Parallel Programming with (Py)OpenCL for Fun and Profit

Gordon Inggs
Github: Gordonei
Currently Amazon Web Services, EC2
Soon City of Cape Town

Talk Outline:

- **Why PyOpenCL (~15 mins)**
- **How to *x* with (Py)OpenCL (~30 mins)**
  - ***Program CPU, GPUs and more***
  - ***Manipulate Memory***
  - ***Do things in Parallel****
- Repo: https://github.com/Gordonei/ctpug-july-2018

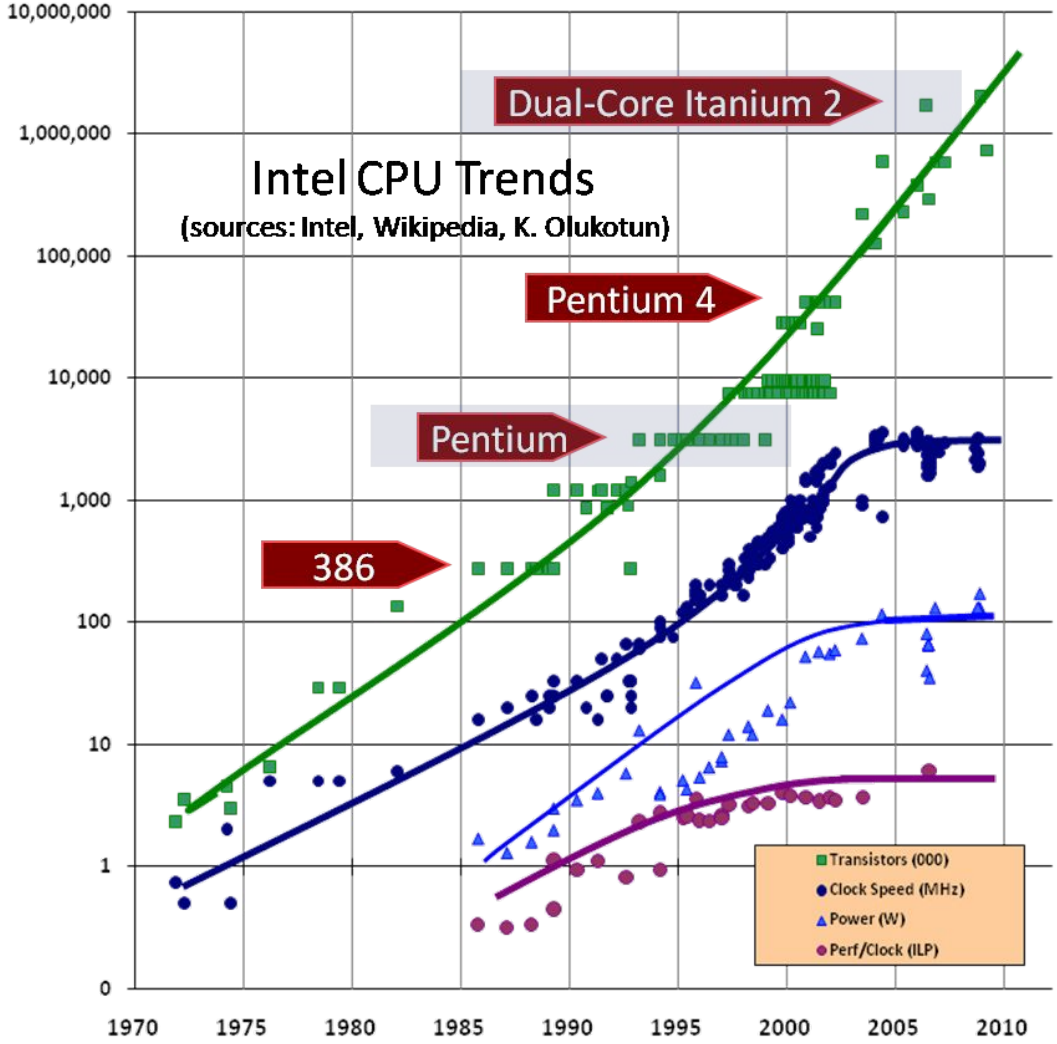
Not concurrently

Talk Approach:

- Tell and Show - trusty vector sum example
- Please interrupt

Why OpenCL?
~=
Why heterogeneous computing?

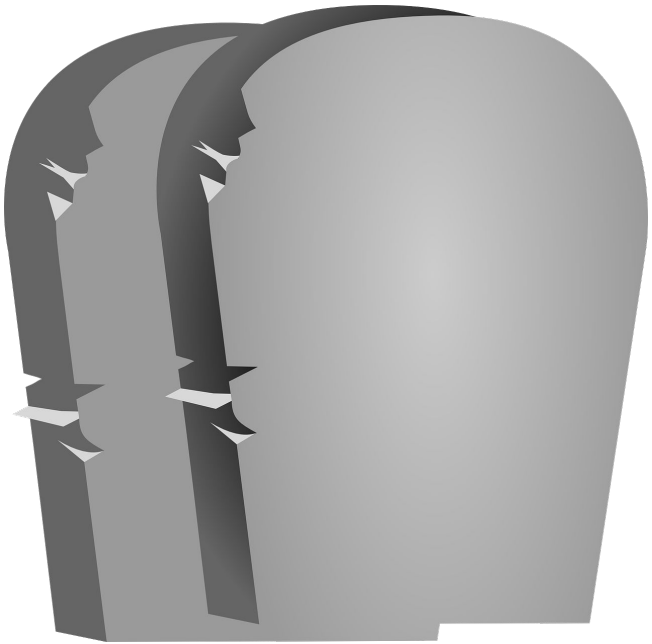
Why computers are becoming stranger (and why that’s a good thing)



Source:
Herb Sutter,
The Free Lunch is Over

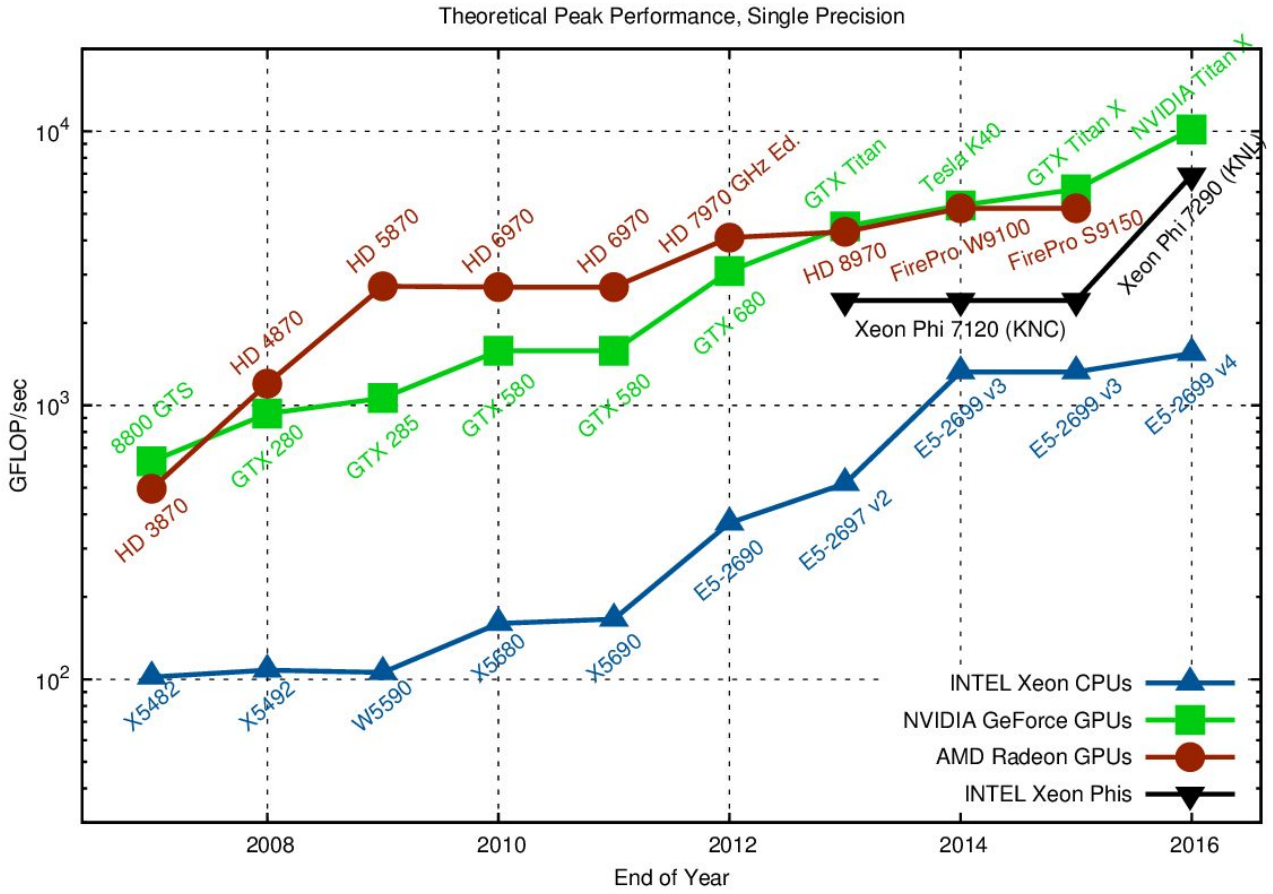
The End of CPU Frequency Scaling

- Moore’s Law:
  - Transistors on the an IC doubling every ~2 years,
  - 1971 to ~2020
  - RIP
- Dennard Scaling:
  - Constant power density
  - 1974 to 2005
  - RIP
- Off-chip IO and Caching
  - Grow at a much slower rate
  - Cache hierarchy can hide this, but ...



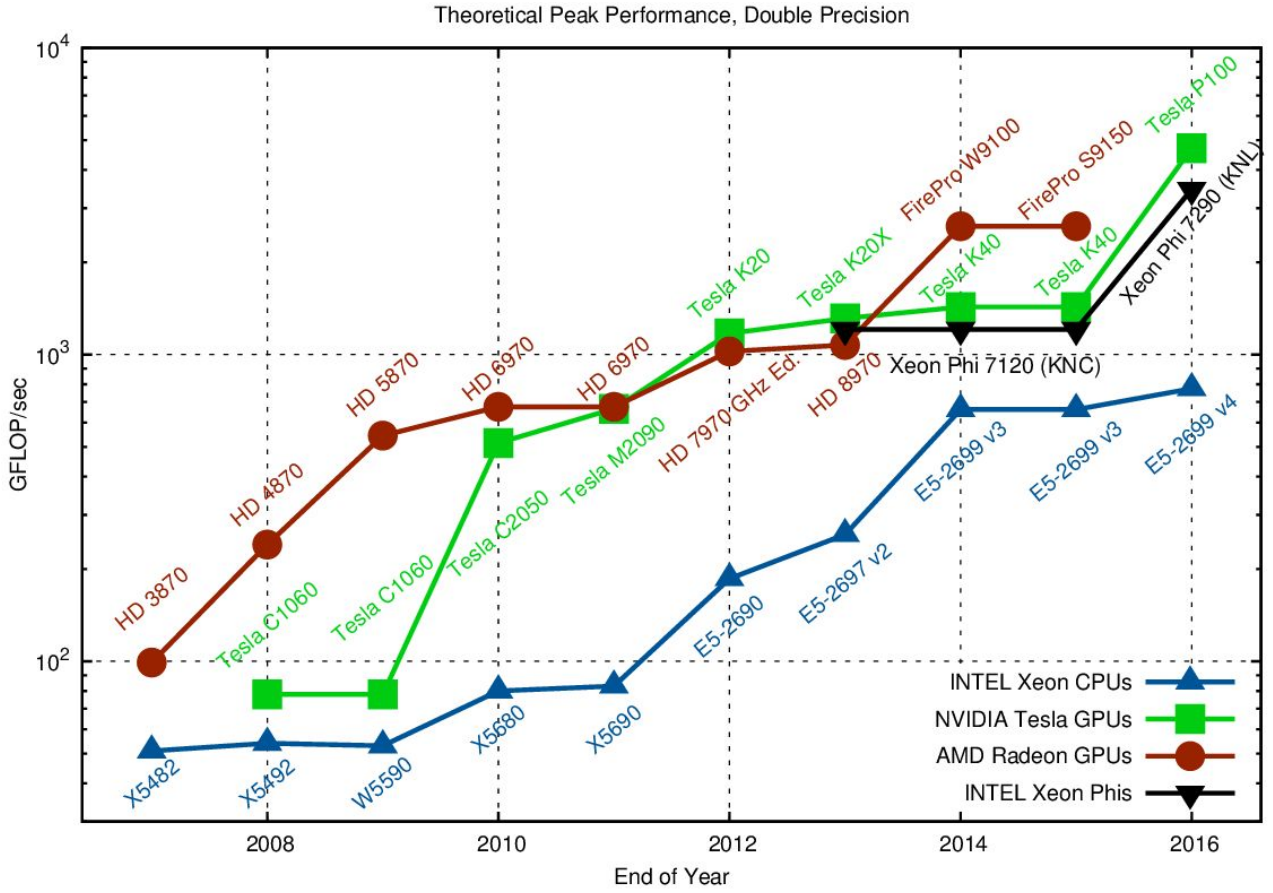
Let’s look
at the alternatives

Performance



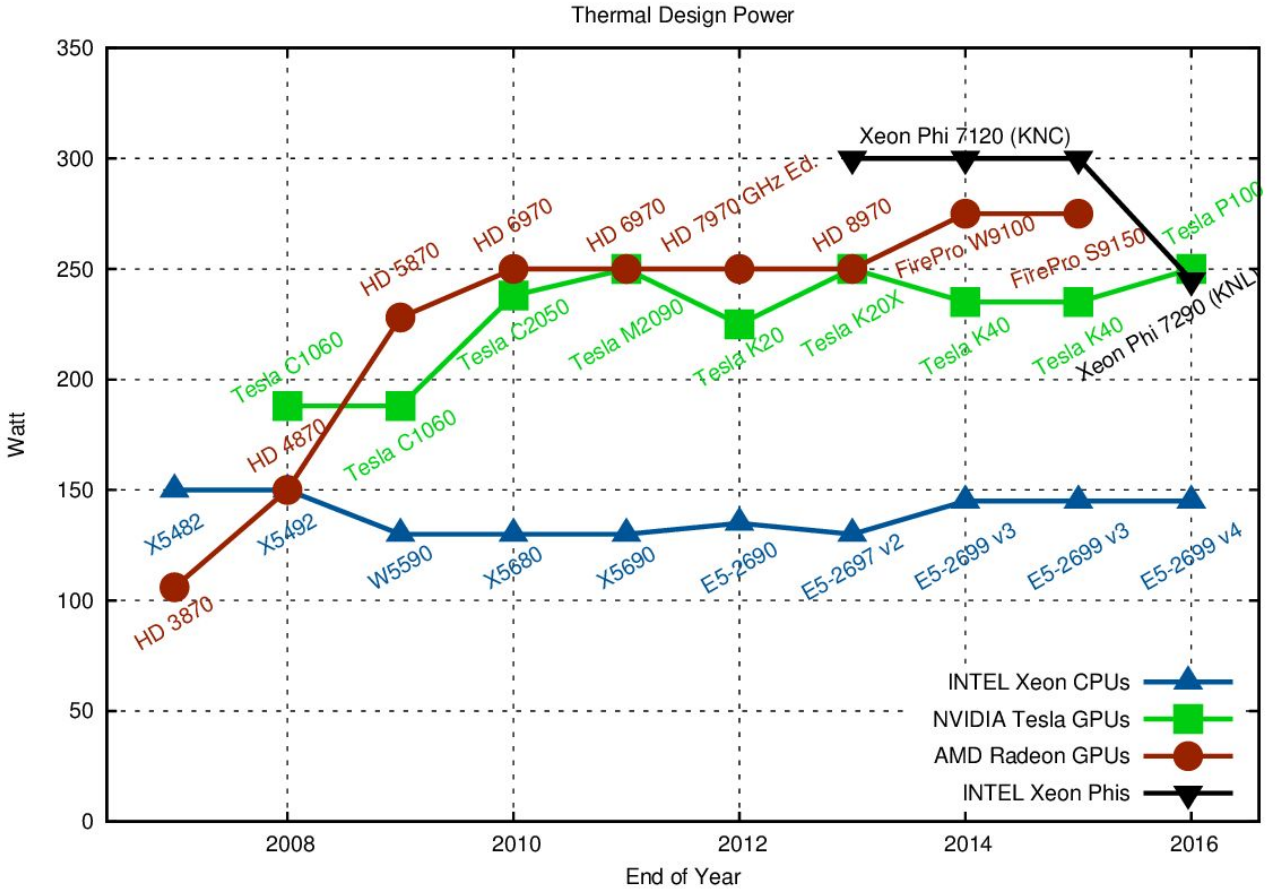
Source: Karl Rupp, ViennaCL

Performance (II)



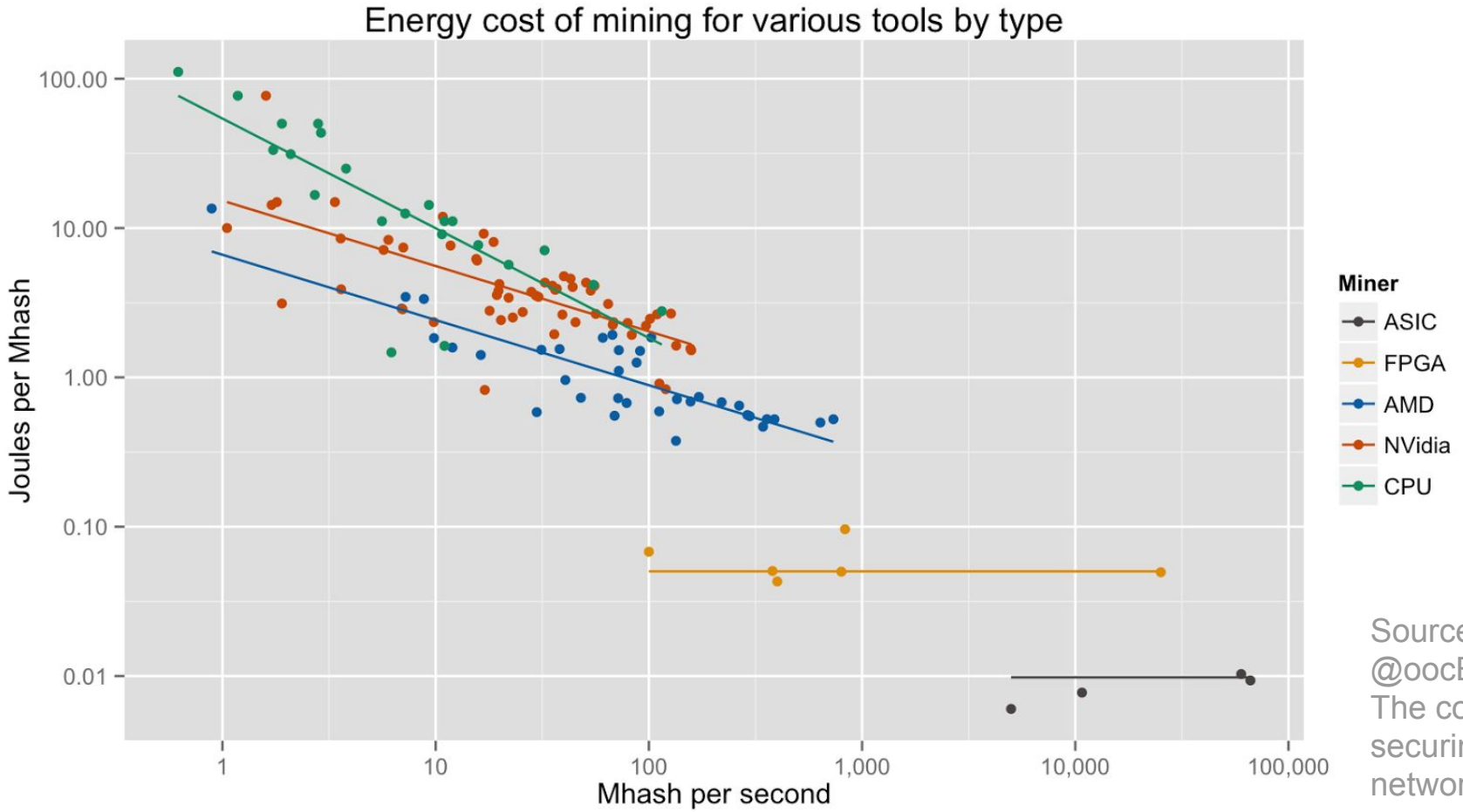
Source: Karl Rupp, ViennaCL

Energy



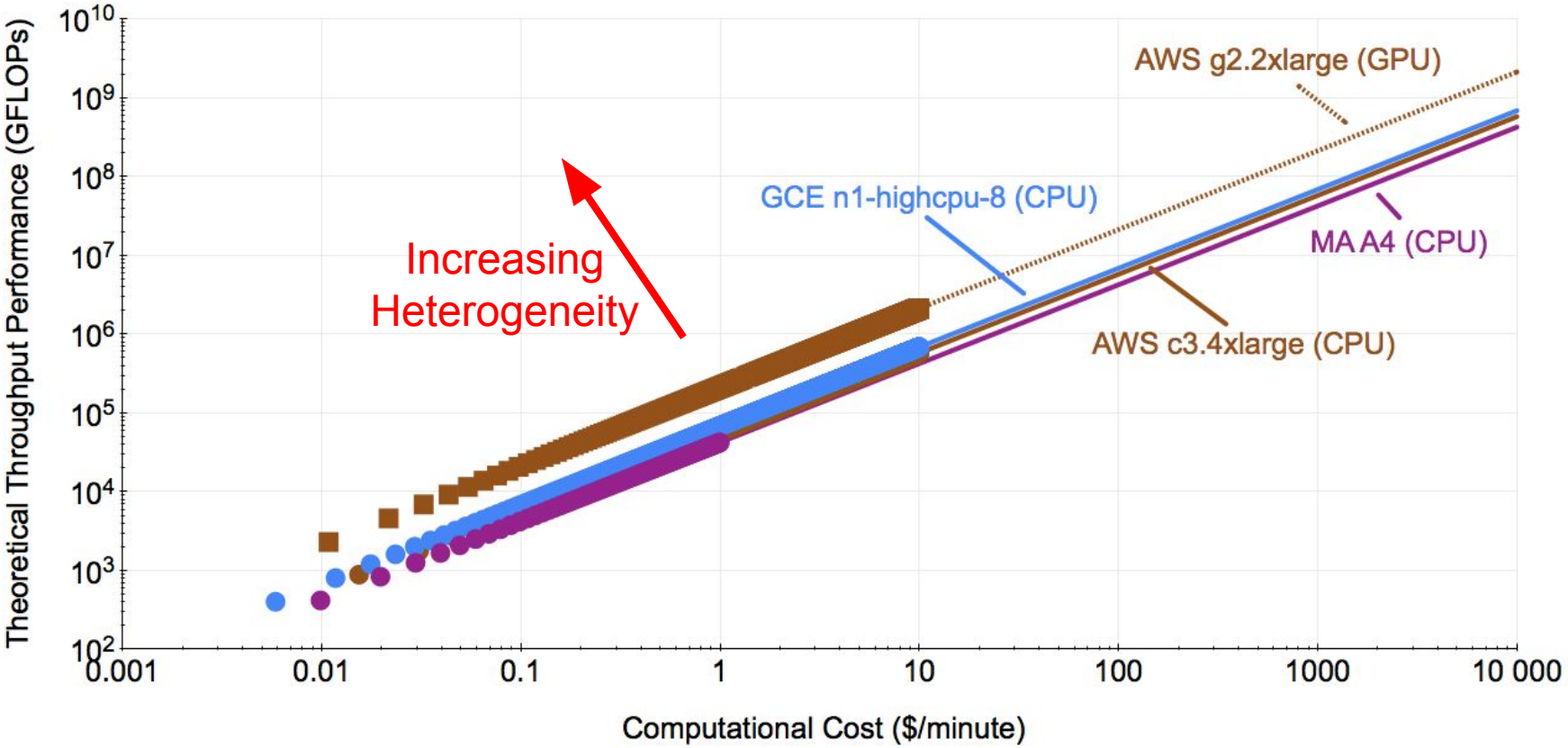
Source: Karl Rupp, ViennaCL

Energy (II)



Source: @oocBlog, The cost of securing the network

Cost



(... as of March 2015)

Heterogeneous Computing sounds great, but...

The Challenges of Heterogeneous Computing™

1. The Orientation Problem

2. The IO Problem

3. The Conceptual Problem

I want to show you how (Py)OpenCL helps address the challenges of heterogeneous computing:

- Orientation Problem
  - Program multicore CPUs and GPUs
  - Inspect device characteristics at runtime
- IO Problem
  - Transfer data efficiently
  - Use device memory hierarchy effectively
- Conceptual problem
  - Express task vs data parallelism
  - Exploit the comparative advantages of the devices available

Part I
Programming Fancy Devices (with PyOpenCL)

In this part...

- How to compile an OpenCL program (and what that is)

- How to run an OpenCL program with Python host code

source: Khronos Group

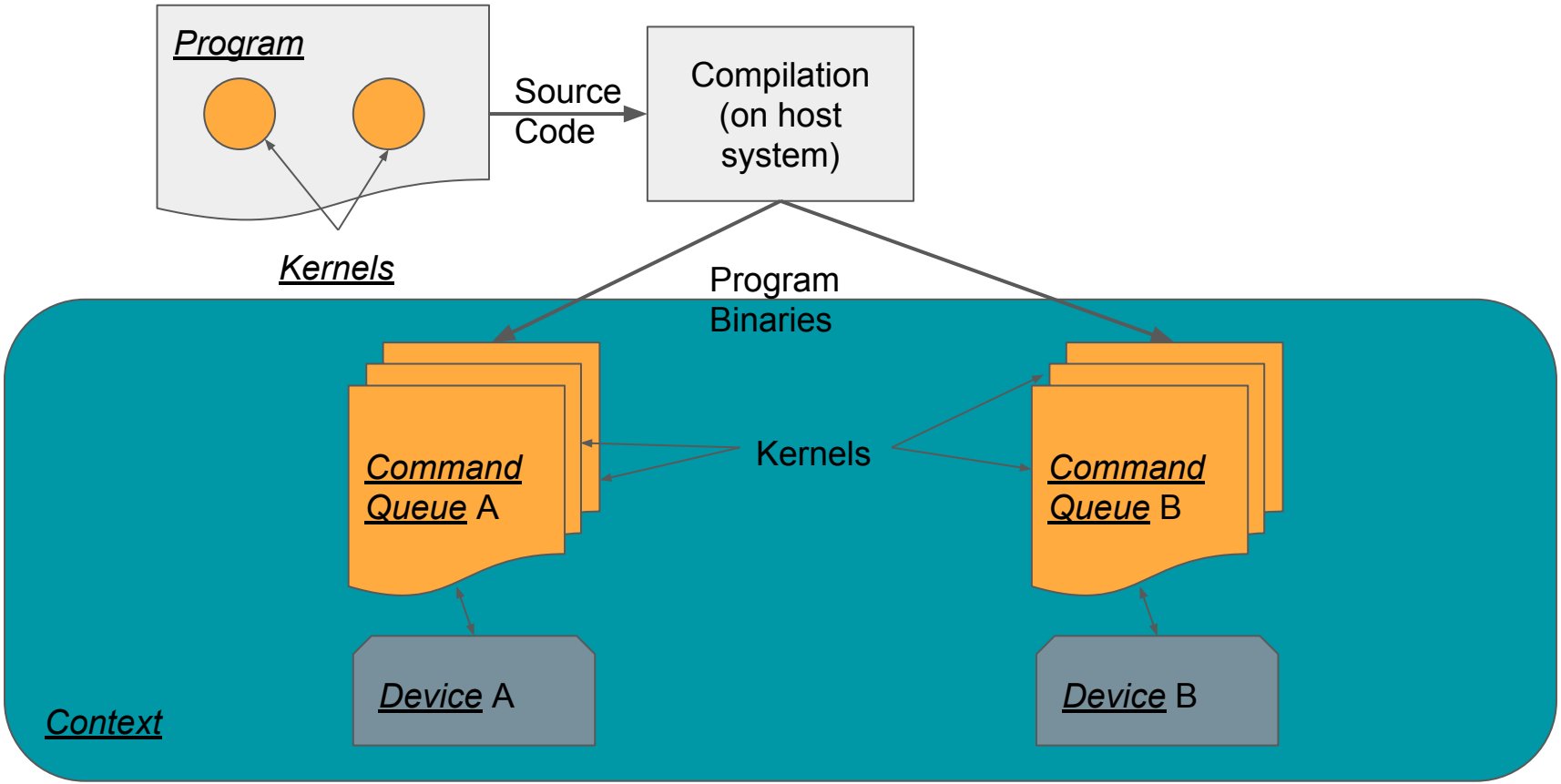
What is OpenCL?

- ‘OpenCL™ (Open Computing Language) is the open, royalty-free standard for cross-platform, parallel programming of diverse processors found in personal computers, servers, mobile devices and embedded platforms.’

- Portable Heterogeneous Computing - not performance portability

- 2 APIs (programming and runtime) and a kernel language

OpenCL Programming API Abstractions



OpenCL Programming API Rules

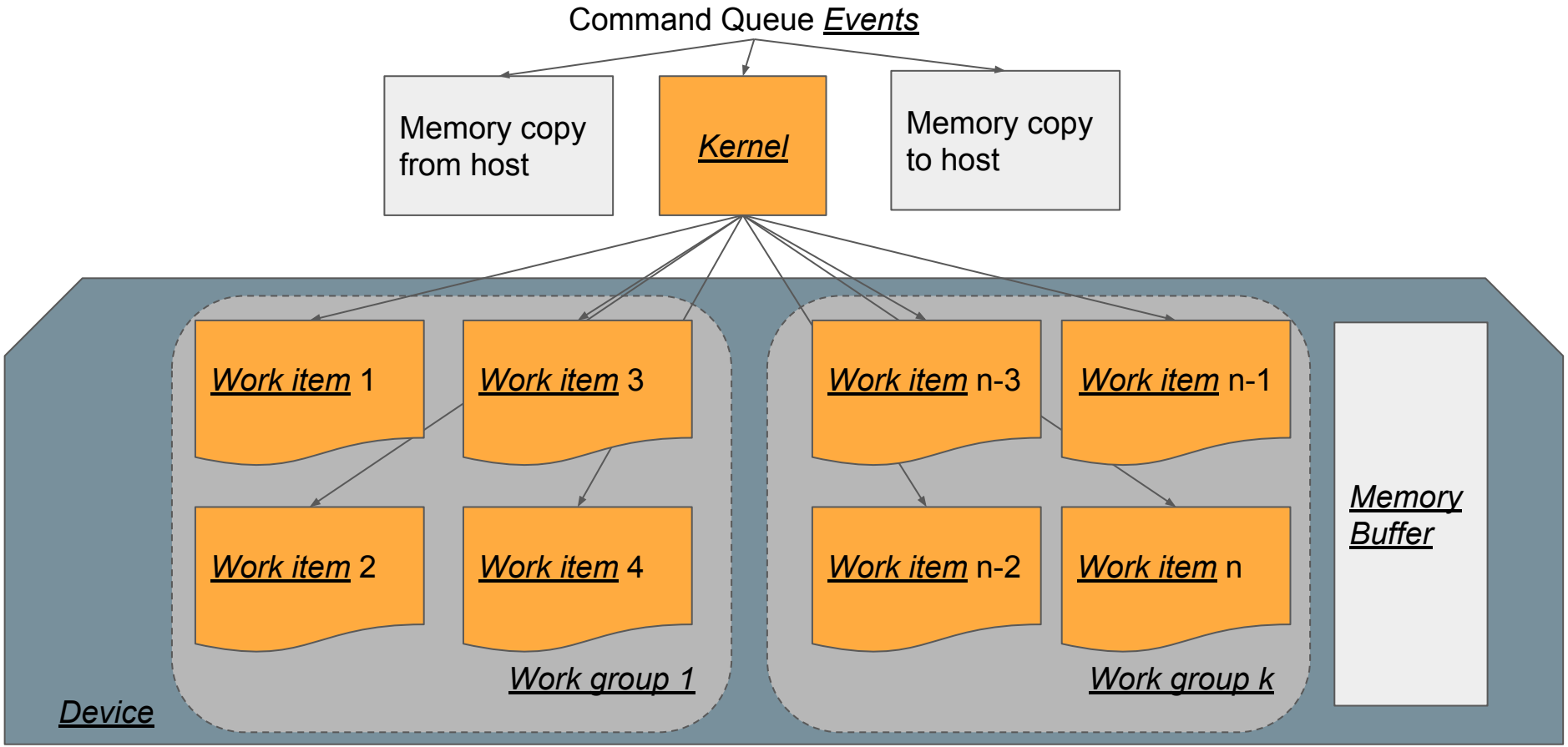
1. Program source code must conform to OpenCL kernel code specification

   (C99 without dynamic memory allocation)

2. One Platform (vendor implementation / compiler) per Context:
   a. NVIDIA
   b. Intel
   c. AMD
   d. Apple
   e. POCL

Let’s build a program!

OpenCL Runtime API Abstractions



OpenCL Runtime API Rules

1. The event ordering of the queue is respected, unless otherwise specified

2. Device memory can only be managed from the host

3. Each work item runs identical code and may execute in any order

4. Work item indices can have up to 3 dimension

Let’s run a program!

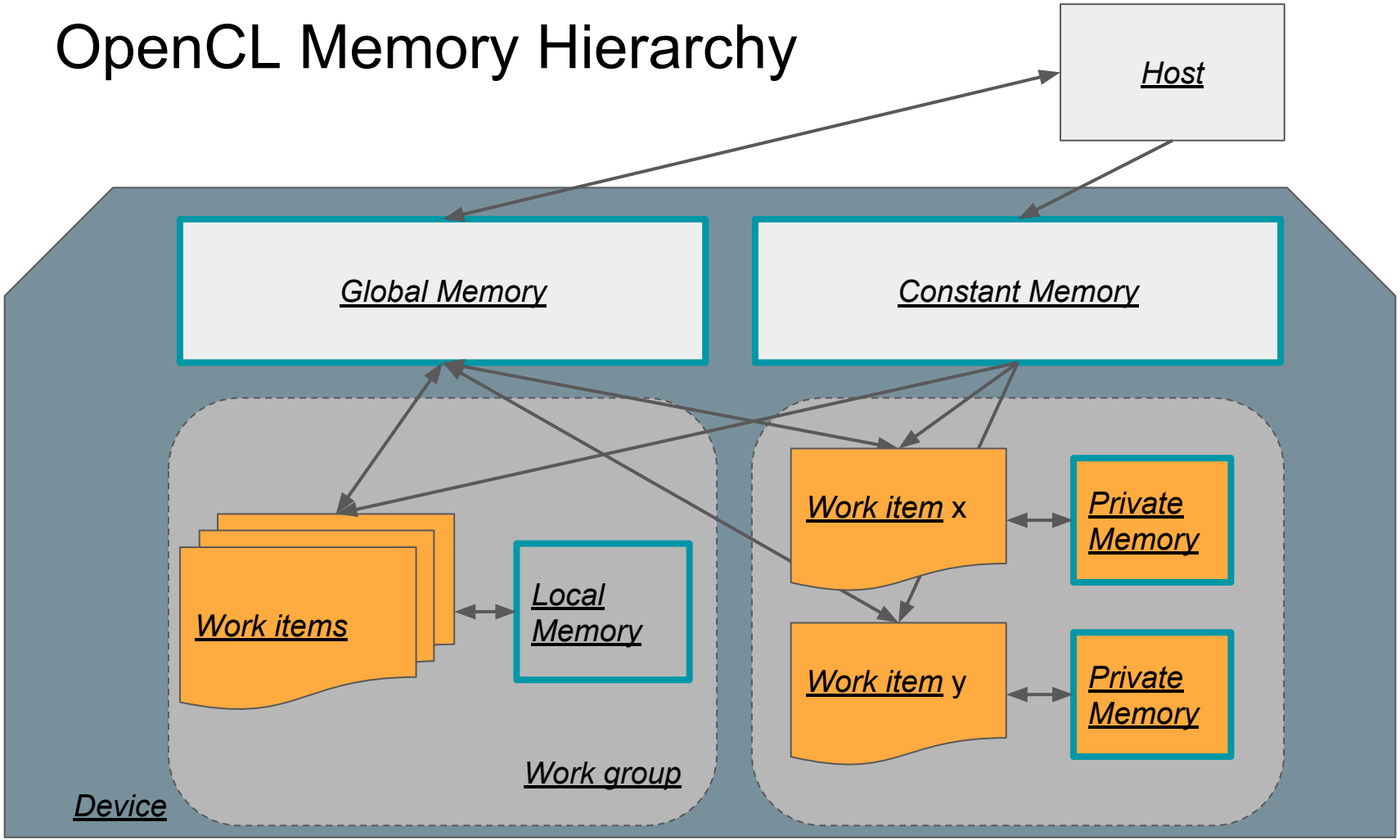
Part II
Moving bits around (with PyOpenCL)

In this part...

- How to declare global memory buffers and copy to them

- How to declare private and local memory buffers, and make use of them

OpenCL Memory Hierarchy



Global Memory

- Roughly equivalent to RAM (i.e. 100s of cycles to access)

- Global Memory Flags:
  - READ_ONLY
  - WRITE_ONLY
  - READ_WRITE

Local Memory

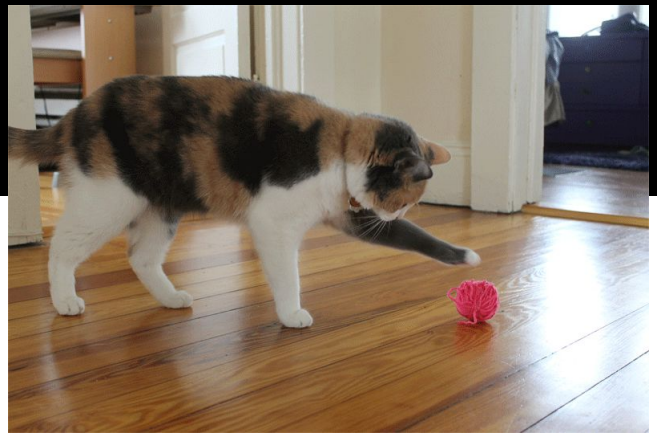
- Roughly equivalent to L3 and L2 Cache (10s of cycles to access)
- Shared within a work-group
- Concurrent write behaviour is undefined

Private Memory

- Roughly equivalent to L1 Cache (1s cycles to access)
- Only accessible within a work-item
- Implicitly used

Let’s manipulate some memory!

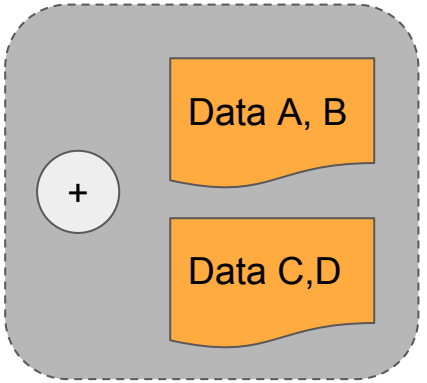
Part III
Doing a lot, at once (with PyOpenCL)



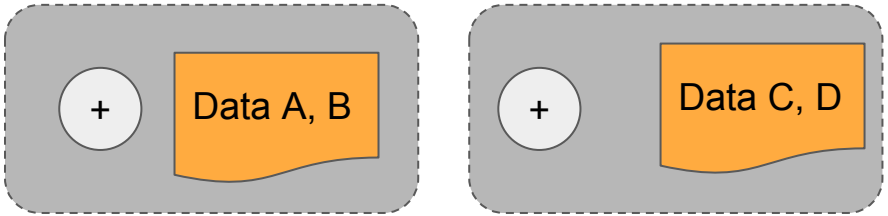
Let’s inspect our devices!

Task vs Data Parallelism

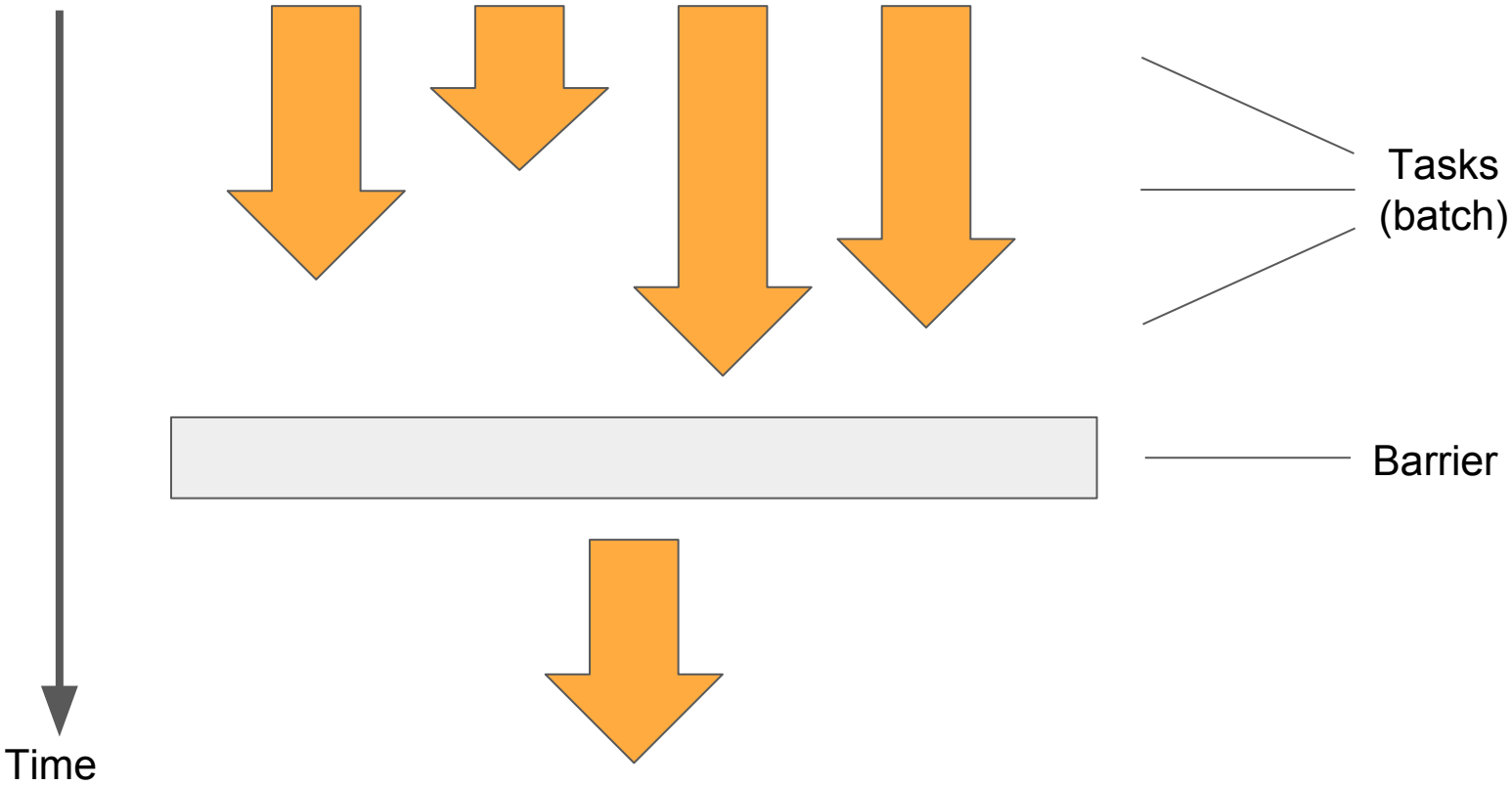
- Flynn’s Taxonomy - SIMD vs MIMD



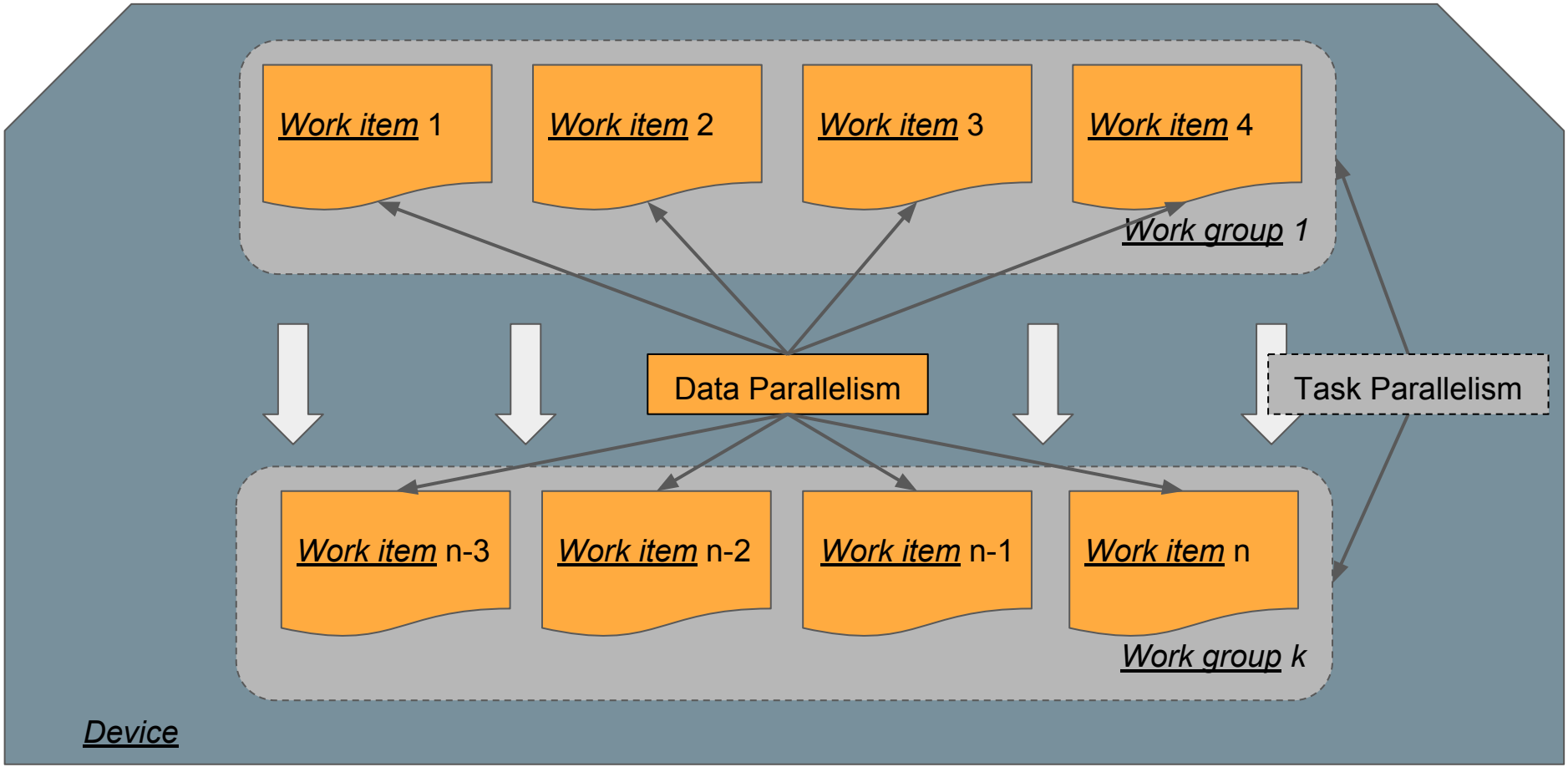
- Broadly:
  - Multicore CPUs => MIMD
  - GPUs => SIMD



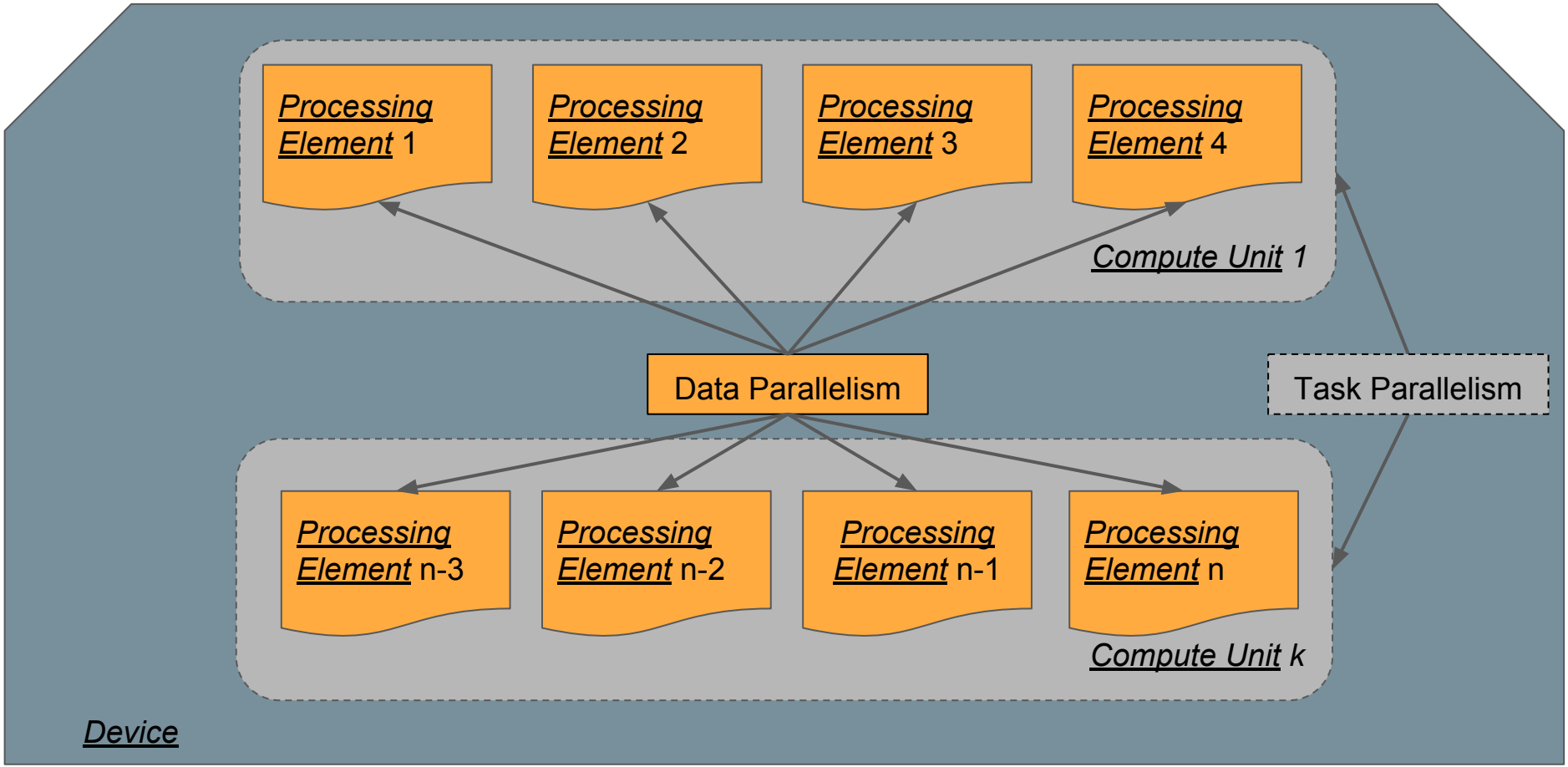
Batch Synchronous Processing



OpenCL Task vs Data Parallelism Abstractions

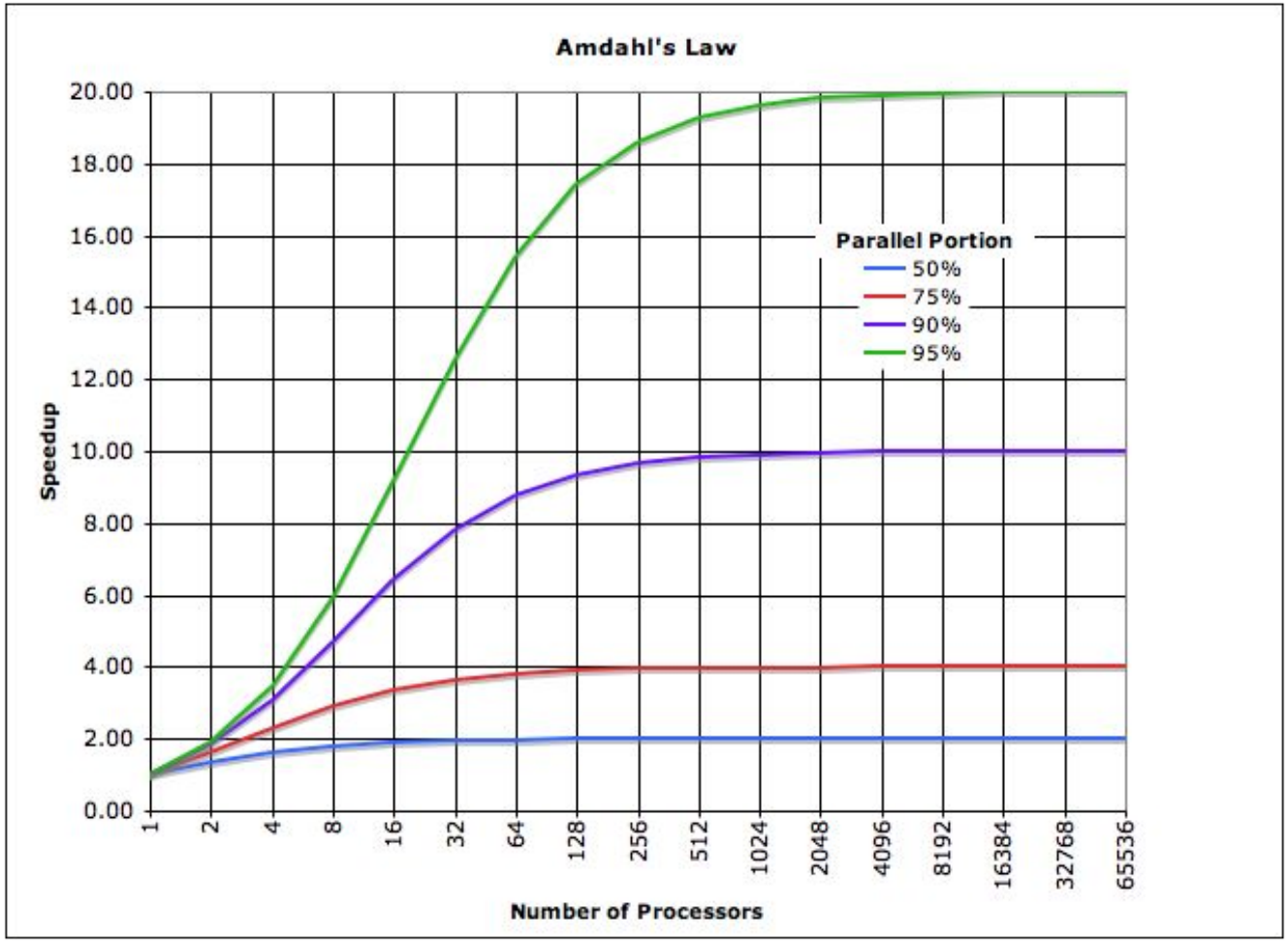


OpenCL Task vs Data Parallelism Device Abstractions



Let’s do a lot, at once!

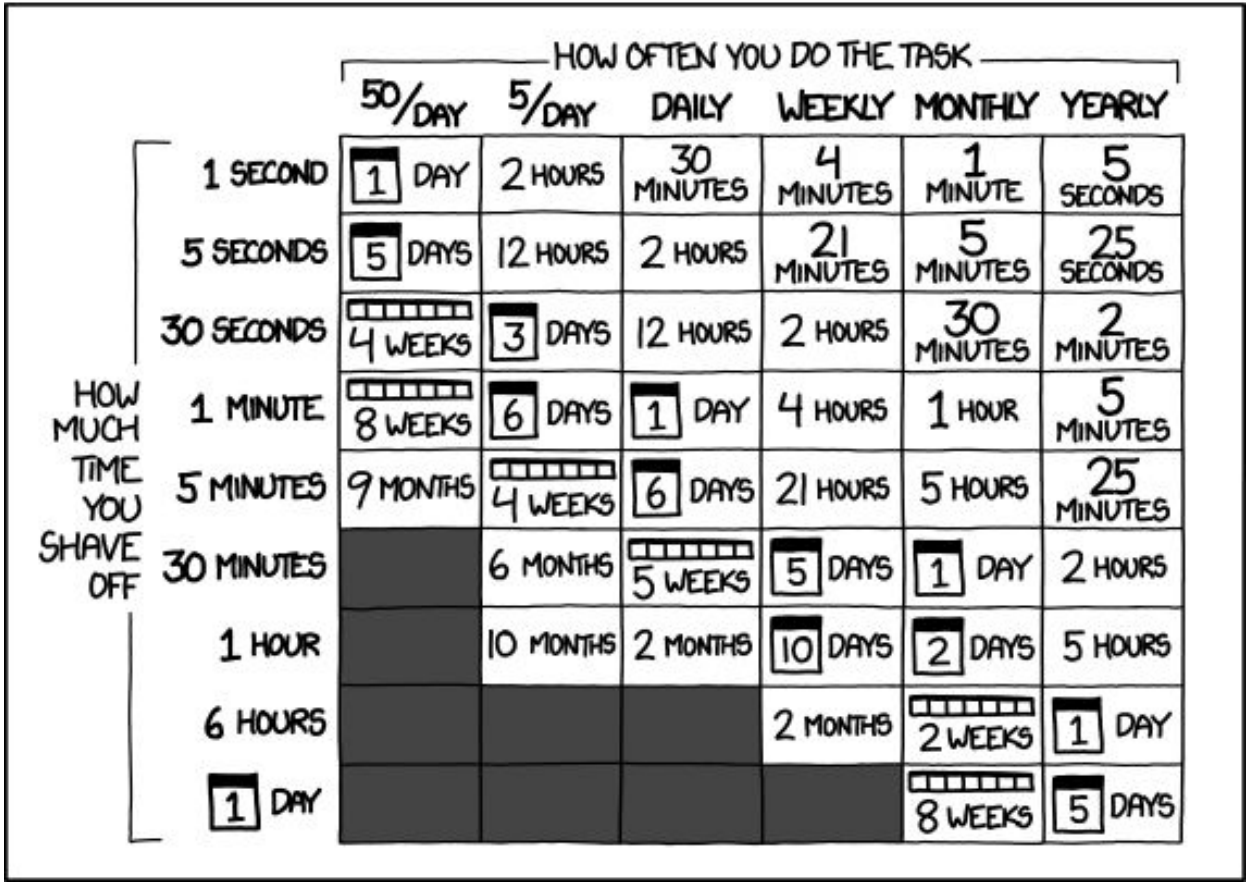
Ca(t)veats



Source: WikiCommons

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?

(ACROSS FIVE YEARS)



Source: XKCD

Thank you

???

Recommended Reading List

- Gene Amdahl, *Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities*
- Page and Luk, *Compiling Occam into field-programmable gate arrays*
- Herb Sutter, *The Free Lunch is Over*
- Asanovic et al., *The Landscape of Parallel Computing Research: A View from Berkeley*
- Tsugio Makimoto, *The Hot Decade of Field Programmable Technologies*
- Lee et al., *Debunking the 100X GPU vs CPU myth*
- Che et al., *Rodinia: A Benchmark Suite for Heterogeneous Computing*
- Thomas et al., *Hardware architectures for Monte-Carlo based financial simulations*
- Mike Giles, *Some (strong) opinions on HPC and the use of GPUs*

More on cost

Name	**Type**	**Price** **(spot price)**	**Ethereum Revenue****
AWS EC2 g3.16xlarge	4 x NVIDIA Tesla M60 GPUs	$0.08 / min ($0.03 / min)	$0.0016 / min
AWS EC2 p2.16xlarge	16 x NVIDIA Tesla K80 GPUs	$0.26 / min ($0.06 / min)	$0.0048 / min
MS Azure NC24r	4 x NVIDIA Tesla K80 GPUs	$0.66 / min	$0.0012 / min
AWS EC2 f1.16xlarge	8 x Xilinx VU9P FPGAs	$0.24 / min ($0.02 / min)	?
GCE P100**	1 x NVIDIA Tesla P100 GPU	$0.04 / min	?
GCE K80**	1 x NVIDIA Tesla K80 GPU	$0.01 / min	$0.0003 / min

*Instance not included.
** As of 5pm, 2017/10/03.