PatrickStar

Parallel Training System for Big Models via Chunk-based Dynamic Memory Management

Jiarui Fang

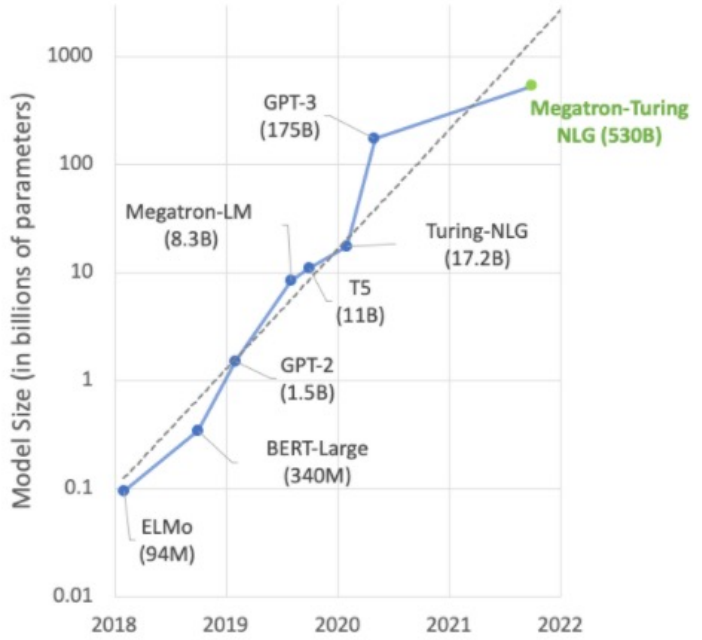
fangjiarui123@gmail.com

2022/06/27

Background: Pre-trained Models (PTM)
A New Paradigm for AI

- Big Model Era (BERT, GPT3)
  - Transformers as backbones are very powerful
  - Pretrain phase: Feature Extracting on Massive Data
  - Finetune phase：Finetuning on Personal Data

- Parameters of PTM grow exponentially
  - Mixture-of-experts (MOE) Models
  - **Dense Models**
    - **More computing/comm. requirements**
    - **More Challenging for System Design**



Background: Pre-trained Models

- A game for a few big players
  - **Economical Cost**:
    - Microsoft used 10,000 V100 Data Center for GPT3(175B) Pretraining , Finetuning used at least 256xV100
    - 1 time GPT3 pretrain cost 12M$
  - **Carbon Cost**:
    - Carbon footprint for training GPT-3 same as driving to our natural satellite and back

- How to Democratize Big Models?
  - Pretrain Phase：Improve hardware **efficiency** to lower the cost
  - Finetuning Phase：Lower hardware **requirement** to reuse pretrained efforts

Background: Pre-trained Models

Memory Wall: (2B model - 32GB GPU memory)

1. model data (**Constant,** numel of param M)

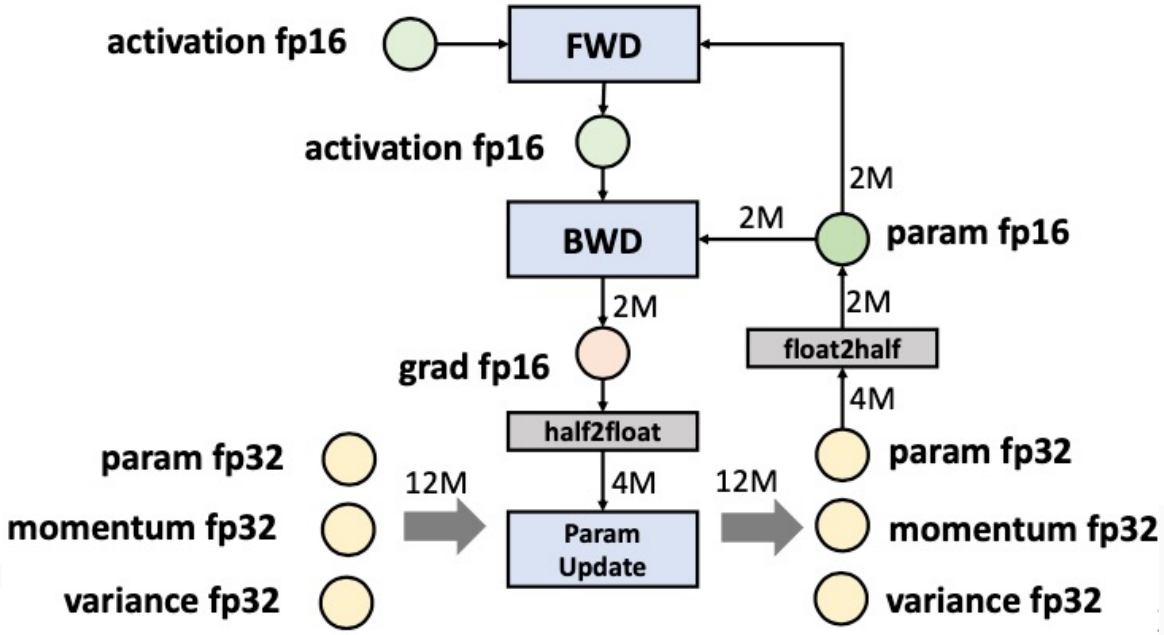
- param fp16 (2M)
- grad fp16 (2M)
- param fp32 (4M)
- momentum fp32 (4M)
- Variance fp32 (4M)

Optimizer States (OS)

2. non-model data (Changing)

- Activations
- Temp. mem. of operators

FWD: forward propagation
BWD: backward propagation



Computing Graph with
Mixed precision training

Backgrounds: Related works for PTM training

- Parallel Training: store model-data in multi-GPU
  - Data Parallel : ZeRO-DP
  - Modell Parallel : Megatron-LM, Mesh-TensorFlow (human efforts)
  - Pipeline Parallel : Gpipe ， PipeDream (bubble Problem)
- Heterogenous Training: store model-data in CPU+GPU+SSD
  - ZeRO-Offload/ZeRO-Infinity, L2L
- Activation Footprint Optimization:
  - Gradient Checkpointing, a.k.a Activation Rematerization
  - Activation CPU Offloading

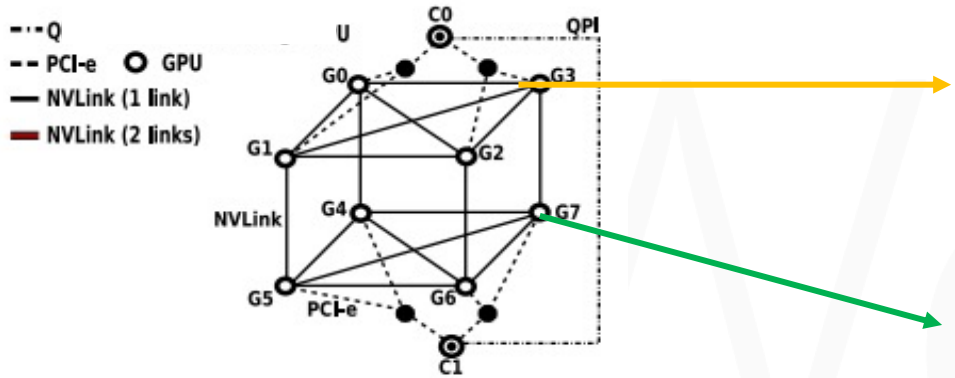
Backgrounds: Related works for PTM training

- Parallel Training: store model-data in multi-GPU
  - Data Parallel : ZeRO-DP
  - Modell Parallel : Megatron-LM, Mesh-TensorFlow (human efforts)
  - Pipeline Parallel : Gpipe ， PipeDream (bubble Problem)
- Heterogenous Training: store model-data in CPU+GPU+SSD
  - ZeRO-Offload/ZeRO-Infinity, L2L
- Activation Footprint Optimization:
  - Gradient Checkpointing, a.k.a activation rematerization
  - Activation CPU Offloading

deepspeed

Backgrounds: Related works for PTM training

- PatrickStar focuses on **Heterogenous Training**
  - Economical: CPU memory is cheap, GPU memory is expensive
  - Effective: Hierarchical memory similar to classical computer architect idea (virtual memory, multi-level cache, etc.)
  - Proven: DeepSpeed stage3 (Zero-Offload) vs. stage2 (zeroDP) claimed 10x model scale
- Hardware: a typical CPU-GPU cluster used for training



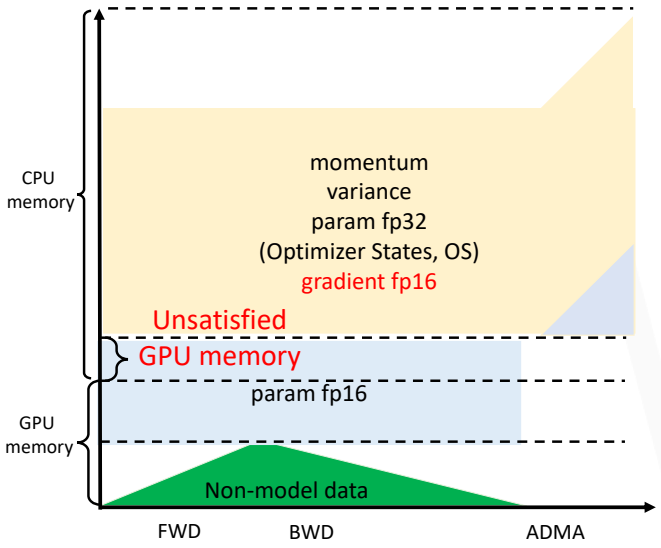
PCI-e：CPU-GPU
P2P peak bandwidth 13GB/s，message size >4MB

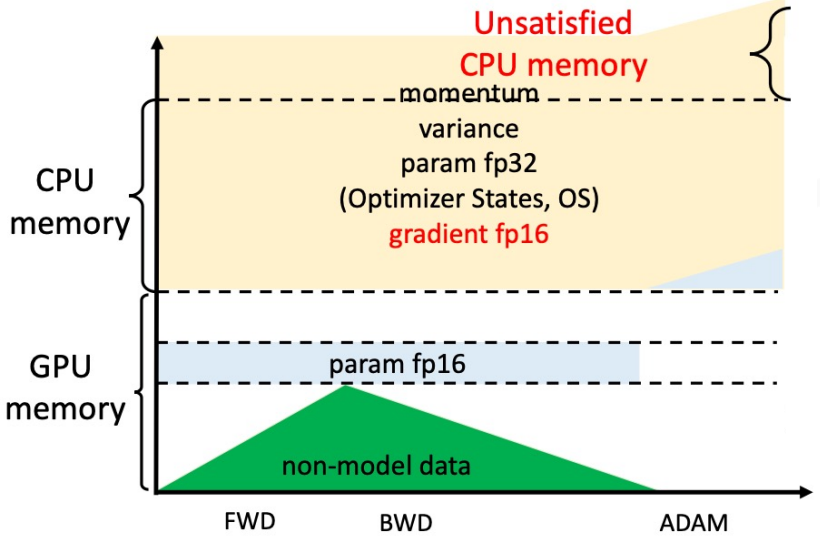
NVLink：inter-GPU
Collective peak bandwidth 60-120 GB/s，message size > 16MB

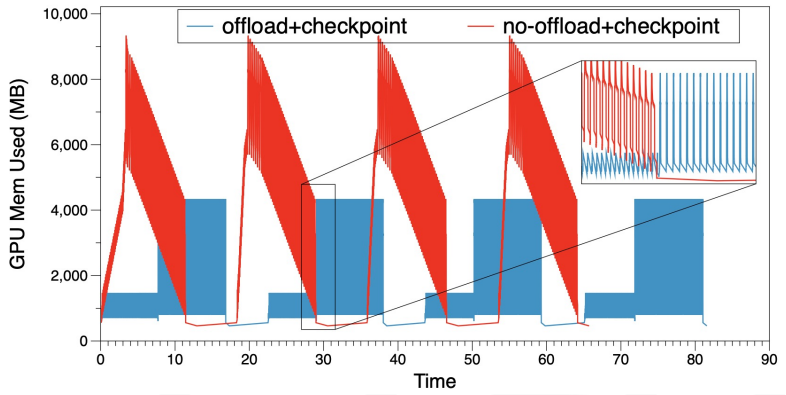
A typical network topology, 1-CPU-multi-GPU

Motivation: Flaws of the ZeRO-offload in DeepSpeed

- Design: statically partition model data on CPU and GPU in advance of training.
  - param fp16 (2M) on GPU; OS+grad fp16 (14M) on CPU
  - transmit grad fp16(2M) GPU->CPU during FWD+BWD
  - transmit param fp16(2M) CPU->GPU after ADAM
- Suboptimal computing and memory efficiency.
  - 4xV100 1.5TB DRAM DGX-2H: **30B** model (25% of peak comp. performance)
  - 4xV100 240GB DRAM YARD: scale to **6B** model only (41% of peak comp. perf.)







real non-model data changing curve

Design Overview: Chunk-based dynamic memory mgr.

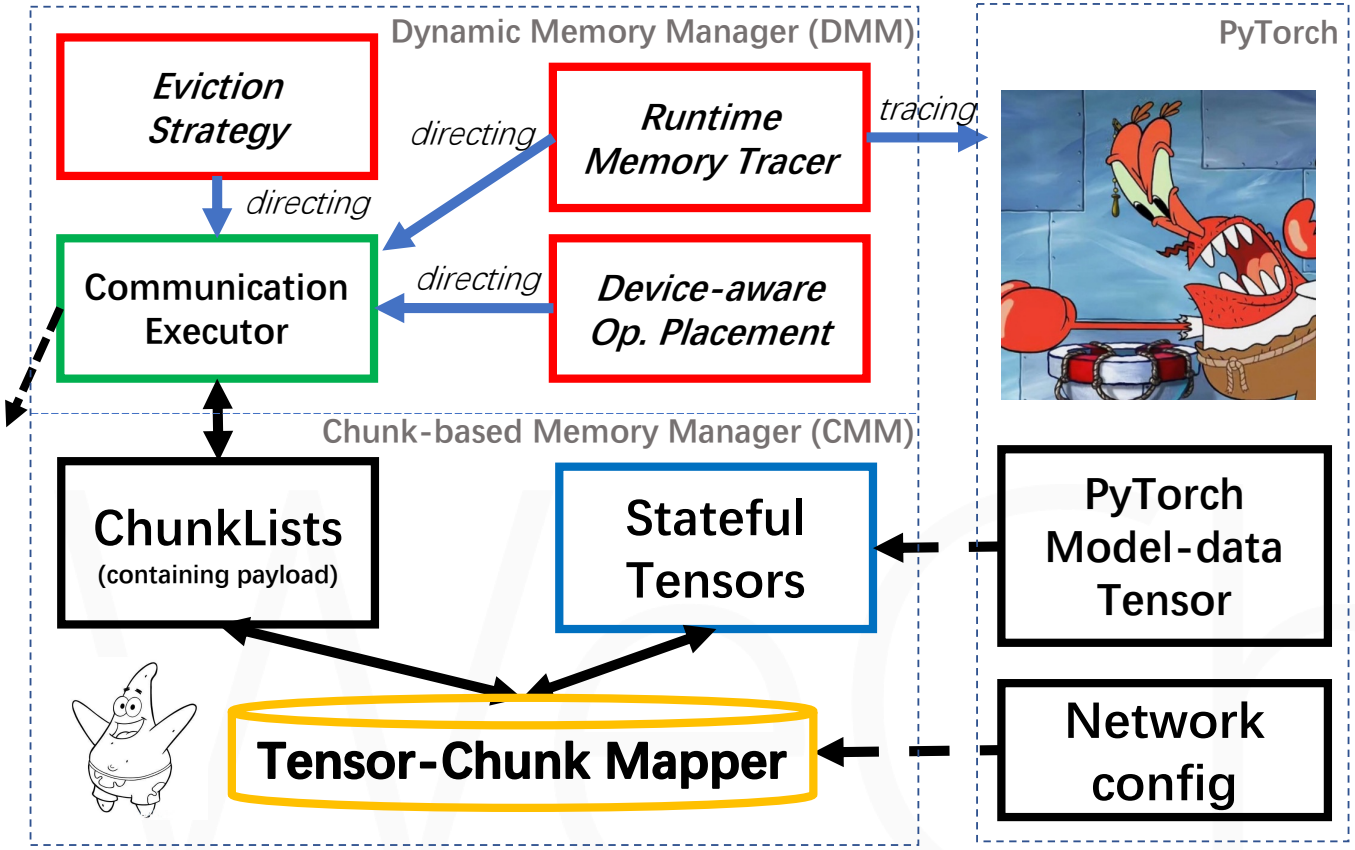
- **Idea**: Arrange model data tensors into Chunks and dynamically adjust their CPU-GPU layouts, similar to *virtual memory*
- **Chunk-based vs** Transmitting in Tensors:
  - Chunk: a block of memory contains the same number of elements. e.g. 128 M.
  - Large message size ensures high PCI-e/NVLink bandwidth utilization
- **Dynamic vs** Static Partition**:**
  - **Real-time** adjustment of model data layout according to the near future memory situations.
  - If GPU/CPU memory is about to overflow, tensors not in use are evicted to CPU/GPU

Software Architecture

- **PatrickStar**: the porter at Krusty Krab
- PyTorch-Mr. Krabs; CPU/GPU-containers; Tensor-Item; Chunk-Box;

Hands: Moving Chunks (Boxes) around Contains (CPU/GPU Memory)

CPU+ GPU Memory Space



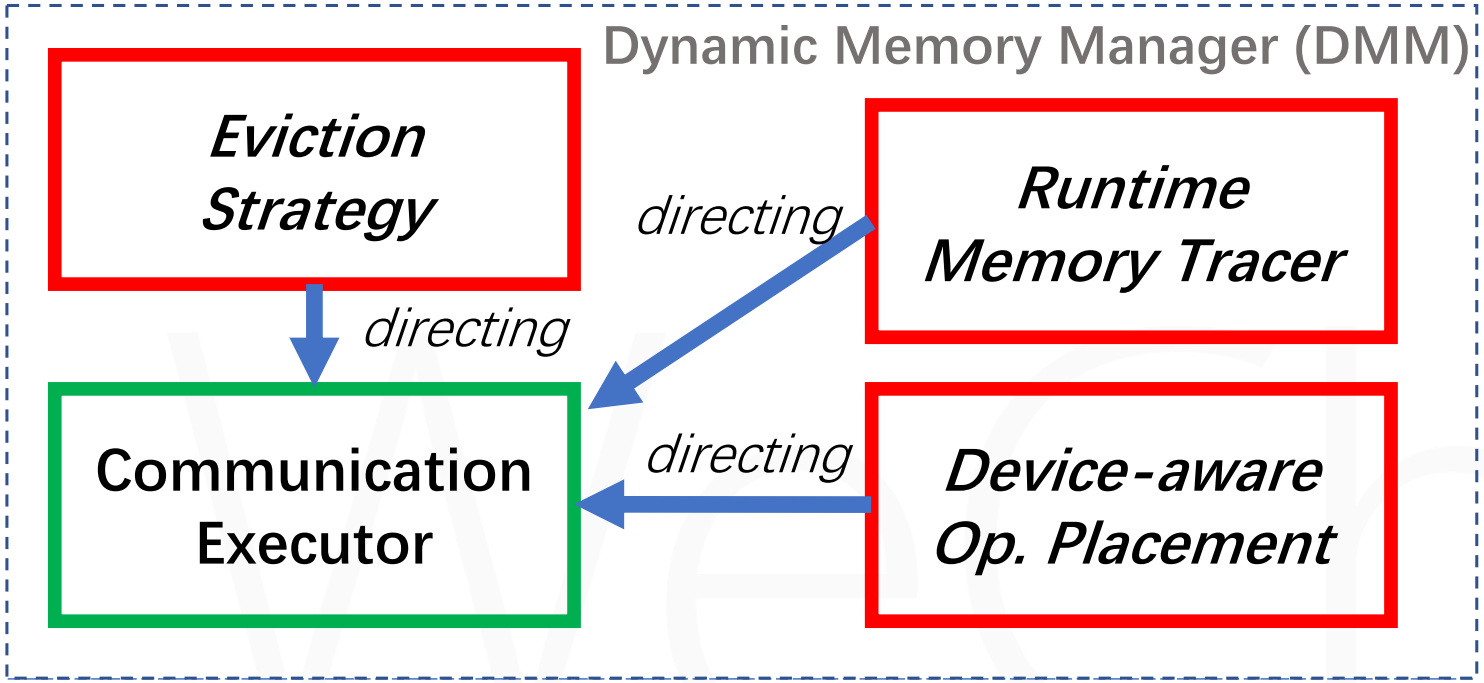
Manual: Optimal work procedure. Help the porter saving energy.

Packing Info：A map for Tensor (Items) and Chunk (Box)

Item Info: Records Item (tensor) states during training (store opening)

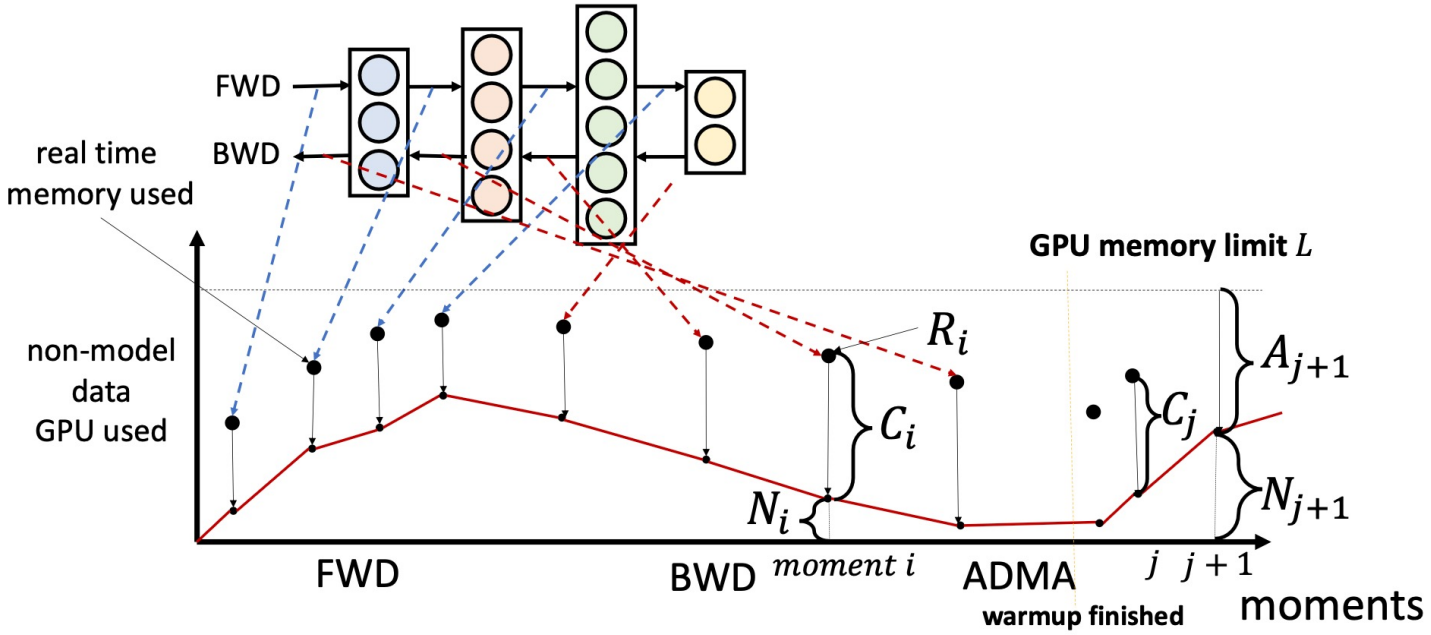
Dive into the DMM: Dynamic Memory Management

- Target: Adjust model-data layout **timely**, before each Op. execution.
- **Runtime Mem. Tracer**: Depict non-model data volume changing.
- **Eviction Strategy:** Evict tensors smartly.
- **Device-aware Op Placement:** Place Op. on CPU/GPU smartly.



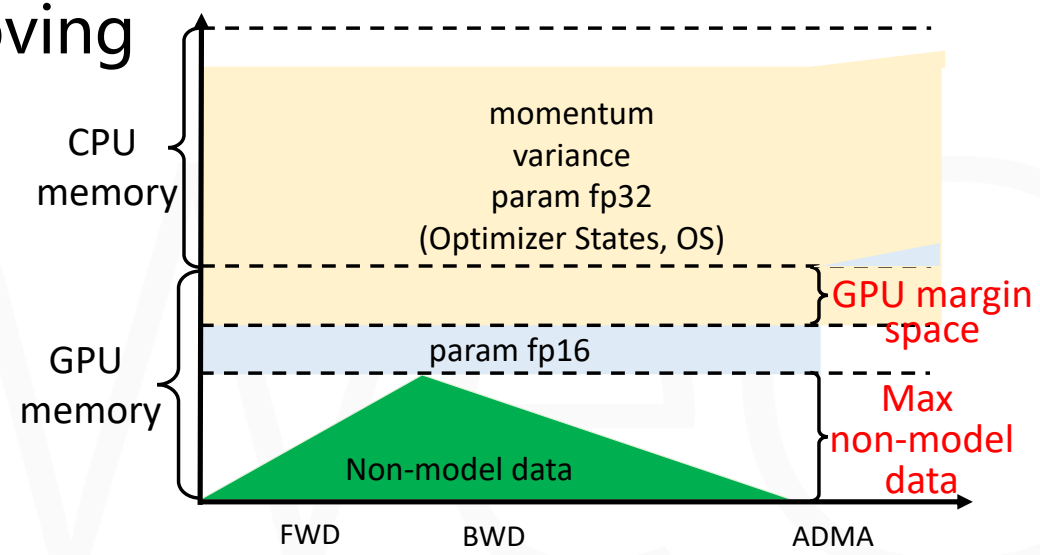
DMM: Runtime Memory Tracer

- Tracing non-model data changing in warmup iterations:
  - model-data and non-model data compete for memory resources.
  - **Warmup**: non-model data (N) = memory usage (R) – allocated model data (C)
  - **Non-warmup**: available model data (A) = overall mem. (L) – non-model data (N)



DMM: Device-aware Operator Placement

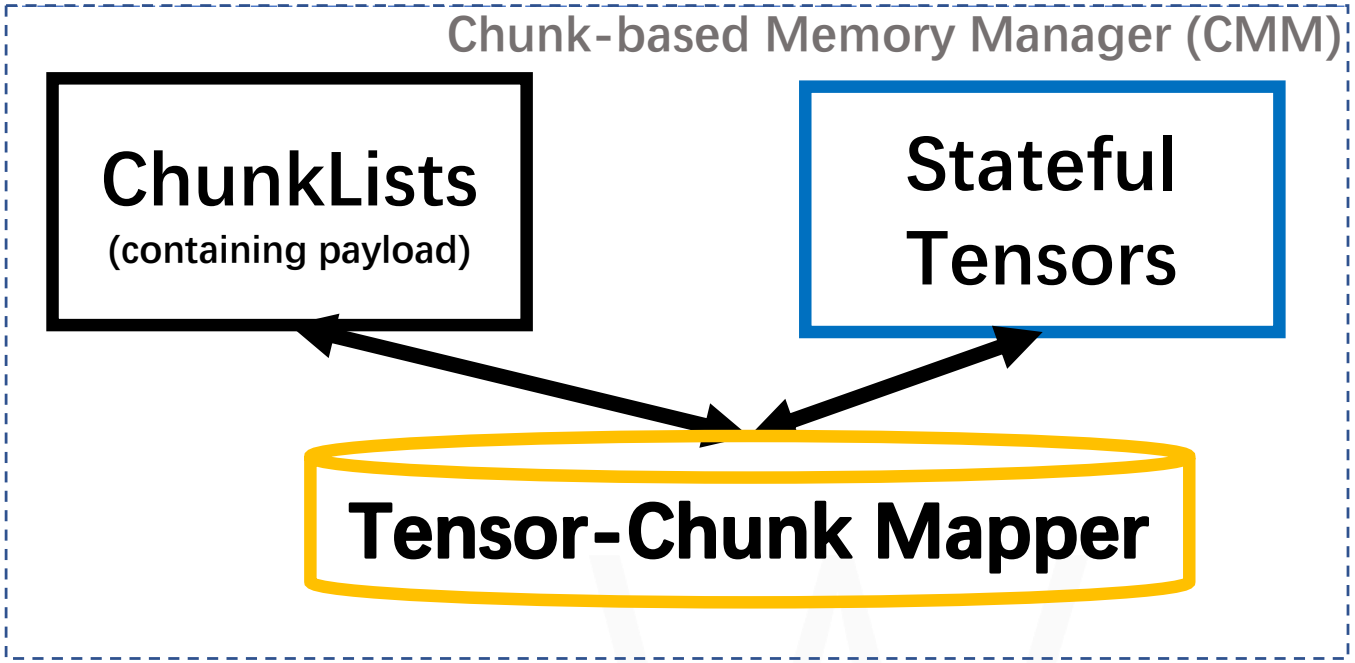
- Operators in DNN can be classified into two categories
  - Compute-intensive：nn.Linear（must run on GPU）
  - Memory-intensive：ADAM（CPU/GPU）
- Smartly layout memory-intensive Ops
  - GPU Margin Space = GPU mem. size. – (peak non-model data + param fp16 model data)
  - Put OS tensors on GPU as much as possible to avoid updated param fp16 data moving

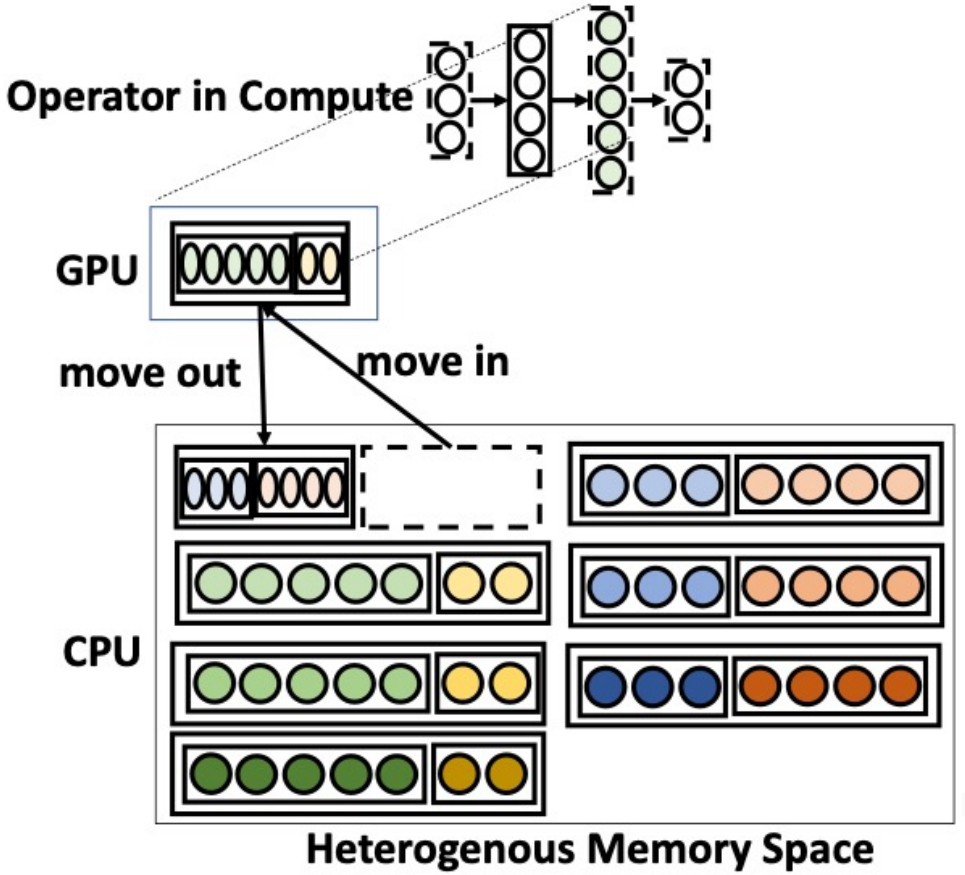


DMM: Eviction Strategy

- Adjust model-data layout : Prepare enough memory space to satisfy need of the operator execution.
- If memory is full, how to evict not-in-used model data tensors?
  - Evict the longest future reference tensors on this computing device, since we know exactly when each tensor is in use.
  - Belady ‘s OPT algorithm in Cache design

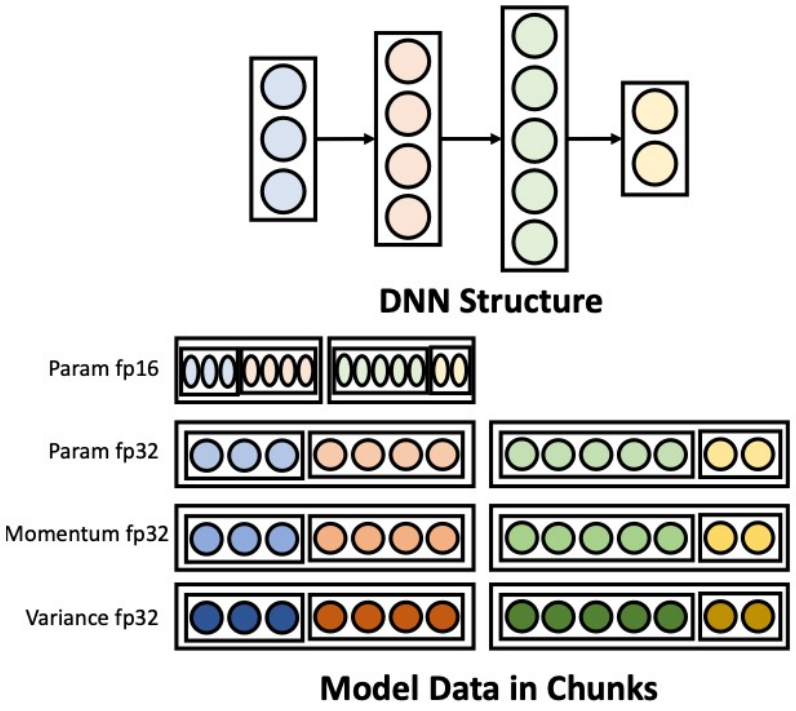
Dive into the CMM: Chunk-based Memory Management





CMM：Chunk-Tensor-Mapper

- Chunk-Tensor-Mapper：Generated before training
- 4xChunkList: param fp16, param fp32, variance, momentum
  - The tensors are arranged in the initialization order
  - Avoid tensors of the same Operator reside on 2 different chunks



	YARD			SuperPod			
Model	10B	15B	18B	20B	40B	60B	68B
SIZE (M)	288	480	312	288	288	416	416
UTIL.(%)	94.47	92.62	91.5	90.5	90.6	92.2	97.4

A light-weight tool search best chunk size for different models

CMM : Stateful Tensor

- State implies the possible location of chunks in CPU+GPU memory.
  - A tensor is stateful, belonging to one of 5 states.

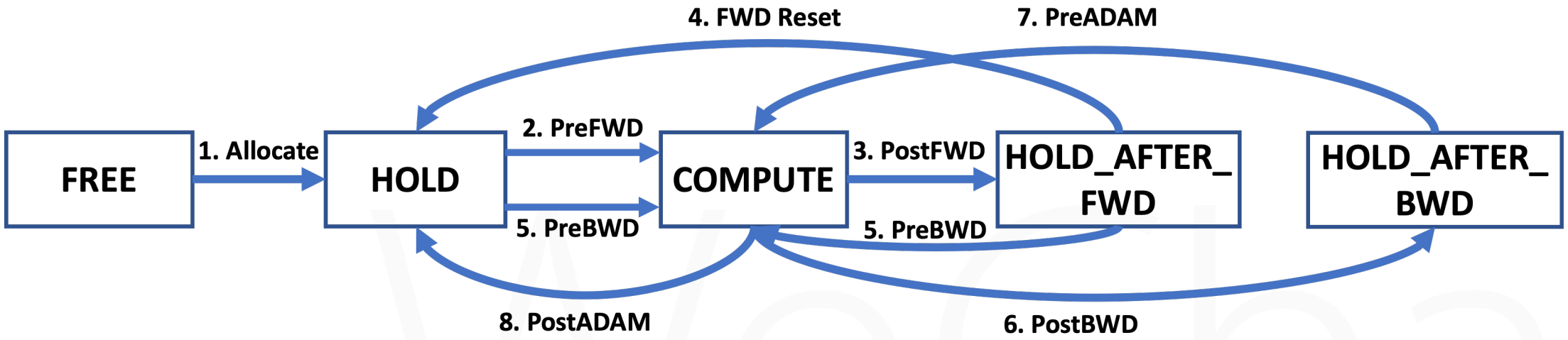
STATE NAME	EXPLANATION	PLACEMENT
FREE	No payload space	-
COMPUTE	Participate in computing	Computing Device
HOLD	Hold payload	CPU or GPU
HOLD_AFTER_FWD	Hold payload after FWD	CPU or GPU
HOLD_AFTER_BWD	Hold payload after BWD	CPU or GPU

HOLD-like state

  - State transition is triggered before every operator execution
  - The state of a Chunk is determined by the state of all Tensors it manages
    - ∀(FREE): If all tensors are FREE, memory can be reused by other chunks or be released
    - ∃(COMPUTE): If One of its tensor is COMPUTE, the Chunk must be placed on the target device
    - ~∃(COMPUTE) ∩ ∃(HOLD-like) ： If there is no COMPUTE tensor and there is a tensor that is HOLD-like, the Chunk can be freely placed in either CPU or GPU memory

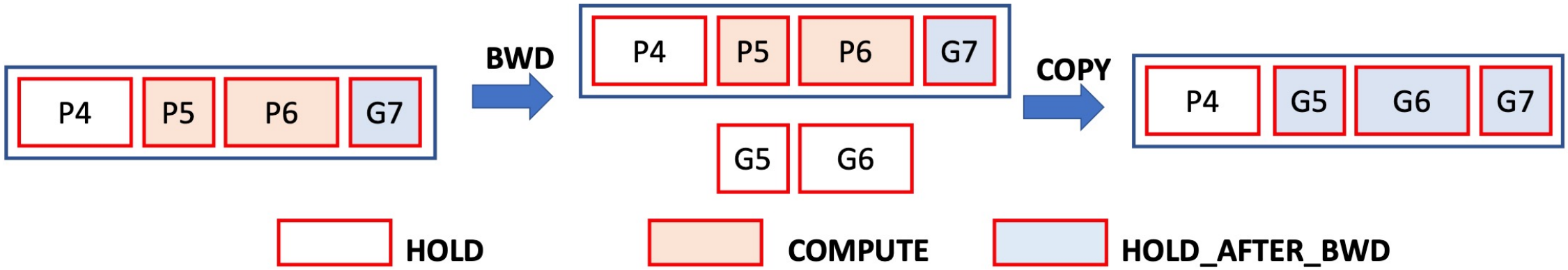
CMM : Tensor State Transition

- Tensor works as a finite-state machine
  - 5 states are necessary, since activation rematerization will introduce FWD on part of operators during the BWD phase.



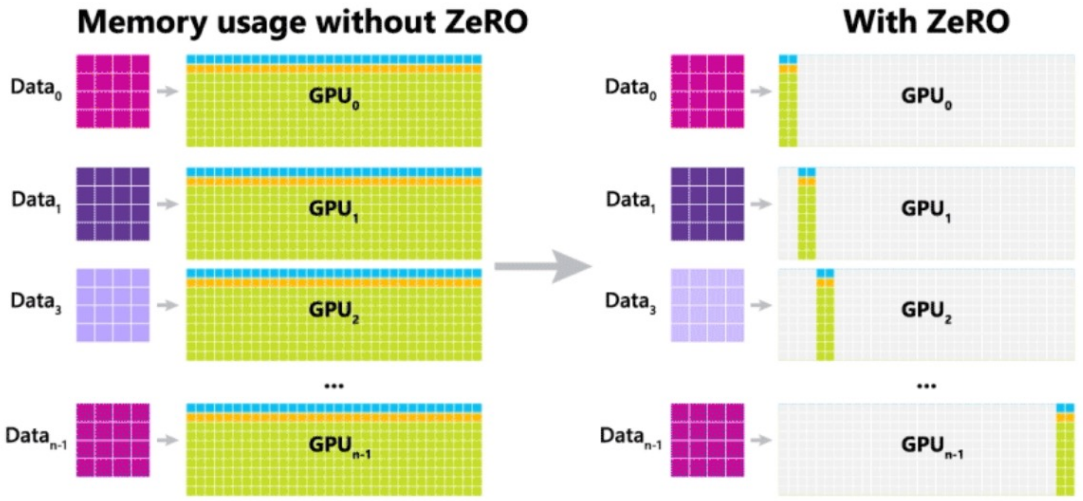
CMM: Chunk Reuse

- Chunk reuse eliminates grad fp16 ChunkList memory footprint
  - Transformer Models are linear-structured
  - Grad fp16 has no life-cycle overlap with Param fp16
- Peak model data footprint $14M < 18M$ (DeepSpeed)



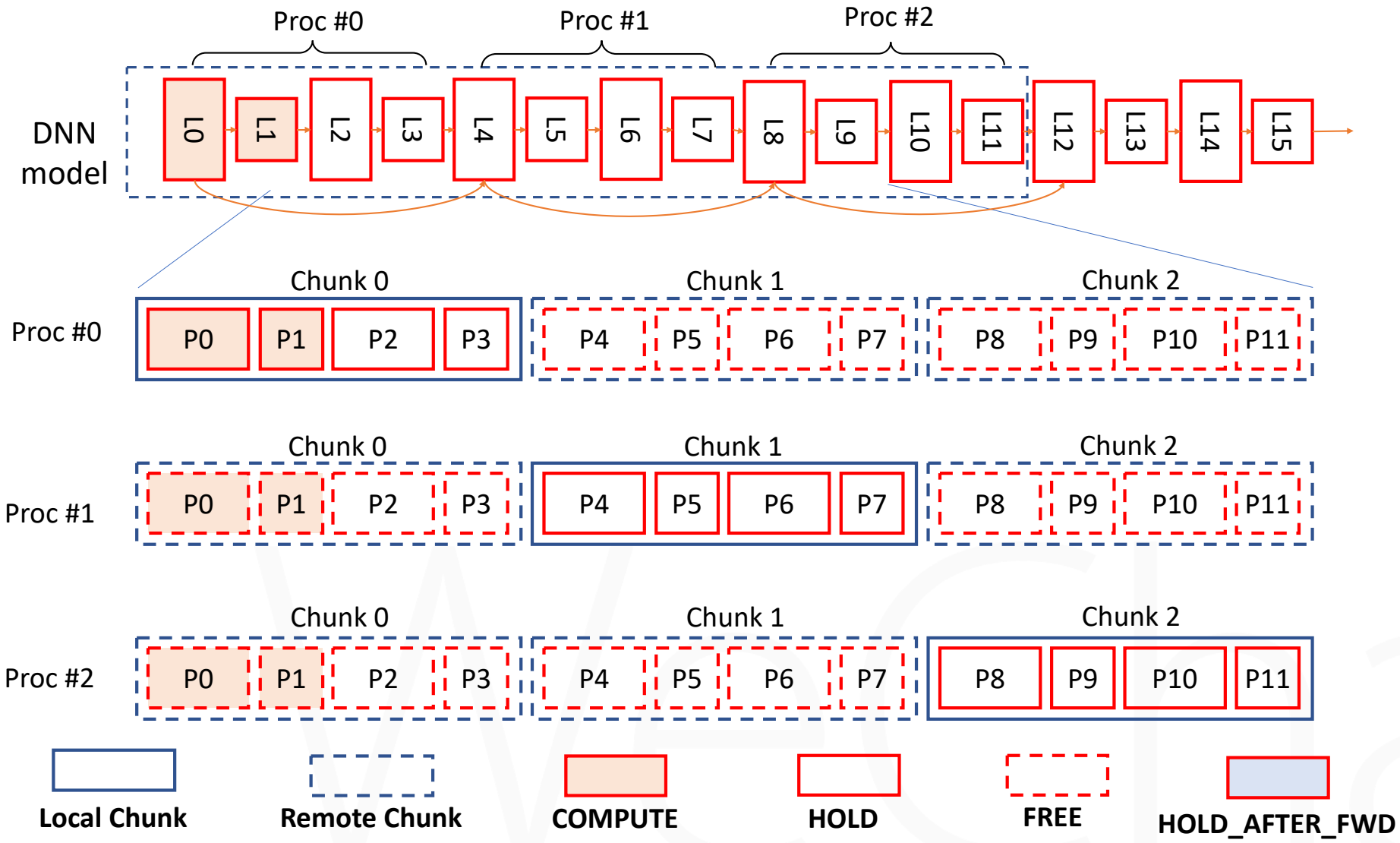
CMM: Scaling To Multiple GPUs

- CMM is compatible with ZeRO-DP (SC 20)
  - Multiple-processing: each process in charge of a GPU
  - Training works as DP, but the param are **sharded** among processes.
  - Need extra gather communication to collect remote shards.
  - **DeepSpeed**： Communicating in granularity of tensors.
  - **PatrickStar**：Communicating in Chunks

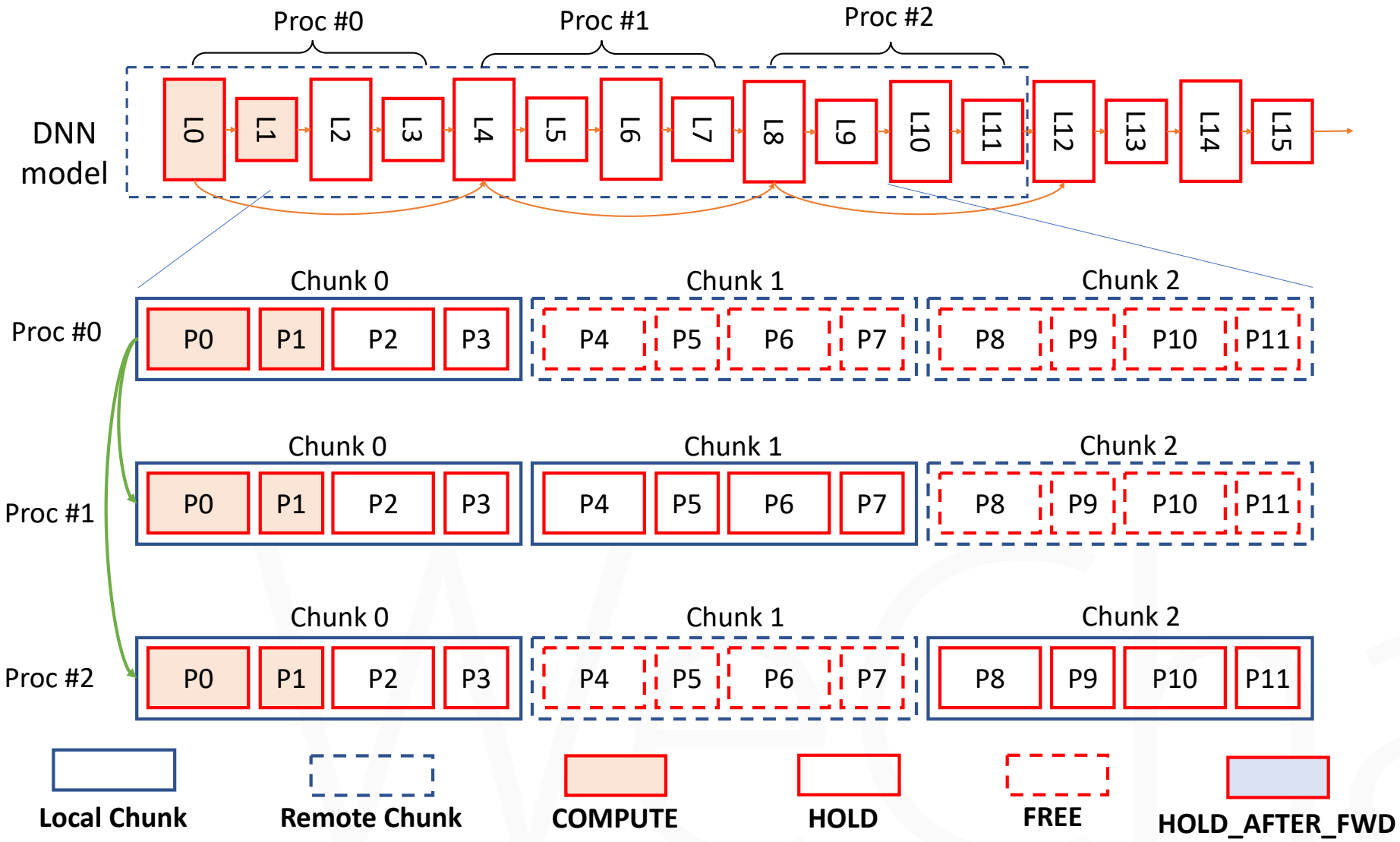


Picture Courtesy of Microsoft Blog

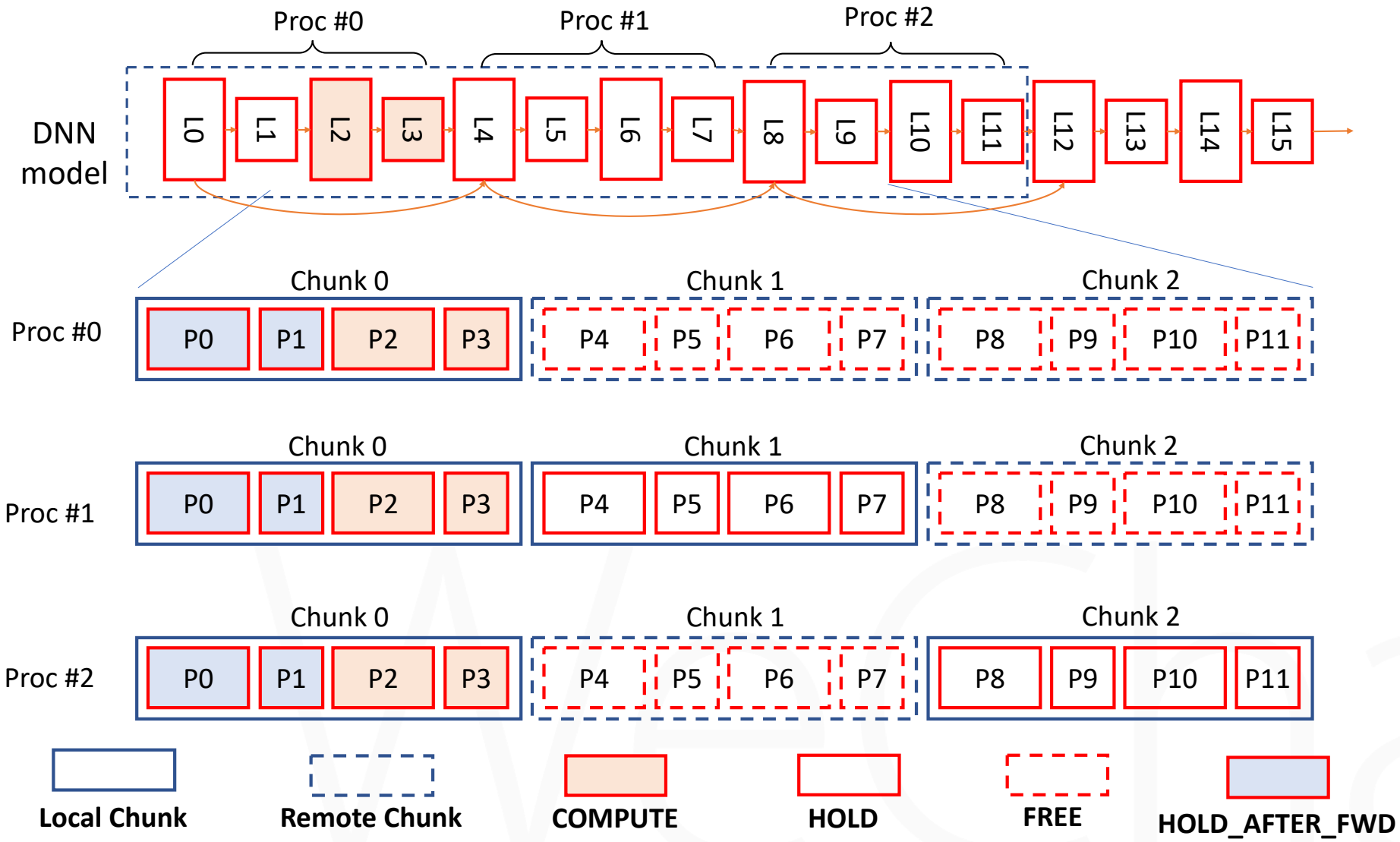
Pre Op0 FWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
Proc #1
Proc #2
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_FWD

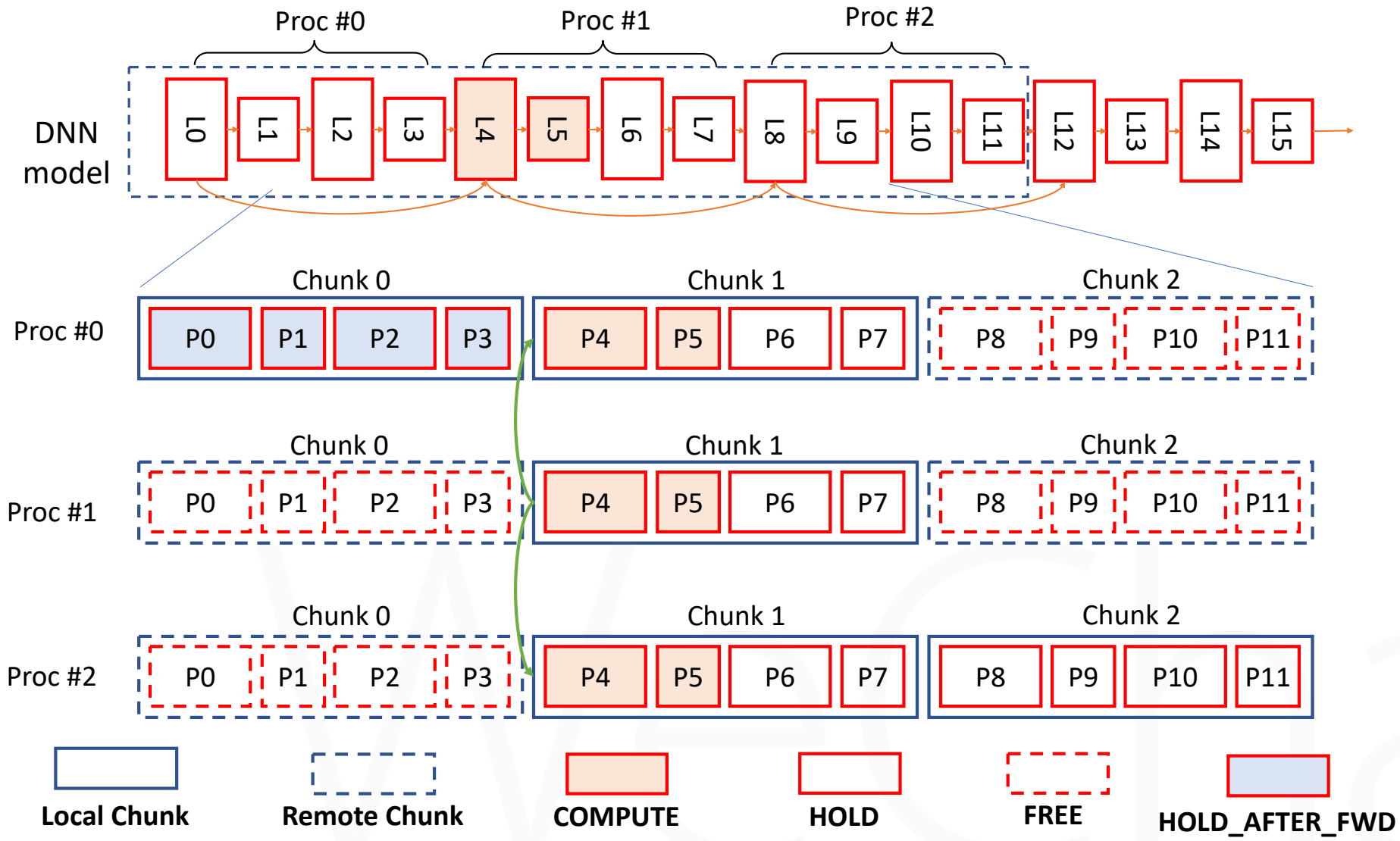
Op0 FWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
Proc #1
Proc #2
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_FWD

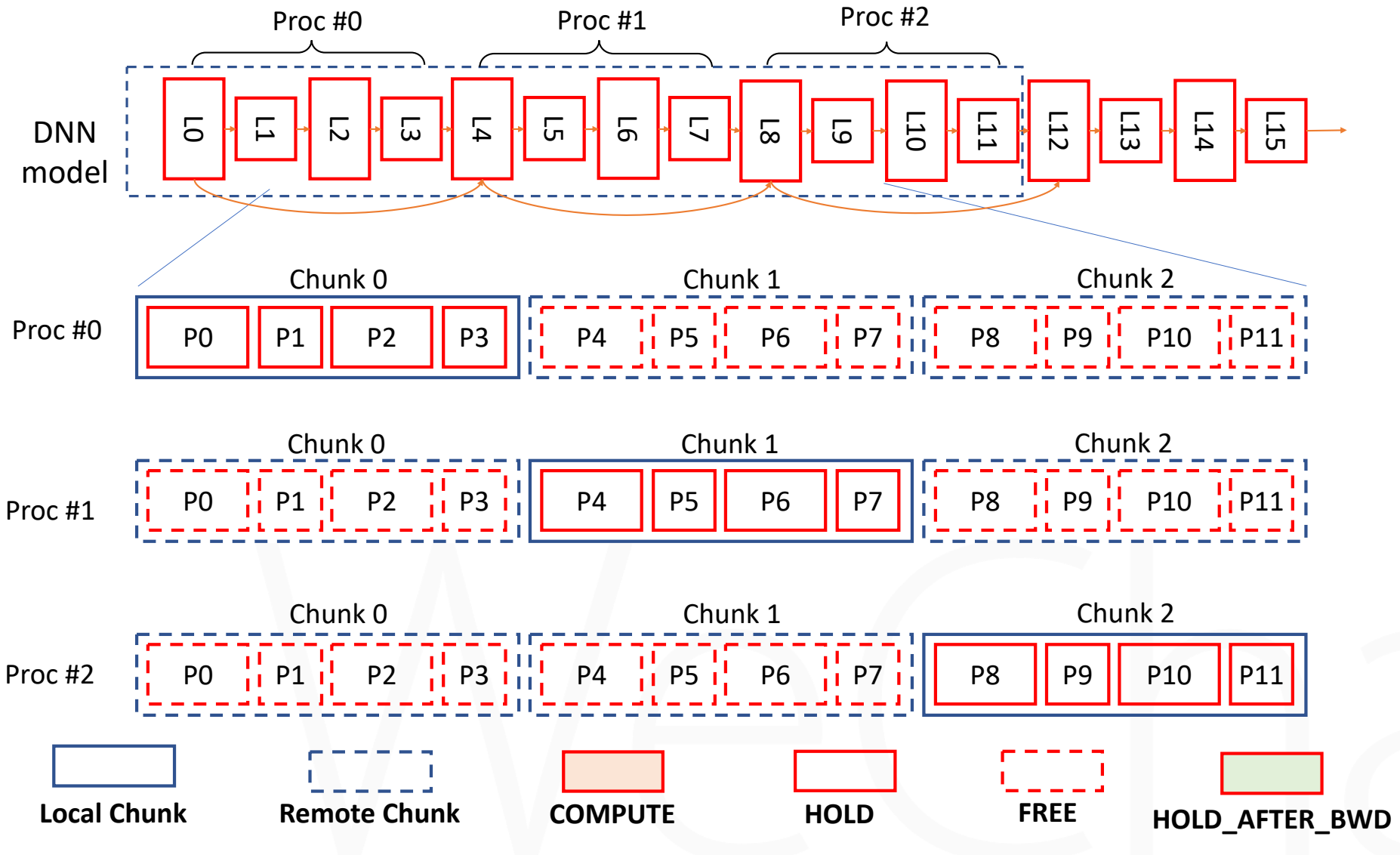
Op1 FWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
Proc #1
Proc #2
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_FWD

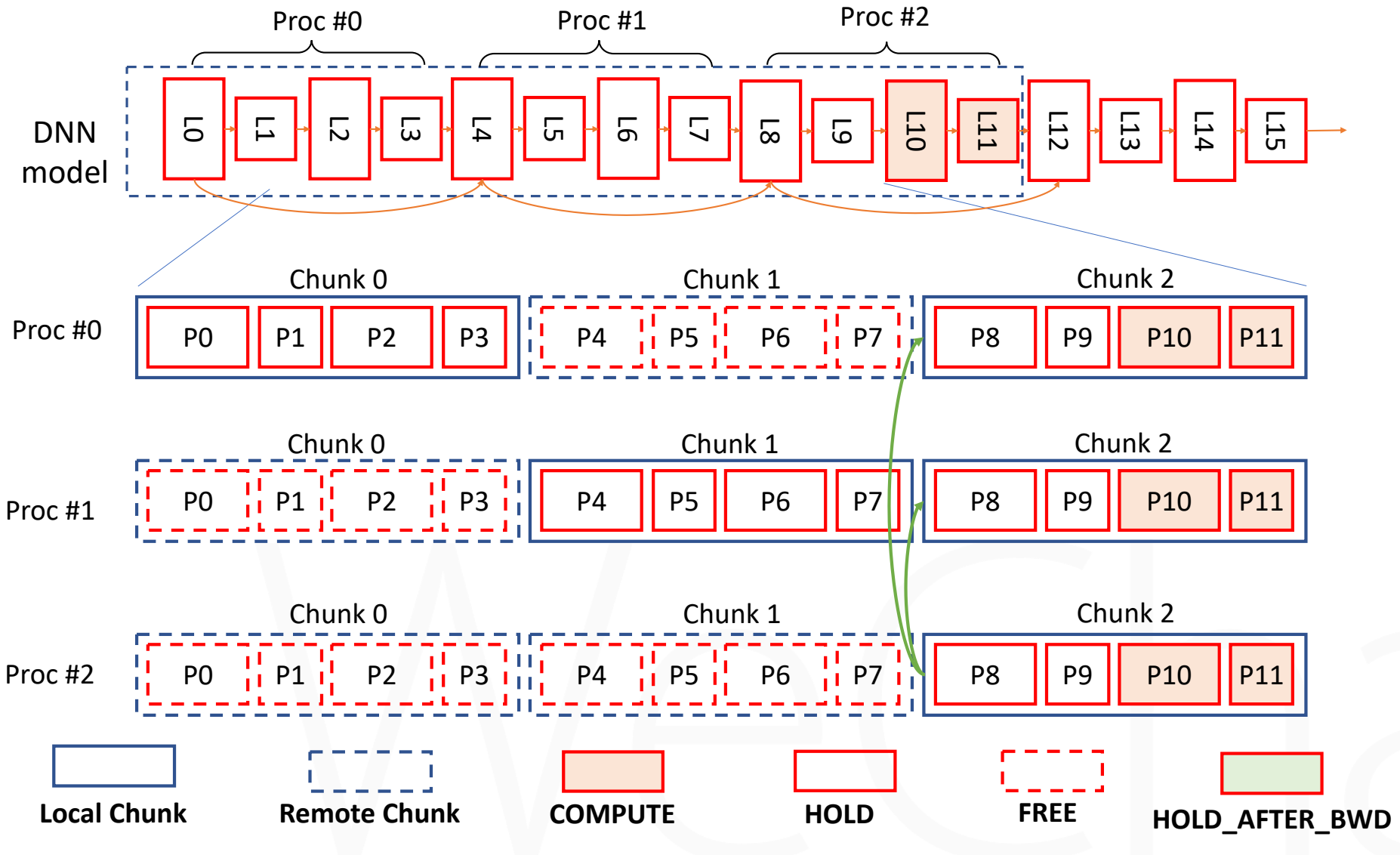
Op2 FWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
Proc #1
Proc #2
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_FWD

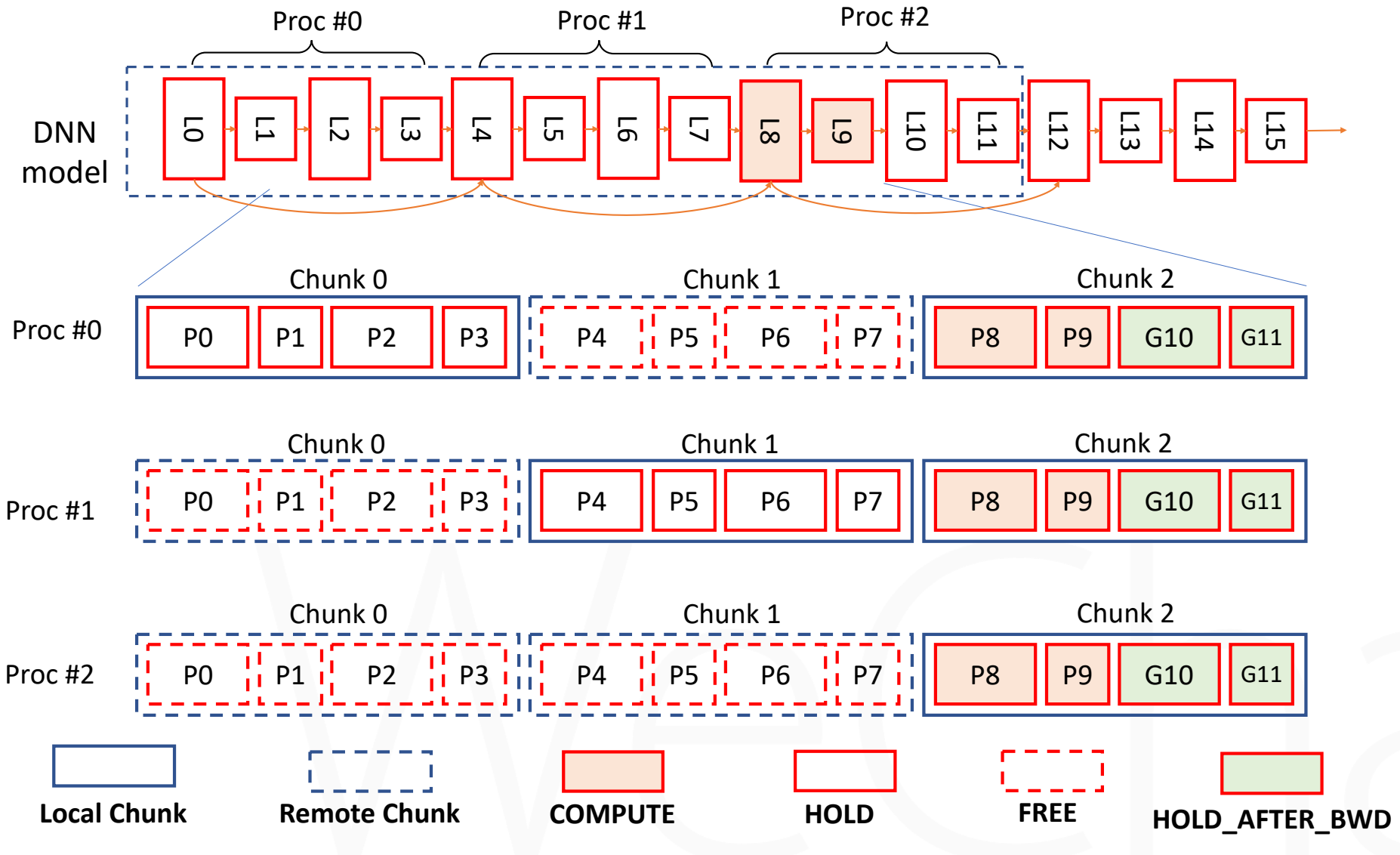
Pre Op5 BWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
Proc #1
Proc #2
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_BWD

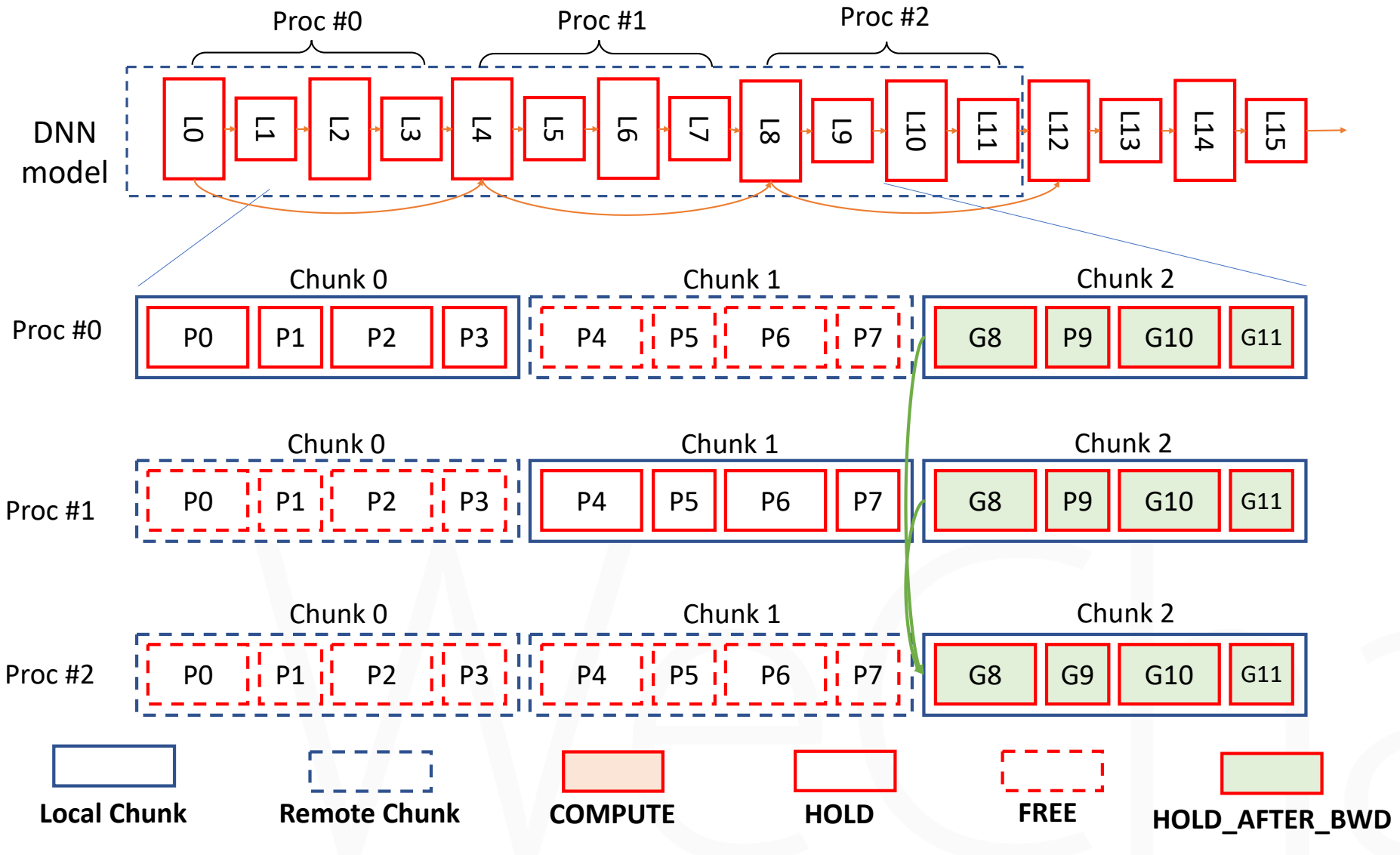
Op5 BWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
Proc #1
Proc #2
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_BWD

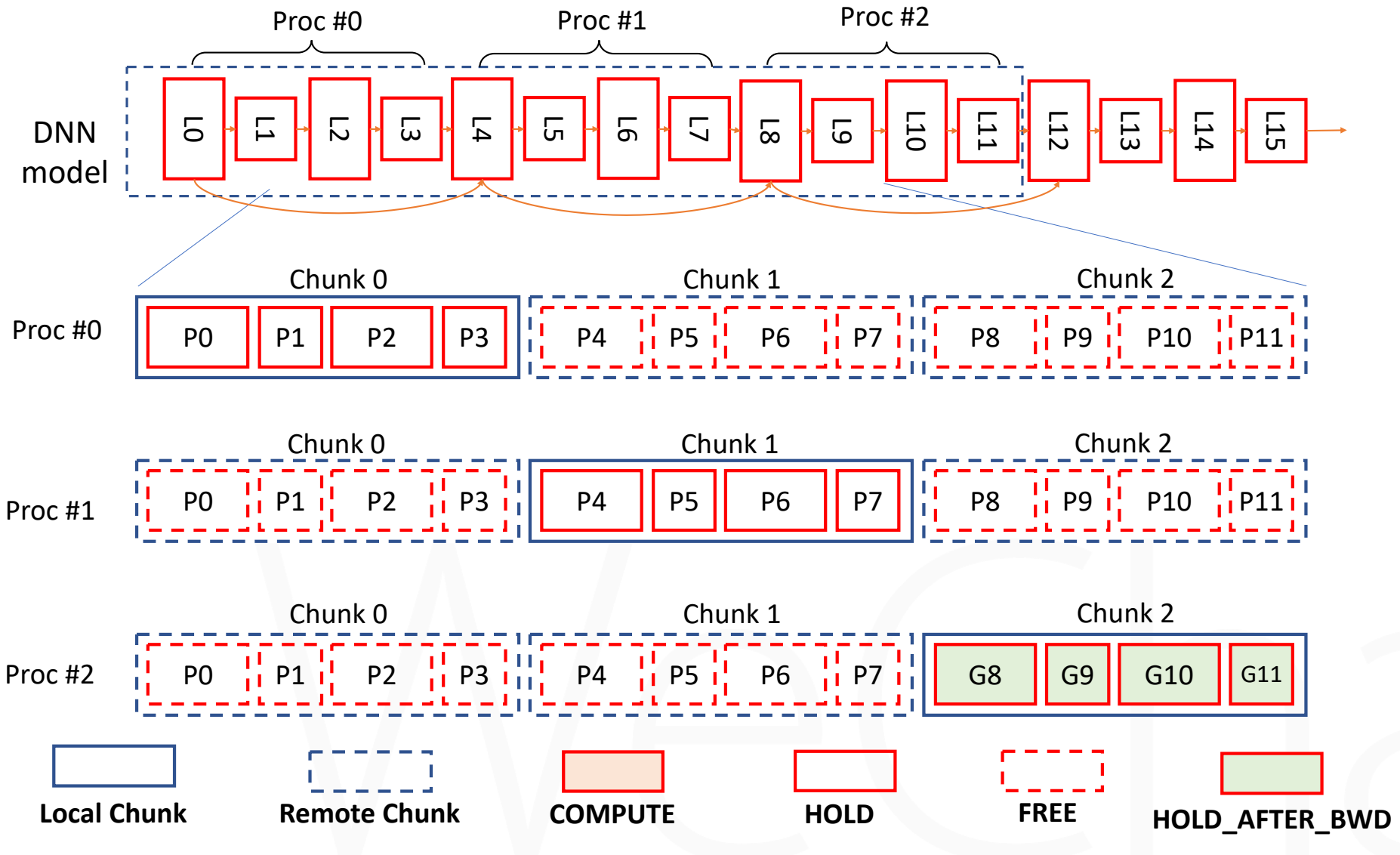
Op4 BWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
G10
G11
Proc #1
Proc #2
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_BWD

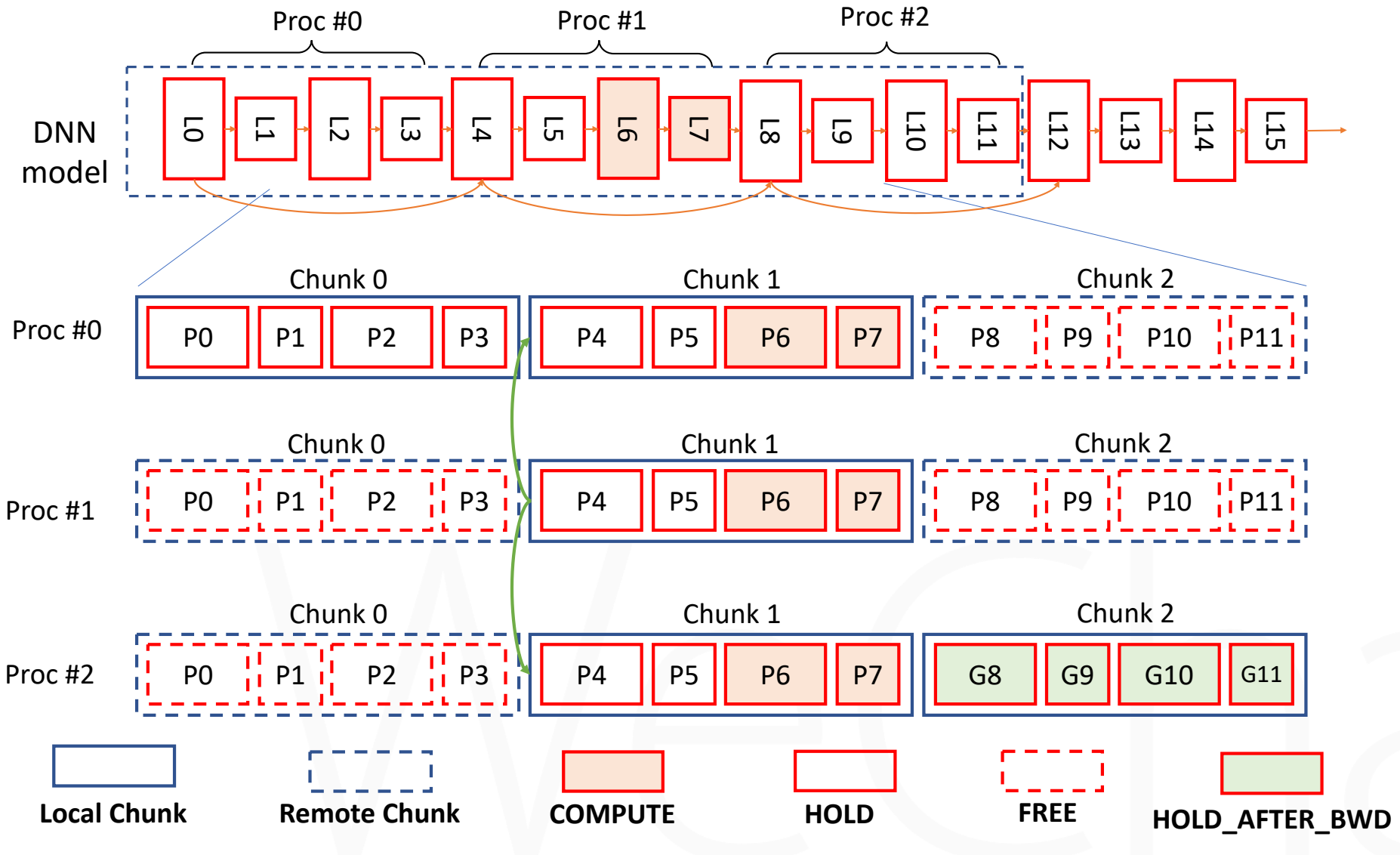
Op4 BWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
P0
P1
P2
P3
P4
P5
P6
P7
G8
P9
G10
G11
Proc #1
Proc #2
G9
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_BWD

Pre Op3 BWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
Proc #1
Proc #2
G8
G9
G10
G11
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_BWD

Op3 BWD


Proc #0
Proc #1
Proc #2
DNN model
L0
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12
L13
L14
L15
Chunk 0
Chunk 1
Chunk 2
Proc #0
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
Proc #1
Proc #2
G8
G9
G10
G11
Local Chunk
Remote Chunk
COMPUTE
HOLD
FREE
HOLD_AFTER_BWD

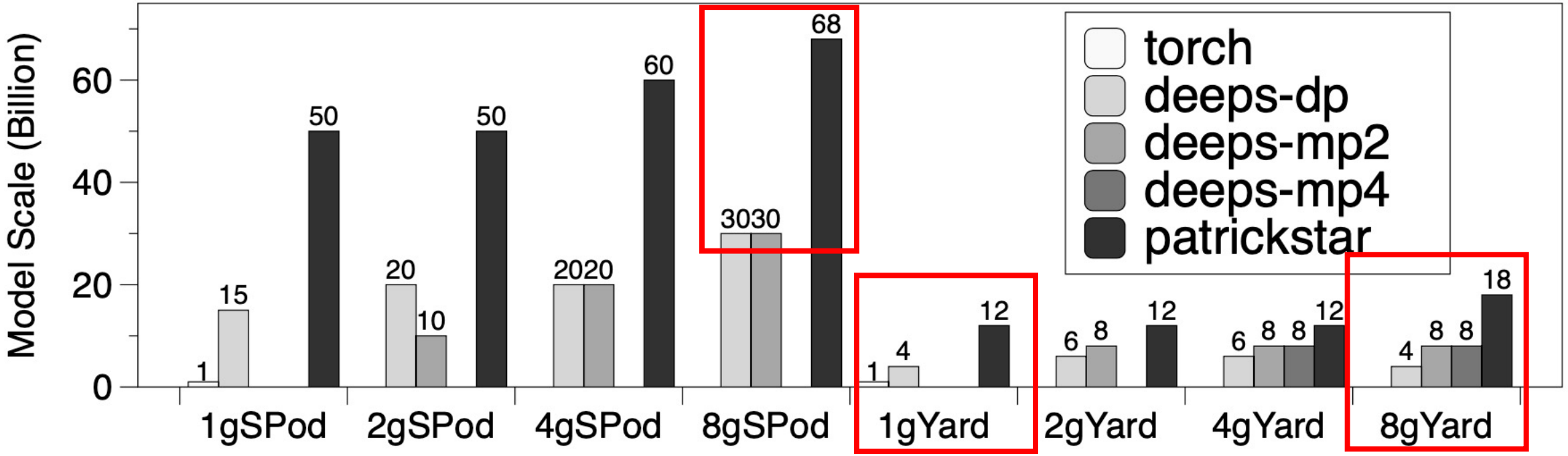
Evaluation: Methodology

- Testbed:
  - Tencent YARD cloud: 8x40GB V100 240GB DRAM 12-core CPU
  - SuperPod: 8x32GB A100 1TB DRAM 192-core CPU per node(8x)
- Baseline
  - PyTorch Distributed-DataParallel (DDP)
  - DeepSpeed stage3 official example (DP+MP hybrid parallelism)
- Model Config: 1B~68B Parameters (huggingface gpt)

#params	#layer	hidden dim	#params	#layer	hidden dim
1,2 B	20,40	2048	6,8B	53, 72	3072
4B	64	2304	10, 12	78, 90	4096
15, 18B	50, 60	4096	20, 30B	25, 37	8192
40, 50, 60B	50, 62, 75	8192	68B	66	9126

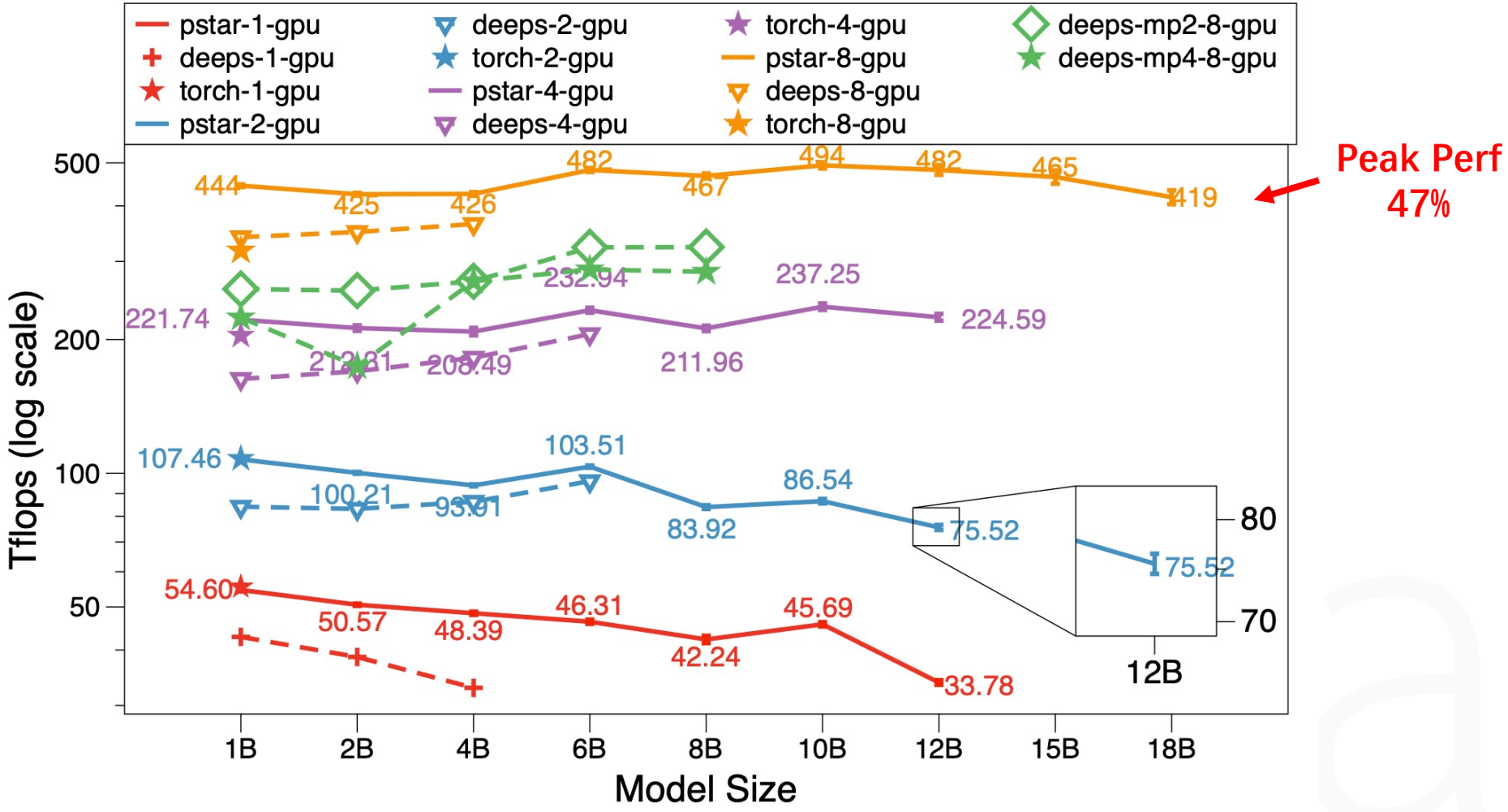
Results: Model Scale

- YARD-1xGPU: PatrickStar vs. DeepSpeed(**3x**), vs. PyTorch DDP (**12x**)
- YARD-8xGPU: PatrickStar vs. DeepSpeed 8B to 18B (**2.25x**).
- SuperPod-8xGPU: PatrickStar vs. DeepSpeed 30B to 68B (**2.27x).**
- **Memory efficiency : 18B@YARD** 18×14=252GB vs. 291.2 GB 86% utilization of overall memory space



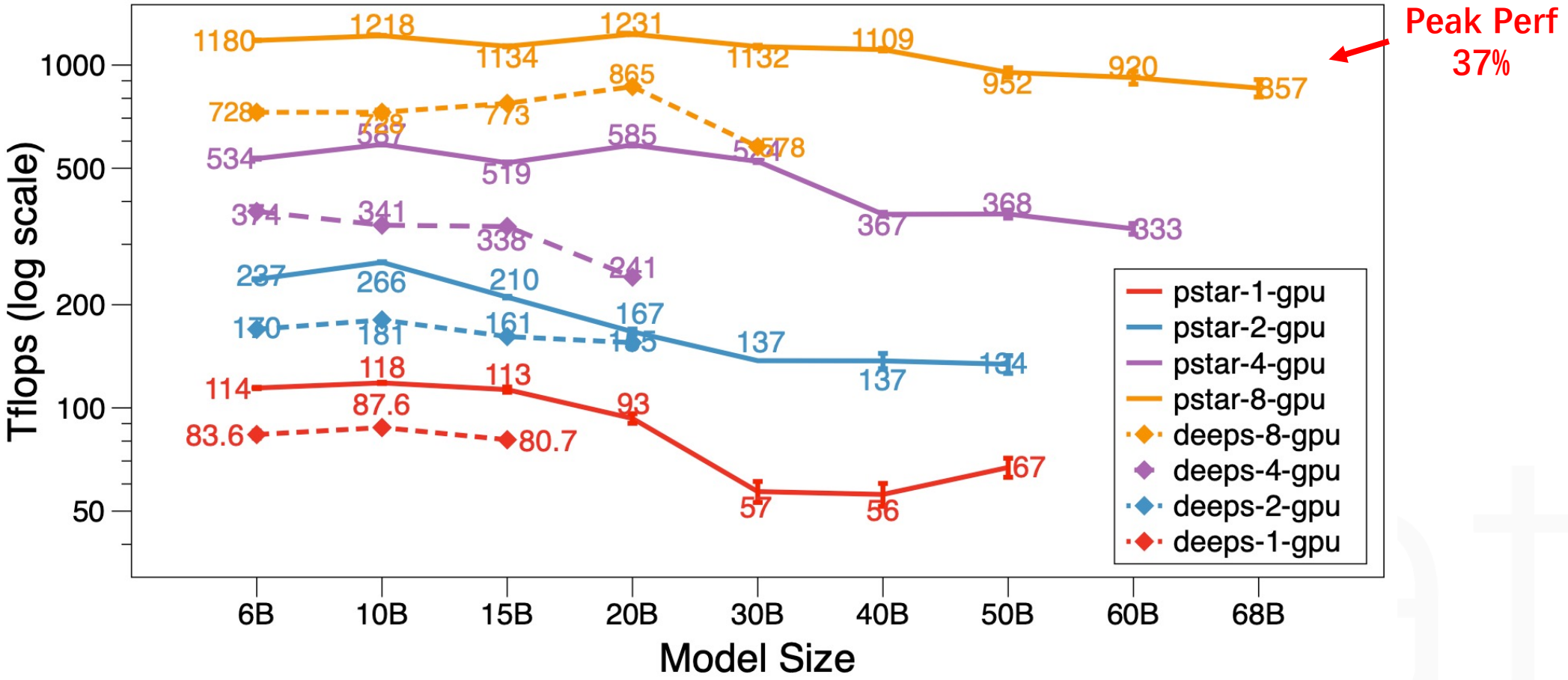
Results: Throughput on YARD

- PatrickStar is more efficient on DeepSpeed not supports cases (1.08x-1.47x, on average 1.23x)



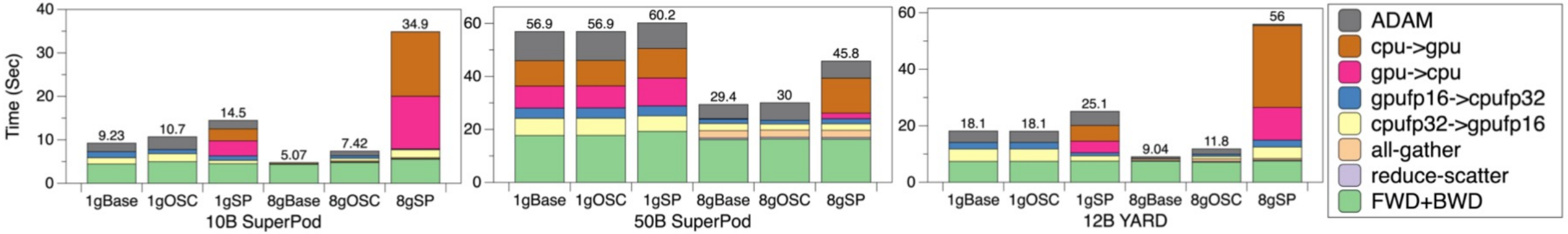
Results: Throughput on SuperPod

- PatrickStar speedup to DeepSpeed more significantly with more CPU memory (1.07x-2.43x, on average 1.53x)

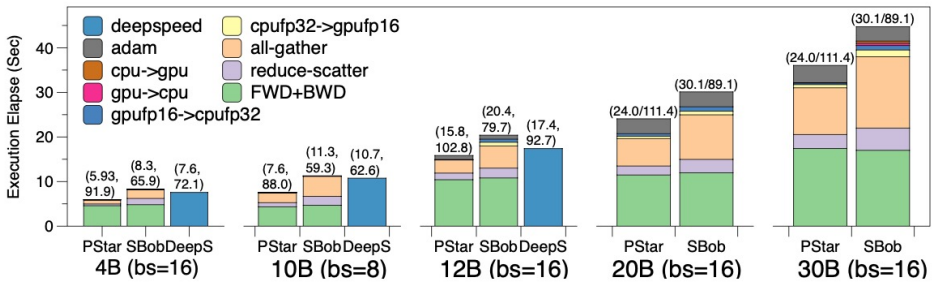


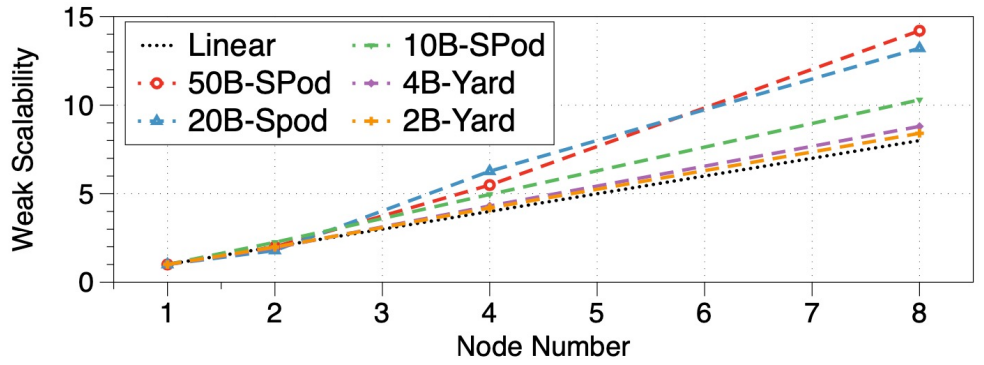
Ablation Analysis

- 👍Device-aware placement：XgBase vs. XgOSC(OS always on CPU)
- 👍🏿Runtime Memory Tracer：XgBase vs. XgSP (static partition)



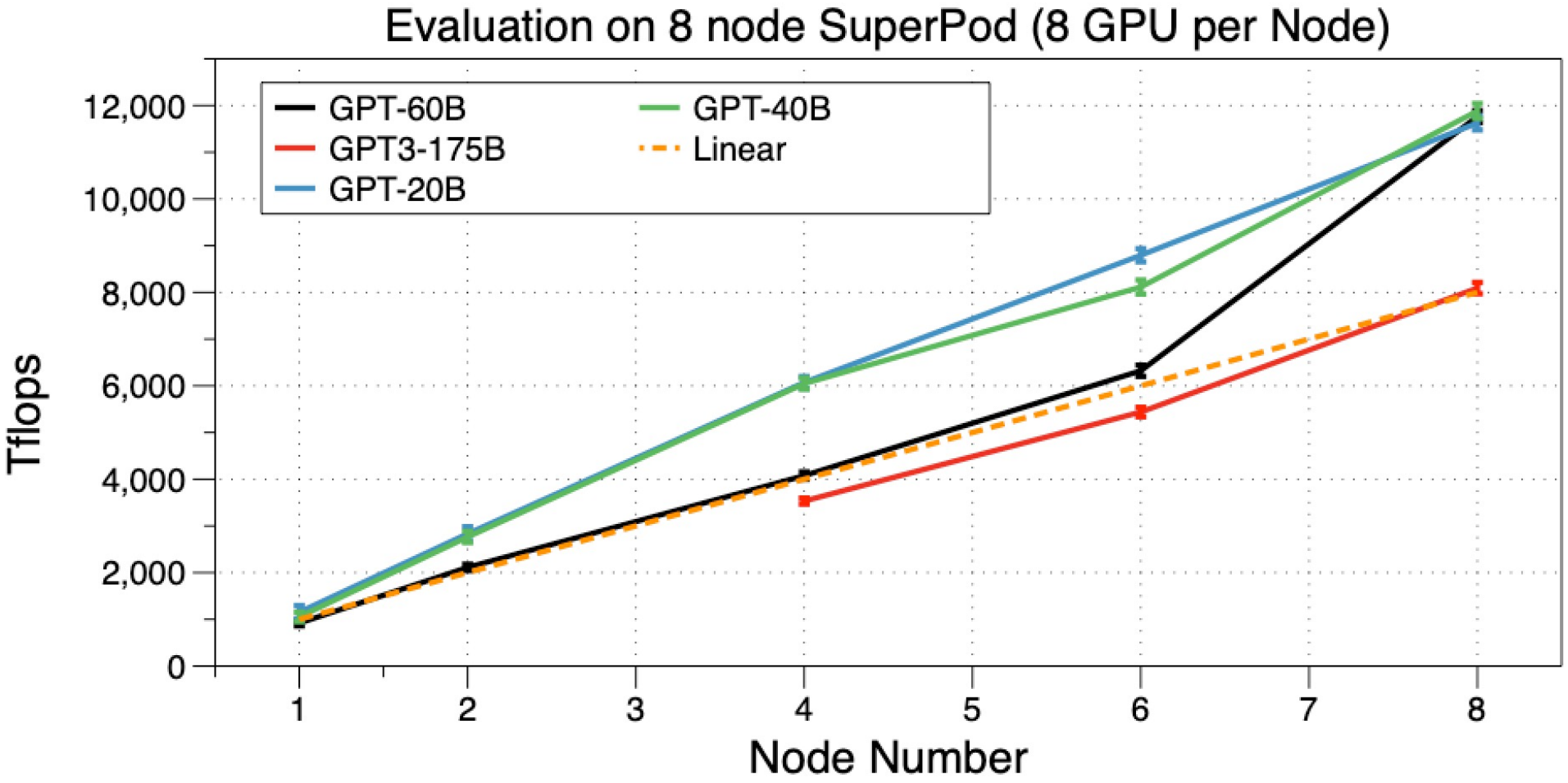
- 👍🏾CMM: Sbob system only DMM





Results: Scale to 8 nodes (64 GPU)

- 👏32xA100(4 node) is able to run GPT3 training
- 💪**Superlinear** scalability, 8 PetaFlops for GPT3@8node



Test Report from NVIDIA

- NVIDIA proves that PatrickStar uses fewer nodes to train GPT3 and the efficiency is comparable to that of Megatron-LM, and the efficiency reaches 40% of peak performance

Repro codes: Megatron-LM/examples/pretrain_gpt3_175B.sh

GPT3-175B, theoretical lower bounder of computing : 96BSlh^2 * (1 + S/6h + V/16lh) (Refer to paper: https://arxiv.org/pdf/2104.04473.pdf)

NVIDIA HQ number: https://github.com/nvidia/megatron-lm#gpt-3-example

Node Scale	Parallelism(TP*PP*DP)	B	l	h	S	V	Total required computing (FLOPs)	Peak computing(FP16, TFlops)	Ideal time(ms)	Real time (ms)	Efficiency
16	8*16*1	192	96	12288	2048	51200	5.63871E+17	39936	14119.3746	32550.21712	0.433772056
32	8*16*2	384	96	12288	2048	51200	1.12774E+18	79872	14119.3746	32834.27679	0.430019357
64	8*16*4	768	96	12288	2048	51200	2.25549E+18	159744	14119.3746	32672.6705	0.432146329
80	8*16*5	960	96	12288	2048	51200	2.81936E+18	199680	14119.3746	32850.00978	0.429813406
8	patricstar-数据并行	512	96	12288	1024	30522	7.40896E+17	19968	37104.16956	90444.179	0.410243865
128	8*16*8	1536	96	12288	2048	51200	4.51097E+18	319488	14119.3746	32000	0.441230456

Quick summary:
1. GPT-3 is basically linear-scaling through 16 -> 32 -> 64 -> 80 nodes.
2. For GPU utilization metrics, test number is bascially aligned with NVIDIA HQ 128 nodes number.

Related Open-Sourced Project

- https://github.com/Tencent/PatrickStar
- Preprint: *PatrickStar: Parallel Training of Pre-trained Models via a Chunk-based Memory Management*
- https://github.com/hpcaitech/ColossalAI
  - Already integrated the DMM module

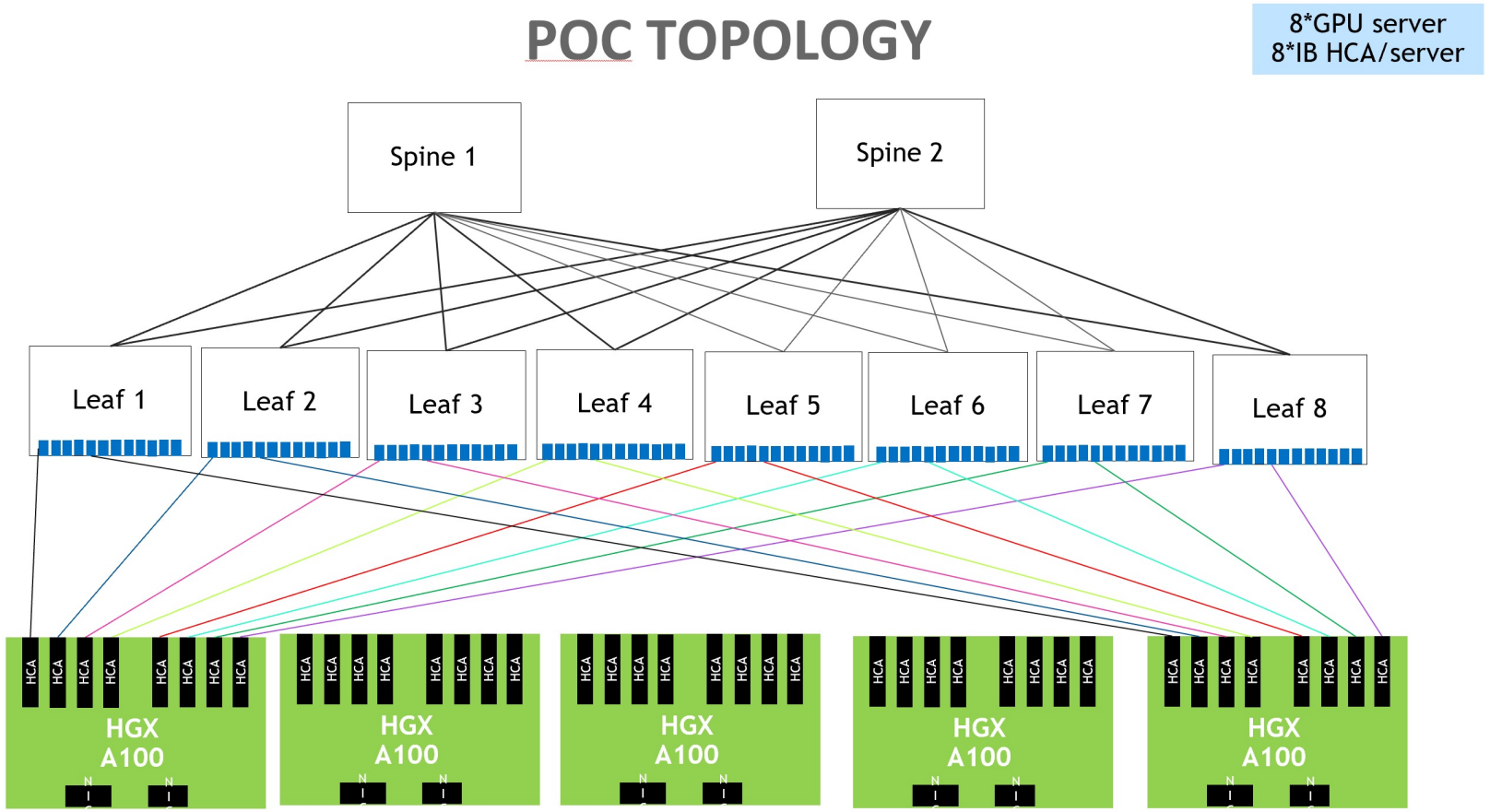
Start Heterogeneous Training in Lines

```
zero = dict(
    model_config=dict(
        tensor_placement_policy='auto',
        shard_strategy=TensorShardStrategy(),
        reuse_fp16_shard=True
    ),
    optimizer_config=dict(initial_scale=2**5, gpu_margin_mem_ratio=0.2)
)
```

(back to top)

Q&A

SuperPod Network Topology



SuperPod Spec

Basic information	
Server Model	H3C G5500 Gen5
CPU	AMD EPYC 7K62 48-Core Processor
GPU	Delta A100 40GB
Mem	1T DDR4 2933 MT/s
Disk	4 x 3.2T NVMe SSD
NIC	8 * ConnectX6 200Gb/s HCA
OS	Tlinux 4.14.105-1
GPU driver	470.57.02
MOFED	5.4-1.0.3.0
Docker	2020.10.8
NGC	PyTorch 21.08(CUDA 11.4.1, cuDNN 8.2.2.26, NCCL 2.10.3)
环境要求	NCCL版本要求：> 2.9x 低版本nccl跑步起来