EPICS – Kafka Forwarder

By

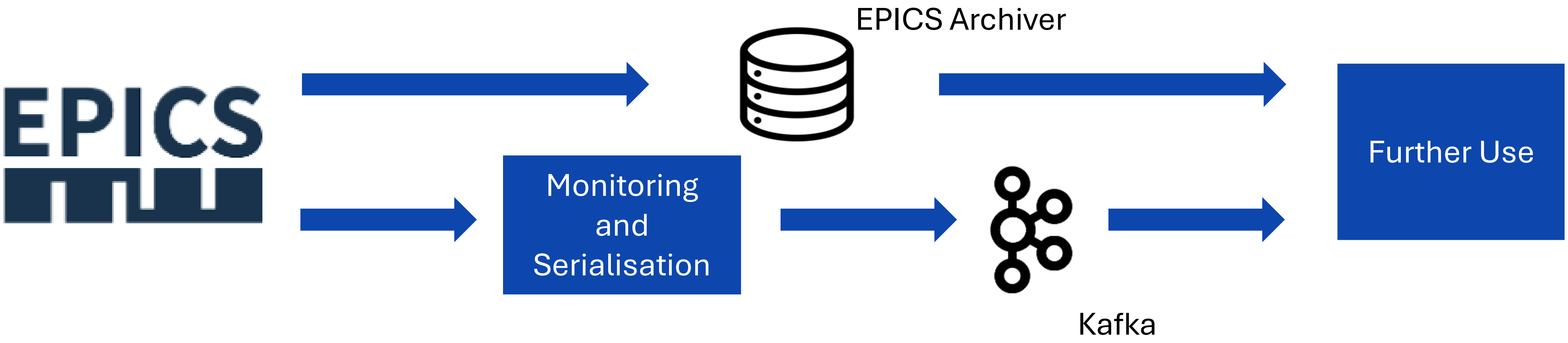
Anuj Mittal

Tata Institute of Fundamental Research, Hyderabad

EPAC Data Management and Data Acquisition

- Data volumes of up to 5GB/s and multiple PB over a year.
- Both operational and experimental data from multiple detectors.
- Data Acquisition Requirements:
  - Interfacing with control systems (EPICS).
  - Data will have attached metadata.
- Existing EPICS DAQ solutions (eg. EPICS Archiver Appliance) proved to be inadequate for the use case, so the idea was to use a central data broker, in this case, Apache Kafka.

Data Acquisition Initial Stage



Monitoring and Serialization

- ADKafka (Images)
- EPAC Forwarder (Scalars and Waveforms)

Prior Implementations

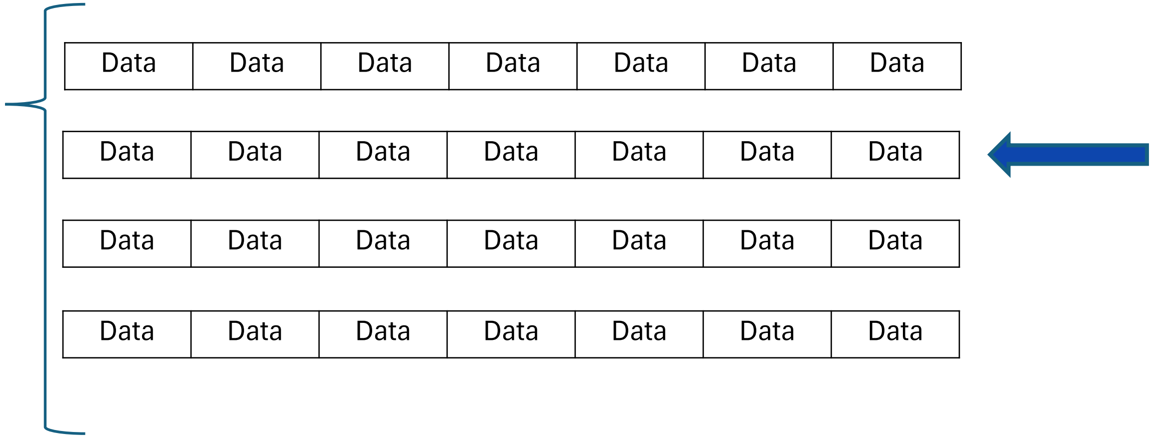
- We expand on work done by the European Spallation Source (ESS).
  - Schemas
  - Plugins
  - Forwarder
- Already in use at ISIS and other facilities.
- https://github.com/ess-dmsc

Why Kafka?

- Real-time data at scale.
- High-throughput, fault-tolerant messaging.
- Producers and consumers work independently.
- Complex data types.

Each topic represents one data source

Consumers can read data from anywhere



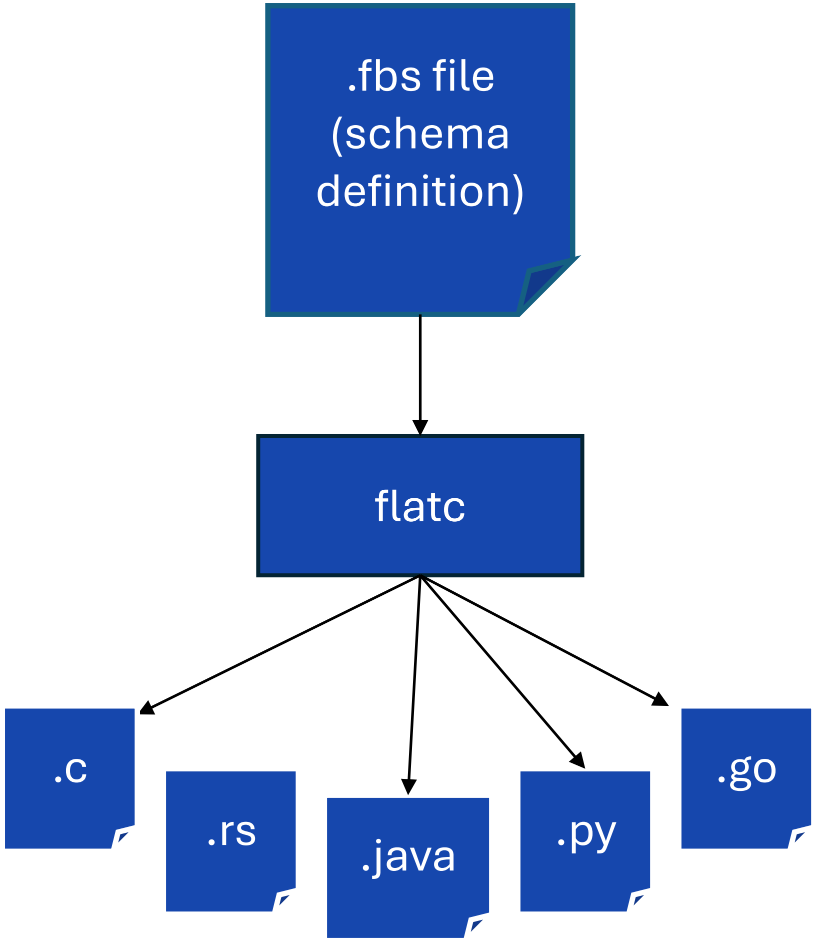
Data	Data	Data	Data	Data	Data	Data
Data	Data	Data	Data	Data	Data	Data
Data	Data	Data	Data	Data	Data	Data
Data	Data	Data	Data	Data	Data	Data

Producers add data to the end

Data may have **metadata** attached

Data Serialisation

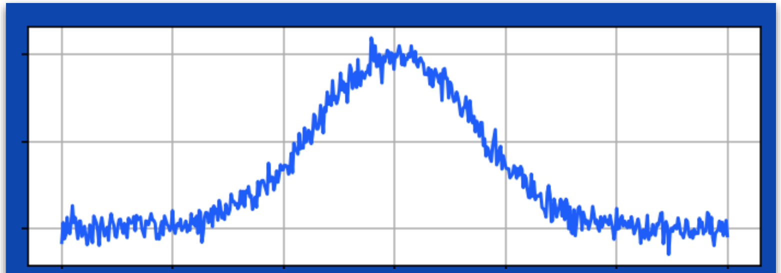
- No standard – Kafka can handle any bytes.
- We follow the lead of ESS and use FlatBuffers:
  - Fast.
  - Memory-Efficient.
  - Defined schemas and strong typing.
  - Schema can be evolved.



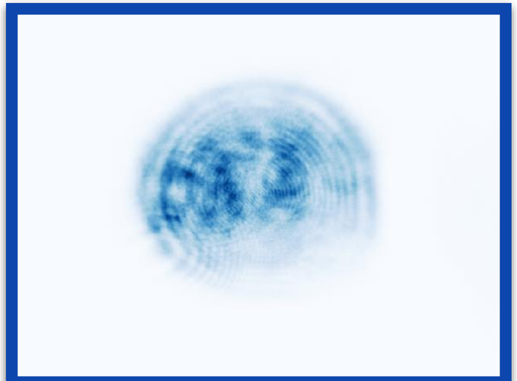
Data Types

13.52 J

Scalar



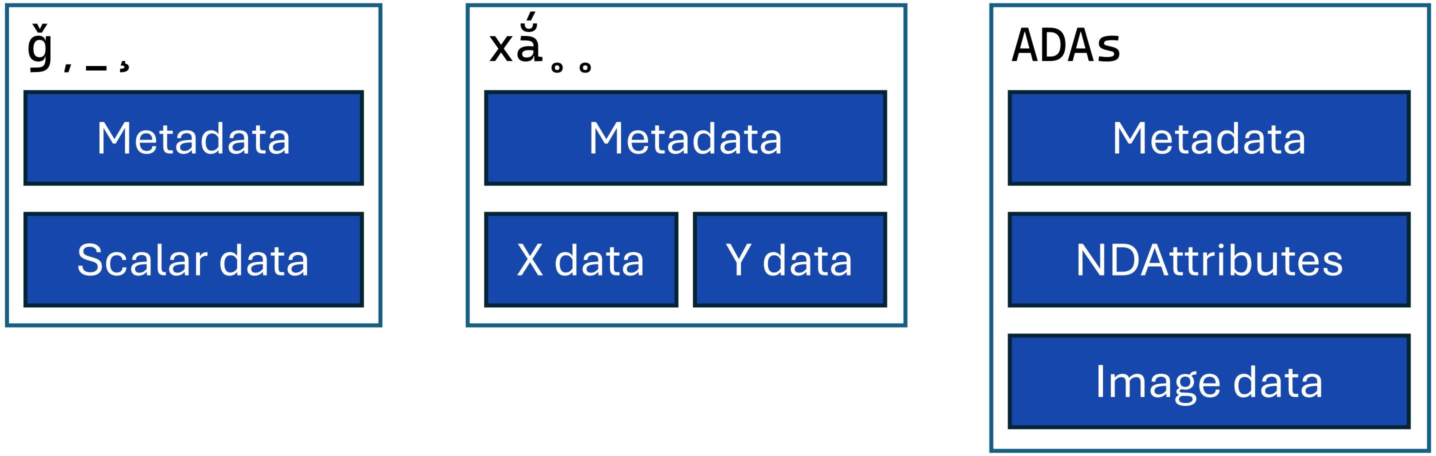
Waveform



Image

Data schemas

- ESS has many different schemas
- f142 for scalars and ADAr for images .
- Our own (wa00) for waveforms: combination of two arrays from two different PVs .



Data Schema Example - wa00

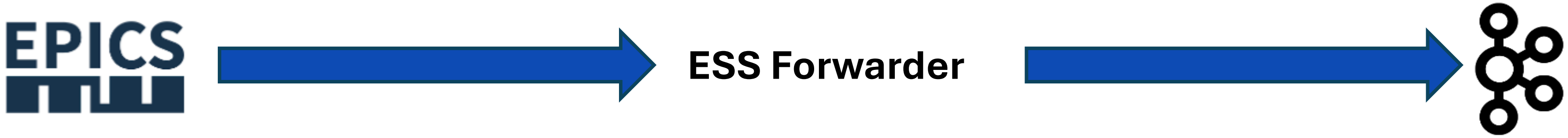
```
table WaveFormArray {
    timestamp: ulong;           // Timestamp in nanoseconds since UNIX epoch
    x_timestamp: ulong;           // Timestamp in nanoseconds since UNIX epoch
    x_data_type: DType;         // The type of the data stored in the x_data array
    y_data_type: DType;         // The type of the data stored in the y_data array
    x_data: [ubyte] (required);  // Elements in the x array
    y_data: [ubyte] (required);  // Elements in the x array
    x_unit: string;
    y_unit: string;
}

root_type WaveFormArray;
```

From x PV

From y PV

EPICS-Kafka interface: Scalar and Forwarder

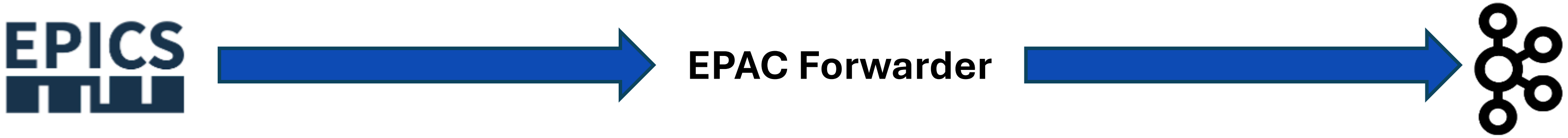


- ESS has a Forwarder.
    - Monitors PV.
    - Produces Kafka message when PV updates.
- Challenges:
    - Not enough metadata (eg. EGU).
    - No support for custom metadata.

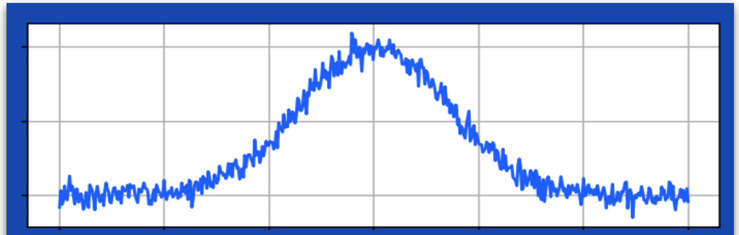
13.52 J

Scalar

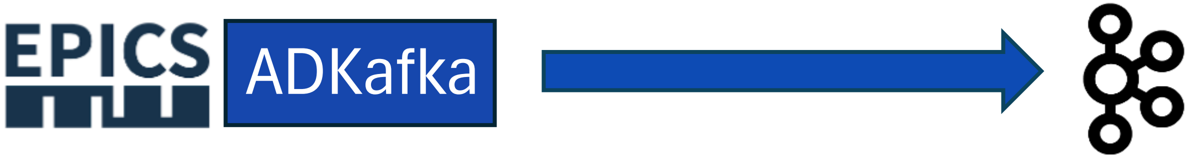
The challenge with Waveforms: Forwarder



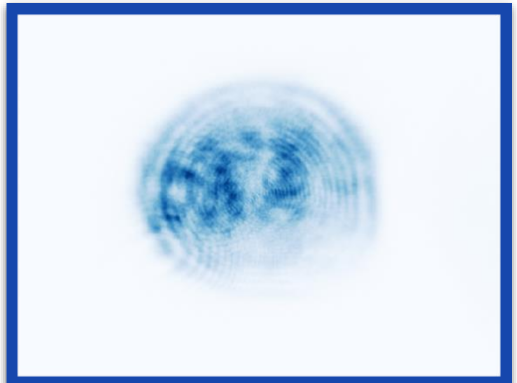
- Waveforms are a combination of two arrays from two different PVs.
- ESS Forwarder Challenges:
  - Large codebase
  - Complex to integrate 2 PV based waveform.
- Custom EPAC Forwarder was built.
  - Smaller python codebase < 400 lines.
  - Static configuration file.
  - Handles both Scalars and Waveforms.


Waveform

EPICS-Kafka interface: Images and ADKafka



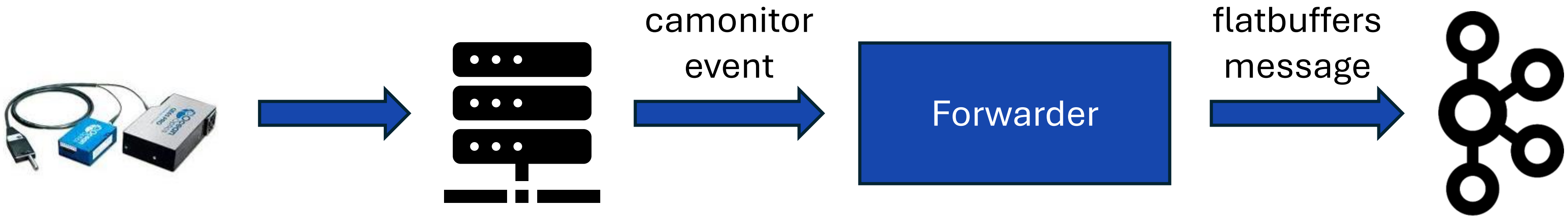
- Images are handled in NDArrays.
  - NDAttributes as metadata.
- ADKafka plugin by ESS.
  - Plugin for AreaDetector.
  - Serialises via the ADAr FlatBuffer to send to Kafka.



Image

Forwarder Summary

- The forwarder:
  - Monitors PVs based on user provided configuration.
  - Takes relevant data and serialises it based on flatbuffer schema.
  - Produces the serialised data as a Kafka message.



Future Work

- f142 schema does not include units as a metadata quantity.
- Creating a unified schema.
- Moving from channel access to PV Access.
- Shifting from ADKafka to Forwarder for images.

Thank you for listening!