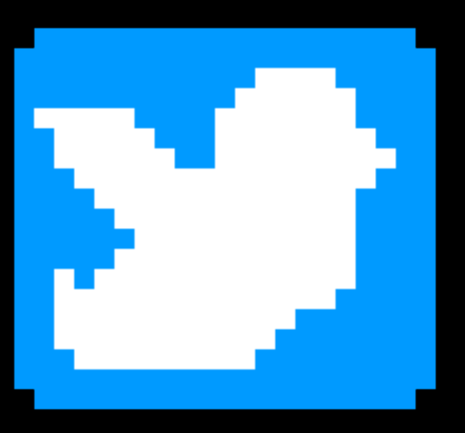
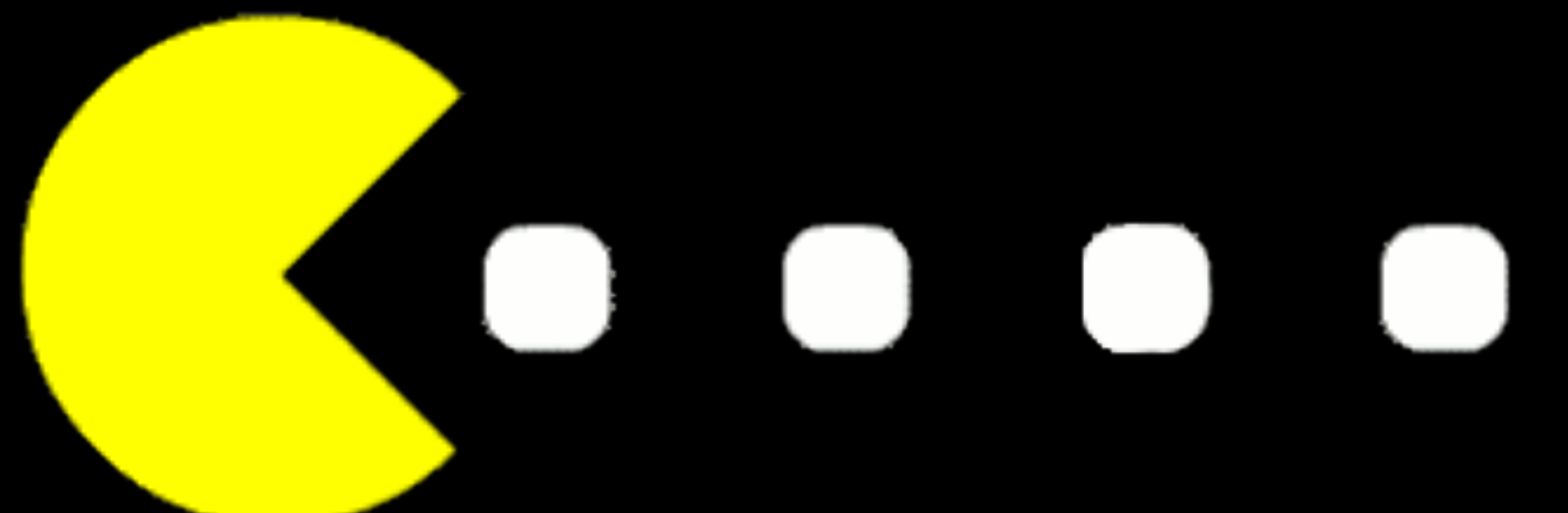
MULTIPLAYER PAC-MAN WITH RSOCKET

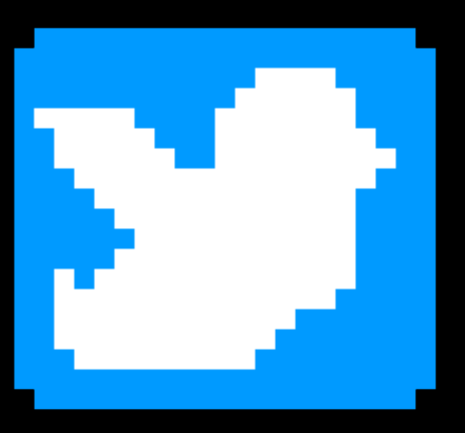


@netifi_inc

@OlehDokuka

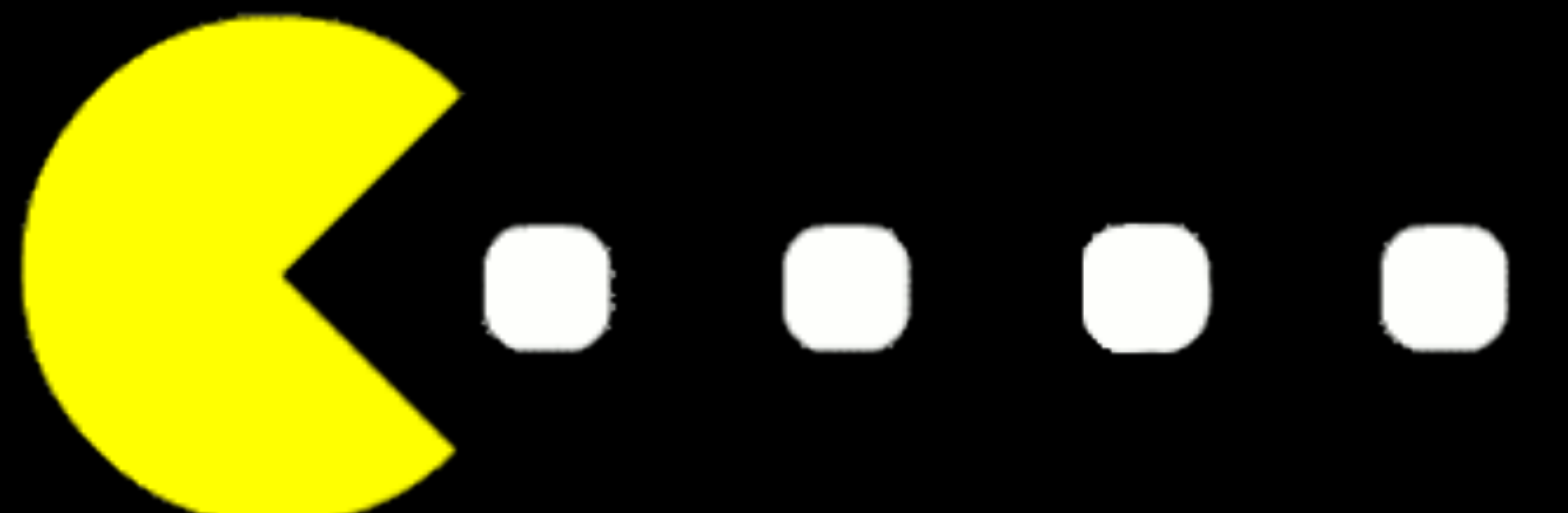


MULTIPLAYER PAC-MAN WITH RSOCKET



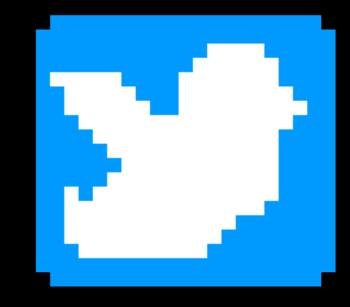
@netifi_inc

@OlehDokuka



Oleh Dokuka

- WORK FOR NETIFI
- REACTIVE GEEK
- REACTOR 3 CONTRIBUTOR
- RSOCKET PROJECT TEAM MEMBER
- BOOKS AUTHOR



@OlehDokuka

Hands-On Reactive Programming in Spring 5

Build cloud-ready, reactive systems with Spring 5 and Project Reactor

Oleh Dokuka and Igor Lozynskyi

Agenda

Agenda

- DEFINE PROBLEM

Agenda

- DEFINE PROBLEM
- COMPARE PROTOCOLS

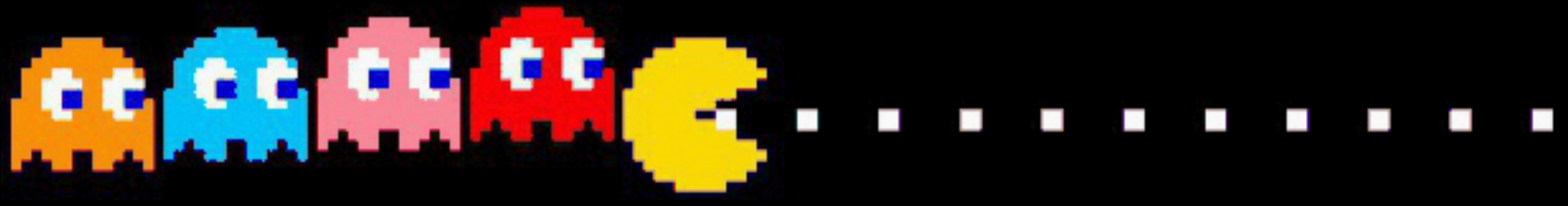
Agenda

- DEFINE PROBLEM
- COMPARE PROTOCOLS
- HAVE FUN

Agenda

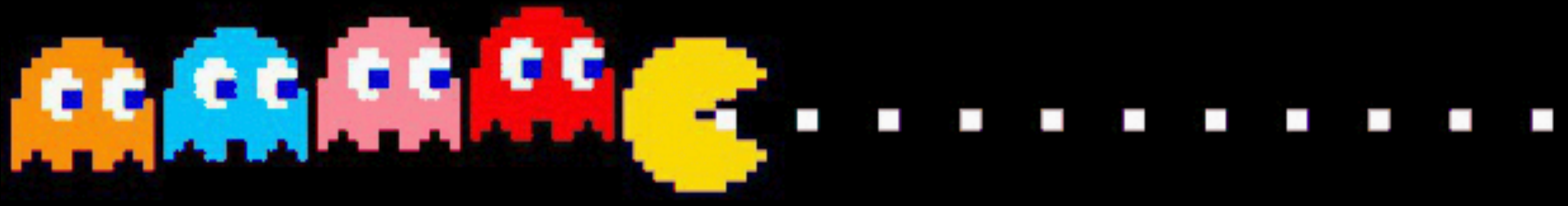
- DEFINE PROBLEM
- COMPARE PROTOCOLS
- HAVE FUN
- DEFINE THE BEST PROTOCOL

MULTIPLAYER



REQUIREMENTS

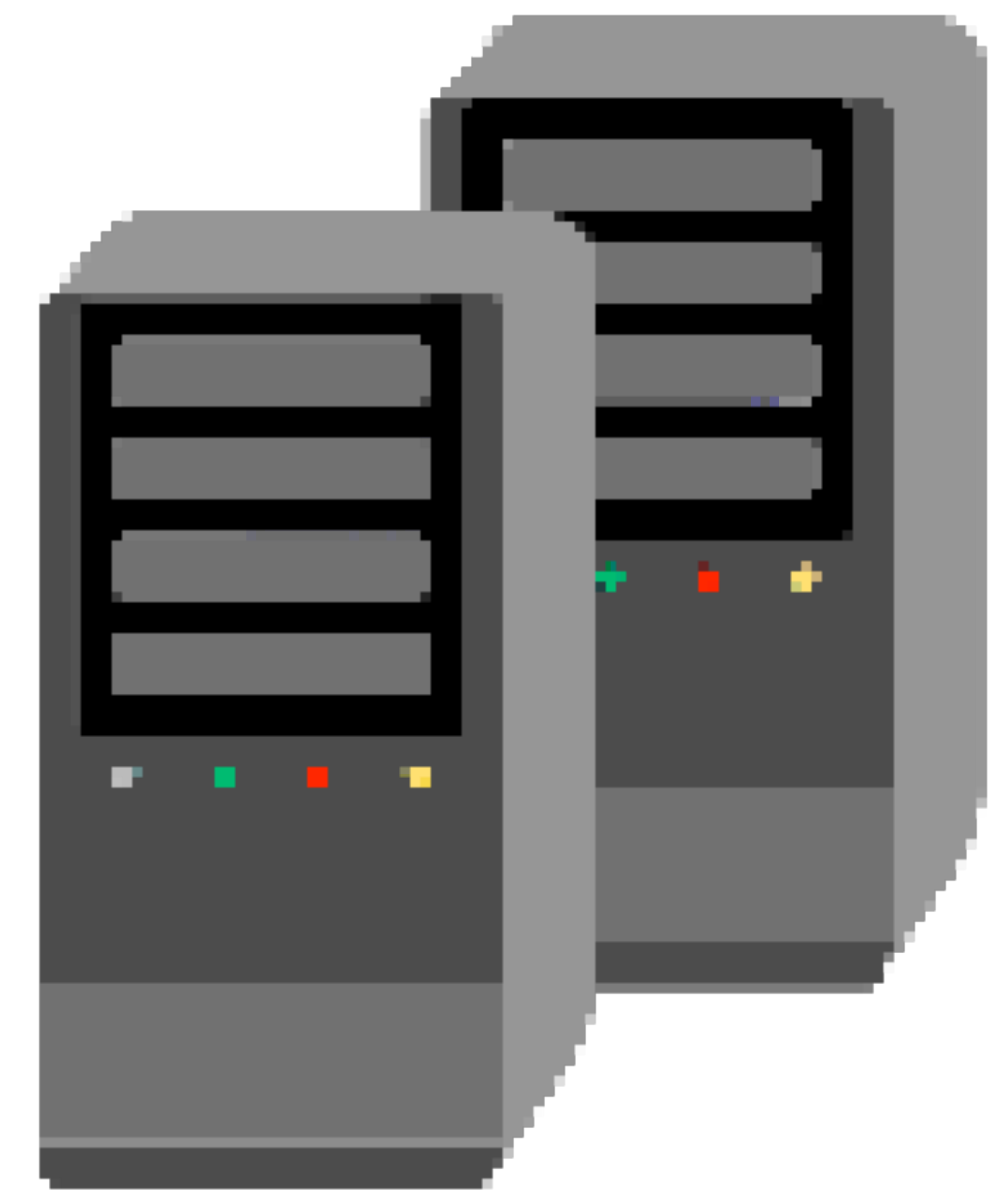
MULTIPLAYER



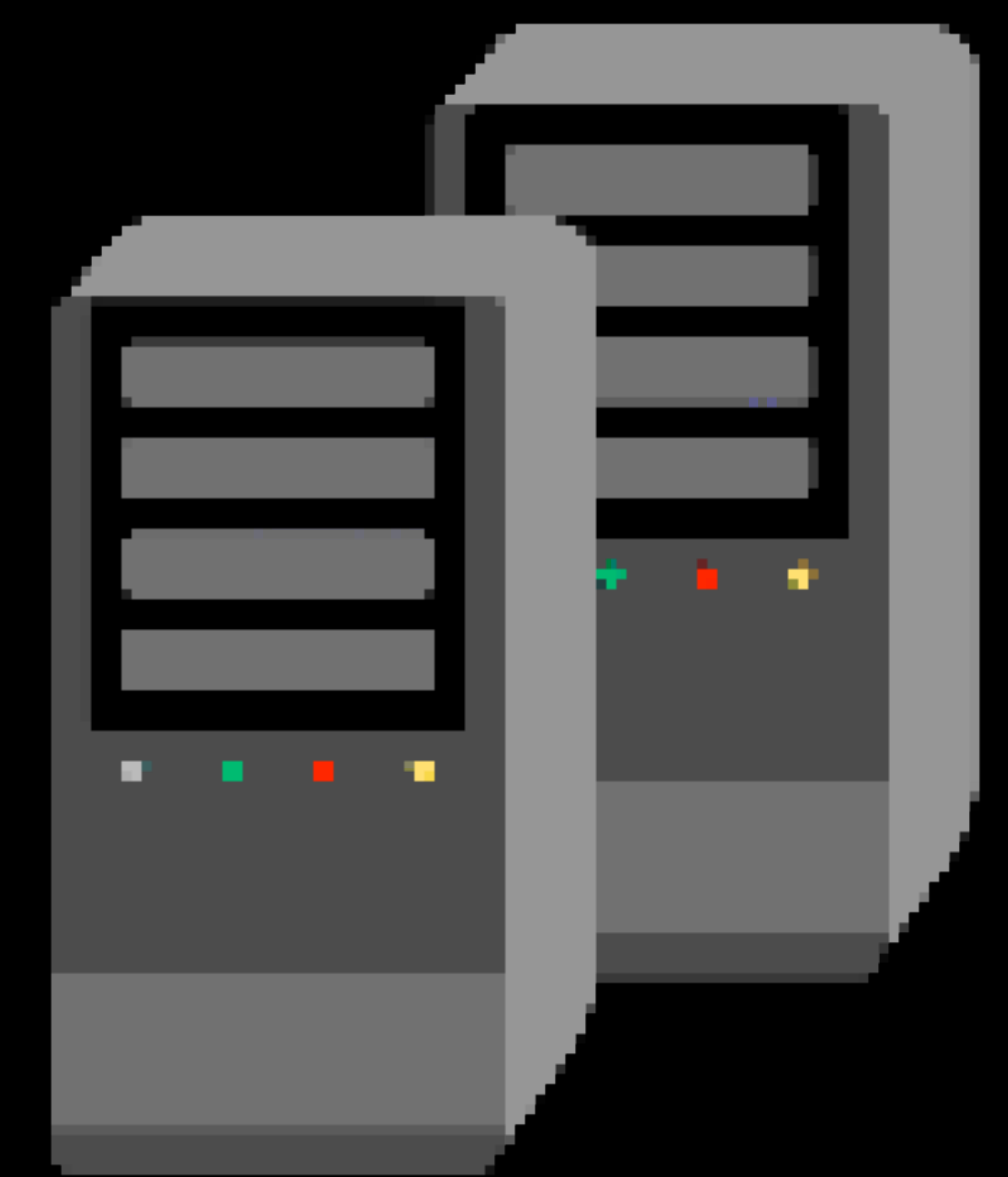
REQUIREMENTS

Step 0: Load

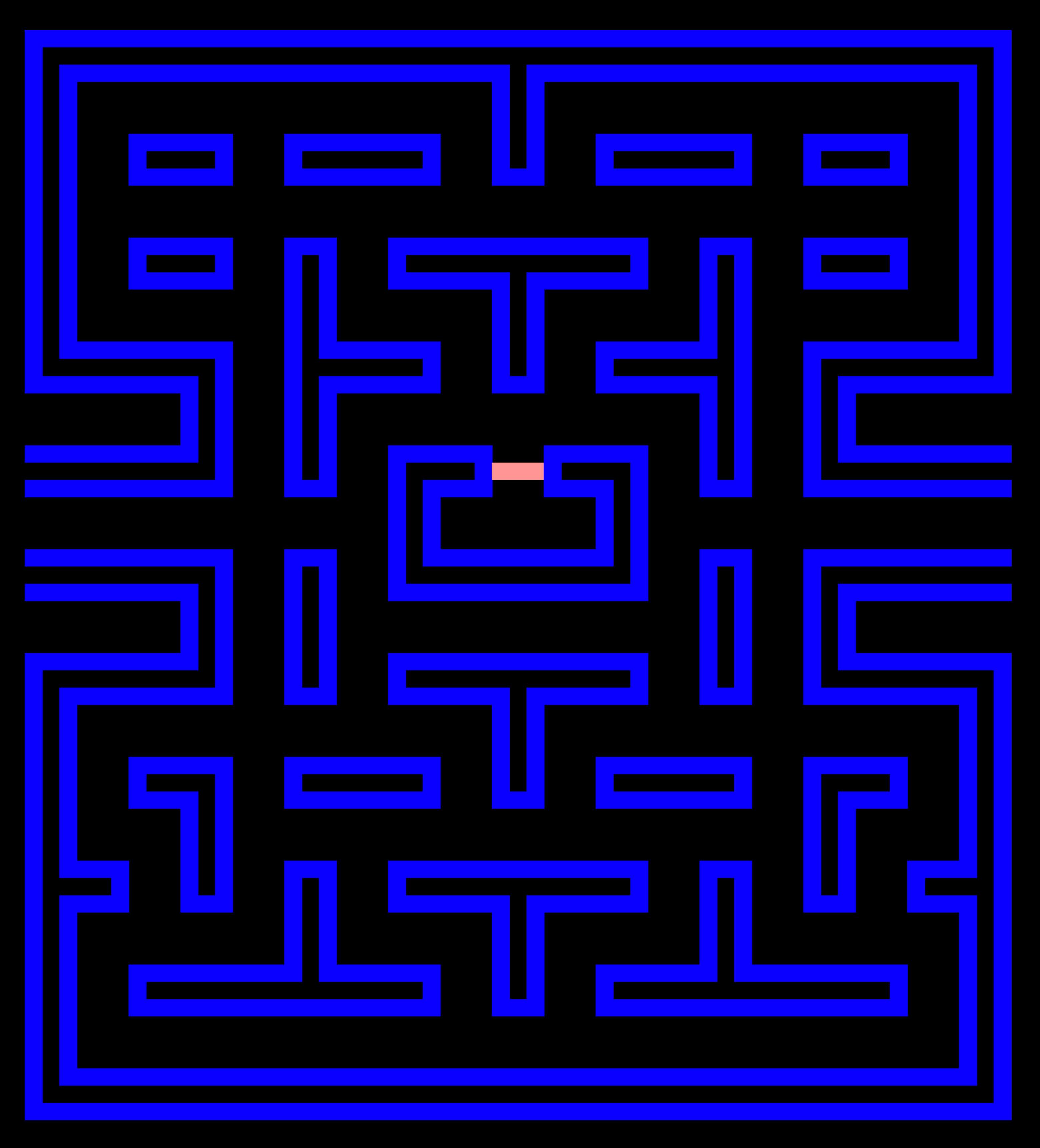
Step 0: Load

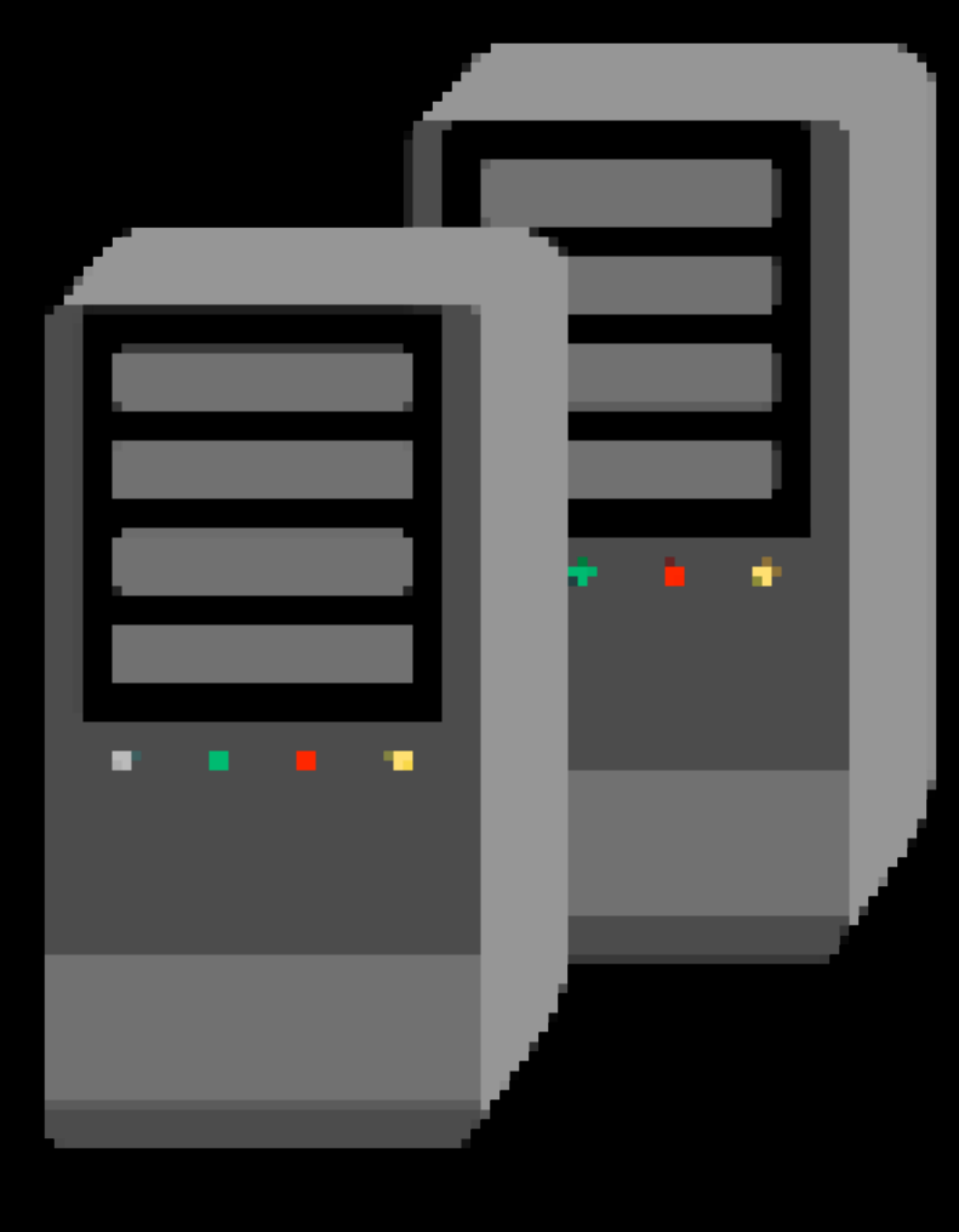


Step 0: Load

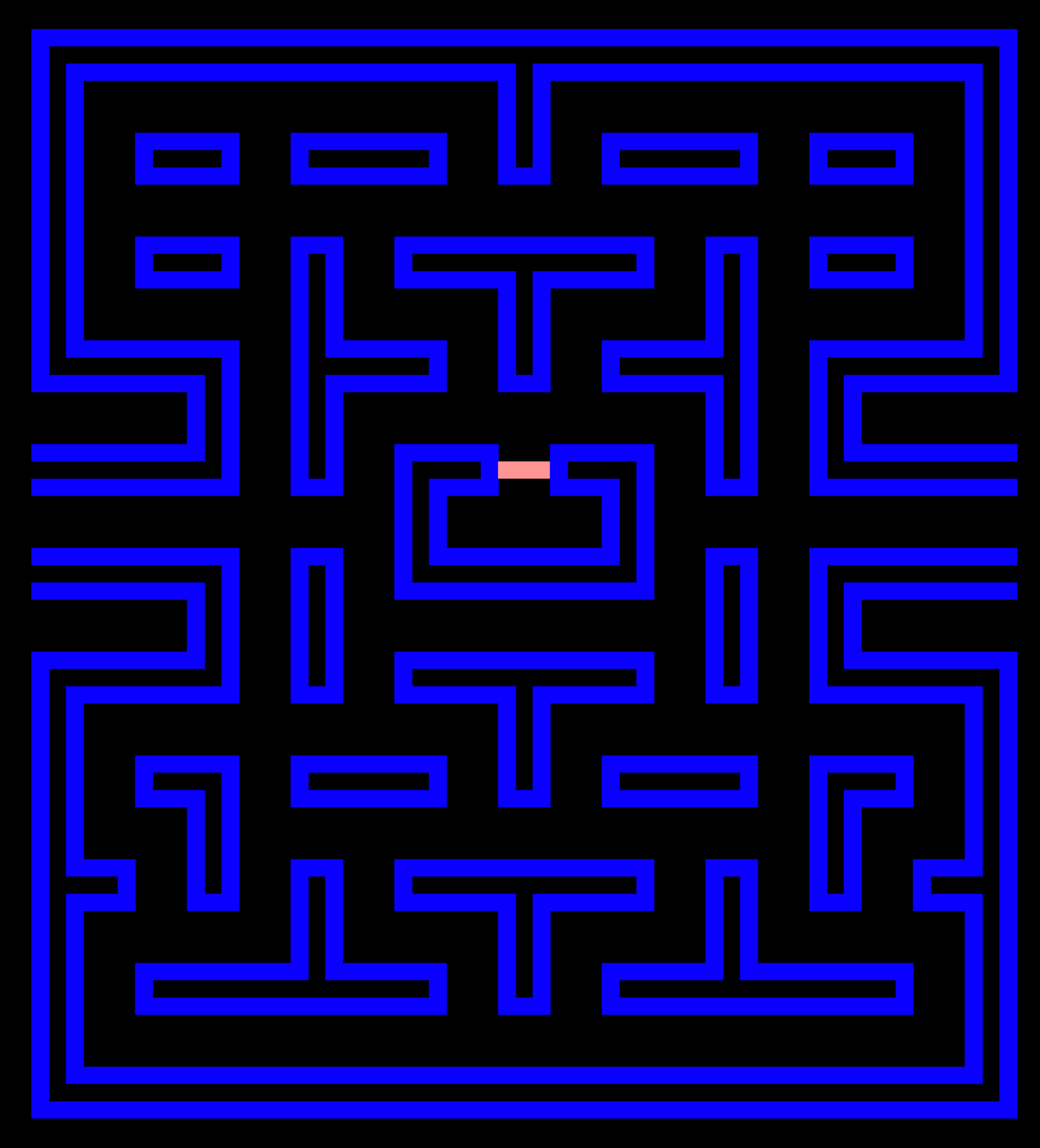


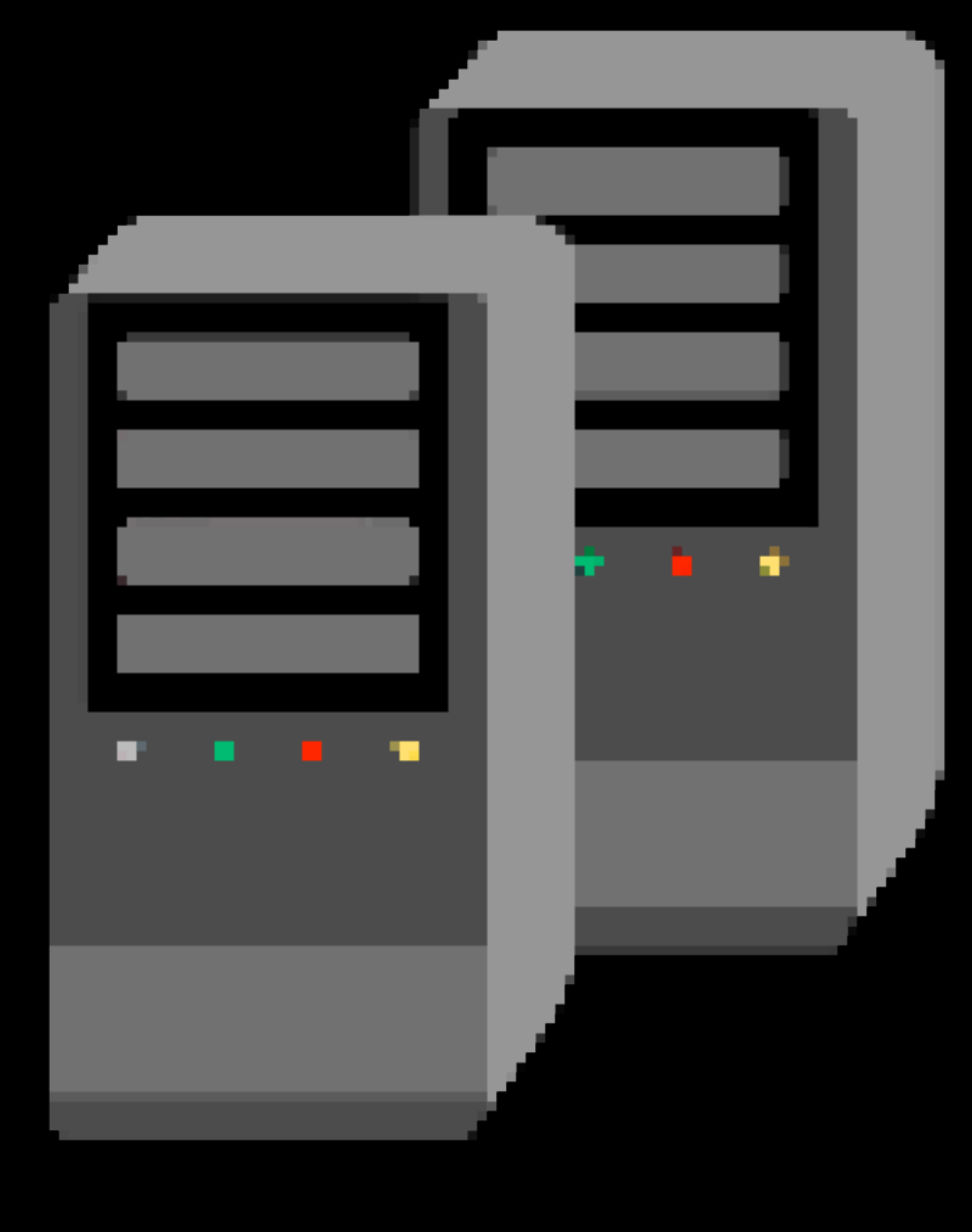
Step 0: Load



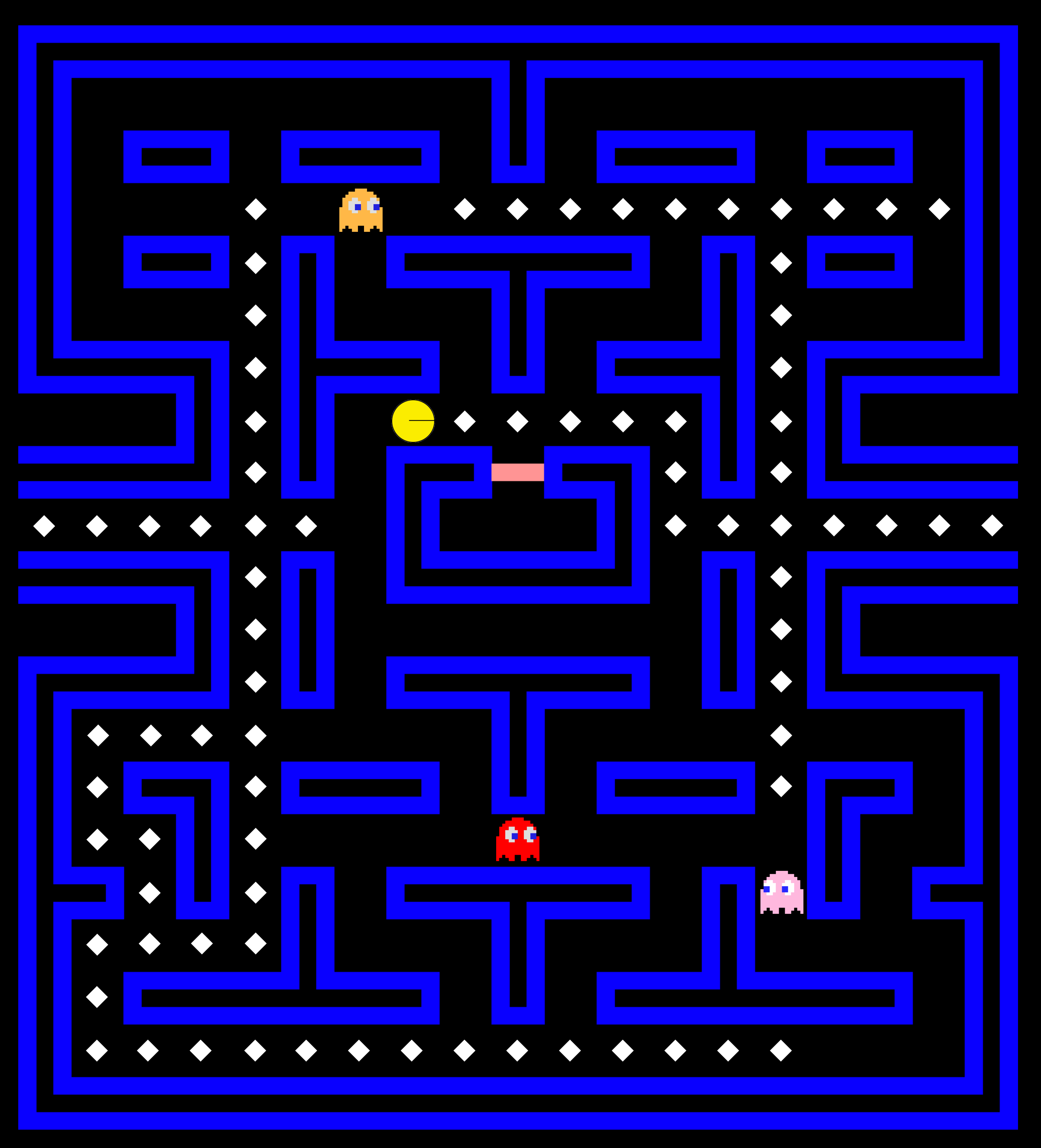


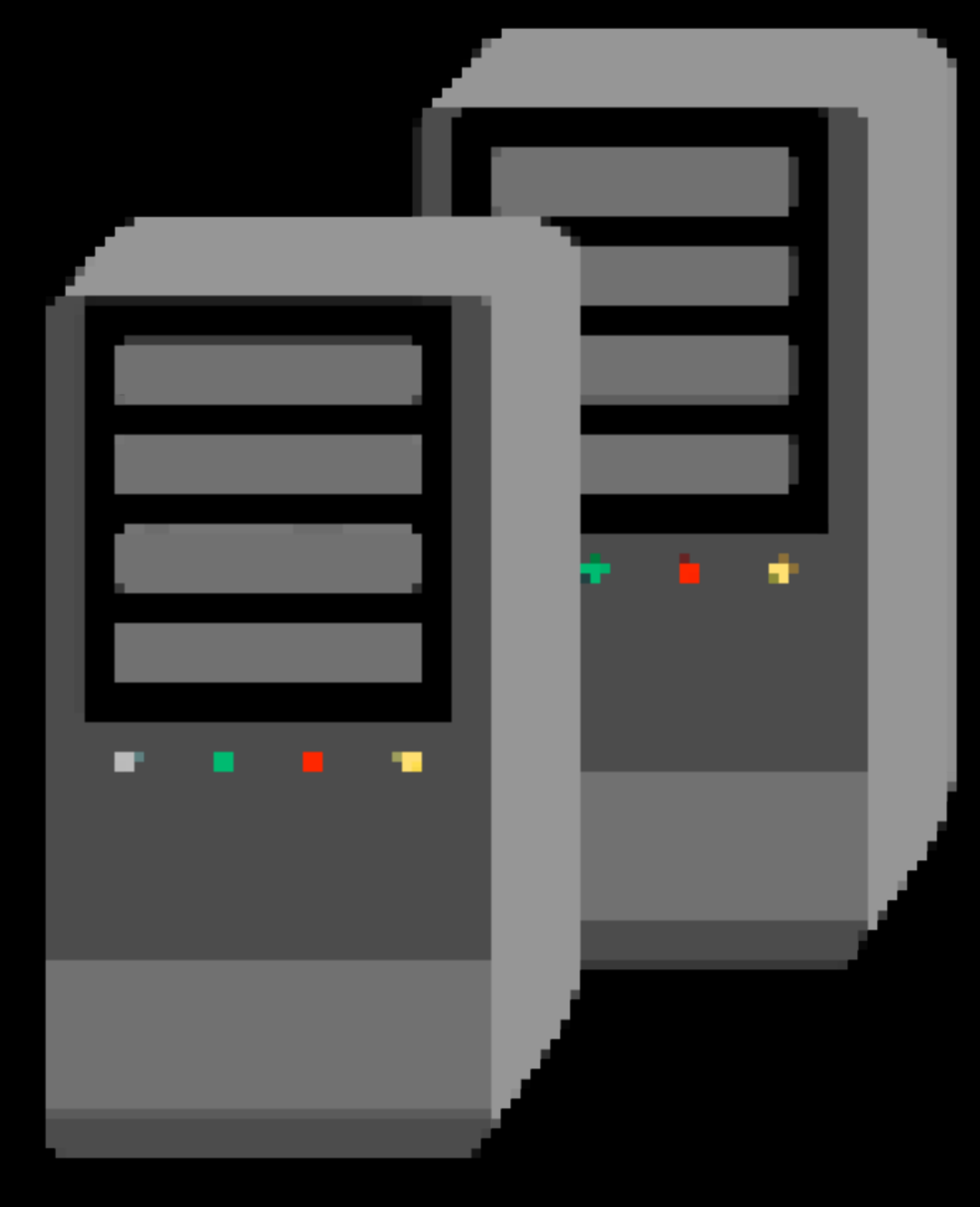
Step 1: Setup



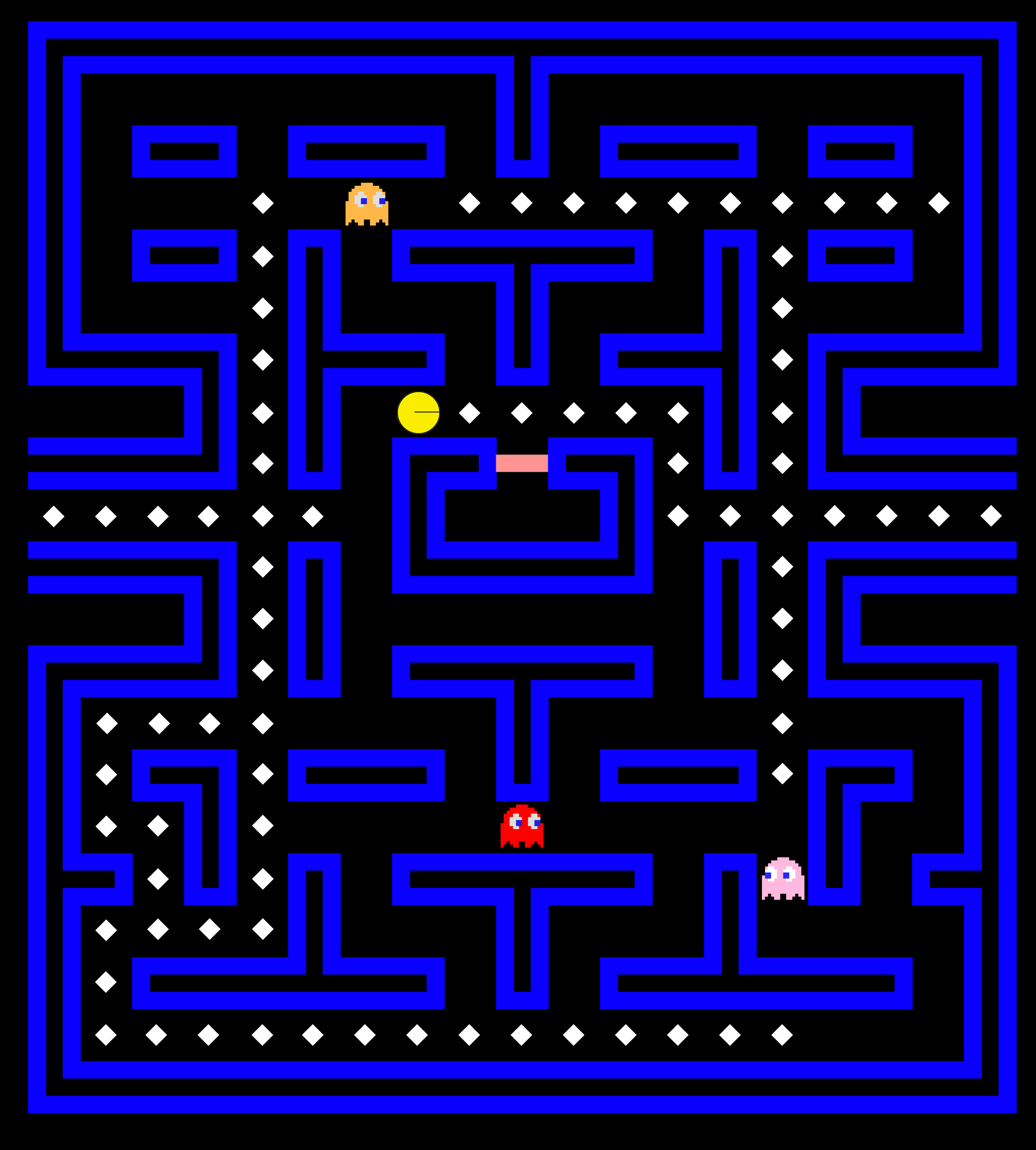


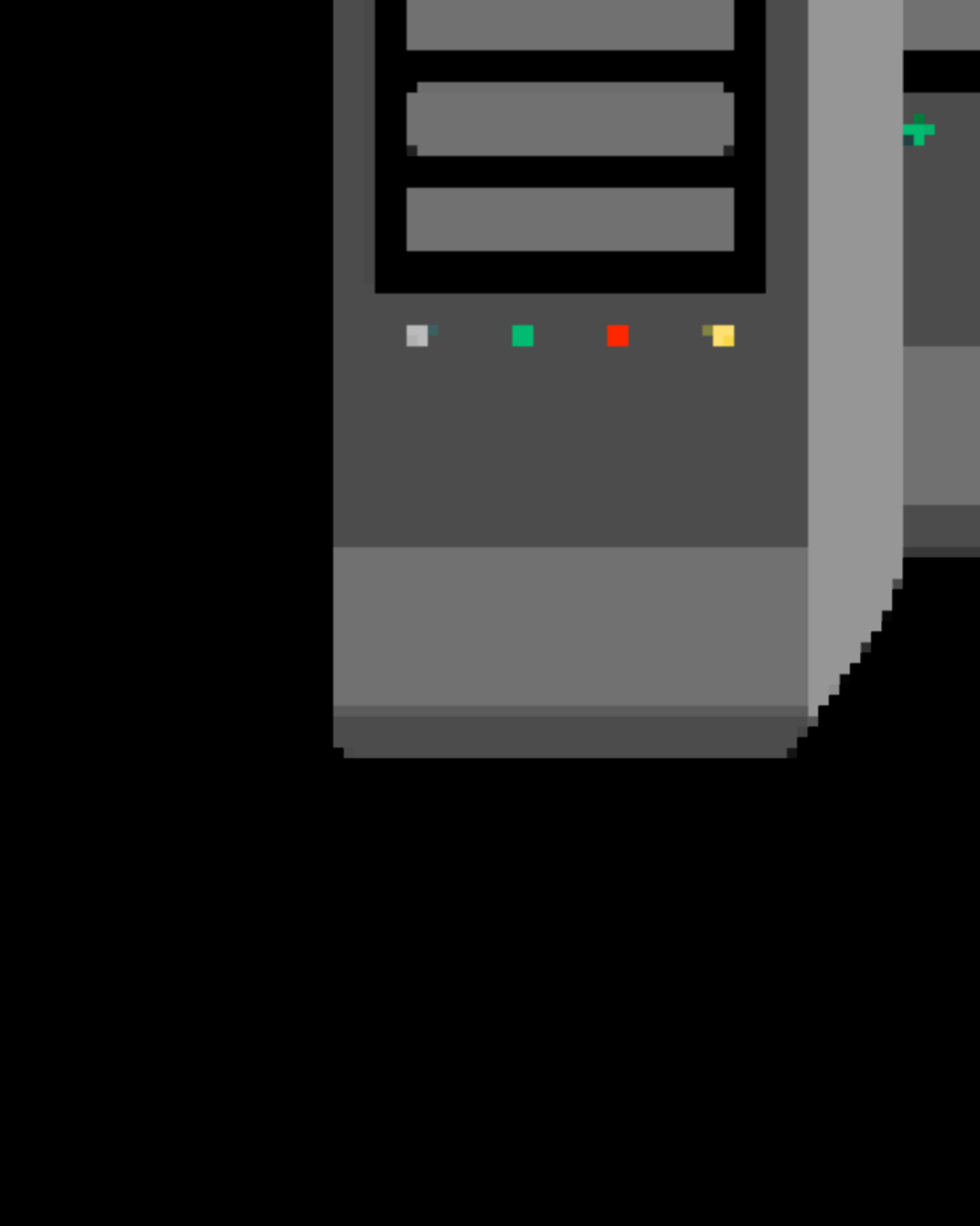
Step 1: Setup



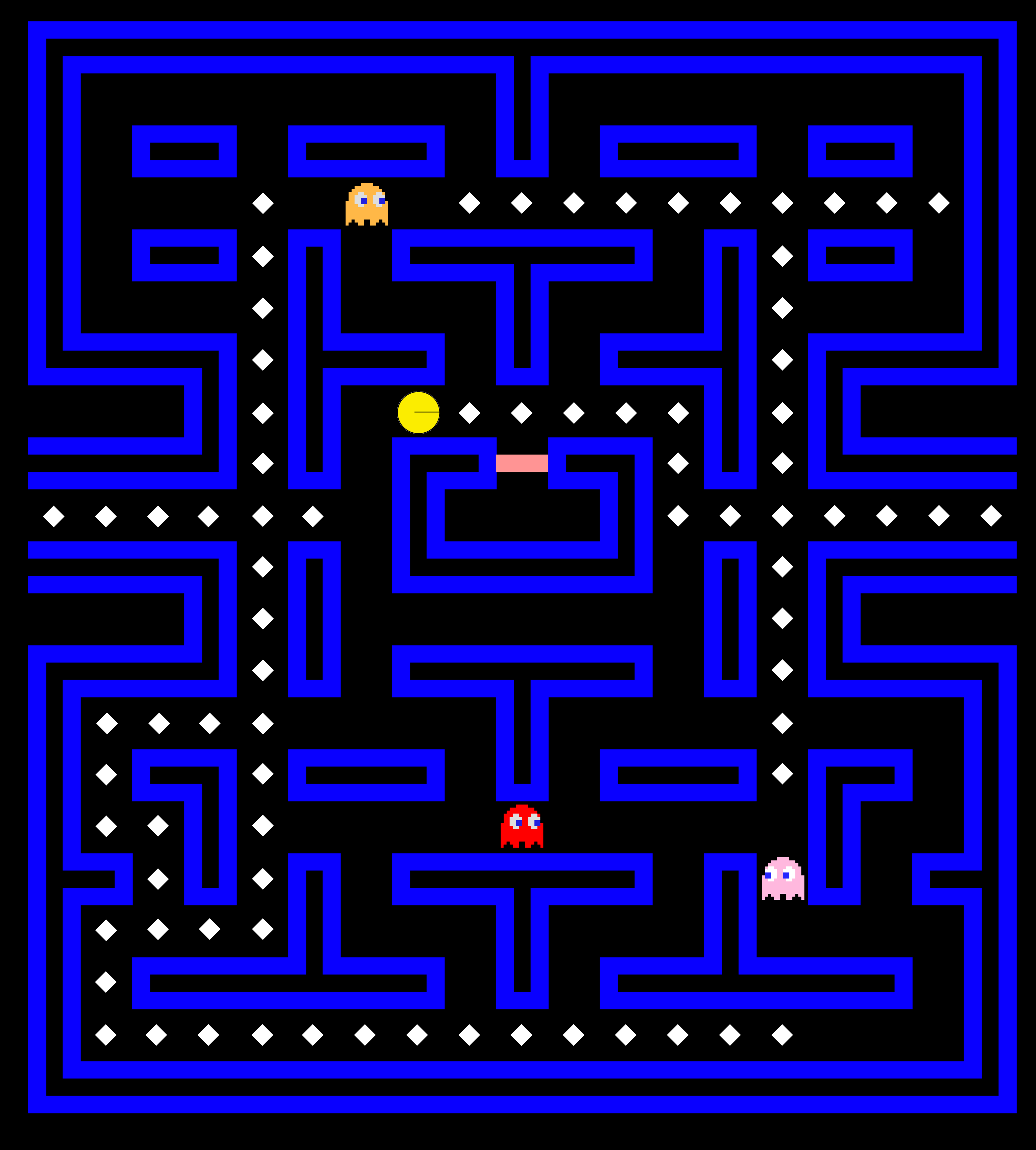


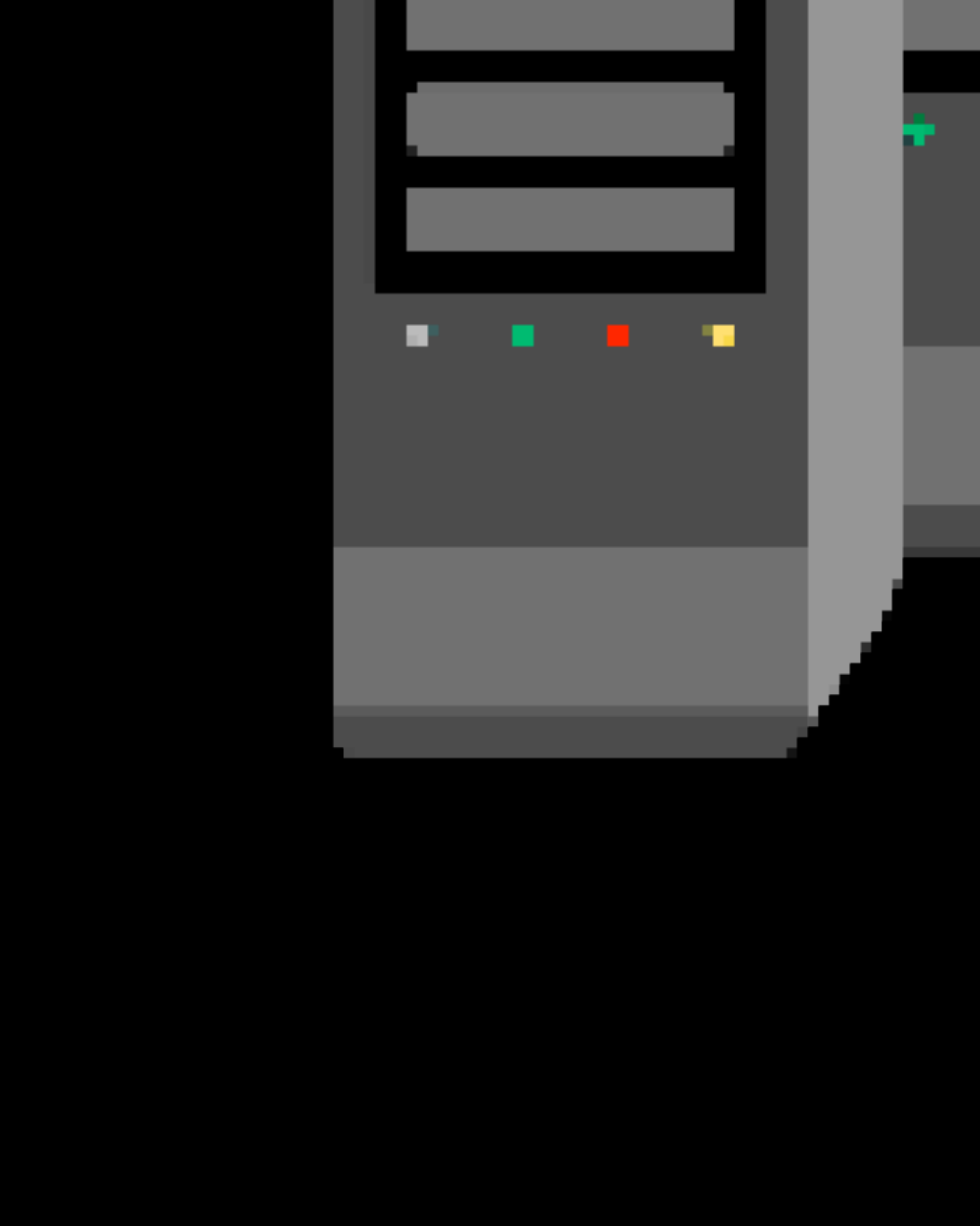
Step 2: Location



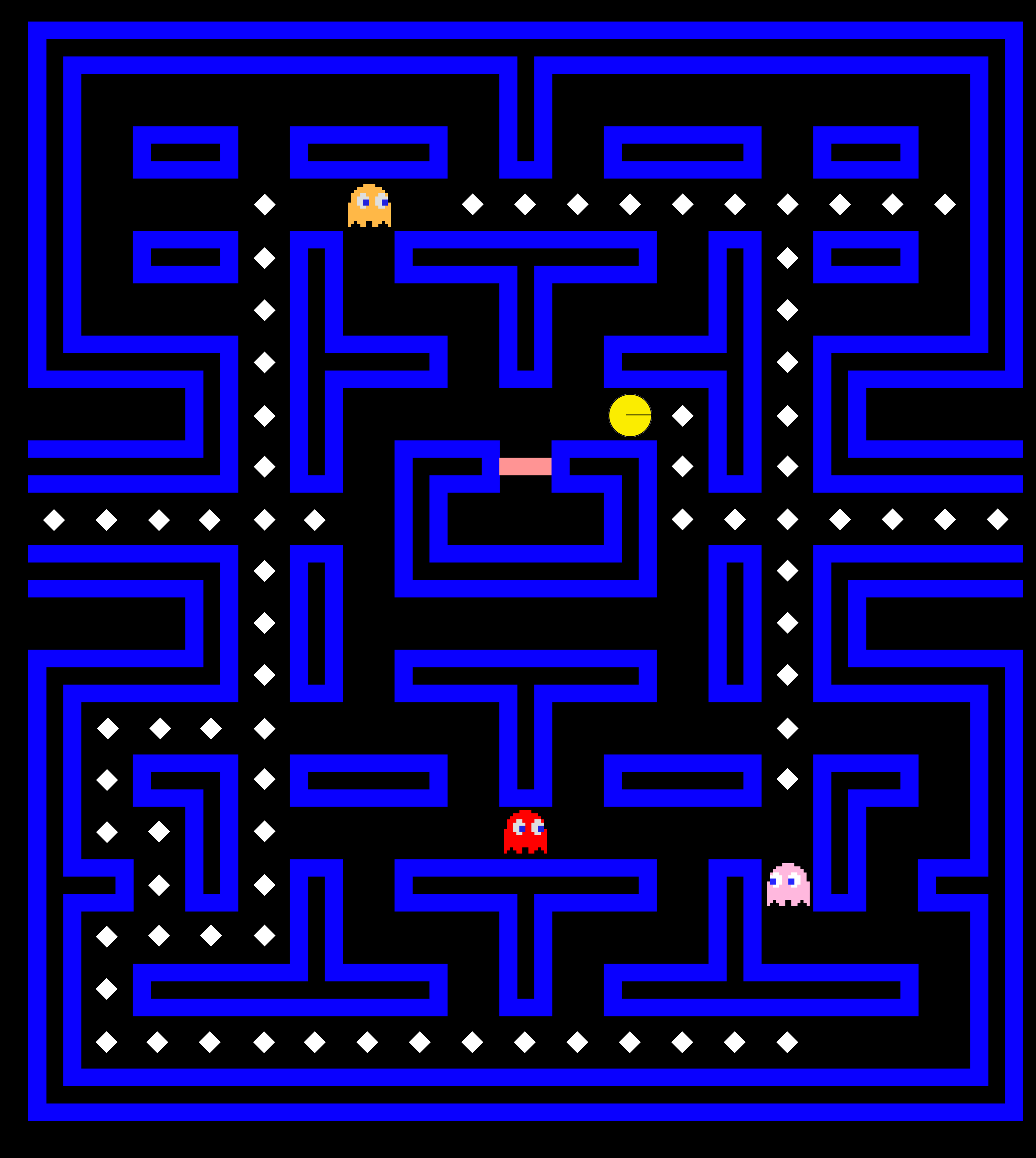


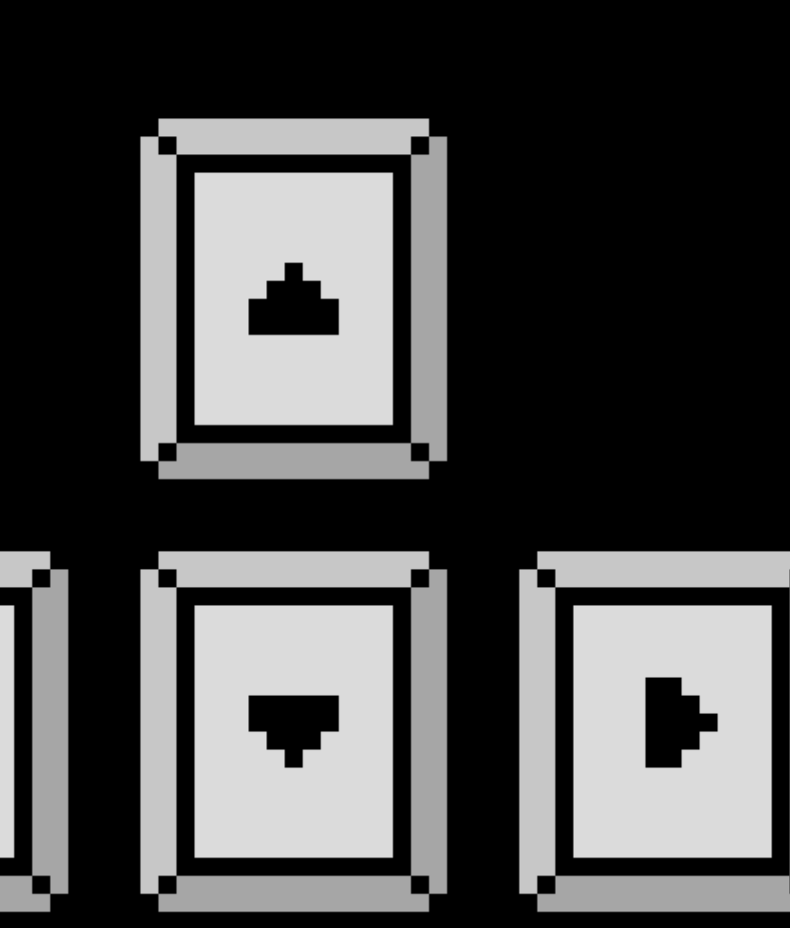
Step 2: Location

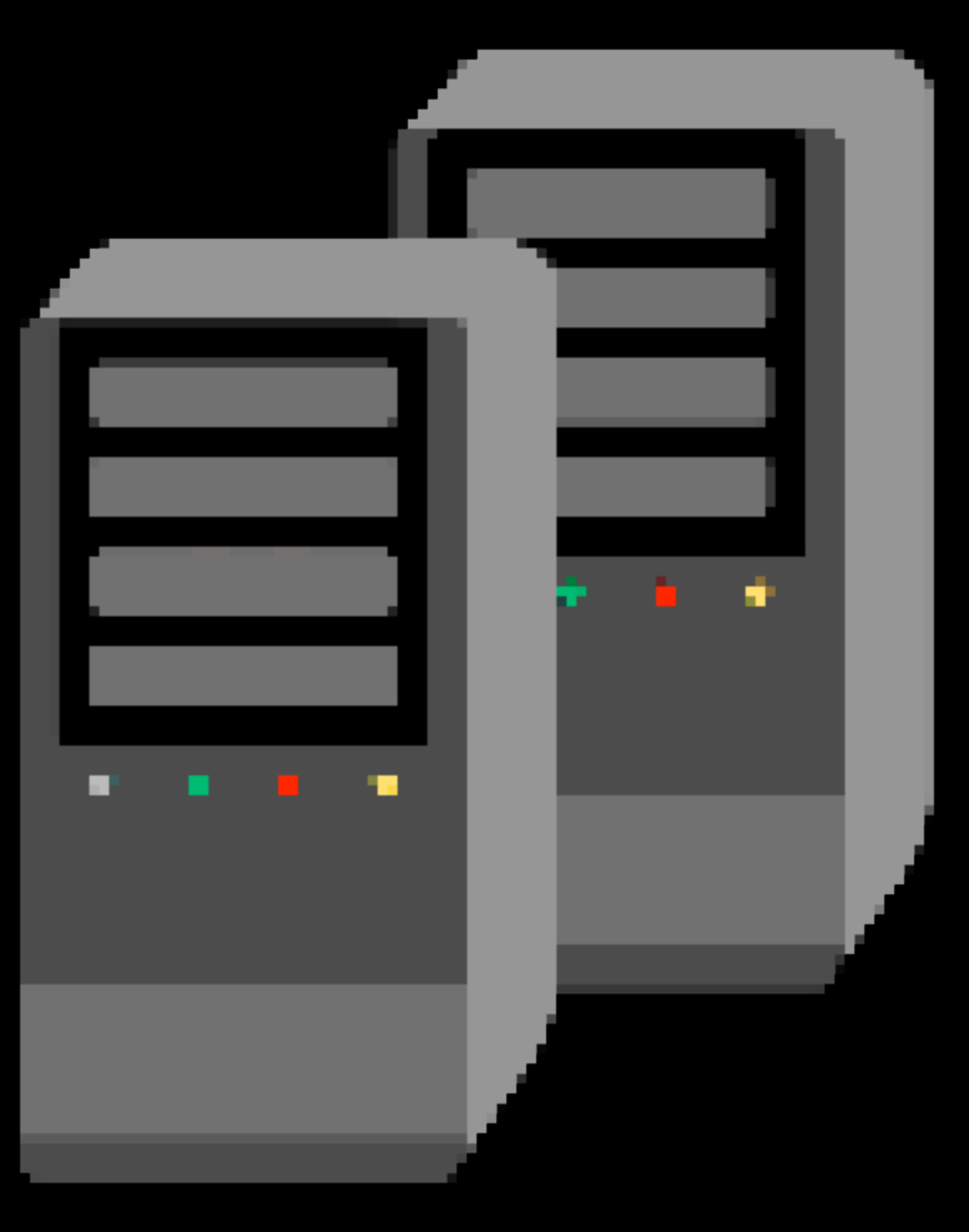




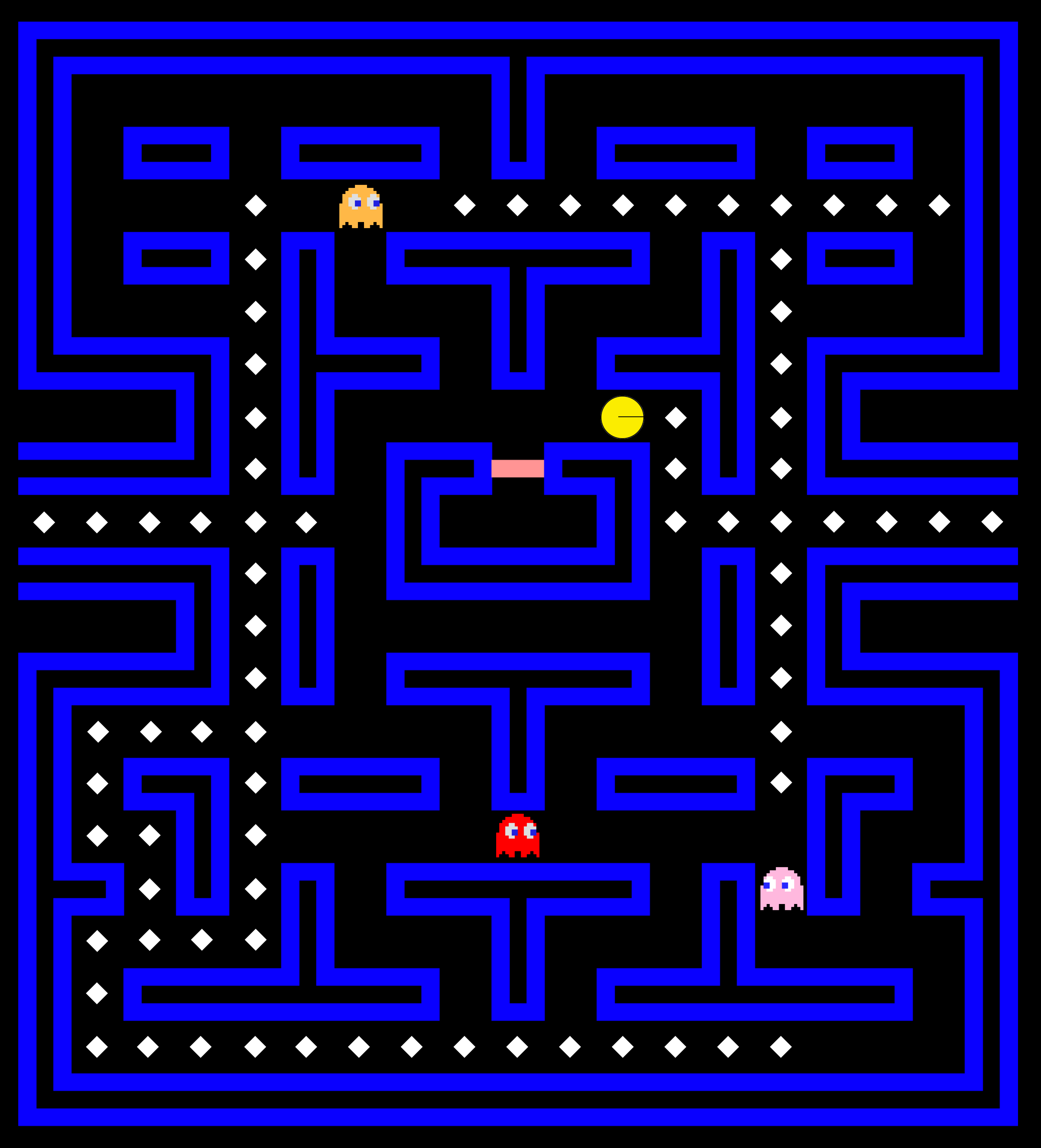
Step 2: Location

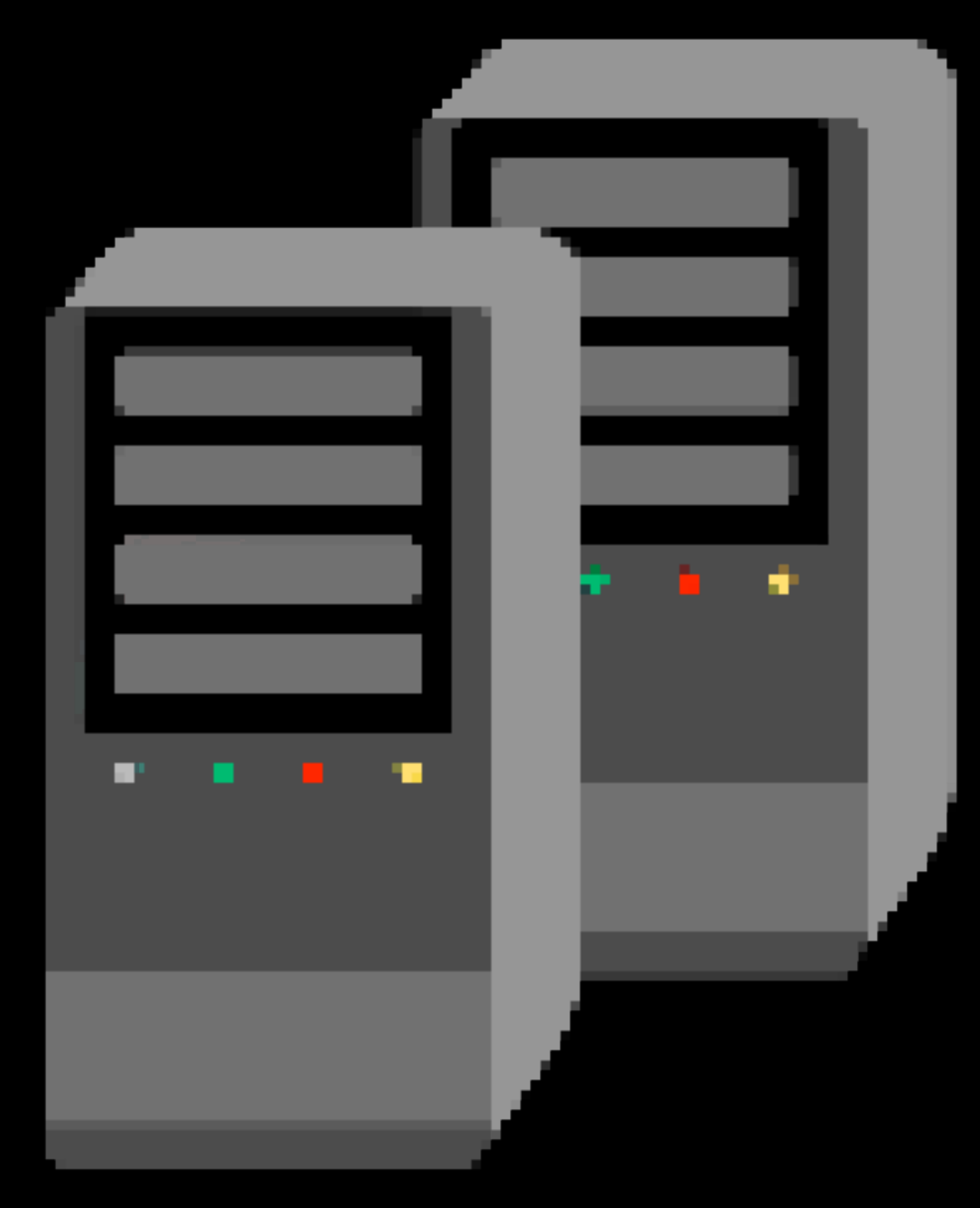




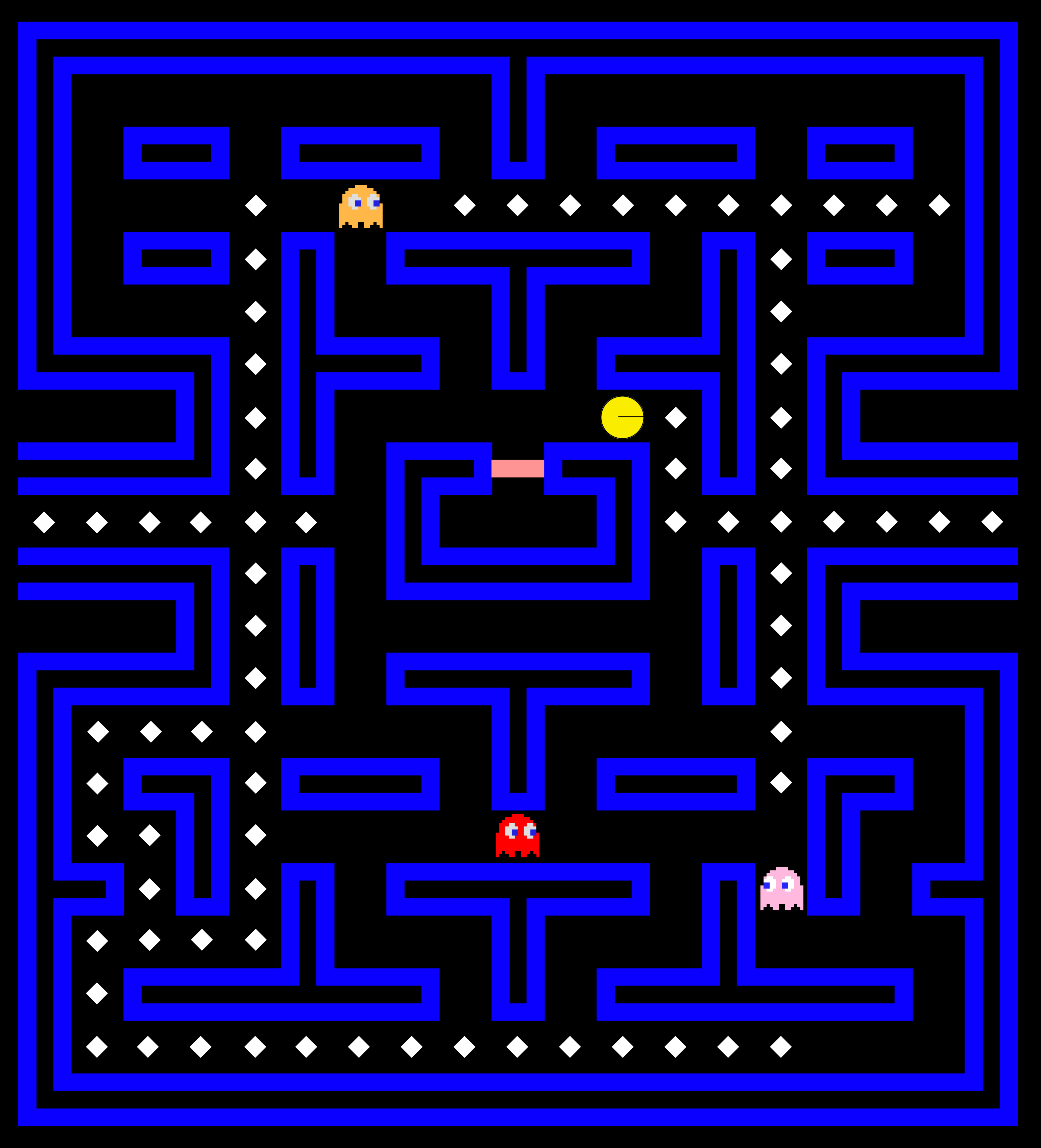


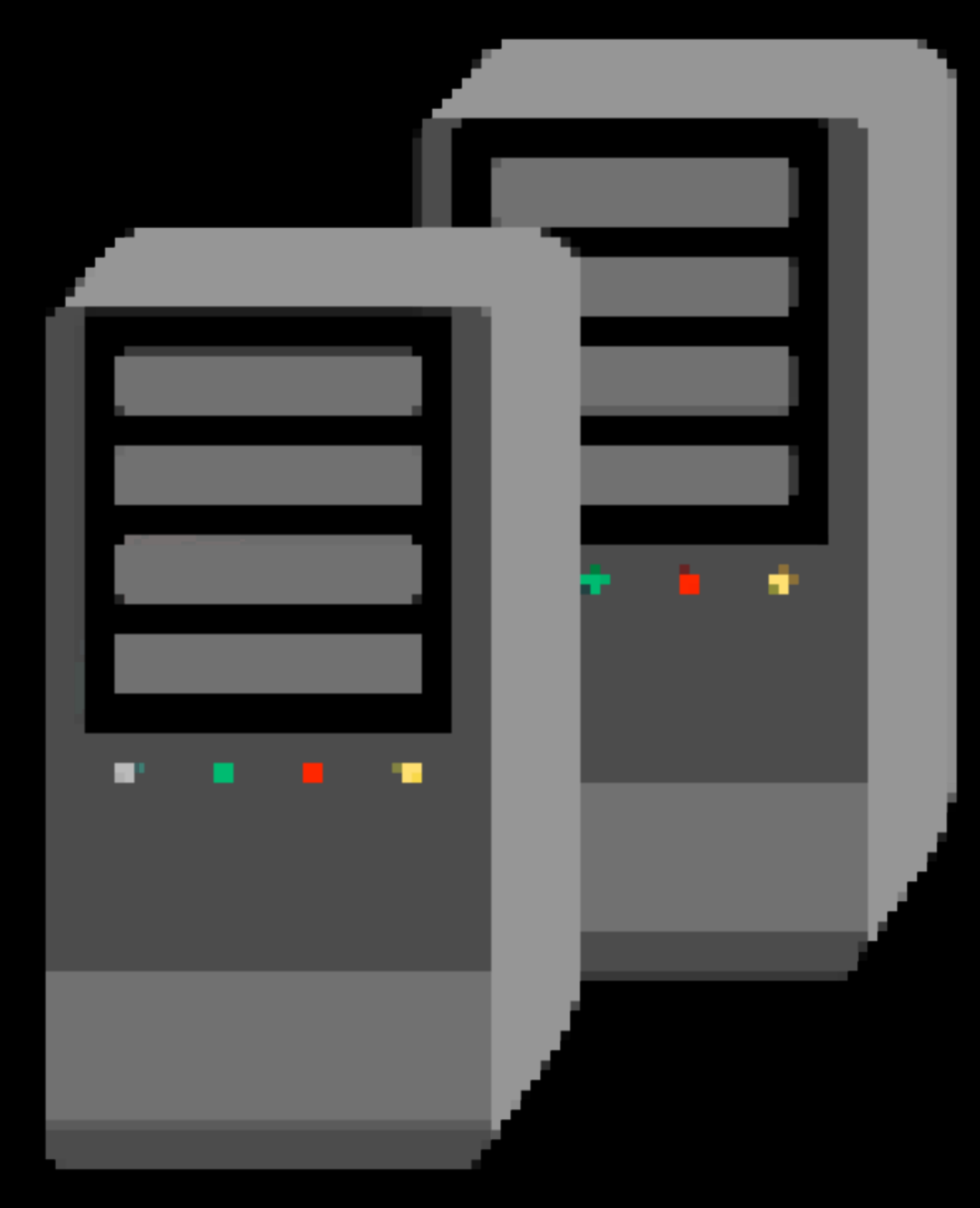
Step 3: Updates



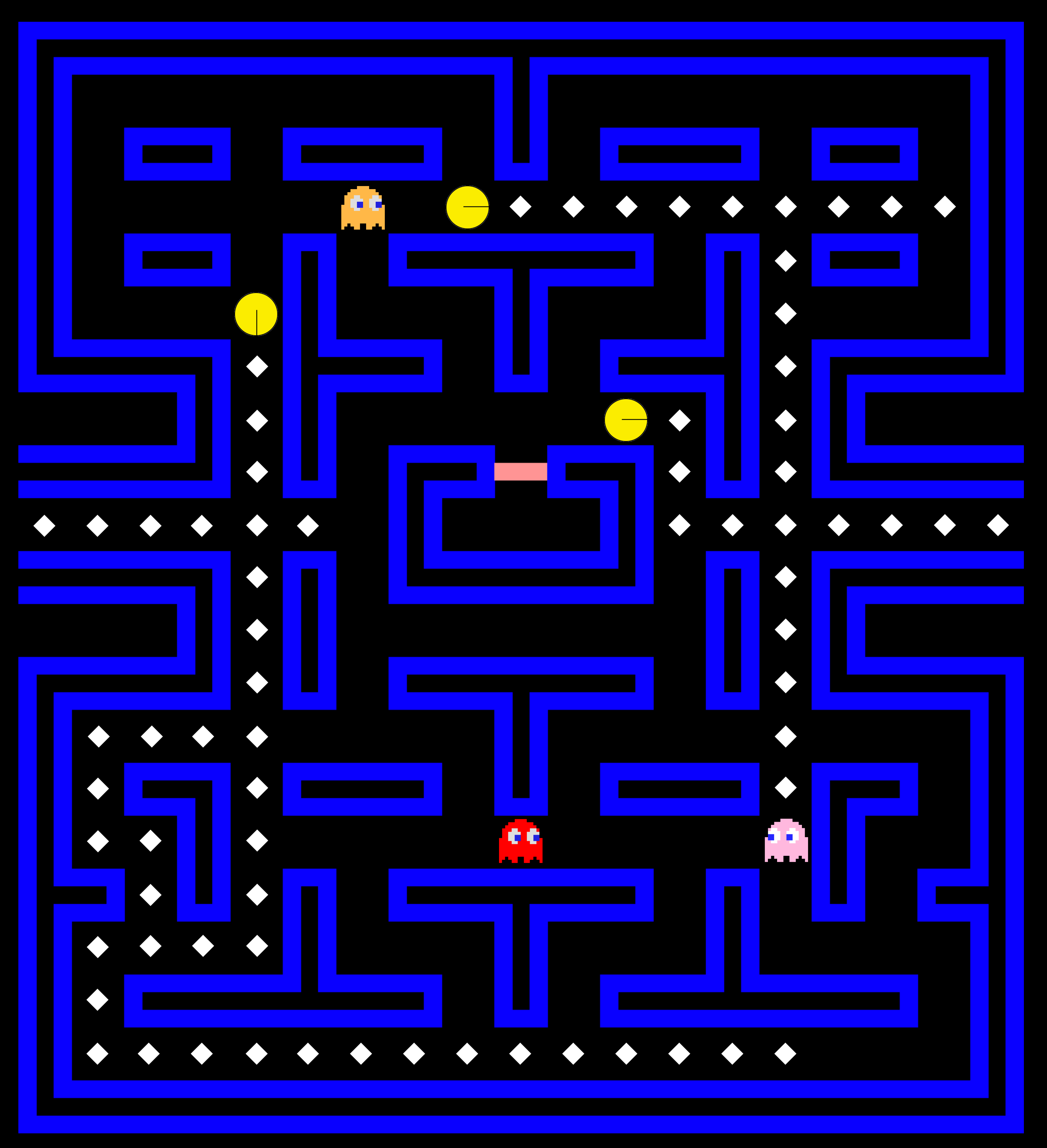


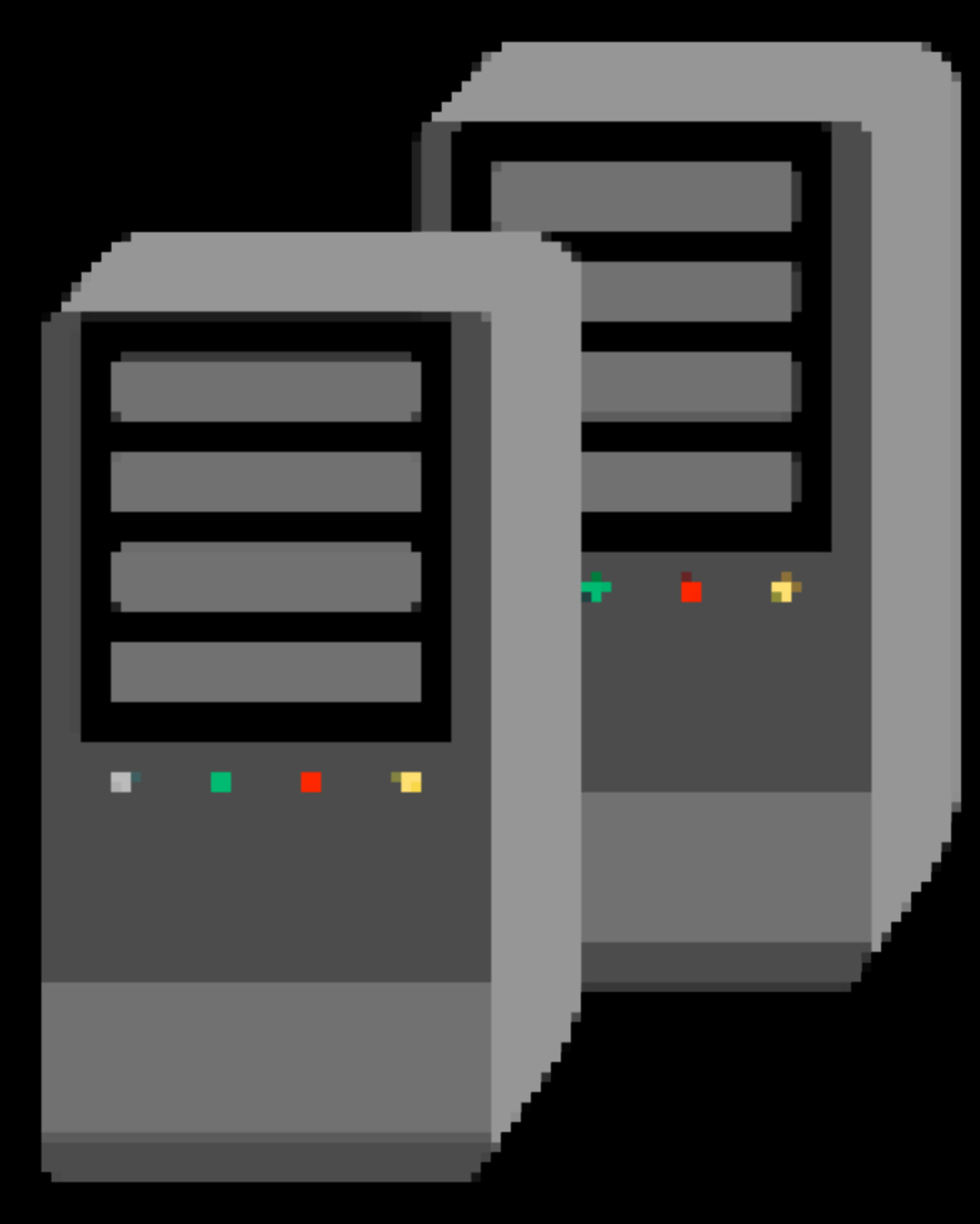
Step 3: Updates



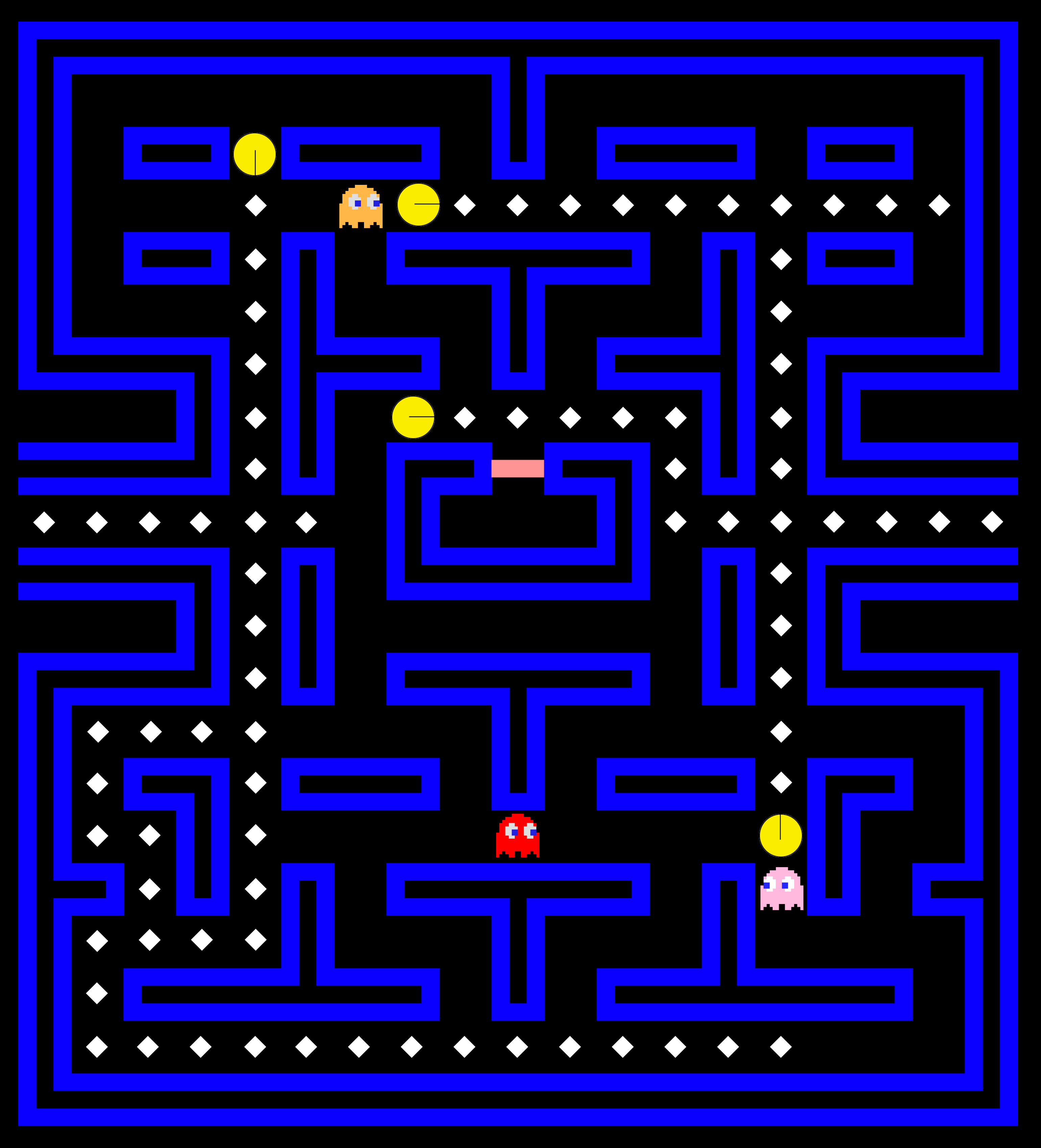


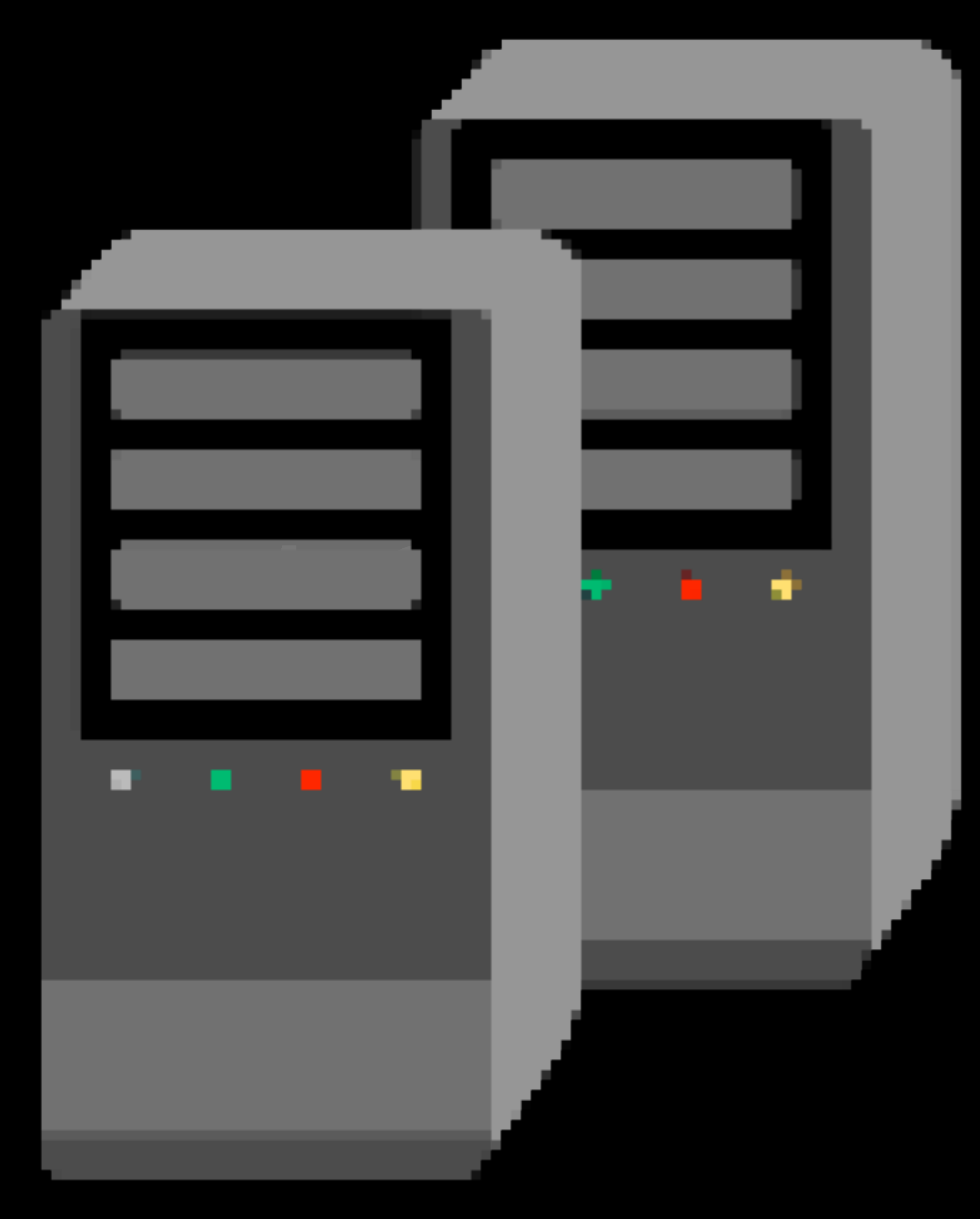
Step 3: Updates



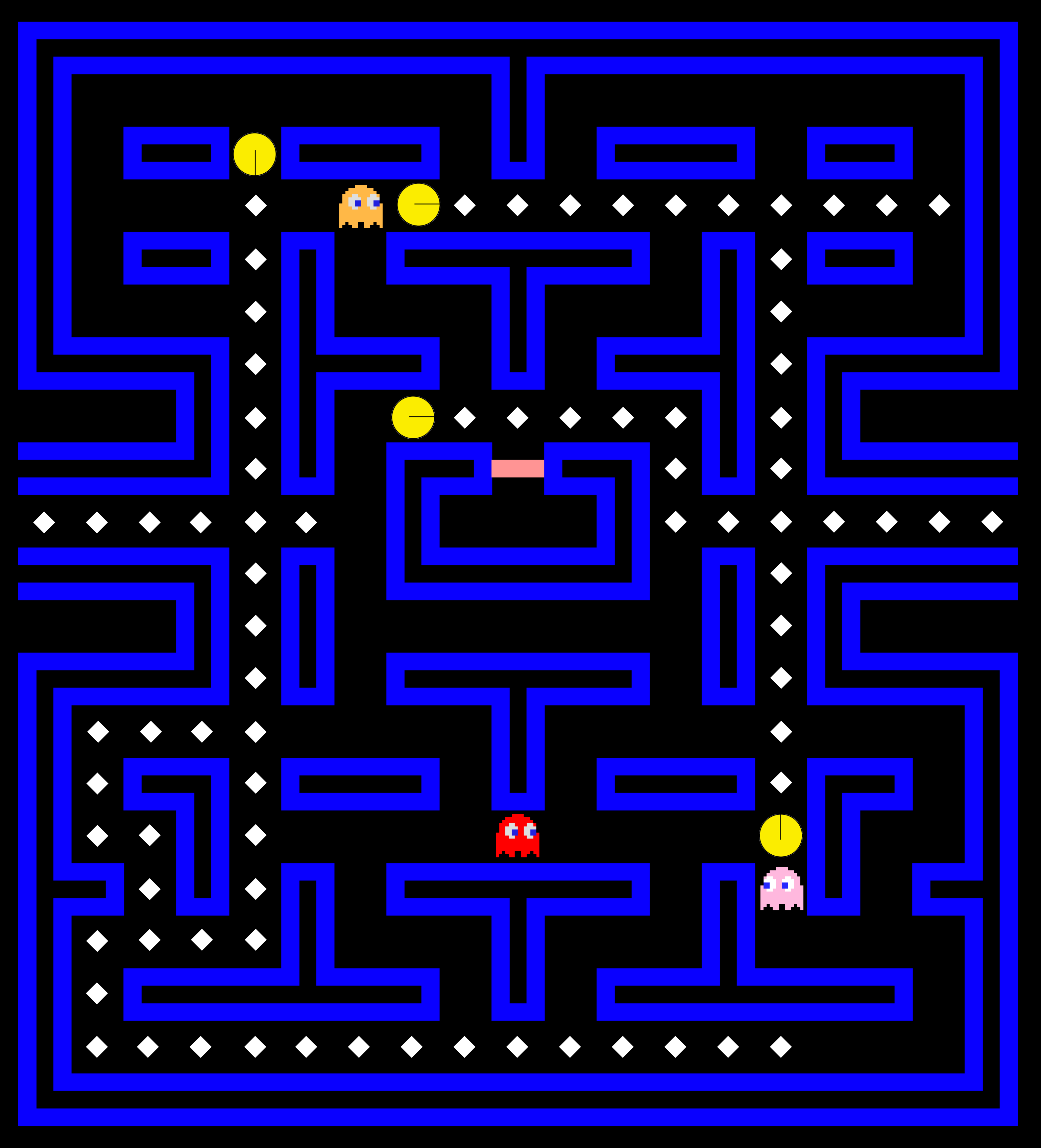


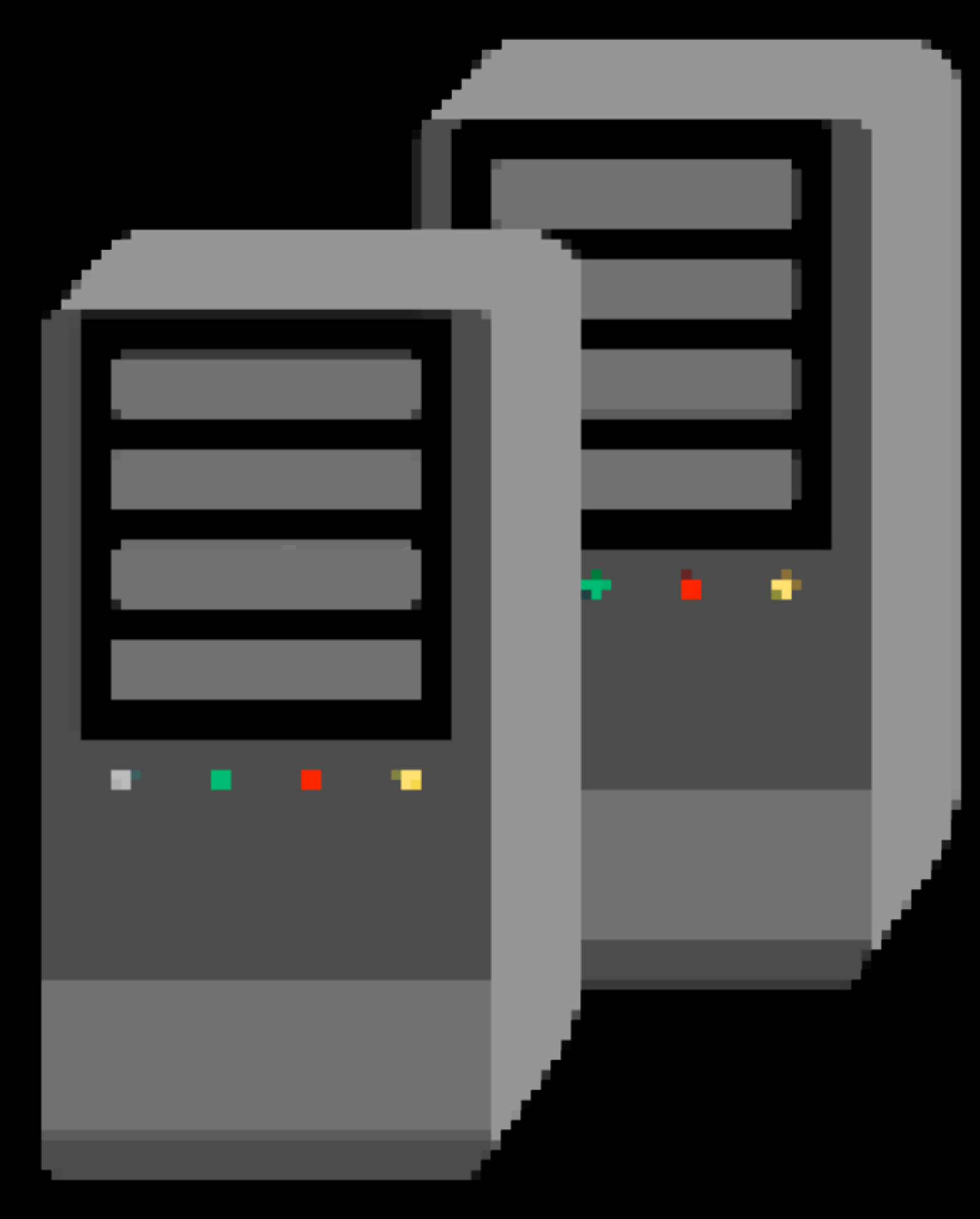
Step 3: Updates



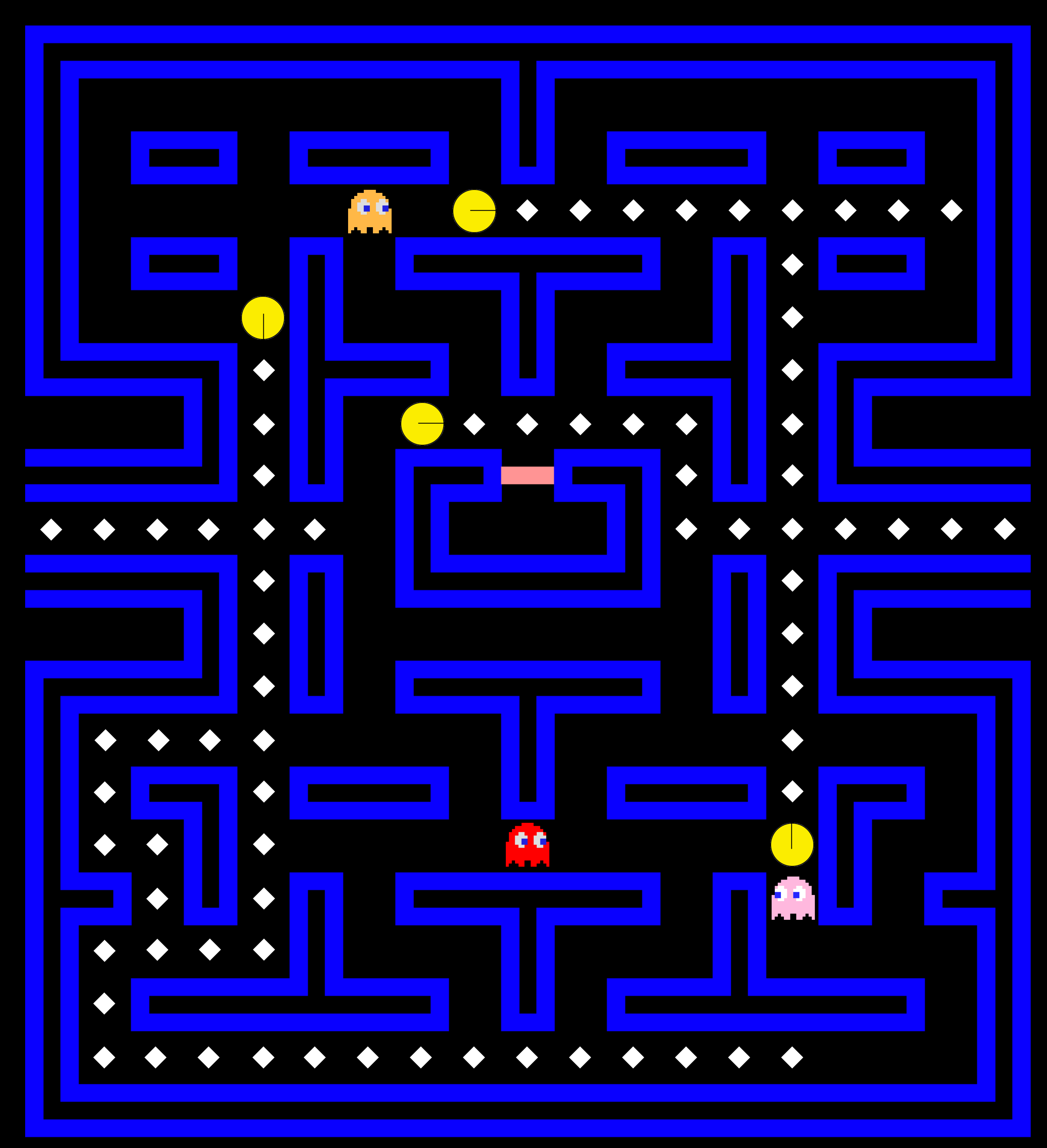


Step 3: Updates



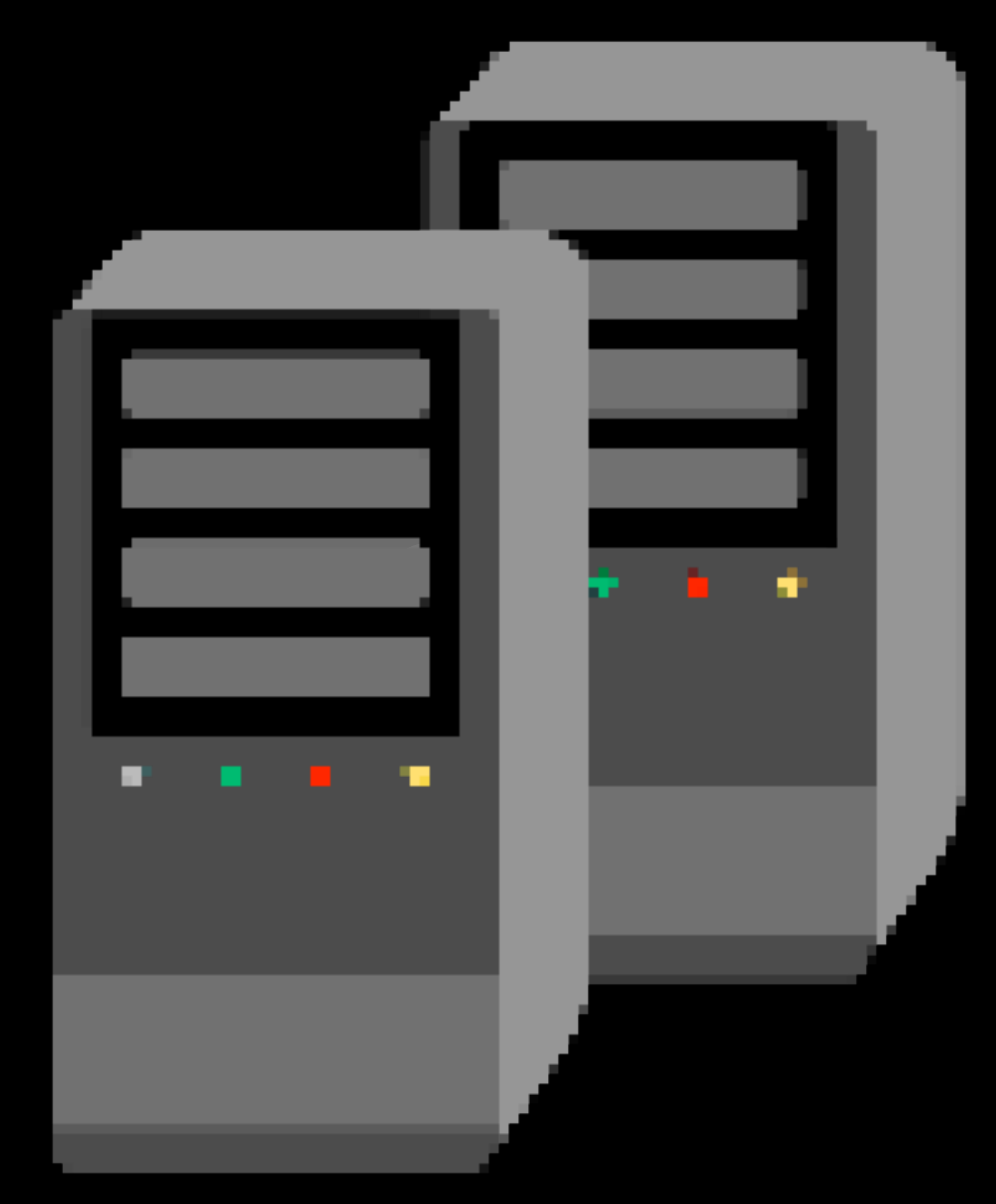


Step 3: Updates

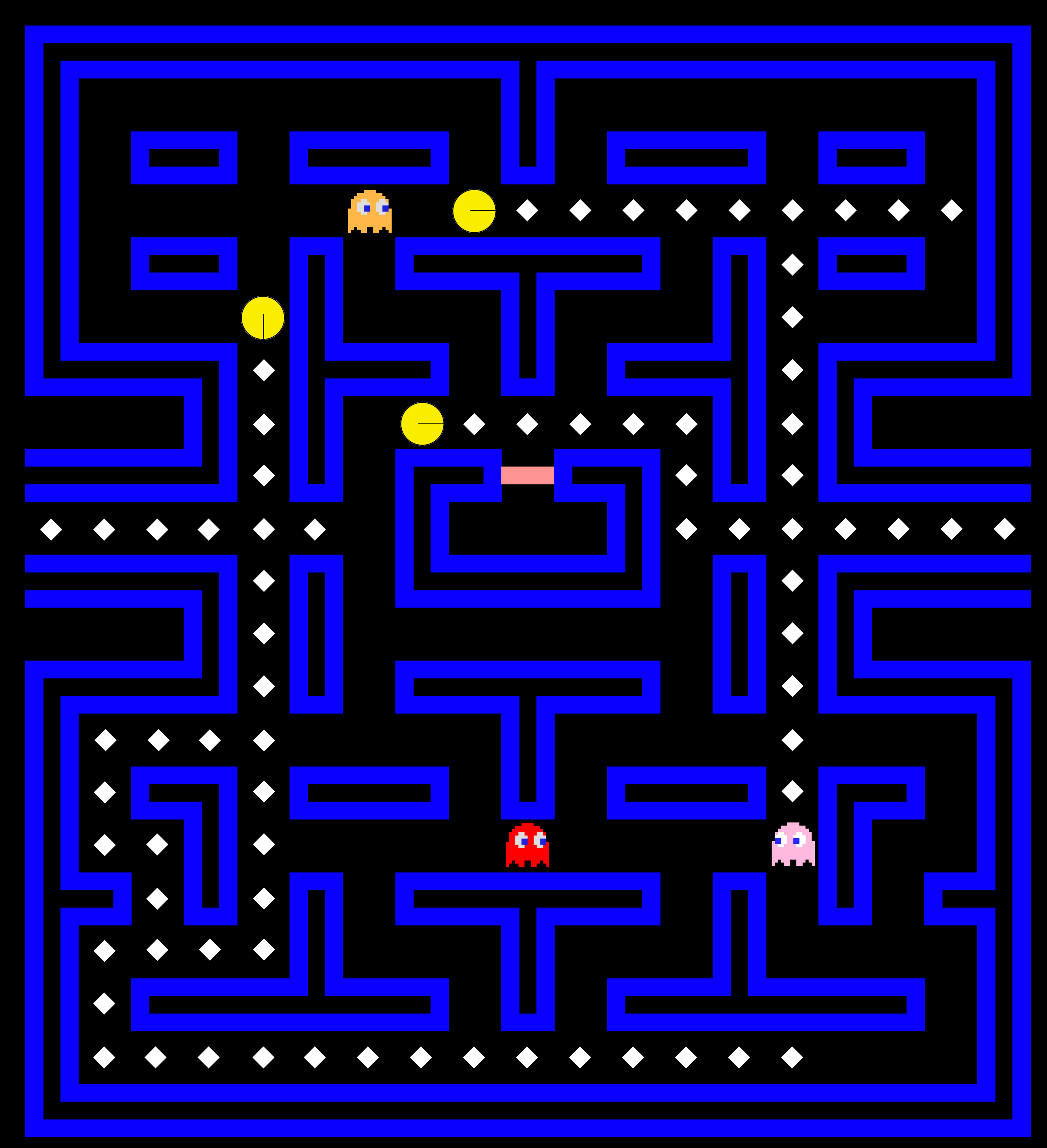


Scoreboard

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

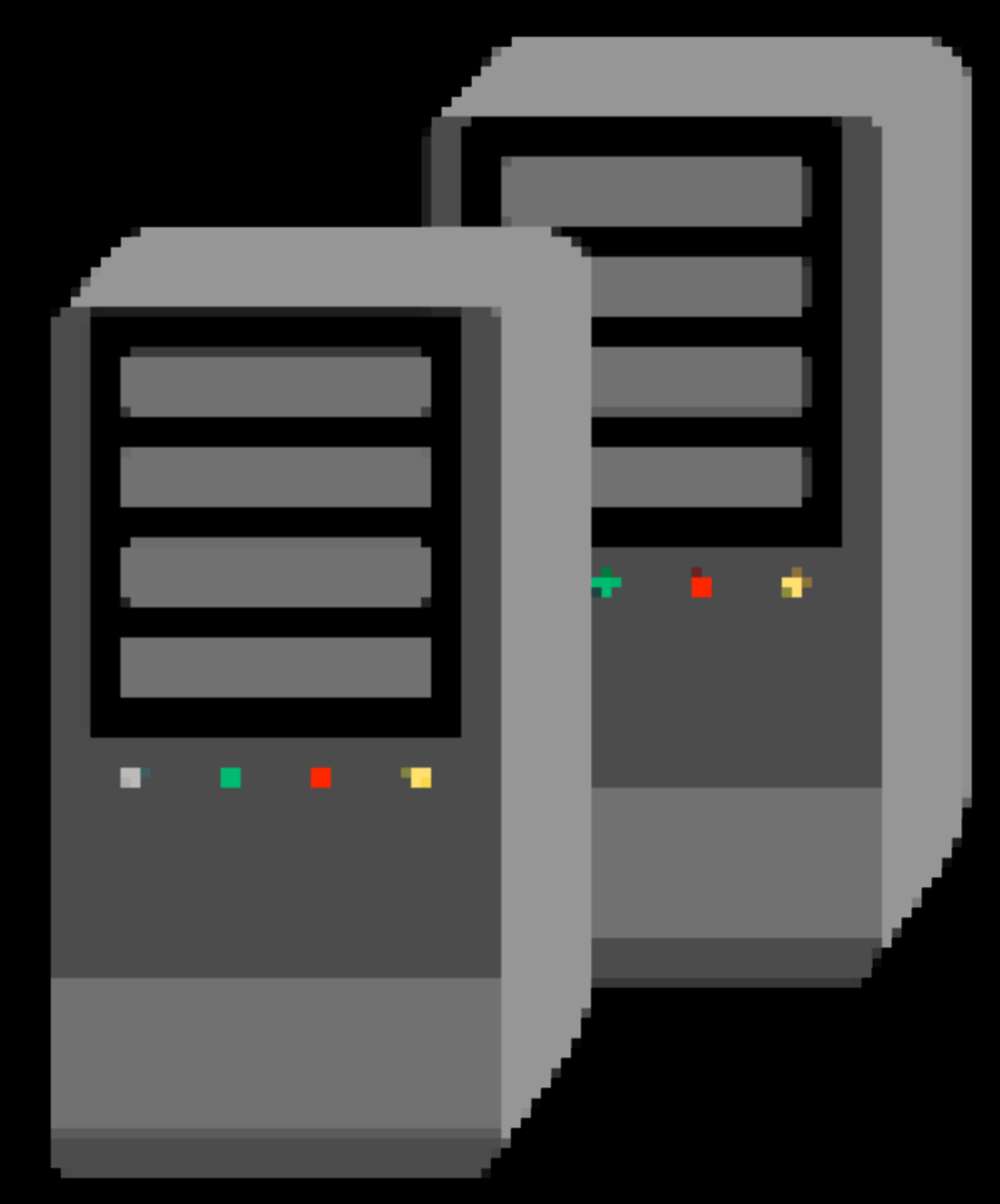


Step 3: Updates



Scoreboard

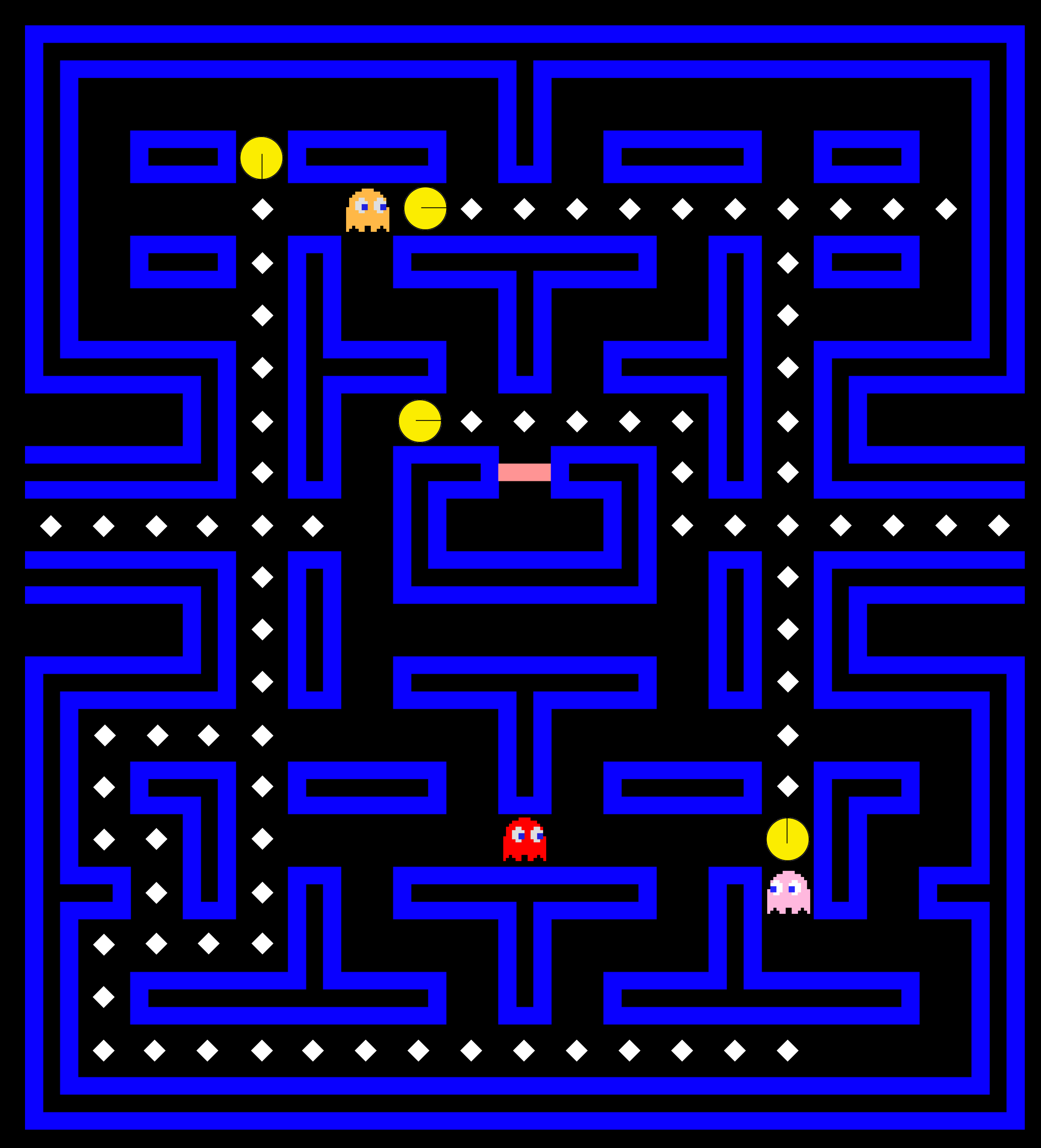
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7

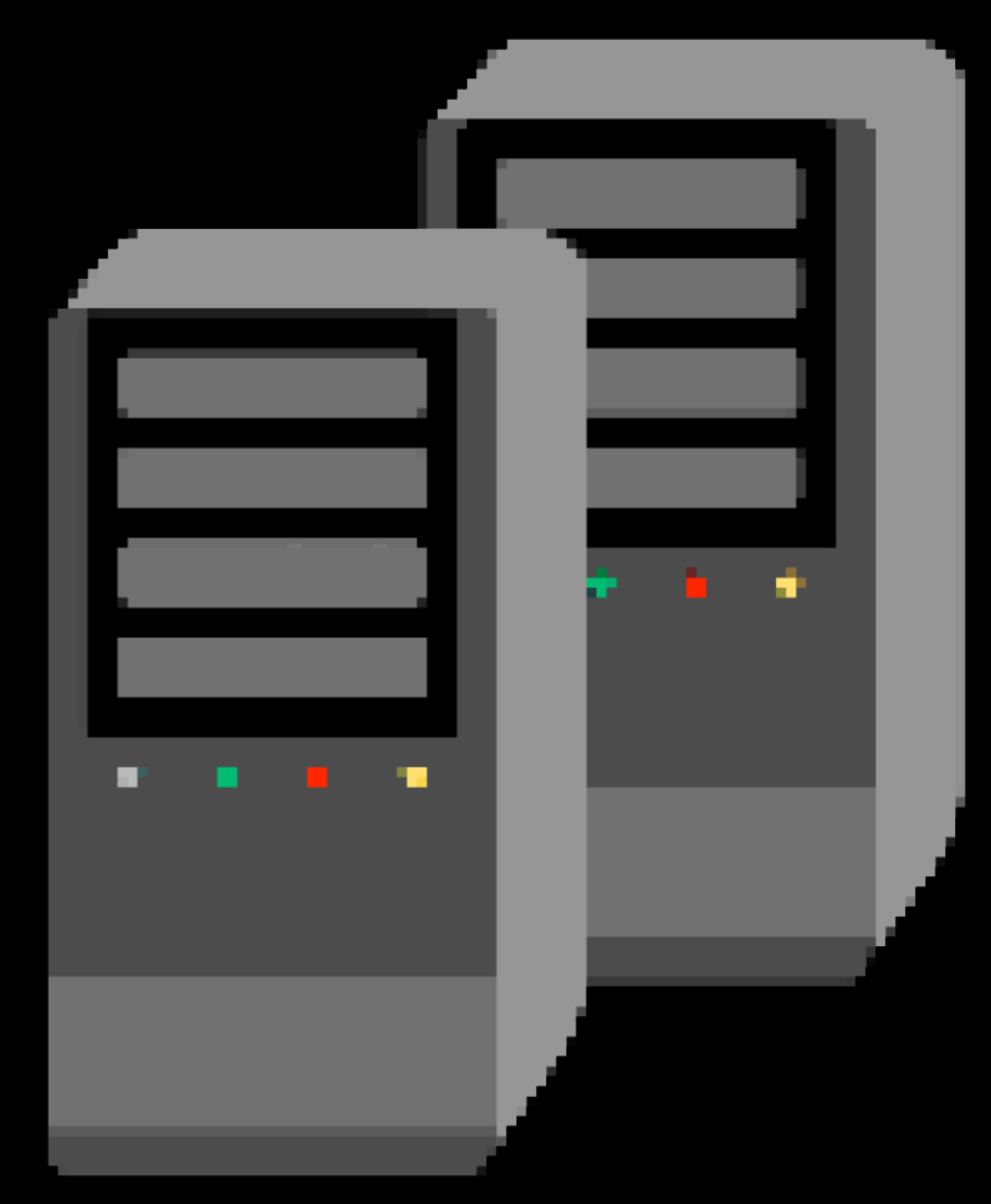


ATTENTION!

This is still about microservices

Real Enterprise

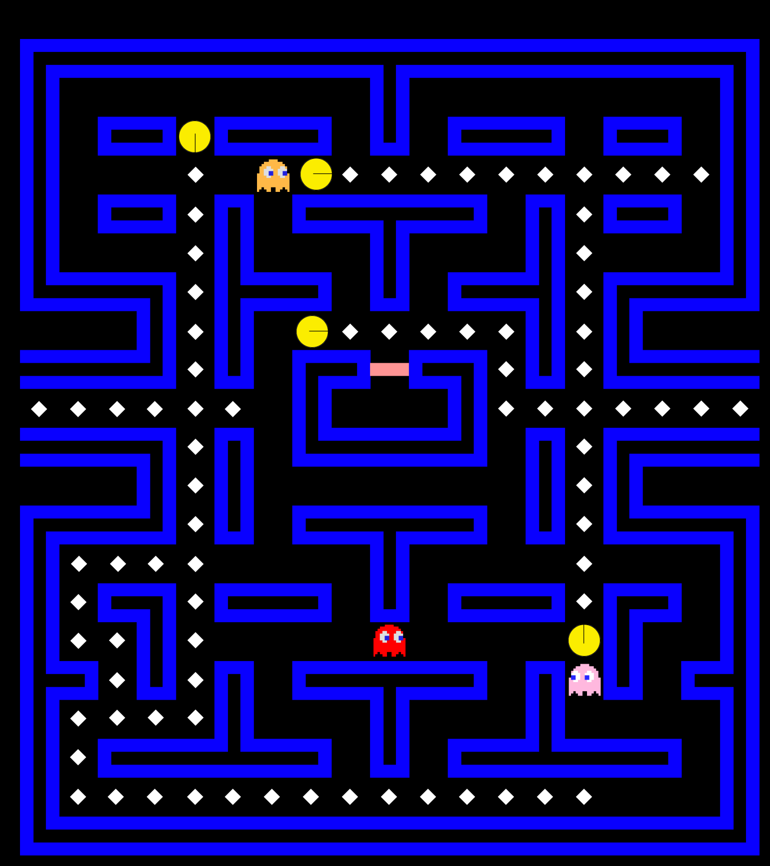




Scoreboard

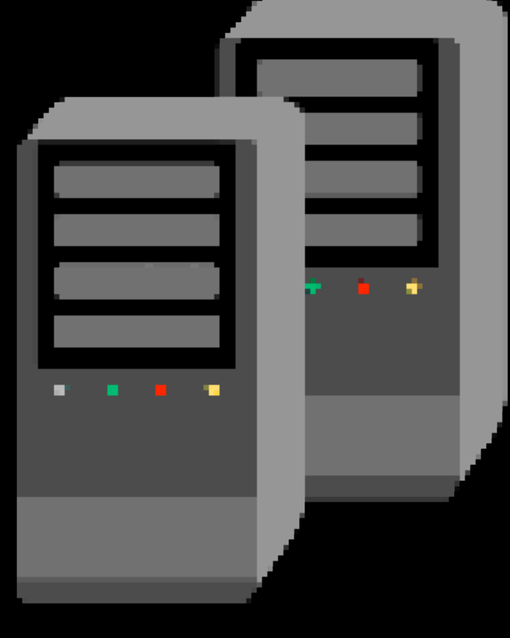
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7

Real Enterprise

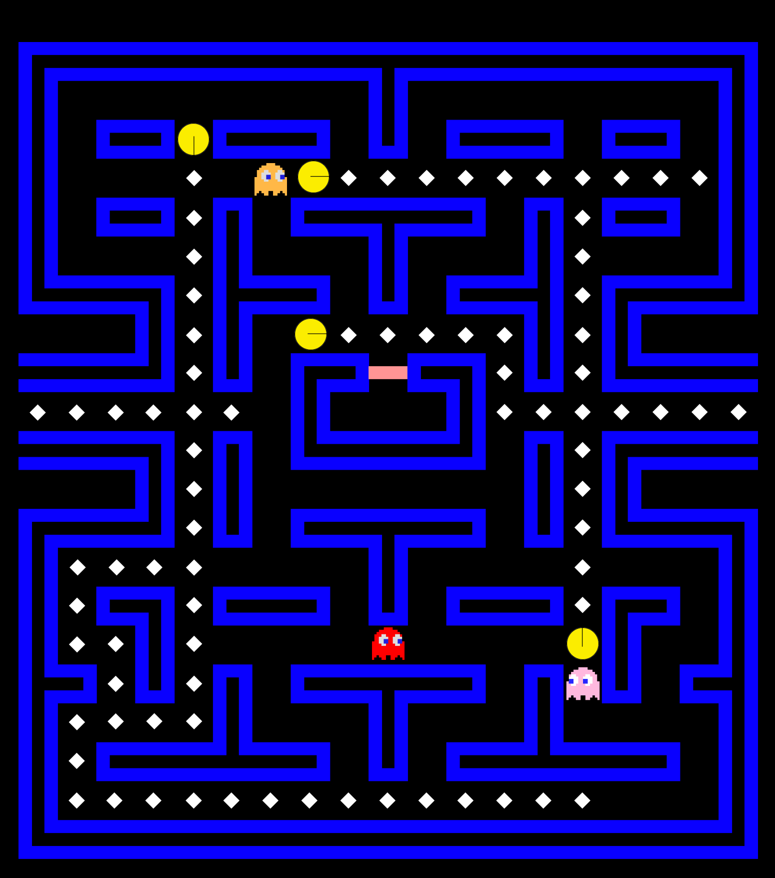


Scoreboard

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

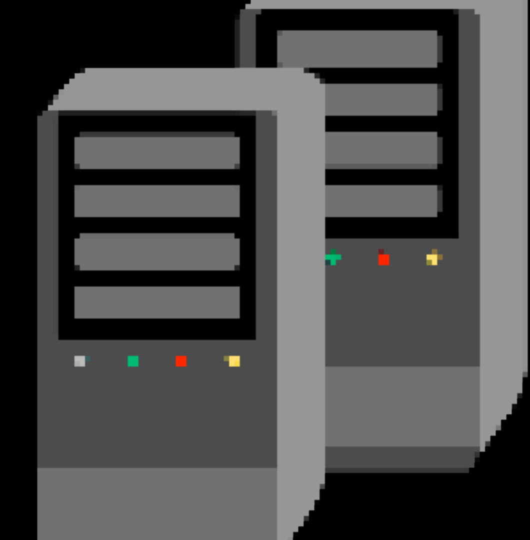


Real Enterprise

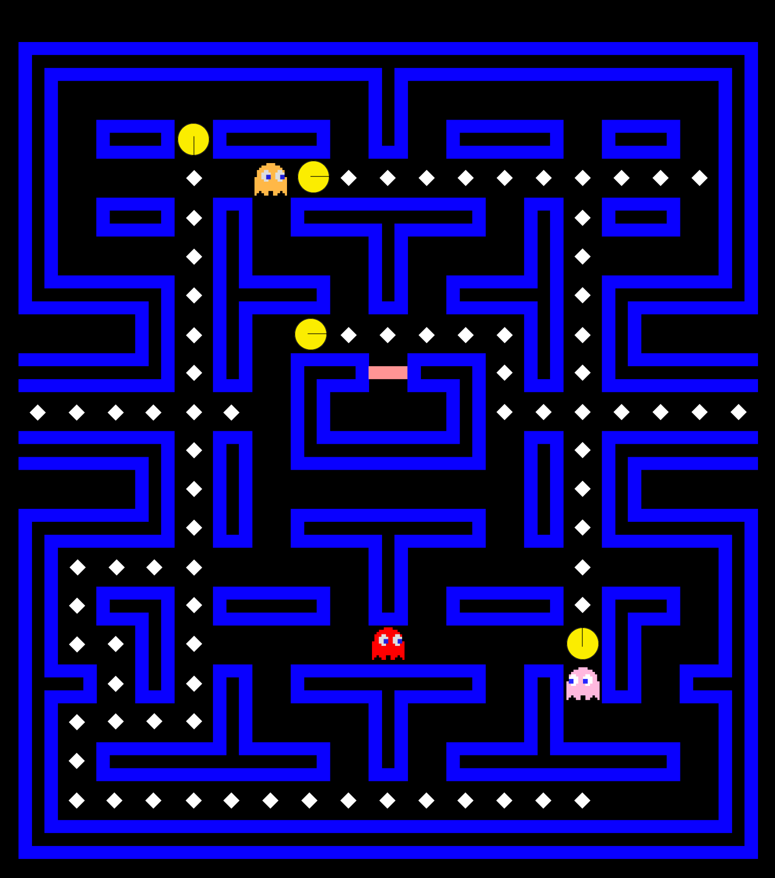


Scoreboard

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

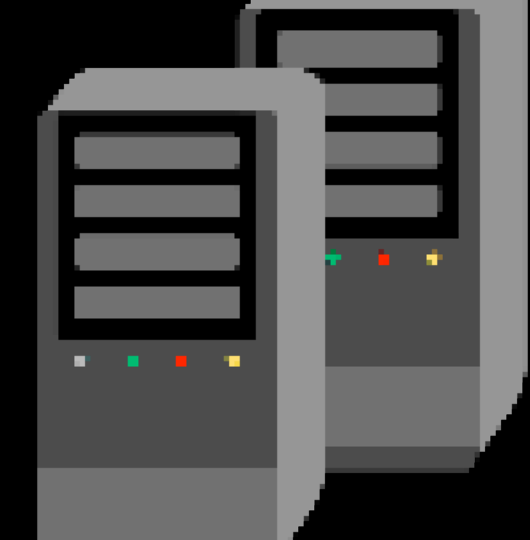


Real Enterprise

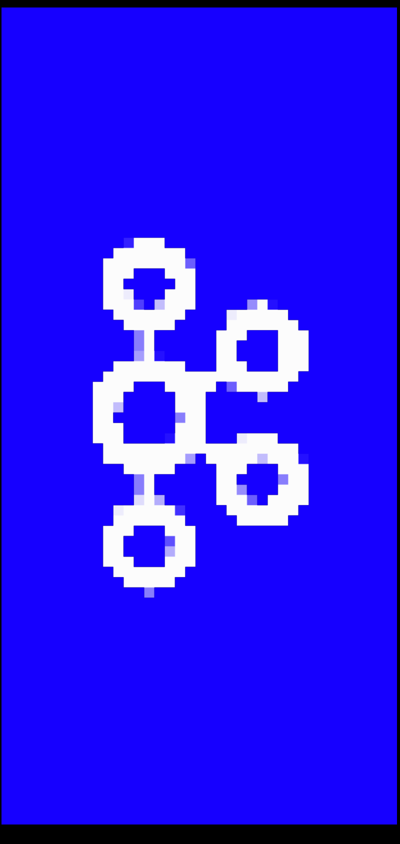


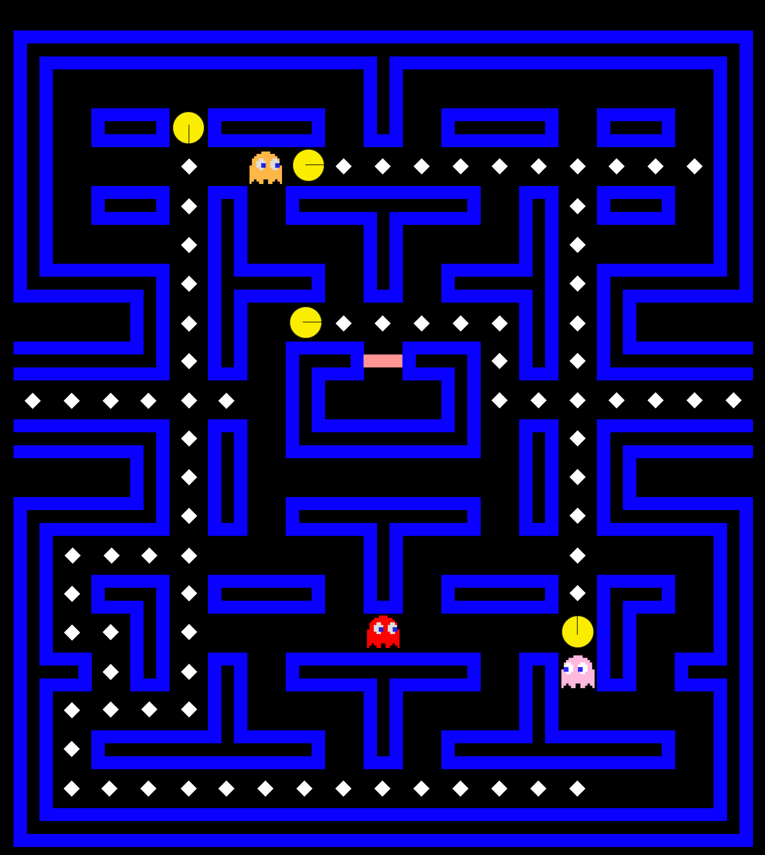
Scoreboard

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



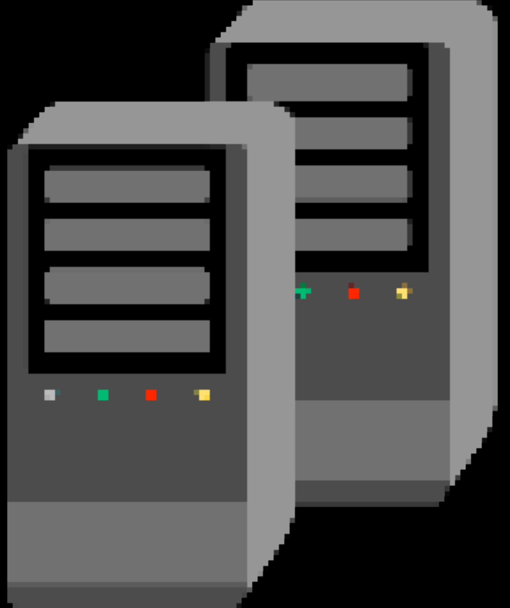
Real Enterprise



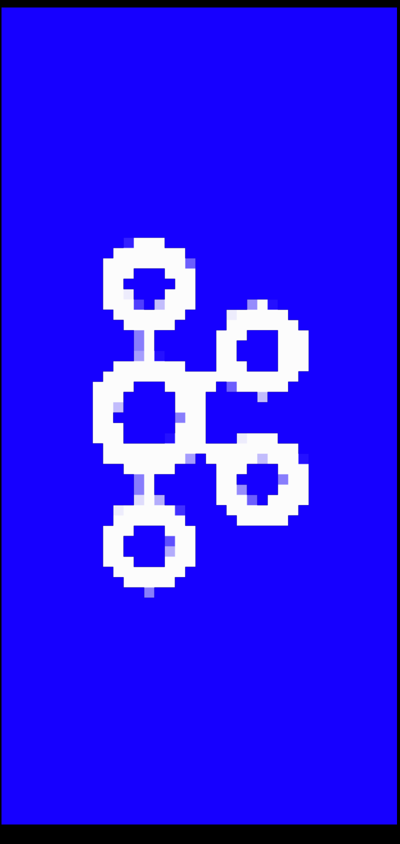


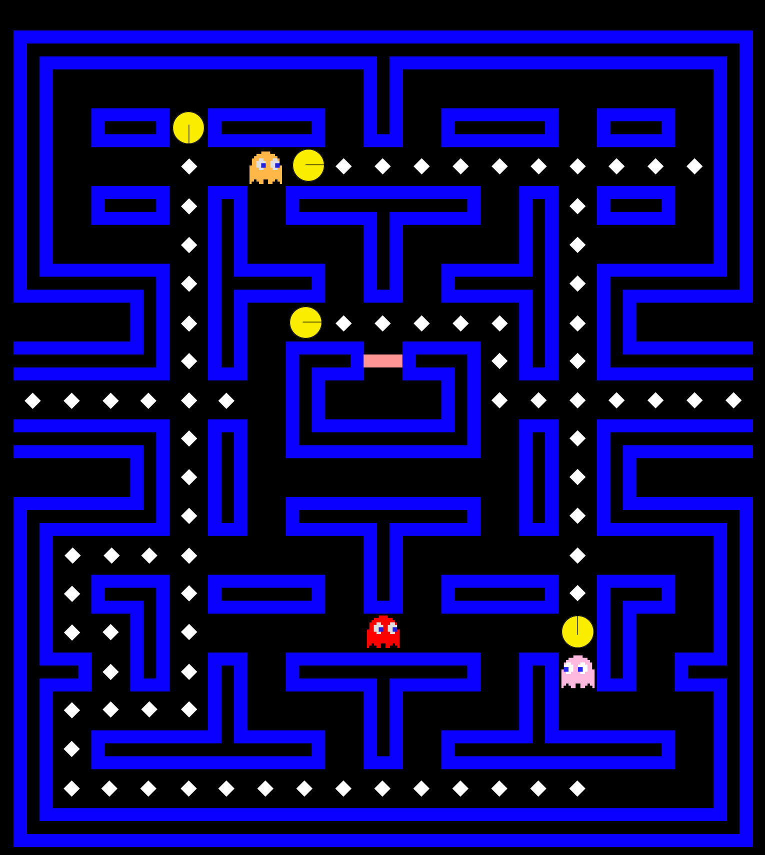
Scoreboard

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



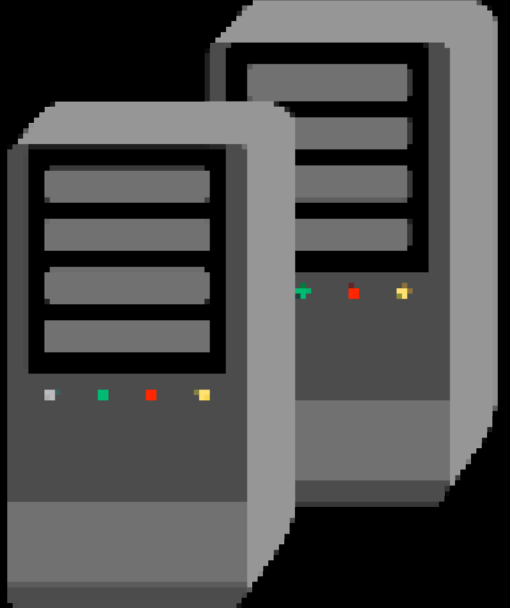
Real Enterprise



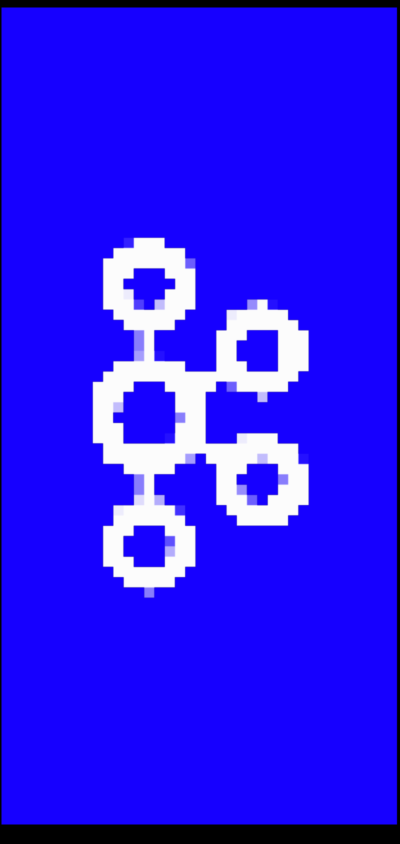


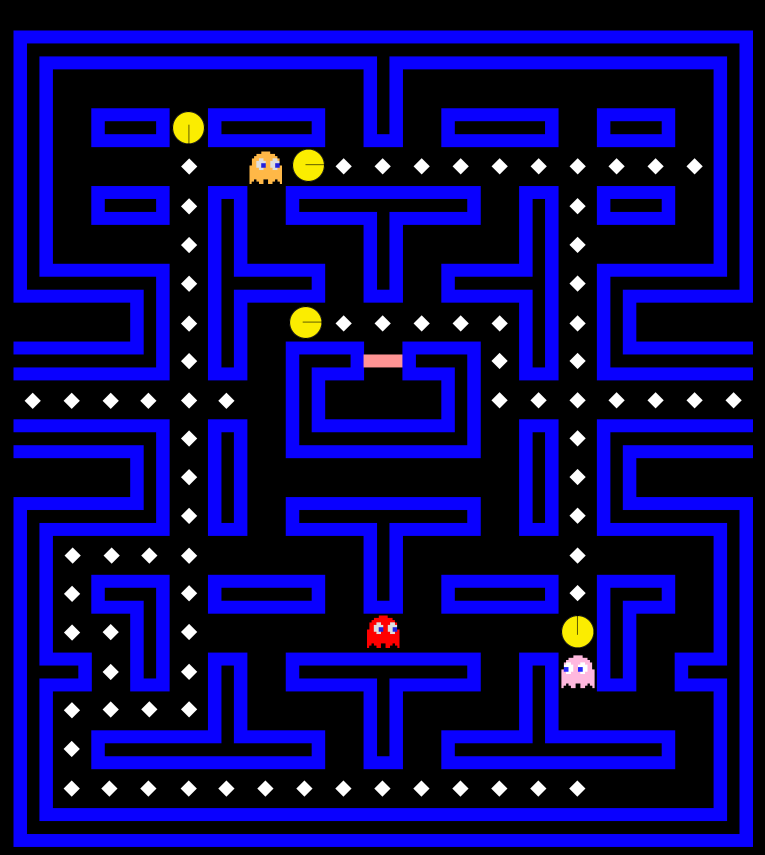
Scoreboard

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



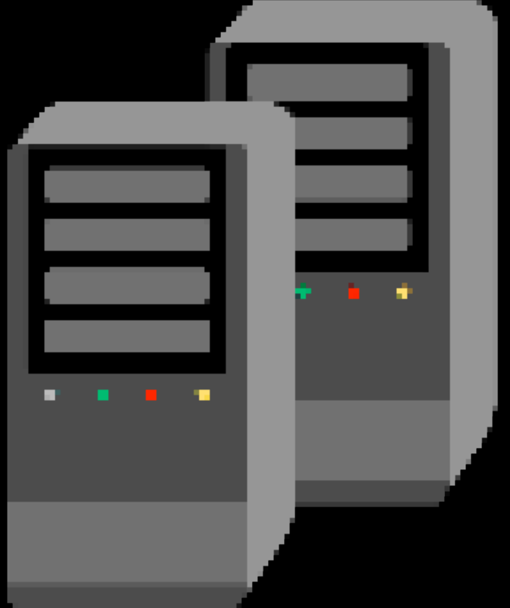
Real Enterprise



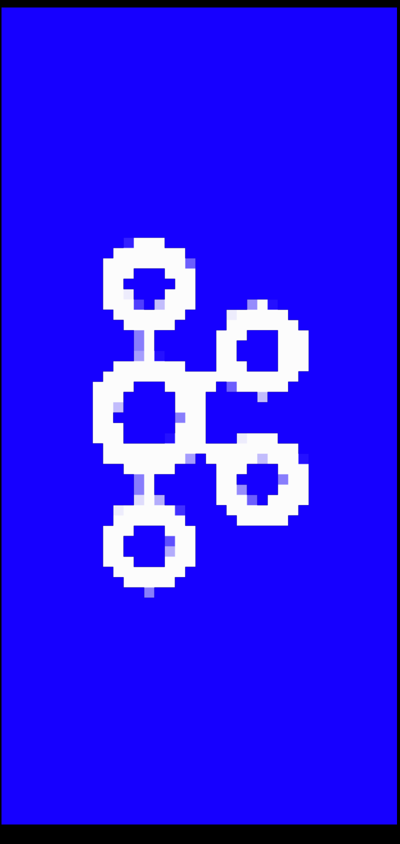


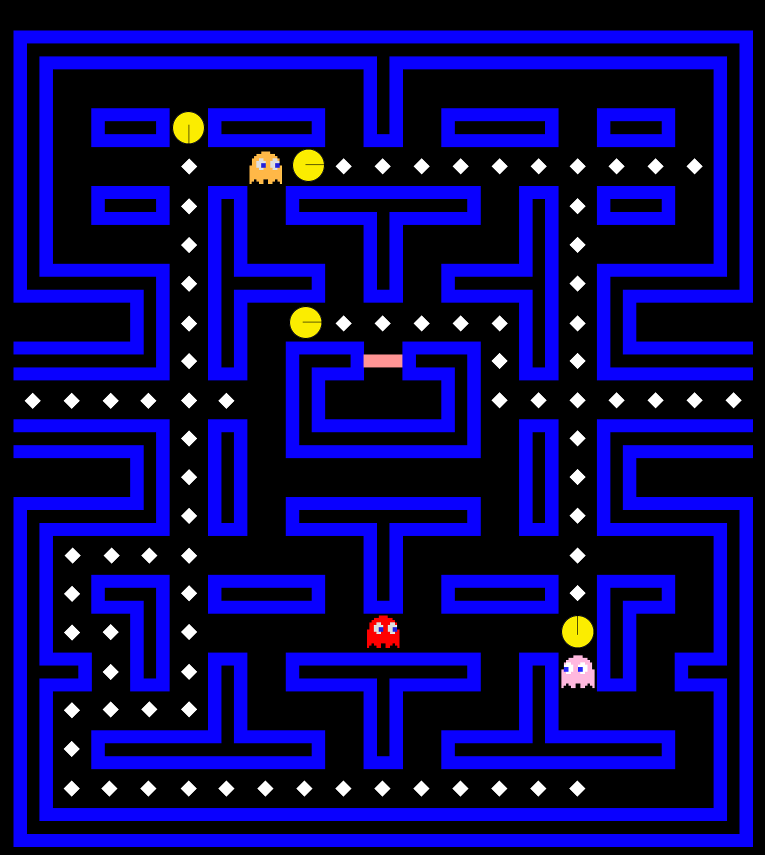
Scoreboard

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



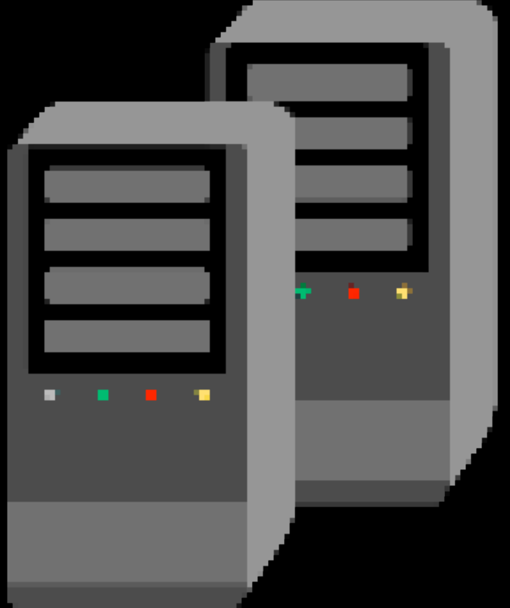
Real Enterprise



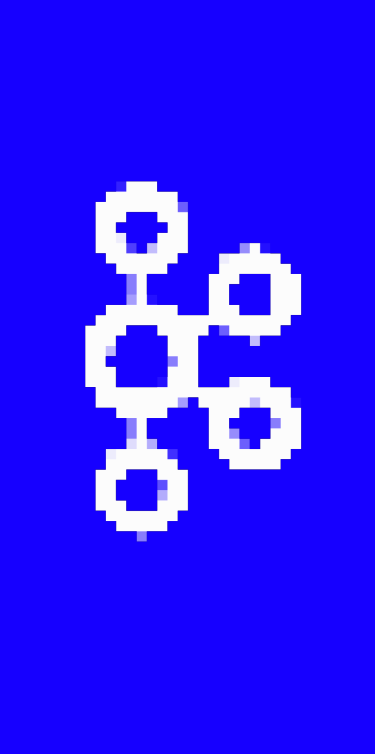


Scoreboard

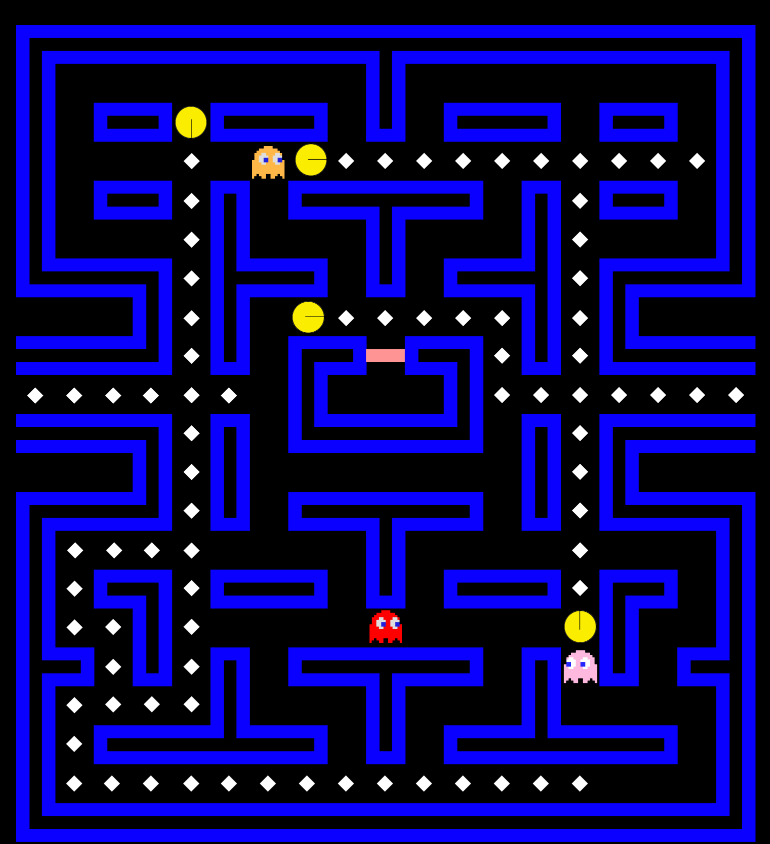
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7



Real Enterprise



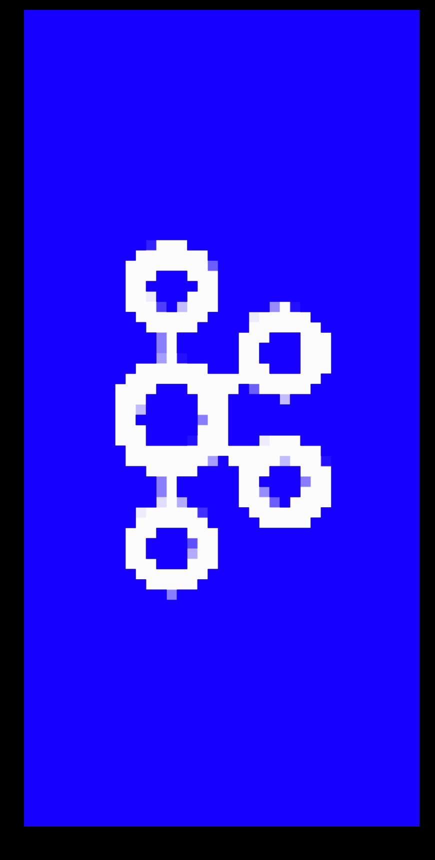
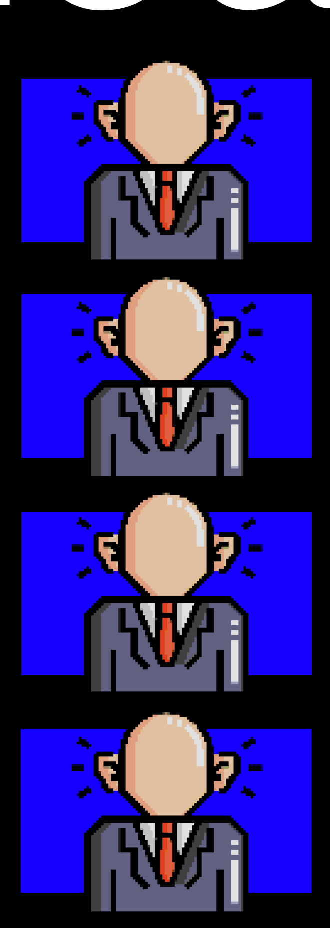
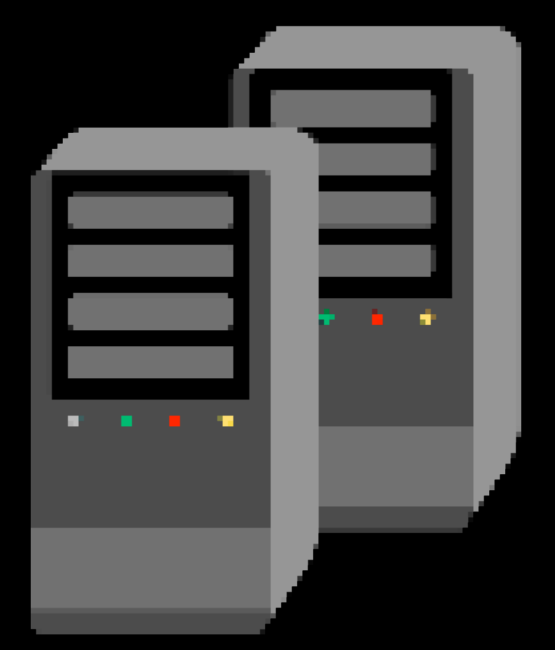
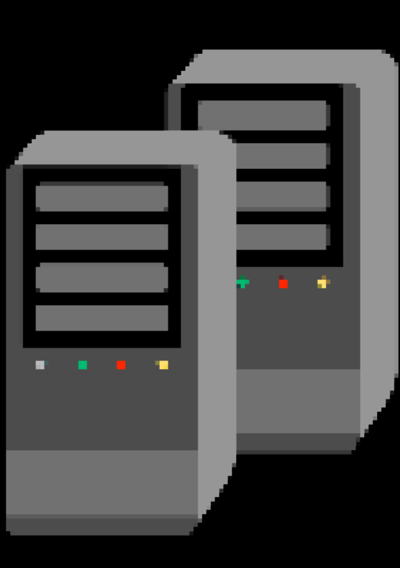
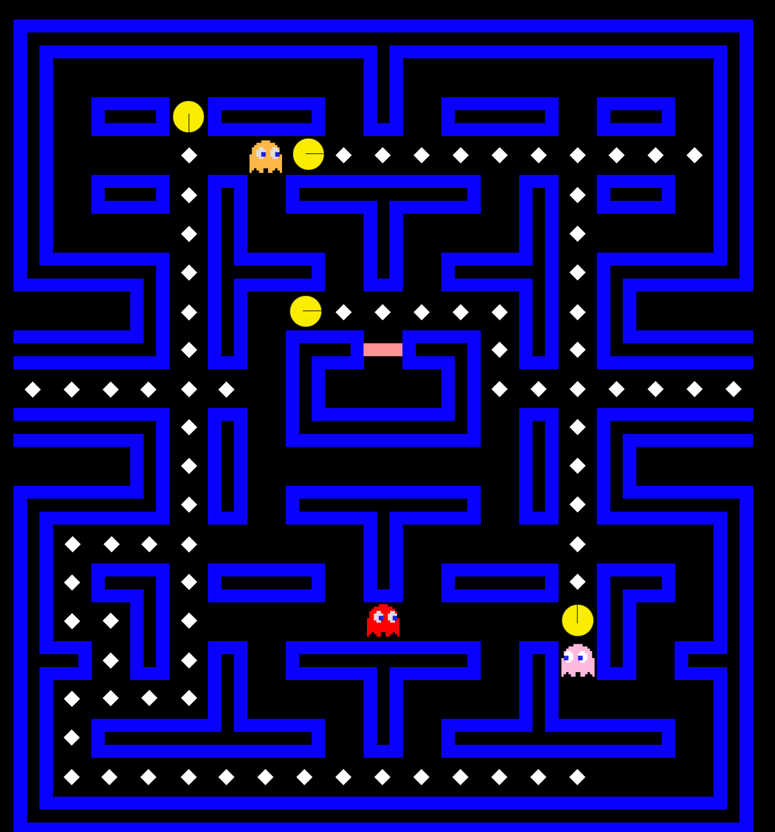
← Elastic Storage



Scoreboard

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

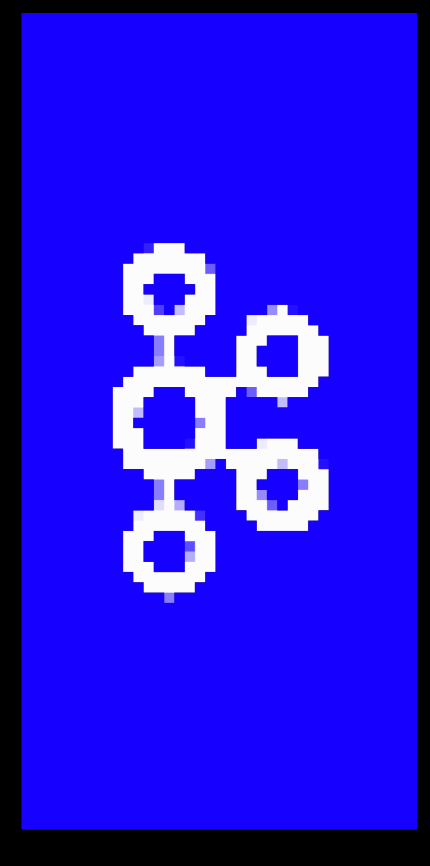
Real Enterprise

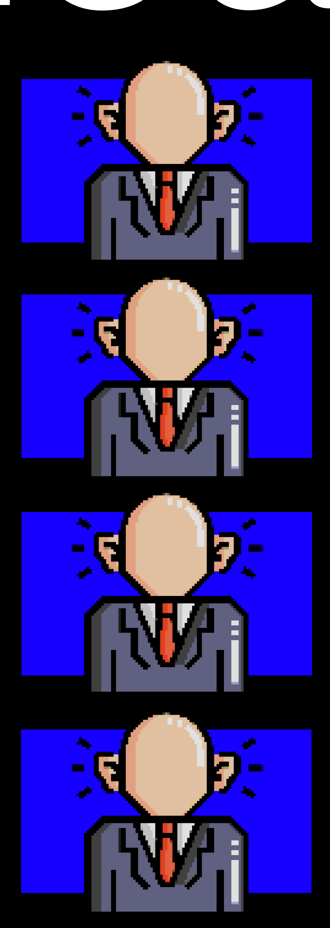






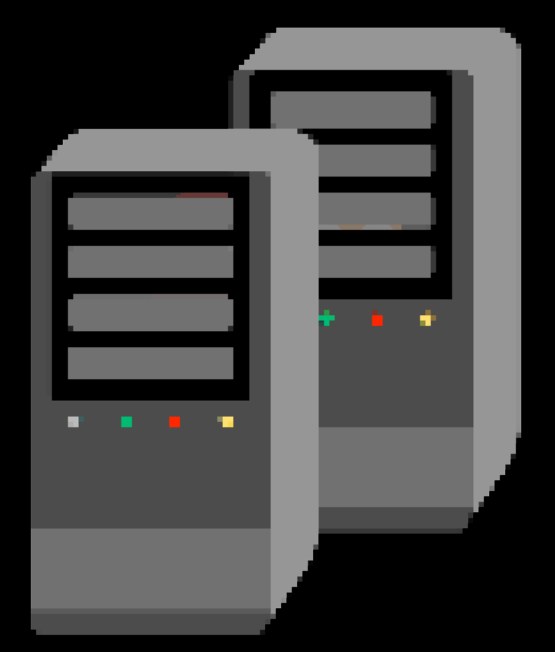
Scoreboard

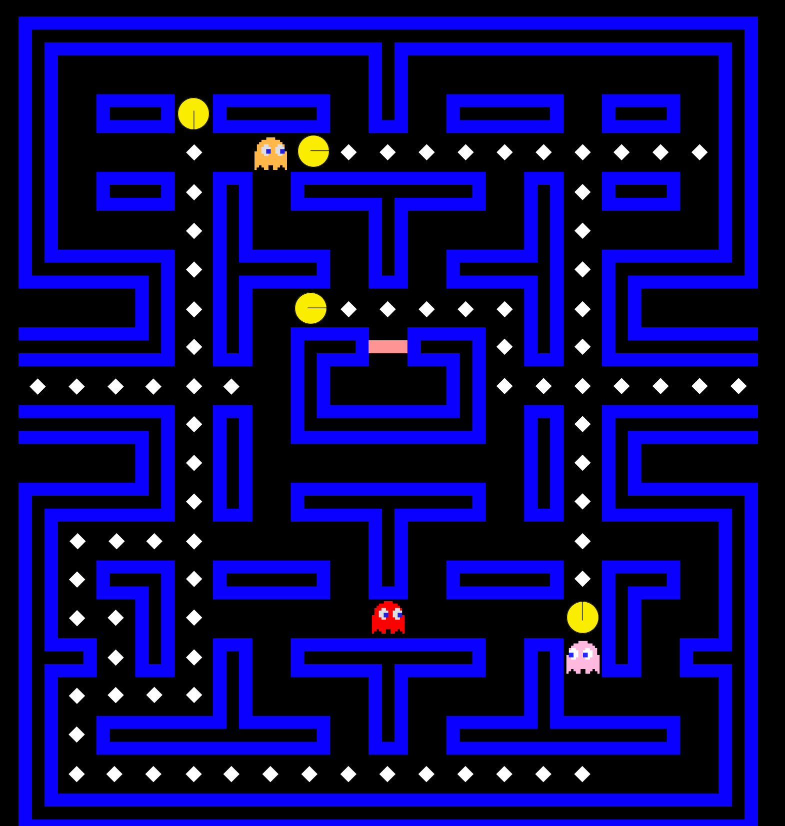
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7

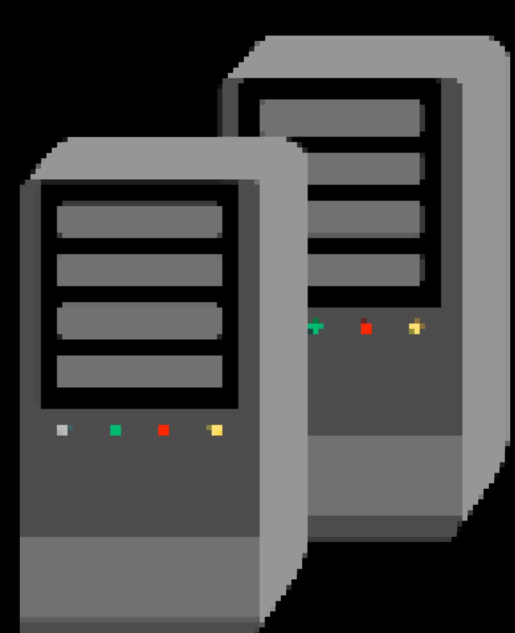
Real Enterprise







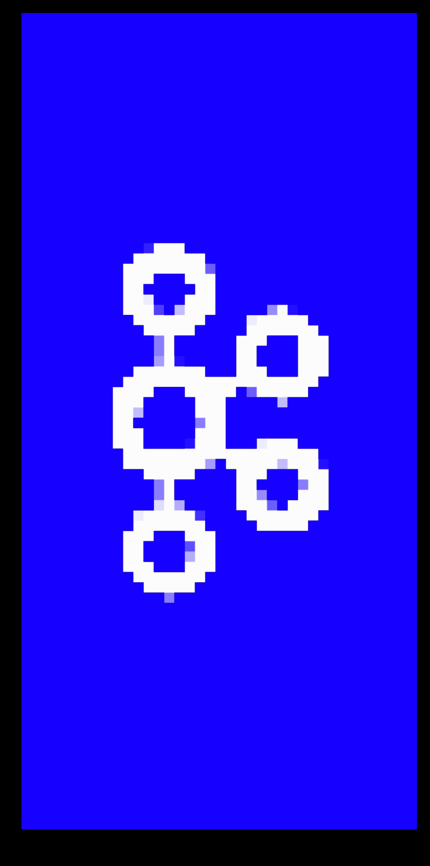
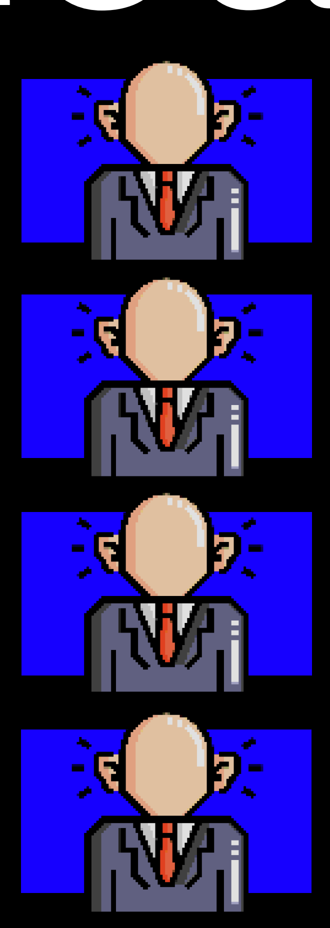
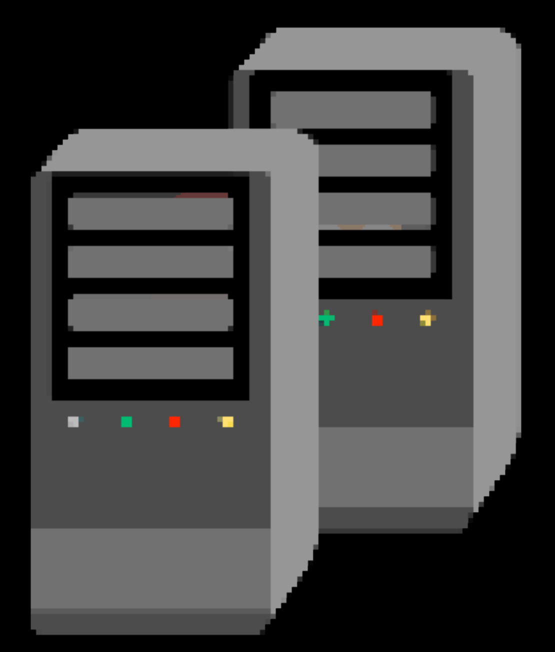


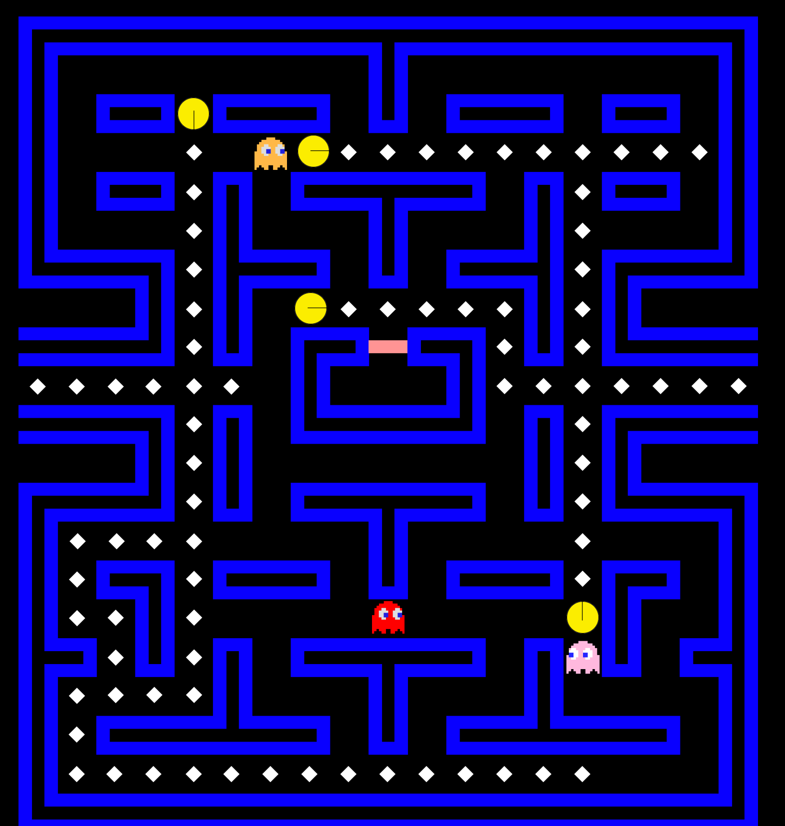


Scoreboard

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

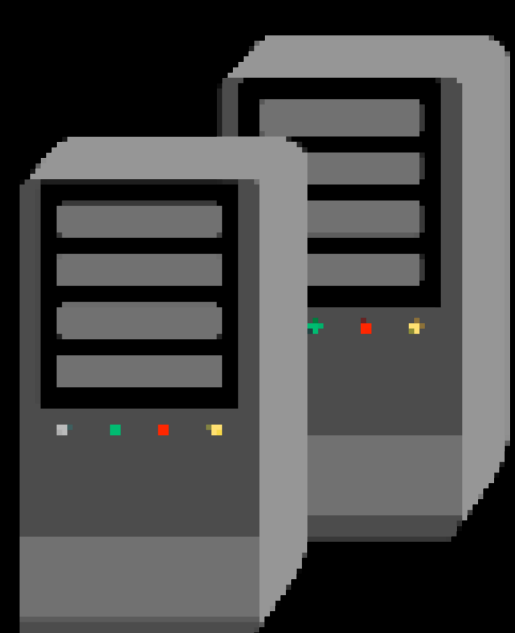
Real Enterprise

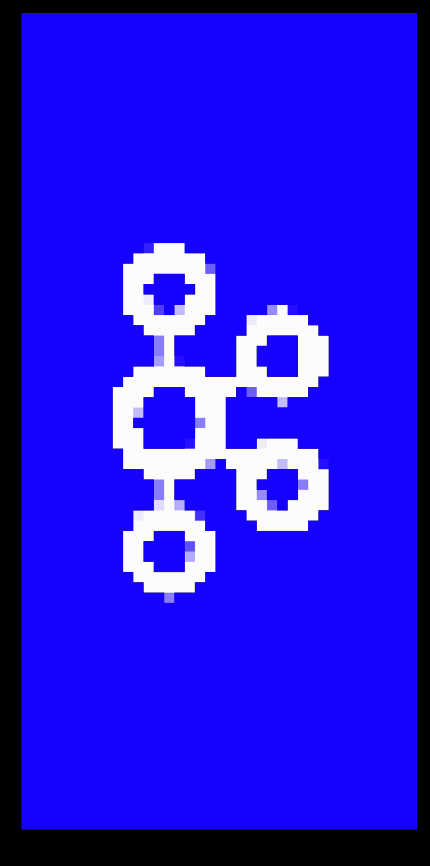


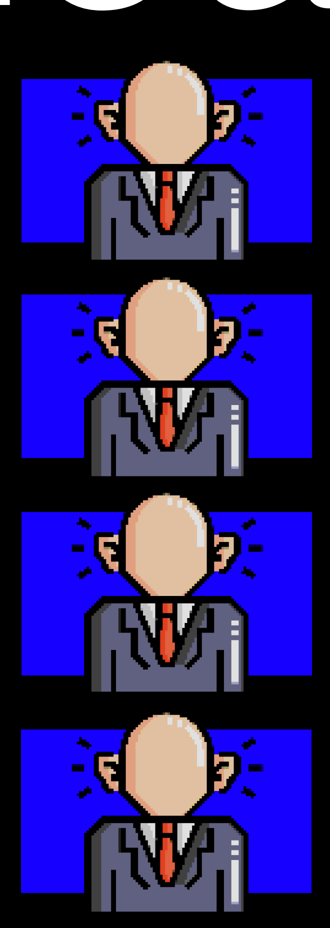
Scoreboard

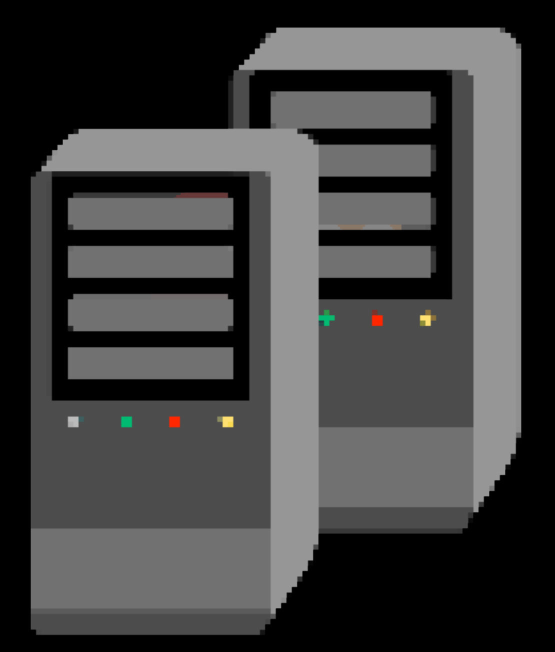
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7

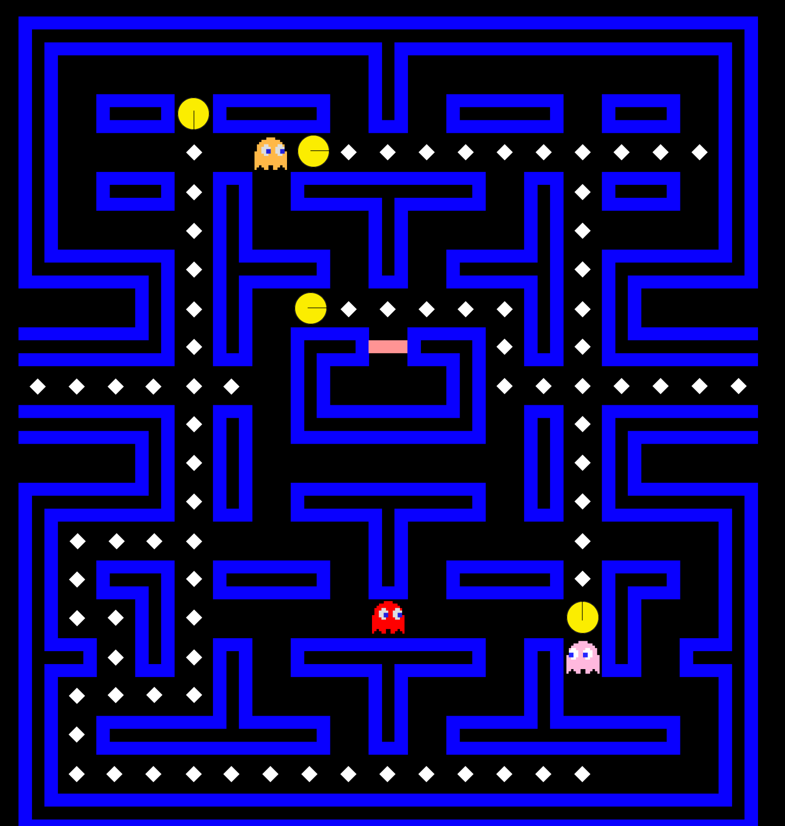


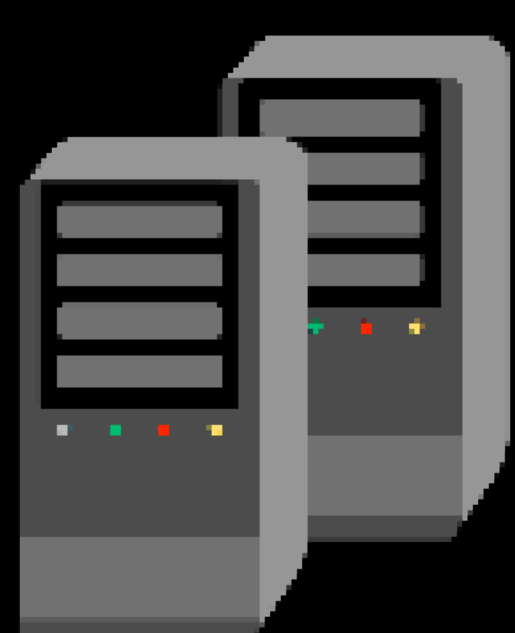
Real Enterprise







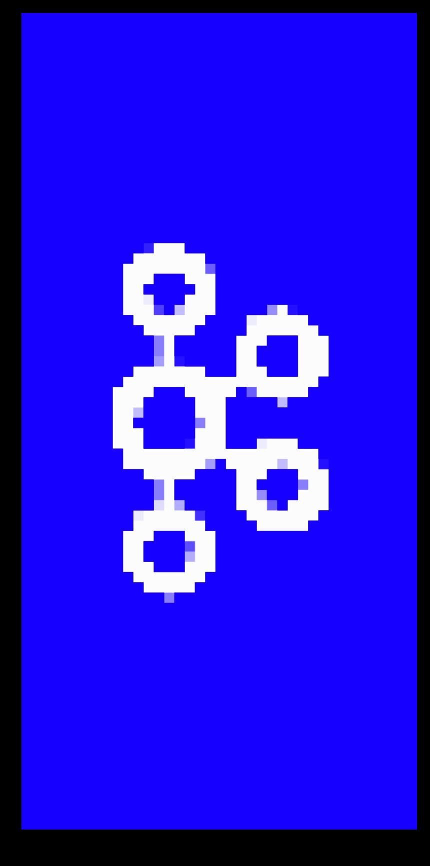


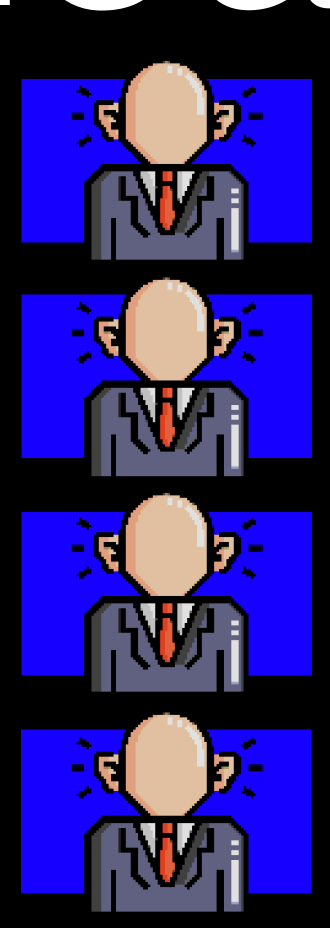


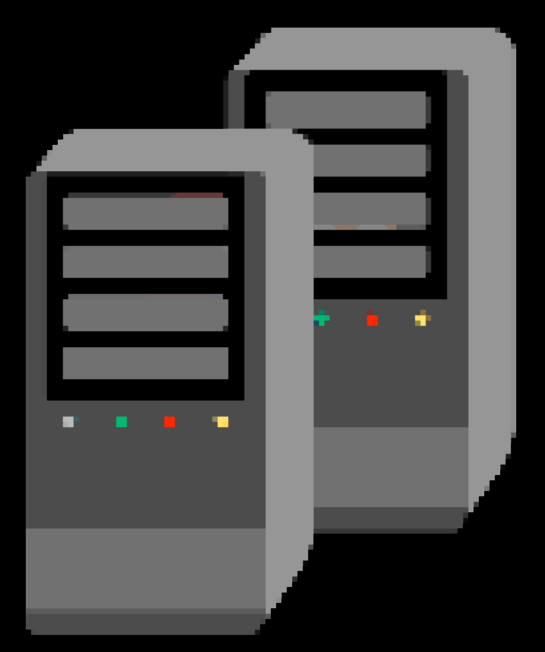
Scoreboard

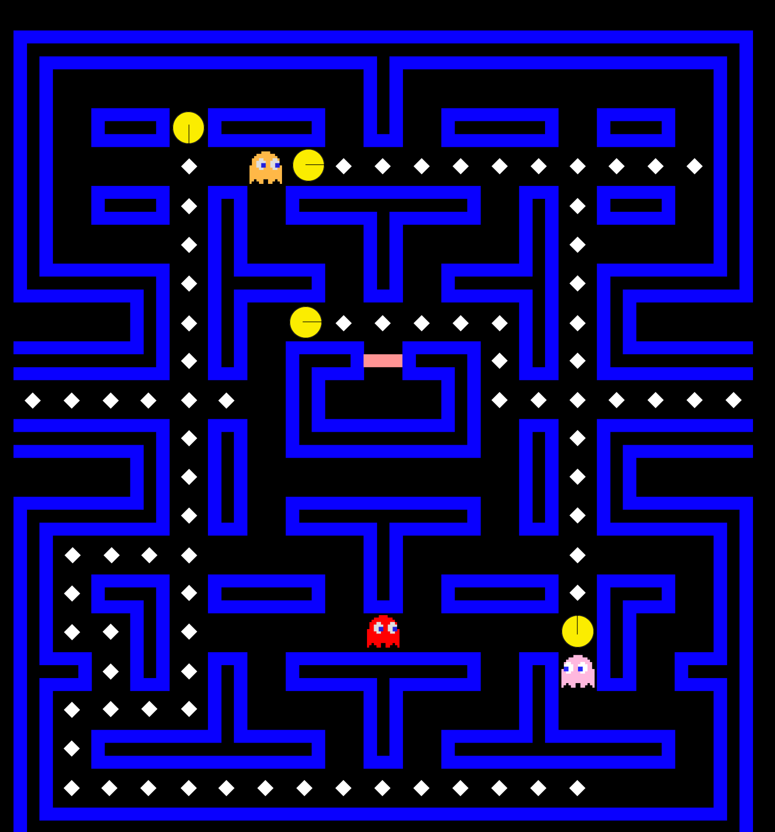
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7

Real Enterprise





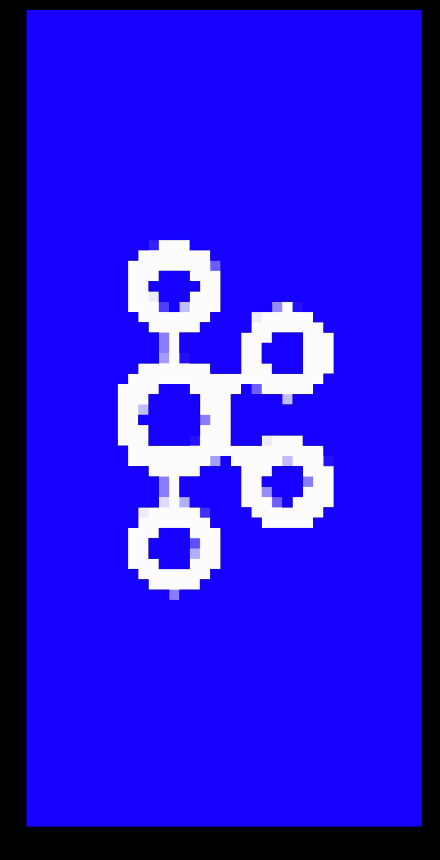


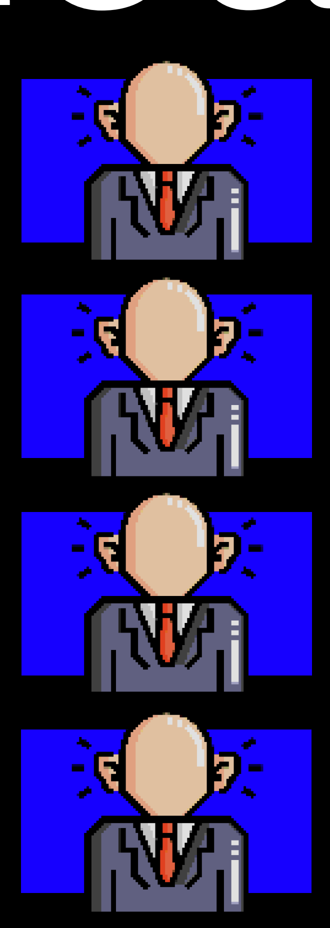


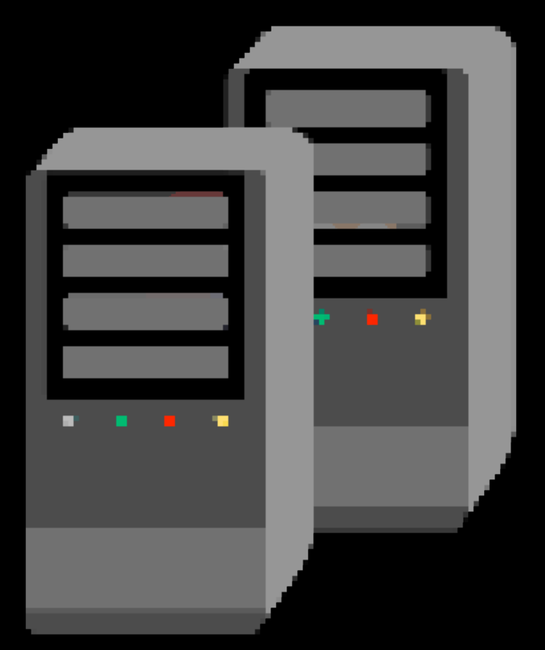
Scoreboard

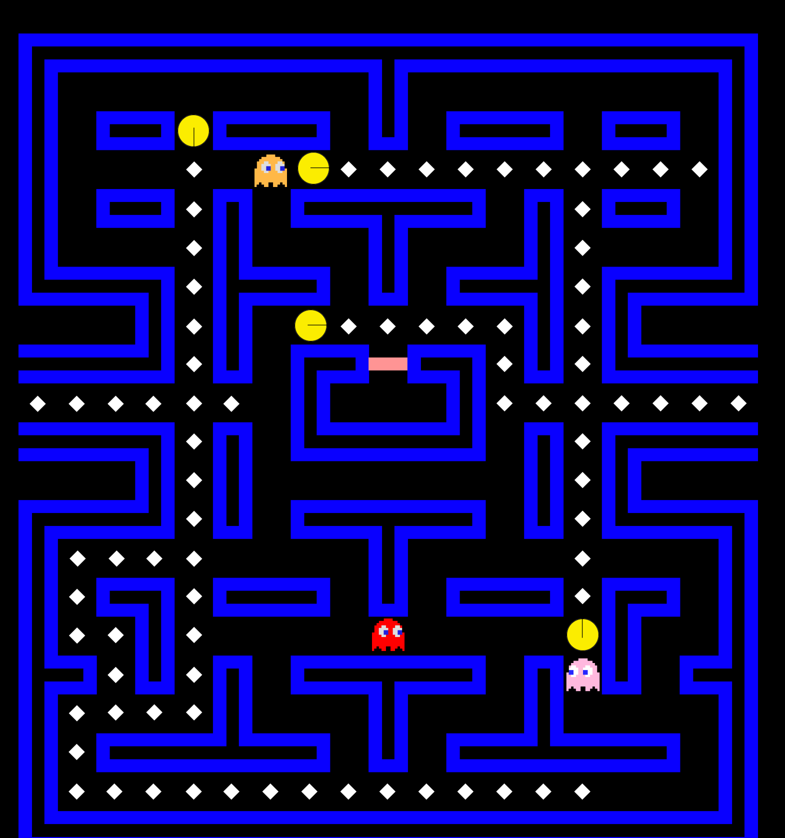
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7

Real Enterprise



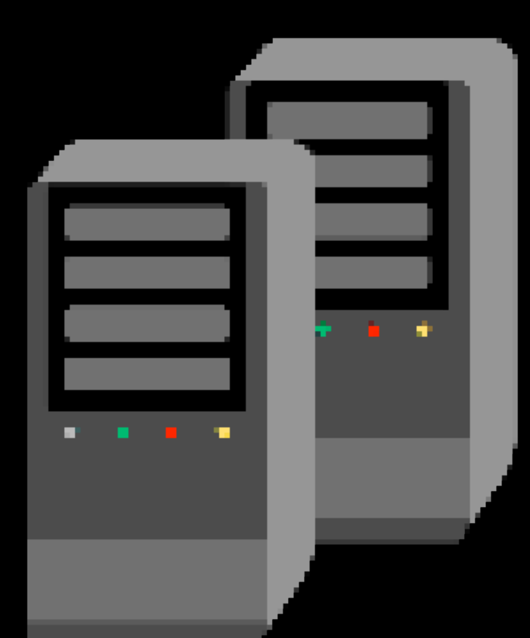




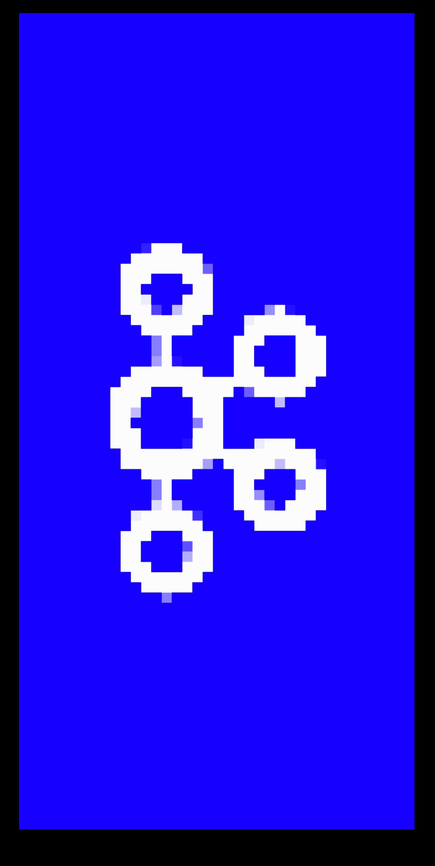


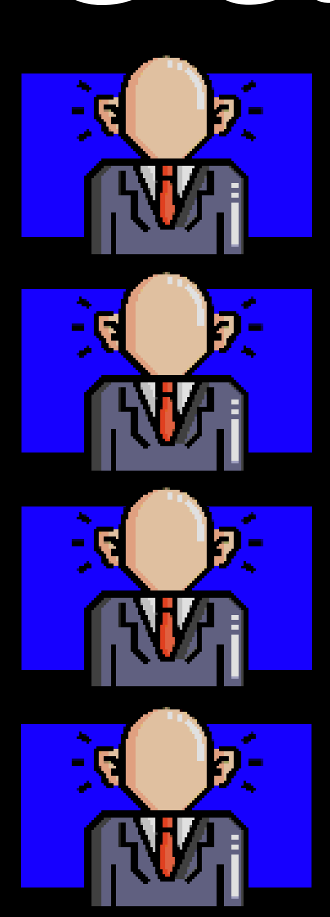
Scoreboard

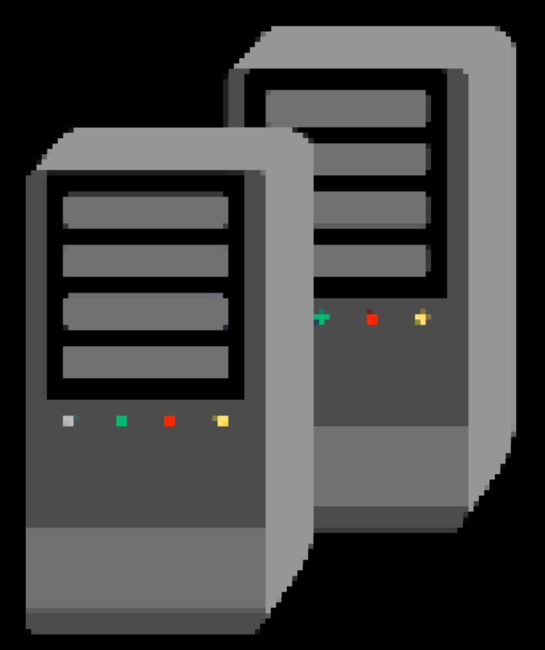
100. Player 1
88. Player 3
80. Player 5
75. Player 2
68. Player 6
30. Player 4
7. Player 7

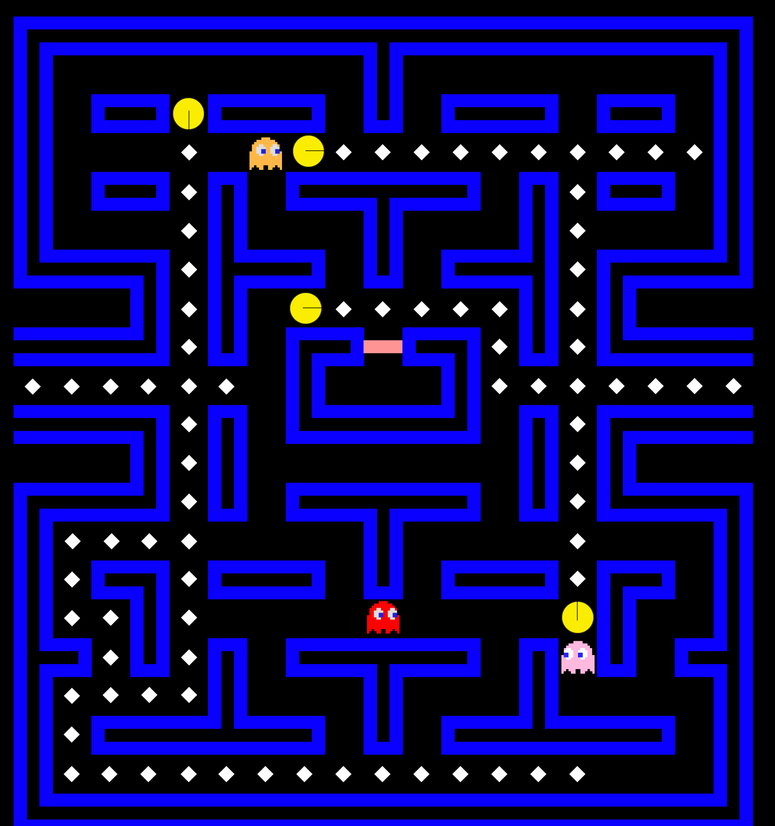


Real Enterprise









Scoreboard

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

To Summarize

To Summarize

- SERVER PUSH

To Summarize

- SERVER PUSH
- PLAIN REQUEST-RESPONSE

To Summarize

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

To Summarize

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

To Summarize

To Summarize

- MACHINE LEARNING PIPE

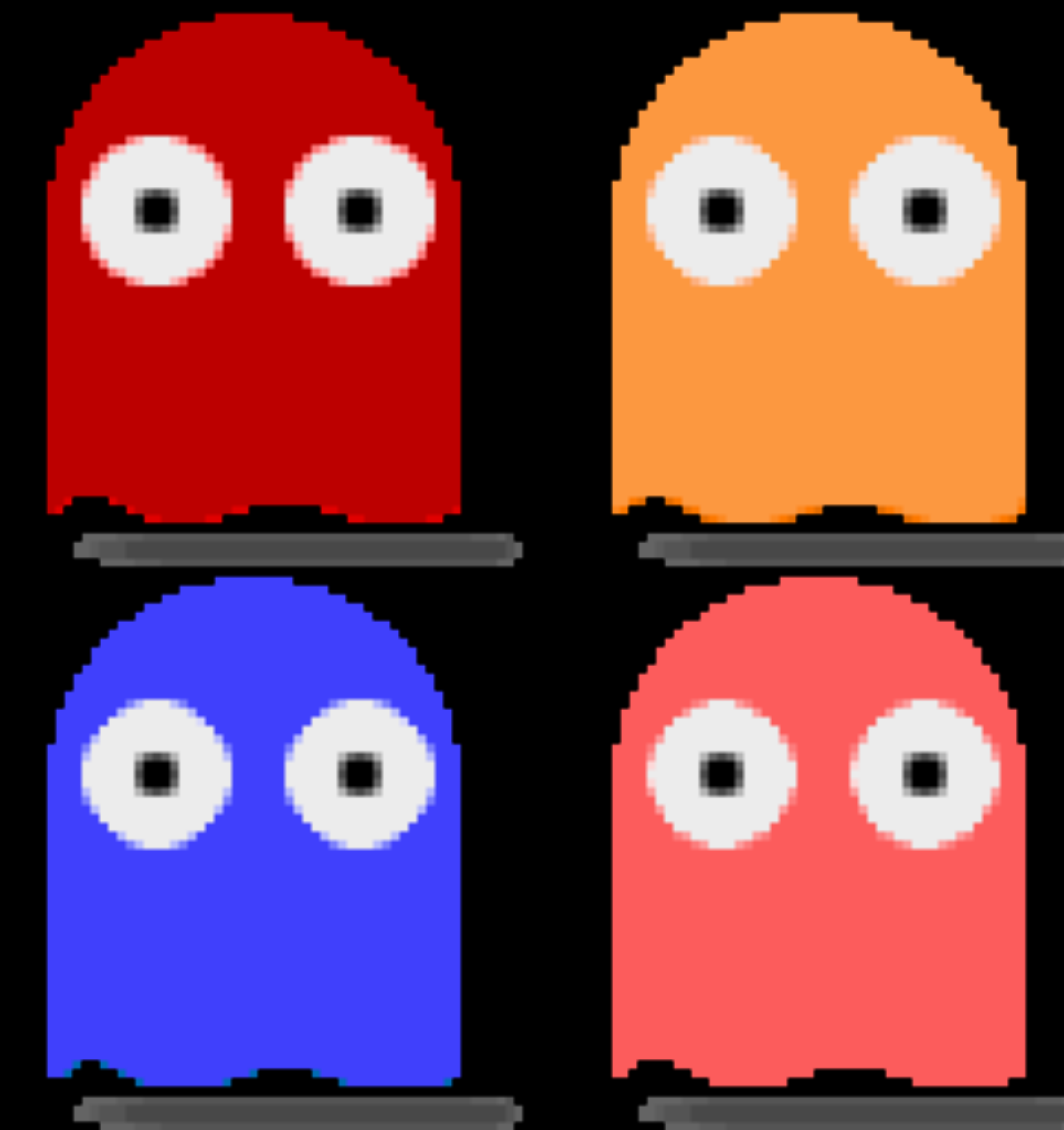
To Summarize

- MACHINE LEARNING PIPE
- WHERE SUBSCRIBER CAN WORK SLOW OR FAST

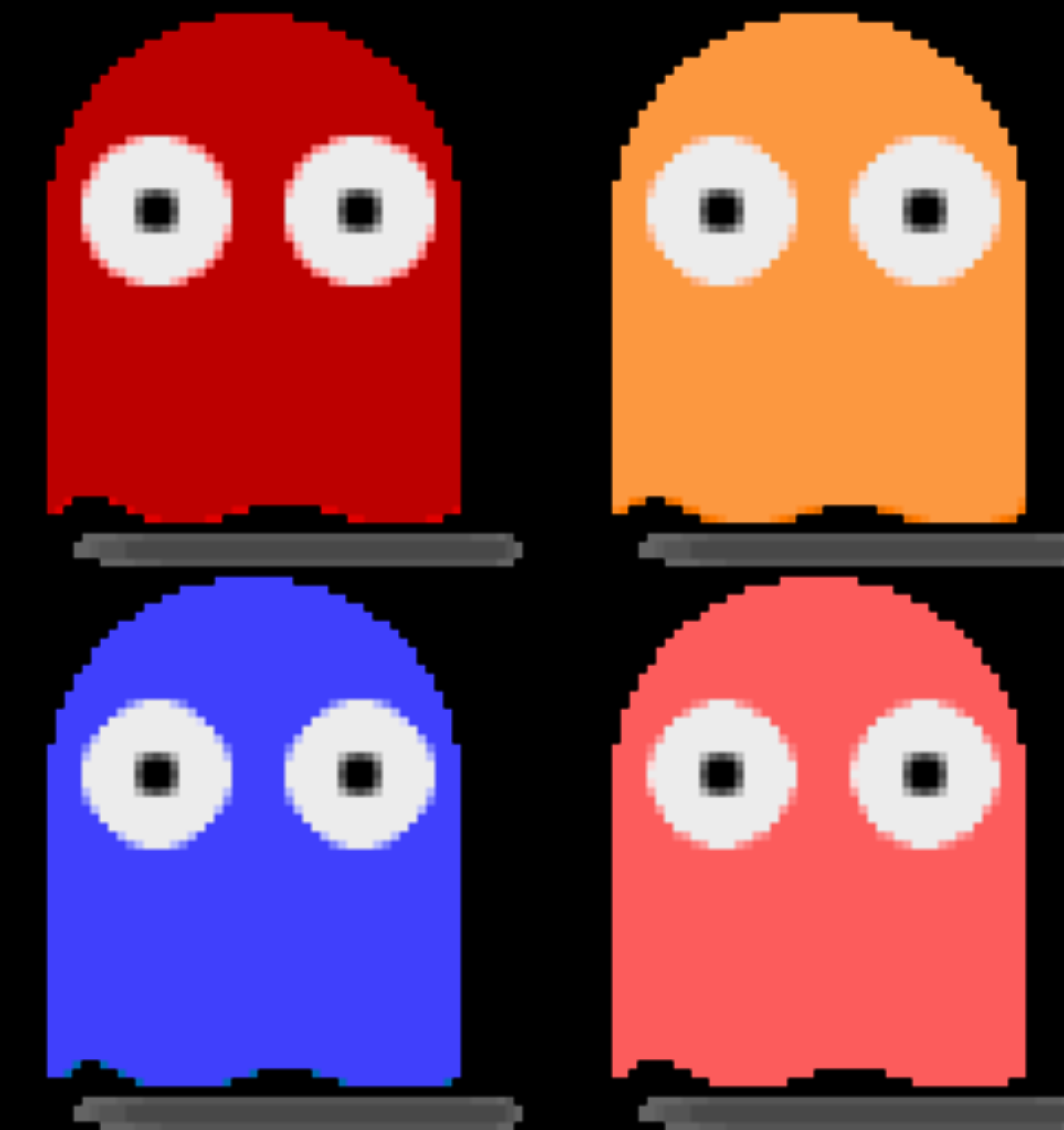
To Summarize

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

TOOLKIT



TOOLKIT



Back-end

Back-end

- SPRING FRAMEWORK 5

Back-end

- SPRING FRAMEWORK 5
- PROJECT REACTOR 3

Back-end

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

Front-end

Front-end

- PHASER 3

Front-end

- PHASER 3
- REACTOR-JS

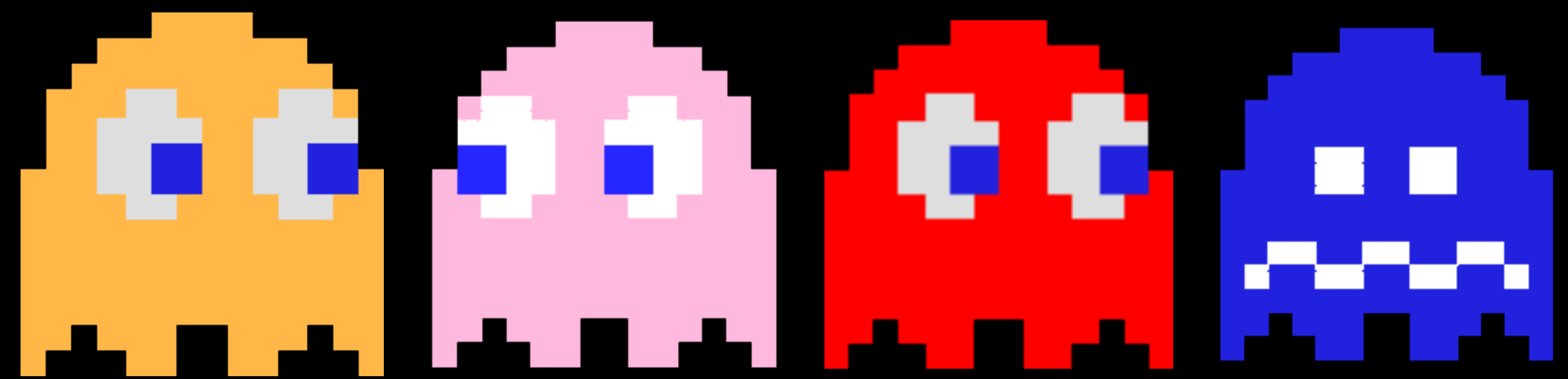
Front-end

- PHASER 3
- REACTOR-JS
- TYPESCRIPT

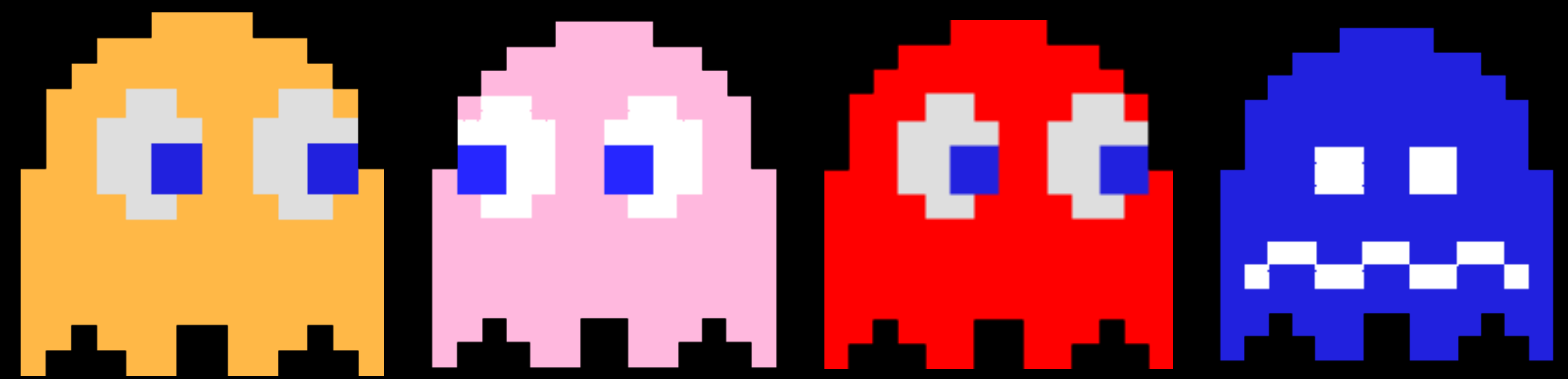
Front-end

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

OLD HTTP WAY



OLD HTTP WAY



Why HTTP?

Why HTTP?

- PLAIN AND SIMPLE

Why HTTP?

- PLAIN AND SIMPLE
- USED FOR MANY YEARS

```java
@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)));
  }
}
```

```java
@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)));
  }
}
```

is.gd/webflux

DEMO

Why NOT HTTP?

Why NOT HTTP?

- TEXT MESSAGE OVERHEAD

Why NOT HTTP?

- TEXT MESSAGE OVERHEAD
- INEFFICIENT RESOURCE USAGE

Why NOT HTTP?

- TEXT MESSAGE OVERHEAD
- INEFFICIENT RESOURCE USAGE
- SLOW PERFORMANCE

Why NOT HTTP?

- TEXT MESSAGE OVERHEAD
- INEFFICIENT RESOURCE USAGE
- SLOW PERFORMANCE
- COMMUNICATION RIGIDITY

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

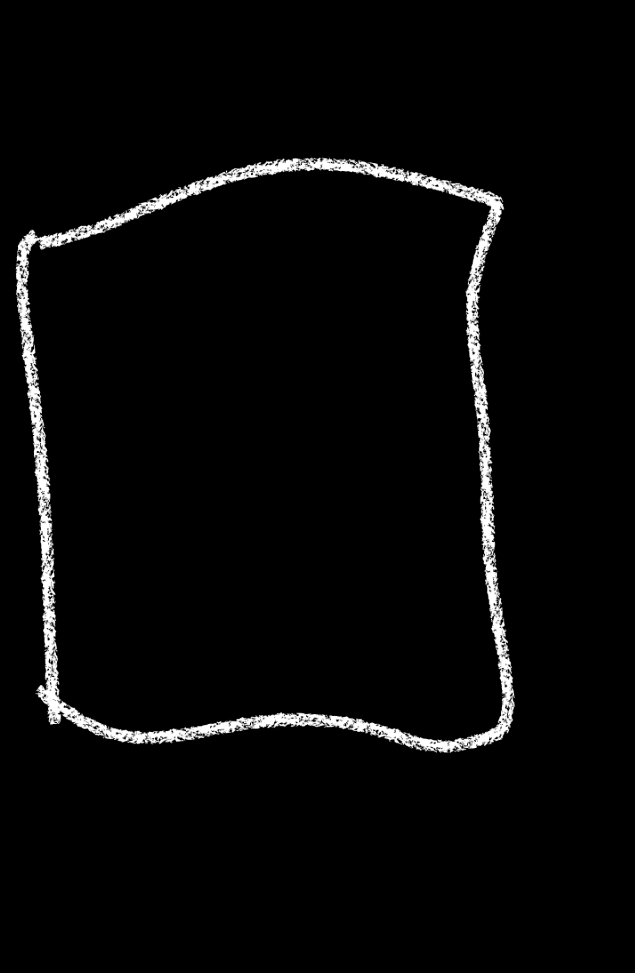
Timeouts

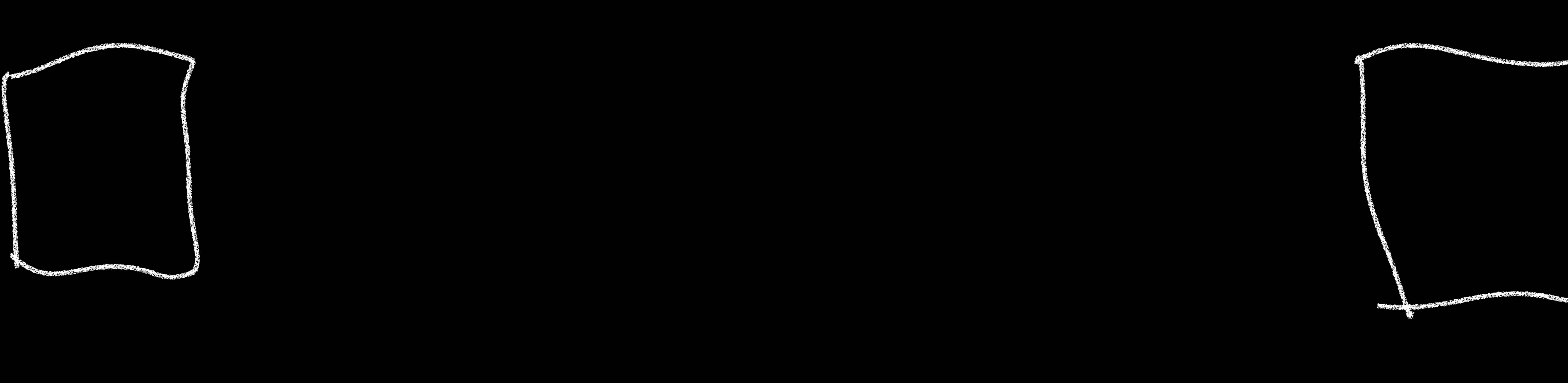
Circuit breaking

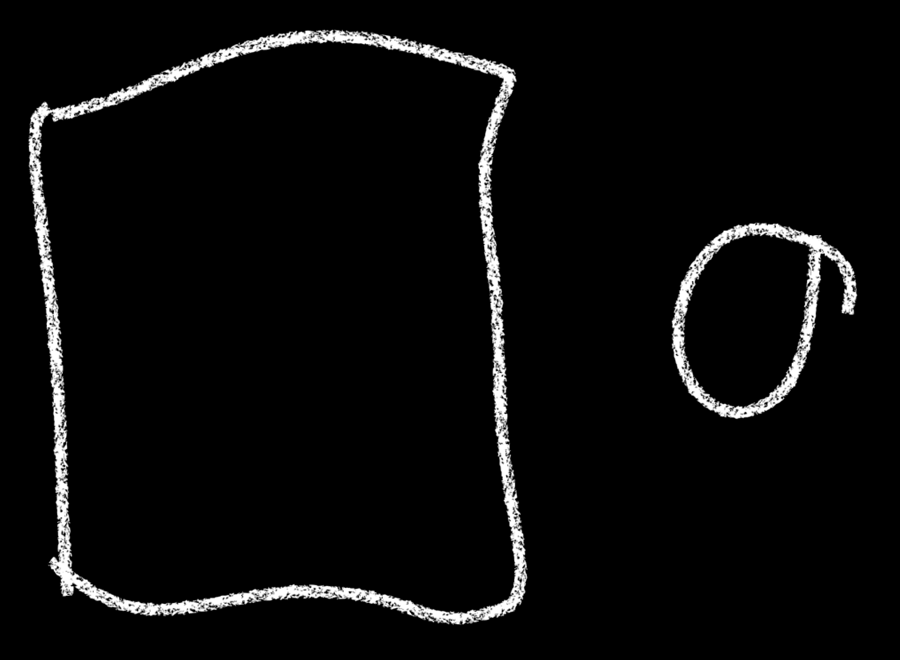
Thundering herds

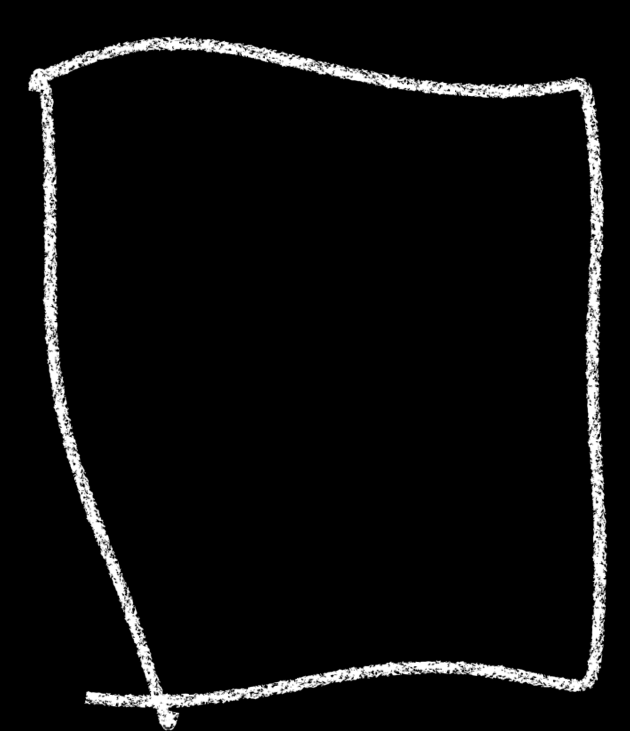
Cascading failure

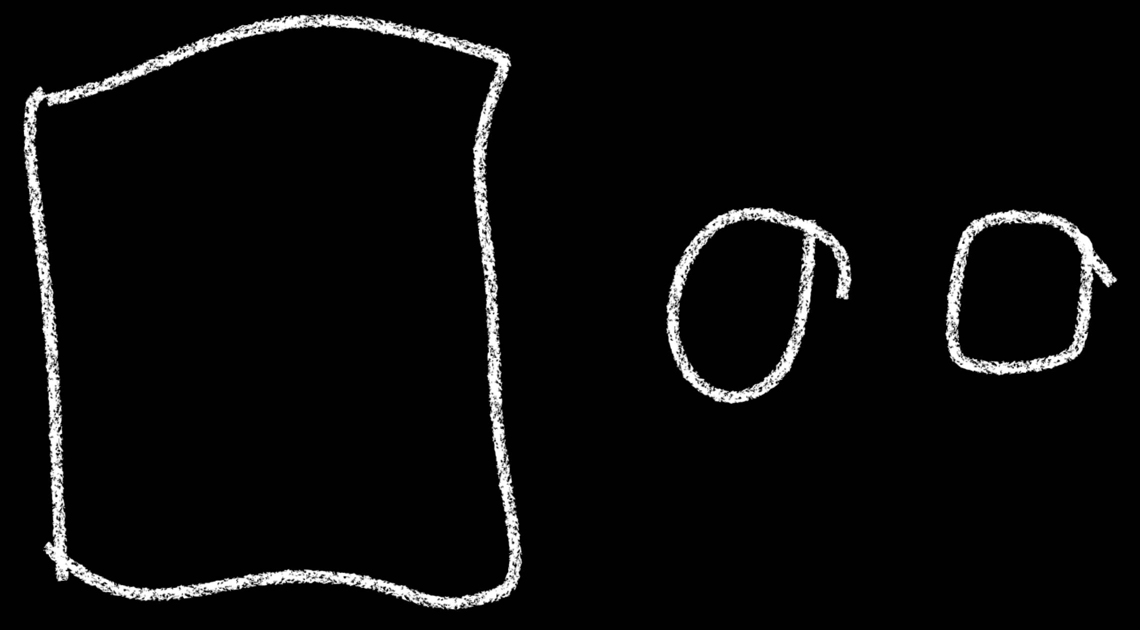
Configuration

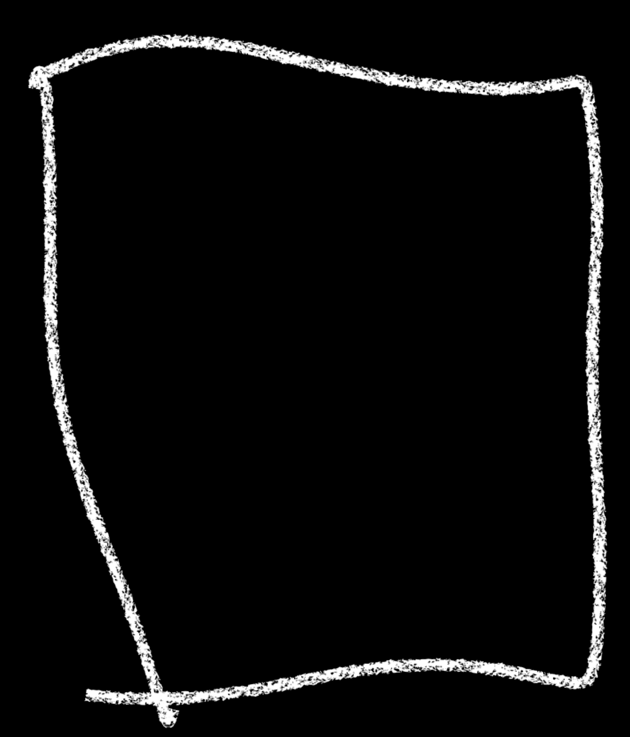


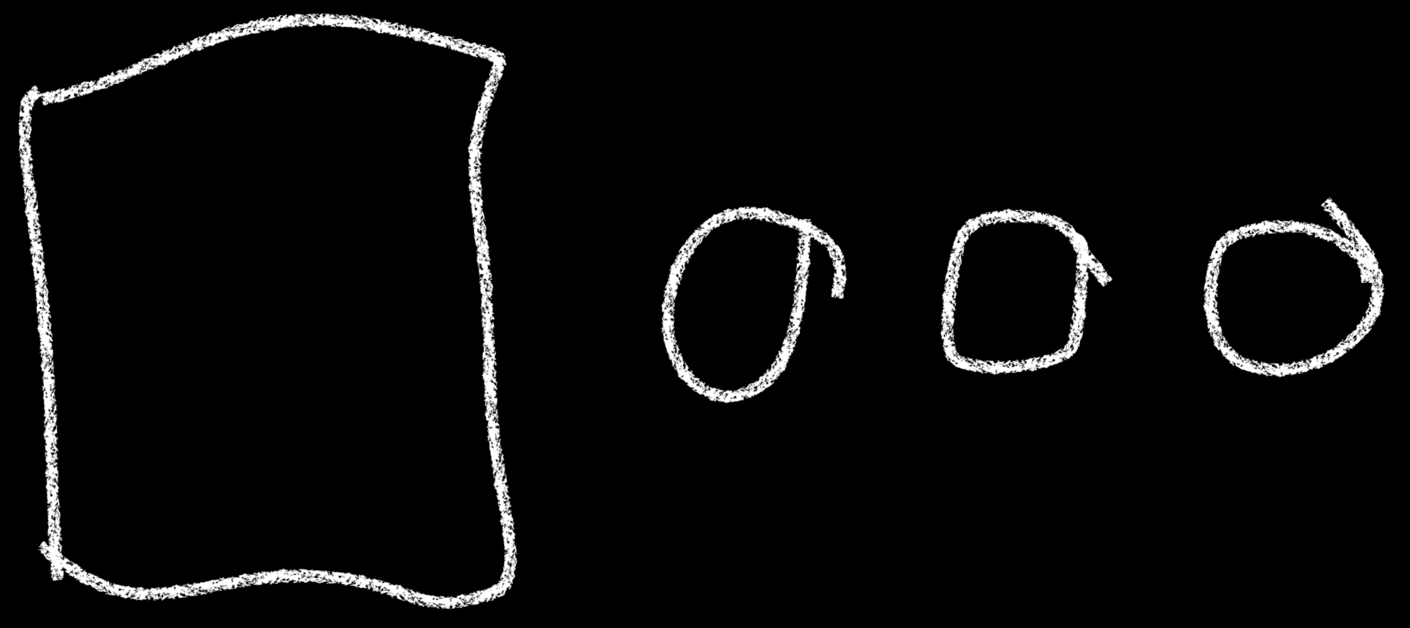


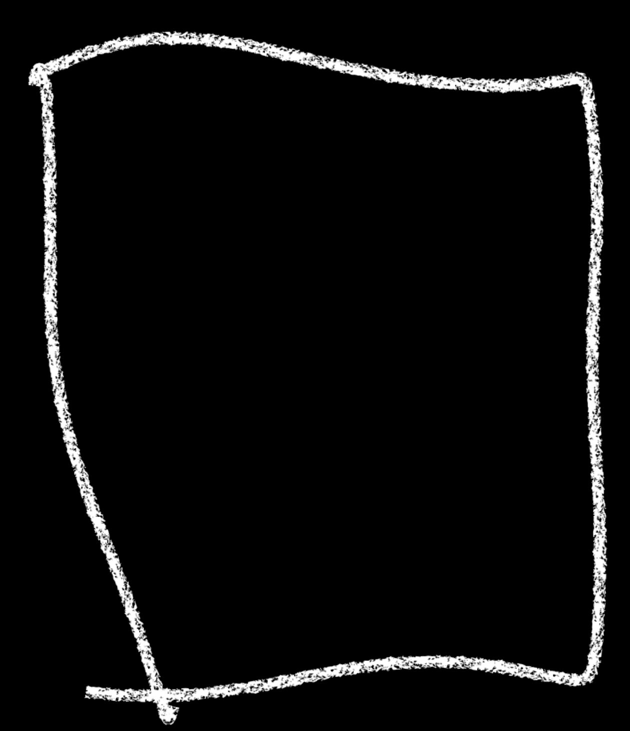


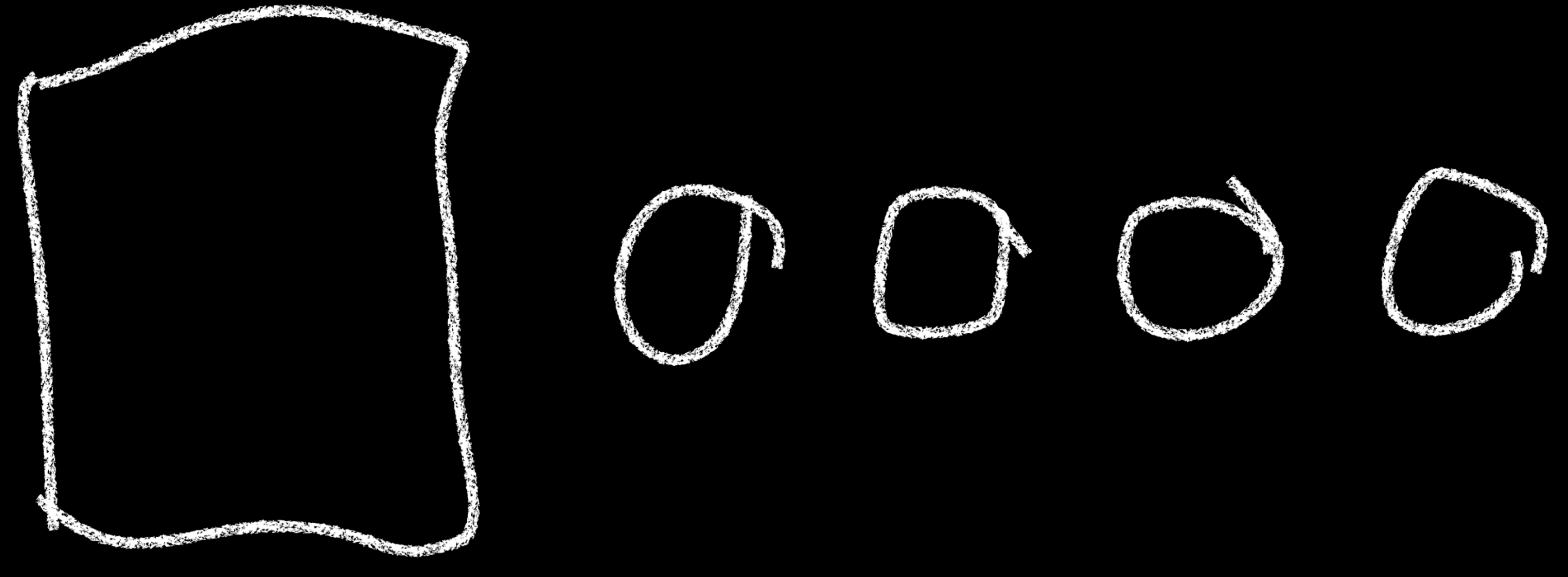


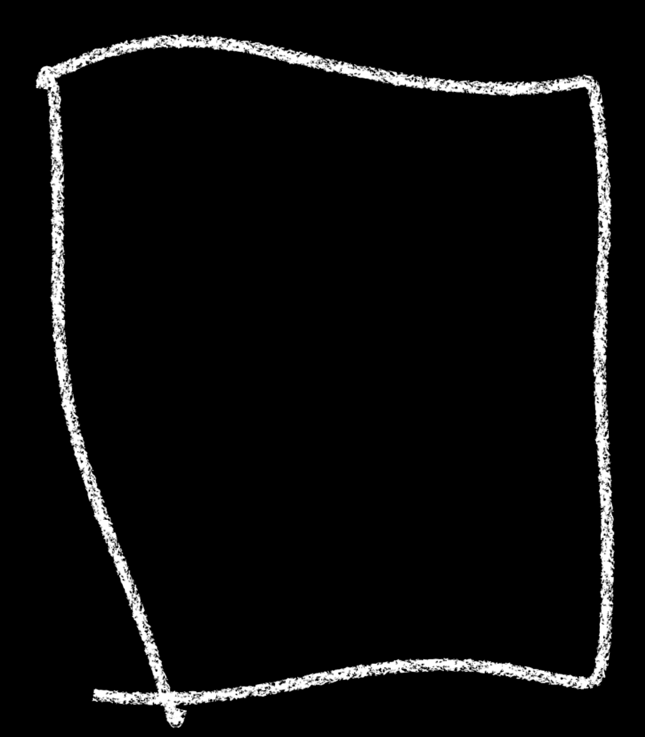


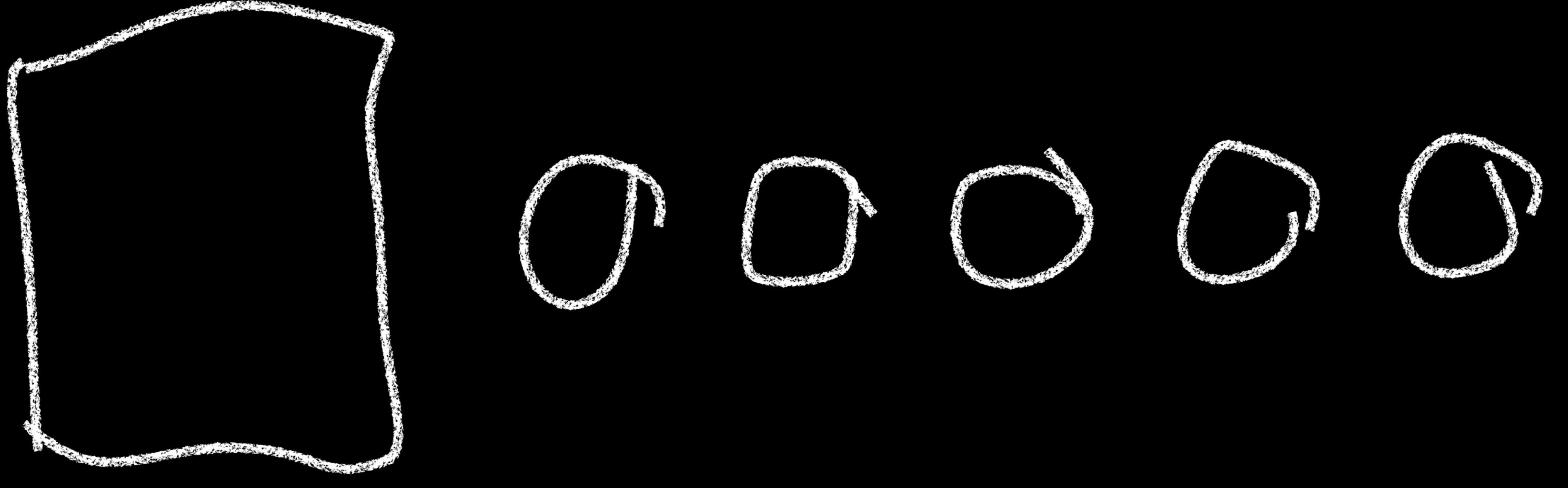


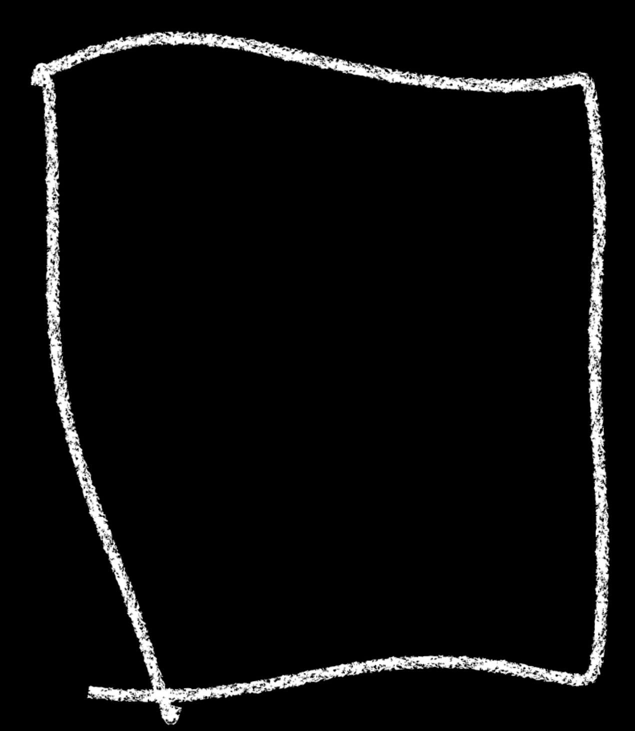


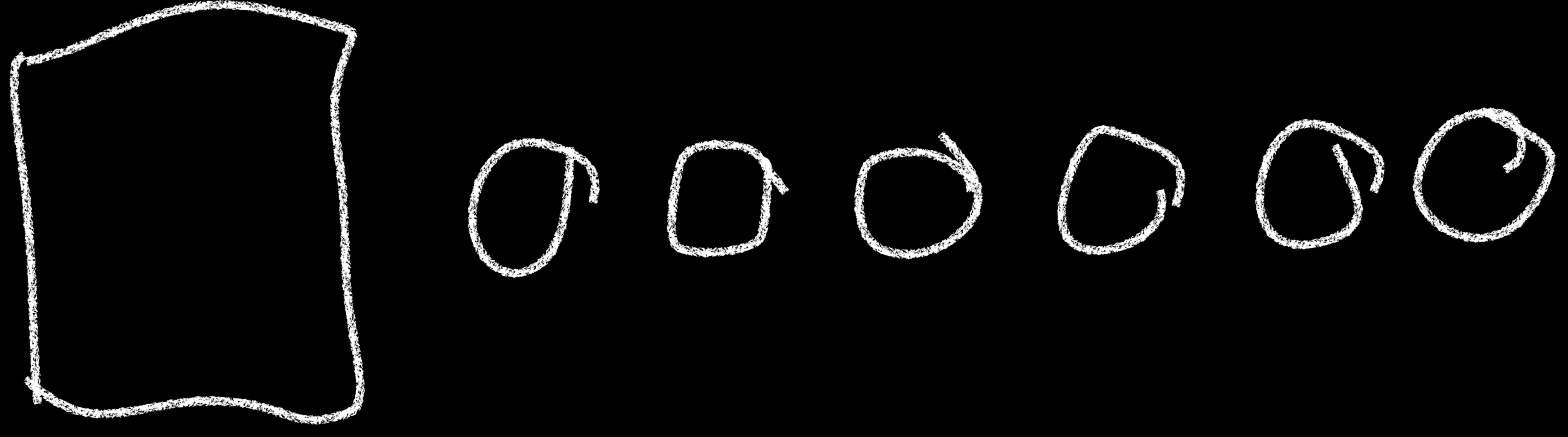
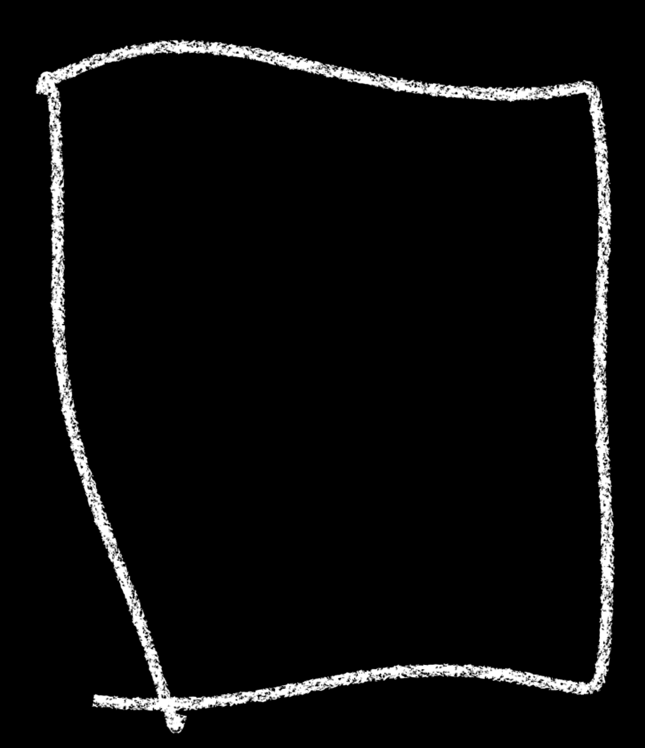


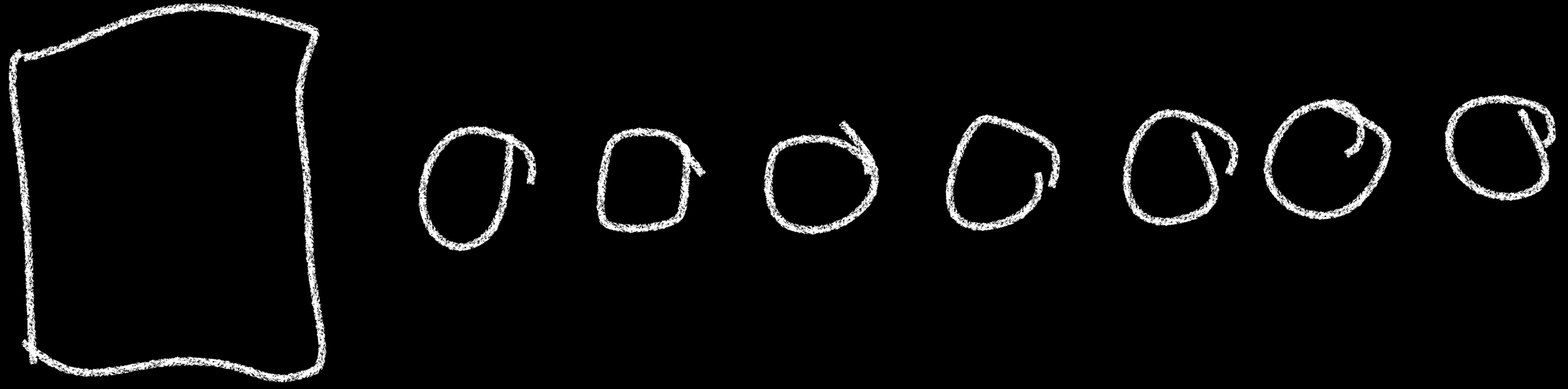


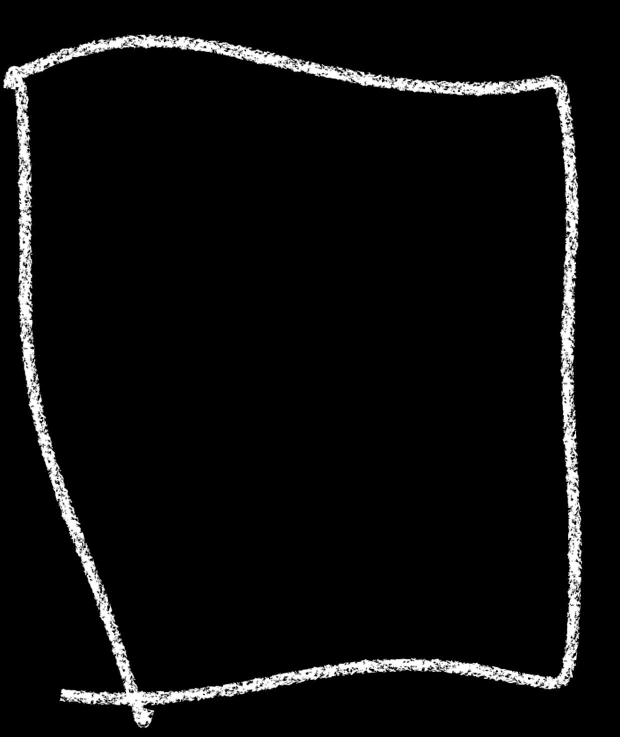


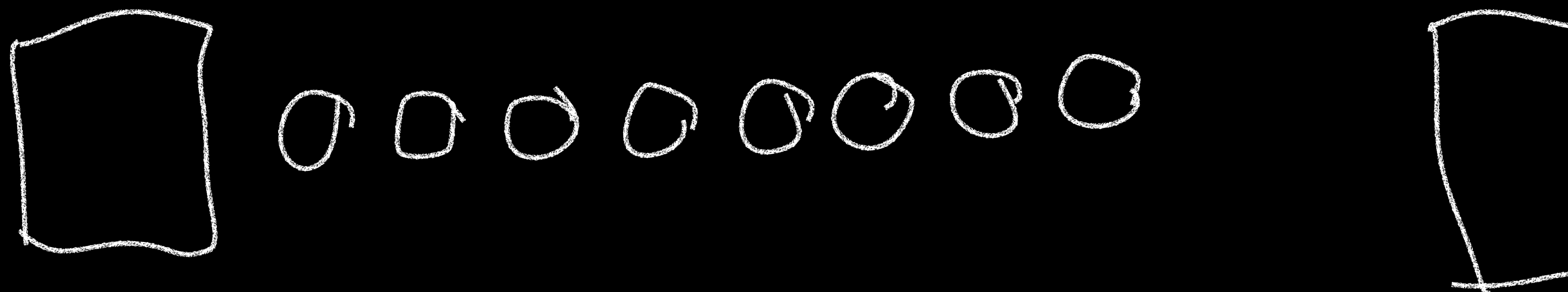


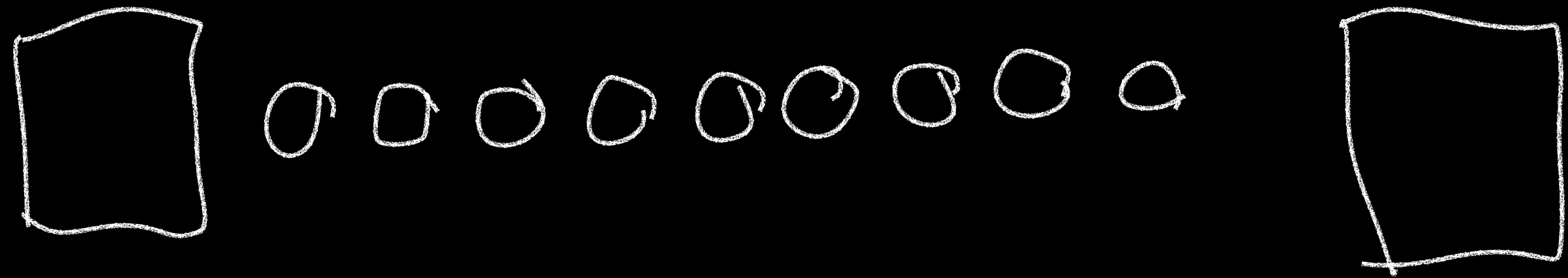


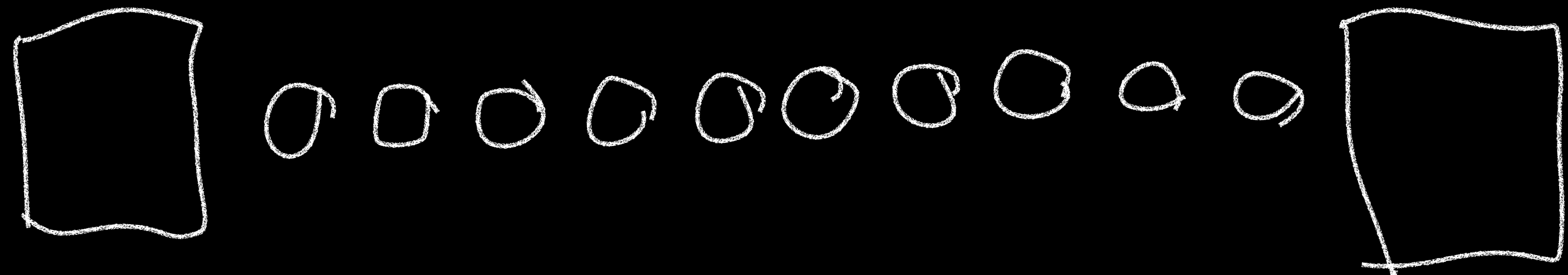



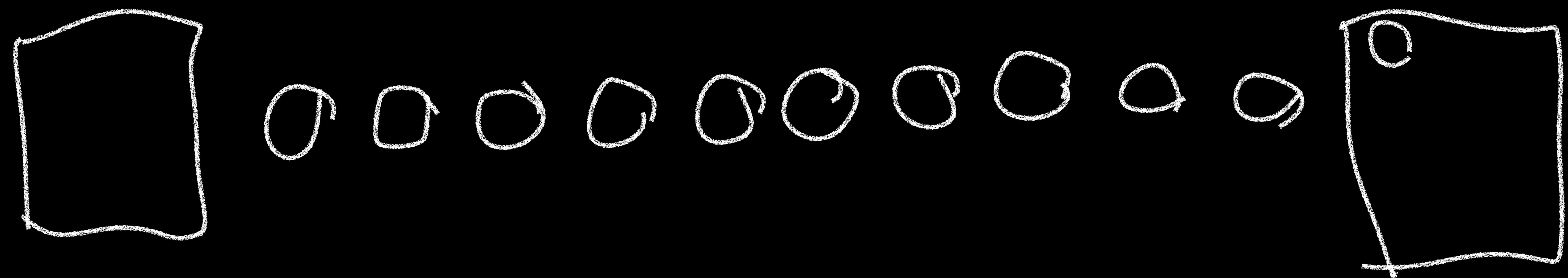


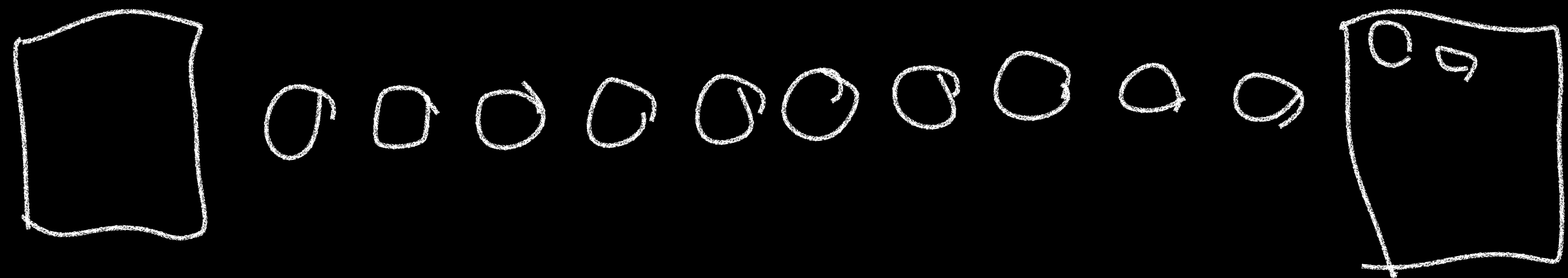


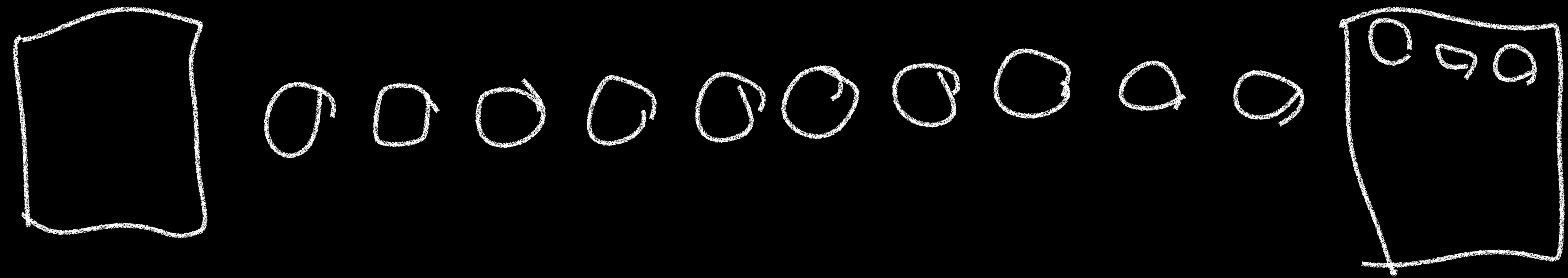


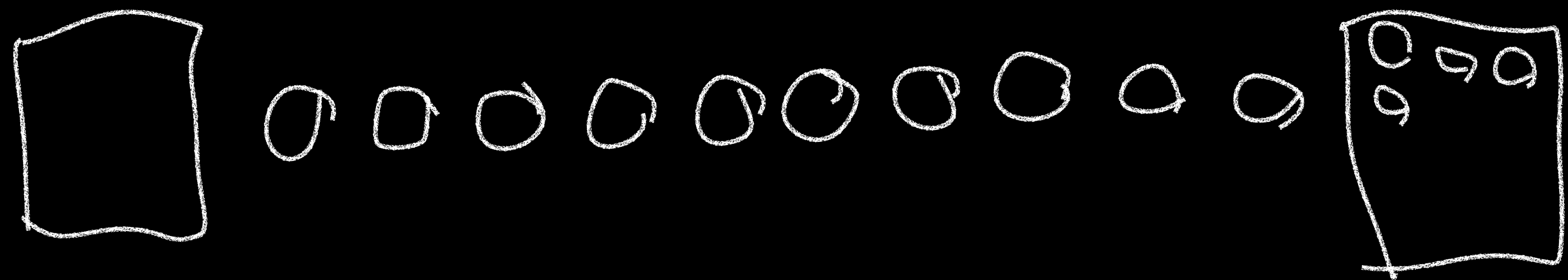


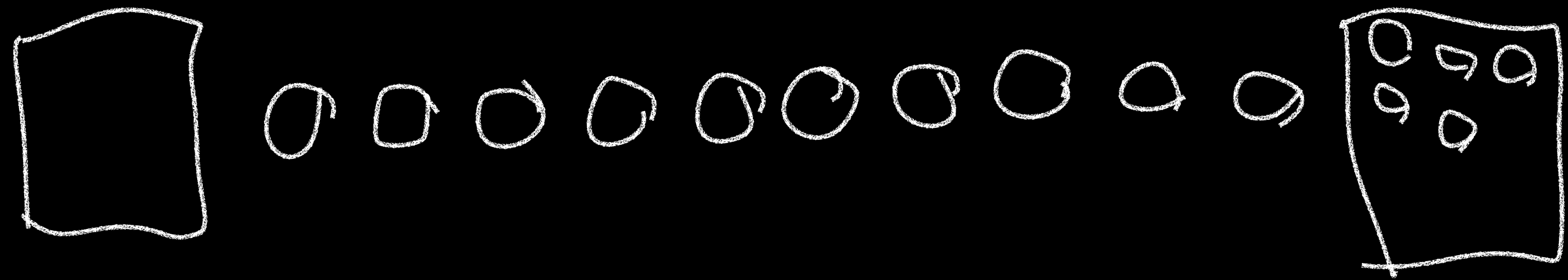


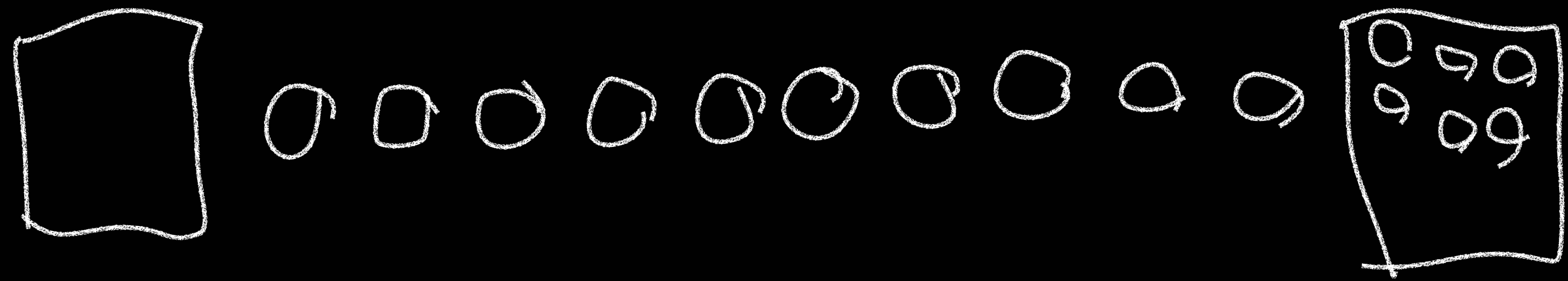


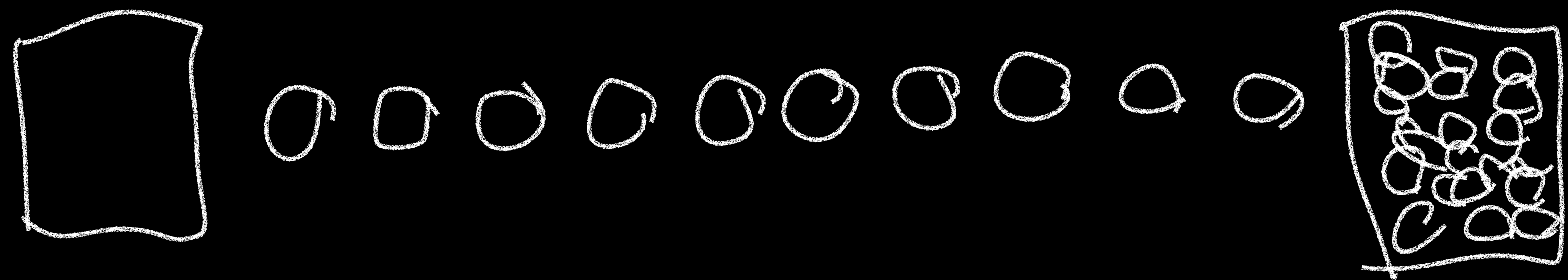


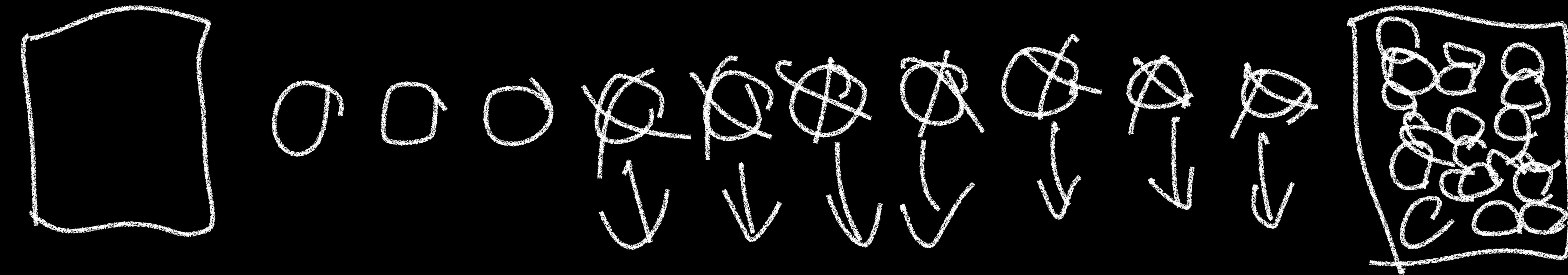


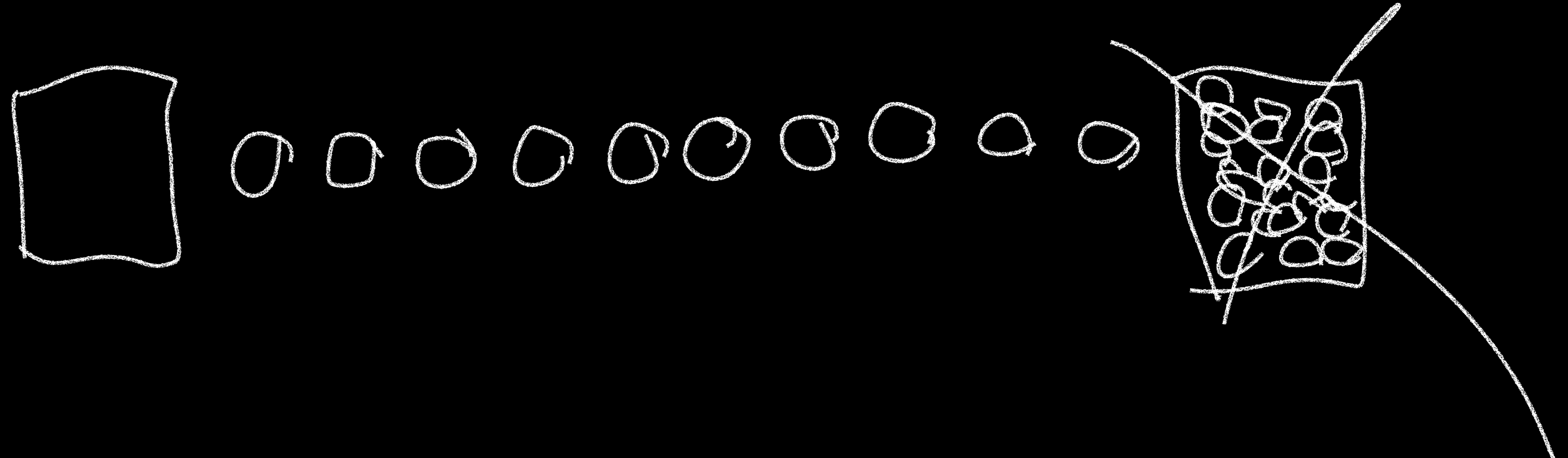




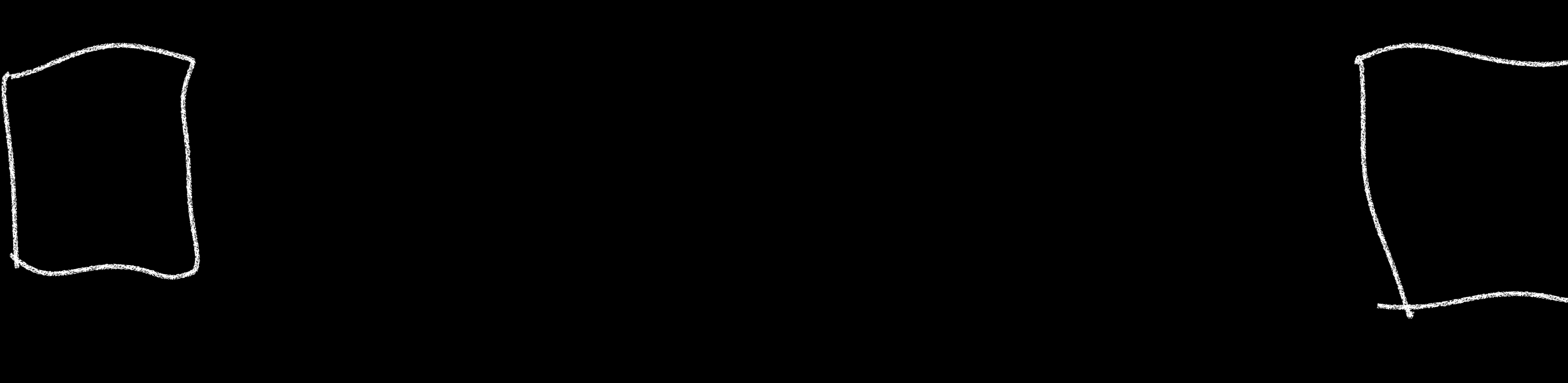




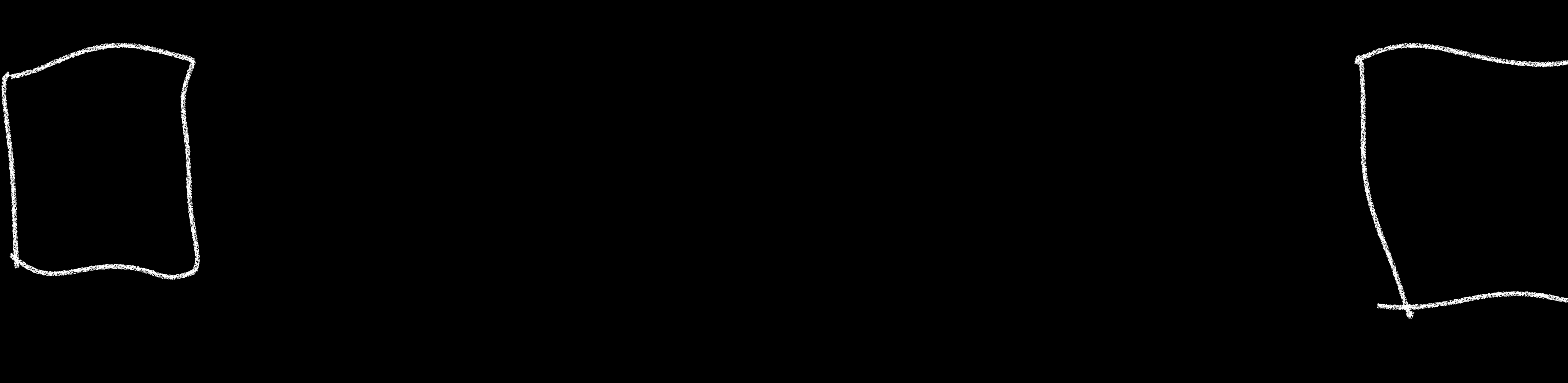


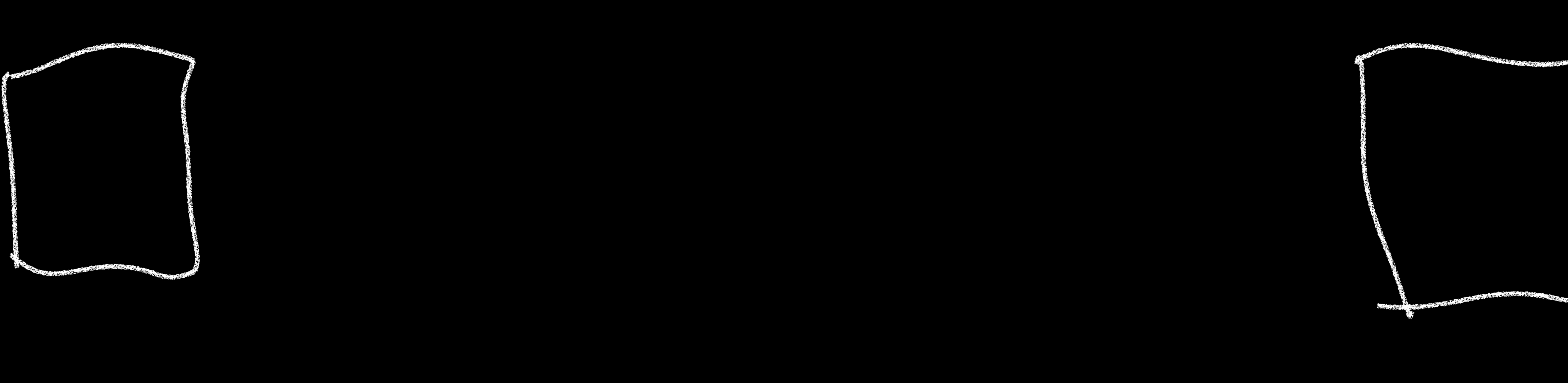


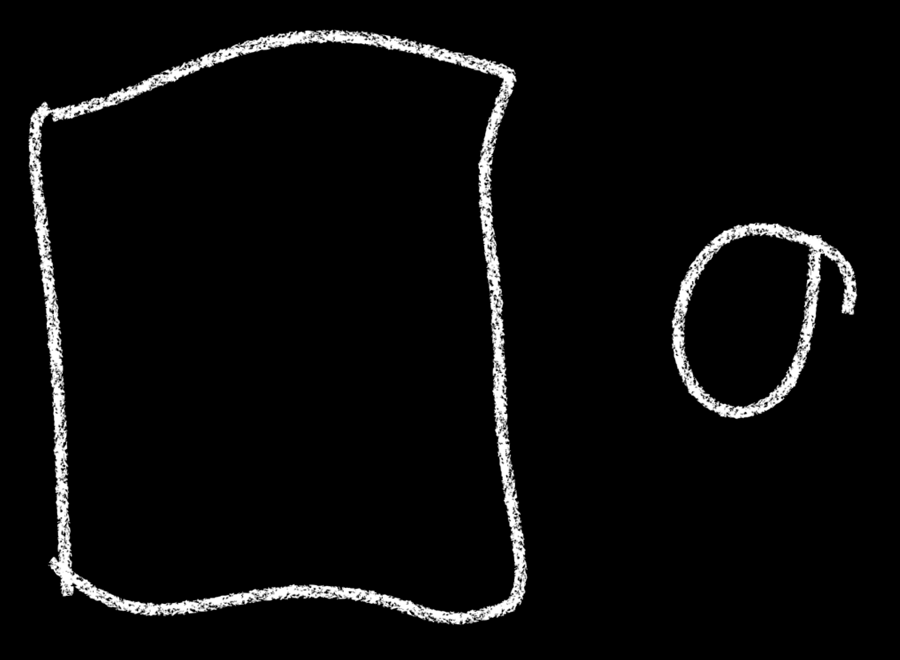
We need Backpressure

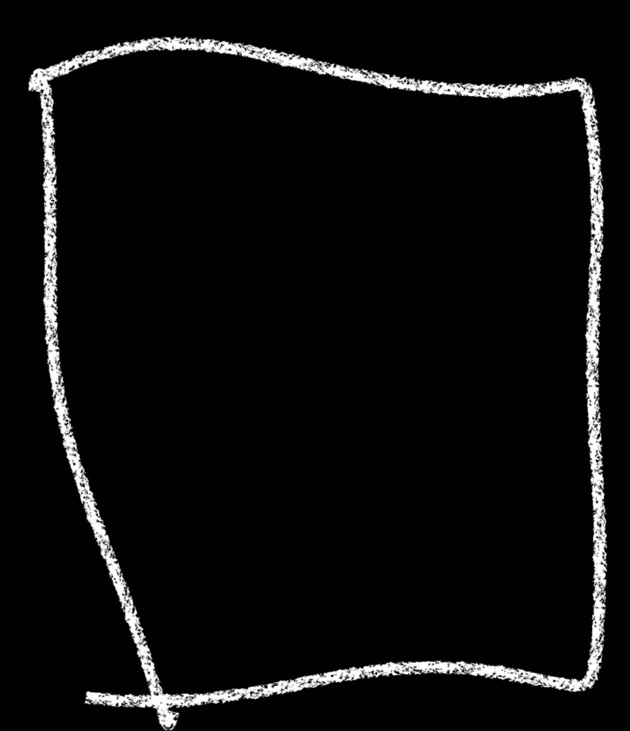


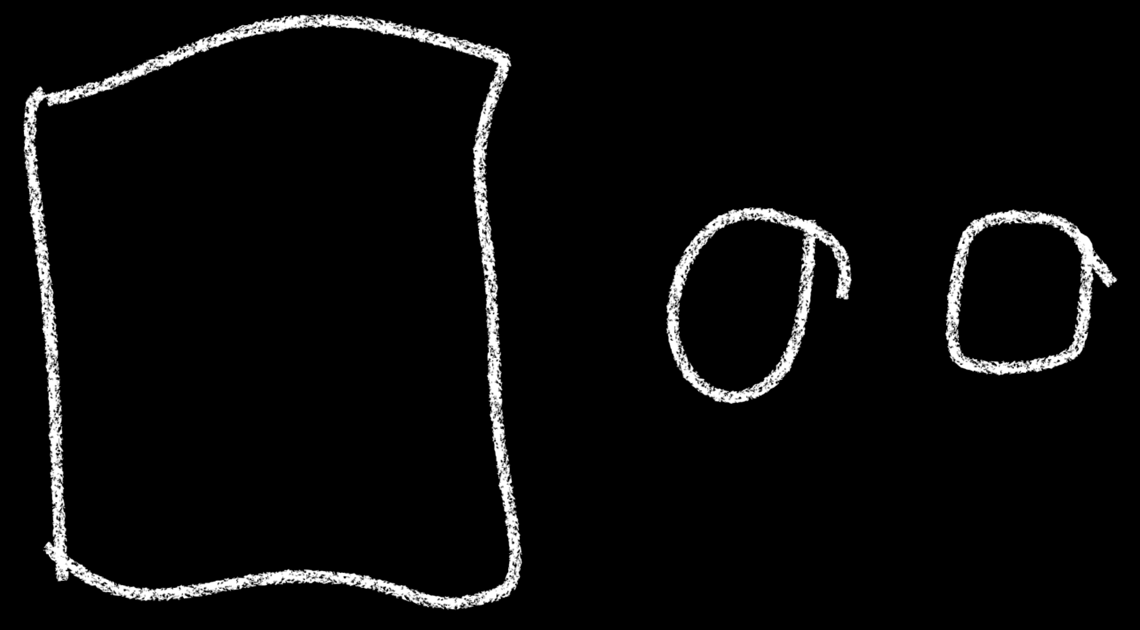
requst(5)

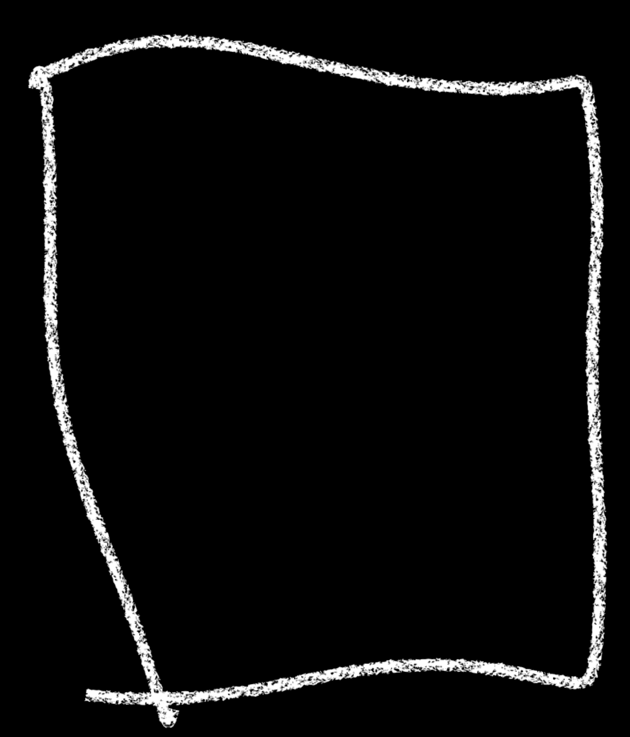


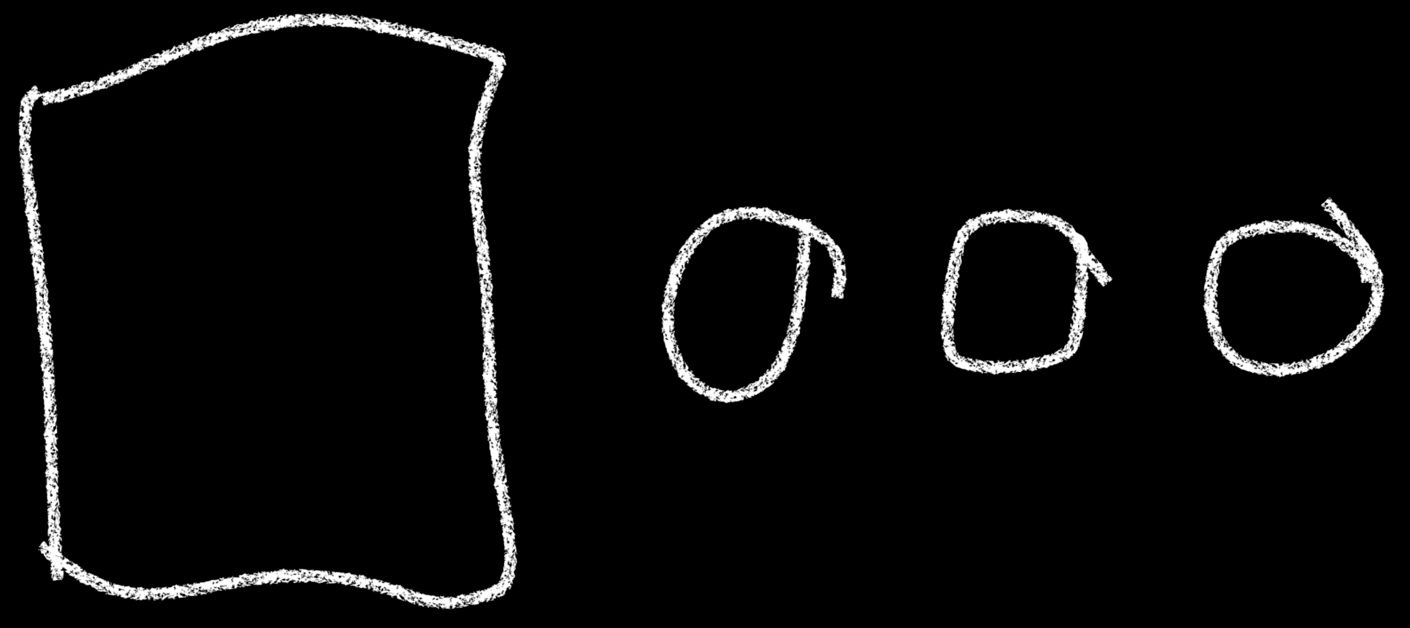


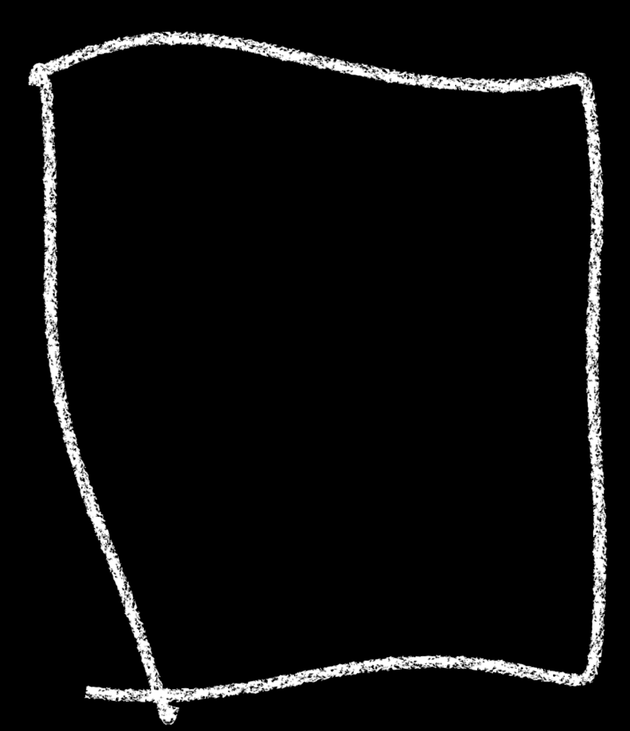

σ

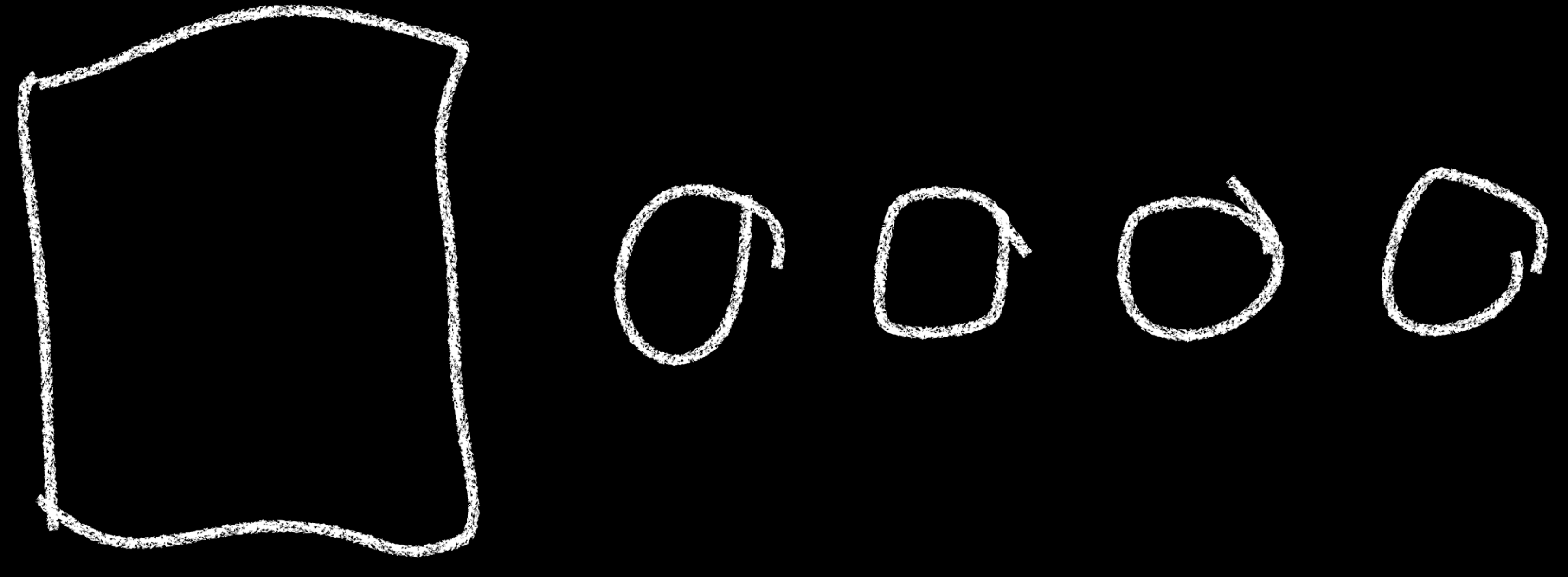


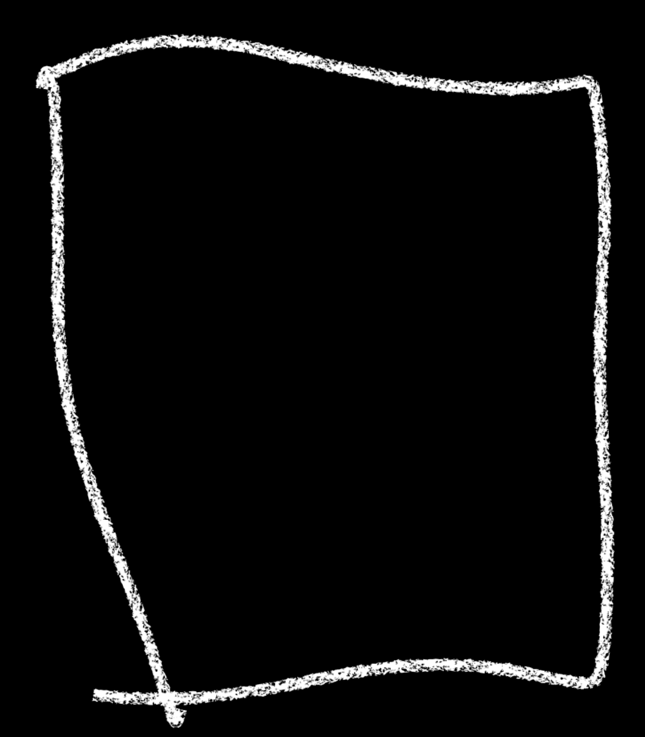


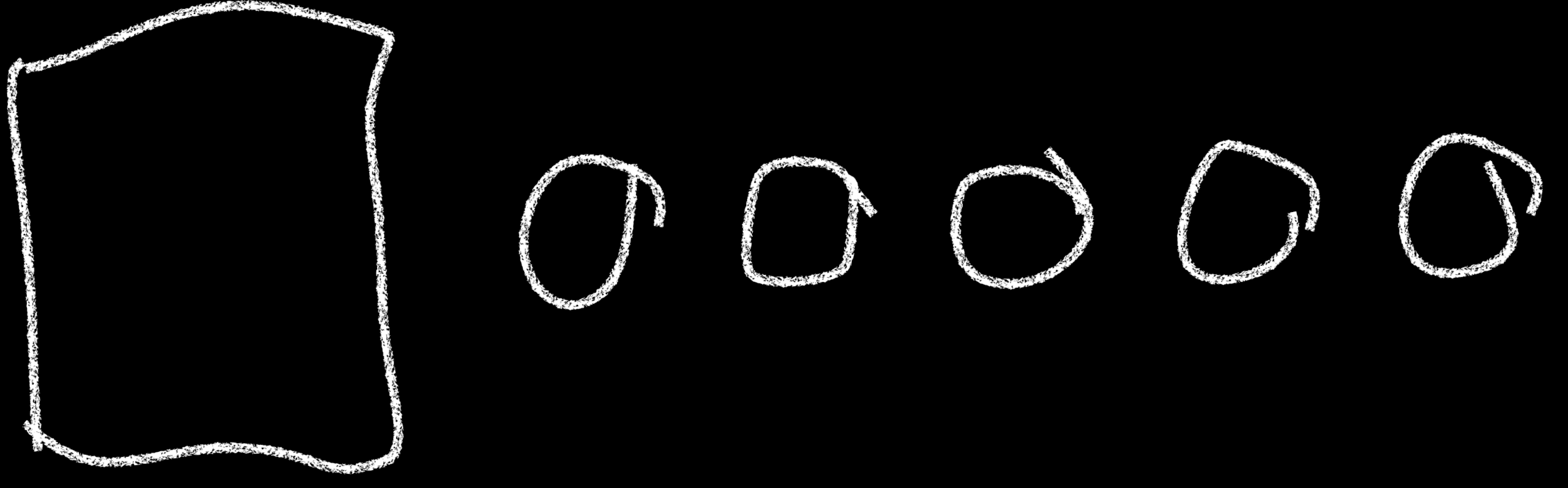


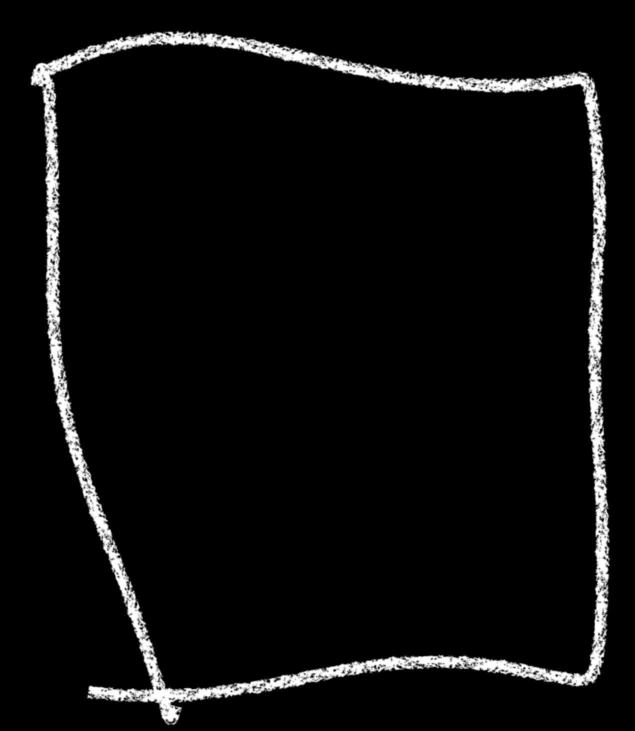


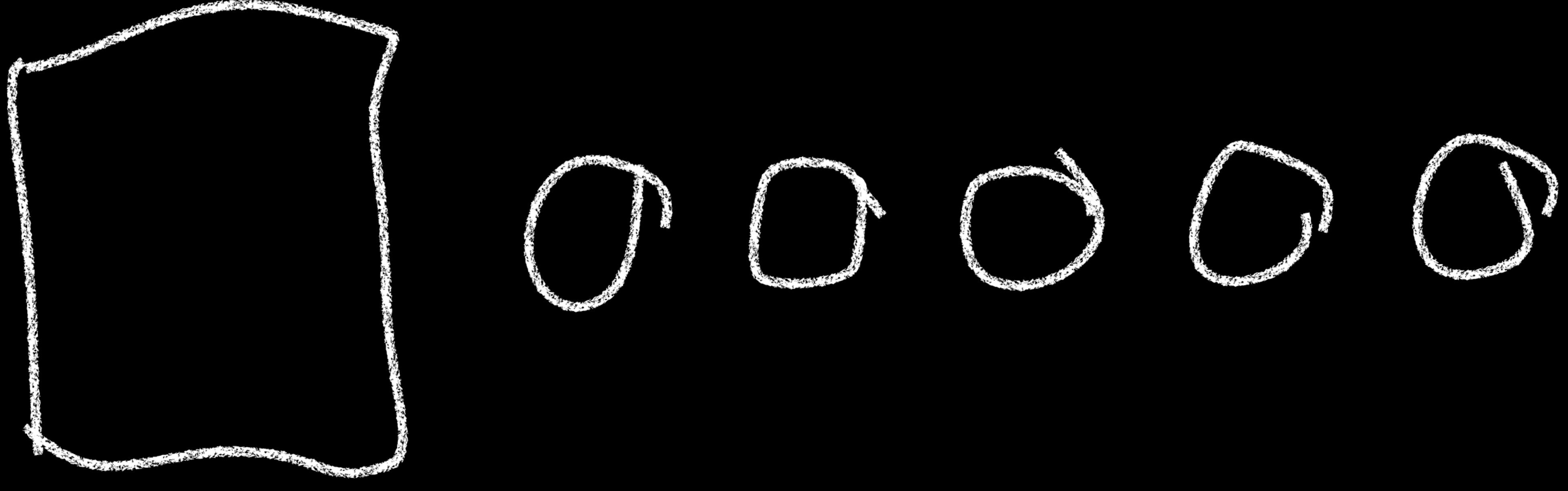
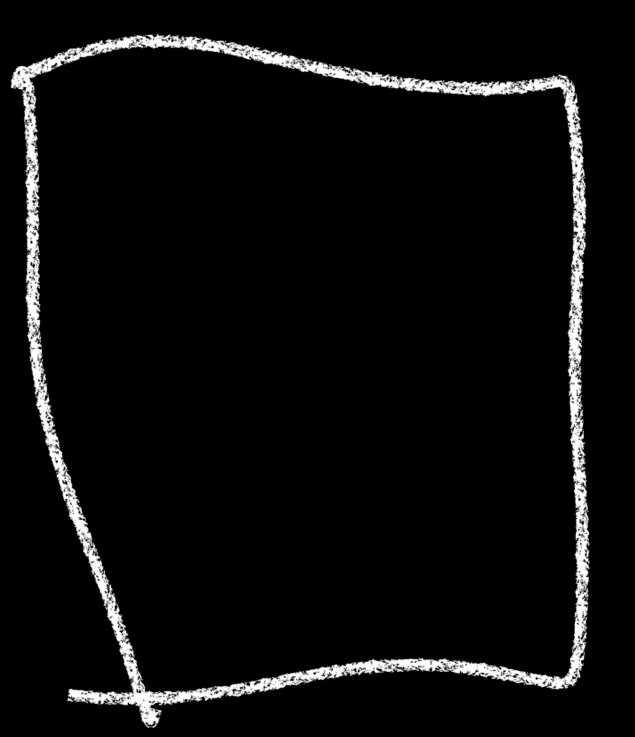


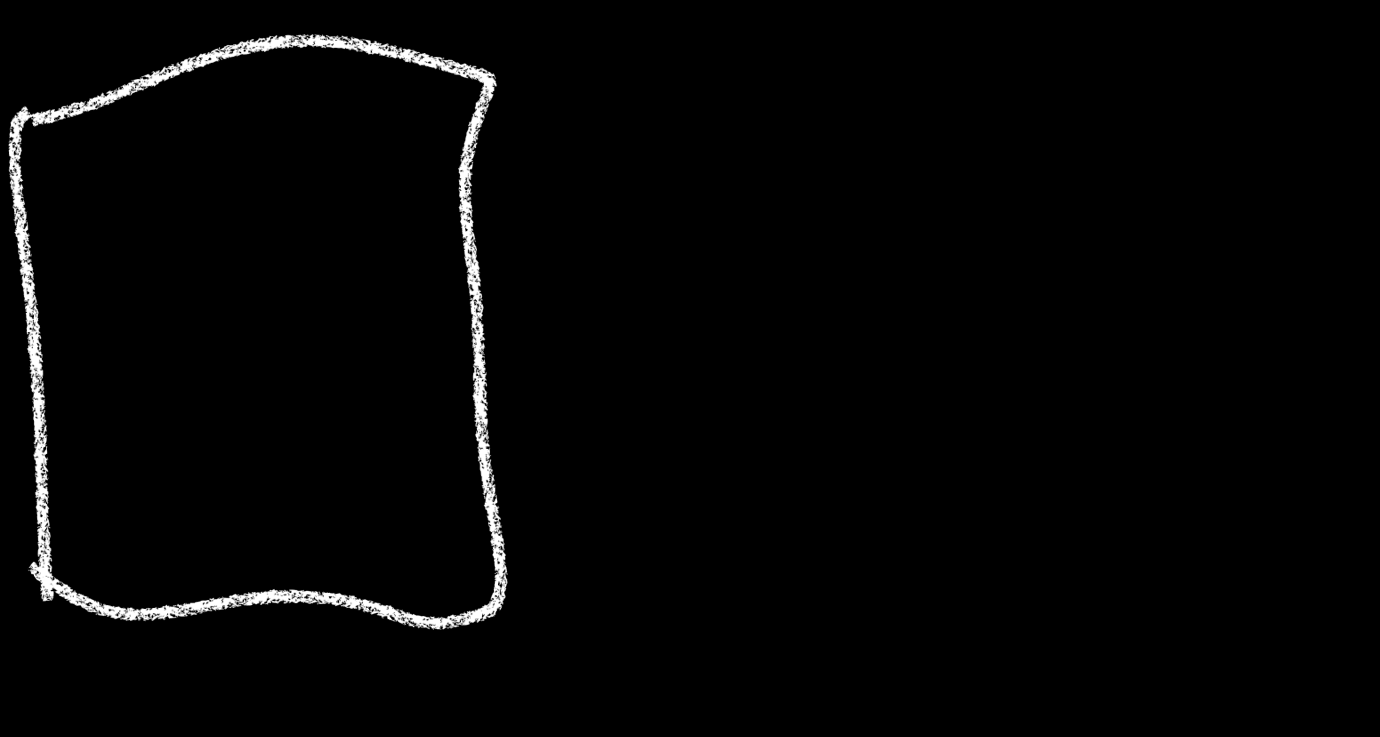


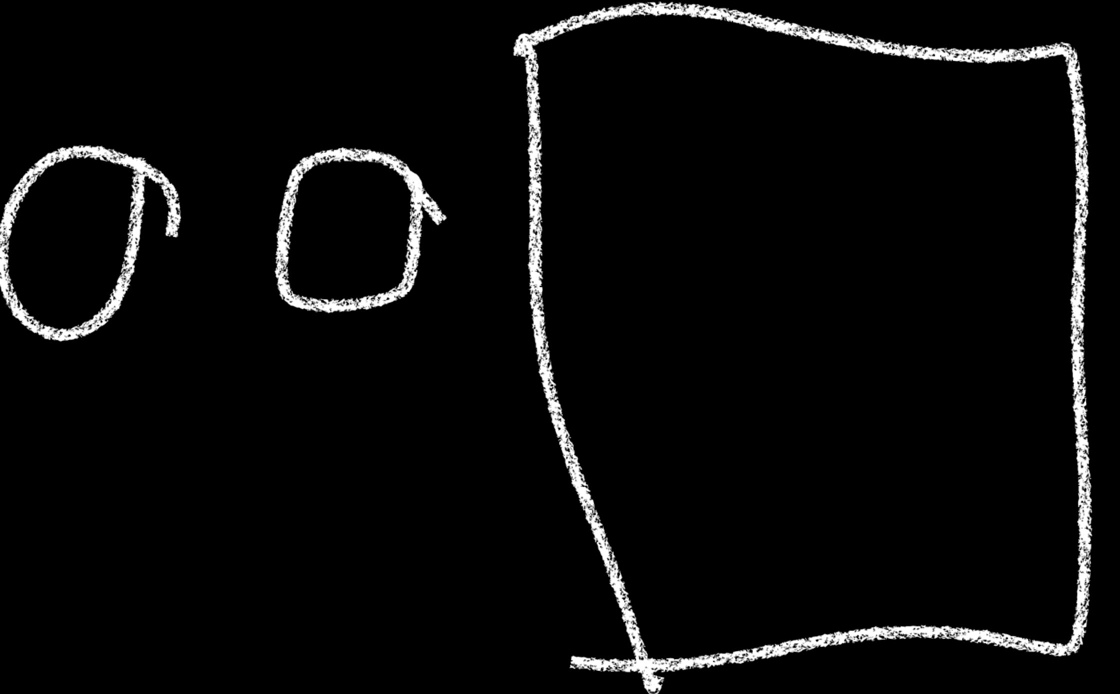




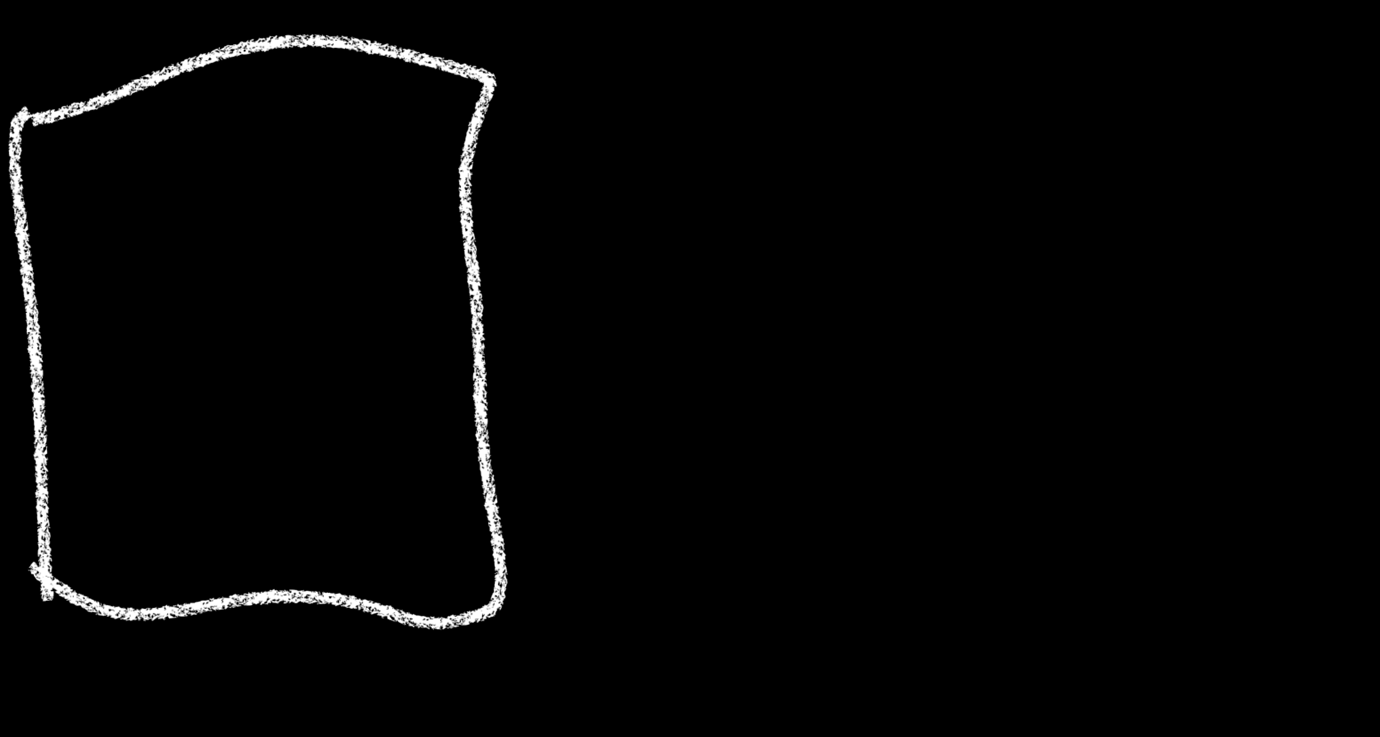


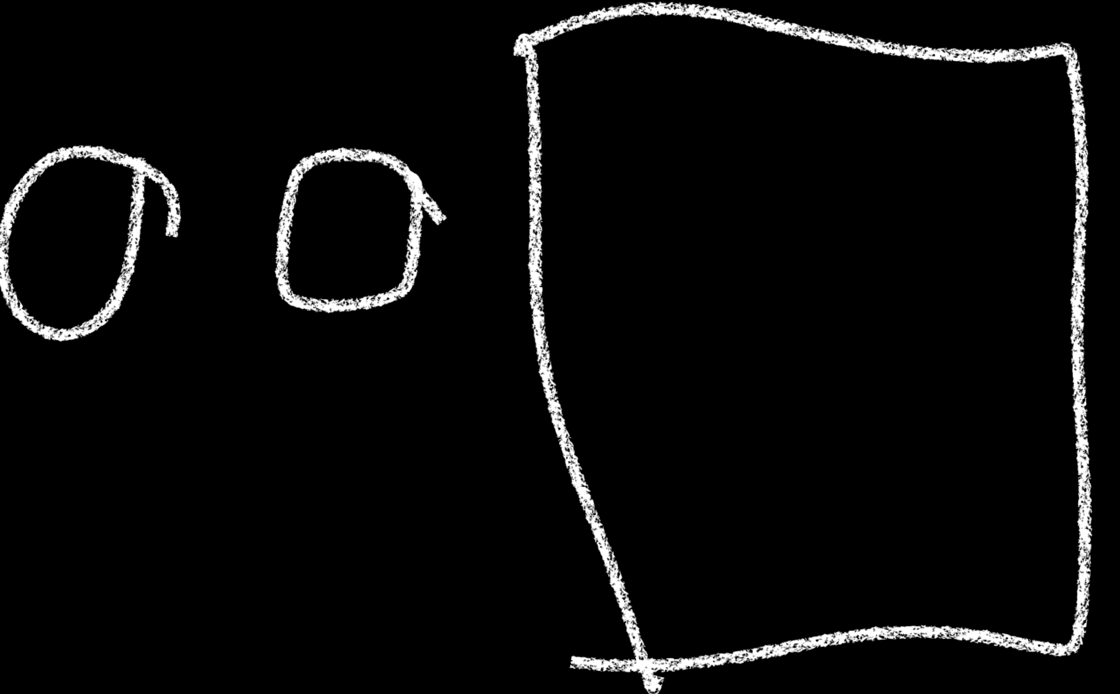







requst(10)





REACTIVE-STREAMS

```java
interface Publisher<T> {
  void subscribe(Subscriber<? super T> s);
}

    interface Subscriber<T> {
      void onSubscribe(Subscription s);
      void onNext(T t);
      void onError(Throwable t);
      void onComplete();
    }

interface Subscription {
  void request(long n);
  void cancel();
}
```

REACTIVE-STREAMS

```
interface Publisher<T> {
  void subscribe(Subscriber<? super T> s);
}

interface Subscriber<T> {
  void onSubscribe(Subscription s);
  void onNext(T t);
  void onError(Throwable t);
  void onComplete();
}

interface Subscription {
  void request(long n);
  void cancel();
}
```

PROTOCOLS

PROTOCOLS

PROTOCOLS

- HTTP/1.x

PROTOCOLS

- HTTP/1.x

PROTOCOLS

- HTTP/1.x
- HTTP/2

PROTOCOLS

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

PROTOCOLS

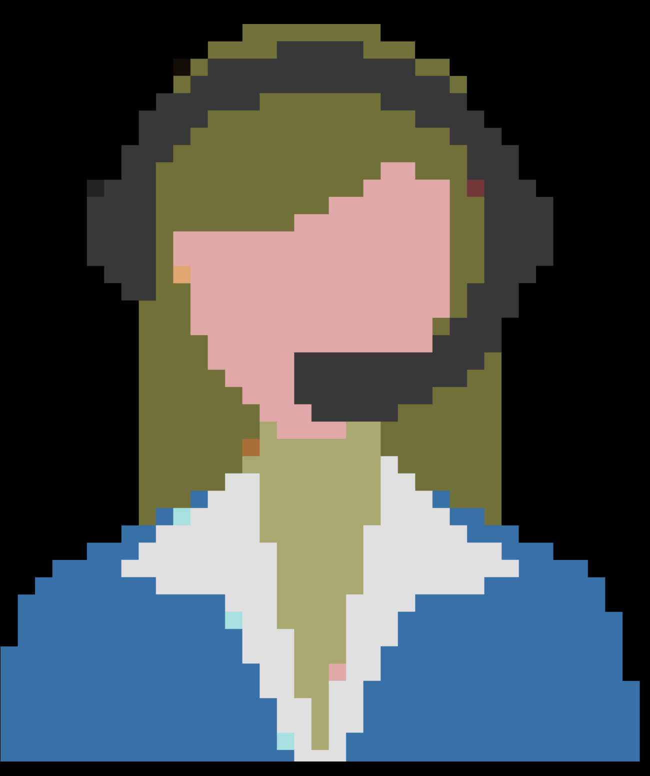
- HTTP/1.x
- HTTP/2
- WEBSOCKET

PROTOCOLS

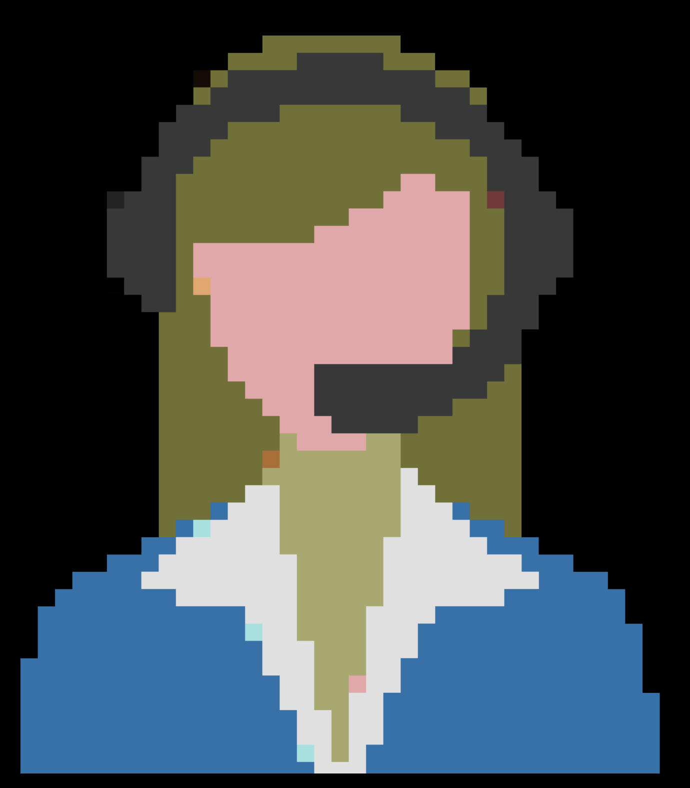
- HTTP/1.x
- HTTP/2
- WEBSOCKET
- ???

COMPARISON

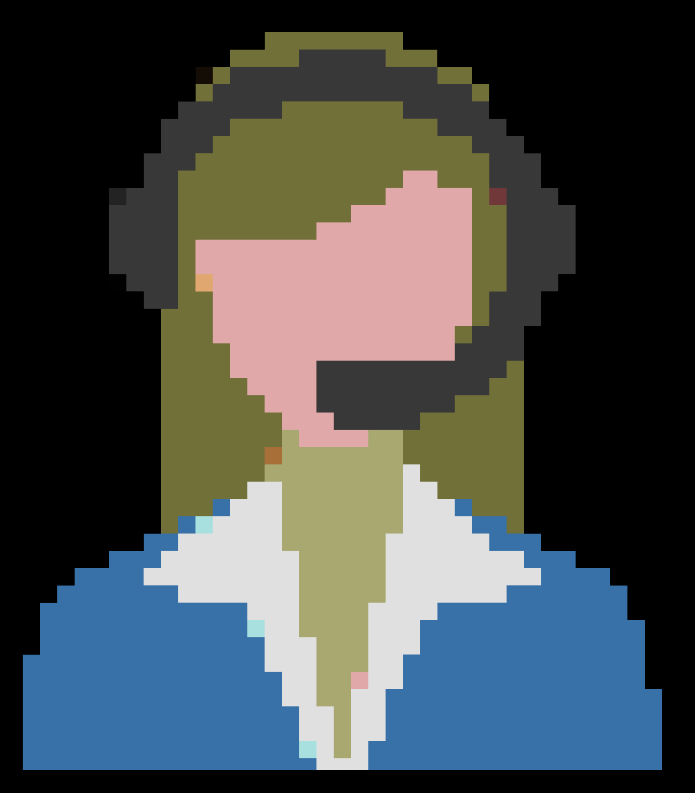
- MAINTAINABILITY
  - Frameworks
  - Community/Adoption



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



- MAINTAINABILITY
  - Frameworks
  - Community/Adoption



- STABILITY
  - Can work OK in unpredicted cases

- PERFORMANCE

WEBSOCKET

/

HTTP2

WEBSOCKET

/

HTTP2

WEBSOCKET

Why WebSocket?

Why WebSocket?

- NO OVERHEAD ~ TCP

Why WebSocket?

- NO OVERHEAD ~ TCP
- HIGH-PERFORMANCE

Why NOT WebSocket?

Why NOT WebSocket?

- COMPLEX DEVELOPMENT

Why NOT WebSocket?

- COMPLEX DEVELOPMENT
- REINVENT APPLICATION PROTOCOL

Existing Solutions

Existing Solutions

- SOCKJS + STOMP

Existing Solutions

- SOCKJS + STOMP
- SOCKET.IO

SOCKET.IO

Why Socket.IO?

Why Socket.IO?

- MOST POPULAR IN JS WORLD

Why Socket.IO?

- MOST POPULAR IN JS WORLD
- TOPIC BASED BINARY/TEXT MESSAGING

Why Socket.IO?

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

is.gd/socketio

DEMO

Why Not Socket.IO

Why Not Socket.IO

- NO INTEGRATION WITH SPRING

Why Not Socket.IO

```
final SocketIOServer server = new SocketIOServer(configuration);

context.addApplicationListener(event -> {
    if (event instanceof ContextClosedEvent || event instanceof 
ContextStoppedEvent || event instanceof ApplicationFailedEvent) {
        server.stop();
    }
});
```

Why Not Socket.IO

```
final SocketIOServer server = new SocketIOServer(configuration);

context.addApplicationListener(event -> {
    if (event instanceof ContextClosedEvent || event instanceof
ContextStoppedEvent || event instanceof ApplicationFailedEvent) {
        server.stop();
    }
});
```

Why Not Socket.IO

```
final SocketIOServer server = new SocketIOServer(configuration);

context.addApplicationListener(event -> {
    if (event instanceof ContextClosedEvent || event instanceof
ContextStoppedEvent || event instanceof ApplicationFailedEvent) {
        server.stop();
    }
});
```

Why Not Socket.IO

- NO INTEGRATION WITH SPRING
- JS (CALLBACKS) CODE STYLE

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) -> {});
```

Why Not Socket.IO

- NO INTEGRATION WITH SPRING
- JS (CALLBACKS) CODE STYLE
- NO ACCESS TO BYTEBUF

Why Not Socket.IO

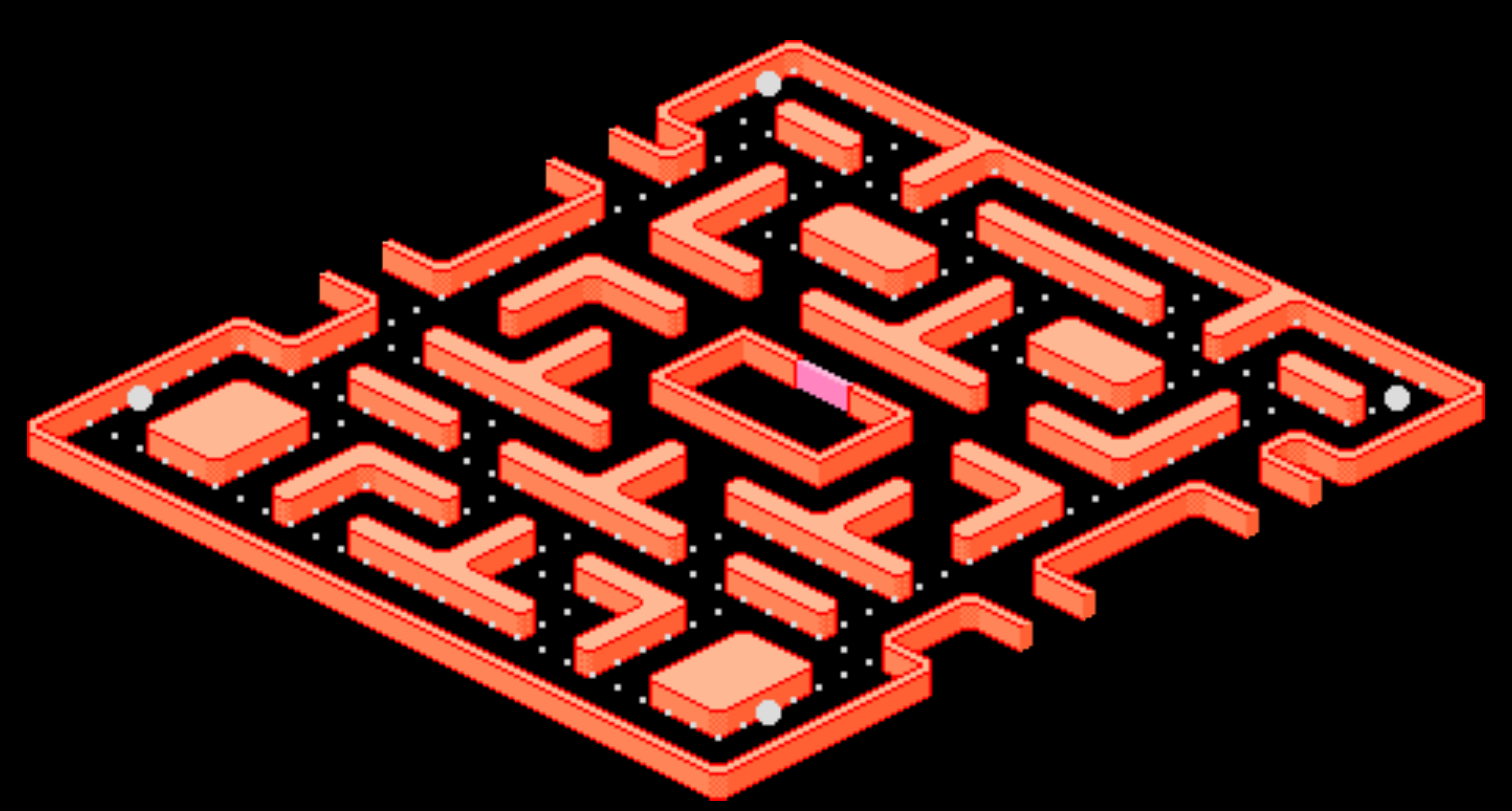
```
server.addConnectListener(client -> {});

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

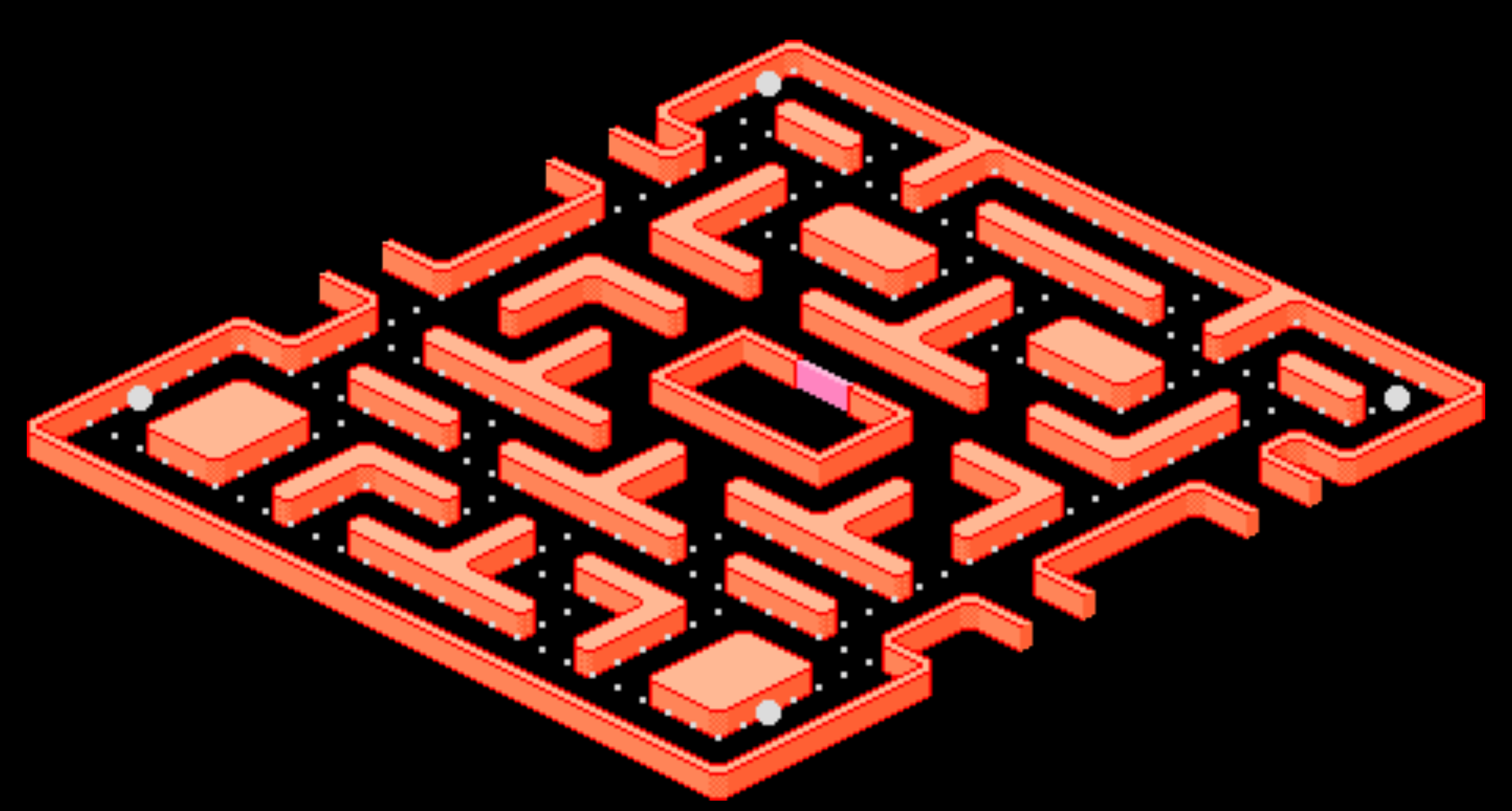
Where it is good

- REALLY GOOD AT JS

GRPC WAY



GRPC WAY



Why gRPC?

Why gRPC?

- BUILT ON TOP OF HTTP/2

Why gRPC?

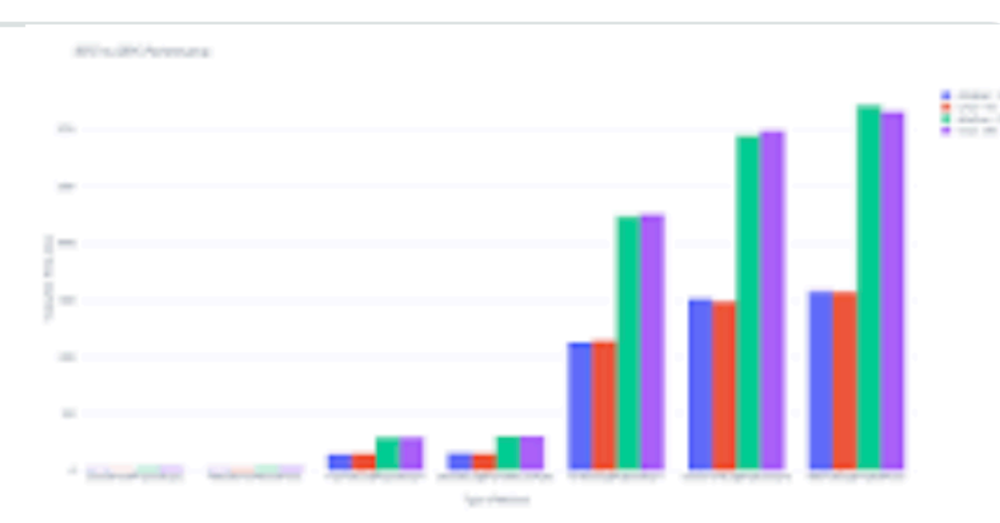
grpc performance vs rest

Все Картинки Видео Новости Покупки Ещё Настройки Инструменты

Результатов: примерно 166 000 (0,46 сек.)

Совет. По этому запросу вы можете найти сайты **на русском языке**. Указать предпочтительные языки для результатов поиска можно в разделе Настройки.

gRPC is roughly 7 times faster than **REST** when receiving data & roughly 10 times faster than **REST** when sending data for this specific payload. This is mainly due to the tight packing of the Protocol Buffers and the use of HTTP/2 by **gRPC**. 2 апр. 2019 г.



Evaluating Performance of REST vs. gRPC - Ruwan Fernando ...

https://medium.com › evaluating-performance-of-rest-vs-grpc-1b8bdf0b22da

Why gRPC?

- BUILT ON TOP OF HTTP/2
- EASY TO BUILD API WITH PROTOBUF

Why gRPC?

- BUILT ON
- EASY TO

src
@types
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
resources

Why gRPC?

- BUILT ON TOP OF HTTP/2

```
service GameService {
    rpc start (Nickname) returns (Config) {}
}

service PlayerService {
    rpc locate(stream Location) returns (google.protobuf.Empty) {}

    rpc players(google.protobuf.Empty) returns (stream Player) {}
}
```

Why gRPC?

- BUILT ON TOP OF HTTP/2

```
service GameService {
    rpc start (Nickname) returns (Config) {}
}

service PlayerService {
    rpc locate(stream Location) returns (google.protobuf.Empty) {}

    rpc players(google.protobuf.Empty) returns (stream Player) {}
}
```

Why gRPC?

- BUILT ON TOP OF HTTP/2

```
service GameService {
    rpc start (Nickname) returns (Config) {}
}

service PlayerService {
    rpc locate(stream Location) returns (google.protobuf.Empty) {}

    rpc players(google.protobuf.Empty) returns (stream Player) {}
}
```

Why gRPC?

- BUILT ON TOP OF HTTP/2

```
service GameService {
    rpc start (Nickname) returns (Config) {}
}

service PlayerService {
    rpc locate(stream Location) returns (google.protobuf.Empty) {}

    rpc players(google.protobuf.Empty) returns (stream Player) {}
}
```

Why gRPC?

- BUILT ON TOP OF HTTP/2
- EASY TO BUILD API WITH PROTOBUF
- GOOD DEVELOPMENT EXPERIENCE

Why gRPC?

- BUILT (
- EASY T
- GOOD

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

  plugins {
    grpc {
      artifact = "io.grpc:protoc-gen-grpc-java"
    }
  }

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

Why gRPC?

- BUILT ON
- EASY TO
- GOOD DE

```
▼ src
  ▶ @types
  ▼ generated
    ▼ main
      ▶ grpc
      ▶ java
      ▼ reactorGRpc
        ▼ org.coinen.pacman
            ReactorExtrasServiceGrpc
            ReactorGameServiceGrpc
            ReactorLocationServiceGrpc
            ReactorMapServiceGrpc
            ReactorPlayerServiceGrpc
            ReactorScoreServiceGrpc
            ReactorSetupServiceGrpc
```

Why gRPC?

- BUILT ON TOP OF HTTP/2
- EASY TO BUILD API WITH PROTOBUF
- GOOD DEVELOPMENT EXPERIENCE
- SEAMLESS INTEGRATION WITH SPRING

Why gRPC?

- BUILT ON TOP OF HTTP/2

```
@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()));
    }
}
```

Why gRPC?

- BUILT ON TOP OF HTTP/2

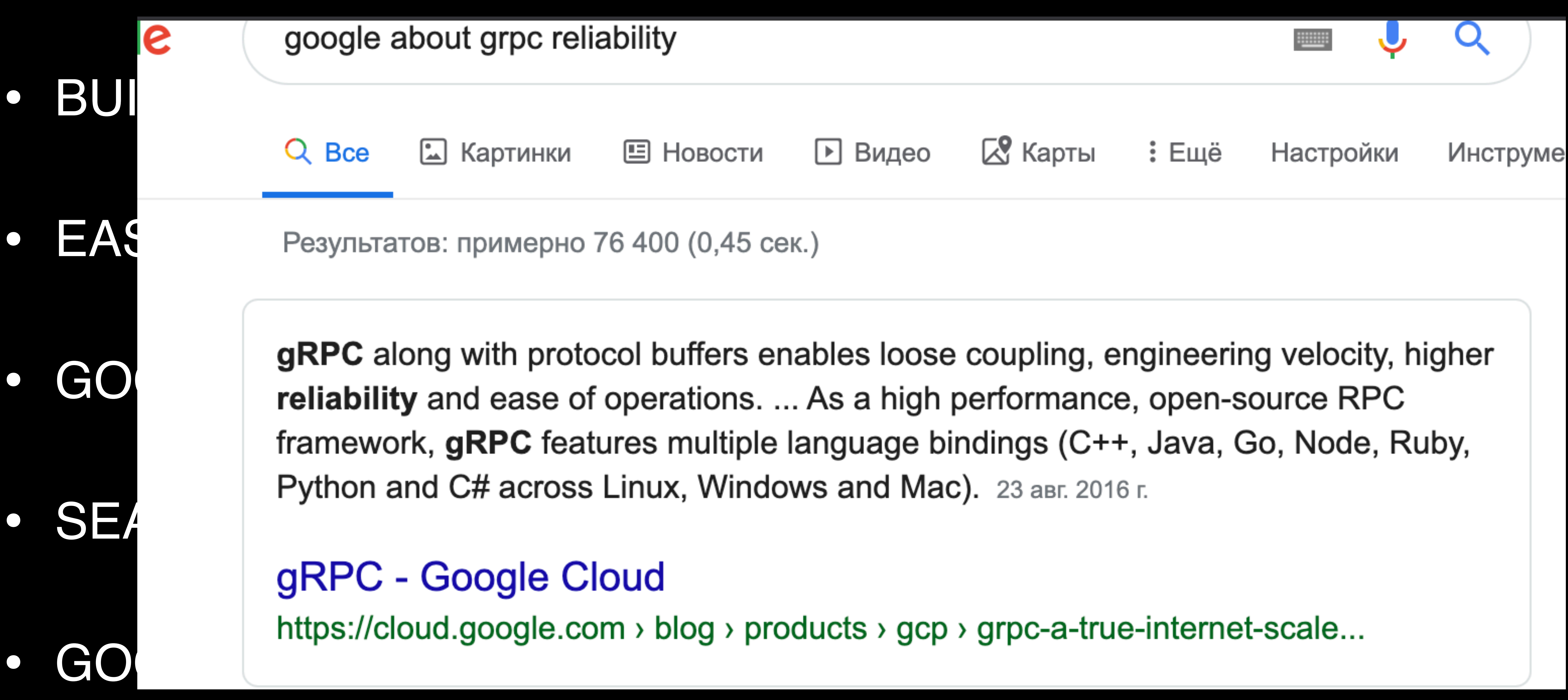
```
@GRpcService
public class GrpcPlayerController extends ReactorPlayerServiceGrpc.PlayerServiceImplBase {

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

Why gRPC?

- BUILT ON TOP OF HTTP/2
- EASY TO BUILD API WITH PROTOBUF
- GOOD DEVELOPMENT EXPERIENCE
- SEAMLESS INTEGRATION WITH SPRING
- GOOGLE SAYS IT IS SUPER STABLE

Why gRPC?

- BUI
- EAS
- GO
- SEA
- GO



is.gd/rgrpc

DEMO

Why I’m still lagging?

Leaderboard
1. OlegDokuka - 0

Leaderboard
1. OlegDokuka - 0

LIMITATIONS

LIMITATIONS

Hypertext Transfer Protocol Version 2 (HTTP/2)

draft-ietf-httpbis-http2-latest

Abstract

This specification describes an optimized expression of the semantics of the Hypertext Transfer Protocol (HTTP), referred to as HTTP version 2 (HTTP/2). HTTP/2 enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection. It also introduces unsolicited push of representations from servers to clients.

This specification is an alternative to, but does not obsolete, the HTTP/1.1 message syntax. HTTP's existing semantics remain unchanged.

LIMITATIONS

8. HTTP Message Exchanges

HTTP/2 is intended to be as compatible as possible with current uses of HTTP. This means that, from the application perspective, the features of the protocol are largely unchanged. To achieve this, all request and response semantics are preserved, although the syntax of conveying those semantics has changed.

Thus, the specification and requirements of HTTP/1.1 Semantics and Content [RFC7231], Conditional Requests [RFC7232], Range Requests [RFC7233], Caching [RFC7234], and Authentication [RFC7235] are applicable to HTTP/2. Selected portions of HTTP/1.1 Message Syntax and Routing [RFC7230], such as the HTTP and HTTPS URI schemes, are also applicable in HTTP/2, but the expression of those semantics for this protocol are defined in the sections below.

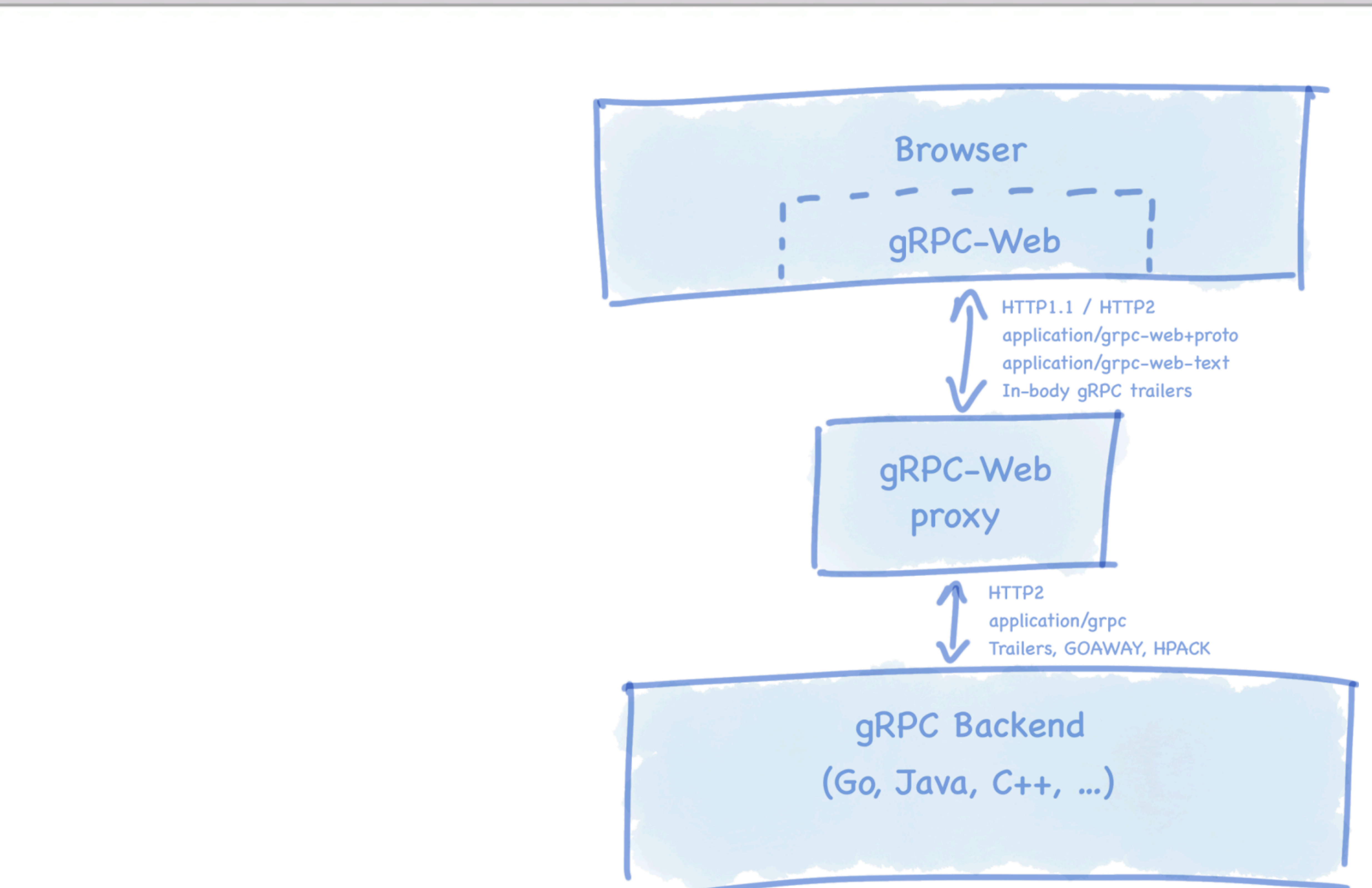
LIMITATIONS

8.2.1 Push Requests

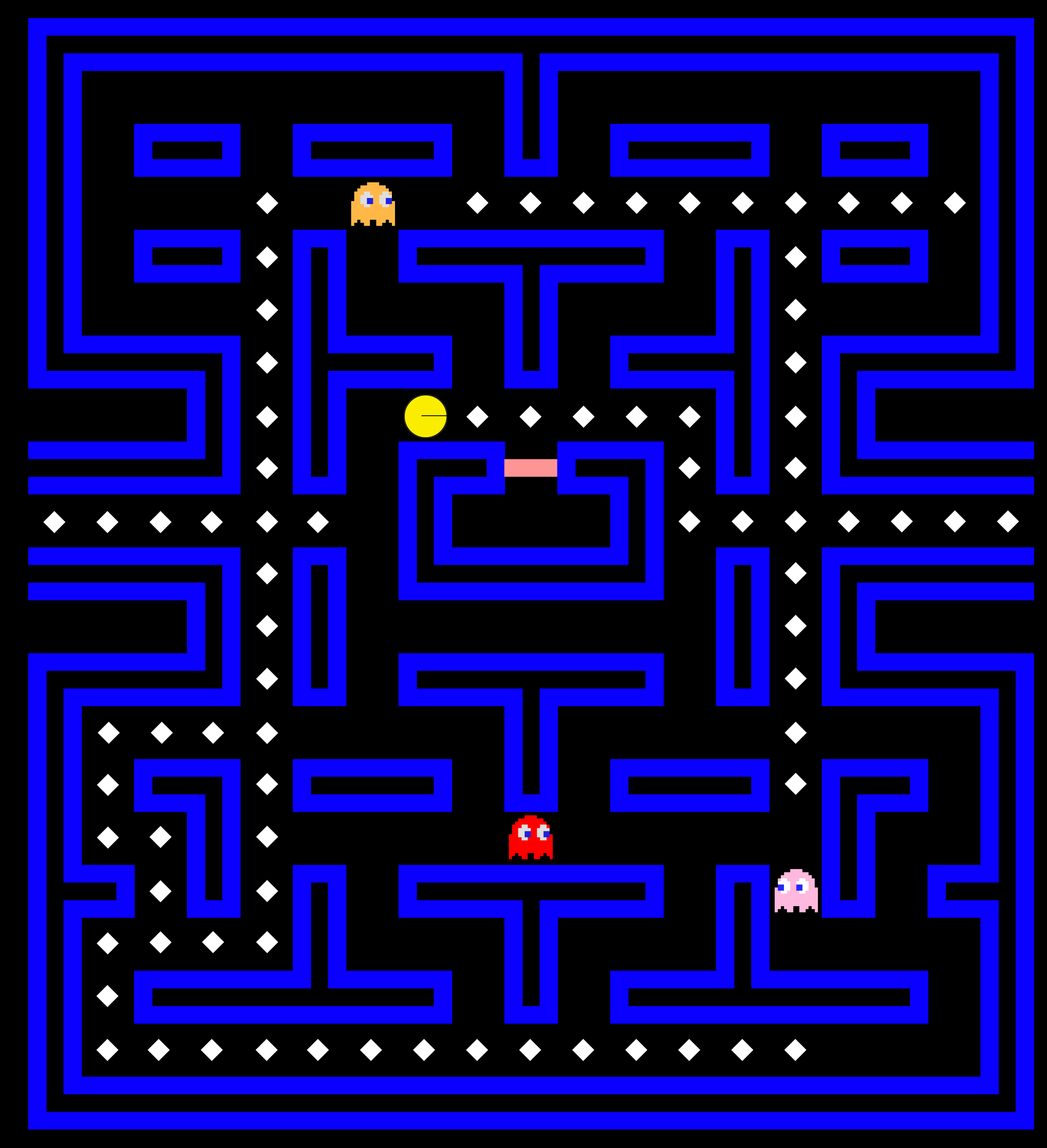
Server push is semantically equivalent to a server responding to a request; however, in this case, that request is also sent by the server, as a PUSH_PROMISE frame.

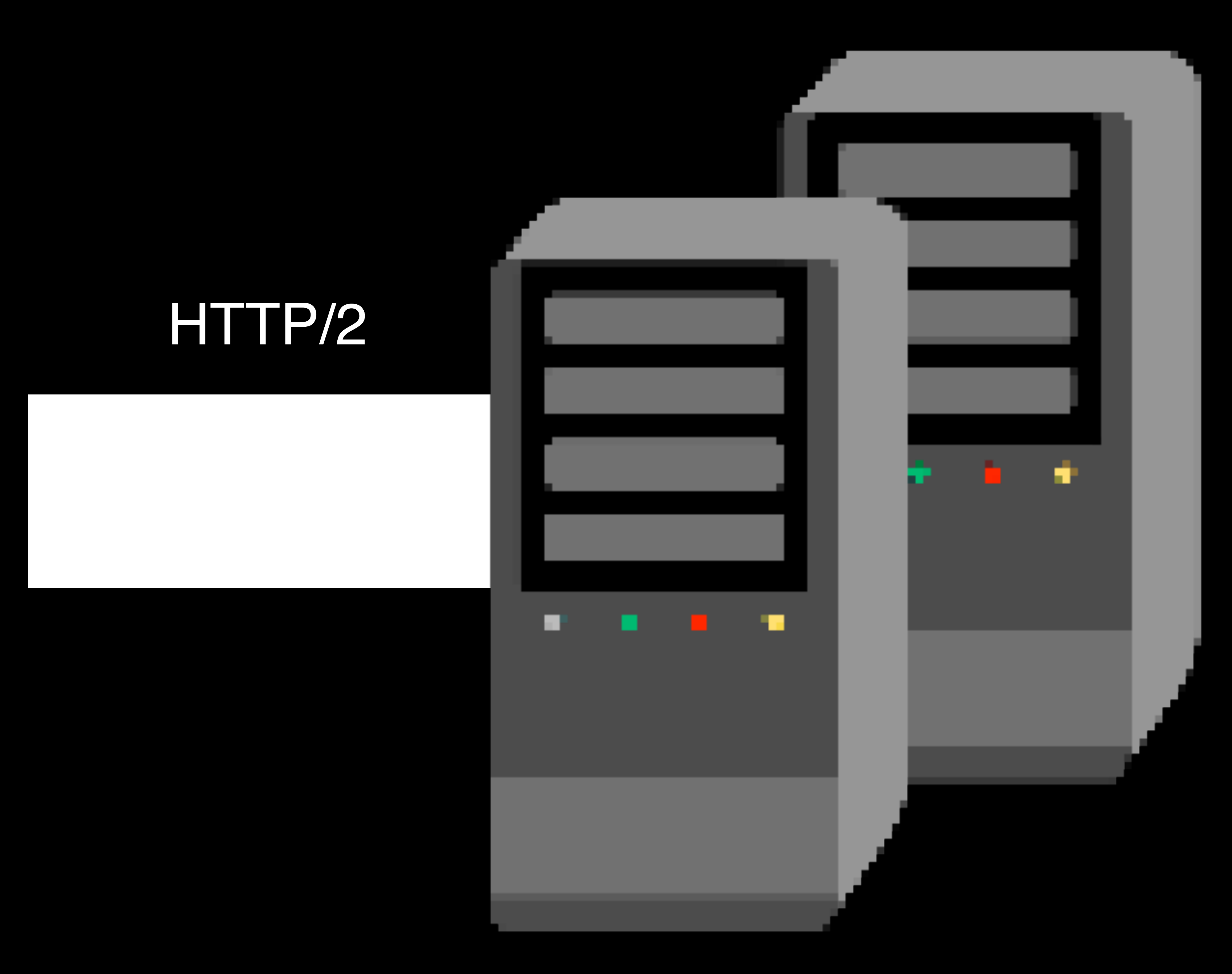
The PUSH_PROMISE frame includes a header block that contains a complete set of request header fields that the server attributes to the request. It is not possible to push a response to a request that includes a request body.

Pushed responses are always associated with an explicit request from the client. The PUSH_PROMISE frames sent by the server are sent on that explicit request's stream. The PUSH_PROMISE frame also includes a promised stream identifier, chosen from the stream identifiers available to the server (see Section 5.1.1).

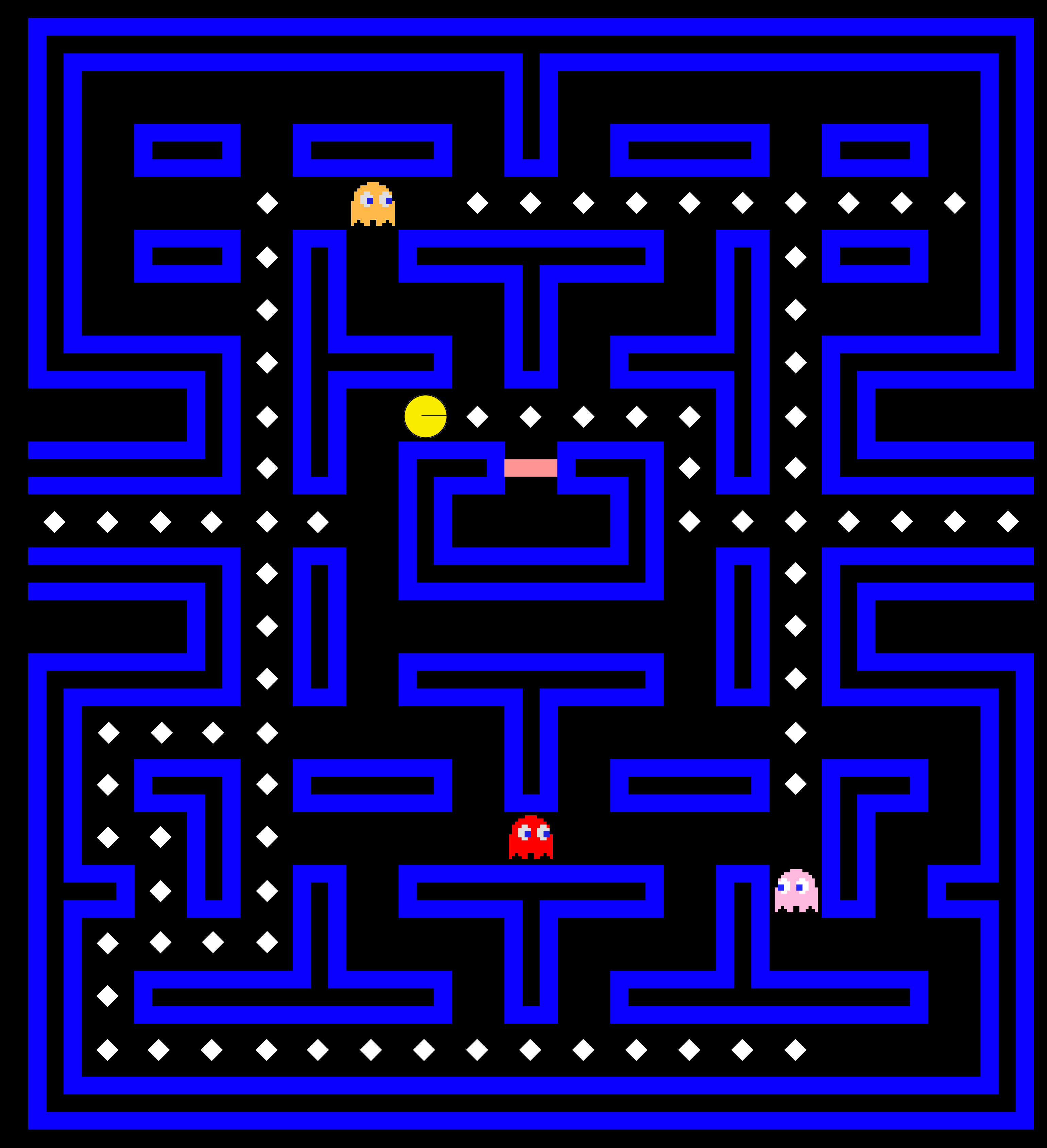

Browser
gRPC-Web
HTTP1.1 / HTTP2
application/grpc-web+proto
application/grpc-web-text
In-body gRPC trailers
gRPC-Web
proxy
HTTP2
application/grpc
Trailers, GOAWAY, HPACK
gRPC Backend
(Go, Java, C++, ...)

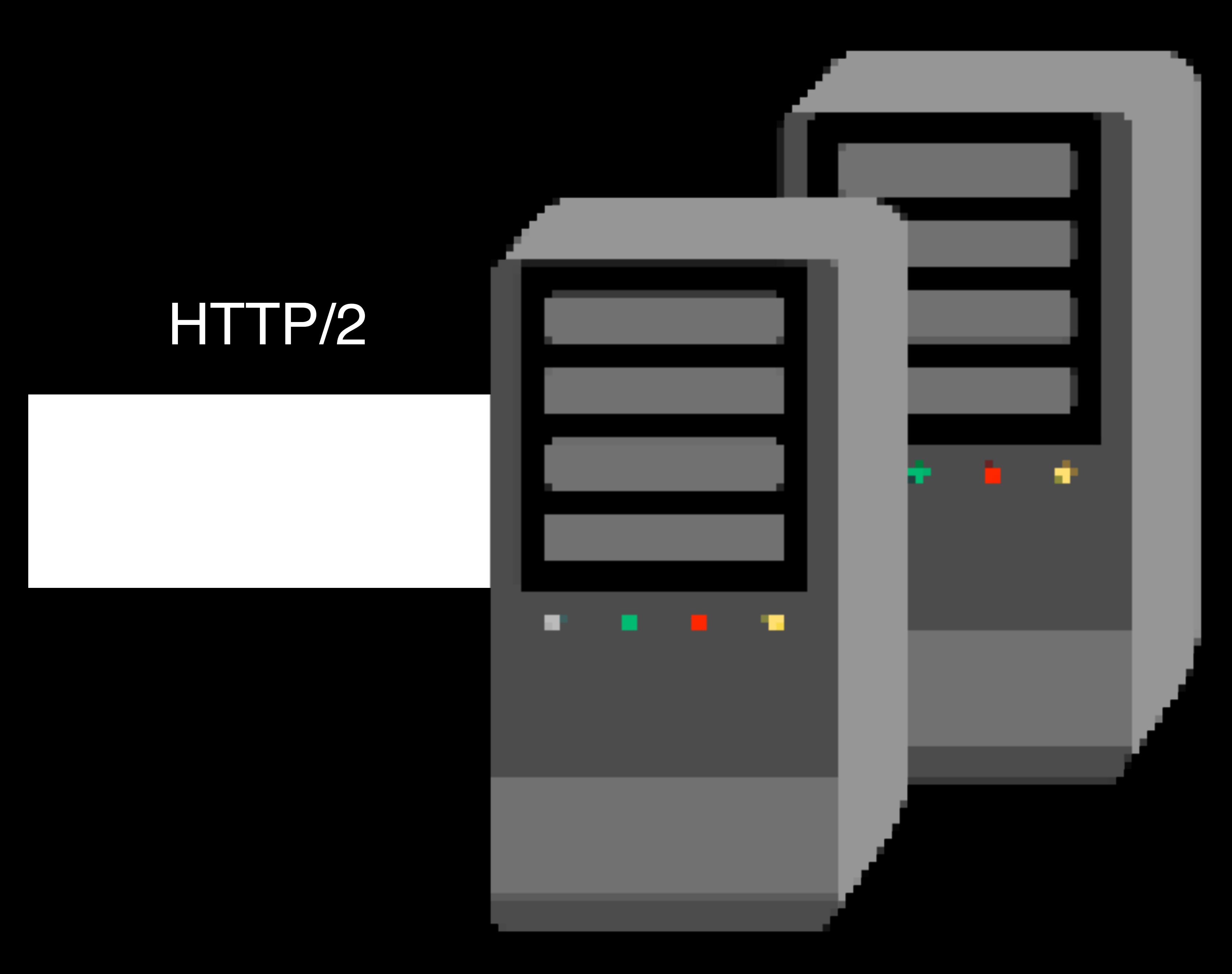
GRPC-WEB



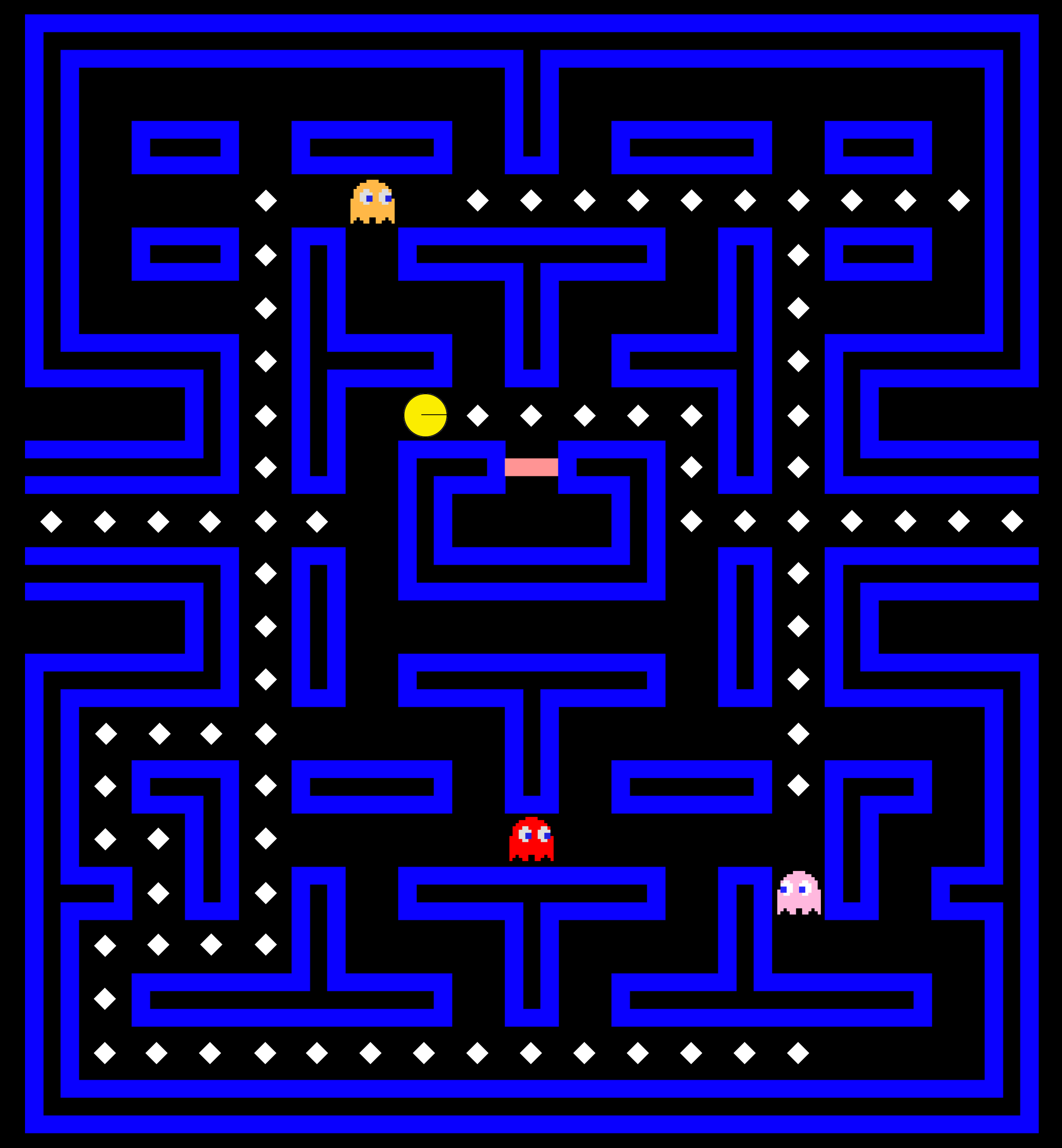


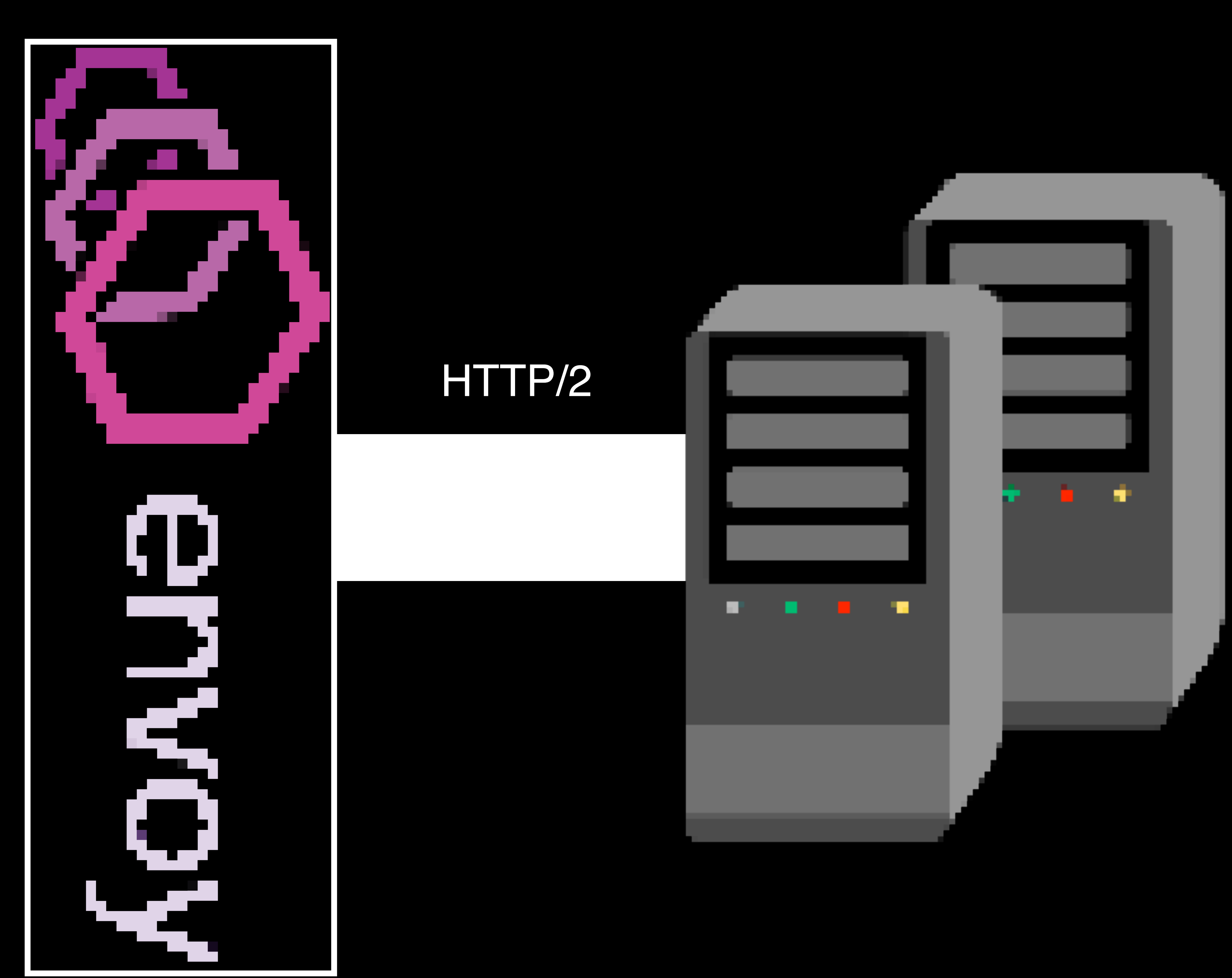
GRPC-WEB



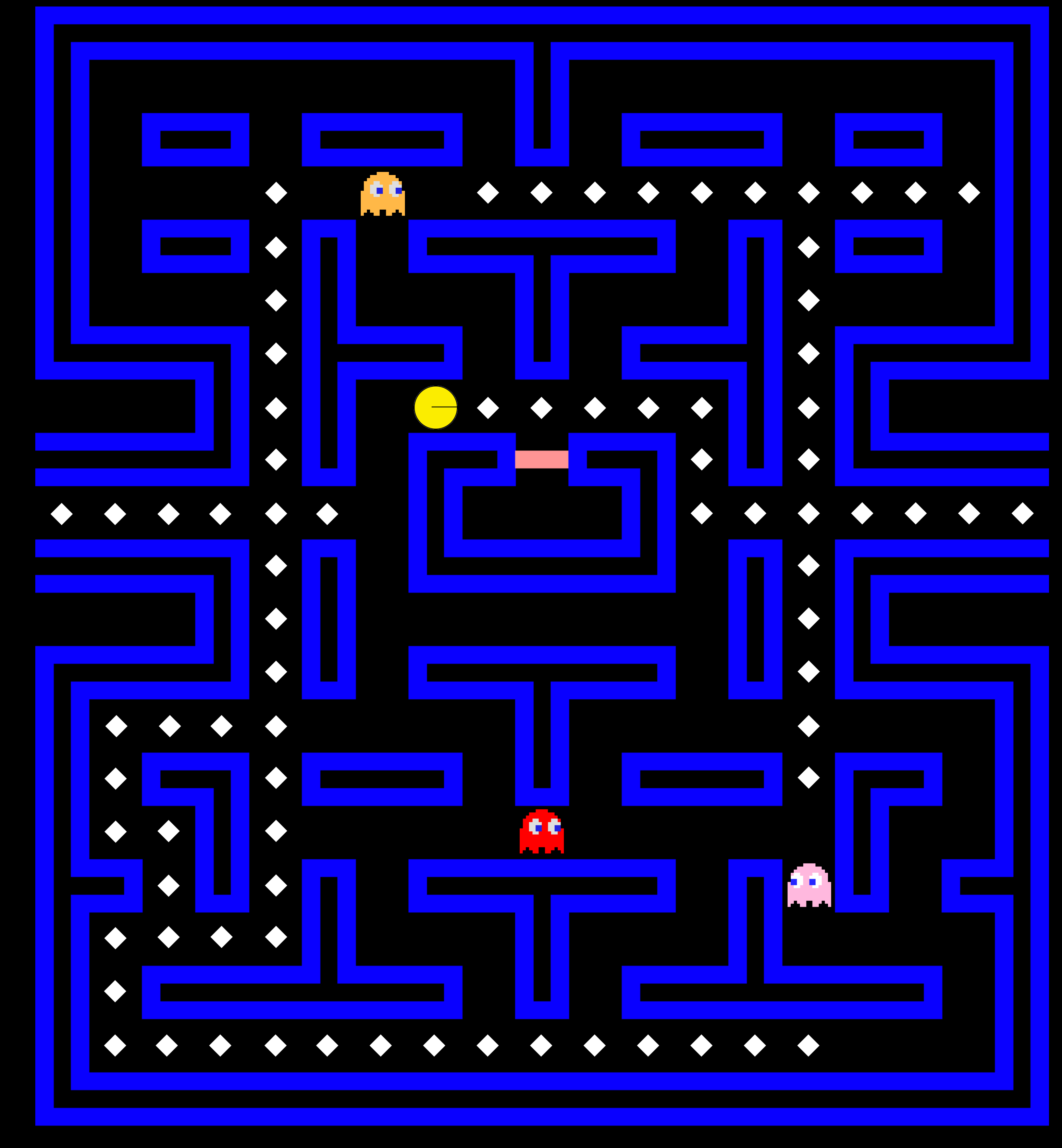


GRPC-WEB

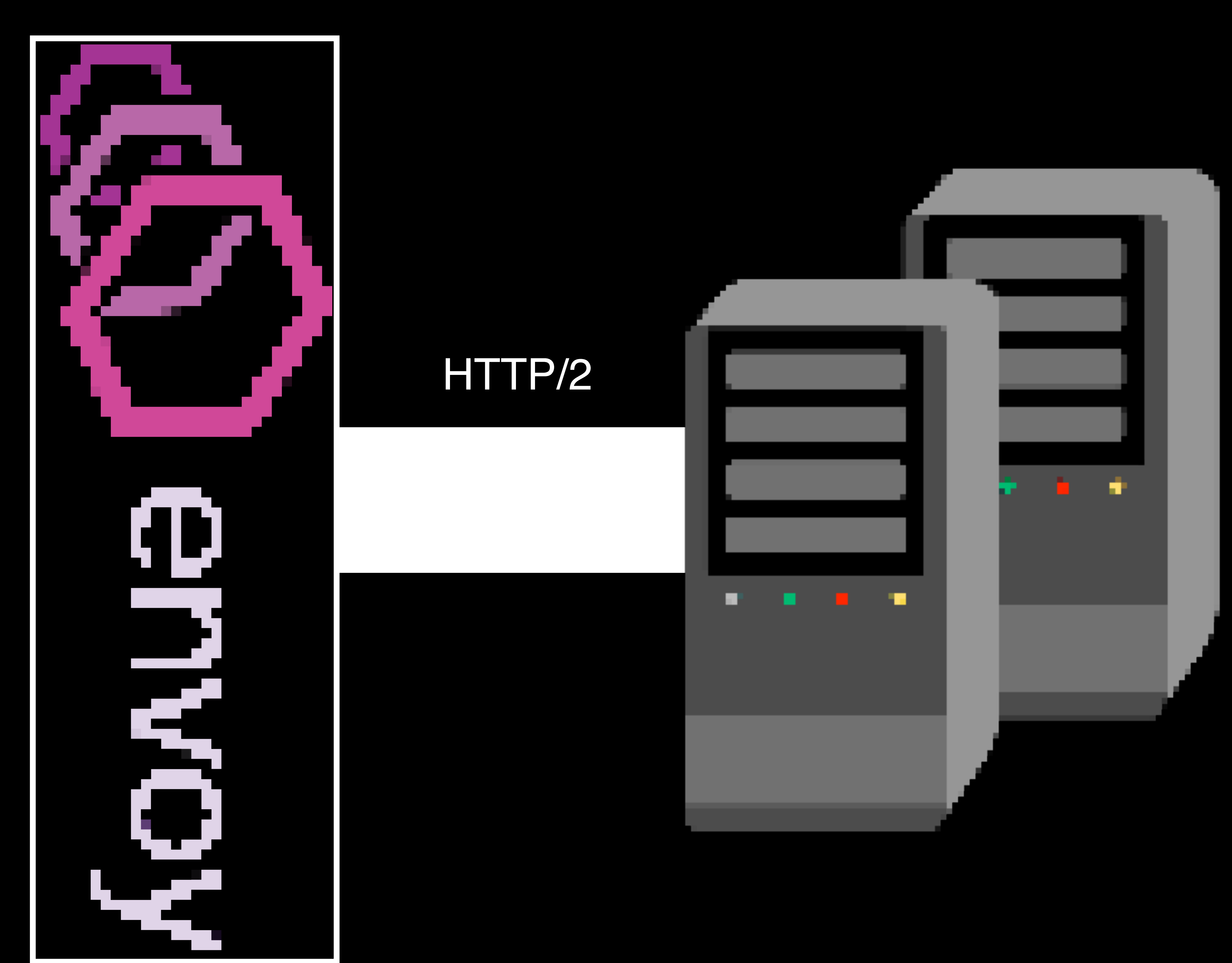




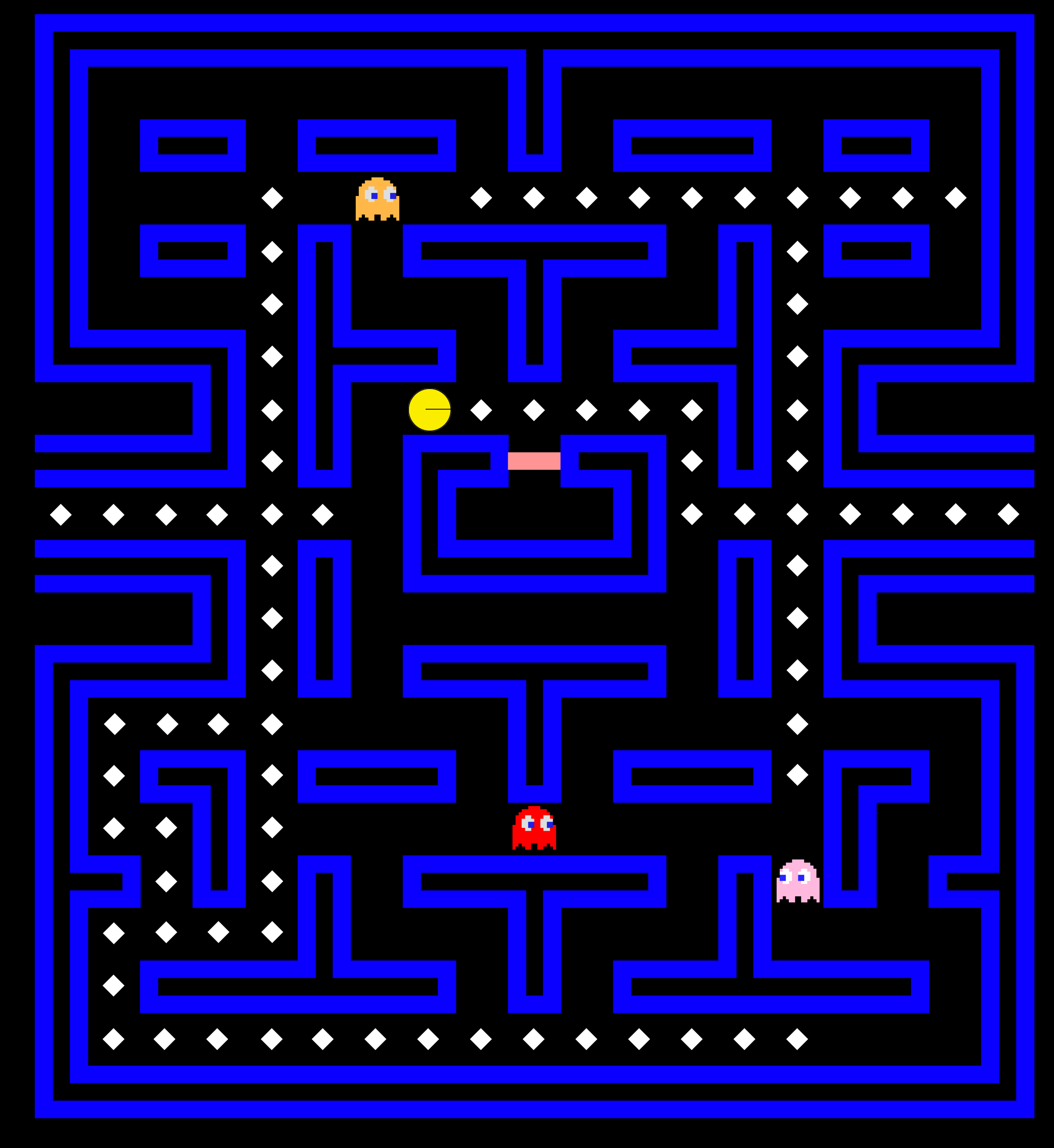
GRPC-WEB

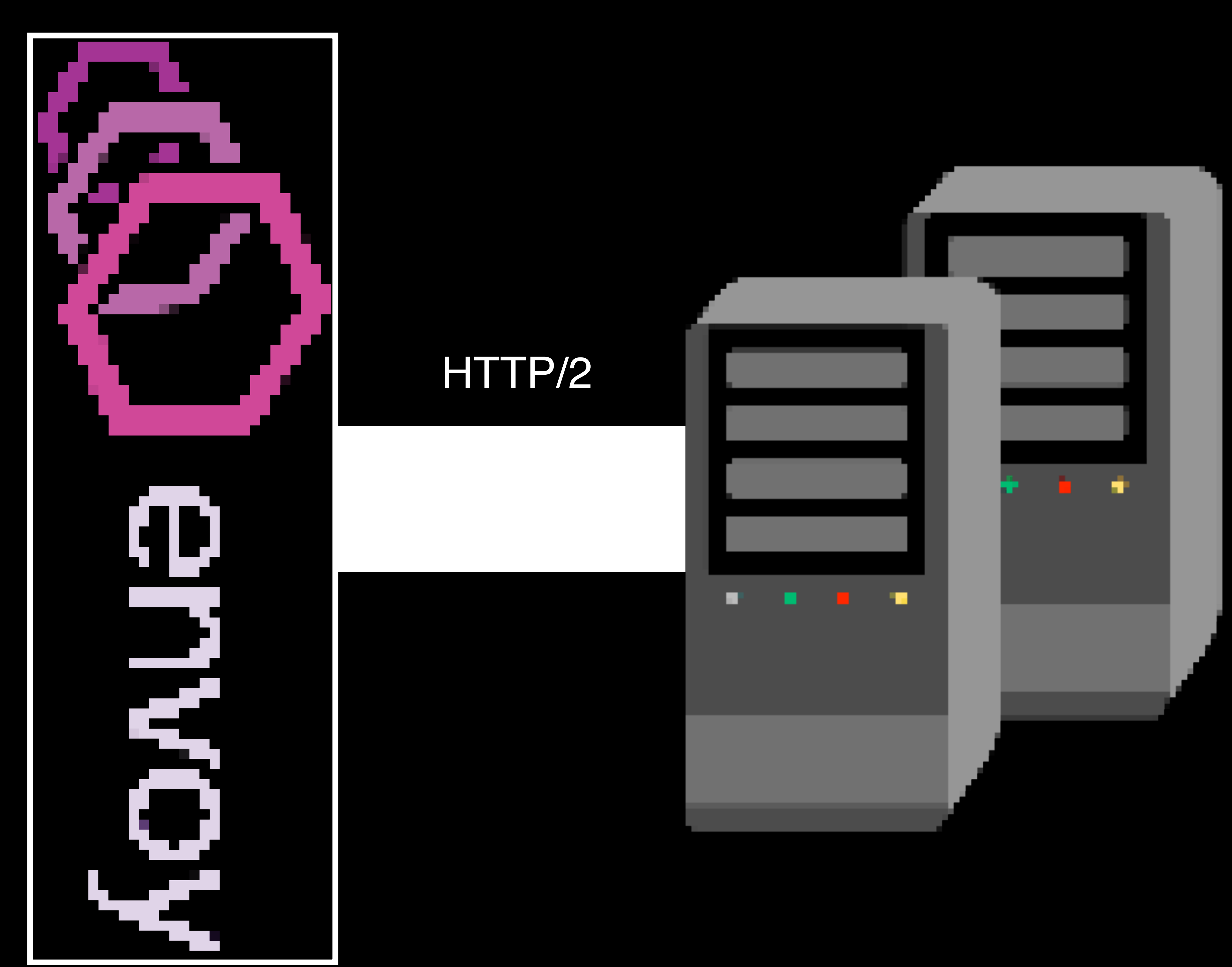


HTTP/1.x



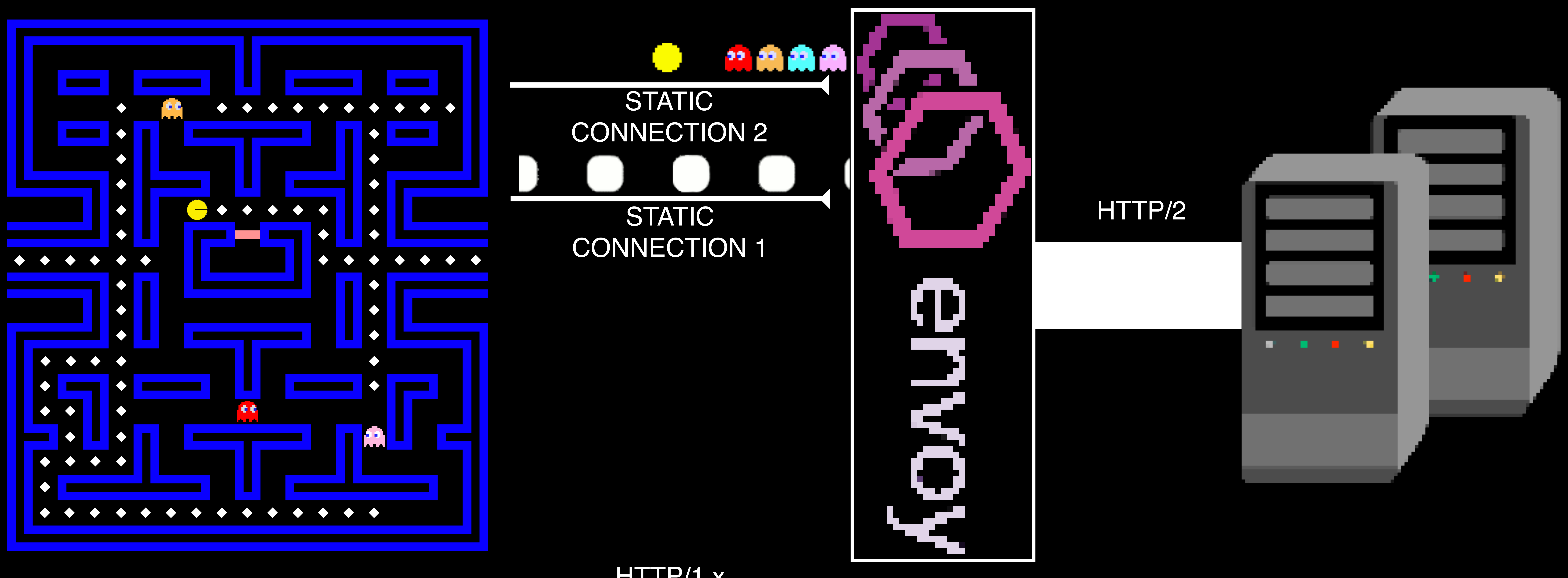
GRPC-WEB



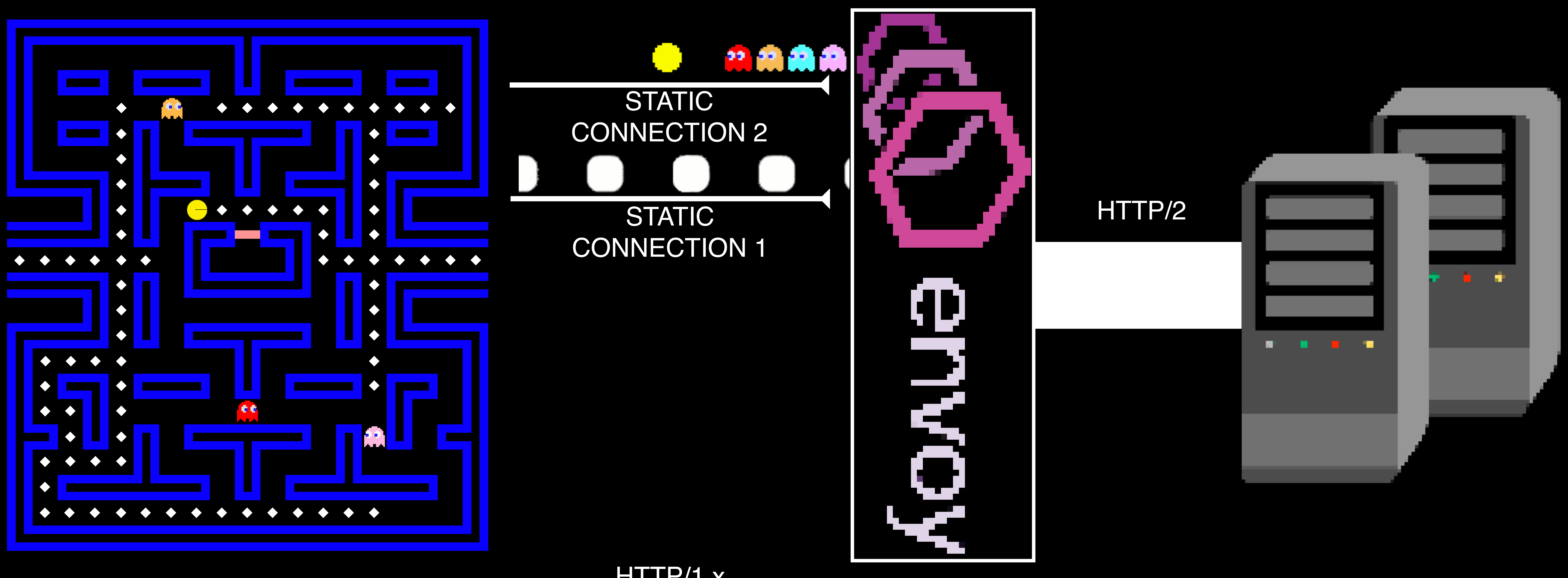


HTTP/1.x

GRPC-WEB



GRPC-WEB



“WE HAVE BACKPRESSURE CONTROL”

–gRPC

GRPC SUBSCRIBER

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

GRPC SUBSCRIBER

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

gRPC PUBLISHER

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

gRPC PUBLISHER

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

Simplify implementation of back-pressure in StreamObserver-based stub #1549

jhump opened this issue on Mar 14, 2016 · 28 comments

ejona86 2 days ago • edited

Member

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

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

> I'm also concerned about race conditions where the server thread checks isReady and then goes to sleep, but the callback comes in between the isReady call and actually going to sleep. I think that can't happen if 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.)

Simplify implementation of back-pressure in StreamObserver-based stub #1549

jhump opened this issue on Mar 14, 2016 · 28 comments

ejona86 2 days ago • edited ▾ Member

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

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

> I'm also concerned about race conditions where the server thread checks isReady and then goes to sleep, but the callback comes in between the isReady call and actually going to sleep. I think that can't happen if 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.)

Simplify implementation of back-pressure in StreamObserver-based stub #1549

jhump opened this issue on Mar 14, 2016 · 28 comments

ejona86 2 days ago • edited

Member

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

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

> I'm also concerned about race conditions where the server thread checks isReady and then goes to sleep, but the callback comes in between the isReady call and actually going to sleep. I think that can't happen if 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.)

Simplify implementation of back-pressure in StreamObserver-based stub #1549

jhump opened this issue on Mar 14, 2016 · 28 comments

ejona86 2 days ago • edited

Member

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

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

> I'm also concerned about race conditions where the server thread checks isReady and then goes to sleep, but the callback comes in between the isReady call and actually going to sleep. I think that can't happen if 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.)

Simplify implementation of back-pressure in StreamObserver-based stub #1549

jhump opened this issue on Mar 14, 2016 · 28 comments

ejona86 2 days ago • edited Member

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

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

> I'm also concerned about race conditions where the server thread checks isReady and then goes to sleep, but the callback comes in between the isReady call and actually going to sleep. I think that can't happen if 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.)

...see cases where isReady returns false, and I'm using a similar
@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.

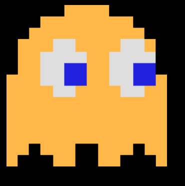
> the API does not actually specify how isReady and onReady are internally synchronized.

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.)

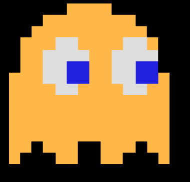
PUBLISHER

PUBLISHER

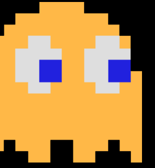


WHAT IF

PUBLISHER

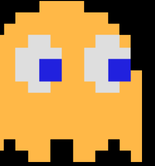


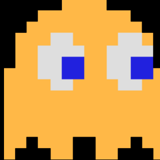
WHAT IF

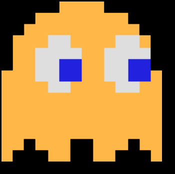
PUBLISHER 

PUBLISHER

WHAT IF

PUBLISHER 

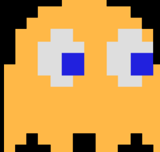
PUBLISHER 

PUBLISHER 

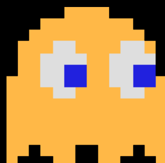
WHAT IF

PUBLISHER

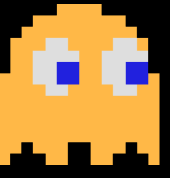
PUBLISHER



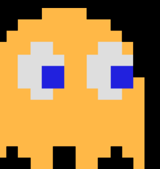
PUBLISHER

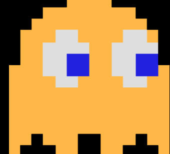


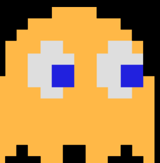
PUBLISHER

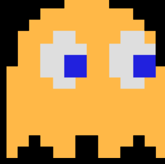


WHAT IF

PUBLISHER 

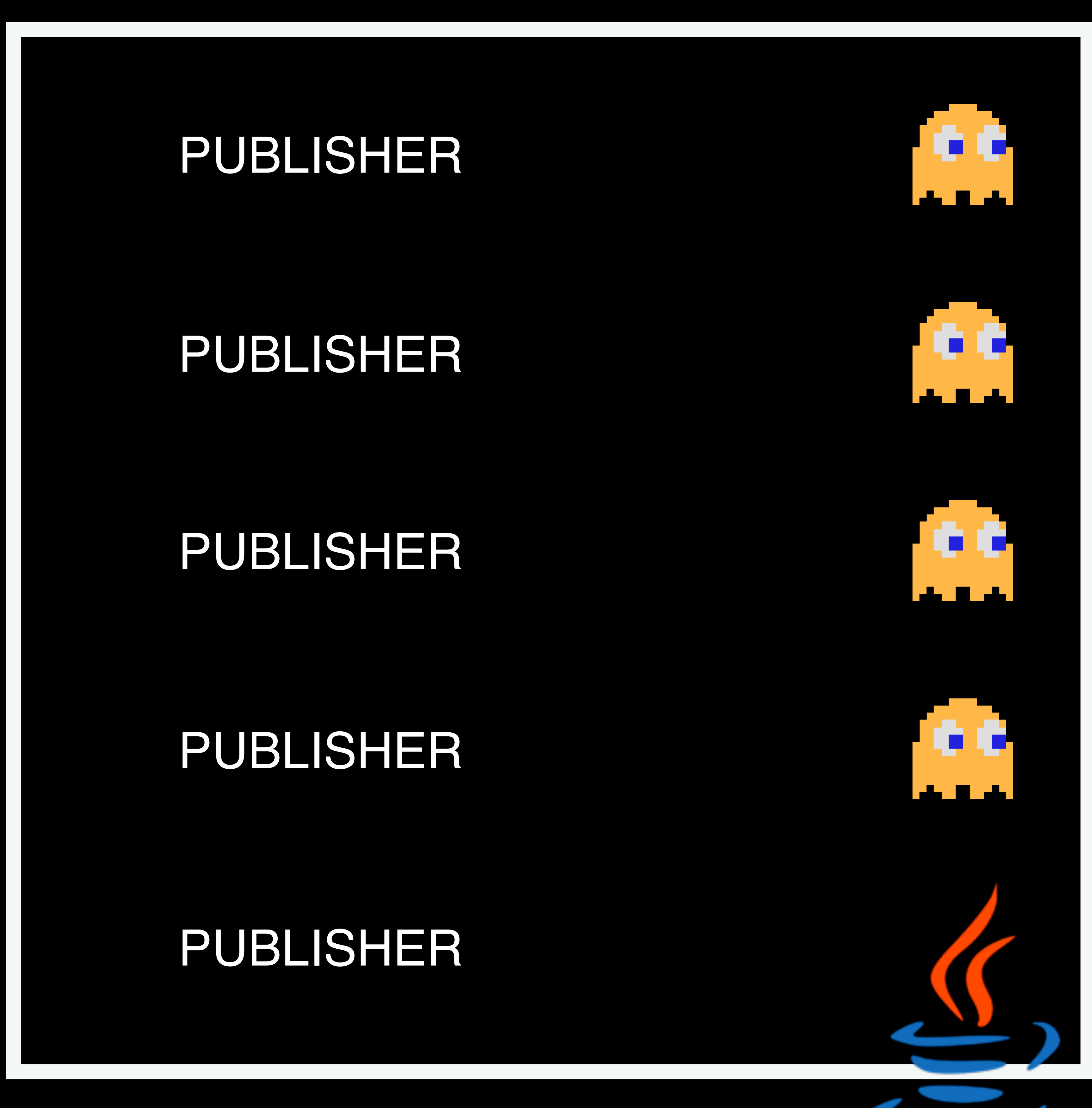
PUBLISHER 

PUBLISHER 

PUBLISHER 

PUBLISHER

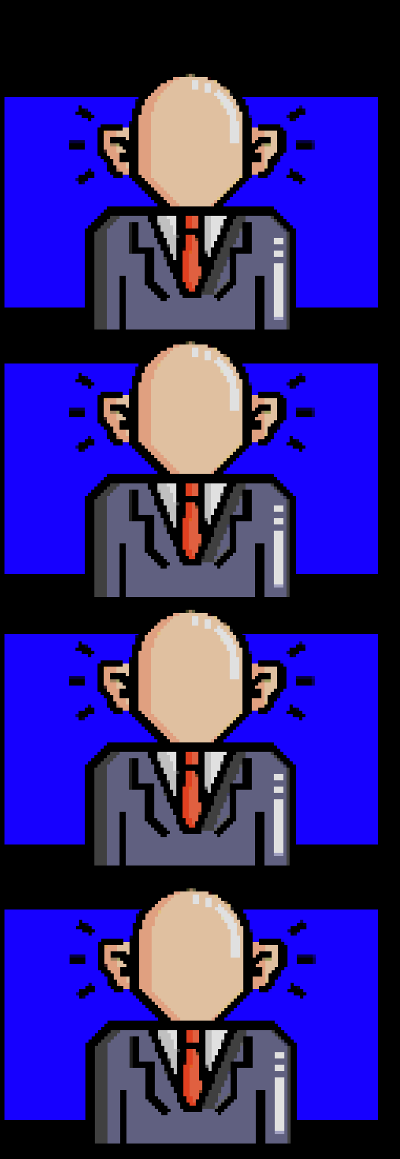
WHAT IF

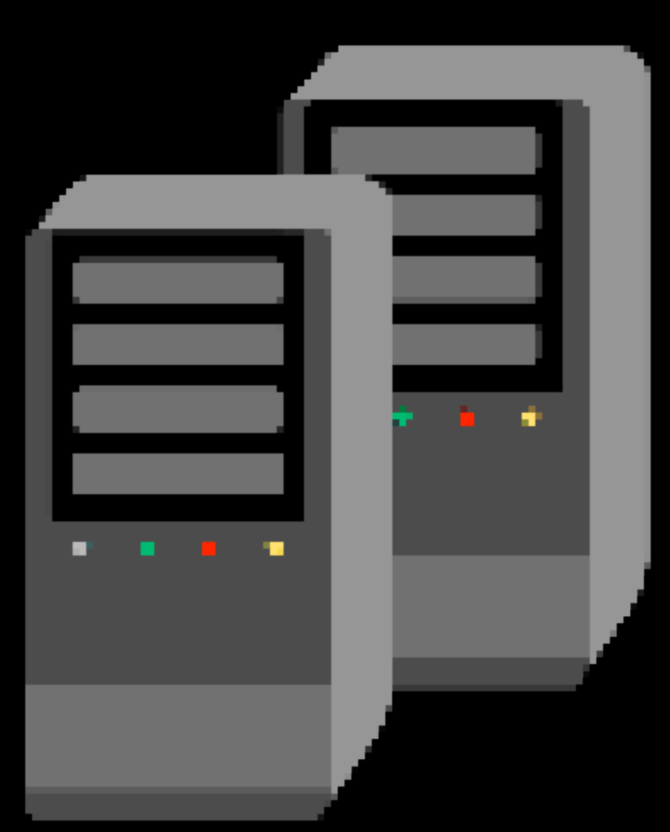


WHAT IF

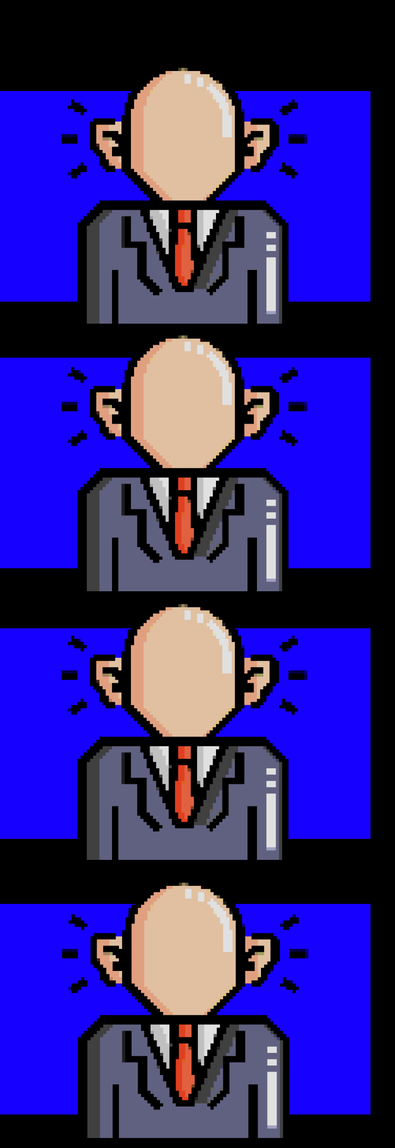
THAT MAY OVERPRODUCE

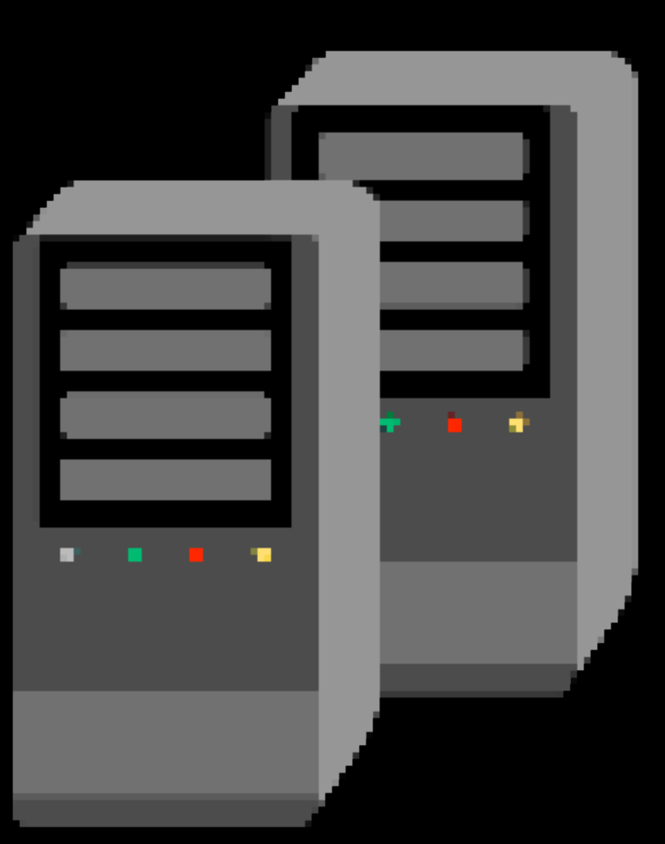
Scenario



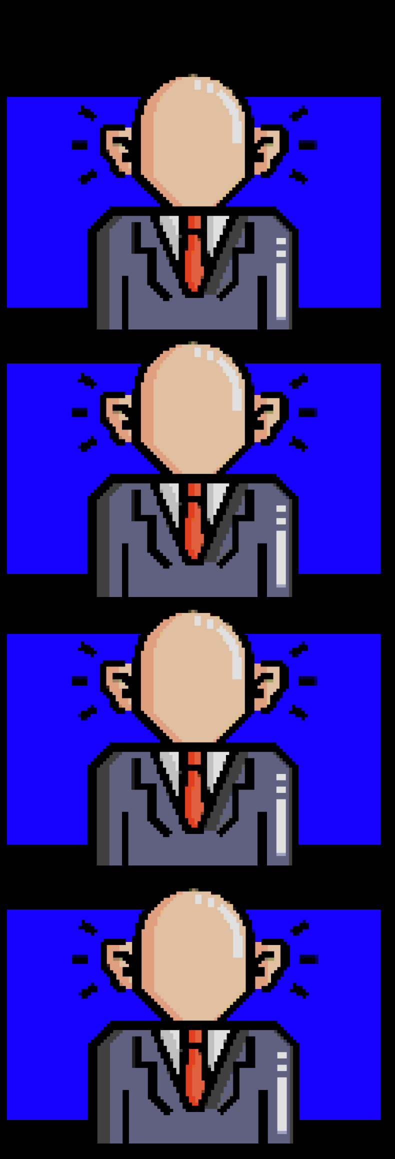


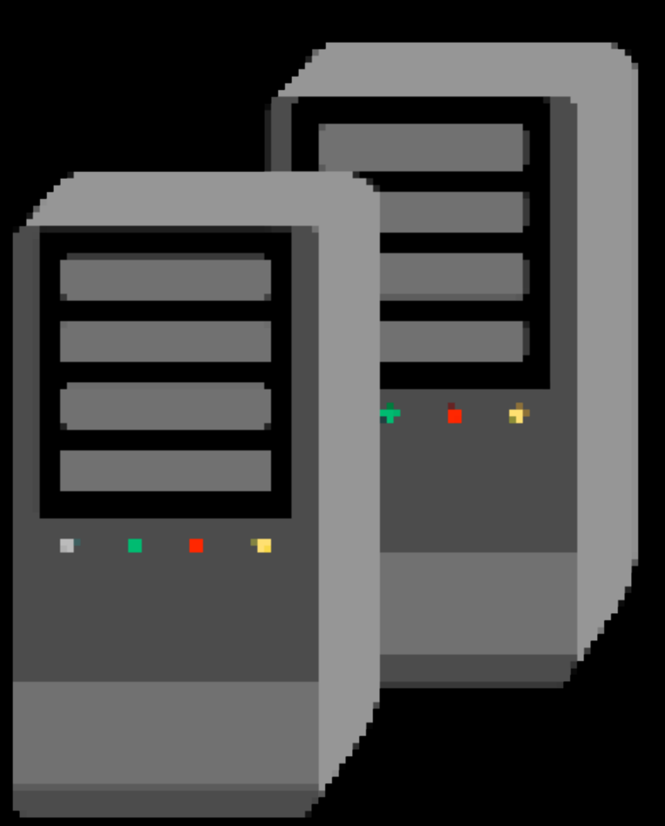
Scenario



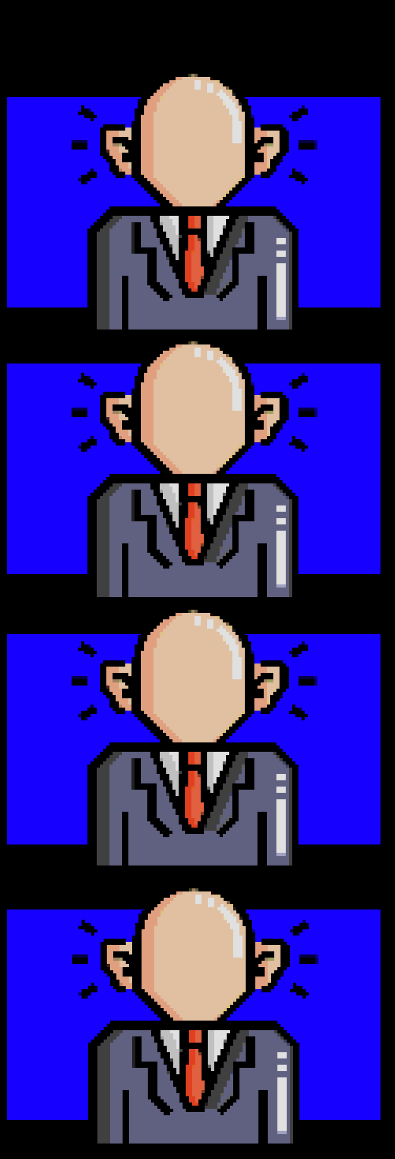


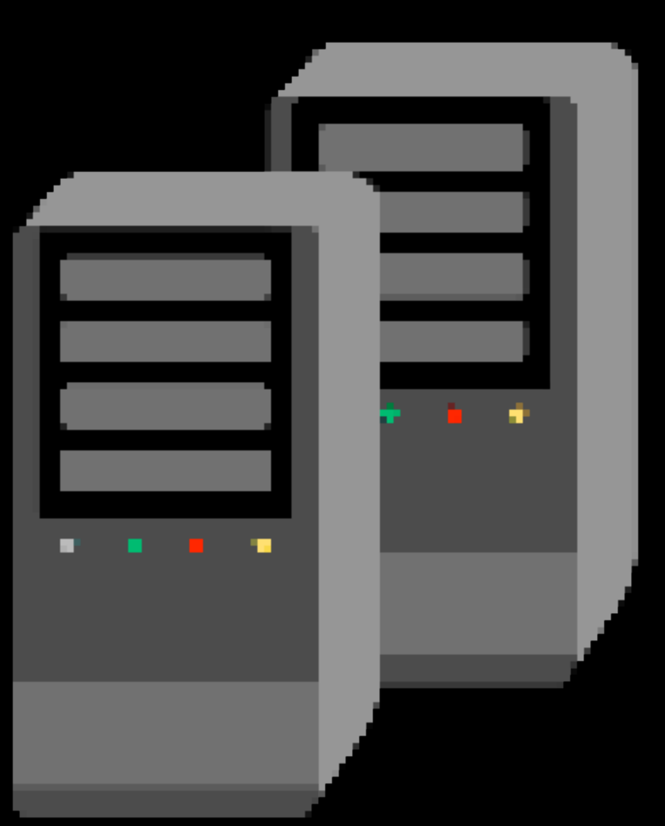
Scenario



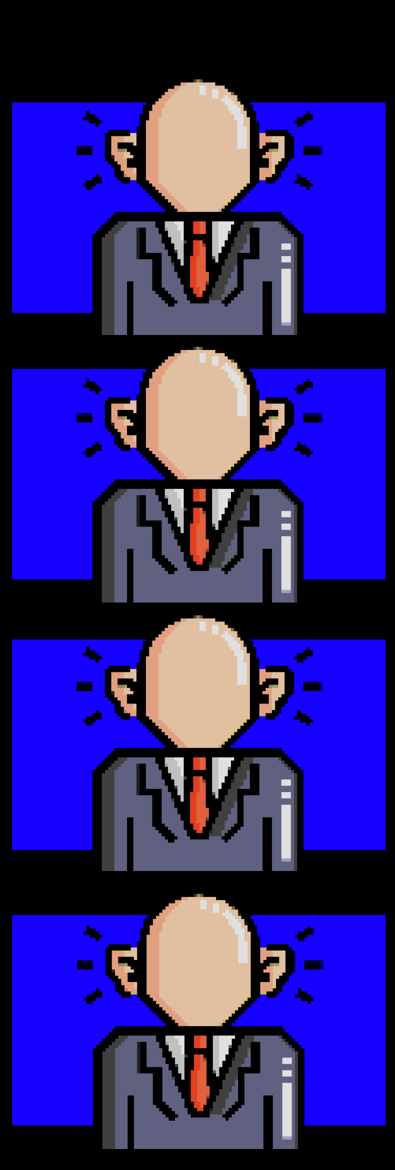


Scenario

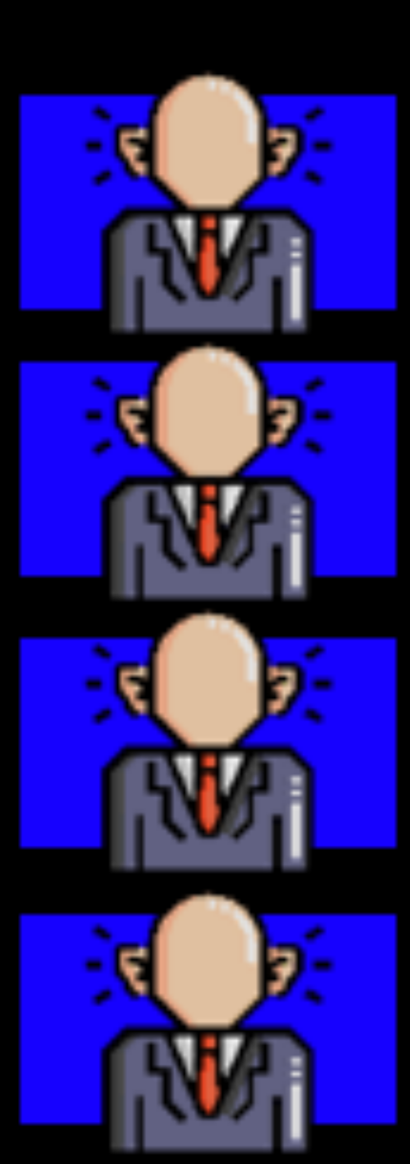




Scenario

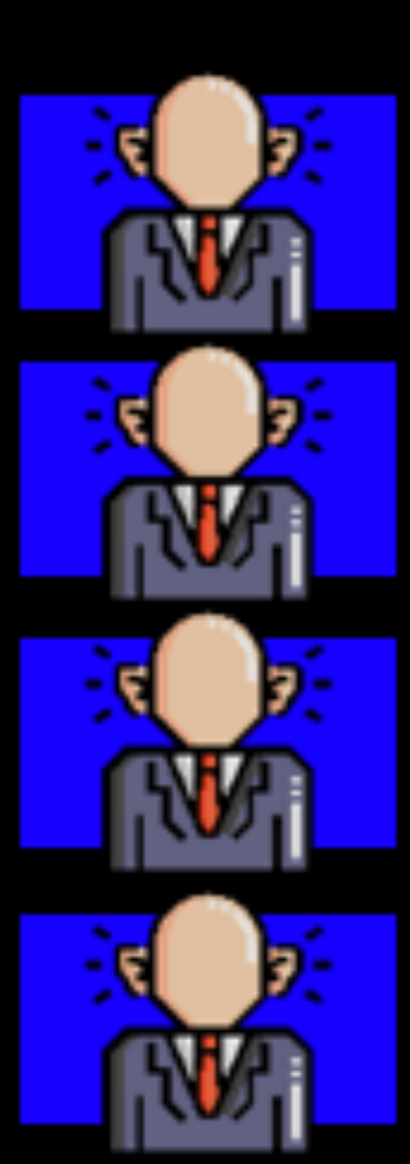


Scenario



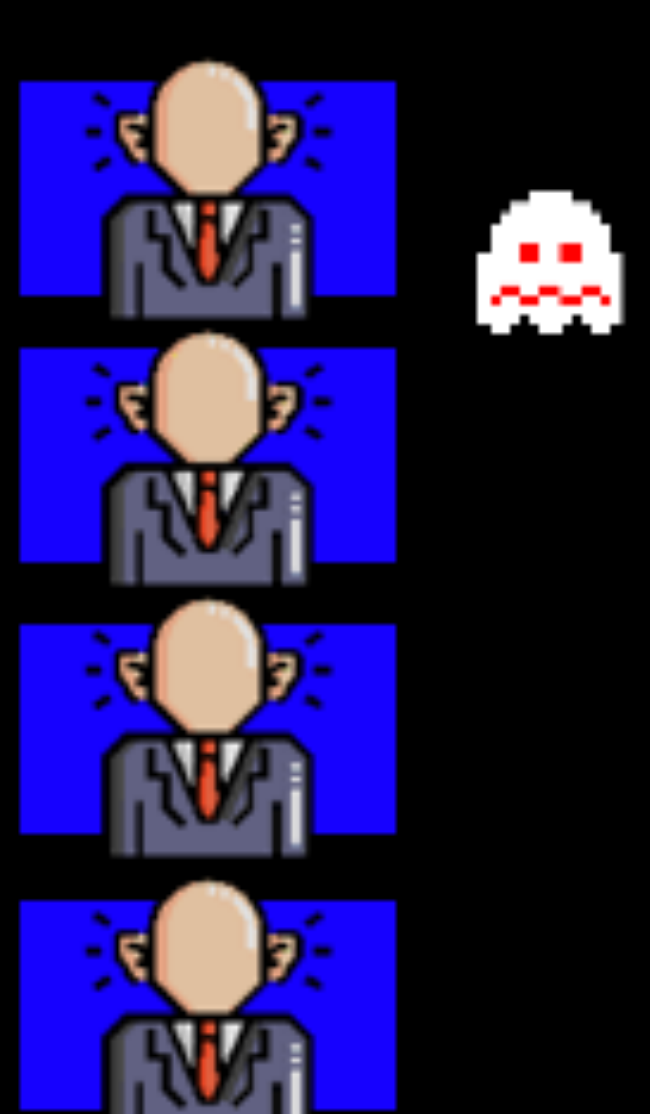
https://youtu.be/KxAjxUrGbcY

Scenario



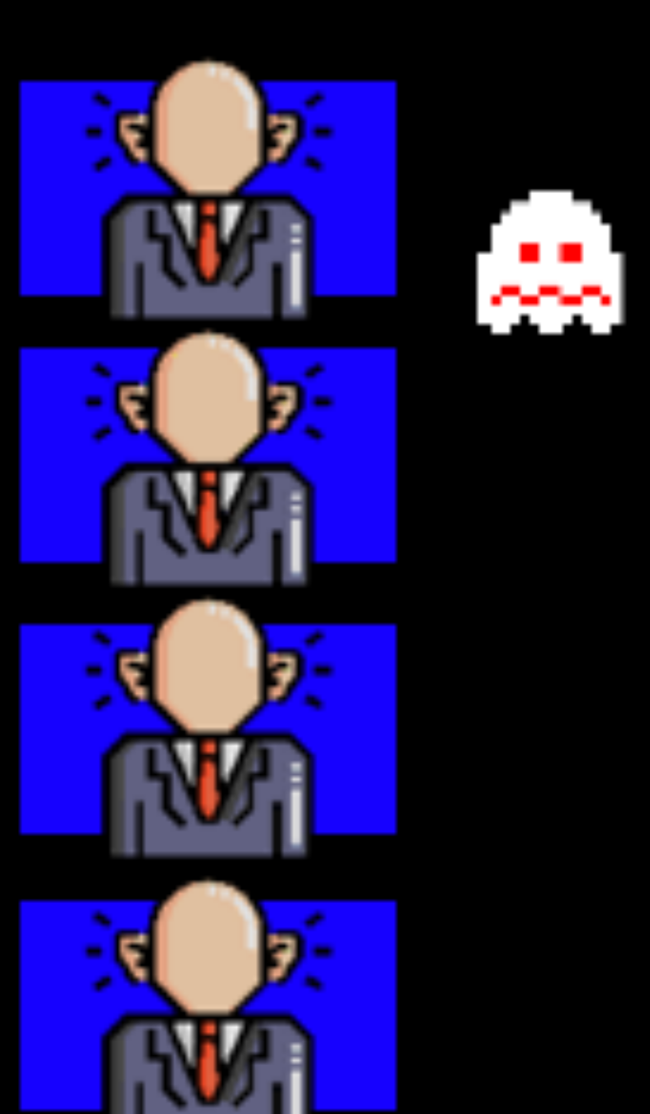
https://youtu.be/KxAjxUrGbcY

Scenario



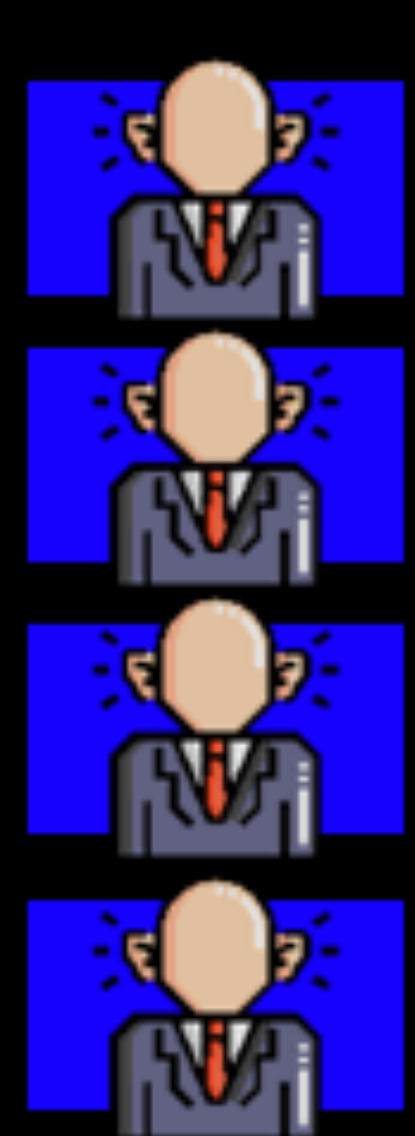
https://youtu.be/KxAjxUrGbcY

Scenario



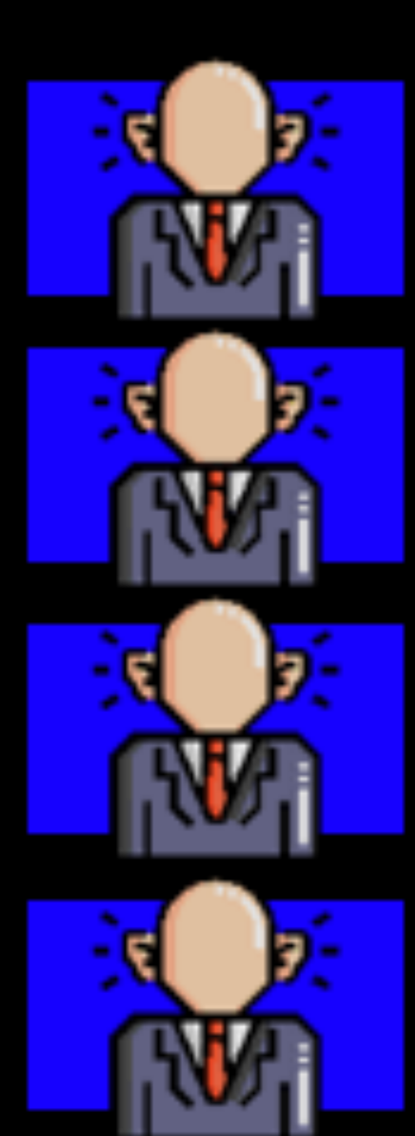
https://youtu.be/KxAjxUrGbcY

Scenario



https://youtu.be/KxAjxUrGbcY

Scenario



https://youtu.be/KxAjxUrGbcY

gRPC Publisher

game_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

gRPC Subscribet

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

gRPC Subscribe
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

gRPC Subscriber
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

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
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

CloudNative JVM Setup

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

```
-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

Summary

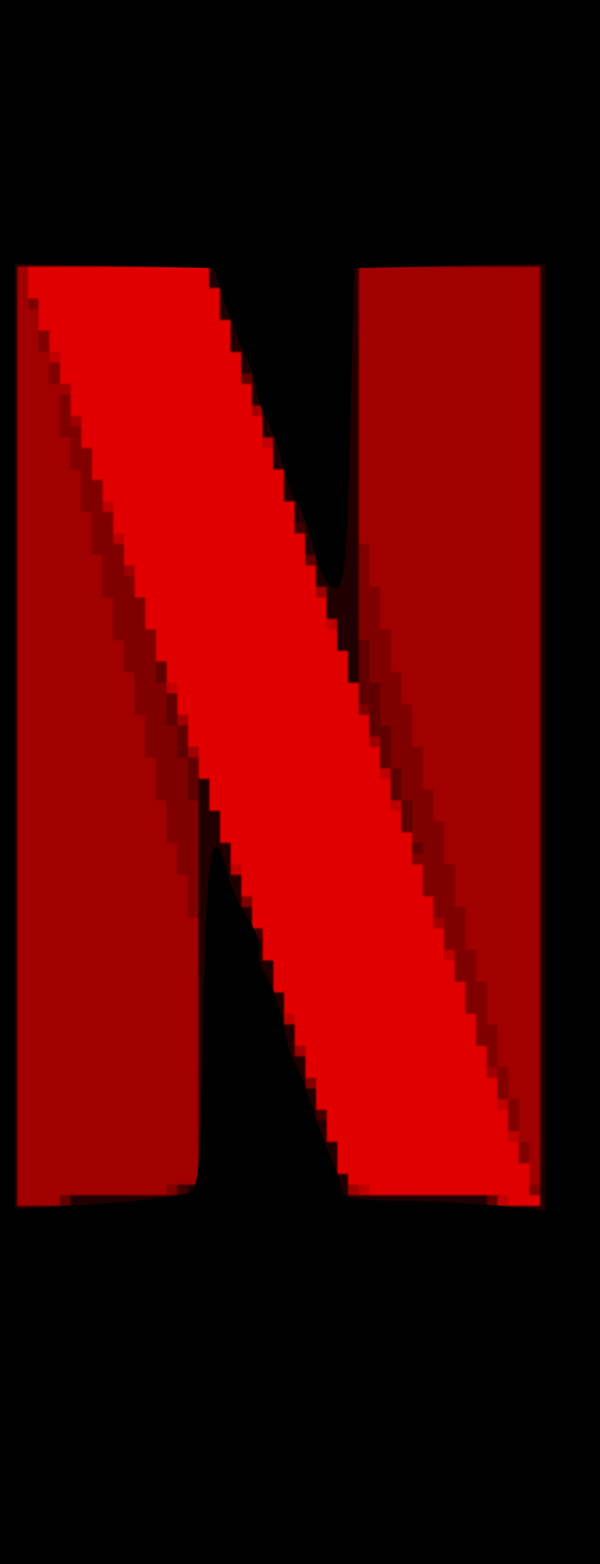
- Everything is either SLOW, HARD to implement or LACKS browser support

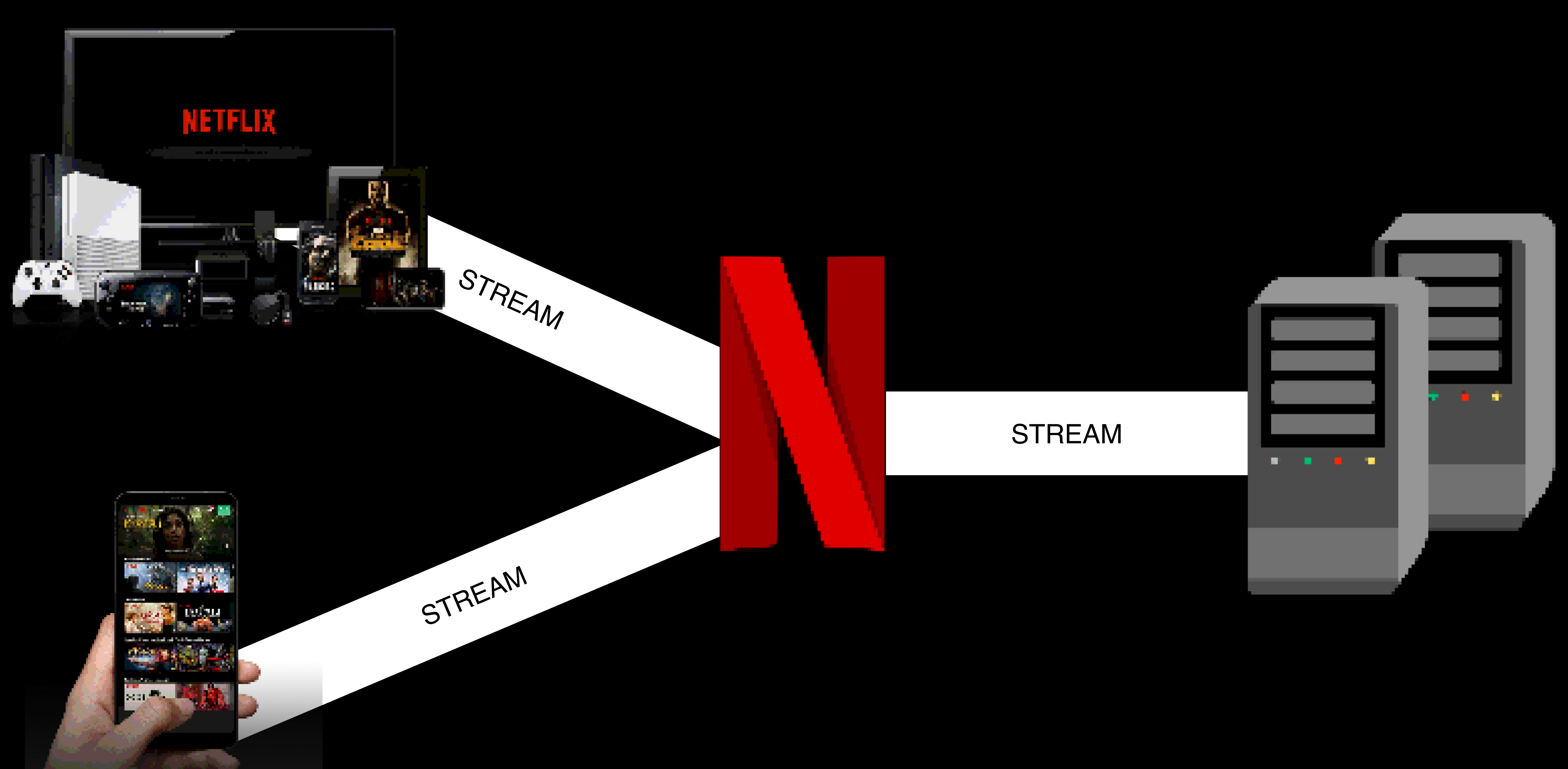
Summary

- Everything is either SLOW, HARD to implement or LACKS browser support
- Flow control is far from needed

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???



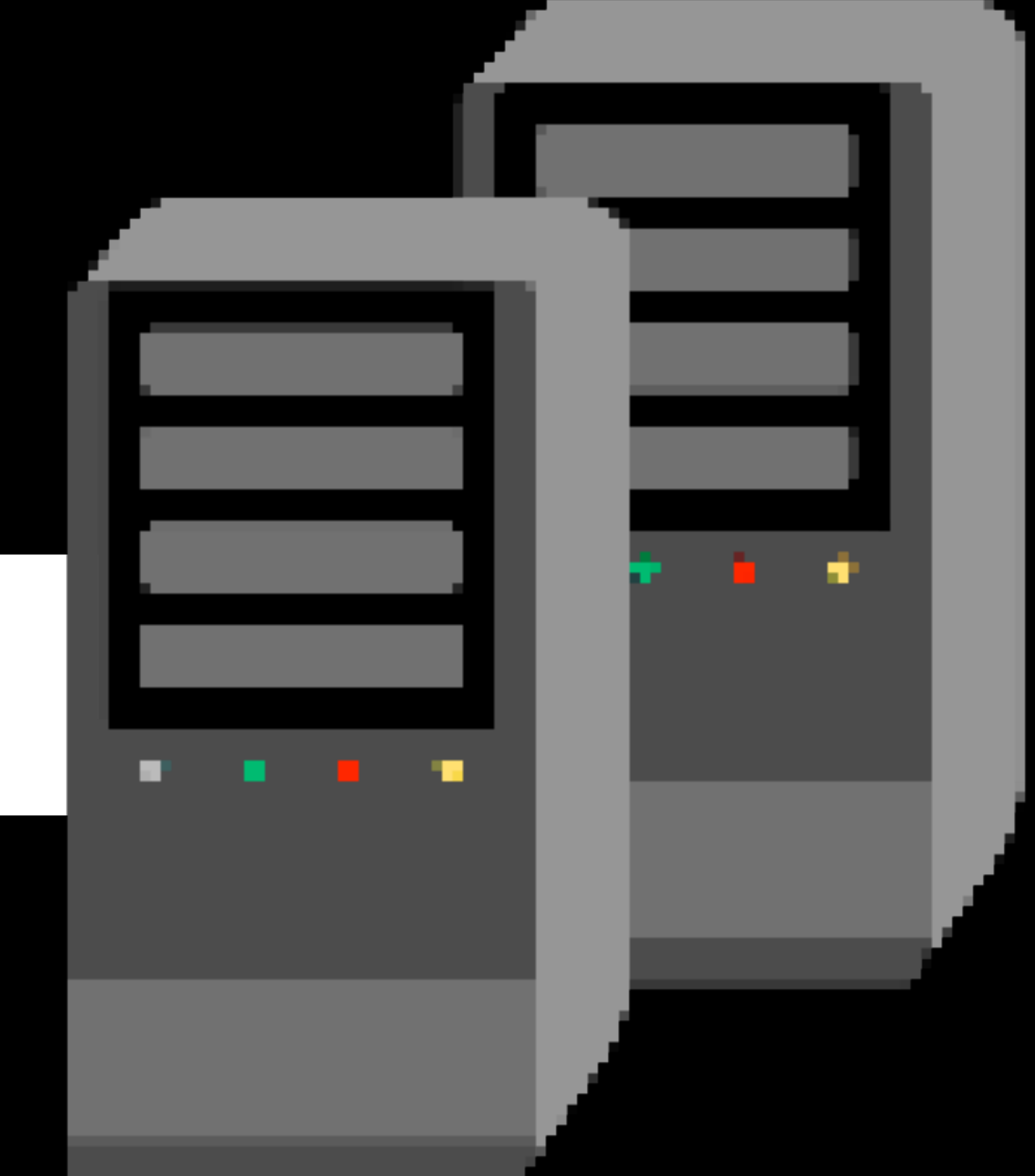

NETFLIX
STREAM
STREAM
STREAM

NETFLIX

STREAM

STREAM

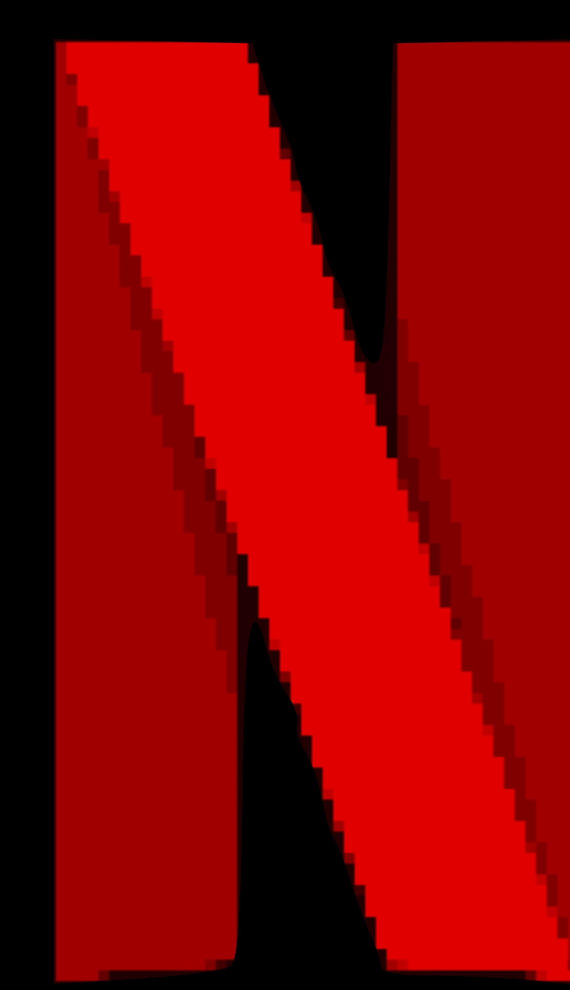
STREAM



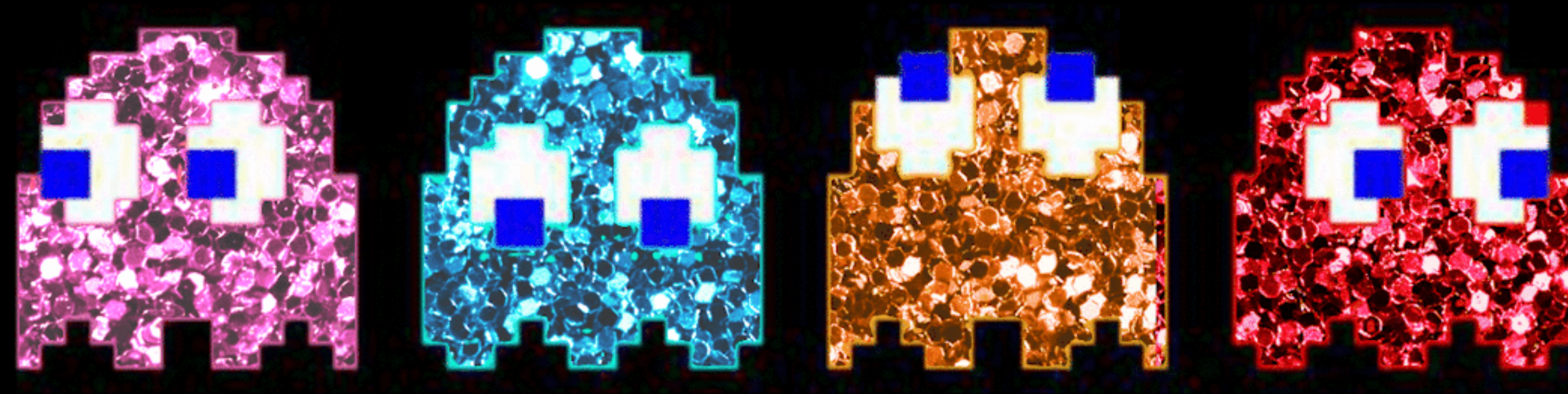
Netflix case study on gRPC

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

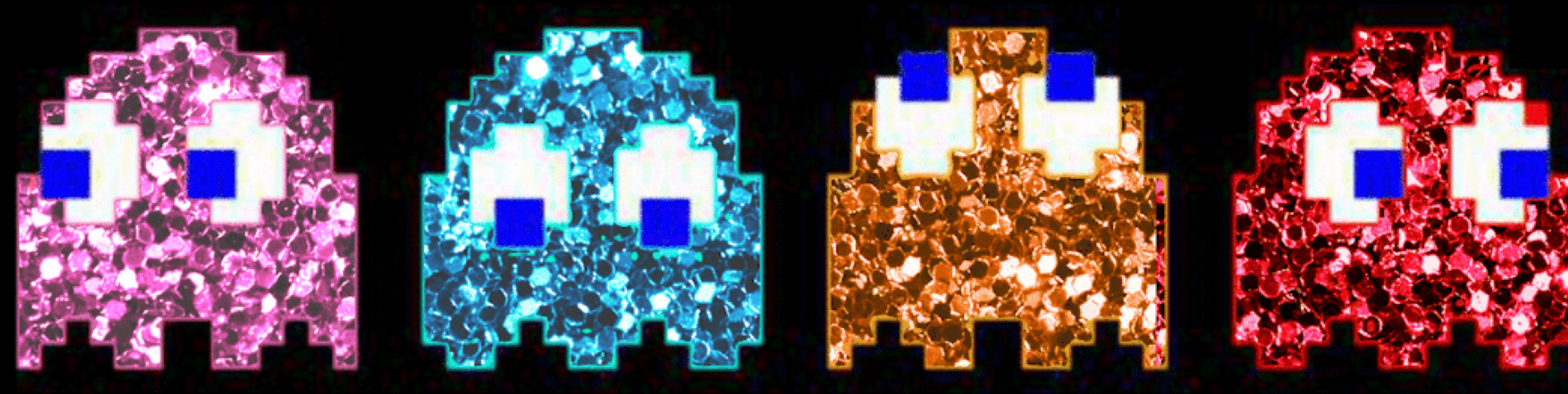
  https://bit.ly/2FUvHG3



RSOCKET WAY



RSOCKET WAY



What is RSocket?

What is RSocket?

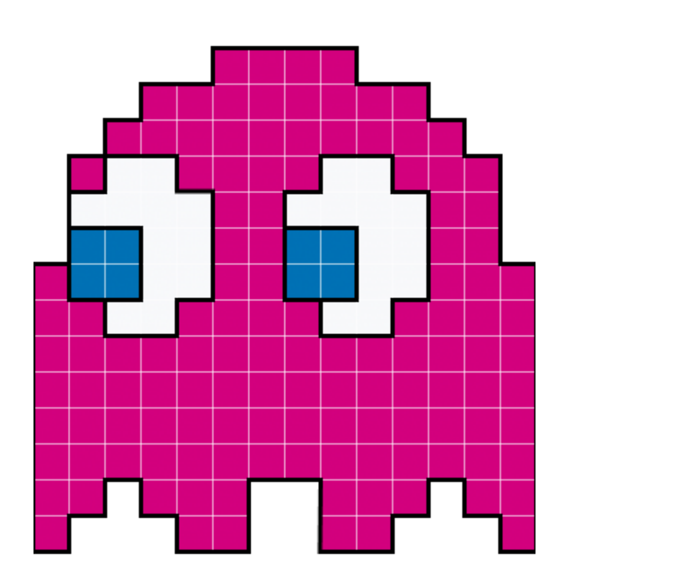
REACTIVE-STREAMS as NETWORK PROTOCOL

is.gd/rsocket

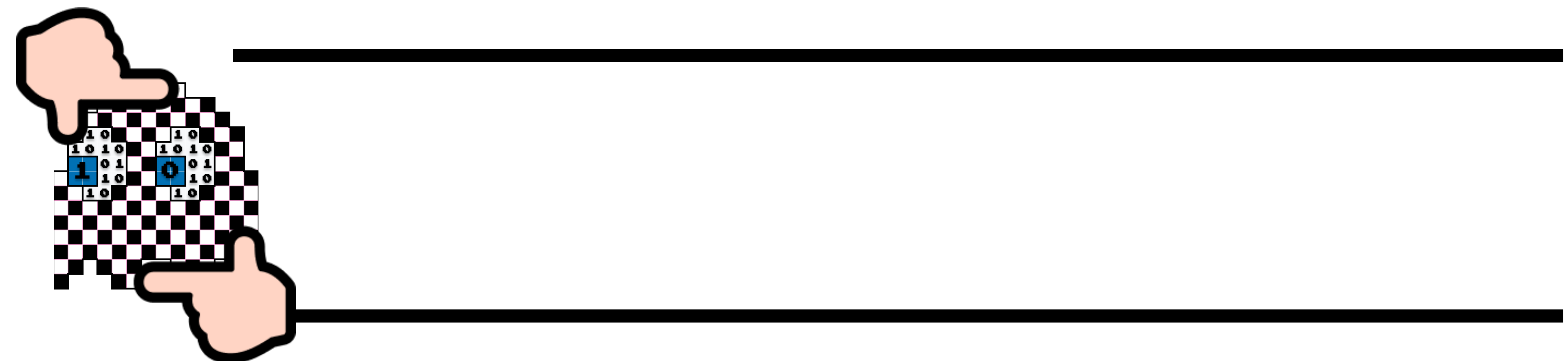
DEMO

Binary

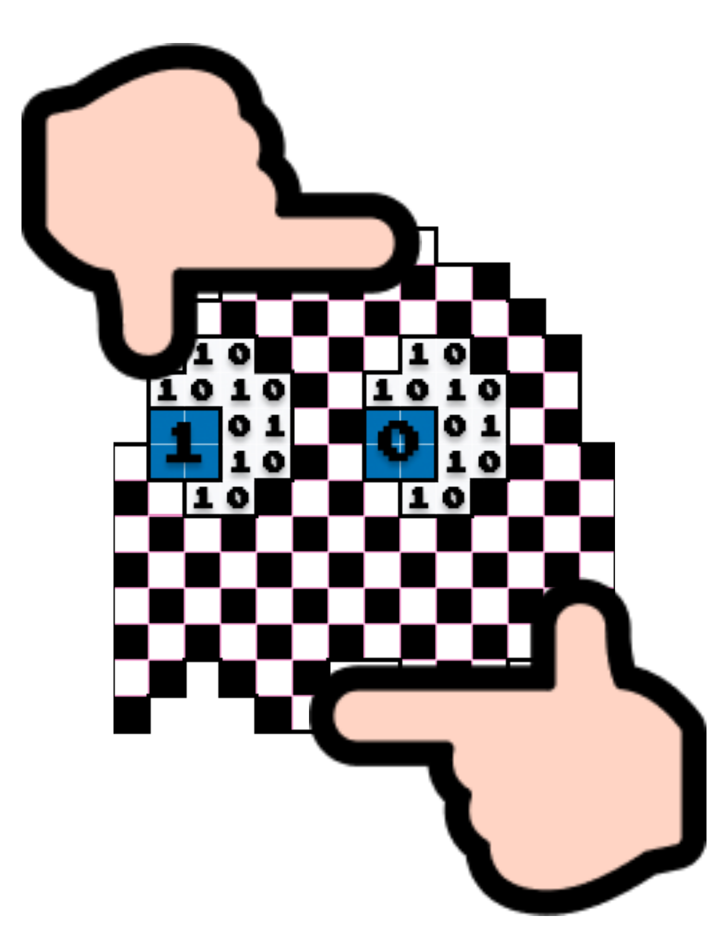
Binary



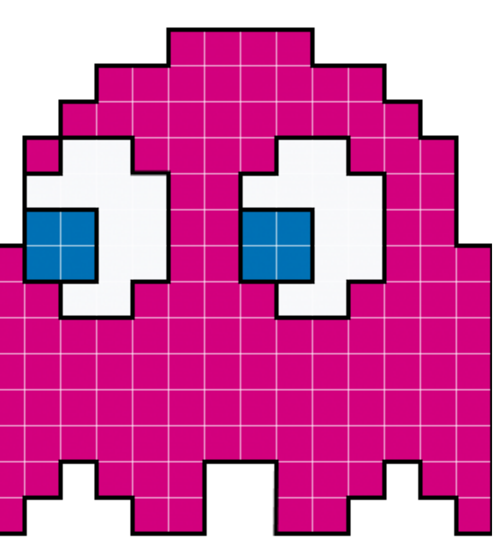
Binary



Binary

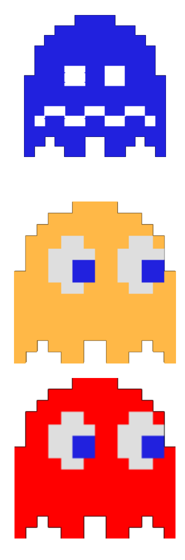


Binary

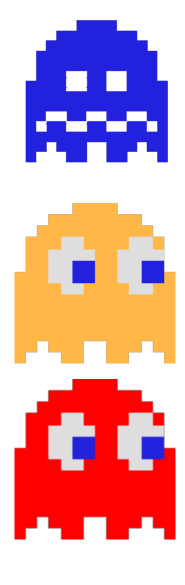


Multiplexed

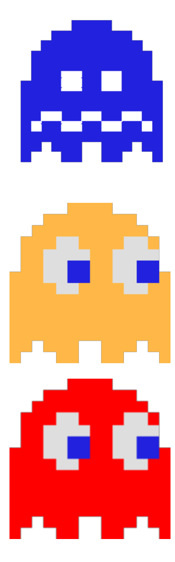
Multiplexed



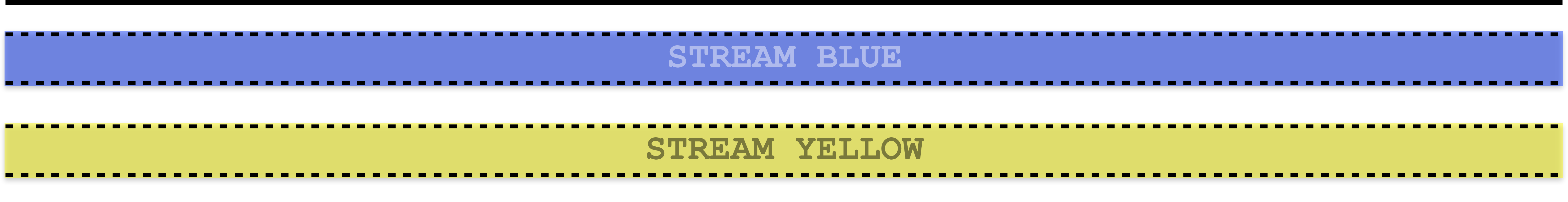
Multiplexed

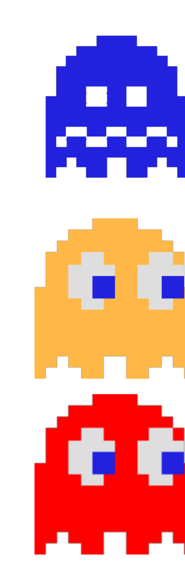


Multiplexed

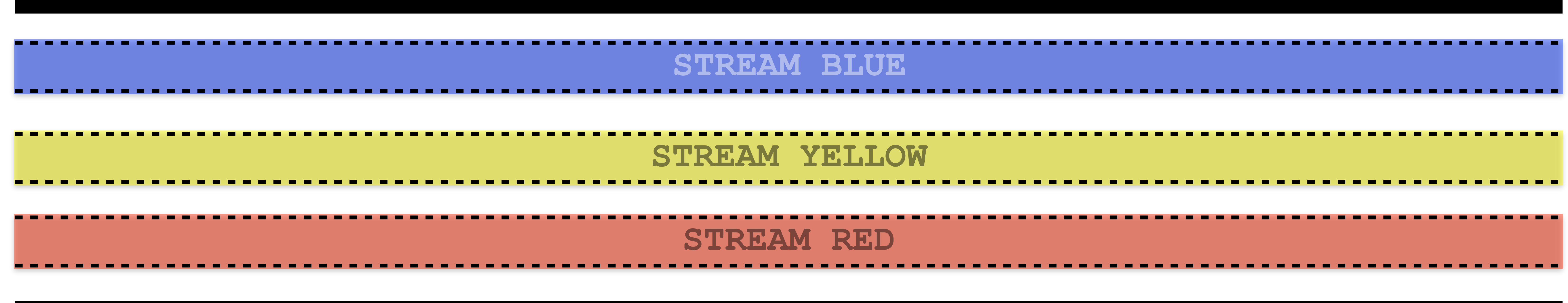


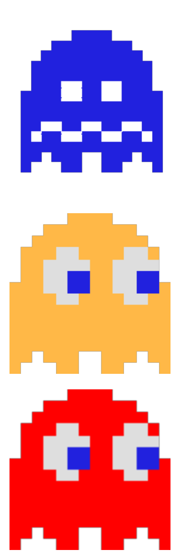
Multiplexed



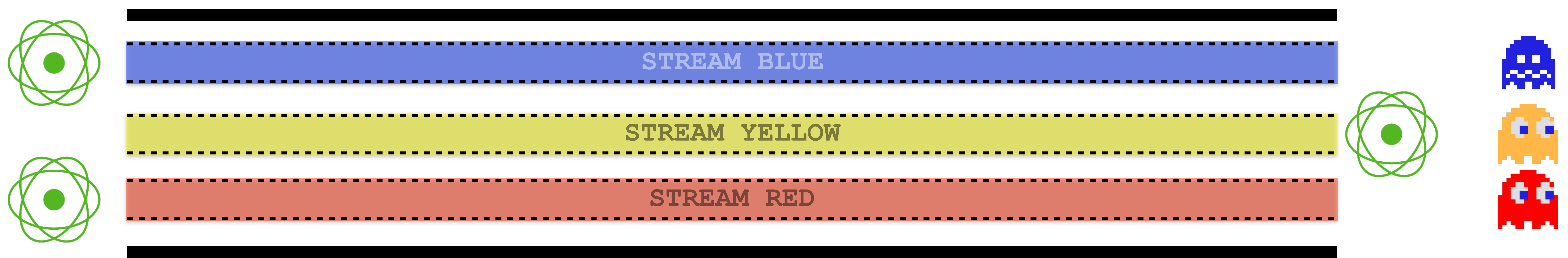


Multiplexed





Multiplexed



Project Reactor

Transport Agnostic

Transport Agnostic

WebSocket

Transport Agnostic

TCP

Transport Agnostic

QUIC

Transport Agnostic

Aeron

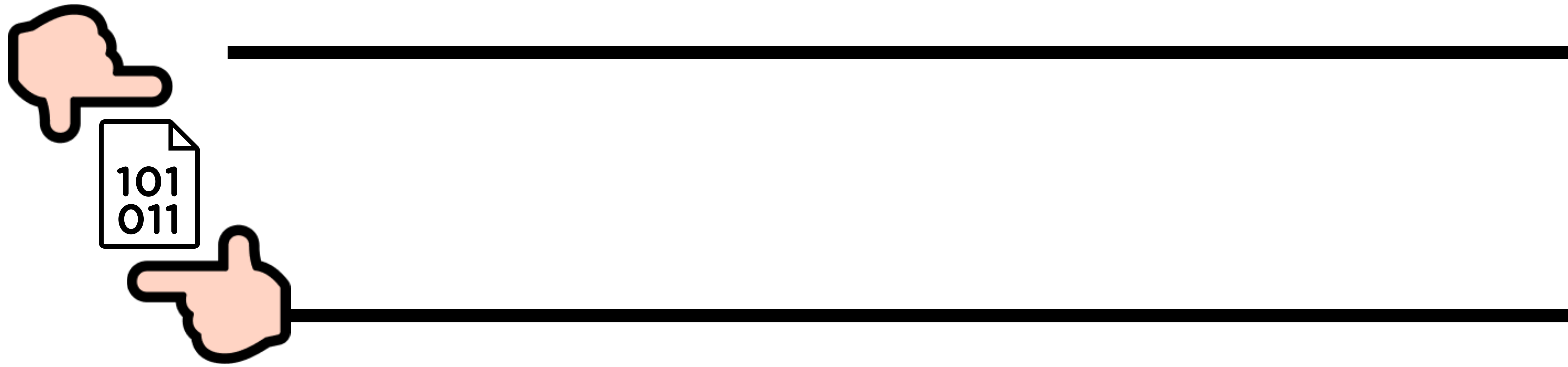
Reactive-Streams

Backpressure

Reactive-Streams

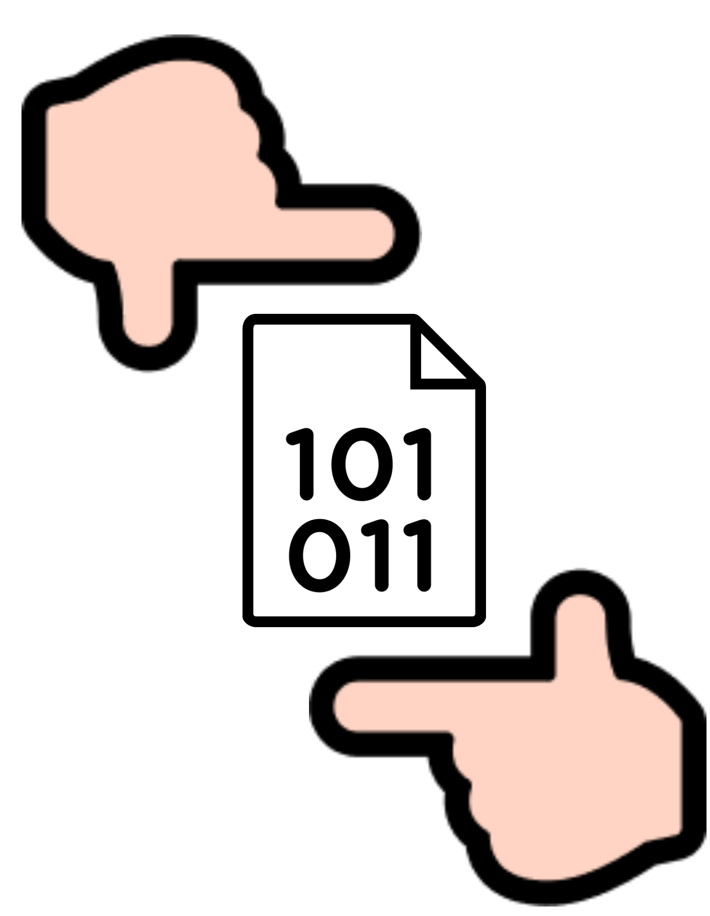
Backpressure

Reactive-Streams
Backpressure



Reactive-Streams

Backpressure

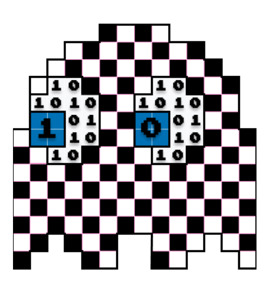


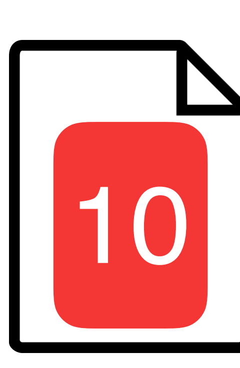
Reactive-Streams

Backpressure

Reactive-Streams

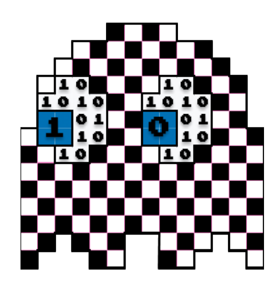
Backpressure





Reactive-Streams

Backpressure



Peer-to-peer

Client can implement request handler

CLIENT

SERVER

Peer-to-peer

Client can implement request handler

CLIENT

SERVER

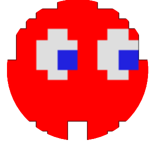
Peer-to-peer

Client can implement request handler

CLIENT

SERVER

STREAM RED



Peer-to-peer

Client can implement request handler

CLIENT

SERVER

STREAM BLUE

STREAM RED

Notable Features

Notable Features

- LEASING - GIVE CAPACITY TO CLIENTS, AVOID CIRCUIT BREAKERS (CONCEPT IS BUILT-IN IN THE PROTOCOL)

Notable Features

- LEASING - GIVE CAPACITY TO CLIENTS, AVOID CIRCUIT BREAKERS (CONCEPT IS BUILT-IN IN THE PROTOCOL)
- RESUMABILITY - RESUME STREAMS IF CONNECTION HAS LOST (MOBILE CONNECTIVITY CASE)

Notable Features

- LEASING - GIVE CAPACITY TO CLIENTS, AVOID CIRCUIT BREAKERS (CONCEPT IS BUILT-IN IN THE PROTOCOL)
- RESUMABILITY - RESUME STREAMS IF CONNECTION HAS LOST (MOBILE CONNECTIVITY CASE)
- FRAGMENTATION - SPLIT LARGE PAYLOADS INTO SMALLER CHUNKS

SOME CODE

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

```
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

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

SPRING-MESSAGING

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

SPRING-MESSAGING

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

SPRING-MESSAGING

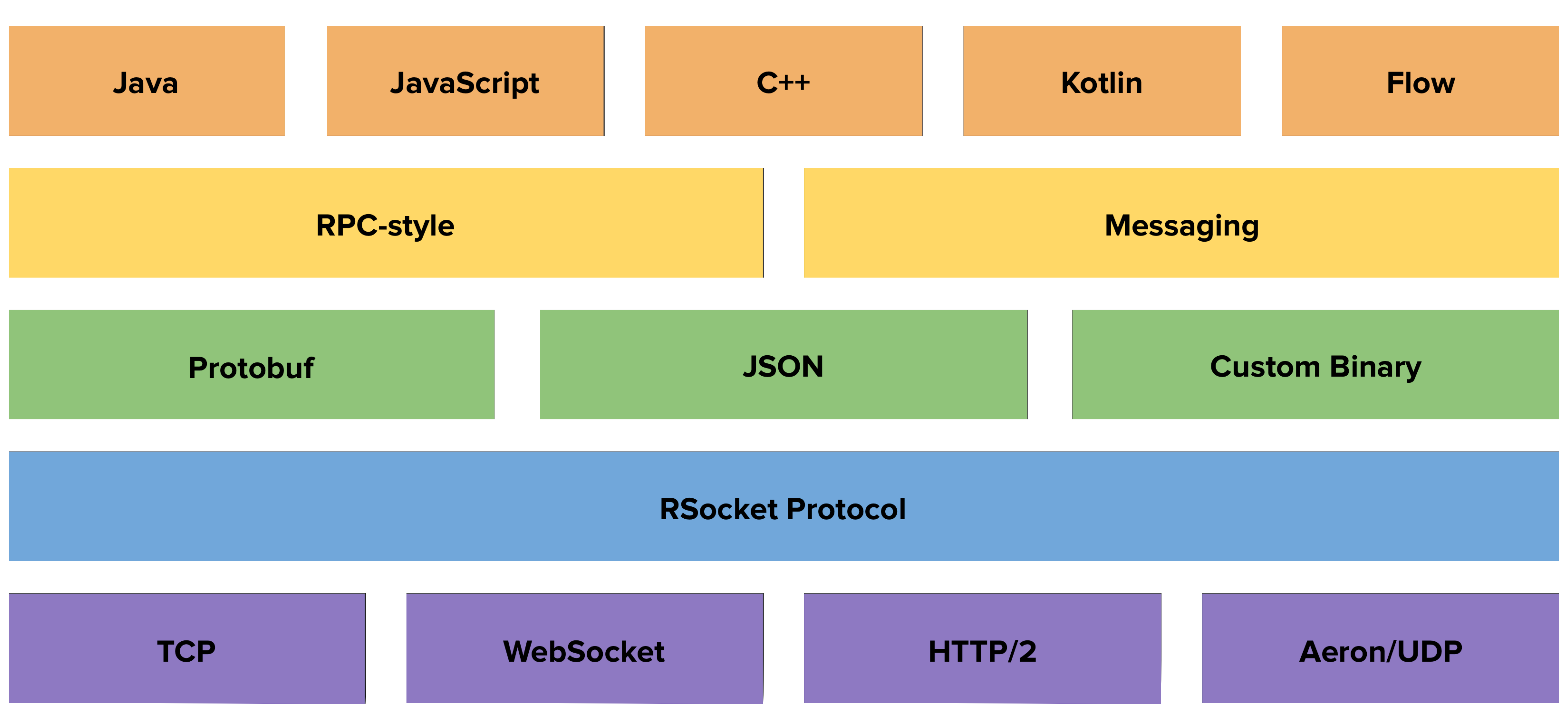
```
@Controller
@MessageMapping(“my.route.name”)
public class ExtrasController {
    …

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

SPRING-MESSAGING

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

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

SPRING-MESSAGING

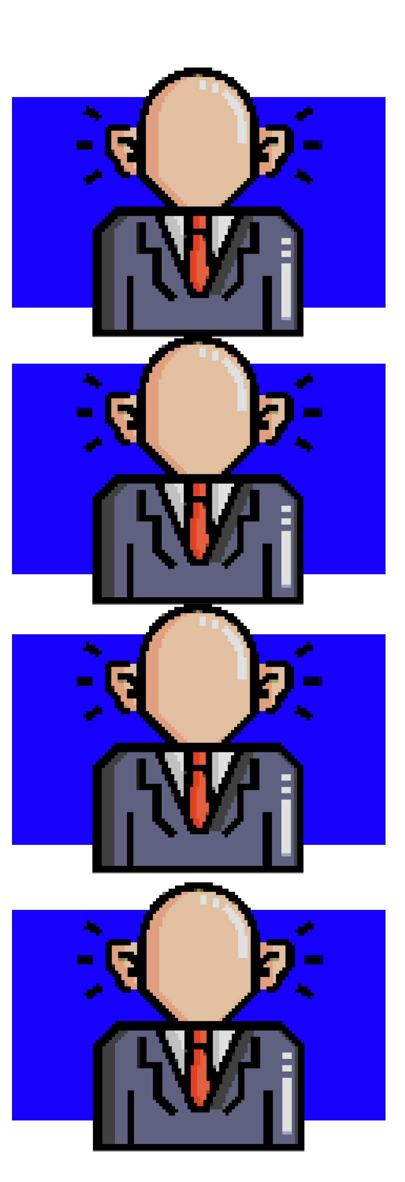
```
@Controller
@MessageMapping("my.route.name")
public class ExtrasController {
  …

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

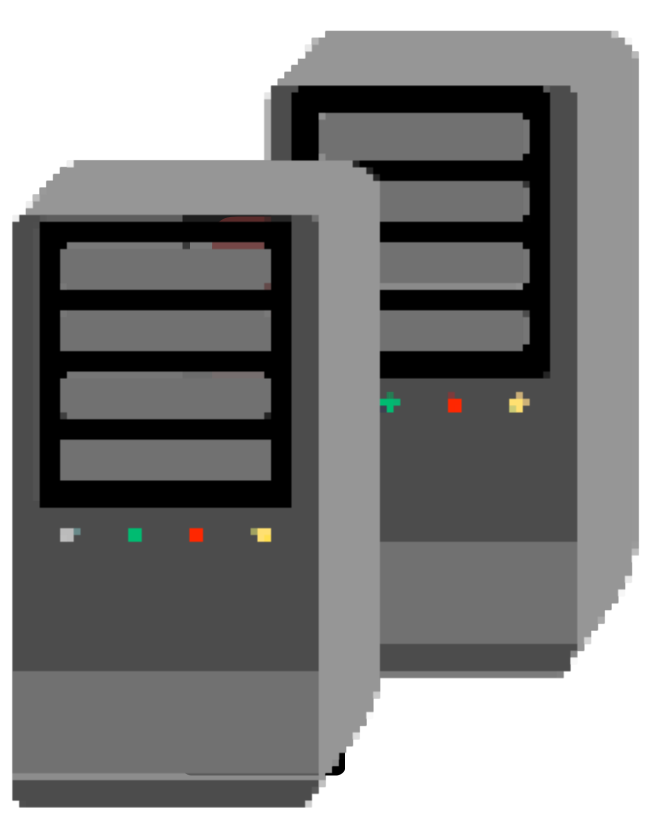


Java
JavaScript
C++
Kotlin
Flow
RPC-style
Messaging
Protobuf
JSON
Custom Binary
RSocket Protocol
TCP
WebSocket
HTTP/2
Aeron/UDP

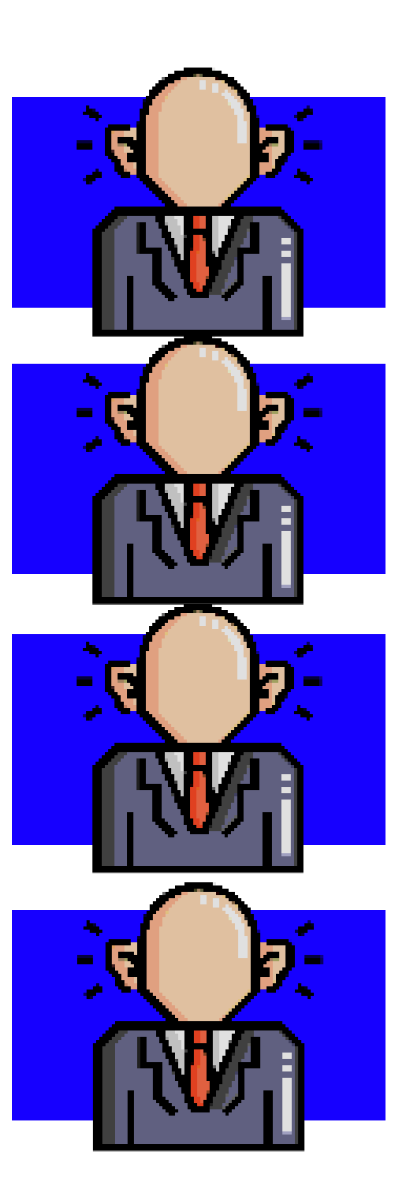
STRESS TEST

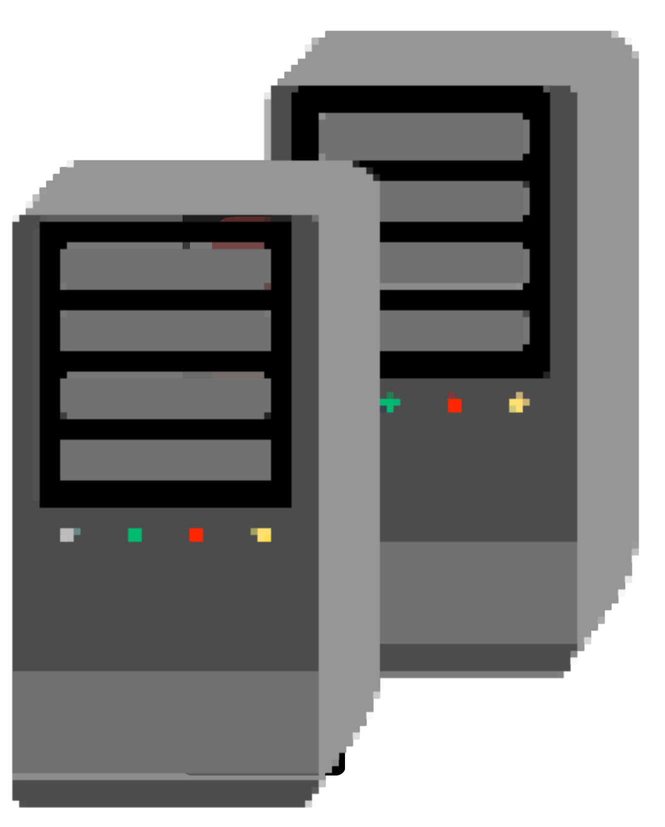
Scenario



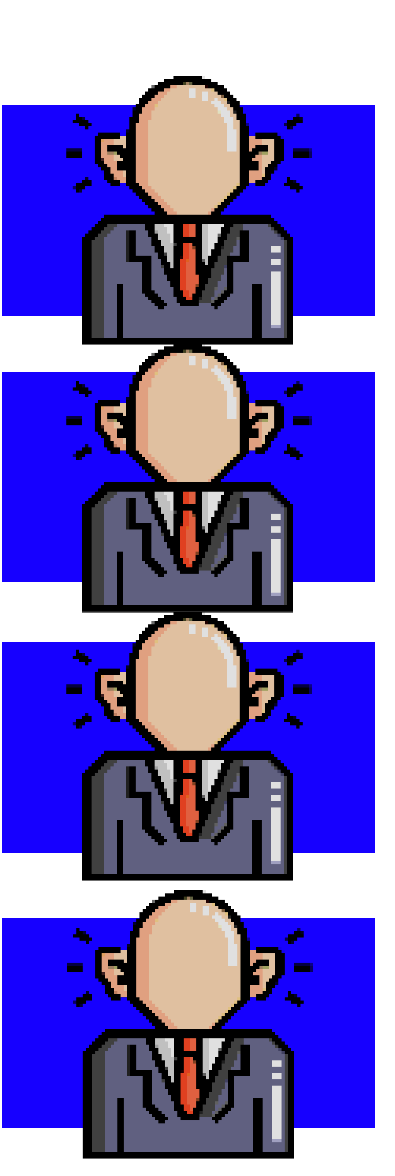


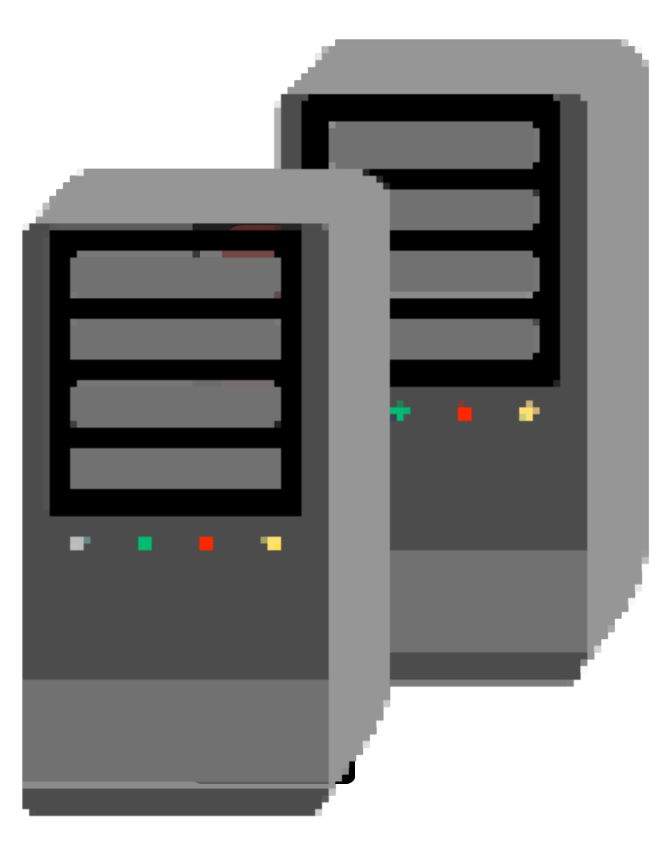
Scenario



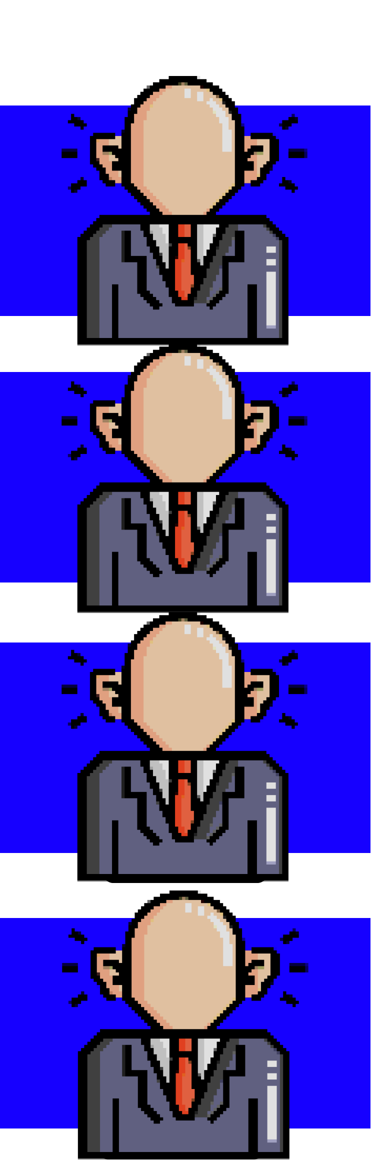


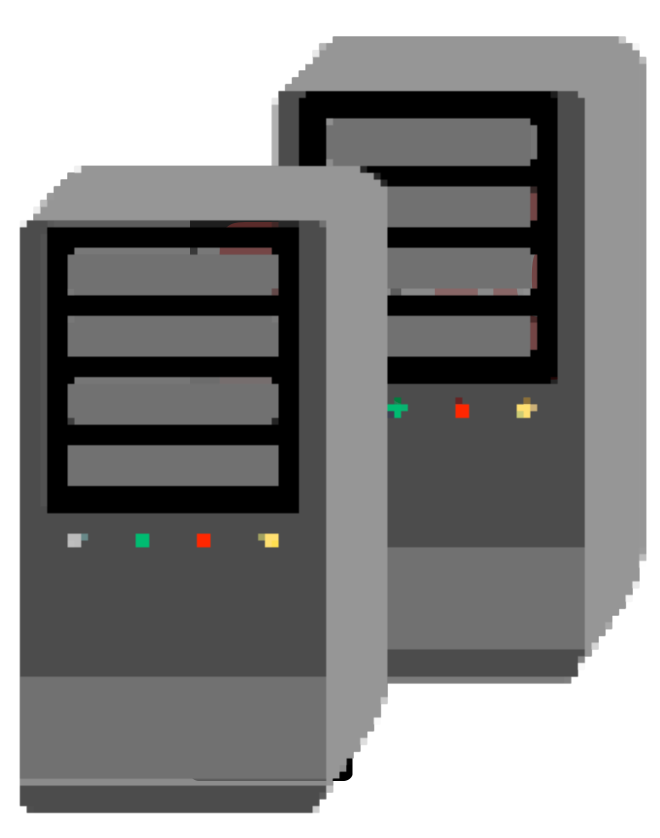
Scenario



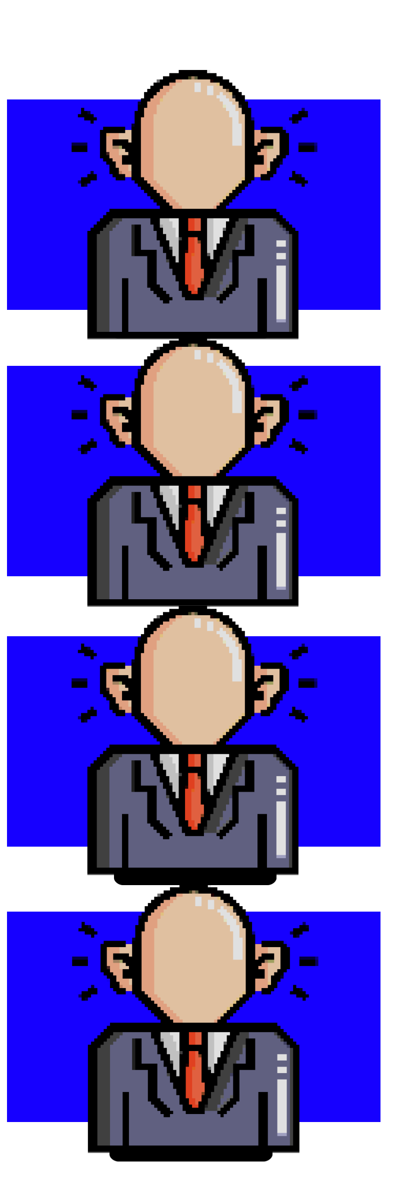


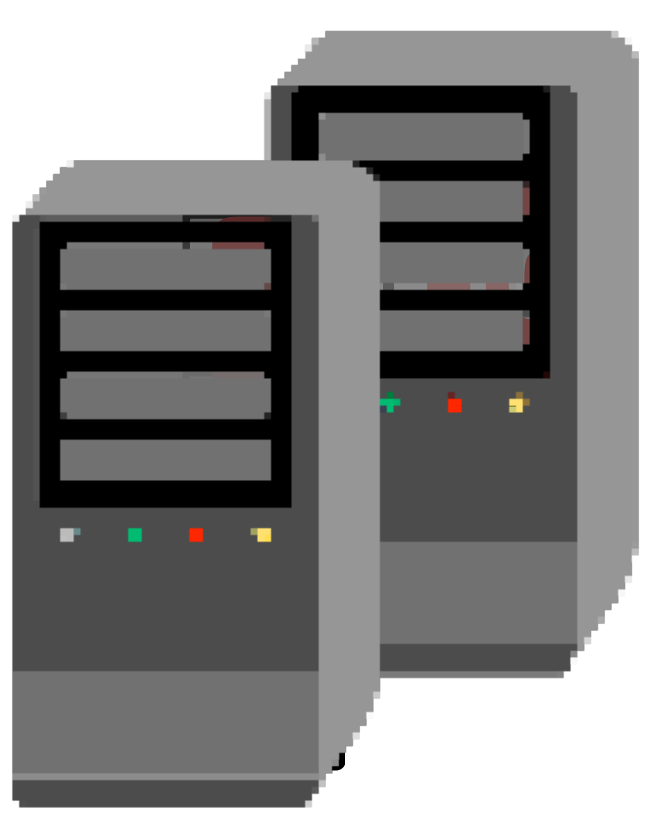
Scenario



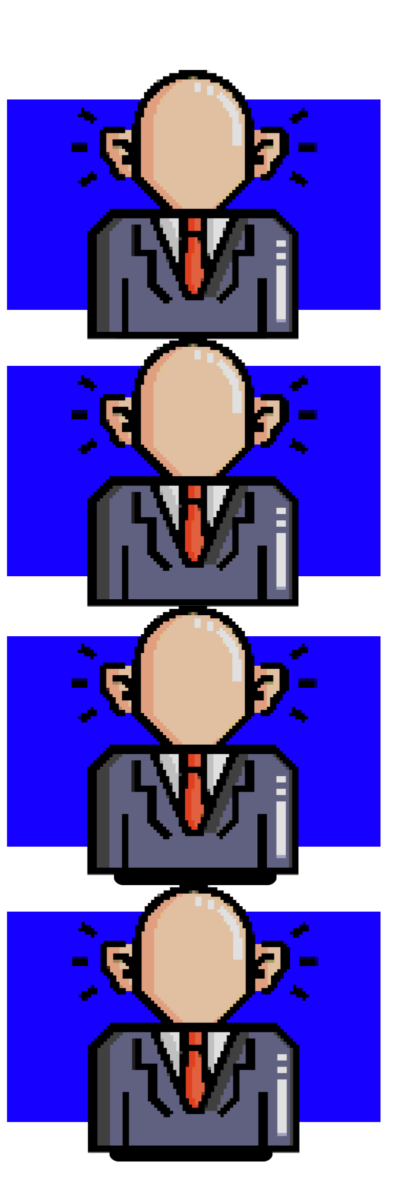


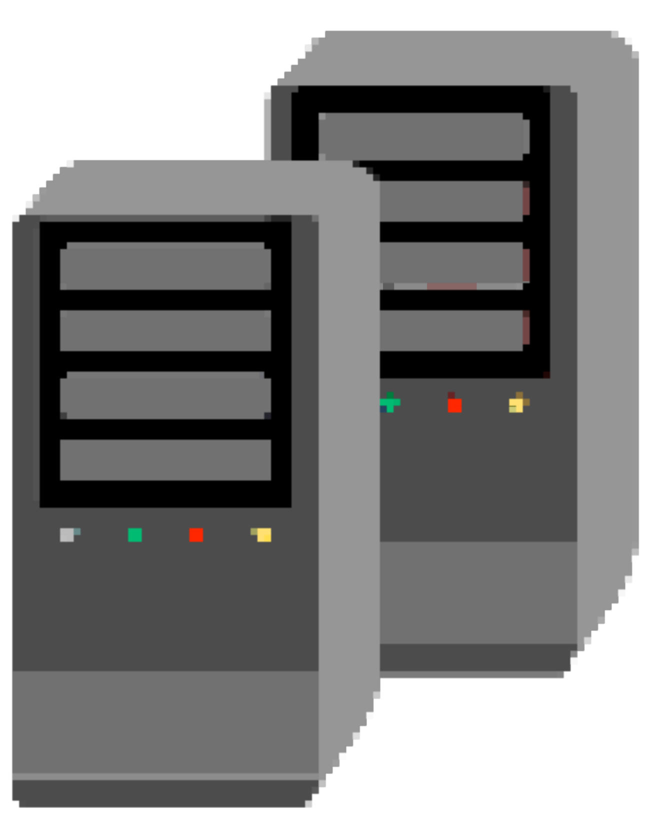
Scenario





Scenario





RSocket Publisher

game_rsocket_server_end_to_end_throughput.mean_value

350K
300K
250K
200K
150K
100K
50K
0

15:23 15:23:30 15:24 15:24:30 15:25 15:25:30 15:26 15:26:30 15:27 15:27:30 15:28 15:28:30 15:29 15:29:30 15:30 15:30:30

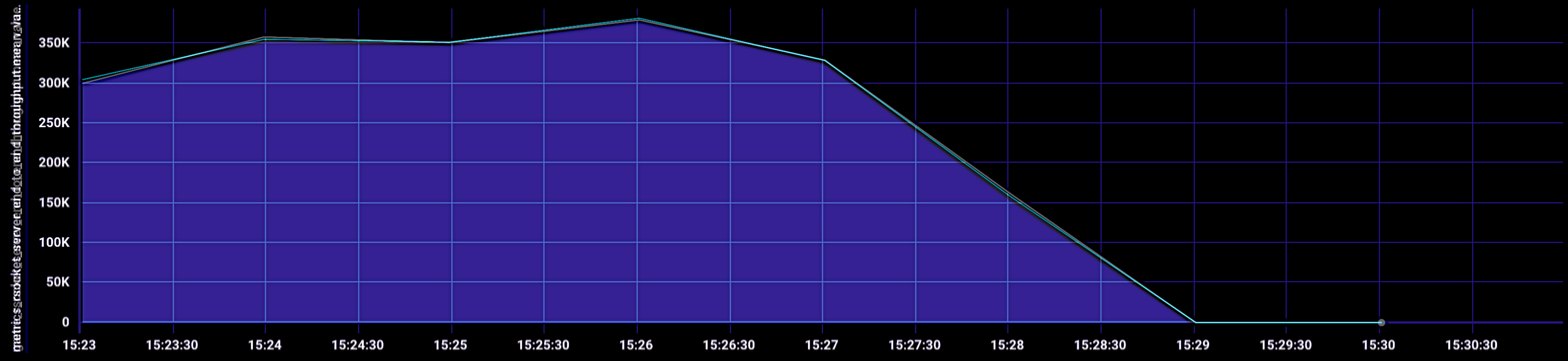
RSocket Subscriber

metrics_rsocket_server_end_to_end_throughput.mean_va...

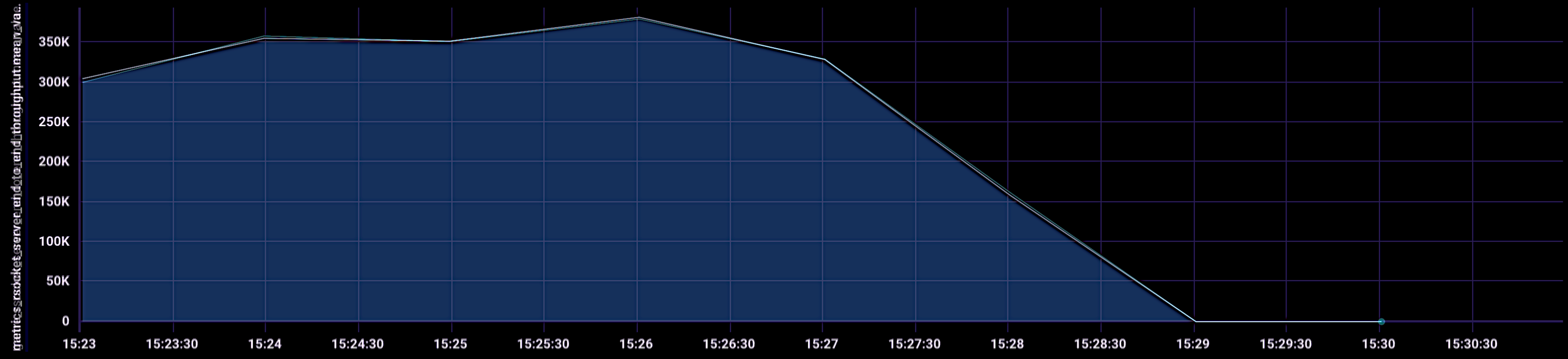
350K
300K
250K
200K
150K
100K
50K
0

15:23 15:23:30 15:24 15:24:30 15:25 15:25:30 15:26 15:26:30 15:27 15:27:30 15:28 15:28:30 15:29 15:29:30 15:30 15:30:30

RSocket Subscriber



RSocket Subscriber



Advantages

Advantages

- SIMPLICITY IN DEVELOPMENT

Advantages

- SIMPLICITY IN DEVELOPMENT
- EFFICIENT RESOURCE USAGE

Advantages

- SIMPLICITY IN DEVELOPMENT
- EFFICIENT RESOURCE USAGE
- HIGH PERFORMANCE

Advantages

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

Advantages

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

Disadvantages

Disadvantages

- STILL UNDER DEVELOPMENT

Disadvantages

- STILL UNDER DEVELOPMENT
- NARROW ADOPTION (FOR NOW)

Maintainers

NETFLIX

REACTIVE
FOUNDATION

Summary

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

Summary

Summary

- EACH PROTOCOL HAS IT`S BENEFITS

Summary

- EACH PROTOCOL HAS IT`S BENEFITS
- SOCKET.IO IS THE BEST IN JS WORLD

Summary

- EACH PROTOCOL HAS IT`S BENEFITS
- SOCKET.IO IS THE BEST IN JS WORLD
- gRPC PERFORMS REALLY WELL FOR SERVER

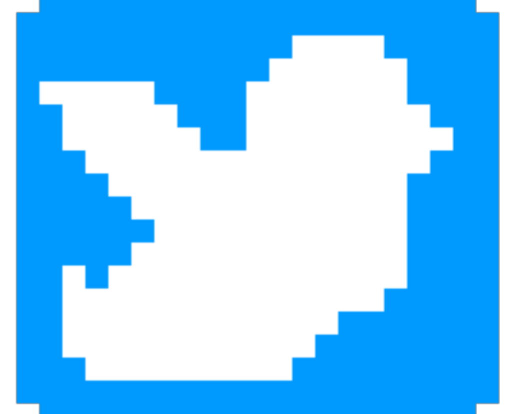
Summary

- 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

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 COVERS MOST OF CLOUD NATIVE USE CASES

Resources



@OlehDokuka

@netifi_inc

Resources

@OlehDokuka

@netifi_inc

- COMMUNITY -> https://community.netifi.com

Resources

@OlehDokuka

@netifi_inc

- COMMUNITY -> https://community.netifi.com
- VIDEO CHANNEL -> https://bit.ly/2Fku9VC

Resources

@OlehDokuka

@netifi_inc

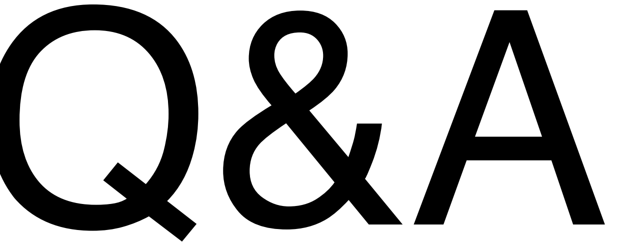
- COMMUNITY -> https://community.netifi.com
- VIDEO CHANNEL -> https://bit.ly/2Fku9VC
- RSOCKET IN SPRING -> https://bit.ly/2OiUmrD

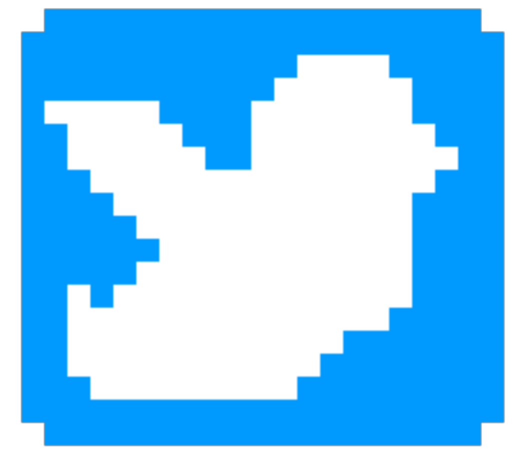
Resources

@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





@OlehDokuka

@netifi_inc

SOURCE CODE