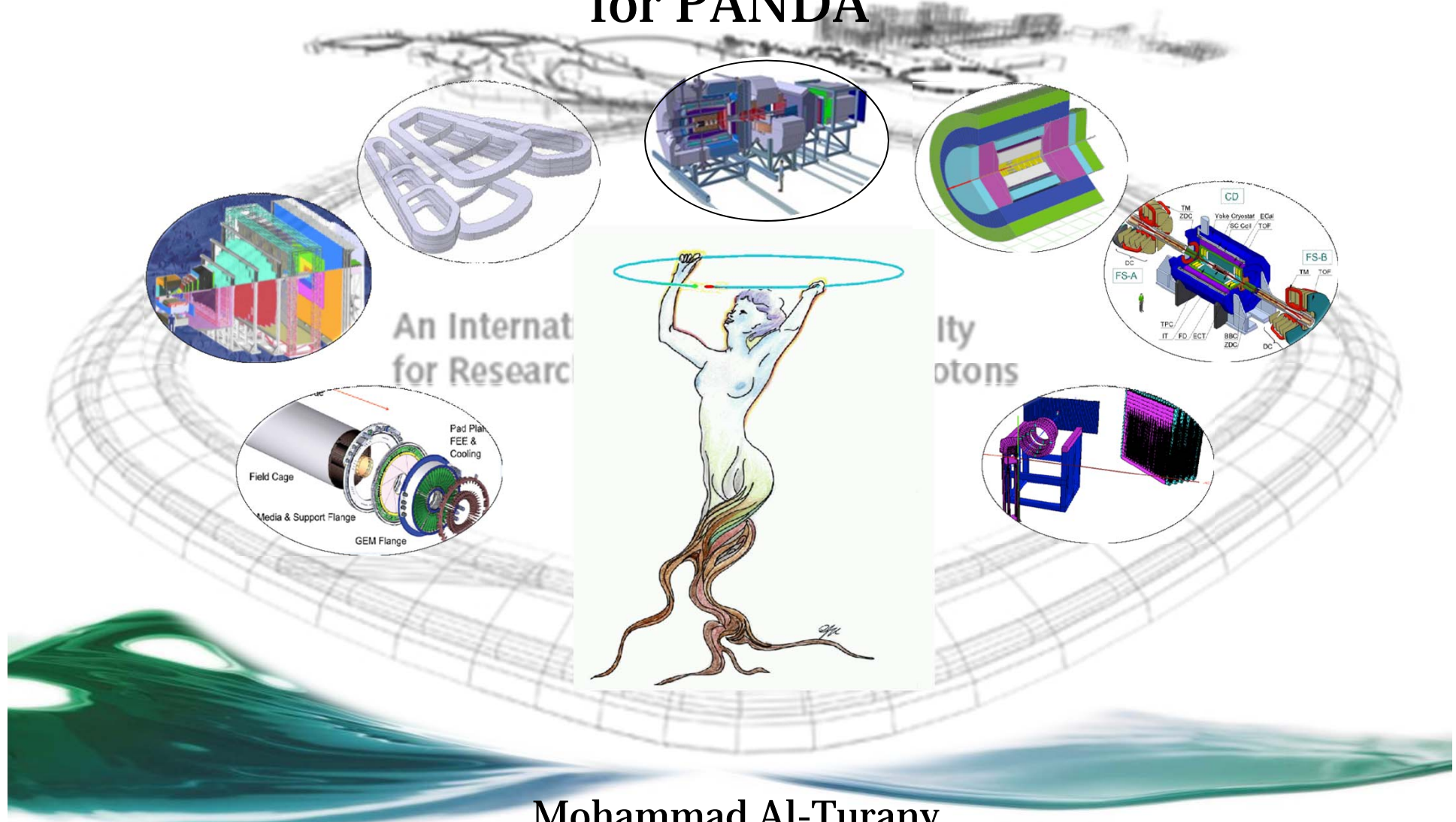


GPU based Online Tracking and Other Developments for PANDA

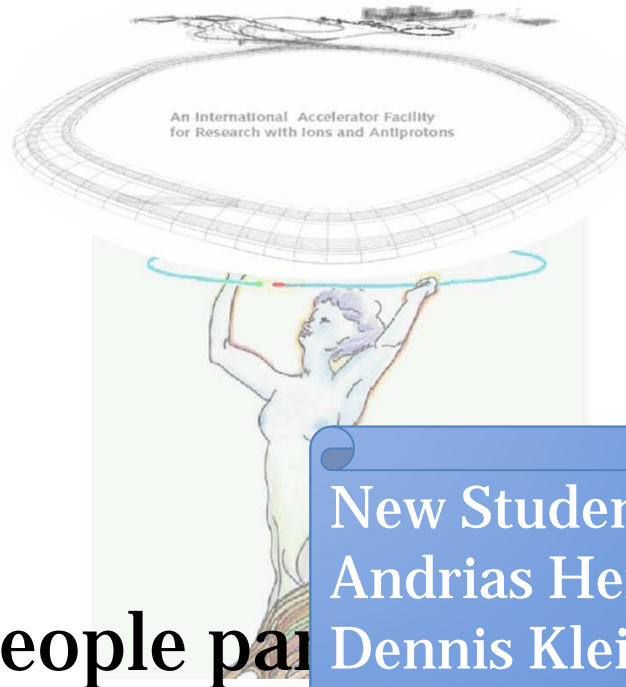


Mohammad Al-Turany
(GSI-Scientific Computing/Panda Collaboration)

FairRoot Developers:

Core Team:

Mohammad Al-Turany	SC/CBM/PANDA
Denis Bertini	SC/CBM/R3B
Florian Uhlig	SC/CBM
Radek Karabowicz	SC/PANDA
Dmytro Kresan	SC/R3B
Tobias Stockmanns	PANDA



New Students working on Online issues:

Andrias Herten	FZJ	(PANDA)
Dennis Klein	KOSI	(SC)

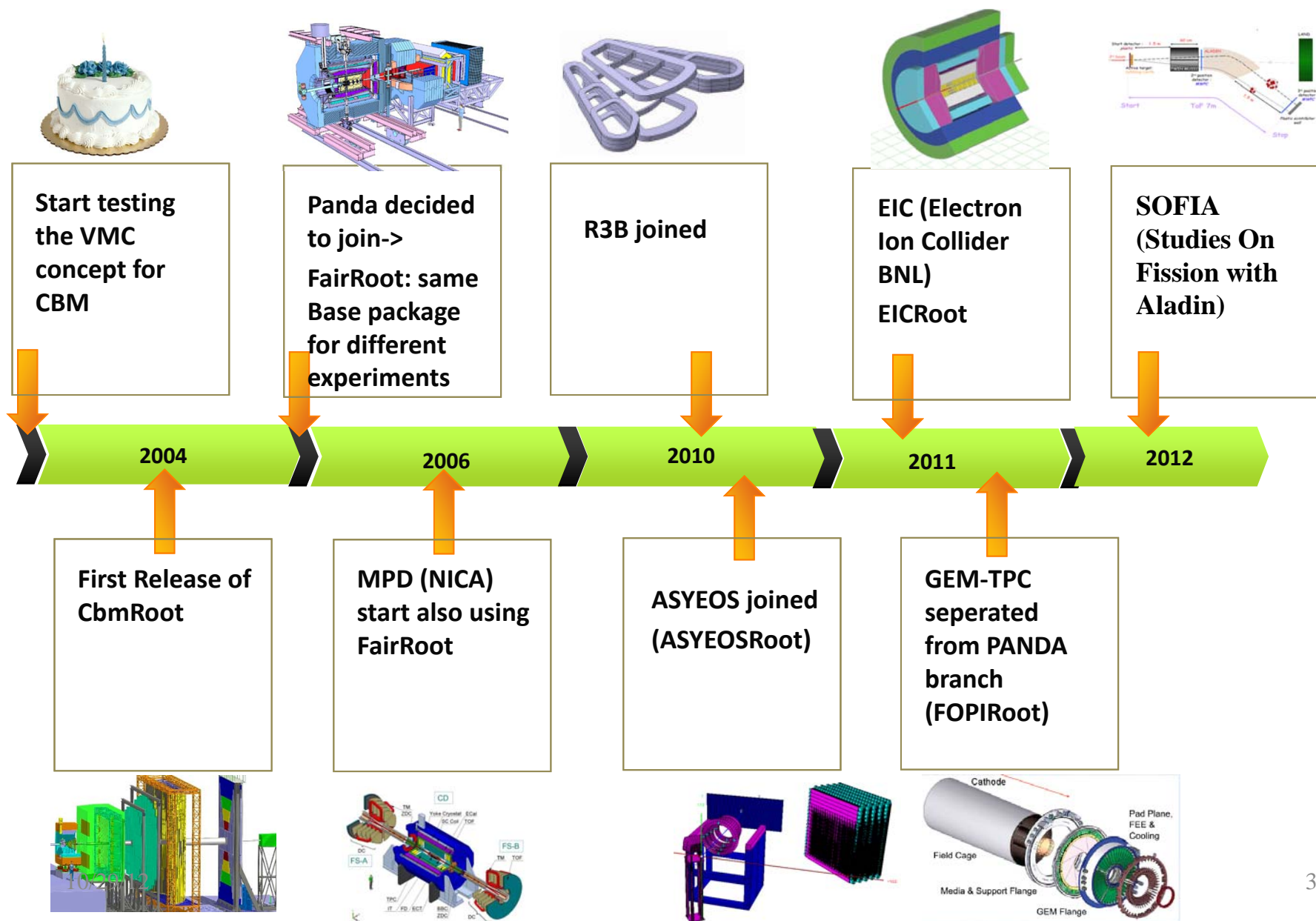
People part
to major f

Ilse König	HADES
Volker Friese	CBM
Olaf Hartman	PANDA

the project started end of

2003

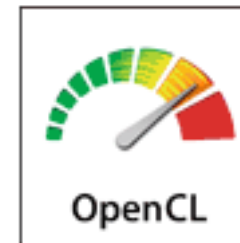
FairRoot : Timeline



Using GPUs for **Online** applications in FairRoot



CUDA





Which hardware? Which Software?

- What ever hardware we choose for now it will be **obsolete in 5 years**
- The question:
“How to parallelize the online reconstruction?”
should have the highest priority, after that we can implement the code **on any suitable hardware** that is available at that time
- In FairRoot we decided to use the currently most suitable hardware for the problem we try to solve (**Experiment online tracking**)



Why NVIDIA?

- Support a **true cache hierarchy** in combination with on-chip shared memory
- Support **ECC**, it detects and corrects errors before system is affected. It also Protects register files, shared memories, L1 and L2 cache, and DRAM
- Limited, but increasing support of **C++**
- **Concurrent Kernel Execution**
- Tesla product family is designed ground-up for parallel computing and offers exclusive computing features.



Why NVidia?

Architectural differences

ATI

- Adopts very long instruction word (**VLIW**) processors to carry out computations in a vector-like fashion (Performance of programs largely depends on there packing ratio)
- The L1 cache on the HD 5870 can only be used to cache image objects and constants.
- Only image objects and constants use the L2 in HD 5870

NVidia

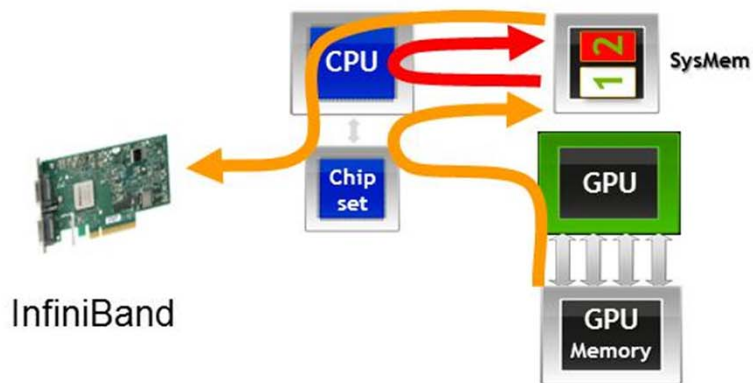
- Use multi-threading execution to execute code in a Single-Instruction-Multiple-Thread (**SIMT**) fashion and explores thread-level parallelism to achieve high performance
- **The L1 cache is configurable** to different sizes and can be disabled by setting a compiler flag.
- All global memory accesses go through the L2 in GTX 580

Why NVidia? GPUDirect:

Without GPUDirect

Same data copied three times:

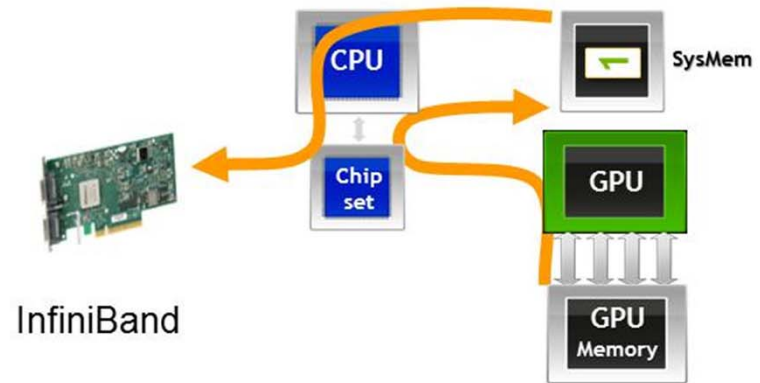
1. GPU writes to pinned system1
2. CPU copies from system1 to system2
3. InfiniBand driver copies from system2



With GPUDirect

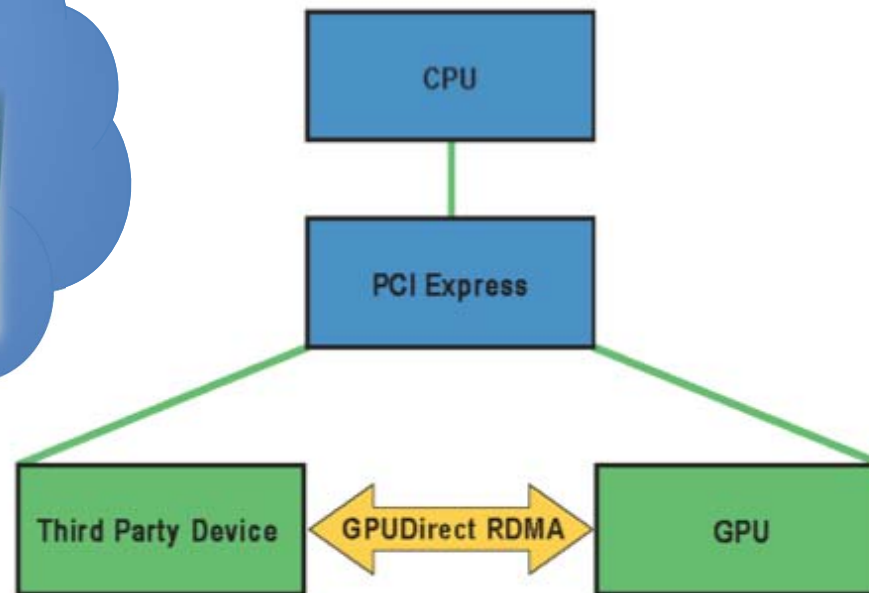
Data only copied twice

Sharing pinned system memory makes system-to-system copy unnecessary



Why NVidia:

Enables a direct path for communication between the GPU and a peer device using standard features of PCI Express



RDMA for GPUDirect within the Linux Device Driver Model

http://developer.download.nvidia.com/compute/cuda/5_0/rc/docs/GPUDirect_RDMA.pdf

Why NVidia?

Dynamic Parallelism :

Adds the capability for the GPU to generate new work for itself

- All ATI cards
- OpenCL with all cards
- Older NVidia



Dynamic Parallelism

GPU Adapts to Data, Dynamically Launches New Threads

The diagram compares the execution models of Fermi and Kepler GPUs. On the left, 'Fermi GPU' shows a CPU sending work to a single GPU, which then manually dispatches work to multiple GPU threads. On the right, 'Kepler GPU' shows a CPU sending work to a single GPU, which then dynamically launches a tree of new GPU threads, allowing the GPU to adaptively subdivide work based on the data.

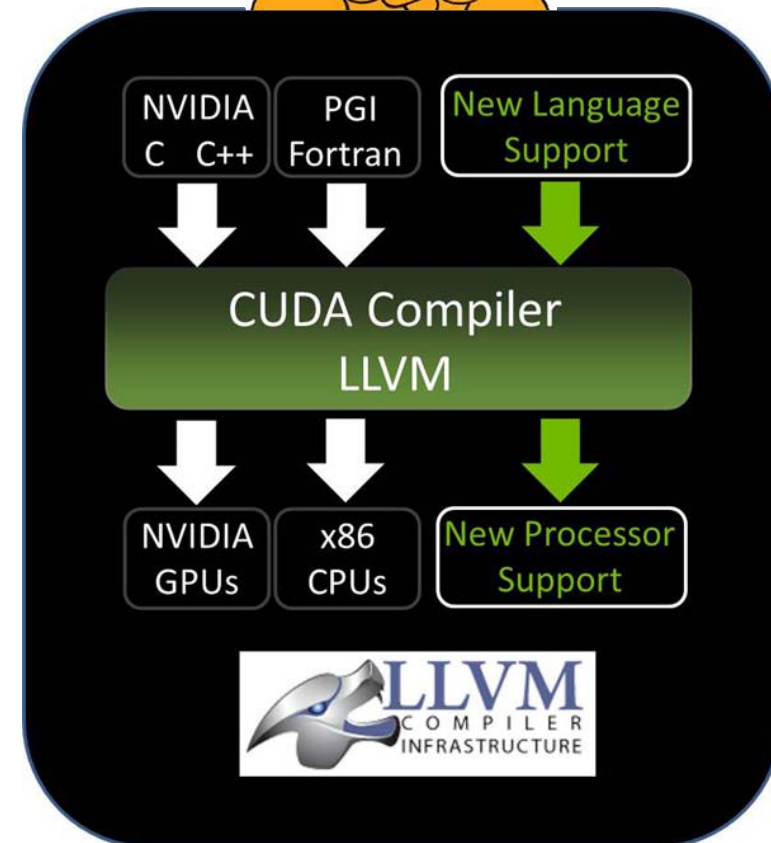
Dynamic Parallelism

Makes GPU Computing Easier & Broadens Reach

Three heatmaps illustrate different levels of dynamic parallelism. The first, 'Too coarse', shows a large grid with a few large rectangular blocks. The second, 'Too fine', shows a very dense grid with many small squares. The third, 'Just right', shows a grid where the block sizes are adaptively adjusted to match the underlying data distribution, with larger blocks in high-intensity regions and smaller blocks in low-intensity regions.

Why CUDA?

- **Open Source**
- CUDA is an architecture designed to let you do your work, rather than forcing your work to fit within a limited set of performance libraries.
- The software development environment is reasonable and straightforward
- Limited, but increasing support of C++





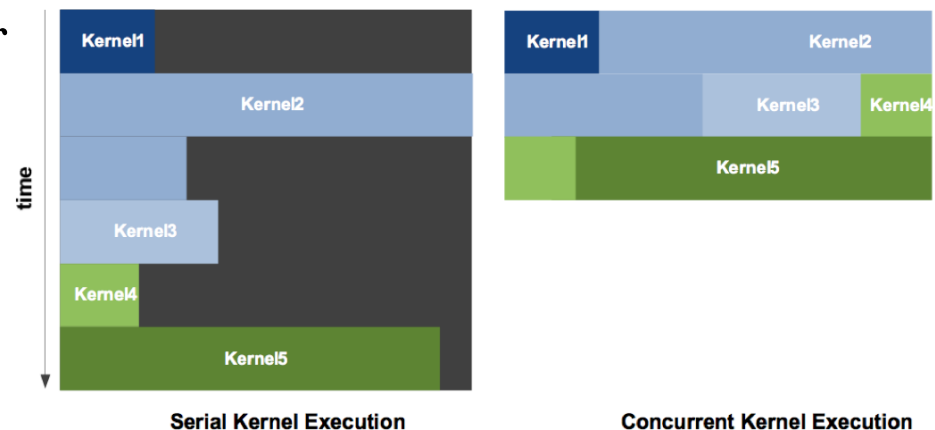
CUDA has a very useful open source, infrastructure libraries

- **Thrust:** Allows you to program parallel architectures using an interface similar to the C++ Standard Template Library (STL).
- **CUBLAS:** Basic Linear Algebra Subprograms
- **CURAND:** Simple and efficient generation of high-quality pseudorandom and quasirandom numbers
- **CUSPARSE:** A set of basic linear algebra subroutines used for handling sparse matrices
- **CUFFT:** Fast Fourier Transform (FFT) library.

What about OpenCL on NVIDIA?



- OpenCL is portable, but it is **not fully performance portable** (there's a bunch of papers that states exactly this, also across GPU vendors)
- **Concurrent Kernel Execution** is only available with CUDA and NVIDIA (see <http://devgurus.amd.com/thread/159535>)
 - Simultaneous execution of small kernels utilize whole GPU.
 - Overlapping kernel execution with device to host memory copy.
- In CUDA it is possible to reconfigure memory



Activities@GSI for PANDA:

Oleksiy Rybalchenko & Mohammad Al-Turany



- We port the code developed at Giessen for FPGA to GPU
- Work is ongoing on:
 - Comparing the performance, scalability, re-usability, ...etc, with other hardware and/or software techniques on the market
 - We plan to build a computing node prototype based on GPU that can make tracking on the fly.

Porting track finder/Fitter to CUDA

Original code is optimized for FPGA

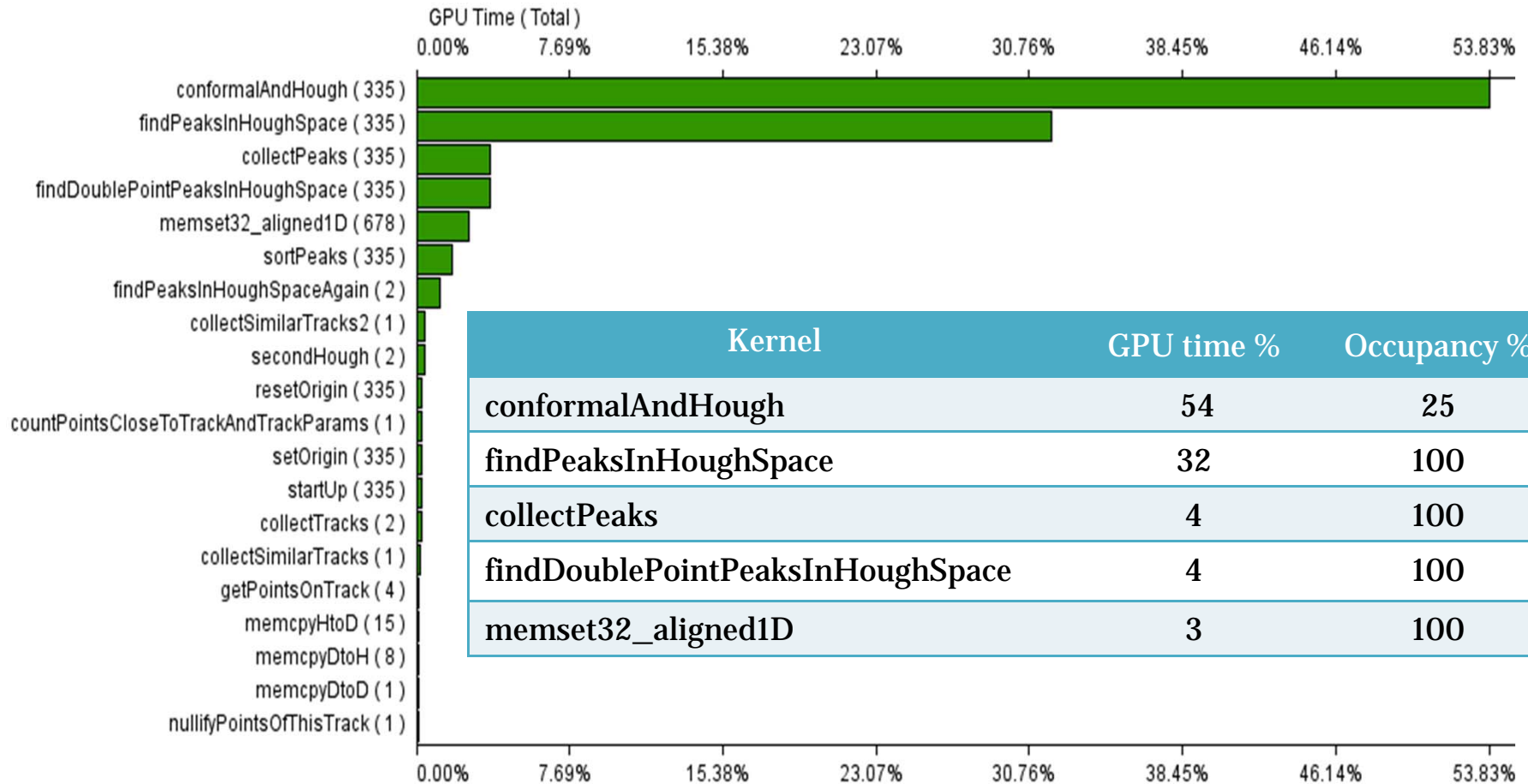
Lookup tables are used for the mathematical functions (Code is designed to work on FPGA)

Redesign the code into many functions (kernels)

Use the standard mathematical libraries delivered by NVIDIA

Profiler output for GPU time for each kernel

Gpu Time Summary Plot





The results are comparable:

CPU Results

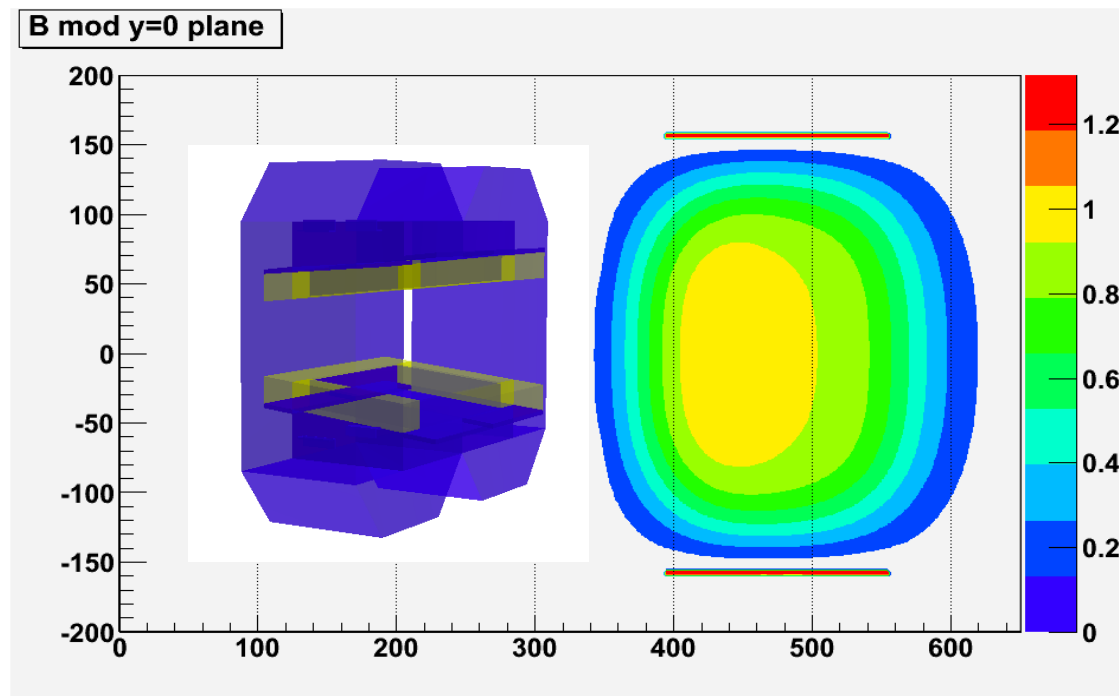
found tracks in first step:	1615
collect similar tracks:	done
number of tracks:	158
recalculate tracks:	done
number of tracks:	14

GPU Results

found tracks in first step:	1609
collect similar tracks:	done
number of tracks:	151
recalculate tracks:	done
number of tracks:	14

Using Texture memory for field maps:

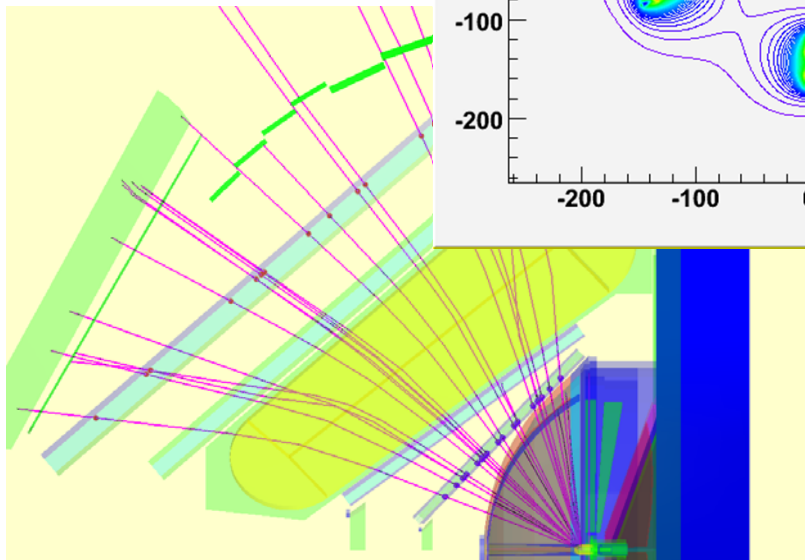
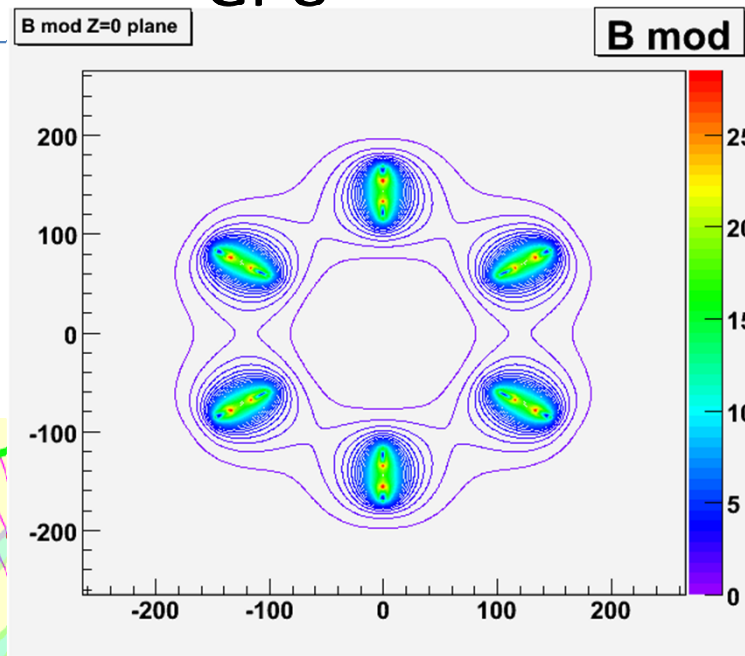
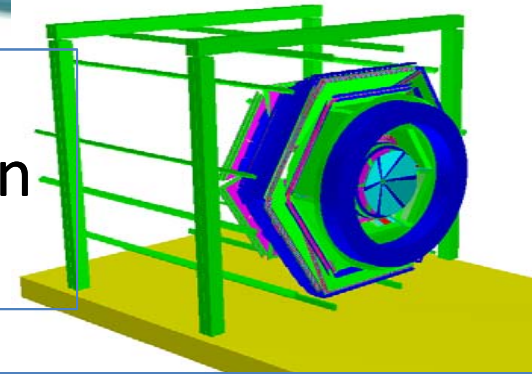
Track propagation (RK4) using PANDA dipole Field



Trk/ Event	NVS 290	8400 GT	8800 GT	Tesla
10	3	3	3.5	6
50	4.4	6	11	28
100	4.8	7.3	12.3	47
200	4.8	7.5	14.5	49
500	4.5	7.9	18.5	80
1000	5	8.1	21	111
2000	5	8	21	137
5000	5	8.4	21	175

Speedup : up to factor 175

HADES field map in Texture memory: Track propagation (RK4) in magnetic field on GPU



Propagation /Event	Tesla (CPU/GPU)
10	11
50	15
100	15
200	24
500	34
700	41

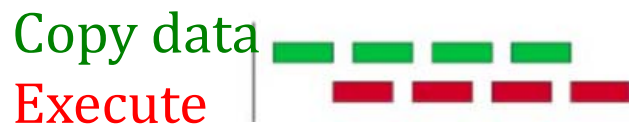
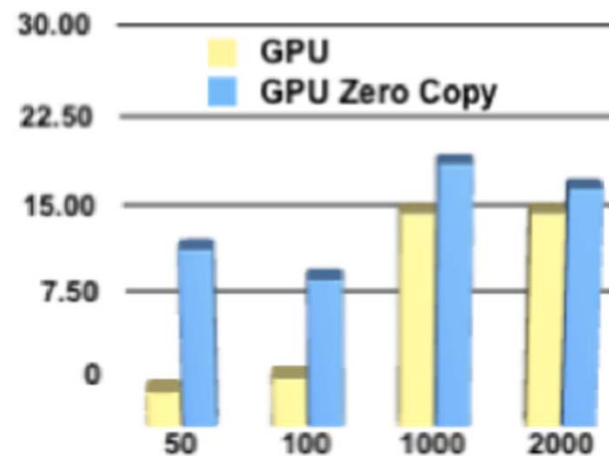
Track Propagation Speedup
by a factor 40

Some NVidia's specific features could make factors in performance depending on the problem we deal with!

Track + Vertex Fitting (PANDA): The same program code, same hardware, just using Pinned Memory instead of Mem-Copy!

CPU time/GPU time

Track/Event	50	100	1000	2000
GPU	3.0	4.2	18	18
GPU (Zero Copy)	15	13	22	20





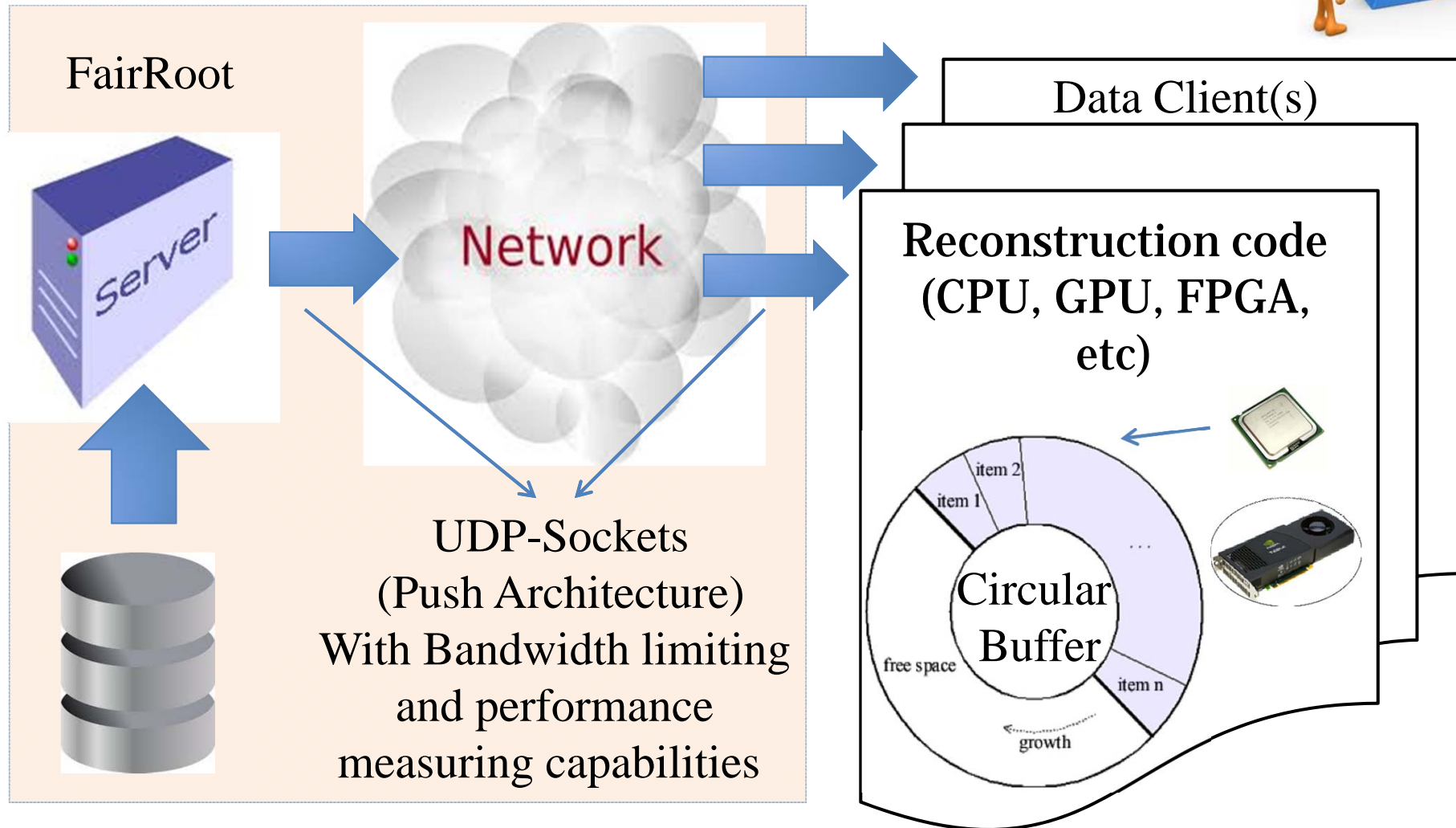
At this time:

- CUDA is fully integrated into the FairRoot build system
- CMake creates shared libraries for cuda par
- FairCuda is a class which wraps CUDA implemented functions so that they can be used directly from ROOT CINT or compiled code
- Similar to FairCuda we could do: **FairOpenCL**, but till today we did not see one single **working algorithm** in OpenCL for any experiment! As soon as this change we can easily support it.

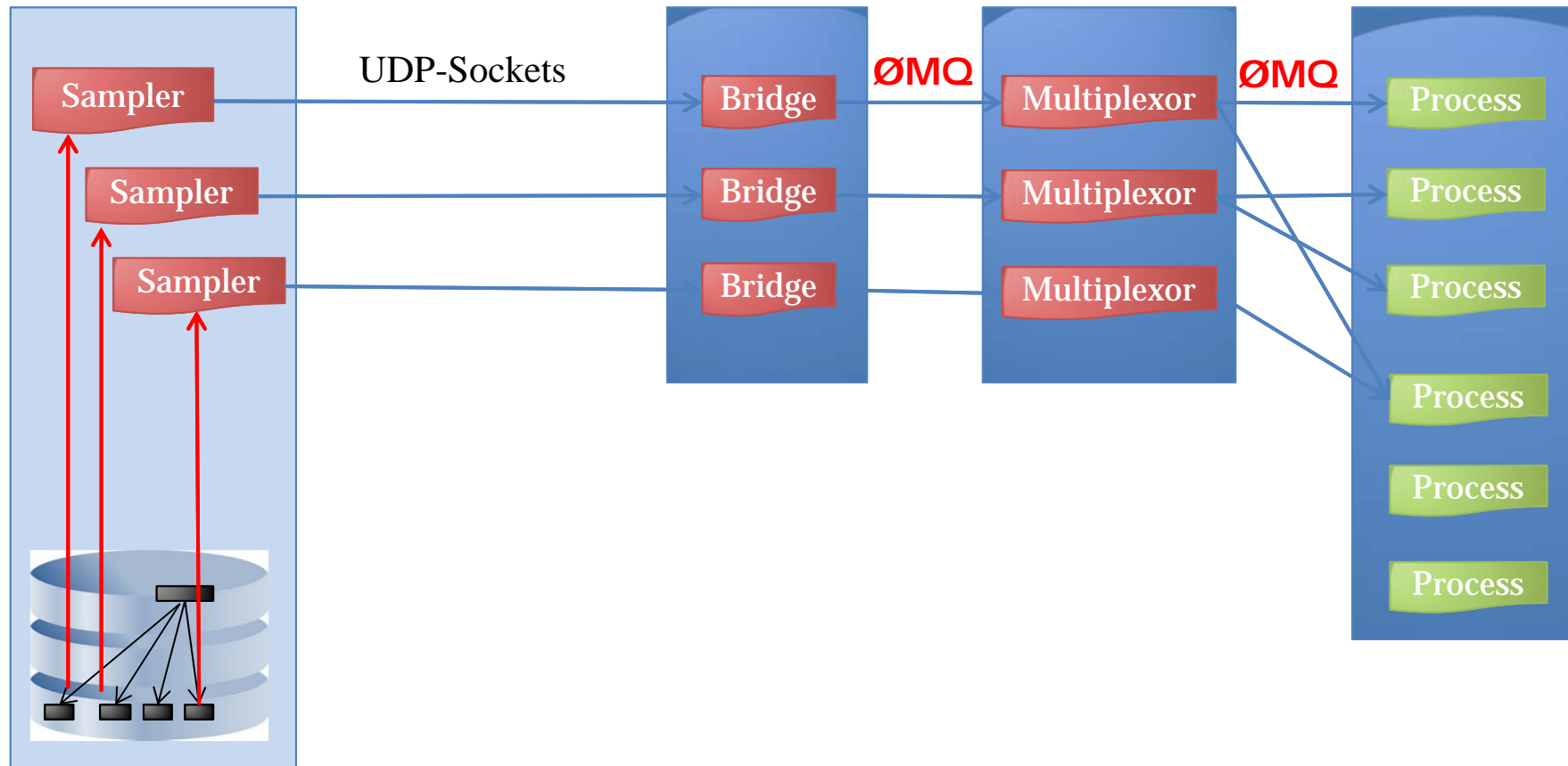
Event source simulation



Event source simulation: A tool for real time testing of online algorithms



Event source simulations (Dennis Klein & Mohammad Al-Turany)






Finally

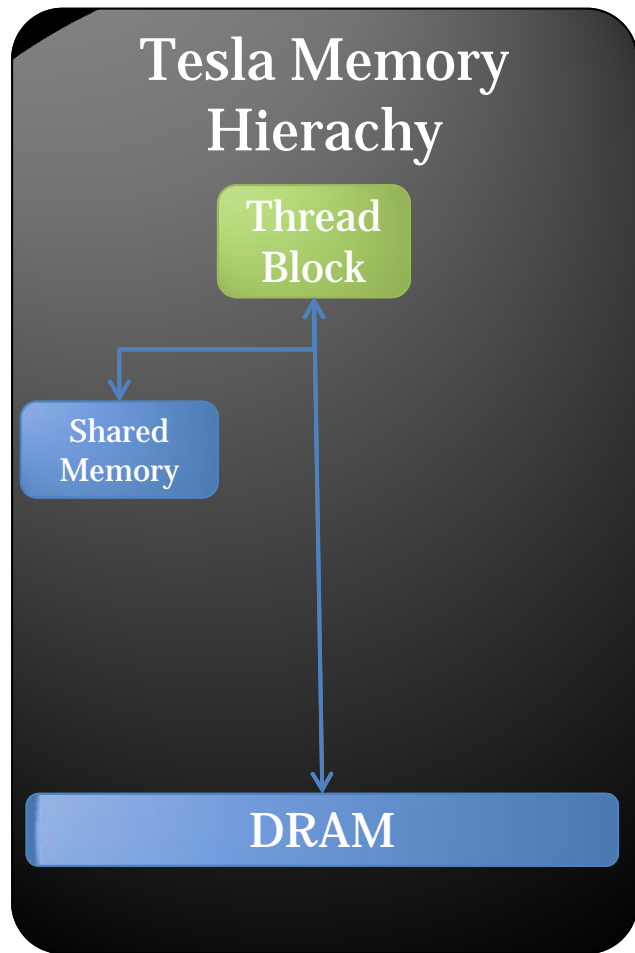
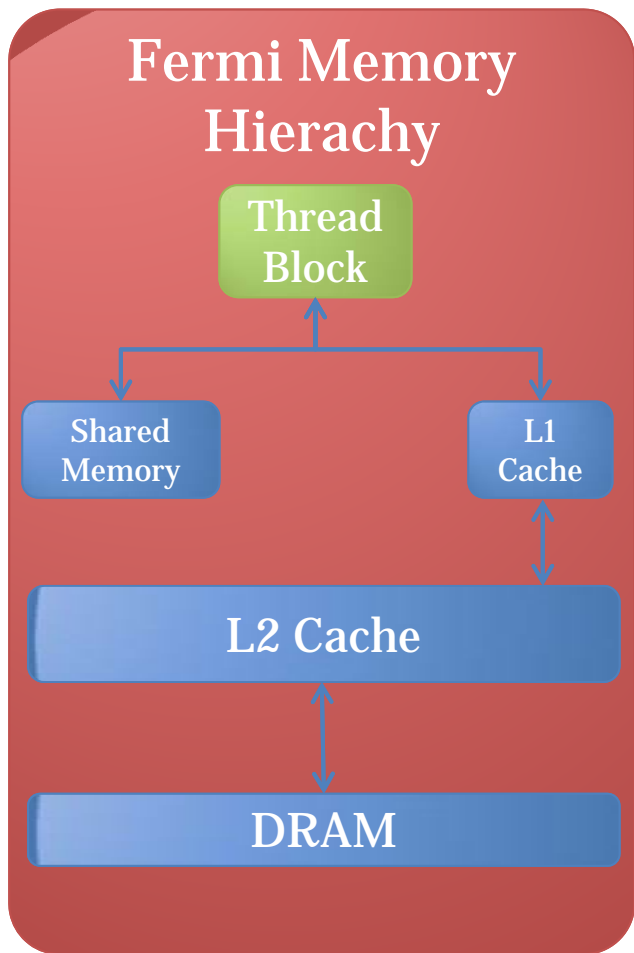
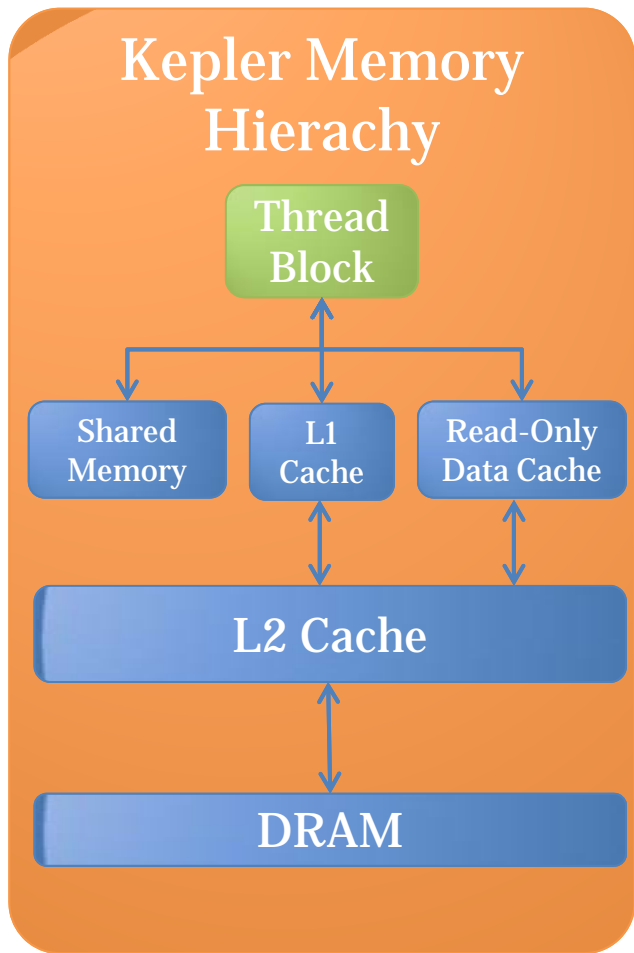
What ever we use now, the valuable part - **figuring out how to decompose our problem into massively parallel code for simple small cores with very little memory** - is going to be pretty easily portable between programming models.





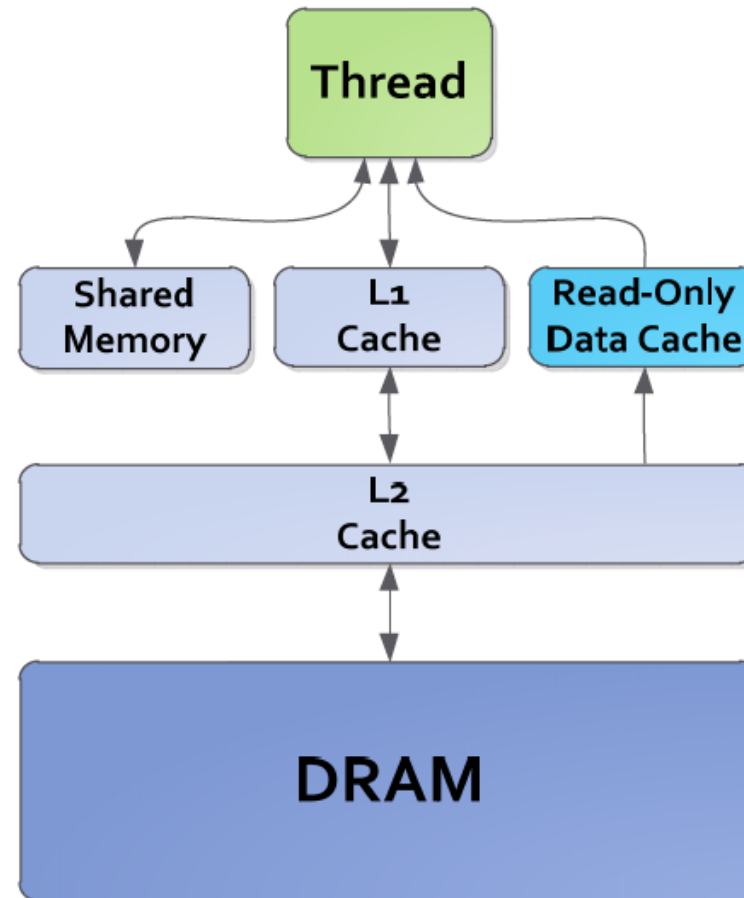
Backup and Discussion

- 
- A socket library that acts as a concurrency framework.
 - Faster than TCP, for clustered products and supercomputing.
 - Carries messages across inproc, IPC, TCP, and multicast.
 - **Connect N-to-N** via fanout, pubsub, pipeline, request-reply.
 - A synch I/O for scalable multicore message-passing apps.
 - 30+ languages including C, C++, Java, .NET, Python.
 - Most OSes including **Linux**, Windows, OS X, **PPC405/PPC440**.
 - Large and active open source community.
 - **LGPL free software** with full commercial support from iMatix.



Kepler Memory Hierarchy

- Kepler GK110 also enables compiler directed use of an additional new cache for read-only data
- In addition to the L1 cache, Kepler introduces a **48KB** cache for data that is known to be read-only for the duration of the function. ***In the Fermi generation, this cache was accessible only by the Texture unit.***





Comparing apples with apples (Commodity Cards)

ATI

card	Memory (GB) GDDR5	Price (Euro)
HD 5870	1.0	250
HD 6970	2.0	280
HD 7970	3.0	400

NVIDIA

card	Memory (GB) GDDR5	Price (Euro)
GTX 480	1.5	250
GTX 580	1.5	300
GTX 680	2.0	400

Comparing oranges with oranges (Professional Cards)

NVIDIA

Card Quadro	GB	Memory Bandwidth (GB/sec)	PCIe	Price (Euro)
4000	2.0	89.6	2.0x16	550
5000	2.5	120	2.0x16	1500
6000	6.0	144	2.0x16	3500

ATI

Card FirePro	GB	Memory Bandwidth (GB/sec)	PCIe	Price (Euro)
V7800	2.0	89.6	2.0x16	550
V8800	2.0	89.6	2.0x16	1040
V9800	4.0	89.6	2.1x16	2700

NVIDIA Tesla

Card	GB	Memory Bandwidth (GB/sec)	PCIe	Price (Euro)
C2070	6.0	144	2.0x16	2000
M2050	3.0	144	2.0x16	2200
M2070	6.0	144	2.0x16	2500
M2090	6.0	144	2.1x16	2800

RDMA: Eliminate CPU bandwidth and latency bottlenecks using direct memory access (DMA) between GPUs and other PCIe devices, resulting in significantly improved MPI SendRecv efficiency between GPUs and other nodes

