GPGPU introduction and network applications

PacketShaders, SSLShader

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™

Agenda

- GPGPU Introduction
  - Computer graphics background
  - GPGPUs – past, present and future
- PacketShader – A GPU-Accelerated Software Router
- SSLShader – A GPU-Accelerated SSL encryption/decryption proxy

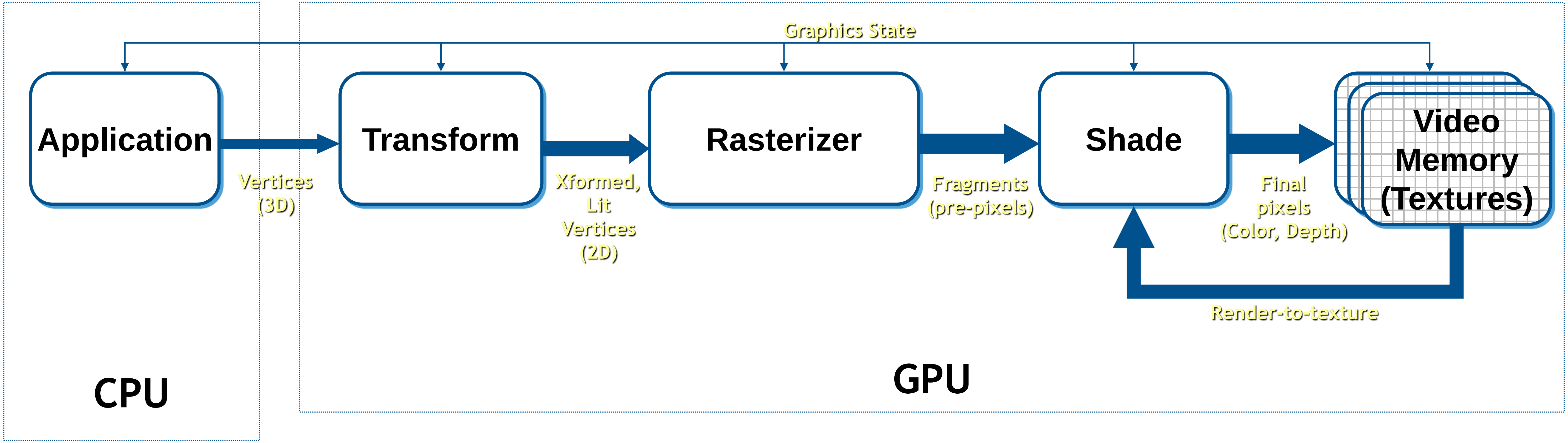
GPGPU Intro

GPU = Graphics Processing Unit

- The heart of graphics cards
- Mainly used for real-time 3D game rendering
  - Massively-parallel processing capacity

(Ubisoft's AVARTAR, from http://ubi.com)

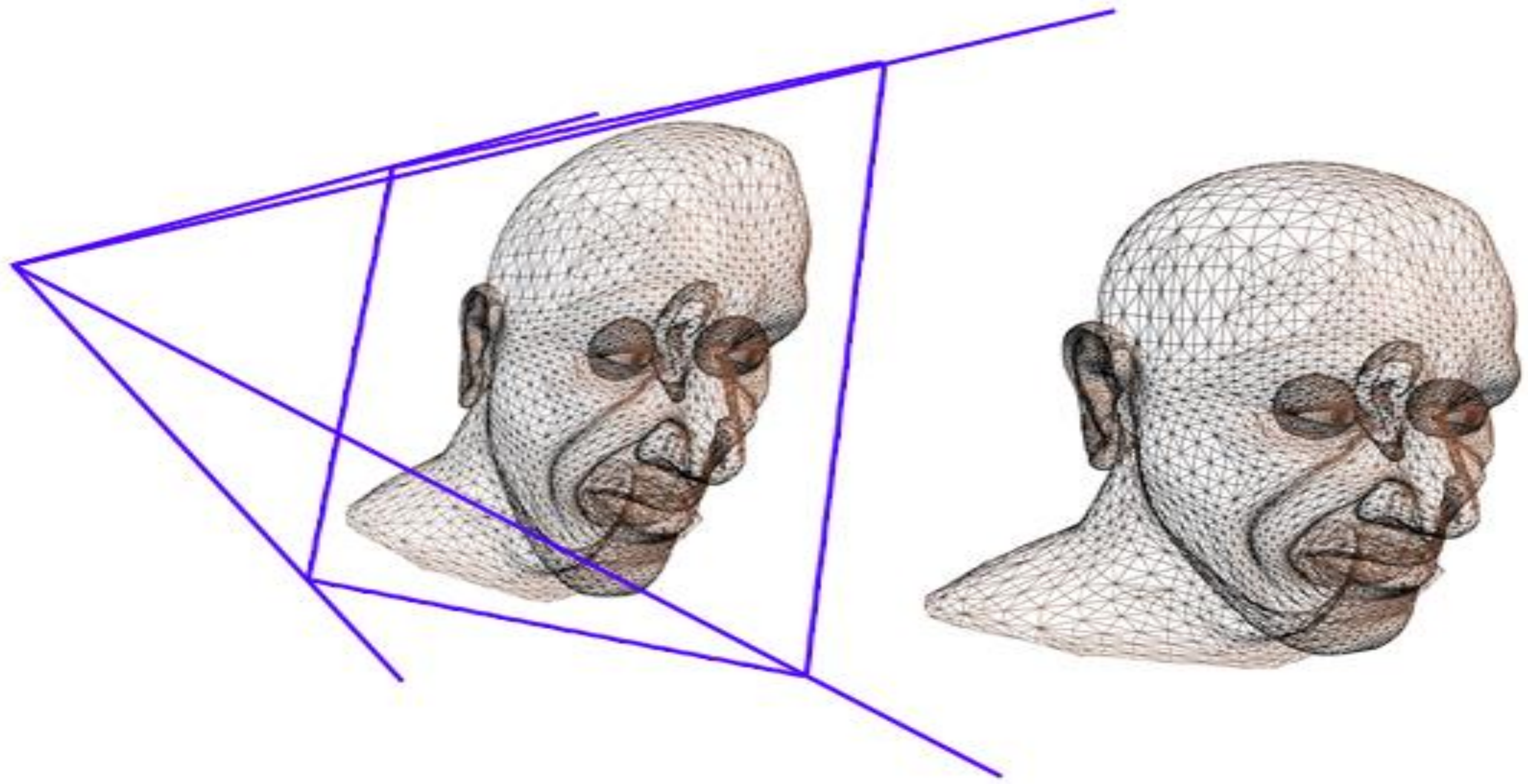
GPU Fundamentals: The Graphics Pipeline



- A simplified graphics pipeline
  - Note that pipe widths vary
  - Many caches, FIFOs, and so on not shown

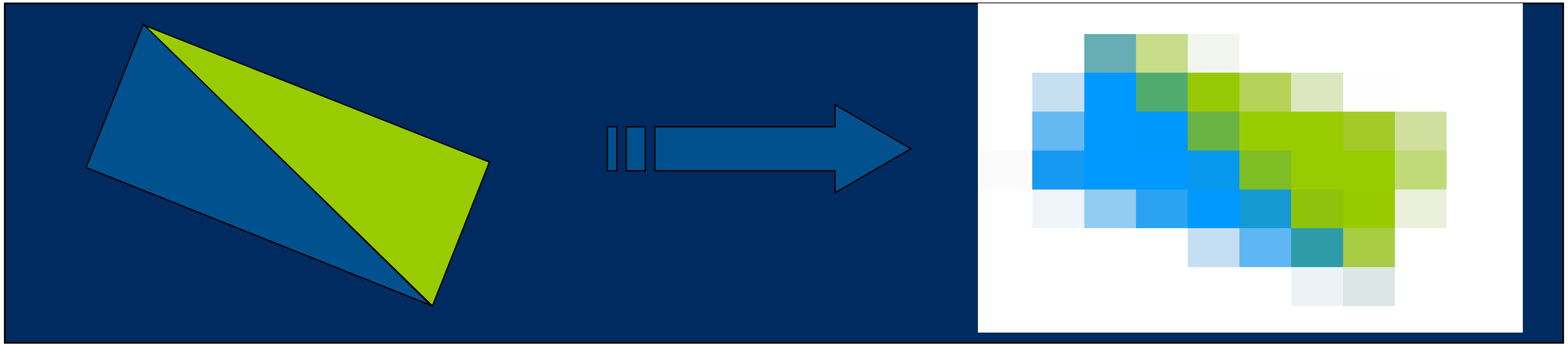
GPU Pipeline: Transform

- Vertex Processor (multiple operate in parallel)
  - Transform from “world space” to “image space”
  - Compute per-vertex lighting



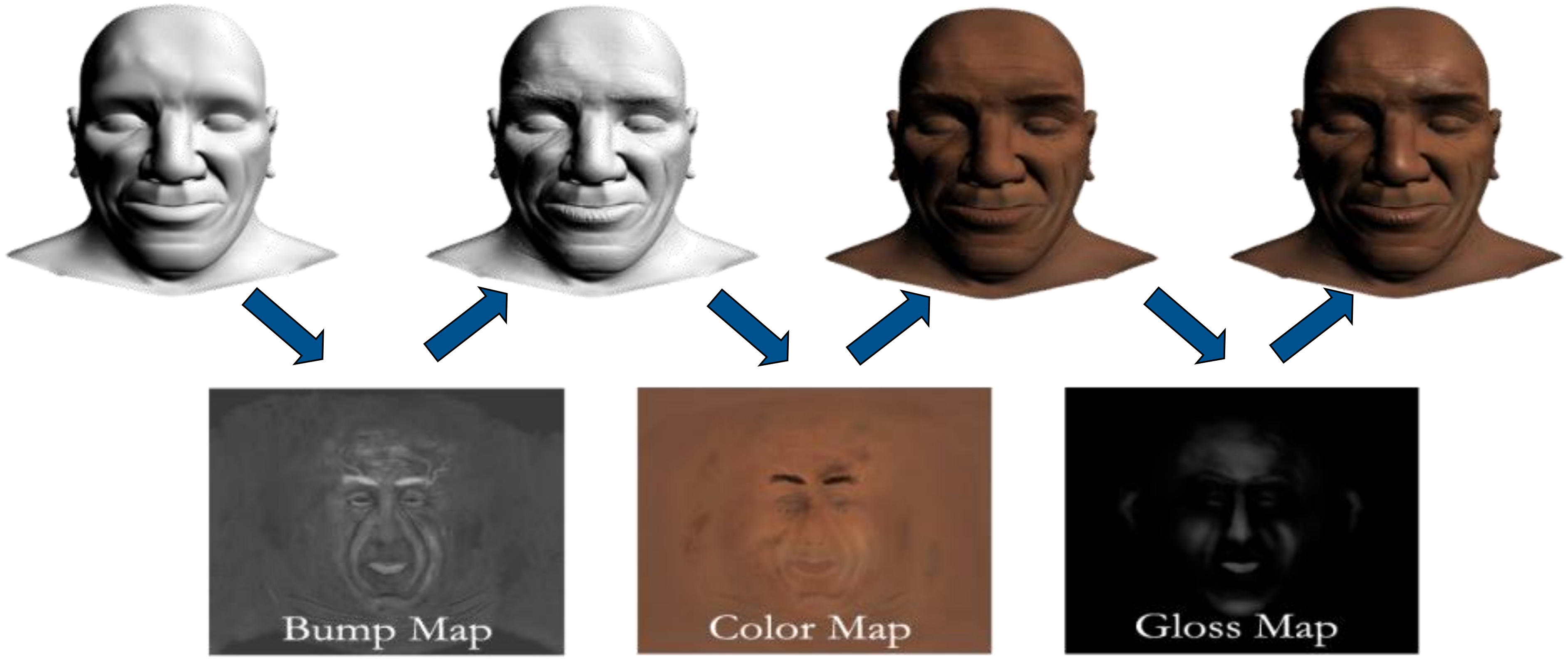
GPU Pipeline: Rasterizer

- Rasterizer
  - Convert geometric rep. (vertex) to image rep. (fragment)
    - Fragment = image fragment
      - Pixel + associated data: color, depth, stencil, etc.
  - Interpolate per-vertex quantities across pixels

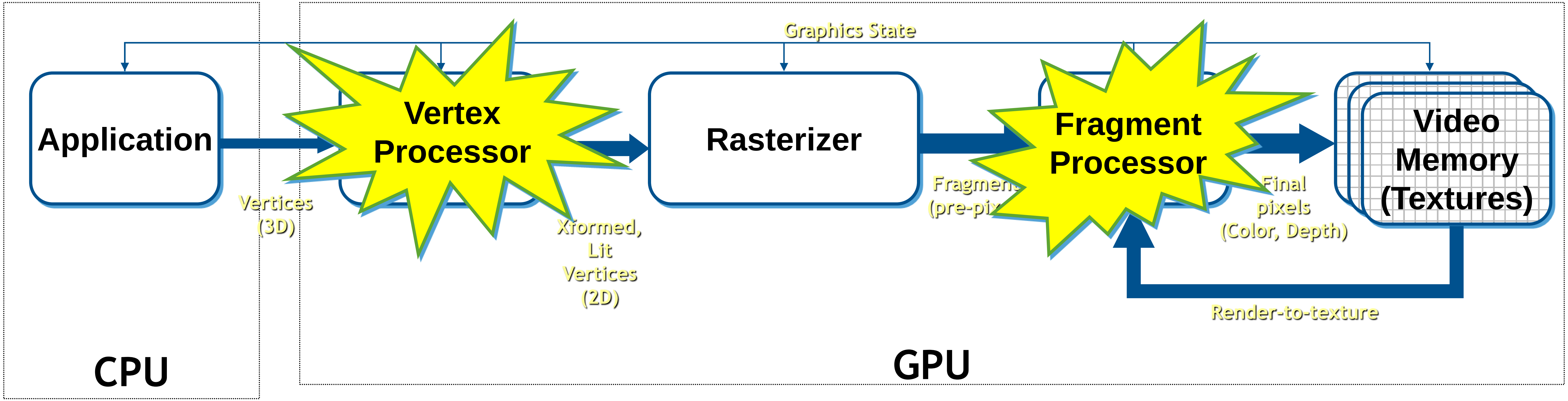


GPU Pipeline: Shade

- Fragment Processors (multiple in parallel)
  - Compute a color for each pixel
  - Optionally read colors from textures (images)



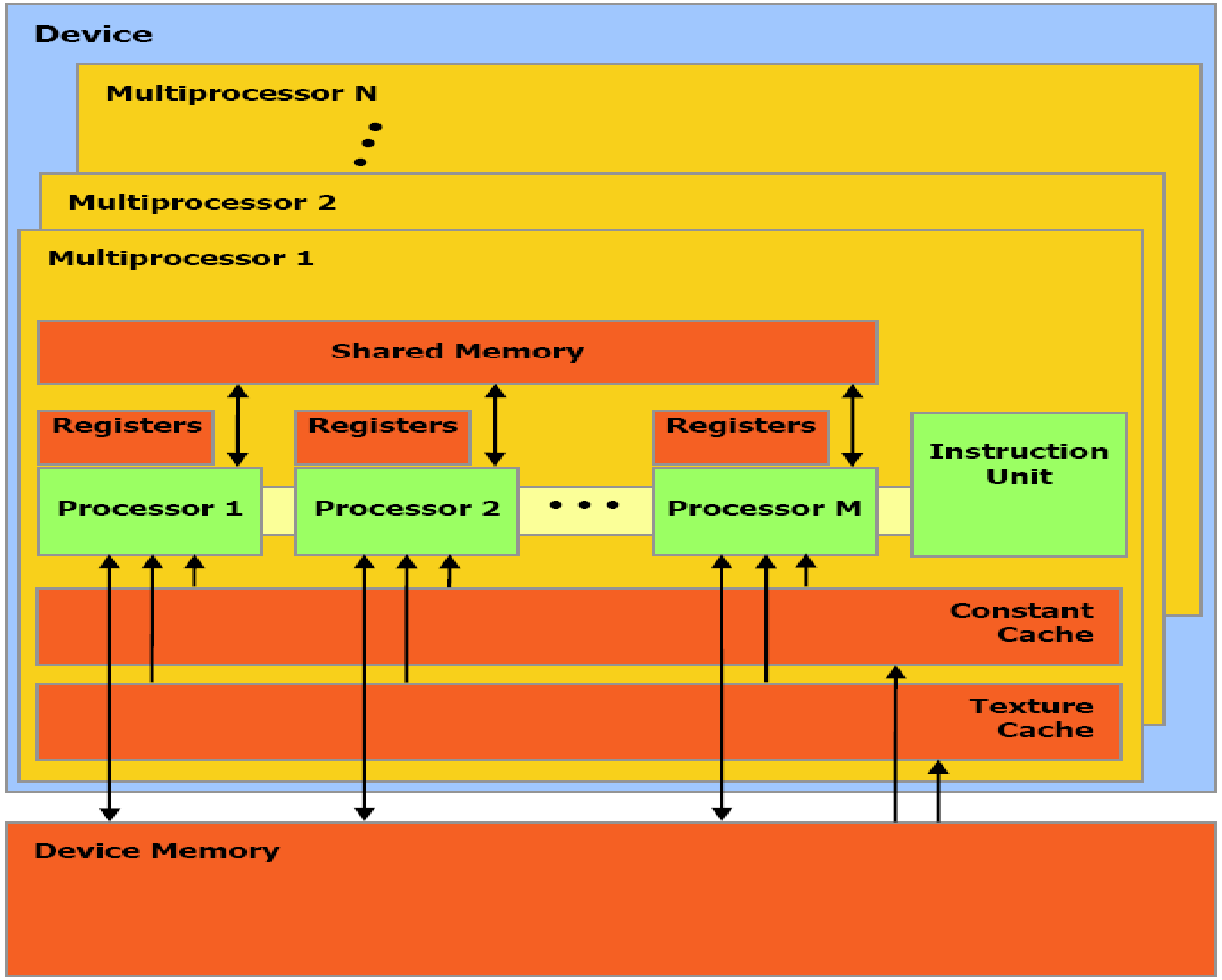
GPU Fundamentals: The *Modern* Graphics Pipeline



- Programmable vertex processor!
- Programmable pixel processor!

nVidia G80 GPU Architecture Overview

- 16 Multiprocessors Blocks
- Each MP Block Has:
  - 8 Streaming Processors (IEEE 754 spfp compliant)
  - 16K Shared Memory
  - 64K Constant Cache
  - 8K Texture Cache
- Each processor can access all of the memory at 86Gb/s, but with different latencies:
- Shared – 2 cycle latency
- Device – 300 cycle latency



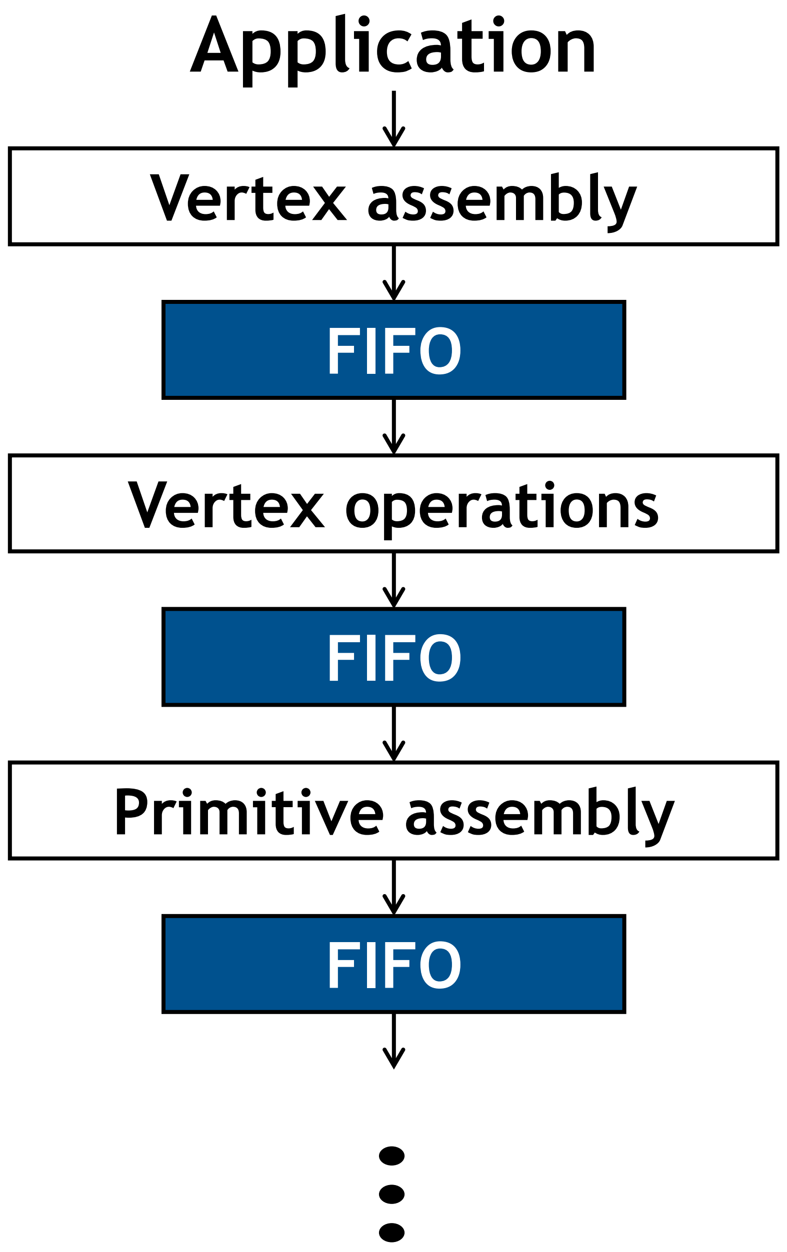
Queueing

FIFO buffering (first-in, first-out) is provided between task stages

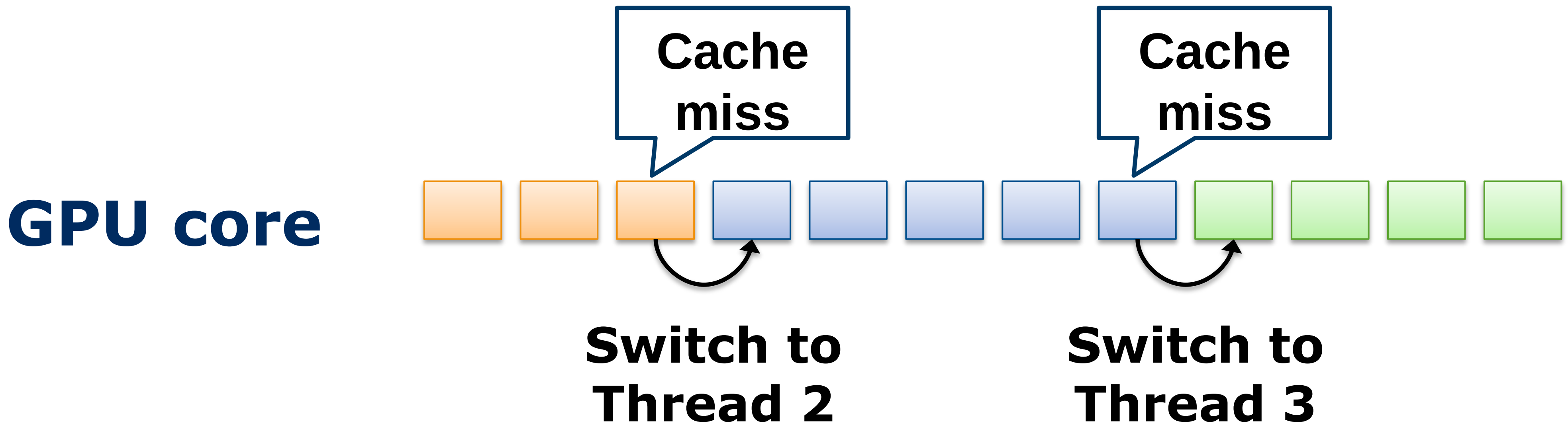
- Accommodates variation in execution time
- Provides elasticity to allow unified load balancing to work

FIFOs can also be unified

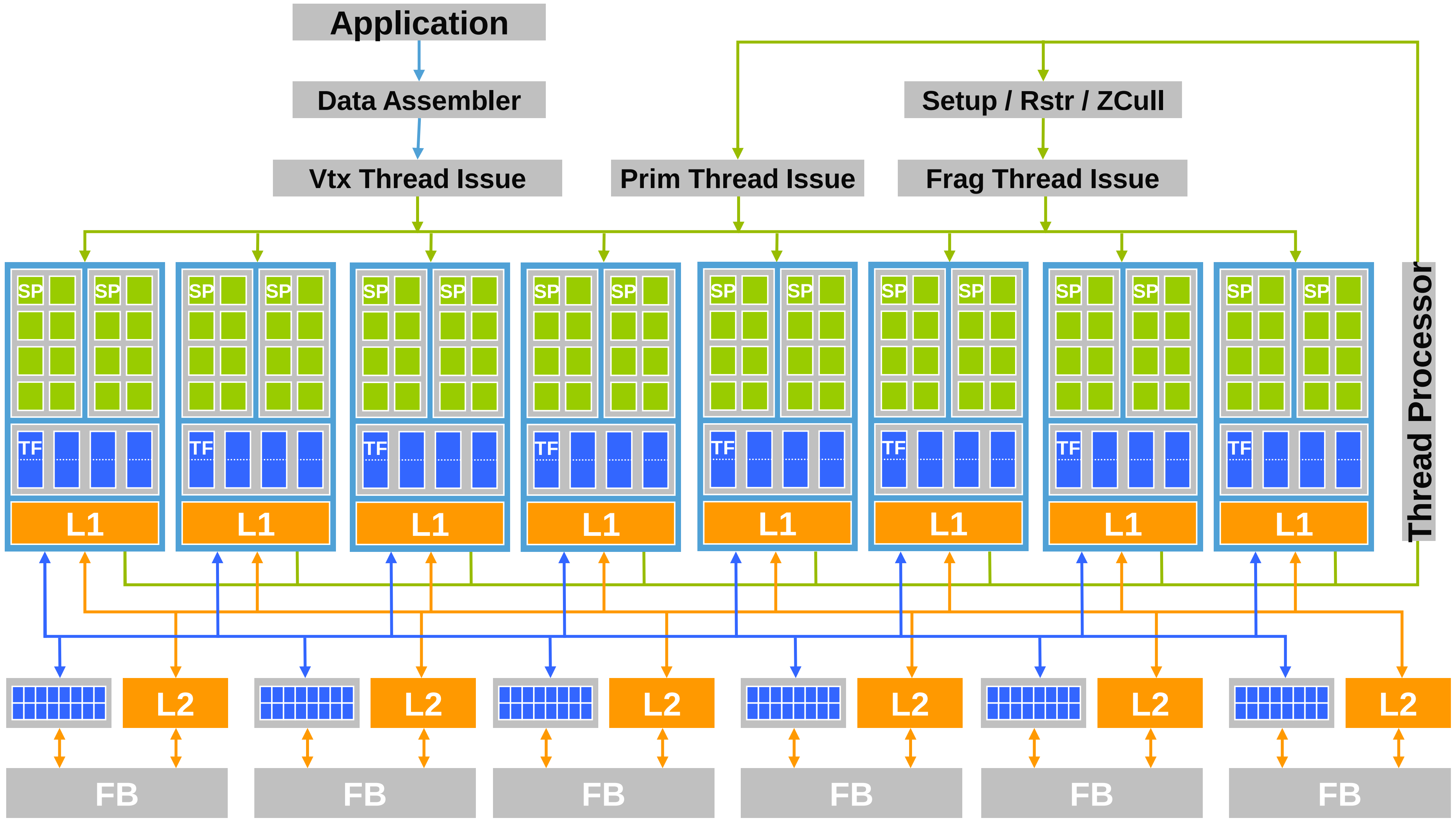
- Share a single large memory with multiple head-tail pairs
- Allocate as required

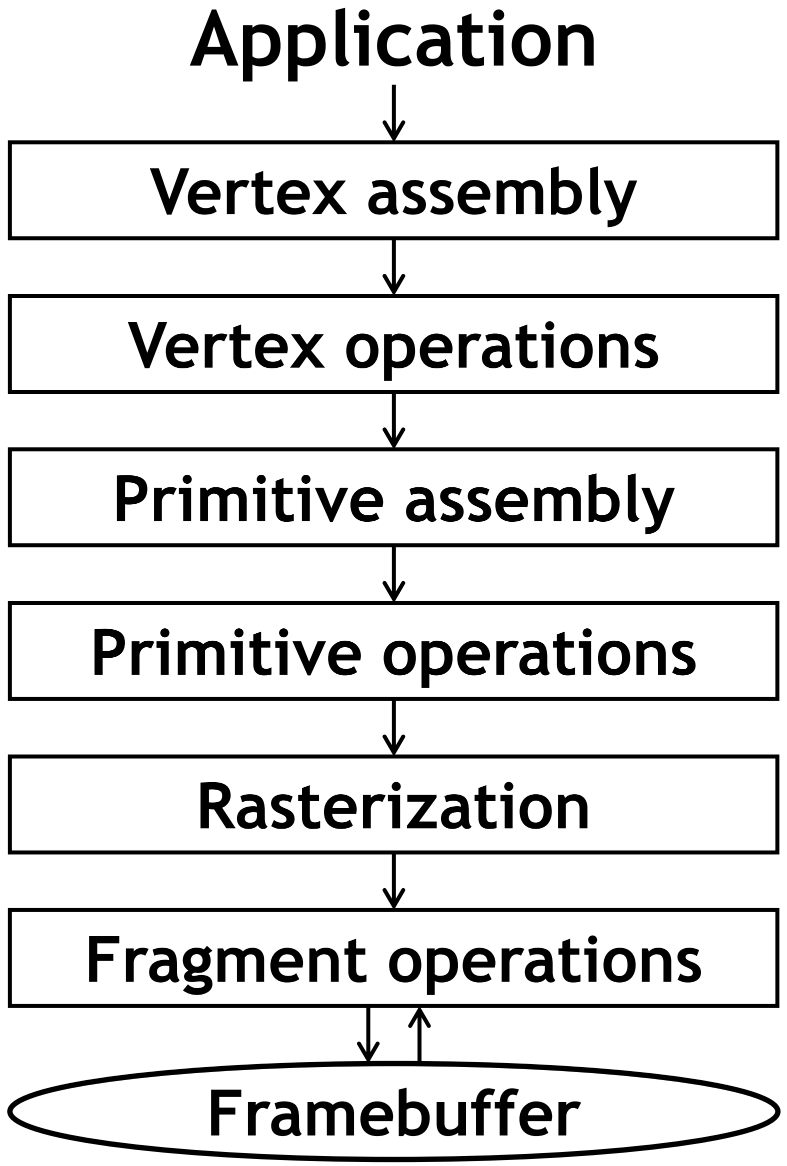


- GPU can effectively hide memory latency



Implementation vs. architecture model



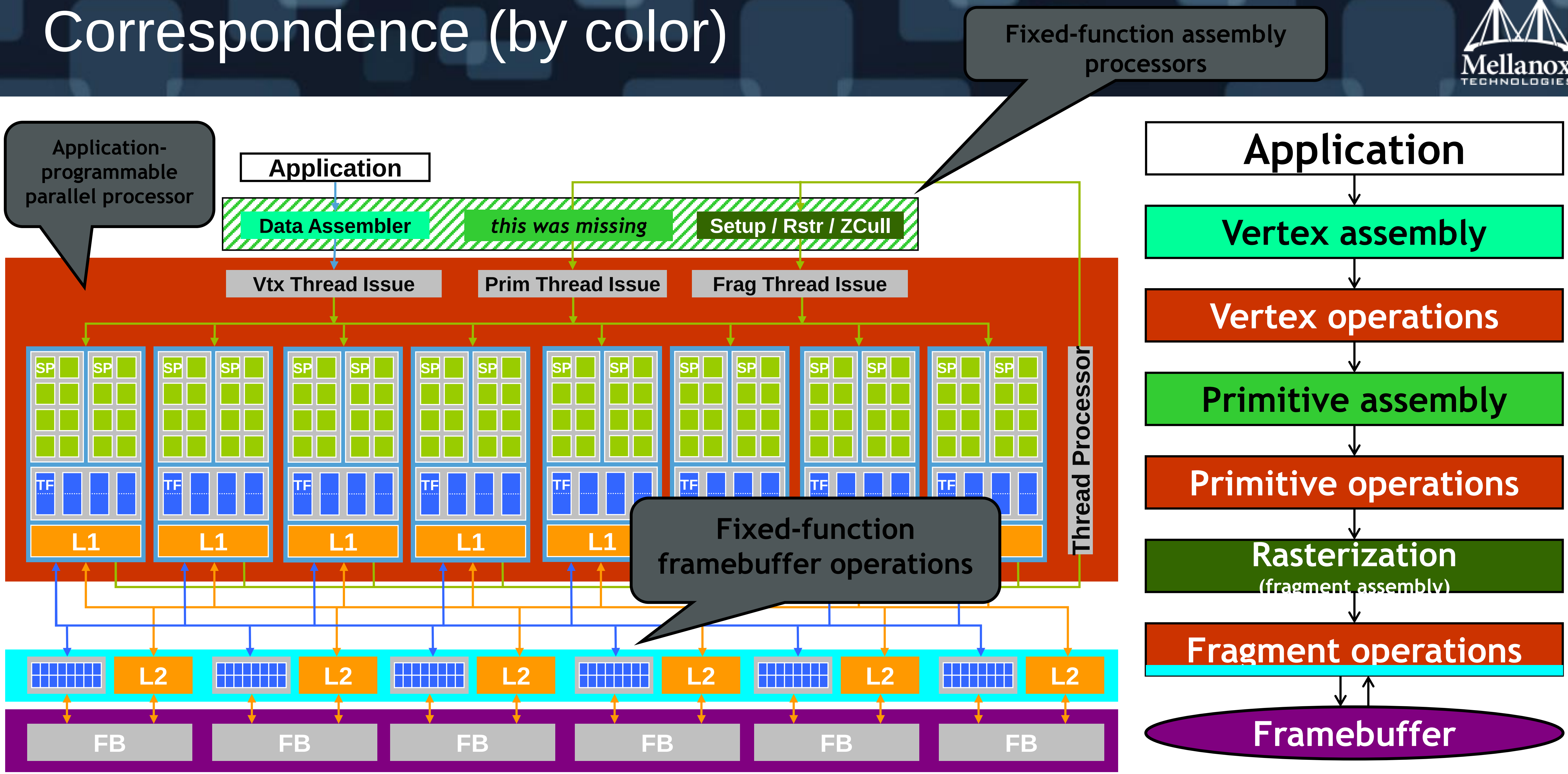


NVIDIA GeForce 8800

OpenGL Pipeline

Source : NVIDIA

Correspondence (by color)

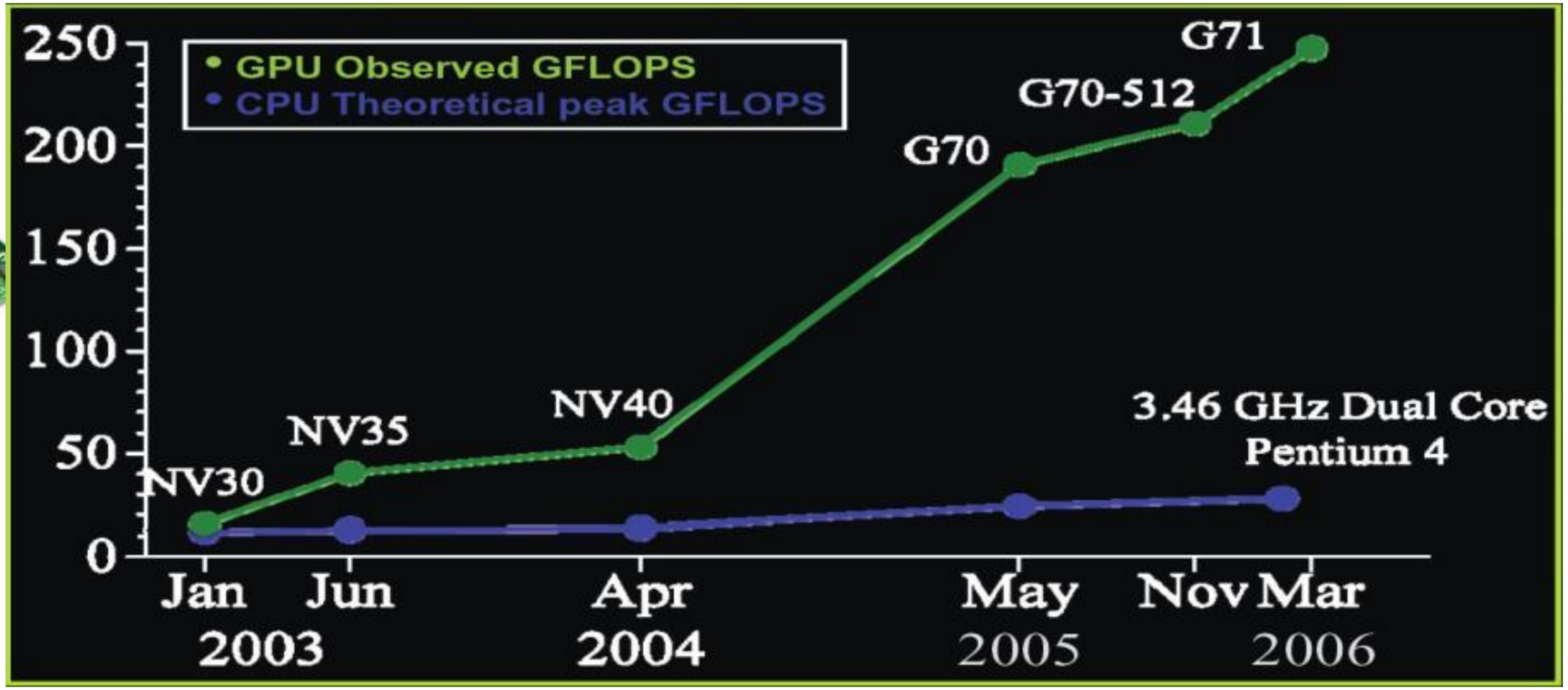


NVIDIA GeForce 8800

OpenGL Pipeline

The nVidia G80 GPU

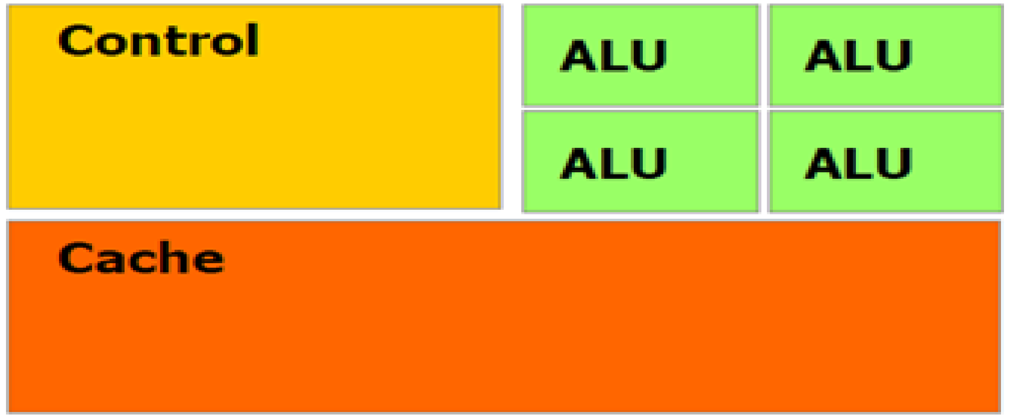
- 128 streaming floating point processors @1.5Ghz
- 1.5 Gb Shared RAM with 86Gb/s bandwidth
- 500 Gflop on one chip (single precision)



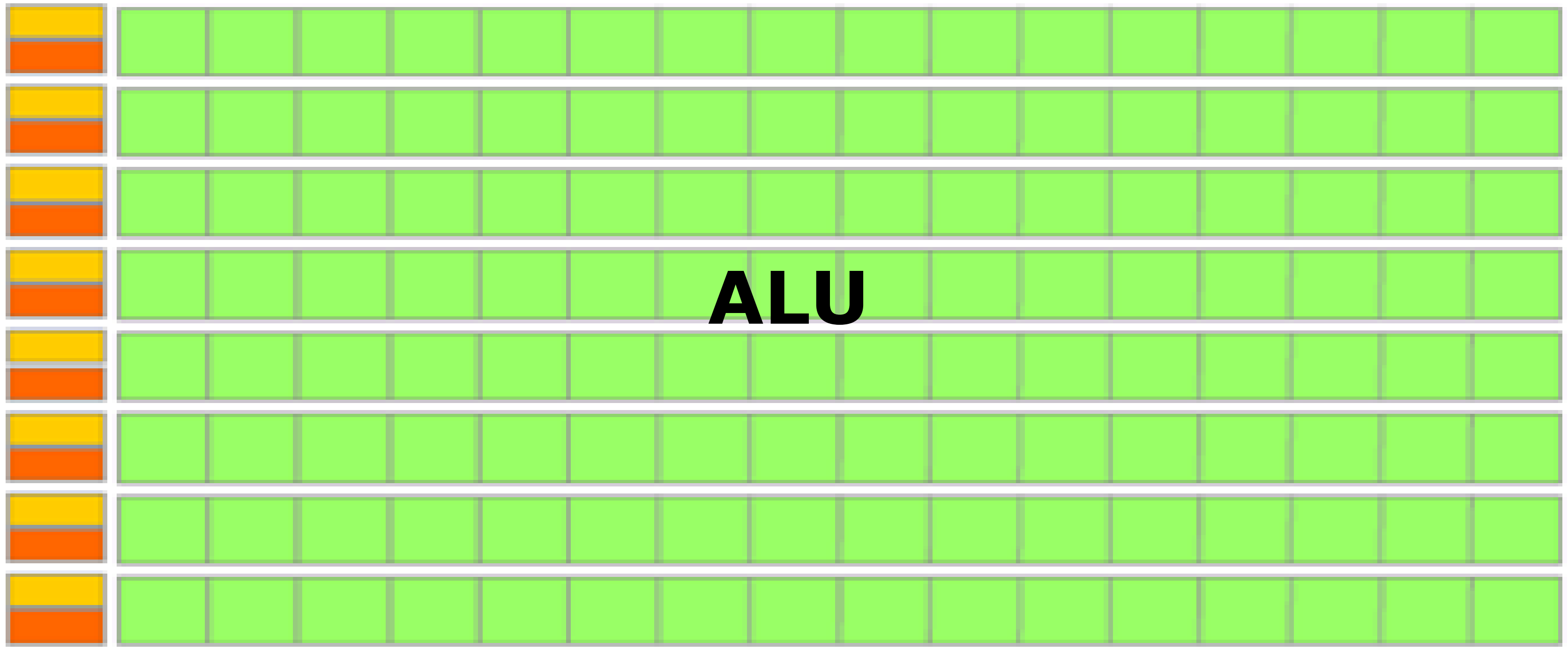
Why are GPU’s so fast?

- Entertainment Industry has driven the economy of these chips?
  - Males age 15-35 buy $10B in video games / year
- Moore’s Law ++
- Simplified design (stream processing)
  - Huge parallelism – maps well to hardware
  - Latency hiding using the parallelism
- Single-chip designs.

“Silicon Budget” in CPU and GPU



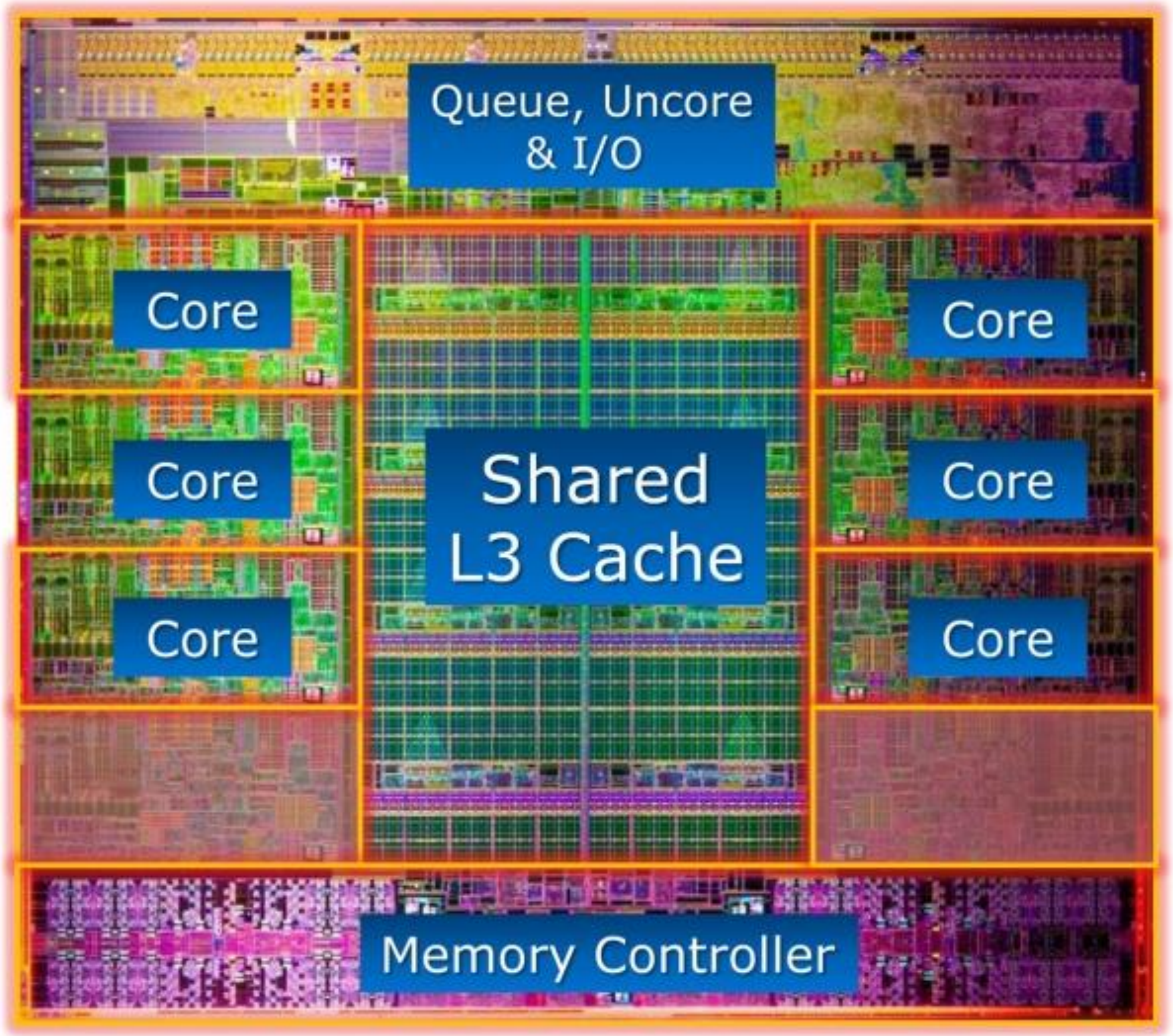
Xeon X5550:
4 cores
731M transistors



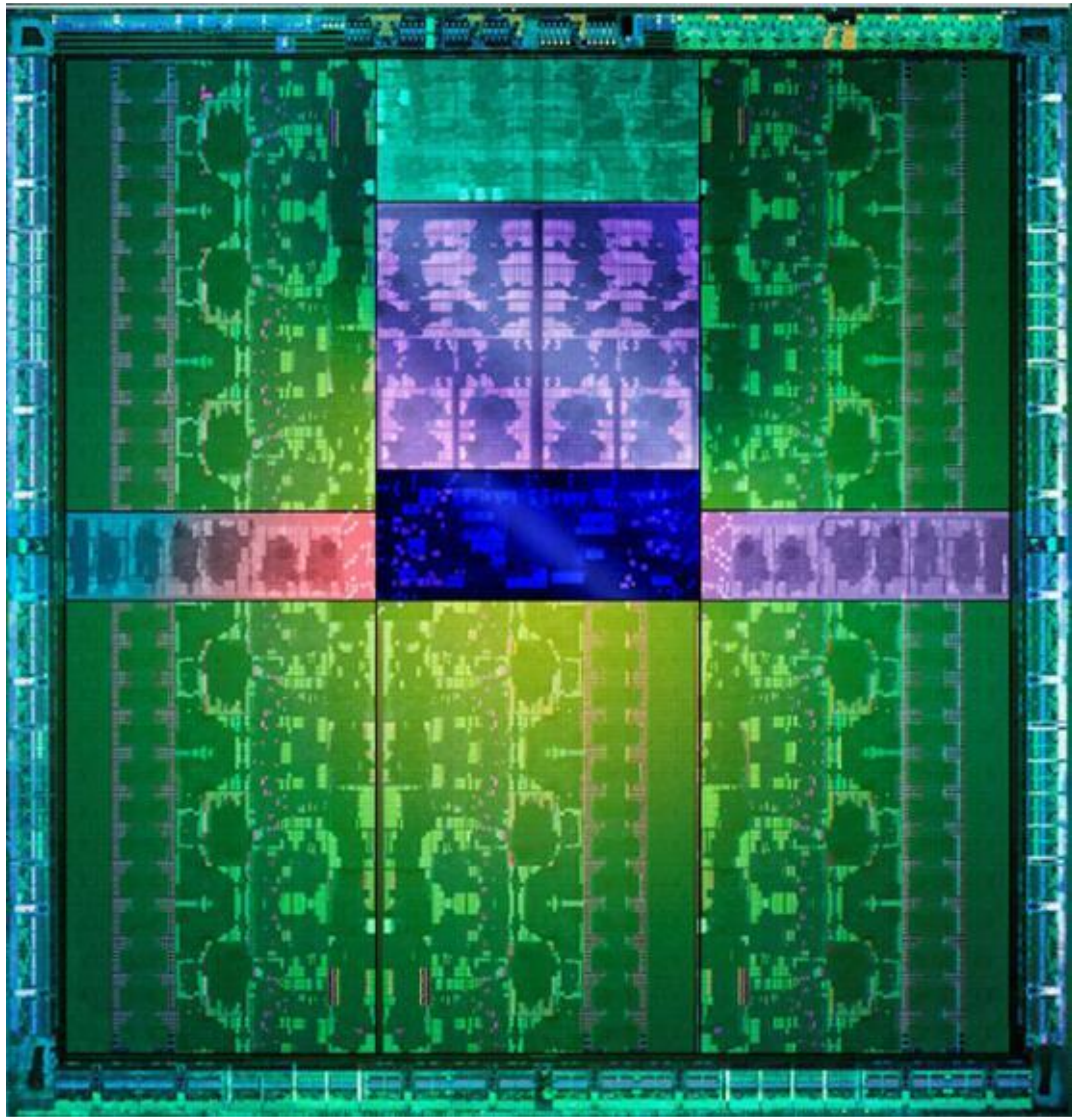
GTX480:
480 cores
3,200M transistors

19

Floorplans comparison



CPU - Core i7



GPU – nVidia Kepler

GPU - A Specialized Processor

- **Very Efficient For**
  - Fast Parallel Floating Point Processing
  - Single Instruction Multiple Data Operations
  - High Computation per Memory Access

- **Not As Efficient For**
  - Double Precision – situation is improving
  - Logical Operations on Integer Data
  - Branching-Intensive Operations
  - Random Access, Memory-Intensive Operations

GPGPU!

- Programable stream processor
  - Huge number of ALUs
  - Huge memory bandwidth
- Programming was painful
  - OpenGL-SL – **Shader Language**
  - Requires deep understanding of computers graphics
  - Huge applications speedup when done correctly
- CUDA/OpenCL
  - C-like code
  - Massively multi-threaded
  - Simple to port existing code (but not to get good performance)

CUDA – Single Instruction Multiple Threads

Example code: vector addition (C = A + B)

CPU code

```
void VecAdd(
int *A, int *B, int *C, int N)
{
    //iterate over N elements
    for(int i = 0; i < N; i++)
        C[i] = A[i] + B[i]
}

VecAdd(A, B, C, N);
```

GPU code

```
__global__ void VecAdd(
int *A, int *B, int *C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i]
}

//Launch N threads
VecAdd<<<1, N>>>(A, B, C);
```

Achieving Performance in CUDA

- Almost all C code will compile to be CUDA code
  - But will run slower
  - Single threaded operation - ~50x slower than CPU code
- Must expose parallelism
- Careful with memory accesses
  - Thread scheduling helps hide memory access latency
  - But even this runs out
- Moving target
  - Performance optimizations are strongly HW and SW platform dependent
- Can make huge difference
  - 100x and even more

PacketShader

A GPU-Accelerated Software Router

High Performance Software Router

- Work by Sangjin Han, Keon Jang, KyoungSoo Park and Sue Moon
  - Advanced Networking Lab, CS, KAIST
  - Networked and Distributed Computing Systems Lab, EE, KAIST
- 40 Gbps packet forwarding in a single box
  - IPv4, 64B packets
  - Bigger packet sizes – bounded by PCI-e bandwidth
- 20 Gbps IPsec tunneling
  - For 1024B packets
  - 10 Gbps for 64B packets

Software Router

- Despite its name, not limited to IP routing
  - You can implement whatever you want on it.
- Driven by software
  - Flexible
  - Friendly development environments
- Based on commodity hardware
  - Cheap
  - Fast evolution

Now 10 Gigabit NIC is a commodity

- From \$200 – \$300 per port
  - Great opportunity for software routers

amazon.com

Mellanox ConnectX-3 Gigabit Ethernet Card

~~$638.00~~ $373.98

In Stock

Achilles' Heel of Software Routers

- Low performance
  - Due to CPU bottleneck

Year	Ref.	H/W	IPv4 Throughput
2008	Egi et al.	Two quad-core CPUs	3.5 Gbps
2008	"Enhanced SR" Bolla et al.	Two quad-core CPUs	4.2 Gbps
2009	"RouteBricks" Dobrescu et al.	Two quad-core CPUs (2.8 GHz)	8.7 Gbps

- **Not capable of supporting even a single 10G port**

Per-Packet CPU Cycles for 10G

Cycles needed

IPv4	1,200 (Packet I/O)	+	600 (IPv4 lookup)	= 1,800 cycles
IPv6	1,200 (Packet I/O)	+	1,600 (IPv6 lookup)	= 2,800
IPsec	1,200 (Packet I/O)	+	5,400 … (Encryption and hashing)	= 6,600

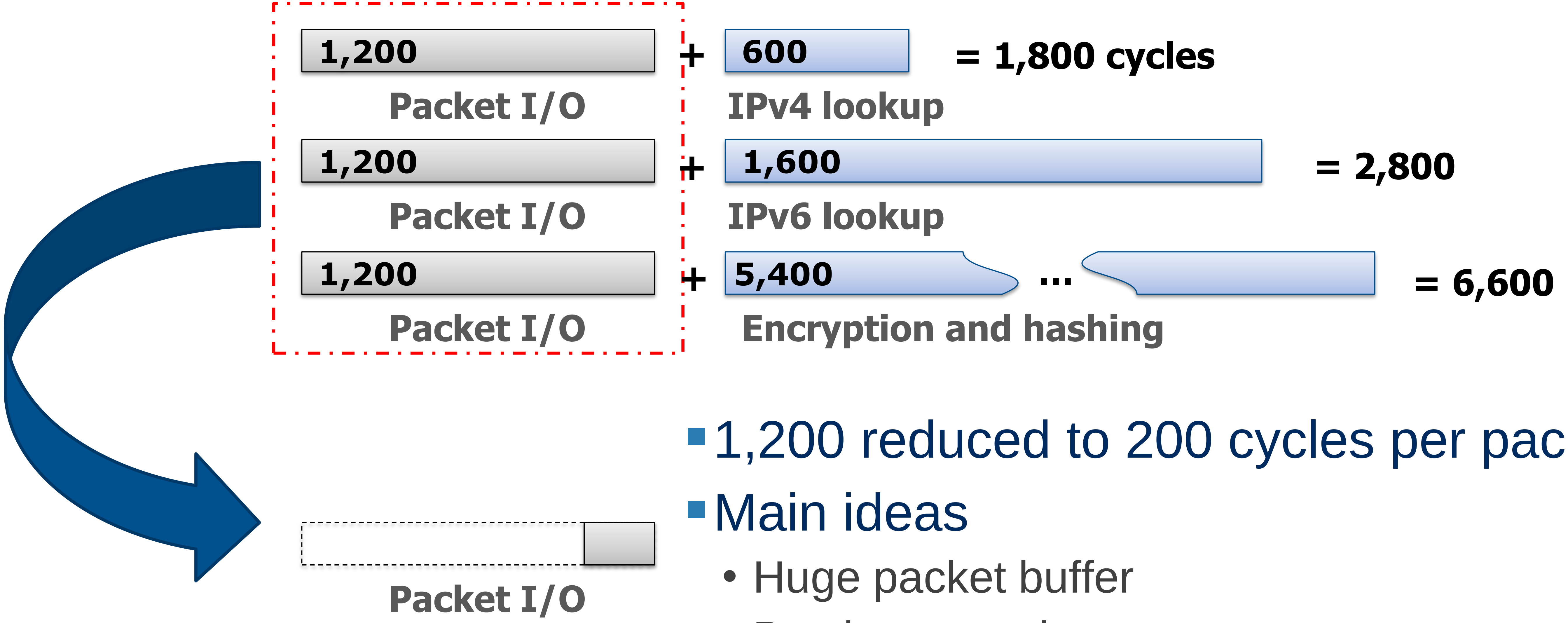
Your budget

1,400 cycles

10G, min-sized packets, dual quad-core 2.66GHz CPUs

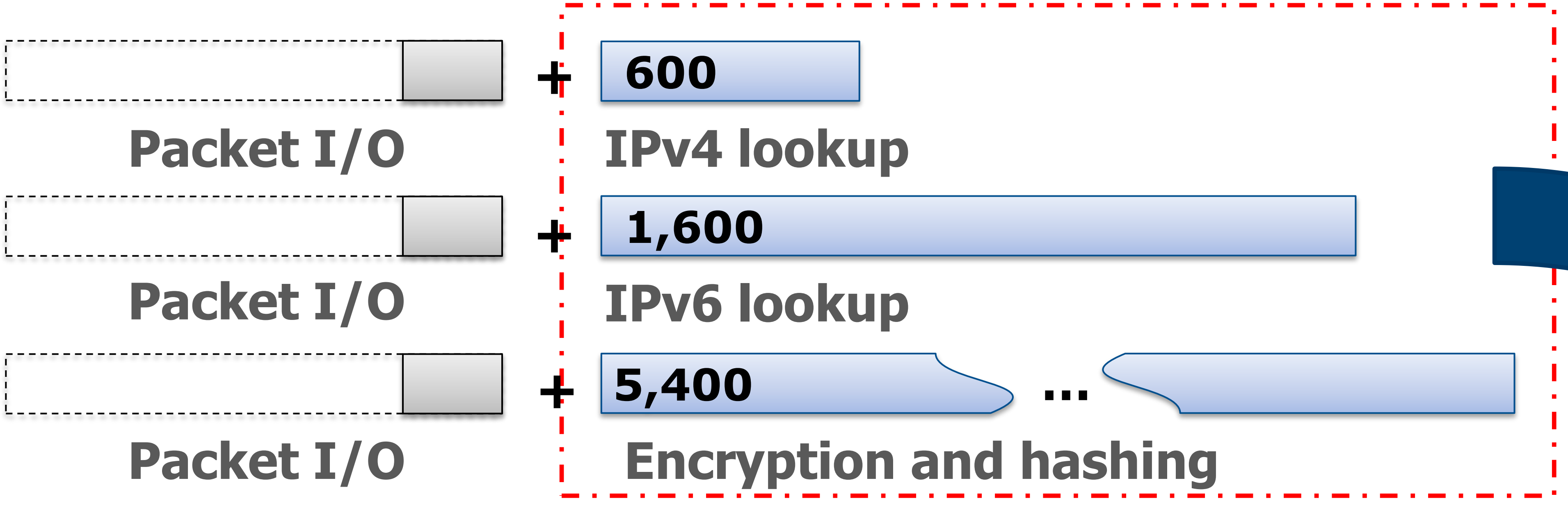
(in x86, cycle numbers are from RouteBricks [Dobrescu09] and PacketShader)

PacketShader Approach 1: I/O Optimization

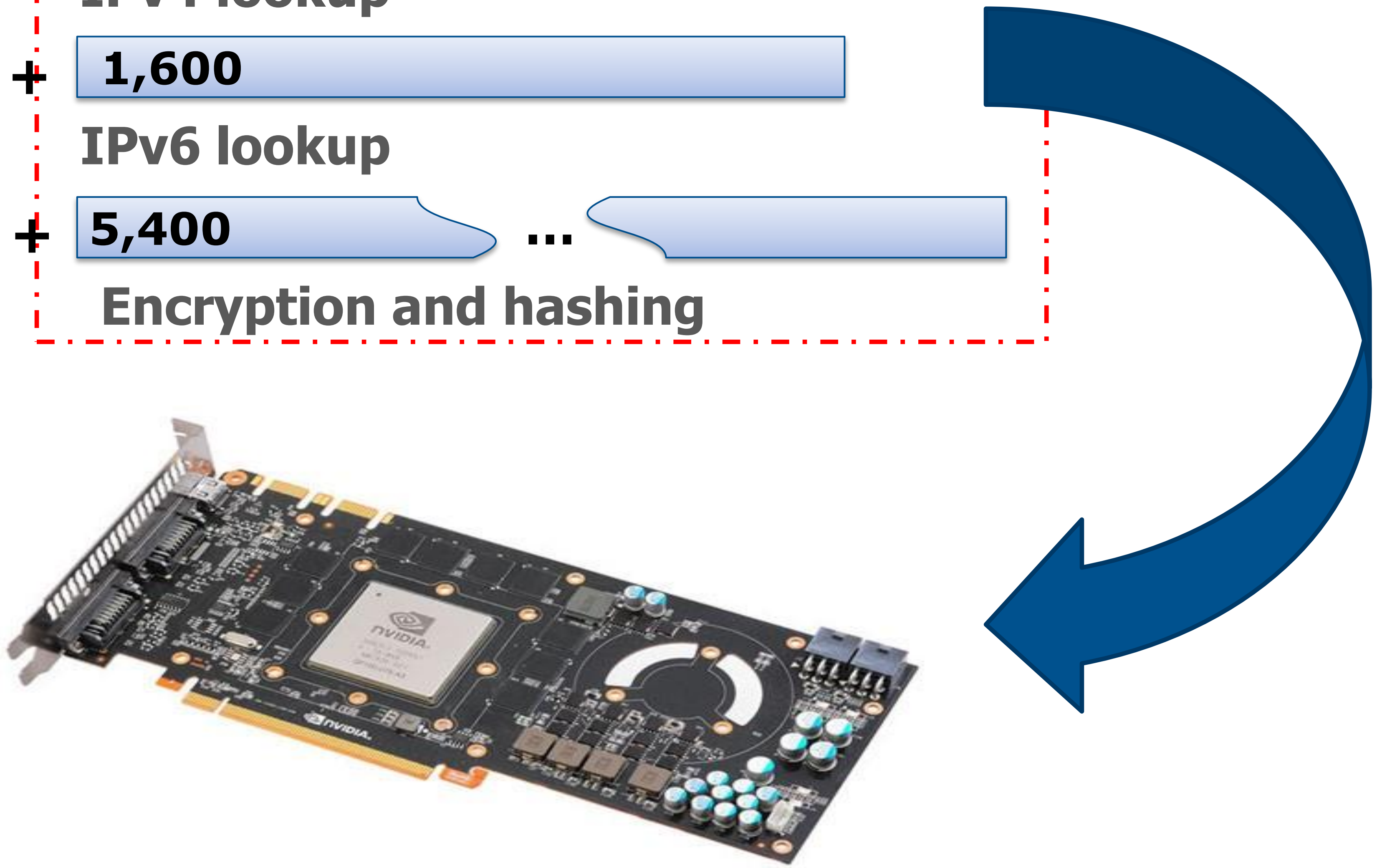


- 1,200 reduced to 200 cycles per packet
- Main ideas
  - Huge packet buffer
  - Batch processing
- Allocating SKBs – 50% of CPU time

PacketShader Approach 2: GPU Offloading



- GPU Offloading for
  - Memory-intensive or
  - Compute-intensive operations
- Main topic of this talk



GPU FOR PACKET PROCESSING

Advantages of GPU for Packet Processing

1. Raw computation power

2. Memory access latency

3. Memory bandwidth

- Comparison between
  - Intel X5550 CPU
  - NVIDIA GTX480 GPU

(1/3) Raw Computation Power

- Compute-intensive operations in software routers
  - Hashing, encryption, pattern matching, network coding, compression, etc.
  - GPU can help!

Instructions/sec

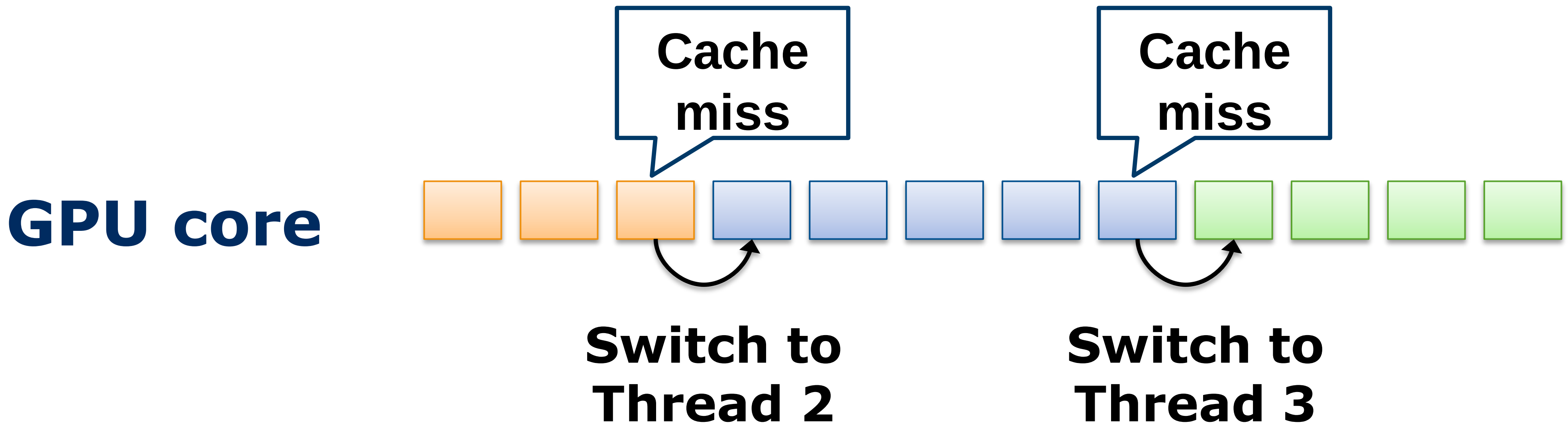
CPU: 43 × 10^9
= 2.66 (GHz) ×
4 (# of cores) ×
4 (4-way superscalar)

<

GPU: 672 × 10^9
= 1.4 (GHz) ×
480 (# of cores)

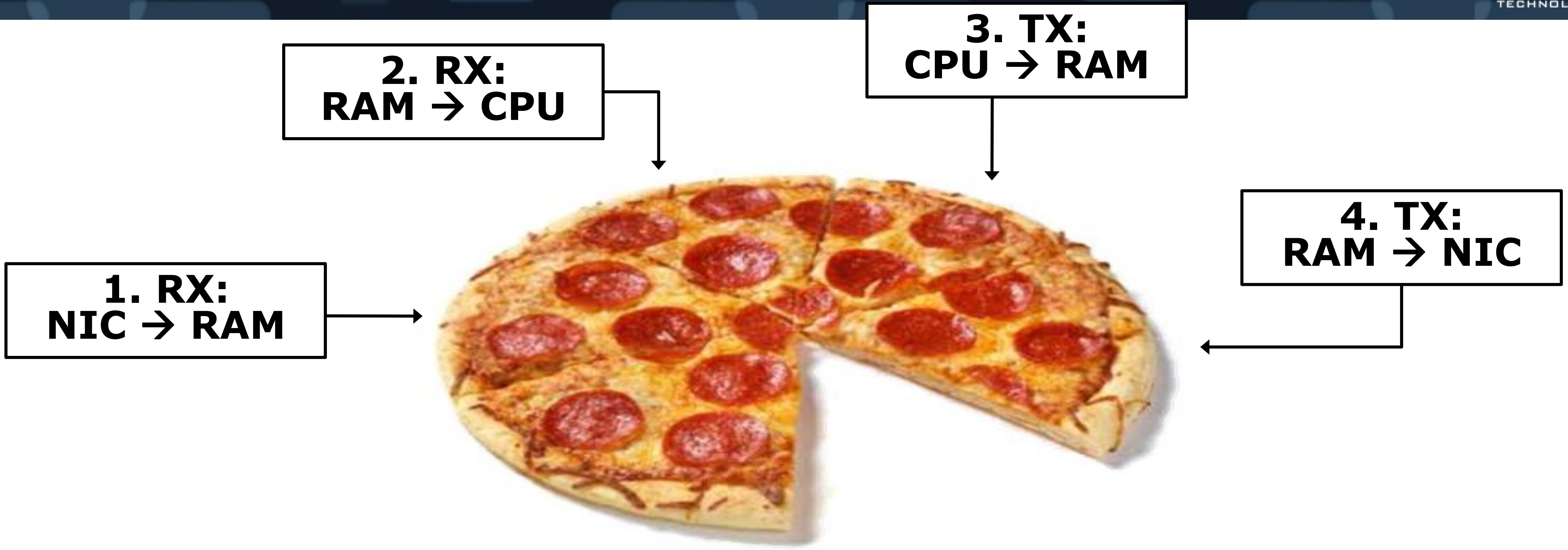
(2/3) Memory Access Latency

- Software router → lots of cache misses
  - GPU can effectively hide memory latency



(3/3) Memory Bandwidth

CPU’s memory bandwidth (theoretical): 32 GB/s



CPU’s memory bandwidth (<u>empirical</u>) < 25 GB/s

(3/3) Memory Bandwidth

Your budget for packet processing can be less 10 GB/s

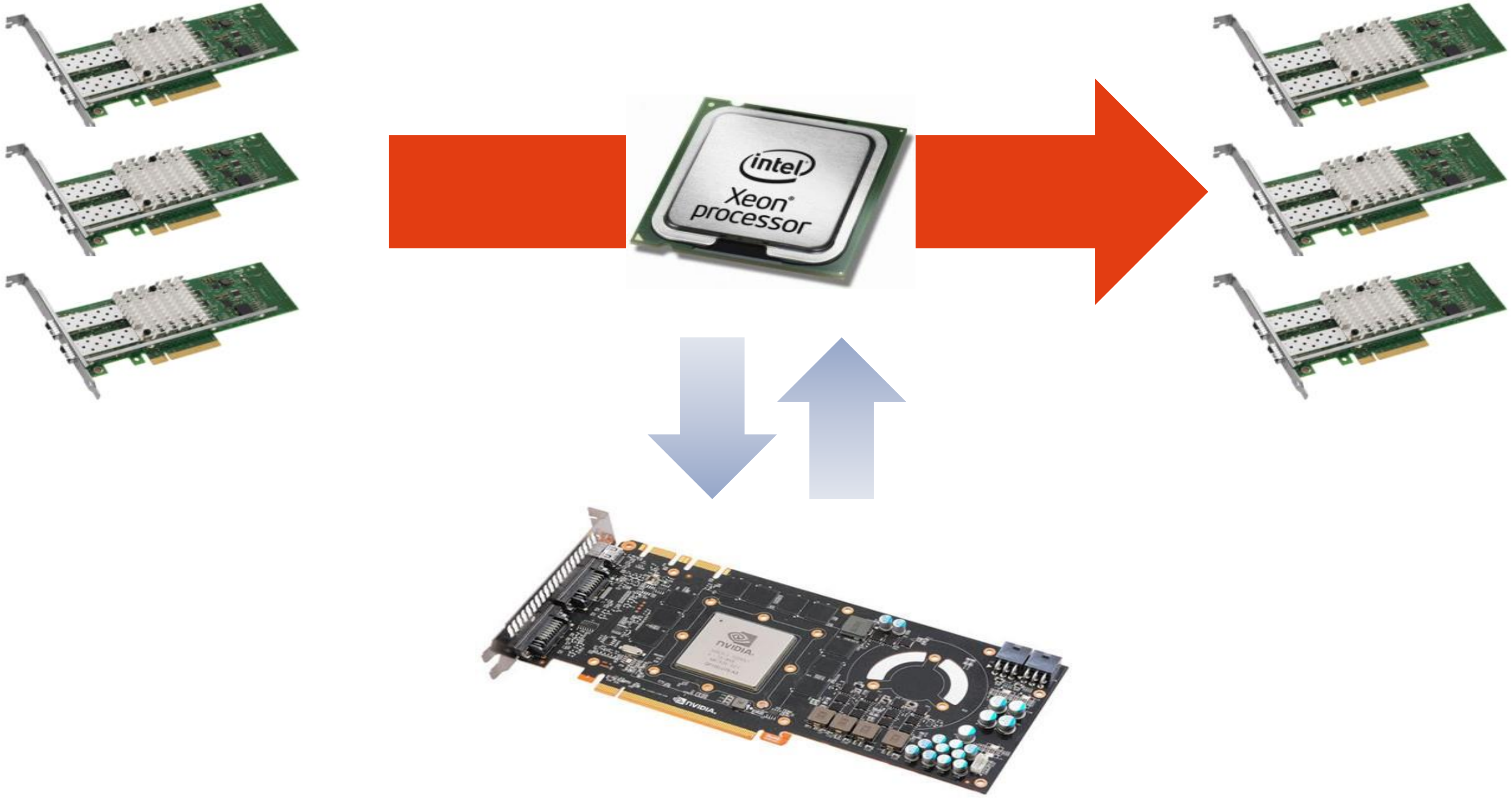
(3/3) Memory Bandwidth

~~**Your budget for packet processing can be less 10 GB/s**~~

GPU’s memory bandwidth: 174GB/s

HOW TO USE GPU

Basic Idea

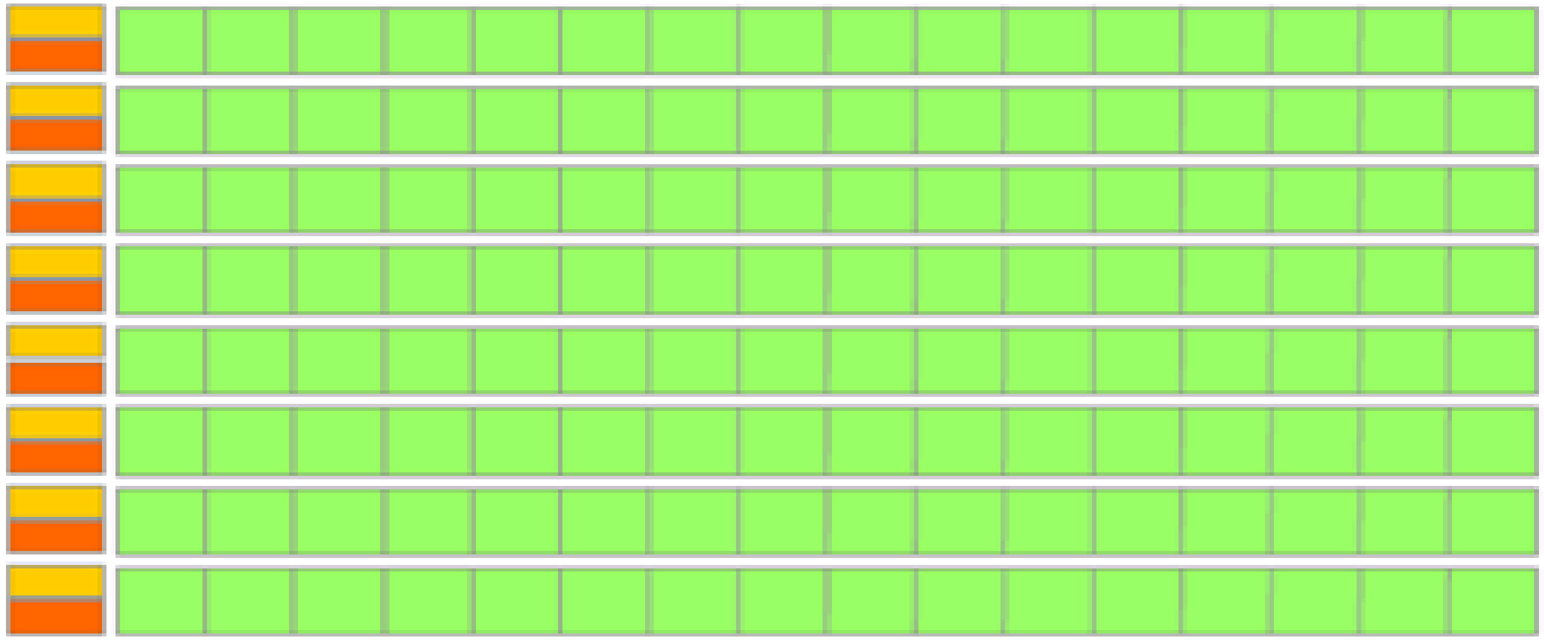


Offload core operations to GPU

(e.g., forwarding table lookup)

Recap

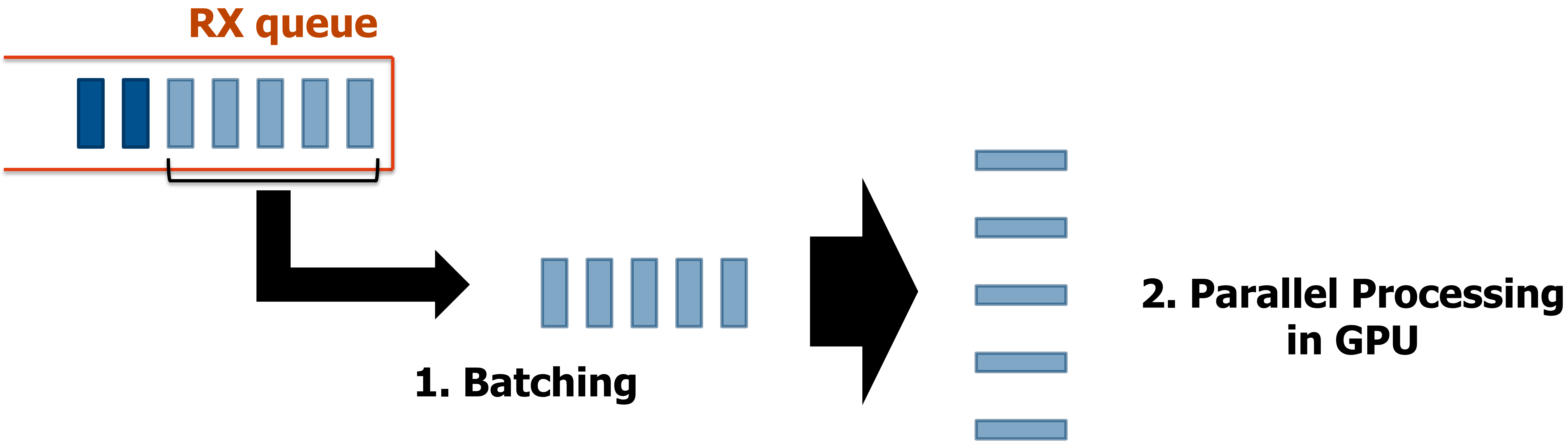
- For GPU, more parallelism, more throughput



GTX480: 480 cores

Parallelism in Packet Processing

- The key insight
  - Stateless packet processing = parallelizable



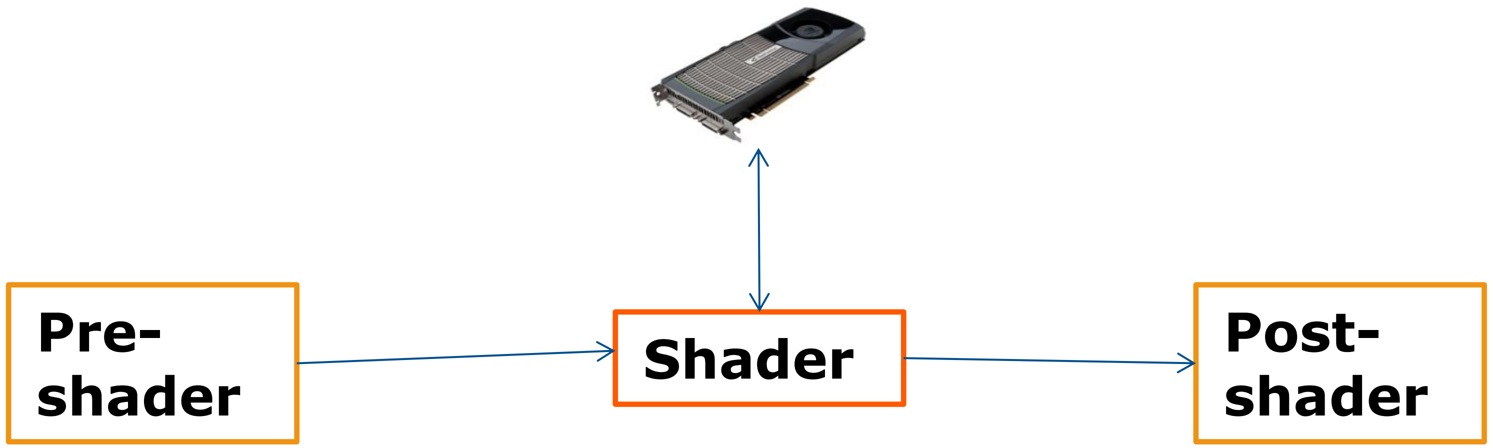
Batching → Long Latency?

- Fast link = enough # of packets in a small time window
- 10 GbE link
  - up to 1,000 packets only in 67μs
- Much less time with 40 or 100 GbE

PACKETSHADER DESIGN

Basic Design

- Three stages in a streamline



- IPv4 forwarding example

- Checksum, TTL
- Format check
- …

Collected
dst. IP addrs

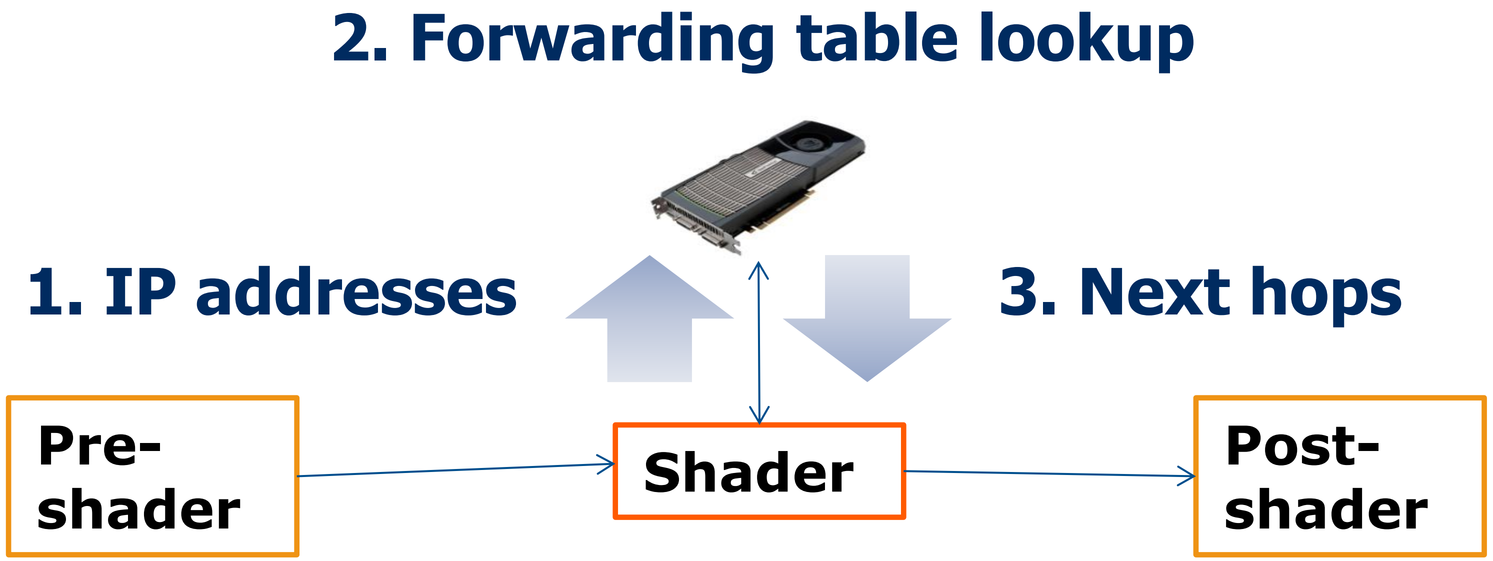
Pre-
shader

Shader

Post-
shader

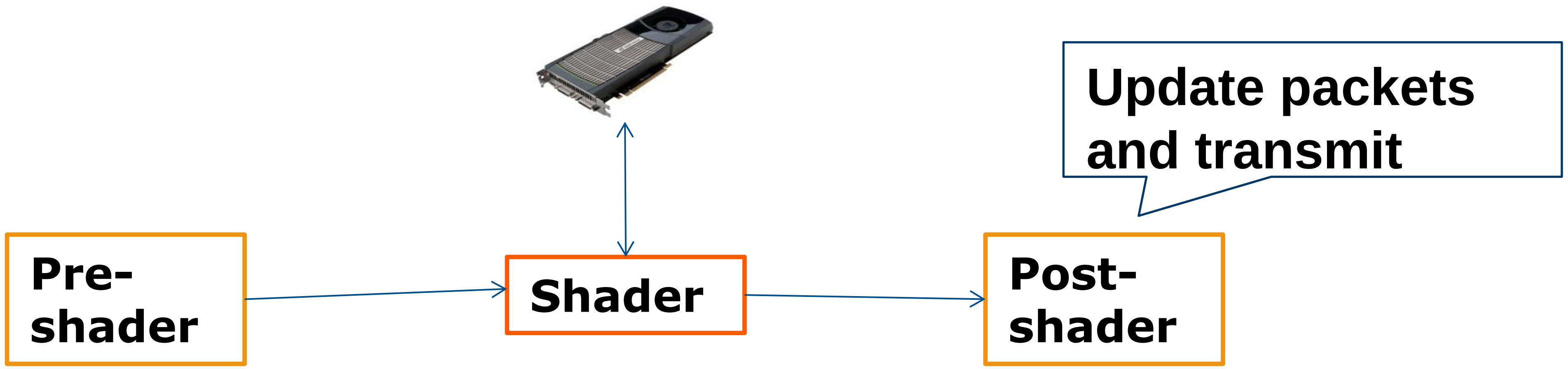
Some packets
go to slow-path

- IPv4 forwarding example

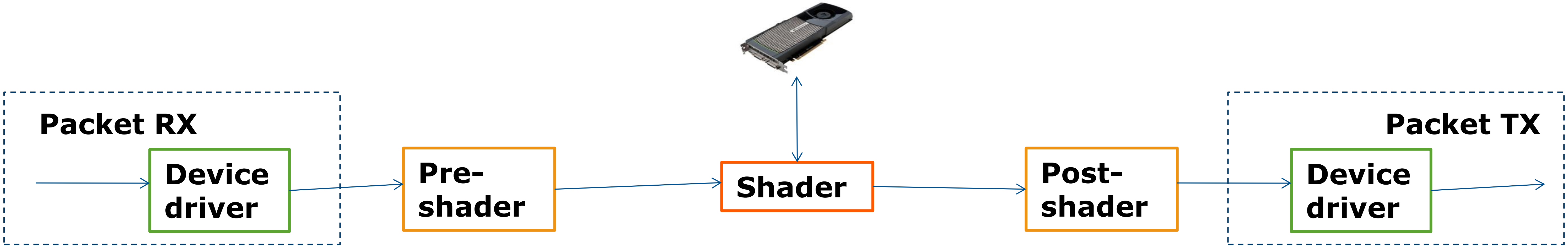


Packet’s Journey (3/3)

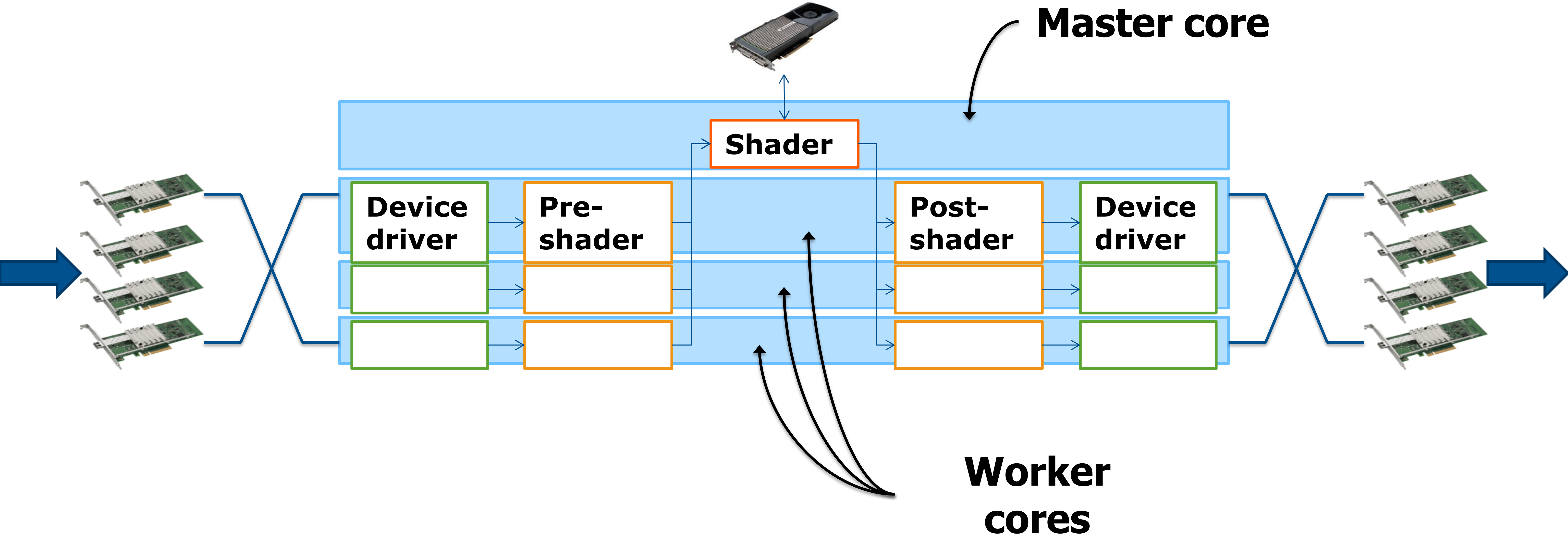
- IPv4 forwarding example



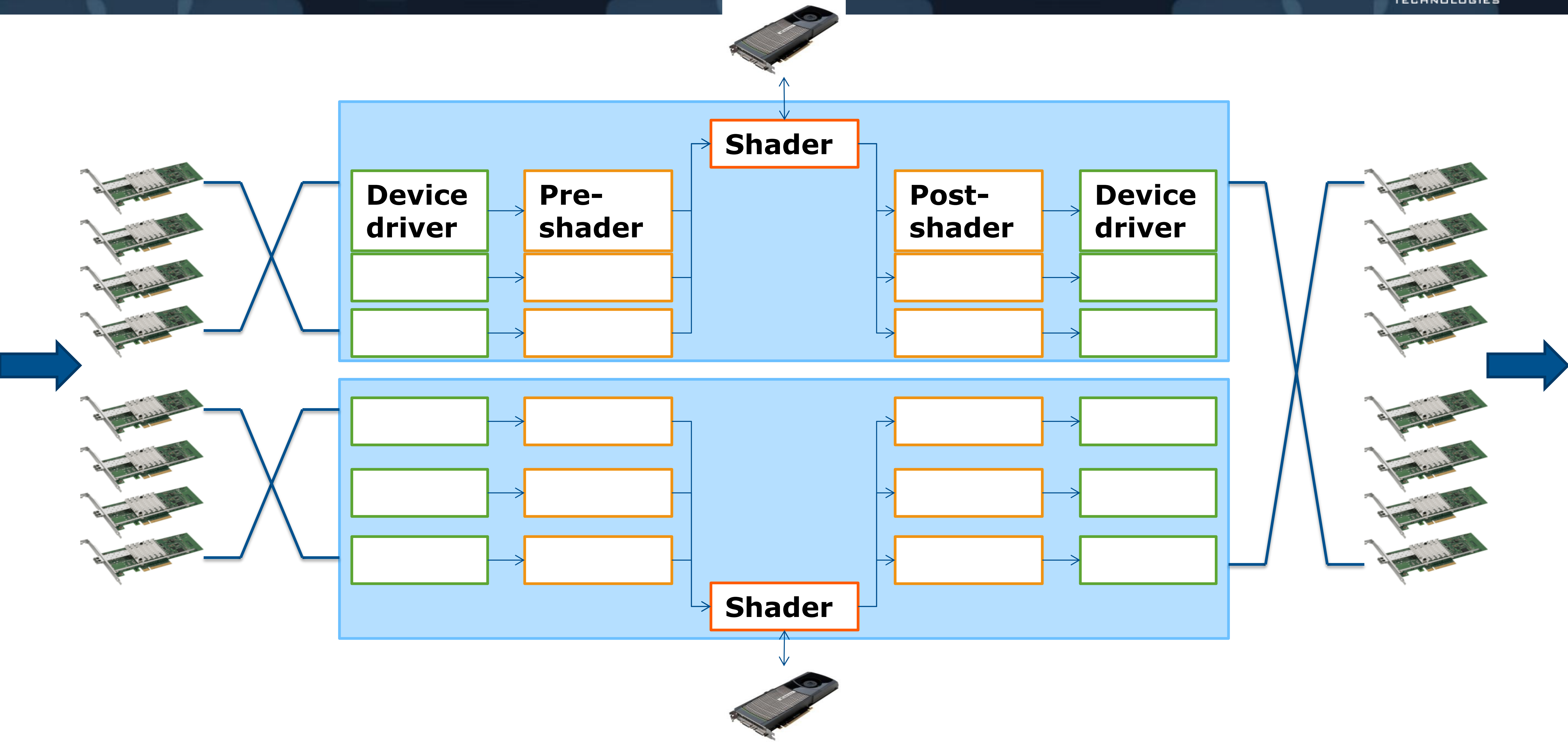
Interfacing with NICs



Scaling with a Multi-Core CPU



Scaling with Multiple Multi-Core CPUs



EVALUATION

Hardware Setup

CPU:

Quad-core, 2.66 GHz

Total 8 CPU cores

NIC:




Dual-port 10 GbE

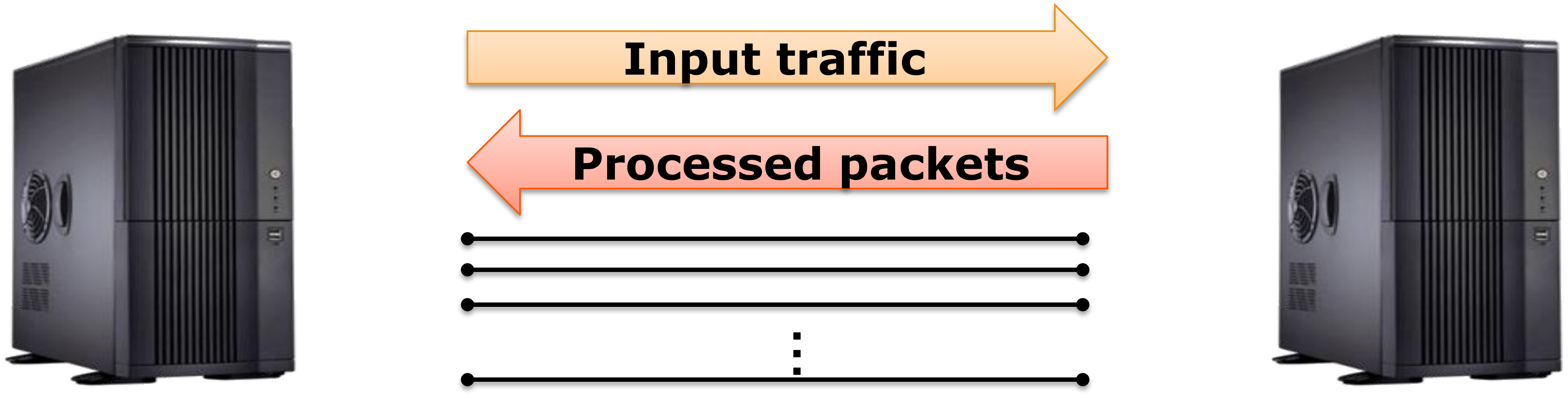
Total 80 Gbps

GPU:


480 cores, 1.4 GHz

Total 960 cores

Experimental Setup



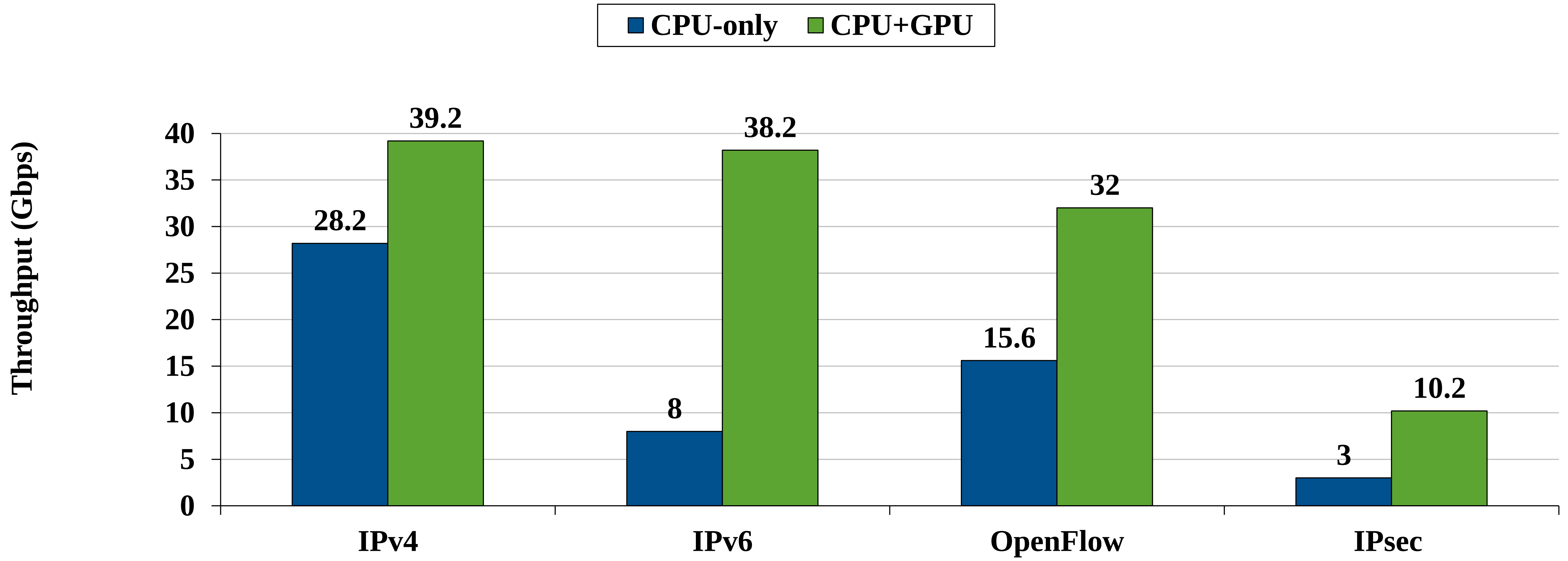
8 × 10 GbE links

Packet generator

(Up to 80 Gbps)

PacketShader

Results (w/ 64B packets)

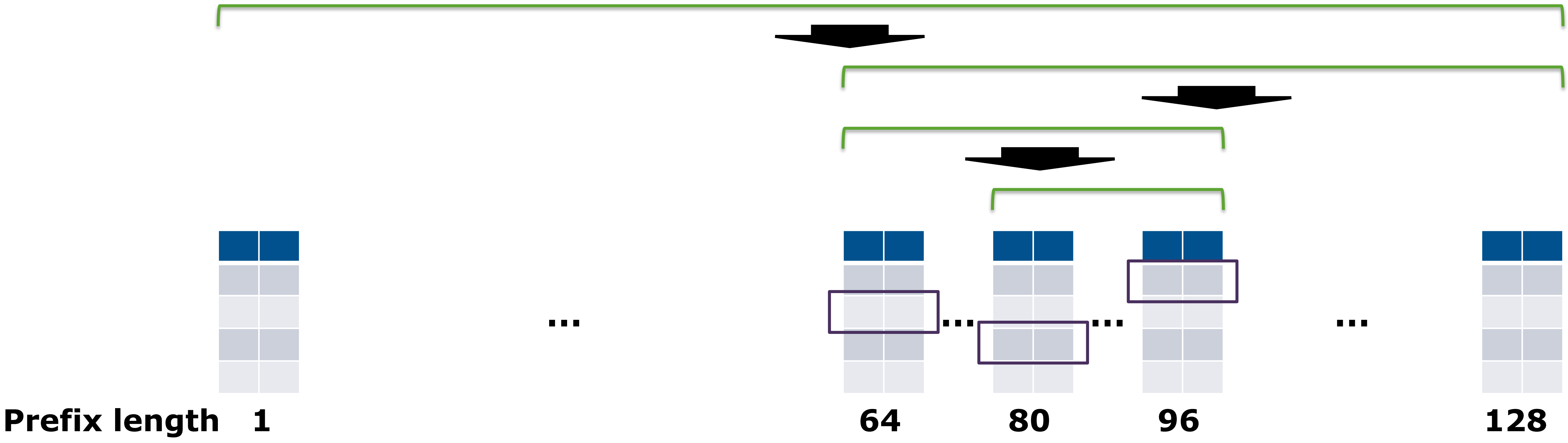


GPU speedup | **1.4x** | **4.8x** | **2.1x** | **3.5x**

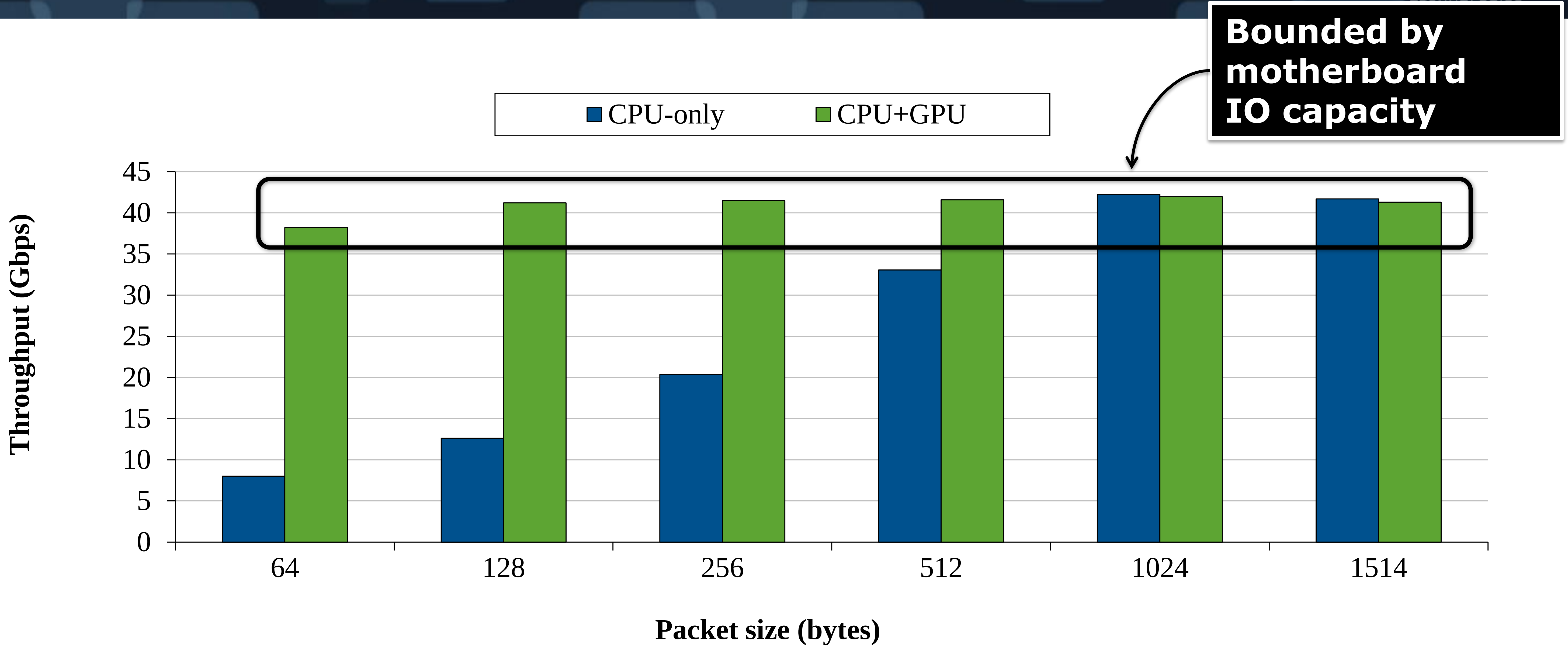
Example 1: IPv6 forwarding

- Longest prefix matching on 128-bit IPv6 addresses

- Algorithm: binary search on hash tables [Waldvogel97]
  - 7 hashings + 7 memory accesses

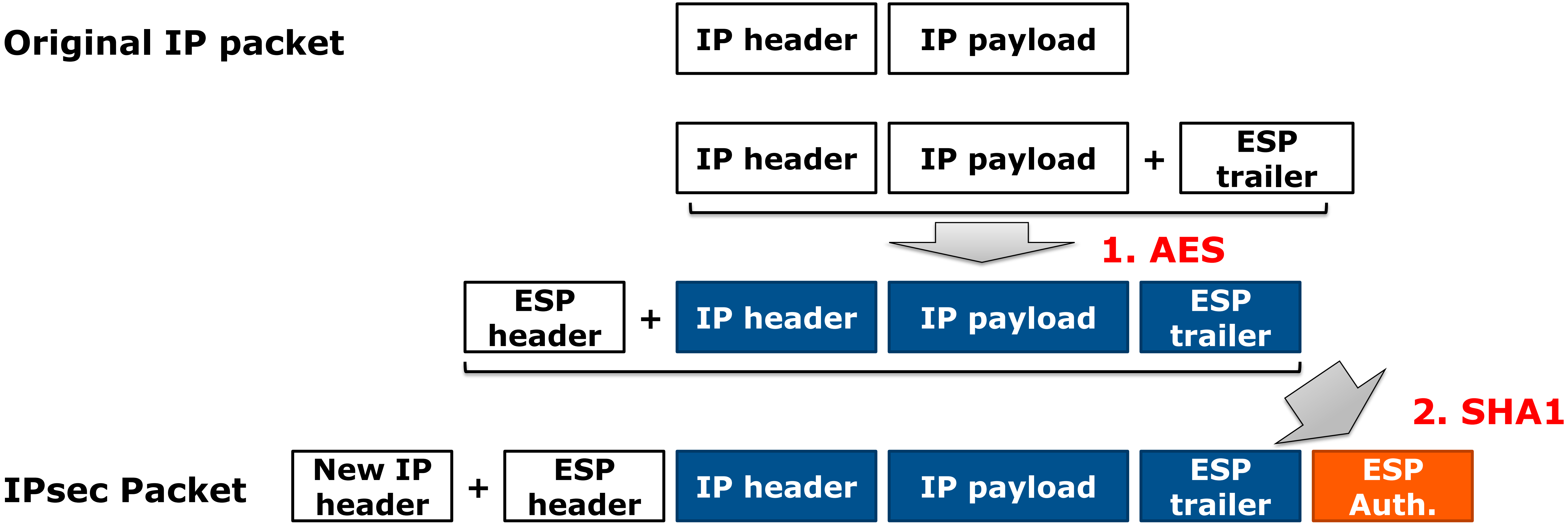


Example 1: IPv6 forwarding



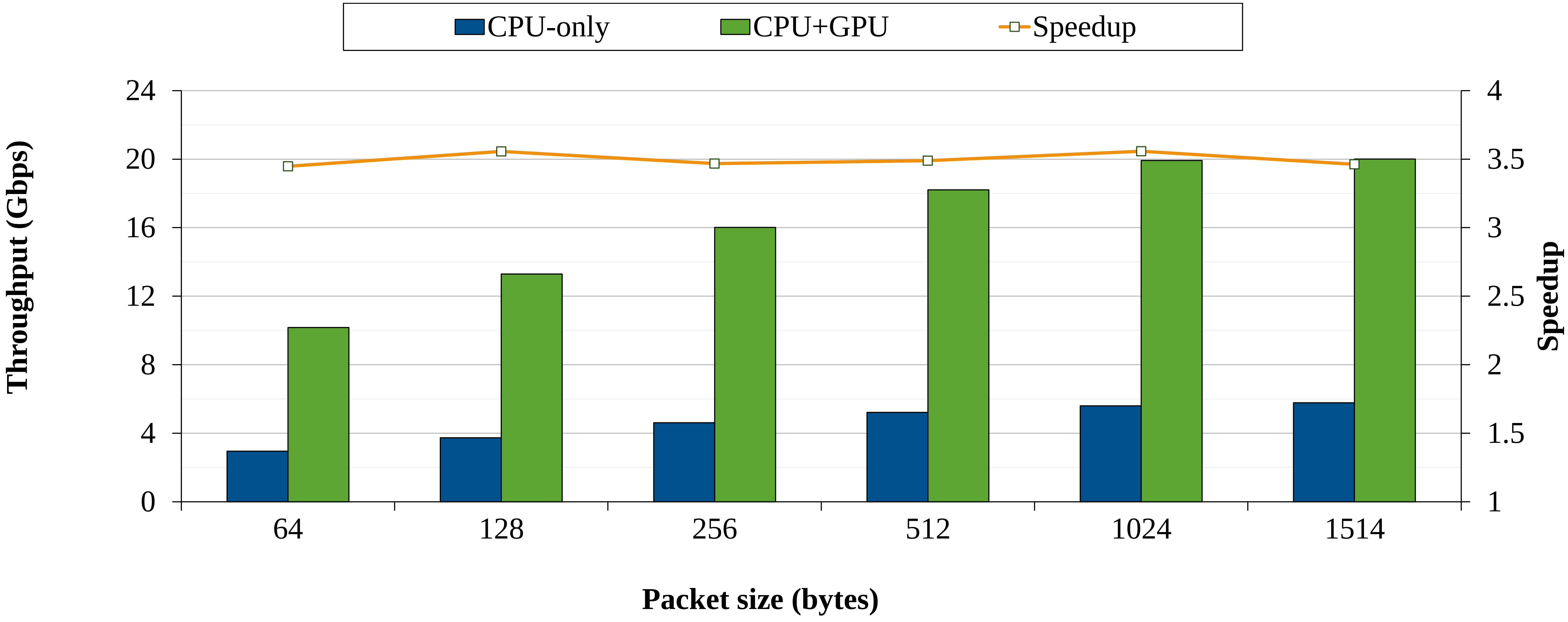
(Routing table was randomly generated with 200K entries)

- ESP (Encapsulating Security Payload) Tunnel mode
  - with AES-CTR (encryption) and SHA1 (authentication)



Example 2: IPsec tunneling

- 3.5x speedup



Year	Ref.	H/W	IPv4 Throughput
2008	Egi *et al*.	Two quad-core CPUs	3.5 Gbps
2008	"Enhanced SR" Bolla *et al*.	Two quad-core CPUs	4.2 Gbps
2009	"RouteBricks" Dobrescu *et al*.	Two quad-core CPUs (2.8 GHz)	8.7 Gbps
2010	PacketShader (CPU-only)	Two quad-core CPUs (2.66 GHz)	28.2 Gbps
2010	PacketShader (CPU+GPU)	Two quad-core CPUs + two GPUs	39.2 Gbps

Kernel

User

Conclusions

- GPU
  - a great opportunity for fast packet processing
- PacketShader
  - Optimized packet I/O + GPU acceleration
  - scalable with
    - # of multi-core CPUs, GPUs, and high-speed NICs
- Current Prototype
  - Supports IPv4, IPv6, OpenFlow, and IPsec
  - 40 Gbps performance on a single PC

Future Work

- Control plane integration
  - Dynamic routing protocols with Quagga or Xorp
- Multi-functional, modular programming environment
  - Integration with Click? [Kohler99]
- Opportunistic offloading
  - CPU at low load
  - GPU at high load
- Stateful packet processing

SSLShader

A GPU-Accelerated Software Router

Thank You

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™