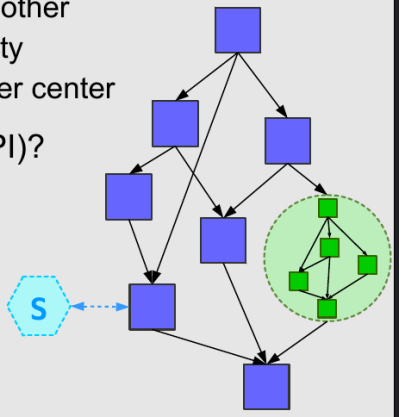
Frameworks and Infrastructure

Mohammand Al-Turany, Paolo Calafiura, Marco Clemencic

Heterogeneous Computing

- This is not a talk (just) about GPUs
- If we can figure out a generic mechanism to keep a GPU busy using outside data, the world opens up
  - means we can potentially offload ANY task from one CPU to another
  - "tasks" are probably groups of Algorithms for greater data locality
  - CPUs don't have to be on same node, or even in same computer center
- Can we turn all data communications into messages (*à la* MPI)?
  - less work if all nodes are configured the same way
    - can turn Event Store / DataHandles into message endpoints
    - ATLAS EventService on steroids
    - normal access to "services"
  - more work if truly heterogeneous
    - have to encapsulate all data needed for a transaction
  - probably better suited to specific problems that don't need "extra" data
- this is not a new concept: ART and FairMQ

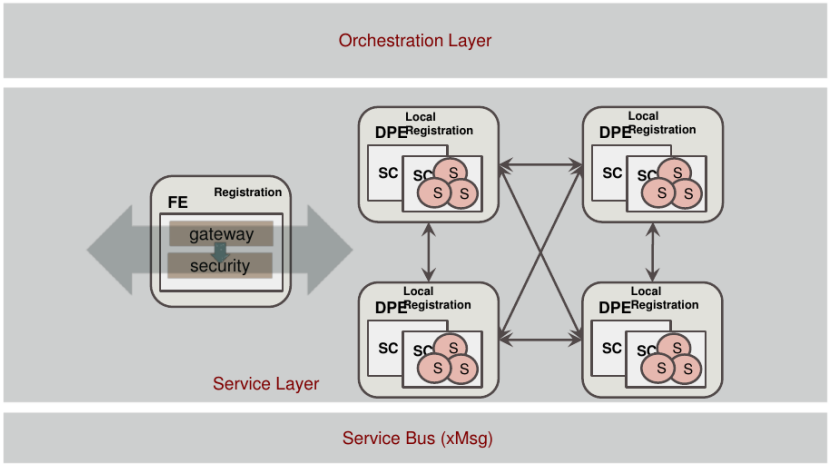


Can we envisage a common simulator?

Simulator

- Need to create a simulator of the system to understand how all the parts fit together
  - ultimate metric is **event throughput** (per bitcoin)
- lots of tunable parameters (let's just think about GPUs for now):
  - hardware configuration (GPU/node, GPU model, interlink, *etc*)
  - fraction of event processing time that can be offloaded to GPU
  - how much data needs to be sent to GPU to make it useful
  - given a certain event size (lumi dependent), how many events do we need to process on one node to keep GPU full
  - what's the latency of sending data to GPU / getting it back
  - if data comes from outside the node, what's the penalty for marshalling it
  - how many inbound connections are needed to saturate GPU
  - after a CPU has offloaded a task to a GPU, can it continue on other work?
    - more work in same event?
    - start a new event?
    - is it worth checkpointing the event, and reloading it when GPU data returns?

CLARA architecture



Different paradigm:

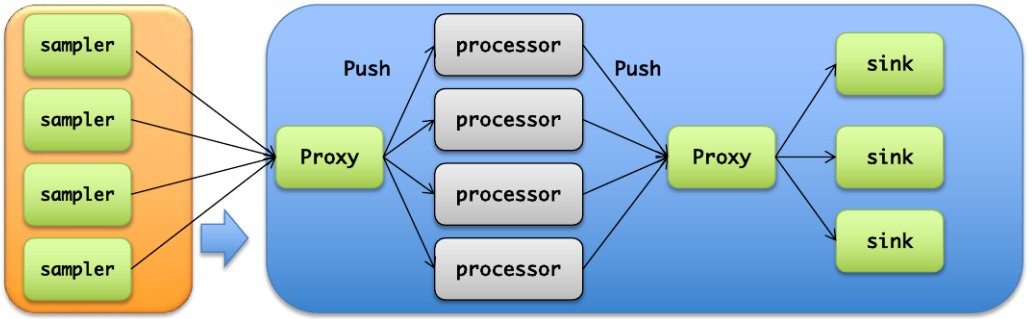
- actors
- data flow
- message passing

What about a common framework/toolkit for message passing and data transfer?

FairMQ

The Data Processing Component of ALFA

- Multi-process concept (specialized **devices**)
- Data-flow model: Message queues for data exchange, technology agnostic



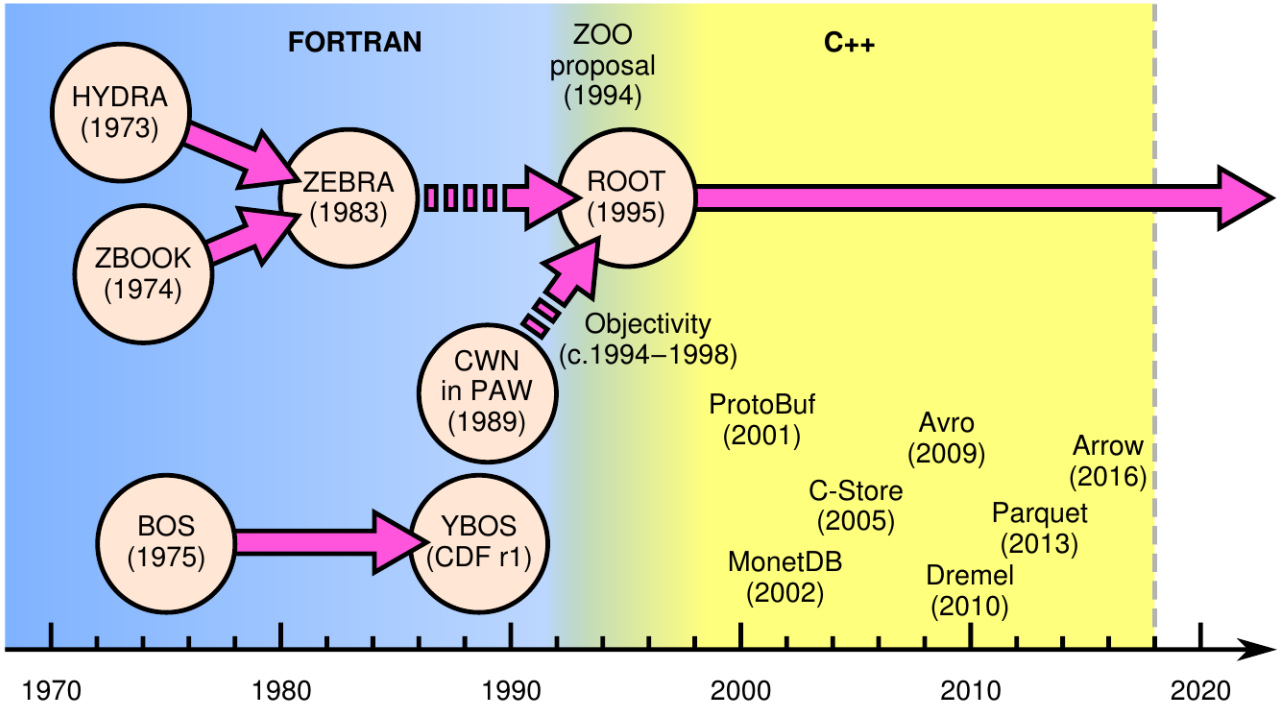
Design Goals

- Scalability, Maintainability, Reliability
  - efficient use of multi-core architectures
- Reusable with common data processing components
  - Reduce cost of new developments, agile development

ROOT covers most of our archiving needs, but not documented.

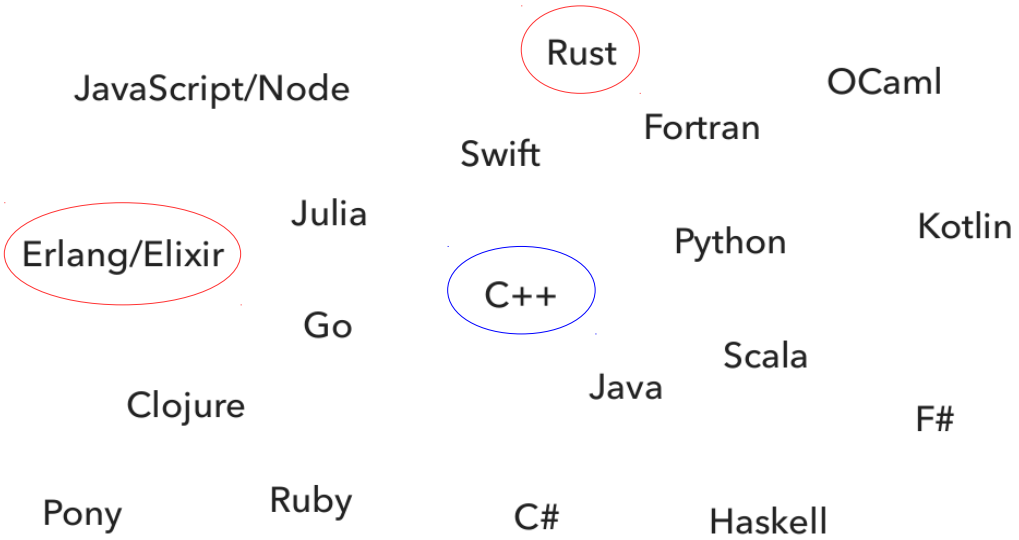
Other solutions are much better for interprocess communication.

45 years of serialization formats in (and out of) HEP



PROGRAMMING LANGUAGES

CURRENT LANDSCAPE



C++ is a powerful generic language.

Erlang **IS** actors and messages.

Rust is very strict
(memory leaks are impossible).

CWP Data Processing Frameworks

A Guided Discussion

The panorama doesn’t seem to settle…

E.g. we focused on tasks and TBB, but now we also have data flow and actors.

A framework dedicated meet up proposed for CHEP and/or later in the year.

3 months into 2018 - Where are we?

- Overall goals are papers, workshop reports, analysis results, and software architectures.
- Set direction for further R&D
- Defining a path towards a new production product development
- Integration of results into existing software components.

Thanks to
all speakers and attendees
for the discussions