



Introduction to Compute-in-Memory

Laura Fick, Dave Fick

Mythic

Outline of Talk

- Introduction to Compute-In-Memory
 - What memory?
 - When is it useful?
 - When is it not useful?
- Case Studies:
 - SSD compute-in-memory for list intersection
 - Mythic's analog compute-in-memory for GPS correlation
 - Mythic's analog compute-in-memory for neural networks
- Conclusion

Compute-in-Memory in a Nutshell

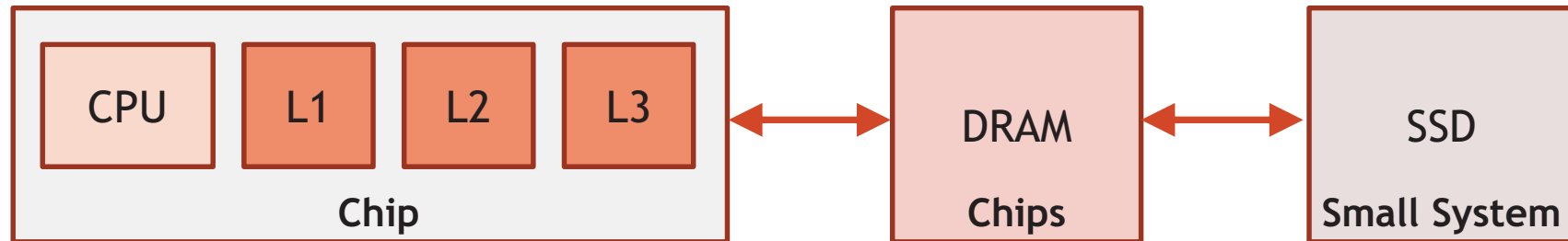
- Compute-in-memory is a broad range of techniques with a single underlying principal:

Sometimes it is better to create additional compute at the memory than move data from the memory to the main compute.*

** = faster and/or more efficient*

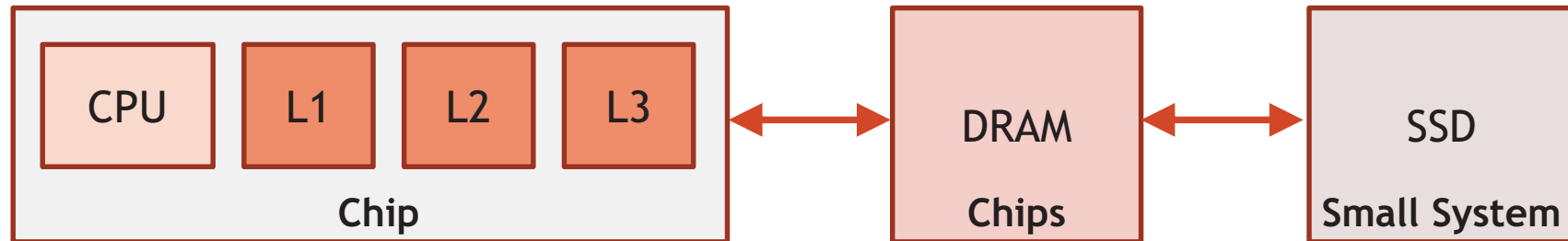
- *This is particularly important for today's data processing applications, and post-Moore design.*

“In Memory” is Relative



- “Compute-in-Memory” is relatively to an existing system that has both compute and memory.
- Let’s consider a standard memory system
 - There are progressively larger caches (including DRAM)
 - Memory closer to CPU is faster & smaller
 - Main memory is in SSD or HDD

“In Memory” is Relative



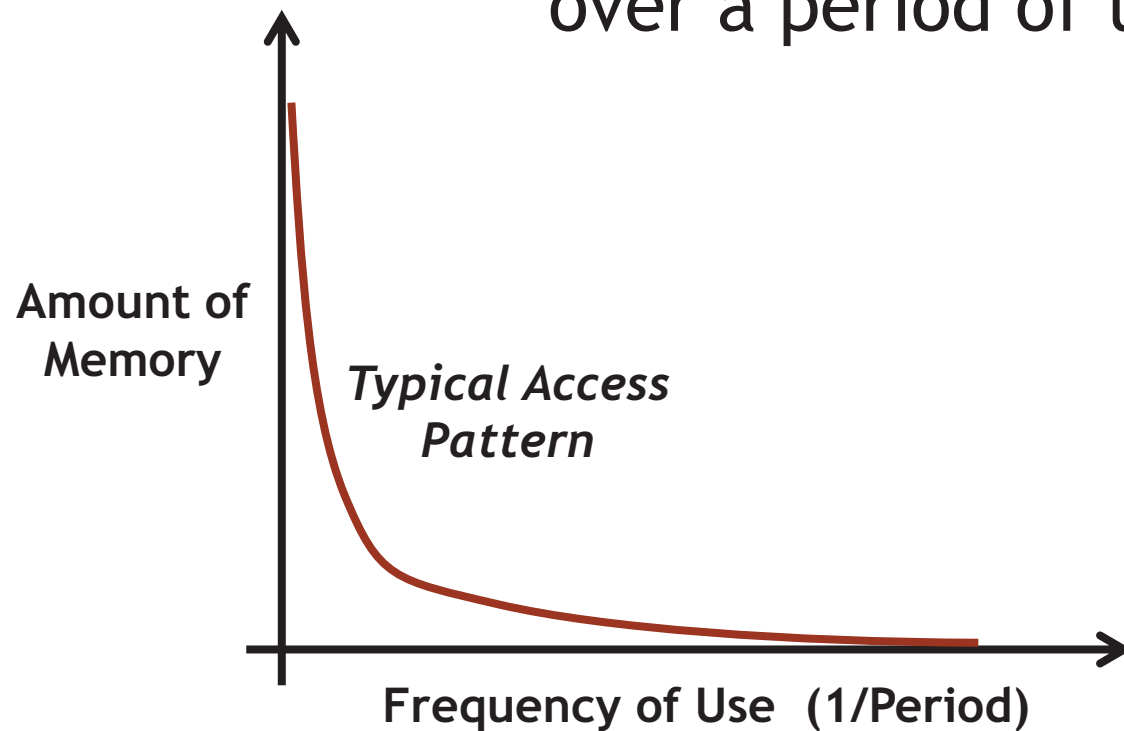
- Typically, CPU works on L1 Cache
- “Compute-in-Memory” could mean...
 - Compute at L2/L3
 - Compute in the DRAM chips
 - Compute in the SSD
 - Compute in an accelerator that contains memory

Memory Systems are Built for Data Access Patterns

- The existing memory structure has built-in assumptions:
 - Temporal Locality
 - If at one point a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future.
 - Spatial Locality
 - If a particular storage location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future.
 - Probability, not Certainty
 - We do not know exactly what data will be needed next

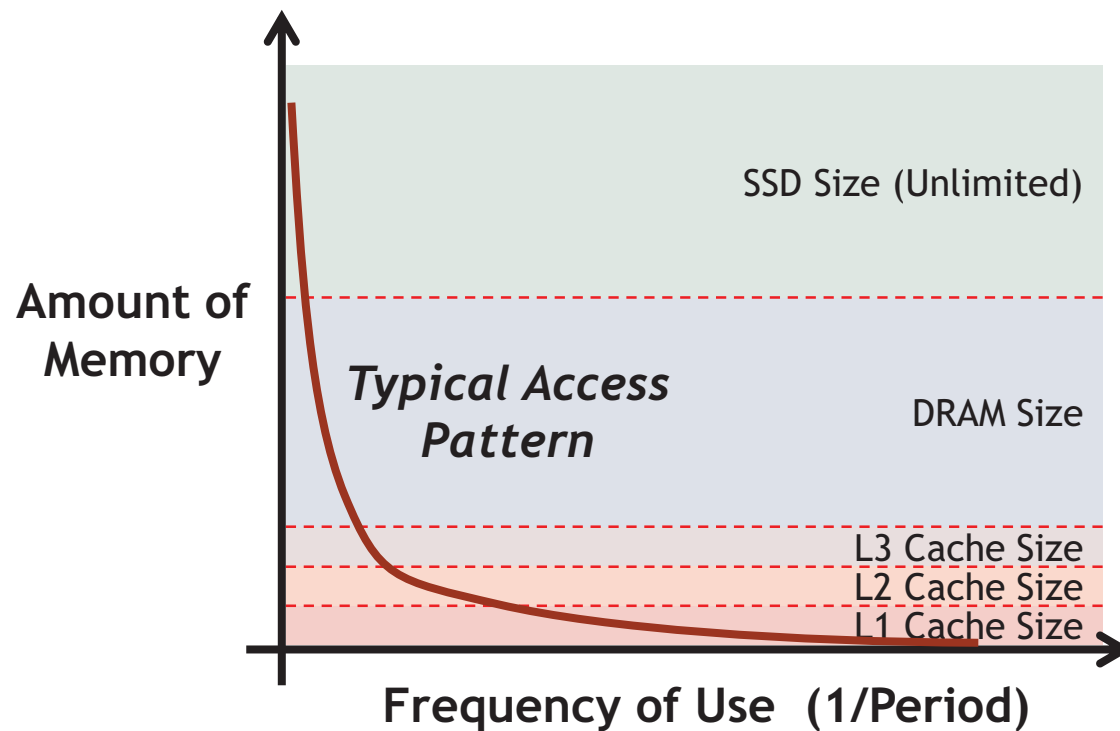
Data Access Patterns: Analysis Via “Working Set”

Working set: the amount of memory needed by the application over a period of time



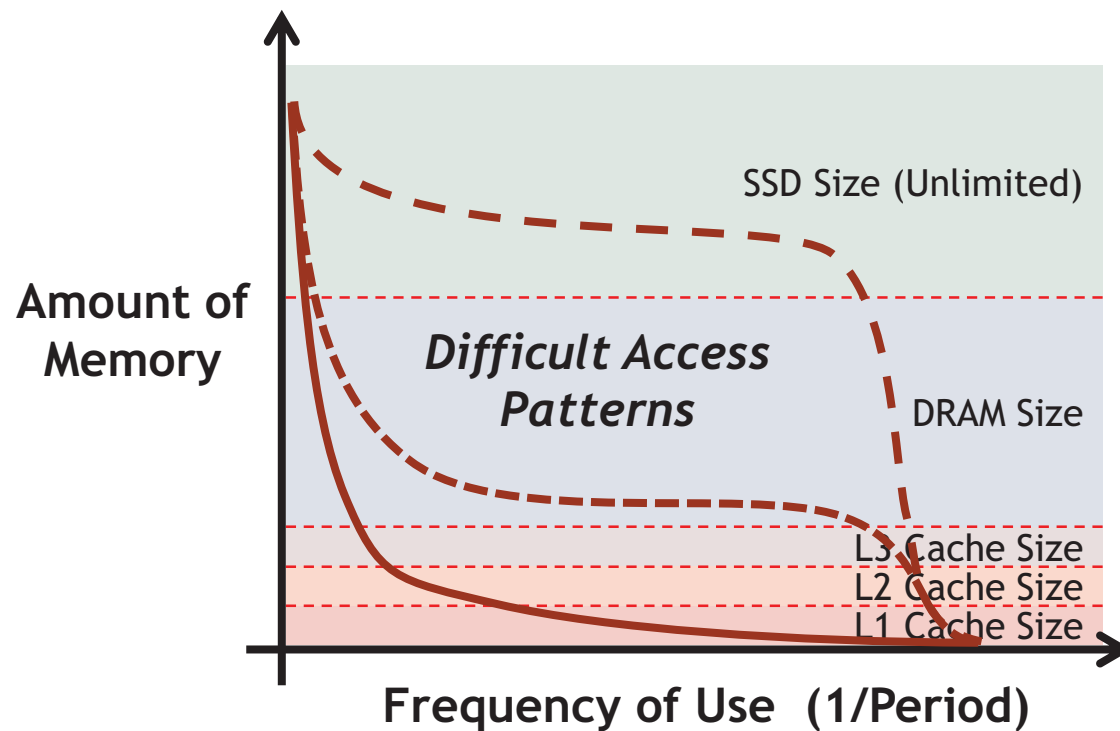
Caches Capture Small Working Sets

The cache system captures larger and larger working sets. The mostly infrequently used data is stored in system storage (SSD)



Compute-in-Memory Captures More Difficult Access Patterns

Some applications have access patterns that do not “play nice” with the traditional memory hierarchy



Compute-in-Memory systems can target these applications.

Other Reasons for Compute-in-Memory

- **Deterministic Data Patterns**
 - In some cases, we know the exact data access pattern to be performed, so hierarchical memory systems do not provide a benefit and are inefficient.
- **ASIC Capabilities Further Minimize Data Movement**
 - In other cases, we can build in ASIC capabilities that take advantage of known data patterns.
- **Analog Computation**
 - In extreme cases, analog computation can be added to achieve most minimal data movement.

Compute-in-Memory is Not Always Useful

- General Purpose Computing
 - You need an application to take advantage of.
- Low Application Importance
 - If this application is not >90% of the system time or power, then you will not be able to get a 10x improvement.
- Small Working Sets
 - Compute-in-memory often requires relatively large working sets to make sense.
 - Applications that fit in L1 cache are hard to improve.



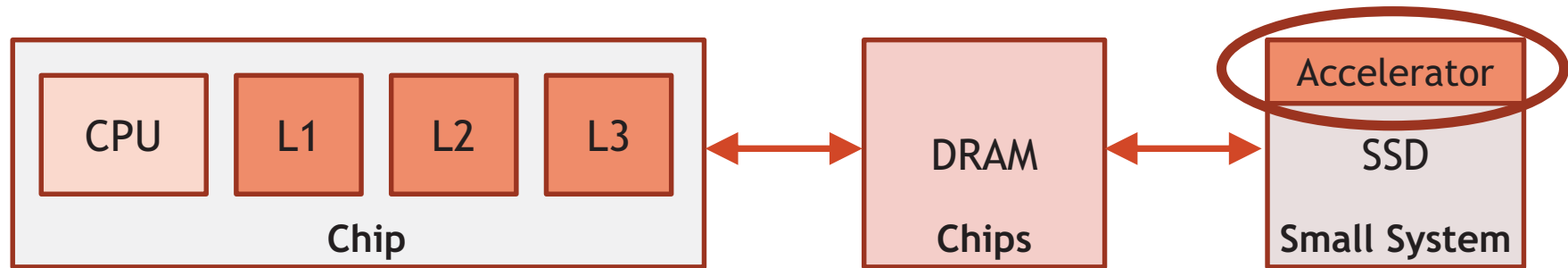
SSD In-Storage Computing for List Intersection

*J. Wang, D. Park, Y.S. Kee, Y.
Papakonstantinou, S. Swanson*

UC San Diego, Samsung

Data Management on New Hardware, 2016

Remember: “In Memory” is Relative



- Typically, CPU works on L1 Cache
- “Compute-in-Memory” could mean...
 - Compute at L2/L3
 - Compute in the DRAM chips
 - **Compute in the SSD!**
 - Compute in an accelerator that contains memory

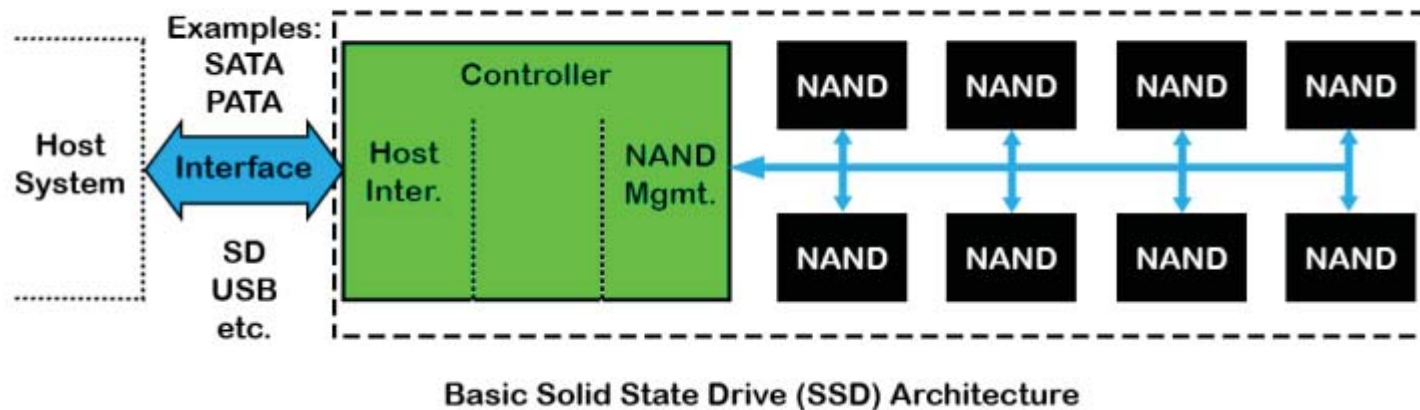
Case Study: List Intersection

- List intersection finds the common elements in a set of data
- Intersection is prominent in search engines and analytics queries (**lots of data**)
- Speed of list intersection is dependent on many parameters: algorithm, list length and correlation

Are the data usage requirements amenable to compute-in-memory?

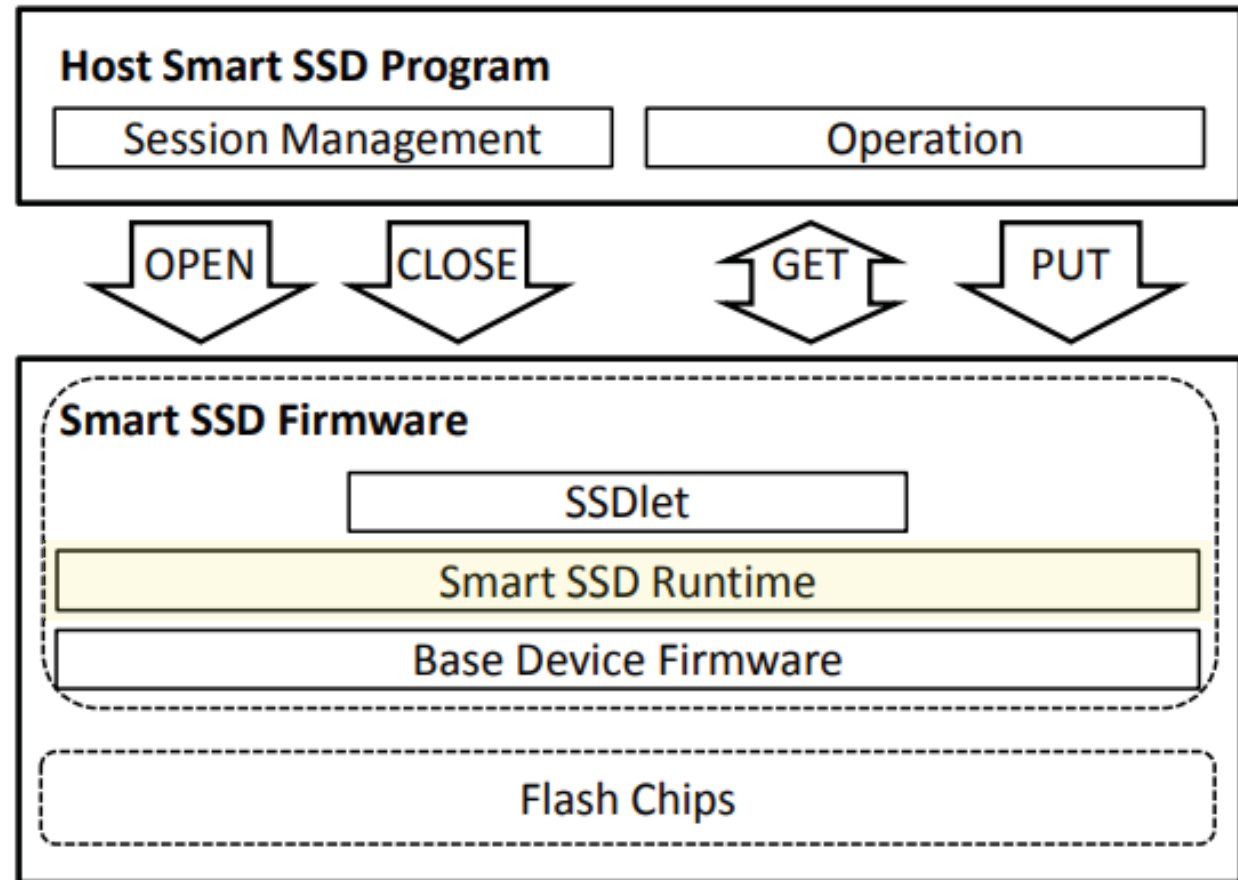
SSD Architecture

- Modern SSDs are typically architected with a higher (2-4x) internal bandwidth than the host interface bandwidth
- Using the SSD controller as an off-load engine to execute some programs



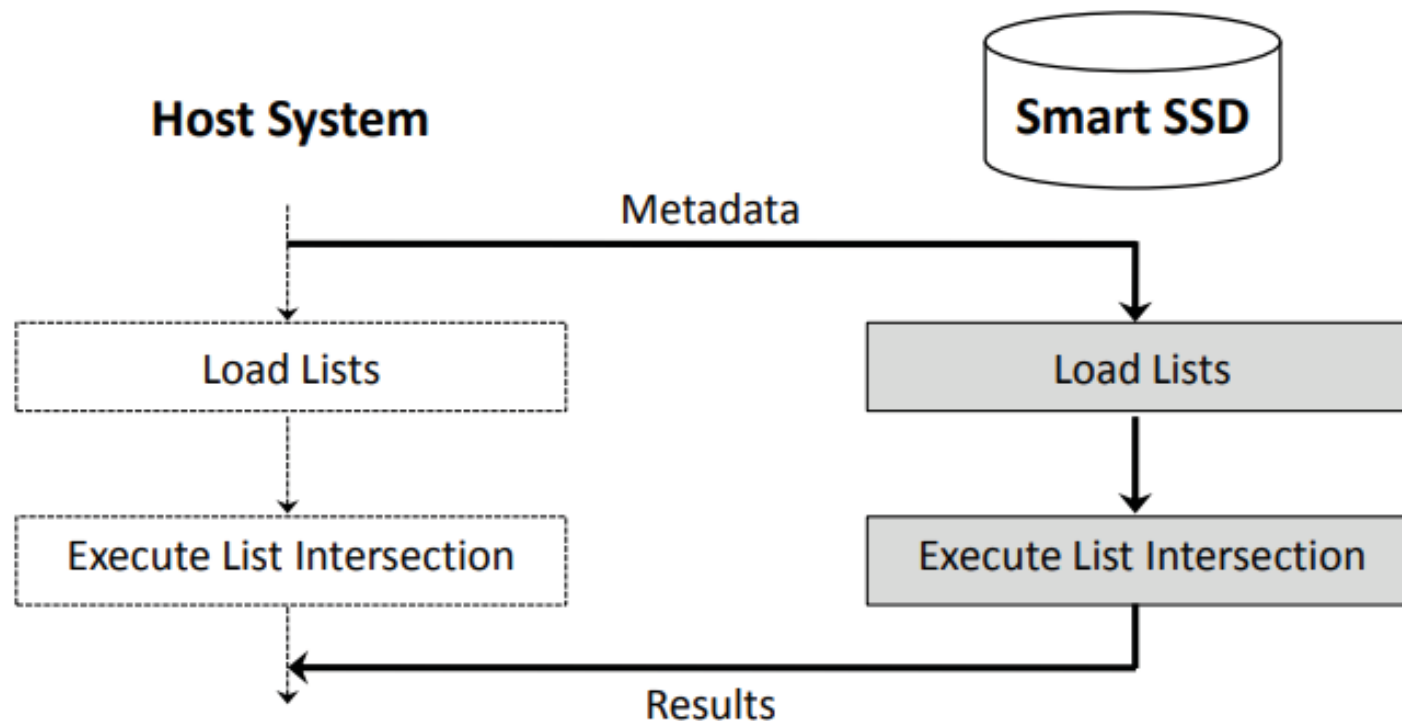
Smart SSD Architecture

New firmware
layer for
running these
algorithms



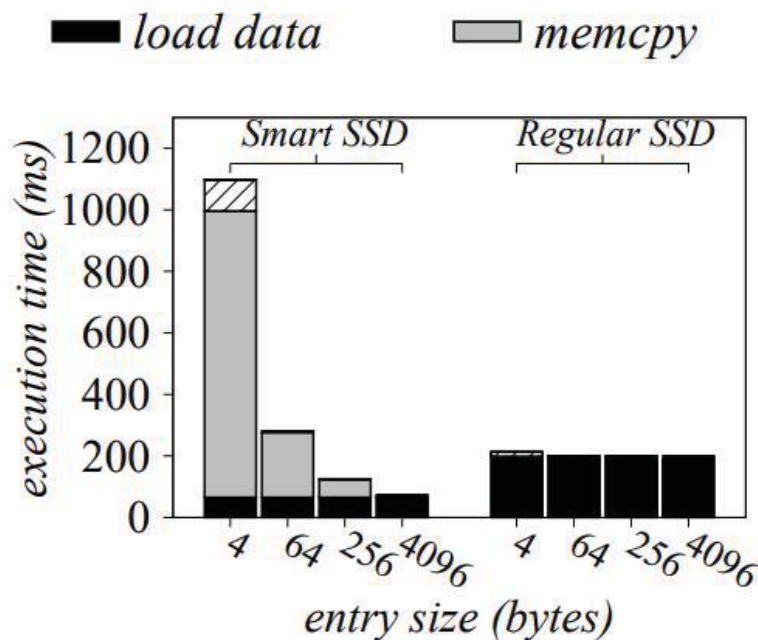
Smart SSD Host Interaction

- Host system sends only metadata to the Smart SSD (addresses, lengths of lists)
- Load lists are stored on the Smart SSD

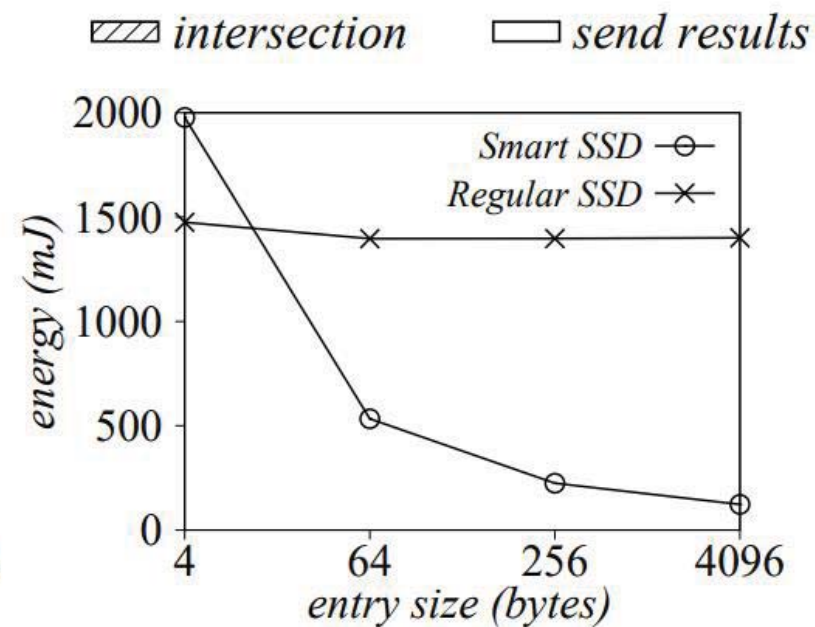


Effects on Smart SSD Performance: Entry Size

- Larger entry size improves both execution time and energy compared to regular SSD implementation
- Above 256 entry size, load data is dominant



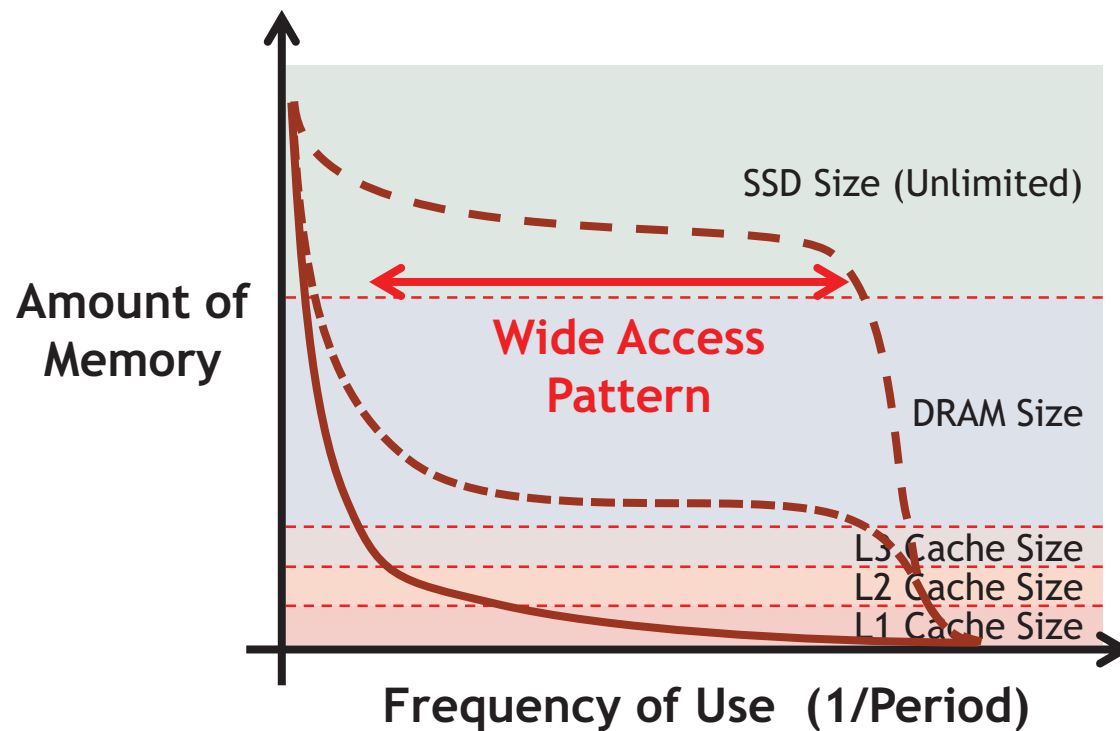
(a) execution time (ms)



(b) energy consumption (mJ)

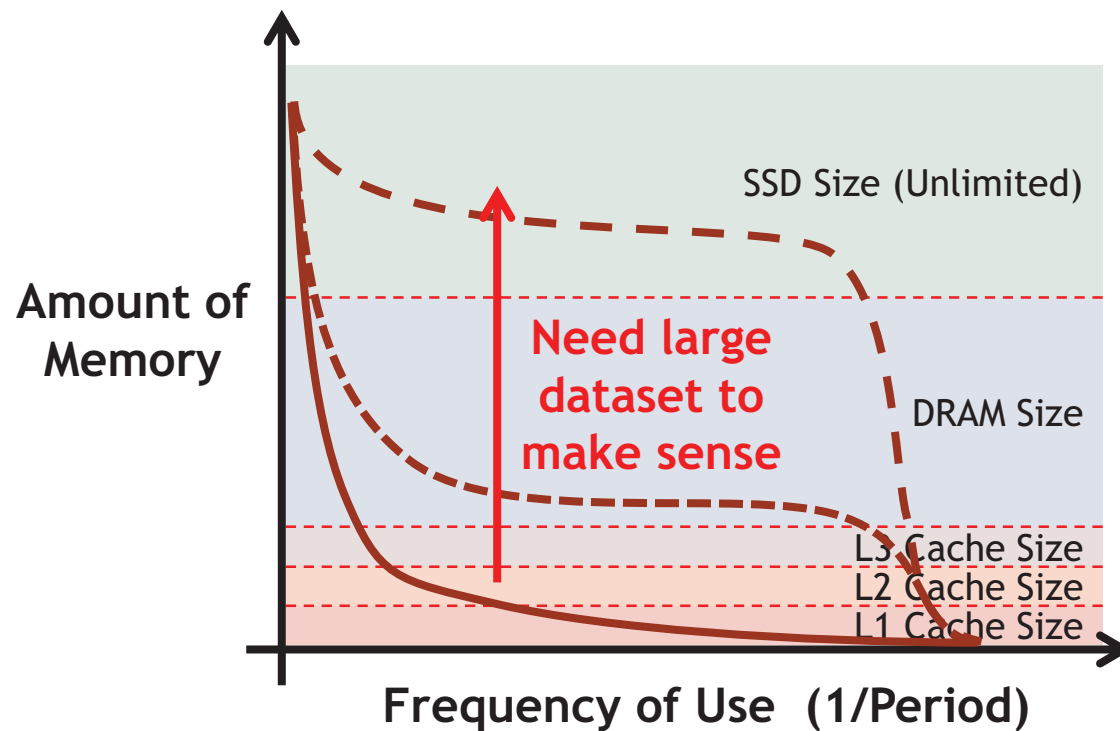
Remember: Uniform workload is better!

- Compute-in-memory is better for uniform data workload applications (wide access pattern)
- Consistently accessing a large set of data



However: Larger amount of data is better too

- Workload looks similar, except for the *amount* of data
- Compute-in-memory benefits applications that have memory loads as the primary bottleneck



A 36.8 2b-TOPS/W Self Calibrating GPS Accelerator Implemented Using Analog Calculation in 65nm LP CMOS

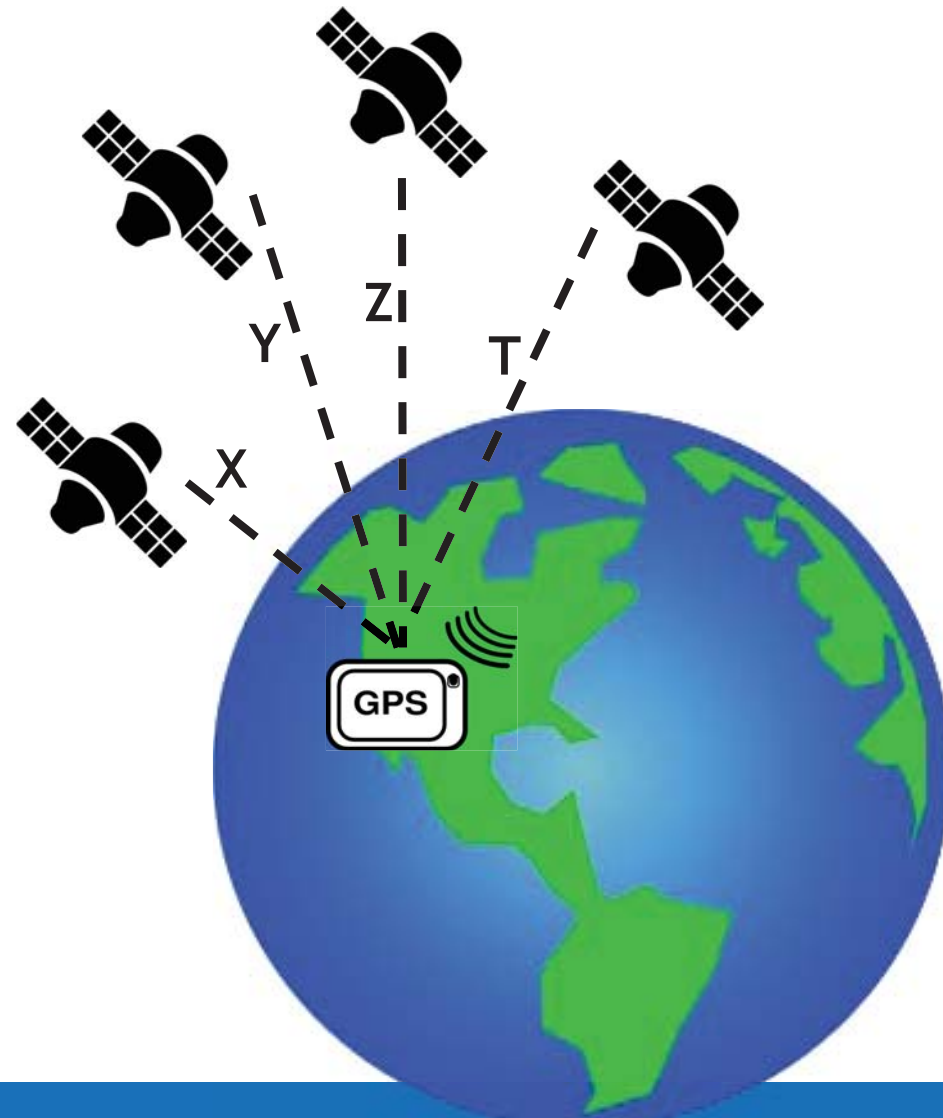
**Skylar Skrzyniarz^{1,2}, Laura Fick², Jinal Shah², Yejoong Kim²,
Dennis Sylvester², David Blaauw², David Fick¹, Michael B. Henry¹**

¹Mythic (fka Isocline), Austin, TX

²University of Michigan, Ann Arbor, MI

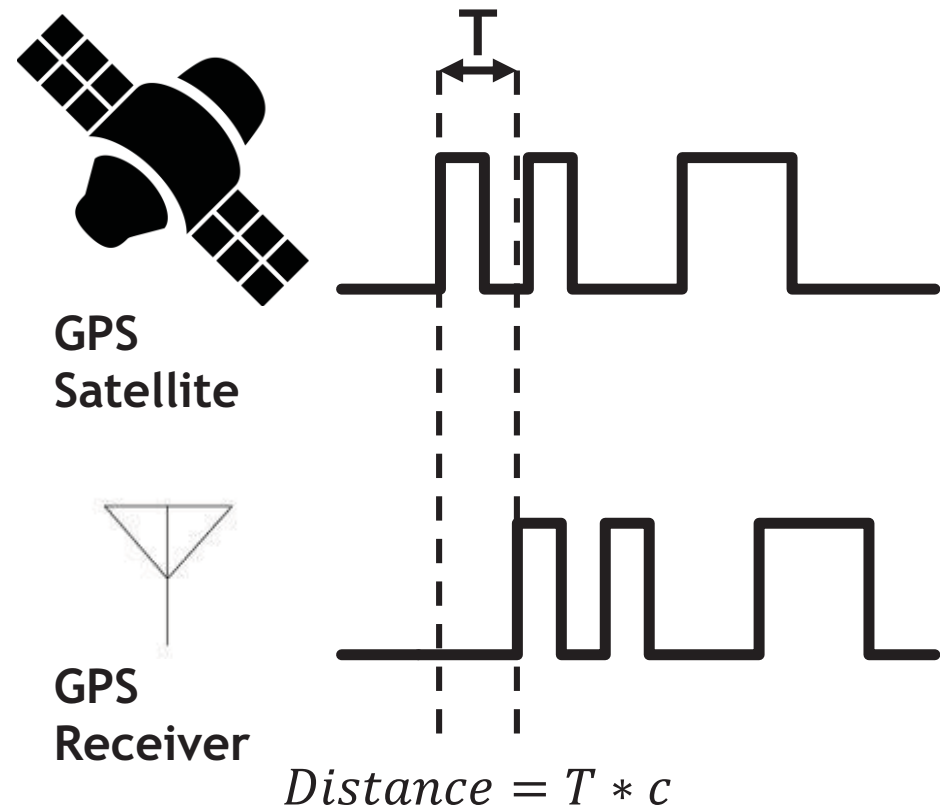
GPS is a 4-Dimensional Search

- Need to find X, Y, Z, and T
- Calculated by acquiring time offset from 4+ satellites
- GPS is a CDMA signal
 - Received below thermal noise floor



Executing GPS Acquisition: The Problem

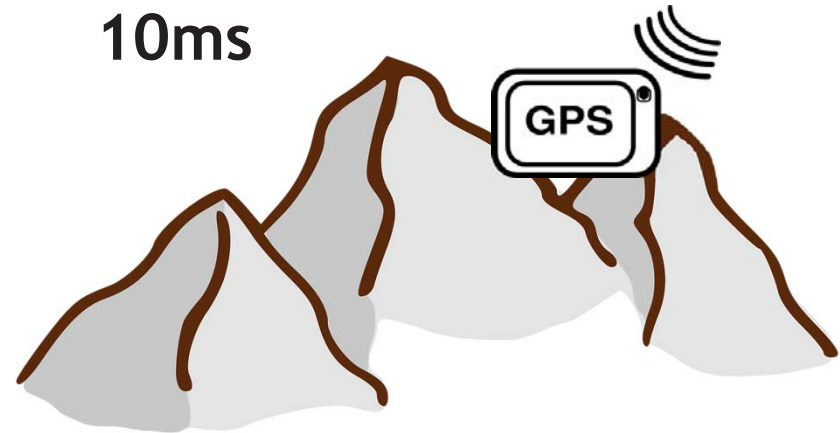
- Time-domain search
- Timing offset (T) is calculated
 - Done through correlation
 - 1000's of 2-bit vector multiply
 - 1000's of results to accumulate
- Pattern is very long
 - Reject noise
 - Increase gain



Executing GPS Acquisition: The Problem

- Requires many operations
 - $(10-350) \times 10^9$ for civilian
 - 35×10^{12} for military
- **Energy intensive**
 - 19-665 mJ per satellite
- **Time consuming**
 - 10-350 ms per satellite

strong signal $\approx 19\text{mJ}$,
10ms



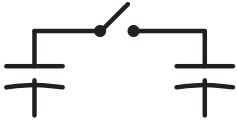

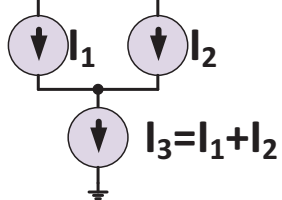
weak signal $\approx 665\text{mJ}$,
350ms



Analog Calculation as the Solution



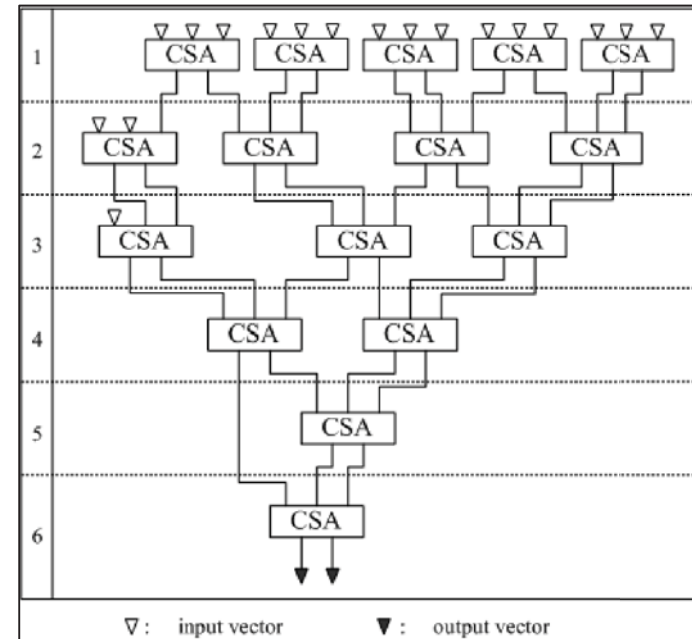
Analog Calculation

	<u>Charge</u> 	<u>Conductance</u> 	<u>Current</u> 
<u>Example Uses</u>	<ul style="list-style-type: none"> • Multiply 	<ul style="list-style-type: none"> • Absolute-value-of-difference 	<ul style="list-style-type: none"> • Summation
<u>Advantages</u>	<ul style="list-style-type: none"> • Energy efficient • Result is a voltage 	<ul style="list-style-type: none"> • Energy efficient • Area efficient 	<ul style="list-style-type: none"> • Can use many inputs • Process tolerant
<u>Disadvantages</u>	<ul style="list-style-type: none"> • Area • Variation 	<ul style="list-style-type: none"> • Special device (flash) • Variation 	<ul style="list-style-type: none"> • Lower energy efficiency

Kramer, A.H

Analog Calculation: Adding 4096 2-Bit Numbers

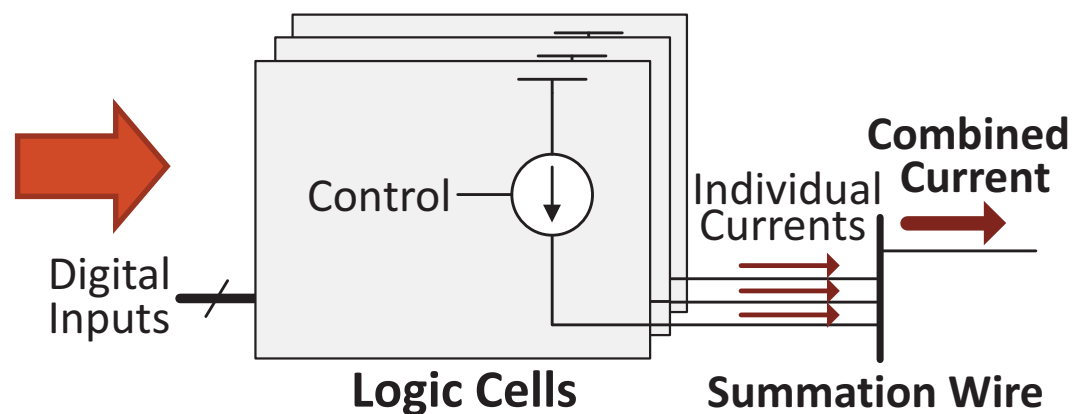
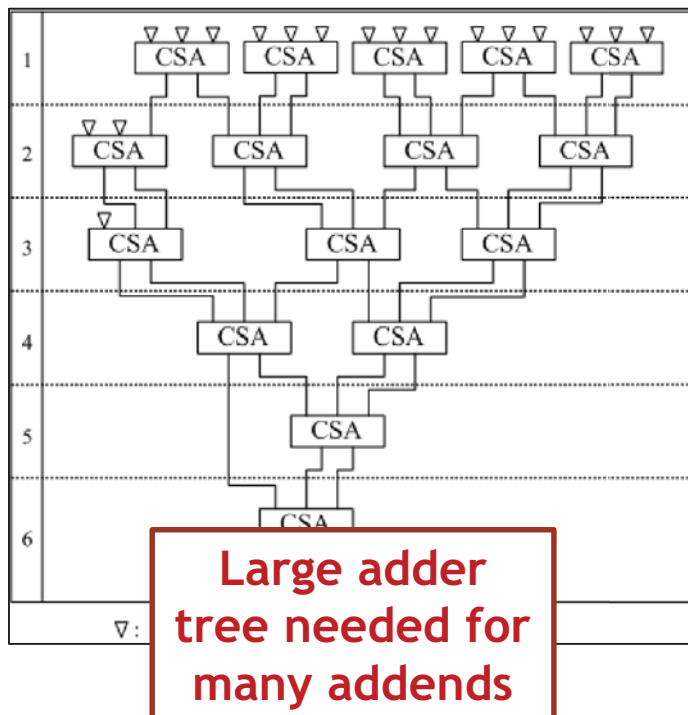
- Digital:
 - 8168 full adders
 - 15 stage tree



**Large adder
tree needed for
many addends**

Analog Calculation: Adding 4096 2-Bit Numbers

- Digital:
 - 8168 full adders
 - 15 stage tree
- Analog:
 - Current-mode summation

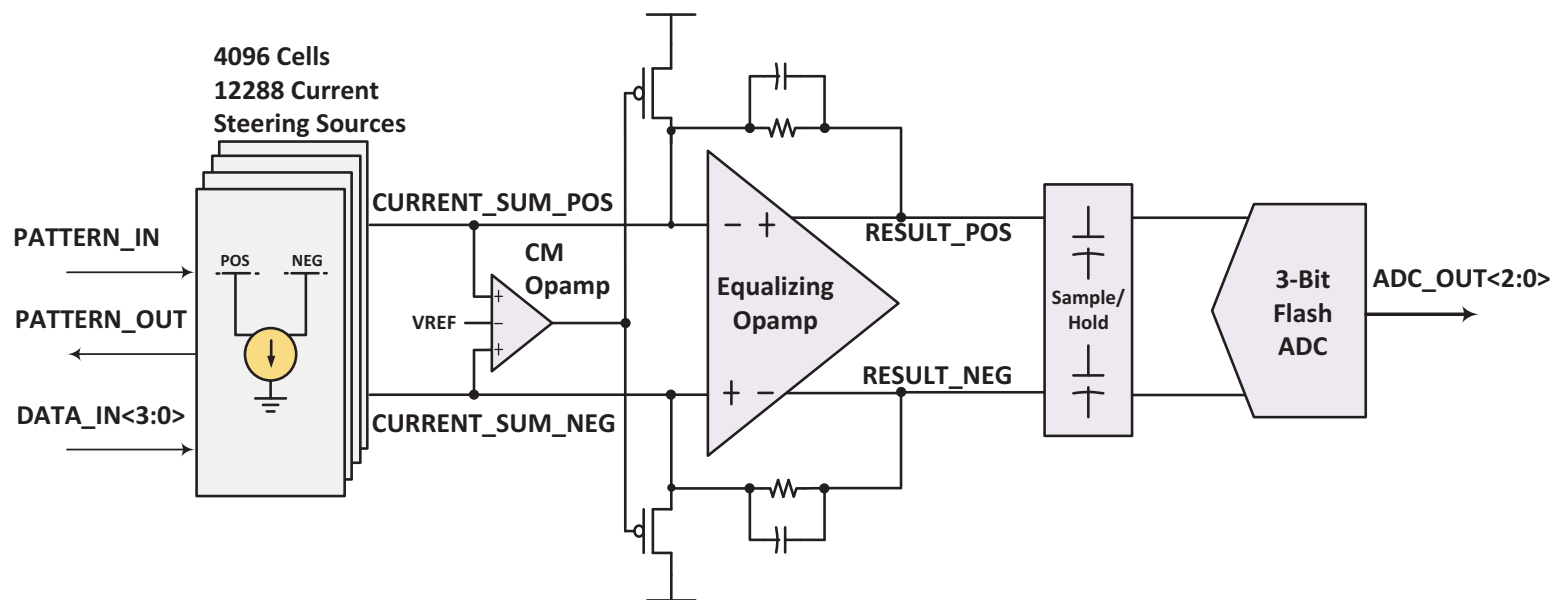
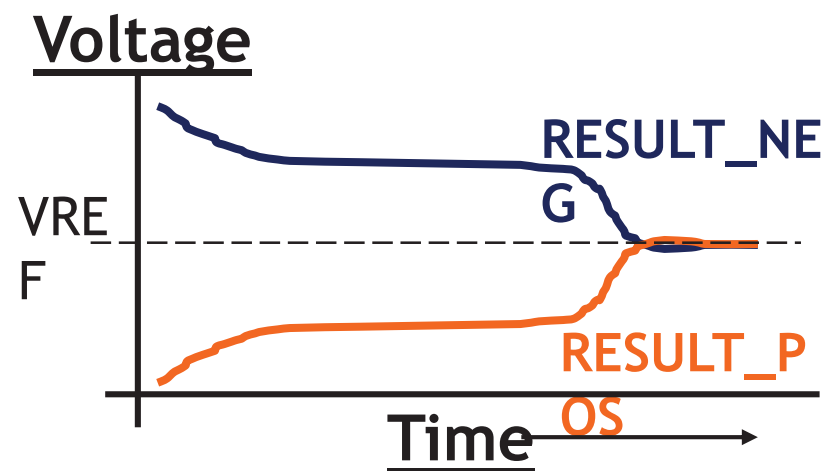
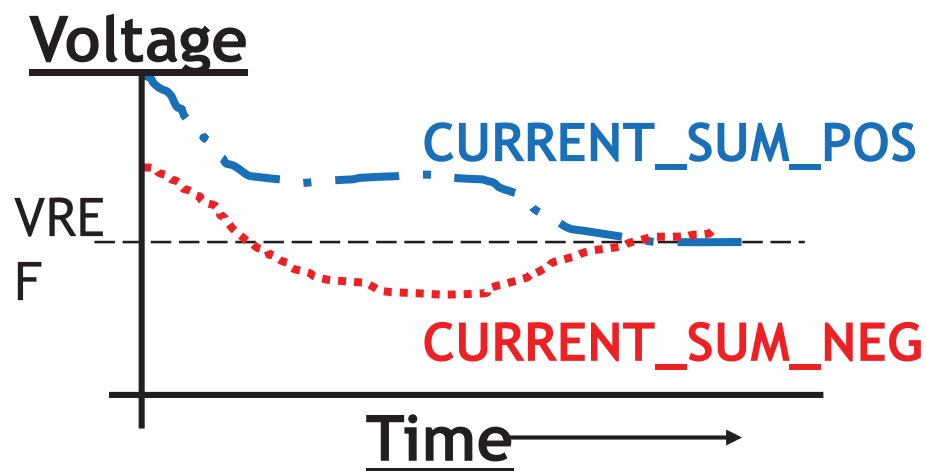


Single Wire Summation
Regardless of how
many addends

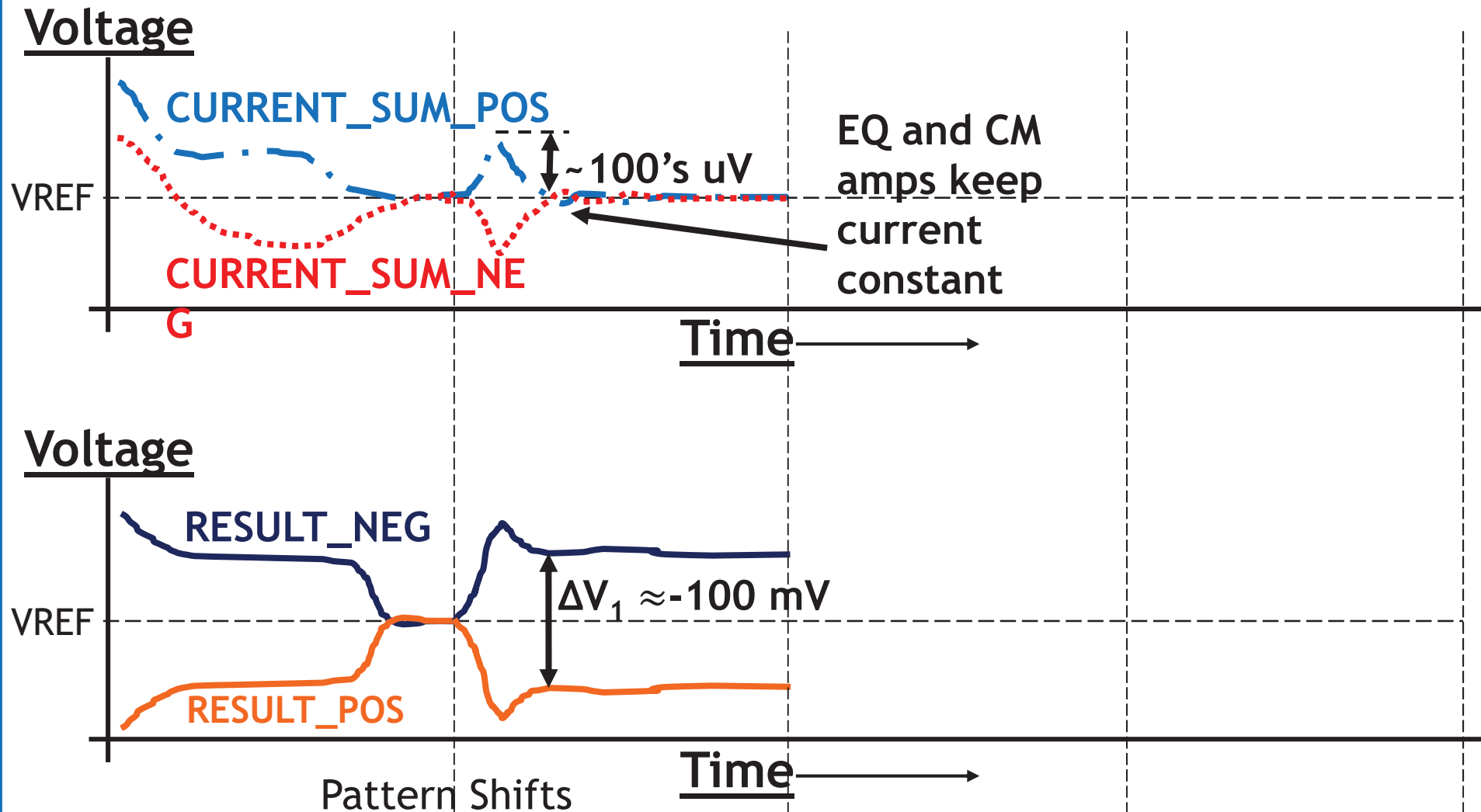
Analog Calculation: Application Advantages

- Main challenge of analog calculation is added noise
 - Noise ↓ as input terms ↑
 - GPS has many input terms
- Correlation result has narrow dynamic range
 - Zero mean result
 - 48.5-to-51.5% vs. 0-to-100%
 - High resolution
 - 0.5% change in current per LSB
 - Circuits heavily optimized for this small range

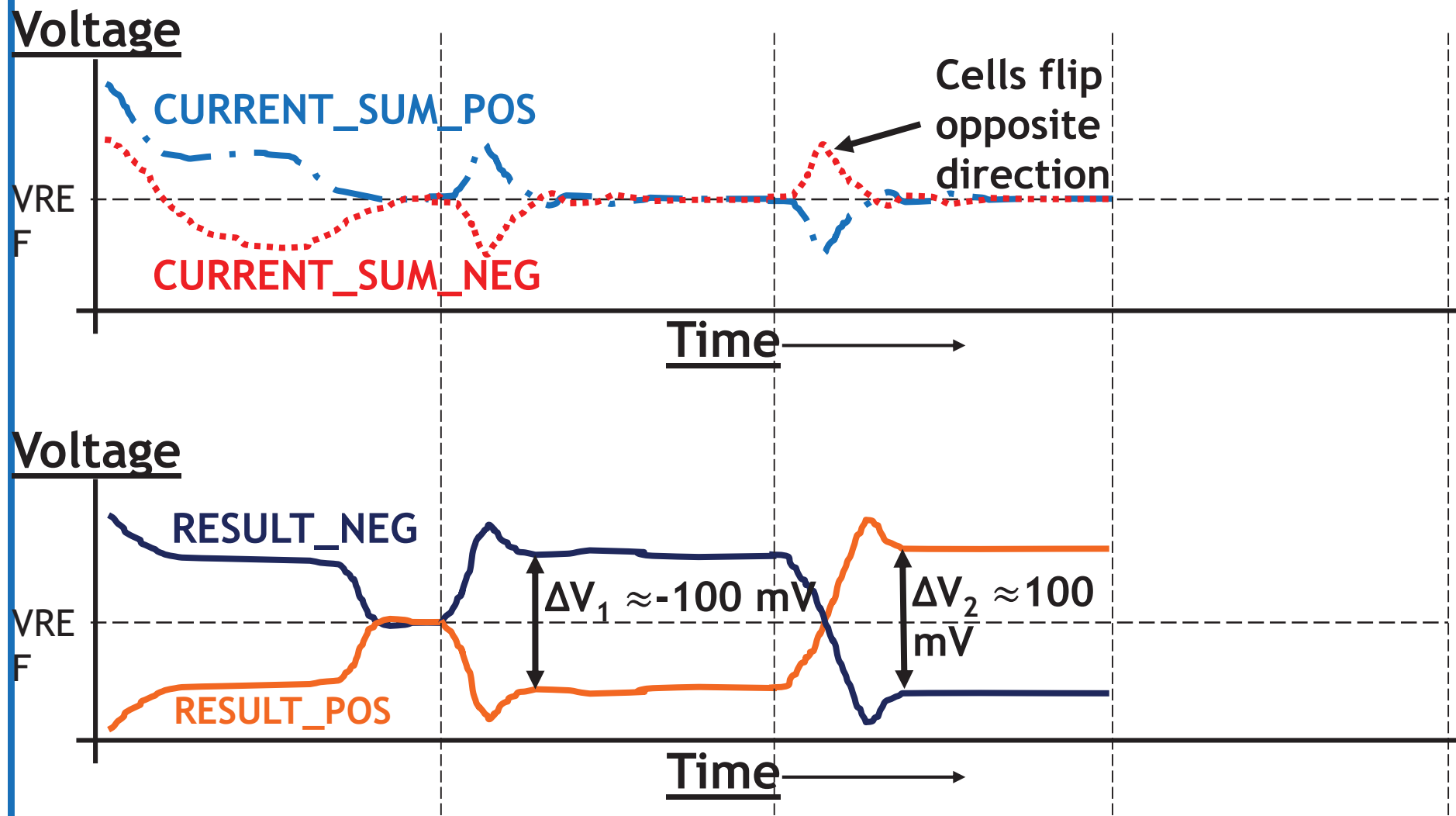
Performing a Calculation: Initialization



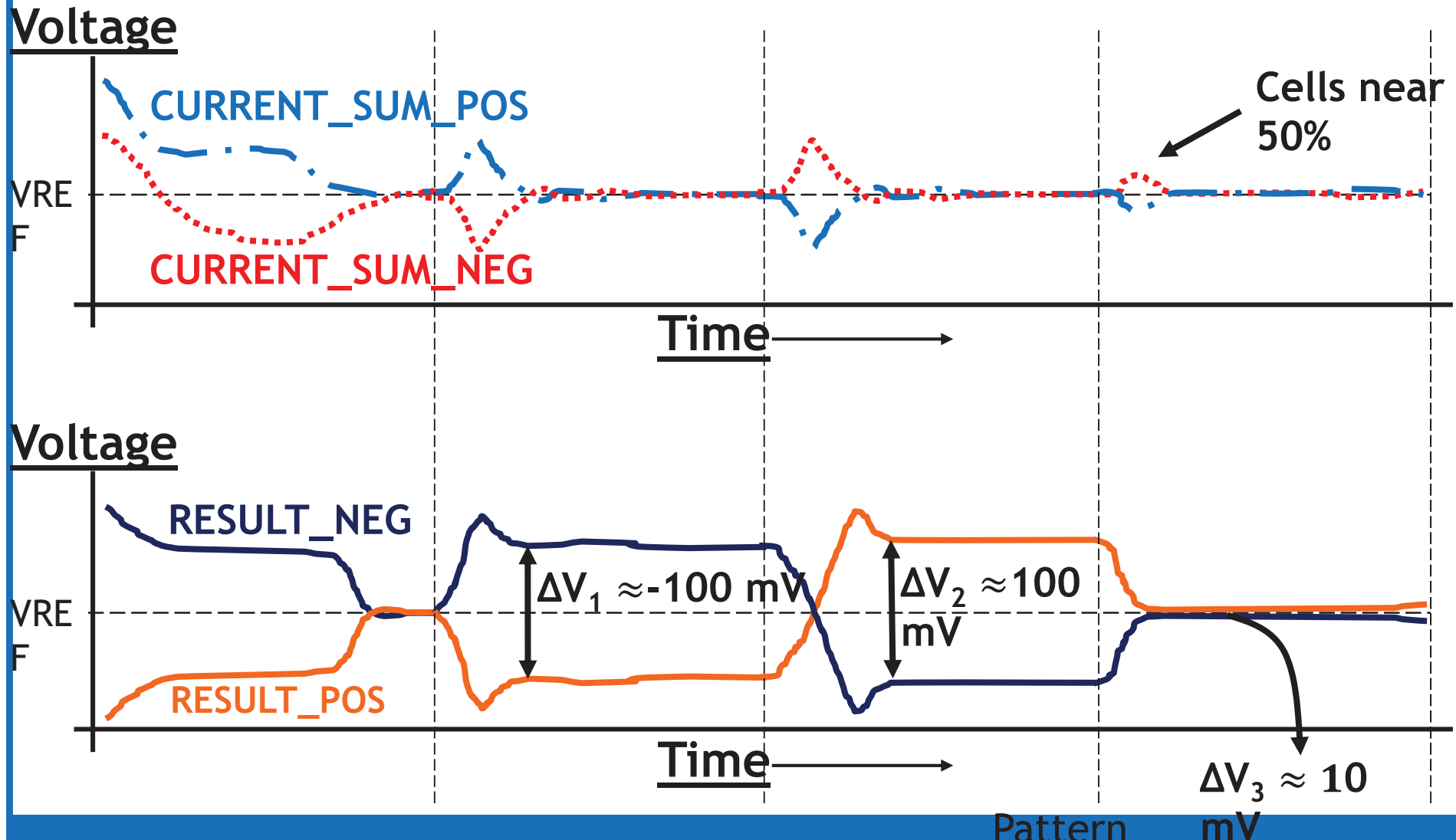
Performing a Calculation: Execution (48.5%)



Performing a Calculation: Execution (51.5%)

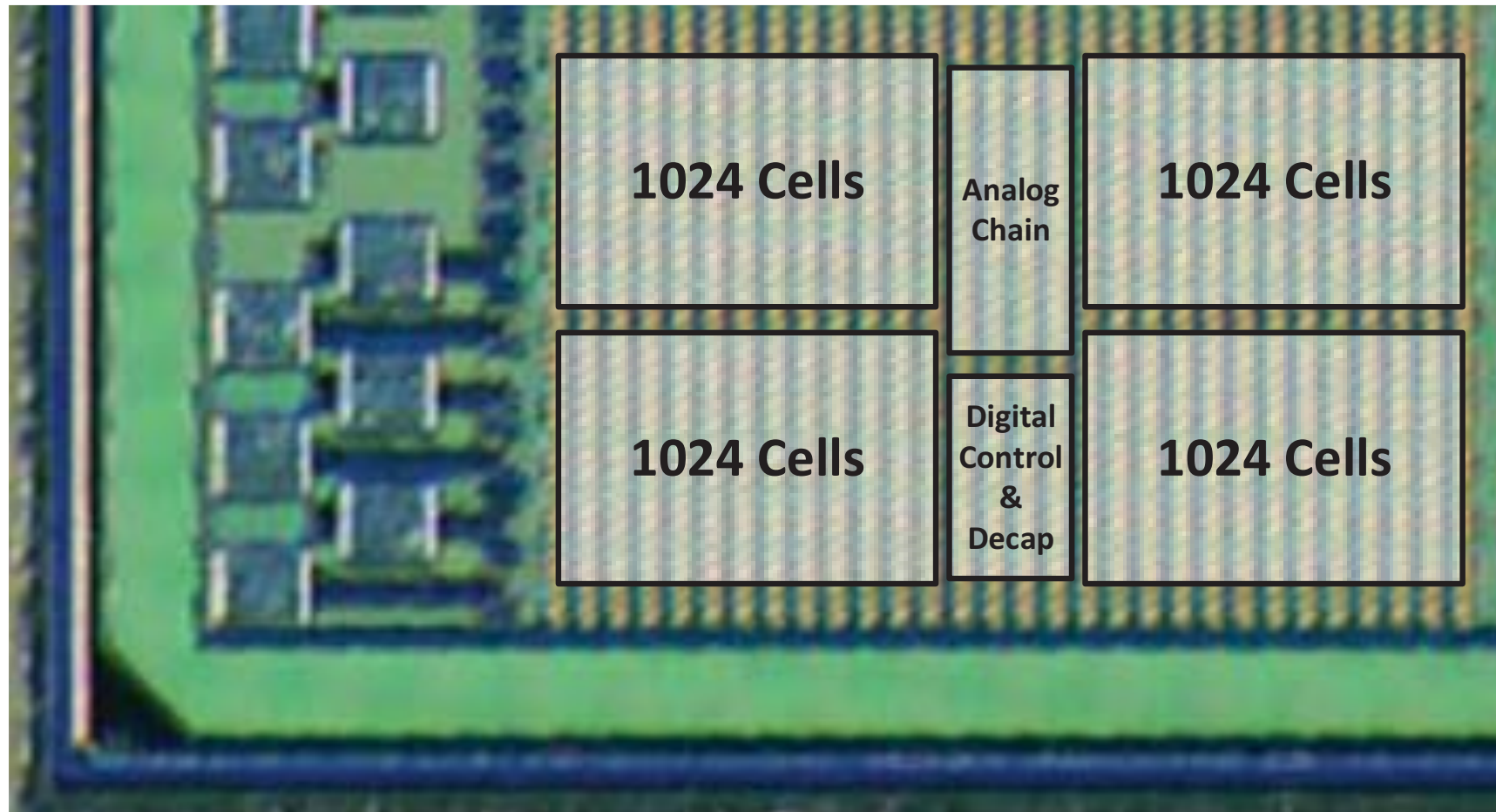


Performing a Calculation: Execution (50%)



Die Photo

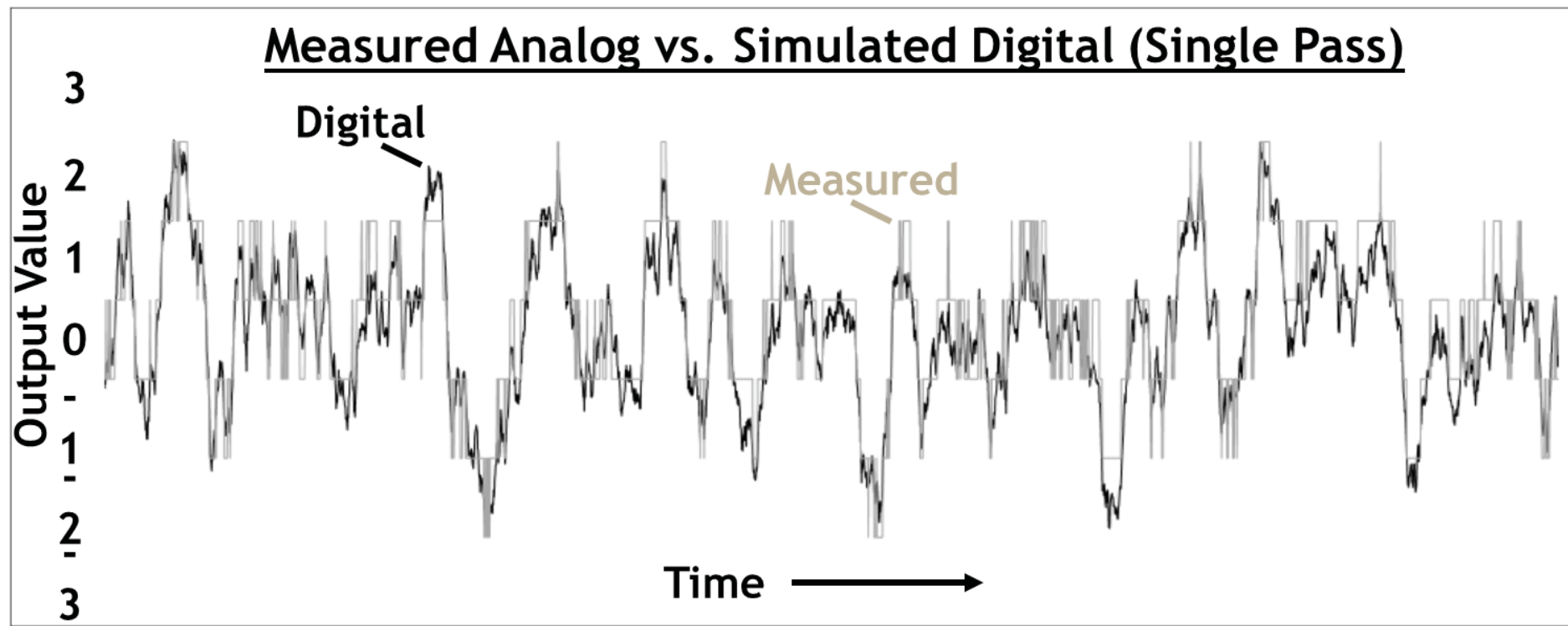
- TSMC65LP ■ All biases generated on-chip ■ 0.325 mm²



Results:

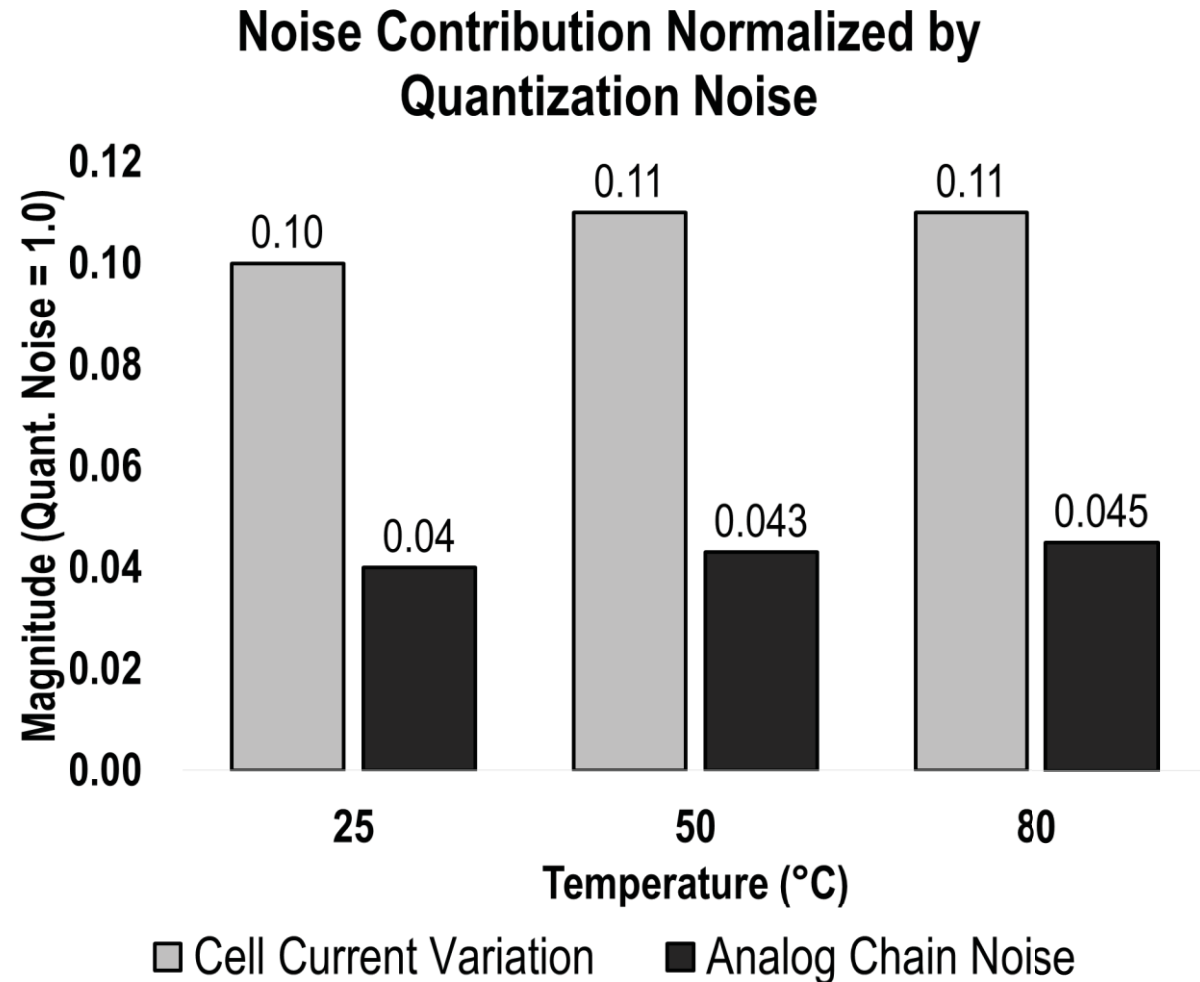
Implementation vs. Ideal

- Digital simulation is ideal
- Single pass through matched filter
 - 4096 correlation calculations
- Measured output overlaid digital
 - 170 MHz
 - 25°C



Results: Analog Computation

- Signal has inherent noise
 - RF front-end
- Analog computation noise:
 - 10× lower
 - Dominated by current source variation



Results: Comparison

- 340-27,000× performance increase
- 67× energy efficiency increase
- Scalable for application

	This Work	MITRE	JSSC'05	ISCAS'11
Technology	65 nm	180 nm	350 nm	130 nm
V_{DD} (V)	1.20 (Analog) 1.15 (Digital)	1.8	2.0	1.0
Clock Frequency (MHz)	170	20.46	8	0.2
Power (mW)	18.9*	1,900	2	0.0004
TOPS	0.70*	1.05	2.05E-3	2.56E-5
TOPS/W	36.8	0.55	1.05	64.0
TOPS/mm²	2.154*	0.0119	0.0038	0.000197
Vector Length	4,096*	51,150	256	128
Quantization	2-bit	2-bit	Analog	Analog
Area (mm²)	0.325*	88.0	0.54	0.13
Topology	Digital storage/ switched current	All digital	Analog storage/ switched current	Analog storage/ switched capacitor

*The design could be tiled to proportionally scale these metrics.

Conclusion

- Implemented a 36.8 2b-TOPS/W matched filter for GPS application
- Uses analog calculation to achieve improvement in:
 - Energy
 - 67× gain in energy efficiency compared to all-digital implementation
 - Performance
 - 340-27,000× higher performance than previous analog implementations
 - Area
- Analog calculation has negligible noise contributions



Analog Computation in Flash Memory for Datacenter-scale AI Inference in a Small Chip


Dave Fick, CTO/Founder
Mike Henry, CEO/Founder

HotChips 2018

DNNs are Largely Multiply-Accumulate

Primary DNN Calculation is Input Vector * Weight Matrix = Output Vector

Input Data	Neuron Weights	Outputs Equations
$[X_0 \quad X_1 \quad \dots \quad X_N]$	$\begin{bmatrix} A_0 & B_0 & C_0 \\ A_1 & B_1 & C_1 \\ \dots & \dots & \dots \\ A_N & B_N & C_N \end{bmatrix}$	$= \begin{bmatrix} Y_A = X_0A_0 + X_1A_1 + X_2A_2 \\ Y_B = X_0B_0 + X_1B_1 + X_2B_2 \\ Y_C = X_0C_0 + X_1C_1 + X_2C_2 \end{bmatrix}$

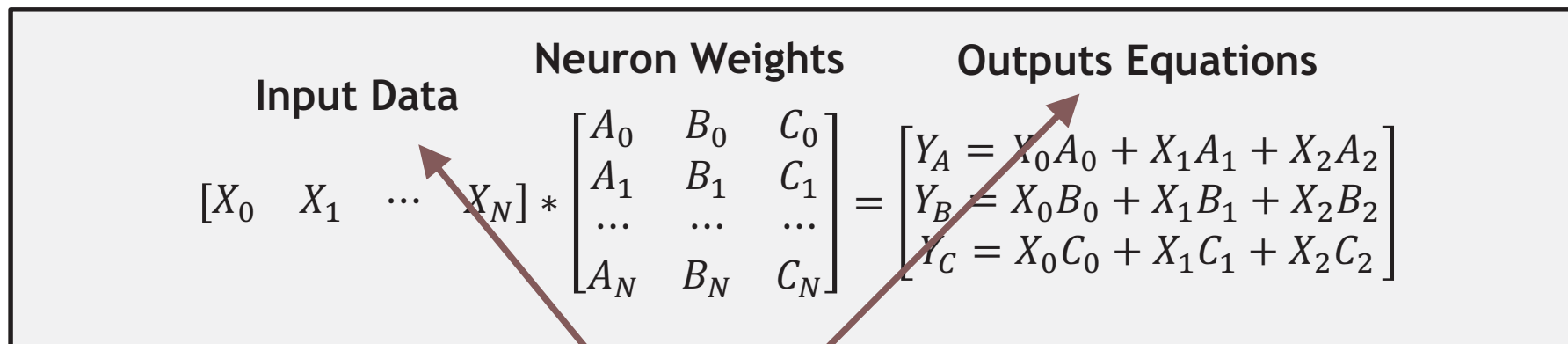


Key Operation: Multiply-Accumulate, or “MAC”

Figure of Merit: How many picojoules to execute a MAC?

Memory Access Includes Weight Data and Intermediate Data

“Weight Data”



“Intermediate Data”

For a 1000 input, 1000 neuron matrix....

Intermediate Data Accesses are Naturally Amortized

$$\begin{array}{ccc} \text{1,000 Inputs} & \text{1,000,000 Weights} & \text{1,000 Outputs} \\ [X_0] \text{ } X_1 \text{ } \dots \text{ } X_N & * \begin{bmatrix} A_0 & B_0 & C_0 \\ A_1 & B_1 & C_1 \\ \dots & \dots & \dots \\ A_N & B_N & C_N \end{bmatrix} & = \begin{bmatrix} Y_A = X_0 A_0 + X_1 A_1 + X_2 A_2 \\ Y_B = X_0 B_0 + X_1 B_1 + X_2 B_2 \\ Y_C = X_0 C_0 + X_1 C_1 + X_2 C_2 \end{bmatrix} \end{array}$$

Intermediate data accesses are amortized **64-1024x** since they are used in many MAC operations

For a 1000 input, 1000 neuron matrix....

Weight Data Accesses are Not Amortized

$$\begin{array}{ccc} \text{1,000 Inputs} & \text{1,000,000 Weights} & \text{1,000 Outputs} \\ [X_0 \ X_1 \ \dots \ X_N] * \begin{bmatrix} A_0 & B_0 & C_0 \\ A_1 & B_1 & C_1 \\ \dots & \dots & \dots \\ A_N & B_N & C_N \end{bmatrix} & = & \begin{bmatrix} Y_A = X_0A_0 + X_1A_1 + X_2A_2 \\ Y_B = X_0B_0 + X_1B_1 + X_2B_2 \\ Y_C = X_0C_0 + X_1C_1 + X_2C_2 \end{bmatrix} \end{array}$$

Weight data could need to be stored in *DRAM*, and it does not have the same amortization as the intermediate data

DNN Processing is All About Weight Memory

Network	Weights	MACs	...@ 30 FPS
AlexNet ¹	61 M	725 M	22 B
ResNet-18 ¹	11 M	1.8 B	54 B
ResNet-50 ¹	23 M	3.5 B	105 B
VGG-19 ¹	144 M	22 B	660 B
OpenPose ²	46 M	180 B	5400 B

Very hard to fit this
in an Edge solution

- ✓ 10+M parameters to store
- ✓ 20+B memory accesses
- ✓ How do we achieve...
 - High Energy Efficiency
 - High Performance
 - “Edge” Power Budget (e.g., 5W)

¹: 224x224 resolution

²: 656x368 resolution

Common Techniques for Reducing Weight Energy Consumption

Weight Re-use

- **Focus on CNN**
 - Re-use weights for multiple windows
 - Can build specialized structures
 - ☹ *Not all problems map to CNN well*
- **Focus on Large Batch**
 - Re-use weights for multiple inputs
 - ☹ *Edge is often batch=1*
 - ☹ *Increases latency*

Weight Reduction

- **Shrink the Model**
 - Use a smaller network that can fit on-chip (e.g., SqueezeNet)
 - 😊 *Possibly reduced capability*
- **Compress the Model**
 - Use sparsity to eliminate up to 99% of the parameters
 - Use literal compression
 - 😊 *Possibly reduced capability*
- **Reduce Weight Precision**
 - 32b Floating Point => 2-8b Integer
 - 😊 *Possibly reduced capability*

Key Question: Use DRAM or Not?

Benefits of DRAM

- 😊 Can fit arbitrarily large models
- 😊 Not as much SRAM needed on chip

Drawbacks of DRAM

- 😞 Huge energy cost for reading weights
- 😞 Limited bandwidth getting to weight data
- 😞 Variable energy efficiency & performance depending on application

Common NN Accelerator Design Points

	Enterprise With DRAM	Enterprise No-DRAM	Edge With DRAM	Edge No-DRAM
SRAM	<50 MB	100+ MB	< 5 MB	< 5 MB
DRAM	8+ GB	-	4-8 GB	-
Power	70+ W	70+ W	3-5 W	1-3 W
Sparsity	Light	Light	Moderate	Heavy
Precision	32f / 16f / 8i	32f / 16f / 8i	8i	1-8i
Accuracy	Great	Great	Moderate	Poor
Performance	High	High	Very Low	Very Low
Efficiency	25 pJ/MAC	2 pJ/MAC	10 pJ/MAC	5 pJ/MAC

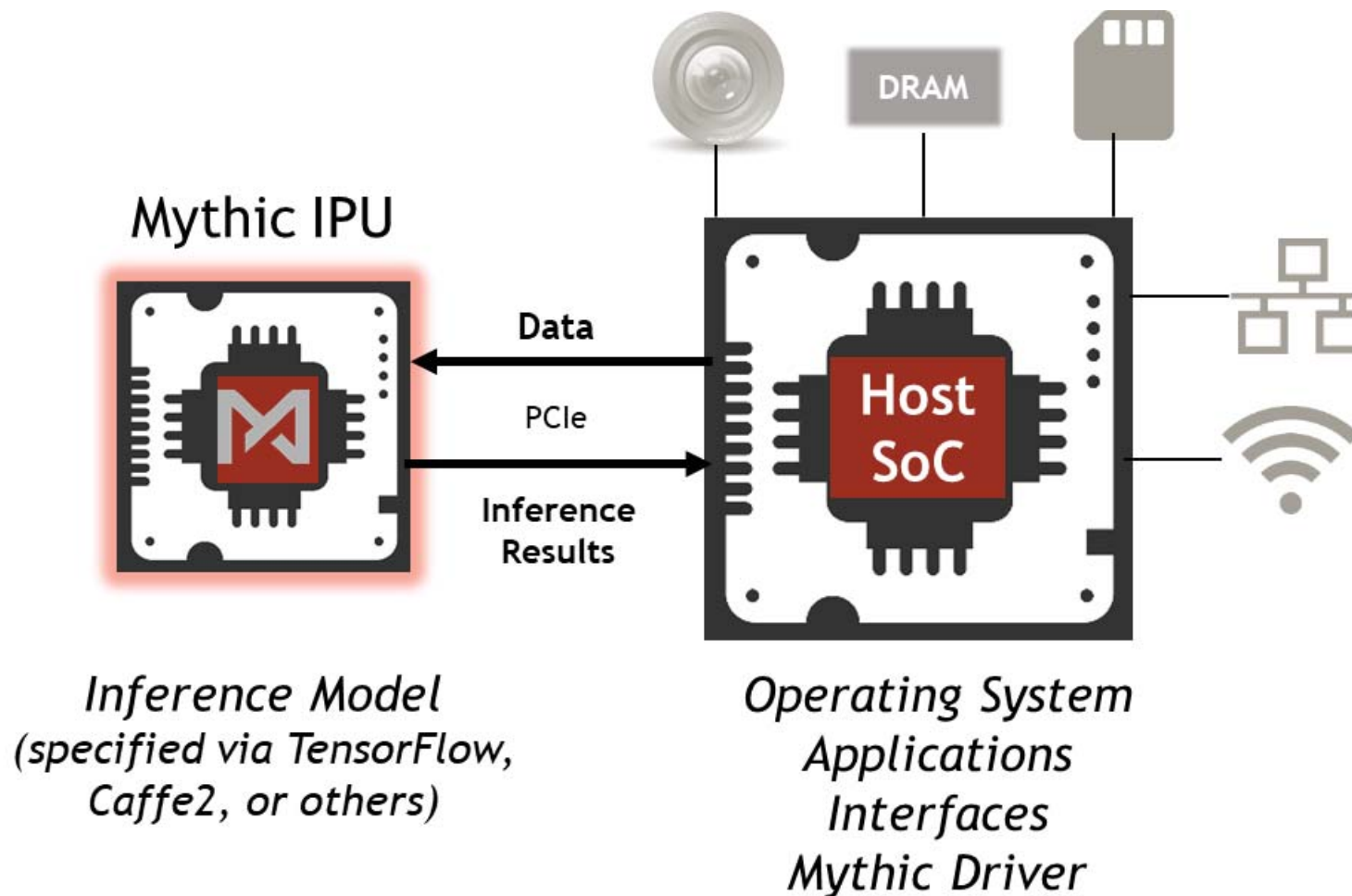
Mythic is Fundamentally Different

	Enterprise With DRAM	Enterprise No-DRAM	Edge With DRAM	Edge No-DRAM	Mythic NVM
SRAM	<50 MB	100+ MB	< 5 MB	< 5 MB	< 5 MB
DRAM	8+ GB	-	4-8 GB	-	-
Power	70+ W	70+ W	3-5 W	1-3 W	1-5 W
Sparsity	Light	Light	Moderate	Heavy	None
Precision	32f / 16f / 8i	32f / 16f / 8i	8i	1-8i	1-8i
Accuracy	Great	Great	Moderate	Poor	Great
Performance	High	High	Very Low	Very Low	High
Efficiency	25 pJ/MAC	2 pJ/MAC	10 pJ/MAC	5 pJ/MAC	0.5 pJ/MAC

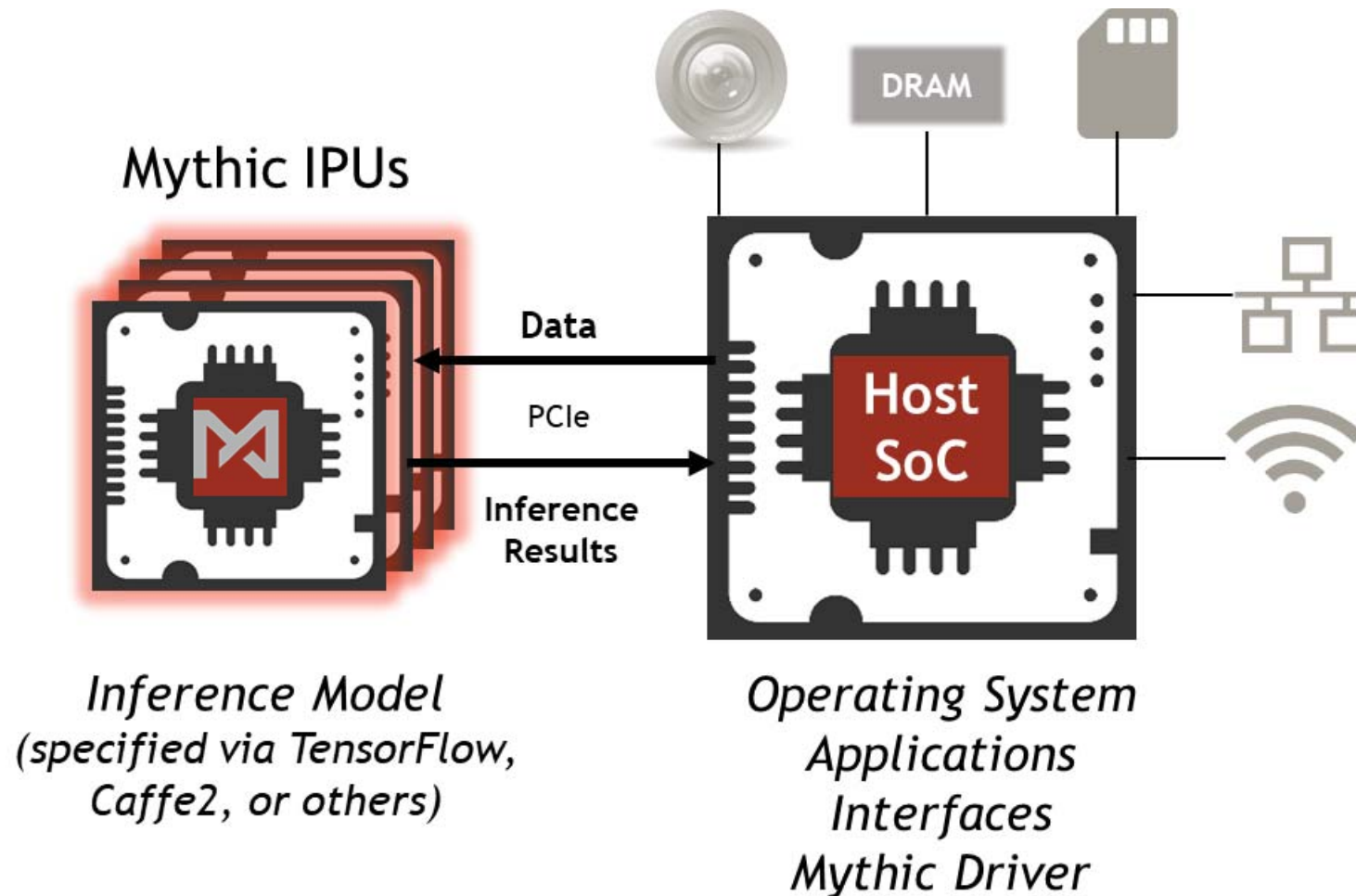
Mythic is Fundamentally Different

	Enterprise With DRAM	Enterprise No-DRAM	Edge With DRAM	Edge No-DRAM	Mythic NVM
SRAM	<50 MB	100+ MB	< 5 MB	< 5 MB	< 5 MB
DRAM	8+ GB	-	4-8 GB	-	-
Power	70+ W	70+ W	3-5 W	1-3 W	1-5 W
Sparsity	Light	Also, Mythic does this in a 40nm process, compared to 7/10/16nm			None
Precision	32f / 18i				1-8i
Accuracy	Great	Great	Moderate	Poor	Great
Performance	High	High	Very Low	Very Low	High
Efficiency	25 pJ/MAC	2 pJ/MAC	10 pJ/MAC	5 pJ/MAC	0.5 pJ/MAC

Mythic is a PCIe Accelerator

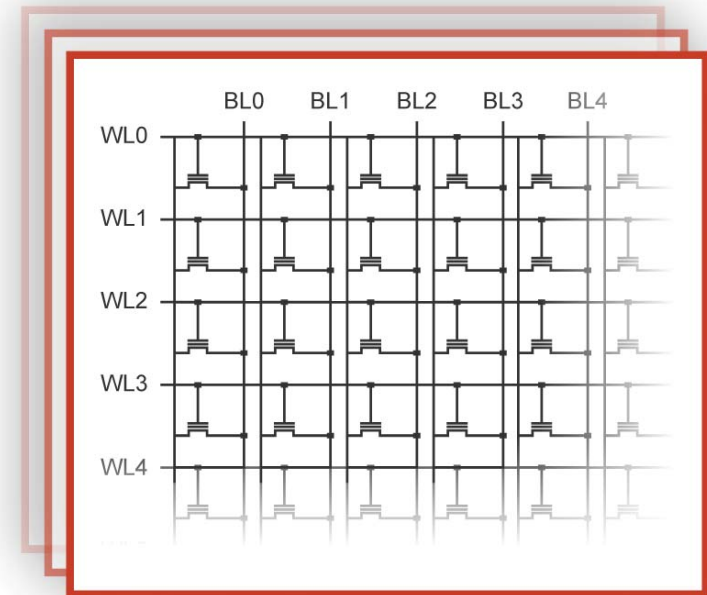


We Also Support Multiple IPU's



Mythic's New Architecture Merges Enterprise and Edge

- ✓ Mythic introduces the ***Matrix Multiplying Memory***
 - Never read weights
- ✓ This effectively makes weight memory access ***energy-free*** (only pay for MAC)
- ✓ And eliminates the need for...
 - Batch > 1
 - CNN Focus
 - Sparsity or Compression
 - Nerfed DNN Models



*Made possible with
Mixed-Signal Computing
on embedded flash*

Revisiting Matrix Multiply

Primary DNN Calculation is Input Vector * Weight Matrix = Output Vector

Input Data	Neuron Weights	Outputs Equations
$[X_0 \quad X_1 \quad \dots \quad X_N]$	$\begin{bmatrix} A_0 & B_0 & C_0 \\ A_1 & B_1 & C_1 \\ \dots & \dots & \dots \\ A_N & B_N & C_N \end{bmatrix}$	$\begin{bmatrix} Y_A = X_0A_0 + X_1A_1 + X_2A_2 \\ Y_B = X_0B_0 + X_1B_1 + X_2B_2 \\ Y_C = X_0C_0 + X_1C_1 + X_2C_2 \end{bmatrix}$

Flash Transistors

Analog Circuits Give us the MAC We Need

Flash transistors can be modeled as **variable resistors** representing the weight

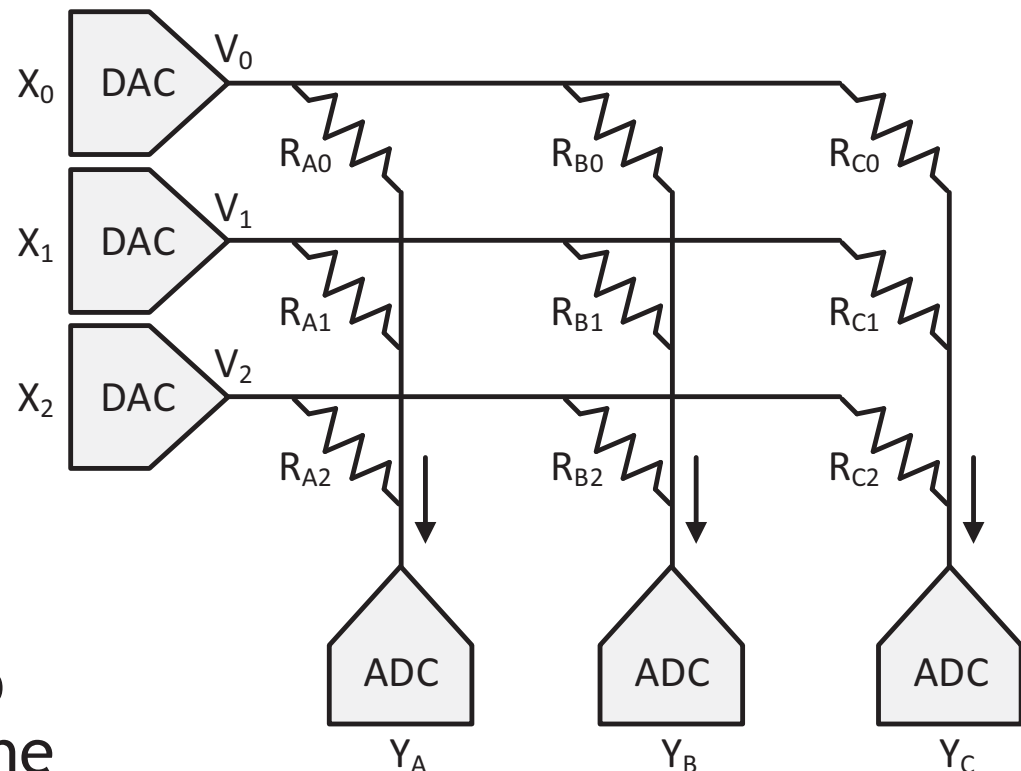
The $V=IR$ current equation will achieve the math we need:

Inputs (X) = DAC

Weights (R) = Flash transistors

Outputs (Y) = ADC Outputs

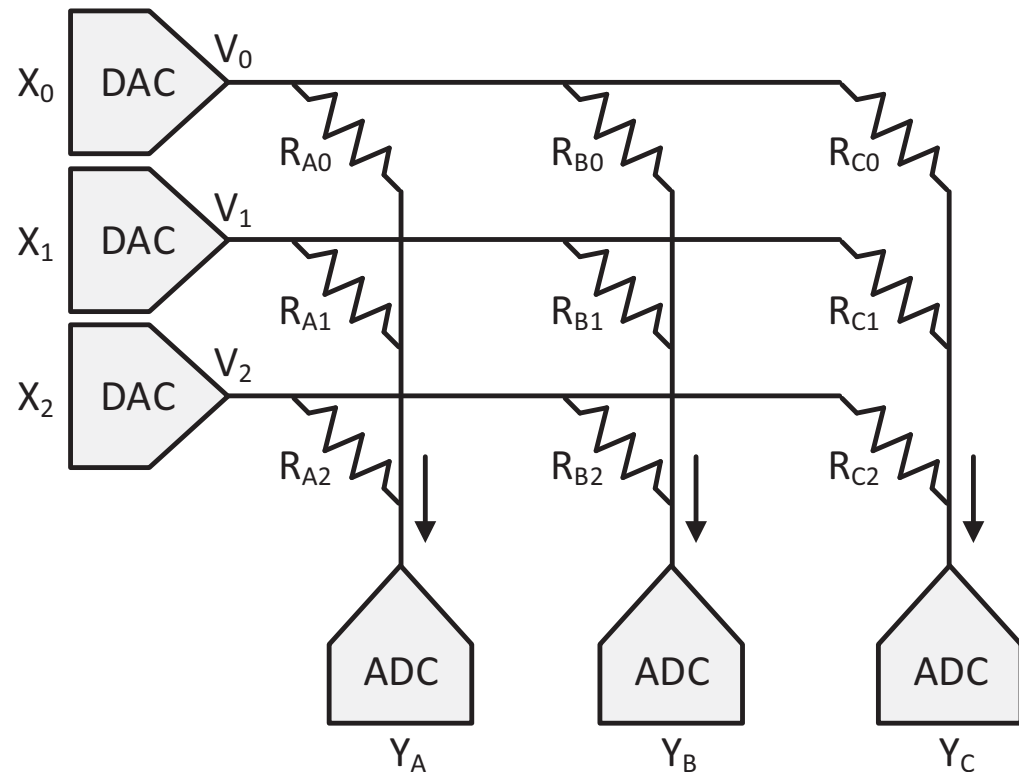
The ADCs convert current to digital codes, and provide the non-linearity needed for DNN



DACs & ADCs Give Us a Flexible Architecture

We have a **digital** top-level architecture:

- Interconnect
- Intermediate data storage
- Programmability
(XLA/ONNX => Mythic IPU)

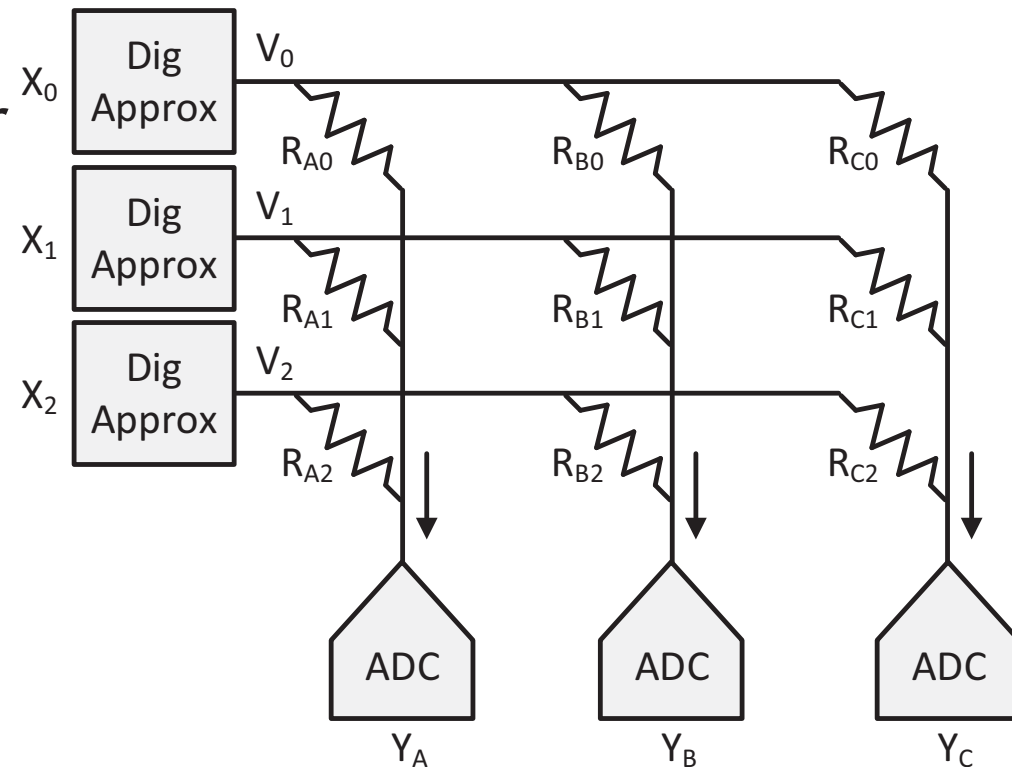


To Simplify we use Digital Approximation

To improve time-to-market,
we have left the Input
DAC as a future endeavor

We achieve the same
result through digital
approximation

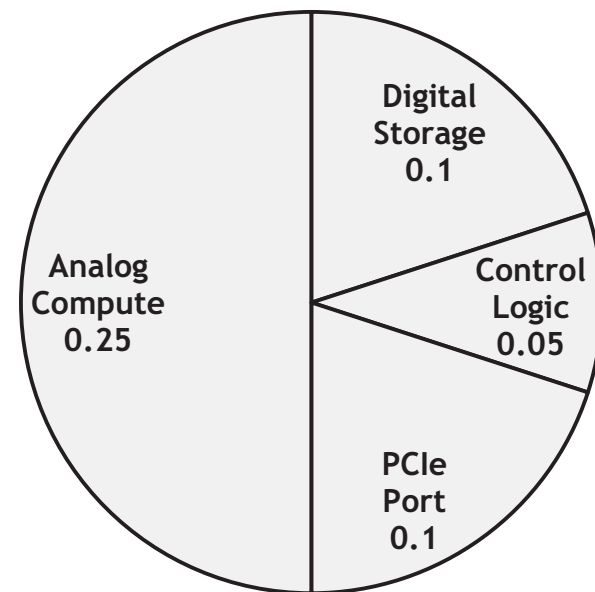
Silver lining: we have
future improvements
available



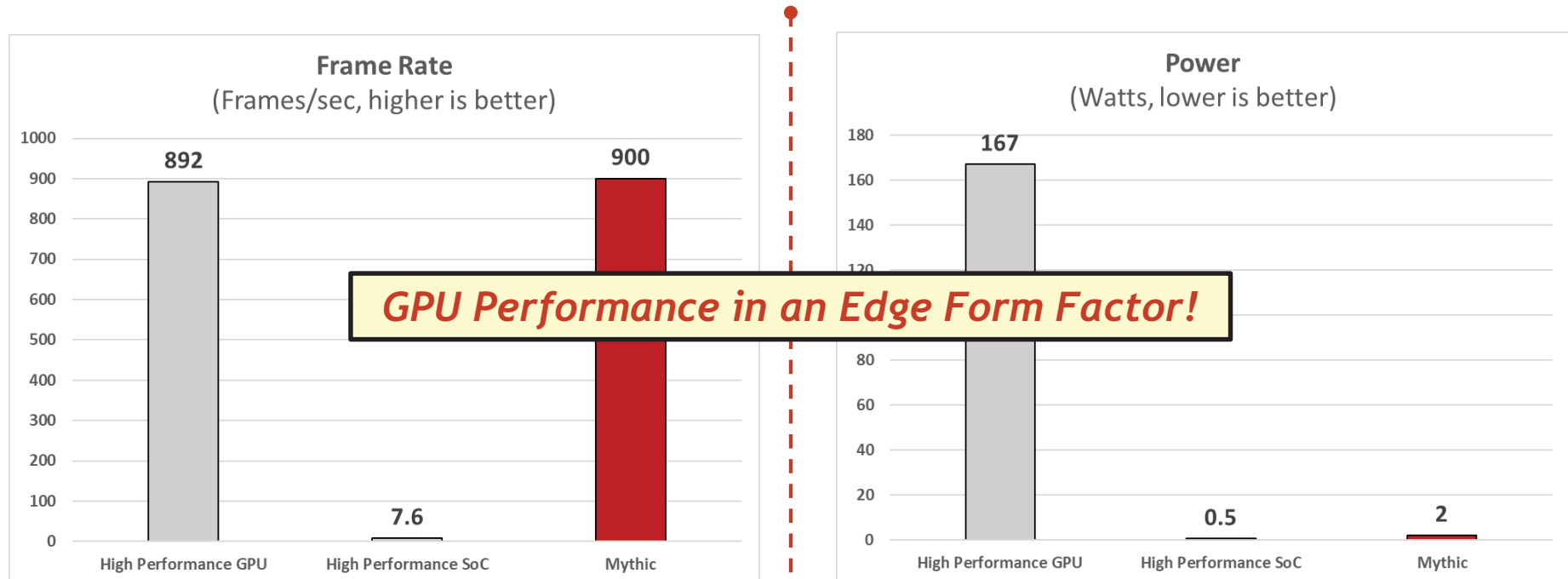
We Account For All Energy Consumed

- ✓ Numbers are for a typical application, e.g. ResNet-50
 - Batch size = 1
 - We are relatively application-agnostic (especially compared to DRAM-based systems)
- ✓ 8b analog compute accounts for about half of our energy
 - We can also run lower precision
 - Control, storage, and PCIe accounts for the other half

Energy (pJ/MAC)
Total = 0.5

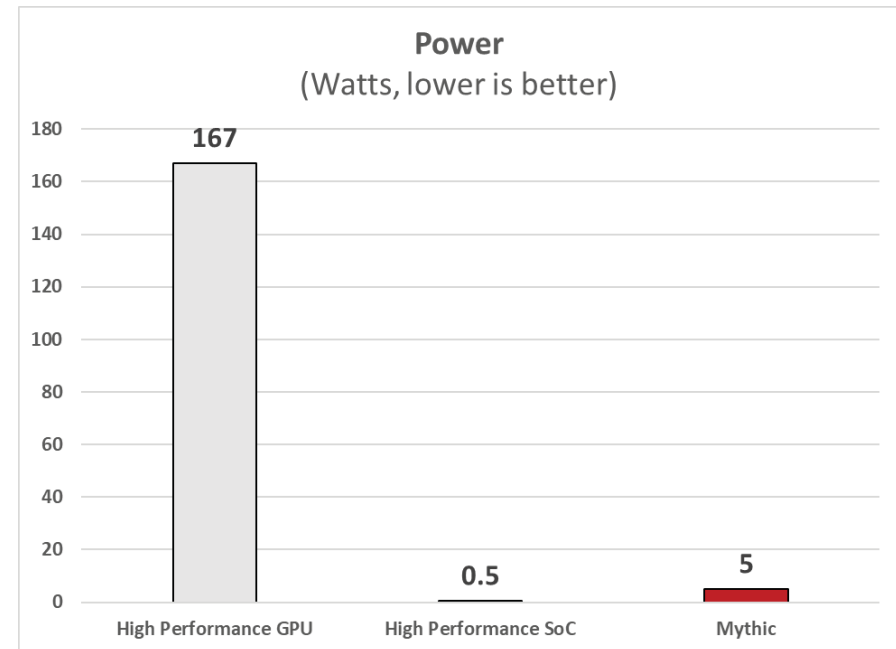
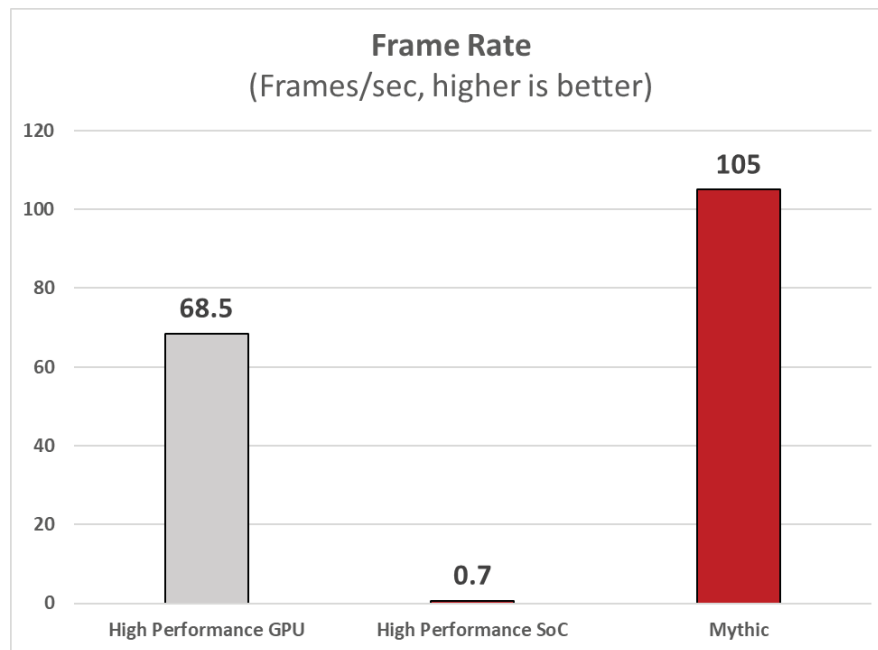


Example Application: ResNet-50



Running at 224x224 resolution. Mythic estimated, GPU/SoC measured

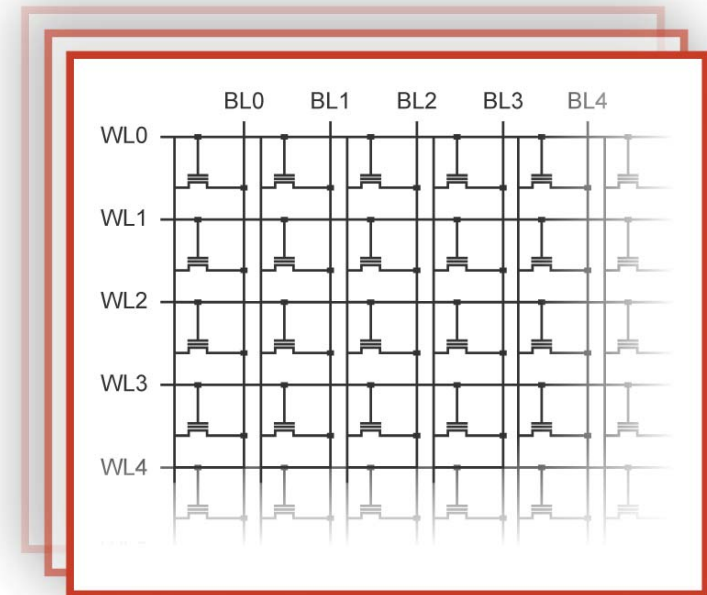
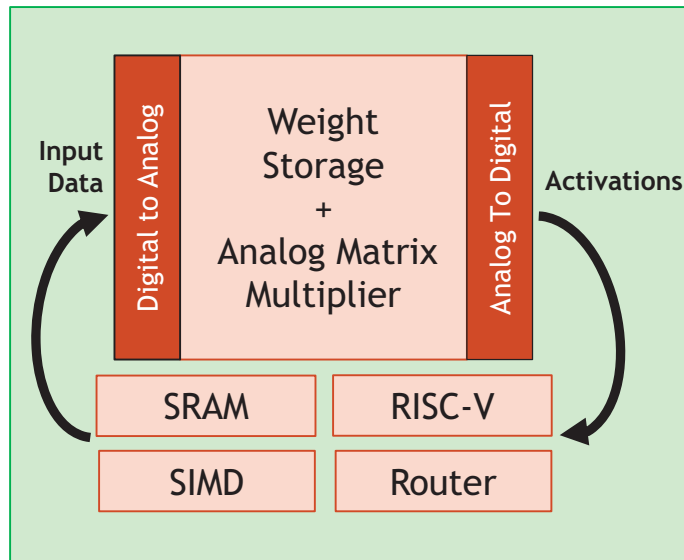
Example Application: OpenPose



Running at 656x368 resolution. Mythic estimated, GPU/SoC measured

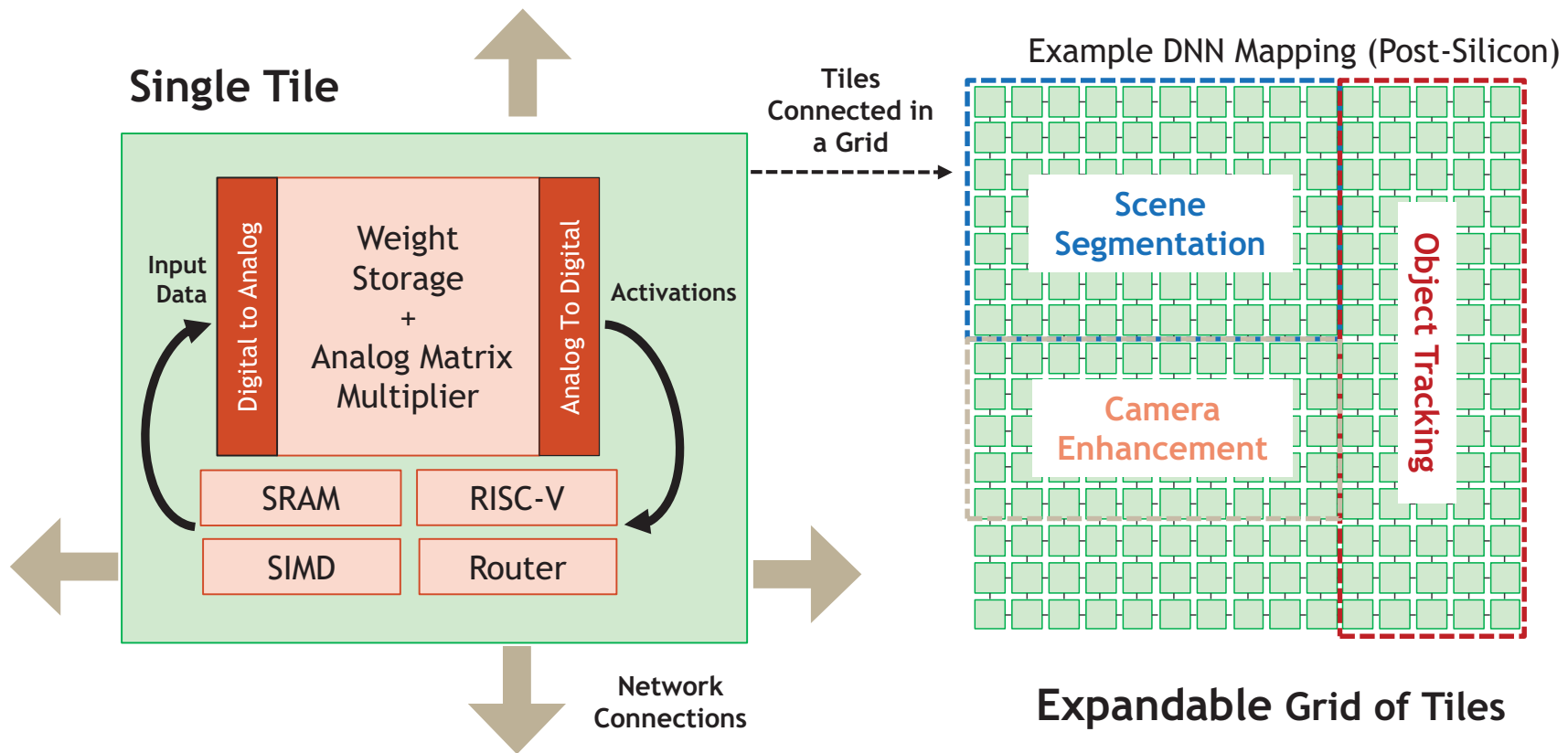
Mythic Mixed-Signal Computing

Single Tile



*Made possible with
Mixed-Signal Computing
on embedded flash*

Mythic Mixed-Signal Computing



System Overview

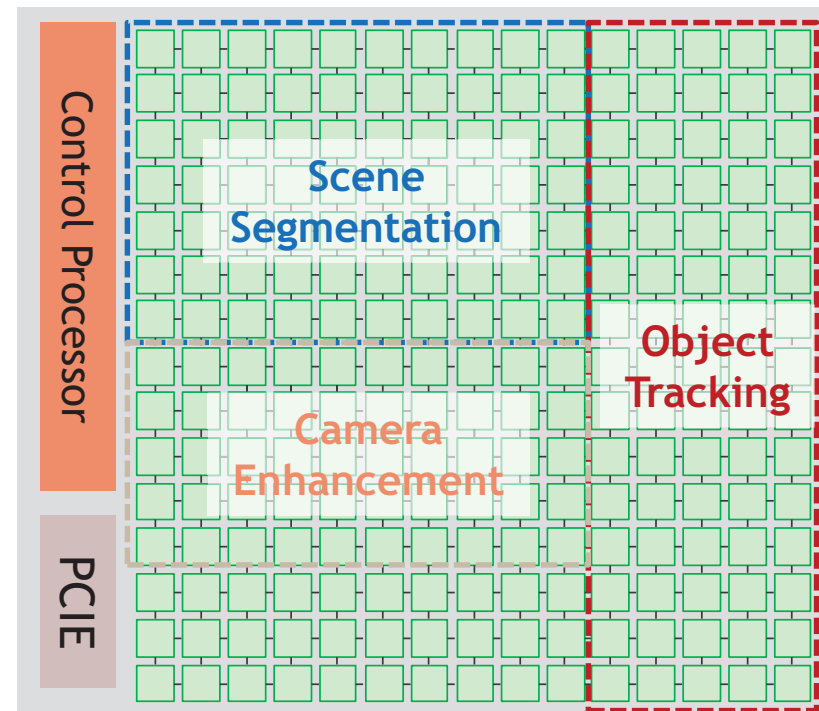
Initial Product

- 50M weight capacity
- PCIe 2.1 x4
- Basic Control Processor

Envisioned Customizations (Gen 1)

- Up to 250M weight capacity
- PCIe 2.1 x16
- USB 3.0/2.0
- Direct Audio/Video Interfaces
- Enhanced Control Processor (e.g., ARM)

Intelligence Processing Unit (IPU)





Wrapping Up

What is Possible with Compute-in-Memory?

- >10x improvement in energy efficiency
- >10x improvement in performance
- Application specific benefits
 - Not every algorithm can benefit from CiM!
 - Some benefit more than others

Compute-in-Memory Considerations

- What does the working set look like?
 - Is it “wide”?
 - Is it “large”?
- How important is this algorithm to our system?
 - Does it use up 90% of something?
- How predictable are our data patterns?
 - Can we reduce data movement somehow?