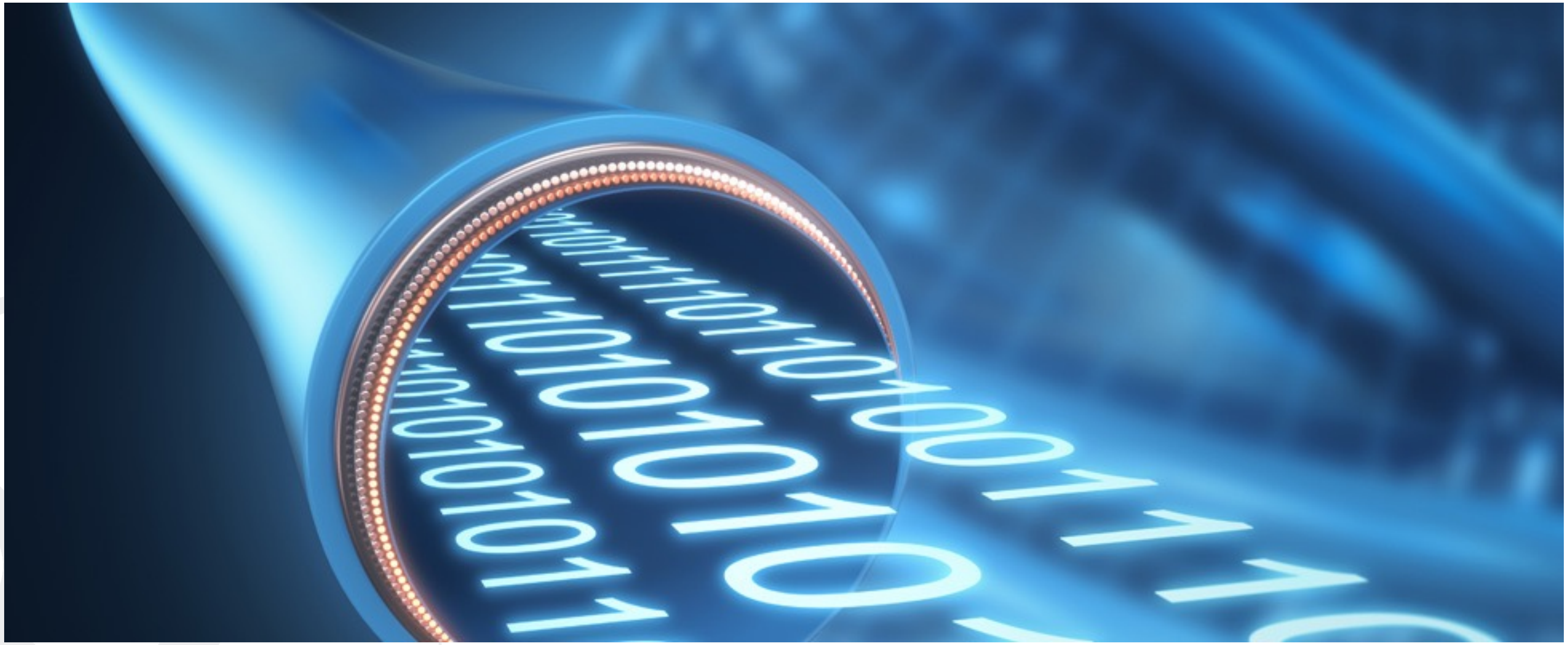
Computing Models that Feature Streaming



Graham Heyes – Jefferson Lab

Friday September 30th, 2022

Topics

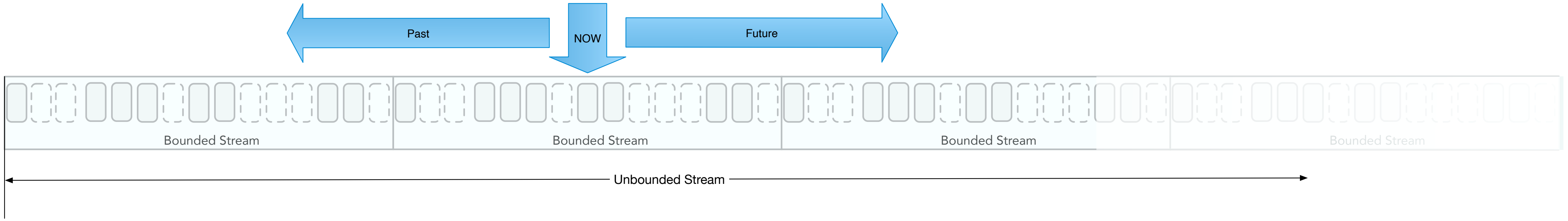
- Definition of streaming data
- The current dominant data processing model
- The need for a new approach
- Data Streaming as a disruptive model
- Science use cases
- Computing models
- Hardware architectures
- Data steering
- Summary

What is streaming data?

- For the purposes of this talk streaming data is:
  - Data that is generated continuously by many data sources, which typically send data records simultaneously in a time ordered sequence
  - Data that needs to be processed sequentially and incrementally on a record-by-record basis or over sliding time windows and used for a wide variety of analytics including correlations, aggregations, filtering, and sampling
  - Data that is typically considered to be “in motion” from a source through various processing stages
  - Data that typically has some time sensitivity or criticality requiring prompt processing
    - To prevent backup as new data arrives
    - To steer the data source
    - To respond to events in real-time
- Streaming data formats typically have a metadata wrapper for each record that contains
  - Data type, data source, position of the record in the stream

Bound an unbound data streams

- A bound data stream is finite stream where each record is well defined and the same format
  - Typically Bound data has a known ending point
  - A data taking run for an experiment is a bound stream of events
  - A YouTube clip is a bound stream of video frames
- Unbound data is unpredictable, infinite, and not always sequential
  - Example continuous monitoring of an instrument looking for rare events
    - There is no concept of a run, the duration is effectively infinite
    - The events occur at random
    - The events can be of randomly varying size. Event A could take longer to acquire or process than event B and so the order in the stream becomes B then A where the order of occurrence was A then B
- An unbound stream of bounded streams is possible (binge watching YouTube)



- How does this data type fit into existing computing models? Is something different needed?

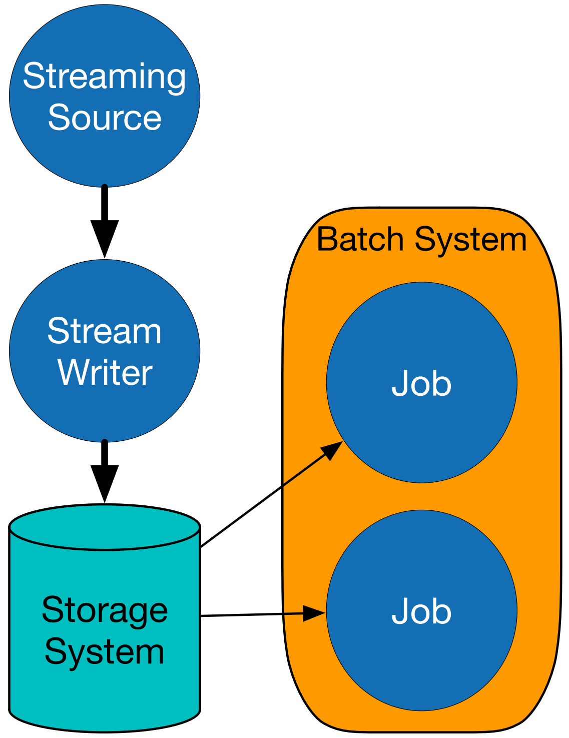
Current dominant data processing model for experimental science

- Data is acquired by online workflows and processed by offline workflows
  - These are usually decoupled with significant time period between acquisition and processing
- Between online and offline data is stored in a hierarchical storage system, typically files in a Posix filesystem
- The batch processing model dominates offline data processing for NP and HEP
- It relies on the need to repetitively process many thousands of files of data
  - Typically, a workflow manager retrieves files from storage and queues up batch jobs to process them
  - Jobs are managed by a batch system (SLURM) and executed when resources become available
- Advantages are
  - Automation of workflows – launch processing of thousands of files
  - Simplicity – each job is relatively independent of its peers
  - Efficient use of resources – jobs run when resources are available.
  - Assigning a range of priorities can keep a cluster busy without impact on high priority jobs

- Does this model fit streaming data? Statement: Batch processing is not a good fit

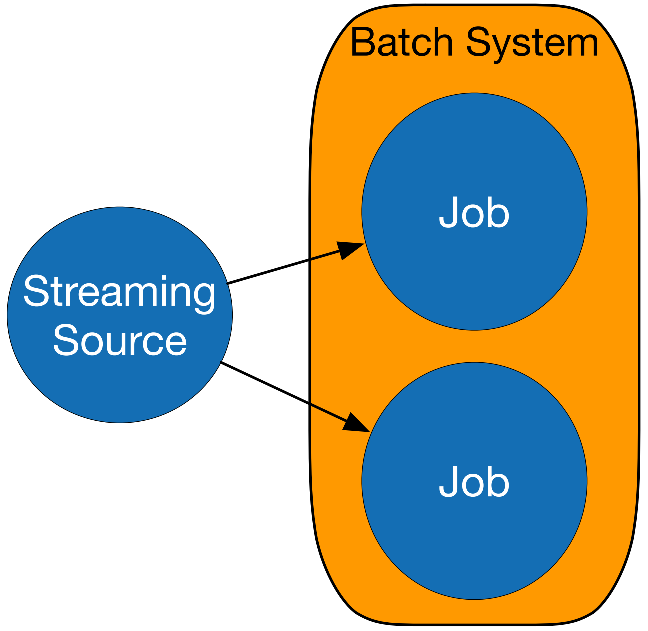
Scenario 1 – Write it to a file and sort it out later

- In this scenario the data streams terminate in a process that writes the records into one or more files
- Simple to set up but hard to manage and optimize if there are many parallel streams
  - One stream per file = parallelism but potentially lots of small files
  - Aggregate streams and write all data to the same file = bottlenecks
  - What does a file represent?
    - Unit of time?
    - Unit of data?
  - How to handle cases where data spans file boundaries?
  - How to prevent the writer from being a bottleneck?
  - File management – what deletes files so that the storage doesn't fill?
  - Batch system starts jobs when resources become available, this is non-deterministic.
    - Hard to implement real-time or even near-real-time processing
- This is a workable solution but making it work loses many benefits of streaming data



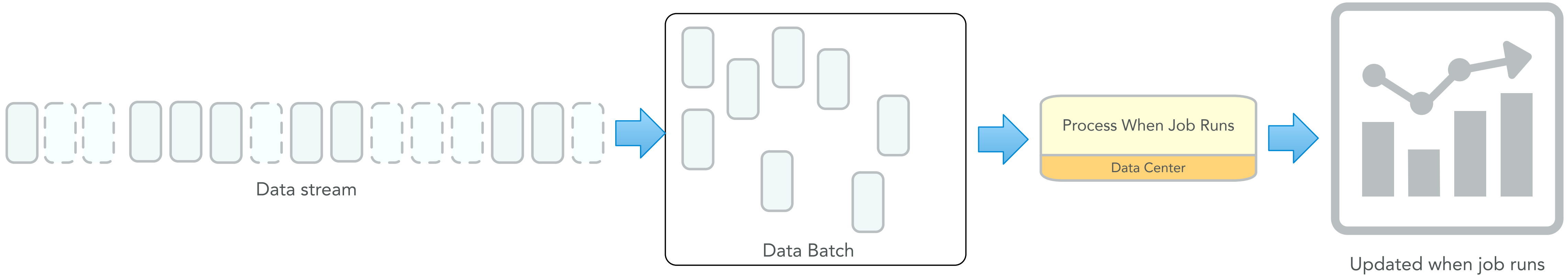
Scenario 2 – Start batch jobs that connect to streaming data sources

- In this scenario the data streams connect to jobs in the batch system directly. Streaming data all the way
  - The batch system starts the jobs when resources are available
  - These jobs must remain allocated for the duration of the experiment
  - Only works if the data is streaming over a network – otherwise need to allocate specific nodes with custom hardware
  - Needs an external orchestrator
    - Which nodes are running what?
    - Which data goes where?
    - Starting, stopping, monitoring, recovery
  - Unless running on a dedicated node the job has no control over what else is running on the node
    - Reliability?
    - Performance?
- This is a viable solution, but making it work requires sidelining the batch system except at initial startup

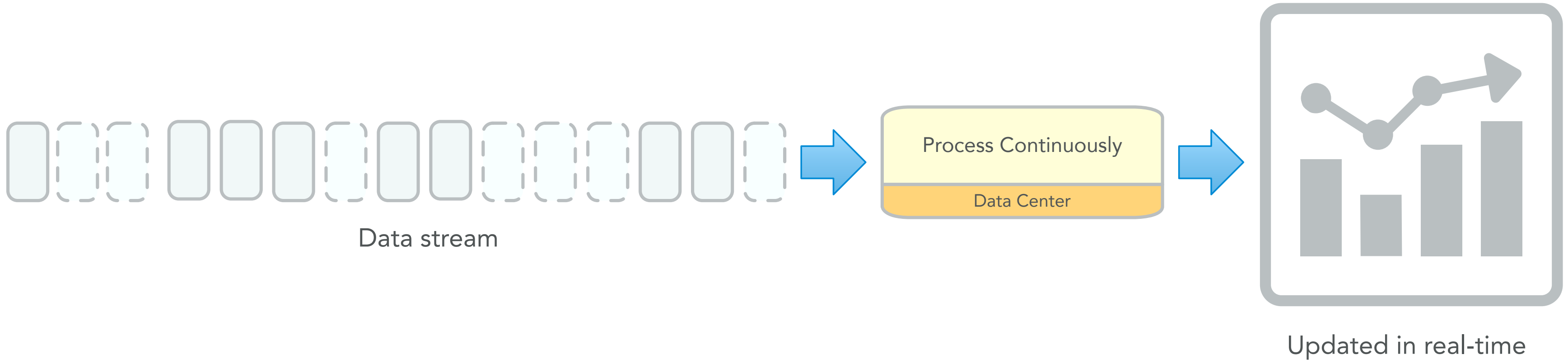


Jefferson Lab

Comparison of batch processing and stream processing



Batch jobs only run when resources are available and, typically limited duration. Possible contention between jobs on the same node. Data is no longer streaming but sorted into batches. Results ready when job ends.



Batch system only used to start the processing. Must allocate resources for duration of experiment. Ideally data flow can't stop so guaranteed service quality is needed. Results update in real-time.

What models are a good fit?

Jefferson Lab

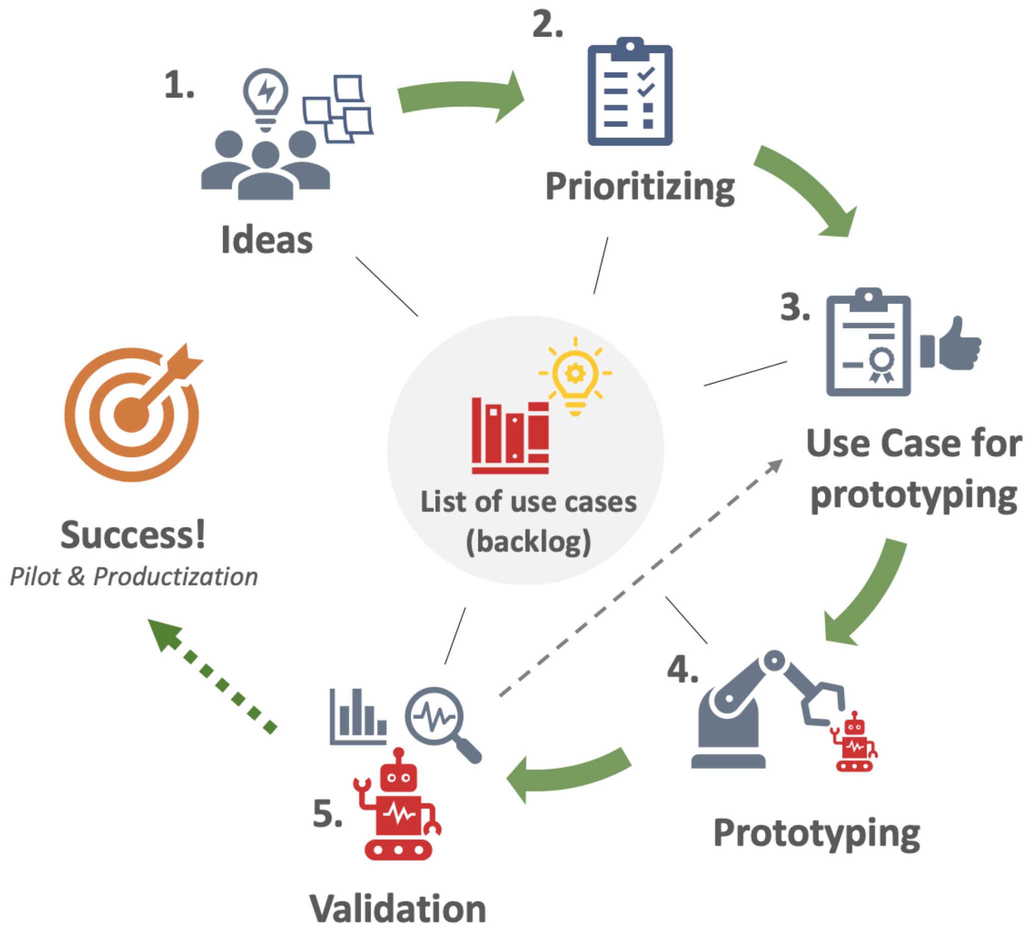
Data Streaming as a disruptive model

- Instead of trying to force streaming data processing to conform to a pre-existing model, like batch processing, treat it as a disruptive model in its own right.
  - Approach followed by Amazon, Google, Netflix, etc.
- Key components:
  - *Flexible data integration*: The ability to transport data in the format required, regardless of whether the data is structured or semi structured, direct or customized, static or changing
  - *Data inference*: Make it easy to map data and control how it is processed
  - *Code engines*: The ability to embed code to enrich and cleanse data, create alerts, and detect anomalies in stream in real time
  - *Live monitoring*: Real-time visibility into data streams for monitoring behavior, identifying potential discrepancies, and debugging data records
  - *Pipeline transparency*: Provides continuous views of data in motion and notifications that allow users to view incoming events, monitor throughput and latency, and identify errors in real time.
  - *Agility:* The ability to dynamically expand or contract and migrate allocated resources.
  - *Support for heterogeneity:* The ability to allocate a tailored resource unconstrained by predefined hardware choices
- *How does it apply to science?*

Science use cases

Any good computing model is driven by use cases that present challenges that it must solve. Here are four to consider

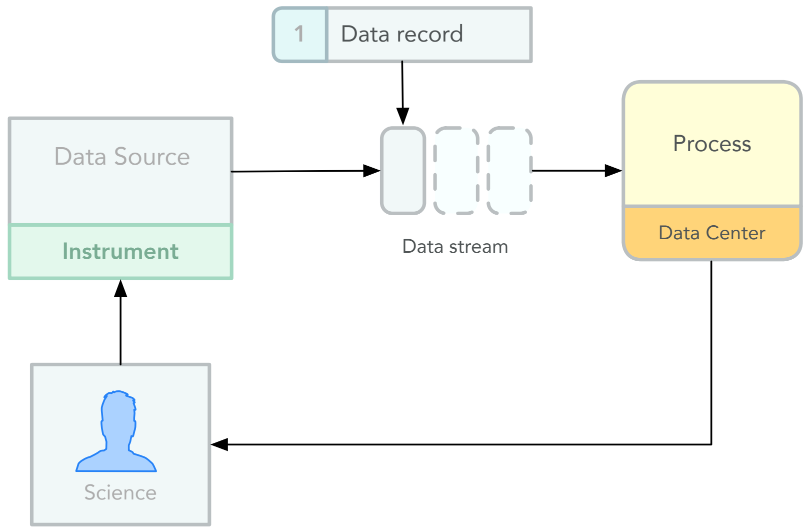
1. Time critical workflows
2. Data driven workloads
3. Distributed data sources
4. Complementary data sources



Jefferson Lab

1) Time Critical Workflows

- Across a range of science programs, experiments follow time-critical use patterns
  - Success of the science depends upon data being processed within a specified time after acquisition
  - Examples:
    - Real time data filtering
    - Processing a data sample to inform instrument calibration (example: drift chamber stabilization)
    - Data quality monitoring
    - Real time processing to
      - Reduce data archive requirements – NP Level-3 trigger
      - Steer experiments – development of new catalysts
      - Provide input to AI/ML, inference engines, surrogate models
- The key features to guarantee success are
  - timing of data delivery to compute resources
  - timing and continuity of compute resource availability
    - Uptime and quality of service
  - timing of real-time feedback delivery

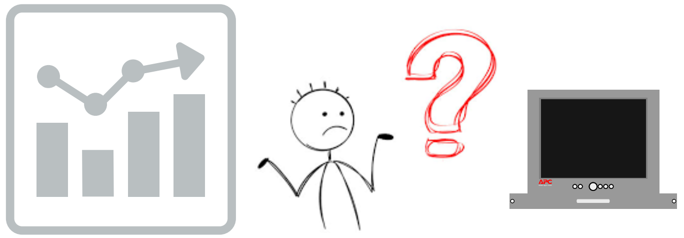


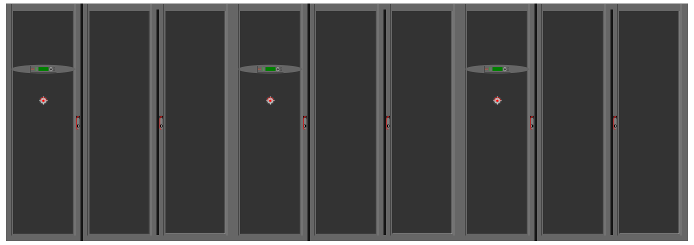
2) Data driven workload

- The scale of computing resource required for an experiment is driven by data rate and complexity
  - NP and HEP:
    - Processing time is a function of detector and event complexity as well as the science being studied
    - Scale of compute resource needed can vary depending on data taking conditions
      - It is frequently hard to guess in advance (overestimates are as bad as underestimates)
  - Light Sources:
    - Data complexity and rate varies between beamlines and, for individual beamlines, from sample to sample
    - Relatively simple sensors with high sample rates
      - Low complexity simplifies processing, but data rate is problematic
    - Complex sensors like high-rate cameras
      - Data is complex AND data rates are high

- Key to success is an adaptive system that can expand or contract as the workload fluctuates.

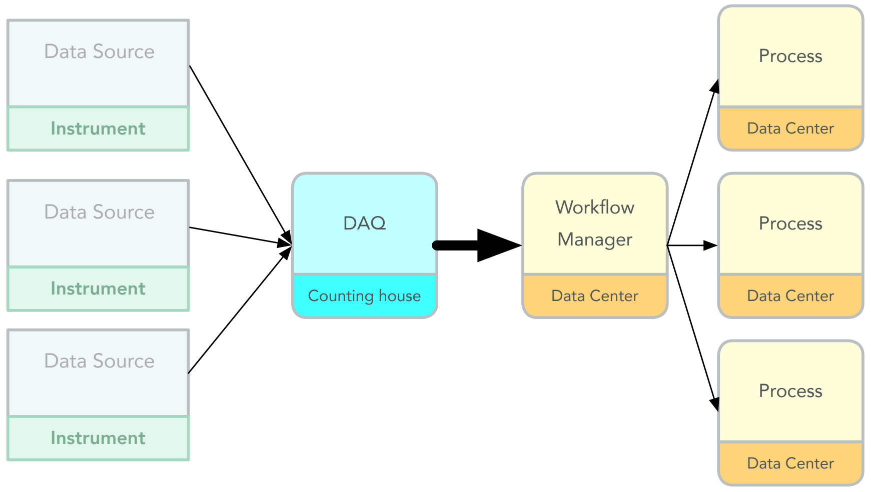
Do I need a laptop or Supercomputer

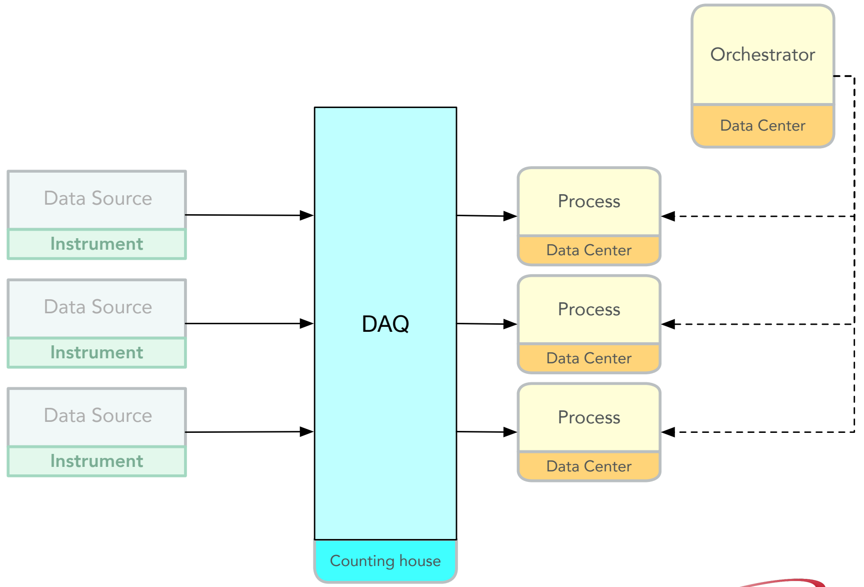




3) Distributed data sources – NP and HEP

- Very familiar to NP and HEP – detectors are composed of sub-detectors, that in turn are groupings of individual detector channels
  - Data sources are spread over a volume of space
  - Also fold in data from other sources, accelerator, beamline, etc.
- Typically, this is dealt with online by an “Event Builder” that combines data records from various detectors for the same interaction into an “event”
- Streaming data in parallel to a computing resource defers event building, could be:
  - Just before reconstruction, or
  - After cluster and track level reconstruction, or
  - No event building at all but infer interactions from patterns of tracks.
- Key takeaway is that streaming data processing is a natural fit, it’s in the definition of streaming data

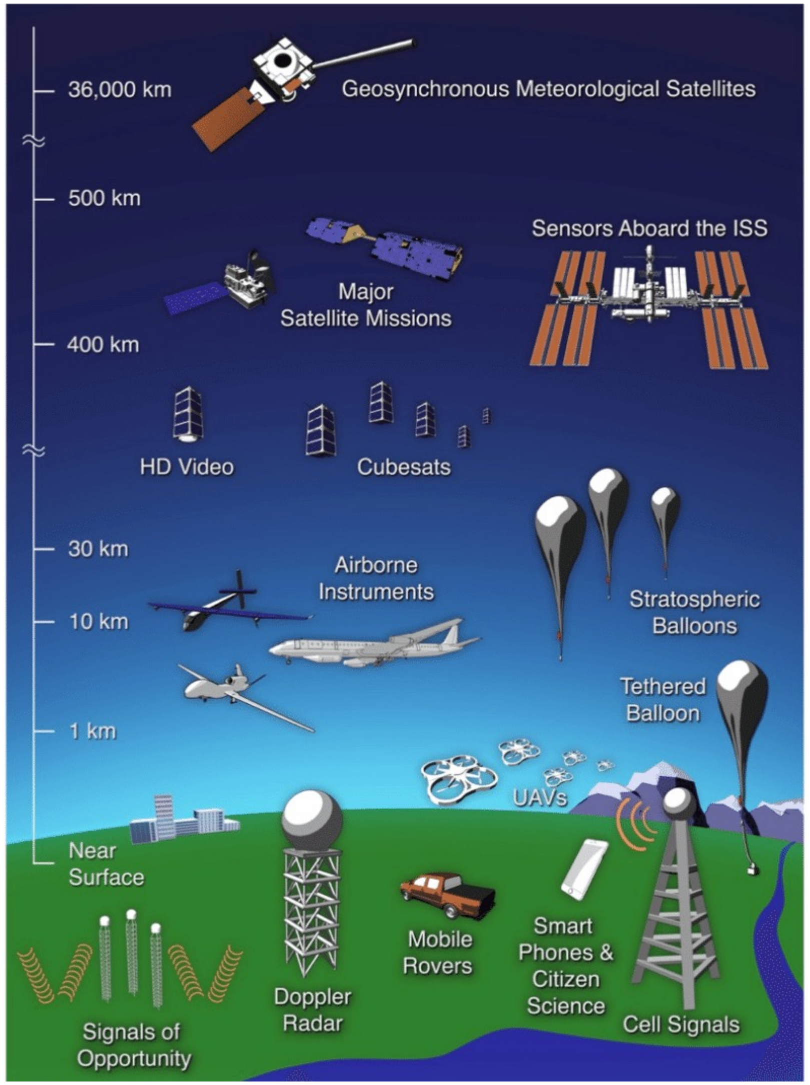




Jefferson Lab

Complementary data sources

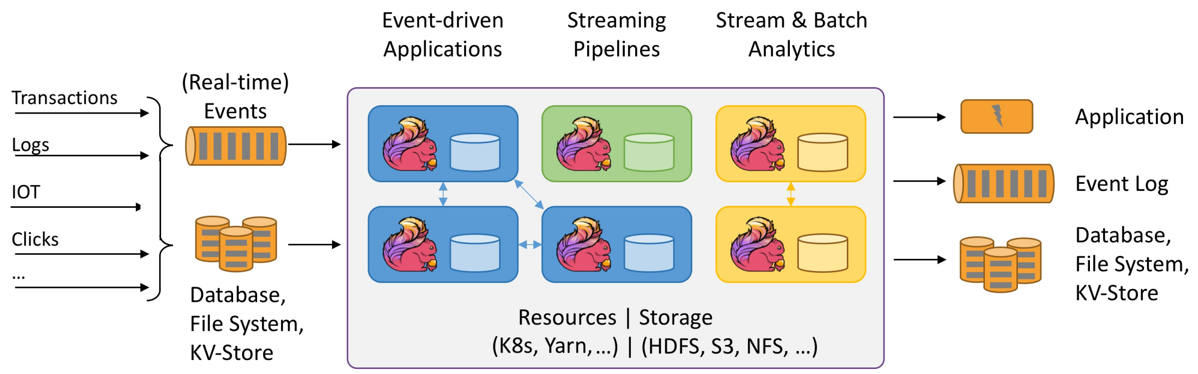
- In some fields observational or experimental data is combined from completely different but complementary sources
  - NASA and NOAA combine satellite data with ground radar, wind gauges, temperature sensors
  - Astronomy combines radio, optical and IR images
  - Light sources use data from one beamline to calculate configuration settings to simultaneously study the same material in another beamline
- There is also interest in running simulation in real-time using models informed by complementary measurements
  - Example, scan a sample to determine shape and structure, simulate interaction with beam using real shape and structure, compare with experiment in real time to inform next run



Jefferson Lab

Frameworks implementing streaming data models

- Streaming data is a hot topic – commercial and open-source solutions exist
  - Amazon
    - Kenesis - ingest, buffer, and process streaming data in real-time
  - Apache
    - Flume - simple and flexible architecture based on streaming data flows, aimed at log data
    - Kafka - open-source distributed event streaming platform
    - Flink - A framework and distributed processing engines for stateful computations over *unbounded and bounded* data streams.
    - Apex - batch and stream processing using Hadoop's data-in-motion architecture by YARN.
  - ASCR – several initiatives, for example research in streaming data reduction
  - NP tailored solutions such as ERSAP at Jlab and HEP DAQ systems
- Question: Innovate or integrate?
  - A little bit of both
  - Some third-party products are maturing
  - Some likely never a good fit
  - Perfection is the enemy of good enough


Event-driven Applications
Streaming Pipelines
Stream & Batch Analytics
Transactions
Logs
IOT
Clicks
...
(Real-time) Events
Database, File System, KV-Store
Resources | Storage (K8s, Yarn, ...) | (HDFS, S3, NFS, ...)
Application
Event Log
Database, File System, KV-Store

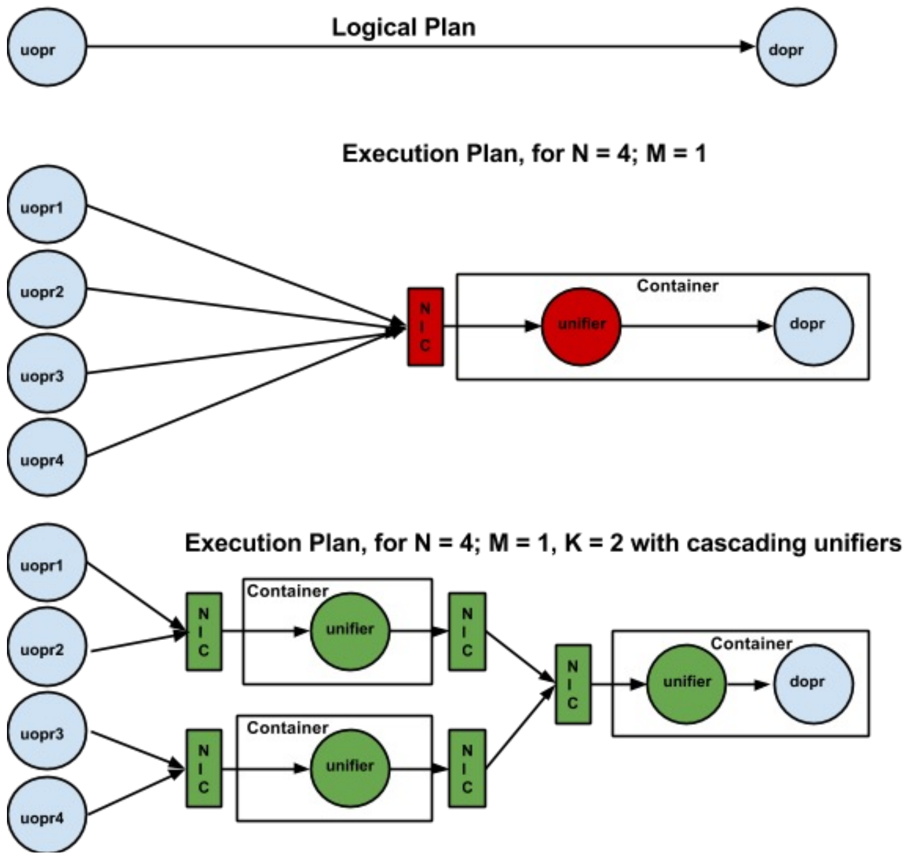
Processing models

- Most of the solutions are fundamentally similar and based on a distributed parallel model
  - Event driven engines networked by data pipelines
  - Data between engines is “in flight” as much as possible
  - Data lakes – unstructured “pools” of data
    - Provide temporary buffering of data to rate match application components
    - Time sync streaming data to allow cross stream data queries
- Stream processing engines are runtime libraries which help developers write code to process streaming data, without dealing with lower-level streaming mechanics.
- There are two major types of processing engines.
  - Compositional Engines
  - Managed Declarative Engines

Compositional Engines

- In compositional stream processing engines, developers define the Directed Acyclic Graph (DAG) in advance and then process the data. This may simplify code, but also means developers need to plan their architecture carefully to avoid inefficient processing.
- **Challenges:** Compositional stream processing are considered the "first generation" of stream processing and can be complex and difficult to manage.
- **Examples:** Compositional engines include Samza, Apex, and Apache Storm.

- Developer builds the application from a kit of parts according to a plan

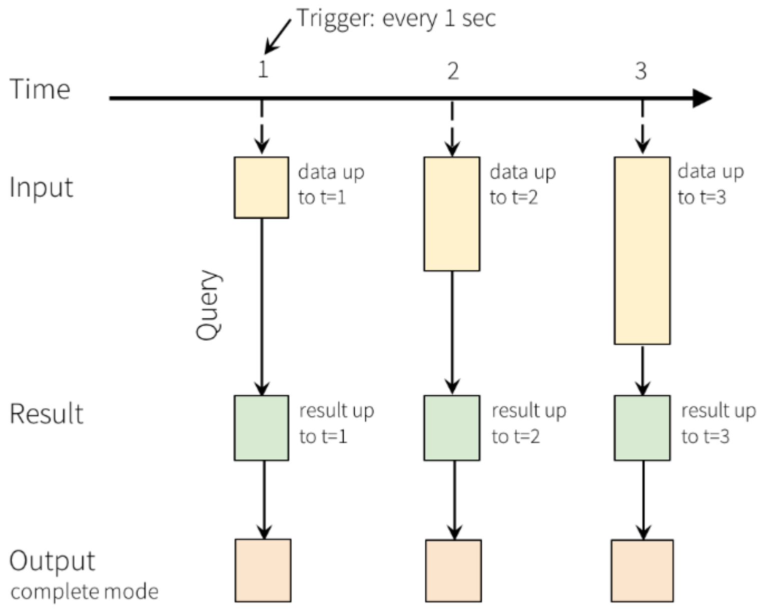


Declarative Engines

- Developers use declarative engines to chain stream processing functions. The engine calculates the DAG as it ingests the data. Developers can specify the DAG explicitly in their code, and the engine optimizes it on the fly.
- **Challenges:** While declarative engines are easier to manage, and have readily-available managed service options, they still require major investments in data engineering to set up the data pipeline, from source to eventual storage and analysis.
- **Examples:** Declarative engines include Apache Spark Streaming and Flink.

- The engine assembles the network based on incoming data or optimizes an existing net

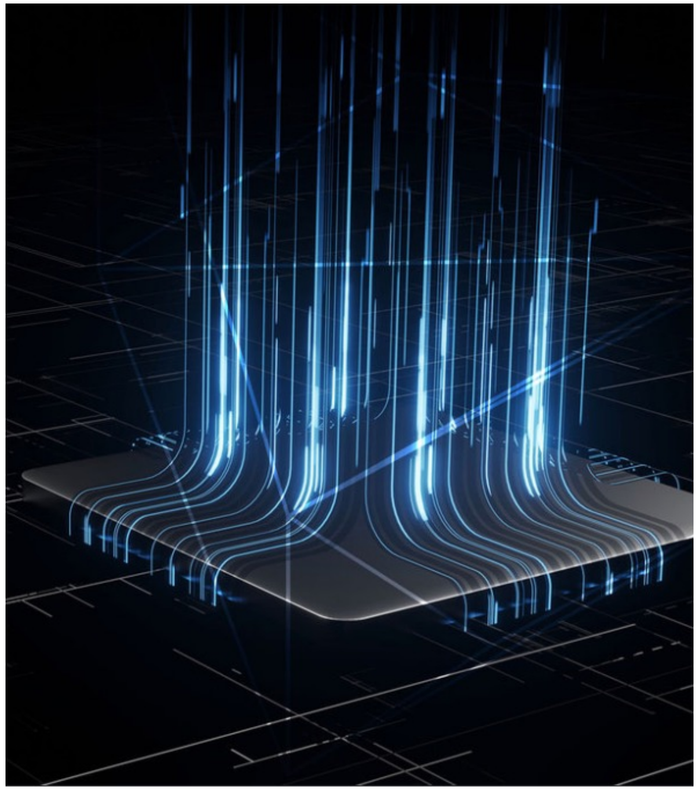
```
spark
  .readStream
  .select($"value".cast("string").alias("jsonData"))
  .select(from_json($"jsonData",jsonSchema).alias("payload"))
  .writeStream
  .trigger("1 seconds")
  .start()
```



Streaming optimized hardware architecture

- The software architectures discussed in the last few slides allow the creation of managed streaming applications
  - Developers provide code to plug into event driven engines
  - Engines are assembled into an application using network graphs
  - A management layer optimizes data flow and performance on the fly

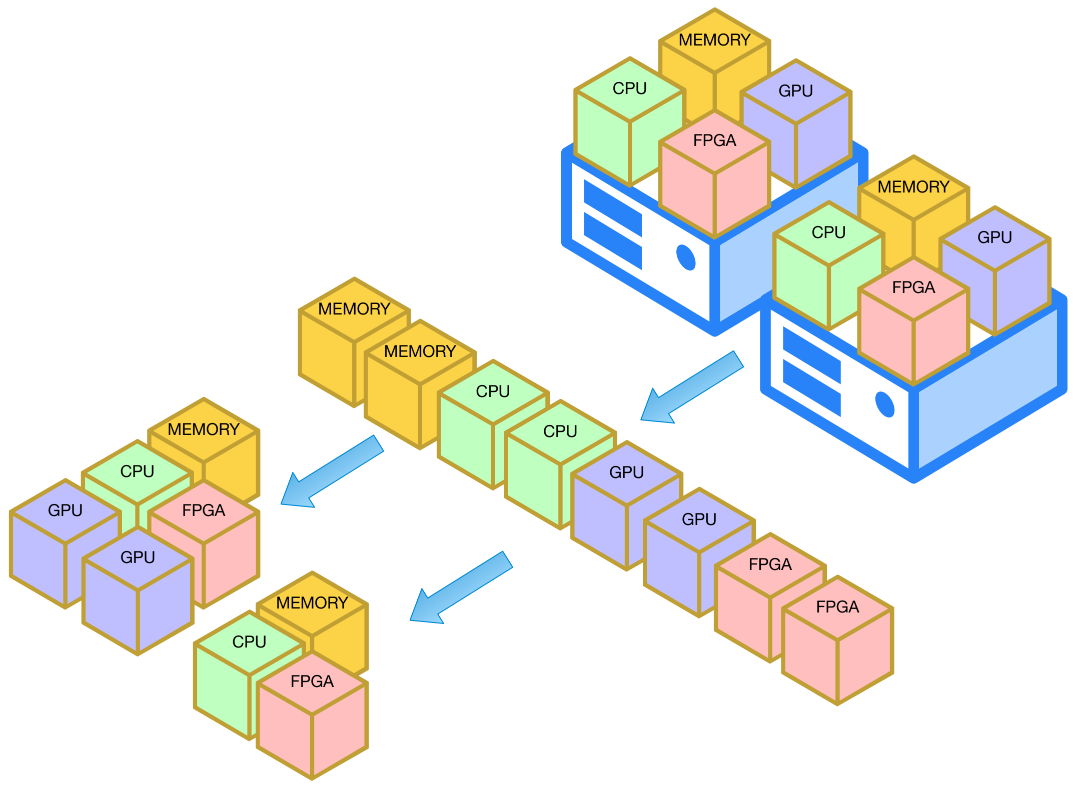
- What sort of hardware architecture would optimally support these applications?
  - Provide guaranteed uptime, performance and service quality
  - Support
    - Data flow optimization on the fly
    - Applications of (somewhat) arbitrary scale on heterogenous hardware
    - Integration of AI/ML and other data science techniques
    - Novel technologies (adapt to future needs)



Disaggregation and reassembly

- Imagine a computing resource with all the right parts for a particular streaming application but not necessarily in the right configuration
- What if we could disaggregate individual nodes into a network of parts and reassemble something that is a better match to our application?
- Technology roadmaps from silicon vendors show inter CPU and GPU networking is either coming or already here (NVIDIA BlueField)
- This would allow adaptive tailoring of hardware architecture to match a streaming application

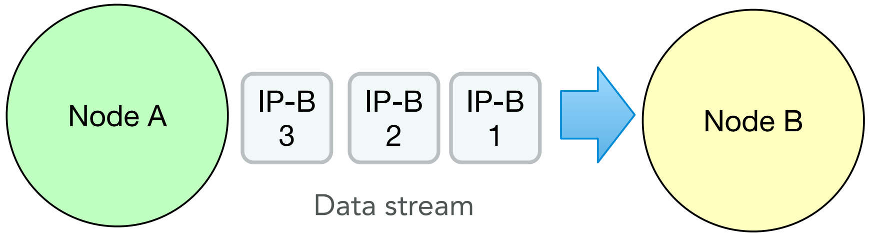
- How do you reconfigure hardware in a running application?



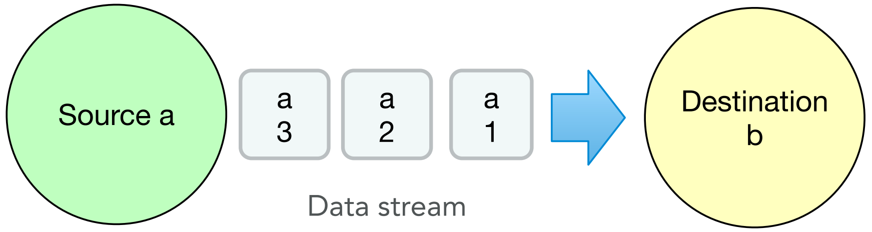
Jefferson Lab

Data flow steering

- An adaptive dynamic architecture must be able to:
  - Reroute data that is in flight when the system architecture reconfigures
  - Allow remote data sources to be agnostic of the destination hardware configuration
  - Provide robustness while preserving agility and performance
- IP based networking identifies hardware endpoints via IP addresses. Data is routed according to *where* it needs to go
- A streaming data application routes data according to *what kind* of data it is and *where it came from*.
- A low-level network protocol to implement this does not exist but can be emulated



- Packets are sent to the address of B
- A and B are IP addresses

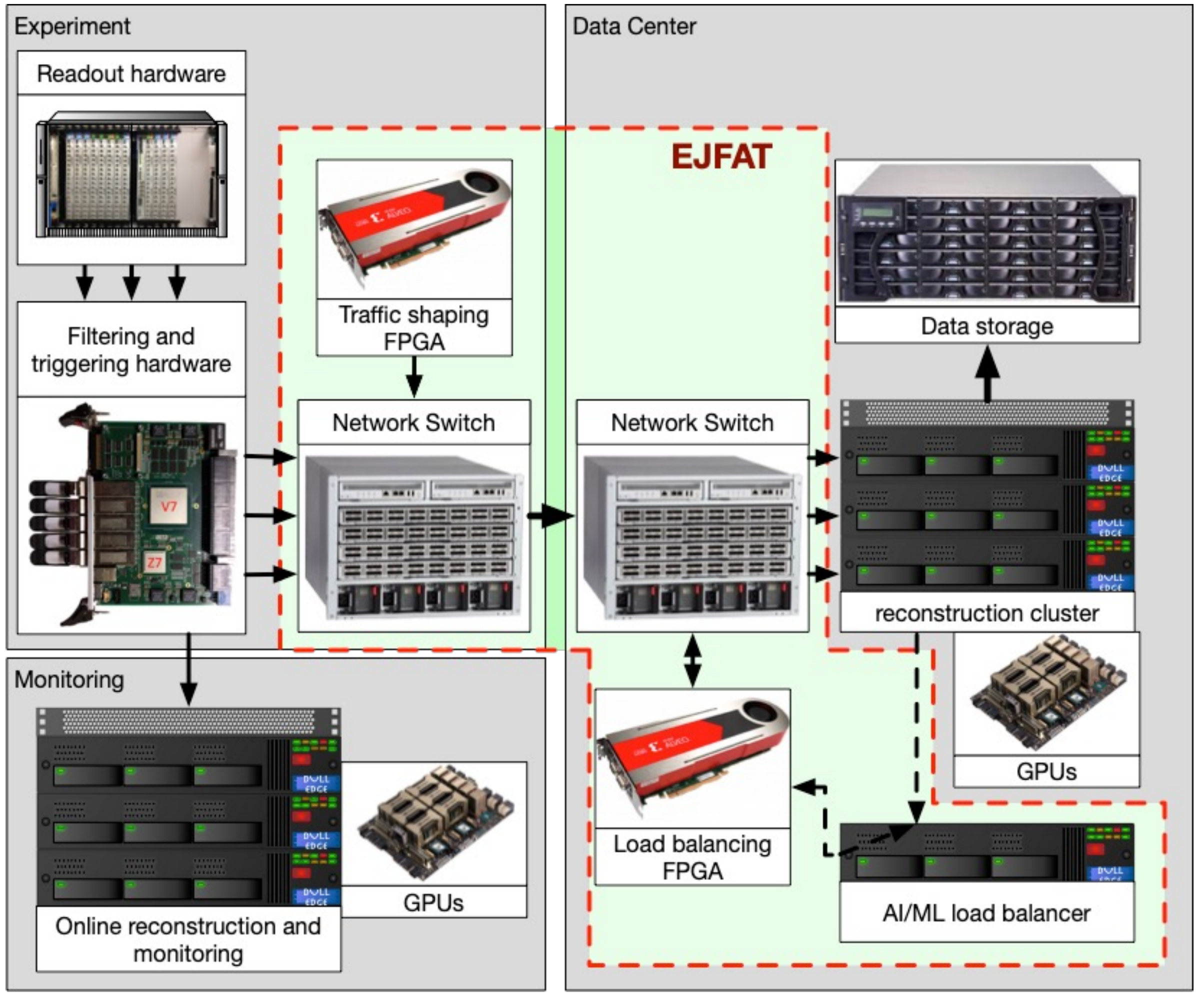


- Packets are routed to "b" based-on source (a), data type and, (optionally) sequence number
- "a" and "b" are logical identifiers not IP addresses

Jefferson Lab

Prototype dynamic steering of streaming data

- The Esnet Jlab FPGA Accelerated Transport project is an example
  - Streaming data format contains metadata describing the data
  - Using standard IP based network all traffic is directed to an FPGA device
  - Firmware modifies IP packet headers to reroute the data based on what kind of data it is and what kind of destination it should stream to
- Initial implementation has static translation tables to implement data distribution schemes
  - Round robin, sorting by time or source
- Goal is dynamic intelligent steering



Jefferson Lab

Summary

- In this talk topics were outlined, and questions asked:
  - Definition of streaming data
    - Does it fit into existing computing models? Is something different needed?
  - The dominant data processing model for experimental science
    - Does that model fit streaming data?
  - Use of the batch model to process streaming data
    - Workable but not a good fit
    - What models are a good fit?
  - Data Streaming as a disruptive model
    - *How does it apply to science?*
    - *Example use cases*
  - Frameworks implementing streaming data models
  - Streaming optimized hardware architecture
    - How do you reconfigure hardware in a running application?
    - Data flow steering

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

- Processing streaming data is a "hot topic" in industry, computer science research, as well as current and future trends in nuclear and high energy physics

- As with all "hot topics" the right balance must be found between innovation and implementation – there's a lot out there when you start to look!

Jefferson Lab