PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference

Aayush Ankit[1,3], Izzat El Hajj[2,4], Sai Rahul Chalamalasetti[3], Geoffrey Ndu[3], Martin Foltin[3], R. Stanley Williams[3], Paolo Faraboschi[3], Wen-mei Hwu[4], John Paul Strachan[3], Kaushik Roy[1], Dejan S Milojicic[3]

PURDUE UNIVERSITY[1]

AMERICAN UNIVERSITY OF BEIRUT[2]

HEWLETT PACKARD ENTERPRISE[3]

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN[4]

SRC JUMP

Hewlett Packard Enterprise

C-BRIC

Motivation: Human vs. Machine Chronicles

1997

IBM Deep Blue vs. Kasparov
IBM RS/6000 32-node server
~15000W

2011

IBM Watson vs. Ritter & Jennings
90 Power 750 Express servers
~200000W

2016

Google AlphaGo vs. Sedol
(1920 CPUs, 280 GPUs)
~300000W

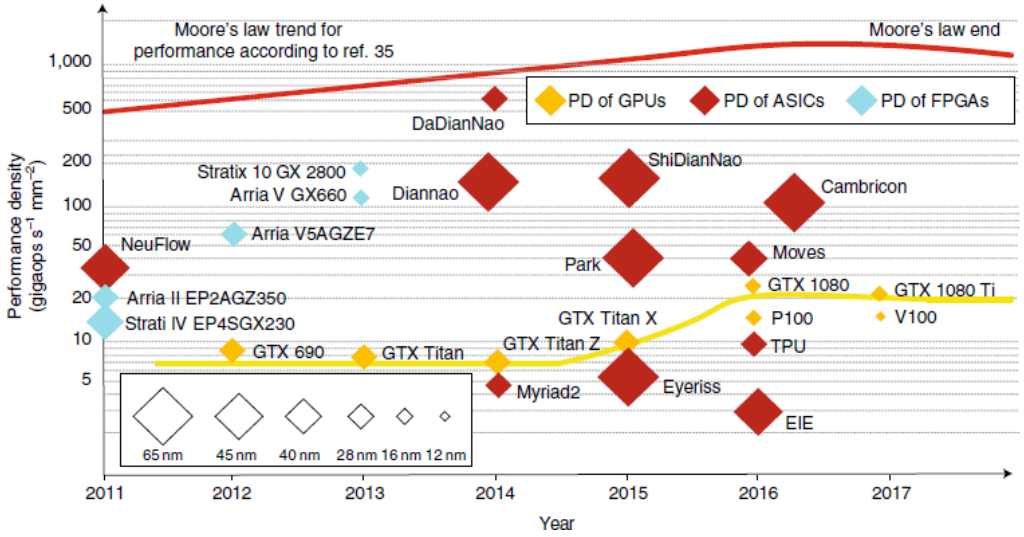
Algorithm performance moving closer

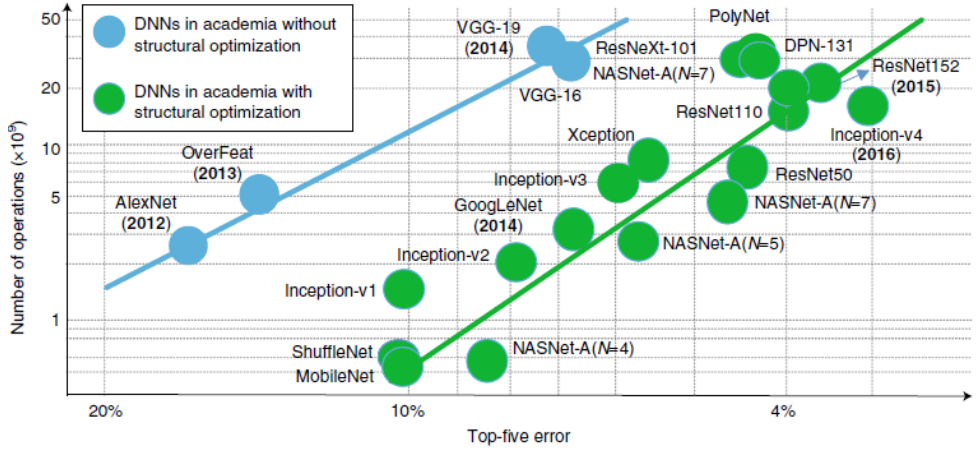
Hardware cost moving farther

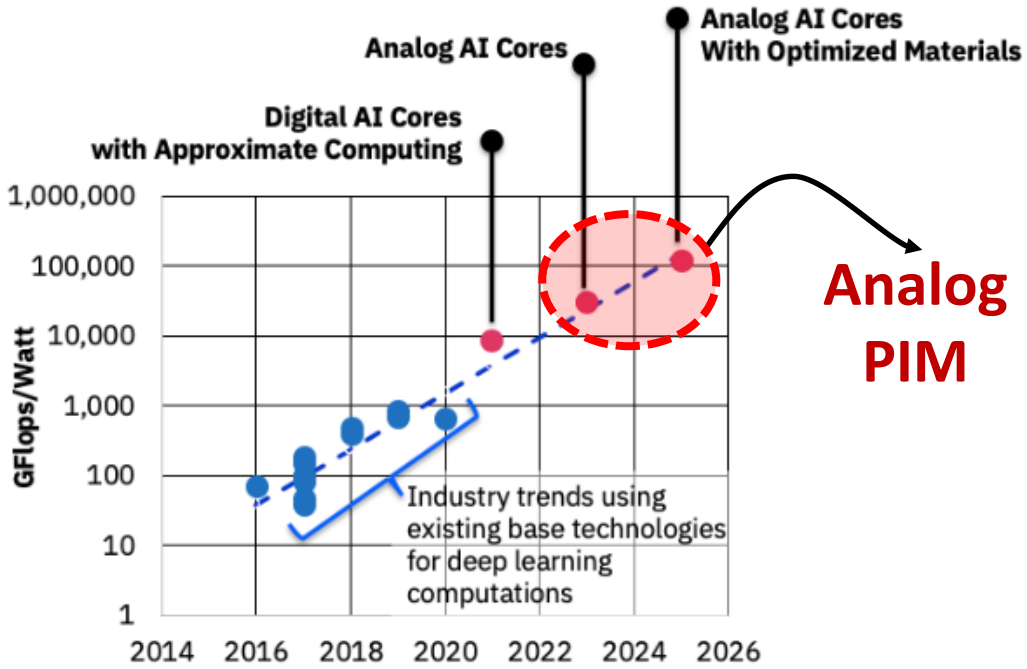
"Aspirations have grown faster than the technology available to satisfy them".

C-BRIC

Motivation: Need to Look Beyond CMOS Digital Accelerators







Ref: IBM AI Hardware Research (https://www.ibm.com/blogs/research)

➢ **ML models are growing faster than CMOS hardware.**

Industrial presence

Hewlett Packard Enterprise

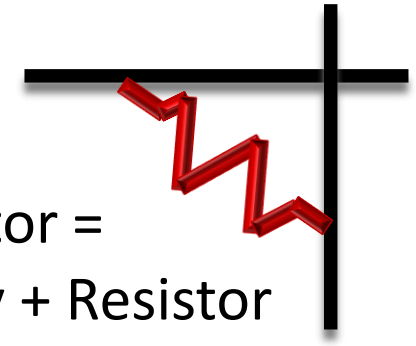
IBM

MYTHIC

SYNTIANT

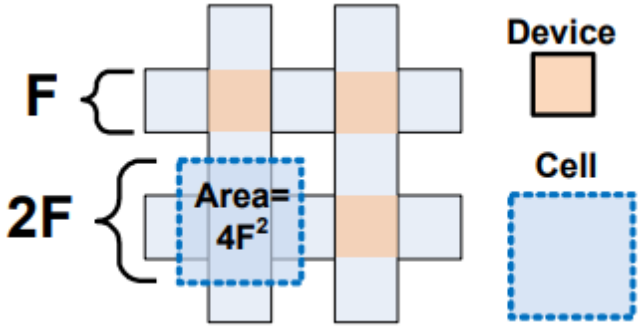
C-BRIC

Background: Memristive Crossbars

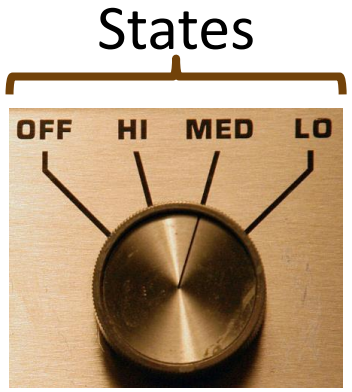


Memristor =
Memory + Resistor

=



+



Memory aspects:
High Density + **On-Die Storage**

Compact cell structure

Tunable Resistance

Array Density: 160MB/mm^2
- Mitigates off-chip data access

>100×

Cell area is 4F^2 vs 120F^2 for CMOS (SRAM).

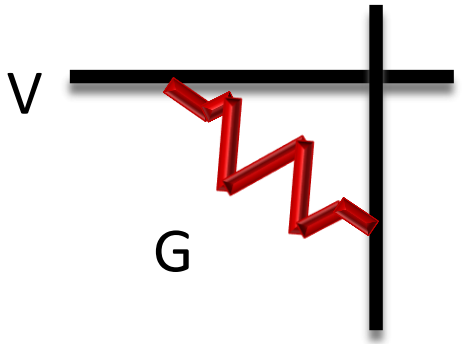
30×

2-6 bits per cell vs 1-bit for CMOS (SRAM).

6×

C-BRIC

Background: Memristive Crossbars



$$I = V * G$$

Compute Aspects:

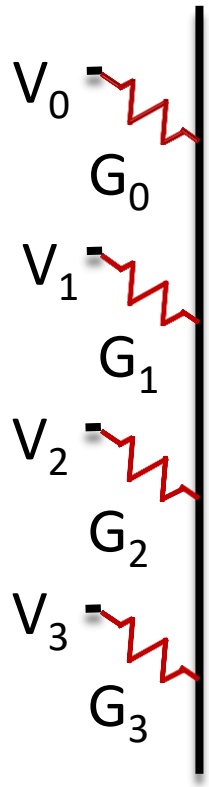
Analog Multiplication

Analog MVM: 1.34pJ/op*

- Efficient compute

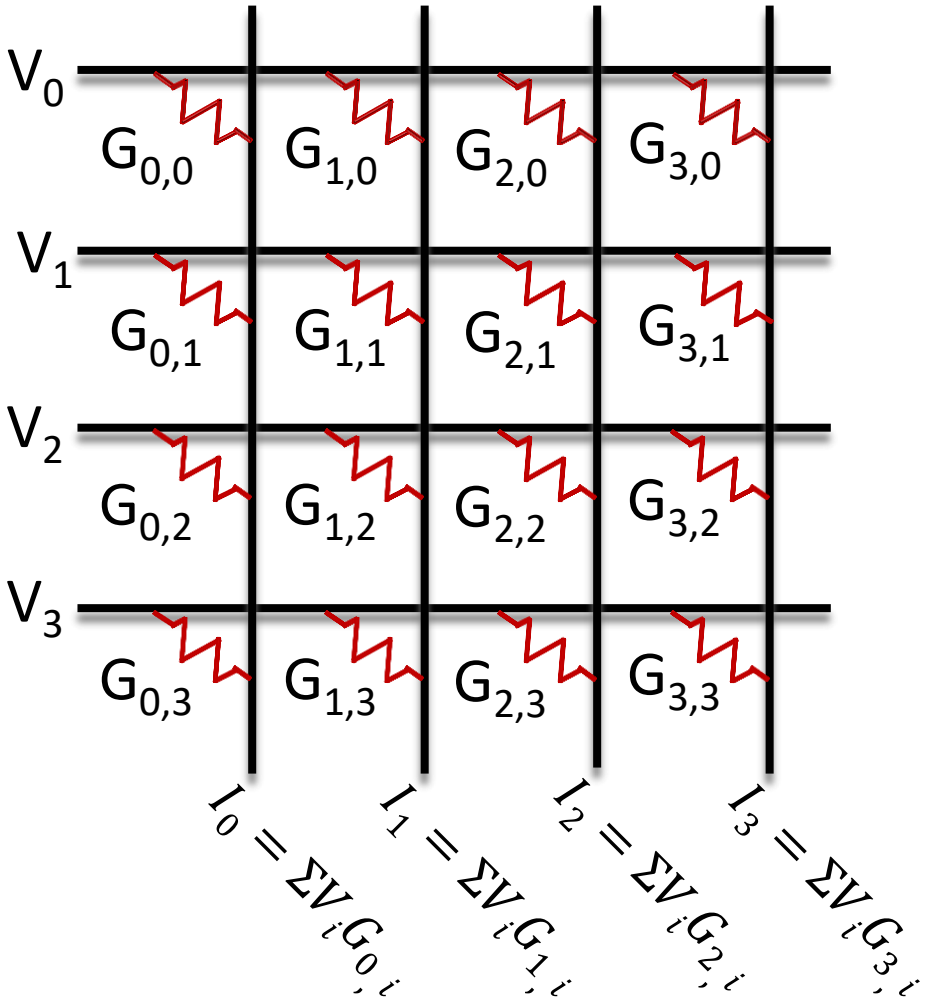
>4×

*32nm technology



$$I = I_0 + I_1 + I_2 + I_3 = \Sigma V_i G_i$$

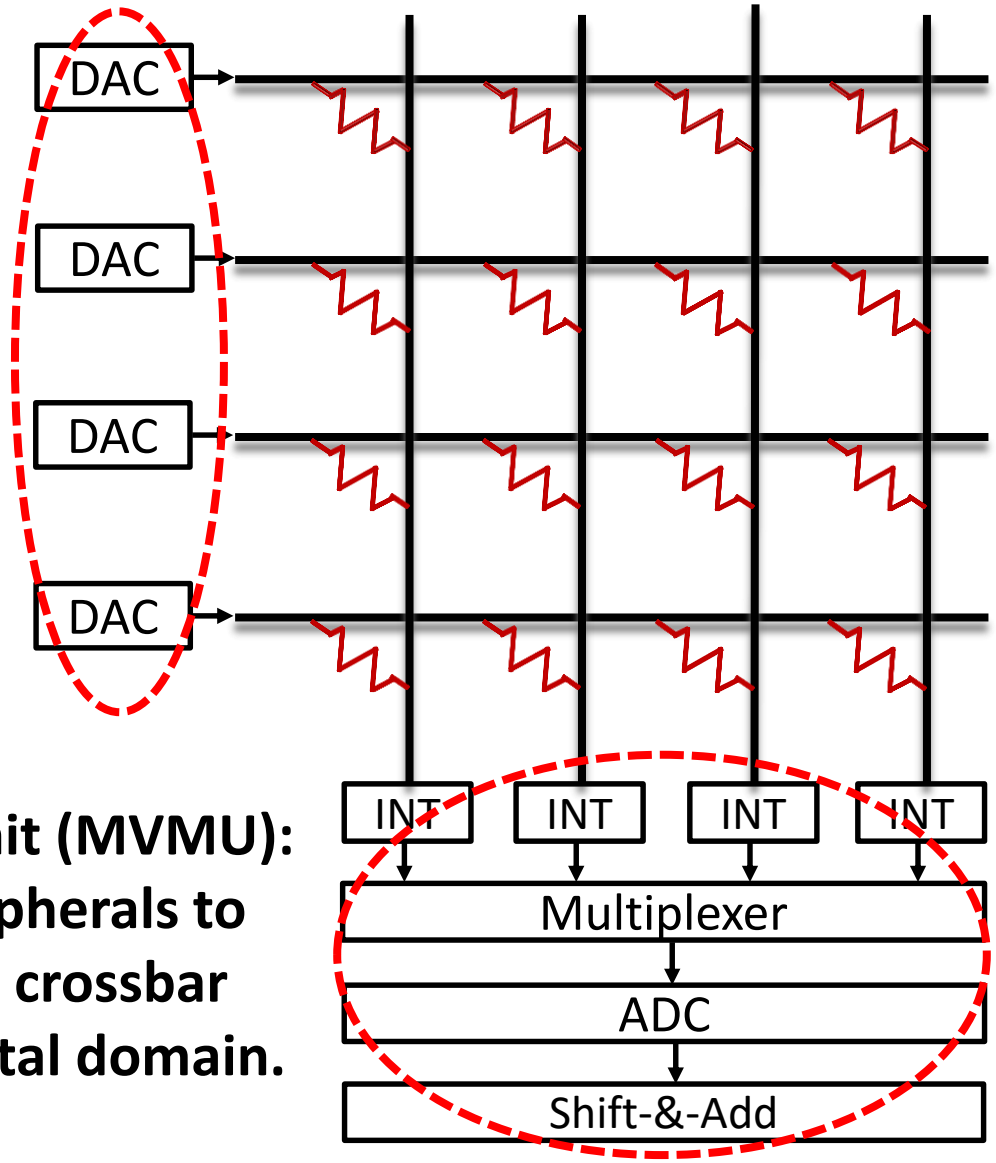
Analog Dot Product



Analog Matrix Vector Multiplication (MVM)

C-BRIC

Background: Past Memristive Based Accelerators

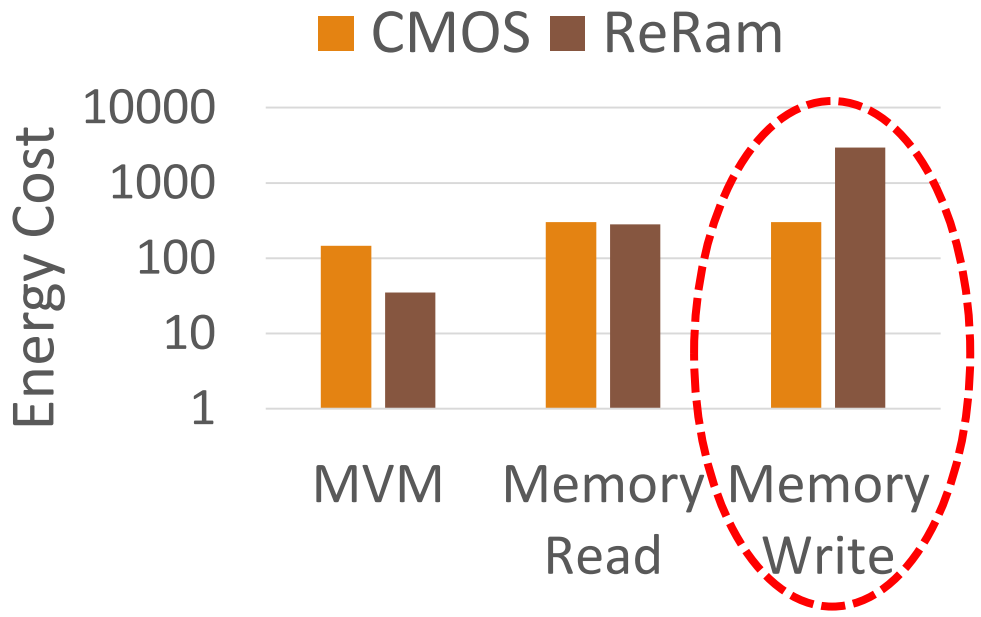


MVM Unit (MVMU): Add peripherals to interface crossbar with digital domain.

- Past accelerators: RENO [DAC'15], ISAAC and PRIME [ISCA'16] achieve significant benefits over CMOS accelerators for limited applications (1 or 2).
- Memristive crossbars are ***not drop-in replacement*** of traditional memory structures (Cache, Register File).
  - General purpose ML architecture ? ☹
  - Programmability ? ☹
- **PUMA is the first general-purpose and programmable accelerator with memristive crossbars.**

C-BRIC

PUMA Architecture: *Spatial scalability*

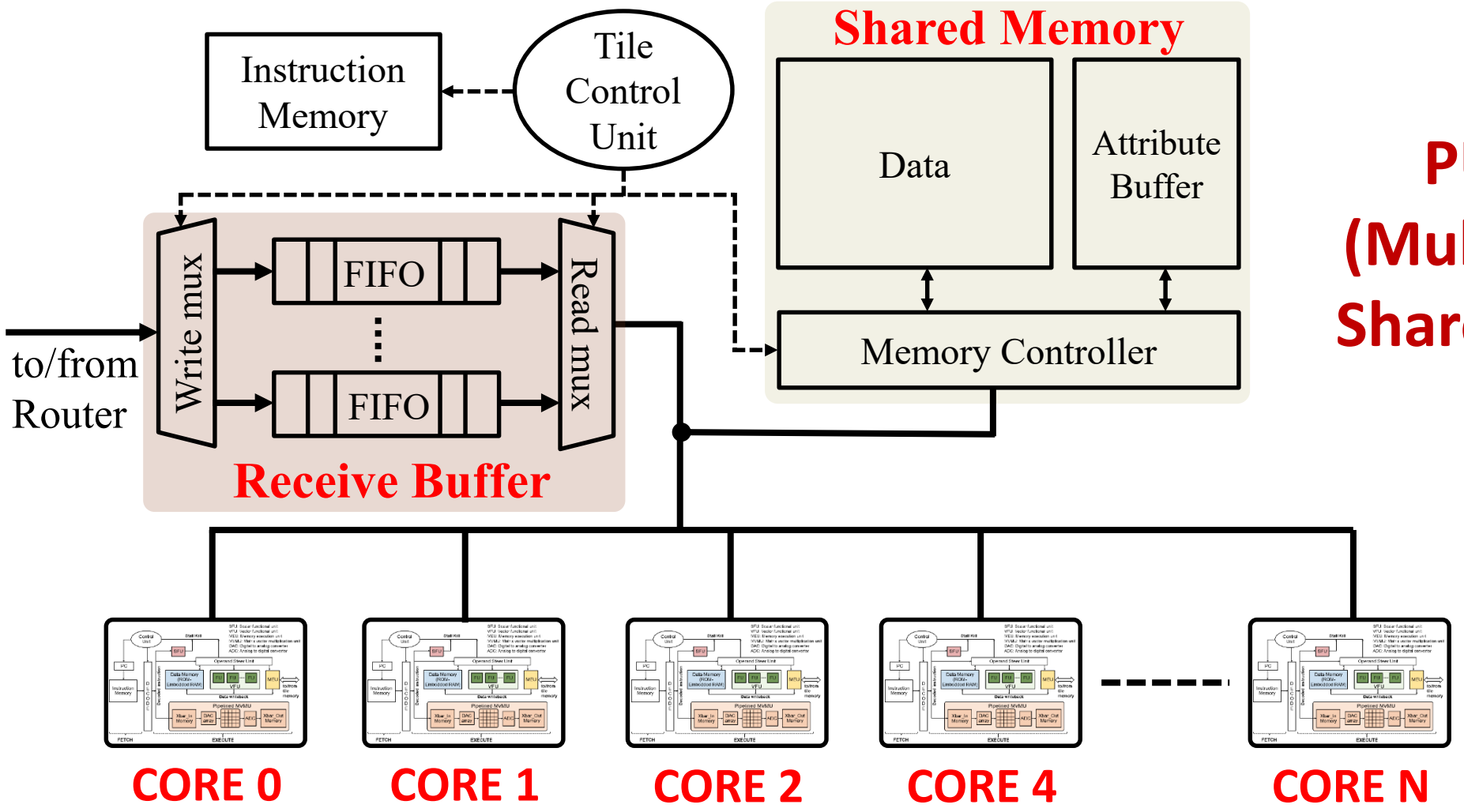


- **Expensive writes**
  - writes consume orders of magnitude more energy/latency than CMOS (SRAM)
- **Limited endurance**
  - typically 10^9 writes, CMOS has infinite endurance

- **Costly writes to memristive crossbars are alleviated in spatial architecture**
  - Model doesn't change during inference (trained once, millions of inference)
- High storage density enables spatial architecture
  - **State-of-the-art apps can be mapped in ~GPU die area, if the high storage density can be retained at system-level**

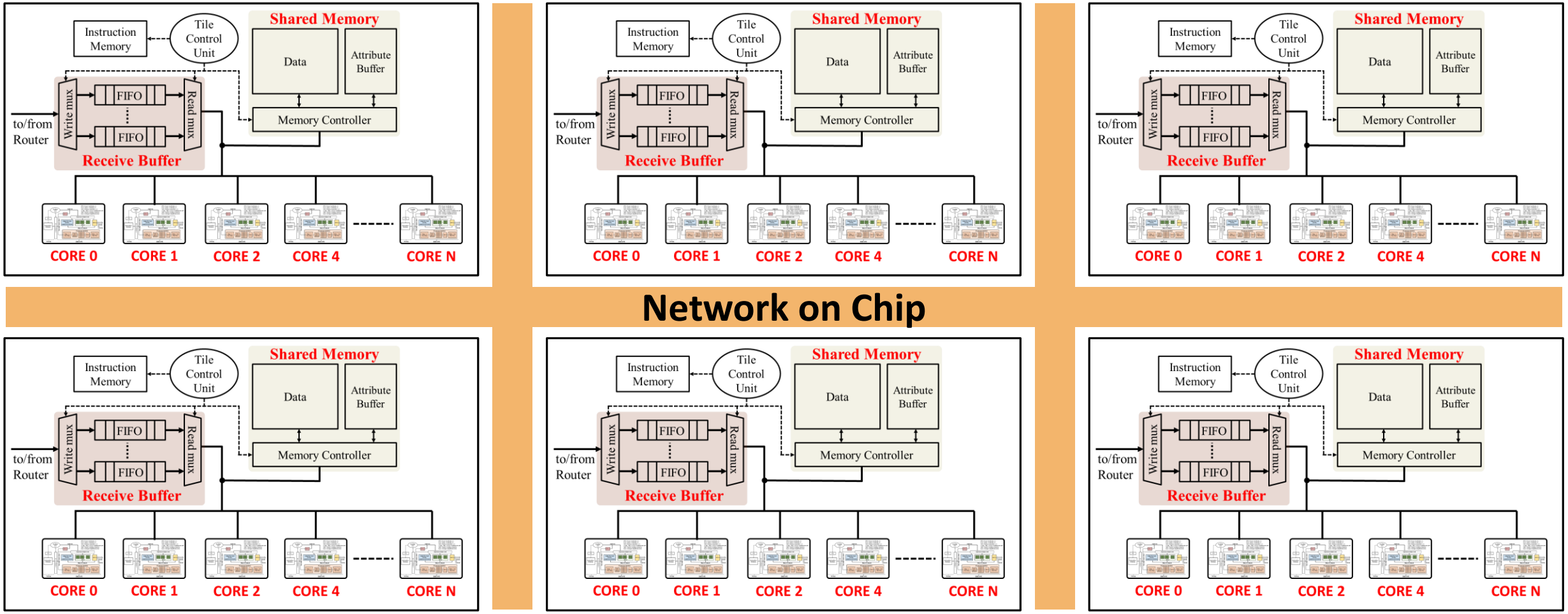
C-BRIC

PUMA Architecture: *Spatial scalability*



PUMA Tile (Multi-core with Shared Memory)

PUMA Architecture: *Spatial scalability*



Massively parallel accelerator –> Amenable to Data-Level Parallelism -> Highly efficient ML inference

C-BRIC

Key challenges

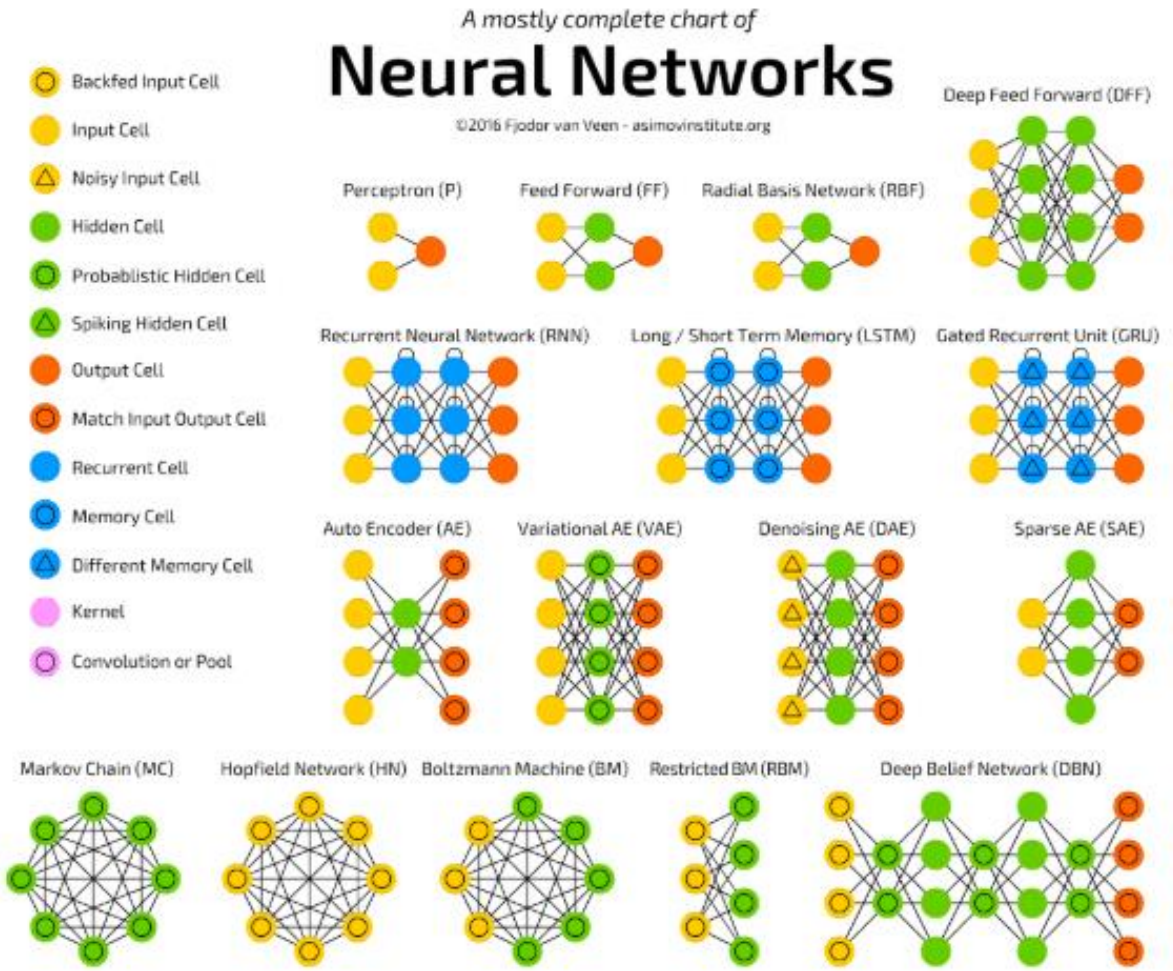
- Enable general-purpose and programmable architecture while preserving the crossbar storage density
  - **Memristive technology has orders of magnitude higher area-efficiency than CMOS technology, for both storage and compute**
  - Storage density drops from 160MB/mm^2 at array-level to 2.6MB/mm^2 at MVMU-level (ISAAC, ISCA'16) !!!

- Optimize data movements
  - **Weights remain stationary, but partial sums and activations lead to data movements**
  - Spatiality incurs high cost for data movement between cores

- **Proposal: Architecture-ISA-Compiler codesign.**

Contribution 1: *Domain-specific ISA*

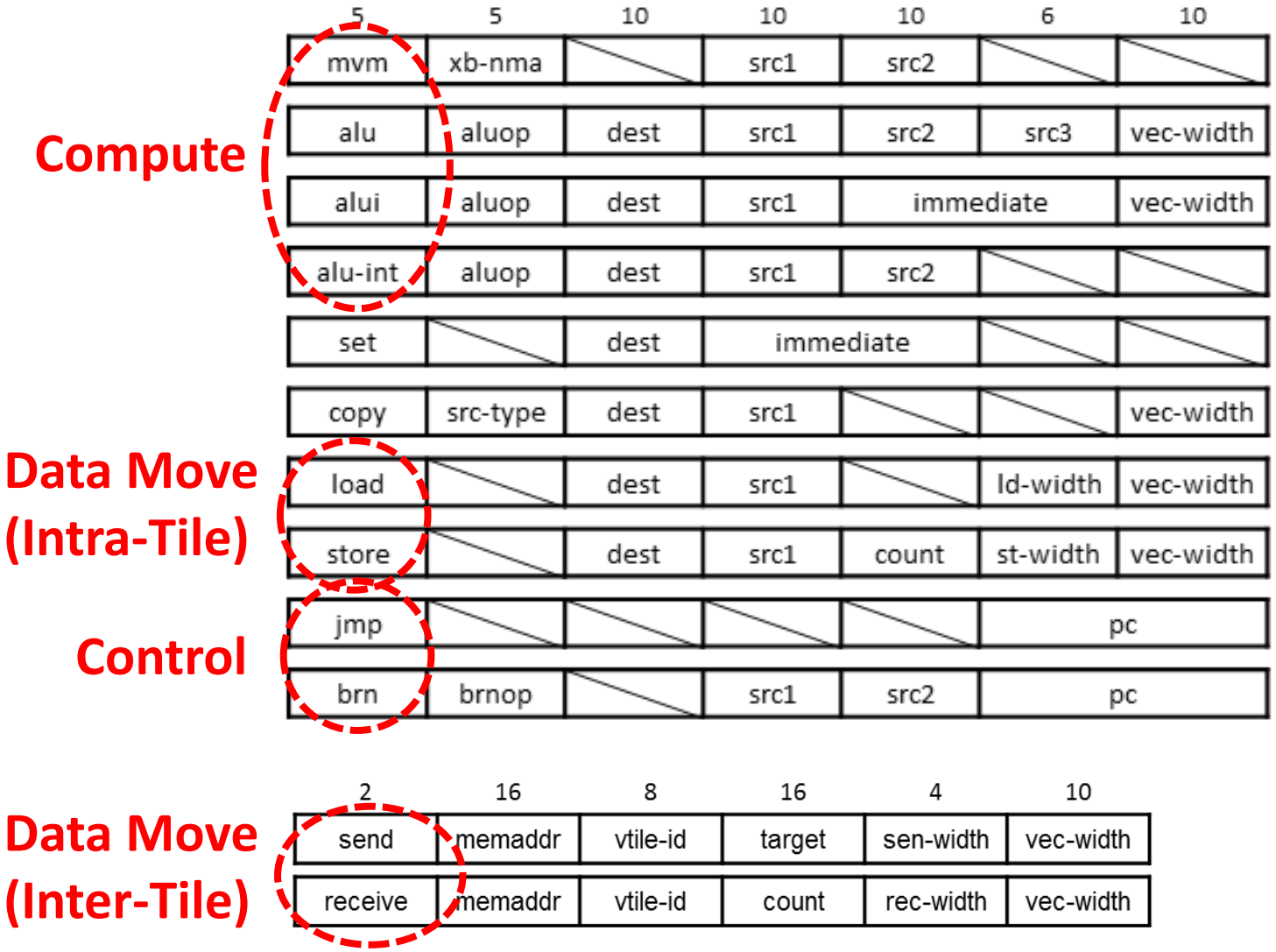
- Large variety of ML workloads
  - Support Vector Machines
  - Neural Networks (>27)
  - …
- State machine approach to represent a functional block, leads to high decoding overheads when scope of workloads is large
- **So many workloads, share many low-level operations**
- ISA for ML for memristive architecture
  - Fine grain instructions compose to make an ML workload
  - Low area-overhead decoder and programmability



http://www.asimovinstitute.org/neural-network-zoo/
*Figure shows partial list

C-BRIC

Contribution 1: *Domain-specific ISA*



- **Compute**
- **Data Move (Intra-Tile): over shared bus, shared memory**
- **Control: loops in workloads with MVMU reuse (weight reuse)**
- **Data Move (Inter-Tile): over network on chip.**

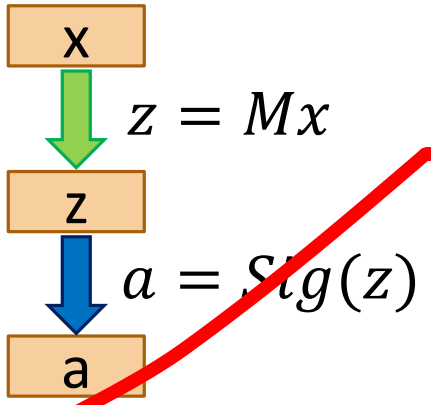
➢ **Wide instructions**

- **Large register address space** to support memristive crossbars (very-wide SIMD)
- ***vector width*** keeps instruction memory low in spatial architecture

C-BRIC

Contribution 1: *Domain-specific ISA*

1 Layer MLP

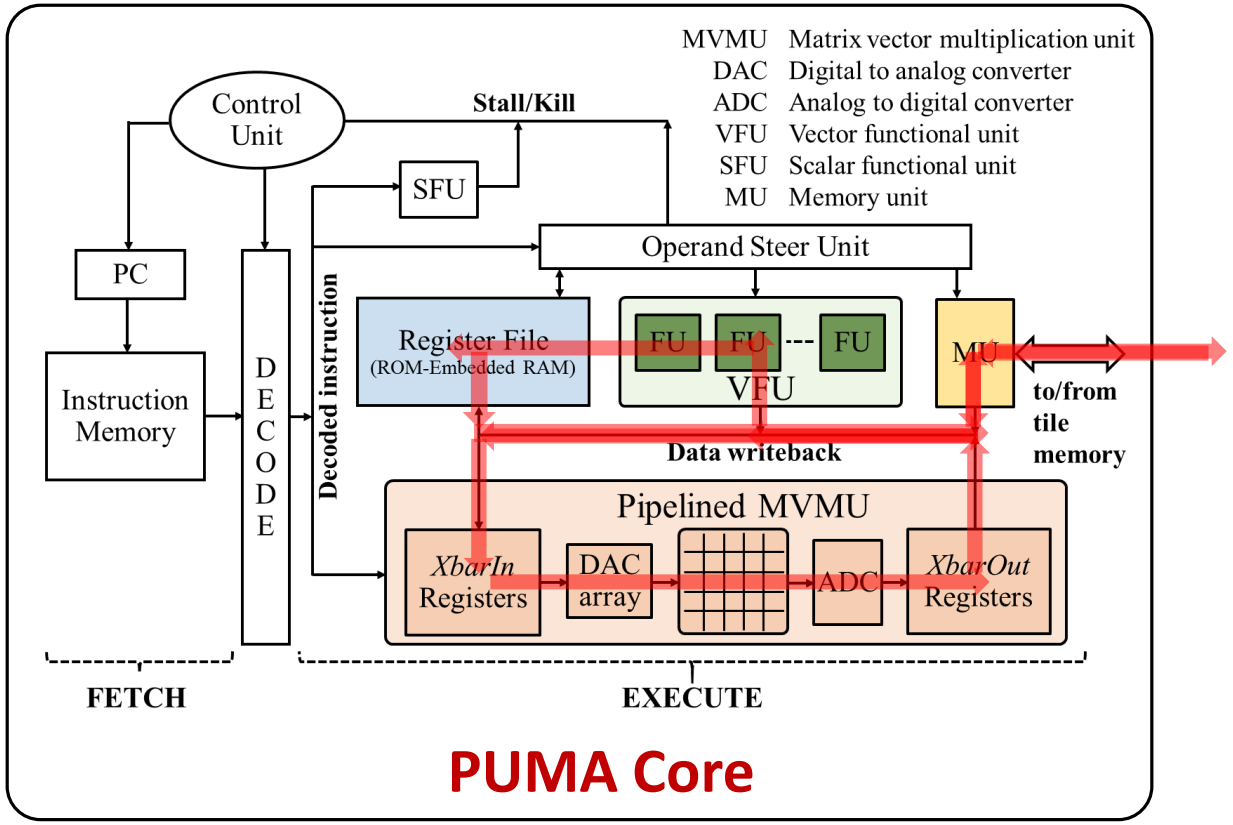


MLP maps on one MVMU

```
LD $m_i1, x, n
MVM 0b01
ALU_SIG $1, $m_o1, n
ST $1, a
```

MLP maps on multiple MVMUs/Cores

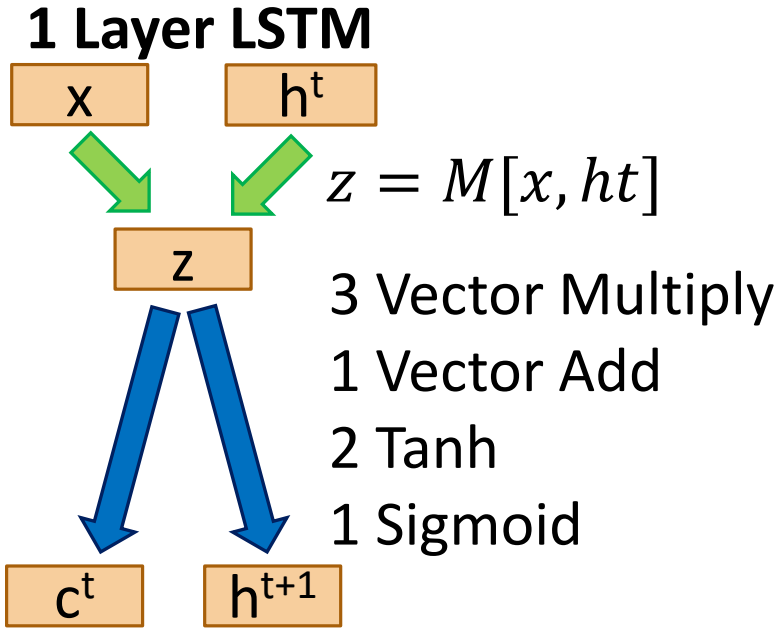
```
MVM 0b01

// for all partial sums
LD xxx // bring partial sums from other cores
ALU_ADD xxx // accumulate partial sums

ALU_SIG $1, $m_o1, n
```

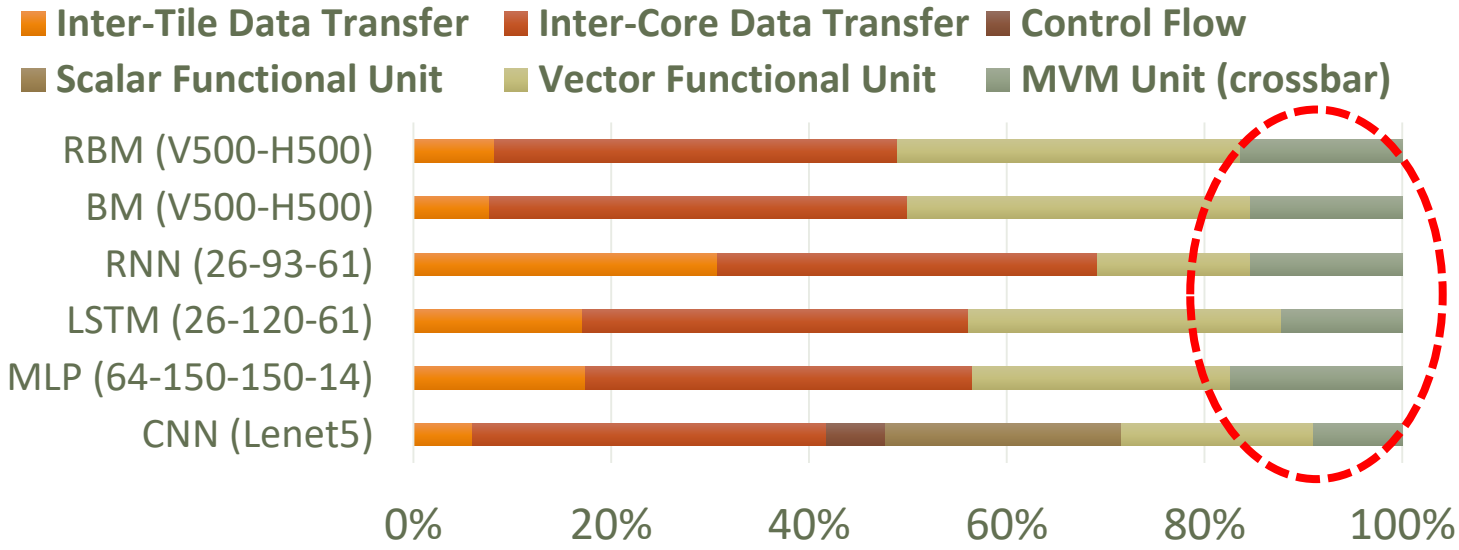


PUMA Core

Larger matrices require more data movements to compute effective MVM.

Contribution 2: *Hybrid Core*

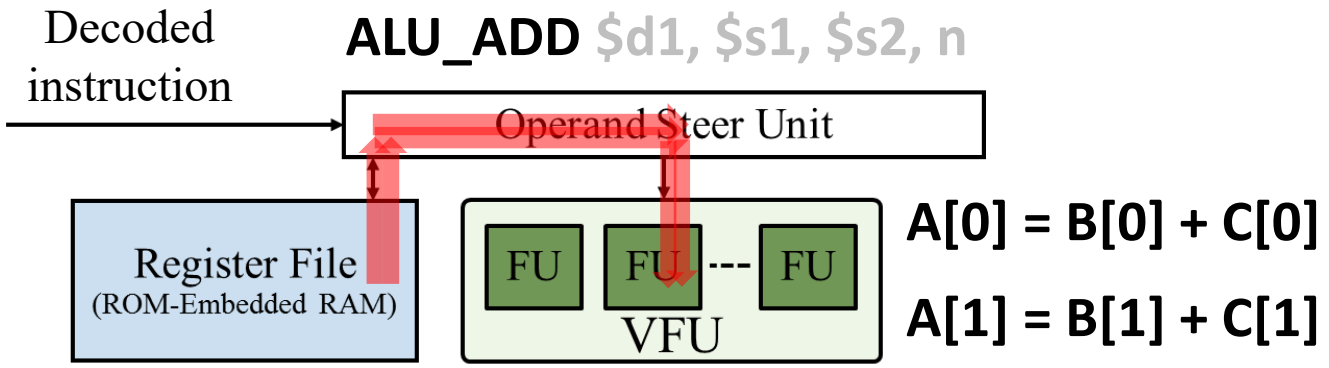




- **MVMU alone cannot execute ML apps.**
  - Vector operations: require previous writes to crossbar
  - Transcendental operations: MVMU (even with writes) can do add/multiply only.
- **Propose core designed with hybrid memristive and CMOS technologies.**
  - Hybrid analog-digital core: general-purpose
  - Co-locating execution units reduces data movements (**near-memory**)

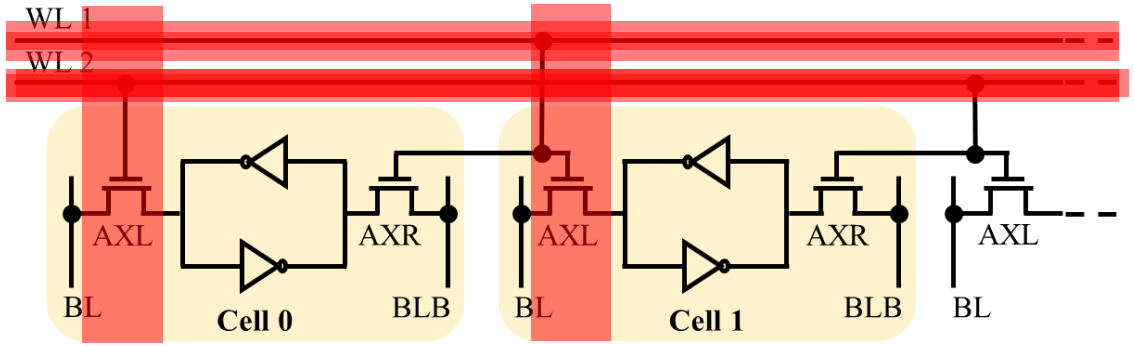
Contribution 2: *Hybrid Core*

- **(déjà vu) Preserving storage density** - Memristive and CMOS technology have orders of magnitude disparity in area-efficiency.



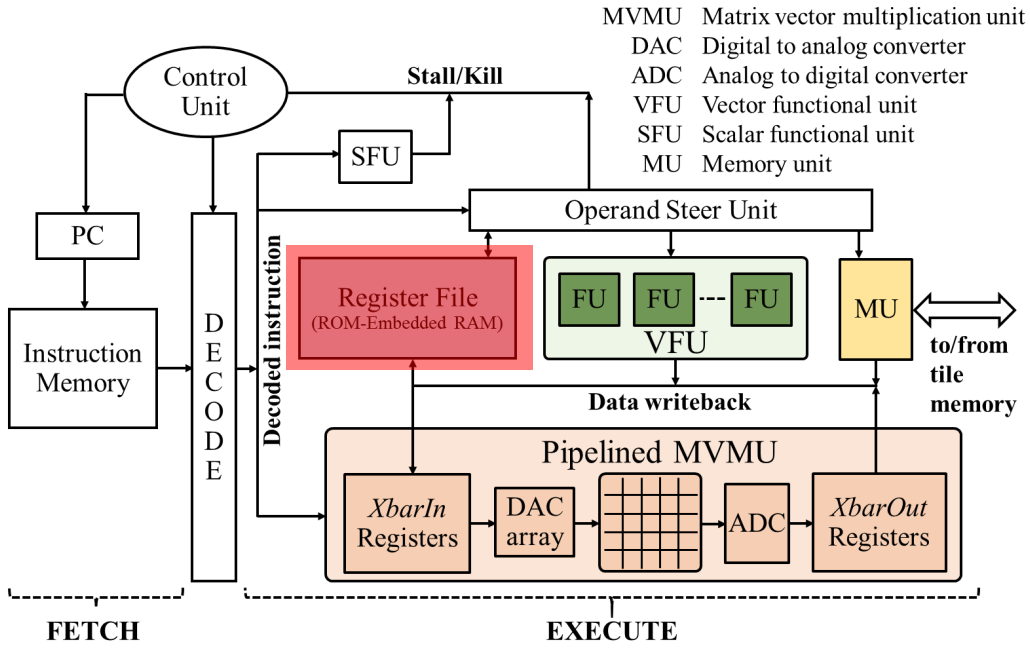
- Temporal SIMD
  - **Low width VFU saves area**
  - Wide vector instruction have low fetch/decode/instruction storage overhead
- VFU needs to be fast enough to not be a bottleneck for MVMU
  - ML workloads: all other operations (vector, transcendental) are preceded by MVM

Contribution 2: *Hybrid Core*



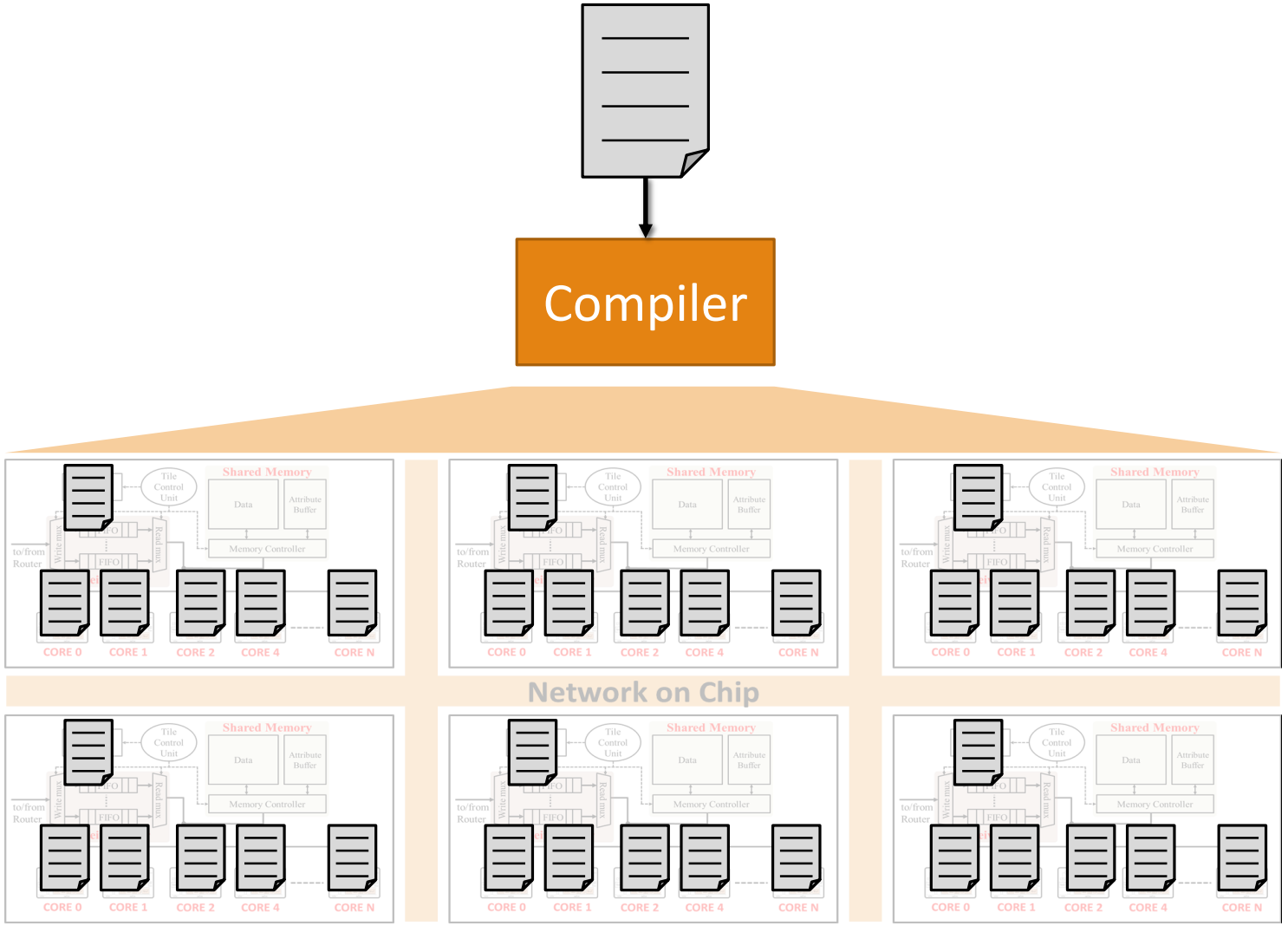
- Add an extra worldline to every row (~1% array area overhead, sense-amps unchanged)
- RAM: Activate both wordlines - WL1 and WL2
- ROM: storage depends on AXL's connection to wordline WL1
  - Write 0s (both wordline active)
  - Selectively write 1s (WL1 active)
  - Read data (typical RAM read)
  - Restore RAM data (ROM read is destructive)

- **Register file buffers operands and becomes an on-demand transcendental function unit (sig, tanh, etc.)**
- Multiple cycles (2-3), needs to be fast enough for MVMU.



Contribution 3: *Compiler optimizations*

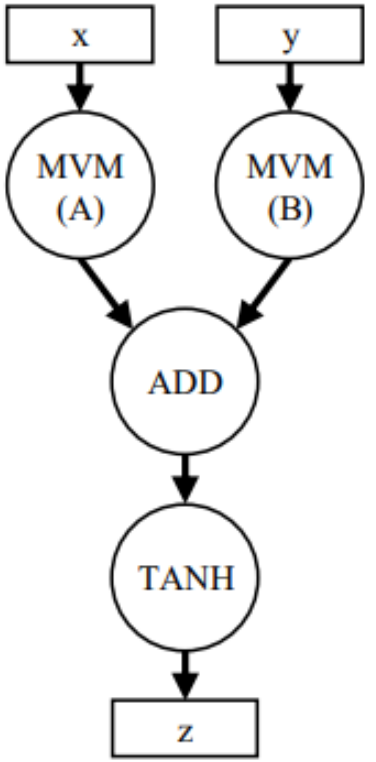
- **Programmability: traditional programming models not suitable for spatial architecture.**
- Writing different code for each core/tile is not scalable as applications grow.
- Optimizations:
  - Graph partitioning: optimizing for data movements
  - MVM Coalescing: extract available ILP



Contribution 3: *Compiler optimizations*

Simple Code Example

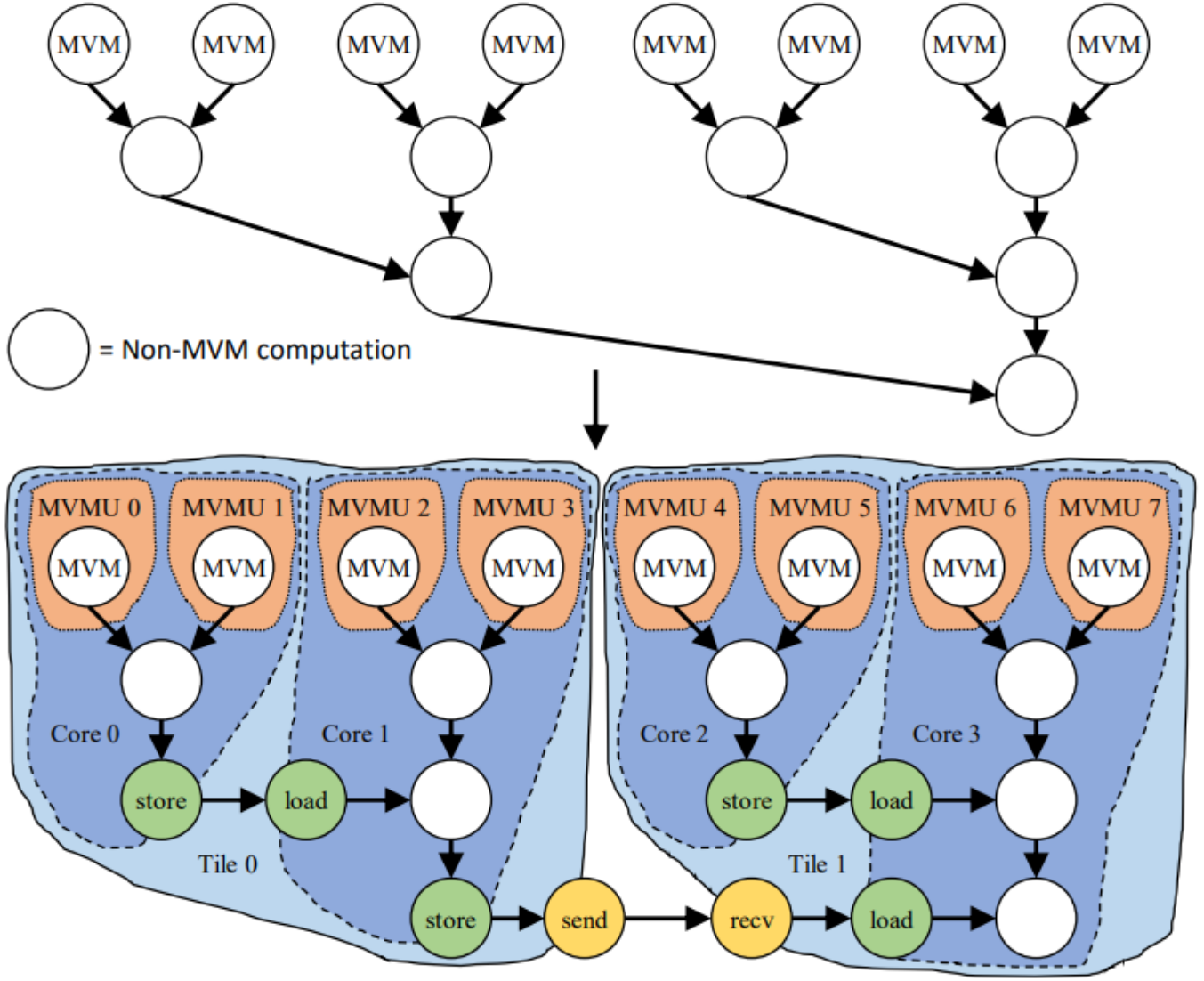
```
01  Model m = Model::create("example");
02
03  InVector x = InVector::create(m, M, "x");
04  InVector y = InVector::create(m, M, "y");
05  OutVector z = OutVector::create(m, N, "z");
06
07  ConstMatrix A = ConstMatrix::create(m, M, N, "A");
08  ConstMatrix B = ConstMatrix::create(m, M, N, "B");
09
10  z = tanh(A*x + B*y);
11
12  g.compile();
```



C-BRIC

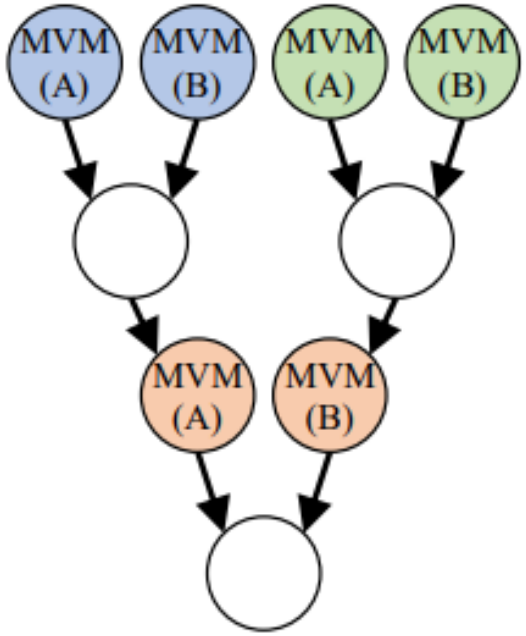
Contribution 3: *Compiler optimizations*

Graph Partitioning Example

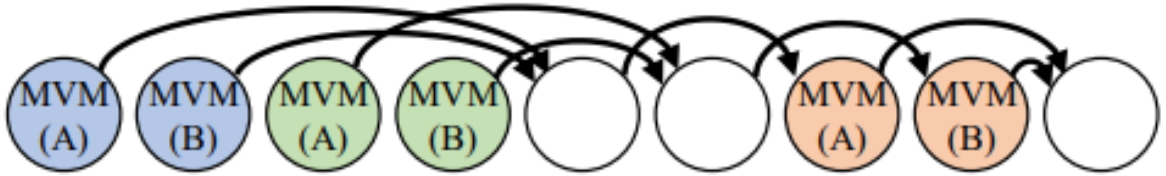


Contribution 3: *Compiler optimizations*

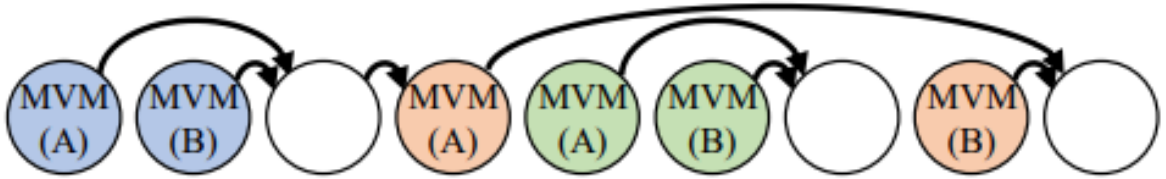
Instruction Scheduling Example



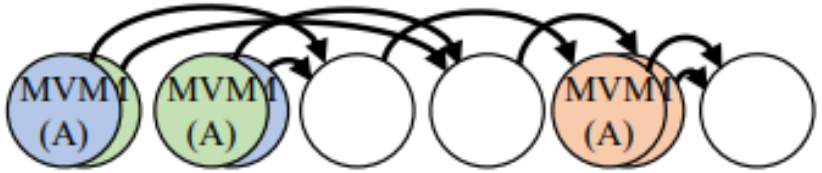
(a) Sub-graph to be scheduled on a single core with two crossbars storing matrices A and B



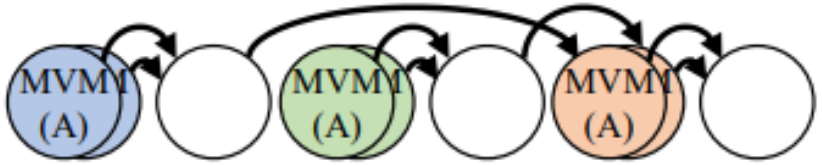
(b) Naïve linearization (high pressure)



(c) Reverse postorder linearization (low pressure)



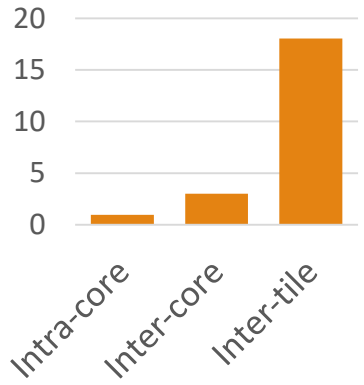
(d) Poor coalescing creates high pressure

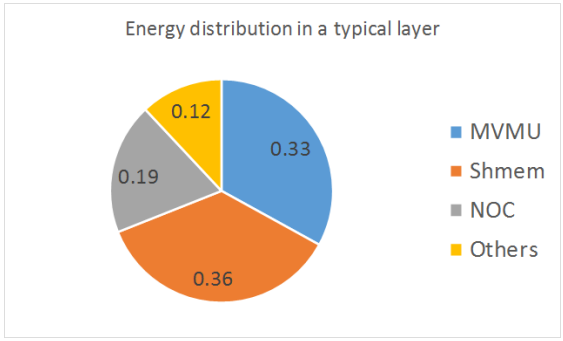


(e) Good coalescing enables low-pressure

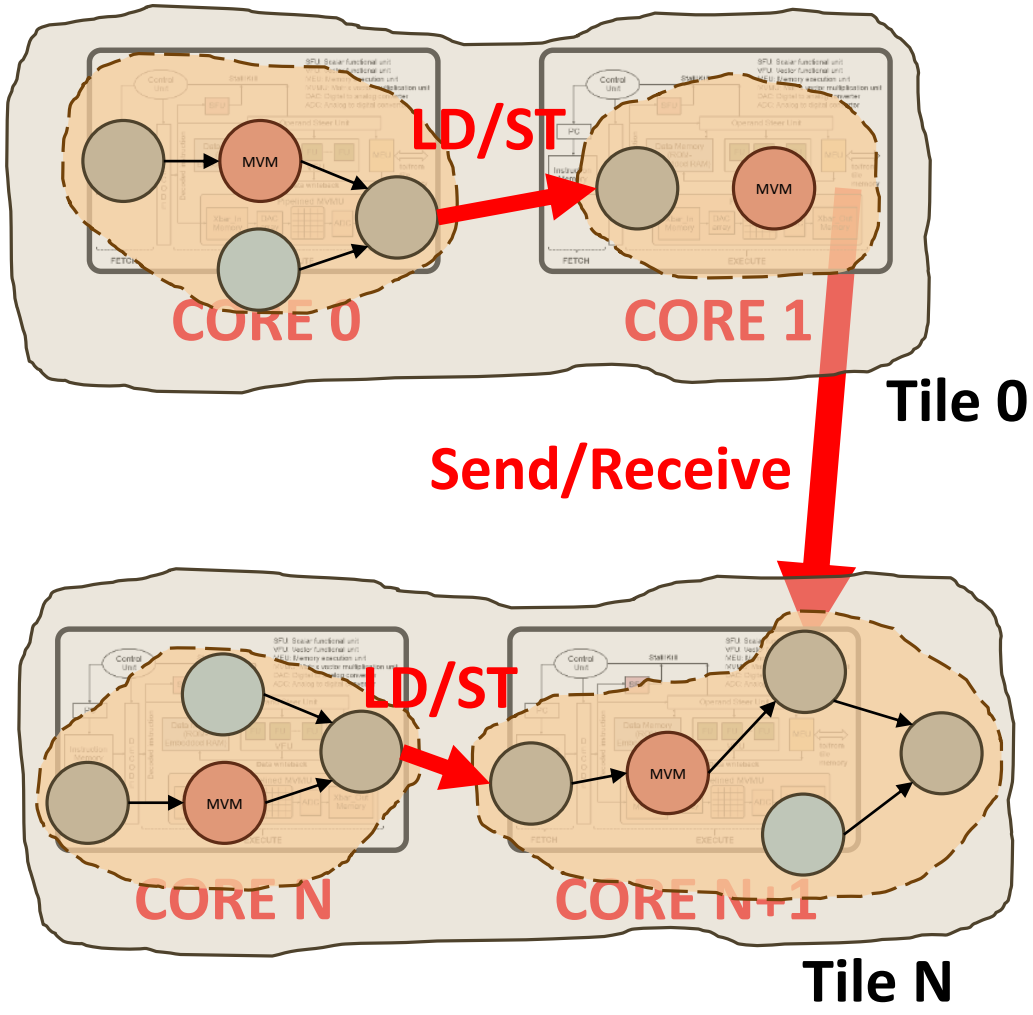
Contribution 3: *Compiler optimizations*

- Data movement is analogous to memory hierarchy in traditional systems, highest contributor in energy profile.





- **First level of graph partitioning, optimizing for locality and inter-core communication (LD/ST).**
- **Second level of graph partitioning, optimizing for locality and inter-tile communication (Send/Receive).**

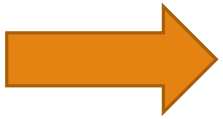


Contribution 3: *Compiler optimizations*

- MVM instructions consume high latency
- **MVM Coalescing**
  - Schedule the required LDs, coalesce the MVMs
- **Area-efficiency, alternatively score-boarding is too expensive.**

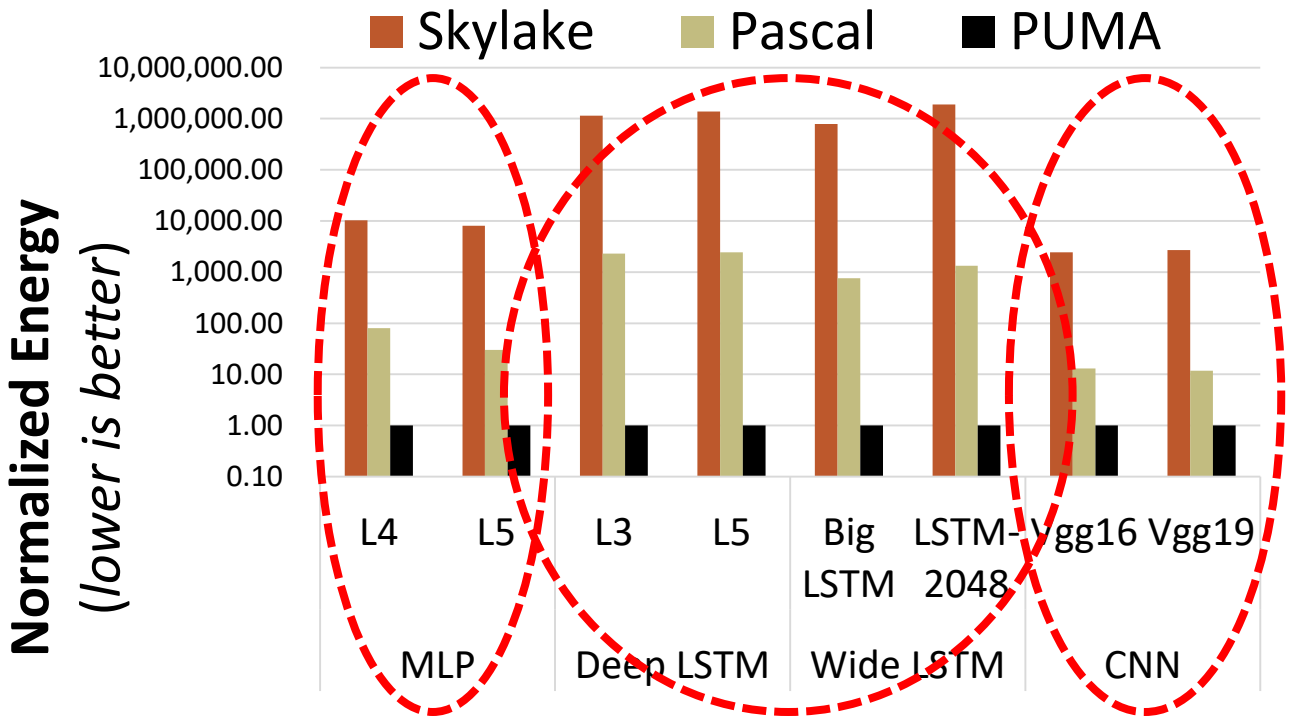
Instruction	Cycles
LD/ST	>24
ALU	128 - 384
MVM	2304
Send/Receive	variable

```
LD $mi1, x, n
MVM 0b01
ALU_SIG $1, $mo1, n
ST $1, a
LD $mi2, x, n
MVM 0b10
ALU_SIG $2, $mo2, n
ST $2, a
```



```
LD $mi1, x, n
LD $mi2, x, n
MVM 0b11
ALU_SIG $1, $mo1, n
ST $1, a
~~MVM 0b10~~
ALU_SIG $2, $mo2, n
ST $2, a
```

Results: Inference Energy (Batch size = 1)



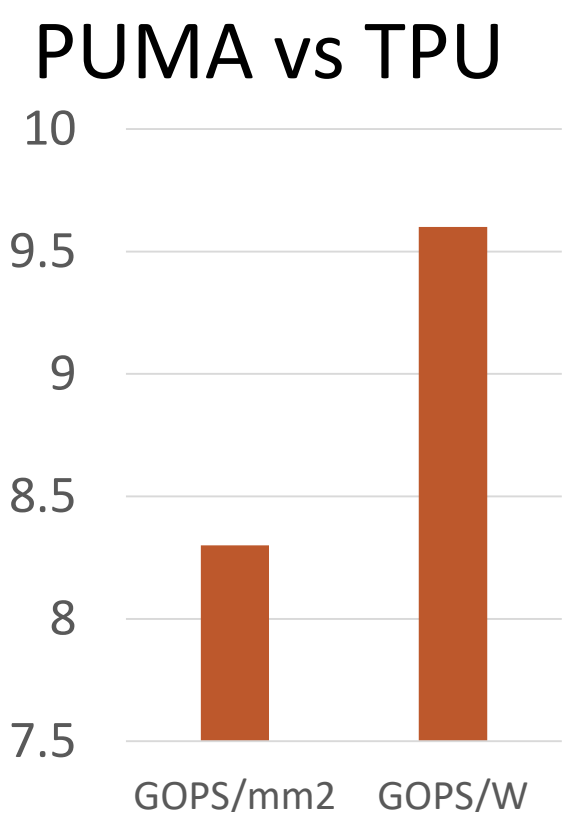
- CNNs show upto 13.0× reduction (least).
  - **High weight reuse, even at batch-size 1.**
- MLPs show upto 80.1× reduction.
  - **No weight reuse, small models.**
- LSTMs show upto 2446× reduction.
  - **Little Weight reuse, large models (billions of parameters).**

- **PUMA achieves massive reduction in inference energy across all platforms for all benchmarks.**

C-BRIC

ASIC Comparison

- **Google TPU comparison**



- **Cost of generality, programmability**
  - 20.7% lower *power efficiency* and 29.2% lower *area efficiency* than ISAAC.

- **Modest cost increase over past memristive accelerators, significantly improves applicability of memristive crossbars.**

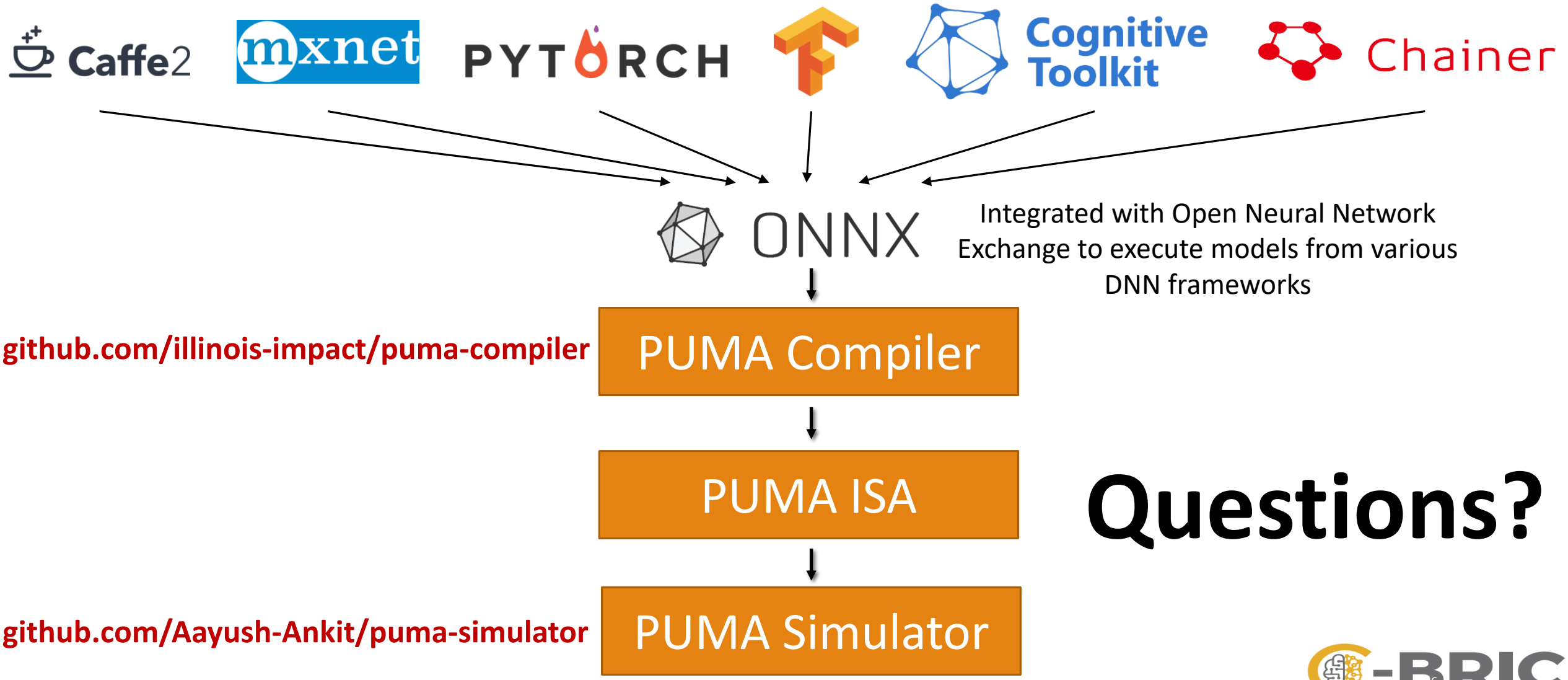
Conclusion

- **First general-purpose and ISA-programmable accelerator built with hybrid CMOS-Memristive technology.**
  - Architecture-ISA-Compiler Codesign
  - Many other optimizations (input shuffling, spatial pipelining, inter-core synchronization, register allocation etc.) and detailed results in paper.

Goals / Constraints	General-purpose	Programmability
Storage Density	Domain Specific ISA Hybrid Core	Domain Specific ISA Compiler Optimizations
Data Movement	Hybrid Core	Compiler Optimizations

C-BRIC

Toolchain for Memristive Accelerators



Questions?