Hardware-Software Codesign

Thomas - POCS - Fall 2023 - EPFL

Outline

Some historical perspective on the HW/SW dance.

Performance Cost of Abstraction

Chasing the Unicorn

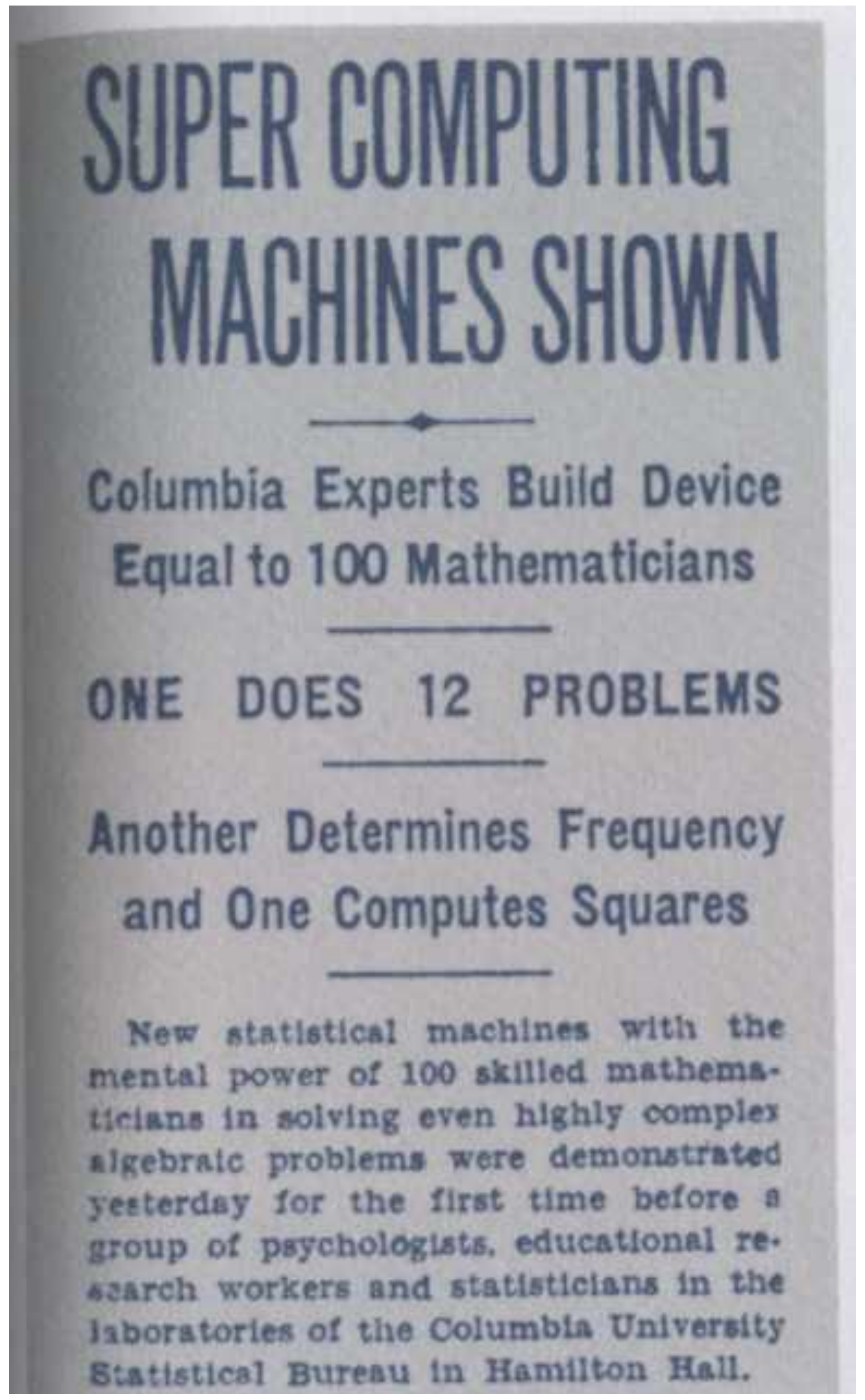
Communication and Interfaces - 3 case studies

Principles of Acceleration

Performance Cost of Specialisation

Hardware/Human Codesign

Pre-Turing Era - 1930s



SUPER COMPUTING MACHINES SHOWN

Columbia Experts Build Device Equal to 100 Mathematicians

ONE DOES 12 PROBLEMS

Another Determines Frequency and One Computes Squares

New statistical machines with the mental power of 100 skilled mathematicians in solving even highly complex algebraic problems were demonstrated yesterday for the first time before a group of psychologists, educational research workers and statisticians in the laboratories of the Columbia University Statistical Bureau in Hamilton Hall.

Fancy tabulating machine could High Value Computations: $\sum_i x_i^2$ or even $\sum_i w_i \cdot x_i$

Technology was electromechanical relays

Typically, quite a few “telescoping" (pipelining) tricks (example)

Humans interact with the kernel performed by the machine:

Machine Input: Feed it punchcards with data

Machine Outputs:

- New punchcards (write once medium, for partially accumulated data)
- Human readable summary on fan-fold paper

Cool computations: SELECT/WHERE/GROUP BY !

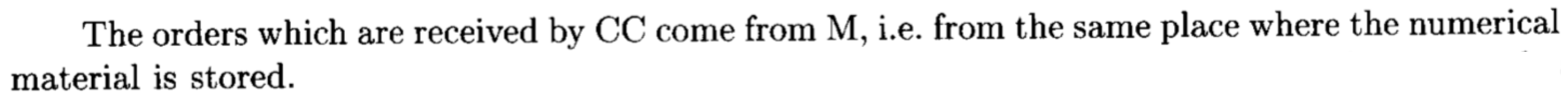
Performance: Could ingest 150 punch cards per minutes

Source: www.columbia.edu/cu/computinghistory/packard.html

1945 - Vacuum Tube’s Era

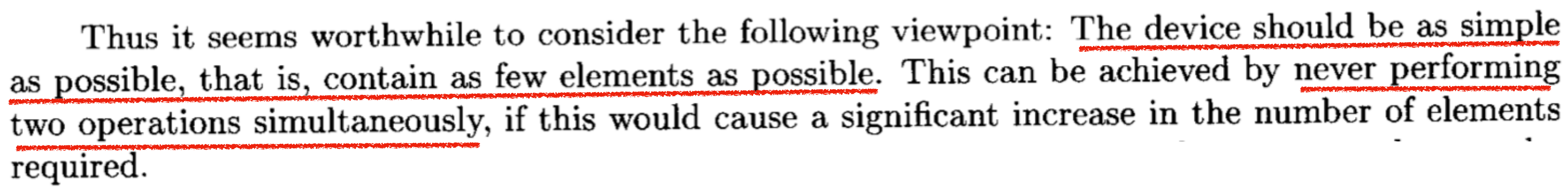
Von Neumann’s magic - Death of hardware, birth of Software

Reports on Building an “Automatic Computing Device” (C[entral], M[emory], R[ecording] (for IO))

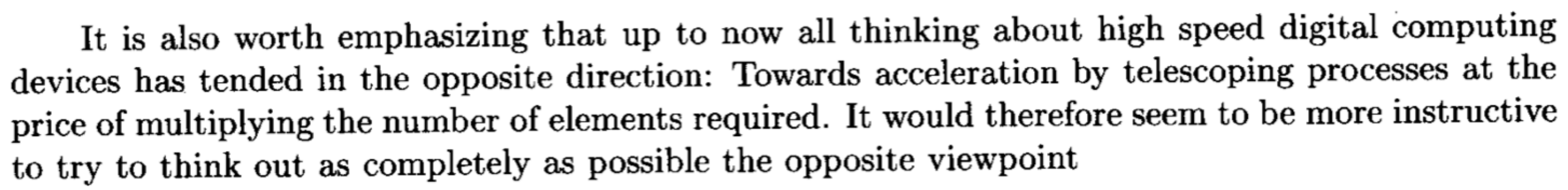


The orders which are received by CC come from M, i.e. from the same place where the numerical material is stored.

Von Neumann’s Insight/Suggestion (Mechanical - ms, Vacuum tube - us)



Thus it seems worthwhile to consider the following viewpoint: The device should be as simple as possible, that is, contain as few elements as possible. This can be achieved by never performing two operations simultaneously, if this would cause a significant increase in the number of elements required.



It is also worth emphasizing that up to now all thinking about high speed digital computing devices has tended in the opposite direction: Towards acceleration by telescoping processes at the price of multiplying the number of elements required. It would therefore seem to be more instructive to try to think out as completely as possible the opposite viewpoint

(https://web.mit.edu/STS.035/www/PDFs/edvac.pdf)

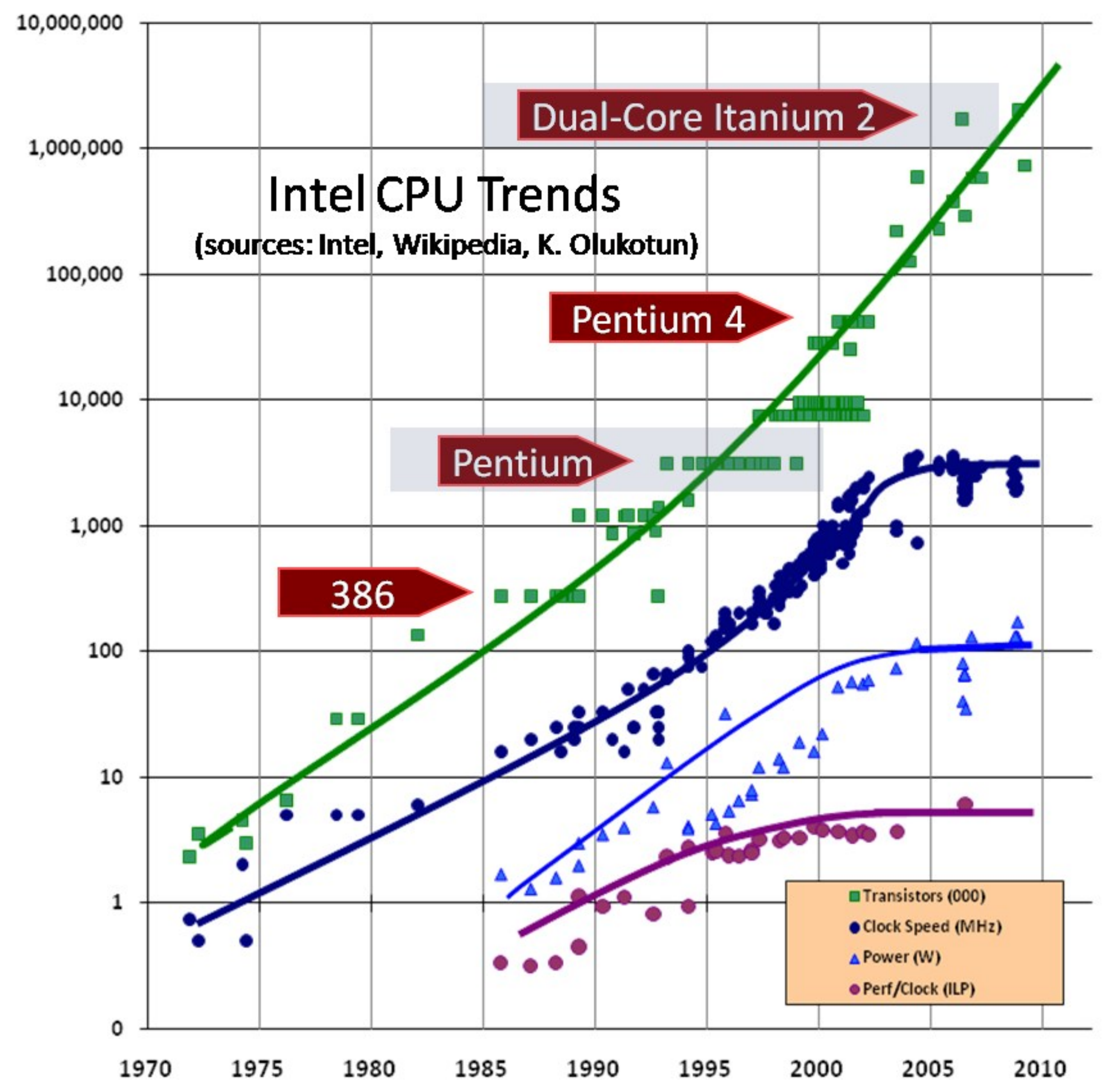
Very rich paper: why binary rocks, thoughts about errors in computing, JIT, biomimetics, brain VS computer, synchrony/asynchrony …

Feynman’s observation

Plenty of Room at the Bottom

1959 Feynman mentally explores the future of miniaturisation: in the future (present) we should be able to do insanely small machines!

Research and industry makes room at the bottom: Moore’s law



https://web.pa.msu.edu/people/yang/RFeynman_plentySpace.pdf

Feynman was right

Transistor Free Lunch Party!

Side-effects:

Smaller transistor ~ faster clock ~ no effort, same design go faster

Smaller transistor ~ more transistors on a given chip, what do we do with them?

Von Neumann’s suggestion of simple machine is dead:

- And now it is not just arithmetic tricks
- New optimisation opportunities!

"All problems in computer science can be solved by another level of indirection …
except for the problem of too many levels of indirection"

~David J. Wheeler (Maybe ?)

Mid 2010s, after 70 years of indirections: time for a spring cleaning

Cost of Abstraction

There is Plenty of Room at The Top

Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that the version executes. "Absolute speedup" is time relative to Python, and "relative speedup," which we show with an additional digit of precision, is time relative to the preceding line. "Fraction of peak" is GFLOPS relative to the computer's peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	—	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45
	TPU v1 (2016)	<0.001	˜90,000.000	~15,000,000	˜270	100% (???)

Source: https://www.microsoft.com/en-us/research/uploads/prod/2020/11/Leiserson-et-al-Theres-plenty-of-room-at-the-top.pdf

TPUv1/Processor: 28/22nm, ˜350mm2, ˜700MHz/˜3GHZ

Remarks

Flame war between C/Java is a fight between 0.01% efficiency and 0.03% efficiency (This depends from program to program, compiler to compiler!)

Modern processors already embed a whole bunch of accelerators: Vector unit, often vastly underused

Algorithm complexities make simplifying assumptions (RAM model) and hides real cost

size(Workforce) usually decreases with speedup :(

Surprisingly, there are also performance costs to specialisation (end of the lecture)

Performance evaporation (Knuth's challenge)

Intuitions about the cost of software abstractions

Knuth's challenge (1989): "Make a thorough analysis of everything your computer does during one second of computation."

(Demo Konata if t < 15 && Linux laptop plugged)

Pragmatically: Modern AI is enabled by codesign

GPT3 evaluation ~

>> Read all the paper books in the EPFL library

Compute a nontrivial (arithmetic intensive) function of all the words present in all the books and some context

Output 1 token

Rinse and repeat, with the updated context of adding the produced token

Hundreds of GB per token! It is a pharaonic amount of compute

The Crumple-Horned Snorkack of Codesign

(Chasing the Unicorn)

The Crumple-Horned Snorkack of Codesign

Some mythical thing that probably does not exist, but is still cool to look for

Write your algorithm in Foo-Lang

# foolc —powerConstraint=1W —areaConstraint=100um2 —clock=3GHz —IPblockLibraries armA72 matmulGoogle128 ether10GBBroadcom -O3 myProgram.fool

-> Produce hardware description (Maybe FPGA configuration or directly ASICs description) + software for the CPUs + software glue code + user app that uses all that

Next Level Unicorn : Foo-Lang is actually C/Python/Haskell/Scala, so we can just reuse existing code

Escape Hatch:

"foolc" does not work for all programs

"foolc" might produce suboptimal code

Progress made while looking for the Crumple-Horned Snorkack

Behavioral VS Structural Verilog, and others High-Level Hardware Languages

Modern High Level Synthesis

System-Level Design Tools

Field Programmable Gate Arrays

Communication and Interfaces

Partitioning and Mapping, Design Space Exploration

Communication and Interfaces

by examples

Case Study 1:
AES instructions -
Tightly coupled accelerators

What is AES

Symmetric key crypto Block Cipher

Compiling with -O3 (for RISCV):

- ~300 straight-line instructions per AES-round (Same for X86-clang)
- Encrypting 128 bits requires 10 rounds -> ~3000 instructions

Question:
If I send a file by email, how many times will the content of the file be AES-ed before it is opened by the recipient?

AES is a “High-Value" Application

AES is a part of most commonly used cipher suites for TLS.

TLS protocol massively used on the internet

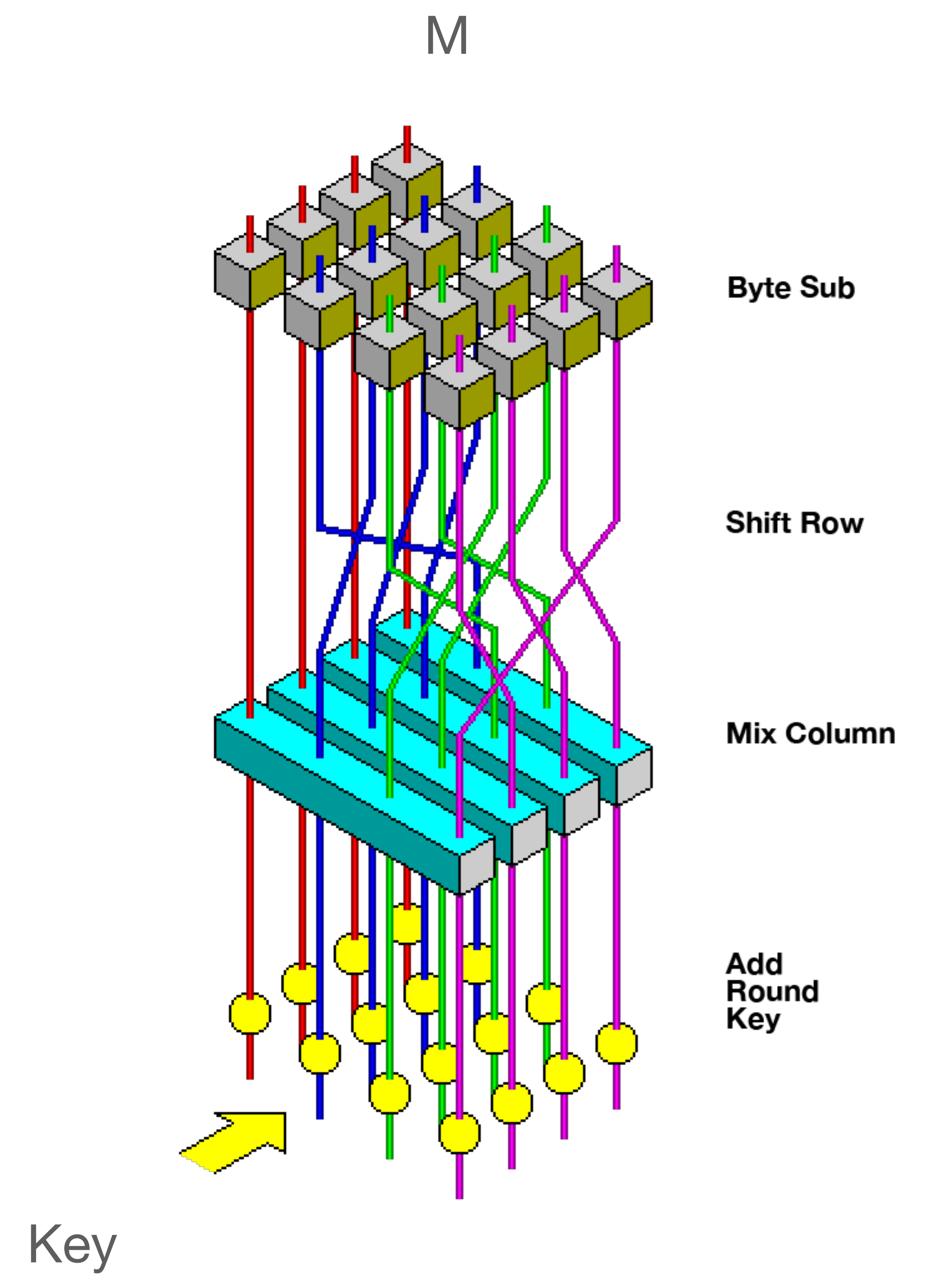
- Most data outputted on the network get crunched through AES
- WIFI also uses AES
- FileVault on Mac
- -> quite Important for server workloads

There are other reasons to accelerate AES (Security)

AES encryption

M and Key are 128 bits (16 * 8 bits)

AES_encrypt(M, Key) = 128 new bits



AESENC

From https://eprint.iacr.org/2016/299.pdf

```
AESENC xmm1, xmm2
```

“Perform one round of an *AES encryption* flow, using one 128-bit data (state) from xmm1 with one 128-bit round key from xmm2”

AESENC can be thought of like ADD, a new arithmetic instruction. Can nicely integrate it in the pipeline of the processor, easy!

Latency: 4 cycles, Throughput: 1 token per cycle

Deployment “challenge":

People should use the AESENC instruction and not their own C implementation anymore!

Scope and Limitations of AESENC-style of acceleration

There are quite a few applications that can be enabled by low-latency instructions

Intel Vector Extensions (MMX -> SSE -> AVX) originally for multimedia and then more general

Compiler are still struggling hard to properly use those extensions

Limitations:

Many computations act on more data than just data that can be held in registers

- Need access to memory

Many computations need hundred of cycles to complete

Why is it a limitation?

Why is it a problem if instruction needs to access a lot of memory?

Accelerators are just devices that compute!

How do devices work?

Examples of successful “accelerators”

DL accelerators (example: Gemmini, NPU, TPU)

Network Interface Cards (NIC)

  TCP/IP stack runs on some NIC

GPU, GPGPU

Sound Card

Basic observations

Wide discrepancy in programmability

Programmable:

- Modern GPU - runs more or less arbitrary Cpp code
- Modern smartNIC can run programs too

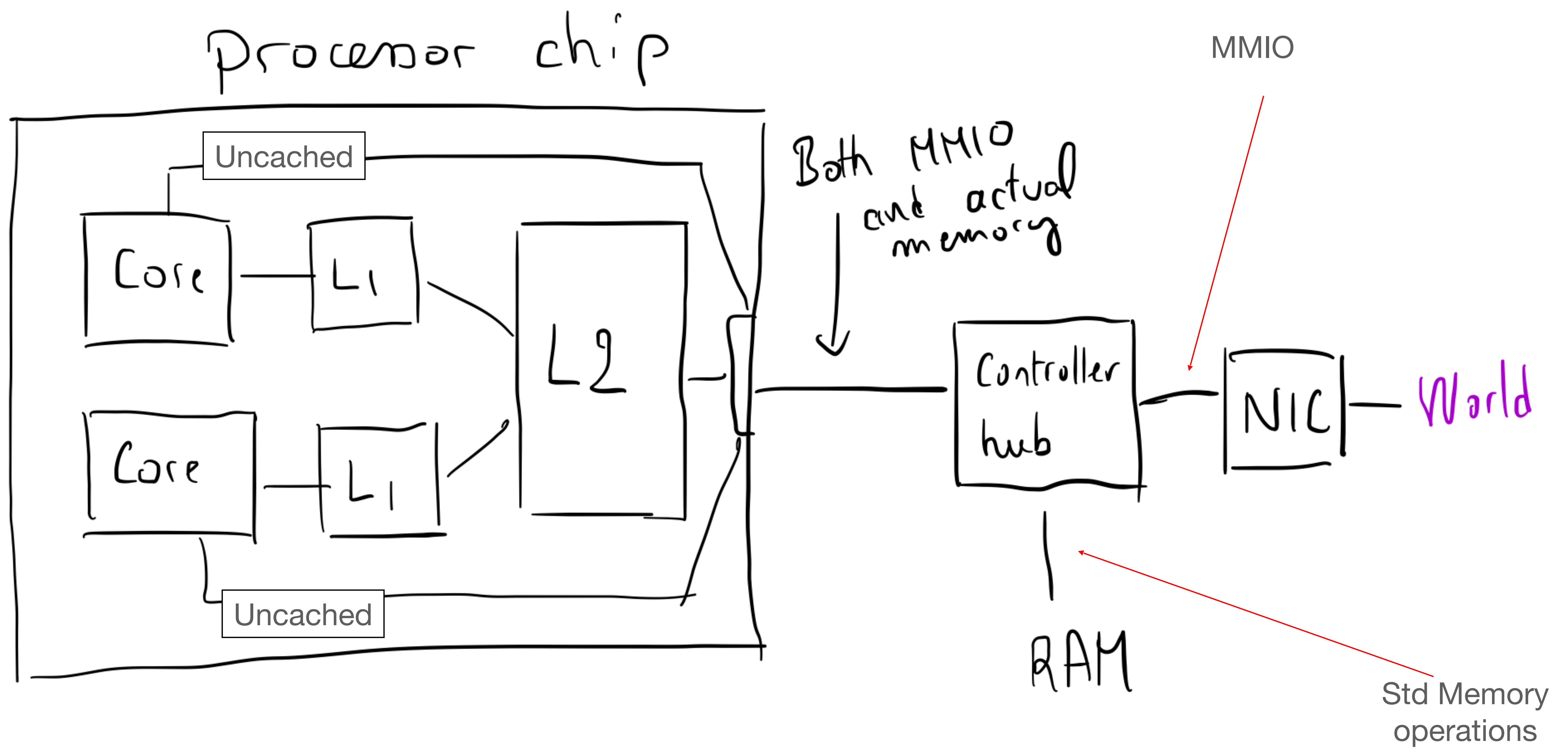
Limited programmability:

- my old ethernet card
- my old GPU
- my old sound card

Here could be a reasonable moment to do a break if we are around 45mn

Case study 2: a simplified-NIC

Approximate High-level picture – NIC



Simplified journey of a packet in a fancy NIC

Processor tells NIC:

- Address in RAM for SEND packets
- Address where NIC should put the RECEIVED packets

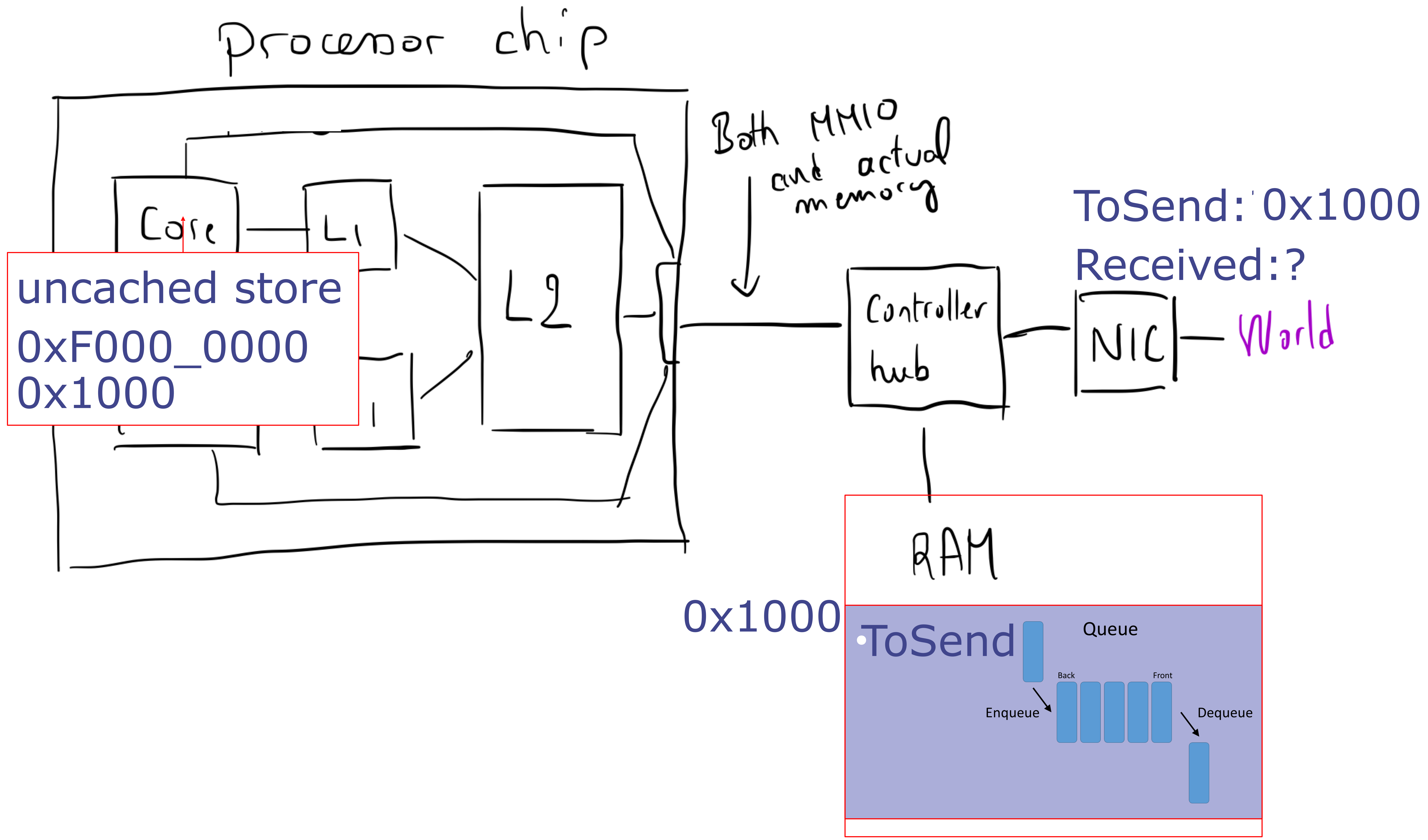
Application generate data to send:

```
“GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
…”
```

Wrap the string in an envelope with an address

Put the envelope in the SEND location

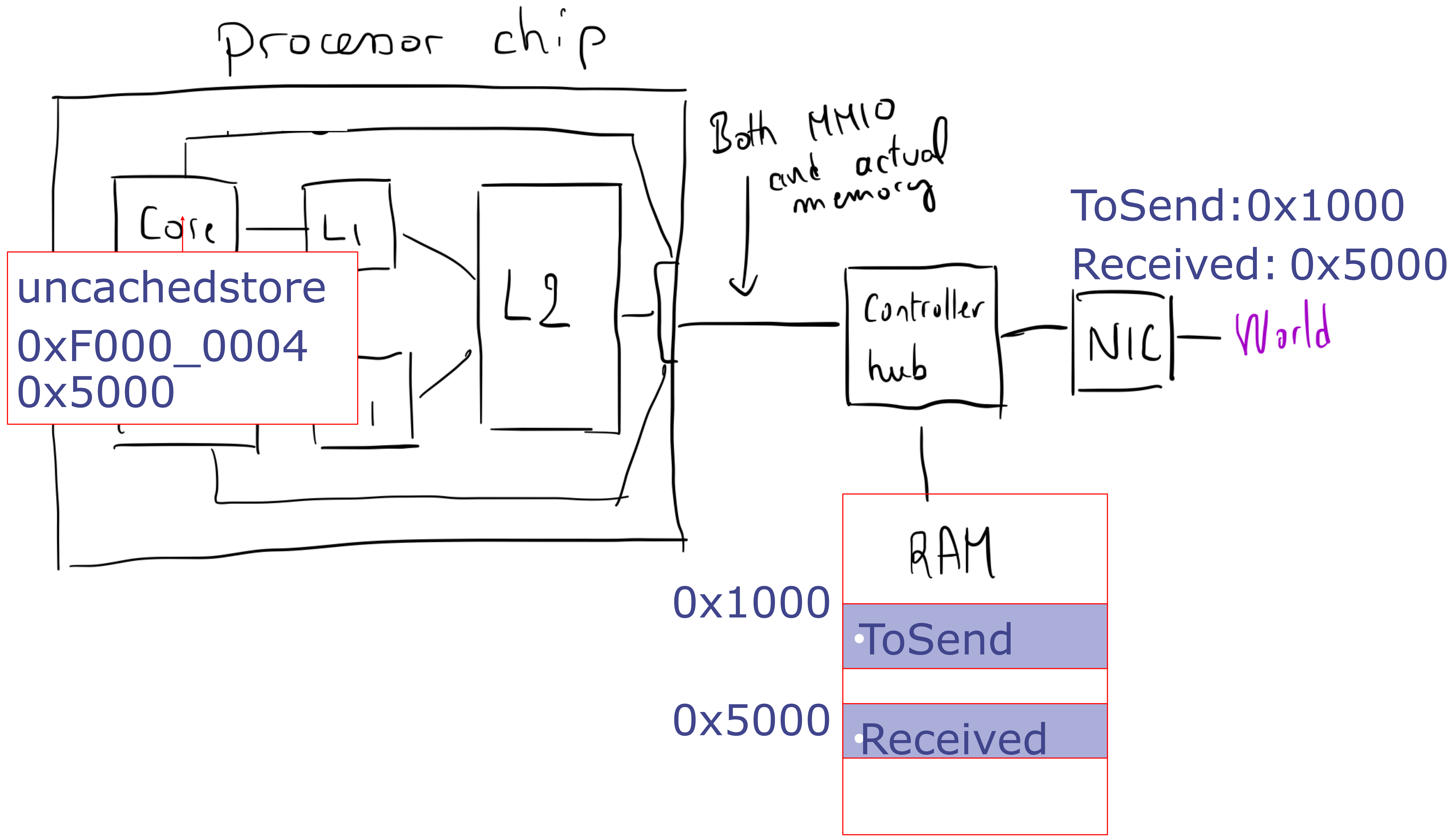
Agreeing on send locations



uncached store
0xF000_0000
0x1000

ToSend: 0x1000
Received:?

Agreeing on receive location

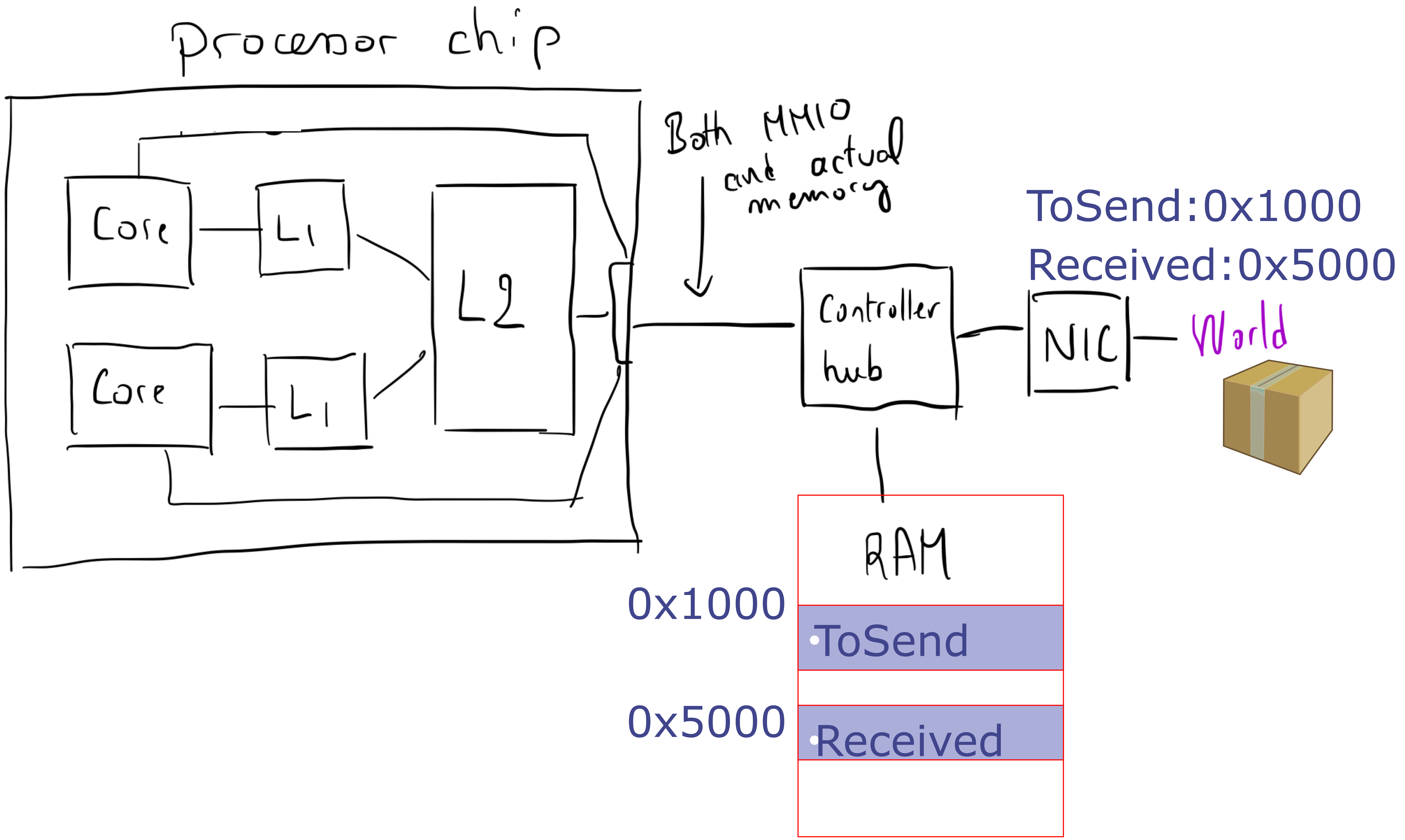


uncachedstore
0xF000_0004
0x5000

ToSend:0x1000
Received: 0x5000

0x1000	ToSend
0x5000	Received

Simplified journey of a received packet



The NIC-CPU system

ToSend and Receive are in-mem circular buffers:

CPU push into ToSend and pull from Received

NIC push into Received and pull from ToSend

The NIC, once configured, operate independently of the core

<u>The NIC performs computations – networking cost little CPU compute</u>

Arrivals and departures

The challenge of synchronisation

How does NIC know when something should be sent?

How does CPU know when something arrived?

Two solutions:

- Active polling – check every 1ms
- Doorbell mechanism:
  - CPU -> NIC: store to special location
  - NIC -> CPU: Interrupts

<u>Characteristic time: ~1us</u>

MMIO vs Shared memory

Through MMIO the processor sends commands and pointers

The data is not sent directly through MMIO, but through shared memory

Why MMIO to NIC must be uncached?

Why stores to ToSend must be uncached?

Using Devices from SW

Write a program to manage the shared buffers with the device:

- Allocate buffers
- Recycle buffers
- Send address of buffers to device
- Doorbell code, etc…

<u>Such a program is named a device driver!</u>

Case study 3 - Robomorphic Computing

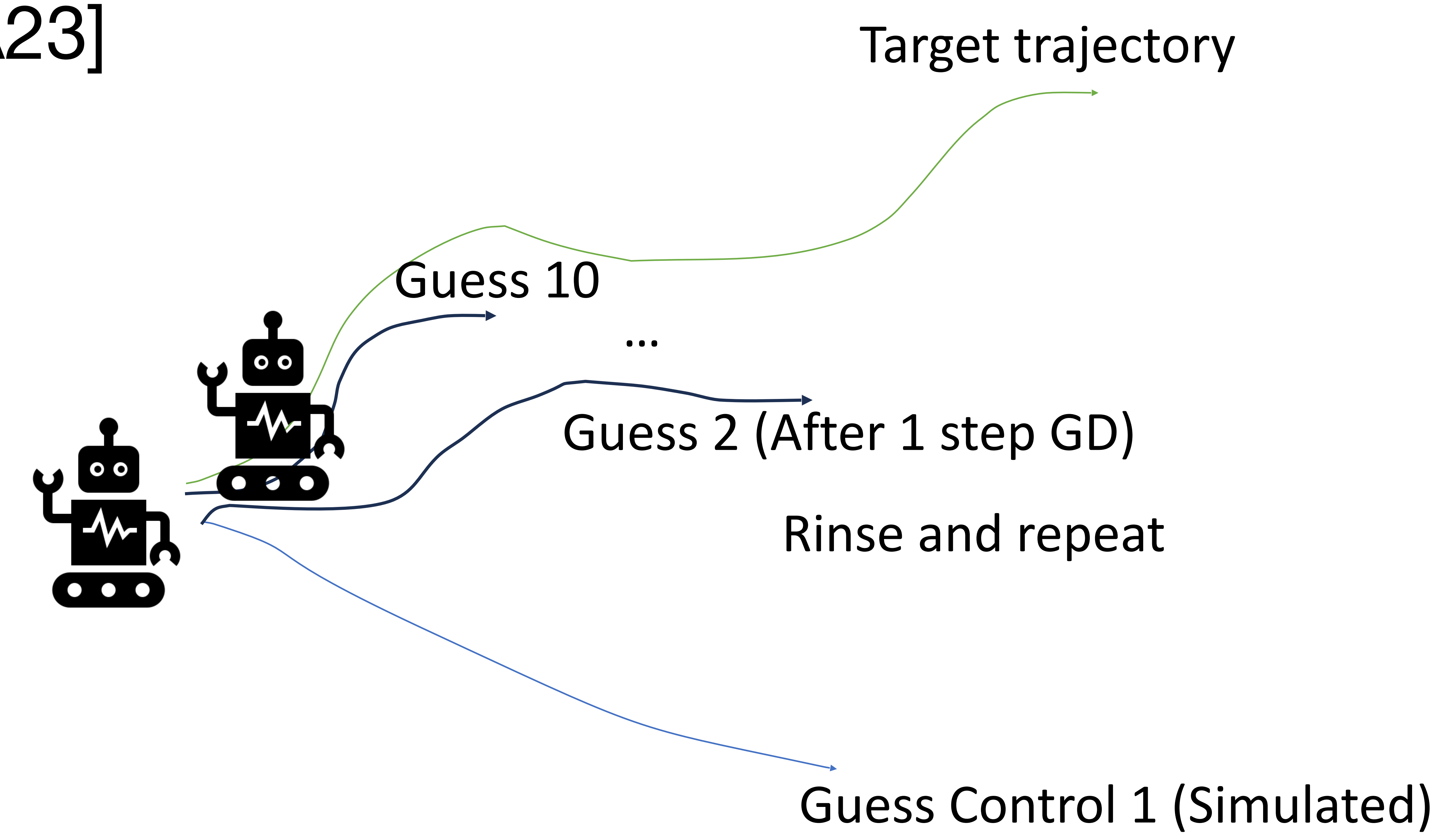
Memory-to-memory accelerators

Accelerator for robots

Sabrina Neuman, Radhika Ghosal, Brian Plancher & al

Domain Specific Computations (accelerators):

- Kernel computations for Model Predictive Control – [ASPLOS21, ICRA2021, ISCA23]



What kind of compute?

As a computer architect - don’t have to understand! Just need quantitative numbers

Control rate, roboticist friends recommend: 300Hz - 1KHz

Need to compute k-time ($10 < k < 100$) time steps in the predicted position of the robot + gradient

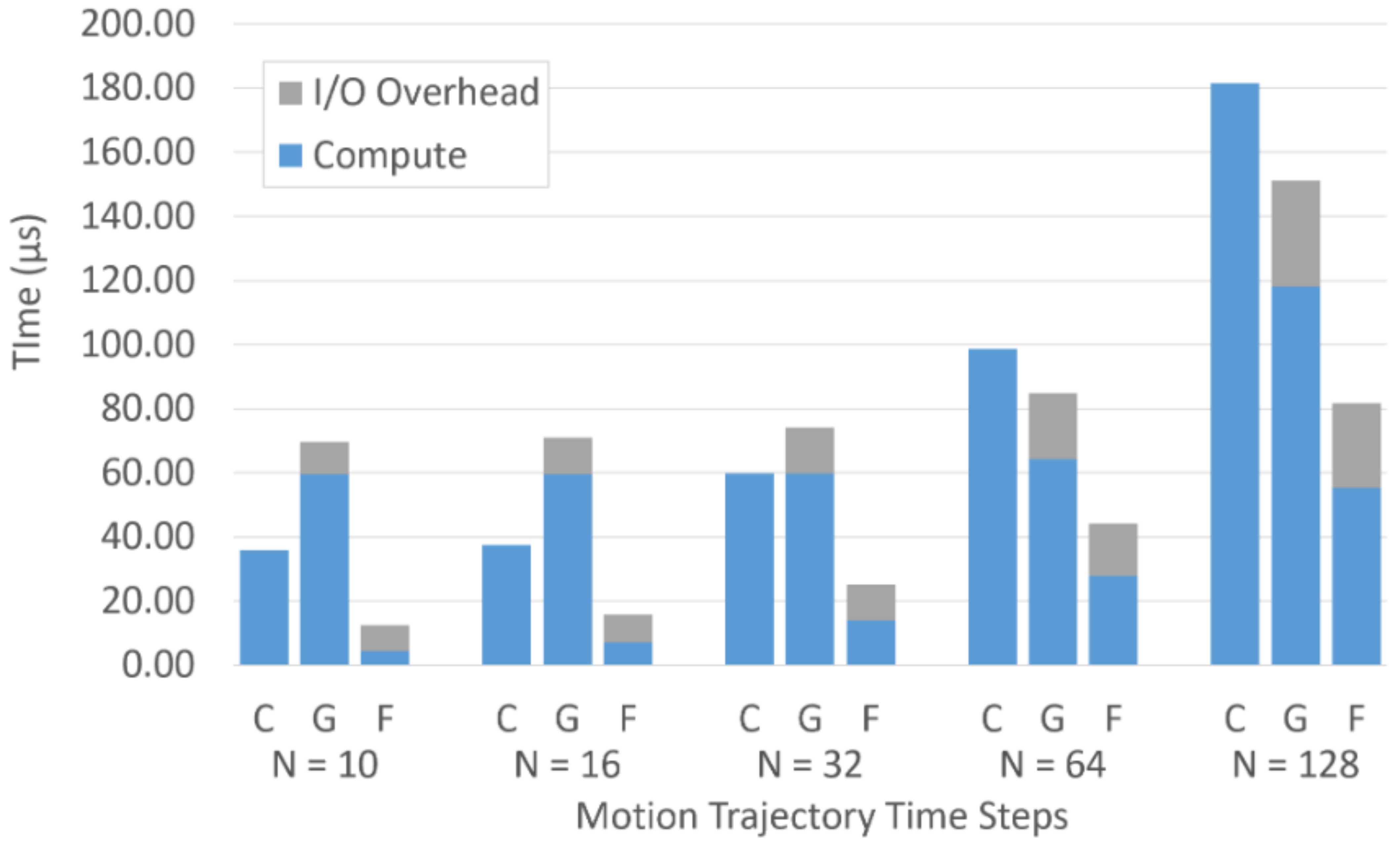
Need to descend the gradient ~3-20 steps

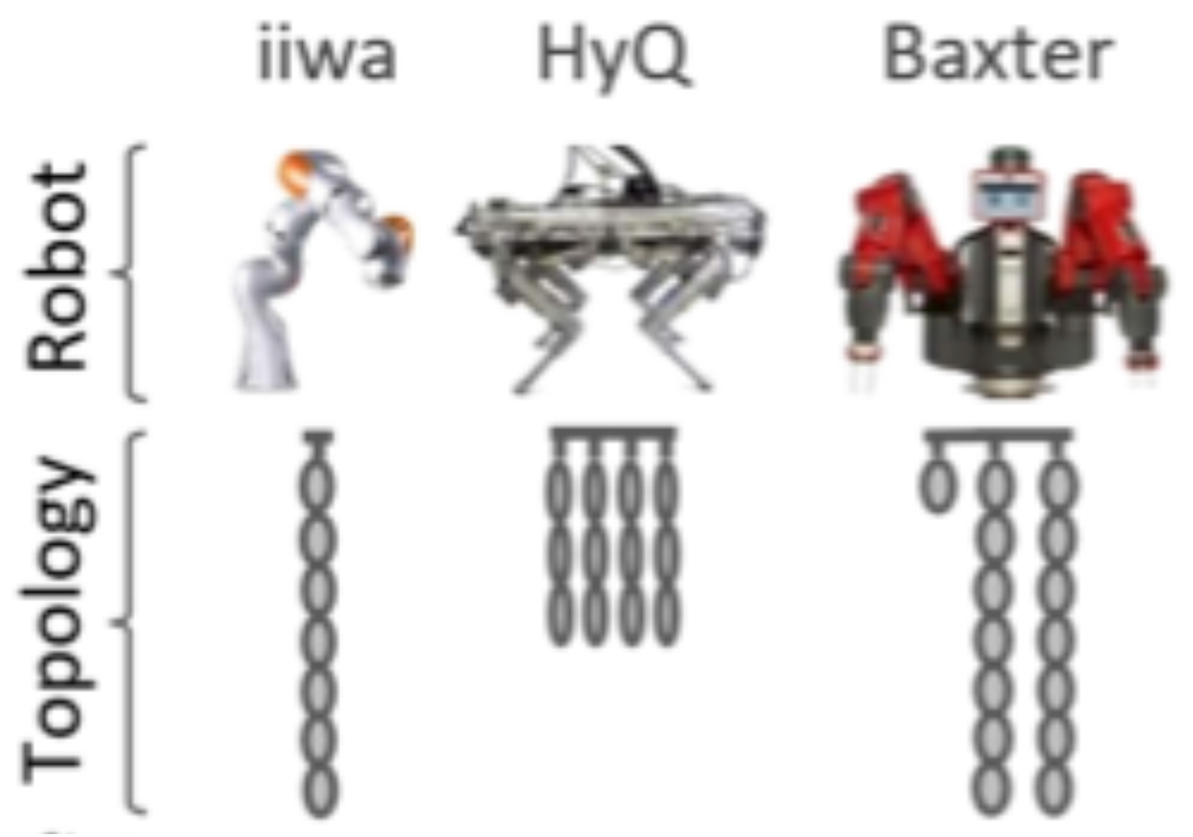
Surprise, surprise: boils down to matrix multiply, matrix adds

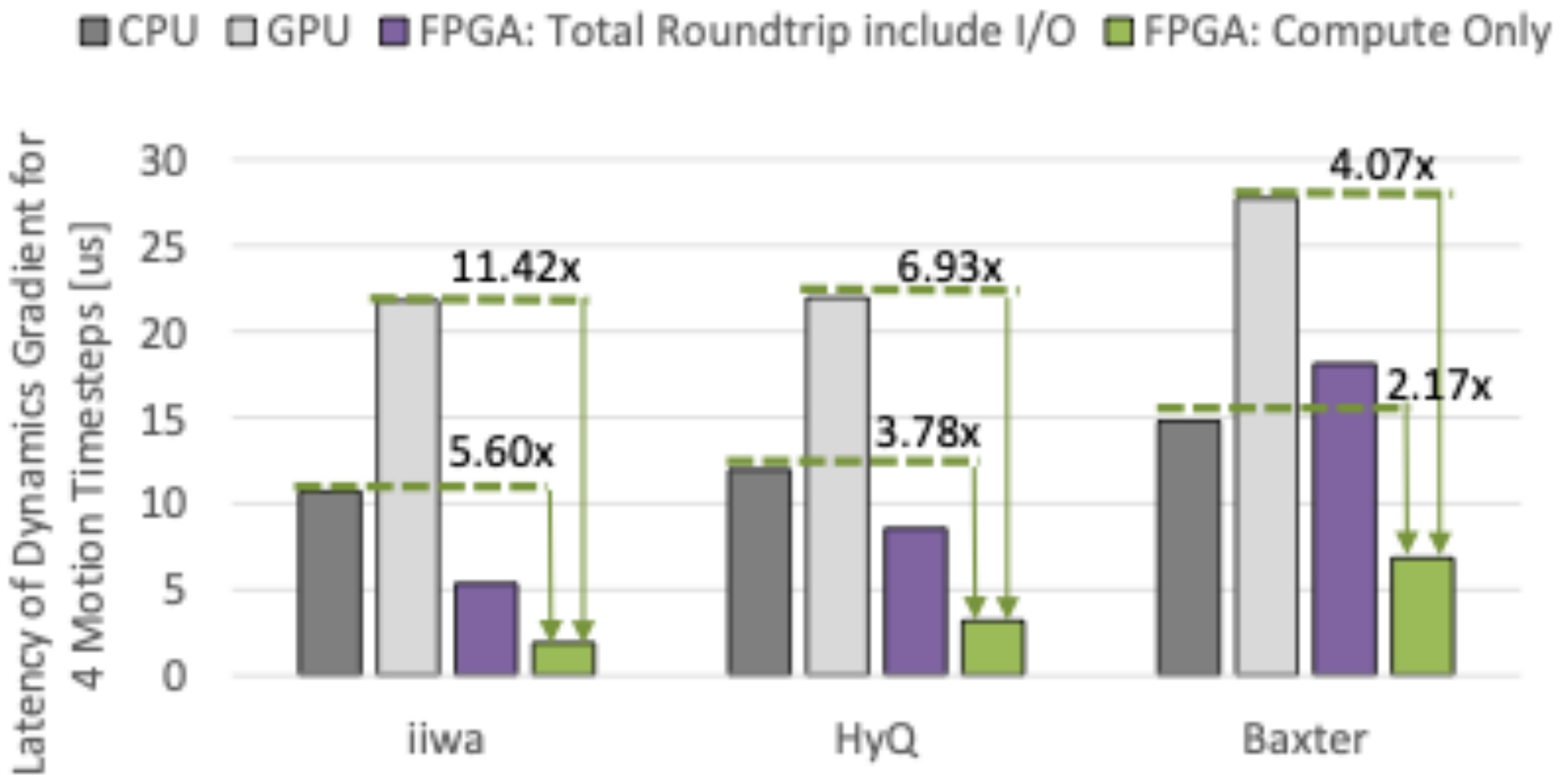
- All those matrices have a fix-structure that depends on topology of the robot
- FPGA has 6000 “multipliers”, how to make use of them most efficiently?

Prototyping an accelerator on FPGA (~10X faster clock in ASIC)

Performance







Performance considerations -

When accelerators can’t do miracles

Latency considerations

On-chip latency:

1 cycle (registers memory) -~10 clock cycles (last level cache)

Off-chip (e.g. PCIE) latency:

1us (~4000 cycles)

A whole lot of CPU work!

Throughput considerations

PCIE throughput (gen 2):

- 500MB/s per lane (up to 16 lanes)
- Reality: a few GB/s
- CPU/DRAM bandwidth ~5-10x more (60-150GB/s)
- Internal GPU bandwidth: ~1TB/s
  - GPU companies work hard to avoid the PCIE bottleneck

More modern PCIE: 1GB/s (resp. 2GB/s) per lane for Gen3 (resp. Gen4)

Throughput takeaways

FPGA being disappointing

If a CPU already manages to saturate memory bandwidth, an accelerator won’t go faster!

FPGA will typically go slower when because PCIE bandwidth < DRAM bandwidth ?

Of the importance of the memory system: “It’s the memory, stupid!”

Richard L. Sites, Digital Equipment Corporation (Microprocessor Reports, 1996)

Source: https://websrv.cecs.uci.edu/~papers/mpr/MPR/ARTICLES/101004.pdf

Some papers promise a better future for PCIE

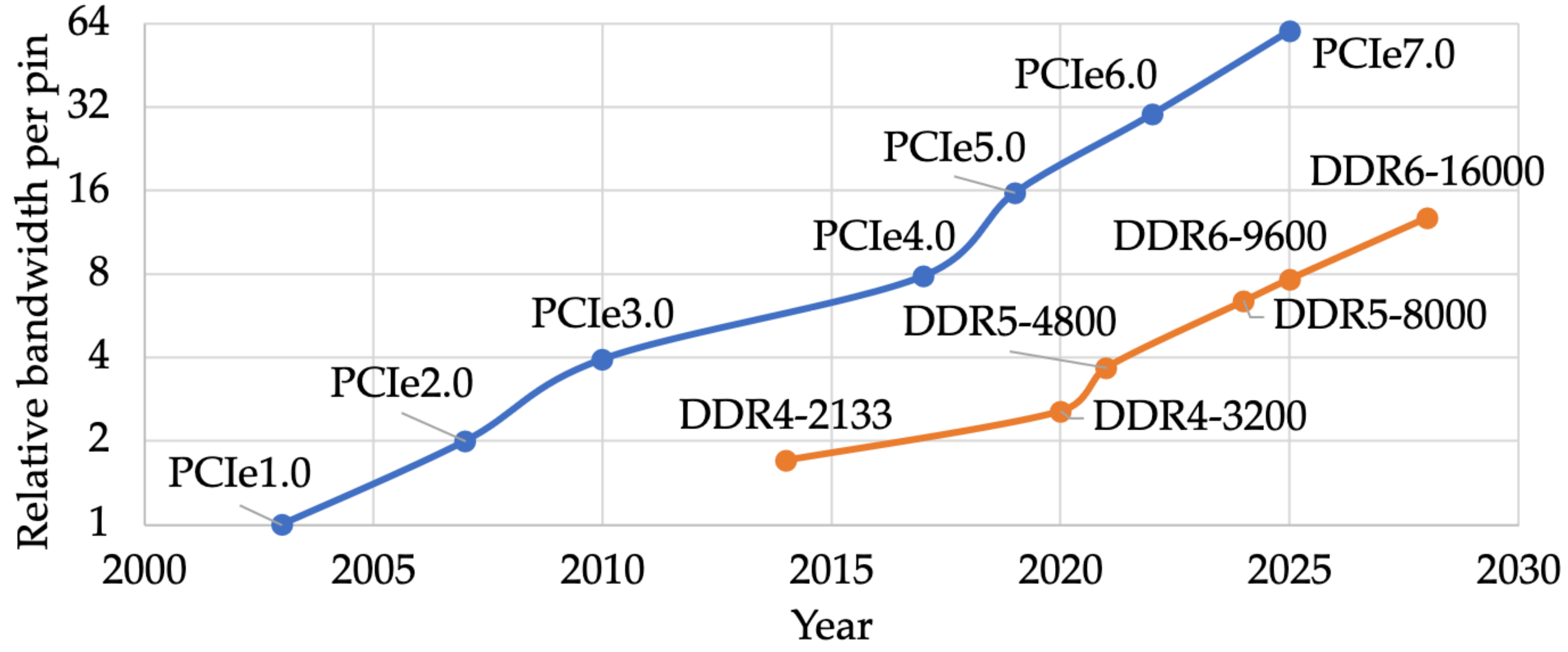


Figure 1: Bandwidth per processor pin for DDR and CXL (PCIe) interface, norm. to PCIe-1.0. Note that y-axis is in log scale.

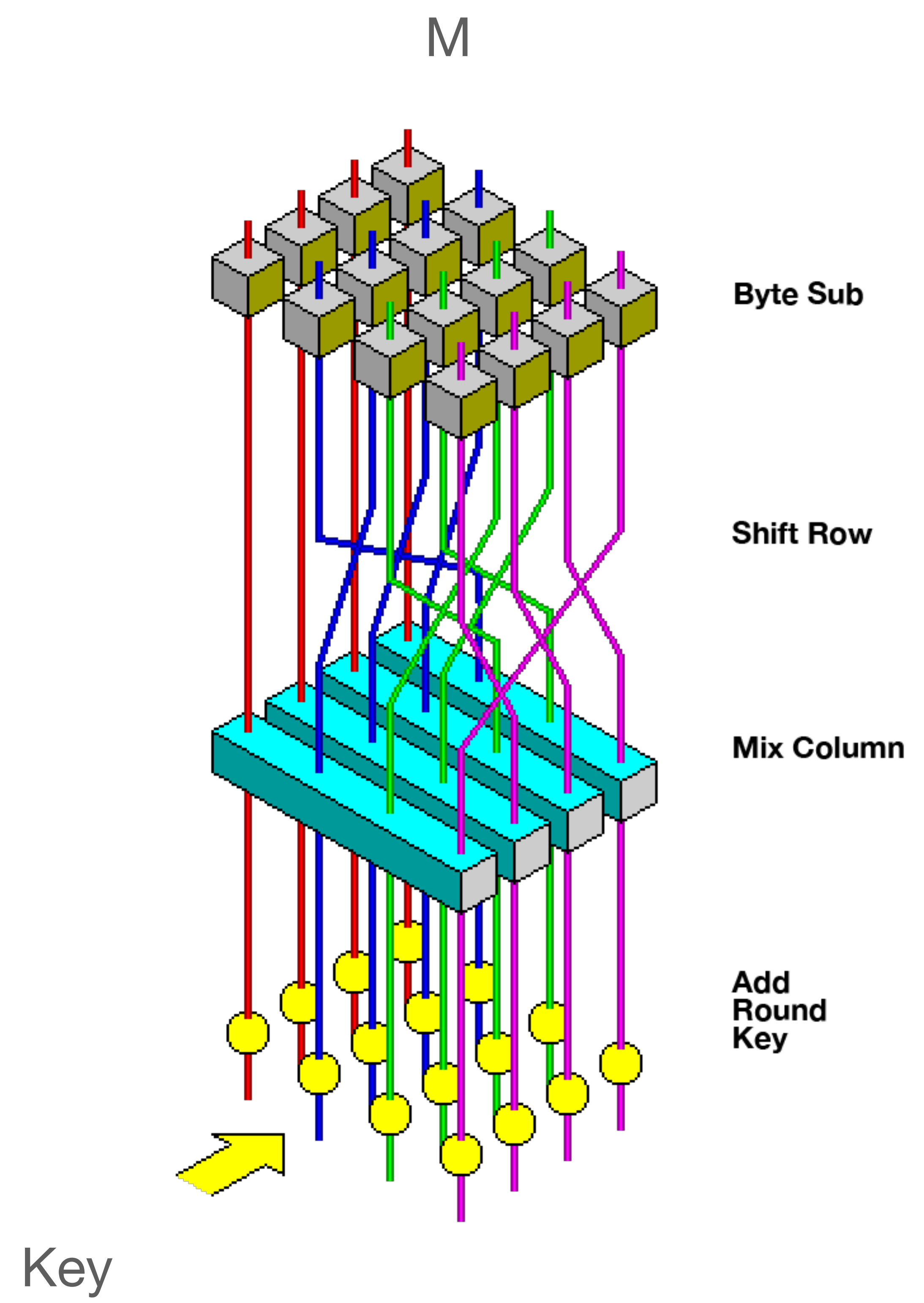
Principles of Accelerators

Guidelines for accelerators
(Borrowed from Bill Dally's talk)

1. Parallelism
2. Locality
3. Optimize Memory Orchestration and Control
4. Custom datatypes and operations

Parallelism

1. Look at the dataflow graph of your computation, at a fine granularity
2. What is the critical path of your computation, compared to the amount of compute nodes?
3. Does it make sense to pipeline the computation?



Locality - Arithmetic Intensity

Reuse the data you bring from memory multiple times!

Arithmetic Intensity is the ratio of arithmetic operations to data movement (bytes)

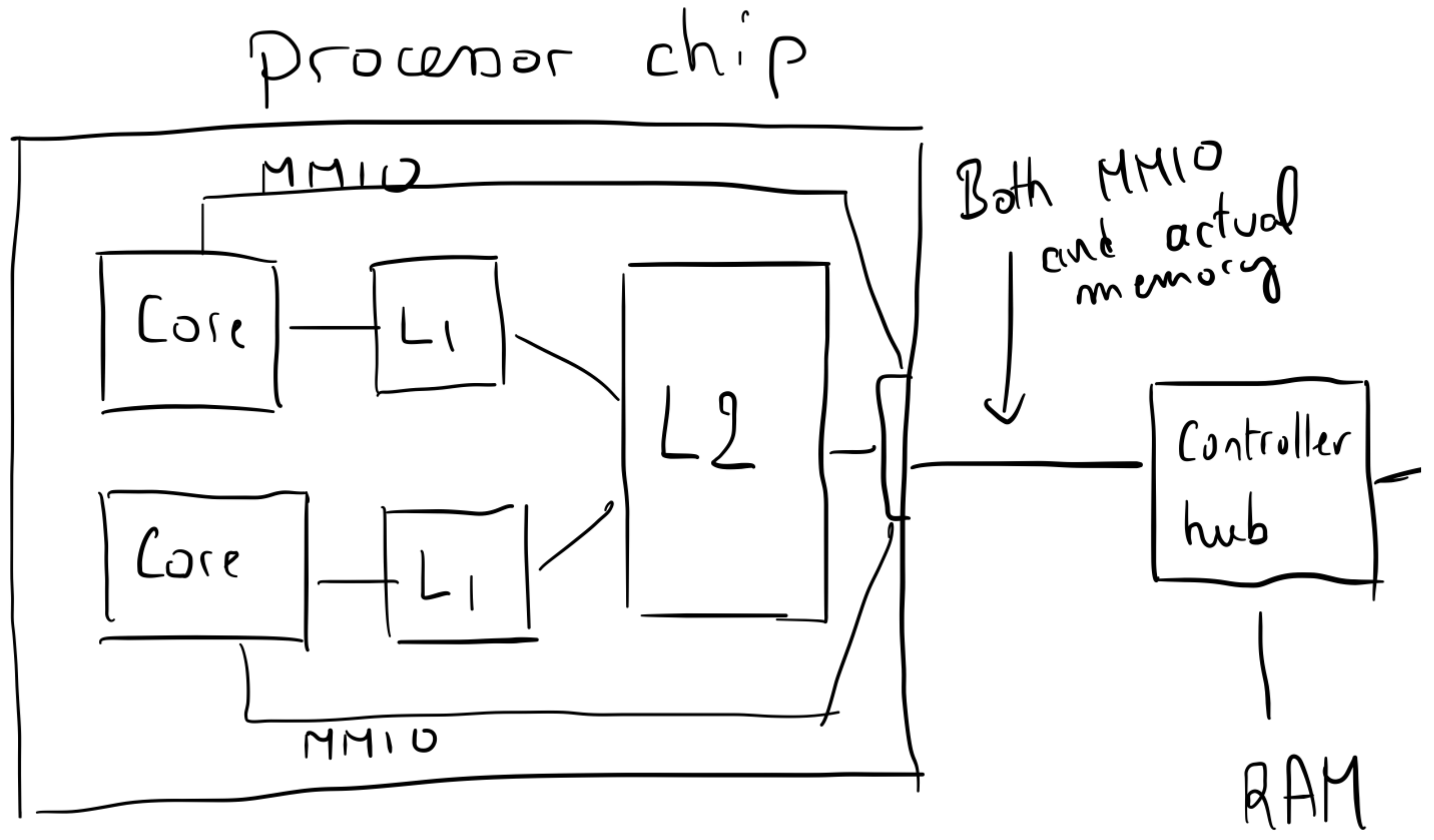
Low Arithmetic Intensity <-> Memory bottlenecked!

Dot product: not so high arithmetic density ~$\frac{m}{2m} = O(1)$

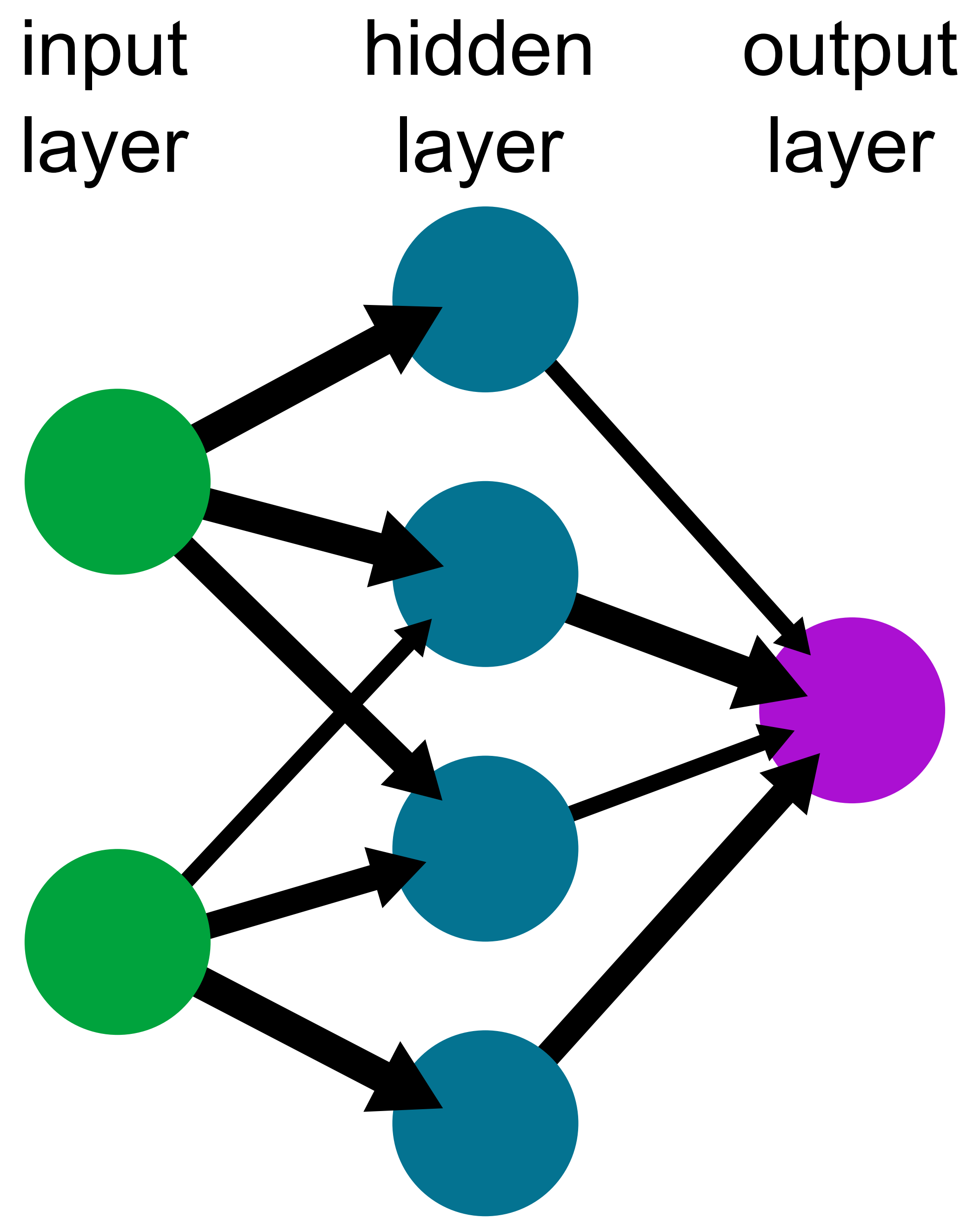
Matrix multiply: much higher arithmetic density ~ $\frac{m^3}{2m^2} = O(m)$

Optimize memory orchestration

Application specific "memory pipelining" / Decoupled execution



A simple neural network

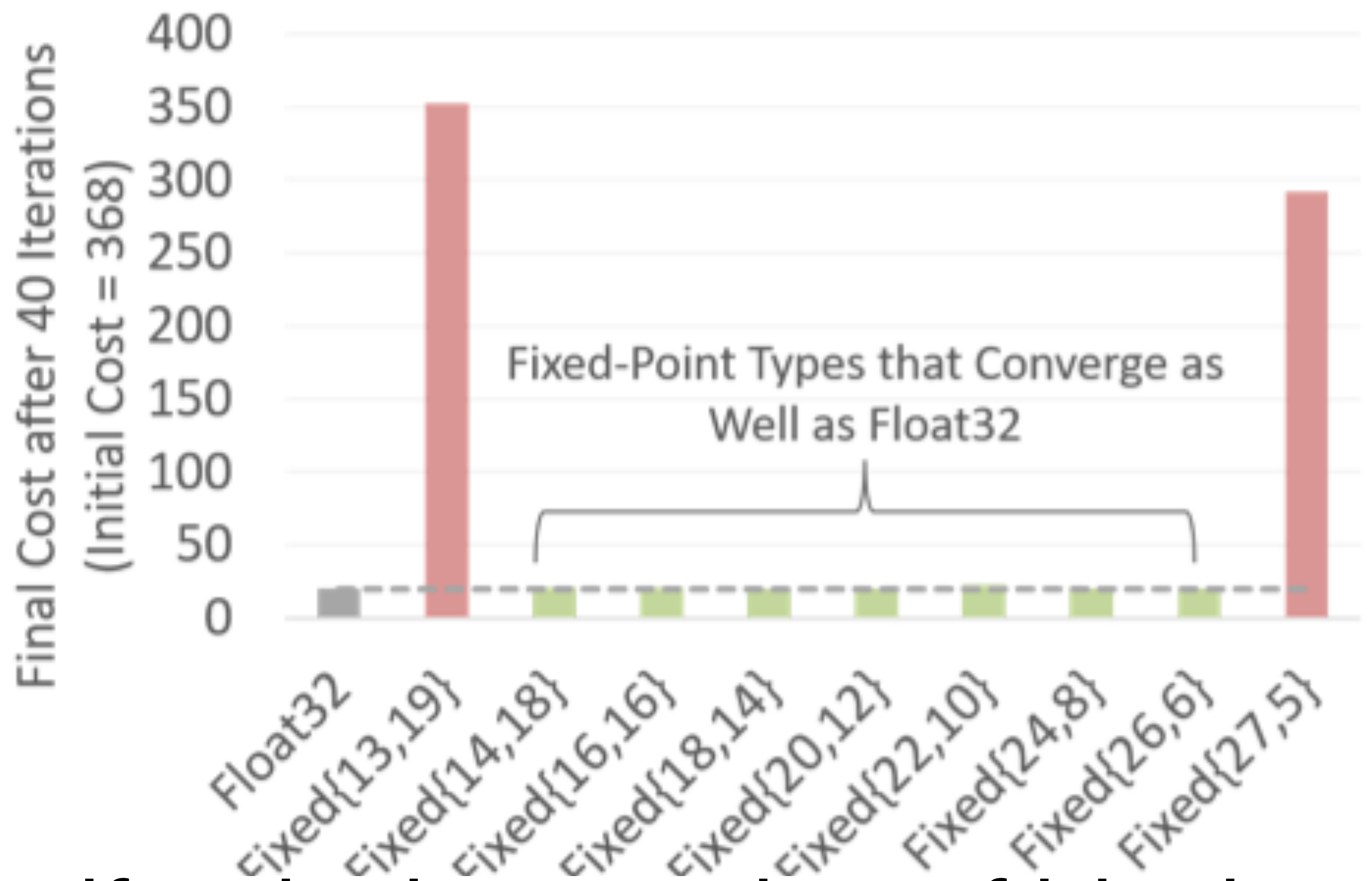


Specialized Datatypes

Software easily goes bloat when working on custom Datatypes with custom operations

Remember AES is expensive in SW because does weird things to 128 bits!

Domain specific analysis of structure of computation:



Modern ML -> lot to gain if reducing number of bits in representation

Joker:
CPU can achieve max speed (random access in large memory/streaming from large memory),
Accelerator = Energy Savings

The Performance Cost of Specialisation

Drawback of Old School HW/SW codesign

Take application X, handwrite code that leverage Accelerator W (V1), performs great

[…] Time elapse

10 years later, Accelerator W (V13)

-> Handwritten code performs poorly (when it works)

Software Integration

The elephants are in the room

Having defined new instructions is not the end of the story:

- If I add AESENC, my libssl library won't suddenly start using it
- Have to worry about cross-platform

If I have matmul 16x16, or matmul (nxm forall n,m<64),

- 512x512 matmul?
- convolution?
- Arbitrary linear/tensor algebra operator

Compilation challenges

The elephants are in the room

All things considered standard compilation is easy:

- Usually not too many different ways to compile

With Domain Specific, program space is typically very complicated

- A*(B*C)*A or (A*B) * (C * A) …
- Compiler must find good sequence of instructions modulo rewrites!
- Every domain has its own set of rewrite - every time requiring a new compiler
- Compiler must consider an accurate cost model of memory!

Compilation – Sync issues

The elephants are in the room

Sync issues:

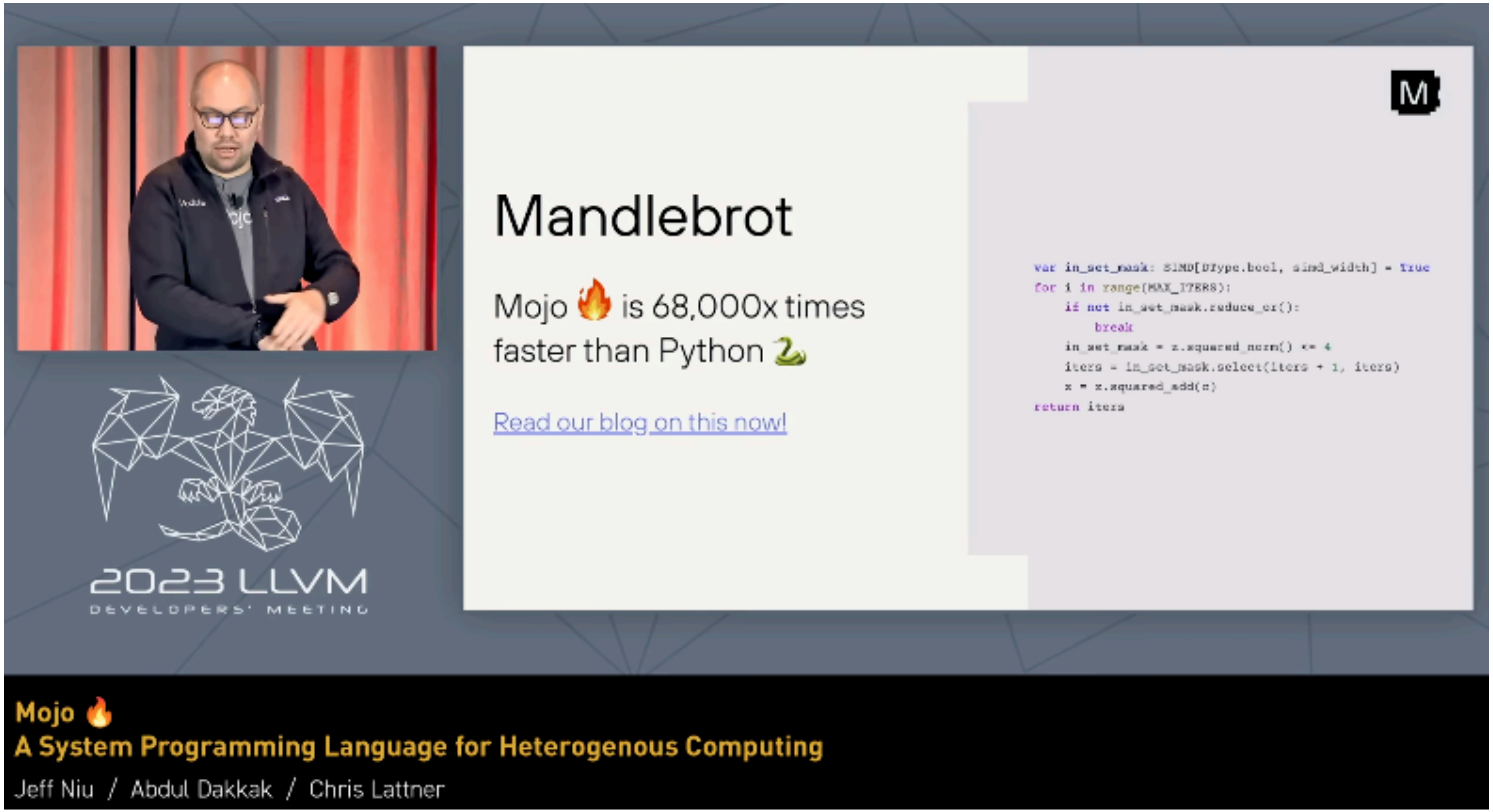
- Need to add explicit data movement instructions if I want to use the data computed on CPU

Hot research ideas:

- Decoupling functionality and scheduling/performance [Halide/Exo]
- Maybe not a fully push-button compilation, more an “assistant-compiler”
- Maybe enable user to augment the compiler - easily add transformations with triggers

Mojo/MLIR - the future of compilers?

https://www.youtube.com/watch?v=SEwTjZvy8vw



Mandlebrot

Mojo 🔥 is 68,000x times faster than Python 🐍

Read our blog on this now!

```
var in_set_mask: SIMD[DType.bool, simd_width] = True
for i in range(MAX_ITERS):
    if not in_set_mask.reduce_or():
        break
    in_set_mask = z.squared_norm() <= 4
    iters = in_set_mask.select(iters + 1, iters)
    z = z.squared_add(c)
return iters
```

Mojo 🔥

A System Programming Language for Heterogenous Computing

Jeff Niu / Abdul Dakkak / Chris Lattner

Conclusion

“Accelerators" and Codesigns are not new - Yesterday they enabled census, today they are enabling AI

There are performance costs to Abstraction

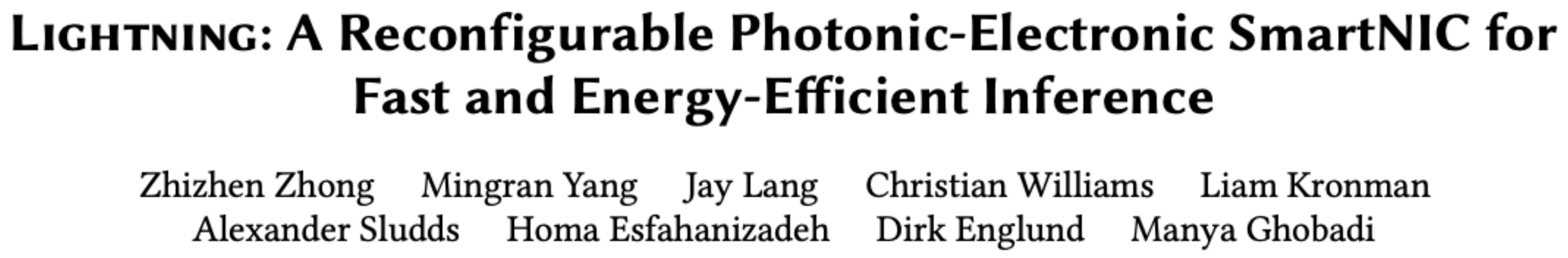
There are good guidelines/models for what can be accelerated

There are hidden performance costs to specialization

There is still work to do to better understand those costs in general

Accelerators 100 years later

SIGCOMM 2023



LIGHTNING: A Reconfigurable Photonic-Electronic SmartNIC for Fast and Energy-Efficient Inference

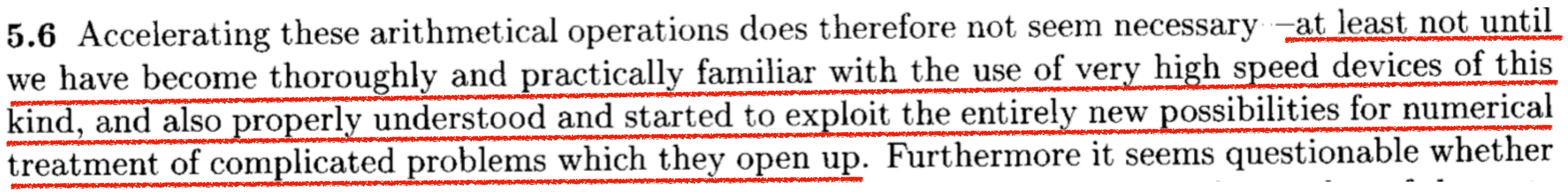
Zhizhen Zhong Mingran Yang Jay Lang Christian Williams Liam Kronman
Alexander Sludds Homa Esfahanizadeh Dirk Englund Manya Ghobadi

2.1 Photonic Vector Dot Product

Same good old $\sum_i w_i . x_i$, with photons

The Missing Quote

Von Neumann's condition



5.6 Accelerating these arithmetical operations does therefore not seem necessary —at least not until we have become thoroughly and practically familiar with the use of very high speed devices of this kind, and also properly understood and started to exploit the entirely new possibilities for numerical treatment of complicated problems which they open up. Furthermore it seems questionable whether

Sources and Inspiration

Bill Dally :

https://www.youtube.com/watch?v=fnd05AeeFN4

David Patterson/John Hennessy Turing Award Speed:

https://www.youtube.com/watch?v=3LVeEjsn8Ts

Chris Lattner & al. , LLVM Dev Mtg (10 days ago):

https://www.youtube.com/watch?v=SEwTjZvy8vw