Date:
M T W T F S S

Publish/Subscribe Pattern

a.k.a. Pub/Sub model a.k.a. pub-sub

Recap : streaming → client-server long-lived connec° where client listens for data.

eg :- 1) chat app
2) like trading of stocks by stockbrokers - they gotta know latest updated stock prices as they gonna be bettin huge money on it.

Say now we wanna expand this sys into a large scale distri. sys.

Issues → handlin network partiⁿs (eg:- clients lose connecⁿ to servers)
→ servers die

In above cases, what happens to the data (eg:- txt msgs) when an issue is encountered? Is the data lost? Will clients be able to retrieve data upon reconnecⁿ?

Soluⁿ → The moment we think of expandin any sys into large scale distri sys, we gotta start thinkin abt persistant storage.

M-1 : Store data in a typical DB.

Problem with M-1 → Can't be used in all cases.
eg → Client sends async opn to server, opn takes

(C) (S) (dB)

some time in server to process and then resp is sent back to client. We don't use a typical dB for this as we gotta wait in req-resp cycle

M-2 → Storage soln. at server level. i.e. Server ≡ storage
eg:- Streamin

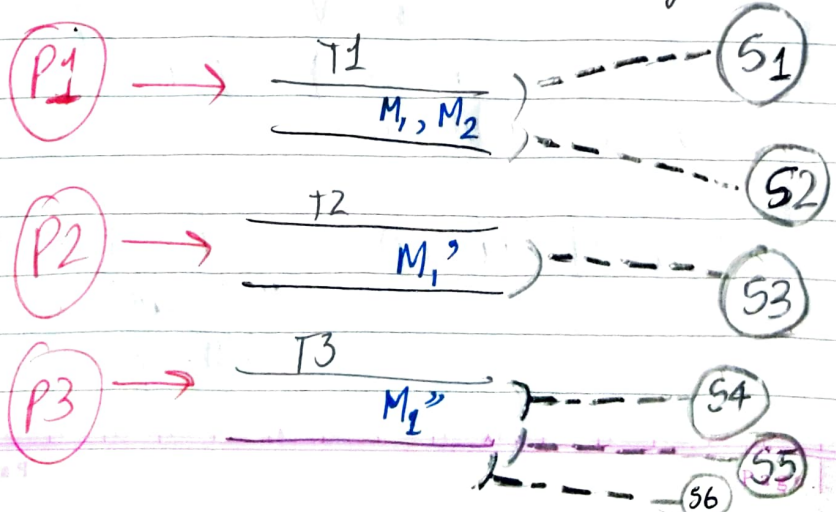
(C) (S≡dB)

Problem with M-2 : We don't wanna keep our business logic (usually in server) and our storage (usually in dB) ~~at the~~ separated

Pub-Sub System

1 A paradigm of 4 entities :

1) Publisher → servers that publish data to topic
2) Subscriber → Clients ~~as~~ listens to data from topics
3) Topic → Channels of specific info. Intermediary b/w publishers and subscribers (who don't directly comm. with each other
4) Msg → Represent → data / opn. of some form i.e. relevant to subscribers eg:- a chat txt, stock trade opn.



(2) Persistent storage in pub-sub pattern

Topics ≡ a dB soln.

∴ All msgs published to topic → effectively stored in a dB soln ≡ persistent

∴ Guarantee :- Msgs in a topic will delivered atleast once to subscribers

Overview logic - (1) Each msg keeps track of subscribers thru some sort of a UID.
(2) Upon receipt of msgs., subscribers send a 'ack' (acknowledgement) to the topic

(3) Idempotent opn.

Problem → (1) Msg. recd by subscriber but suddenly connec" b/w subscriber & topic fails
(2) Subscriber didn't send ack to topic
(3) Upon reconnec", topic thinks sub. doesn't hv msg so its gonna resend it.

Idempotent opn. → Opn. giving same ultimate outcome regardless of how many times it's performed

eg:- Click on 'Register' btn. to register for a webinar

eg of non-idempotent opn. → Like a facebook post (2nd time you press 'Like' btn, post is unliked)

Pub-sub sys has ~~drawbacks~~ drawbacks if the opn is non-idempotent cuz on reconnect of client the opn kinda does a diff thing than it did before.

(4) Ordering of msgs.

Order in which msgs published to topic from publisher = Order in which msgs pushed to sub. frm. topic

Queue (FIFO → 1st in 1st out)

use case: chat msgs appear chronologically, transac" opns. performed sequentially

(5) Replay msgs

Rewinding to a prev. msg or a prev. snapshot in time of the topic

eg:- Git revert

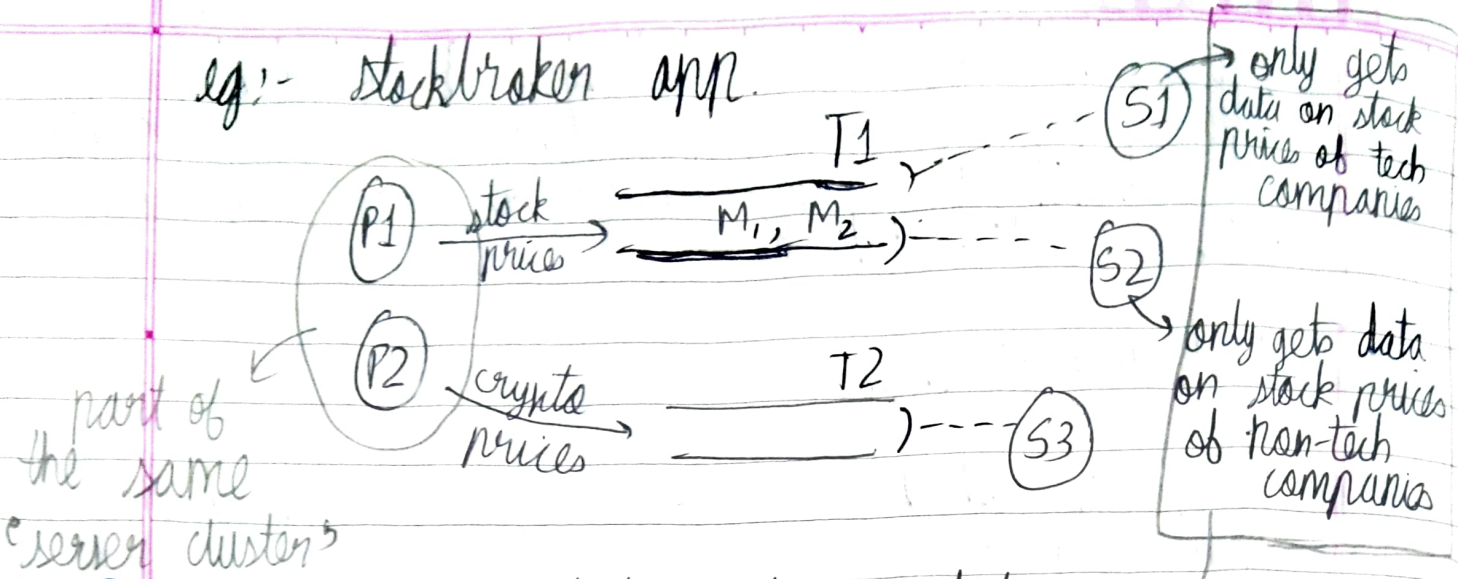
(6) Pub-sub Topics → Channels :: they channel specific data/opns. frm pub to sub.

All topics stored in ~~db~~ a common single dB
Each topic represents a diff type of data

This dB → bi-dir"al (pub. on 1 side, sub. on other)
→ clear separat" of data/opns. in topics

Date:
M T W T F S S

eg:- stockbroker app.



(7) Content based filtering at sub. lvl

Subs who have subscribed to the same topic but get pretty specific data amongst the data stored in the topic.

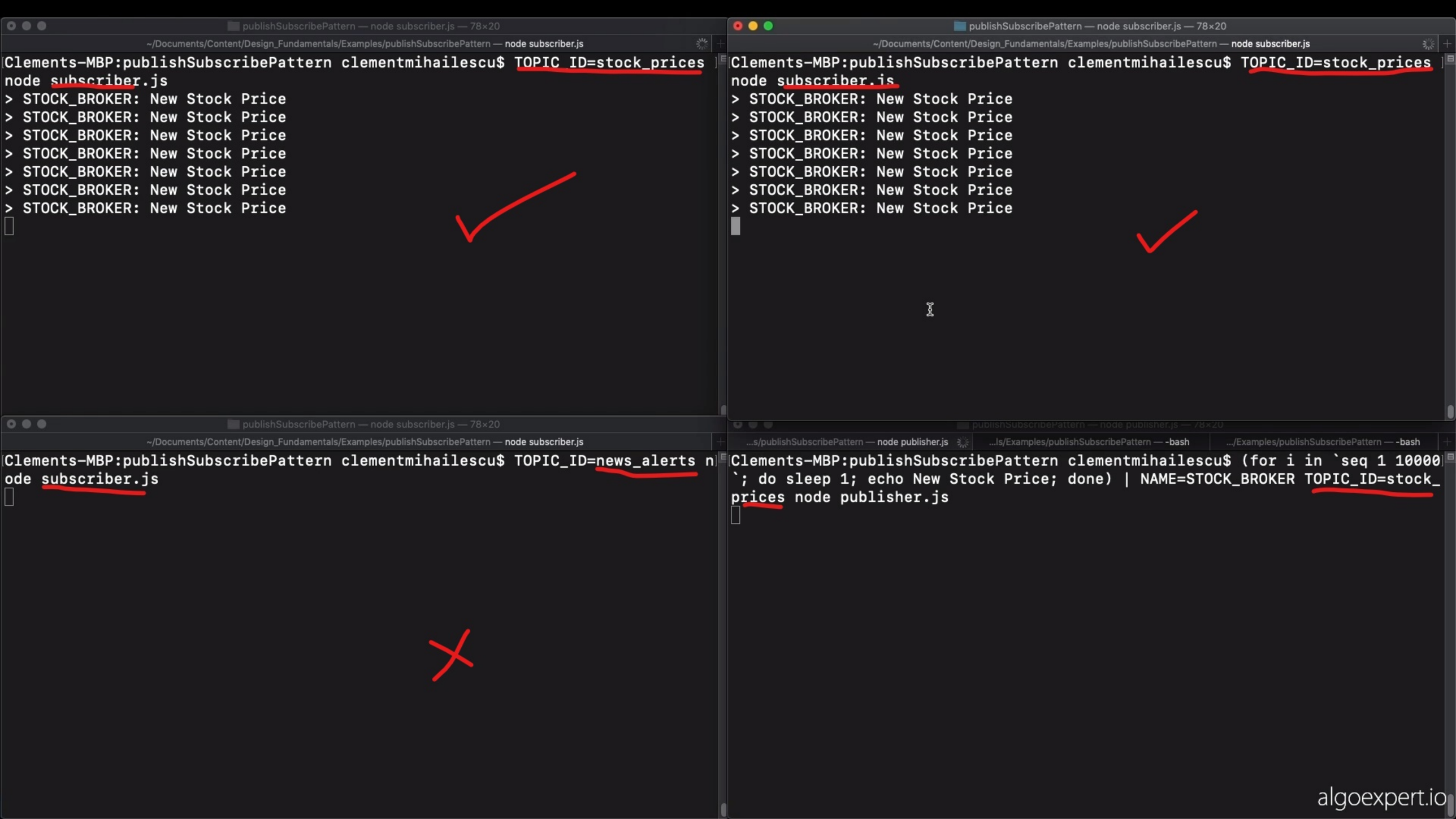
(8) Examples of pub-sub tools/solns

1) Google Cloud Pub-Sub - Everything auto-scales and you don't need to worry cuz this soln. will automatically take care of it by sharding topics

2) Google Cloud Platform, AWS - Provide end-to-end encryp^{n} of msgs s.t. msgs are encrypted over the network whilst inside topics and only subs know how to read em.
eg:- whatsapp msg encryp^{n}

3) Apache Kafka.

LINK TO CODE EXAMPLE OF PUB/SUB PATTERN


publishSubscribePattern — node subscriber.js — 78×20
~/Documents/Content/Design_Fundamentals/Examples/publishSubscribePattern — node subscriber.js
Clements-MBP:publishSubscribePattern clementmihailescu$ TOPIC_ID=stock_prices node subscriber.js
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
publishSubscribePattern — node subscriber.js — 78×20
~/Documents/Content/Design_Fundamentals/Examples/publishSubscribePattern — node subscriber.js
Clements-MBP:publishSubscribePattern clementmihailescu$ TOPIC_ID=stock_prices node subscriber.js
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
publishSubscribePattern — node subscriber.js — 78×20
~/Documents/Content/Design_Fundamentals/Examples/publishSubscribePattern — node subscriber.js
Clements-MBP:publishSubscribePattern clementmihailescu$ TOPIC_ID=news_alerts node subscriber.js
publishSubscribePattern — node publisher.js — 78×20
...s/publishSubscribePattern — node publisher.js
...ls/Examples/publishSubscribePattern — -bash
.../Examples/publishSubscribePattern — -bash
Clements-MBP:publishSubscribePattern clementmihailescu$ (for i in `seq 1 10000`; do sleep 1; echo New Stock Price; done) | NAME=STOCK_BROKER TOPIC_ID=stock_prices node publisher.js
algoexpert.io

publishSubscribePattern — node subscriber.js — 78×20

~/Documents/Content/Design_Fundamentals/Examples/publishSubscribePattern — node subscriber.js

```
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
```

publishSubscribePattern — node subscriber.js — 78×20

~/Documents/Content/Design_Fundamentals/Examples/publishSubscribePattern — node subscriber.js

```
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
^C
Clements-MBP:publishSubscribePattern clementmihailescu$ TOPIC_ID=stock_prices
node subscriber.js
> STOCK_BROKER: New Stock Price
> STOCK_BROKER: New Stock Price
```

publishSubscribePattern — node subscriber.js — 78×20

~/Documents/Content/Design_Fundamentals/Examples/publishSubscribePattern — node subscriber.js

```
Clements-MBP:publishSubscribePattern clementmihailescu$ TOPIC_ID=news_alerts n
ode subscriber.js
> NEWS_STATION: Breaking News
> NEWS_STATION: Breaking News
> NEWS_STATION: Breaking News
> NEWS_STATION: Breaking News
> NEWS_STATION: Breaking News
> NEWS_STATION: Breaking News
> NEWS_STATION: Breaking News
> NEWS_STATION: Breaking News
> NEWS_STATION: Breaking News
> NEWS_STATION: Breaking News
```

publishSubscribePattern — node publisher.js — 78×20

...les/publishSubscribePattern — node publisher.js | ...s/publishSubscribePattern — node publisher.js | .../Examples/publishSubscribePattern — -bash

```
Clements-MBP:publishSubscribePattern clementmihailescu$ (for i in `seq 1 10000
`; do sleep 1; echo Breaking News; done) | NAME=NEWS_STATION TOPIC_ID=news_ale
rts node publisher.js
```