Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

Accelerating GNNs (at Scale)

Lindsey Gray

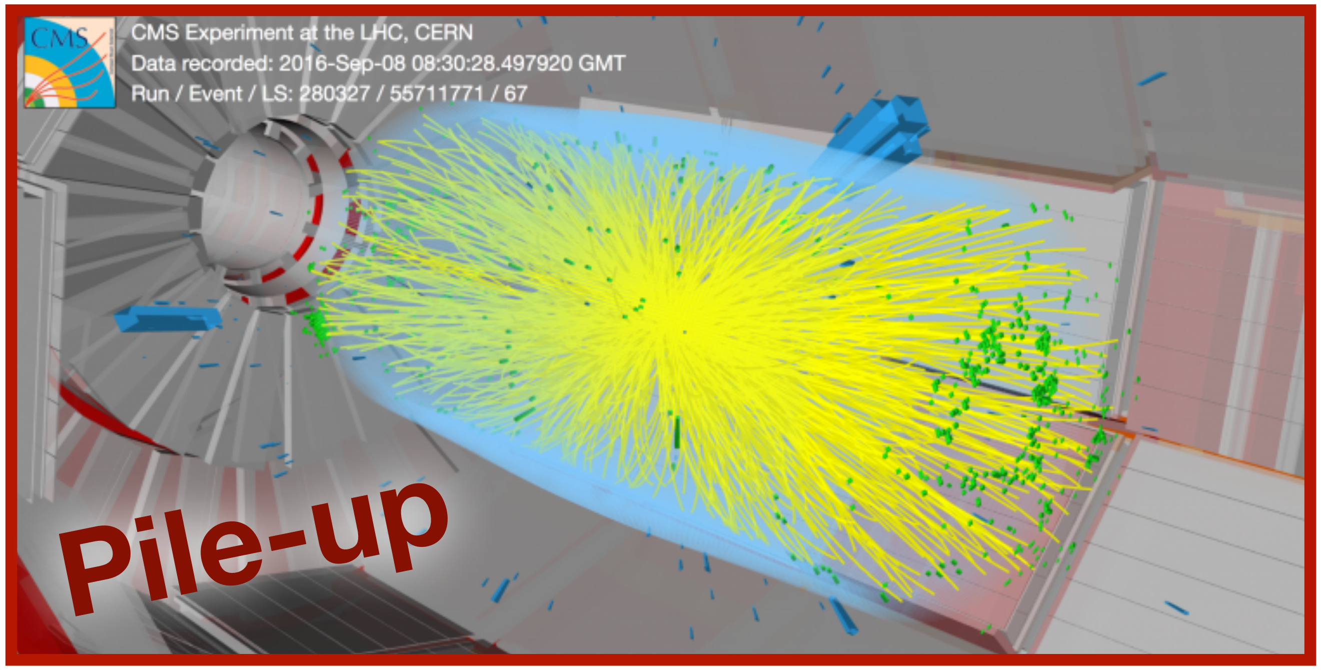
Exa.TrkX All Hands Meeting

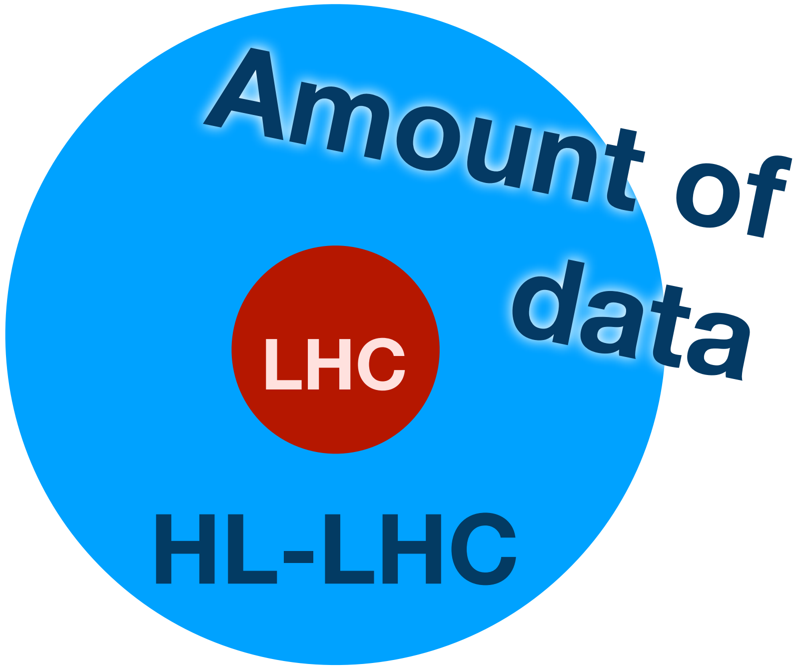
7 April 2020

with material from:

Javier Duarte, Nhan Tran, Kevin Pedro, Burt Holzman, Thomas Klijnsma, Mark Neubauer, Markus Atkinson, Yutaro Iiyama, Jan Kieseler, Matthias Fey, HLS4ML

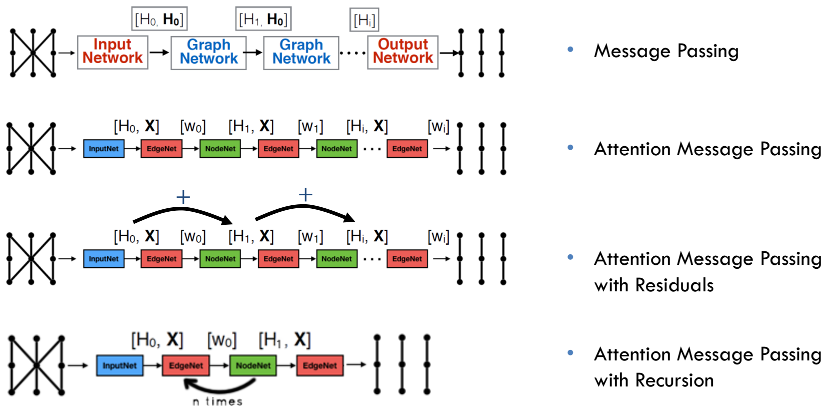
Why do we need GNN acceleration at scale?

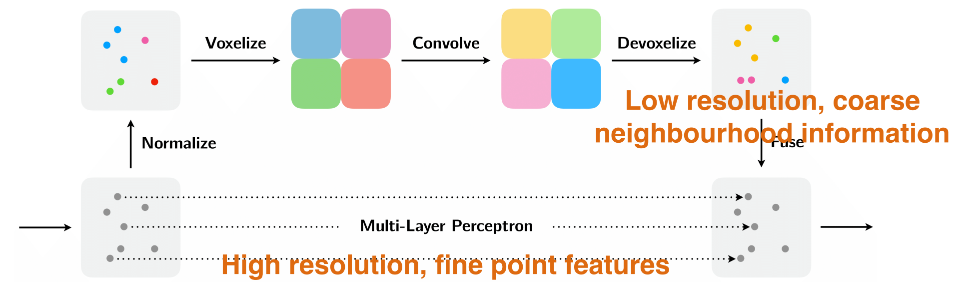


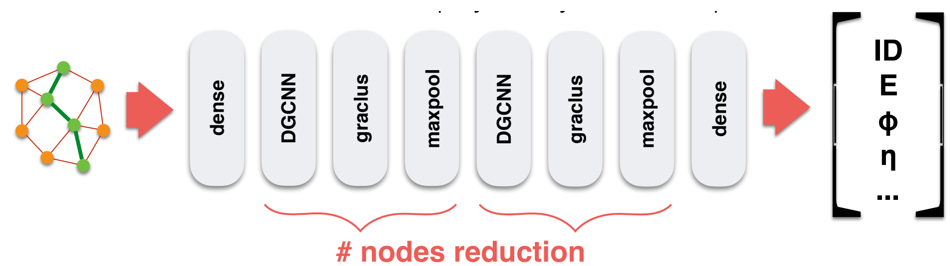


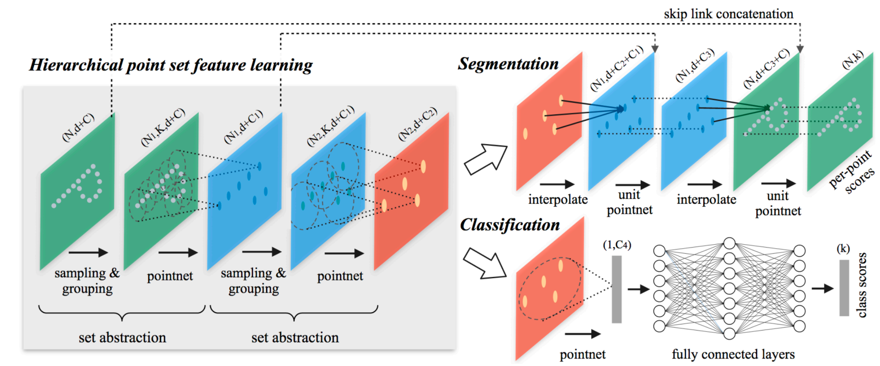
- We will need ML based reconstruction to approach the high-dimension and finely sampled data from HL-LHC
- Most of our detectors in HL-LHC will be > 3D in readout, intrinsically difficult for (most) humans to design traditional algorithms for them
- Even with execution speed improvements from using ML, need to handle 1000s of events per second coming from triggers

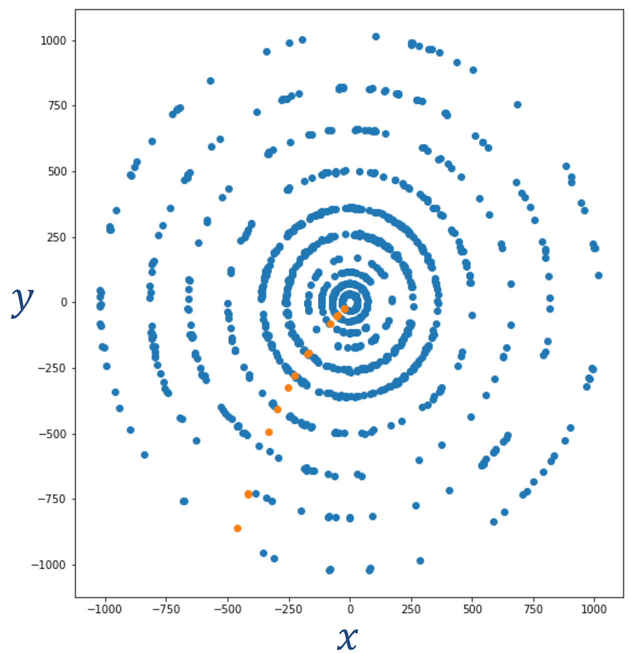
GNNs - A summary of today's models and uses

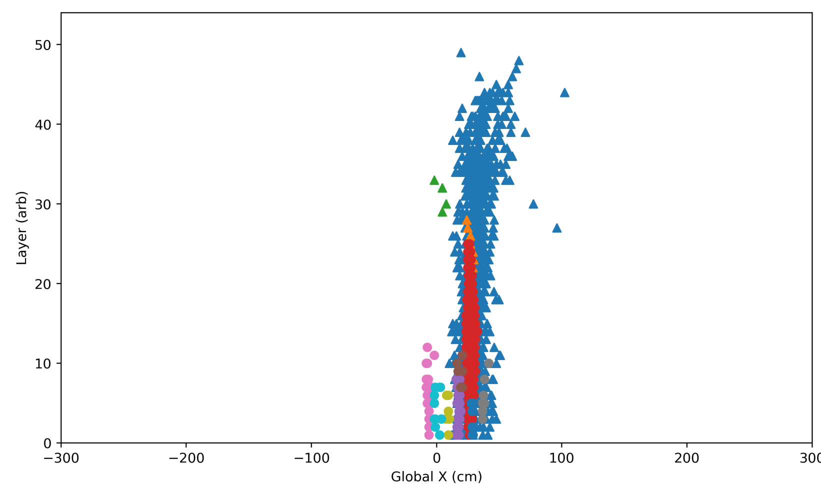


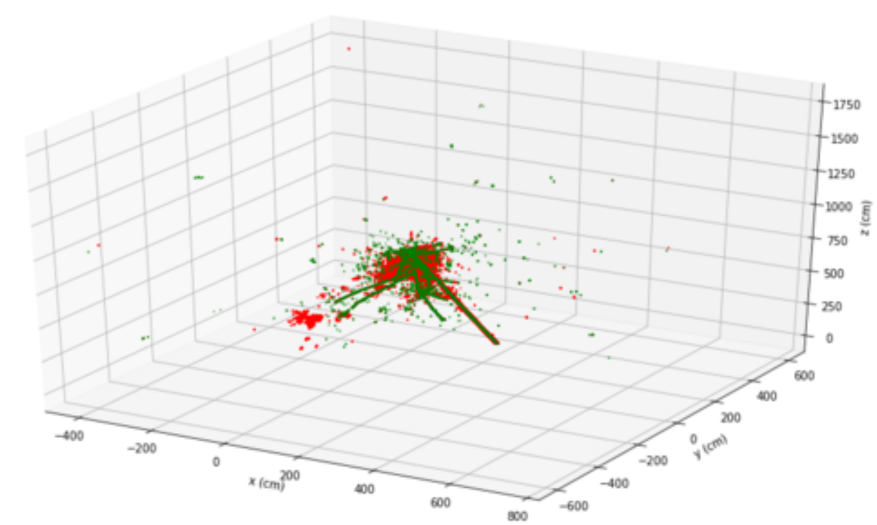












- A variety of networks available closing in on the solution
- We should start now on understanding how to evaluate inference using these models for the data volume we expect, this is no small task

Current State of GNN Acceleration (that I know about…)

“by hand” loading on to GPUs

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('using device %s'%device)
```

- Reports

Minimum when only one vertex loop iteration
Interval = latency of the first vertex loop

(Becomes fixed 258 latency and 87 interval with `continue` in the vertex loop)

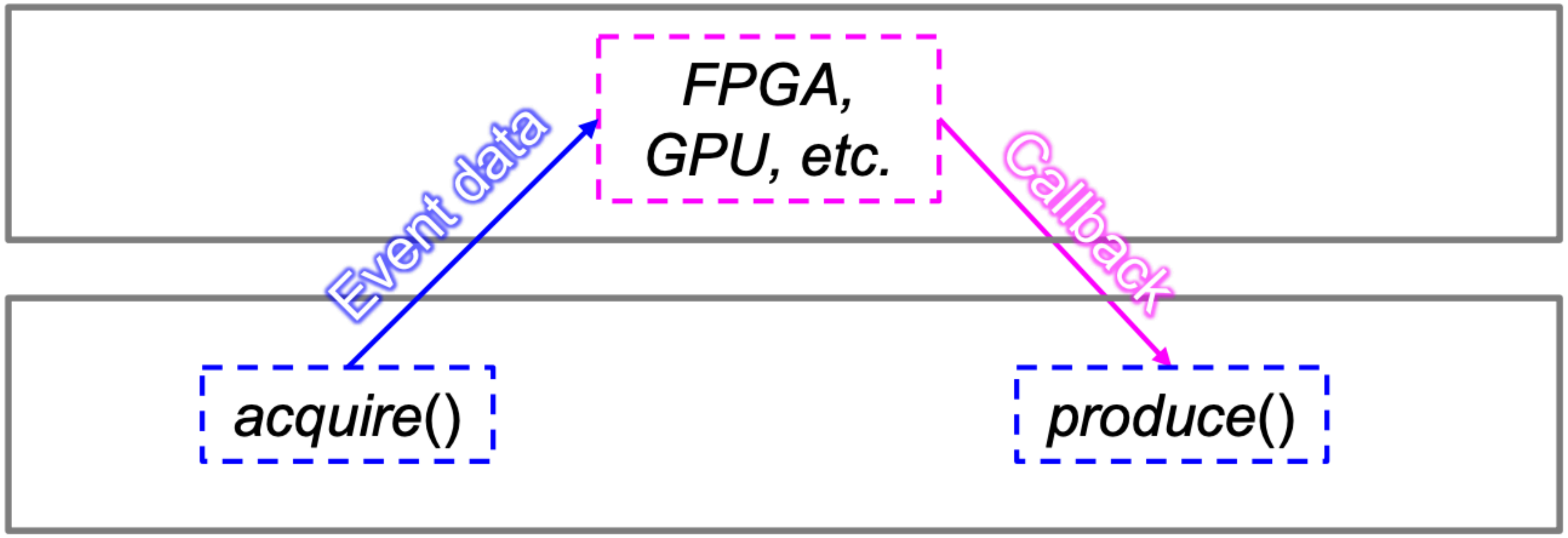
```
+ Latency:
    * Summary:
    +---------+---------+-----------+-----------+-----+-----+----------+
    |  Latency (cycles) |  Latency (absolute)   |  Interval | Pipeline |
    |   min   |   max   |    min    |    max    | min | max |   Type   |
    +---------+---------+-----------+-----------+-----+-----+----------+
    |       69|      251| 0.345 us  | 1.255 us  |   24|   87| dataflow |
    +---------+---------+-----------+-----------+-----+-----+----------+

    |        Instance         |          Module         |  min  |  max  |    min    |    max    | min | max |   Type   |
    +-------------------------+-------------------------+-------+-------+-----------+-----------+-----+-----+----------+
    |garnet_stack_U0          |garnet_stack             |     60|    249| 0.300 us  | 1.245 us  |   24|   87| dataflow |

Utilization
+---------------------+---------+-------+---------+--------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT  | URAM|
+---------------------+---------+-------+---------+--------+-----+
|Total                |      145|   1655|    82562|   99008|    0|
+---------------------+---------+-------+---------+--------+-----+
|Available SLR        |     2160|   2760|   663360|  331680|    0|
+---------------------+---------+-------+---------+--------+-----+
|Utilization SLR (%)  |        6|     59|       12|      29|  100|
+---------------------+---------+-------+---------+--------+-----+
|Available            |     4320|   5520|  1326720|  663360|    0|
+---------------------+---------+-------+---------+--------+-----+
|Utilization (%)      |        3|     29|        6|      14|    0|
+---------------------+---------+-------+---------+--------+-----+
```

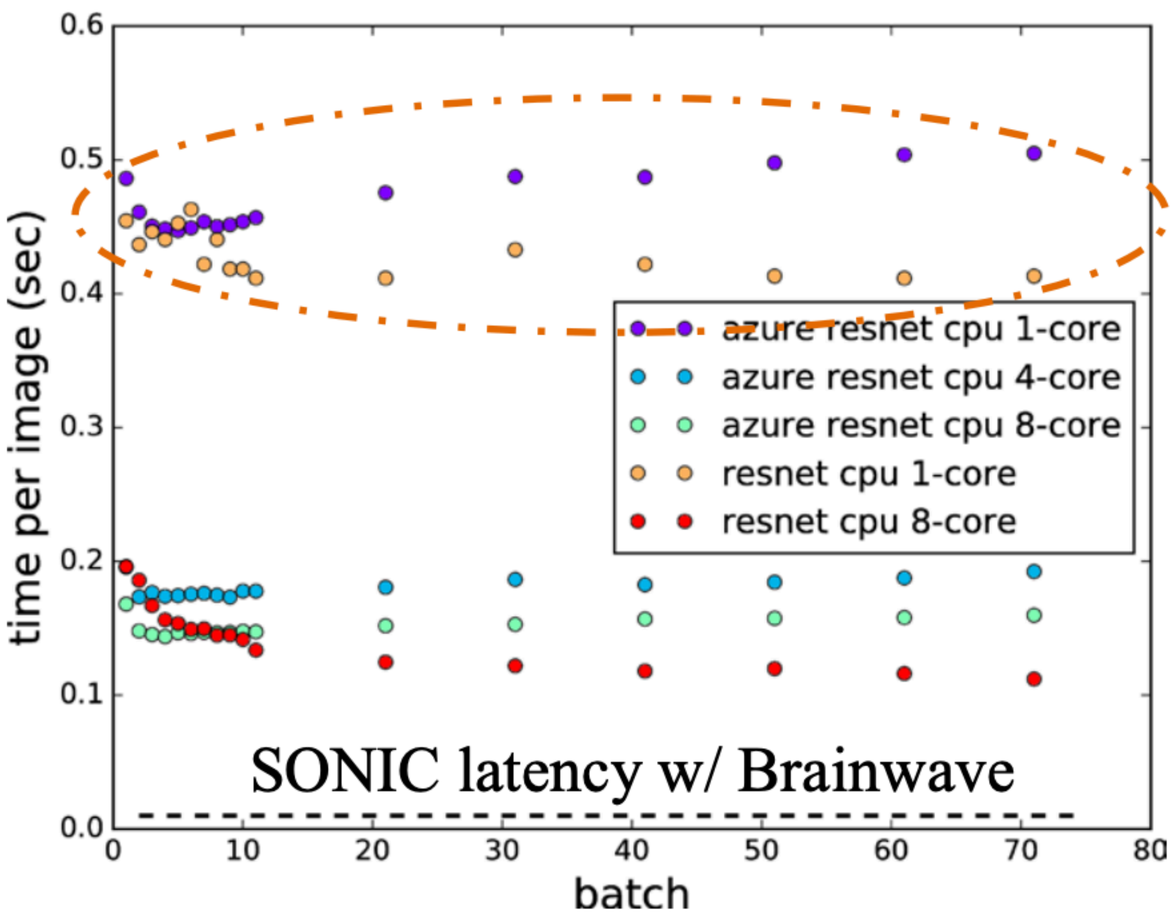
Y. Iiyama / J. Duarte

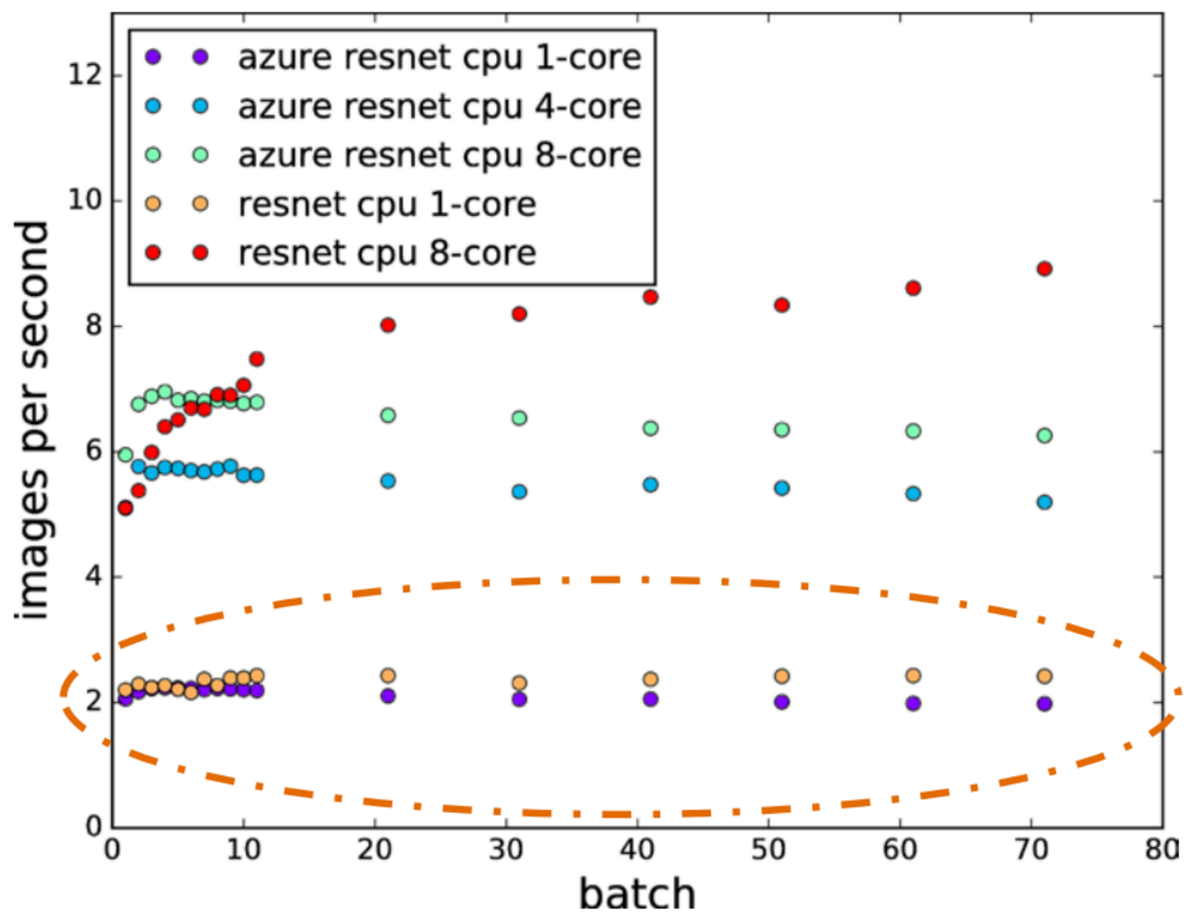
- GPU acceleration available (but the GPU needs to be on the machine)

- HEP.TrkX original network written for ~4 tracks on FPGA
- GarNet implementation recently achieved
- Initiation interval issues (time until available again), latency manageable

Inference as a Service

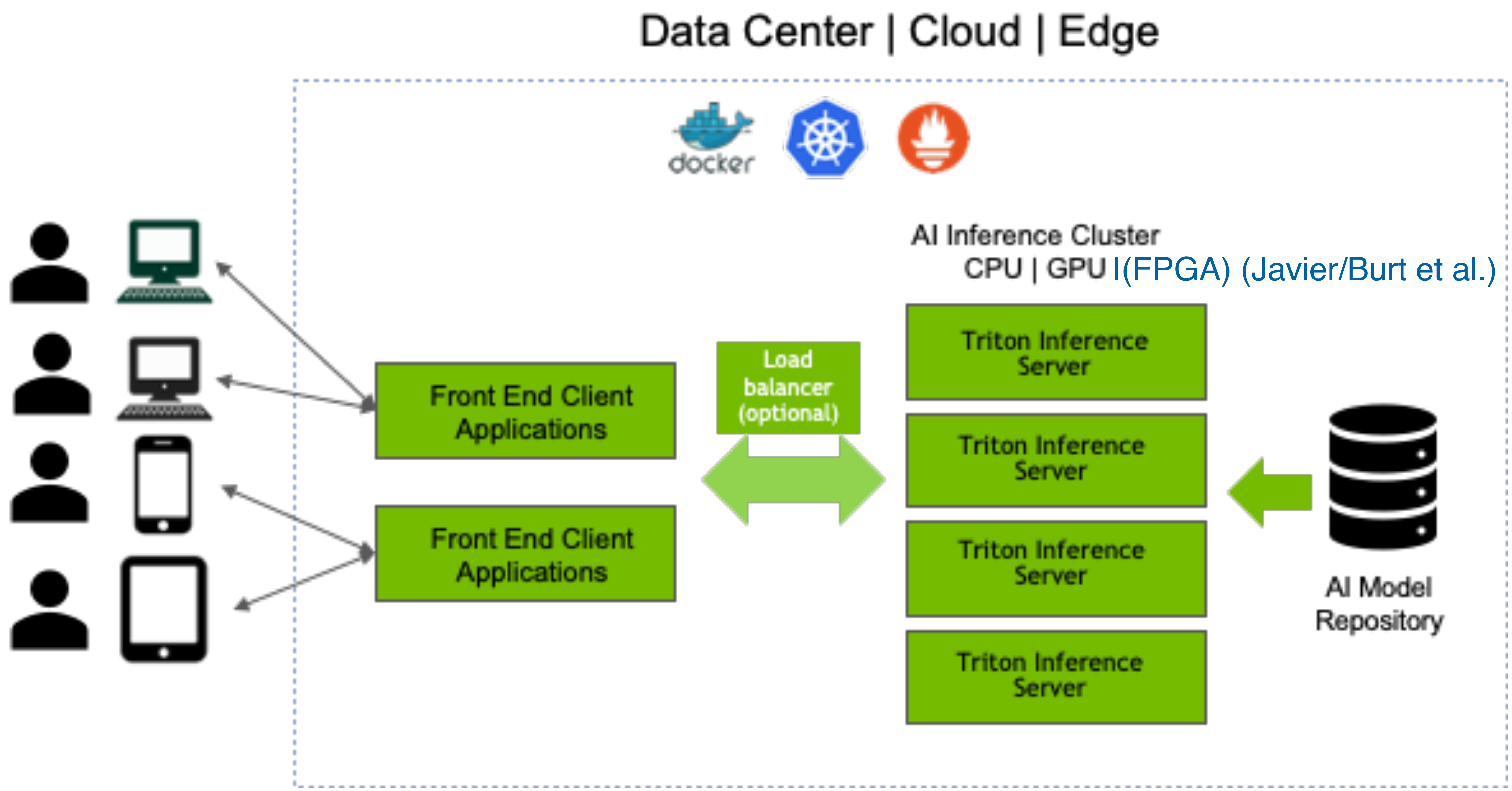






- Lets you scale inference resources independently to match experiment need

NVIDIA Triton Inference Server



- Off-the-shelf mostly battle-hardened platform for receiving and dispatching requests for inference using containers
- Database of models allows on-demand requests for inference with no requirement for loading the model on the calling device

Requirements for using TRITON

- Model must be torch script ‘jittable’ or ONNX compatible
  - pytorch’s own JIT compiler specialized for for their models
  - callable from C++

- Any external library has to be packaged with the image in a very particular way

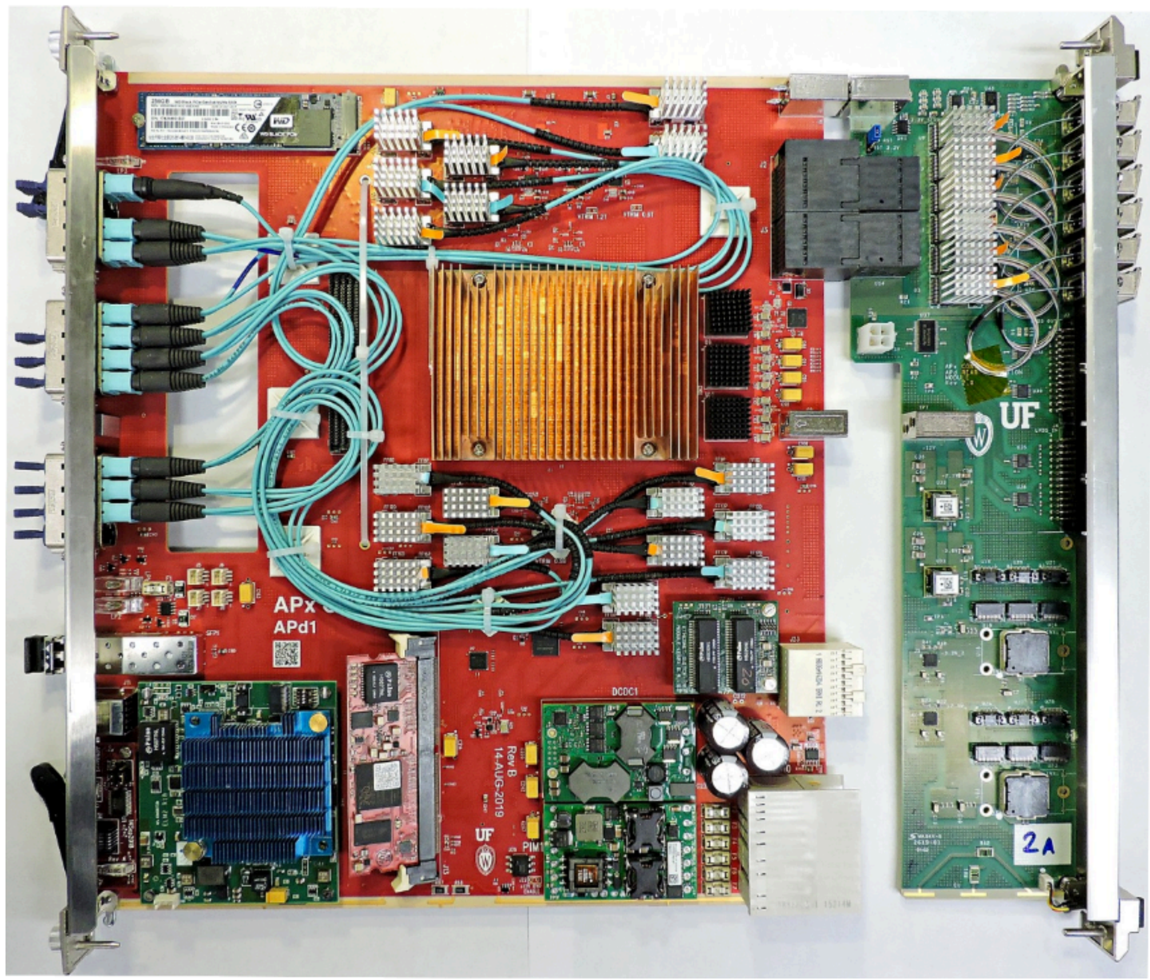
- Any external library has to already be jit-scriptable or ONNX compatible
  - Makes it prohibitively difficult to use your favorite python module
  - Until very recently the pytorch geometric dependencies weren’t integrated this way
  - The pytorch geometric base classes are inherently not nit-compatible
  - This means that right now you have to rewrite models once you figure them out (boooo)

- Working with Matthias Fey to yield jittable synthesis of models implemented in pytorch geometric
  - i.e. you go “.jittable()” on your model and it writes it for you

- Still, this is very clearly the best supported method for scaling inference as a service
  - and is extensible to doing inference on FPGAs as well

Directions for FPGA Acceleration

Real time (L1) applications



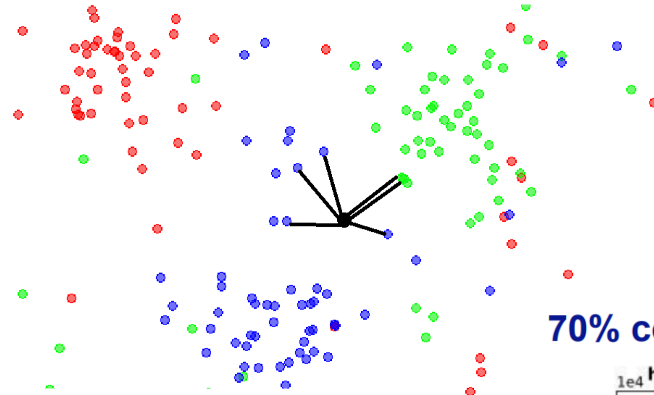
Coprocessor Applications

- Two major directions for optimization: real time & coprocessor
  - rather different optimization requirements
- Co-processors have less strict latency and space requirements, typically
- Figuring out real-time implementations helps us bring better algorithms to L1

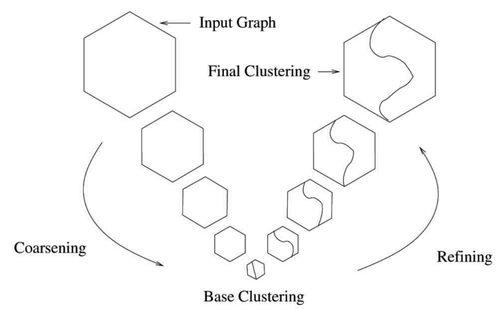
Factoring problems in scaling GNNs on FPGAs (my take)

Graph Algorithms

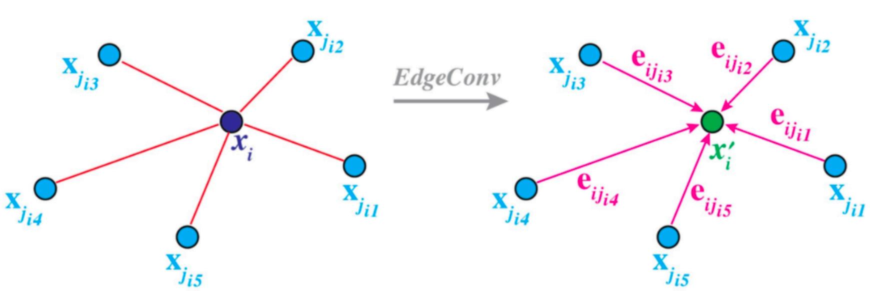
k-Nearest-Neighbours



graph clustering

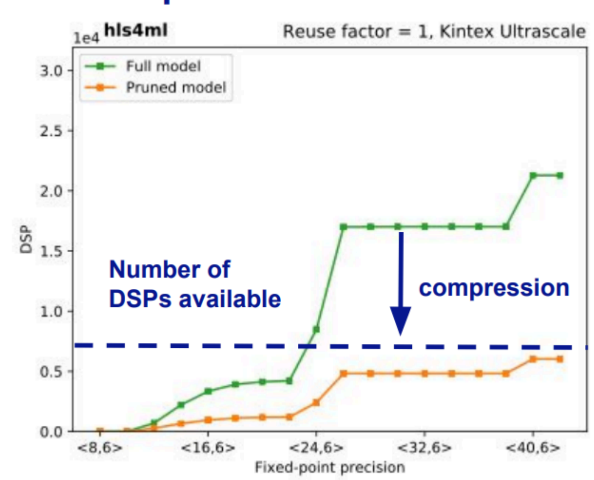


message passing

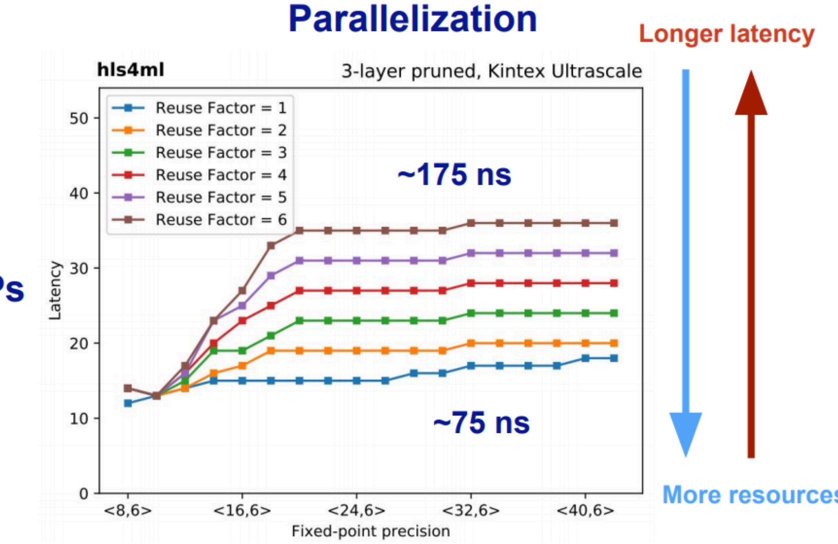


Fully Connected Network Evals

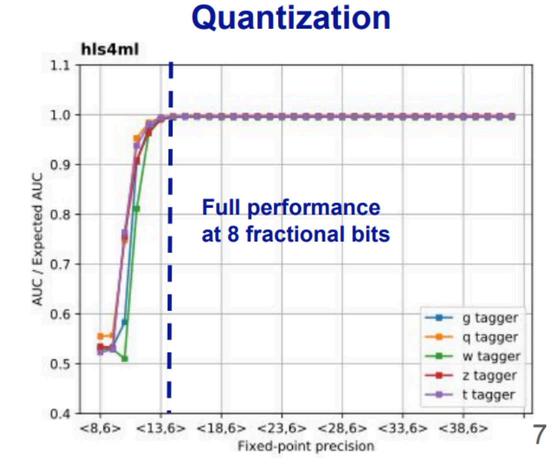
70% compression ~ 70% fewer DSPs



Parallelization



Quantization



- Instead of implementing a fully-integrated GNN, why not try using what's already there

- Graph algorithms are the real missing piece, fully connected networks well studied

Fermilab

The big problem(s?)

- All of these graph algorithms operate on variable inputs
  - Any solution we're going to implement in FPGAs will need to operate on fixed numbers of points to make them compile-time static
  - So for really large graphs we are stuck processing them iteratively
  - This is prohibitive for real-time applications

- GarNet (from Yutaro et al.) does get around some of these issues by effectively embedding the graph algorithms in a neural network
  - Maybe there's some mileage there to go?
  - It's sort of like a learned k-means

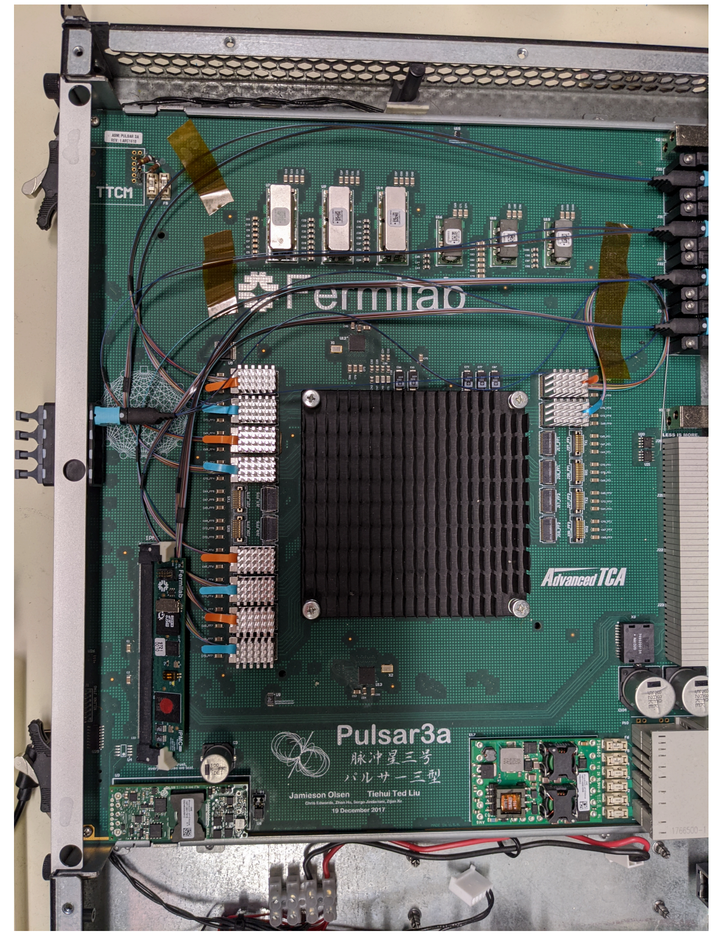
- There is significant possibility for busting up the problem into sectors, etc.
  - This is how people are typically approaching the problem, and it makes sense
  - However, you pay for sectors in post processing algorithms and space on FPGA

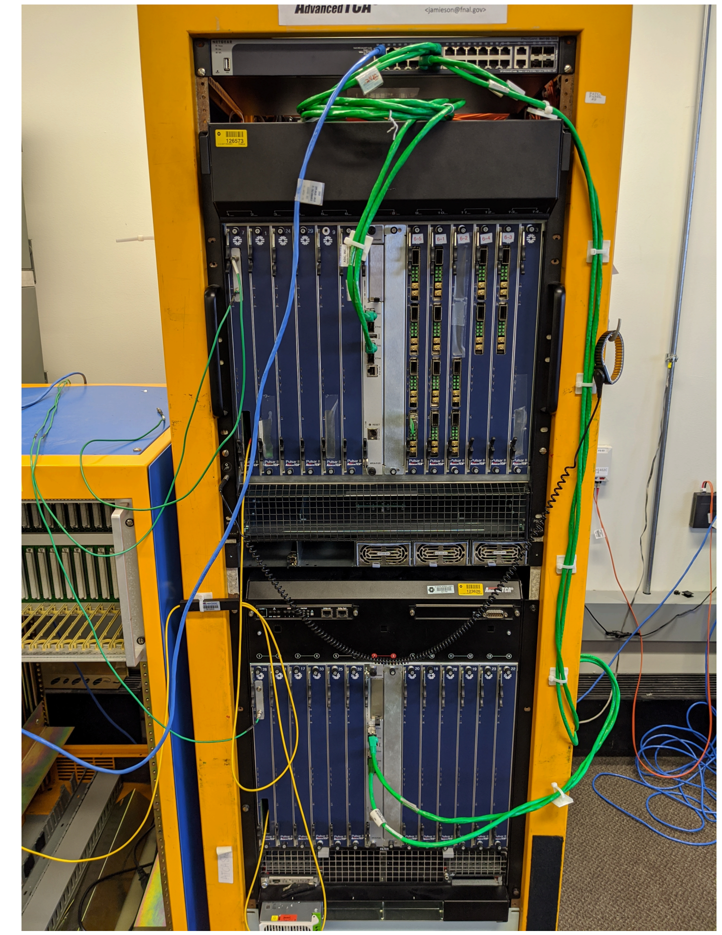
- There is some work already in HLS4ML towards distributing networks over multiple FPGAS (Javier, Yutaro, Mark, Markus, et al.)
  - We have to distribute the network *and* the graph, it's a bit of a harder problem

My tack on this

Start with co-processors

Then scale coprocessors on trigger HW

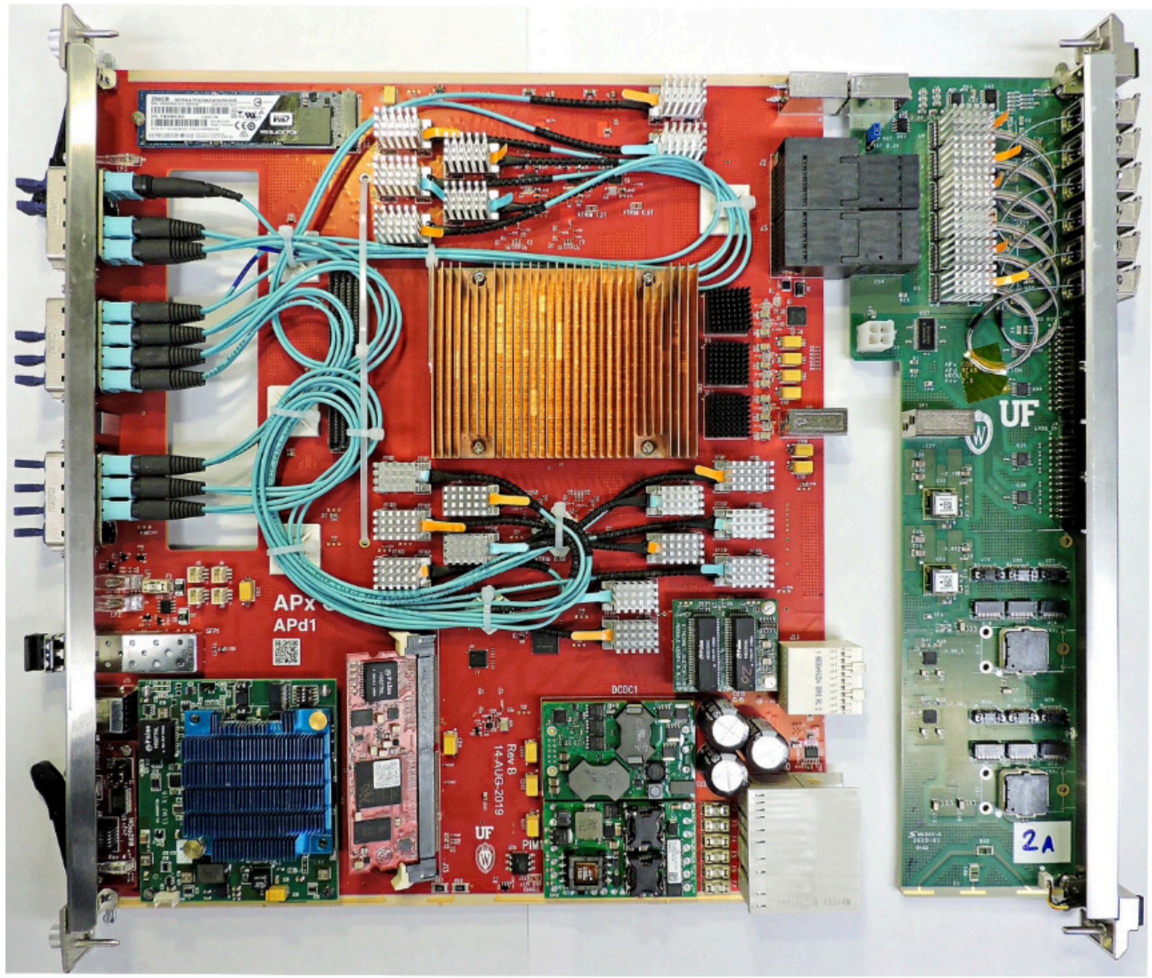




- Seems prudent to focus on developing co-processors first to understand the graph algorithms and how best to integrate them with existing DNN inference
- What topologies of data exchange work the best?
- Then learn how to implement co-processor style setups on trigger hardware

and then… engineer realtime algorithms

use knowledge from scaling graphs to implement real time algorithms



- Use knowledge gained on coprocessors to yield a real-time implementation

- Likely that this occurs in tandem with co-processor development
  - people already working on both anyway!
  - I thought it may be useful to factor the approach a bit and focus thinking

Conclusions and Outlook

- ML is going to be a cornerstone of next generation experiments
  - Allows us to scale reconstruction and analysis algorithms to new levels of complexity
- There are technologies today that let us scale our inference capacity
  - Instead of asking if we can fit a GPU on each compute node, we can just scale to the right number of GPUs
- FPGAs offer improved power density and speed but are a bit at odds with the rather flexible nature of GNNs
  - It will take time to understand how to scale the algorithms
- A factored approach may help us in understanding the right way to apply GNNs on FPGAs, yielding the best computing performance.
- Bringing GNNs to micro-second level evaluation times will expose powerful techniques to HL-LHC triggering and analysis strategies
  - The physics case for this stuff is pretty easy to write down
  - The work needed to accelerate GNNs is at the intersection of software, hardware, and infrastructure