Systems for Machine Learning (CMSC828G)

Performance Challenges and Modeling

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

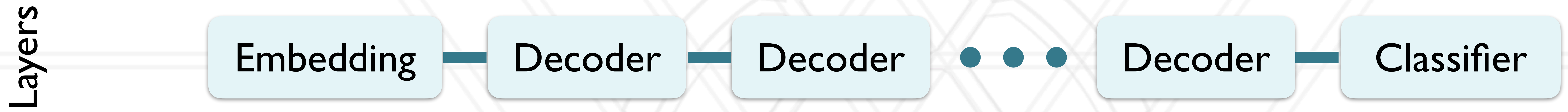
Announcements

- Assignment 2 is posted, due on March 6
- Class participation: due at 9 AM on the day of the lecture
  - Questions / Discussion topics: every lecture
  - Short presentation: once in the semester

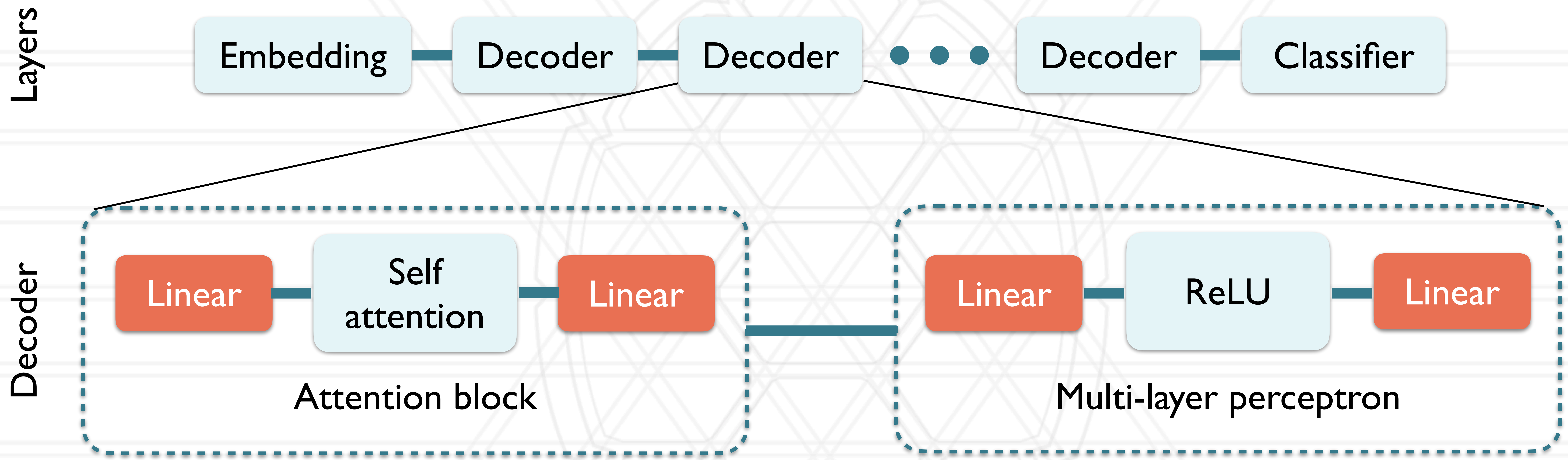
Four types of DL workloads

- Training: also called *pre-training*
- Fine-tuning
- Inference: serving the model
  - Offline: single user
  - Online: single or multiple users

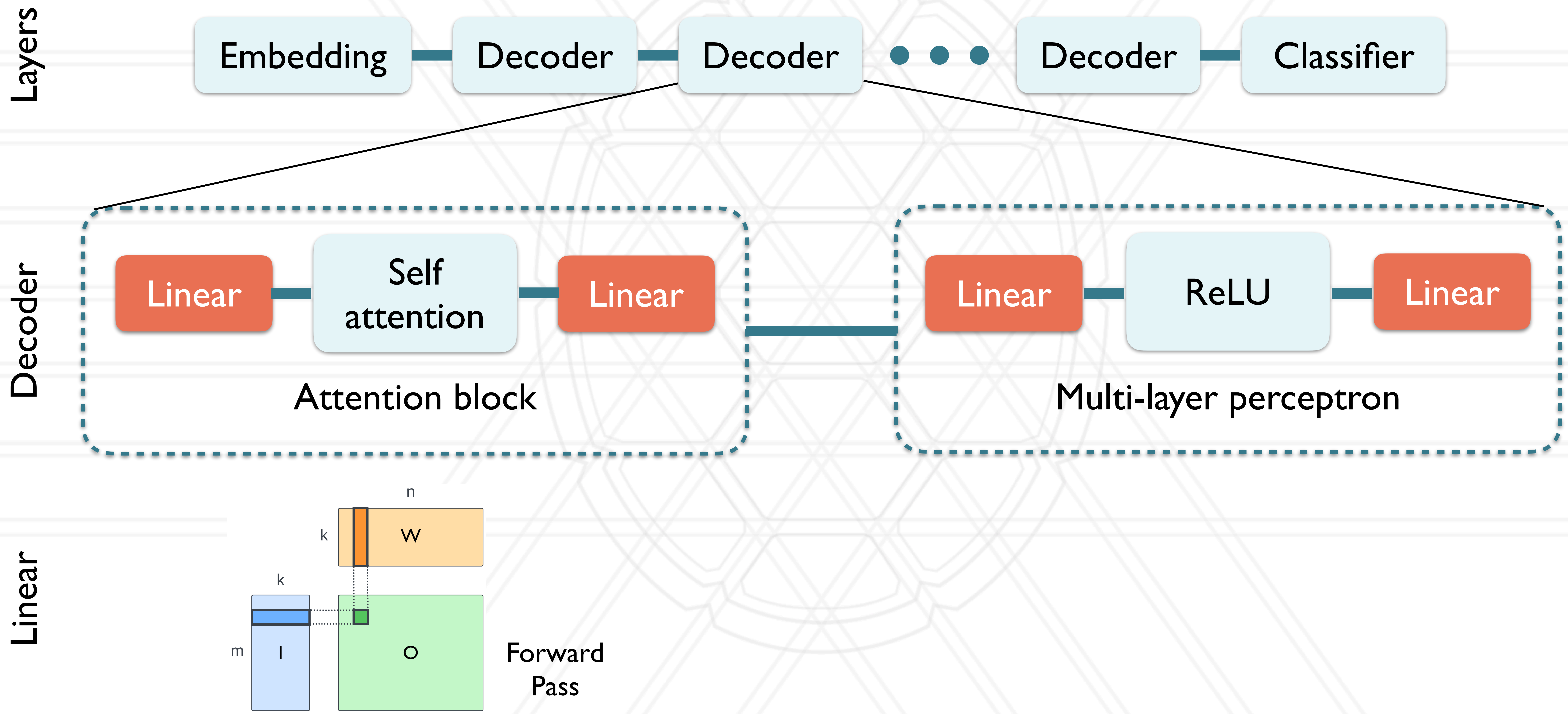
Compute work in transformer models



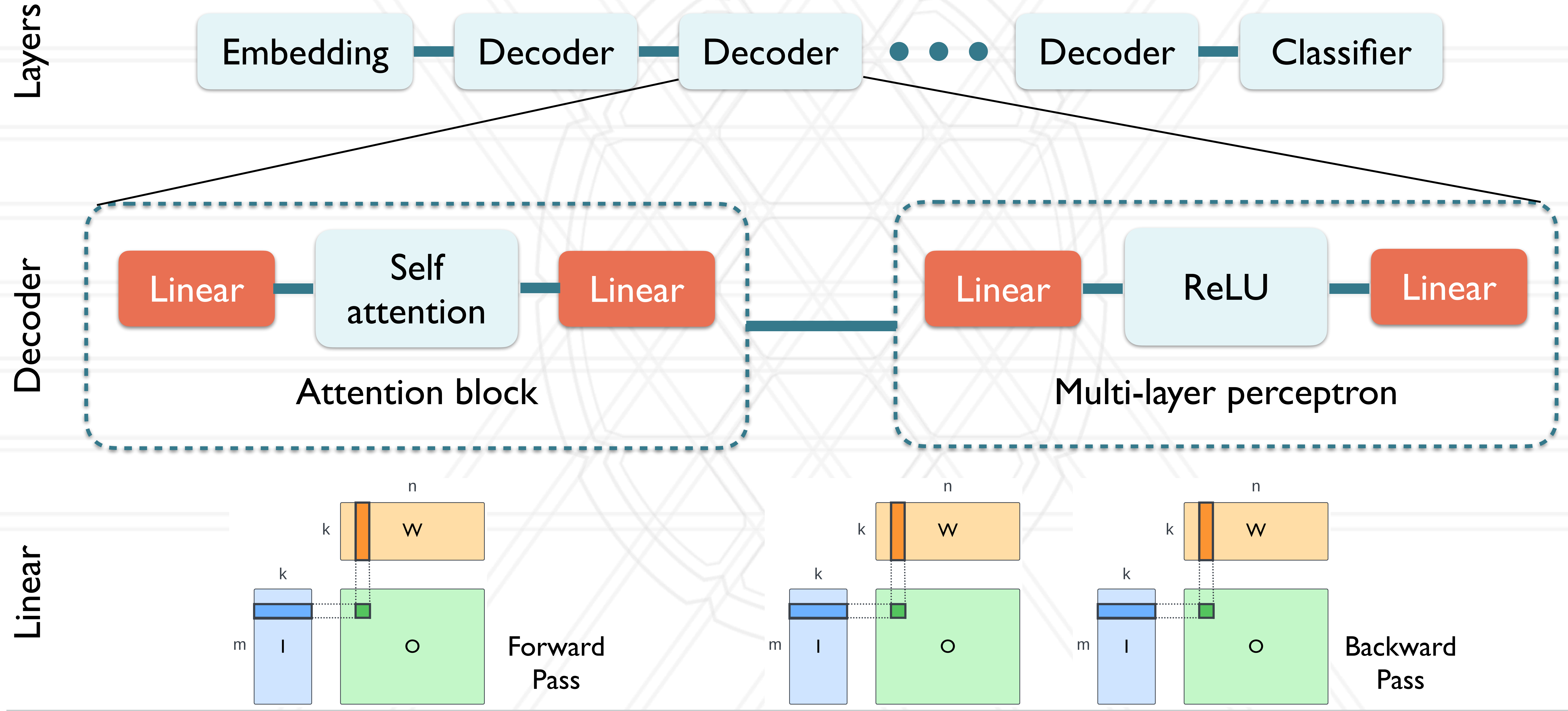
Compute work in transformer models



Compute work in transformer models



Compute work in transformer models



Two main challenges: memory and speed

Two main challenges: memory and speed

- The largest model you can run on an H100 96 GB GPU is around 3.5-4 billion parameters
- On a single node (with four H100 GPUs): around ~16 billion parameters model

Two main challenges: memory and speed

- The largest model you can run on an H100 96 GB GPU is around 3.5-4 billion parameters
- On a single node (with four H100 GPUs): around ~16 billion parameters model

Memory constraints

Two main challenges: memory and speed

- The largest model you can run on an H100 96 GB GPU is around 3.5-4 billion parameters
- On a single node (with four H100 GPUs): around ~16 billion parameters model

Memory constraints

- Training a 16B parameter would take 33 years!

Two main challenges: memory and speed

- The largest model you can run on an H100 96 GB GPU is around 3.5-4 billion parameters
- On a single node (with four H100 GPUs): around ~16 billion parameters model

Memory constraints

- Training a 16B parameter would take 33 years!

Speed

Two main challenges: memory and speed

- The largest model you can run on an H100 96 GB GPU is around 3.5-4 billion parameters
- On a single node (with four H100 GPUs): around ~16 billion parameters model

Memory constraints

- Training a 16B parameter would take 33 years!

Speed

- OpenAI's GPT 4.0 is estimated to have 1.8 trillion parameters
- Meta's Llama-3.1-405B has more than 400 billion parameters

Memory challenges

- Challenge: fit larger and larger models on hardware with limited memory?
- Store and compute in reduced precision
  - Lower precision: fp32 → fp16 / bf16
  - Mixed-precision: do some operations in fp32 and some in fp16, also store some quantities in fp32 and some in fp16
- Alternative approach: use distributed memory
  - Train/fine-tune/infer on more than one GPU/node

Speed: single GPU performance

- Challenge: Ensure great performance on a GPU
- Running compute kernels efficiently
- Moving data efficiently (global to shared/local memory within the GPU or CPU ⇆ GPU)
- Systems approaches: optimizing kernel performance, reducing amount of data moved
  - e.g. flash attention
- ML approaches: alternative methods to solve the same problem
  - Reducing model size: quantization, exploiting sparsity, mixture-of-experts, …
  - e.g. first (Adam) and second-order (KFAC, Shampoo, …) optimizers

Speed: multi-GPU or parallel performance

- Challenge: ensuring scalability as model sizes and number of GPUs used increases
- Challenge: communication / data movement overheads
- Approaches:
  - Clever parallel algorithms: data / tensor / pipeline / hybrid
  - Optimizing collective operations
  - Enabling / ensuring compute-communication overlap

Speed: I/O performance

- Challenges: ensuring that data movement from / to disk is not a bottleneck
- Issue: data is read into the CPU memory and then needs to be transferred into GPU memory

Challenges specific to the workload

- Training / Fine-tuning: scalability might be a larger issue
- Inference: could get away with a smaller number of GPUs
  - Offline: latency and throughput
    - Time to first token, throughput (tokens/sec)
  - Online: handling lots of requests
    - Fairness (scheduling of requests)

Performance metrics

- Wall clock time: end-to-end time or time-to-solution
- Time per batch (iteration)
- Throughput: batches/s, token/s
- Utilization: what % of time is the GPU utilized (not idle)
- Flop/s: floating point operations per second
- Parallel speedup and efficiency

How do you calculate flop/s?

- Floating point operations per second
- Time a certain code region or kernel and gather/calculate empirical or model flops for the same region
- Function of the algorithm, input (data), hardware

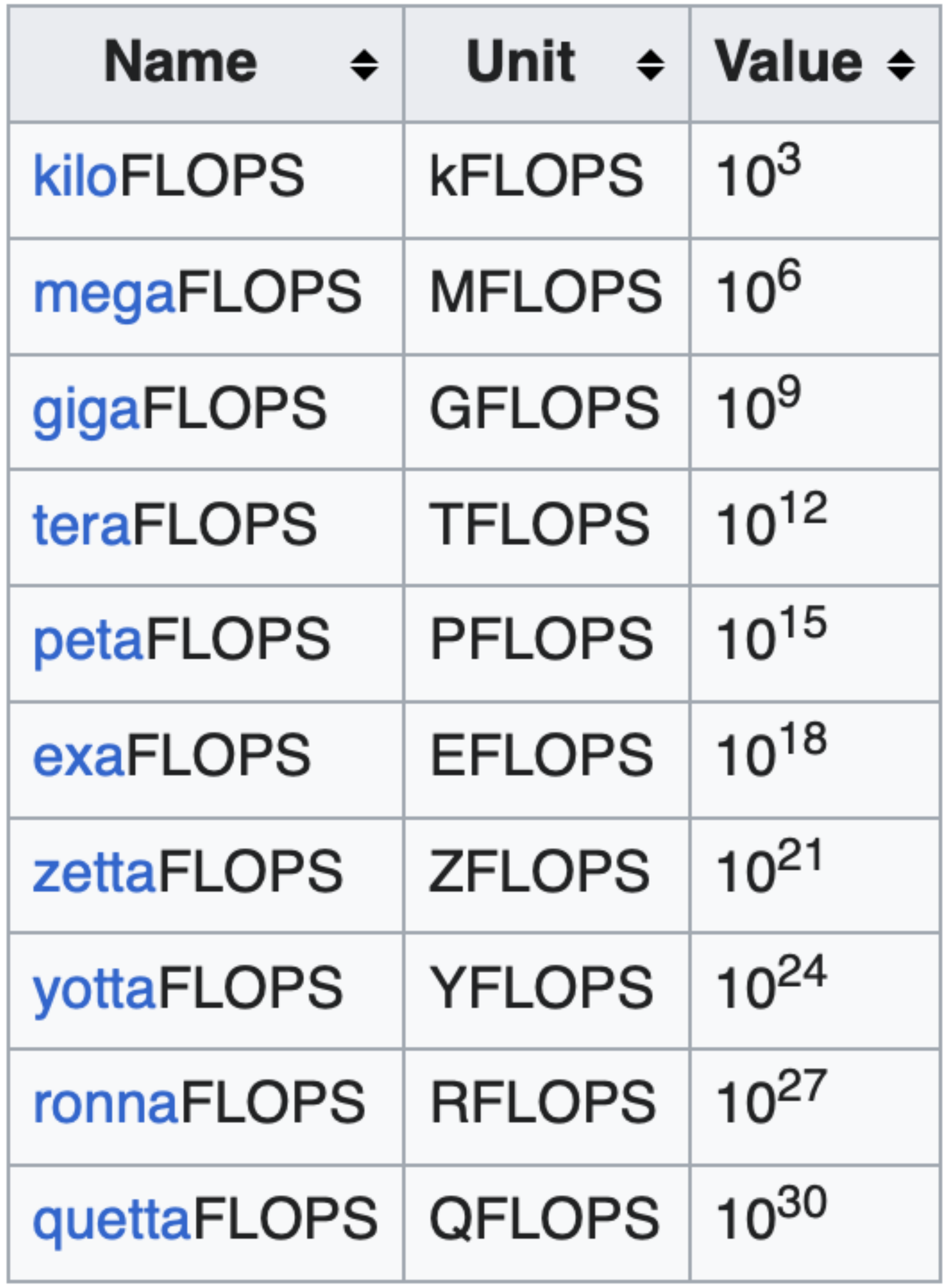
$$\text{flop/s} = \frac{flops}{time}$$

Empirical versus model flops

- Model flops (theoretical): calculated analytically by looking at the code and counting the number of floating point operations
- Empirical flops (achieved): obtained using performance tools that inspect hardware counters (registers)
    - For example, `PAPI_FP_OPS` on CPUs
    - `DCGM_FI_PROF_PIPE_FP{64,32,16}_ACTIVE` and `DCGM_FI_PROF_PIPE_TENSOR_ACTIVE` on GPUs

Multipliers for flops

- Floating point operations: flops
- Floating point operations per second: flop/s

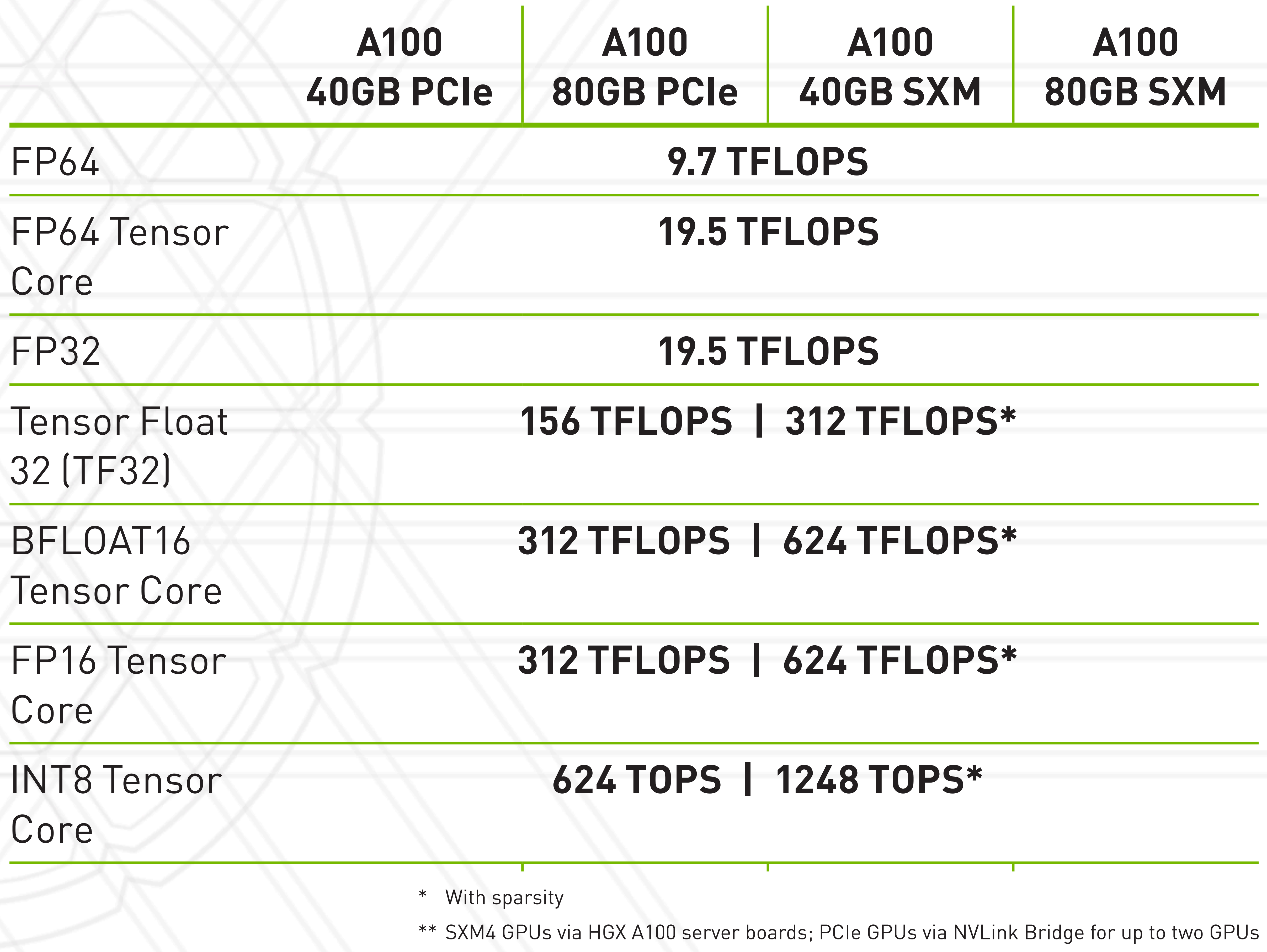


Name	Unit	Value
kiloFLOPS	kFLOPS	10^3
megaFLOPS	MFLOPS	10^6
gigaFLOPS	GFLOPS	10^9
teraFLOPS	TFLOPS	10^{12}
petaFLOPS	PFLOPS	10^{15}
exaFLOPS	EFLOPS	10^{18}
zettaFLOPS	ZFLOPS	10^{21}
yottaFLOPS	YFLOPS	10^{24}
ronnaFLOPS	RFLOPS	10^{27}
quettaFLOPS	QFLOPS	10^{30}

https://en.wikipedia.org/wiki/Floating_point_operations_per_second

Vendor advertised vs. actual flop/s

- Sustained flops on A100: 280 Tflop/s (90% of peak)
- Sustained flops on H100: 813 Tflop/s (82% of peak)
- Sustained flop/s on MI250X: 125 Tflop/s on 1 GCD (65% of peak)



	A100 40GB PCIe	A100 80GB PCIe	A100 40GB SXM	A100 80GB SXM
FP64	9.7 TFLOPS			
FP64 Tensor Core	19.5 TFLOPS			
FP32	19.5 TFLOPS			
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*			
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
INT8 Tensor Core	624 TOPS | 1248 TOPS*			

* With sparsity

** SXM4 GPUs via HGX A100 server boards; PCIe GPUs via NVLink Bridge for up to two GPUs

https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf

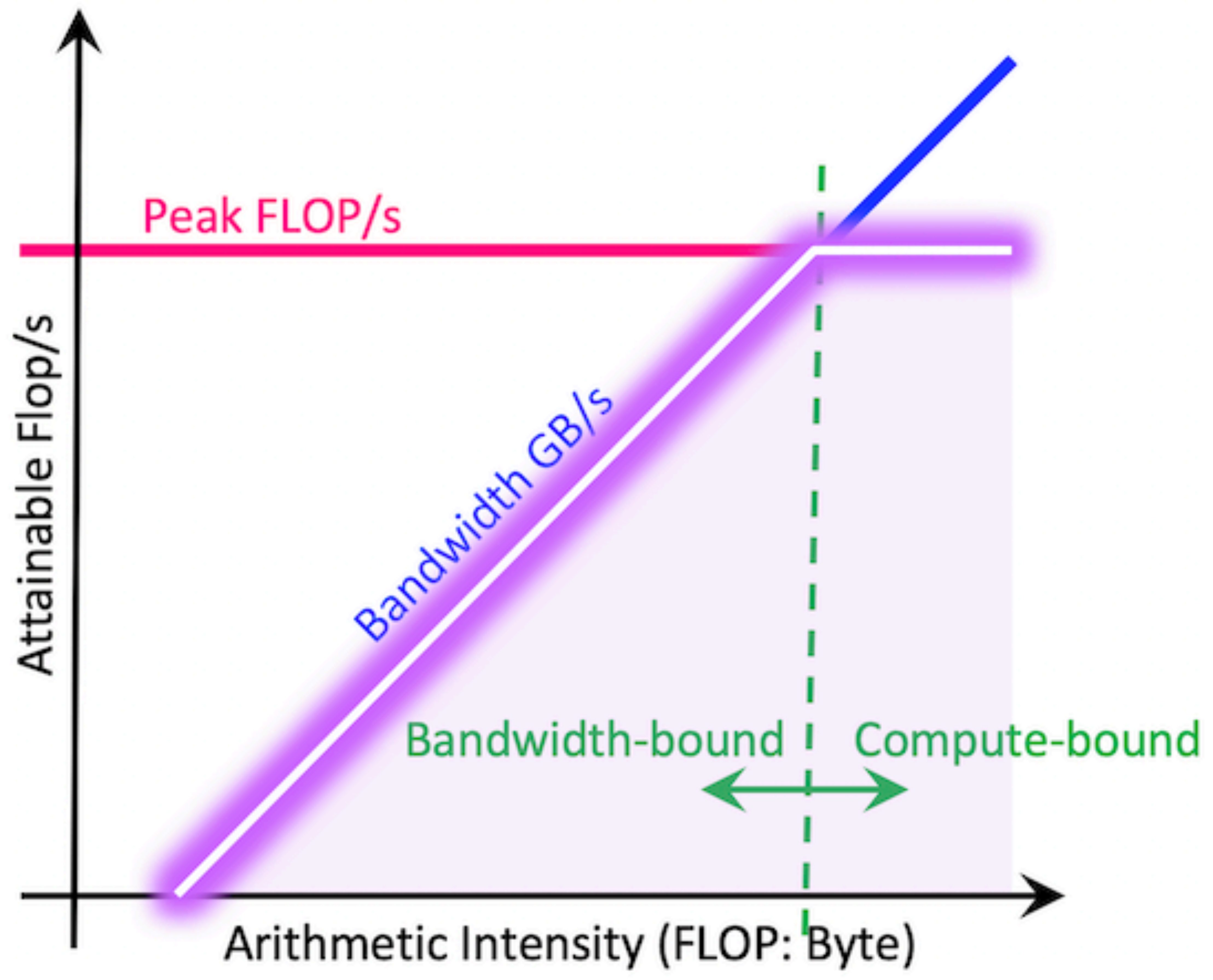
Compute versus memory bound

- Is performance limited by floating point operations or by data movement in memory?
- Important for deciding how to optimize
- Arithmetic intensity: ratio of arithmetic operations to bytes moved (loads/stores)
- What is the best attainable intensity: ratio of peak flop/s to peak memory bandwidth

$$\mathrm{ArIn} = \frac{flop}{bytes\_moved}$$

Roofline model

- Compare achieved arithmetic intensity to best attainable machine intensity
- Best attainable intensity for a hardware is sometimes called the "machine balance"

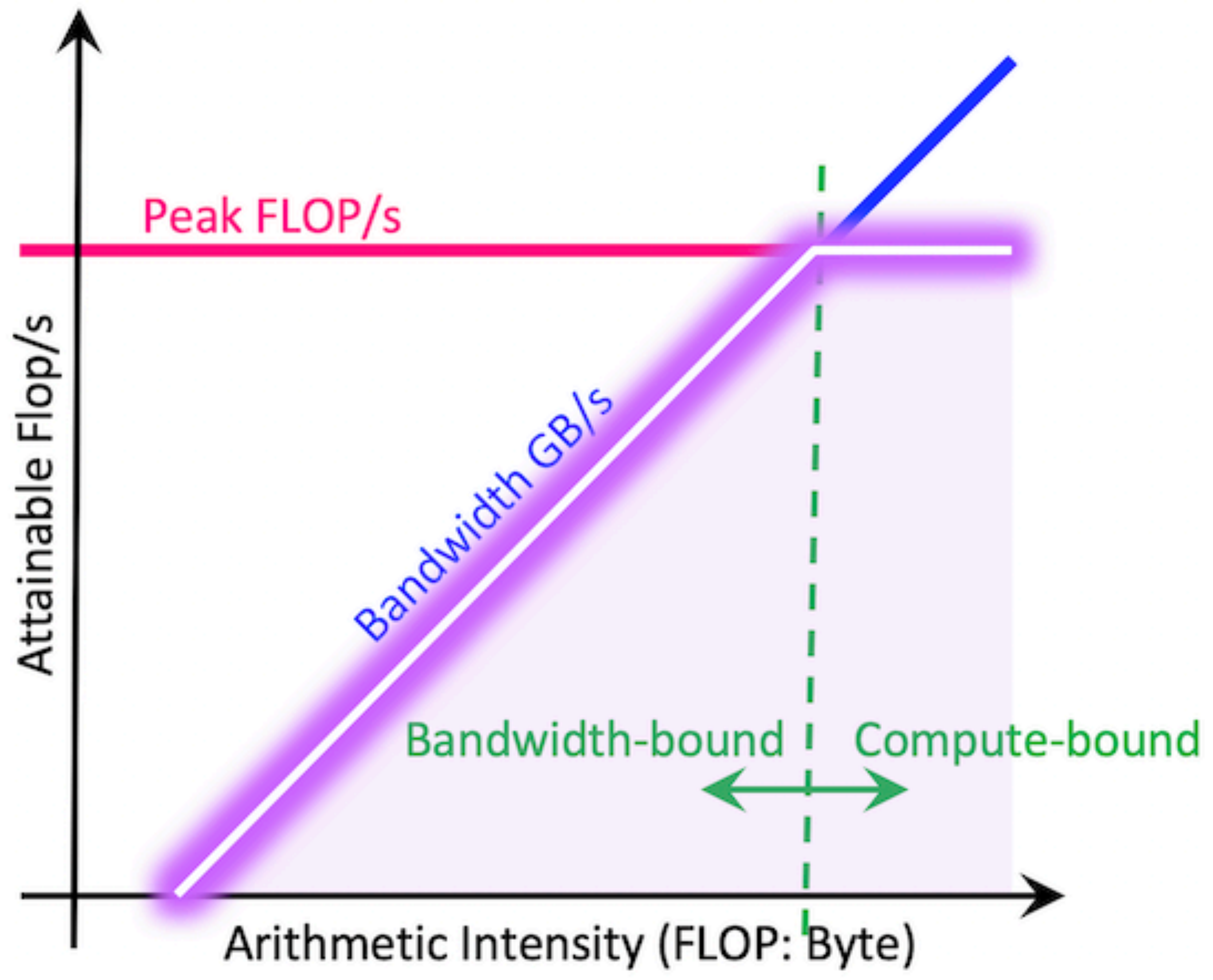


https://docs.nersc.gov/tools/performance/roofline/

Roofline model

- Compare achieved arithmetic intensity to best attainable machine intensity
- Best attainable intensity for a hardware is sometimes called the "machine balance"

Usually applied to individual kernels



https://docs.nersc.gov/tools/performance/roofline/

saxpy

- Flops:
- Bytes moved:

Weak versus strong scaling

- Strong scaling: ***Fixed total*** problem size as we run on more processes
  - Sorting n numbers on 1 process, 2 processes, 4 processes, …
  - Problem size per process decreases with increase in number of processes
- Weak scaling: ***Fixed*** problem size ***per process*** but *increasing total* problem size as we run on more processes
  - Sorting n numbers on 1 process
  - 2n numbers on 2 processes
  - 4n numbers on 4 processes

Speedup and efficiency

- (Parallel) Speedup: Ratio of execution time on one process to that on p processes

$$\text{Speedup} = \frac{t_1}{t_p}$$

- (Parallel) Efficiency: Speedup per process

$$\text{Efficiency} = \frac{t_1}{t_p \times p}$$

Empirical performance analysis

- Two parts to doing empirical performance analysis
  - measurement: gather/collect performance data from a program execution
  - analysis/visualization: analyze the measurements to identify performance issues
- Simplest tool: adding timers in the code manually and using print statements

Performance tools

- Tracing tools
  - Capture entire execution trace, typically via instrumentation
- Profiling tools
  - Provide aggregated information
  - Typically use statistical sampling
- Many tools can do both
- Examples: PyTorch profiler, pyinstrument
- Nsight compute, Nsight systems, rocprof

UNIVERSITY OF
MARYLAND