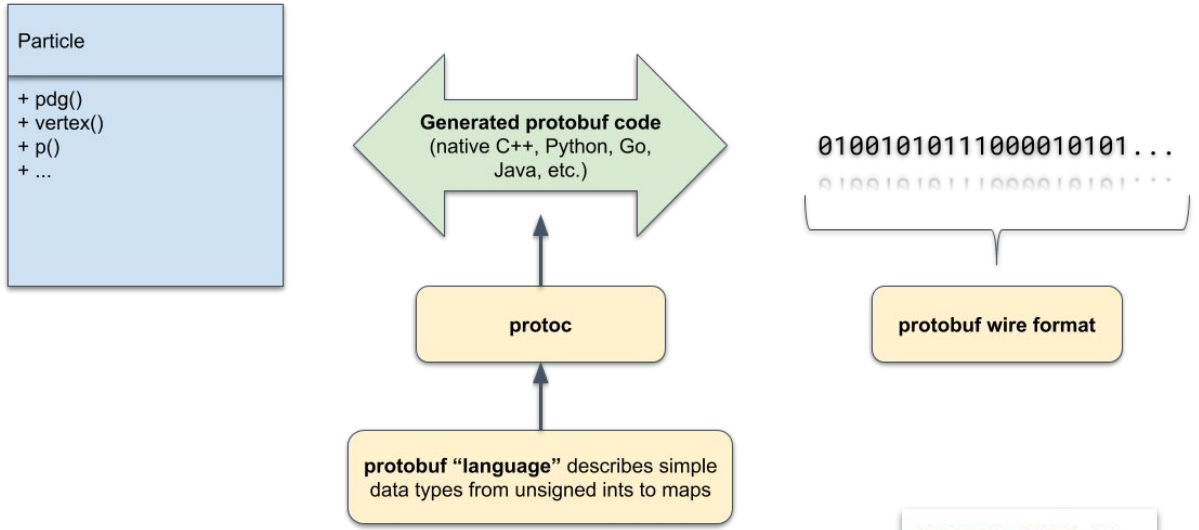
Protocol buffers encoding FPGA acceleration

Rafael Acuña MS EE (Arizona State University)
EIC Streaming Readout Consortium
Streaming Readout VI

Protocol Buffers

- (De-)Serializable representations of data
- Protobuf language describes the language-agnostic data model
- Protobuf compiler (protoc) generates code for language-specific applications
- Uses field identifiers: great for describing sparse data
- **Exhibits variable-length encoding for integers (lossless, localized compression)**



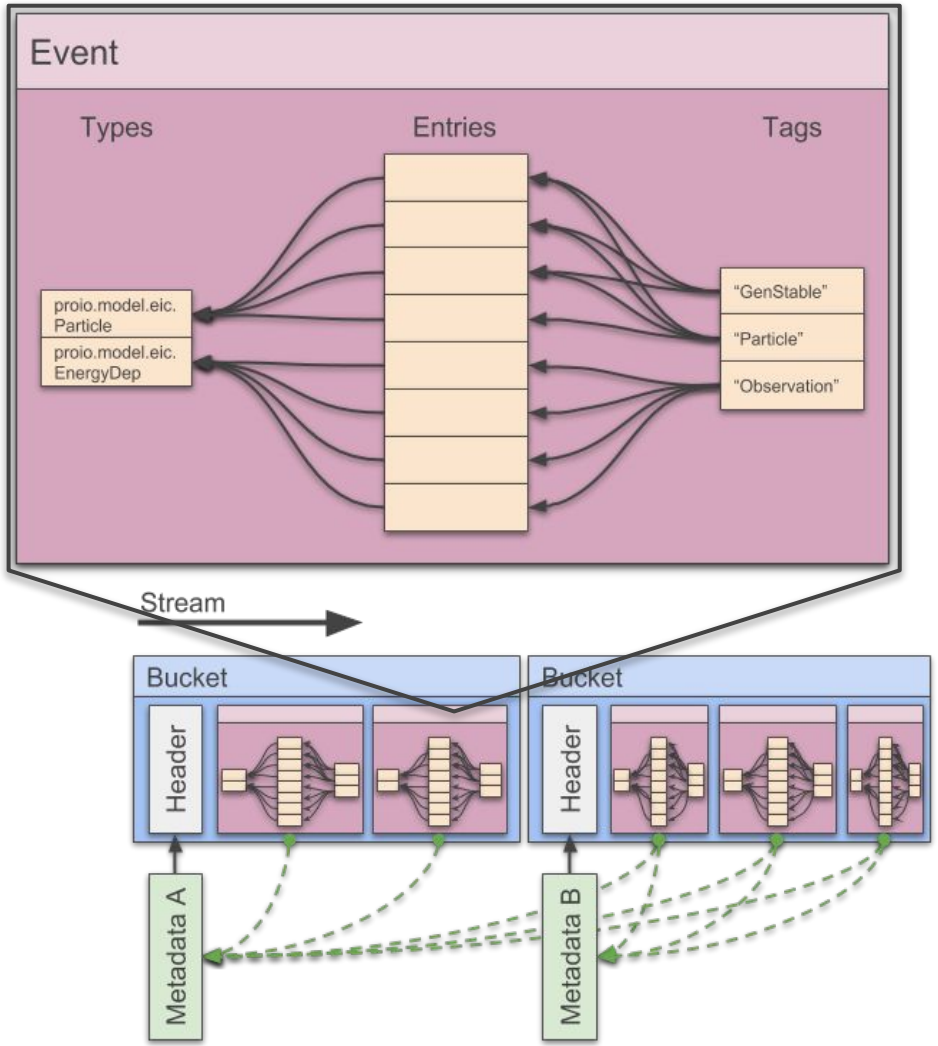

```
message XYZTL {
    sint64 x = 1;
    sint64 y = 2;
    sint64 z = 3;
    sint64 t = 4;
}
```

example protobuf code

ProIO

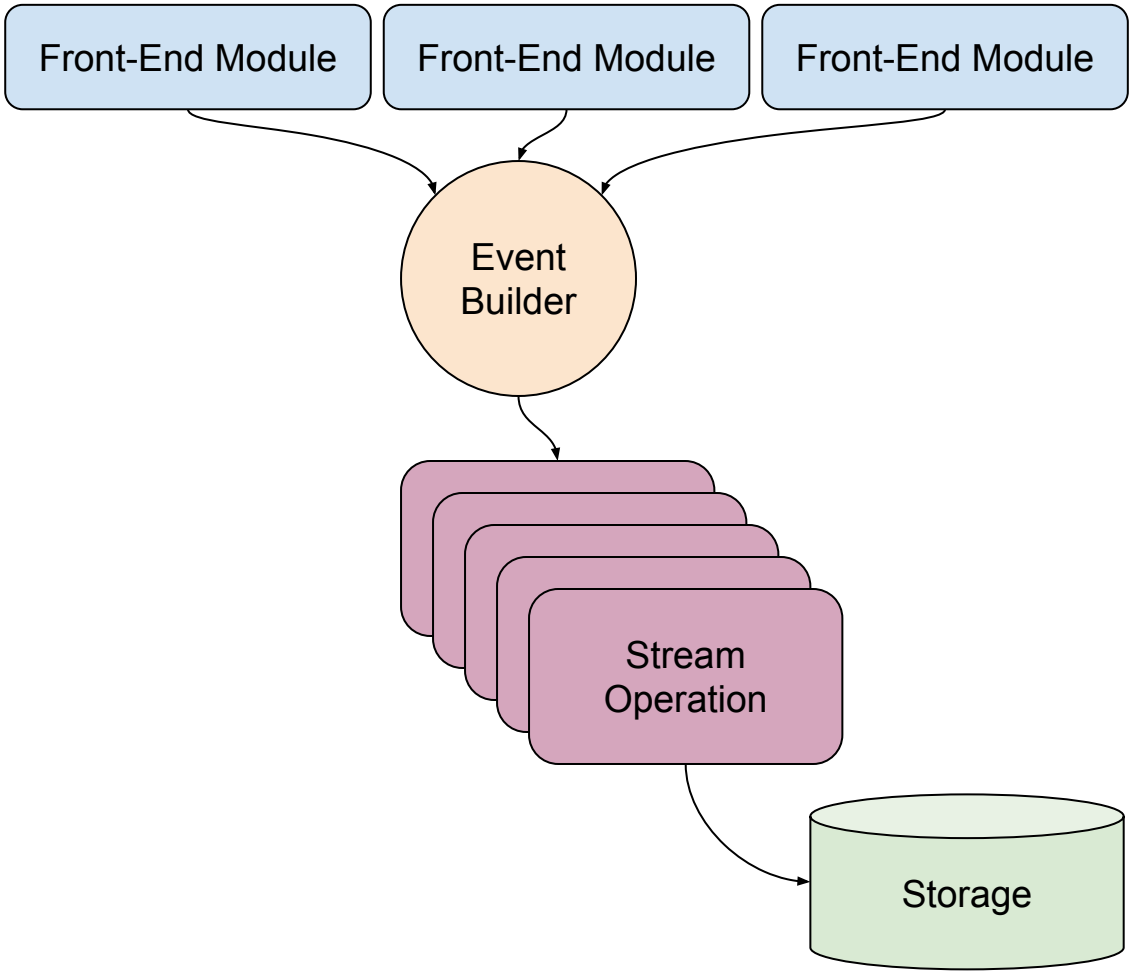
- Wraps protobuf messages in a data stream, providing
  - event structure
  - self description
  - additional general-purpose compression (such as LZ4)
  - lazy deserialization
- Self description means that information about the source protobuf data model is provided by the stream

DOI 10.1016/j.cpc.2019.03.018



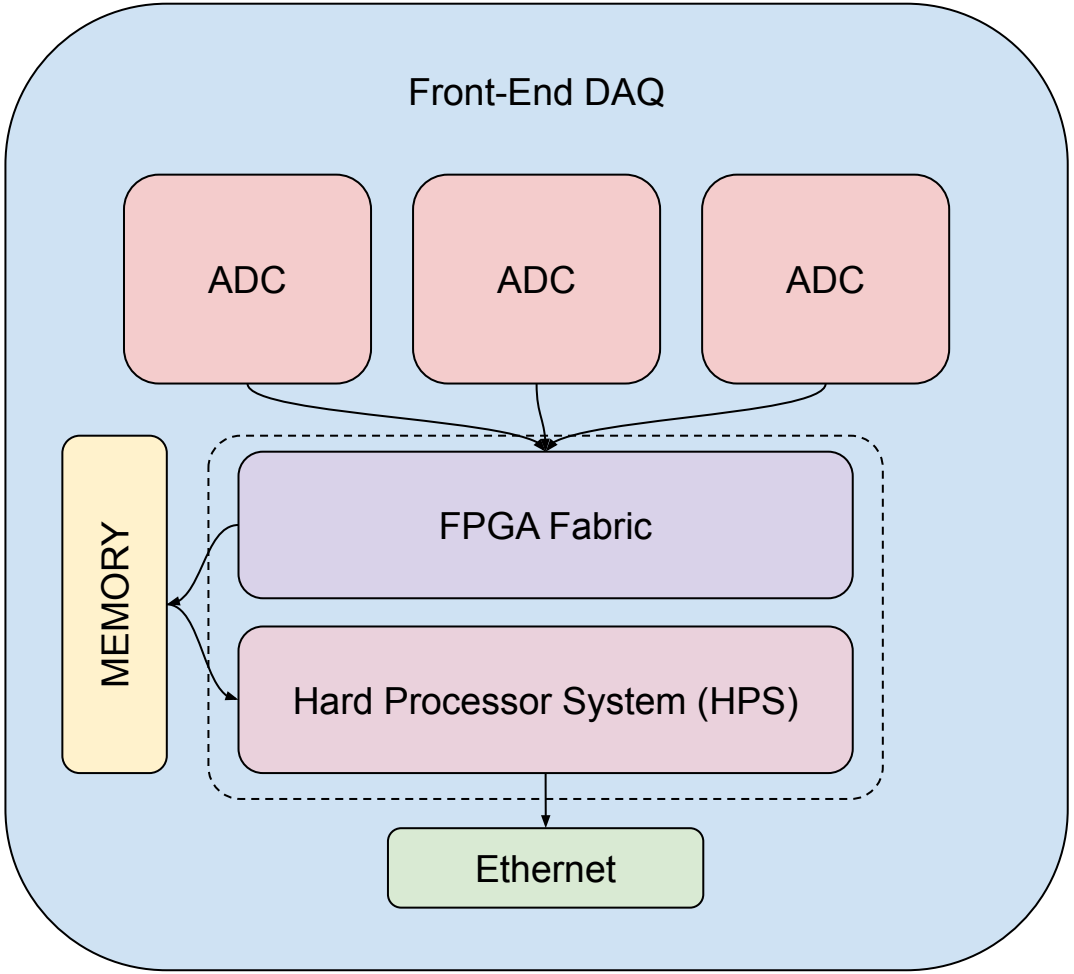
ProIO Usage in Streaming Readout

- ProIO is “event-oriented”, so mostly useful once events are constructed from various detector data
- Read-modify-write of events can be efficient due to “lazy decoding”
- Protobuf encoding provides substantial compression of raw data (factor of 2 or more for common scenarios)
- Stream encoding can begin at the front-end electronics



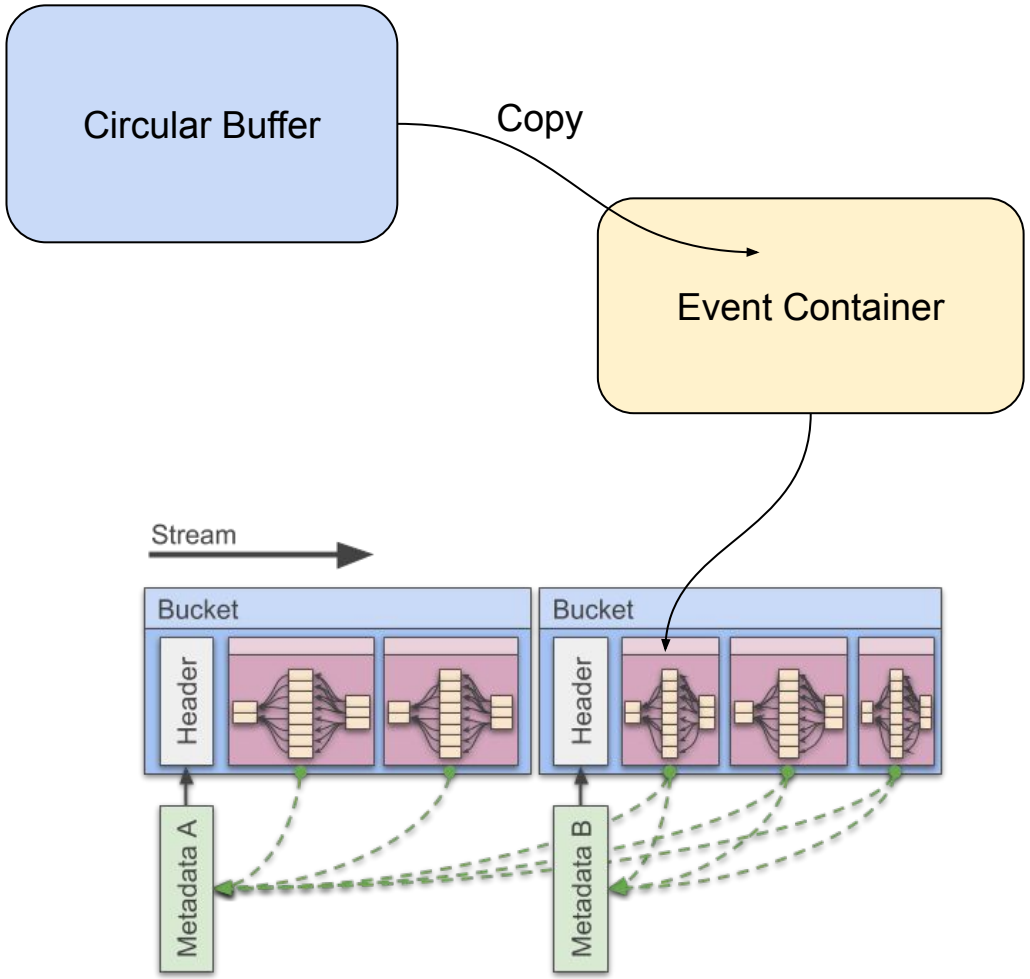
Our Small Scale Use Case

- Application: current-mode gas detector for proton therapy beam profiling
- FPGA acquires waveform samples from ADCs, and passes to circular memory buffer viewed by hard processor system (HPS)
- Software on HPS establishes connection to data aggregation server, and writes ProIO stream
- **Challenge: protobuf variable integer encoding is almost as CPU intensive as general-purpose compression! HPS cannot handle very many channels before being CPU bound**



Offloading Encoding to FPGA

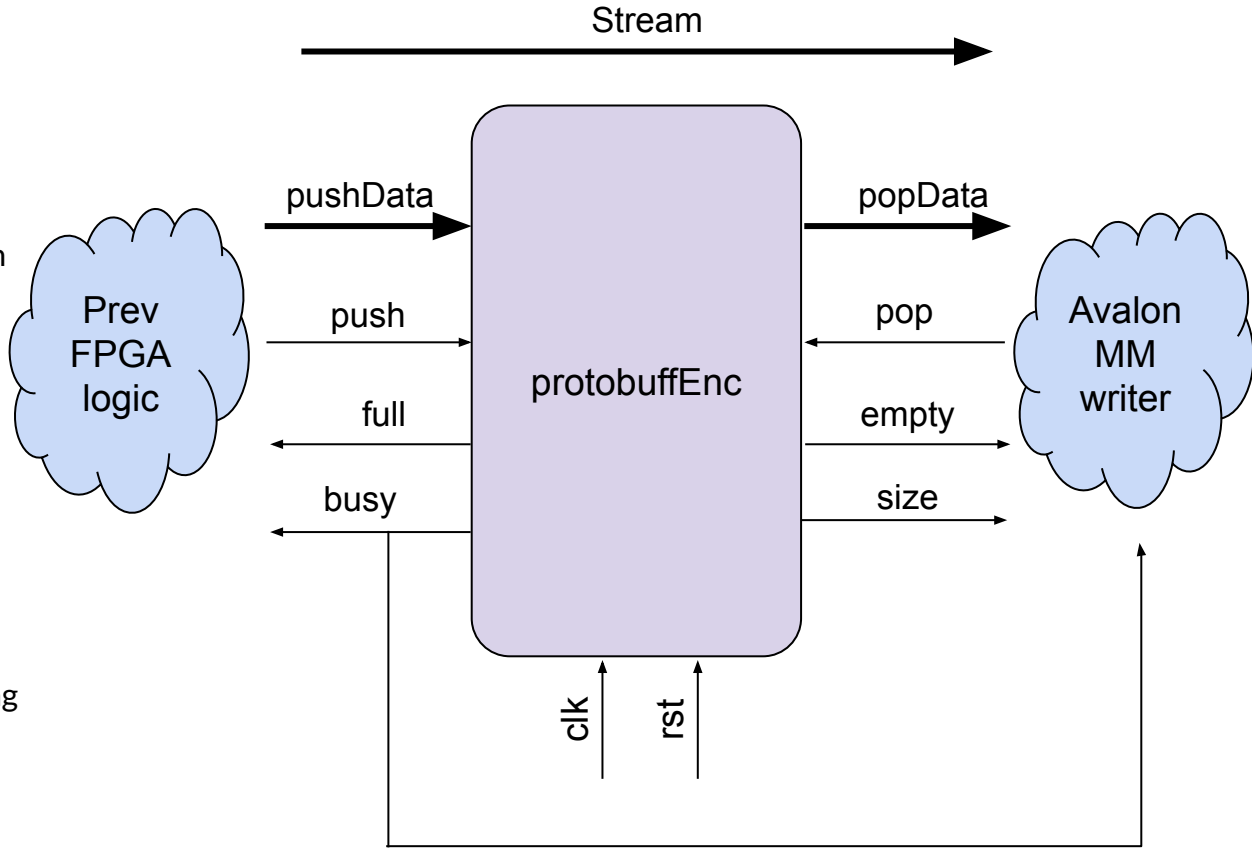
- **Solution: offload the variable integer encoding to FPGA**
- Instead of filling fixed-length data sets into circular memory buffer, fill variable-length arrays of samples encoded in protobuf format
- Job of the CPU is then simply to pack and ship! Maximizing use of 1G ethernet. **Still self descriptive**, so anything downstream knows exactly how to decode
- **Next challenge: hardware "accelerator" design**



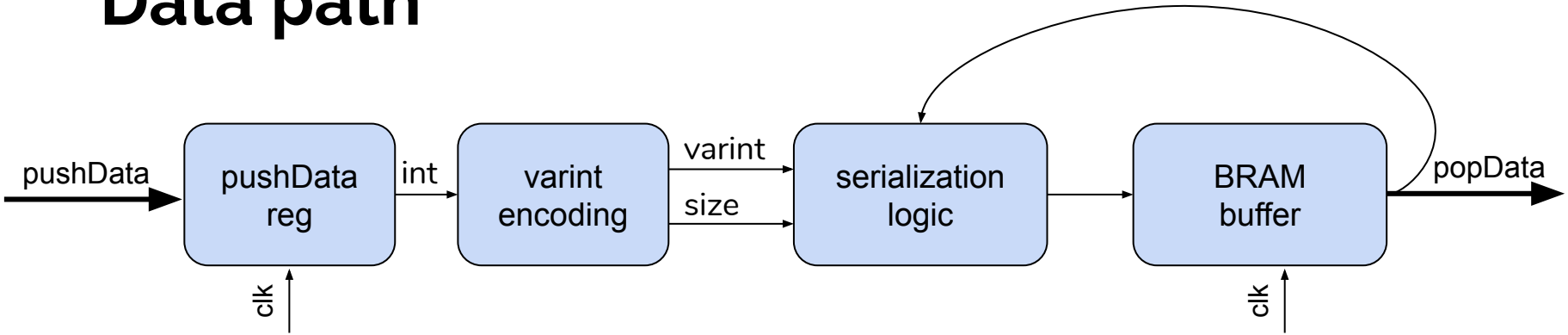
Accelerator design

Interface

- **Goals**: Perform variable integer encoding and memory serialization on data stream
- Typical FIFO (first in first out) hardware interface
- FPGA BRAMs for memory serialization buffer
- SystemVerilog parameters provide **flexible hardware**: I/O bit widths, buffer size.
- **Gotcha:** Proceed to pop until empty only after pushing array corresponding to one sample



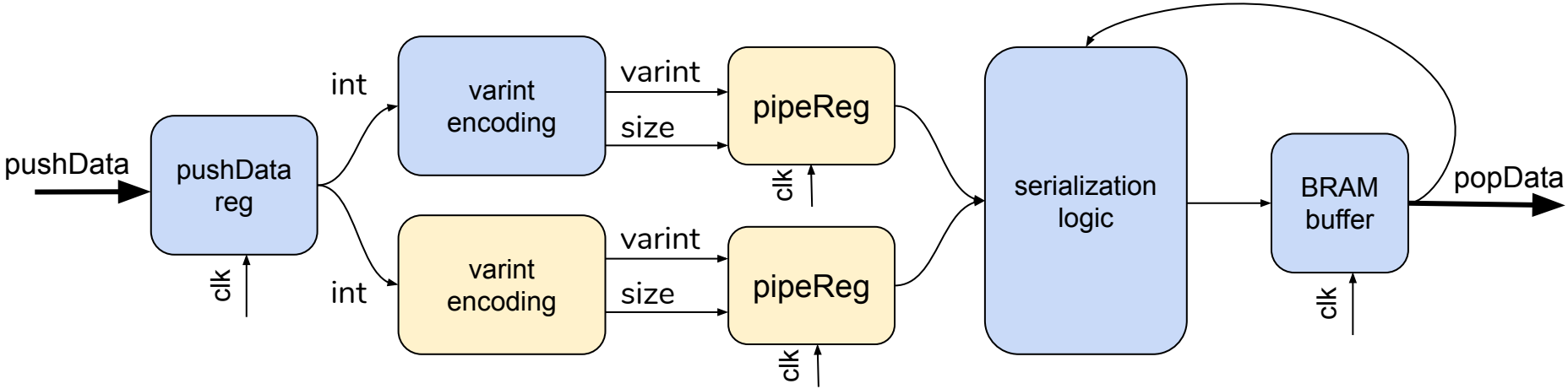
Data path



- Majority of resources utilized on serialization logic
  - several wide muxes
  - shifts required to accomodate for utilized bytes in a single address of the buffer
- Encode ~1 int per clock cycle
- This data path worked seamlessly for our use case (100-300 Mbps)

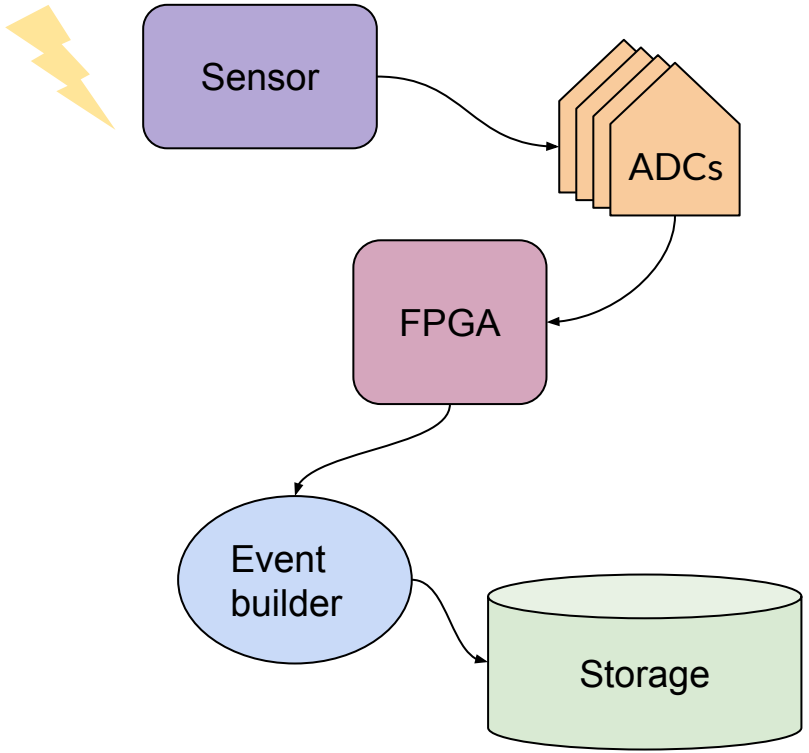
Maximizing throughput

- For more demanding designs, pipelining and parallelism can be introduced
- Pipelining allows greater clock speeds, therefore increases throughput
  - **At a price**: Inherent hardware tradeoff between power and speed



System improvements

- Reduced CPU utilization reduced by a factor of 4+
- Avoid CPU and networking bottleneck, allowing for higher sampling rate in ADCs if needed
- Any FPGA based data acquisition system can benefit from this approach



Summary

- Protocol Buffers provide language agnostic representations of data
- ProIO enables high level event structure and operations
- Protobuff accelerator in FPGA encodes data stream seamlessly
  - Reduce ethernet bandwidth utilization
  - Offloads CPU
- CPU task is reduced to pack and ship

Future Work

- Extend accelerator flexibility with a code generator (Genesis2, pyMTL)
  - Verilog language bounded ATM
- Create general purpose Avalon MM writer to manage accelerator output
- Feedback and comments are greatly appreciated!