tinyML® Summit

Miniature dreams can come true...

March 28-30, 2022 | San Francisco Bay Area

www.tinyML.org

Mastering the 3 Pillars of AI Acceleration: Algorithms, Hardware and Software

Swagath Venkataramani

IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

TinyML Summit 2022

The evolution of AI: Past, Present and Future

Narrow AI

Single task, single domain

Superhuman accuracy and speed for certain tasks

Broad AI

Multi-task, multi-domain

Multi-modal

Distributed AI

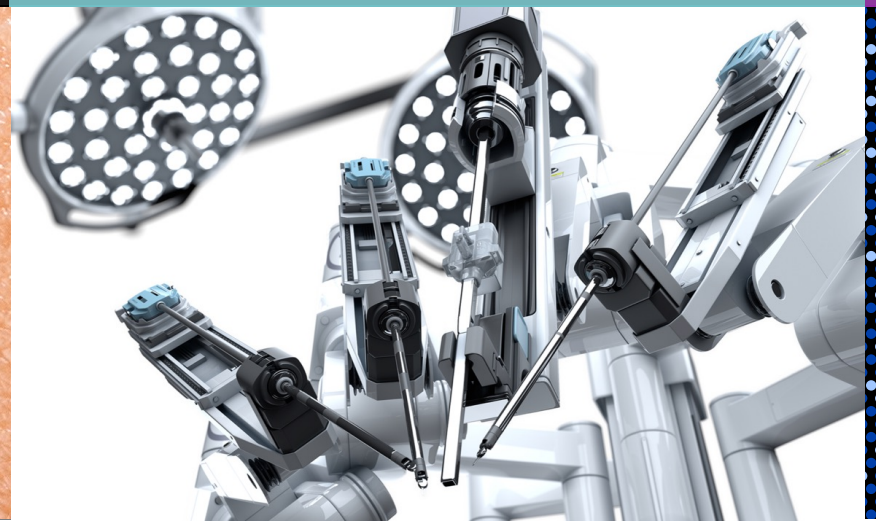
Explainable

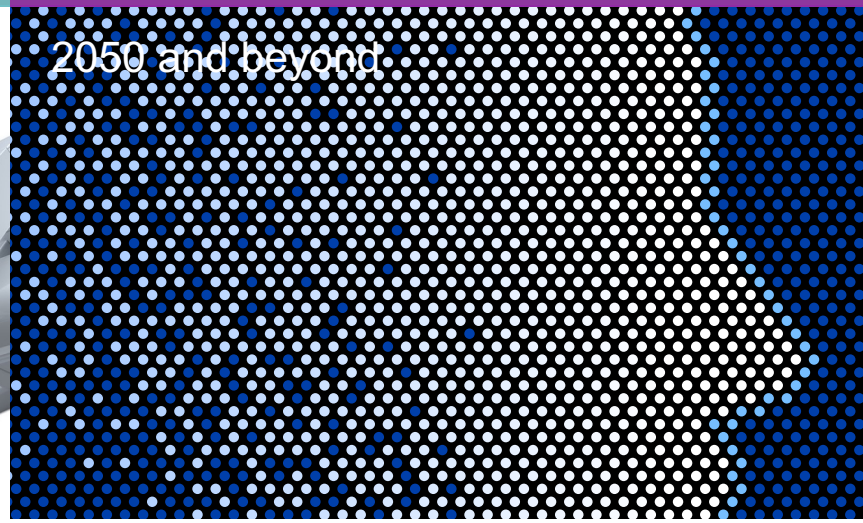
▼ We are here

General AI

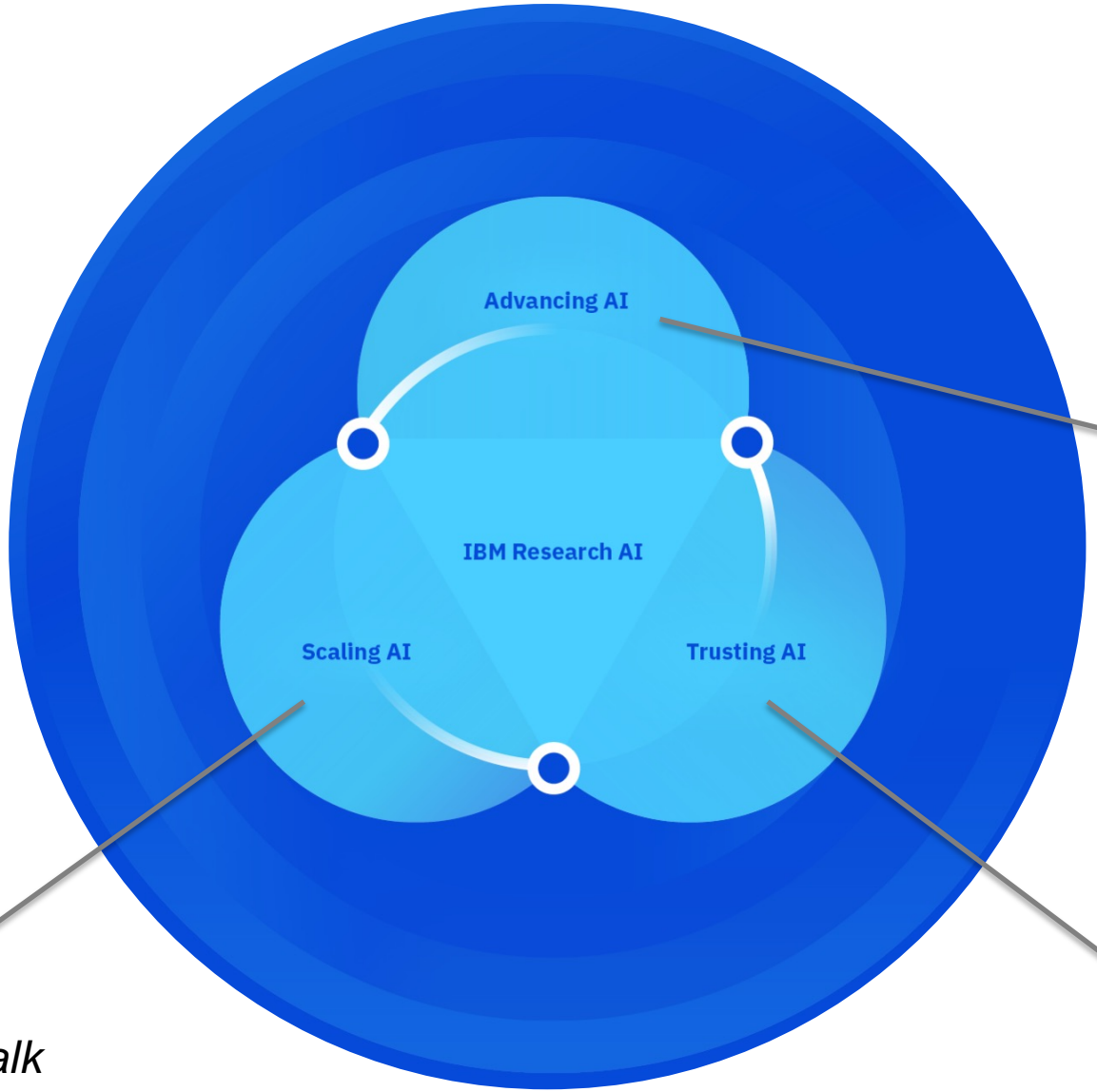
Cross-domain learning and reasoning

Broad autonomy





The path to Broad AI



Advancing AI
Powering advances in perception, reasoning and understanding to help AI address complex human-like tasks.

Scaling AI
Novel technologies across the full computing stack that make AI faster, easier, and able to scale to larger and more complex problems.

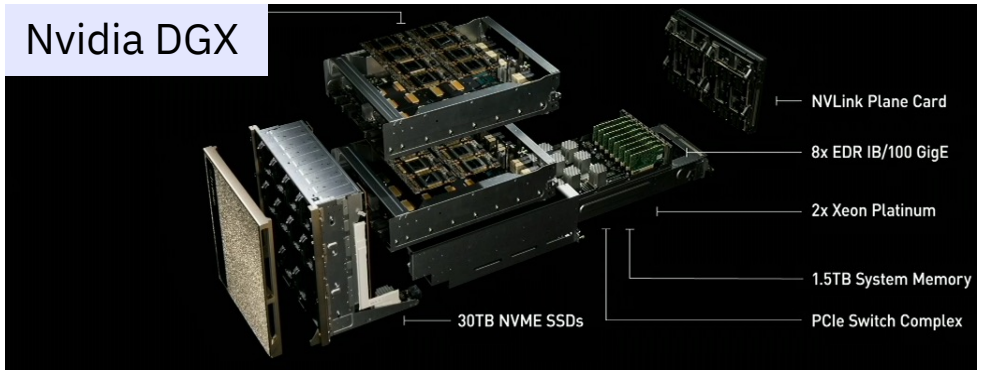
Focus of this talk

Trusting AI
Novel techniques to instrument key dimensions of trust and enable AI solutions that inspire confidence.

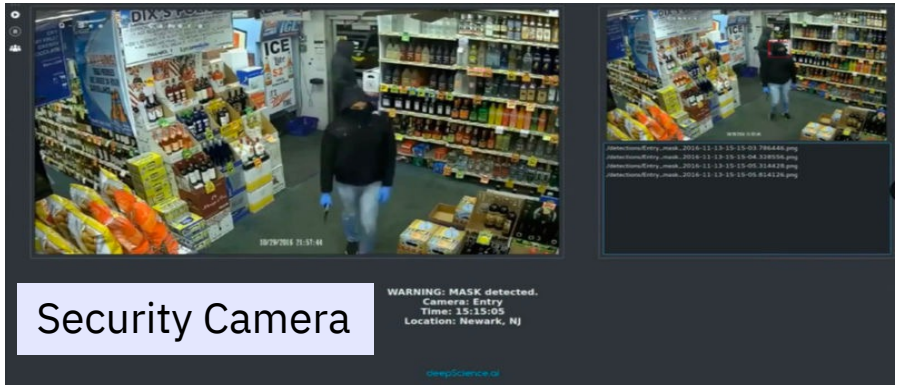
Deep Learning (DL) Training and Inference Use Cases

- **Model Training**
  - Typically on-prem
  - Customized GPU / ASIC Cluster of > 16 chips
  - Days-weeks to train DL models
- **Server Inference**
  - Most cases allow some form of batching (i.e. latency insensitive)
  - Currently CPU dominated – transitioning to PCIe attached accelerators
- **Transactional Inference**
  - Extremely latency sensitive – difficult to batch DL jobs.
  - Use cases : Financial industry, insurance,....
  - PCIe attached and on-CPU chip accelerators
- **Autonomous Driving (Latency sensitive)**
  - Largely focused around image, LIDAR / other sensor processing & fusion
  - Accuracy is extremely important
  - Huge # of Ops and extremely latency sensitive → customized SoCs
- **Inference on Mobile / IoT devices**
  - Security, Mobile, Home Appliances. Drones,....
  - Deep Learning based Object detection, Image Classification, Translation
  - Slightly lower accuracy tolerable

Nvidia DGX



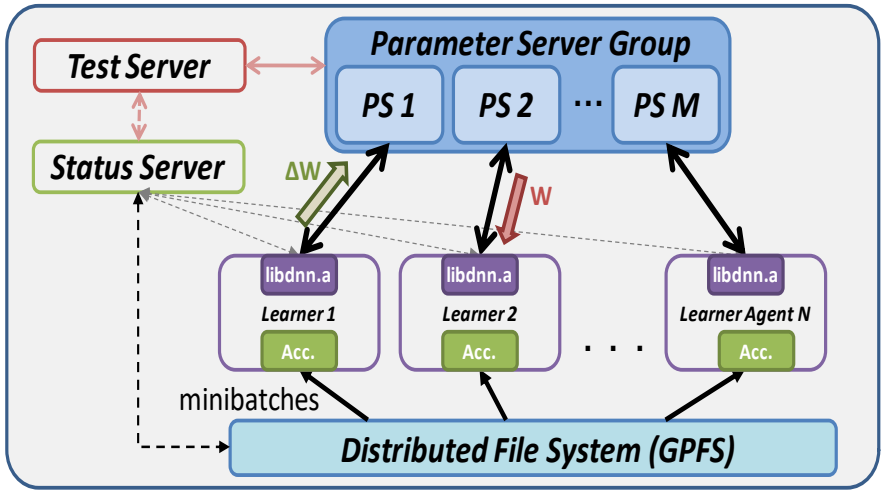
Self Driving Car



Security Camera

Deep Learning System Bottlenecks

Deep Learning Training



- **Large system training :**
  - Multiple **300W** GPUs/ASICs connected on a node (using **proprietary links**) + Multiple Nodes connected via Infiniband

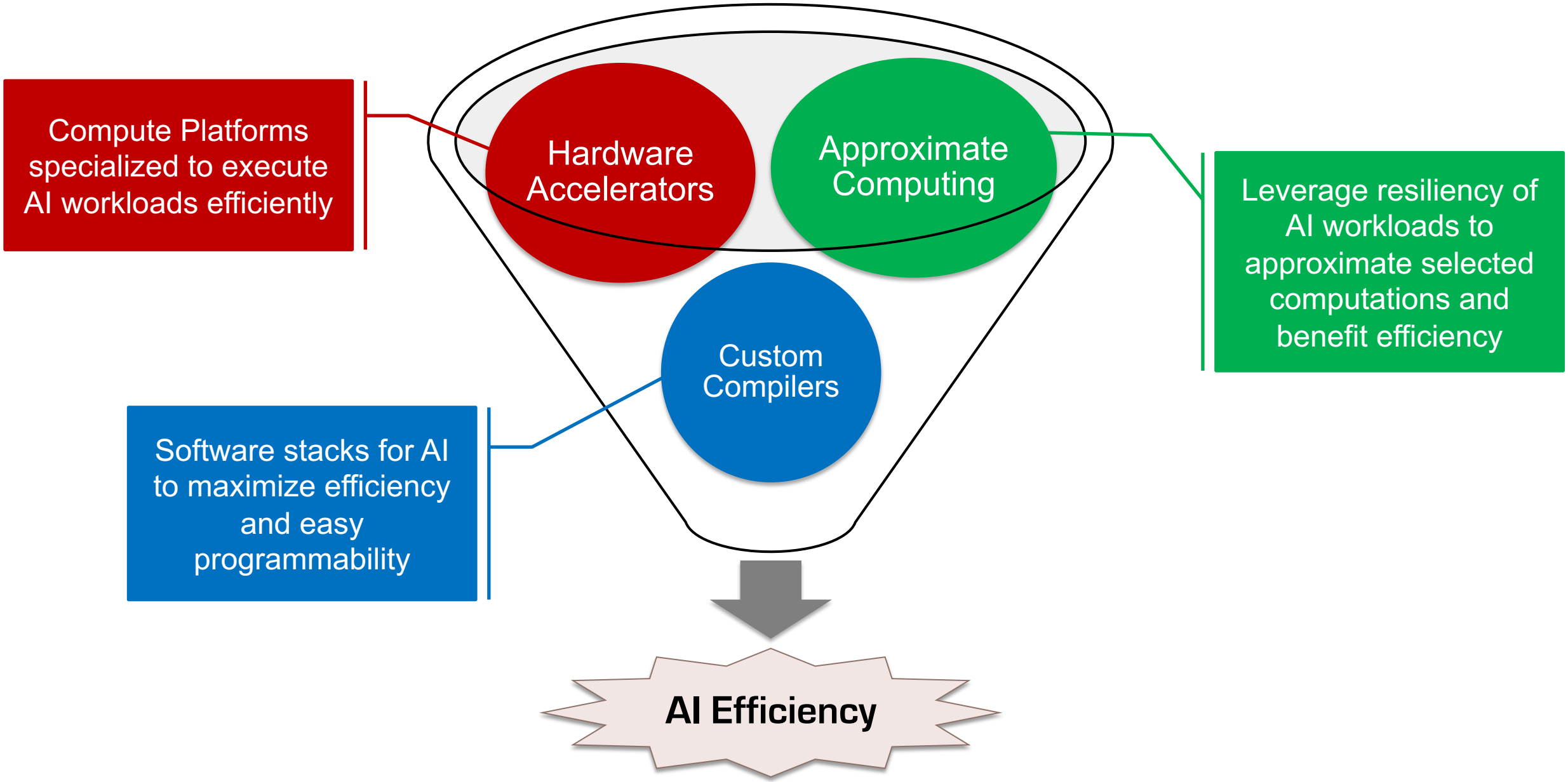
- **Training Bottlenecks:**
  - **Computation**: Minibatch SGD
    - Peak GPU / ASIC Flops capability
    - Utilization : Typical GPU utilization ~ 10 – 35% - depends on minibatch size (higher the better) – **limited by memory bandwidth**!
  - **Communication**: Different synchronization schemes possible
    - Overheads depend on superminibatch size and # of learners but limited by DL convergence - **limited by chip-to-chip bandwidth**.

Deep Learning Inference

- **Typically single-chip (board) solution**
  - **75W** PCIe attached
  - Virtualization (multi-thread /multi-process / multi-VM) support critical

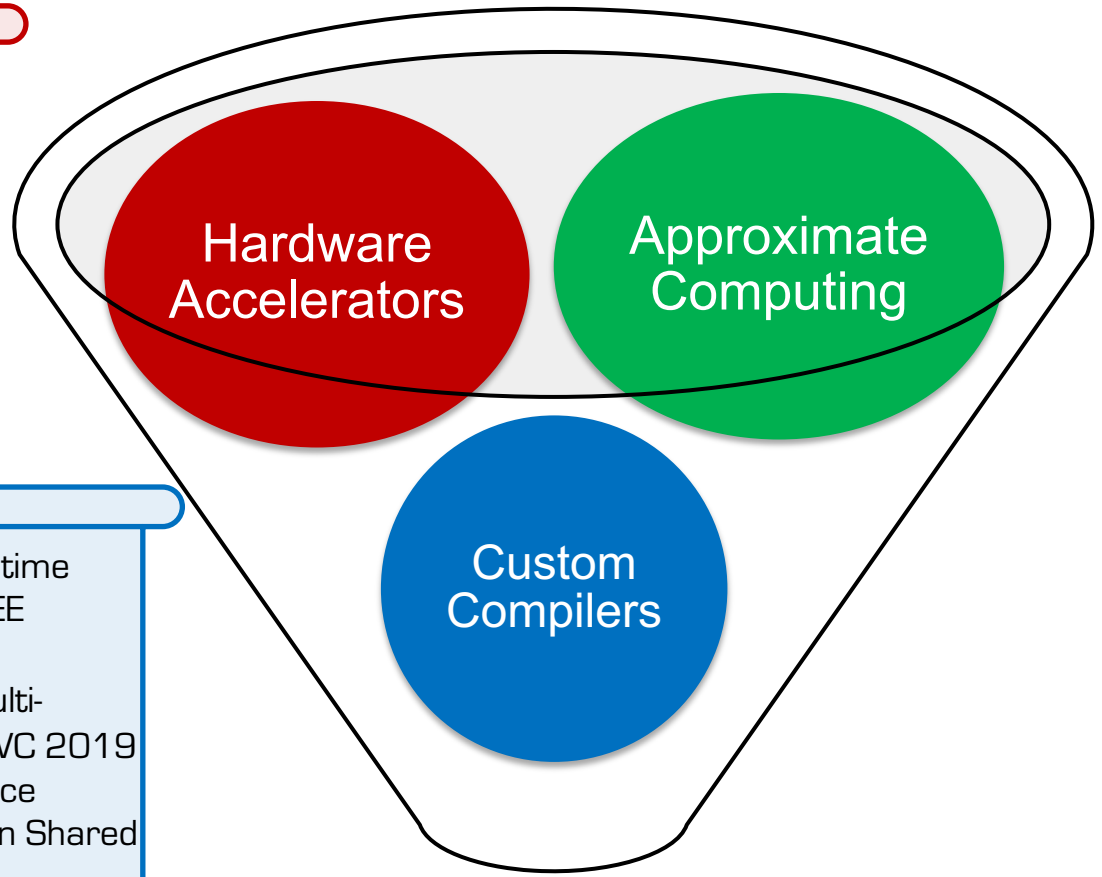
- **Inference Bottlenecks:**
  - **Computation** bottlenecks:
    - Peak GPU / ASIC Flops capability.
    - Utilization : Typically higher than training since fewer tensors – but still **limited by memory bandwidth**
  - No communication bottlenecks.
    - Multi-chip inference not common

AI System Design: Ingredients



AI System Design: Ingredients

- "A scalable multi-tera ops deep learning processor core for AI training and inference", VLSI 2018
- "DLFloat: A 16-b Floating Point format designed for Deep Learning Training and Inference", ARITH 2019



- "DeepTools: Compiler and Execution Runtime Extensions for RaPiD AI Accelerator", IEEE MICRO 2019
- Performance-driven Programming of Multi-TFLOP Deep Learning Accelerators, IISWC 2019
- Design Space Exploration for Performance Optimization of Deep Neural Networks on Shared Memory Accelerators, PACT 2017
- "Memory and Interconnect Optimizations for Peta-Scale Deep Learning Systems", HIPC 2019
- A Compiler for Deep Neural Network Accelerators to Generate Optimized Code for a Wide Range of Data Parameters from a Hand-crafted Computation Kernel, COOLCHIPS 2019

AI Efficiency

Training:

- Deep learning with limited numerical precision, ICML 2015
- Adacomp: Adaptive residual gradient compression for data-parallel distributed training, AAAI 2018
- Training deep neural networks with 8-bit floating point numbers, NeurIPS 2018
- Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, NeurIPS 2019

Inference:

- Pact: Parameterized clipping activation for quantized neural networks, arXiv 2018
- Bridging the accuracy gap for 2-bit quantized neural networks (QNN), SysML 2018
- Compensated-DNN: energy efficient low-precision deep neural networks by compensating quantization errors, DAC 2018
- BiScaled-DNN: Quantizing Long-tailed Data structures with Two Scale Factors for Deep Neural Networks, DAC 2019

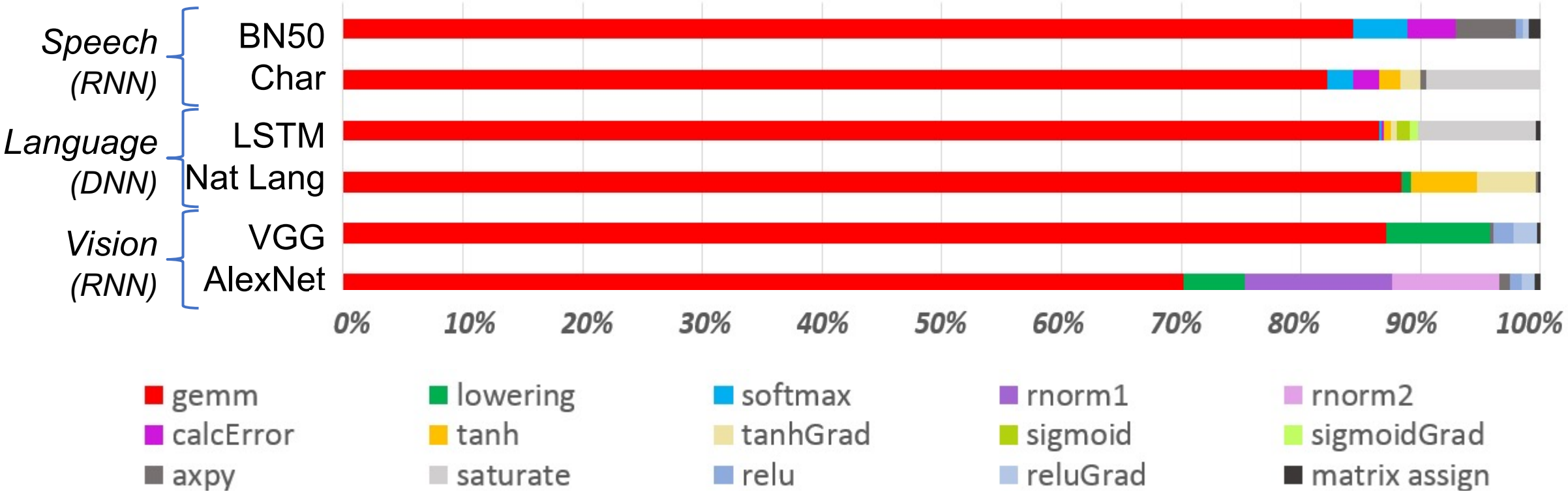
Hardware Accelerators

Deep Learning (DL) Accelerator: Hardware Design Principles

1. **End-to-end performance**
   - Parallel computation, high utilization, high data bandwidth
   - Support minibatch sizes down to as low (1 if possible)
     - Useful for transactional inference and extremely scaled training use cases.
2. **DL model accuracy**
   - DL operations require a mix of various precisions (fp32, fp16, .. INT2)
   - Design optimized for mixed-precision processing engines
     - Support for higher precision reduces efficiency of low-precision hardware
3. **Power efficiency**
   - It's an accelerator: application power should be dominated by compute elements
4. **Flexibility and programmability**
   - Support dataflow diversity and development of future algorithms
   - Architecturally maximize on-chip reuse: Access to data is as important as compute
5. **Scalability**
   - Single-core vs. Multi-core approach
   - Effective core-to-core and chip-to-chip communication

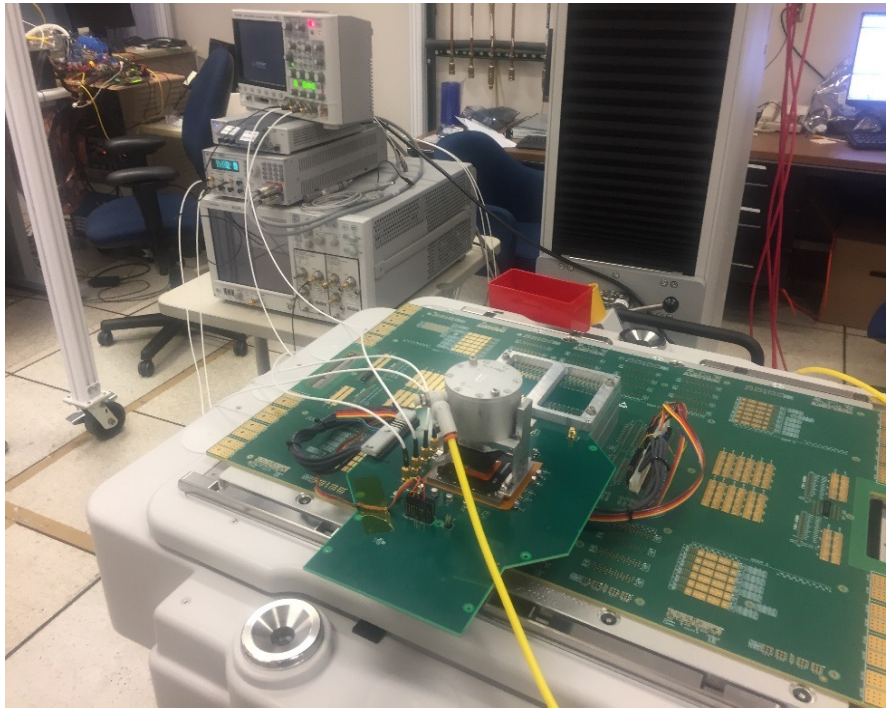
Workload Profiling

- **Op count is dominated by matrix operations and a small set of other functions**
  - Convolution/Matrix multiplication
  - Vector operations: Point-wise functions with/without reduction
- **All functions are highly parallelizable**



RaPiD: 14nm 1.5 GHz DL accelerator core

- **Many highly tuned fp pipelines and high bandwidth throughout [Performance]**
  - Customized dataflow architectures
  - Algorithm/program/ISA/hardware co-designed specifically for DL workloads
- **Balanced multiple-precision support [Accuracy]**
  - Precision chosen for each computation, for training and inference
- **Simplify logic in and around compute pipelines [Power]**
  - Carefully curated ISAs
  - Streamlined control logic
  - PEs use > 80% of power
- **ISA-accessible communications network [Programmability/Scalability]**

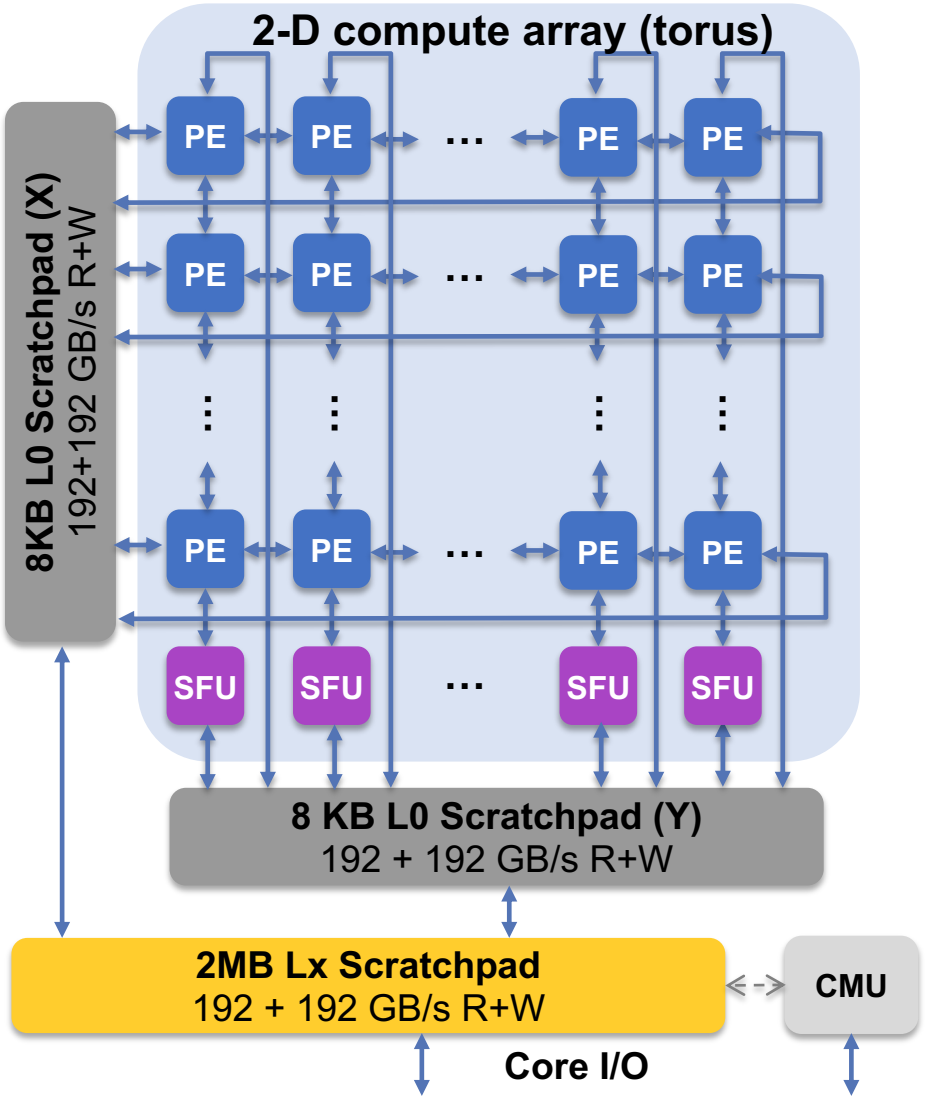


- Peak performance of 1.5 TFLOPS fp16, 12 TOPS ternary and 25 TOPS binary
- Sustained utilization >90% on multiple neural-network topologies
- Core in+out bandwidth of 96+96 GB/s for scalability

A scalable multi-teraops deep learning processor core for AI training and inference, Fleischer et al, VLSI 2018.

12

RaPiD Core Microarchitecture

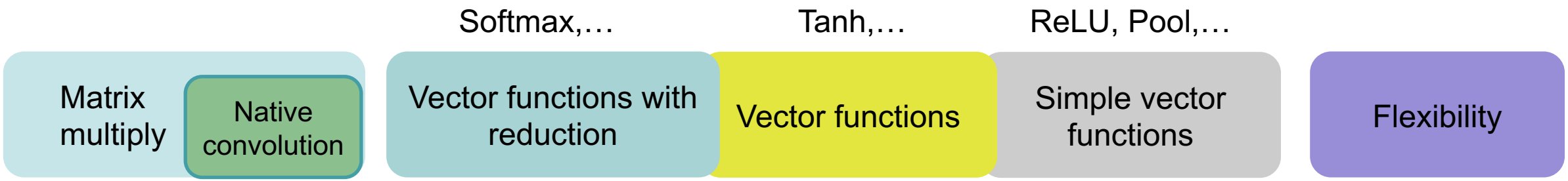


- **Dataflow with scratchpad hierarchy**
- Customized dataflow architecture – **hybrid SIMD-Dataflow** vs. traditional Dataflow
- **Reduced-precision** based PEs (Processing Elements) for ***matrix / convolution ops***
  - Support for precisions down to 2-bits (FP16 / FP8 / INT4 / INT2)
  - Minimum accumulation precision (HW chunking techniques for ALUs)
- **Limited FP32 SFUs (Special Function Units)** for ***vector (linear/non-linear) ops***
  - Needed primarily for softmax, batchnorm and axpy for training
- **Directly-addressable multi-level scratchpads** (Software managed) for high utilization (core efficiency)
- Double-buffering in Lx and high bandwidth between Lx-L0-compute

A scalable multi-teraops deep learning processor core for AI training and inference, Fleischer et al, VLSI 2018.

ISA features

Features to support algorithm variations and future development



FP16 dataflow

FMA operands from data flowing H/V and in local regs

Special-purpose data movement ops

Native support for reduction

Estimate ops for FP16 and FP32

Independent matrix and vector programs

Single-instruction ReLu, etc.

Implement one algorithm with different precisions

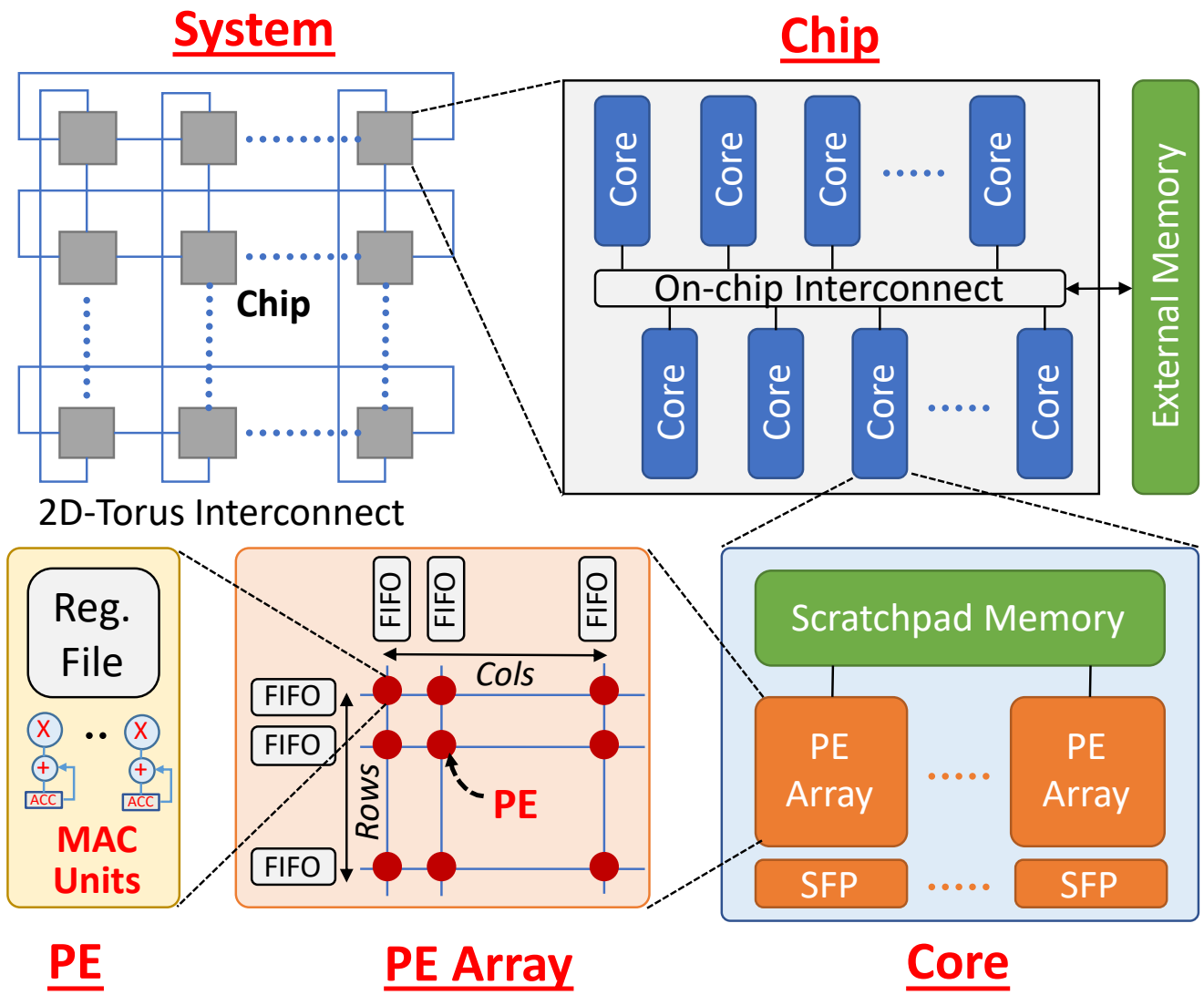
Dataflow mapping by ISA

A scalable multi-teraops deep learning processor core for AI training and inference, Fleischer et al, VLSI 2018.

Peta-Scale Deep Learning System

- **Customized server class system for training deep learning models**

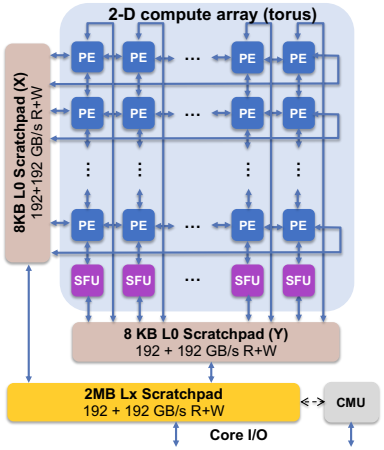
- **5 tiered hierarchy**
  - **PE**: SIMD multiply-and-accumulate engines
  - **PE array**: 2D array of Pes
  - **Core**: Multiple PE arrays with shared memory and special
  - **Chip**: Ring of cores
  - **System**: 2D torus of chips



Custom Compilers

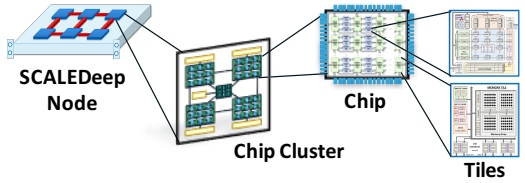
Programmable Deep Learning Accelerator Systems

Low-power IP cores ⟷ **Server-class Systems**



MAPLE [PACT10]
NeuFlow [CVPRW11]
DianNao [ASPLOS14]
Eyeriss [ISSCC16]
DANA [PACT15]
Minerva [ISCA16]
Multi-TOPS Core [VLSI18]
.....

DaDianNao[MICRO14]
NeuroCube[ISCA16]
TPU [ISCA17]
ScaleDeep [ISCA17]
.....



- **Each architecture represents a different point in energy *vs.* throughput trade-off**
  - Demonstrate impressive peak performance and processing efficiency
  - *Programmable* → Flexibility to execute DNNs of various shapes and sizes

Challenge:

- How do we program accelerators to achieve best possible ***system utilization*** for any given DNN?
- How do we achieve performance ***without sacrificing end-user (non-expert users) productivity***?

Programming Deep Learning Accelerators

- **DNNs are static dataflow graphs**
  - No data dependent execution paths, irregular memory access patterns etc.
  - Possible to define a space of mapping configurations and identify the best configuration *offline*
  - Performance estimation to reasonable accuracy through *analytical* analysis

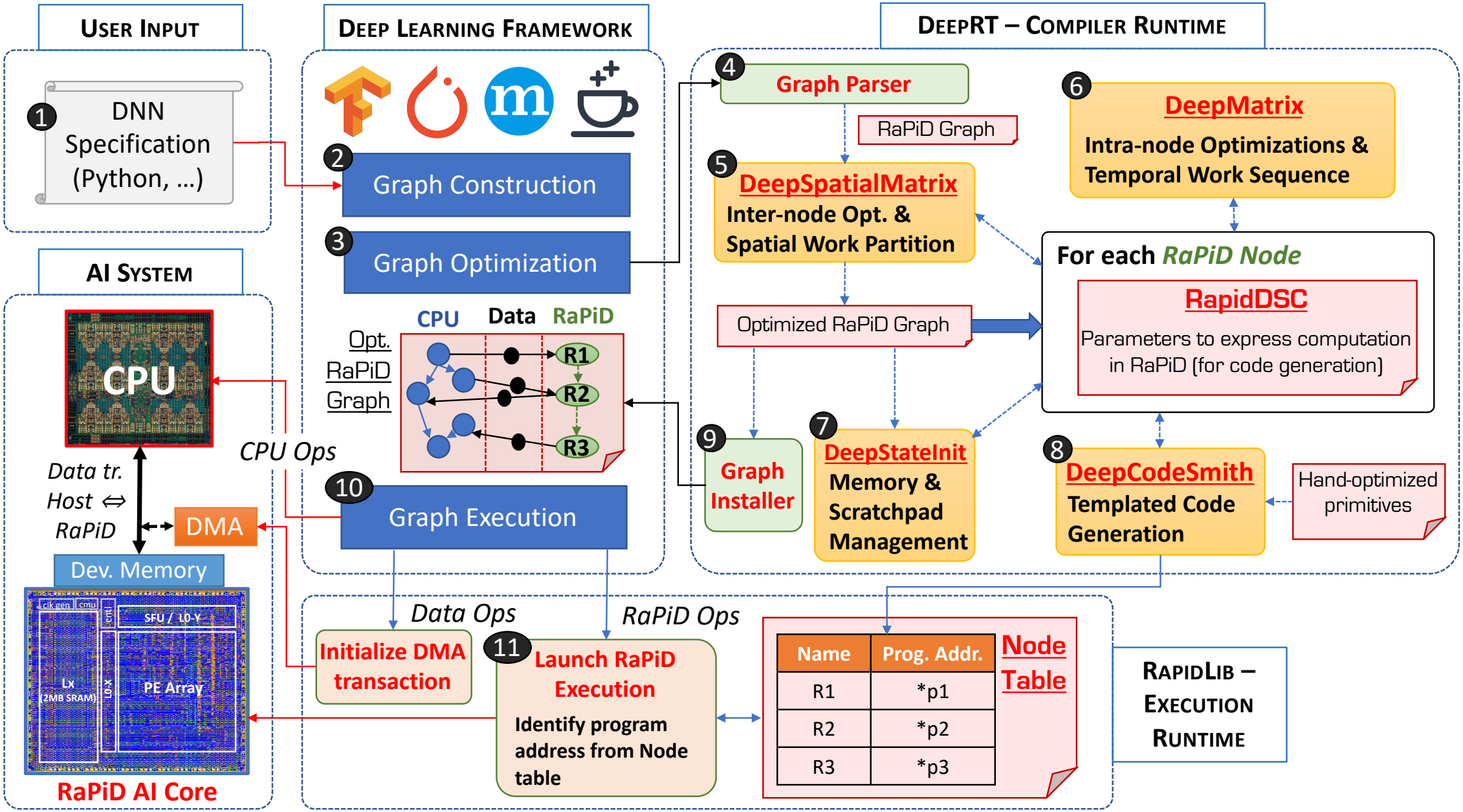
- **DNN functionality can be expressed using small set (tens) of primitives**
  - For example, VGG11 network contains **>10 billion** scalar ops, but expressible with **6** functions: *Convolution, Matrix-multiplication, ReLU, Max pooling, Softmax* & *Bias-add*
  - Complexity of how each function is optimally realized is hidden behind library/API calls

- **But, significant heterogeneity in shape and size of each data-structure, which makes each operation computationally unique**

Layer	CONV1	CONV2	…	CONV3_2	CONV4_1	…	CONV5_2	…	FC_7
Feature size	64x 224x224	128x 112x112		256x 56x56	512x 28x28		512x 14x14		4096x 1x1
Ops/By	25.78	372.58		842.51	519.06		180.63		1.00

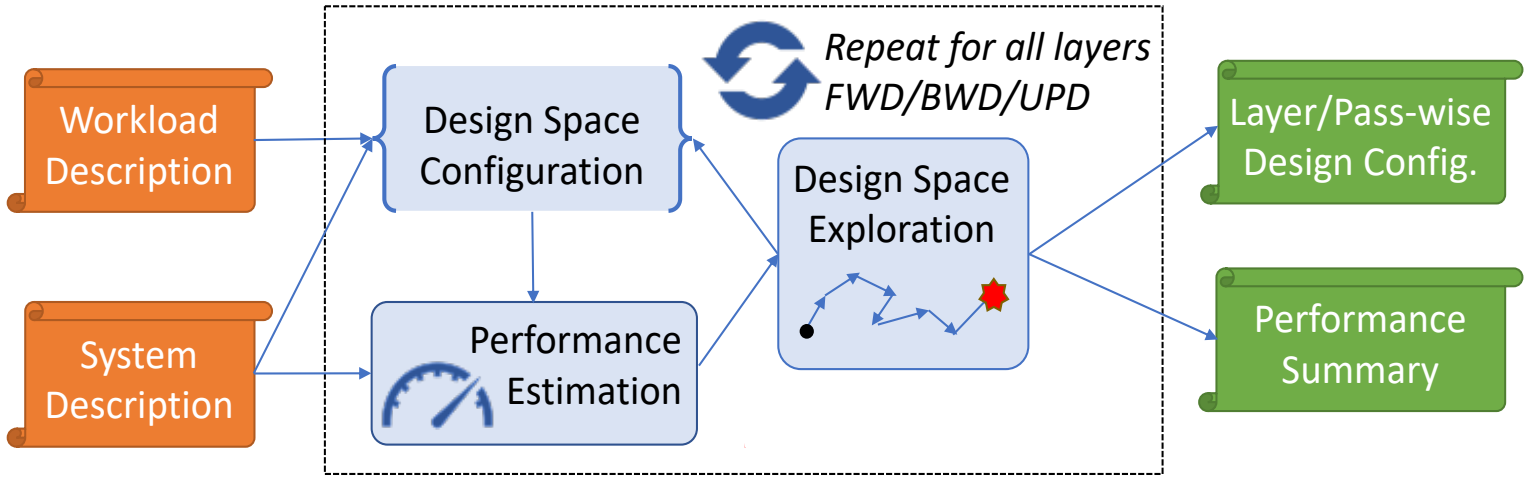
  - **Insight**: Each layer/op needs to be *programmed differently*

DeepTools: Software Stack for AI



DeepSpatialMatrix (DSM) +DeepMatrix (DM) : Mapping DNNs on Accelerators

- **DSM+DM: A systematic method to map DNNs on to any given accelerator system**
  - How the **compute needs to be partitioned** to across processing elements?
  - How much **data to stage** in each memory, respecting capacity constraints?
  - How the **data movement** needs to be orchestrated, given bandwidth limitations?
- **Key steps in DeepSpatialMatrix:**
  - Defines a "**design space configuration**"
    - Hierarchical workload mapping across chips, cores, and PE arrays
  - Uses analytical model to **estimate performance** of a given configuration
    - Includes cost for every data-transfer, compute operation at each level
  - Includes a **design space exploration** methodology to identify the performance optimal configuration



Design Space Characterization (RapidDsc)

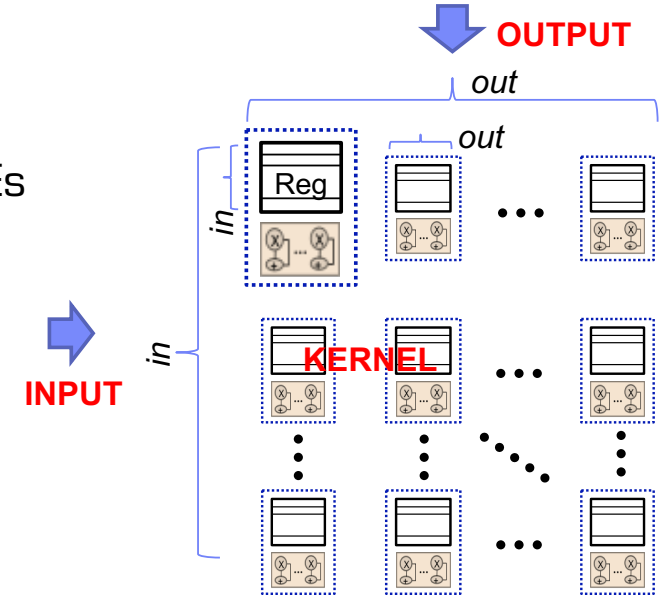
- **Spatial Work Division: Defines the work division across cores and chips**
  - Also defines how data is organized in memory and scratchpad of different cores
- **Dataflow: Orchestrate compute within PE array**
  - Data-structure/workload dimension mapped along rows, columns and held stationery in register file
  - Constrains for valid dataflows
- **Given the dataflow, the overall computation can be expressed as a nested sequence of loops**
- **Tile Sizes: Defines limits for how data-structures are chunked across memory hierarchy levels**
- **Loop Order: Determines the order in which each data-structure is traversed**
- **Data-transfer Location: Capacity and reuse of each data-structure at each level of memory**

[~100 Parameters]

attr N = <*dim tuple*>	*Total workload*
attr ChipD = <*dim tuple*> ***attr*** CoreD = <*dim tuple*>	*Spatial work division across chips and cores*
attr P = <*dim tuple*>	*Work fed to PE array*
attr B = <*dim tuple*> ***attr*** T = <*dim tuple*>	*Data-staging params for 2-level data tiling*
attr loopOrder = <*list* {stage,dim}>	*Seq. of nested loops: #loop stages * #dims*
attr dataStructures = <*list* { ***attr*** layout = <*dim tuple*> ***attr*** dataTransfers = <*list* {*src,dst,type,loopLoc*}> }>	*Info ∀ datastructure: Memory layout, data-transfers and their source, dst., and location within loops*
attr compPrimitives = <*list*>	*Primitive PE/SFP ops.*

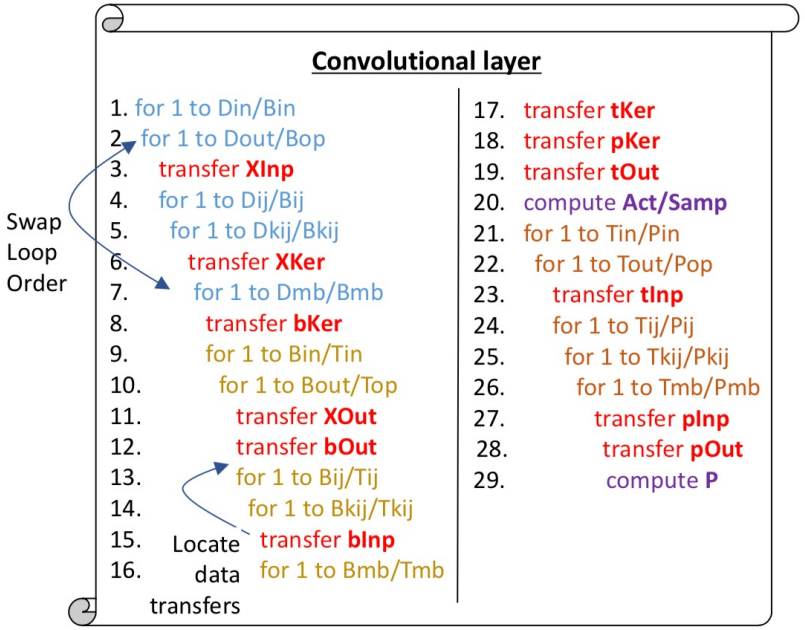
Key Optimizations

1. **Dataflow selection [Within Core, Within Node]**
   - The **direction** we flow the elements of each *data-structure* (INP, OUT, KER) to PEs
   - The **dimension** of that data-structure that is *spatially* mapped
   - Input vs. Output vs. Weight stationery....



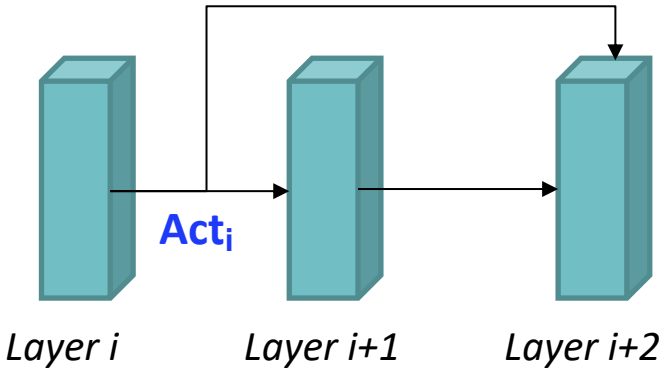
2. **Temporal Work Sequence [Within Core, Within Node]**
   - Define loop structures and tile sizes
   - Balance computation with data fetch cost
   - Identify critical data structures and maximize its reuse

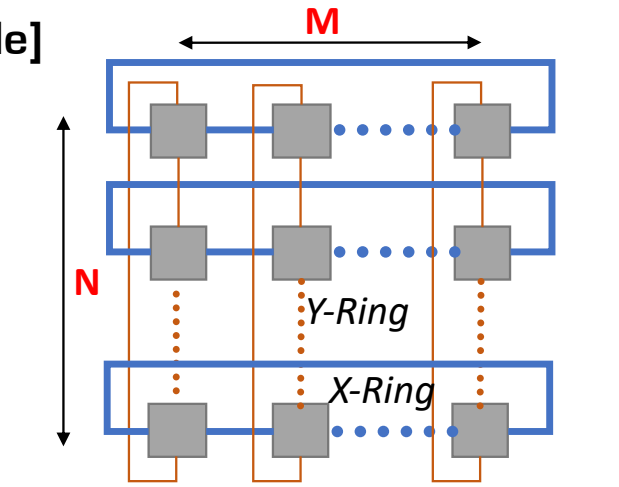
3. **Operation fusion [Within Core, Across Nodes]**
   - Fuse successive operations in the computation graph
   - Eliminates access to memory
   - Eg. Convolution with ReLU and Pooling.



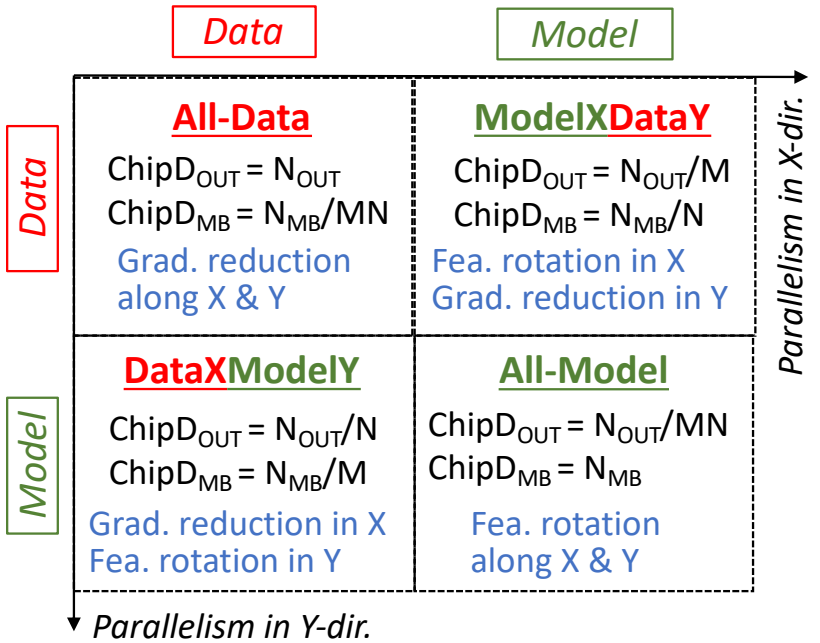
Key Optimizations

4. **Inter-layer Memory Reuse [Across Cores, Across Nodes]**
   - Hold output of a node in local scratchpad anticipating reuse in future nodes
   - Given finite on-chip capacity, which data-structures to hold and what is its impact on overall performance?
5. **Dynamic Spatial Minibatching [Across Cores, Across Nodes]**
   - Dynamically change working set size (minibatch) layer-wise based on available on-chip capacity
6. **Type of parallelism [Across Chips, Within Node]**
   - How work is split work across chips?
   - Determines communication between chips
   - Data vs. Model vs. hybrid parallelisms
7. **Timestep Pipelining [Across Chips, Across Nodes]**
   - Map multiple layers spatially across the system and execute them in pipelined fashion
   - Applies to LSTM and seq-to-seq models with a timestep dimension





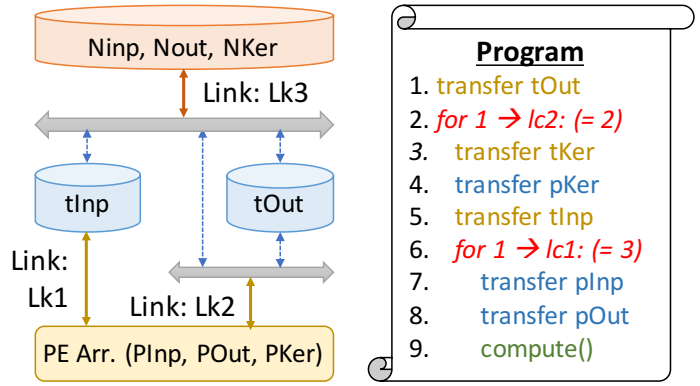
2D-tours w. asymmetric bandwidth: **X-ring BW** > **Y-ring BW**



Configuration Estimation

- **A waveform based approach to compute execution cycles for a given RapidDsc**
  - Explicitly accounts for each compute iteration, overlapped and visible data-transfers
  - Components: Compute time, overlapped data-transfer time, non-overlapped data transfer time, auxiliary compute time, pipe bubbles

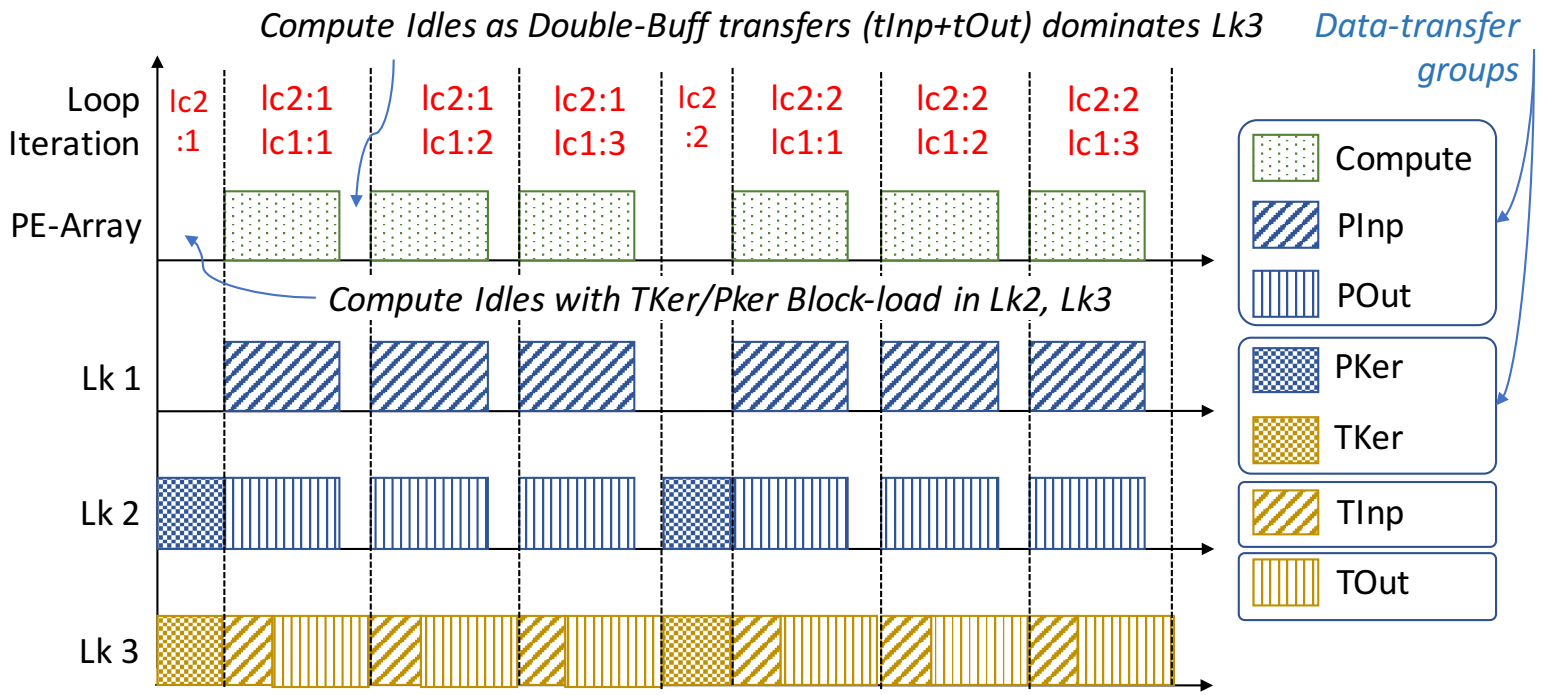
Total Execution Time =
MAX (Compute time, CommOverlap time)
+ CommNonOverlap time
+ Aux ComputeTime
+ PipeBubble



Program

1. transfer tOut
2. *for 1 → lc2: (= 2)*
3. transfer tKer
4. transfer pKer
5. transfer tInp
6. *for 1 → lc1: (= 3)*
7. transfer pInp
8. transfer pOut
9. compute()

Trans.	Type	Link
pInp	Stream.	Lk1
POut	Stream	Lk2
pKer	Blk-Load	Lk2
tInp	Dbl-Buff.	Lk3
tOut	Dbl-Buff.	Lk3
tKer	Stream.	Lk3



Results

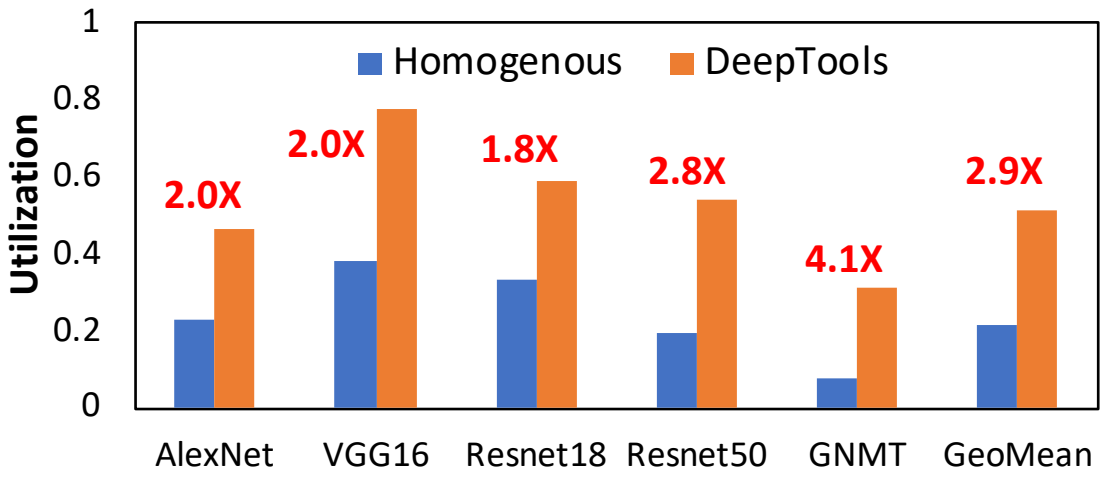
- **Architecture: System with 8 PFLOPs (half-precision) peak processing power**
  - 64 chips, 32 cores/chip, 1024 MACs/core
- **Performance model calibrated with measurements from fabricated chip at 14nm**
- **DeepTools explored over a million mapping configurations in <15 mins**

- **Benchmark: Heterogenous selection of DNNs**
  - Convolutional neural networks
    - AlexNet, VGG16: Many compute-heavy layers
    - ResNet18, Resnet50: Many lean layers that are memory-bound
  - LSTM network
    - GNMT: Very small work per layer per timestep

- **1.8X-4.1X performance improvement over hand-tuned mapping**

Bold → Baseline configuration; *{italics}* → Range used for sensitivity studies

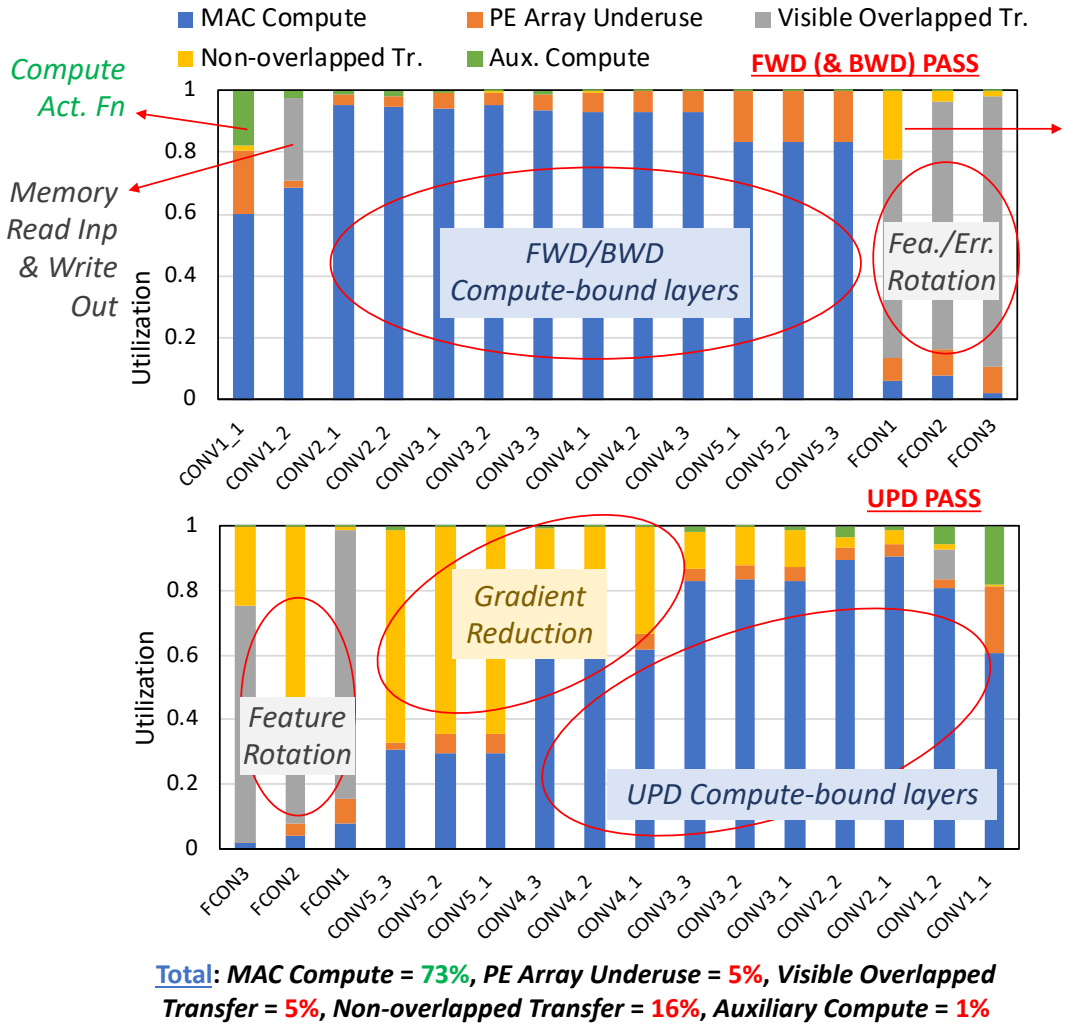
System Params.	Number of Chips			**64** *{16-256}*
	Chip Params.	Number of Cores		**32**
		Core Params.	Num. of MACs (FP16)	**1024**
			Spad Mem. (MB)	**1** *{0.5-4}*
			Spad. Bandwidth (GBps)	**128**
			Frequency (GHz)	**2**
		Chip Topology		**Ring**
		Core-to-Core Bandwidth (GBps)		**256**
		External Mem. Capacity (GB)		**8**
		External Mem. Bandwidth (GBps)		**256** @ 80% eff.
	System Topology			**2D-Torus** Chips X,Y: **4, 16***{4,64}*
	Chip-to-chip Bandwidth (GBps)			Symm. - X: **80** Y: **80**; Asymm. - X: **120** *{30-240}* Y: **40** *{10-80}*



VGG16: Layerwise study

- DSM+DM automatically modulates how each layer is mapped based on the layer's characteristics – Initial CONV vs. Mid CONV vs. Fully-connected layers
- Detailed view into performance bottlenecks for each layer

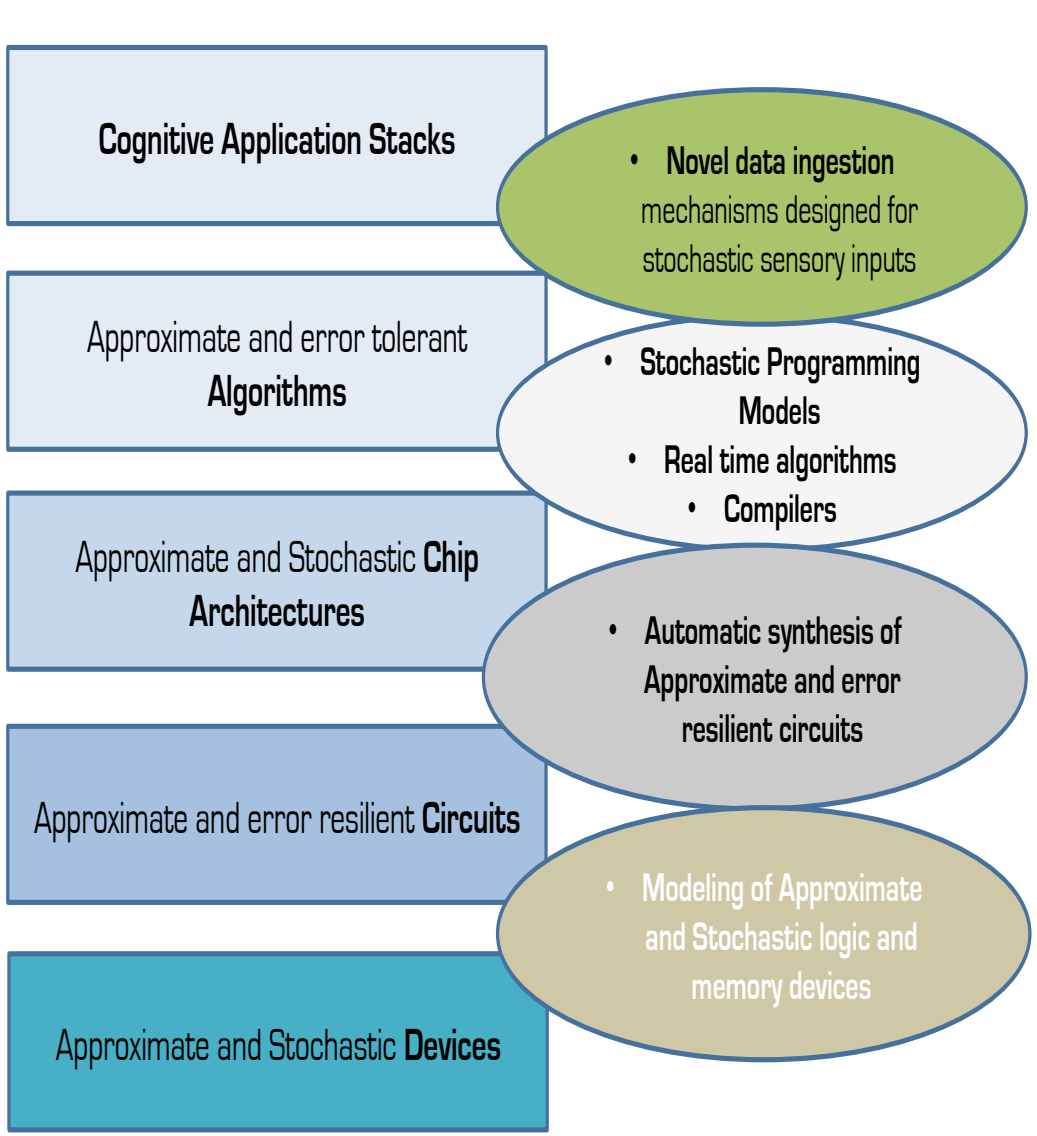
Layer FWD/BWD/UPD	Parallelism Type	CoreD Split				
		in	*out*	*ij*	*mb*	*kij*
CONV1_1	Data			32		
CONV1_2	Data			32		
CONV2_2	Data		2	8	2	
CONV3_1	Data	4	4	2		
CONV4_3	Data		8	2	2	
CONV5_1	Data	8	4			
FCON1	Model				32	
FCON2	DataX-ModelY		4		8	
FCON3	DataX-ModelY				32	



Total: *MAC Compute* = 73%, *PE Array Underuse* = 5%, *Visible Overlapped Transfer* = 5%, *Non-overlapped Transfer* = 16%, *Auxiliary Compute* = 1%

Approximate Computing

Approximate Computing Overview and Techniques



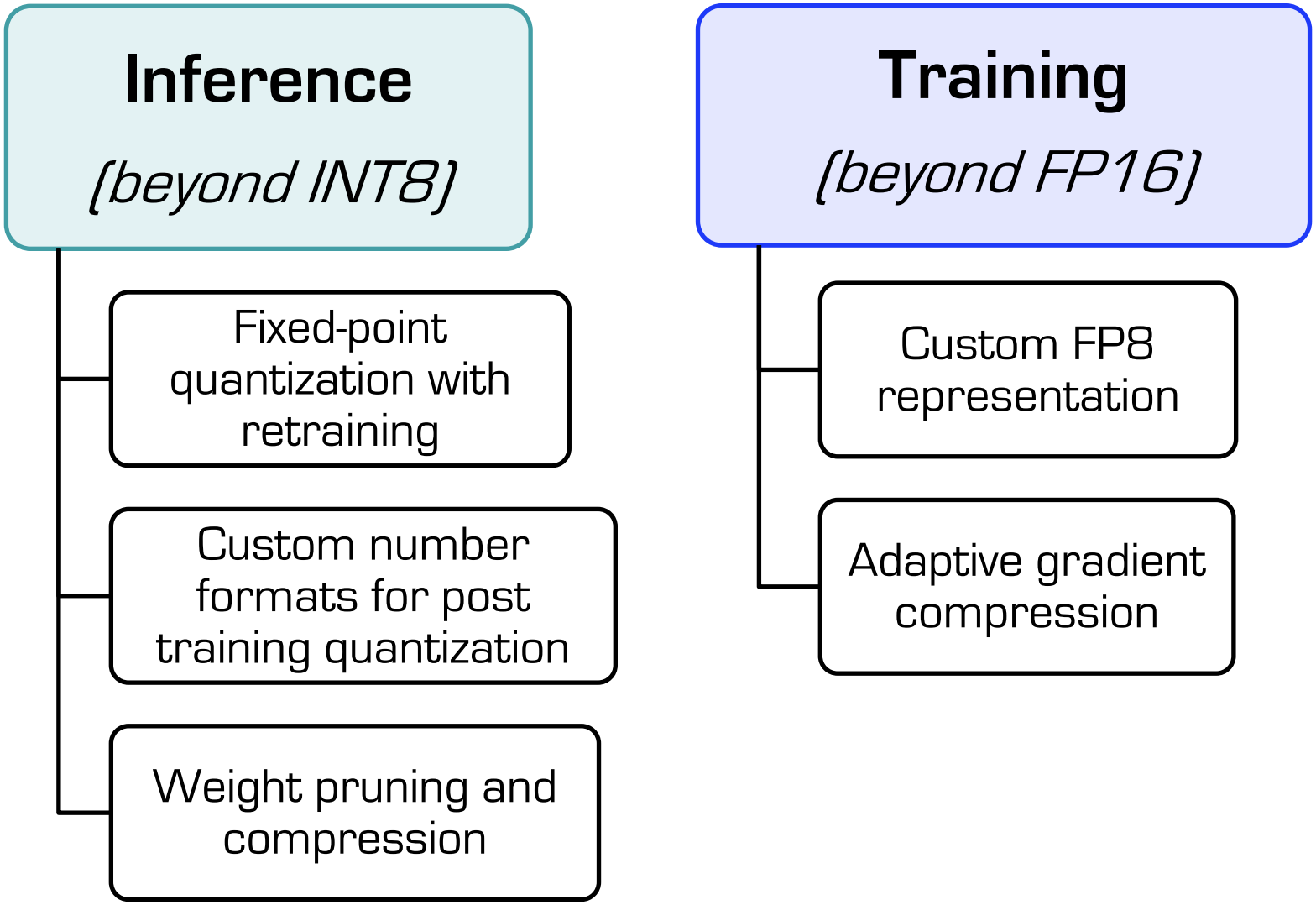
Near Term

Medium Term

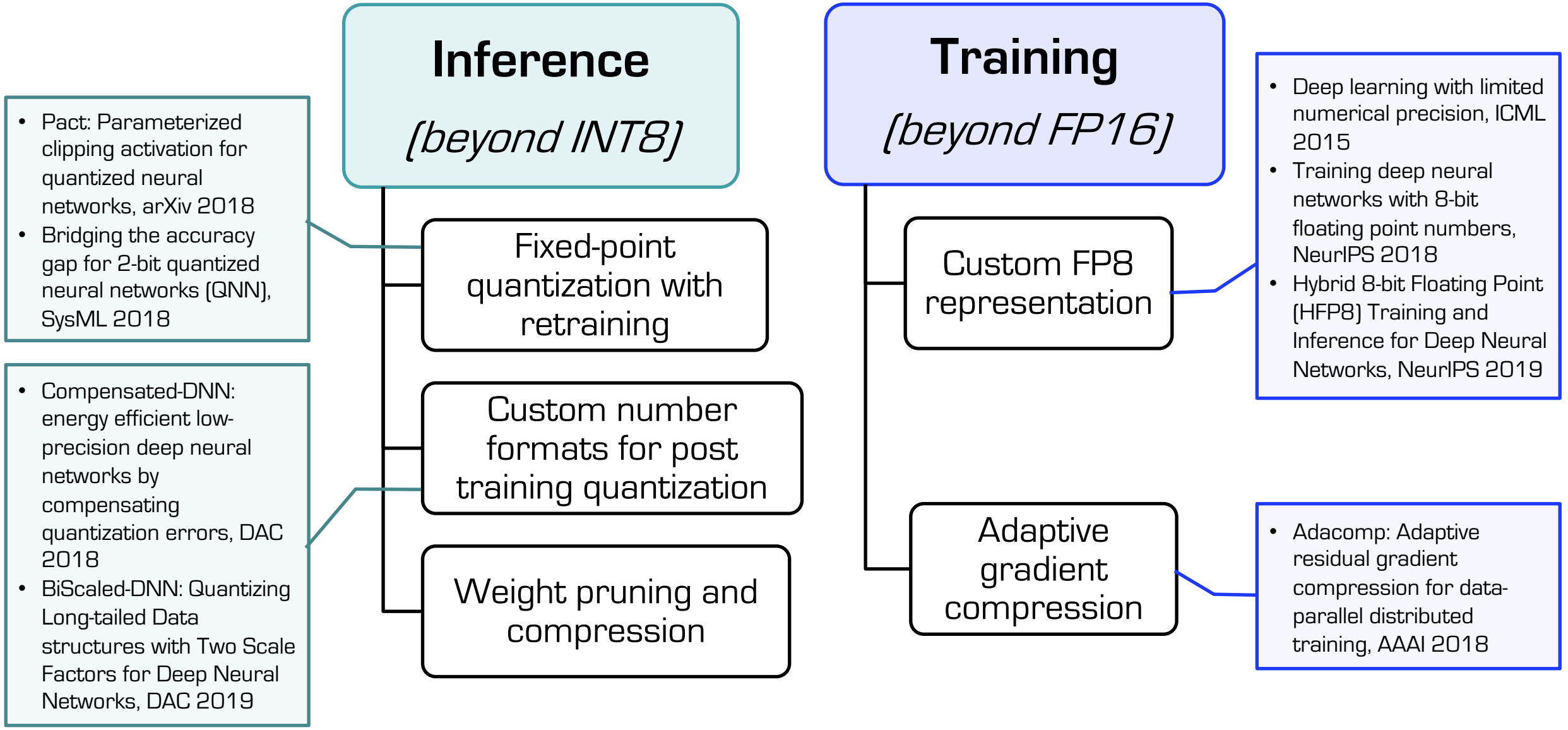
Long Term

- **Large spectrum** of cross-stack approximate computing techniques available.
- 3 Primary techniques (already) being used widely in DL
  - **Precision:**
    - **Scaled precision** for Training and Inference
    - Maximum bang for the buck (**quadratic gains in efficiency w. precision**)
  - **Compression:**
    - Lossy compression to minimize data communicated between ASICs for training.
  - **Synchronization:**
    - (Mostly) SW techniques to minimize synchronization overheads for distributed training.

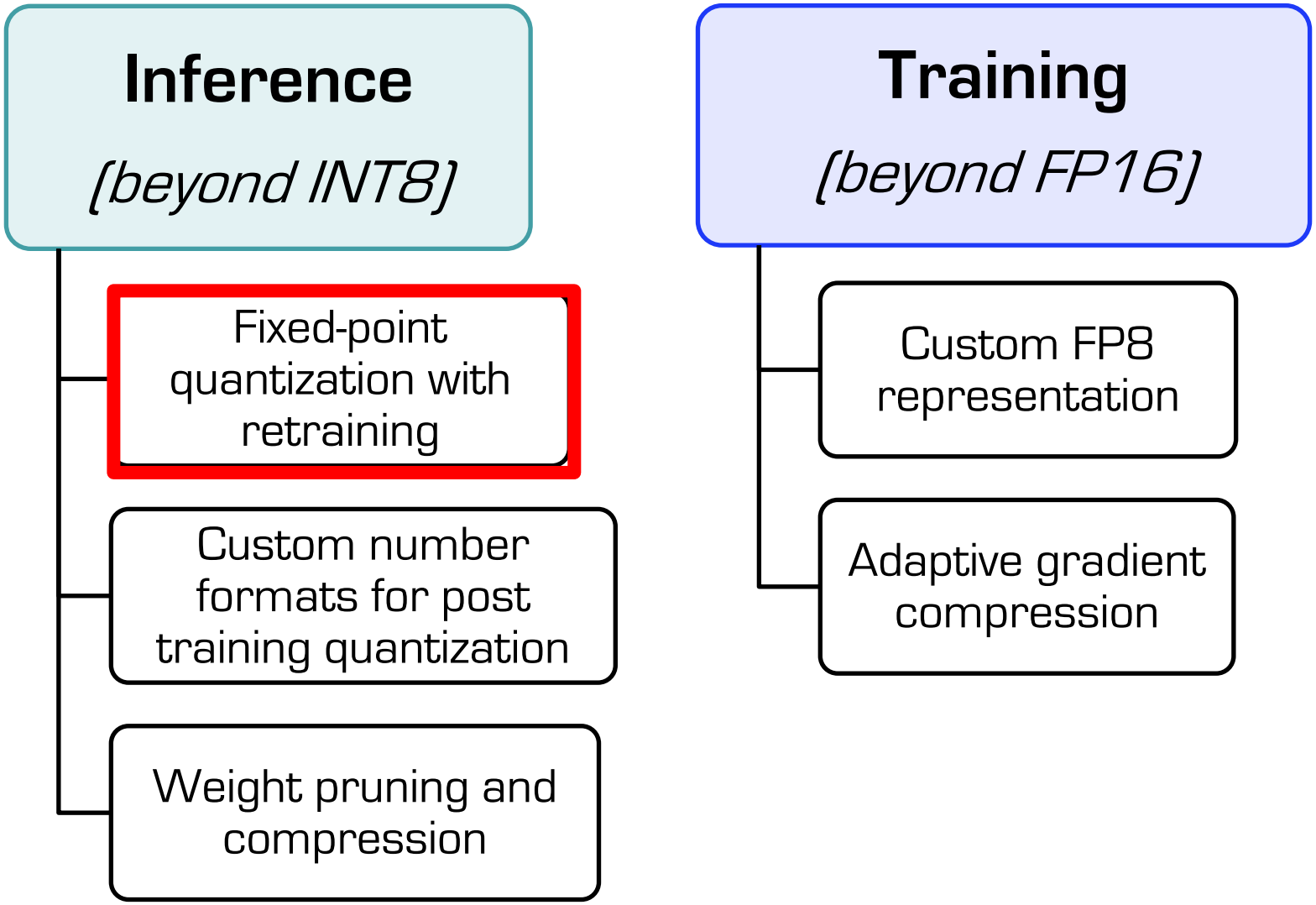
Algorithmic Approximations for AI



Algorithmic Approximations for AI

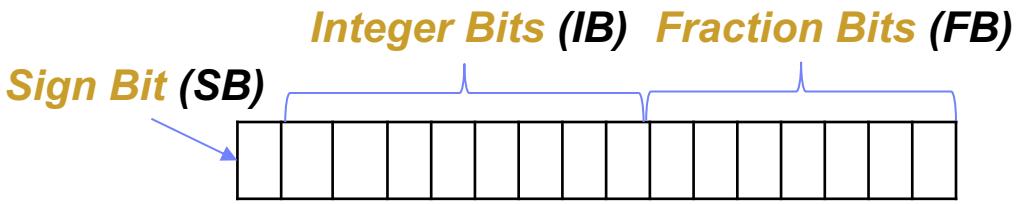


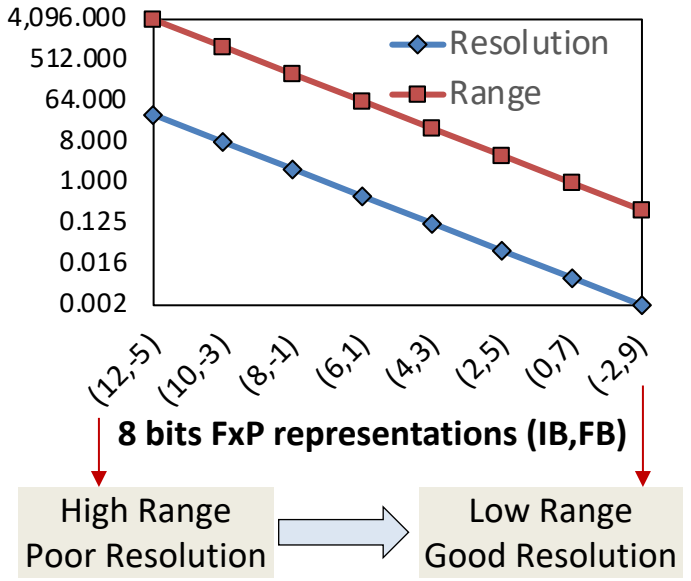
Algorithmic Approximations for AI



Quantized Deep Neural Networks

- **Implement DNN weights and activations using Fixed-Point (FxP) representation**





  - Number of IB and FB bits determine the *range* and *resolution* respectively
  - The dynamic range of weights and activations are different for each layer

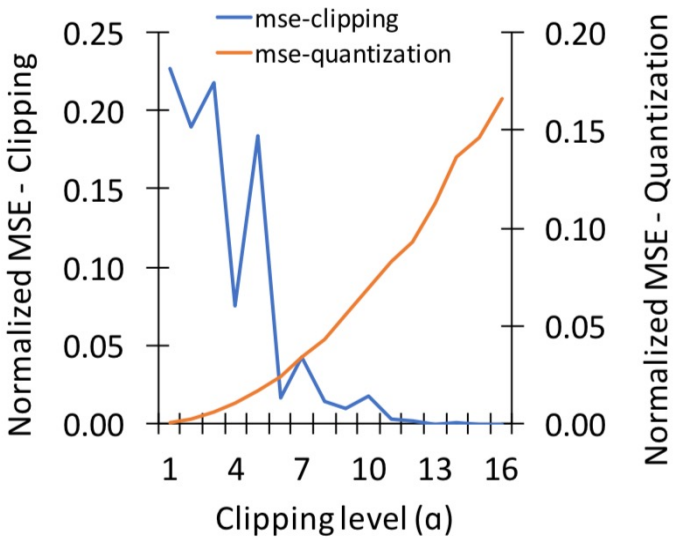
- **Advantages**:
  - Smaller ALUs ➔ lead to power and performance benefits
  - Smaller memory footprint
  - Smaller data elements ➔ increase data transfer efficiency

- **Challenge: Invariably suffer from quantization errors, which degrades accuracy**
  - Choose the right range and resolution for each data-structure
  - Retrain the network considering quantization errors

New Techniques for Inference with Hyper-Scaled Precision

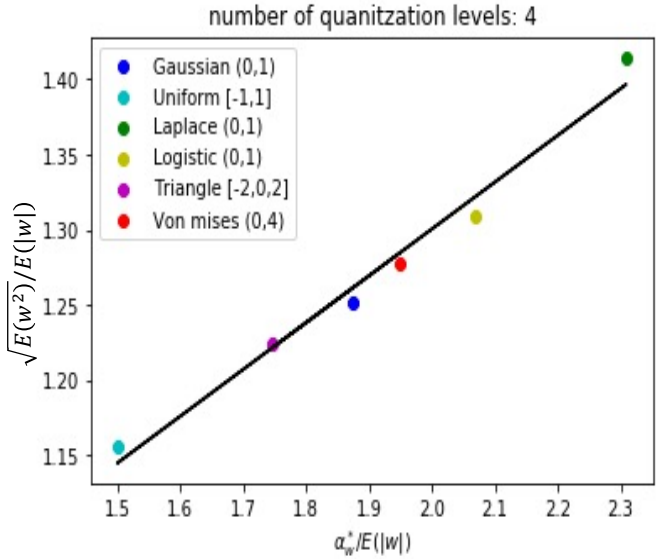
Activation Quantization
PArameterized Clipping acTivation (PACT)

– Automatic tuning of clipping level to balance clipping vs quantization error



Weight Quantization
Statistics Aware Weight Binning (SAWB)

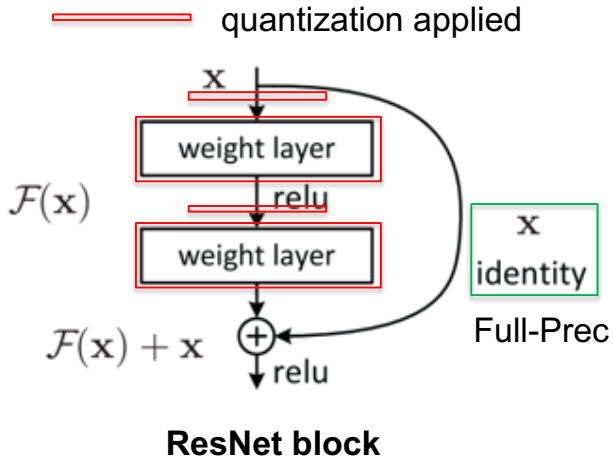
– Exploit weight statistics to better capture shape of weight



$$\alpha_w^* = c_1 \cdot \sqrt{E(w^2)} - c_2 \cdot E(|w|)$$

Quantization in the Presence of Shortcut Connections
Full-Precision Shortcut

– Enhance gradient-flow of training by not quantizing shortcut

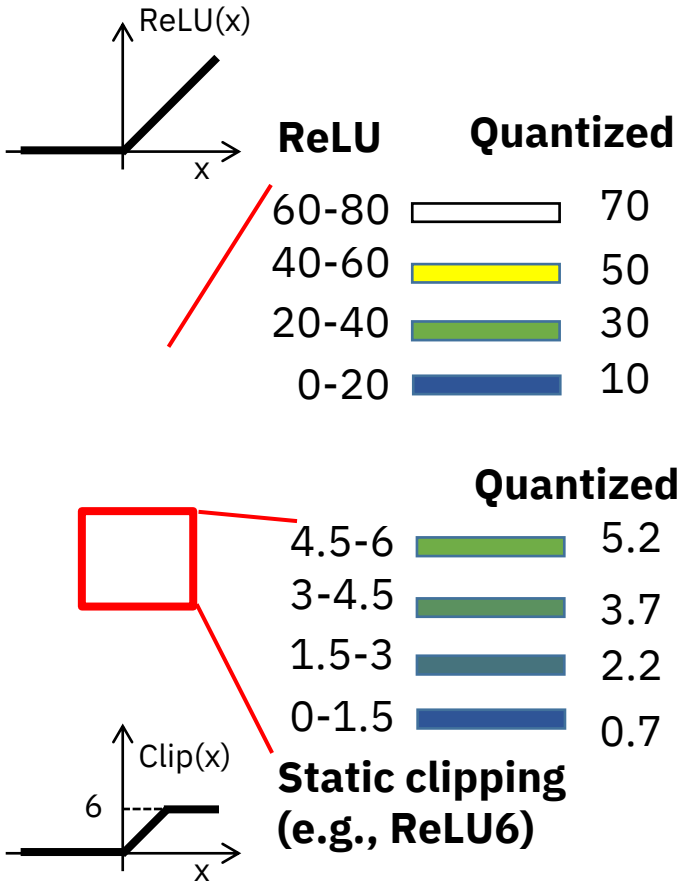


ResNet block

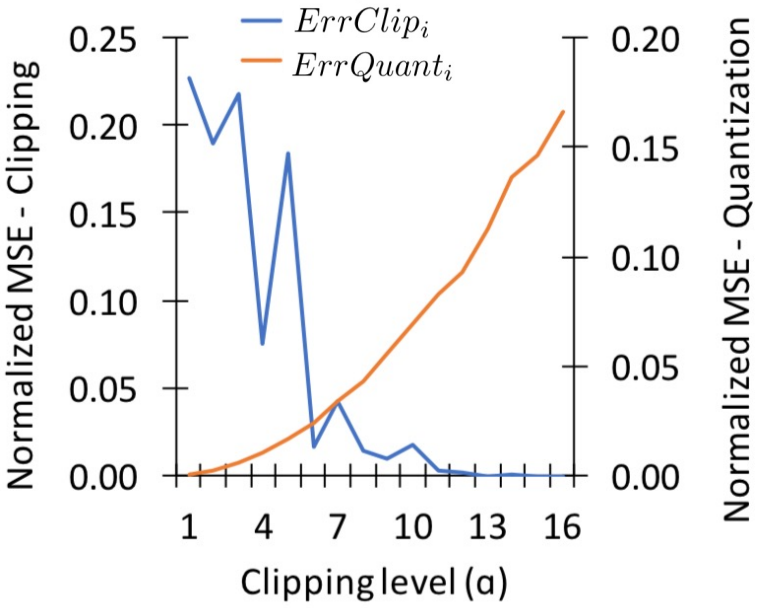
J. Choi et. al (presented at SysML 2019)

Inference : Challenges in Low Precision Activation Quantization

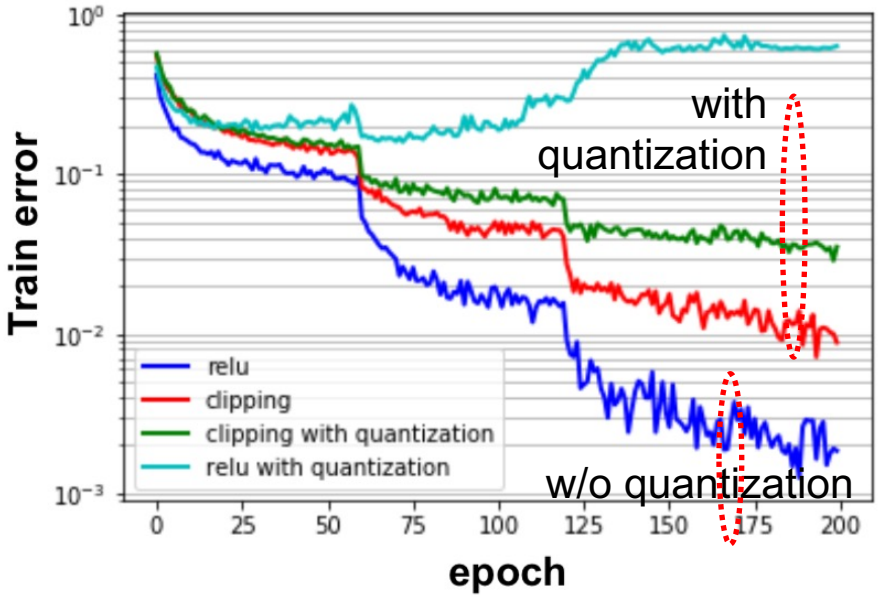
- **Trade-offs when quantizing activation**
  - **ReLU**: *Cover large dynamic range* (better convergence) ➔ **Suffer large quantization error**
  - **Static clipping**: *Clip outliers* (lower quant error) ➔ Accuracy drop due to ***diminishing gradients***



Trade-off between Quantization & Clipping Error



CIFAR10 ResNet20: **ReLU vs Clipping when quantizing activation**

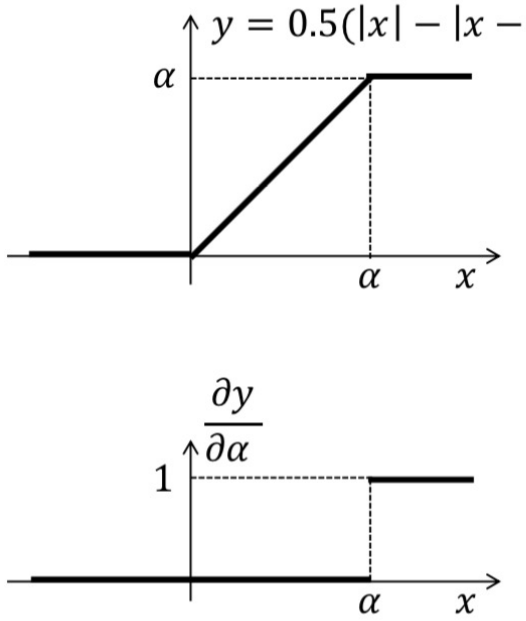


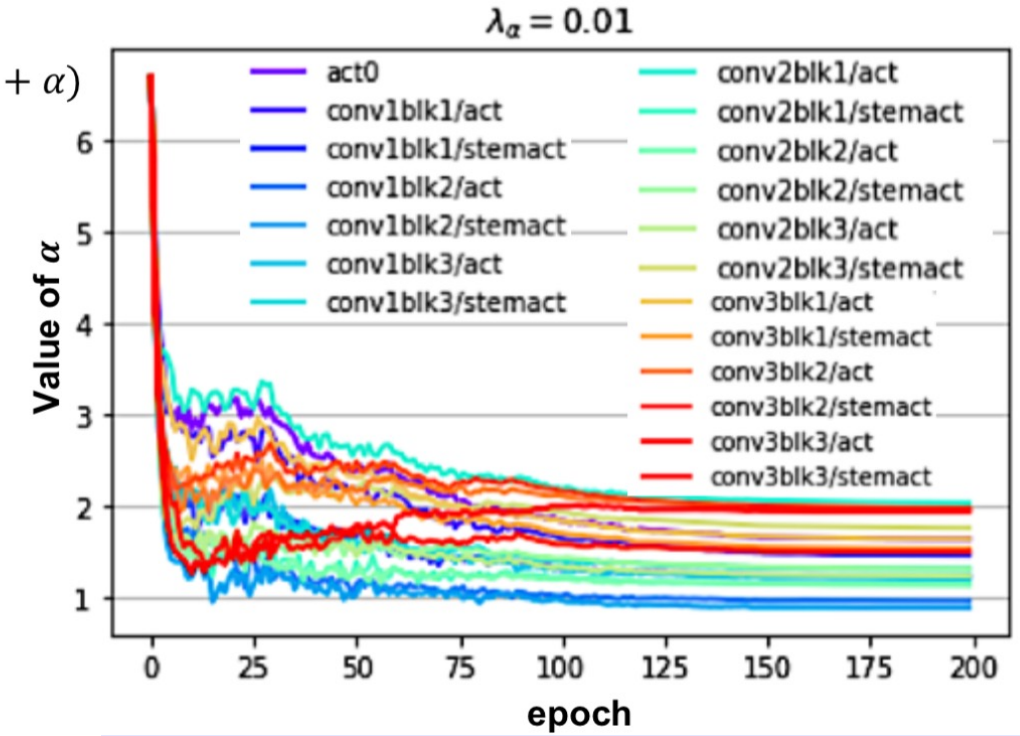
Hard to find sweet spot ➔ *Automatic tuning via training!*

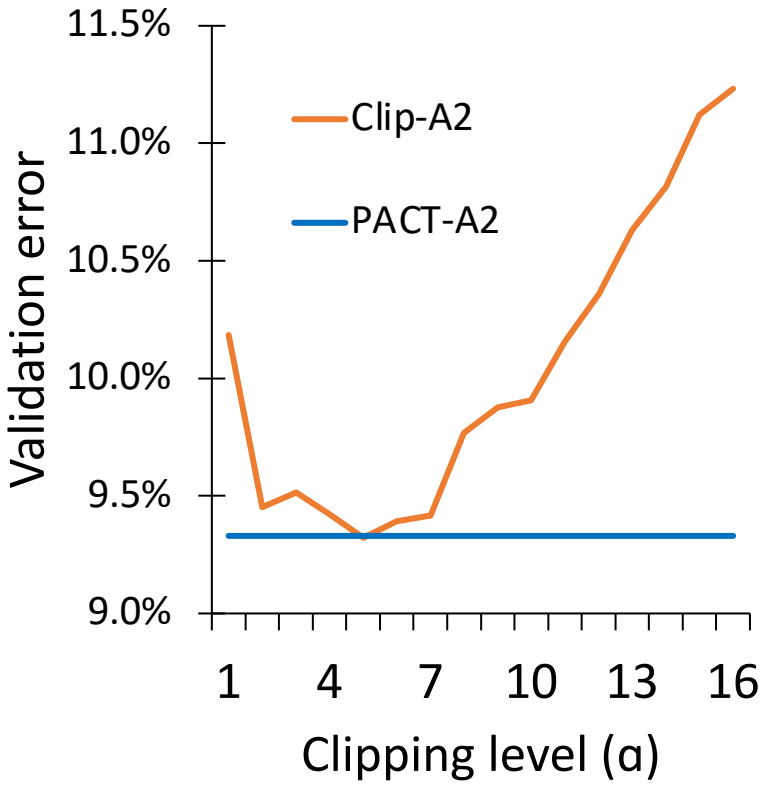
J. Choi et. al (presented at SysML 2019)

Activation Quantization : PArameterized Clipping acTivation (PACT)

- **Clipping level (= α) as a *trainable parameter* → Auto-tuned by Backprop**
  - α is *initialized with large value* to emulate ReLU in the beginning
  - L2-regularization on α → Converges to low magnitude to reduce quantization error
- **Ex: CIFAR10-ResNet20 with PACT**
  - PACT automatically finds the best clipping level without expensive sweeping over α







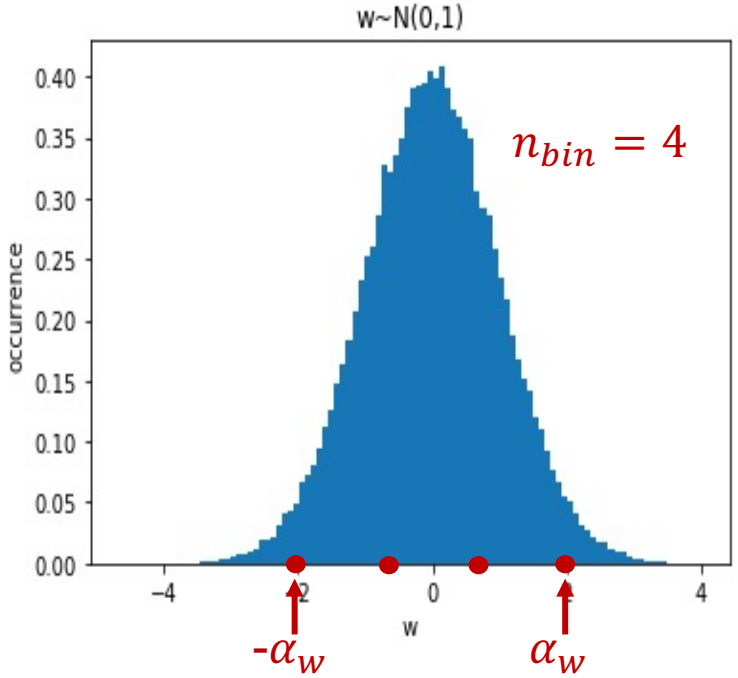
J. Choi et. al (presented at SysML 2019)

Inference : Challenges in Low Precision Weight Quantization

- **Objective: Find a good *quantization scale* (α_w)**
  - Assumption: symmetric distribution (usually true…), Uniform quantization (for simple HW)
  - Given a quantization scale, quantized weights are exclusively determined
  - Goal: find α_w that minimizes quantization error

$$\alpha_w^* = \arg\min_{\alpha_w} ||w - w_q||^2$$

- **Previous approach: Find α_w with respect to $E(|W|)$**
  - E.g., XNOR-Net, Ternary-Weight-Quant, etc.,
  - Pros: Simple (sometimes with analytic solution)
  - Cons: $E(|W|)$ is not enough to capture shape of W



Ex: XNOR-Net (Rastegari et al., 2016)

$$J(\mathbf{B}, \alpha) = \|\mathbf{W} - \alpha\mathbf{B}\|^2$$

$$\alpha^*, \mathbf{B}^* = \underset{\alpha, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha)$$

$$\alpha^* = \frac{\mathbf{W}^\top \operatorname{sign}(\mathbf{W})}{n} = \frac{\sum |\mathbf{W}_i|}{n} = \frac{1}{n}\|\mathbf{W}\|_{\ell 1}$$

Need an analytic solution that better characterizes weight distribution

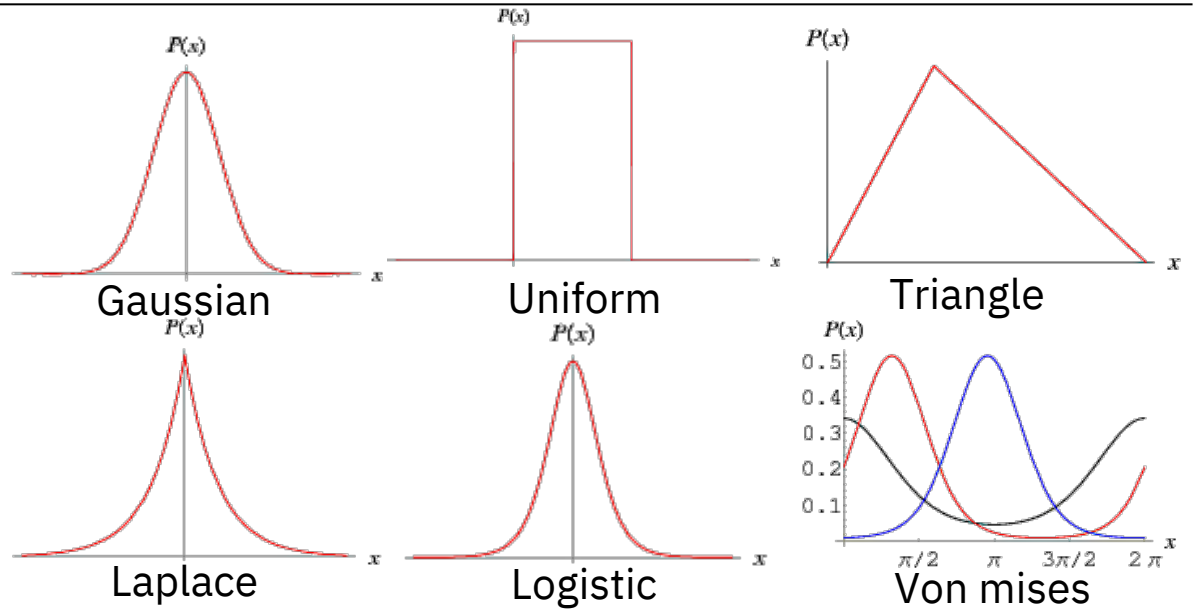
J. Choi et. al (presented at SysML 2019)

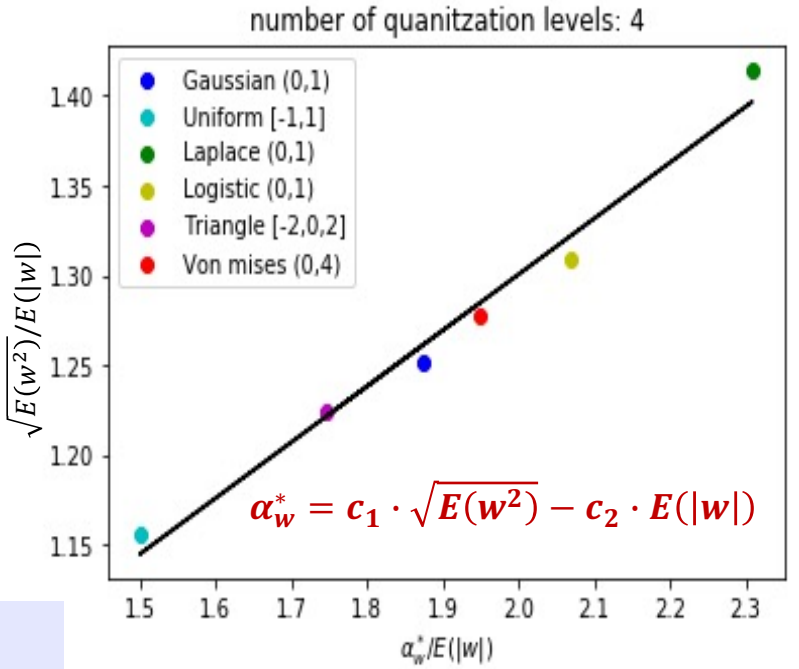
Weight Quantization : Statistics Aware Weight Binning (SAWB)

- **How to better capture shape of weight?**
  - $\boldsymbol{E(|W|)}$ captures the ***representative values***
  - $\boldsymbol{E(W^2)}$ captures the ***overall shape***
  - Use $\boldsymbol{E(|W|)}$ and $\boldsymbol{E(W^2)}$ for finding best $\boldsymbol{\alpha_w}$?

$$\boldsymbol{\alpha_w^* = c_1 \cdot \sqrt{E(w^2)} - c_2 \cdot E(|w|)}$$

- **Verification:**
  - Take 6 representative distributions with varying variance: Gaussian, Uniform, Triangle, Laplace, Logistic, Von mises
  - Sweep over $\boldsymbol{\alpha_w}$ to find one with smallest MSE(W-Wq)
  - Linear regression to find relationship among α_w^*, $E(|W|)$ and $E(W^2)$





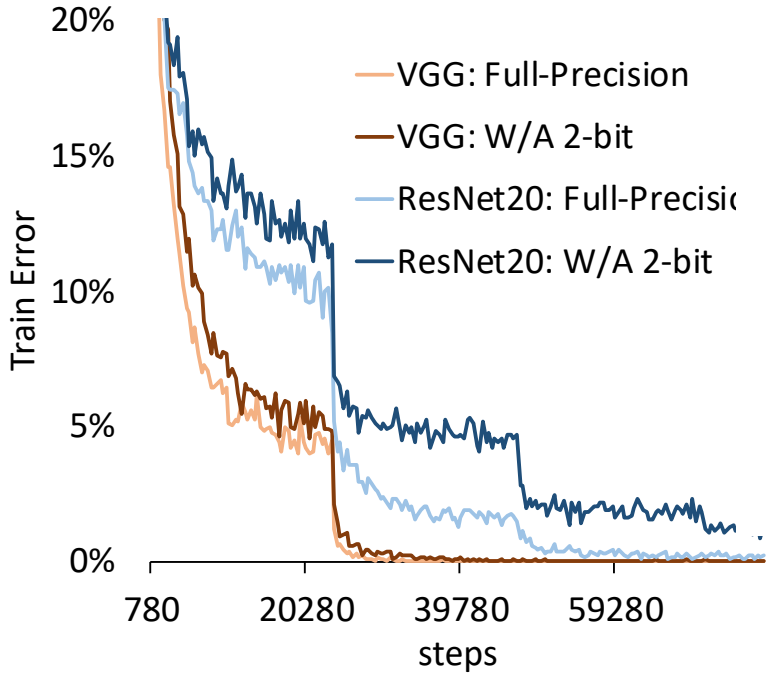
Observation: α_w^* is characterized by $E(|W|)$ and $E(W^2)$ ➔ *Analytic solution* to find the *best scale*

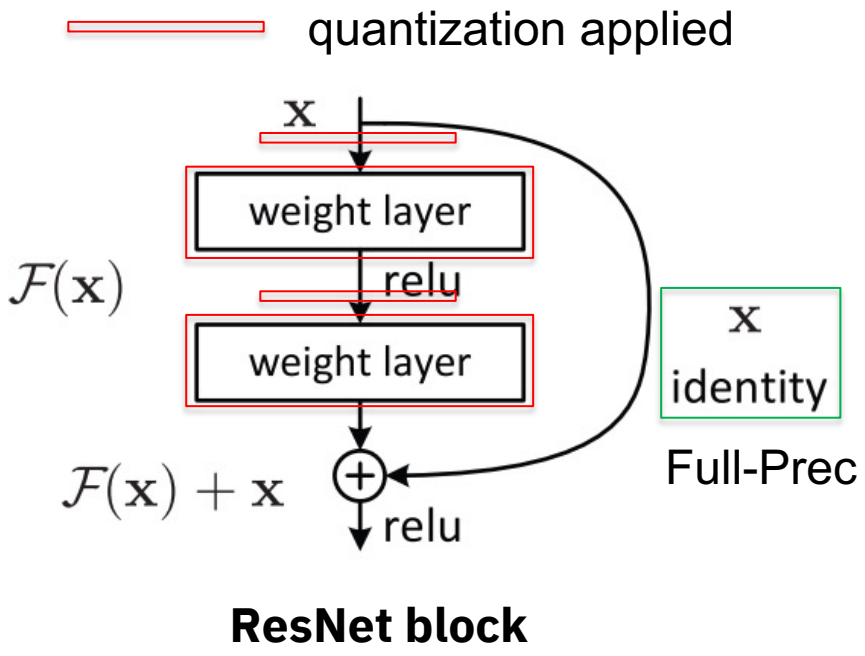
J. Choi et. al (presented at SysML 2019)

Full Precision Short-Cut for Inference (FPSC)

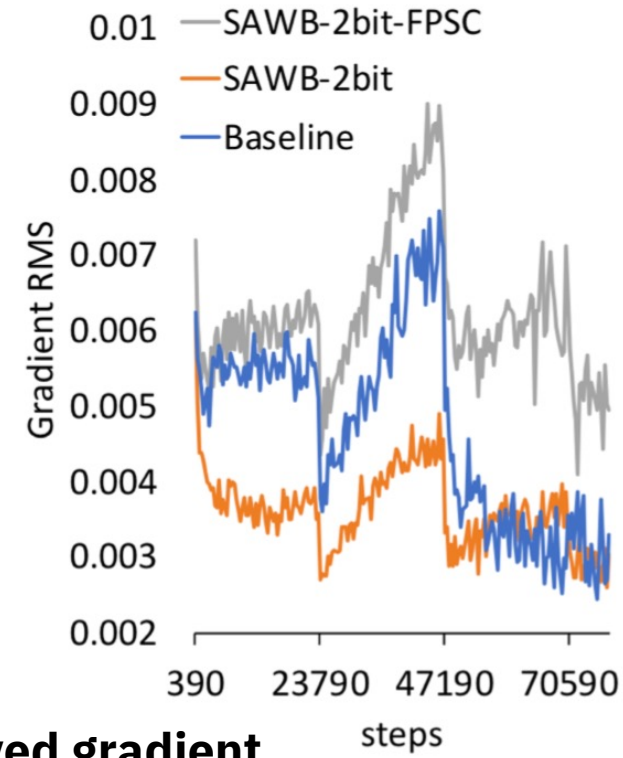
- **Observation: ResNet is more sensitive to quant-error than VGG-like networks**
  - Short-cut in ResNet helps gradients to flow → Quantization on shortcut hinders training
- **Full-Prec Short-Cut: Avoid quantization at input activation and weight in short-cut**
  - Short-cut involves small weights → Full-Prec does not harm performance (<1% ResNet18)
  - FPSC allows larger magnitude gradients → Improved gradient flow

QNN training: VGG vs ResNet





ResNet block



Improved gradient flow by FPSC

J. Choi et. al (presented at SysML 2019)

Hyper-Scaled Precision : Inference Accuracies on Models

CIFAR10 Dataset

Name	Baseline	Quantized	Degradation
	< ResNet20: W2-A32 >		
SAWB-fpsc	**91.8**	**91.6**	**0.2**
DoReFa	91.8	90.9	1.0
LQ-Nets	92.1	91.8	0.3
TWN	91.8	90.9	0.9
TTQ	91.8	91.1	0.6
	< ResNet20: W32-A2 >		
PACT-fpsc	**91.5**	**91.4**	**0.2**
DoReFa	91.5	90.1	1.4
ReLU6	91.5	91.0	0.5
	< ResNet20: W2-A2 >		
SAWB-PACT-fpsc	**91.5**	**90.8**	**0.7**
DoReFa	91.5	88.2	3.3
LQ-Nets	92.1	90.2	1.9
	< VGG: W2-A2 >		
SAWB-PACT-fpsc	**93.8**	**93.8**	**0.1**
LQ-Nets	93.8	93.5	0.3
QIP (lambda=0.5)	94.1	93.9	0.2

ImageNet Dataset

Name	Baseline	Quantized	Degradation
	< AlexNet >		
SAWB-PACT-fpsc	**58.3**	**57.2**	**1.1**
LQ-NETs	61.8	57.4	4.4
QIP (lambda=0.0)	58.1	55.7	2.4
DoReFa-Net	55.1	53.6	1.5
HWGQ	58.5	52.7	5.8
BalancedQ	57.1	55.7	1.4
WEQ	57.1	50.6	6.5
WRPN-x1	57.2	51.3	5.9
WRPN-x2	60.5	55.8	4.7
WRPN-x2, W2-A4	60.5	57.2	3.3
LearningReg	58.0	54.1	3.9
	< ResNet18 >		
SAWB-PACT-fpsc	**70.4**	**67.0**	**3.4**
LQ-NETs	70.3	64.9	5.4
QIP (lambda=0.0)	69.2	65.4	3.8
DoReFa	70.2	62.6	7.6
HWGQ	67.3	59.6	7.7
BalancedQ	68.2	59.4	8.8
LearningReg	68.1	61.7	6.4
	< ResNet50 >		
SAWB-PACT-fpsc	**76.9**	**74.2**	**2.7**
LQ-NETs	76.4	71.5	4.9
Apprentice (W2-A8)	76.2	71.5	3.4
UNIQ (W4-A8)	76.0	73.4	2.6

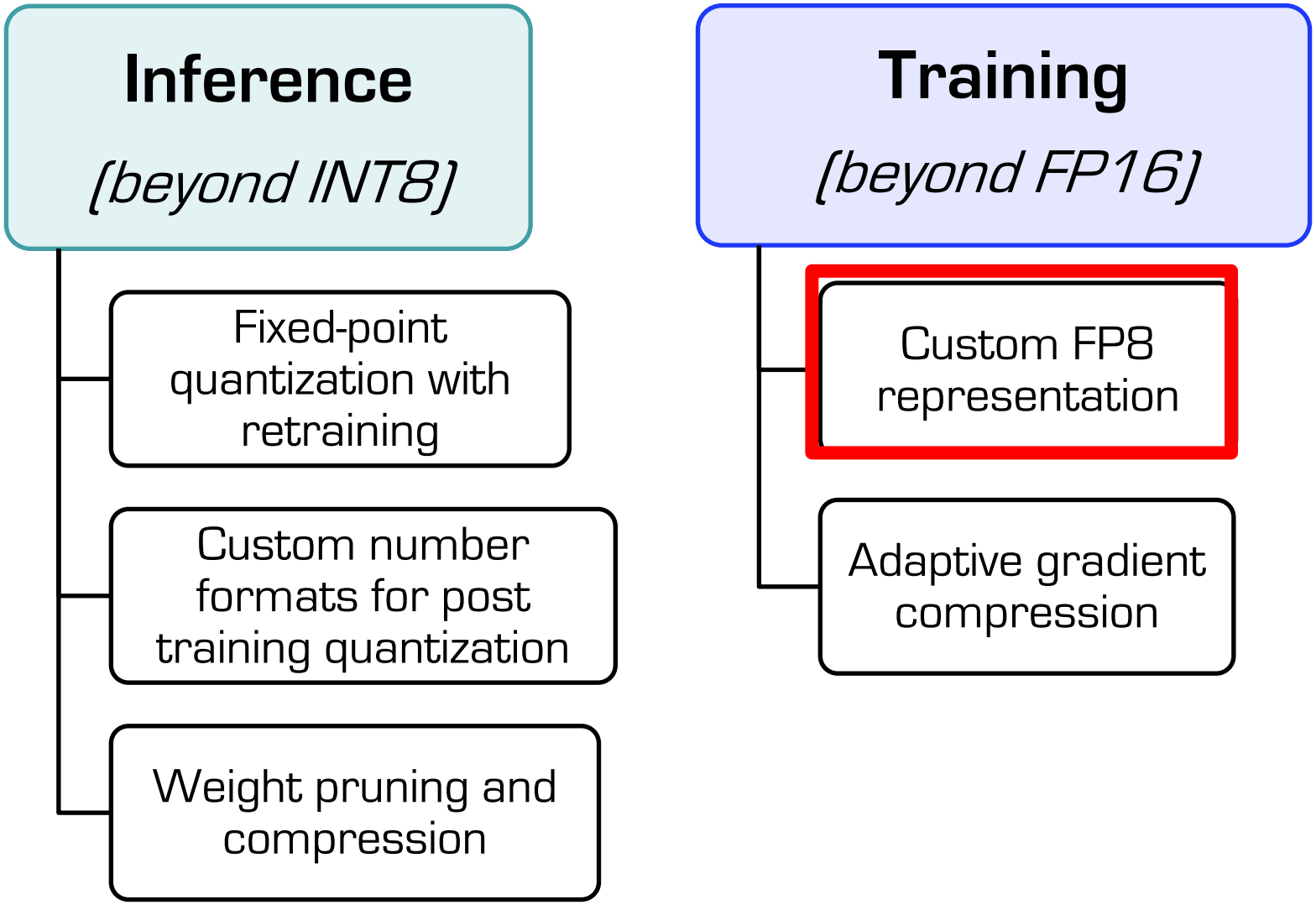
Lowest accuracy degradation

Accuracy higher than prior work

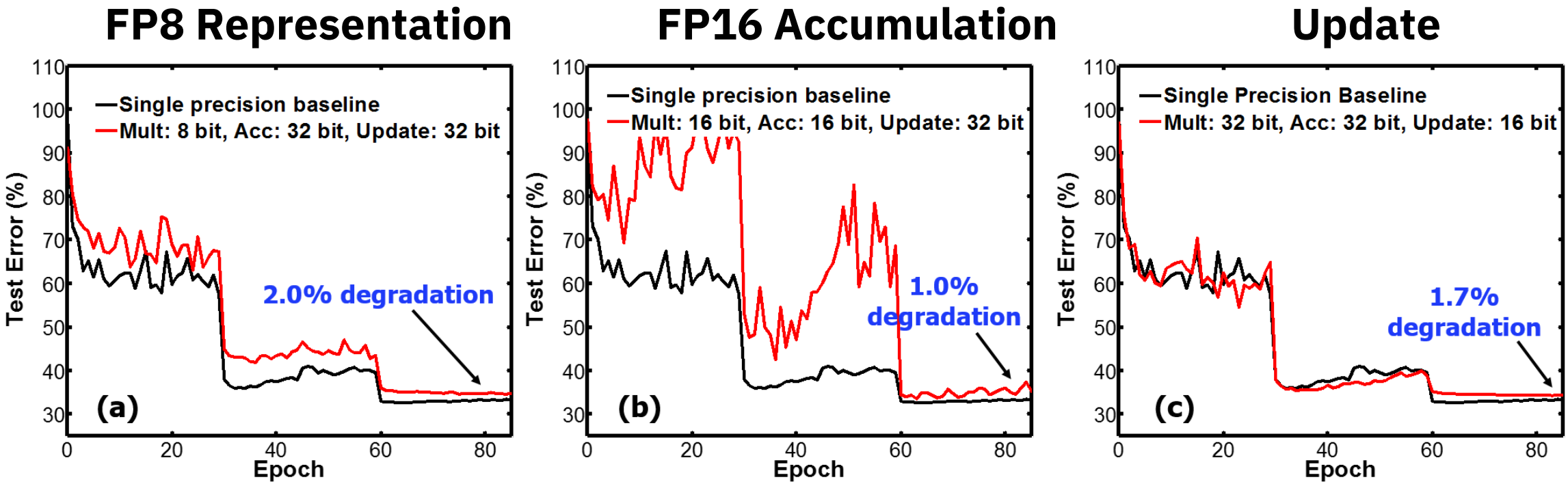
No loss of accuracy for 4-bit Inference

J. Choi et. al (presented at SysML 2019) `

Algorithmic Approximations for AI



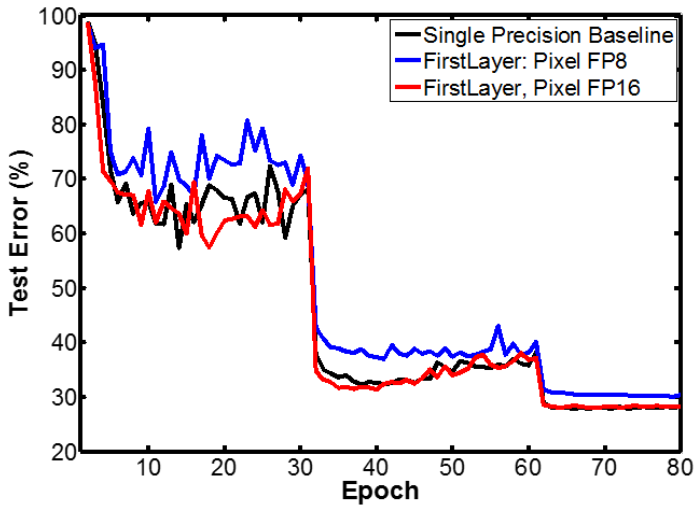
Training Challenges Beyond FP16

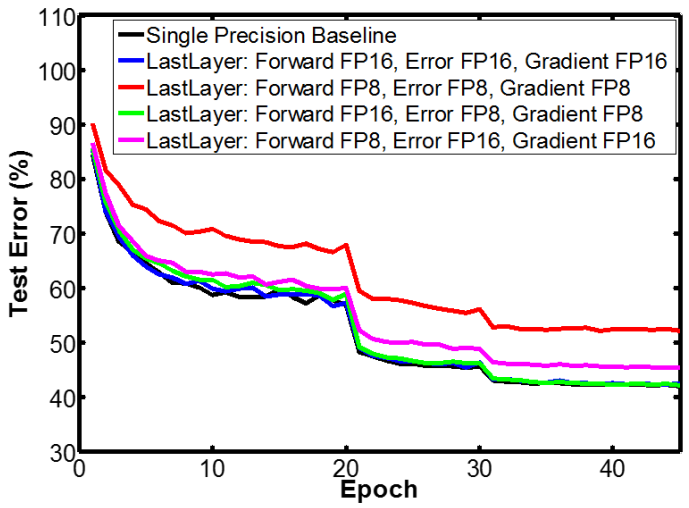


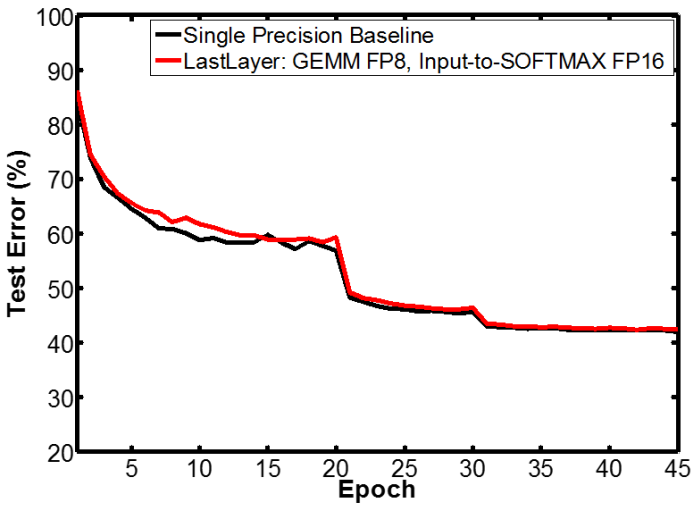
- State of the art Training systems use FP16 for data representations and FP32 for accumulations & weight updates.
- Challenge: To reduce these precisions down to FP8 for representation and FP16 for accumulation & weight updates.
- Goal: Increase training-performance by 4X over today's systems!

N. Wang et. al (presented at NeurIPS 2018)

FP8 Training : FP8 Data Representation







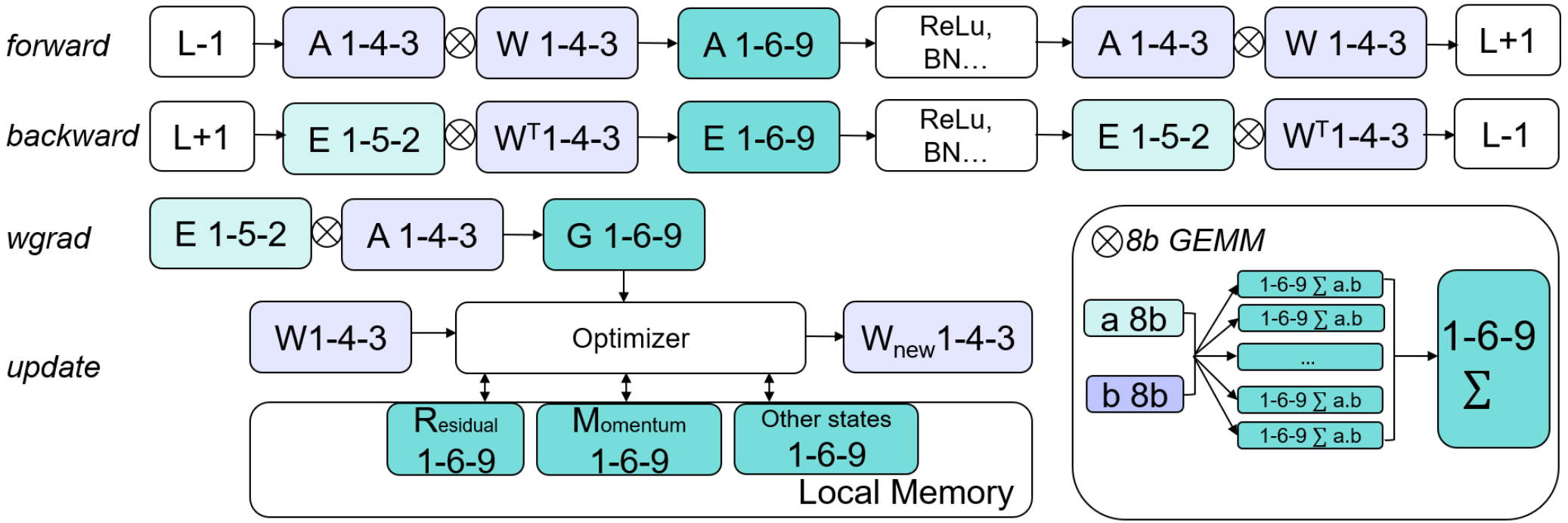
- First layer and last layers are very sensitive to lower-precision (FP8) – especially for image processing (keep them in FP16).

- Input to softmax is sensitive to low-precision : possible to keep last layer in FP8 but outputs need to be preserved in FP16.

N. Wang et. al (NeurIPS 2018), and X. Sun et. al (NeurIPS 2019)

FP8 Training : Hybrid FP8 Data Representation

- Multiple FP8 formats investigated
- FWD FP 1-4-3 and BWD FP 1-5-2 (sign-exponent-mantissa)
- FP 1-4-3 has an exponent bias of 4 to cover small numbers, [2^{-11}, 30]



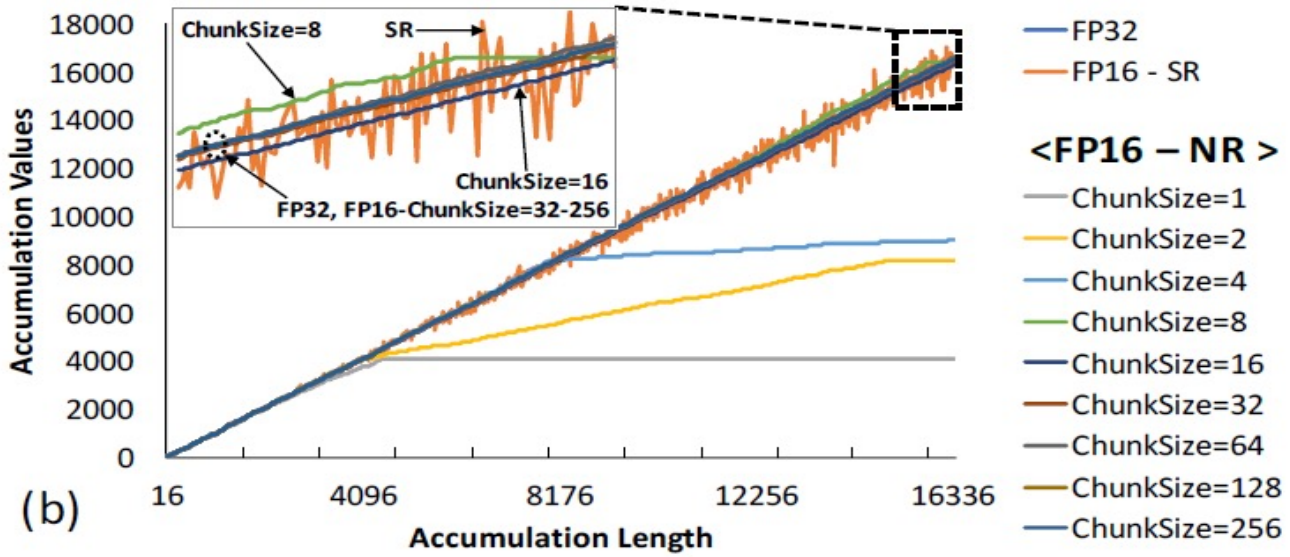
N. Wang et. al (NeurIPS 2018), and X. Sun et. al (NeurIPS 2019)

FP8 Training : FP16 Accumulations

Input: $\{x_n\}_{n=1:N}, \{y_n\}_{n=1:N}\ (FP_{mult})$,
Parameter: chunk size CL
Output: sum (FP_{acc})

$sum = 0.0;\ idx = 0;\ num_{ch} = N/CL$
for n=1:num_{ch} {
  $sum_{ch} = 0.0$
  for i=1:CL {
    idx++
    $tmp = x_{idx} \cdot y_{idx}$ (in FP_{mult})
    sum_{ch} += tmp (in FP_{acc}) }
  sum += sum_{ch} (in FP_{acc}) }

(a)

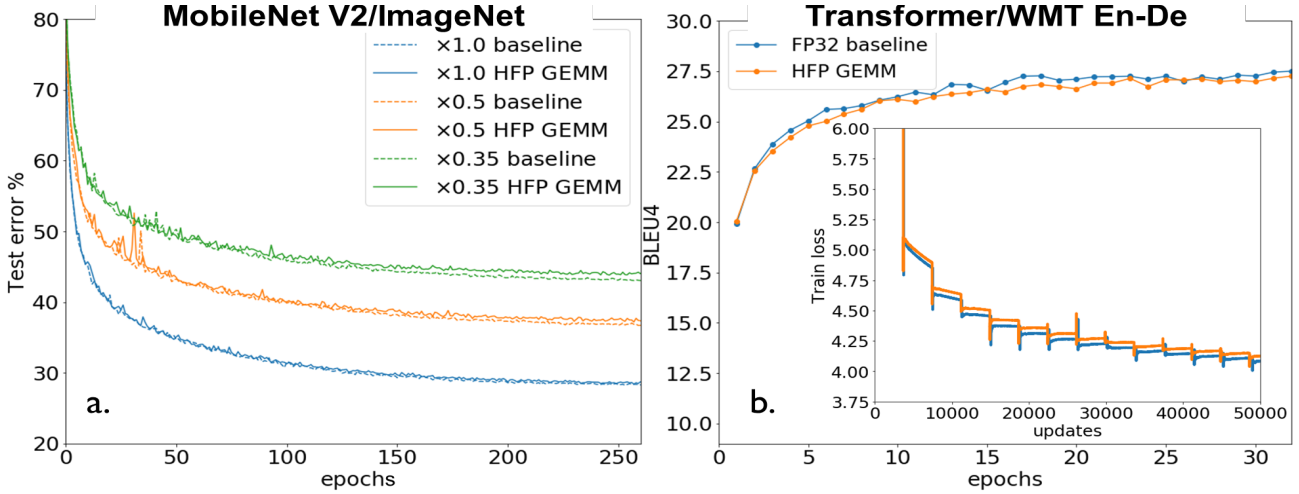


(b)

- **Identified low-precision swamping as the key challenge with FP16 accumulations (current state of the art is FP32 accumulations).**
  - Floating point addition involves right-shift of the smaller operands by the difference in exponents. In case of large-to-small number addition, small numbers maybe partially or completely truncated causing information loss.
- **Chunk-based accumulations (Hierarchical accumulations) fix this problem.**
  - Enables FP16 accumulation (may need hardware support)

N. Wang et. al (NeurIPS 2018), and X. Sun et. al (NeurIPS 2019)

FP8 Training Results



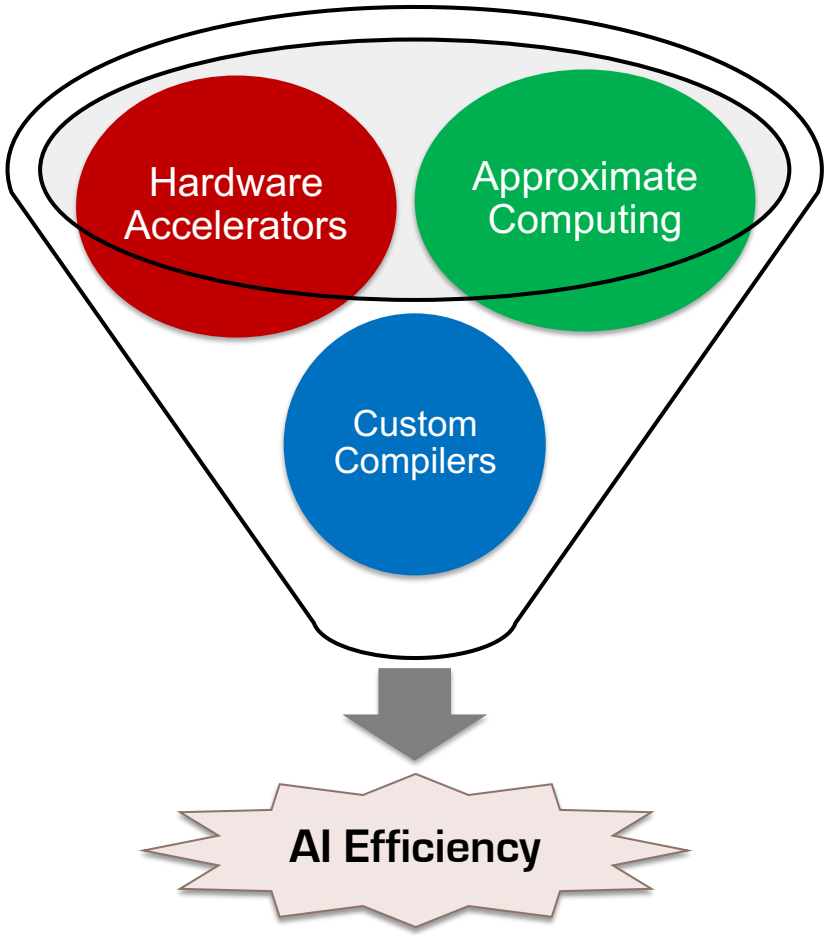
Model(Dataset) Accuracy or [other metrics]	Baseline(FP32)	HFP8 + Round-off update
AlexNet (ImageNet)	57.28	57.21
ResNet18 (ImageNet)	69.38	69.39
ResNet50 (ImageNet)	76.44	76.22
MobileNetV2 (ImageNet)	71.81	71.61
DenseNet121 (ImageNet)	74.76	74.65
2-LSTM (PennTreeBank)[Test ppl.]	83.66	83.86
Transformer-base (WMT14 En-De)[BLEU]	27.50	27.27
4-bidirectional-LSTM Speech (SWB300)[WER]	9.90	10.00
MaskRCNN(ResNet50) (COCO)[Box/Mask AP]	33.58/29.27	33.06/28.86
SSD-Lite(MobileNetV2) (VOC)[mAP]	68.79	68.72

- Combination of selective FP8 precisions, chunk-based accumulations and stochastic weight updates enable baseline accuracies with FP8 Training for a wide variety of neural networks.

N. Wang et. al (NeurIPS 2018), and X. Sun et. al (NeurIPS 2019)

Summary

- **AI workloads have revolutionized application landscape**
  - Enabled new apps and services
  - Impose extreme computational challenges
- **Need to rethink computing stack to boost efficiency of AI workloads**
- **3 key ingredients to building an efficient AI system:**
  - **Hardware Accelerators**: Specialized computing systems for AI
  - **Custom Compilers**: Software stack to extract efficiency without sacrificing end-user productivity
  - **Approximate Computing**: Leverage resiliency to approximate computations to benefit efficiency
- **Many exciting opportunities for the future as workloads continue to grow and evolve!**



tinyML Summit 2022 Sponsors

Copyright Notice

www.tinyml.org