VIP Briefing on GPU Accelerator Technology

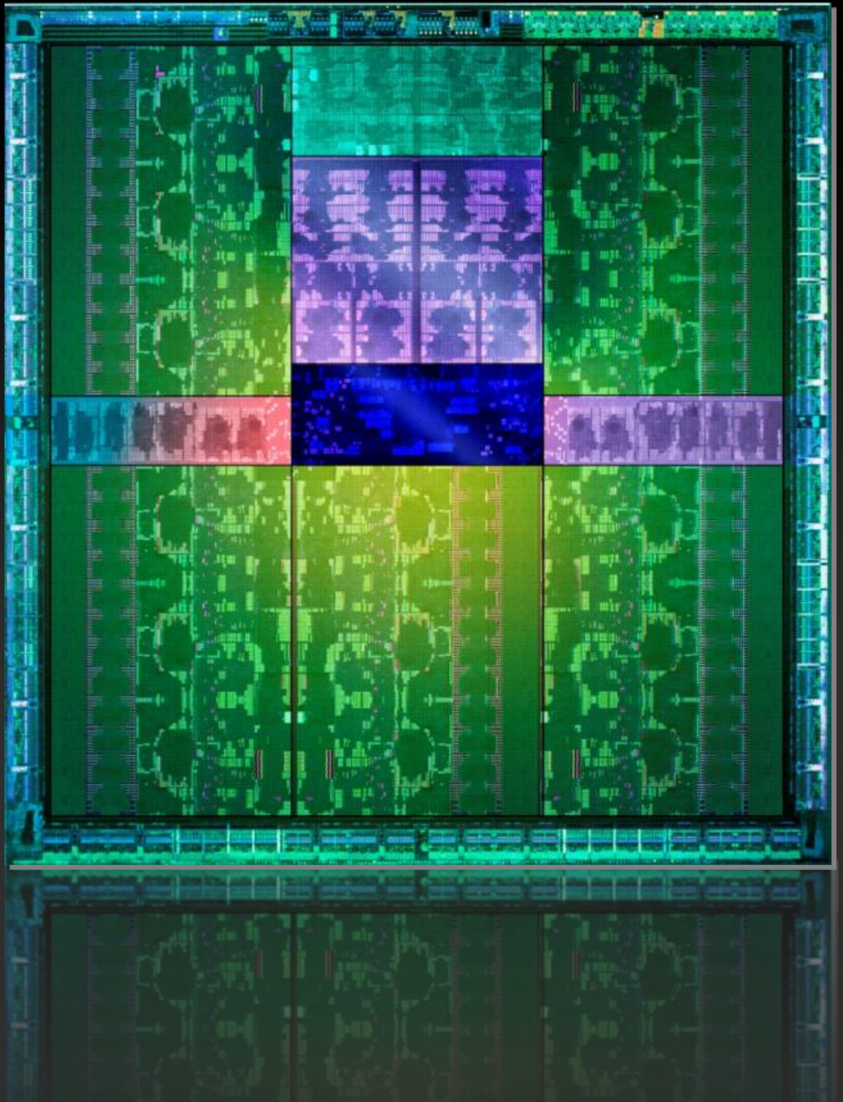
Steve Scott, CTO of Tesla
Ian Buck, GM of GPU Computing
Dr. Dirk Pleiter, Juelich

Long Term Goals for Tesla

Power
Efficiency

Ease of
Programming
And Portability

Application
Space
Coverage



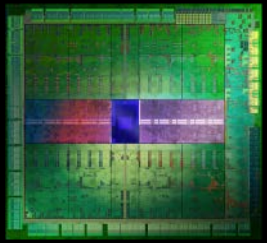
KEPLER

THE WORLD'S FASTEST, MOST EFFICIENT HPC ACCELERATOR

SMX	*(power efficiency)*
Hyper-Q Dynamic Parallelism	*(programmability and application coverage)*

Tesla K10

Dual GK104 GPUs



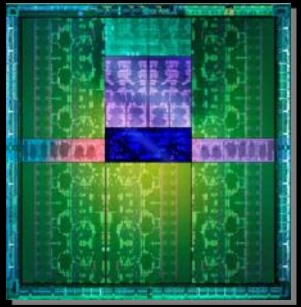
3x Single Precision

Video, Signal, Life Sciences, Seismic

Available Now

Tesla K20

GK110 GPU



3x Double Precision

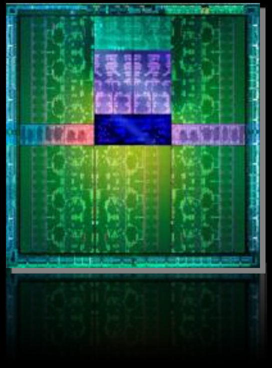
Hyper-Q & Dynamic Parallelism

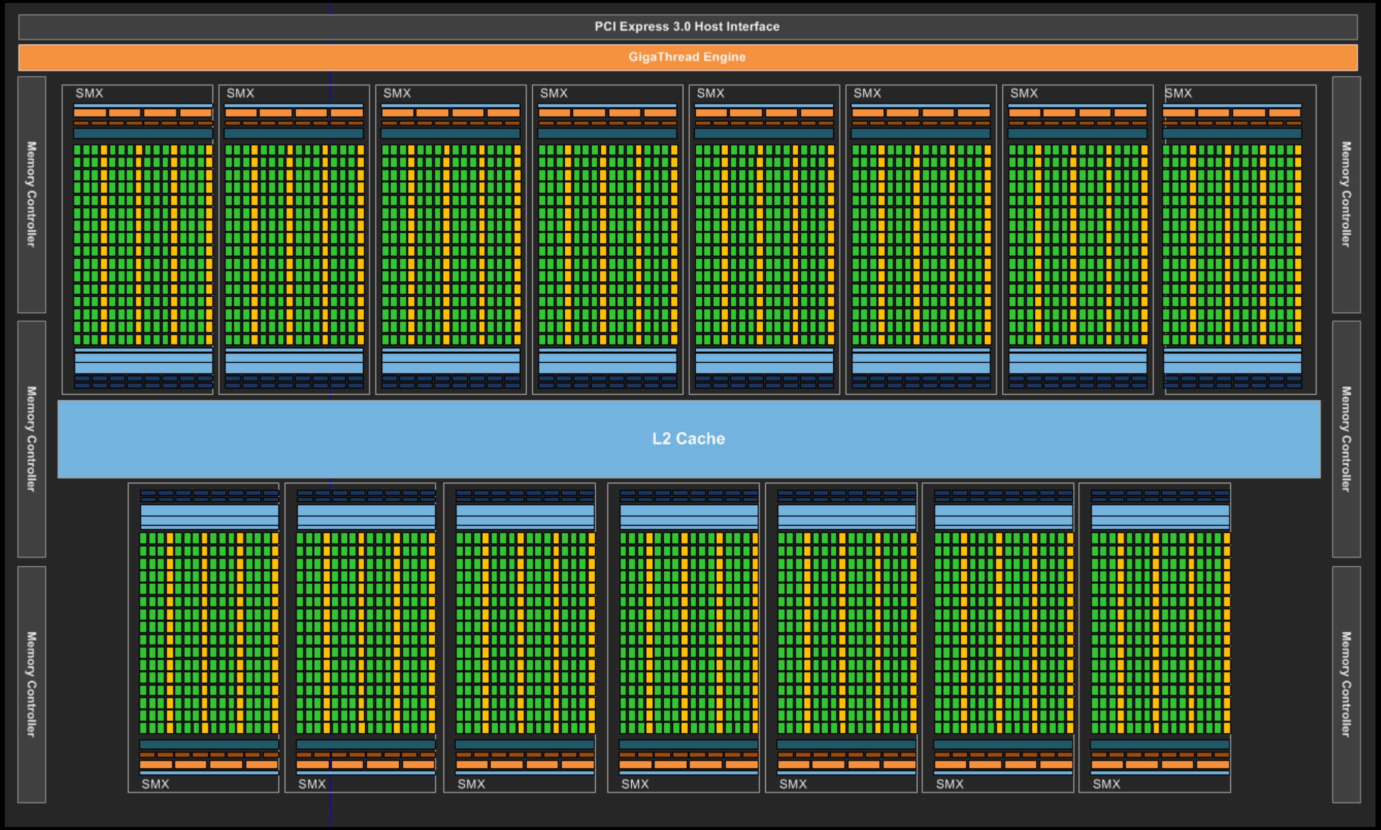
CFD, FEA, Finance, Physics, etc.

Available Q4 2012

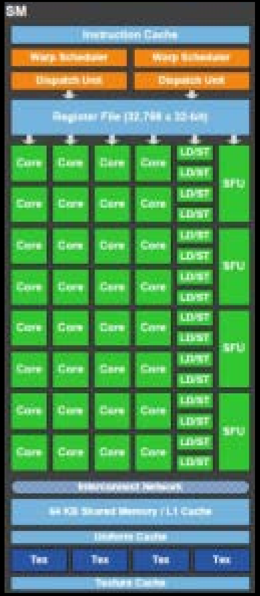
Kepler GK110 Block Diagram

- 7.1B Transistors
- 15 SMX units
- > 1 TFLOP FP64
- 1.5 MB L2 Cache
- 384-bit GDDR5
  - ~250 GB/s
- PCI Express Gen3





Kepler GK110 SMX vs Fermi SM



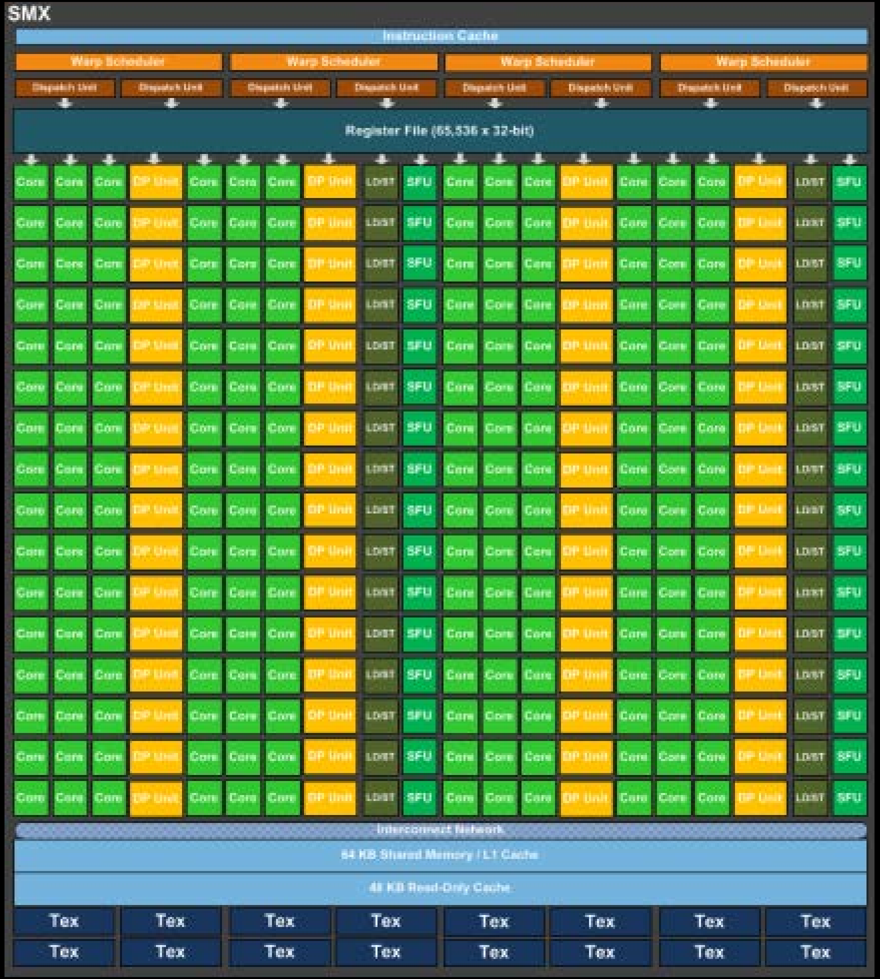
3x sustained perf/W

Ground up redesign for perf/W

6x the SP FP units

4x the DP FP units

Significantly slower FU clocks



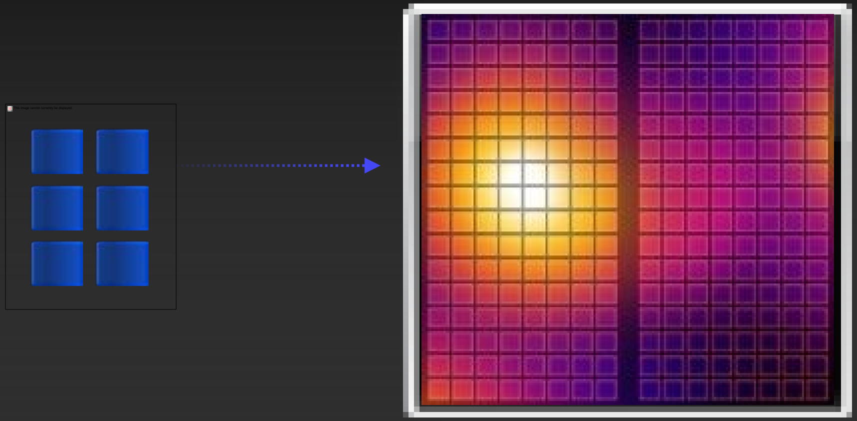
Selected Kepler ISA Enhancements

- Larger number of registers per thread
  - 63 in Fermi → 255 in Kepler
  - Common performance limited in Fermi due to register spilling
  - Significant performance improvement for some codes (e.g.: 5.3x on Quda QCD!)
- Atomic operations
  - Added int64 to match int32
  - Added functional units → 2-10x performance gains
- SHFL instruction for data exchange amongst threads of a warp
  - Broadcast, shifts, butterflies
  - Useful for sorts, reductions, etc.
- Loads through texture memory
  - Higher bandwidth and flexibility for read-only data (const__restrict)

Hyper-Q

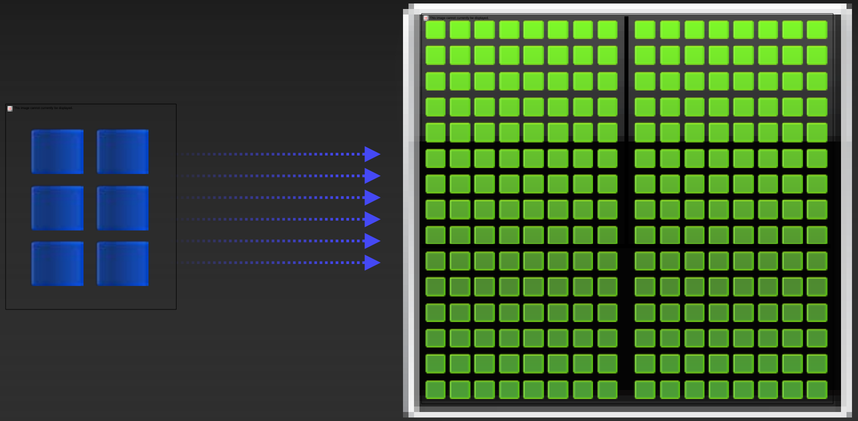
FERMI

1 Work Queue

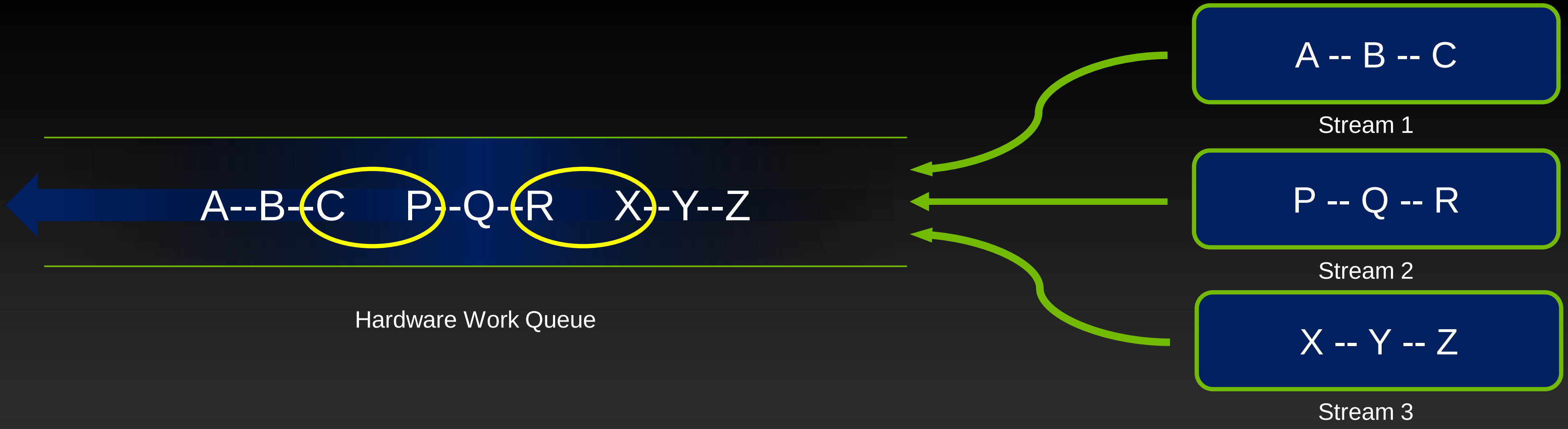


KEPLER

32 Concurrent Work Queues



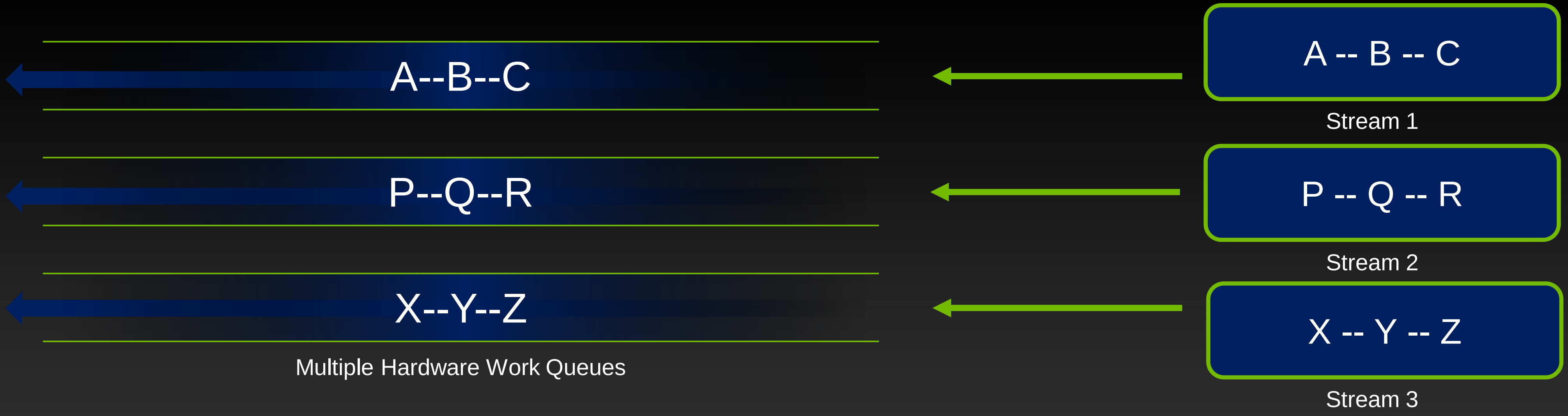
Fermi Concurrency



Fermi allows 16-way concurrency

- Up to 16 grids can run at once
- But CUDA streams multiplex into a single queue
- Overlap only at stream edges

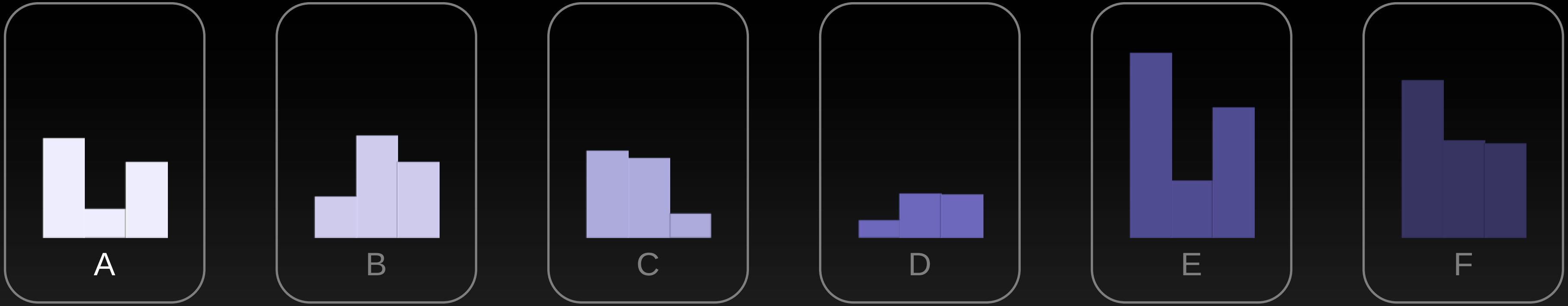
Kepler Improved Concurrency



Kepler allows 32-way concurrency

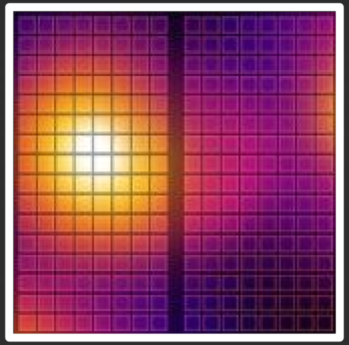
- One work queue per stream
- Concurrency at full-stream level
- No inter-stream dependencies

Fermi: Time-Division Multiprocess

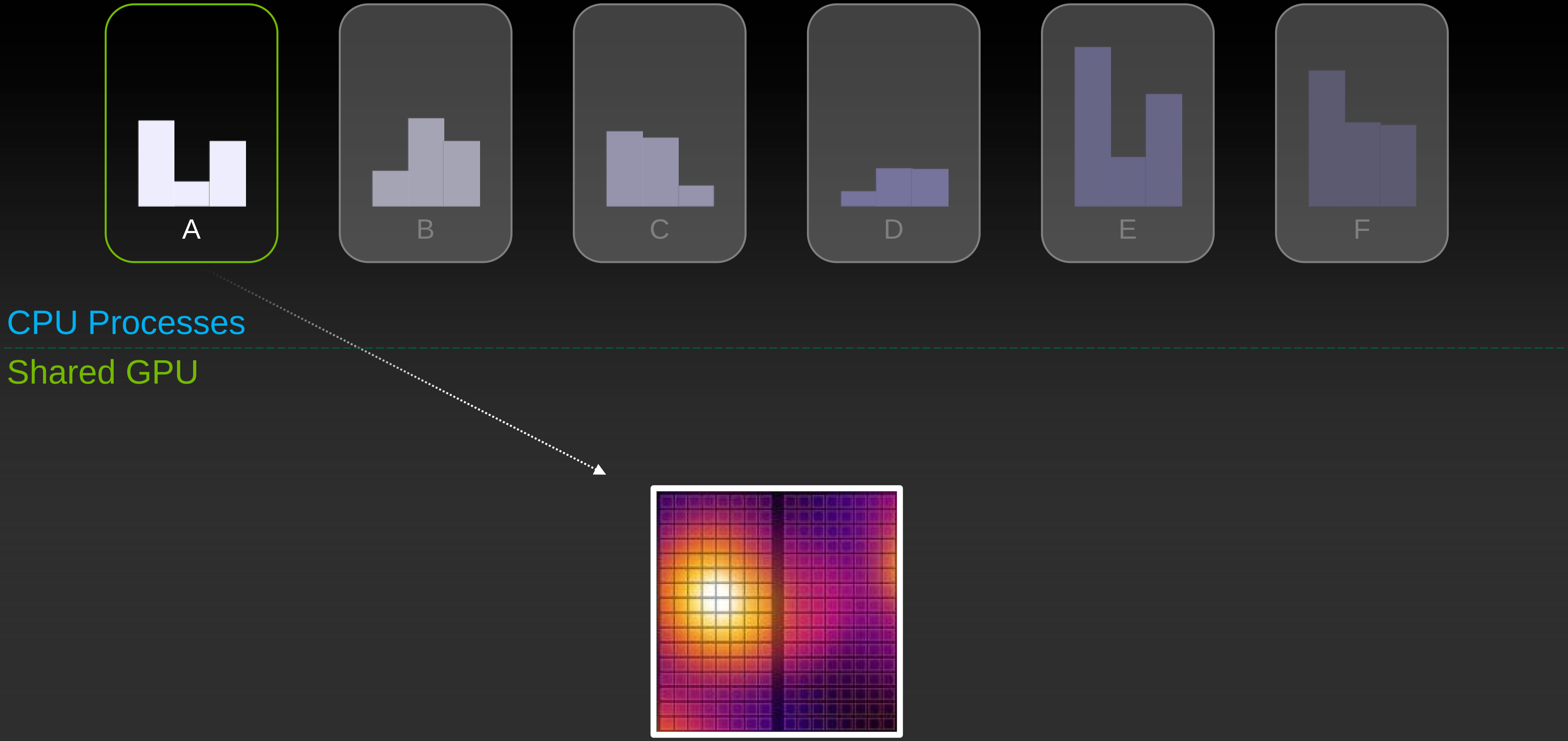


CPU Processes

Shared GPU



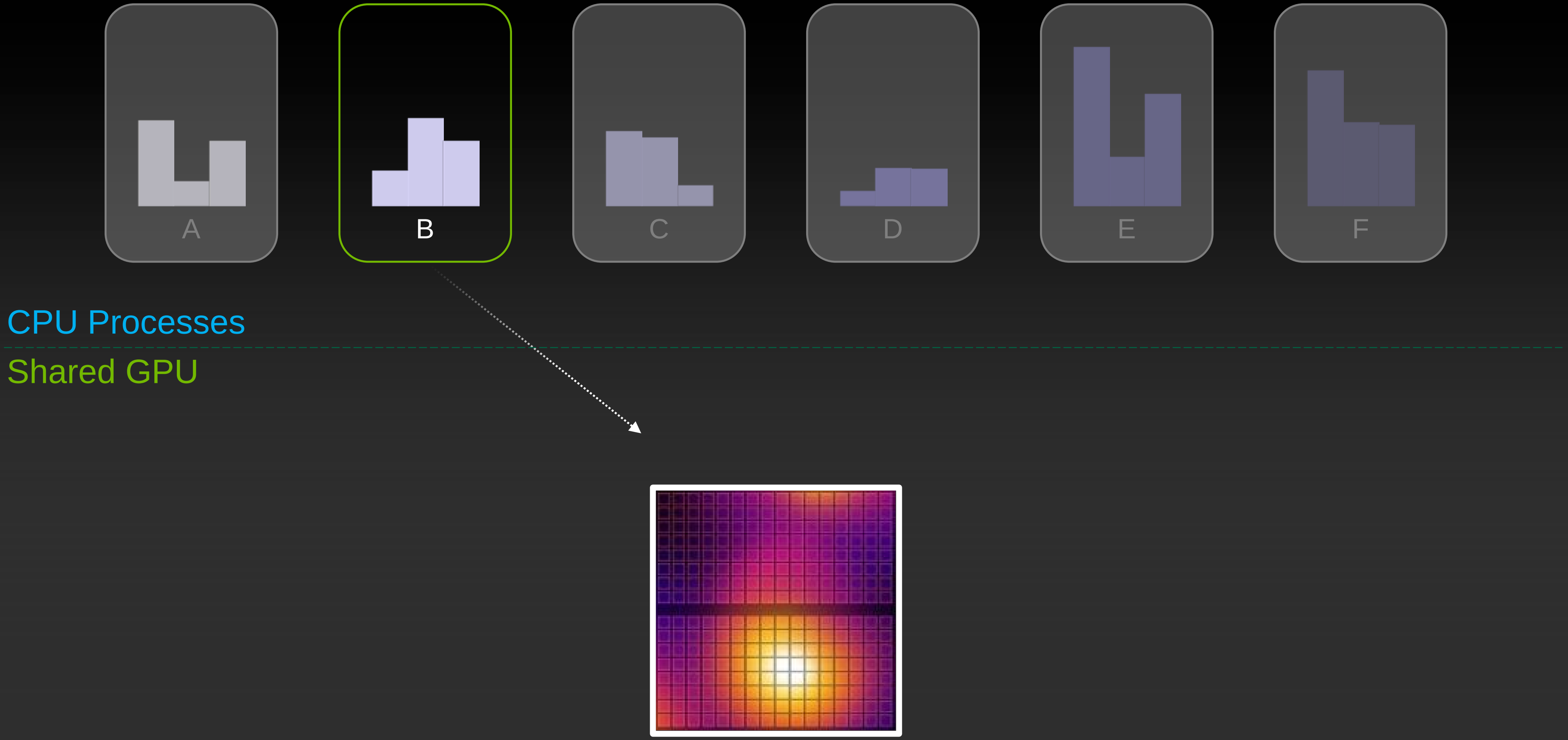
Fermi: Time-Division Multiprocess



CPU Processes

Shared GPU

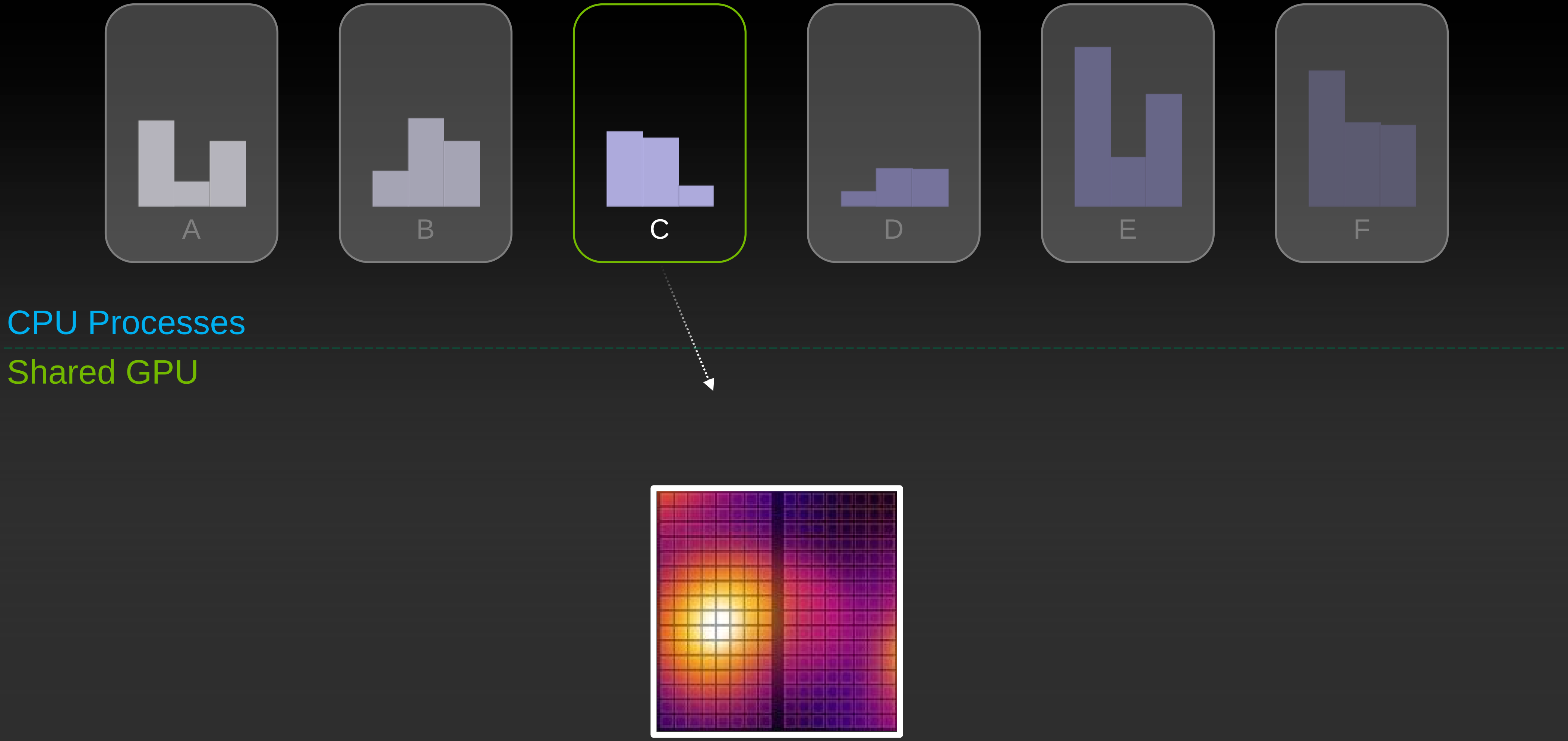
Fermi: Time-Division Multiprocess



CPU Processes

Shared GPU

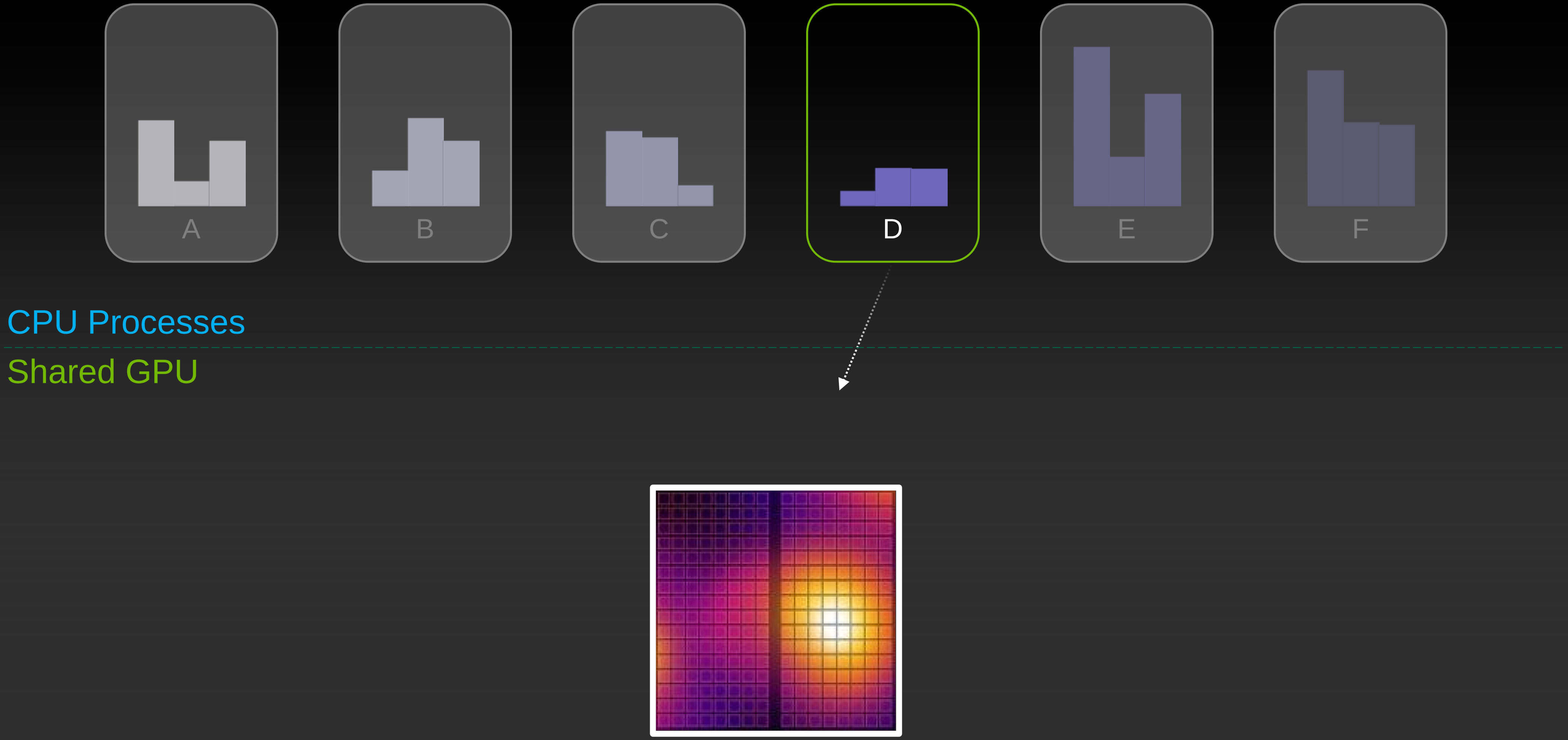
Fermi: Time-Division Multiprocess



CPU Processes

Shared GPU

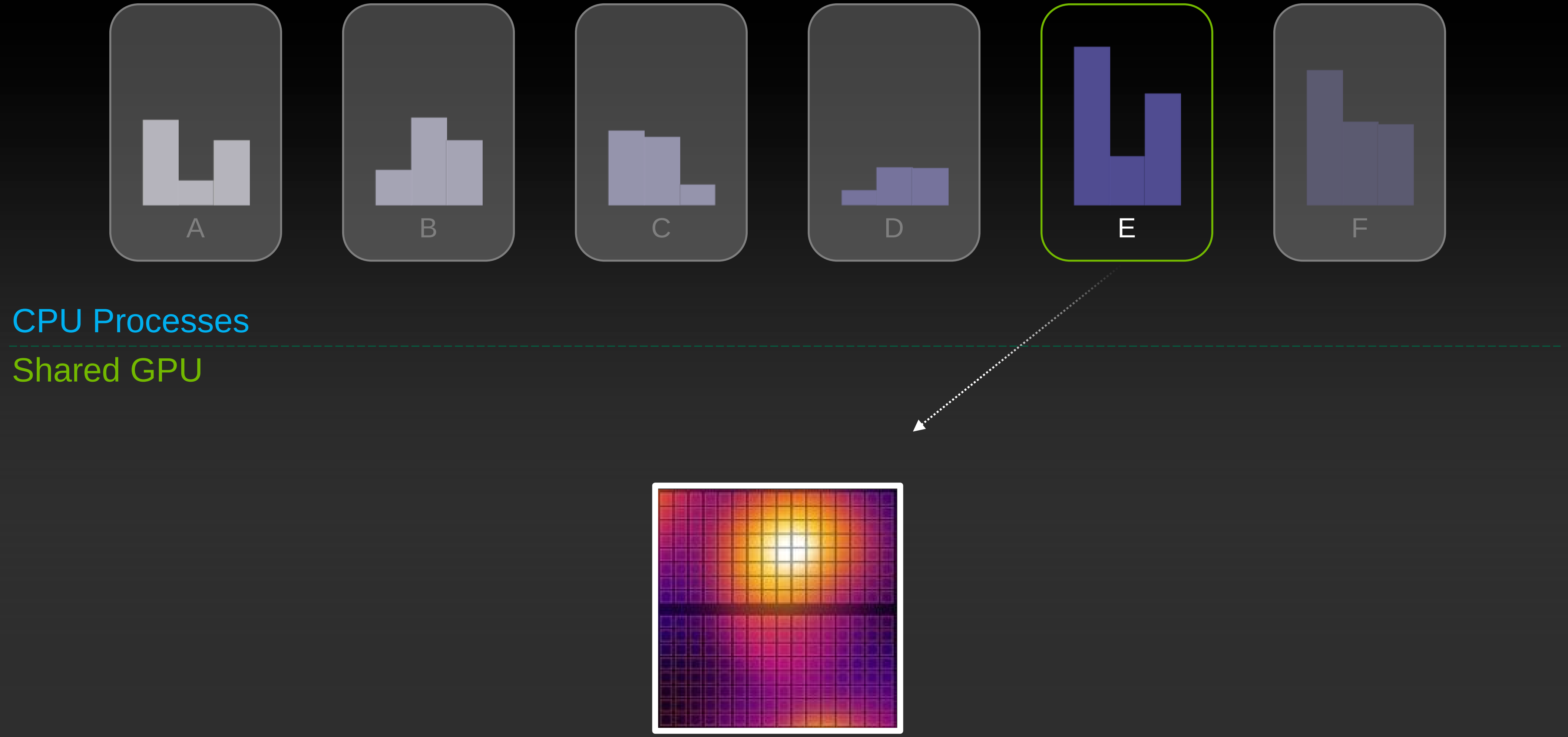
Fermi: Time-Division Multiprocess



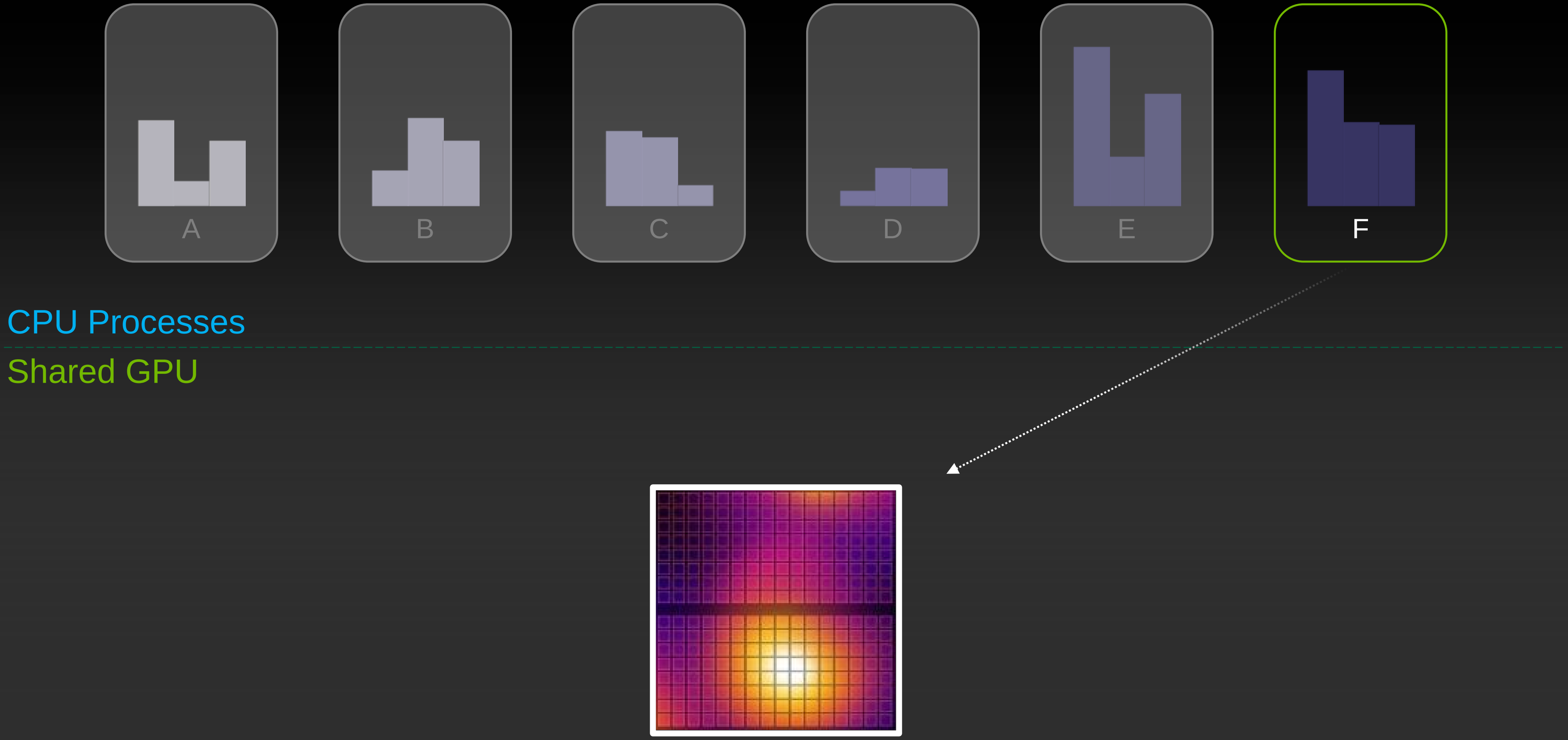
CPU Processes

Shared GPU

Fermi: Time-Division Multiprocess



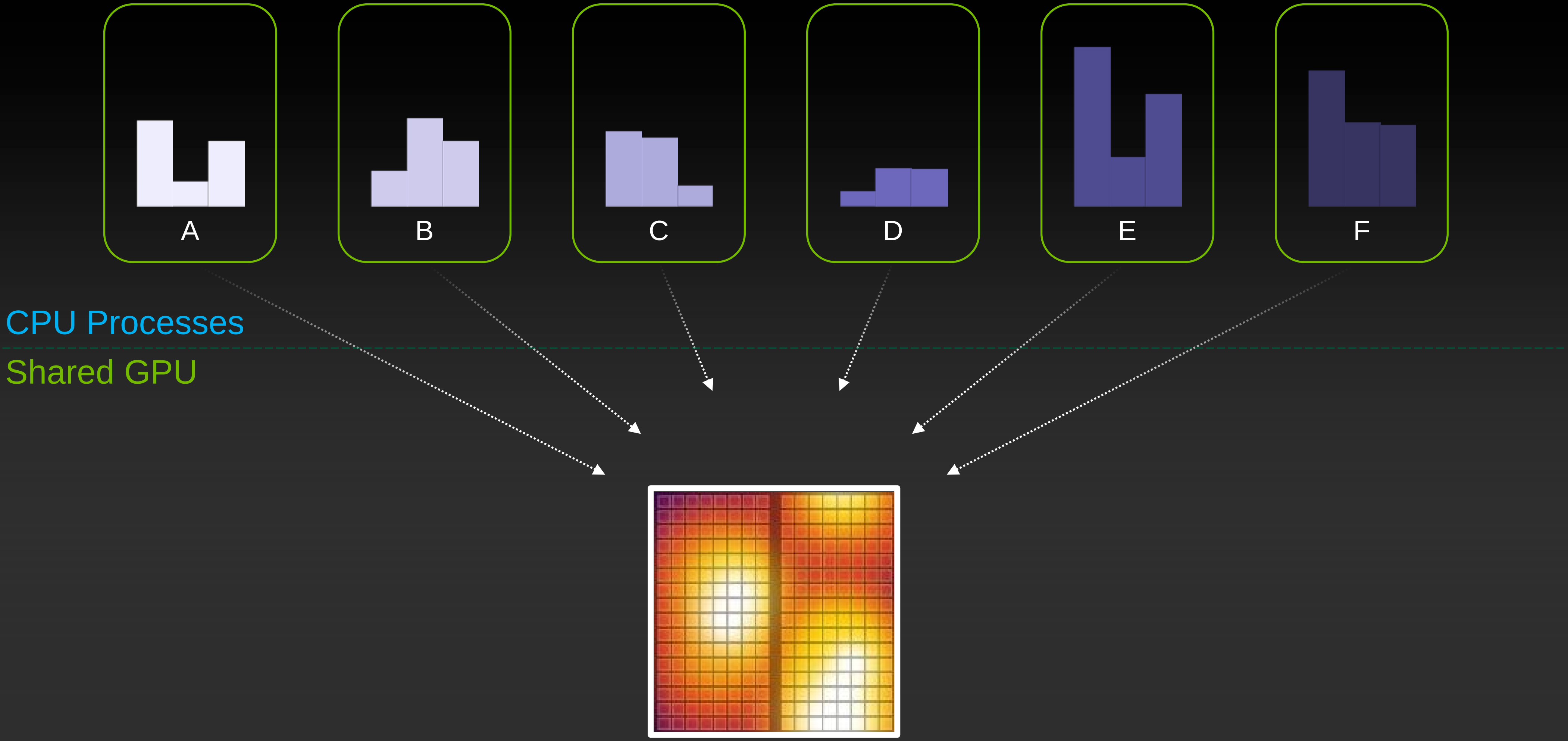
Fermi: Time-Division Multiprocess



CPU Processes

Shared GPU

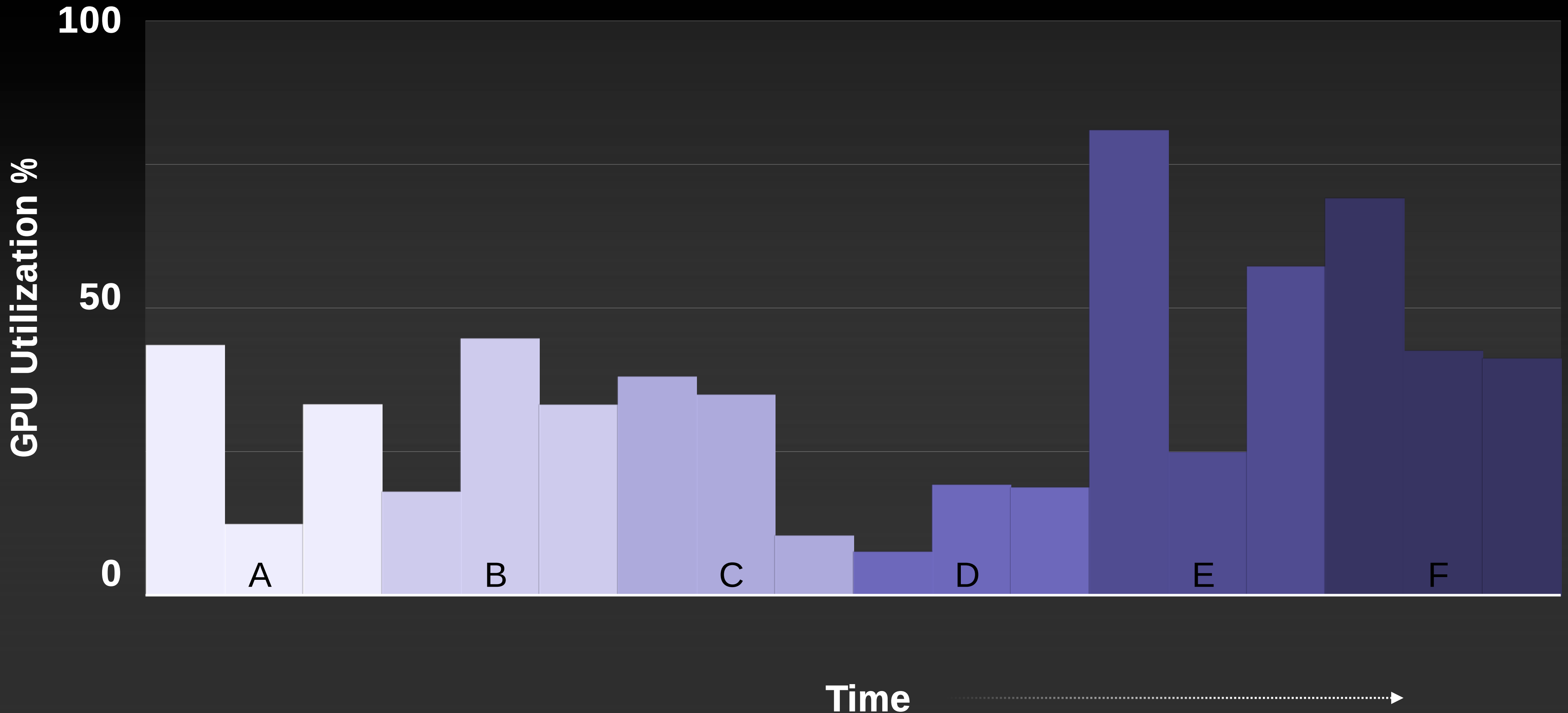
Kepler Hyper-Q: Simultaneous Multiprocess



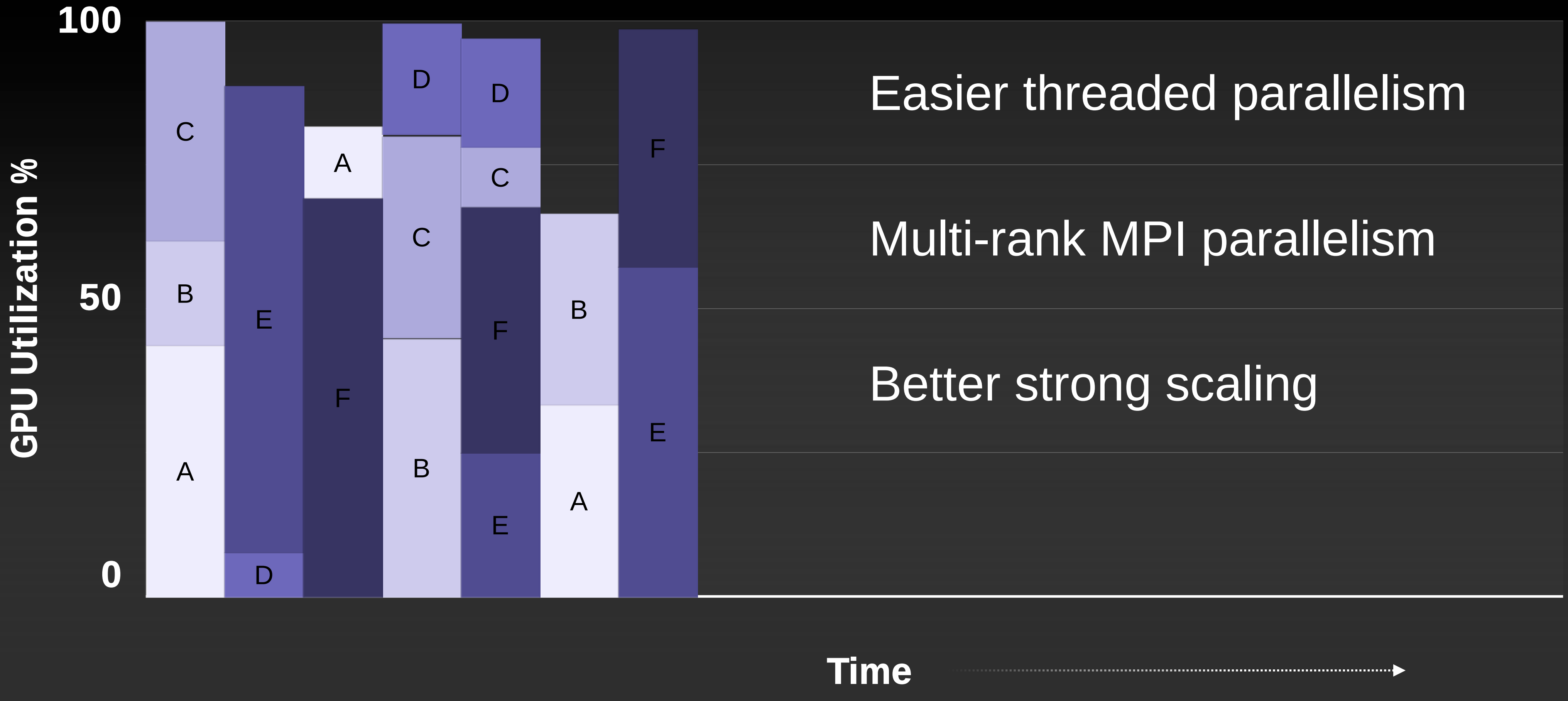
CPU Processes

Shared GPU

Without Hyper-Q



With Hyper-Q



Easier threaded parallelism

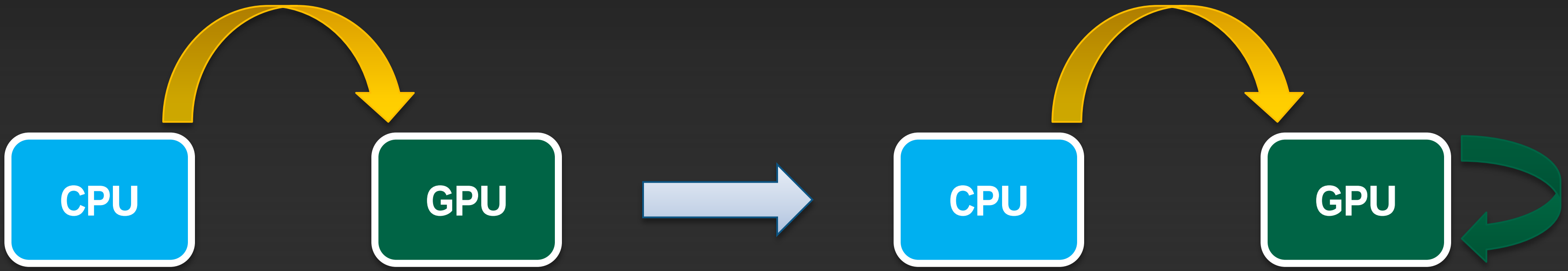
Multi-rank MPI parallelism

Better strong scaling

Dynamic Parallelism

The ability for any GPU thread to launch a parallel GPU kernel

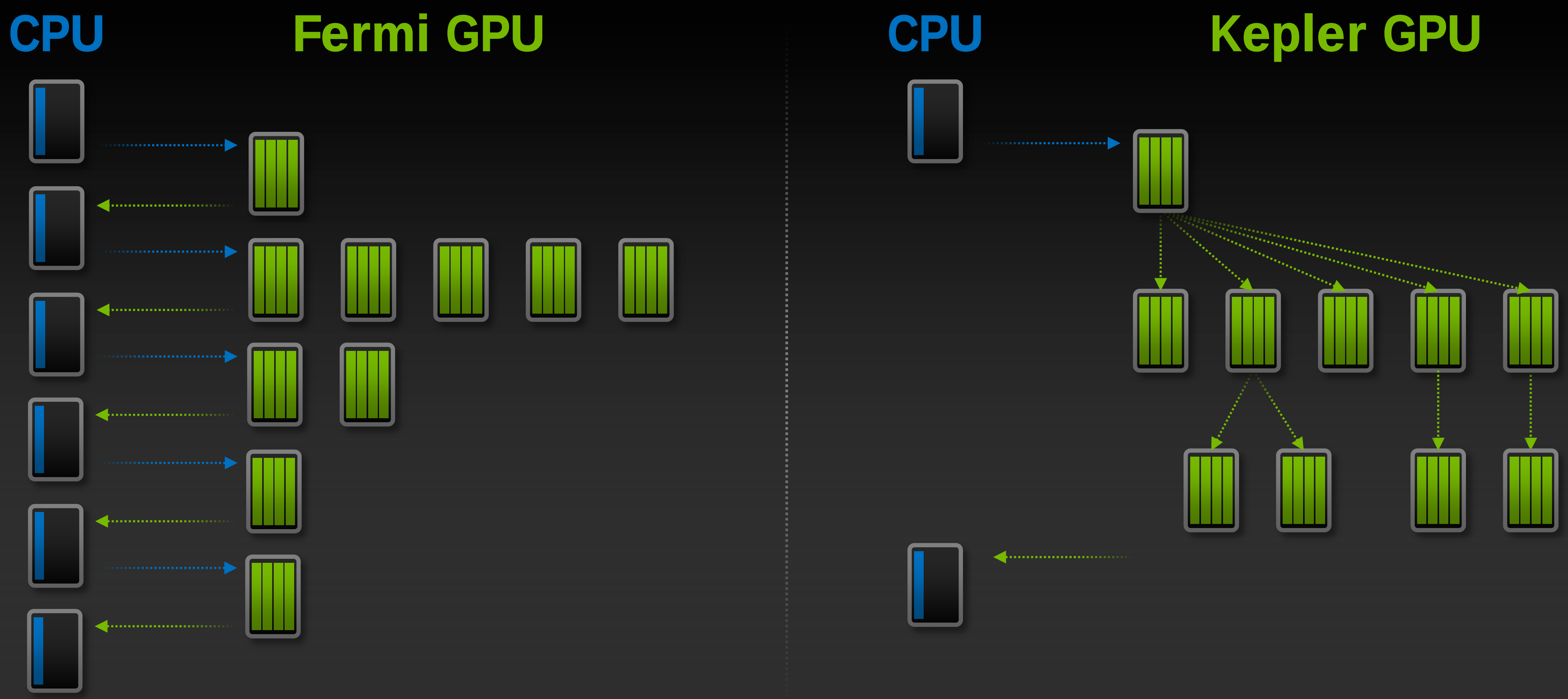
- Dynamically
- Simultaneously
- Independently



Fermi: Only CPU can generate GPU work

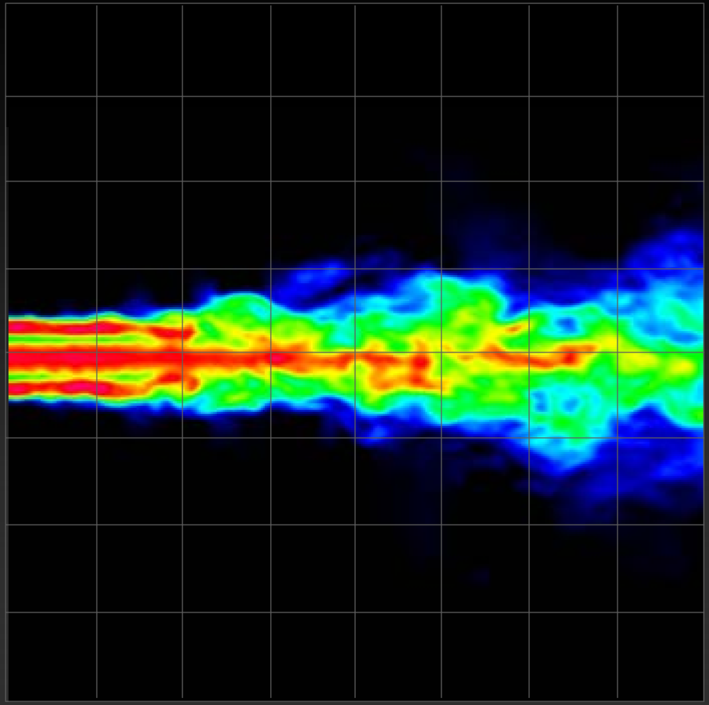
Kepler: GPU can generate work for itself

Dynamic Parallelism



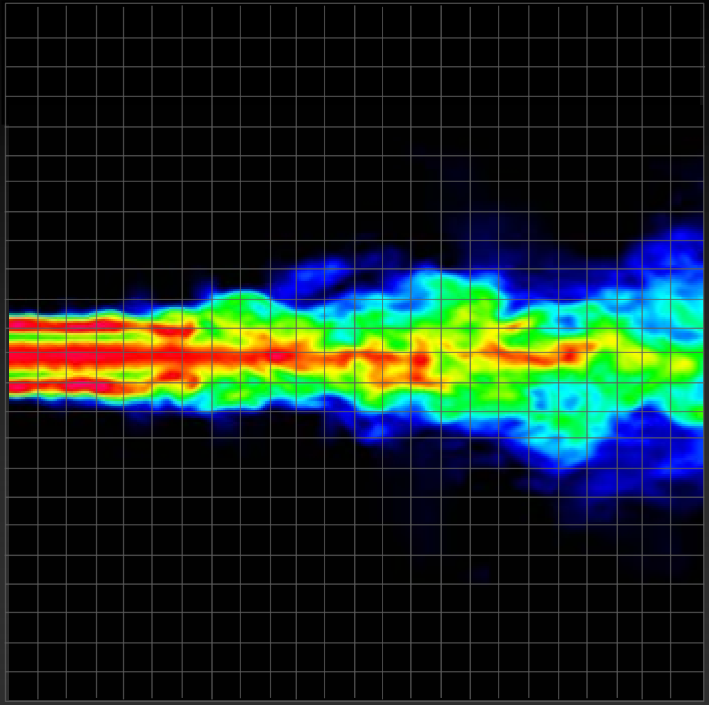
Dynamic Work Generation

Coarse grid



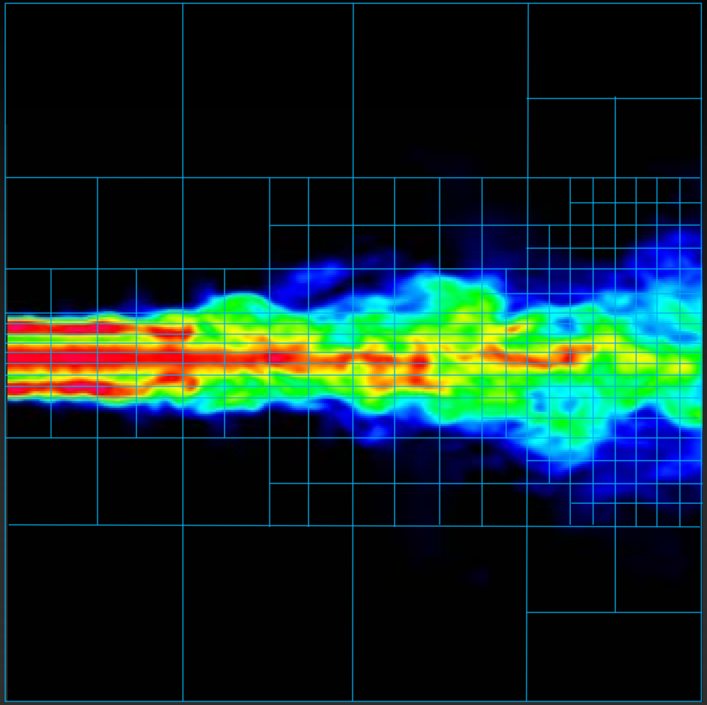
Higher Performance
Lower Accuracy

Fine grid



Higher Accuracy
Lower Performance

Dynamic grid



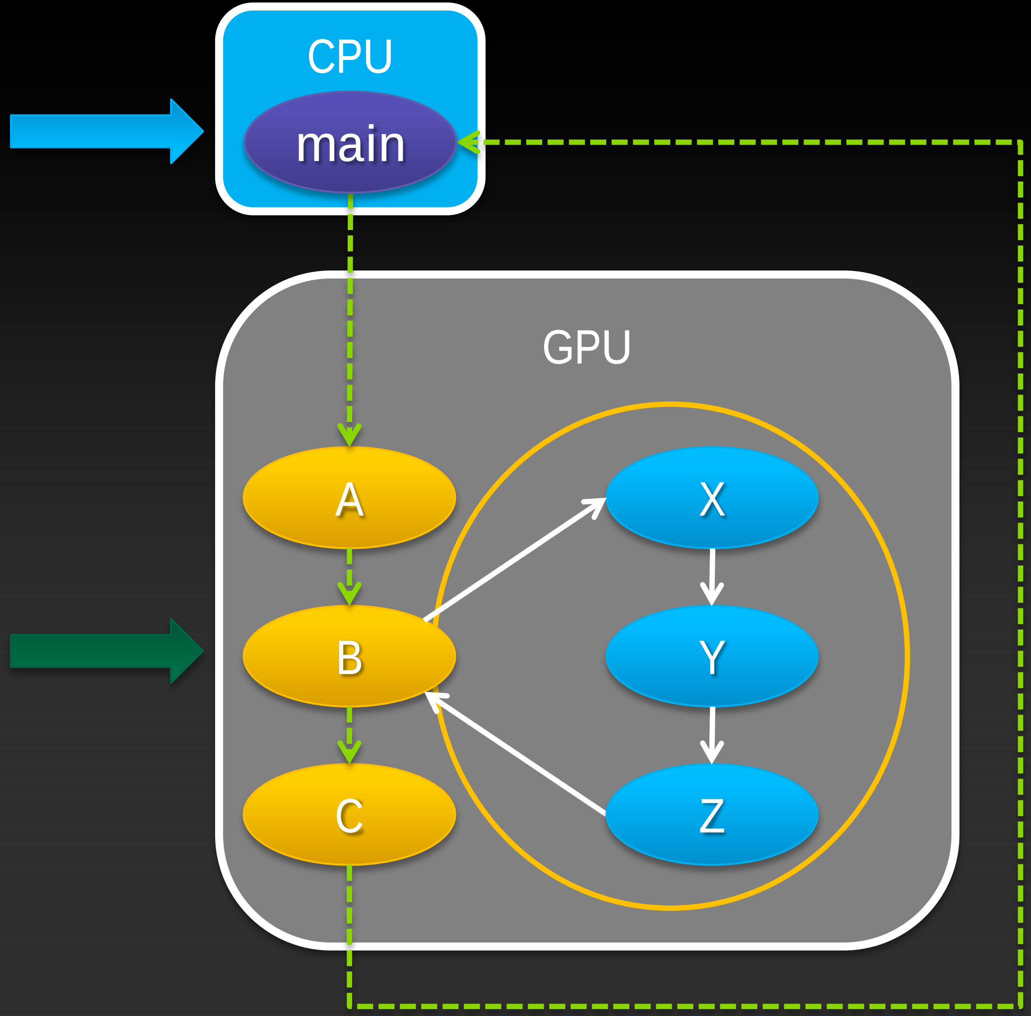
Target performance where accuracy is required

```
ears:     3898.34
IES:      280875
IES/SEC:7310286
```

Familiar Syntax and Programming Model

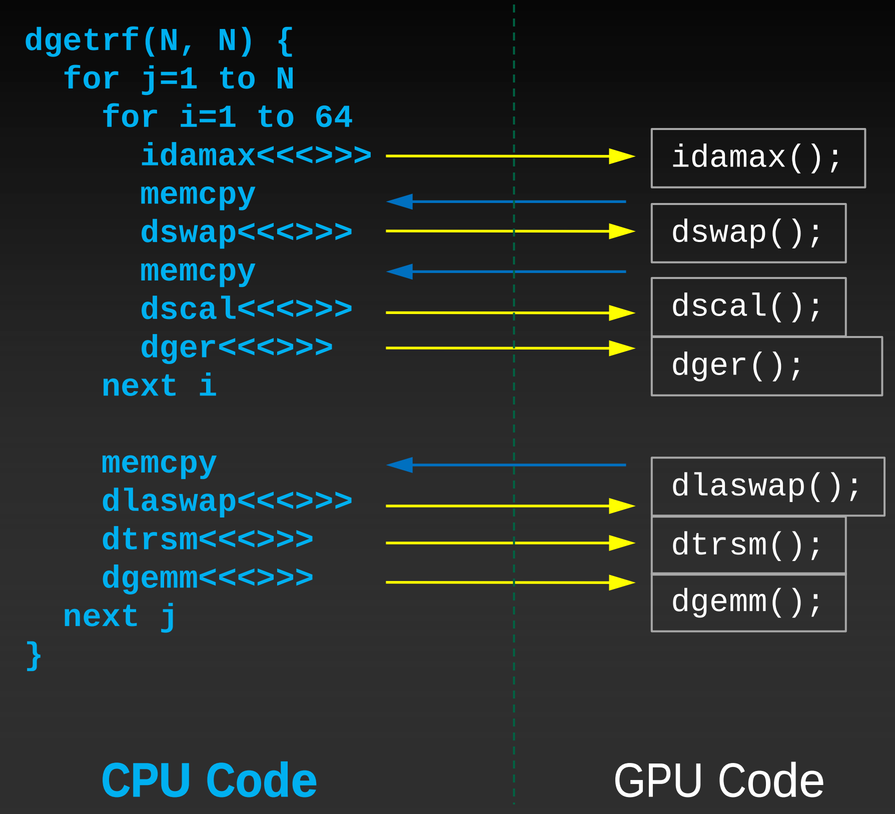
```
int main() {
    float *data;
    setup(data);

    A <<< ... >>> (data);
    B <<< ... >>> (data);
    C <<< ... >>> (data);

    cudaDeviceSynchronize();
    return 0;
}
```

```
__global__ void B(float *data)
{
    do_stuff(data);

    X <<< ... >>> (data);
    Y <<< ... >>> (data);
    Z <<< ... >>> (data);
    cudaDeviceSynchronize();

    do_more_stuff(data);
}
```



Simpler Code: LU Example

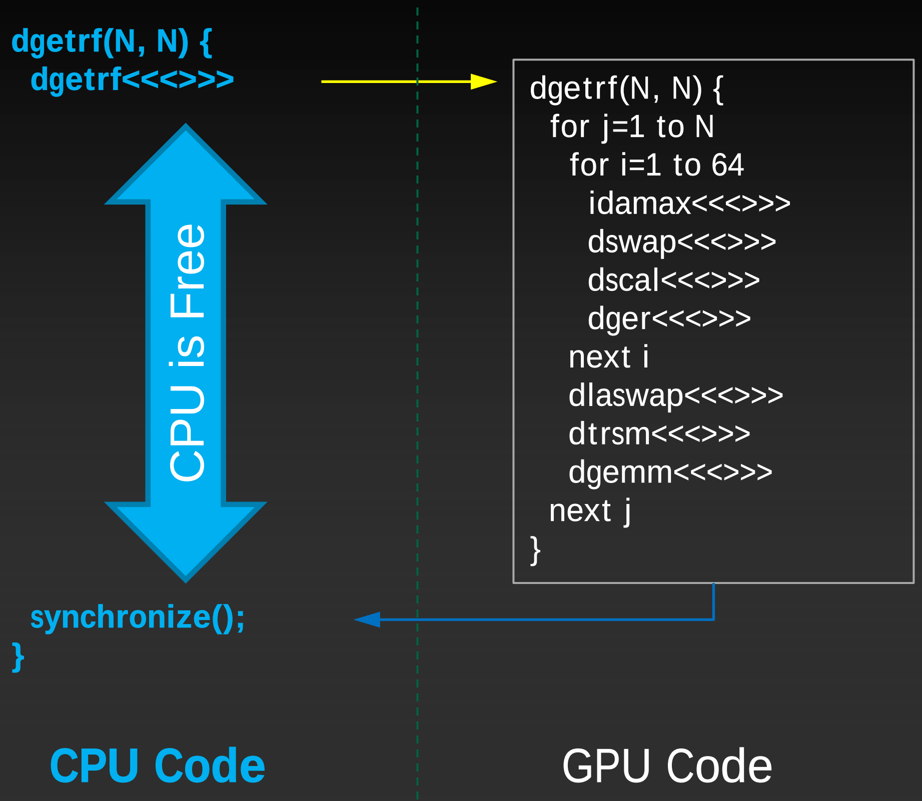
LU decomposition (Fermi)



```
dgetrf(N, N) {
  for j=1 to N
    for i=1 to 64
      idamax<<<>>>
      memcpy
      dswap<<<>>>
      memcpy
      dscal<<<>>>
      dger<<<>>>
    next i

    memcpy
    dlaswap<<<>>>
    dtrsm<<<>>>
    dgemm<<<>>>
  next j
}
```

```
idamax();
dswap();
dscal();
dger();
dlaswap();
dtrsm();
dgemm();
```

LU decomposition (Kepler)



```
dgetrf(N, N) {
 dgetrf<<<>>>

 synchronize();
}
```

```
dgetrf(N, N) {
  for j=1 to N
    for i=1 to 64
      idamax<<<>>>
      dswap<<<>>>
      dscal<<<>>>
      dger<<<>>>
    next i
    dlaswap<<<>>>
    dtrsm<<<>>>
    dgemm<<<>>>
  next j
}
```

CUDA By the Numbers:

>375,000,000	CUDA-Capable GPUs
>1,000,000	Toolkit Downloads
>120,000	Active Developers
>500	Universities Teaching CUDA

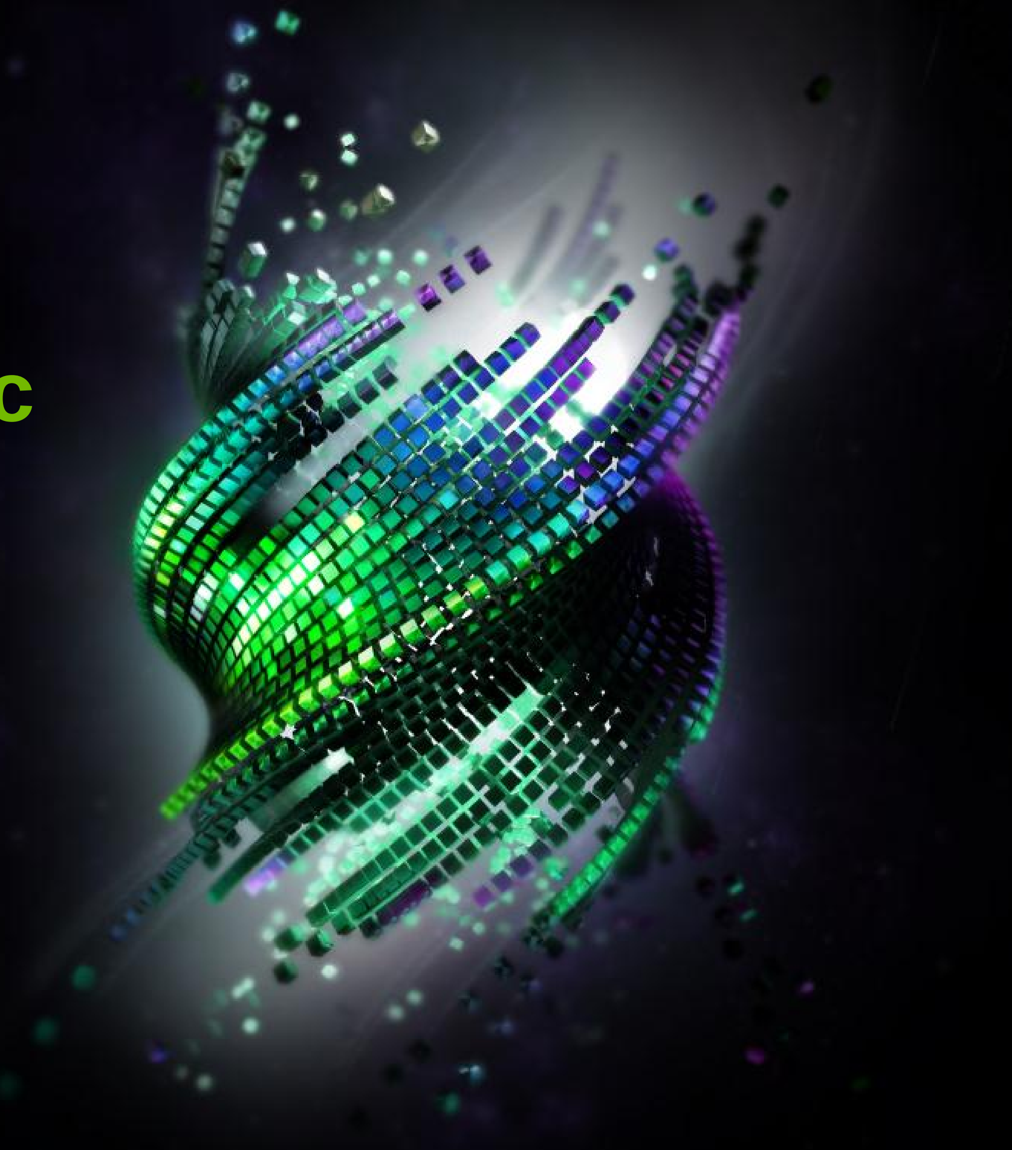
CUDA 5

Nsight™ for Linux & Mac

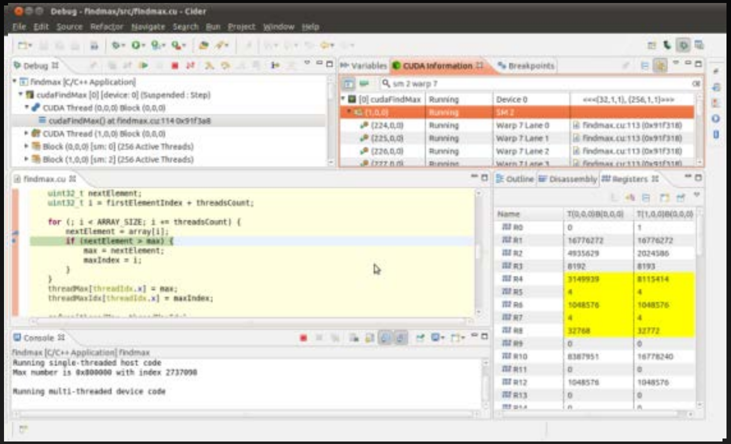
NVIDIA GPUDirect™

Library Object Linking

Preview Release
Now Available



NVIDIA® Nsight™ Eclipse Edition


Debug - findmax/src/findmax.cu - Cider
File Edit Source Refactor Navigate Search Run Project Window Help
Debug
Variables
CUDA Information
Breakpoints
sm 2 warp 7
findmax.cu
Outline
Disassembly
Registers
Console

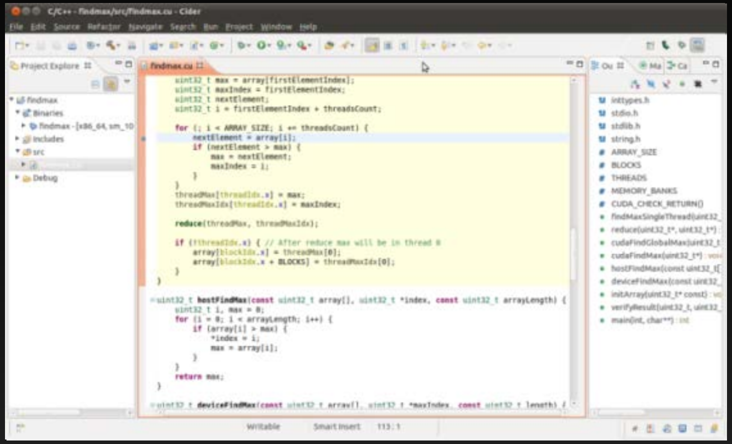
```
findmax [C/C++ Application]
  cudaFindMax [0] [device: 0] (Suspended : Step)
    CUDA Thread (0,0,0) Block (0,0,0)
      cudaFindMax() at findmax.cu:114 0x91f3a8
    CUDA Thread (1,0,0) Block (0,0,0)
    Block (0,0,0) [sm: 0] (256 Active Threads)
    Block (1,0,0) [sm: 2] (256 Active Threads)
```

[0] cudaFindMax	Running	Device 0	<<<(32,1,1),(256,1,1)>>>
(1,0,0)	Running	SM 2	
(224,0,0)	Running	Warp 7 Lane 0	findmax.cu:113 (0x91f318)
(225,0,0)	Running	Warp 7 Lane 1	findmax.cu:113 (0x91f318)
(226,0,0)	Running	Warp 7 Lane 2	findmax.cu:113 (0x91f318)
(227,0,0)	Running	Warp 7 Lane 3	findmax.cu:113 (0x91f318)

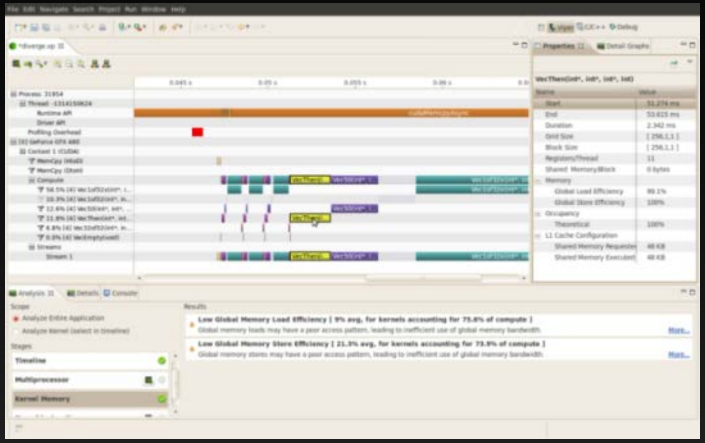
```
    uint32_t nextElement;
    uint32_t i = firstElementIndex + threadsCount;

    for (; i < ARRAY_SIZE; i += threadsCount) {
        nextElement = array[i];
        if (nextElement > max) {
            max = nextElement;
            maxIndex = i;
        }
    }
    threadMax[threadIdx.x] = max;
    threadMaxIdx[threadIdx.x] = maxIndex;
```

Name	T(0,0,0)B(0,0,0)	T(1,0,0)B(0,0,0)
R0	0	1
R1	16776272	16776272
R2	4935629	2024586
R3	8192	8193
R4	3149939	8115414
R5	4	4
R6	1048576	1048576
R7	4	4
R8	32768	32772
R9	0	0
R10	8387951	16778240
R11	0	0
R12	1048576	1048576
R13	0	0

```
findmax [C/C++ Application] findmax
Running single-threaded host code
Max number is 0x800000 with index 2737098

Running multi-threaded device code
```


C/C++ - findmax/src/findmax.cu - Cider
File Edit Source Refactor Navigate Search Run Project Window Help
Project Explore
findmax.cu
findmax
Binaries
findmax - [x86_64, sm_10
Includes
src
Debug
inttypes.h
stdio.h
stdlib.h
string.h
ARRAY_SIZE
BLOCKS
THREADS
MEMORY_BANKS
CUDA_CHECK_RETURN()
findMaxSingleThread(uint32_
reduce(uint32_t*, uint32_t*)
cudaFindGlobalMax(uint32_t
cudaFindMax(uint32_t*) : void
hostFindMax(const uint32_t[]
deviceFindMax(const uint32_
initArray(uint32_t* const) : void
verifyResult(uint32_t, uint32_
main(int, char**) : int
Writable
Smart Insert
113 : 1

```
    uint32_t max = array[firstElementIndex];
    uint32_t maxIndex = firstElementIndex;
    uint32_t nextElement;
    uint32_t i = firstElementIndex + threadsCount;

    for (; i < ARRAY_SIZE; i += threadsCount) {
        nextElement = array[i];
        if (nextElement > max) {
            max = nextElement;
            maxIndex = i;
        }
    }
    threadMax[threadIdx.x] = max;
    threadMaxIdx[threadIdx.x] = maxIndex;

    reduce(threadMax, threadMaxIdx);

    if (!threadIdx.x) { // After reduce max will be in thread 0
        array[blockIdx.x] = threadMax[0];
        array[blockIdx.x + BLOCKS] = threadMaxIdx[0];
    }
}

uint32_t hostFindMax(const uint32_t array[], uint32_t *index, const uint32_t arrayLength) {
    uint32_t i, max = 0;
    for (i = 0; i < arrayLength; i++) {
        if (array[i] > max) {
            *index = i;
            max = array[i];
        }
    }
    return max;
}

uint32_t deviceFindMax(const uint32_t array[], uint32_t *maxIndex, const uint32_t length) {
```



Low Global Memory Load Efficiency [9% avg. for kernels accounting for 75.6% of compute]
Global memory loads may have a poor access pattern, leading to inefficient use of global memory bandwidth.

Low Global Memory Store Efficiency [21.3% avg. for kernels accounting for 73.9% of compute]
Global memory stores may have a poor access pattern, leading to inefficient use of global memory bandwidth.

CUDA-Aware Editor

- Automated CPU to GPU code refactoring
- Semantic highlighting of CUDA code
- Integrated code samples & docs

Nsight Debugger

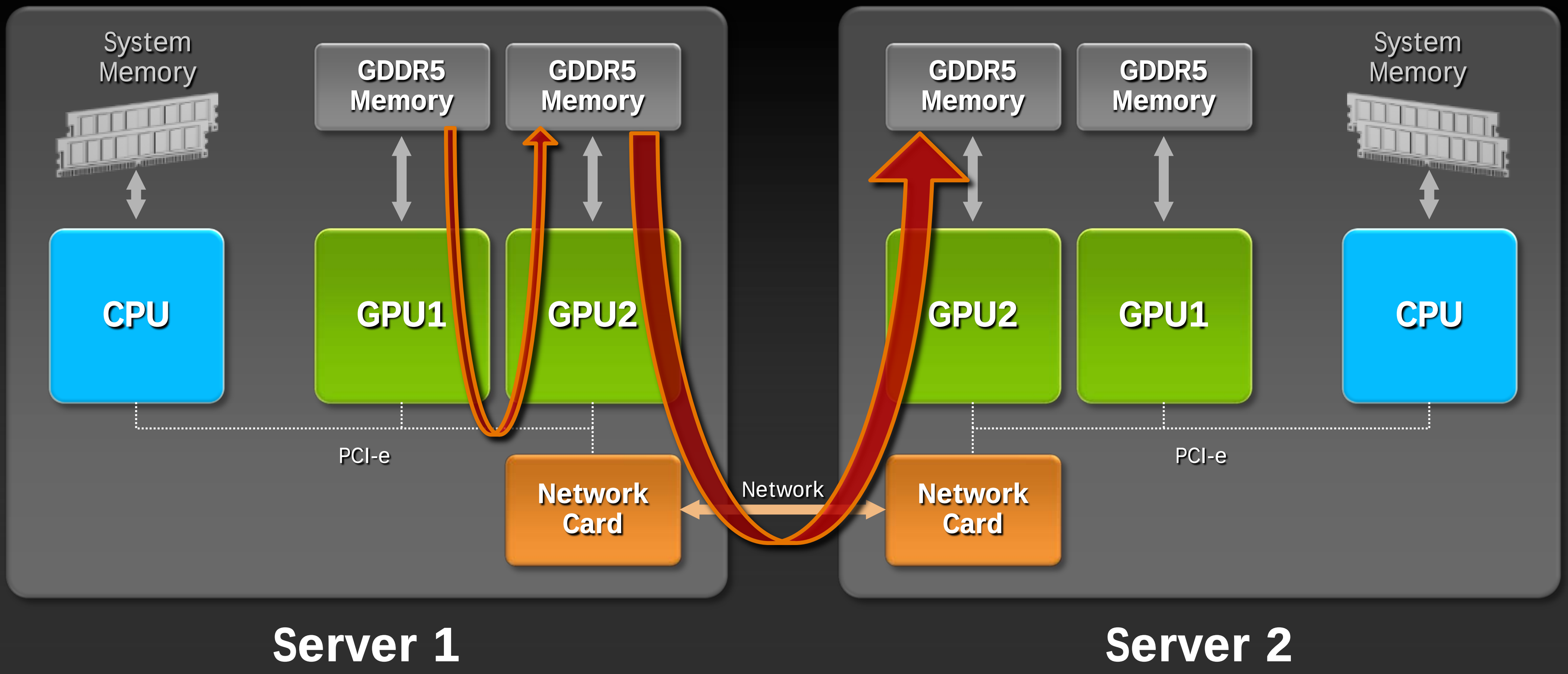
- Simultaneously debug of CPU and GPU
- Inspect variables across CUDA threads
- Use breakpoints & single-step debugging

Nsight Profiler

- Quickly identifies performance issues
- Integrated expert system
- Automated analysis
- Source line correlation

Available for Linux and Mac OS

Kepler Enables Full NVIDIA GPUDirect™



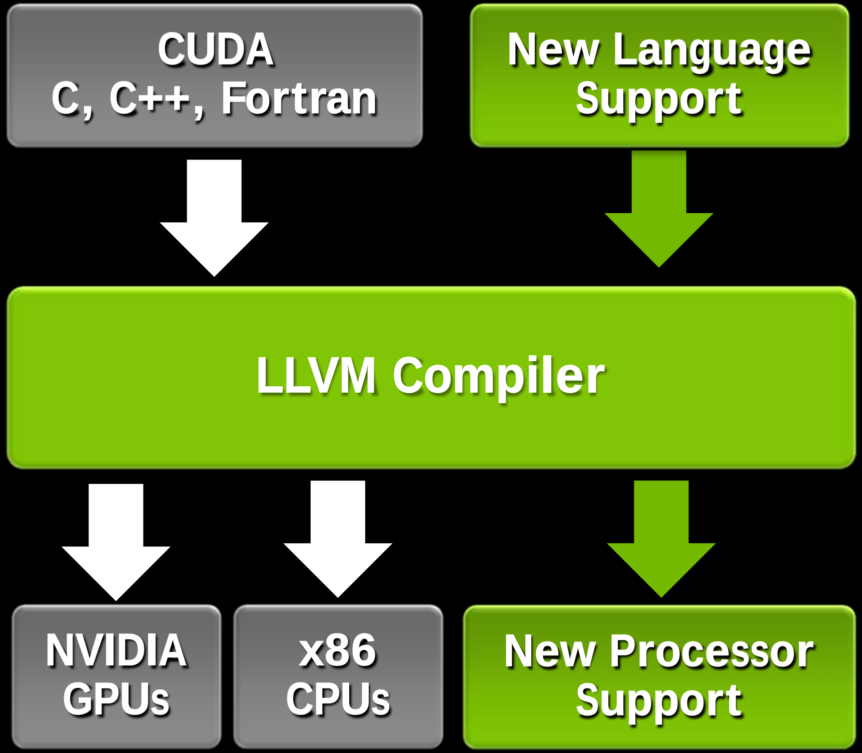
GPU Computing with LLVM

Developers want to build front-ends for

Java, Python, R, DSLs

Target other processors like

ARM, FPGA, GPUs, x86



CUDA
C, C++, Fortran

New Language Support

LLVM Compiler

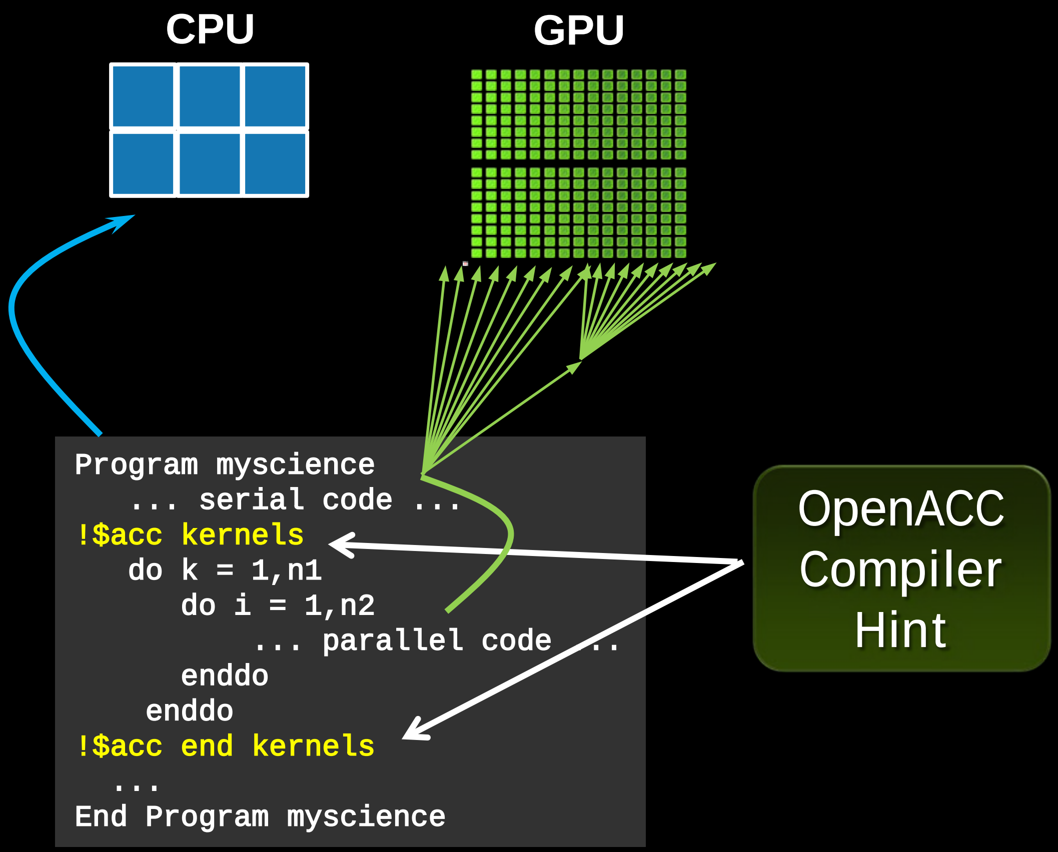
NVIDIA GPUs

x86 CPUs

New Processor Support

LLVM COMPILER INFRASTRUCTURE

OpenACC Directives



```
Program myscience
   ... serial code ...
!$acc kernels
   do k = 1,n1
      do i = 1,n2
          ... parallel code ...
      enddo
    enddo
!$acc end kernels
  ...
End Program myscience
```

Your original
Fortran or C code

Simple Compiler hints

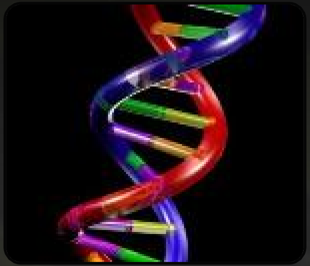
Compiler Parallelizes code

Portability, Productivity,
Performance

Performance: Leveraging GPU

Reading DNA nucleotide sequences

Shanghai JiaoTong University



4 directives

16x faster

Designing circuits for quantum computing

UIST, Macedonia

1 week

40x faster

Extracting image features in real-time

Aselsan

3 directives

4.1x faster

HydroC- Galaxy Formation

PRACE Benchmark Code, CAPS

1 week

3x faster

Real-time Derivative Valuation

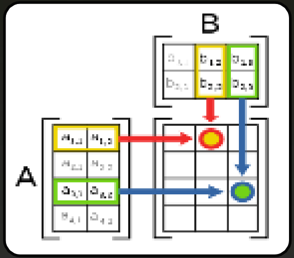
Opel Blue, Ltd

Few hours

70x faster

Matrix Matrix Multiply

Independent Research Scientist



4 directives

6.4x faster

Enabling ARM Ecosystem: CARMA DevKit

CUDA on ARM

CUDA GPU

Tegra ARM CPU

Tegra 3 Quad-core ARM A9
Quadro 1000M (96 CUDA cores)
Ubuntu

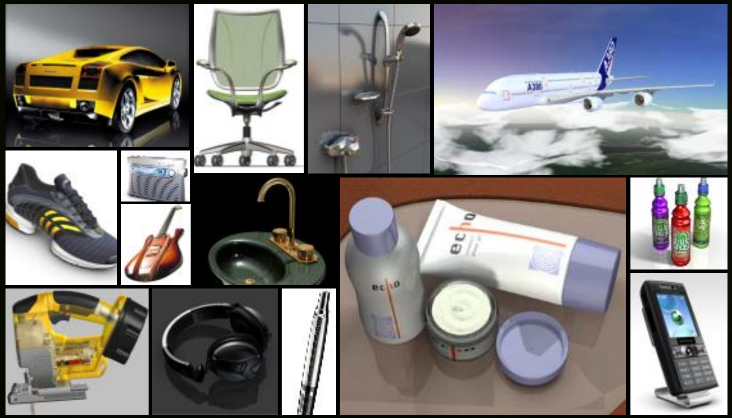
Gigabit Ethernet
SATA Connector
HDMI, DisplayPort, USB

The Day Job That Makes It All Possible…

- **Leverage volume graphics market to serve HPC**
  - HPC needs outstrip HPC market’s ability to fund the development
  - Computational graphics and compute are *highly* aligned

Tegra

GeForce



Quadro

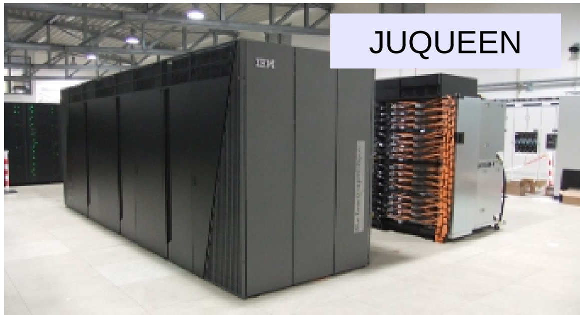
Jülich-NVIDIA Application Lab

19. June 2012 | Dirk Pleiter (JSC)

Supercomputing at Forschungszentrum Jülich

Role of the **Jülich Supercomputing Centre (JSC)**:

- **Operation** of supercomputers for local, national and European scientists
- **User support** including support of research communities by means of simulation laboratories
- **R&D** on future IT technologies, algorithms, tools, GRID, etc.
- **Education** and training of users


JUQUEEN

JUROPA

JUPACE

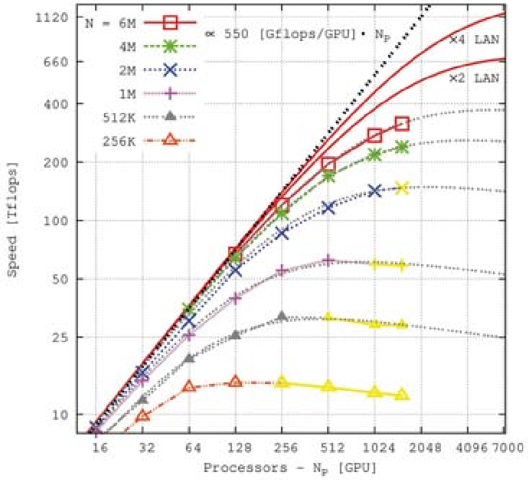
Our view on GPU computing

- Performance acceleration for a significant set of relevant scientific applications
- **JUDGE** = Jülich Dedicated GPU Environment
  - 206 node IBM iDataPlex cluster
  - Dual-CPU, dual-GPU nodes
  - About 240 TFlops (peak)
  - Partitions dedicated to astrophysics and brain research

- Large potential for **energy efficient computing**
  - JUDGE is #14 on Green500 (Nov. 2011)
  - Need for efficient utilisation of all computing devices

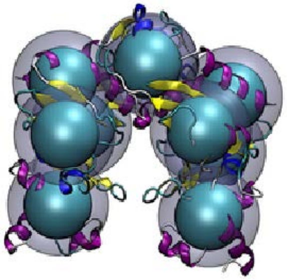
Jülich-NVIDIA Application Lab

- Lab hosted at JSC
- **Mission statement**
  - Enable scientific applications for GPU-based architectures
  - Provide support for optimization
  - Investigate performance and scaling

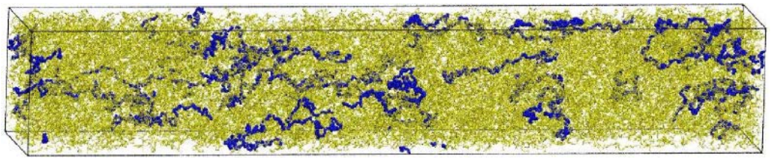


[R. Spurzem et al., 2012]

- **Targeted research areas**
  - Astrophysics and astronomy
  - Computational medicine and neuroscience
  - Elementary particle physics
  - Material science
  - Protein folding



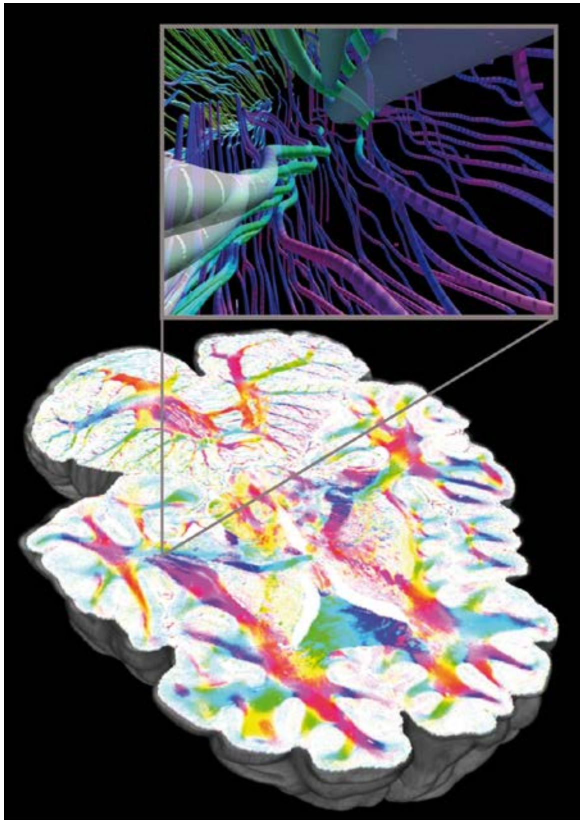
[O. Zimmerrmann, 2011]



[G. Sutmann et al., 2011]

Pilot application: JuBrain

- The **Jülich Brain Model** will display selected aspects of the brain's structural organization such as cortical areas and fiber tracts
  - Improve understanding of fiber operation
  - Help treating neurological disease
- Procedure
  - Preparation of brain sections
  - Image processing
  - 3D reconstruction and fiber tractography
- Already today significant speed-up using GPUs



[M. Axer et al., 2012]

Questions?