

Cassandra - Intro

Big Data Systems



Dr. Rubi Boim

Cassandra

- (Most popular) NoSQL wide column database
- Open source

Motivation (for this course)

- Main database selected for the course
- Hands on practice
- Platform to learn wide column data modeling

Agenda

- History
- Architecture
- Data model (Original)
- Data model (CQL)
- Examples

Agenda

- **History**
- Architecture
- Data model (Original)
- Data model (CQL)
- Examples

History

- Create by Facebook in 2008
paper: Cassandra - A Decentralized Structured Storage System
- By one of the authors of Dynamo
Avinash Lakshman was previously at Amazon, then moved to FB

Facebook's motivation

- Design a system to fulfill the storage needs of the **inbox search problem**
 - Billions of writes per day
 - Scale (much more) with the number of users
- A year later Cassandra was used to power much more services within Facebook

Interesting fact

- Facebook released Cassandra as Open source
- Later on Facebook abandoned Cassandra and moved to different technologies
HBase to power the inbox search
- However Cassandra today is used by endless number of companies
Apple, Twitter, Netflix, Uber and much more

Initial requirements (2008)

- Access / manage petabytes of data in real time
- Wide applicability
- Highly scalable
- Highly available
- **Completely distributed (P2P)**
- **Tunable consistency**
- **Fast**

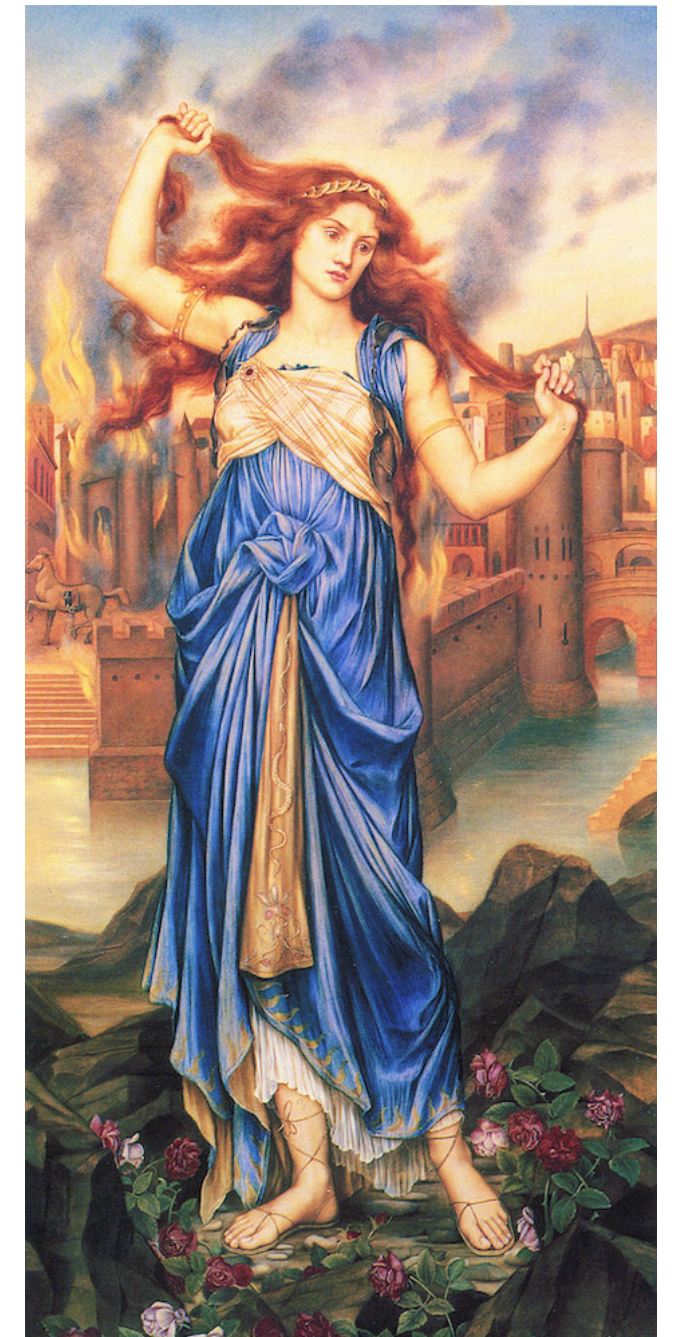
Agenda

- History
- **Architecture**
- Data model (Original)
- Data model (CQL)
- Examples

Architecture

Cassandra: daughter of Dynamo and Bigtable

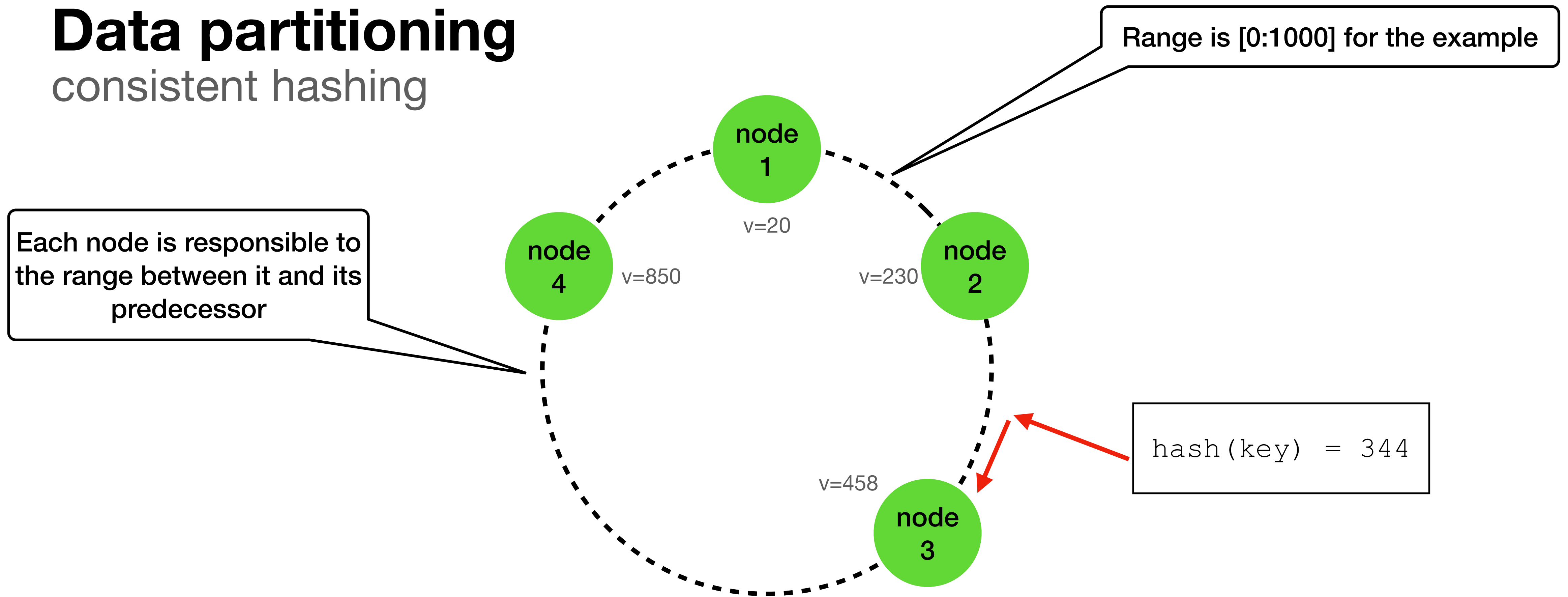
- **Dynamo**
Distributed storage, replication
- **Bigtable**
Data model, storage engine



Cassandra was designed as a best-in-class combination of both systems

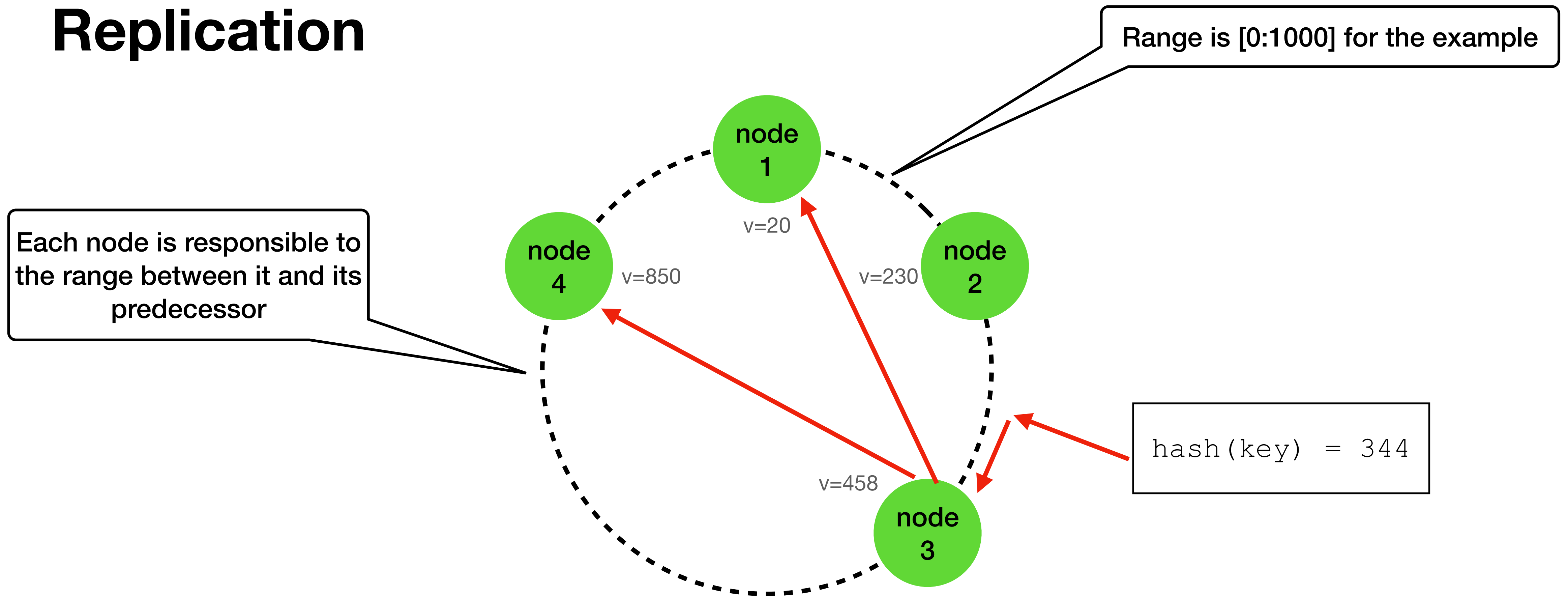
Dynamo → Cassandra (1)

Data partitioning consistent hashing



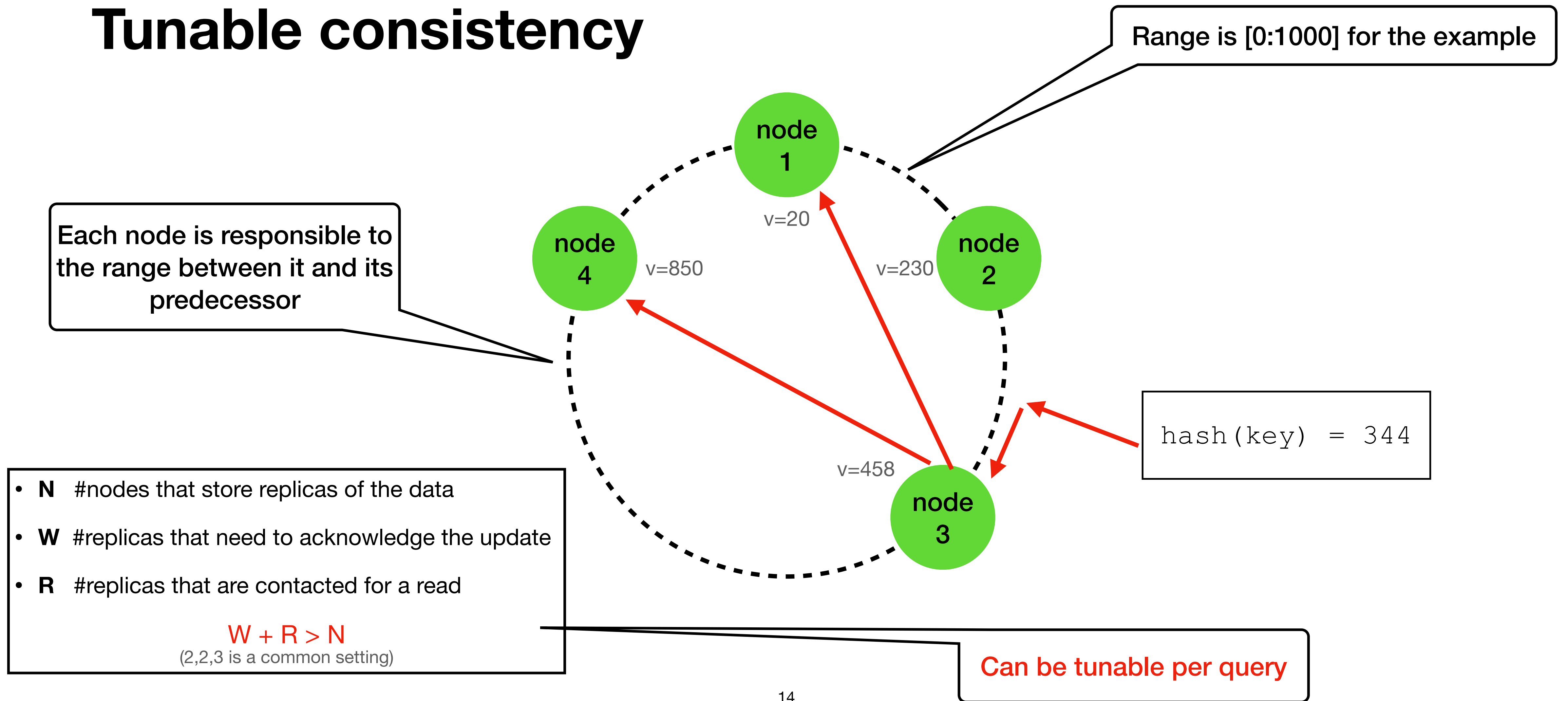
Dynamo → Cassandra (2)

Replication



Dynamo → Cassandra (3)

Tunable consistency

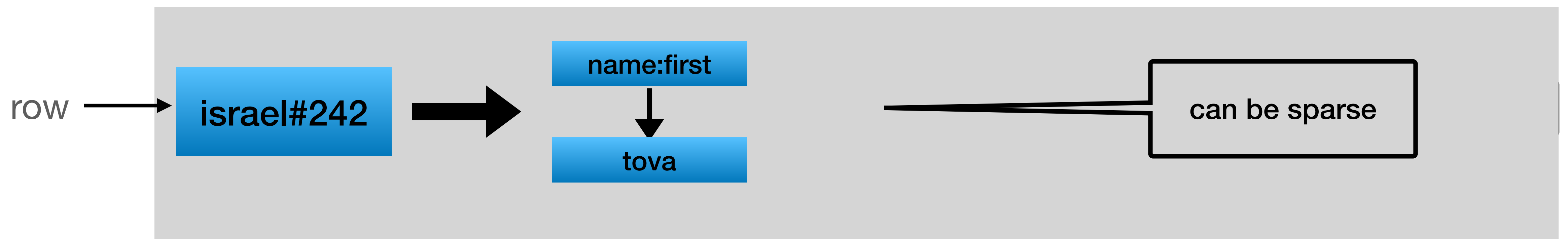
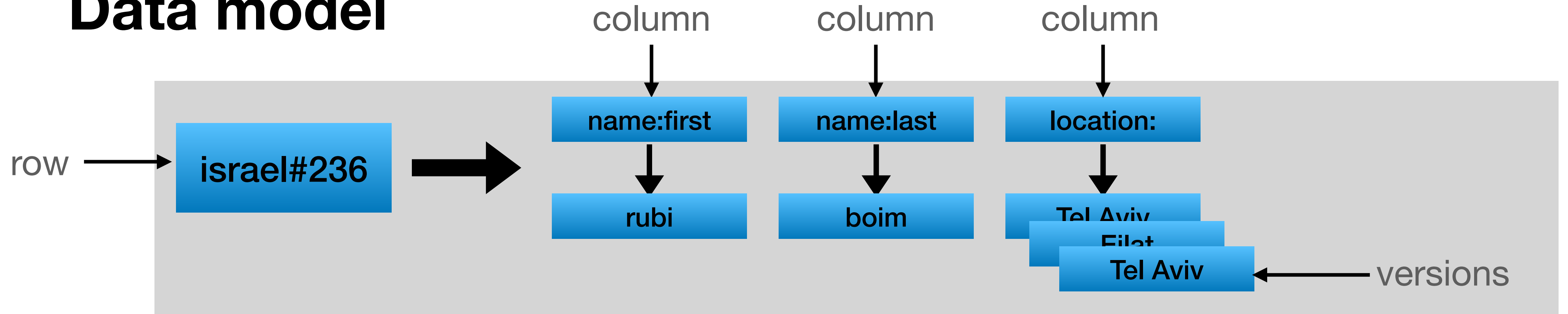


Dynamo → Cassandra (4)

- Distributed cluster membership
fully distributed P2P, masterless
- Failure detection
gossip
- Incremental scale-out on commodity hardware
different types of hardware size

Bigtable —> Cassandra (1)

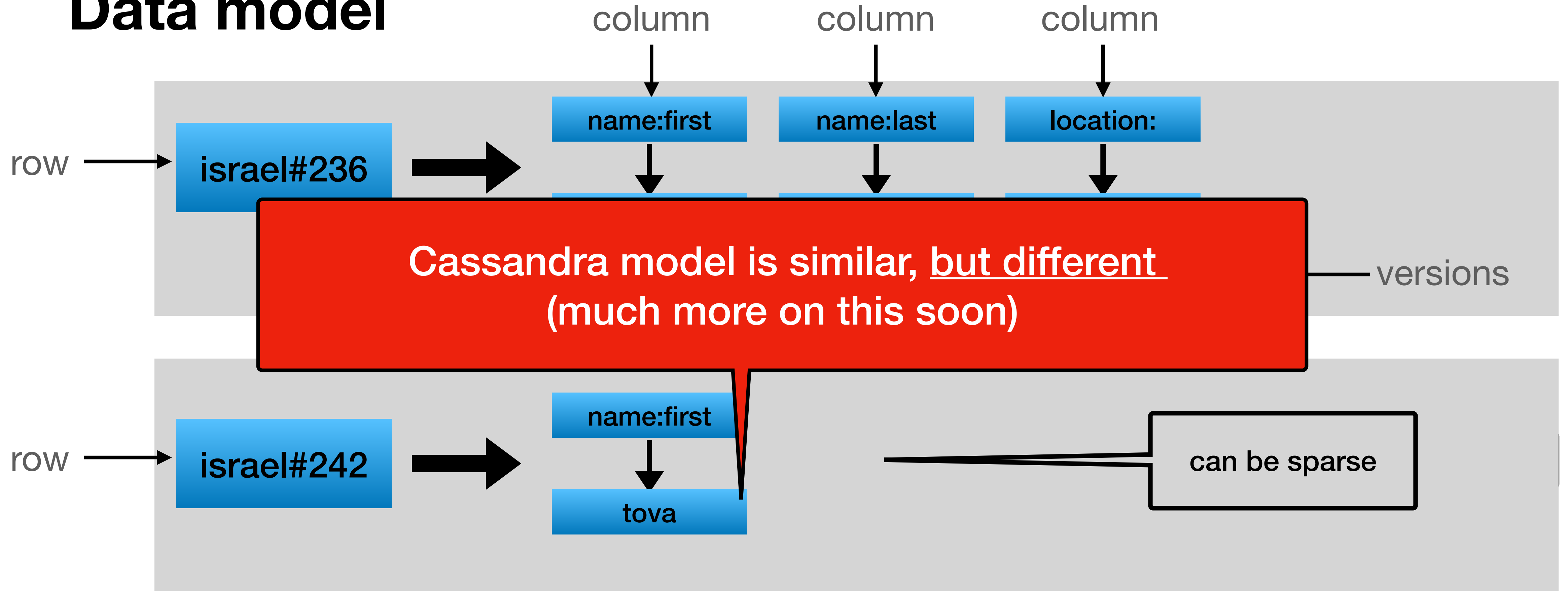
Data model



<row:string, column:string, timestamp:int64> —> string

Bigtable —> Cassandra (1)

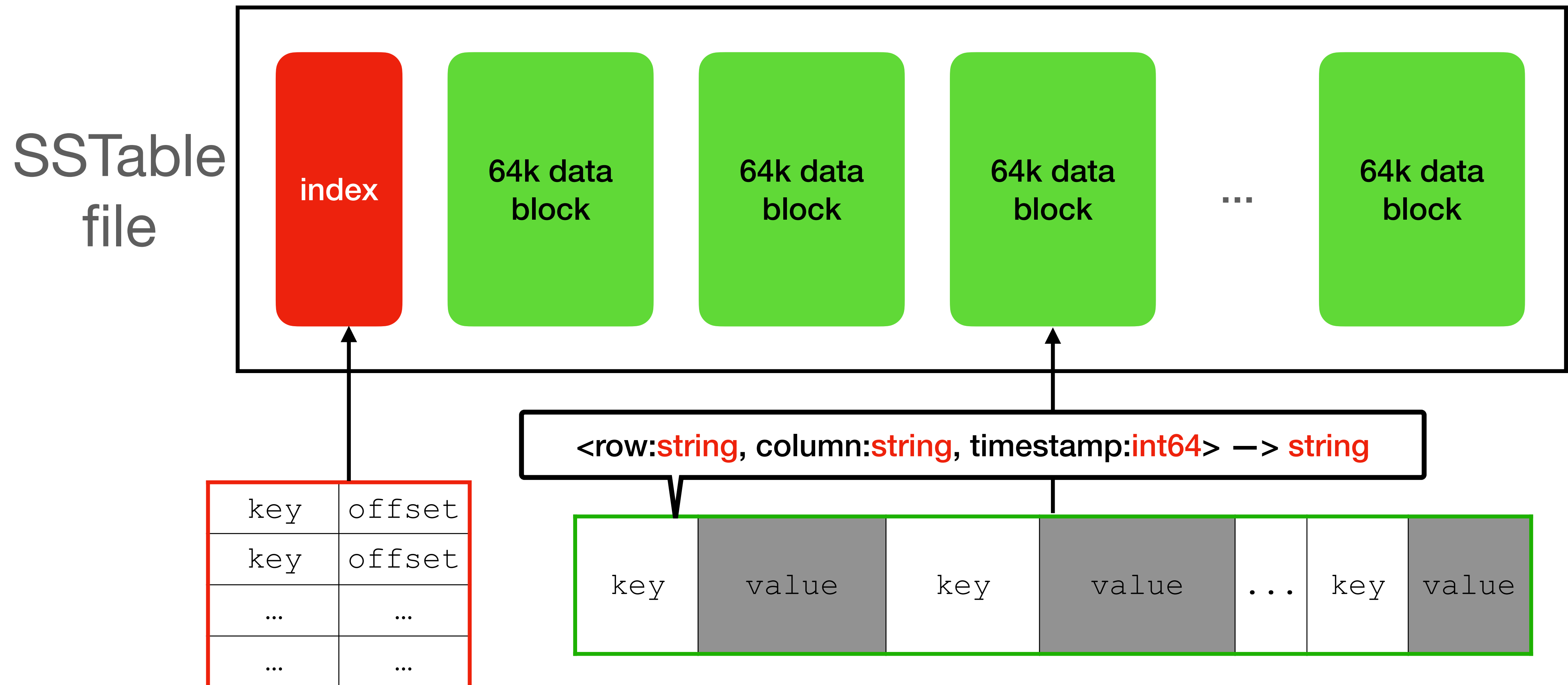
Data model



<row:string, column:string, timestamp:int64> —> string

Bigtable → Cassandra (2)

Storage engine - SSTables



Bigtable —> Cassandra (2)

Storage

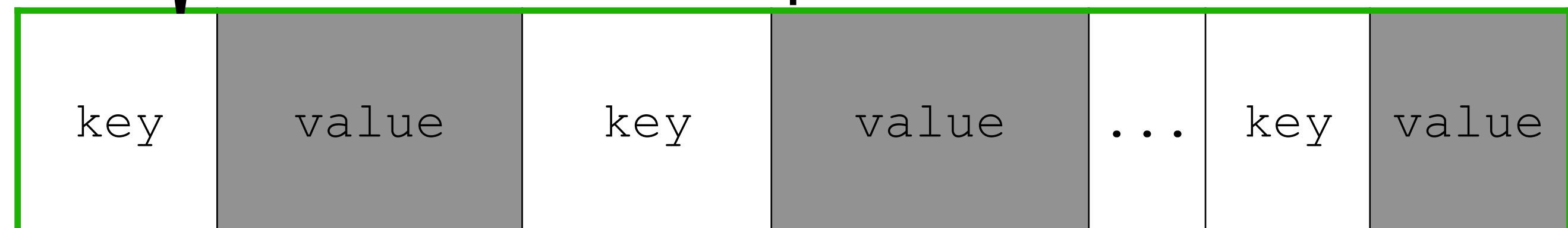
Writes are super fast because they do NOT require a read.

SSTable
file

Even If only a single column is added/deleted

`<row:string | column:string, timestamp:int64> -> string`

key	offset
key	offset
...	...
...	...



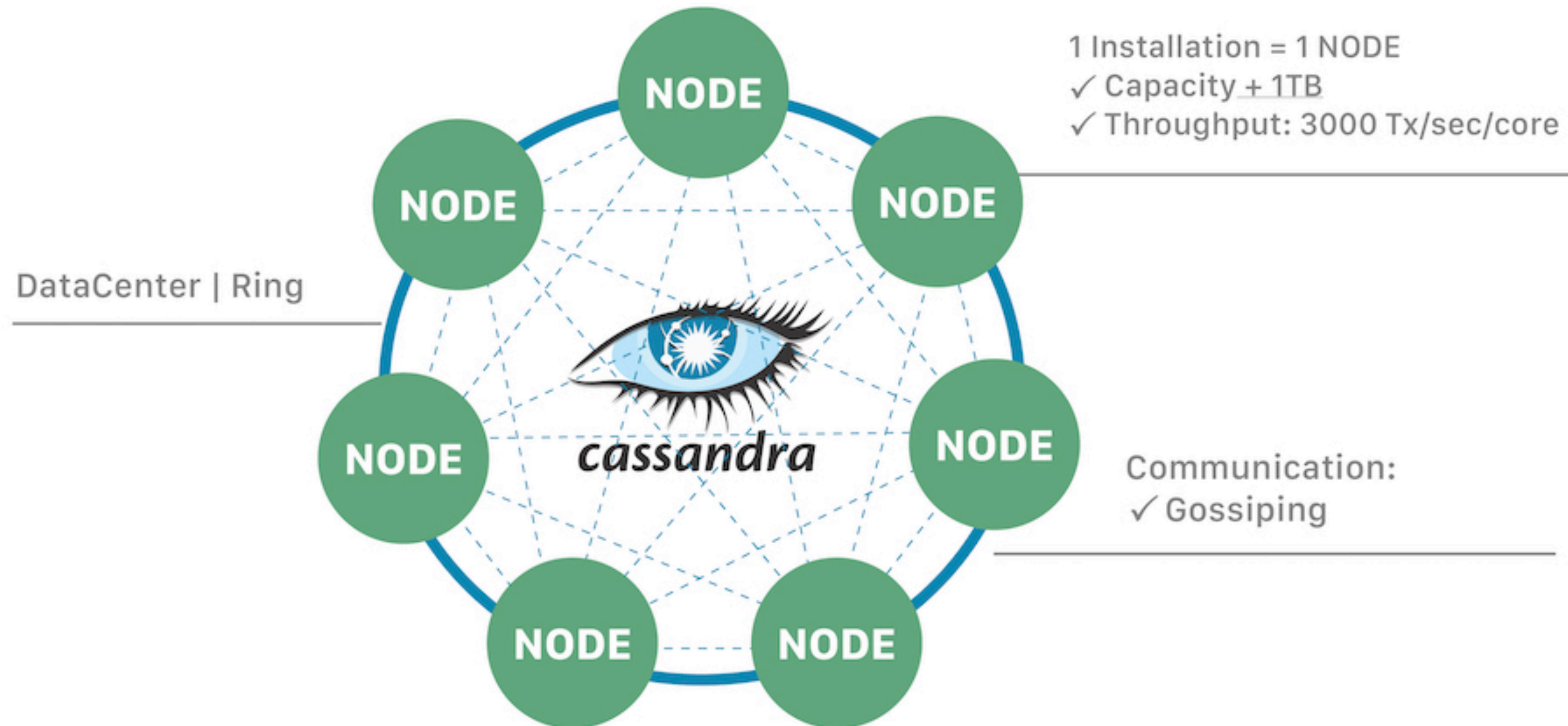
Bigtable —> Cassandra (3)

Storage engine - the rest of the gang

- CommitLog
- Memtable
- SSTable (and compactions)
- Bloom filters

Cassandra ring

ApacheCassandra™ = NoSQL Distributed Database



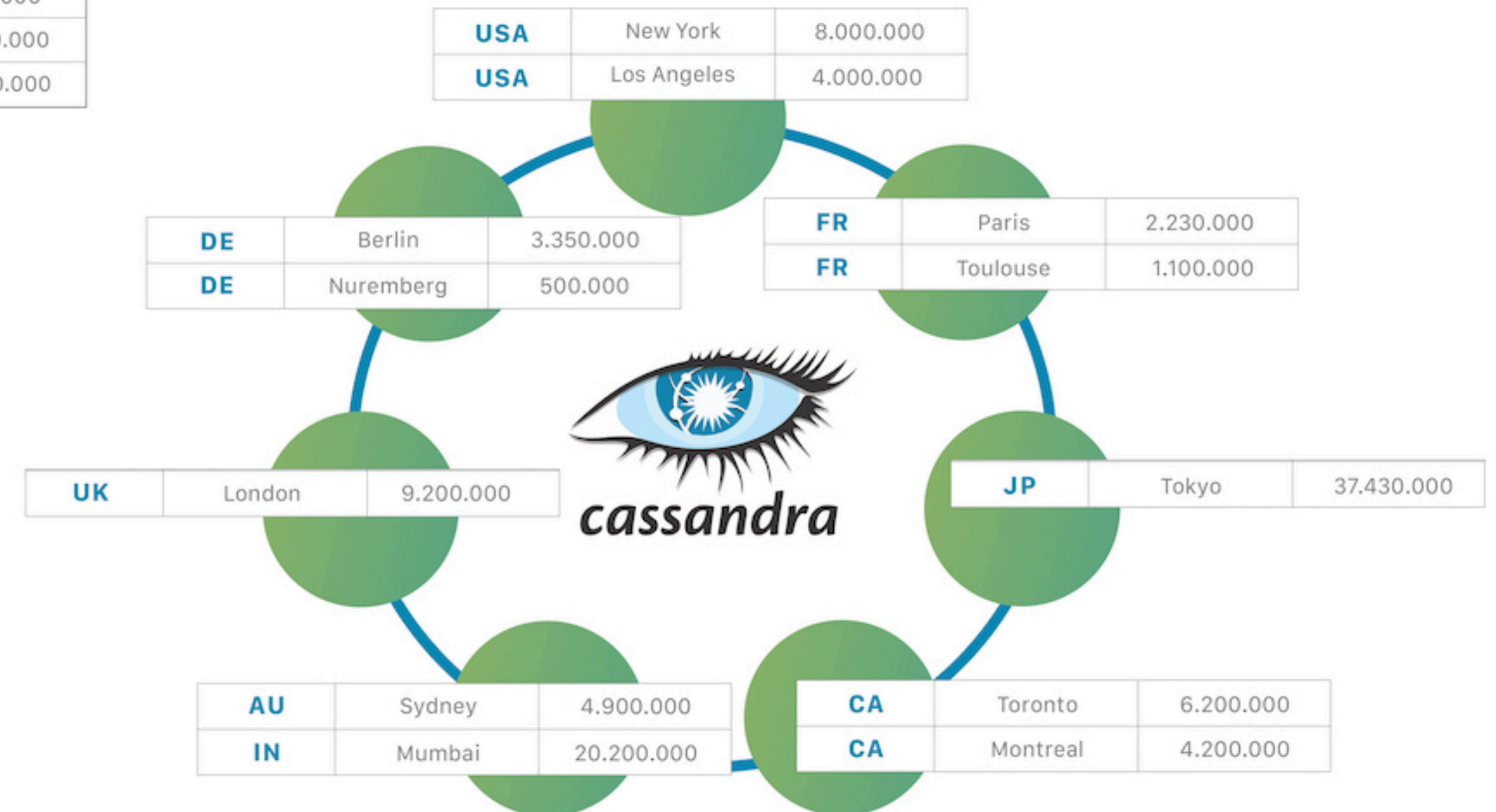
Cassandra - partitions



COUNTRY	CITY	POPULATION
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

Partitions are similar to ROWs in Bigtable
(In a few slides...)



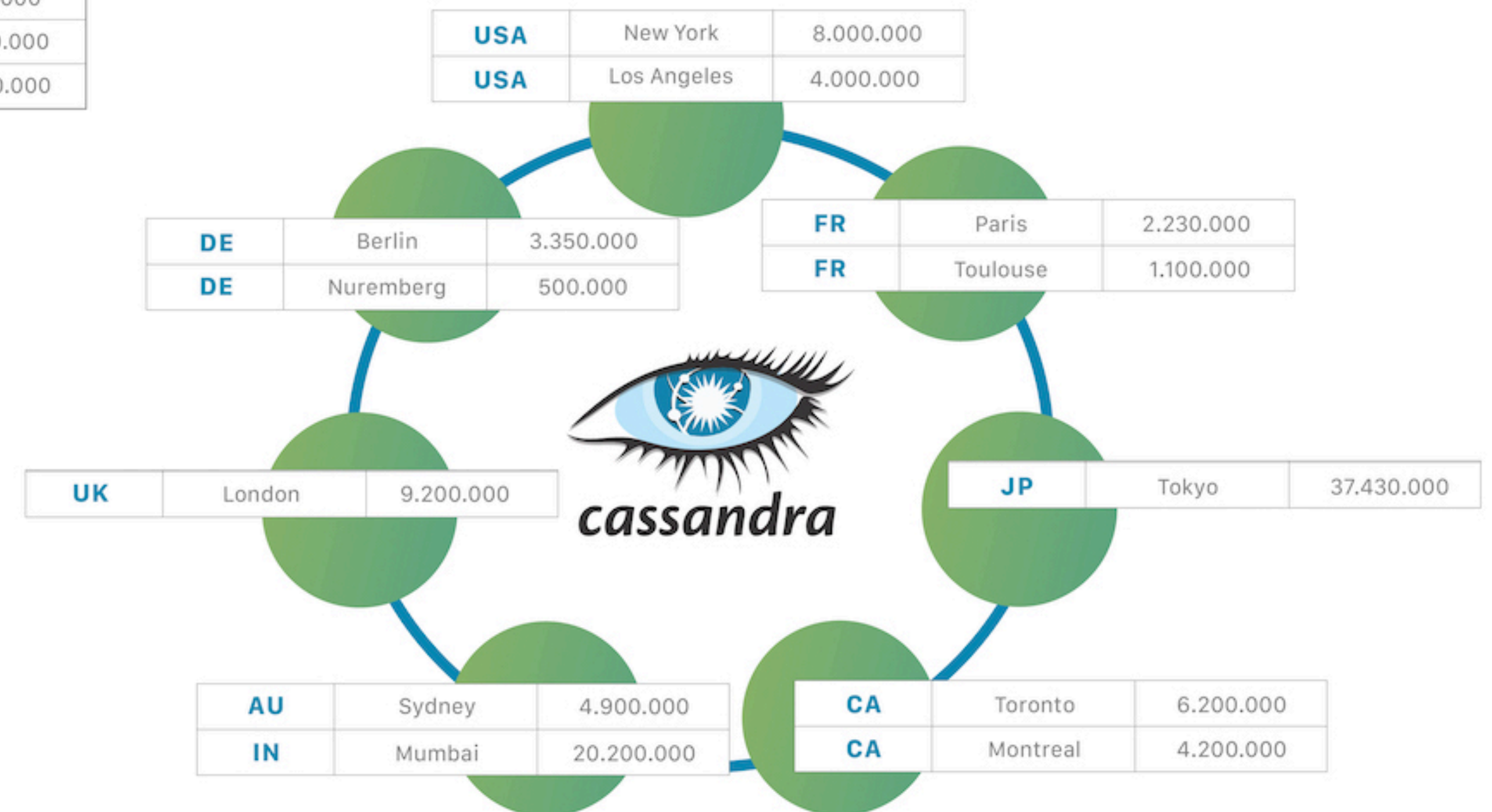
Cassandra - partitions



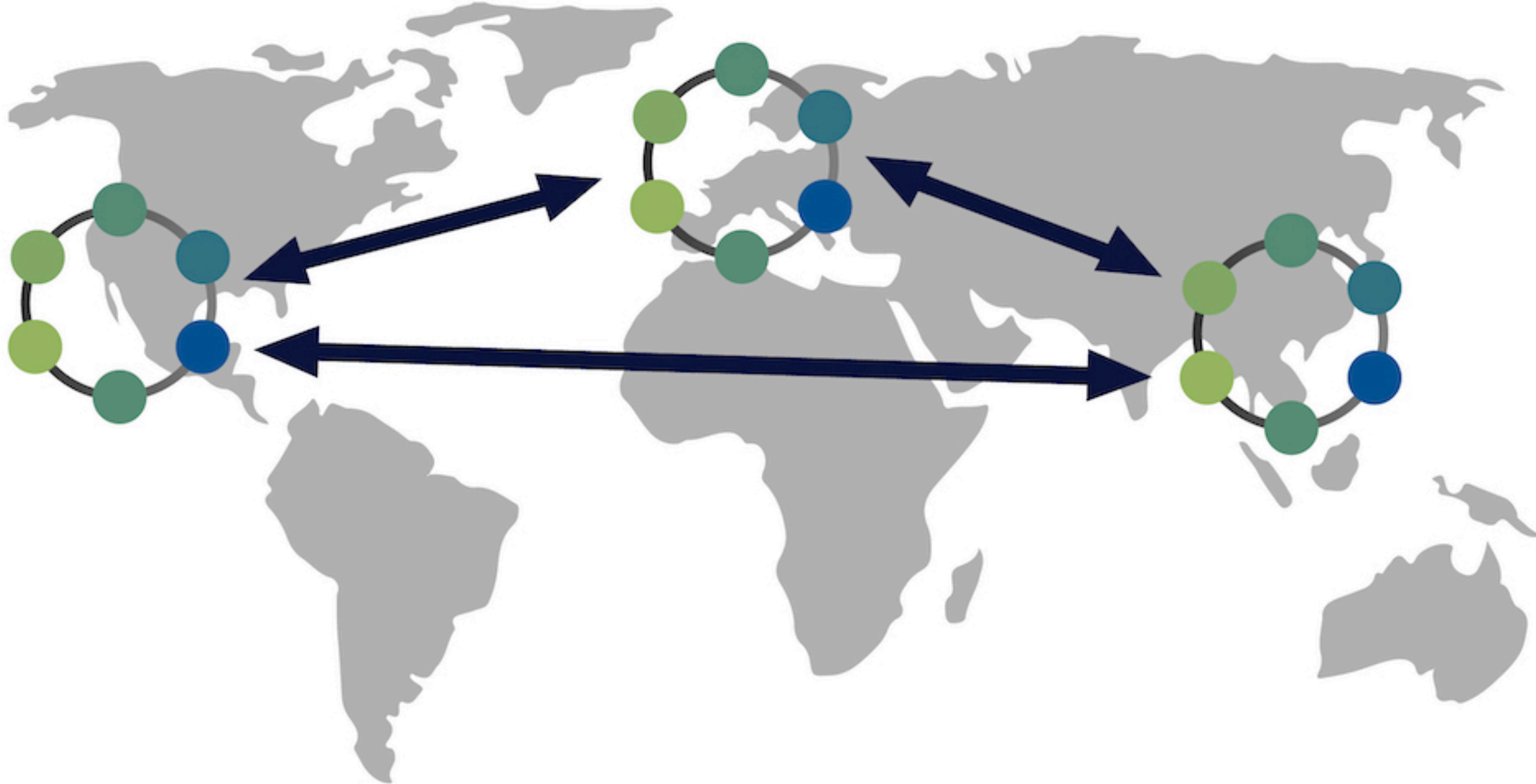
COUNTRY	CITY	POPULATION
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key

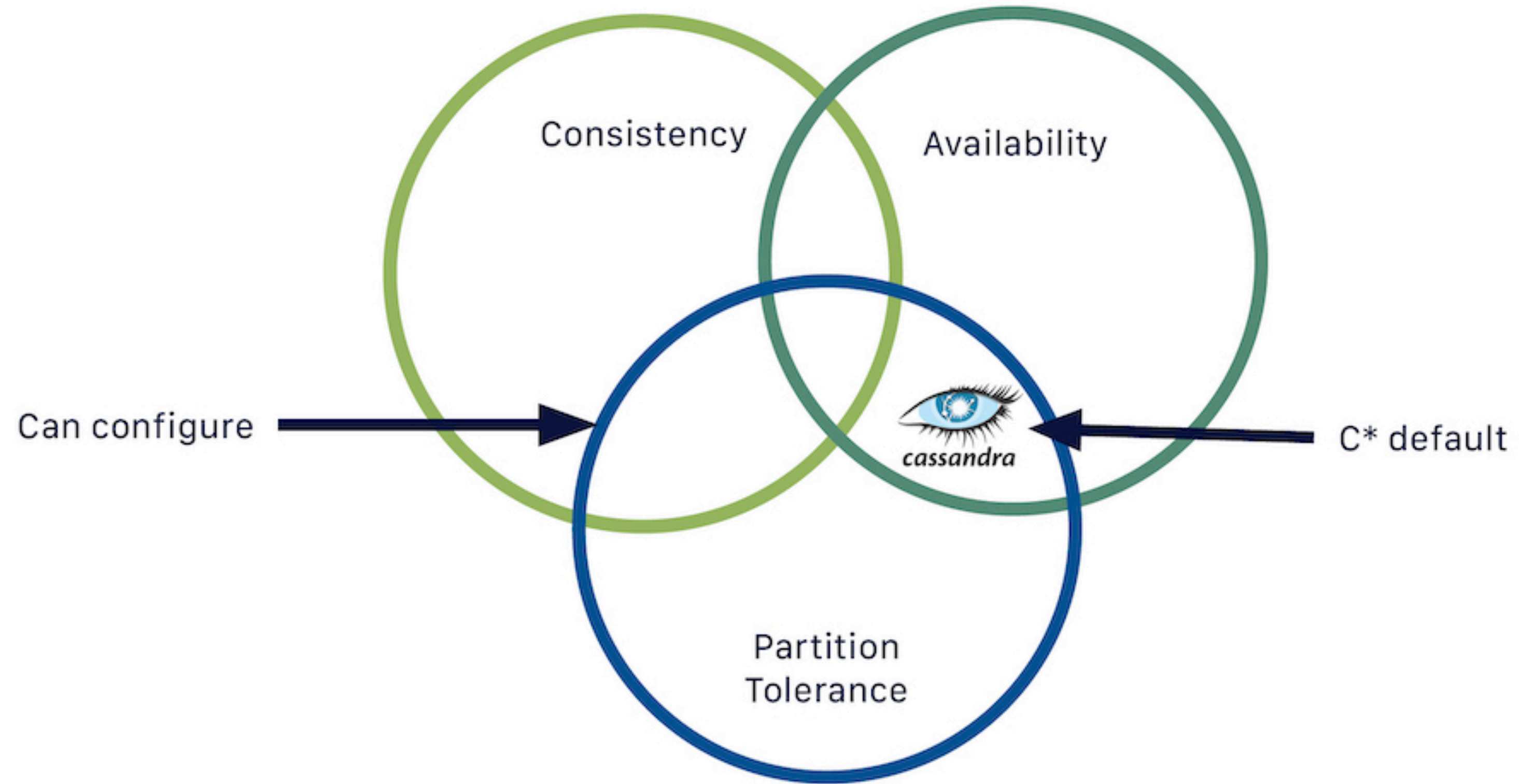
But Bigtable is a wide column DB -
How did we get to “relational” table?
(In a few slides...)



Cassandra - globally distributed

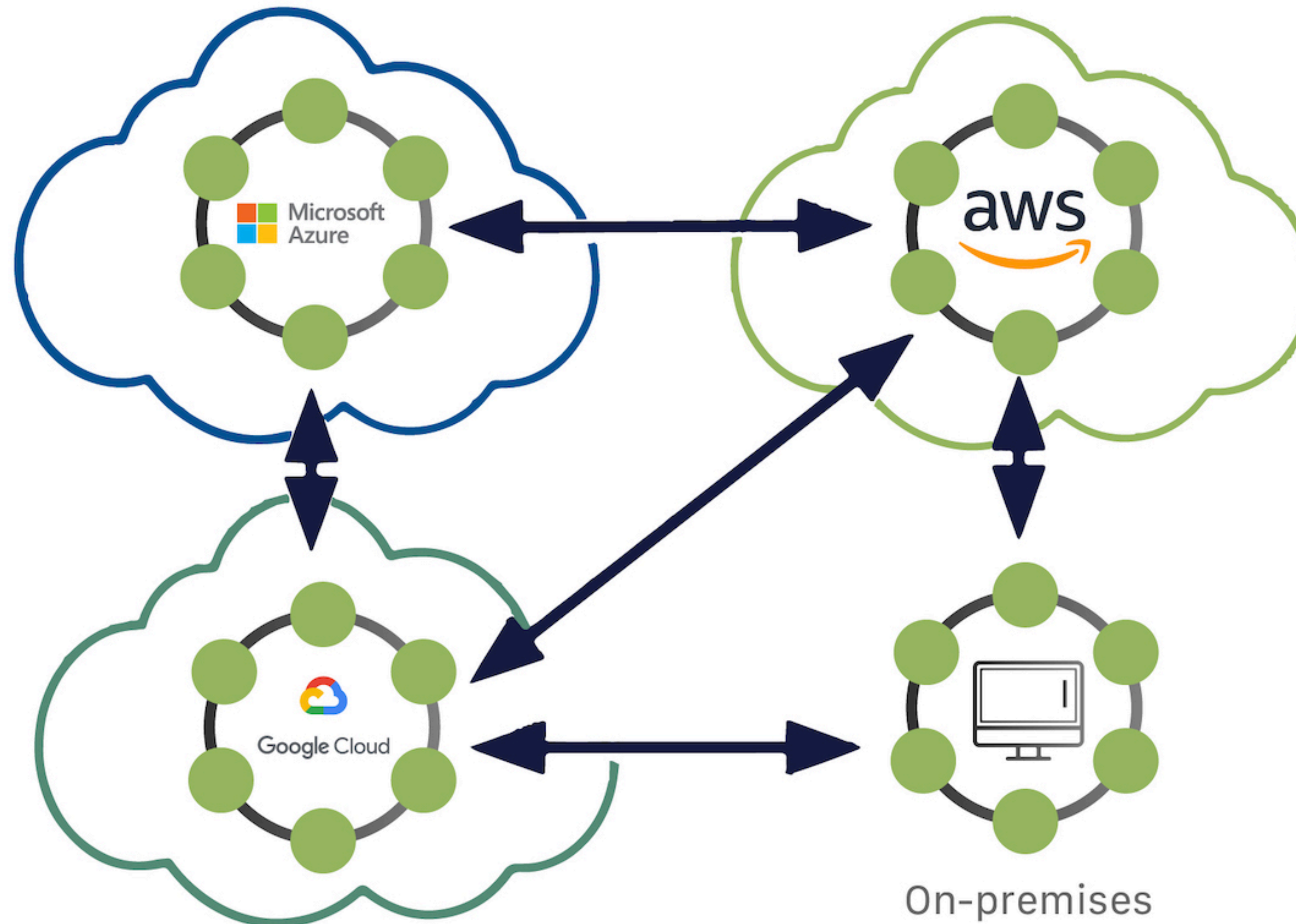


Cassandra - tunable consistency



Cassandra - flexibility

Open source



Agenda

- History
- Architecture
- **Data model (Original)**
- Data model (CQL)
- Examples

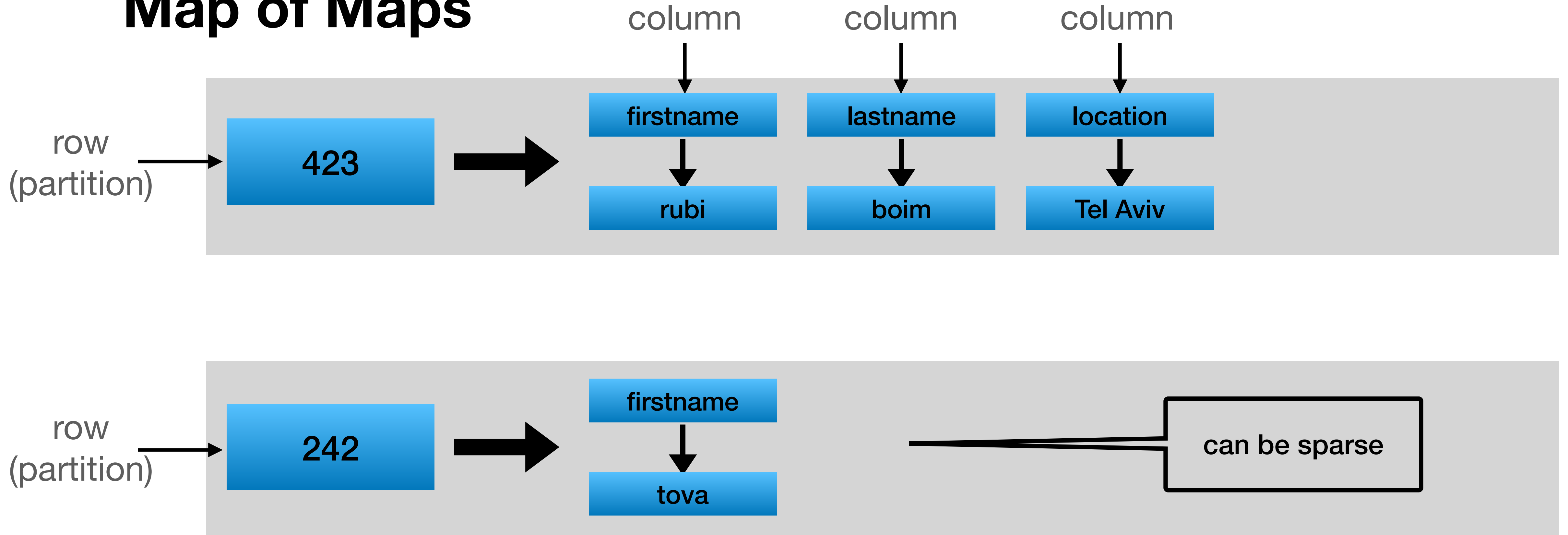
Data model (original)

Map of Maps

- Wide column
- Similar to Bigtable
 - Without the versions
 - Rows are NOT ordered

Data model (original)

Map of Maps



`<row:string, column:string> -> string`

Data model (original)

API

- Thrift API
 - Standard / Open source
- Straightforward API
“put / get / delete”

Data model (original)

Schema less

- When you create a table, it has no columns
 - Originally a table was called “column family”
this will change in a few slides
- Each row can add / remove columns
- Data is saved as raw bytes



Note: a “column family” in Bigtable is something else

Data model (original)

S

Think about it, if you look at the schema of the database
you only see something like:

```
table users  
table movies
```

...

No info (id, name, title, rating...) is available at the schema level.

You would need to “explore” the data to understand it

- Data is saved as raw bytes

Data model (original)

Schema less - main problem

- Hard to understand the logic
 - Need to read code and explore data
- Data errors (no strict forcing for reading/writing data)
for example, you save `long` values but read `int`

Data model - Update

After a few years a MAJOR update to Cassandra

- The storage had not changed at all
- Same wide column
- But the data modeling / API is completely different

“New” data model - CQL

Main goal

- Add “schema” to tables (“similar” to relational DB)
- Use an API “similar” to SQL

video_id	title	year	duration
1	Bad Boys	1995	119
2	Top Gun	1986	110
3	American Pie	1999	96

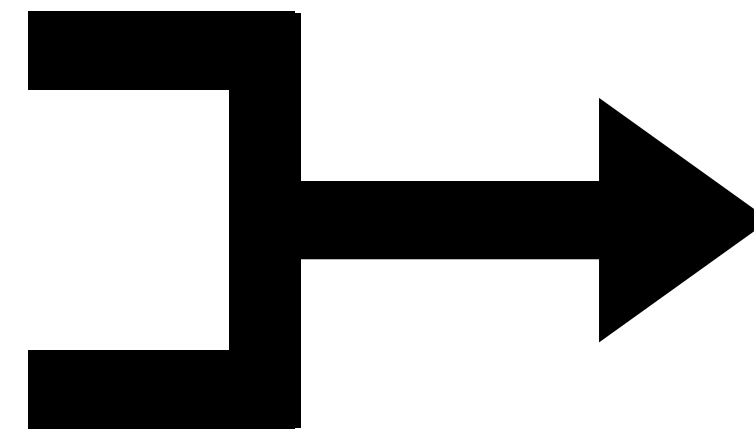
- Use the same storage/model behind the scenes

Agenda

- History
- Architecture
- Data model (Original)
- **Data model (CQL)**
- Examples

Data model (CQL) - TLDR

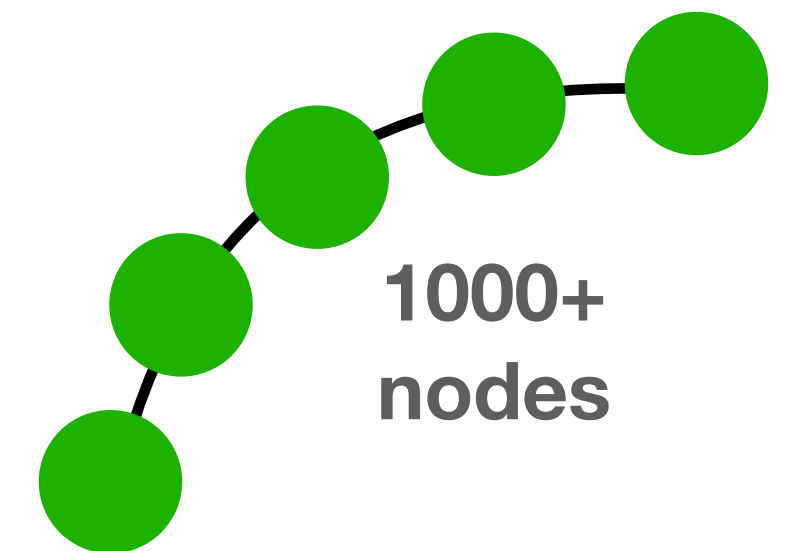
- A table in Cassandra has
 - Partition keys
 - Clustering columns
 - Columns



Primary key

Partition Key

- Partition key: the first part of the primary key
- `hash(partition key)` defines the partition
- Dynamo will map between partitions and nodes
partition range: $[-2^{63}$ to $+2^{63}]$



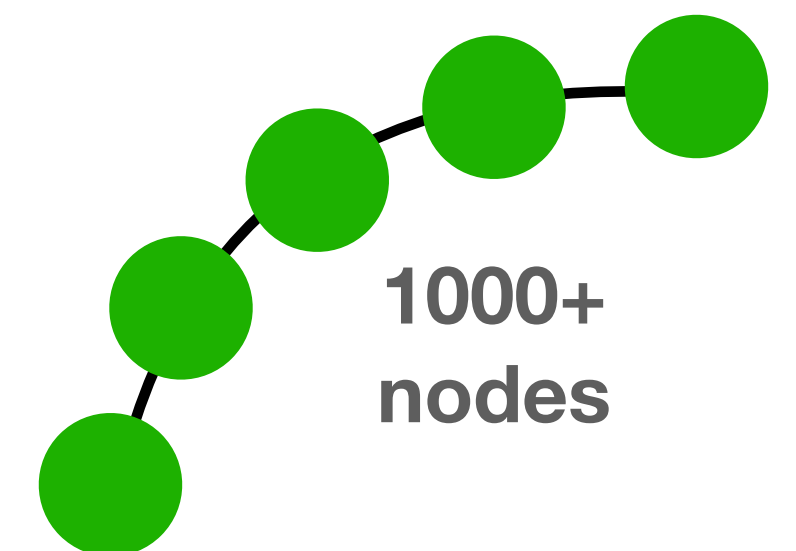
Partition Key

- Partition key: the first part of the primary key

- hash (p

A partition is similar to a ROW in Bigtable

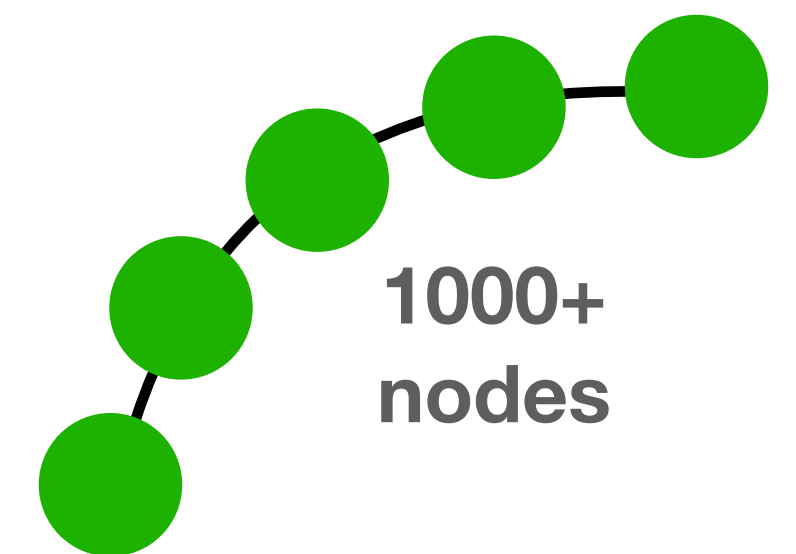
- Dynamo will map between partitions and nodes
partition range: $[-2^{63}$ to $+2^{63}]$



Partition Key

- Partition key: the first part of the primary key
- `hash(partition key)` defines the partition
- Dynamo will map between partitions and nodes
partition range: $[-2^{63}$ to $+2^{63}]$

Throughout the rest of the course, whenever you would see the “1000 nodes ring” image think how the slide’s information would be applied on large scale



Classic (relational) view

```
CREATE TABLE movies (  
  video_id    BIGINT,  
  title       TEXT,  
  year        INT,  
  duration    INT,  
  PRIMARY KEY (video_id)  
);
```

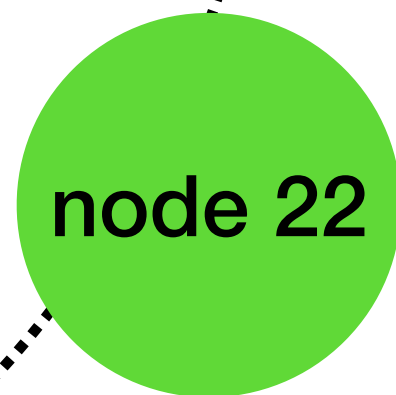
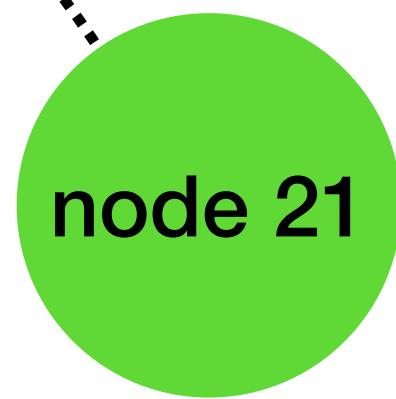
movies	
video_id	K
title	
year	
duration	

What is the partition key of Top Gun?

movies			
video_id	title	year	duration
1	Bad Boys	1995	119
2	Top Gun	1986	110
3	American Pie	1999	96

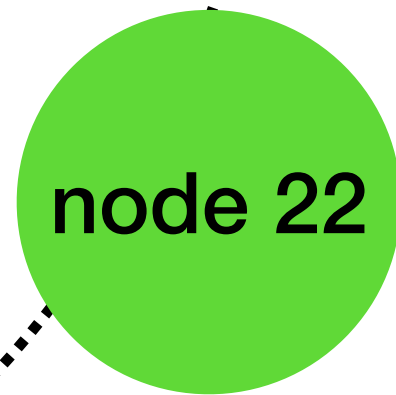
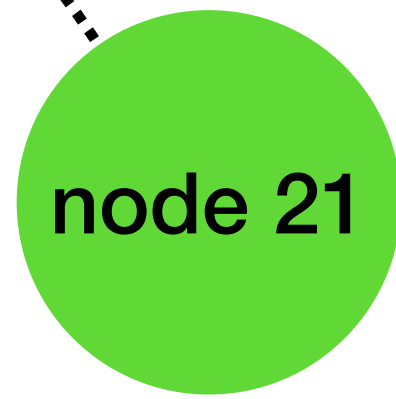
Relational → wide columns

video_id	title	year	duration
1	Bad Boys	1995	119
2	Top Gun	1986	110
3	American Pie	1999	96



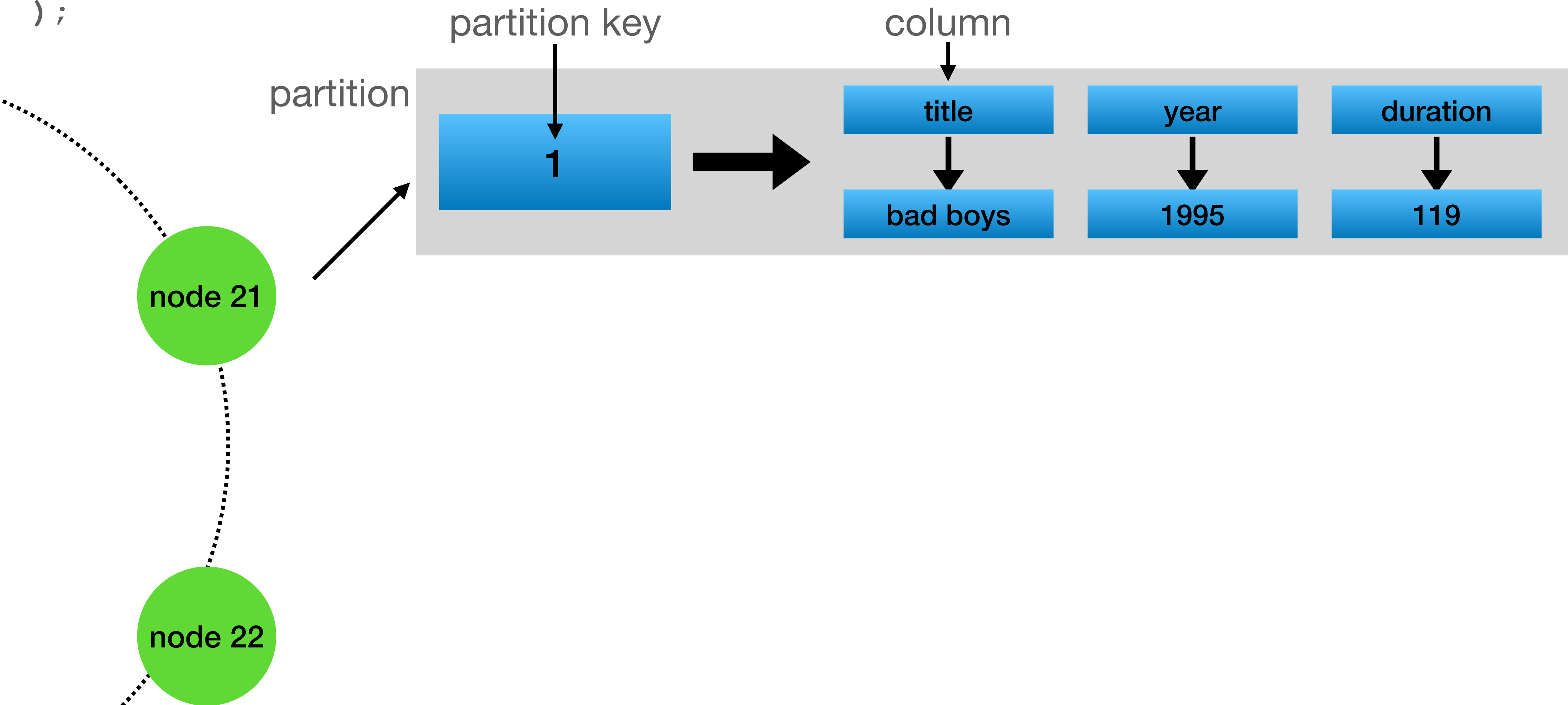
```
CREATE TABLE movies (  
  video_id      BIGINT,  
  title         TEXT  
  year          INT,  
  duration      INT,  
  PRIMARY KEY (video_id)  
);
```

video_id	title	year	duration
1	Bad Boys	1995	119
2	Top Gun	1986	110
3	American Pie	1999	96



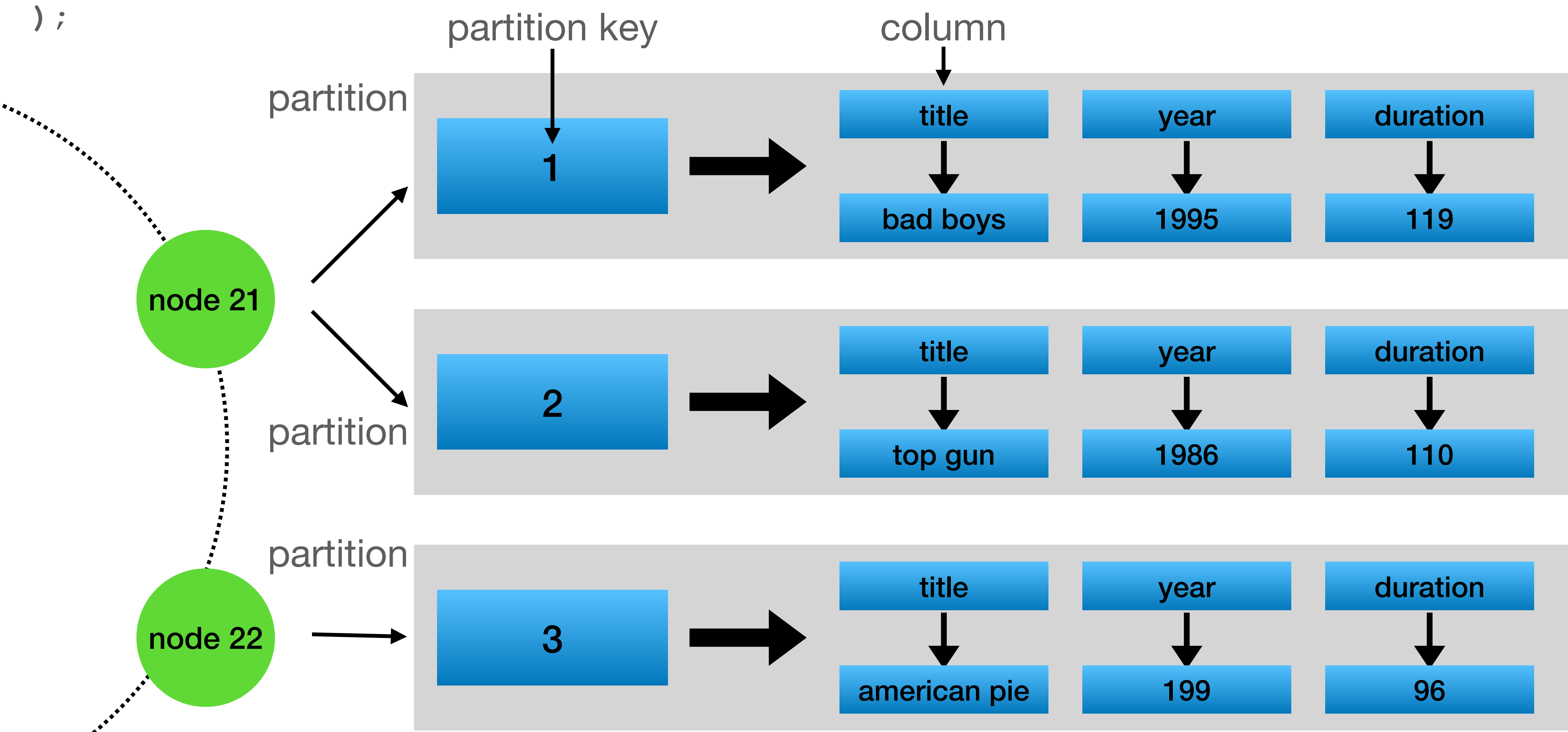
```
CREATE TABLE movies (  
  video_id      BIGINT,  
  title        TEXT  
  year         INT,  
  duration     INT,  
  PRIMARY KEY (video_id)  
);
```

video_id	title	year	duration
1	Bad Boys	1995	119
2	Top Gun	1986	110
3	American Pie	1999	96



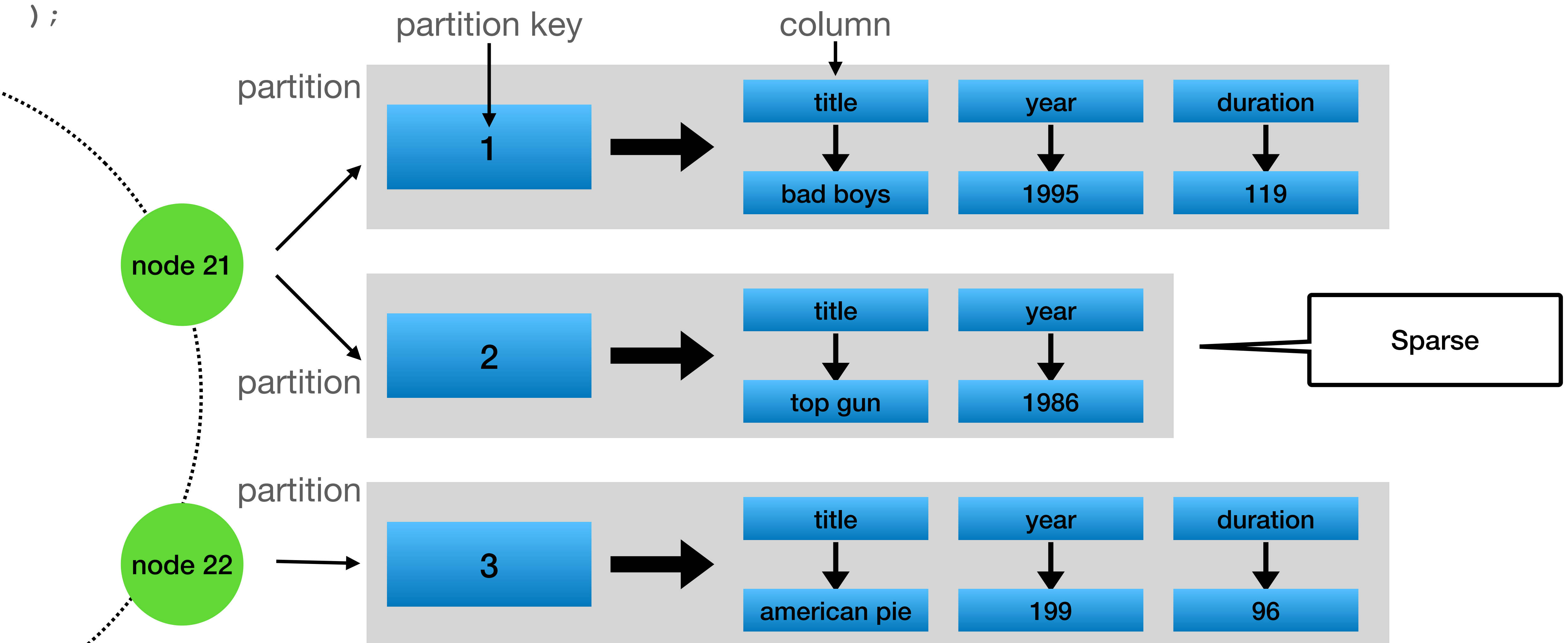
```
CREATE TABLE movies (  
  video_id      BIGINT,  
  title         TEXT  
  year          INT,  
  duration      INT,  
  PRIMARY KEY (video_id)  
);
```

video_id	title	year	duration
1	Bad Boys	1995	119
2	Top Gun	1986	110
3	American Pie	1999	96



```
CREATE TABLE movies (  
  video_id      BIGINT,  
  title         TEXT  
  year          INT,  
  duration      INT,  
  PRIMARY KEY (video_id)  
);
```

video_id	title	year	duration
1	Bad Boys	1995	119
2	Top Gun	1986	
3	American Pie	1999	96

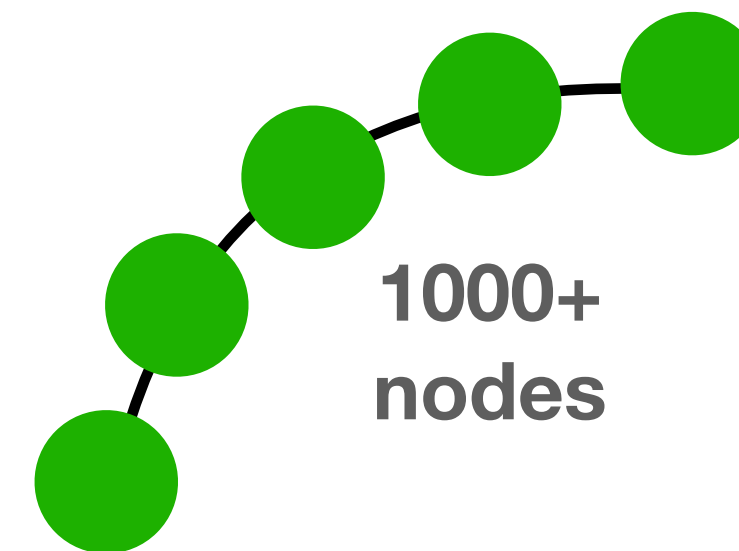


SELECTing data efficiently

- Cassandra needs to know where the data is located
—> **We need to provide the partition key**

```
SELECT * FROM movies WHERE id = 1
```

Partition key



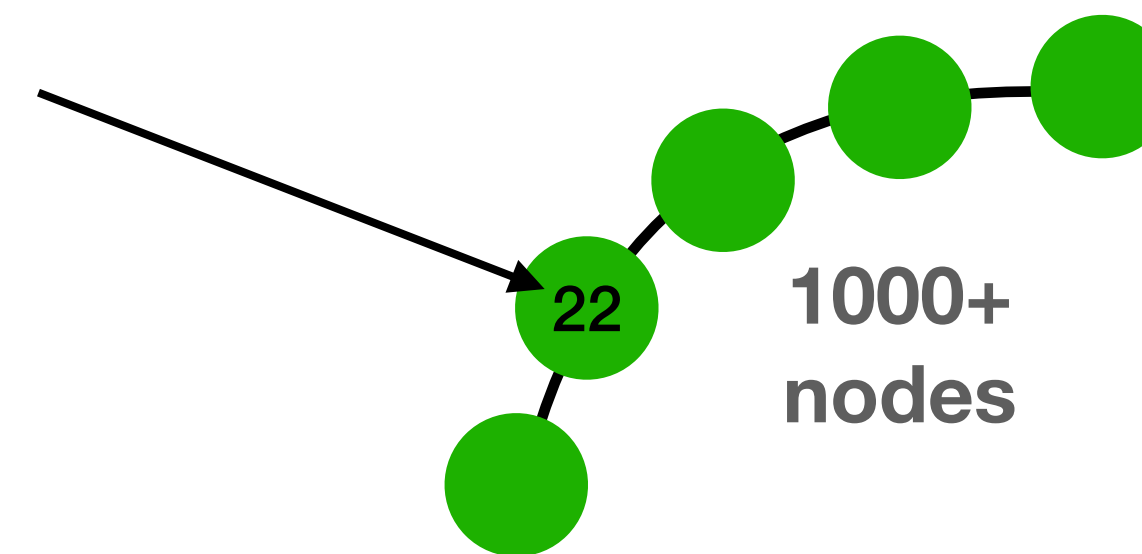
SELECTing data efficiently

- Cassandra needs to know where the data is located
—> **We need to provide the partition key**

```
SELECT * FROM movies WHERE id = 1
```

Partition key

```
partition key = 1  
—>  
token = 12312  
—>  
data is on node 22
```

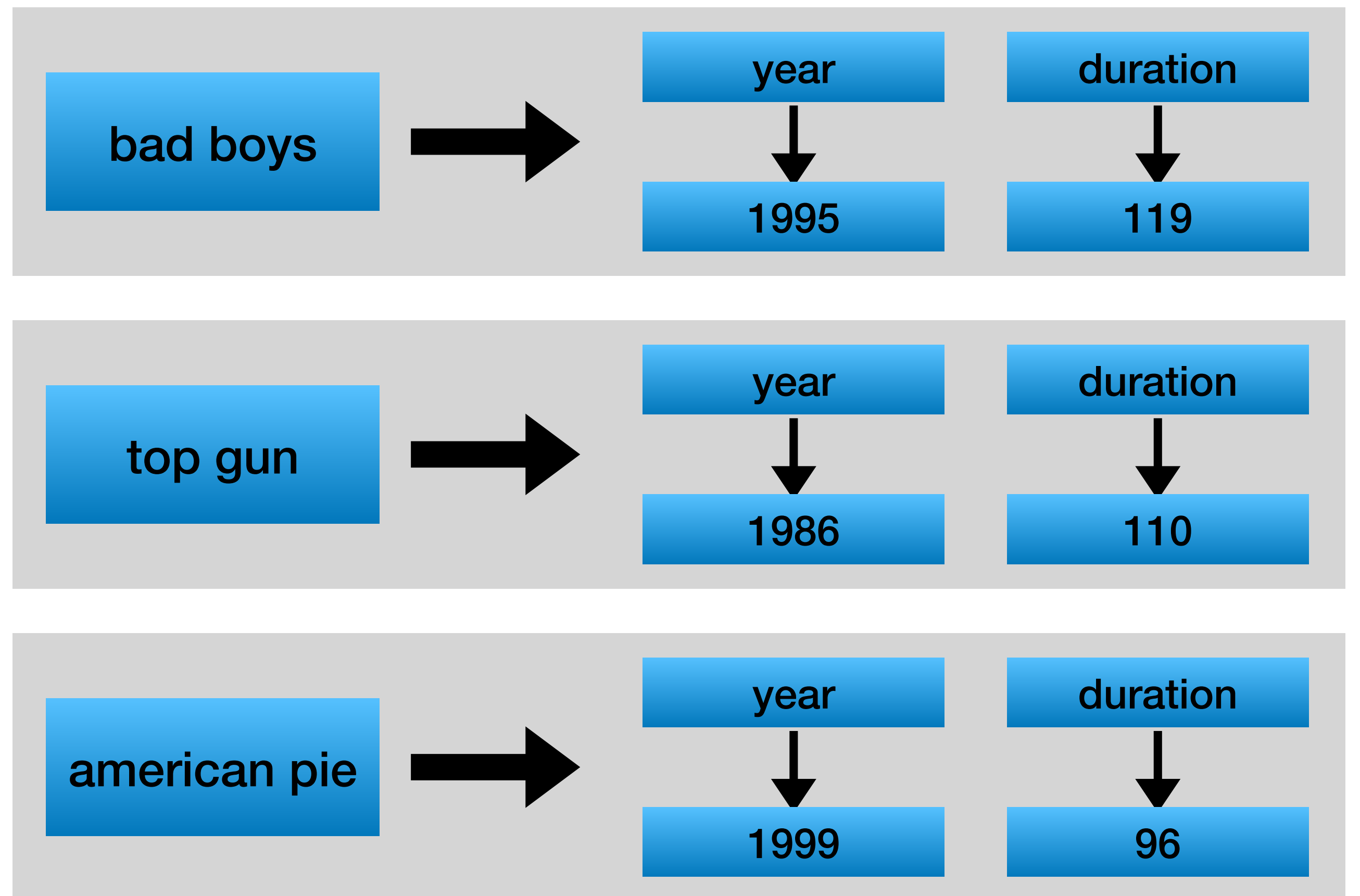


Simple partition key

- Single column (similar to previous example)

```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY (title)  
);
```

The title is the key / partition key



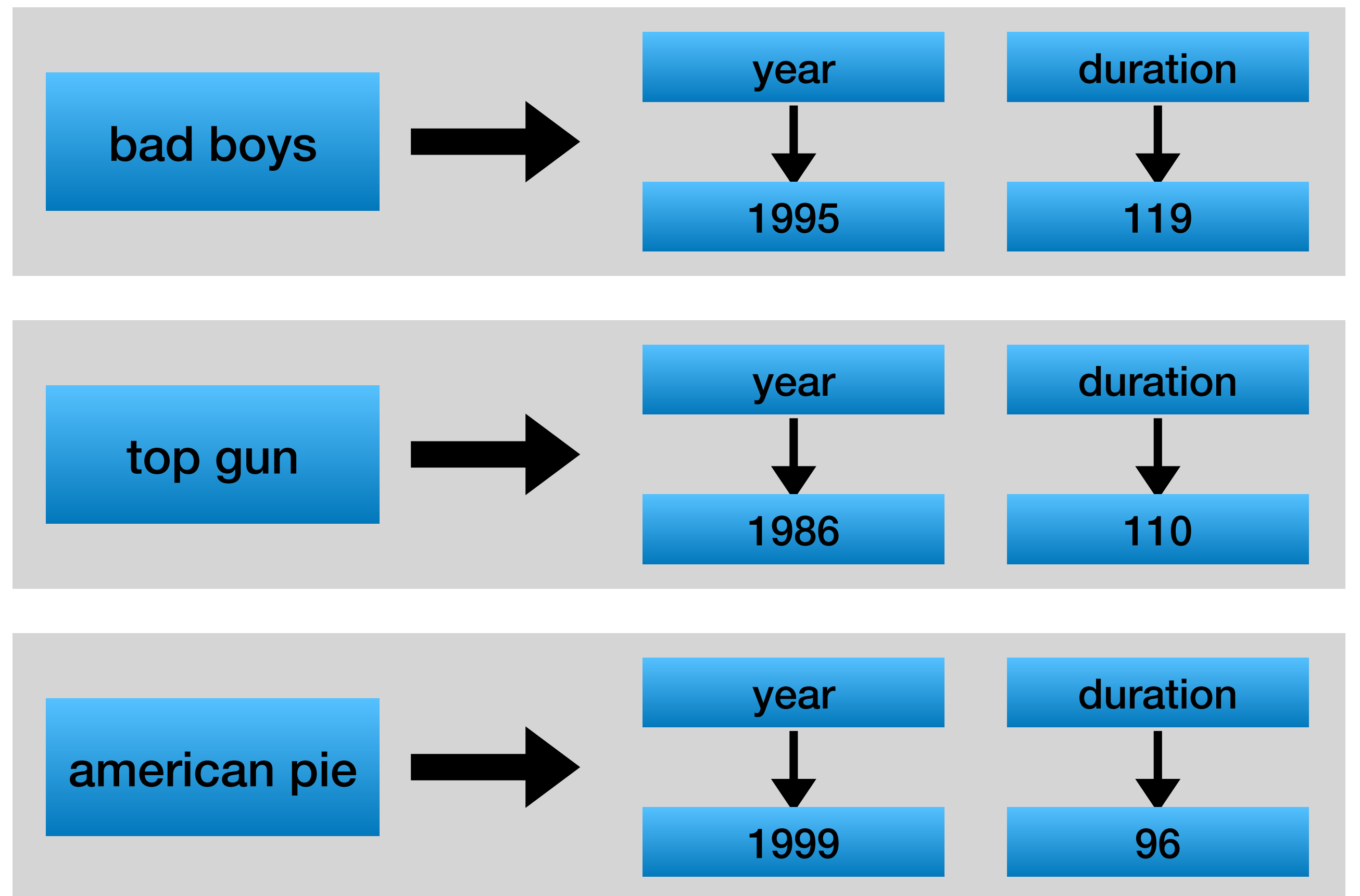
Simple partition key

- Single column (similar to previous example)

```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY (title)  
);
```

The title is the key / partition key

What happens if we have a remake?

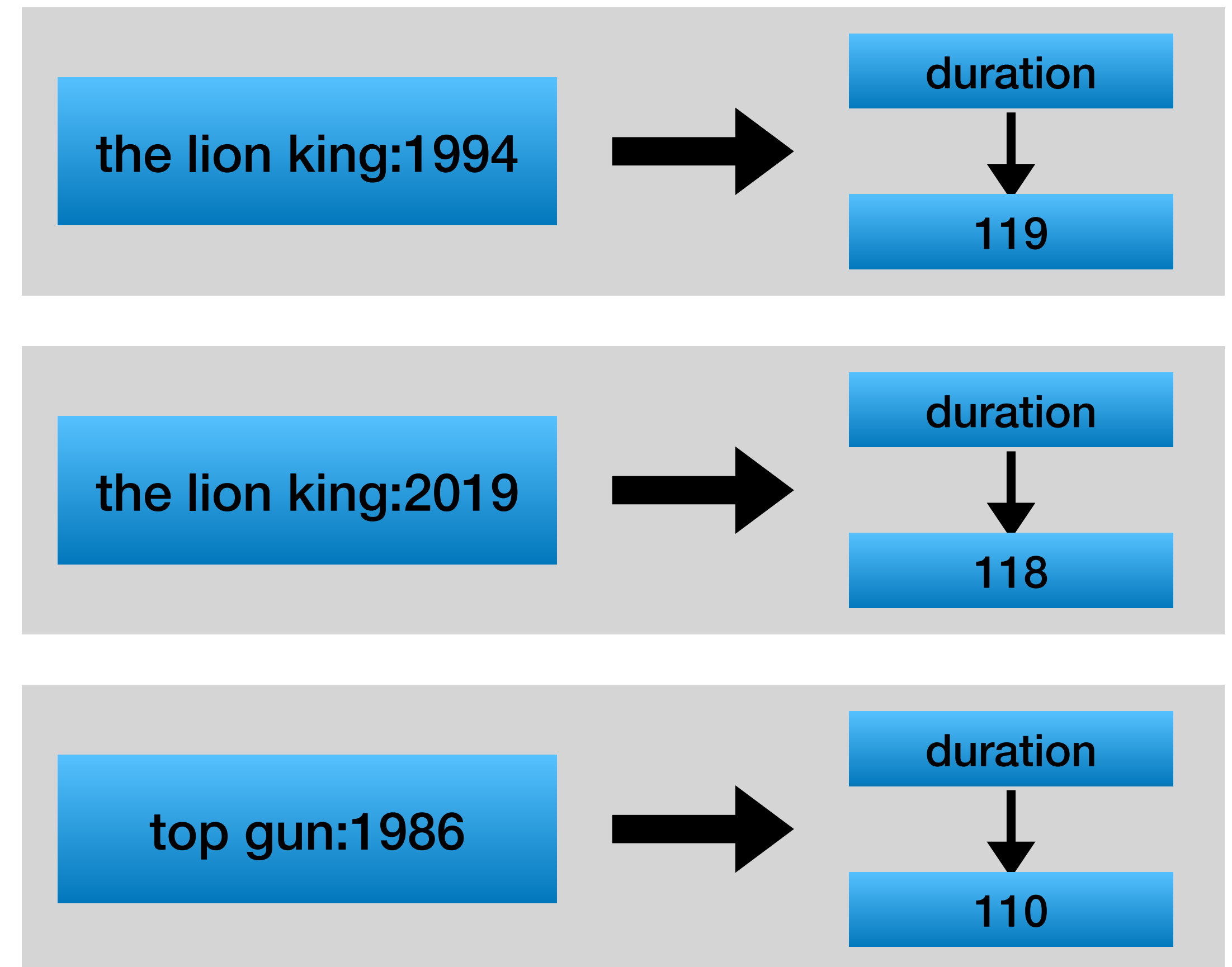


Composite partition key

- Multi value primary key —> single partition key

```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY ((title, year))  
);
```

Note the double parentheses
(more on that later)

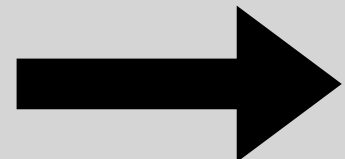


Comparison

```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY (title)  
);
```

```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY ((title, year))  
);
```

the lion king



year



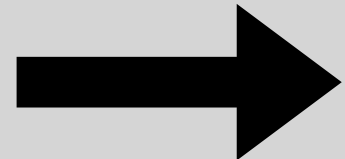
2019

duration



118

top gun



year



1986

duration



110

the lion king:2019



duration



118

top gun:1986



duration

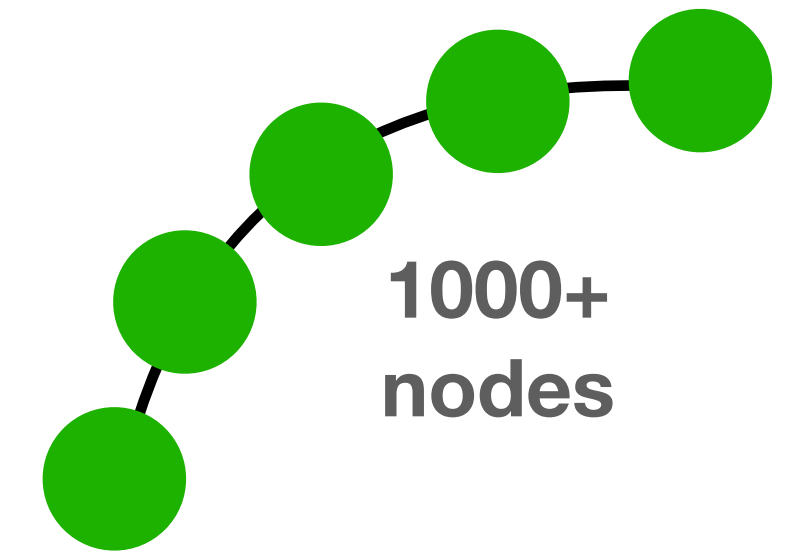


110

Comparison

```
SELECT * FROM movies  
WHERE title = "top gun"
```

What happens in each case? Why?



```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY (title)  
);
```

```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY ((title, year))  
);
```

the lion king



year

2019

duration

118



the lion king:2019



duration

118



top gun



year

1986

duration

110



top gun:1986



duration

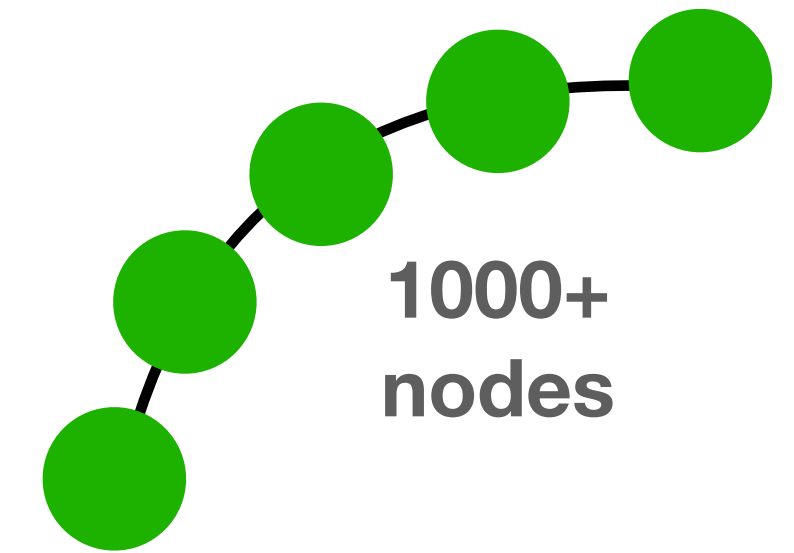
110



Comparison

```
SELECT * FROM movies  
WHERE title = "top gun"
```

What happens in each case? Why?

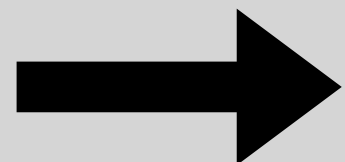


```
CREATE TABLE "movie"  
  title      TEXT  
  year       INT  
  duration   INT  
  PRIMARY KEY (title,  
);
```

query partition key = "top gun"
->
token = 12312
->
data is on node 22 - GREAT

```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY ((title, year))  
);
```

the lion king



year

2019

duration

118



the lion king:2019

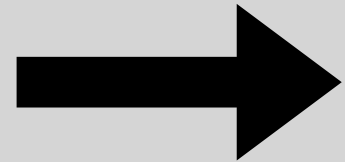


duration

118



top gun



year

1986

duration

110



top gun:1986



duration

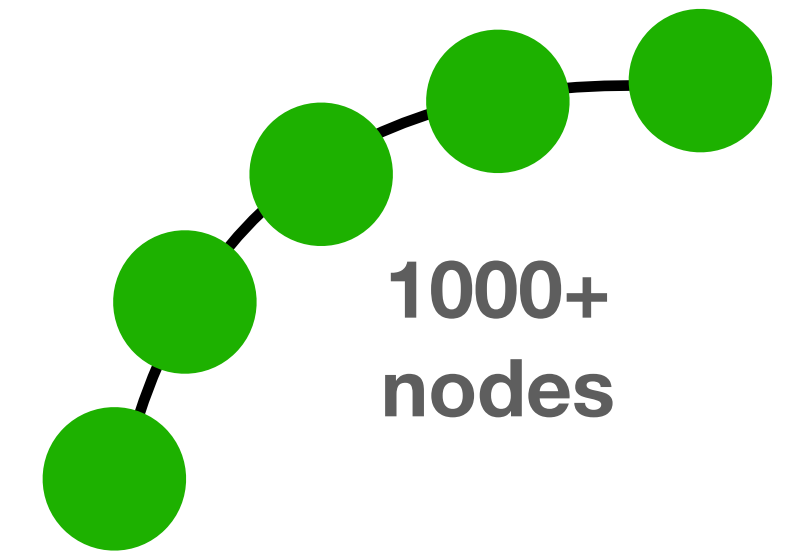
110



Comparison

```
SELECT * FROM movies  
WHERE title = "top gun"
```

What happens in each case? Why?



```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY (title  
) ;
```

```
CREATE TABLE "movies" (  
  title      TEXT  
  year       INT,  
  duration   INT,  
  PRIMARY KEY ((title, year))
```

query partition key = "top gun"
→
error - cannot calculate token
(hash function expects title + year)

the lion king



year

2019

duration

118

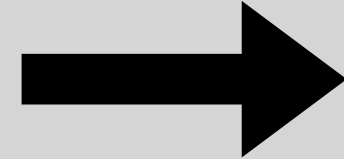
the lion king:2019



duration

118

top gun



year

1986

duration

110

top gun:1986



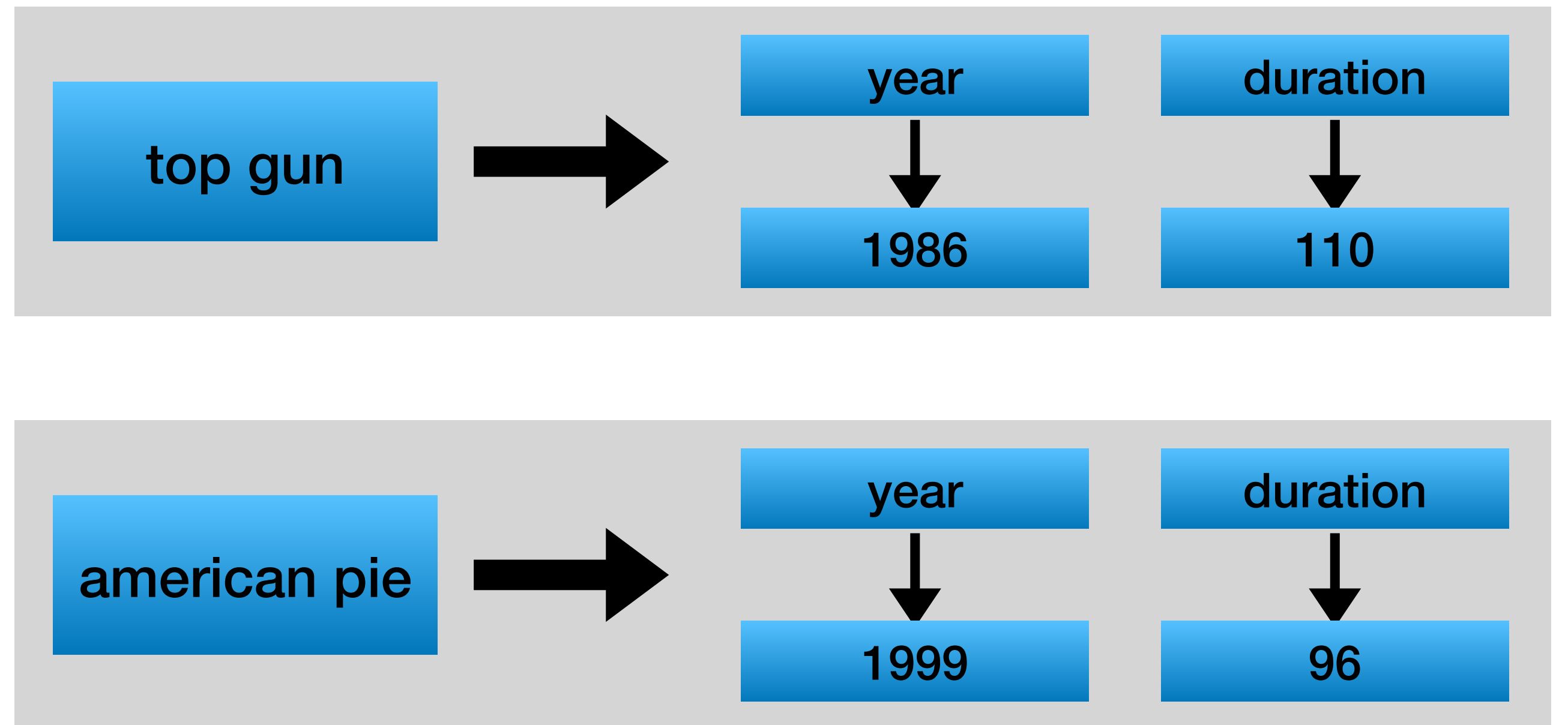
duration

110

Discussion

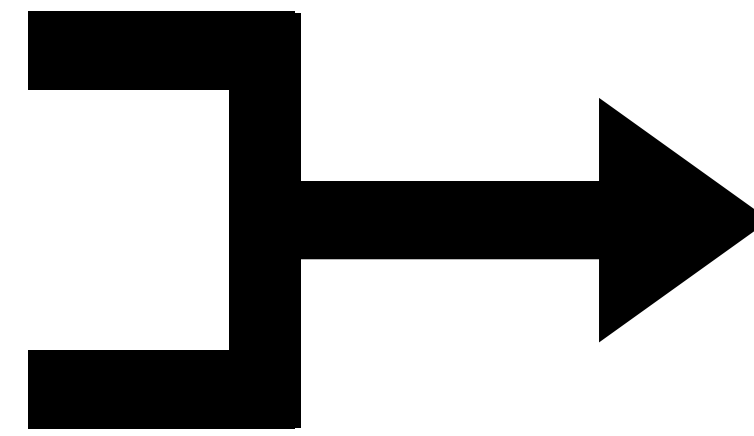
- Can you create a statement that generates a “wide row”?

```
CREATE TABLE "movies" (  
  title      TEXT  
  year      INT,  
  duration  INT,  
  PRIMARY KEY (title)  
);
```



Reminder

- A table in Cassandra has
 - Partition keys
 - Clustering columns
 - Columns



Primary key

Clustering columns

- Defines the order of the data stored within a partition
- Part of the primary key

```
CREATE TABLE "movies" (  
  year      INT,  
  title     TEXT,  
  duration  INT,  
  rating    DOUBLE,  
  PRIMARY KEY ((year), title)  
);
```

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

**double parentheses defines the partition keys,
the rest are the clustering columns**

Clustering columns

- Defines the order of the data stored within a partition
- Part of the primary key

```
CREATE TABLE "movies" (  
  year      INT,  
  title     TEXT,  
  duration  INT,  
  rating    DOUBLE,  
  PRIMARY KEY ((year), title)  
);
```

double parentheses defines the partition keys,
the rest are the clustering columns

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

Sorted within a **single** partition

Clustering columns - format on disk

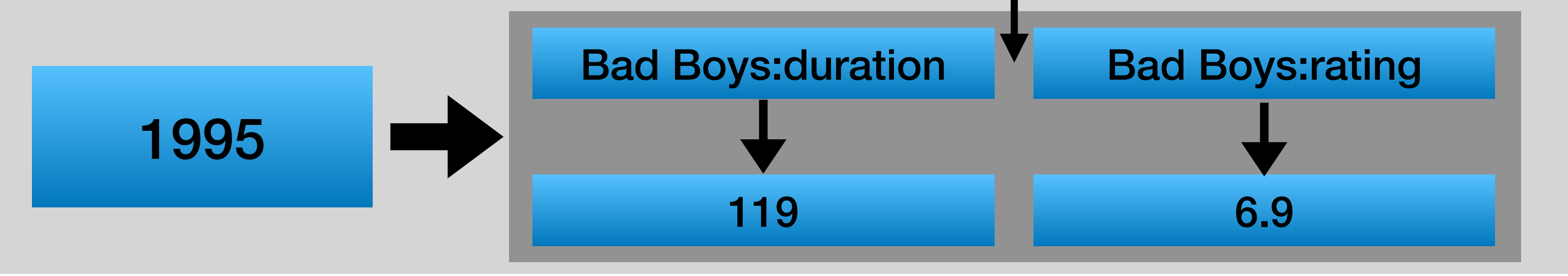
year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

```
CREATE TABLE "movies" (  
    year          INT,  
    title         TEXT,  
    duration      INT,  
    rating        DOUBLE,  
    PRIMARY KEY ((year), title)  
);
```

Clustering columns - format on disk

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

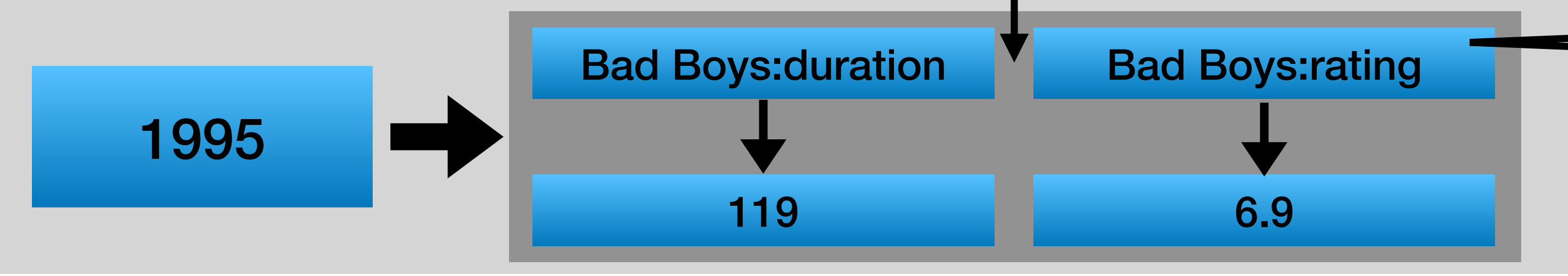
```
CREATE TABLE "movies" (  
  year          INT,  
  title        TEXT,  
  duration     INT,  
  rating       DOUBLE,  
  PRIMARY KEY ((year), title)  
);
```



Clustering columns - format on disk

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

```
CREATE TABLE "movies" (  
  year          INT,  
  title         TEXT,  
  duration      INT,  
  rating        DOUBLE,  
  PRIMARY KEY ((year), title)  
);
```

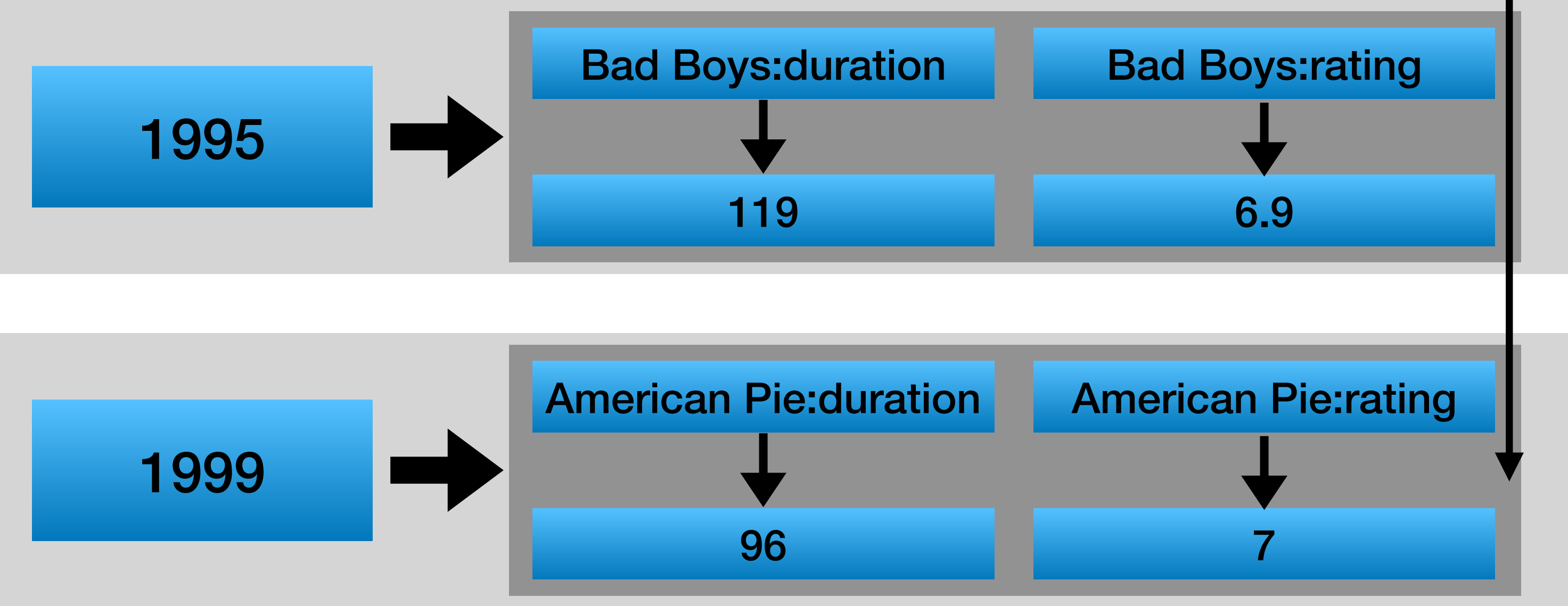


The clustering columns are the prefix of the columns

Clustering columns - format on disk

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

```
CREATE TABLE "movies" (  
  year          INT,  
  title        TEXT,  
  duration     INT,  
  rating       DOUBLE,  
  PRIMARY KEY ((year), title)  
);
```



Clustering columns - format on disk

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

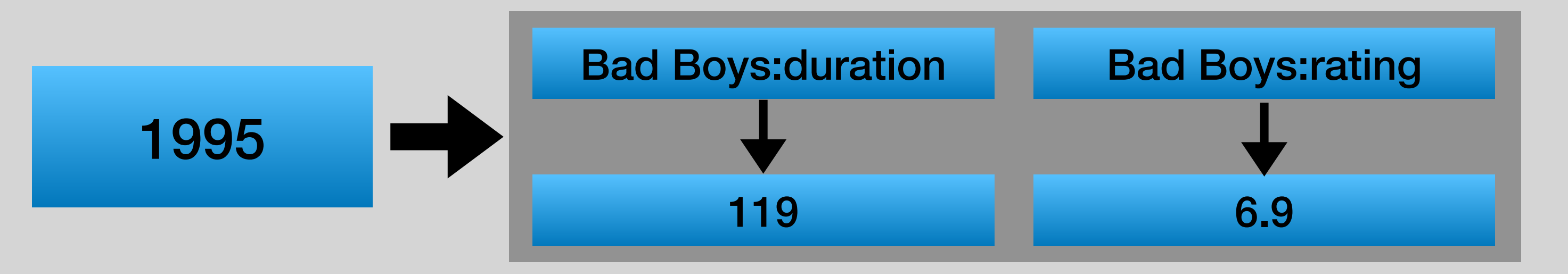
```
CREATE TABLE "movies" (  
  year          INT,  
  title        TEXT,  
  duration     INT,  
  rating       DOUBLE,  
  PRIMARY KEY ((year), title)  
);
```



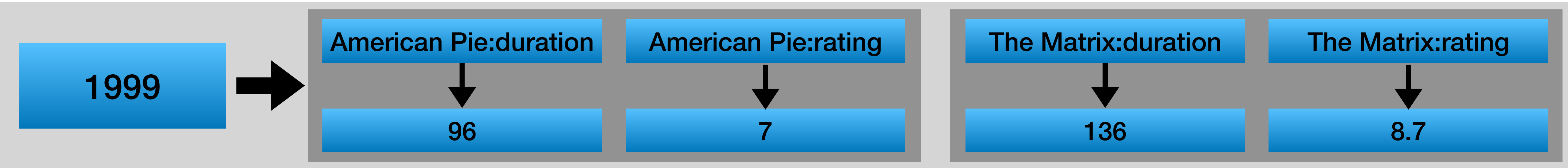
Clustering columns - format on disk

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

```
CREATE TABLE "movies" (  
  year          INT,  
  title        TEXT,  
  duration     INT,  
  rating       DOUBLE,  
  PRIMARY KEY ((year), title)  
);
```



Can this statement generate wide partitions?



Clustering columns - format on disk

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

```
CREATE TABLE "movies" (  
  year          INT,  
  title         TEXT,  
  duration      INT,  
  rating        DOUBLE,  
  PRIMARY KEY ((year), title)  
);
```

YES!

For example, what will happen if there are 10k movies in 1999?

Bad Boys:rating

6.9

Can this statement generate wide partitions?

1999

American Pie:duration

96

American Pie:rating

7

The Matrix:duration

136

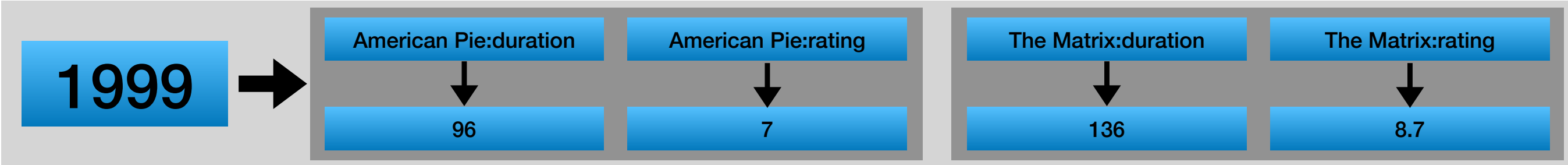
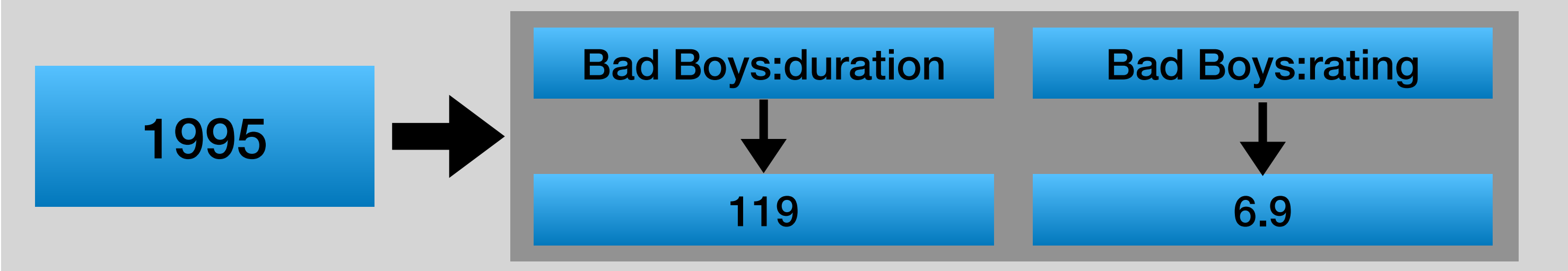
The Matrix:rating

8.7

Clustering columns - format on disk

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

```
CREATE TABLE "movies" (  
    year          INT,  
    title         TEXT,  
    duration      INT,  
    rating        DOUBLE,  
    PRIMARY KEY ((year), title)  
);
```



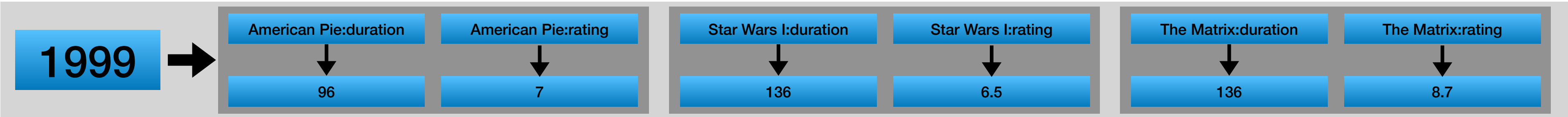
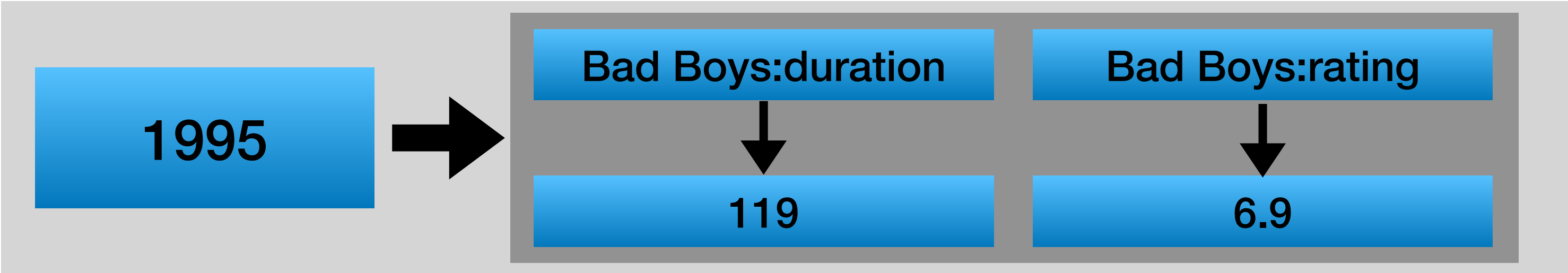
);
Can this statement generate wide partitions?

Clustering columns - format on disk

year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	Star Wars I	136	6.5
1999	The Matrix	136	8.7

```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((year), title)
);
```

Can this statement generate wide partitions?

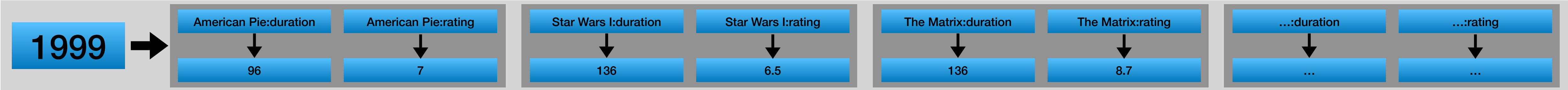
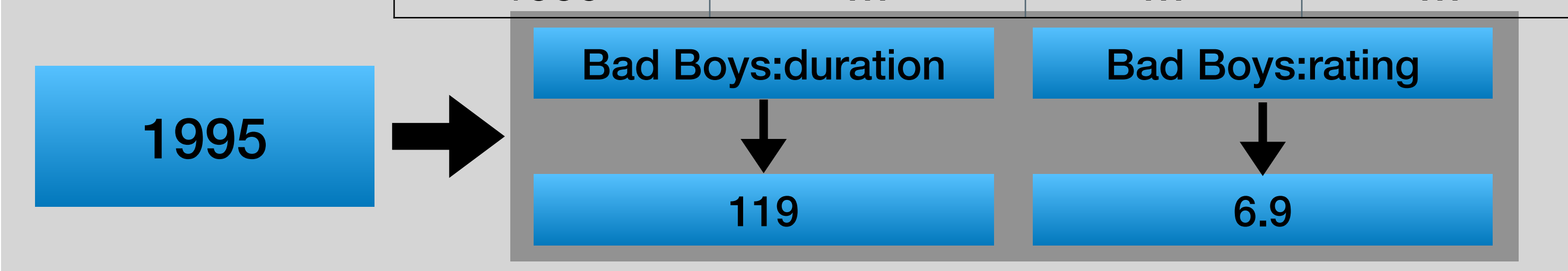


Clustering columns - format on disk

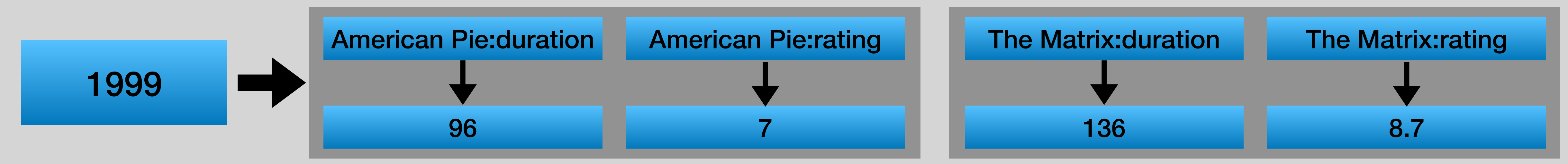
year	title	duration	rating
1995	Bad Boys	119	6.9
1999	American Pie	96	7
1999	Star Wars I	136	6.5
1999	The Matrix	136	8.7
1999

```
CREATE TABLE "movies" (
    year          INT,
    title         TEXT,
    duration      INT,
    rating        DOUBLE,
    PRIMARY KEY ((year), title)
);
```

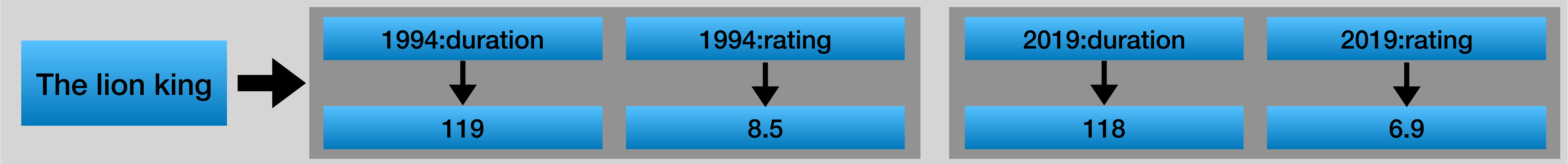
Can this statement generate wide partitions?



Clustering columns - format on disk (2)



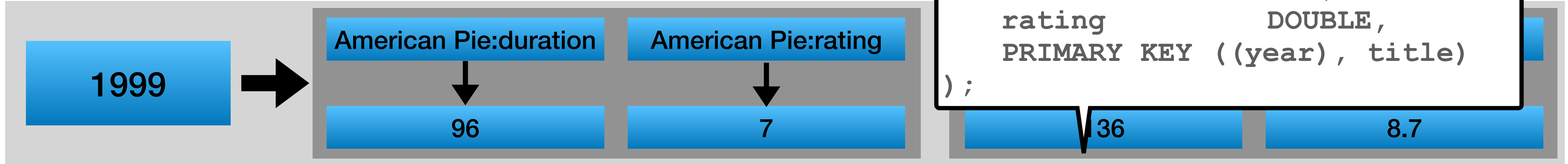
What is the difference?



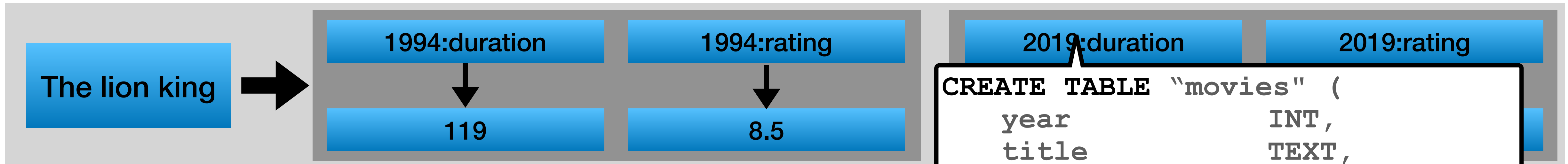
year	title	duration	rating
1994	The lion king	119	8.5
2019	The lion king	118	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

Clustering columns - form

```
CREATE TABLE "movies" (
  year      INT,
  title     TEXT,
  duration  INT,
  rating    DOUBLE,
  PRIMARY KEY ((year), title)
);
```



What is the difference?

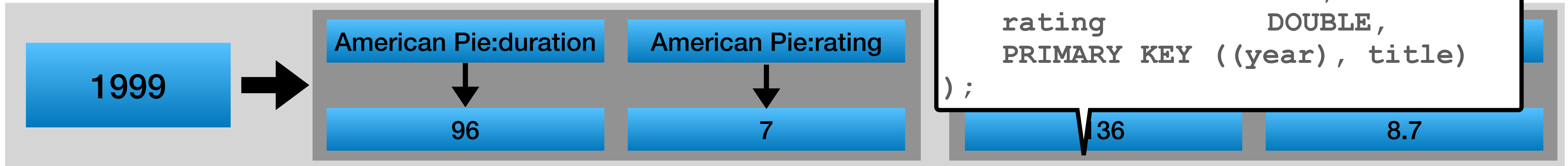


```
CREATE TABLE "movies" (
  year      INT,
  title     TEXT,
  duration  INT,
  rating    DOUBLE,
  PRIMARY KEY ((title), year)
);
```

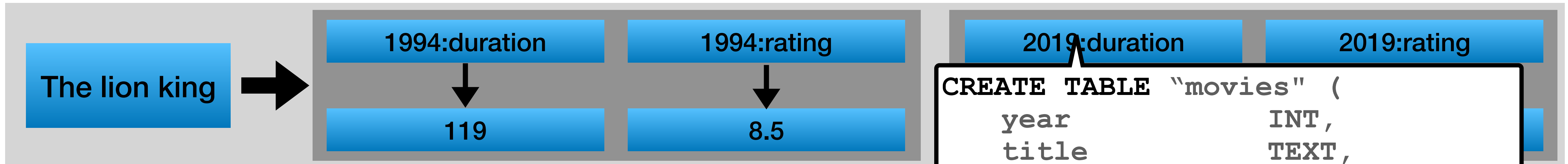
year	title	duration	rating
1994	The lion king	119	8.5
2019	The lion king	118	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

Clustering columns - form

```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((year), title)
);
```



When to use which version?

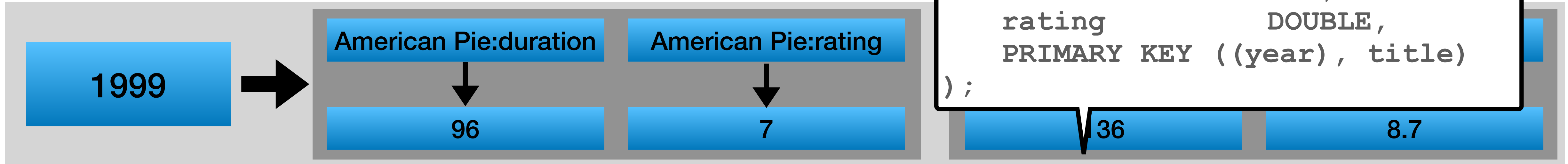


```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((title), year)
);
```

year	title	duration	rating
1994	The lion king	119	8.5
2019	The lion king	118	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

Clustering columns - form

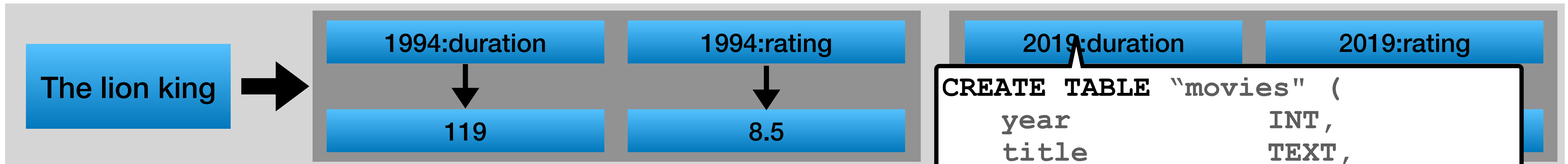
```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((year), title)
);
```



When to use which version?

Points to consider

1. The way we will query the data
2. The size of the partition

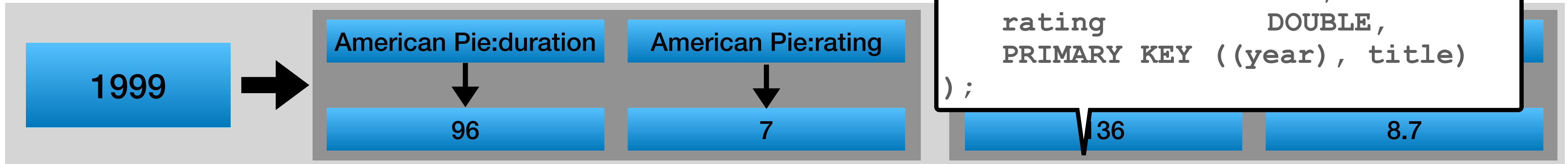


```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((title), year)
);
```

year	title	duration	rating
1994	The lion king	119	8.5
2019	The lion king	118	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

Clustering columns - form

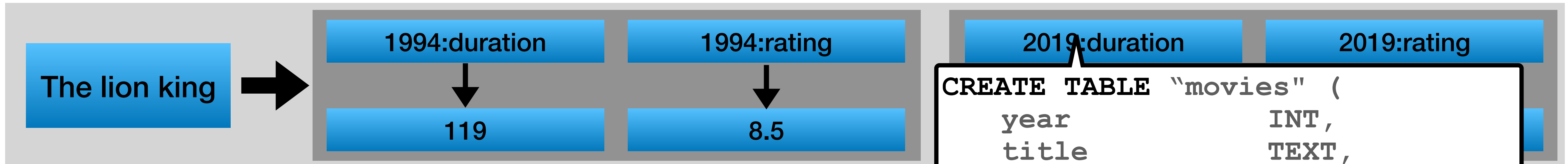
```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating       DOUBLE,
  PRIMARY KEY ((year), title)
);
```



If we know the year and the title at query time, we can query both versions

then to use which version?

- Points to consider
1. The way we will query the data
 2. The size of the partition

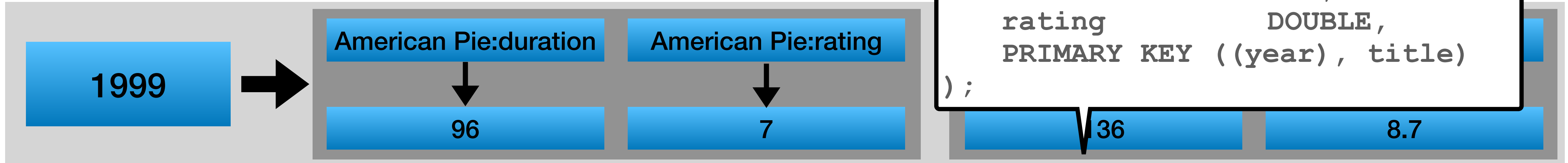


```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating       DOUBLE,
  PRIMARY KEY ((title), year)
);
```

year	title	duration	rating
1994	The lion king	119	8.5
2019	The lion king	118	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

Clustering columns - form

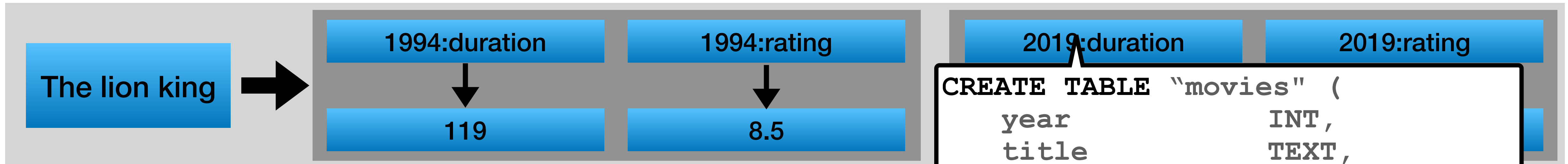
```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((year), title)
);
```



If we want to query by year

en to use which version?

- Points to consider
1. The way we will query the data
 2. The size of the partition

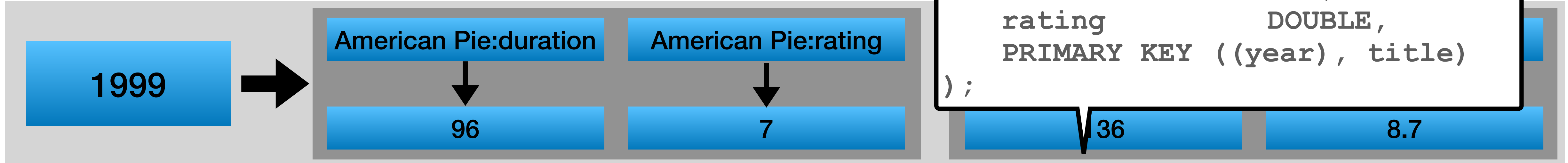


```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((title), year)
);
```

year	title	duration	rating
1994	The lion king	119	8.5
2019	The lion king	118	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

Clustering columns - form

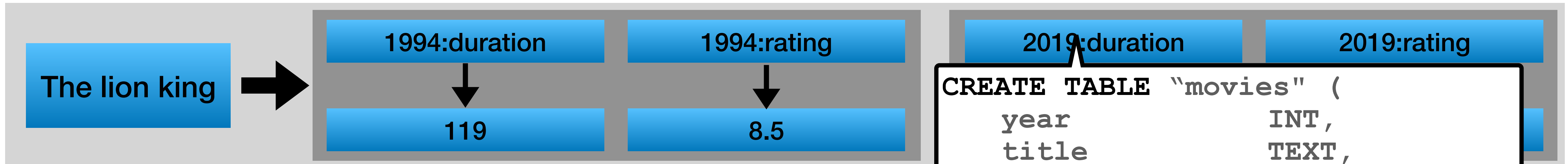
```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((year), title)
);
```



If we want to query by title

then to use which version?

- Points to consider
1. The way we will query the data
 2. The size of the partition

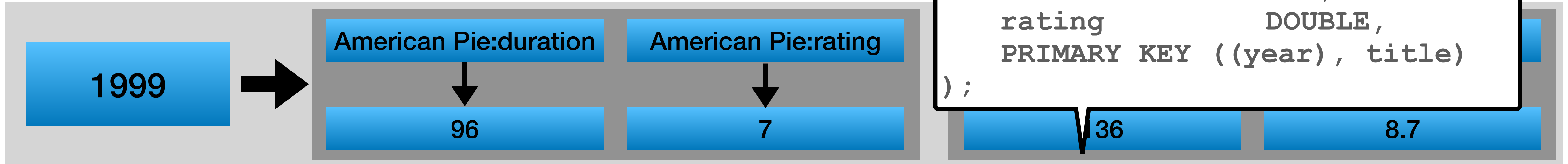


```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((title), year)
);
```

year	title	duration	rating
1994	The lion king	119	8.5
2019	The lion king	118	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

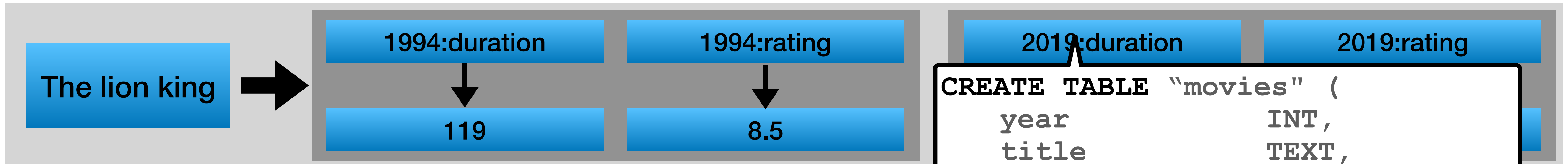
Clustering columns - form

```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((year), title)
);
```



Which version will probably create bigger partitions?

- Points to consider**
1. The way we will query the data
 2. The size of the partition

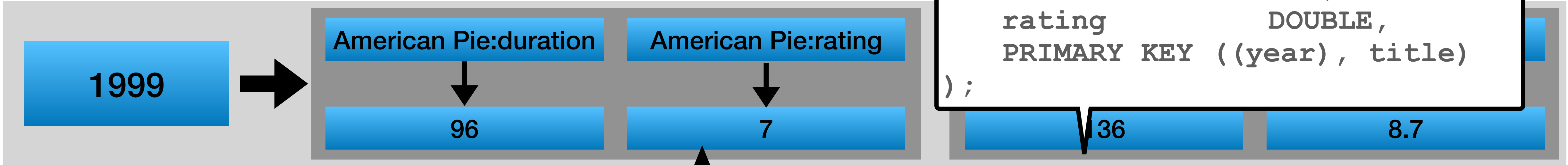


```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((title), year)
);
```

year	title	duration	rating
1994	The lion king	119	8.5
2019	The lion king	118	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

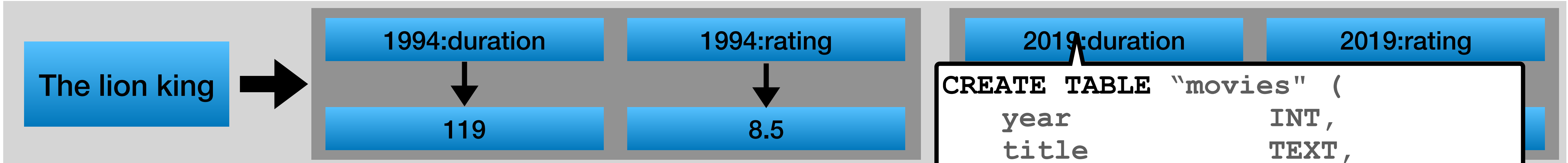
Clustering columns - form

```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((year), title)
);
```



Which version will probably create bigger partitions?

- Points to consider**
1. The way we will query the data
 2. The size of the partition



```
CREATE TABLE "movies" (
  year          INT,
  title         TEXT,
  duration      INT,
  rating        DOUBLE,
  PRIMARY KEY ((title), year)
);
```

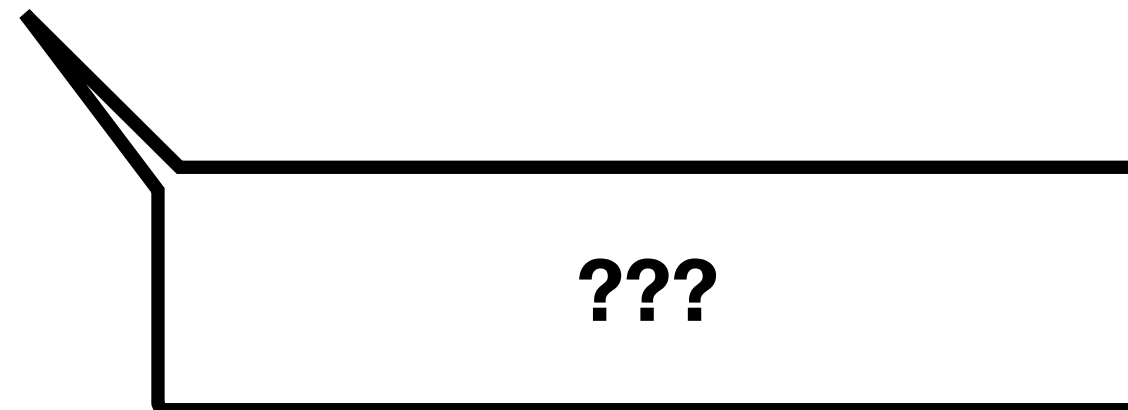
year	title	duration	rating
1994	The lion king	119	8.5
2019	The lion king	118	6.9
1999	American Pie	96	7
1999	The Matrix	136	8.7

Note on partition size

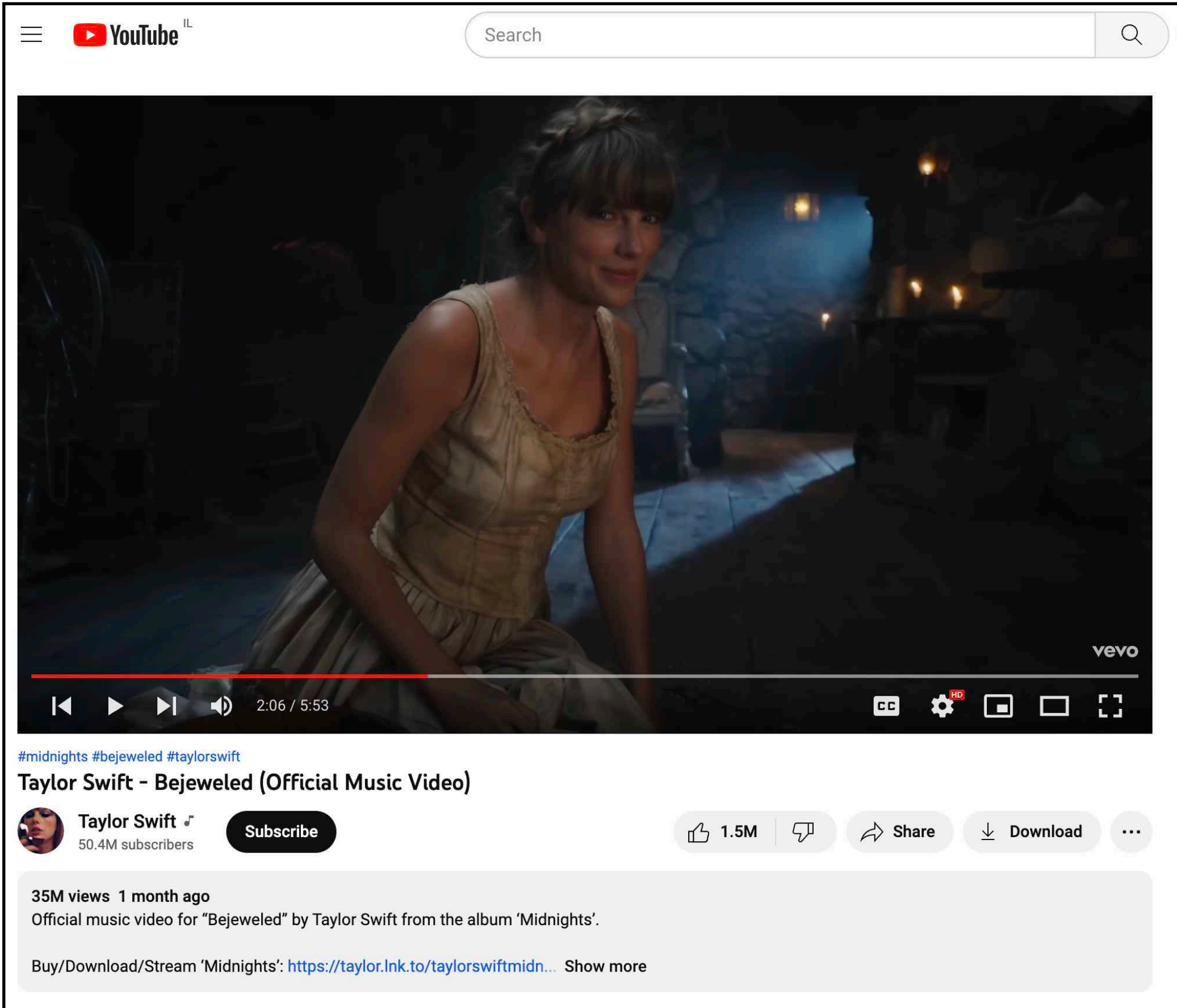
- The partition size is a crucial attribute for Cassandra performance and maintenance
- Ideally less than 100MB
 - Current versions supports much larger partitions (in GB)
- Can you think of a data model with a partition larger than 10GB?

Note on partition size

- The partition size is a crucial attribute for Cassandra performance and maintenance
- Ideally less than 100MB
 - Current versions supports much larger partitions (in GB)
- Can you think of a data model with a partition larger than 10GB?



Note on partition size



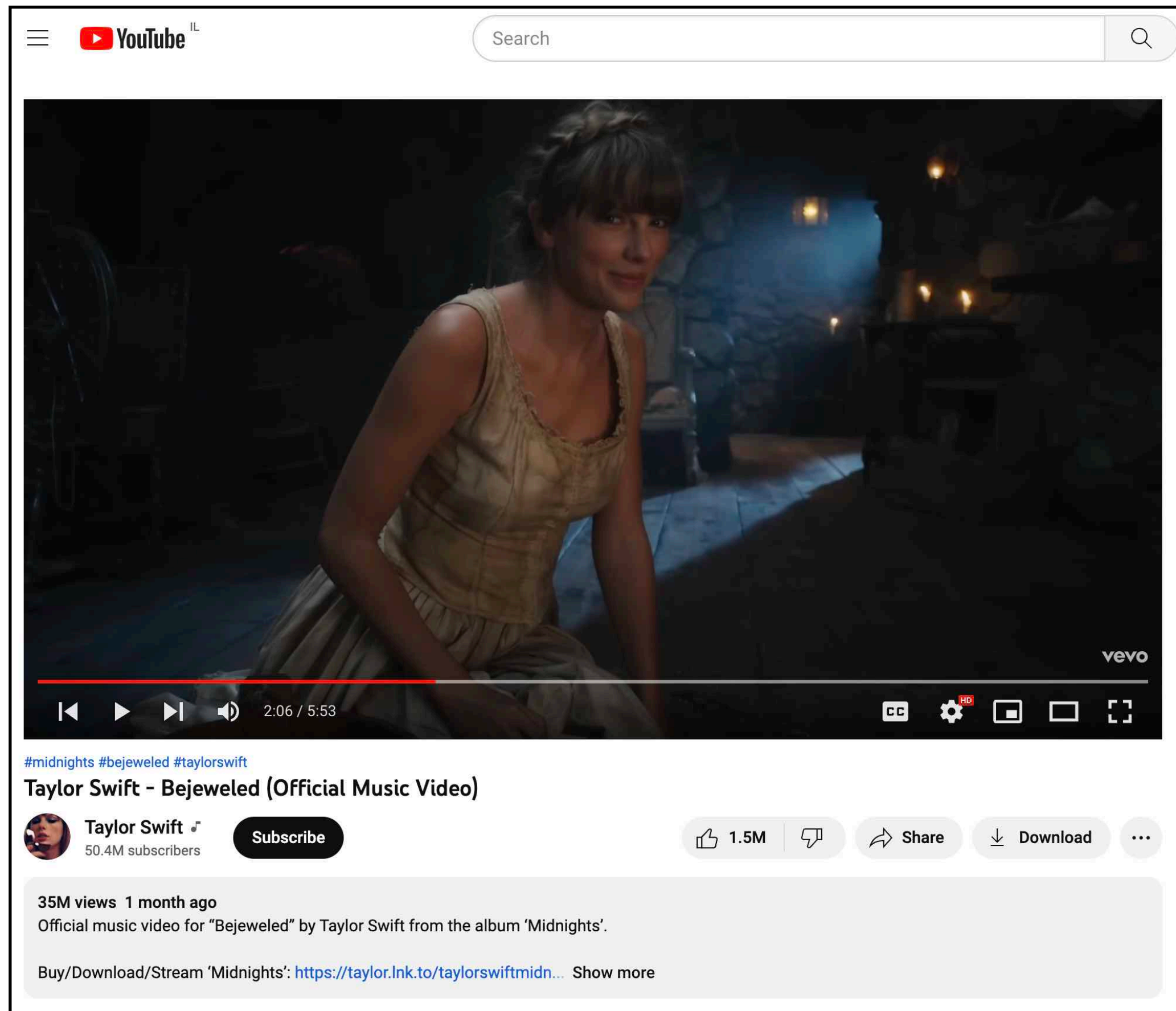
The image shows a screenshot of a YouTube video player. At the top left is the YouTube logo and a search bar. The video frame shows Taylor Swift in a dark, atmospheric setting, wearing a light-colored, sleeveless dress. Below the video is a control bar with play/pause, volume, and progress indicators (2:06 / 5:53). Below the video player, the video title is "Taylor Swift - Bejeweled (Official Music Video)". To the left of the title is the Taylor Swift channel name and a "Subscribe" button. To the right are icons for likes (1.5M), comments, share, and download. Below the title, it says "35M views 1 month ago" and "Official music video for 'Bejeweled' by Taylor Swift from the album 'Midnights'." At the bottom, there is a link to buy/download/stream the album: "Buy/Download/Stream 'Midnights': [https://taylor.lnk.to/taylorswiftmidn...](\"https://taylor.lnk.to/taylorswiftmidn...\") Show more".

Note on partition size

```
CREATE TABLE "youtube_views" (  
  video_id      BIGINT,  
  view_id       TIMEUUID,  
  user_id       BIGINT,  
  percentage    INT,  
  ...  
  PRIMARY KEY ((video_id), view_id)  
);
```

Assume YouTube saves for each video view some details (userID, device, time, percentage viewed...).

For simplicity, lets say 0.1k for each view



The screenshot shows a YouTube video player for Taylor Swift's "Bejeweled (Official Music Video)". The video is paused at 2:06 / 5:53. The video player includes standard controls like play/pause, volume, and full screen. Below the video, the title "Taylor Swift - Bejeweled (Official Music Video)" is displayed, along with the channel name "Taylor Swift" and "50.4M subscribers". The video has 1.5M likes and 35M views. The description mentions it's the official music video for "Bejeweled" from the album "Midnights".

Note on partition size

```
CREATE TABLE "youtube_views" (  
  video_id      BIGINT,  
  view_id       TIMEUUID,  
  user_id       BIGINT,  
  percentage    INT,  
  ...  
  PRIMARY KEY ((video_id), view_id)  
);
```

Assume YouTube saves for each video view some details (userID, device, time, percentage viewed...).

For simplicity, lets say 0.1k for each view

#midnights #bejeweled #taylorswift
Taylor Swift - Bejeweled (Official Music Video)
Taylor Swift 50.4M subscribers
Subscribe
1.5M
Share
Download
35M views 1 month ago
Official music video for "Bejeweled" by Taylor Swift from the album 'Midnights'.
Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

35m * 0.1k ≈ 3.5GB

Note on partition size

#midnights #bejeweled #taylorswift
Taylor Swift - Bejeweled (Official Music Video)
Taylor Swift 50.4M subscribers **Subscribe**
1.5M
35M views 1 month ago
Official music video for "Bejeweled" by Taylor Swift from the album 'Midnights'.
Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

#TSAntiHeroChallenge #midnights #antihero
Taylor Swift - Anti-Hero
Taylor Swift 50.4M subscribers **Subscribe**
2.6M
74M views 1 month ago
Official music video for "Anti-Hero" by Taylor Swift from the album 'Midnights'.
Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

35m * 0.1k ≈ 3.5GB

Note on partition size

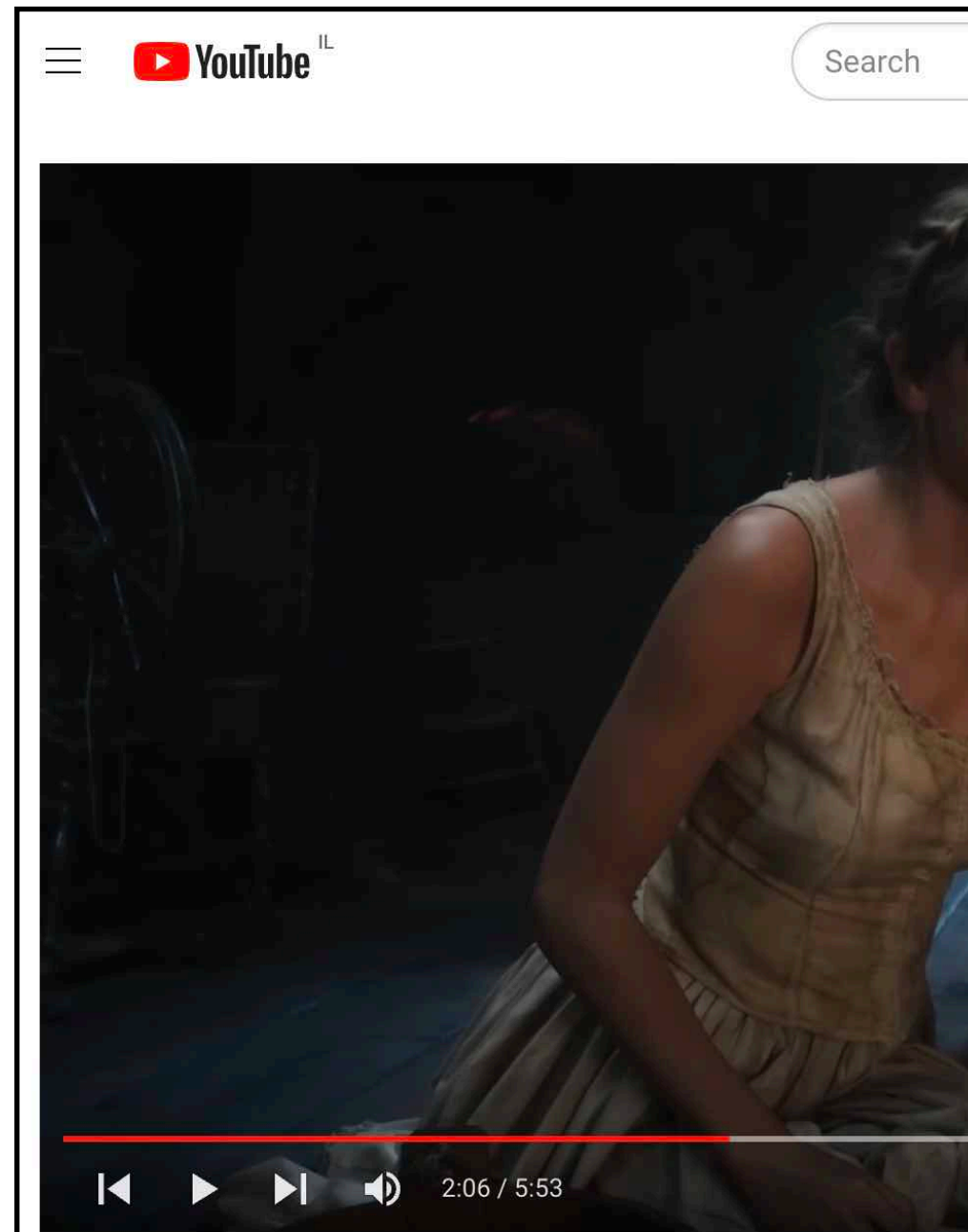
#midnights #bejeweled #taylorswift
Taylor Swift - Bejeweled (Official Music Video)
Taylor Swift 50.4M subscribers
35M views 1 month ago
Official music video for "Bejeweled" by Taylor Swift from the album 'Midnights'.
Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

$35m * 0.1k \approx 3.5GB$

#TSAntiHeroChallenge #midnights #antihero
Taylor Swift - Anti-Hero
Taylor Swift 50.4M subscribers
74M views 1 month ago
Official music video for "Anti-Hero" by Taylor Swift from the album 'Midnights'.
Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

$74m * 0.1k \approx 7.4GB$

Note on



#midnights #bejeweled #taylorswift
Taylor Swift - Bejeweled (Official Music Video)

Taylor Swift 50.4M subscribers **Subscribe**

35M views 1 month ago
Official music video for "Bejeweled" by Taylor Swift from the album 'Midnights'.

Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

35m * 0.1k ≈ 3.5GB

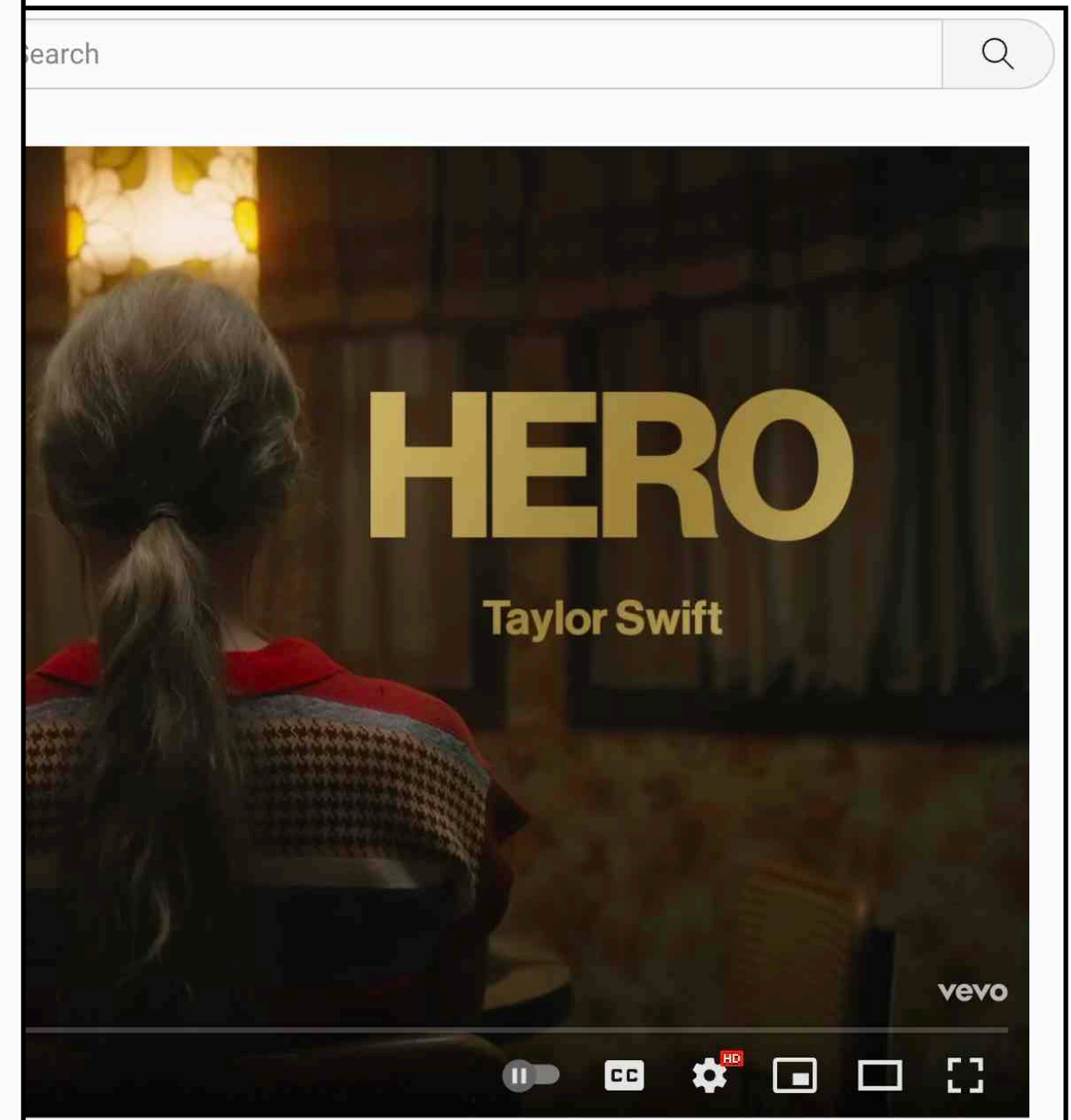
Taylor Swift - Shake It Off

Taylor Swift 50.4M subscribers **Subscribe**

3.2B views 8 years ago
▶ Exclusive Merch: <https://store.taylorswift.com>

▶ Follow Taylor Swift Online Show more

1.5M **Share** **Download** ...



Taylor Swift - Anti-Hero (Official Music Video)

Taylor Swift 50.4M subscribers **Subscribe**


74M views 1 month ago
Official music video for "Anti-Hero" by Taylor Swift from the album 'Midnights'.

Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

2.6M **Share** **Download** ...

74m * 0.1k ≈ 7.4GB

Note on



YouTube interface showing the video "Taylor Swift - Shake It Off". The video player shows Taylor Swift in a red jacket holding a boombox. The video title is "Taylor Swift - Shake It Off" and the channel is "Taylor Swift" with 50.4M subscribers. The video has 3.2B views and was uploaded 8 years ago. The video player shows a progress bar at 0:18 / 4:01. The video player includes standard YouTube controls like play, volume, and full screen.

Taylor Swift - Shake It Off

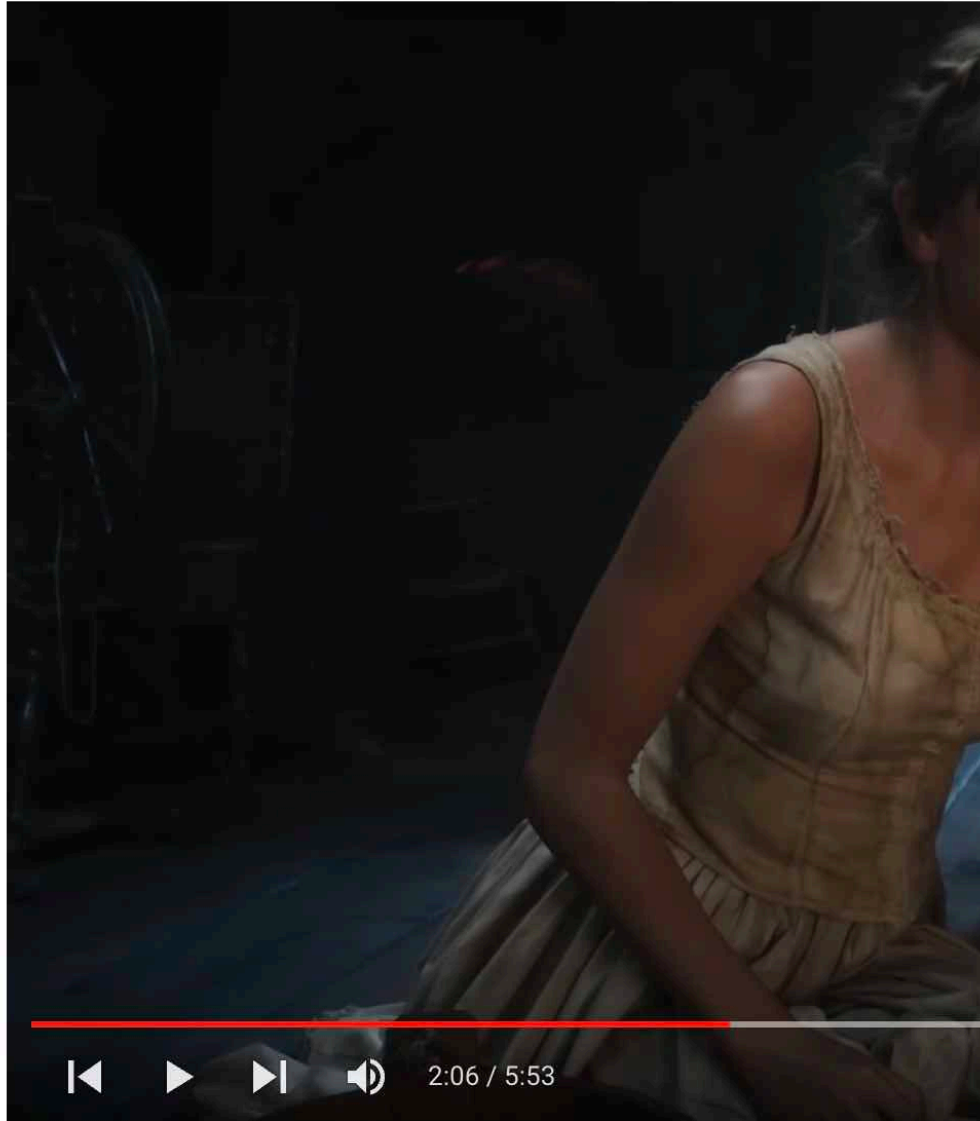
Taylor Swift 50.4M subscribers

3.2B views 8 years ago

Exclusive Merch: <https://store.taylorswift.com>

Follow Taylor Swift Online Show more

$3.2b * 0.1k \approx 320GB (!)$



YouTube interface showing the video "Taylor Swift - Bejeweled (Official Music Video)". The video player shows Taylor Swift in a gold dress. The video title is "Taylor Swift - Bejeweled (Official Music Video)" and the channel is "Taylor Swift" with 50.4M subscribers. The video has 35M views and was uploaded 1 month ago. The video player shows a progress bar at 2:06 / 5:53. The video player includes standard YouTube controls like play, volume, and full screen.

Taylor Swift - Bejeweled (Official Music Video)


Taylor Swift 50.4M subscribers

35M views 1 month ago

Official music video for "Bejeweled" by Taylor Swift from the album 'Midnights'.

Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

$35m * 0.1k \approx 3.5GB$



YouTube interface showing the video "Taylor Swift - Anti-Hero". The video player shows Taylor Swift from behind in a red and grey sweater. The video title is "HERO Taylor Swift" and the channel is "Taylor Swift" with 50.4M subscribers. The video has 2.6M views and was uploaded 1 month ago. The video player shows a progress bar at 0:00 / 0:00. The video player includes standard YouTube controls like play, volume, and full screen.

HERO Taylor Swift

Taylor Swift 50.4M subscribers

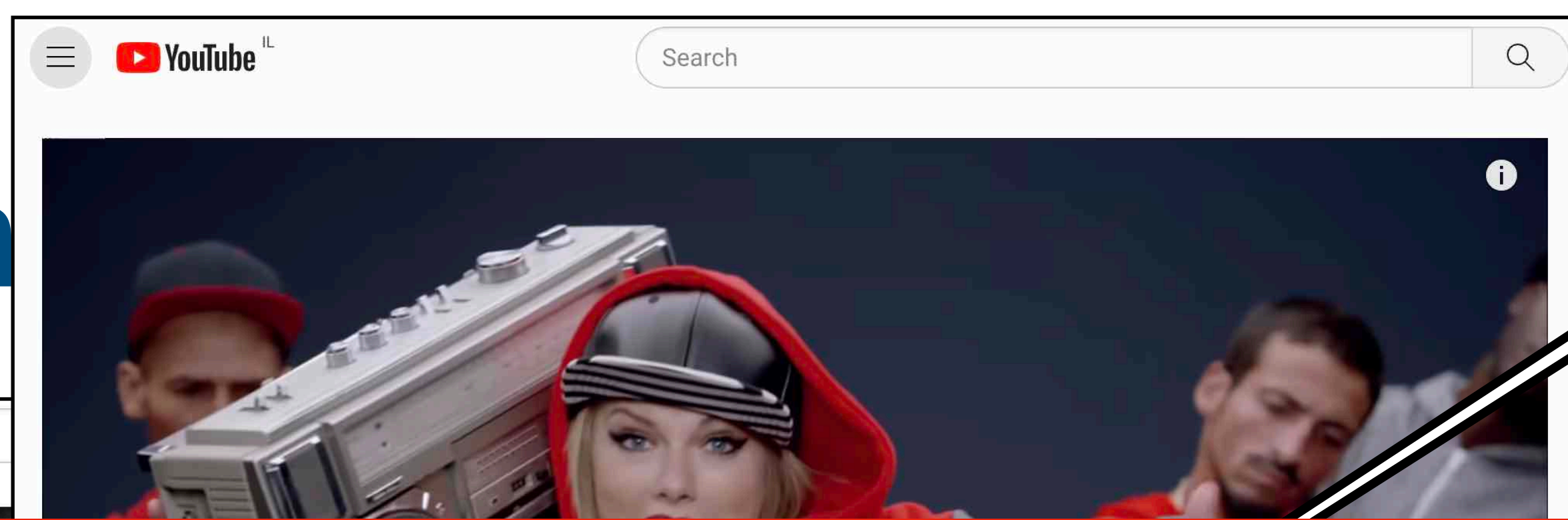
2.6M views 1 month ago

Official music video for "Anti-Hero" by Taylor Swift from the album 'Midnights'.

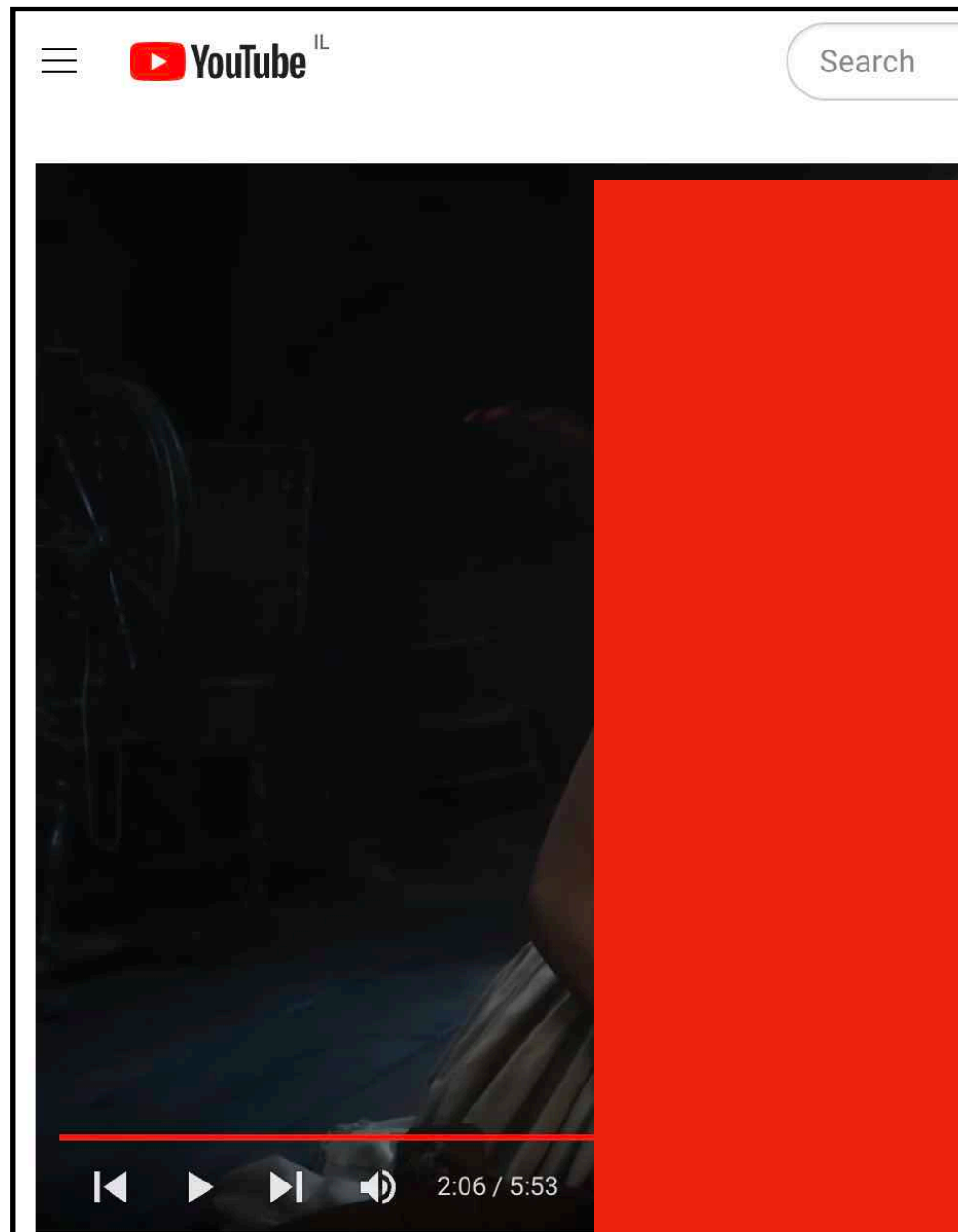
Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

$74m * 0.1k \approx 7.4GB$

Note on



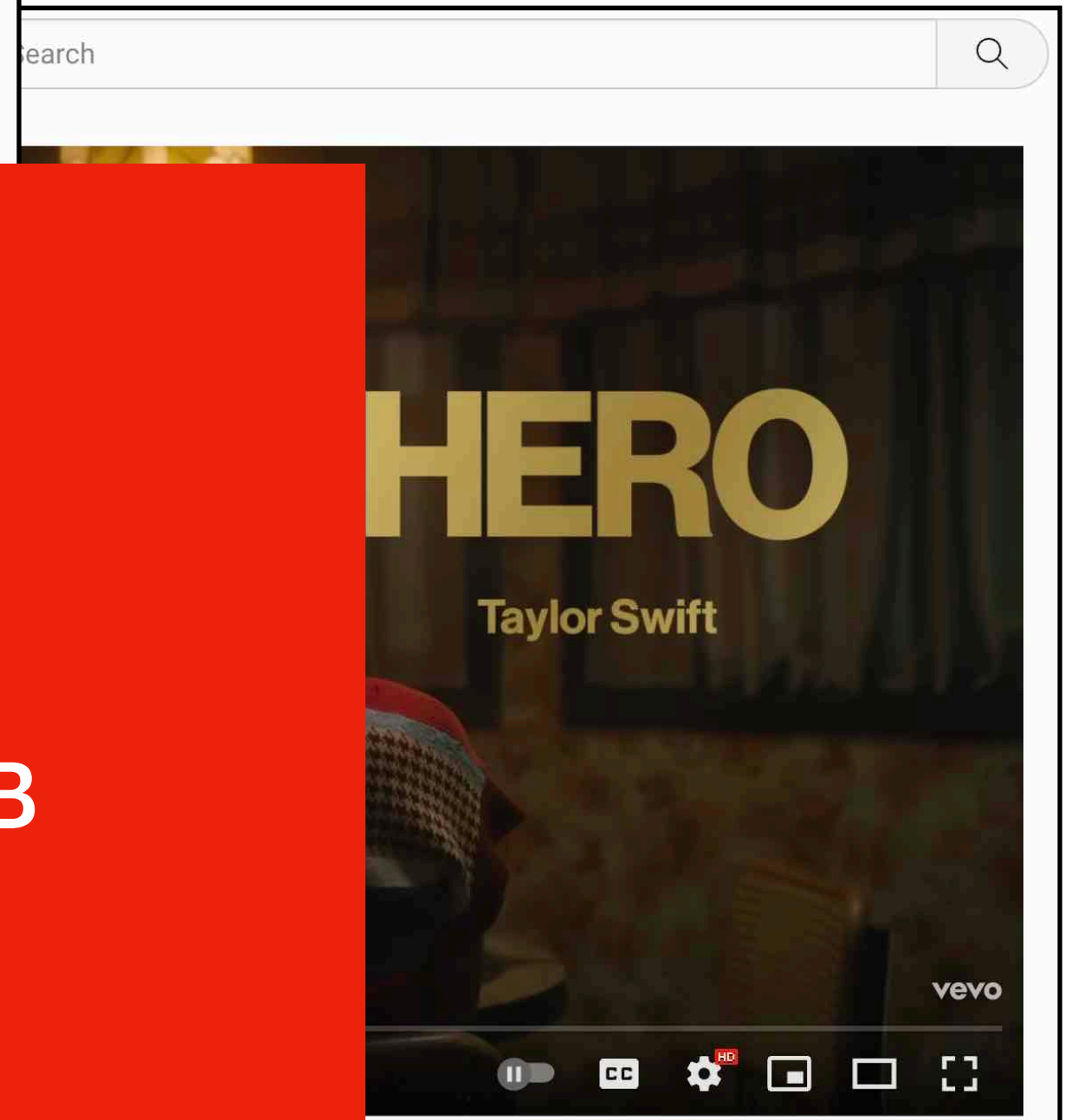
$3.2b * 0.1k \approx 320GB (!)$



REMINDER
Each partition should ideally be less than 100MB

#midnights #bejeweled #taylorswift
Taylor Swift - Bejeweled (Official Music Video)
Taylor Swift 50.4M subscribers
35M views 1 month ago
Official music video for "Bejeweled" by Taylor Swift from the album 'Midnights'.
Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

$35m * 0.1k \approx 3.5GB$



2.6M
74M views 1 month ago
Official music video for "Anti-Hero" by Taylor Swift from the album 'Midnights'.
Buy/Download/Stream 'Midnights': <https://taylor.lnk.to/taylorswiftmidn...> Show more

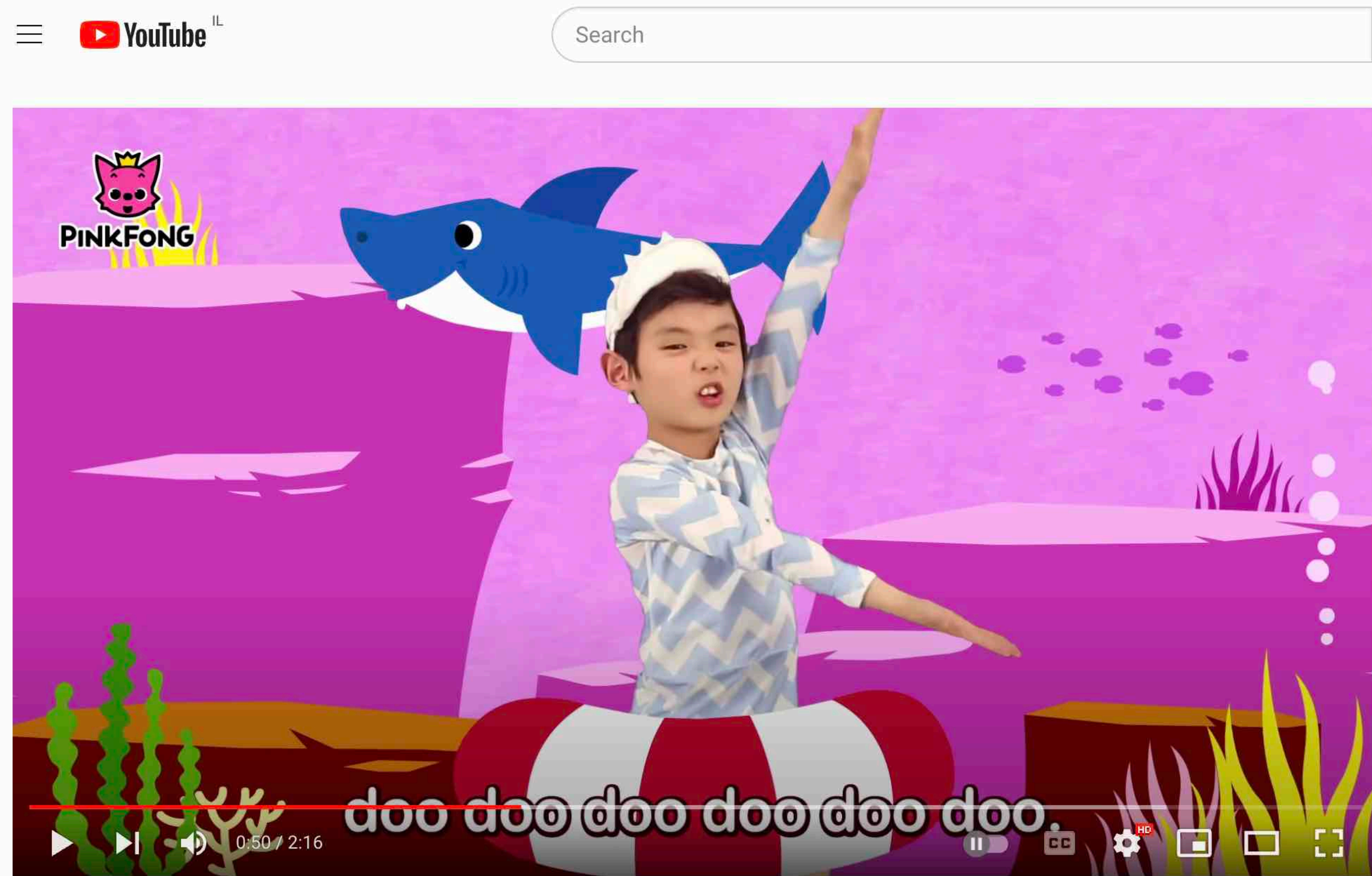
$74m * 0.1k \approx 7.4GB$

Top 30 most-viewed YouTube videos^[3]

No. ↕	Video name ↕	Uploader ↕	Views (billions) ↕	Publication date ↕	Note
1.	" Baby Shark Dance " ^[4]	Pinkfong Baby Shark - Kids' Songs & Stories	11.83	June 17, 2016	[A]
2.	" Despacito " ^[7]	Luis Fonsi	8.02	January 12, 2017	[B]
3.	" Johny Johny Yes Papa " ^[14]	LooLoo Kids	6.54	October 8, 2016	[C]
4.	" Shape of You " ^[15]	Ed Sheeran	5.86	January 30, 2017	[D]
5.	" Bath Song " ^[17]	Cocomelon – Nursery Rhymes	5.81	May 2, 2018	[E]
6.	" See You Again " ^[18]	Wiz Khalifa	5.71	April 6, 2015	[F]
7.	" Phonics Song with Two Words " ^[23]	ChuChu TV	5.04	March 6, 2014	[G]
8.	" Uptown Funk " ^[24]	Mark Ronson	4.77	November 19, 2014	[H]
9.	" Learning Colors – Colorful Eggs on a Farm " ^[25]	Miroshka TV	4.74	February 27, 2018	[I]
10.	" Wheels on the Bus " ^[26]	Cocomelon – Nursery Rhymes	4.69	May 24, 2018	
11.	" Gangnam Style " ^[27]	Psy	4.62	July 15, 2012	[J]
12.	" <i>Masha and the Bear</i> – Recipe for Disaster " ^[32]	Get Movies	4.52	January 31, 2012	[K]
13.	" Dame Tu Cosita " ^[33]	El Chombo	4.15	April 5, 2018	
14.	" Sugar " ^[34]	Maroon 5	3.79	January 14, 2015	
15.	" Roar " ^[35]	Katy Perry	3.69	September 5, 2013	
16.	" Counting Stars " ^[36]	OneRepublic	3.69	May 31, 2013	
17.	" Axel F " ^[37]	Crazy Frog	3.63	June 16, 2009	
18.	" Sorry " ^[38]	Justin Bieber	3.61	October 22, 2015	
19.	" Thinking Out Loud " ^[39]	Ed Sheeran	3.52	October 7, 2014	
20.	" Baa Baa Black Sheep " ^[40]	Cocomelon – Nursery Rhymes	3.44	June 25, 2018	
21.	" Dark Horse " ^[41]	Katy Perry	3.40	February 20, 2014	
22.	" Faded " ^[42]	Alan Walker	3.38	December 3, 2015	
23.	" Girls Like You " ^[43]	Maroon 5	3.35	May 31, 2018	
24.	" Let Her Go " ^[44]	Passenger	3.35	July 25, 2012	
25.	" Waka Waka (This Time for Africa) " ^[45]	Shakira	3.34	June 4, 2010	
26.	" Perfect " ^[46]	Ed Sheeran	3.31	November 9, 2017	
27.	" Bailando " ^[47]	Enrique Iglesias	3.30	April 11, 2014	
28.	" Lean On " ^[48]	Major Lazer	3.29	March 22, 2015	
29.	" Humpty the train on a fruits ride " ^[49]	Kiddiestv Hindi – Nursery Rhymes & Kids Songs	3.24	January 26, 2018	
30.	" Shake It Off " ^[50]	Taylor Swift	3.23	August 18, 2014	

As of December 8, 2022

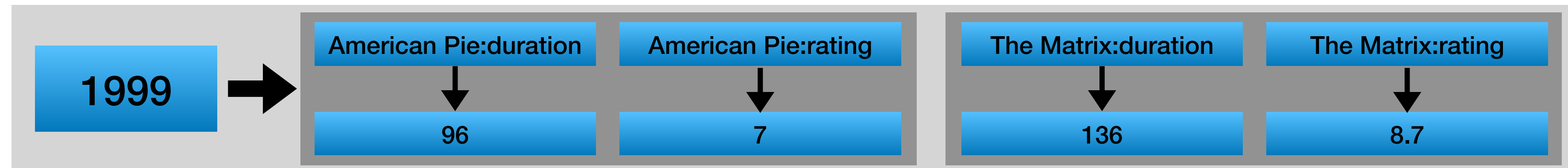
Top 30 most-viewed YouTube videos^[3]

No. ↕	Video name ↕	Uploader ↕	Views (billions) ↕	Publication date ↕	Note
1.	"Baby Shark Dance" ^[4]	Pinkfong Baby Shark - Kids' Songs & Stories	11.83	June 17, 2016	[A]
2.	 <p>The screenshot shows the YouTube interface for the video 'Baby Shark Dance'. At the top, there's a search bar and the YouTube logo. The video player shows a young boy in a blue and white patterned shirt and a white shark hat, dancing in a colorful, stylized underwater scene. A large blue shark is behind him, and there are smaller fish and coral in the background. The video title 'Baby Shark Dance #babys shark Most Viewed Video Animal Songs PINKFONG Songs for Children' is visible below the player. The channel name 'Pinkfong Baby Shark - Kids' Songs & St...' is also shown, along with a 'Subscribe' button and engagement icons for likes (38M), comments, shares, and downloads. A red box highlights the text '11B views 6 years ago'.</p>				
28.	"Lean On"	Major Lazer	3.25	March 22, 2015	
29.	"Humpty the train on a fruits ride" ^[49]	Kiddiestv Hindi – Nursery Rhymes & Kids Songs	3.24	January 26, 2018	
30.	"Shake It Off" ^[50]	Taylor Swift	3.23	August 18, 2014	

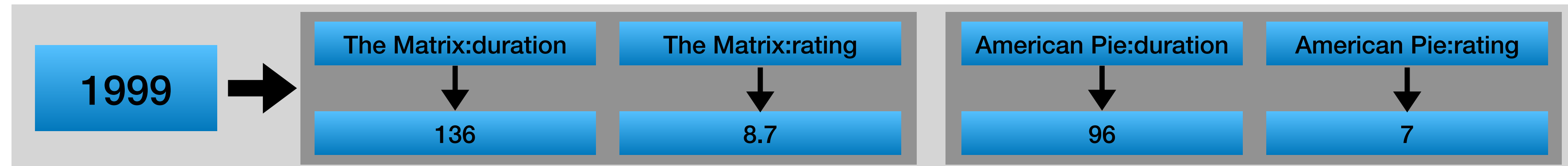
Clustering columns - ordering

- Default order is ascending (descending optional)

```
CREATE TABLE "movies" (  
  year          INT,  
  title         TEXT,  
  duration      INT,  
  rating        DOUBLE,  
  PRIMARY KEY ((year), title)  
);
```



```
CREATE TABLE "movies" (  
  year          INT,  
  title         TEXT,  
  duration      INT,  
  rating        DOUBLE,  
  PRIMARY KEY ((year), title)  
) WITH CLUSTERING ORDER BY (title DESC);
```

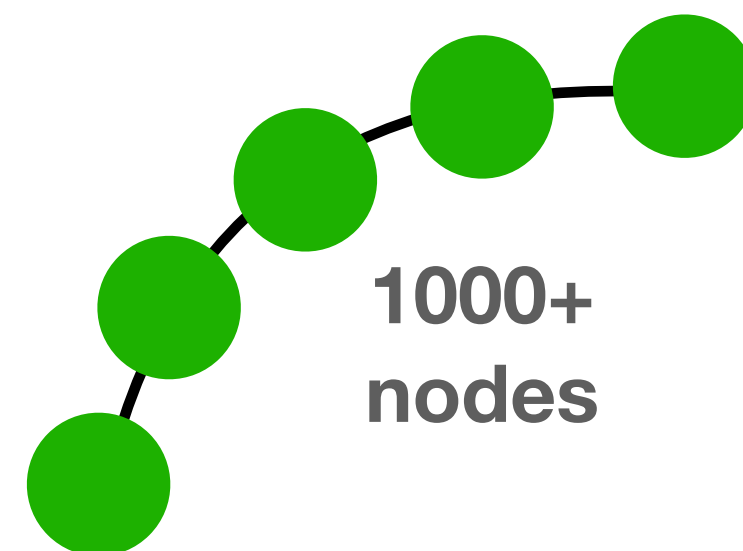
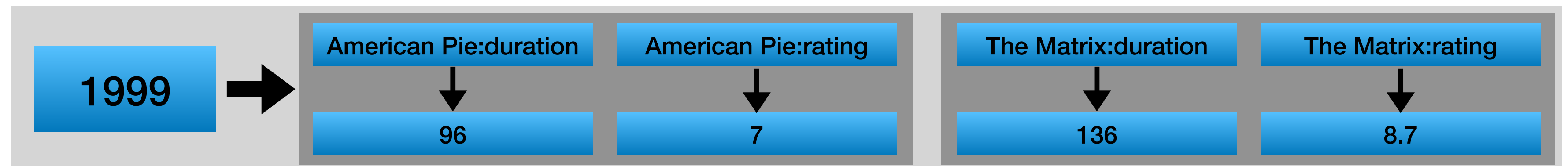


Clustering columns - queries

- Clustering columns are ordered —> **fast queries**

```
SELECT * FROM movies
WHERE year = 1999 AND
      title = "The Matrix"
```

```
CREATE TABLE "movies" (
  year      INT,
  title     TEXT,
  duration  INT,
  rating    DOUBLE,
  PRIMARY KEY ((year), title)
);
```

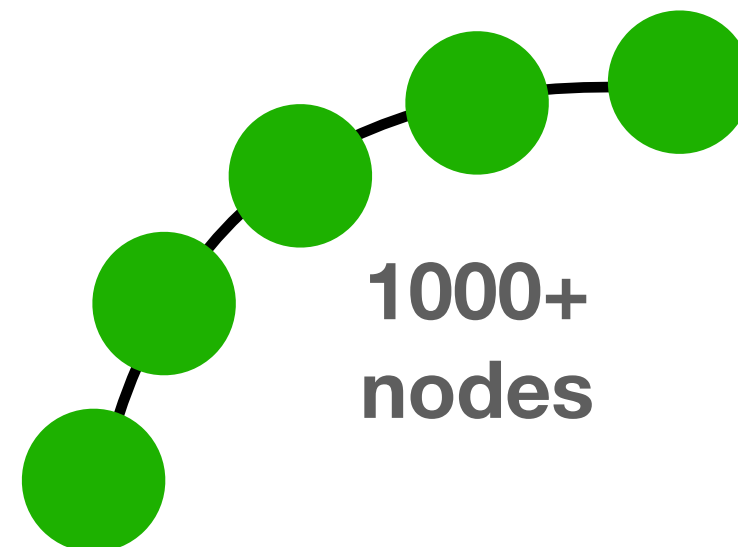
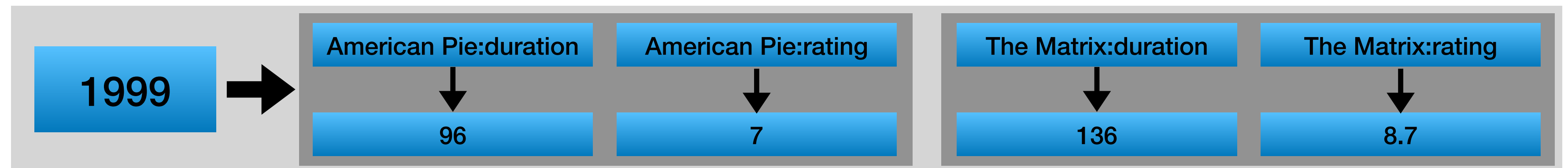


Clustering columns - range queries

- Lexicographic order for text

```
SELECT * FROM movies
WHERE year = 1999 AND
      title >= "The Matrix"
```

```
CREATE TABLE "movies" (
  year      INT,
  title     TEXT,
  duration  INT,
  rating    DOUBLE,
  PRIMARY KEY ((year), title)
);
```



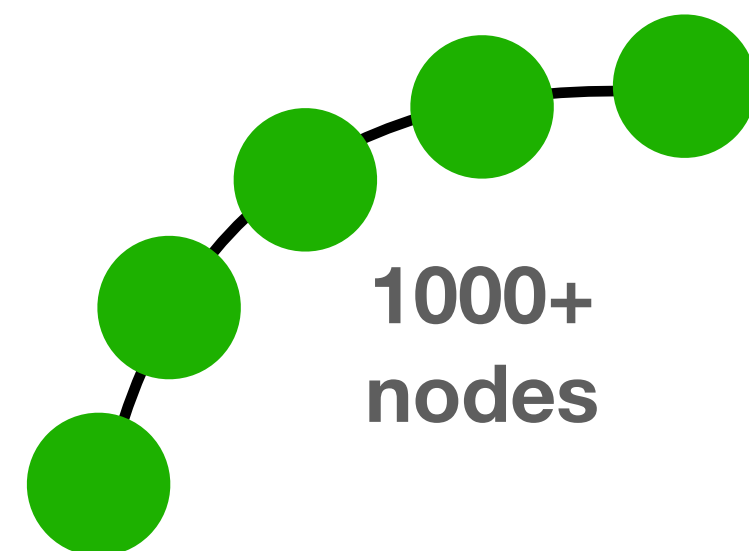
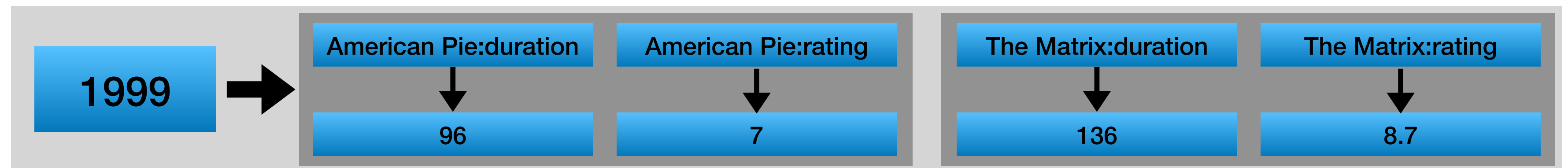
Clustering columns - range queries

- Lexicographic order for text

Can you think of an example for this query?

```
SELECT * FROM movies
WHERE year = 1999 AND
      title >= "The Matrix"
```

```
CREATE TABLE "movies" (
  year      INT,
  title     TEXT,
  duration  INT,
  rating    DOUBLE,
  PRIMARY KEY ((year), title)
);
```



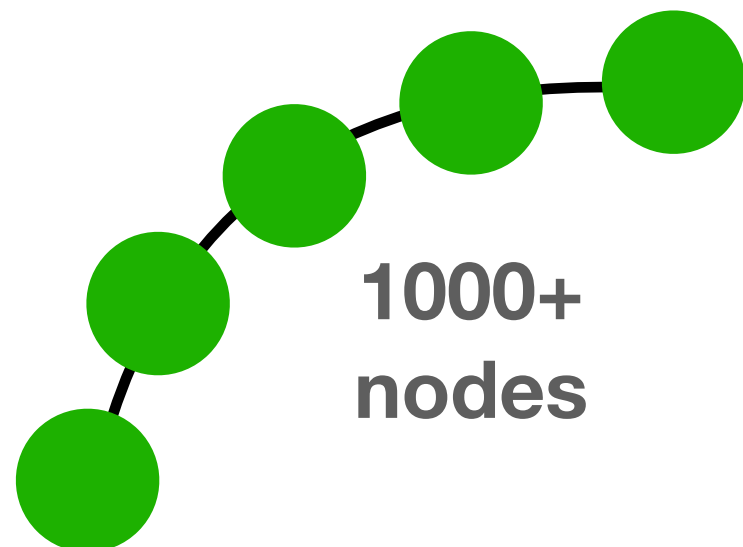
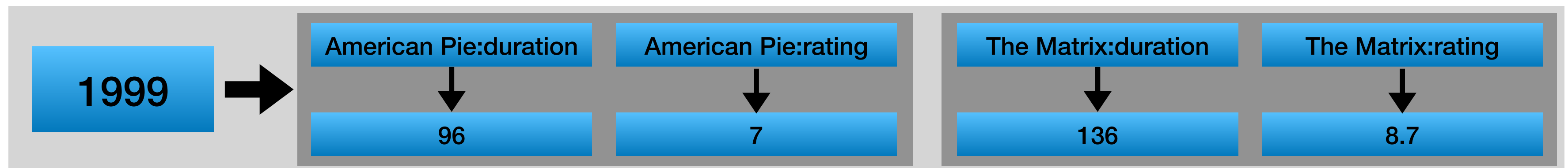
Clustering columns - range queries

- Lexicographic order for text

Can you think of an example for this query?
Showing movies by year with paging

```
SELECT * FROM movies
WHERE year = 1999 AND
      title >= "The Matrix"
```

```
CREATE TABLE "movies" (
  year      INT,
  title     TEXT,
  duration  INT,
  rating    DOUBLE,
  PRIMARY KEY ((year), title)
);
```

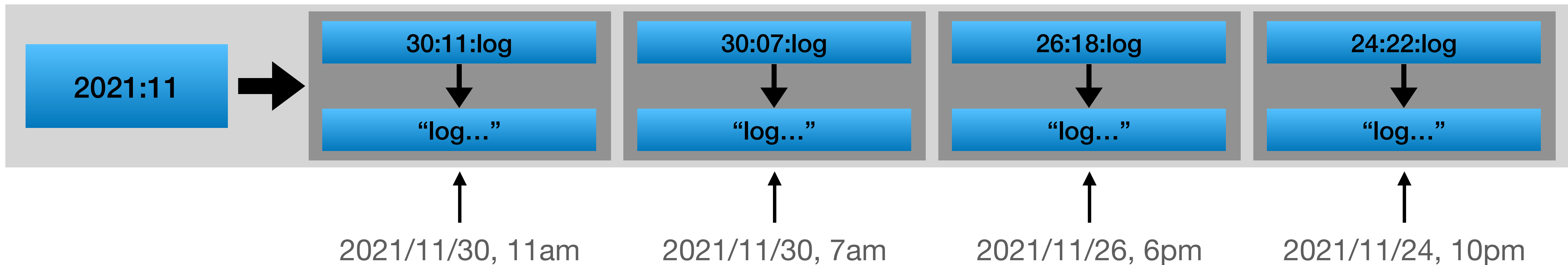


Clustering columns - more than one

```
CREATE TABLE "logs" (  
    year          INT,  
    month         INT,  
    day           INT,  
    hour          INT,  
    log           TEXT,  
    PRIMARY KEY ((year, month), day, hour)  
) WITH CLUSTERING ORDER BY (day DESC, hour DESC);
```


Clustering columns - more than one

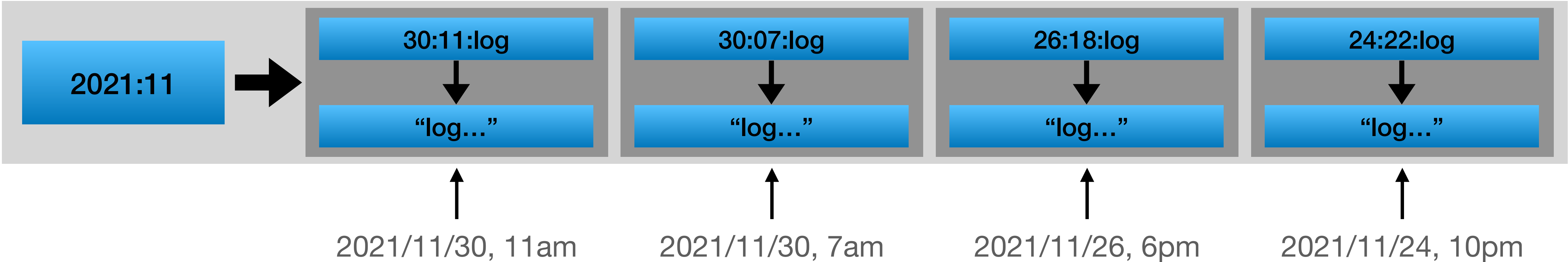
```
CREATE TABLE "logs" (  
  year          INT,  
  month        INT,  
  day          INT,  
  hour         INT,  
  log          TEXT,  
  PRIMARY KEY ((year, month), day, hour)  
) WITH CLUSTERING ORDER BY (day DESC, hour DESC);
```



Clustering columns - more than one

```
CREATE TABLE "logs" (  
  year          INT,  
  month        INT,  
  day          INT,  
  hour         INT,  
  log          TEXT,  
  PRIMARY KEY ((year, month), day, hour)  
) WITH CLUSTERING ORDER BY (day DESC, hour DESC);
```

When querying - WHERE needs to follow the order of the clustering columns



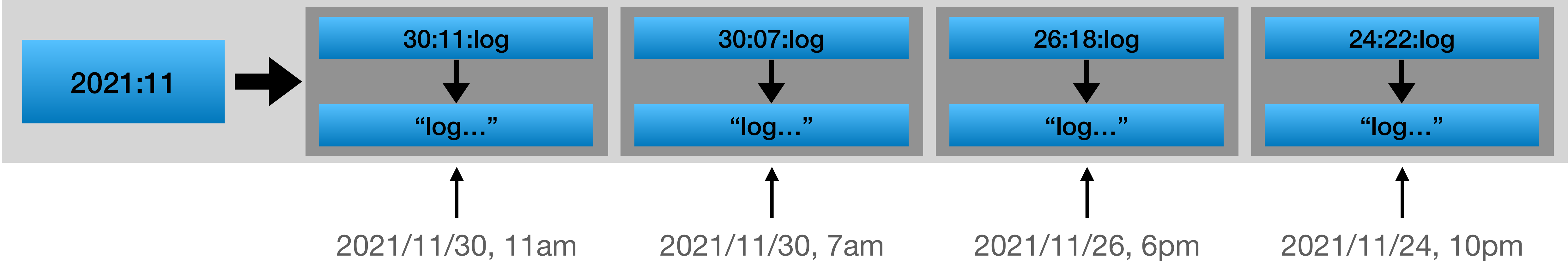
Clustering columns - more than one

```
CREATE TABLE "logs" (  
  year          INT,  
  month         INT,  
  day           INT,  
  hour          INT,  
  log           TEXT,  
  PRIMARY KEY ((year, month), day, hour)  
) WITH CLUSTERING ORDER BY (day DESC, hour DESC)
```

When querying - WHERE needs to follow the order of the clustering columns

Is this valid?

```
SELECT * FROM logs  
WHERE year = 2021 AND  
      month = 11 AND  
      hour = 18
```



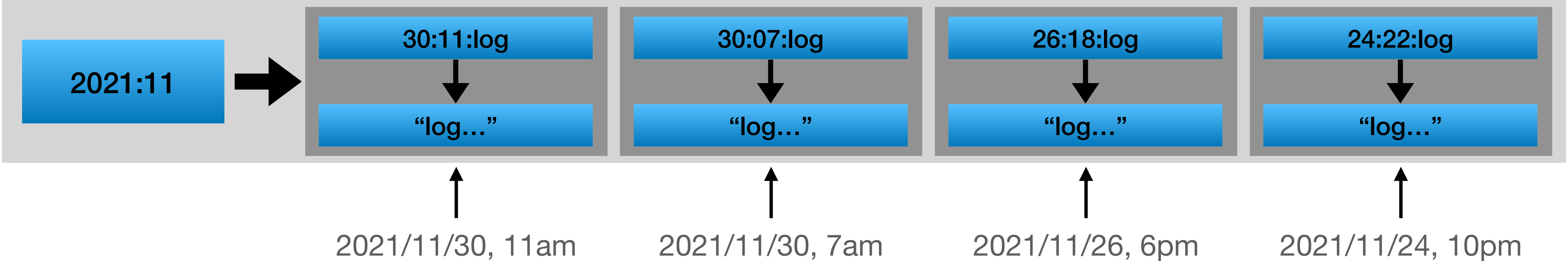
Clustering columns - more than one

```
CREATE TABLE "logs" (  
  year          INT,  
  month        INT,  
  day          INT,  
  hour         INT,  
  log          TEXT,  
  PRIMARY KEY ((year, month), day, hour)  
) WITH CLUSTERING ORDER BY (day DESC, hour DESC)
```

When querying - WHERE needs to follow the order of the clustering columns

Is this valid?
NO - day is required

```
SELECT * FROM logs  
WHERE year = 2021 AND  
      month = 11 AND  
      hour = 18
```



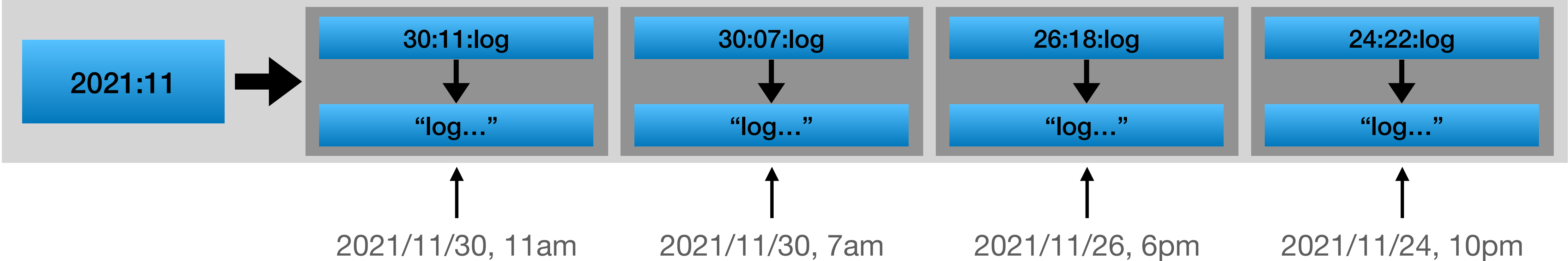
Clustering columns - more than one

```
CREATE TABLE "logs" (  
  year          INT,  
  month        INT,  
  day          INT,  
  hour         INT,  
  log          TEXT,  
  PRIMARY KEY ((year, month), day, hour)  
) WITH CLUSTERING ORDER BY (day DESC, hour DESC)
```

When querying - WHERE needs to follow the order of the clustering columns

Is this valid?

```
SELECT * FROM logs  
WHERE year = 2021 AND  
      month = 11 AND  
      day   = 26
```



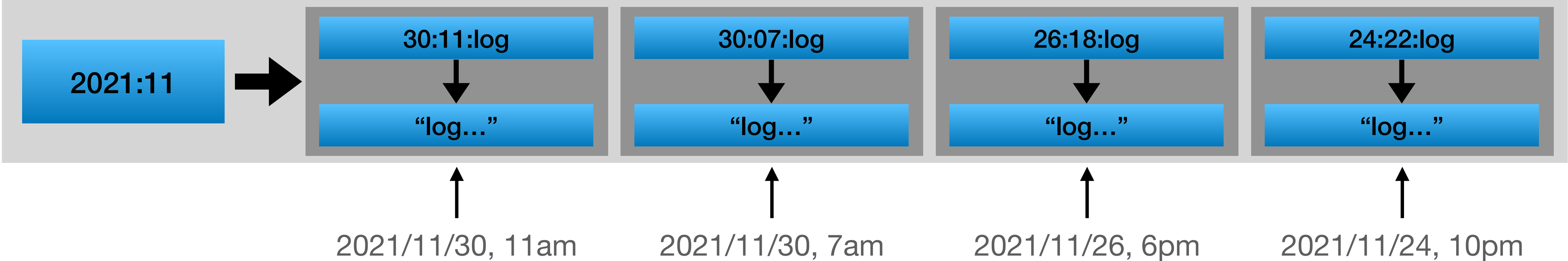
Clustering columns - more than one

```
CREATE TABLE "logs" (  
  year          INT,  
  month        INT,  
  day          INT,  
  hour         INT,  
  log          TEXT,  
  PRIMARY KEY ((year, month), day, hour)  
) WITH CLUSTERING ORDER BY (day DESC, hour DESC)
```

When querying - WHERE needs to follow the order of the clustering columns

Is this valid?
YES

```
SELECT * FROM logs  
WHERE year = 2021 AND  
      month = 11 AND  
      day   = 26
```



Agenda

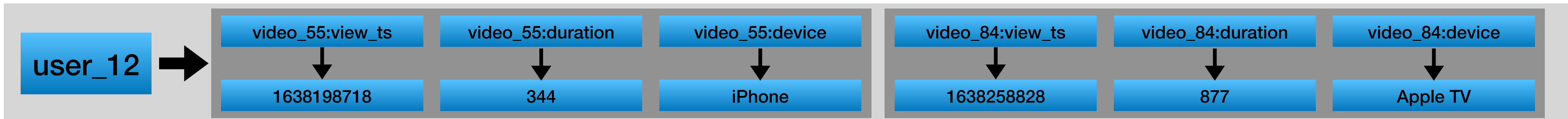
- History
- Architecture
- Data model (Original)
- Data model (CQL)
- **Examples**

Example (1)

user_id	video_id	video_ts	duration	device
user_12	video_55	1638198718	344	iPhone
user_12	video_84	1638258828	877	Apple TV

- Saving user viewing history

```
CREATE TABLE "user_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_ts     TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((user_id), video_id)  
);
```



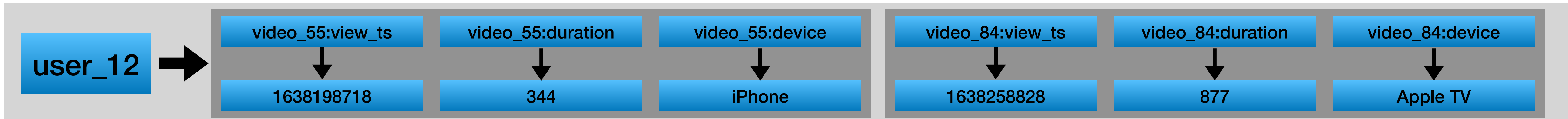
Example (1)

user_id	video_id	video_ts	duration	device
user_12	video_55	1638198718	344	iPhone
user_12	video_84	1638258828	877	Apple TV

- Saving user viewing history

How would you support saving more than one view per video?

```
CREATE TABLE "user_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_ts      TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((user_id), video_id)  
);
```

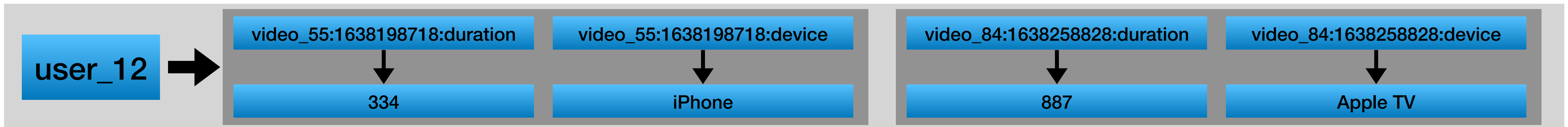


Example (1)

user_id	video_id	video_ts	duration	device
user_12	video_55	1638198718	344	iPhone
user_12	video_84	1638258828	877	Apple TV

- Saving user viewing history

```
CREATE TABLE "user_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_ts     TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((user_id), video_id, view_ts)  
);
```



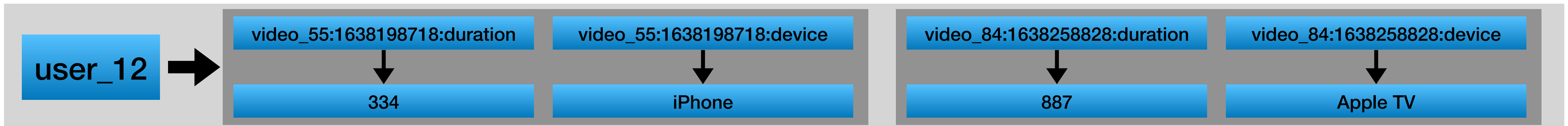
Example (1)

user_id	video_id	video_ts	duration	device
user_12	video_55	1638198718	344	iPhone
user_12	video_84	1638258828	877	Apple TV

- Saving user viewing history

What is the order the results are returned?

```
CREATE TABLE "user_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_ts     TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((user_id), video_id, view_ts)  
);
```



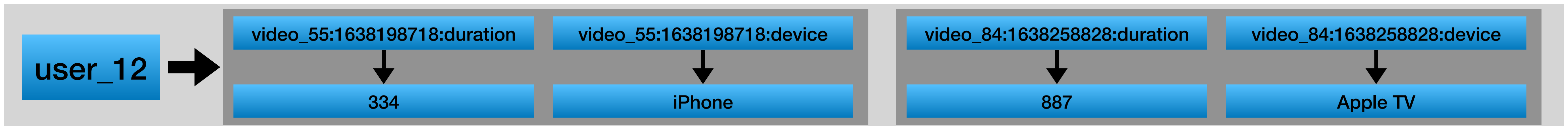
Example (1)

user_id	video_id	video_ts	duration	device
user_12	video_55	1638198718	344	iPhone
user_12	video_84	1638258828	877	Apple TV

- Saving user viewing history

How would you support getting the results by the view_ts and not by video_id?

```
CREATE TABLE "user_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_ts     TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((user_id), video_id, view_ts)  
);
```

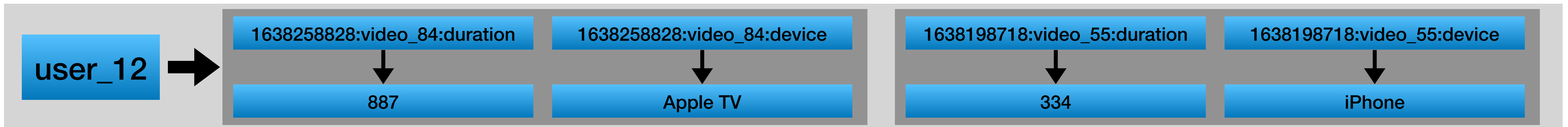


Example (1)

user_id	video_ts	video_id	duration	device
user_12	1638258828	video_84	877	Apple TV
user_12	1638198718	video_55	344	iPhone

- Saving user viewing history

```
CREATE TABLE "user_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_ts      TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((user_id), view_ts, video_id)  
) WITH CLUSTERING ORDER BY (view_ts DESC);
```

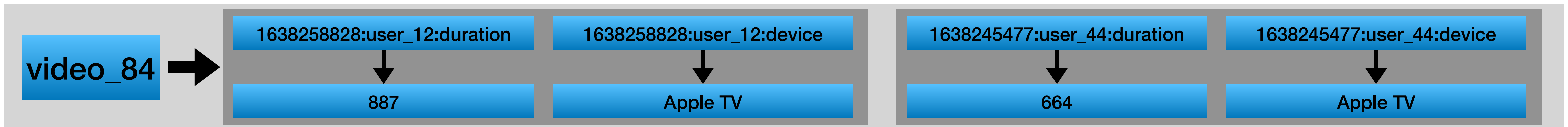


Example (2)

video_id	video_ts	user_id	duration	device
video_84	1638258828	user_12	877	Apple TV
video_84	1638245477	user_44	644	Apple TV

- Saving user viewing history for a video

```
CREATE TABLE "video_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_ts     TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((video_id), view_ts, user_id)  
) WITH CLUSTERING ORDER BY (view_ts DESC);
```



Example (2)

video_id	video_ts	user_id	duration	device
video_84	1638258828	user_12	877	Apple TV
video_84	1638245477	user_44	644	Apple TV

- Saving user viewing history for a video

For games of thrones this partition could be too big. What would you do?

```
CREATE TABLE "video_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_ts     TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((video_id), view_ts, user_id)  
) WITH CLUSTERING ORDER BY (view_ts DESC);
```



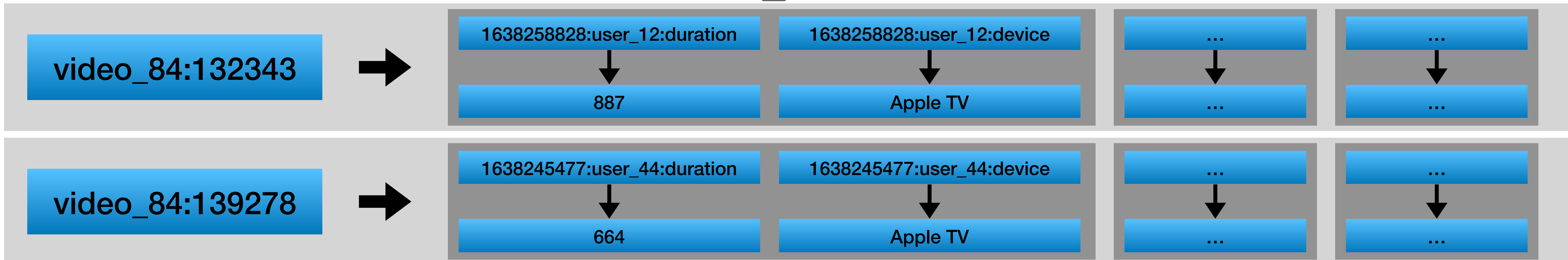
Example (2)

video_id	view_day	video_ts	user_id	duration	device
video_84	132343	1638258828	user_12	877	Apple TV
video_84	139278	1638245477	user_44	644	Apple TV

- Saving user viewing history for a video

```
CREATE TABLE "video_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_day     INT,  
  view_ts      TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((video_id, view_day), view_ts, user_id)  
) WITH CLUSTERING ORDER BY (view_ts DESC);
```

view_day = same as timestamp, just by days and not by milliseconds



Example (2)

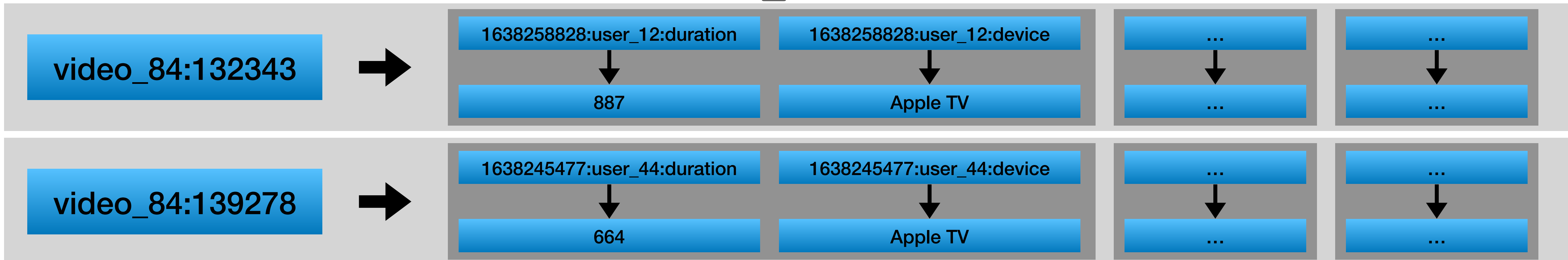
video_id	view_day	video_ts	user_id	duration	device
video_84	132343	1638258828	user_12	877	Apple TV
video_84	139278	1638245477	user_44	644	Apple TV

- Saving user viewing history for a video

```
CREATE TABLE "video_viewing_history" (  
  user_id      TEXT,  
  video_id     BIGINT,  
  view_day     INT,  
  view_ts      TIMESTAMP,  
  duration     INT,  
  device       TEXT,  
  PRIMARY KEY ((video_id, view_day), view_ts, user_id)  
) WITH CLUSTERING ORDER BY (view_ts DESC);
```

But what if 10m users watch a new episode on the day it is first released?

A lot more on "data modeling" later...

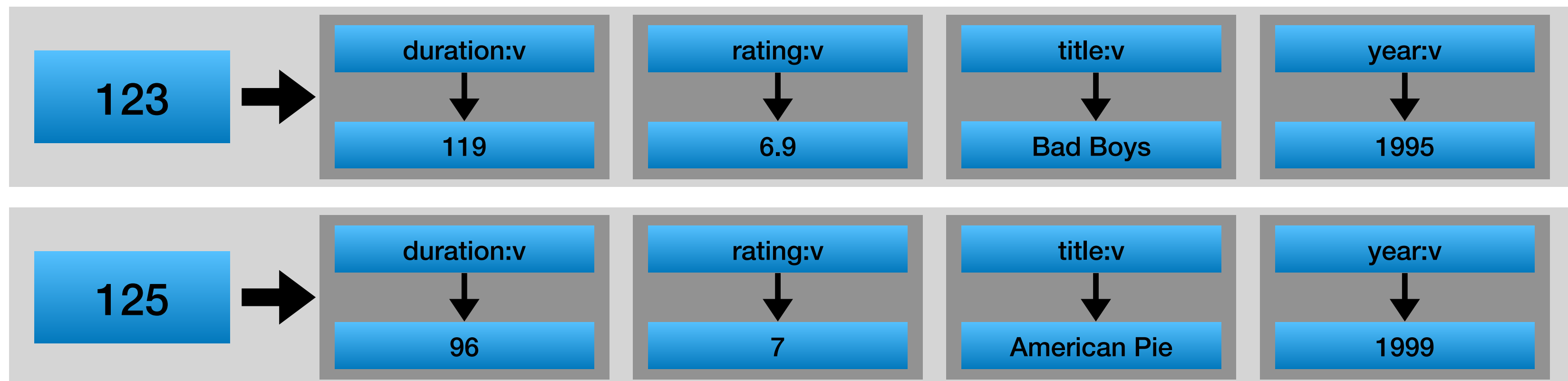


Example (3)

- Raw map of maps (old data model)

```
CREATE TABLE "test_map_of_maps" (  
  row_key      TEXT,  
  k            TEXT,  
  v            BLOB,  
  PRIMARY KEY (row_key, k)  
);
```

row_key	k	v
123	title	Bad Boys
123	year	1995
123	duration	119
123	rating	6.9
125	title	American Pie
125	year	1999
125	duration	96
125	rating	7



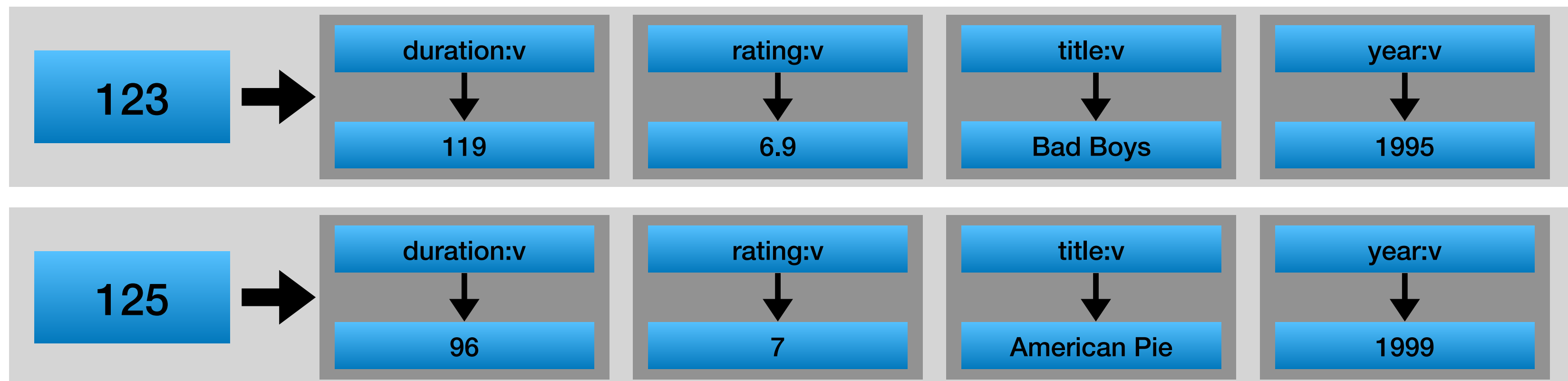
Example (3)

- Raw map of maps (old data model)

row_key	k	v
123	title	Bad Boys
123	year	1995
123	duration	119
123	rating	6.9
125	title	American Pie
125	year	1999
125	duration	96
125	rating	7

```
CREATE TABLE "test_map_of_maps" (  
  row_key      TEXT,  
  k            TEXT,  
  v            BLOB,  
  PRIMARY KEY (row_key, k)  
);
```

Can you simulate google's Bigtable model (that is, with versions)?



Example (3)

- Bigtable data model
without sorting on row keys

```
CREATE TABLE "test_map_of_maps" (  
  row_key      TEXT,  
  k            TEXT,  
  t            INT,  
  v            BLOB,  
  PRIMARY KEY (row_key, k, t)  
);
```

row_key	k	t	v
123	title	0	Bad Boys
123	year	0	1995
123	duration	0	119
123	rating	0	6.9
123	rating	1	7.2
123	rating	2	7.1
125	title	0	American
125	year	0	1999
125	duration	0	96
125	rating	0	7

t == version

