

# Apache Kafka

**Modern alternative for arranging the architecture of larger information systems and the integration of several systems**

11.10.2022



Haaga-Helia

# Apache Kafka

- “Apache Kafka is an open-source **distributed event streaming platform** used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.” (<http://kafka.apache.org>)
- ” More than **80% of all Fortune 100 companies** trust, and use Kafka” (<http://kafka.apache.org>)
- Usually you should not use the creators/owners of technology as source, especially when very positive claims are used. Here the two statements above have been proven by practice though.
- Kafka is also developed by the community together and it's not proprietary project.
- And as we are not doing thesis or other academic research those quotes are maybe ok

# Publish-subscribe pattern

- An example of the messaging patterns.
- (Only extra reading: [https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe\\_pattern](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern))
- **Publisher** publishes **messages/events** to **stream/queue/broker**. Publisher does not know the subscribers
- **Subscriber subscribes to receive** messages/events from the stream/queue/broker. Subscriber does not need to know the publishers
- These systems often provide these characteristics or benefits:
  - Distributed system with minimal dependencies
  - Sub-systems written even with other programming languages as long as the message is of a common format
  - Asynchronous operation, Possibly/probably buffering and message retention for long time / 'forever' if we so like
  - First come first served principle if we so wish. Or some priorities used.

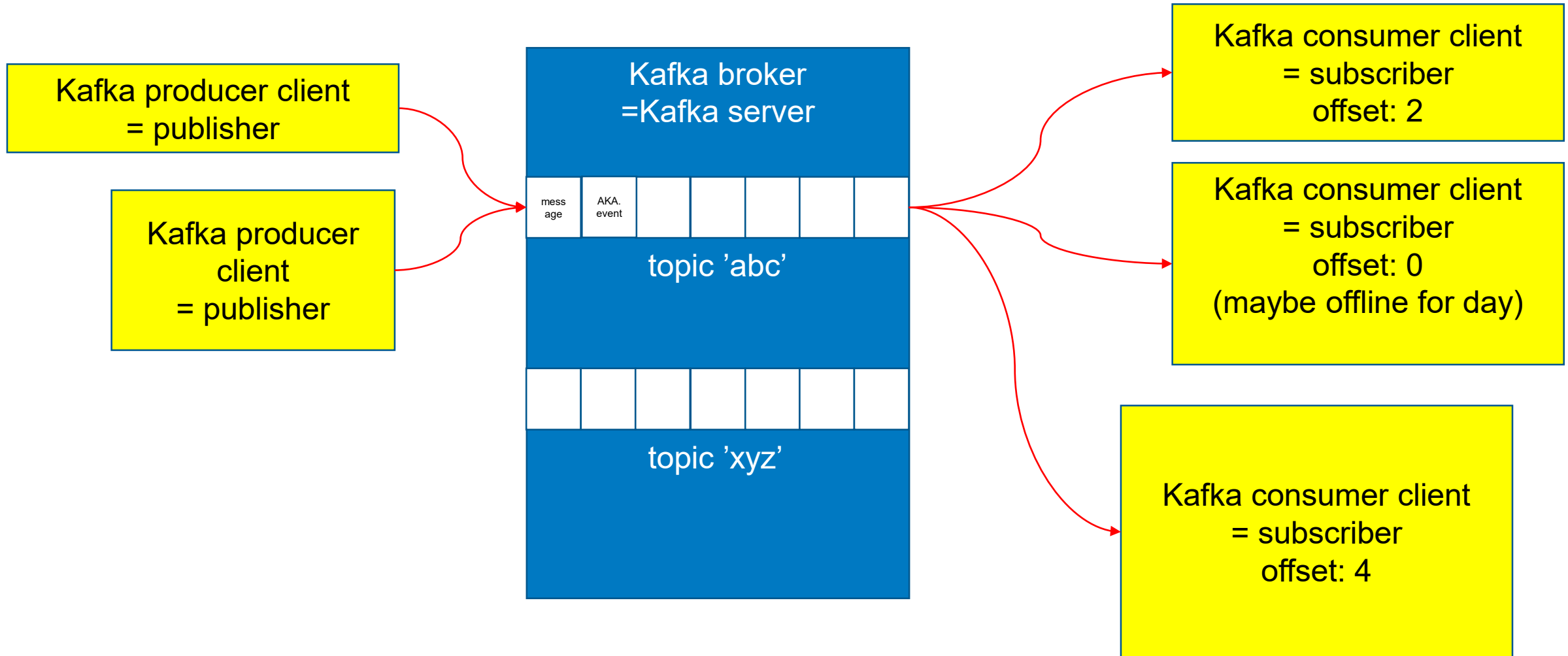
# Loosely related terms for discussing this topic

- **Event / Message** - Message or signal or other data from system or subsystem to another
- **Broker / Stream / Bus / (Queue) / (Buffer)** - "Server" needed for relaying messages
- **Publisher / Producer** – Application or process that creates messages to the server
- **Subscriber / Consumer** - Application or process that will receive messages from the server
- **Topic / Category / Channel / Feed / Tag** – Way to have multiple streams in same broker process, thus also connect the ends more than one way (A might produce message to topic X, but consume a message from topic Y. B might do vice versa)

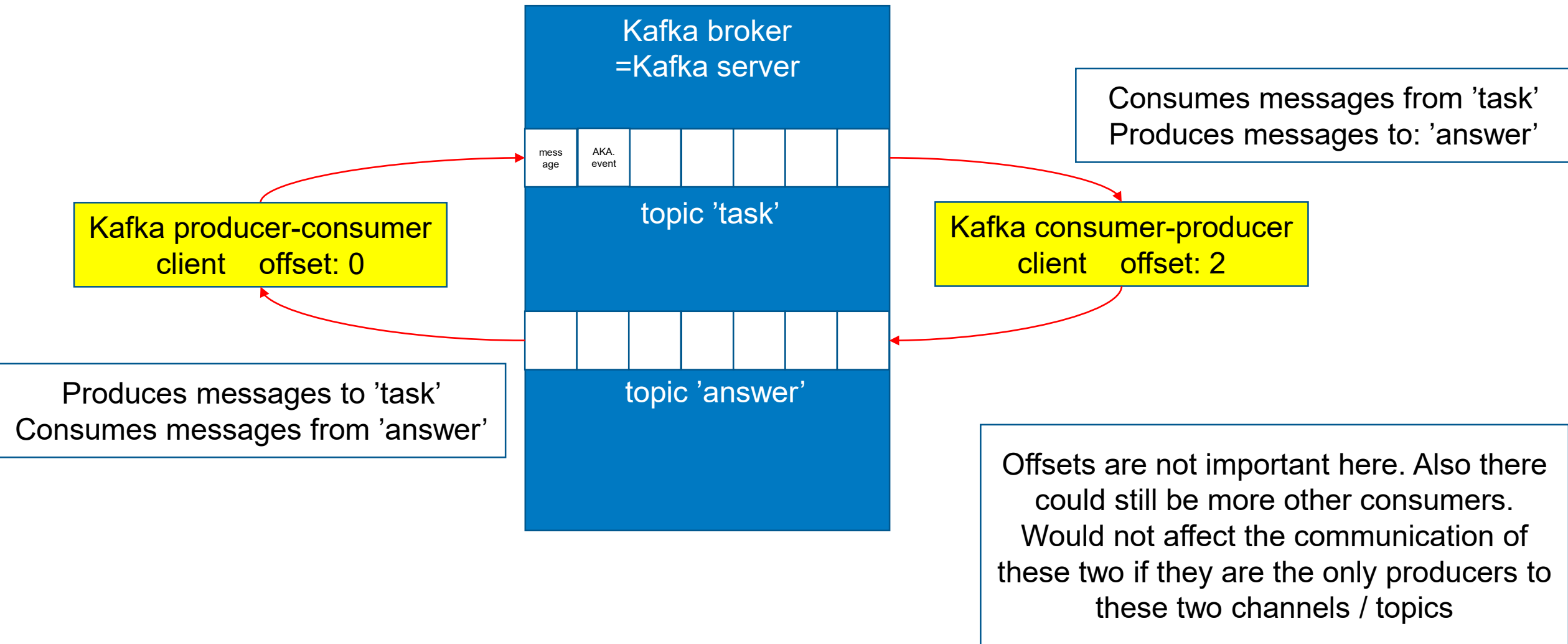
# Kafka uses some of those common terms

- **Event** - Message or signal or other data from system or subsystem to another
- **Broker** - "Server" needed for relaying messages
- **Producer** – Application or process that creates messages to the server
- **Consumer, subscribe** - Application or process that will receive messages from the server
- **Topic** – Way to have multiple streams in same broker process, thus also connect the ends more than one way (A might produce message to topic X, but consume a message from topic Y. B might do vice versa)
- Offset – Basically an index on how much a certain Consumer has already handled of the arrived messages
- <https://data-flair.training/blogs/kafka-terminologies/>
- Not so important Kafka terms now (related to performance, reliability, modularity, etc. not the first simple operation you need):
  - Cluster, Node, Partition, Log, Replica, Leader, Follower, Consumer group

# Distributed Event Stream architecture – e.g. Kafka



# How two systems can 'discuss' in Kafka



# Create your own Kafka test?

- You can write Kafka clients (publishers and/or subscribers) with at least these languages:
  - <https://cwiki.apache.org/confluence/display/KAFKA/Clients>
  - E.g. Node.js, C/C++, .NET, Java, Python, Rust, Ruby, Swift... Basically any language that has any relevance nowadays
- Advance step by step, testing each step to reduce error possibilities.
- There is an Assignment available that first is just writing based on model and making it to run



# Steps of Kafka demo/tryout system development

- Download Kafka runnables (Kafka server and Zookeeper) (Or use them from Docker image repository)
- Configure Kafka server and Zookeeper process (Or the docker container for them)
- **Start** the Kafka server
- Setup a Topic (A name for the Topic, kind of a Channel)
- Code your Kafka producer client with your preferred programming language and runtime so that it puts some message to server's Topic
  - Start the producer
- Code your Kafka consumer client with your preferred programming language and runtime so that it subscribes to receive those messages.
  - Language does not have to be same as the producer. But possibly you want to use the same for simplicity
  - Start the consumer



# Happy hacking!

Remember also worklife balance and outdoor activities