Threads to Grids: Heterogeneous Computing

- or -

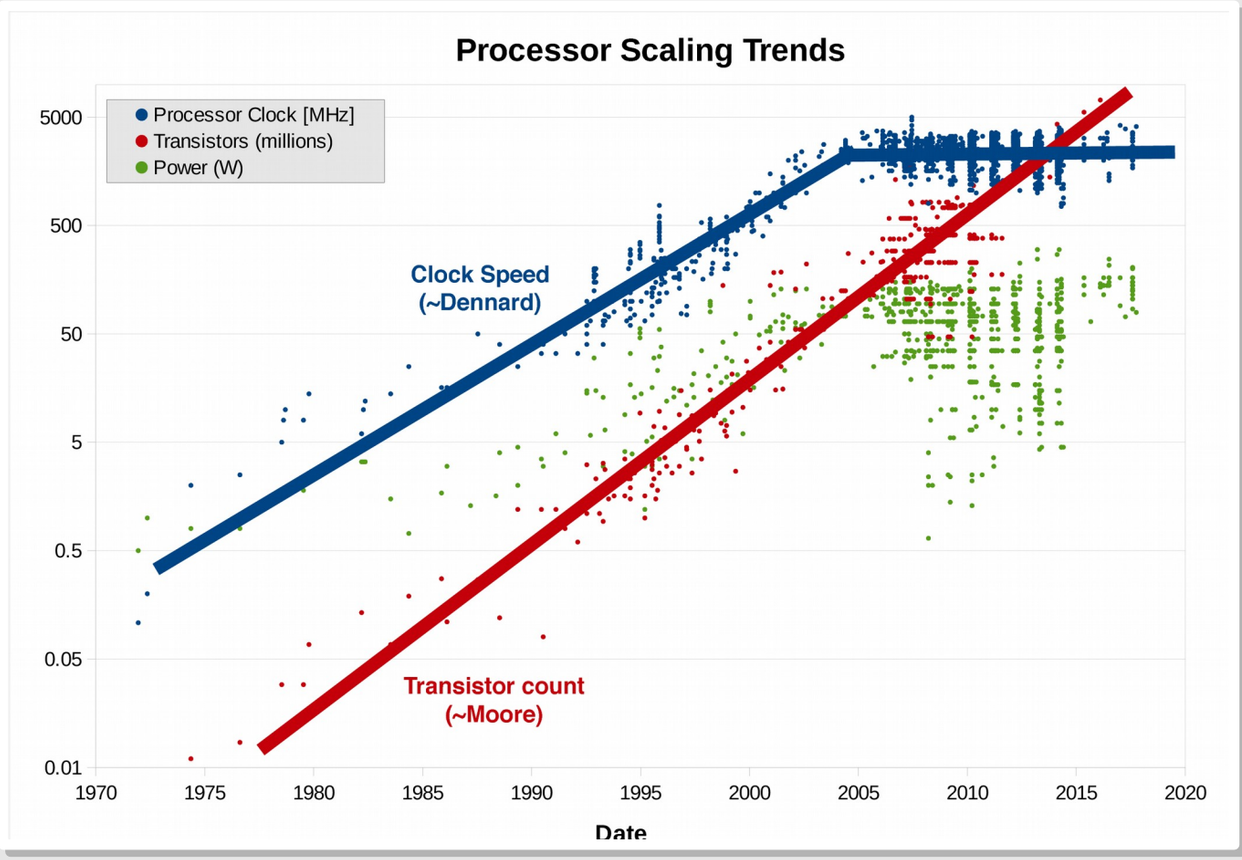
The Future's So Scary, I Should Probably Retire Before Run 4

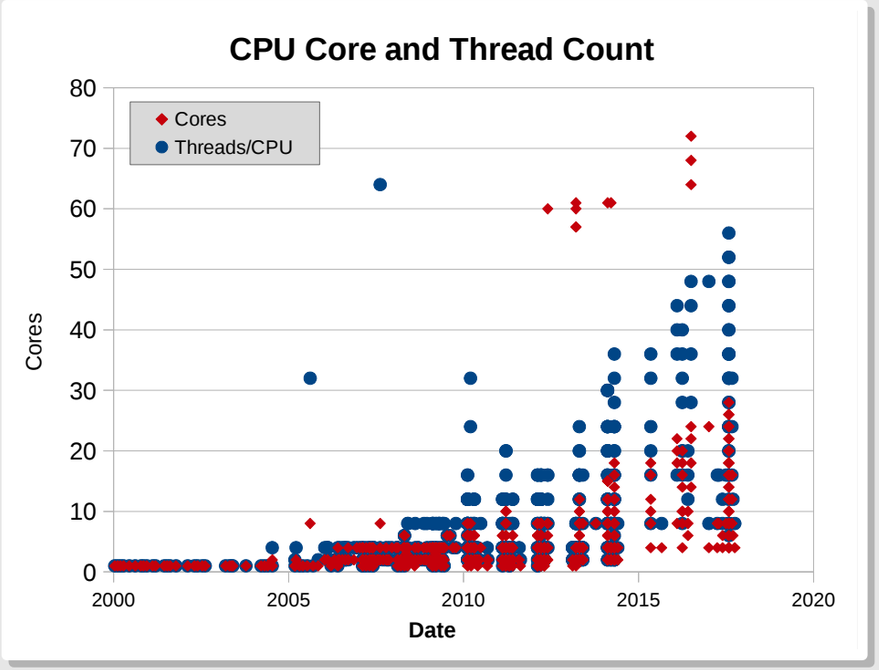
Charles Leggett, Chris Jones

Joint WLCG/HSF Workshop: Frameworks and Infrastructure
Mar 27 2018

The Past Through Tomorrow

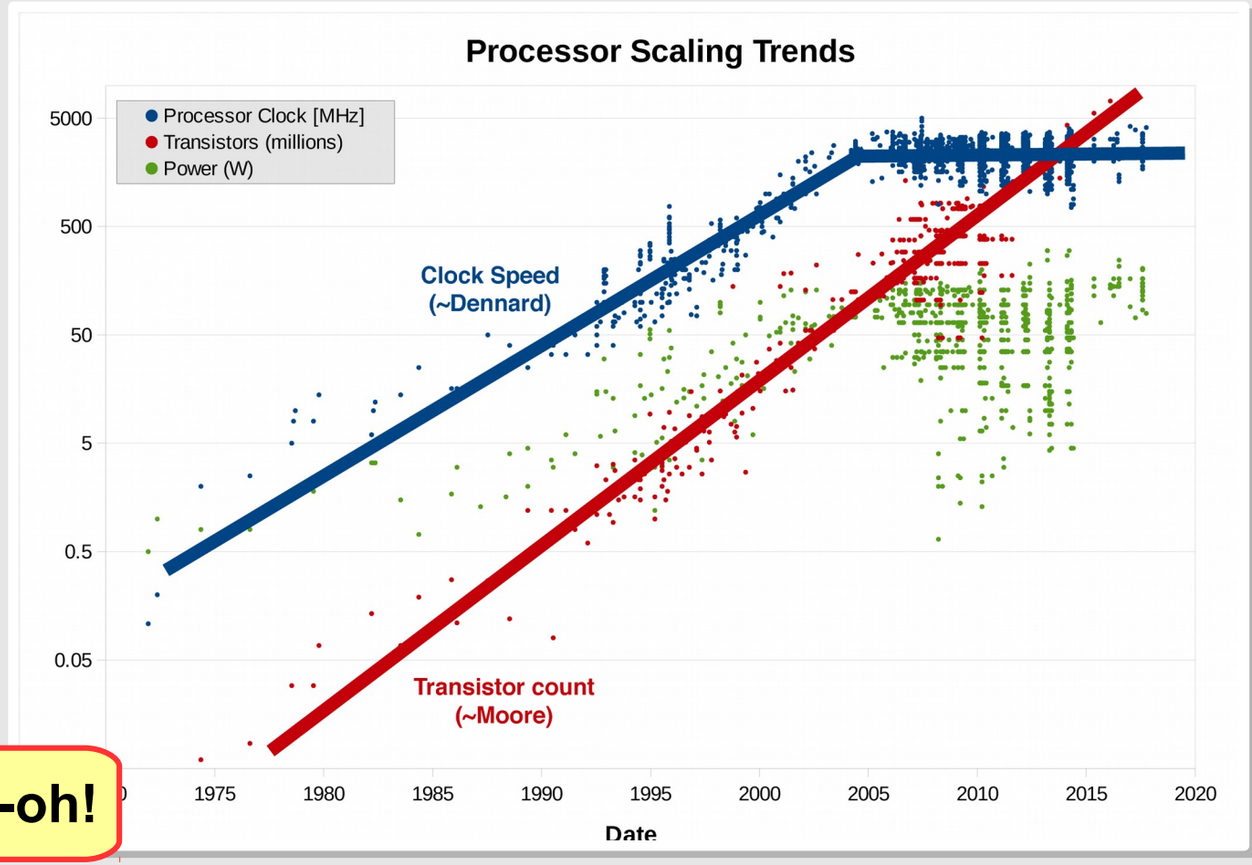
- 10 years ago we saw and predicted a major change in computing architecture driven by CPU design limitations
  - smaller, less powerful CPUs
  - less memory per core
- In response, we have heavily invested in multi-threading to better make use available resources

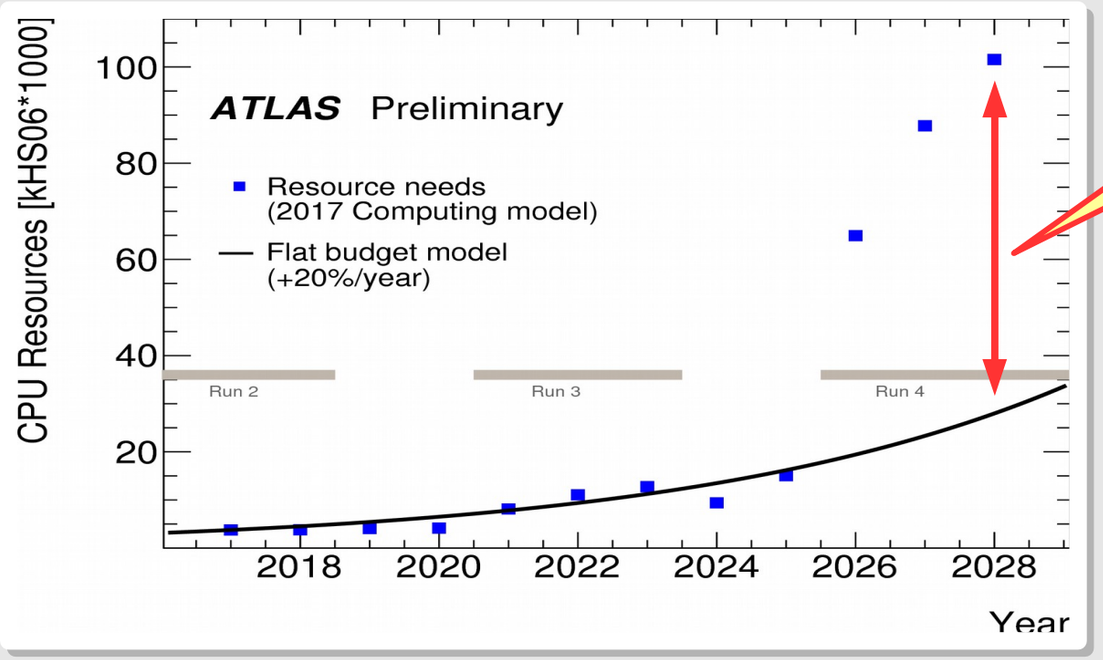




The Past Through Tomorrow

- 10 years ago we saw and predicted a major change in computing architecture driven by CPU design limitations
  - smaller, less powerful CPUs
  - less memory per core
- In response, we have heavily invested in multi-threading to better make use available resources



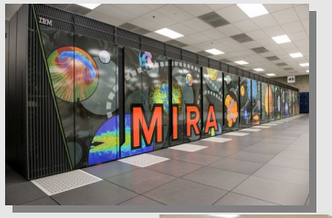


oh-oh!

- Unfortunately, we will need **much** more computing power in the not so distant future than we have budgeted for
- Where can we find it?

Some Current US Supercomputers

- Argonne: Mira (2012)
  - 49152 PowerPC A2 (16 core, 16GB/CPU)



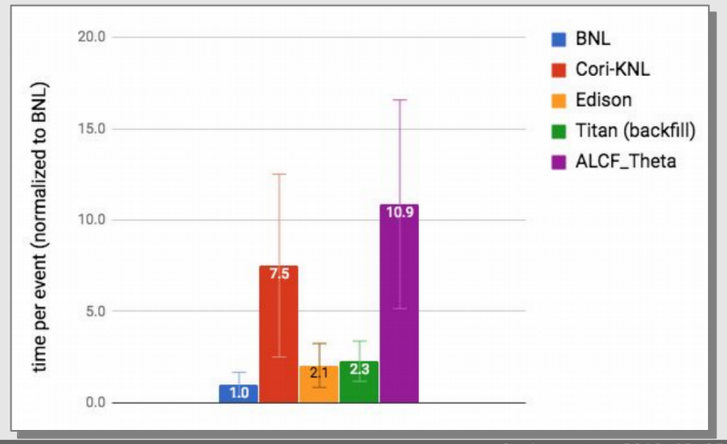
- LBL: Edison (2013)
  - 11172 Intel Xeon IvyBridge (12 core, 32 GB/CPU)

- Oak Ridge: Titan (2013)
  - 18688 AMD Opteron 6274 (16 cores, 32GB/CPU)
  - 18688 Nvidia Tesla K20X GPU (6GB)

- LBL: Cori (2017)
  - 4776 Intel Xeon Hawell E5-2698 (16 core, 96 GB/CPU)
  - 9688 Intel KNL 7250 (68 core, 96 GB/CPU)

- Argonne: Theta (2017)
  - 4392 Intel KNL 7230 (64 core, 192 GB/CPU)

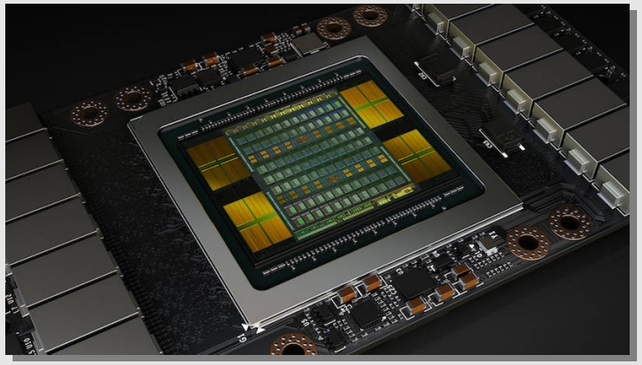
- LANL: Trinity (2017)
  - 9436 Xeon E5 + 9984 KNL



Next Generation Supercomputers

- Oak Ridge: Summit (2018)
  - 9200 IBM Power9 AC922 (24 core, 256GB/CPU)
  - 27600 (3x) NVIDIA Volta V100 + NVLink

GPUs are very popular for HPCs due to efficiency (power/space) for providing lots of FLOPS



- LLNL: Sierra (2018)
  - (many; fewer than Summit?) IBM Power9 AC922
  - 2x NVIDIA Volta V100 + NVLink
- LBL: NERSC-9 (2020)
  - ??? - was supposed to be successor to KNL
  - will instead probably be a Sierra/Summit like system
- Argonne: Aurora (2021?)
  - ??? - was supposed to be successor to KNL
  - "novel architecture"
  - https://www.alcf.anl.gov/alcf-aurora-2021-early-science-program-data-and-learning-call-proposals

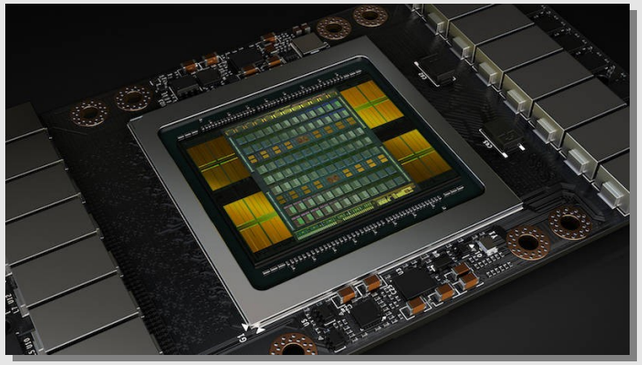
Commercial Systems

- NVIDIA GPU Cloud Computing
- NVIDIA GPU Cloud Gaming

Next Generation Supercomputers

- Oak Ridge: Summit (2018)
  - 9200 IBM Power9 AC922 (24 core, 256GB/CPU)
  - 27600 (3x) NVIDIA Volta V100 + NVLink

GPUs are very popular for HPCs due to efficiency (power/space) for providing lots of FLOPS



- LLNL: Sierra (2018)
  - (many; fewer than Summit?) IBM Power9 AC922
  - 2x NVIDIA Volta V100 + NVLink

- LBL: NERSC-9 (2020)
  - ??? - was supposed to be successor to KNL
  - will instead probably be a Sierra/Summit like system

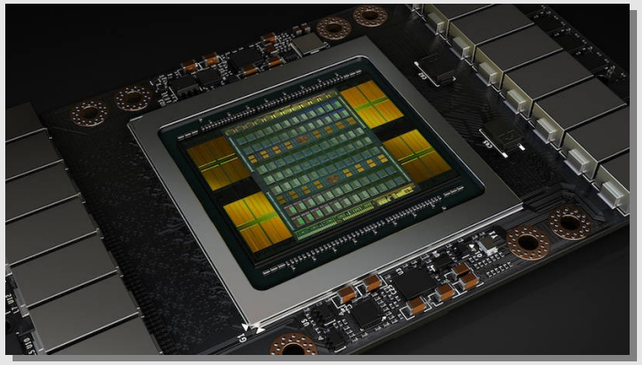
- Argonne: Aurora (2021?)
  - ??? - was supposed to be successor to KNL
  - "novel architecture"
  - https://www.alcf.anl.gov/alcf-aurora-2021-early-science-program-data-and-lea

GPUs are the way of the future

Next Generation Supercomputers

- Oak Ridge: Summit (2018)
  - 9200 IBM Power9 AC922 (24 core, 256GB/CPU)
  - 27600 (3x) NVIDIA Volta V100 + NVLink

GPUs are very popular for HPCs due to efficiency (power/space) for providing lots of FLOPS



- LLNL: Sierra (2018)
  - (many; fewer than Summit?) IBM Power9 AC922
  - 2x NVIDIA Volta V100 + NVLink

- LBL: NERSC-9 (2020)
  - ??? - was supposed to be successor to KNL
  - will instead probably be a Sierra/Summit like system

- Argonne: Aurora (2021?)
  - ??? - was supposed to be successor to KNL
  - "novel architecture"
  - https://www.alcf.anl.gov/alcf-aurora-2021-early-science-program-data-and-learning-call-proposals

the future sucks (for us)

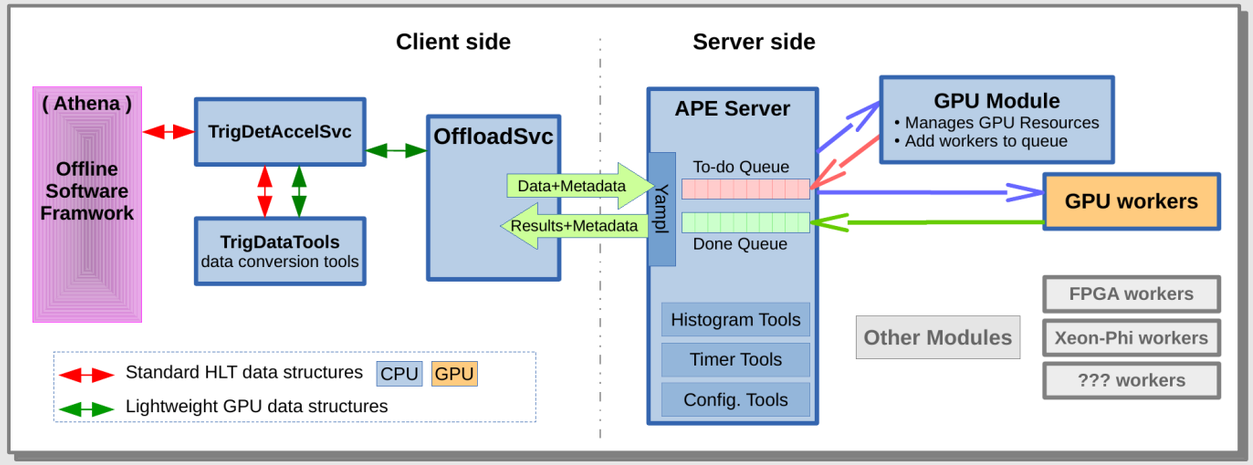
GPUs in HEP

- There have been many attempts to make use of GPUs for HEP
- In general, GPUs are not really practical to use as just another regular compute engine
  - excel at some tasks, fail horribly at others
  - code/kernels need to be rewritten to take advantage of hardware / memory layout
  - work best with SIMD style processing
- Some types of HEP code are well suited for GPUs
  - some pattern recognition (*eg* tracking)
  - G4 EM and neutral physics
  - Calorimeter clustering
- Some things don't
  - anything branchy, sorts, etc
  - lots of code, little data
- Just because we have one way of doing something now that's designed for a CPU, doesn't mean we need to do it the same way on a GPU
  - track fitting w/ kalman -> machine learning for pattern reco

Current GPU Usage in ATLAS

- Investigated for use in Trigger
  - Track seeding and topological cluster formation in Calorimeter
  - 3 stages:
    - convert ATLAS EDM objects -> plain arrays
    - offload to GPU via APE server
    - retrieve data from GPU, convert to ATLAS EDM
  - Significant speedup found in GPU based processing:
    - individual Alg 10x faster
    - total throughput 1.4x
  - APE server can be used to offload to GPU, FPGA, other network sinks

athenaMT can oversubscribe threads to efficiently schedule blocking tasks - no extra coding needed to offload and wait



- Hard to saturate GPU
  - required (many) more than 55 clients to saturate a single NVidia GTX1080 (pascal)
    - V100 (Summit/Sierra) is several times more powerful than a GTX1080
  - adding more clients runs into bottlenecks with network transfer and ability of server to manage large numbers of data transfer connections

Current GPU Usage in CMS

- Prototype of generic offloading mechanism integrated with framework
  - offload to GPU, FPGA, anther process

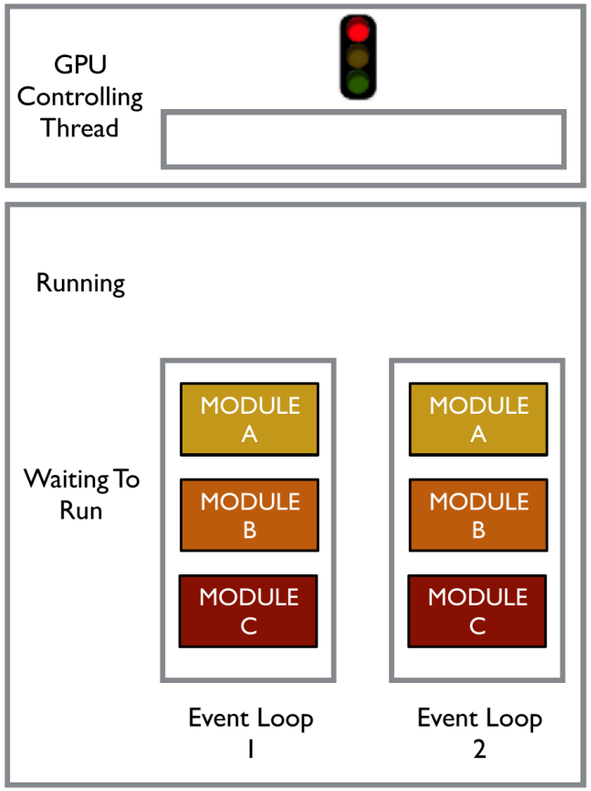
Setup

TBB controls running modules

Can have concurrent processing of multiple events

Have separate helper thread to control GPU

Waits until enough work has been buffered before running GPU kernel



Current GPU Usage in CMS

- Prototype of generic offloading mechanism integrated with framework
  - offload to GPU, FPGA, anther process

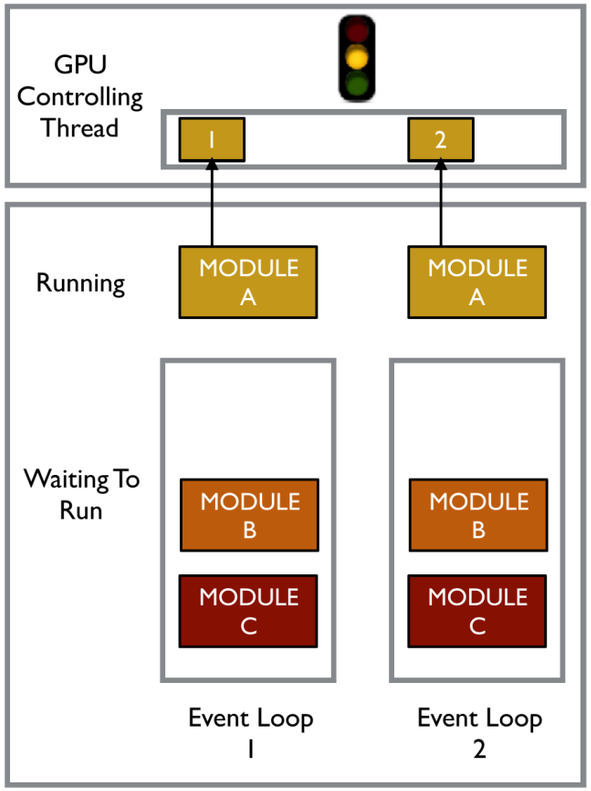
Acquire

Module acquires method called

Used to pull data from Event

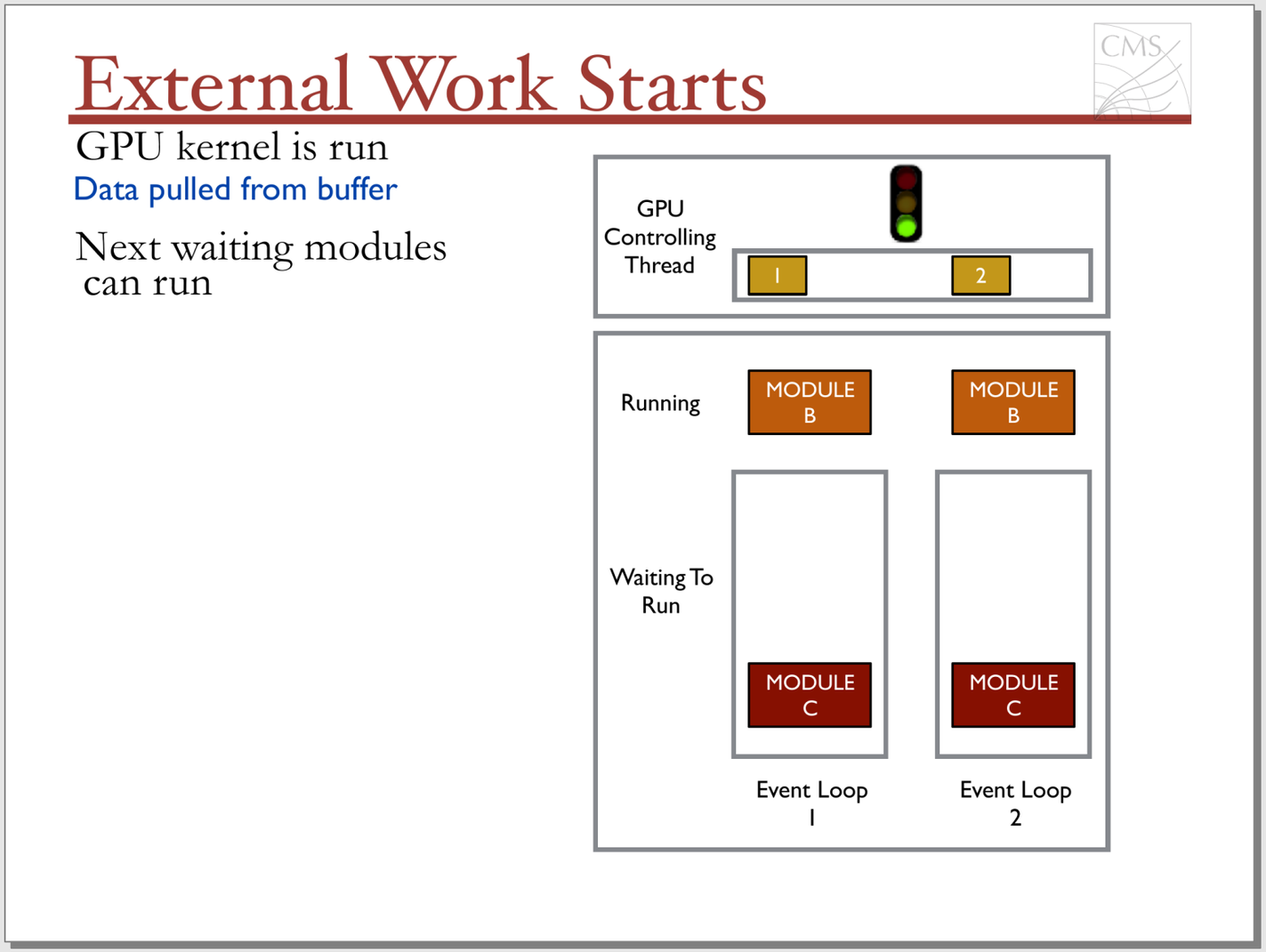
Copies data to buffer

Includes a callback to start next phase of module running



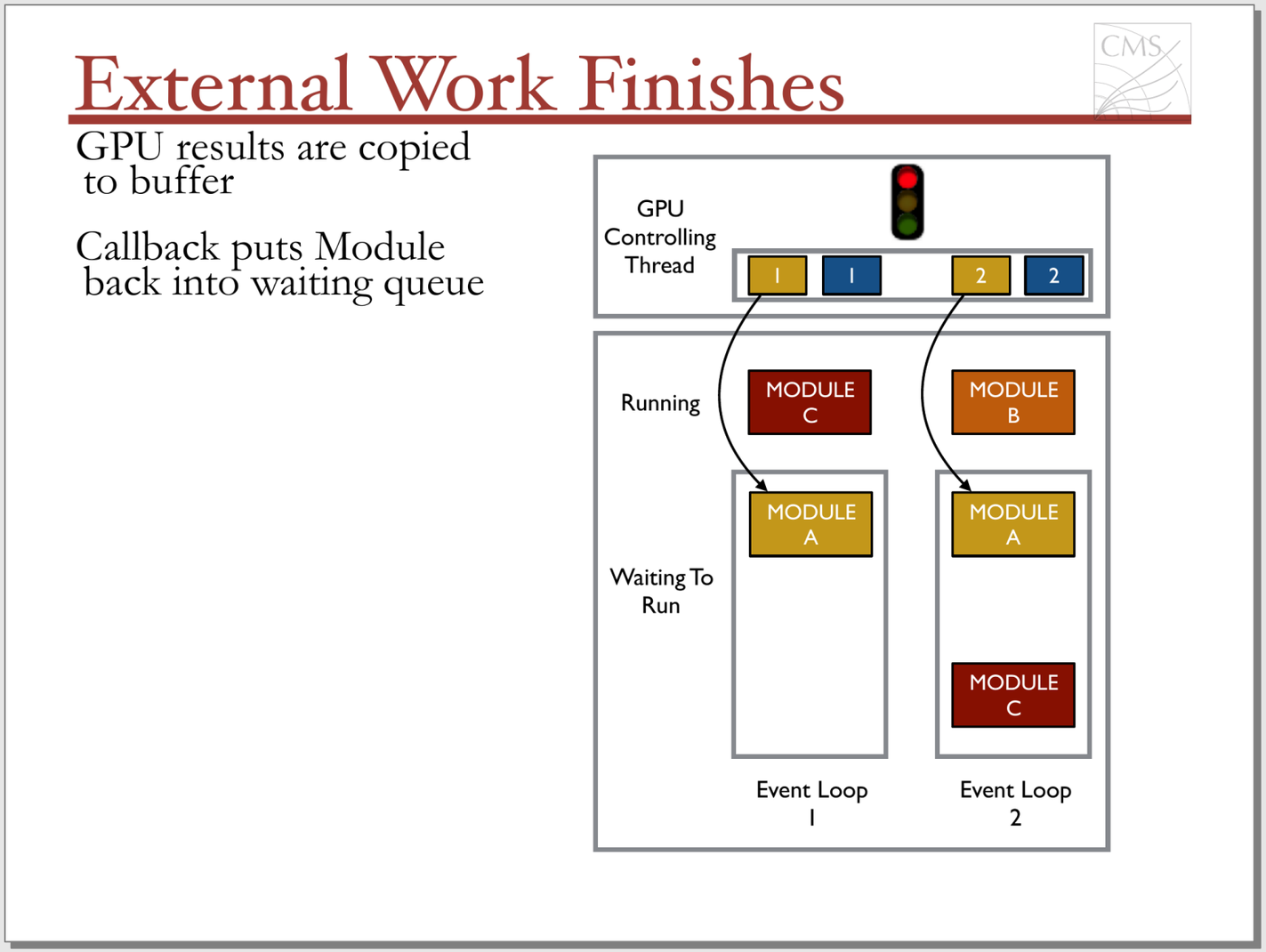
Current GPU Usage in CMS

- Prototype of generic offloading mechanism integrated with framework
  - offload to GPU, FPGA, anther process



Current GPU Usage in CMS

- Prototype of generic offloading mechanism integrated with framework
  - offload to GPU, FPGA, anther process



Current GPU Usage in CMS

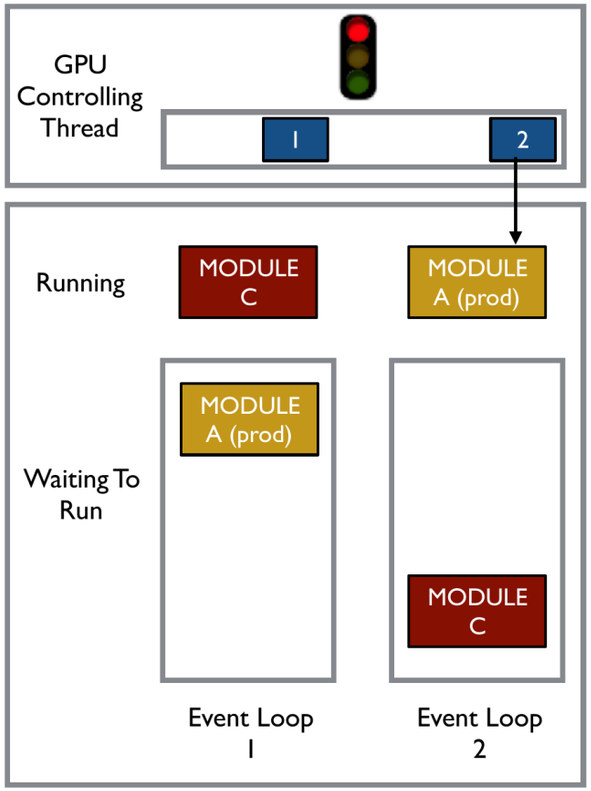
- Prototype of generic offloading mechanism integrated with framework
  - offload to GPU, FPGA, anther process

Produce

Produce method of module is called

Pulls data from buffer

Data used to create objects to put into event

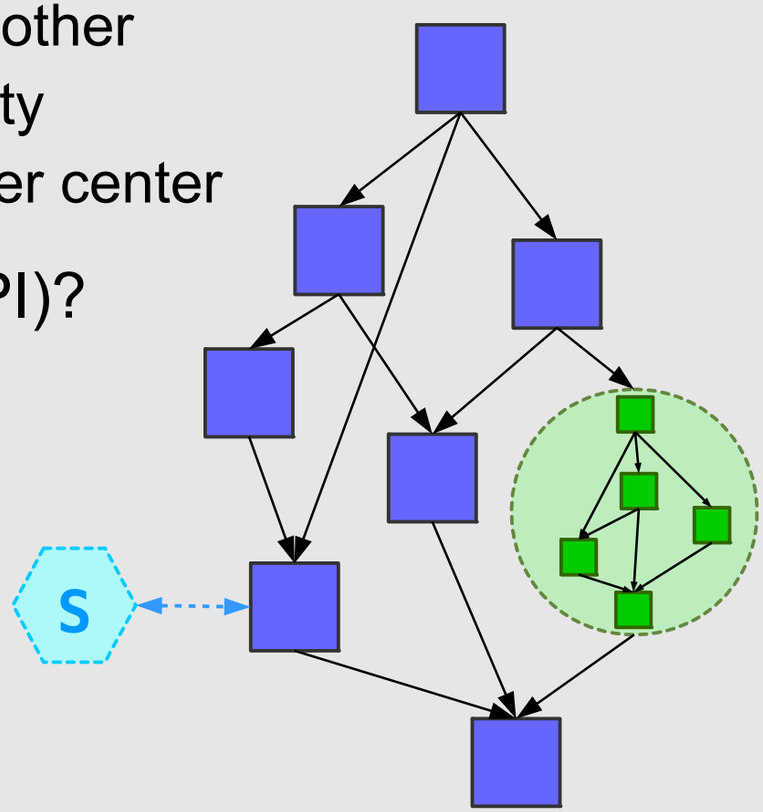


How To Keep A Hungry GPU Well Fed

- Let's assume we can execute a non insignificant percentage of our code on a GPU. How do we use it efficiently?
- Buffer data for many events, feed data to GPU all at once
  - what to do with CPU while waiting for basket of GPU work to be filled?
    - checkpoint?
    - start processing the next event?
      - » how many events can we process concurrently?
      - » w/ MT, say a few hundred MB per concurrent event
      - » high multiplicity events are better
- Receive data from outside node
  - intra-node sub-event concurrency
  - job scheduling becomes more complicated
  - CPU attached to GPU just does data/code scheduling
- Depends on CPU/GPU ratio

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



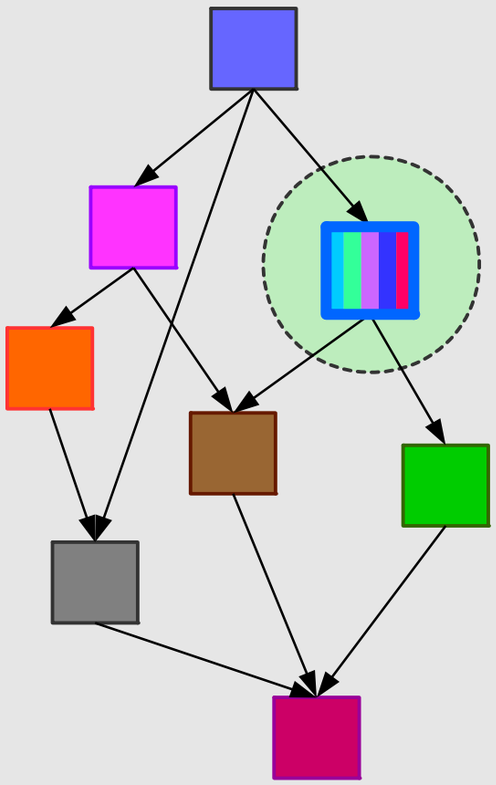
Implications for Frameworks

- True sub-event inter-nodal concurrency
  - both Algorithm/Task based
  - regional (*à la* ROI)

- What percentage of Algorithms/Tasks need data that's entirely encapsulated in a DataHandle?
  - this would be easiest to offload

- Algorithms that use "Service" data
  - is this static across many events, or only loaded at initialize()?
  - can it be easily encapsulated into the offloaded task?

  - or don't offload this type of task



Implications for I/O

- Need to be able to trigger conversion (*eg* bytestream) for individual data objects at arbitrary times.
  - is this something that's worth offloading?
- Need to accumulate/collect "bytestreamed" objects from multiple events in suitable baskets before shipping them to other nodes or offloading to co-processor.
- And of course, we need to do the reverse with data received
- Will probably never want to do this for online, due to latencies

Implications for Scheduling

- Scheduling becomes much harder
  - or possibly much easier
  - depends on level of data encapsulation

- If data fully encapsulated in a task, can more easily send any task to any node
  - can we use checkpointing to load appropriate code state into the node?

- If data is NOT fully encapsulated (eg, need event/run/lumi dependent Service data), want to schedule task that use the same Service configuration on the same nodes
  - benefit from bi-directional scheduling queries to determine what's already running on a node

- Partial events get shipped around. How to know that all bits have been fully assembled to declare than an event has finished processing?

- What happens if one part of one event in the middle of a 1000 event basket produces an error?

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?

Experiment Agnostic Common Problems

- Offloading Service to control external resources
  - integration with Job Scheduling Services to discover remote resources
- Algorithmic code conversion
- Efficient marshalling / unmarshalling of data to offload system
  - CUDA / GPU
  - MPI / zeroMQ
  - FPGA – High Level Synthesis (HLS), openCL, etc
  - Is it possible to abstract gory details?
- Common problems, but are there common solutions?
  - are we insufficiently alike?
  - very different EDMs
    - rewrite EDMs in more C-struct style?

Conclusions

- The next generation of HPCs are not our friends
  - we need a major paradigm shift (again!) in our event processing to be able to make efficient use of them
- If we can solve this problem, we'll be able to use a lot more diverse resources
- Alternatively, we say f-it, and leave a lot of silicon on the floor
  - depending on the complexity of the problem, and our CPU allocations/budget, this is not totally unreasonable
  - DayaBay only using 30% of cores on a Cori node, but they don't care
- Or we give up on HPCs entirely and invest in building out our grid infrastructure with machines that work better for us
  - cost of hardware vs. cost of people-power
  - depending on what industry does, may not have the option of renting cloud compute time
- Are we sure GPUs are really the future? If so, how long will they be the future?
  - don't want to code ourselves into a corner. how many times will we need to rewrite everything?