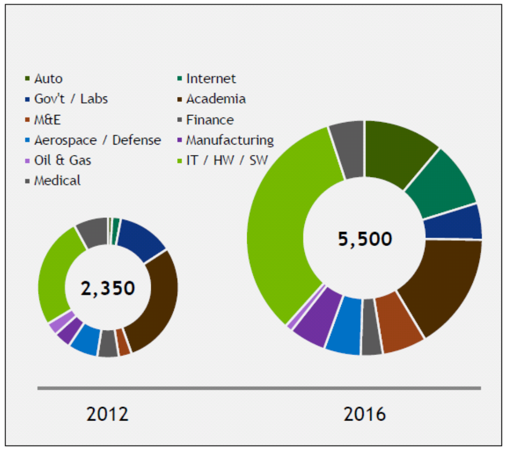
Applications of GPUs

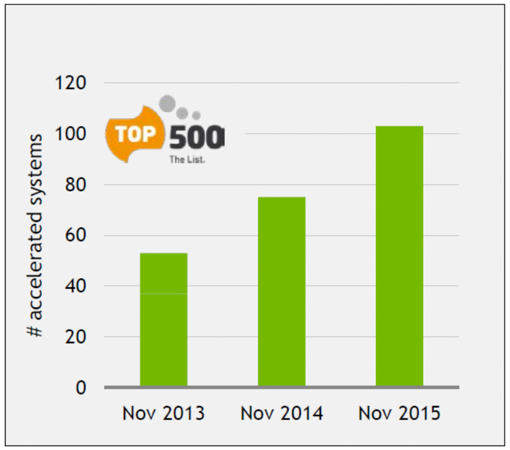
Felice Pantaleo

CERN – EP Department

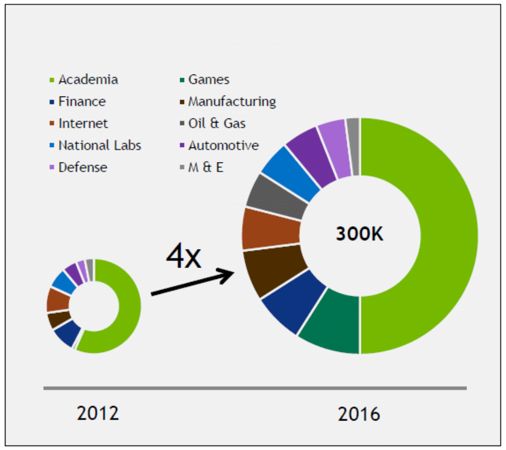
LEAPS IN ADOPTION



2X GTC Attendees



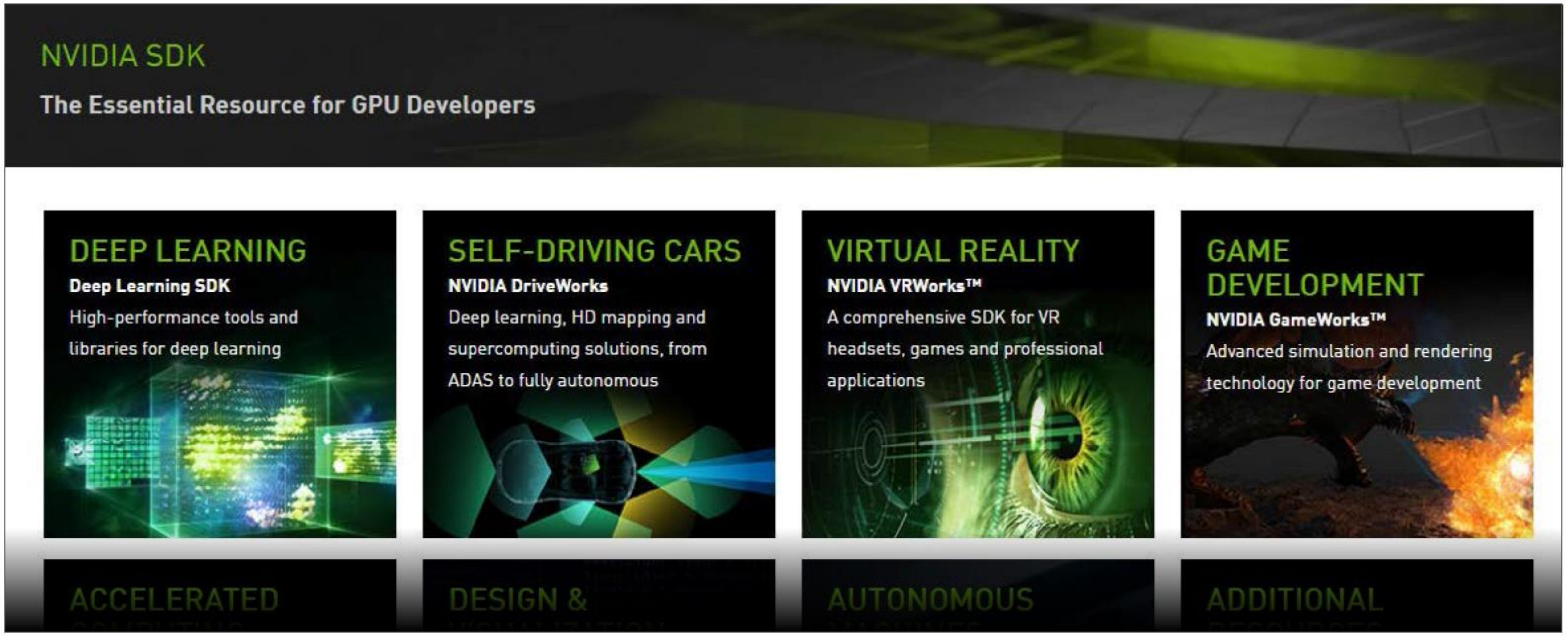
2X Accelerated Systems,
96% of New Systems on NVIDIA



4X CUDA Developers,
10X in Hyperscale + Auto

NVIDIA SDK

The Essential Resource for GPU Computing



NVIDIA SDK

The Essential Resource for GPU Developers

DEEP LEARNING
Deep Learning SDK
High-performance tools and libraries for deep learning

SELF-DRIVING CARS
NVIDIA DriveWorks
Deep learning, HD mapping and supercomputing solutions, from ADAS to fully autonomous

VIRTUAL REALITY
NVIDIA VRWorks™
A comprehensive SDK for VR headsets, games and professional applications

GAME DEVELOPMENT
NVIDIA GameWorks™
Advanced simulation and rendering technology for game development

ACCELERATED

DESIGN &

AUTONOMOUS

ADDITIONAL

developer.nvidia.com | Available Now

NVIDIA GAMEWORKS

Volumetric Lighting | Voxel Accelerated Ambient Occlusion | Hybrid Frustum Traced Shadows

Available Now

COMPUTEWORKS | GAMEWORKS | VRWORKS | DESIGNWORKS | DRIVEWORKS | JETPACK

PhysX

Hair**Works**

Wave**Works**

Flame**Works**

and other technologies such as:
Clothing, VXGI, Flex, Destruction

NVIDIA DESIGNWORKS

Adobe support of MDL | Siemens NX adopts Iray

COMPUTEWORKS GAMEWORKS VRWORKS DESIGNWORKS DRIVEWORKS JETPACK

Iray

MDL

OptiX

Path Rendering

and other technologies such as:
GL Extensions, GRID, GPU Direct for Video, Mosaic, VXGI, Warp and Blend

NVIDIA VRWORKS

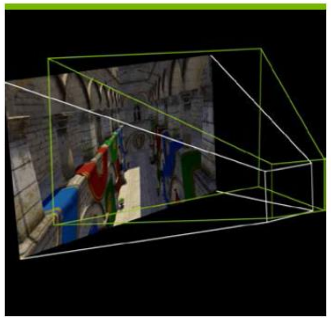
Oculus Rift and HTC Vive integration | Epic, Max Play and Unity game engines

Available Now

COMPUTEWORKS | GAMEWORKS | VRWORKS | DESIGNWORKS | DRIVEWORKS | JETPACK

Multi-Res Shading

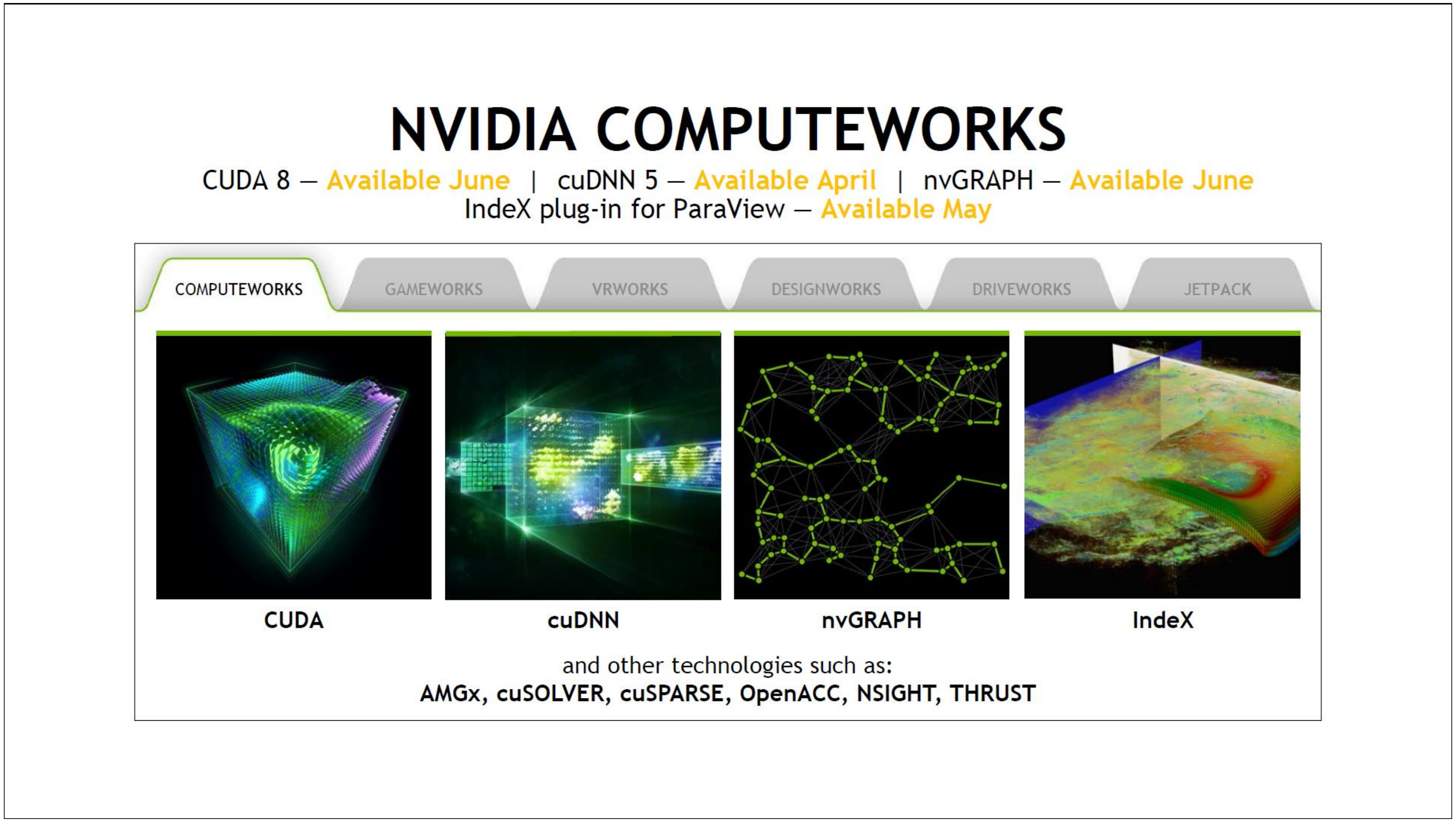
VR SLI


Context Priority

Warp and Blend

and other technologies such as:

Direct Mode, GPUDirect for Video



NVIDIA COMPUTEWORKS

CUDA 8 — Available June | cuDNN 5 — Available April | nvGRAPH — Available June
IndexX plug-in for ParaView — Available May

COMPUTEWORKS | GAMEWORKS | VRWORKS | DESIGNWORKS | DRIVEWORKS | JETPACK

CUDA | cuDNN | nvGRAPH | IndeX

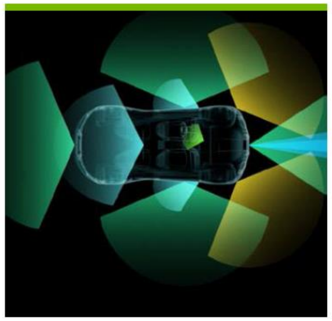
and other technologies such as:
AMGx, cuSOLVER, cuSPARSE, OpenACC, NSIGHT, THRUST

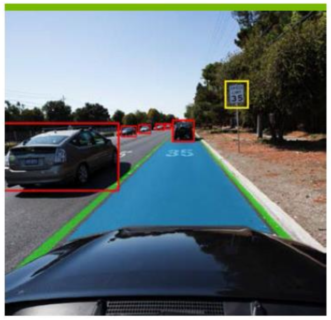
NVIDIA DRIVEWORKS

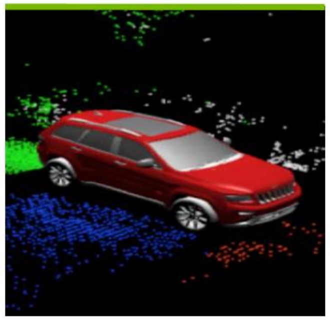
JPL — Available Now | EAP — Available Q2'16
General Release — Available Q1'17

UNDER CONSTRUCTION

COMPUTEWORKS GAMEWORKS VRWORKS DESIGNWORKS DRIVEWORKS JETPACK


SensorFusion


Detection


Localization

HD Maps

and other technologies such as:
Driving, Planning

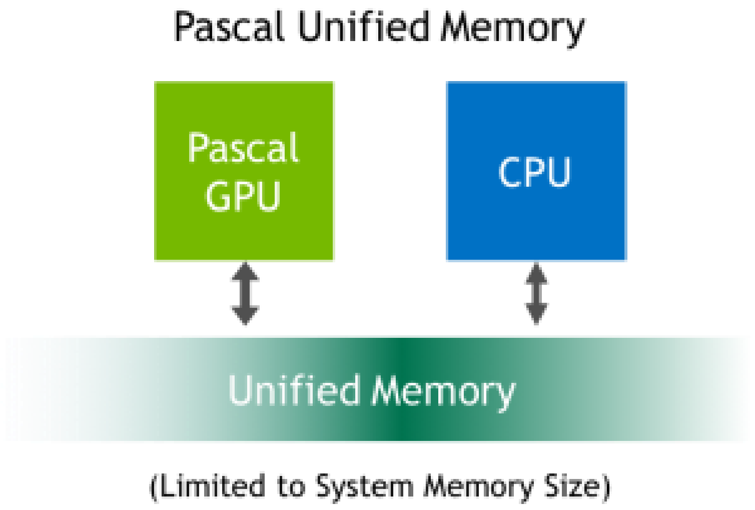
WORLD'S FIRST AUTONOMOUS CAR RACE

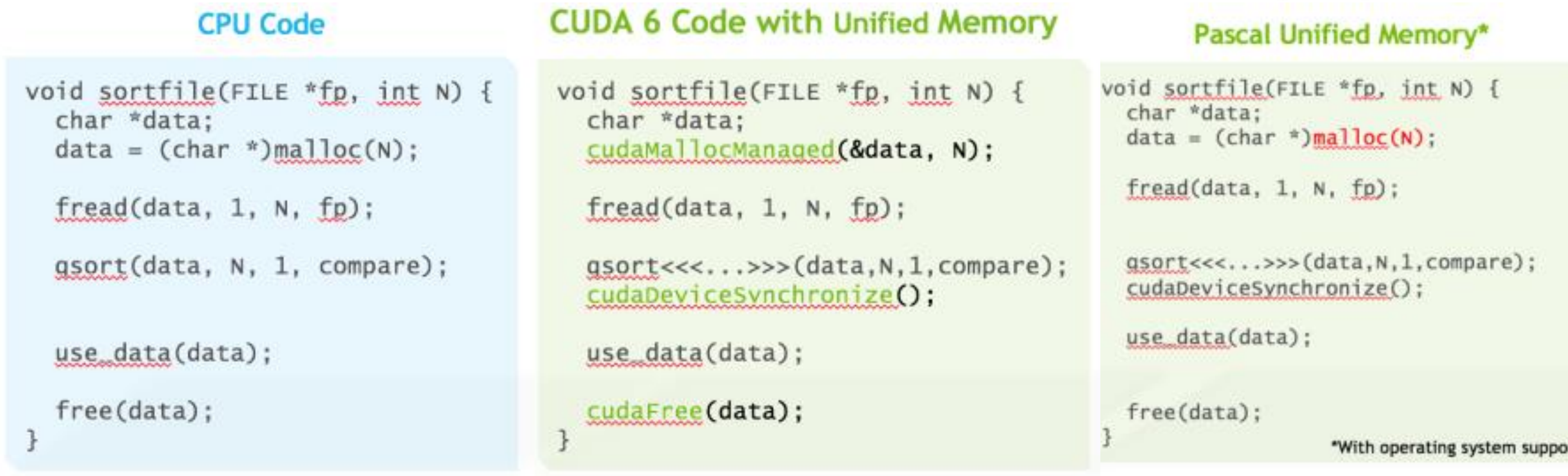
10 teams, 20 identical cars | DRIVE PX 2: The "brain" of every car | 2016/17 Formula E season

ROBORACE

CUDA 8

- New Unified Memory capabilities
- Powerful new profiling capabilities
- Improved compiler performance and heterogeneous lambda support





CPU Code

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

CUDA 6 Code with Unified Memory

```
void sortfile(FILE *fp, int N) {
  char *data;
  cudaMallocManaged(&data, N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  cudaFree(data);
}
```

Pascal Unified Memory*

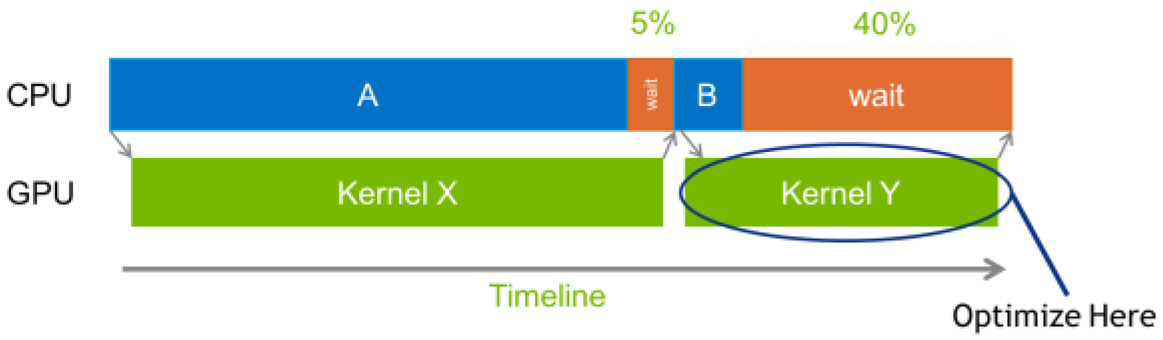
```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);


  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);


  free(data);
}
```

*With operating system suppo

With operating system support, Pascal is capable of supporting unified memory with the default system allocator. Here, malloc is all that is needed to allocate memory accessible from any CPU or GPU in the system

With page faulting on GP100, locality can be ensured even for programs with sparse data access, where the pages accessed by the CPU or GPU cannot be known ahead of time, and where the CPU and GPU access parts of the same array allocations simultaneously.

Dependency Analysis



experimental lambda support

```
void saxpy(float *x, float *y, float a, int N) {
    using namespace thrust;
    auto r = counting_iterator(0);

    auto lambda = [=] __host__ __device__ (int i) {
      y[i] = a * x[i] + y[i];
    };

    if(N > gpuThreshold)
      for_each(device, r, r+N, lambda);
    else
      for_each(host, r, r+N, lambda);
}
```

Programming GPUs with compiler directives

GPU Programming

Accelerators by directives

- CUDA can be low-level and closely coupled to the GPU
- With CUDA, the user needs to write specialist kernels:
  - - Hard to write and debug
  - - Hard to optimise for specific GPU
  - - Hard to maintain
  - + Performance boost
  - + Flexible
- Programming GPUs using directives could be an option if the user already knows how to program parallel processors
- Knowledge of thread synchronization mechanisms and contention avoidance techniques still required

Memory and Execution Models

Execution model:

- Concept of an accelerator region
- Accelerator task tied to accelerator it starts on
- Complete at known locations, barriers, “acc_sync”, program exit
- Data motion directives provide hints to the compiler on where to place data that accelerator regions access

Examples

OpenMP 4.0

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute
for (i=0; i<N; i += num_blocks)
     #pragma omp parallel for
     for (b = i; b < i+num_blocks; b++)
          B[b] += sin(B[b]);
```

OpenACC

```
#pragma acc parallel copy(B[0:N]) num_gangs(numblocks)\
                     vector_length(bsize)
#pragma acc loop gang vector
for (i=0; i<N; ++i) {
       B[i] += sin(B[i]);
}
```

Libraries

GPU Programming

Thrust

- Parallel algorithms and data structures
- Based on the standard C++
- Can perform different operations on the GPU like:
  - Sort
  - Scan
  - Transform
  - Reductions
- Easy to integrate in an existing software framework
- Implementation details invisible to the user
- Info: https://developer.nvidia.com/thrust

Thrust samples

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
#include <cstdlib>

int main(void)
{
    // generate 32M random numbers on the host
    thrust::host_vector<int> h_vec(32 << 20);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer data to the device
    thrust::device_vector<int> d_vec = h_vec;

    // sort data on the device (846M keys per second on GeForce GTX 480)
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```

Thrust samples (ctd.)

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>
#include <cstdlib>

int main(void)
{
  // generate random data on the host
  thrust::host_vector<int> h_vec(100);
  thrust::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer to device and compute sum
  thrust::device_vector<int> d_vec = h_vec;
  int x = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::plus<int>());
  return 0;
}
```

ArrayFire

- Interface for C, C++, Java, R, Fortran
- Supports CUDA and OpenCL (NVIDIA GPUs, Xeon Phi, CPUs x86, ARM)
- API based on the concept of Array
- Contains hundreds of functions: arithmetic, linear algebra, statistics, signal processing, image processing, and related algorithms
- It can execute loop iterations in parallel with gfor

ArrayFire sample

- gfor distributes all the iterations of a for-loop to the GPU threads

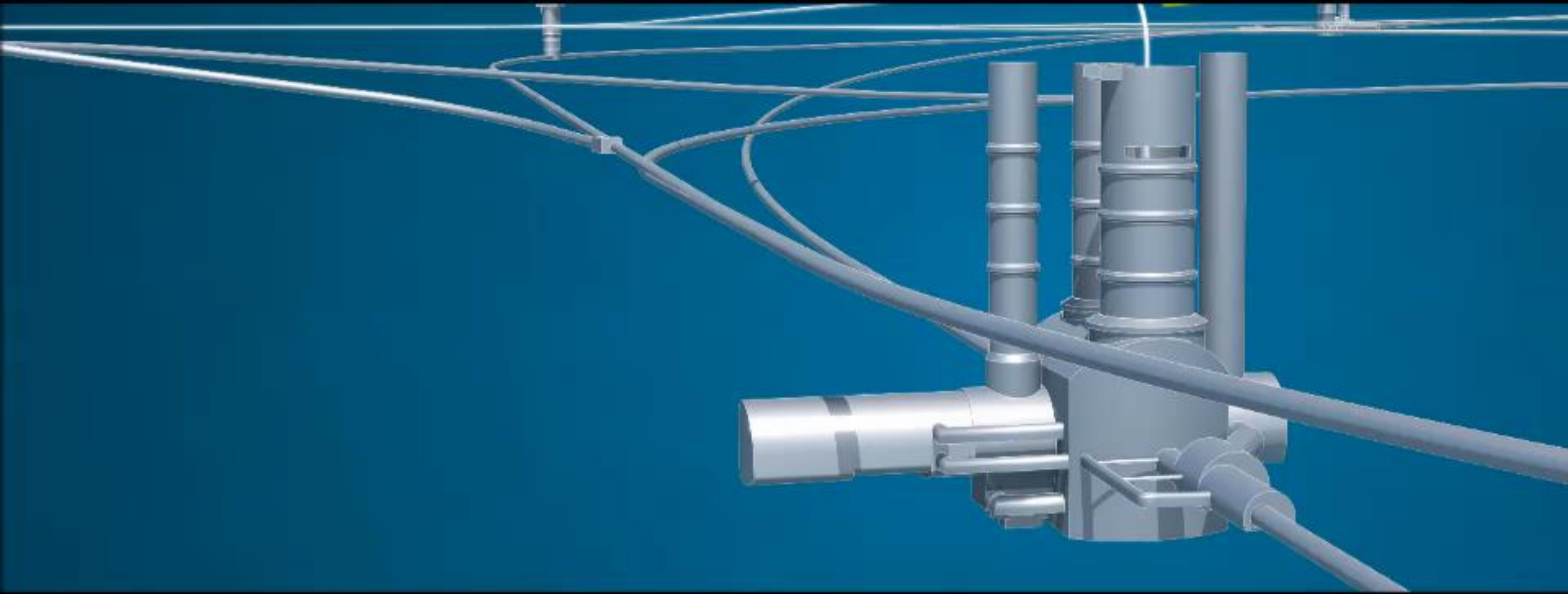
```
for (int i = 0; i < n; ++i)
    A(i) = A(i) + 1;
```

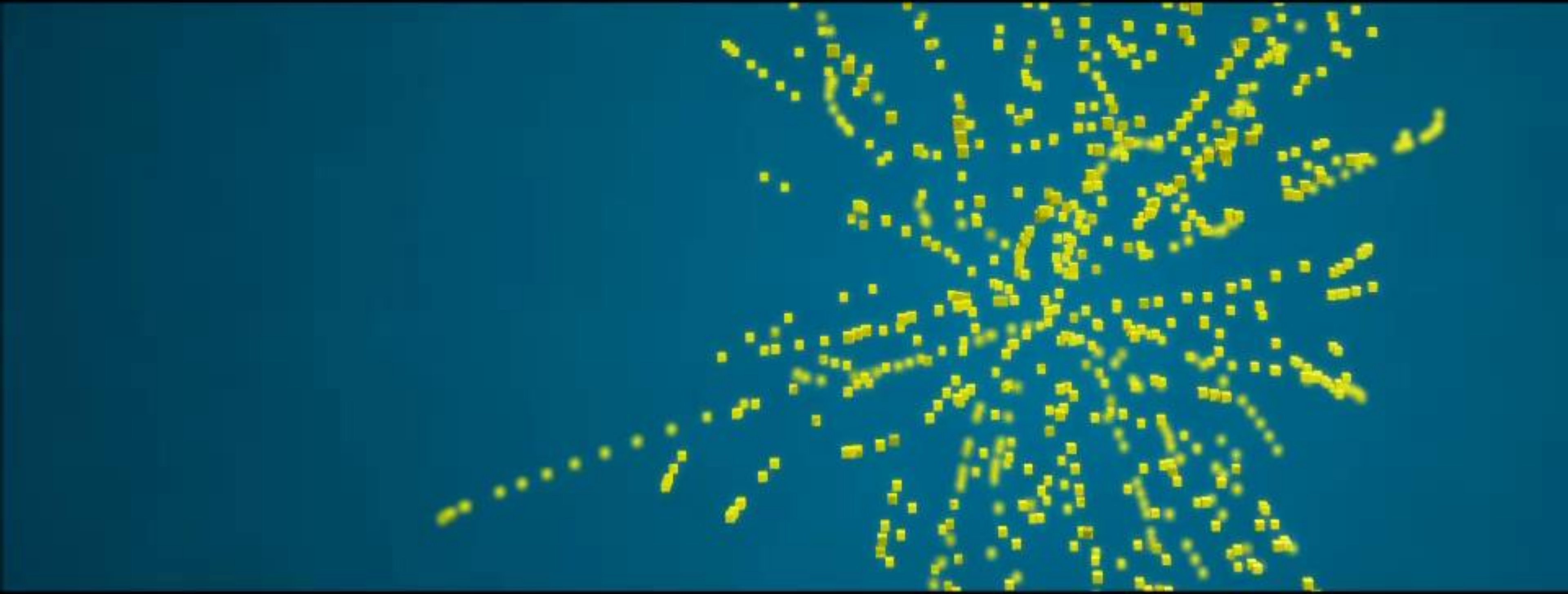
ArrayFire sample

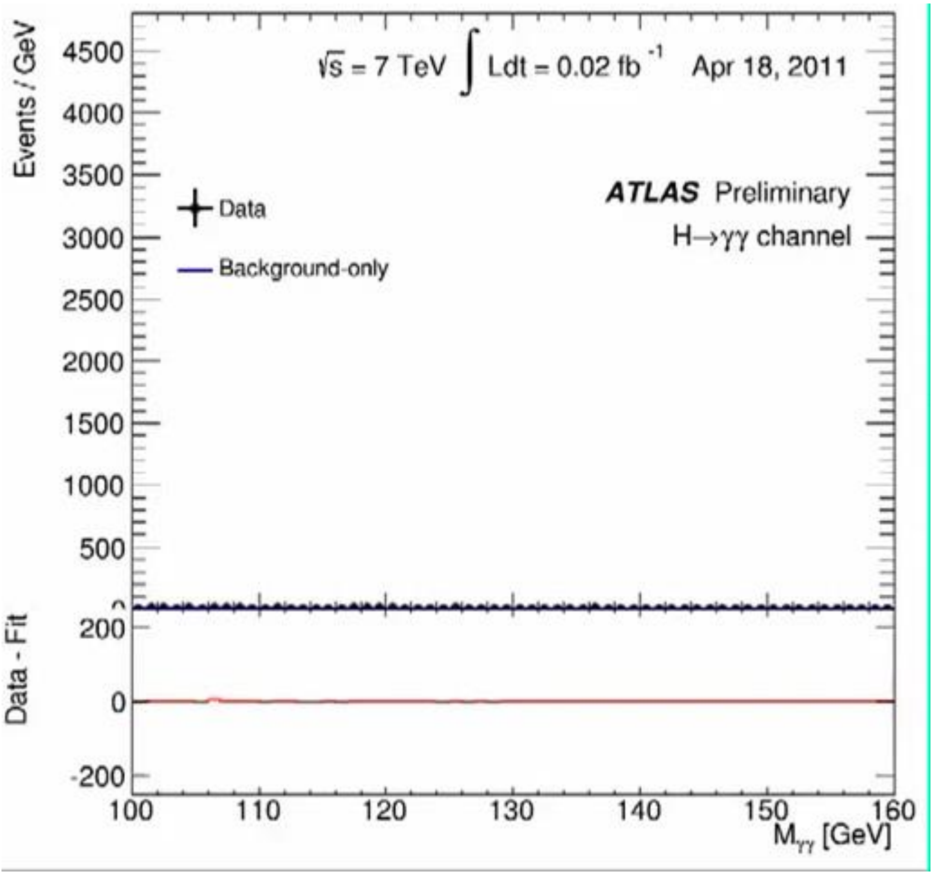
- gfor distributes all the iterations of a for-loop to the GPU threads

```
gfor (array i, n)
     A(i) = A(i) + 1;
```

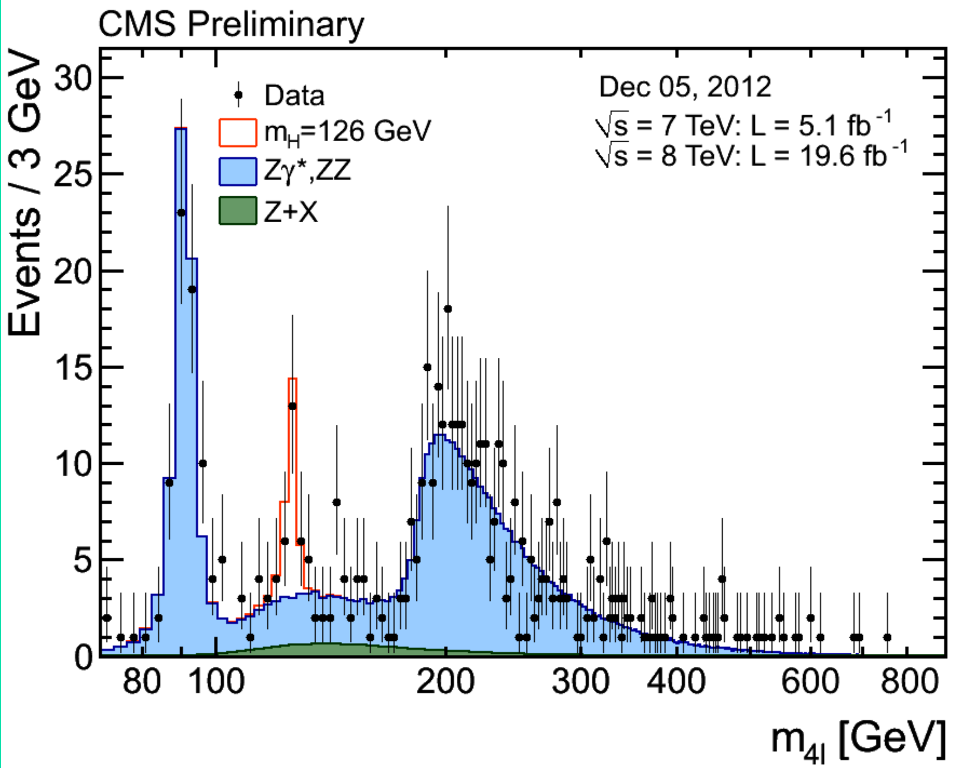
In HEP case, data would need to be transformed into Structures of arrays, in order to enable efficient parallelism and memory access pattern.







$\sqrt{s}$ = 7 TeV ∫ Ldt = 0.02 fb^{-1} Apr 18, 2011
ATLAS Preliminary
H→γγ channel
Data
Background-only
Events / GeV
Data - Fit
$M_{\gamma\gamma}$ [GeV]



CMS Preliminary
Dec 05, 2012
$\sqrt{s}$ = 7 TeV: L = 5.1 fb^{-1}
$\sqrt{s}$ = 8 TeV: L = 19.6 fb^{-1}
Data
m_H=126 GeV
Zγ*,ZZ
Z+X
Events / 3 GeV
m_{4l} [GeV]

Is there a place for GPUs in all this?

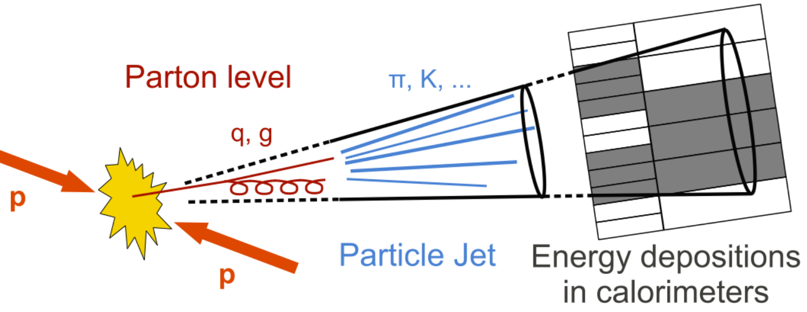
- At trigger level:
  - Controlled environment
  - High throughput density required
- On the WLCG:
  - Software running on very different/diverse hardware
    - Starting from Pentium 4 to Broadwell
  - Today's philosophy consists in "one size fit all"
    - Legacy software runs on both legacy and new hardware
  - Experiments pushing to higher and higher data rates
  - WLCG strategy: live within ~fixed budgets
  - Make better use of resources: **the approach is changing**
- Power consumption is becoming a hot-spot in the total bill
  - Especially in European Data Centers
- This will be even more important with the HL-LHC upgrade
  - Cope with 2-3x the amount of data

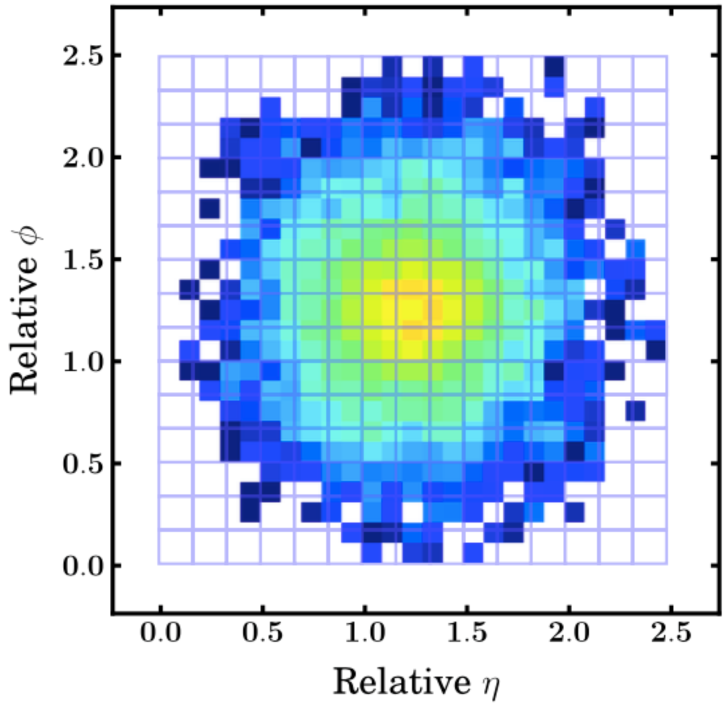
Deep Neural Networks @LHC

- Growing interest @CERN in Deep Neural Network applications for LHC experiments
- Assess data quality with trained algorithms (replacing as much as possible 24/7 shifts of humans)
- Take fast decisions (keep or reject the event) @ trigger w/o having to reconstruct the full event (save fraction of online CPU for other usages)
- Jet identification (see next slide)
- DNN training naturally happens on GPUs. Currently investigating the possibility of creating O(50) GPU clusters @CERN
  - give people access to GPUs for small-scale project
  - will increase usage of GPUs between experiment average "users"
  - might serve as a seed for a larger cluster on the time scale of HL-LHC (where DNN might become integrated in our standard reconstruction software)

Jet Imagining with DNN

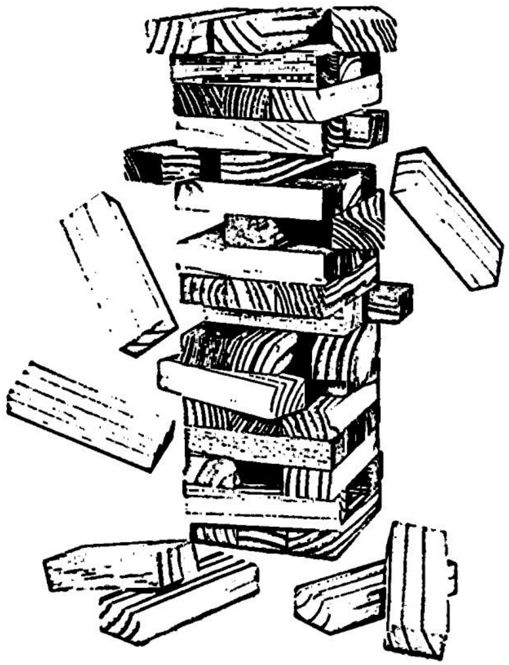
- A Jet is a shower of stable particles initiated by an unstable jet "mother particle" (a quark, a gluon, a Higgs boson, etc)
- Jet tagging is the identification of the nature of the jet mother from the features of the jet
- Normally, jets are
  - reconstructed with custom/physics-driven unsupervised algorithms (jet algorithms)
  - identified with supervised ML algorithms (BDT, NN, …)
- Ongoing work to explore DNN, e.g. "recycling" progresses on image processing
- Growing interest in advanced DNN → Growing interest in having GPU clusters at hand for training





See DS@LHC Talk

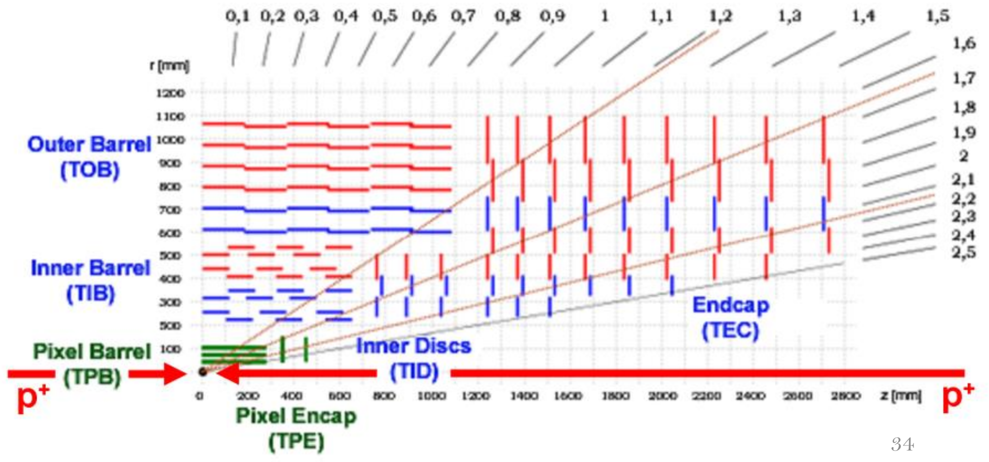
PATATRACK



Tracking at CMS

- Particles produced in the collisions leave traces (hits) as they fly through the detector
- The innermost detector of CMS is called **Tracker**
- **Tracking**: the art of associate each hit to the particle that left it
- The collection of all the hits left by the same particle in the tracker along with some additional information (e.g. momentum, charge) defines a **track**
- **Pile-up**: # of p-p collisions per bunch crossing

CMS
Compact Muon Solenoid
Solénoïde compact pour muons

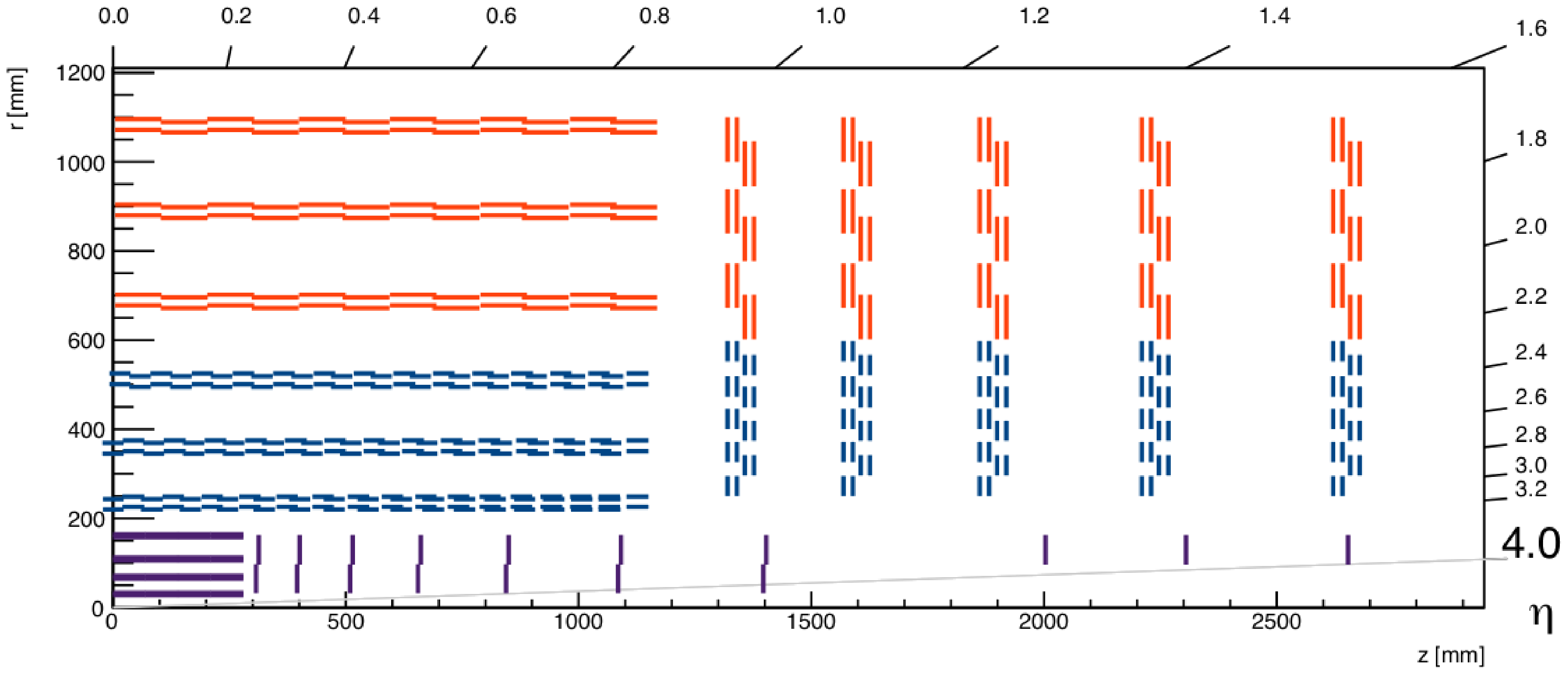


PATATRACK

- PATATRACK
  - It is a hybrid software to run on heterogeneous HPC platforms for emulating a GPU-based track trigger, data transfer and synchronization
- Tracker data partitioning
  - Fast simulation on fast geometry and uniform magnetic field
  - The information produced by the whole tracker cannot be processed by one GPU in a trigger environment
    - However this is possible at HLT and Reconstruction stages
- Low-latency data transfers between network interfaces and multiple GPUs (GPU Direct)
- Parallel Algorithms
- Cellular Automaton executes completely in-cache for lowest latency

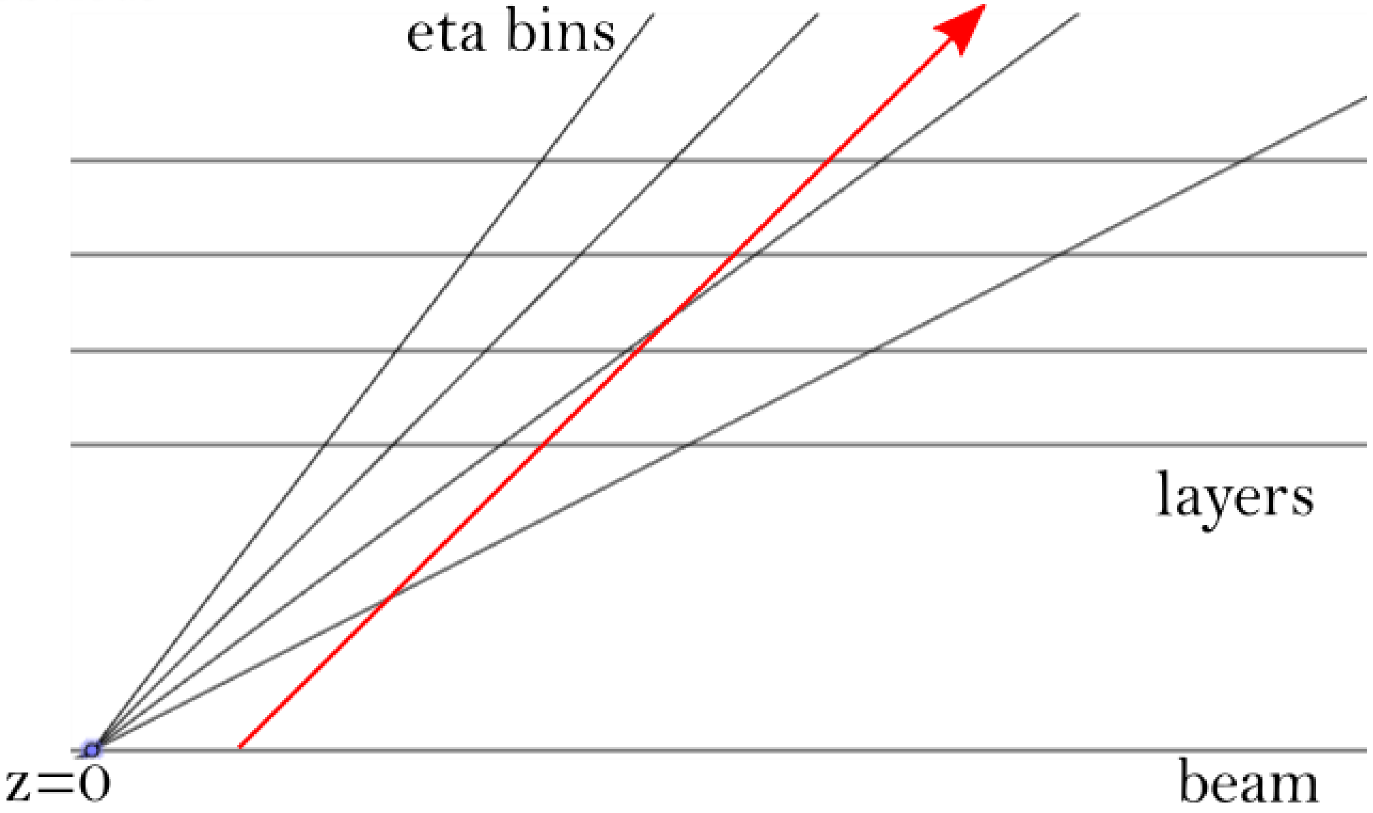
Partitioning

- Tracks ~straight if seen from a longitudinal perspective (z,R) plane
- Number of tracks approx. uniform in η



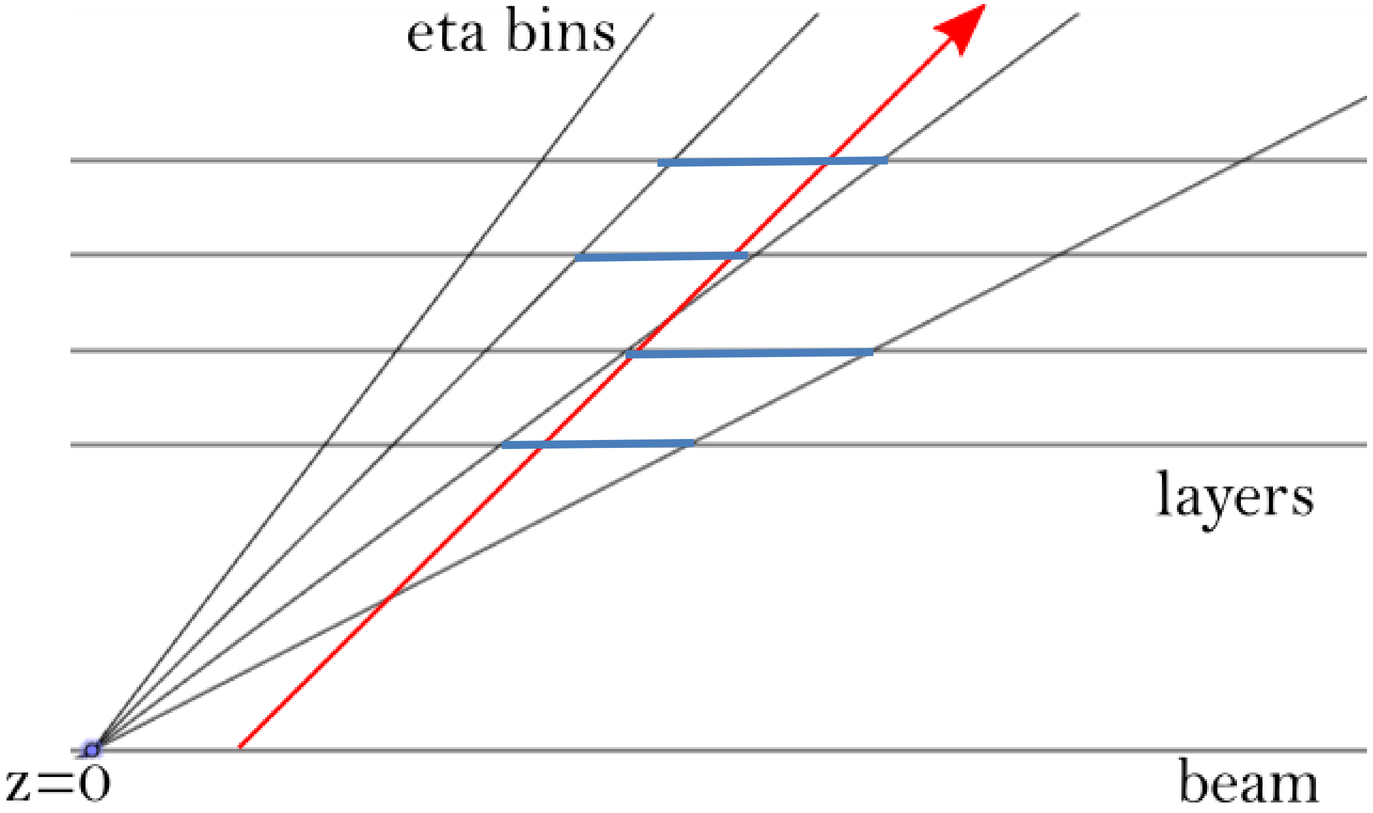
Partitioning (ctd.)

- Eta bins could have been treated independently
  - Pile-up and longitudinal impact parameter (displacement of the collision point along the z-axis) limit this hypothesis
  - Area on the next layer that needs to be scanned for compatible Stubs searching not obvious



Partitioning (ctd.)

- Simulation for different longitudinal impact parameters
- Lists of segments on subsequent layers evaluated beforehand
- Each streaming multiprocessor on a GPU is in charge of one list



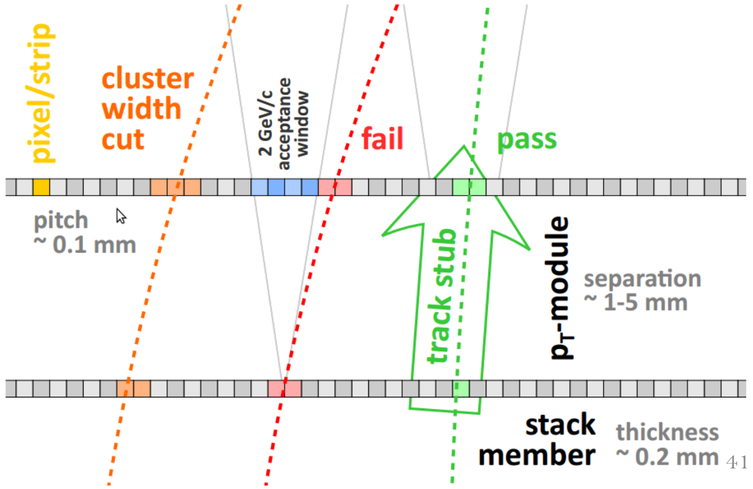
In-Shared-Memory Cellular Automaton

Mondrian of Life

Input Data

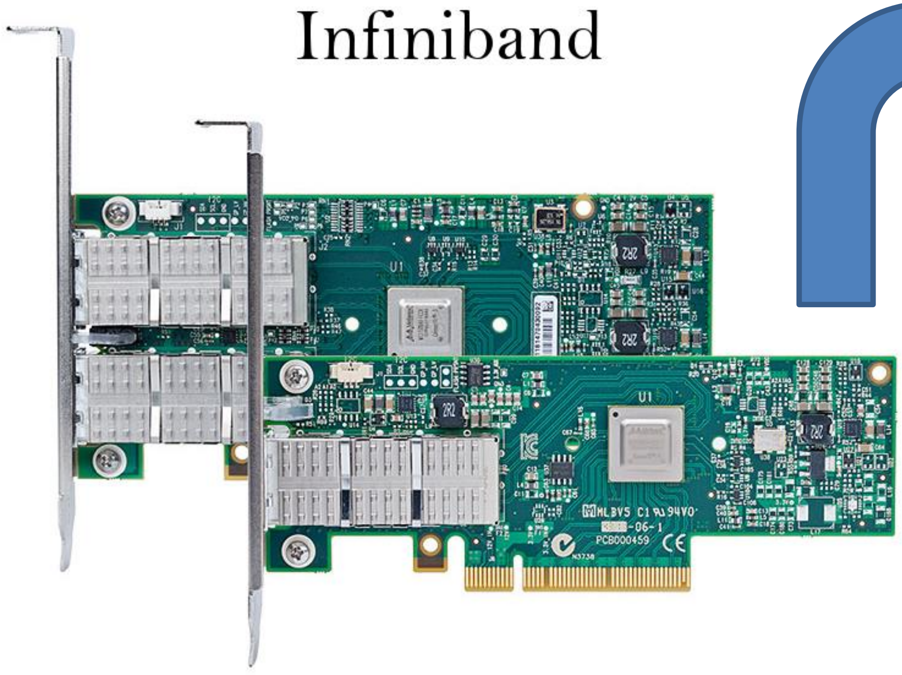
- The input data required by the Cellular Automaton Algorithm is a array of stubs contained in a η-chain in a specific ϕ–sector
- In order to minimize the number of messages sent, this array-payload is wrapped around a custom communication protocol header
- This header contains :
  - the ID of the packet
  - the size in # of stubs
  - the number of layers contained in the packet
  - the offset of the first stub in a specific layer wrt to the beginning of the payload

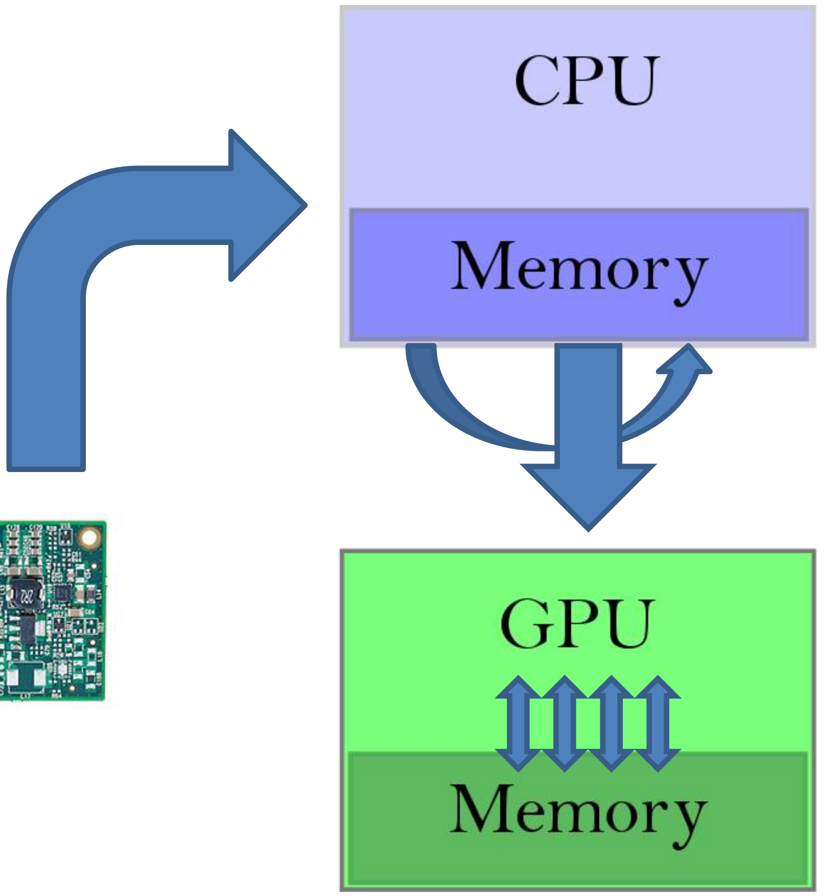
- This packet is then passed as an array of MPI_BYTE type (a.k.a. unsigned char)
- Memory pre-allocated on the GPU to host a packet received from the NIC and loaded directly into the GPU global memory



Data movement without GPUDirect

- Stubs from each chain are sent to a different GPU Streaming Multiprocessor/different node where the kernel runs completely in shared memory

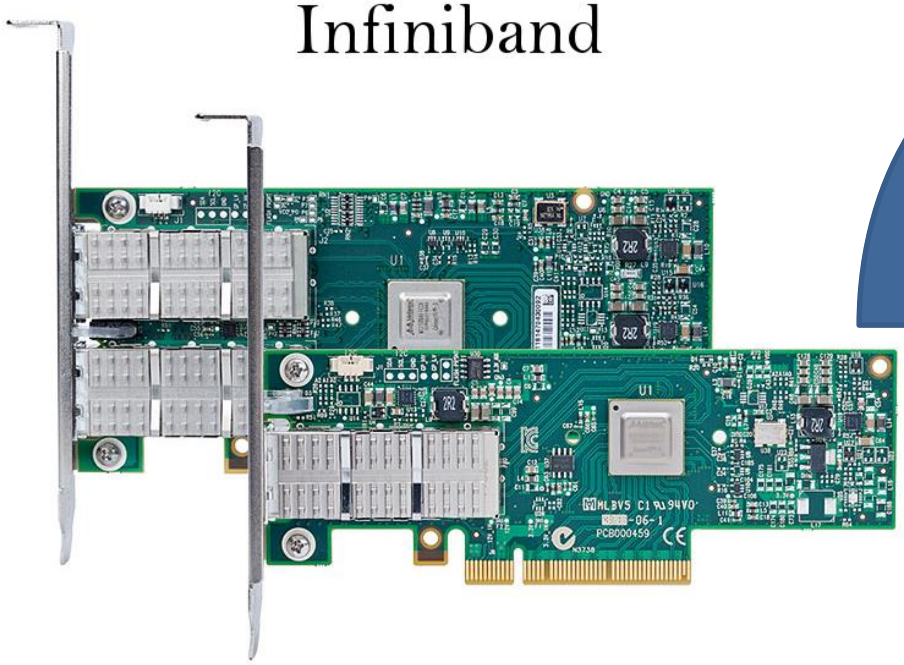


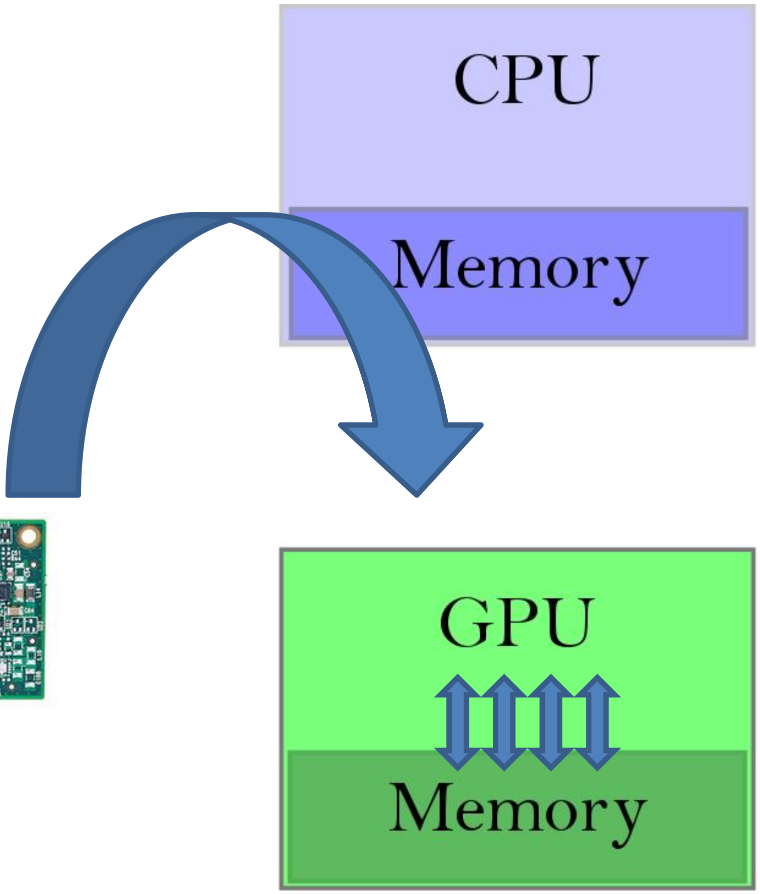


Data movement with GPUDirect

- GPUDirect accelerated communication with network and storage devices
- GPUDirect supports RDMA allowing latencies ~1us and link bandwidth ~7GB/s

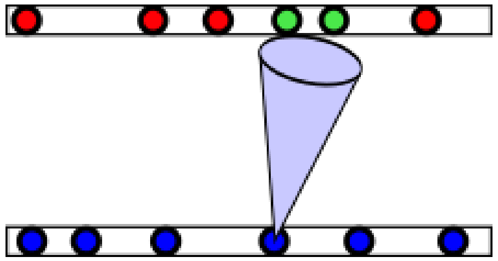
Infiniband



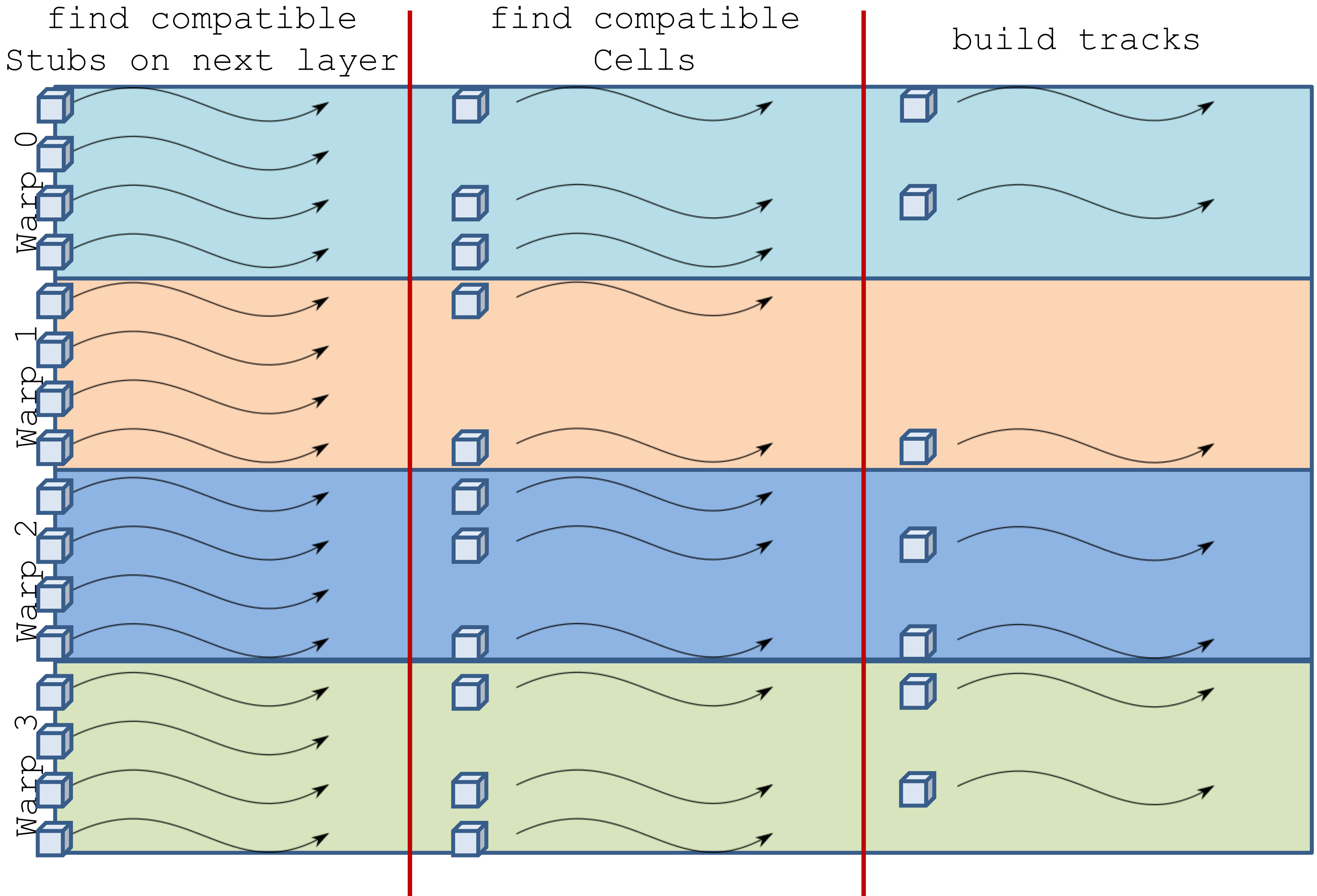


Cell creations

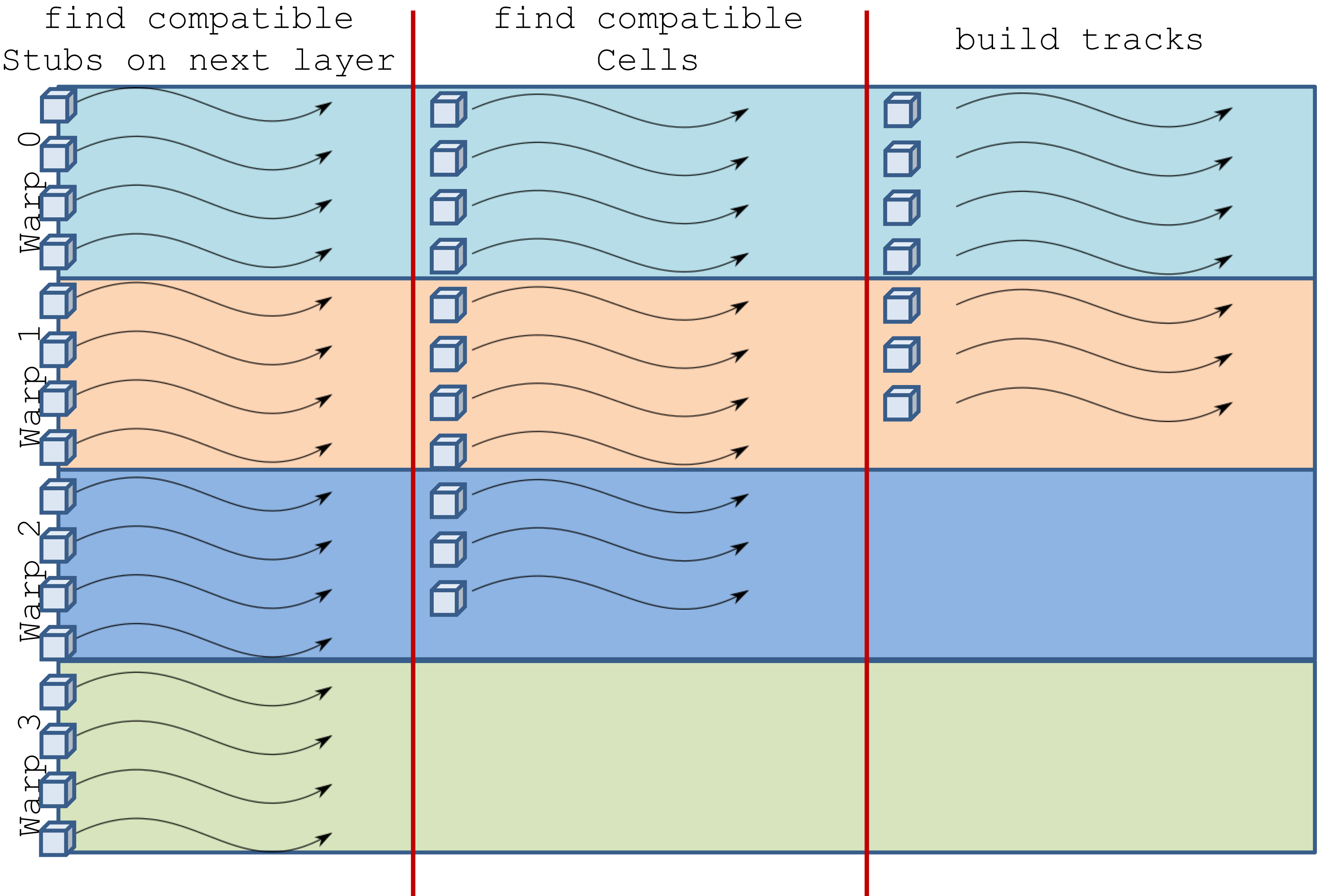
- A Cell is a class containing information about:
  - Two stubs
  - Neighboring Cells on the following layers
  - Neighboring Cells on the previous layers
- It can be constructed starting from a doublet of Stubs with similar η and ϕ
- In principle, the creation of Stubs can be executed in parallel:
  - 1024 CUDA Threads are spawned (32 warps)
  - Each warp takes one Stub on a layer and looks for a Stub in η-ϕ window (in parallel in groups of 32)
  - Every time a stub with similar eta/phi is found a new cell is created
  - Shared memory for cells is pre-allocated
  - Created cells are visible to all the threads



Mitigating Branch Divergence



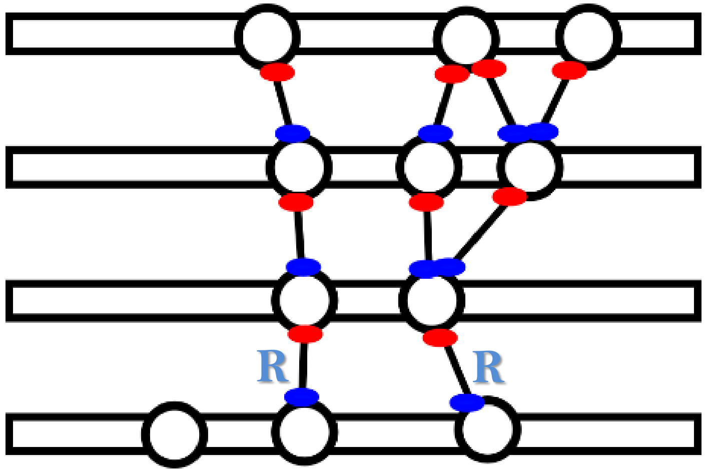
Mitigating Branch Divergence



Neighbor finding

- We have now a CUDAQueue that contains all the Cells
- Threads are now reshuffled
  - each thread is now associated to a Cell independently of the Cell's layer
  - If two Cells have one Stub in common and their η and ϕ are similar they are become Inner and Outer Neighbors one of the other

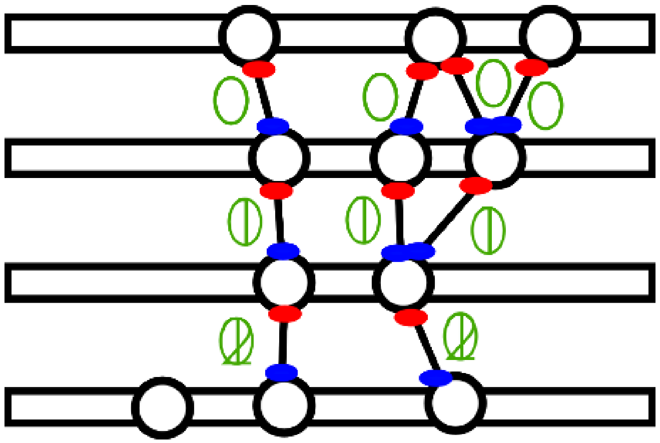
- This step will create a graph of interconnected Cells
- Now it's time to play Game of Life and make the graph evolve!



Evolution

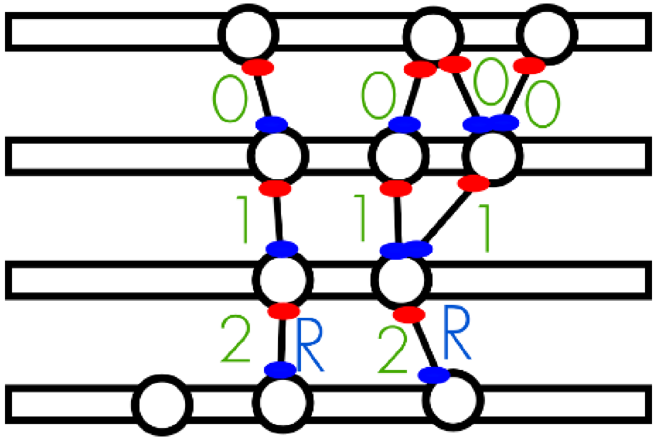
- In the evolution stage each thread is associated to one Cell
- In order for evolution to start all the Cells' State is set to 0
- The number of generations to evolve is set to (number of layers – 2)
- At each time step, each thread:
  - For each Cell, if there is at least a outer neighbor sharing the same state, its state is increasing by 1.

T=0



Track building

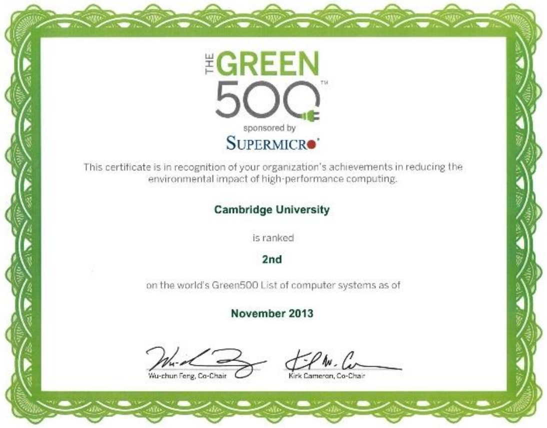
- Now that the evolution stage has completed each thread checks whether its Cell is a Root Cell i.e.:
  - It has no inner neighbor
  - Its state is higher than a threshold
- Found Root Cells are hence pushed in a CUDAQueue
- Each thread is associated to a Root Cell and performs a recursive Depth First Search on the Cells Graph
  - While traversing its trie, a thread adds a cell to a track if the Cell state is decreasing



Tests and Results

Wilkes - Acknowledgement

- Wilkes, technical specifications:
  - 128 node Dell™ PowerEdge™ T620
  - Dual-Socket Hexa-Core Intel E5-2630 (1536 cores in total)
  - 64 GByte memory per node
  - Scientific Linux release 6.6 (Carbon)
  - MLNX_OFED 2.4-1.0.4 SW stack
  - Dual-rail Mellanox Connect-IB FDR InfiniBand
  - NVIDIA® Tesla K20 GPUs (2 GPUs per node)
- Wilkes, facts:
  - The UK's fastest academic cluster, deployed November 2013
  - Specific design to allow GPU Direct over RDMA from both GPU within the node
  - GPU LINPACK performance of ~240 TF
  - Ranked second in the worldwide Green500 ranking Nov 2013 (3,631.86 MFLOPS/W)


THE GREEN 500™
sponsored by SUPERMICRO

This certificate is in recognition of your organization's achievements in reducing the environmental impact of high-performance computing.

Cambridge University

is ranked

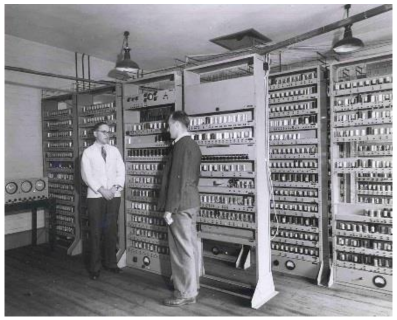
2nd

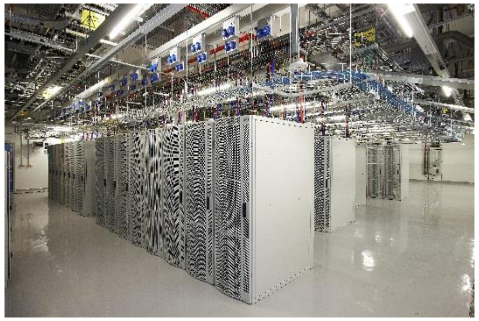
on the world's Green500 List of computer systems as of

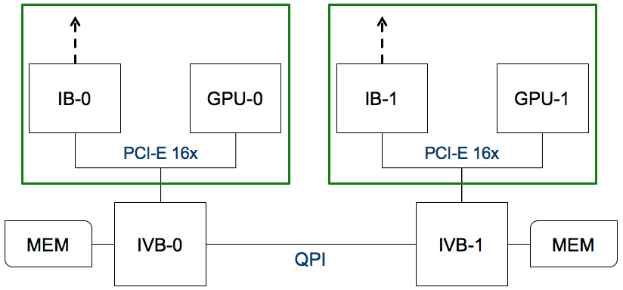
November 2013

[illegible]
Wu-chun Feng, Co-Chair

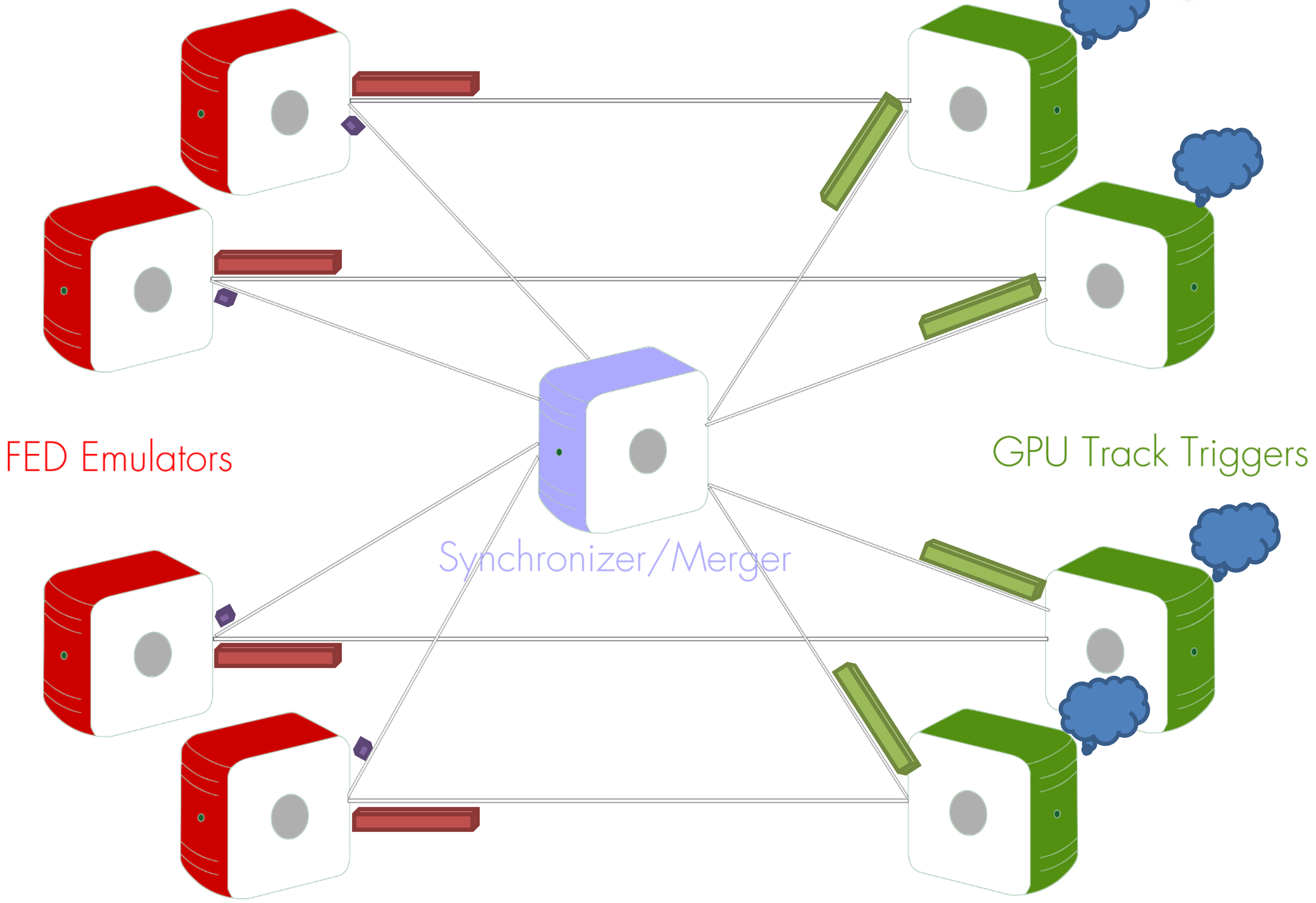
[illegible]
Kirk Cameron, Co-Chair







Test Configuration

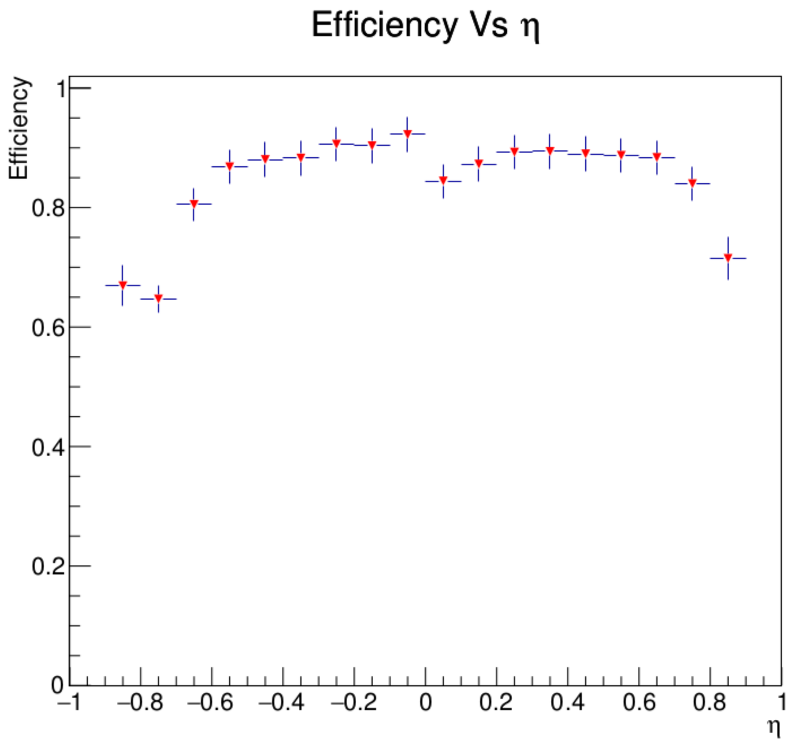


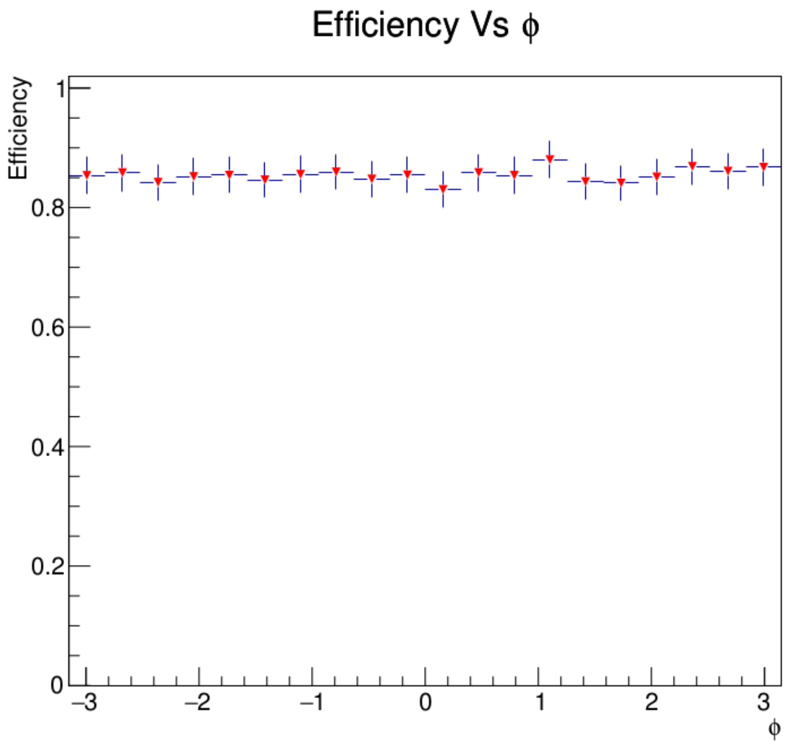
Tests

- Detector divided in:
  - Trasversal section: 8 sectors, $\Delta\phi = \pi/2$, each overlapping with the next one by $\pi/4$
  - Longitudinal section: 8 sectors, $\Delta\eta=0.8$
- Requirements for the creation of a Cell: $\Delta\eta < 0.1$, $\Delta\phi < 0.12$
- Requirements for two cells to be neighbors: $\Delta\eta < 0.1$, $\Delta\phi < 0.12$
- 40 π guns:
  - 2 GeV < pT < 100 GeV
  - $-1 < \eta < +1$

Results

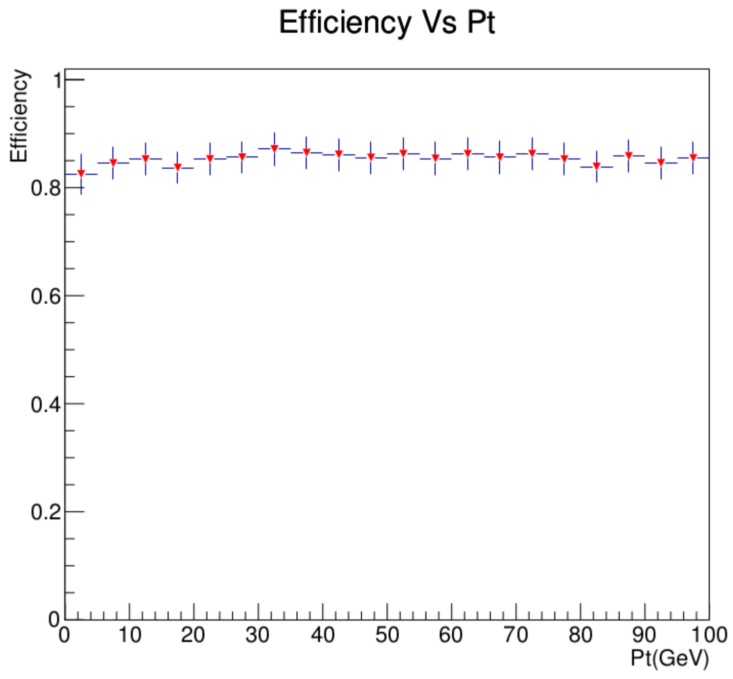
- The efficiency of the algorithm could be improved by extending the neighbor search to the second next layer
- Avg. efficiency: 0.81

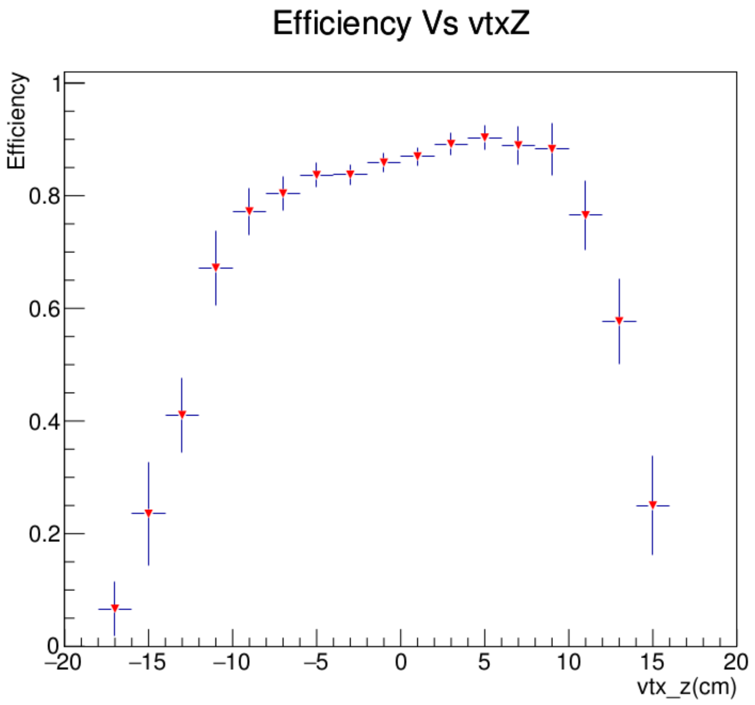




Results

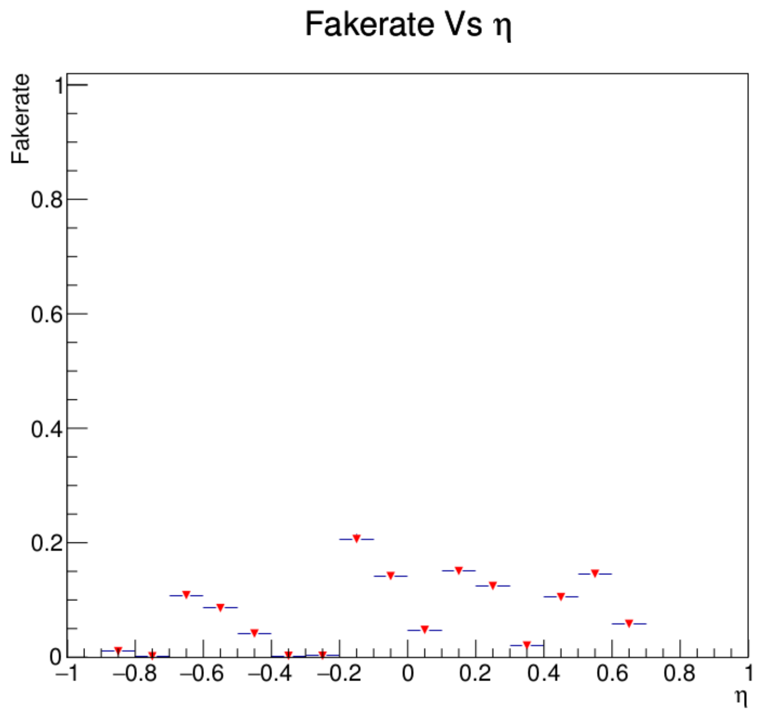
- The efficiency of the algorithm could be improved by extending the neighbor search to the second next layer
- Avg. efficiency: 0.81

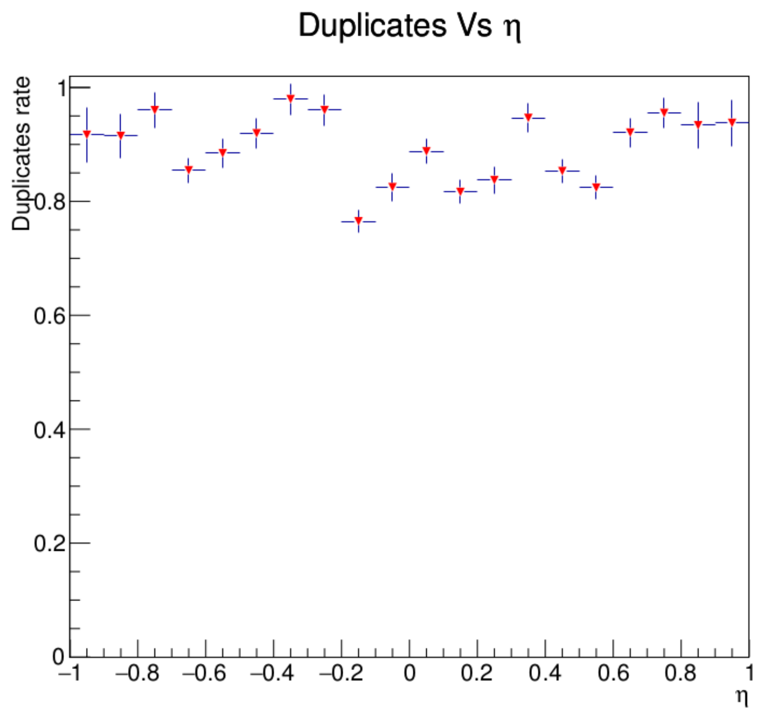




Results

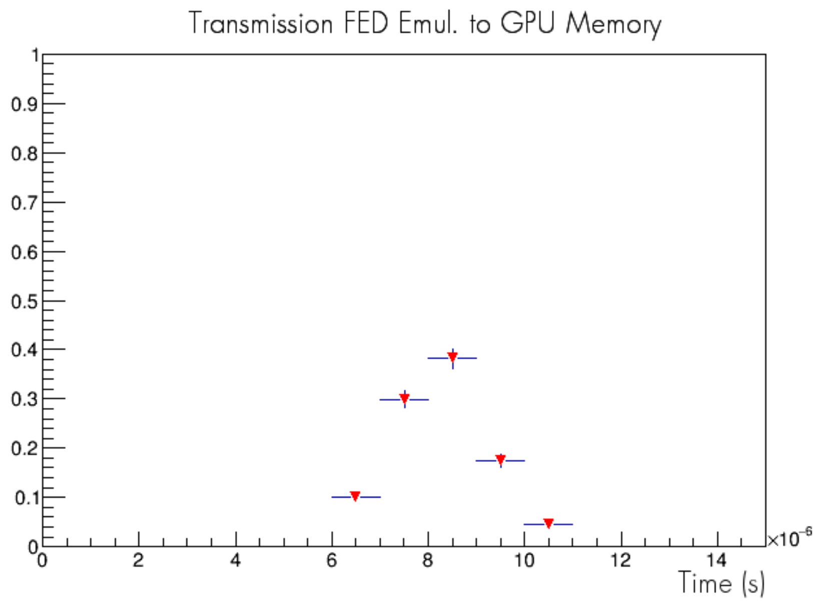
- The fake rate could be reduced by adding further conditions to the formation of the tracks, or by adding another step
- The duplicate rate is mainly due to the overlapping regions and can hence be reduced further

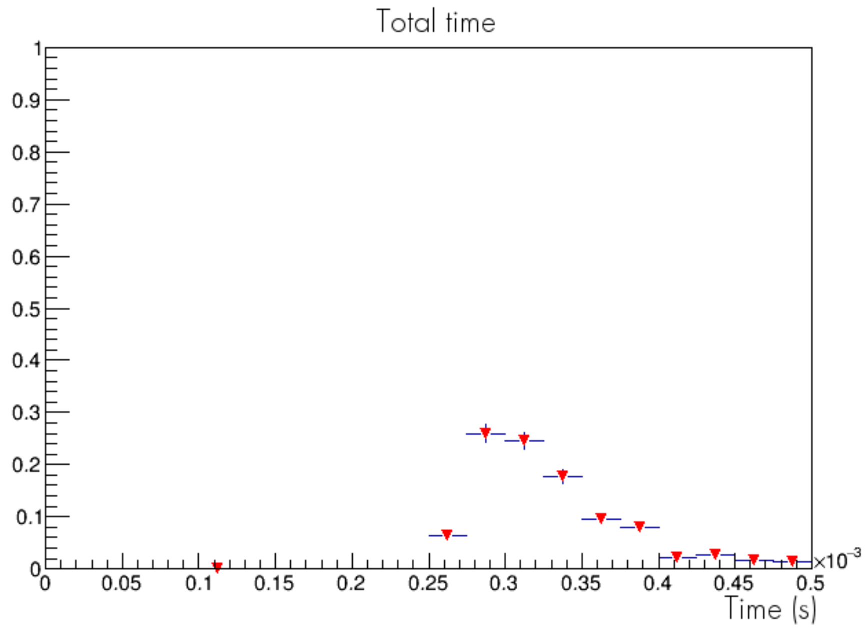


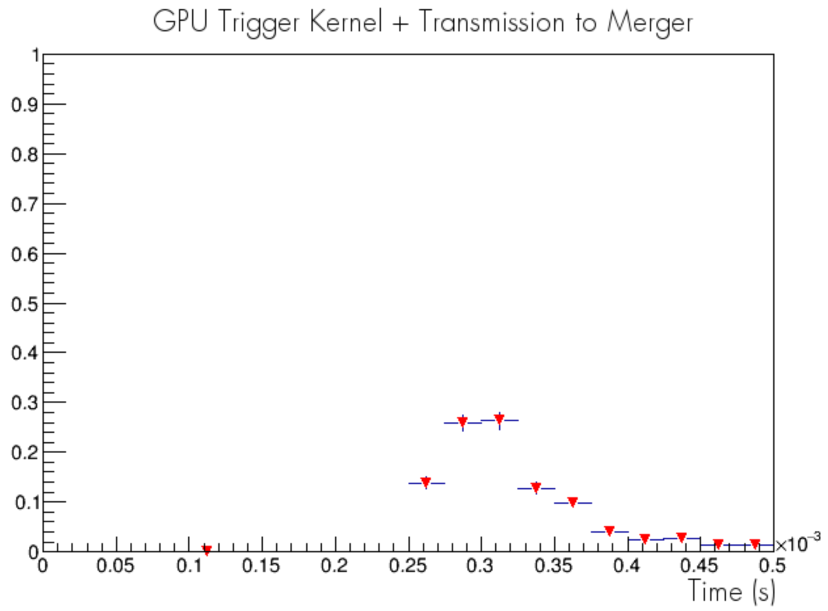


Results (ctd.)

- GPU Direct very promising
  - Data transmitted between nodes with lowest latency
- Track Reconstruction highly dependent on the combinatorics
- Ping times are included (t ~3μs)







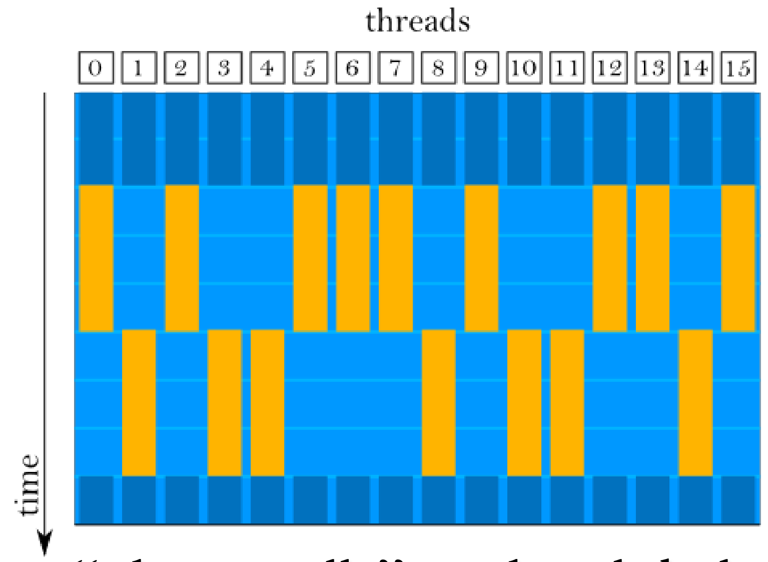
Conclusion

- PATATRACK framework developed as initial liaison between the HEP and HPC fields
  - All the present efforts focused at single process on single node opportunistic use of HPC infrastructures
- Communication, speed links and latencies already ready for the detectors of the near future
  - Gilder's law still valid
- High bandwidth memories and high capacities (tens of GBs) would allow higher time buffers at the lowest stages of the trigger
- Discussion with hardware producers to reduce latencies and better meet our needs

Questions?

Backup

Branch Divergence



```
if (x[tid]>42)
{
  foo(x[tid]);
}else
{
  oof(x[tid]);
}
```

- Avoid frequent "phone calls" to the global memory and read/write the data locally as much as possible before updating global values
- Make use of registers and shared memory for aggregating partial results
- Requires storage resources to keep copies of data structures

CUDA Queue

```
template< int maxSize, class T>
struct CUDAQueue
{
    __inline__ __device__
    int push(const T& element) {

        auto previousSize = atomicAdd(&m_size, 1);
        if(previousSize<maxSize)
        {
            m_data[previousSize] = element;
            return previousSize;
        } else {
            atomicSub(&m_size, 1);
            return -1;
    };
```