System Design Interview Patterns - Contending U

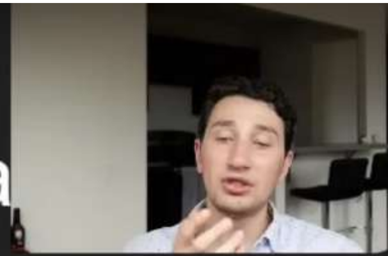
Situation: Many writes to the same key, such that they conflict with one another

Examples: Distributed counter, order inventory on popular products

Solutions:

- Writes to the same database, use locking (naive solution, too slow)
- Multiple database leaders, eventual convergence via automatic merging
  - Very relevant for counting, see something similar to our version vector from before
- Stream processing
  - Each event can go in a log based message broker, and processed in small batches

System Design Interview Patterns - Derived Data



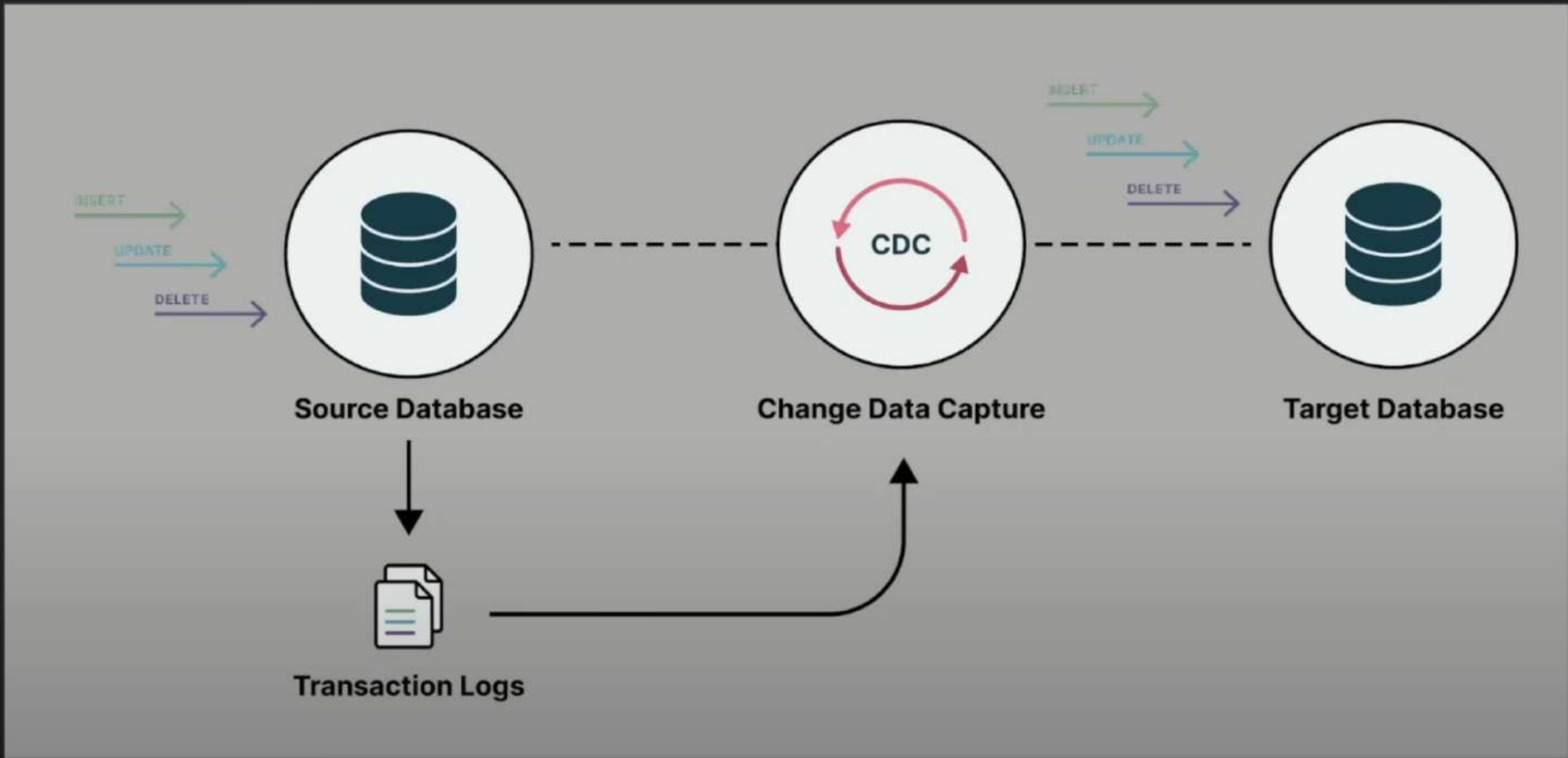
Situation: Need to keep two datasets in sync with one another

Example: Global secondary indexes, data transformations on slow database which accepts writes to faster view that is read optimized

Solutions:

- Two phase commit (naive, slows down writes but may be unavoidable)
- Change data capture
  - When updating one database, sink its changes into a log based message broker, and use a stateful consumer to enrich the data and sink it to another view
  - Avoids an expensive write, but means the derived dataset is eventually consistent with the original

System Design Interviews - Derived Data



System Design Interview Patterns - Fan Out

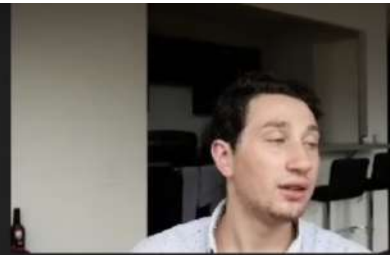
Situation: Deliver data at write time directly to multiple interested parties to avoid expensive read queries or many active connections on receiving devices

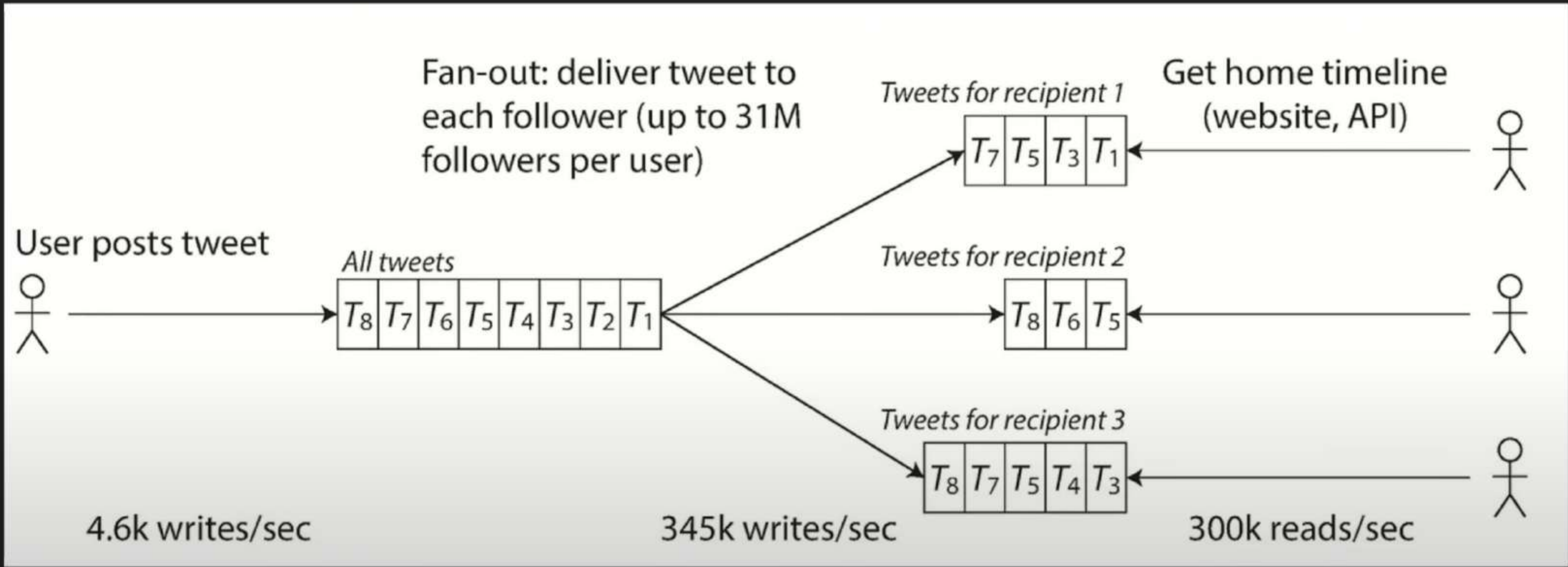
Examples: Push notifications, news feed, mutual friend lists, stock price delivery

Solutions:

- Synchronous delivery to every interested party (naive, request will time out)
- Asynchronous delivery via stream processing
    - Sink message to log based message broker, in consumer logic figure out all parties that are interested, and send to them accordingly

System Design Interview Patterns - Fan Out





System Design Interview Patterns - Proximity Sea

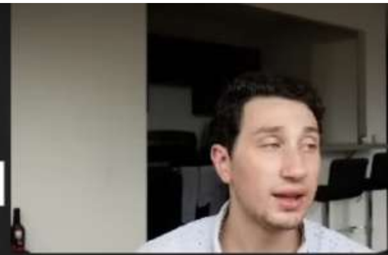
Situation: Find close items in a database to you

Examples: Uber driver search, Doordash, Yelp, AirBnb, Find My Friends

Solutions:

- Build indexes on latitude and longitude (naive, too slow)
- Use a geospatial index
    - Data likely will need to be partitioned, use bounding boxes as partition methodology
    - Certain geographic areas much more popular than others, shards should have even amount of load, not necessarily geographic coverage (e.g. shard for New York City, shard for Alaska)

Systems Design Interview Patterns - Job Schedu



Situation: Run a series of tasks on one worker in a cluster

Examples: Build a job scheduler, Netflix/YouTube video upload encoding

Solutions:

- Round robin jobs into log based message broker partitions (naive)
  - One consumer per partition, slow jobs delay other jobs in that partition
- In memory message broker delivering messages round robin to workers
  - Slow jobs do not prevent jobs behind them in the queue from running

Systems Design Interview Patterns - Aggregation

Situation: Distributed messages that need to be aggregated by some key or time

Examples: Metrics/logs, job scheduler completion messages, data enrichment

Solutions:

- Write everything to a distributed database, run batch job (naive, slow results)
- Stream processing, all messages go into a log based message broker partition based on their aggregation key
    - Stream processing consumer frameworks handle fault tolerance for us

System Design Interview Patterns - Idempotence

Situation: You do not want to see the output of your task more than once

Examples: One time execution in job scheduler, confirmation emails to users

Solutions:

- Two phase commit (naive, slow)
  - Ensures task is marked as completed from one data source at the same time it is performed
- Idempotency keys
  - Store a unique ID for the task and check if we’ve seen it before for all incoming tasks
  - If reaching an external service can send task ID and external service can reject if it has seen before (fencing tokens)

Systems Design Interview Patterns - Durable Dat

Situation: You have data that absolutely cannot be lost once written

Examples: Financial transactions

Solutions:

- Synchronous replication (naive, if any replica fails no writes can be made)
- Distributed consensus
    - Can tolerate node failures and still proceed
    - Fairly slow for reads and writes, so using change data capture to derive read optimized data views can be very beneficial here