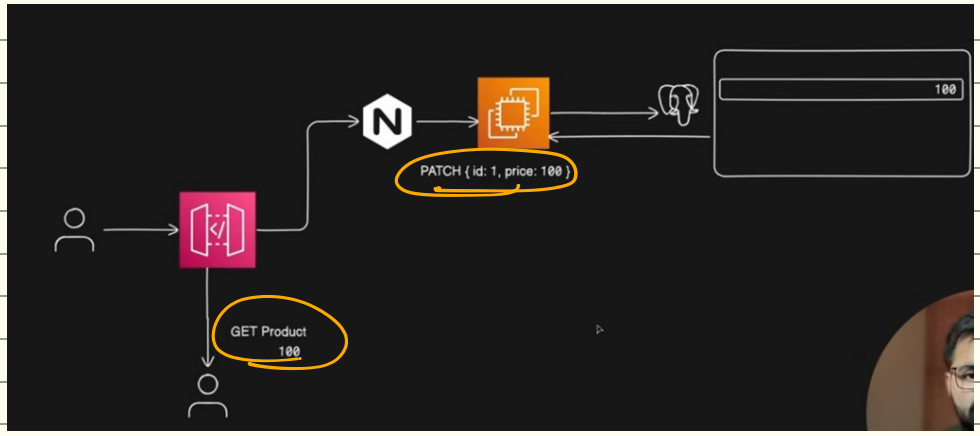
SYSTEM DESIGN

→ Event Sourcing.

→ Events :-

(i). Add Item to Cart
(ii). Checkout Item
(iii). Product Update.

Example :-



- If we update a row in a DB, but the server gets a GET request just a few ms before the PATCH updation command it becomes very difficult for state management and data updation and process.

→ Video Processing pipeline using Event sourcing.

STATES
(i). Processing.
(ii) Failed.
(iii). Processed.

ROUGH PSUEDOCODE :-

(i). USER uploads video
db update status "UPLOADED"

(ii). when worker picks video
db update status "PENDING"

(iii). When worker is done re-upload processed video
db update status "SUCCESS"

∵ If the query for DB fails for updating the state, because

(i). Query was ran wrong.
(ii). DB was busy.

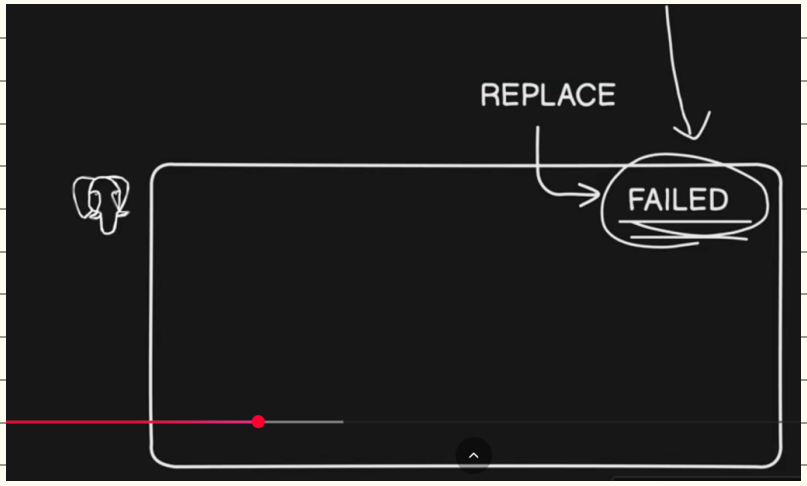
then it becomes very difficult to maintain the state of the server.

Here's where event sourcing comes into the picture.

∴ Event Sourcing states that our source of truth are events.

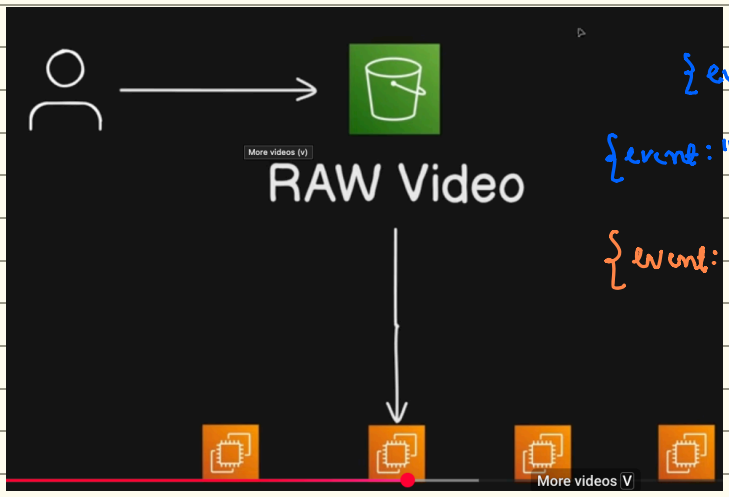
→ In event - sourcing,

- we do not update in the database, in DB we do atomic operations (we directly replace the status).



∴ Always add event log.

Event - log only works on append only logs.



{event: "Video Upload", data: {path: "} + timestamp

{event: "VideoProcessing Init", data: {path: "} + timestamp

{event: "VideProcessing Success", data: {path: "} + timestamp

→ Hydration :- Using events it re-constructs the state.

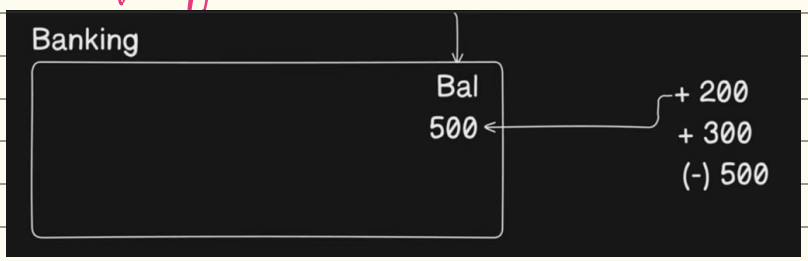
↳ Uploaded

↳ Processing

↳ Success

```
{ event: "VideoUpload", data:{ path: '' }, timestamp }
{ event: "VideoProccessingInit", data:{ path: '' }, timestamp }
{ event: "VideoProccessingSucess", data:{ path: '' }, timestamp }
```

→ In a banking application :-



Updating the DB for maintaining account balance, is atomic process.

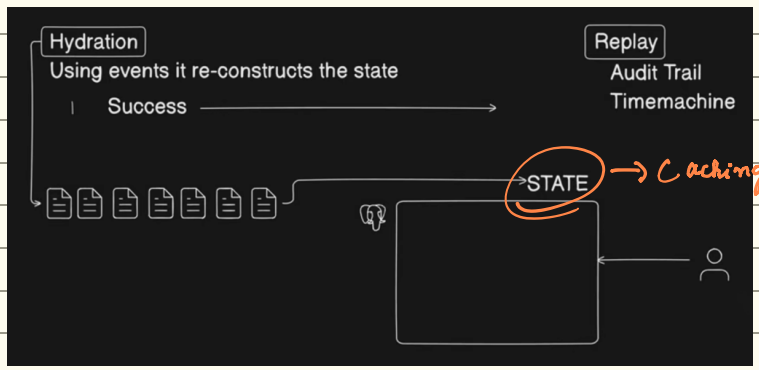
INSTEAD maintain event logs.

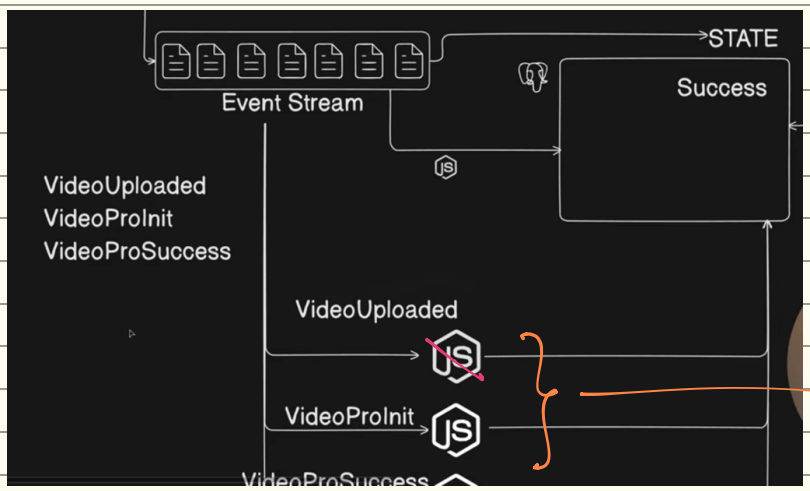
→ Replay :- Audit Trail., reconciliation of events if something goes wrong.

→ Timemachine :- Previous or past logs of different time stamp :-

ex :- 1 month, 1Yr, 5Yr. can be fetched using event logs.

→ Hydration is a very slow and time-consuming process, hence we can cache the state.

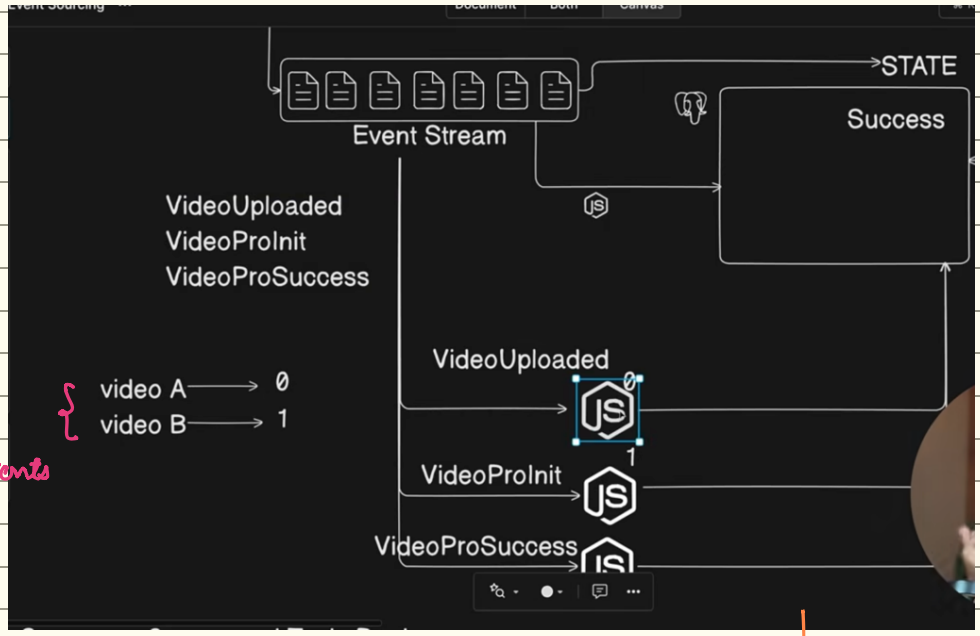




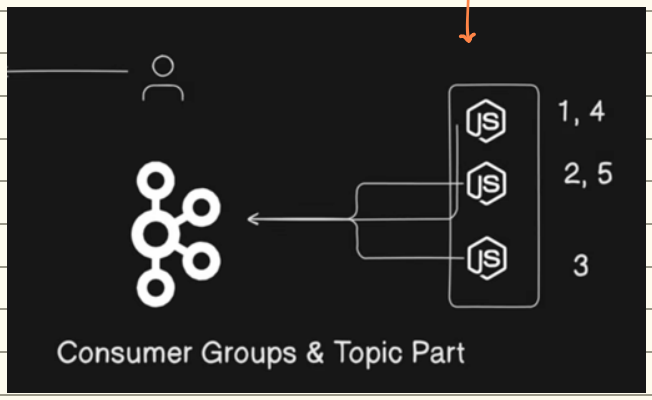
Different workers are assigned to maintain event Logs.

Now, if any worker is slow and consumed and other workers have fast up and updated the events, but now the crashed worker has again updated the event wrongly, so in this case we use

Consumer Groups and Topic partitions.



So now, we partition the events



→ Event Sourcing :-

Event sourcing is a software design pattern where changes to an application's state are stored as a sequence of events, rather than just the current state. Instead of updating a database directly, each state-changing action is recorded as an event in an append-only log. This allows for reconstructing the application's state at any point in time by replaying these events.