Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

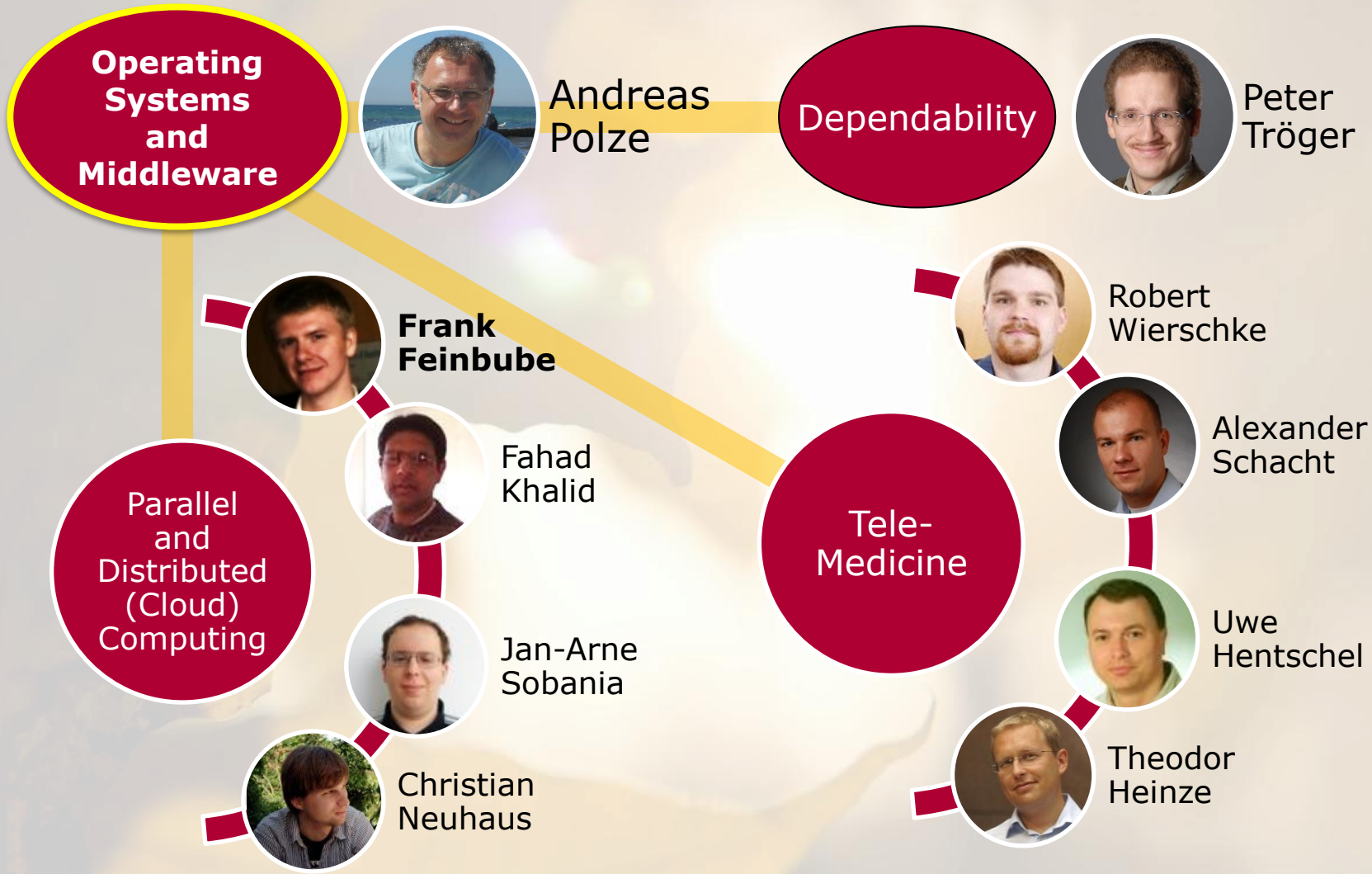
Code Mobilität: Adaptive Programmiermodelle für GPU-Computing

Frank Feinbube

Hasso Plattner Institute
Operating Systems and Middleware
Prof. Dr. Andreas Polze

Hasso Plattner Institute
University of Potsdam

Operating Systems and Middleware



The beautiful new world of Hybrid Compute Environments?

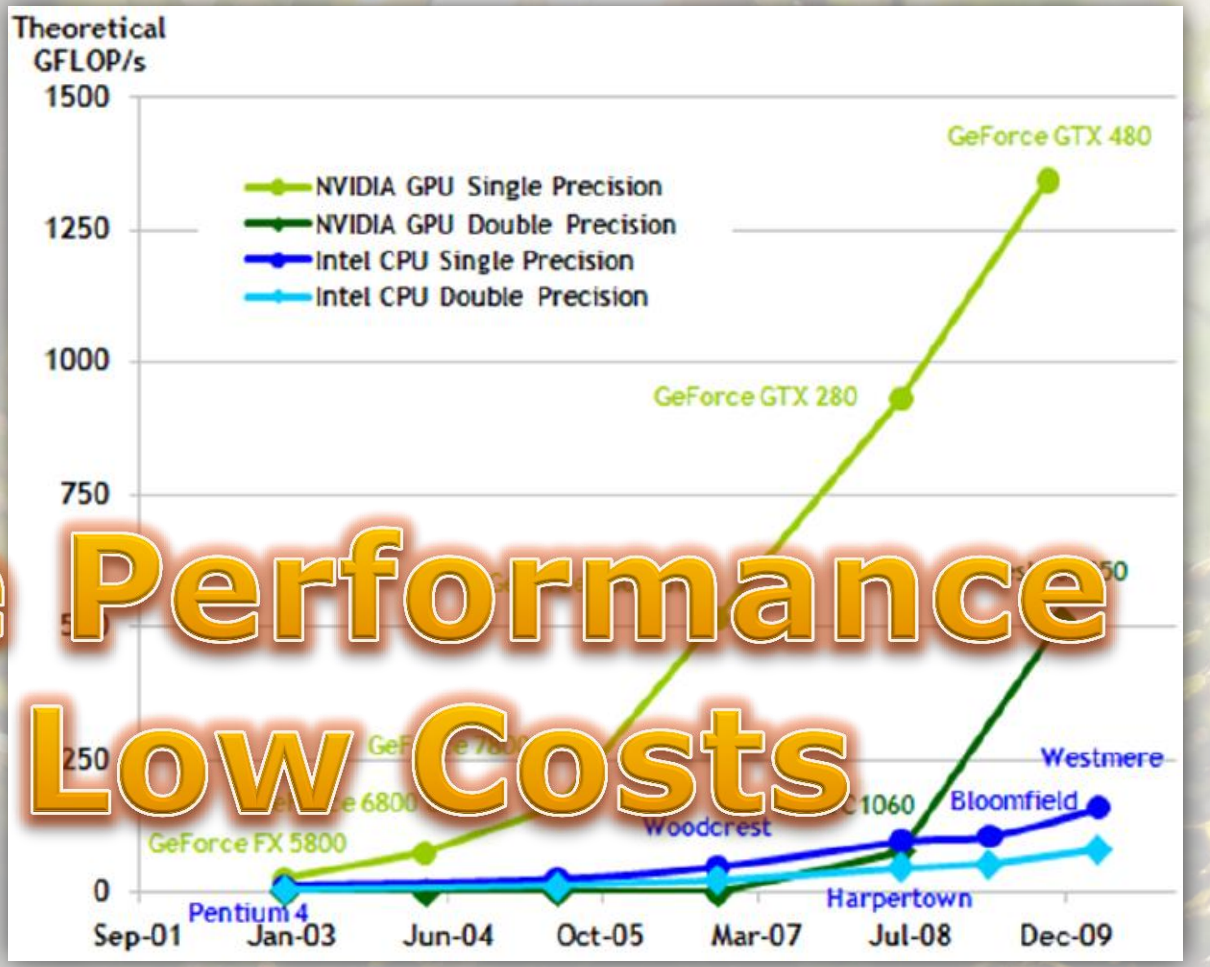
The beautiful new world of Hybrid Compute Environments?

Medical, Finance
Military, Research
Video and Photo
Energy, ...

Variety of Successful Applications

http://www.nvidia.com/object/cuda_apps_flash_new.html
http://www.nvidia.com/object/cuda_testimonials.html

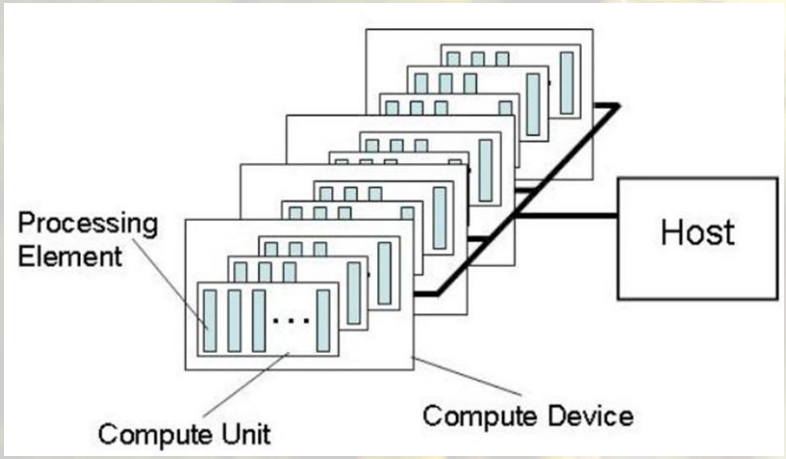
The beautiful new world of Hybrid Compute Environments?

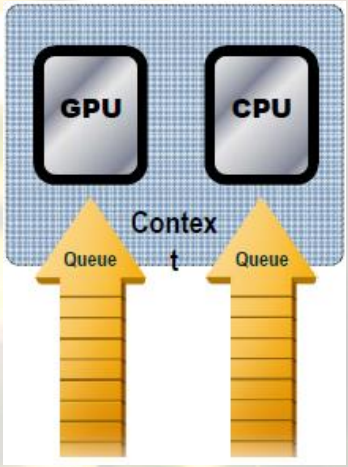


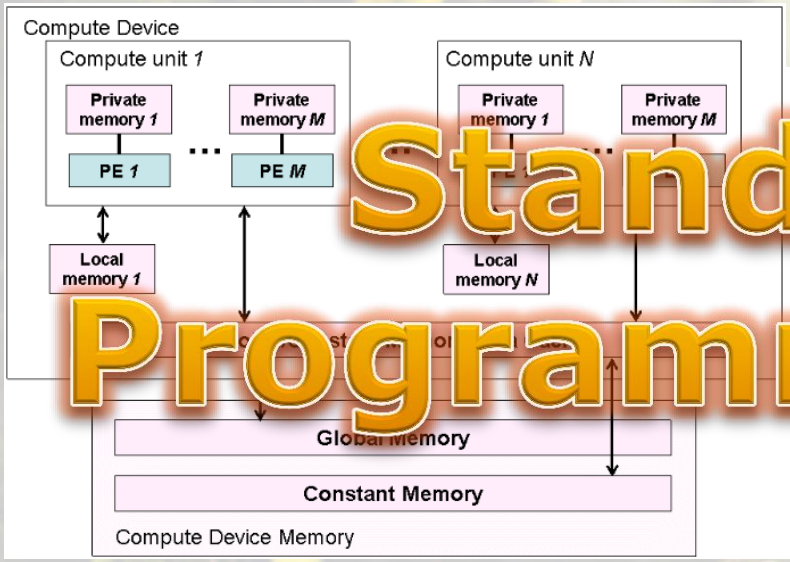
Huge Performance

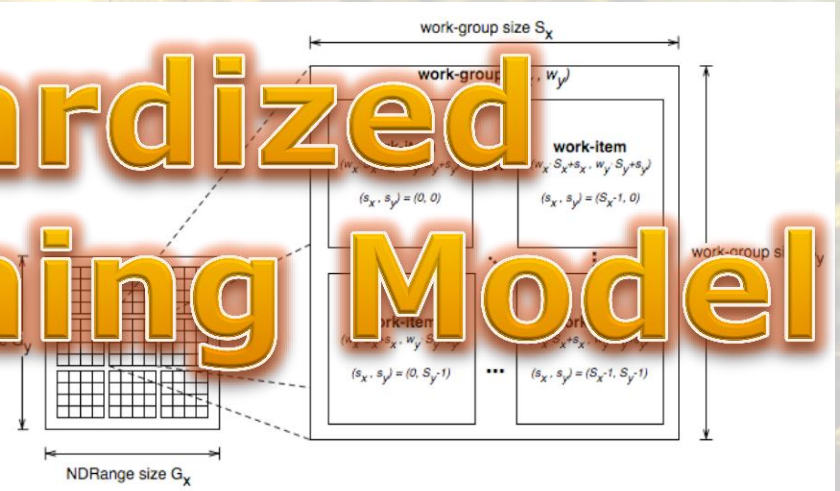
+ Low Costs

The beautiful new world of Hybrid Compute Environments?







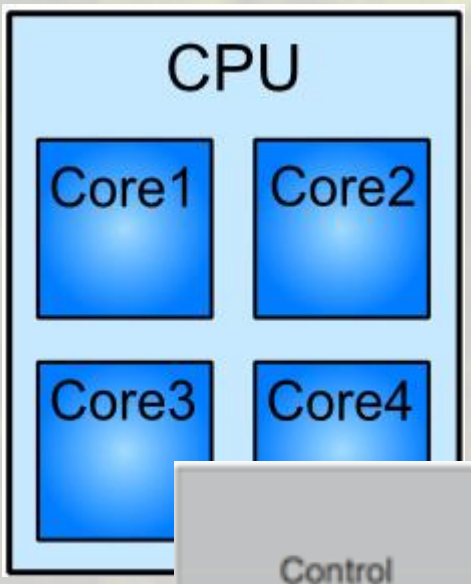


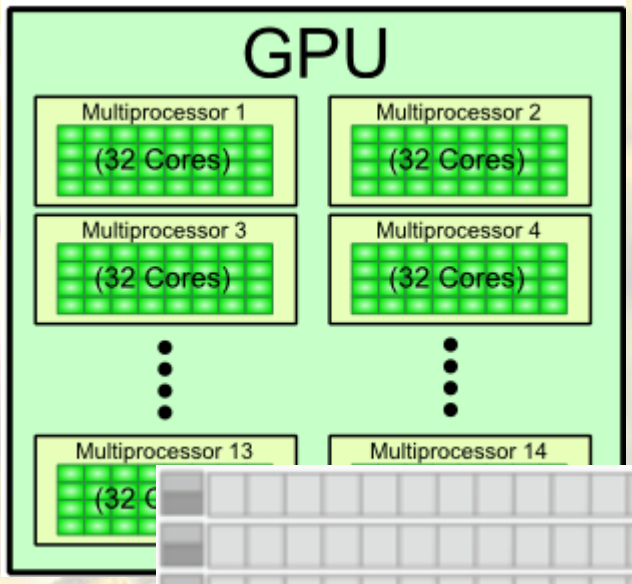
Standardized Programming Model

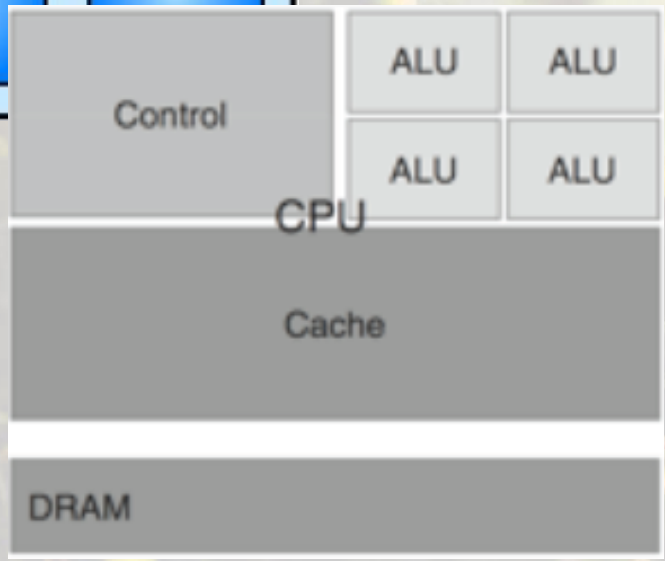
History of GPU Computing

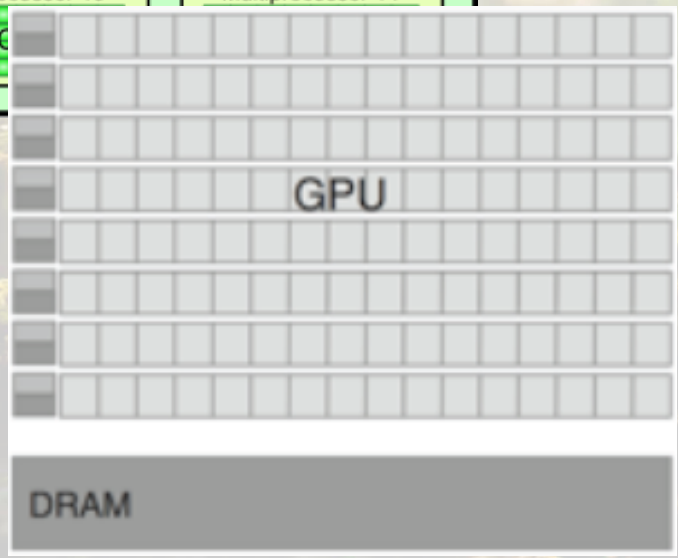
Fixed Function Graphic Pipelines	• 1980s-1990s; configurable, not programmable; first APIs (DirectX, OpenGL); Vertex Processing
Programmable Real-Time Graphics	• Since 2001: APIs for Vertex Shading, Pixel Shading and access to texture; DirectX9
Unified Graphics and Computing Processors	• 2006: NVIDIAs G80; unified processors arrays; three programmable shading stages; DirectX10
General Purpose GPU (GPGPU)	• compute problem as native graphic operations; algorithms as shaders; data in textures
GPU Computing	• Programming CUDA; shaders programmable; load and store instructions; barriers; atomics

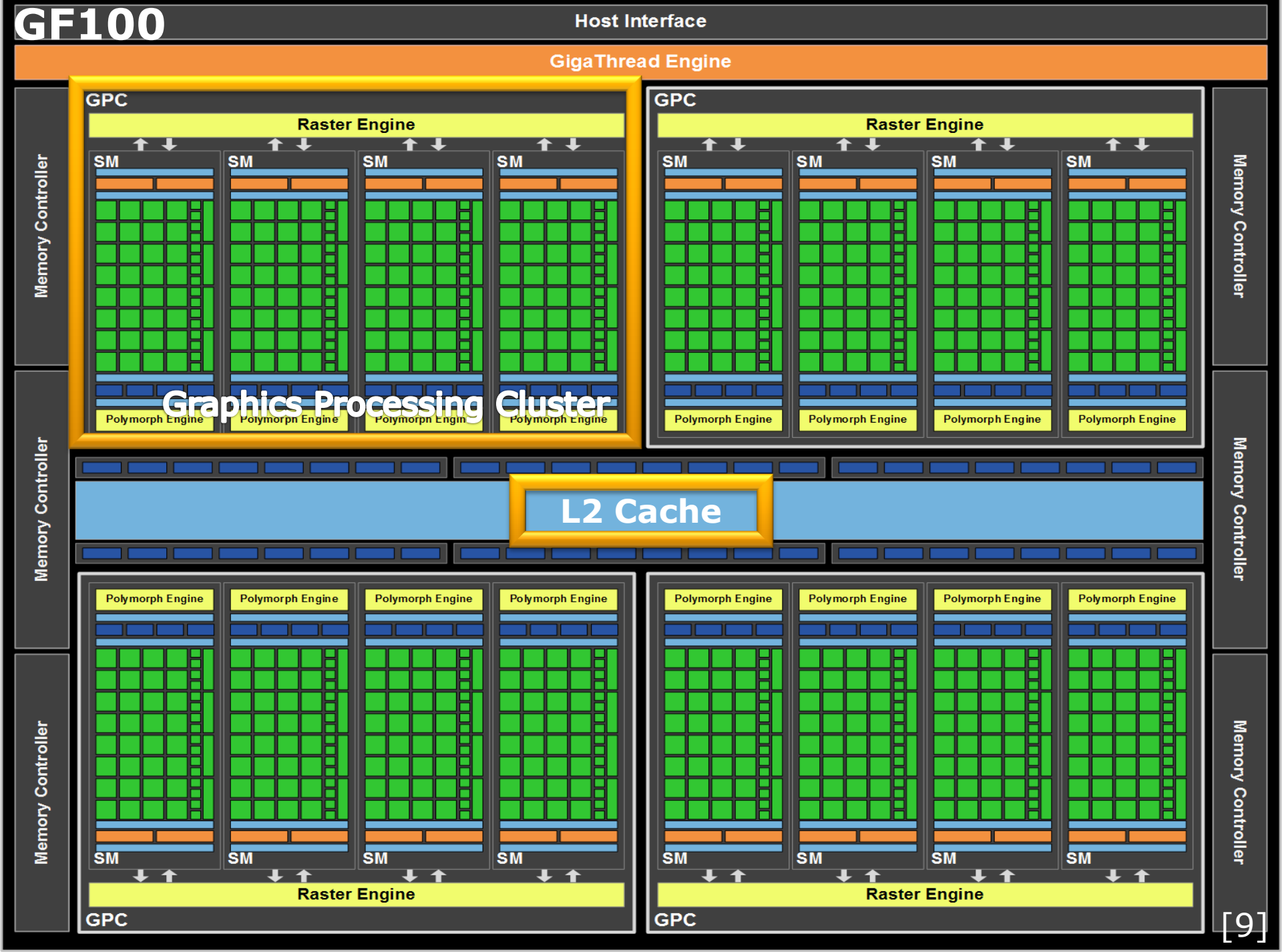
CPU vs. GPU Architecture



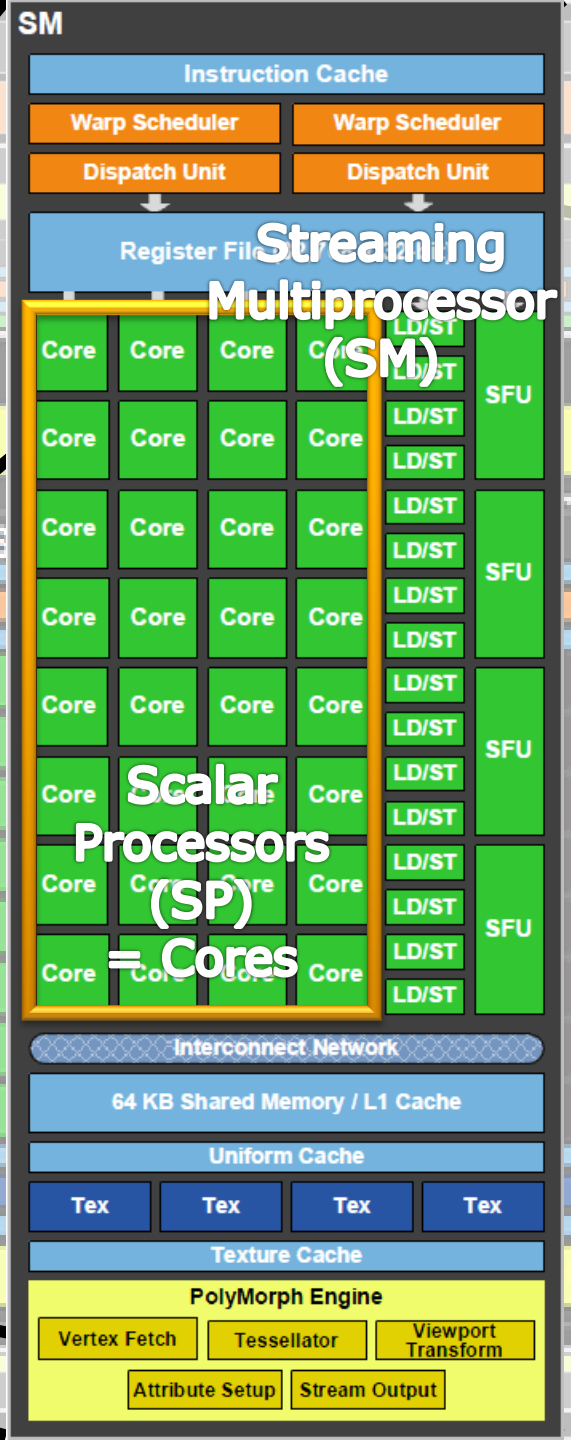




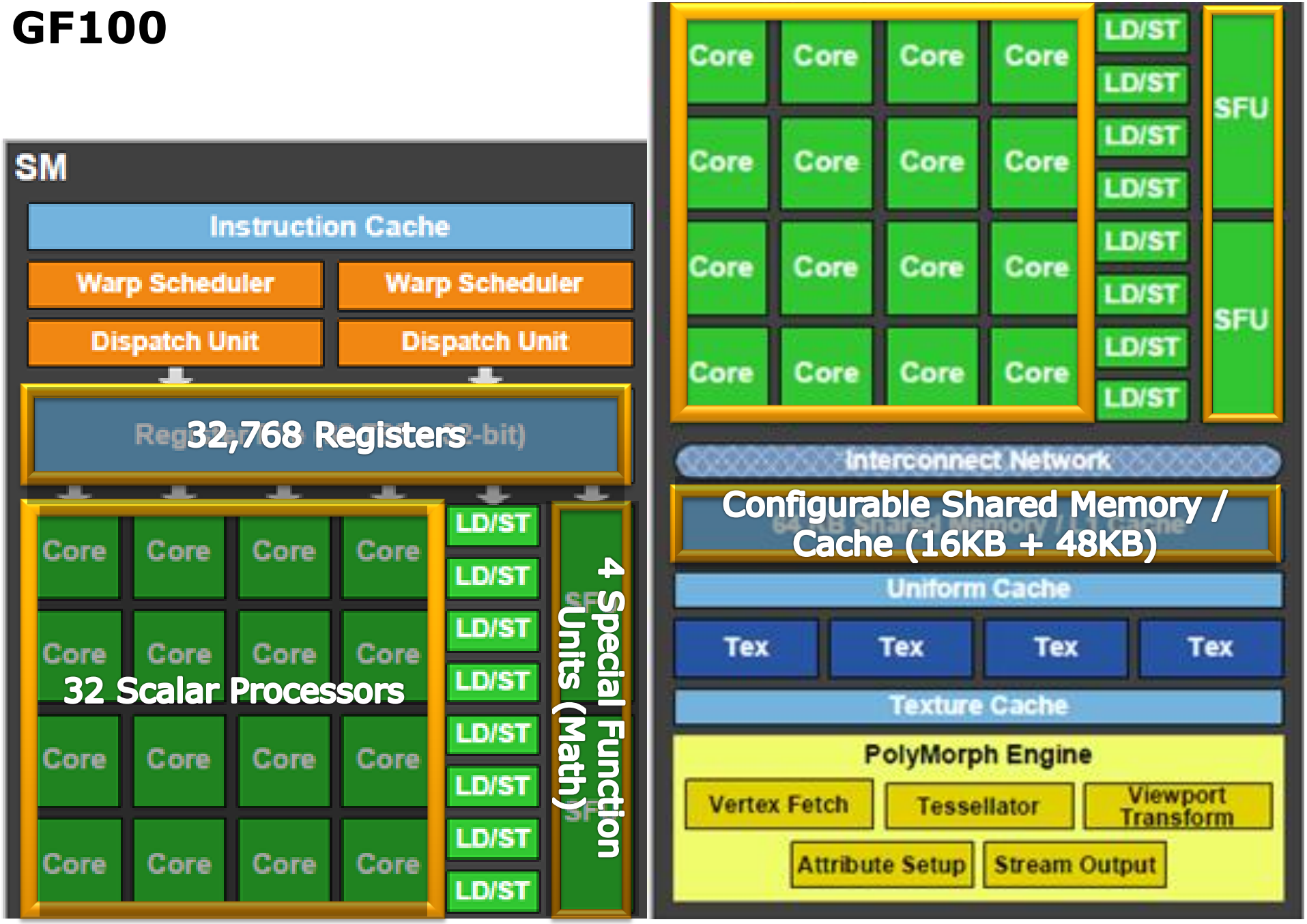



GF100
Host Interface
GigaThread Engine
Memory Controller
Memory Controller
Memory Controller
Memory Controller
Memory Controller
Memory Controller
GPC
Raster Engine
SM
Polymorph Engine
Graphics Processing Cluster
L2 Cache
GPC
Raster Engine
GPC
Raster Engine
GPC
Raster Engine

GF100

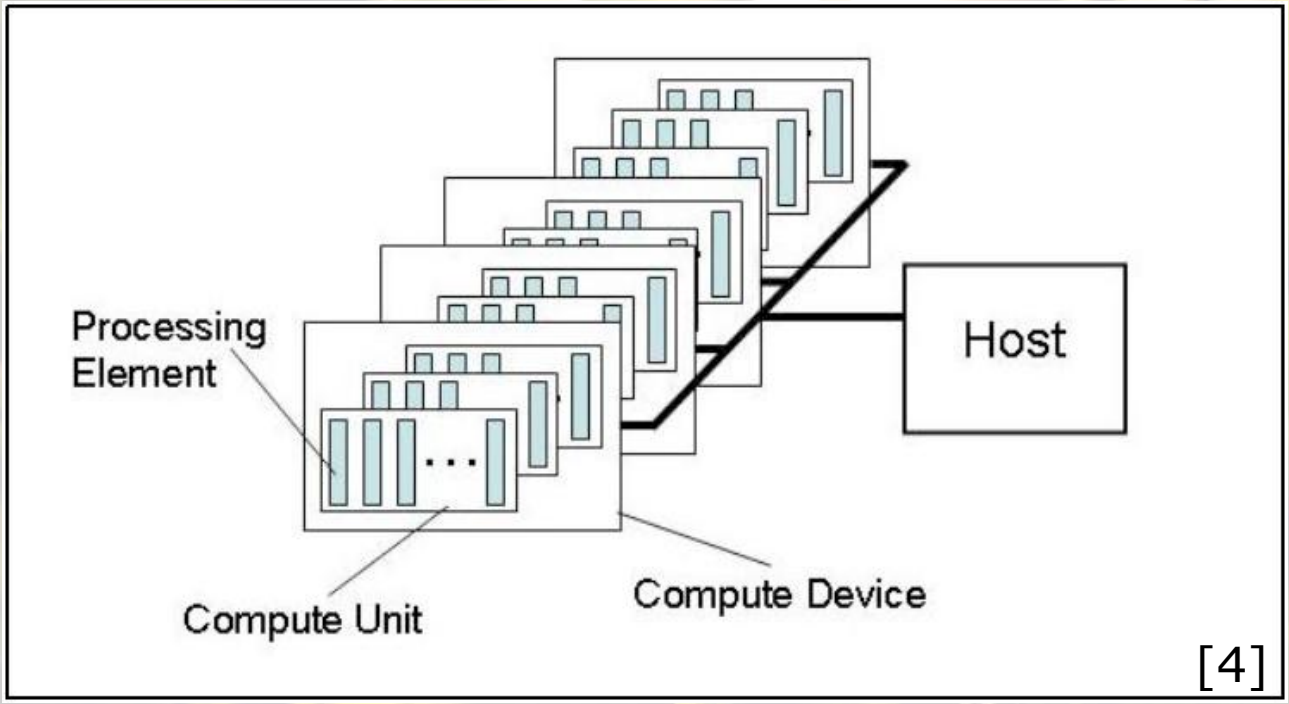

SM
Instruction Cache
Warp Scheduler
Warp Scheduler
Dispatch Unit
Dispatch Unit
Register File
Streaming Multiprocessor (SM)
Core
LD/ST
SFU
Scalar Processors (SP) = Cores
Interconnect Network
64 KB Shared Memory / L1 Cache
Uniform Cache
Tex
Texture Cache
PolyMorph Engine
Vertex Fetch
Tessellator
Viewport Transform
Attribute Setup
Stream Output

GF100


SM
Instruction Cache
Warp Scheduler
Warp Scheduler
Dispatch Unit
Dispatch Unit
32,768 Registers
Core
32 Scalar Processors
LD/ST
4 Special Function Units (Math)
SFU
Interconnect Network
Configurable Shared Memory / Cache (16KB + 48KB)
Uniform Cache
Tex
Texture Cache
PolyMorph Engine
Vertex Fetch
Tessellator
Viewport Transform
Attribute Setup
Stream Output

[9]

OpenCL Platform Model

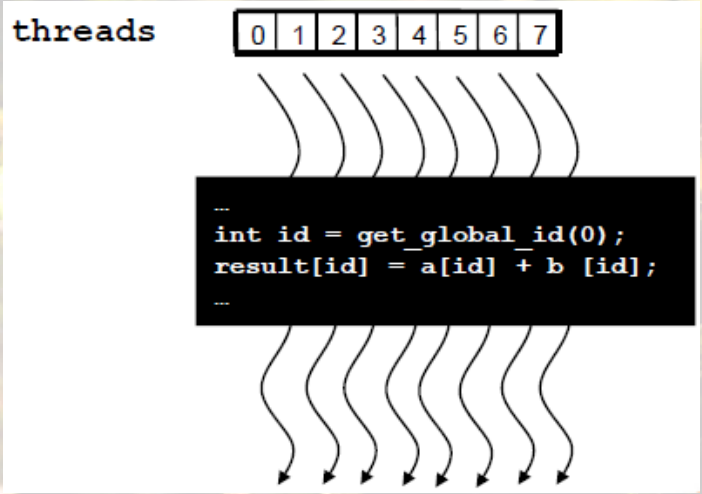


- OpenCL exposes CPUs, GPUs, and other Accelerators as “devices”
- Each “device” contains one or more “compute units”, i.e. cores, SMs,...
- Each “compute unit” contains one or more SIMD “processing elements”

OpenCL Execution Model

An OpenCL kernel is executed by an array of work items.

- All work items run the same code (SPMD)
- Each work item has an index that it uses to compute memory addresses and make control decisions

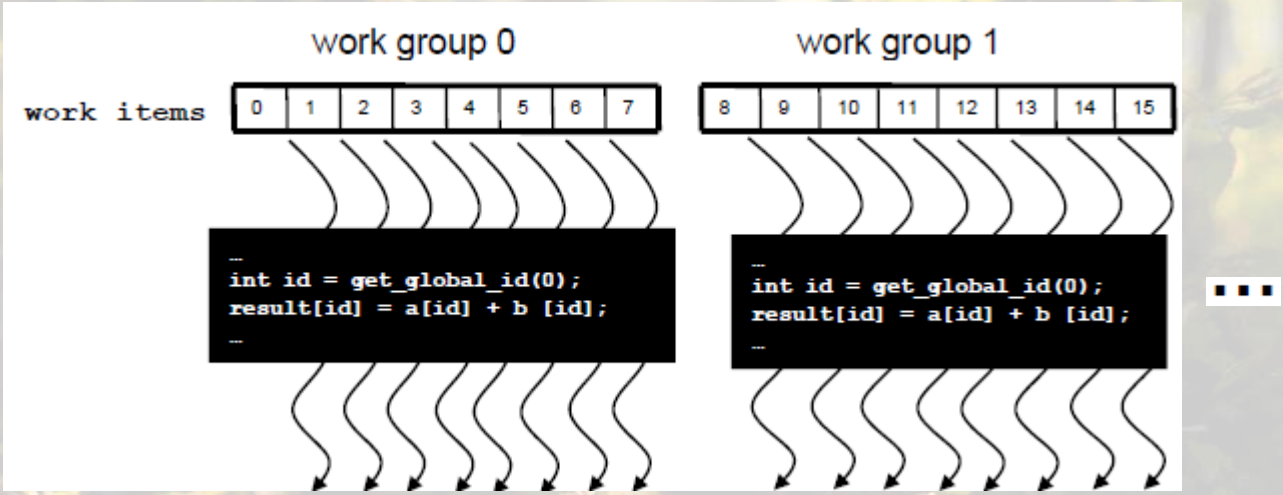


[1]

Work Groups: Scalable Cooperation

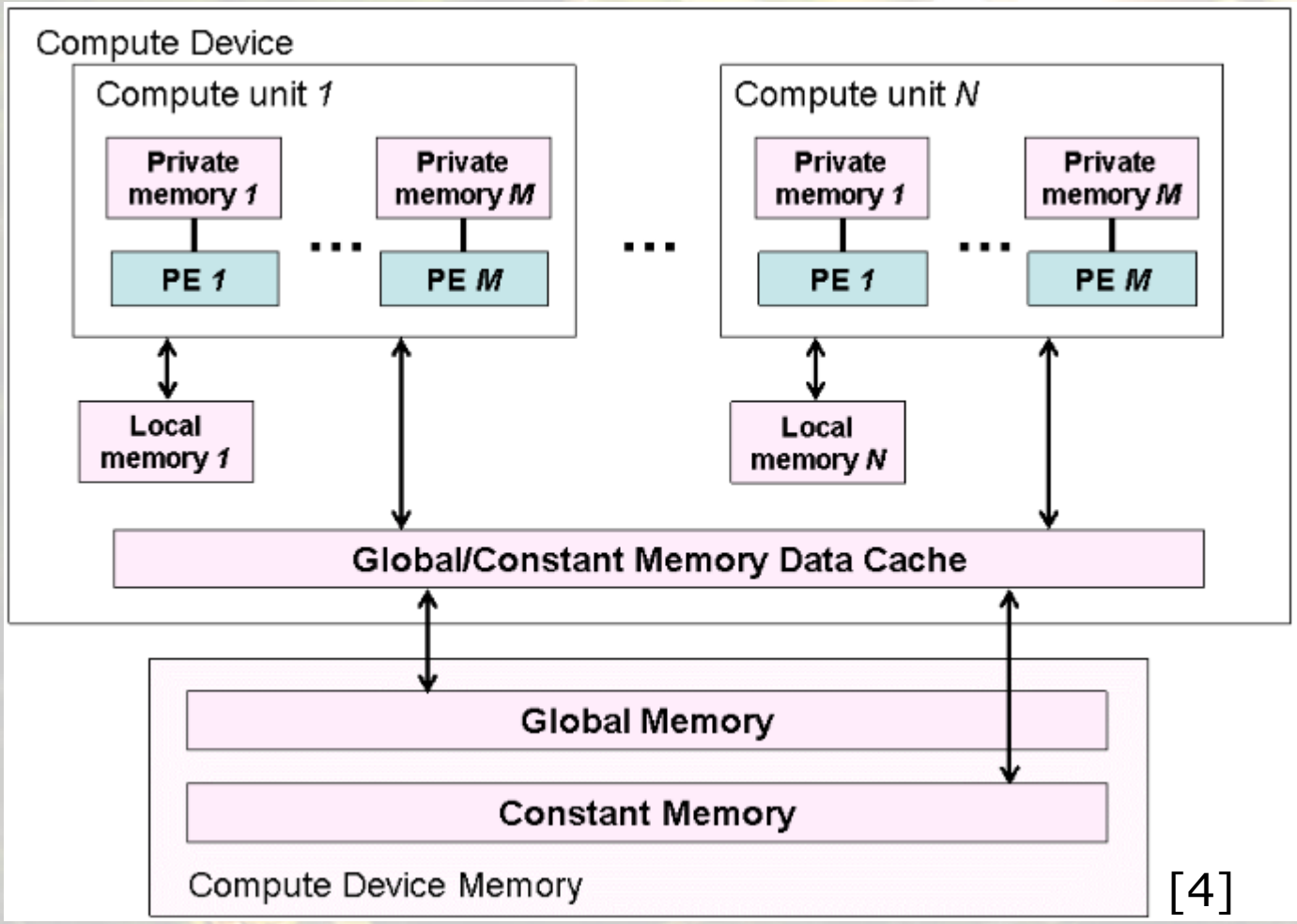
Divide monolithic work item array into work groups

- Work items within a work group cooperate via **shared memory, atomic operations** and **barrier synchronization**
- Work items in different work groups cannot cooperate



[1]

OpenCL Memory Architecture



[4]

Private
Per work-item

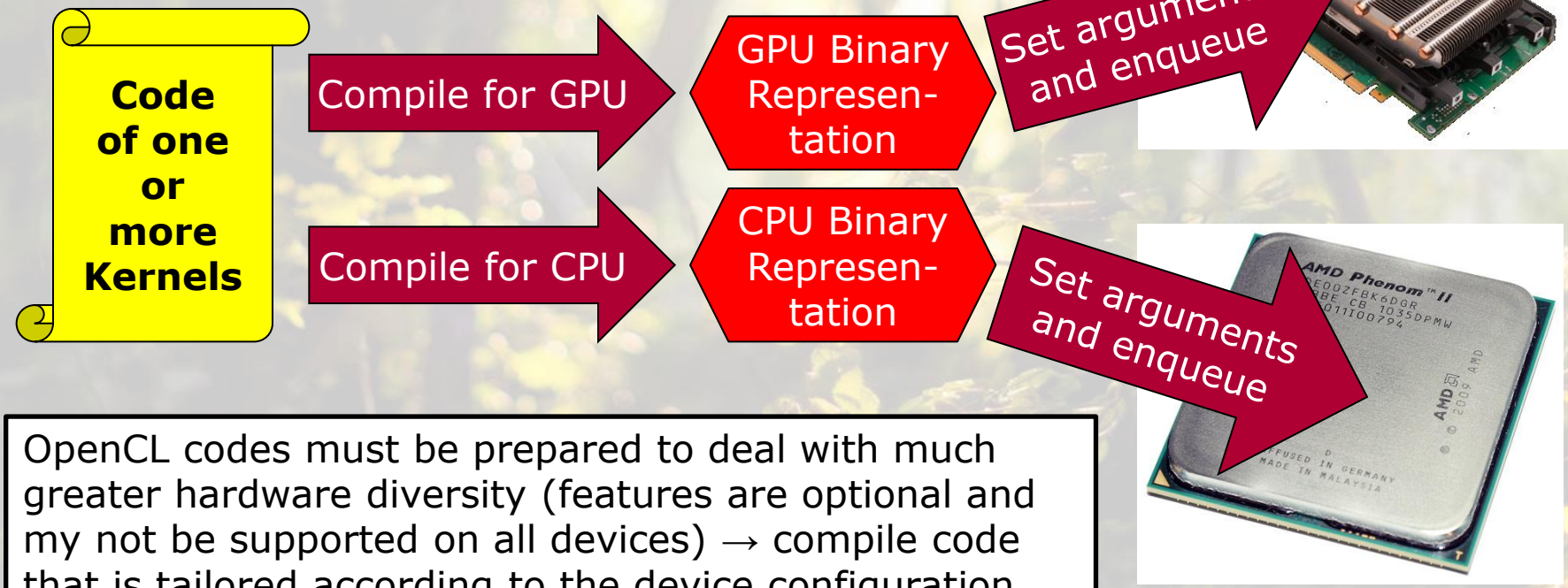
Local
Shared within a workgroup

Global/ Constant
Visible to all workgroups

Host Memory
On the CPU

Building and Executing OpenCL Code

Kernel **Program** **Device**

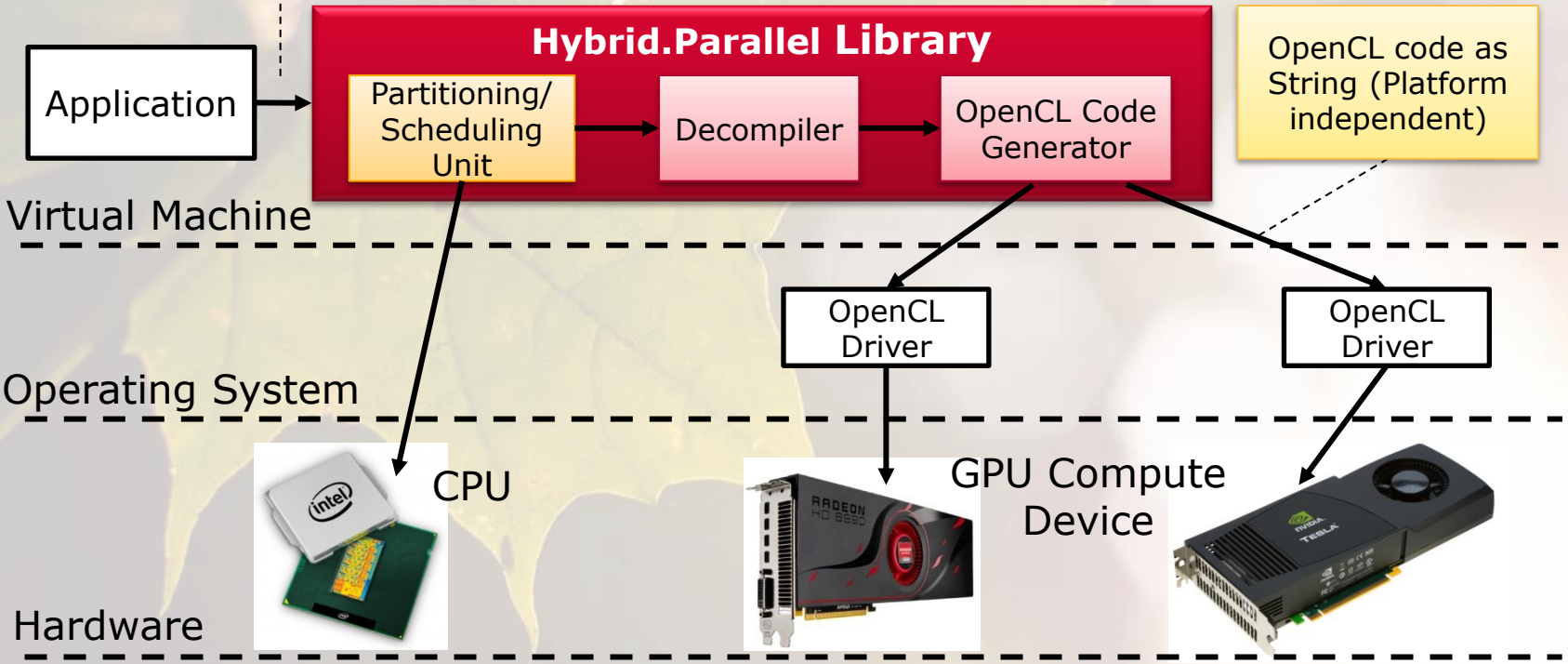


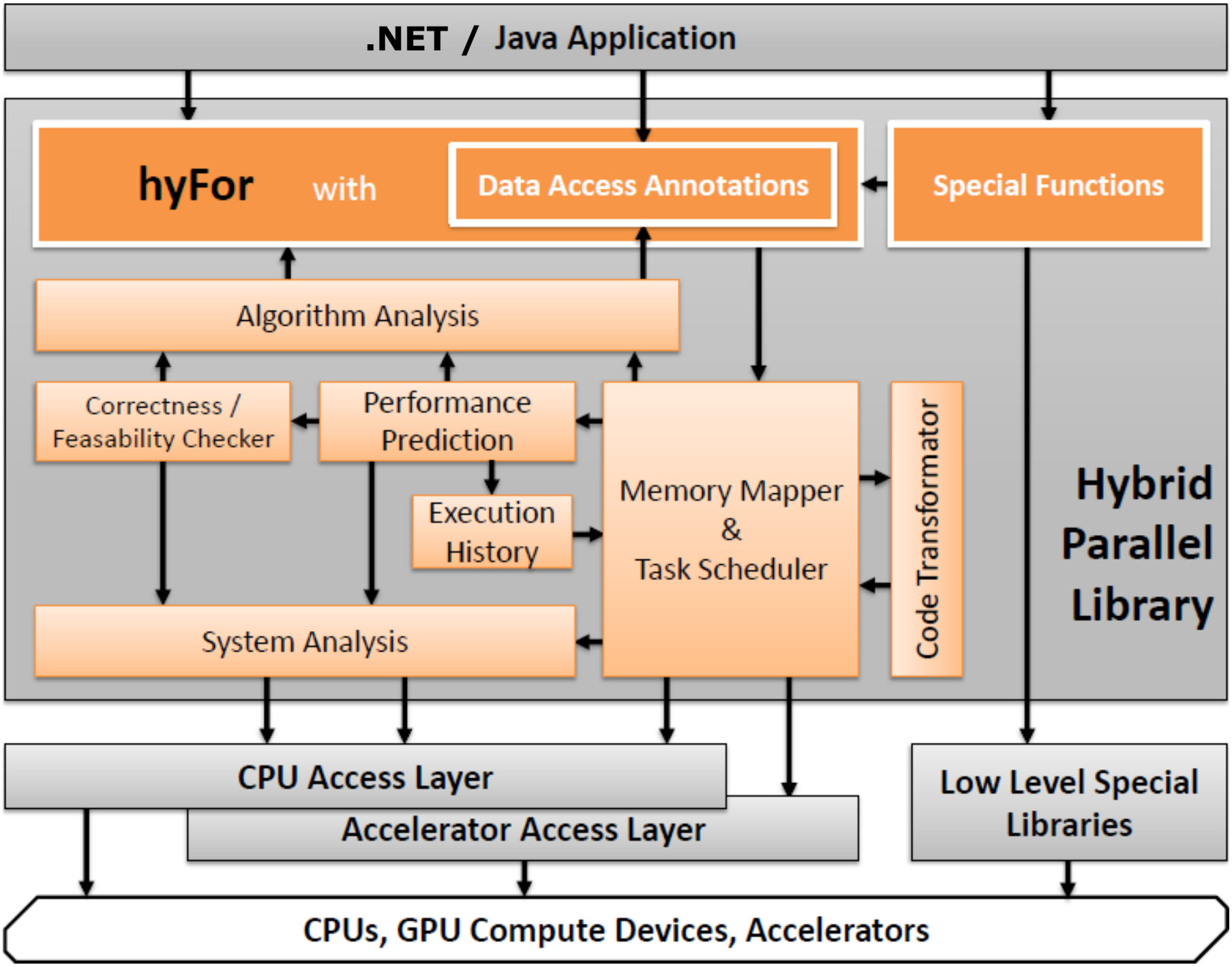
OpenCL codes must be prepared to deal with much greater hardware diversity (features are optional and my not be supported on all devices) → compile code that is tailored according to the device configuration

The Hybrid Parallel Project

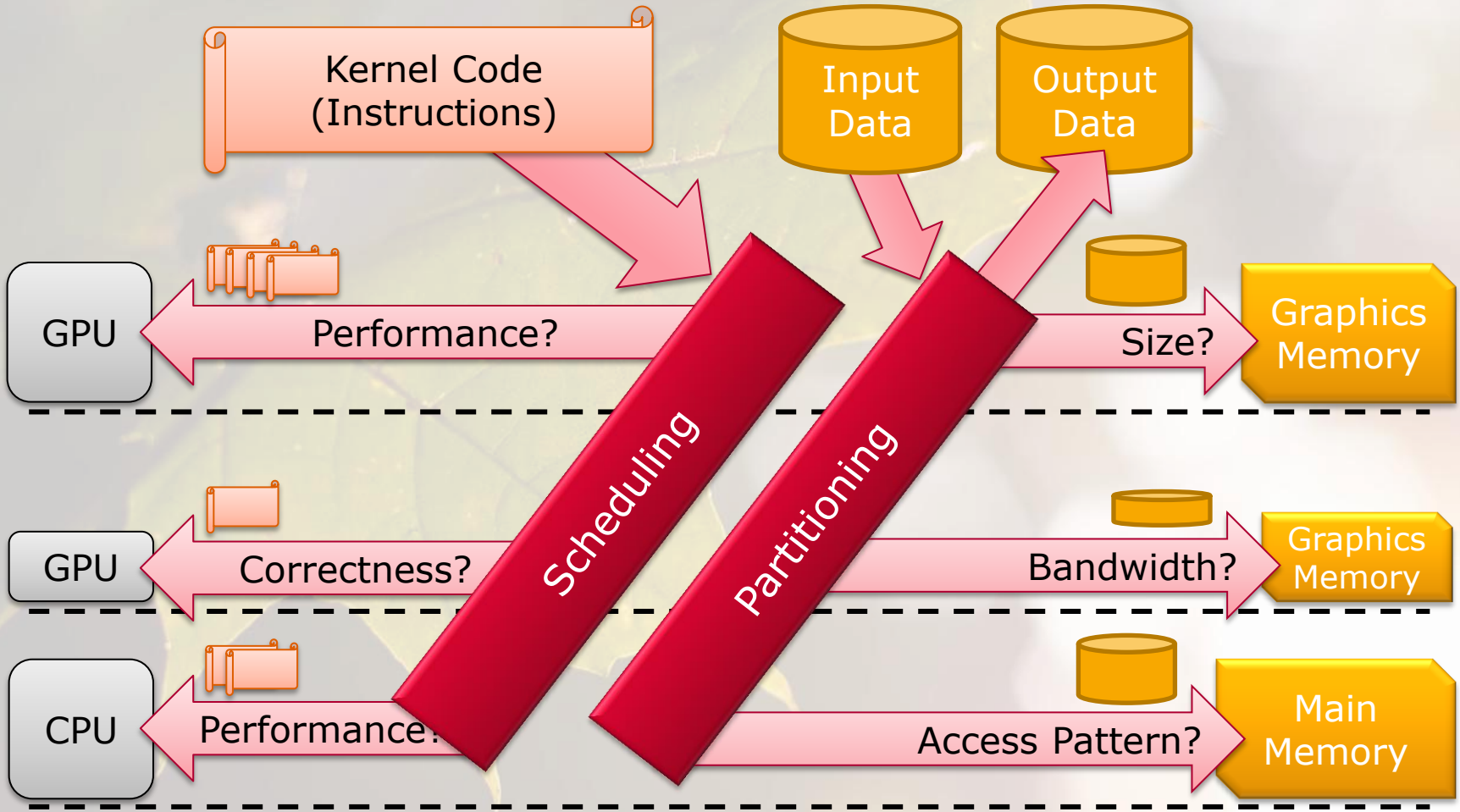

```
// based on Task Parallel Library
Threading.Tasks.Parallel.For(0, dataSize, action);

// our Library
Hybrid.Parallel.For(0, dataSize, action);
```

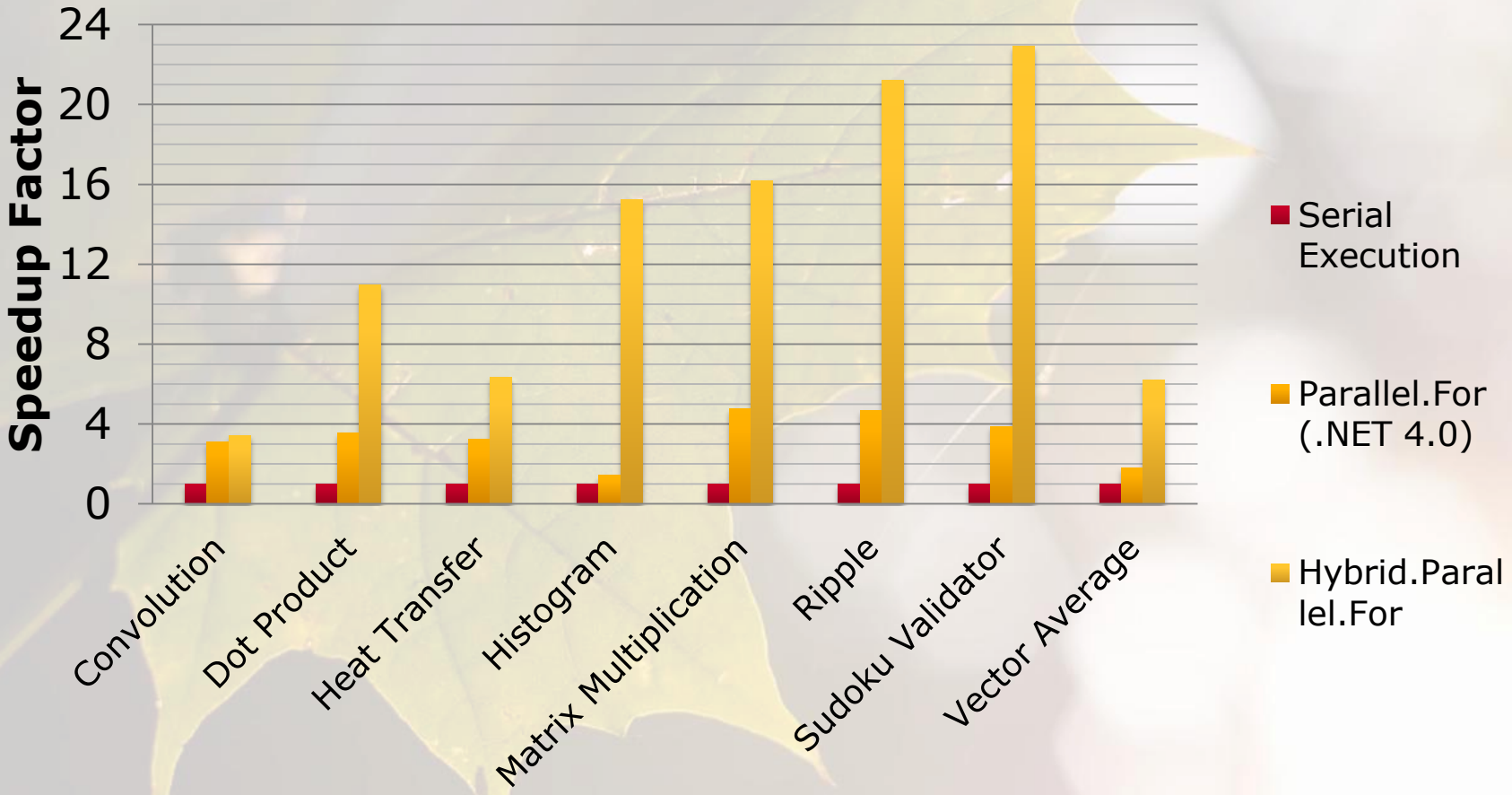



.NET / Java Application
hyFor with
Data Access Annotations
Special Functions
Algorithm Analysis
Correctness / Feasability Checker
Performance Prediction
Execution History
Memory Mapper & Task Scheduler
Code Transformator
System Analysis
Hybrid Parallel Library
CPU Access Layer
Accelerator Access Layer
Low Level Special Libraries
CPUs, GPU Compute Devices, Accelerators

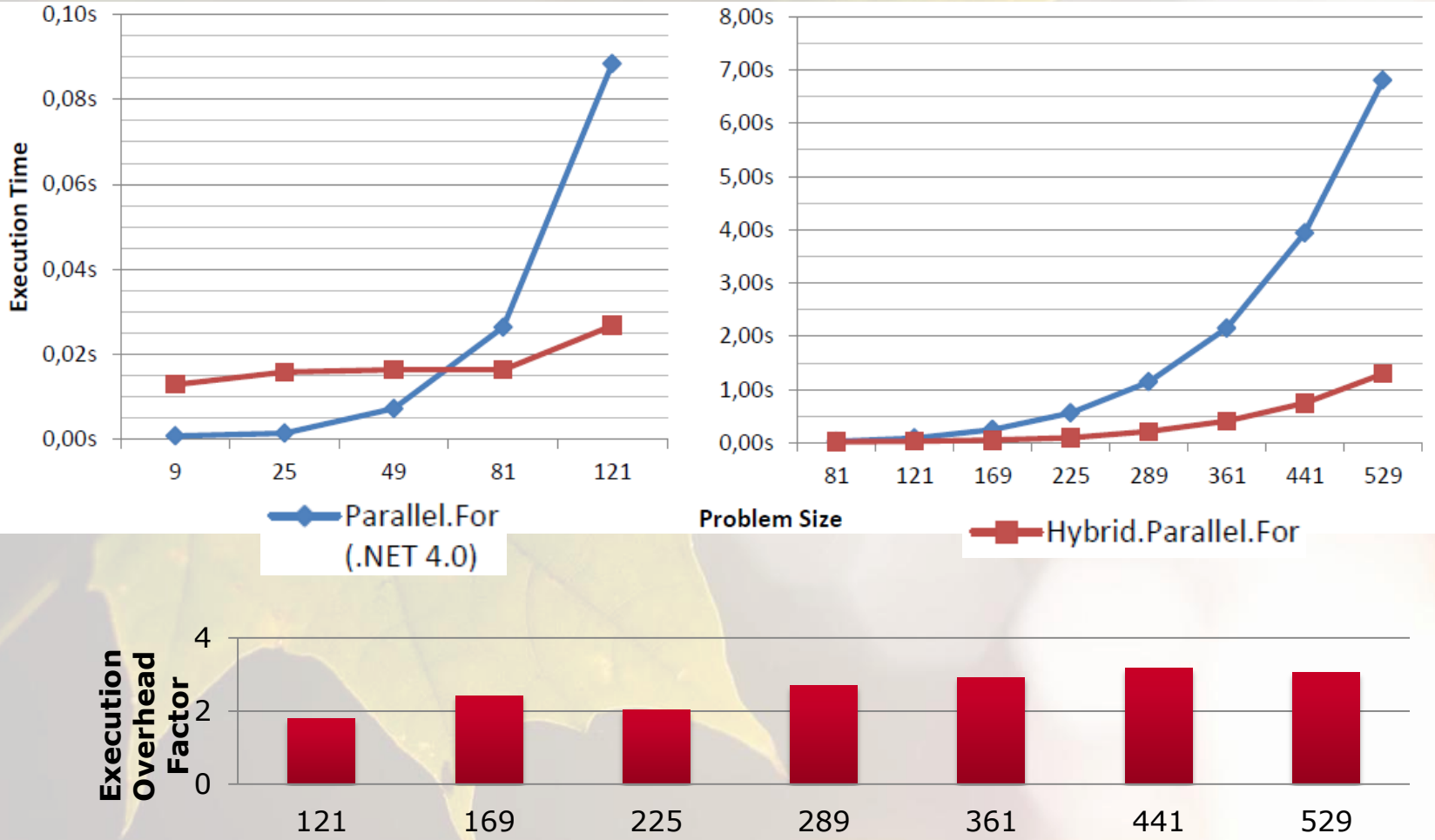
The Hybrid Parallel Project



Hybrid Parallel For .NET Evaluation



Hybrid Parallel For .NET Evaluation



Using the GPU for Operating System Services

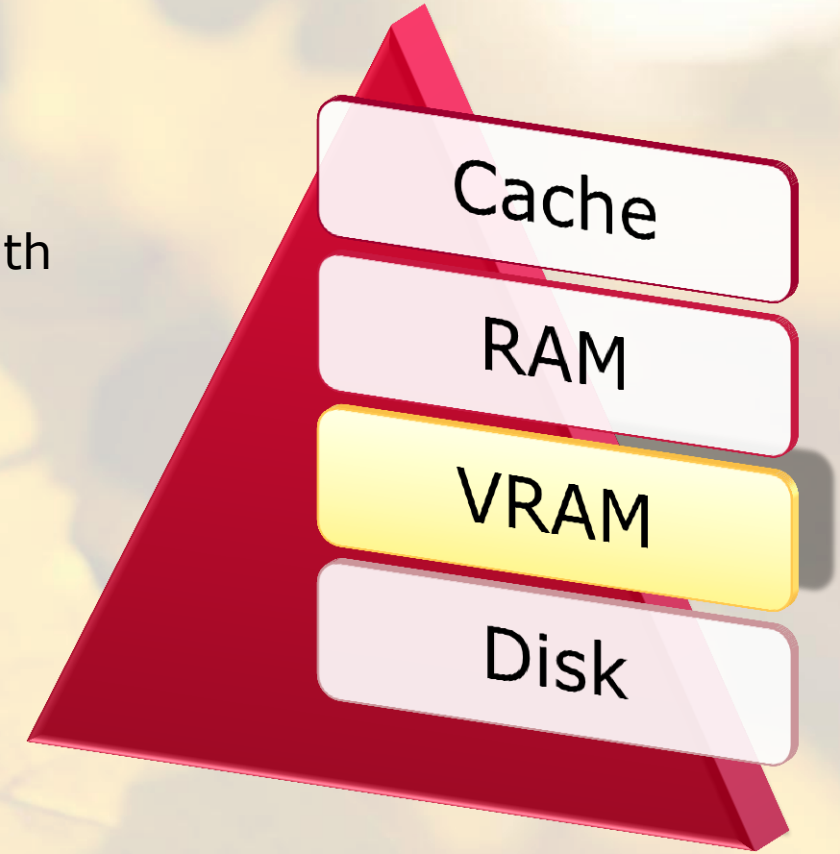
„Operating Systems must support GPU abstractions“

– Rossbach, Currey, Witchel [HotOS11]

- **PTask** API by Rossbach et al. (based on CUDA)
  - Workload as acyclic graphs of ptasks → executed on GPU
  - Proof-of-concept: encryption / decryption for Linux EncFS
- **GPUStore** by Sun et al. (based on CUDA)
  - Specially tailored for storage systems
  - GPUs for encryption and RAID functionality
- **Gdev** framework by Kato et al. (based on Direct Rendering Int.)
  - Reverse engineering of graphics card driver internals

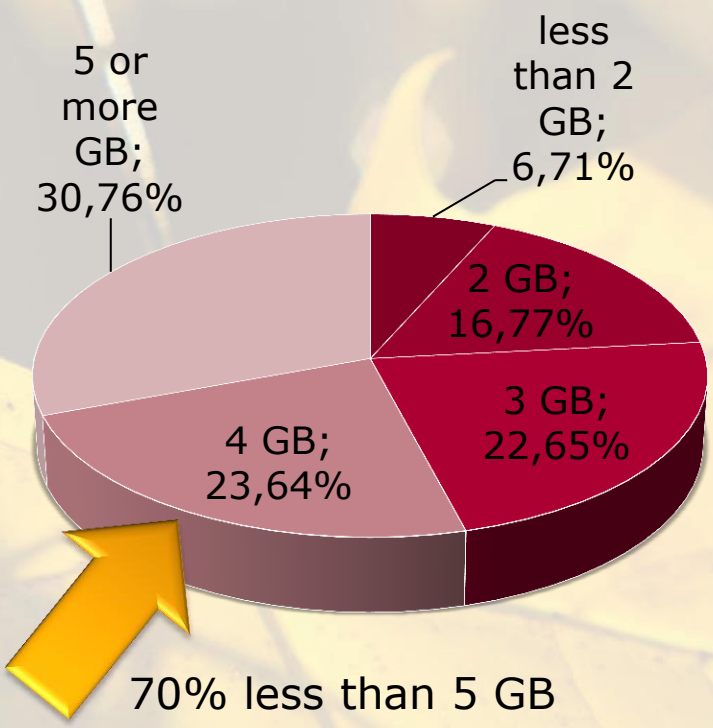
GPU Page Cache

- Use Global Memory as paging backing store
  - Typical desktop systems with VRAM of 1+ GB
  - VRAM unused most of the time
- Decrease paging operation latencies

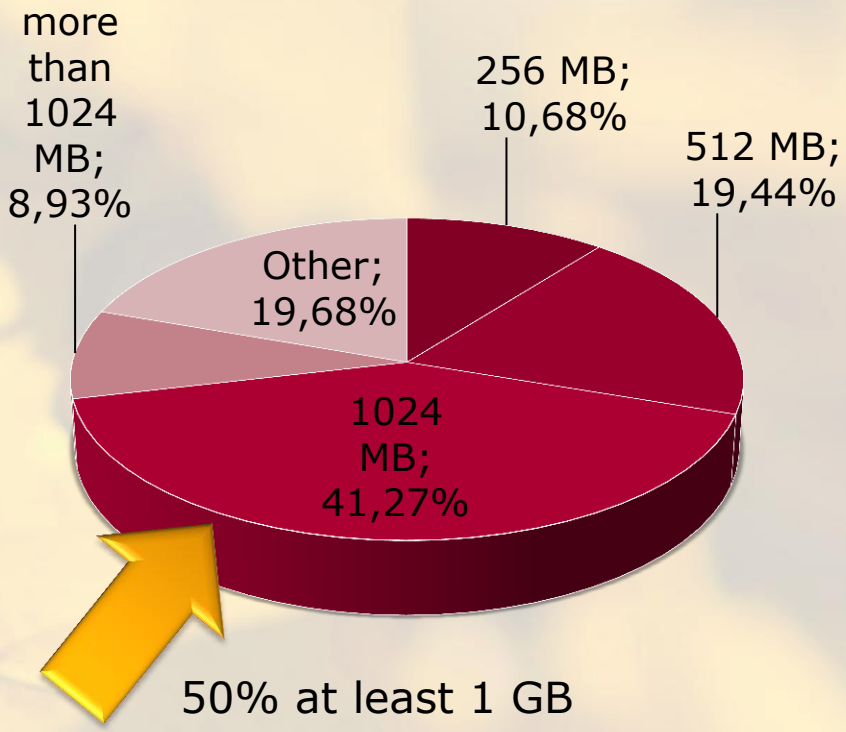


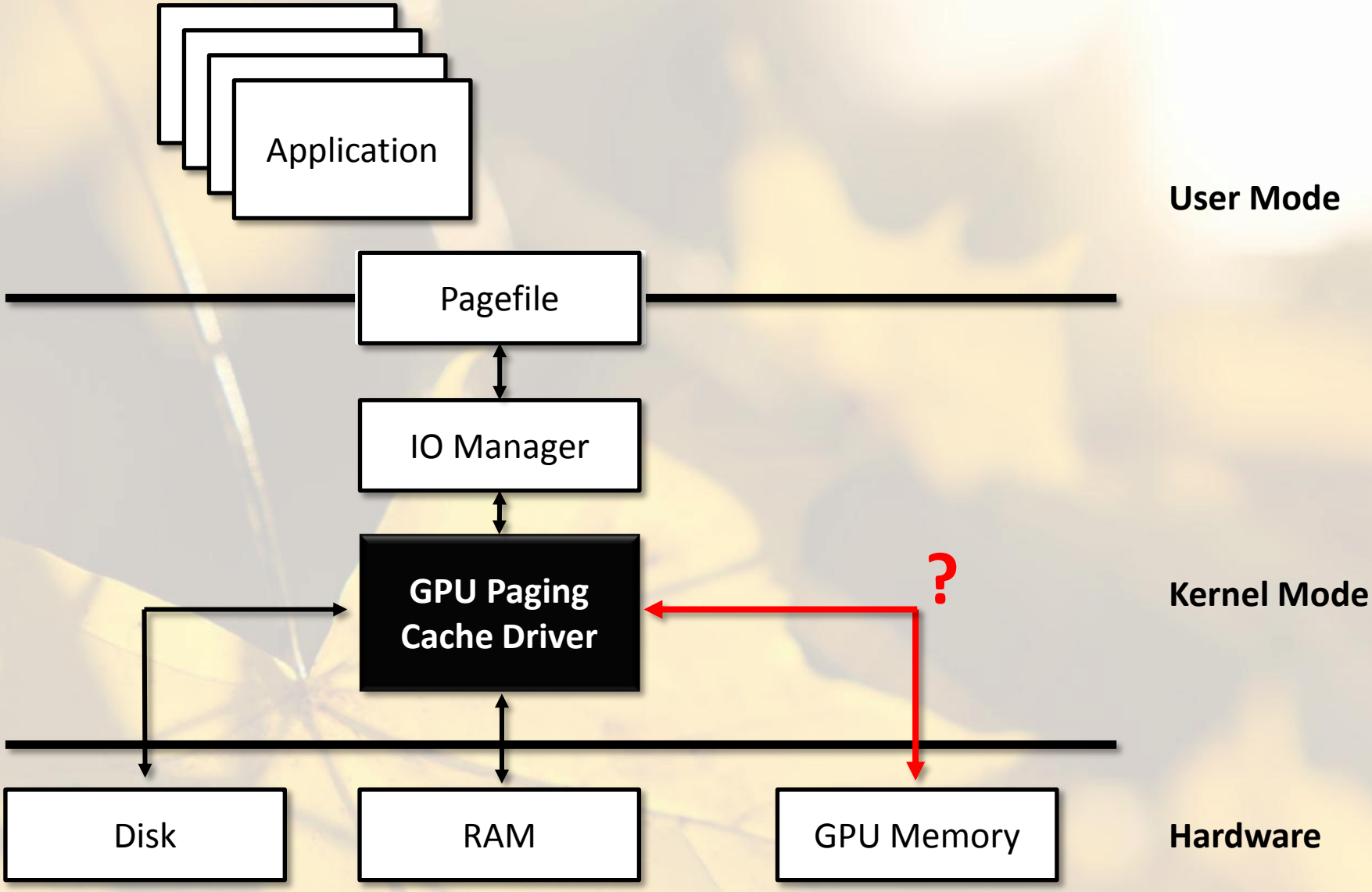
Hardware Survey

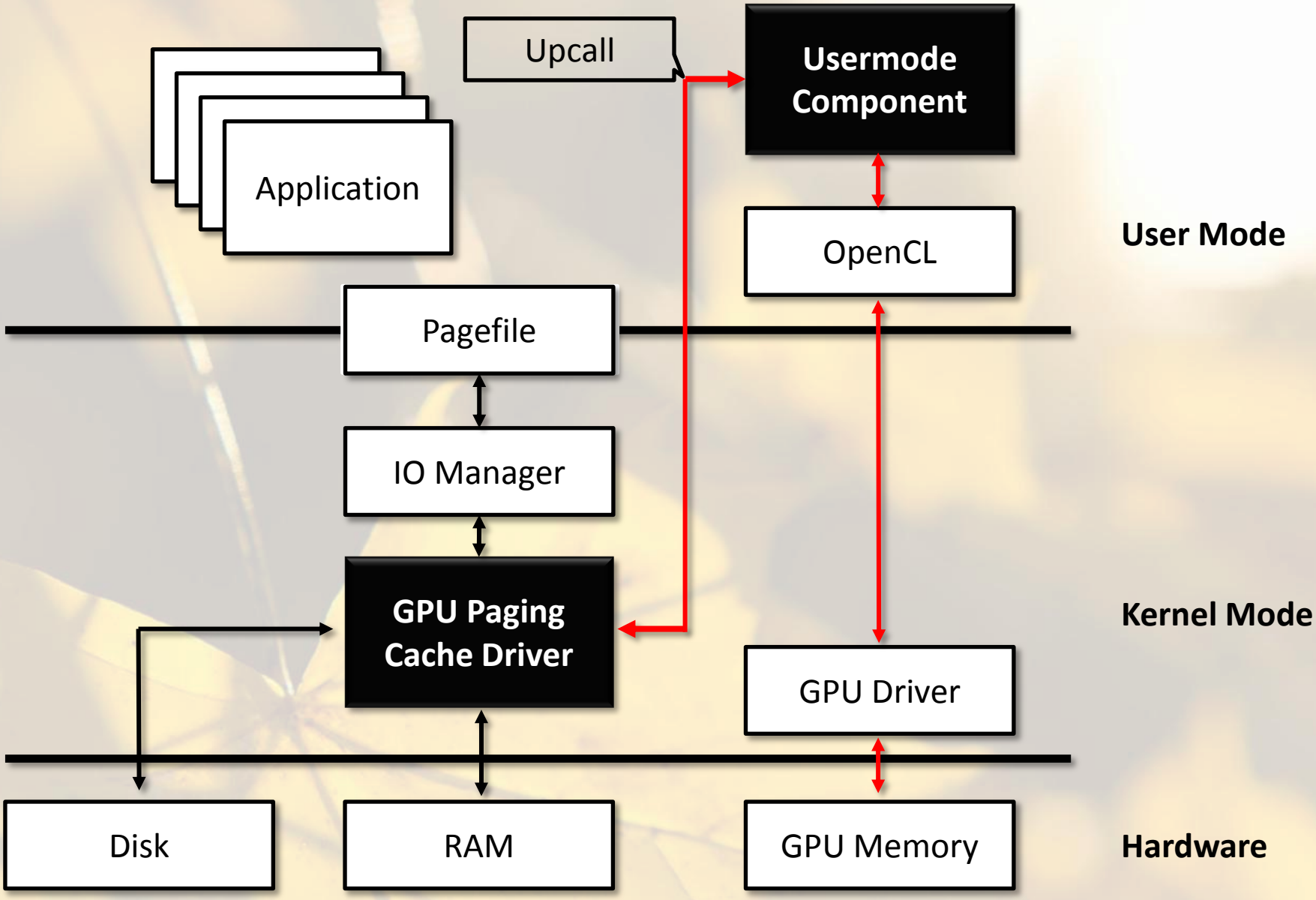
System RAM Distribution
(Steam HW Survey May 2012)

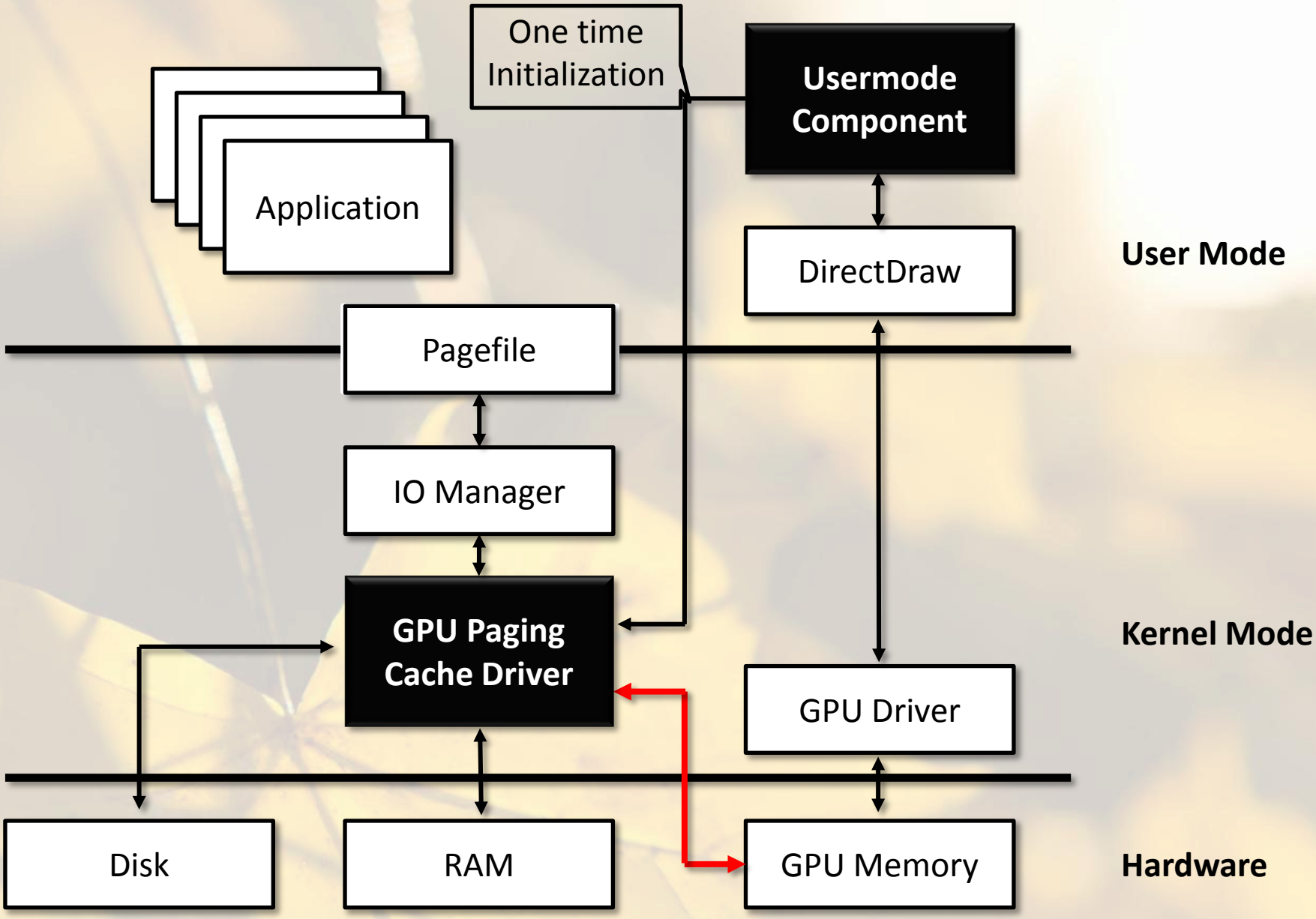


VRAM Distribution
(Steam HW Survey May 2012)

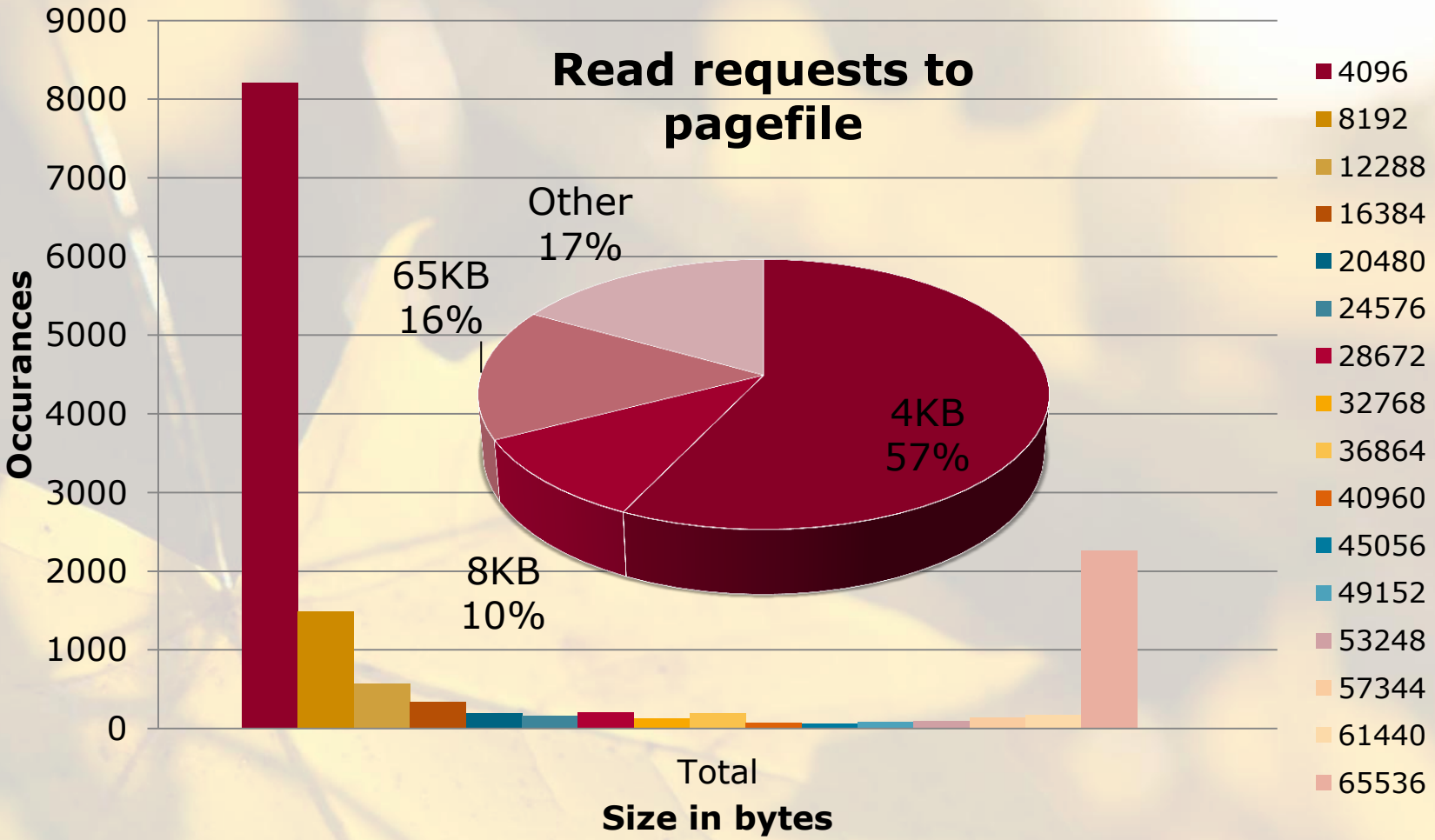



Application
User Mode
Pagefile
IO Manager
GPU Paging
Cache Driver
?
Kernel Mode
Disk
RAM
GPU Memory
Hardware

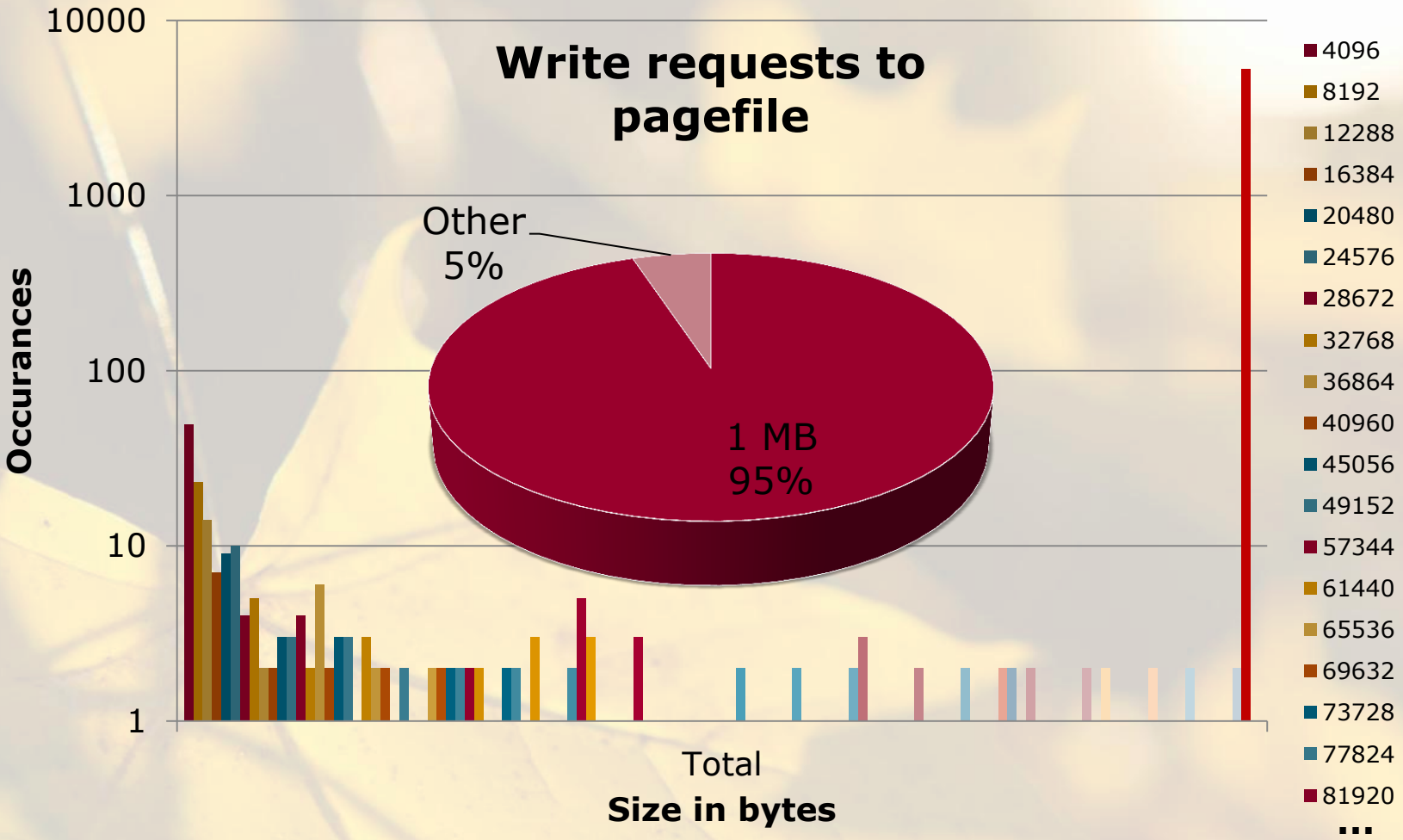

Upcall
Usermode Component
Application
OpenCL
User Mode
Pagefile
IO Manager
GPU Paging Cache Driver
Kernel Mode
GPU Driver
Disk
RAM
GPU Memory
Hardware


One time Initialization
Usermode Component
Application
DirectDraw
User Mode
Pagefile
IO Manager
GPU Paging Cache Driver
Kernel Mode
GPU Driver
Disk
RAM
GPU Memory
Hardware

Paging Request Read



Paging Request Write



Performance Metric

$$bandwidth = \frac{1}{RTT}$$

$$RTT = RTT_{Read} + RTT_{Write}$$

$$RTT_{Read} = \sum_{i=4KB}^{1MB} \frac{Probability_{Read}(i)}{Bandwidth_{Read}(i)}$$

$$RTT_{Write} = \sum_{i=4KB}^{1MB} \frac{Probability_{Write}(i)}{Bandwidth_{Write}(i)}$$

$$bandwidth = \frac{1}{\sum_{i=4KB}^{1MB} \frac{Probability_{Read}(i)}{Bandwidth_{Read}(i)} + \sum_{i=4KB}^{1MB} \frac{Probability_{Write}(i)}{Bandwidth_{Write}(i)}}$$

Tested Hardware

ID	CPU	GPU	Memory	Disk	Operating System
2600k 460gtx	Intel Core i7-2600K 3,7 Ghz	Nvidia GeForce GTX 460 1023 MB GDDR5	8 GB DDR3	Intel Rapid Storage 64 GB SSD	Windows 7 Ultimate SP1 64 bit
E8500 275gtx	Intel Core 2 Duo E8500 3,17 Ghz	Nvidia GeForce GTX 275 896 MB GDDR3	8 GB DDR2	1 TB HDD	Windows 7 Ultimate SP1 64 bit
2620m 540mgt	Intel Core i7-2620M 2,7 Ghz	Nvidia GeForce GT 540M 3 GB GDDR3	8 GB DDR3	240 GB SSD	Windows 7 Ultimate SP1 64 bit

Evaluation

	Disk	User Mode GPU-based Paging Cache	Kernel Mode GPU-based Paging Cache	System Memory
2600k 460gtx	18,33 MB/s	112,78 MB/s	106,58 MB/s	1061,18 MB/s
2620m 540mgt	24,96 MB/s	22,01 MB/s	105,97 MB/s	986,77 MB/s
E8500 275gtx	5,37 MB/s	57,69 MB/s	207,14 MB/s	629,28 MB/s

Conclusion

„Operating Systems must support GPU abstractions“

- GPU as yet another processor
  - Resource abstraction, scheduling, security
- Transparent data transfer and execution on all types of accelerators