DataStax

Developers

Build a **Multiplayer Real-time Game** with Astra Streaming

01

Today's workshop
Housekeeping

02

Messaging systems
"why not just a DB?"

03

Storage
Persisting game status

04

Pulsar / Astra streaming
A modern message system

05

Game architecture
Highlights of this game

06

What's next?
Quiz & Homework

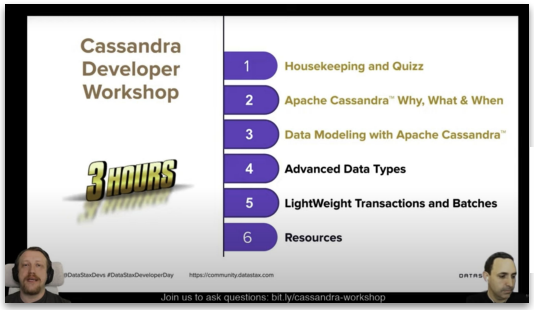
Agenda

Housekeeping

Get the most out of this workshop !

Live and interactive

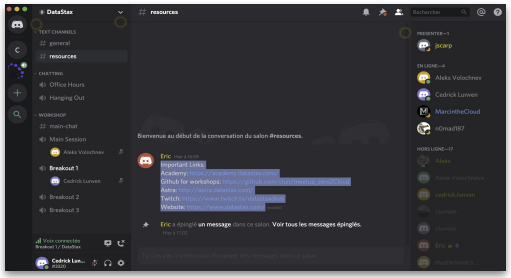
Livestream: youtube.com/DataStaxDevs



YouTube

Twitch

Questions: https://dtsx.io/discord

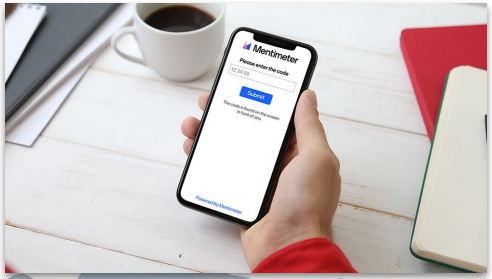


Discord

YouTube

Available on the iPhone App Store

Games menti.com



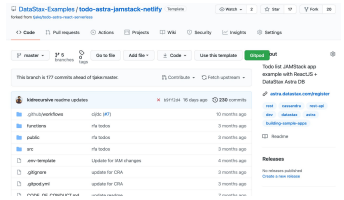
Mentimeter

GET IT ON Google play

Attend the live sessions

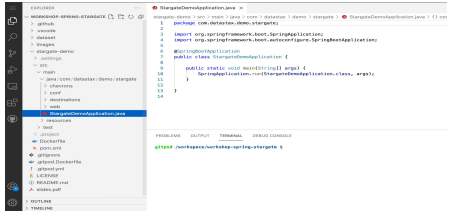
Nothing to install !

Source code + exercises + slides



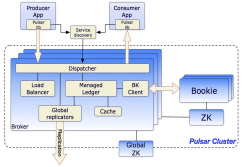
GitHub

IDE



Gitpod

Messaging system (based on Pulsar)



DataStax Astra STREAMING

Database + GraphQL + PlayGround

DataStax Astra DB

Do the hands-on

Do the lab and get your badge!

Let's start !

01

Today's workshop
Housekeeping

02

Messaging systems
"why not just a DB?"

03

Storage
Persisting game status

04

Pulsar / Astra streaming
A modern message system

05

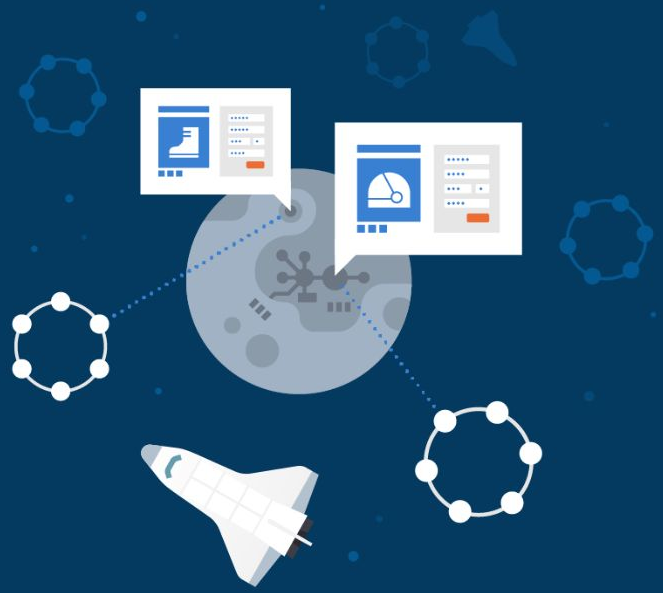
Game architecture
Highlights of this game

06

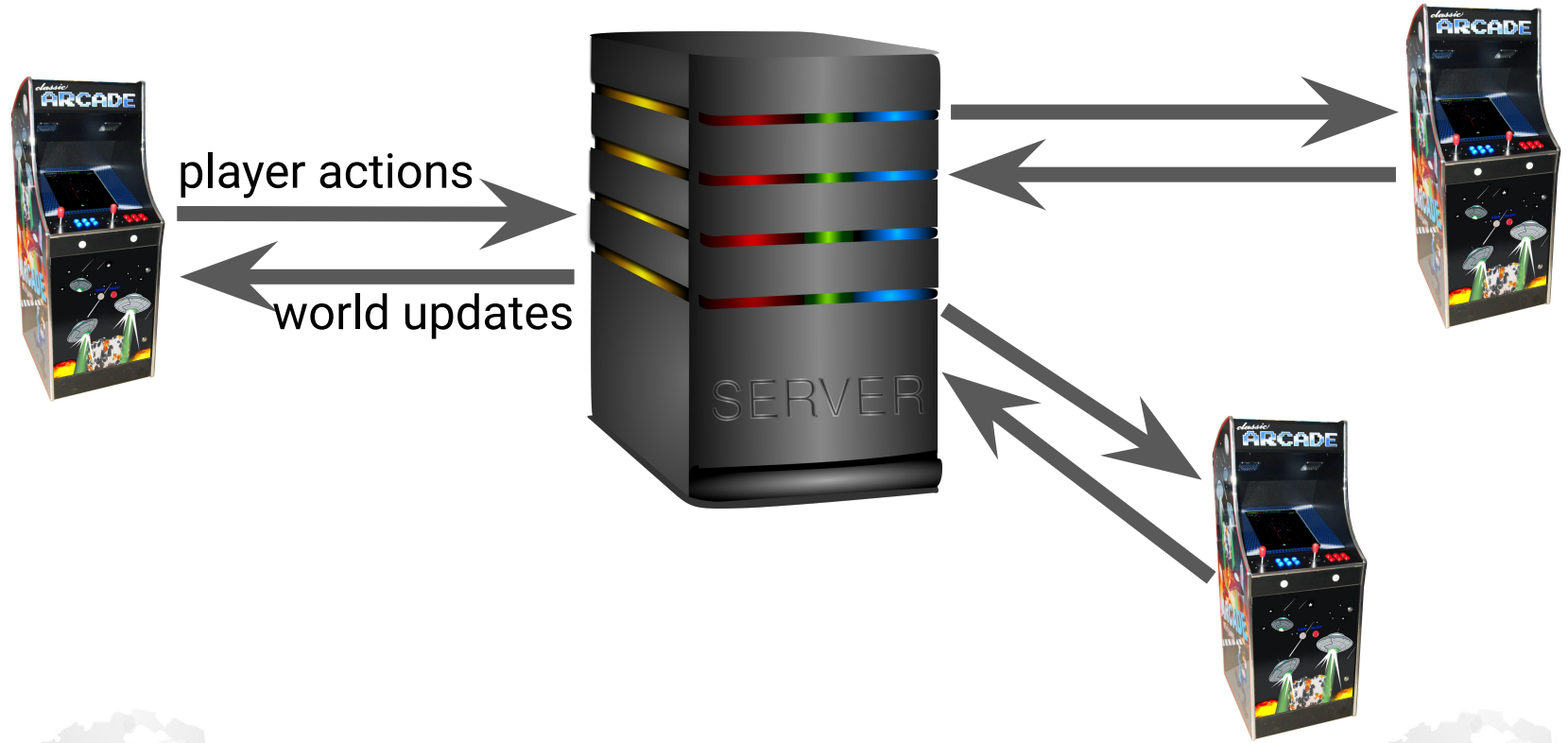
What's next?
Quiz & Homework

Agenda

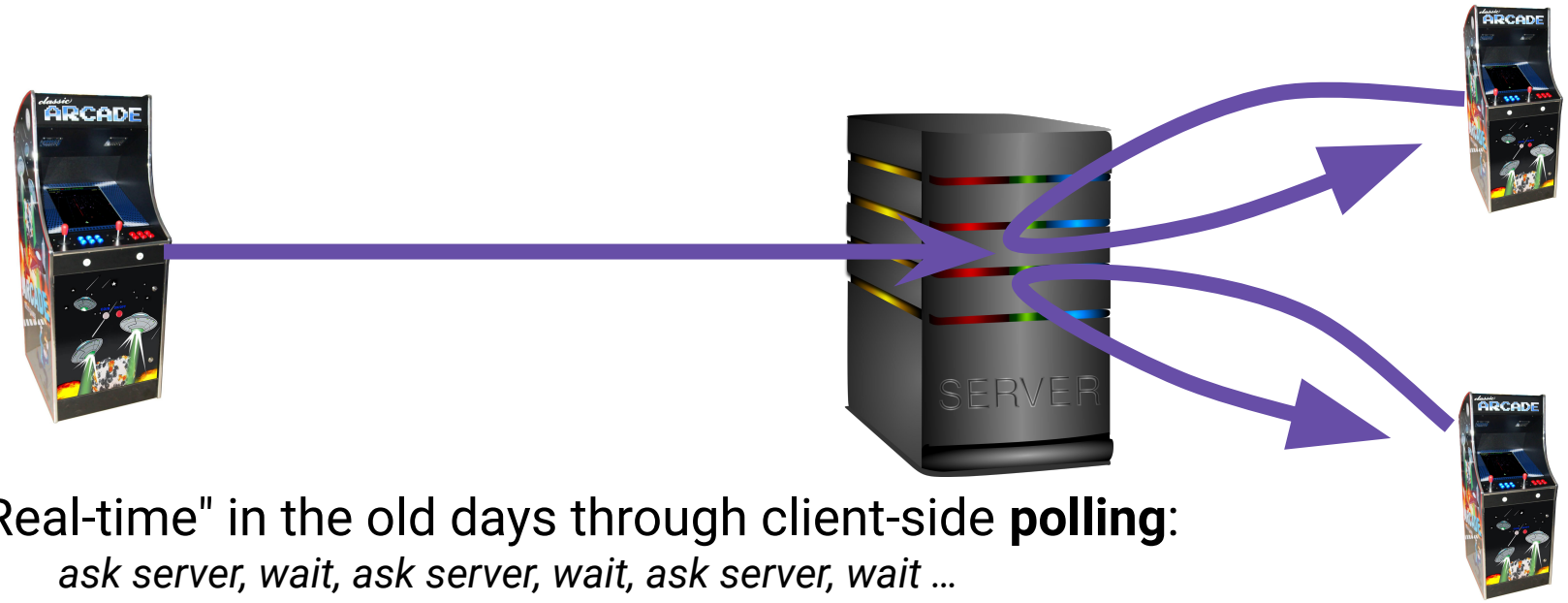
Messaging systems



What engine will power the game?



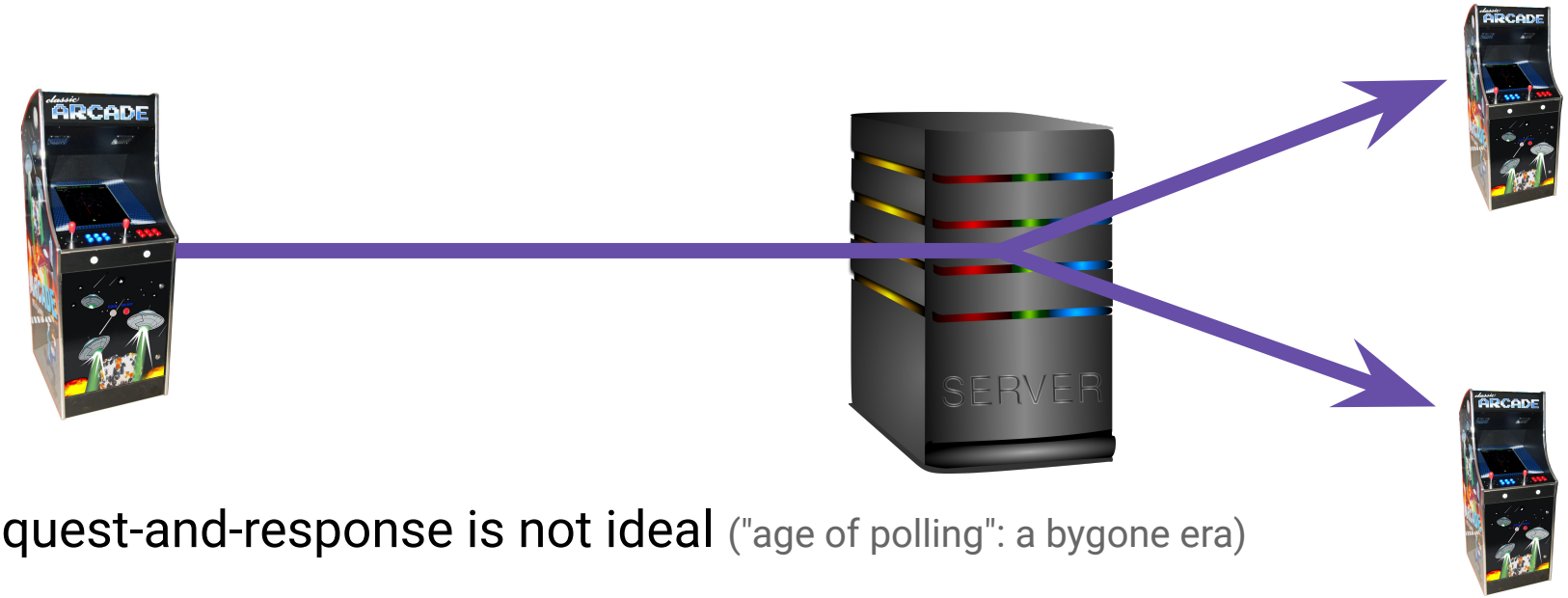
So, you want to build a multiplayer real-time game ... ?



"Real-time" in the old days through client-side **polling**:

ask server, wait, ask server, wait, ask server, wait …

- **x** A flood of request-and-responses (overhead!)
- **x** Not really real-time (latencies)
- **x** Waste of server resources
- **x** Sometimes a mess to handle in the application



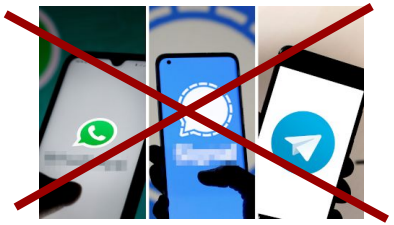
Request-and-response is not ideal ("age of polling": a bygone era)

Rather: server pushing data to clients at once.

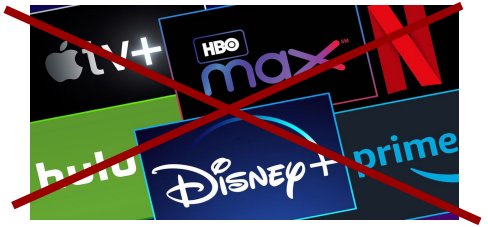
⇒ We need a messaging platform

⇒ And server-initiated communication

- "messaging systems": we DO NOT mean this



- "streaming": we DO NOT mean this



- We rather mean: **backend services specialized to convey data between other backend systems. Data that is made progressively available in small pieces as time goes by.**

- **Born out of the need for:**
  - non-blocking communication
  - "push, not pull"
  - decoupling
  - faster response to new data

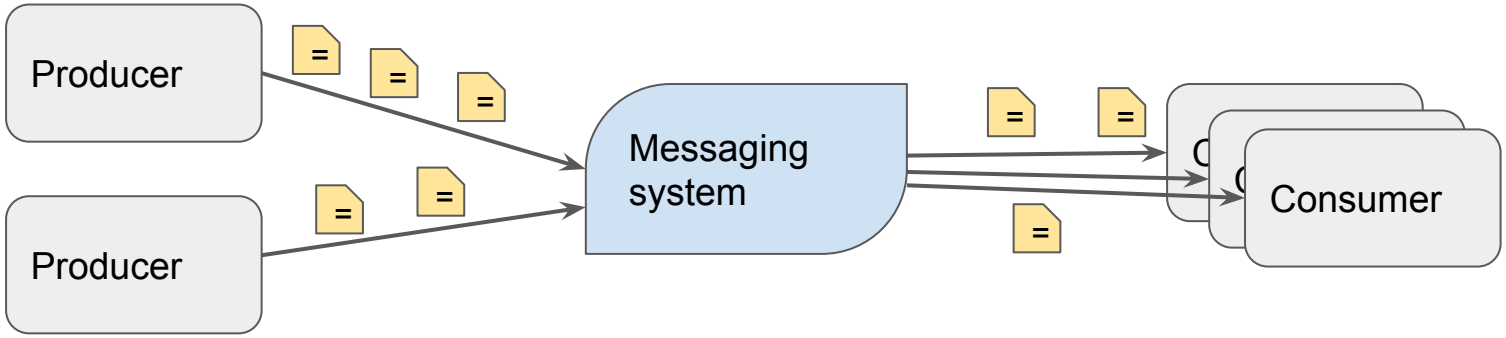
- **Good match with modern microservice architectures**

- **Receive and deliver "messages"**
  - decouple sending and receiving (also: cross-language)
  - Act as a "buffer" for communication
  - various messaging patterns (queue, pub/sub)
  - (optional) Message schemas: JSON, Avro, …

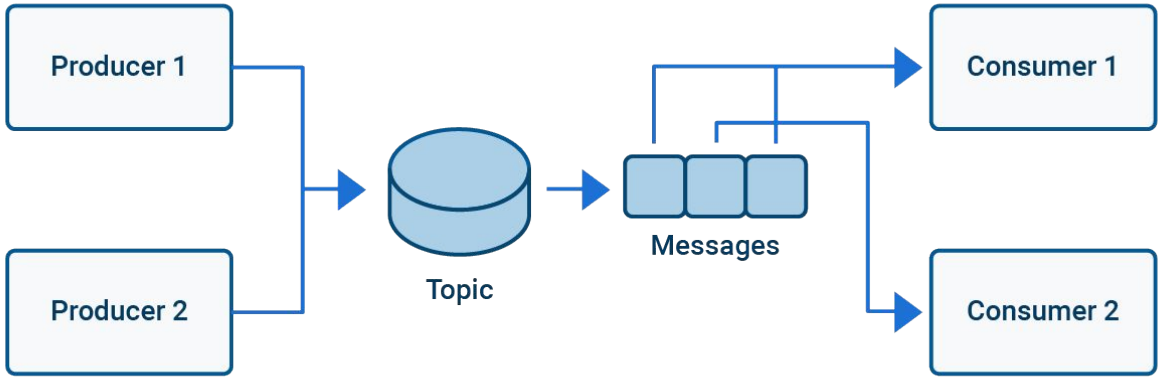
- **We live in a "golden age for messaging services"**
  RabbitMQ 2007, Apache Kafka 2009, Apache Pulsar 2016

- **Use cases for messaging:**
  - the dataset is never complete
  - continuous stream of events/data-points
  - shared queues (items to process)
  - …

- **In practice:** IoT, social media, car automation, finance, online gaming, fraud detection, clickstream ...



- **Asynchronous ⇒ messages retained**
- **Delivery semantics ⇒ receipt acknowledgement**
  (the hard part is correct delivery)

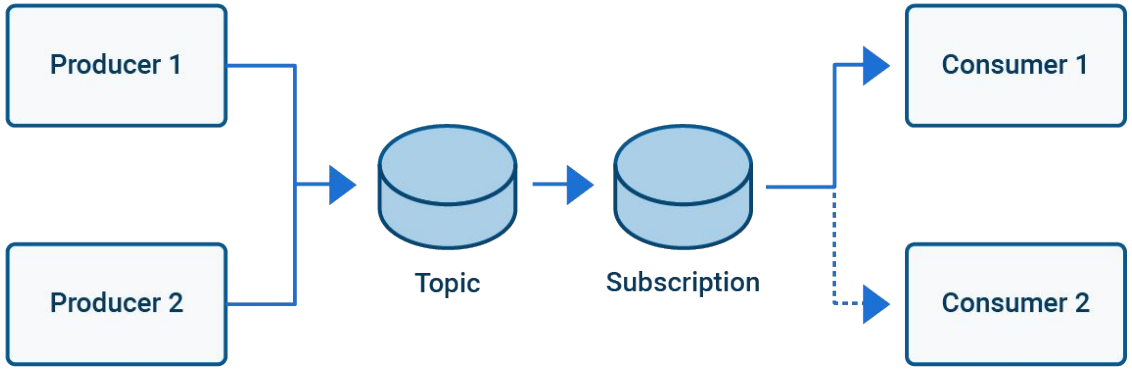


QUEUING

- Exactly-once semantics
- Backlog of messages
- Distributed processing
- Out-of-order possible
- Message failover
- Easy to scale

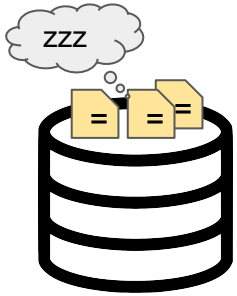
Consumption pattern 1: **Queuing**

PUB-SUB



- Producers publish to Topic
- Consumers subscribe to Topic
- Multiple consumers
- Easy to extend

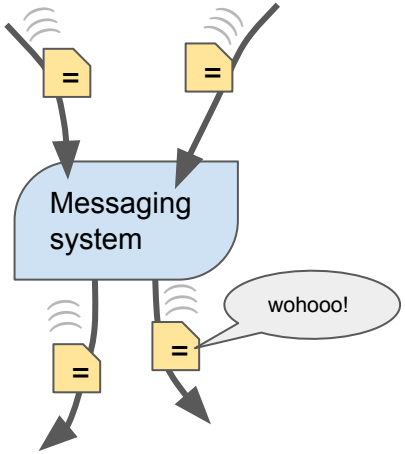
"**Each** player shall receive **all updates** to the game world"



Databases
data at rest
(in "tables/collections")

VS

Messaging
data in transit
(through "topics")



But: blurred boundaries

Most messaging systems offer some *persistence*

Some databases equipped with *CDC* *("change data capture")*

Mixing the two: a very common pattern anyway

Messaging != Databases

01

Today's workshop
Housekeeping

02

Messaging systems
"why not just a DB?"

03

Storage
Persisting game status

04

Pulsar / Astra streaming
A modern message system

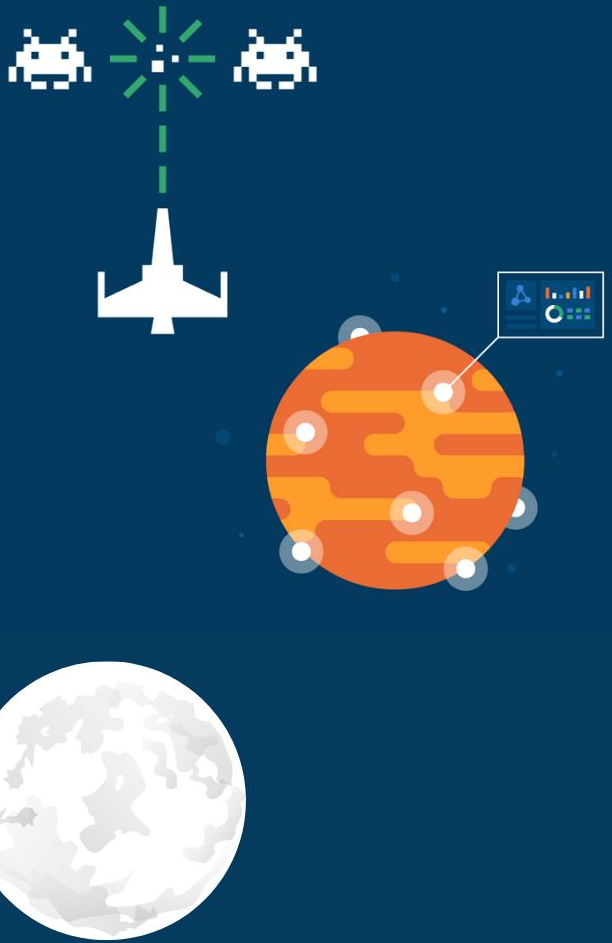
05

Game architecture
Highlights of this game

06

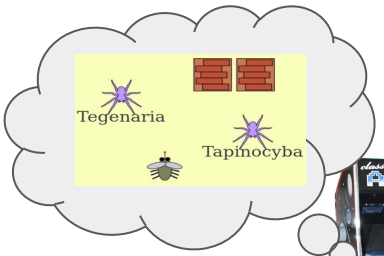
What's next?
Quiz & Homework

Agenda

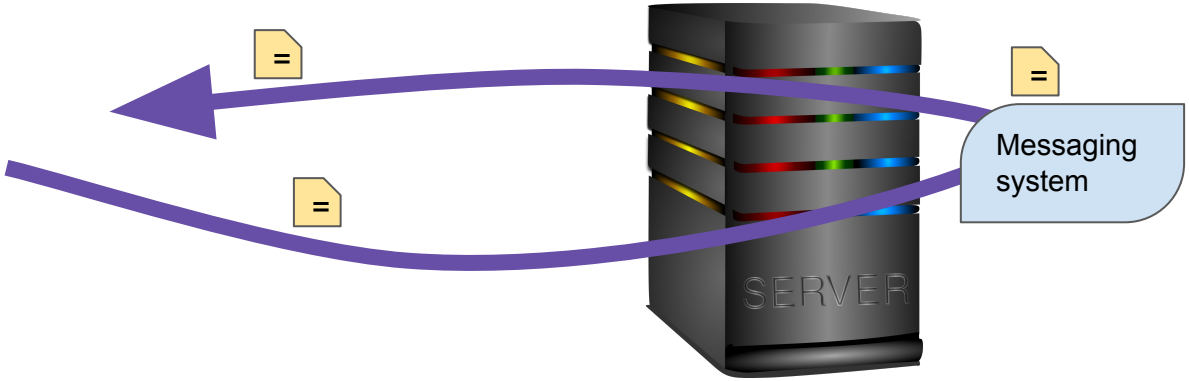


Storage

Making the game richer with server-side persistent storage



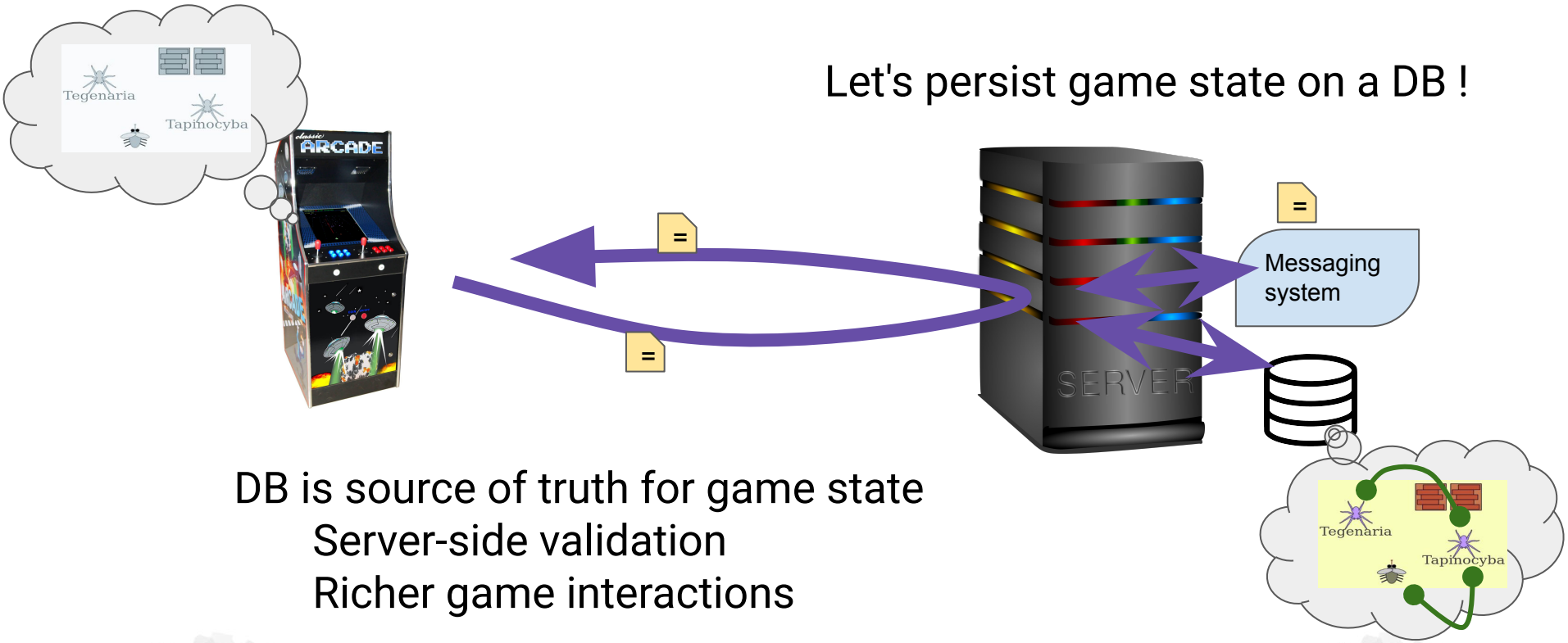
A server that just routes messages



Knowledge of game world on client-side only
(cf. "event-sourcing" architectures…)
Limited player-player and player-world interactions

A simple "game" (no storage)

Let's persist game state on a DB !



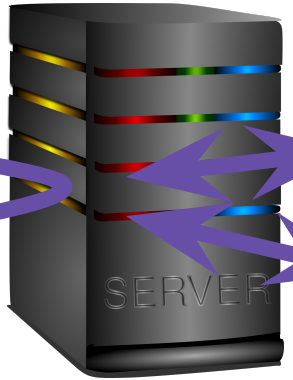
DB is source of truth for game state

- Server-side validation
- Richer game interactions

Server-side persistent storage !

=

=



Messaging system

Messages look like:

- "player A now in (x, y)"
- "player B enters game"
- "there's food at (x, y)"
- ...

DB looks like:

```
token@cqlsh:drapetisca> SELECT kind, name, x, y
FROM objects_by_game_id WHERE game_id = f910b8a4
-fdea-49da-a377-7c1f01e5e59c LIMIT 10;

 kind   | name          | x  | y
--------+---------------+----+----
 player | Hylyphantes | 18 | 10
   food |    food_0001 | 15 |  3
   food |    food_0002 | 27 |  1
   food |    food_0000 | 17 |  5
  brick |  brick_0001 | 12 |  7
  brick |  brick_0019 | 24 |  0
  brick |  brick_0007 | 35 | 20
  brick |  brick_0008 | 31 |  7
  brick |  brick_0014 | 19 | 20
  brick |  brick_0017 | 28 | 13

(10 rows)
token@cqlsh:drapetisca>
```

Game as messages VS game on DB

"fundamental theorem of event sourcing"

$$\text{State}(t1) = \int_{t1} (\text{messages})\, dt$$

State looks like:

```
token@cqlsh:drapetisca> SELECT kind, name, x, y
FROM objects_by_game_id WHERE game_id = f910b8a4
-fdea-49da-a377-7c1f01e5e59c LIMIT 10;

 kind   | name         | x  | y
--------+--------------+----+----
 player | Hylyphantes | 18 | 10
   food |   food_0001 | 15 |  3
   food |   food_0002 | 27 |  1
   food |   food_0000 | 17 |  5
  brick |  brick_0001 | 12 |  7
  brick |  brick_0019 | 24 |  0
  brick |  brick_0007 | 35 | 20
  brick |  brick_0008 | 31 |  7
  brick |  brick_0014 | 19 | 20
  brick |  brick_0017 | 28 | 13

(10 rows)
token@cqlsh:drapetisca>
```

Messages look like:

- `"player A now in (x, y)"`
- `"player B enters game"`
- `"there's food at (x, y)"`
- `...`

A side note

01

Today's workshop
Housekeeping

02

Messaging systems
"why not just a DB?"

03

Storage
Persisting game status

04

Pulsar / Astra streaming
A modern message system

05

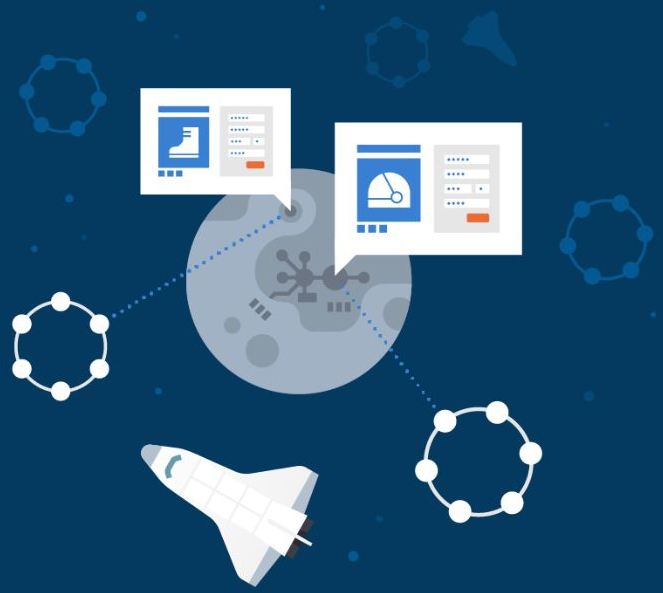
Game architecture
Highlights of this game

06

What's next?
Quiz & Homework

Agenda

Pulsar/Astra Streaming

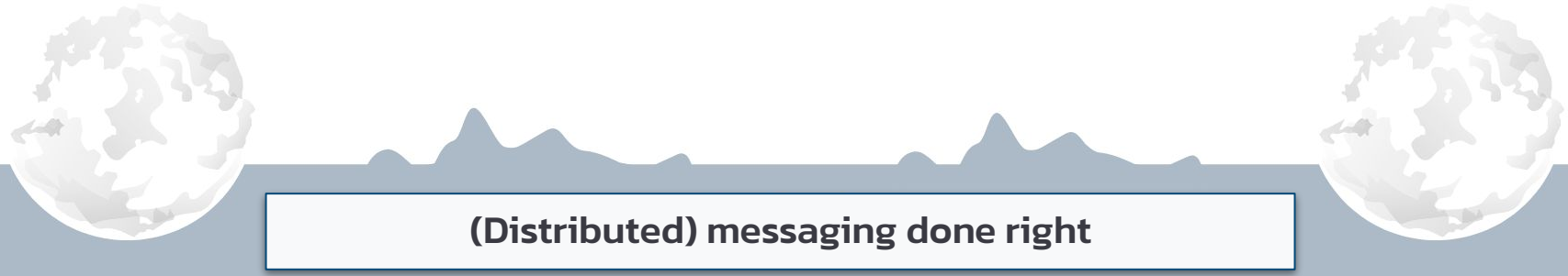


A modern messaging system

… in the cloud !

High-performance, correct delivery, failover, replication, scaling…
Technically it's a tough challenge
⇒ Especially for a distributed system !

Consider also operations/management
… that's why today we rely on a managed solution!



Astra STREAMING

- **Yahoo ⇒ Apache Foundation**
- Supports pub/sub **and** queueing
- Split compute from storage
- Modern features:
  - Multi-tenancy
  - Horizontal scalability
  - High performance / Low latency
  - Geo-replication

  + schemas, functions, Pulsar SQL, deduplication, IO (sinks/sources) ...

- **Managed Pulsar in the cloud currently FREE open beta**
- Create ready-to-use streaming topics
- Tight integration with Astra DB available *(as between Pulsar and Cassandra)*

Apache Pulsar ⇒ Astra Streaming

Fully-qualified topic name:
`persistent://gameserver/default/worldupdates`

PULSAR

Instance (one or more clusters)

Tenant: 1

Tenant: 2

Namespace: `default`

topic

topic

topic:
`worldupdates`

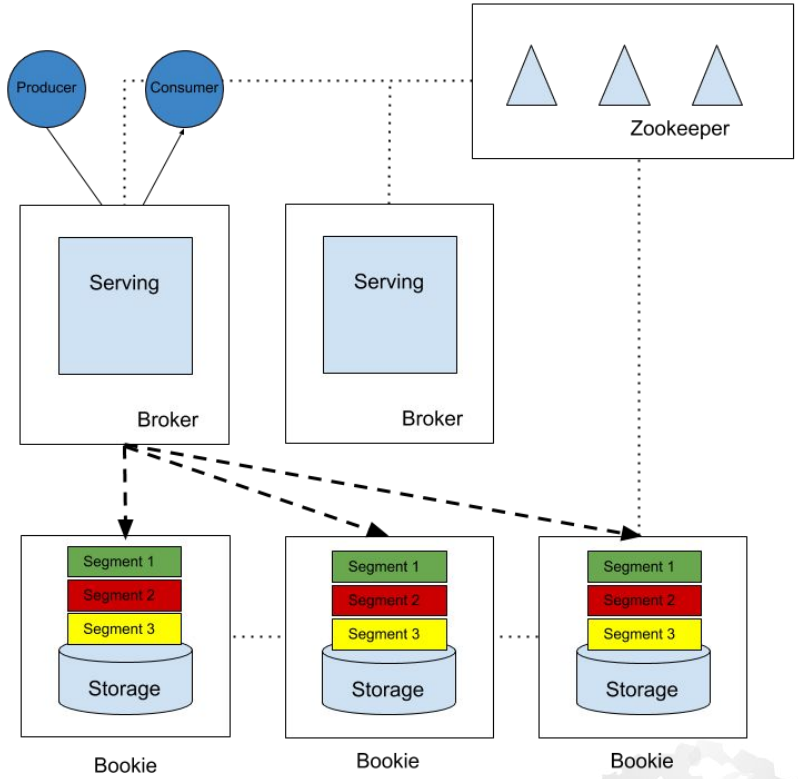
Namespace:
`iot`

Namespace:
`finance`

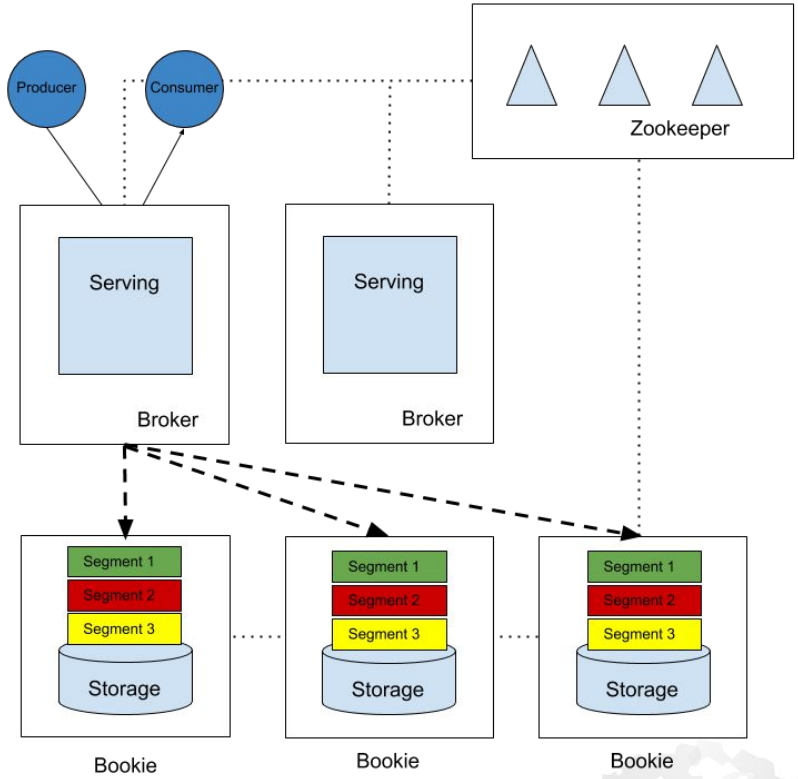
Tenant: `gameserver`
(YOU!)

Pulsar messaging, logical structure

- Distributed, tiered architecture
- Separates compute from storage
- Zookeeper holds metadata for the cluster
- Stateless Broker handles
  producers and consumers
- Storage is handled by Apache BookKeeper



- BookKeeper (“bookies”) distributed, append-only log
- Data is broken into ledgers and segments written to multiple bookies
- Producer acknowledged when quorum of bookies acknowledge
- No single bookie holds entire log



Pulsar architecture / 2

HANDS-ON #1: Setup

- Create Astra Account
- Create Streaming instance
  - ... and get the secrets
- Create DB instance
  - ... and get the secrets

01

Today's workshop
Housekeeping

02

Messaging systems
"why not just a DB?"

03

Storage
Persisting game status

04

Pulsar / Astra streaming
A modern message system

05

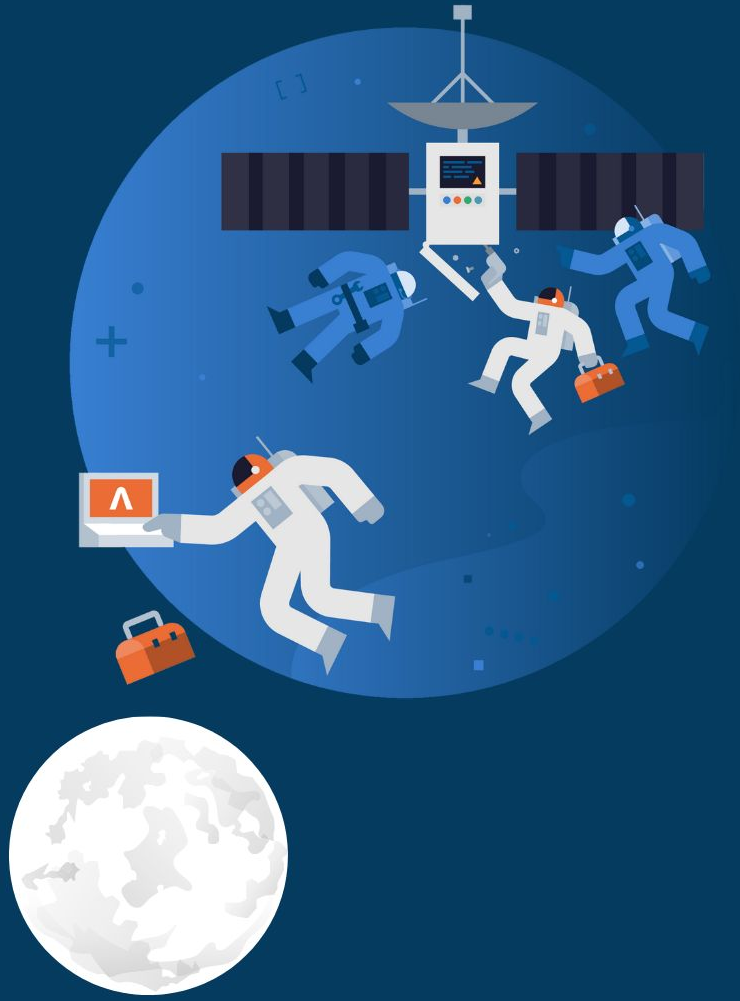
Game architecture
Highlights of this game

06

What's next?
Quiz & Homework

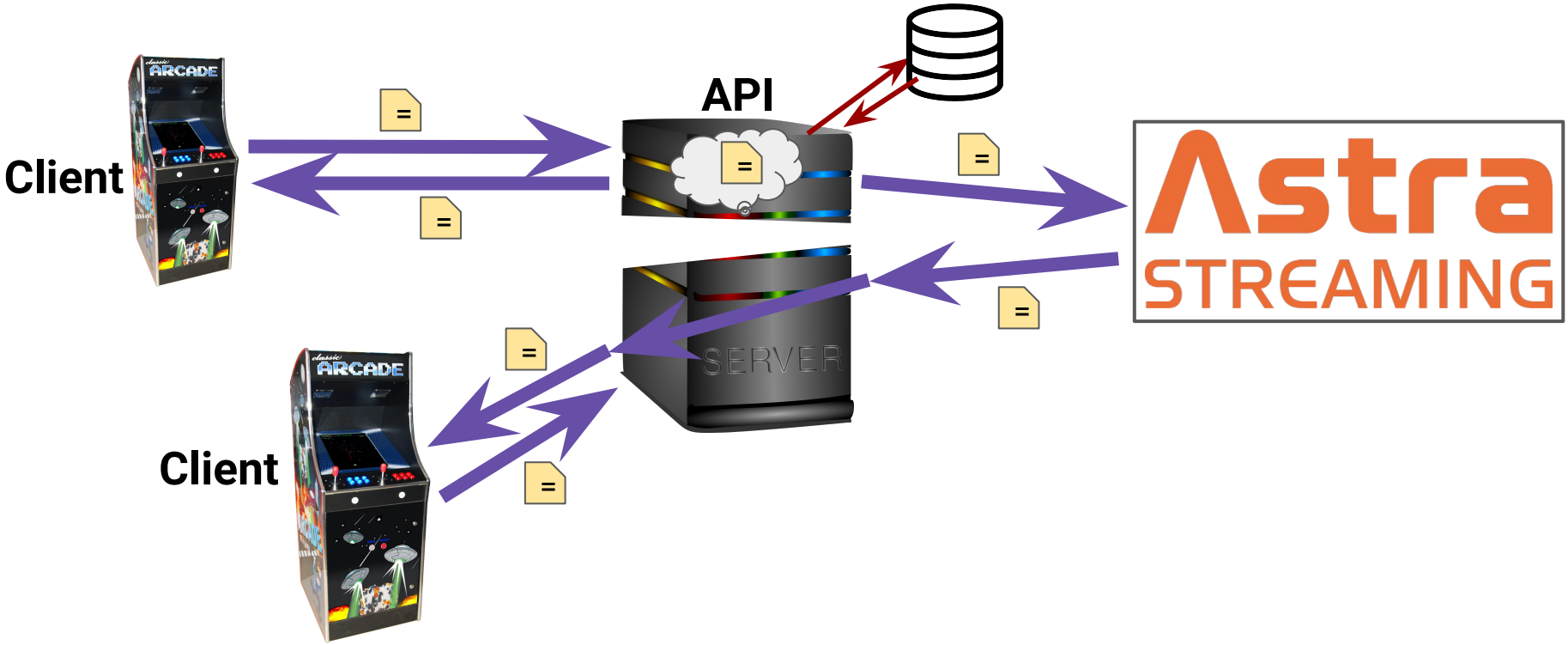
Agenda

Game architecture

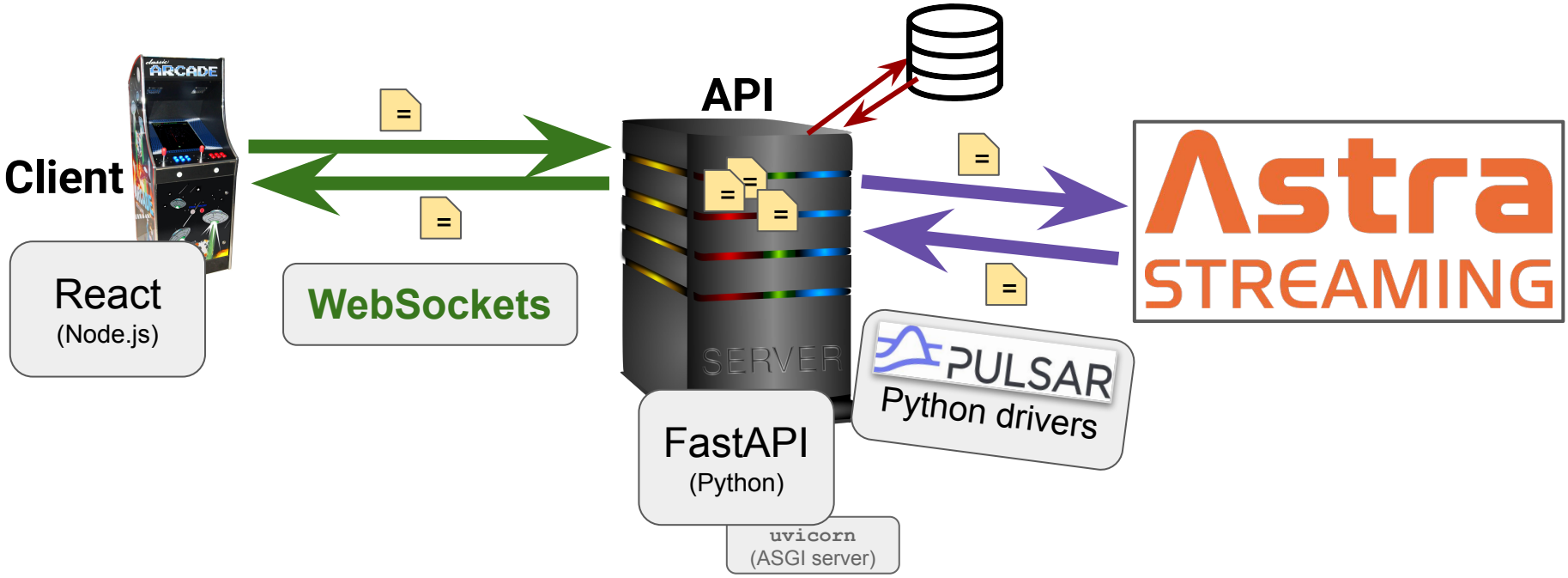


How do the parts work together ?

(Today we just scratch the surface.
Have a look at the README and the code!)



Game architecture



Technologies

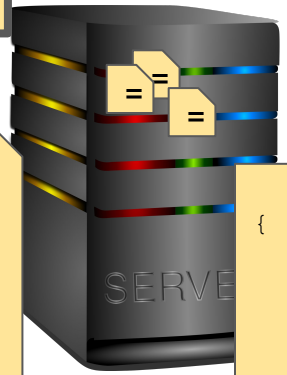
Client

```
{
  "messageType": "player",
  "payload": {
    "generation": 35,
    "name": "Mark",
    "x": 13,
    "y": 9
  }
}
```

validation
enrichment

```
{
  "messageType": "player",
  "playerID": "a40f3...",
  "payload": {
    "generation": 35,
    "name": "Mark",
    "x": 13,
    "y": 9
  }
}
```

API

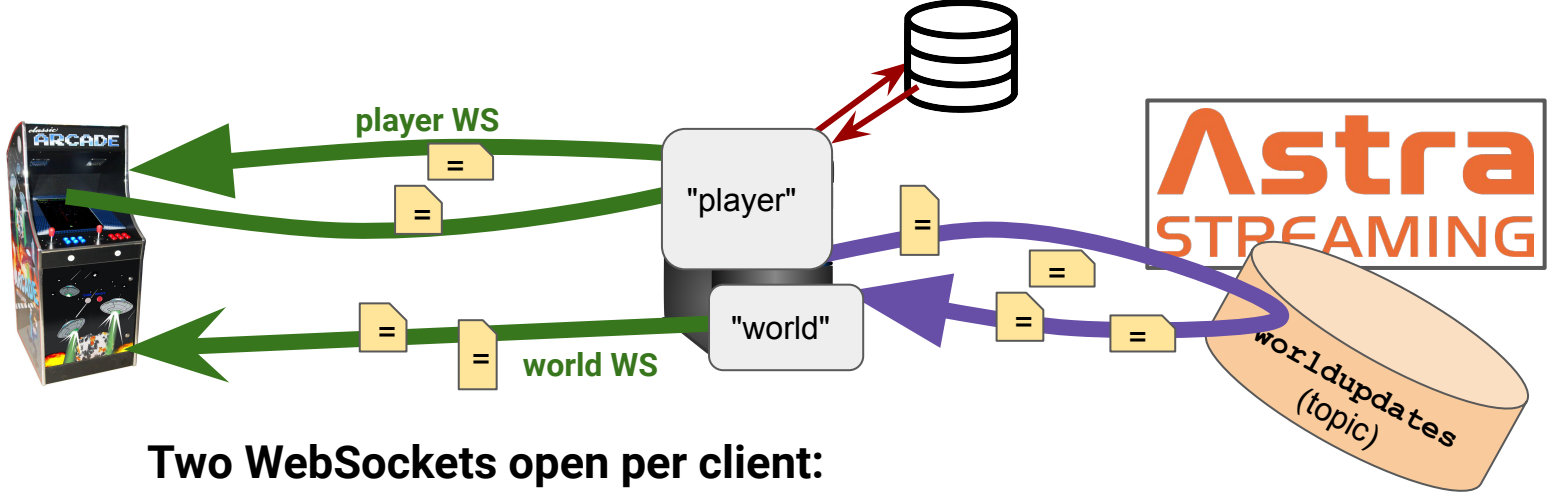



```
{
  "messageType": "player",
  "playerID": "a40f3...",
  "payload": {
    "generation": 35,
    "name": "Mark",
    "x": 13,
    "y": 9
  }
}
```

```
{
  "messageType": "player",
  "playerID": "a40f3...",
  "payload": {
    "generation": 35,
    "name": "Mark",
    "x": 13,
    "y": 9
  }
}
```

Astra
STREAMING

Messaging protocol



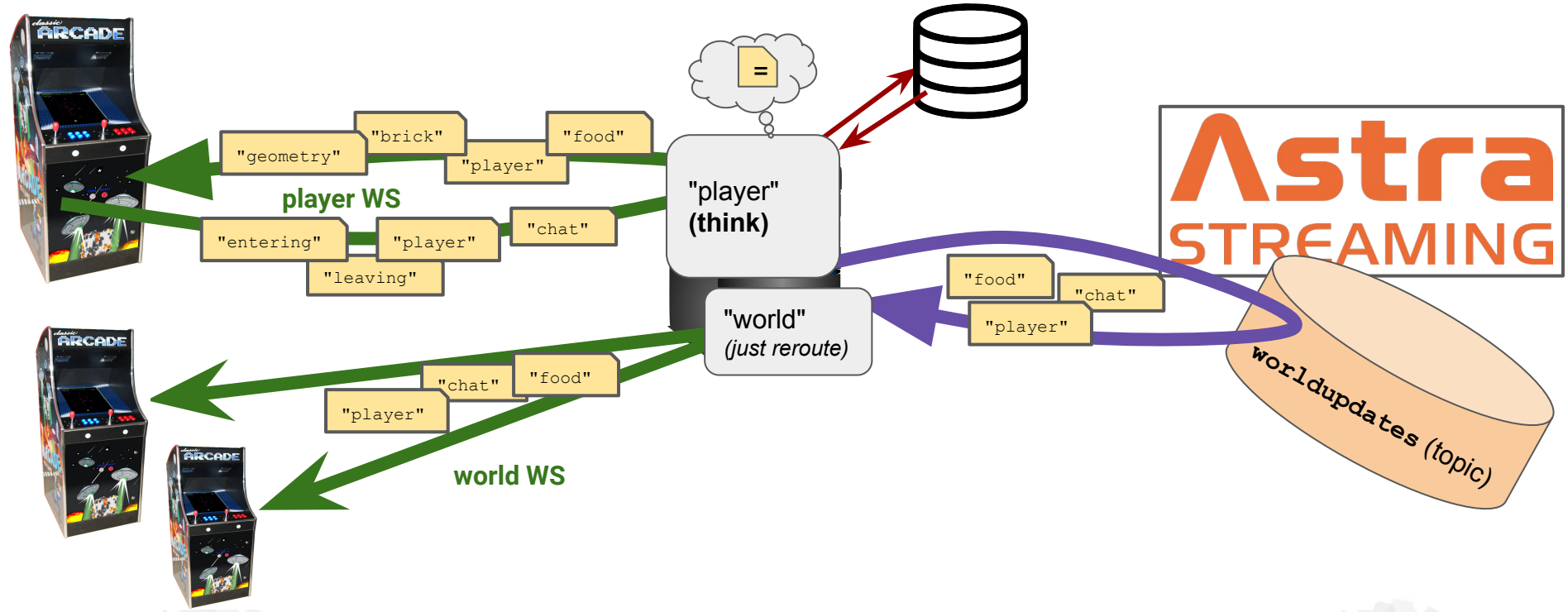
Two WebSockets open per client:

"player": client-to-server and server-to-THAT-client

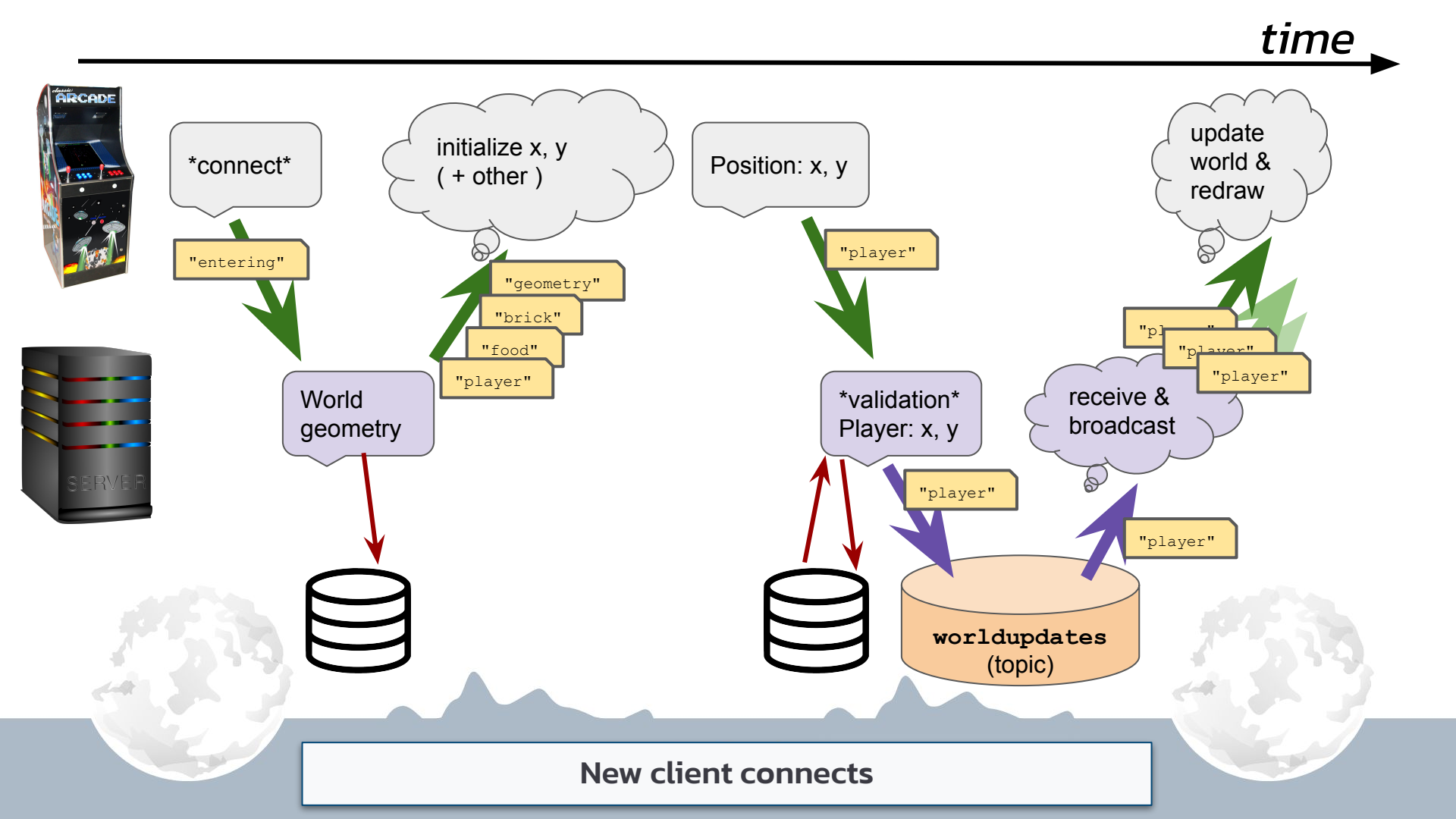
"world": picks up from Streaming and routes to ALL clients

All messages through a single Streaming topic:

- no schema, just (JSON) message serialization
- simplified setup, focus on core concepts
- in a real-life app: several topics is better



`messageType = . . .`


time
connect
"entering"
World geometry
initialize x, y
(+ other)
"geometry"
"brick"
"food"
"player"
Position: x, y
"player"
validation
Player: x, y
"player"
worldupdates
(topic)
update
world &
redraw
receive &
broadcast
"player"
"player"
"player"
New client connects

time

player action
(state change)

Position: x, y

"player"

is it about me?
⇒ update my x, y

update
world &
redraw

validation:
* boundaries
* other players
* obstacles
* food items

validation
Player: x, y

"player"

receive &
broadcast

"player"

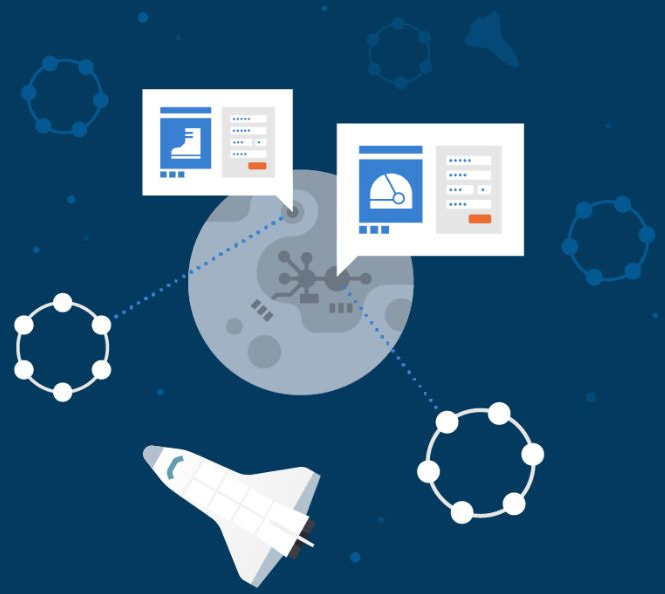
"player"

"player"

"player"

worldupdates
(topic)

Gameplay

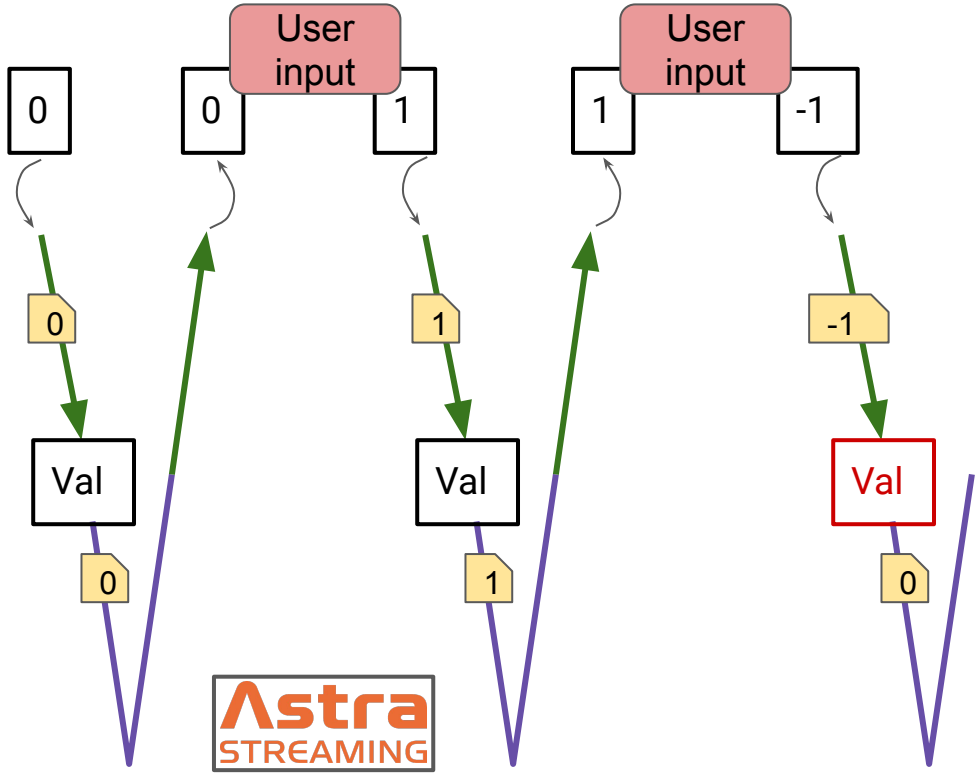


Case study: flickering spiders

a.k.a. *"Async messaging requires care"*

Position is in client memory and validated by the API

time



Astra STREAMING

Async and server-side validation / 1

Client receives position updates (and may fire new updates) *time*

User input

User input

3 3 4 5 4 5 4 5

3 4 5 4 5

Val Val Val Val Val

3 4 5 4 5

Astra STREAMING

Oh, no!
An infinite loop of spurious updates!

Async and server-side validation / 2

Solution: discard **obsolete** server updates

time

A client-side ever-increasing "generation number"

User input

User input

3 0

3 0

4 1

5 2

2

5 2

3 0

4 1

5 2

Val

Val

Val

Astra STREAMING

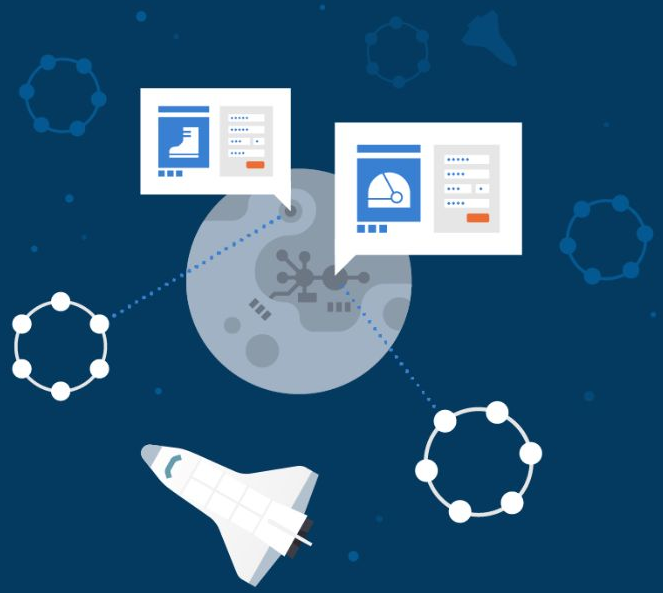
Message has "1", client is at "2": update discarded. ⇒ Loops disappear.

Async and server-side validation / 3

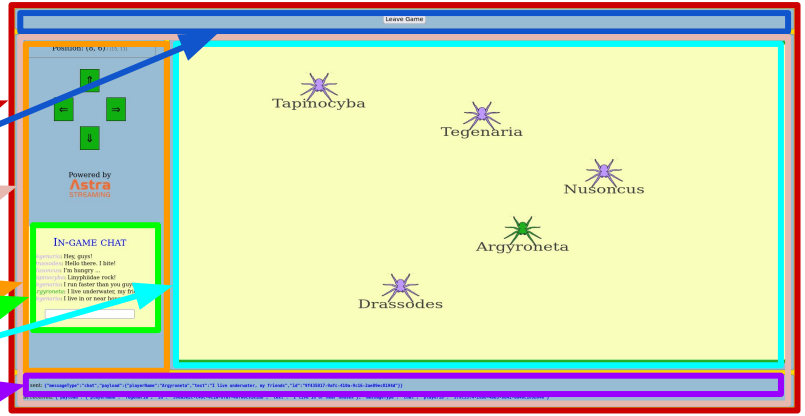
HANDS-ON #2: Run & Play

- Open in Gitpod
- Launch the API
  - ... giving it all secrets
- Launch the client
- Play the game !
- *(Optional)* invite your friends

Additional topics



```
1  let pws = null
   let wws = null

2  const App = () => {

3    React_States = useState(...)
     React_References = useRef(...)

     keyDown_Handler = ...

4    useEffect( () => {
       if(inGame) {
         if ( both_Websockets === null ){
           both_Websockets = new Websocket(apiAddress)
           wws.onmessage = evt => {
5            parse_and_interpret(evt ...)
             // check messageType (player/geometry/chat)
             // HERE WE MODIFY REACT STATE AND NEED REFERENCES
           }
         }
       } else {
6        pws.send(goodbyeMessage)
         both_Websockets.disconnect()
         both_Websockets = null
       }
     }, [inGame])

     useEffect( () => {
7      pws.send(myPlayerUpdate ...)
     }, [coordinates, etc])

     return (
8      App
         PlayerForm
         GameArea     // if inGame
           sidebar
             GameInputs
               buttons, etc
               ChatArea
           GameField
           statusbar
     )

   }
```



Client app structure

WebSocket on client side: **subscription to events**

Problem: we're closing over a React state

```
import { useState } from "react"

const [generation, setGeneration] = useState(0)

// WebSocket event subscription
wws.onmessage = evt => {
  // Stale value because of closure!
  console.log(generation)
}
```

```
import { useState, useRef } from "react"

const [generation, setGeneration] = useState(0)

const generationRef = useRef()
generationRef.current = generation

// WebSocket event subscription
wws.onmessage = evt => {
  // This is what we want!
  console.log(generationRef.current)
}
```

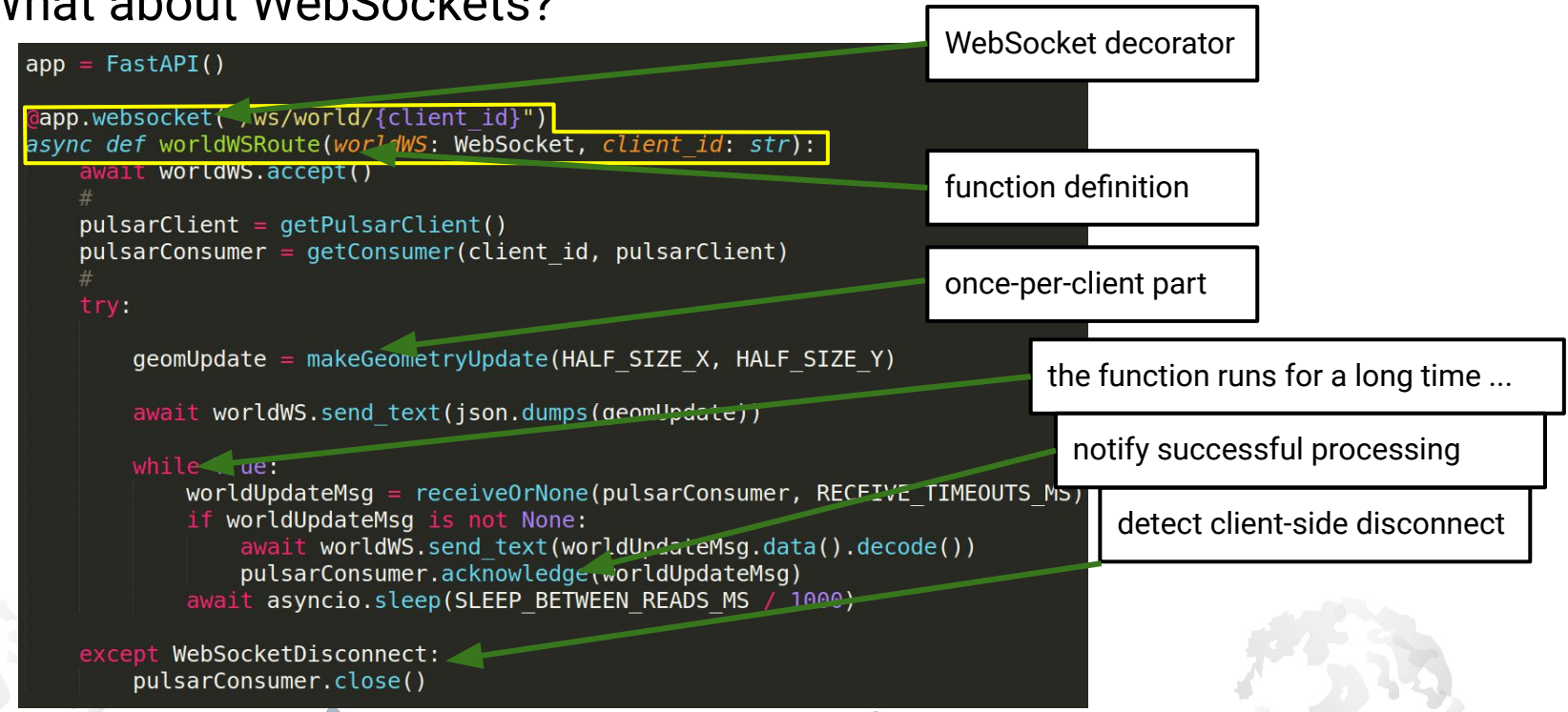
Solution: `useRef` to access a state within a closure!

A modern, high-perf Python library to create APIs

- WebSockets support
- native **`async/await`** support in route functions
- Minimal boilerplate
- Sophisticated route dependency handling
- Asynchronous post-response task support
- Well documented (`fastapi.tiangolo.com`)

Routes = functions, return value = response (etc etc…)

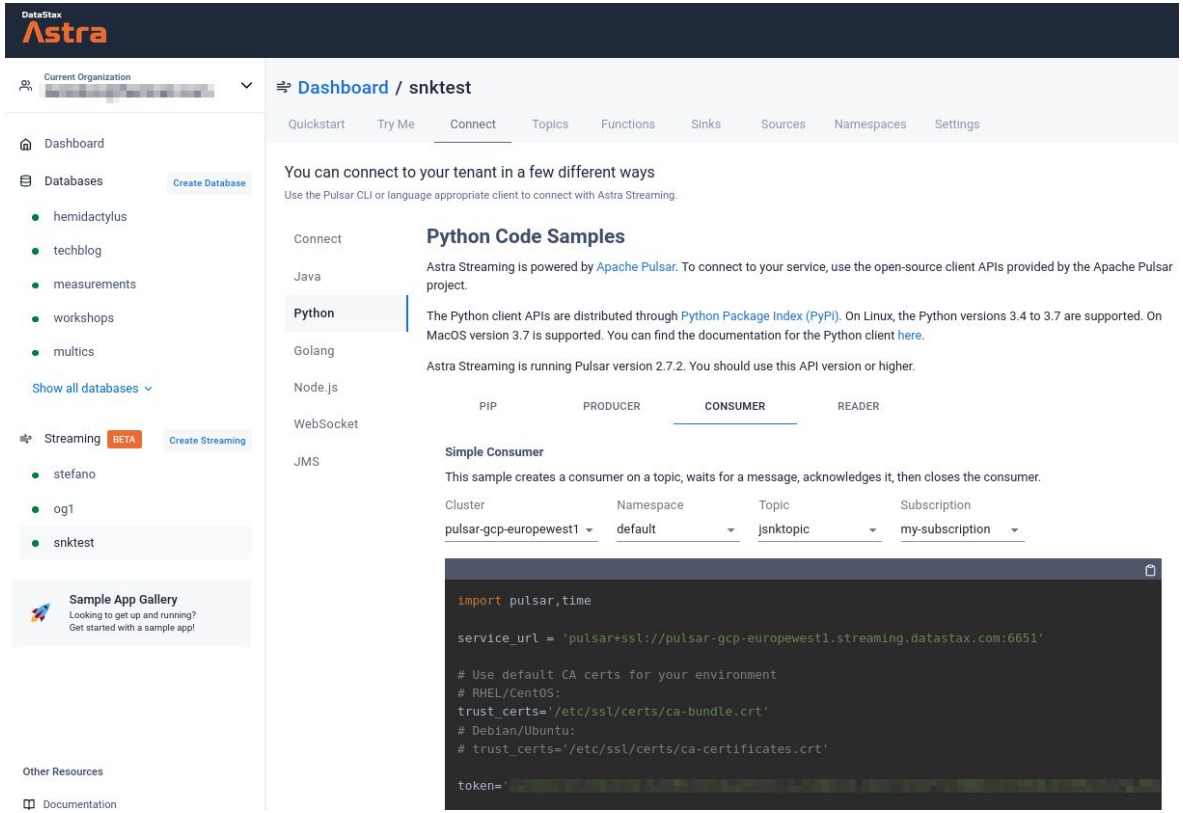
What about WebSockets?



FastAPI, code layout

```
# pip install pulsar-client==2.7.3
import pulsar

client = pulsar.Client(
  SERVICE_URL,
  authentication=pulsar.AuthenticationToken(ASTRA_TOKEN),
  tls_trust_certs_file_path=TRUST_CERTS,
)

streamingTopic = 'persistent://gameserver-0123/default/worldupdates'

# consume messages
consumer = client.subscribe(streamingTopic, 'this-subscription-id')
receivedMessage = consumer.receive(timeout)
# (process message ...)
consumer.acknowledge(receivedMessage)

# alternatively: client.subscribe(... message_listener=myListenerFunction)

# produce messages
producer = client.create_producer(streamingTopic)
pulsarProducer.send('{"message":"hello!"}'.encode('utf-8'))
```

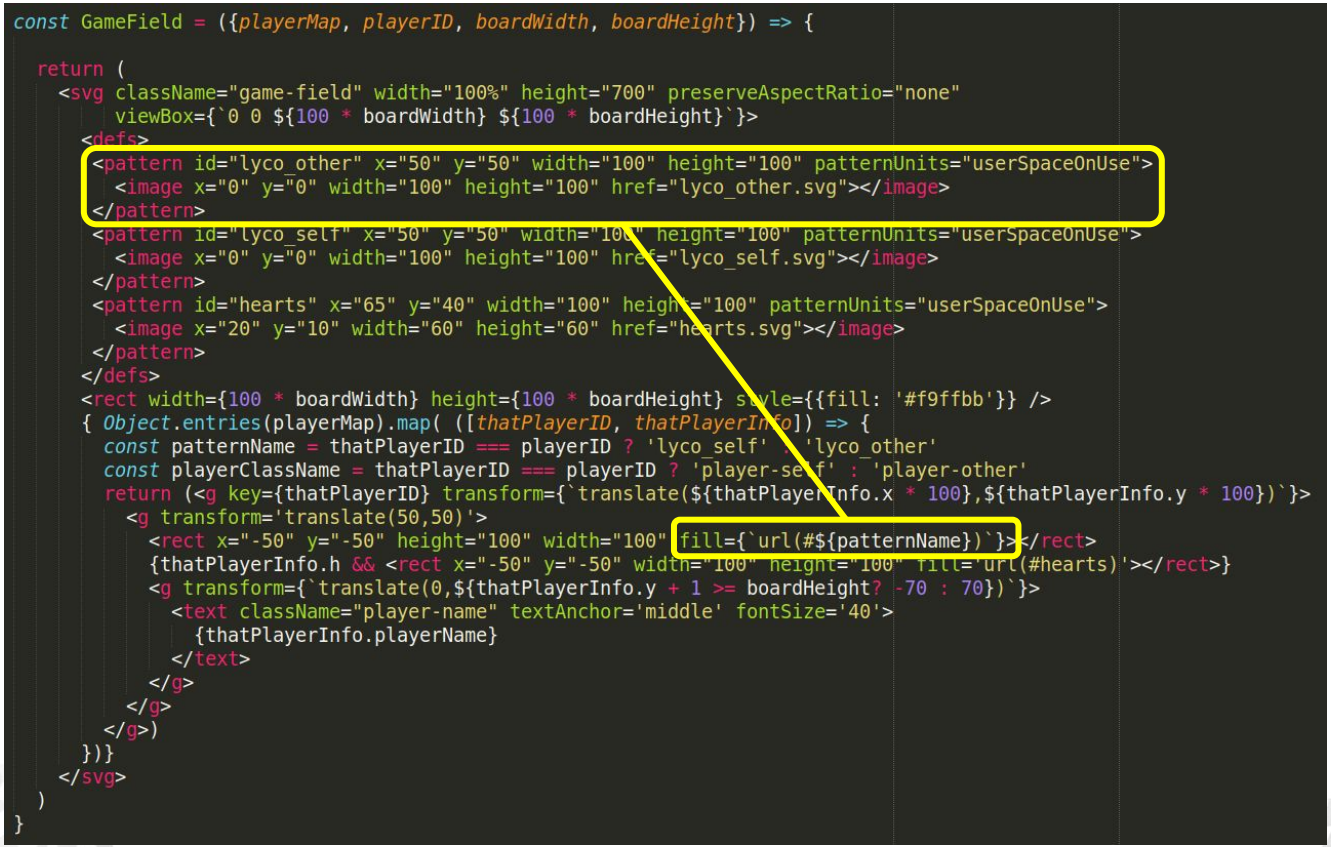
Pulsar drivers for Python



Python quickstart on Astra Streaming UI

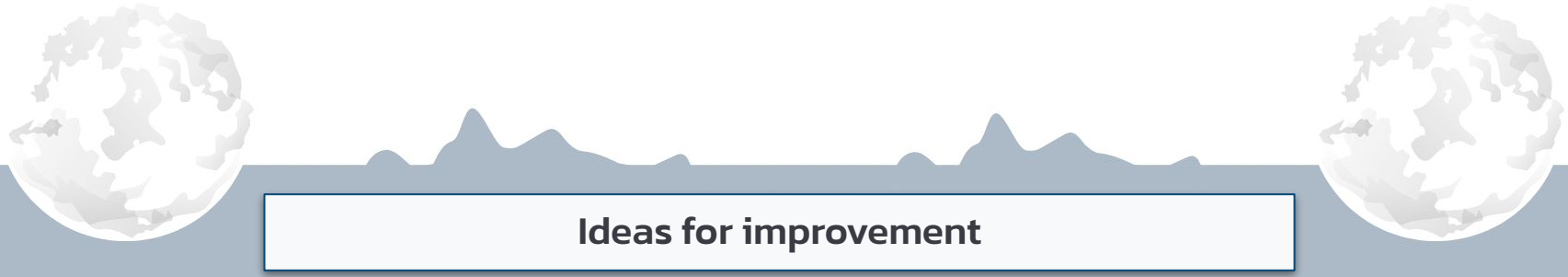
```python
try:
    # first we tell this client how the game-field looks like
    geomUpdate = makeGeometryUpdate(HALF_SIZE_X, HALF_SIZE_Y)
    # Note: this message is built here (i.e. no Pulsar involved)
    # and directly sent to a single client, the one who just connected:
    await worldWS.send_text(json.dumps(geomUpdate))
    while True:
        worldUpdateMsg = receiveOrNone(p
        if worldUpdateMsg is not None:
            # We forward any update from
            # for all clients out there:
            await worldWS.send_text(worl
        await asyncio.sleep(SLEEP_BETWEE
except WebSocketDisconnect:
```

```python
def receiveOrNone(consumer, timeout):
    """
    A modified 'receive' function for a Pulsar topic
    that handles timeouts so that when the topic is empty
    it simply returns None.
    """
    try:
        msg = consumer.receive(timeout)
        return msg
    except Exception as e:
        if 'timeout' in str(e).lower():
            return None
        else:
            raise e
```



React & SVG Graphics ("the sprite way")

- **schemaful** messages at Pulsar level (⇒ several topics)
- a **sink** to automate persistence to Astra DB (requires schemaful)
- More advanced "speculative client-side rendering"



Ideas for improvement

01

Today's workshop
Housekeeping

02

Messaging systems
"why not just a DB?"

03

Storage
Persisting game status

04

Pulsar / Astra streaming
A modern message system

05

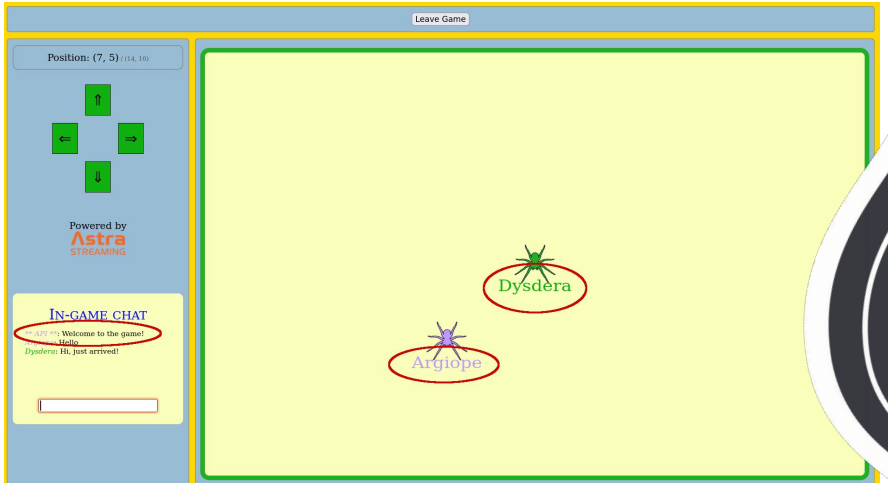
Game architecture
Highlights of this game

06

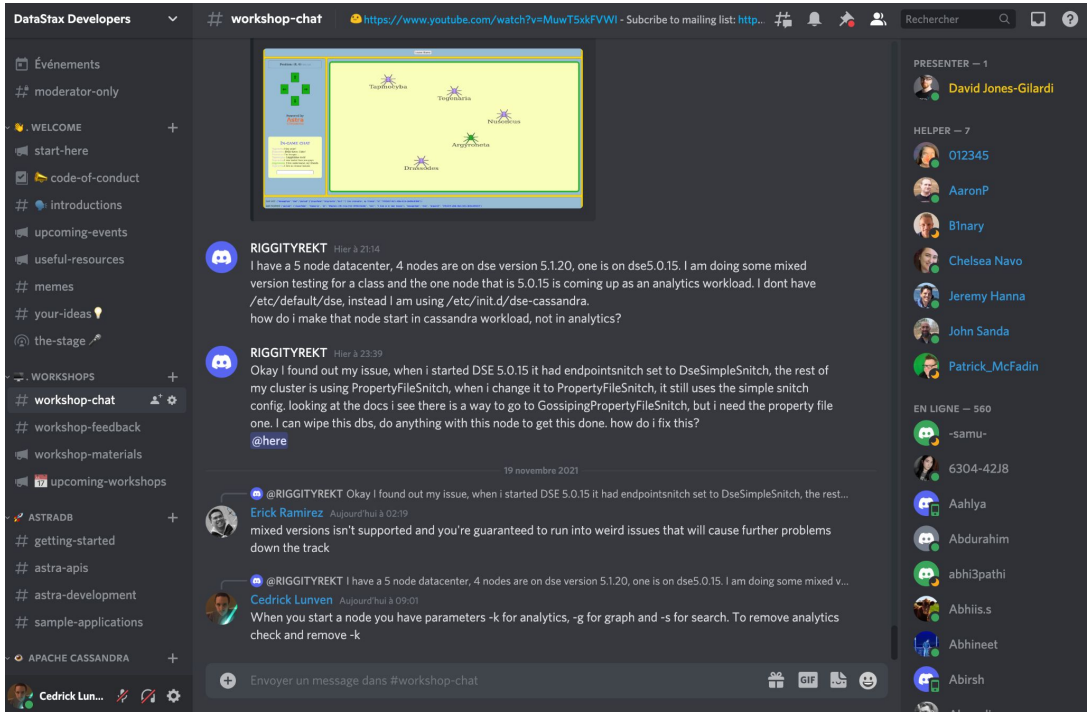
What's next?
Quiz & Homework

Agenda

Questions + coding exercise



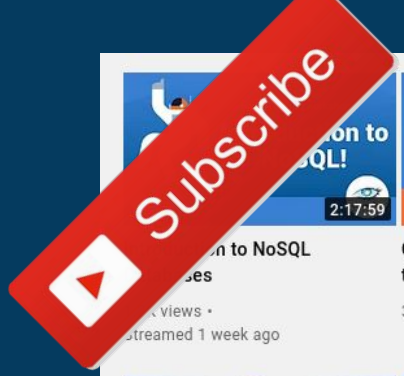
Lab & Badge !

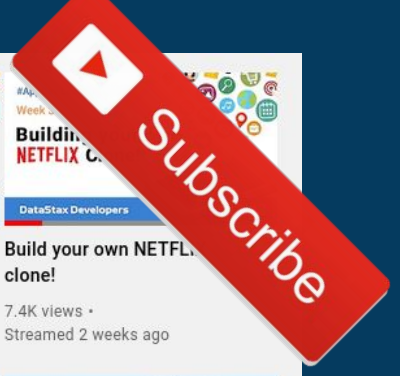


!discord

dtsx.io/discord

Datastax Developers Discord (18k+)





…n to NoSQL …ses

…k views •
…Streamed 1 week ago

Crash Course | Introduction to Cassandra for Developers

331 views • 1 week ago

Introduction to NoSQL Databases

3.5K views •
Streamed 1 week ago

Introduction to NoSQL Databases

10K views •
Streamed 1 week ago

Build your own NETFLIX clone!

4K views •
Streamed 2 weeks ago

Build your own NETFL… clone!

7.4K views •
Streamed 2 weeks ago

Astra Streaming Demo

177 views • 2 weeks ago

Kubernetes Ingress Management with Traefik…

496 views •
Streamed 2 weeks ago

Build your own TikTok clone!

1.9K views •
Streamed 3 weeks ago

Build your own TikTok Clone!

4K views •
Streamed 3 weeks ago

How to use the Connect Driver in Astra DB

113 views • 4 weeks ago

How to use the CQL Console in Astra DB

39 views • 4 weeks ago

How to create an Authentication Token in…

37 views • 4 weeks ago

How to use the Data Loader in Astra DB

62 views • 4 weeks ago

Astra DB Sample App Gallery

36 views • 4 weeks ago

How to use Secure Connect in Astra DB

42 views • 4 weeks ago

Cassandra Day India: CL Room (Workshops)

2.4K views •
Streamed 4 weeks ago

Cassandra Day India: RF Room (Talks)

1.3K views •
Streamed 1 month ago

Thank You!