

# 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



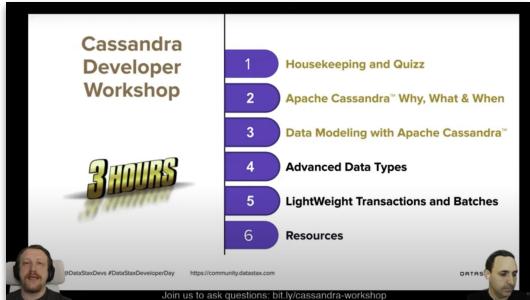
# Housekeeping

Get the most out of this workshop !



Live and interactive

**Livestream:** [youtube.com/DataStaxDevs](https://youtube.com/DataStaxDevs)

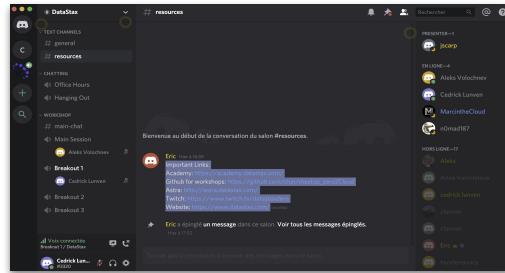


YouTube



Twitch

**Questions:** <https://dtsx.io/discord>



**Discord**

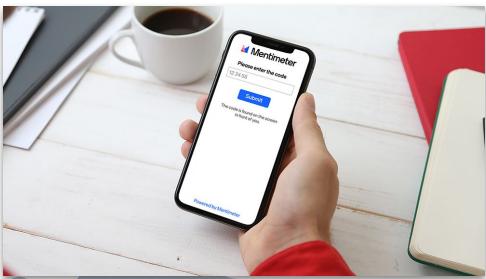


**YouTube**



Available on the iPhone App Store

**Games** [menti.com](https://menti.com)



Mentimeter

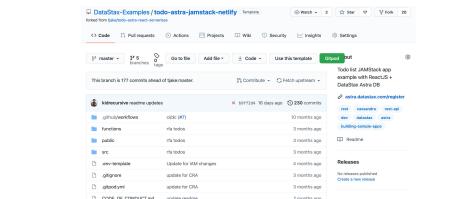


GET IT ON  
Google play

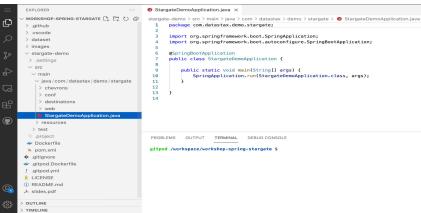
Attend the live sessions

*Nothing to install !*

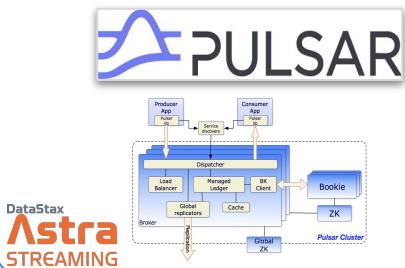
**Source code + exercises + slides**



IDE



## Messaging system (based on Pulsar)

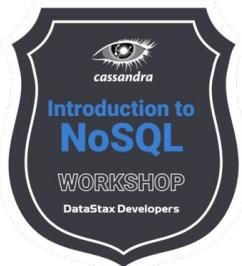


Database + GraphQL + PlayGround



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**

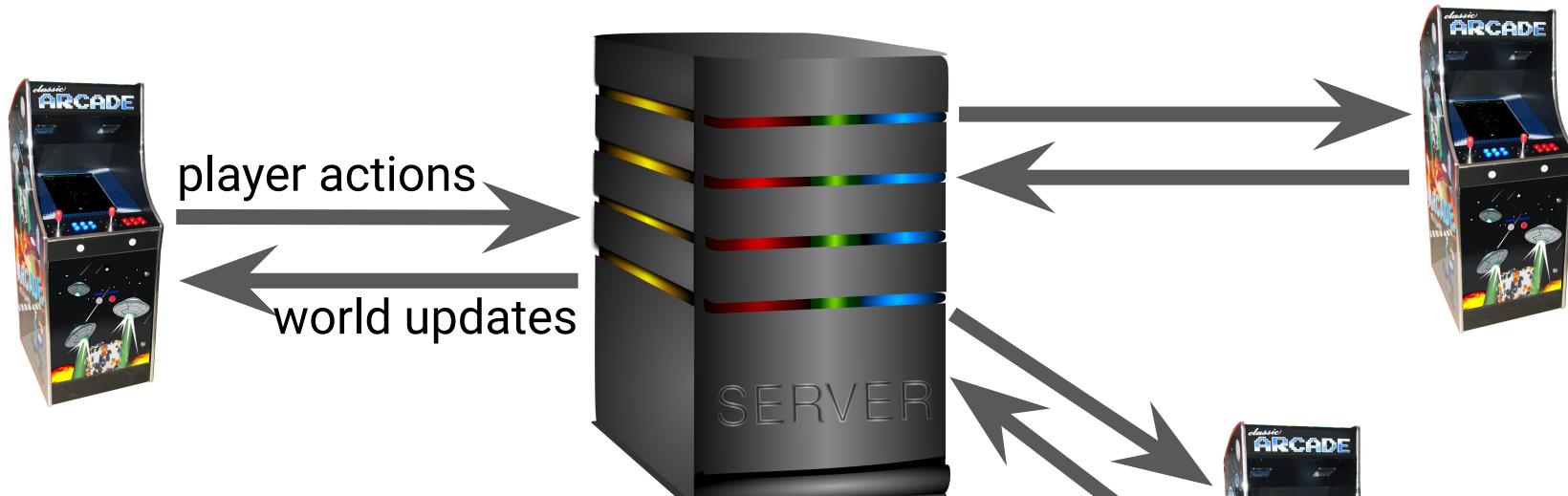


**Agenda**

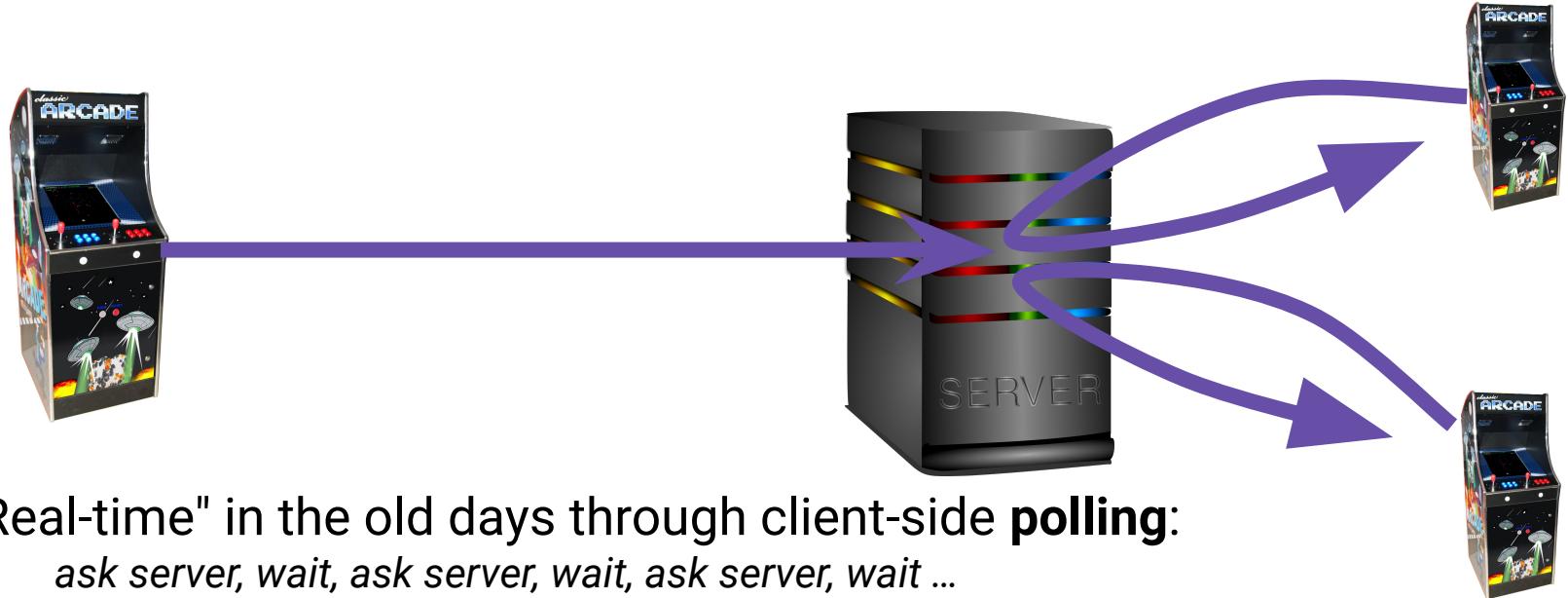
# Messaging systems

What engine will power the game?



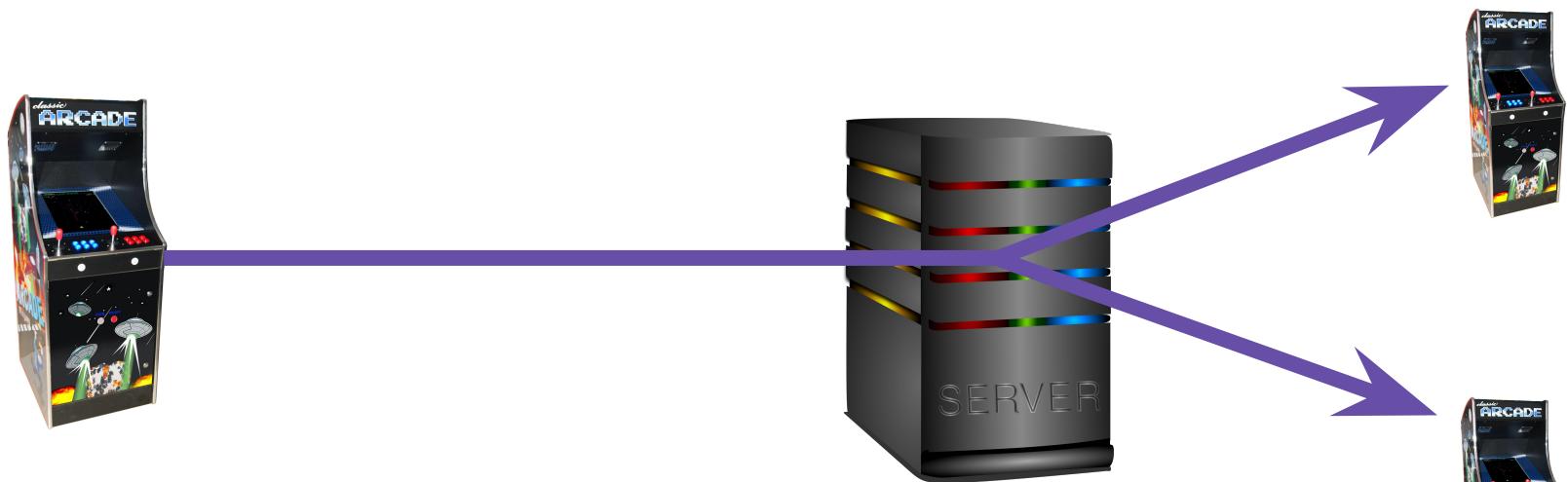


So, you want to build a multiplayer real-time game ... ?



"Real-time" in the old days through client-side **polling**:  
*ask server, wait, ask server, wait, ask server, wait ...*

- ✗ A flood of request-and-responses (overhead!)
- ✗ Not really real-time (latencies)
- ✗ Waste of server resources
- ✗ 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.

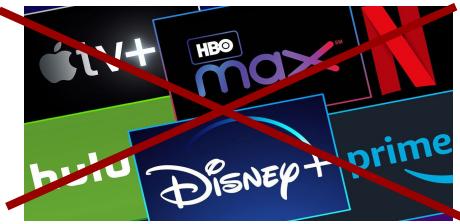
⇒ We need a messaging platform

⇒ And server-initiated communication

- "messaging systems": we DO NOT mean this



- "streaming": we DO NOT mean this

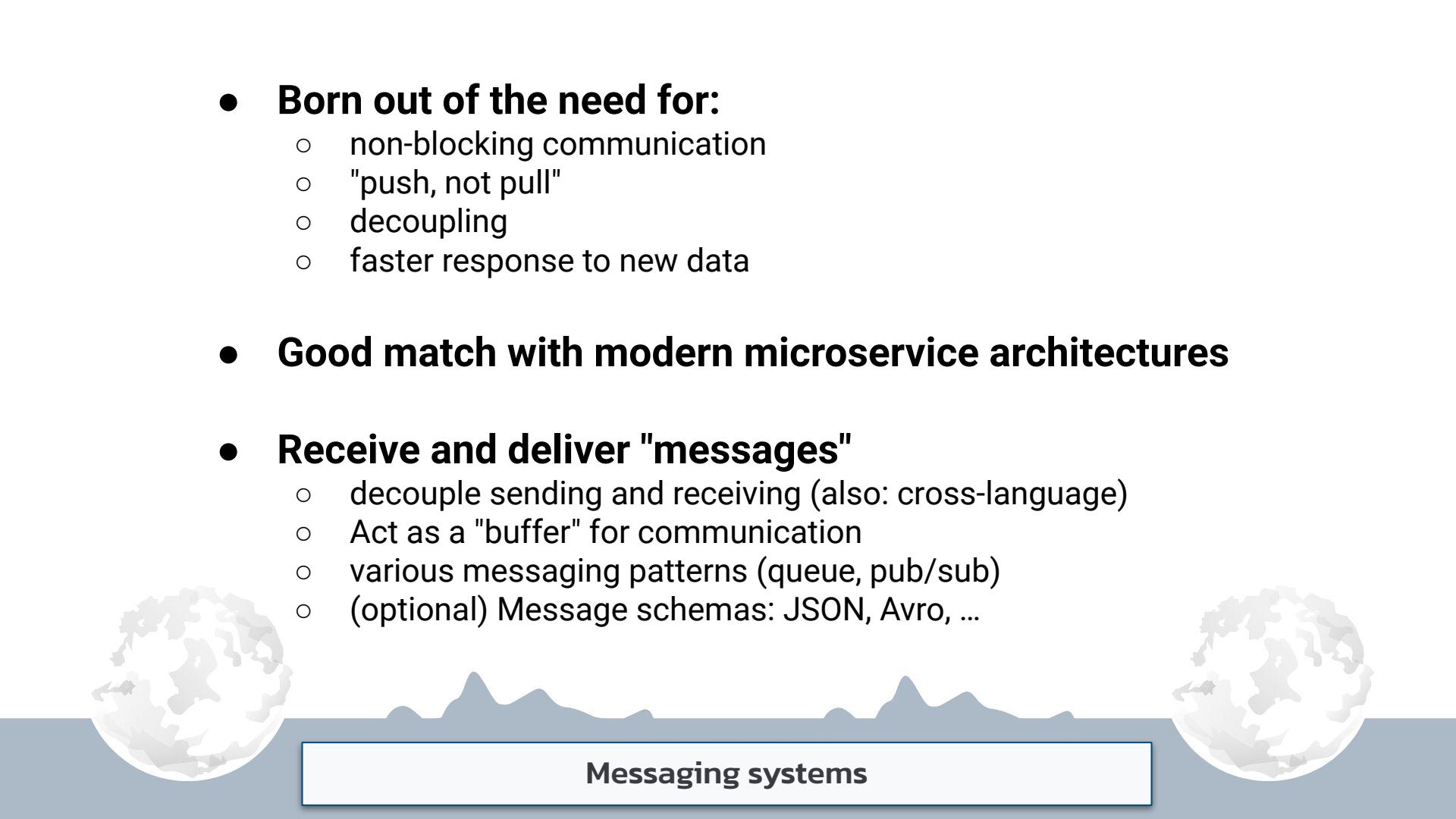


- We rather mean: **backend services specialized to convey data between other backend systems. Data that is made progressively available in small pieces as time goes by.**



Messaging, streaming

- **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, ...

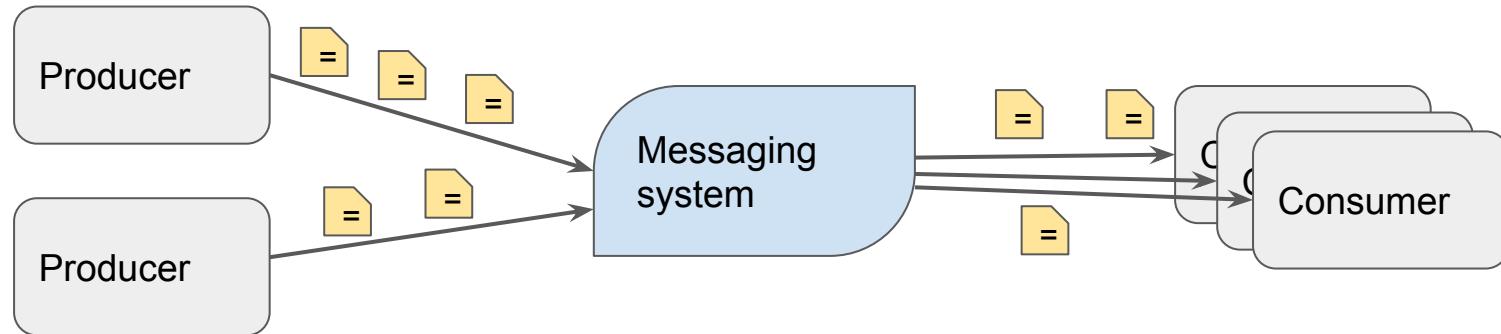


Messaging systems

- **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 ...



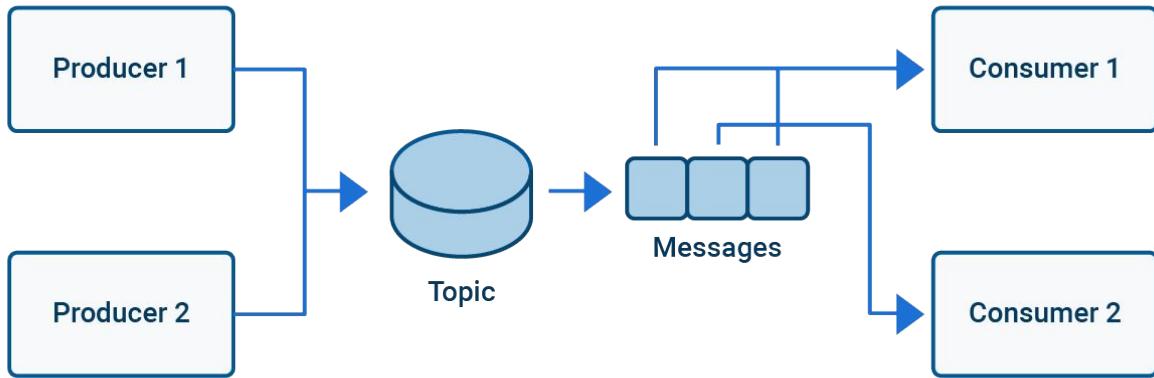
Messaging systems



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



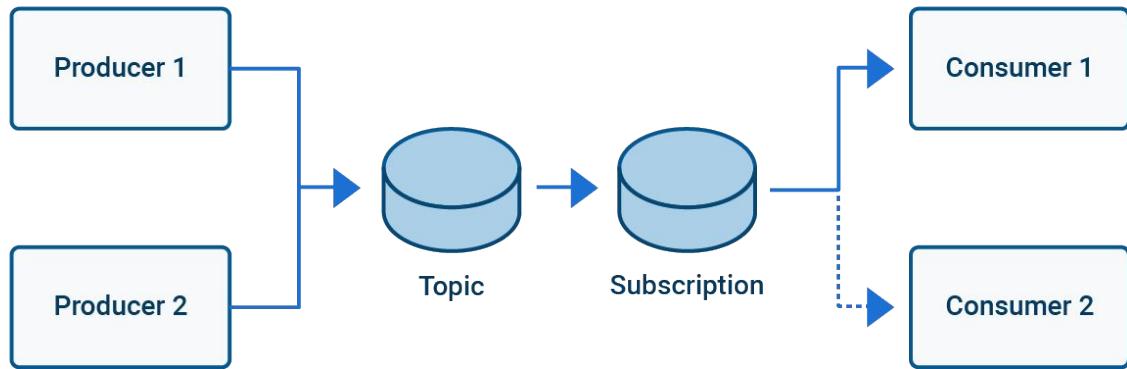
## QUEUING



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

Consumption pattern 1: Queuing

## PUB-SUB



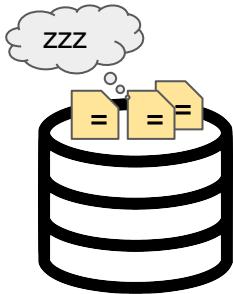
- Producers publish to Topic
- Consumers subscribe to Topic
- Multiple consumers
- Easy to extend

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



Consumption pattern 2: Pub-Sub





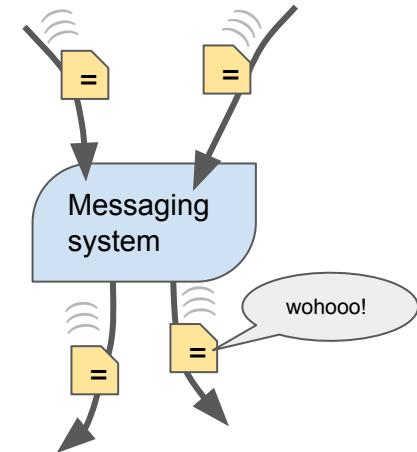
Databases  
*data at rest*

(in "tables/collections")



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

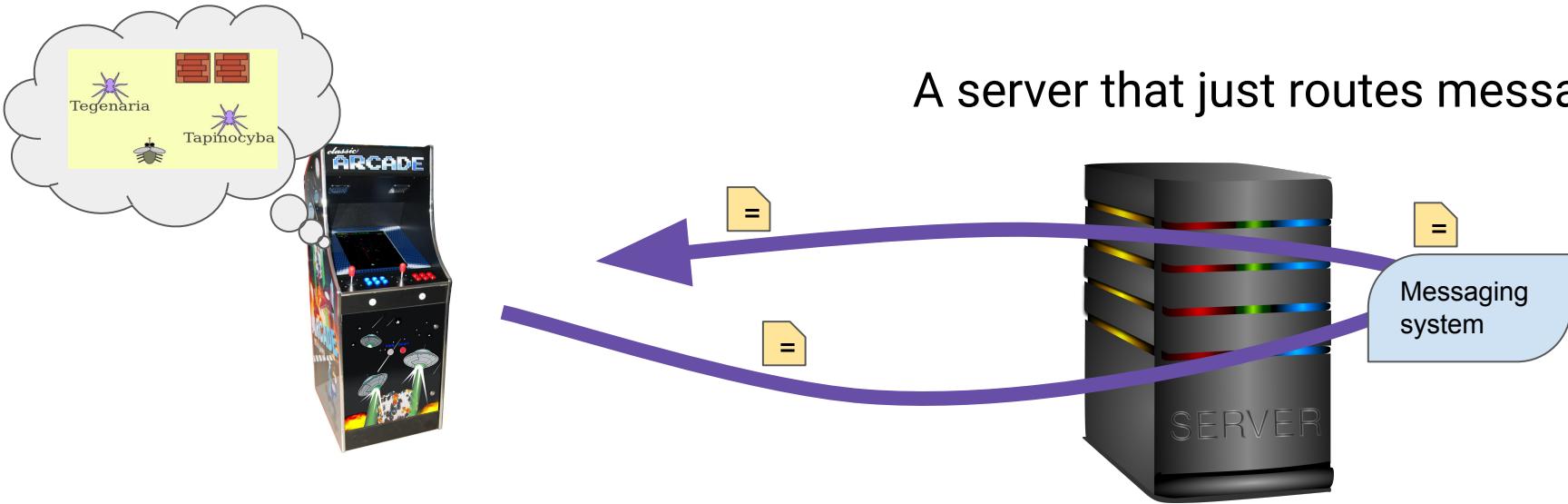


A space-themed illustration featuring a white jet flying from the bottom left towards two green alien invaders at the top. A dashed green line connects the jet to a central orange planet with white clouds. A small white circle is on the planet's surface, connected by a line to a blue square icon containing a bar chart and other data symbols.

# Storage

Making the game richer with server-side persistent storage

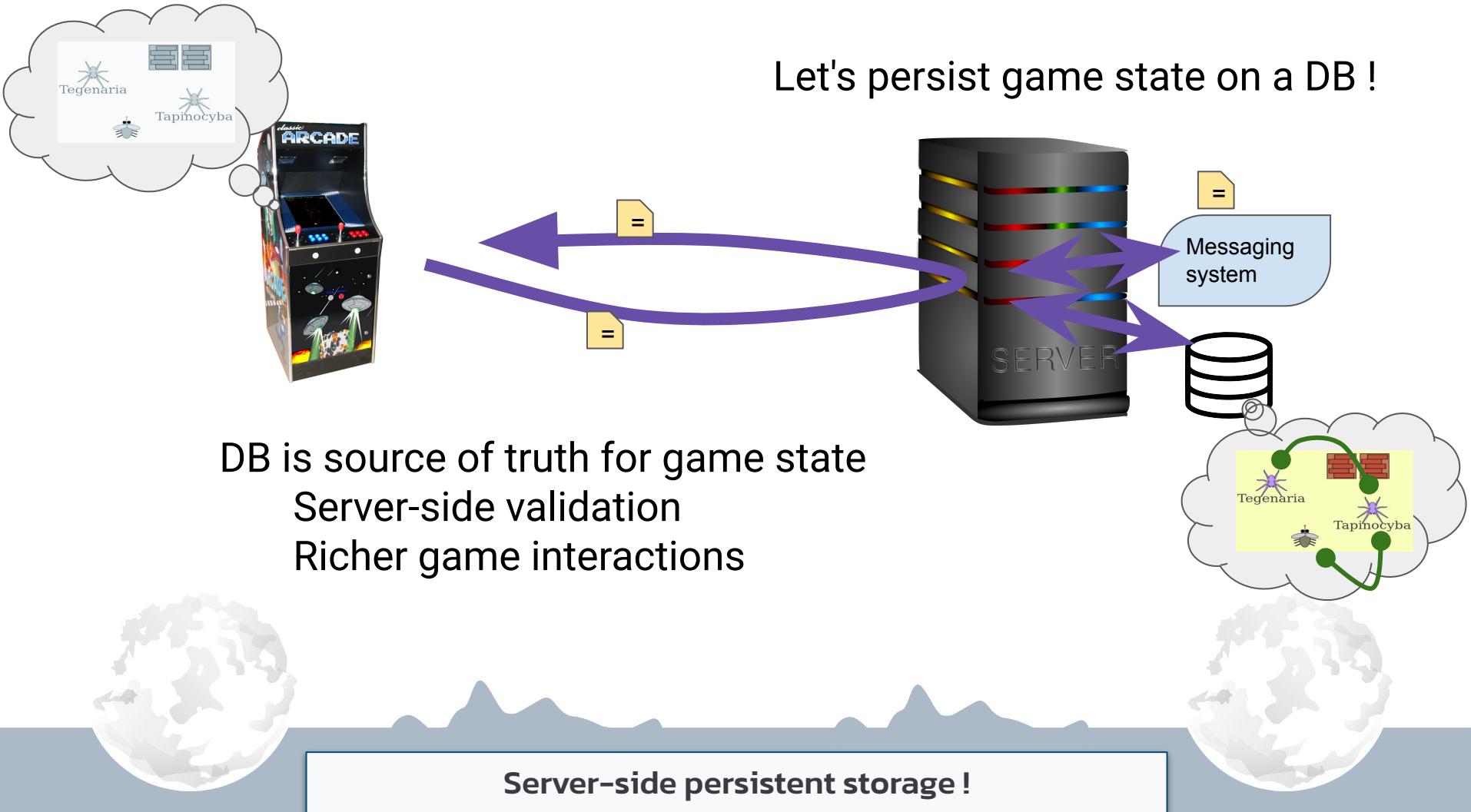
## A server that just routes messages

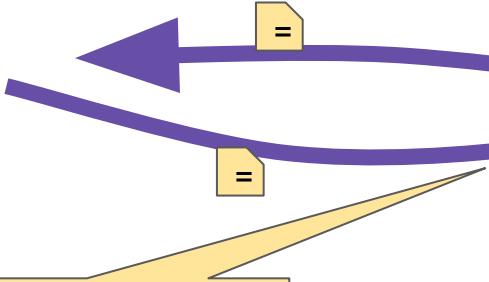


Knowledge of game world on client-side only  
(cf. "event-sourcing" architectures...)  
Limited player-player and player-world interactions

A simple "game" (no storage)

# Let's persist game state on a DB !





### Messages look like:

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



Messaging system



### 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 | 23 |
| 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 | 6
| 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

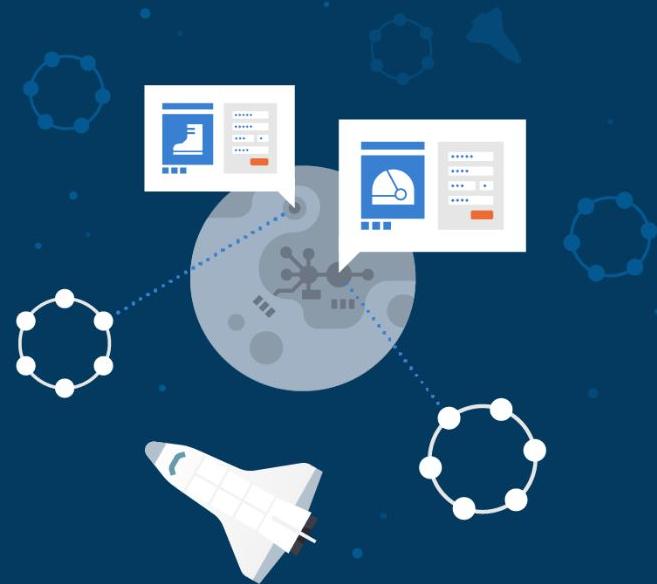
**What's next?**  
**Quiz & Homework**



**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!



**(Distributed) messaging done right**



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

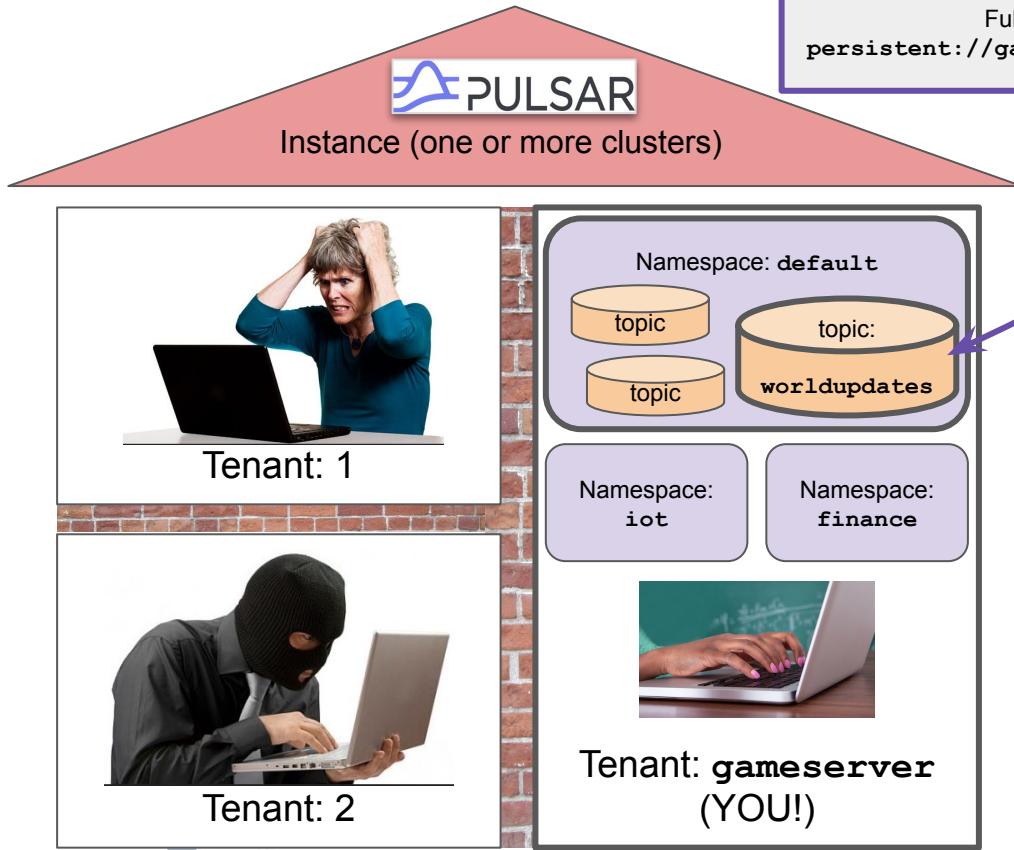


+ schemas, functions, Pulsar SQL, deduplication, IO (sinks/sources) ...

- 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)

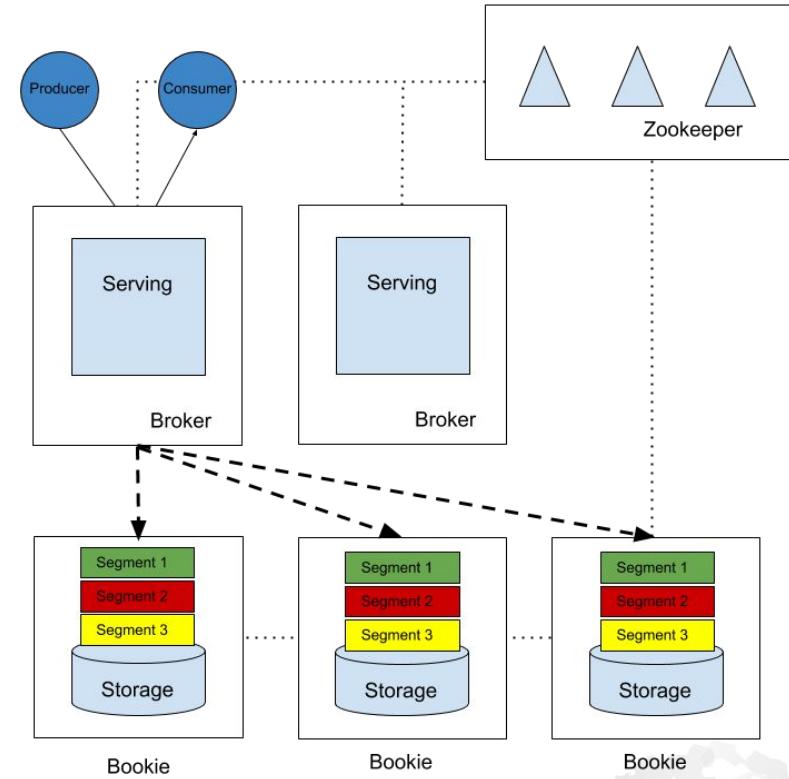


Apache Pulsar ⇒ Astra Streaming

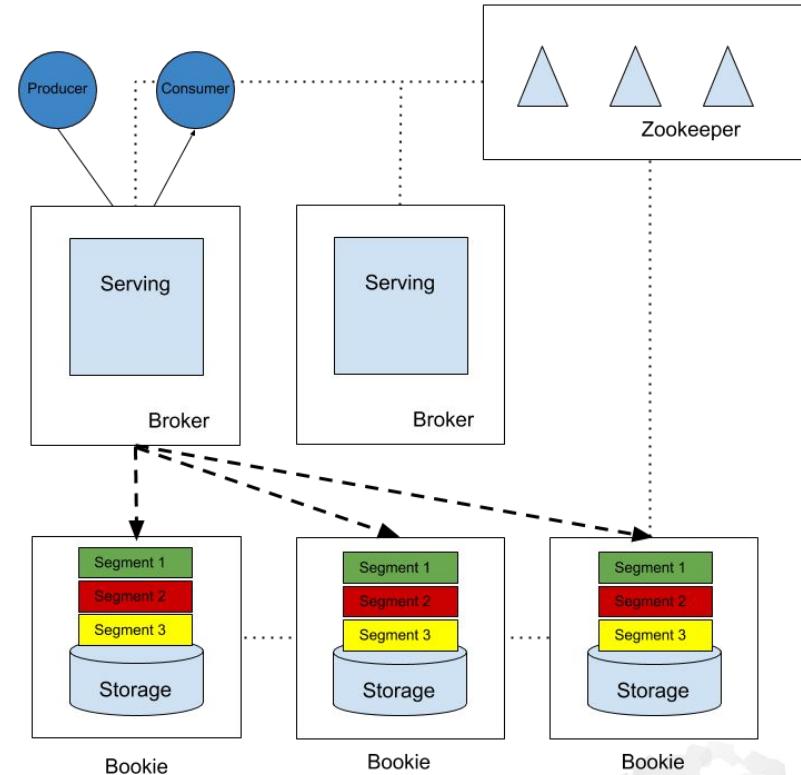


Pulsar messaging, logical structure

- 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





## 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**

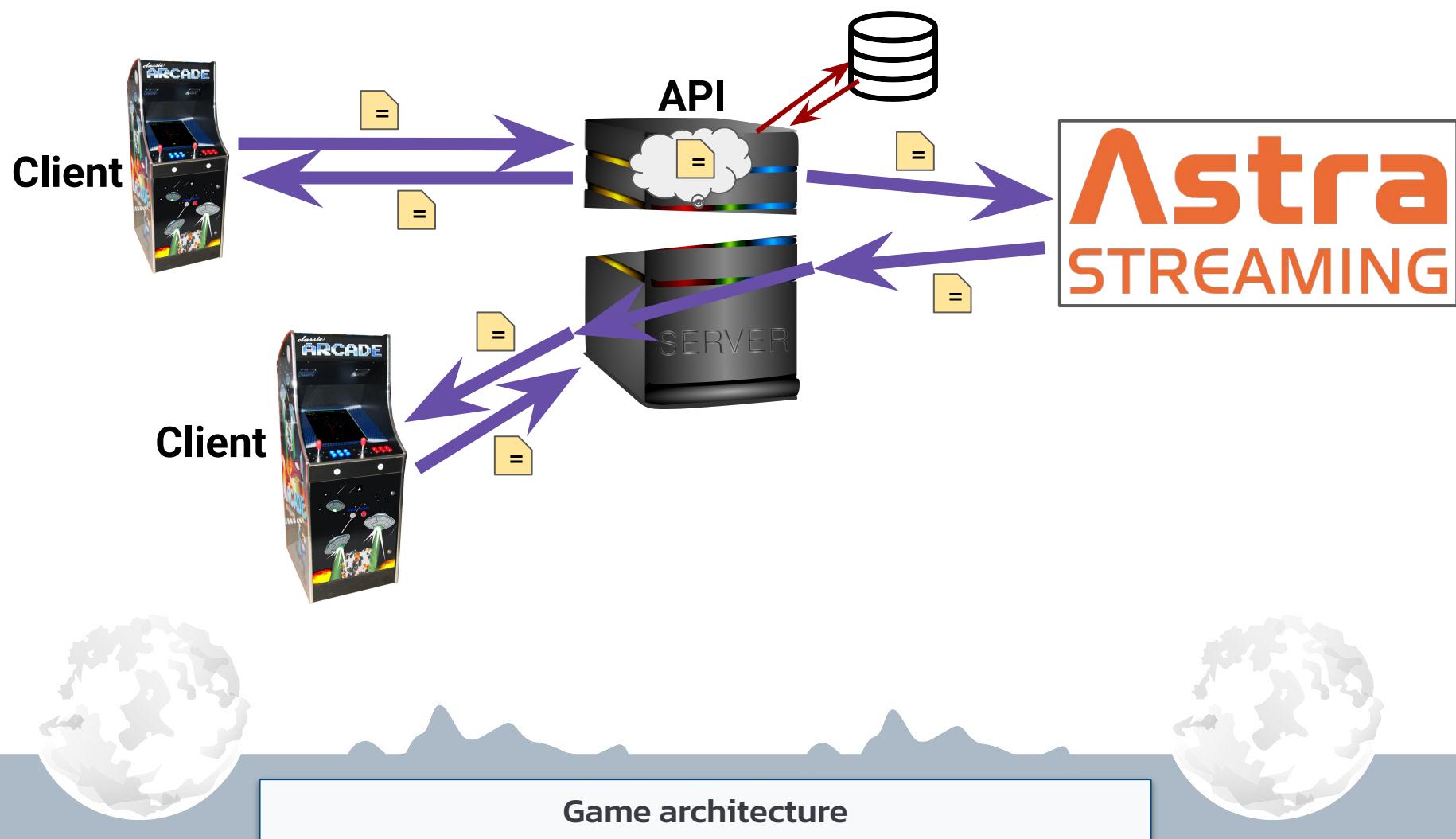
An illustration of four astronauts in white spacesuits working on a satellite in space. One astronaut is on the left, another is on the right, and two more are on the top. The satellite has a large dish antenna and solar panels. The background is a dark blue circle representing Earth's horizon.

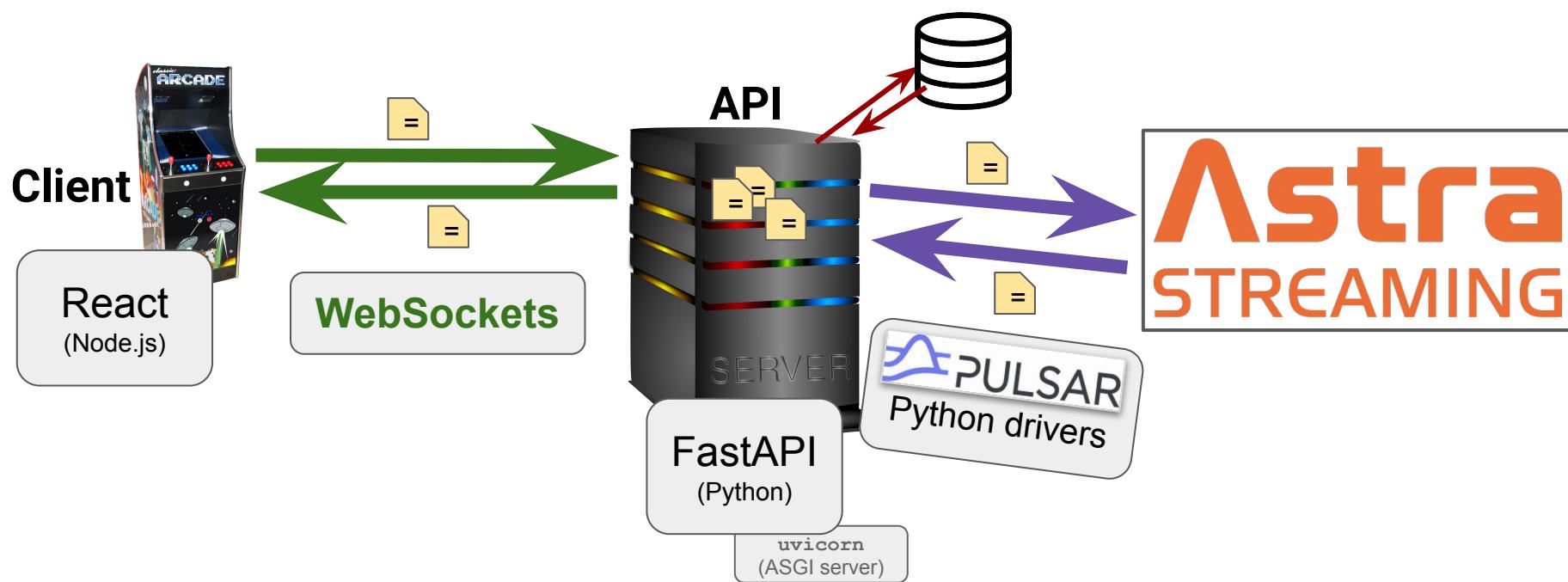
# Game architecture

How do the parts work together ?

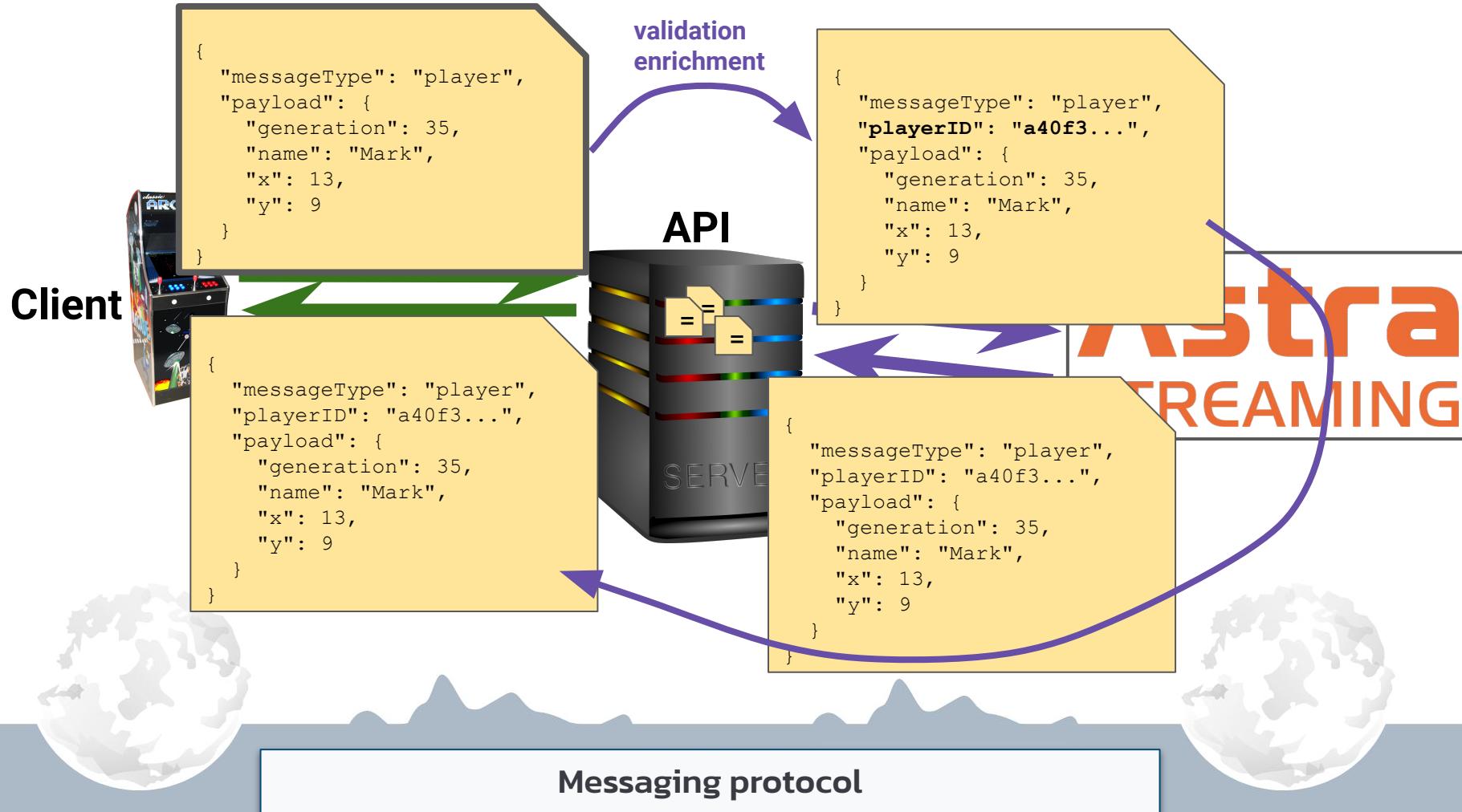
*(Today we just scratch the surface.  
Have a look at the README and the code!)*

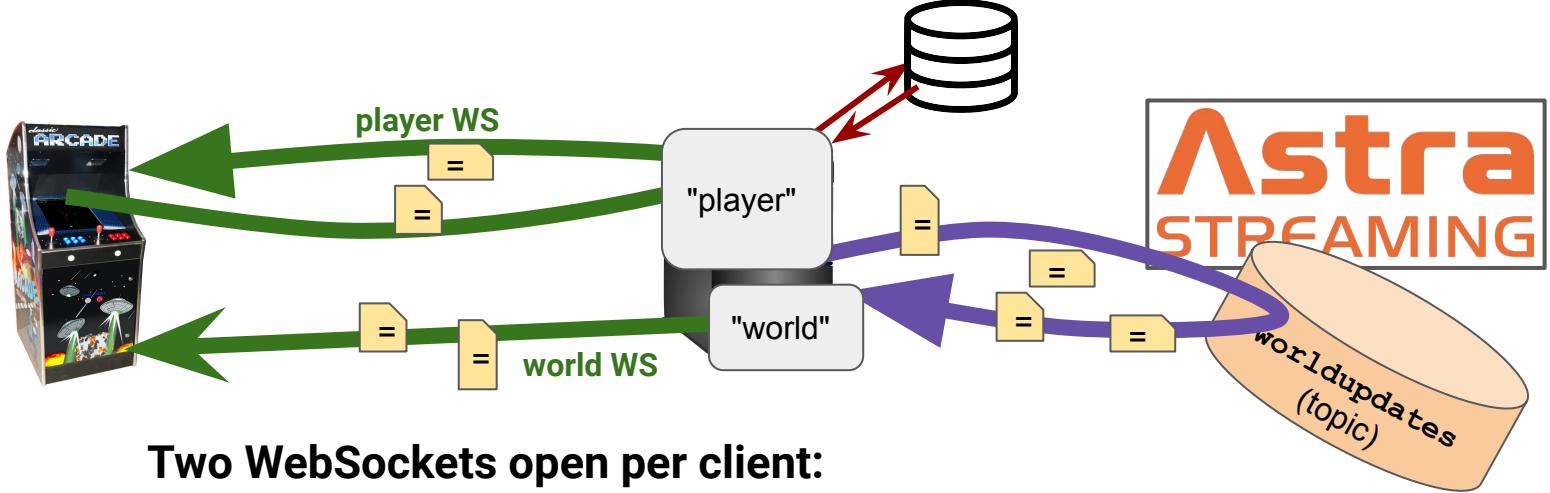






Technologies





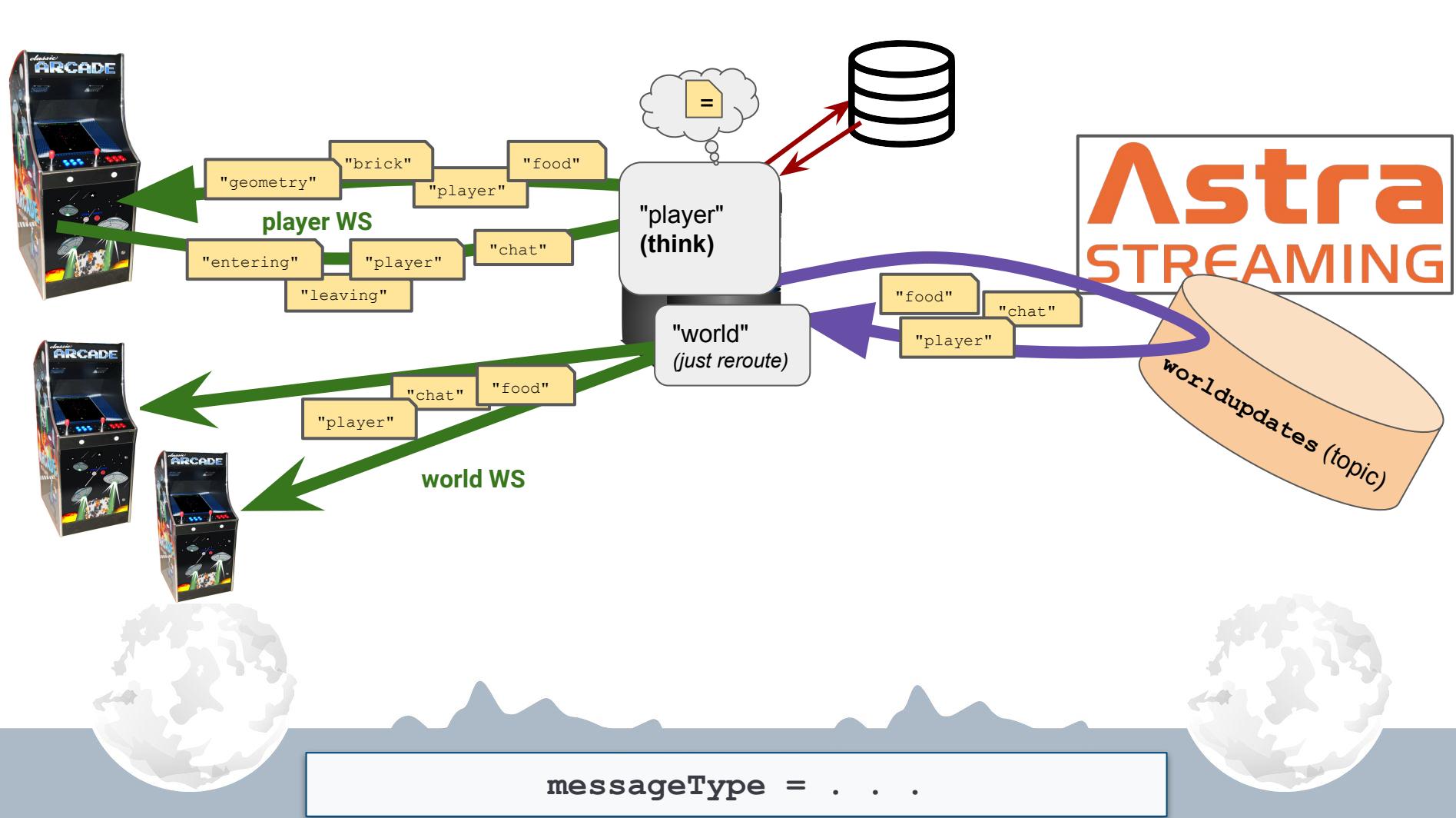
## Two WebSockets open per client:

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

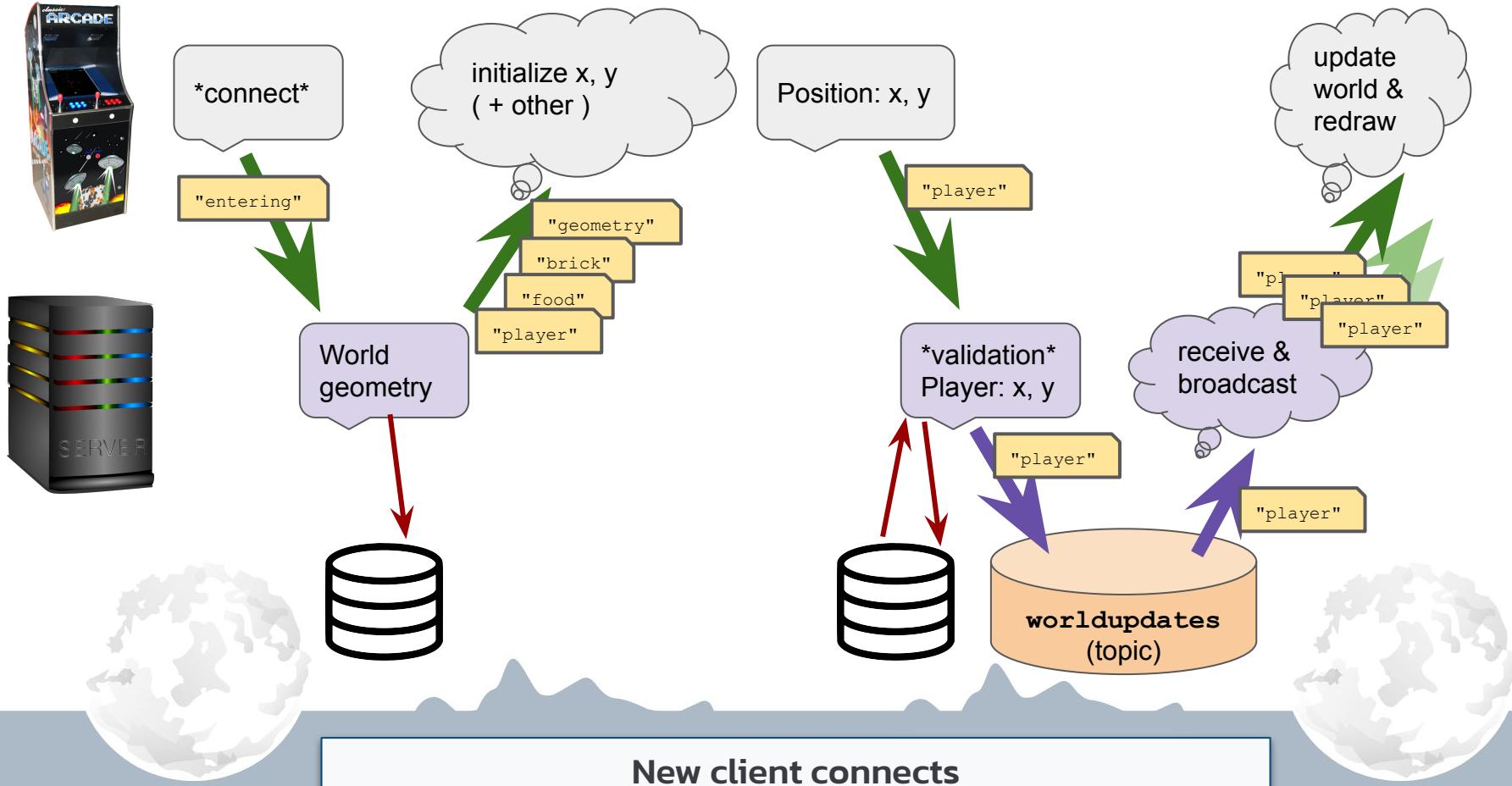
"world": picks up from Streaming and routes to ALL clients

## 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



*time*



*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

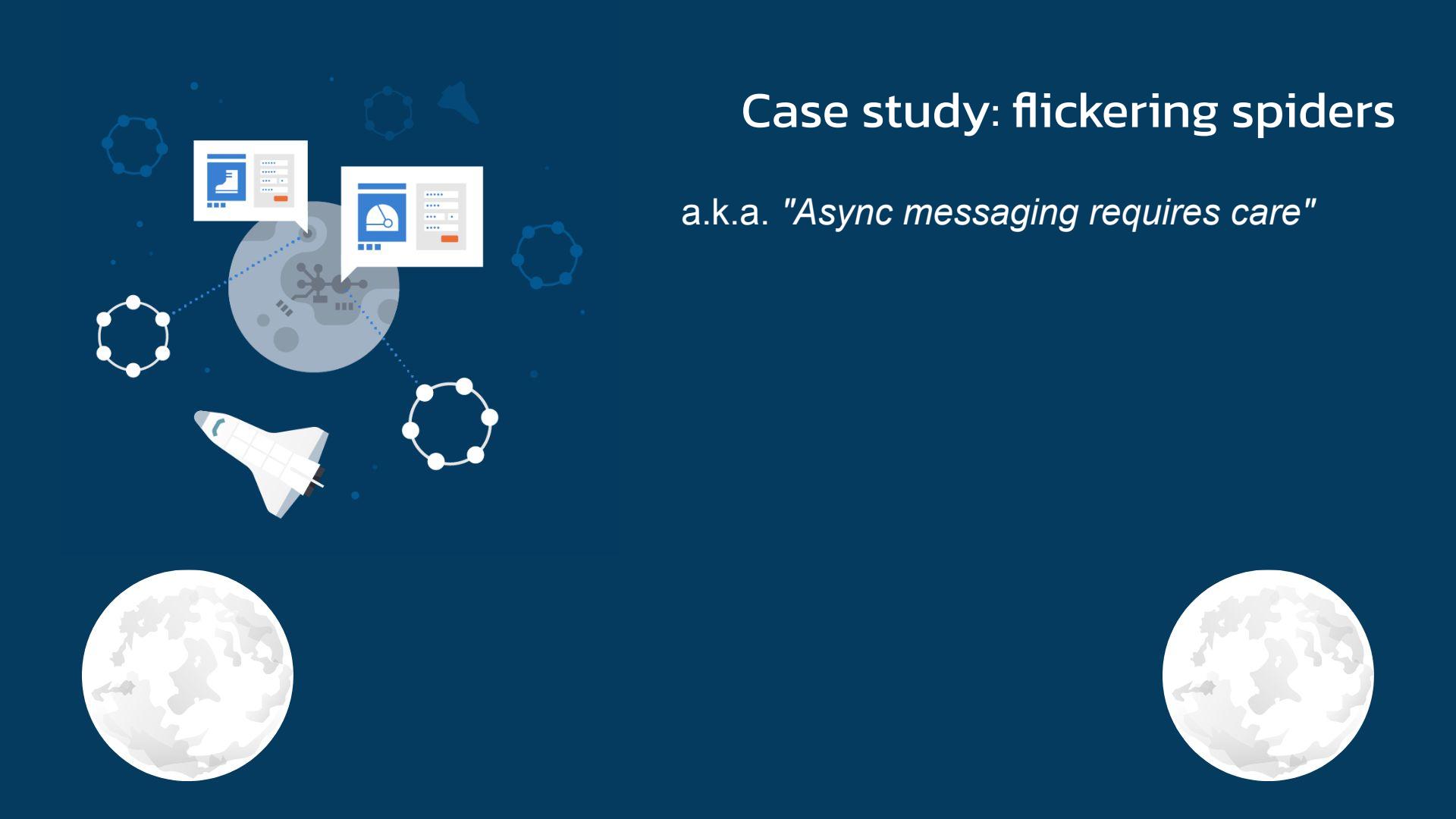
"player"

"player"

"player"

worldupdates  
(topic)

Gameplay

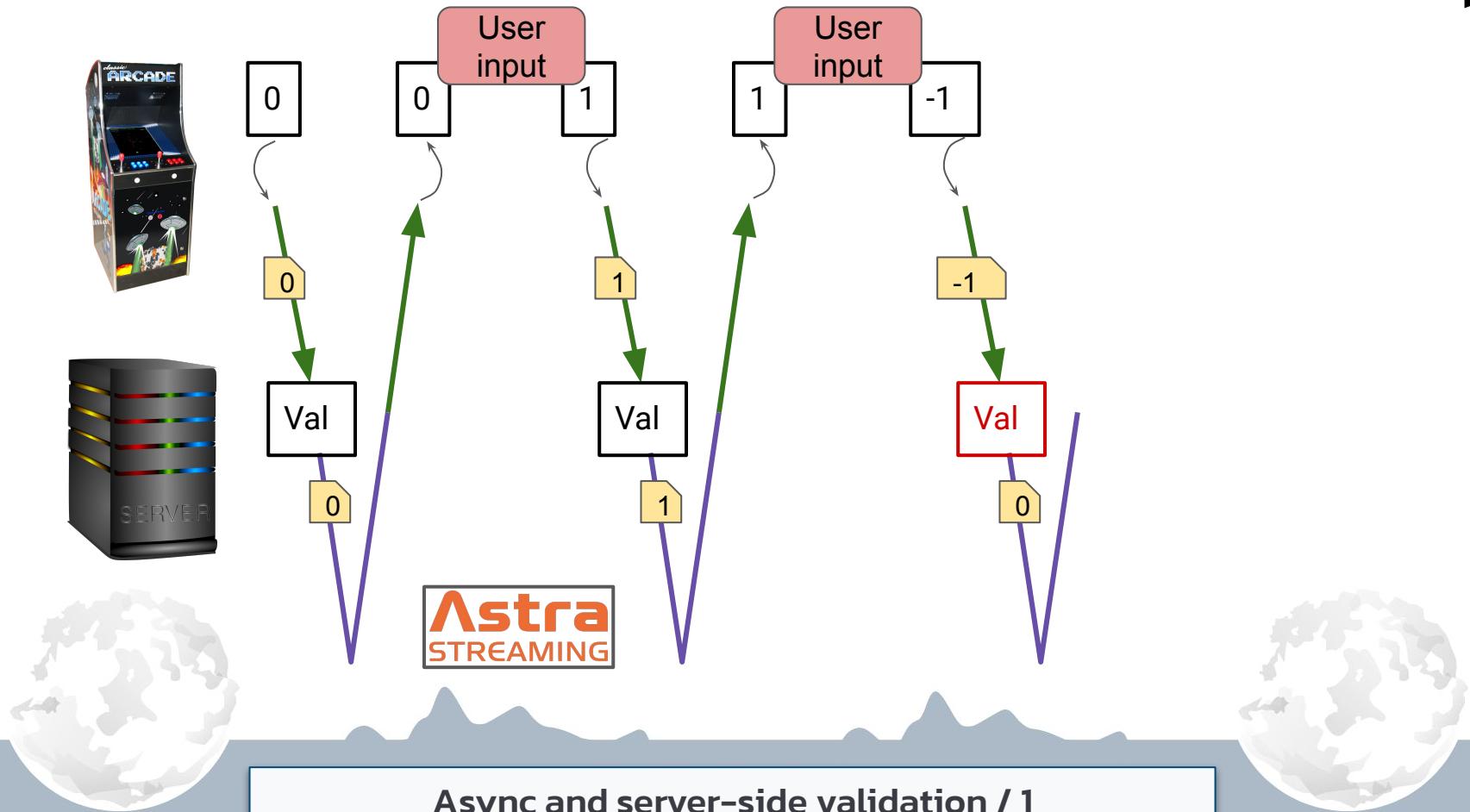


# Case study: flickering spiders

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

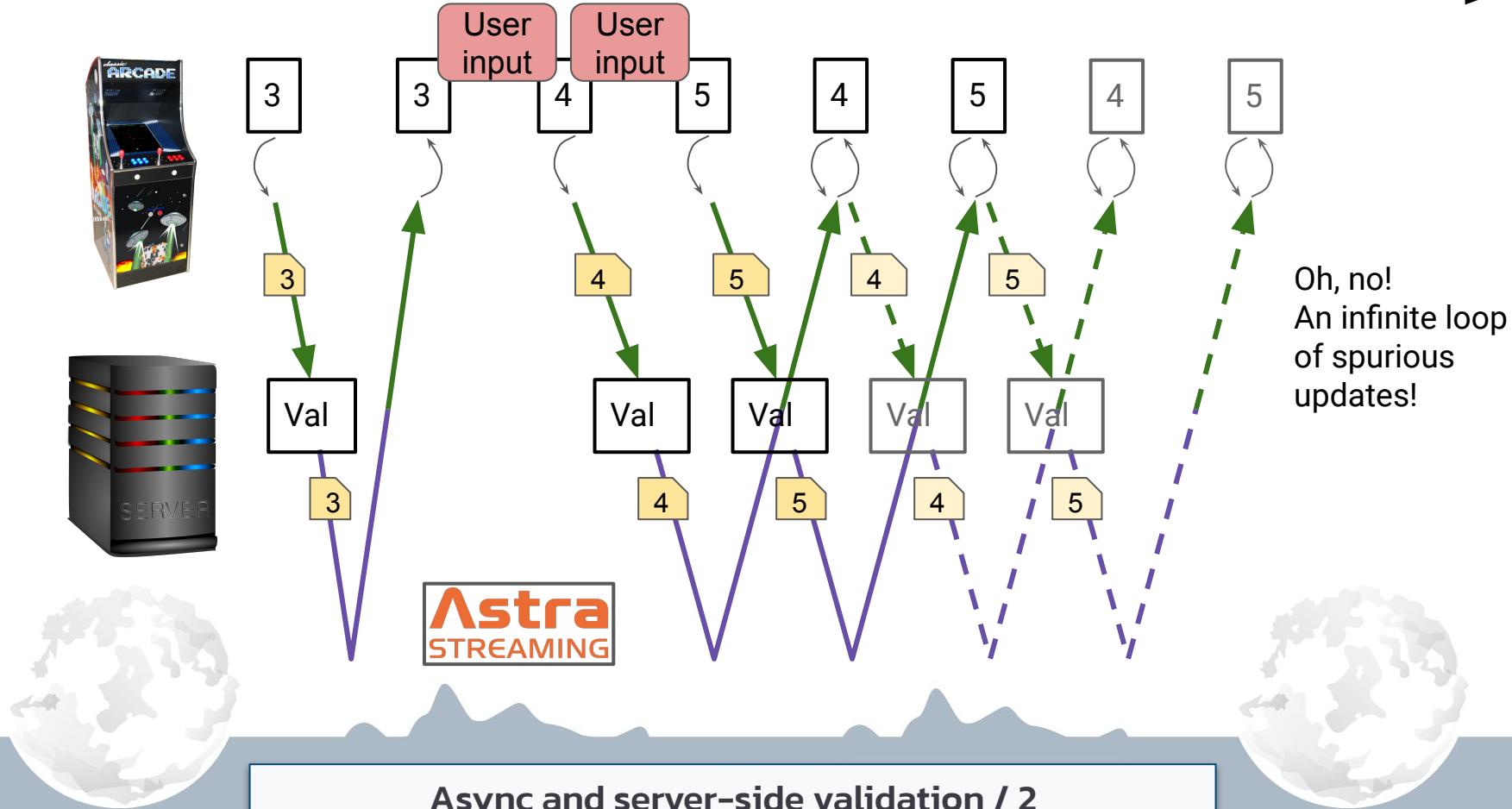
# Position is in client memory and validated by the API

*time* →



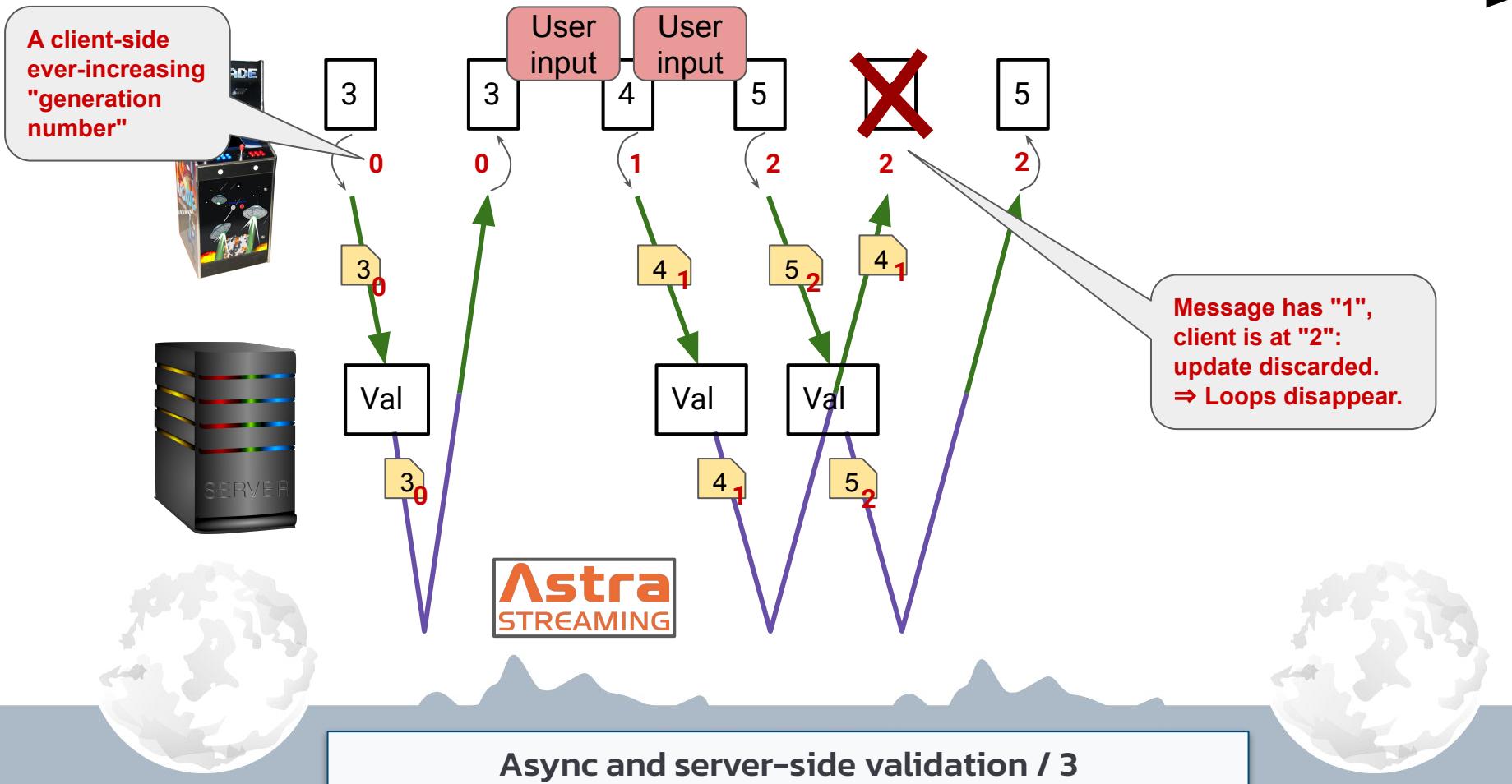
Client receives position updates (and may fire new updates)

*time* →



# Solution: discard obsolete server updates

time →





## 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



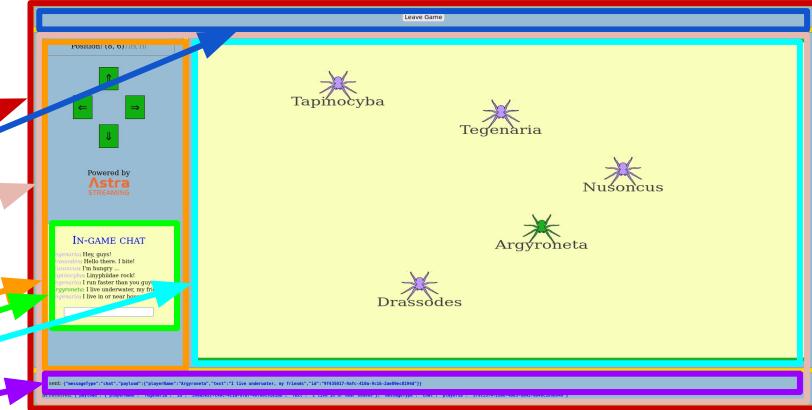
# Additional topics



```

1 let pws = null
2 let wws = null
3
4 const App = () => {
5   React_States = useState(...)
6   React_References = useRef(...)
7
8   keyDown_Handler = ...
9
10  useEffect( () => {
11    if(inGame) {
12      if ( both_Websockets === null ){
13        both_Websockets = new WebSocket(apiAddress)
14        wws.onmessage = evt => {
15          parse_and_interpret(evt ...)
16          // check messageType (player/geometry/chat)
17          // HERE WE MODIFY REACT STATE AND NEED REFERENCES
18        }
19      } else {
20        pws.send(goodbyeMessage)
21        both_Websockets.disconnect()
22        both_Websockets = null
23      }
24    }, [inGame])
25
26    useEffect( () => {
27      pws.send(myPlayerUpdate ...)
28    }, [coordinates, etc])
29
30    return (
31      App
32      PlayerForm
33      GameArea // if inGame
34      sidebar
35      GameInputs
36      buttons, etc
37      ChatArea
38      GameField
39      statusbar
40    )
41  }

```



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](https://fastapi.tiangolo.com))



FastAPI



# Routes = functions, return value = response (etc etc...)

## What about WebSockets?

```
app = FastAPI()

@app.websocket('/ws/world/{client_id}')
async def worldWSRoute(worldWS: WebSocket, client_id: str):
    await worldWS.accept()
    #
    pulsarClient = getPulsarClient()
    pulsarConsumer = getConsumer(client_id, pulsarClient)
    #
    try:
        geomUpdate = makeGeometryUpdate(HALF_SIZE_X, HALF_SIZE_Y)
        await worldWS.send_text(json.dumps(geomUpdate))

        while True:
            worldUpdateMsg = receiveOrNone(pulsarConsumer, RECEIVE_TIMEOUT_MS)
            if worldUpdateMsg is not None:
                await worldWS.send_text(worldUpdateMsg.data().decode())
                pulsarConsumer.acknowledge(worldUpdateMsg)
            await asyncio.sleep(SLEEP_BETWEEN_READS_MS / 1000)

    except WebSocketDisconnect:
        pulsarConsumer.close()
```

WebSocket decorator

function definition

once-per-client part

the function runs for a long time ...

notify successful processing

detect client-side disconnect

FastAPI, code layout

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

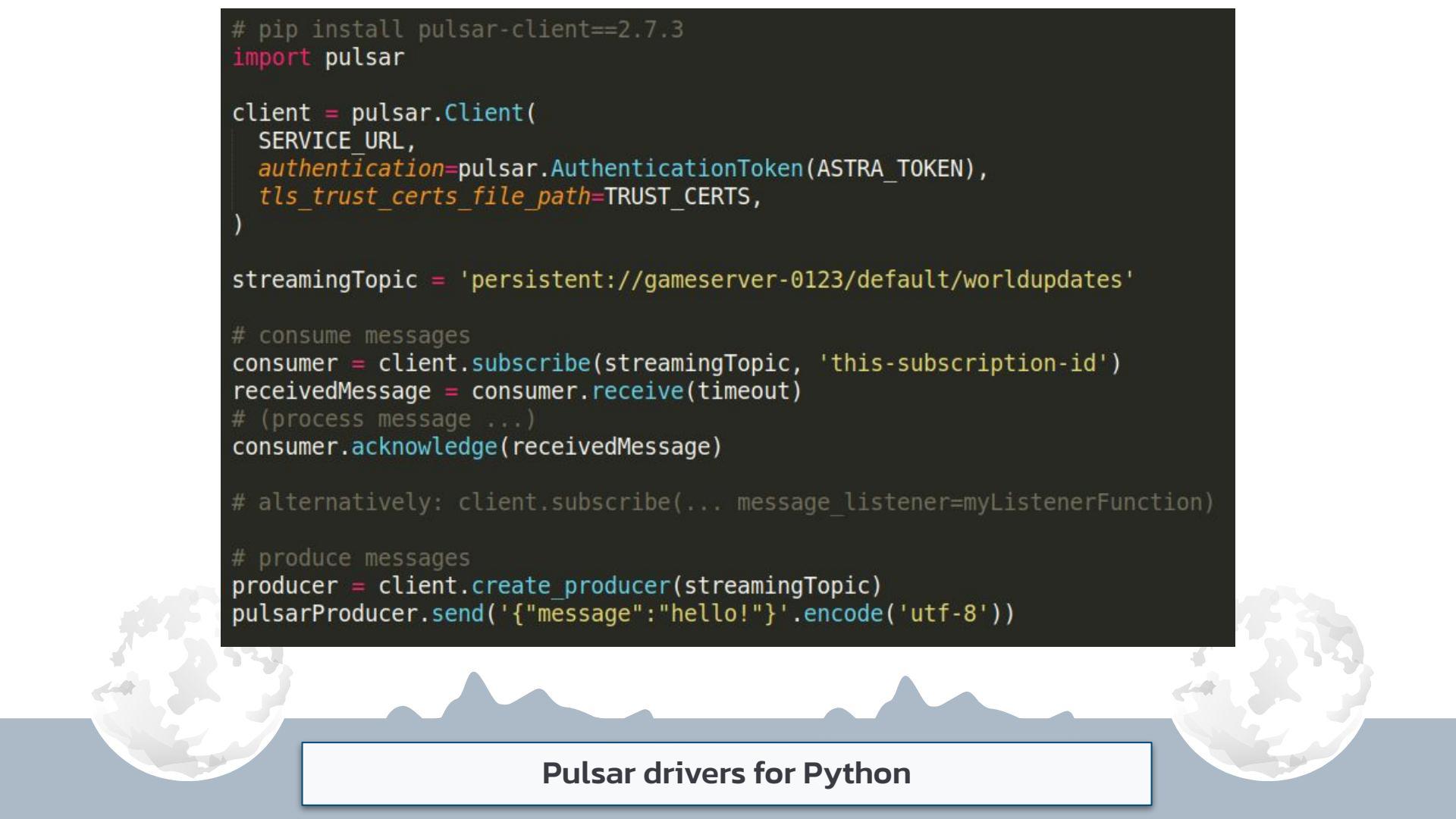
client = pulsar.Client(
    SERVICE_URL,
    authentication=pulsar.AuthenticationToken(ASTRA_TOKEN),
    tls_trust_certs_file_path=TRUST_CERTS,
)

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

DataStax  
**Astra**

Current Organization [REDACTED]

Dashboard Databases Create Database hemidactylus techblog measurements workshops multics Show all databases Streaming BETA Create Streaming stefano og1 snktest

Sample App Gallery Looking to get up and running? Get started with a sample app!

Other Resources Documentation

## Dashboard / snktest

Quickstart Try Me Connect Topics Functions Sinks Sources Namespaces Settings

You can connect to your tenant in a few different ways  
Use the Pulsar CLI or language appropriate client to connect with Astra Streaming.

Connect

Java Python Golang Node.js WebSocket JMS

**Python Code Samples**

Astra Streaming is powered by [Apache Pulsar](#). To connect to your service, use the open-source client APIs provided by the Apache Pulsar project.

The Python client APIs are distributed through [Python Package Index \(PyPi\)](#). On Linux, the Python versions 3.4 to 3.7 are supported. On MacOS version 3.7 is supported. You can find the documentation for the Python client [here](#).

Astra Streaming is running Pulsar version 2.7.2. You should use this API version or higher.

PIP PRODUCER CONSUMER READER

**Simple Consumer**

This sample creates a consumer on a topic, waits for a message, acknowledges it, then closes the consumer.

| Cluster                | Namespace | Topic     | Subscription    |
|------------------------|-----------|-----------|-----------------|
| pulsar-gcp-europewest1 | default   | jsnktopic | my-subscription |

```
import pulsar, time

service_url = 'pulsar+ssl://pulsar-gcp-europewest1.streaming.datastax.com:6651'

# Use default CA certs for your environment
# RHEL/CentOS:
trust_certs= '/etc/ssl/certs/ca-bundle.crt'
# Debian/Ubuntu:
# trust_certs= '/etc/ssl/certs/ca-certificates.crt'

token=
```

## Python quickstart on Astra Streaming UI

```
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(worldConsumer, SLEEP_BETWEEN_UPDATES_MS)  
        if worldUpdateMsg is not None:  
            # We forward any update from this consumer to all clients  
            # for all clients out there:  
            await worldWS.send_text(worldUpdateMsg)  
        await asyncio.sleep(SLEEP_BETWEEN_UPDATES_MS)  
except WebSocketDisconnect:
```

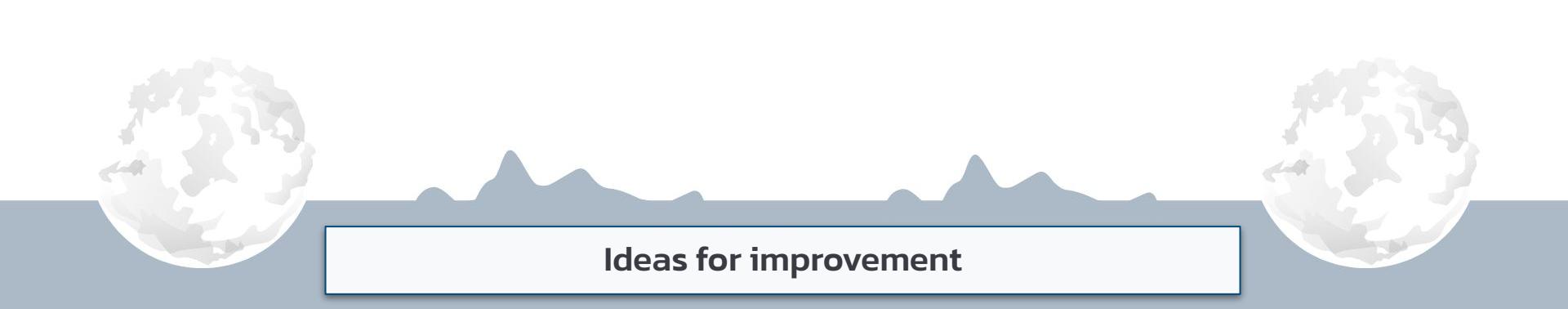
```
def receiveOrNone(consumer, timeout):  
    """  
    A modified 'receive' function for a Pulsar topic  
    that handles timeouts so that when the topic is empty  
    it simply returns None.  
    """  
  
    try:  
        msg = consumer.receive(timeout)  
        return msg  
    except Exception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

```
const GameField = ({playerMap, playerID, boardWidth, boardHeight}) => {

  return (
    <svg className="game-field" width="100%" height="700" preserveAspectRatio="none"
      viewBox={`0 0 ${100 * boardWidth} ${100 * boardHeight}`}>
      <defs>
        <pattern id="lyco_other" x="50" y="50" width="100" height="100" patternUnits="userSpaceOnUse">
          <image x="0" y="0" width="100" height="100" href="lyco_other.svg"></image>
        </pattern>
        <pattern id="lyco_self" x="50" y="50" width="100" height="100" patternUnits="userSpaceOnUse">
          <image x="0" y="0" width="100" height="100" href="lyco_self.svg"></image>
        </pattern>
        <pattern id="hearts" x="65" y="40" width="100" height="100" patternUnits="userSpaceOnUse">
          <image x="20" y="10" width="60" height="60" href="hearts.svg"></image>
        </pattern>
      </defs>
      <rect width={100 * boardWidth} height={100 * boardHeight} style={{fill: '#f9ffbb'}} />
      { Object.entries(playerMap).map( ([thatPlayerID, thatPlayerInfo]) => {
        const patternName = thatPlayerID === playerID ? 'lyco_self' : 'lyco_other'
        const playerClassName = thatPlayerID === playerID ? 'player-self' : 'player-other'
        return (<g key={thatPlayerID} transform={`translate(${thatPlayerInfo.x * 100}, ${thatPlayerInfo.y * 100})`}>
          <g transform='translate(50,50)'>
            <rect x="-50" y="-50" height="100" width="100" fill={`${url(${patternName})}`}></rect>
            {thatPlayerInfo.h && <rect x="-50" y="-50" width="100" height="100" fill={url(#hearts)}></rect>}
            <g transform={`translate(0,${thatPlayerInfo.y + 1 >= boardHeight? -70 : 70})`}>
              <text className="player-name" textAnchor='middle' fontSize='40'>
                {thatPlayerInfo.playerName}
              </text>
            </g>
          </g>
        </g>
      ))}
    </svg>
  )
}
```

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

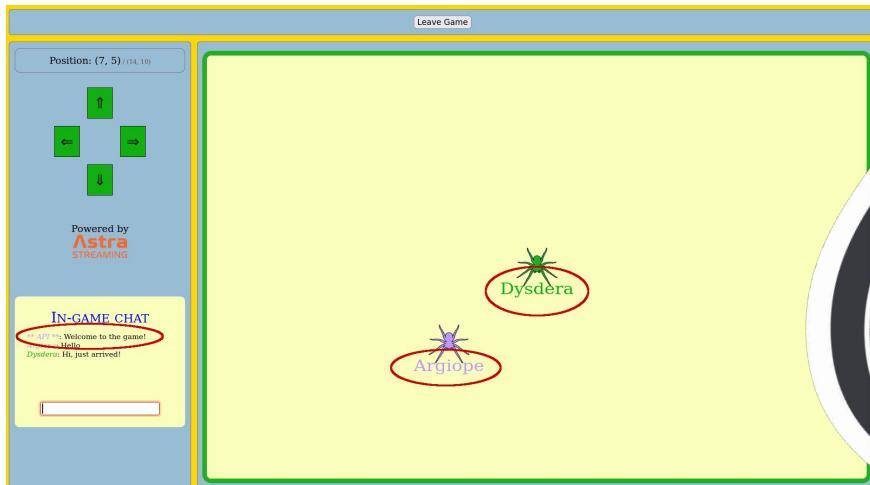
**What's next?**  
**Quiz & Homework**



Agenda



# Questions + coding exercise



Lab & Badge !



DataStax Developers

# workshop-chat

<https://www.youtube.com/watch?v=MuwT5xxFVVI> - Subscribe to mailing list: [http...](http://)

Rechercher

PRESENTER — 1  
David Jones-Gilardi

HELPER — 7  
012345  
AaronP  
B1nary  
Chelsea Navo  
Jeremy Hanna  
John Sanda  
Patrick\_McFadin

EN LIGNE — 560  
-samu-  
6304-42JB  
Aahlya  
Abdurahim  
abhi3pathi  
Abhiis.s  
Abhineet  
Abirsh

Événements  
moderator-only  
. WELCOME  
start-here  
code-of-conduct  
introductions  
upcoming-events  
useful-resources  
memes  
your-ideas  
@the-stage

WORKSHOPS  
# workshop-chat  
# workshop-feedback  
workshop-materials  
upcoming-workshops

ASTRA DB  
getting-started  
astra-apis  
astra-development  
sample-applications

APACHE CASSANDRA  
Cedrick Lun...

RIGGITYREKT Hier à 21:14  
I have a 5 node datacenter, 4 nodes are on dse version 5.1.20, one is on dse5.0.15. I am doing some mixed version testing for a class and the one node that is 5.0.15 is coming up as an analytics workload. I dont have /etc/default/dse, instead I am using /etc/init.d/dse-cassandra.  
how do I make that node start in cassandra workload, not in analytics?

RIGGITYREKT Hier à 23:39  
Okay I found out my issue, when i started DSE 5.0.15 it had endpointsnitch set to DseSimpleSnitch, the rest of my cluster is using PropertyFileSnitch, when i change it to PropertyFileSnitch, it still uses the simple snitch config. looking at the docs i see there is a way to go to GossipingPropertyFileSnitch, but i need the property file one. I can wipe this dbs, do anything with this node to get this done. how do i fix this?  
@here

Erick Ramirez Aujourd'hui à 02:19  
mixed versions isn't supported and you're guaranteed to run into weird issues that will cause further problems down the track

Cedrick Lunen Aujourd'hui à 09:01  
When you start a node you have parameters -k for analytics, -g for graph and -s for search. To remove analytics check and remove -k

Envoyer un message dans #workshop-chat

# !discord

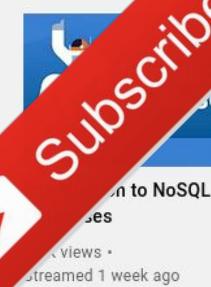
[dtsx.io/discord](https://dtsx.io/discord)



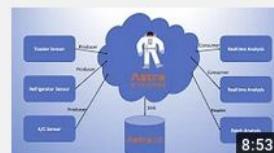
DataStax Developers Discord (18k+)



# Subscribe



# Subscribe



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!

