ZOMI 大模型：分布式训练

流水并行原理 Pipeline Parallel

大模型业务全流程

数据 & 模型算法

2. 数据处理

- 开源数据
- 数据预处理
- 向量数据库

3. 模型算法

- LLM模型架构
- 多模态一切皆Tokens

模型训练 & 微调

4. 模型训练

- 混合精度
- 梯度检查
- 梯度累积
- …
- 5. 分布式并行
- 训练集群稳定性

6. 模型微调

- 全参微调
- 低参微调
- 指令微调

模型验证 & 推理部署

7. 模型验证

- 下游任务
- 测评标准
- bench mark

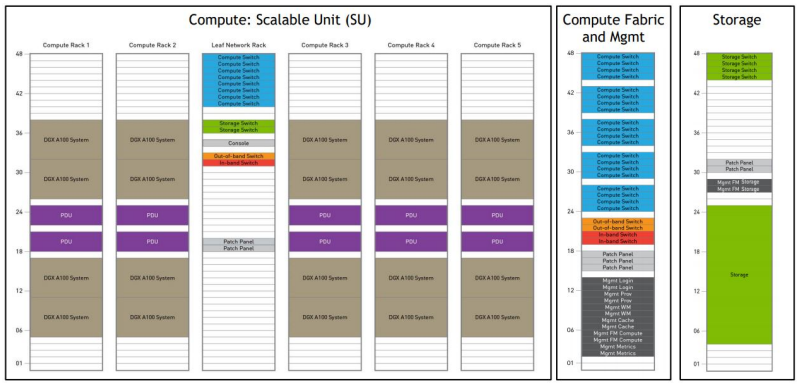
8. 推理与智能体

- 量化压缩
- 推理加速
- 9. Agent 智能体

集群算力准备

1. AI 集群建设

- 计算、存储、网络
- AI 集群机房建设
- AI 集群上线与运维

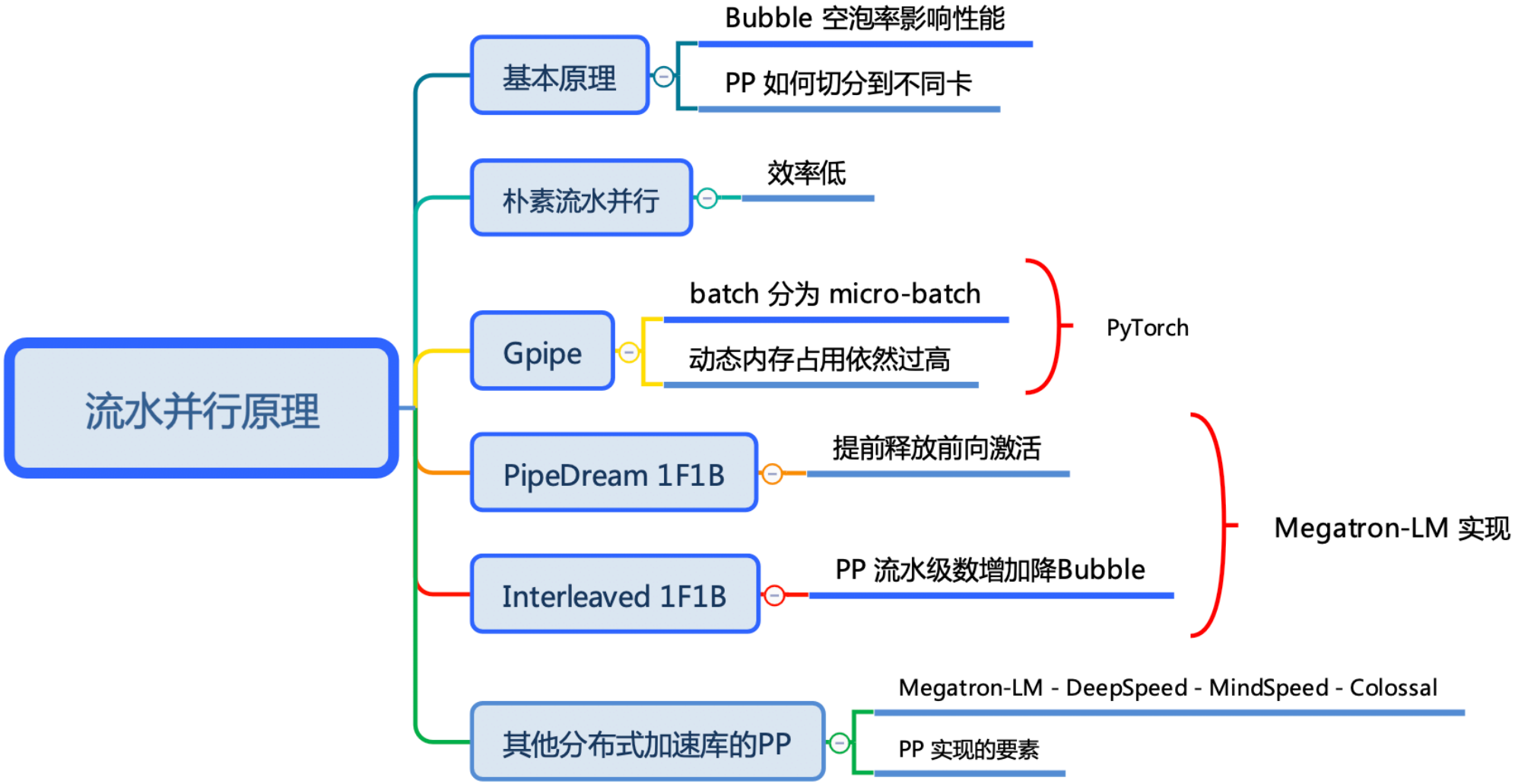


大模型系列 – 分布式训练加速

- **具体内容**

1. **分布式加速库：**
   - 业界常用分布式加速库 & 作用
2. **DeepSpeed 特性：**
   - 基本概念 - 整体框架 – Zero-1/2/3 – ZeRO-Offload – ZeRO-Infinity
3. **Megatron 特性：**
   1. 总体介绍 – 整体流程 – 并行配置 – DP – TP – PP

思维导图

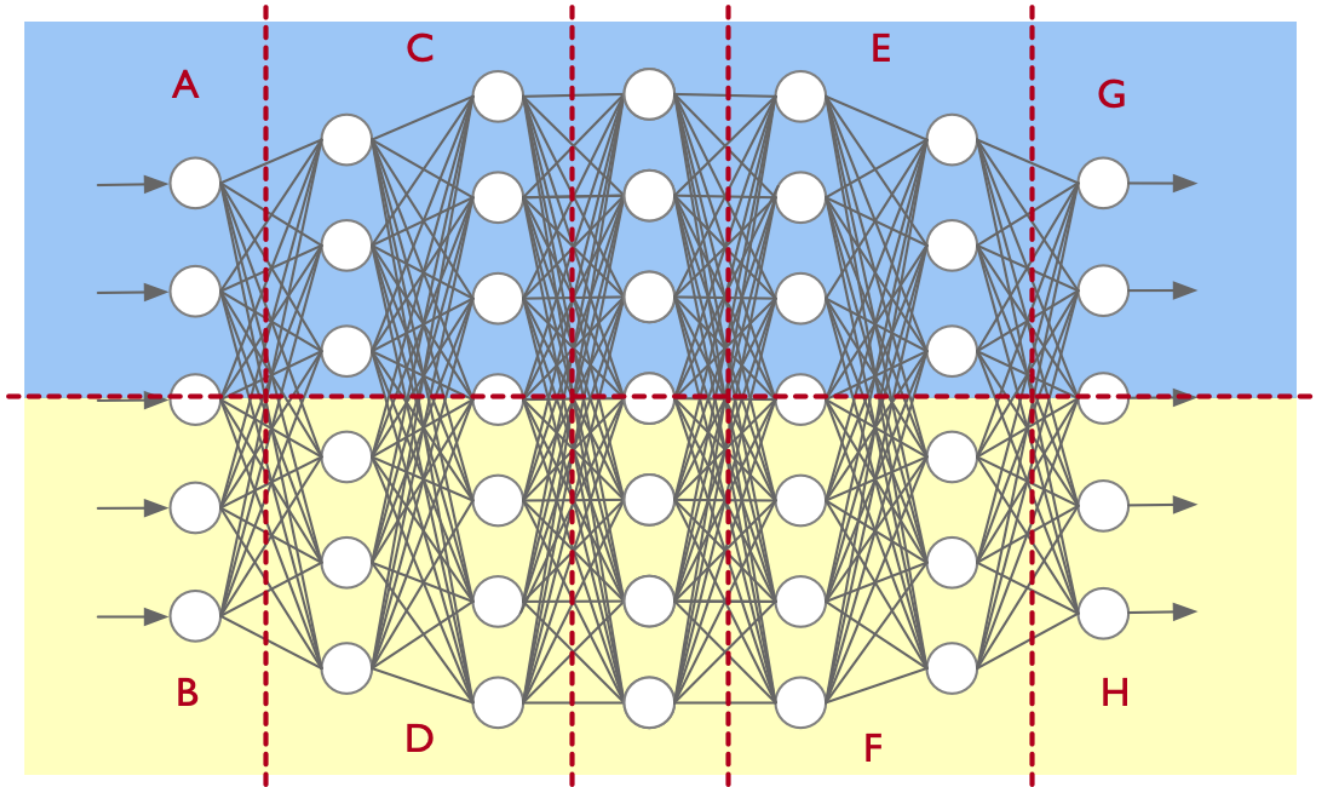

流水并行原理
基本原理
Bubble 空泡率影响性能
PP 如何切分到不同卡
朴素流水并行
效率低
Gpipe
batch 分为 micro-batch
动态内存占用依然过高
PyTorch
PipeDream 1F1B
提前释放前向激活
Interleaved 1F1B
PP 流水级数增加降Bubble
Megatron-LM 实现
其他分布式加速库的PP
Megatron-LM - DeepSpeed - MindSpeed - Colossal
PP 实现的要素

Megatron-LM 01

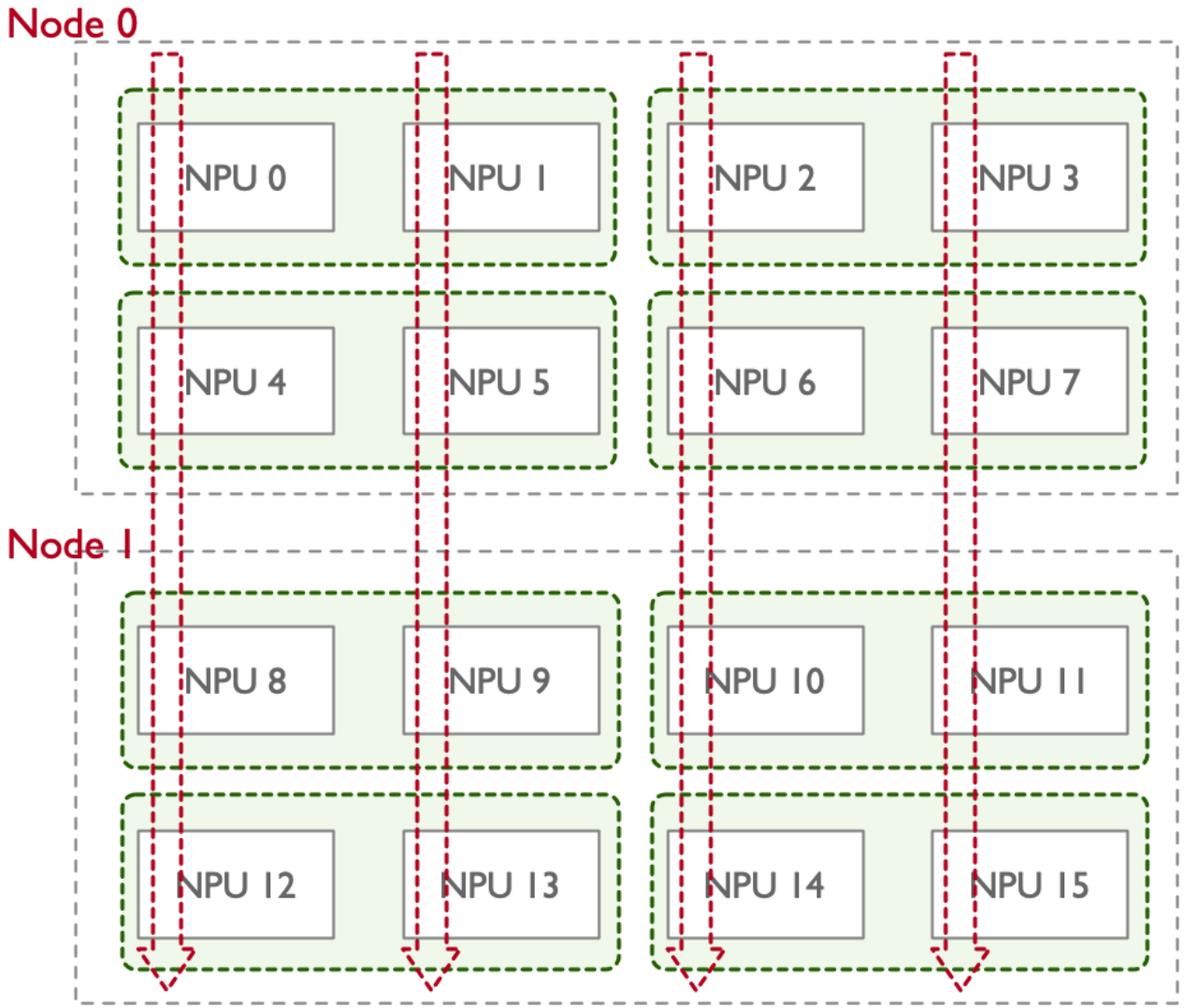
并行配置回顾

思考

1. 如何把大模型按照模型层数或者切分好的模块，分块放到对应的 NPU 上？

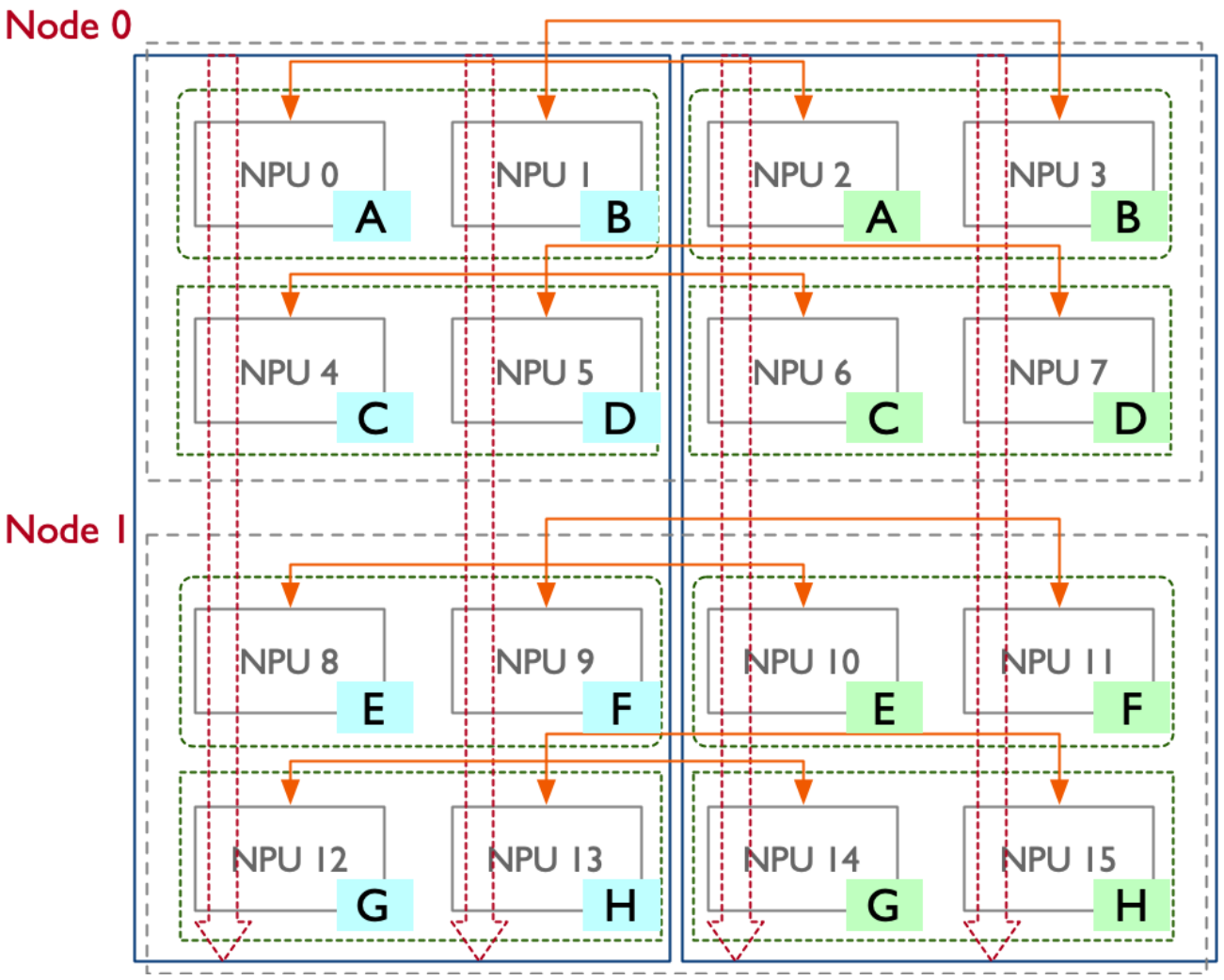


PP 进程组 rank 分配



- world size = 16，PP = 4
- group16 / 4 = 4
  - 流水线 group 每隔 n // p 个取一个，stage i 的 rank 范围是：[(i-1) * n//p, (i) * n//p]
  - 分为 4 个进程组 group，每个进程组有 4 个 rank，每个 rank 执行 Pipeline 一个 Stage
  - Pipeline Stage 为 4， 串行执行。即 group1 [g0, g4, g8, g12] 4 个 NPU 串行。

建立模型与 NPU 卡间关系

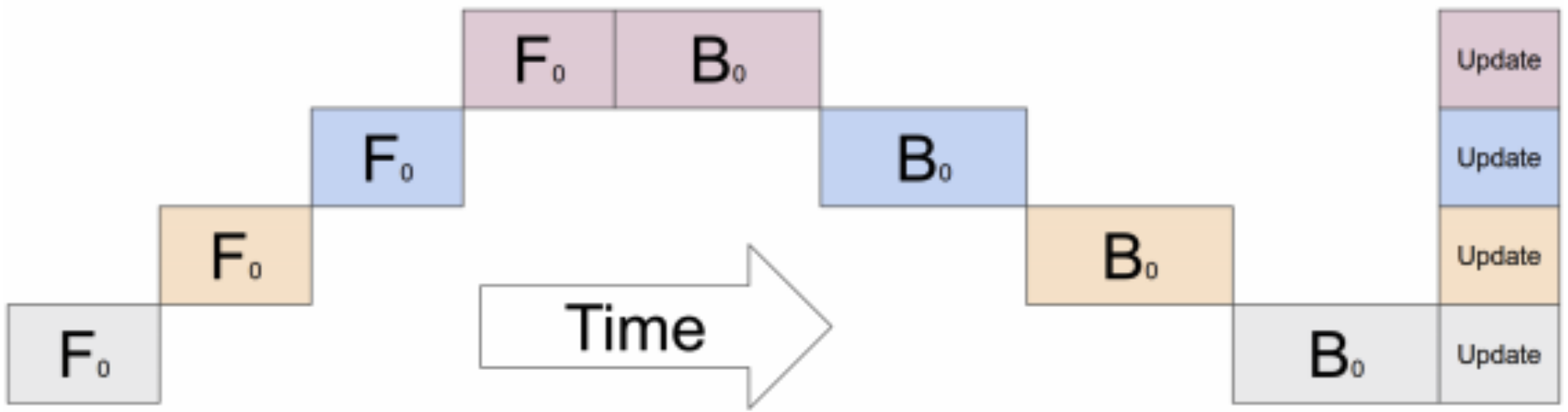


1. 同名子模块具有同样的网络模型结构与参数，可进行数据并行：
   - e.g.，两个 A 可以数据并行
1. 纵向层间可进行流水线串行：
   - e.g., A → C → E → G
1. 横向是流水线的一个stage：
   - e.g., 从 0 开始，相邻为 A & B 为 TP

02 朴素流水并行

朴素流水线并行

- K 为 NPU 数量，朴素流水线 Bubble 时间为 $O(K-1/K)$
  1. NPU 数量越多时，空置比例接近 1，浪费计算资源；
  2. 通信和计算缺乏重叠 overlap，网络传输中间输出 (FWD) 和梯度 (BWD) 时，没有 NPU 执行计算；
  3. 消耗大量内存，先执行 Forward 的 NPU 将保留整个 batch 的缓存激活，直到优化器更新 Update。

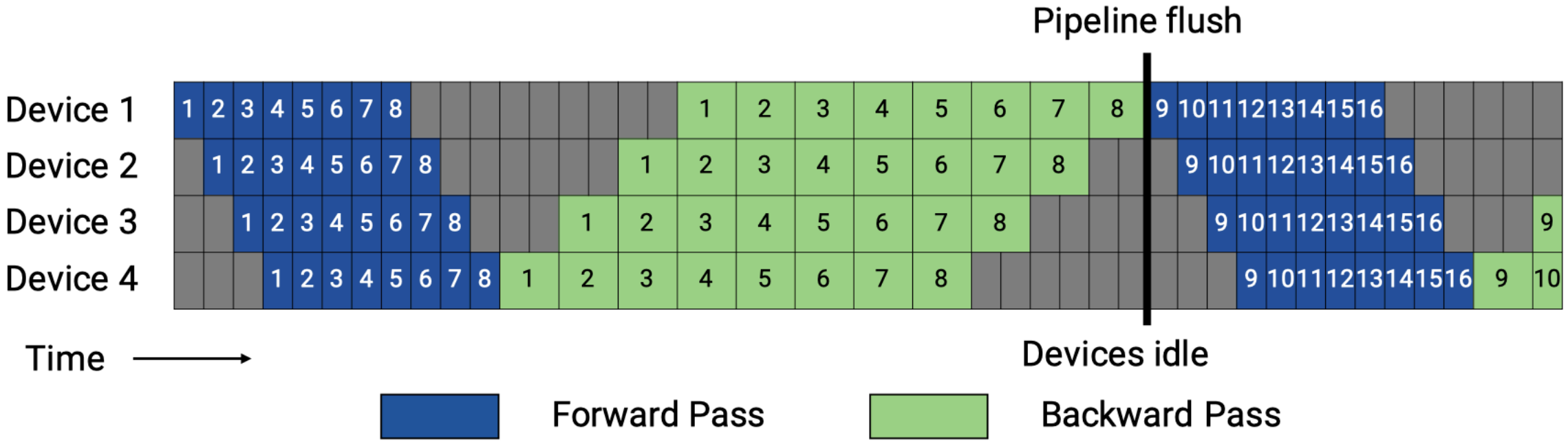


03 Megatron-LM

流水并行 Gpipe

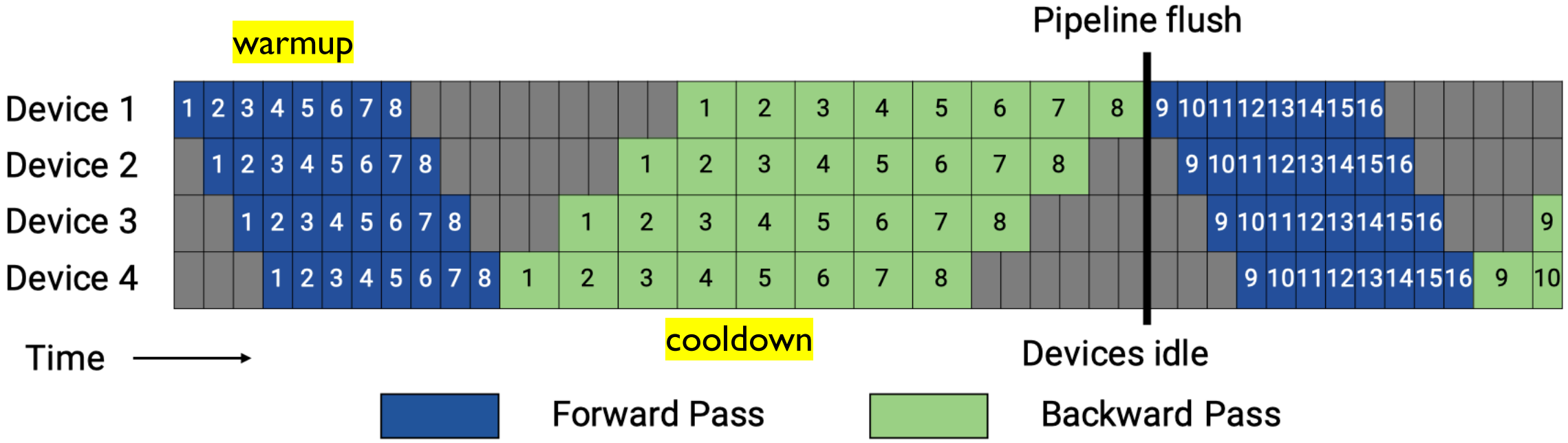
Gpipe, also named Fill-Drain

- **总体**：先前向流水，再反向流水。将 Transformer 按层切分 NPU1-4，一个 batch size 切分 8 个 micro-batch。
- **前向**：每 micro-batch 从 NPU 1 流向 NPU4；….
- **反向**：NPU4 计算 batch1 梯度并更新对应层参数，梯度传向 NPU3；….



Gpipe also named Fill-Drain

- For simplicity, we assume that the backward pass takes twice as long as the forward pass. The efficiency of the pip eline schedule does not depend on this factor. The optimizer is stepped and weight parameters updated at the pip eline flush to ensure strict optimizer semantics, leading to idle NPUs and a pipeline bubble.



Gpipe Bubble 计算

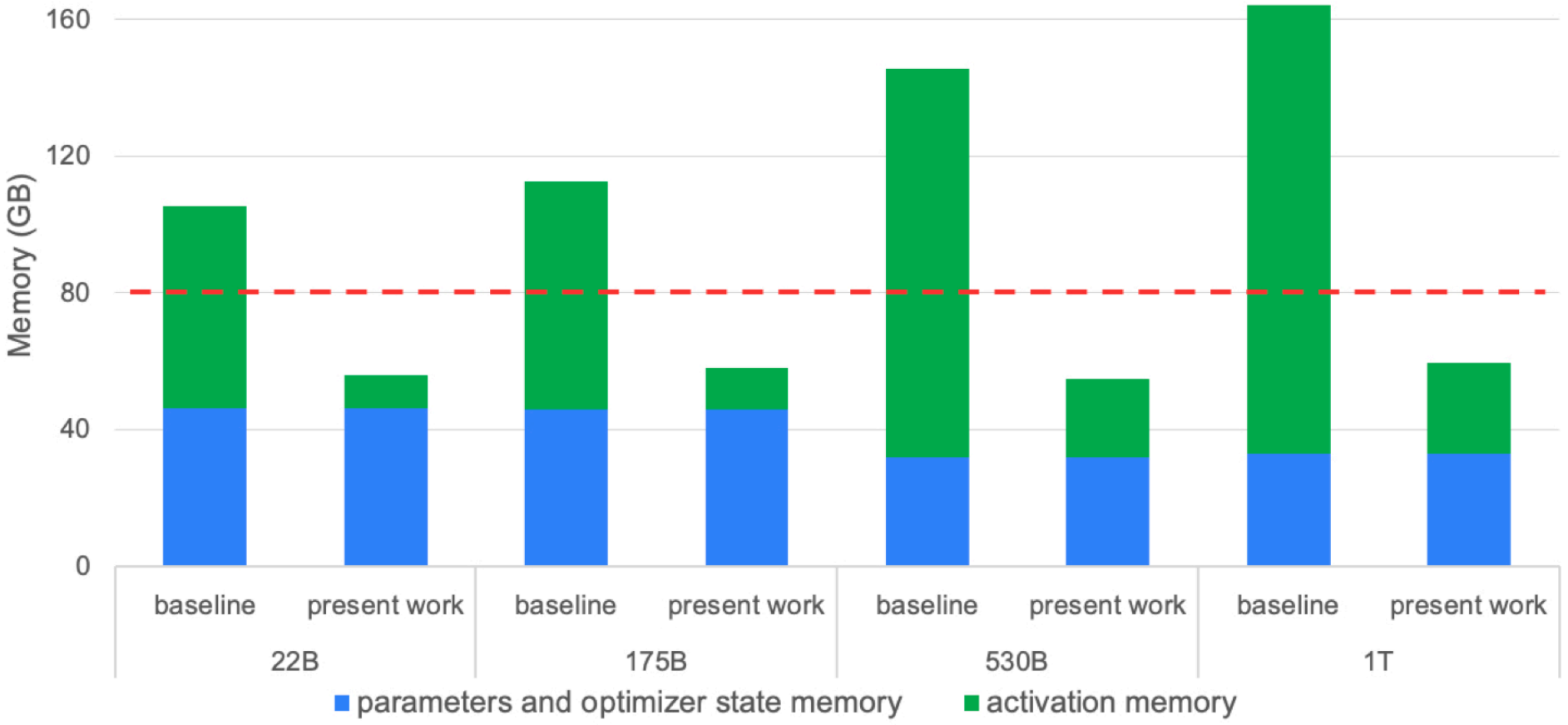
- **空泡率：**t_{pb} 为 bubble 耗时，m 为 micro-batch，p 为 pipeline stages，t_{id} 为理想的迭代时间，t_f 表示前向时间，t_b 表示反向时间，pipeline bubble 占据共有 $(p-1)$ 个前向和反向，$t_{pb} = (p-1)(t_f + t_b)$，理想时间 $t_{id} = m(t_f + t_b)$：

$$bubble\ time\ fraction = \frac{t_{pb}}{t_{id}} = \frac{p-1}{m}$$

- **目标：**降低空泡比率，需要 $m \gg p$，but 动态内存峰值占用高。每个 micro-batch 反向算梯度计算，都需要前向激活值，m 个 micro-batch 前向结束时（CoolDown），达到内存占用的峰值。

降低 Gpipe 内存

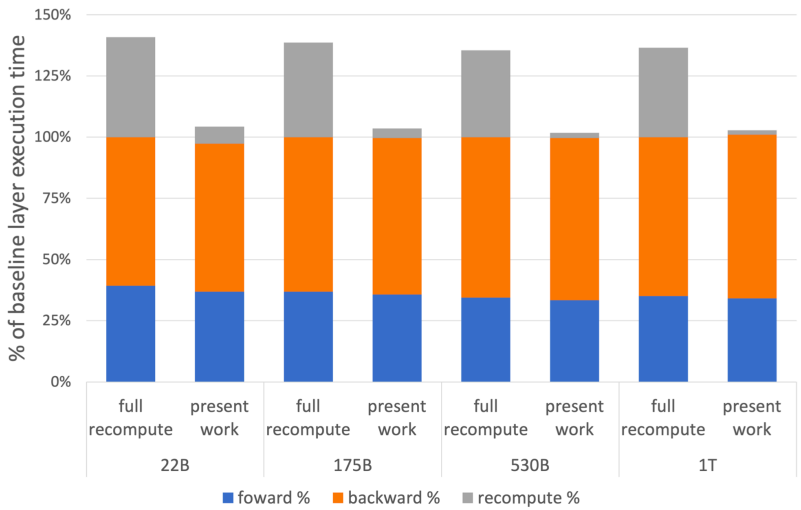
- 解决思路：Re-Materialization / Activation-Checkpointing，激活重计算



降低 Gpipe 内存

Configuration	Activations Memory Per Transformer Layer
no parallelism	$sbh\left(34+5\frac{as}{h}\right)$
tensor parallel (baseline)	$sbh\left(10+\frac{24}{t}+5\frac{as}{ht}\right)$
tensor + sequence parallel	$sbh\left(\frac{34}{t}+5\frac{as}{ht}\right)$
tensor parallel + selective activation recomputation	$sbh\left(10+\frac{24}{t}\right)$
tensor parallel + sequence parallel + selective activation recomputation	$sbh(\frac{34}{t})$
full activation recomputation	$sbh(2)$

Table 2: Activations memory, in bytes, per transformer layer for different techniques.



Model Size	Iteration Time (seconds)		Throughput Increase	Model FLOPs Utilization	Hardware FLOPs Utilization
	Full Recompute	Present Work			
22B	1.42	1.10	29.0%	41.5%	43.7%
175B	18.13	13.75	31.8%	51.4%	52.8%
530B	49.05	37.83	29.7%	56.0%	57.0%
1T	94.42	71.49	32.1%	56.3%	57.0%

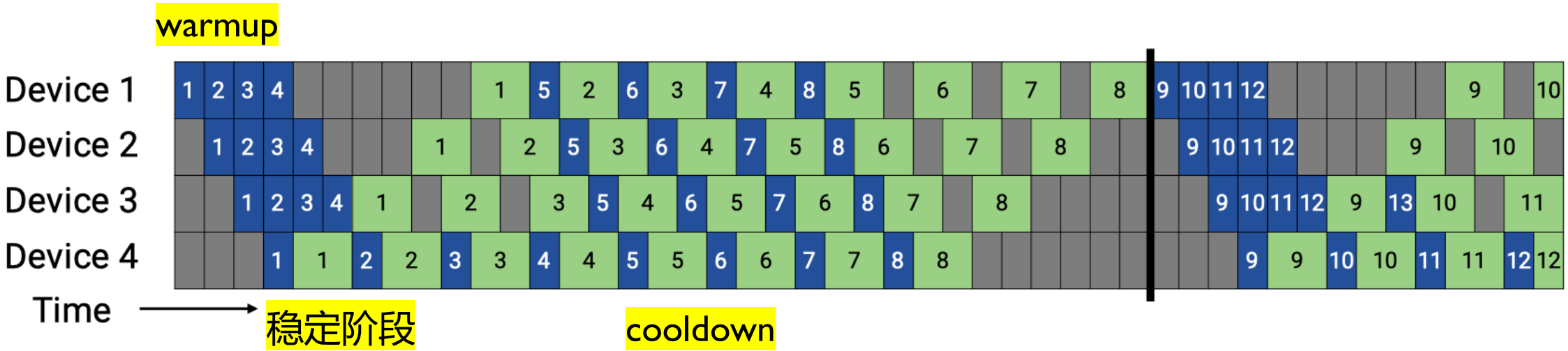
Table 5: End-to-end iteration time. Our approach results in throughput increase of around 30%.

04 Megatron-LM

流水并行 1F1B

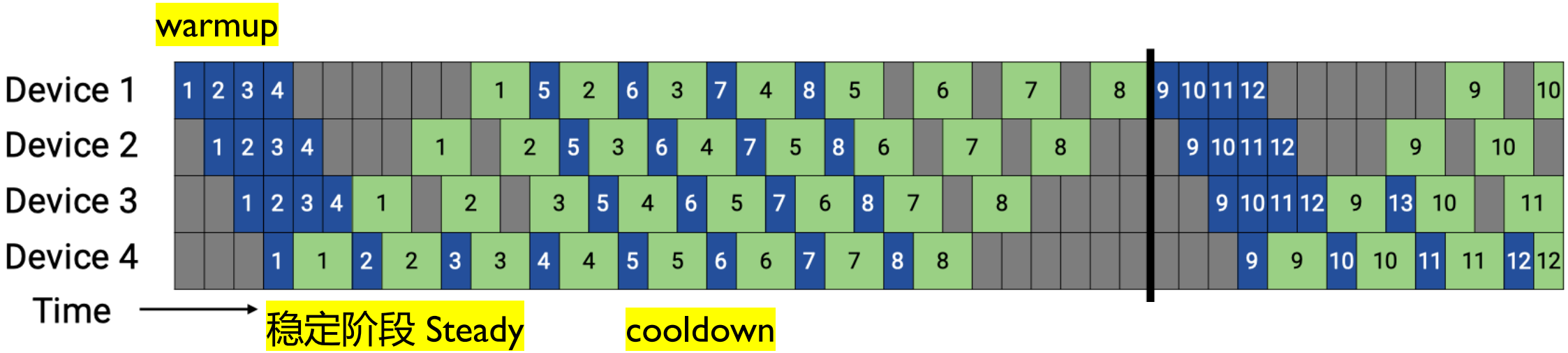
One Forward and One Backward(1F1B) also named PipeDream

- Megatron 实现时候将 warmup 和 cooldown 单独拆分出来，普通开启 VP 即 1F1B；
- 每个 NPU 上最少只需要保存 1 份 micro-batch 的激活值，最多需要保存 p 份激活值。
  1. NPU 4 峰值动态内存（激活值）只有 1 份；
  2. NPU 1 峰值动态内存（激活值）最多只有 4 分；



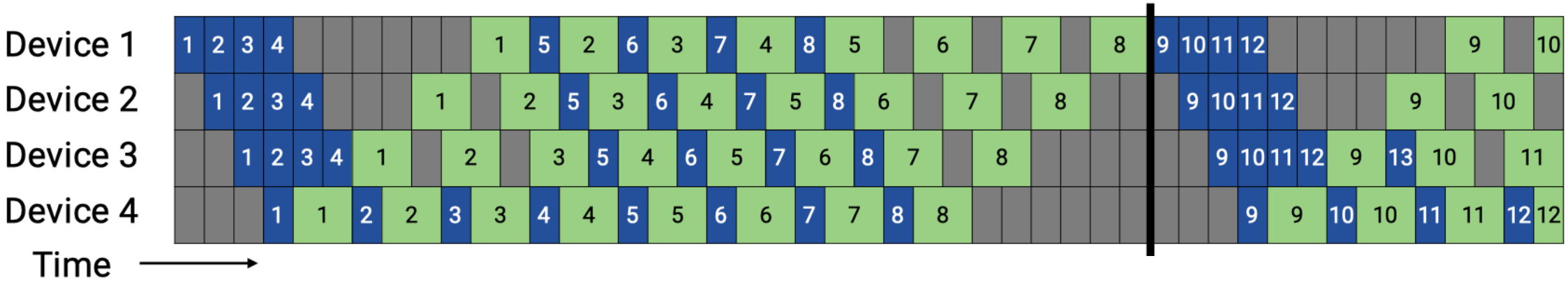
One Forward and One Backward(1F1B) also named PipeDream

- 激活值数量上限从 micro-batch 数量 m 变成 pipeline stage 阶段 p ，那么：
  1. 可以依据 $Bubble\ fraction$ 公式 $m \gg p$ 原则，增加 micro-batch 数量，从而降低 Bubble 占比；
  2. 在不增加 micro-batch 情况下，增加 seq-length，让大模型能够放下更多 Tokens。



One Forward and One Backward(1F1B) also named PipeDream

- **内存占用：**
  - G-pipe 到 PipeDream 进化，通过及时安排反向过程，将前向激活值释放掉，避免积累太多激活值占用内存。

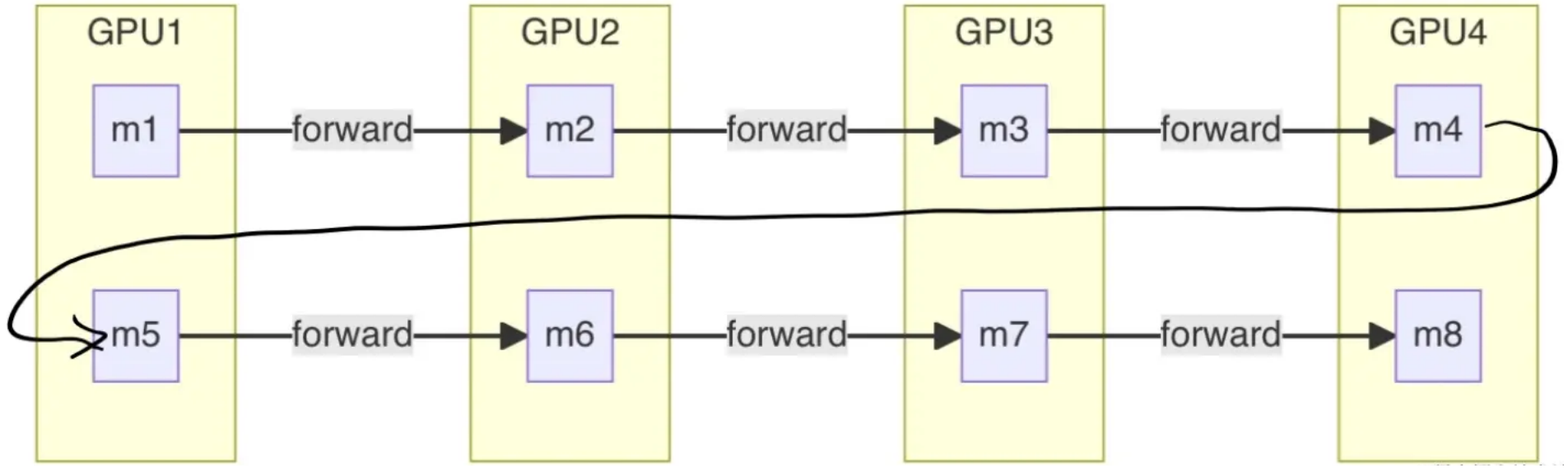


05 Megatron-LM

流水并行 Interleaved

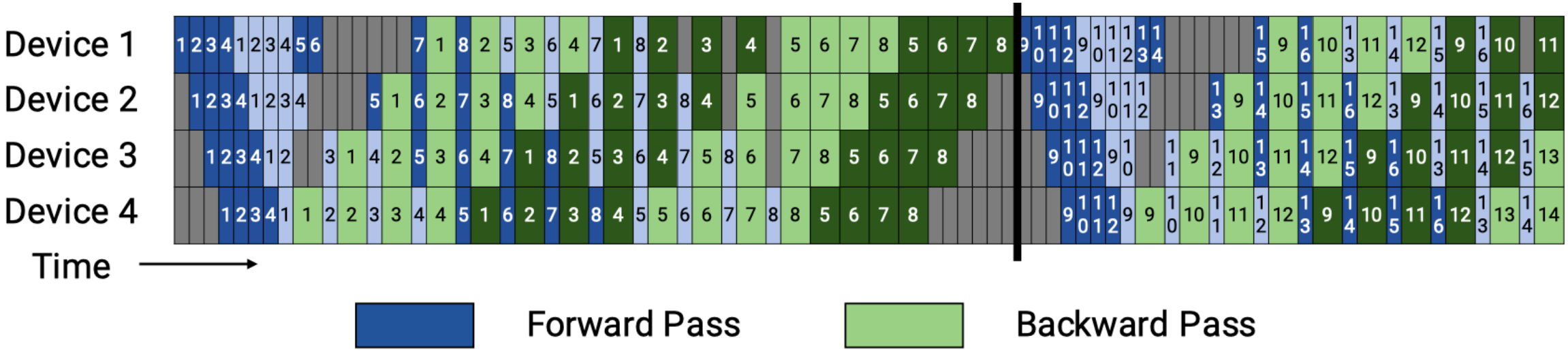
virtual pipeline 原理

- IF1B 可以使得流水并行增加 micro-batch 数量，从而降低 Bubble 占比。**将流水线划分更细**：
  1. 增加 micro-batch 数量，而不是增加每个micro-batch 的样本数，增加每次迭代耗时；
  2. NPU 数量固定，每个 NPU 上可以执行多个 Transformer Layer，So like these：



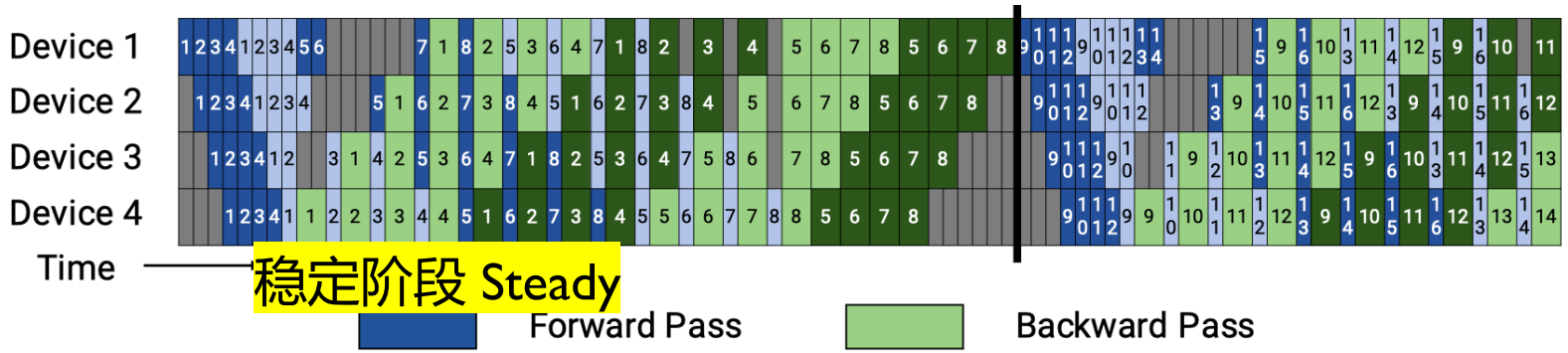
Virtual pipeline also named Interleaved 1F1B

- virtual pipeline stage=2，即每个 NPU 同时处于 4 条流水线中，2 条前向流水线，2 条后向流水线。有效降低 bubble fraction，从而提升了设备利用率：
  1. 深蓝色 NPU1 执行 LLM Layer1 前向，浅蓝色 NPU1 执行 LLM Layer5 前向；
  2. 浅绿色 NPU1 执行 LLM layer1 反向，深绿色 NPU1 执行 LLM layer5 反向；



Virtual Pipeline also named Interleaved 1F1B

- Why named Interleaved 1F1B or 1F1B-interleaving ?
  - 在 steady 的时候也是 1F1B 的形式，叫做 1F1B-interleaving



Interleaved 1F1B Bubble 计算

- **空泡率：** t_{pb}^{int} 为 Interleaved bubble 耗时，m 为 micro-batch，p 为 pipeline stages，v 为 virtual pipeline stage，t_{id} 为理想的迭代时间，前向时间为 t_f/v，反向时间为 t_b/v，pipeline bubble 占据共有 $(p-1)$ 个前向和反向，$t_{pb}^{int} = (p-1)(t_f + t_b)/v$，理想时间 $t_{id} = m(t_f + t_b)$：

$$bubble\ time\ fraction^{nit} = \frac{t_{pb}^{int}}{t_{id}} = \frac{p-1}{mv}$$

- 空泡率除了跟 micro batch 成反比，与 v 也成反比；

问 题

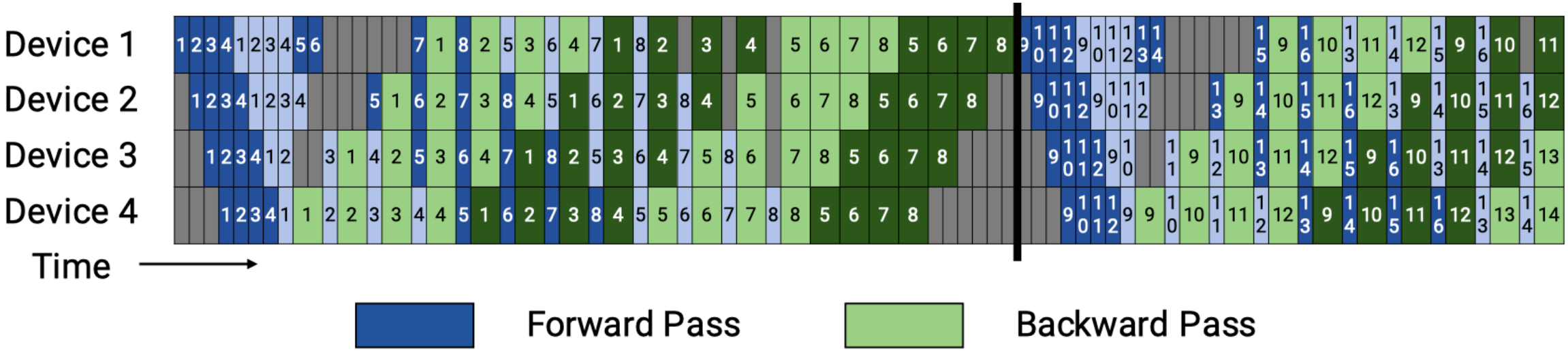
NPU 间点对点通信次数为 virtual pipeline stage 倍。

思考

1. NPU 间点对点通信次数为 virtual pipeline stage 倍，大模型通信占比过高，会极大降低 MFU/HFU 利用率的？！
   - 计算与通信的流水进行掩盖 overlap，因此带来的集群性能影响不是很明显~~

Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM

- **空泡率：**
  - 空泡率提升主要从 1F1B 到 1F1B-interleaving 进化。pipeline 并行基本规律是 pipeline 流水级数越多，ove rhead 就越小。



Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM

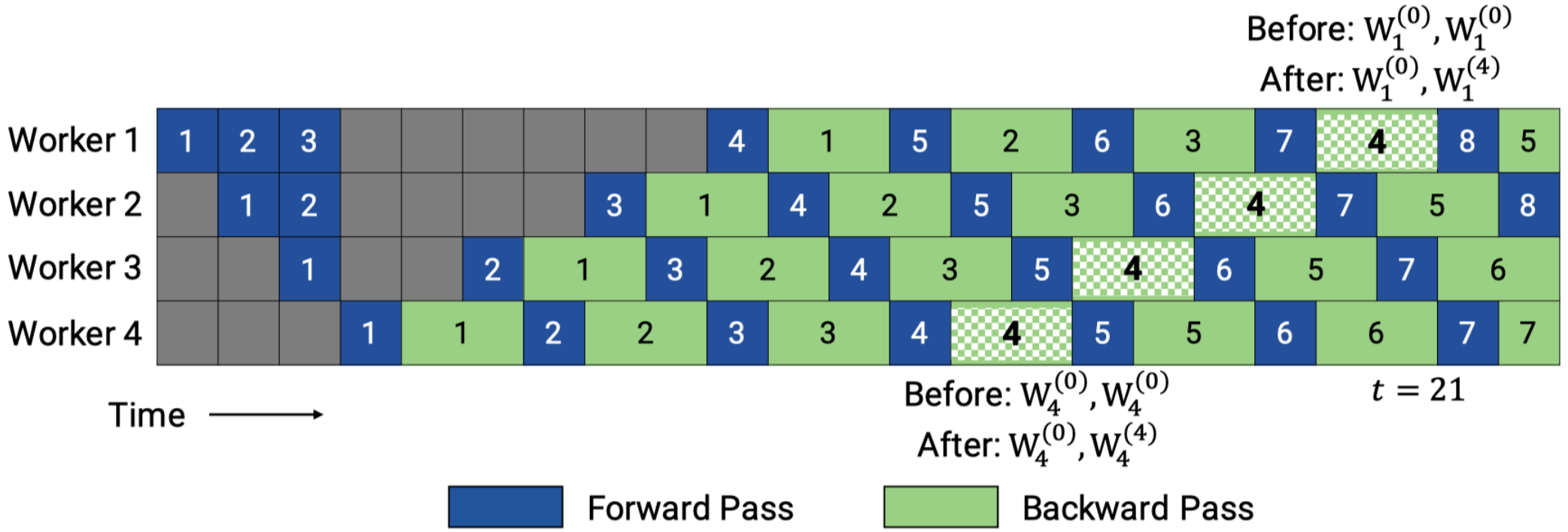
- 大模型在 Scaling law 下增加硬件的资源，不仅仅是增加 DP 域，通过增加 PP 域能够更加合理地分布模型结构在 AI 集群。提升集群的整体性能：吞吐（Throughput）和 模型利用率（MFU）

Number of parameters (billion)	Attention heads	Hidden size	Number of layers	Tensor model parallel size	Pipeline model-parallel size	Number of GPUs	Batch size	Achieved teraFlOP/s per GPU	Percentage of theoretical peak FLOP/s	Achieved aggregate petaFLOP/s
1.7	24	2304	24	1	1	32	512	137	44%	4.4
3.6	32	3072	30	2	1	64	512	138	44%	8.8
7.5	32	4096	36	4	1	128	512	142	46%	18.2
18.4	48	6144	40	8	1	256	1024	135	43%	34.6
39.1	64	8192	48	8	2	512	1536	138	44%	70.8
76.1	80	10240	60	8	4	1024	1792	140	45%	143.8
145.6	96	12288	80	8	8	1536	2304	148	47%	227.1
310.1	128	16384	96	8	16	1920	2160	155	50%	297.4
529.6	128	20480	105	8	35	2520	2520	163	52%	410.2
1008.0	160	25600	128	8	64	3072	3072	163	52%	502.0

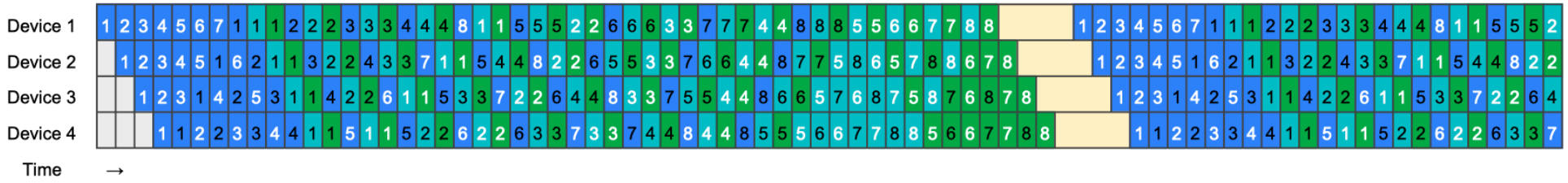
Table 1: Weak-scaling throughput for GPT models ranging from 1 billion to 1 trillion parameters.

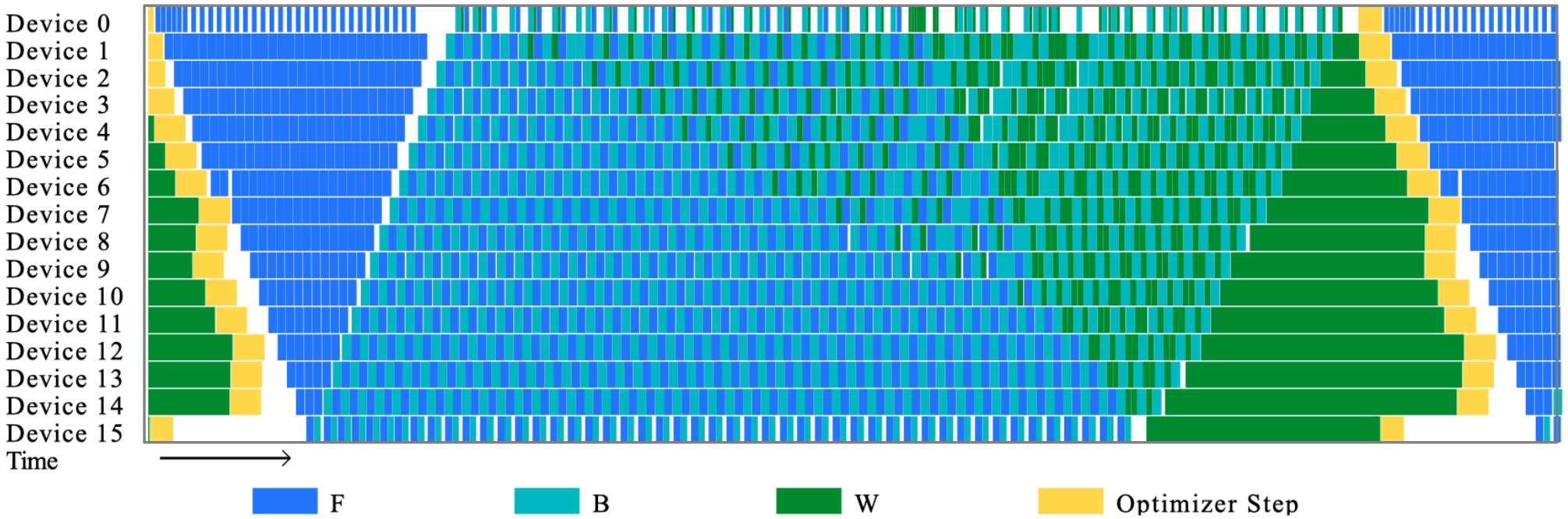
06 其他流水并行

PipeDream-2BW

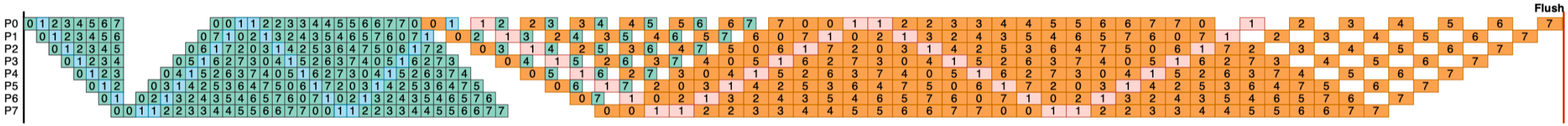


ZB-V schedule

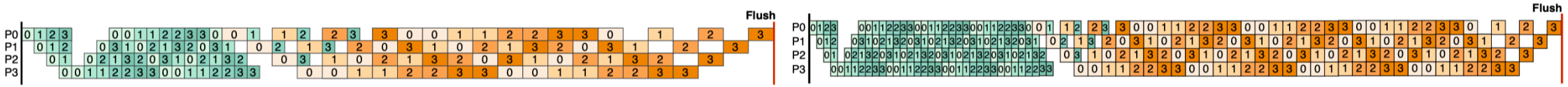




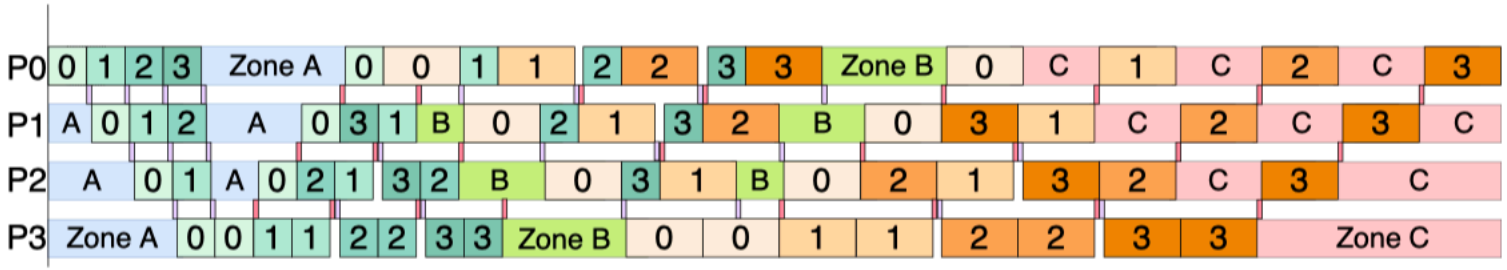
Hanayo wave-like pipeline



(a) wave=2, devices=8



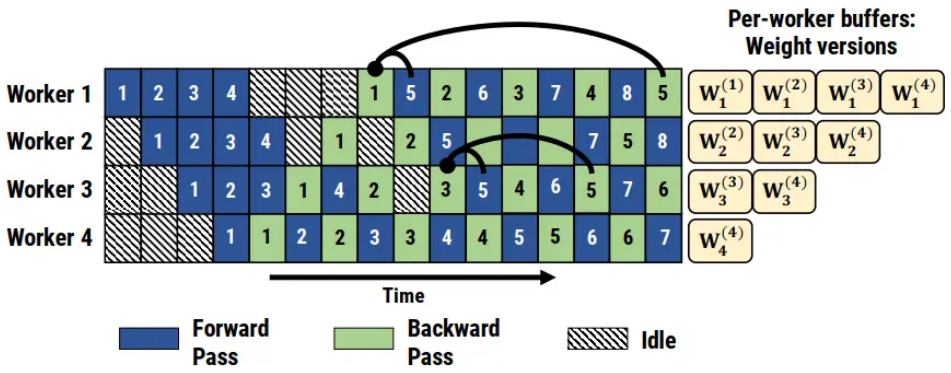
(b) wave=2 and wave=4, devices=4

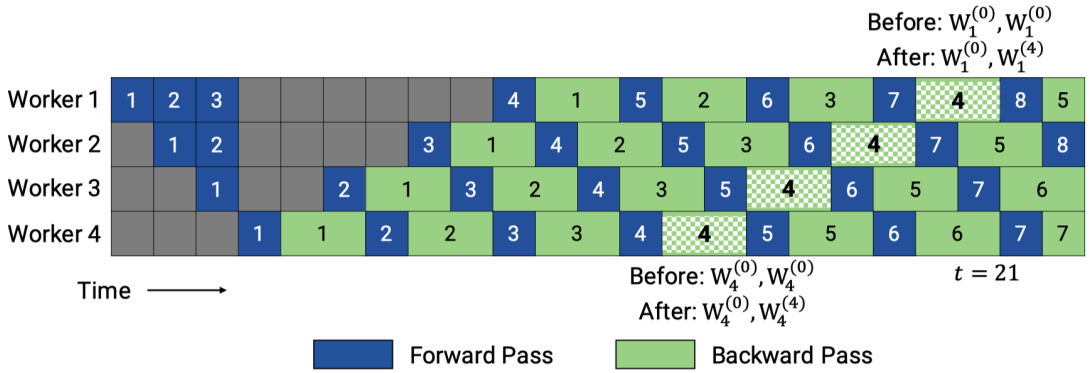


07 分布式加速库的PP

PP 分类

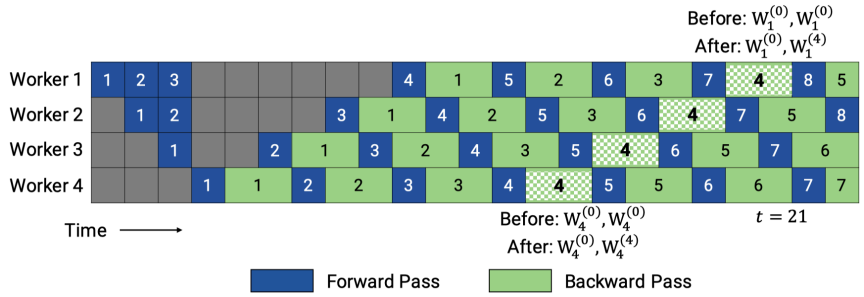
- 可细分为同步流水线并行和异步流水线并行：
  1. Sync-PP 同步流水：代表 GPipe，PipeDream-flush 等；
  2. Async-PP 异步流水：代表 PipeDream，PipeDream-2BW 等

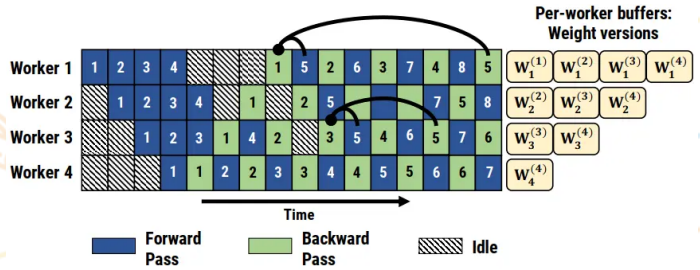




分布式训练框架流水线并行方案

- **同异步区别在哪？**
  - 同步：需要引入更多流水线 Bubble，会降低集群训练吞吐；
  - 异步：彻底消除训练 Timeline Bubble，但引入不同权重版本来解决权重更新过期问题。





PP 实现要素

- 实现一个多进程的流水并行。有 3 要素：
  1. 等待上一个节点的执行结果 receive；
  2. 执行当前节点的操作 exec；
  3. 将结果传给下一个节点 send；

分布式训练框架 PP 方案

- PyTorch：默认提供 Gpipe，使用 F-then-B 调度策略；
- DeepSpeed：PipeDream-Flush，非交错式 1F1B 调度策略；
- Megatron-LM：基于 PipeDream-Flush 进行改进，提供一种交错式 1F1B（VPP）调度策略；
- Colossal-AI：基于 Megatron-LM 的 PipeDream-Flush，提供了非交错和交错调度策略；
- Ascend MindSpeed：基于 Megatron-LM 的 PipeDream-Flush，提供了非交错和交错调度策略；

美国五星上将麦克阿瑟曾经说过：

PipeDream-Flush + 1F1B 才是经典呀

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

Course chenzomi12.github.io

GitHub github.com/chenzomi12/DeepLearningSystem