Building Abstractions at the Hardware-software Boundary

HELLO CPPNorth

Andrew Bitar

Compiler Technical Lead & Manager
abitar@groq.com

MY GOAL TODAY

How should we approach building **abstractions** in a computing era beyond CPUs?

turing lecture

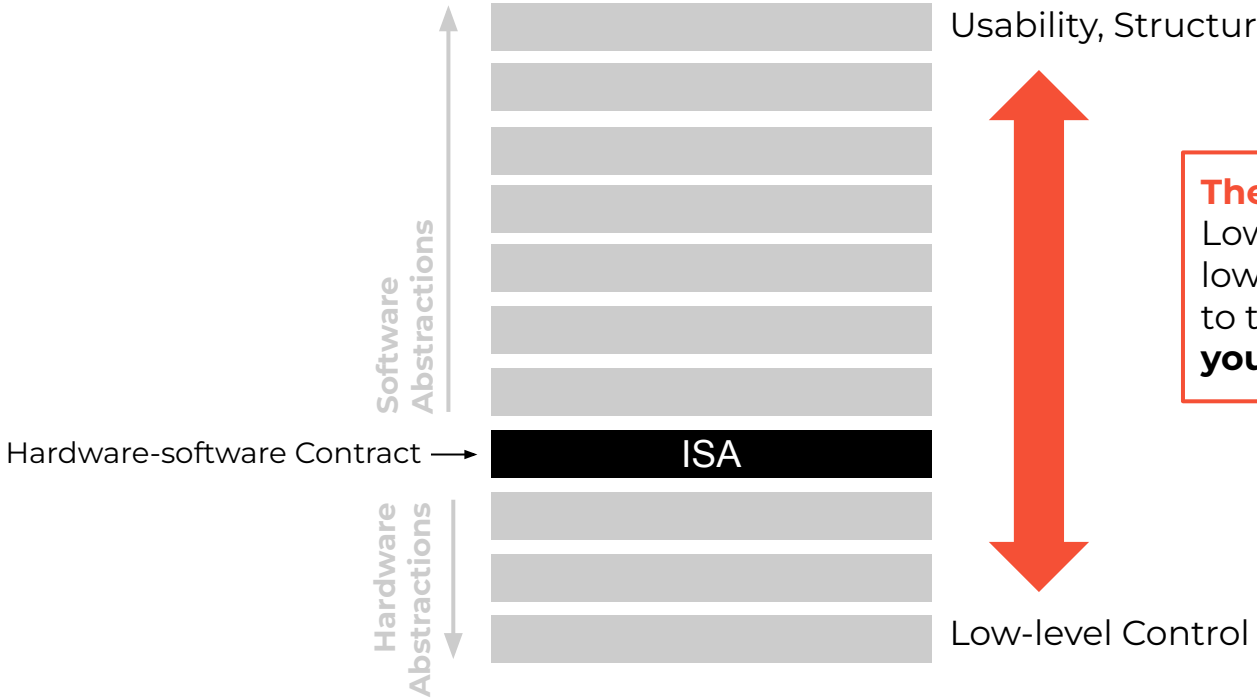
In Computing, Abstractions Are **Ubiquitous**

IN COMPUTING

Abstractions Are Ubiquitous

Applications	Functions, types, inheritance, template metaprogramming
Programming Languages	Domain-specific languages (e.g. PyTorch, TensorFlow, P4) Functional programming languages (e.g. Haskell) Scripting languages (e.g. Python) Augmenting lower-level languages with higher-level constructs (C → C++ → C++20)
Compilers	Multi-pass compilers, LLVM, MLIR Gradual IR “lowering” from higher-level representations down to machine language
Hardware	CAD flows (HDL → Tech-Mapped Netlist → P&R) Logic design (block hierarchies, interfaces)

The Abstraction Trade-off



The Abstraction Contract:
Lower-level abstractions hide lower-level details, and promise to take care of them **better than you could**!

The Abstraction Trade-off

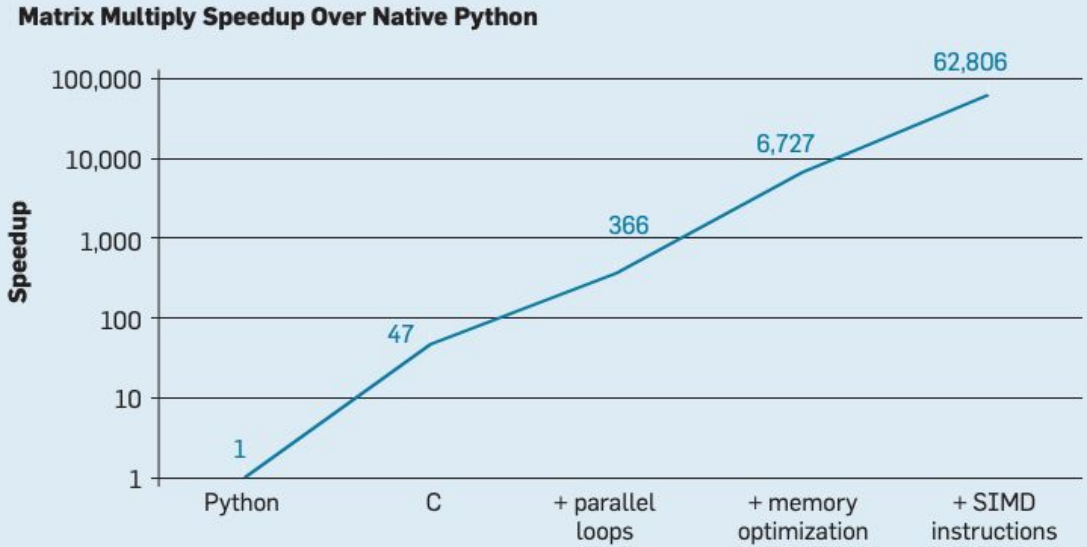
Abstractions rarely come free

"There are no zero-cost abstractions."
Chandler Carruth, CppCon'19

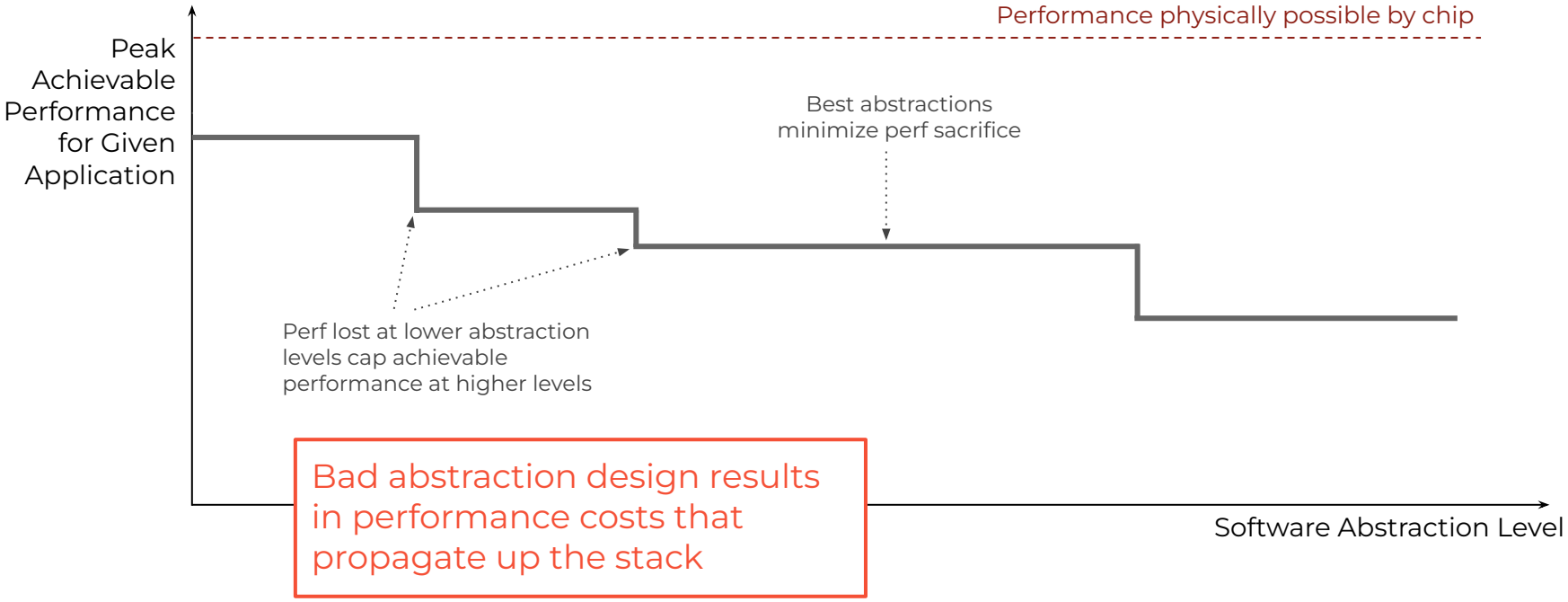
Even leveraging an abstraction as simple as `std::unique_ptr can` come at a runtime cost!

Hennessy and Patterson. A New Golden Age for Computer Architecture. **ISCA'18.**

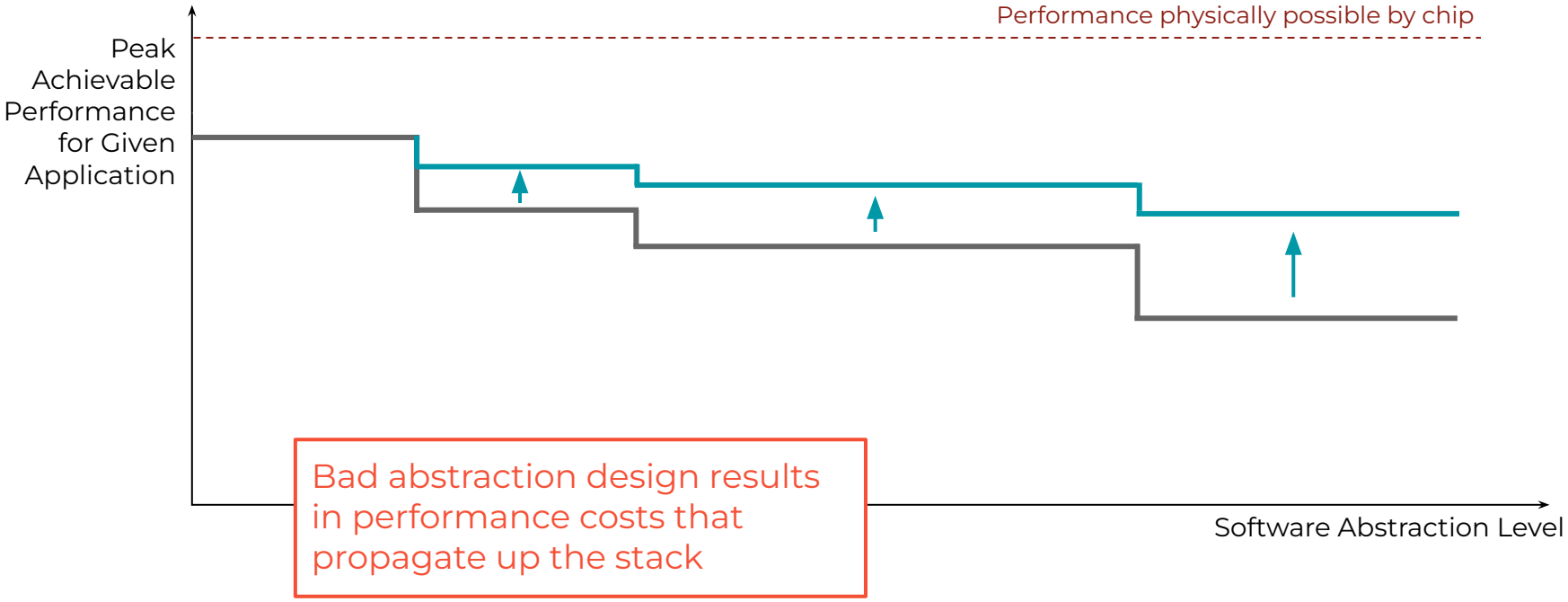
Figure 7. Potential speedup of matrix multiply in Python for four optimizations.



The Abstraction Trade-off

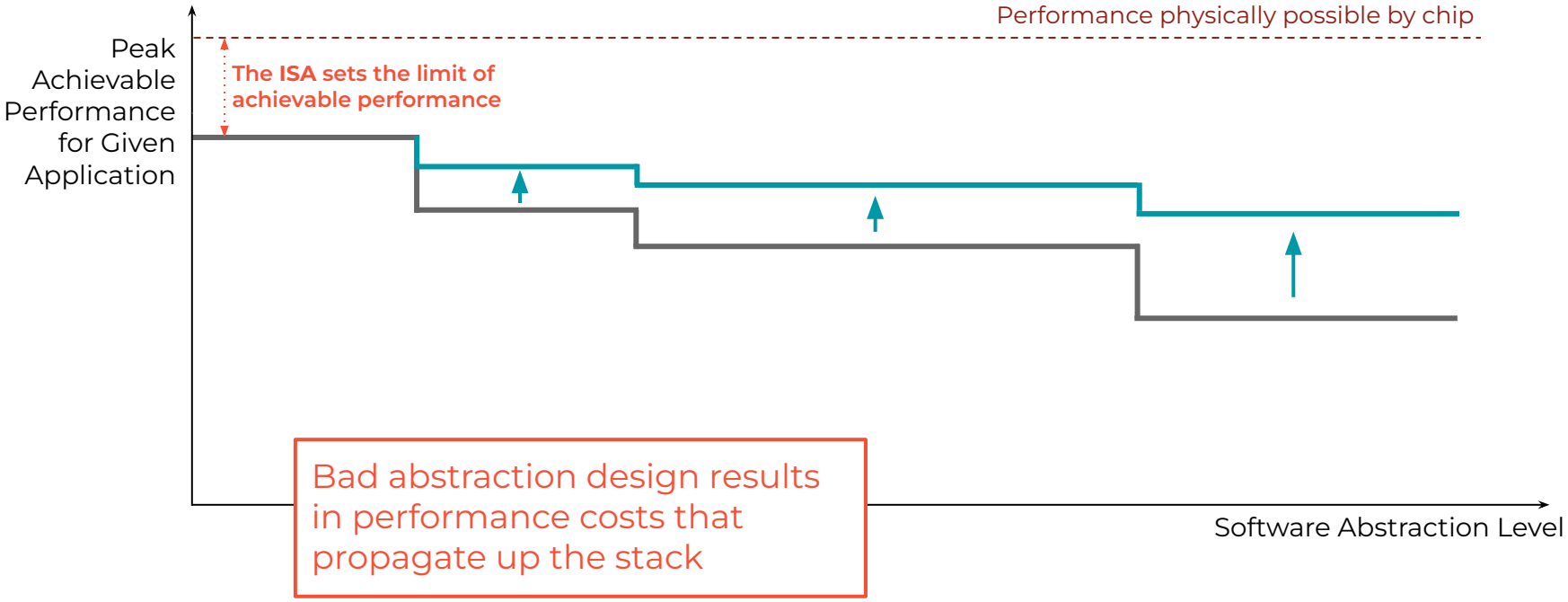


The Abstraction Trade-off

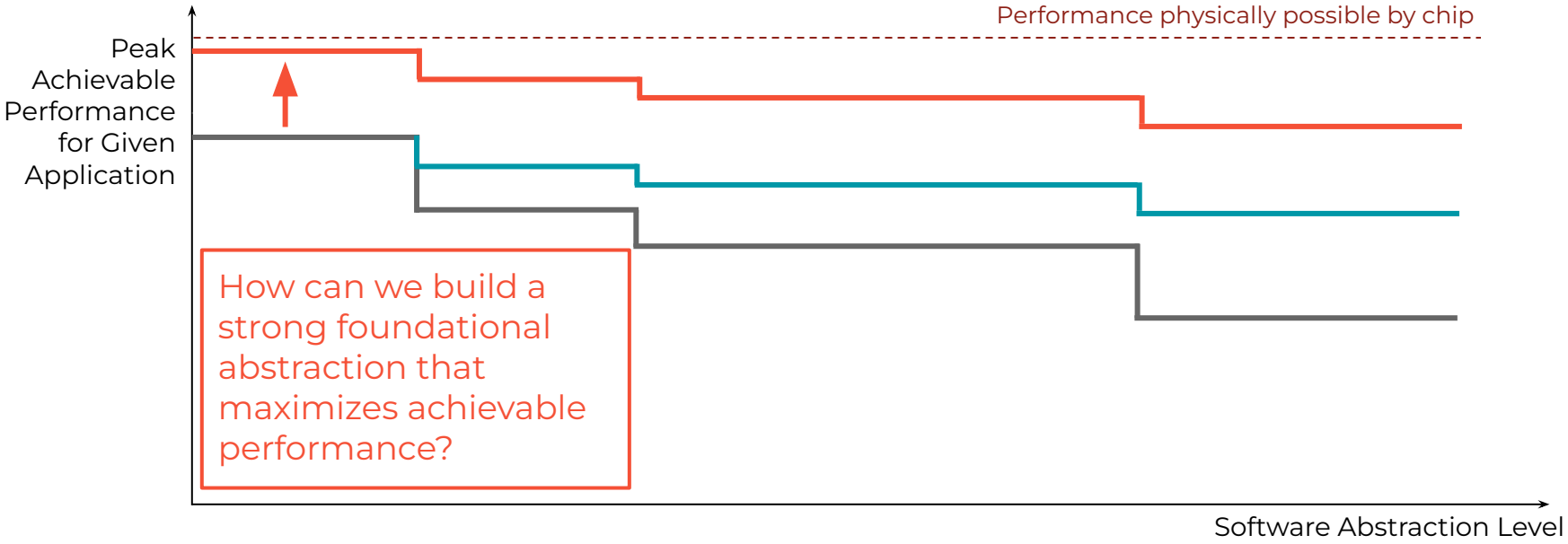


Bad abstraction design results in performance costs that propagate up the stack

The Abstraction Trade-off



The Abstraction Trade-off



ARCHITECTURE

The Bedrock of Abstractions

The First Golden Age

For computer architecture

Complex Instruction Set Computer (CISC) was born in an era when programs were primarily written manually using assembly languages

Complex instructions that performed multiple low-level operations (e.g. load + ALU + store) helped **raise the abstraction level** for assembly writers while also reducing instruction memory footprint

With the advent of high-level languages, the question changed:

- "What assembly language would programmers use?" became "What instructions would compilers generate?"

CISC → RISC

Computer architects began observing that using a Reduced Instruction Set Computer (RISC) improved performance and efficiency

Moving from CISC to RISC **lowers the abstraction level of the ISA**

- Moving complex control logic from hardware to software → more chip area available for compute and memory

As noted in the Hennessy & Patterson paper - experiments showed CISC ISAs using 75% of instructions per program, but executed 5–6 more cycles per instruction → **overall RISC was 4x faster**

- Software able to make better decisions when given lower-level control of the hardware

RISC ➔ VLIW?

Even with RISC, CPU architecture hides a lot of complex control logic from software

Caching, prefetching, out-of-order execution, branch prediction

- Area cost of this control hardware reduces available area for raw compute

Very Long Instruction Word (VLIW) to the rescue?

The next natural step in the trend of shifting work and control from hardware to software

- Execution ordering, data prefetching, branch resolution
- Compiler responsible for exploiting instruction-level parallelism (ILP) available in a program

Itanium…

Compiler struggled to achieve good ILP due to limited view of dynamic hardware behaviour

- Cache misses, dynamic mem dependencies, dynamic branches

The Hardware Lottery

In 2012, AlexNet was the breakthrough for Deep Neural Networks

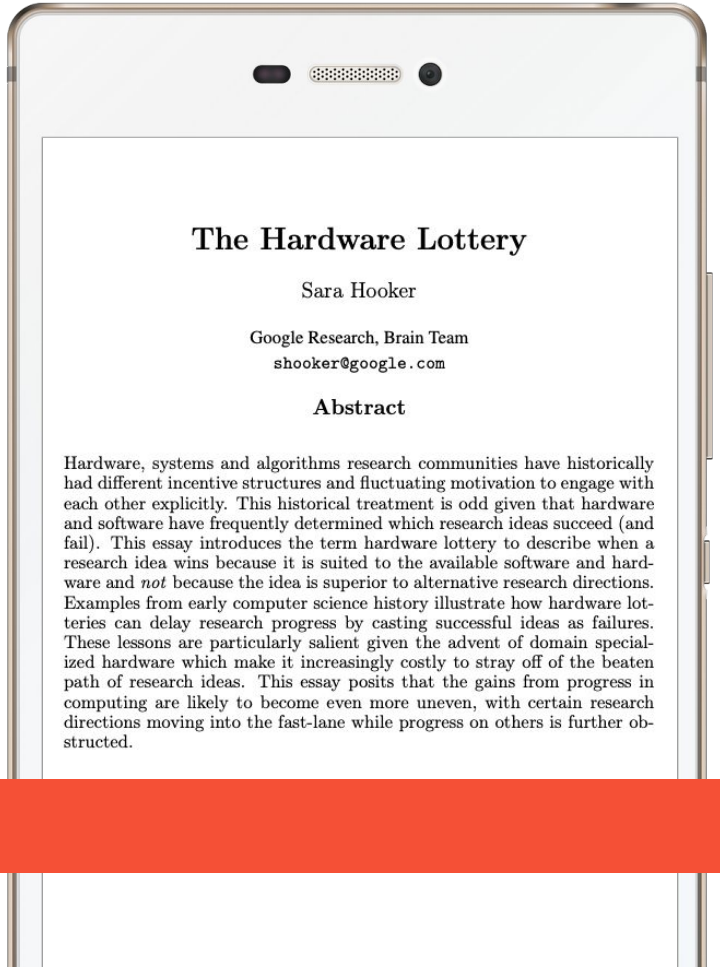
Convolutions and Backprop have existed since the 1980s

But CPUs were poorly suited for DNNs

- CPUs pay significant chip area for handling control-flow dominated programs
- DNNs are data-flow heavy workloads, with little-to-no dynamic control-flow

Evolution of GPUs in the 2000s opened up an architecture better suited for dataflow-heavy workloads like DNNs

Reduce dedicated control-flow logic = increased chip area dedicated to data-flow compute (i.e. fixed- and floating-point multipliers and adders)



The Hardware Lottery

Sara Hooker

Google Research, Brain Team
shooker@google.com

Abstract

Hardware, systems and algorithms research communities have historically had different incentive structures and fluctuating motivation to engage with each other explicitly. This historical treatment is odd given that hardware and software have frequently determined which research ideas succeed (and fail). This essay introduces the term hardware lottery to describe when a research idea wins because it is suited to the available software and hardware and *not* because the idea is superior to alternative research directions. Examples from early computer science history illustrate how hardware lotteries can delay research progress by casting successful ideas as failures. These lessons are particularly salient given the advent of domain specialized hardware which make it increasingly costly to stray off of the beaten path of research ideas. This essay posits that the gains from progress in computing are likely to become even more uneven, with certain research directions moving into the fast-lane while progress on others is further obstructed.

Watch Sara Hooker's GroqDay talk: groq.link/gd2hwlottery

GPUs ➔ GPGPUs?

What started as a Domain-specific Accelerator (DSA) began to change to a more general-purpose processor

Moving some control-flow responsibilities to software [e.g. execution order, prefetching]

- Didn't we try this with VLIW? What's changed?

Emergence of Software 2.0 [see: Andrej Karpathy] ➔ data-flow dominated workloads as better alternatives to solving problems

A New Golden Age in Computing

The end of CPU hegemony

Growing demand in dataflow dominated compute

Slowing of Moore's Law and Dennard Scaling

CPU "abstraction" no longer the only foundation for developing software

Dawn of a new golden age in computing

Hennessy & Patterson: "A New Golden Age for Computer Architecture"

Lattner: "A New Golden Age of Compilers"

Karpathy: "Software 2.0" → A New Golden Age for Algorithms

In the past, hardware (CPUs) has defined our software

The hardware-software abstraction contract has been reopened!

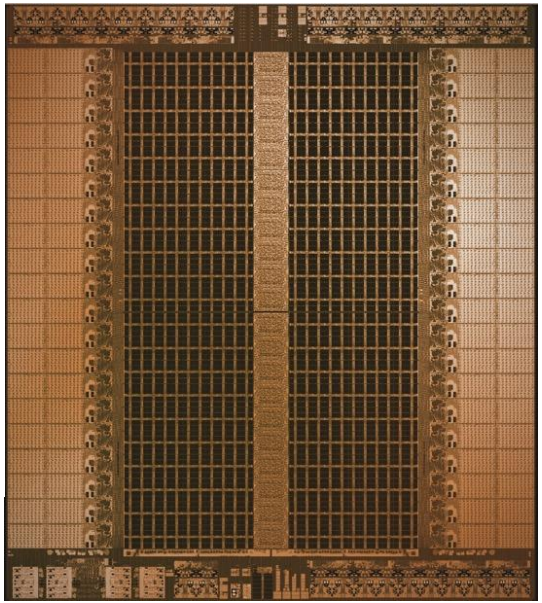
Opportunity now to achieve "software-defined hardware"

Software-defined Hardware

Groq is Radically Simplifying Compute

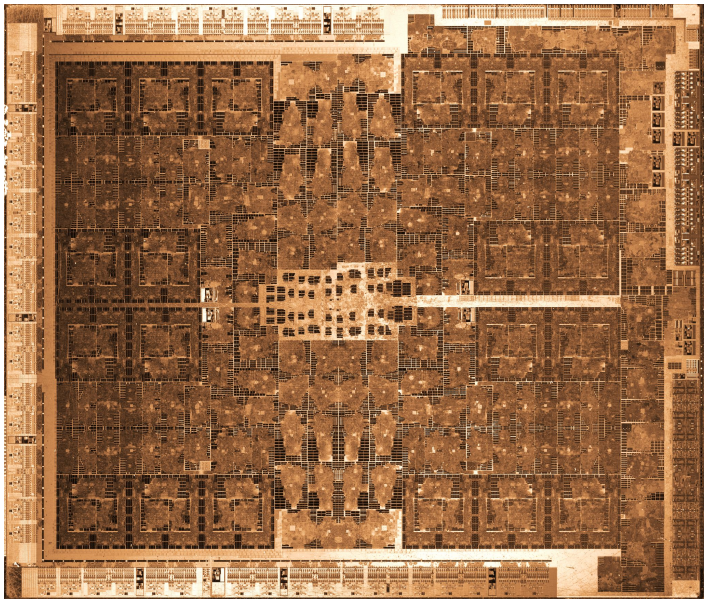
Less hardware control = more compute & memory!

GroqChip™ 1



Simplified design enables **Compute Performance**

Competitive Chip Example



Complex design for processing data results in **Compute Costs**

2020

Appears in the *Proceedings of the International Symposium on Computer Architecture* (ISCA-47), June 1, 2020.

Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads

Dennis Abts, Jonathan Ross, Jonathan Sparling, Mark Wong-VanHaren, Max Baker, Tom Hawkins, Andrew Bell, John Thompson, Temesghen Kahsai, Garrin Kimmell, Jennifer Hwang, Rebekah Leslie-Hurd, Michael Bye, E.R. Creswick, Matthew Boyd, Mahitha Venigalla, Evan Laforge, Jon Purdy, Purushotham Kamath, Dinesh Maheshwari, Michael Beidler, Geert Rosseel, Omar Ahmad, Gleb Gagarin, Richard Czekalski, Ashay Rane, Sahil Parmar, Jeff Werner, Jim Sproch, Adrian Macias, Brian Kurtz

Groq, Inc. · Mountain View, California

***Abstract*—In this paper, we introduce the Tensor Streaming Processor (TSP) architecture, a functionally-sliced microarchitecture with memory units interleaved with vector and matrix deep learning functional units in order to take advantage of dataflow locality of deep learning operations. The TSP is built based on two key observations: (1) machine learning workloads exhibit abundant data parallelism, which can be readily mapped to tensors in hardware, and (2) a simple and deterministic processor with producer-consumer stream programming model enables precise reasoning and control of hardware components, achieving good performance and power efficiency. The TSP is designed to exploit parallelism inherent in machine-learning workloads including instruction-level, memory concurrency, data and model parallelism, while guaranteeing determinism by eliminating all reactive elements in the hardware (*e.g.* arbiters, and caches). Early ResNet50 image classification results demonstrate 20.4K processed images per second (IPS) with a batch-size of one—a 4× improvement compared to other modern GPUs and accelerators [44]. Our first ASIC implementation of the TSP architecture yields a computational density of more than 1 TeraOp/s per square mm of silicon for its 25×29 mm 14nm chip operating at a nominal clock frequency of 900 MHz. The TSP demonstrates a novel hardware-software approach to achieve fast, yet predictable, performance on machine-learning workloads within a desired power envelope.**

I. Introduction

The world is increasingly turning to computationally-intensive deep learning algorithms to solve important problems in science, transportation, security, and beyond. These workloads continue to grow both in size and complexity, presenting serious scalability, performance, and usability challenges for traditional CPU and GPU architectures. Keeping pace with this demand requires that we provision abundant on-chip ALUs and utilize them at near-peak performance throughout program execution. Unfortunately, hardware complexity of many microarchitectures makes it difficult to reason about runtime stalls. Furthermore, while microarchitectural enhancements such as caches, branch predictors, and prefetchers help tremendously in improving performance, they do not bound worst-case performance.

Over the last decade, data center operators have installed many-core systems as a stalwart fixture in warehouse-scale computers (WSC) [7], [11], [13]. These increasingly heterogeneous systems have dozens of processing cores that vary widely in form and function, including GPUs, TPUs, FPGAs, and smart IO controllers for efficient remote direct memory access. Such efforts have largely focused on accelerating deep neural network *training* and *inference* performance for a wide range of workloads: convolutional neural networks (CNNs) for recommendation algorithms, computer vision (CV) and image classification; recurrent neural networks (RNNs) for natural language processing [20], [58]; and now attention and transformer models [10], [55]. The increased computational requirements of these models has been the catalyst for a resurgence of architectural innovation [21].

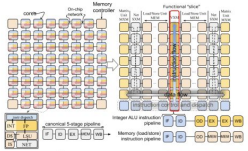


Fig. 1. Conventional 2D mesh of cores (a) reorganized into a functionally sliced arrangement of tiles (b).

The demand for domain-specific architectures, as well as their widespread adoption into cloud computing ecosystems [34], [48], provides a unique opportunity for novel microarchitectures targeting deep learning applications. A range of innovative solutions are emerging—from Cerebras' wafer-scale integration [15] to more traditional chip multiprocessors like GraphCore's IPU [25]. Broadly speaking, all of these efforts are focused on delivering more compute capability per floor tile in the data center. In other words, the focus is on

2022

A Software-defined Tensor Streaming Multiprocessor for Large-scale Machine Learning

Dennis Abts, Groq Inc.
Garrin Kimmell, Groq Inc.
Andrew Ling, Groq Inc.
John Kim, KAIST/Groq Inc.
Matt Boyd, Groq Inc.
Andrew Bitar, Groq Inc.
Sahil Parmar, Groq Inc.
Ibrahim Ahmed, Groq Inc.
Roberto DiCecco, Groq Inc.
David Han, Groq Inc.
John Thompson, Groq Inc.
Michael Bye, Groq Inc.
Jennifer Hwang, Groq Inc.
Jeremy Fowers, Groq Inc.
Peter Lillian, Groq Inc.
Ashwin Murthy, Groq Inc.
Elyas Mehtabuddin, Groq Inc.
Chetan Tekur, Groq Inc.
Thomas Sohmers, Groq Inc.
Kris Kang, Groq Inc.
Stephen Maresh, Groq Inc.
Jonathan Ross, Groq Inc.

ABSTRACT

We describe our novel commercial software-defined approach for large-scale interconnection networks of tensor streaming processing (TSP) elements. The system architecture includes packaging, routing, and flow control of the interconnection network of TSPs. We describe the communication and synchronization primitives of a bandwidth-rich substrate for global communication. This scalable communication fabric provides the backbone for large-scale systems based on a software-defined Dragonfly topology, ultimately yielding a parallel machine learning system with elasticity to support a variety of workloads, both training and inference. We extend the TSP's producer-consumer stream programming model to include global memory which is implemented as logically shared, but physically distributed SRAM on-chip memory. Each TSP contributes 220 MiBytes to the global memory capacity, with the maximum capacity limited only by the network's scale -- the maximum number of endpoints in the system. The TSP acts as both a processing element (endpoint) and network switch for moving tensors across the communication links. We describe a novel software-controlled networking approach that avoids the latency variation introduced by dynamic contention for network links. We describe the topology, routing and flow control to characterize the performance of the network that serves as the fabric for a large-scale parallel machine learning system with up to 10,440 TSPs and more than 2 TeraBytes of global memory accessible in less than 3 microseconds of end-to-end system latency.

ACM ISBN 978-1-4503-8610-4/22/06.
https://doi.org/10.1145/3470496.3527405

CCS CONCEPTS

• **Computer systems organization → Interconnection architectures**.

KEYWORDS

Machine Learning, Tensor Streaming Processor, Dragonfly, Software Scheduling

1 INTRODUCTION

Historically, high-performance computing (HPC) systems were broadly categorized as *capability* or *capacity* systems. This dichotomy arises because of communication latency and bandwidth trade-offs when we apply more processing elements (PEs) to a fixed-size problem (*strong* scaling) with the goal of minimizing the program's execution time. Alternatively, we can deploy more PEs to increase throughput (i.e. *weak* scaling). This duality requires both novel chip architectures [7][1][36] in the underlying PEs *and* a scalable system architecture with high throughput (bisection bandwidth) and low end-to-end latency (low network diameter) for fine-grained communication necessary to efficiently handle both strong and weak scaling. For example, the network demand for training an ML model, often requiring data parallelism (weak scaling), differs from inference on that same model using (pipelined) model parallelism (ie. strong scaling). The multiprocessor system, interconnection network, and the programming model of the individual processing elements work in unison to collectively execute the different "layers" of a deep learning network. It is this set of sub-tasks, expressed as individual PE programs, that are distributed among the computing elements and responsible for carrying out, or executing, the specifics of the machine learning model.

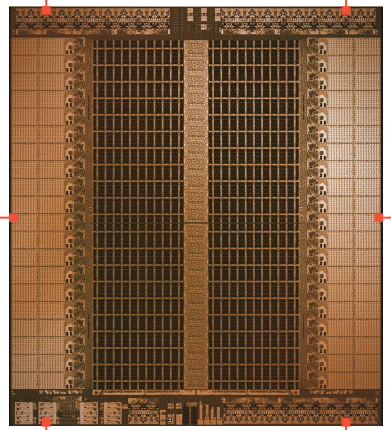
The burgeoning parameter space of natural language processing (NLP) models like GPT-3 [6] use 100s-of-billions of parameters to

GroqChip Scalable Architecture

SRAM Memory
Massive concurrency
80 TB/s of BW
Stride insensitive

Groq TruePoint™ Matrix
4x Engines
320x320 fused dot product
Integer and floating point

Programmable Vector Units
5,120 Vector ALUs* for high performance



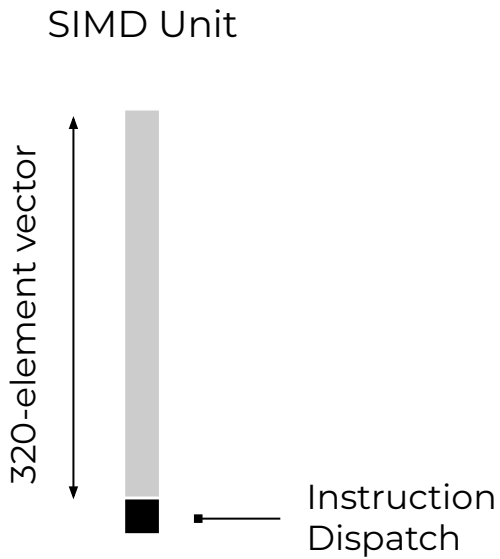
GroqChip 1

Networking
480 GB/s bandwidth
Extensible network scalability
Multiple topologies

Data Switch
Shift, Transpose, Permuter for improved data movement and data reshapes

Instruction Control
Multiple instruction queues for instruction parallelism

GroqChip Building Blocks



GroqChip Building Blocks

Build different types of specialized SIMD units

MXM
Matrix-Vector /
Matrix-Matrix Multiply

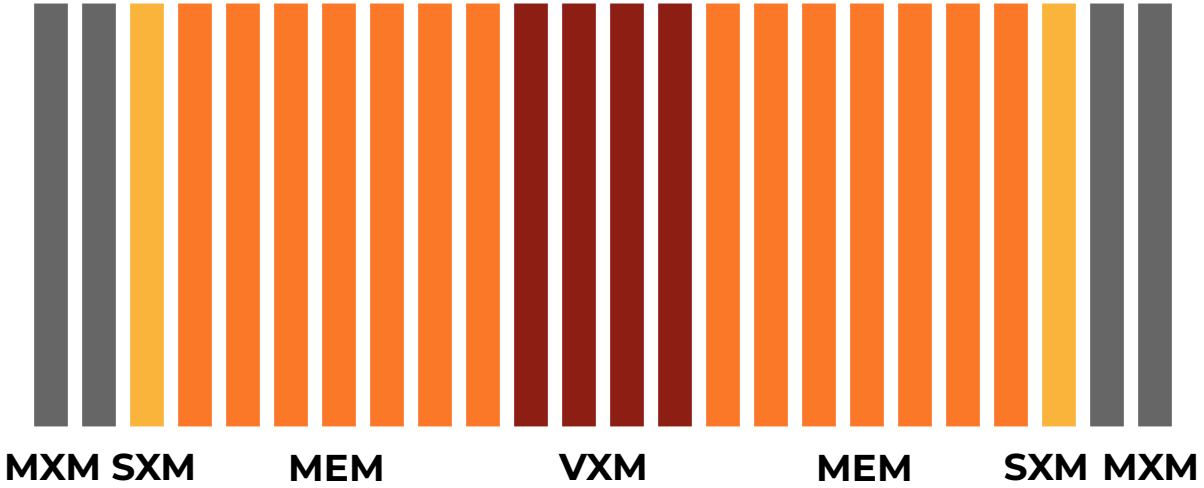
VXM
Vector-Vector
Operations

SXM
Data Reshapes

MEM
On-chip SRAM

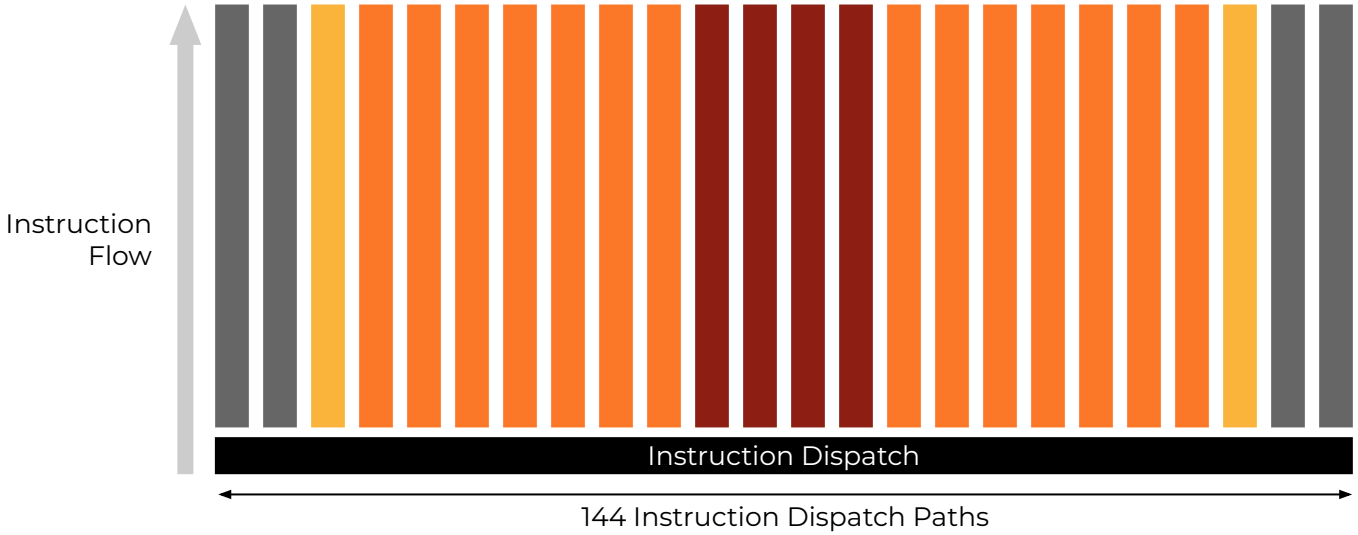
GroqChip Building Blocks

Lay out SIMD units across chip area



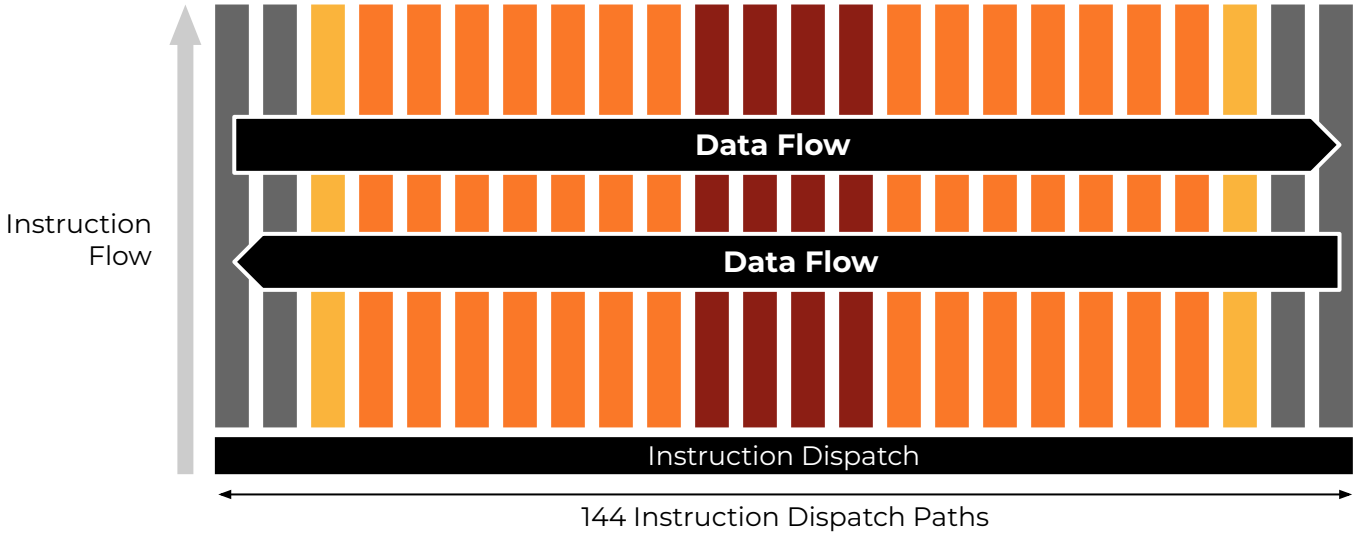
GroqChip Building Blocks

Synchronized instruction dispatch across all SIMD units for lockstep execution



GroqChip Building Blocks

High-bandwidth “Stream Registers” for passing data between units



BUILDING A

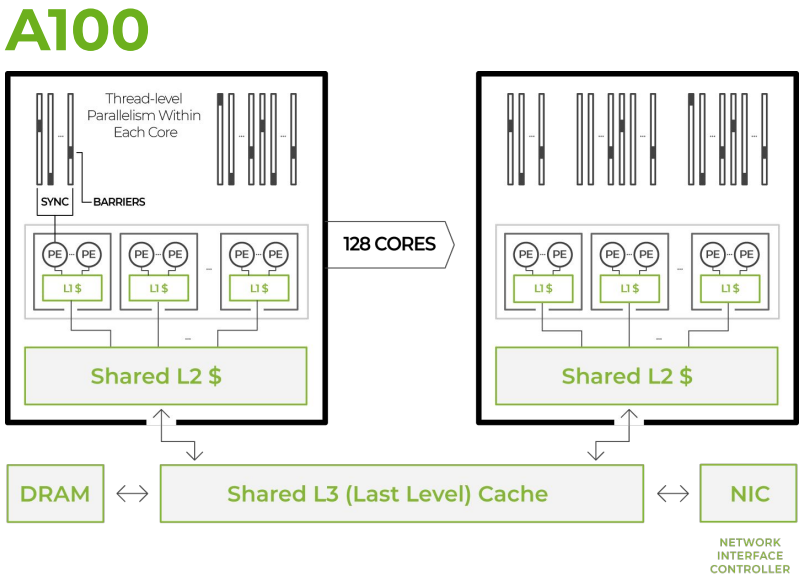
Compiler For GroqChip

Building Compilers For Spatial Architectures

Challenges when building an automated compiler for spatial architectures

Unpredictable memory latency

- Non-deterministic memory (caching, DRAM) reduces compiler ability to hide memory latency



Building Compilers For Spatial Architectures

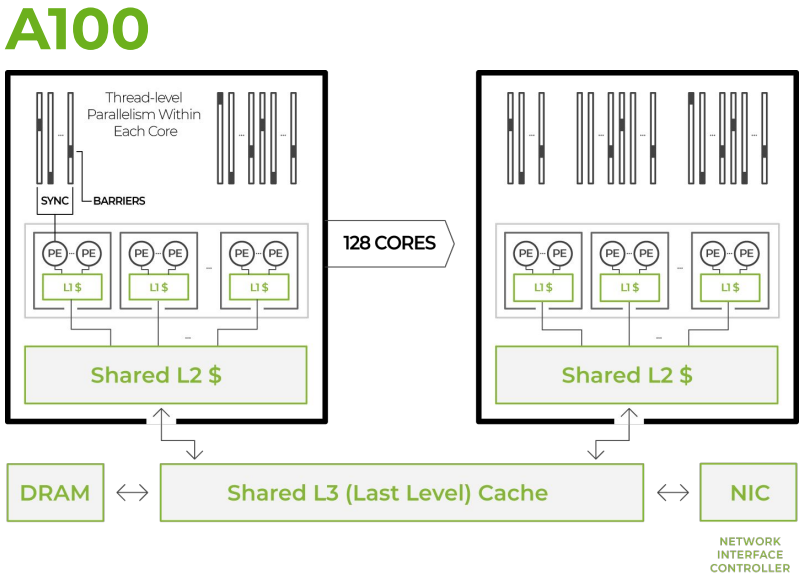
Challenges when building an automated compiler for spatial architectures

Unpredictable memory latency

- Non-deterministic memory (caching, DRAM) reduces compiler ability to hide memory latency

Thread synchronization

- Unsynchronized thread execution makes compile time resource allocation a challenge



Building Compilers For Spatial Architectures

Challenges when building an automated compiler for spatial architectures

Unpredictable memory latency

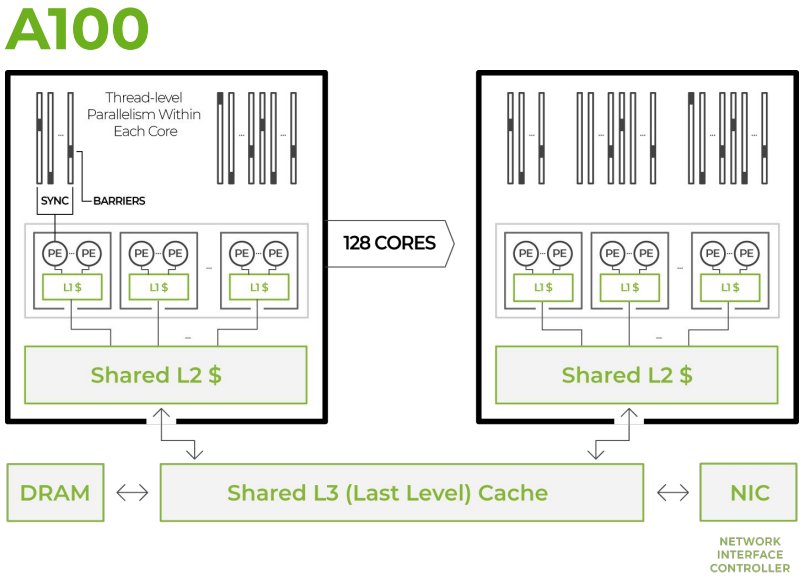
- Non-deterministic memory (caching, DRAM) reduces compiler ability to hide memory latency

Thread synchronization

- Unsynchronized thread execution makes compile time resource allocation a challenge

Complex interconnect

- Dynamic hardware (e.g. arbiters, queues, virtual channels) makes it difficult to resolve data movement at compile time
  - E.g. See Networks-on-Chip (NoC) in CMPs

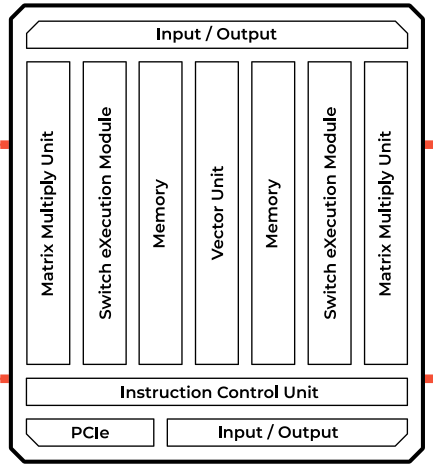


Building a Compiler For GroqChip

Groq Arch Abstraction

Software-controlled Memory → No Hardware Caching

Specialized, Location-specific Functional Units



Simple Interconnect

Lockstep Execution of Functional Unit (“threads”)

An ISA That Empowers Software

Software-controlled memory enabled by low-level abstraction exposed by architecture

No dynamic hardware caching

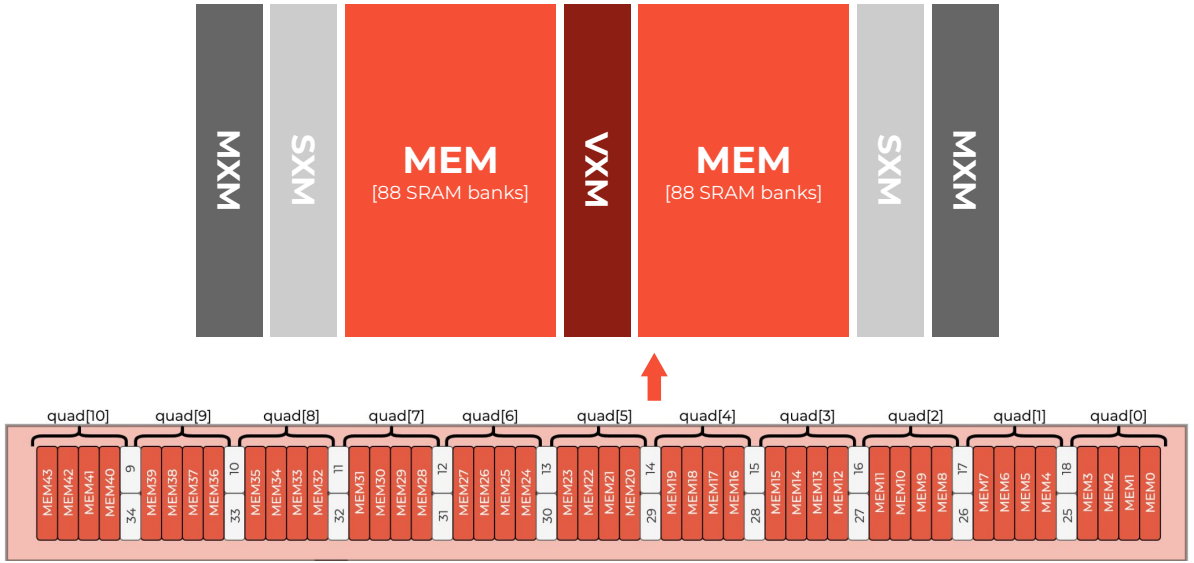
- Compiler aware of all data locations at any given point in time

Flat memory hierarchy (no L1, L2, L3, etc)

- Memory exposed to software as a set of physical banks that are directly addressed

Large on-chip memory capacity (220 MB) at high-bandwidth with (80TBps) reduces need to spill to non-deterministic DRAM

- Provides enough “scratchpad” memory to hide external memory accesses behind compute



An ISA That Empowers Software

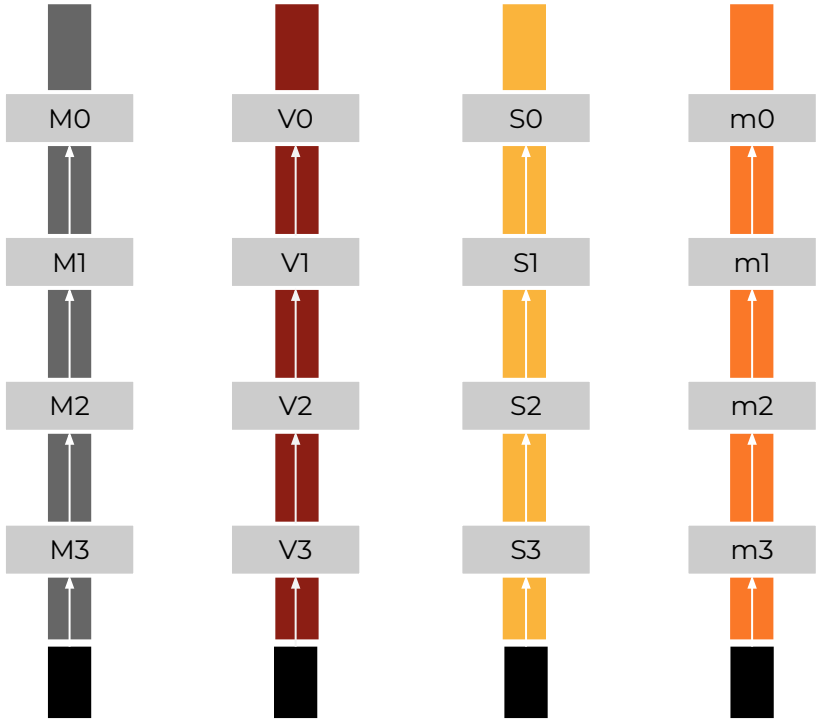
Compiler empowered to perform cycle-accurate instruction scheduling

Functional units execute in lockstep

- One instruction issued per cycle at each dispatch path
  - Can be viewed as fully-pipelined 144-wide VLIW instructions

Little hardware control needed for managing instruction execution

- < 3% area overhead for instruction dispatch logic



An ISA That Empowers Software

Explicit time and space of instruction execution exposed by ISA to the compiler

Each Functional Unit (FU) type provides its own low-level instruction set

Number of FUs of each type and relative positions on chip are exposed to software

Compiler can choose to leverage multiple FUs for more concurrency or more pipelining

Instruction Set

Low level

320-vector ops

Explicit resource selection

Explicit scheduling

Function	Instruction

ICU	`NOP N`
	`Ifetch`
	`Sync`
	`Notify`
	`Config`
	`Repeat n,d`

MEM	`Read a,s`
	`Write a,s`
	`Gather s, map`
	`Scatter s, map`
	`Countdown d`
	`Step a`
	`Iterations n`

VXM	unary operation
	binary operation
	type conversions
	`ReLU`
	`TanH`
	`Exp`
	`RSqrt`

MXM	`LW`
	`IW`
	`ABC`
	`ACC`

SXM	`Shift up/down N`
	`Permute map`
	`Distribute map`
	`Rotate stream`
	`Transpose sg16`

C2C	`Deskew`
	`Send`
	`Receive`

An ISA That Empowers Software

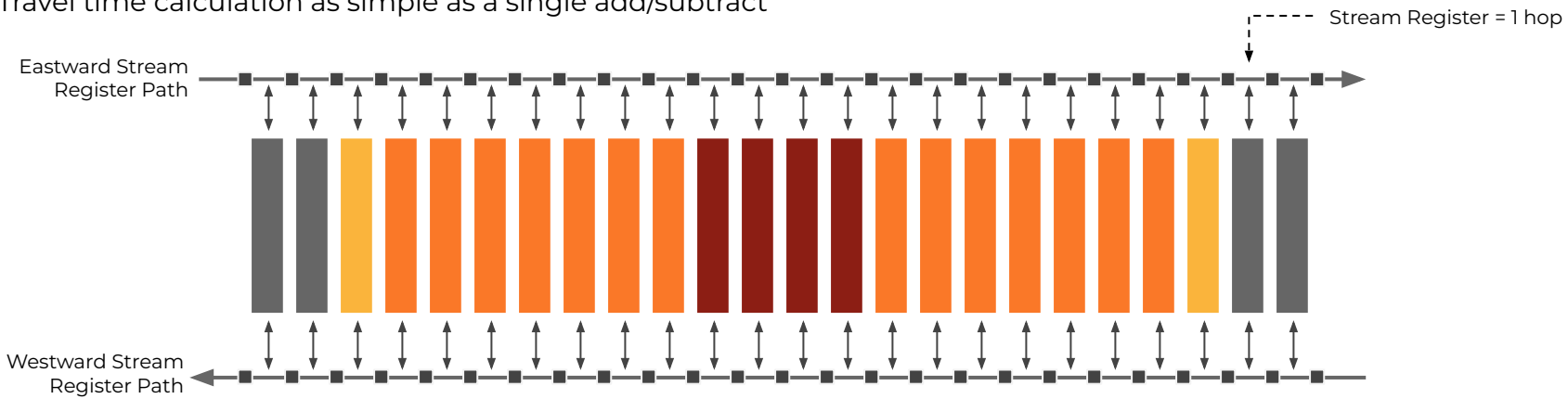
Compiler can quickly reason about all data movement between FUs

Simple, one-dimensional interconnect for inter-FU communication

- Eastward and westward paths made up of arrays of “stream registers”
- Stream register = one-cycle hop

No arbiters / queues = software can easily reason about exact data movement without simulation

Travel time calculation as simple as a single add/subtract



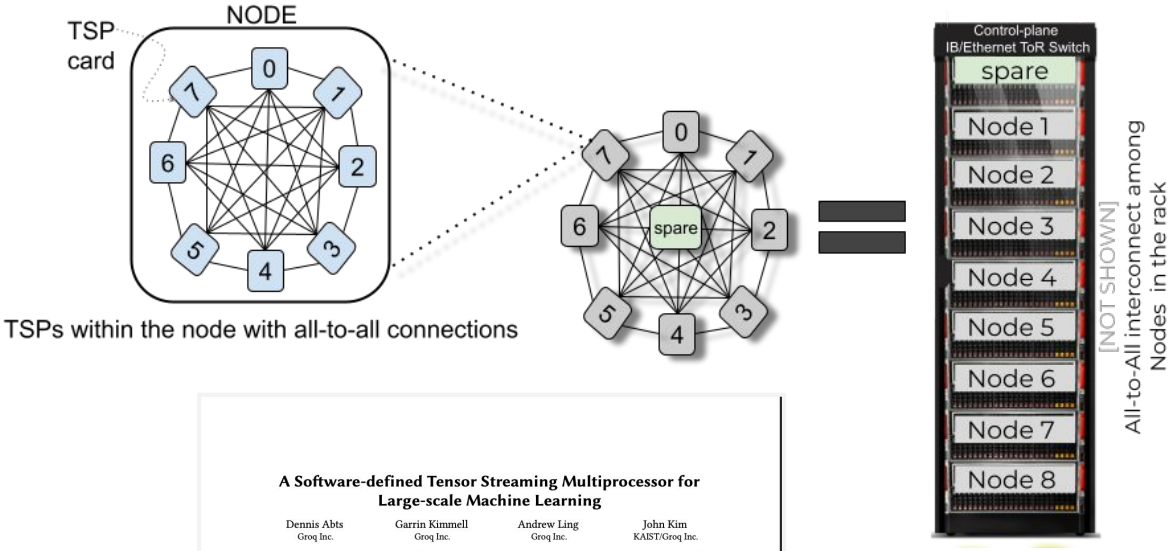
An ISA That Empowers Software

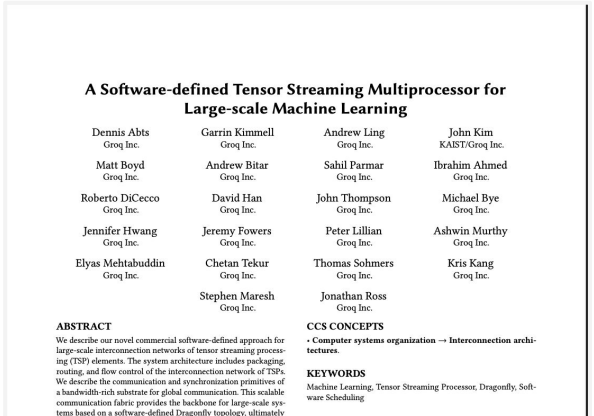
Lockstep execution extended to multiple GroqChip processors

C2C interface ensures synchronization across GroqChips

Low-level abstraction allows compiler to view multiple GroqChip processors as a single device

- Can be scheduled as “one” “Software-scheduled network”





A Software-defined Tensor Streaming Multiprocessor for Large-scale Machine Learning

Dennis Abts, Groq Inc.
Garrin Kimmell, Groq Inc.
Andrew Ling, Groq Inc.
John Kim, KAIST/Groq Inc.
Matt Boyd, Groq Inc.
Andrew Bitar, Groq Inc.
Sahil Parmar, Groq Inc.
Ibrahim Ahmed, Groq Inc.
Roberto DiCecco, Groq Inc.
David Han, Groq Inc.
John Thompson, Groq Inc.
Michael Bye, Groq Inc.
Jennifer Hwang, Groq Inc.
Jeremy Fowers, Groq Inc.
Peter Lillian, Groq Inc.
Ashwin Murthy, Groq Inc.
Elyas Mehtabuddin, Groq Inc.
Chetan Tekur, Groq Inc.
Thomas Sohmers, Groq Inc.
Kris Kang, Groq Inc.
Stephen Maresh, Groq Inc.
Jonathan Ross, Groq Inc.

ABSTRACT

We describe our novel commercial software-defined approach for large-scale interconnection networks of tensor streaming processing (TSP) elements. The system architecture includes packaging, routing, and flow control of the interconnection network of TSPs. We describe the communication and synchronization primitives of a bandwidth-rich substrate for global communication. This scalable communication fabric provides the backbone for large-scale systems based on a software-defined Dragonfly topology, ultimately

CCS CONCEPTS

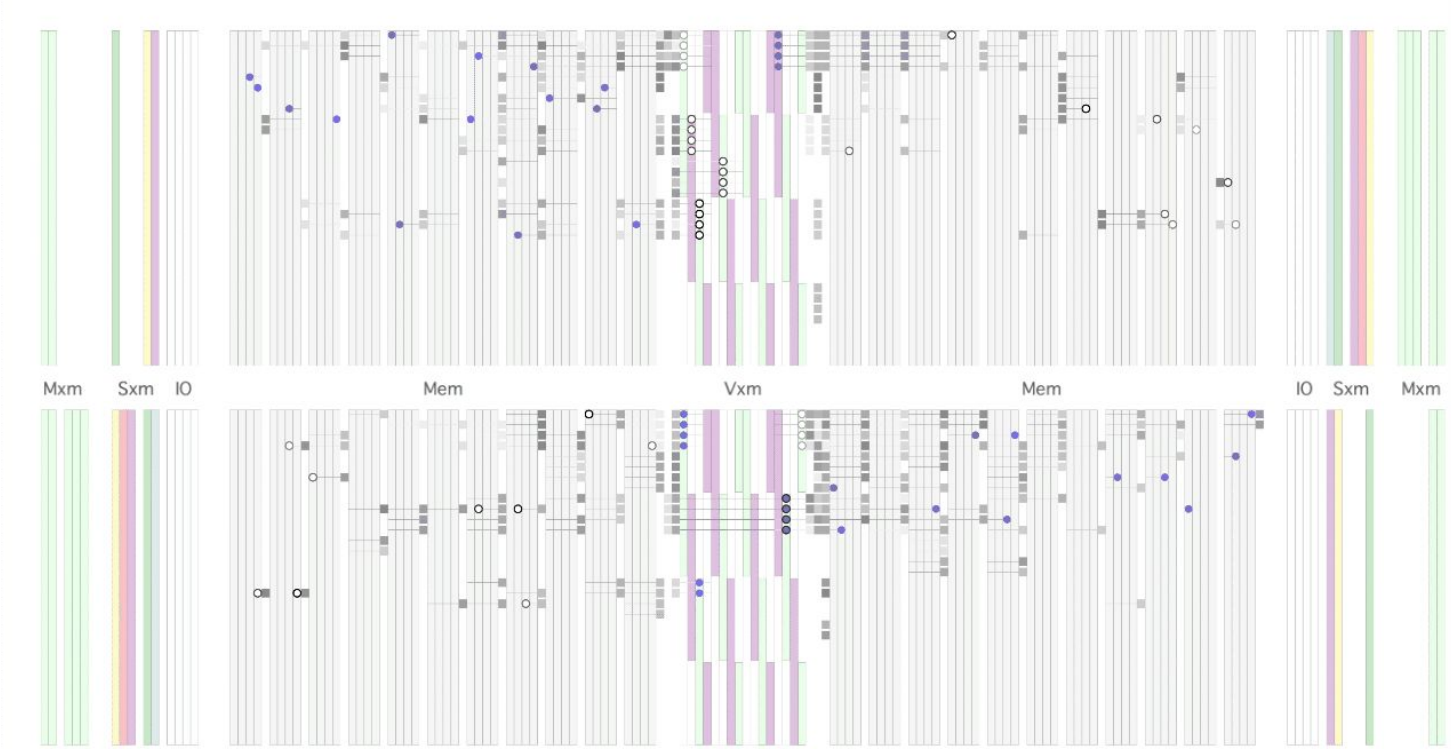
• **Computer systems organization** → **Interconnection architectures.**

KEYWORDS

Machine Learning, Tensor Streaming Processor, Dragonfly, Software Scheduling

Power of Data Orchestration

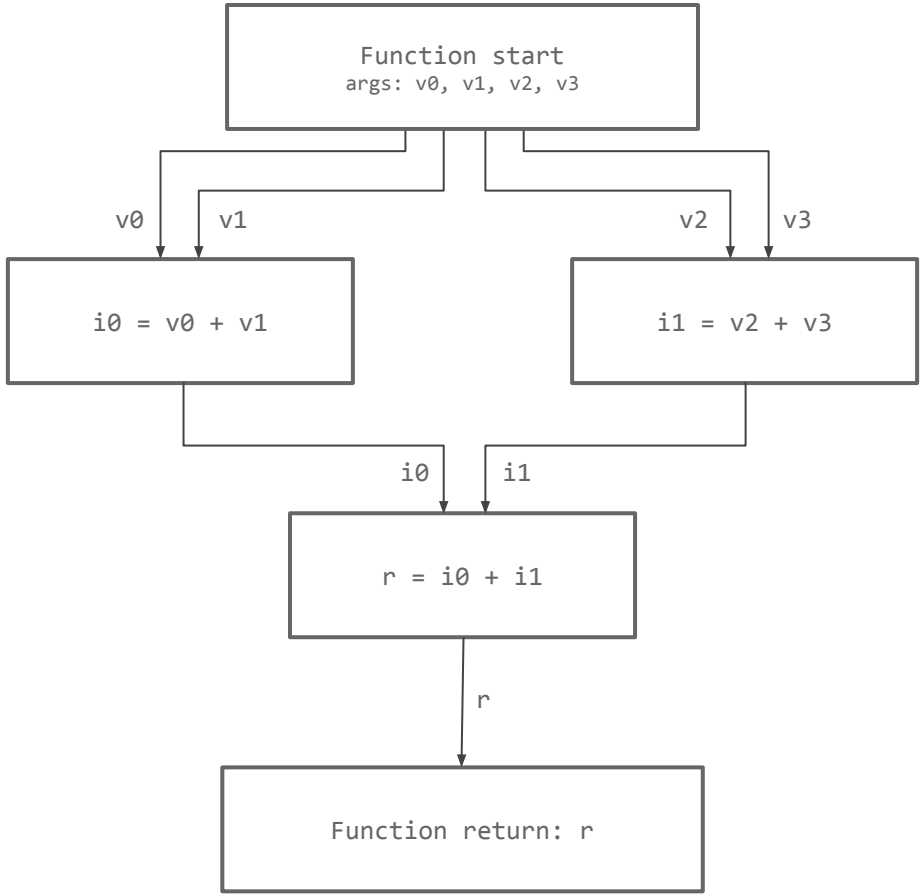
Given to Groq Compiler



EXAMPLE

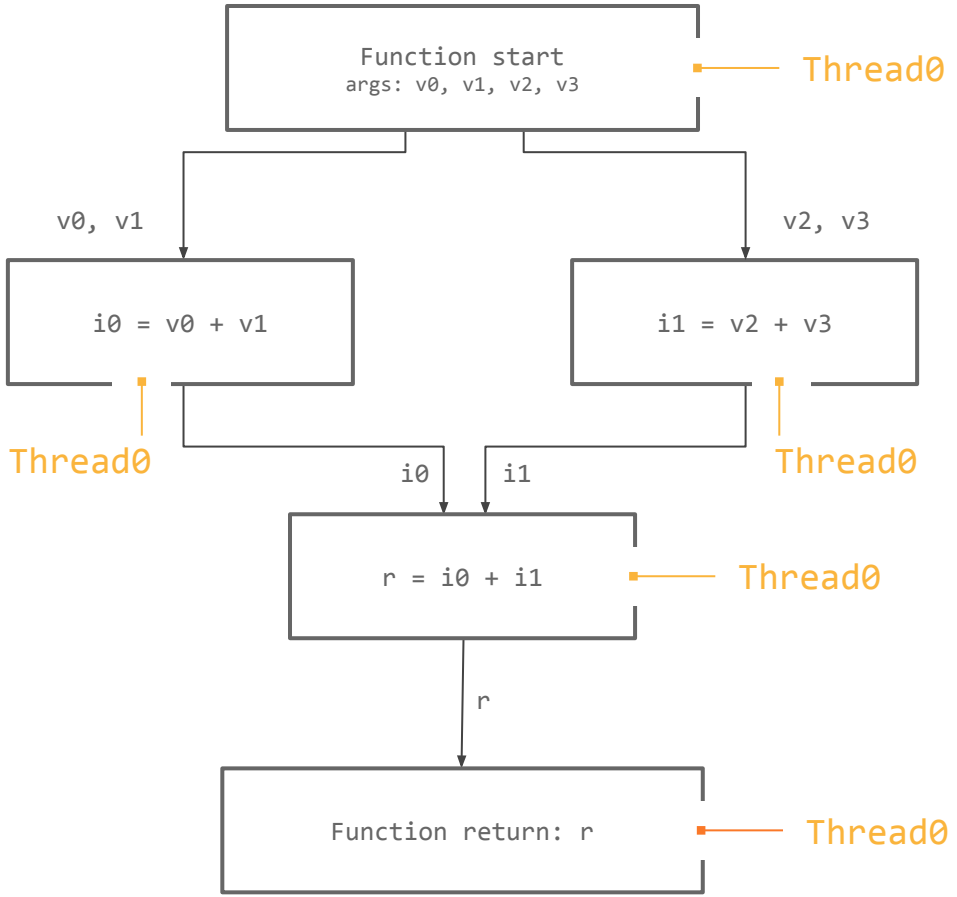
Vectors

Add four vectors



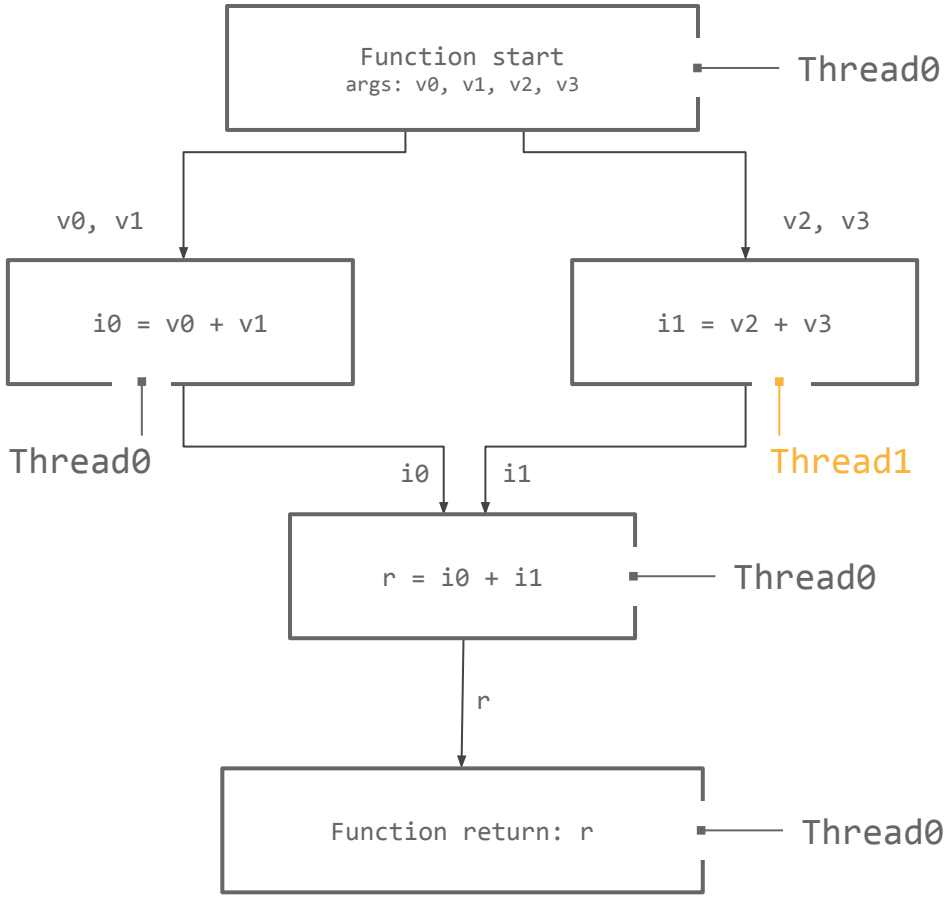
EXAMPLE

Singlethreaded



EXAMPLE

Multithreaded



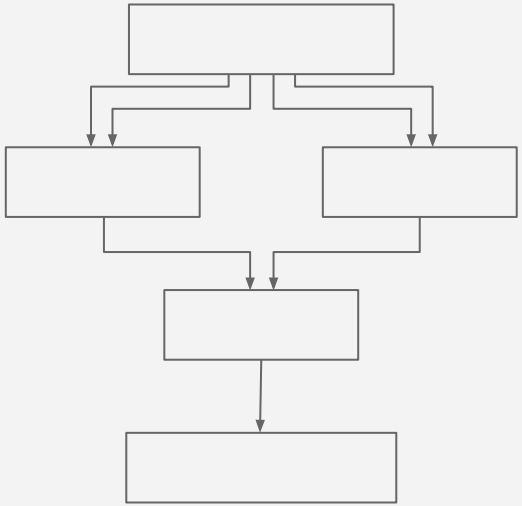
EXAMPLE

Multithreaded

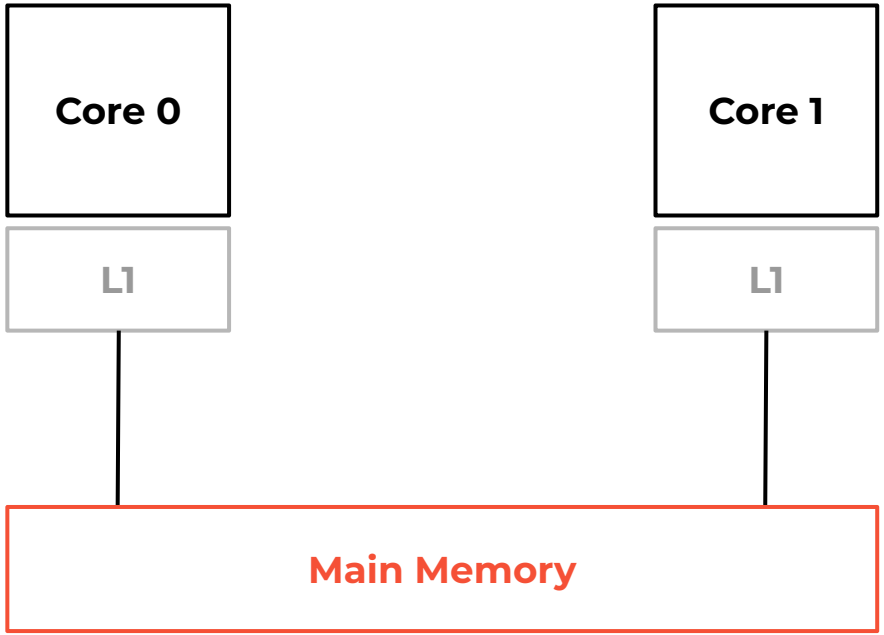
Compiled program contents

1. `function addFour`
   a. `dispatch Thread1(addTwo)`
   b. `execute addTwo`
   c. `join Thread1`
   d. `execute addTwo`
   e. `return`
2. `function addTwo`
   a. `load operand vectors`
   b. `add operand vectors`
   c. `store result`
   d. `return`

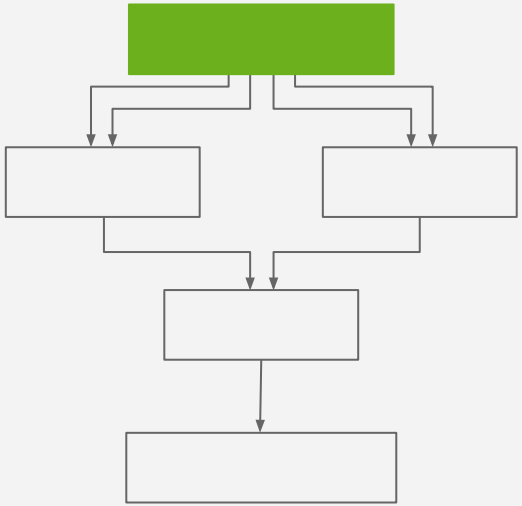
On Multicore SIMD Arch



THEORETICAL MULTITHREADED
SIMD PROCESSOR WITH CACHE

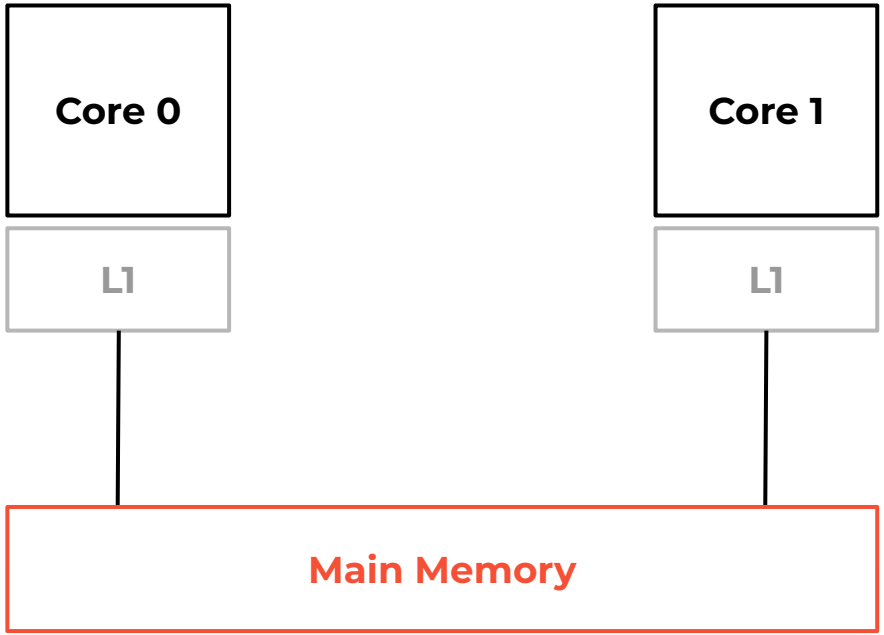


On Multicore SIMD Arch

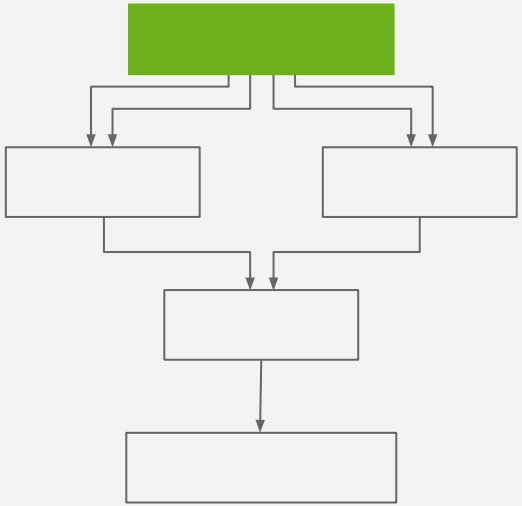


Time = 0

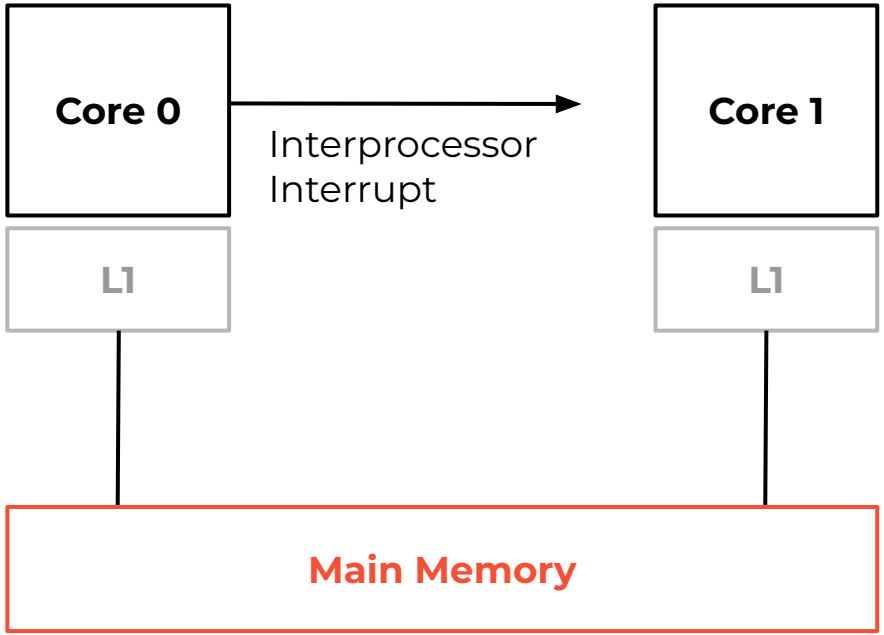
Executing:
addFour



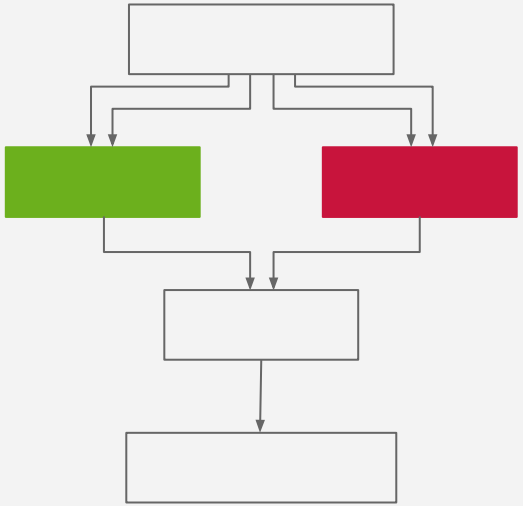
On Multicore SIMD Arch



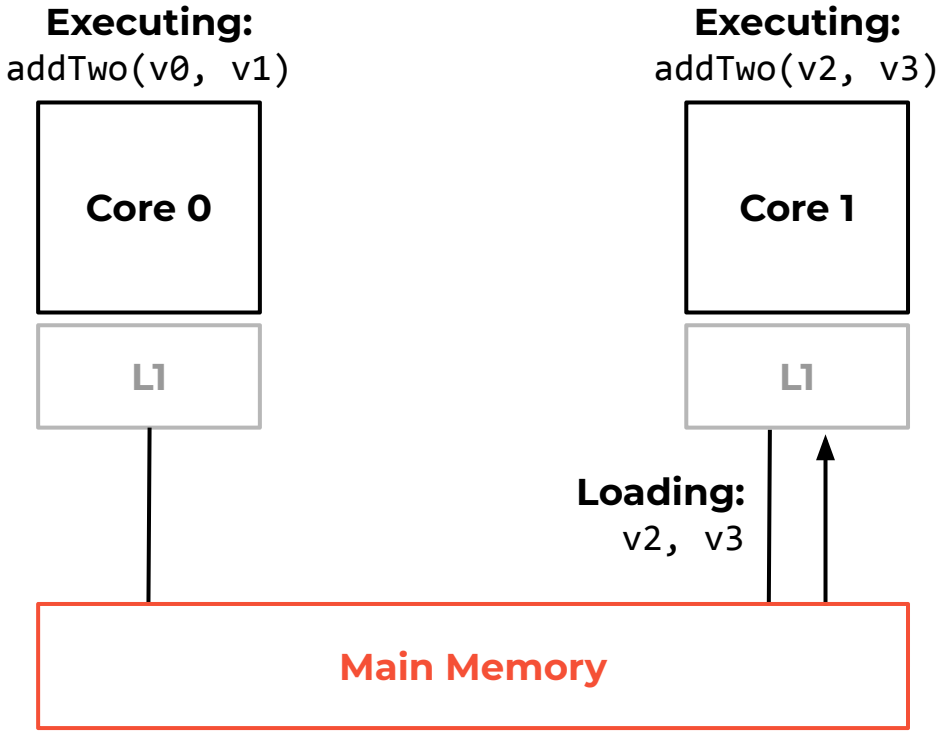
Executing:
`addFour`



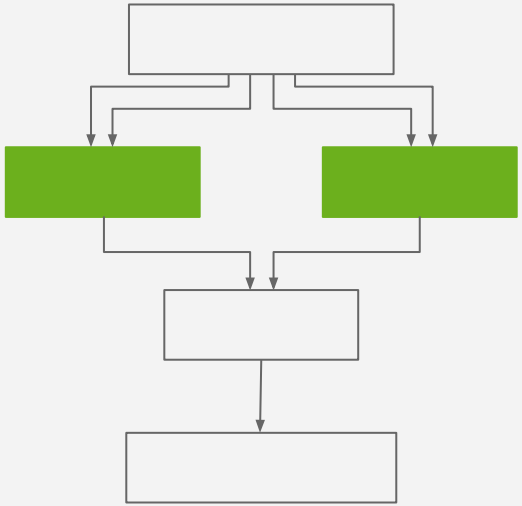
On Multicore SIMD Arch

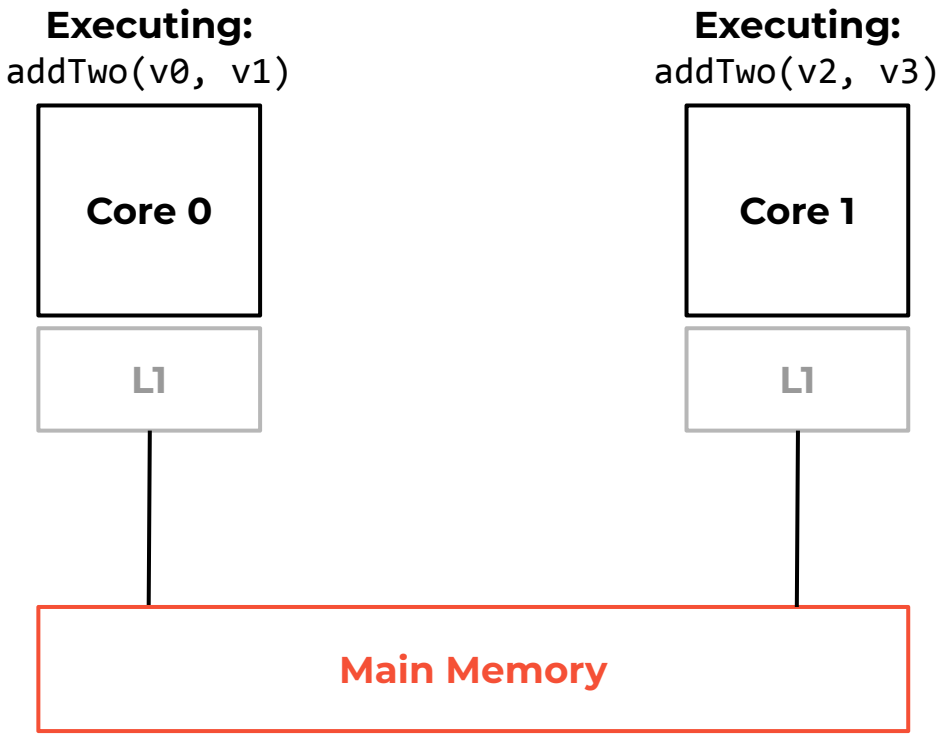


$$\text{Time} = t_{\text{dispatch}}$$

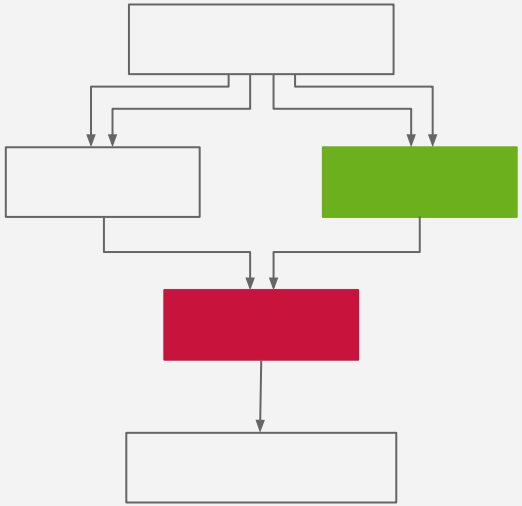


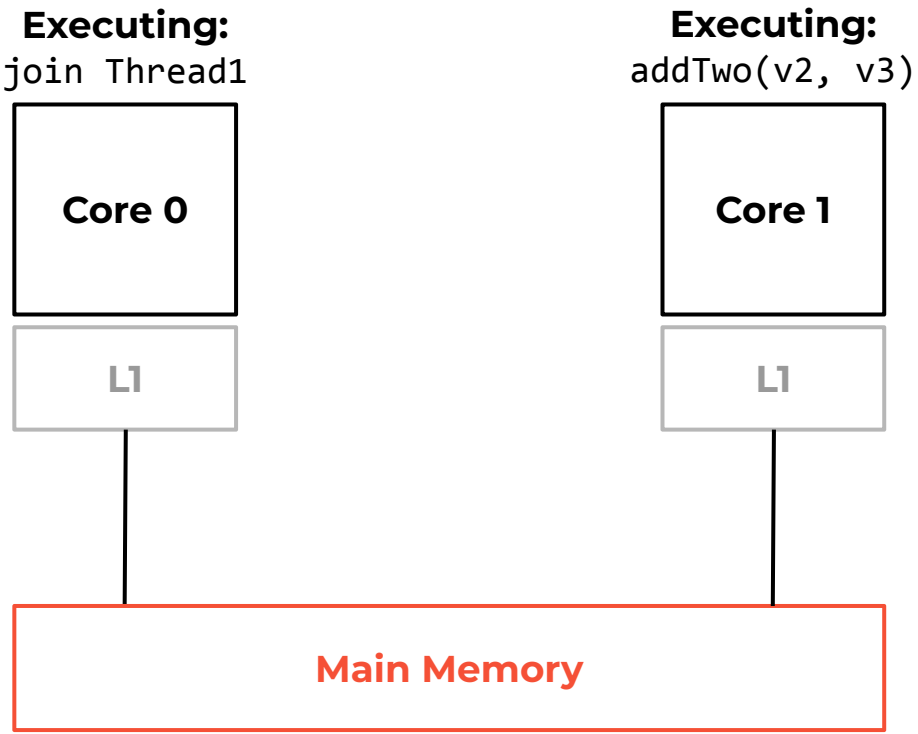
On Multicore SIMD Arch



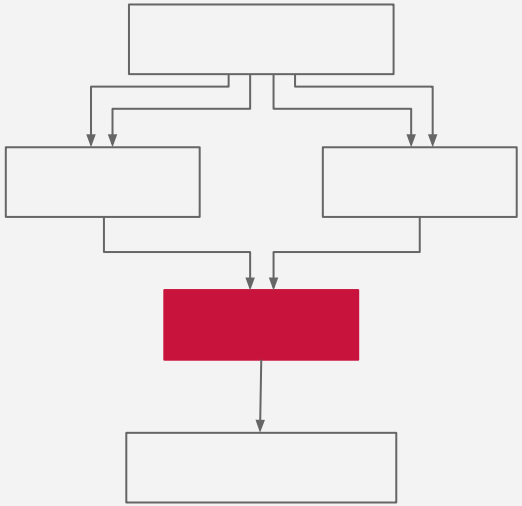


On Multicore SIMD Arch





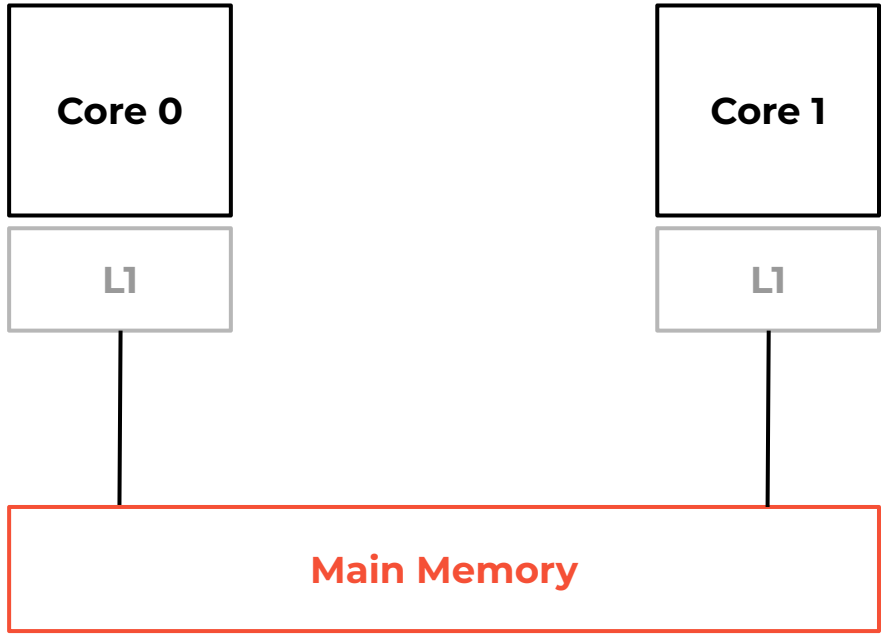
On Multicore SIMD Arch



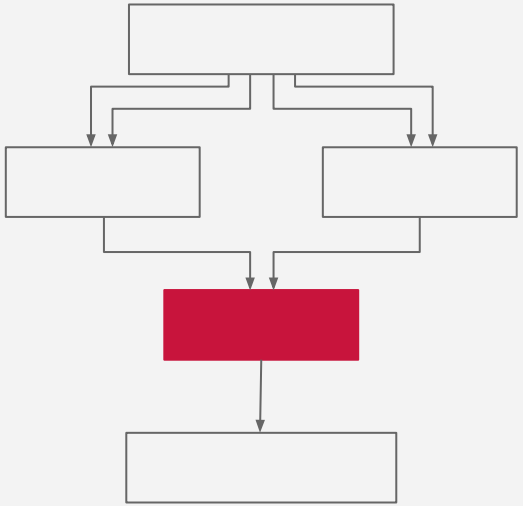
$$\text{Time} = t_{\text{dispatch}} + t_{\text{add}}$$

Executing:

```
join Thread1
```

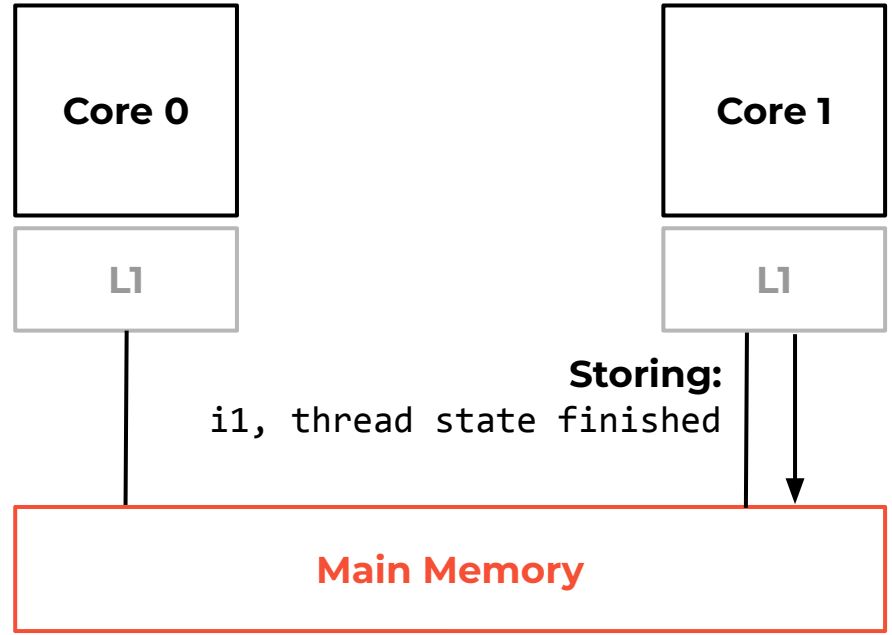


On Multicore SIMD Arch

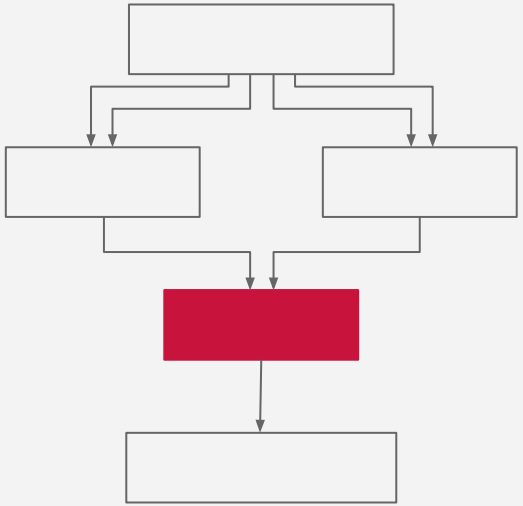


$$\text{Time} = t_{\text{dispatch}} + t_{\text{add}} + t_{\text{store0}} + t_{\text{store1}}$$

Executing:
`join Thread1`

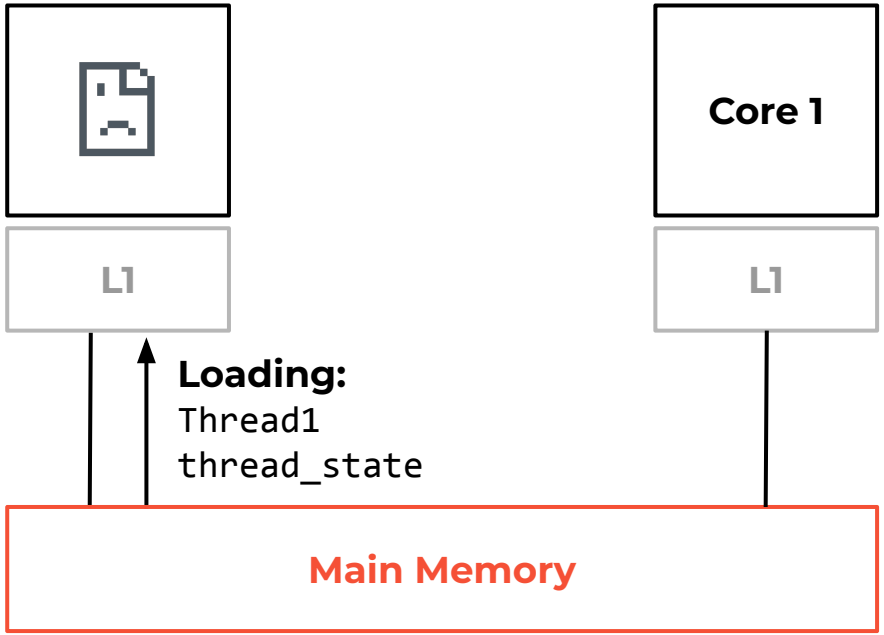


On Multicore SIMD Arch

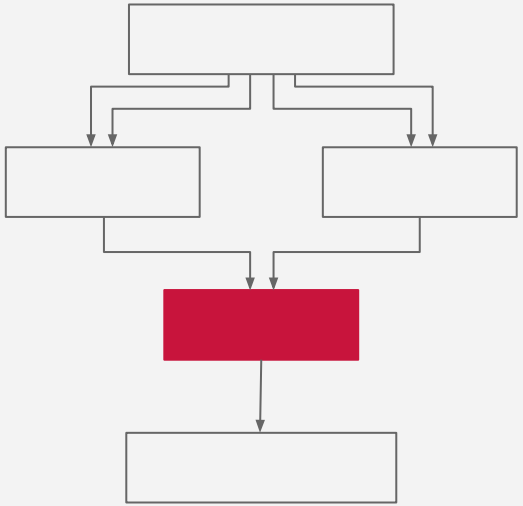


$$\text{Time} = t_{\text{dispatch}} + t_{\text{add}} + t_{\text{store0}} + t_{\text{store1}}$$

Executing:
`cache miss` on
`Thread1 thread_state`

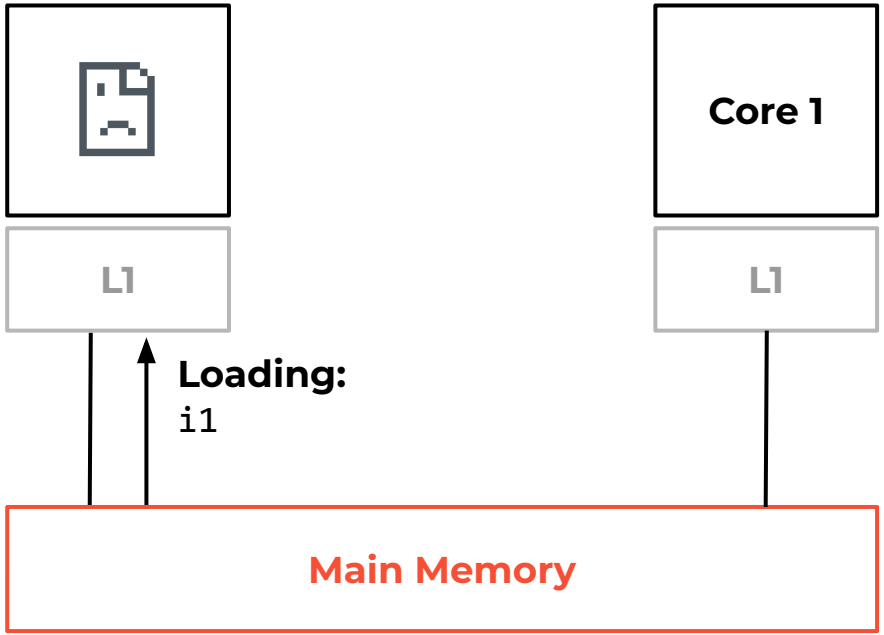


On Multicore SIMD Arch

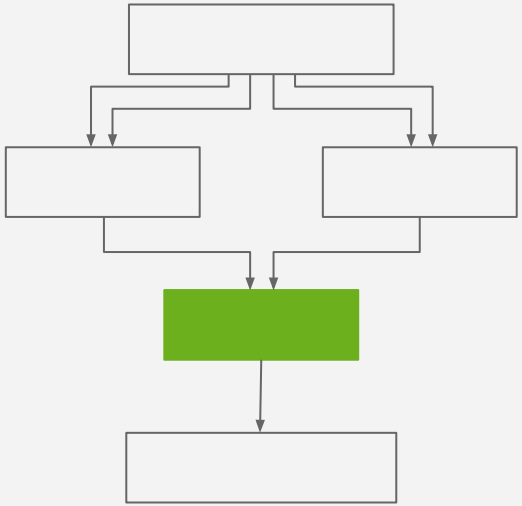


$$\text{Time} = t_{\text{dispatch}} + t_{\text{add}} + t_{\text{store0}} + t_{\text{store1}} + t_{\text{load0}} + t_{\text{load1}}$$

Executing:
`cache miss on i1`



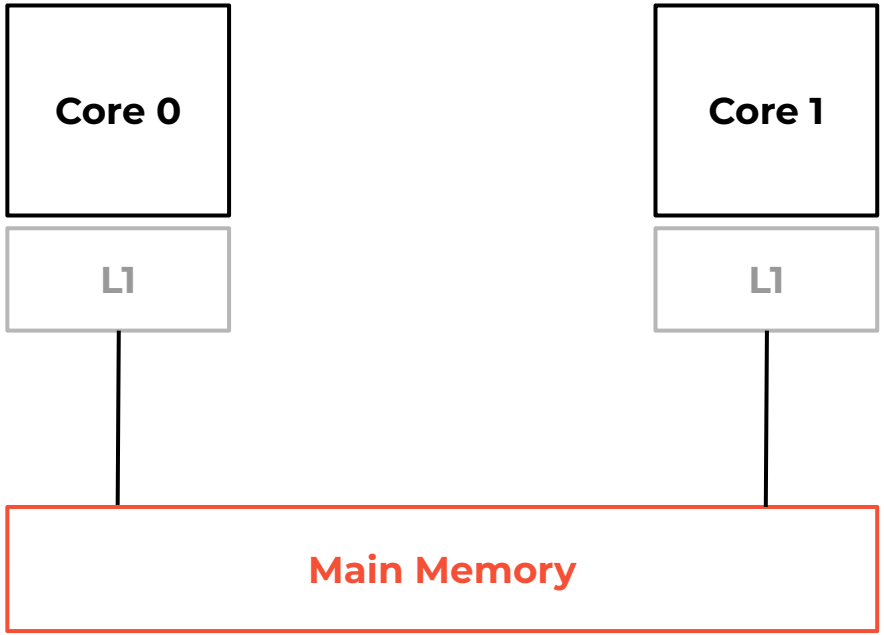
On Multicore SIMD Arch



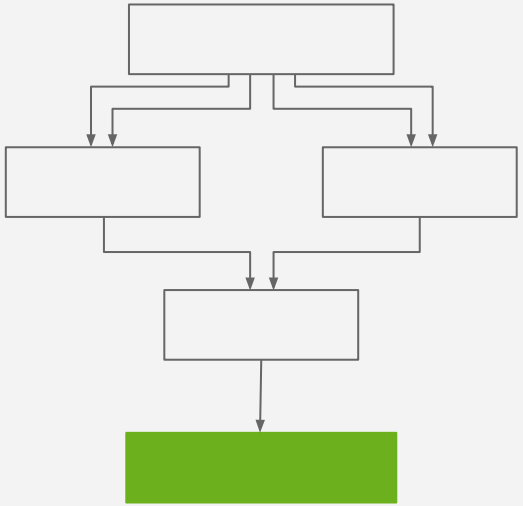
$$\text{Time} = t_{\text{dispatch}} + t_{\text{add}} + t_{\text{store0}} + t_{\text{store1}} + t_{\text{load0}} + t_{\text{load1}}$$

Executing:

`addTwo(i0, i1)`



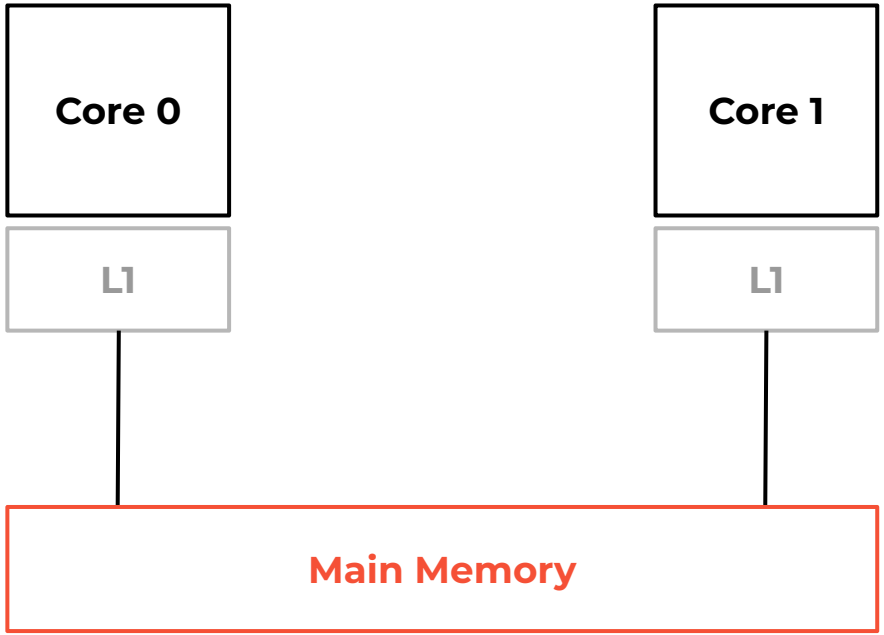
On Multicore SIMD Arch



$$\text{Time} = t_{\text{dispatch}} + 2 * t_{\text{add}} + t_{\text{store0}} + t_{\text{store1}} + t_{\text{load0}} + t_{\text{load1}}$$

Executing:

`return`



On Multicore SIMD Arch

Code for synchronization must be present due to unresolvable times

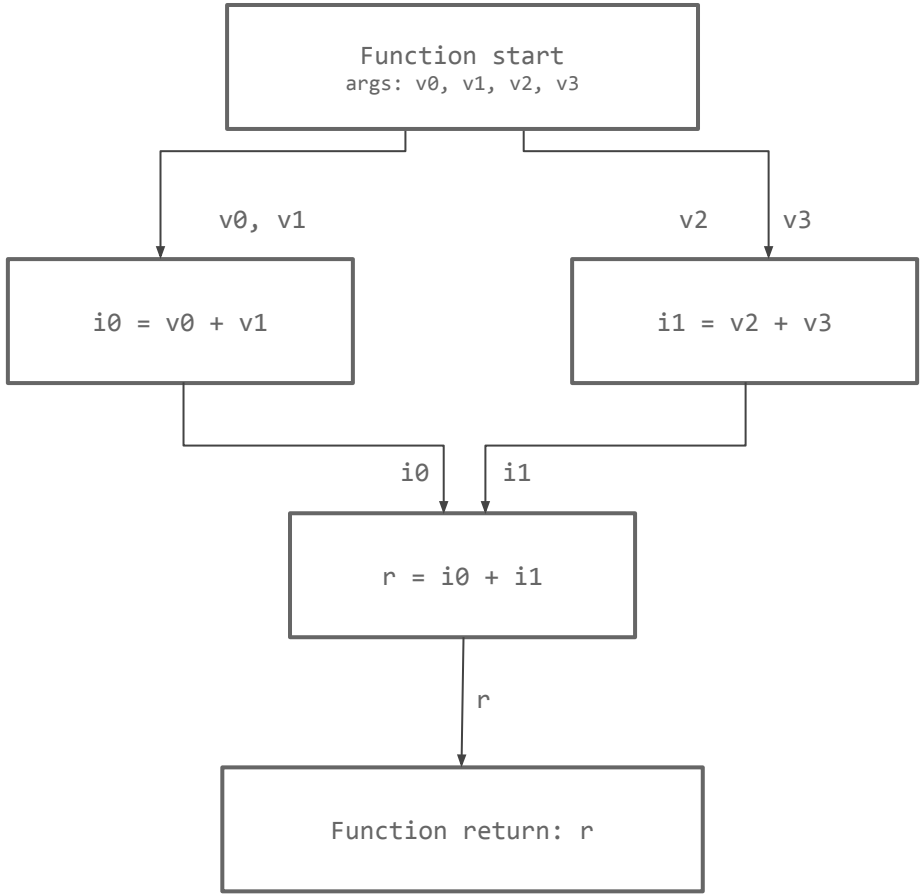
$$\text{Latency} = t_{\text{dispatch}} + 2 * t_{\text{add}} + t_{\text{store0}} + t_{\text{store1}} + t_{\text{load0}} + t_{\text{load1}}$$

t_{dispatch} unknown, implemented via interrupt

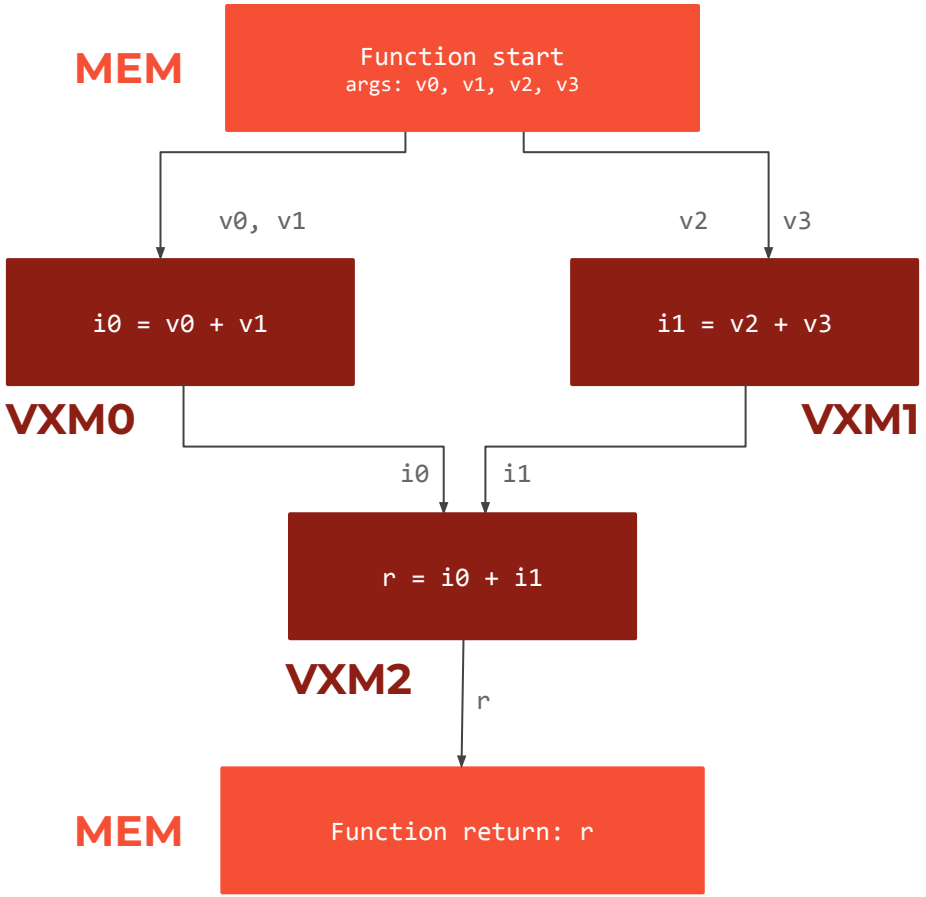
t_{store0}, t_{store1}, t_{load0}, t_{load1} unknown, will vary based on memory performance of system

Red = unresolvable at compile time
Green = resolvable at compile time

On GroqChip


Function start
args: v0, v1, v2, v3
v0, v1
v2
v3
i0 = v0 + v1
i1 = v2 + v3
i0
i1
r = i0 + i1
r
Function return: r

On GroqChip


MEM
Function start
args: v0, v1, v2, v3
v0, v1
v2
v3
i0 = v0 + v1
i1 = v2 + v3
VXM0
VXM1
i0
i1
r = i0 + i1
VXM2
r
MEM
Function return: r

On GroqChip

Compiled program contents

1. `MEM read v0, v1, v2, v3 → streams 0, 1, 2, 3 west`
2. `VXM0 add streams 0, 1 → stream 0 west`
3. `VXM1 add streams 2, 3 → stream 1 west`
4. `VXM2 add streams 0, 1 → stream 0 east`
5. `MEM store stream 0`

On GroqChip

Compiled program contents: Instructions

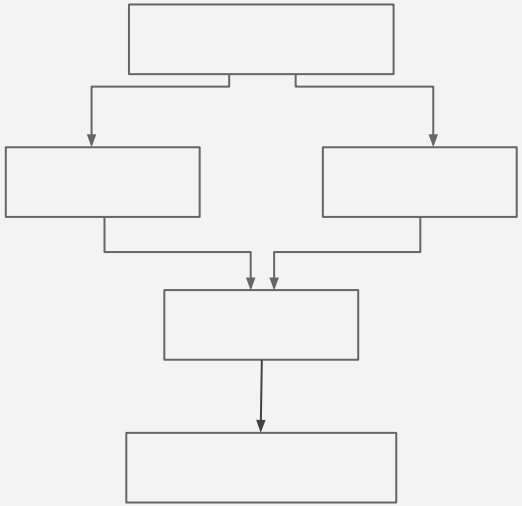
```
MEM0    0 read v0 → streams 0 west store stream 0 t_read + 2 * t_travel + 2 * t_add read v1 → streams 1
        west

MEM1    0 read v1 → streams 1 west

MEM2    0 read v1 → streams 2 west

MEM3    0 read v1 → streams 3 west

VXM0    add streams 0, 1 -> stream 0 west t_read + t_travel

VXM1    add streams 2, 3 -> stream 1 west t_read + t_travel

VXM2    add streams 0, 1 -> stream 0 east t_read + t_travel + t_add
```

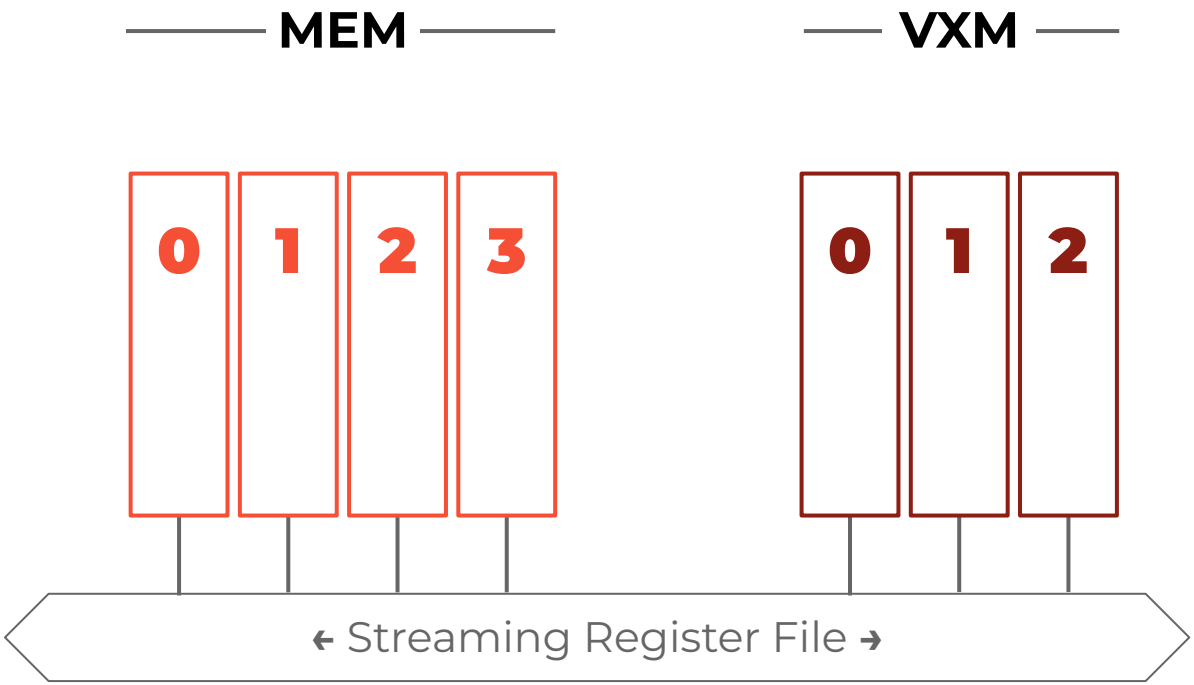

On GroqChip

Compiled program contents: Instructions

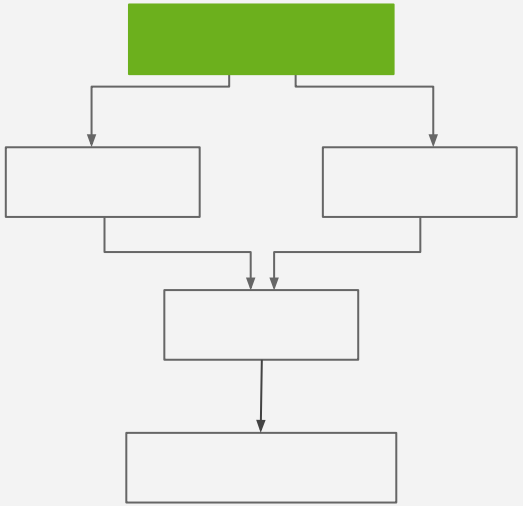
Block	Time	Instruction
MEM0	0 $t_{read} + 2 * t_{travel} + 2 * t_{add}$	read v0 → streams 0 west store stream 0
MEM1	0	read v1 → streams 1 west
MEM2	0	read v2 → streams 2 west
MEM3	0	read v3 → streams 3 west
VXM0	$t_{read} + t_{travel}$	add streams 0, 1 → stream 0 west
VXM1	$t_{read} + t_{travel}$	add streams 2, 3 → stream 1 west
VXM2	$t_{read} + t_{travel} + t_{add}$	add streams 0, 1 → stream 0 east

On GroqChip

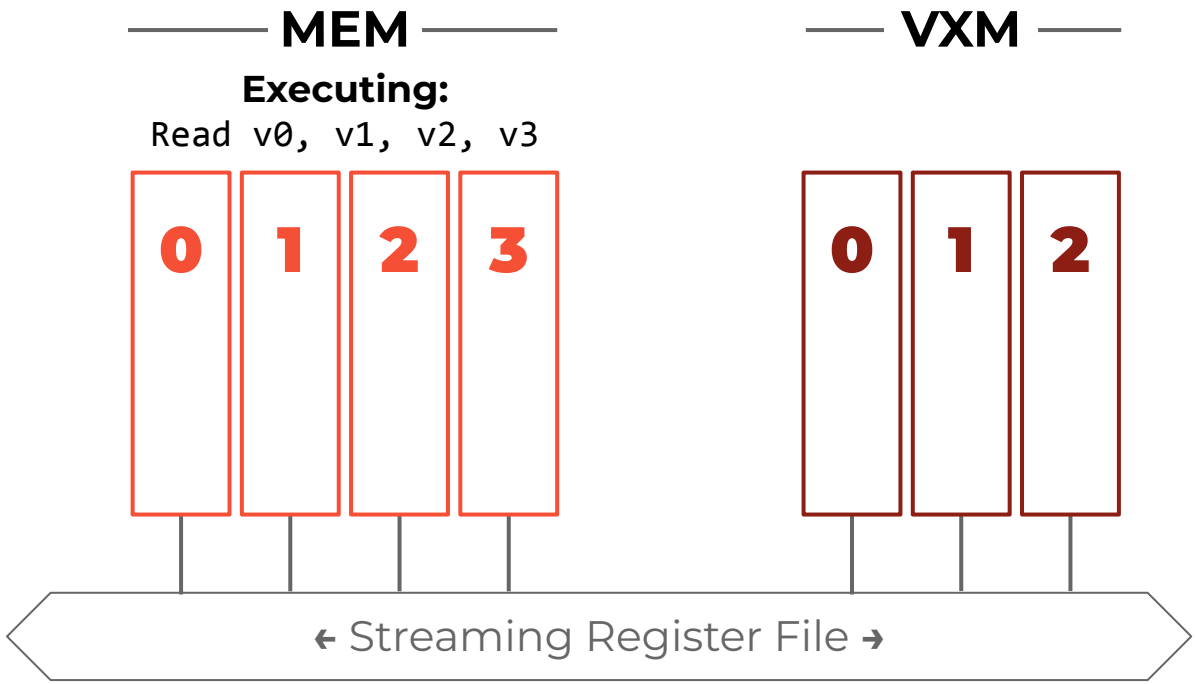




On GroqChip

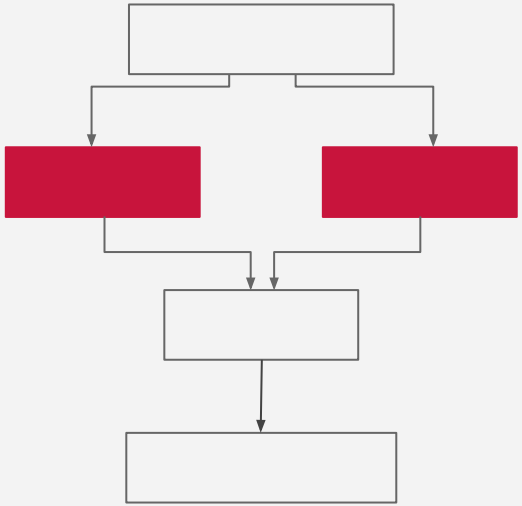


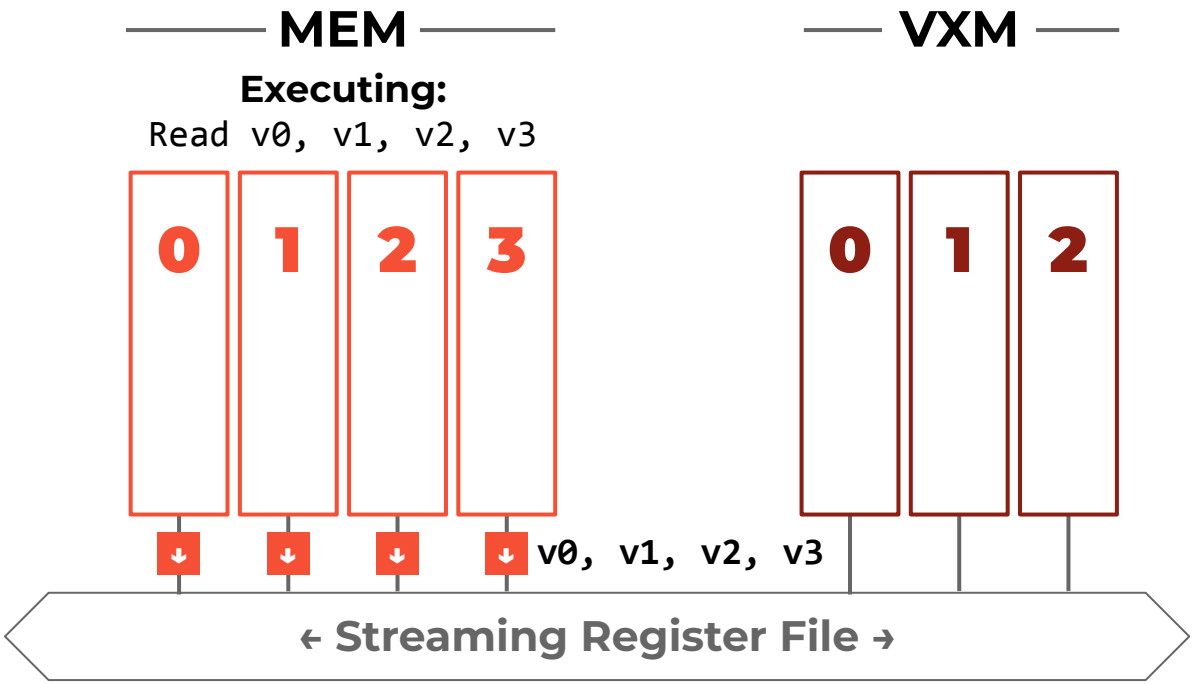
Time = 0



On GroqChip

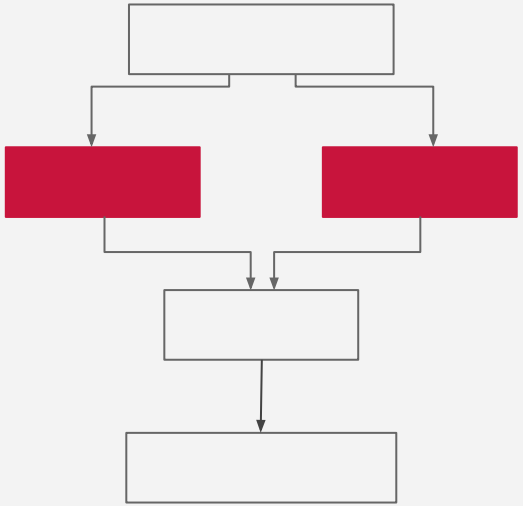
$$\text{Time} = t_{\text{read}}$$

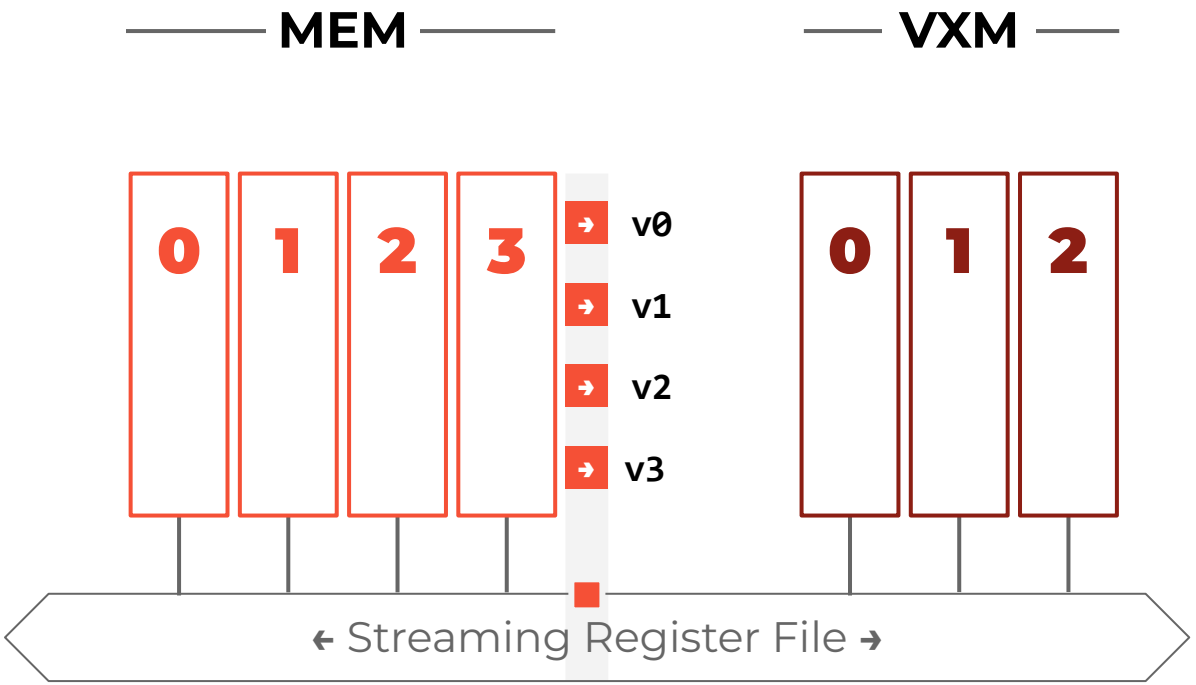




On GroqChip

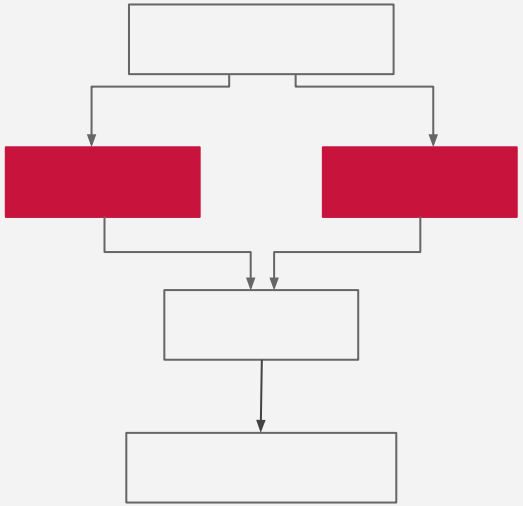
$$\text{Time} = t_{\text{read}} + 0.25 * t_{\text{travel}}$$

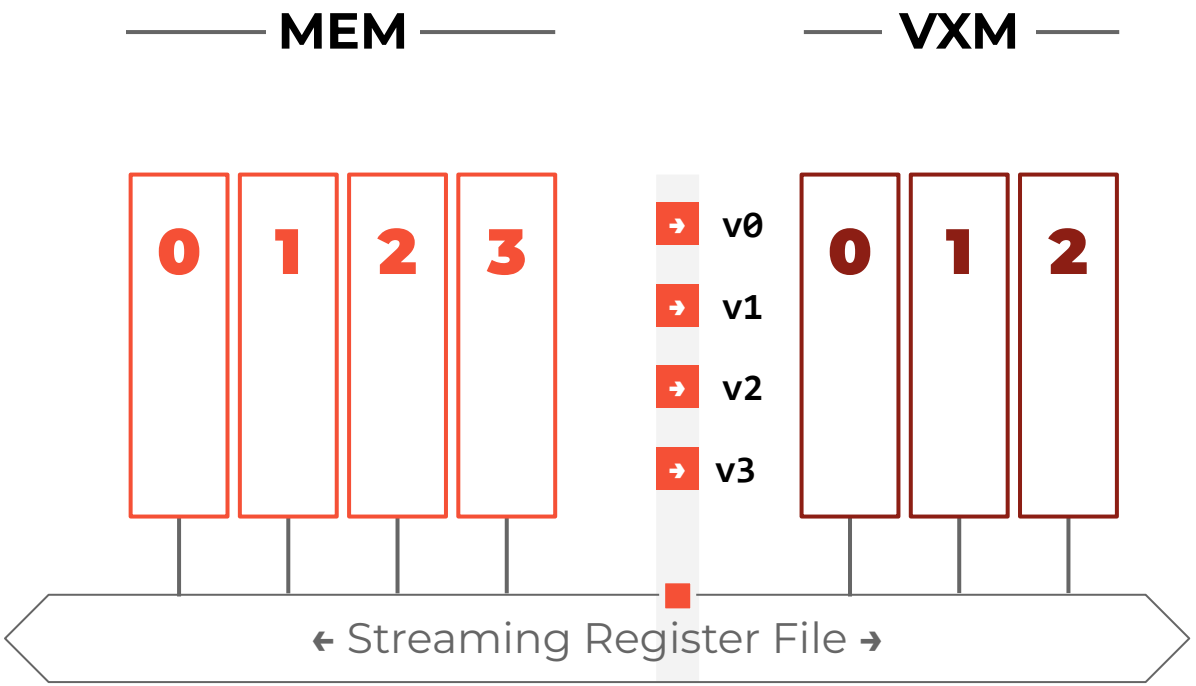




On GroqChip

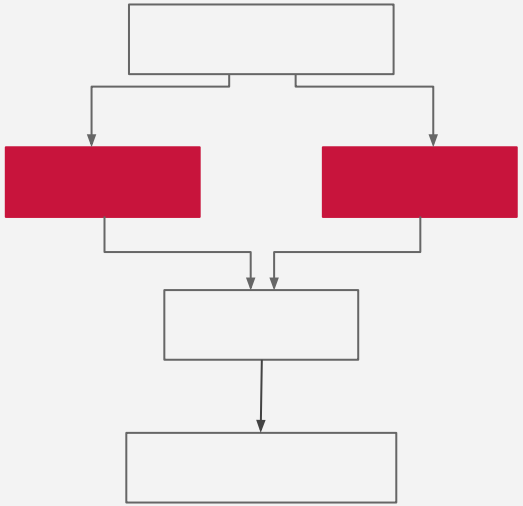
$$\text{Time} = t_{\text{read}} + 0.5 * t_{\text{travel}}$$

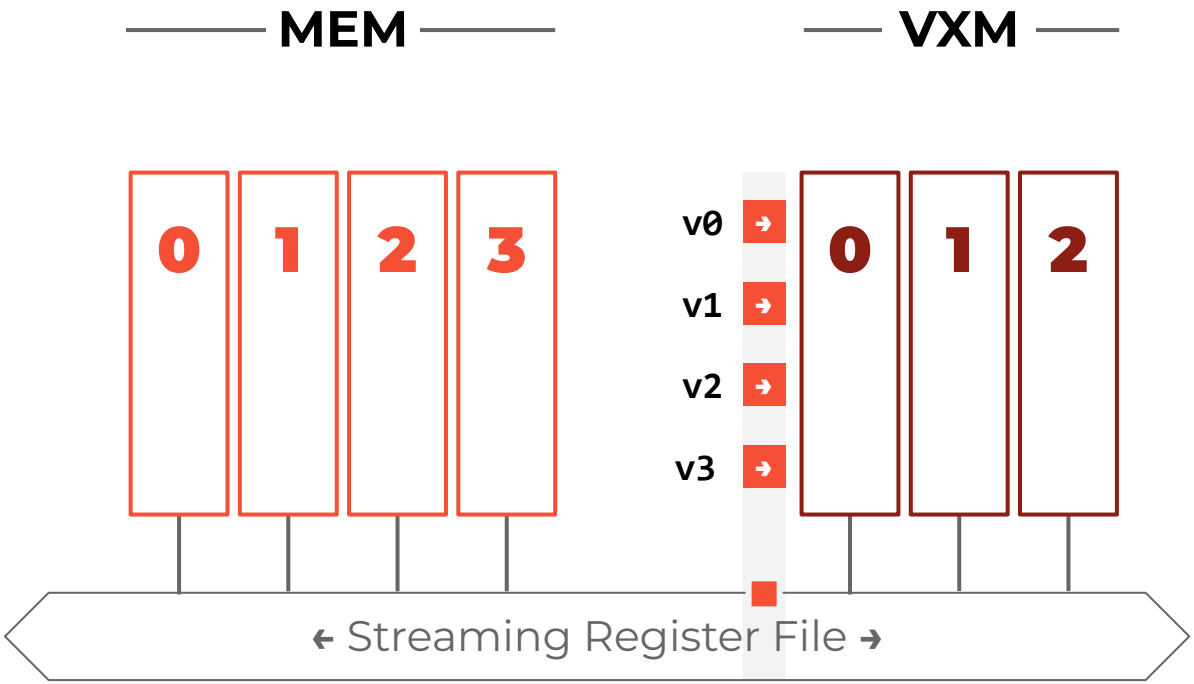




On GroqChip

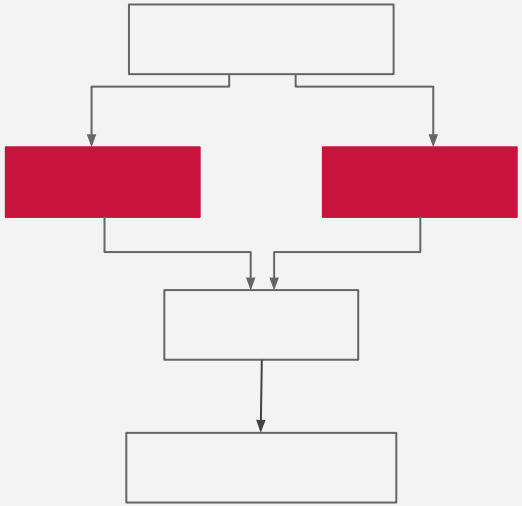
$$\text{Time} = t_{\text{read}} + 0.75 * t_{\text{travel}}$$

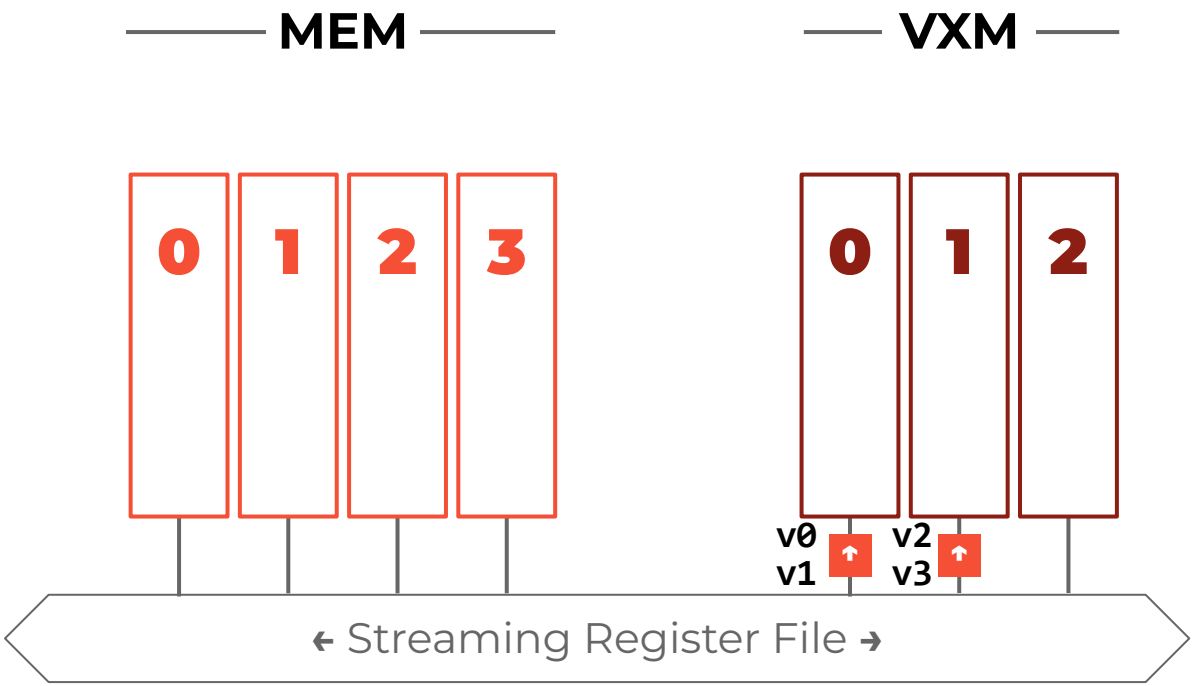




On GroqChip

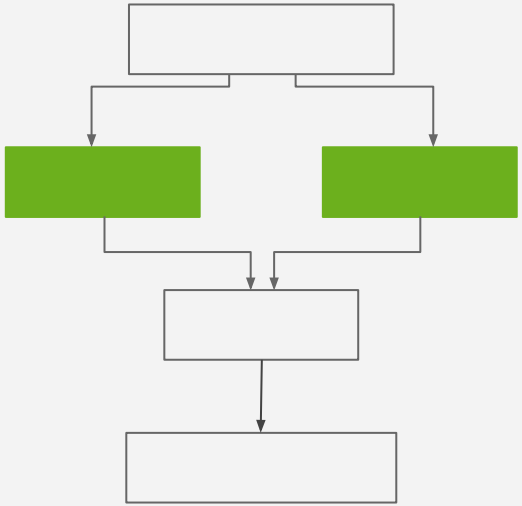
$$\text{Time} = t_{\text{read}} + t_{\text{travel}}$$

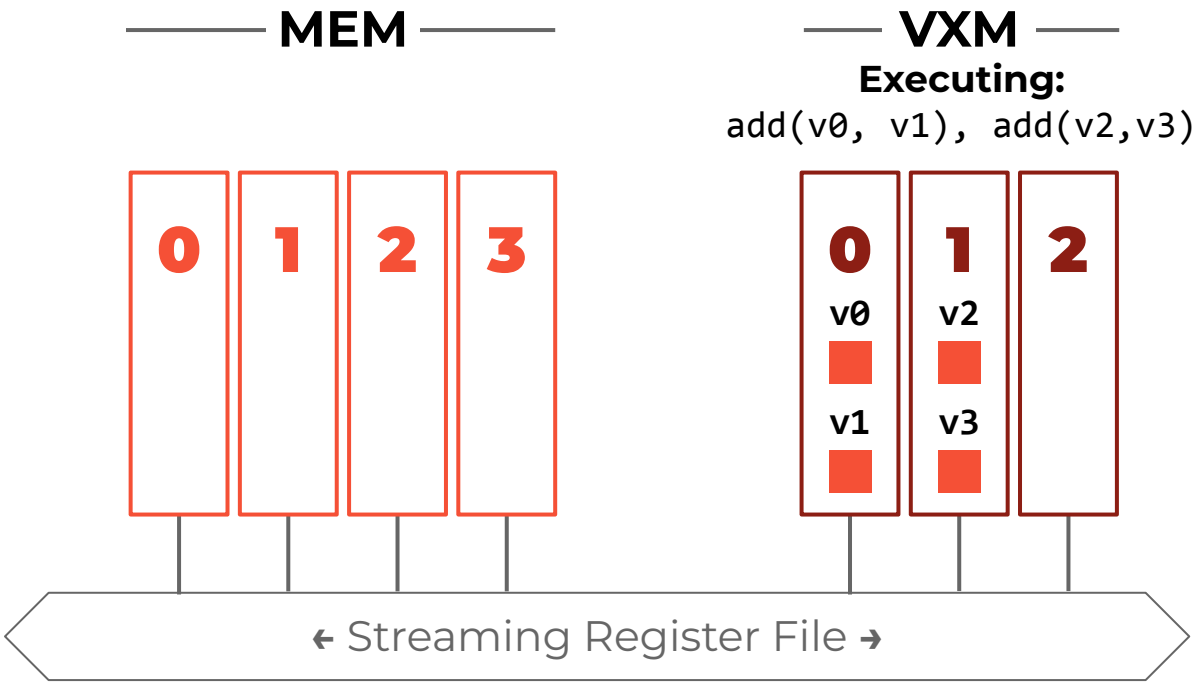




On GroqChip

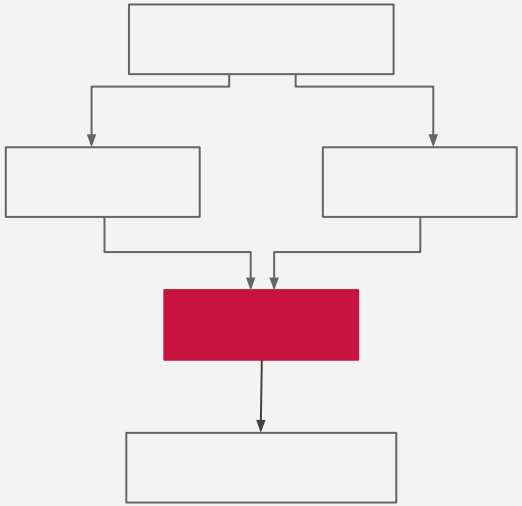
$$\text{Time} = t_{\text{read}} + t_{\text{travel}}$$

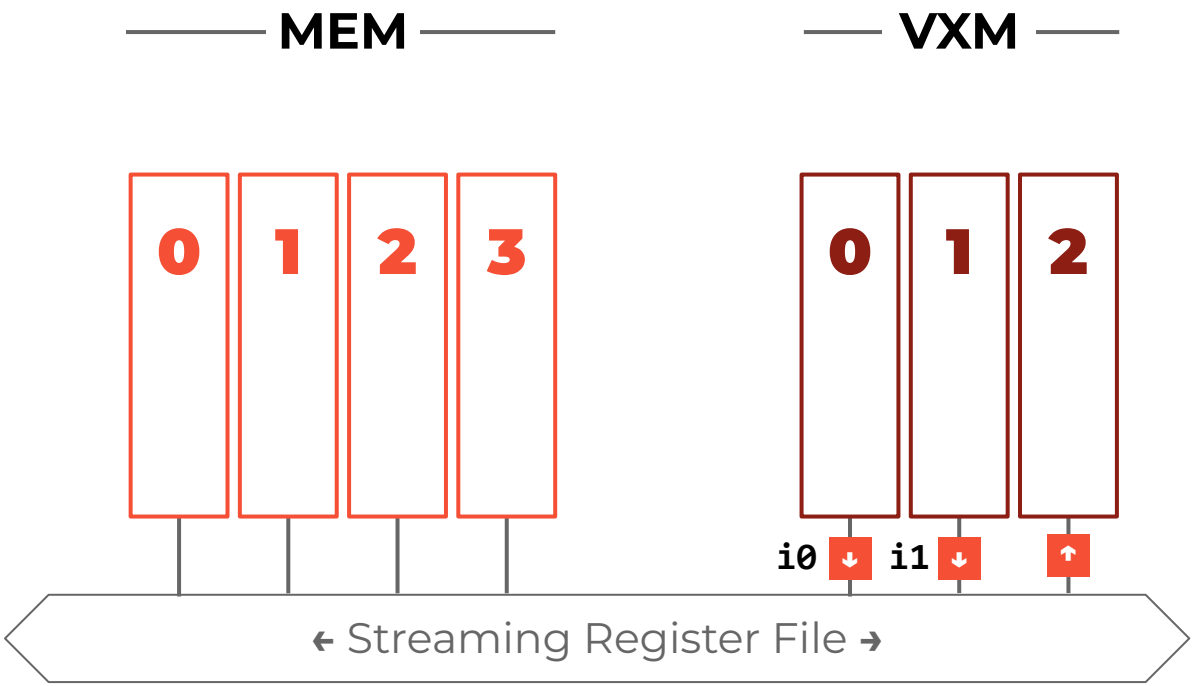




On GroqChip

$$\text{Time} = t_{\text{read}} + t_{\text{travel}} + t_{\text{add}}$$

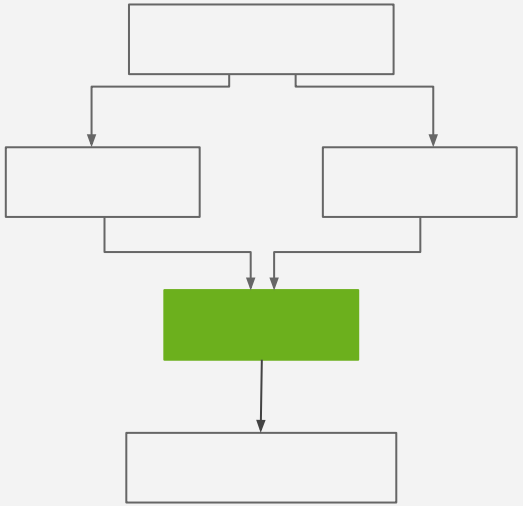


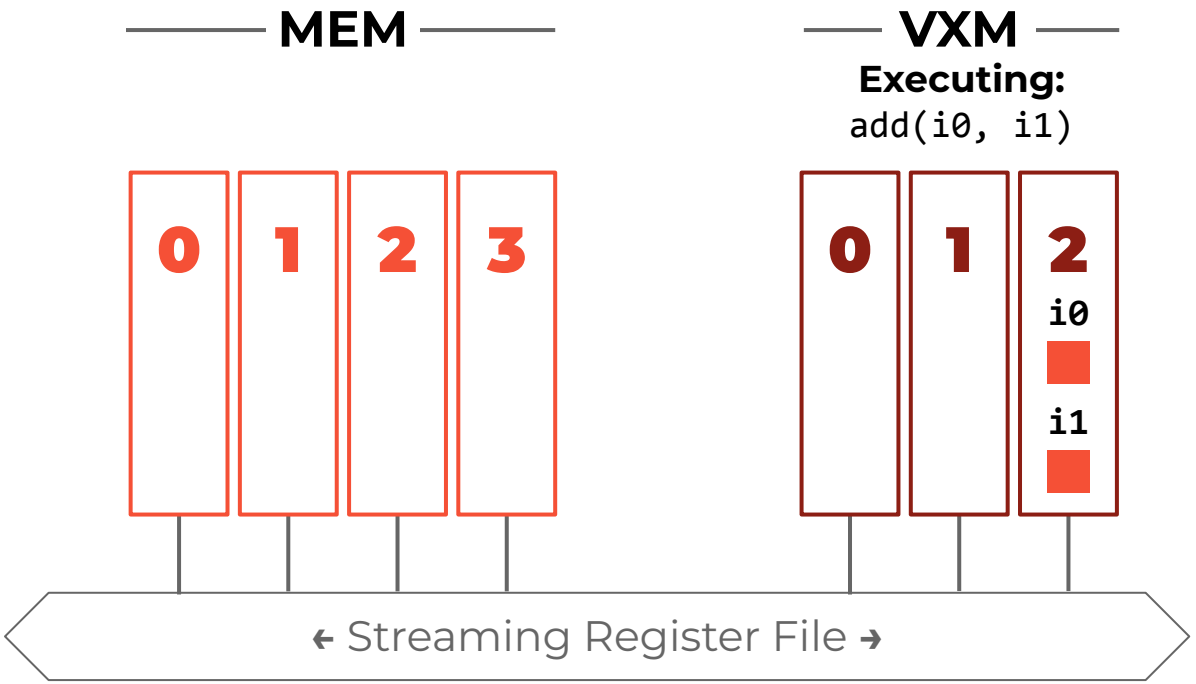


 Building Abstractions at the Hardware-software Boundary

On GroqChip

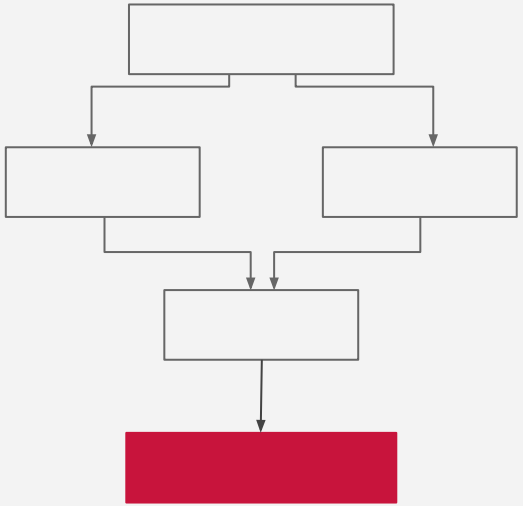
$$\text{Time} = t_{\text{read}} + t_{\text{travel}} + t_{\text{add}}$$

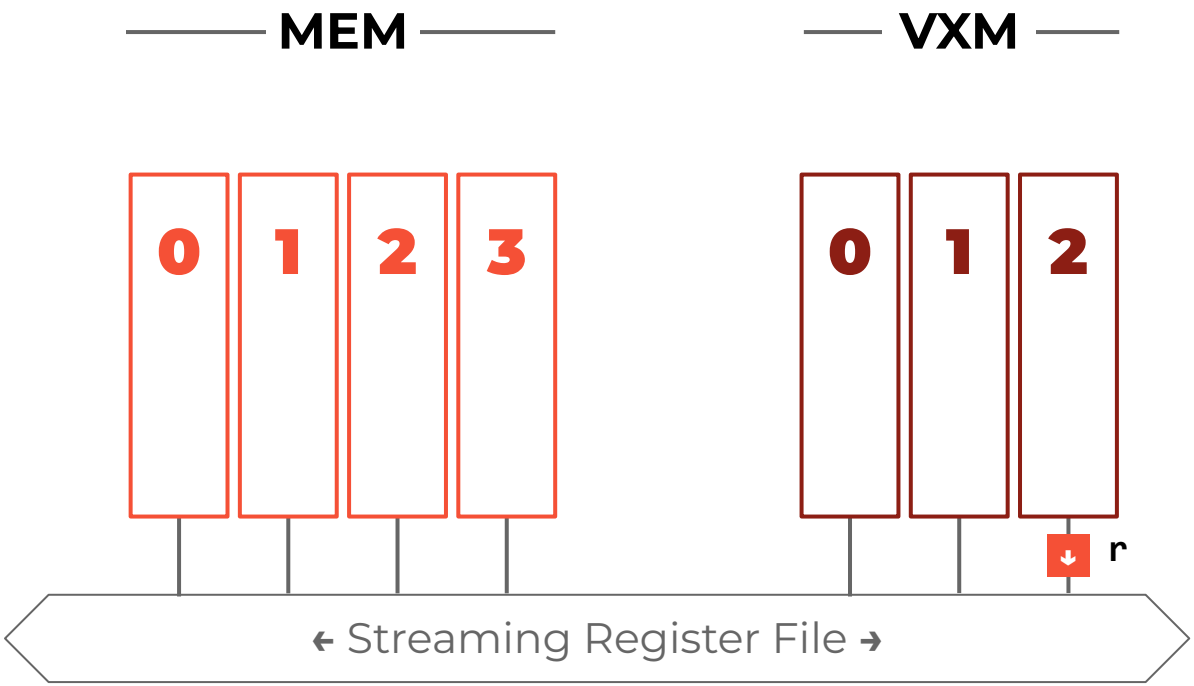




On GroqChip

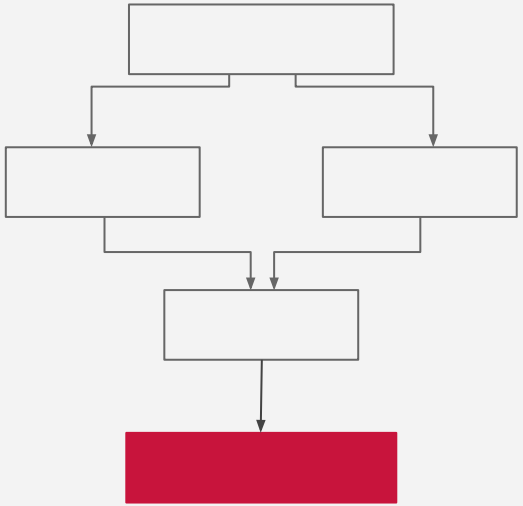
$$\text{Time} = t_{\text{read}} + t_{\text{travel}} + 2 * t_{\text{add}}$$

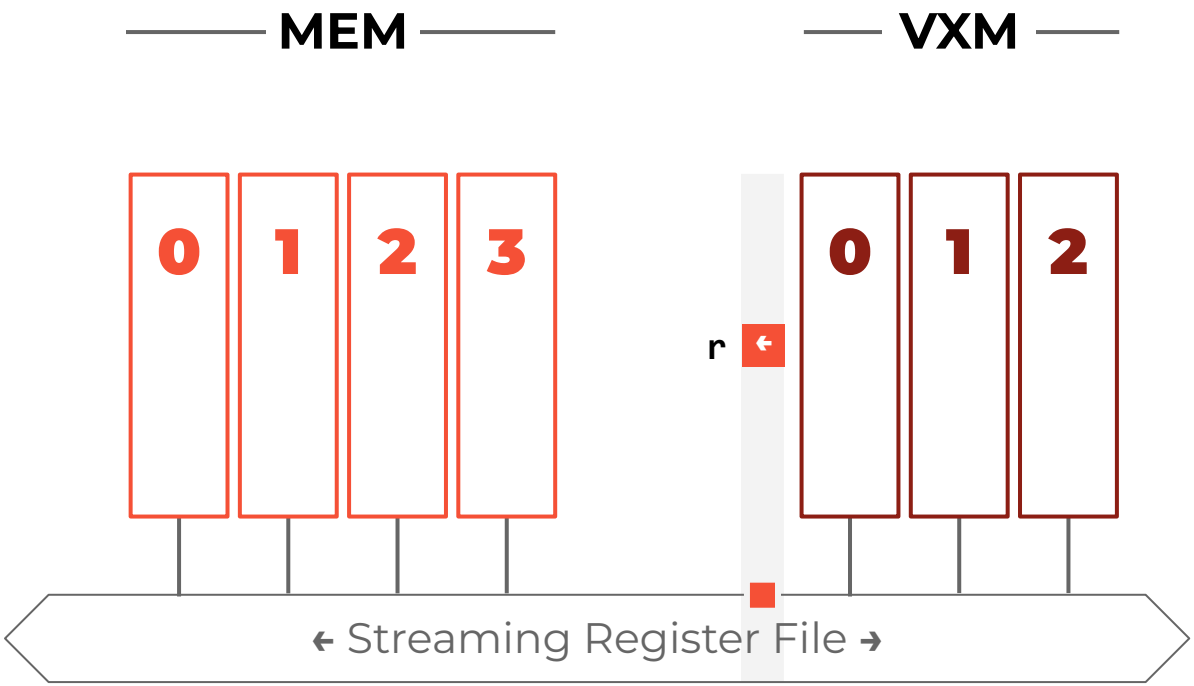




On GroqChip

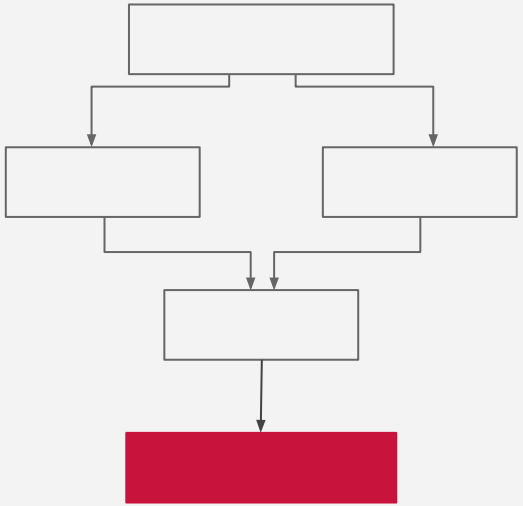
$$\text{Time} = t_{\text{read}} + 1.25 * t_{\text{travel}} + 2 * t_{\text{add}}$$

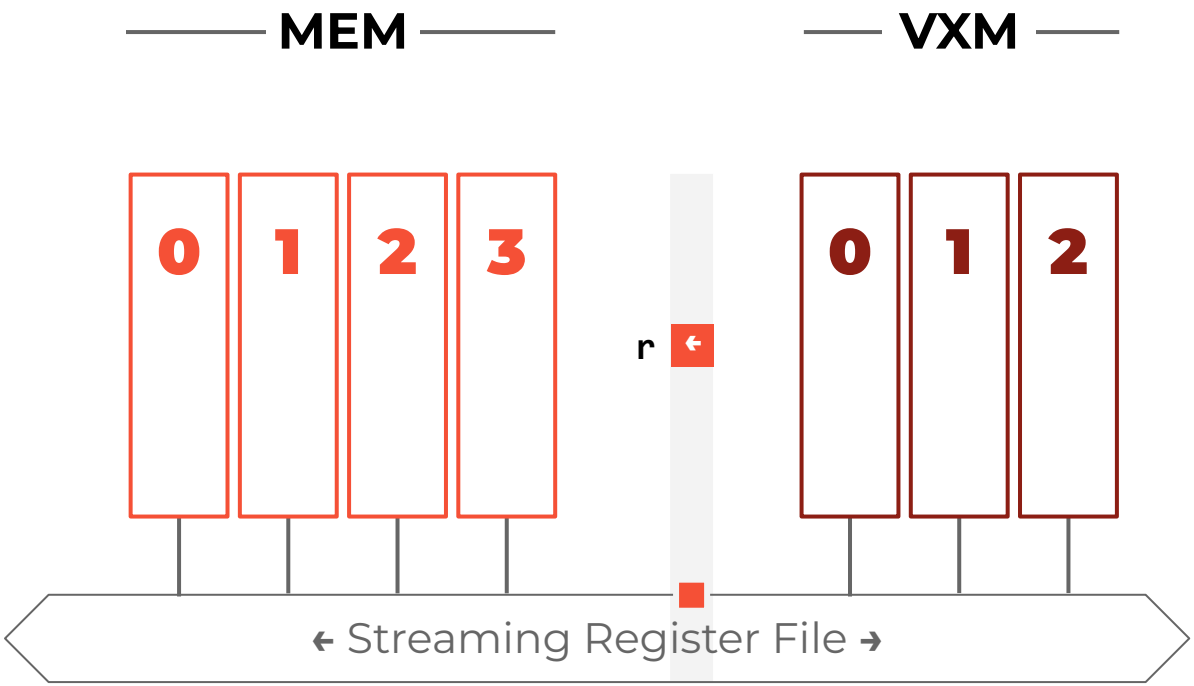




On GroqChip

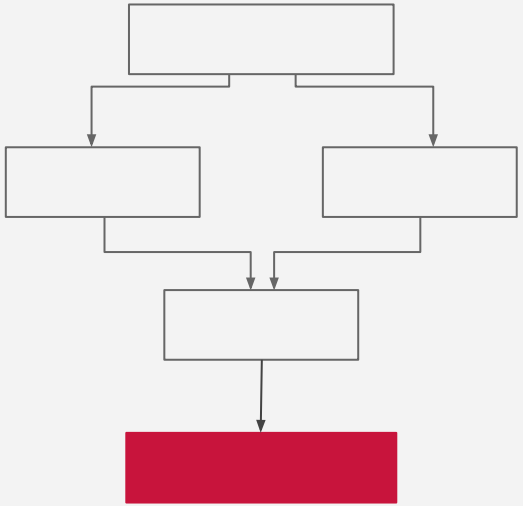
$$\text{Time} = t_{\text{read}} + 1.5 * t_{\text{travel}} + 2 * t_{\text{add}}$$

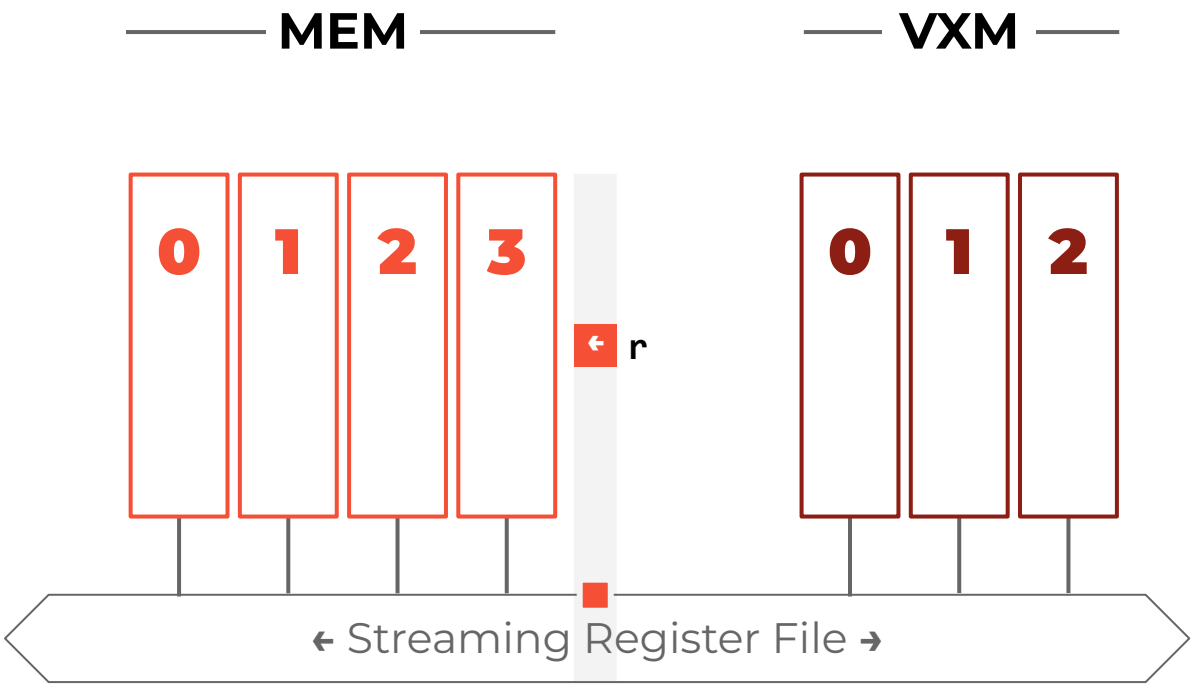




On GroqChip

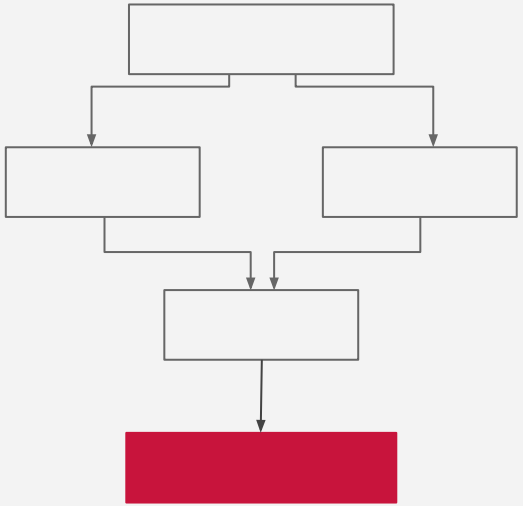
$$\text{Time} = t_{\text{read}} + 1.75 * t_{\text{travel}} + 2 * t_{\text{add}}$$

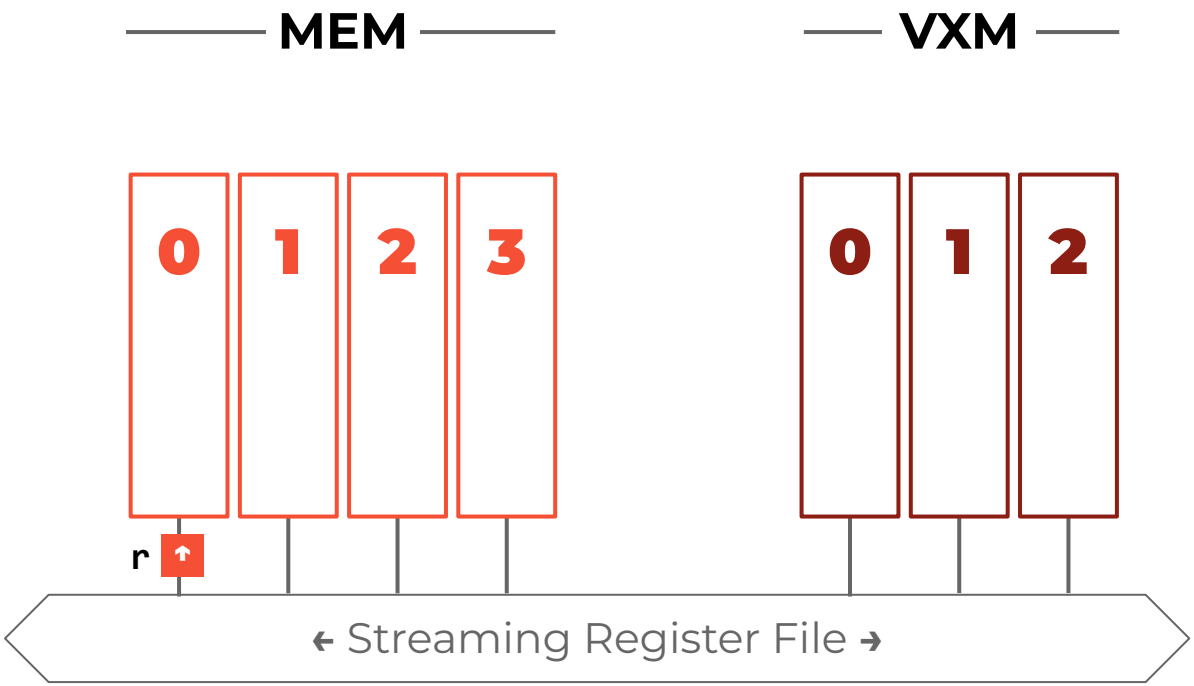




On GroqChip

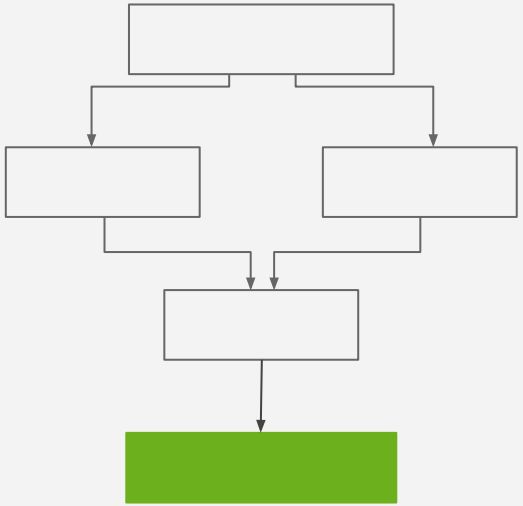
$$\text{Time} = t_{\text{read}} + 2 * t_{\text{travel}} + 2 * t_{\text{add}}$$

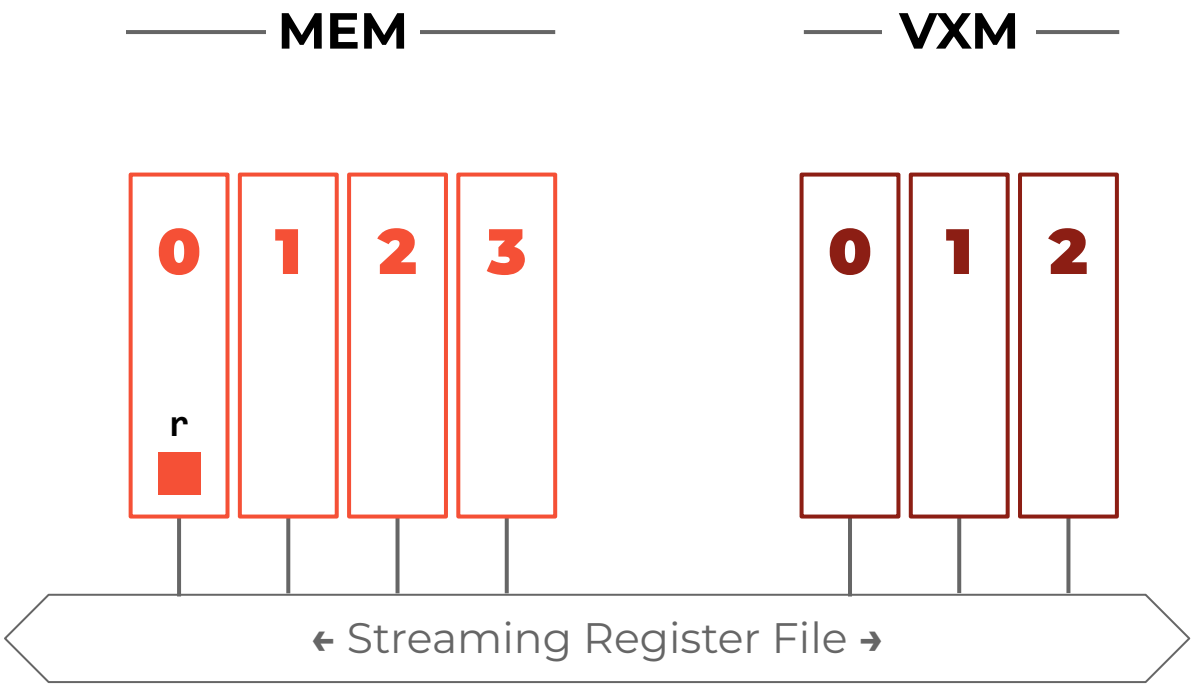




On GroqChip

$$\text{Time} = t_{\text{read}} + 2 * t_{\text{travel}} + 2 * t_{\text{add}} + t_{\text{store}}$$





Comparing Latencies

GroqChip

$$\text{Latency} = t_{read} + 2 * t_{travel} + 2 * t_{add} + t_{write}$$

Theoretical SIMD

$$\text{Latency} = t_{dispatch} + 2 * t_{add} + t_{store0} + t_{store1} + t_{load0} + t_{load1}$$

Unpredictable hardware causes pain for users

Predictable chips such as the GroqChip offers a holistic view of hardware

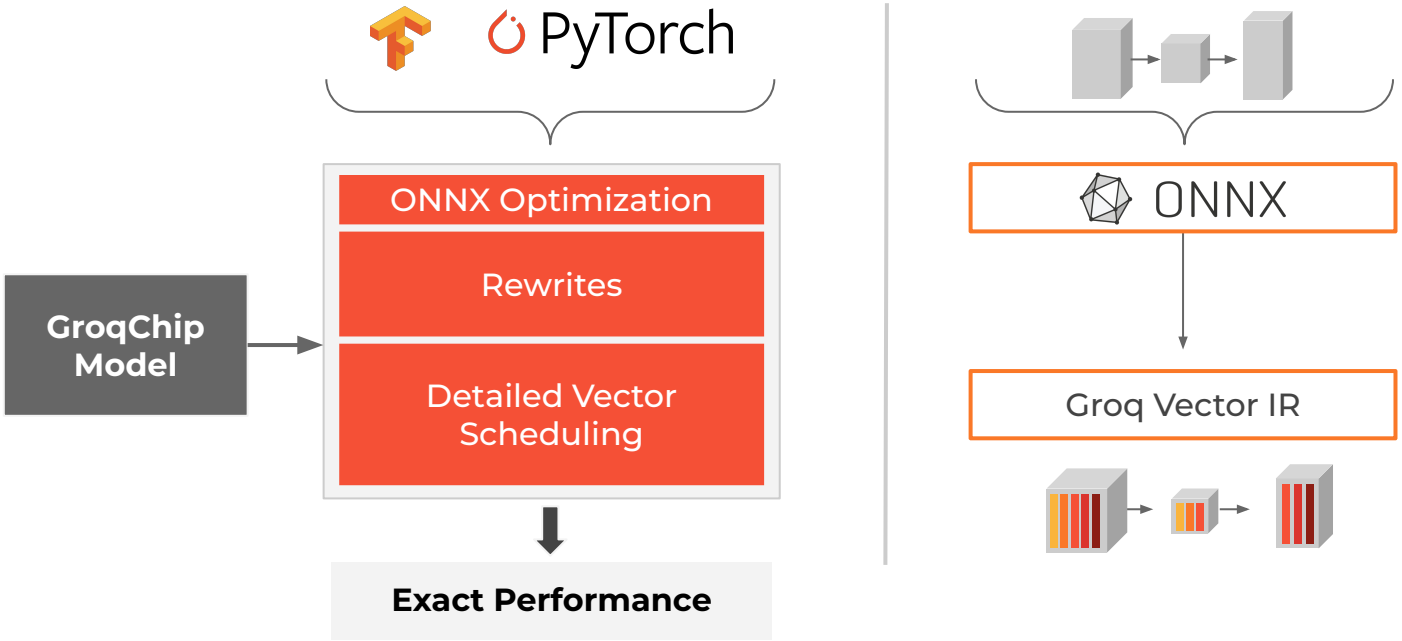
This holistic view enables effortless parallelism

Red = unresolvable at compile time
Green = resolvable at compile time

HARDWARE-SOFTWARE CO-DESIGN WITH The Groq Compiler

Groq Compiler

Flow



Kernel-less approach to HPC compilation

Building an Abstraction Model of GroqChip

Architecture provides a natural fit for an abstraction model containing all information needed for compilation

Specialized FUs with common interface ➔ C++ templated polymorphism

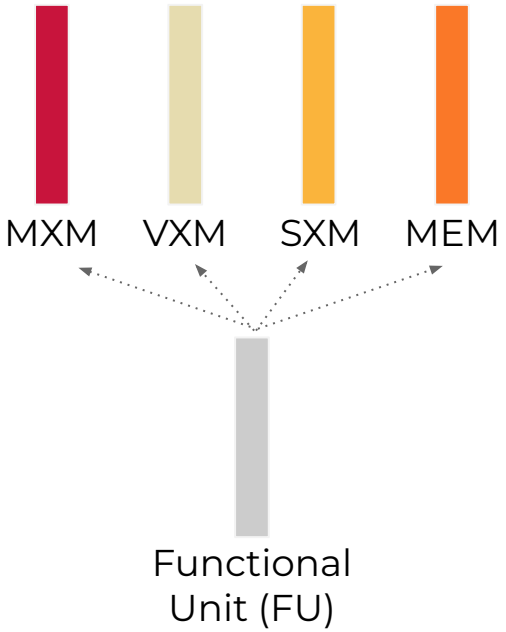
All instruction timing quickly resolvable at compile time via simple lookup table

All data movement quickly resolvable at compile time

- One dimensional interconnect abstracted as “timezone” indices

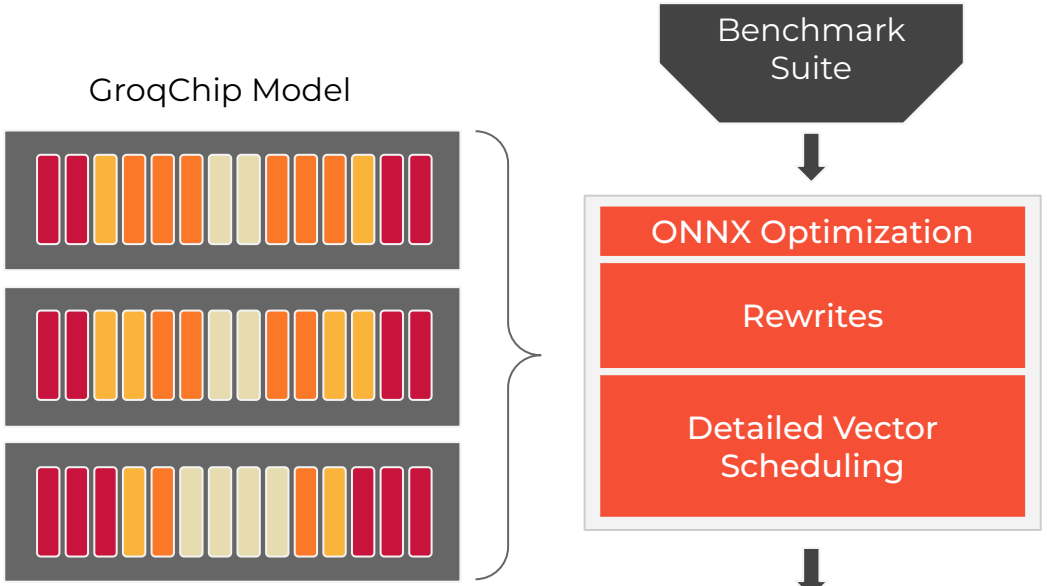
Efficient cycle-accurate resource allocation tracking via specialized data structures

- E.g. Bitvectors for fast range-allocation & lookup across time + space



Hardware-software Co-design

Compiler-driven architecture exploration



Fast architecture exploration using chip model abstraction

Deterministic architecture allows exact performance to be known at compile time - no hardware/simulation needed!

High-level Chip Models in FPGA CAD

FPGA CAD tools have long been an example of hardware-software co-design via software abstractions of the chip

FPGA CAD compiles HDL down to a bitstream configuring the chip (LUTs, DSPs, BRAM, routing)

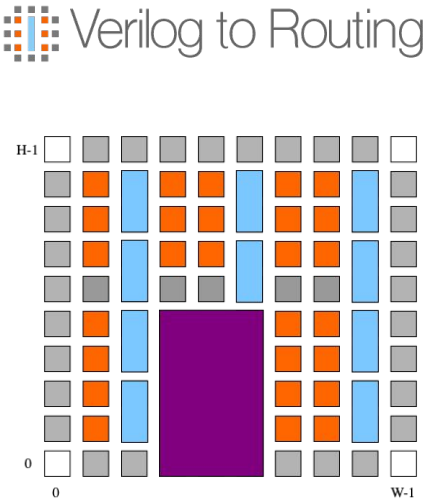
Metrics of the bitstream are statically determined (resource utilization, fmax)

Compilation requires a detailed, low-level chip model

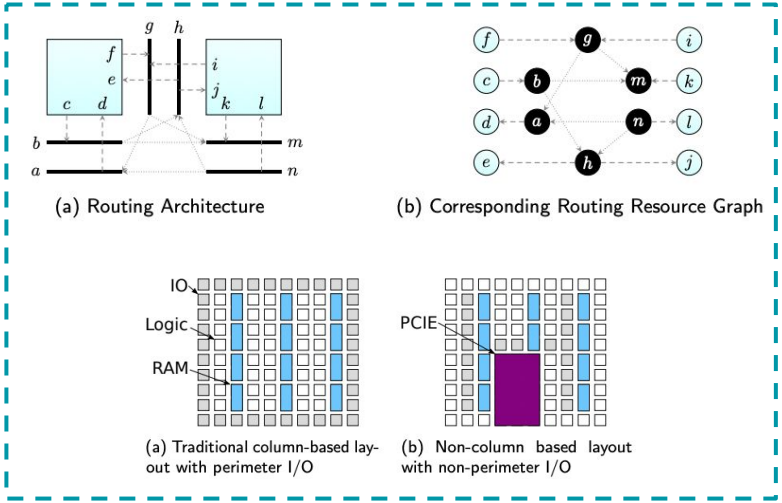
Verilog-to-Routing (VTR) is an open-source FPGA CAD tool used for FPGA architecture exploration

VTR can compile for any FPGA architecture that fits within its chip model framework

Originally introduced in VPR [Betz & Rose, FPGA'00]



High-level Chip Models in FPGA CAD



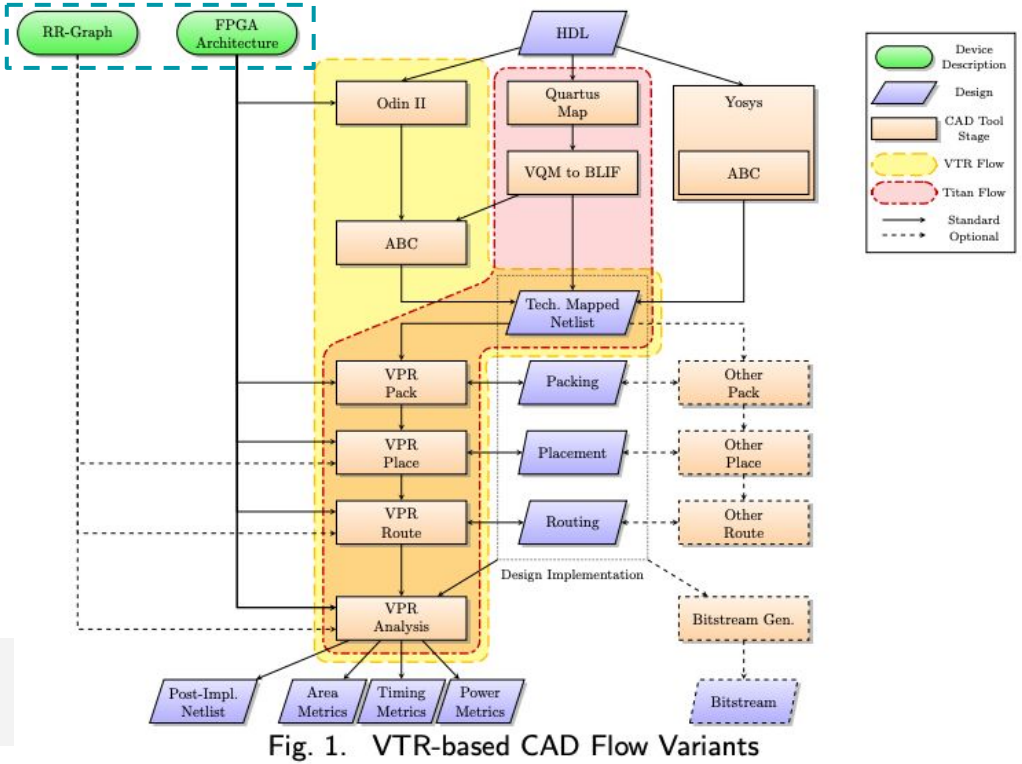


Fig. 1. VTR-based CAD Flow Variants

Murray, et al. VTR 8: High Performance CAD and Customizable FPGA Architecture Modelling

Final Thoughts

A New Golden Age of Computing

TSP, TPU, RISC-V, GPU, FPGA, CGRA

The Contract Between Software and Hardware Has Been Reopened for Negotiation

What responsibilities are owned by the compiler vs. the hardware?

Abstractions Are Key

If low-level control is given to software, can it be effectively used?

If low-level control is hidden from software, what performance may be lost?

What is the hardware cost?

Will the abstraction “leak”?

Designing a Good Abstraction Requires Thinking Carefully About the Target Applications

Data-flow vs control-flow

Hardware-software co-design is key for developing end-to-end solutions that meets the challenging problems we face today

Cpp North

Thank You

Contact me or
the Groq team at
info@groq.com

WE ARE HIRING

FOLLOW US ON:

groq™