



**Aeron**  
*High-Performance  
Open Source  
Message Transport*

**Martin Thompson - @mjpt777**



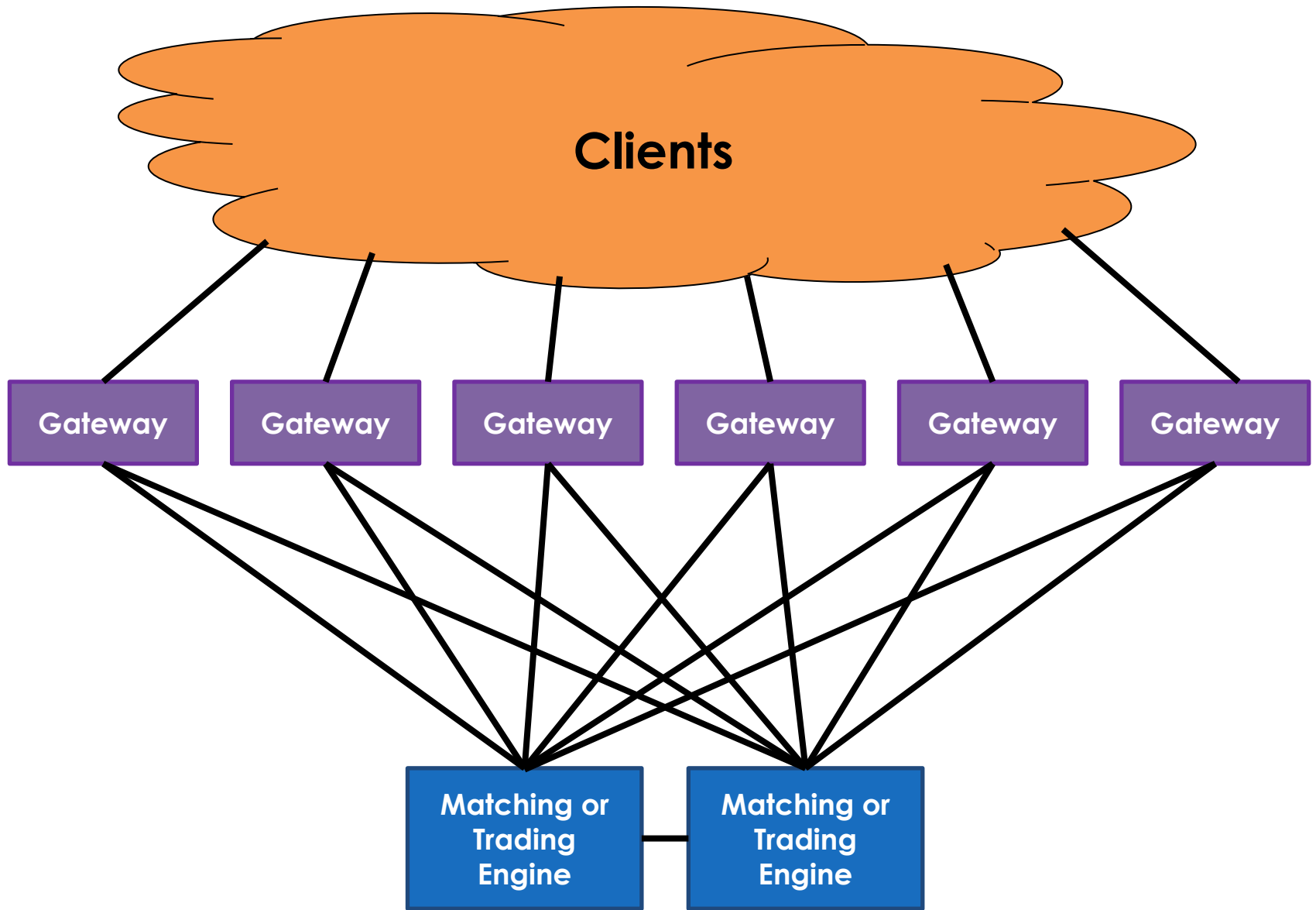
1. Why build another **Product**?
2. What **Features** are really needed?
3. How does one **Design** for this?
4. What did we **Learn** on the way?
5. What's the **Roadmap**?

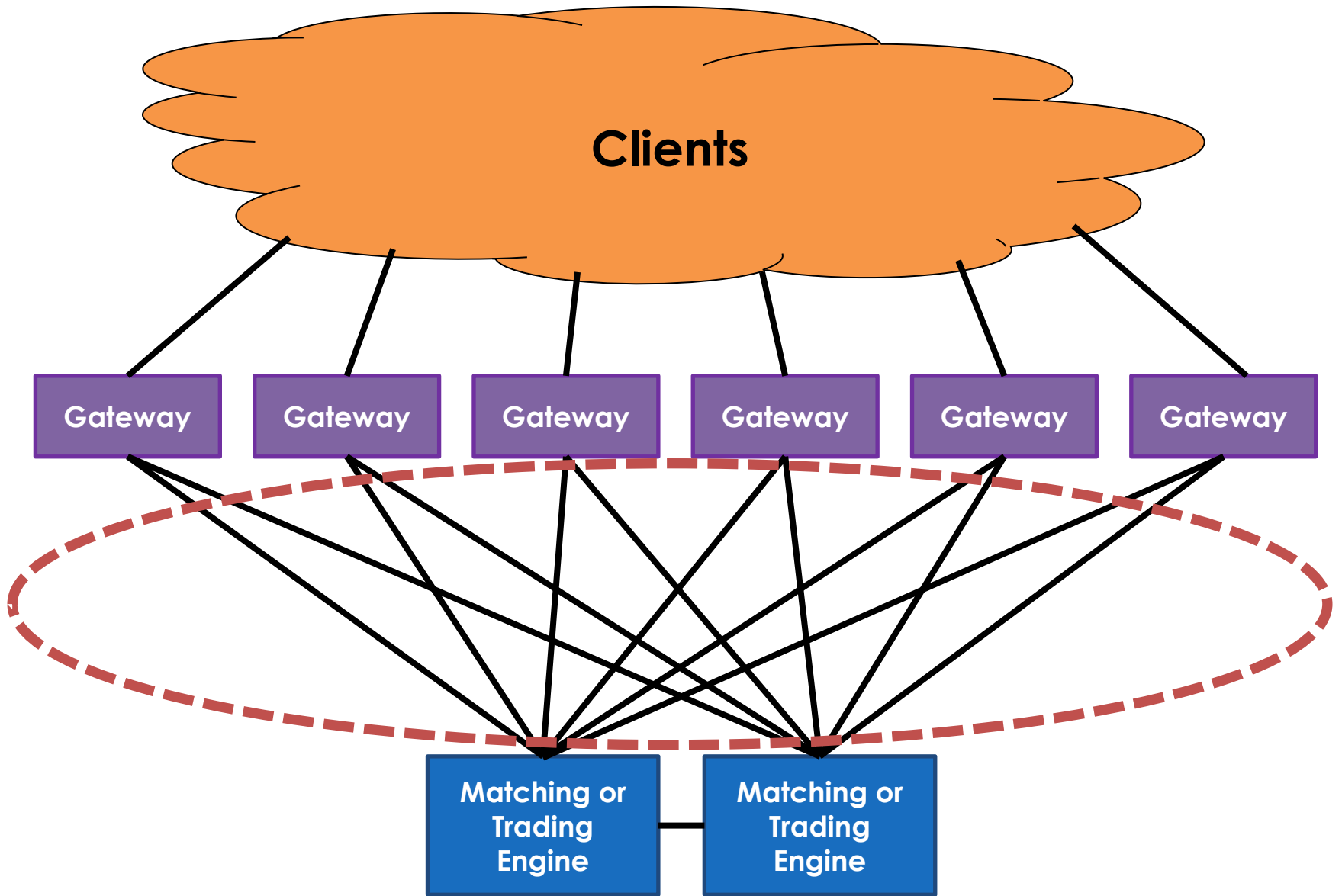
***1. Why build another product?***



***Not Invented Here!***

***There's a story here...***





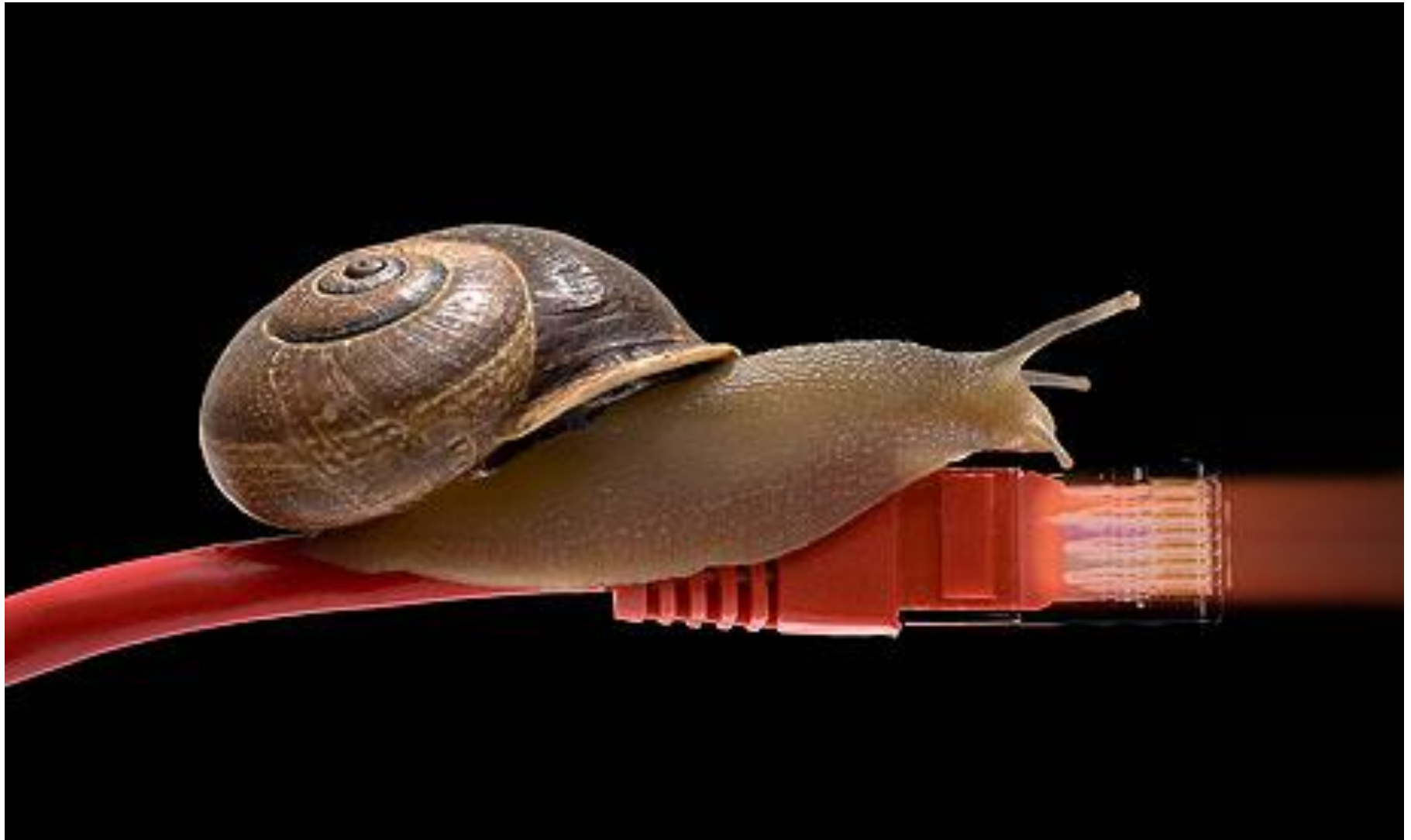


***But many others could benefit***

# Feature Bloat & Complexity



# Not Fast Enough





Low-Latency is key



We are in a new world

***Multi-core, Multi-socket,  
Cloud...***

We are in a new world

***Multi-core, Multi-socket,  
Cloud...***

***UDP, IPC, InfiniBand,  
RDMA, PCI-e***

***Aeron is trying a new approach***



# The Team

**Todd Montgomery**



**Richard Warburton**



**Martin Thompson**

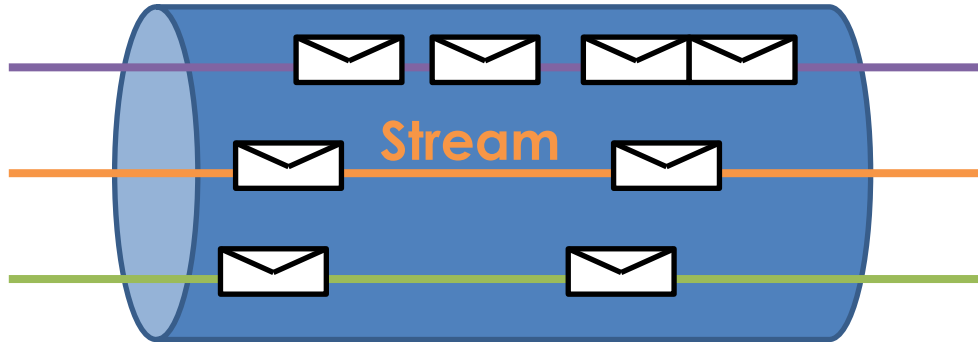
***2. What features  
are really needed?***

# Messaging

**Publishers**



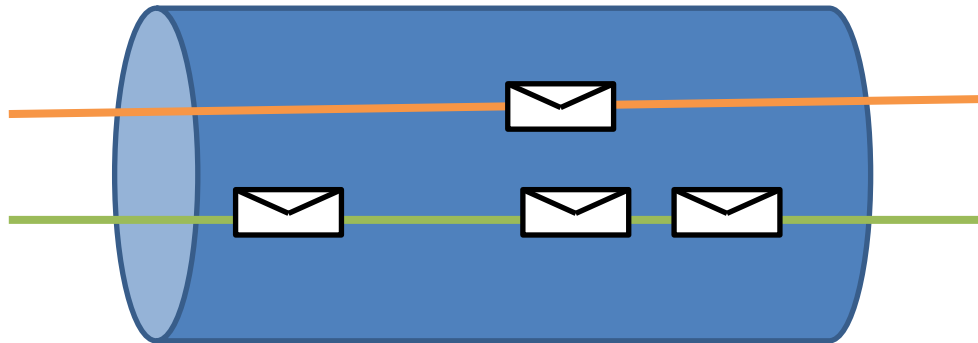
**Channel**



**Subscribers**



**Channel**



***A library, **not a framework**, on  
which other abstractions and  
applications can be built***

# ***Composable Design***

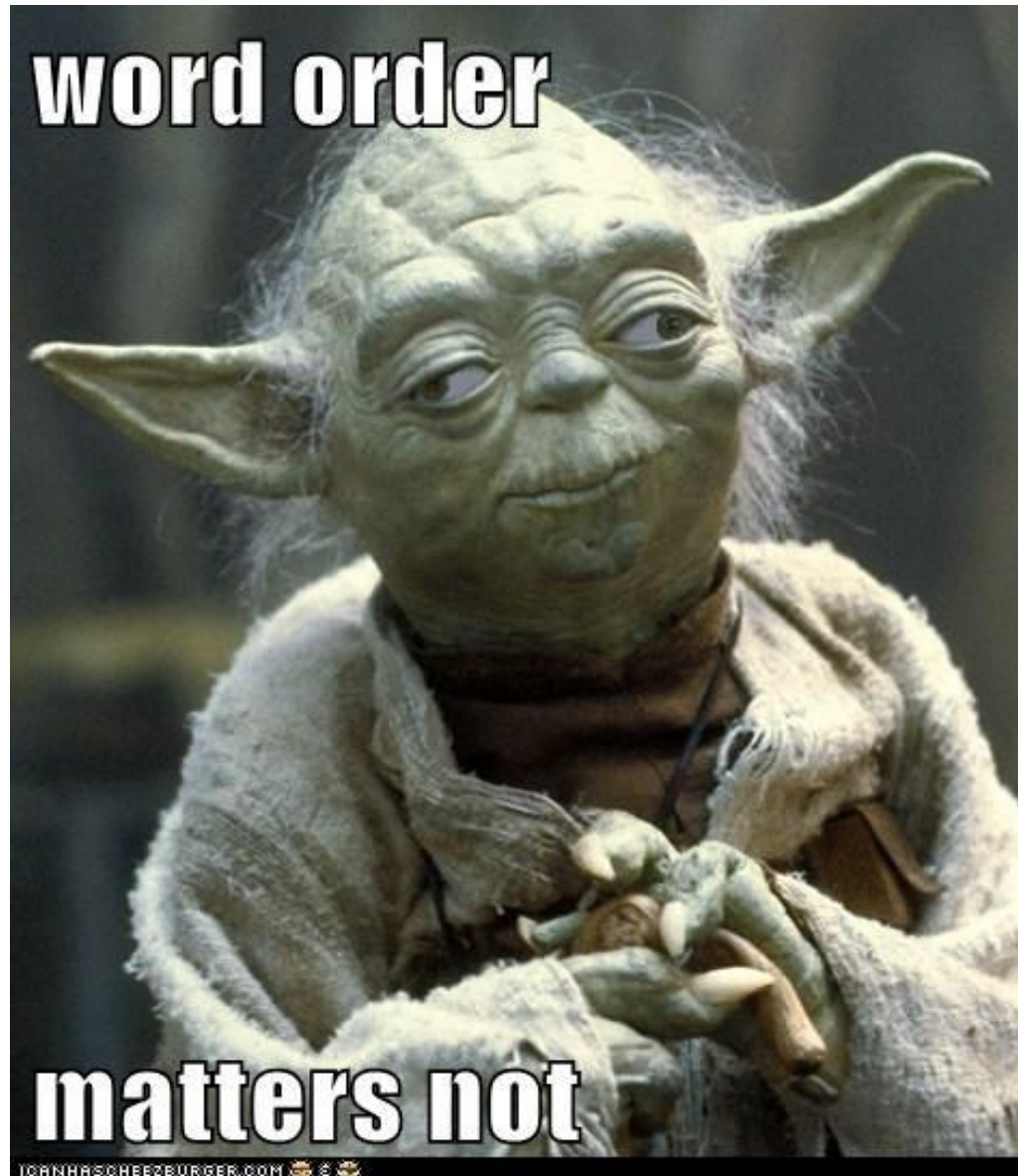
# ***OSI layer 4 Transport for message oriented streams***

## OSI Layer 4 (Transport) Services

- 1. Connection Oriented Communication**
- 2. Reliability**
- 3. Flow Control**
- 4. Congestion Avoidance/Control**
- 5. Multiplexing**



# Connection Oriented Communication



# Reliability



# Flow Control

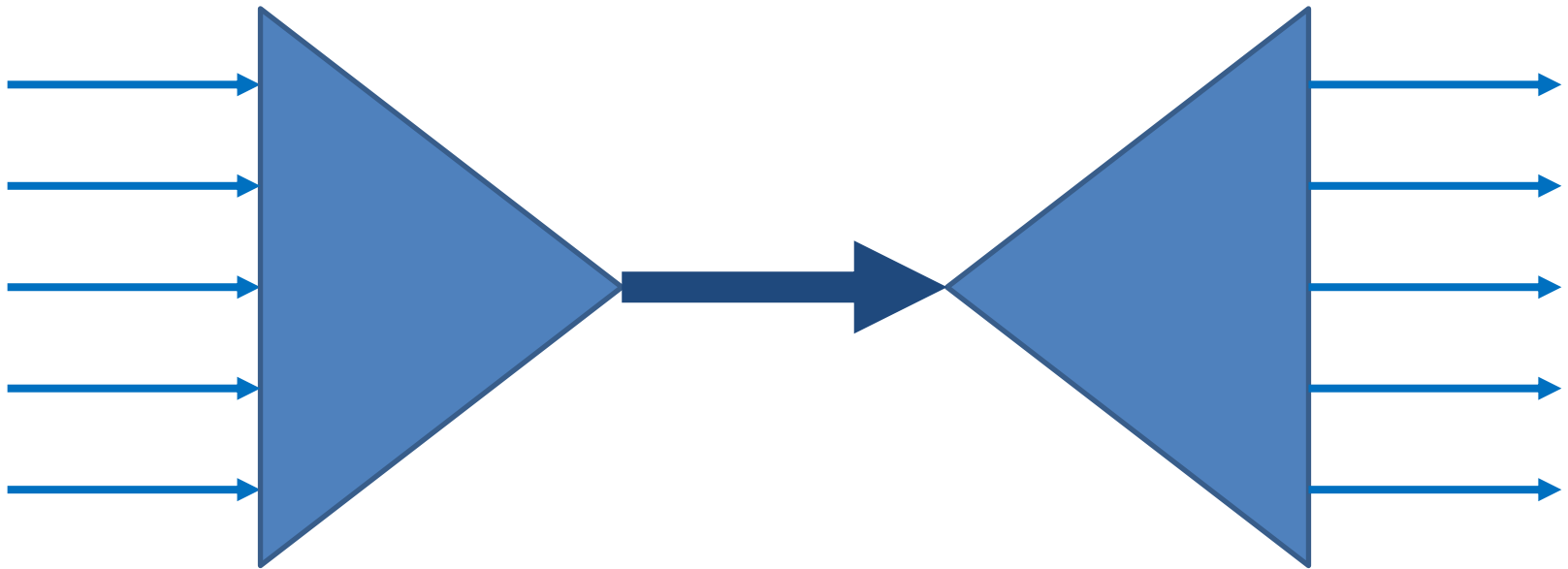




# Congestion Avoidance/Control



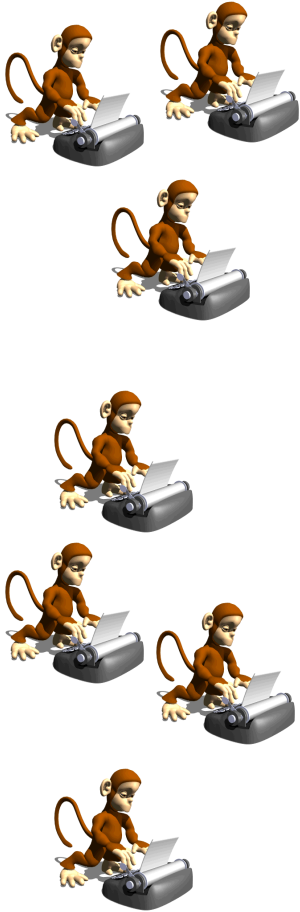
# Multiplexing



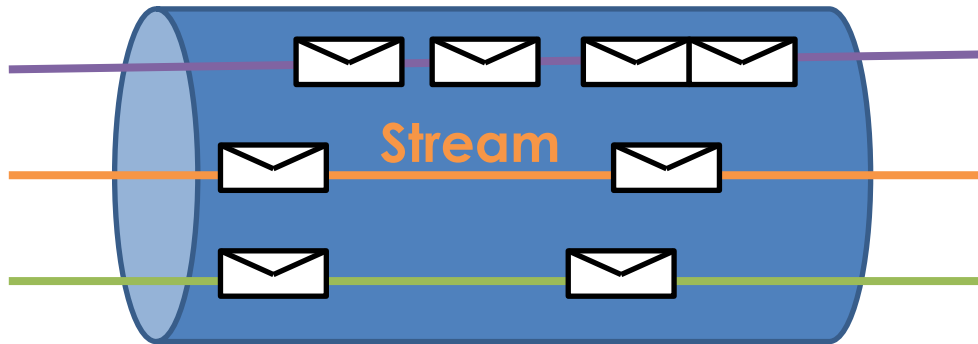
***Multi-Everything World!***

# Multi-Everything World

## Publishers



## Channel



## Subscribers





***Endpoints that scale***

**3. *How does one  
design for this?***

# Design Principles

1. **Garbage free in steady state running**
2. **Smart Batching in the message path**
3. **Wait-free algos in the message path**
4. **Non-blocking IO in the message path**
5. **No exceptional cases in message path**
6. **Apply the *Single Writer Principle***
7. **Prefer unshared state**
8. **Avoid unnecessary data copies**

**It's all about 3 things**

**It's all about 3 things**

# **1. System Architecture**

**It's all about 3 things**

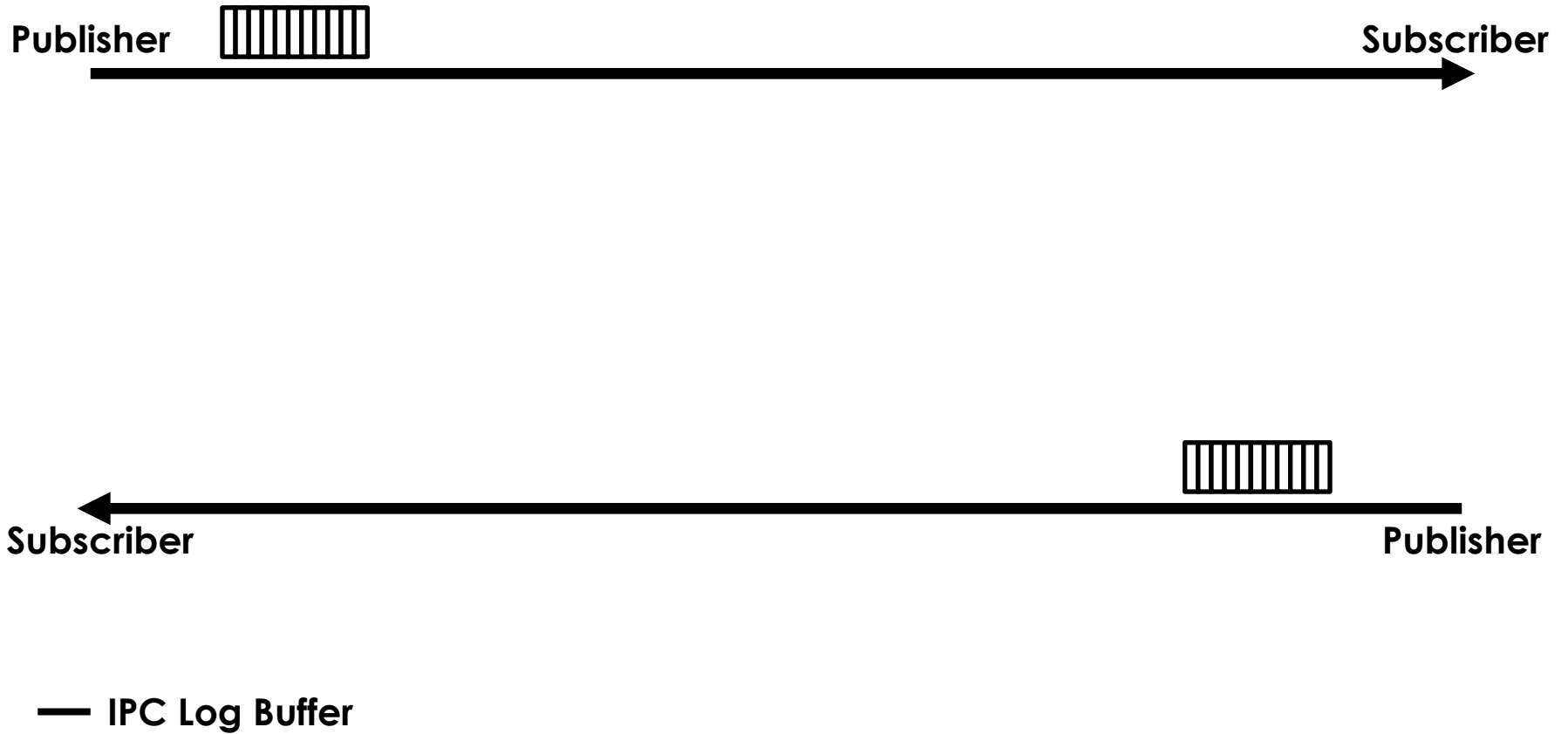
- 1. System Architecture**
- 2. Data Structures**

## **It's all about 3 things**

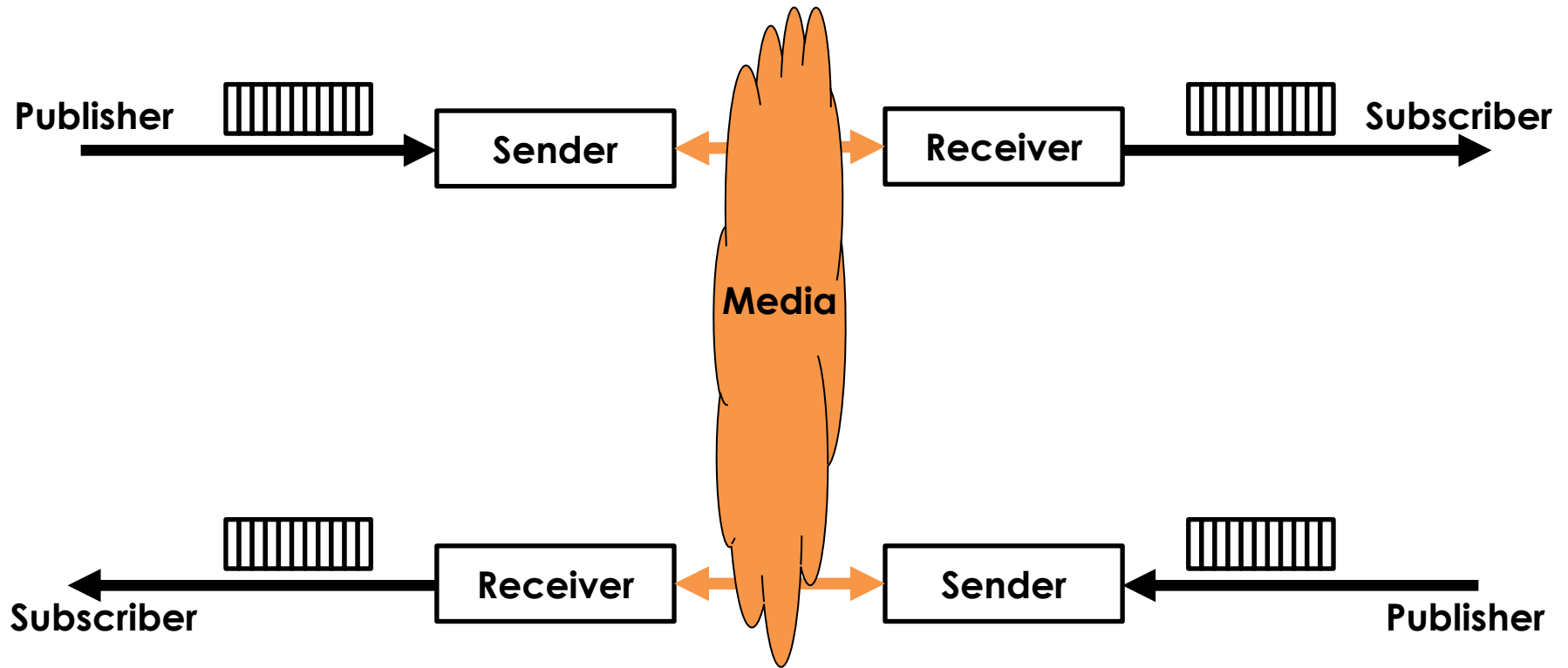
- 1. System Architecture**
- 2. Data Structures**
- 3. Protocols of Interaction**



# Architecture



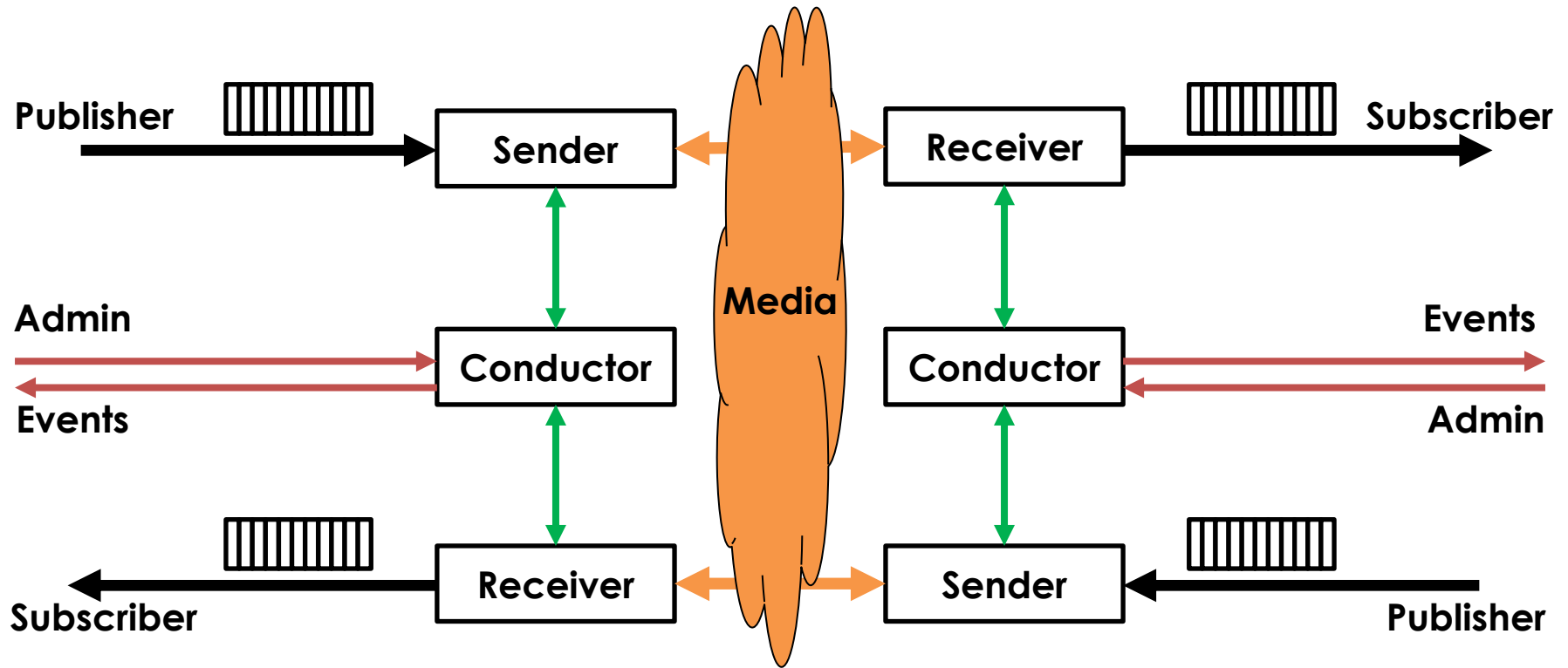
# Architecture



— IPC Log Buffer

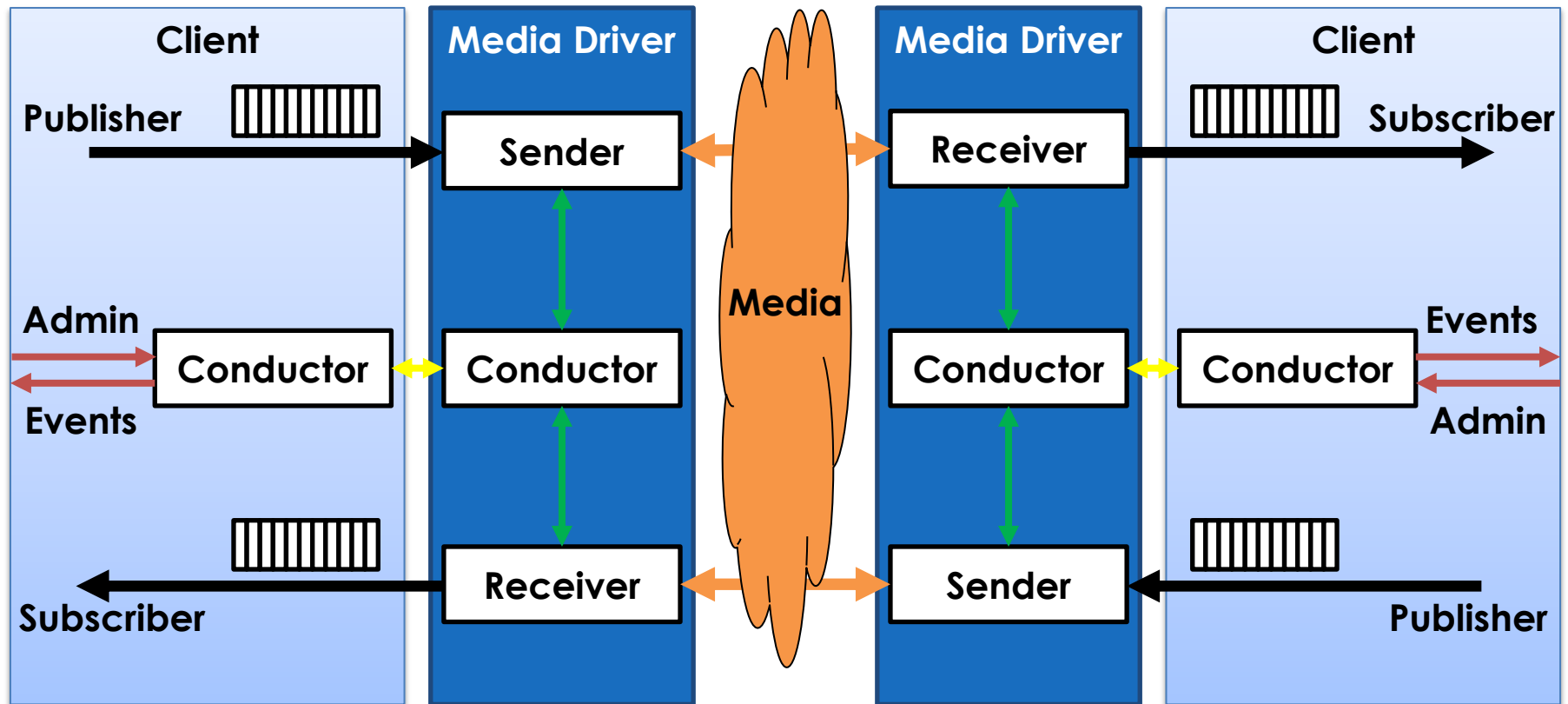
— Media (UDP, InfiniBand, PCI-e 3.0)

# Architecture



- IPC Log Buffer
- Media (UDP, InfiniBand, PCI-e 3.0)
- Function/Method Call
- Volatile Fields & Queues

# Architecture



- IPC Log Buffer
- Media (UDP, InfiniBand, PCI-e 3.0)
- Function/Method Call
- Volatile Fields & Queues
- IPC Ring/Broadcast Buffer

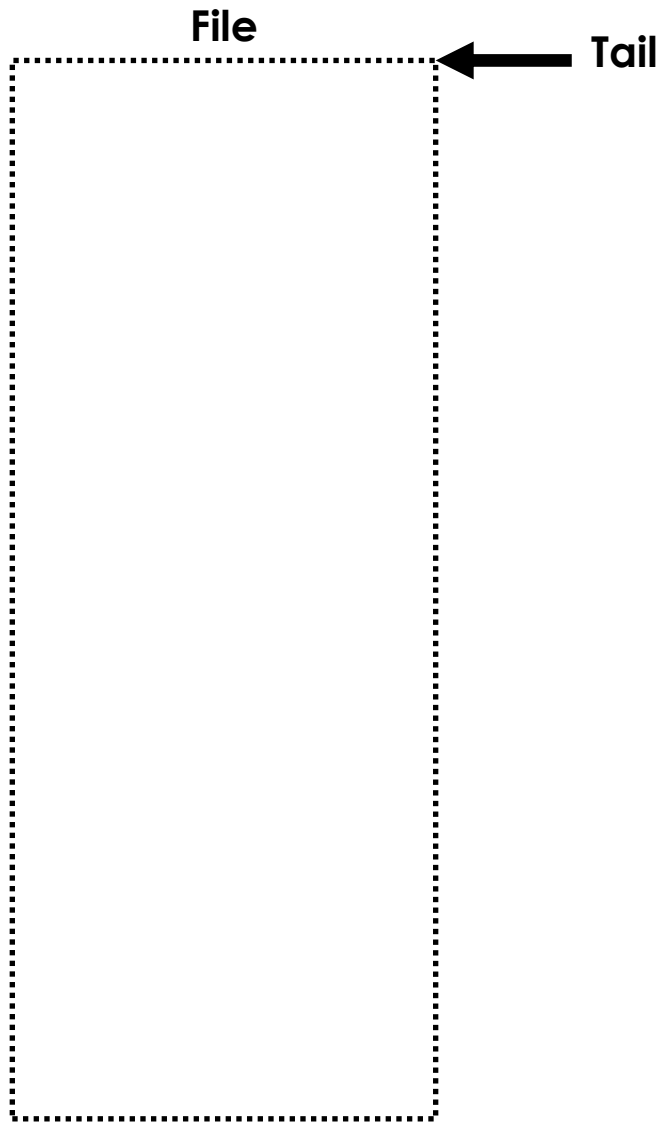
# Data Structures

- **Maps**
- **IPC Ring Buffers**
- **IPC Broadcast Buffers**
- **ITC Queues**
- **Dynamic Arrays**
- **Log Buffers**

What does Aeron do?

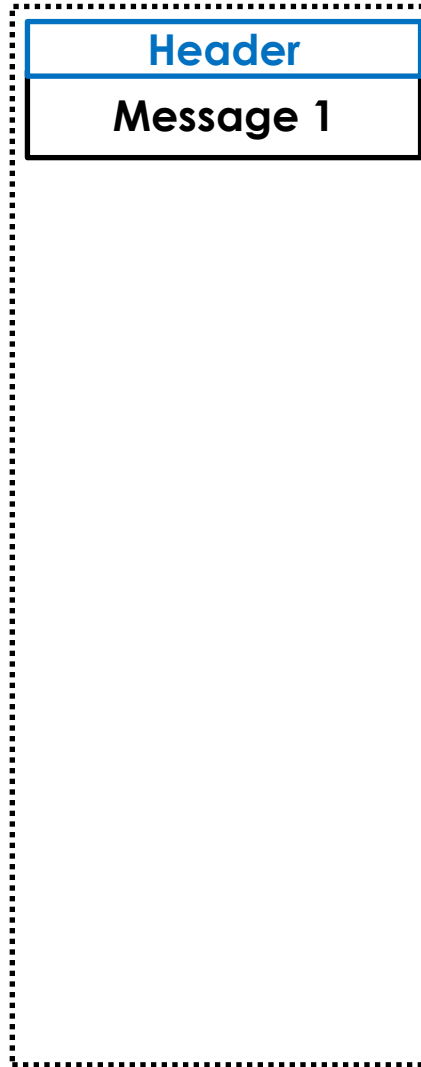
**Creates a**  
***replicated persistent log***  
***of messages***

***How would you design a log?***





File

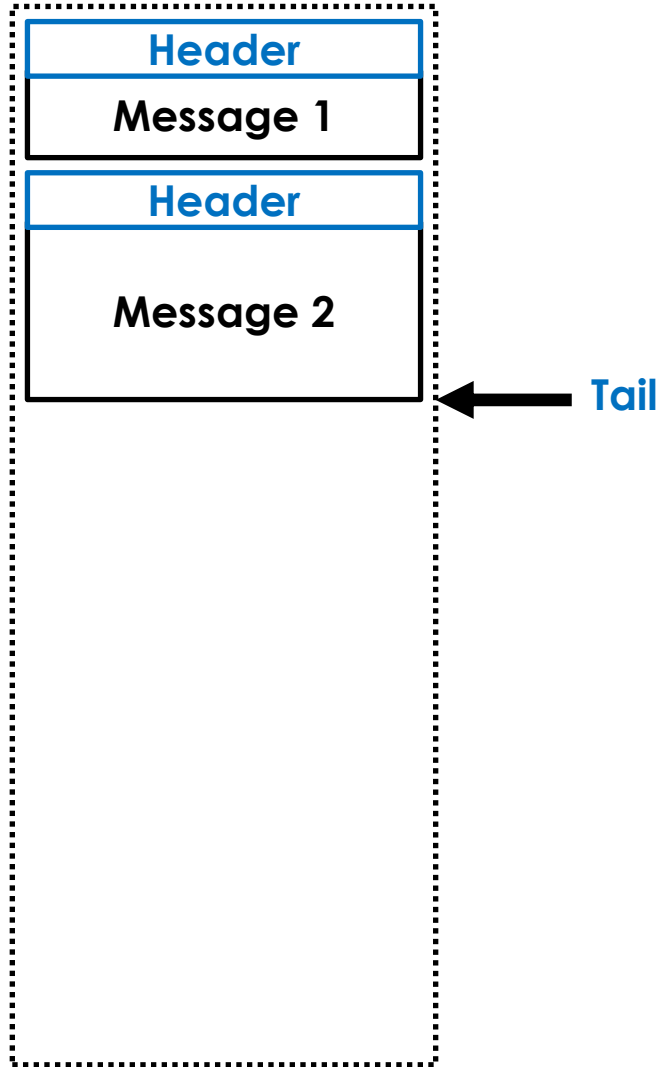


Header

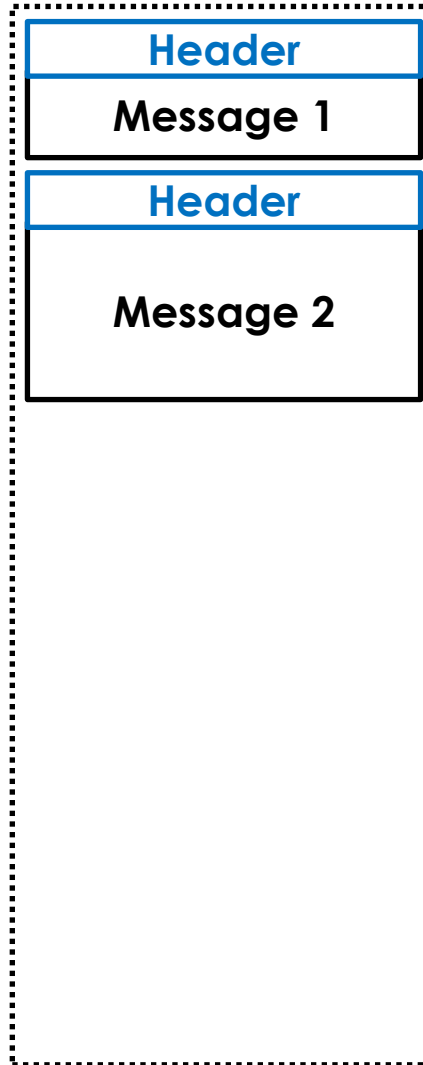
Message 1

Tail

# File

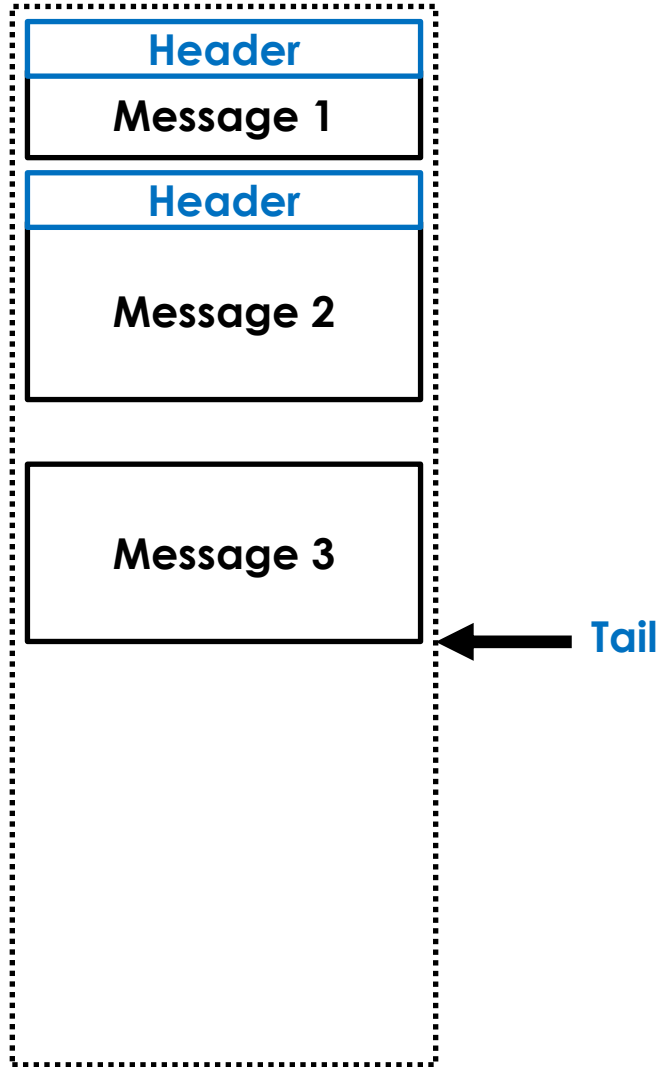


# File

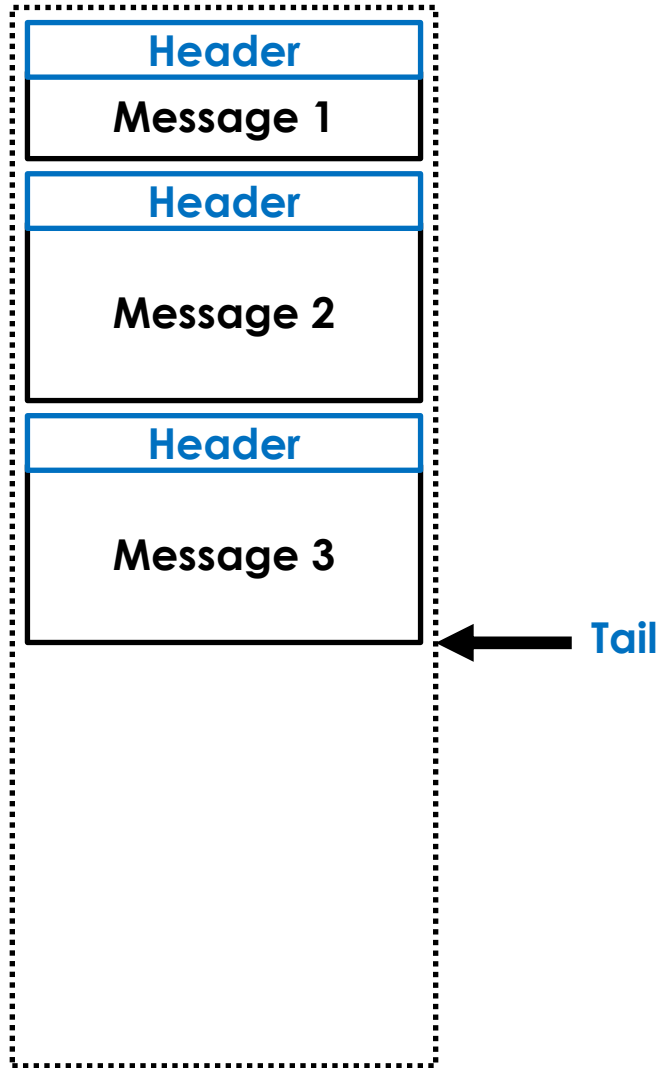


← Tail

# File



## File



***Persistent data structures can  
be safe to read without locks***

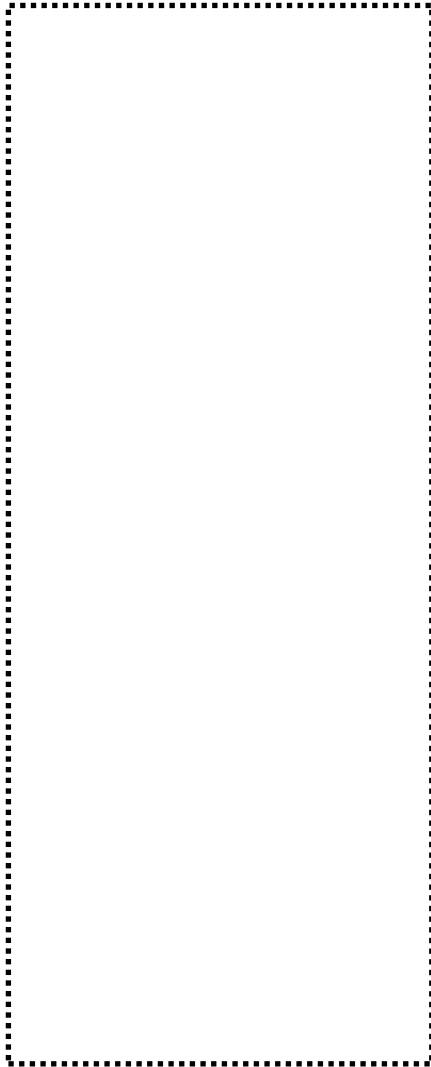
***One big file that  
goes on forever?***

***No!!!***

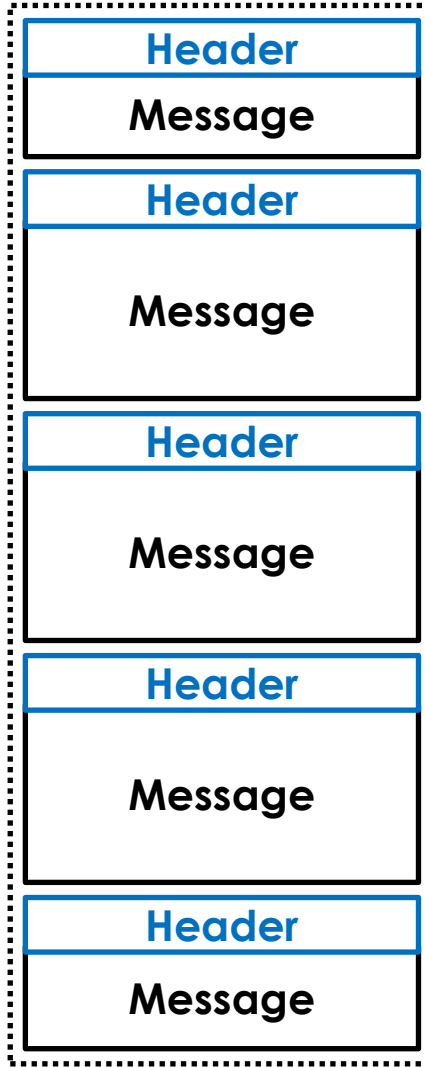
***Page faults, page cache churn,  
VM pressure, ...***



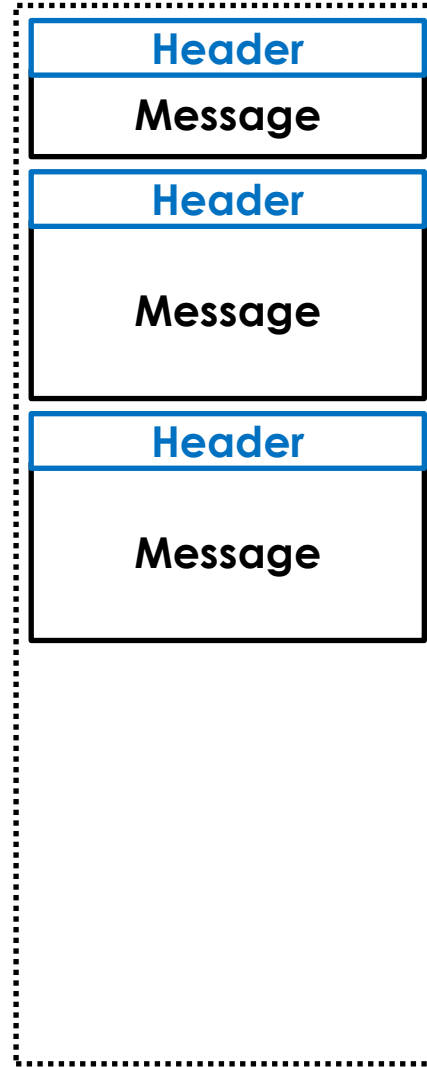
Clean



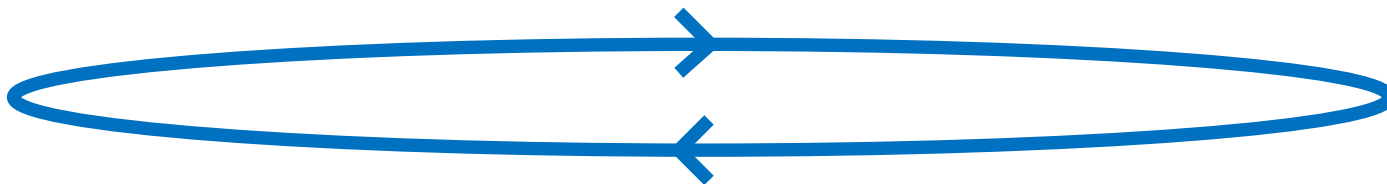
Dirty



Active

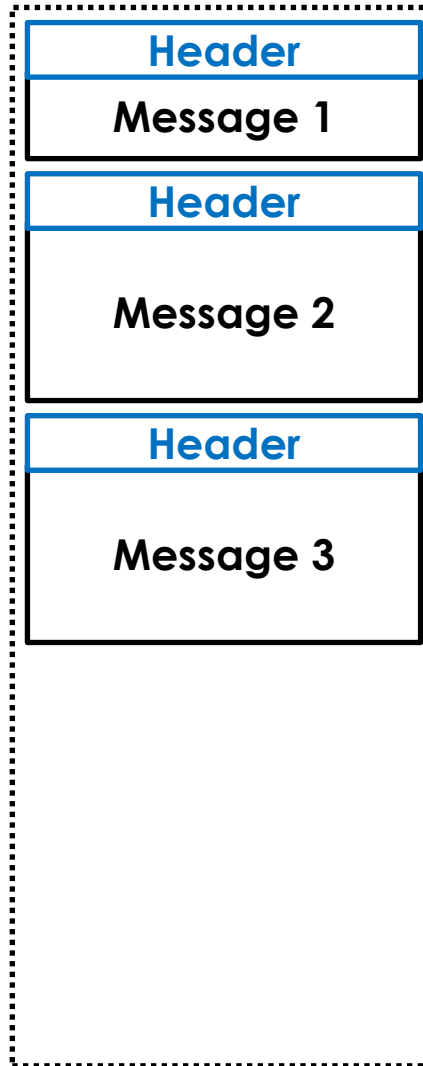


Tail



***How do we stay “wait-free”?***

## File

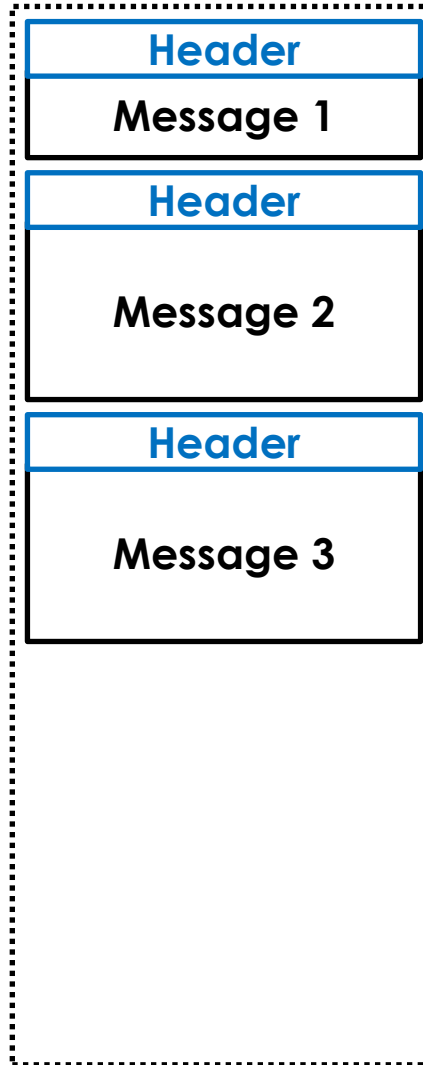


Tail

Message X

Message Y

## File

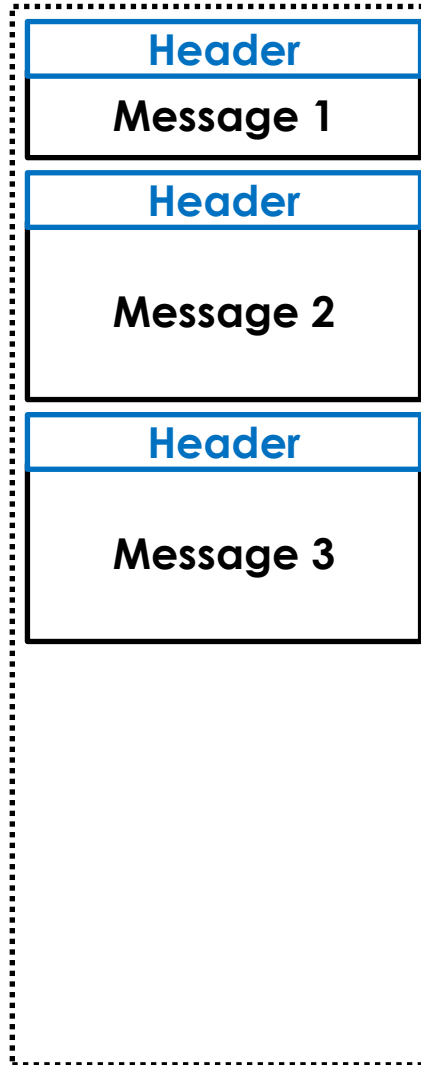


Message X

Message Y

← Tail

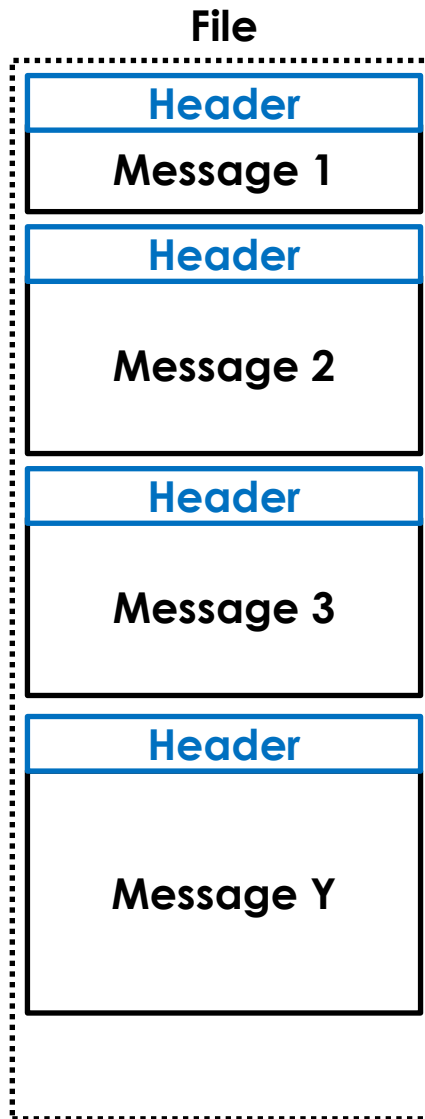
File



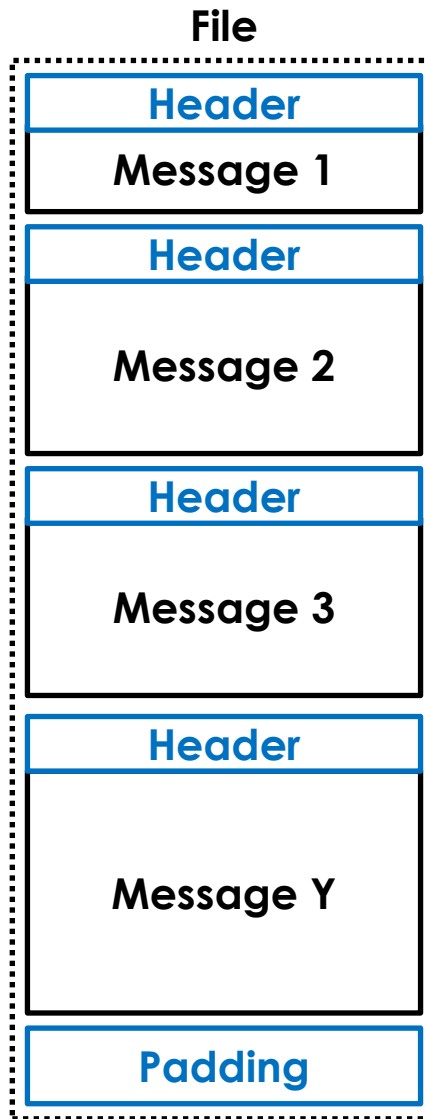
Message X

Message Y

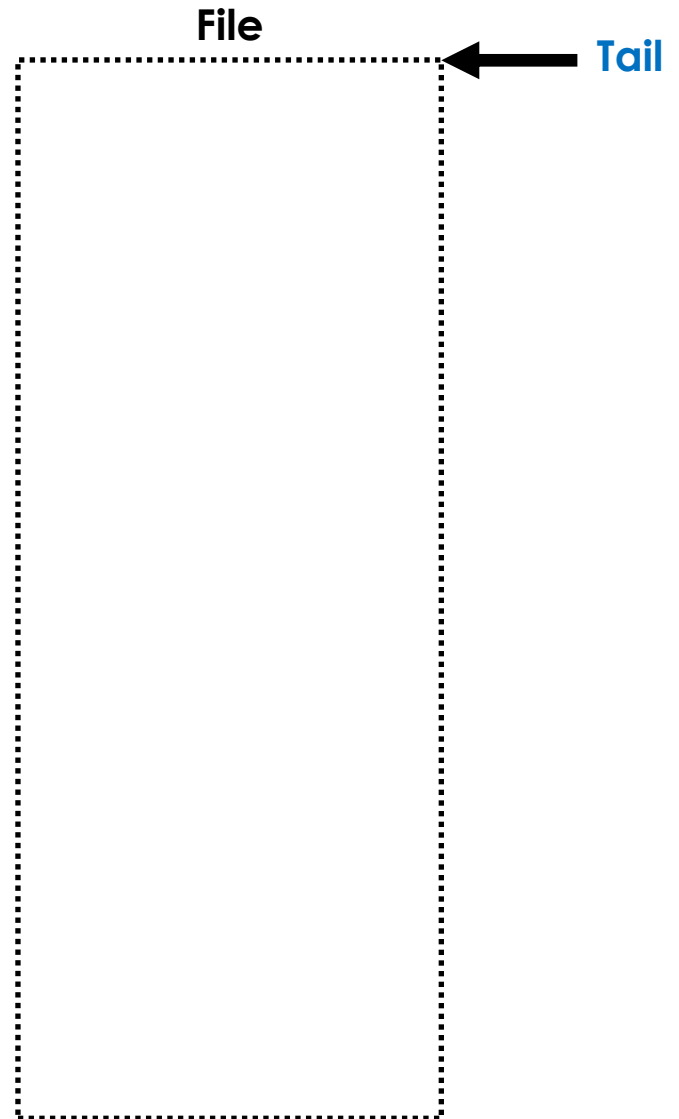
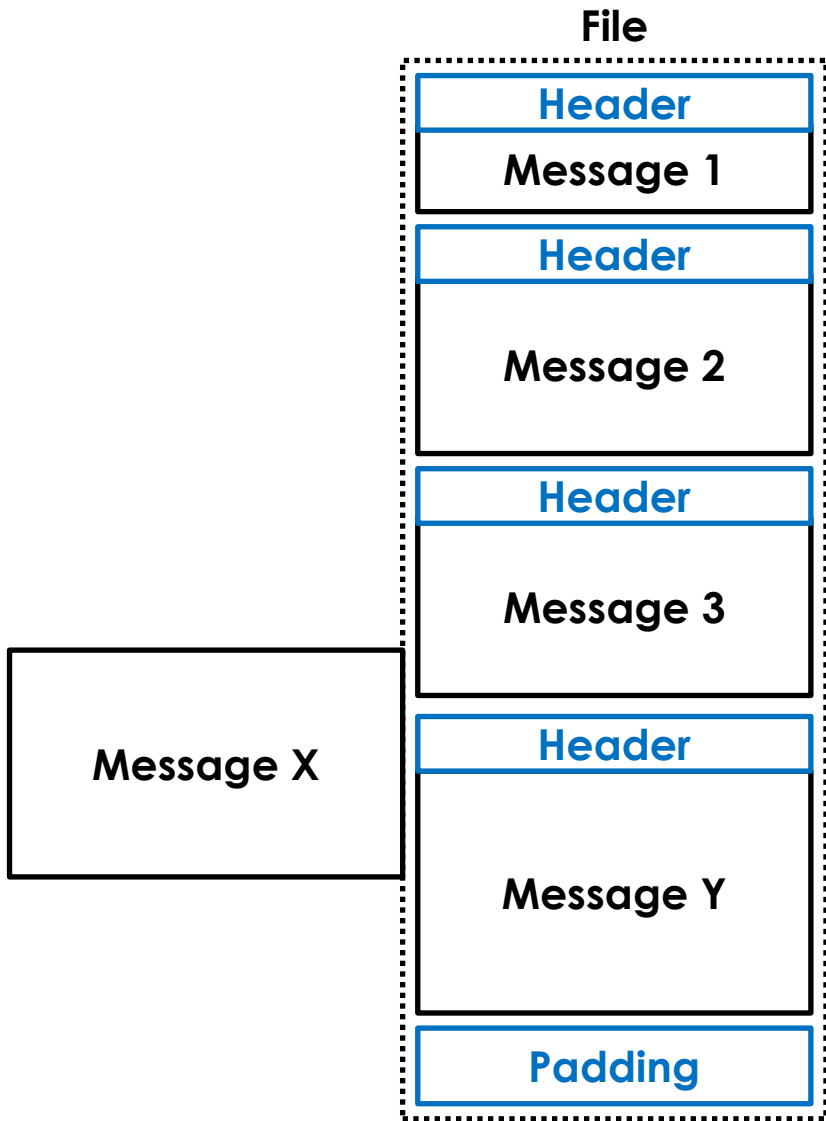
← Tail



← Tail

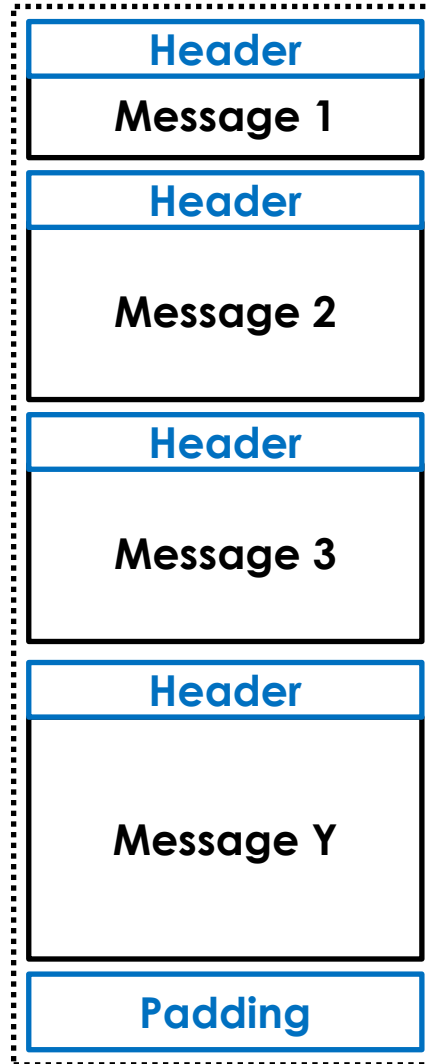


← Tail

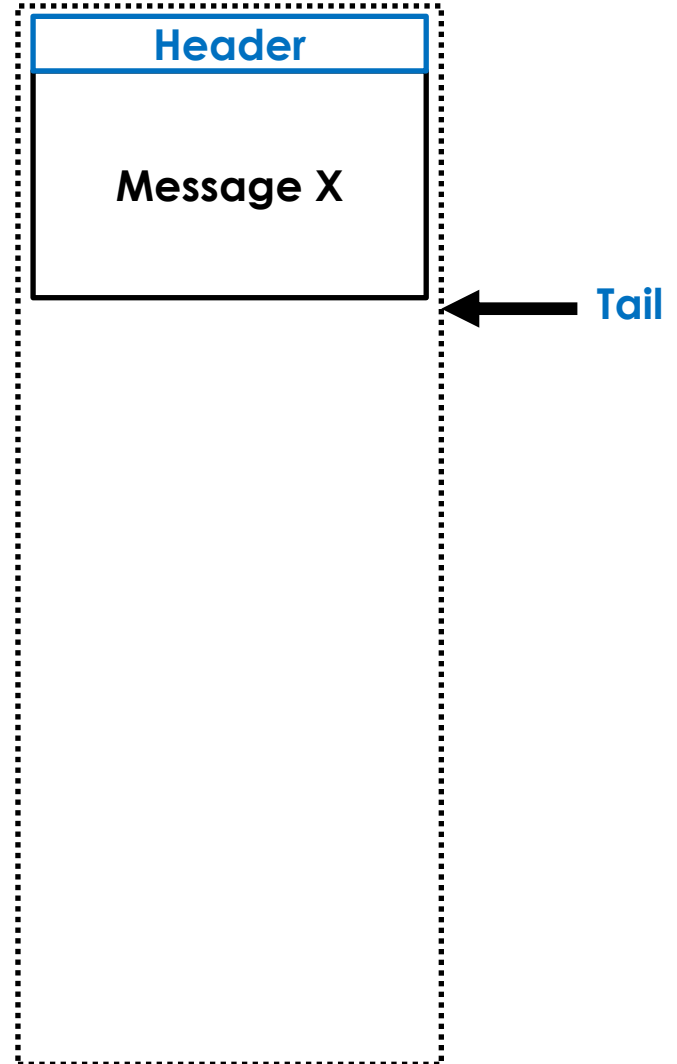




File

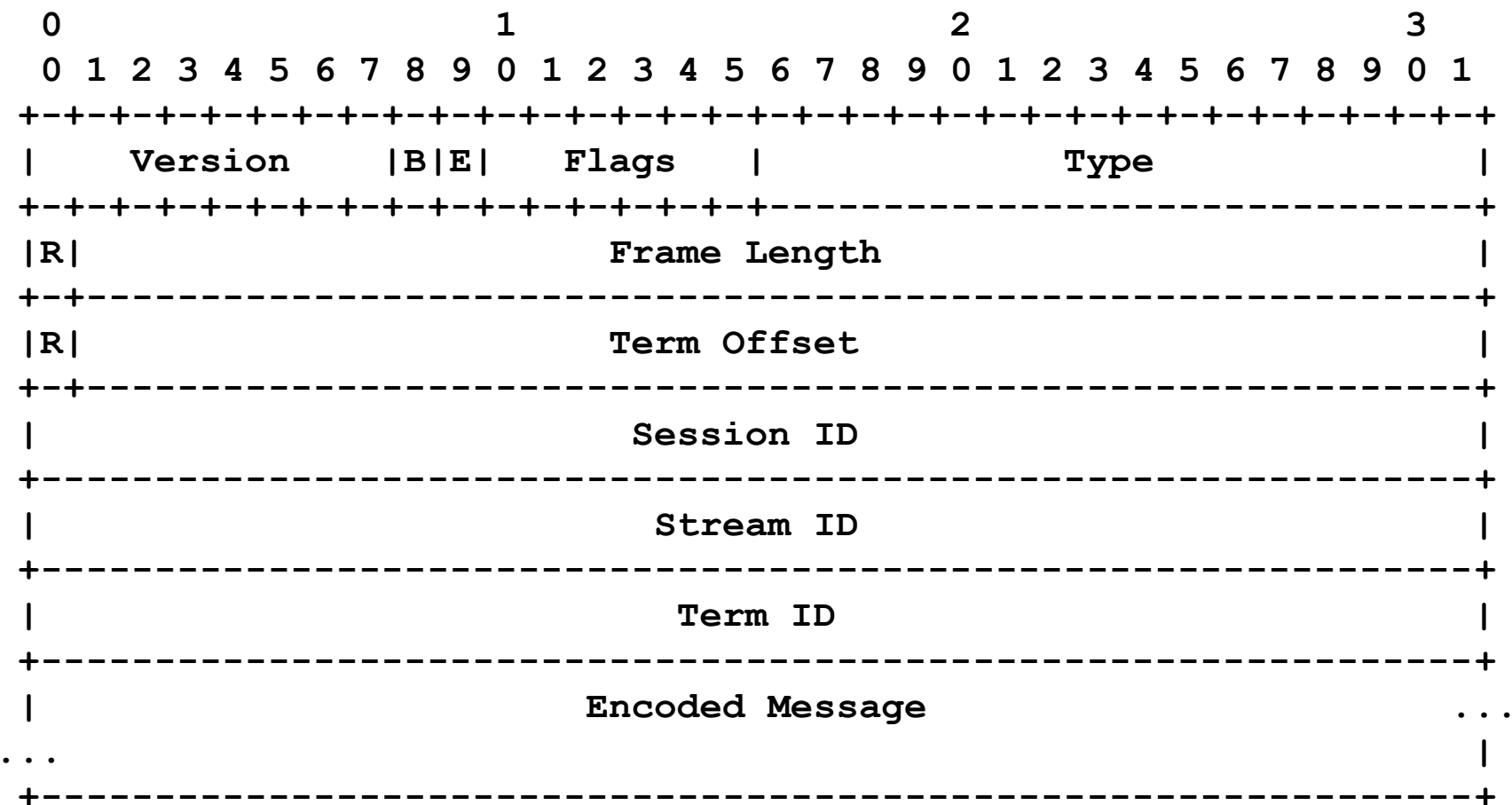


File



***What's in a header?***

# Data Message Header



***Unique identification of a byte  
within each stream across time  
(streamId, sessionId,  
termId, termOffset)***

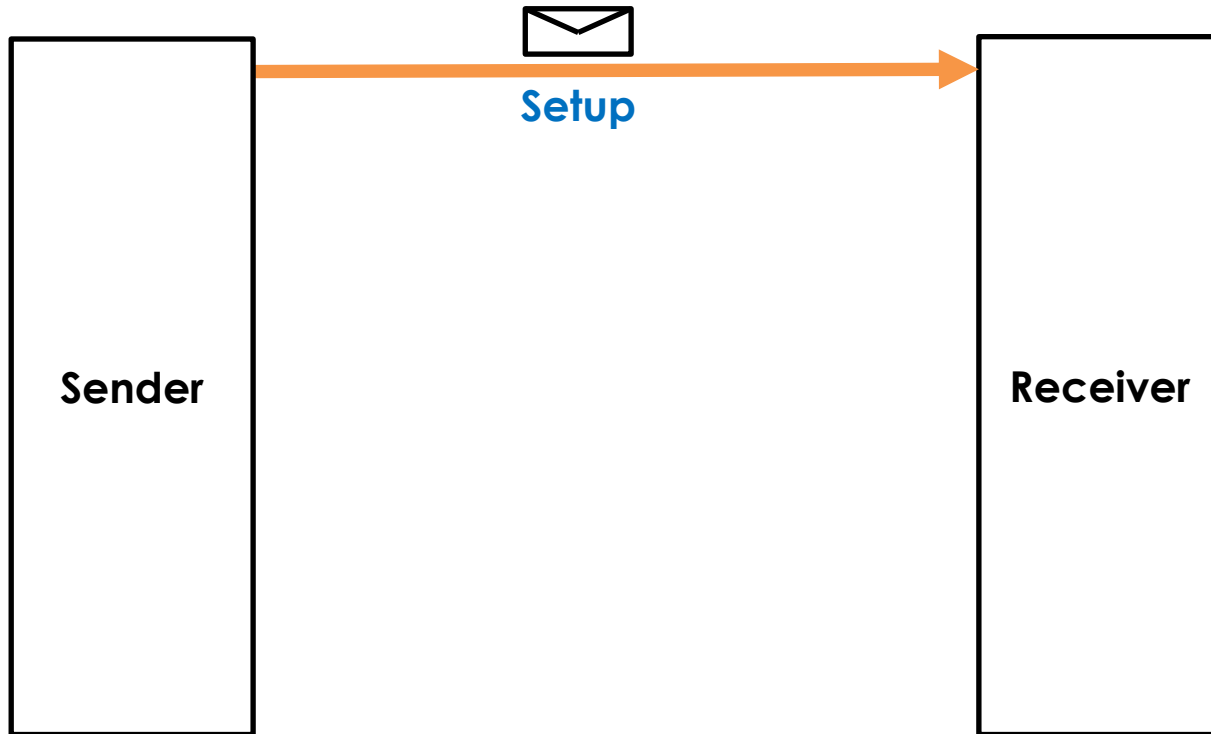
***How do we replicate a log?***

***We need a Protocol of  
messages***

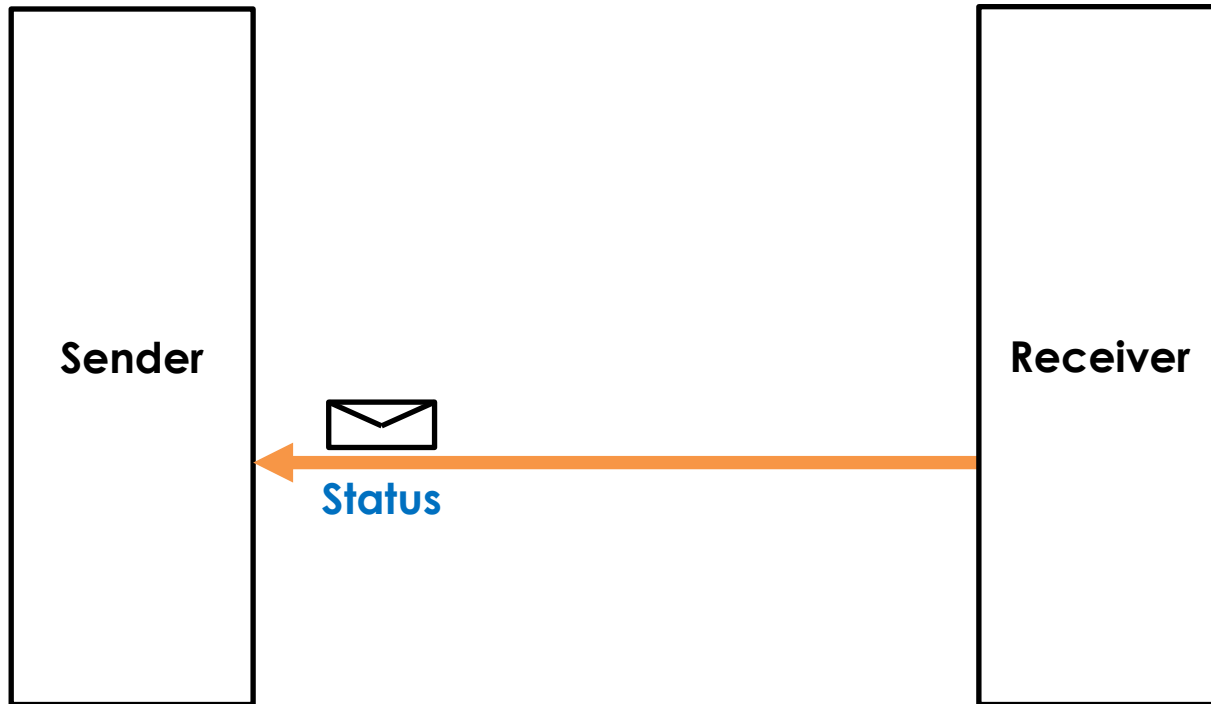


**Sender**

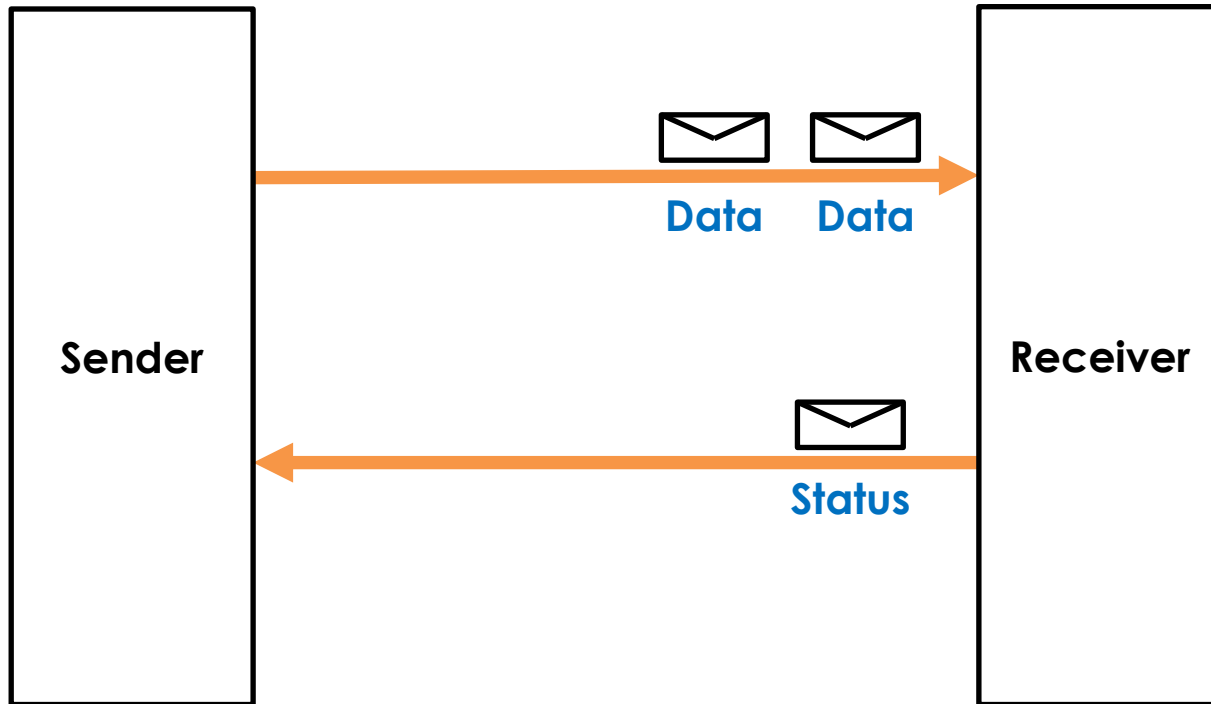
**Receiver**

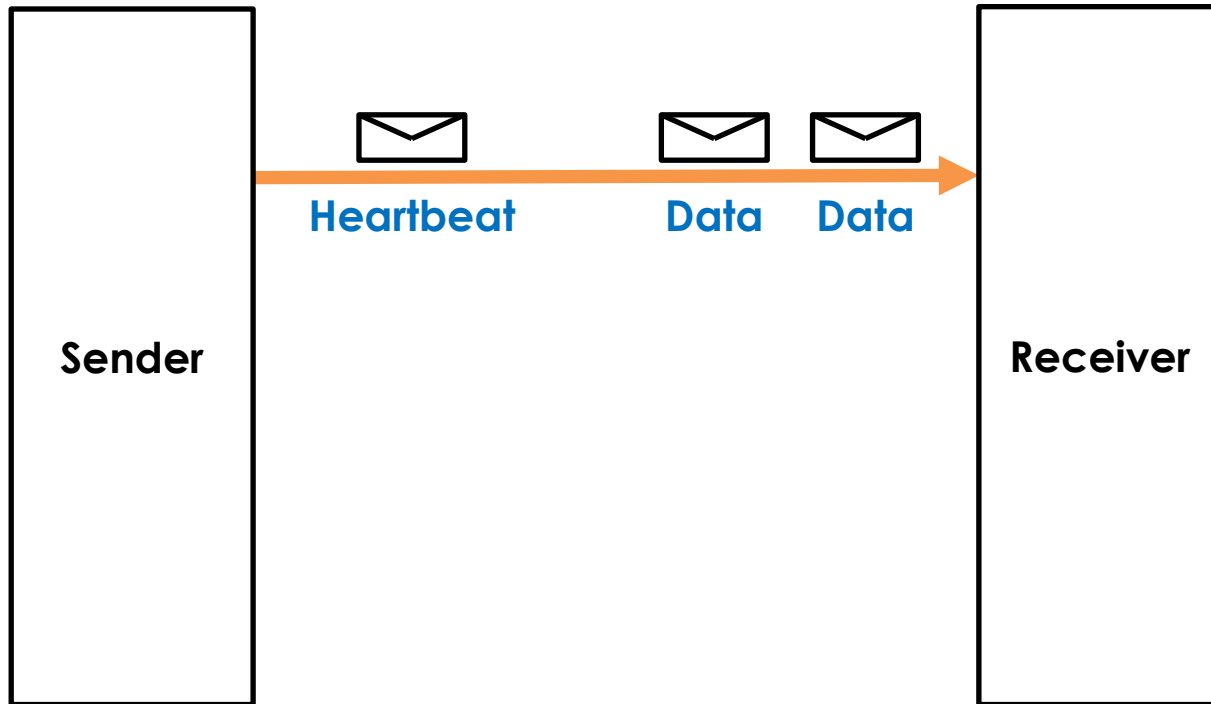


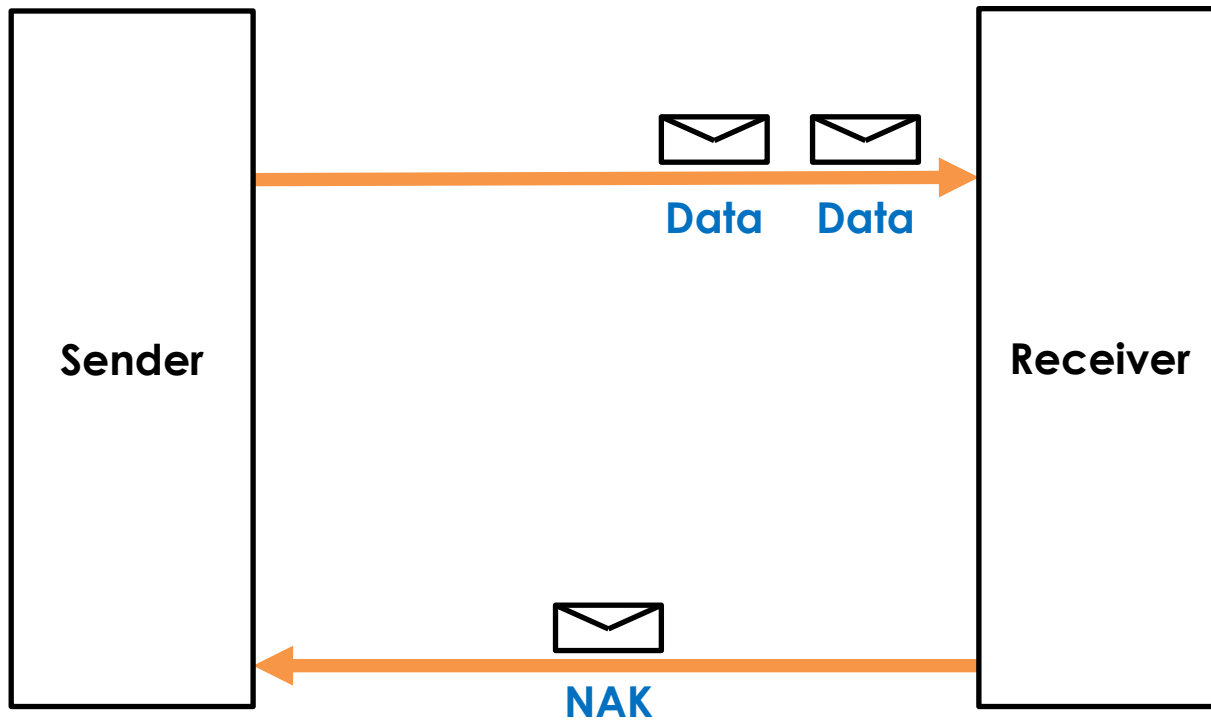


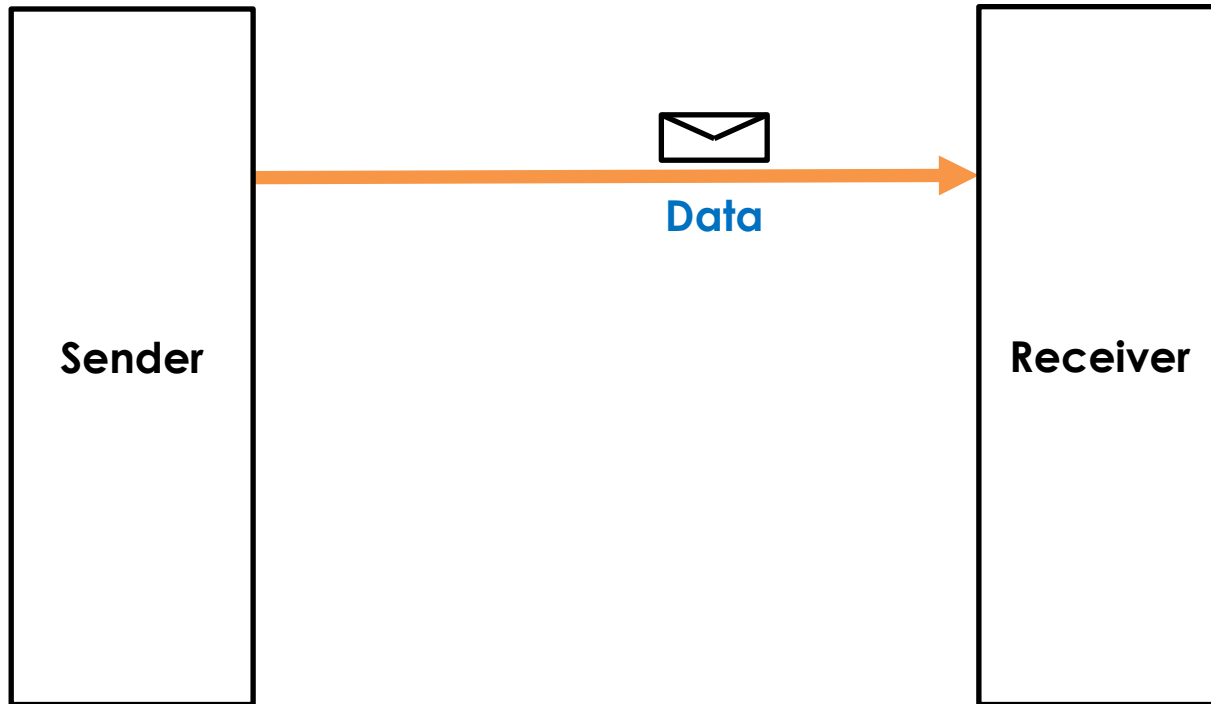




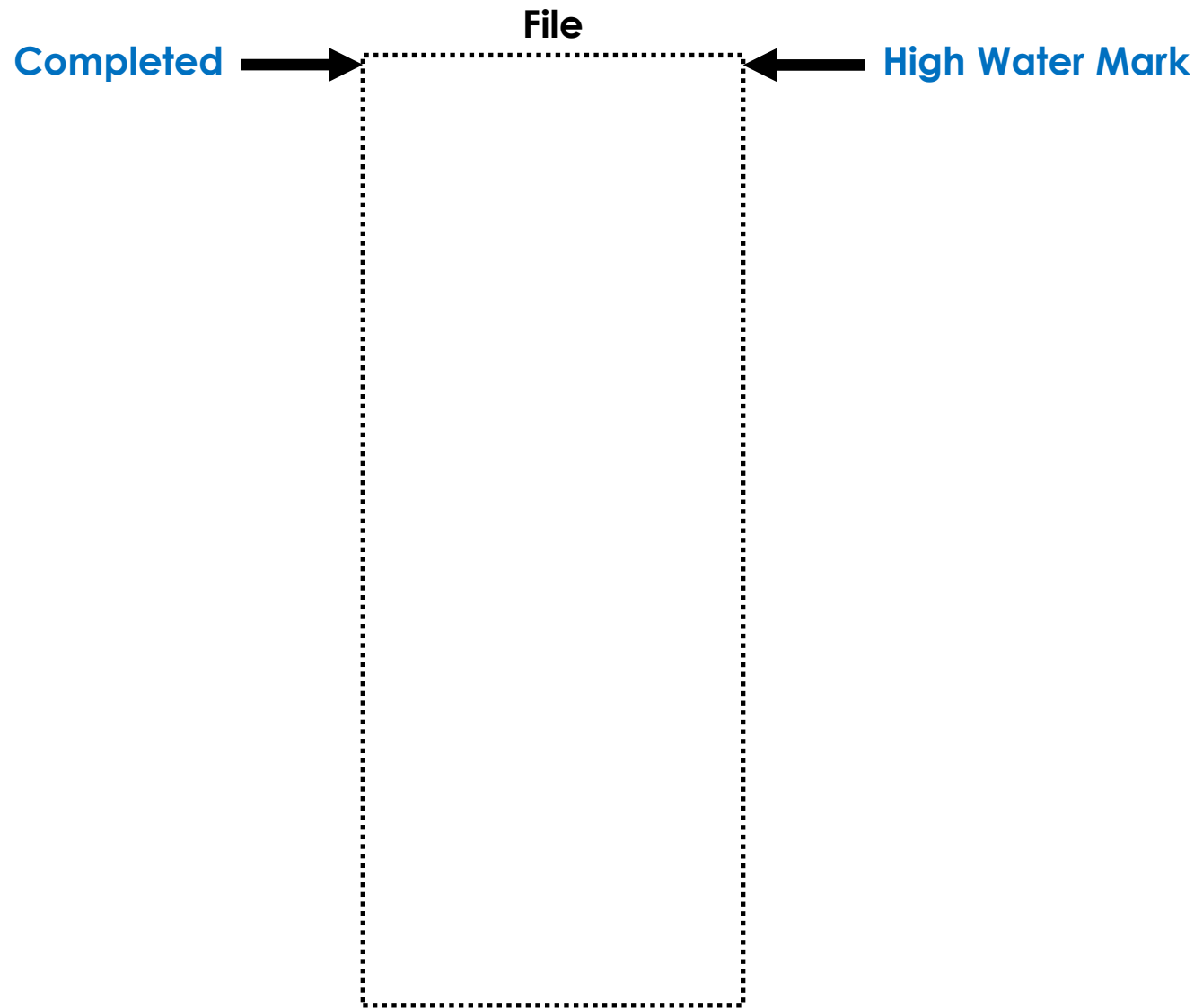




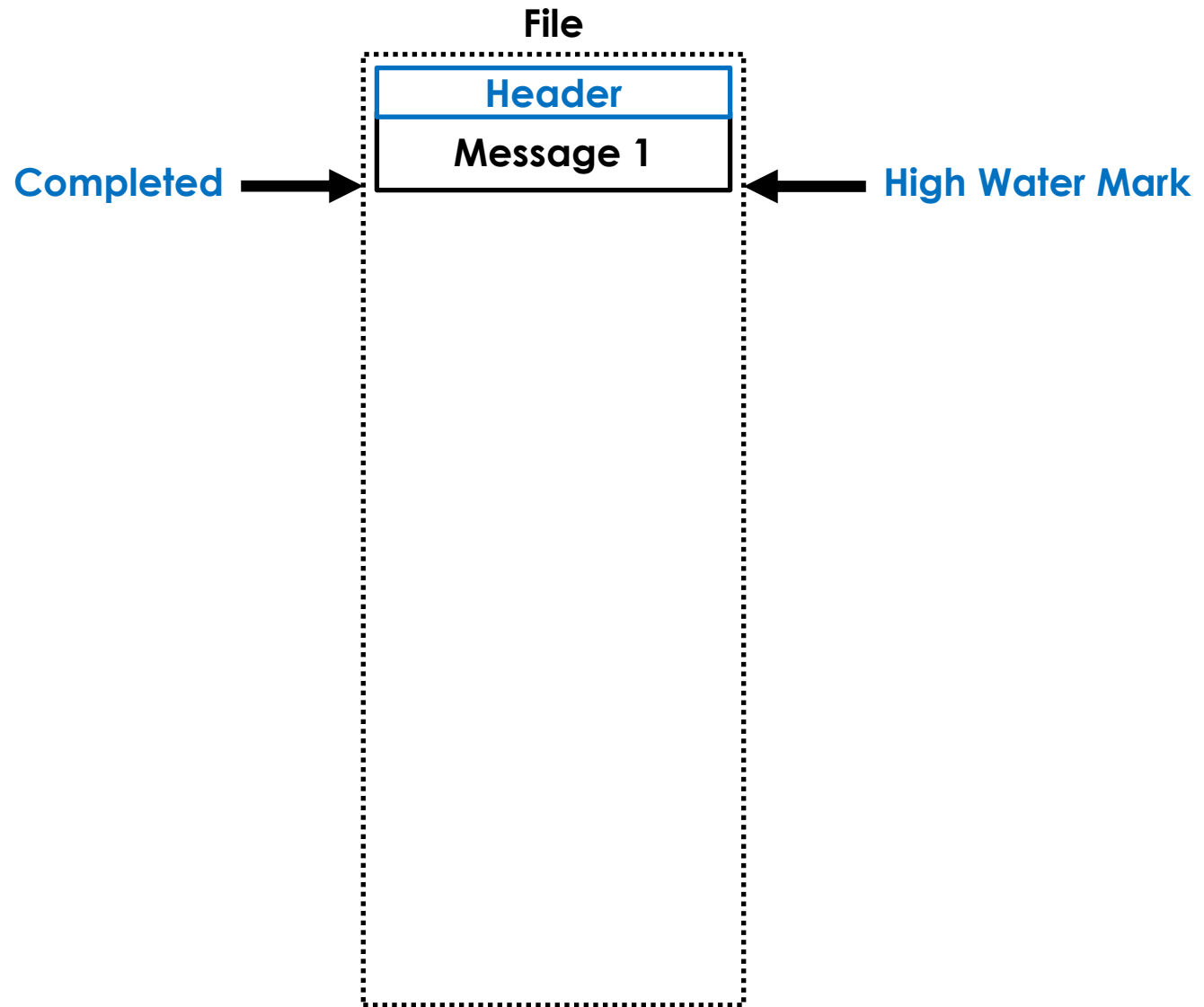


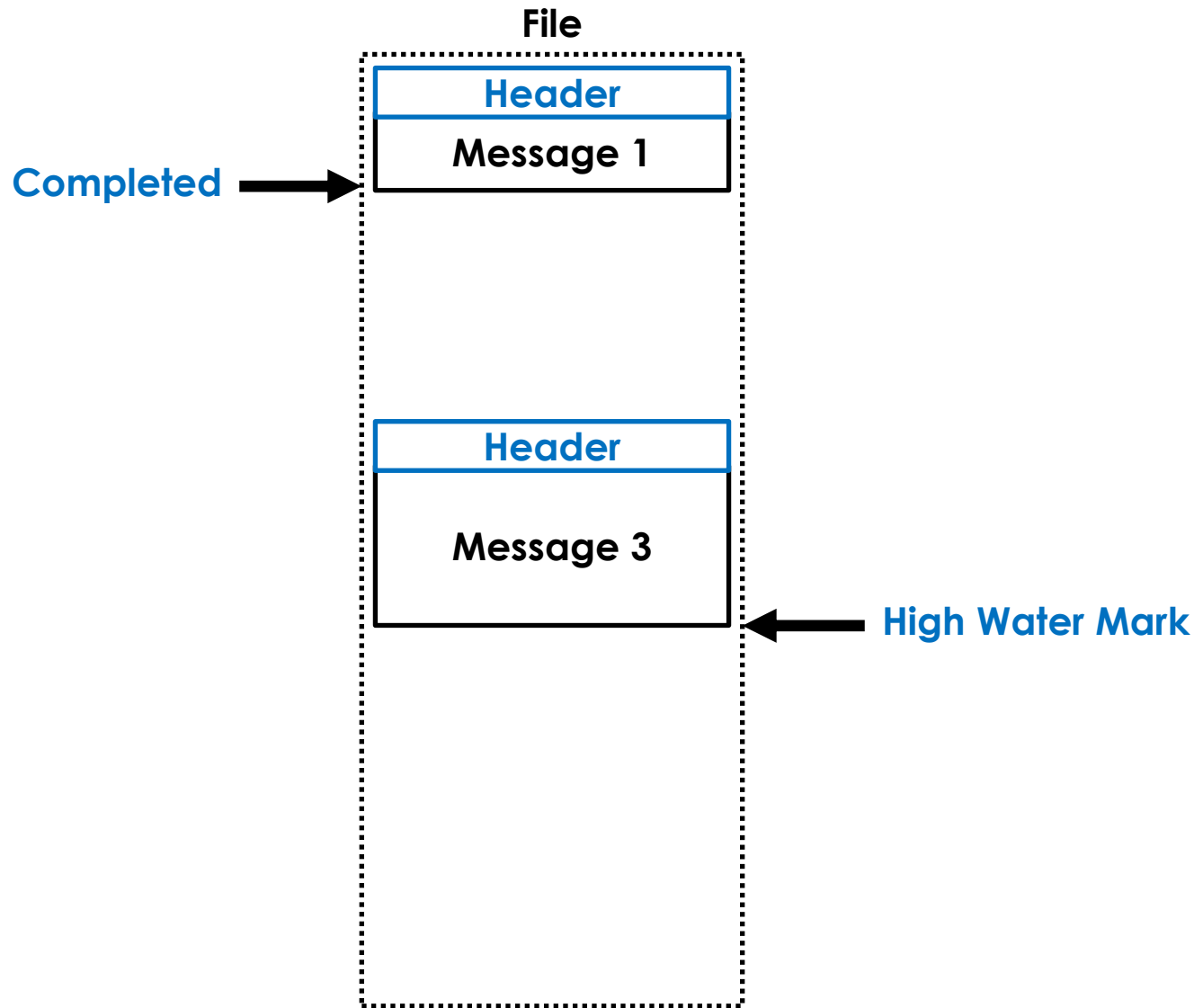


***How are message streams  
reassembled?***

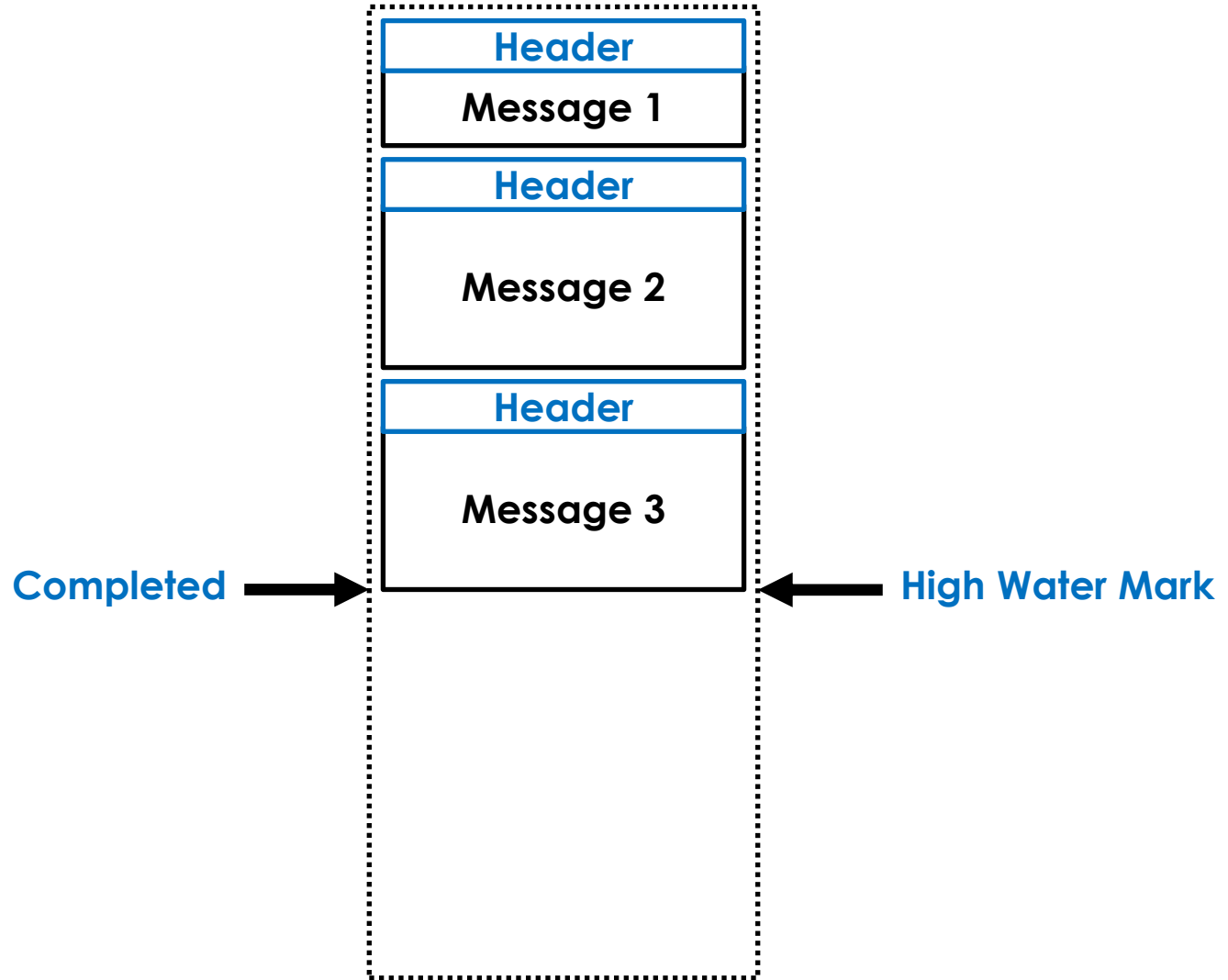








# File



***What if a gap is never filled?***

***How do we know what is  
consumed?***

***Publishers, Senders,  
Receivers, and Subscribers  
all keep position counters***

***Counters are the key to  
flow control and monitoring***

***Protocols can be more subtle  
than you think...***



***What about  
“Self similar behaviour”?***

***4. What did we learn  
on the way?***

***Humans suck at estimation!!!***

***Building distributed systems  
is **Hard!*****

***We have more defensive code  
than feature code***

***This does not mean the  
code is riddled with  
exception handlers  
– Yuk!!!***

***Building distributed systems  
is Rewarding!***

# ***Monitoring and Debugging***



***Loss, throughput, and buffer size  
are all strongly related!!!***

**Pro Tip:** Know your OS network parameters and how to tune them

***Aeron can track  
application consumption***

***Aeron can track  
application consumption  
– No need for the Disruptor***

**Some parts of Java really suck!**

Some parts of Java really suck!

***Unsigned Types?***

Some parts of Java really suck!

***Unsigned Types?***

***NIO (most of) - Locks***

Some parts of Java really suck!

*Unsigned Types?*

*NIO (most of) - Locks*

*Off-heap, PAUSE, Signals, etc.*



Some parts of Java really suck!

***Unsigned Types?***

***NIO (most of) - Locks***

***Off-heap, PAUSE, Signals, etc.***

***String Encoding***

Some parts of Java really suck!

***Unsigned Types?***

***NIO (most of) - Locks***

***Off-heap, PAUSE, Signals, etc.***

***String Encoding***

***Managing External Resources***

Some parts of Java really suck!

*Unsigned Types?*

*NIO (most of) - Locks*

*Off-heap, PAUSE, Signals, etc.*

*String Encoding*

*Managing External Resources*

*Selectors - GC*

# Bytes!!!

```
public void main(final String[] args)
{
    byte a = 0b0000_0001;
    byte b = 0b0000_0010;

    byte flags = a | b;

    System.out.printf(
        "flags=%s\n",
        Integer.toBinaryString(flags) );
}
```

# Bytes!!!

```
public void main(final String[] args)
{
    byte a = 0b0000_0001;
    byte b = 0b0000_0010;

    byte flags = a | b;

    System.out.printf(
        "flags=%s\n",
        Integer.toBinaryString(flags) );
}
```

# Bytes!!!

```
public void main(final String[] args)
{
    byte a = 0b0000_0001;
    byte b = 0b0000_0010;

    byte flags = a | b;

    System.out.println(
        "Flags: " + Integer.toBinaryString(flags));
}
```

**Error:(8, 24) java: incompatible types:  
possible lossy conversion from int to byte**

**Some parts of Java are really nice!**

Some parts of Java are really nice!

***Tooling – IDEs, Gradle, HdrHistogram***



Some parts of Java are really nice!

***Tooling – IDEs, Gradle, HdrHistogram***

***Lambdas & Method Handles***

Some parts of Java are really nice!

***Tooling – IDEs, Gradle, HdrHistogram***

***Lambdas & Method Handles***

***Bytecode Instrumentation***

Some parts of Java are really nice!

***Tooling – IDEs, Gradle, HdrHistogram***

***Lambdas & Method Handles***

***Bytecode Instrumentation***

***Unsafe!!! + Java 8***

Some parts of Java are really nice!

***Tooling – IDEs, Gradle, HdrHistogram***

***Lambdas & Method Handles***

***Bytecode Instrumentation***

***Unsafe!!! + Java 8***

***The Optimiser***

Some parts of Java are really nice!

***Tooling – IDEs, Gradle, HdrHistogram***

***Lambdas & Method Handles***

***Bytecode Instrumentation***

***Unsafe!!! + Java 8***

***The Optimiser – Love/Hate***

Some parts of Java are really nice!

***Tooling – IDEs, Gradle, HdrHistogram***

***Lambdas & Method Handles***

***Bytecode Instrumentation***

***Unsafe!!! + Java 8***

***The Optimiser – Love/Hate***

***Garbage Collection!!!***

## ***5. What's the Roadmap?***

***We are major feature complete!***

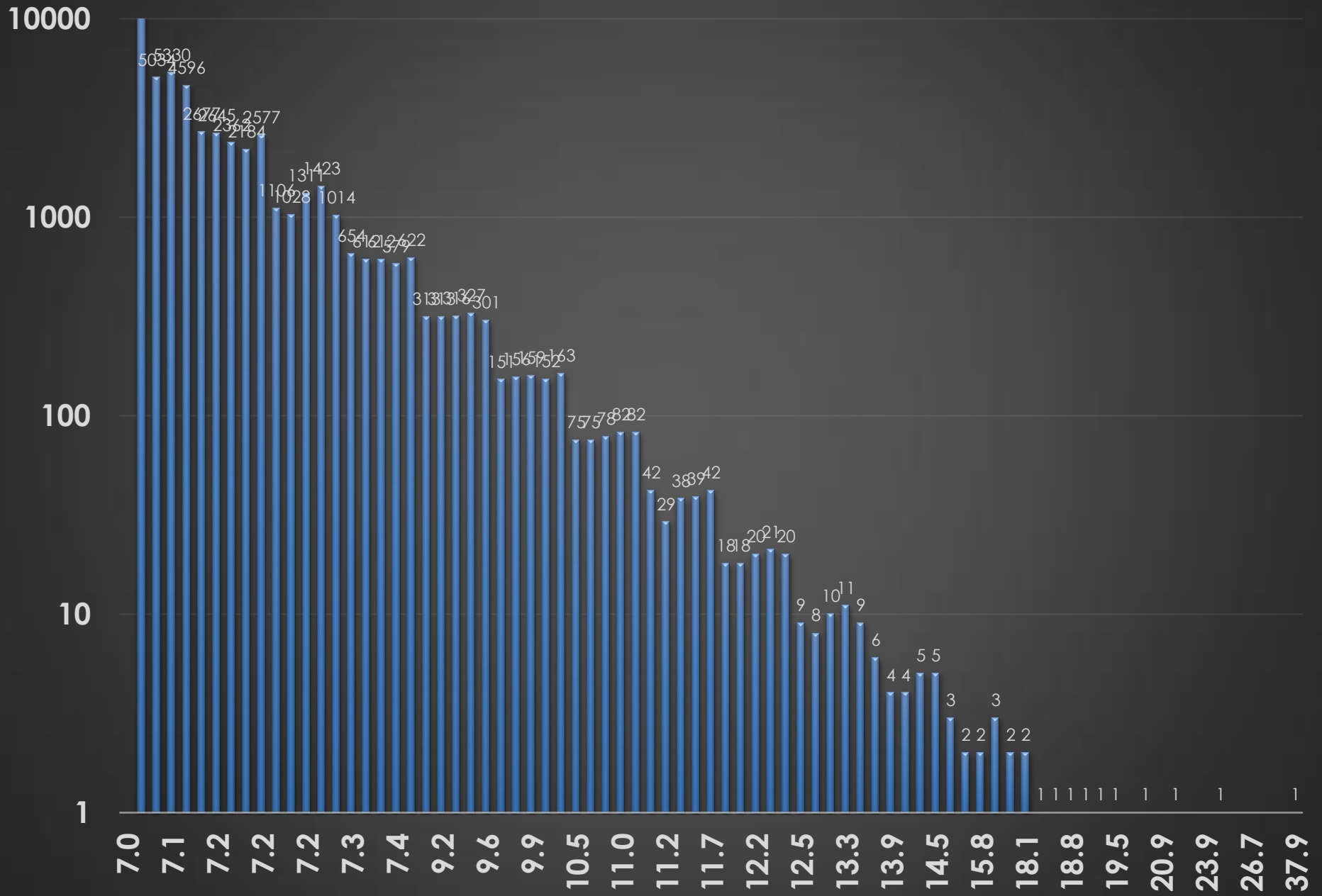


***Finished one pass at  
Profiling and Tuning***

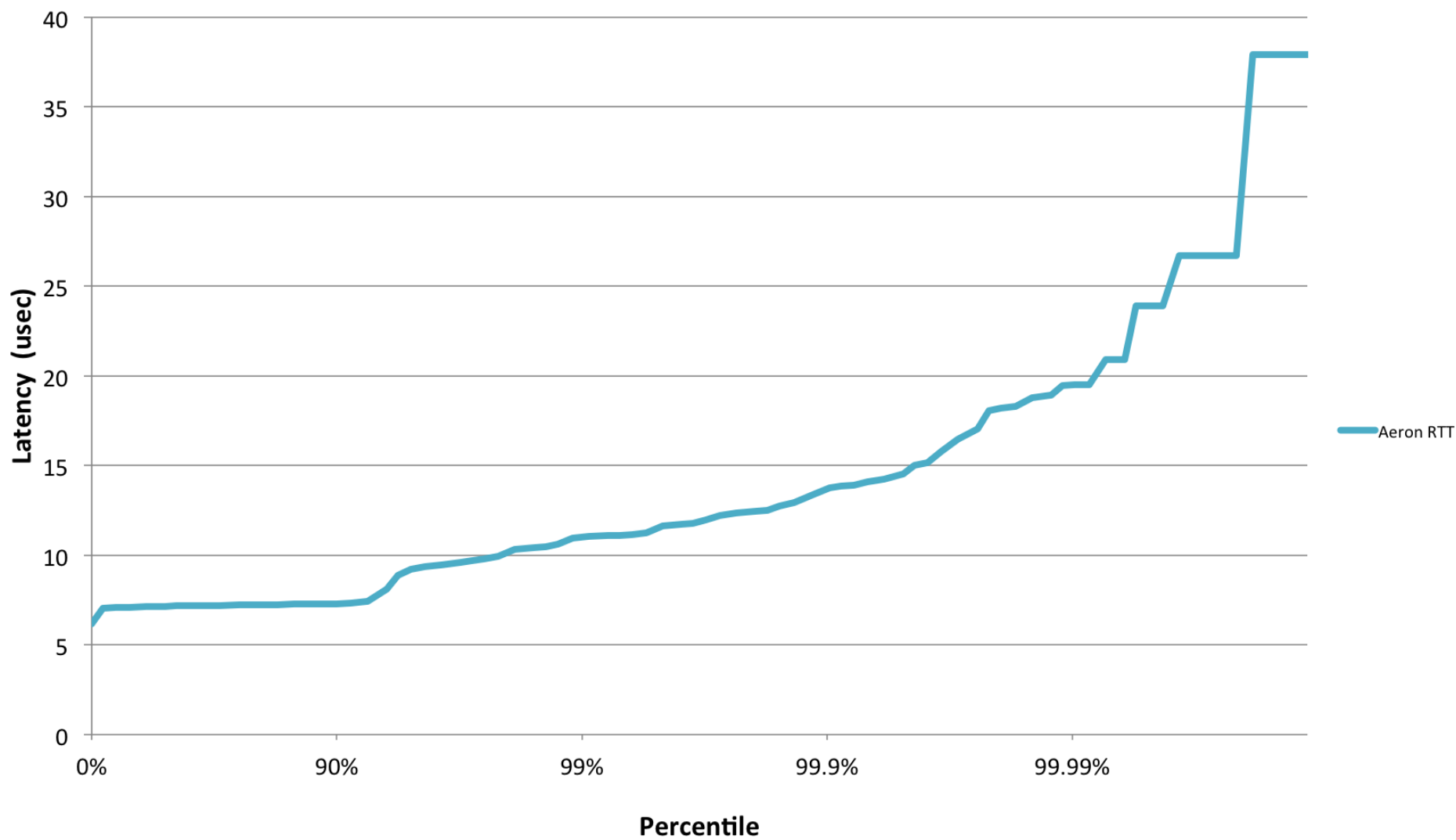
***Things are looking **very** good***

***20 Million 40 byte  
messages per second!!!***

# Latency Distribution ( $\mu$ s)



# RTT Latency by Percentile Distribution



**C++ *Port coming next***

***Then IPC and Infiniband***

***We have discussed FPGA  
implementations with 3<sup>rd</sup> Parties***



***In closing...***

A black short-sleeved t-shirt is laid flat against a white background. The t-shirt has a crew neck and a small white tag with a colorful logo inside the collar. The text "Do epic shit, or die trying." is printed in a bold, white, sans-serif font across the chest.

**Do epic shit,  
or die trying.**

Where can I find it?

***<https://github.com/real-logic/Aeron>***

# Questions?

Blog: <http://mechanical-sympathy.blogspot.com/>

Twitter: @mjpt777

***“Any intelligent fool can make things bigger,  
more complex, and more violent.***

***It takes a touch of genius, and a lot of  
courage, to move in the opposite direction.”***

**- Albert Einstein**