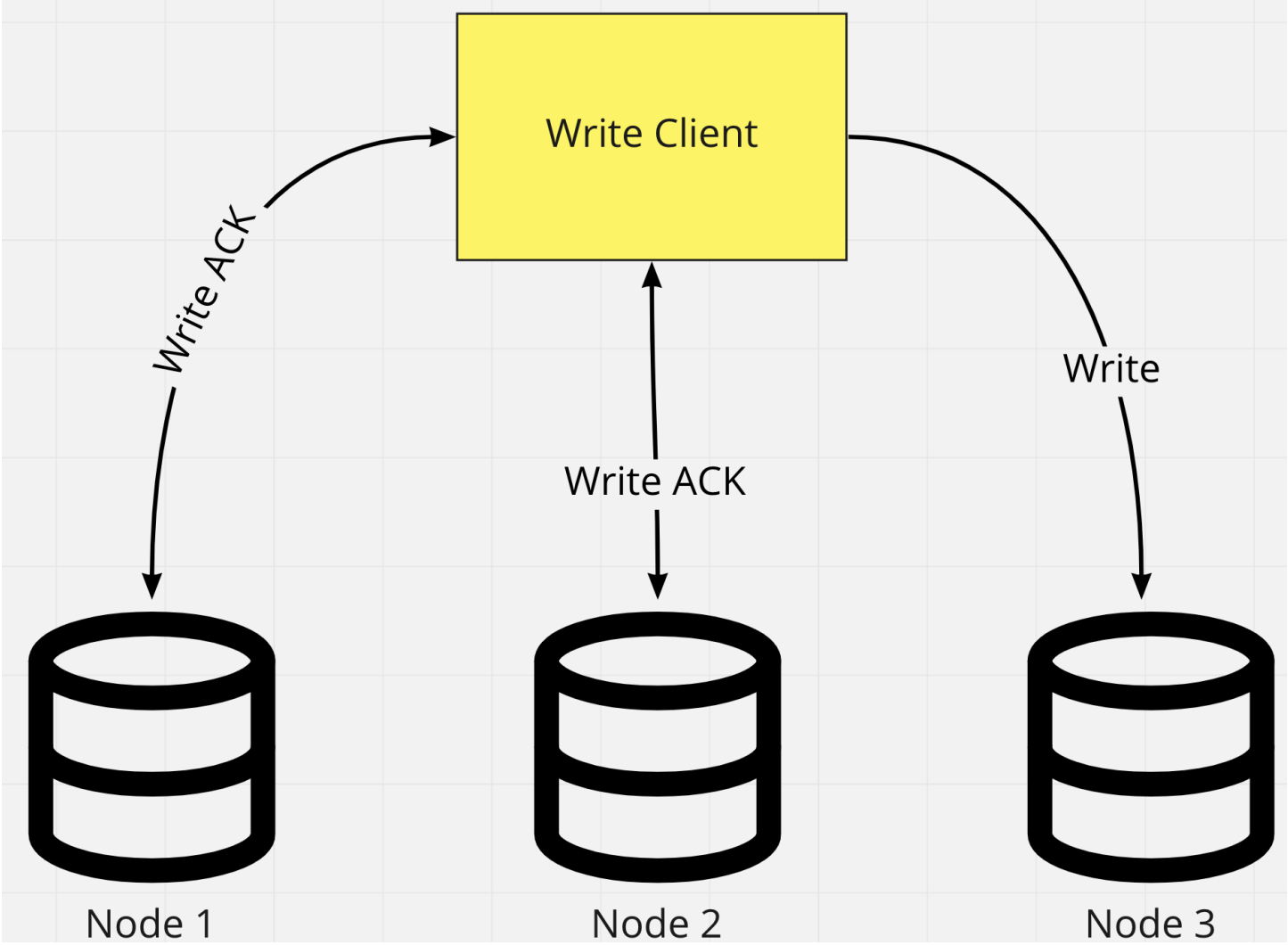
#youtube #database

Leaderless Replication

How is it different?

- No **one node** that serves all write requests
- **Peer nodes**, all of them accept read and writes
- No single point of failure
- Decentralized design
- Better availability
- Mostly non relational databases

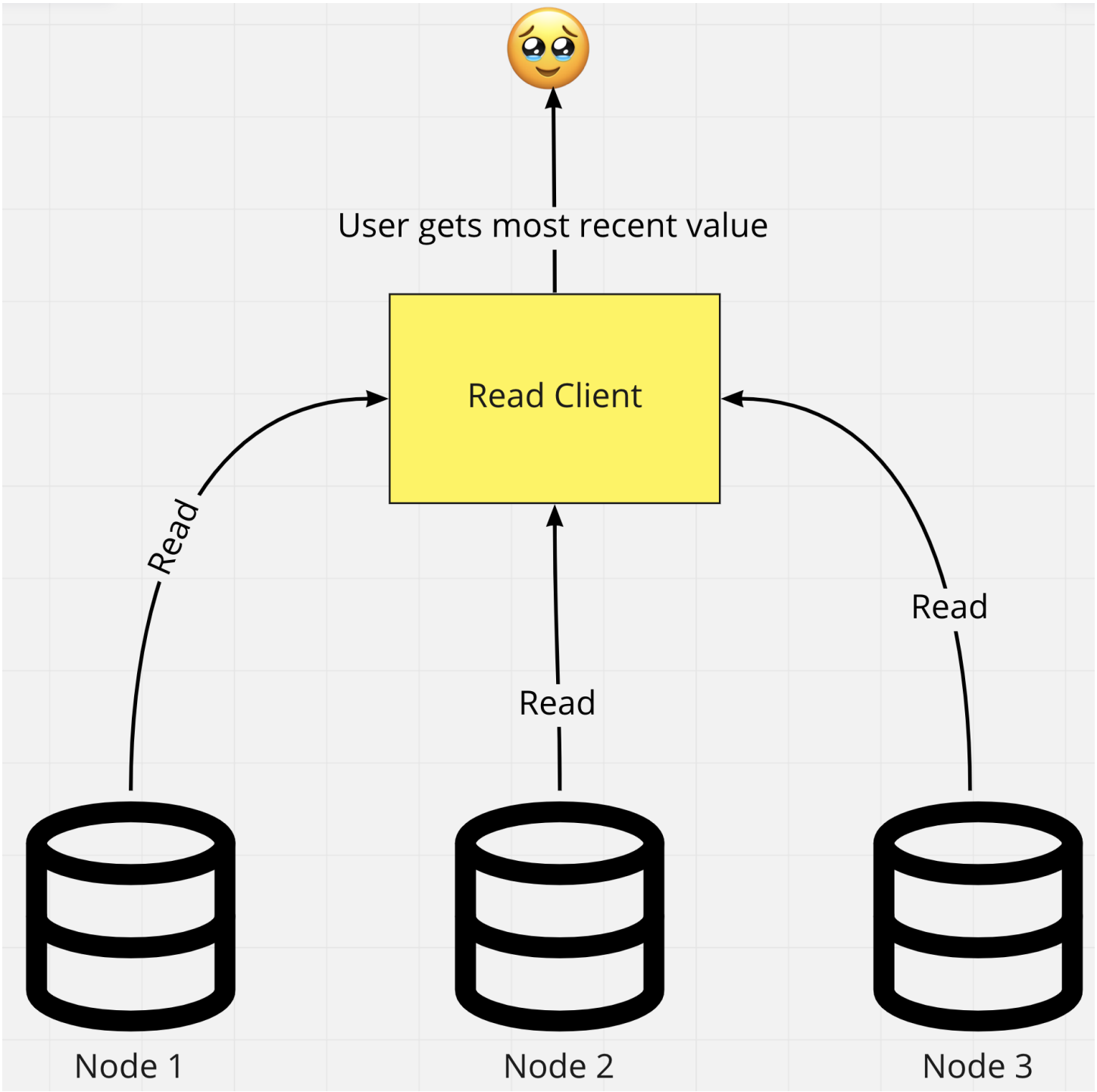
How does writes work?



Steps:

- Client sends new data to more than one node
- Waits for acknowledgements from X nodes
    - X is configurable
- When all ACKs received, the write is successful
- **All nodes don't need to ACK for a write to be successful**

How does reads work?



Steps:

- Client tries to read data from **X nodes**
  - Again, X is configurable
- Client resolves **conflicting data** using some metadata
  - Some kind of ID or Version Number
  - Timestamp
- Client returns **latest** data to user

What happens to nodes with old data?

- Nodes can have **stale data** due to network errors
- Two ways to go around it
  - Read Repair
  - Background process

Read Repair

- **During read request**, client knows which nodes have stale data
- Client figures out what's the latest data
- Client writes this latest data to stale nodes

Background Process

- Background jobs to look for data differences amongst nodes
- Writes latest data to stale nodes **asynchronously**

Biggest Tradeoff

- More available
- Less consistent
- **Configurable consistency, but at the expense of availability**
- Typically use it in write intensive applications, where availability is more important than consistency.

Example Databases

- Cassandra
- DynamoDB