# CASSANDRA DATA MODELING STRATEGIES

Rohit Bhardwaj
Principal Cloud Engineer
rbhardwaj@kronos.com
Twitter: rbhardwaj1

# AGENDA

- What is Cassandra?
- Cassandra data Model
- Cassandra Query Language
- TTLs, Tomstones and counters
- Designing multiple dimensions
- Cassandra Modeling strategies

# ACID

- Atomicity: All operations are performed or none of them are. If one part of the transaction fails, then all fail.
- Consistency: The transaction must meet all rules defined by the system at all times; there are never any half-completed transactions.
- Isolation: Each transaction is independent unto itself.
- Durability: Once complete, the transaction cannot be undone.

- Would ACID work with Bigdata ?

# REPLICATION

```
SELECT COUNT(ArtifactID) FROM Document WHERE AccessControlListID_D IN (1,1000062) AND
(ArtifactID IN
 (SELECT ArtifactID FROM Document WHERE AccessControlListID_D IN (1,1000062)
  AND EXISTS
  (SELECT CodeArtifactID FROM CodeArtifact WHERE AssociatedArtifactID = Document.ArtifactID
   AND CodeArtifactID IN (17375543,17375544)
 ))
 OR ArtifactID IN
 (SELECT ArtifactID FROM Document WHERE AccessControlListID_D IN (1,1000062) AND
  (EXISTS
   (SELECT CodeArtifactID FROM CodeArtifact
    WHERE AssociatedArtifactID = Document.ArtifactID AND CodeArtifactID IN (13002091,13002080,17018689,13002017)
   )
   AND NOT EXISTS
   (SELECT CodeArtifactID FROM CodeArtifact WHERE AssociatedArtifactID = Document.ArtifactID
    AND CodeArtifactID IN (16851390,17018659)
  ))
))
```

# DB SHARDING

| Key | Name | Description | Stock | Price | LastOrdered |
|---|---|---|---|---|---|
| ARC1 | Arc welder | 250 Amps | 8 | 119.00 | 25-Nov-2013 |
| BRK8 | Bracket | 250mm | 46 | 5.66 | 18-Nov-2013 |
| BRK9 | Bracket | 400mm | 82 | 6.98 | 1-Jul-2013 |
| HOS8 | Hose | 1/2" | 27 | 27.50 | 18-Aug-2013 |
| WGT4 | Widget | Green | 16 | 13.99 | 3-Feb-2013 |
| WGT6 | Widget | Purple | 76 | 13.99 | 31-Mar-2013 |

| Key | Name | Description | Stock | Price | LastOrdered |
|---|---|---|---|---|---|
| ARC1 | Arc welder | 250 Amps | 8 | 119.00 | 25-Nov-2013 |
| BRK8 | Bracket | 250mm | 46 | 5.66 | 18-Nov-2013 |
| BRK9 | Bracket | 400mm | 82 | 6.98 | 1-Jul-2013 |

| Key | Name | Description | Stock | Price | LastOrdered |
|---|---|---|---|---|---|
| HOS8 | Hose | 1/2" | 27 | 27.50 | 18-Aug-2013 |
| WGT4 | Widget | Green | 16 | 13.99 | 3-Feb-2013 |
| WGT6 | Widget | Purple | 76 | 13.99 | 31-Mar-2013 |

# BASE

- Basically Available – the system guarantees some level of availability to the data even in regards to node failures. The data may be stale, but will still give and accept responses.

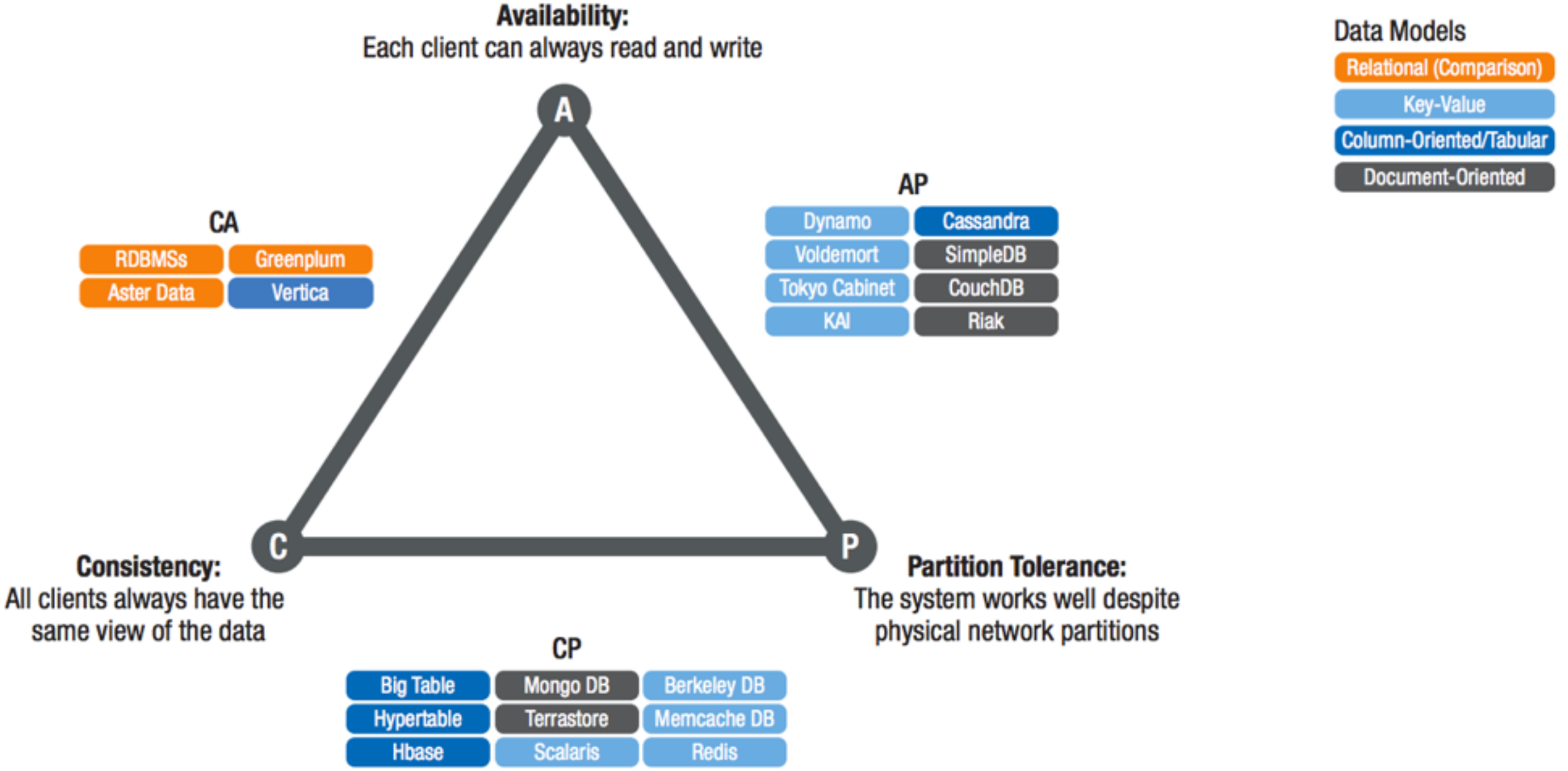
- Soft State – the data is in a constant state of flux; so, while a response may be given, the freshness or consistency of the data is not guaranteed to be the most current.

- Eventual Consistency – the data will eventually be consistent through all nodes and in all databases, but not every transaction at every moment. It will reach some guaranteed state eventually.

| *ACID* | *BASE* |
| --- | --- |
| • Strong Consistency | • Weak Consistency – stale data OK |
| • Isolation | • Availability first |
| • Focus on "commit" | • Best effort |
| • Nested transactions | • Approximate answers OK |
| • Availability? | • Aggressive (optimistic) |
| • Conservative (pessimistic) | • Simpler! |
| • Difficult evolution (e.g. schema) | • Faster |
| | • Easier evolution |

| AVAILABILITY LEVEL[11] | ESTIMATED DOWNTIME PER YEAR |
|---|---|
| 99.999% | 5 minutes |
| 99.99% | 52 minutes |
| 99.9% | 8.5 hours |
| 99% | 3.5 days or about 87 hours |

# CAP THEOREM

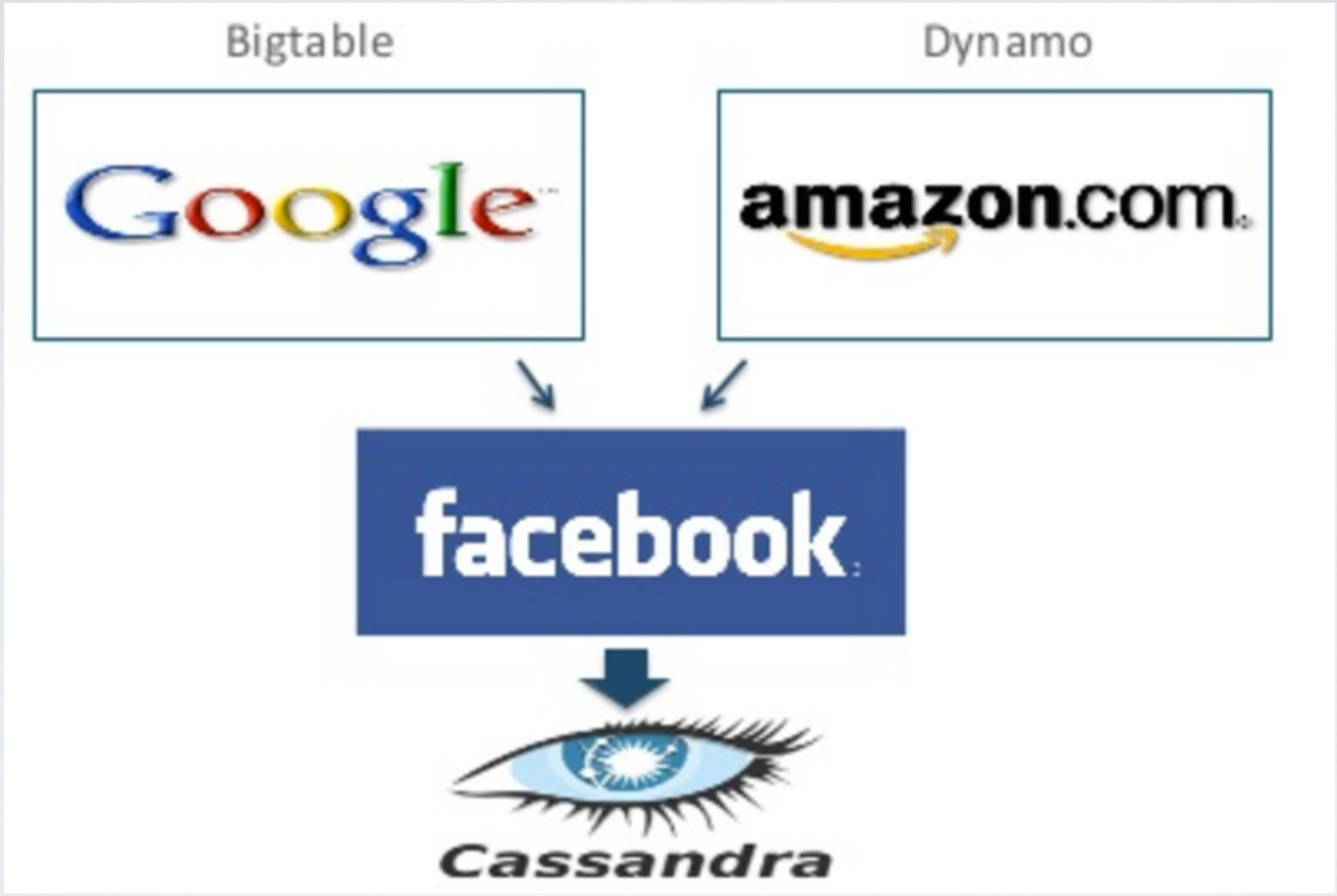


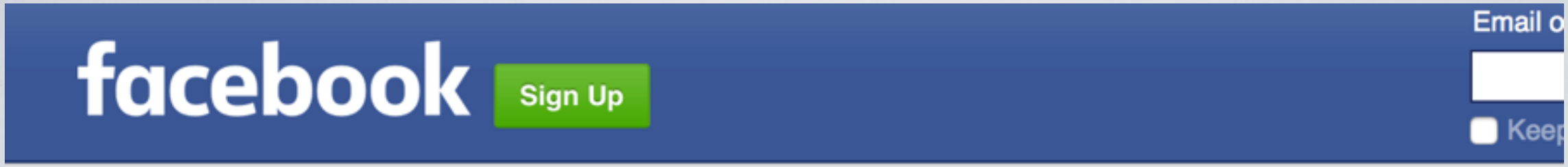

**BASE (Basically Available, Soft-State, Eventual Consistency) data store**

| SELECTION | CHARACTERISTICS | EXAMPLES |
| --- | --- | --- |
| C + A (No P) | 2-phase commits<br>Cache validation protocols | Single-site databases<br>Cluster databases<br>LDAP<br>xFS file system |
| C + P (no A) | Pessimistic locking Make minority partitions unavailable | Distributed databases<br>Distributed locking<br>Majority protocols |
| A + P (no C) | Expirations/leases<br>Conflict resolution<br>Optimistic | Coda<br>Web caching<br>DNS |

# Cassandra

A highly scalable, eventually consistent, distributed, structured key-value store.


Bigtable
Dynamo
Google
amazon.com
facebook
Cassandra




# Cassandra – A structured storage system on a P2P Network

By Avinash Lakshman on Monday, August 25, 2008 at 4:31pm

When I joined Facebook I was eagerly looking forward to a new challenge. Fortunately, Facebook cannot be accused of a lack of challenging assignments. Prashant Malik,a colleague in Facebook from the Search team, was thinking about how to solve the Inbox Search problem. This challenge is about storing reverse indices of Facebook messages that Facebook users send and receive while communicating with their friends on the Facebook network. The amount of data to be stored, the rate of growth of the data and the requirement to serve it within strict SLAs made it very apparent that a new storage solution was absolutely essential. The solution needed to scale incrementally and in a cost effective fashion. Traditional data storage solutions just wouldn't fit the bill. The aim was to design a solution that not only solved the Inbox Search problem but also provided a system as a storage infrastructure for many problems of the same nature. Hence was born Cassandra. To keep up with Facebook tradition, Prashant and I started the implementation of Cassandra about a year ago in one of our Hackthons.

https://www.facebook.com/notes/facebook-engineering/cassandra-a-structured-storage-system-on-a-p2p-network/24413138919/

www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

# Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall
and Werner Vogels

Amazon.com

## ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

# Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

*Google, Inc.*

cassandra.apache.org

# Cassandra

Home | Download | Getting Started | Planet Cassandra | Contribute

Welcome | Video | Slides

## Welcome to Apache Cassandra ™

The Apache Cassandra database is the right choice when you need scalability and high availability without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data. Cassandra's support for replicating across multiple datacenters is best-in-class, providing lower latency for your users and the peace of mind of knowing that you can survive regional outages.

Cassandra's data model offers the convenience of column indexes with the performance of log-structured updates, strong support for denormalization and materialized views, and powerful built-in caching.

**Download**

Tick-Tock release **3.4** (Changes)
3.0.x release **3.0.4** (Changes)
2.2.x release **2.2.5** (Changes)

Download options

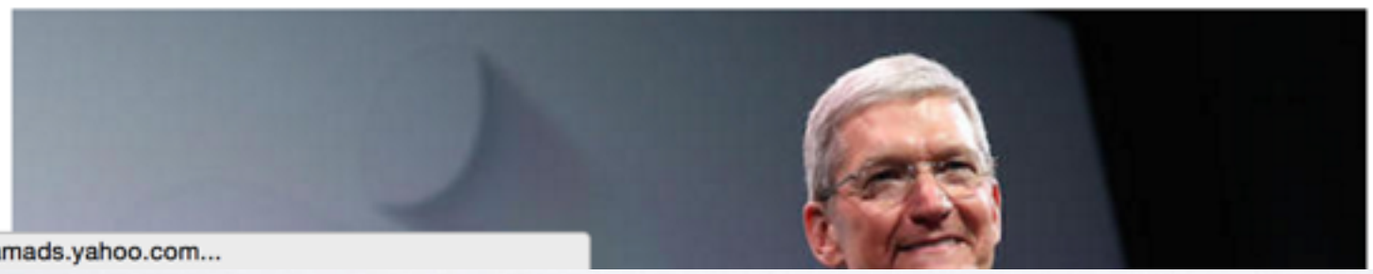
SOFTWARE

# Apple's secret NoSQL sauce includes a hefty dose of Cassandra

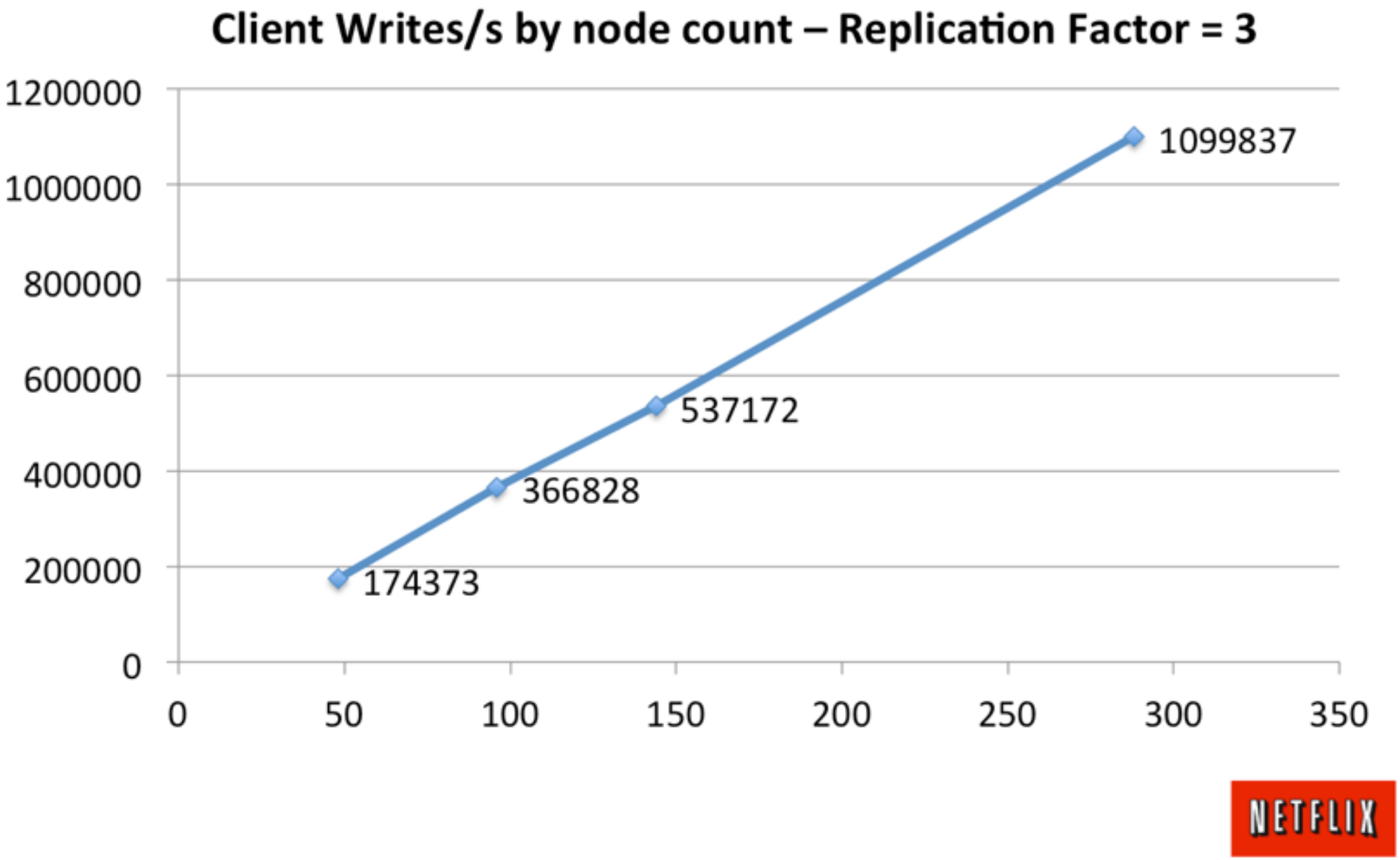
If you want to scale like Apple, you might need to consider Cassandra. Matt Asay explains.

By Matt Asay | September 16, 2015, 6:49 AM PST





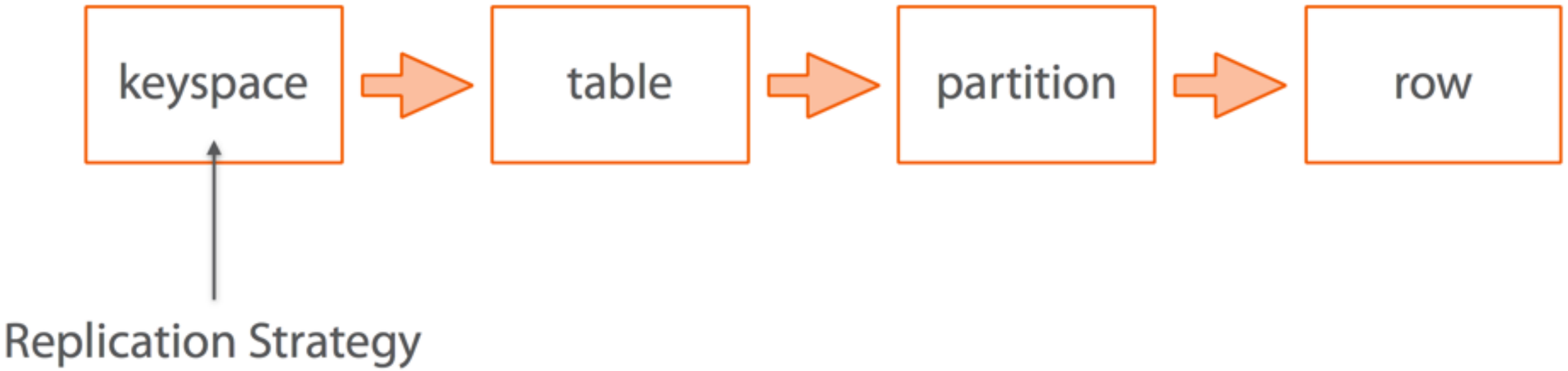
# Scale-Up Linearity




NETFLIX

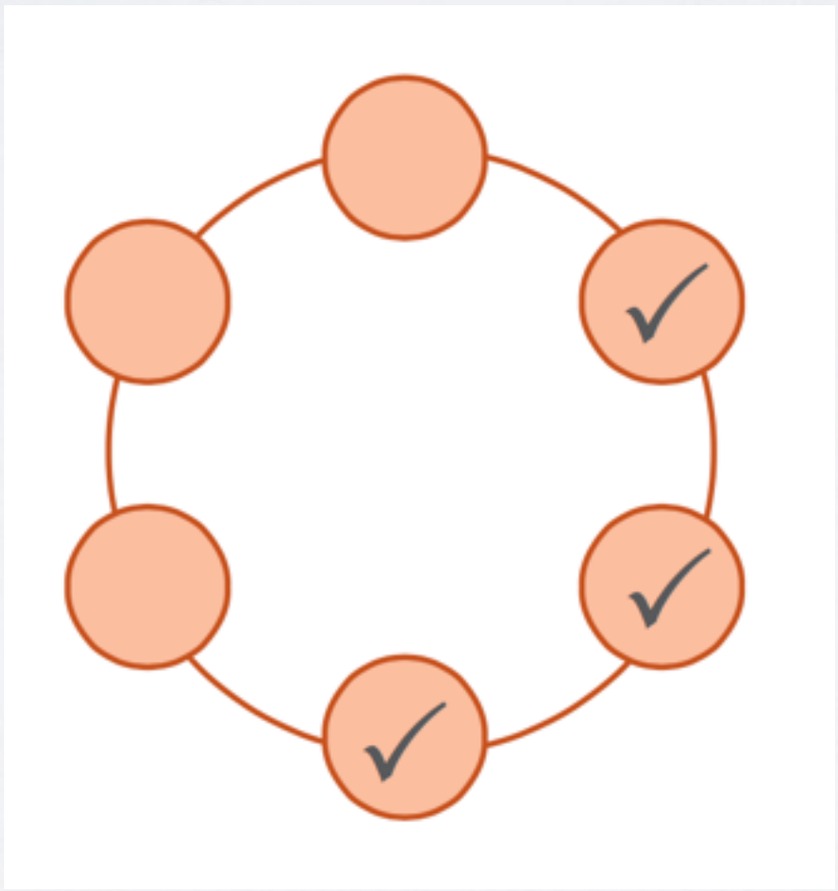
http://image.slidesharecdn.com/cassandraeu1-120330040410-phpapp01/95/cassandra-eu-2012-netflixs-cassandra-architecture-and-open-source-efforts-5-728.jpg?cb=1333080333

# CASSANDRA TERMINOLOGY



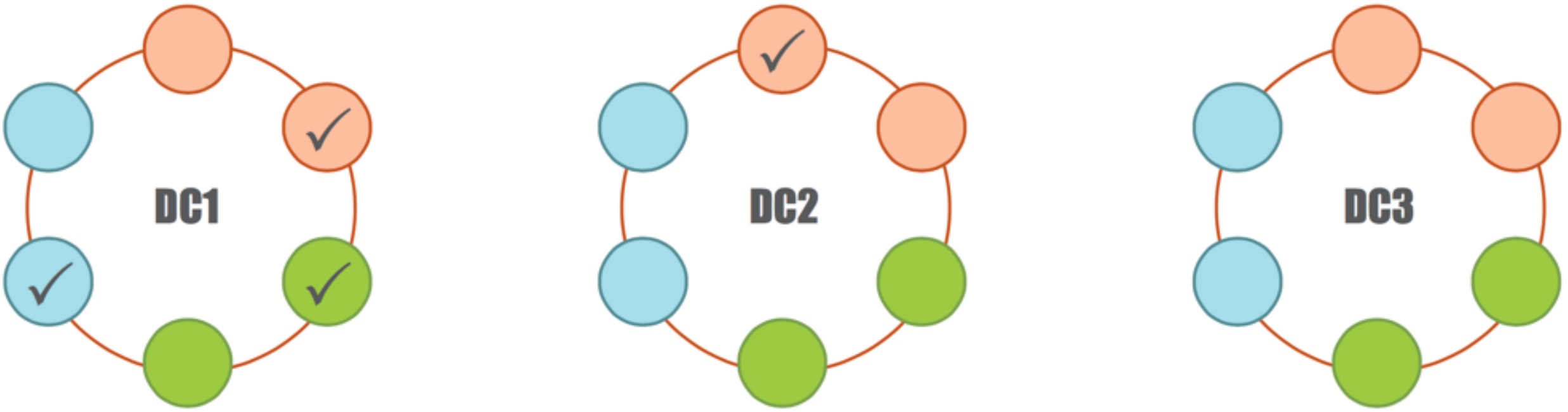
# SIMPLE STRATEGY

- CREATE KEYSPACE nfjs WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };

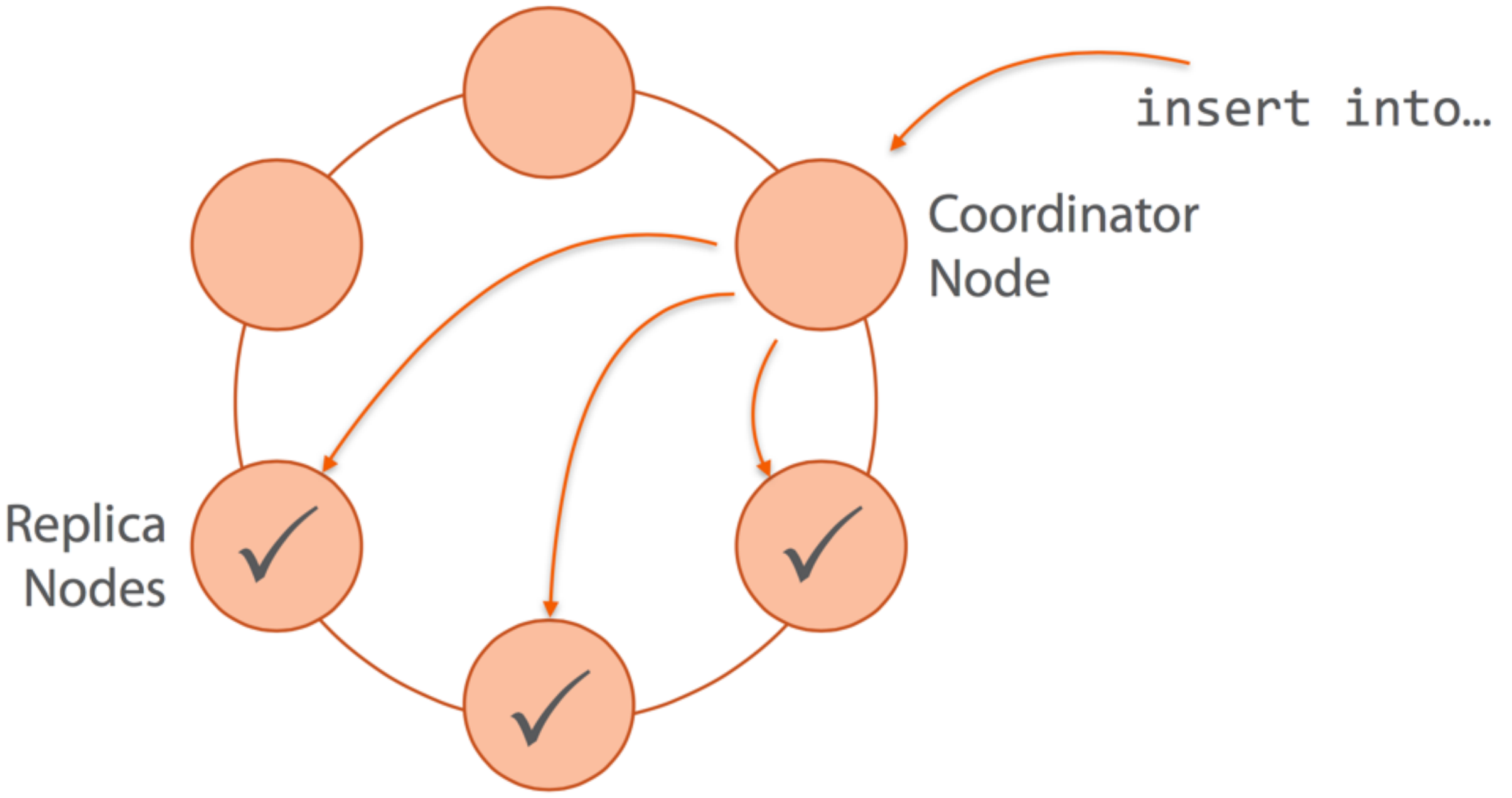


# NETWORK TOPOLOGY STRATEGY

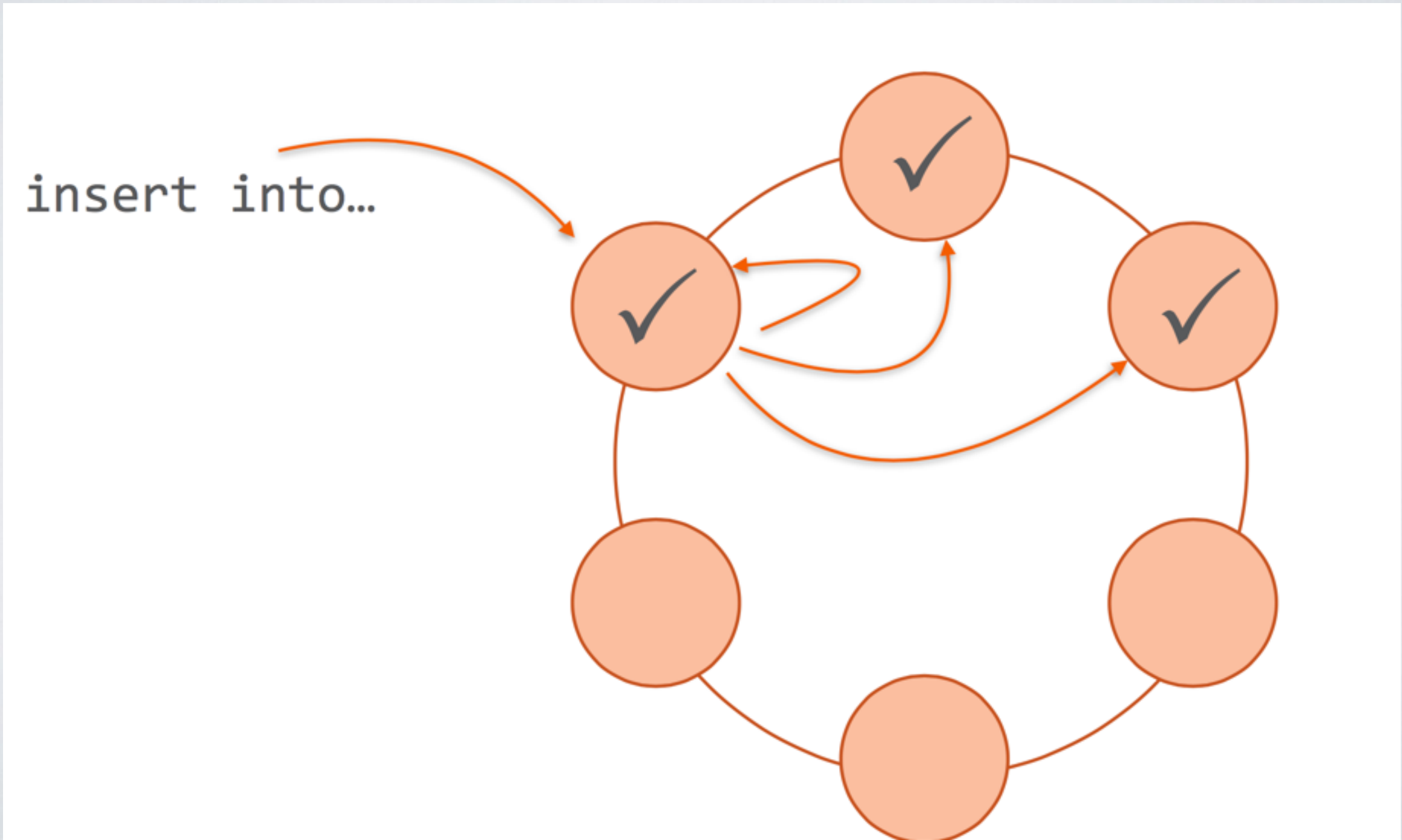
- create KEYSPACE nfjs WITH REPLICATION = {'class': 'NetworkTopologyStrategy', 'DC1': 3, 'DC2', 1};

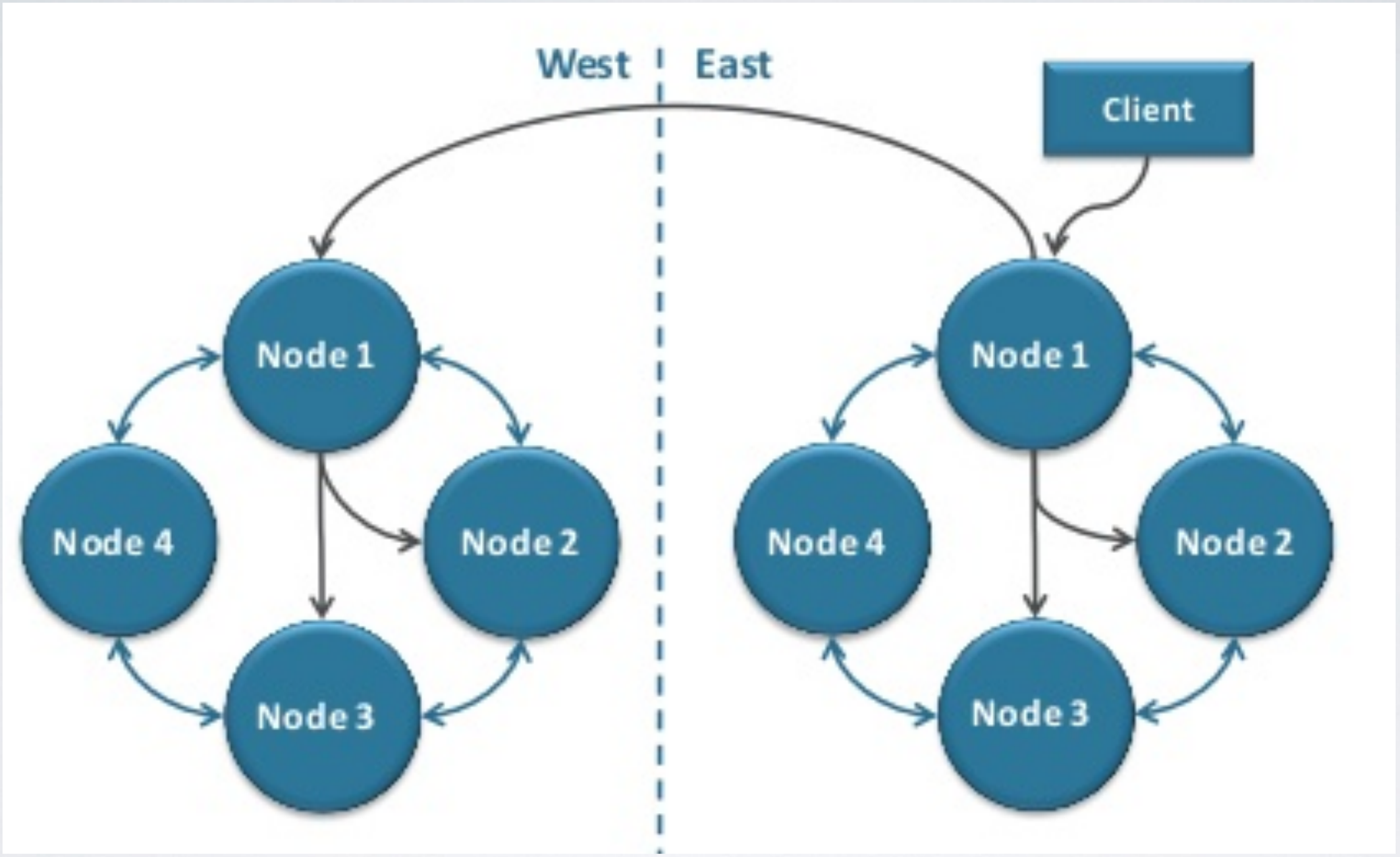


# READS AND WRITES IN CASSANDRA



# READS AND WRITES IN CASSANDRA

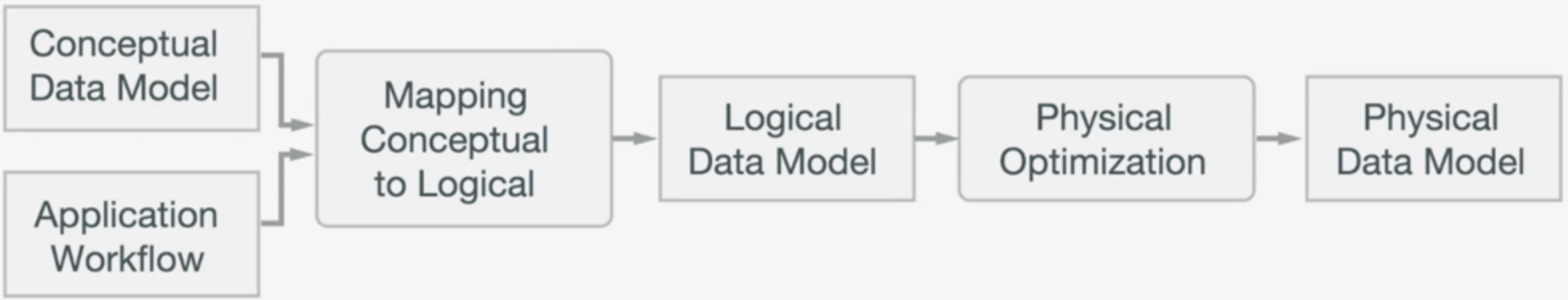



West
East
Client
Node 1
Node 4
Node 2
Node 3
Node 1
Node 4
Node 2
Node 3

# CASSANDRA DEMO

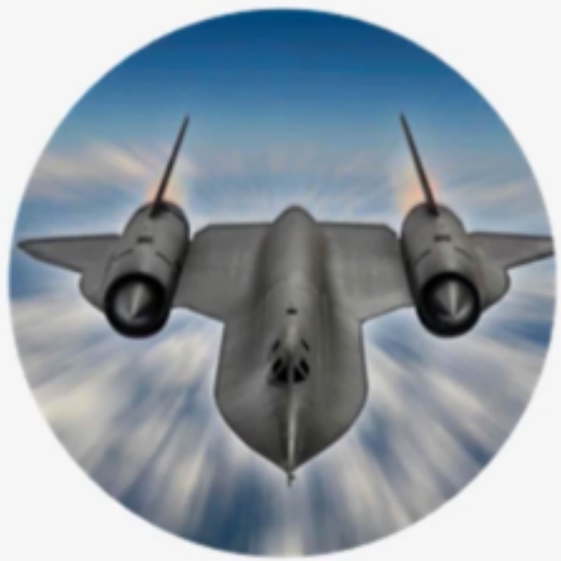
- http://www.planetcassandra.org/try-cassandra/

# CASSANDRA DATA MODEL

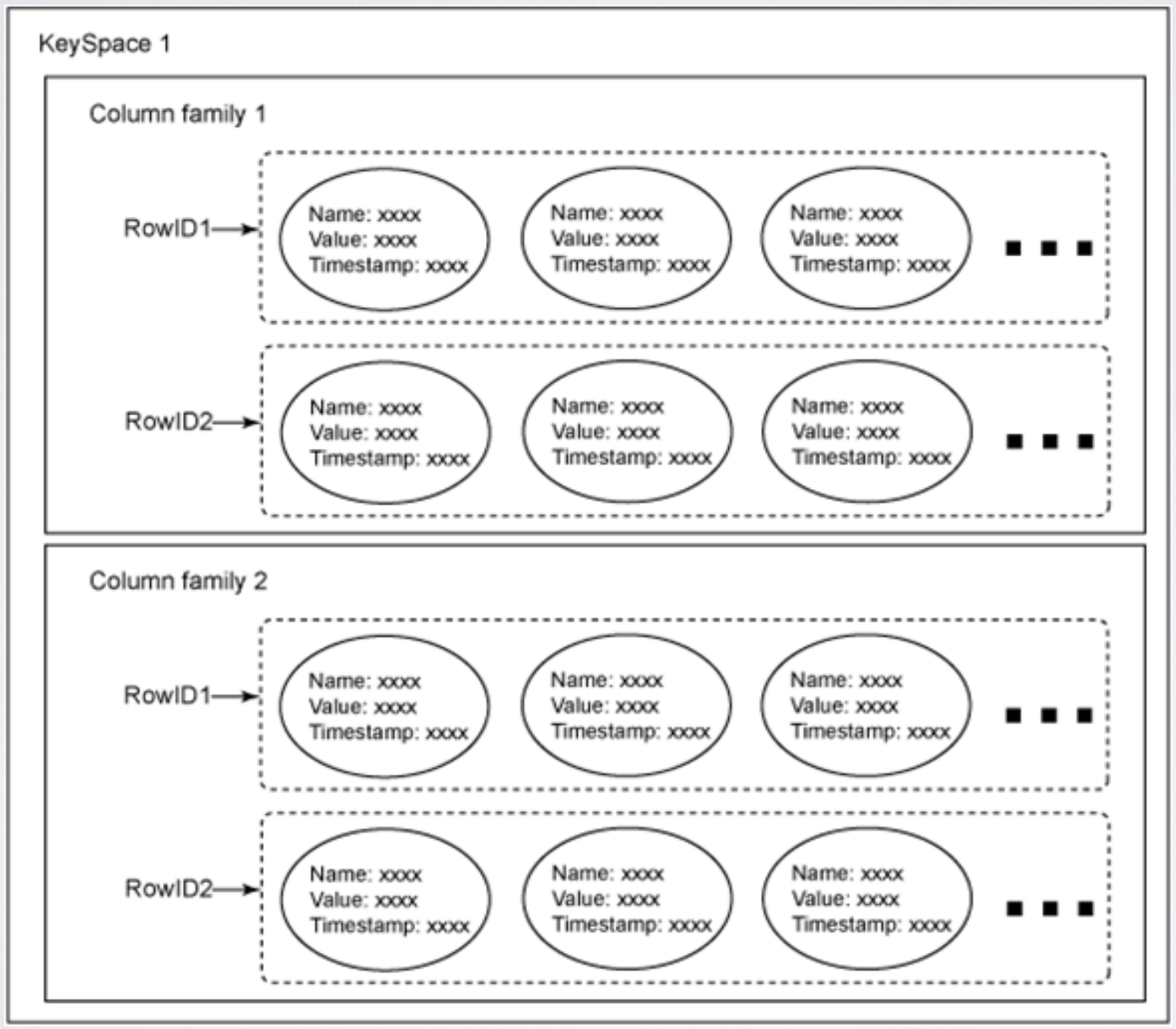


# QUERIES



# KEY-VALUE MODEL



Column Name

**Column Family: Tweets**

| | | | |
|---|---|---|---|
| 1234e530-8b82-11df | Text | User_ID | Date |
| | Hello, World! | 39823 | 2009-03-25T19:20:30 |

| | | | |
|---|---|---|---|
| 22615e20-8b82-11df | Text | User_ID | Date |
| | Gooooal! | 592 | 2009-03-25T19:25:43 |

Key

Column Value

# COLUMN NAMES CAN STORE VALUES

**Column Family: User_Timelines**

| | | | |
|---|---|---|---|
| 39823 | cef7be80-8b88-11df | 1234e530-8b82-11df | ... |
| | – | – | ... |
| 592 | f0137940-8b8a-11df | 22615e20-8b82-11df | ... |
| | – | – | ... |

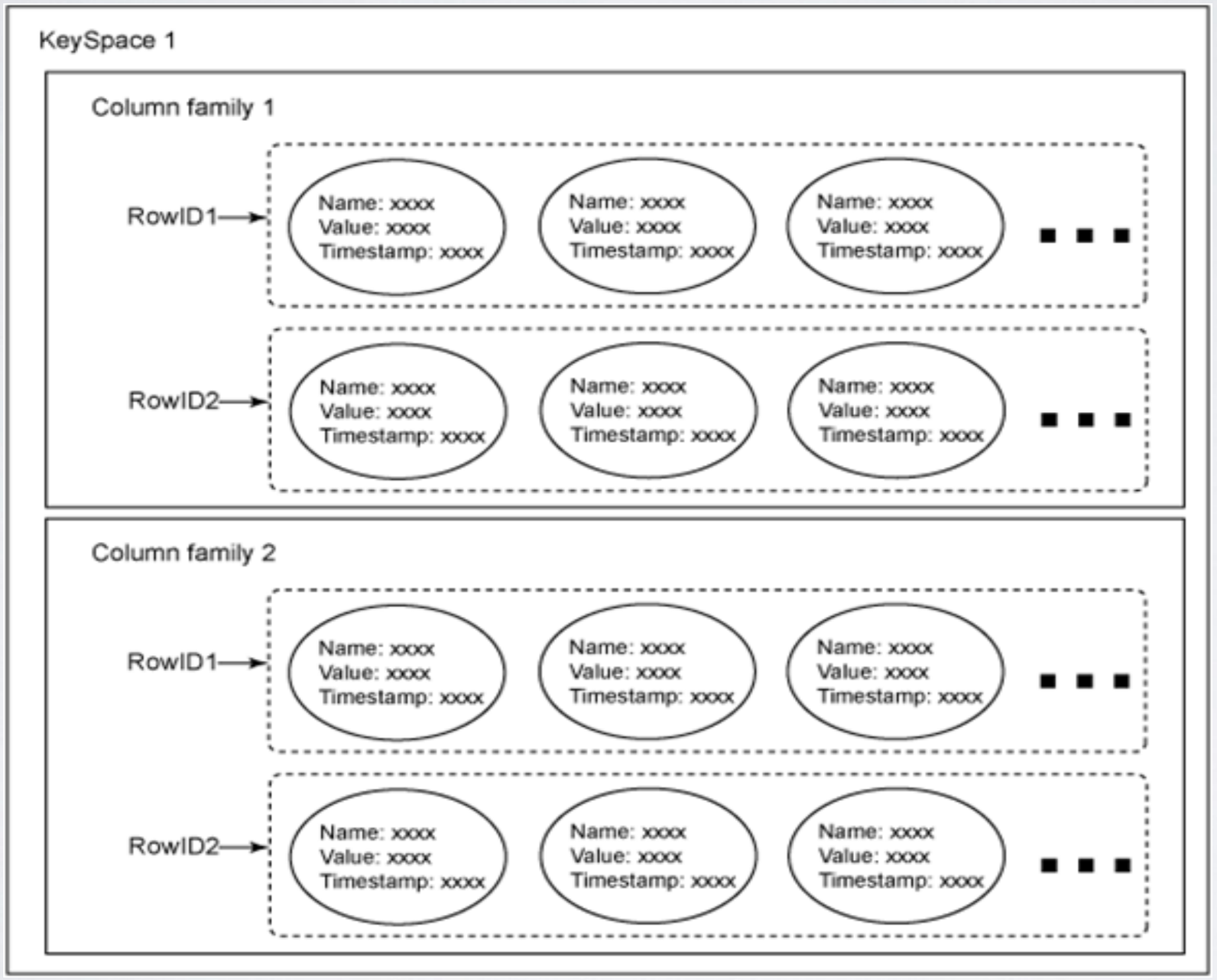
• • •

# UUID AND TIMEUUID

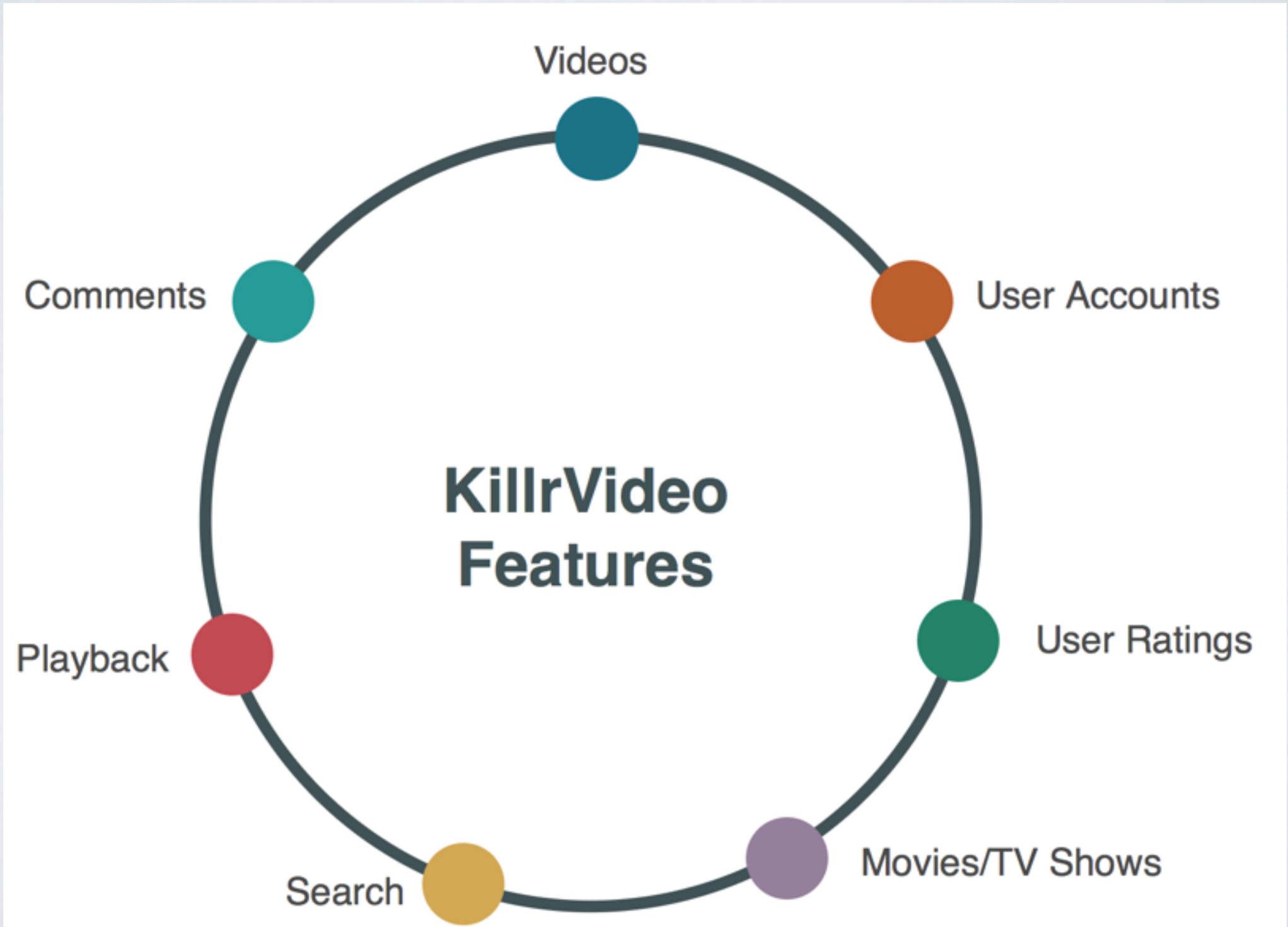
- Universally Unique Identifier
  - Ex: 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328
  - Generate via uuid()

- TIMEUUID embeds a TIMESTAMP value
  - Ex: 1be43390-9fe4-11e3-8d05-425861b86ab6
  - Sortable
  - Generate via now()

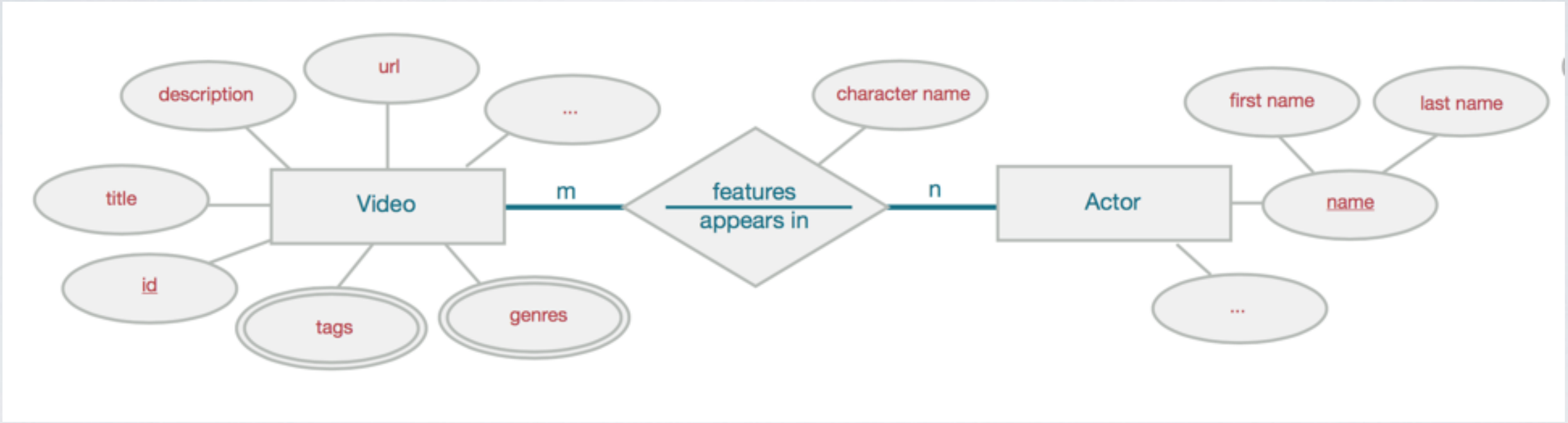
# KEYSPACE


KeySpace 1
Column family 1
RowID1
Name: xxxx
Value: xxxx
Timestamp: xxxx
Name: xxxx
Value: xxxx
Timestamp: xxxx
Name: xxxx
Value: xxxx
Timestamp: xxxx
RowID2
Name: xxxx
Value: xxxx
Timestamp: xxxx
Name: xxxx
Value: xxxx
Timestamp: xxxx
Name: xxxx
Value: xxxx
Timestamp: xxxx
Column family 2
RowID1
Name: xxxx
Value: xxxx
Timestamp: xxxx
Name: xxxx
Value: xxxx
Timestamp: xxxx
Name: xxxx
Value: xxxx
Timestamp: xxxx
RowID2
Name: xxxx
Value: xxxx
Timestamp: xxxx
Name: xxxx
Value: xxxx
Timestamp: xxxx
Name: xxxx
Value: xxxx
Timestamp: xxxx

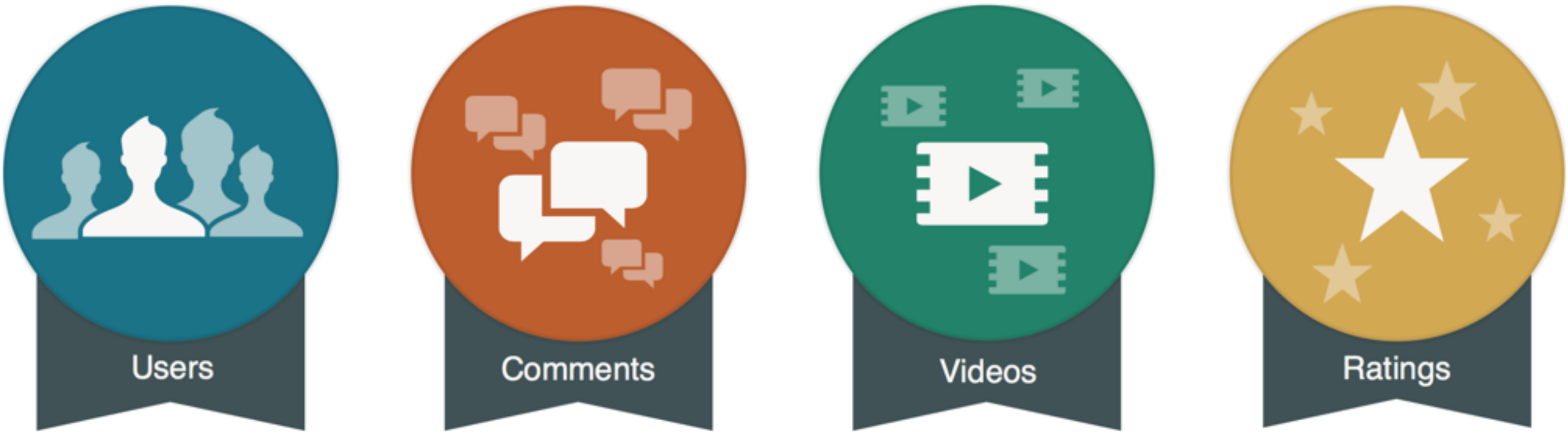
# KILLRVIDEO.COM

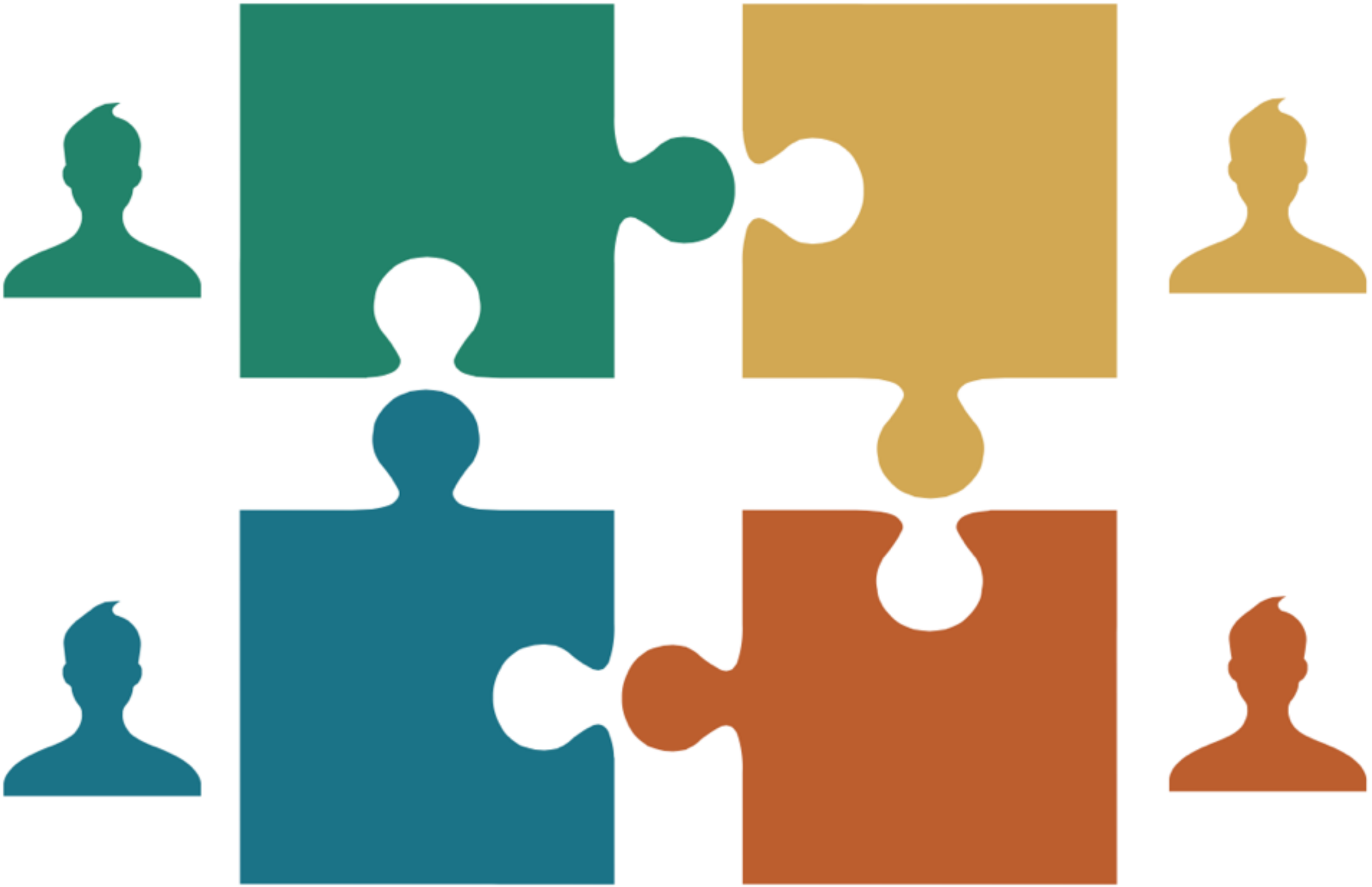


# CONCEPTUAL DATA MODELING



# PURPOSE

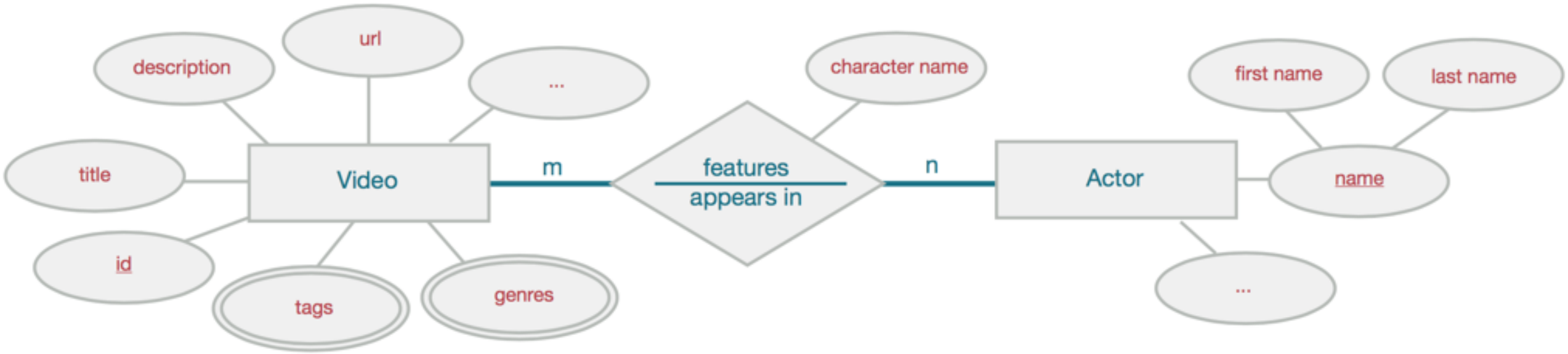




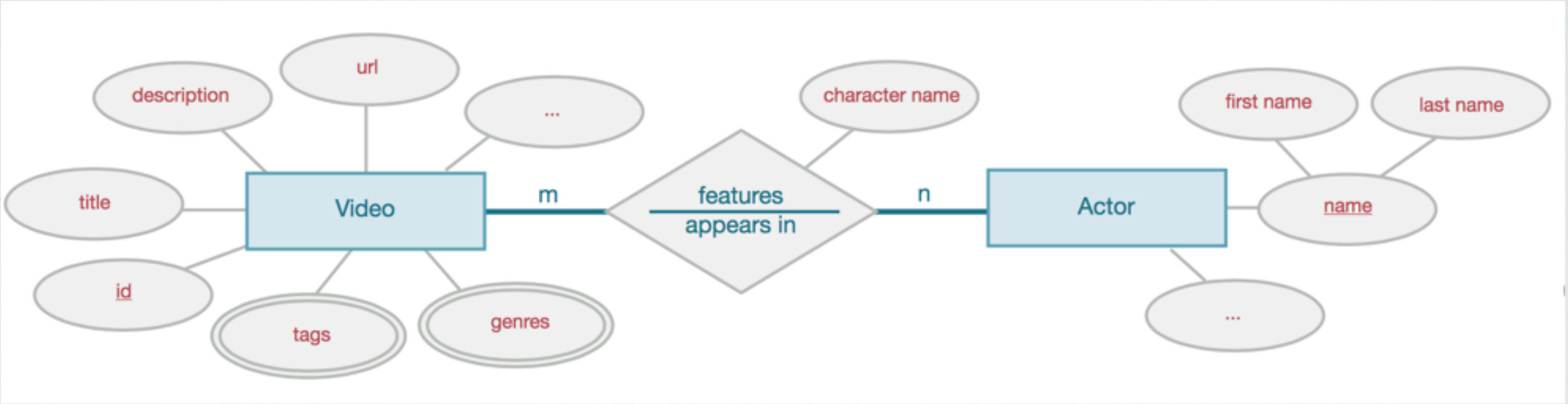
# ABSTRACTION

# ABSTRACTION

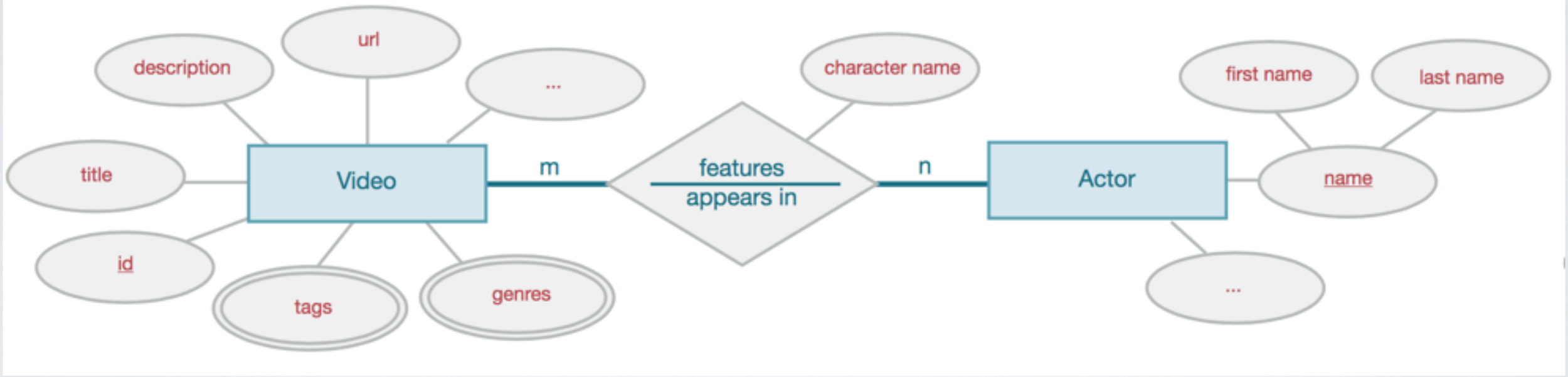
# ENTITY-RELATIONSHIP (ER) MODEL



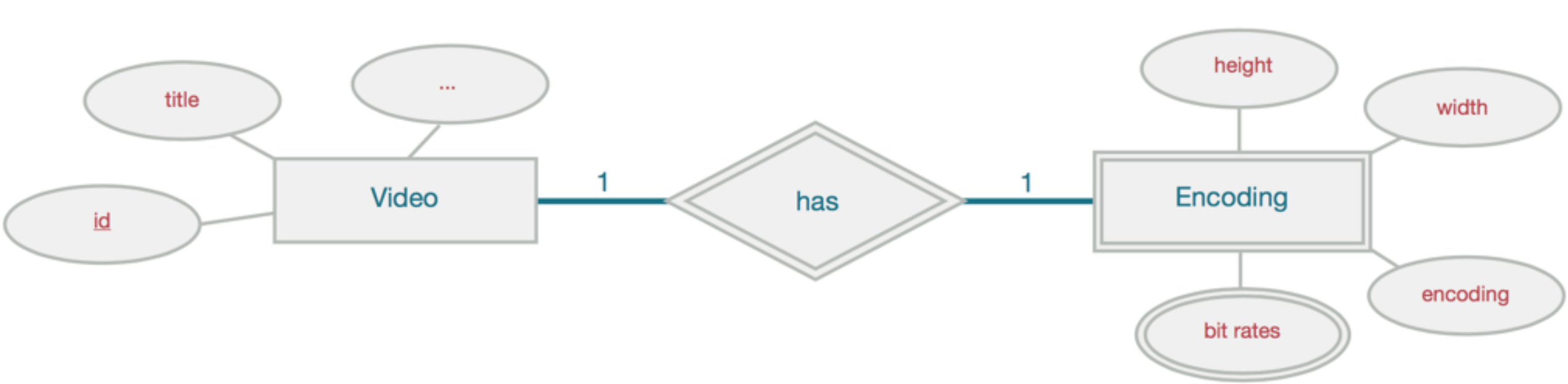
# ENTITY TYPE


description
url
...
character name
first name
last name
title
Video
m
features
appears in
n
Actor
name
id
tags
genres
...

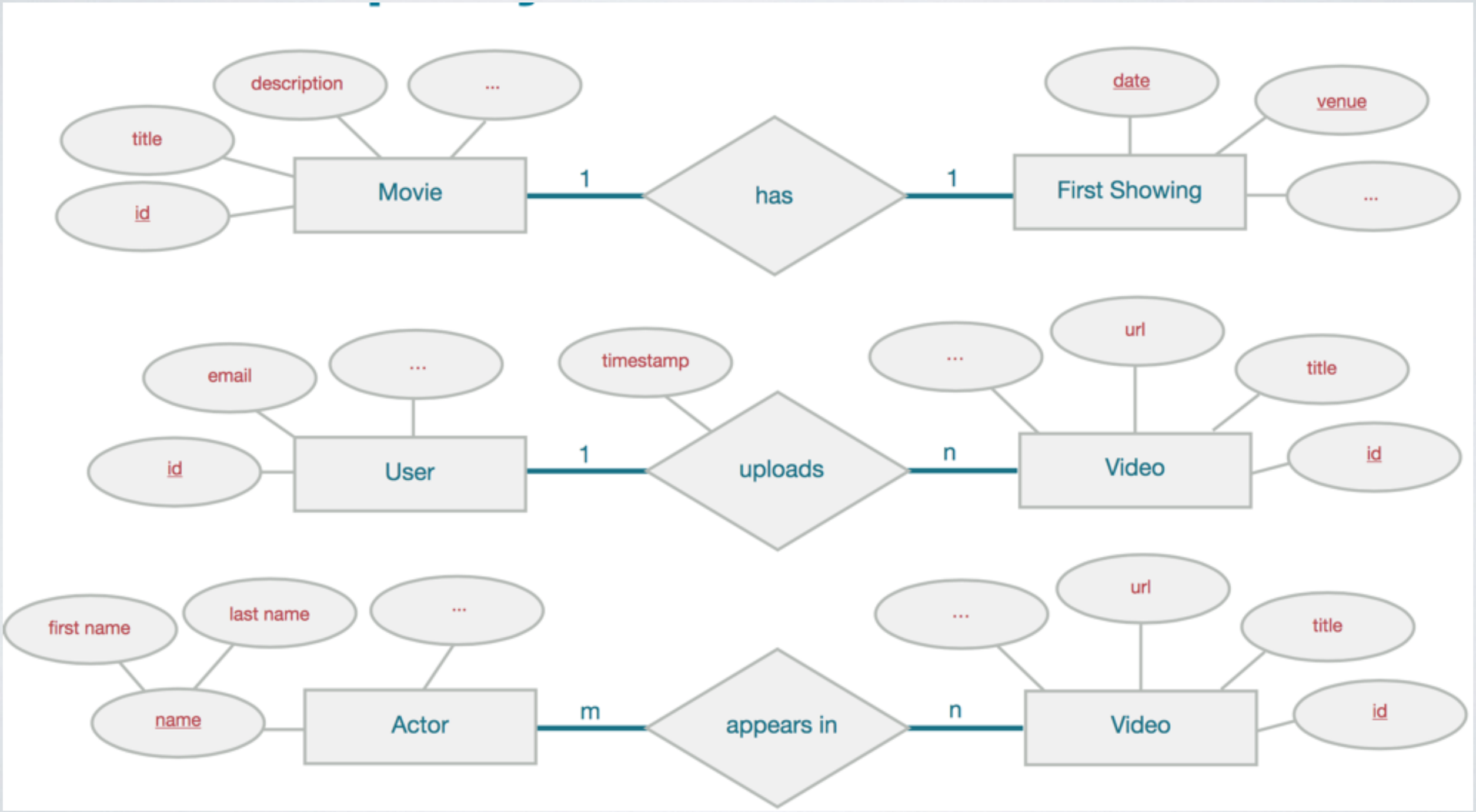
# RELATIONSHIP TYPE



# WEAK ENTITY TYPES



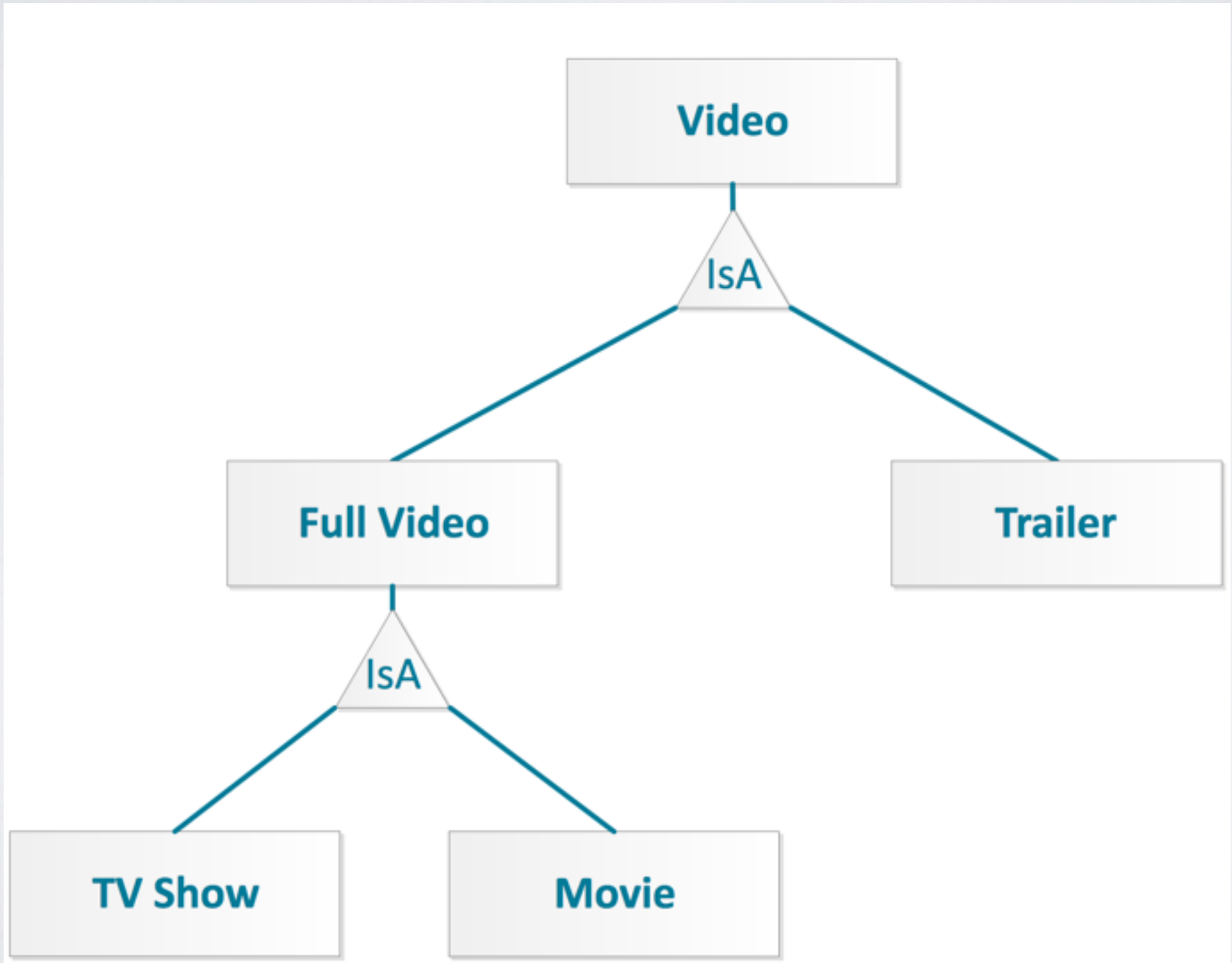
# RELATIONSHIP KEYS



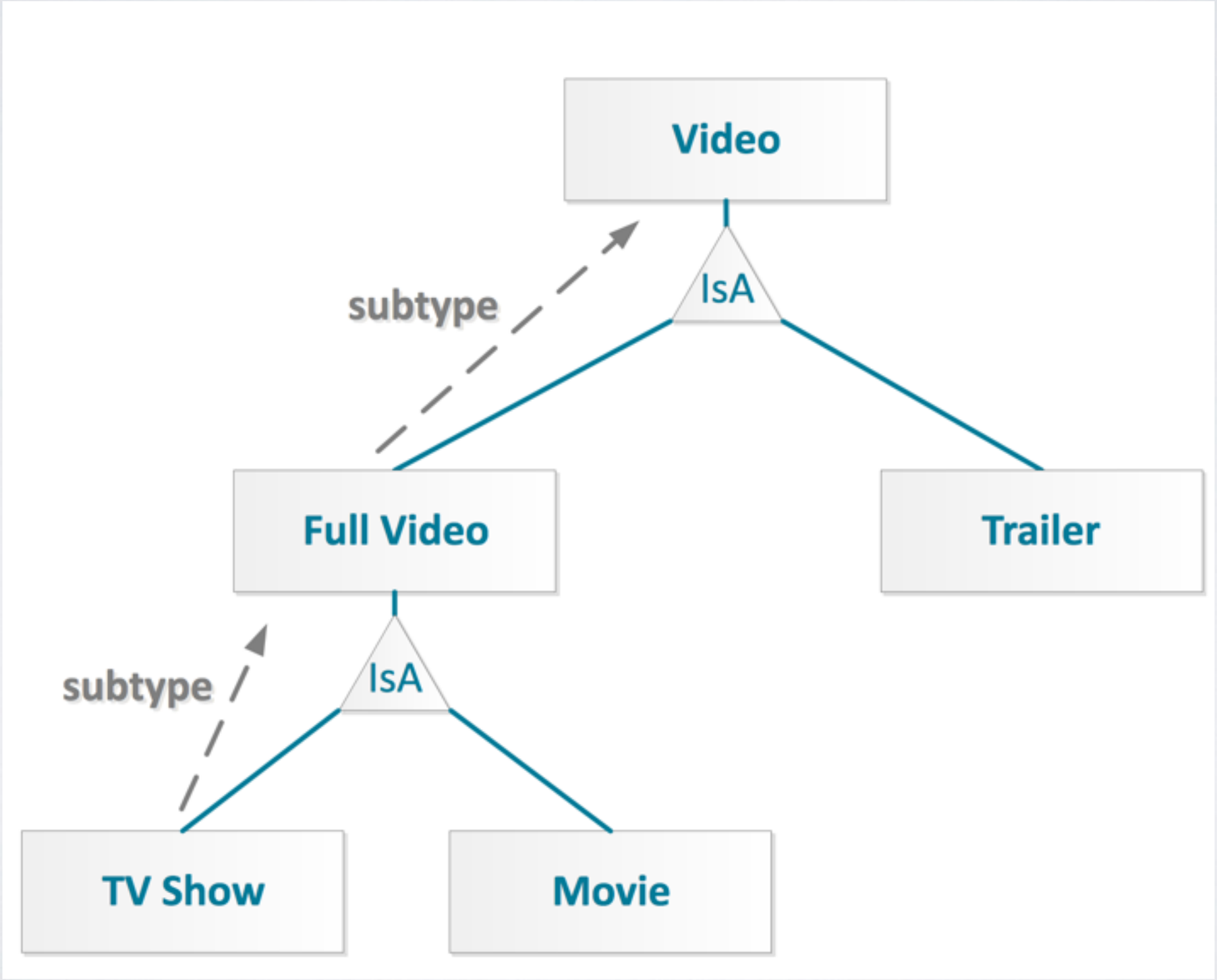
# VIDEOS

- Video
- Full Video
- Trailer
- TV Show
- Movie

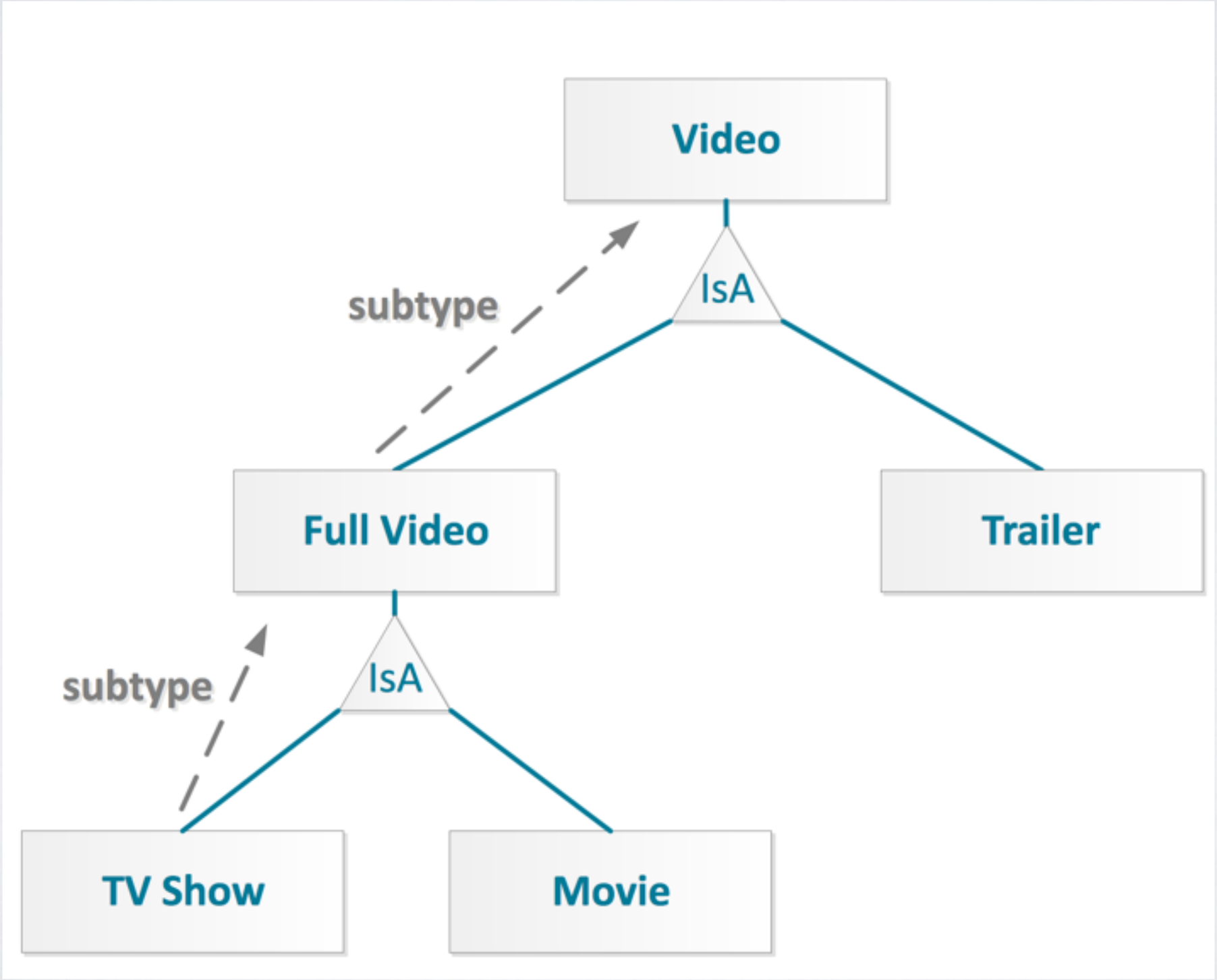
# ENTITY TYPE HIERARCHY



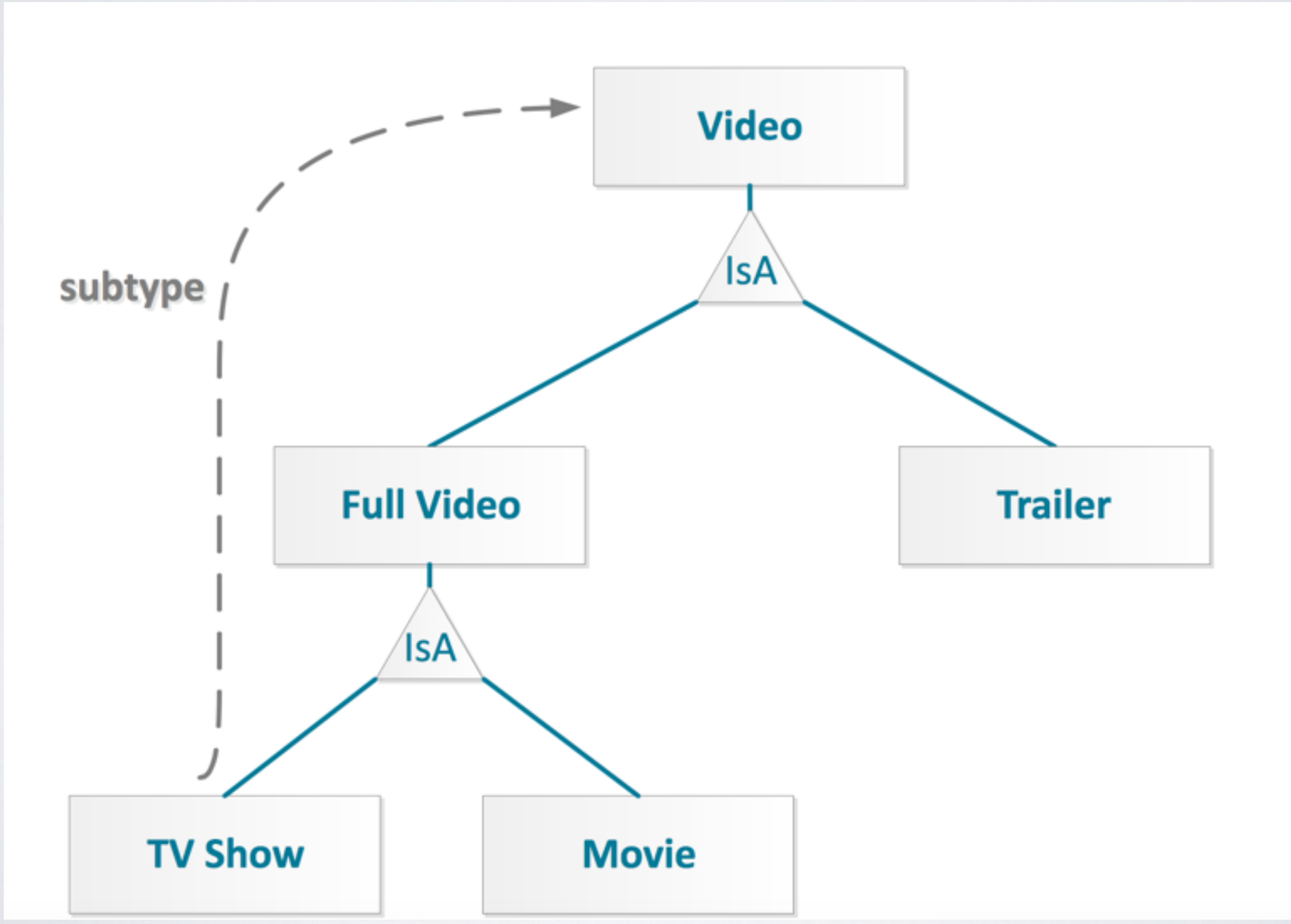
# TRANSITIVITY

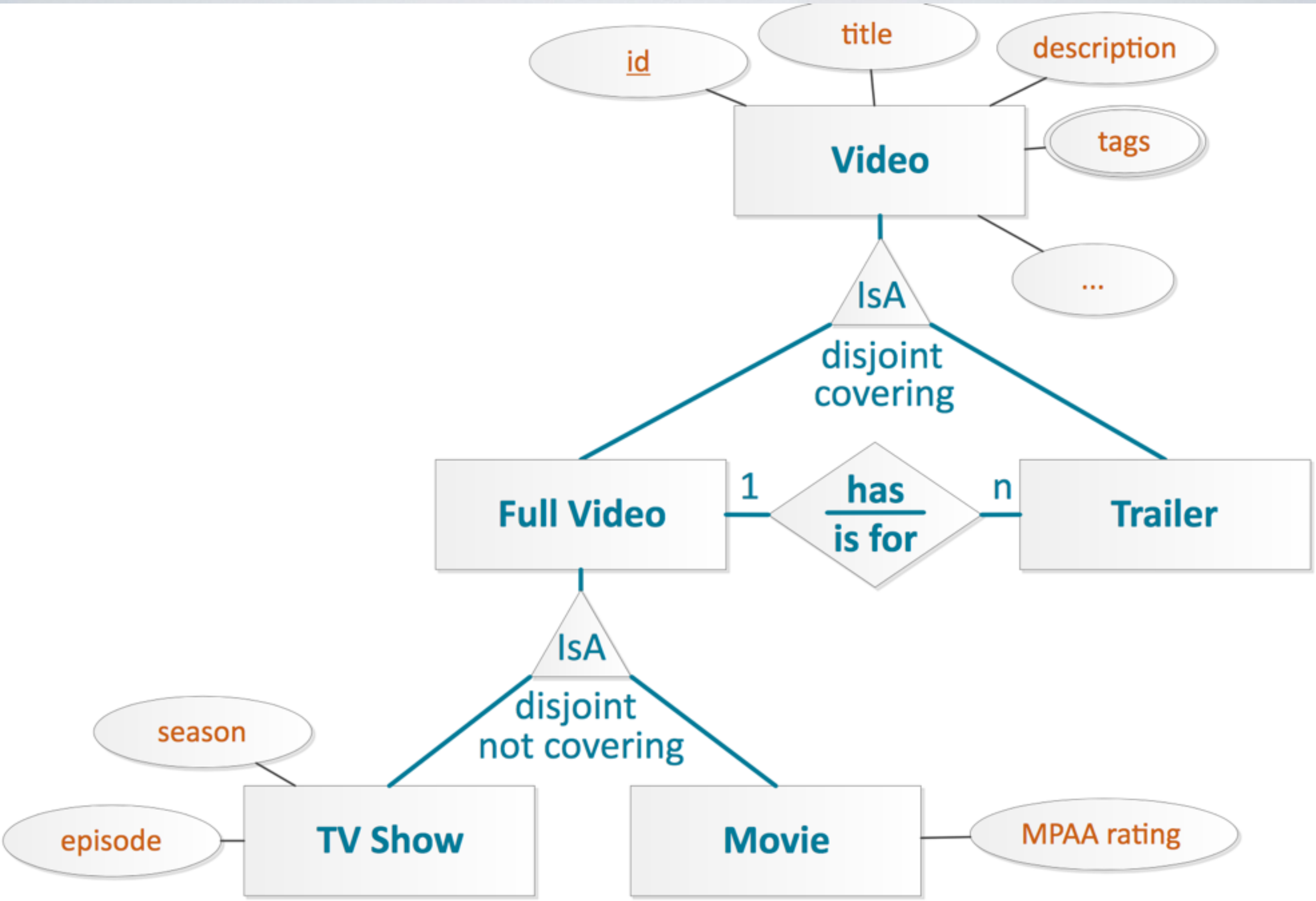


# TRANSITIVITY

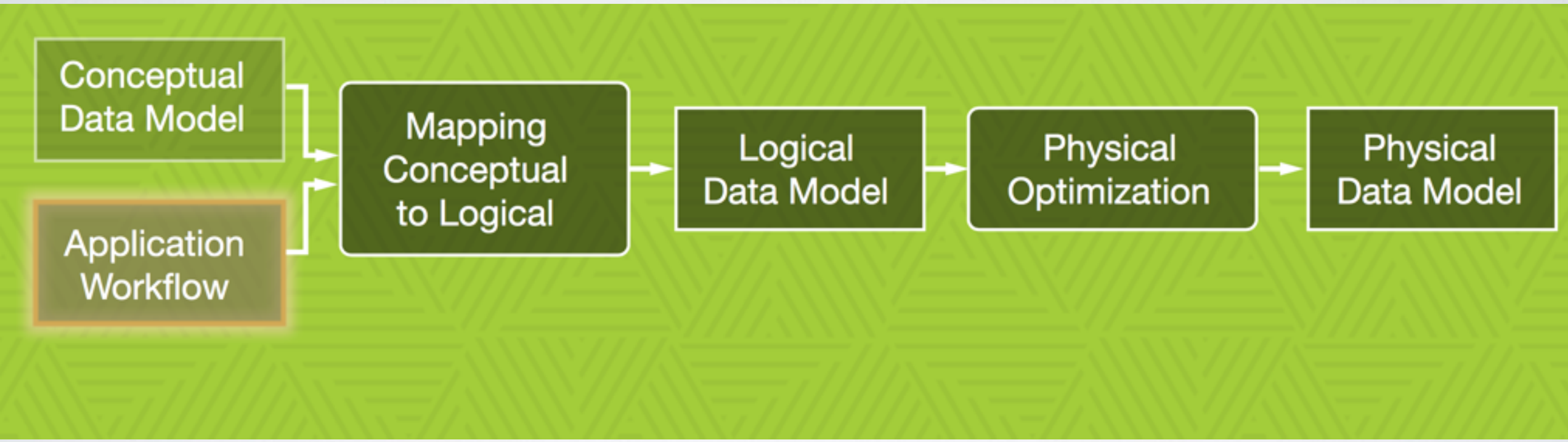


# TRANSITIVITY



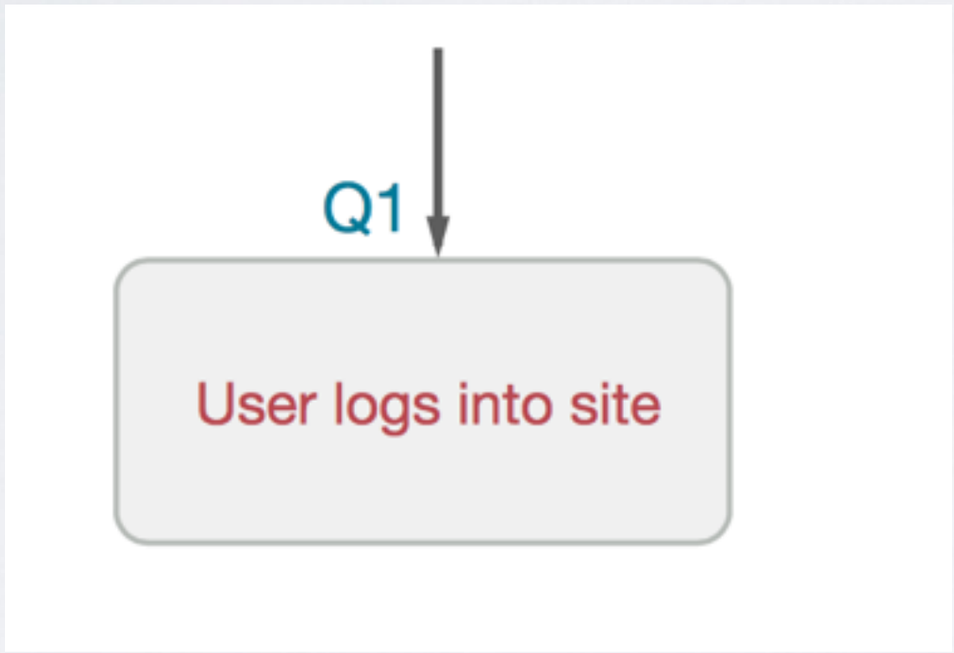

id
title
description
tags
...
Video
IsA
disjoint
covering
Full Video
1
has
is for
n
Trailer
IsA
disjoint
not covering
season
episode
TV Show
Movie
MPAA rating

# APPLICATION WORKFLOW MODEL



# APPLICATION WORKFLOW MODEL

Task: Have a user login to a site




Access Patterns

Q1: Find a user with a specified email

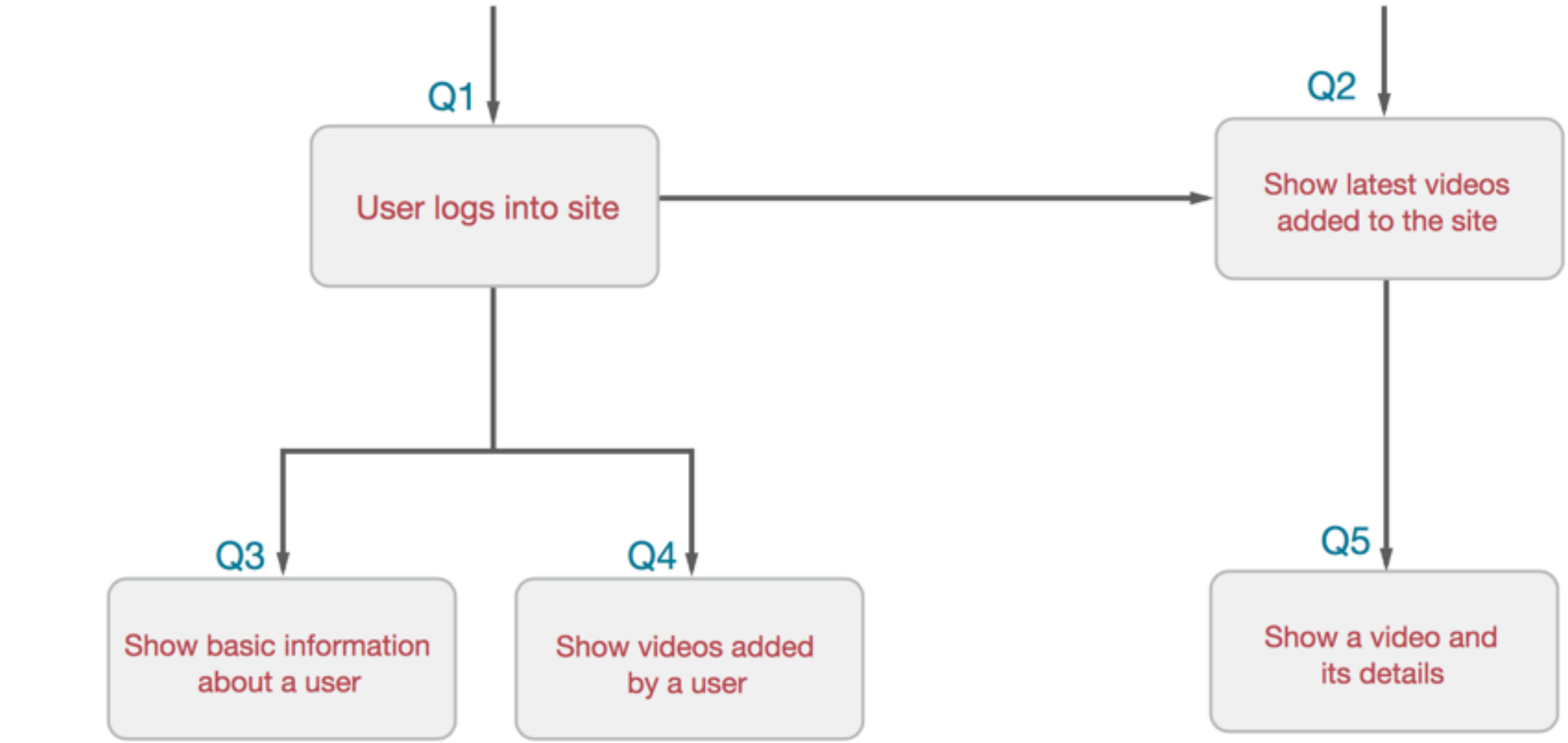
# Task: Show videos that were most recently uploaded


Access Patterns

Q1: Find a user with a specified email

Q2: Find most recently uploaded videos

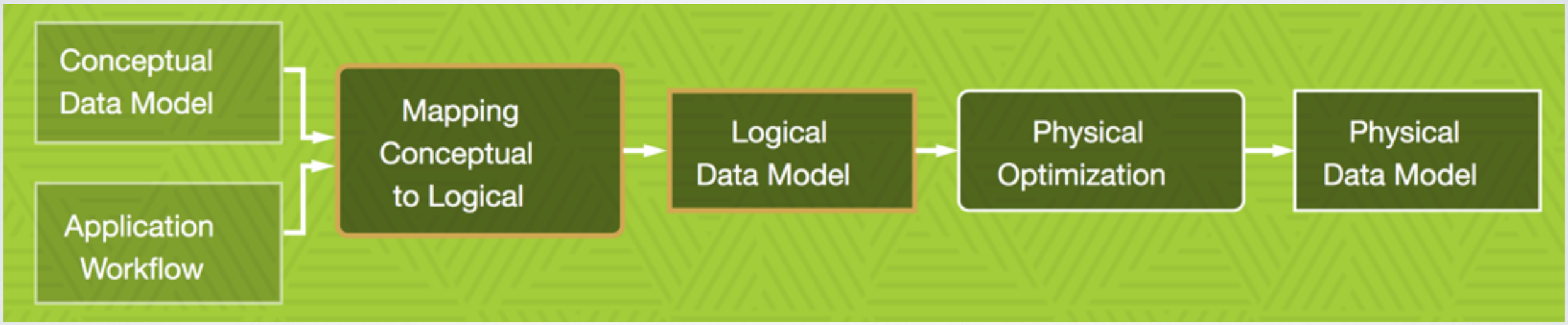
# Task: Find a video that has a particular video id




**ACCESS PATTERNS**

Q1: Find a user with a **specified email**
Q2: Find most recently uploaded **videos**
Q3: Find a **user with a specified id**
Q4: Find videos uploaded by a **user with a known id**(show most recently uploaded videos first)
Q5: Find a video with a **specified video id**

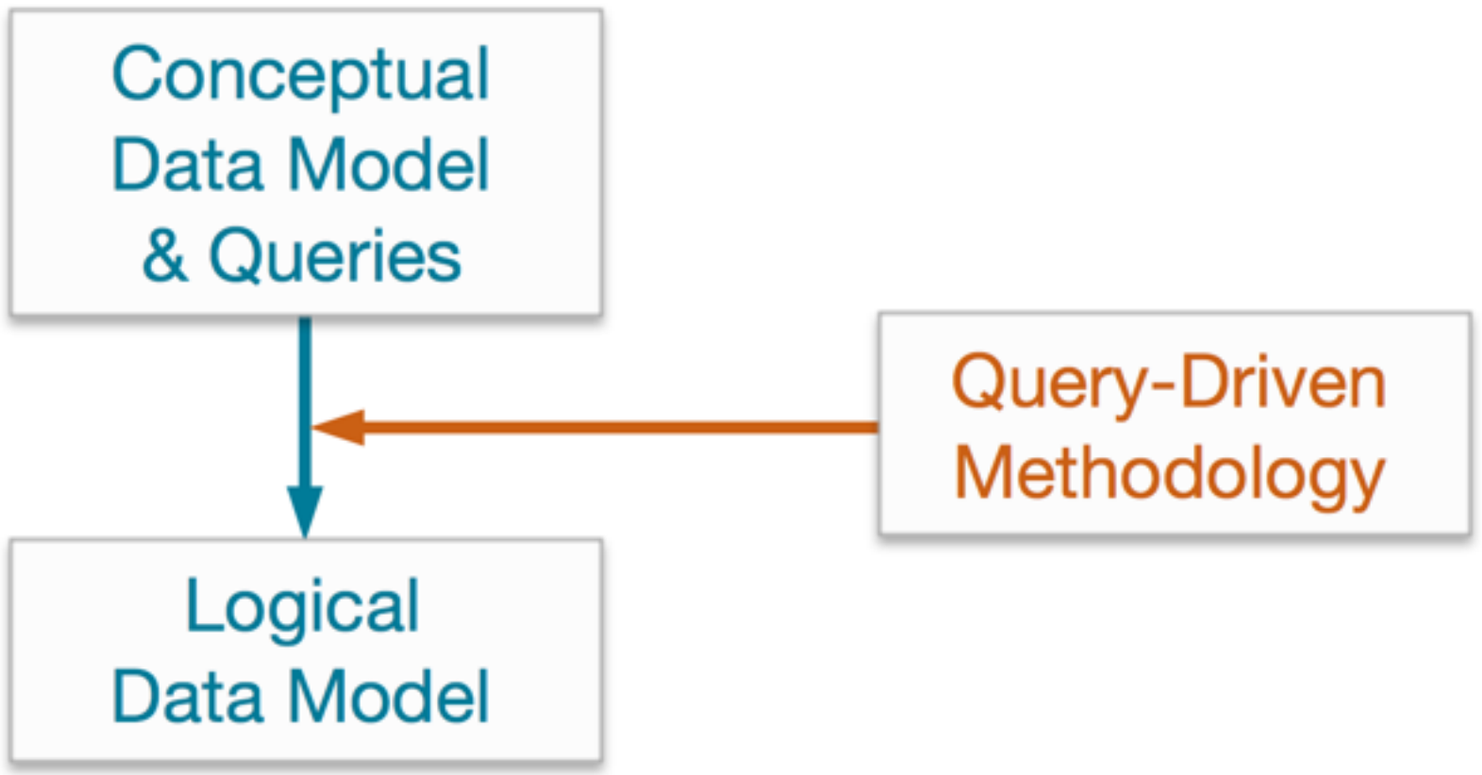
# MAPPING CONCEPTUAL TO LOGICAL



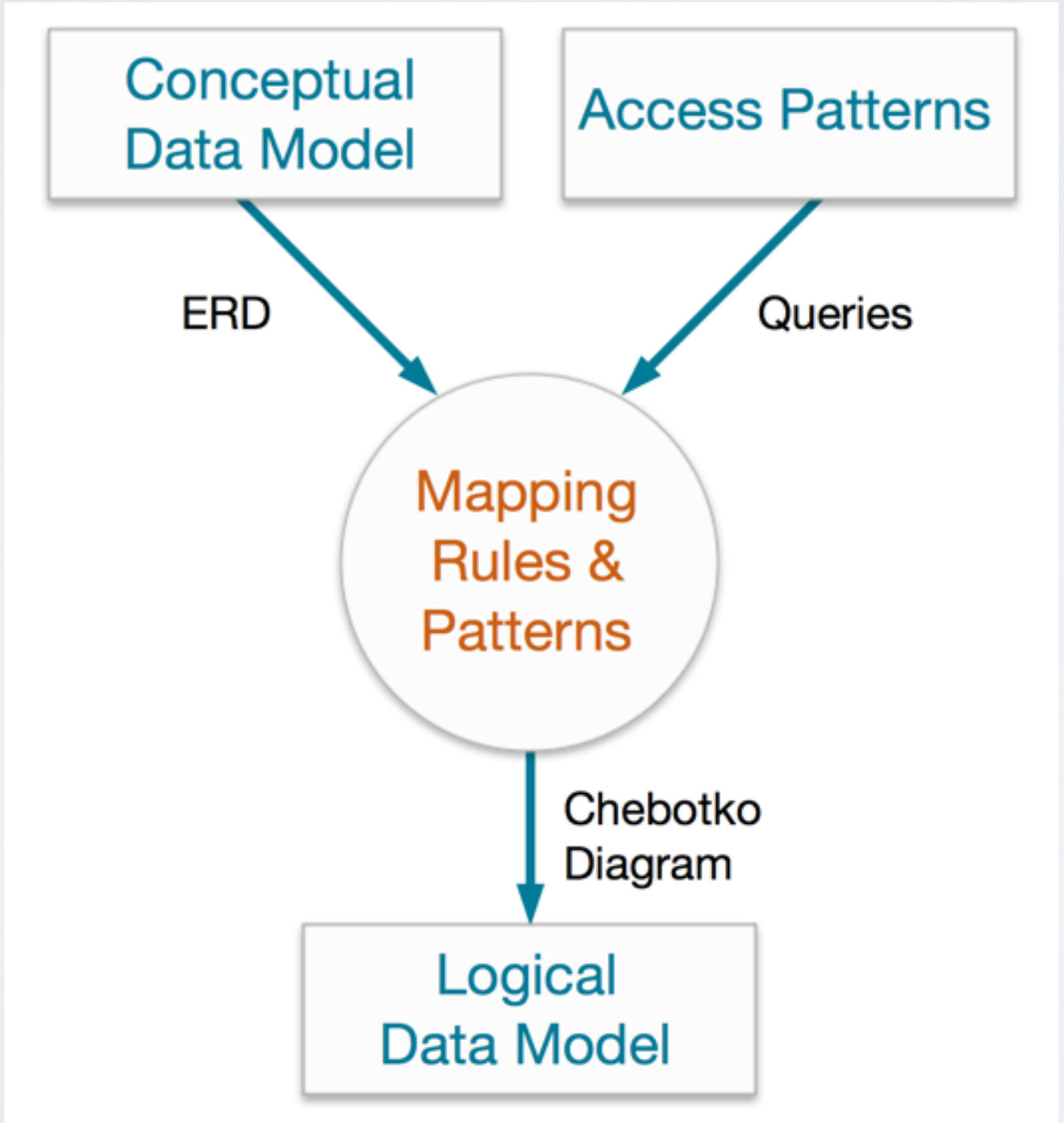
# DATA MODELING METHODOLOGY

## Process to design a logical model

- Uses a top-down approach
- Can be algorithmically defined
- Effective in the long run—vs. dark art

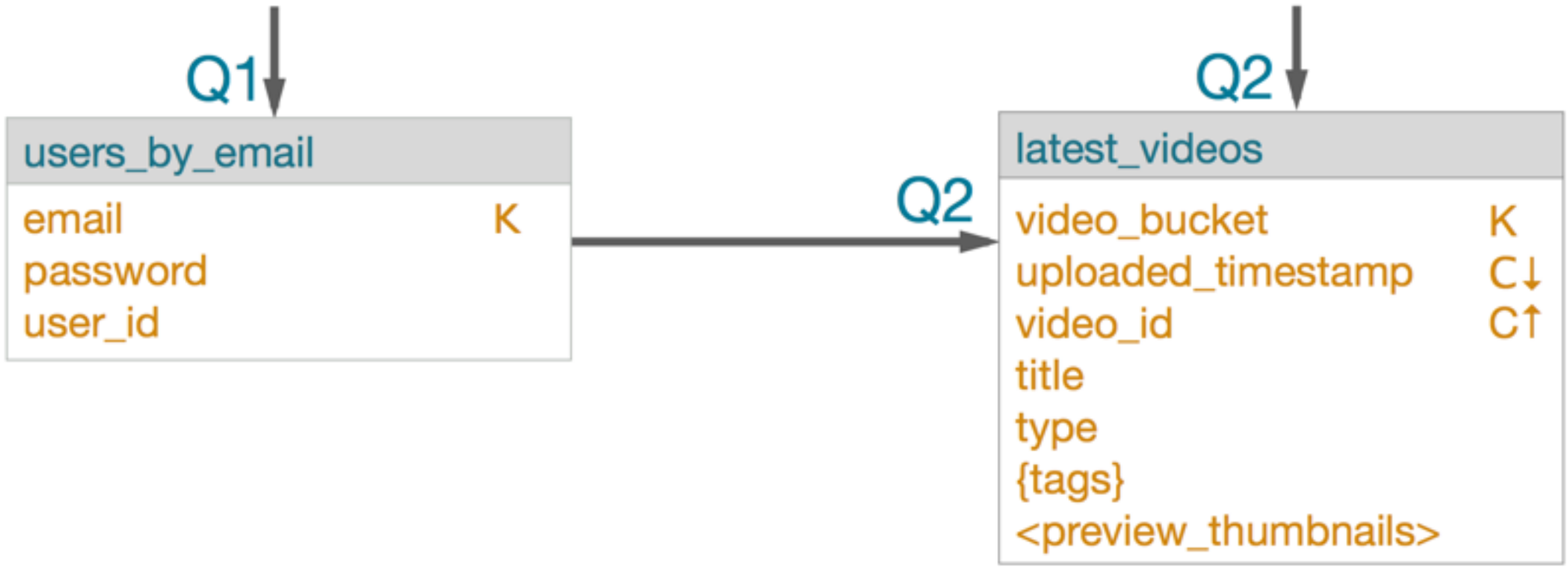


# QUERY-DRIVEN DATA MODELING



# CHEBOTKO DIAGRAM

- Visual diagram for Cassandra tables and access patterns



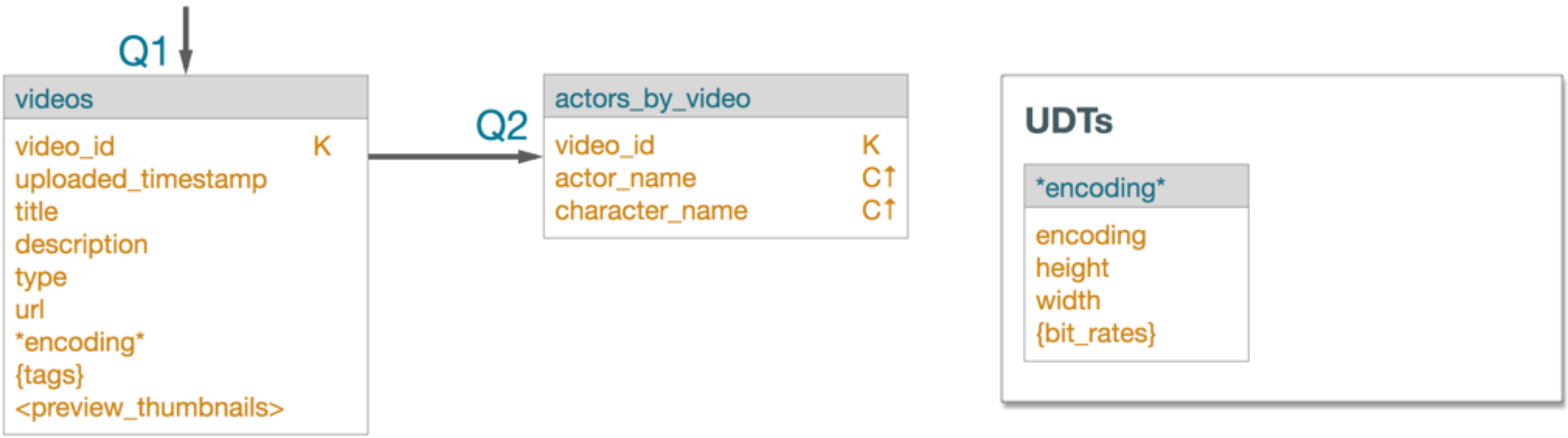

**ACCESS PATTERNS**

Q1: Find a user with a **specified email**

Q2: Find most recently uploaded videos

# CHEBOTKO DIAGRAMS

- Graphical representation of Cassandra database schema design
- Documents the logical and physical data model



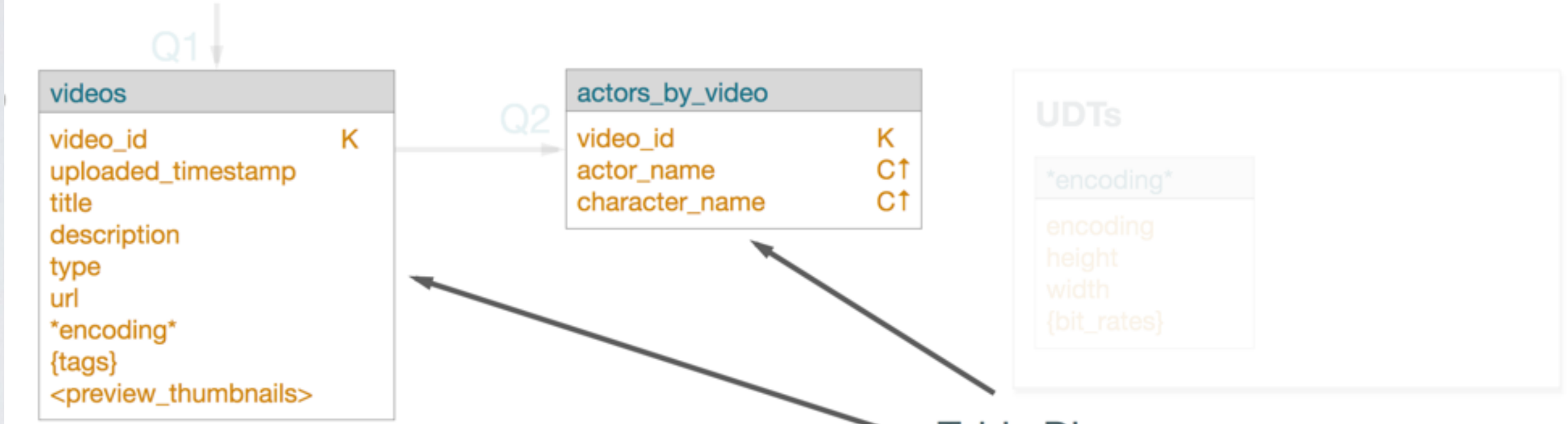

**ACCESS PATTERNS**

Q1: Find a video with a **specified video id**

Q2: Find actors for a **video with a known id** (show actor names in ascending order)

# CHEBOTKO DIAGRAMS

- Graphical representation of Cassandra database schema design
- Documents the logical and physical data model



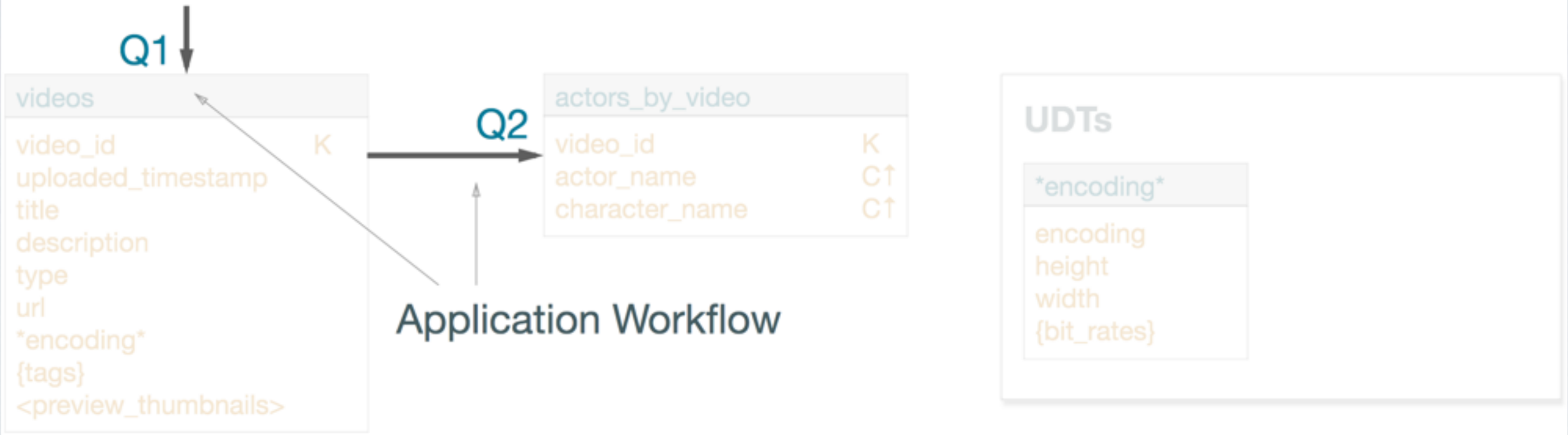

ACCESS PATTERNS

Q1: Find a video with a **specified video id**

Q2: Find actors for a **video with a known id** (show actor names in ascending order)

# CHEBOTKO DIAGRAMS

- Graphical representation of Cassandra database schema design
- Documents the logical and physical data model



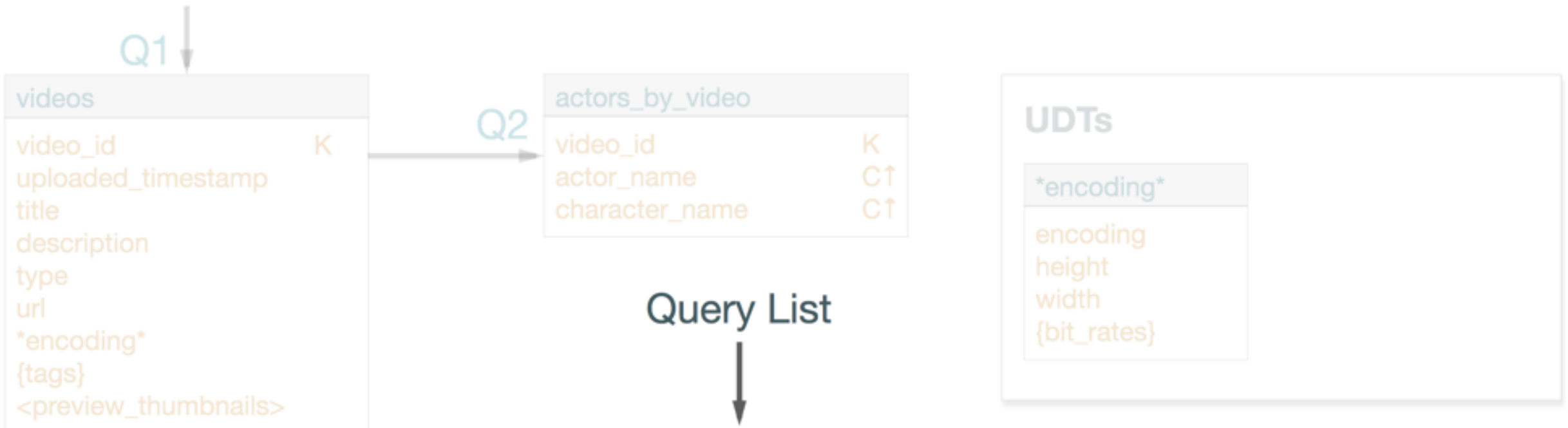

## ACCESS PATTERNS

Q1: Find a video with a **specified video id**

Q2: Find actors for a **video with a known id** (show actor names in ascending order)

# CHEBOTKO DIAGRAMS

- Graphical representation of Cassandra database schema design
- Documents the logical and physical data model




Query List

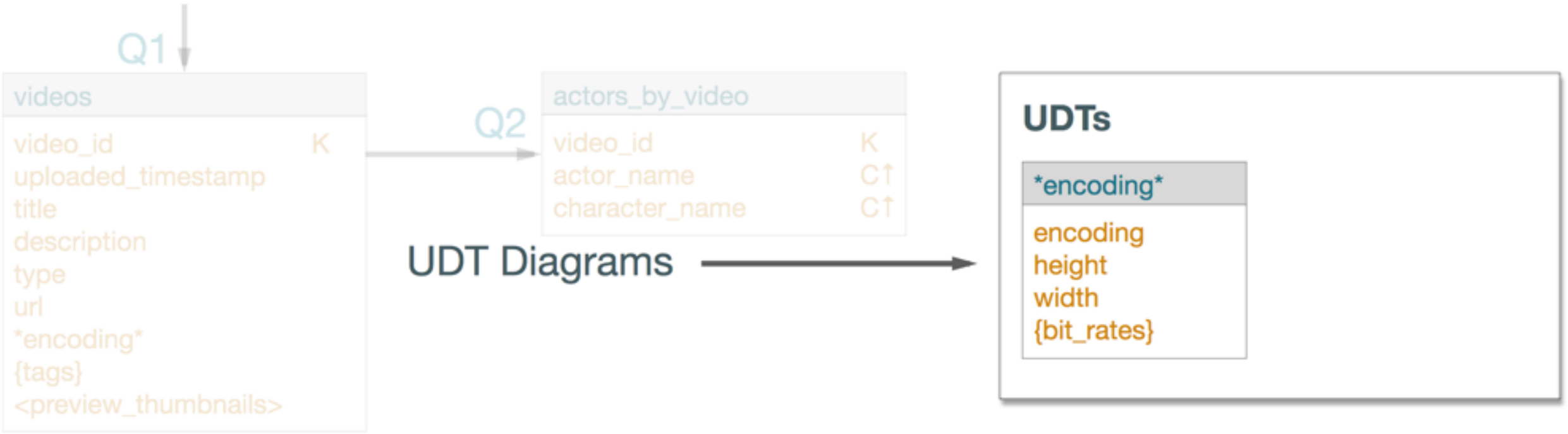
**ACCESS PATTERNS**

Q1: Find a video with a **specified video id**

Q2: Find actors for a **video with a known id** (show actor names in ascending order)

# CHEBOTKO DIAGRAMS

- Graphical representation of Cassandra database schema design
- Documents the logical and physical data model




**ACCESS PATTERNS**

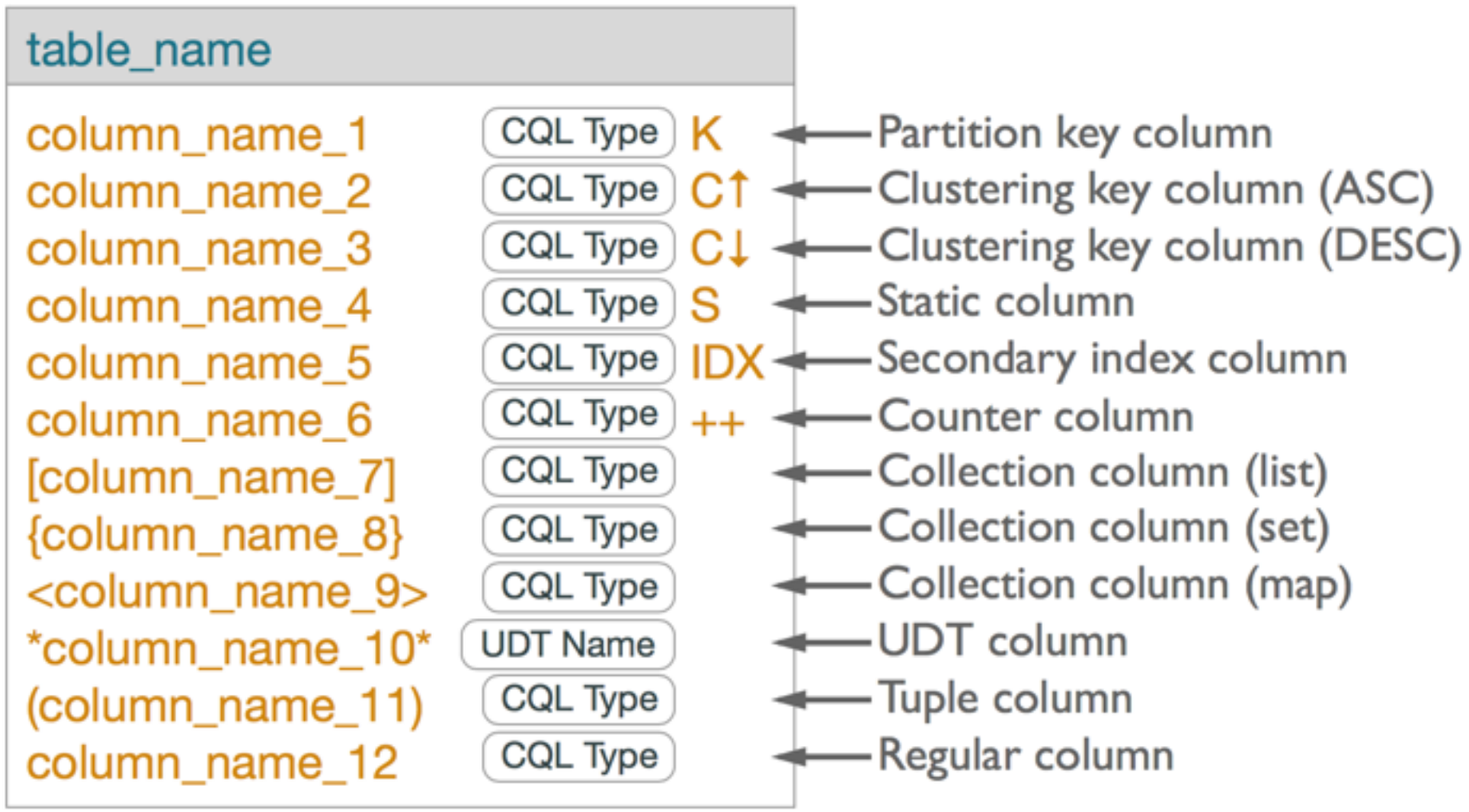
Q1: Find a video with a **specified video id**

Q2: Find actors for a **video with a known id** (show actor names in ascending order)

# CHEBOTKO DIAGRAM NOTATION

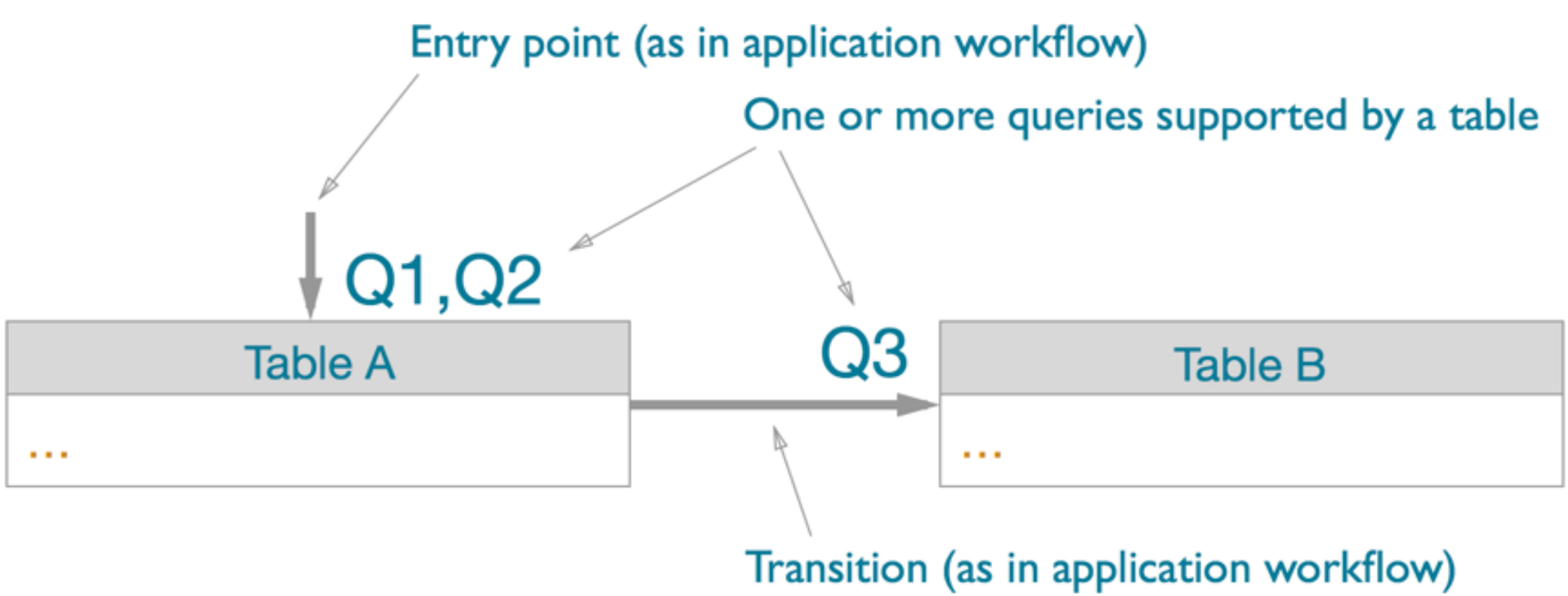
## Table representation

- Logical-level shows column names and properties
- Physical-level also shows the column data type



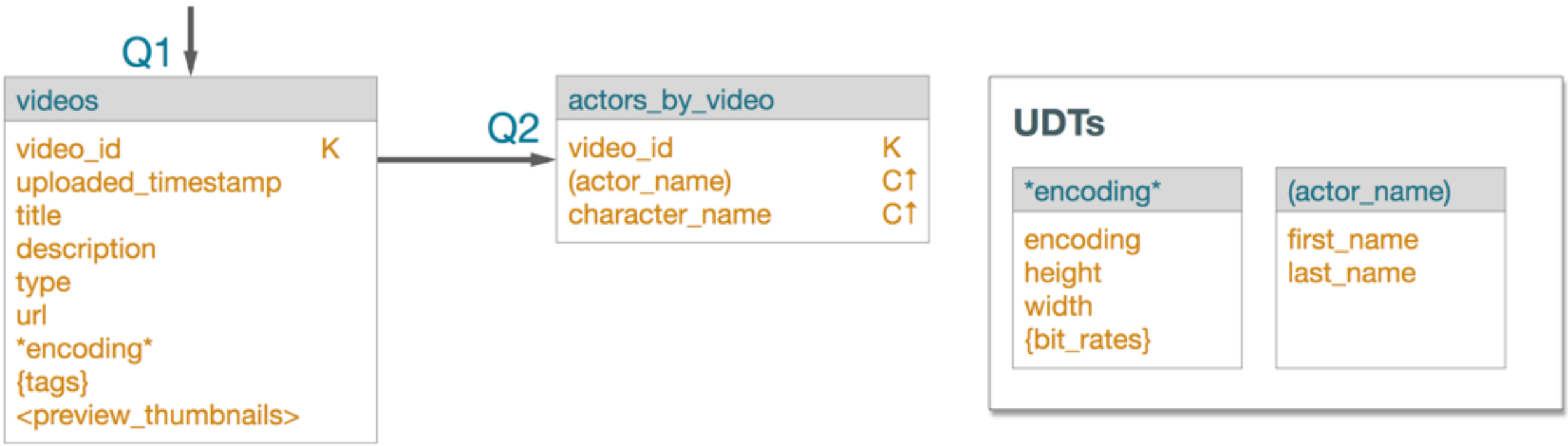
# ACCESS PATTERNS

- Directed links and query label shows how tables are accessed
- Similar to the application workflow, but now with our logical information



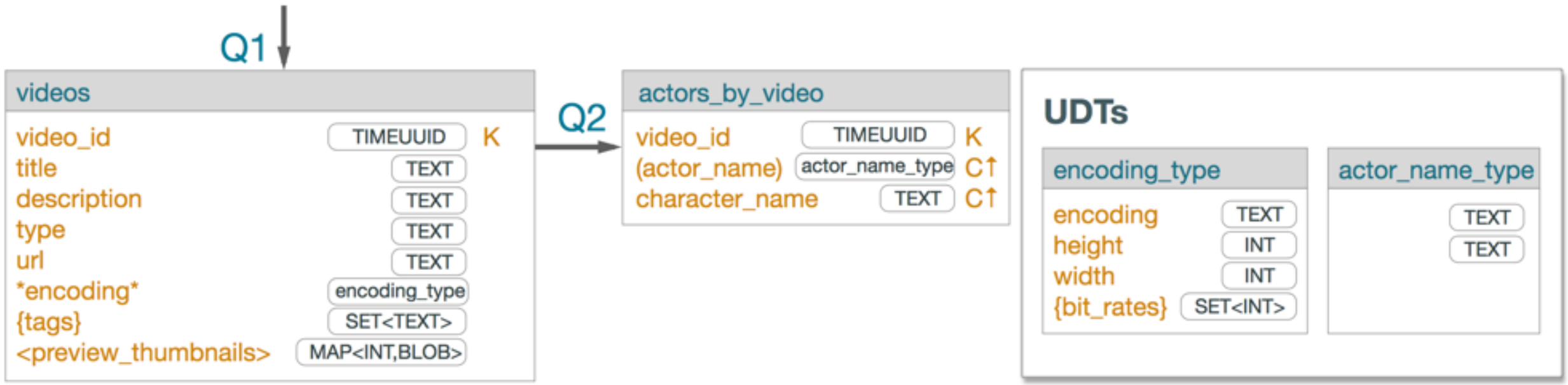
# LOGICAL UDT DIAGRAM

- Represents user defined types and tuples

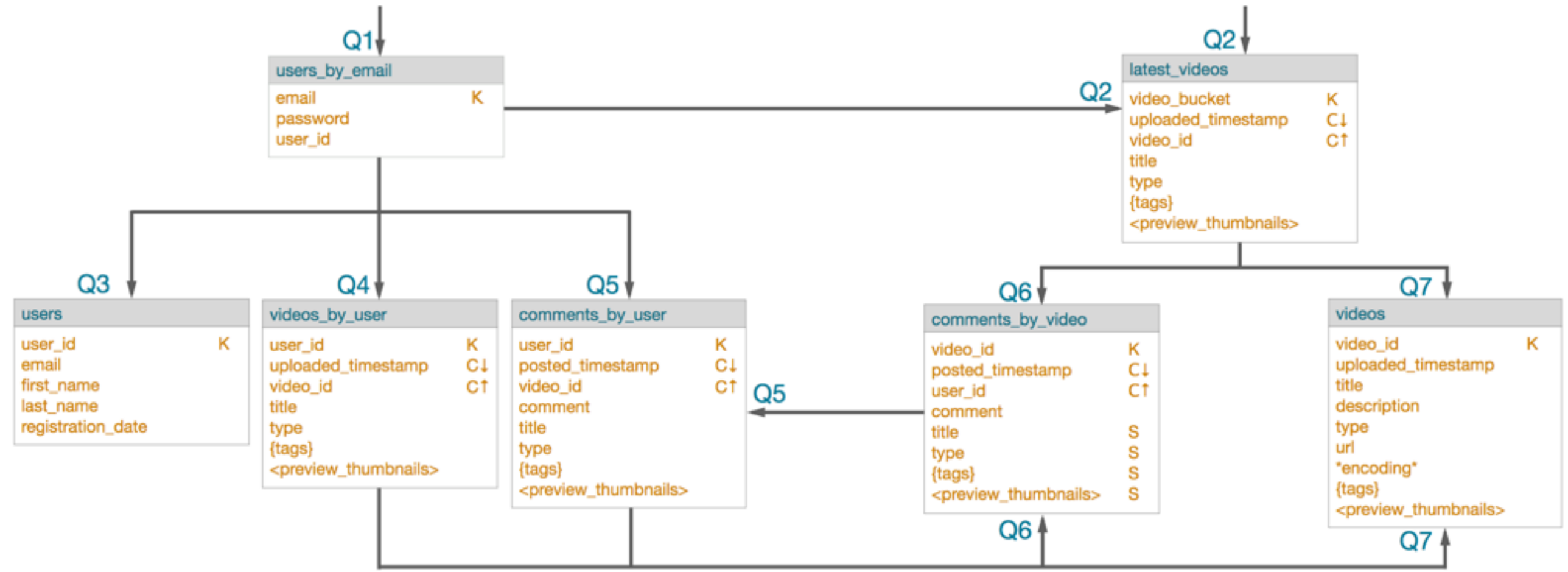


# PHYSICAL UDT DIAGRAM

- Represents user defined types and tuples



# EXAMPLE CHEBOTKO DIAGRAM




**ACCESS PATTERNS**

Q1: Find a user with a **specified email**
Q2: Find most recently uploaded videos
Q3: Find a **user with a specified id**
Q4: Find videos uploaded by a **user with a known id** (show most recently uploaded videos first)
Q5: Find comments posted by a **user with a known id** (show most recently posted comments first)
Q6: Find comments posted for a **video with a known id** (show most recent comments first)
Q7: Find a video with a **specified video id**

**UDTs**

*encoding*
encoding
height
width
{bit_rates}

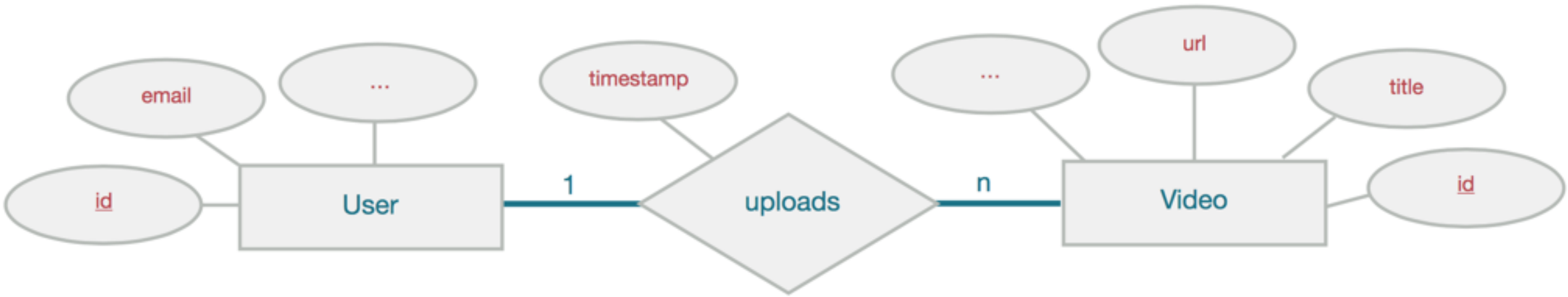
# CASSANDRA DATA MODELING PRINCIPLES

- Know your data
- Know your queries
- Nest data
- Duplicate data

# KNOW YOUR DATA

## Understanding the data is the key to successful design

- Data captured by conceptual data model
- Define what is stored in database
- Preserve properties so that data is organized correctly



# KNOW YOUR DATA

## Key constraints affect schema design

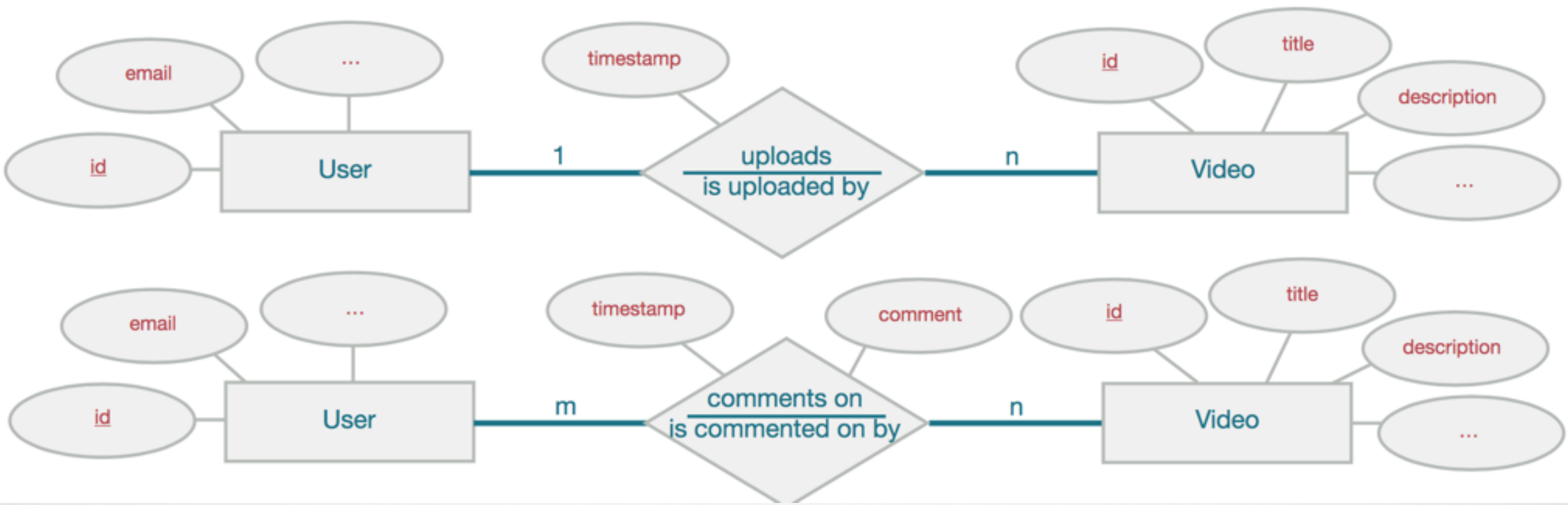
- Entity and relationship keys affect the table primary keys
- Primary key uniquely identifies a row / entity / relationship
- Composed of a key and possibly additional columns

| videos | |
|---|---|
| video_id | K |
| uploaded_timestamp | |
| user_id | |
| title | |
| description | |
| type | |
| *encoding* | |
| {tags} | |
| <preview_thumbnails> | |
| {genres} | |

| videos_by_user | |
|---|---|
| user_id | K |
| uploaded_timestamp | C↓ |
| video_id | C↑ |
| title | |
| type | |
| {tags} | |
| <preview_thumbnails> | |

# KNOW YOUR DATA

## Cardinality constraints affect the key for relationships



# KNOW YOUR QUERIES

## Queries directly affect schema design

- Queries captured by application workflow model
- Table schema design changes if queries change


**ACCESS PATTERNS**

Q1: Find a user with a **specified email**
Q2: Find most recently uploaded **videos**

# KNOW YOUR QUERIES

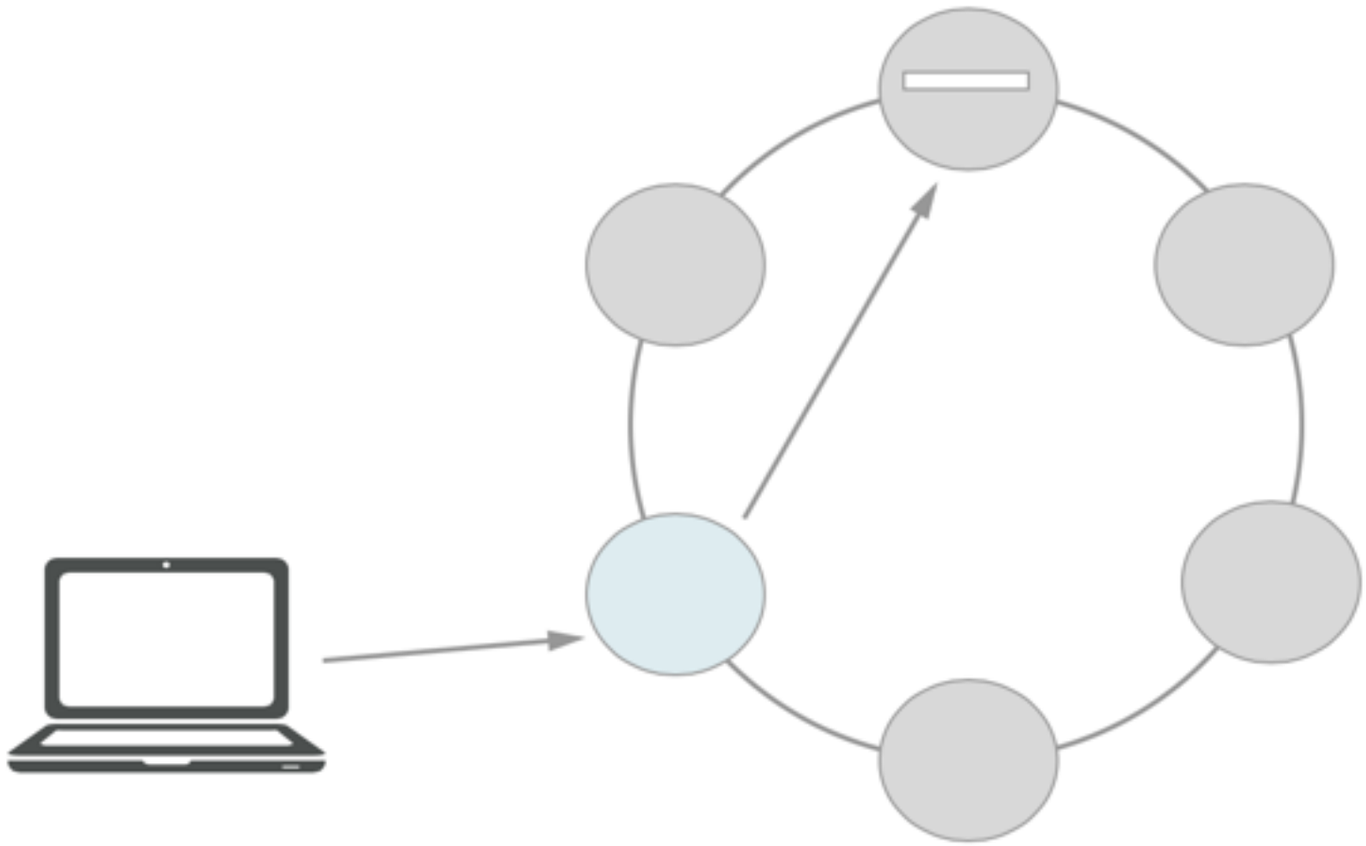
Schema design organizes data to efficiently run queries

- Partition per query — **ideal**
- Partition+ per query — **acceptable**
- Table scan — **anti-pattern**
- Multi-table — **anti-pattern**

# KNOW YOUR QUERIES

## Partition per query

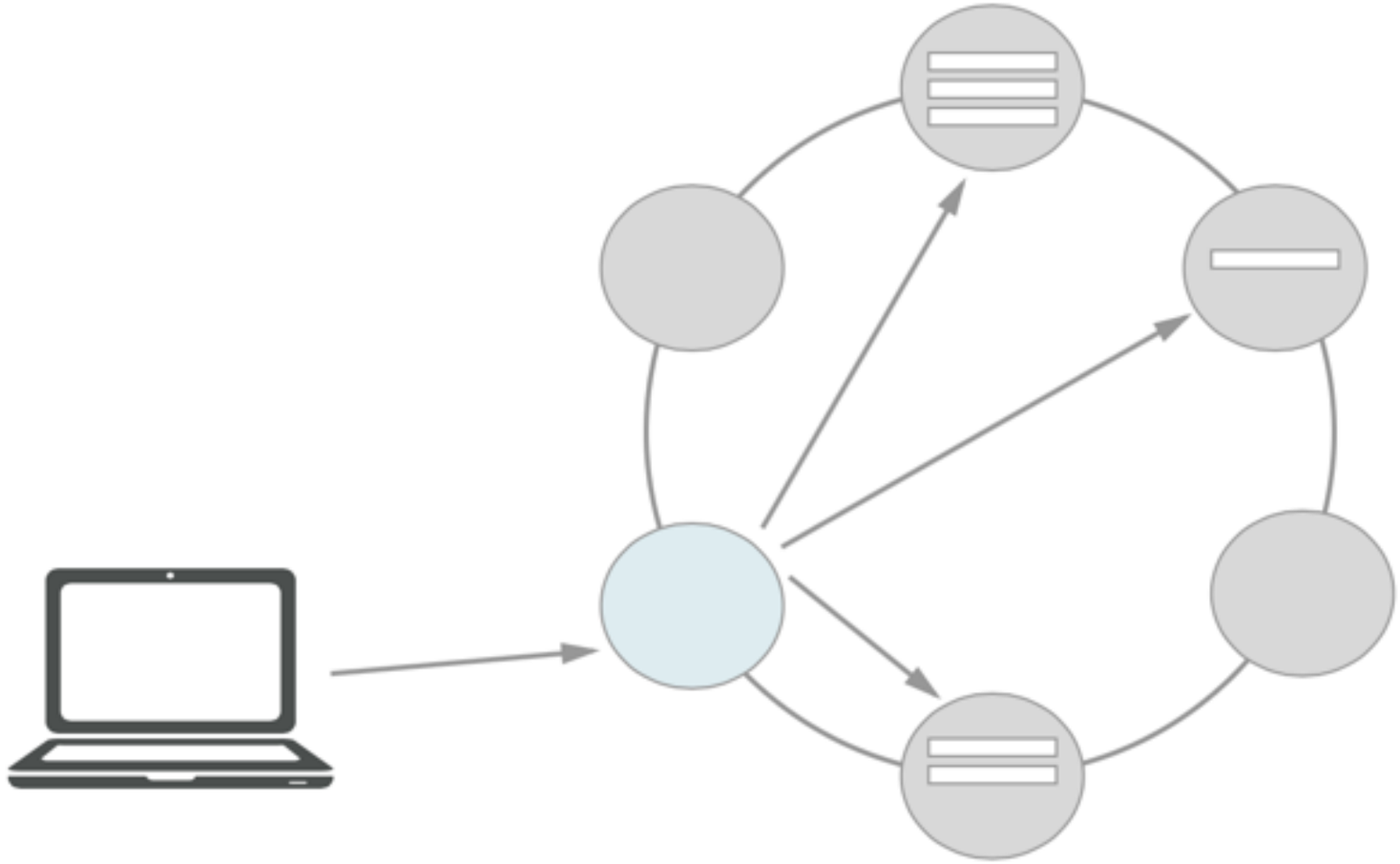
- The most efficient access pattern
- Query accesses only one partition to retrieve results
- Partition can be single-row or multi-row



# KNOW YOUR QUERIES

## Partition+ per query

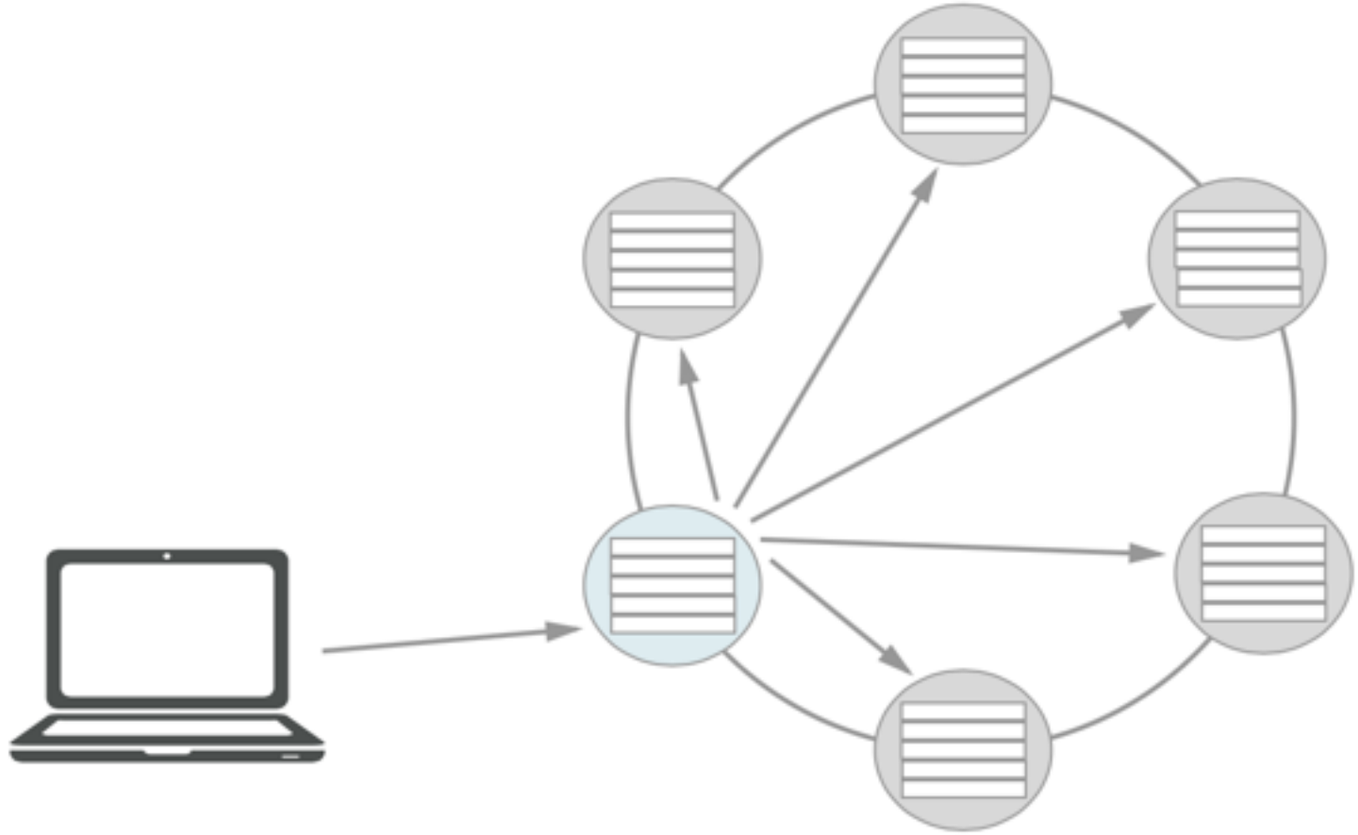
- Less efficient access pattern but not necessarily bad
- Query needs to access multiple partitions to retrieve results



# KNOW YOUR QUERIES

## Table scan and Multi-table

- Least efficient type of query but may be needed in some cases
- Query needs to access all partitions in a table(s) to retrieve results



# NEST DATA

## Data nesting is the main data modeling technique

- Nesting organizes multiple entities into a single partition
- Supports partition per query data access
- Three data nesting mechanisms
    - Clustering columns—multi-row partitions
    - Collection columns
    - User-defined type columns

# NEST DATA

## Clustering columns - primary data nesting mechanism

- Partition key identifies an entity that other entities will nest into
- Values in a clustering column identify the nested entities
- Multiple clustering columns implement multi-level nesting

| videos | |
|---|---|
| video_id | K |
| uploaded_timestamp | |
| user_id | |
| title | |
| description | |
| type | |
| {tags} | |
| <preview_thumbnails> | |
| {genres} | |

| actors_by_video | |
|---|---|
| video_id | K |
| actor_name | C↑ |
| character_name | C↑ |

# NEST DATA

## User-defined type—secondary data nesting mechanism

- Represents one-to-one relationship, but can use in conjunction with collections
- Easier than working with multiple collection columns

| videos_by_user | | |
|---|---|---|
| user_id | UUID | K |
| [videos] | video_type | |

**UDTs**

| video_type | |
|---|---|
| id | TIMEUUID |
| title | TEXT |
| description | TEXT |
| type | TEXT |
| url | TEXT |
| release_date | TIMESTAMP |
| mpaa_rating | TEXT |

# DUPLICATE DATA

## Better to duplicate than to join data

- Partition per query and data nesting may result in data duplication
  - Query results are pre-computed and materialized
  - Data can be duplicated across tables, partitions, and / or rows

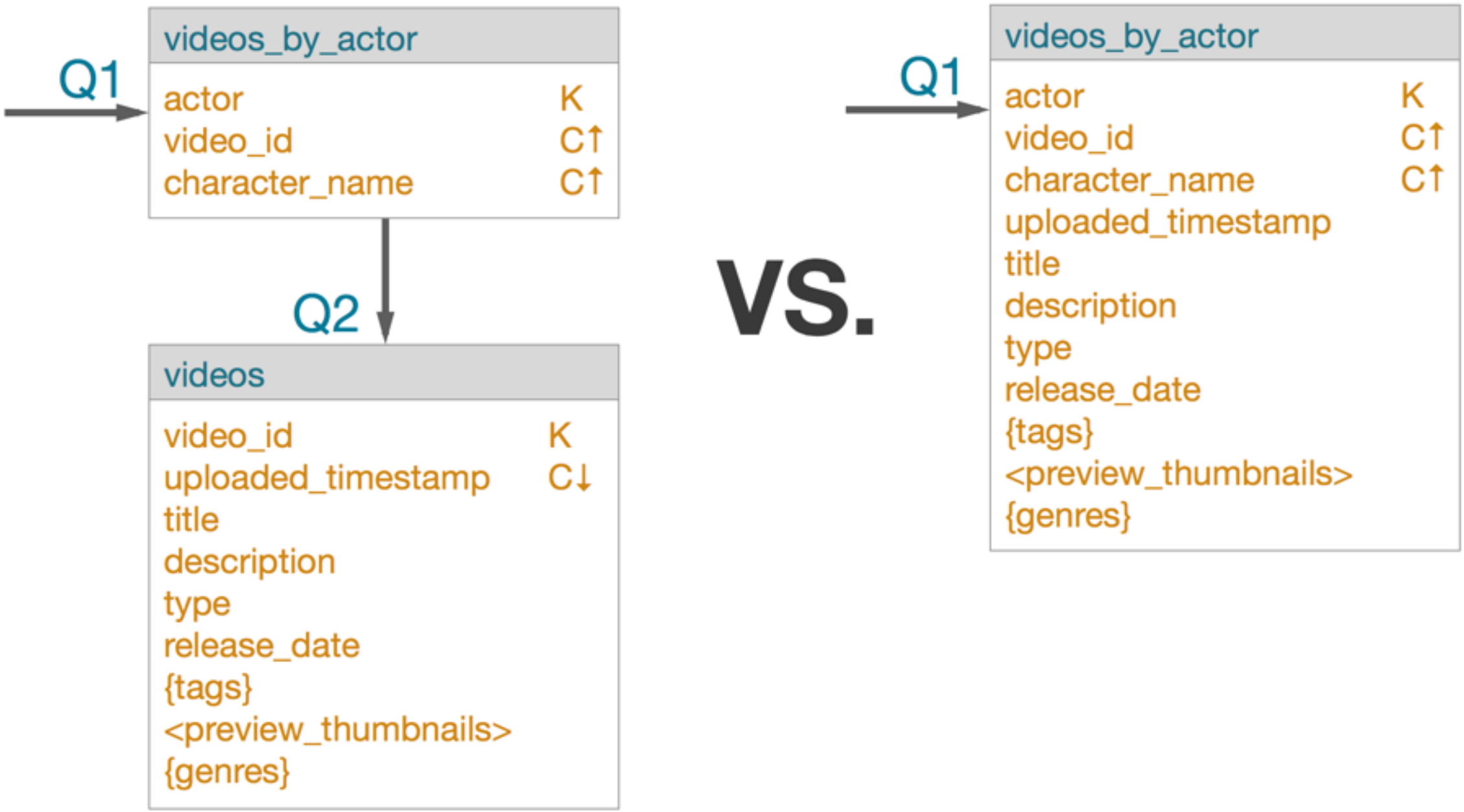
| videos_by_actor | |
|---|---|
| actor | K |
| release_date | C↓ |
| video_id | C↑ |
| title | |
| type | |
| {tags} | |
| <preview_thumbnails> | |

| videos_by_genre | |
|---|---|
| genre | K |
| release_date | C↓ |
| video_id | C↑ |
| title | |
| type | |
| {tags} | |
| <preview_thumbnails> | |

| videos_by_tag | |
|---|---|
| tag | K |
| release_date | C↓ |
| video_id | C↑ |
| title | |
| type | |
| {tags} | |
| <preview_thumbnails> | |

# DUPLICATE DATA

Data duplication can scale, joins cannot



# MAPPING RULES

## For the query-driven methodology

- Mapping rules ensure that a logical data model is correct
- Each query has a corresponding table
- Tables are designed to allow queries to execute properly
- Tables return data in the correct order

# MAPPING RULES

- Mapping Rule 1: Entities and relationships
- Mapping Rule 2: Equality search attributes
- Mapping Rule 3: Inequality search attributes
- Mapping Rule 4: Ordering attributes
- Mapping Rule 5: Key attributes

# MR I: ENTITIES AND RELATIONSHIPS

- Entity and relationship types map to tables
- Entities and relationships map to partitions or rows
- Partition may have data about one or more entities and relationships
- Attributes are represented by columns

Conceptual Model

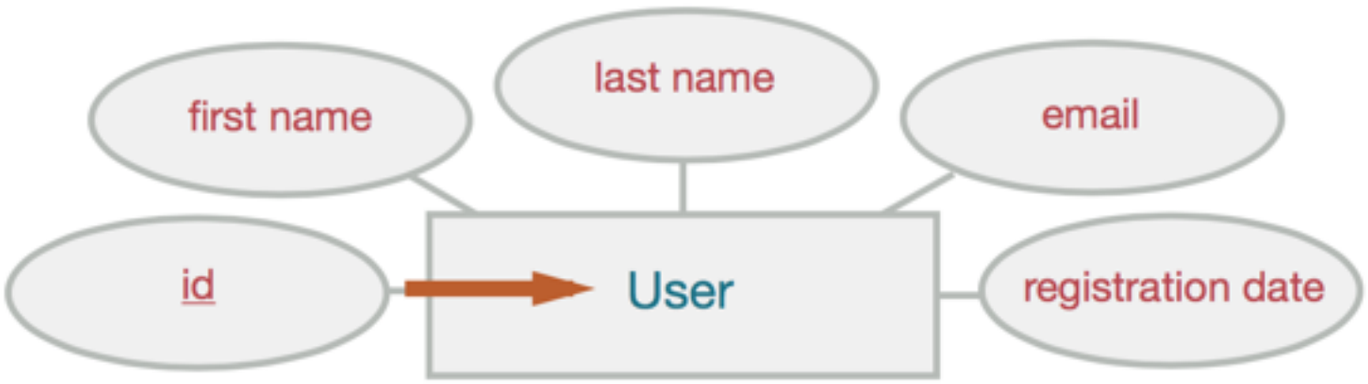

Logical Model

| users | |
|---|---|
| user_id | K |
| email | |
| first_name | |
| last_name | |
| registration_date | |

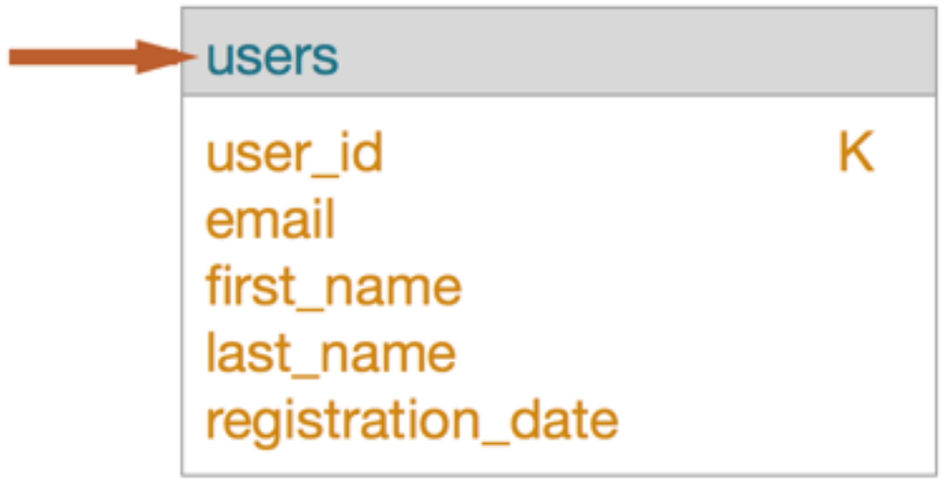
# MR I: ENTITIES AND RELATIONSHIPS

- Entity and relationship types map to tables
- Entities and relationships map to partitions or rows
- Partition may have data about one or more entities and relationships
- Attributes are represented by columns

Conceptual Model



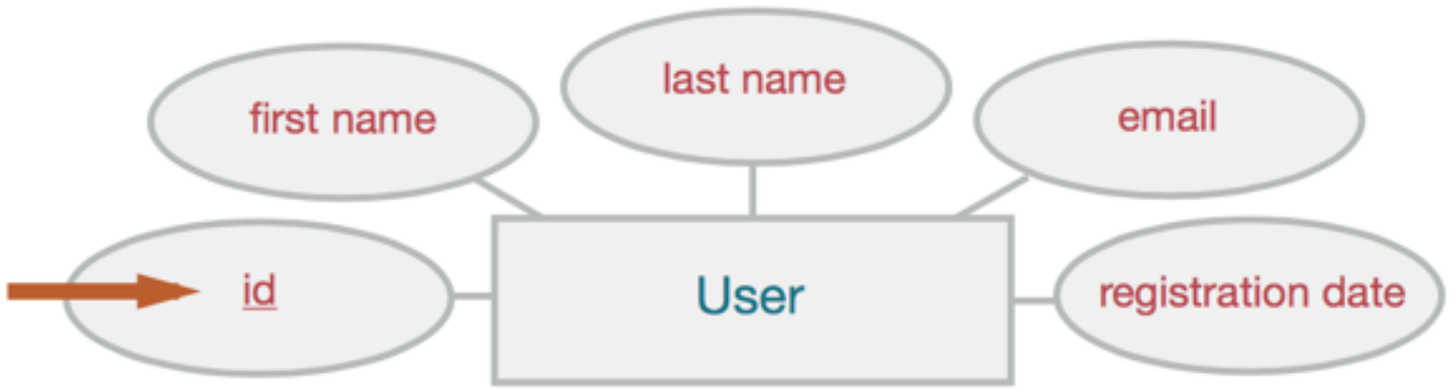

Logical Model



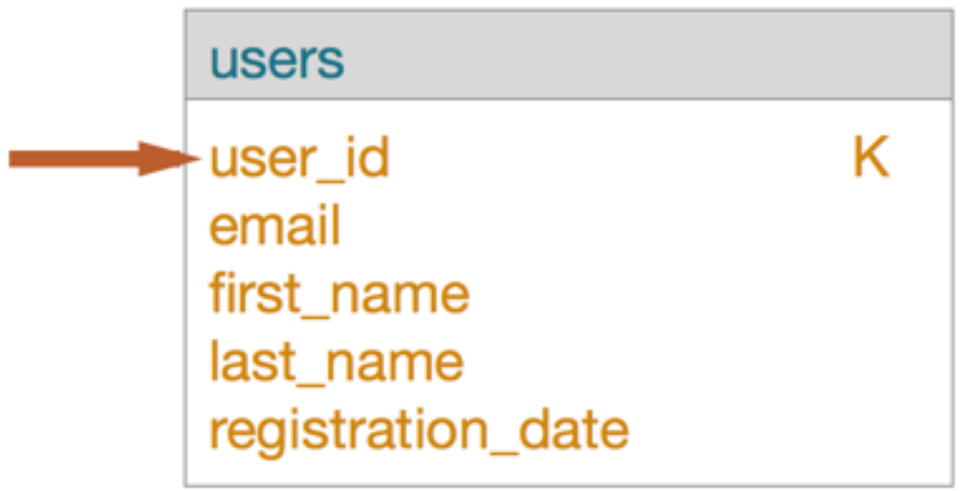
# MR I: ENTITIES AND RELATIONSHIPS

- Entity and relationship types map to tables
- Entities and relationships map to partitions or rows
- Partition may have data about one or more entities and relationships
- Attributes are represented by columns

Conceptual Model

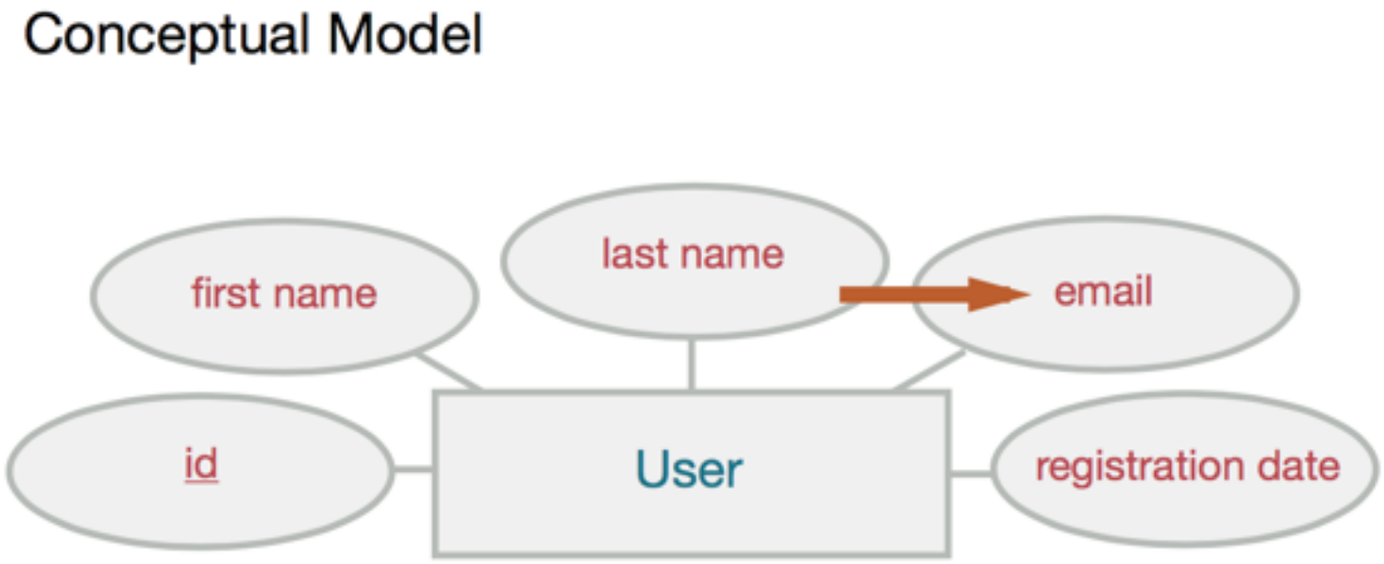



Logical Model



# MR I: ENTITIES AND RELATIONSHIPS

- Entity and relationship types map to tables
- Entities and relationships map to partitions or rows
- Partition may have data about one or more entities and relationships
- Attributes are represented by columns



# MR1: ENTITIES AND RELATIONSHIPS

- Entity and relationship types map to tables
- Entities and relationships map to partitions or rows
- Partition may have data about one or more entities and relationships
- Attributes are represented by columns

Conceptual Model


Logical Model

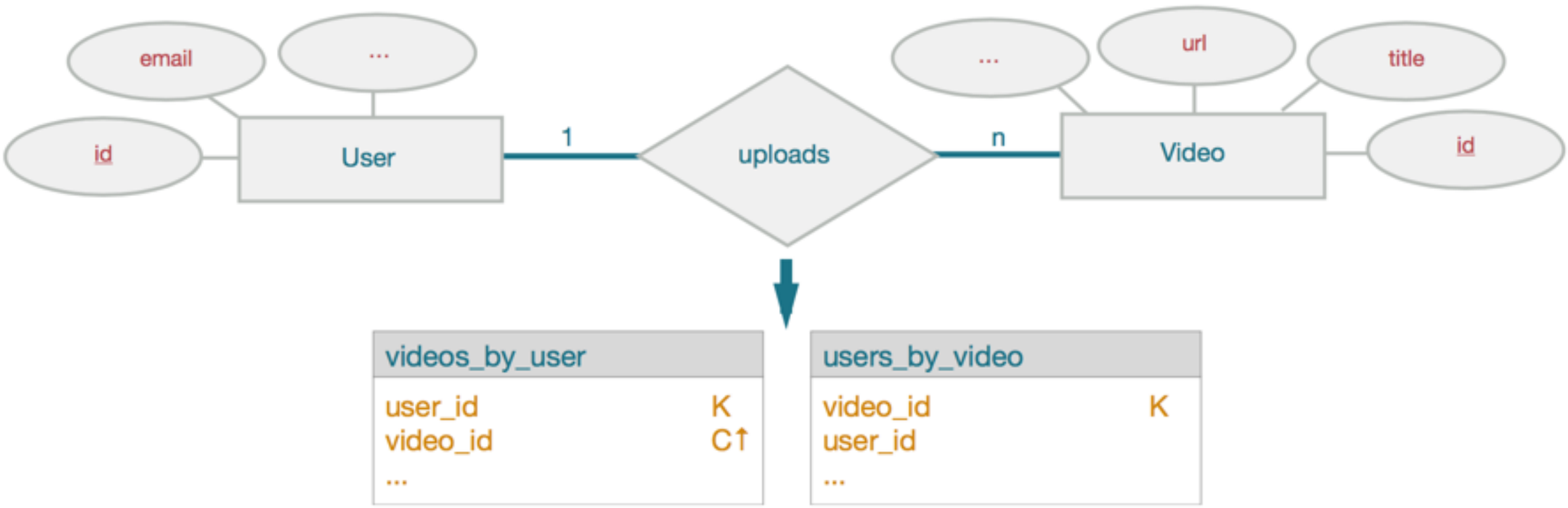
| users | |
|---|---|
| user_id | K |
| email | |
| first_name | |
| last_name | |
| registration_date | |

| user_id | email | first_name | last_name | registration_date |
|---|---|---|---|---|
| a7e78478-0a54-4949-90f3-14ec4cbea40c | jbellis@datastax.com | Jonathan | Ellis | 2010-04-01 00:00:00+0000 |

# MR1: ENTITIES AND RELATIONSHIPS

## Relationship type example

- Each relationship becomes a row in the table
- Relationship type attributes are represented by columns
- Queries and relationship cardinality affects the design of the primary key



# MR2: EQUALITY SEARCH ATTRIBUTES

## Relationship type example

- Equality search attributes become initial columns of a primary key
- Querying on: `last name` and `first name`

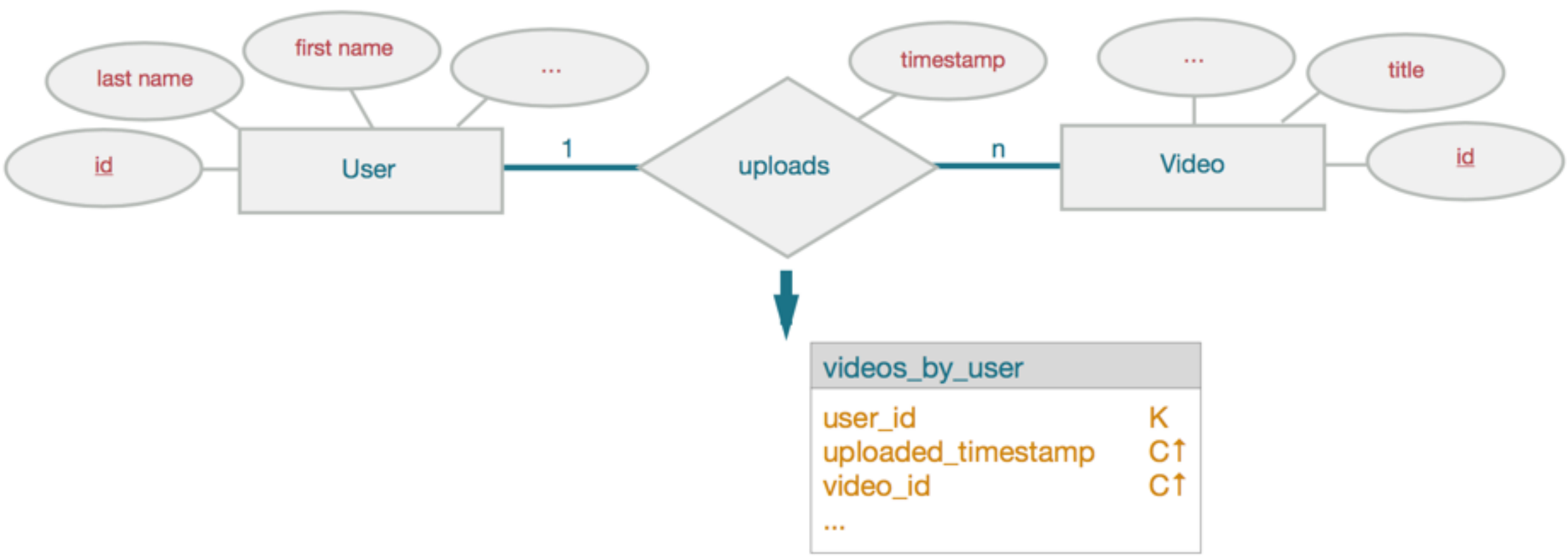

| videos_by_user_1 | |
|---|---|
| last_name | K |
| first_name | K |
| video_id | C↑ |
| ... | |

| videos_by_user_2 | |
|---|---|
| last_name | K |
| first_name | C↑ |
| video_id | C↑ |
| ... | |

# MR3: INEQUALITY SEARCH ATTRIBUTES

## Relationship type example

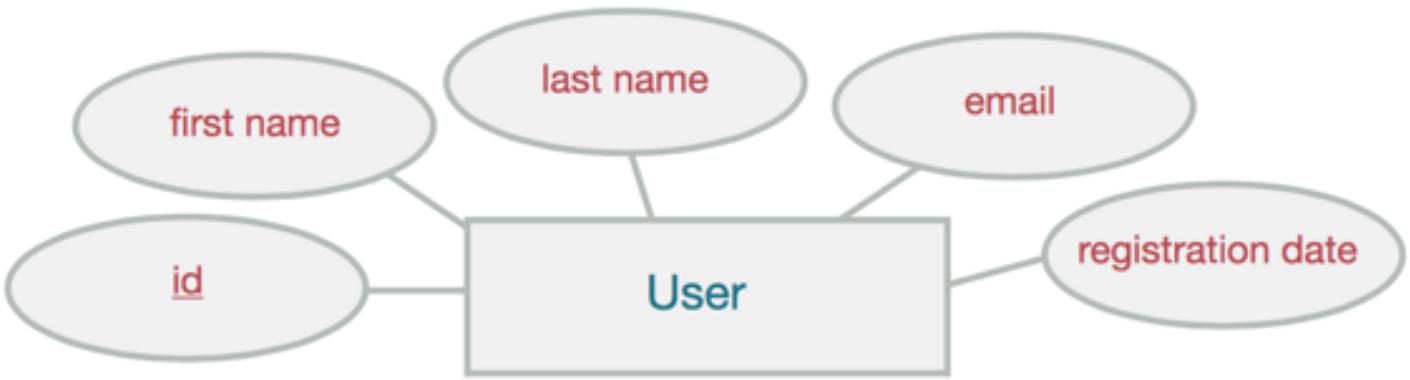
- Inequality search attributes become clustering columns
- Querying on: `user_id` = *?* *and* `uploaded_timestamp` > *?*



# MR4: ORDERING ATTRIBUTES

## Entity type example

- Ordering attributes become clustering columns
- Querying on: `last_name` = *?* *and* `registration_date` > *?*
- Ordering attributes: `registration_date` *(ASC)*

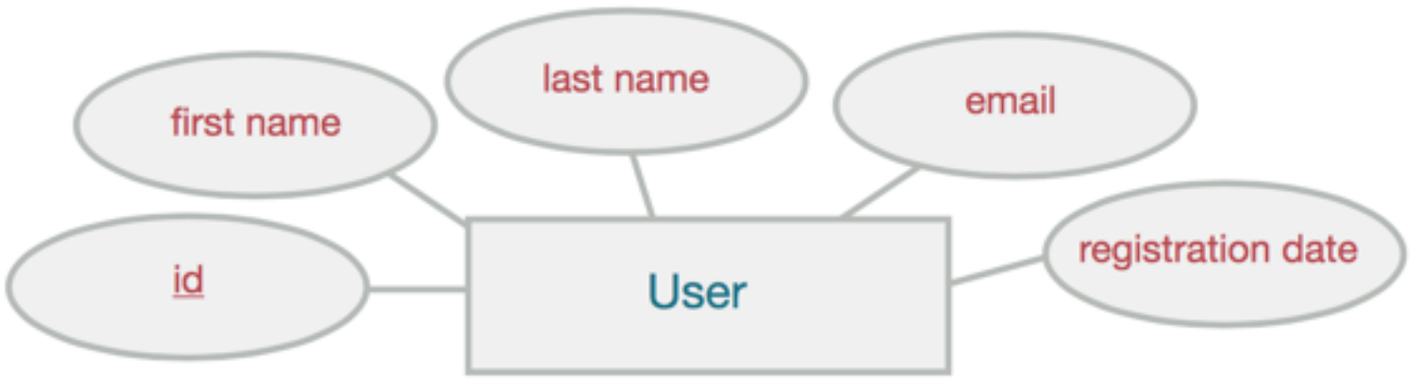



| users_registration_date | |
|---|---|
| last_name | K |
| registration_date | C↑ |
| user_id | C↑ |
| first_name | |
| email | |

# MR5: KEY ATTRIBUTES

## Entity type example

- Entity type key attributes are included as primary key columns
- Queries also affect primary key design



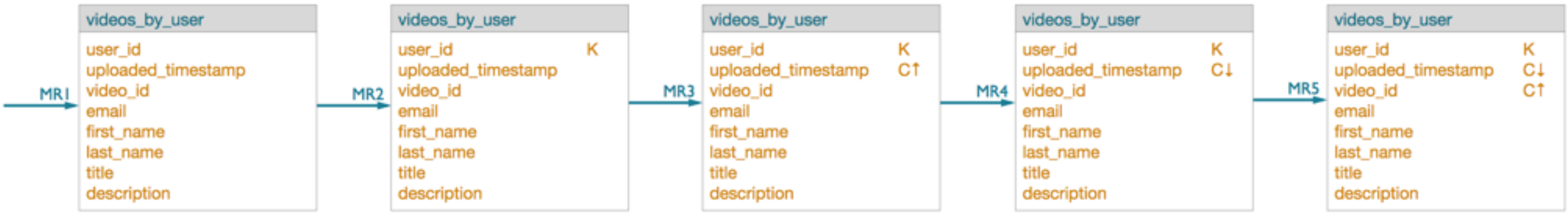

| users | |
|---|---|
| user_id | K |
| email | |
| first_name | |
| last_name | |
| registration_date | |

| users_by_name | |
|---|---|
| last_name | K |
| first_name | K |
| user_id | C↑ |
| email | |
| registration_date | |

# APPLYING MAPPING RULES

- Create a table schema from the conceptual data model and for each query
- Apply the mapping rules in order—MR1—MR2—MR3—MR4—MR5


Query: user_id = ? and uploaded_timestamp > ?
ORDER BY uploaded_timestamp DESC



# PHYSICAL DATA MODEL

# WHAT TO ANALYZE

- Finding the problem
    - Partition size
    - Data redundancy
    - Data consistency
    - Application-side joins
    - Referential integrity constraints
    - Transactions
    - Data aggregation

# DESIGNING THE MODEL

## Finalizing the physical model

| comments_by_user | |
|---|---|
| user_id | K |
| posted_timestamp | C↓ |
| video_id | C↑ |
| comment | |
| title | |
| type | |
| {tags} | |
| <preview_thumbnails> | |

| comments_by_user | | |
|---|---|---|
| user_id | UUID | K |
| posted_timestamp | TIMESTAMP | C↓ |
| video_id | TIMEUUID | C↑ |
| comment | TEXT | |
| title | TEXT | |
| type | TEXT | |
| {tags} | SET<TEXT> | |
| <preview_thumbnails> | MAP<INT,BLOB> | |

# CREATING TABLES

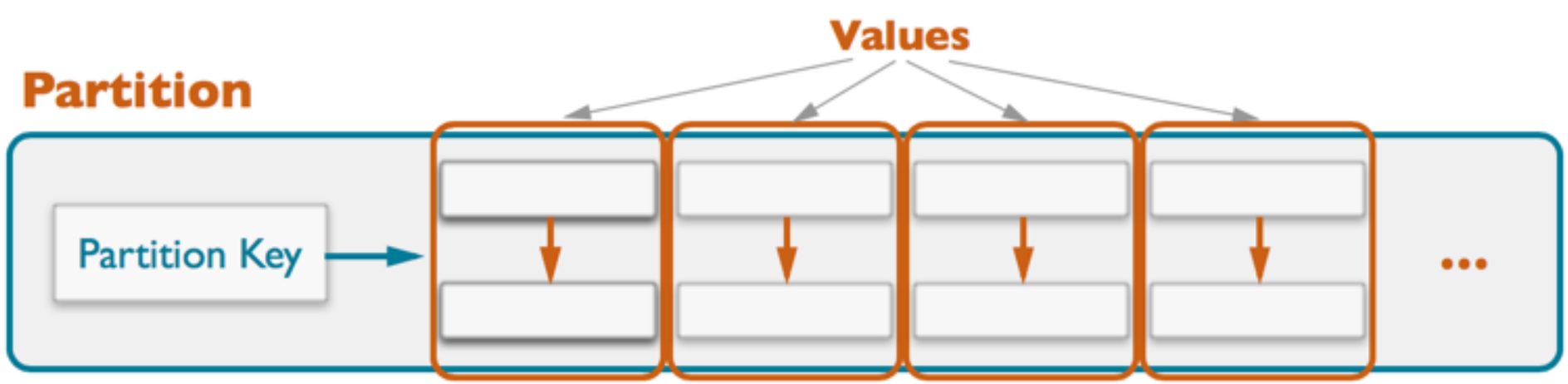
## Writing the CQL statements

```
CREATE TABLE comments_by_user (
        user_id UUID,
        posted_timestamp TIMESTAMP,
        video_id TIMEUUID,
        comment TEXT,
        title TEXT,
        type TEXT,
        tags SET<TEXT>,
        preview_thumbnails MAP<INT, BLOB>,
        PRIMARY KEY ((user_id), posted_timestamp, video_id)
) WITH CLUSTERING ORDER BY (posted_timestamp DESC, video_id ASC);
```

# PARTITION SIZE

Implementation limit on the size of a partition in Cassandra

- Number of values in a partition — 2 billion




- Size of the partition on disk — must be able to fit entirely on a node

# COMPARING CASSANDRA (C*) AND RDBMS

- with RDBMS, a normalized data model is created without considering the exact queries
    - SQL can return almost anything though Joins
- With C*, the data model is designed for specific queries schema is adjusted as new queries introduced
- C*: NO joins, relationships, or foreign keys
    - a separate table is leveraged per query
    - data required by multiple tables is denormalized across those tables

CQL

*stands for*

# Cassandra Query Language

```
cqlsh> help

Documented shell commands:
==========================
CAPTURE  CLS          COPY  DESCRIBE  EXPAND  LOGIN   SERIAL  SOURCE   UNICODE
CLEAR    CONSISTENCY  DESC  EXIT      HELP    PAGING  SHOW    TRACING

CQL help topics:
================
AGGREGATES                CREATE_KEYSPACE           DROP_TRIGGER      TEXT
ALTER_KEYSPACE            CREATE_MATERIALIZED_VIEW  DROP_TYPE         TIME
ALTER_MATERIALIZED_VIEW   CREATE_ROLE               DROP_USER         TIMESTAMP
ALTER_TABLE               CREATE_TABLE              FUNCTIONS         TRUNCATE
ALTER_TYPE                CREATE_TRIGGER            GRANT             TYPES
ALTER_USER                CREATE_TYPE               INSERT            UPDATE
APPLY                     CREATE_USER               INSERT_JSON       USE
ASCII                     DATE                      INT               UUID
BATCH                     DELETE                    JSON
BEGIN                     DROP_AGGREGATE            KEYWORDS
BLOB                      DROP_COLUMNFAMILY         LIST_PERMISSIONS
BOOLEAN                   DROP_FUNCTION             LIST_ROLES
COUNTER                   DROP_INDEX                LIST_USERS
CREATE_AGGREGATE          DROP_KEYSPACE             PERMISSIONS
CREATE_COLUMNFAMILY       DROP_MATERIALIZED_VIEW    REVOKE
CREATE_FUNCTION           DROP_ROLE                 SELECT
CREATE_INDEX              DROP_TABLE                SELECT_JSON

cqlsh>
```

# BASIC DATA TYPES IN CASSANDRA

- Numeric
    - bigint, decimal, double, float, int, varint
- String
    - ascii, text, varchar
- Date
    - timestamp, timeuuid
- Other
    - boolean, uuid, inet, blob

# NAMING YOUR KEYSPACES, TABLES AND COLUMNS

- No hyphens: 2016-sales
- No spaces: 2015 sales
- Double quotes required for initial digits: "2016sales"
- Mixed case is lowercased unless surrounded in double quotes: "firstName"

# CQL

- creating a keyspace - namespace of tables
    - CREATE KEYSPACE demo WITH replication = {‘class’: ’SimpleStrategy’, replication_factor’: 3};
- To use namespace:
-  USE demo;

# CREATING TABLES

- creating tables:

```
CREATE TABLE users(
    email varchar,
    bio varchar,
    birthday timestamp,
    active boolean,
PRIMARY KEY (email));
```

```
CREATE TABLE tweets(
    email varchar,
    time_posted timestamp,
    tweet varchar,
    PRIMARY KEY (email, time_posted));
```

# QUERYING TABLES

- SELECT expression reads one or more records from Cassandra column family and returns a result-set of rows
- SELECT * FROM users;
- SELECT email FROM users WHERE active = true;
- SELECT email FROM users WHERE active = true LIMIT 100;

# INSERTING/UPDATING DATA

- INSERT INTO users (email, bio, birthday, active) VALUES ('john.crost@cnn.com', 'Coach', 646464676600, true);

- UPDATE users SET email='john.crost@cnn.com', bio='Coach', birthday=646464676600, active=true)

# UPSERT

- Under the hood, INSERT and UPDATE are same
- INSERT = UPDATE
- BOTH OPERATIONS REQURE PRIMARY KEY

# WHEN WAS DATA WRITTEN?

- SELECT email, WRITETIME(bio) FROM users;
  - Unix time (e.g. 1430825689)

# DELETING DATA

- Deleting a row
    - DELETE FROM customers WHERE id = '2829';
- Deleting a column
    - DELETE email FROM customers
      WHERE id = '2829';
    - UPDATE customers SET email = null
      WHERE id = '2829';
    - INSERT INTO customers (id, email)
      VALUES ('2829', null);

# DEMO

- Cassandra query language

- http://www.planetcassandra.org/try-cassandra/

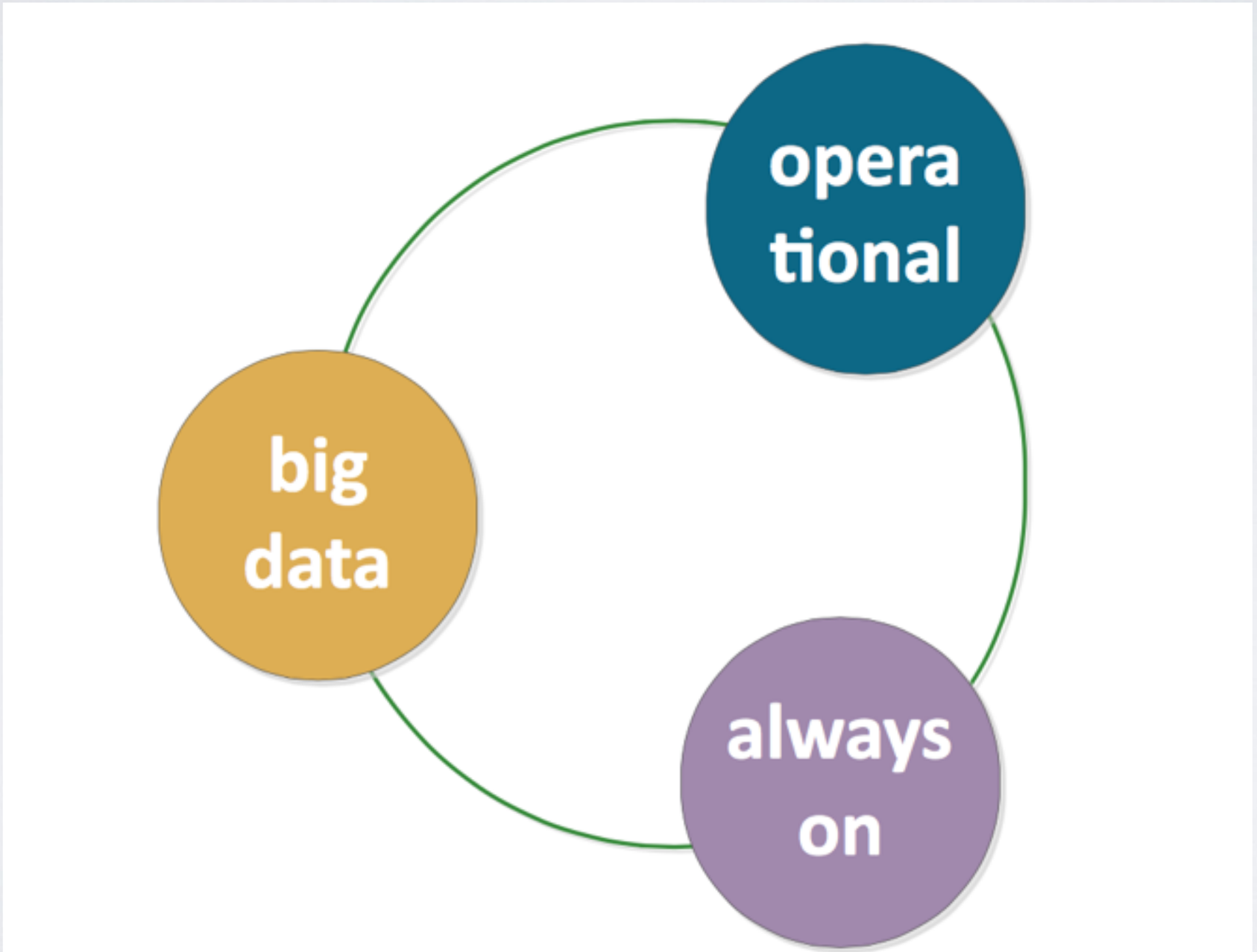
# CASSANDRA ADVANTAGES

- Perfect for time-series data
- High performance
- Decentralization
- Nearly linear scalability
- Replication support
- No single points of failure
- MapReduce support

# CASSANDRA WEAKNESSES

- No referential integrity
- No concept of JOIN
- Querying options for retrieving data are limited
- Sorting data is a design decision
- No GROUP BY
- No support for atomic operations
- If operation fails, changes can still occur
- First think about queries, then about data model

# USECASES

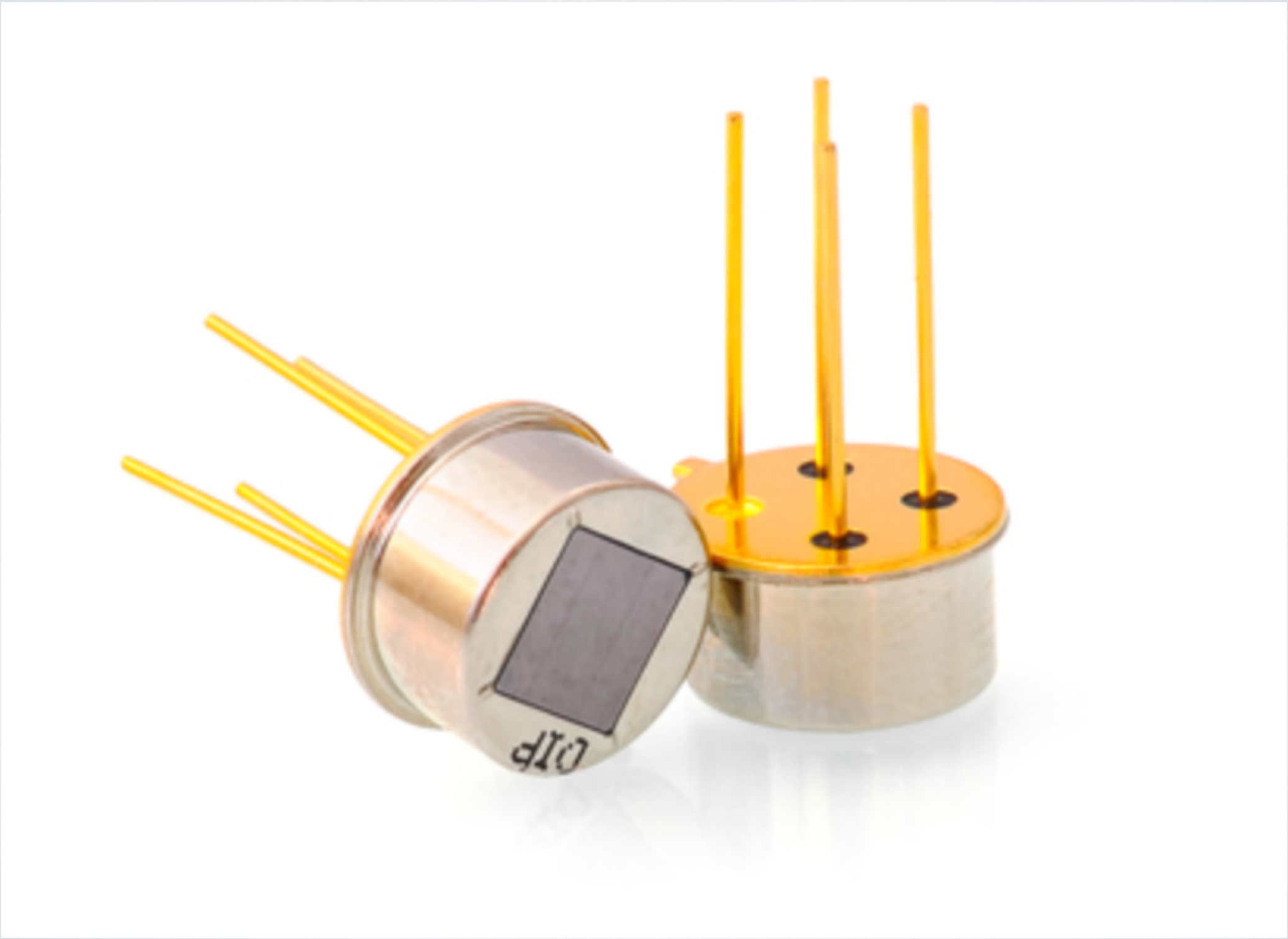


# MAIN FUNCTIONAL USE CASES

| Use Case | Example |
| --- | --- |
| Product catalogs and playlists | Collection of items |
| Internet of things | Sensor data |
| Messaging | Emails, instant messages, alerts, comments |
| Recommendation and personalization | Trend information |
| Fraud detection | Data patterns |

# WHAT COMPANIES?

# SENSOR DATA USECASE



# REFERENCES

- http://docs.datastax.com/en/landing_page/doc/landing_page/current.html
- http://spark.apache.org/
- http://cassandra.apache.org/
- http://www.planetcassandra.org/
- http://demo.gethue.com/home