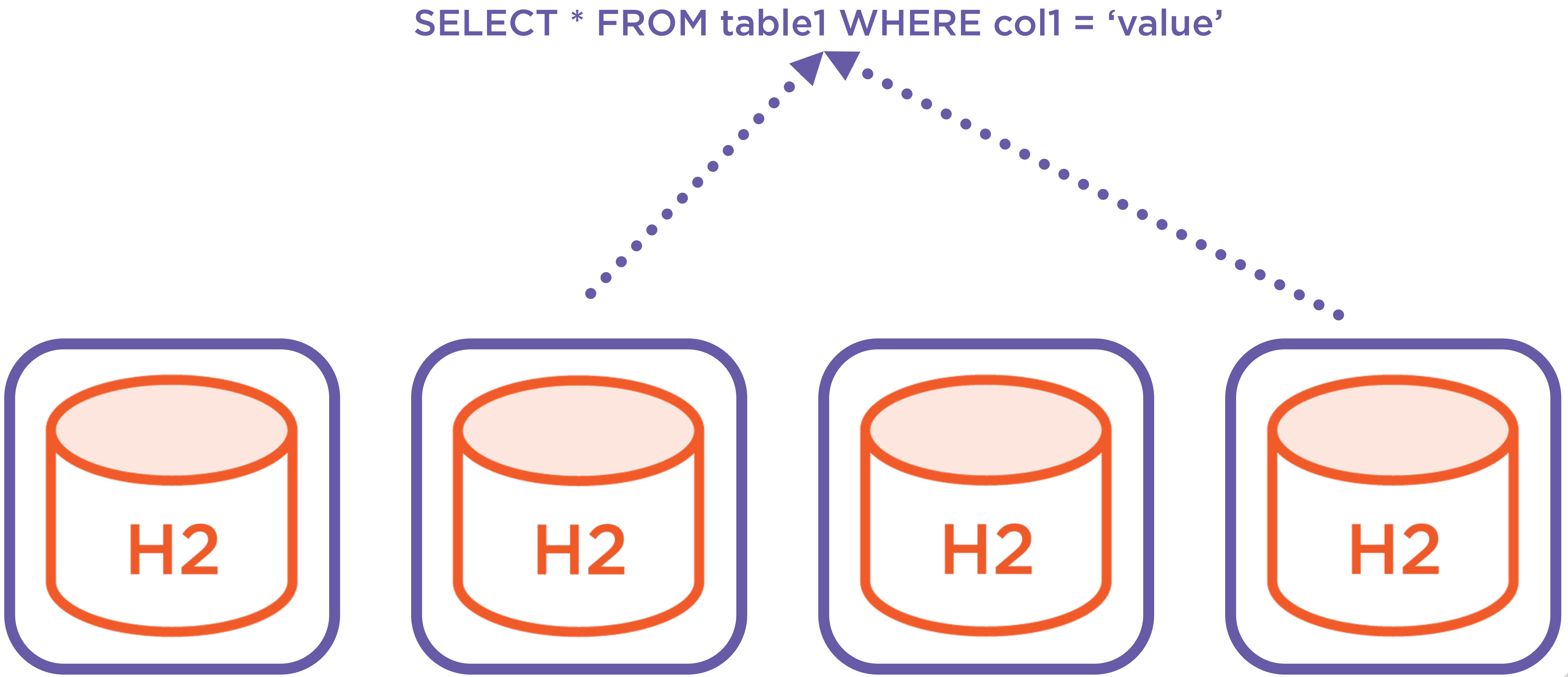

SQL Grid
H2


H2

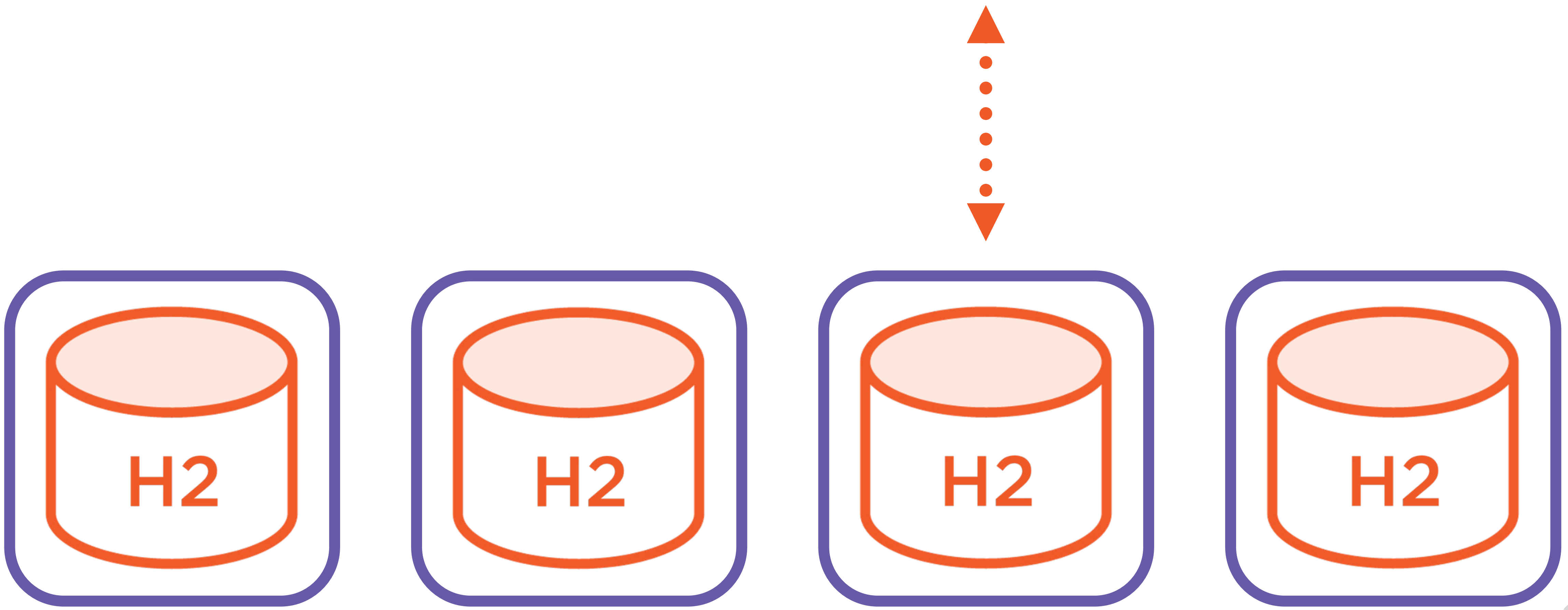
- SQL Parsing
- SQL Optimization
- Execution Planning

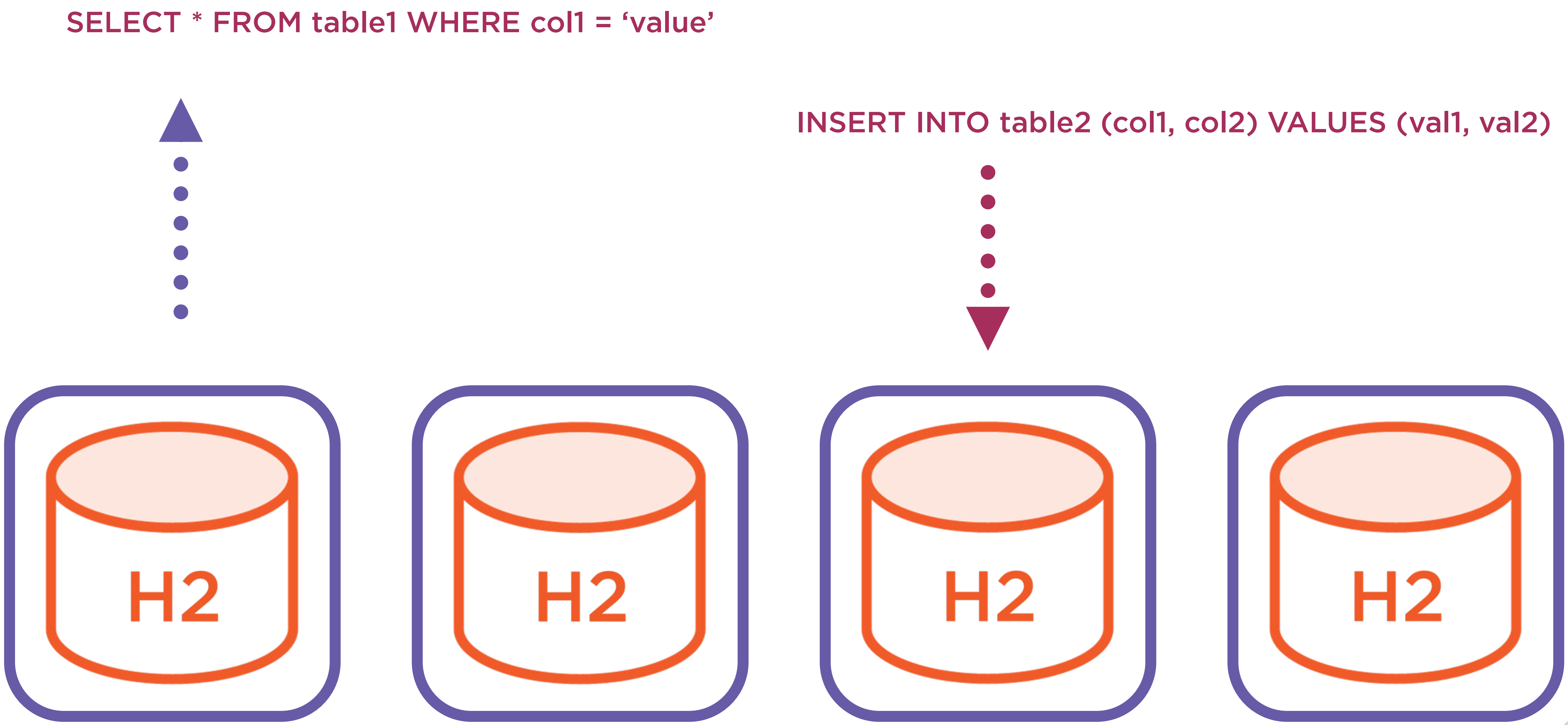



SELECT * FROM table1 WHERE col1 = ‘value’


SELECT * FROM table1 WHERE col1 = ‘value’
H2
H2
H2
H2

SELECT * FROM table1 WHERE col1 = ‘value’

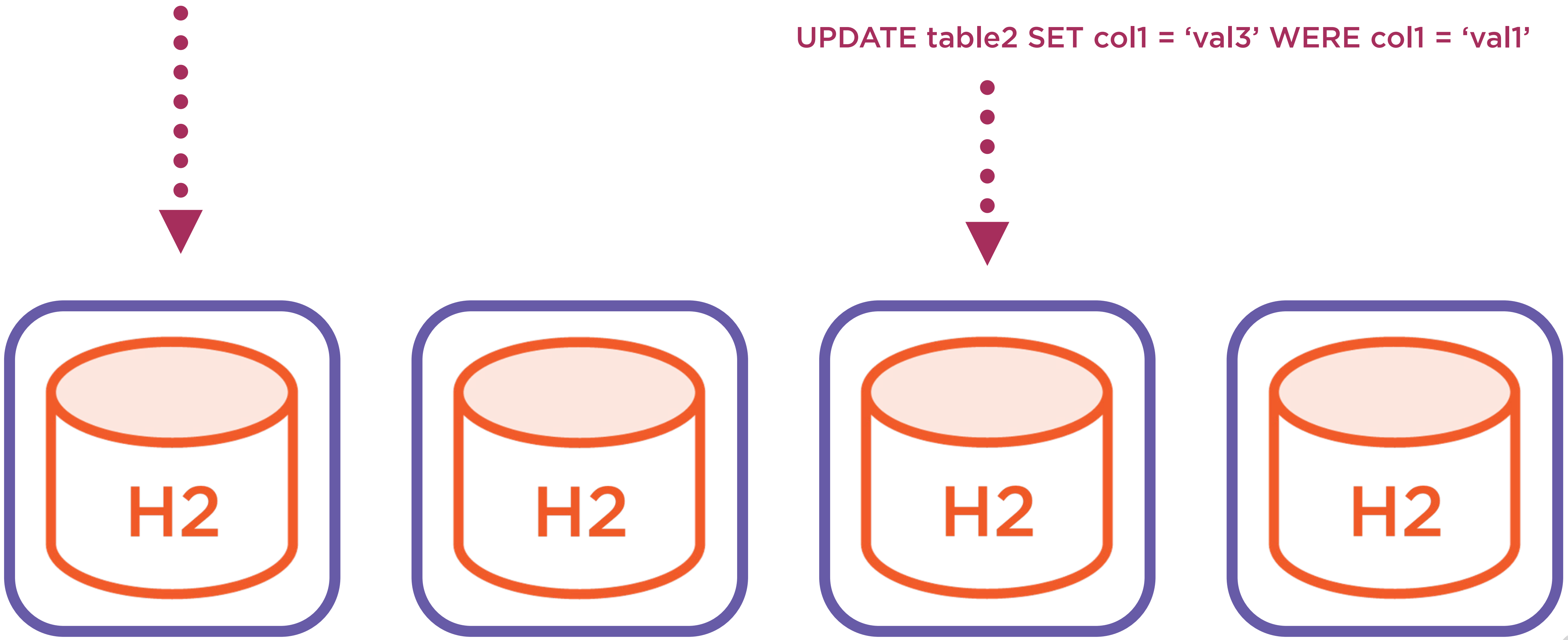


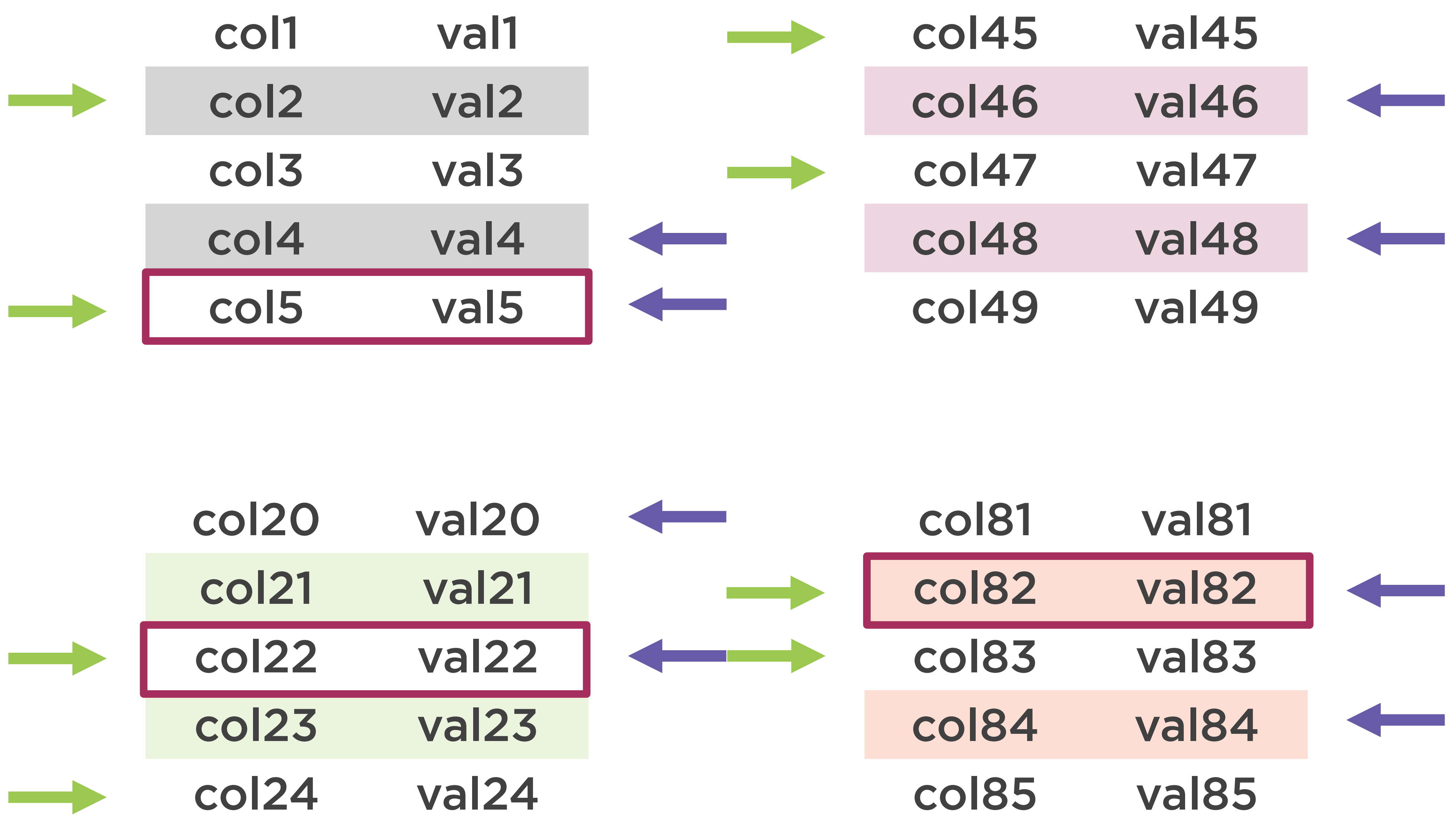

SELECT * FROM table1 WHERE col1 = ‘value’

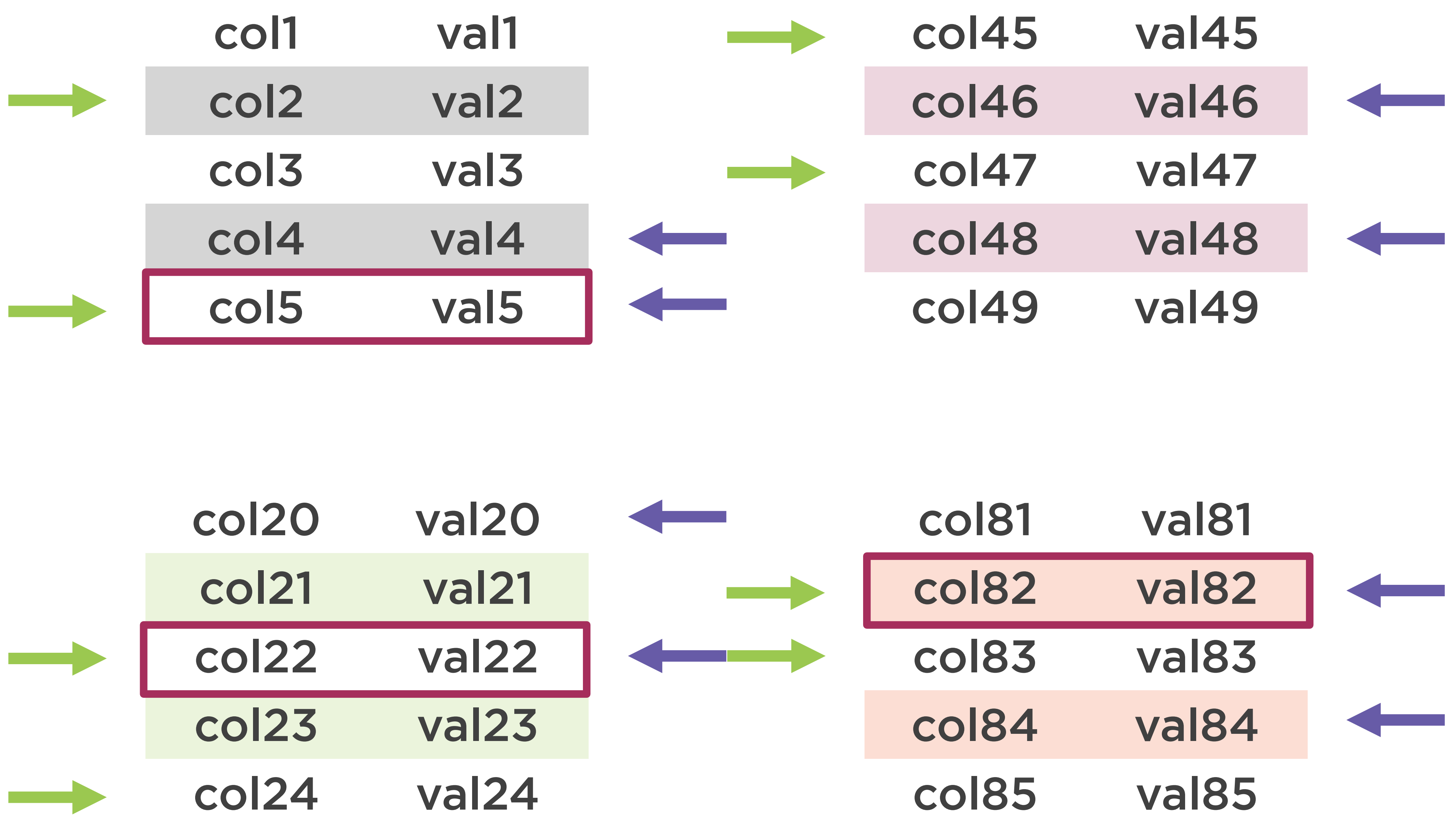
INSERT INTO table2 (col1, col2) VALUES (val1, val2)

DELETE FROM table1 WHERE col1 = ‘value’

UPDATE table2 SET col1 = ‘val3’ WERE col1 = ‘val1’




col1 val1
col2 val2
col3 val3
col4 val4
col5 val5
col45 val45
col46 val46
col47 val47
col48 val48
col49 val49
col20 val20
col21 val21
col22 val22
col23 val23
col24 val24
col81 val81
col82 val82
col83 val83
col84 val84
col85 val85

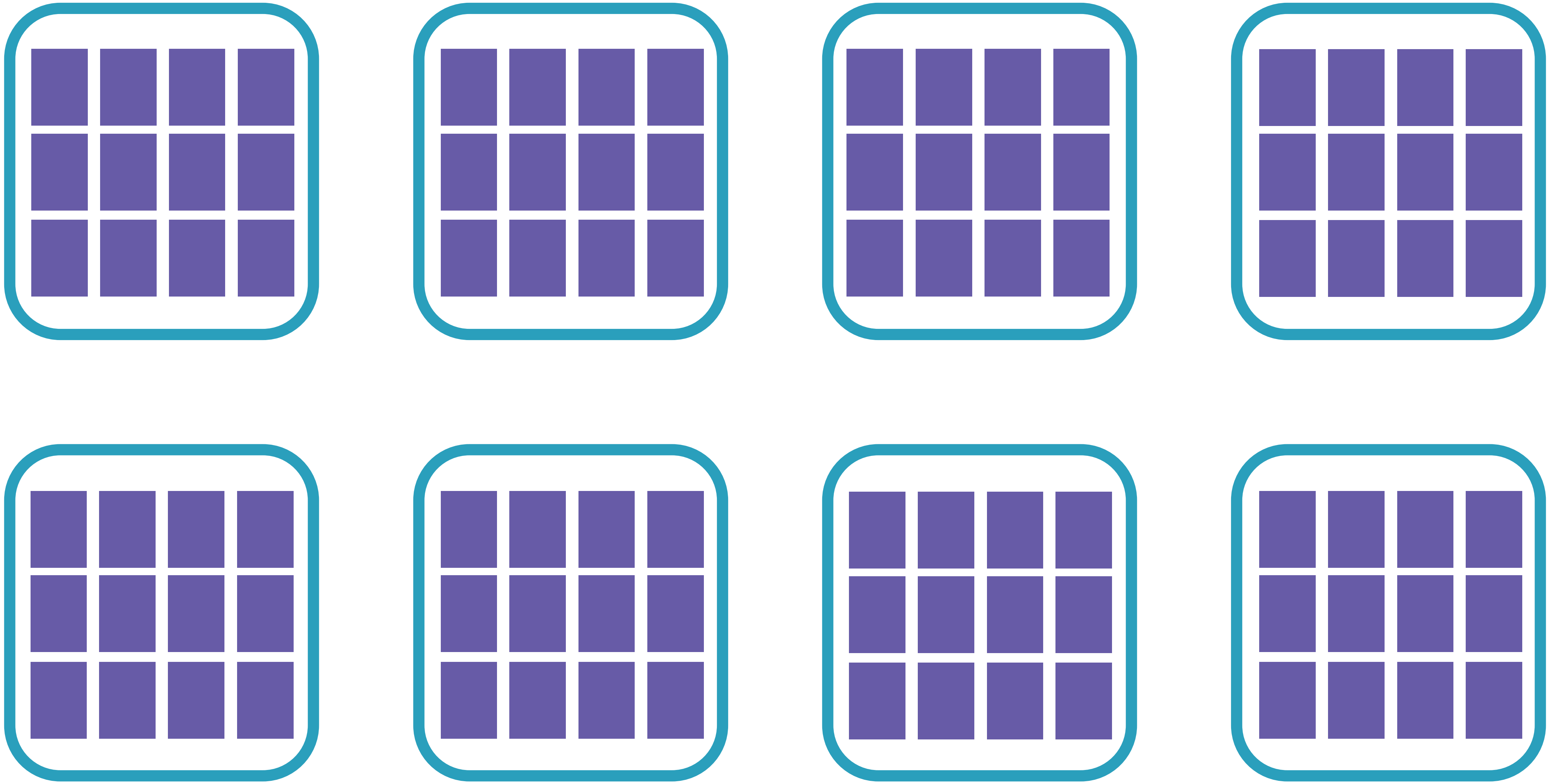


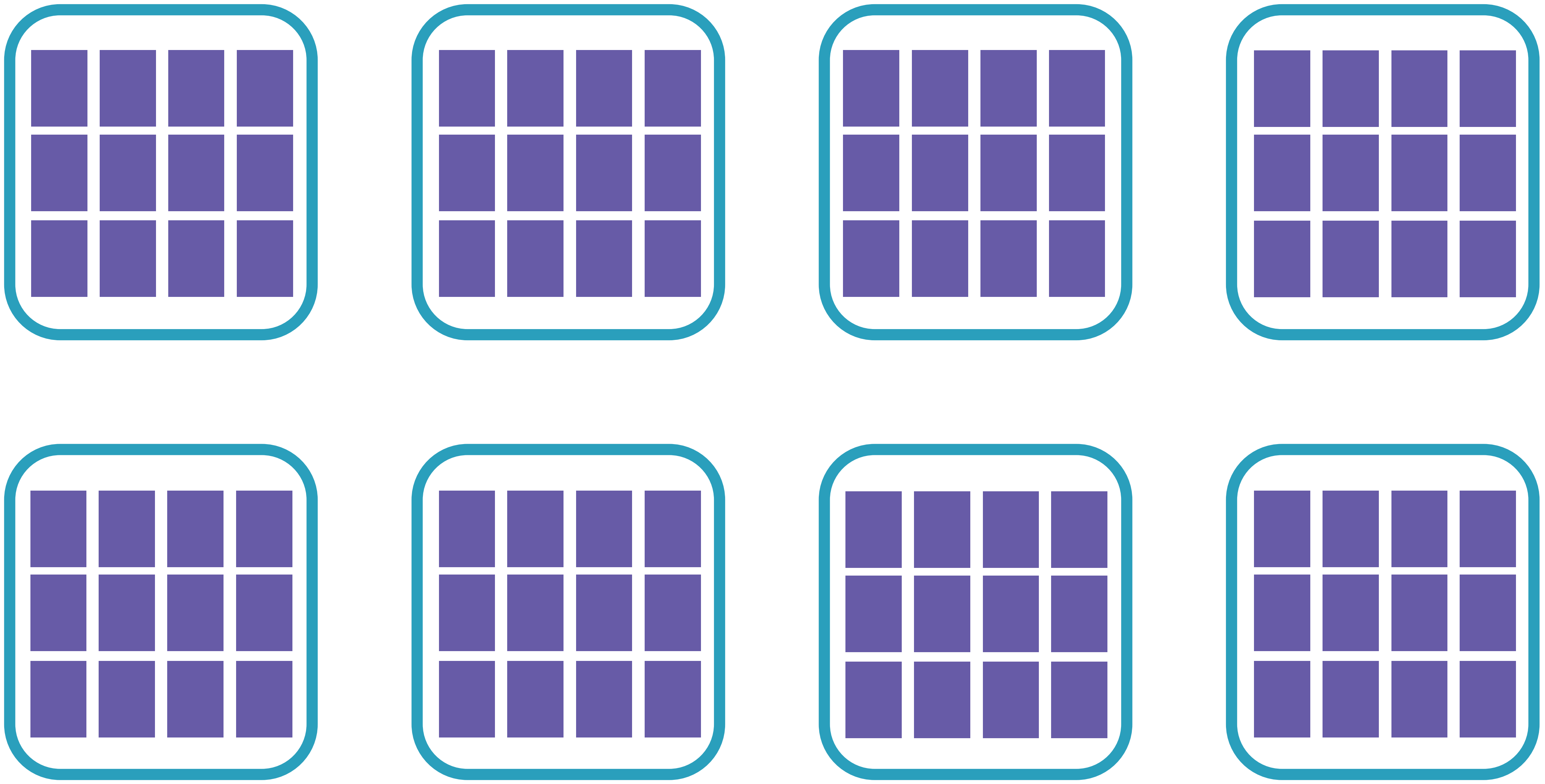
col1	val1
col2	val2
col3	val3
col4	val4
col5	val5

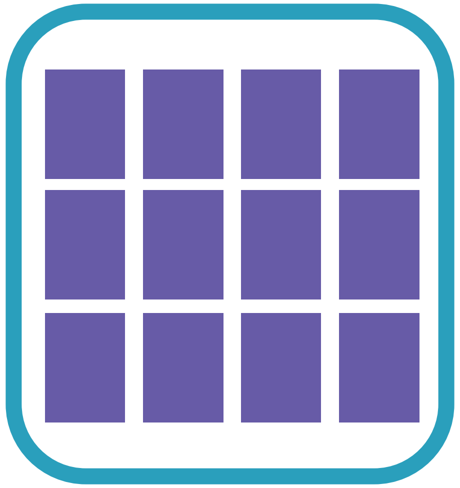
col45	val45
col46	val46
col47	val47
col48	val48
col49	val49

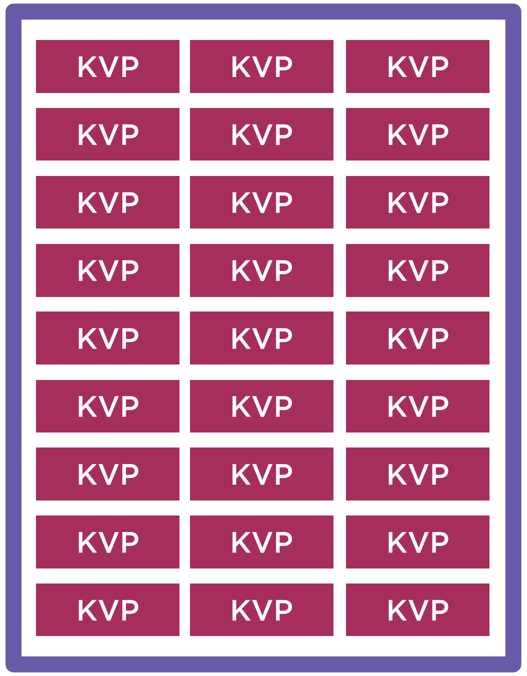
col20	val20
col21	val21
col22	val22
col23	val23
col24	val24

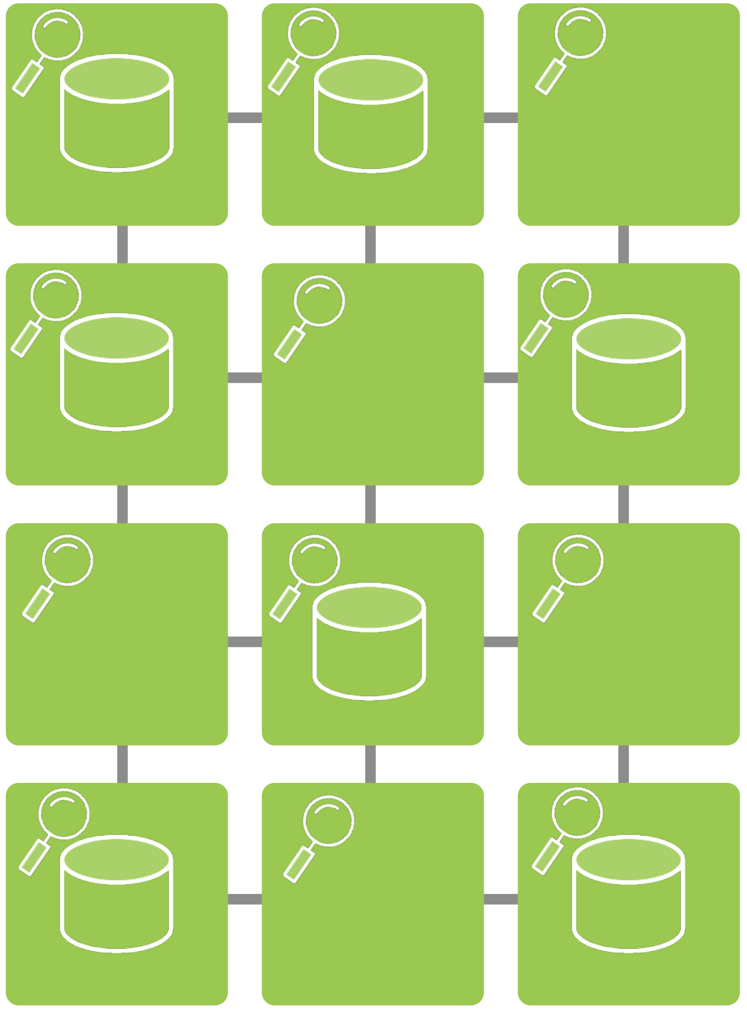
col81	val81
col82	val82
col83	val83
col84	val84
col85	val85



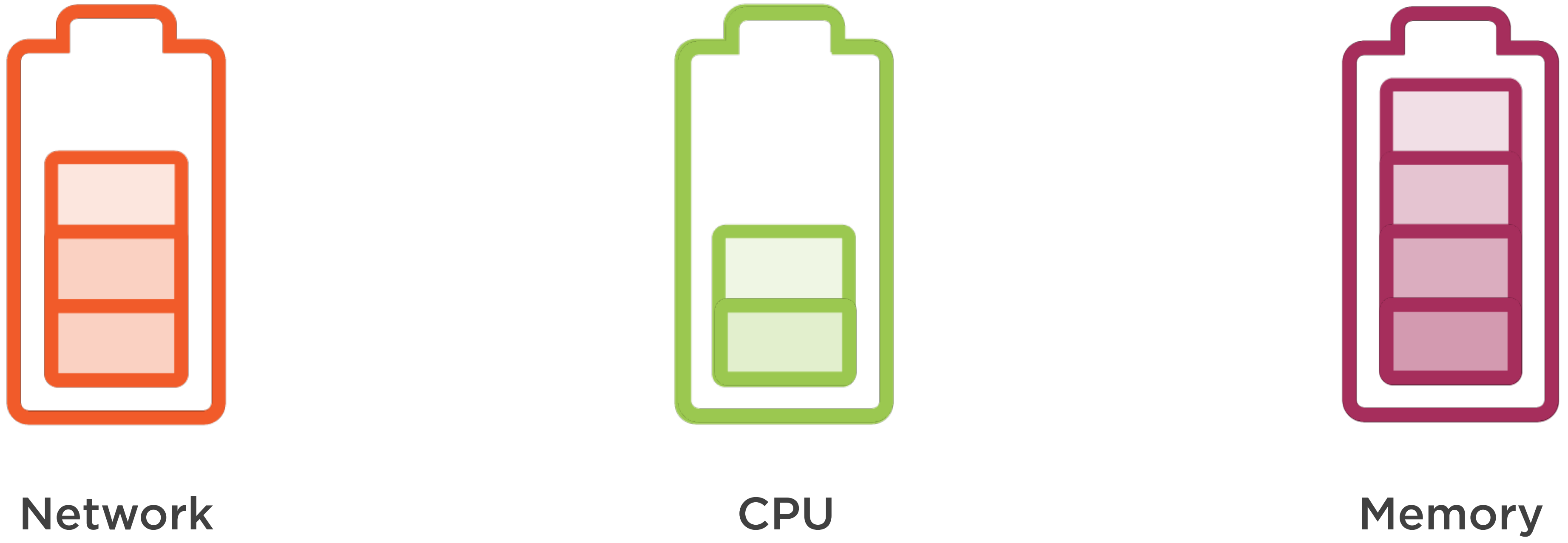


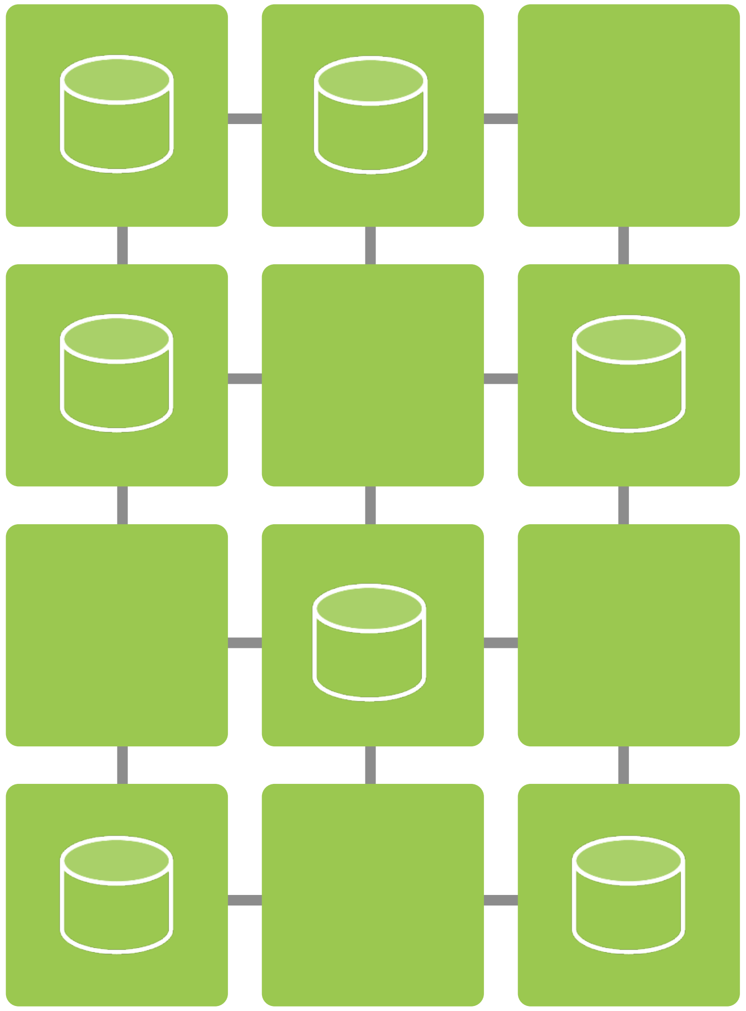


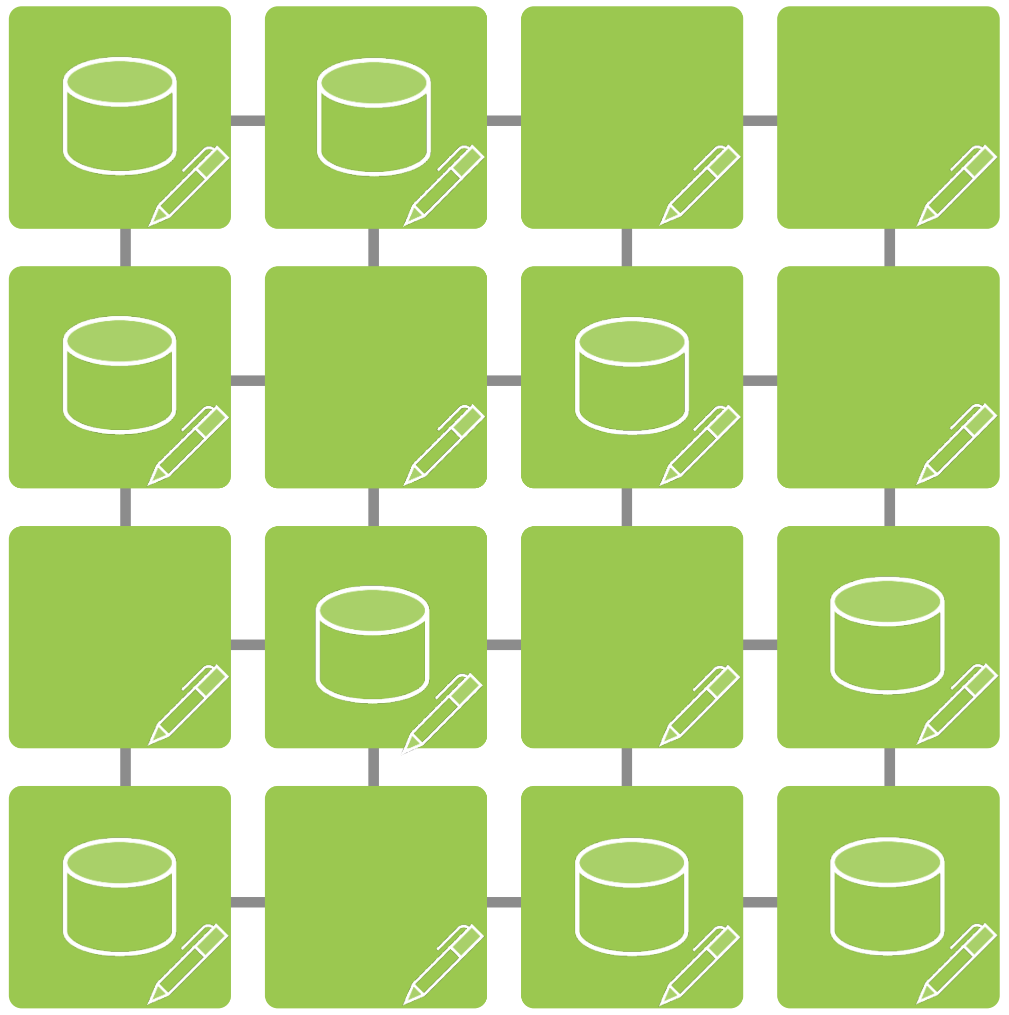

KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP
KVP

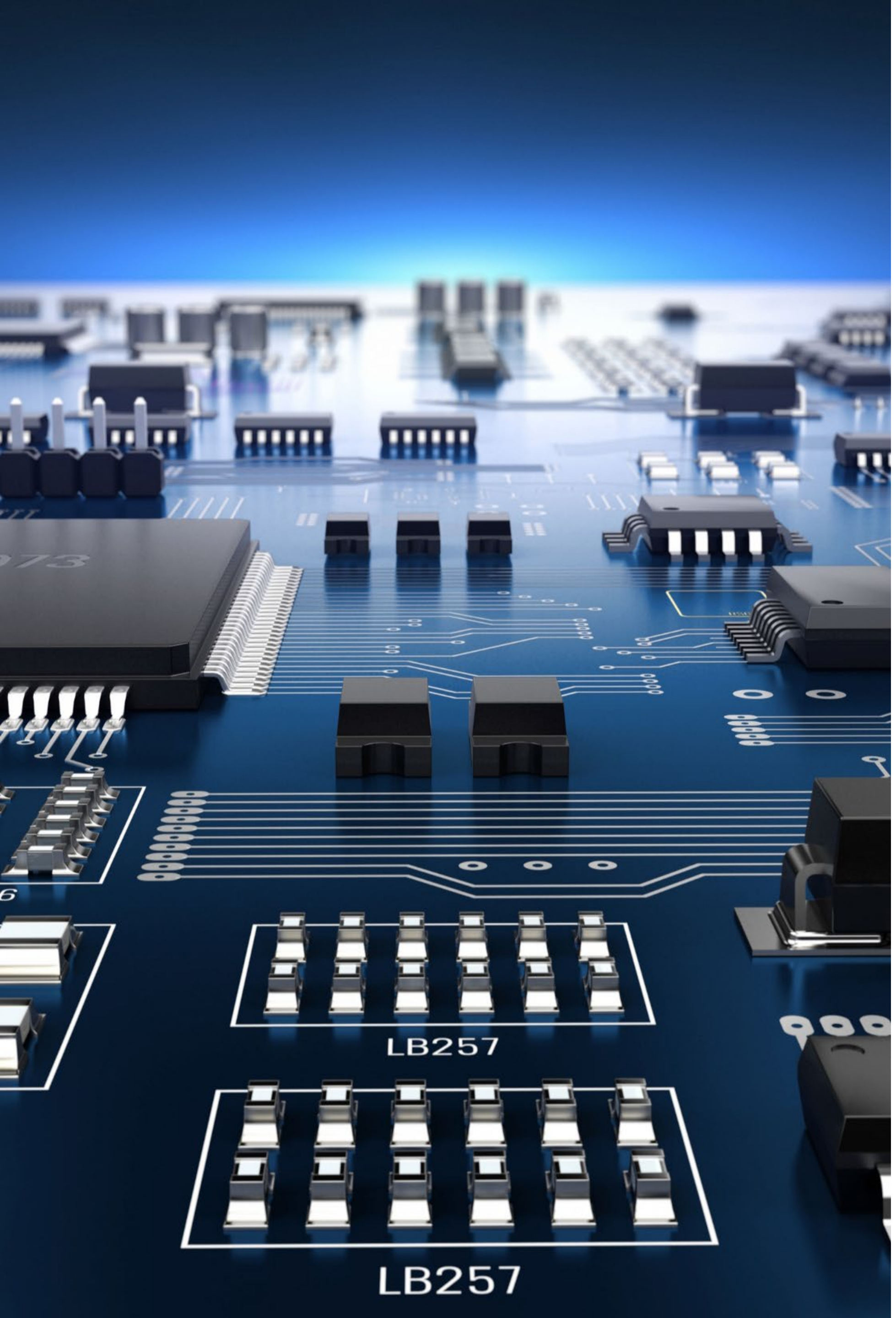


Resources









Low overhead

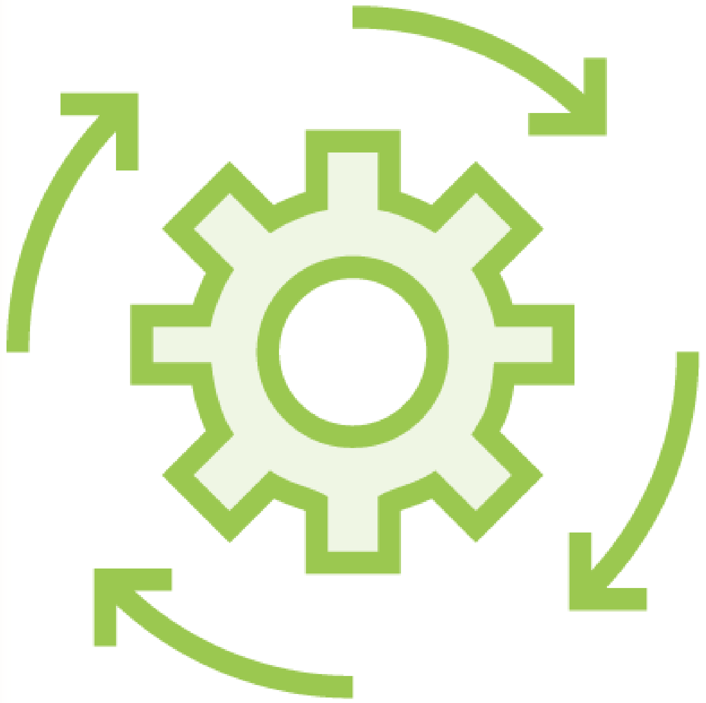
Load balanced

Scalable

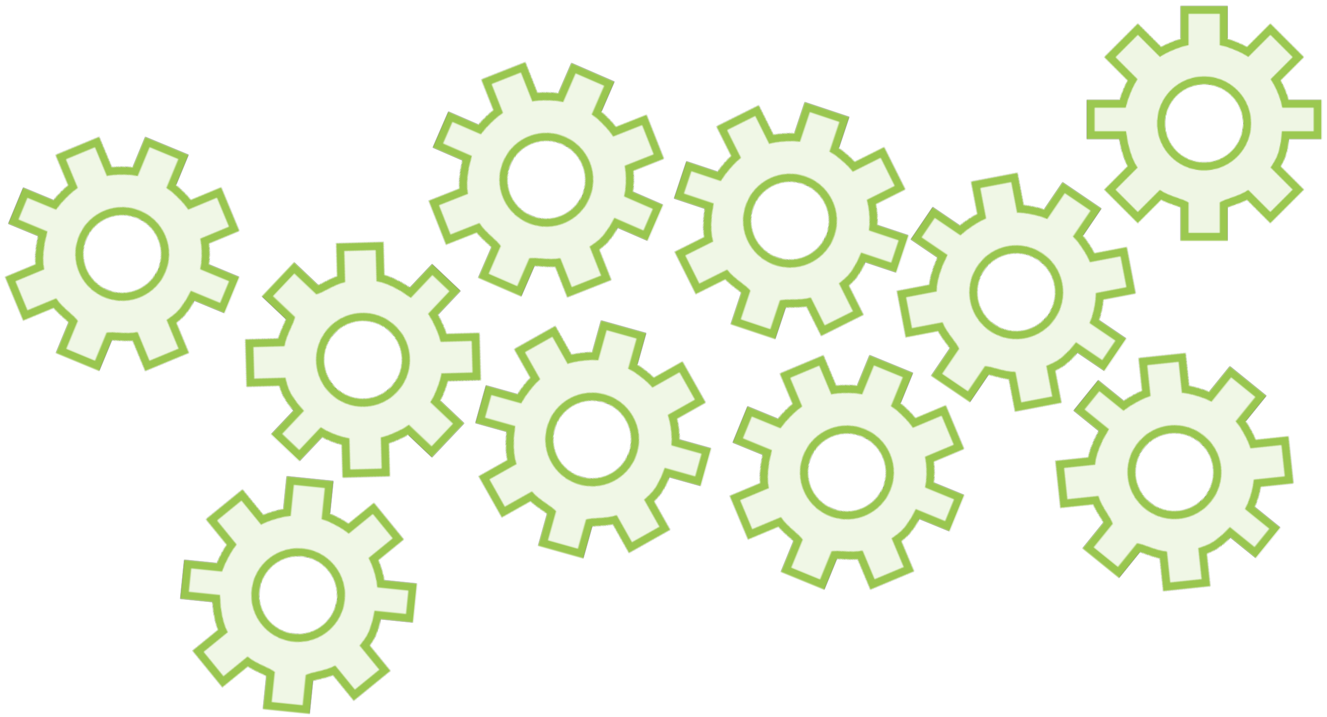
Autonomous

Fault tolerant

Key123ABC



Rendezvous Hashing



Attributes of the Affinity Function

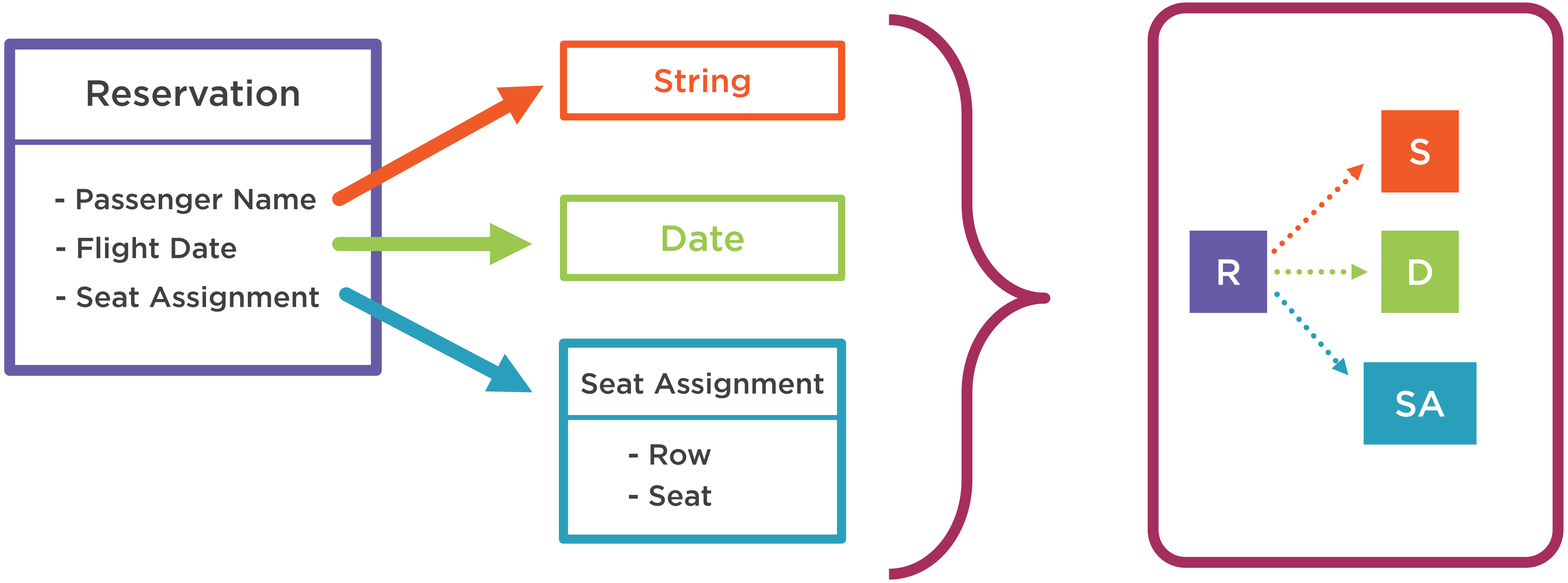
Low Overhead: We get the node and partition with one call

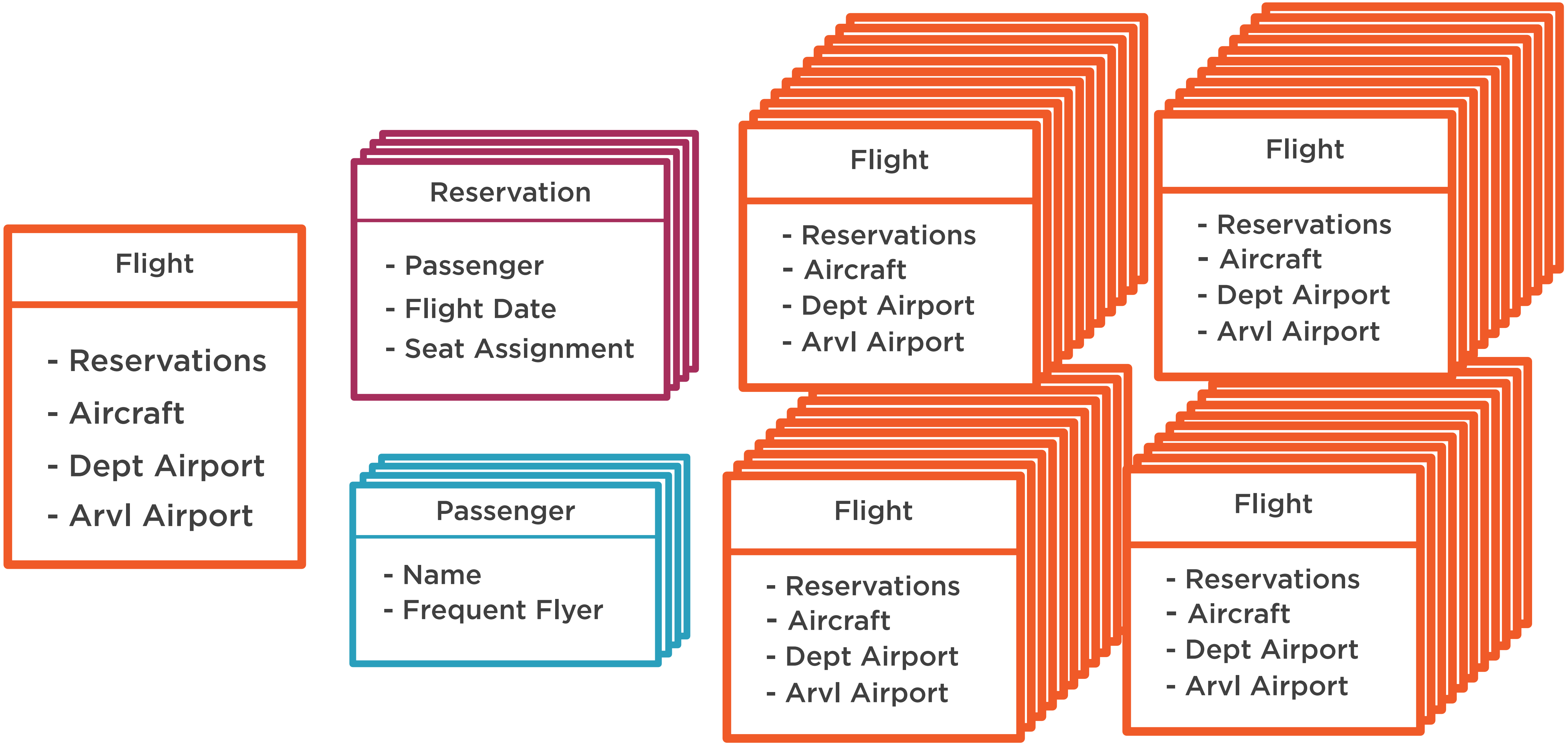
Fault Tolerant: As the topology changes – the load is rebalanced

Smart Load Balancing: Load is distributed in proportion with capacity

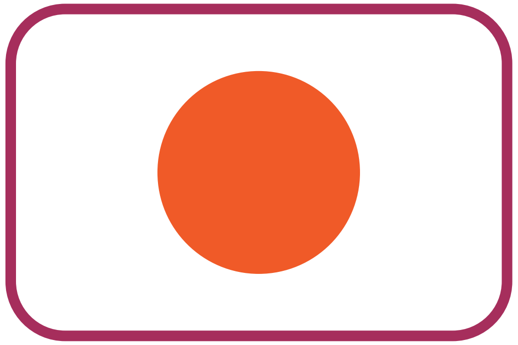
Deterministic: A given key will always result in the same partition / node

Affinity and Collocation


Reservation
- Passenger Name
- Flight Date
- Seat Assignment
String
Date
Seat Assignment
- Row
- Seat
S
R
D
SA

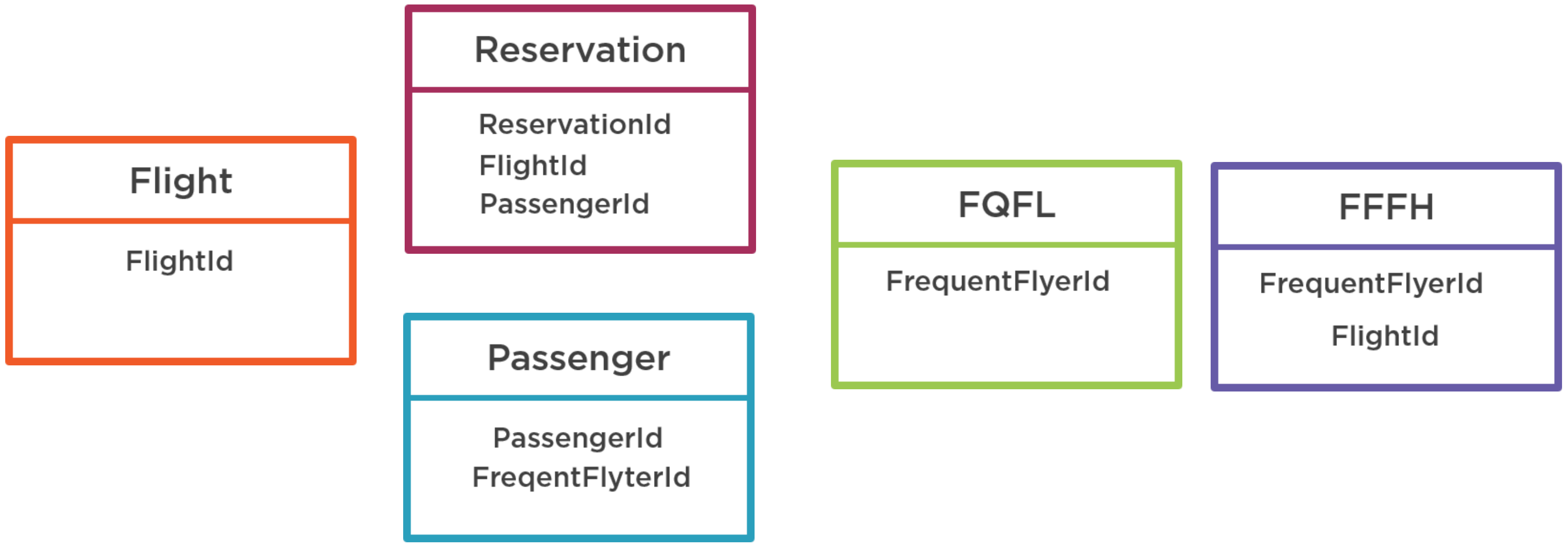

Flight
- Reservations
- Aircraft
- Dept Airport
- Arvl Airport
Reservation
- Passenger
- Flight Date
- Seat Assignment
Passenger
- Name
- Frequent Flyer
Flight
- Reservations
- Aircraft
- Dept Airport
- Arvl Airport
Flight
- Reservations
- Aircraft
- Dept Airport
- Arvl Airport
Flight
- Reservations
- Aircraft
- Dept Airport
- Arvl Airport
Flight
- Reservations
- Aircraft
- Dept Airport
- Arvl Airport

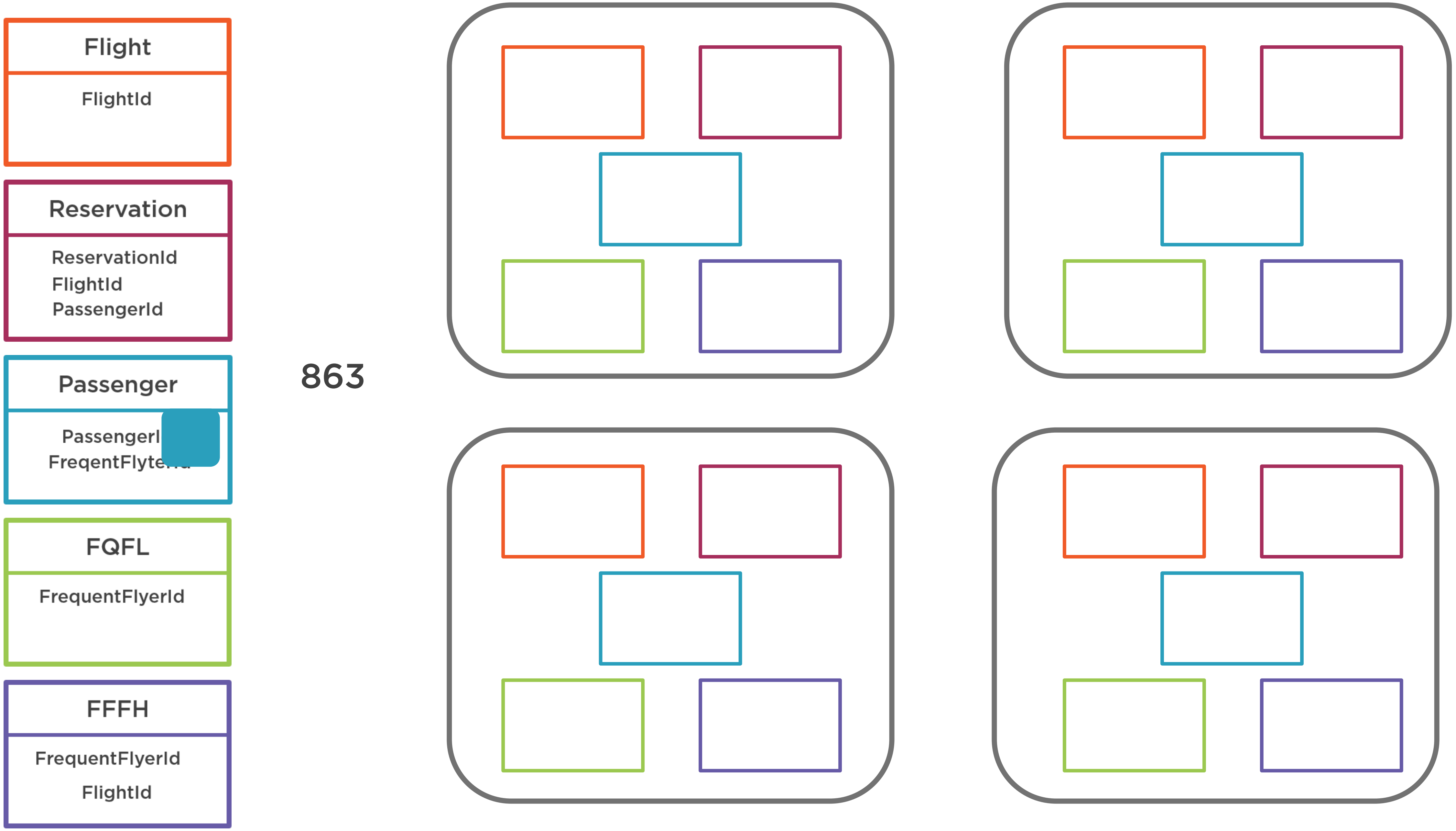
Embedded Objects vs. Referenced Objects

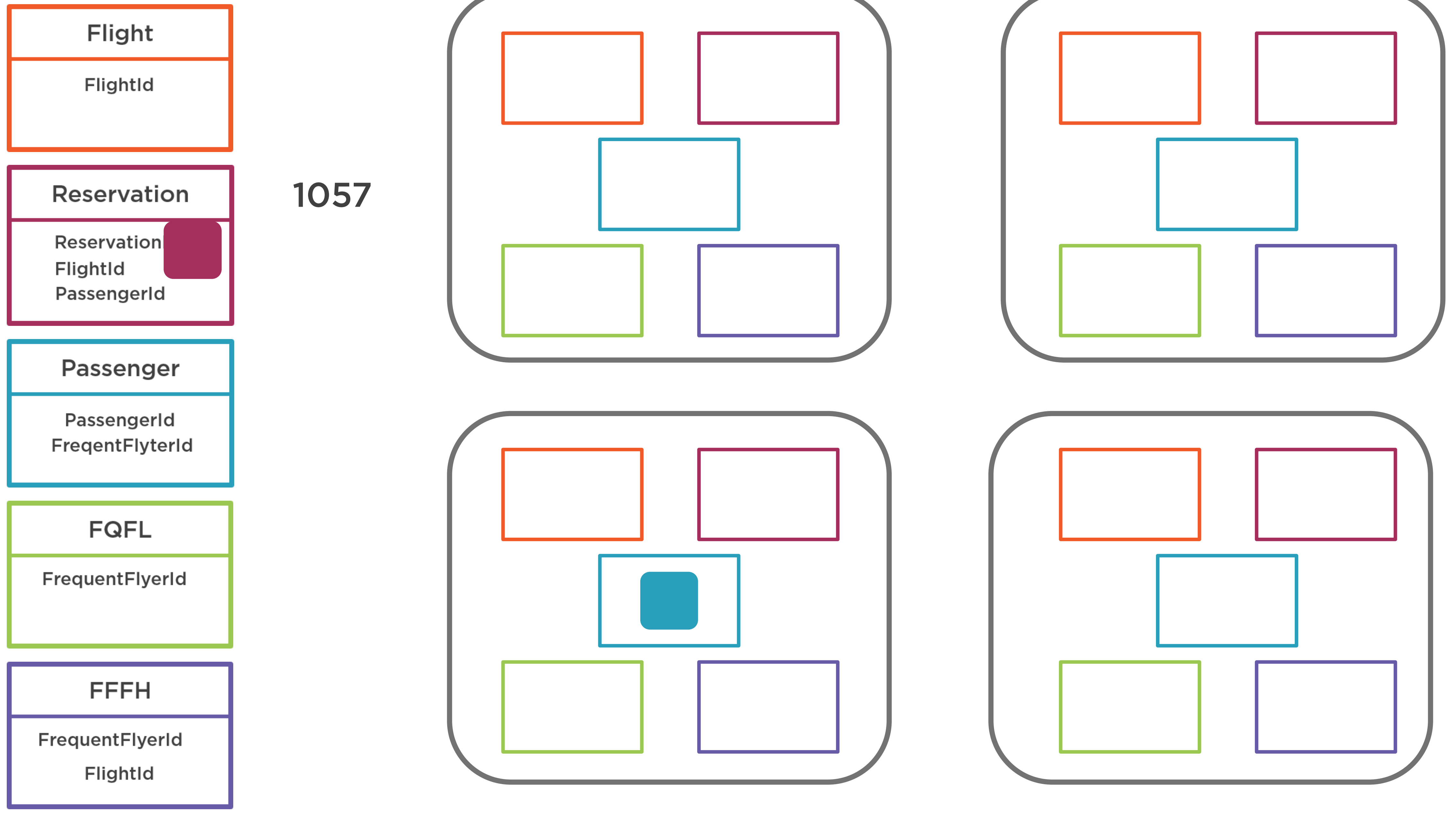


Embedded

By Reference


Flight
FlightId
Reservation
ReservationId
FlightId
PassengerId
Passenger
PassengerId
FreqentFlyterId
FQFL
FrequentFlyerId
FFFH
FrequentFlyerId
FlightId


Flight
FlightId
Reservation
ReservationId
FlightId
PassengerId
Passenger
PassengerI
FreqentFlyte
FQFL
FrequentFlyerId
FFFH
FrequentFlyerId
FlightId
863



Flight
FlightId
Reservation
Reservation
FlightId
PassengerId
Passenger
PassengerId
FreqentFlyterId
FQFL
FrequentFlyerId
FFFH
FrequentFlyerId
FlightId
1057

Flight

FlightId

Reservation

ReservationId
FlightId
PassengerId

Passenger

PassengerId
FreqentFlyterId

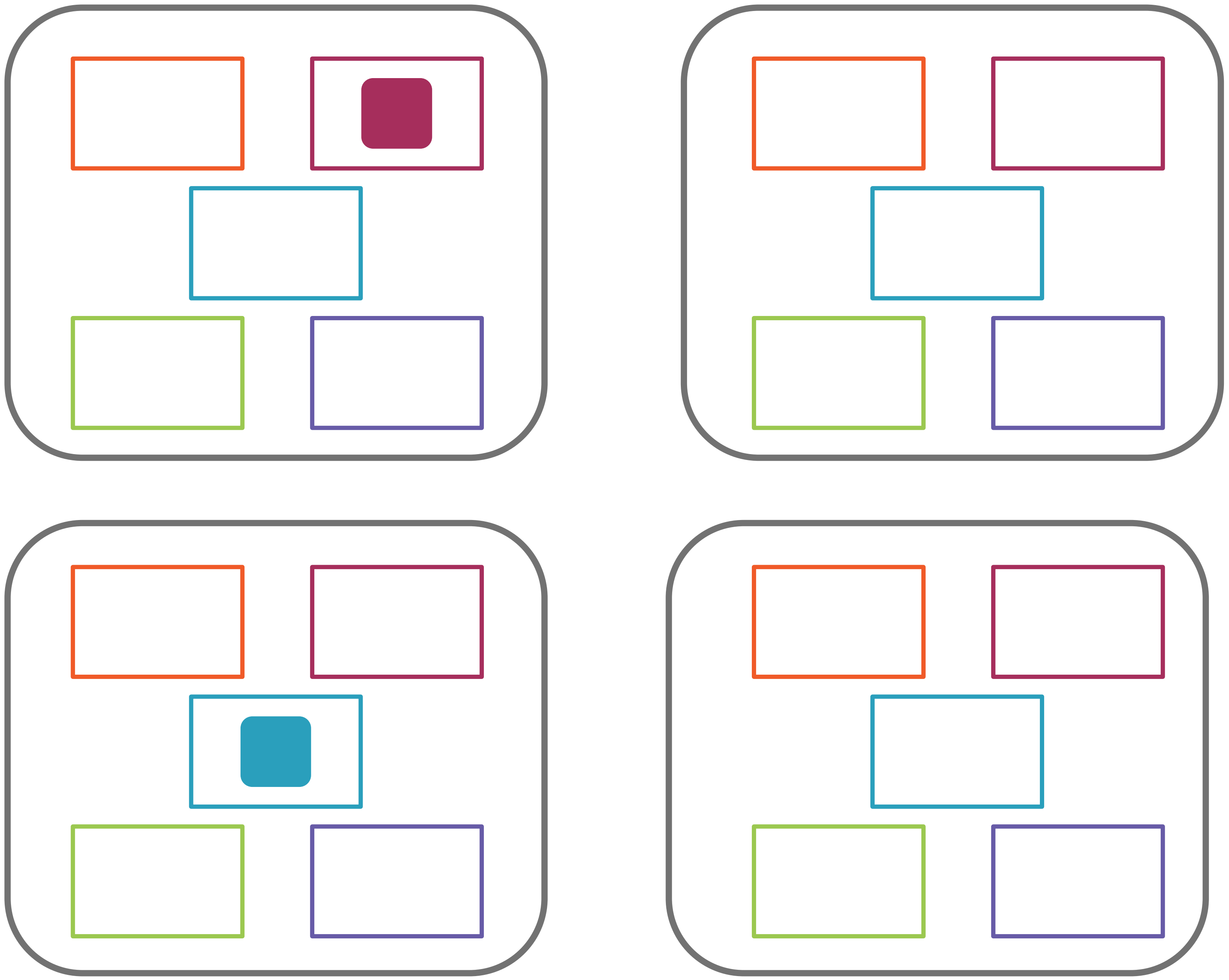
FQFL

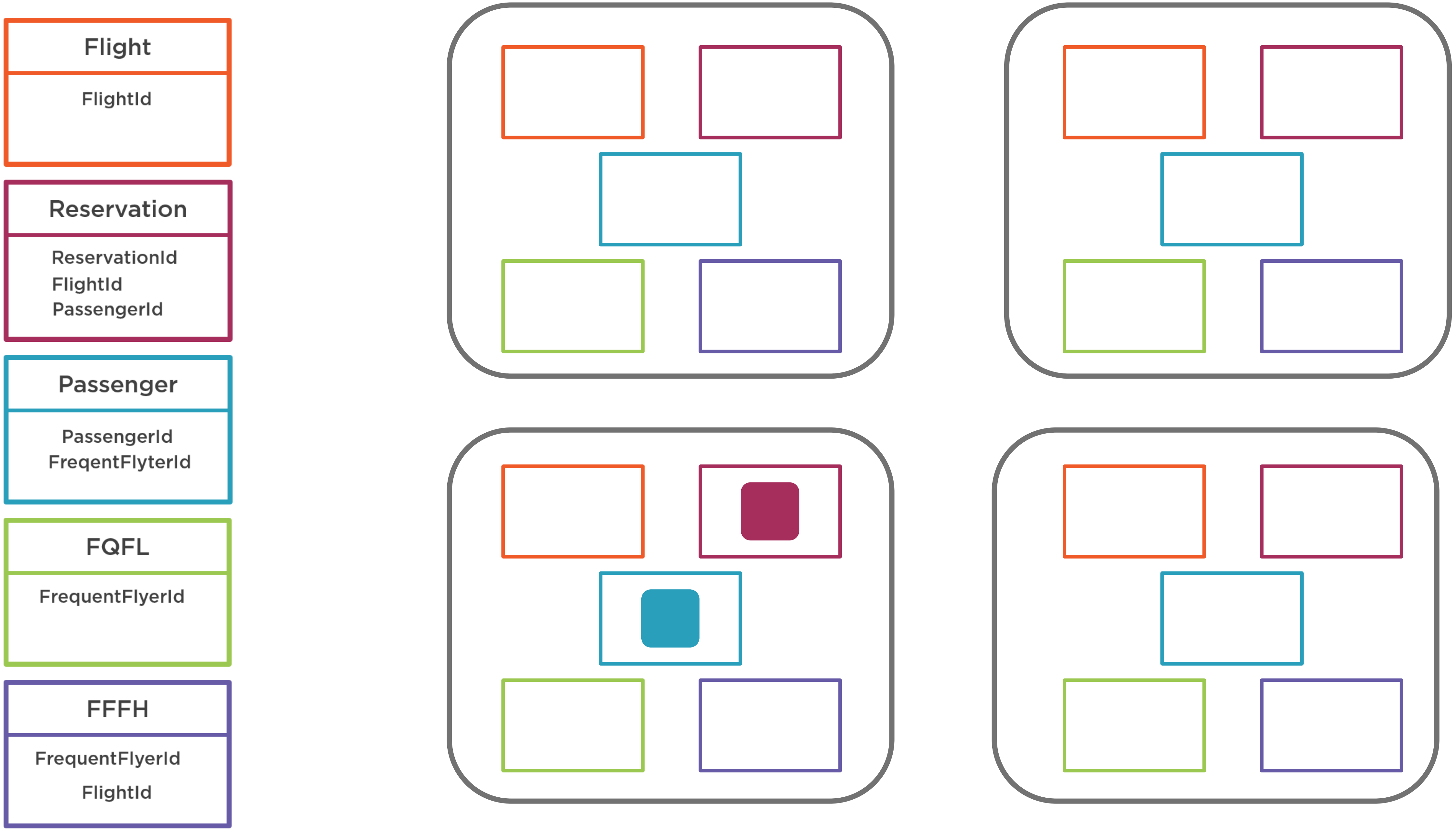
FrequentFlyerId

FFFH

FrequentFlyerId

FlightId




Flight
FlightId
Reservation
ReservationId
FlightId
PassengerId
Passenger
PassengerId
FreqentFlyterId
FQFL
FrequentFlyerId
FFFH
FrequentFlyerId
FlightId

Affinity Key Construction

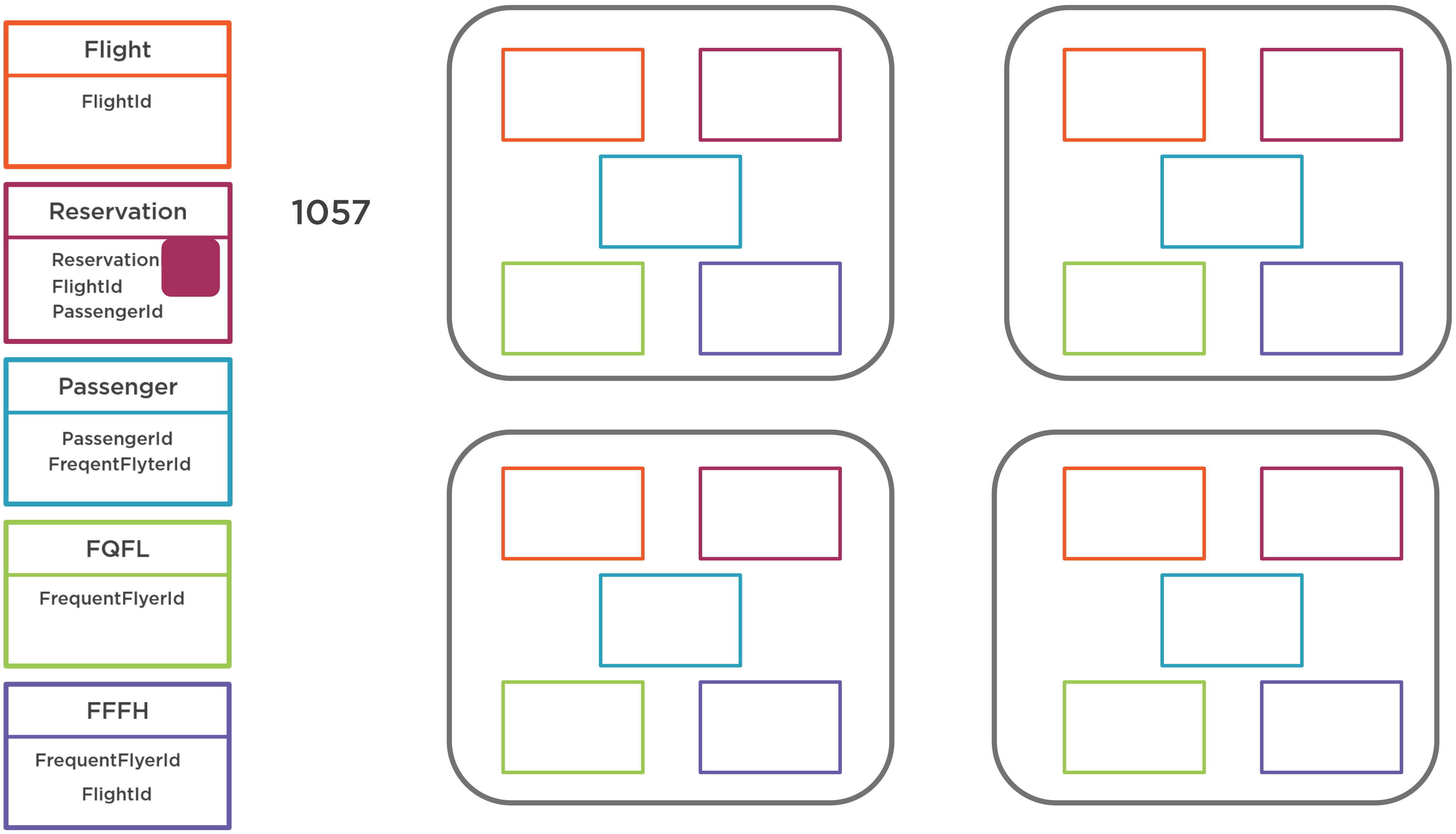
```
public class Reservation
{
     private int reservationId;
     private int passengerId;

     public Reservation(int passengerId, int reservationId) {

          …
     }

     public AffinityKey getKey() {
          return new AffinityKey(passengerId, reservationId);
     }

     …
}
```


Flight
FlightId
Reservation
Reservation
FlightId
PassengerId
Passenger
PassengerId
FreqentFlyterId
FQFL
FrequentFlyerId
FFFH
FrequentFlyerId
FlightId
1057

Flight

FlightId

Reservation

ReservationId
FlightId
PassengerId

Passenger

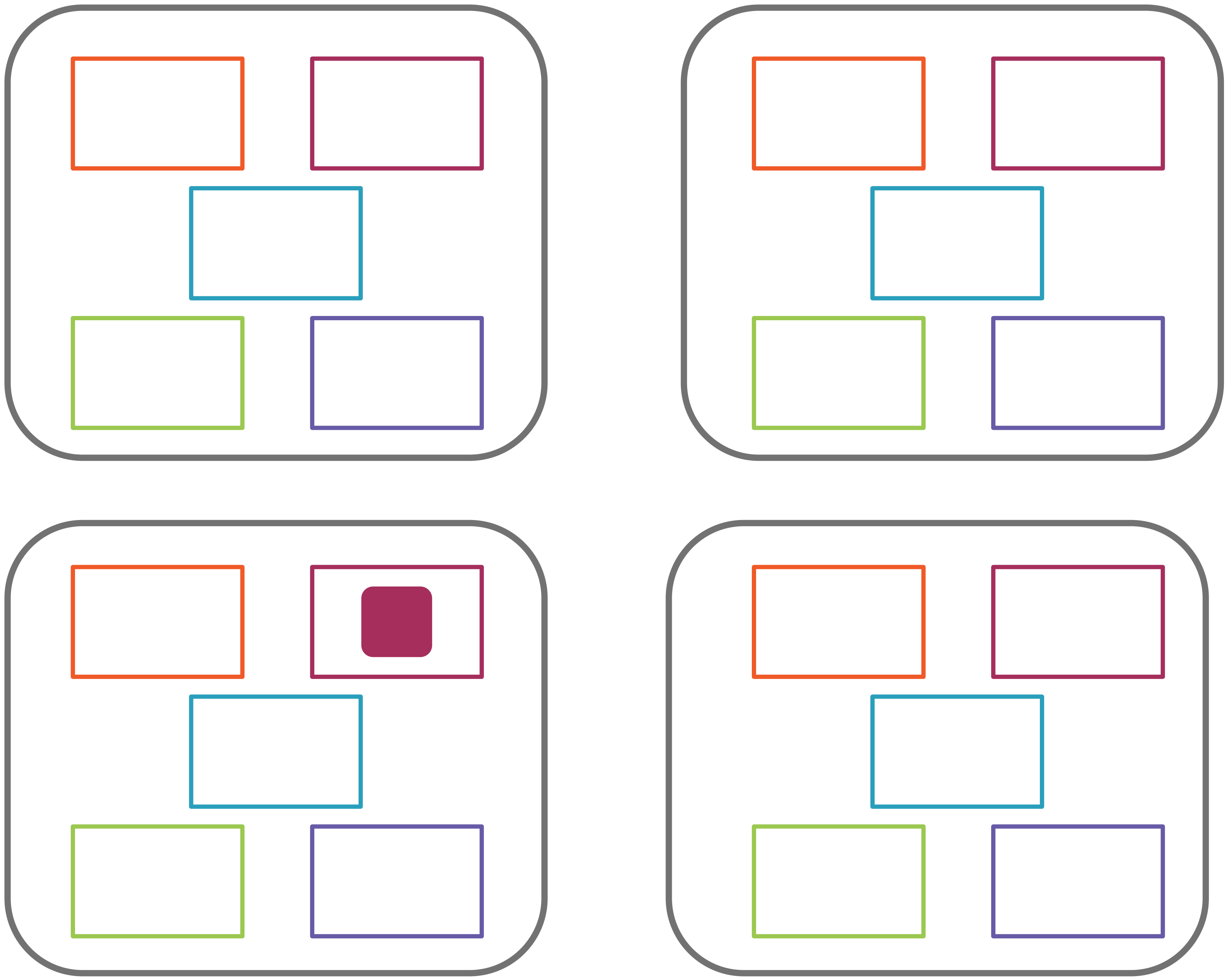
PassengerId
FreqentFlyterId

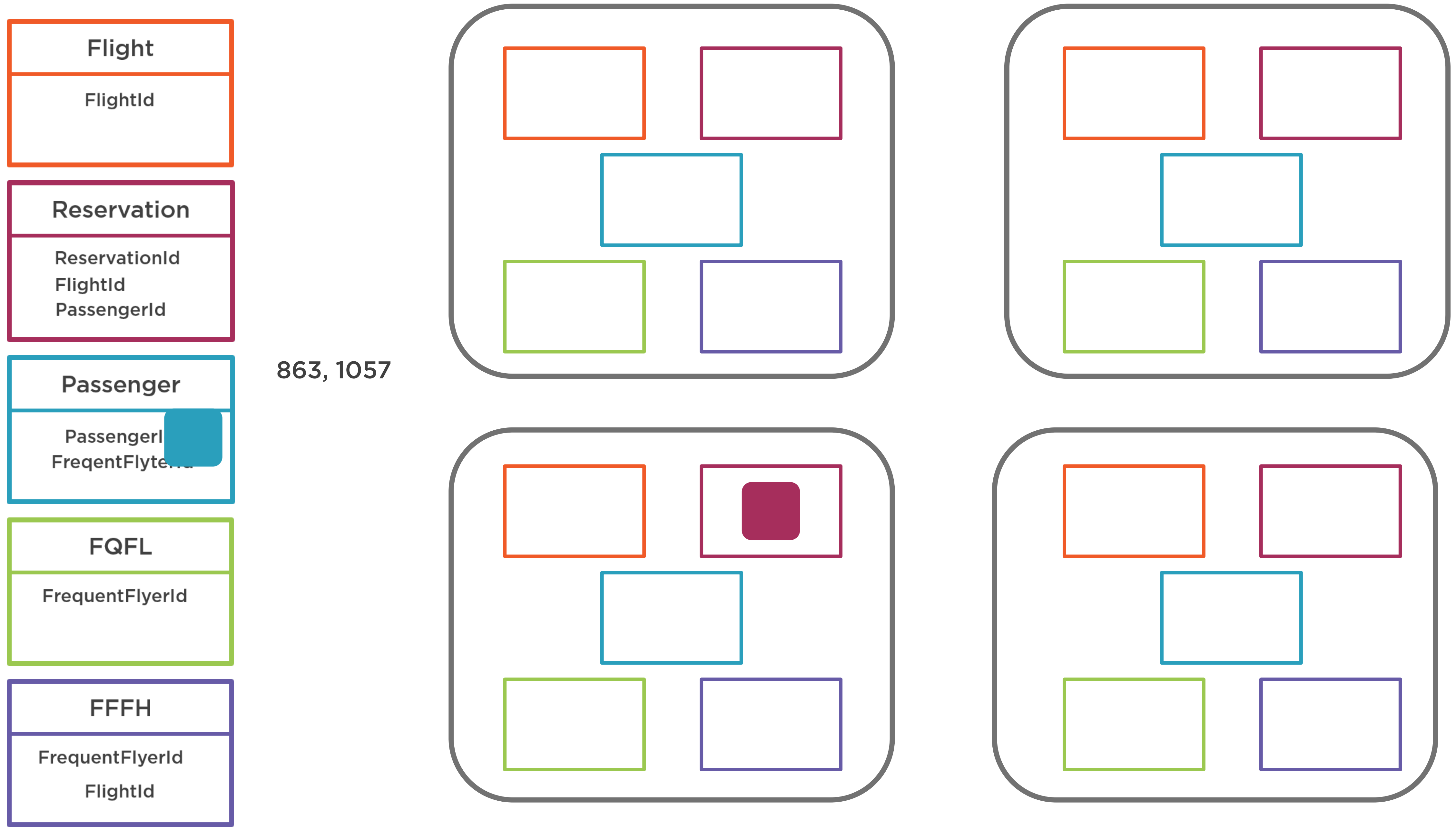
FQFL

FrequentFlyerId

FFFH

FrequentFlyerId
FlightId




Flight
FlightId
Reservation
ReservationId
FlightId
PassengerId
Passenger
PassengerI
FreqentFlyte
FQFL
FrequentFlyerId
FFFH
FrequentFlyerId
FlightId
863, 1057

Flight

FlightId

Reservation

ReservationId
FlightId
PassengerId

Passenger

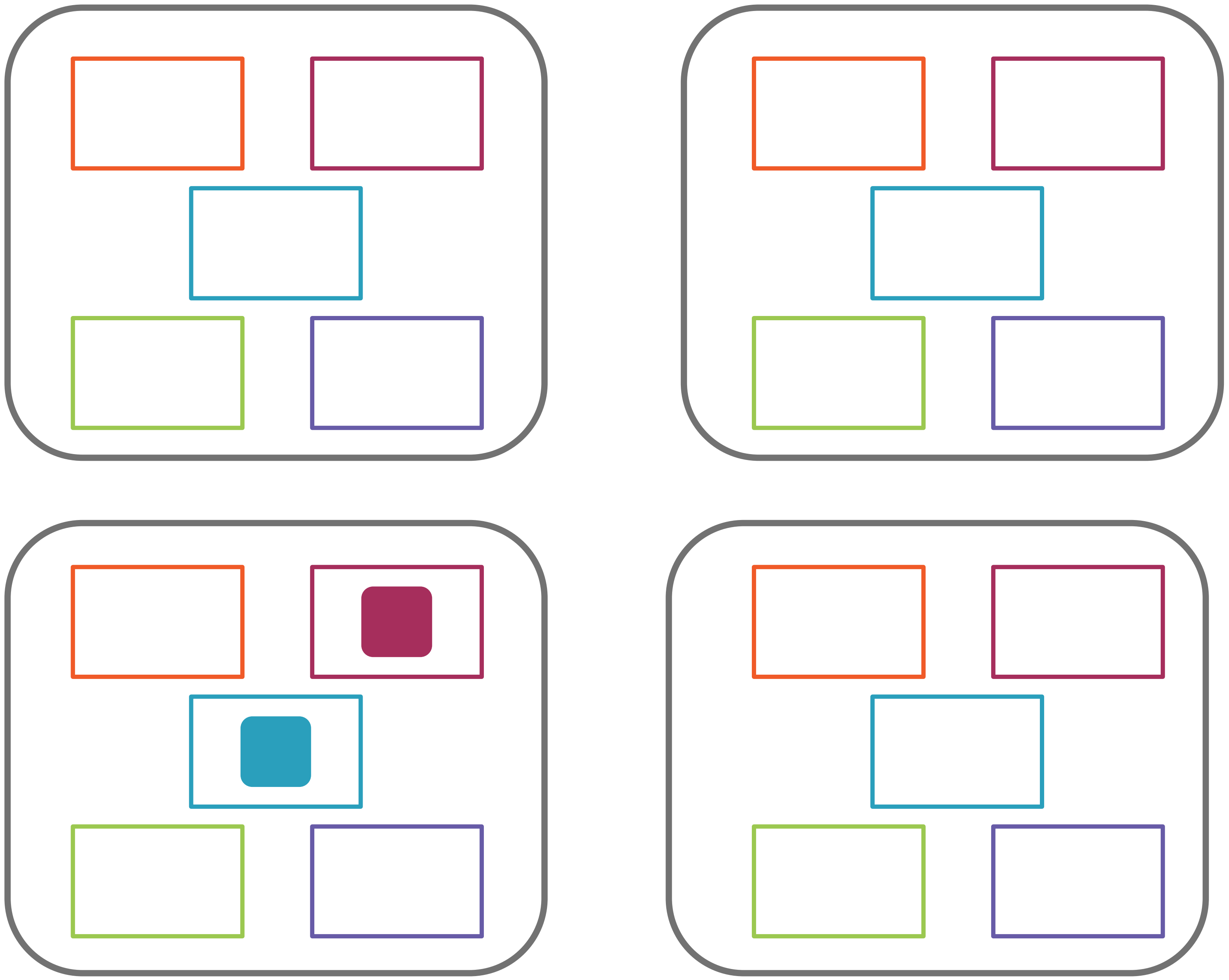
PassengerId
FreqentFlyterId

FQFL

FrequentFlyerId

FFFH

FrequentFlyerId
FlightId



Cache One

- 🔑 Integer

Cache Two

- 🔑 String

Cache One

- 🔑 Double

Flights

- 🔑 Flight Number
- 🔑 Flight Date
- 🔑 Origin
- 🔑 Destination

Flights

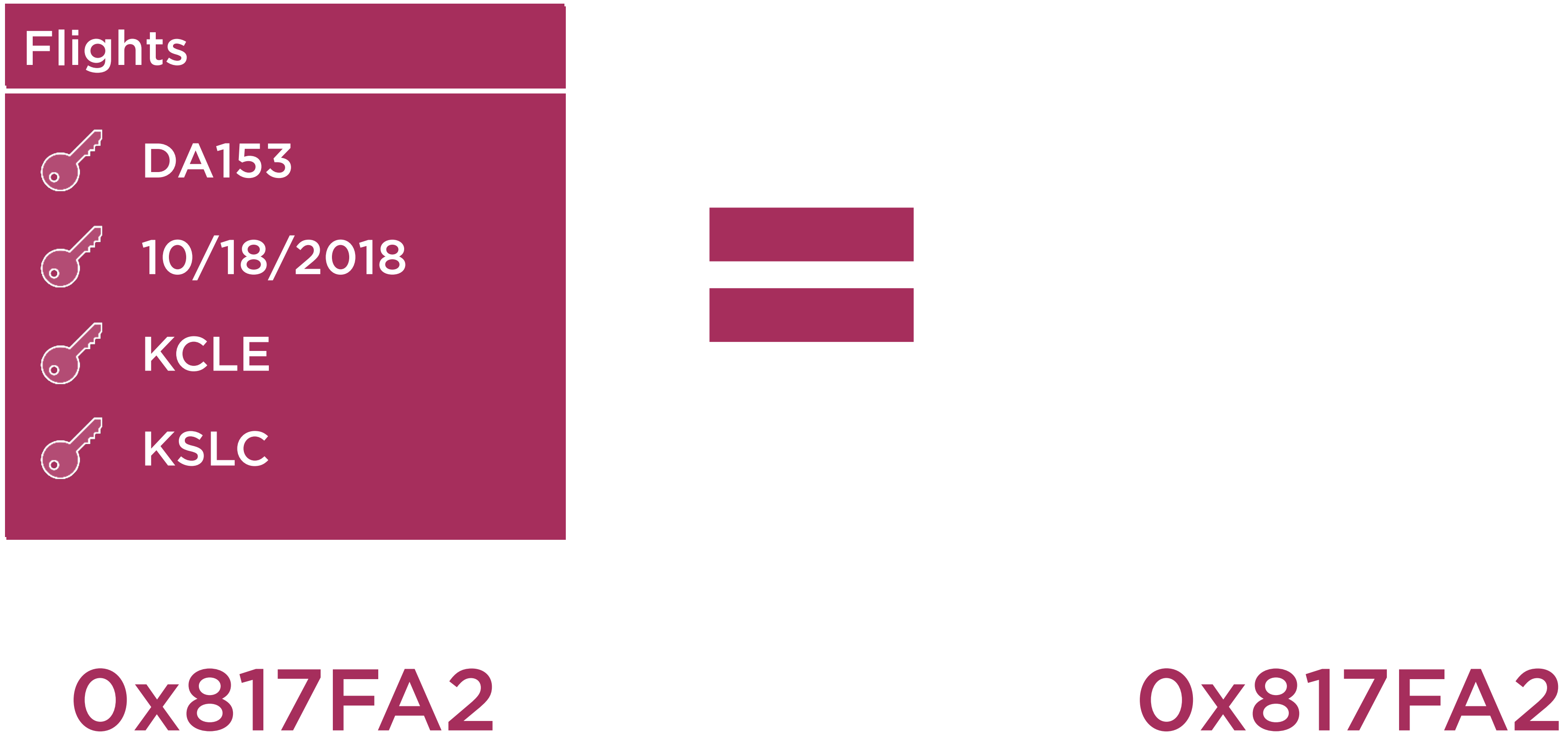
- 🔑 Flight Number
- 🔑 Flight Date
- 🔑 Origin
- 🔑 Destination

Flights

- 🔑 DA153
- 🔑 10/18/2018
- 🔑 KCLE
- 🔑 KSLC

```
de
boolean equals(Object o)

de
int hashCode()
```


Flights
DA153
10/18/2018
KCLE
KSLC
0x817FA2
0x817FA2

Data Queries

QueryCursor = IgniteCache.query(Query)

Extensions of the Abstract Query Class

SqlQuery

SqlFieldsQuery

Extensions of the Abstract Query Class

Wrapper for a SQL statement

QueryCursor<List<?>>

SqlFieldsQuery

Extensions of the Abstract Query Class

SqlQuery

SqlFieldsQuery

Extensions of the Abstract Query Class

SqlQuery

Pass type and WHERE clause

QueryCursor<Entry<Key, Value>>

Flight

FlightId

Reservation

ReservationId
FlightId
PassengerId

Passenger

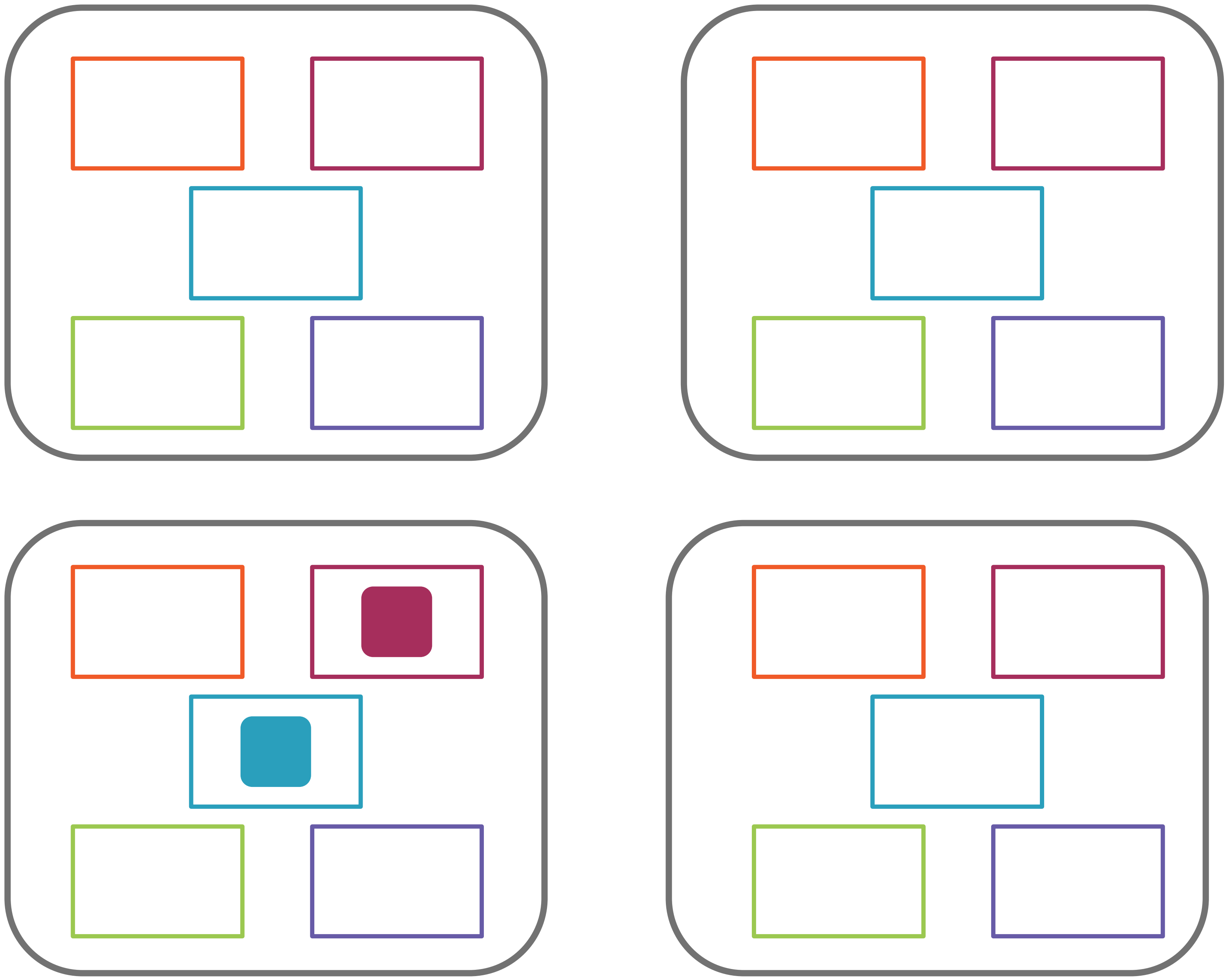
PassengerId
FreqentFlyterId

FQFL

FrequentFlyerId

FFFH

FrequentFlyerId
FlightId



Continuous Query

Initial Query

Remote Filter

Local Listener

The initial query is run immediately.

The remote filter is the ongoing cache listener.

The local listener receives the data that is found on the remote nodes.

Receive 1 notification

All nodes will flush queues

Cache counters sent with data

Client sends confirmation

Bit of code that runs on the server

Sends the data that’s changed

Locks the entry while it’s updating

The SQL API & DML

Use SqlFieldsQuery

```
SqlQuery(Class<?> type, String sql)
```

Apache Ignite SqlQuery class

Only returns data from a cache. Cannot do any DML.

Use SqlFieldsQuery

Subquery must have all data collocated

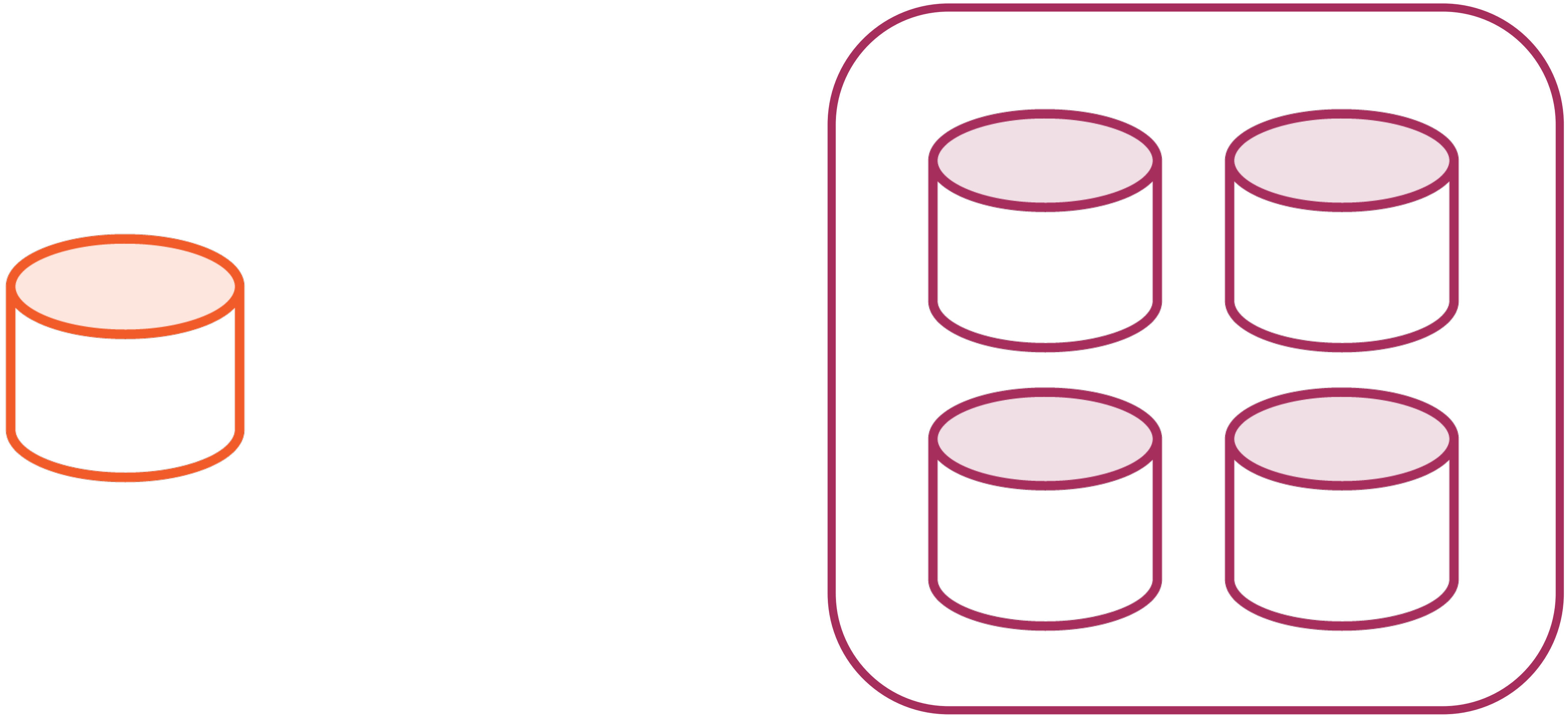
Prefer JCache API methods

- Bulk methods
  - putAll() / getAll()
- Atomic methods
  - putIfAbsent()
  - getAndPutIfAbsent()

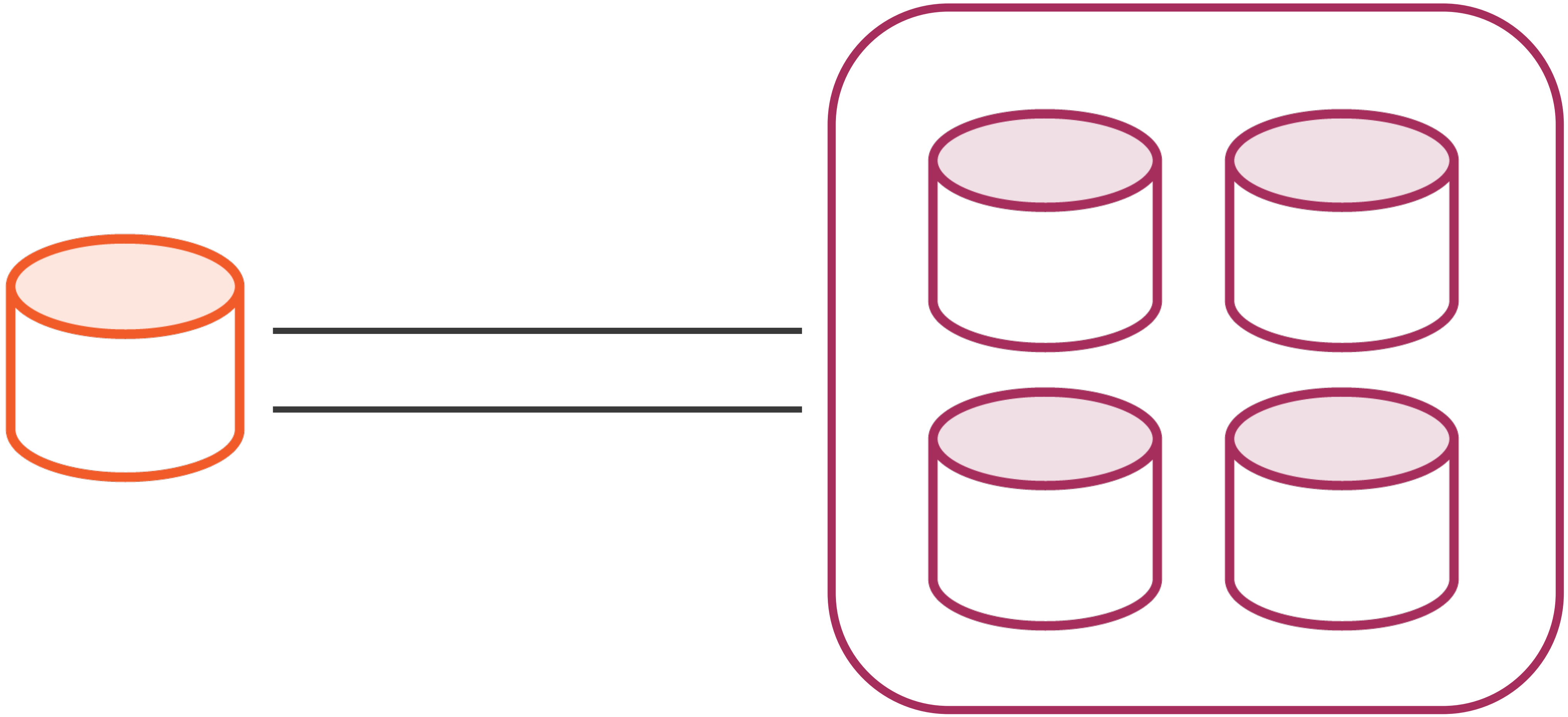
DML methods return rows affected

SQL API transaction support coming soon

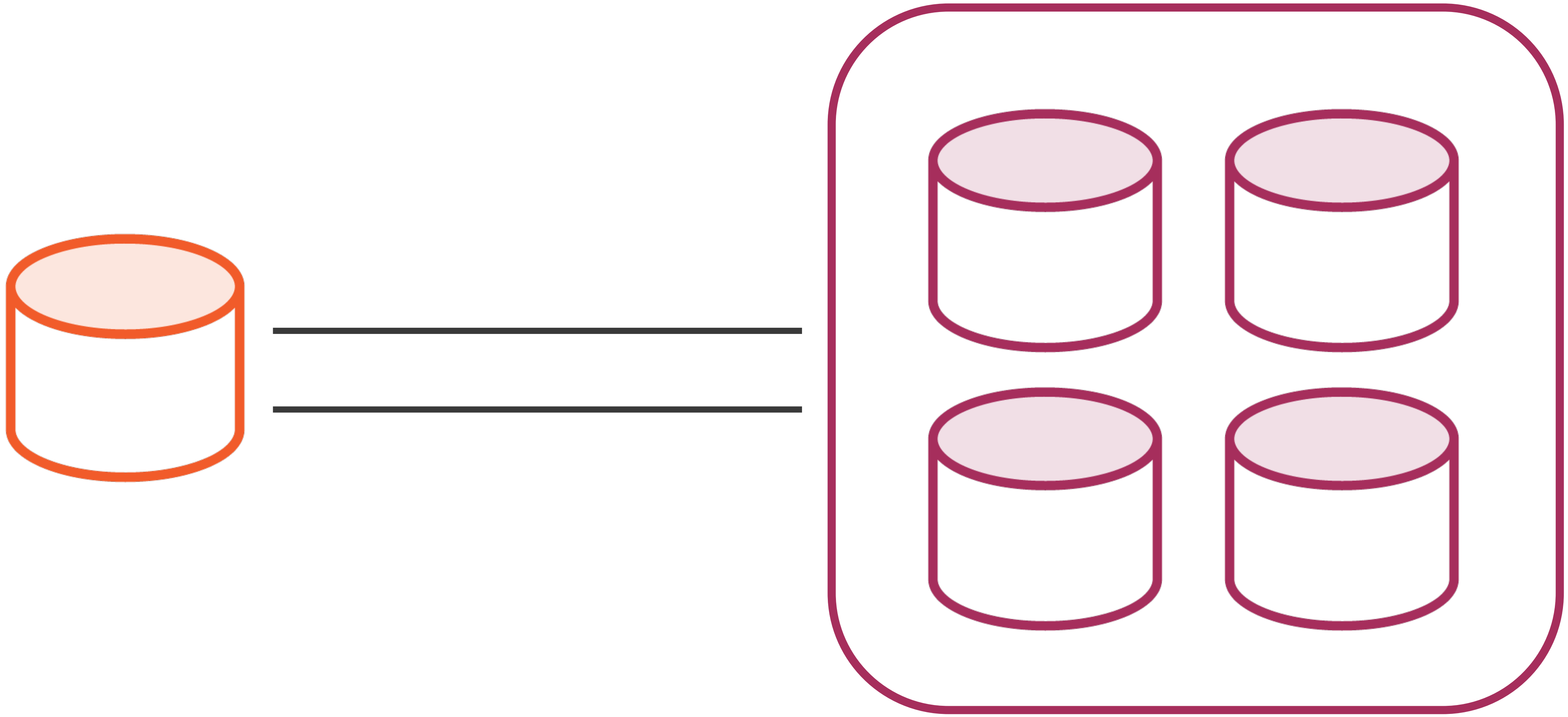
Writing Data to a Cache

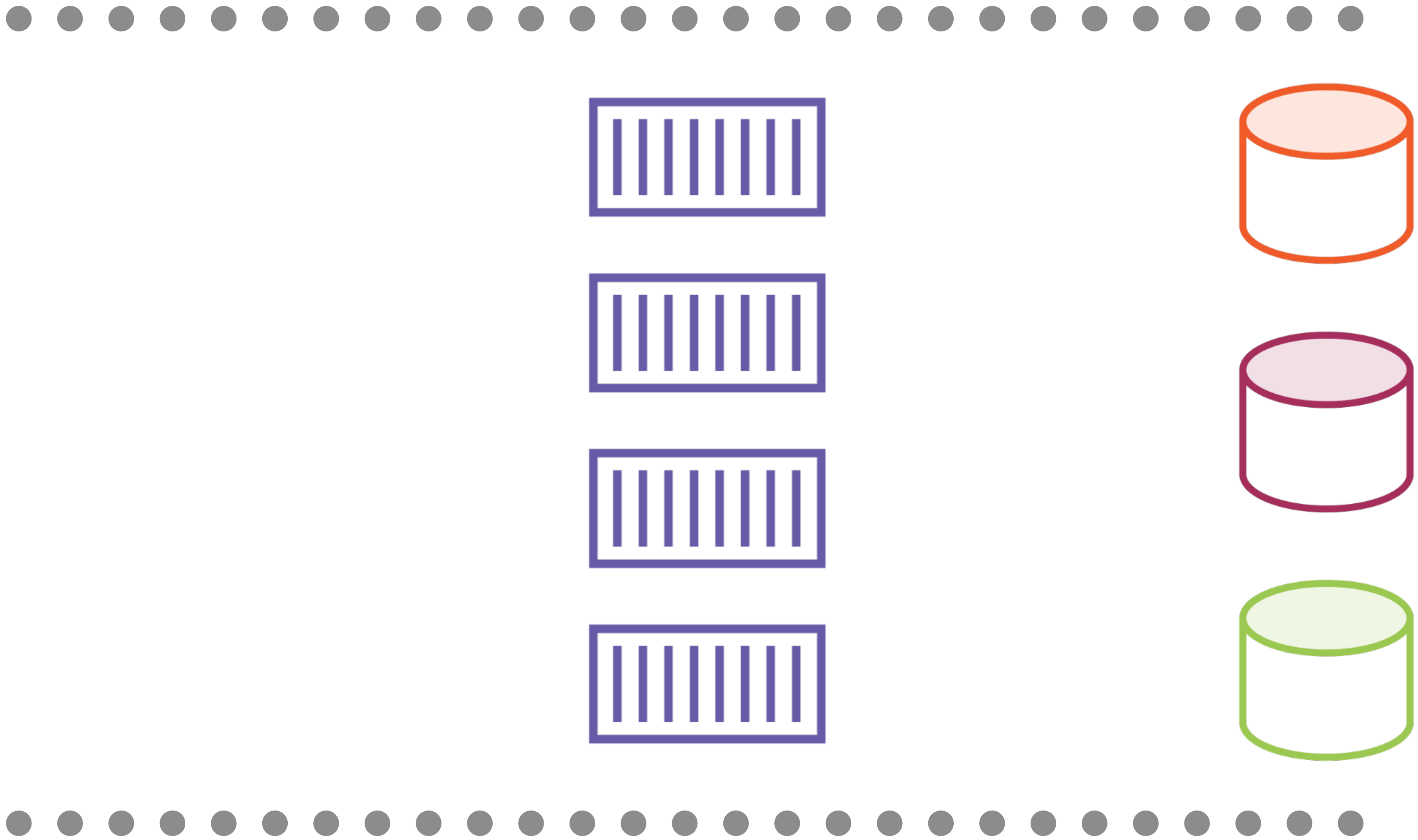


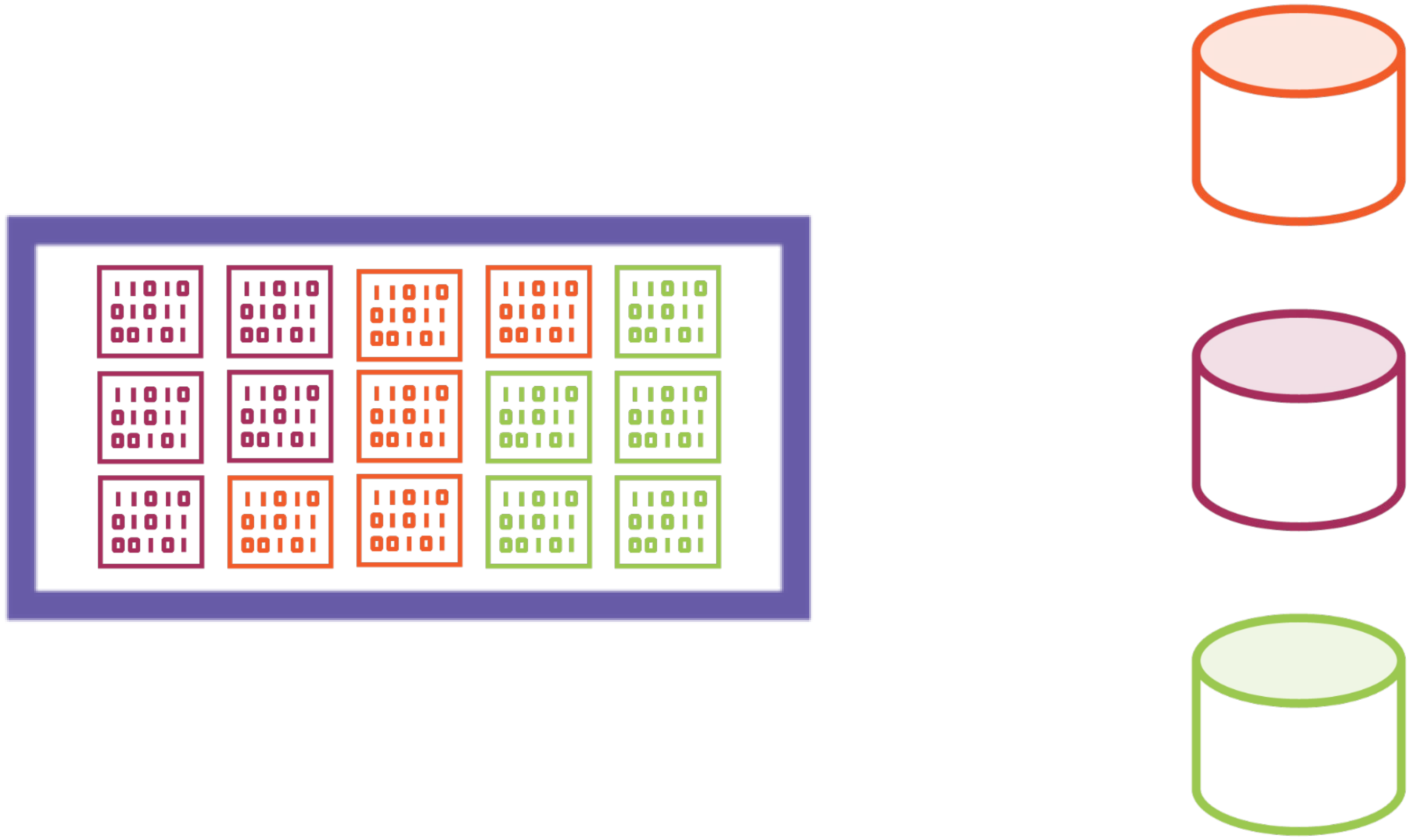
Writing Data to a Cache

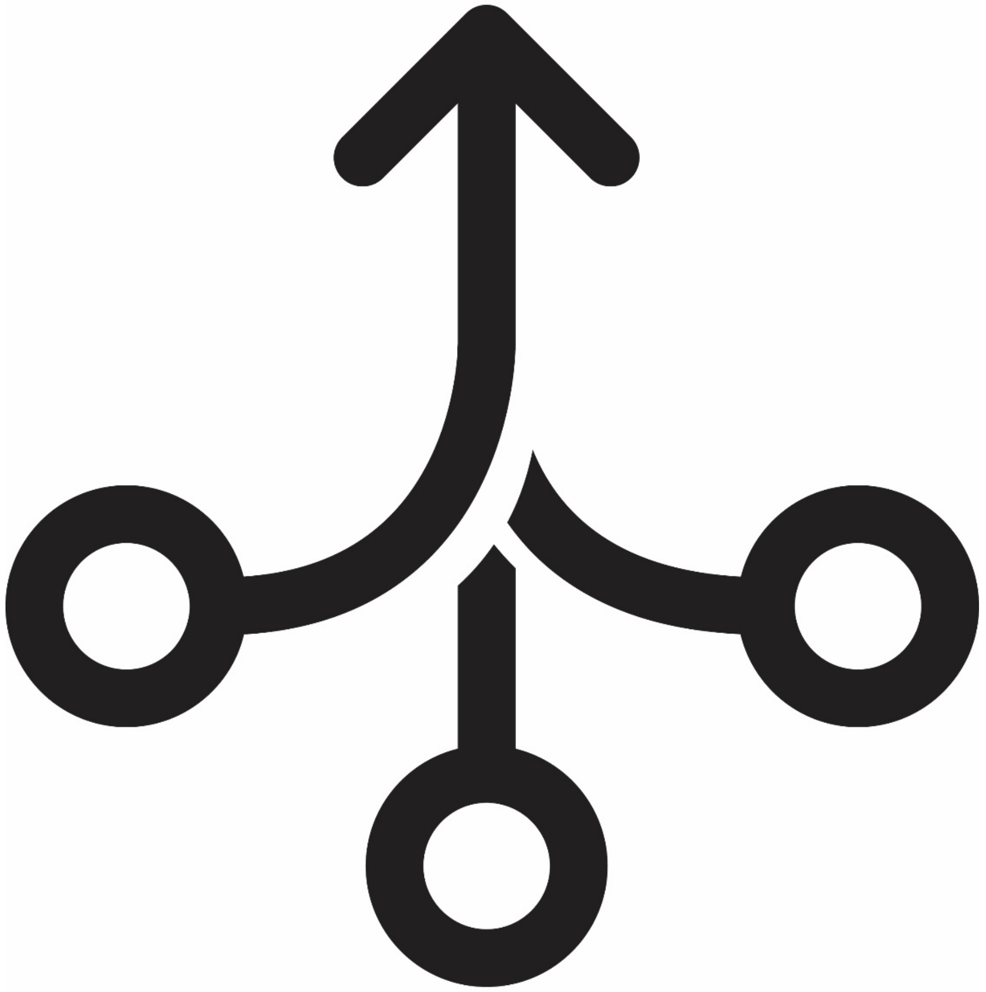


Writing Data to a Cache









Kafka	Camel
JMS	MQTT
Storm	Flink
Twitter	Flume
ZeroMQ	RocketMQ

StreamMultipleTupleExtractor<T,K,V>

T

Incoming message data type

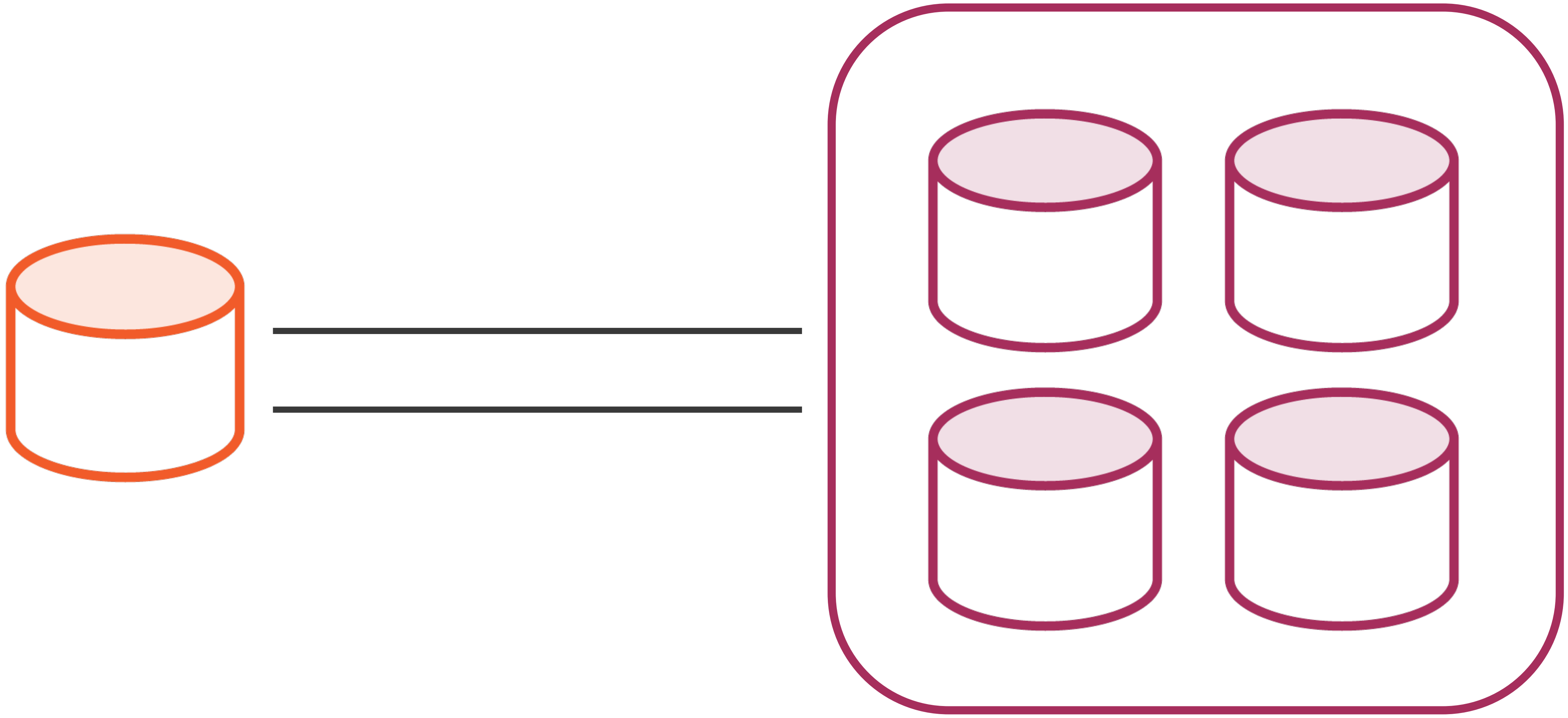
K

Key data type

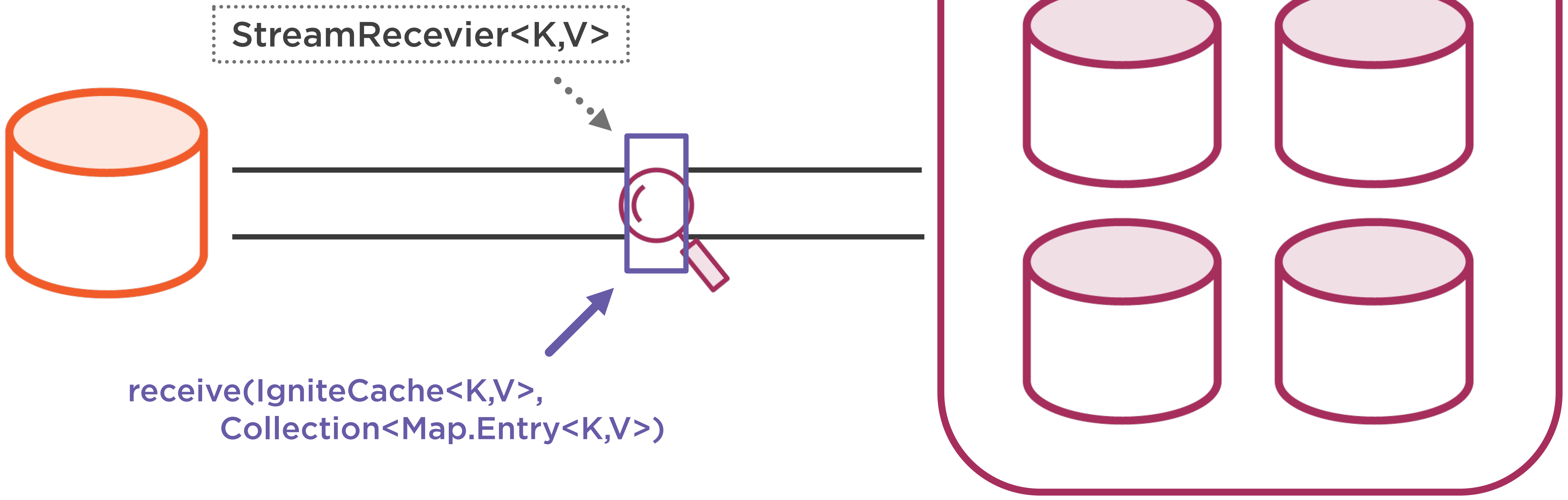
V

Value data type

Stream Receiver



Stream Receiver



Implementations of the Stream Receiver

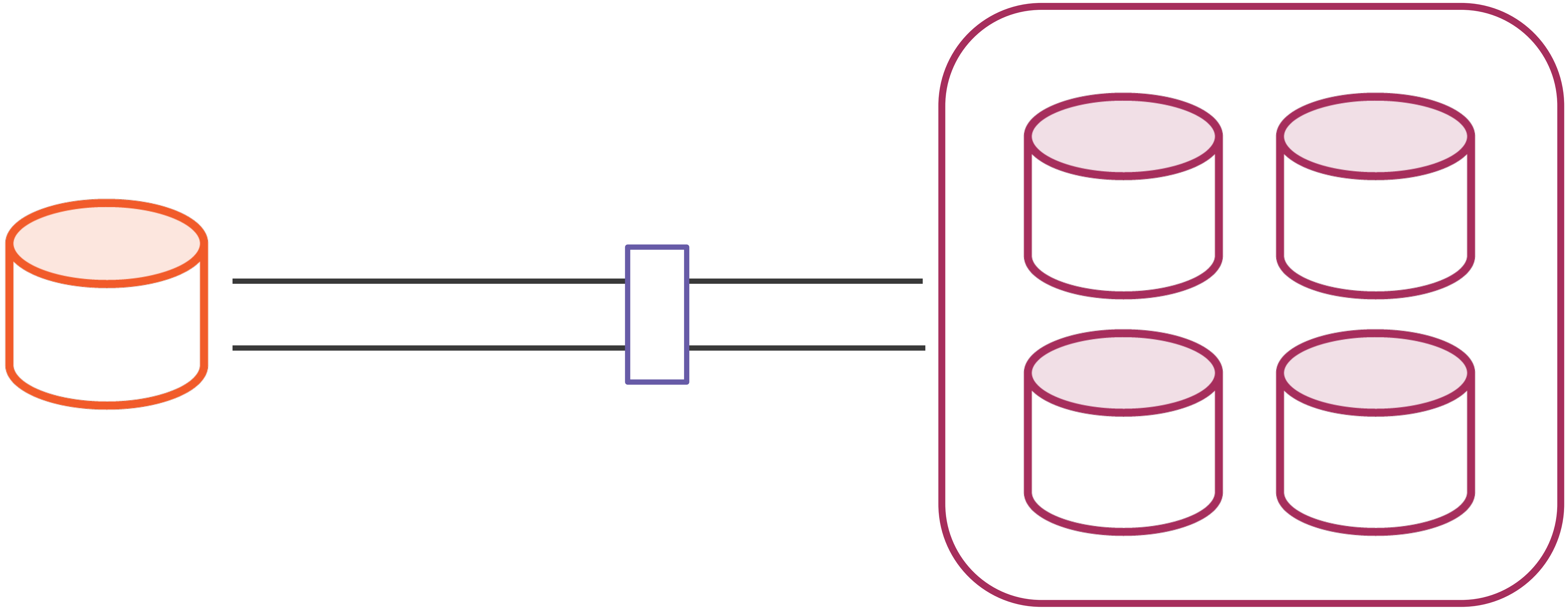
Stream Transformer

Stream Visitor

Stream Transformer

Convenience adapter to update existing values in streaming cache based on the previously cached value.

Stream Transformer



Stream Transformer

Transformer vs. Visitor

Stream Transformer	Stream Visitor
Entries with matches in the cache	All entries in the stream
Provide an entry processor	Takes “StreamReceiver” cache & entries
Automatically writes to cache	Manually write to cache

Flight number

Name of passenger

Assigned seat

Frequent flyer number

Frequent flyer status

Summary

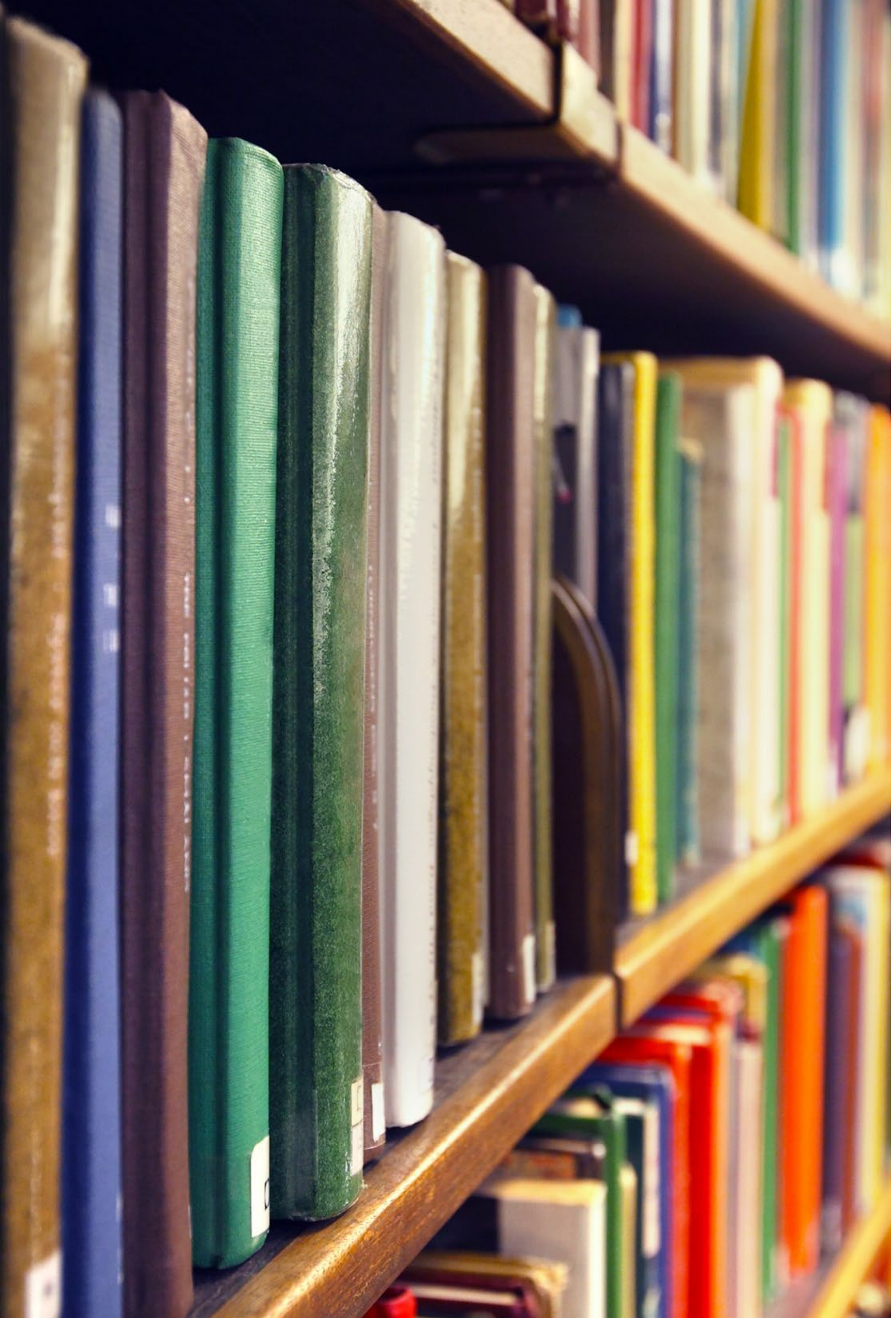
Affinity

- Function
- Data Collocation

The SQL Grid

Continuous queries

Entry processors



Apache Ignite Websites:

https://ignite.apache.org/

http://apacheignite-sql.readme.io/docs

Support:

https://stackoverflow.com/questions/tagged/ignite