RSOCKET

THE FUTURE PROTOCOL

@OlehDokuka

Oleh Dokuka

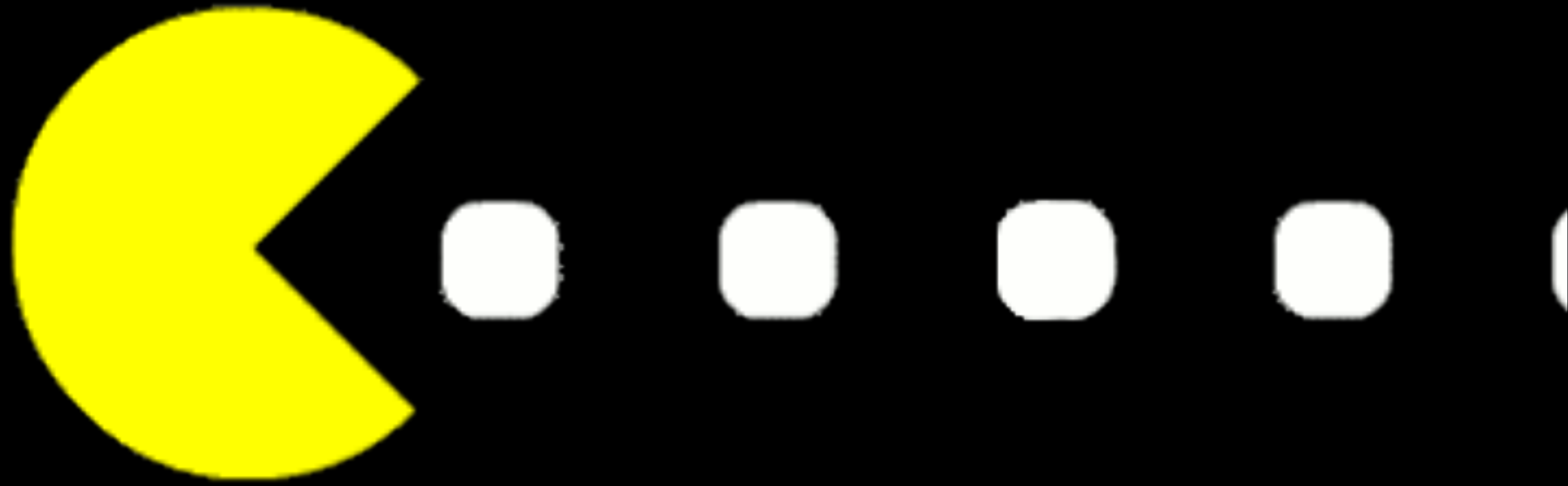
- WORK FOR NETIFI
- REACTIVE GEEK
- REACTOR 3 CONTRIBUTOR
- BOOKS AUTHOR
- LOCAL COMMUNITY BUILDER

🐦 **@OlehDokuka**

MULTIPLAYER PAC-MAN

@netifi_inc

@OlehDokuka



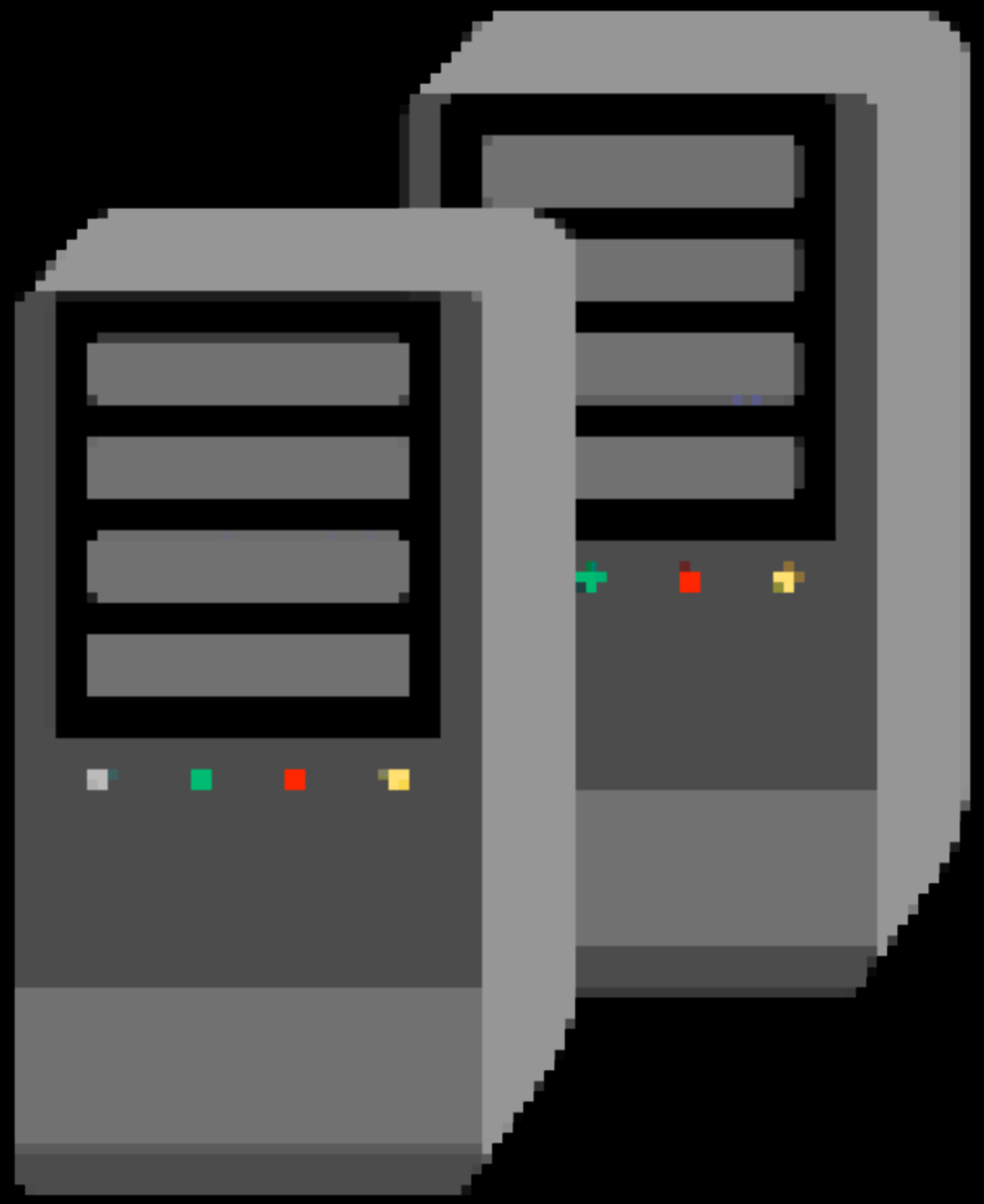
Agenda

- DEFINE PROBLEM
- COMPARE PROTOCOLS
- HAVE FUN
- DEFINE THE BEST PROTOCOL

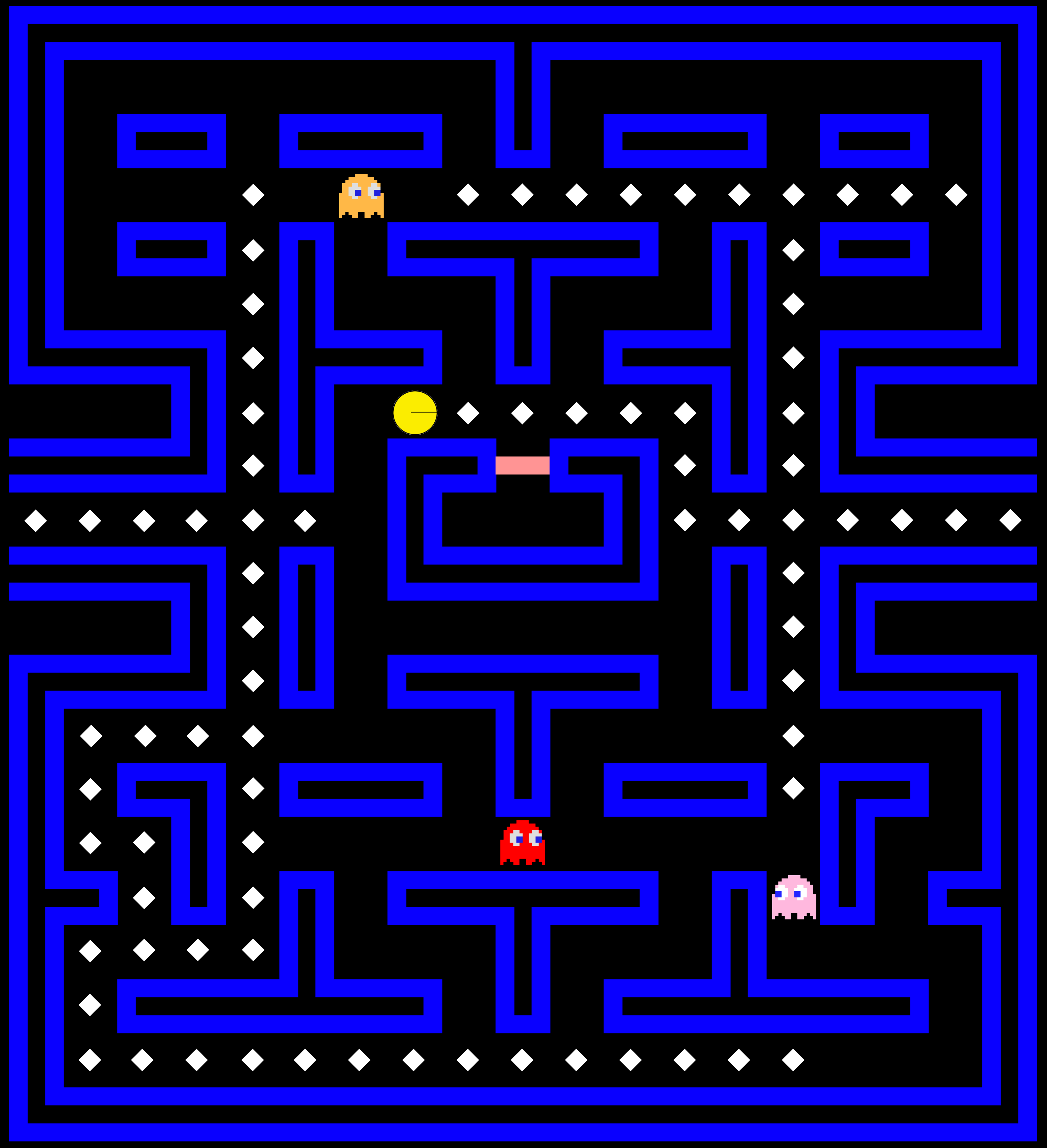
MULTIPLAYER

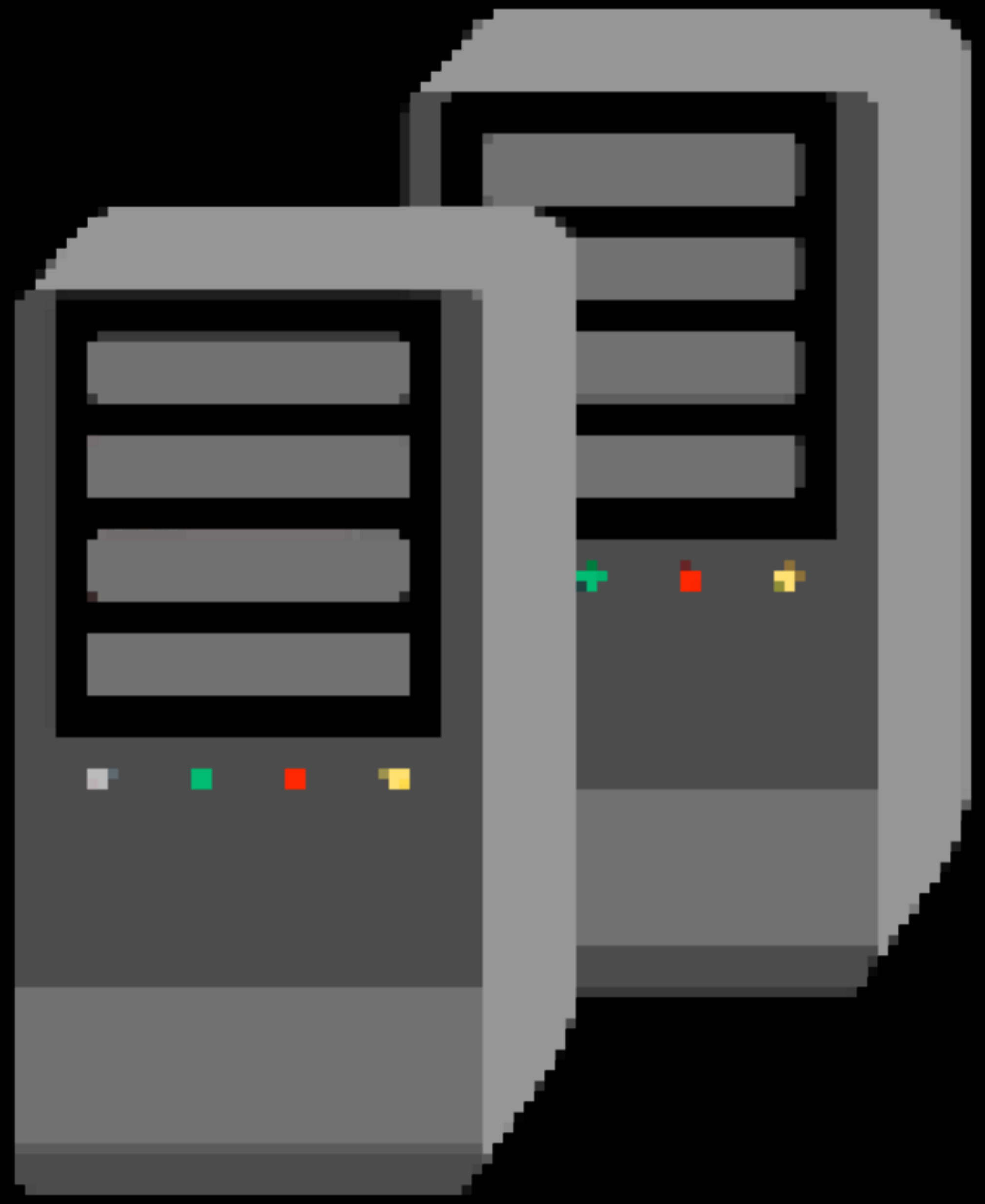
REQUIREMENTS

Step 0: Load

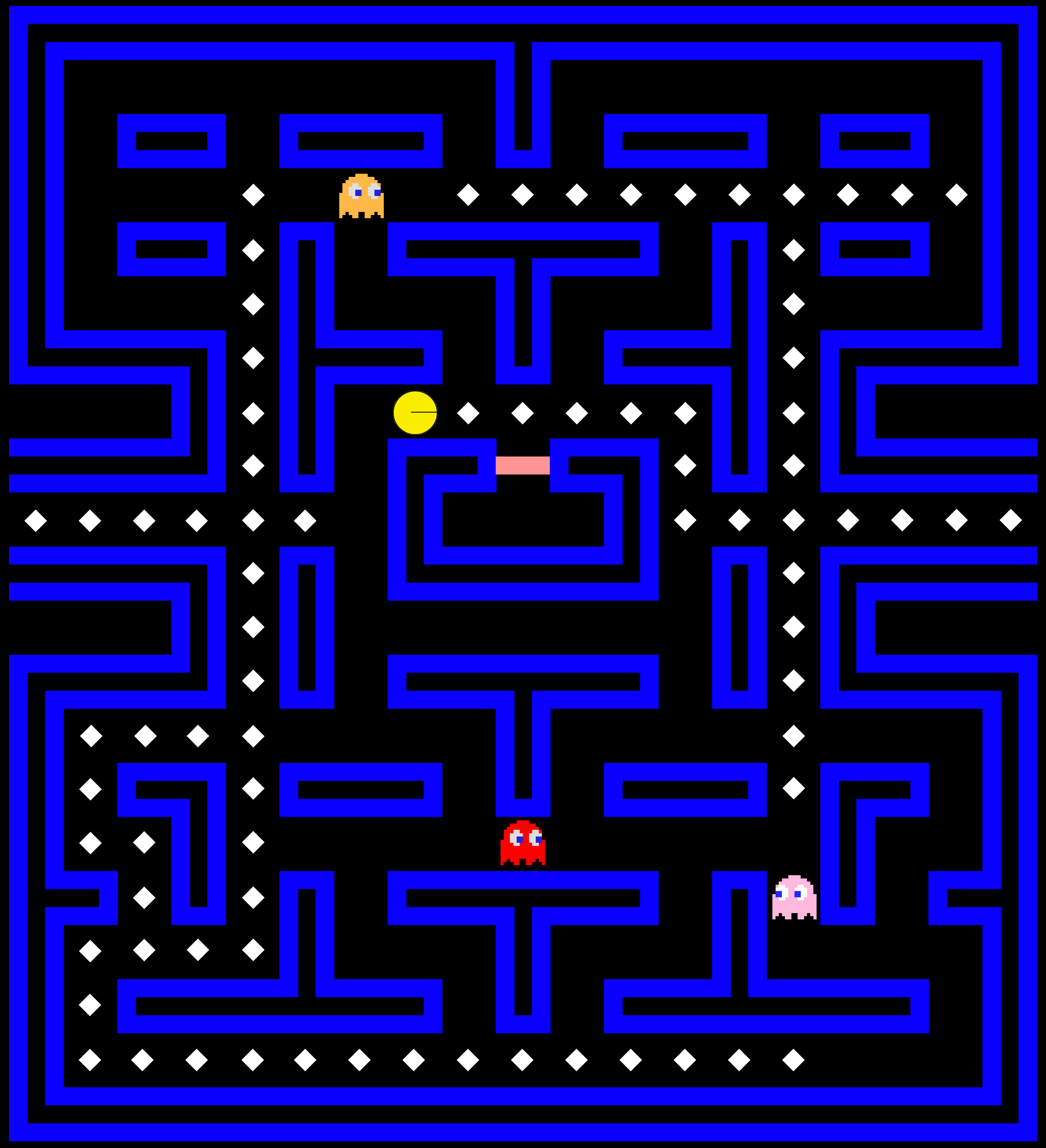


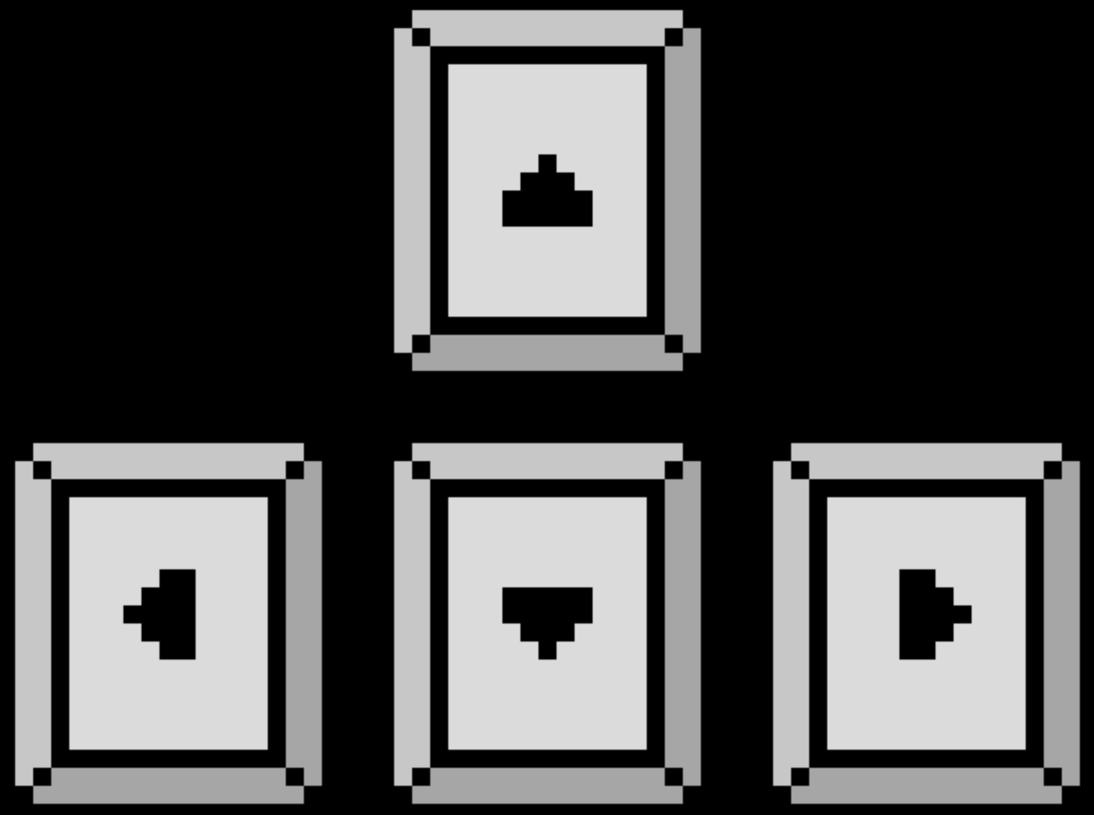
Step 1: Setup

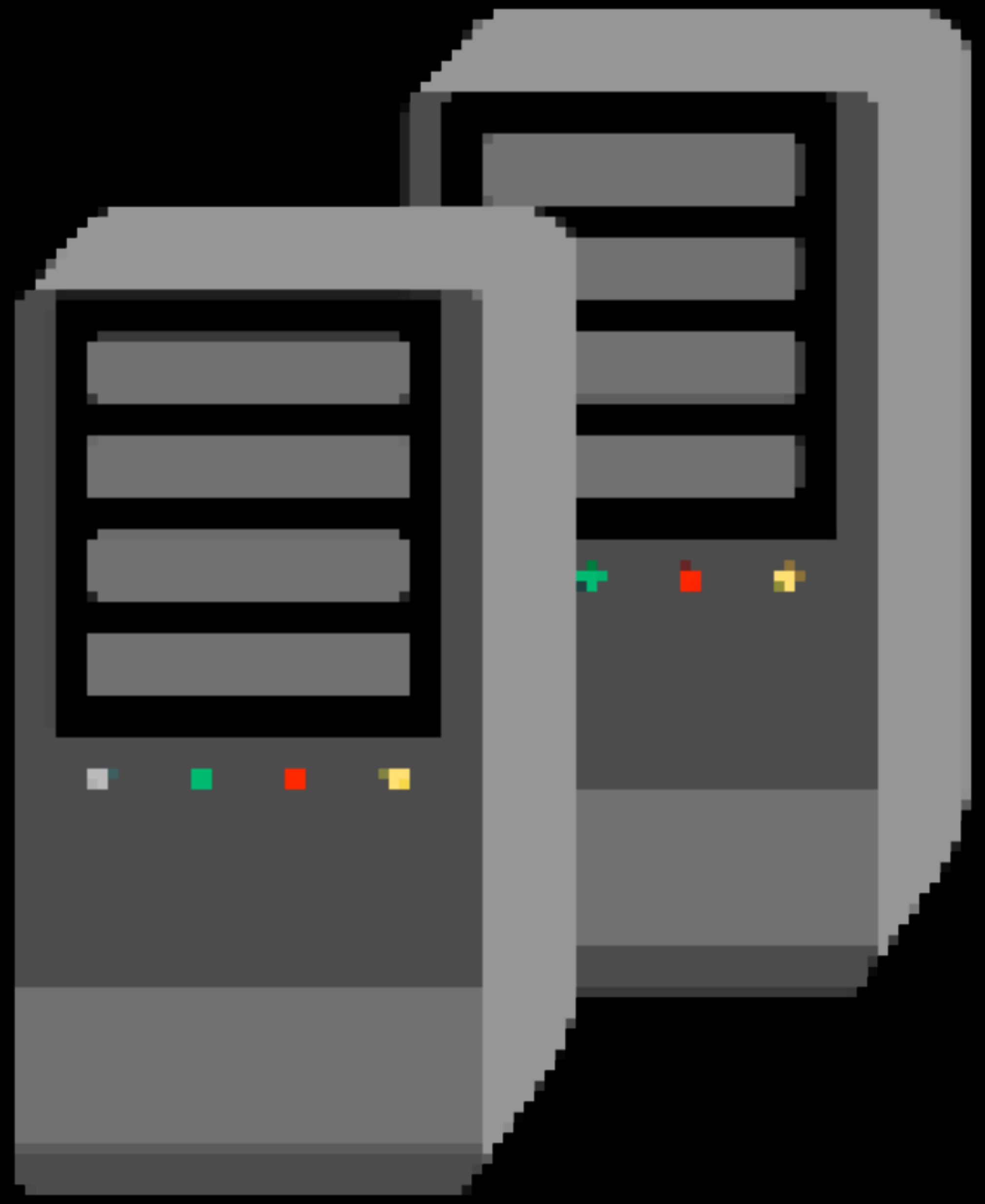




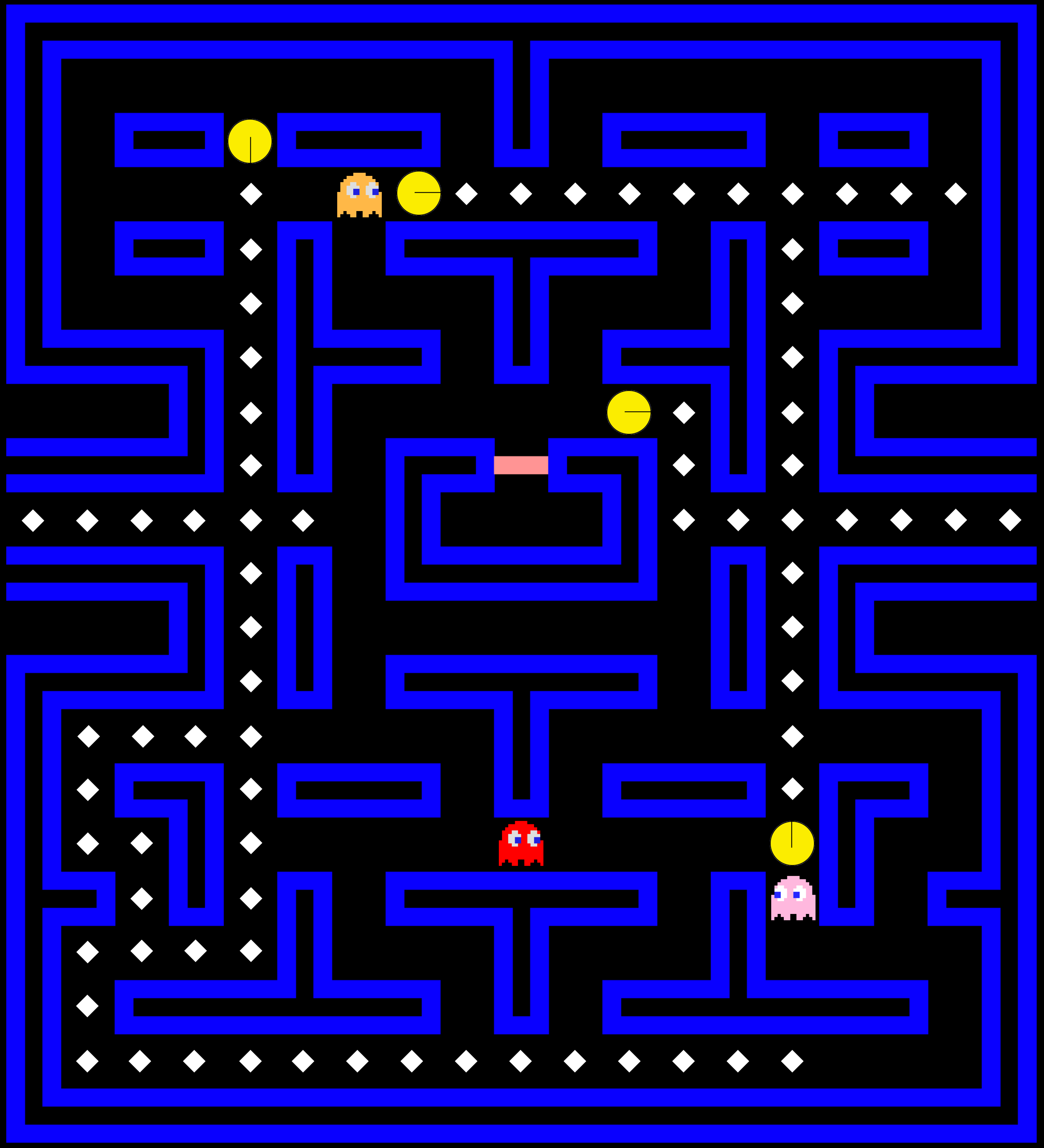
Step 2:Location

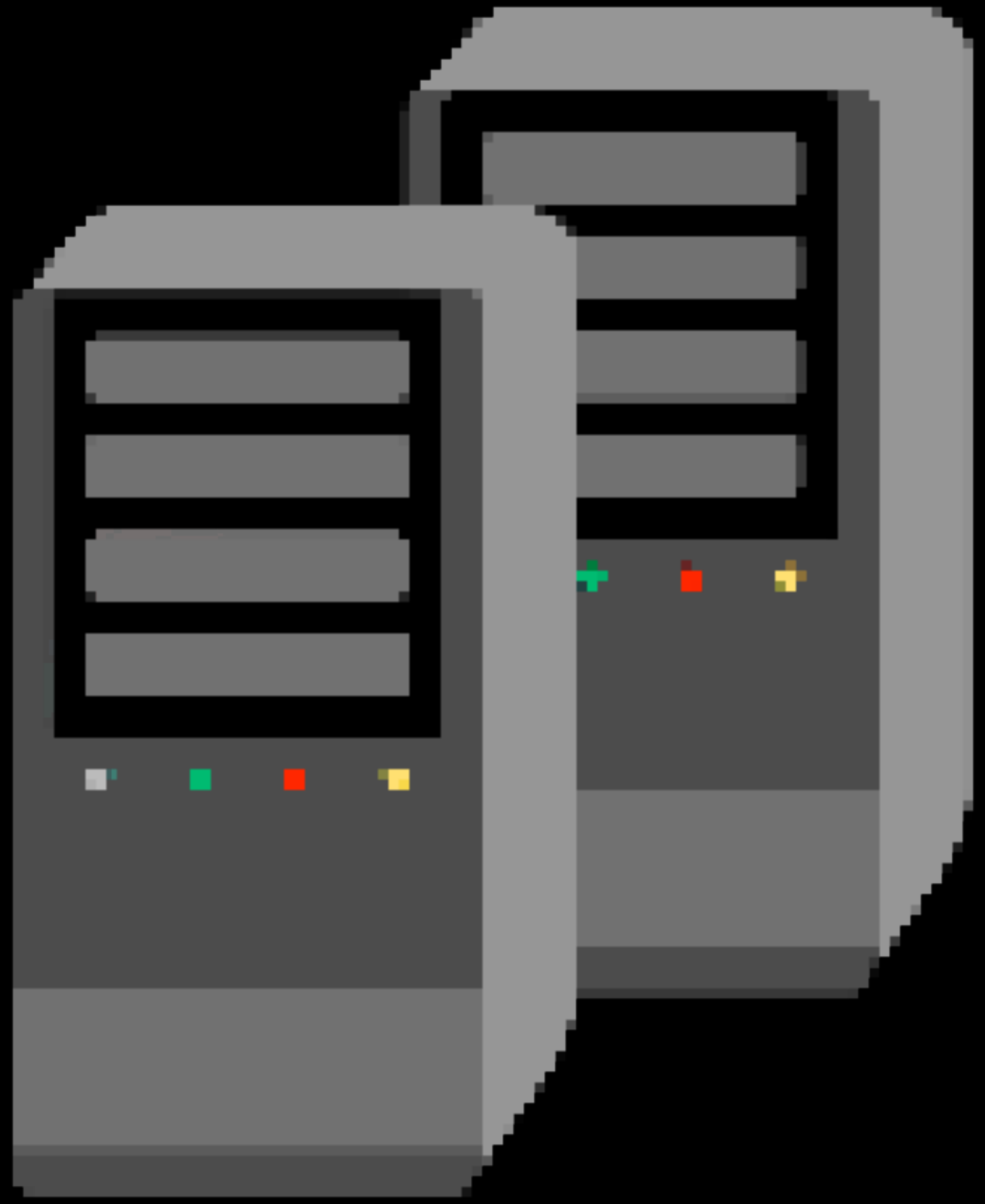




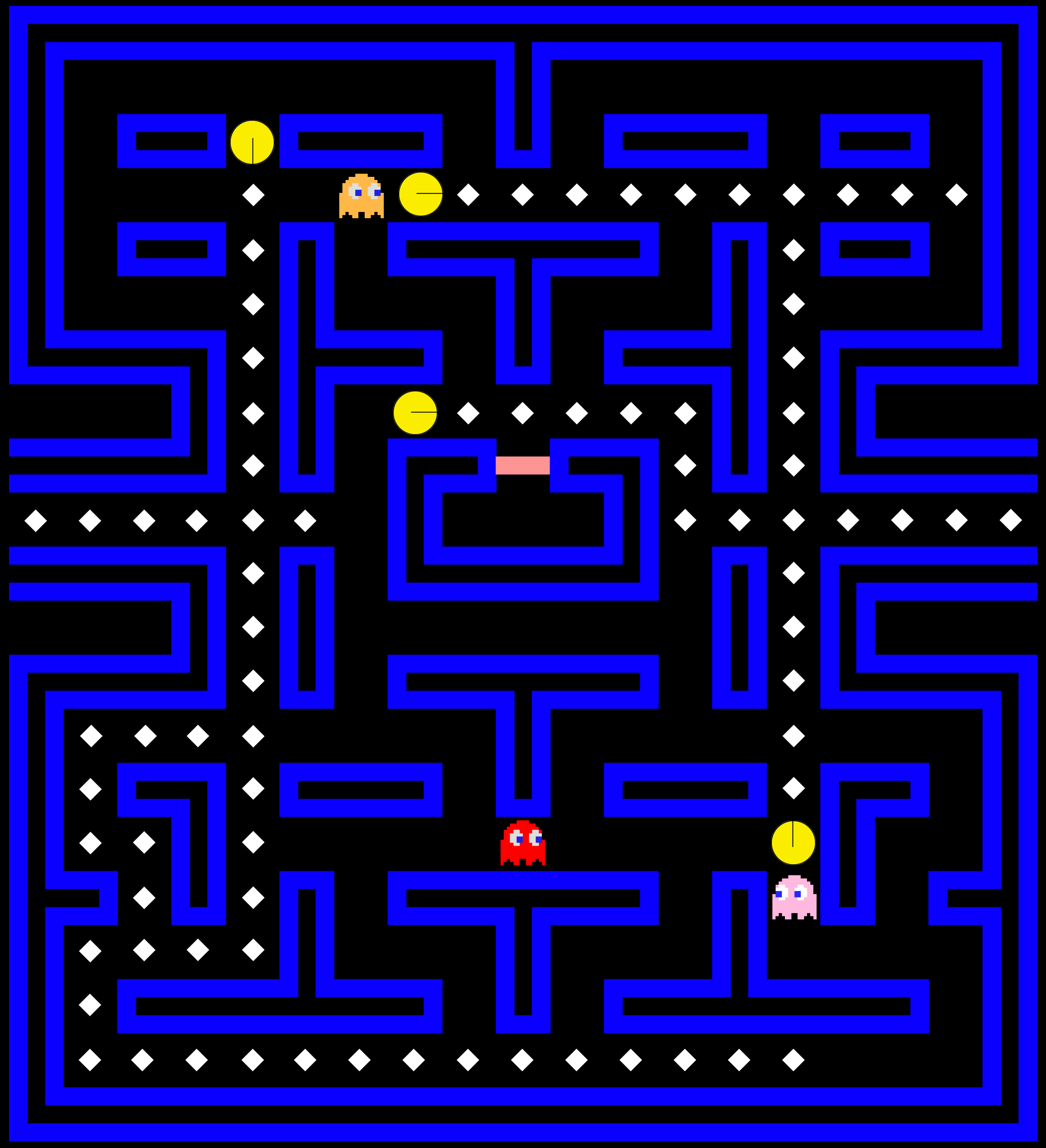


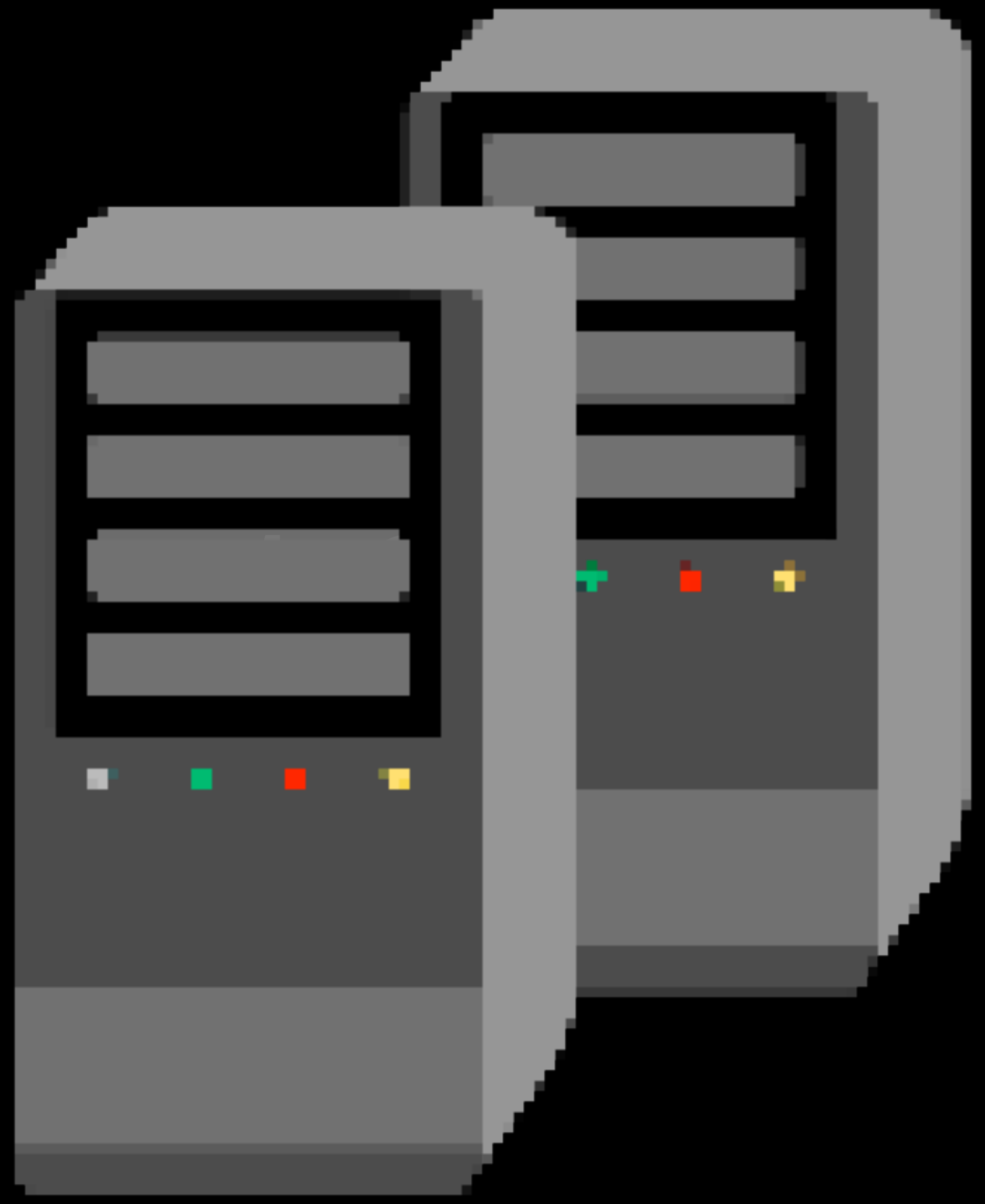
Step 3: Updates





Step 3: Updates





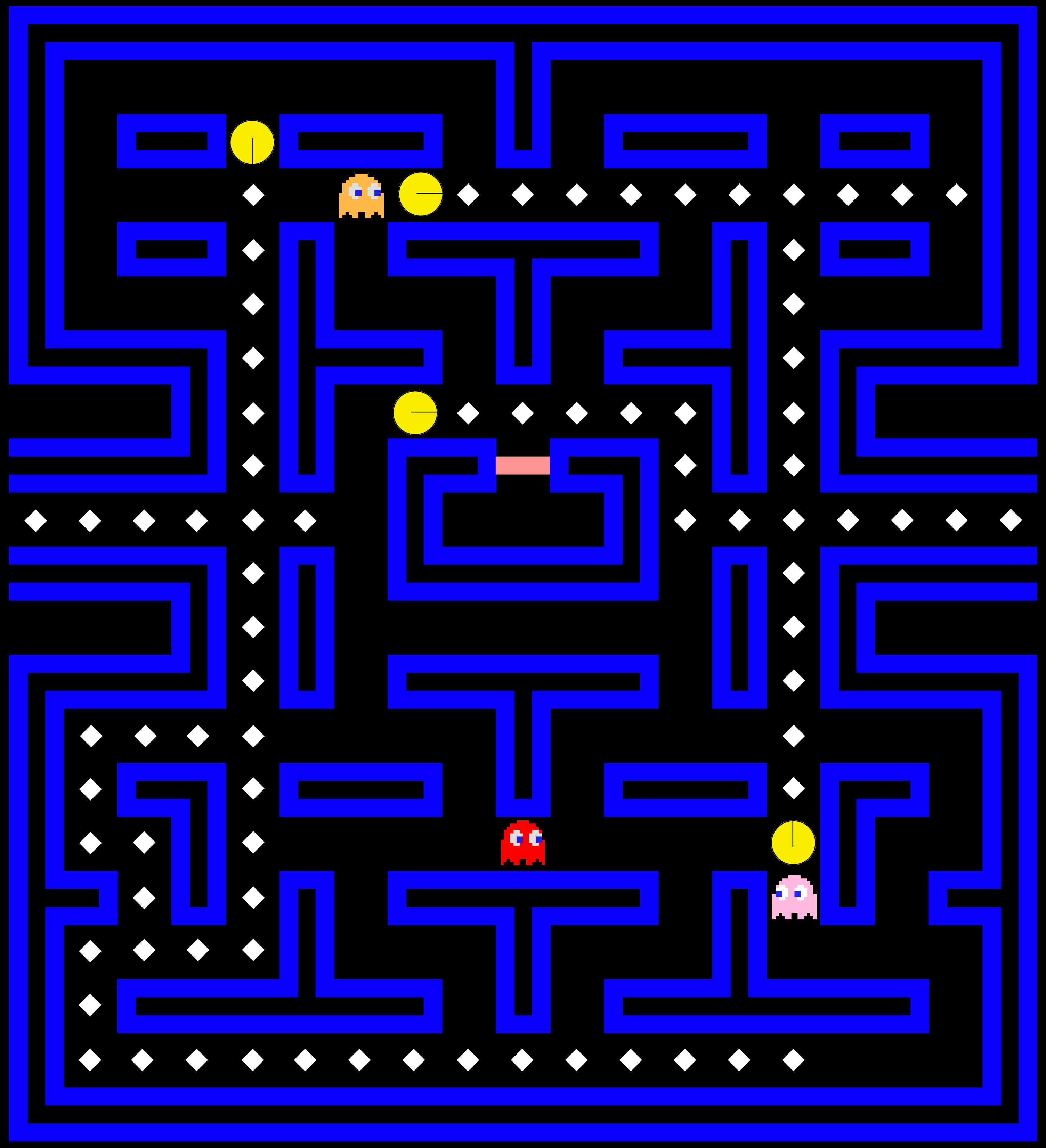
Scoreboard

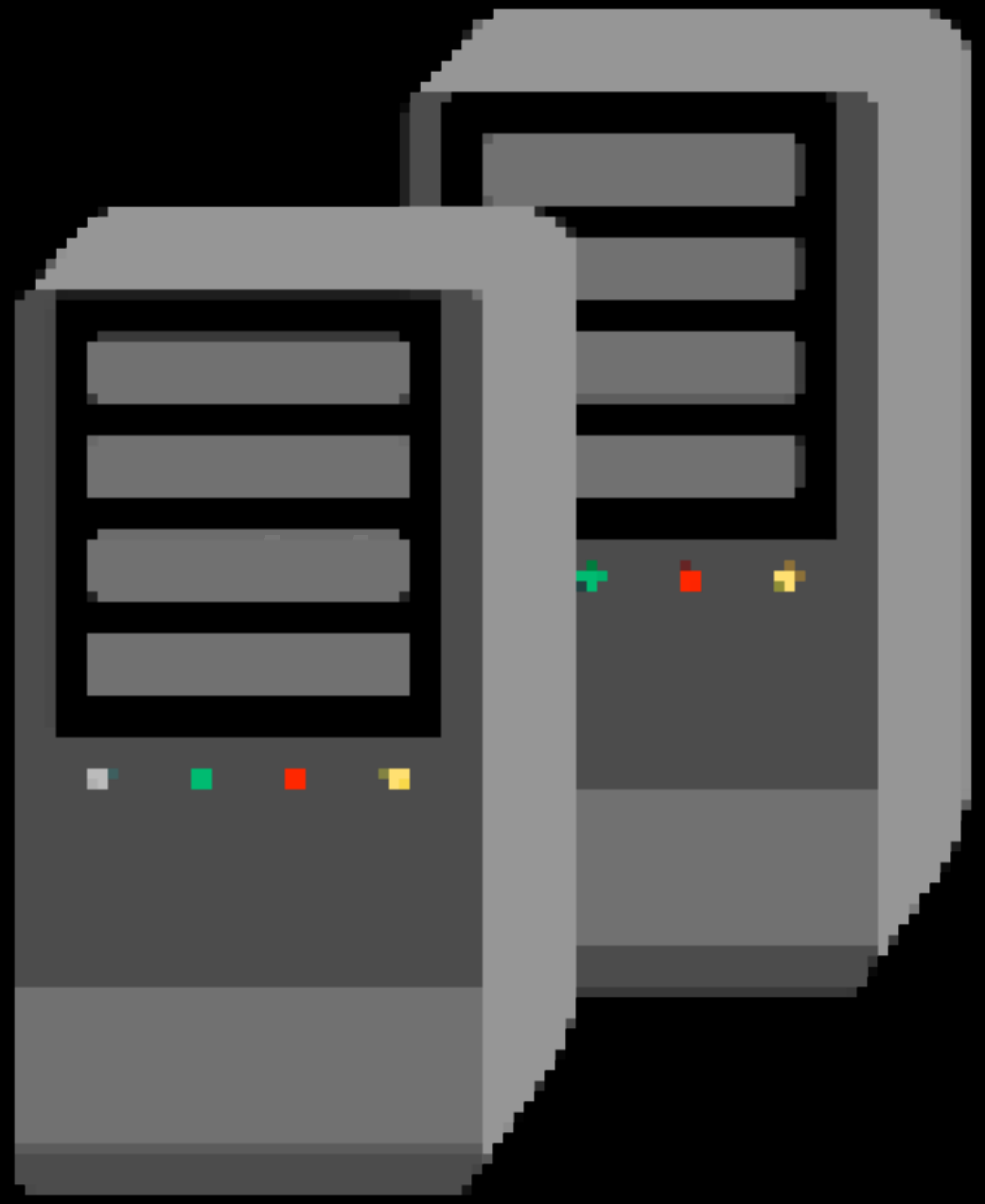
100.	Player 1
88.	Player 3
80.	Player 5
75.	Player 2
68.	Player 6
30.	Player 4
7.	Player 7

ATTENTION!

This is still about microservices

Real Enterprise

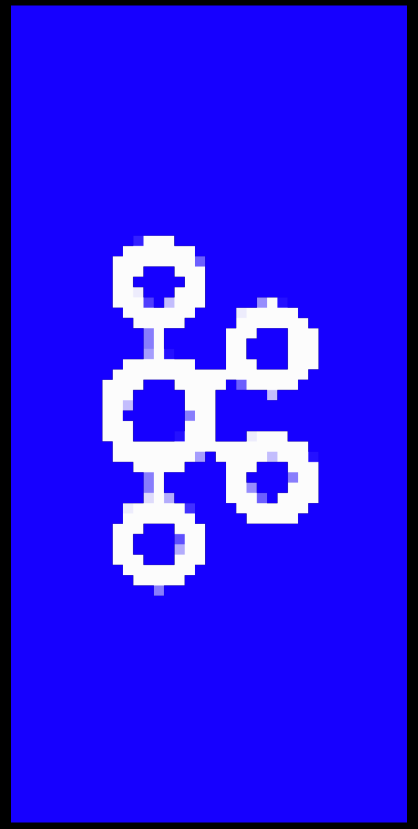


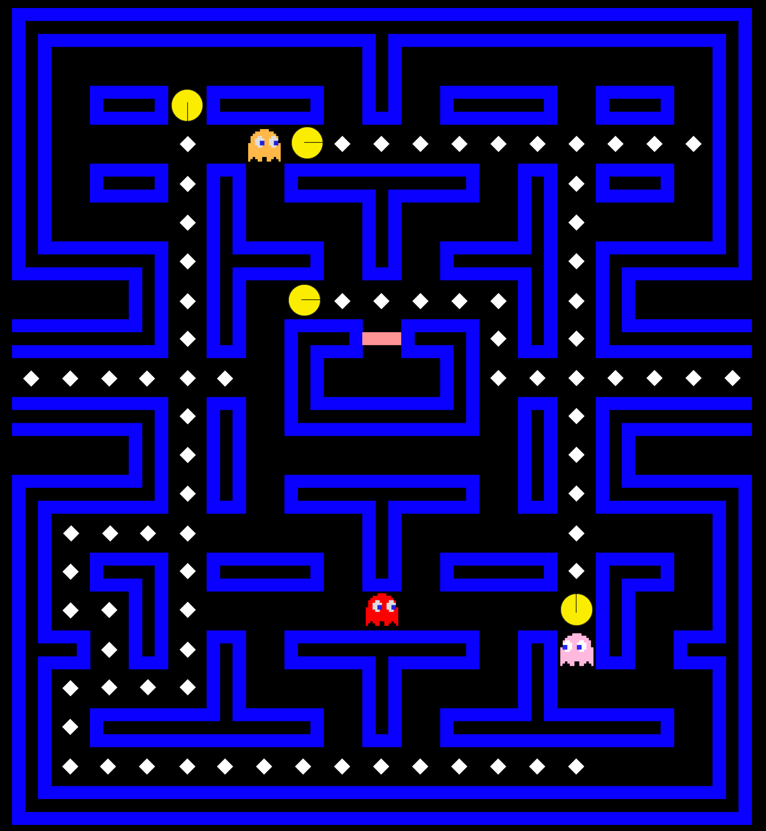


Scoreboard

100.	Player 1
88.	Player 3
80.	Player 5
75.	Player 2
68.	Player 6
30.	Player 4
7.	Player 7

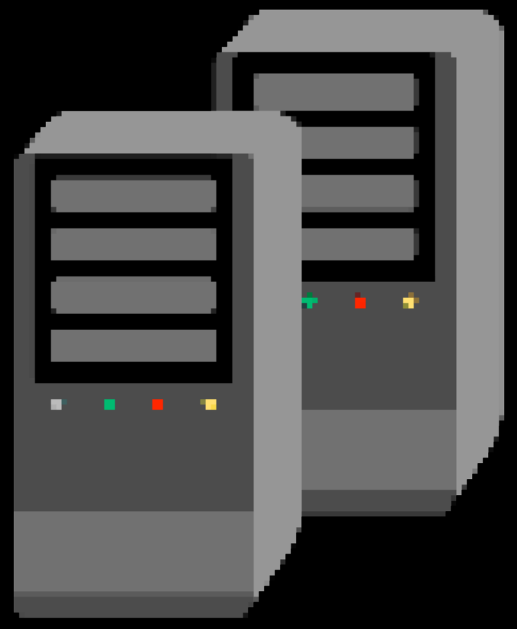
Real Enterprise





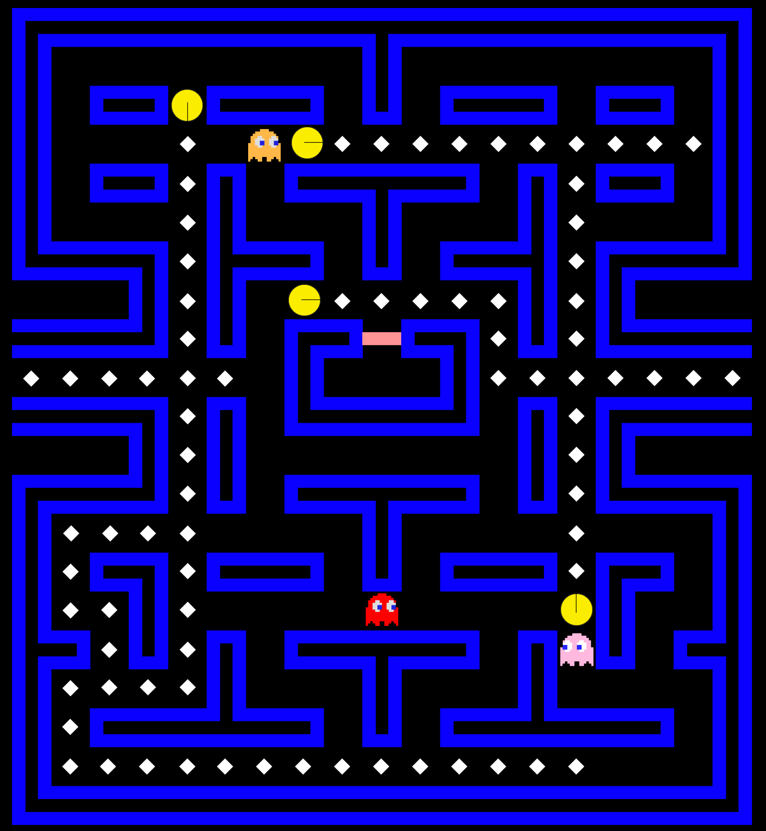
Scoreboard

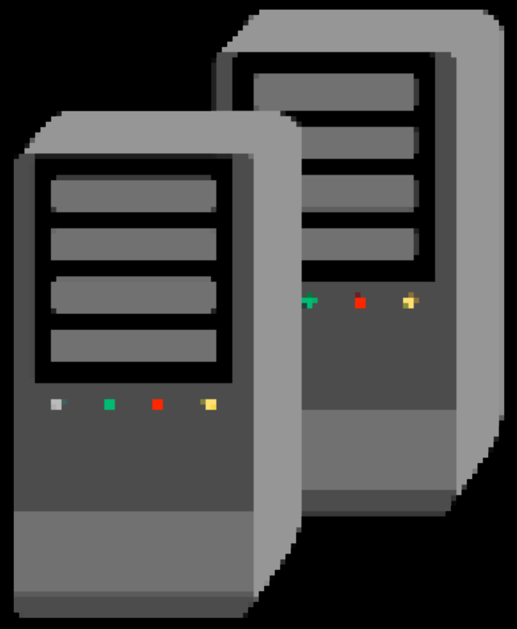
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7



Real Enterprise

← Elastic Storag

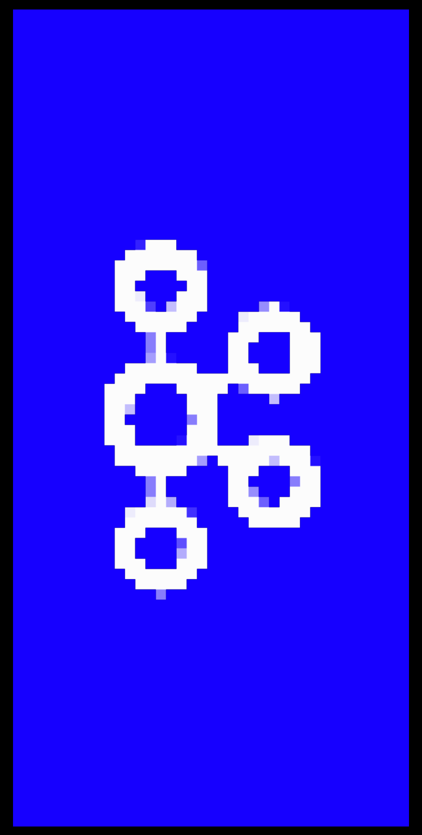


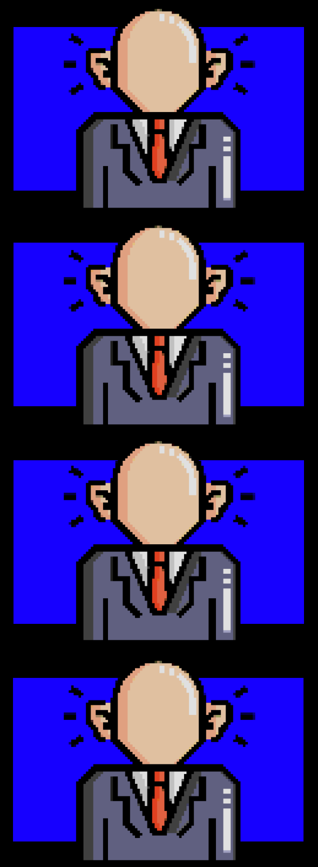


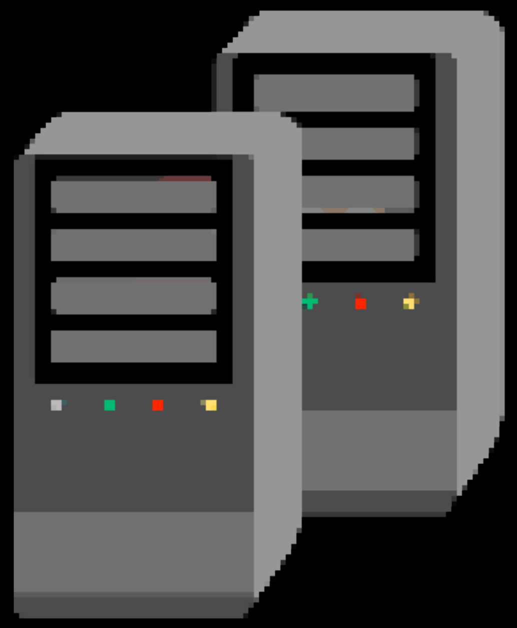
Scoreboard

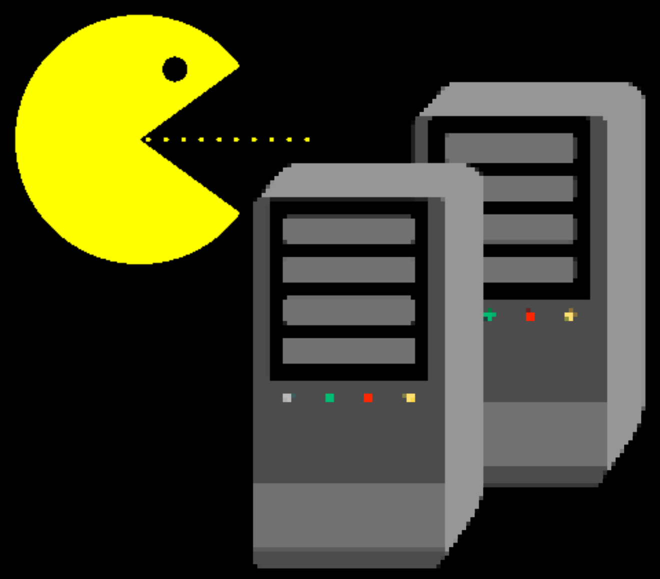
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7

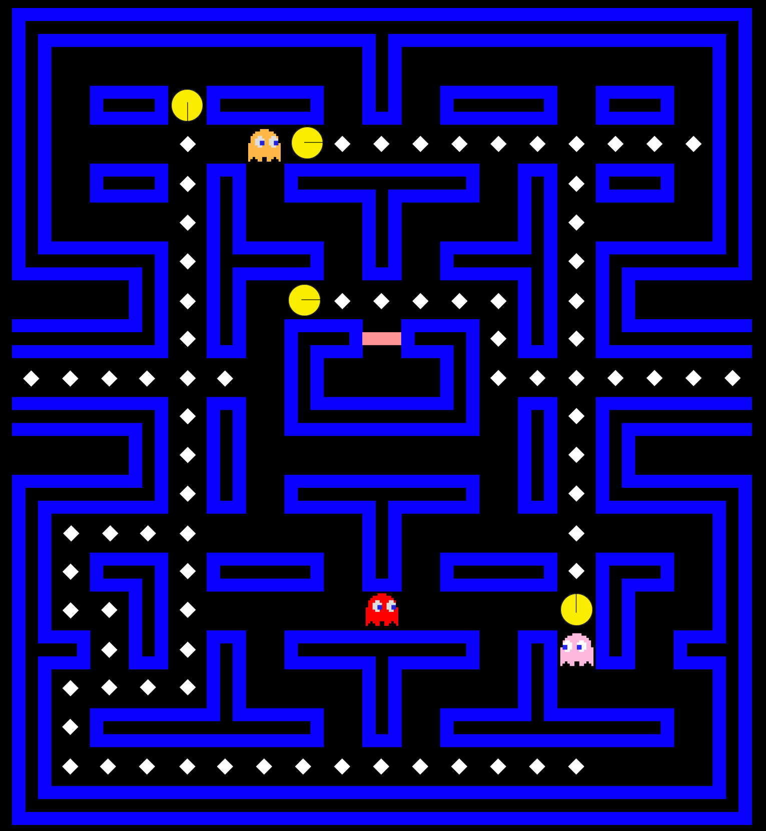
Real Enterprise

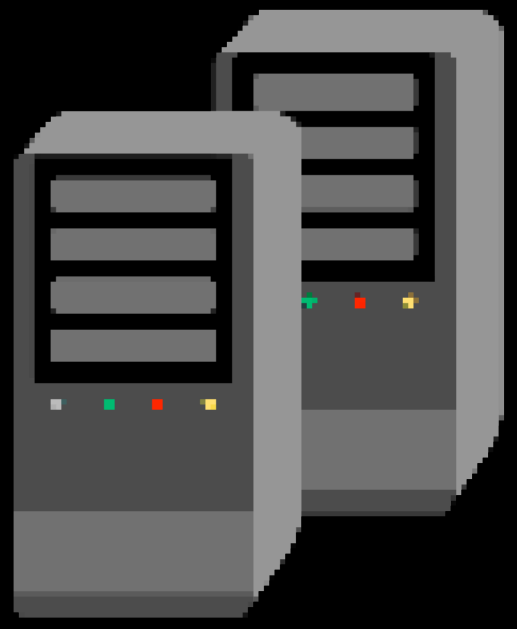












Scoreboard

100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7

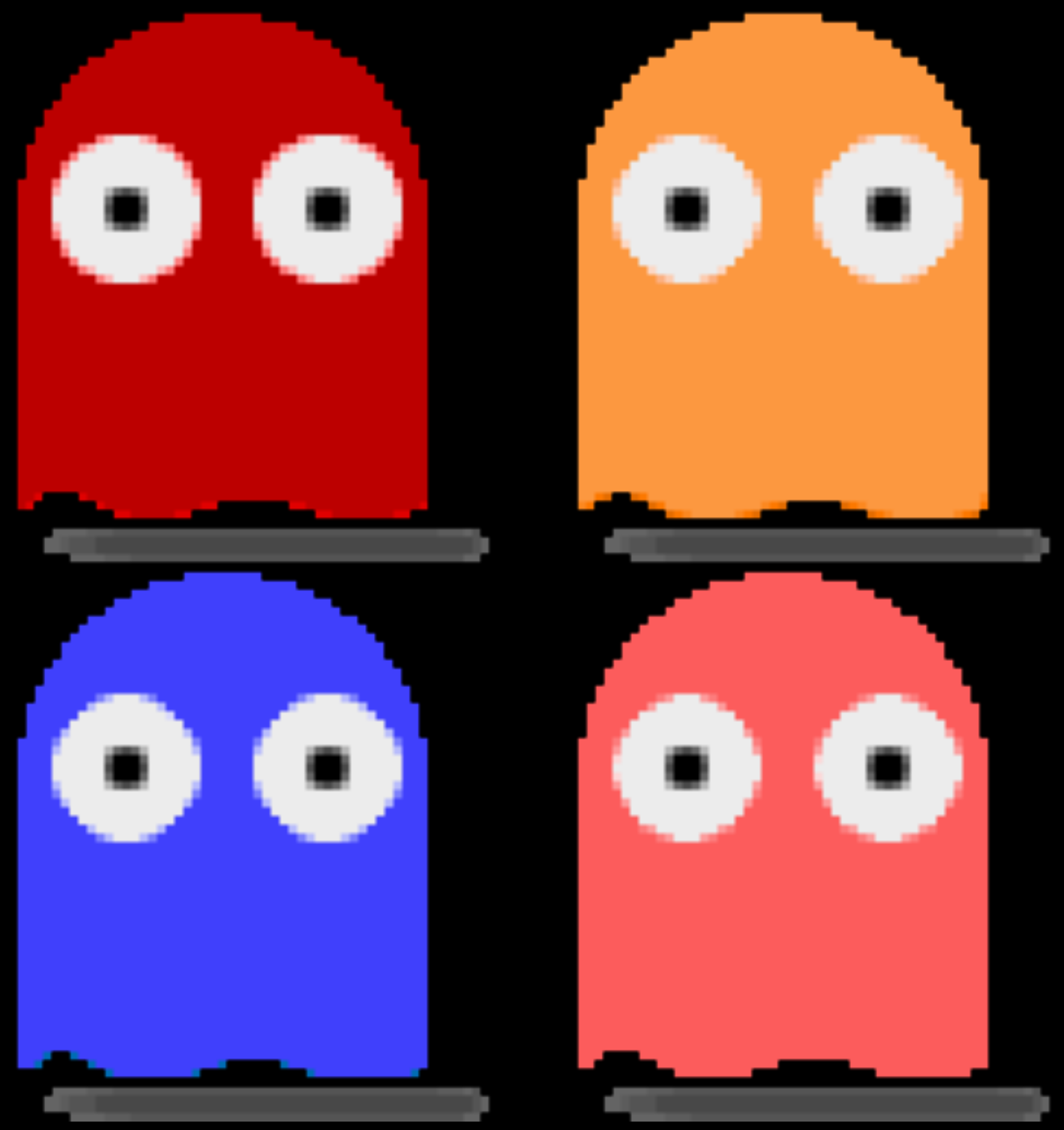
To Summarize

- SERVER PUSH
- PLAIN REQUEST-RESPONSE
- CLIENT-SIDE STREAMING
- SERVER-SIDE STREAMING

To Summarize

- MACHINE LEARNING PIPE
- WHERE SUBSCRIBER CAN WORK SLOW OR FAST
- THIS SHOULD WORK STABLY

TOOLKIT



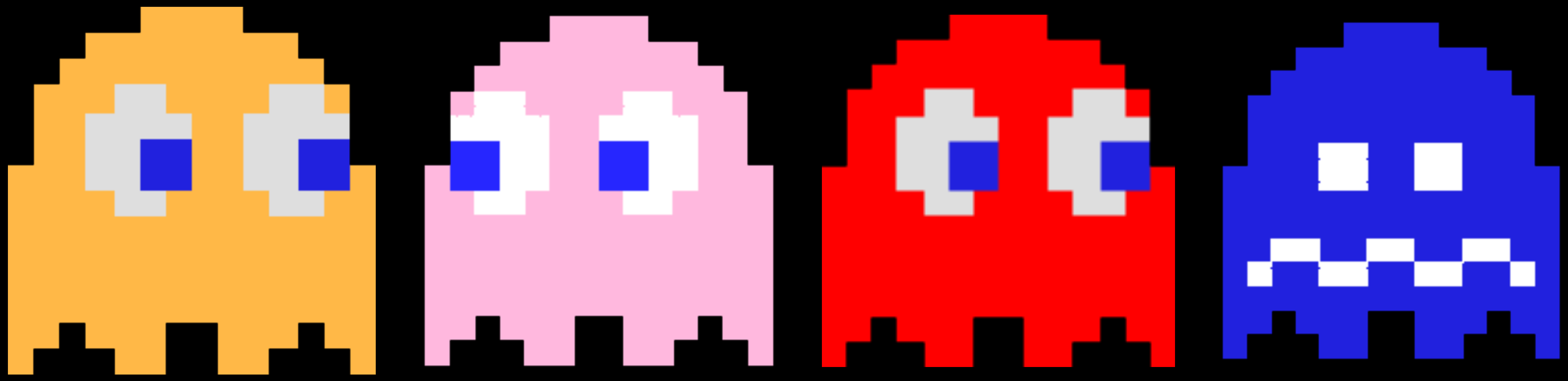
Back-end

- SPRING FRAMEWORK 5
- PROJECT REACTOR 3
- PROTOCOL BUFFER (a.k.a PROTOBUF)

Front-end

- PHASER 3
- REACTOR-JS
- TYPESCRIPT
- PROTOCOL BUFFER (a.k.a PROTOBUF)

OLD HTTP WAY



Why HTTP?

- PLAIN AND SIMPLE
- USED FOR MANY YEARS

```
@RestController
@RequestMapping("/http")
public class HttpGameController {
  ...

  @PostMapping("/start")
  public Mono<Config> start(
      @RequestBody Nickname nickname,
      @CookieValue("uuid") String uuid
  ) {
    return gameService.start(nickname)
      .subscriberContext(Context.of("uuid", UUID.fromString(uuid)));
  }
}
```

DEMO

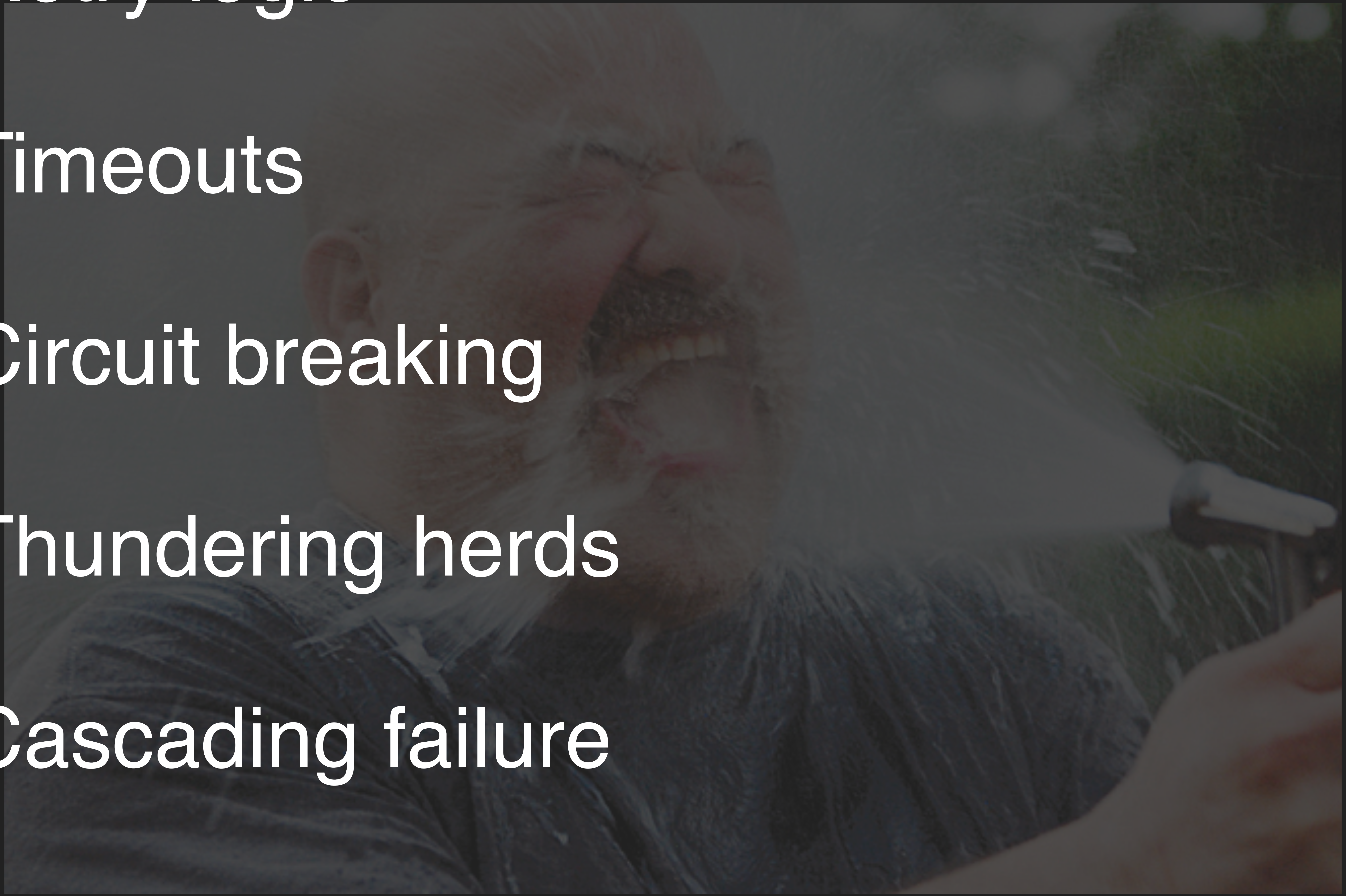
is.gd/webflux

Why NOT HTTP?

- TEXT MESSAGE OVERHEAD
- INEFFICIENT RESOURCE USAGE
- SLOW PERFORMANCE
- COMMUNICATION RIGIDITY
- LACK OF PROPER FLOW CONTROL

HTTP FLOW CONTROL

HTTP FLOW CONTROL



Retry logic

Timeouts

Circuit breaking

Thundering herds

Cascading failure

Configuration

We need Backpressure

PROTOCOLS

PROTOCOLS

- HTTP/1.x
- TCP
- HTTP/2
- ???

PROTOCOLS

- HTTP/1.x
- WEBSOCKET
- HTTP/2
- ???

COMPARISON

- MAINTAINABILITY
    - Frameworks
    - Community/Adoption
- STABILITY
    - Can work OK in unpredicted cases
- PERFORMANCE

WEBSOCKET

Why WebSocket?

- NO OVERHEAD ~ TCP
- HIGH-PERFORMANCE

Why NOT WebSocket?

- COMPLEX DEVELOPMENT
- REINVENT APPLICATION PROTOCOL

Existing Solutions

- SOCKJS + STOMP
- SOCKET.IO

SOCKET.IO

Why Socket.IO?

- MOST POPULAR IN JS WORLD
- TOPIC BASED BINARY/TEXT MESSAGING
- JAVA SERVER BUILT ON TOP OF NETTY

DEMO

is.gd/socketio

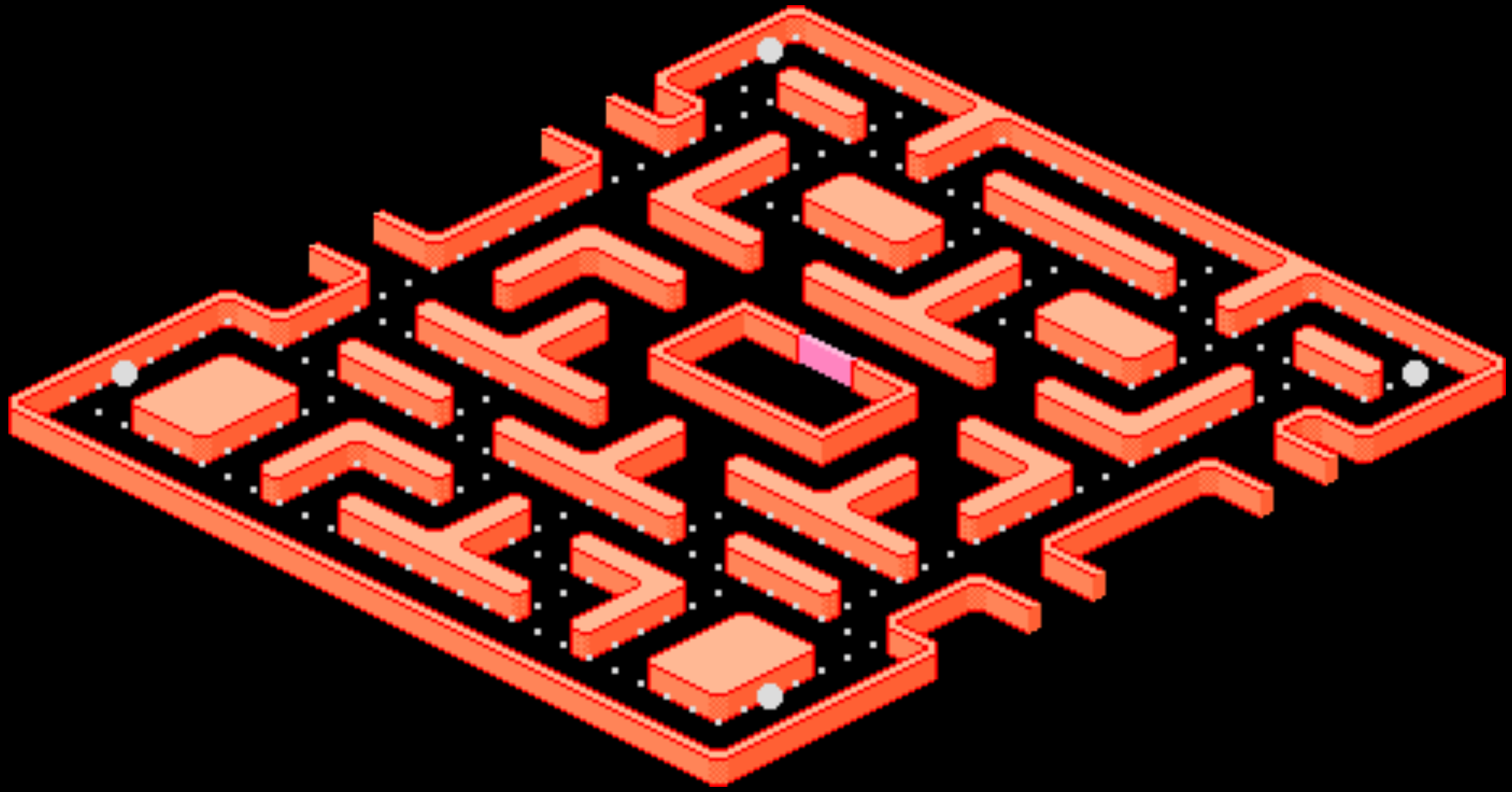
Why Not Socket.IO

```
server.addConnectListener(client -> {});

server.addDisconnectListener(client -> {});

server.addEventListener("start", byte[].class,
  (client, data, ackSender) -> {});

server.addEventListener("locate", byte[].class,
  (client, data, ackRequest) -> {});

server.addEventListener("streamMetricsSnapshots", byte[].class,
  (client, data, ackSender) -> {});
```

Where it is good

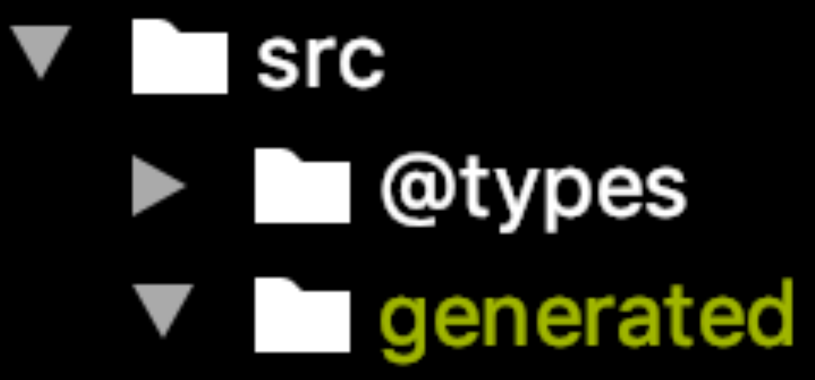
- REALLY GOOD AT JS

GRPC WAY

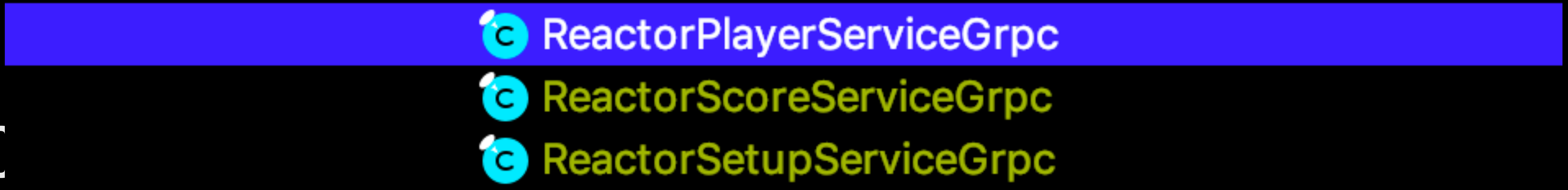


Why gRPC?

```
protobuf {
  protoc {
```



```
se
@GRpcService
public class GrpcPlayerController extends ReactorPlayerServiceGrpc.PlayerServiceImplBase {

    @Override
    public Flux<Player> players(Mono<Empty> message) {
        return playerService
            .players()
            .onBackpressureBuffer()
            .subscriberContext(Context.of("uuid", CONTEXT_UUID_KEY.get()));
    }
}
```



```
}
```

DEMO

is.gd/rgrpc

Why NOT GRPC?

Leaderboard

1. OlegDokuka - 0

Browser

gRPC-Web

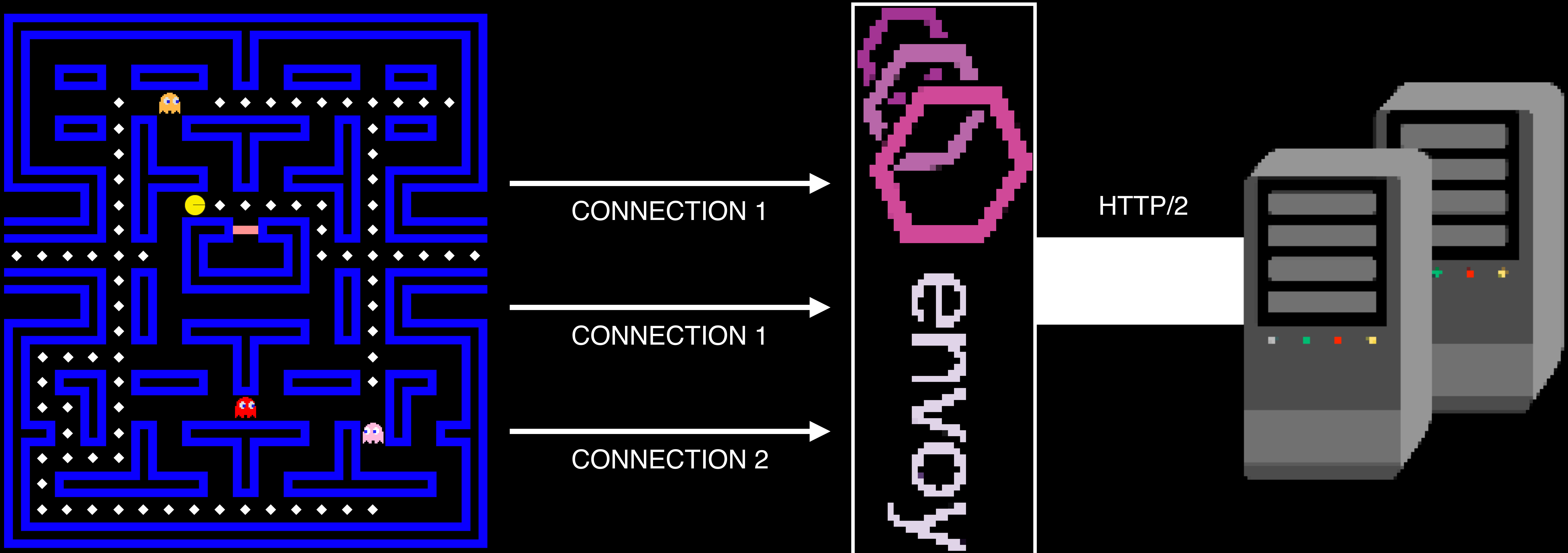
HTTP1.1 / HTTP2
application/grpc-web+proto
application/grpc-web-text
In-body gRPC trailers

gRPC-Web
proxy

HTTP2
application/grpc
Trailers, GOAWAY, HPACK

gRPC Backend
(Go, Java, C++, ...)

GRPC-WEB



CONNECTION 1

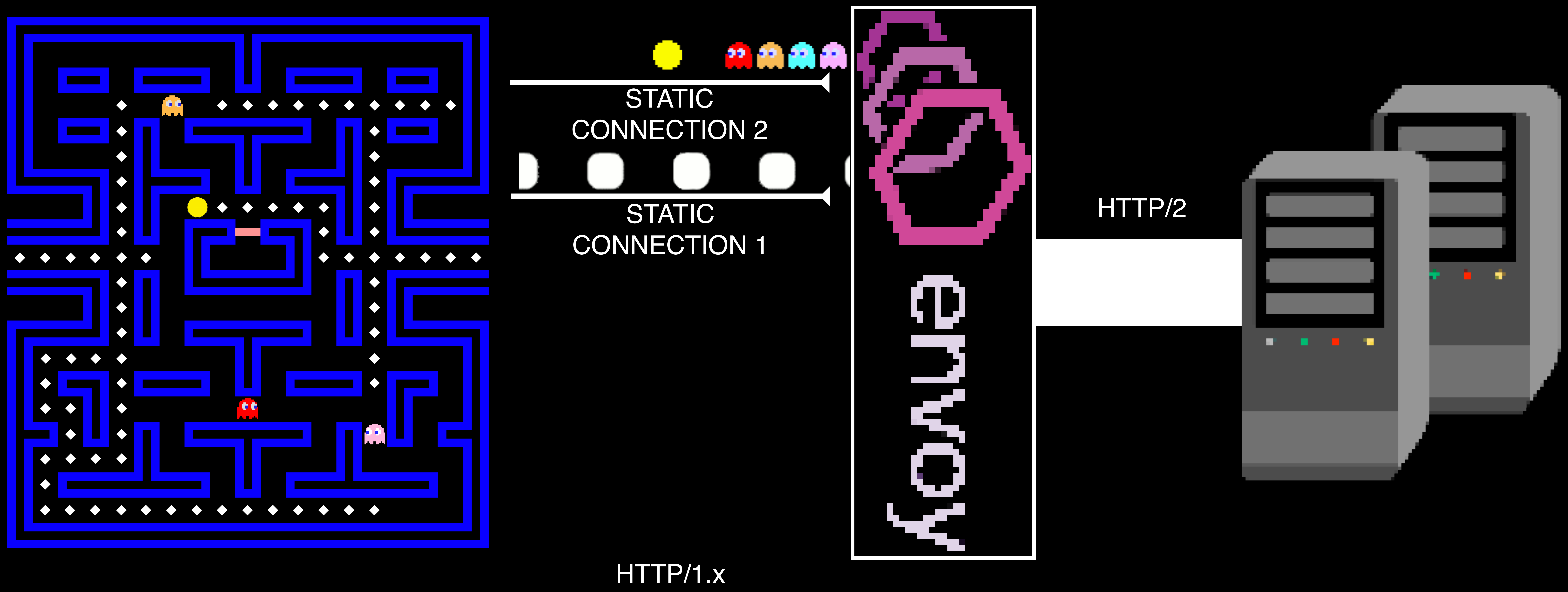
CONNECTION 1

CONNECTION 2

HTTP/1.x

HTTP/2

GRPC-WEB



STATIC CONNECTION 2

STATIC CONNECTION 1

HTTP/2

HTTP/1.x

“WE HAVE BACKPRESSURE CONTROL”

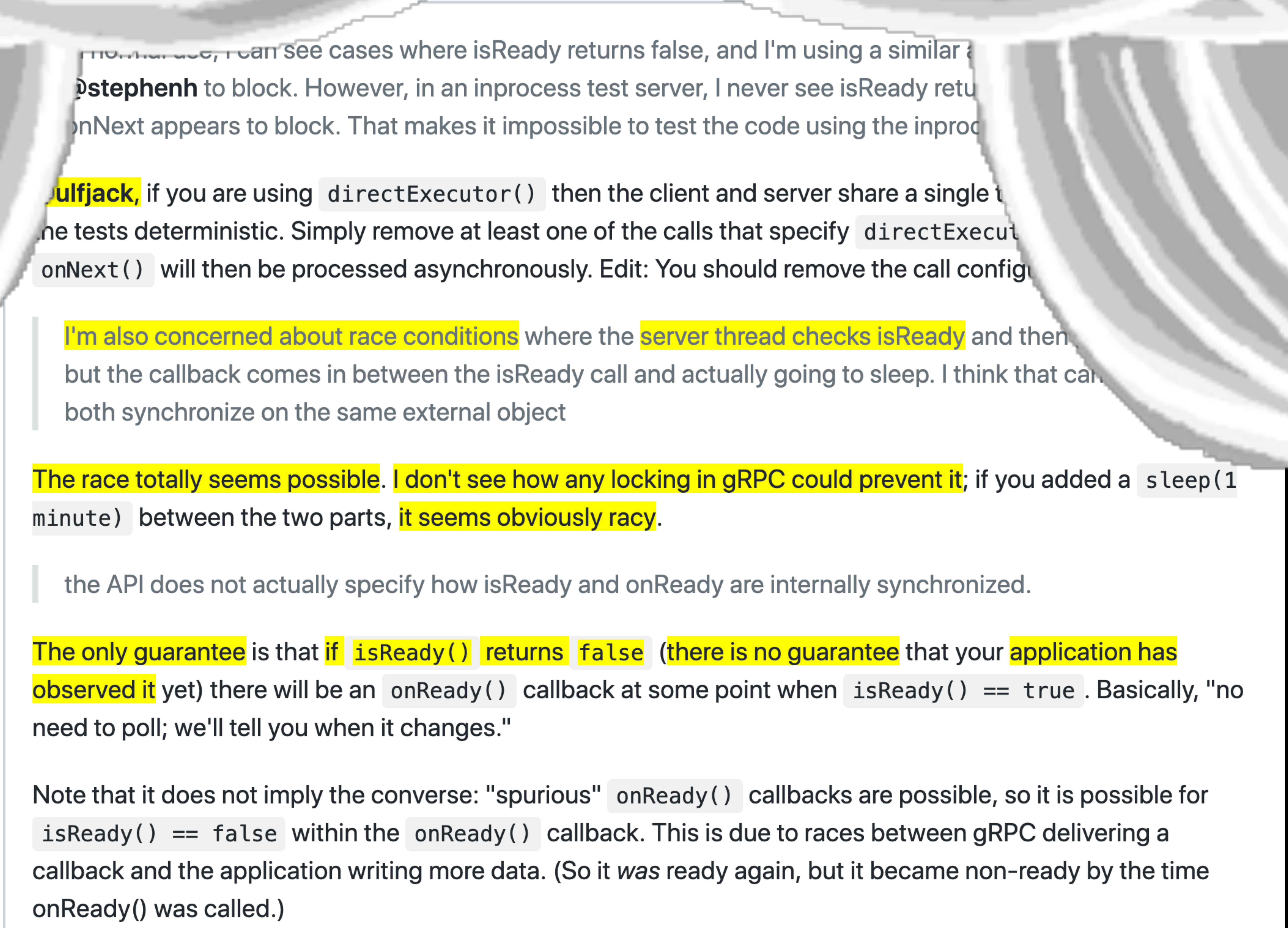
–gRPC

GRPC SUBSCRIBER

```
new CallStreamObserver<>() {
    @Override
    public void onNext(Object value) {
        this.request(5);
    }
}
```

gRPC PUBLISHER

```
if (observer.isReady()) {
    observer.onNext(message);
}
```



normal use, I can see cases where isReady returns false, and I'm using a similar a
@stephenh to block. However, in an inprocess test server, I never see isReady retu
onNext appears to block. That makes it impossible to test the code using the inproc

ulfjack, if you are using `directExecutor()` then the client and server share a single t
he tests deterministic. Simply remove at least one of the calls that specify `directExecut`
`onNext()` will then be processed asynchronously. Edit: You should remove the call config

> I'm also concerned about race conditions where the server thread checks isReady and then
> but the callback comes in between the isReady call and actually going to sleep. I think that ca
> both synchronize on the same external object

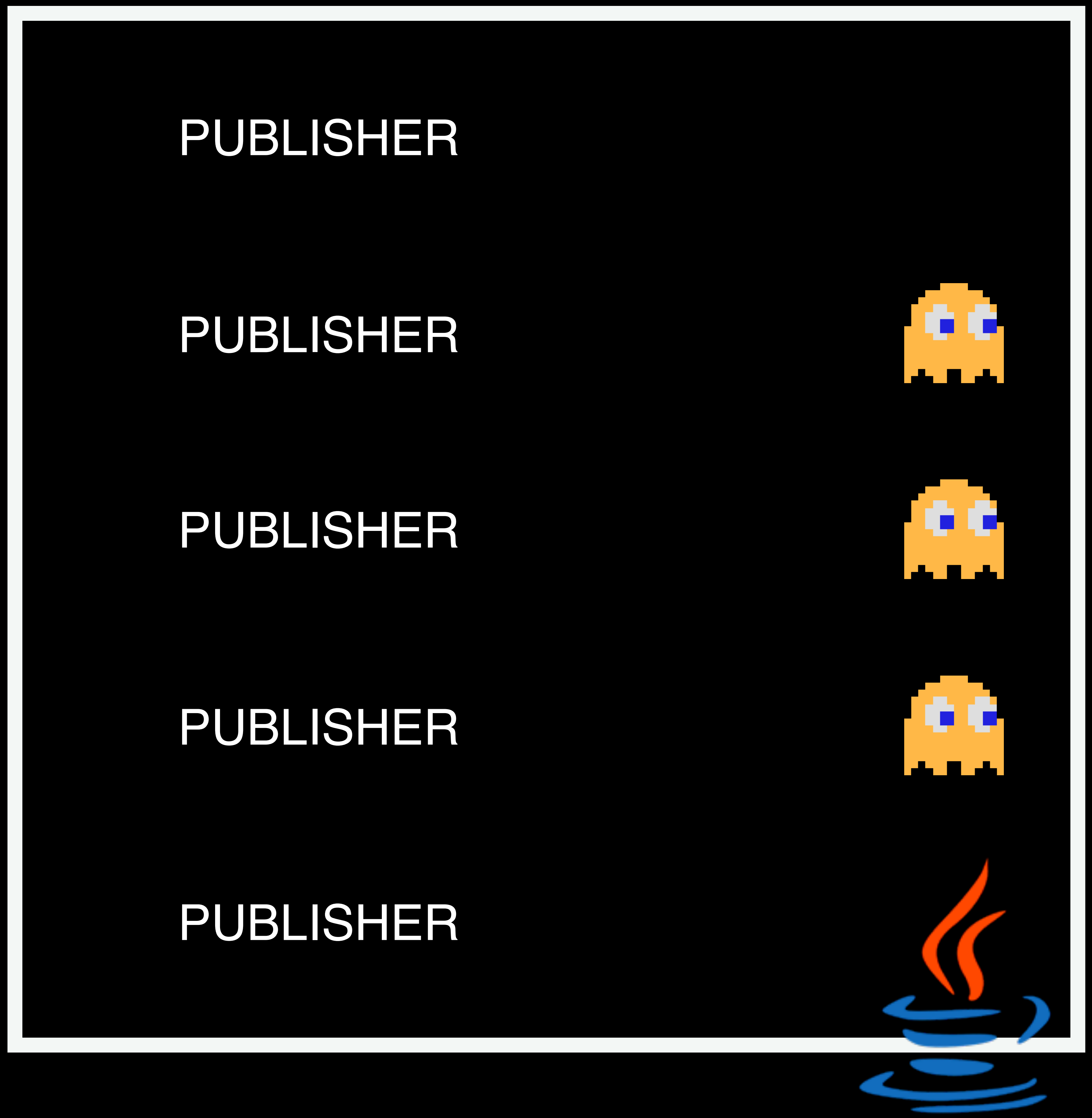
The race totally seems possible. I don't see how any locking in gRPC could prevent it; if you added a `sleep(1 minute)` between the two parts, it seems obviously racy.

> the API does not actually specify how isReady and onReady are internally synchronized.

The only guarantee is that if `isReady()` returns `false` (there is no guarantee that your application has observed it yet) there will be an `onReady()` callback at some point when `isReady() == true`. Basically, "no need to poll; we'll tell you when it changes."

Note that it does not imply the converse: "spurious" `onReady()` callbacks are possible, so it is possible for `isReady() == false` within the `onReady()` callback. This is due to races between gRPC delivering a callback and the application writing more data. (So it *was* ready again, but it became non-ready by the time onReady() was called.)

WHAT IF



PUBLISHER

PUBLISHER

PUBLISHER

PUBLISHER

PUBLISHER

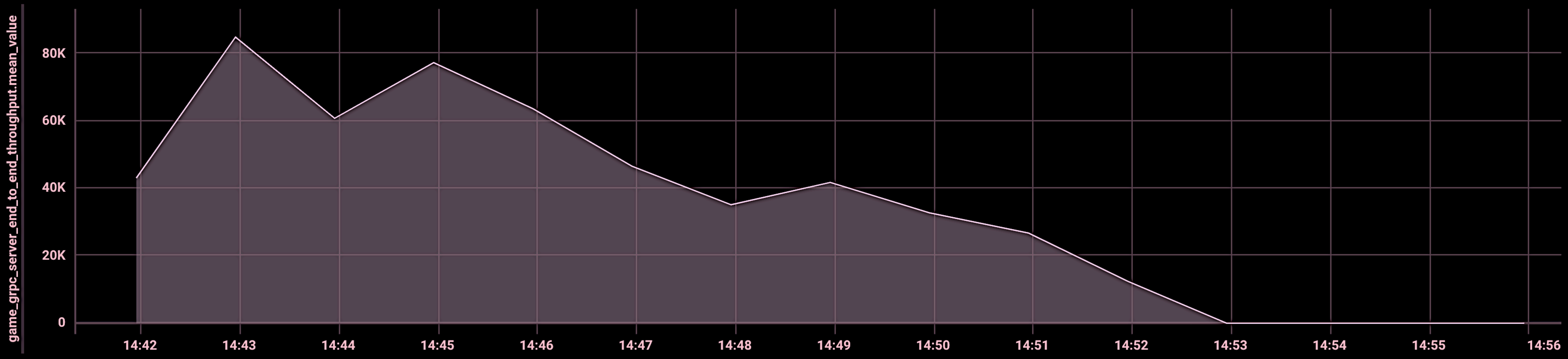
THAT MAY OVERPRODUCE

Scenario

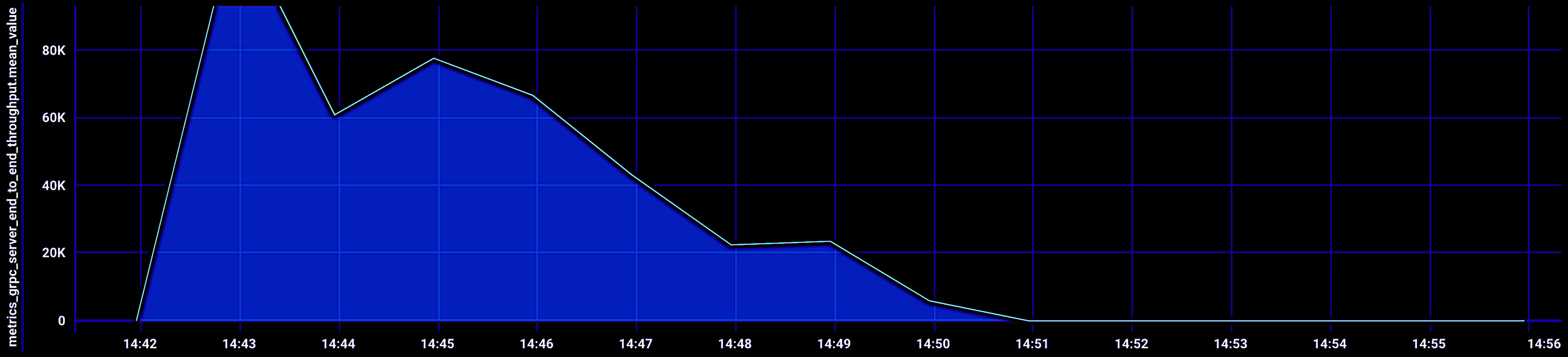
Scenario

https://youtu.be/KxAjxUrGbcY

gRPC Publisher



gRPC Subscribet



gRPC Subscriber
gRPC Publisher
metrics.grpc_server_end_to_end_throughput.mean_value
80K
60K
40K
20K
0
14:42
14:43
14:44
14:45
14:46
14:47
14:48
14:49
14:50
14:51
14:52
14:53
14:54
14:55
14:56

Publisher

Subscriber

game-server-0.0.1.jar (pid 95217)

Buffer Pools
☑ Direct
□ Mapped
Direct

Memory Used: 939,524,383 B

Total Capacity: 939,524,382 B

Count: 61

750 MB
500 MB
250 MB
0 MB
10:54 AM
10:56 AM
10:58 AM
11:00 AM
🟧 Memory Used
🟦 Total Capacity

JVM Setup

- -Xms 256m
- -Xmx 1g
- -XX:MaxDirectMemorySize=1g

AM
🟦 Total Capacity

```
-16] i.g.n.NettyServerTranspo
port failed

java.lang.OutOfMemoryError: D
            at java.base/java.nio
.java:175) ~[na:na]
            at java.base/java.nio
(DirectByteBuffer.java:118) ~
            at java.base/java.nio
```

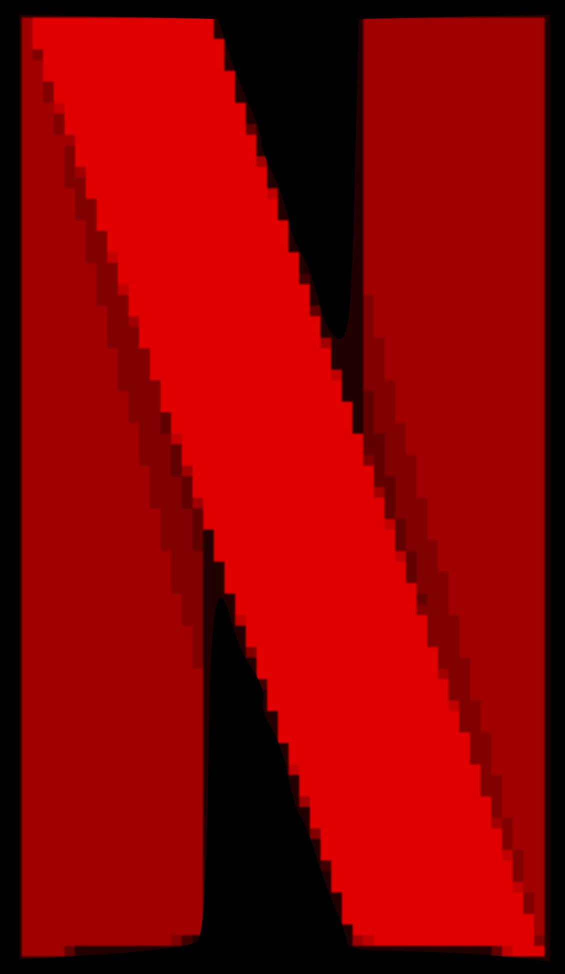
> “WE HAVE BACKPRESSURE CONTROL”
>
> –gRPC

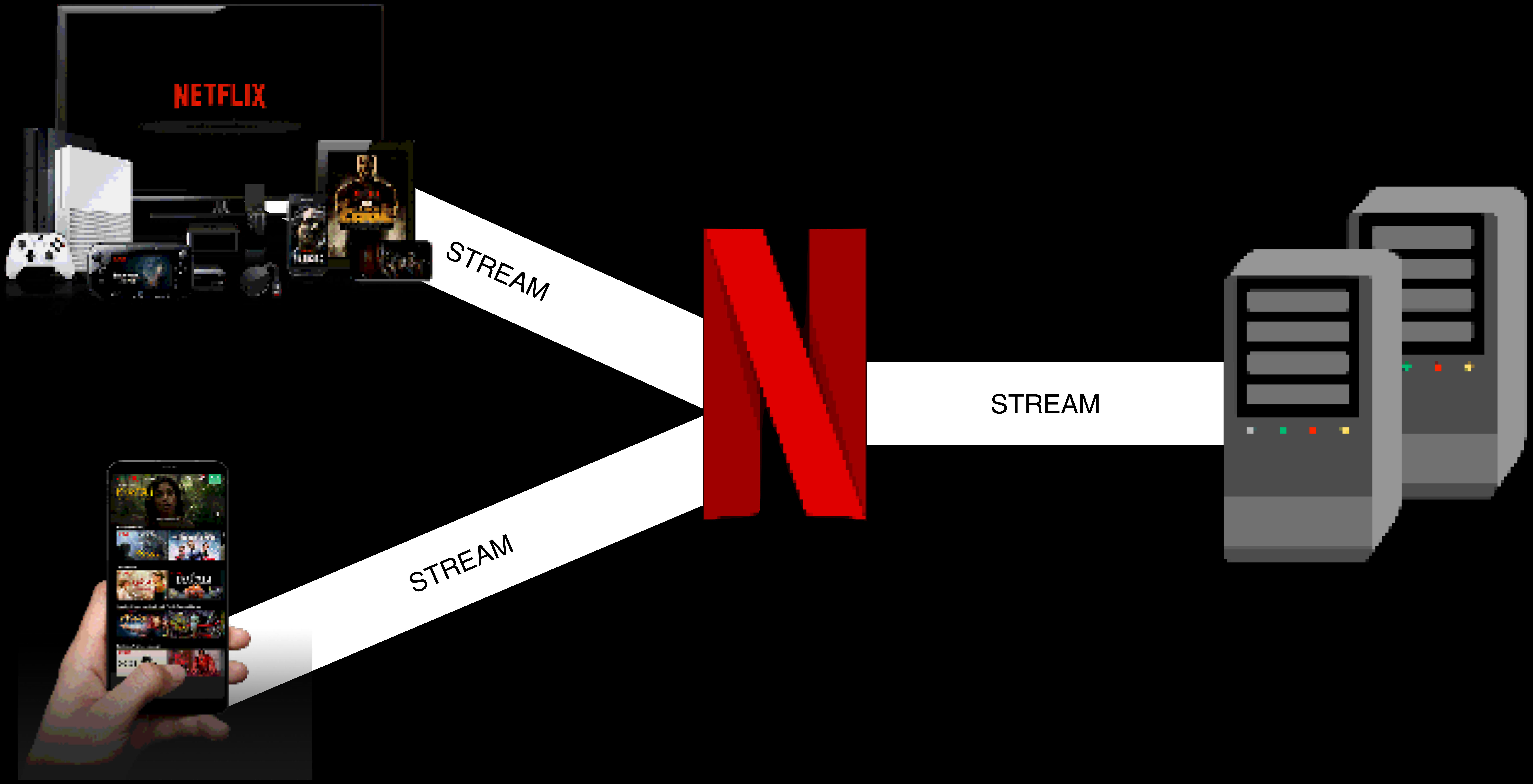
“YEAH… YOU HAVE… BUT… NOT REALLY”

–PRODUCTION

Summary

- Everything is either SLOW, HARD to implement or LACKS browser support
- Flow control is far from needed
- Do you want to waste your time in searching how to solve the problems???





STREAM

STREAM

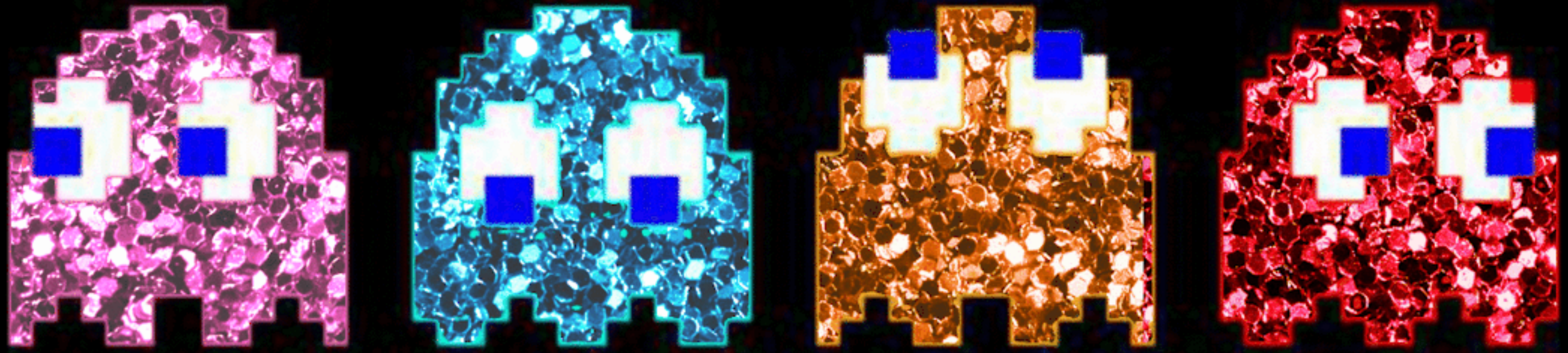
STREAM

Netflix case study on gRPC

- Reactive Streaming Service Networking with Ryland Degnan
  (ex Netflix Edge Platform)

  https://bit.ly/2FUvHG3

RSOCKET WAY



What is RSocket?

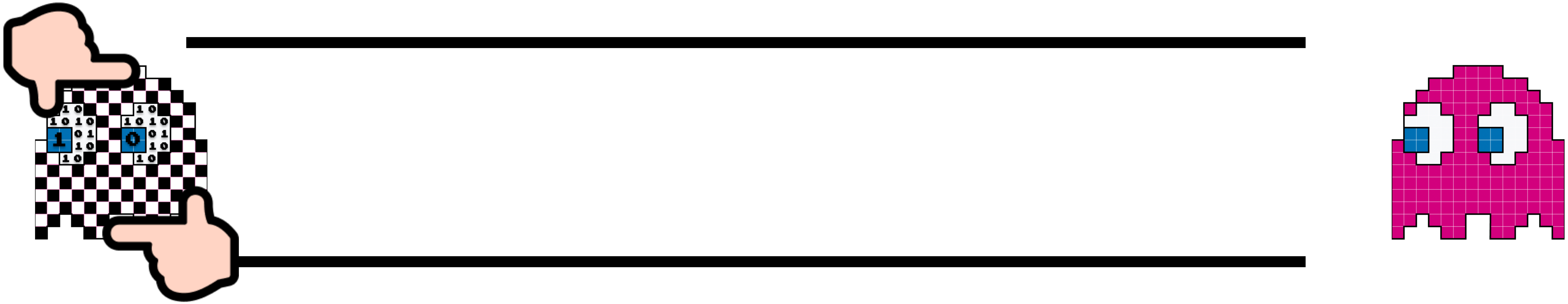
What is RSocket?

REACTIVE-STREAMS as NETWORK PROTOCOL

DEMO

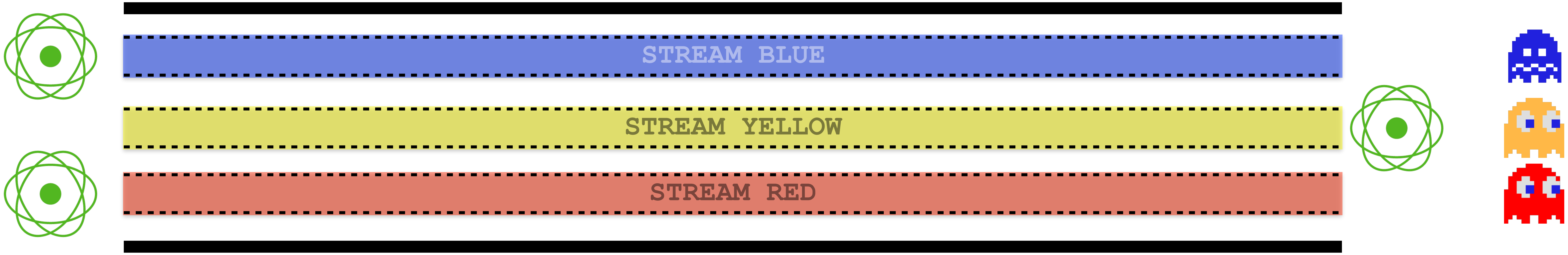
is.gd/rsocket

Binary



Multiplexed

Multiplexed



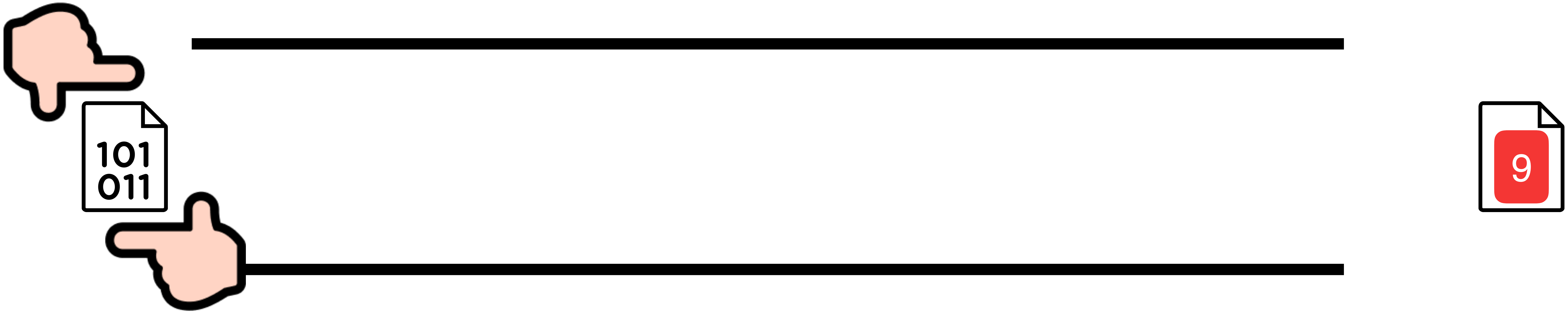
Project Reactor

Transport Agnostic

WebSocket [illegible]

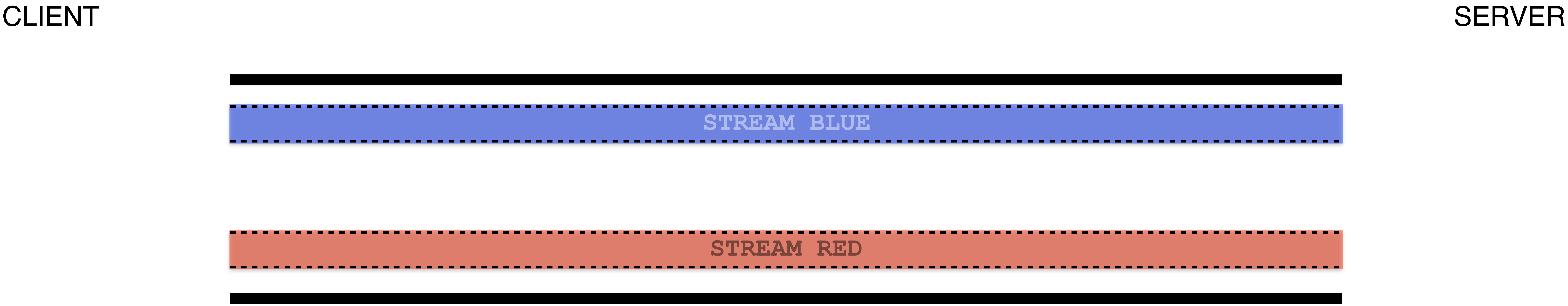
Reactive-Streams

Backpressure



Peer-to-peer

Client can implement request handler



Multi

Interaction modes

Interaction modes

Request-Response

STREAM YELLOW

Interaction modes

Fire-and-Forget

STREAM YELLOW

Interaction modes

Request-Stream

STREAM YELLOW

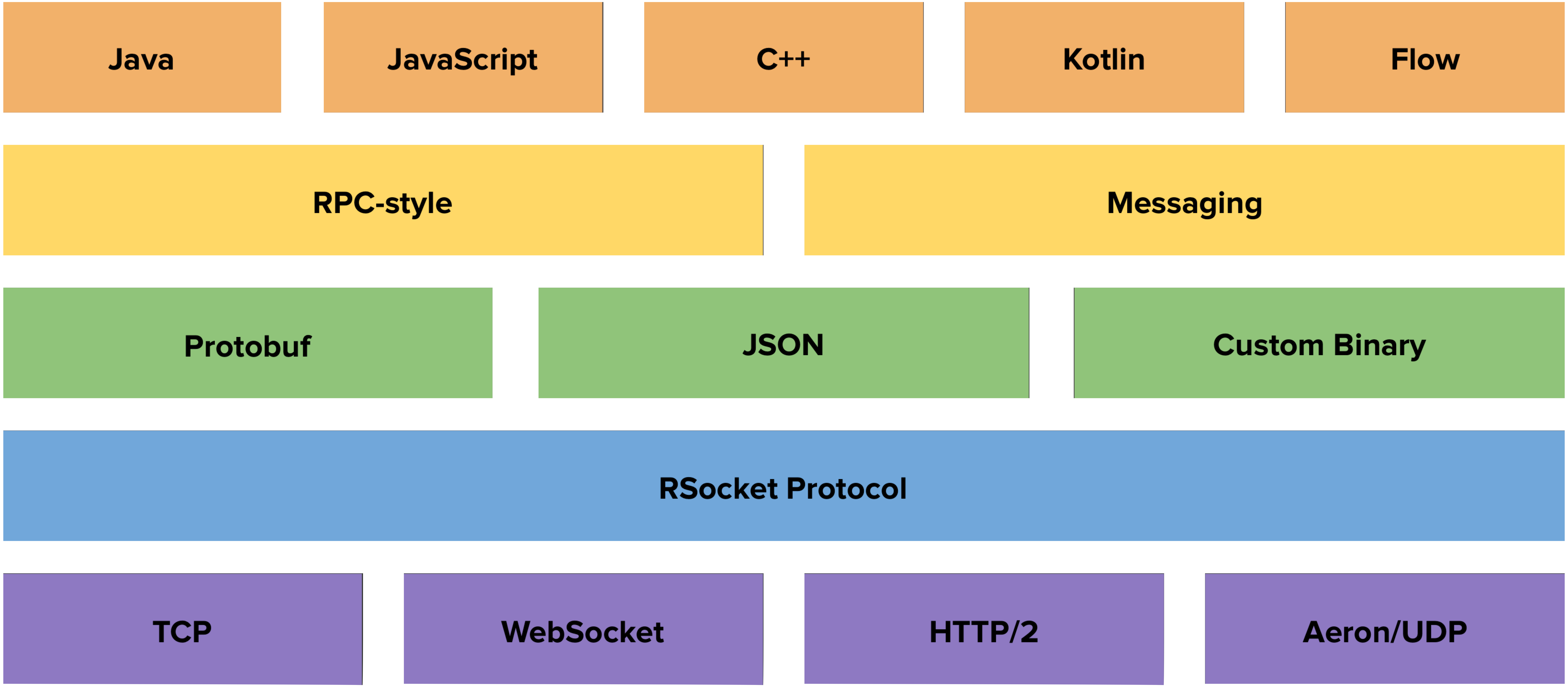
Interaction modes

Stream-Stream

STREAM YELLOW

Notable Features

- LEASING - GIVE CAPACITY TO CLIENTS, AVOID CIRCUIT BREAKERS
- RESUMABILITY - RESUME STREAMS IF CONNECTION LOST
- FRAGMENTATION - SPLIT LARGE PAYLOAD ONTO SMALLER CHUNKS



Java JavaScript C++ Kotlin Flow

RPC-style Messaging

Protobuf JSON Custom Binary

RSocket Protocol

TCP WebSocket HTTP/2 Aeron/UDP

Java

```
RSocketFactory
   .receive()
   .acceptor()
   .transport()
   .start()
   .flatMap(…)
   .block();
```

- Entry
- Server Builder
- Connection Handler
- Transport
- Startup
- Listen For Startup
- Keep Main Thread

To Respond

```
new AbstractRSocket() {
  @Override
  public Flux<Payload> requestStream(Payload payload) {
    return Flux.range(0, 100)
      .map(i -> ByteBufPayload.create("Hello " + i));
  }
}
```

Data Type

```
new AbstractRSocket() {
  @Override
  public Flux<Payload> requestStream(Payload payload) {
    return Flux.range(0, 100)
      .map(i -> ByteBufPayload.create("Hello " + i));
  }
}
```

Java

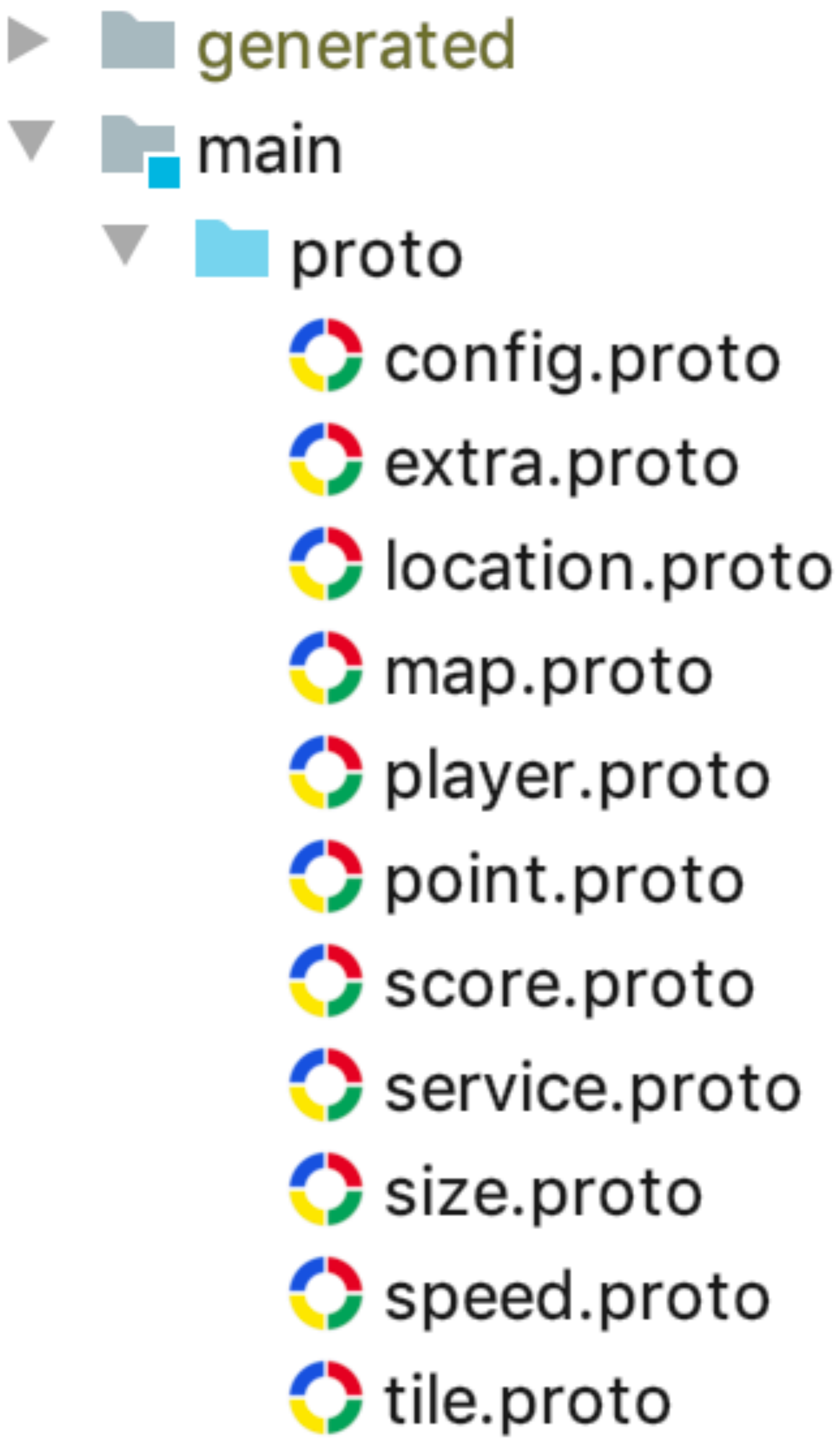
```
RSocketFactory
 .receive()
 .acceptor((setup, sendingSocket) ->
    Mono.just(new AbstractRSocket() {}))
 .transport(WebsocketServerTransport.create(8080))
 .start()
 .flatMap(CloseableChannel::onClose)
 .block();
```

JS

```
new RSocketClient({
    setup: {
        dataMimeType: 'text/plain',
        keepAlive: 1000000,
        lifetime: 100000,
        metadataMimeType: 'text/plain',
    },
    transport: new RSocketWebSocketClient({
        host: 'localhost',
        port: 8080,
    }),
    responder: {
        requestStream: (payload) => {}
    }
});
```

```
RSocket rSocket = …;

rSocket.requestChannel(Flux.range())
       .subscribe();

rSocket.requestStream(payload)  // Flux
       .subscribe();

rSocket.requestResponse(payload)// Mono
       .subscribe();
```

RPC API



```
▶ generated
▼ main
  ▼ proto
      config.proto
      extra.proto
      location.proto
      map.proto
      player.proto
      point.proto
      score.proto
      service.proto
      size.proto
      speed.proto
      tile.proto
```

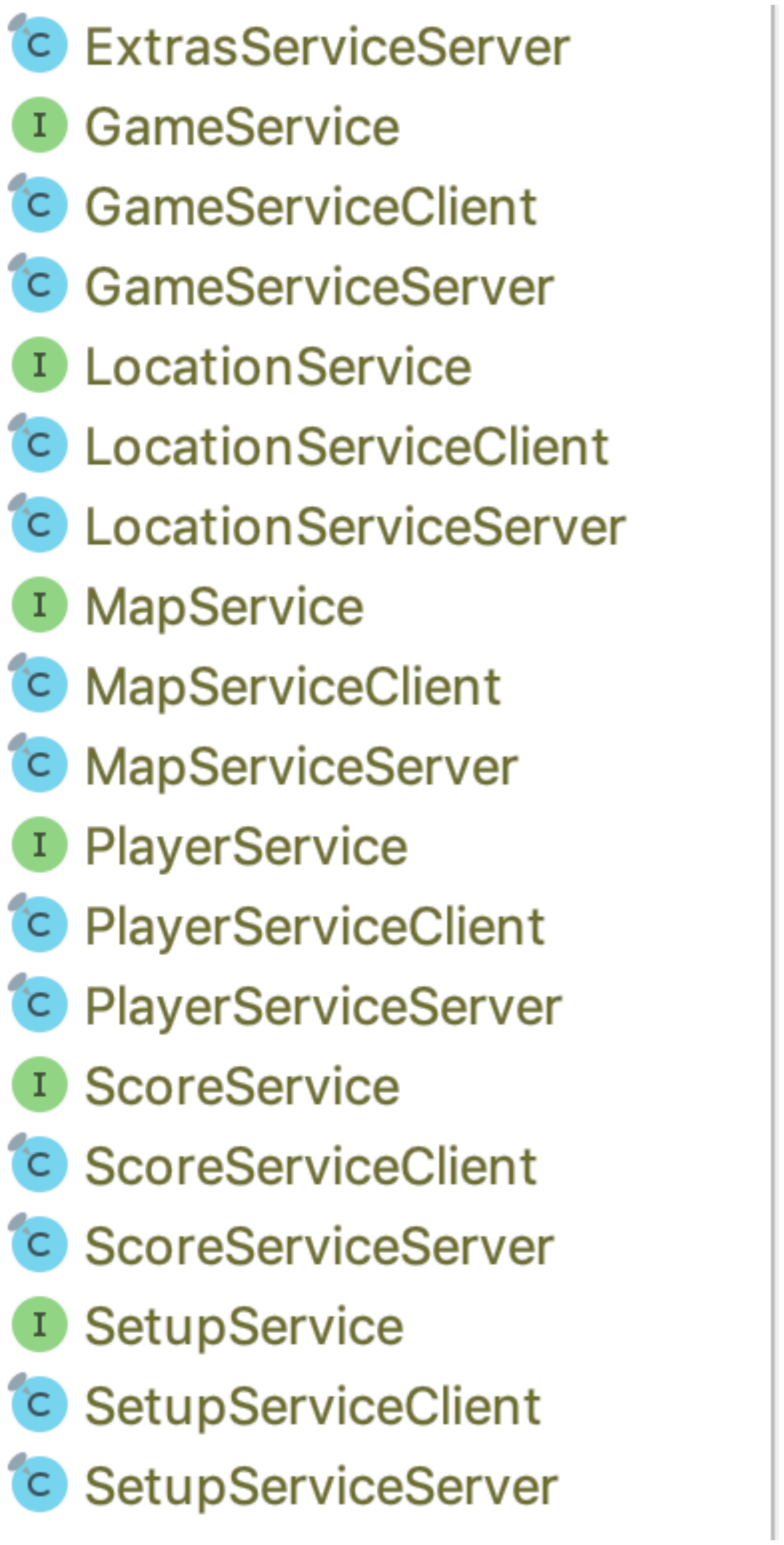
RPC API

```
implementation 'io.rsocket.rpc:rsocket-rpc-core'
```

RPC API

```
protobuf {
  generatedFilesBaseDir = "${projectDir}/src/generated"

  protoc {
    artifact = 'com.google.protobuf:protoc'
  }

  plugins {
    rsocketRpc {
      artifact = "io.rsocket.rpc:rsocket-rpc-protobuf"
    }
  }

  generateProtoTasks {
    ofSourceSet('main')*.plugins {
      rsocketRpc  {}
    }
  }
}
```

RPC API



- C ExtrasServiceServer
- I GameService
- C GameServiceClient
- C GameServiceServer
- I LocationService
- C LocationServiceClient
- C LocationServiceServer
- I MapService
- C MapServiceClient
- C MapServiceServer
- I PlayerService
- C PlayerServiceClient
- C PlayerServiceServer
- I ScoreService
- C ScoreServiceClient
- C ScoreServiceServer
- I SetupService
- C SetupServiceClient
- C SetupServiceServer

RSocketRpcService

```
public class ExtrasController
       implements org.coinen.pacman.ExtrasService {

  …

  @Override
  public Flux<Extra> extras(Empty message, ByteBuf metadata) {
    return extrasService.extras();
  }
}
```

SPRING-MESSAGING

```
implementation 'org.springframework.boot:spring-boot-starter-rsocket'
```

SPRING-MESSAGING

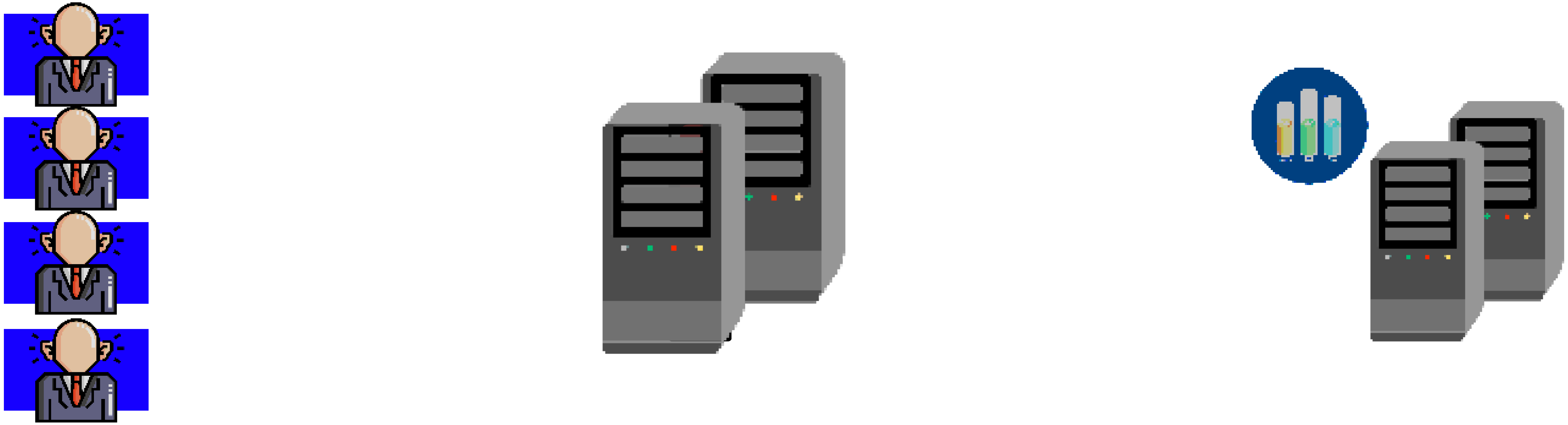
```
server.port=3000
spring.rsocket.server.transport=websocket
```

SPRING-MESSAGING

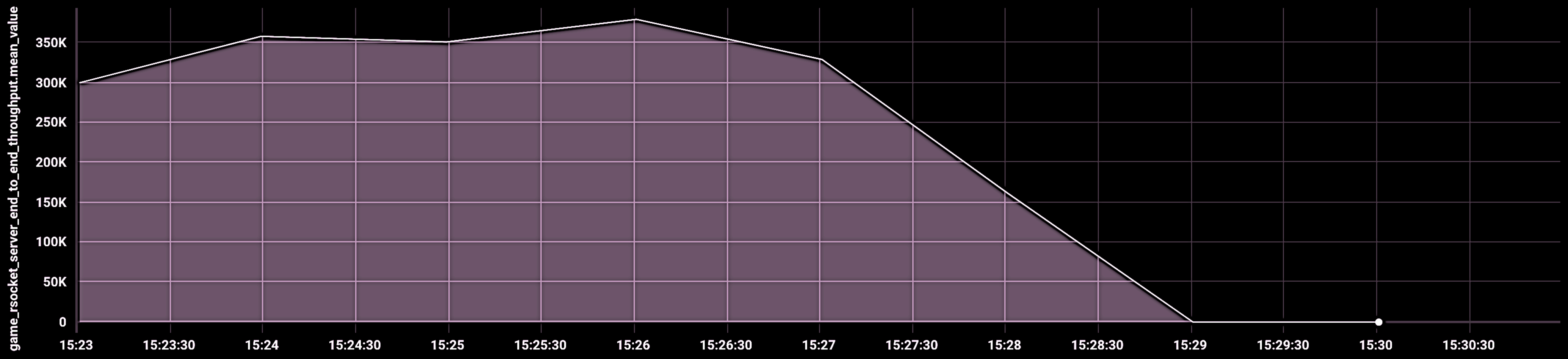
```
@Controller
@MessageMapping(“my.route.name”)
public class ExtrasController {
  …

  @MessageMapping(“handle.extras”)
  public Flux<Extra> extras() {
    return extrasService.extras();
  }
}
```

STRESS TEST

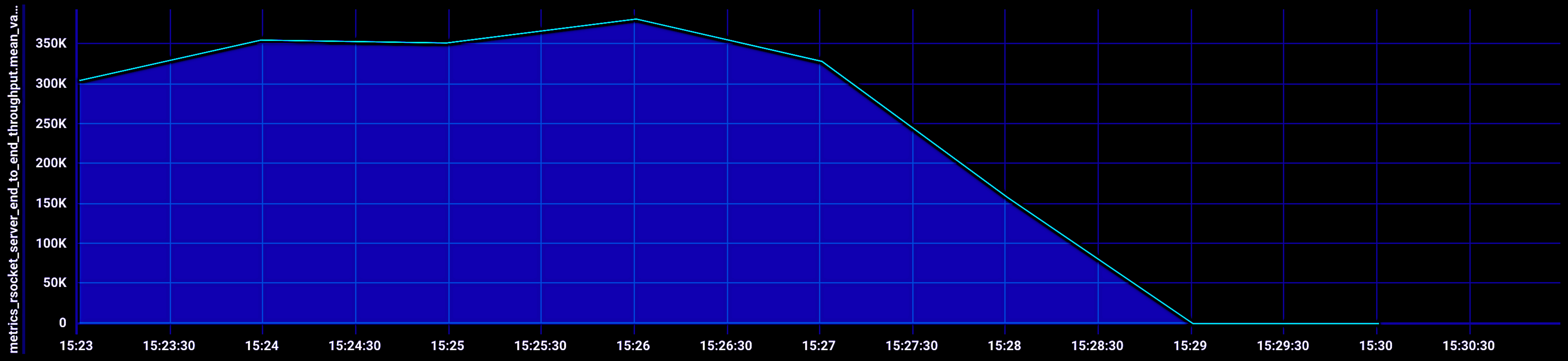
Scenario



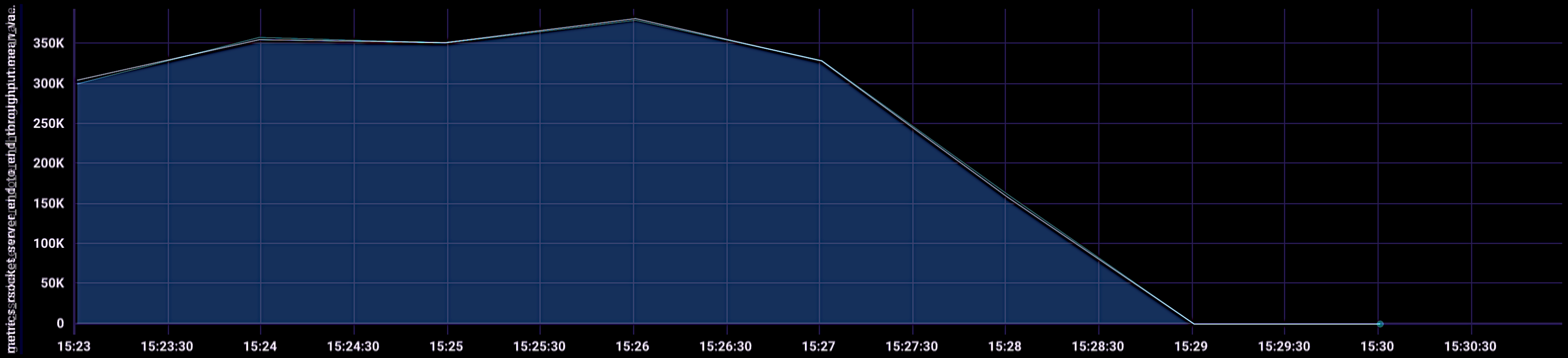
RSocket Publisher



RSocket Subscriber



RSocket Subscriber



Advantages

- SIMPLICITY IN DEVELOPMENT
- EFFICIENT RESOURCE USAGE
- HIGH PERFORMANCE
- HIGH FLEXIBILITY
- EFFECTIVE RELIABILITY

Disadvantages

- STILL UNDER DEVELOPMENT
- NARROW ADOPTION (FOR NOW)

Maintainers

facebook ✓

netifi

Pivotal

Alibaba Cloud

Summary

	PERFORMANCE	RELIABILITY	ADOPTION / COMUNITY	DEVELOPERS EXP
HTTP 1.X	👎	👎	👍	👍
WEBSOCKET	👍	🤔	🤔	😿
GRPC(HTTP/2)	👍 👍	👍/👎	👍	👍
RSOCKET	👍 👍	👍	😿	👍

Summary

- EACH PROTOCOL HAS IT`S BENEFITS
- SOCKET.IO IS THE BEST IN JS WORLD
- gRPC PERFORMS REALLY WELL FOR SERVER
- BUT REACTIVE IS ABOUT RESILIENCY
- WHERE RSOCKET FULLY COVERS OUR USE-CASE

Resources

@OlehDokuka

@netifi_inc

- COMMUNITY -> https://community.netifi.com
- VIDEO CHANNEL -> https://bit.ly/2Fku9VC
- RSOCKET IN SPRING -> https://bit.ly/2OiUmrD
- CLOUD NATIVE RSOCKET -> https://bit.ly/2JvDFdJ