Inside the Kepler Architecture

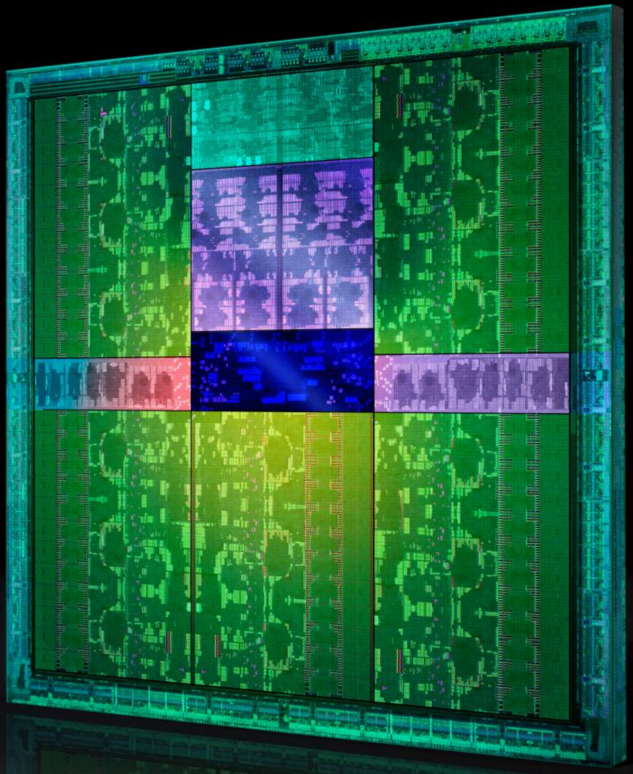
Stephen Jones – CUDA
NVIDIA Corporation

The Kepler GK110 GPU

Performance

Efficiency

Programmability

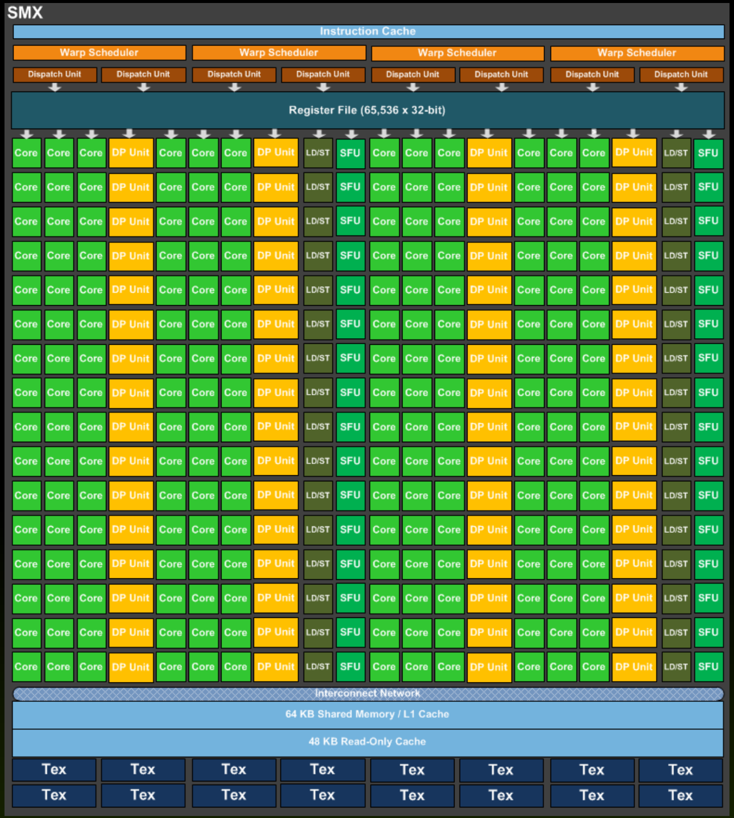


Kepler GK110 Block Diagram

Architecture

- 7.1B Transistors
- Up to 15 SMX units
- > 1 TFLOP FP64
- 1.5 MB L2 Cache
- 384-bit GDDR5

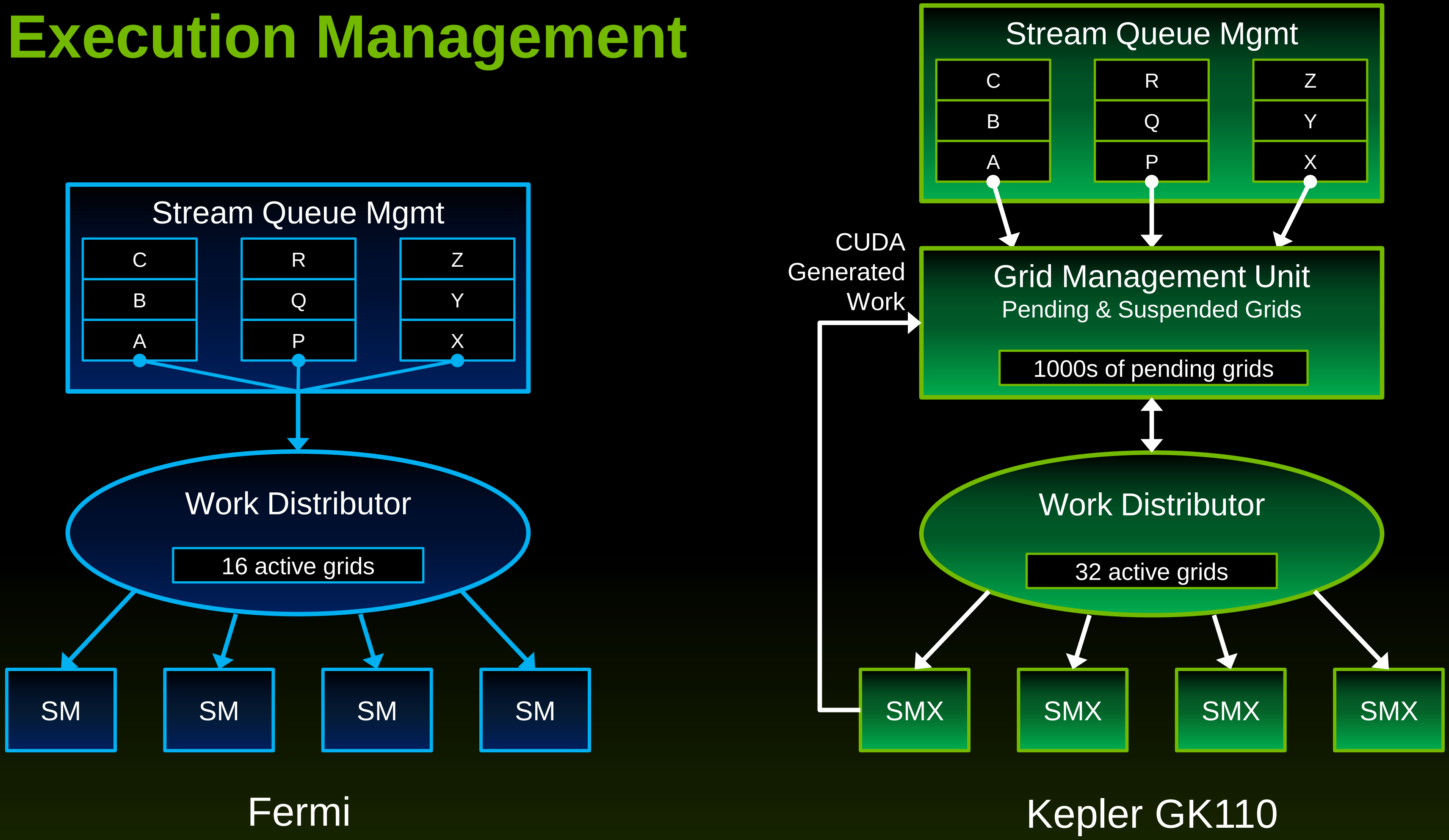
Kepler GK110 SMX vs Fermi SM



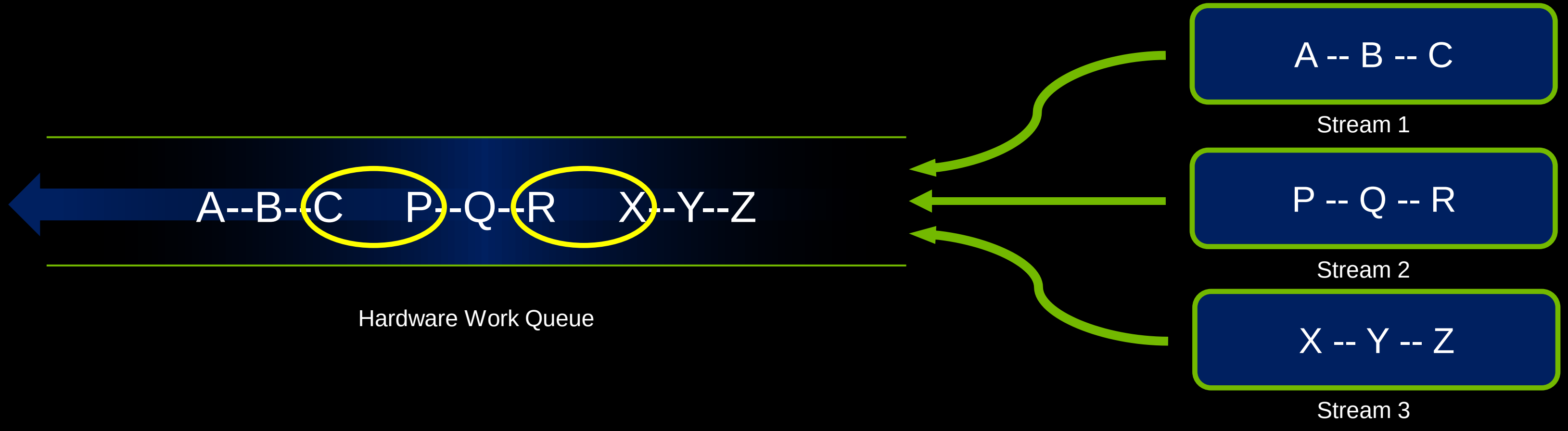
SMX Balance of Resources

Resource	Kepler GK110 vs Fermi
Floating point throughput	**2-3x**
Max Blocks per SMX	**2x**
Max Threads per SMX	**1.3x**
Register File Bandwidth	**2x**
Register File Capacity	**2x**
Shared Memory Bandwidth	**2x**
Shared Memory Capacity	**1x**

Execution Management



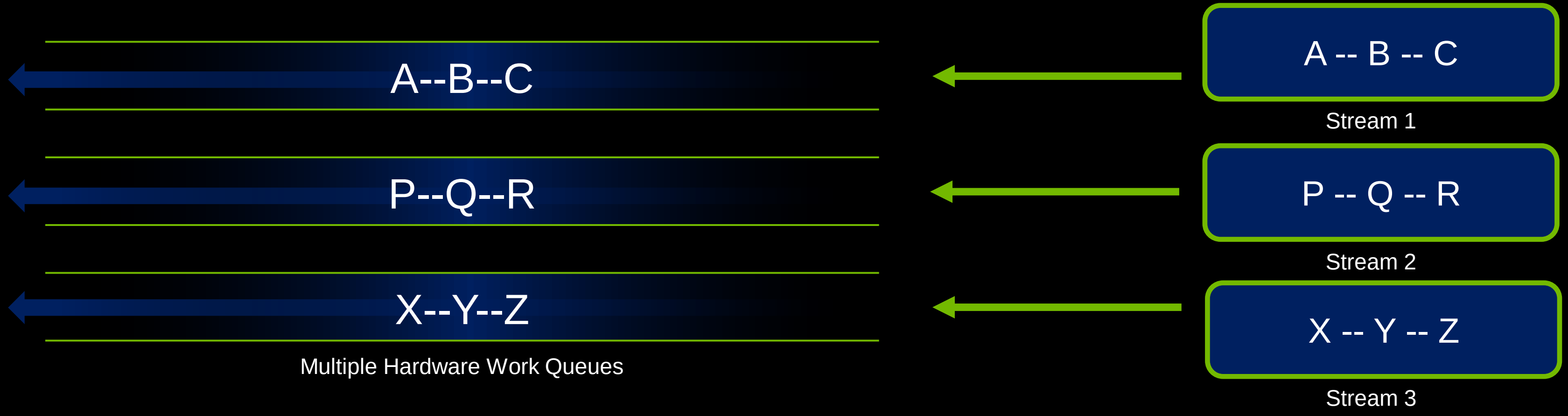
Fermi Concurrency



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

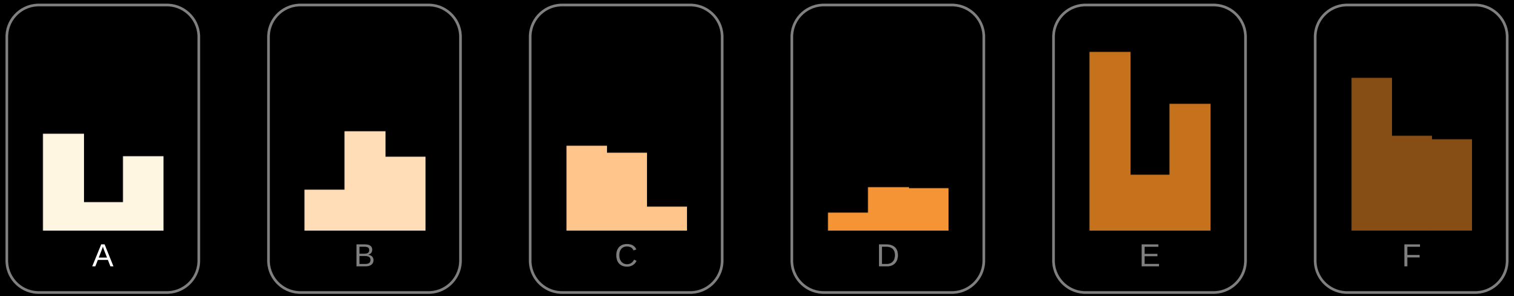
Kepler Improved Concurrency



Kepler allows 32-way concurrency

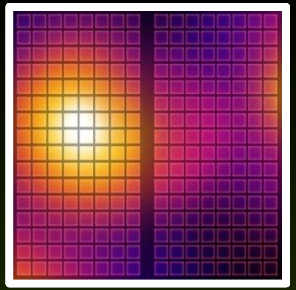
- One work queue per stream
- Concurrency at full-stream level
- No inter-stream dependencies

Hyper-Q: Time-Division Multiprocess

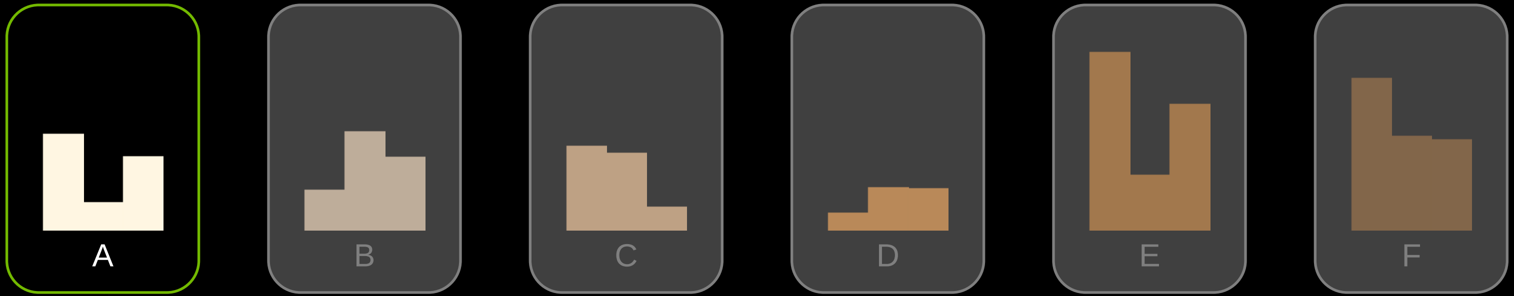


CPU Processes

Shared GPU

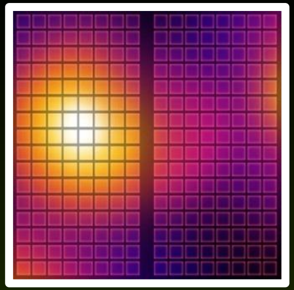


Fermi: Time-Division Multiprocess

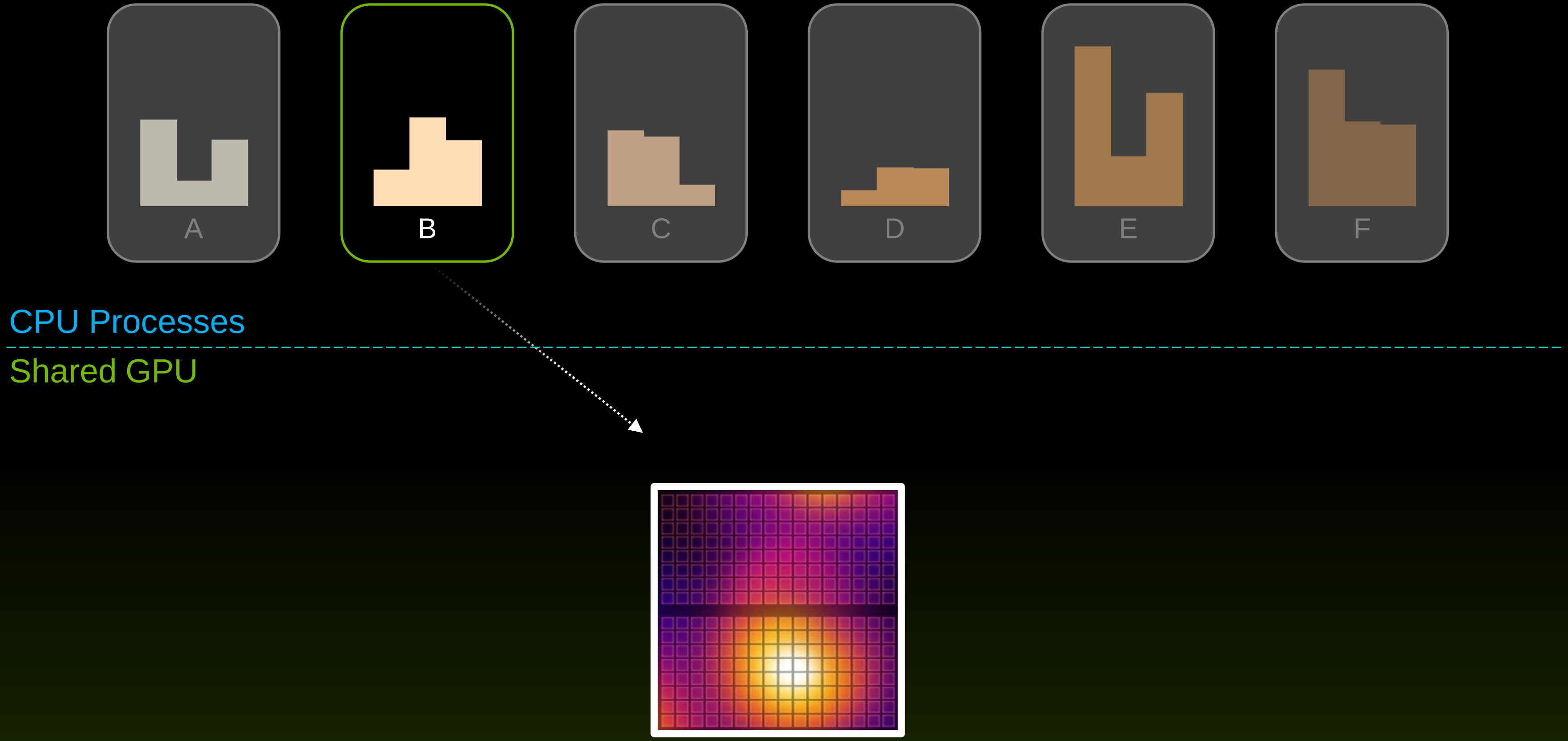


CPU Processes

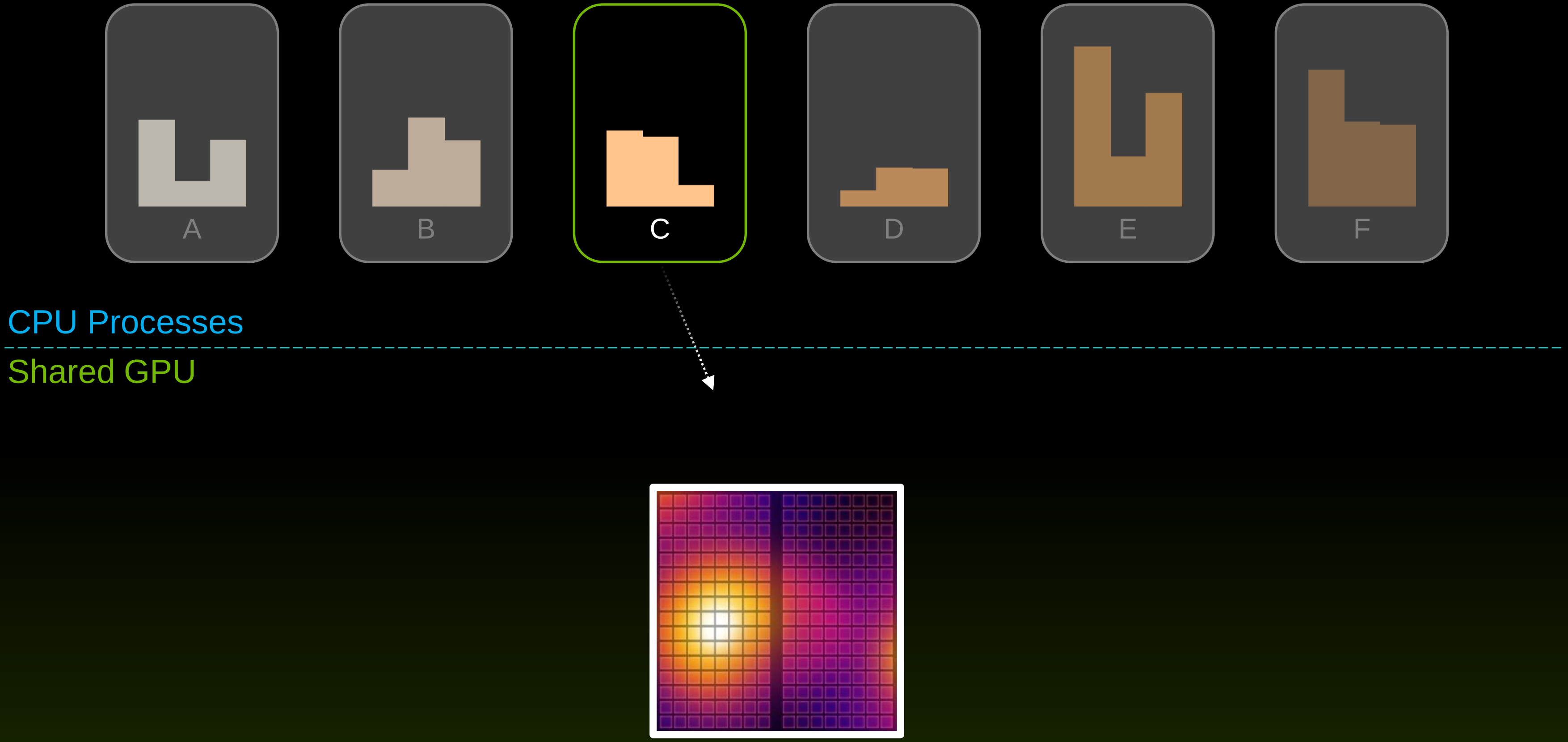
Shared GPU



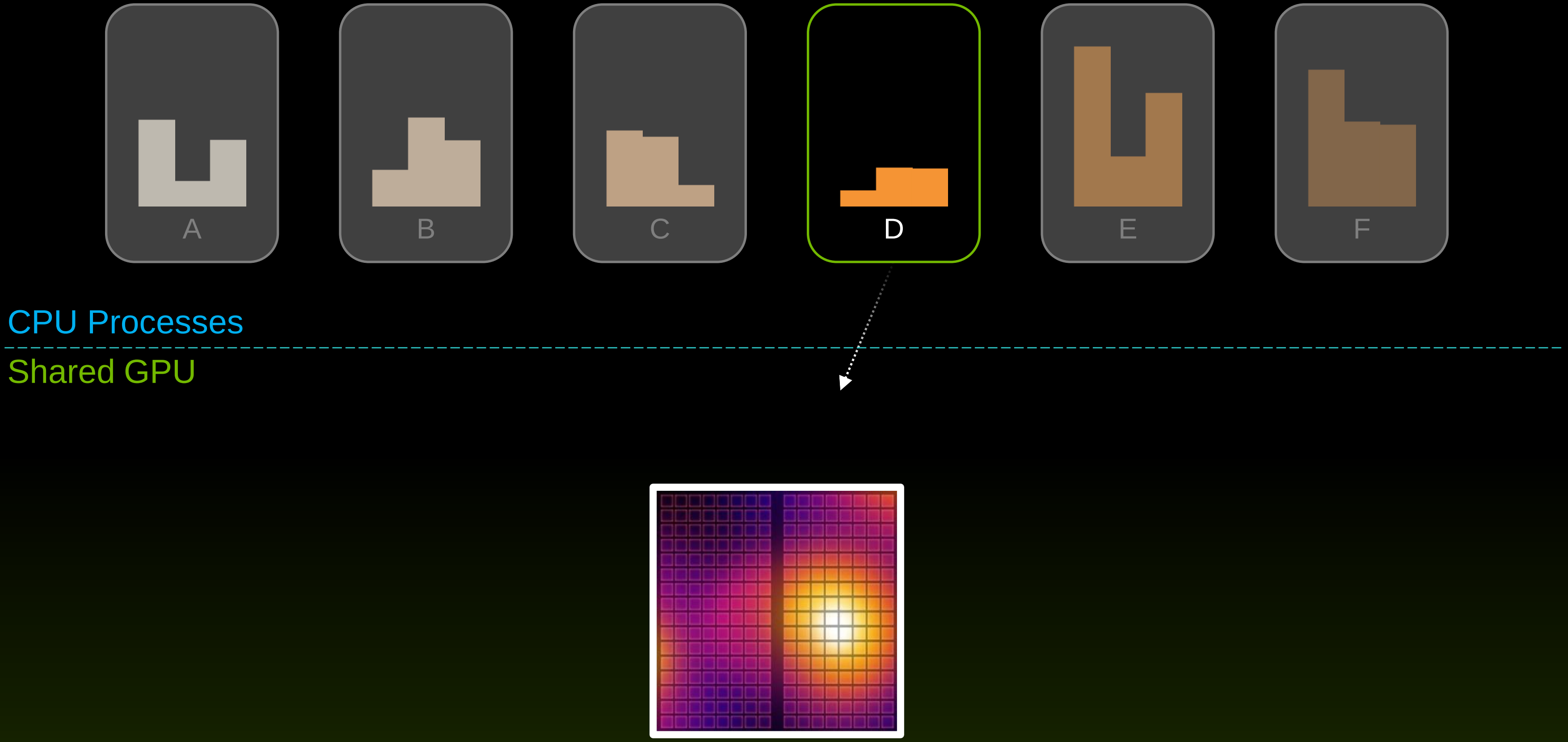
Fermi: Time-Division Multiprocess



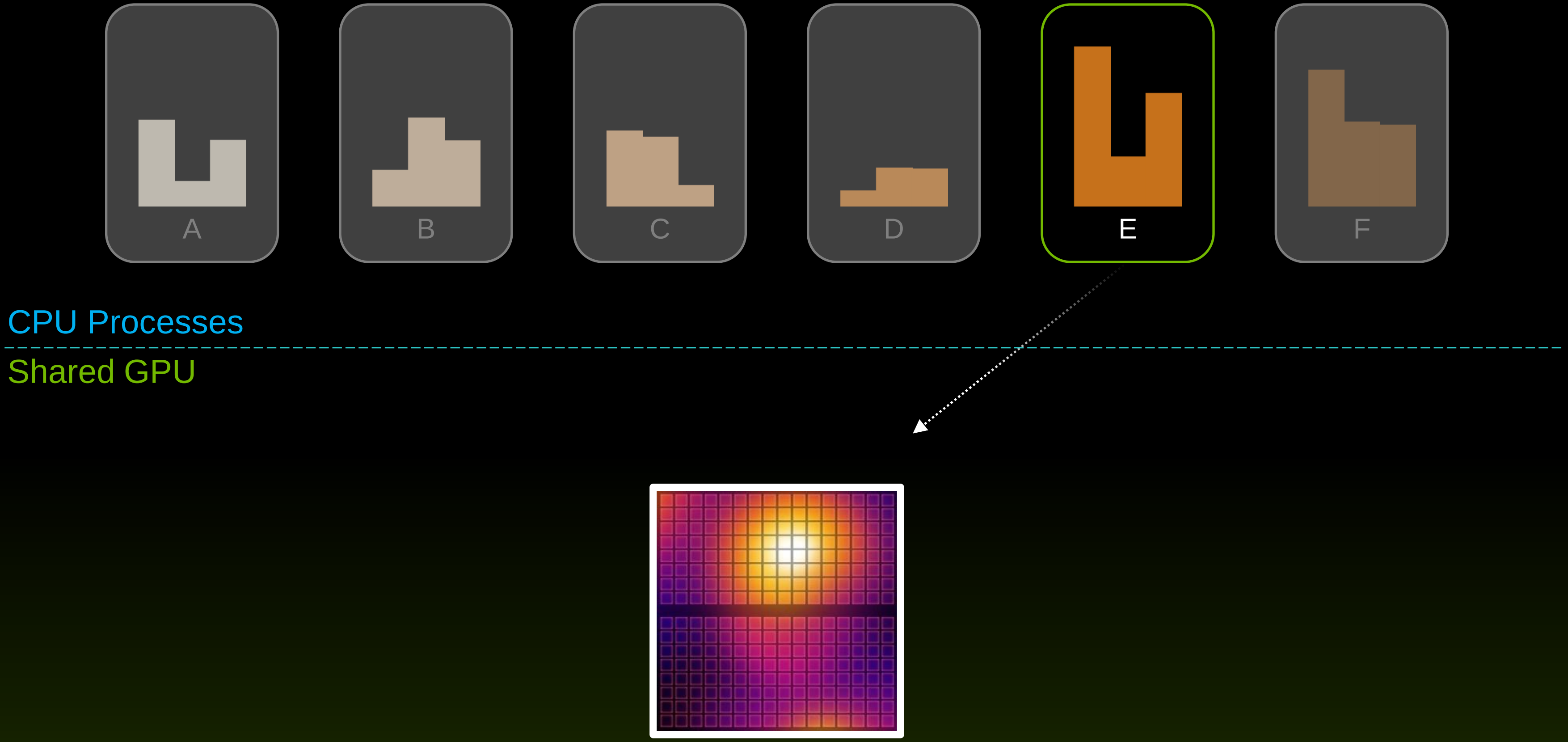
Fermi: Time-Division Multiprocess



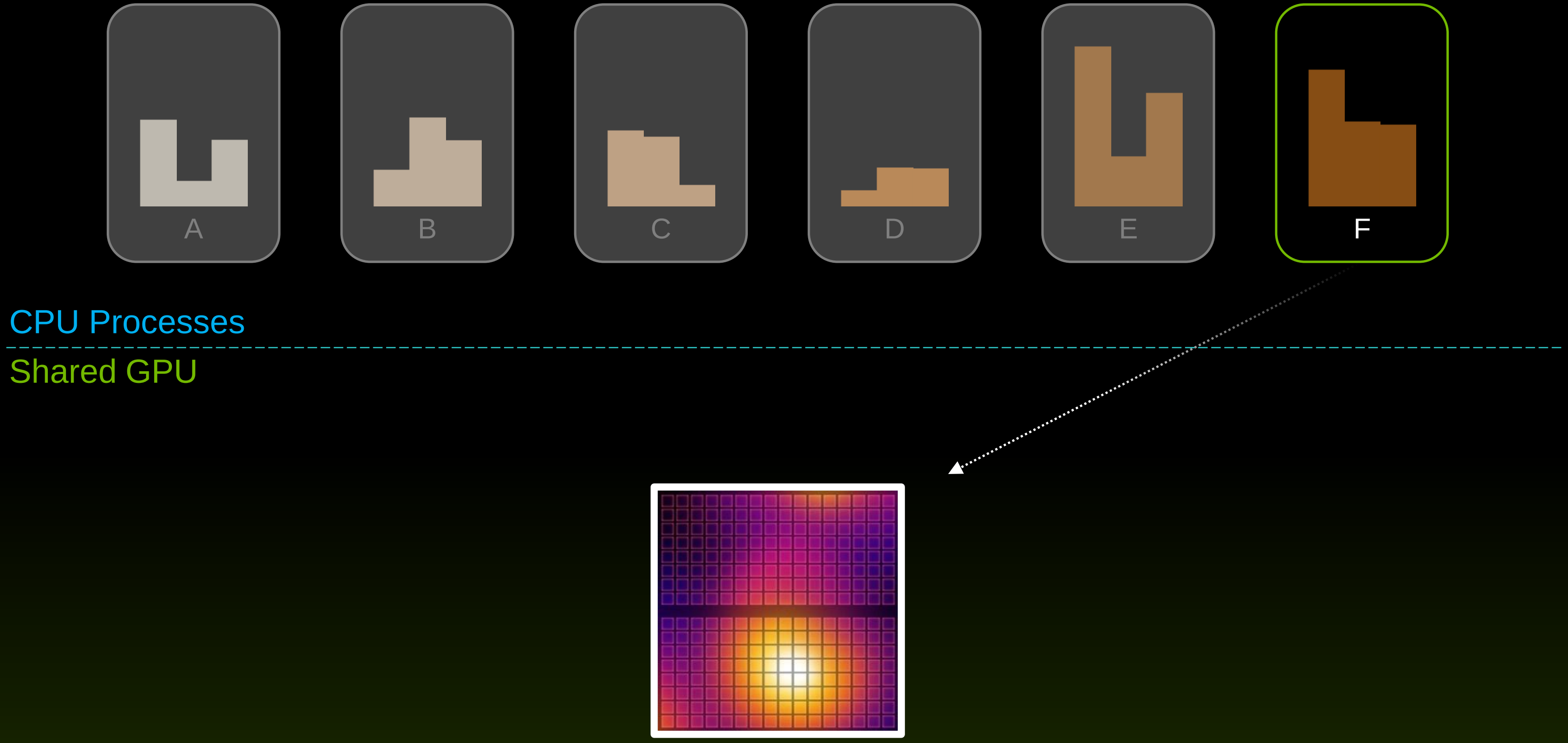
Fermi: Time-Division Multiprocess



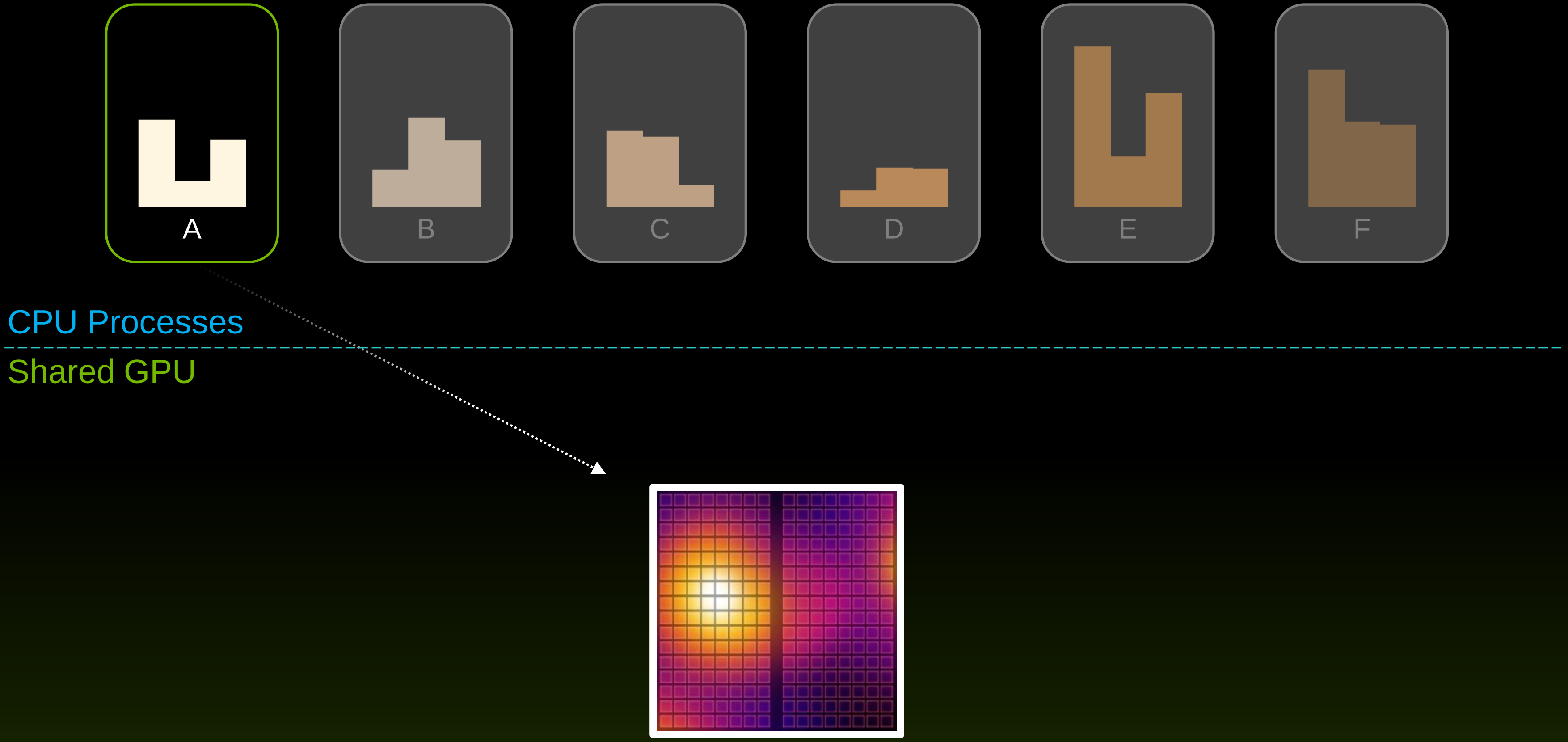
Fermi: Time-Division Multiprocess



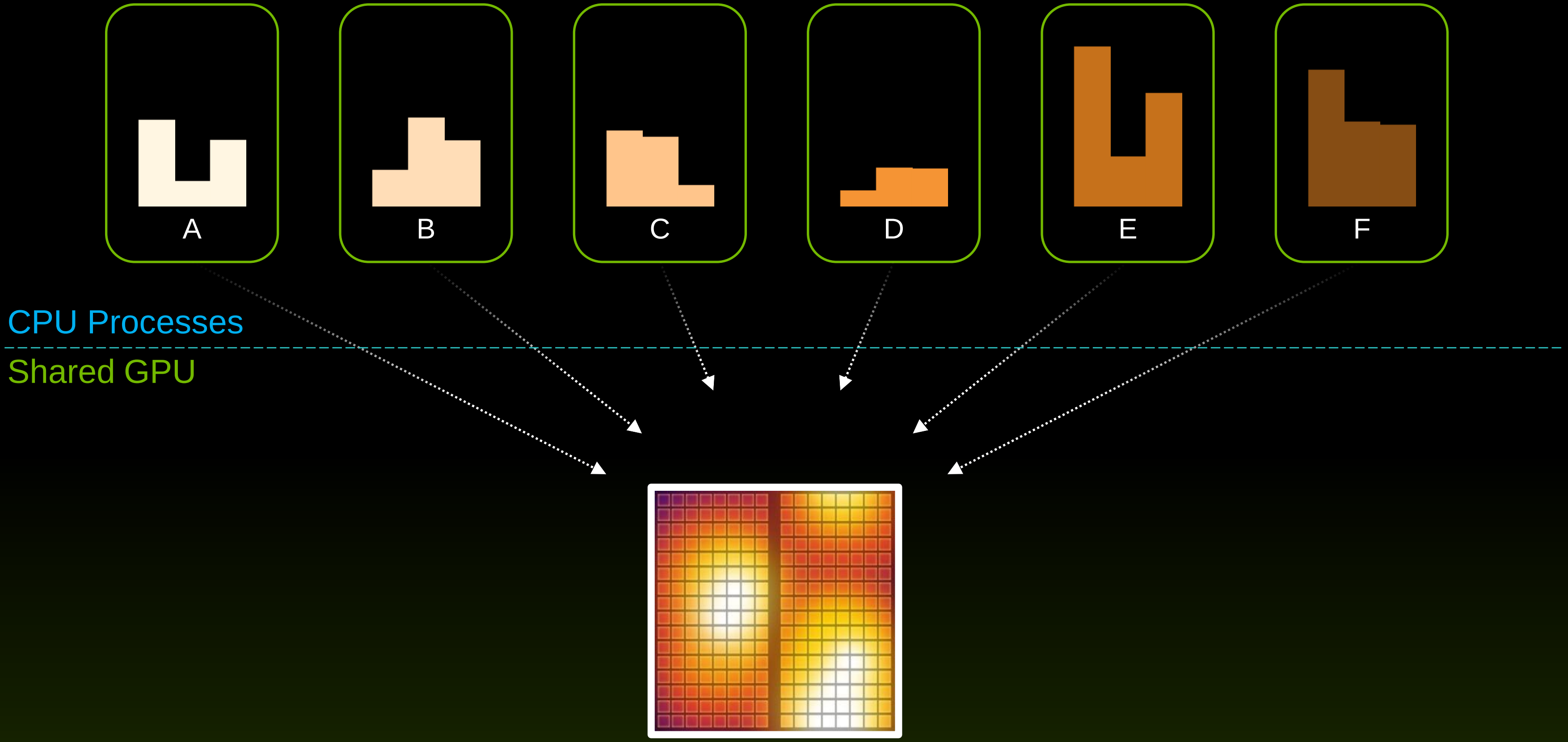
Fermi: Time-Division Multiprocess



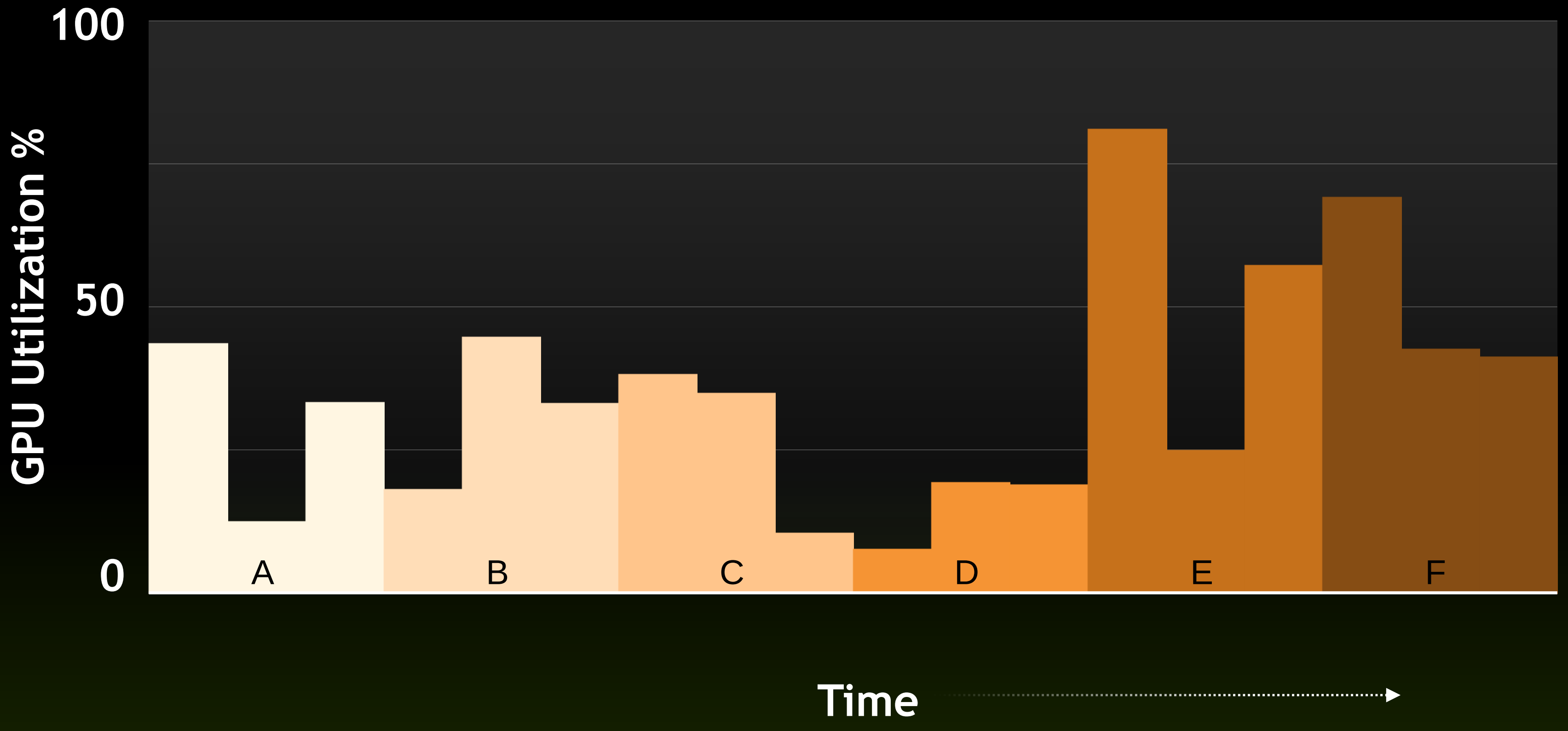
Fermi: Time-Division Multiprocess



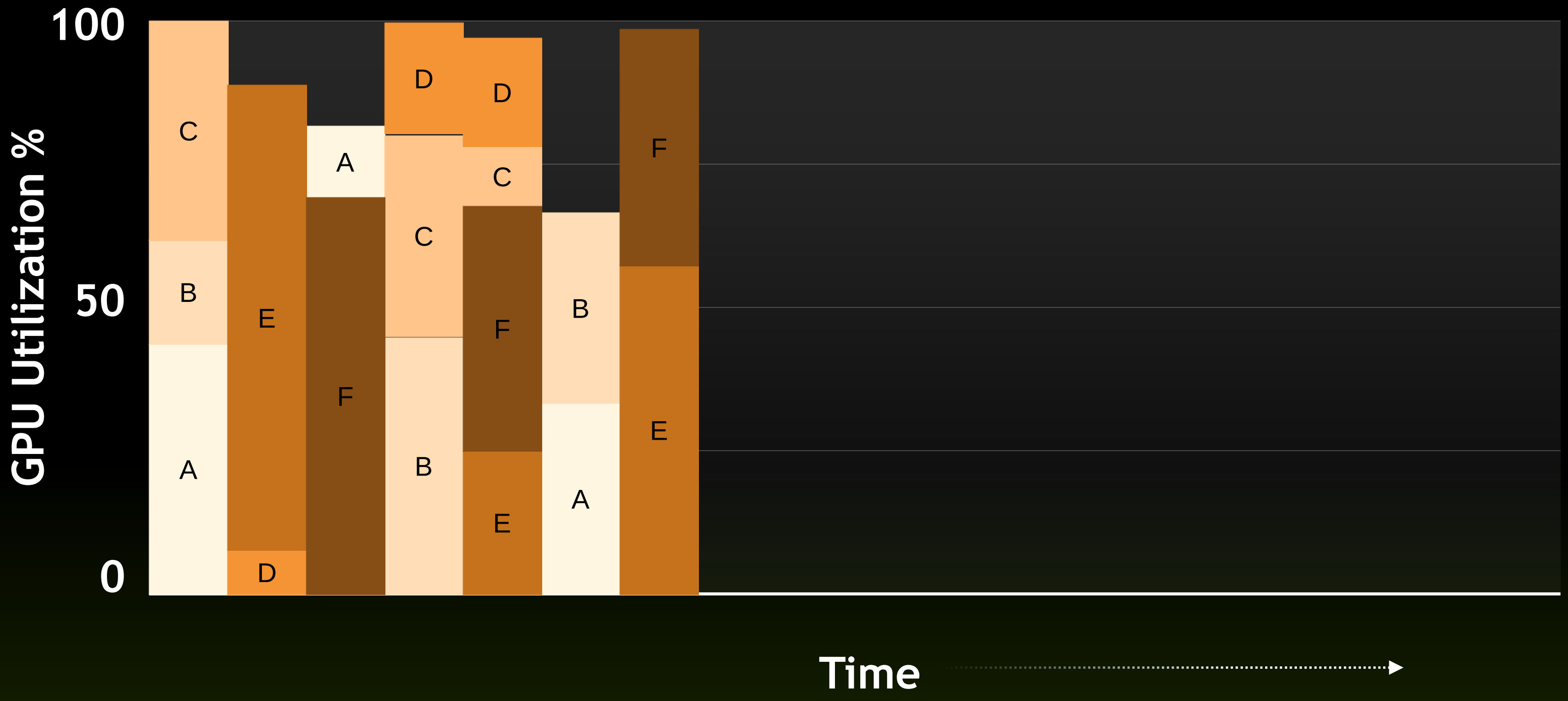
Hyper-Q: Simultaneous Multiprocess



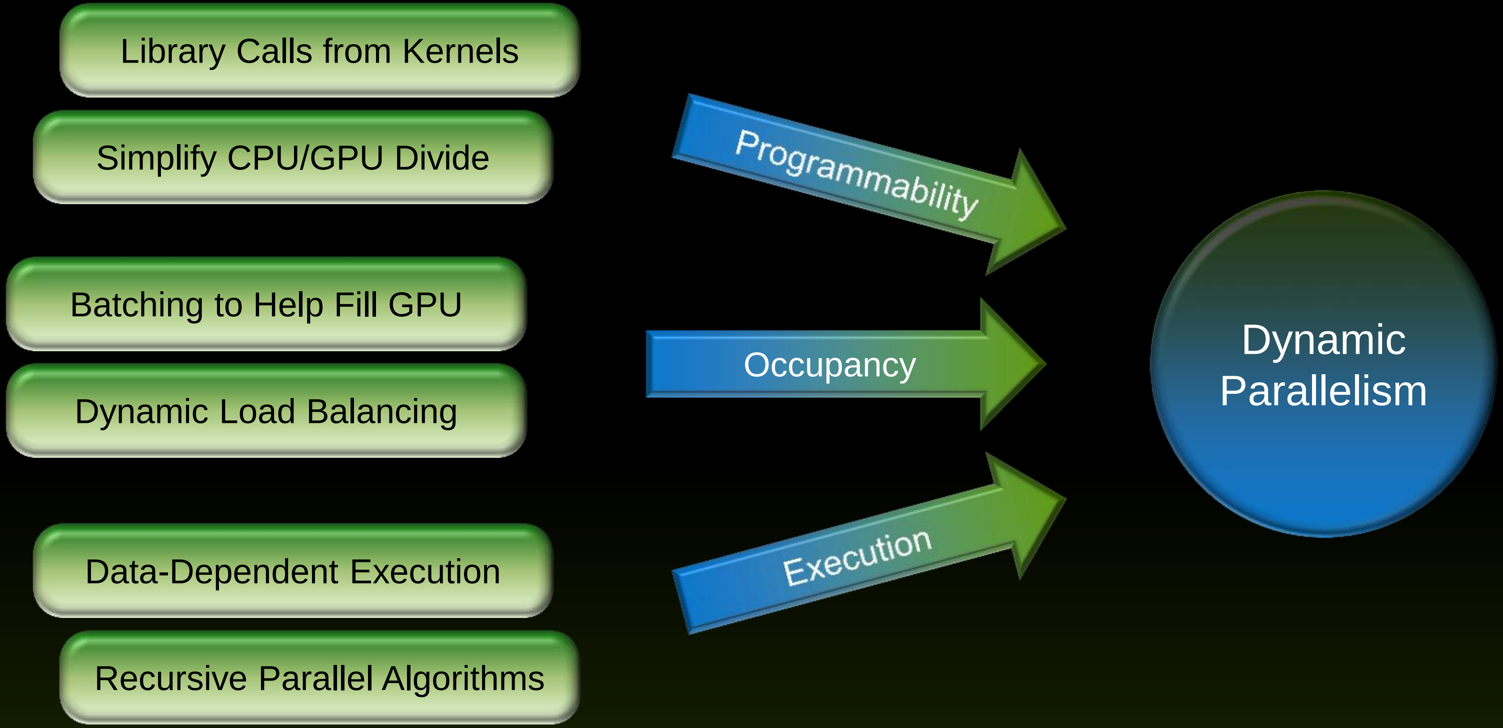
Without Hyper-Q



With Hyper-Q



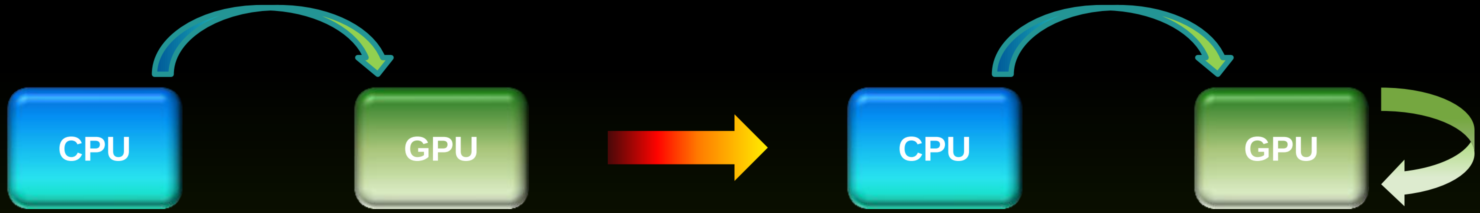
Improving Programmability



Dynamic Parallelism

The ability to launch new grids from the GPU

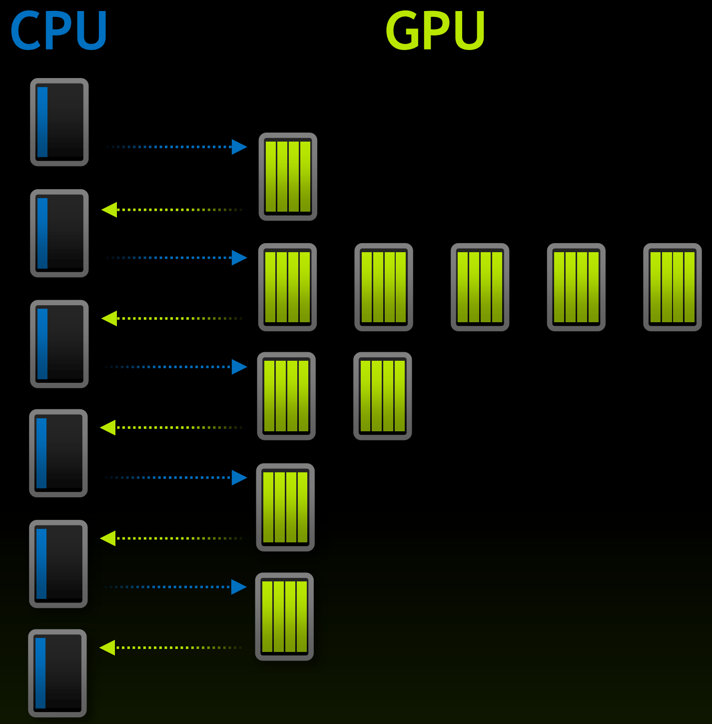
- Dynamically
- Simultaneously
- Independently



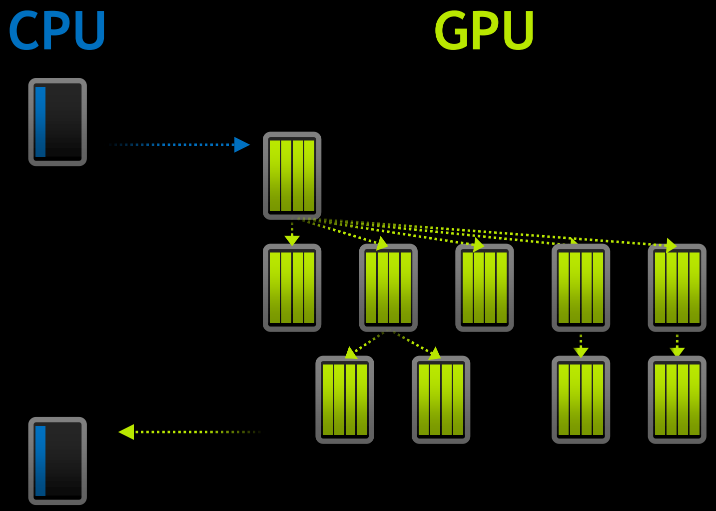
Fermi: Only CPU can generate GPU work

Kepler: GPU can generate work for itself

What Does It Mean?



GPU as Co-Processor



Autonomous, Dynamic Parallelism

Familiar Syntax

```
void main() {
    float *data;
    generate(data);

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

    manage(data);
}
```

CUDA from CPU

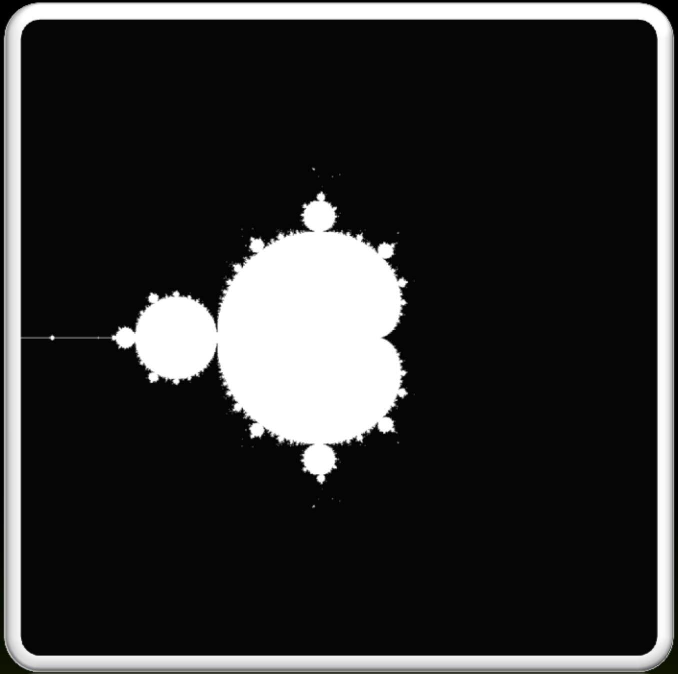
```
__global__ void B(float *data)
{
    generate(data);

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

    manage(data);
}
```

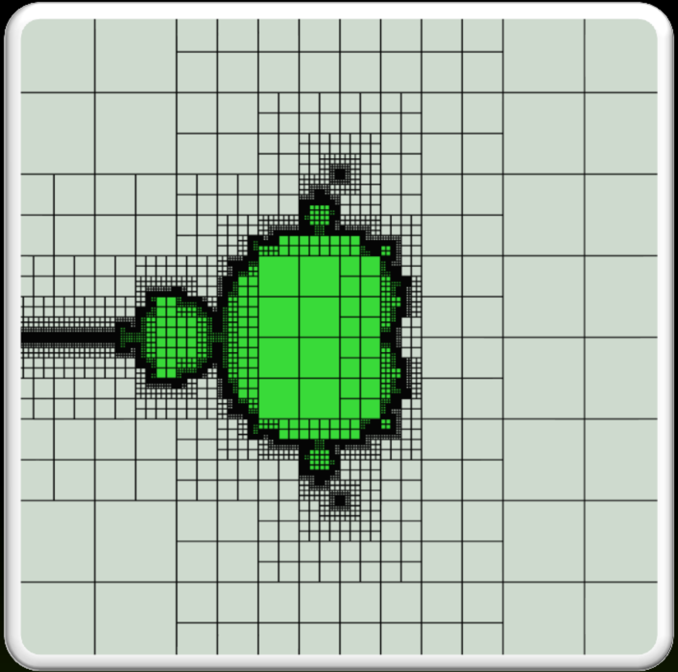
CUDA from GPU

Data-Dependent Parallelism



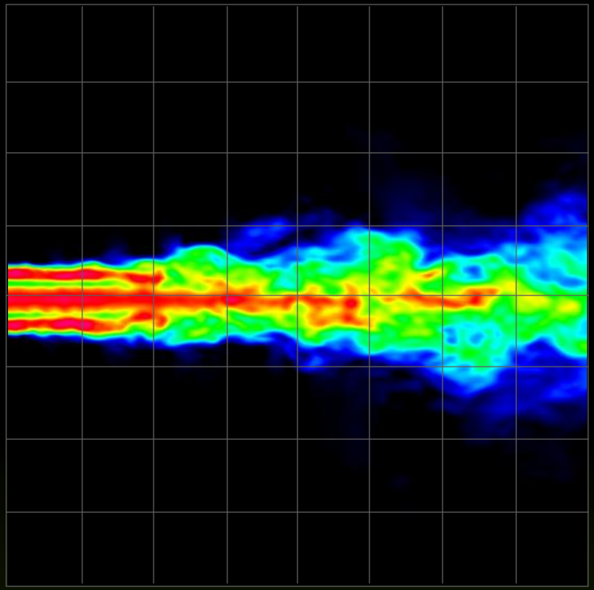
CUDA Today

Computational Power allocated to regions of interest



CUDA on Kepler

Dynamic Work Generation

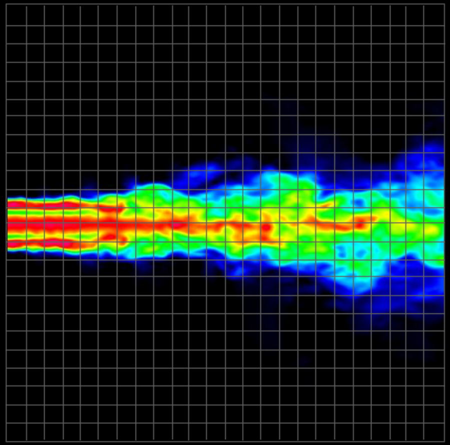


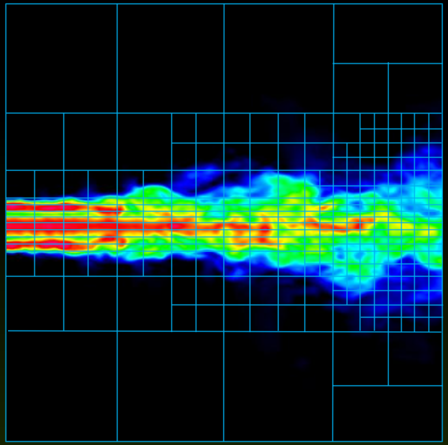
Initial Grid

Statically assign conservative worst-case grid

Dynamically assign performance where accuracy is required

Fixed Grid





Dynamic Grid

Bonsai GPU Tree-Code

Journal of Computational Physics, 231:2825-2839, April 2012

- Jeroen Bédorf, Simon Portegies Zwart
  - Leiden Observatory, The Netherlands
- Evghenii Gaburov
  - CIERA @ Northwestern U.
  - SARA, The Netherlands

- Galaxies generated with: Galatics
  Widrow L. M., Dubinksi J., 2005,
  Astrophysical Journal, 631 838