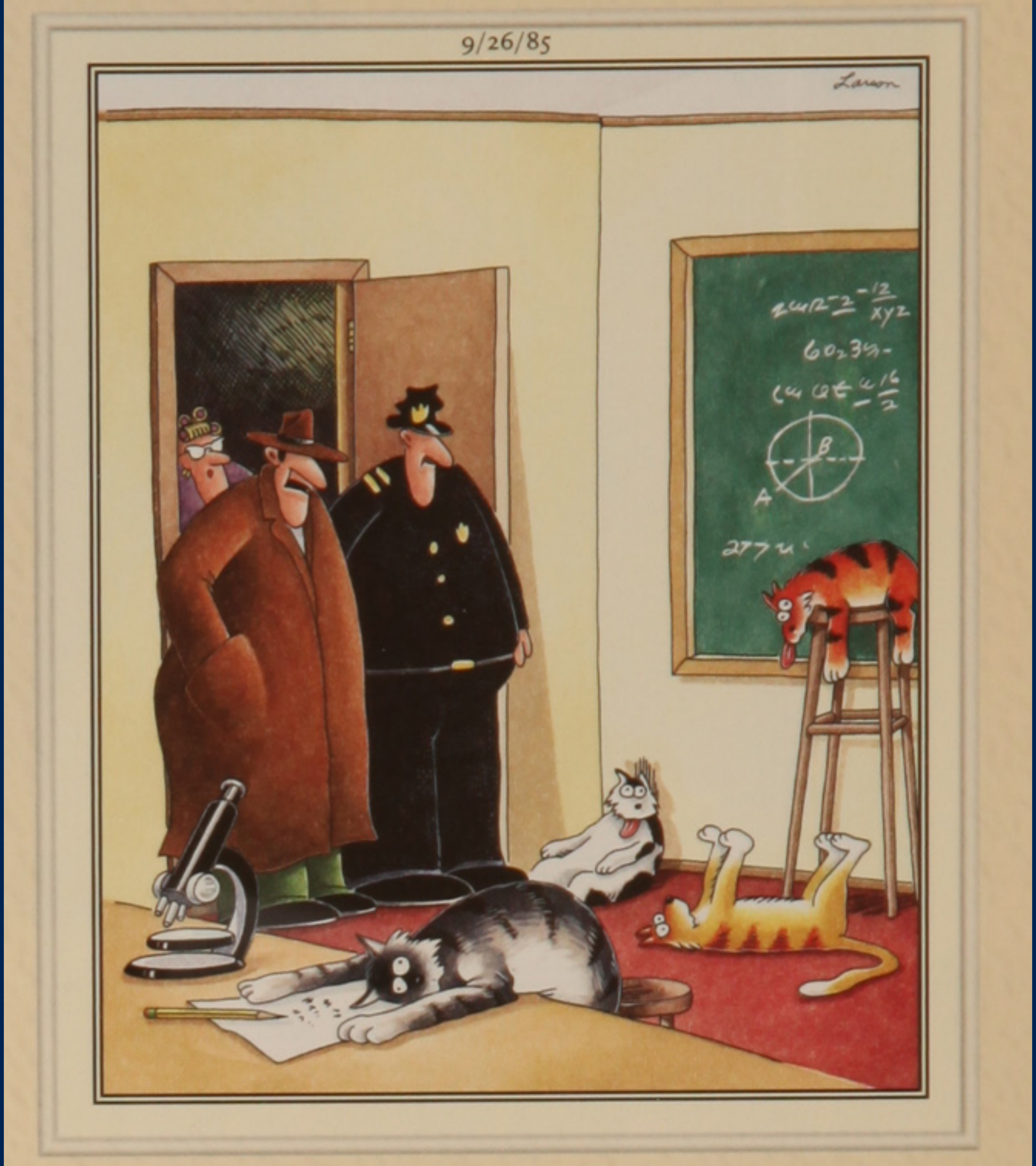
tensorflow and swift

by brett koonce

november 17th, 2018

static.brettkoonce.com/presentations/tensorflow_swift.pdf



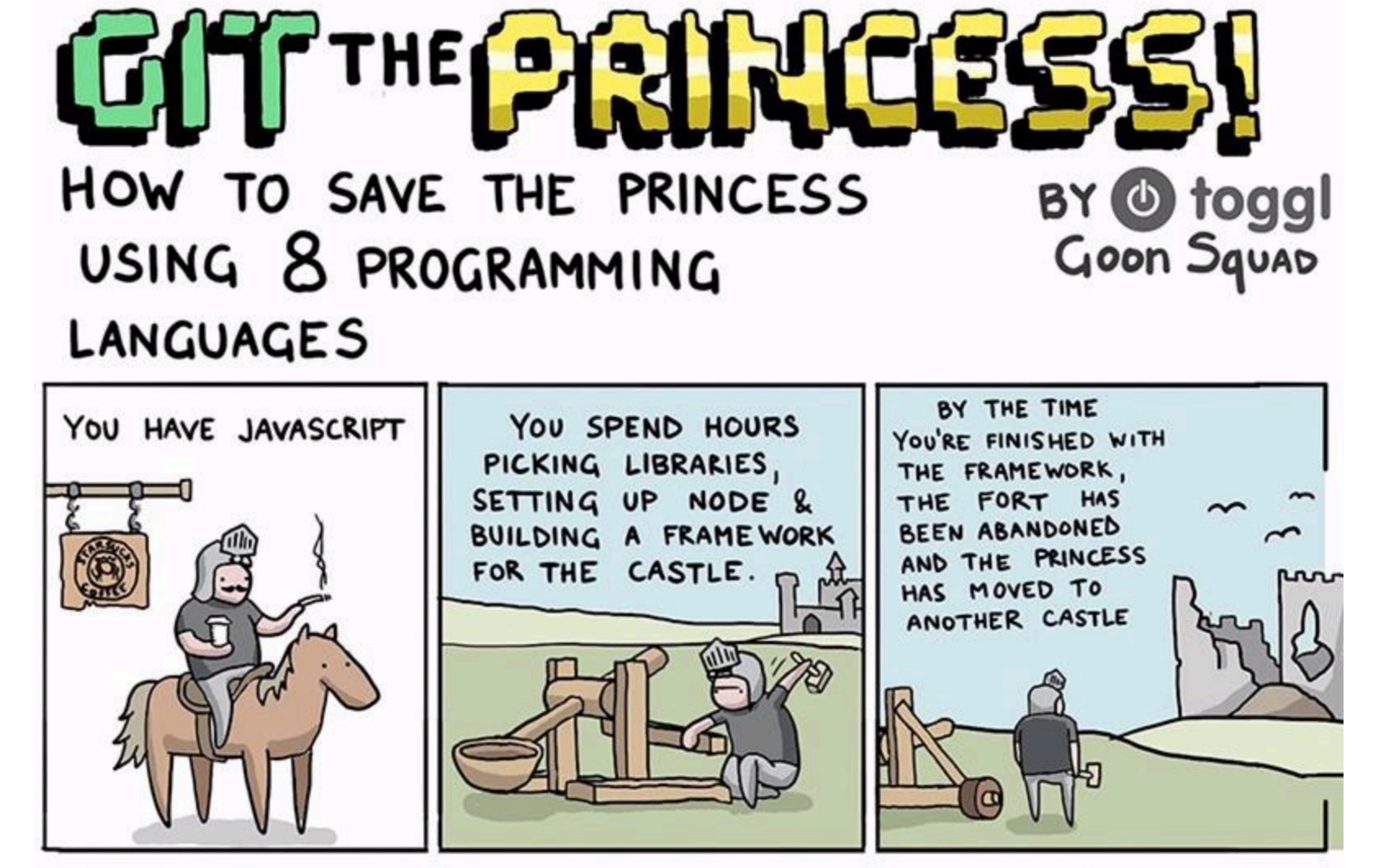
- **“Notice all the computations, theoretical scribblings and lab equipment, Norm. … Yes, curiosity killed these cats.”**

platform

- 0) math / algorithms
- 1) basic virtual machines (juypter)
- 2) cloud software (unix)
- 3) edge (local mobile/embedded)
- 4) custom hardware (tpu, volta, asic)

training

- -1) python, roulette
- 0) calculus/linear algebra basics
- 1) fast.ai 2018 sequence, pytorch
- 2) read, practice
- 3) get into real world


GIT THE PRINCESS!
HOW TO SAVE THE PRINCESS USING 8 PROGRAMMING LANGUAGES
BY toggl
Goon Squad
YOU HAVE JAVASCRIPT
YOU SPEND HOURS PICKING LIBRARIES, SETTING UP NODE & BUILDING A FRAMEWORK FOR THE CASTLE.
BY THE TIME YOU'RE FINISHED WITH THE FRAMEWORK, THE FORT HAS BEEN ABANDONED AND THE PRINCESS HAS MOVED TO ANOTHER CASTLE

five easy pieces

- storage.googleapis.com/tfjs-examples/mnist/dist/index.html
- modeldepot.github.io/tfjs-yolo-tiny-demo/
- magenta.tensorflow.org/js-announce
- poloclub.github.io/ganlab/
- blog.mgechev.com/2018/10/20/transfer-learning-tensorflow-js-data-augmentation-mobile-net/

tensorflow.js demo

- github.com/brettkoonce/mobilenet-tfjs
- mobilenet + tensorflow.js
- docker/node container
- ibm/openwhisk cloud function
- curl + POST + base64 image

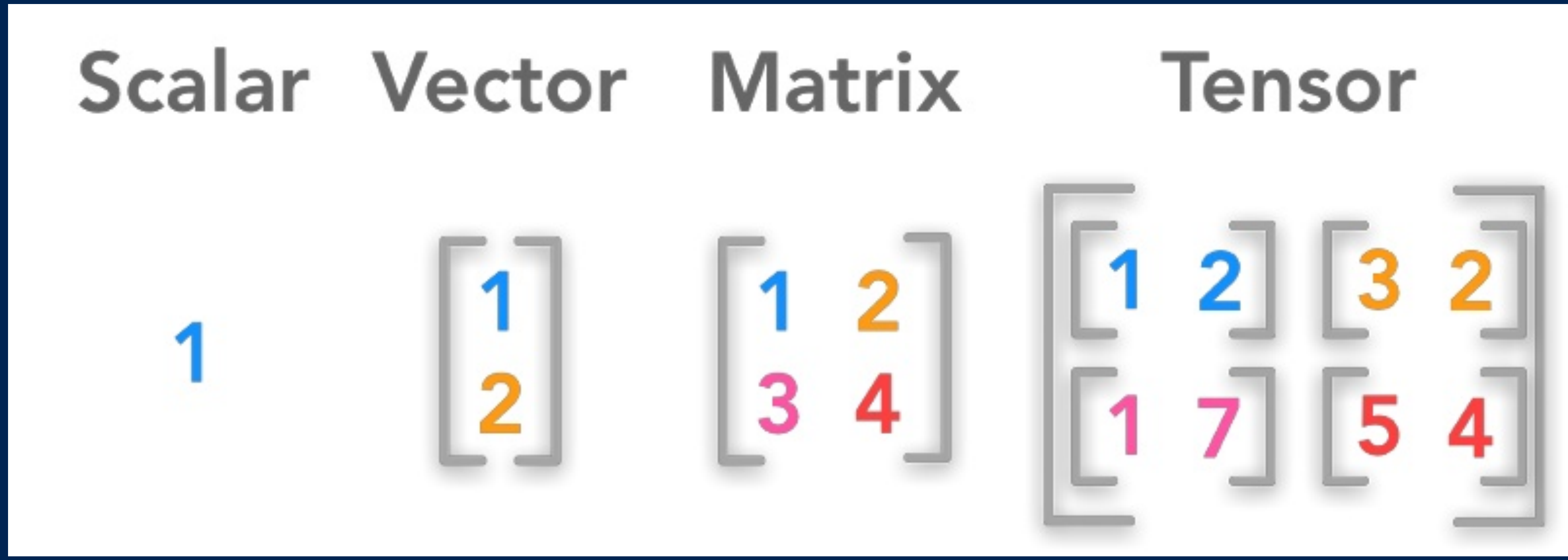
swift

overview

- tensors, flows, combined
- current state of the art
- llvm + swift
- glimpse of the future

tensors

- matrices + algebra
- aX + b —> cX + d —> rules for combining rules
- algebra over matrices
- ...over graphs, type theory



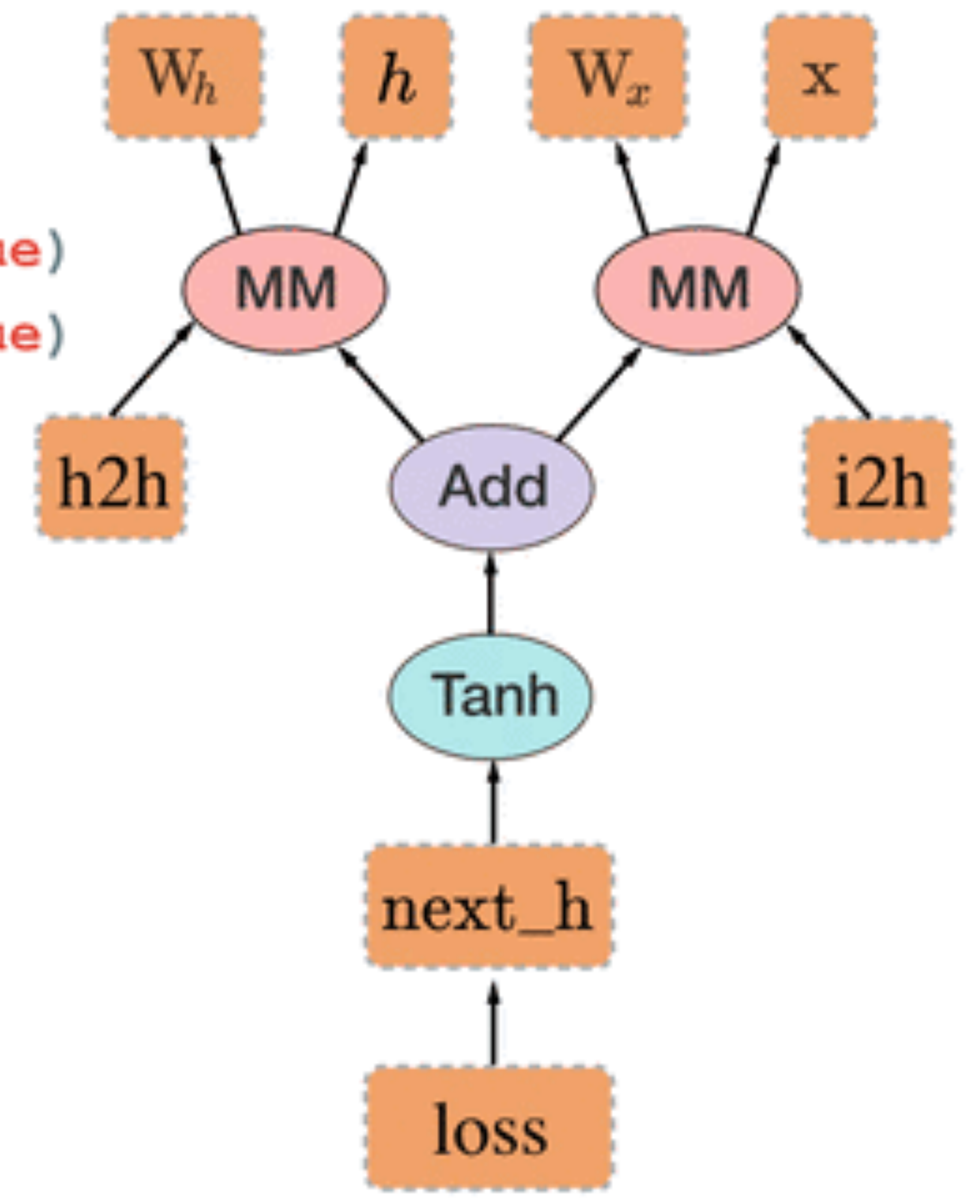
network flows

Back-propagation
uses the dynamically created graph

```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)

h2h = torch.mm(W_h, prev_h.t())
i2h = torch.mm(W_x, x.t())
next_h = h2h + i2h
next_h = next_h.tanh()

loss = next_h.sum()
loss.backward()  # compute gradients!
```



- **pytorch, eager execution, tensorflow 2**

neural turing machines

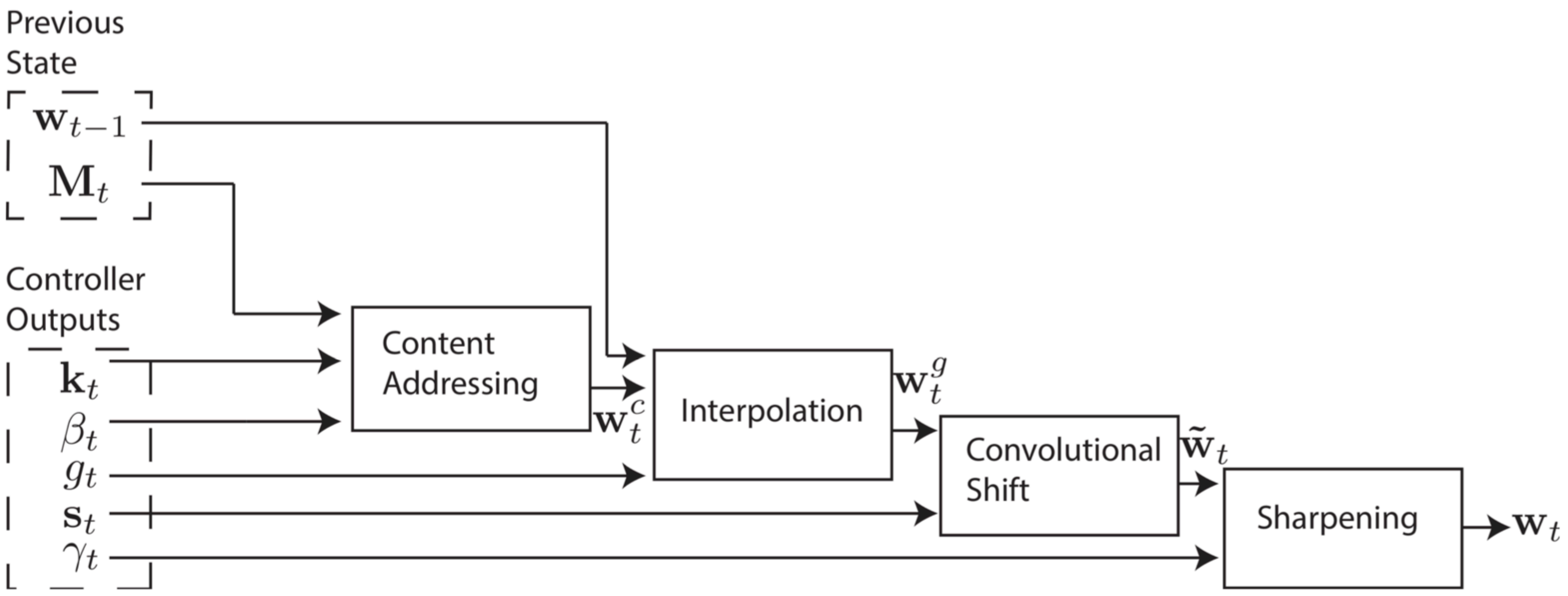


Figure 2: Flow Diagram of the Addressing Mechanism. The *key vector*, $\mathbf{k}_t$, and *key strength*, β_t, are used to perform content-based addressing of the memory matrix, $\mathbf{M}_t$. The resulting content-based weighting is interpolated with the weighting from the previous time step based on the value of the *interpolation gate*, g_t. The *shift weighting*, $\mathbf{s}_t$, determines whether and by how much the weighting is rotated. Finally, depending on γ_t, the weighting is sharpened and used for memory access.

Figure 12: EEG visualization of multi-threaded CPU operations (x-axis is time in μs).

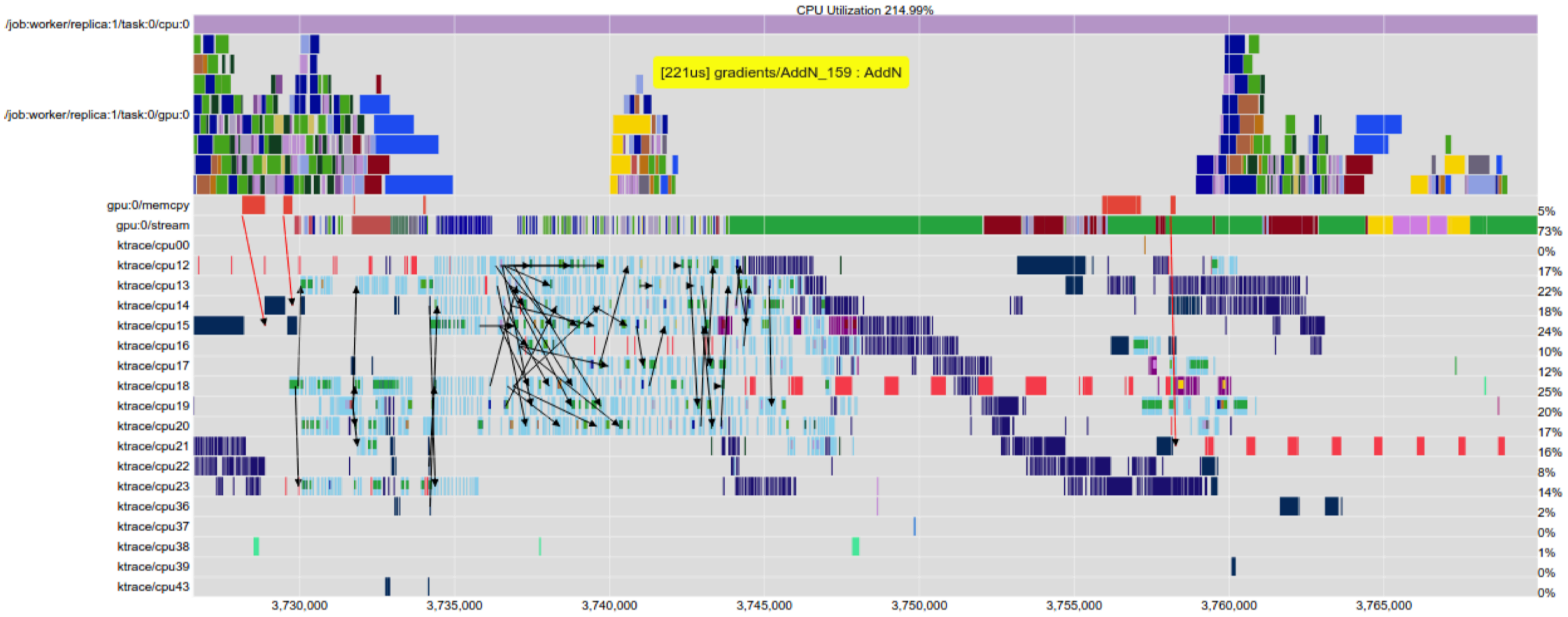
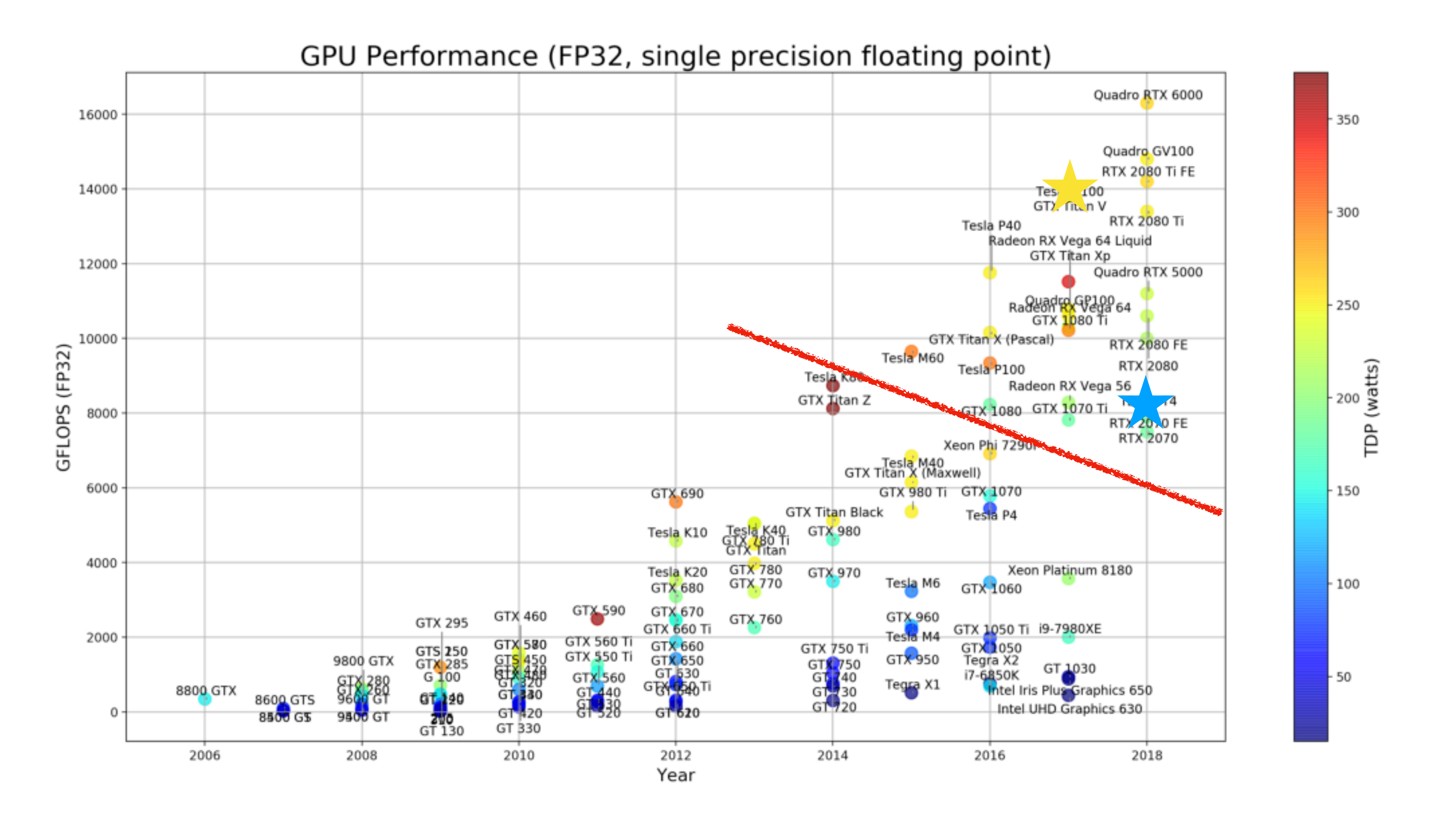
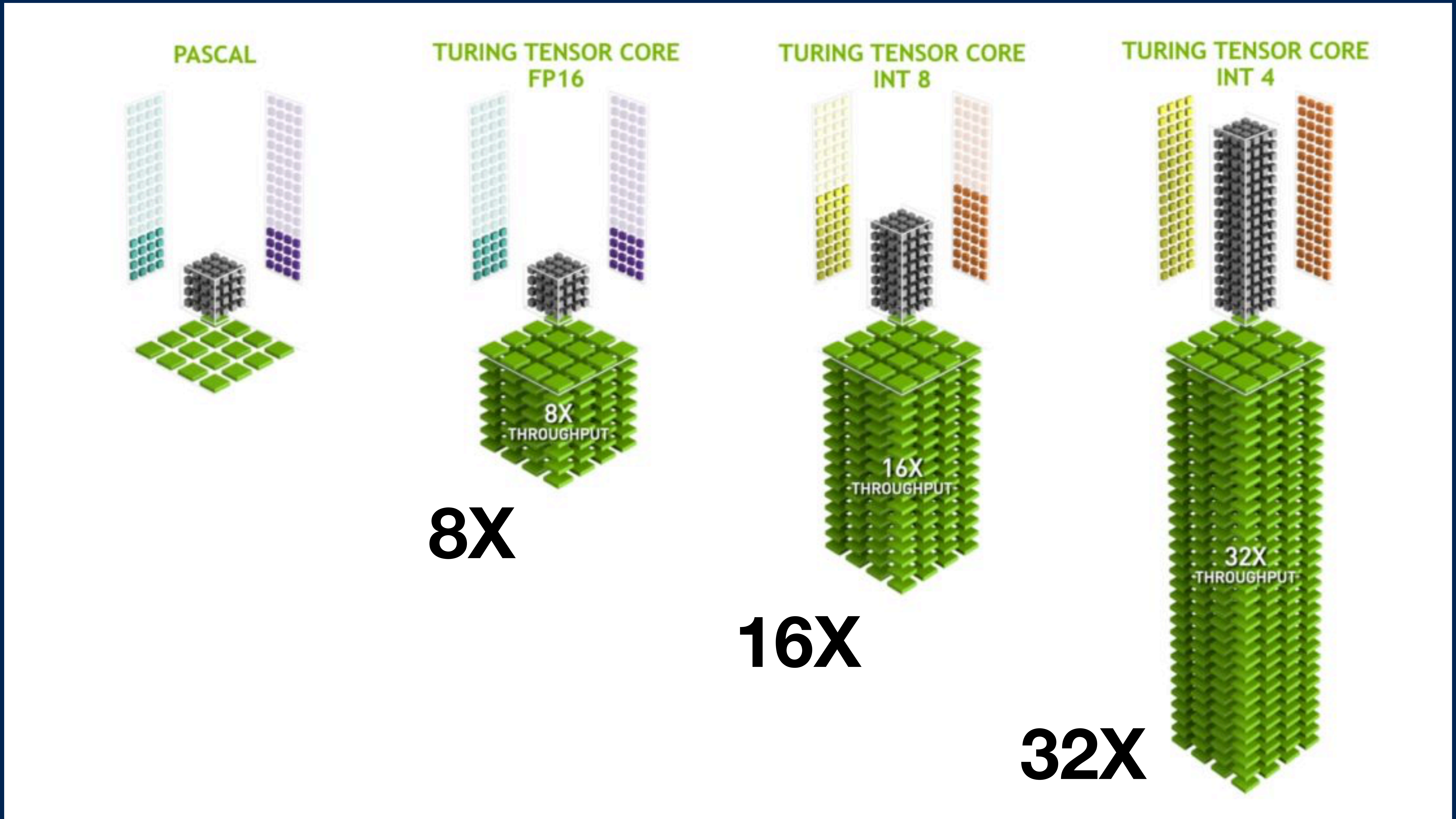


Figure 13: EEG visualization of Inception training showing CPU and GPU activity.

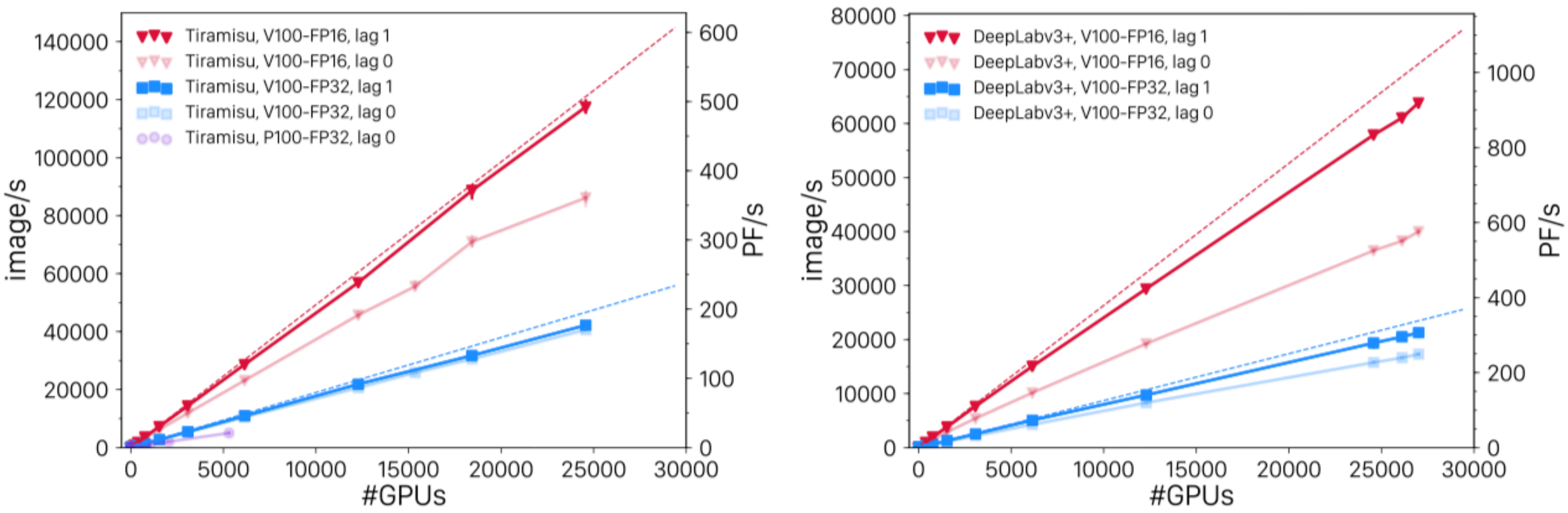
faster hardware



smaller operations




IBM
summit



(a) Tiramisu

(b) DeepLabv3+

Fig. 4: Weak scaling results in terms of images/sec and sustained performance in PF/s on Summit (FP16 and FP32, Tiramisu and DeepLabv3+) and Piz Daint (FP32, Tiramisu). The dashed lines represent the ideal scaling lines for the different architectures and precisions.

- **3500 * dgx-1: scaling, nvlink, nccl, volta**

fp16 case study

- **fast.ai dawnbench recipe:**
  - **algorithms +**
    - **quantized hardware**
    - **quantized software**
    - **distributed training**

Jeff Dean @JeffDean

Following

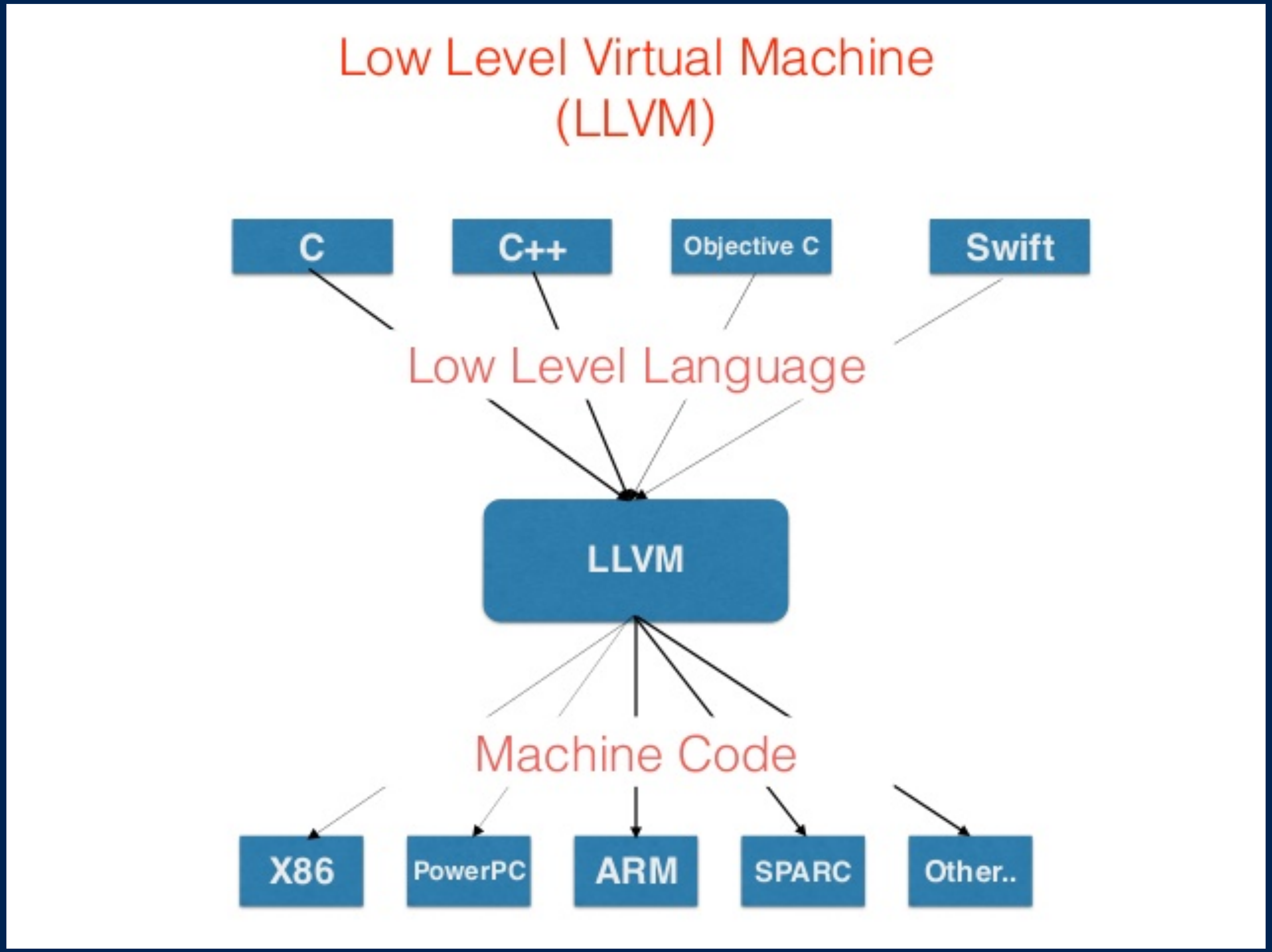
Google Cloud TPUs now offer preemptible pricing at ~70% off the reserved instance pricing. This means, for example, that you can train a ResNet-50 model for ~$7.50 instead of $25, or a Transformer neural translation model for ~$13 instead of $41.

See: cloudplatform.googleblog.com/2018/06/Cloud- ...

Select Open-Source Reference Models	Normal training cost (TF 1.8)	Preemptible training cost (TF 1.8)
ResNet-50 (with optimizations from fast.ai): Image classification	~$25	~$7.50
ResNet-50 (original implementation): Image classification	~$59	~$18
AmoebaNet: Image classification (model architecture evolved from scratch on TPUs to maximize accuracy)	~$49	~$15
RetinaNet: Object detection	~$40	~$12
Transformer: Neural machine translation	~$41	~$13
ASR Transformer: Speech recognition (transcribe speech to text)	~$86	~$27

8:54 AM - 19 Jun 2018

llvm + swift



- opencl, gpu transition
- objective-c, memory, thread safety
- swift, functional programming
- bytecode, recompiled for each device

tensor comprehensions

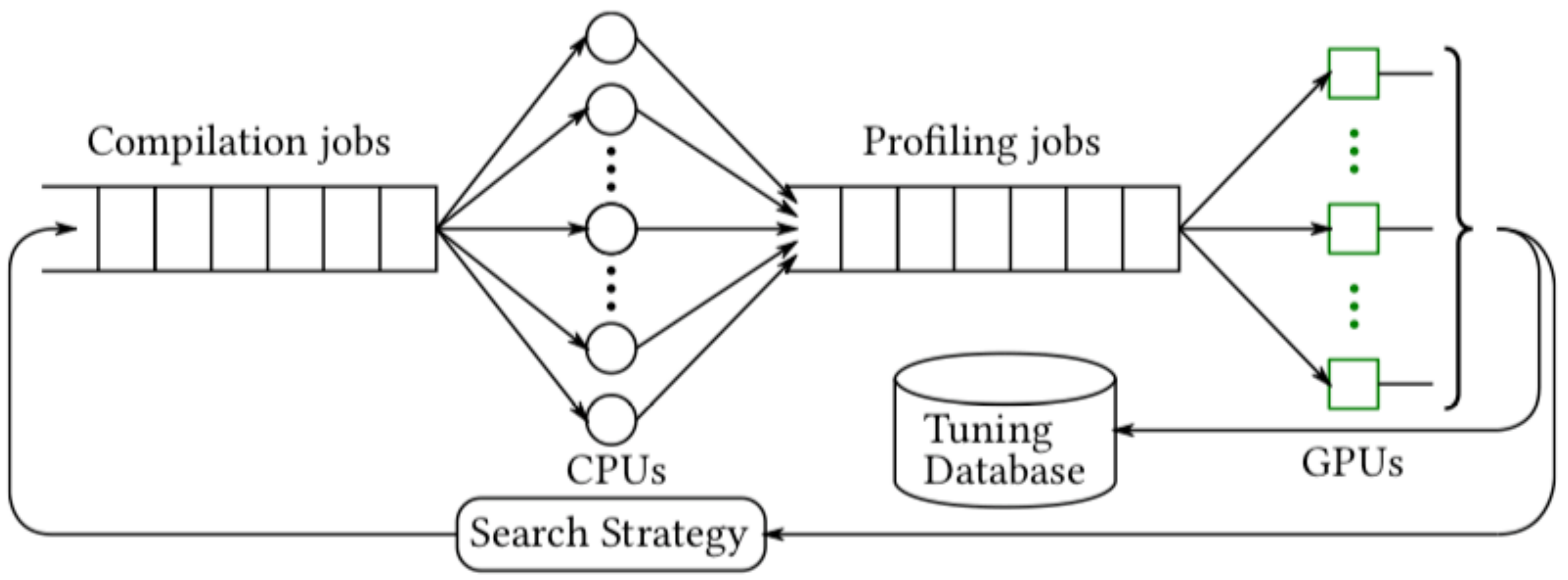


Figure 4: Multithreaded autotuning pipeline for kernels

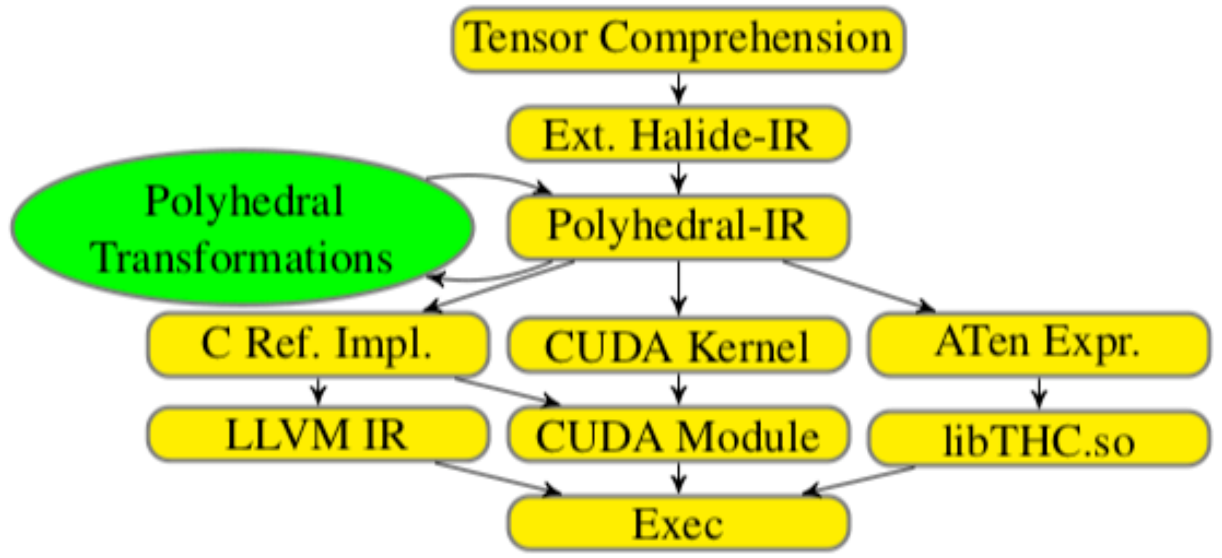


Figure 2: The JIT compilation flow lowers TC to Halide-IR, then to Polyhedral-IR, followed by optimization, code generation and execution

future workloads

Name	LOC	Layers					Nonlinear function	Weights	TPU Ops / Weight Byte	TPU Batch Size	% of Deployed TPUs in July 2016
		FC	Conv	Vector	Pool	Total					
MLP0	100	5				5	ReLU	20M	200	200	61%
MLP1	1000	4				4	ReLU	5M	168	168	
LSTM0	1000	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1500	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1000		16			16	ReLU	8M	2888	8	5%
CNN1	1000	4	72		13	89	ReLU	100M	1750	32	

Table 1. Six NN applications (two per NN type) that represent 95% of the TPU's workload. The columns are the NN name; the number of lines of code; the types and number of layers in the NN (FC is fully connected, Conv is convolution, Vector is self-explanatory, Pool is pooling, which does nonlinear downsizing on the TPU; and TPU application popularity in July 2016. One DNN is RankBrain [Cla15]; one LSTM is a subset of GNM Translate [Wu16]; one CNN is Inception; and the other CNN is DeepMind AlphaGo [Sil16][Jou15].

data types

- int8 (tpu, rtx)
- bfloat16 (tpu, intel)
- int4 (turing)
- -/0/+ networks, signsgd
- bnn, bytenet

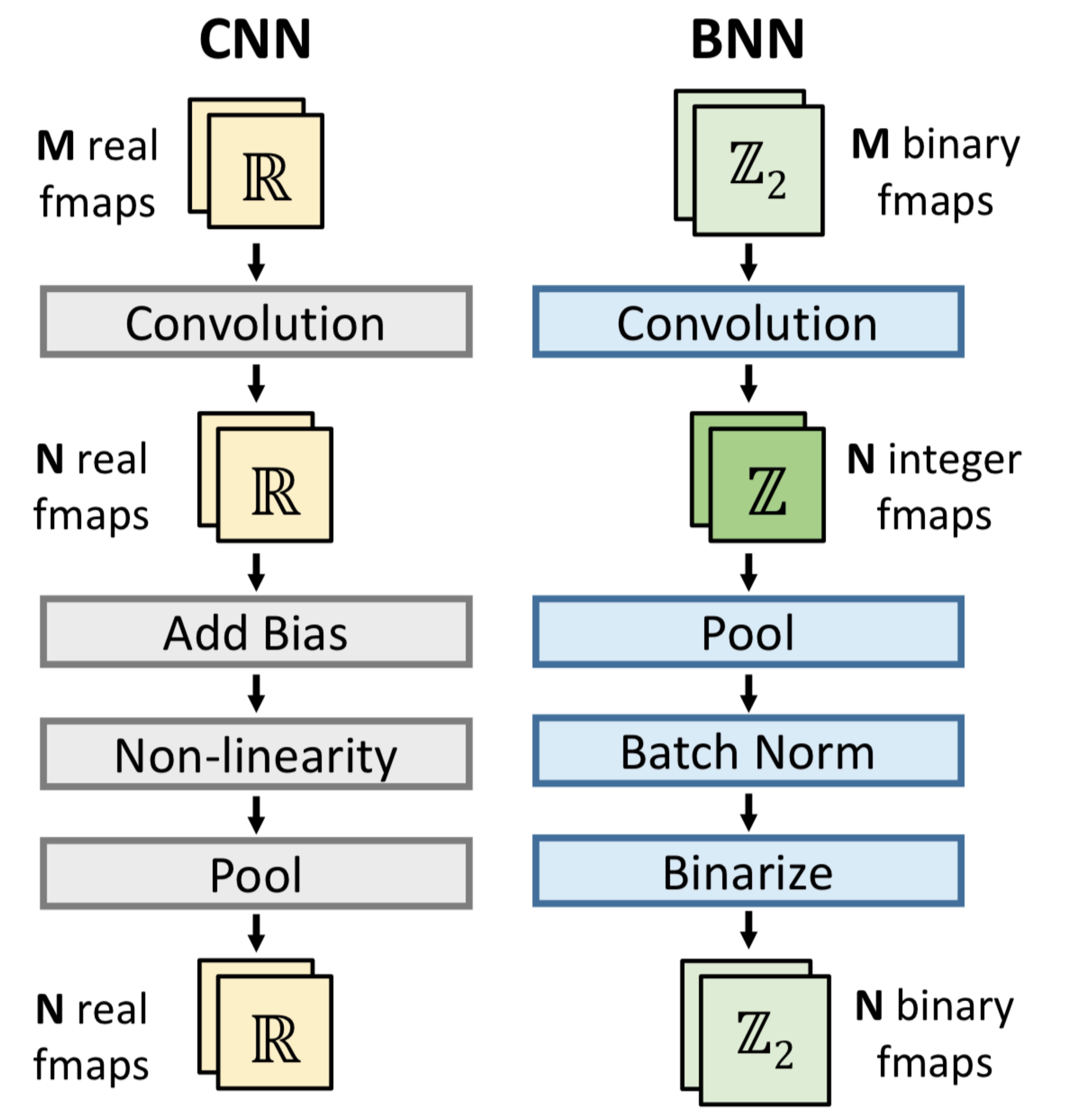
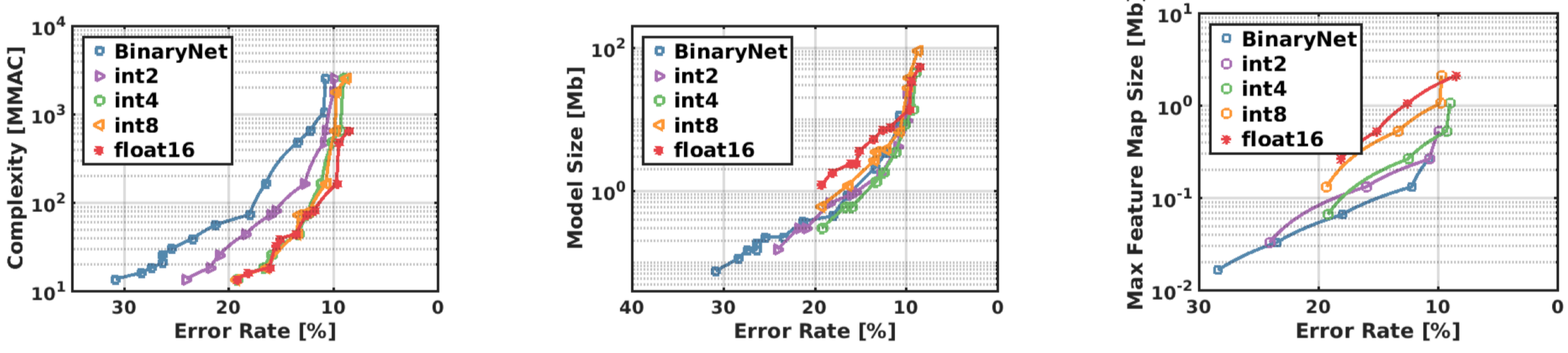


Figure 1: **Comparison of CNNs and BNNs —** Left: the order of operations in a CNN for a conv and pool layer. Right: the (modified) order of operations in the BinaryNet BNN [7]. Pooling is performed early and a batch normalization precedes the binarization to minimize information loss. Biases have been removed from the BNN.

quantized nn



(a) (b) (c)

Fig. 4. QNN networks on CIFAR-10 [18]. (a) computational complexity, (b) model size, (c) Maximum feature map size and the number of bits Q.

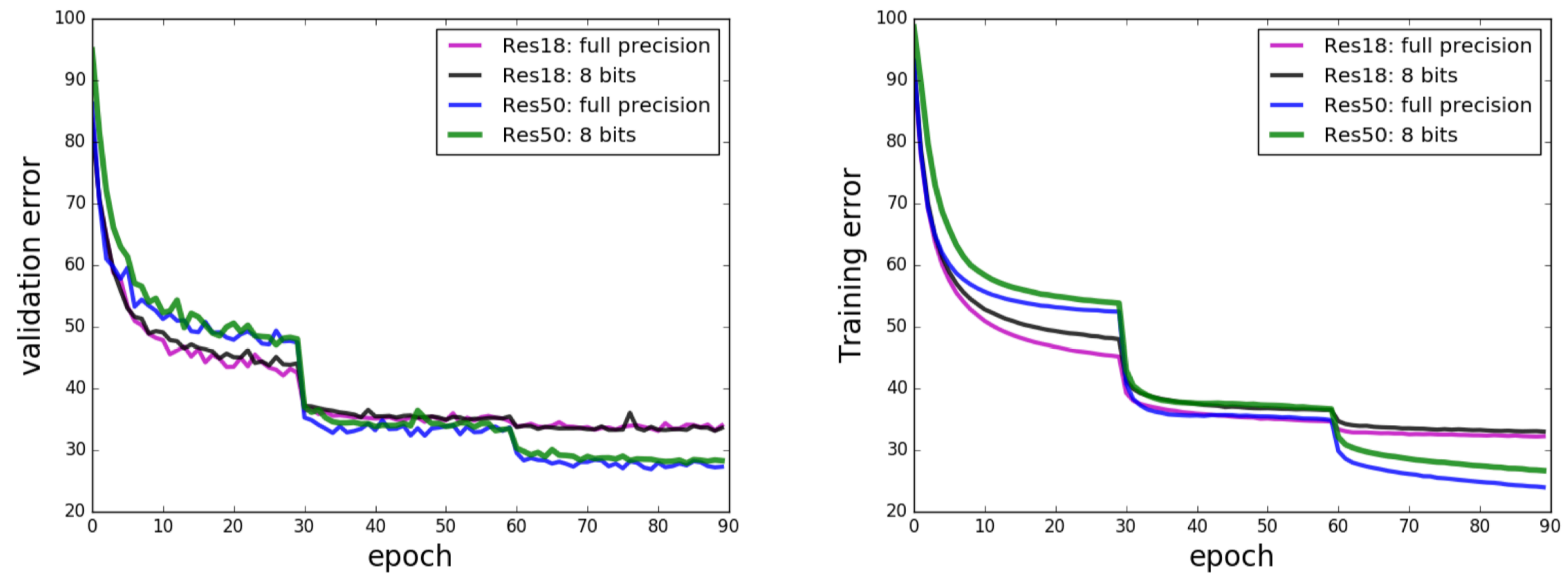
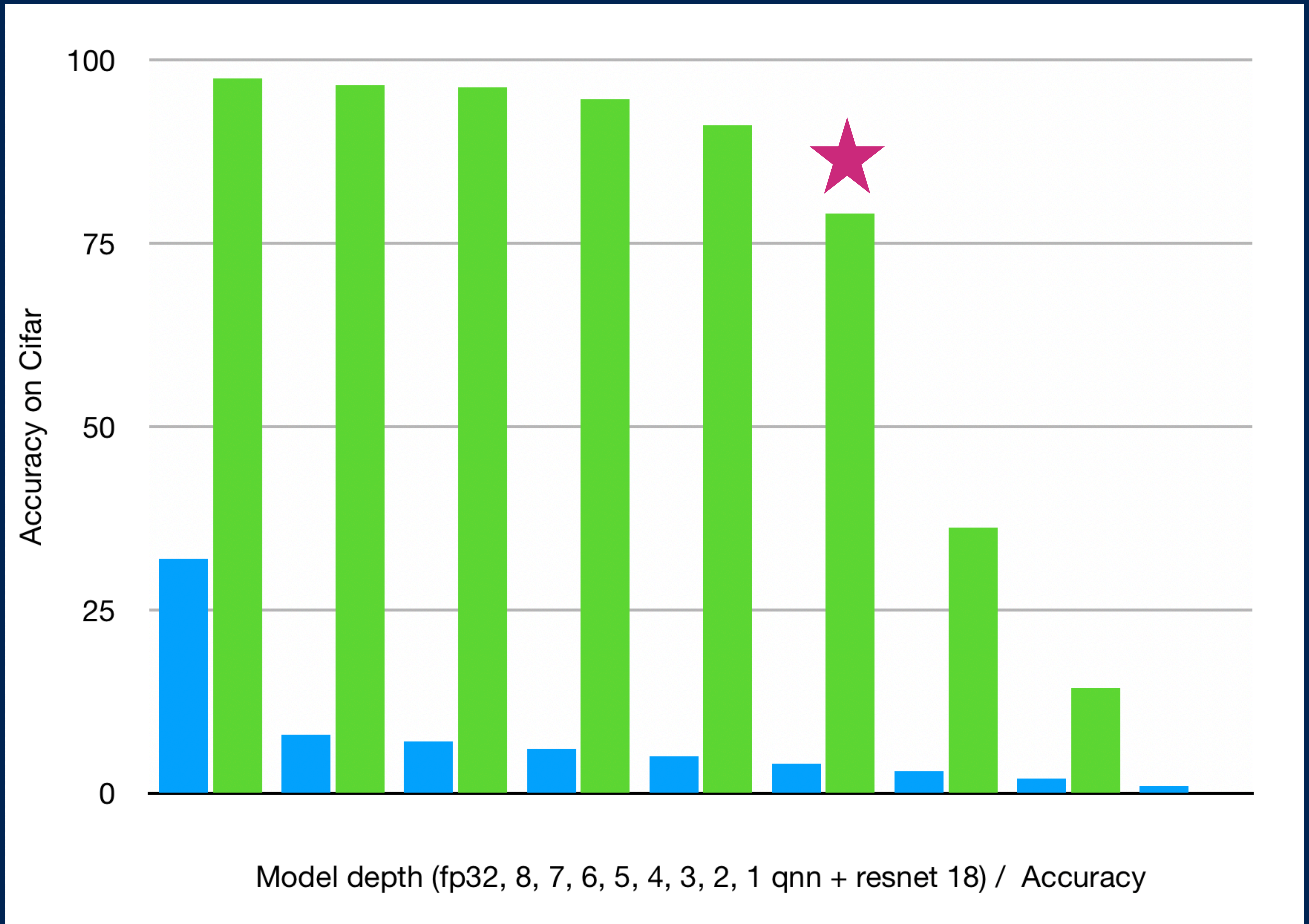


Figure 4: Comparing a full precision run against 8-bit run with Qunatized Back-Propogarion and Range BN (ResNet-18 and ResNet-50 trained on ImageNet).

qnn cifar 10 results

- resnet18 control: fp32
- 8/7/6/5/4/3/2/1 bit resnet18 variants
- github.com/eladhoffer/quantized.pytorch
- demo running on t4 (int4) hardware (THANK YOU GOOGLE CLOUD)



recap

- current state of the art hardware/software
- fp32 —> fp16 —> int8 transition
- llvm + swift
- 4-bit qnn resnet 18 software/hardware

int4 at scale: 2020

- 4 bit hardware + software
- cluster of t4's (~2070 rtx, 260 int4 ToPS)
- cluster of 256 * dgx-3 —> 4k gpu
- $100 / hr —> ~25k/hour —> ~1 exaops
- dgx-1 ~= 1 petaflop —> 1000x scale

thanks for coming!

links

- nvidia turing architecture
- blog.inten.to/hardware-for-deep-learning-part-3-gpu-8906c1644664
- github.com/brettkoonce/mobilenet-tfjs
- quarkworks.co
- brettkoonce.com

papers

- neural turing machines
- tensorflow
- tpu
- Exascale Deep Learning for Climate Analytics
- tensor comprehensions

bnn/qnn papers

- Compressed Optimisation for Non-Convex Problems
- Neural Machine Translation in Linear Time
- Binarized Neural Networks
- Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs
- Scalable Methods for 8-bit Training of Neural Networks
- Minimum Energy Quantized Neural Networks