



# Data Engineering LTAT.02.007

Ass Prof. Riccardo Tommasini
Fabiano Spiga, Hassan Eldeeb, Mohamed Ragab

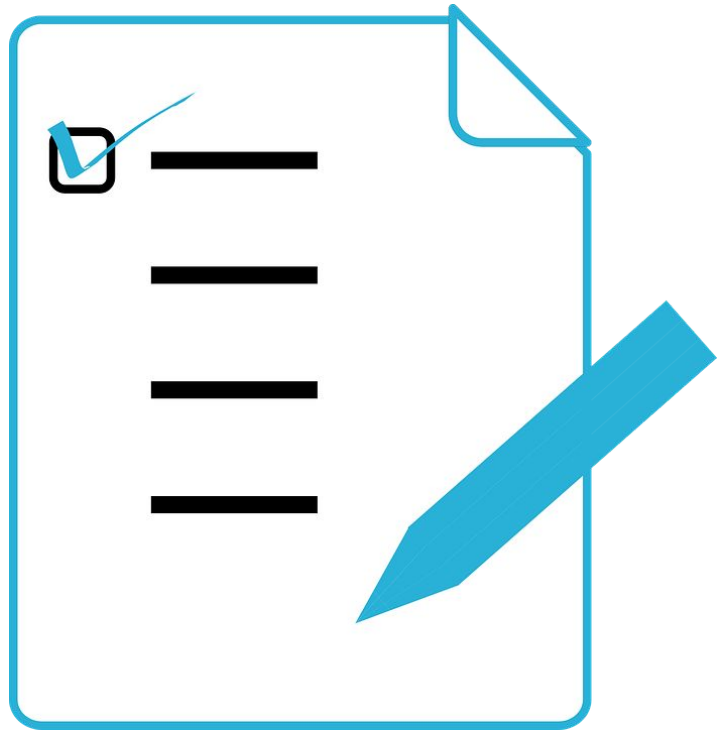
## Key-Value DBS/Stores (REDIS)

Lab. 03

https://courses.cs.ut.ee/2020/dataeng

# Lab Agenda

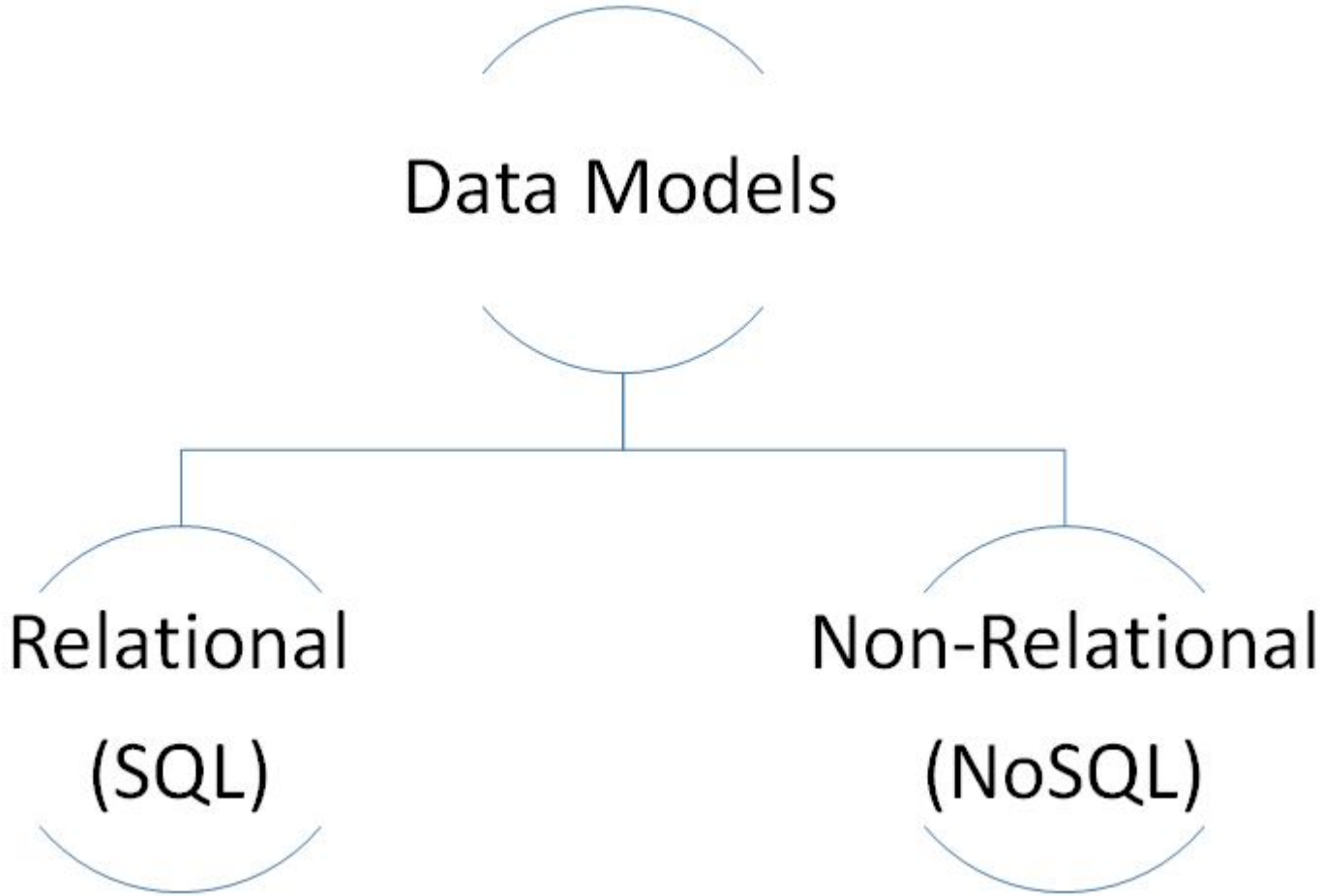
- SQL is Great but...
- NOSQL DBS
- KEY-VALUE Stores
- REDIS
  - What is REDIS?
  - Why We need Redis?
  - REDIS Caching
  - REDIS PUB/SUB
- REDIS Data-Types



# SQL is great but…

- Transactions, Integration, Reporting,..etc.
- Sometimes SQL is not the fit for what we are doing.
- One Size can’t fit all!
- The data isn’t always relational/tabular.
- With the rise of the WWW, Semi-structured or unstructured data can’t fit in tables.

# Data Models

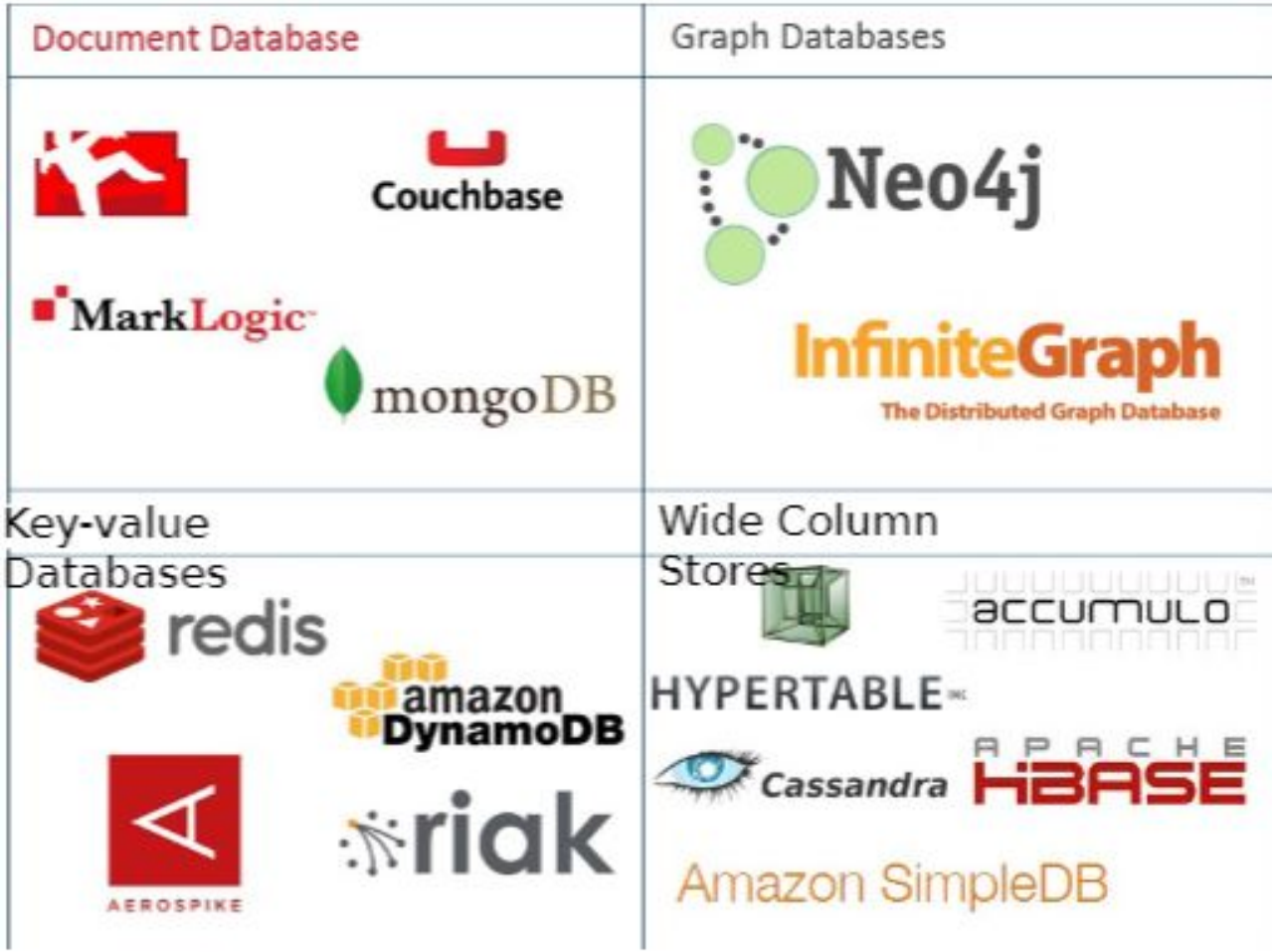


# What NoSQL means?

- Refers to "not-only-SQL" or "non-relational" database.
- NOSQL refer to databases that **don’t support** the **relational** model and **not use SQL**.
- These non-relational DBMS are designed to solve problems where relational databases are not adequate.

# Overview – NoSQL Family

- Data stored in **4 types**:
  - Key-Value pairs
  - Document (JSON, XML,..)
  - Graph
  - Wide-column



# RDBs(SQL) VS. NoSQL (Key Differences)

- **Data Models and Schema**
  - Structured Tabular VS. unstructured data models .
  - Fixed/Predefined VS. dynamic schema
- **Data Structures**
  - Table-based VS. document-based, graph, key-value, wide columns.
- **Scaling**
  - Vertically scalable, typically expensive **--->**
  - Horizontally scalable, much cheaper
- **Development Model**
  - Typically closed-source with licencing fees VS. Open- Source community.

https://www.pinterest.com/pin/161355599127763483/?nic_v1=1a3Uvq1cr

# Key-Value Databases

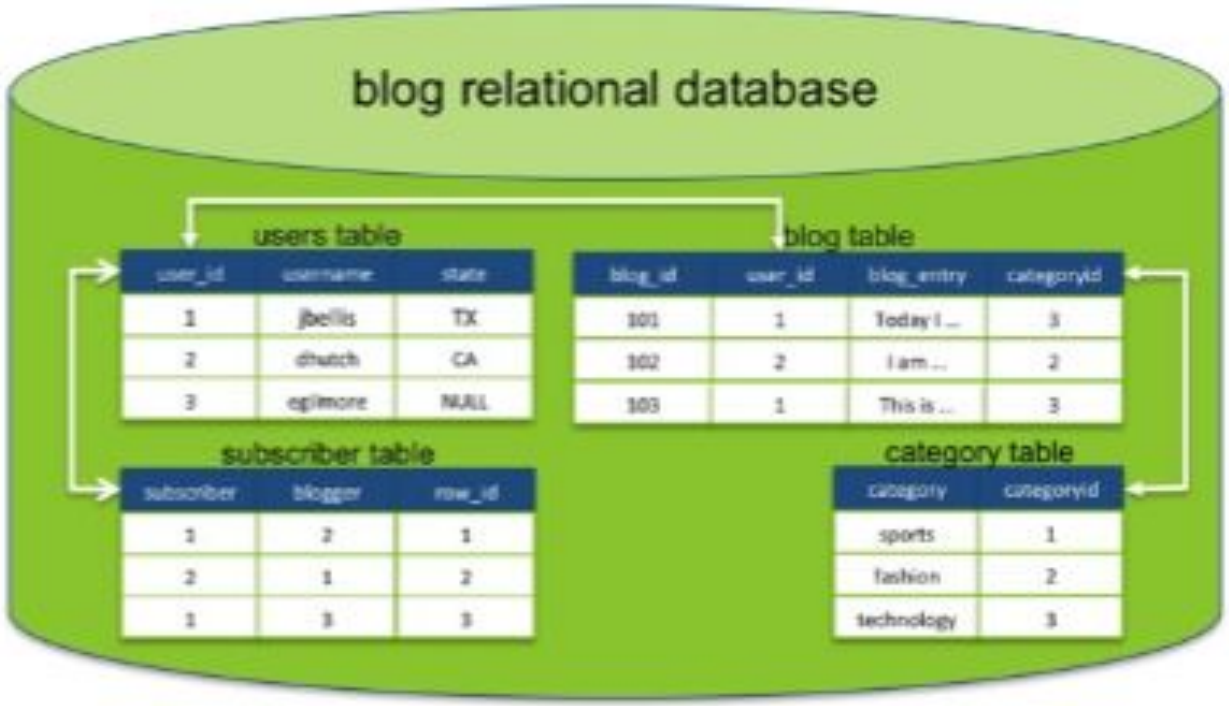
# Why Key-Value Stores?

- Essentially Key-Value stores assign a key to a value.
- Examples:
  - (Twitter.com) tweet_id □ information about the tweet.
  - (amazon.com) item_number□ information about the item
  - (yourbank.com) account_number□ account details
  - …etc.

https://www.coursera.org/learn/cloud-computing

# Isn’t that just a database?!

- Queried using SQL
- Key-based
- Foreign keys
- Indexes
- Joins




SQL queries: SELECT user_id from users
WHERE
username = “jbellis”

https://www.coursera.org/learn/cloud-computing

# Mismatch with Today's Workloads

- **Data: Large and Unstructured**
  - Sometimes there's no predefined schema
- **Lots of random reads and writes**
  - RDBS are not optimized for millions of concurrent reads/writes.
- **Sometimes write-heavy**
  - RDBs are typically optimized for **read-heavy** workloads
- **Foreign-keys rarely needed, Joins infrequent as well**
  - Sometimes what we need is much simpler than this.

https://www.coursera.org/learn/cloud-computing

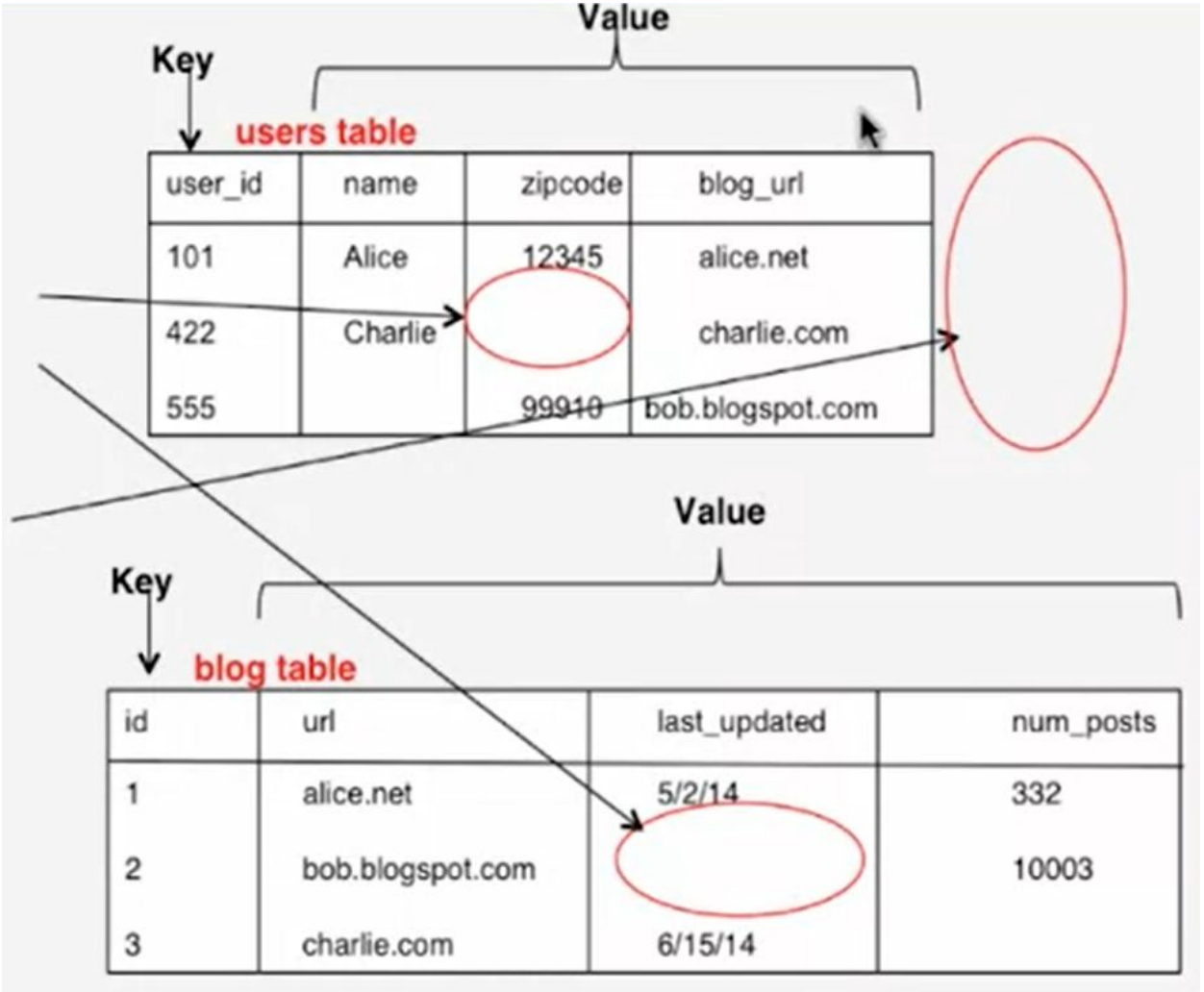
# Needs of Today's Workloads

- SPEED
- Avoid Single Point of Failure (SPoF)
- Low TCO (Total Cost of Operation)
- Fewer System administrators
- Incremental Scalability
- Scale out, not up
  - What?

https://www.coursera.org/learn/cloud-computing

# What is Key-Value Store?

- Unstructured
- No Schema imposed
- Columns missing from some rows
- No foreign keys
- Joins may not be supported as well




https://www.coursera.org/learn/cloud-computing

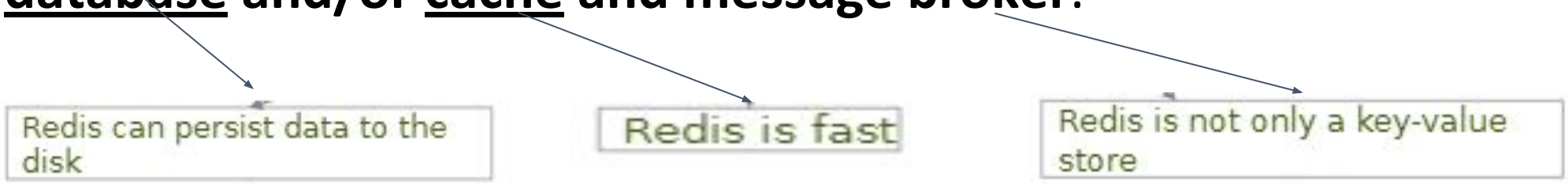
redis

REmote DIctionary
Server

https://redis.io/

# What is Redis ?

- ✔ Redis, standing for **REmote DIctionary Server**,
- ✔ **Open Source, in-memory data structure store** which can be used as a **database and/or cache and message broker**.




- ✔ Supports **multiple** data structures (Strings, Hashes, Lists, Sets,…)
- ✔ Built-in “**Master-Slave**” Replication.

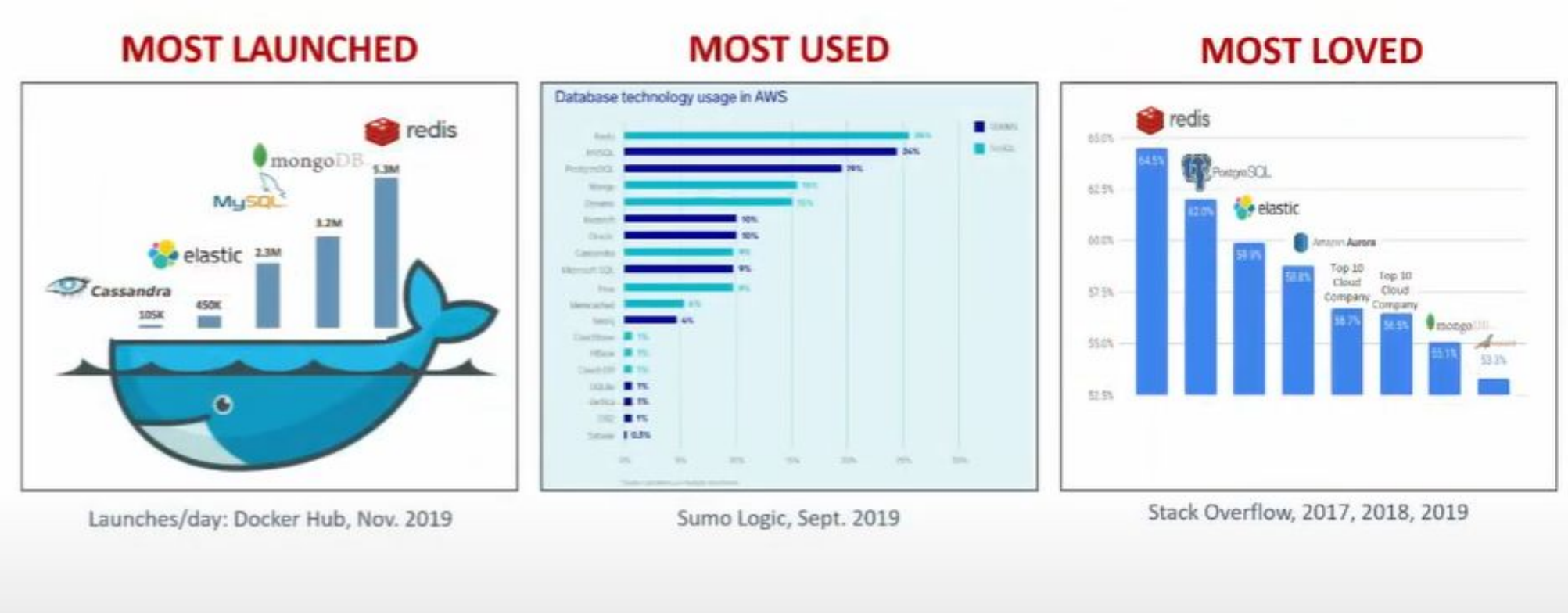
https://redis.io/
https://redis.io/topics/quickstart

# Why We need/use Redis?

- FAST
- NOT CPU Intensive
- Well Documented
- Simple and Versatile
- Scalable

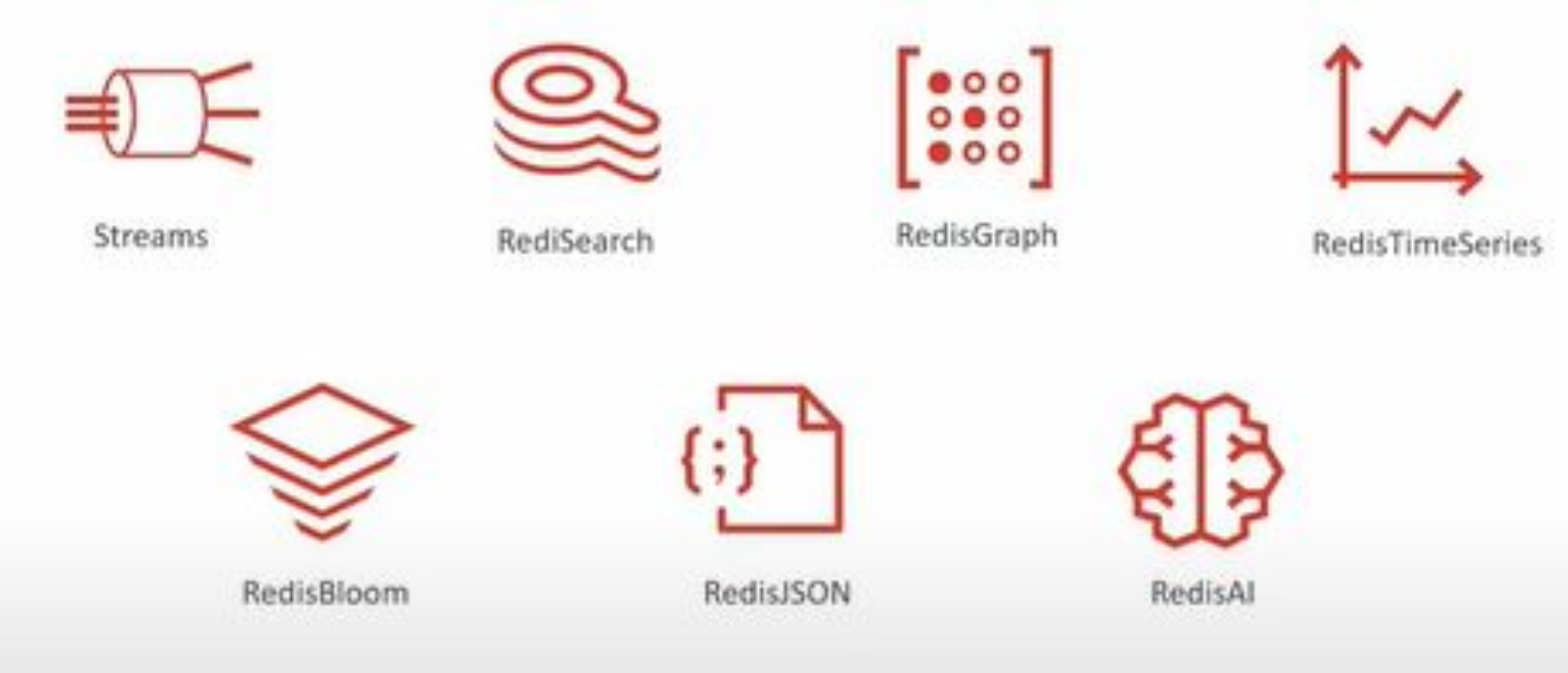

https://redis.io/documentation

# HOW Popular is REDIS?

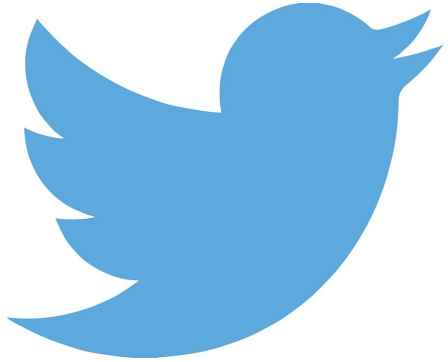



https://www.youtube.com/watch?v=rQNn5i6kiAA&ab_channel=RedisLabs

# Modern Models of REDIS



# Who Uses REDIS?



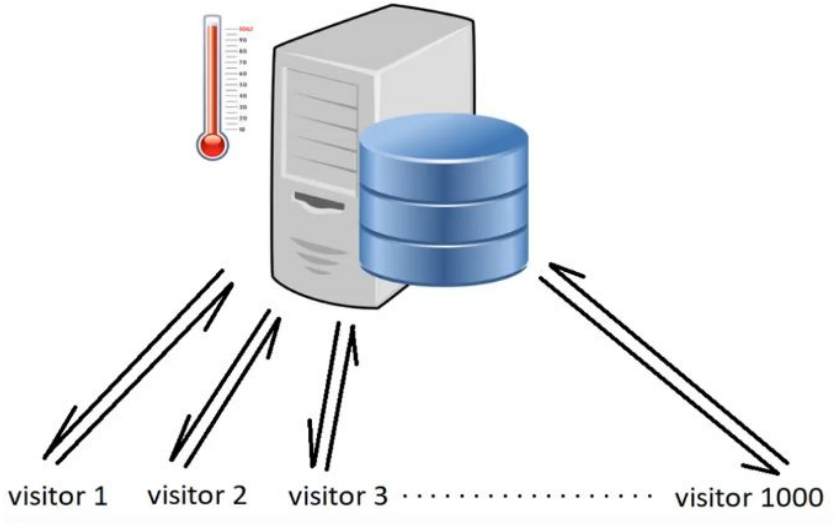


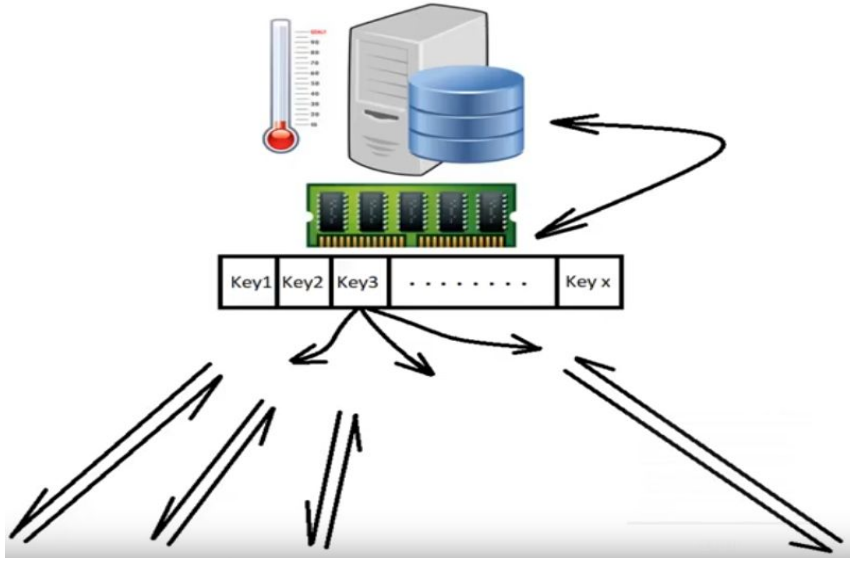
# REDIS Common Use Cases

- Caching
- Analytics (e.g. Counting things)
- Leaderboard (voting applications)
- Messaging & Pub/Sub
- Queues
- Cookie storage & **Expiring/Vanishing** Data
- Distributing Locks
- High I/O Workloads

# Client Side Caching

- A technique used  to create high performance services.
- Normally, when some data is required, the application servers will ask the database about such information.
- When client side caching is used,
  - The application will store the reply of popular queries **directly inside the application memory(in REDIS)**.
  - so that it can reuse such replies later, without contacting the database again.

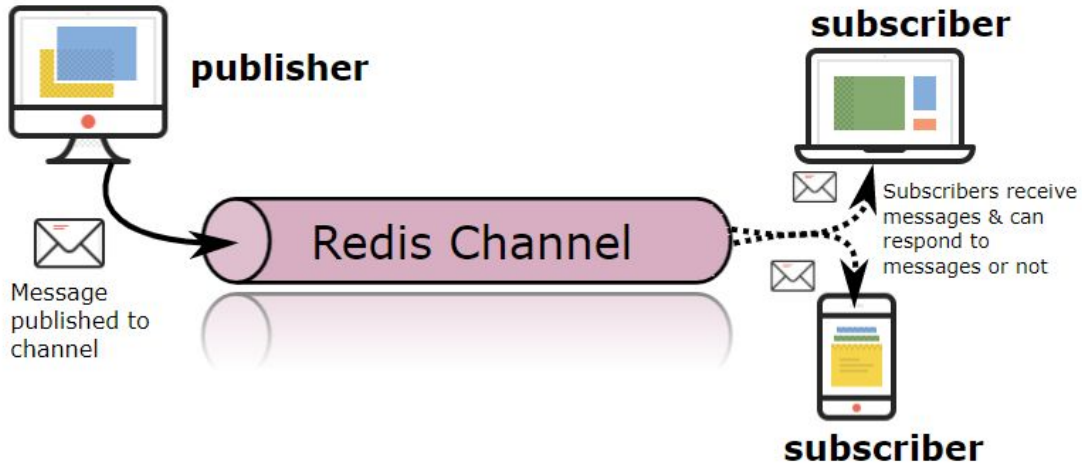


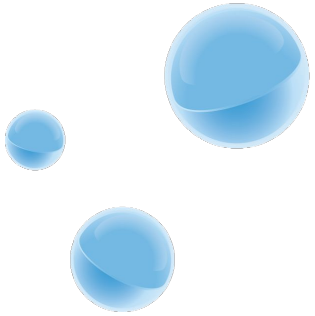
# PUB/SUB in REDIS

- It is mainly characterized by listeners subscribing to channels, with publishers sending **binary string messages** to **channels**.

- The communication is processed by the **broker,** it helps publisher to deliver that information to the subscriber(s).



# Other Caching Tools/DBs



VARNISH

CACHE

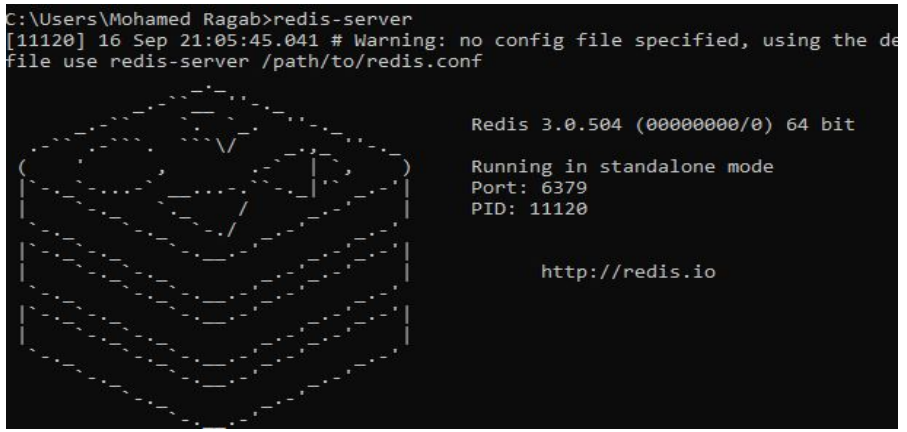

https://memcached.org
/https://varnish-cache.org/

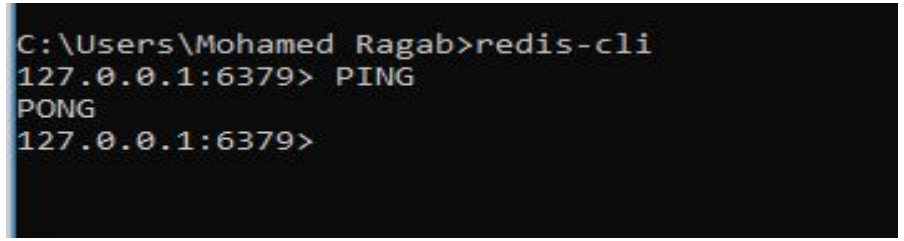
# When to Consider REDIS?

- Speed is critical
- More than just key-value pairs
- Dataset can fit in memory
- Dataset is not critical

# How to access REDSI (Redis Interfaces)

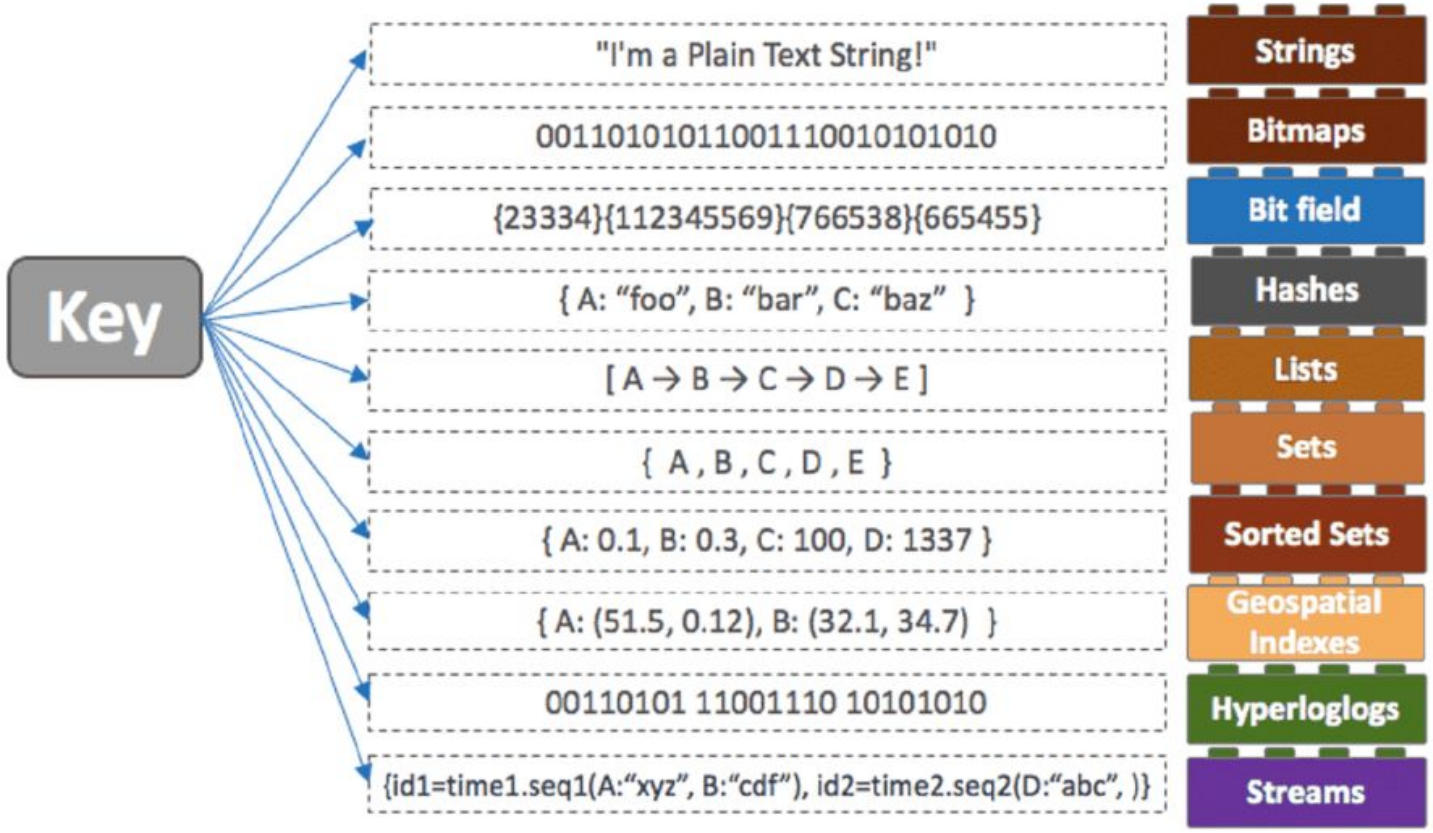
- Try REDIS Online Service
  - This link
- Using  the terminal / any other CLIs.
- Using Redis from Applications (python,Ruby,…)

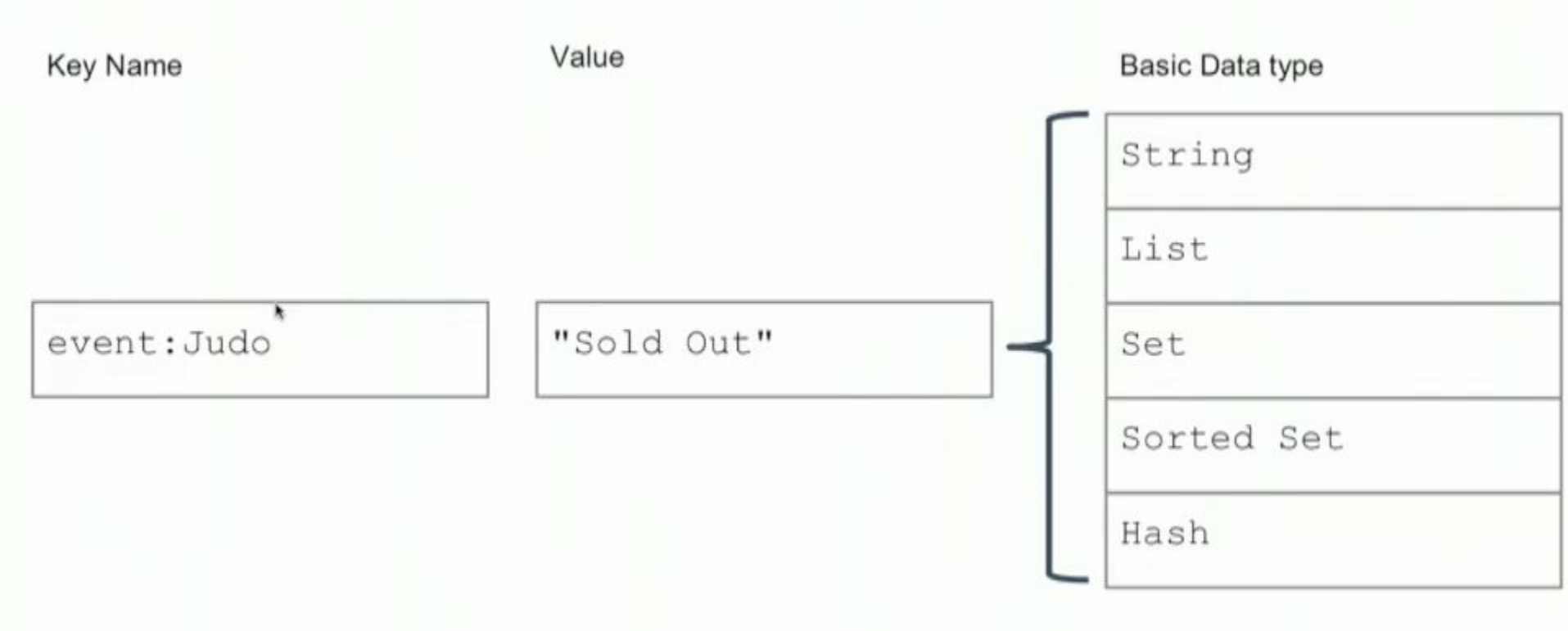

https://redis.io/topics/quickstar t

# REDIS data types



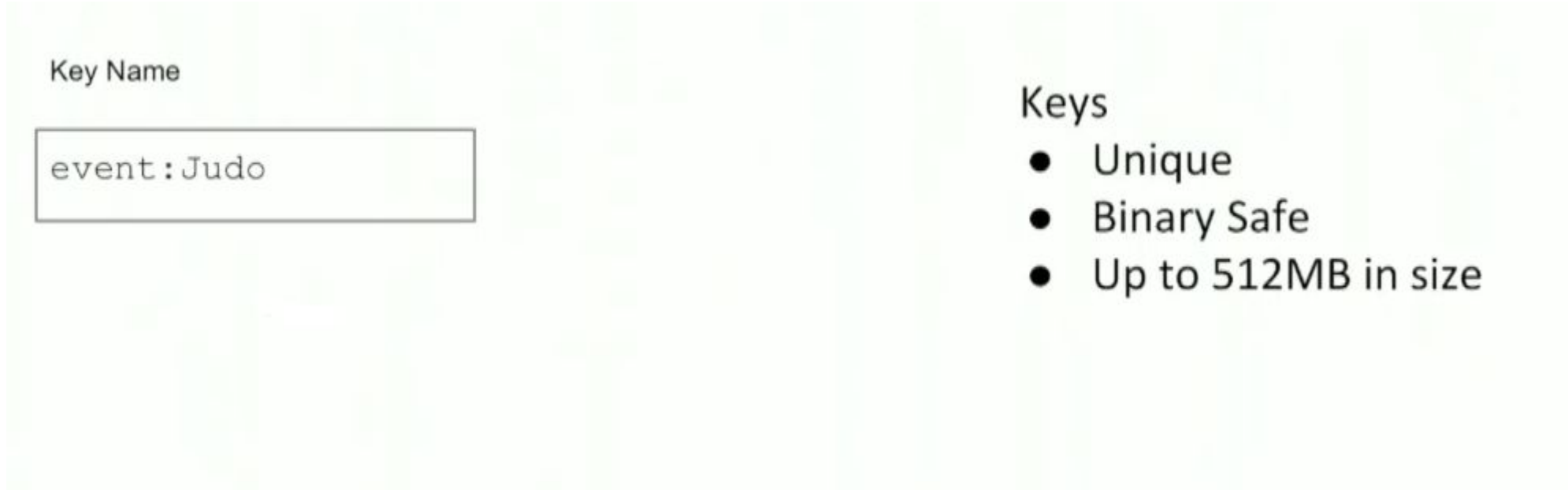

https://redis.io/topics/data-types-intro

# KEYs, VALUEs && Data Types



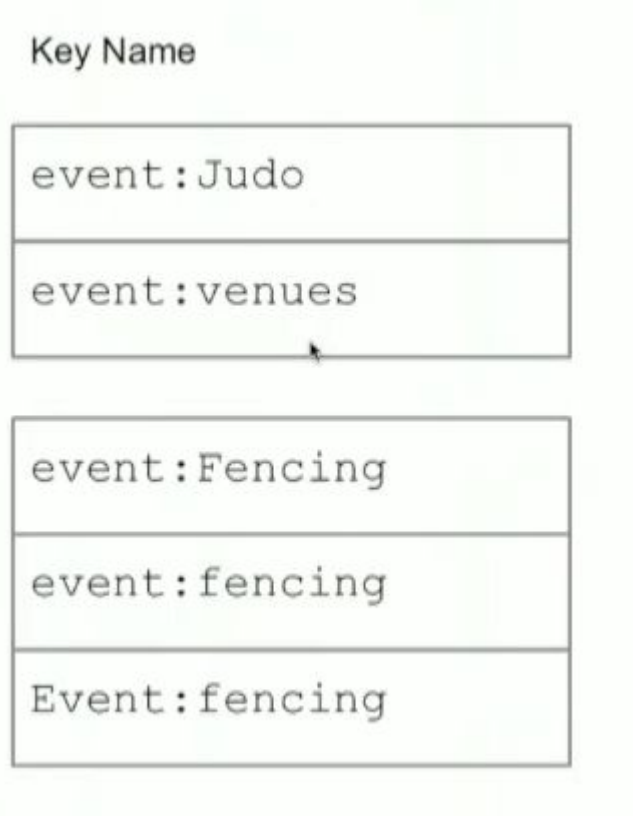

https://youtu.be/ELk_W9BBTDU

# REDIS KEYS


Key Name
event:Judo
Keys
Unique
Binary Safe
Up to 512MB in size

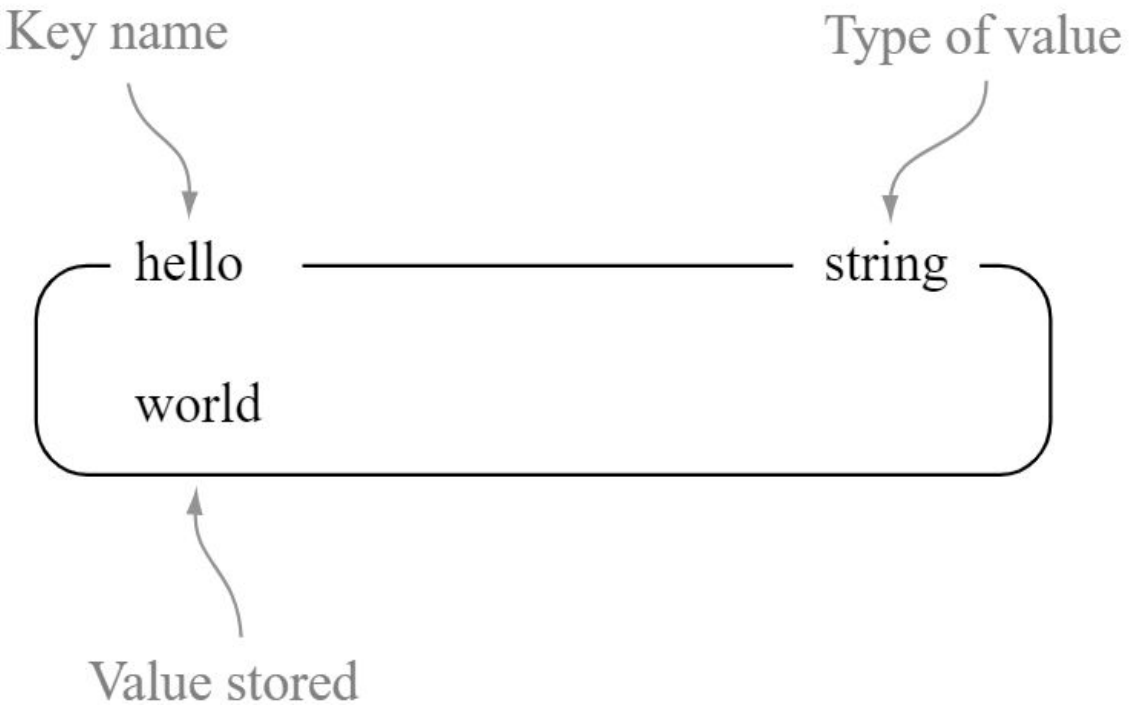
# REDIS KEY Naming

- Up to you!
- Be consistent.
- Colon is a typical community convention
  - Domain Object Name
  - Colon
  - Unique ID
- Plurals are nice for Lists, Sets
- Case Sensitive



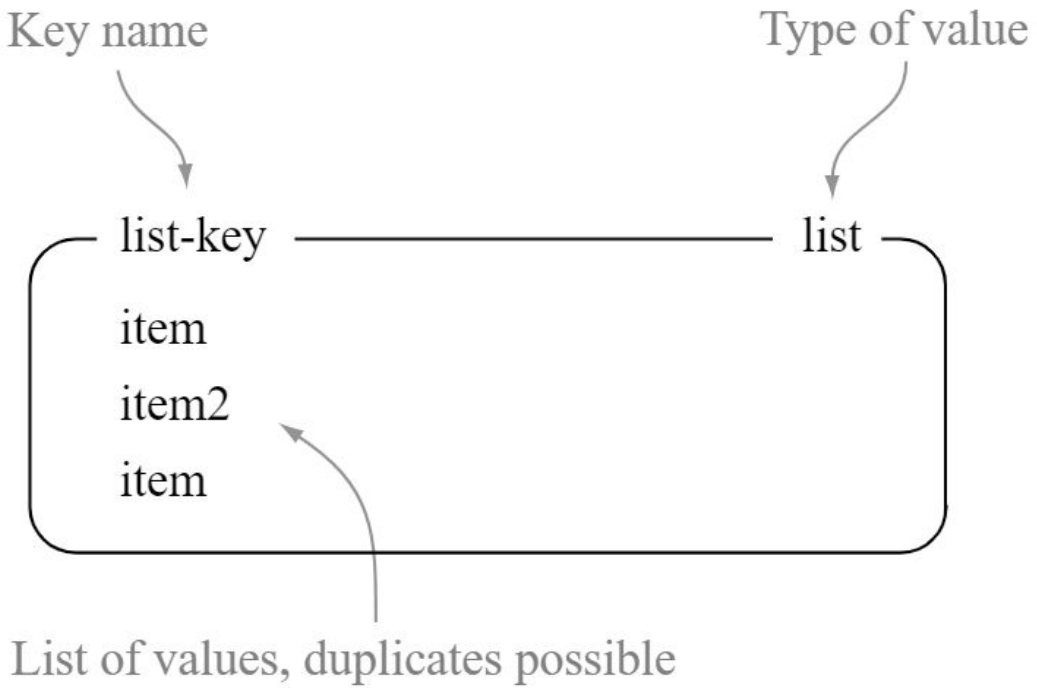
# STRINGS

- The most important type.
- Store Strings, numerical values, serialized JSON, and binary.
  - **Serialization**, and **deserialization**.
- Commonly used for caching when combined with the **EXPIRE** command.
- Strings can handle up to size **512MB**.



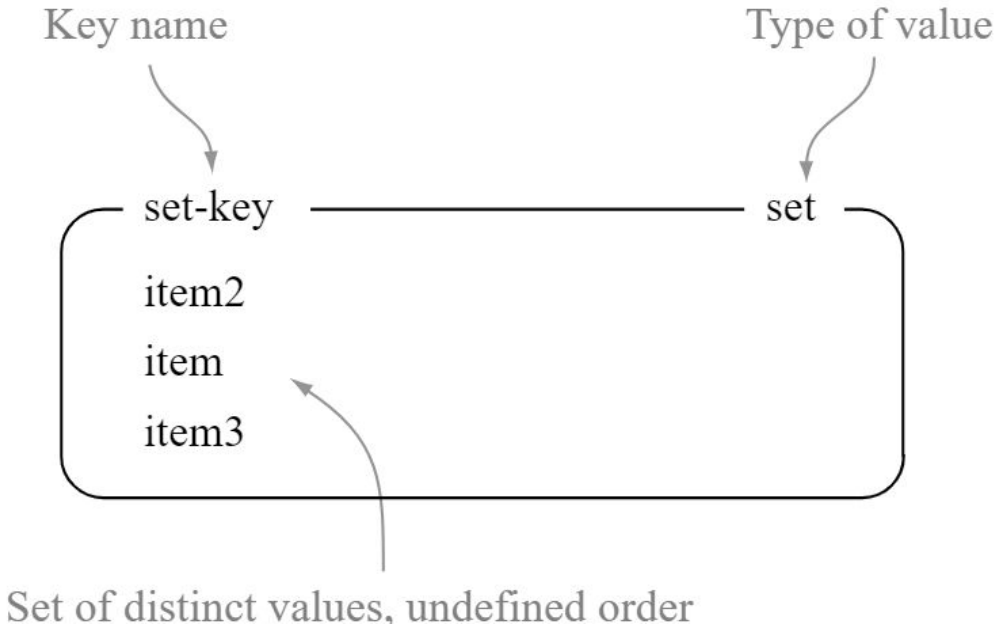
# LISTS

- Think of Lists as an ordered sequence of strings like java ArrayList, Javascript array, or python n lists.
- We can use lists to implement stacks and queues.
- Lists accept duplicates.



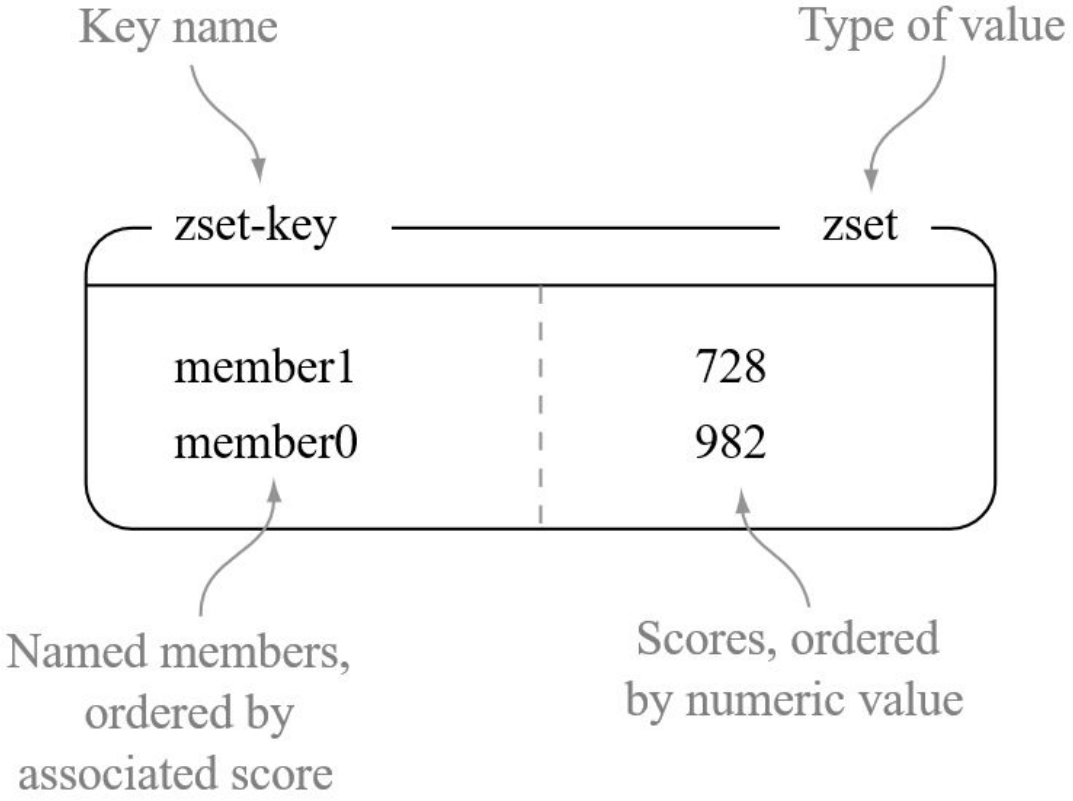
# SETS

- Unordered collection of strings
- Contains no duplicates.
  - This makes Sets supernatural option for de-duplication applications.
- Questions that we can Answer using Sets:
  - Did I see this IP address in the last hour?
  - Is this user online?
  - Has this URL been blacklisted?
- Sets support standard operations:
  - Intersection
  - Difference
  - Union



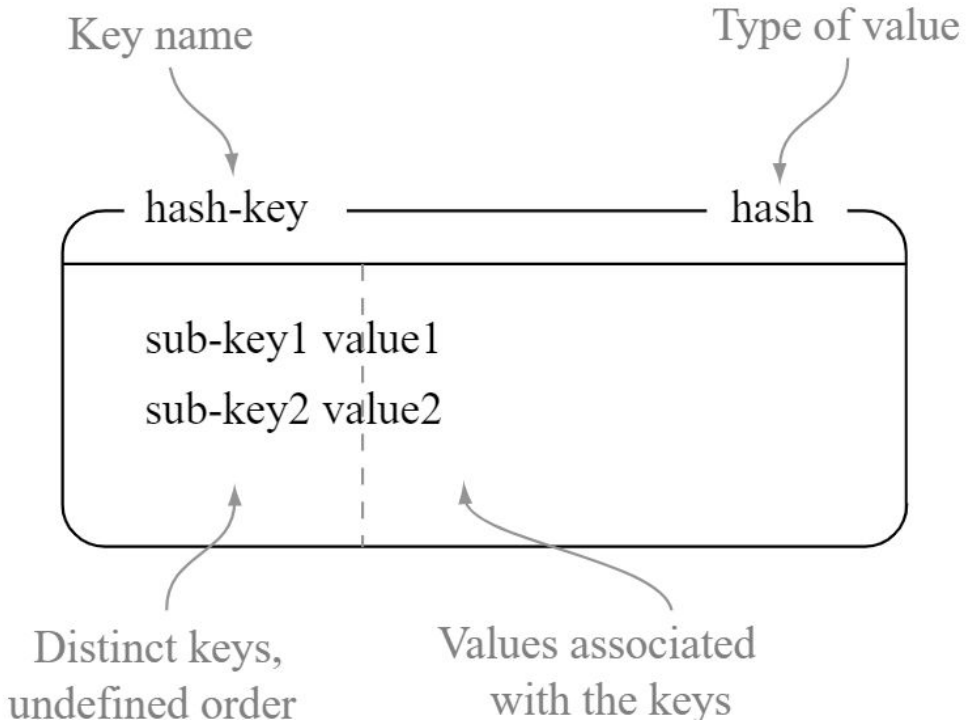
# SORTED SETS

- Ordered collections of unique members.
- These members are ordered according to their associated score.
- Whenever you add to the sorted set, you are specifying a member and a score.
- Sorted Sets keep everything sorted from the beginning.
- Sorted sets are good choice for:
  - priority queues, Low-latency leaderboards, Secondary indexing
- Questions that we can Answer using Sorted Sets (e.g, in an online-game ):
  - Who are the top 10 players?
  - what is the rank of a specific Player?
  - what is the current score of the player?



# HASHES

- Collections of Keys-Value pairs.
- Similar to JSON object, Java HashMAPs, and Python Dictionary.
- They are Mutable:
  - ADD,CHANGE,INCREMENT,REMOVE field value-pairs anytime.
- Hashes are Schema-less.



# Now, It’s time to say ...