CUDA PLATFORM : WHAT'S NOW AND NEXT

Pradeep Gupta,
Manager-Developer Technology, Asia South

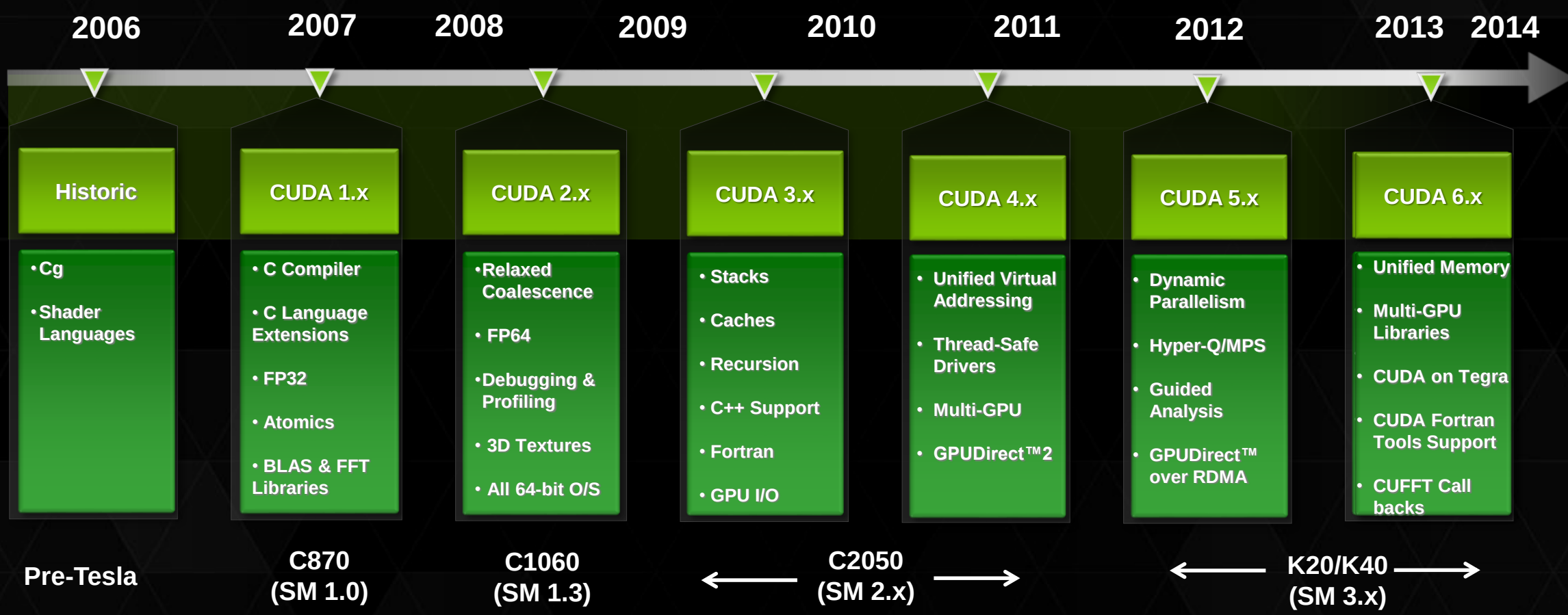
AGENDA

1. Introduction
2. CUDA 6.0/6.5 Updates
3. Jetson Platform
4. What's next

AGENDA

1. Introduction
2. CUDA 6.0/6.5 Updates
3. Jetson Platform
4. What's next

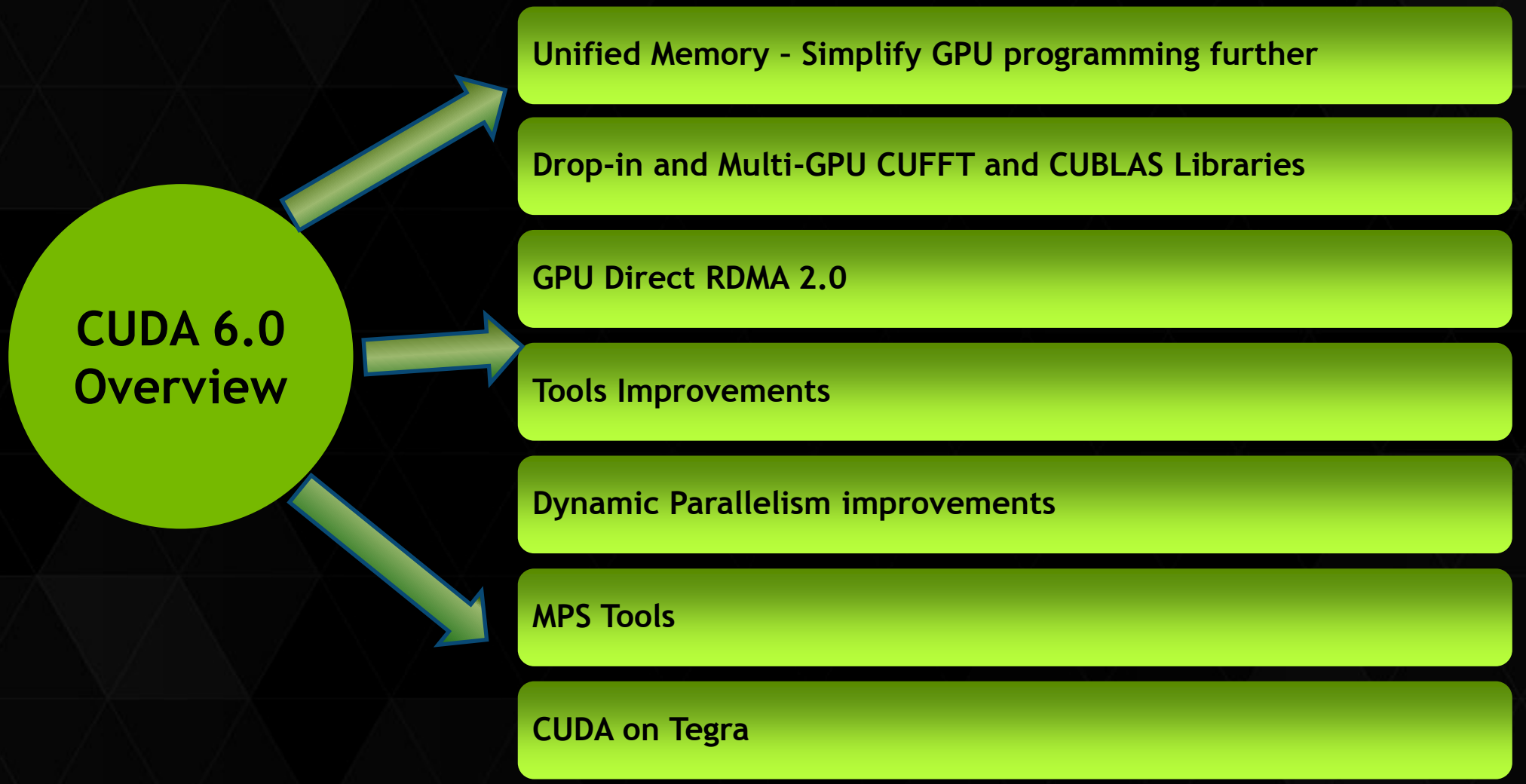
CUDA EVOLUTION



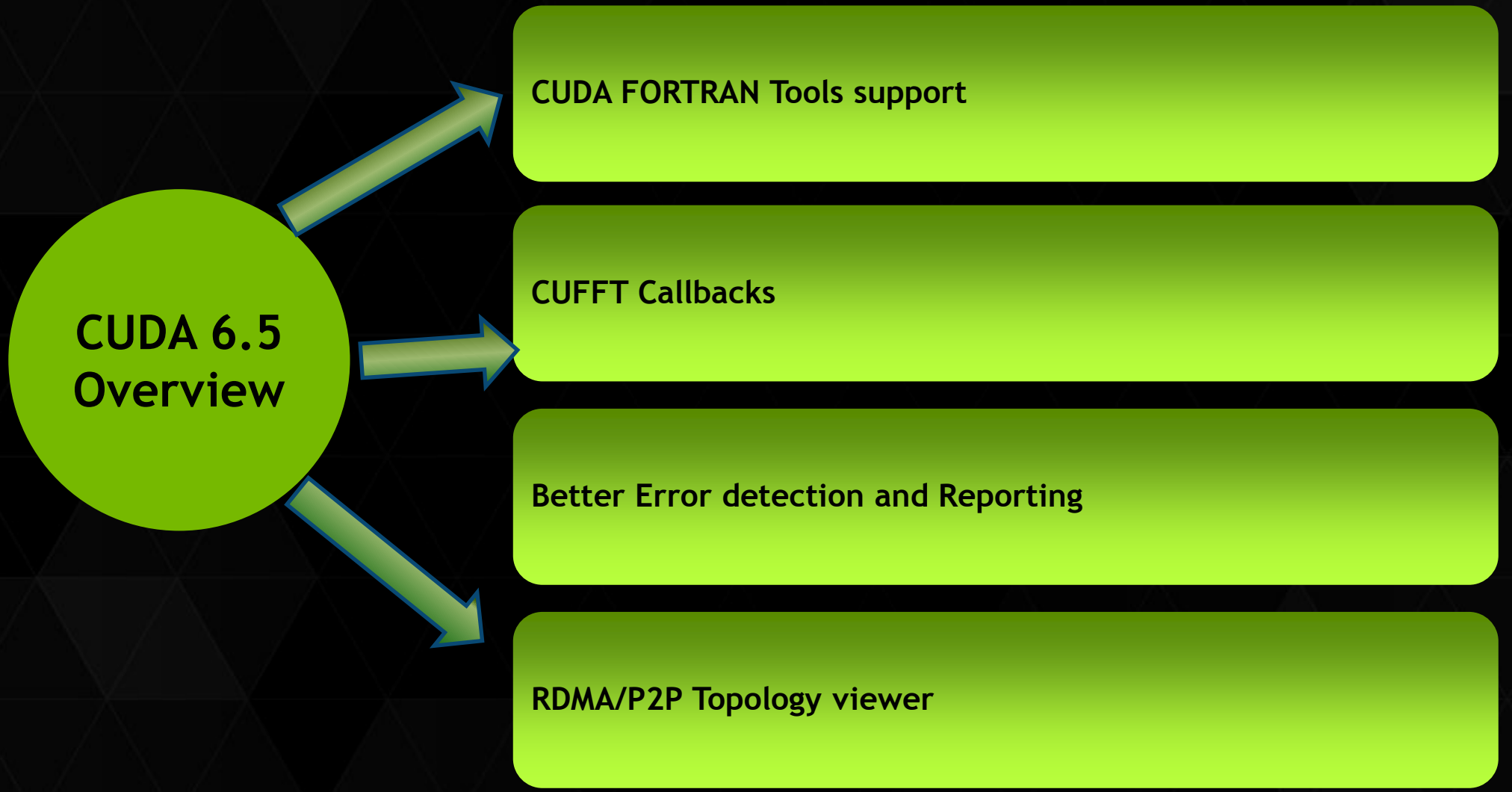
AGENDA

1. Introduction
2. CUDA 6.0/6.5 Updates
3. Jetson Platform
4. What's next

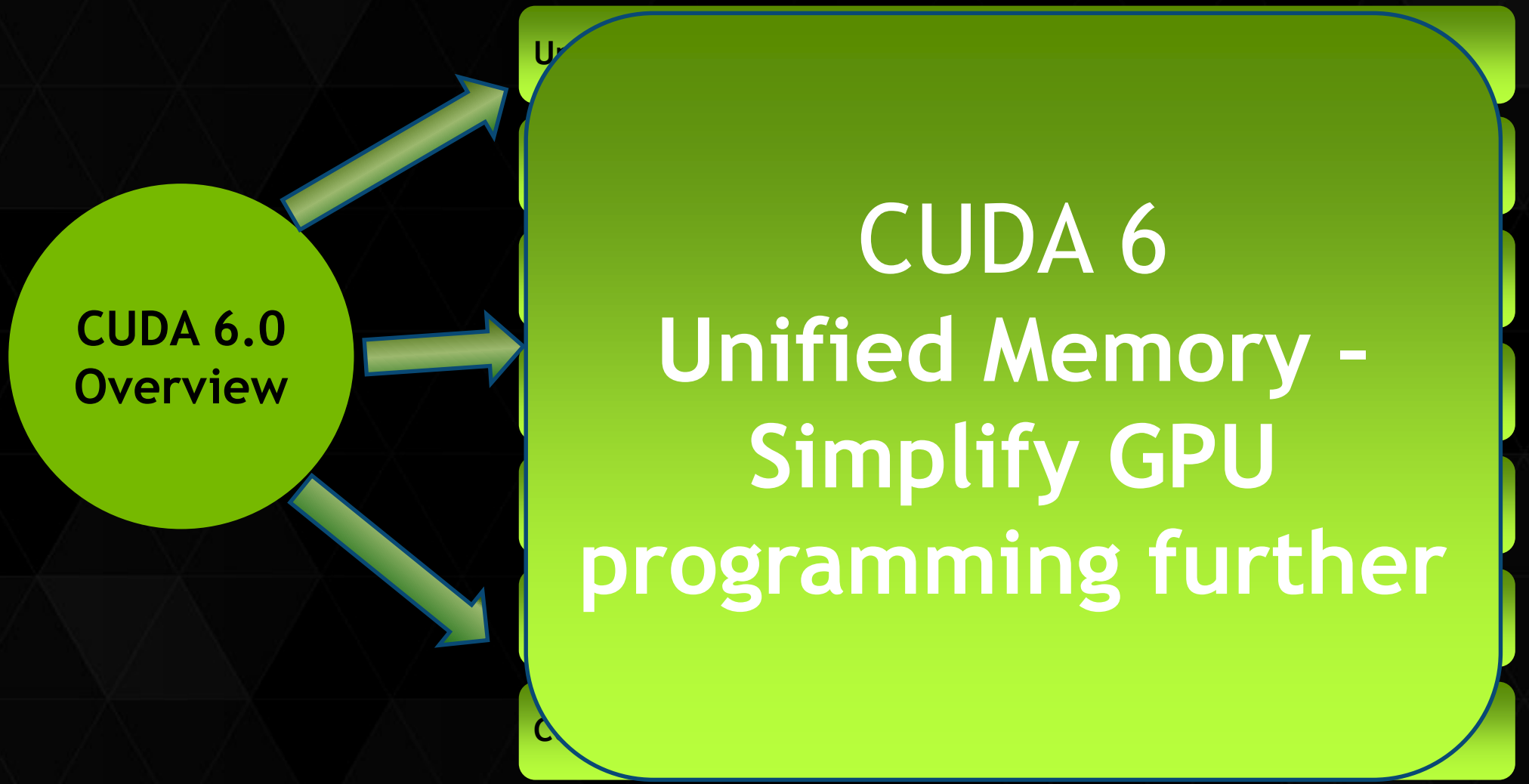
OVERVIEW OF CUDA 6 FEATURES



OVERVIEW OF CUDA 6.5 FEATURES

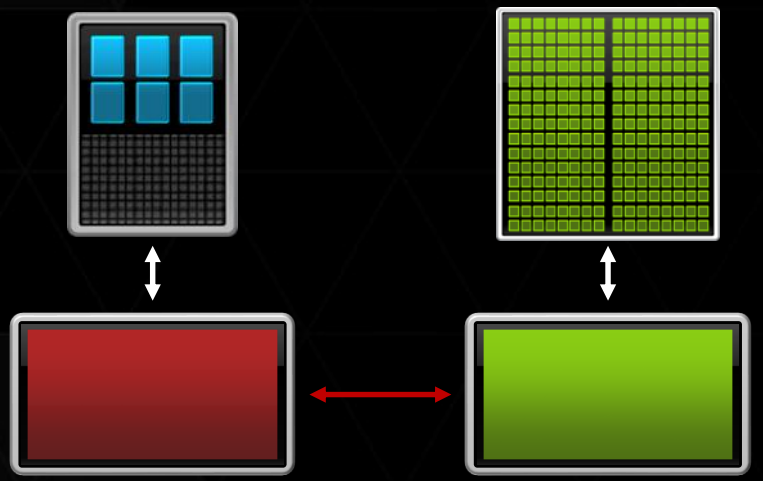


OVERVIEW OF CUDA 6 FEATURES

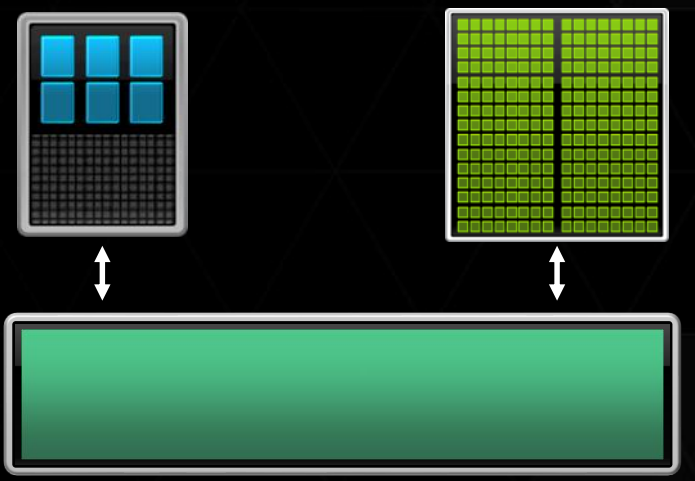


UNIFIED MEMORY
DRAMATICALLY LOWER DEVELOPER EFFORT

Developer View Today



Developer View With Unified Memory



SYSTEM REQUIREMENTS

GPU	Kepler	(GK10x+ , i.e. SM3.0+)
Operating System	64-bit required	
Linux	Kernel 2.6.18+	(all CUDA-supported distros, not ARM)
Windows	Win7 or Win8	(WDDM & TCC no XP/Vista)

SUPER SIMPLIFIED MEMORY MANAGEMENT CODE

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

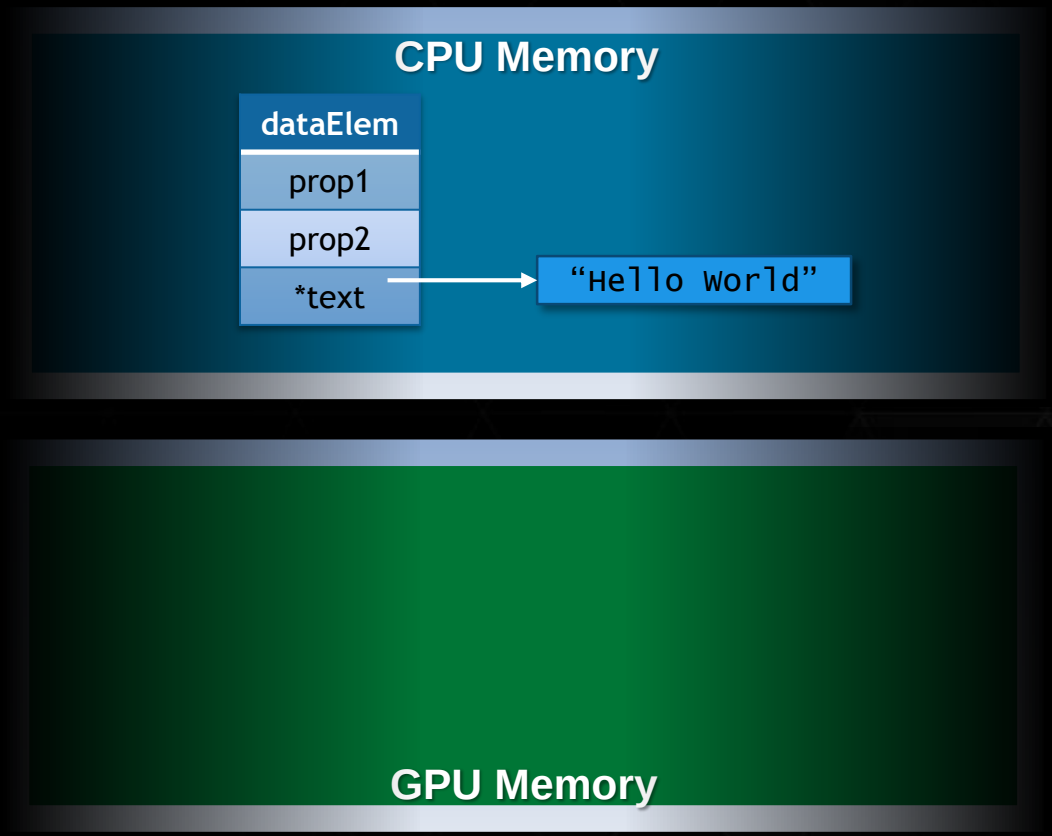
```
void sortfile(FILE *fp, int N) {
  char *data;
  cudaMallocManaged(&data, N);

  fread(data, 1, N, fp);

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

  use_data(data);

  cudaFree(data);
}
```

UNIFIED MEMORY DELIVERS

1. Simpler Programming & Memory Model
   - Single pointer to data, accessible anywhere
   - Tight language integration
   - Greatly simplifies code porting
2. Performance Through Data Locality
   - Migrate data to accessing processor
   - Guarantee global coherency
   - Still allows *cudaMemcpyAsync()* hand tuning

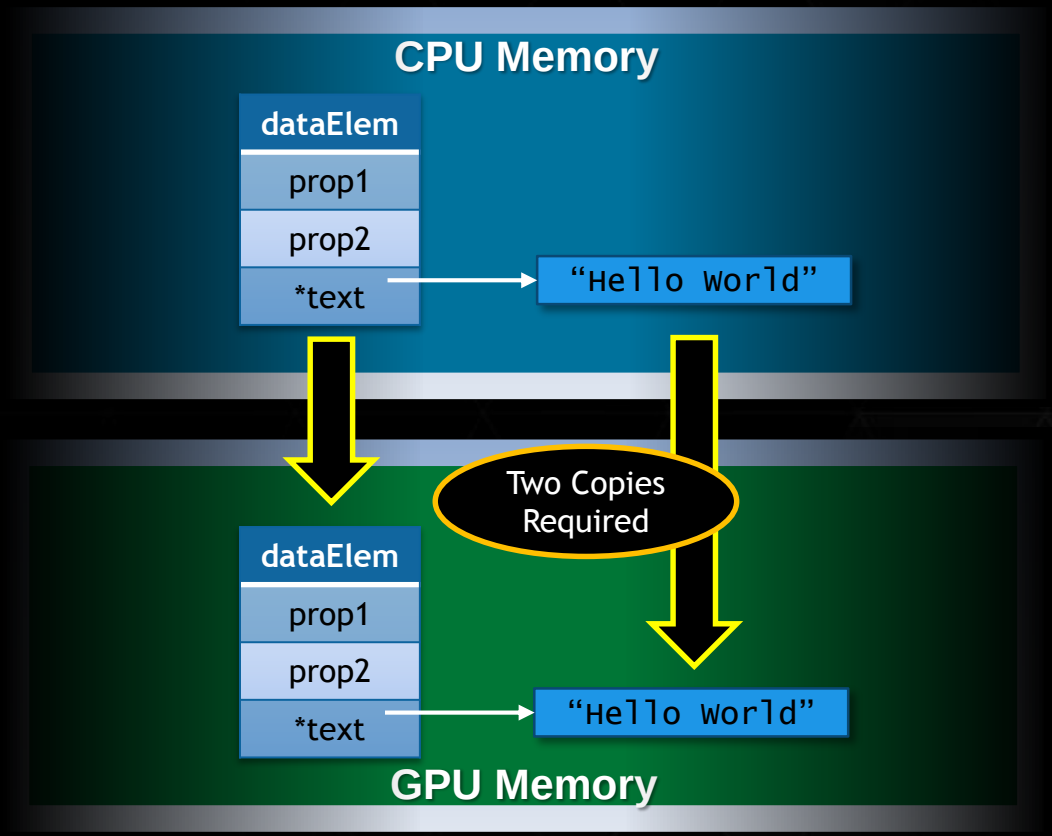
SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

```
struct dataElem
{
    int prop1;
    int prop2;
    char *text;
};
```



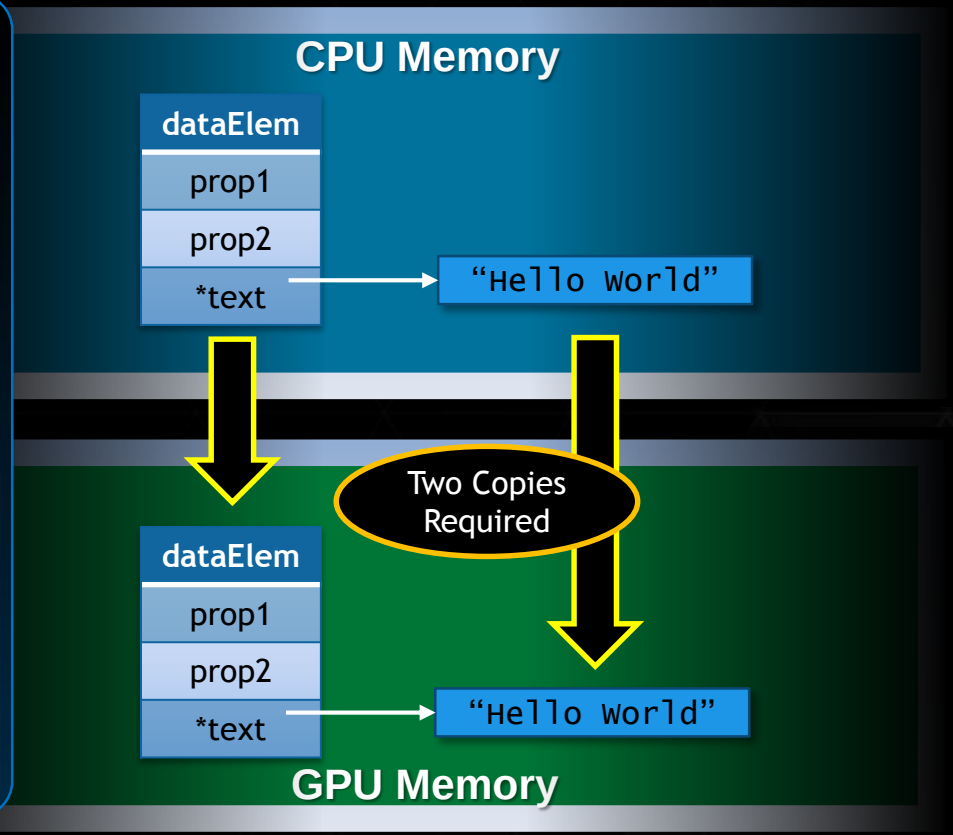
Simpler Memory Model: Eliminate Deep Copies

```
struct dataElem
{
    int prop1;
    int prop2;
    char *text;
};
```



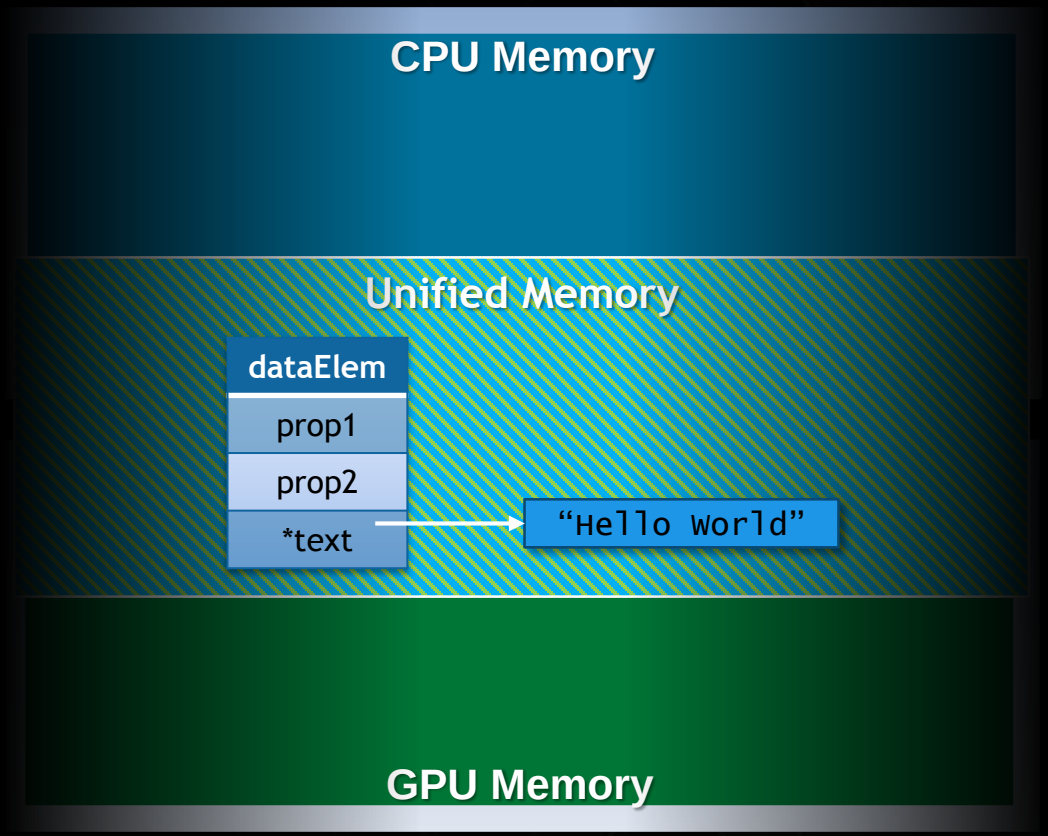
SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

```
void launch(dataElem *elem) {
    dataElem *g_elem;
    char *g_text;

    int textlen = strlen(elem->text);

    // Allocate storage for struct and text
    cudaMalloc(&g_elem, sizeof(dataElem));
    cudaMalloc(&g_text, textlen);

    // Copy up each piece separately, including
    // new "text" pointer value
    cudaMemcpy(g_elem, elem, sizeof(dataElem));
    cudaMemcpy(g_text, elem->text, textlen);
    cudaMemcpy(&(g_elem->text), &g_text,
                                sizeof(g_text));

    // Finally we can launch our kernel, but
    // CPU & GPU use different copies of "elem"
    kernel<<< ... >>>(g_elem);
}
```



SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

```
void launch(dataElem *elem) {
    kernel<<< ... >>>(elem);
}
```



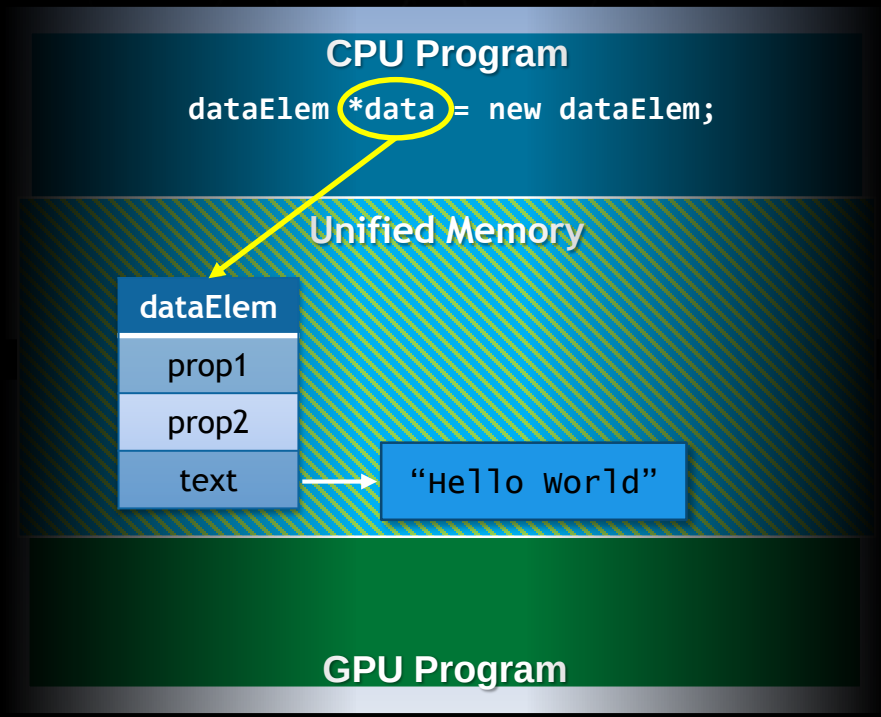
UNIFIED MEMORY WITH C++
A POWERFUL COMBINATION

C++ objects migrate easily when allocated on managed heap

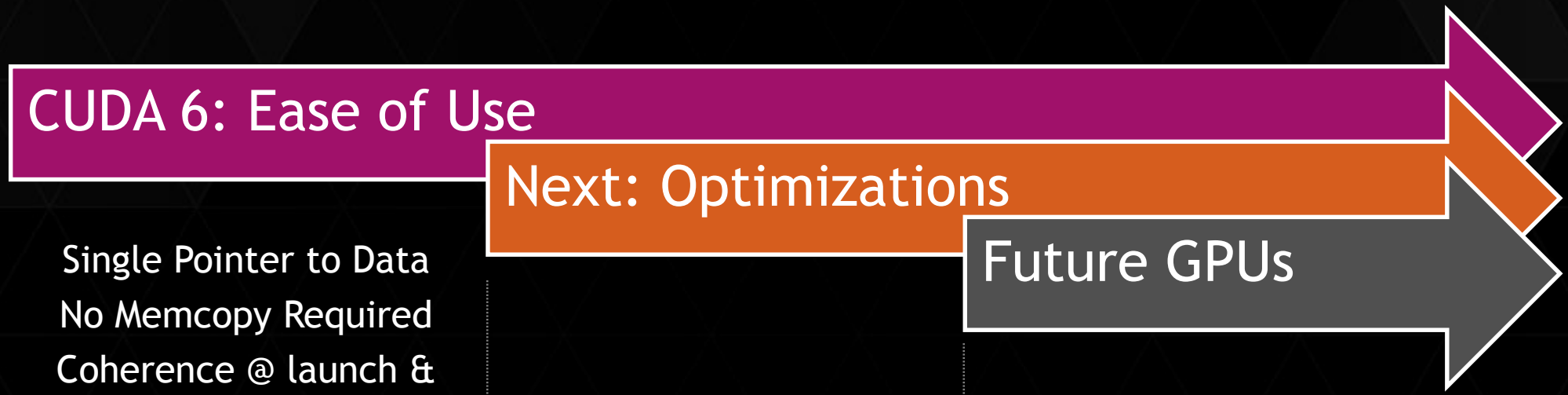
- Overload *new* operator to use C++ in unified memory region
- Deep copies, pass-by-value, pass-by-reference: JUST WORKS

```
class Managed {
    void *operator new(size_t len) {
        void *ptr;
        cudaMallocManaged(&ptr, len);
        return ptr;
    }

    void operator delete(void *ptr) {
        cudaFree(ptr);
    }
};
```

```
// Inherit from "Managed",
// C++ now handles our deep copies
class dataElem : public Managed {
        int prop1;
        int prop2;
        String text;
};
```

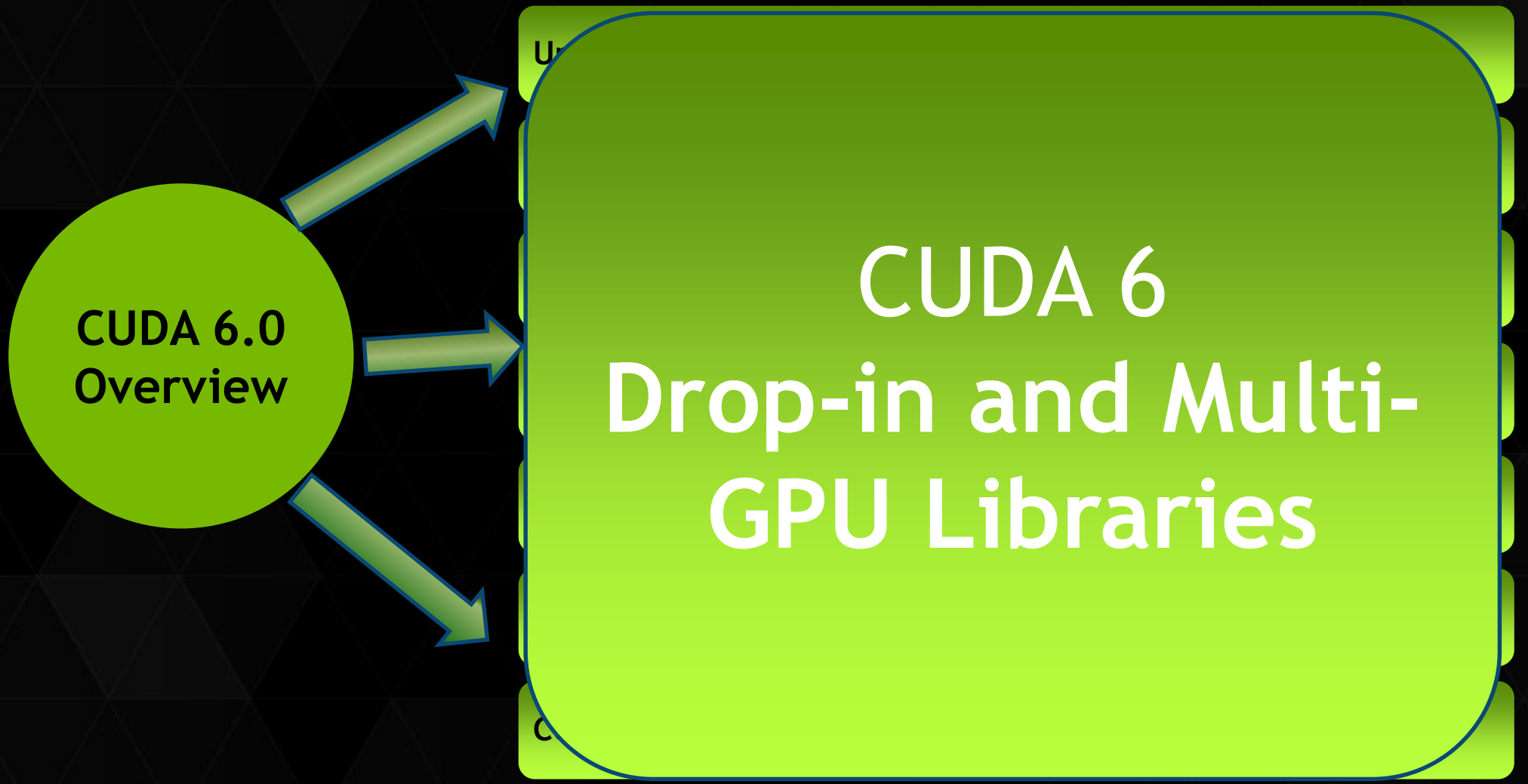


UNIFIED MEMORY ROADMAP



PARALLEL FORALL Learn More: http://bit.ly/um-p4a

OVERVIEW OF CUDA 6 FEATURES

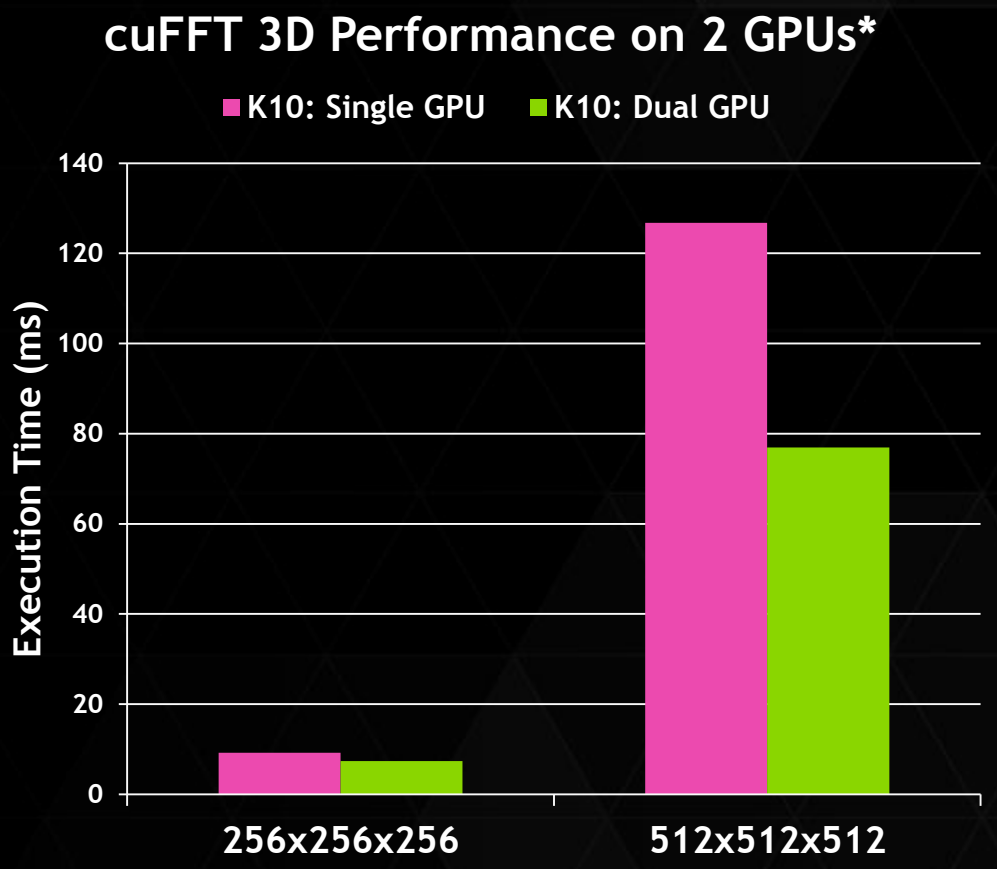


Extended (XT) Library Interfaces

- Automatic Scaling to >1 GPU per node
- cuFFT and cuBLAS level 3
- Out-of-core operations: e.g. very large GEMM
- BLAS 3 Host Interfaces: automatically overlaps memory transfers

MULTI-GPU CUFFT

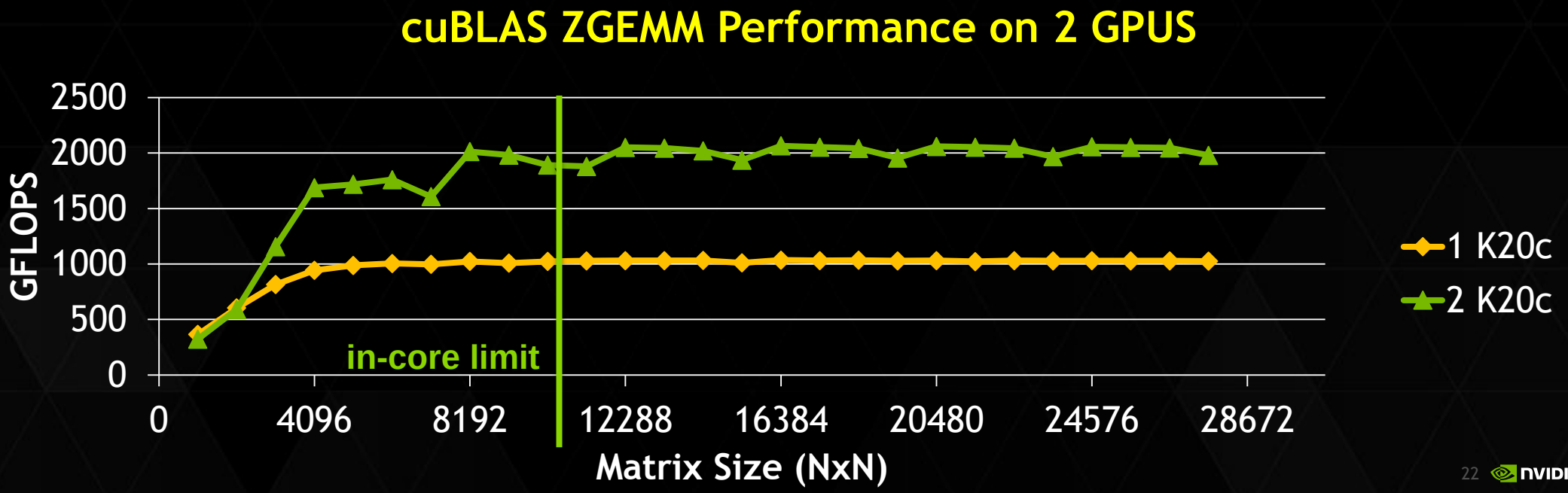
- Single & Batch Transforms across multiple GPUs (max 2 in CUDA 6)
- Tuned for multi-GPU cards (K10)
  - Better scaling for larger transforms



*Does not include memcpy time

MULTI-GPU CUBLAS

- Single function call automatically spreads work across two GPUs
- Source and result data in system memory
- Supports matrices > size of memory (out-of-core)
- All BLAS Level-3 routines



NEW DROP-IN NVBLAS LIBRARY

- Drop-in replacement for CPU-only BLAS
  - Automatically routes standard BLAS3 calls to cuBLAS
  - Optionally configure which routines and matrix sizes are accelerated
  - User provides CPU-only BLAS dynamic library location
- Simply re-link or change library load order

```
gcc myapp.c –lnvblas -lmkl_rt -o myapp
```

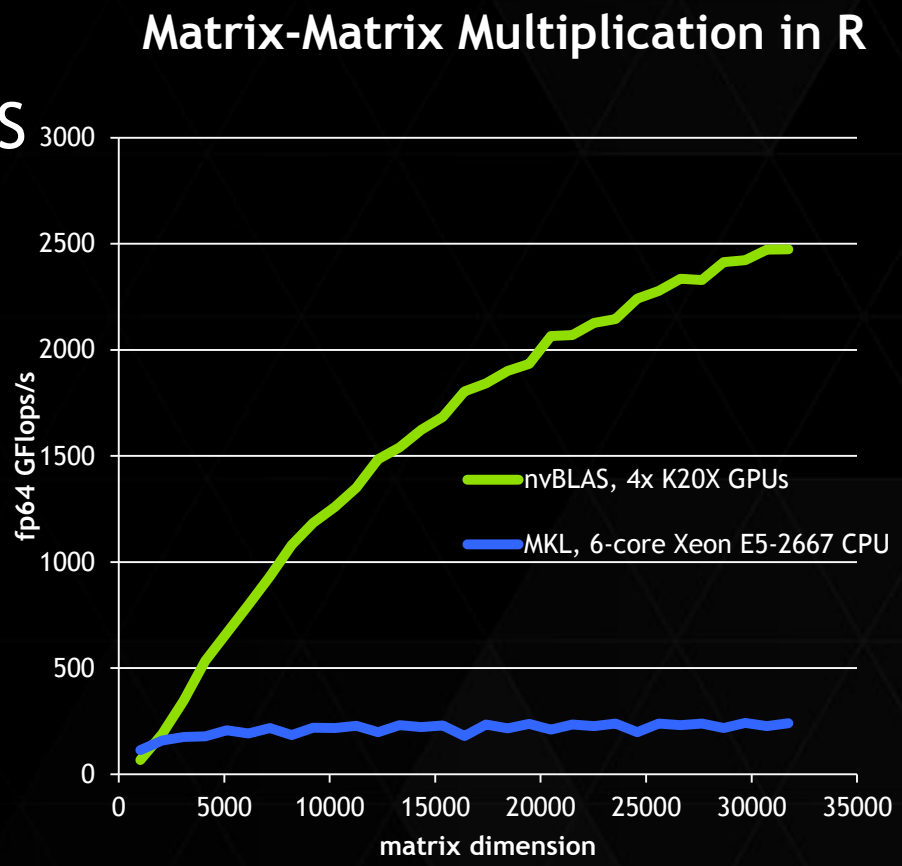
- or -

```
env LD_PRELOAD=libnvblas.so myapp
```

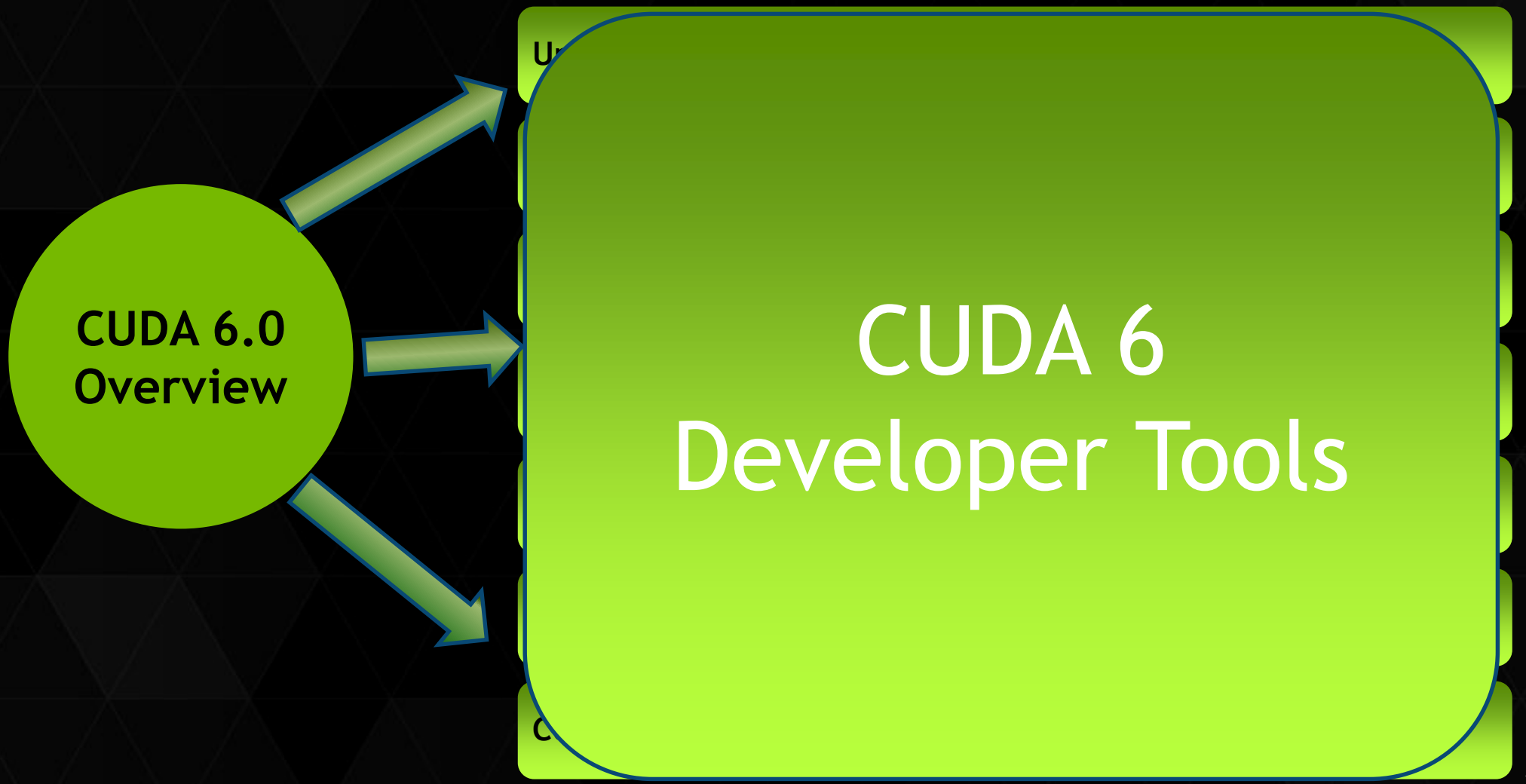
NEW DROP-IN NVBLAS LIBRARY

- Drop-in replacement for CPU-only BLAS
- Automatically route BLAS3 calls to cuBLAS
- Example: Drop-in Speedup for R

```
> LD_PRELOAD=/usr/local/cuda/lib64/libnvblas.so R

> A <- matrix(rnorm(4096*4096), nrow=4096, ncol=4096)
> B <- matrix(rnorm(4096*4096), nrow=4096, ncol=4096)
> system.time(C <- A %*% B)

   user  system elapsed

  0.348   0.142   0.289
```

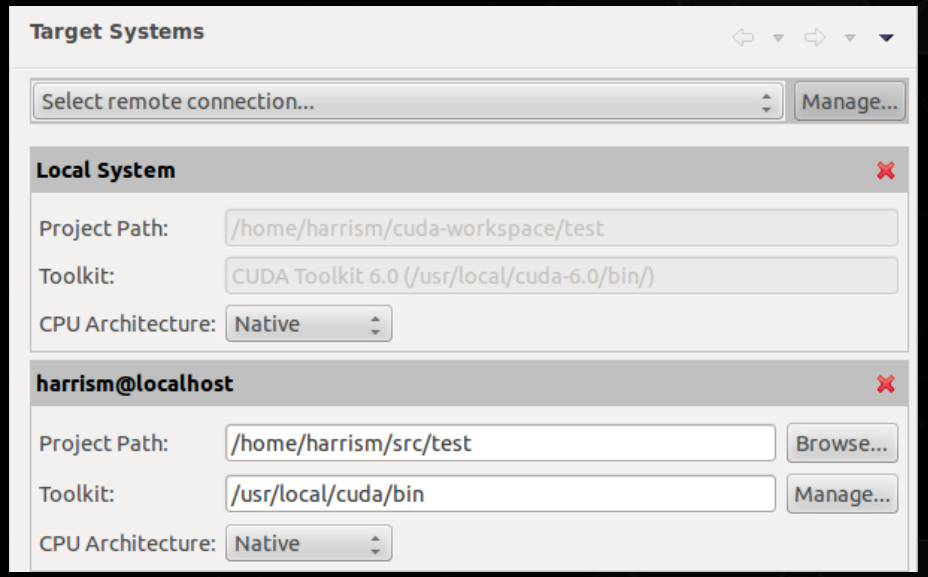


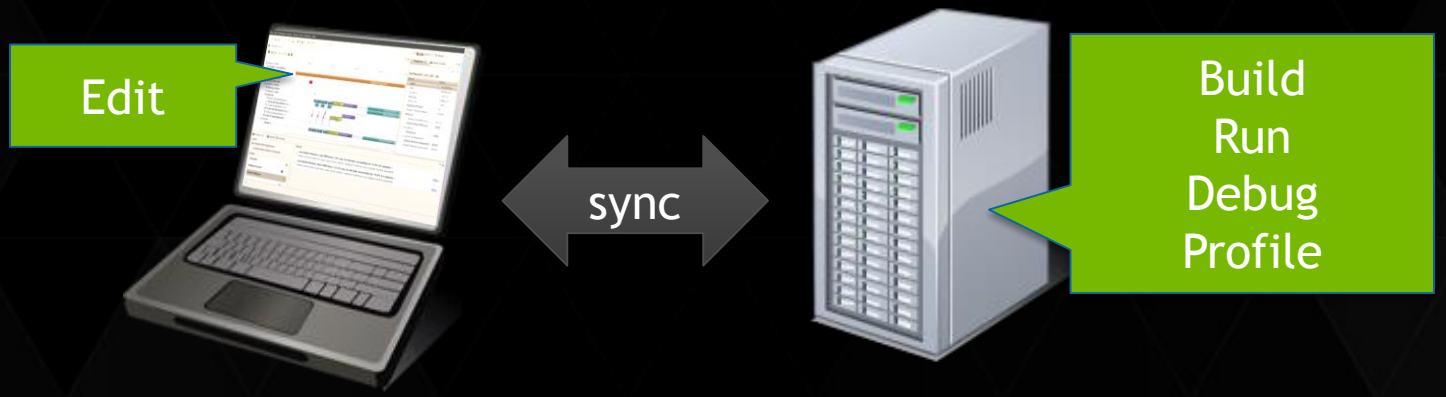
OVERVIEW OF CUDA 6 FEATURES



REMOTE DEVELOPMENT WITH NSIGHT ECLIPSE EDITION

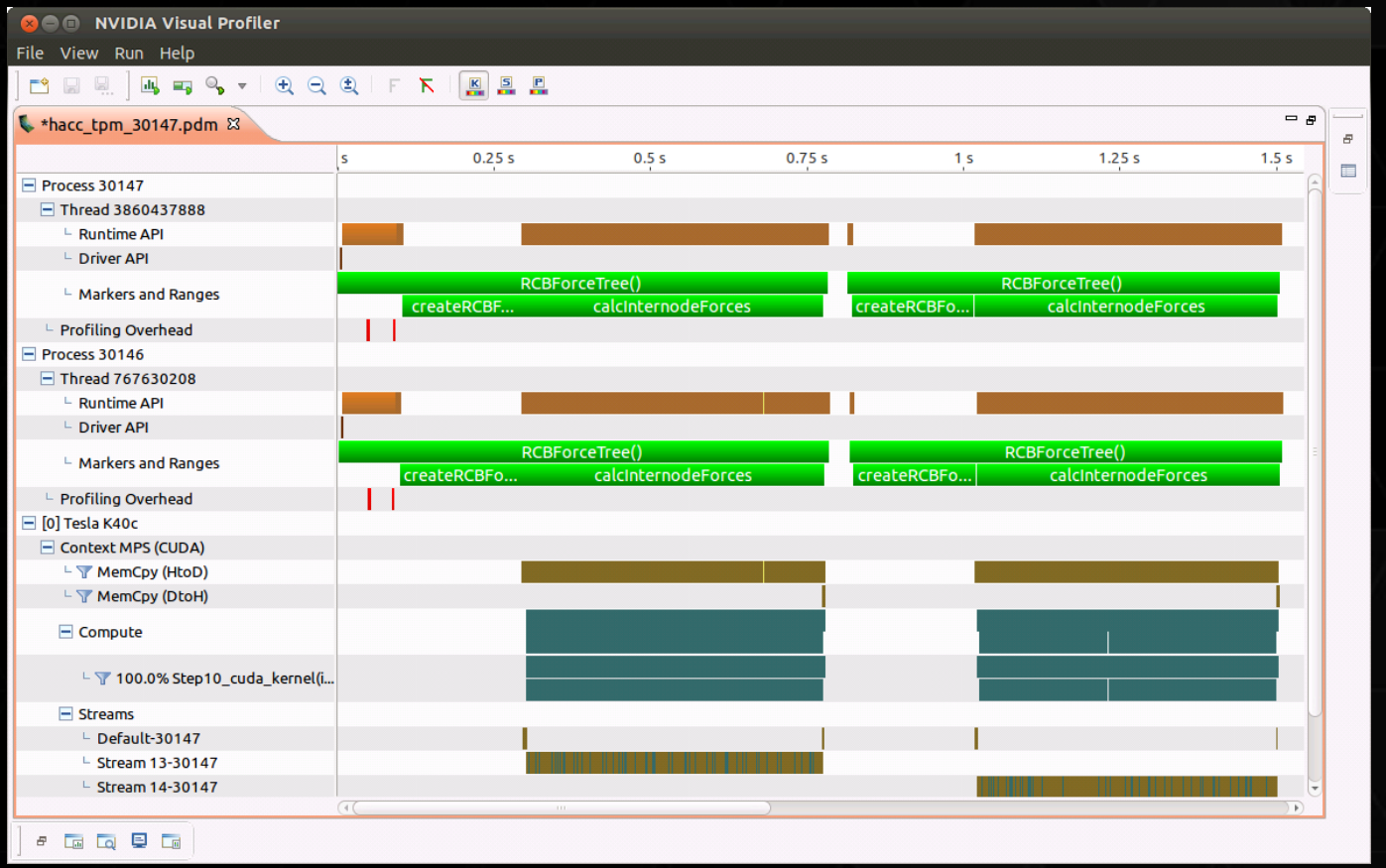
- Local IDE, remote application
  - Edit locally, build & run remotely
  - Automatic sync via ssh
  - Cross-compilation to ARM
- Full debugging & profiling via remote connection





CUDA TOOLS FOR MPS (MULTI-PROCESS SERVER)

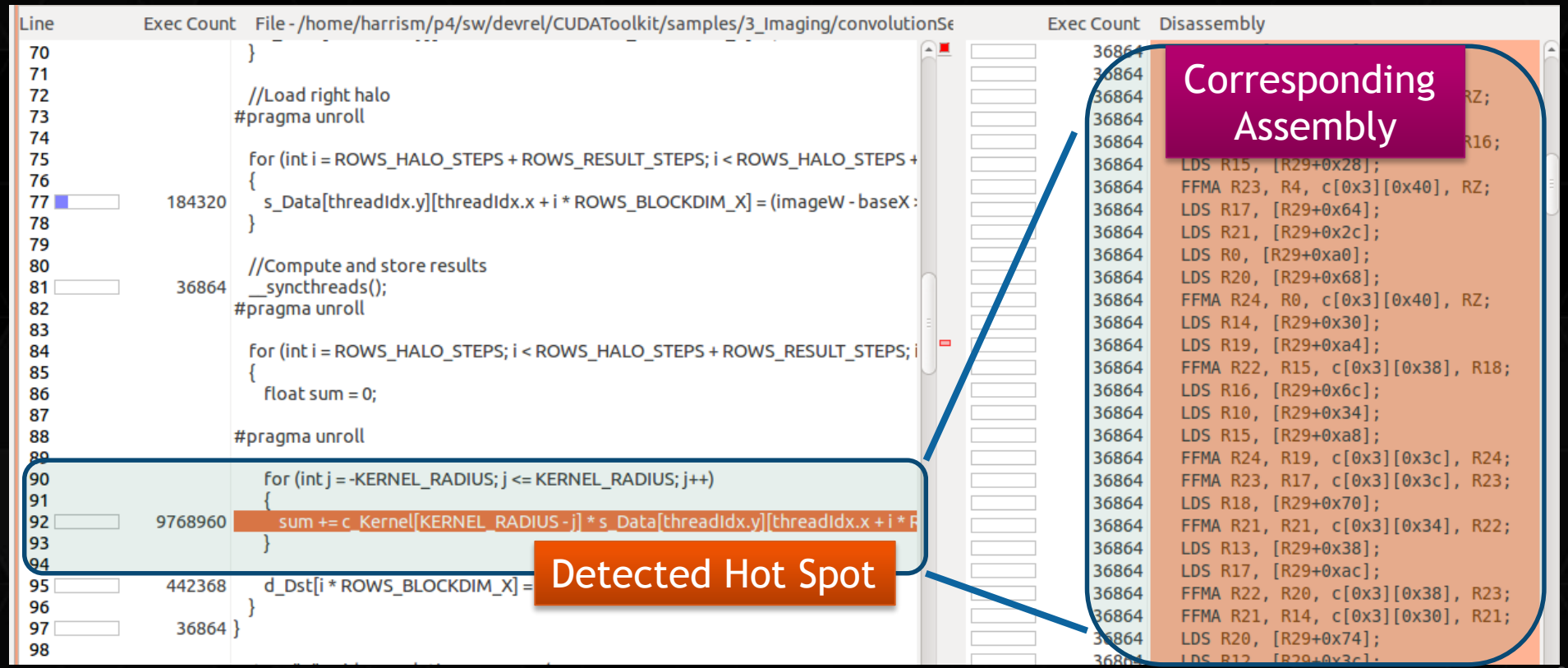
- Profile MPI apps on MPS using nvprof
- Import multi-process MPI ranks into Visual Profiler
- Run CUDA-MEMCHECK on apps running on MPS



DETAILED KERNEL PROFILING

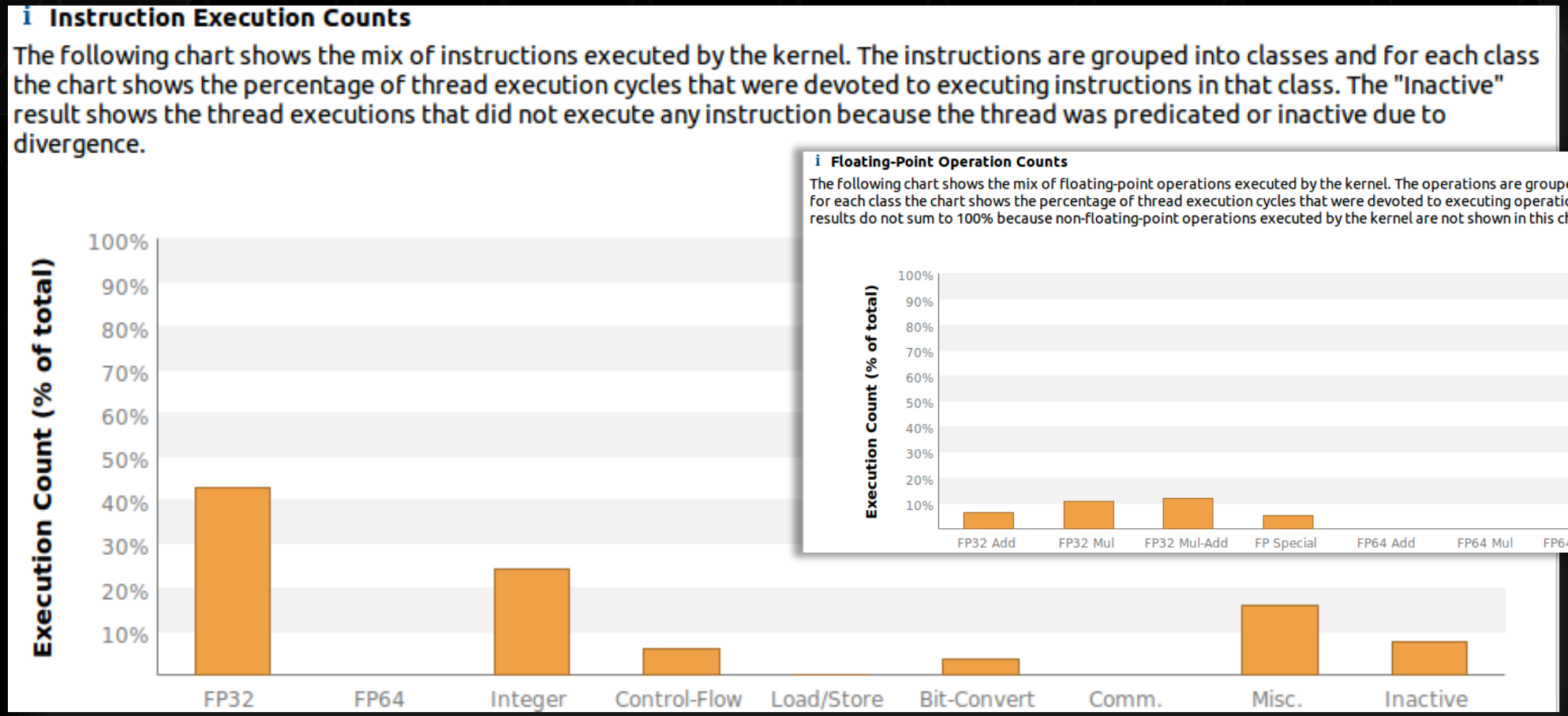
VISUAL PROFILER AND NSIGHT EE

Instruction counts automatically locate hot spots in your code



DETAILED INSTRUCTION MIX VISUALIZATION

VISUAL PROFILER AND NSIGHT EE



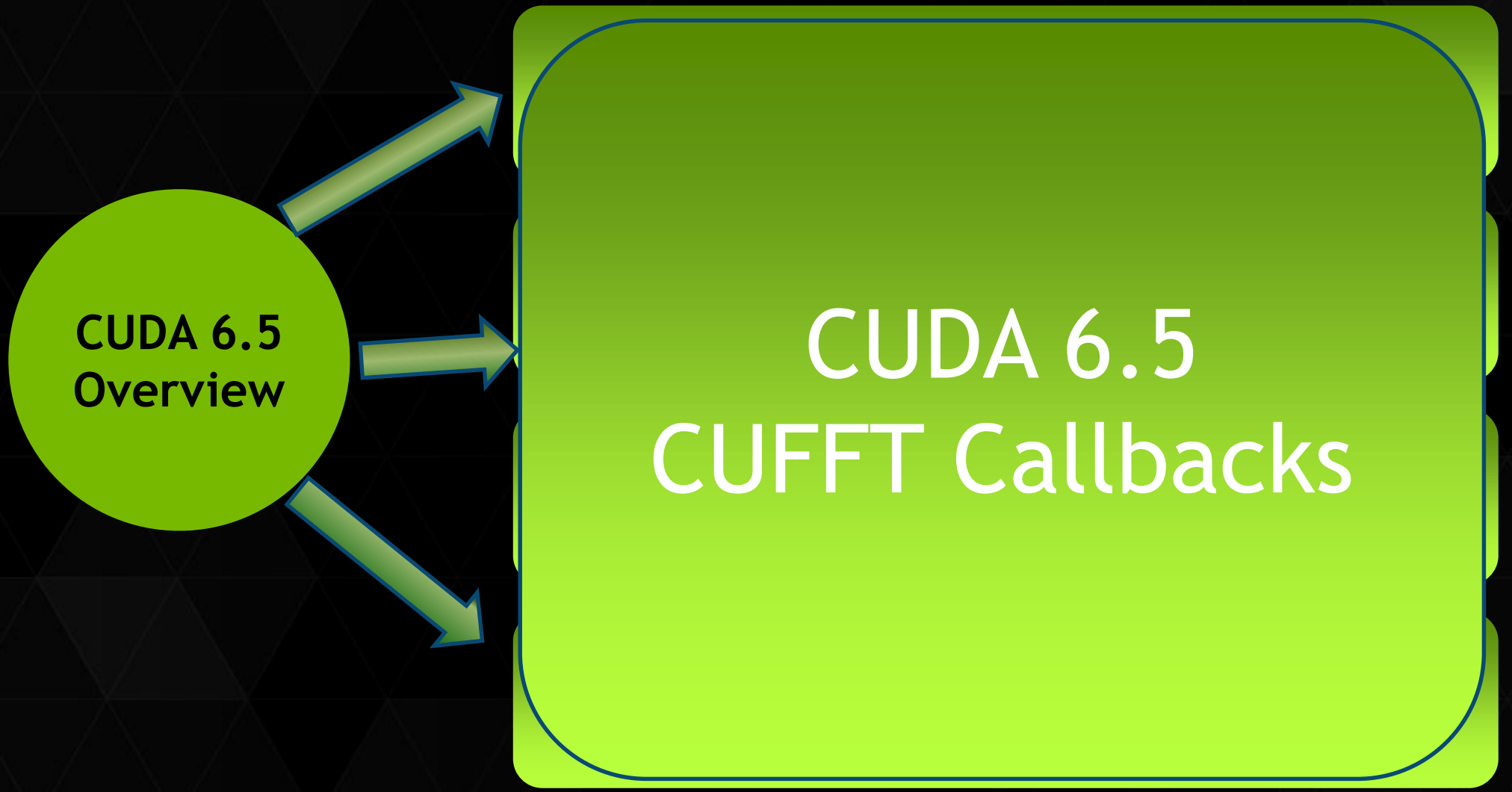
CUDA 6

Dramatically Simplifies Parallel Programming with Unified Memory

Sign up for CUDA Registered Developer Program

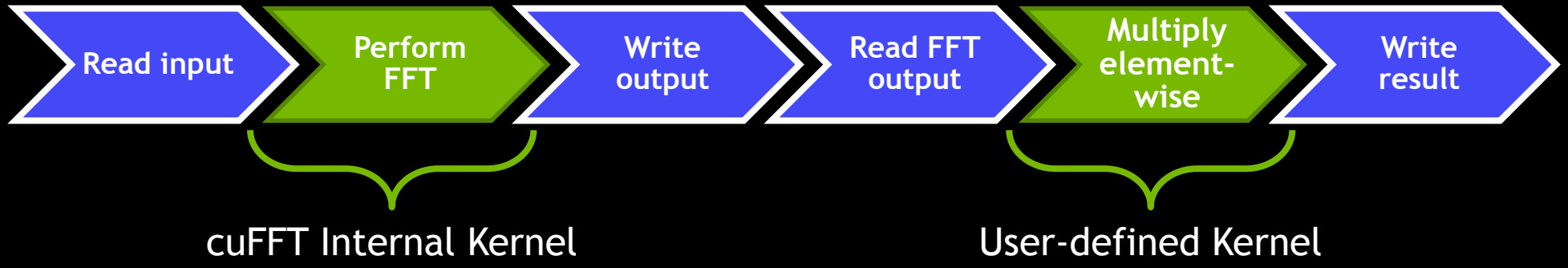
https://developer.nvidia.com/cuda-toolkit

OVERVIEW OF CUDA 6.5 FEATURES

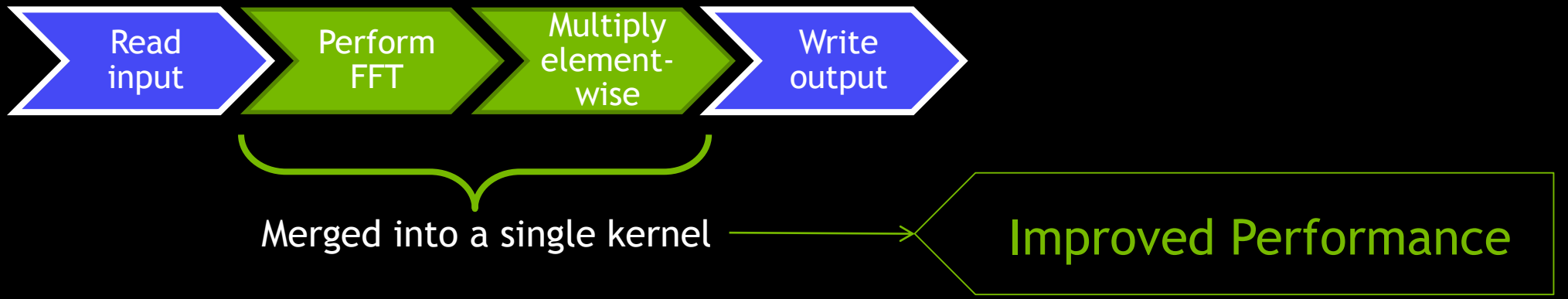


CUDA 6.5 : cuFFT User-Defined Callbacks

Today: 2 kernels, 2 memory roundtrips



With CUDA 6.5: 1 kernel, 1 memory roundtrip



AGENDA

1 Introduction

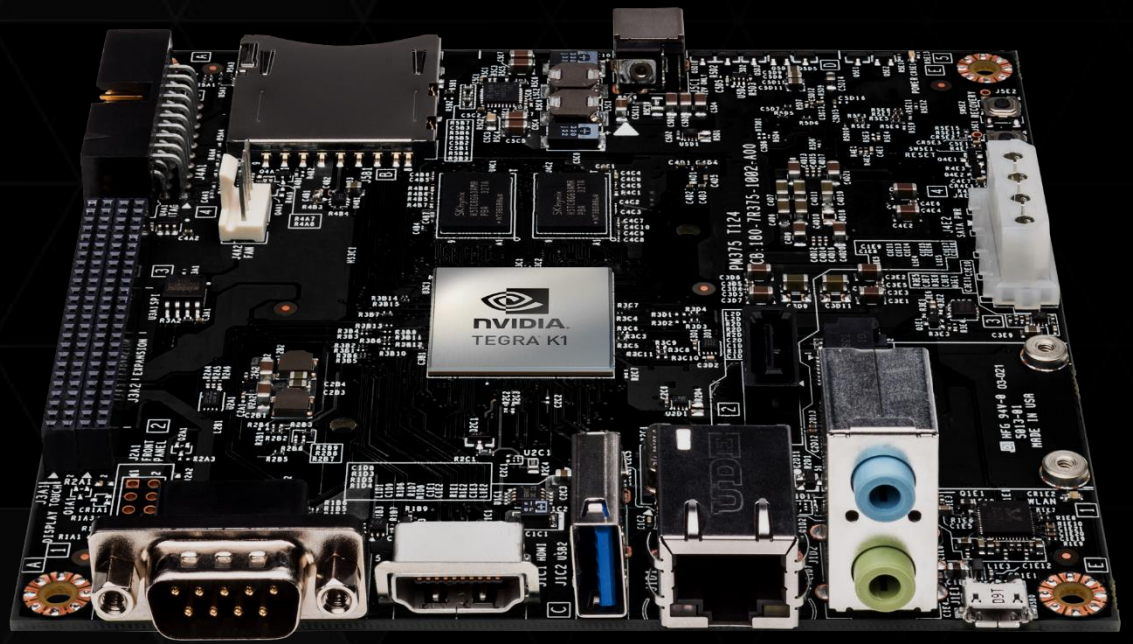
2 CUDA 6.0 Updates

3 Jetson Platform

4 What's Next

JETSON TK1

THE WORLD'S 1st EMBEDDED SUPERCOMPUTER

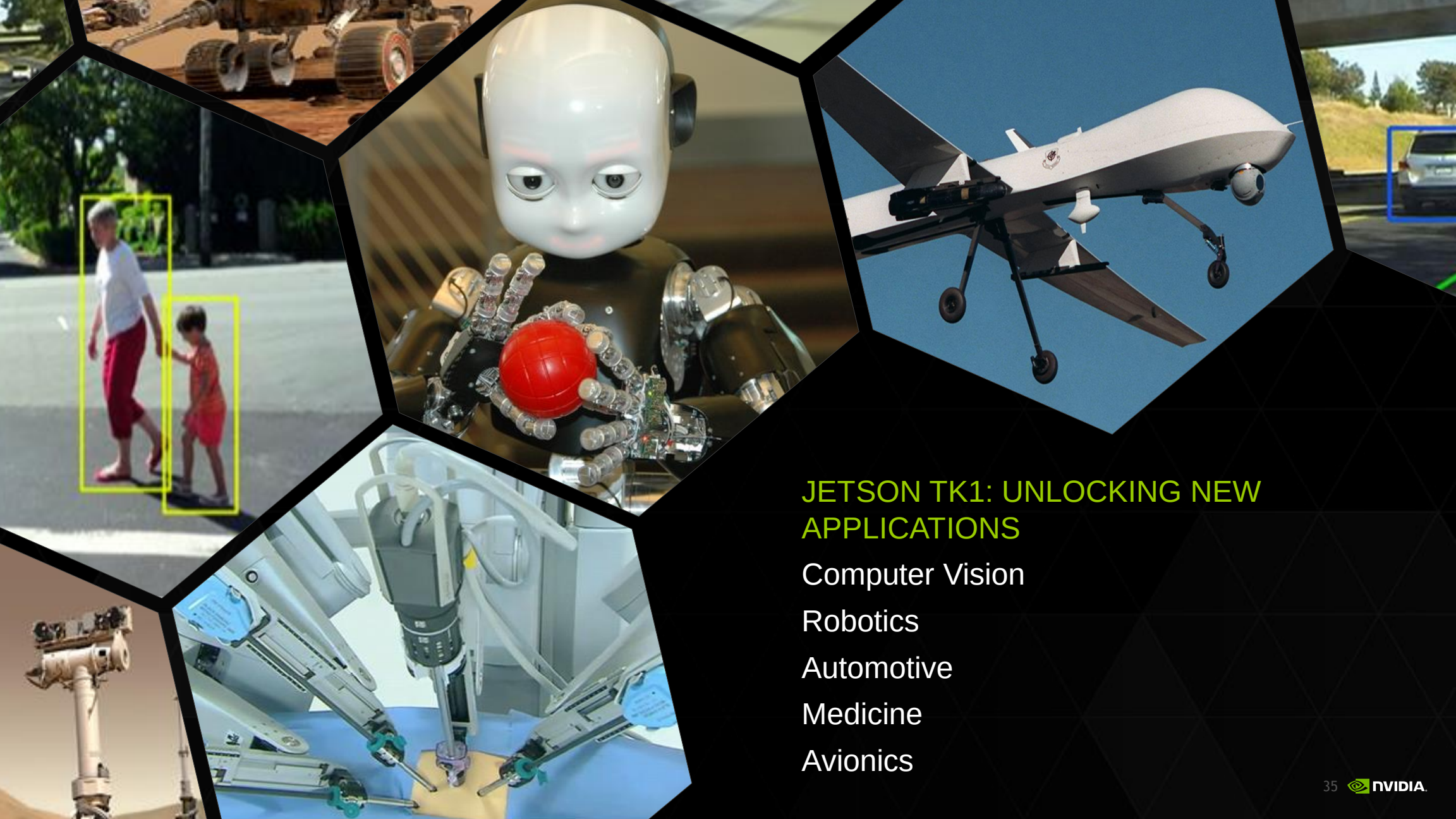


Development Platform for Embedded
Computer Vision, Robotics, Medical

Tegra K1 SoC

CUDA Enabled

$192 (preorder now)



JETSON TK1: UNLOCKING NEW APPLICATIONS

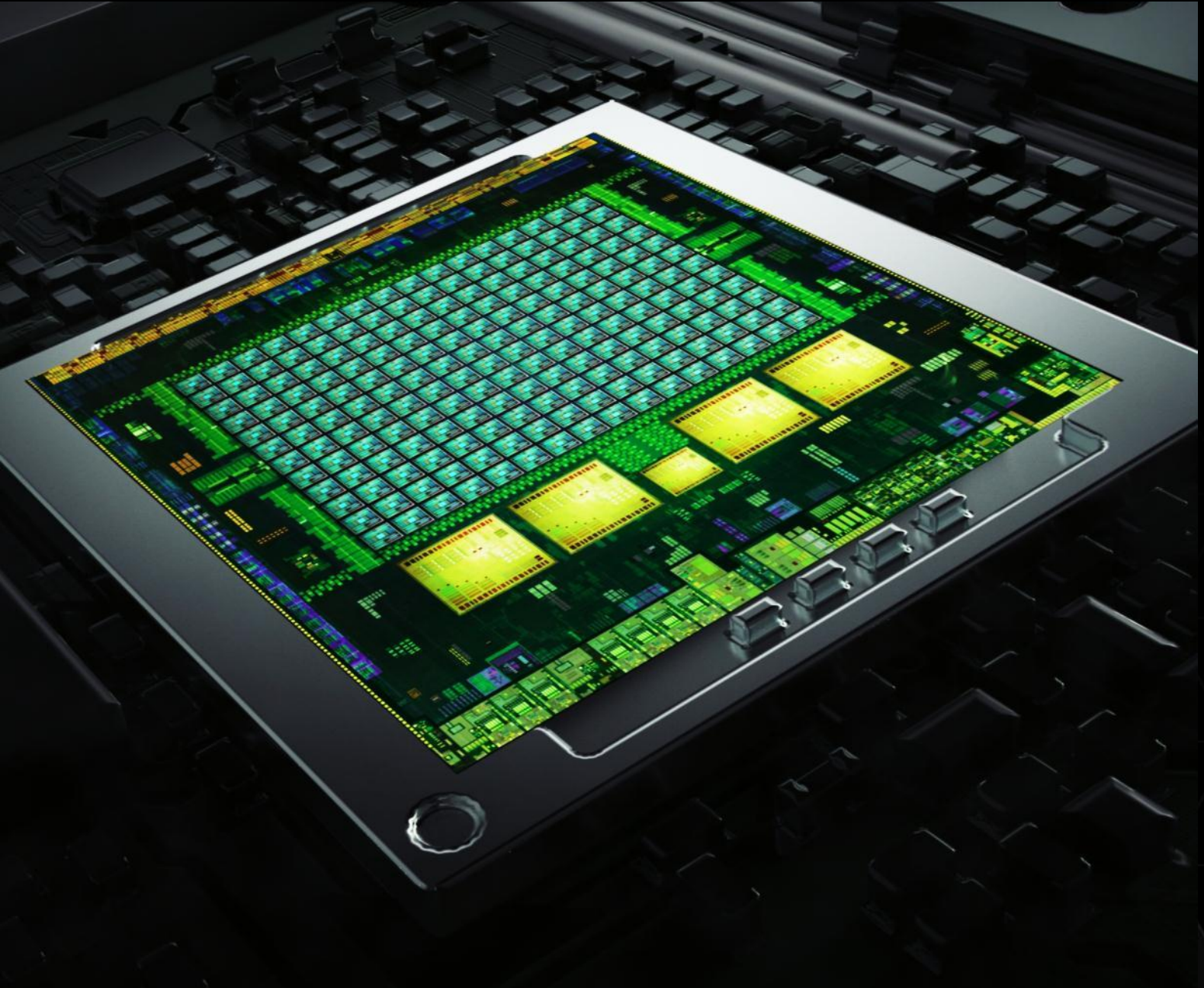
Computer Vision

Robotics

Automotive

Medicine

Avionics



TEGRA K1

IMPOSSIBLY ADVANCED

NVIDIA Kepler Architecture

4-Plus-1 Quad-Core A15

192 NVIDIA CUDA Cores

Compute Capability 3.2

326 GFLOPS

5 Watts

CUDA 6 FOR EMBEDDED APPLICATIONS

- Tegra K1 Supports Full CUDA Toolkit v6.0
- Unified Memory
  - Memory physically unified, separate GPU and CPU caches
  - Same programming model as desktop and server
- OpenGL 4.4, DirectX 11 and OpenGL ES 3.0
- Jetson TK1 runs 32-bit Ubuntu 13.04 Linux for Tegra (L4T)

AGENDA

1 Introduction

2 CUDA 6.0 Updates

3 Jetson Platform

4 What's Next

AND BEYOND…

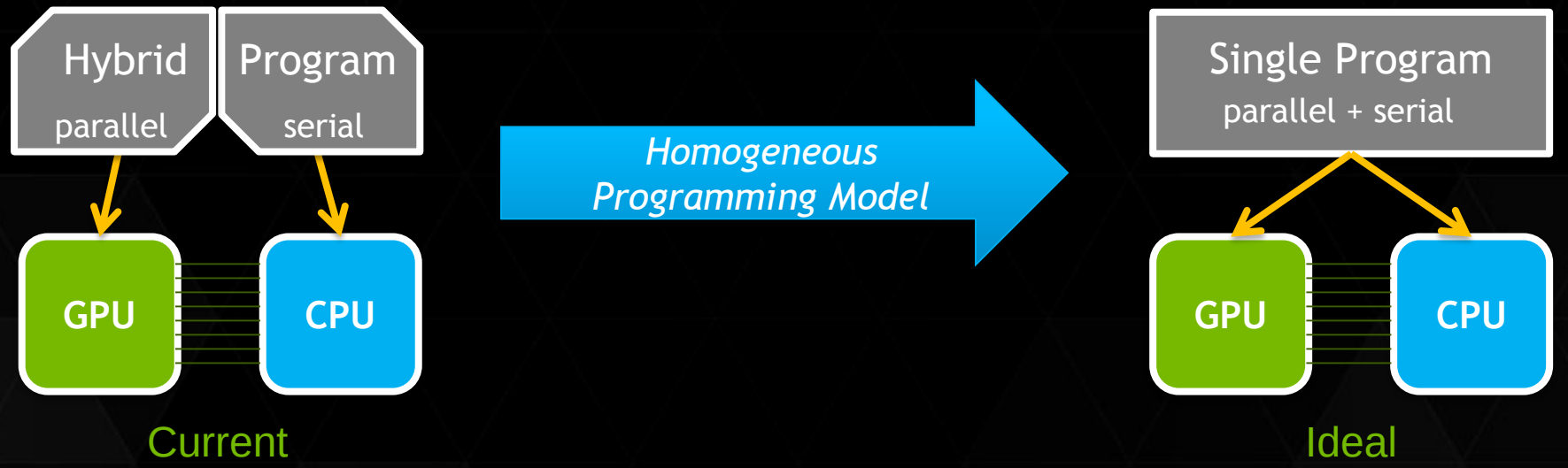
GOALS FOR THE CUDA PLATFORM

Simplicity	• Learn, adopt, & use parallelism with ease
Productivity	• Quickly achieve feature & performance goals
Portability	• Write code that can execute on all targets
Performance	• High absolute performance and scalability

SIMPLER HETEROGENEOUS APPLICATIONS

We want: *homogeneous* programs, *heterogeneous* execution

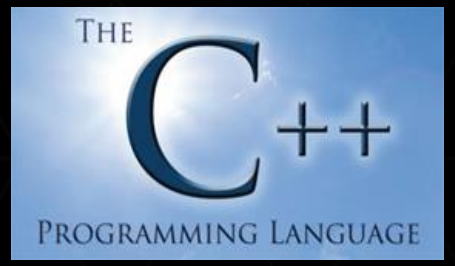
- Unified programming model includes parallelism in language
- Abstract heterogeneous execution via Runtime or Virtual Machine



PARALLELISM IN MAINSTREAM LANGUAGES

- Enable more programmers to write parallel software
- Give programmers the choice of language to use
- GPU support in key languages

GPU COMPUTING IN STANDARD LANGUAGES



Progress on a C++ Standard Parallel Algorithms Library

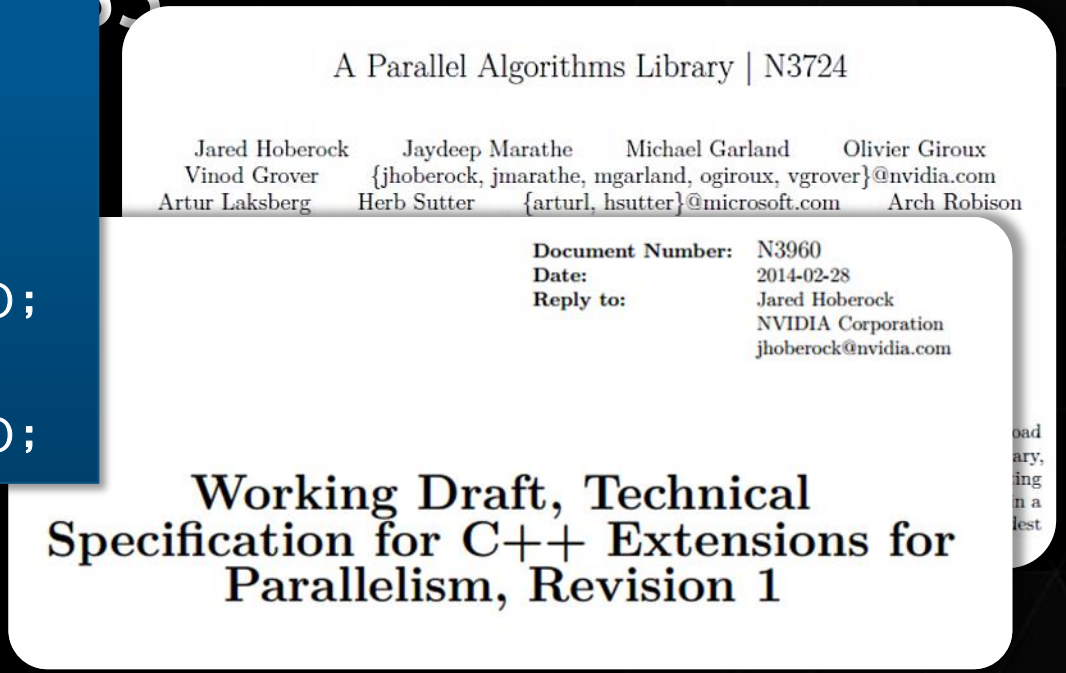
Open Source Compilers Support GPUs

Numba: Open Source Python compiler now supports GPUs

Prototype Java Bytecode Compiler for GPUs

C++ PARALLEL ALGORITHMS LIBRARY PROGRESS

```
std::vector<int> vec = ...

// previous standard sequential loop
std::for_each(vec.begin(), vec.end(), f);

// explicitly sequential loop
std::for_each(std::seq, vec.begin(), vec.end(), f);

// permitting parallel execution
std::for_each(std::par, vec.begin(), vec.end(), f);
```


A Parallel Algorithms Library | N3724

Jared Hoberock Jaydeep Marathe Michael Garland Olivier Giroux
Vinod Grover {jhoberock, jmarathe, mgarland, ogiroux, vgrover}@nvidia.com
Artur Laksberg Herb Sutter {arturl, hsutter}@microsoft.com Arch Robison

Document Number: N3960
Date: 2014-02-28
Reply to: Jared Hoberock
NVIDIA Corporation
jhoberock@nvidia.com

Working Draft, Technical Specification for C++ Extensions for Parallelism, Revision 1

- Complete set of parallel primitives: for_each, sort, reduce, scan, etc.
- ISO C++ committee voted unanimously to accept as official tech. specification working draft

N3960 Technical Specification Working Draft:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3960.pdf
Prototype:
https://github.com/n3554/n3554

Linux GCC Compiler to Support GPU Accelerators

Open Source

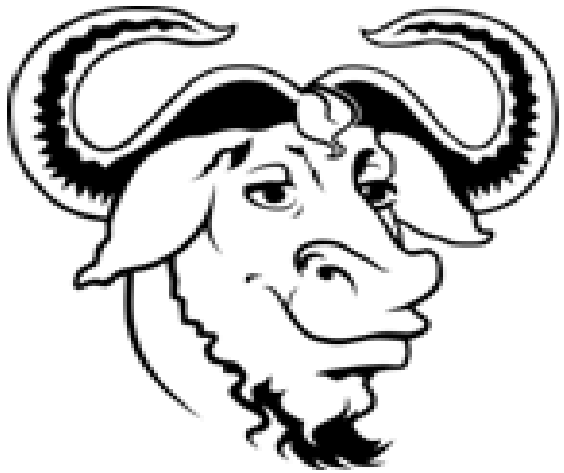
GCC Efforts by Samsung & Mentor Graphics

Pervasive Impact

Free to all Linux users

Mainstream

Most Widely Used HPC Compiler

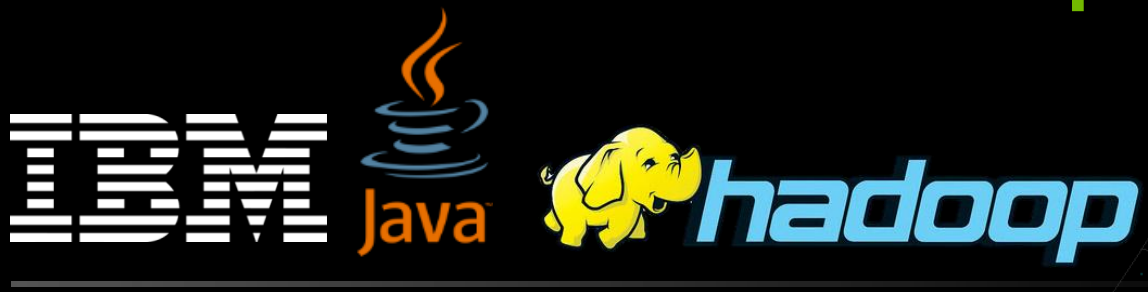


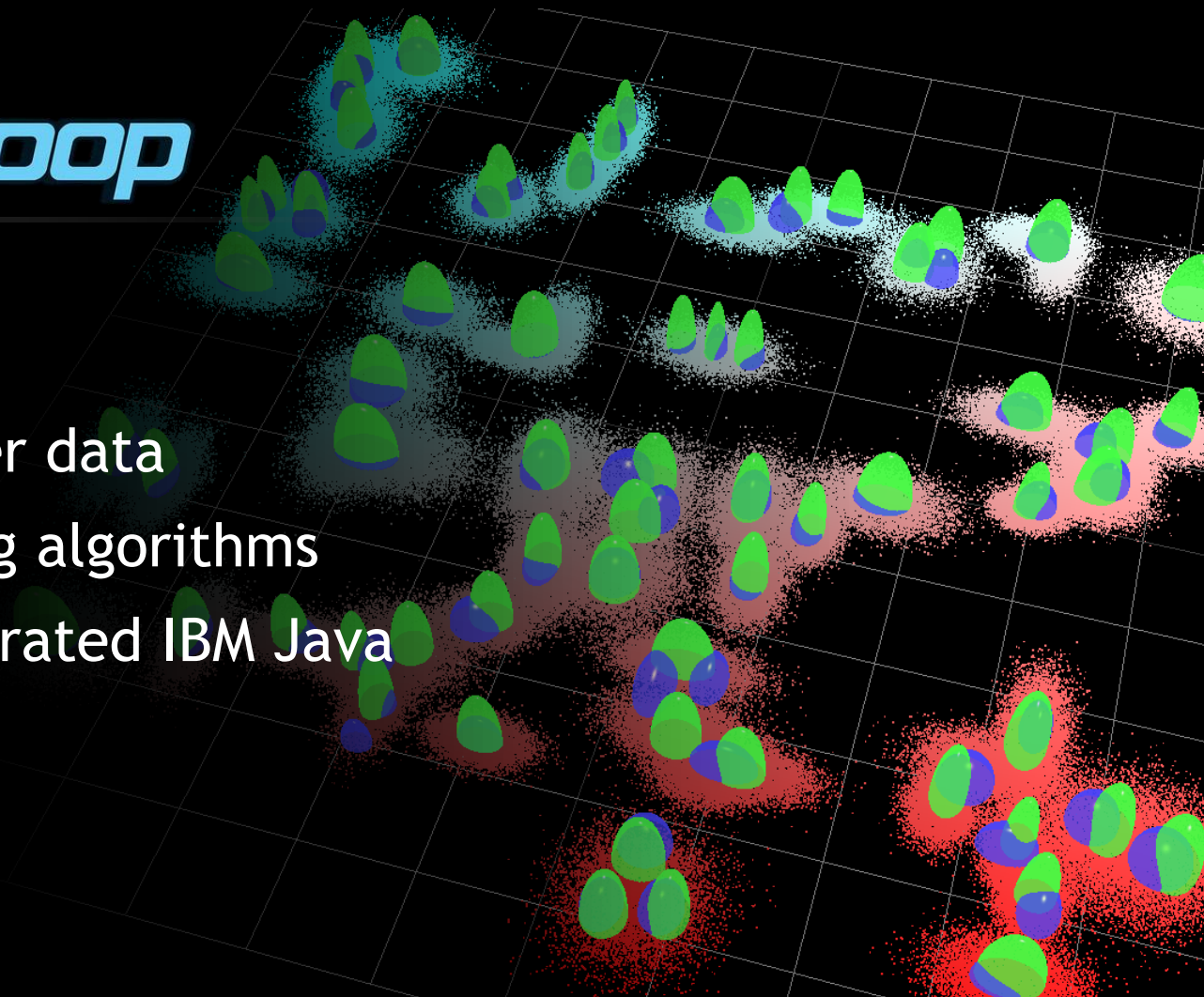
> *"Incorporating OpenACC into GCC is an excellent example of open source and open standards working together to make accelerated computing broadly accessible to all Linux developers."*
>
> Oscar Hernandez
> Oak Ridge National Laboratories

OAK RIDGE National Laboratory

OpenACC Directives for Accelerators

GPU-Accelerated Hadoop





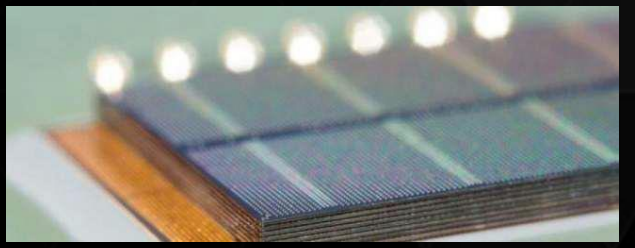
- Extract insights from customer data
- Data Analytics using clustering algorithms
- Developed using CUDA-accelerated IBM Java

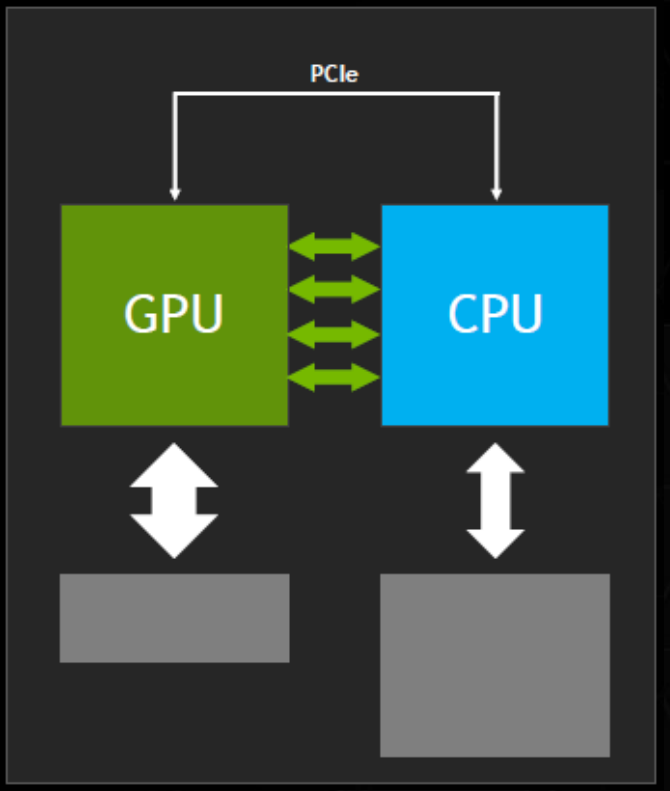
EFFICIENT HETEROGENEOUS SYSTEMS

- Stacked Memory:

  Much higher bandwidth
  2.5x capacity
  4x power efficiency

  - Larger working set, closer and faster to the GPU
- NVLink
  - GPU access with >= CPU bandwidth
- NVLink + Unified Memory
  - Single allocation of data
  - Simpler, Faster





PARALLEL FORALL

The Massively Parallel Programming Blog

Technical posts on GPUs, CUDA, OpenACC, Libraries, C/C++/Python and more

http://devblogs.nvidia.com/parallelforall

In-depth articles and regular series:

- CUDACasts: instructive videos
- CUDA Pro Tips: useful techniques
- CUDA Spotlight Interviews

Thank You