Stream processing using Kafka and Faust

Johannes Ahlmann, Sensatus.io

PyCon Limerick

2020-02-28

About Me

Johannes Ahlmann

- Living in Cork
- Developing in Python since 2002
- Built large-scale Machine Learning solutions using Python, Tensorflow, Kafka, Spark

Sensatus.io

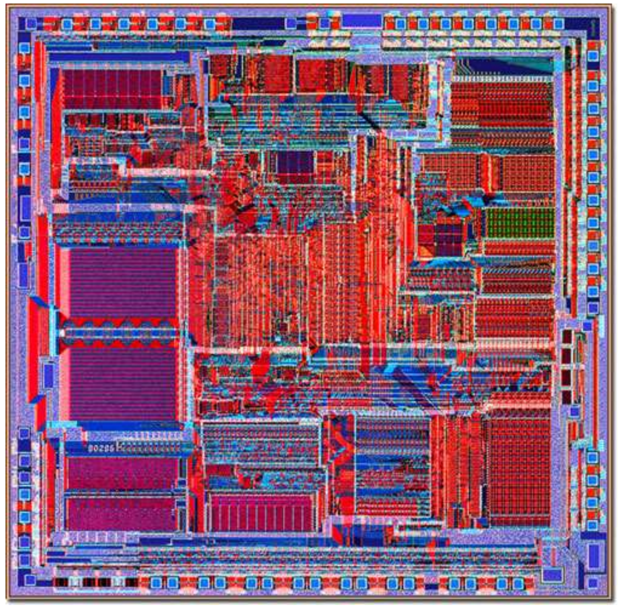
- On-Prem AI Models
- Gathering and Enriching Web Data
- Sales & Client Intelligence

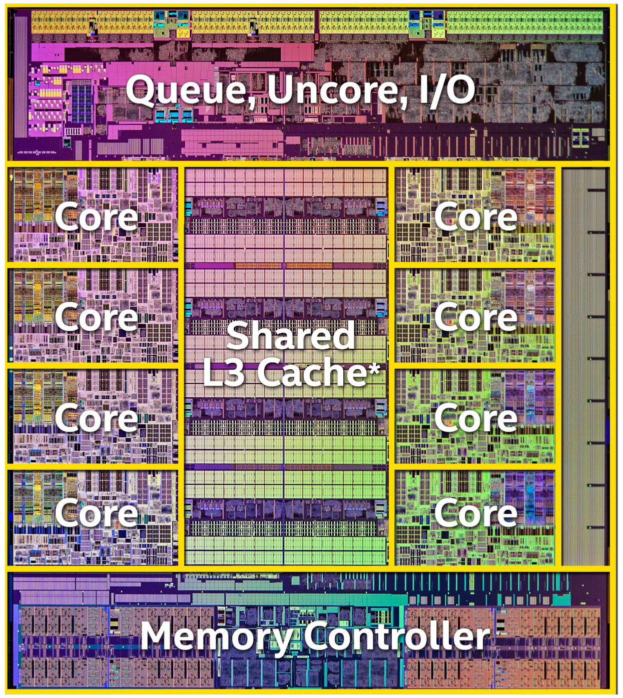
Github: **@codinguncut**

- codinguncut/python_faust

sensatus.io

Times Have Changed


Intel 80286
1 core, 0.13M transistors



Intel i7-9900K
8 cores, 3000M transistors

Challenges

Many fantastic libraries written in C, Fortran, Python, Go

How can we leverage libraries together that are written in different languages, or for different (virtual) machines?

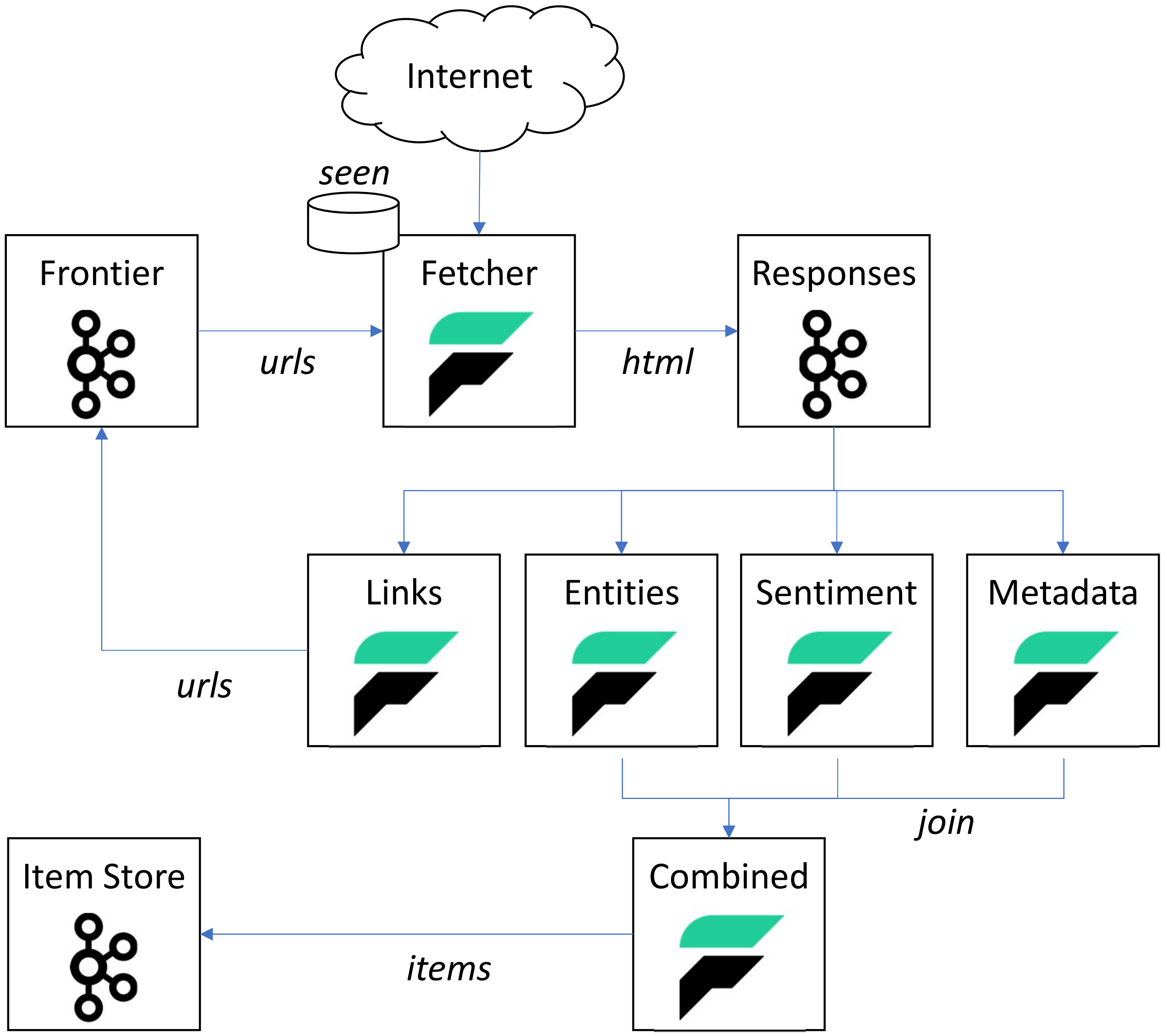
Multiprocessing and concurrency are hard in C, Python, etc.

How can we leverage compute of multiple cores per machine, let alone multiple machines

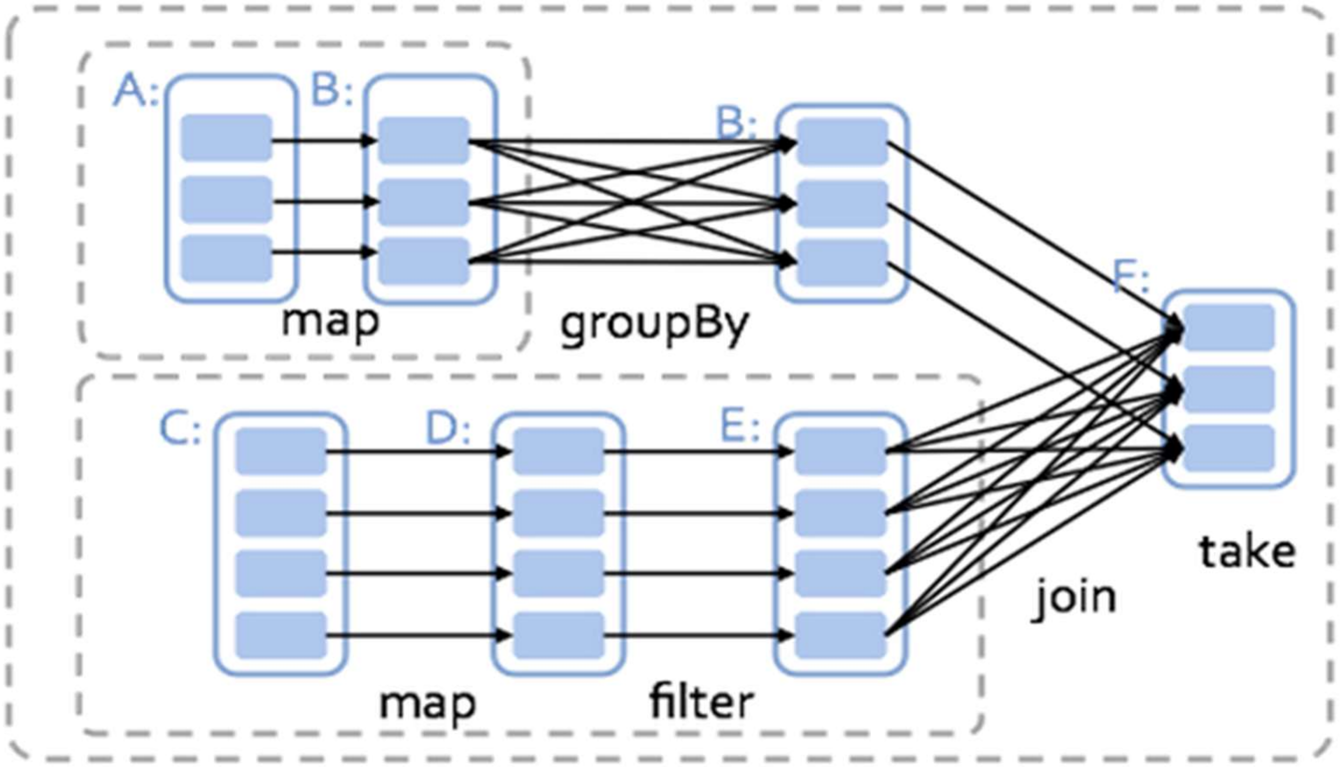
Distributed computing is hard

How can we monitor/ manage/ debug/ reason about it?

Computational Graph, Microservices



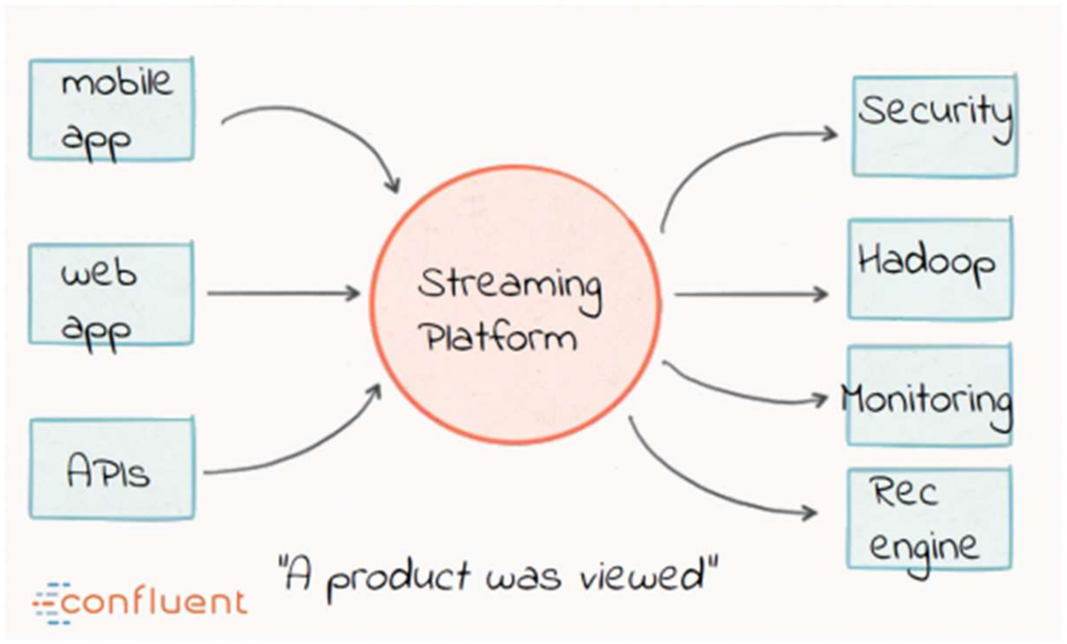
(Spark) Operator Graph



kafka

A high-throughput distributed messaging system

- Decouples Data Pipelines
- Scalable & Fault-Tolerant
- Kafka Functionalities
  - Messaging
  - Processing
  - Storing
- Performance (>100k/s)
  - Batching
  - Zero Copy I/O
  - Leverages OS Cache
- Durability



Core Concepts

- Producers
- Consumers
- Brokers
- Topics
- Zookeeper
    - Offsets
    - Broker Addresses

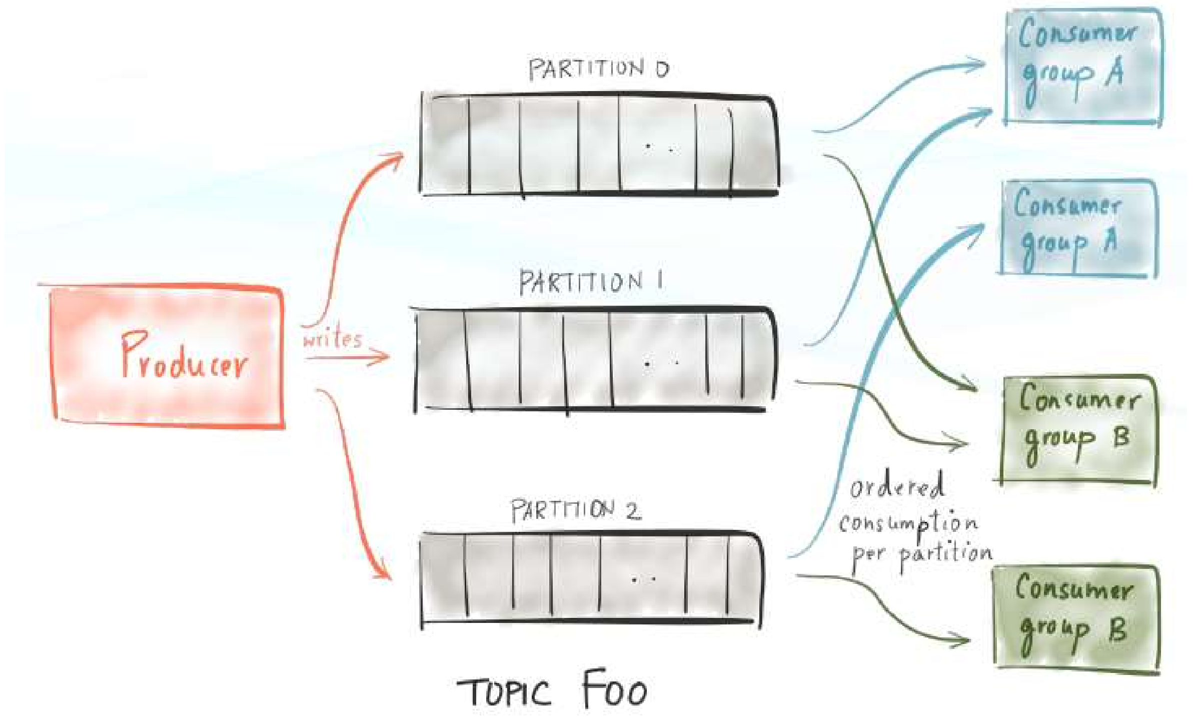
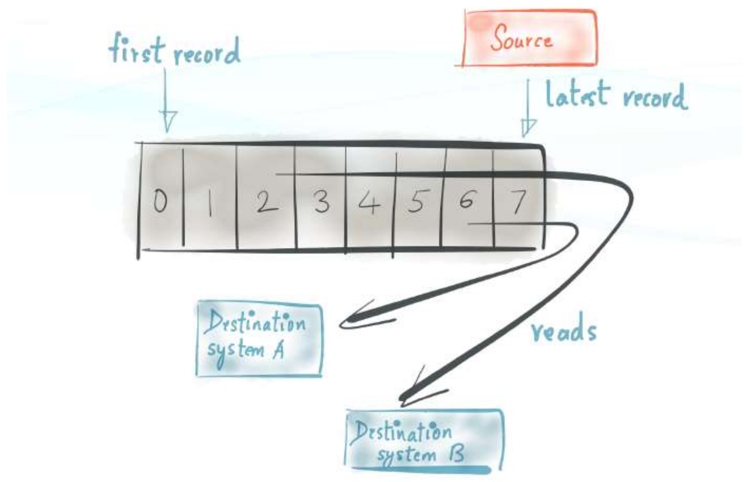
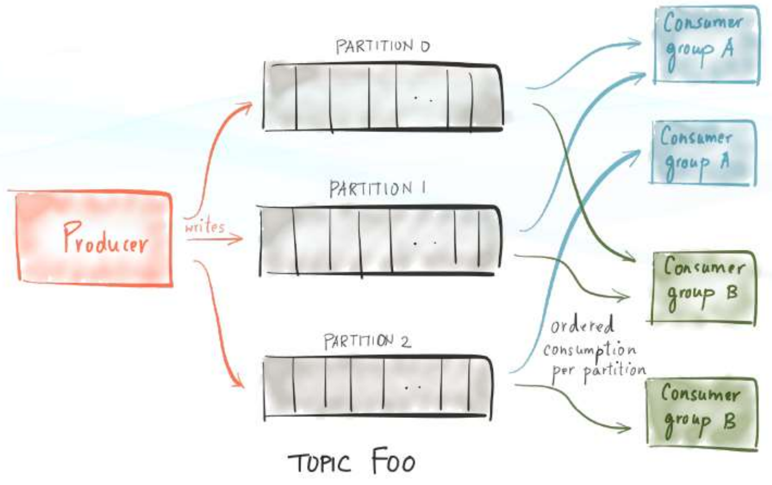


Image Credit: Confluent

Logs & PubSub

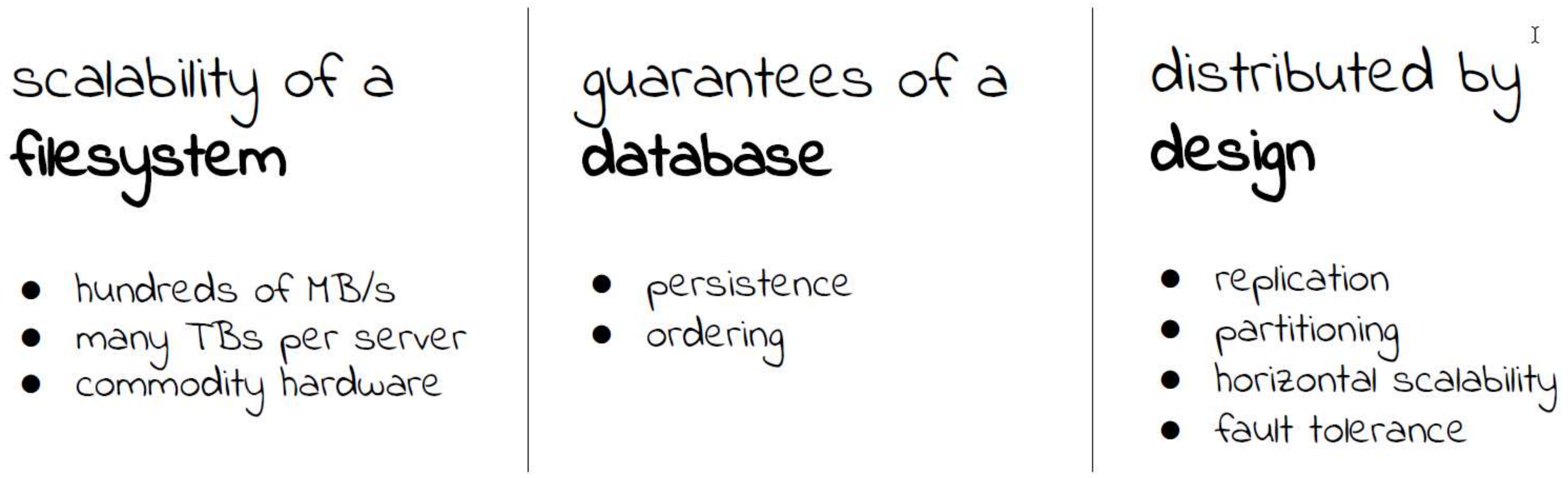
- Consumers can be transient
- Consumer Groups
- Delivery Semantics
  - at least once (default)
  - at most once
  - exactly once
- Retention Policy
- Reprocessing





johannes@sensatus.io

Summary



scalability of a **filesystem**

- hundreds of MB/s
- many TBs per server
- commodity hardware

guarantees of a **database**

- persistence
- ordering

distributed by **design**

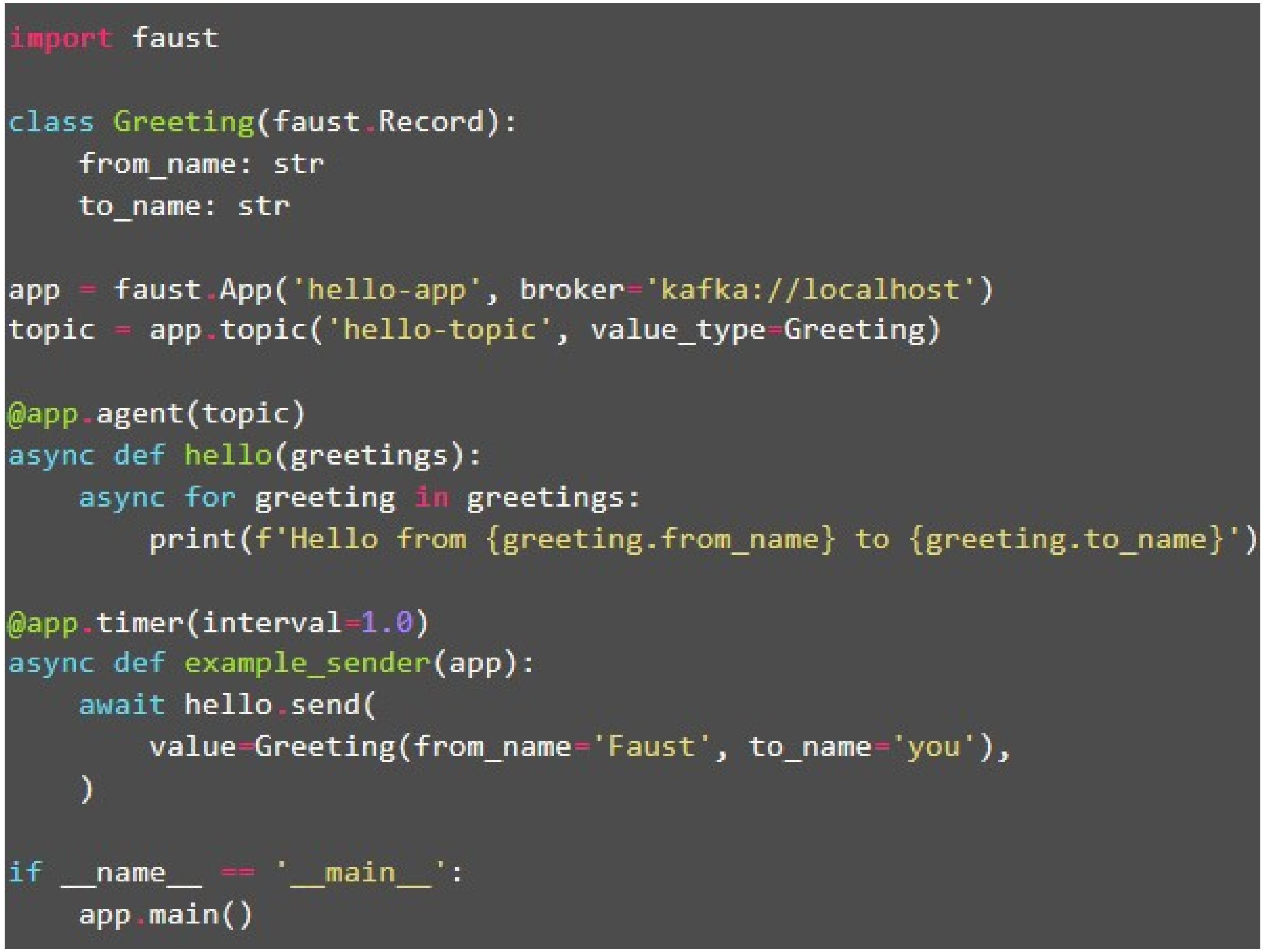
- replication
- partitioning
- horizontal scalability
- fault tolerance

Image Credit: Confluent

F A U S T

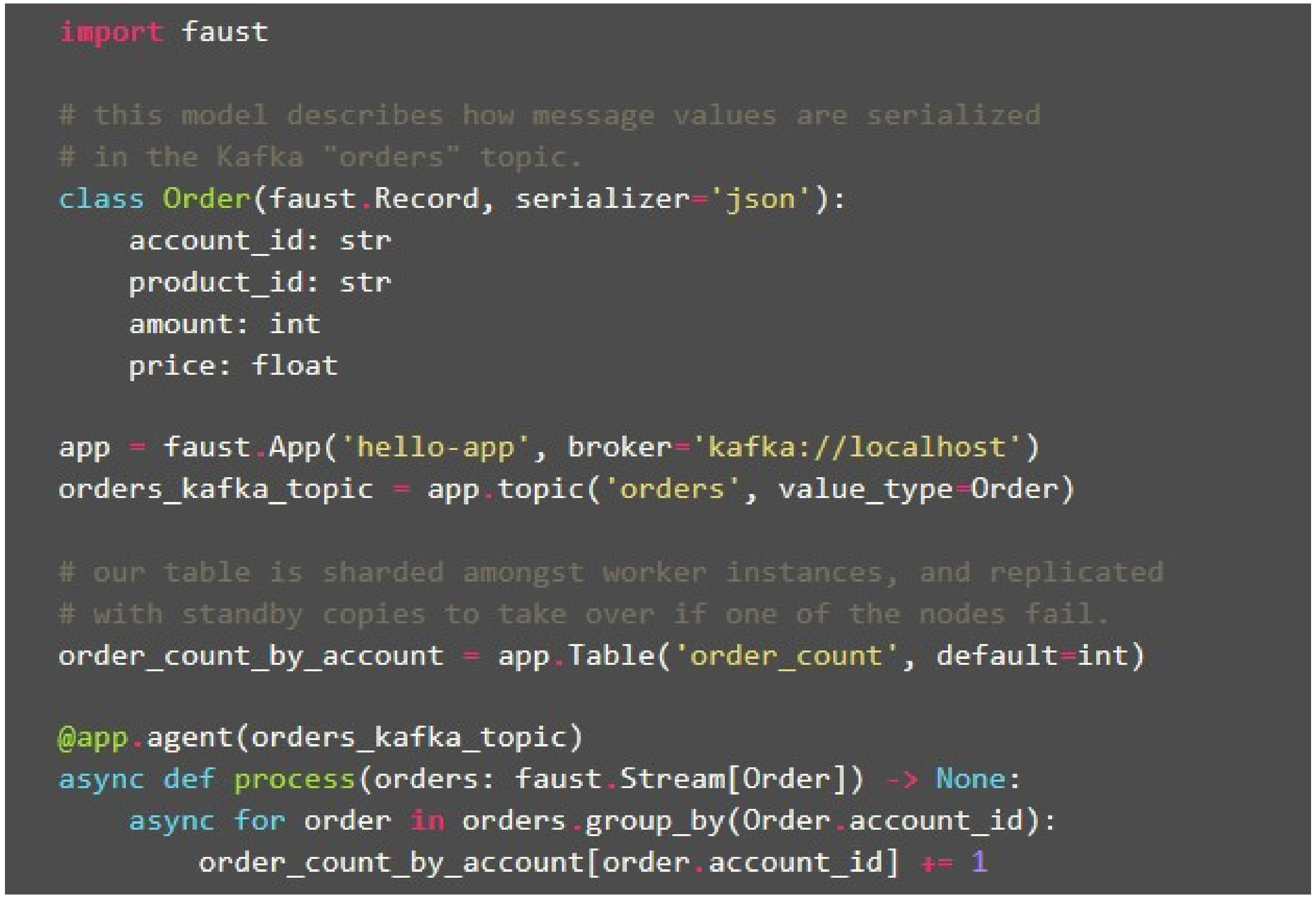
Faust

- Low overhead, start simple, scale out
- Use Kafka as a persistent transport mechanism, even between languages
- Replay, reprocess
- High Availability, Checkpointing, Error Recovery
- Per-agent scaling
- Regression testing
- Python kafka wasn't fun



```python
import faust

class Greeting(faust.Record):
    from_name: str
    to_name: str

app = faust.App('hello-app', broker='kafka://localhost')
topic = app.topic('hello-topic', value_type=Greeting)

@app.agent(topic)
async def hello(greetings):
    async for greeting in greetings:
        print(f'Hello from {greeting.from_name} to {greeting.to_name}')

@app.timer(interval=1.0)
async def example_sender(app):
    await hello.send(
        value=Greeting(from_name='Faust', to_name='you'),
    )

if __name__ == '__main__':
    app.main()
```

Tables

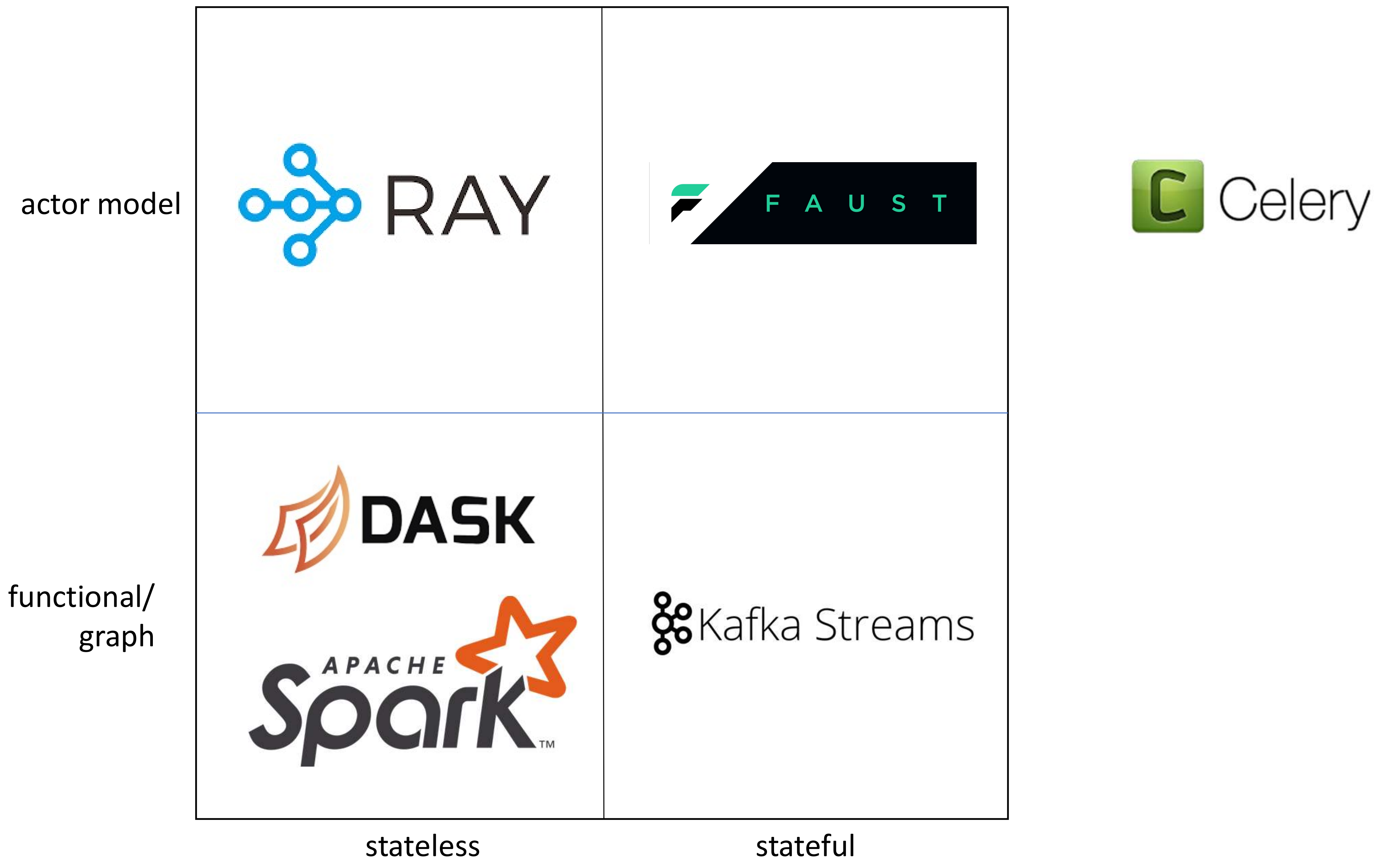


```
import faust

# this model describes how message values are serialized
# in the Kafka "orders" topic.
class Order(faust.Record, serializer='json'):
    account_id: str
    product_id: str
    amount: int
    price: float

app = faust.App('hello-app', broker='kafka://localhost')
orders_kafka_topic = app.topic('orders', value_type=Order)

# our table is sharded amongst worker instances, and replicated
# with standby copies to take over if one of the nodes fail.
order_count_by_account = app.Table('order_count', default=int)

@app.agent(orders_kafka_topic)
async def process(orders: faust.Stream[Order]) -> None:
    async for order in orders.group_by(Order.account_id):
        order_count_by_account[order.account_id] += 1
```

Demo Time

Alternatives

Faust	**A stream processing library, porting the ideas from Kafka Streams to Python.** **It provides both stream processing and event processing, sharing similarity with tools such as Kafka Streams, Apache Spark/Storm/Samza/Flink,**
Ray	A fast and simple framework for building and running distributed applications.
Dask	Provides advanced parallelism for analytics, enabling performance at scale for the tools you love
Spark	A unified analytics engine for large-scale data processing.
Celery	Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well


actor model
functional/ graph
stateless
stateful
RAY
FAUST
Celery
DASK
APACHE Spark
Kafka Streams

Thank you

johannes@sensatus.io