Async Building Blocks

A Streaming Data Drama in Three Acts

Zach Mitchell - flox

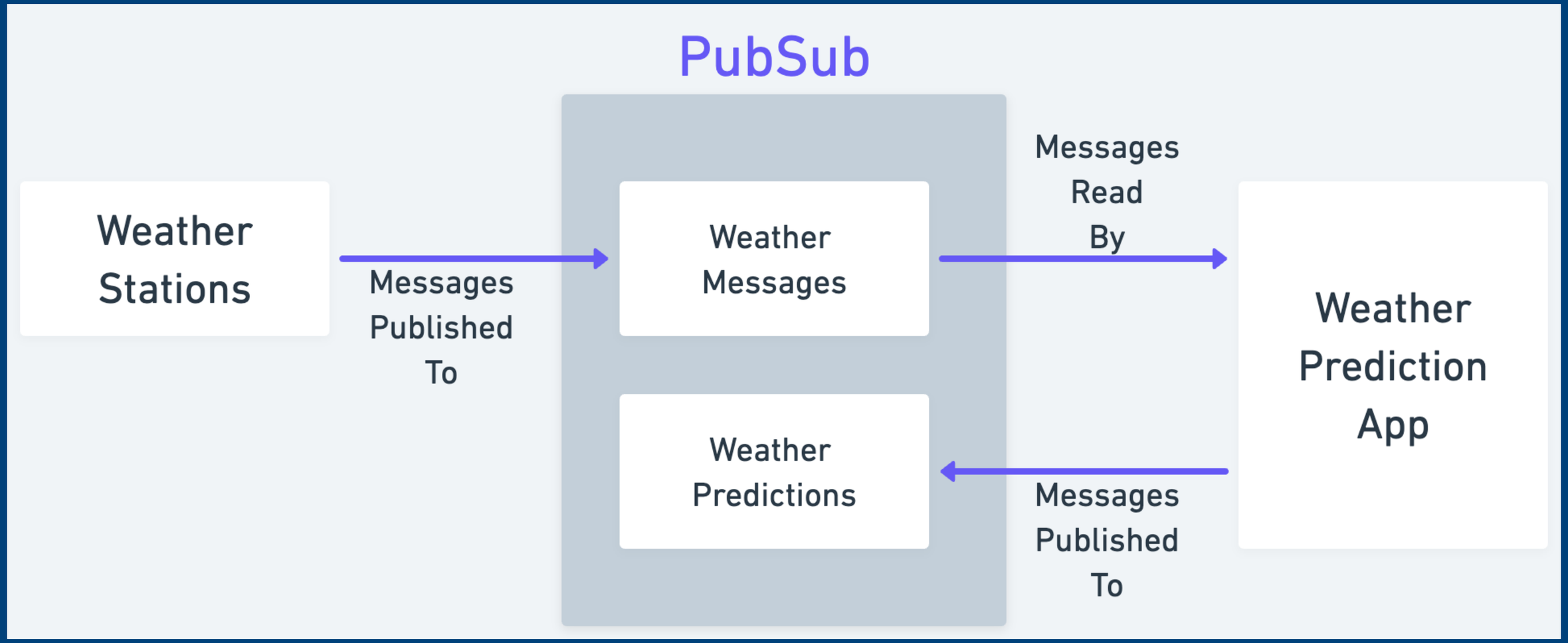
Me

- Rustacean since 2018
- Nix documentation team, working group lead
- Software engineer at flox (flox.dev)

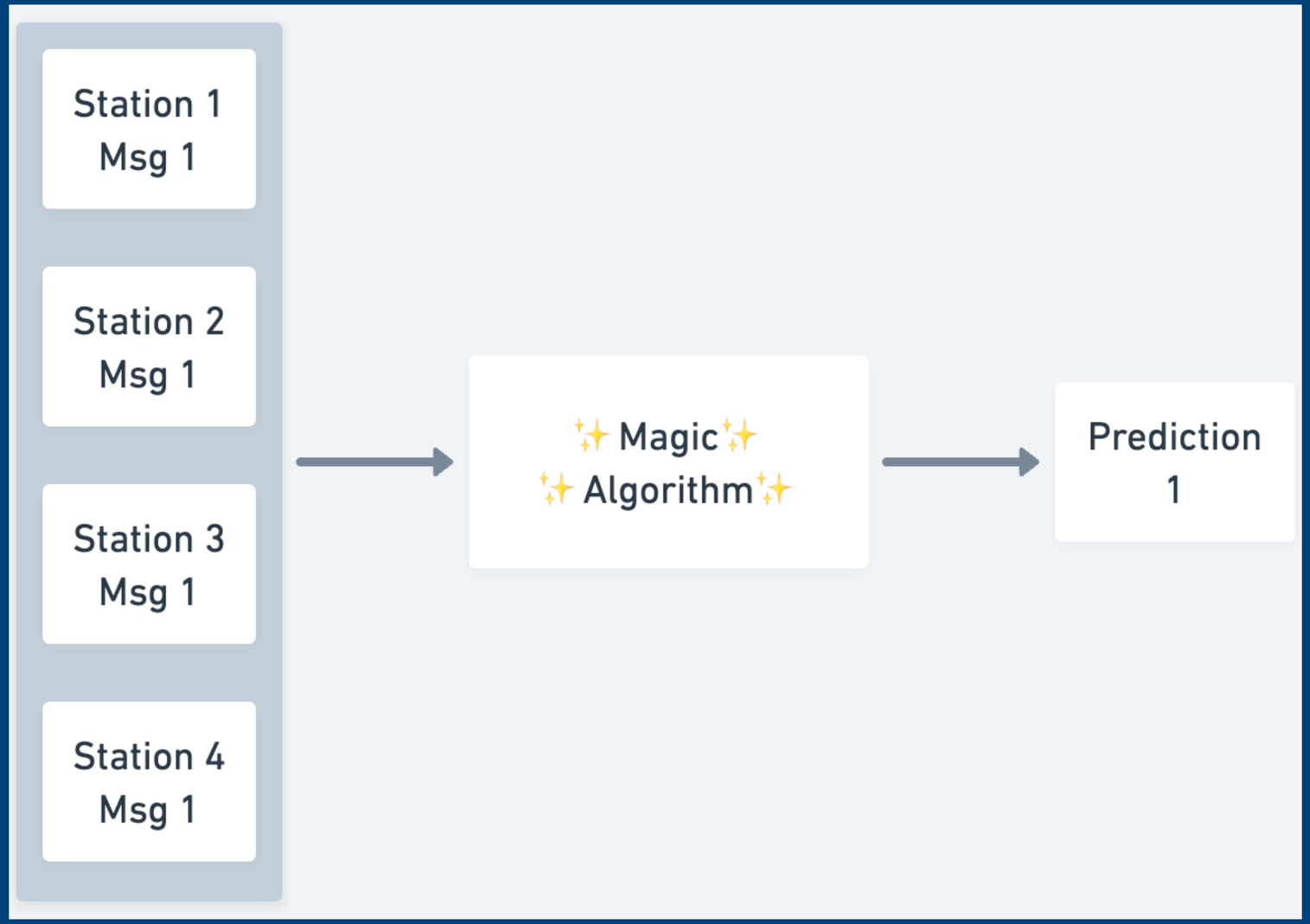
ACT I

Genesis

Setting the stage



Weather predictions



Meet Mary

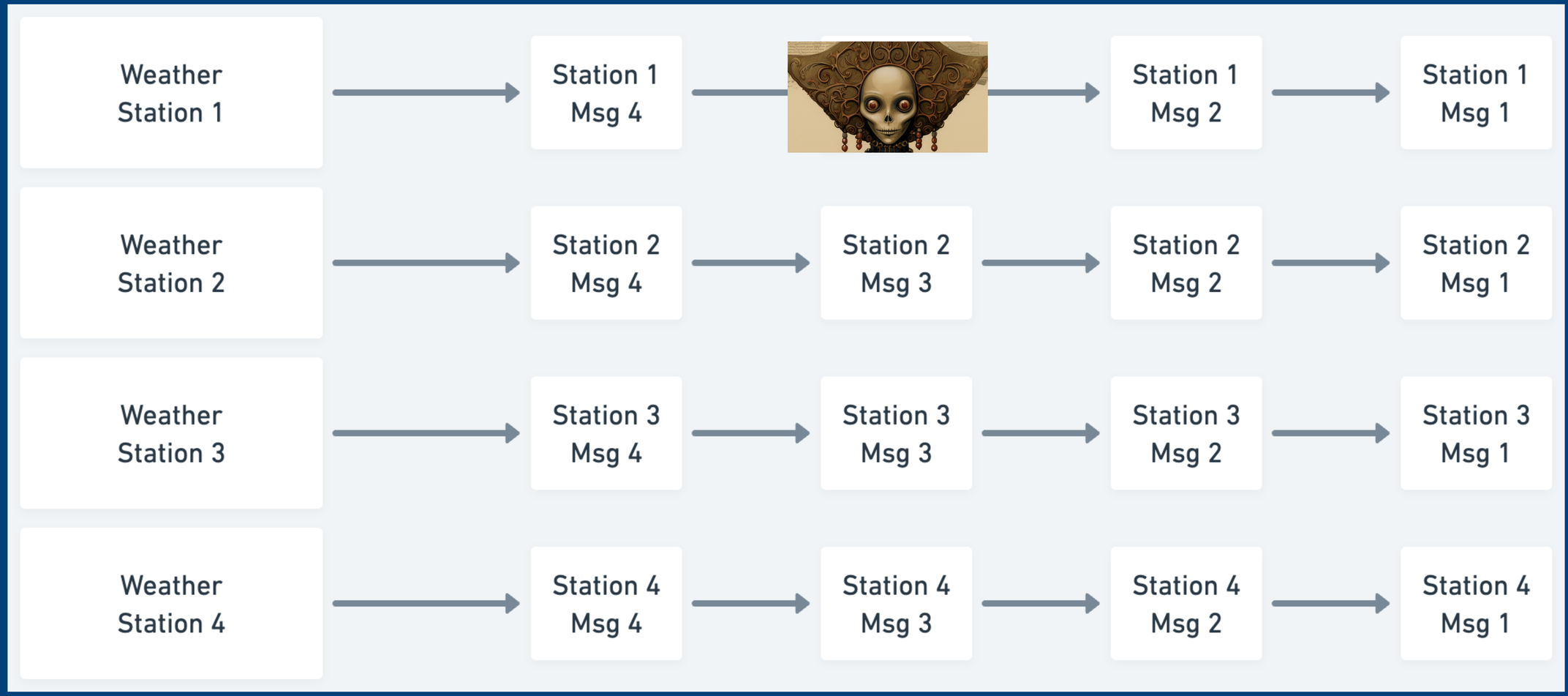
Meet Mary

"An anthropomorphized cartoon envelope named Mary the Message"

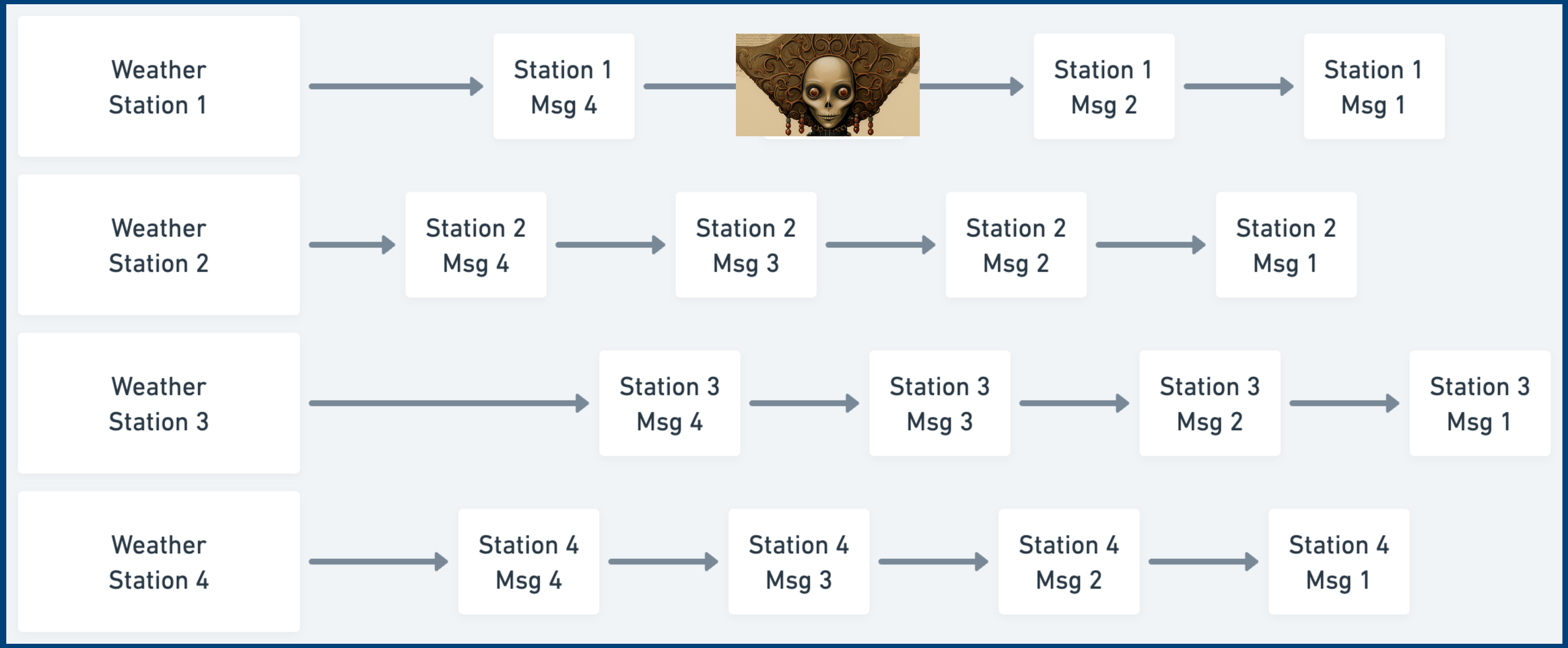
Meet Mary

"An anthropomorphized cartoon envelope named Mary the Message"

Problem: publishing synchronization

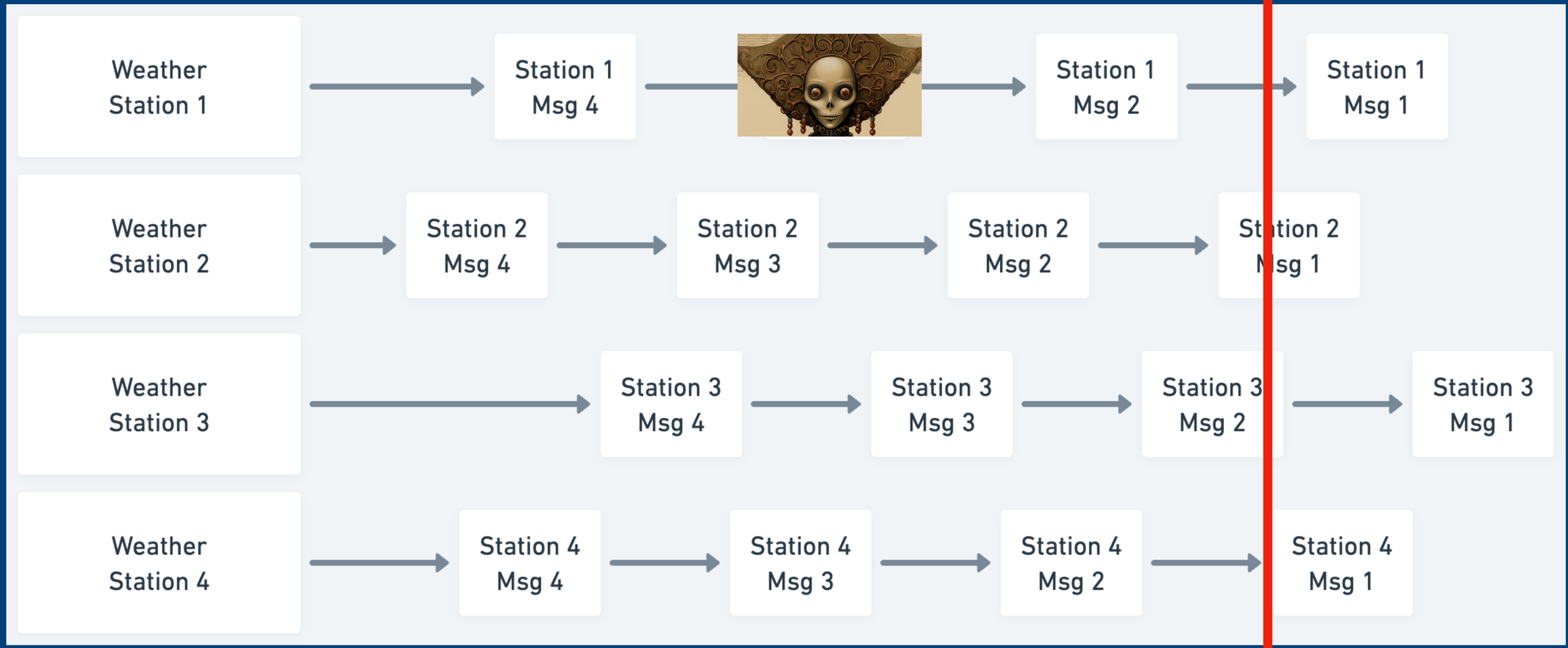


Problem: publishing synchronization



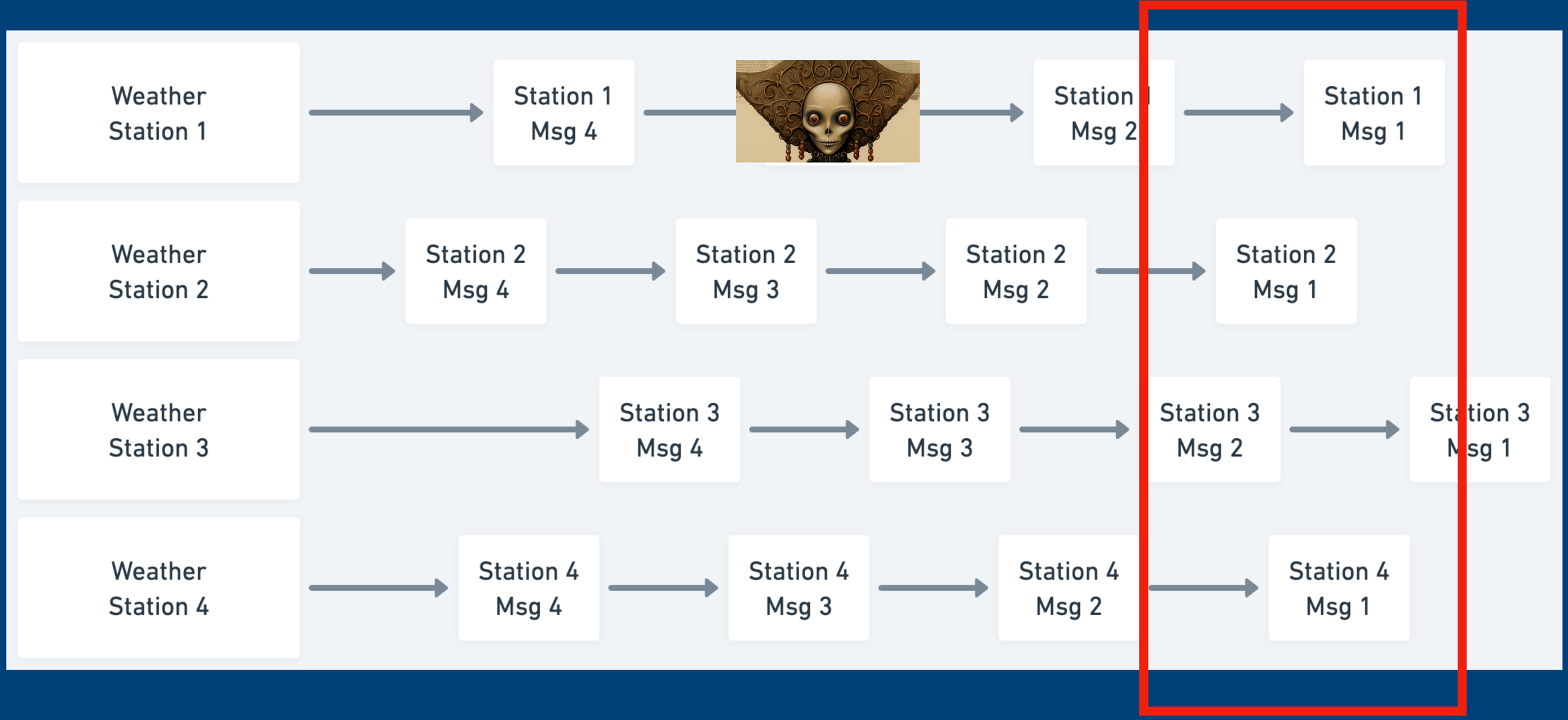
Problem: publishing synchronization

"same time"?

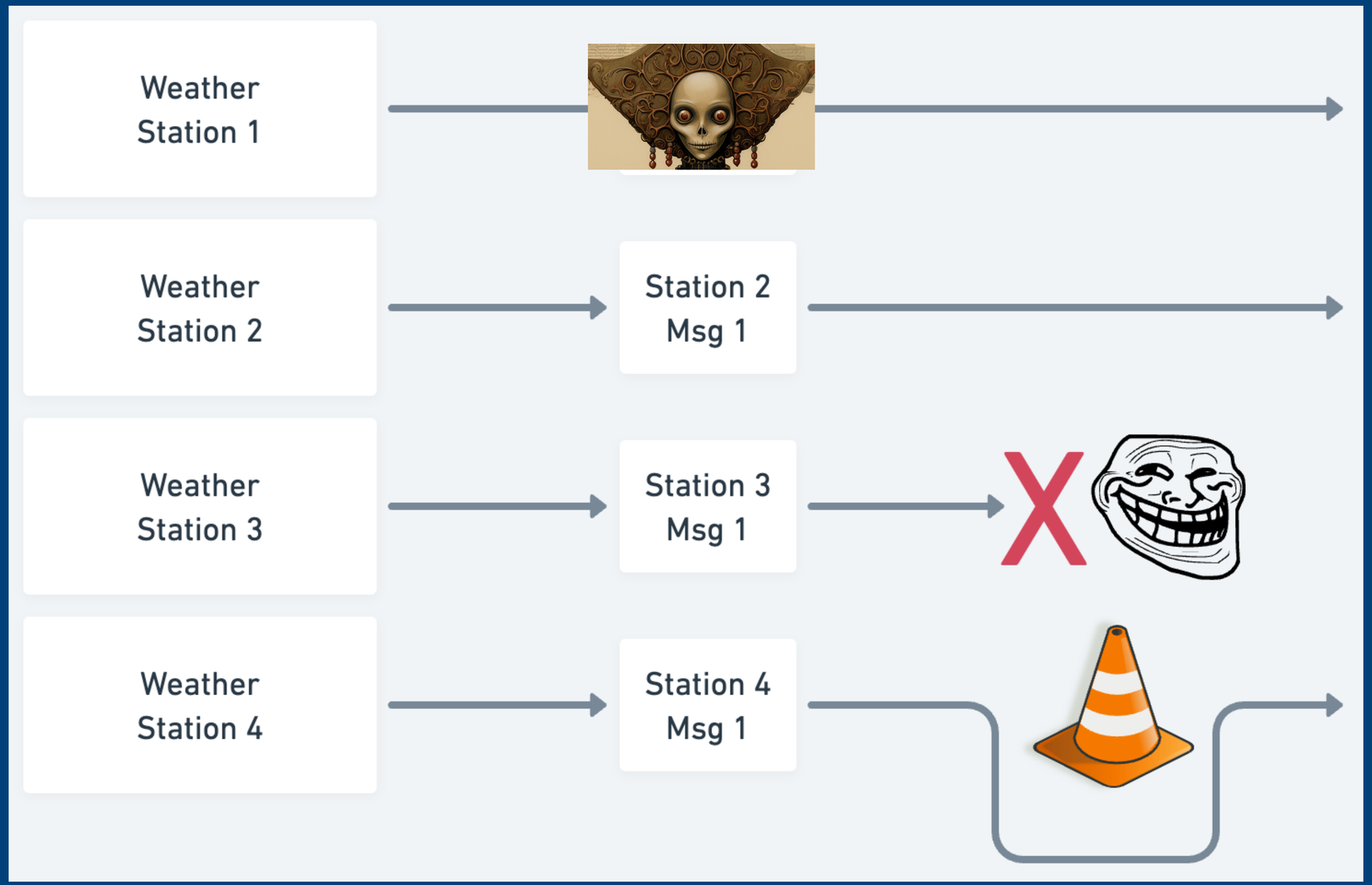


Problem: publishing synchronization

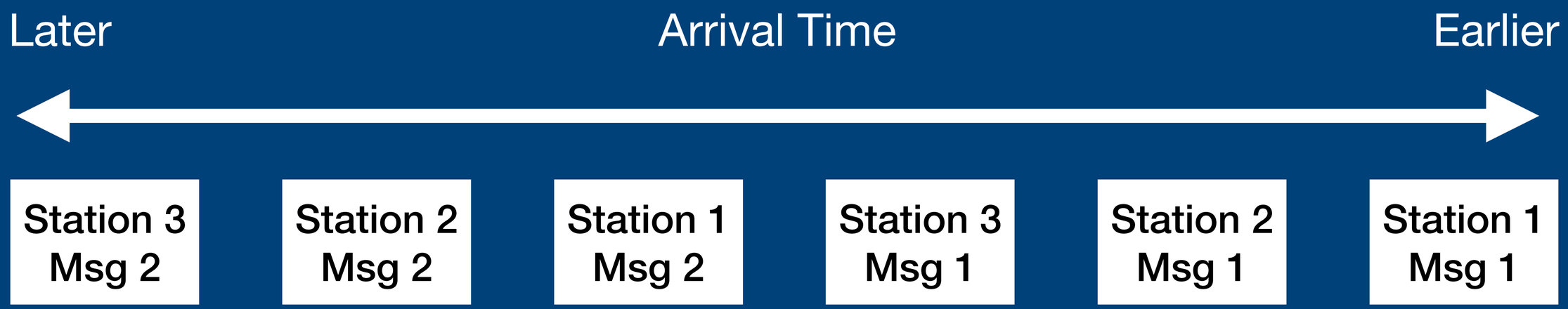
"same time"?



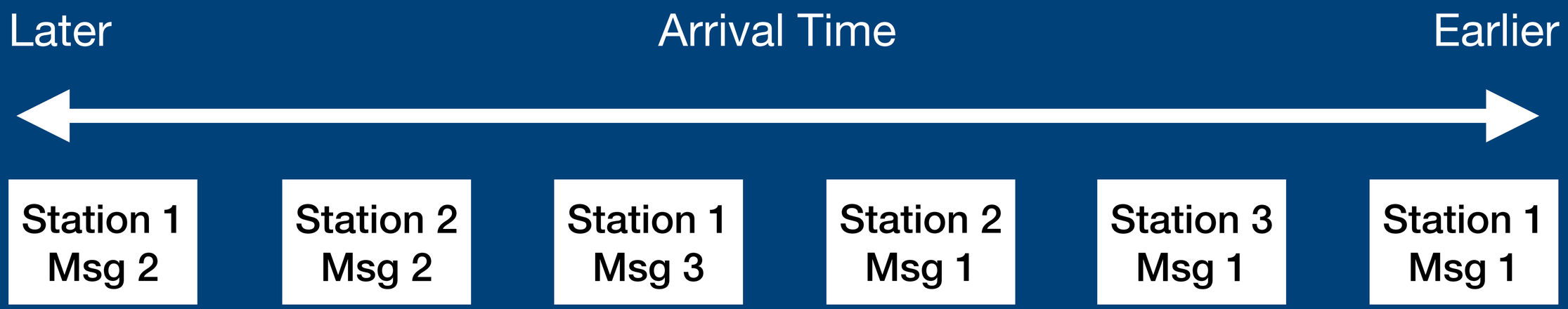
Problem: the internet



Problem: message ordering



Problem: message ordering



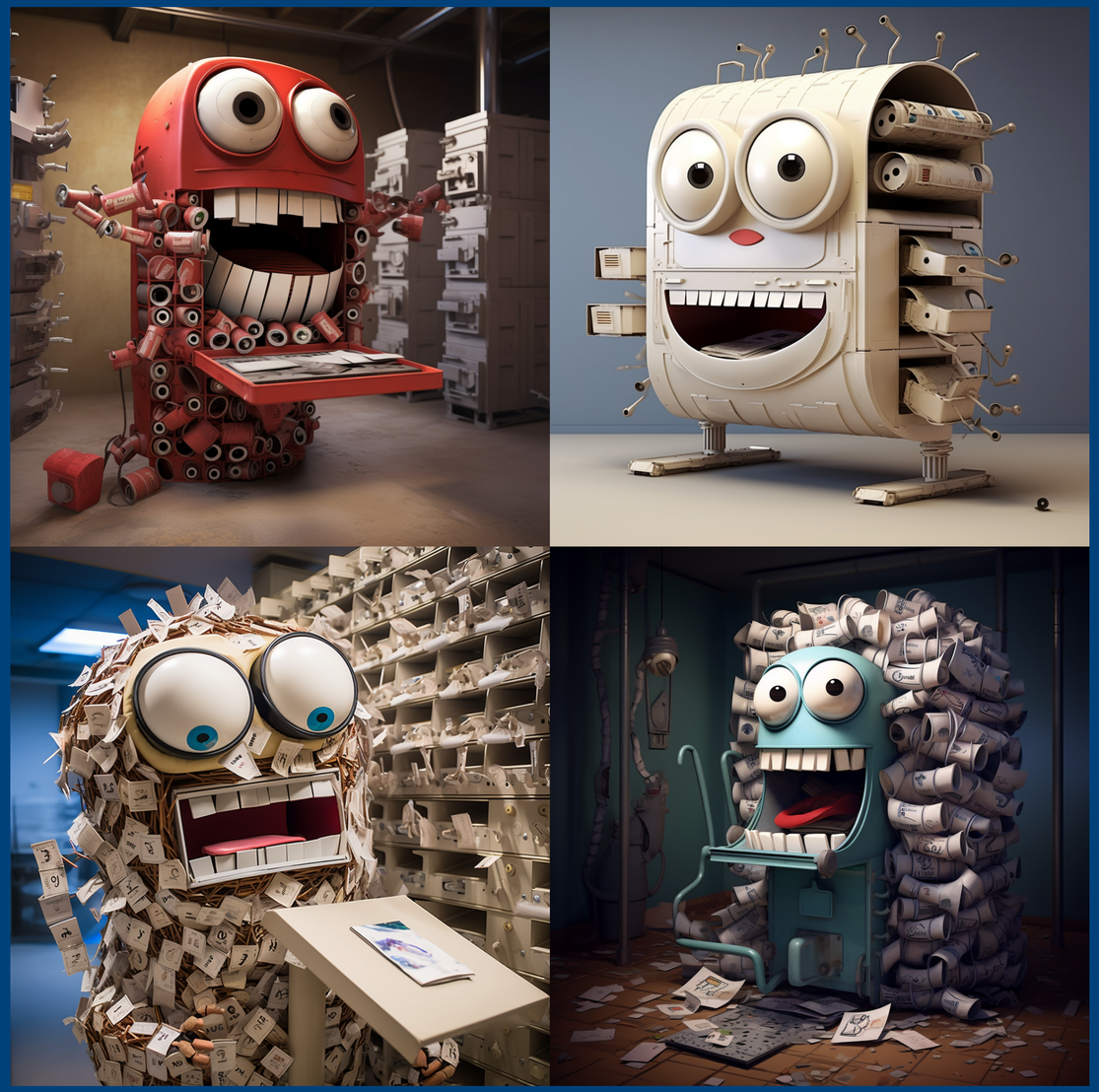
Meet Pablo PubSub

Meet Pablo PubSub

"mail sorting machine
made of tubes
embedded in a
basement, has a mouth
and eyes, cute, cartoon,
male"

Meet Pablo PubSub

"mail sorting machine made of tubes embedded in a basement, has a mouth and eyes, cute, cartoon, male"



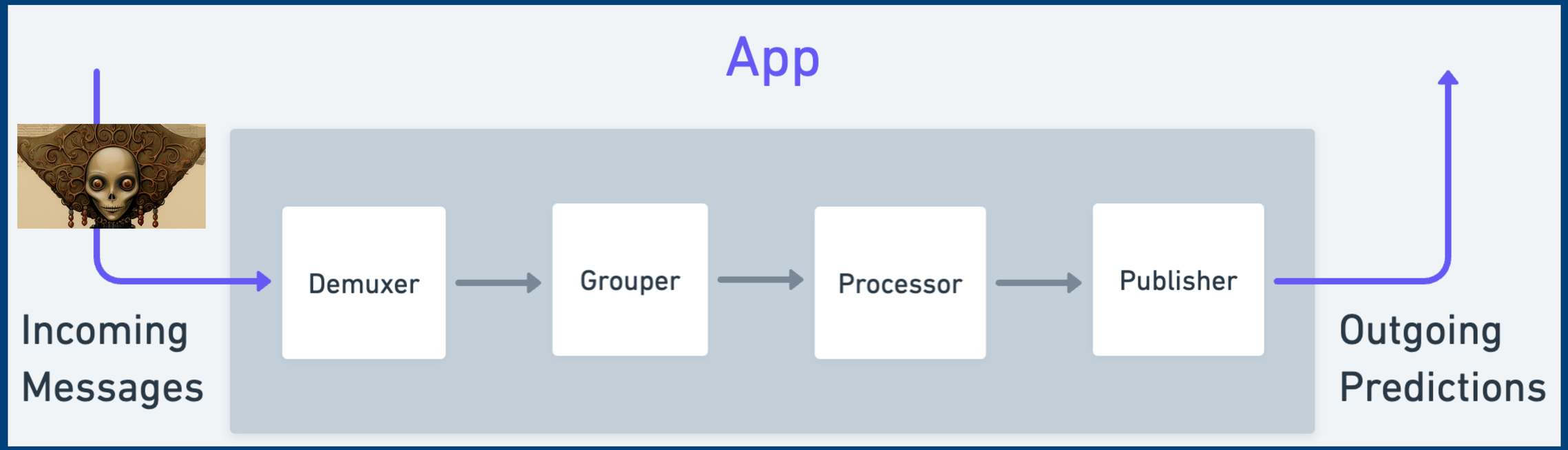
I give up, meet Pablo PubSub

Mary has an existential crisis

ACT II

The App

App overview



Modeling incoming messages

- Something that produces a sequence of messages
  - Synchronous -> Iterator
  - Asynchronous -> Stream
- Low-level interface is in `futures::Stream`
  - Stream-handling goodies are in `futures::StreamExt`
  - Stream-producing goodies are in `async_stream`

Problem: message ordering

- Messages are out of order
- Messages arrive late
- All I want is the next set of messages!
- Solution: buffering!

Meet Beatrix Buffer

Meet Beatrix Buffer

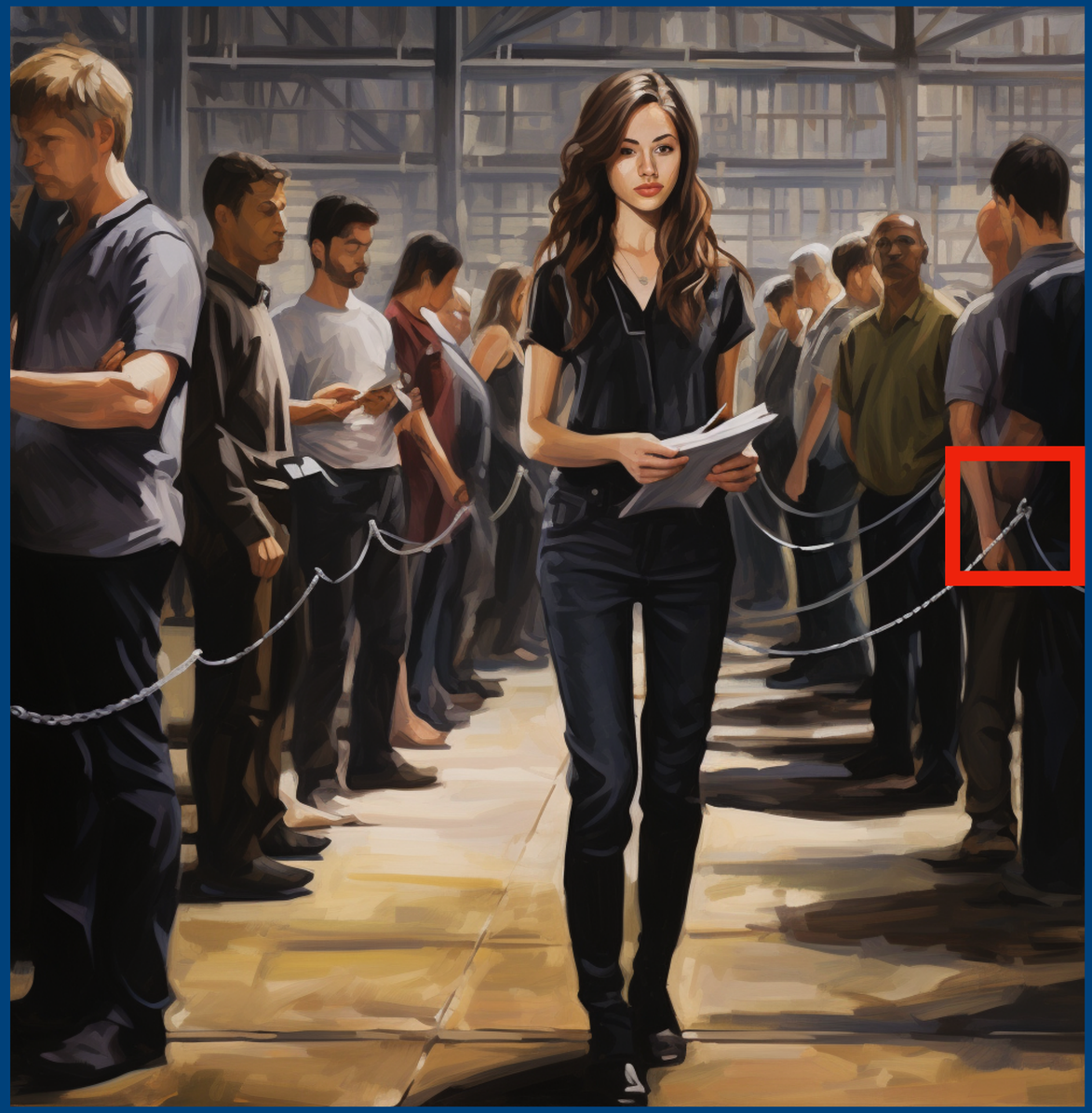
"A woman with a clipboard and a lanyard keeping several people in orderly lines separated by stanchions in an industrial warehouse"

Meet Beatrix Buffer

"A woman with a clipboard and a lanyard keeping several people in orderly lines separated by stanchions in an industrial warehouse"

Meet Beatrix Buffer

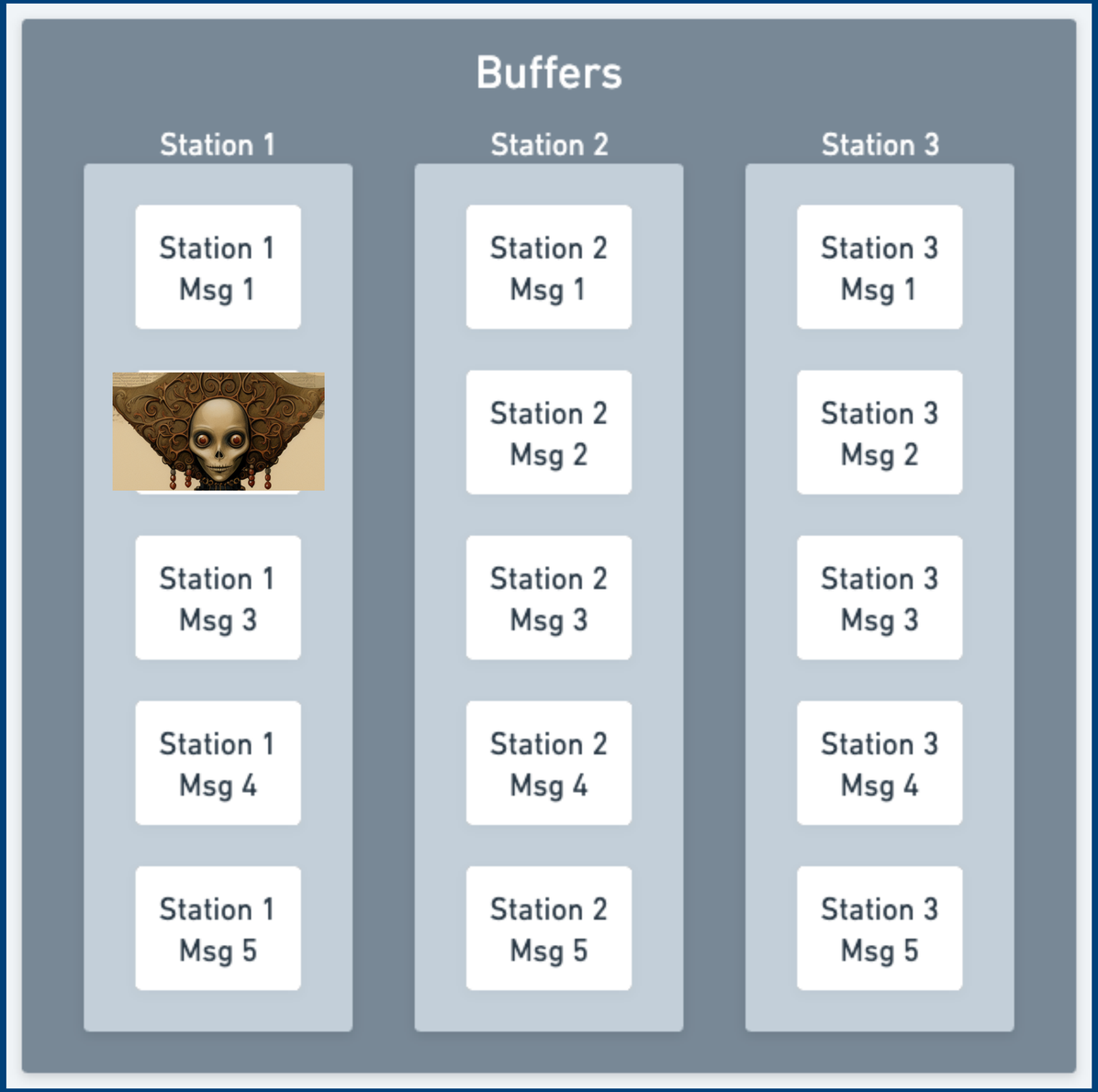
"A woman with a clipboard and a lanyard keeping several people in orderly lines separated by stanchions in an industrial warehouse"



Meet Beatrix Buffer

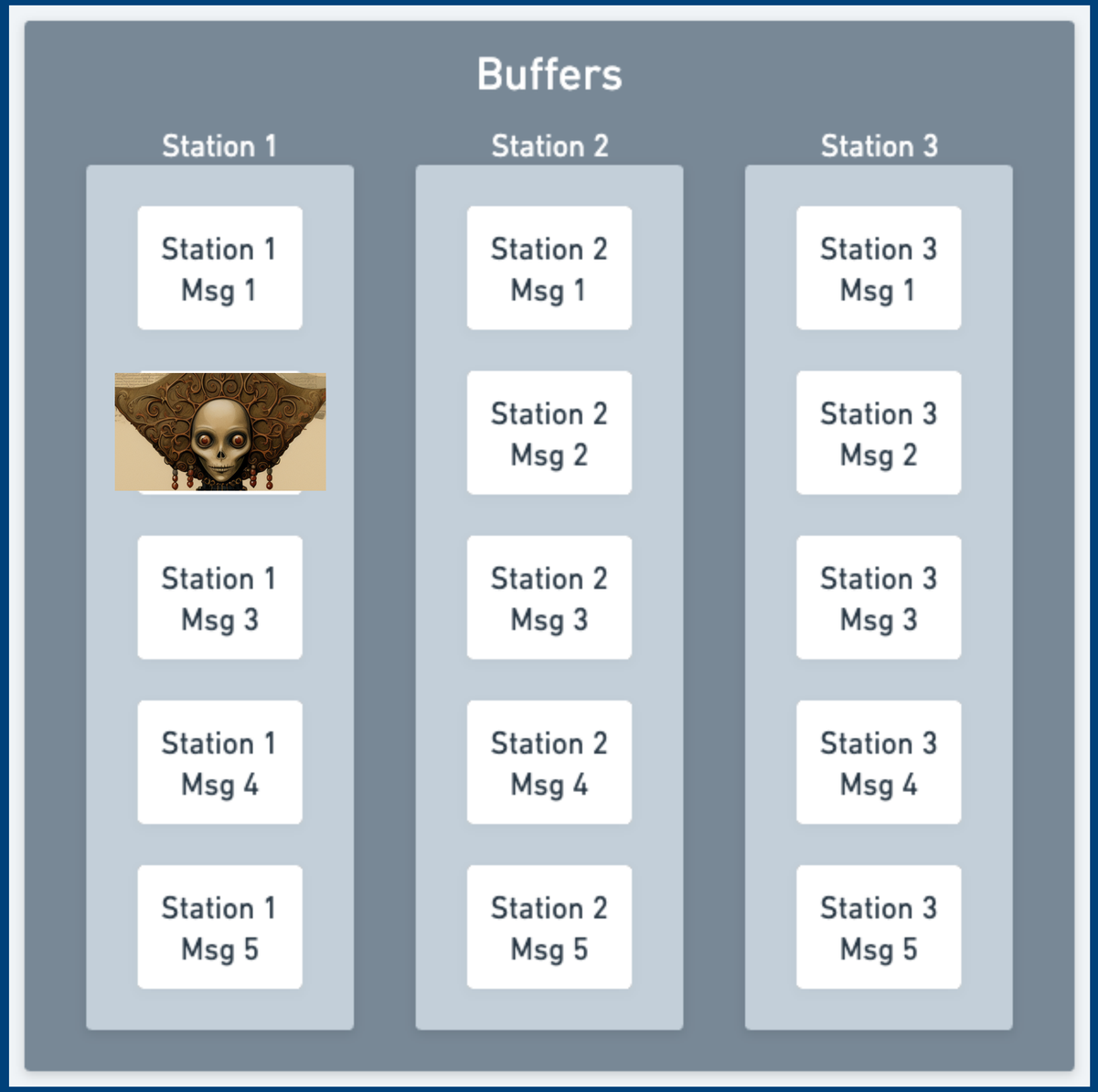
"A woman with a clipboard and a lanyard keeping several people in orderly lines separated by stanchions in an industrial warehouse"

Buffers


Buffers
Station 1
Station 2
Station 3
Station 1 Msg 1
Station 2 Msg 1
Station 3 Msg 1
Station 2 Msg 2
Station 3 Msg 2
Station 1 Msg 3
Station 2 Msg 3
Station 3 Msg 3
Station 1 Msg 4
Station 2 Msg 4
Station 3 Msg 4
Station 1 Msg 5
Station 2 Msg 5
Station 3 Msg 5

Buffers

```
type Buffers =
HashMap<
 u8, // Weather station
 VecDeque<WeatherMessage>
>
```



Meet Dino Demuxer

- "Demux" = "demultiplex"
- Ingests the mixed message stream
- Sends messages to the correct buffer
- Beatrix maintains order within that buffer

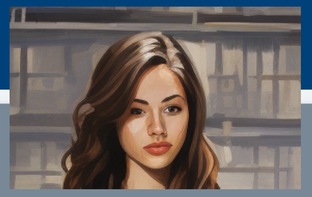
Meet Dino Demuxer

- "Demux" = "demultiplex"
- Ingests the mixed message stream
- Sends messages to the correct buffer
- Beatrix maintains order within that buffer

???

Message insertion

Dino brings the message to the correct buffer

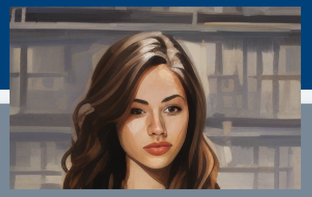


Buffers

Station 1	Station 2	Station 3
Station 1 Msg 2	Station 2 Msg 1	Station 3 Msg 1
Station 1 Msg 4	Station 2 Msg 2	Station 3 Msg 2
Station 1 Msg 3	Station 2 Msg 3	Station 3 Msg 3
Station 1 Msg 5	Station 2 Msg 4	Station 3 Msg 4
	Station 2 Msg 5	Station 3 Msg 5

Message insertion

Dino brings the message to the correct buffer

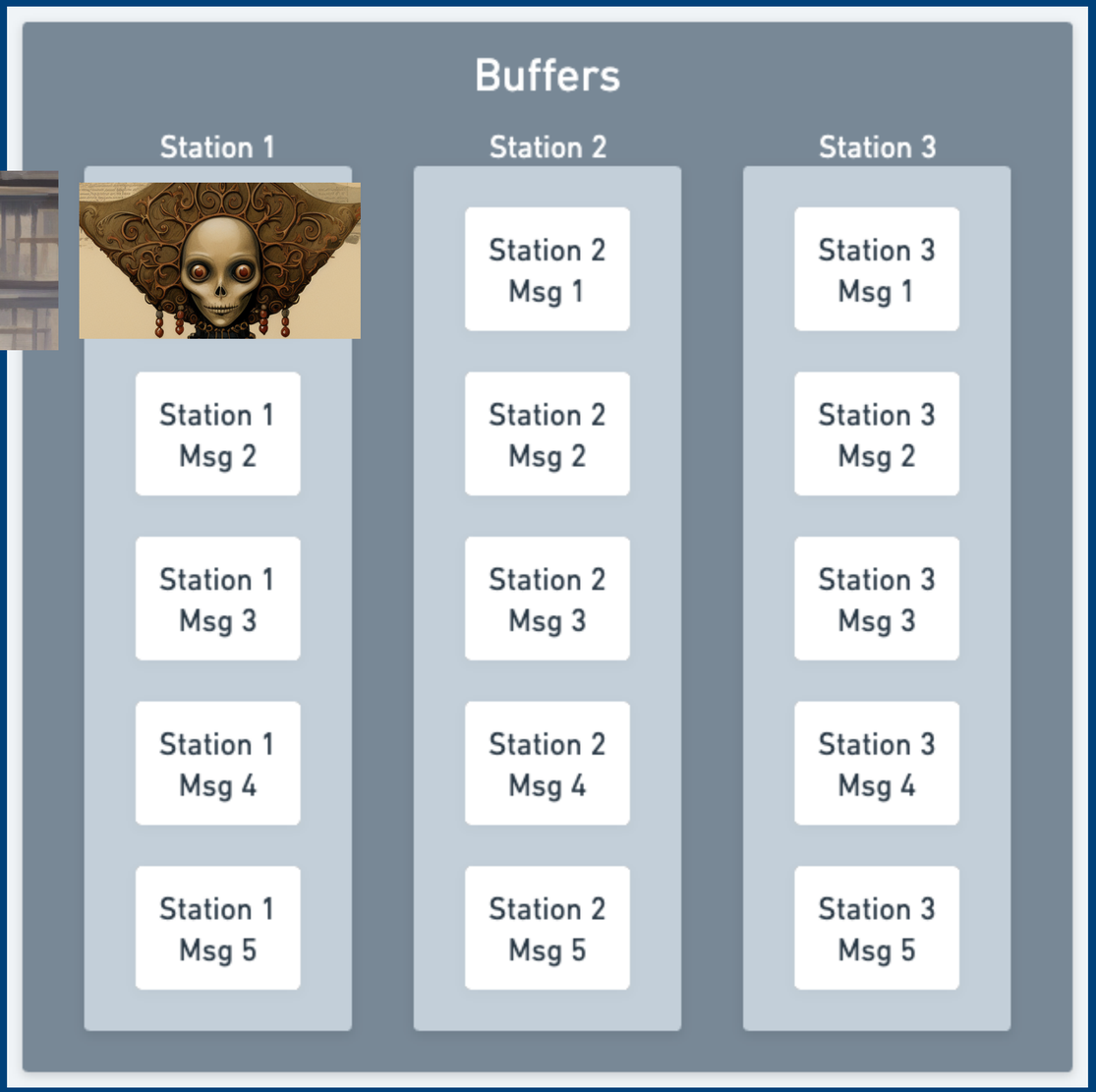


Buffers

Station 1	Station 2	Station 3
Station 1 Msg 2	Station 2 Msg 1	Station 3 Msg 1
Station 1 Msg 4	Station 2 Msg 2	Station 3 Msg 2
Station 1 Msg 3	Station 2 Msg 3	Station 3 Msg 3
Station 1 Msg 5	Station 2 Msg 4	Station 3 Msg 4
	Station 2 Msg 5	Station 3 Msg 5

Message insertion

Beatrix puts the message in the correct order

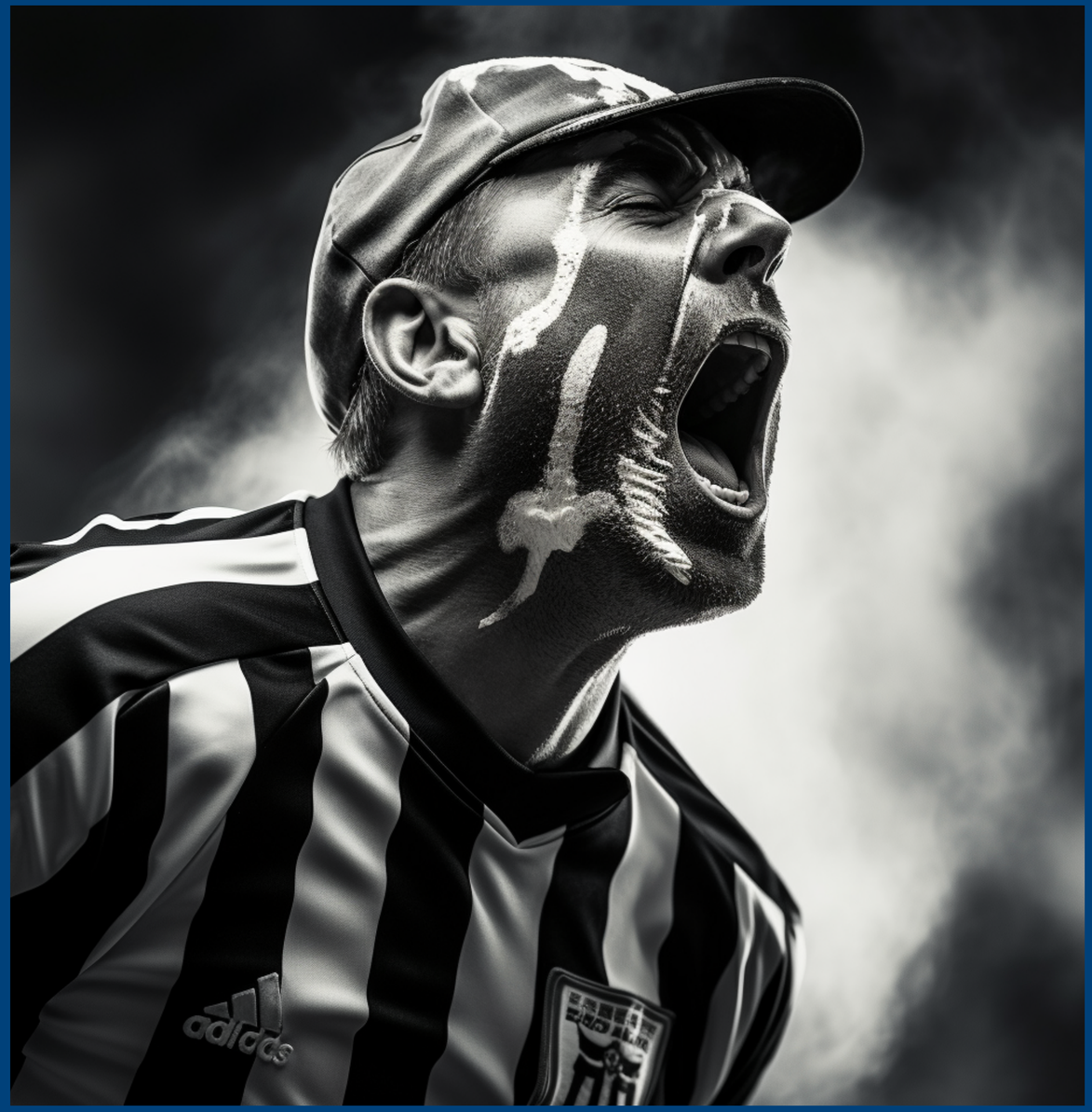


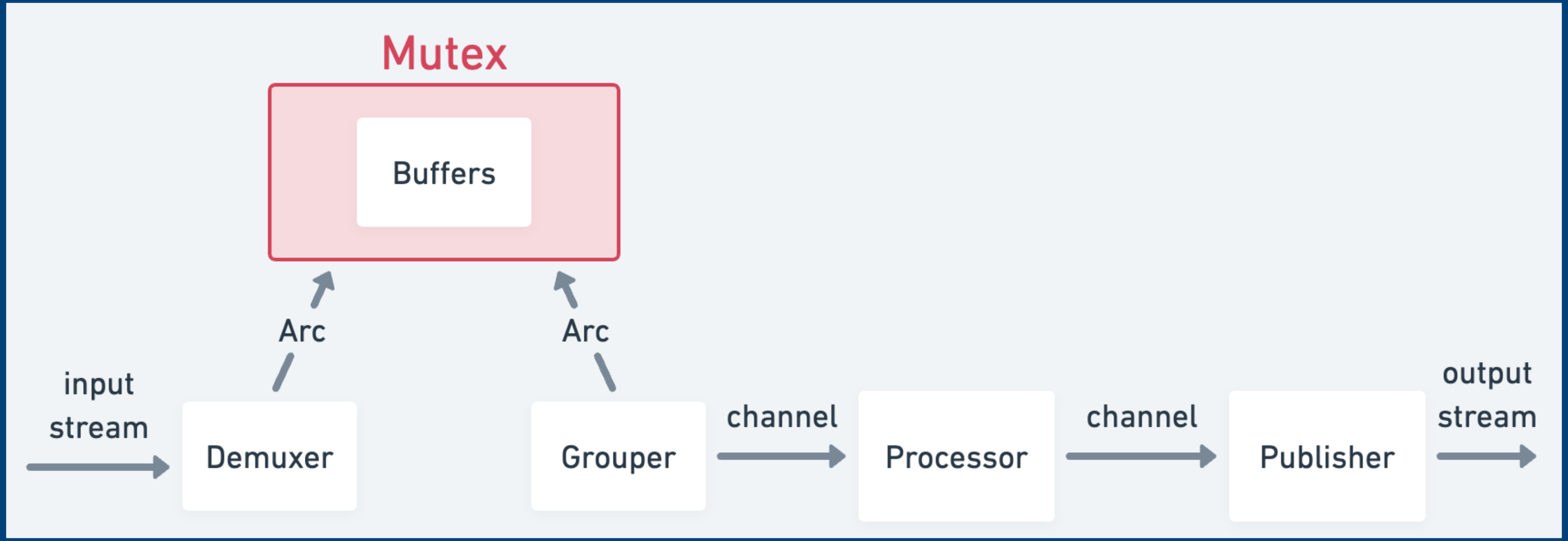
Problem: coordinating buffer access

- Dino tells Mary to wait with Beatrix
- Beatrix puts Mary in the right position in line
- Greta Grouper pulls messages out of the buffer
- How do we prevent a race condition?

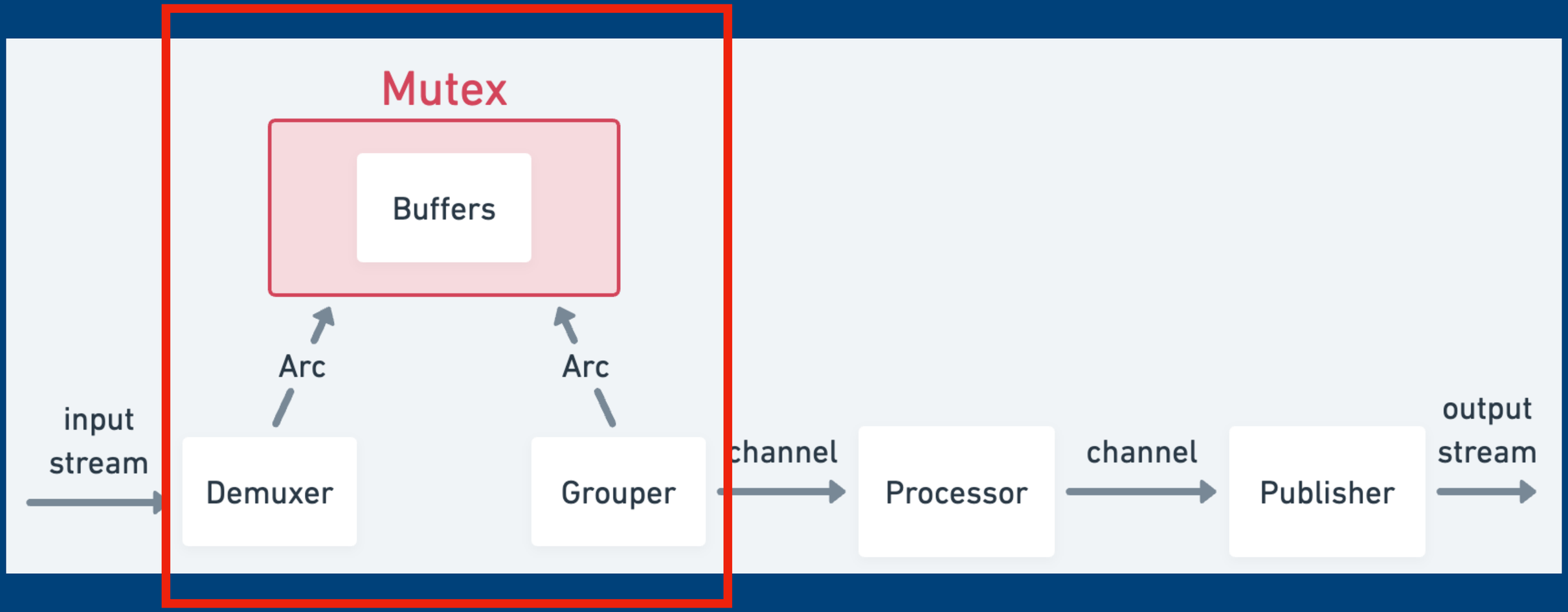
Meet Mitch Mutex

```
Arc<Mutex<Buffers>>
```




Mutex
Buffers
Arc
Arc
input
stream
Demuxer
Grouper
channel
Processor
channel
Publisher
output
stream

YOU ARE HERE


Mutex
Buffers
Arc
Arc
input
stream
Demuxer
Grouper
channel
Processor
channel
Publisher
output
stream

Problem: parallel execution

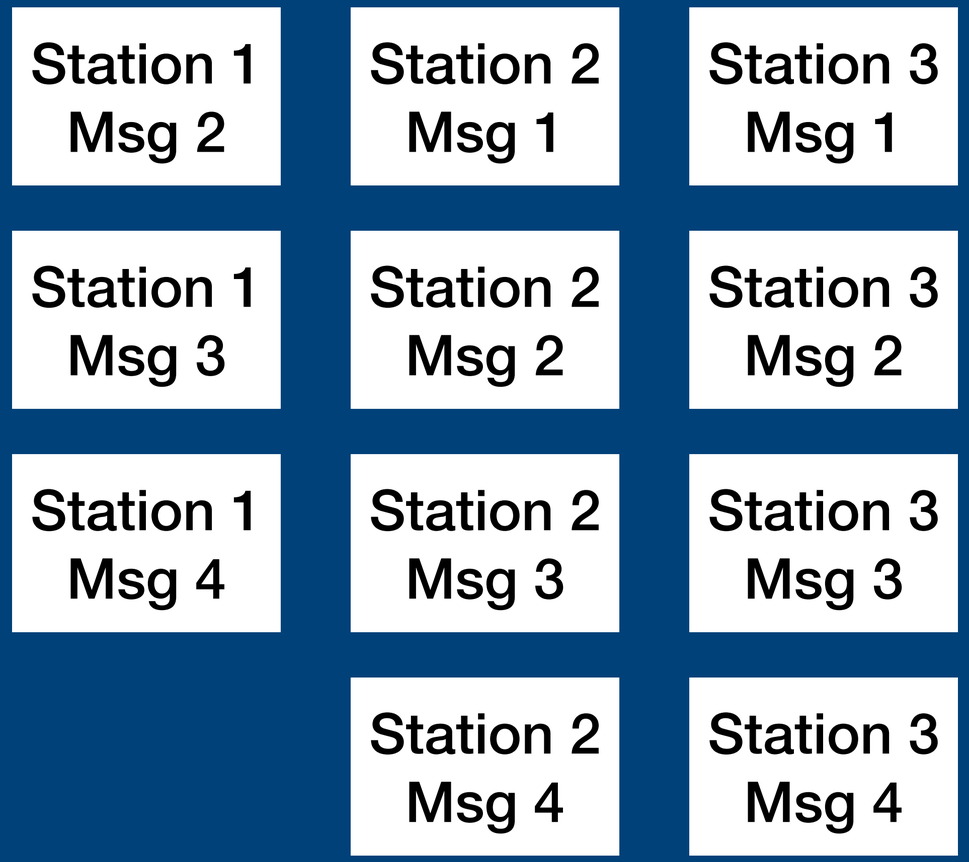
- Futures can execute concurrently on a single thread
- Sometimes you still want them to execute *at the same time*
- Solution: tasks

```
let demuxer_future = demuxer(...);
let demuxer_handle = tokio::spawn(demuxer_future);
...(other tasks)...
futures::select!{...};
```

Problem: grouping messages

- Prediction algorithm wants a snapshot of the world
- We need messages from the "same time"
- Lots of tricky details here
  - How long do you wait for late arrivals?
  - "Same time" doesn't exist
  - etc

Problem: grouping messages



Problem: grouping messages

Station 1 Msg 2	Station 2 Msg 1	Station 3 Msg 1
Station 1 Msg 3	Station 2 Msg 2	Station 3 Msg 2
Station 1 Msg 4	Station 2 Msg 3	Station 3 Msg 3
	Station 2 Msg 4	Station 3 Msg 4

Problem: grouping messages

Station 2 Msg 1	Station 3 Msg 1	
Station 1 Msg 2	Station 2 Msg 2	Station 3 Msg 2
Station 1 Msg 3	Station 2 Msg 3	Station 3 Msg 3
Station 1 Msg 4	Station 2 Msg 4	Station 3 Msg 4

Problem: grouping messages

Station 2 Msg 1	Station 3 Msg 1	
Station 1 Msg 2	Station 2 Msg 2	Station 3 Msg 2
Station 1 Msg 3	Station 2 Msg 3	Station 3 Msg 3
Station 1 Msg 4	Station 2 Msg 4	Station 3 Msg 4

Problem: grouping messages

Station 1 Msg 2	Station 2 Msg 2	Station 3 Msg 2
Station 1 Msg 3	Station 2 Msg 3	Station 3 Msg 3
Station 1 Msg 4	Station 2 Msg 4	Station 3 Msg 4

Station 2
Msg 1

Station 3
Msg 1

Meet Greta Grouper

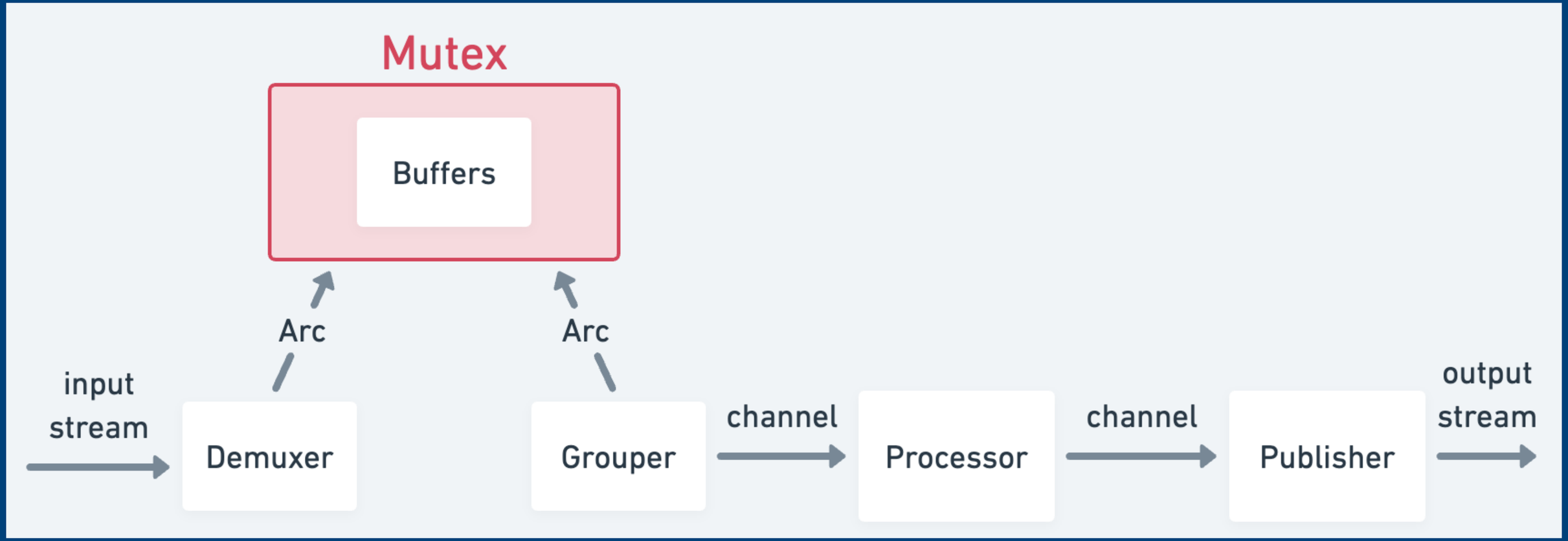
- Pulls messages out of the buffers
- Packages them into a group
- Sends that group off for processing

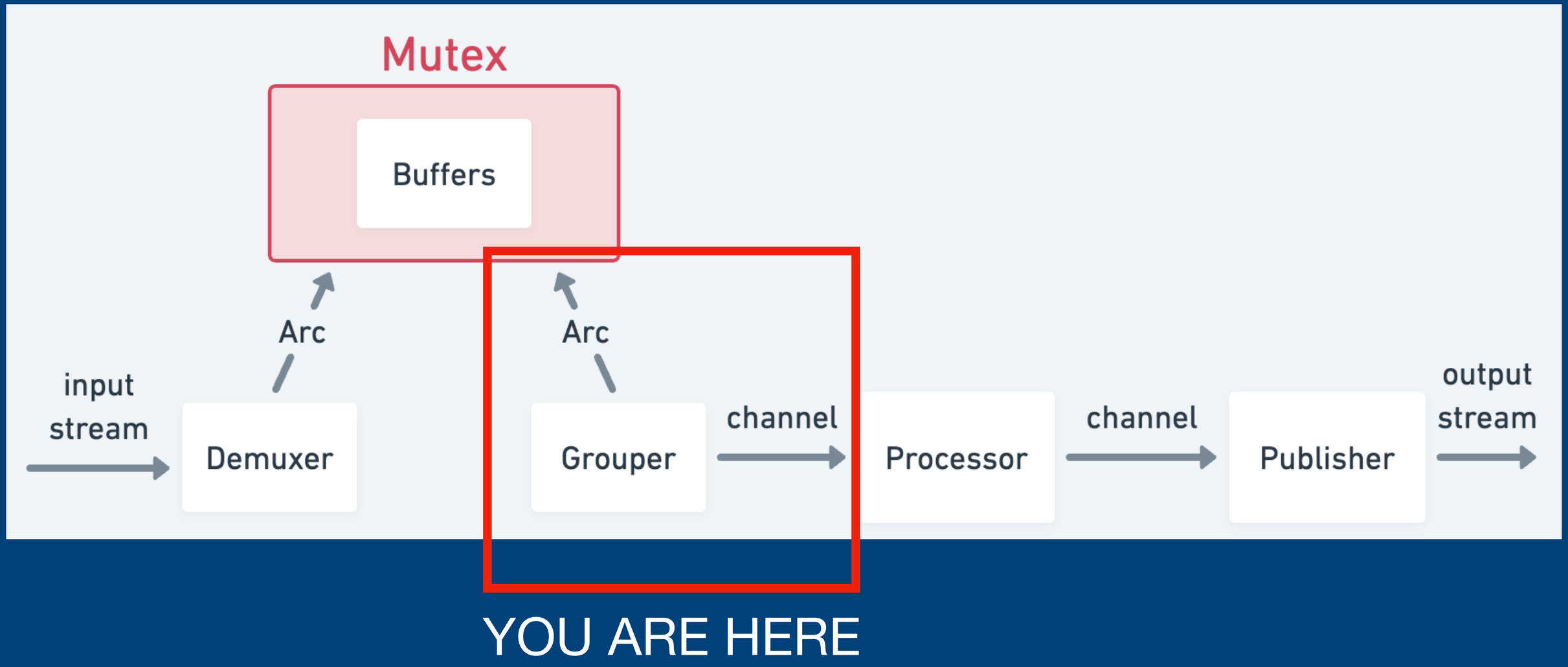
Meet Greta Grouper

- Pulls messages out of the buffers
- Packages them into a group
- Sends that group off for processing

Problem: getting data out of tasks

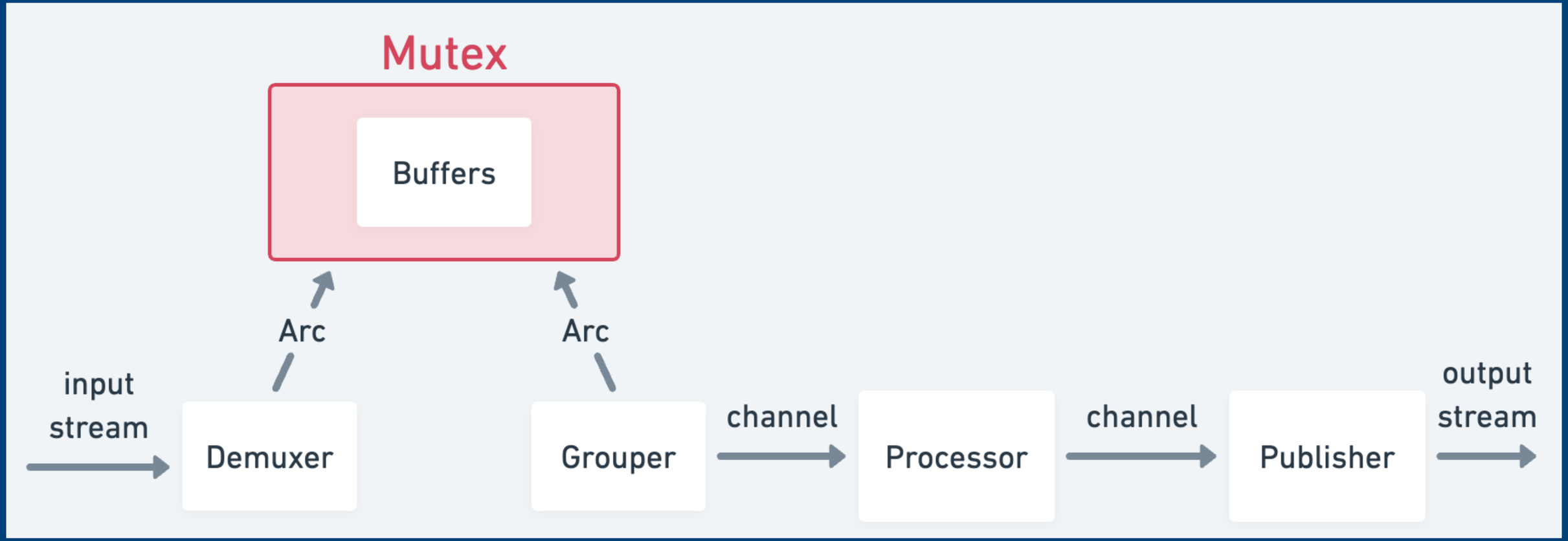
- A task isn't a function, how do you get data out?
- One option: write to an external data structure (e.g. demuxer -> buffers)
- Another option: channels!
  - Basically a computer wormhole
  - `futures::sync::mpsc::channel`

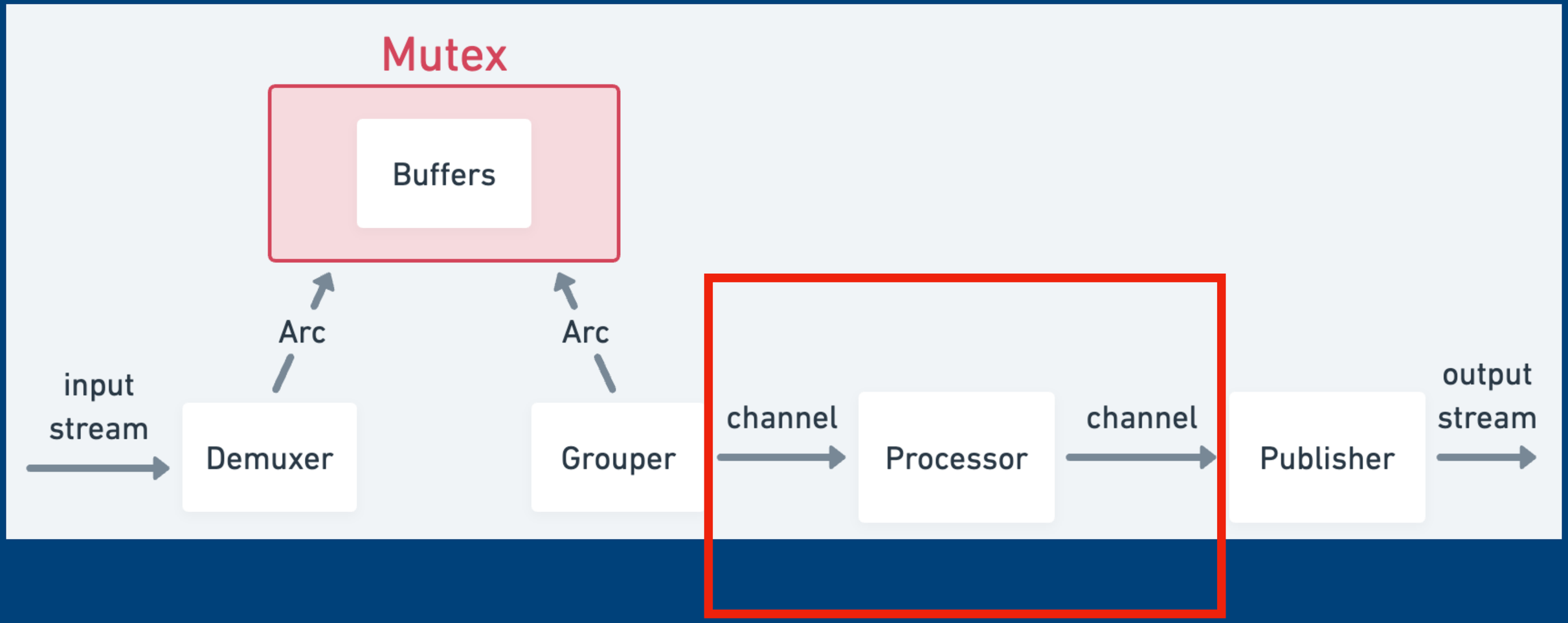

Mutex
Buffers
Arc
Arc
input
stream
Demuxer
Grouper
channel
Processor
channel
Publisher
output
stream


Mutex
Buffers
Arc
Arc
input
stream
Demuxer
Grouper
channel
Processor
channel
Publisher
output
stream
YOU ARE HERE

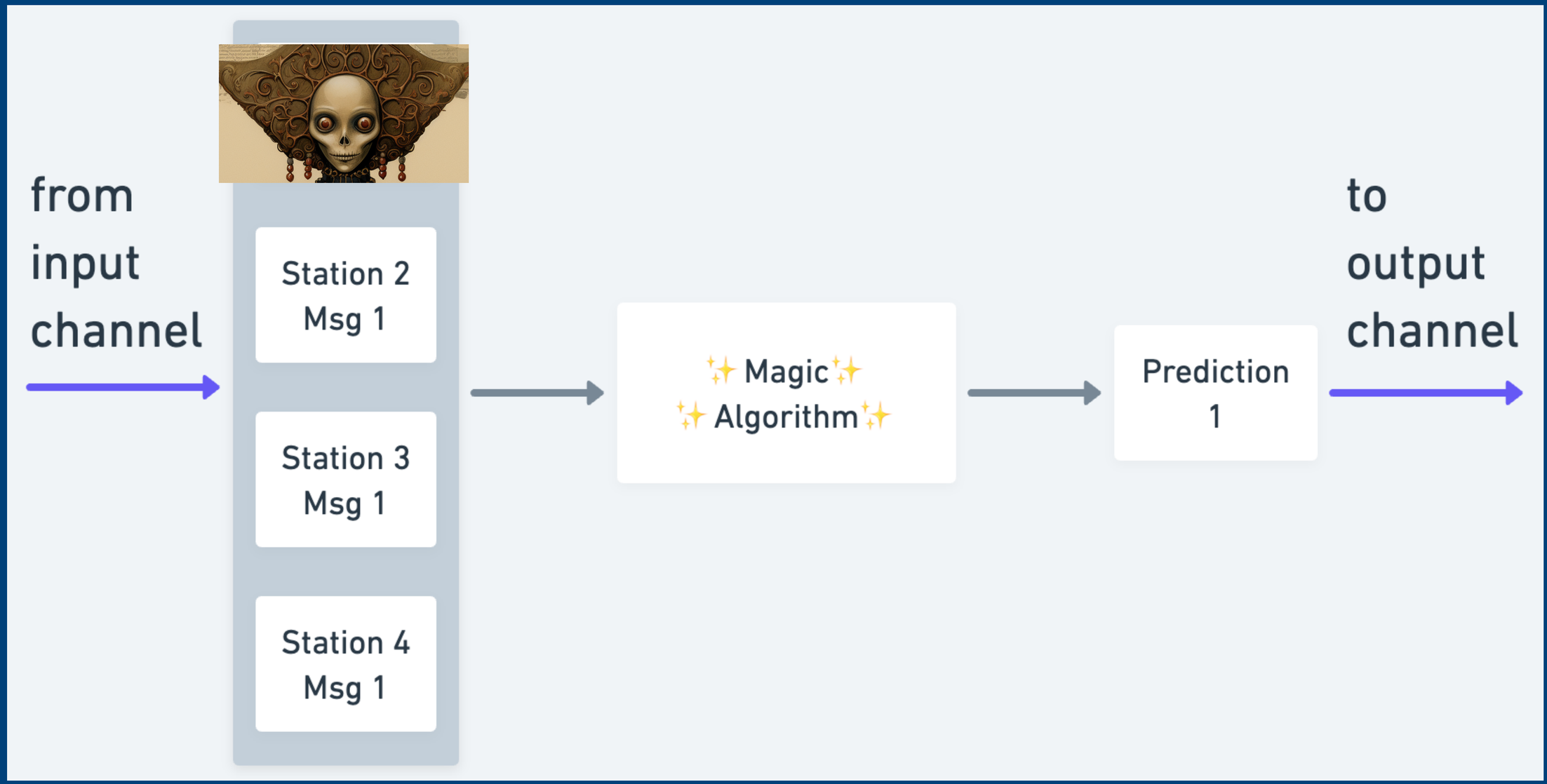
ACT III

Metamorphosis


Mutex
Buffers
Arc
Arc
input
stream
Demuxer
Grouper
channel
Processor
channel
Publisher
output
stream


Mutex
Buffers
Arc
Arc
input
stream
Demuxer
Grouper
channel
Processor
channel
Publisher
output
stream
YOU ARE HERE

Making predictions



Meet The Processor

Meet The Prediction

Meet The Prediction

"A glowing ball of light with several faces"

Meet The Prediction

"A glowing ball of light with several faces"

Meet Padma Publisher

Meet Padma Publisher

FIN

Resources

- Me
  - tinkering.xyz
  - https://hachyderm.io/@zmitchell
  - https://github.com/zmitchell
- Code
  - https://github.com/zmitchell/streaming-weather-app
- Slides
  - https://github.com/zmitchell/talks