Solution Approach (1)

We decided to use **Artery with ByteBuffers**.

To us, all of the other suggested solutions seemed rather inadequate or they are part of Artery's implementation.

- Using **Akka Streams** directly would be a reimplementation without a separate, non-blocking side-channel. Artery itself is based on Akka Streams when using TCP.
- The **Akka http client-server** component is useful when integrating with other external systems. However, this communication is potentially heavier and makes one dependent on an external, uncontrolled system.
- **Splitting into multiple ByteMessages** requires the implementation of an an in-order + exactly–once delievery mechanism. Also it's potentially blocking system messages and has overhead metadata.

Solution Approach (2)

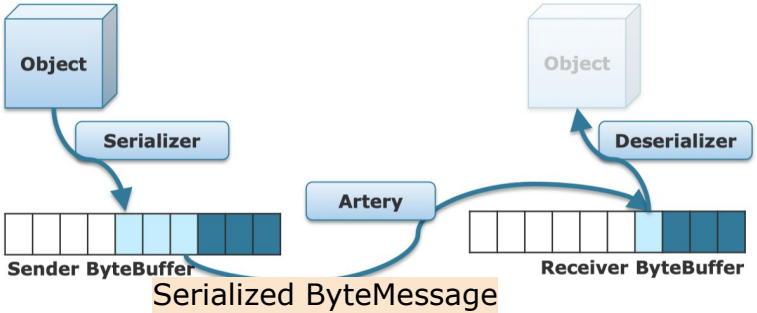

Illustration from the lecture's slides
(Hands on Akka Actor Programming, slide 43)

- We serialize generic messages *T* using a Kryo serializer
- We changed identifier-strings from the ActorRef-objects (sender & receiver) to make them portable
- The custom LargeMessageByteBufSerializer converts the serialized ByteMessage into a byte buffer that is then transferred with Artery.

Problems we encountered

- Lack of documentation on ByteBuffers in combination with Kryo
- The “would-just-work” solutions would require a lot of reimplementation or had disadvantages in real-life use-cases like blocking
- Test Suite didn’t specify the actual requirements but rather worked as redundant regression check. It was not using serialization.
- Limited Debugger options in async environment
- Verbose or hard-to-understand Kryo exceptions

```
[00:08:49.129 ERROR]                    Encoder(akka://ddm)| Failed to serialize mes
sage [ActorSelectionMessage(de.hpi.ddm.actors.LargeMessageProxy$BytesMessage)].
com.esotericsoftware.kryo.KryoException: Buffer overflow. Available: 7, required: 8
```