大规模异构计算

——深度学习背后的驱动力量

2019.12.05 颜深根 执行研究总监 商汤科技

CONTENTS

目录

1 AI复兴：深度学习

2 计算优化：GPU

3 计算以外：内存&访存

4 携手共进：通信优化

5 无所不能：云上AI

6 轻松一刻：应用案例

各项人工智能技术取得突破：深度学习

人脸认证

语音识别

图像分类

围棋博弈

深度学习

深度学习让机器从大量资料中归纳总结经验

这一套技术体系让机器训练学会特定技能

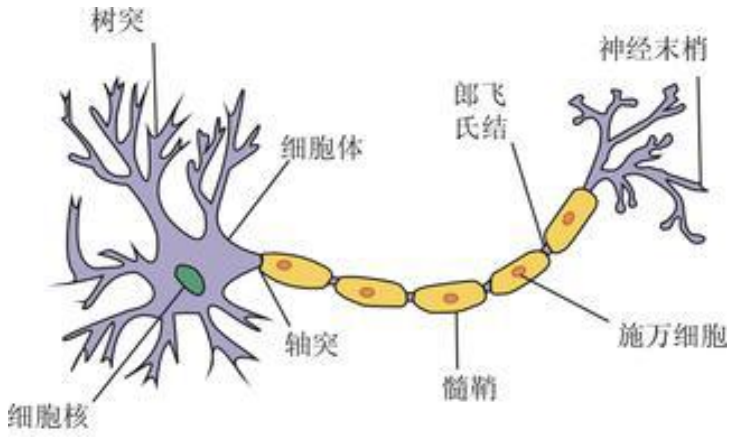
自动驾驶

智能投顾、量化交易

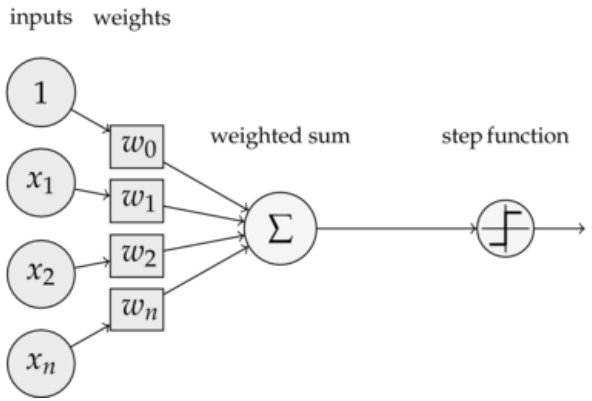
语言理解

深度学习的基本思想

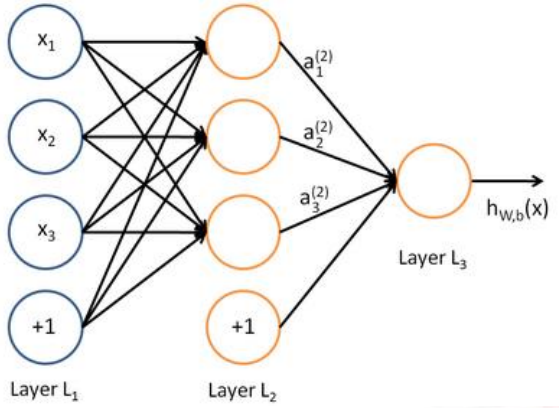
- 人工神经网络
  - 源于对人脑神经元运行机理的研究，始于20世纪40年代
  - 将神经的突触连接结构用抽象化的模型表达
  - 将多个单一神经元结构相互连接得到复杂的网络结构



生物神经元模型



感知器(Perceptron)



多层感知器

BP(Back Propagation)神经网络

●Back Propagation原理

- Rumelhart，McClelland于1985年提出了BP网络的误差反向后传BP(Back Propagation)学习算法

David
Rumelhart

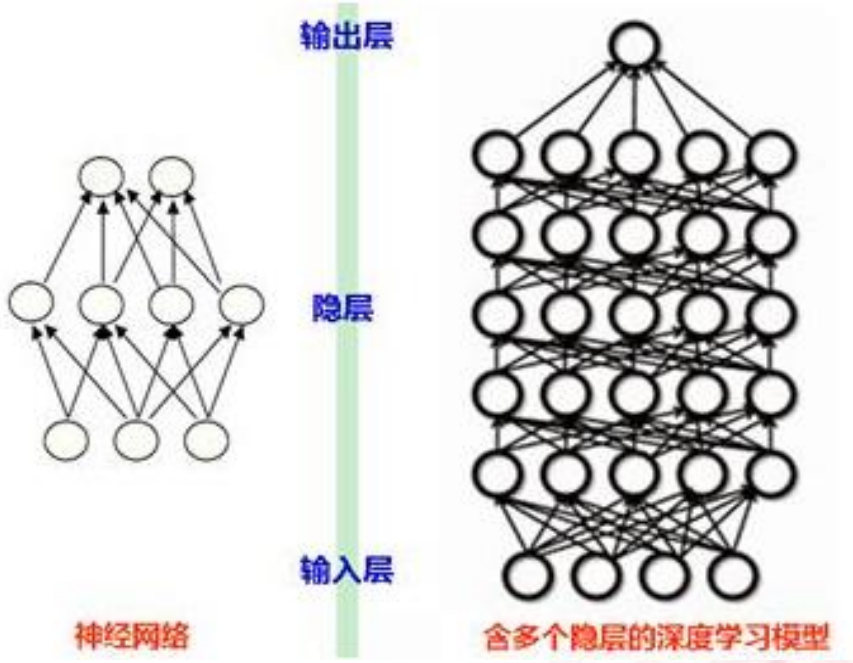
James.
McClelland

- 利用输出后的误差来估计输出层的直接前导层的误差，再用这个误差估计更前一层的误差，如此一层一层的反传下去，就获得了所有其他各层的误差估计。

深度学习：深层神经网络

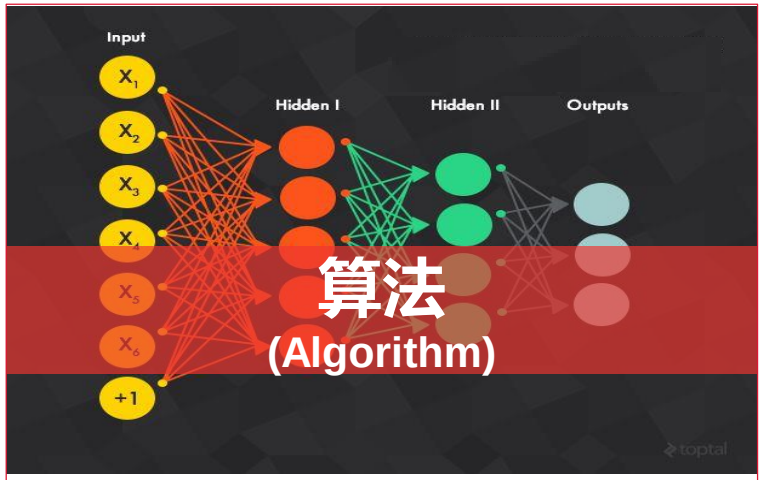
●深层神经网络

- 普适逼近原理(Universal approximation theorem)：
  - 单隐藏层神经网络可以逼近任意函数
- 神经网络的表达能力
  - 参数越多表达能力越强
  - 层数越多表达能力越强
  - 过拟合问题
- 深层神经网络泛化能力更强，训练难度更大
  - 逐层训练，改进训练算法
  - 增加训练样本，增加迭代次数
- 深度学习
  - 深层神经网络
  - 通过海量计算能力，海量训练样本进行训练



深度学习的特点

大量高性能硬件组成的计算能力。
如：分布式GPU集群



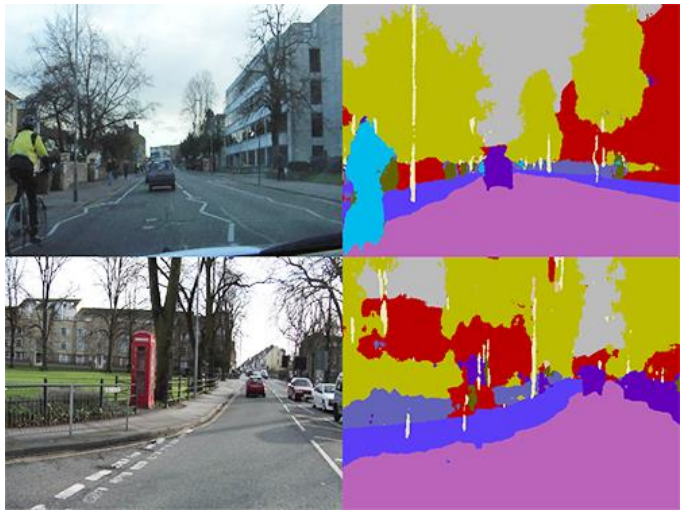
优秀的人工智能算法，比如现在最流行的深度学习相关的DNN、CNN、RNN、GAN…

数据是驱动人工智能取得更好的识别率和精准度的核心因素

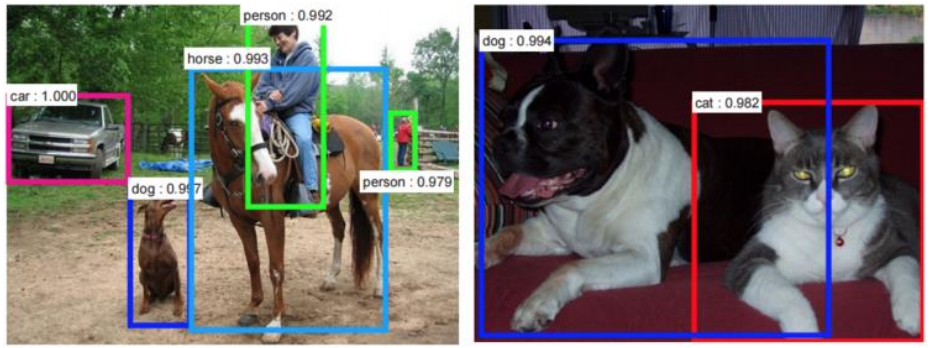
深度学习常用的几种模型

- 自动编码器：Auto-Encoder
- 卷积神经网络：Convolutional Neural Networks(CNN)
- 生成式对抗网络：Generative Adversarial Networks(GAN)
- 循环神经网络：Recurrent Neural Networks(RNN)
- 受限玻尔兹曼机：Restricted Boltzmann Machine(RBM)
- 深度信任网络：Deep Belief Networks(DBN)

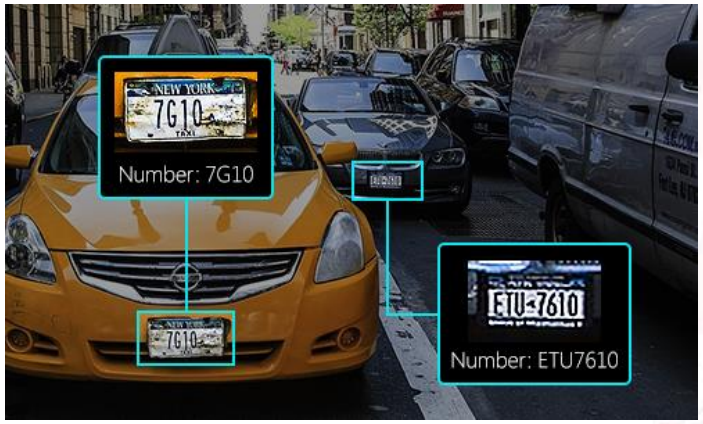
卷积神经网络



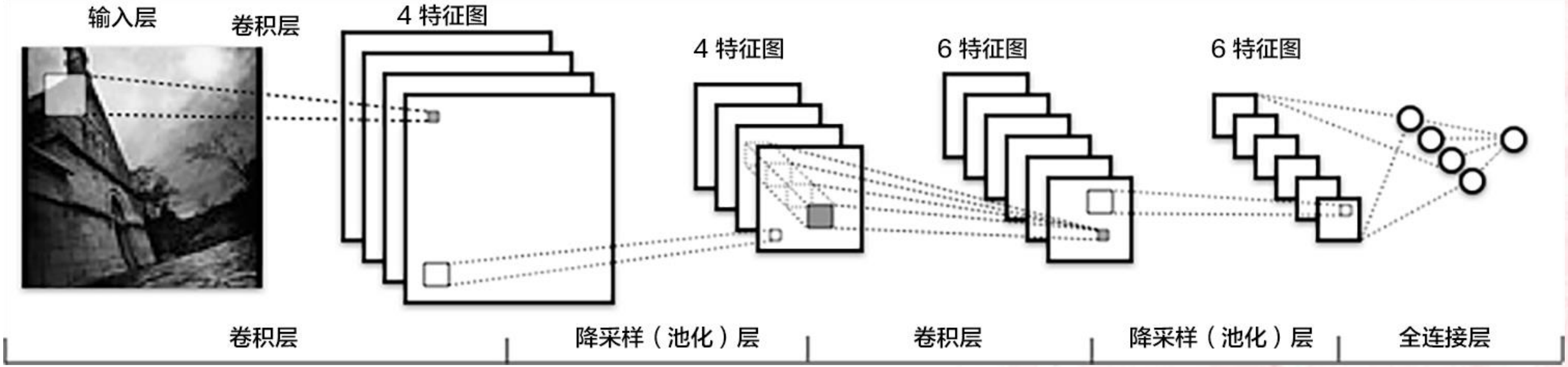
Segmentation



Detection



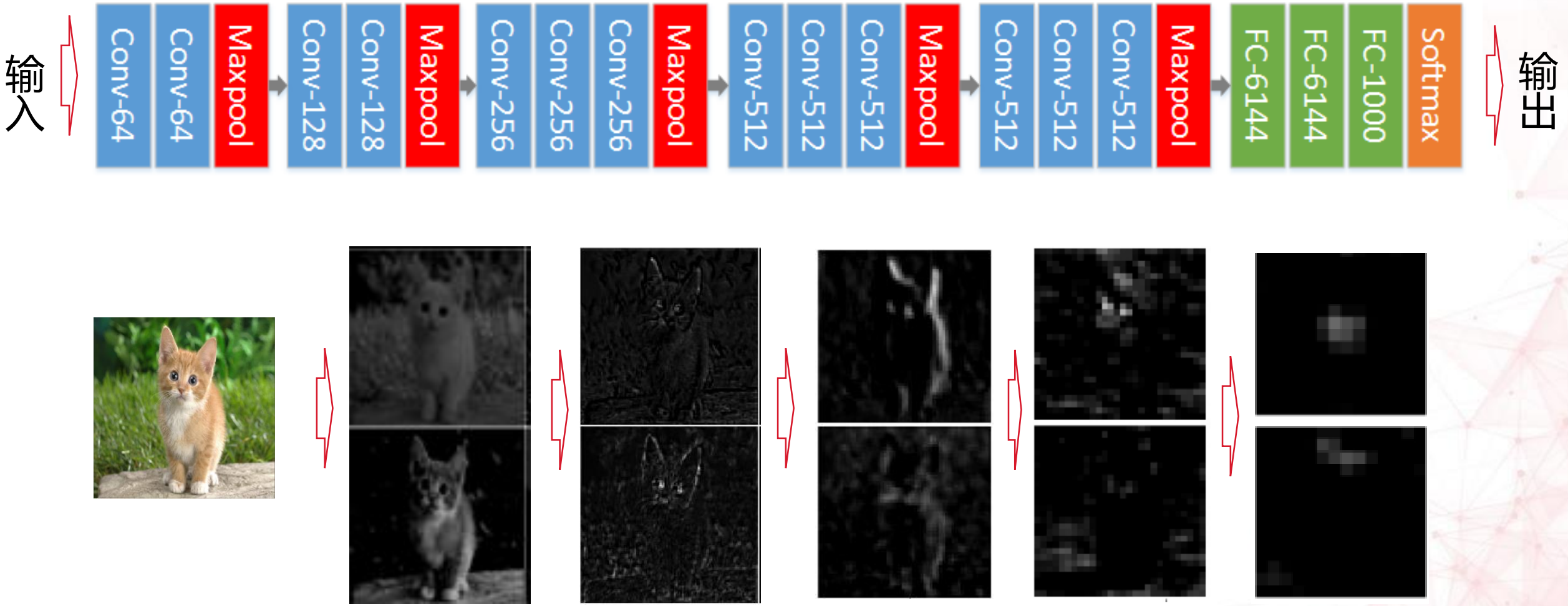
AutoDriving



卷积神经网络的基本结构

- 数据预处理层
  - 数据读取，数据增强，数据采样等等
- 卷积层
  - 提取特征的滤波过程，在图像领域常表现为模板(mask)运算
- 激活函数
  - 增加网络的非线性性，拓展了网络的表达能力
  - 常见的激活函数有三种：Sigmoid函数，tanh函数，ReLu函数
- 池化层
  - 减少数据量，抑制过拟合，提高鲁棒性(max-pooling, average-pooling)
- 全链接层
  - 主要用于分类，现在很多网络也可以不需要全链接层
- Loss层
  - 与需要解决的问题相关，分类常用的Loss为Softmax

卷积神经网络：特征提取示例



CONTENTS

目录

1 AI复兴：深度学习

2 计算优化：GPU

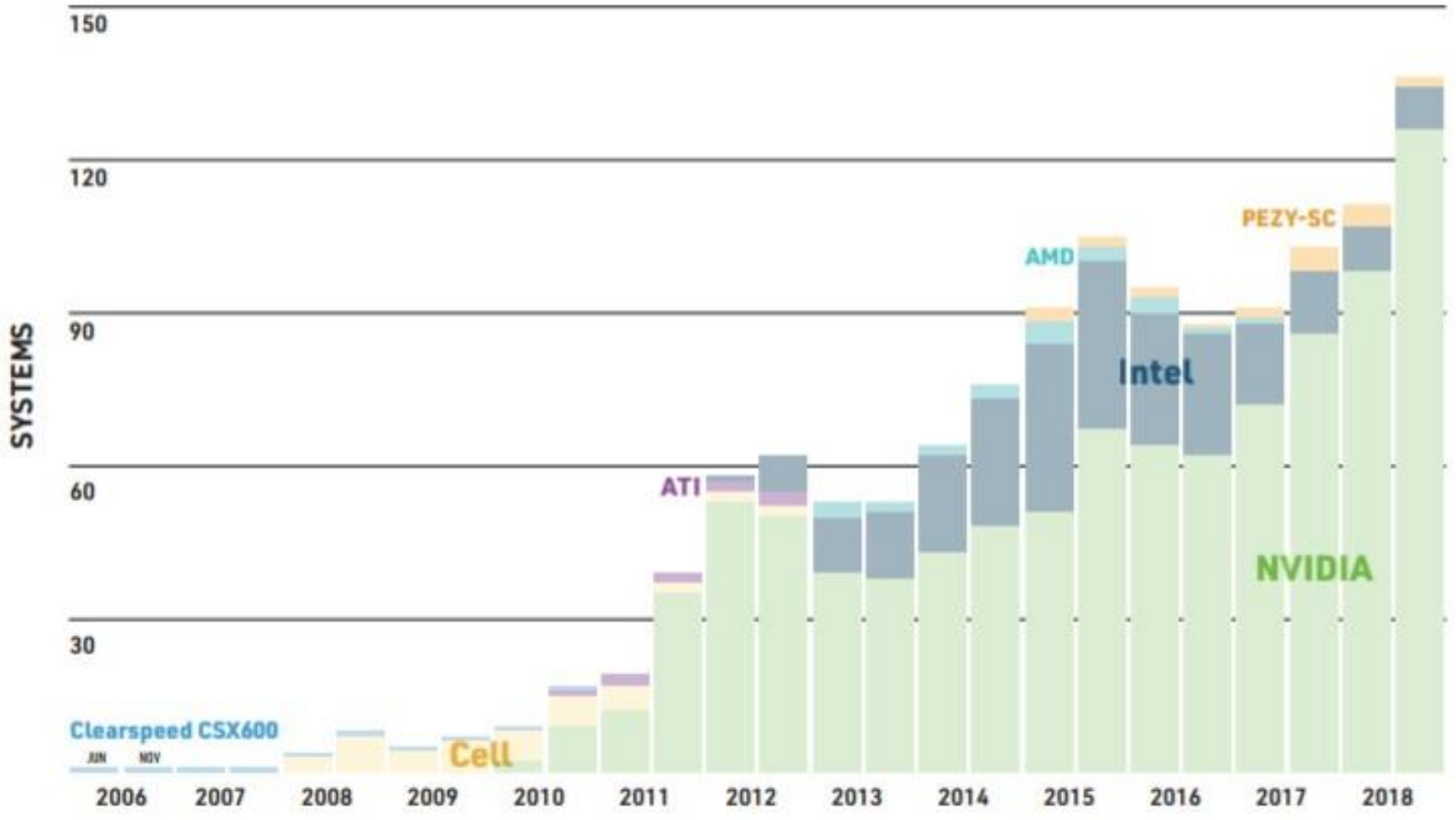
3 计算以外：内存&访存

4 携手共进：通信优化

5 无所不能：云上AI

6 轻松一刻：应用案例

异构计算的兴起



近年HPC TOP500 厂家份额统计

通用 • 复杂

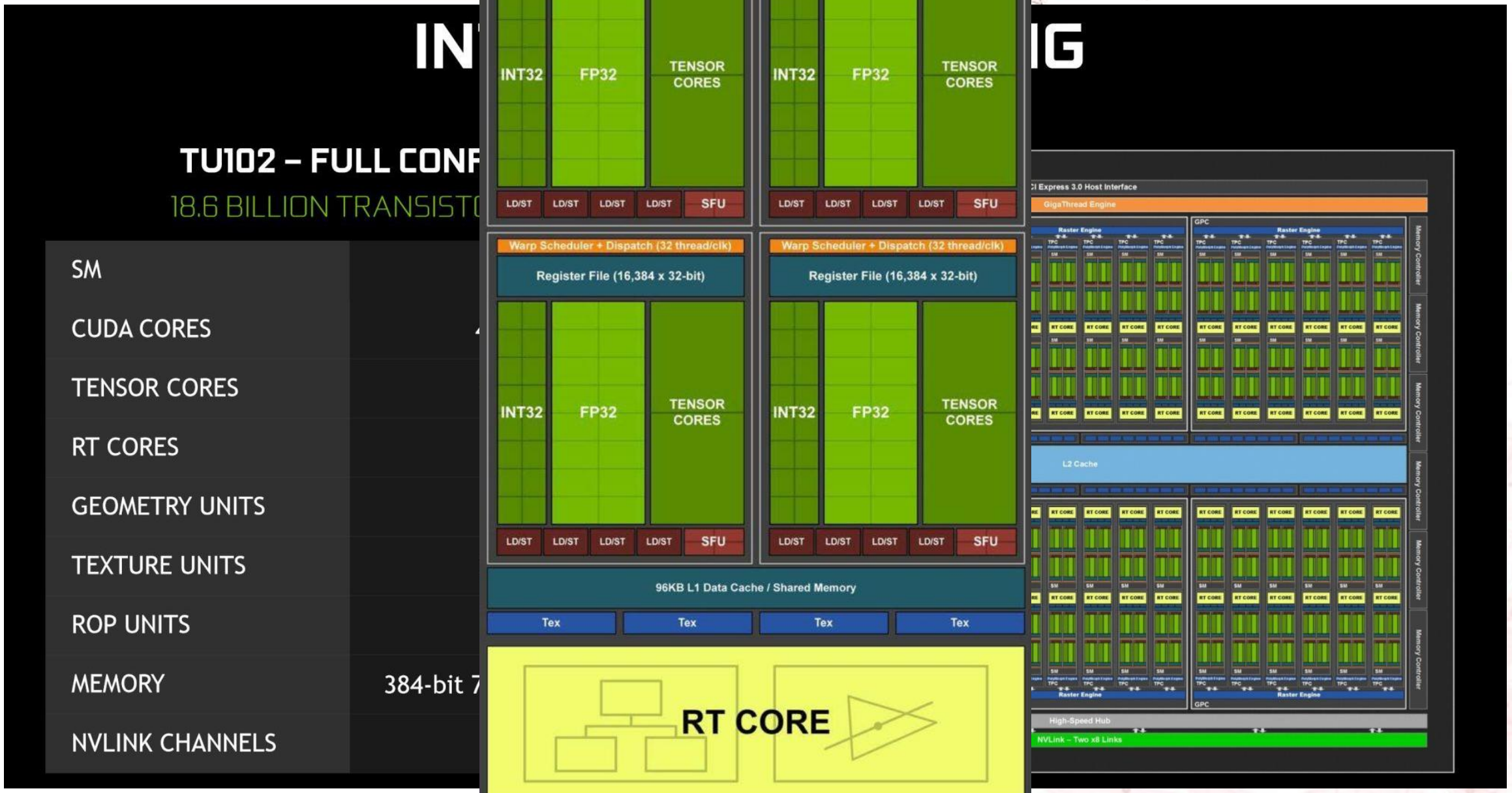
专用 • 精简

串行 • 串行为主

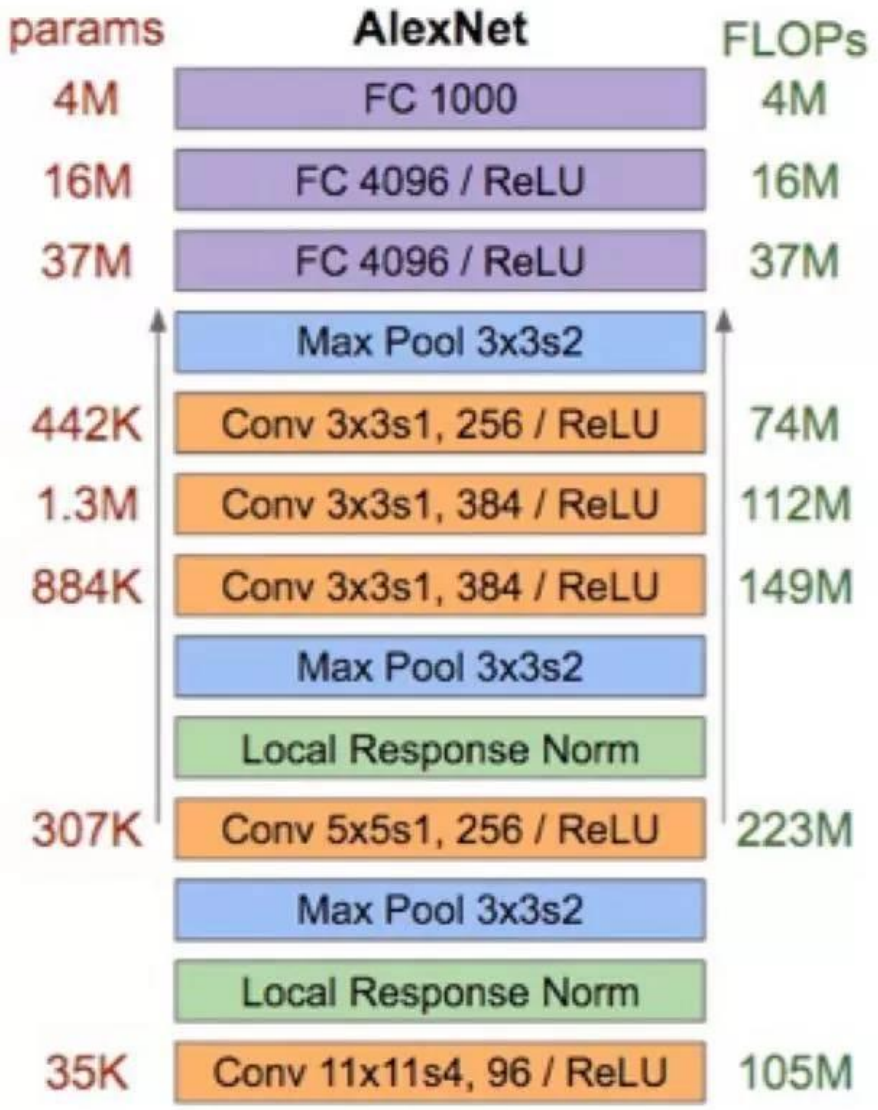
并行 • 并行为主

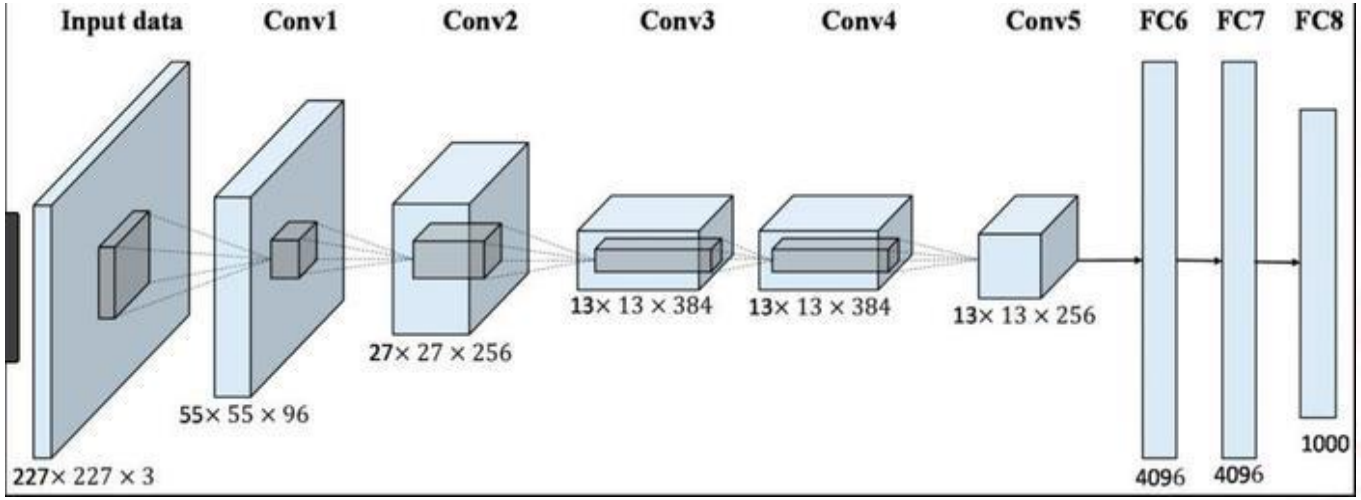
CPU和GPU的特点

Nvidia Turing GPU架



不同操作计算量统计





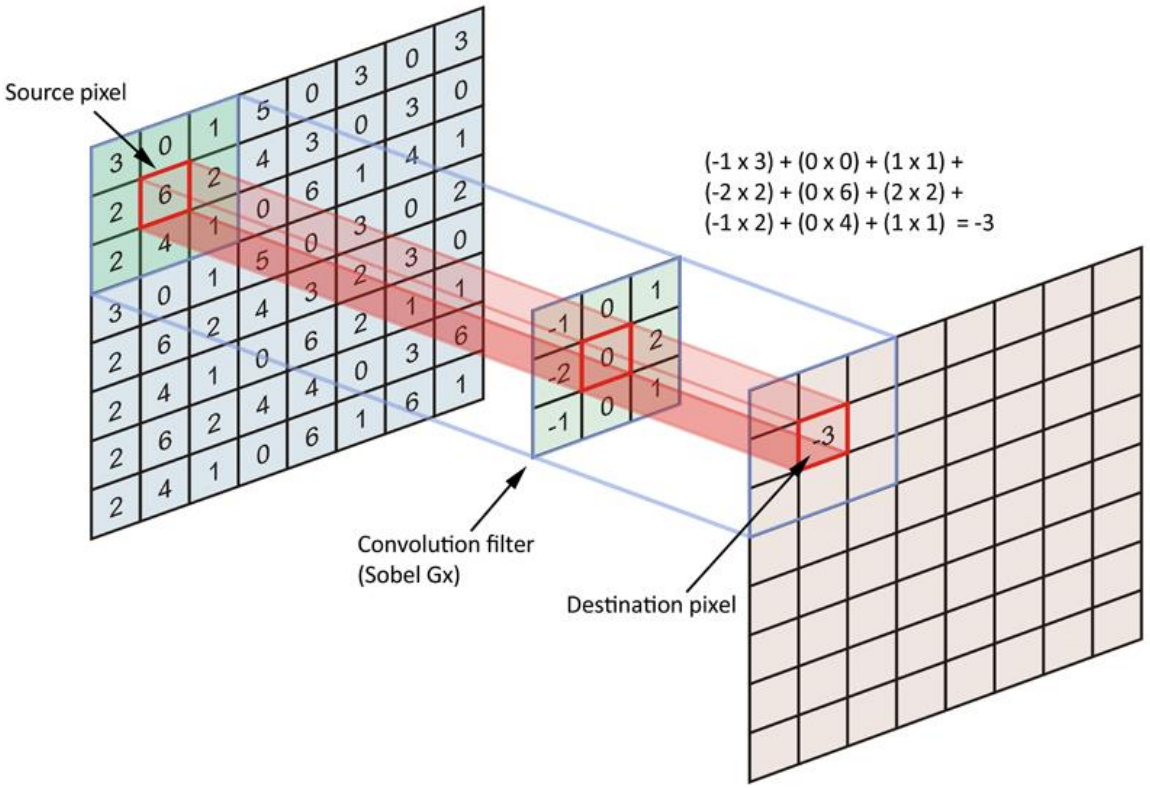
AlexNet	参数数量	参数占比	计算量	计算量占比
卷积层	~3M	5%	663M	92%
全链接层	~57M	95%	57M	8%

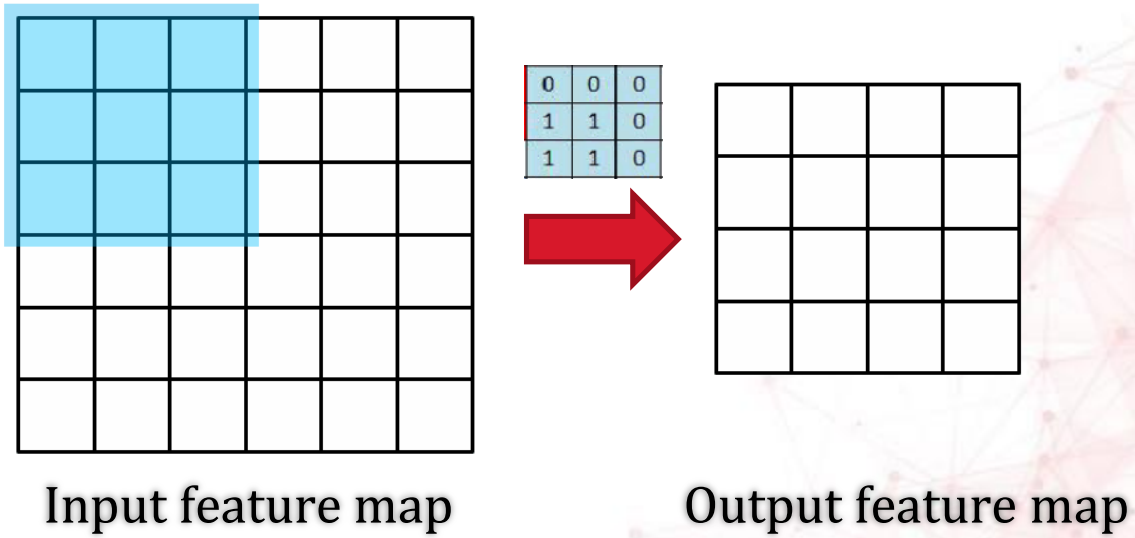
超过90%的计算量在卷积层

卷积计算方法

●卷积计算公式

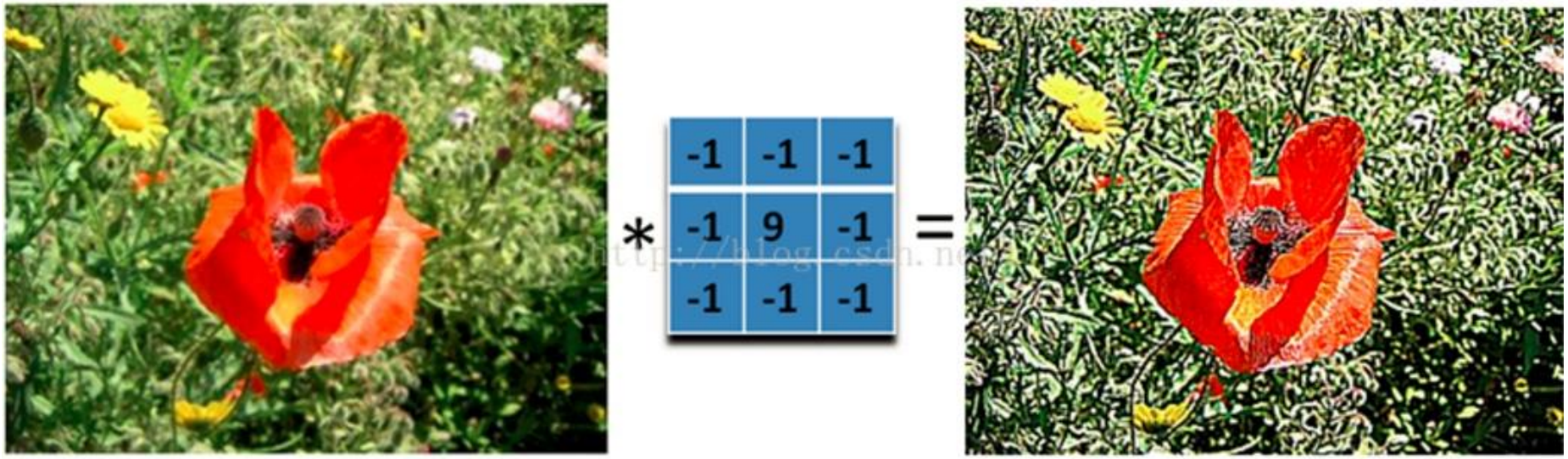
$$\text{Output}[i][j][n] = \sum_{m=0}^{M-1} \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} \text{Input}[i * S + u][j * S + v][m] \times \text{Filter}[n][u][v][m]$$



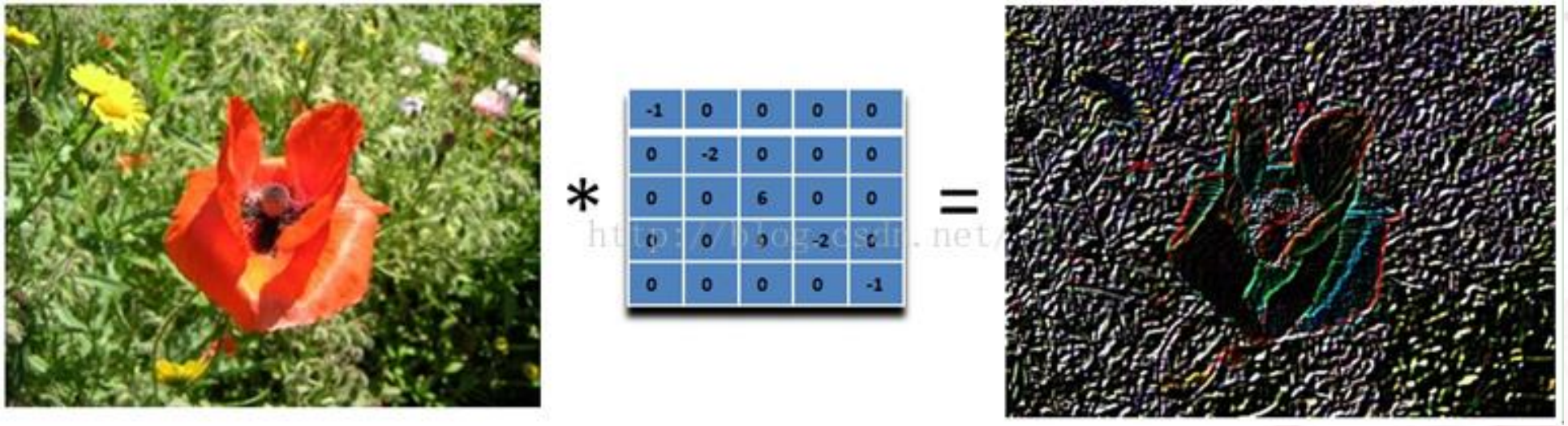


整张图片输入输出（无Padding）

卷积计算方法



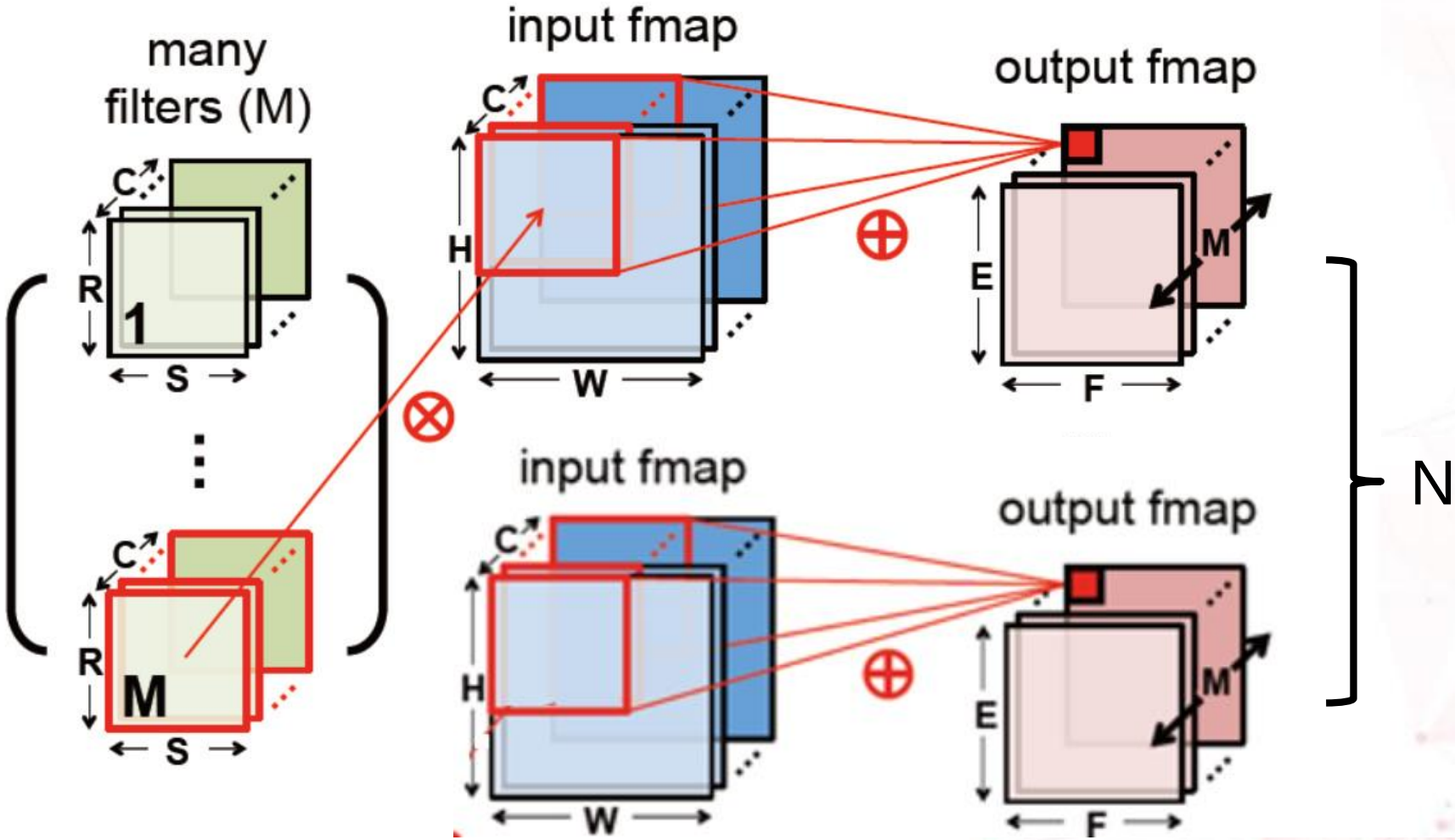
示例1：图像锐化



示例2：边缘检测

卷积神经网络中的卷积

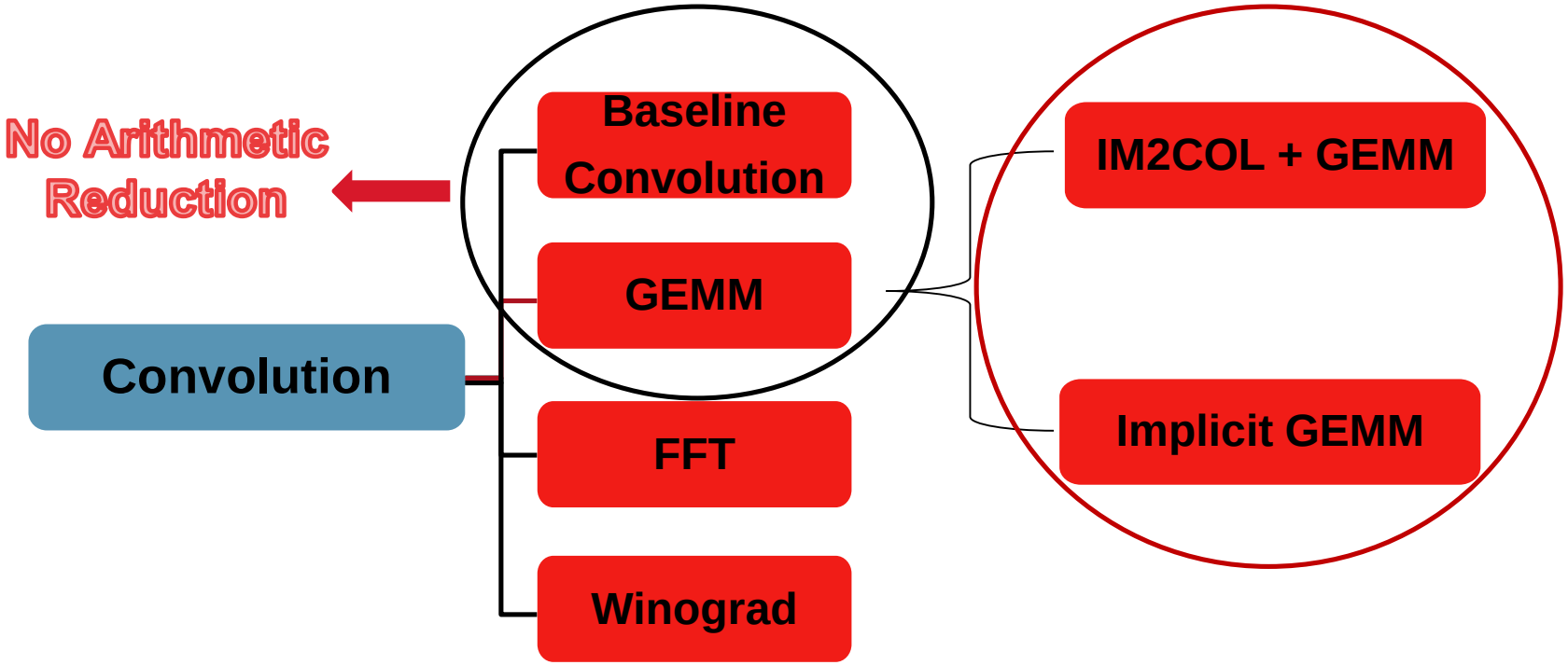
- 输入Feature Map
  - W：宽度
  - H：高度
  - C：通道数
  - N：样本数量
- Filter
  - S：宽度
  - R：高度
  - C：通道数
  - M：输出通道数
- 输出Feature Map
  - F：宽度
  - E：高度
  - M：通道数
  - N：样本数量



计算量= 2 ∗F*E*S*R*C*M*N

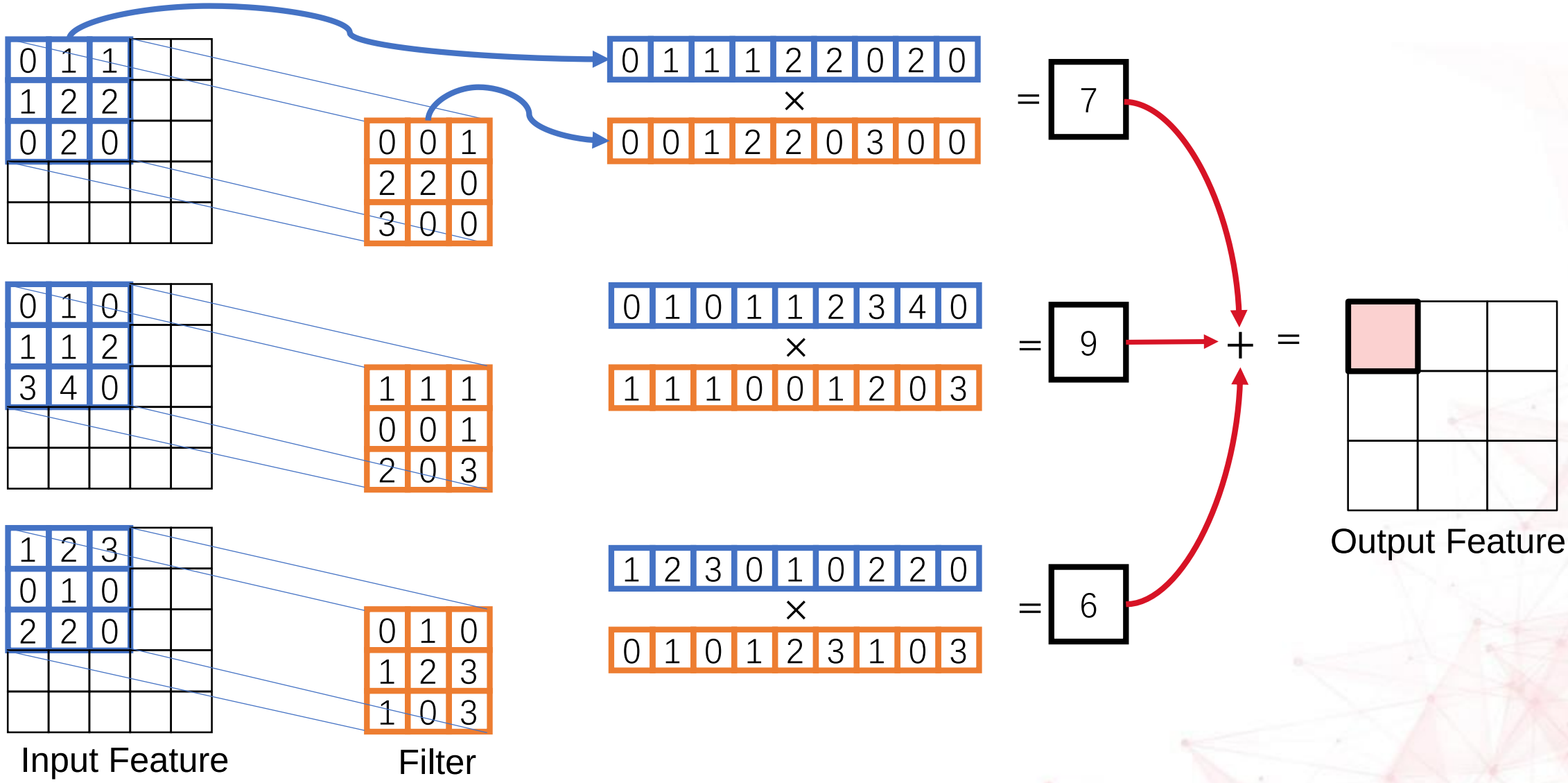
常用卷积计算方法

- 四种卷积计算方法



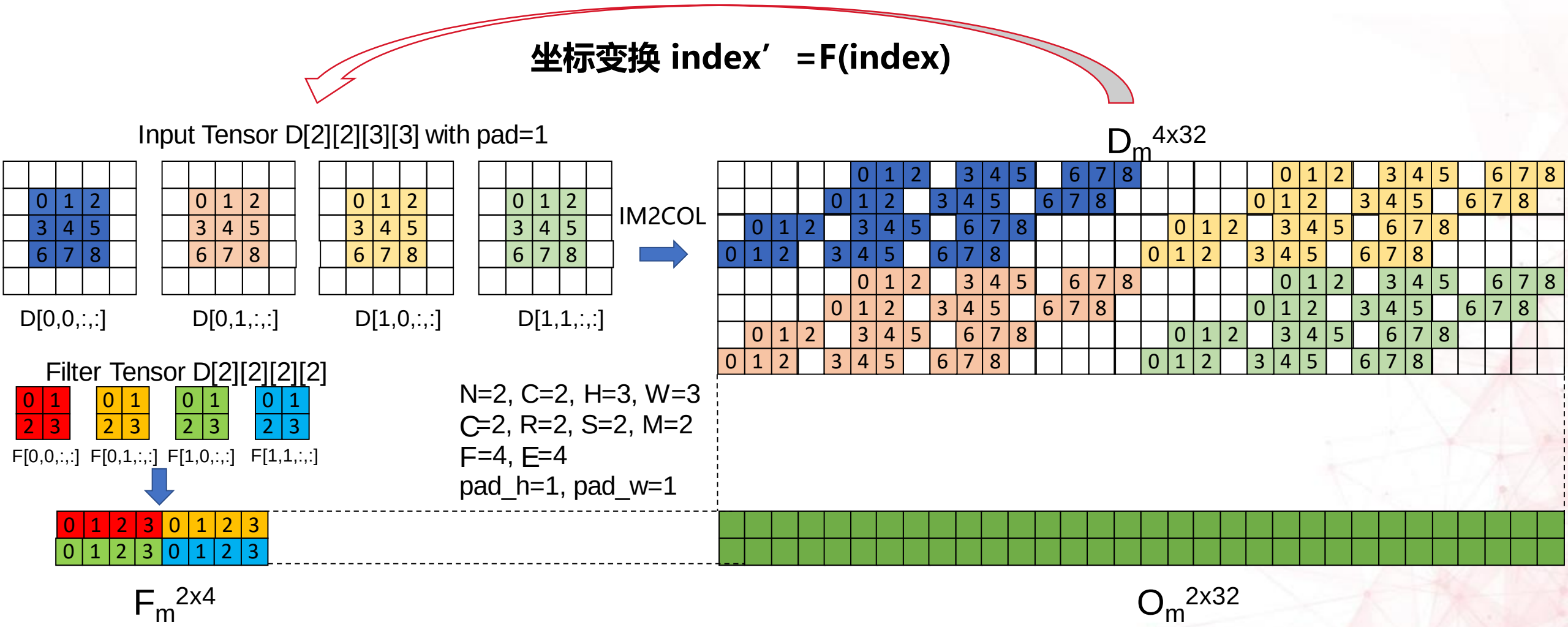
案例：GEMM优化

GEMM is the HEART of Convolution



IM2COL + GEMM

GEMM is the HEART of Convolution



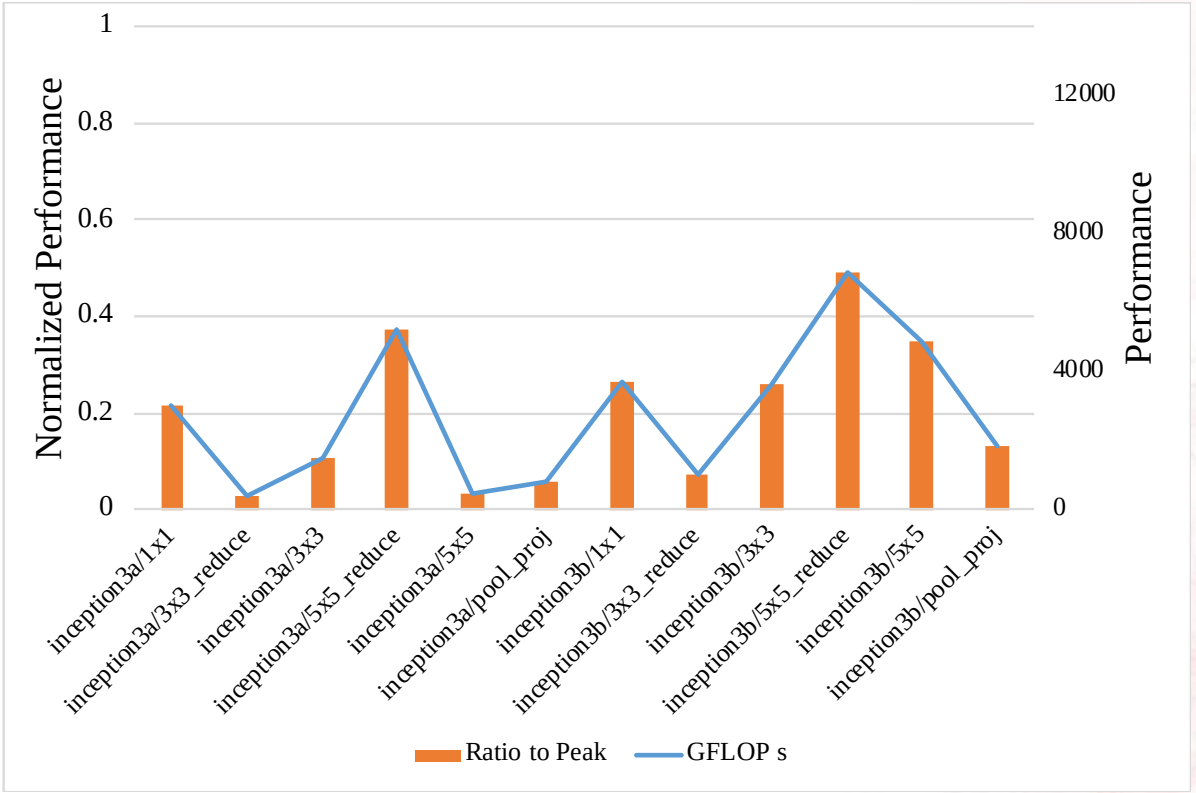
IM2COL + GEMM

Motivation： Small Matrix Problem

Nvidia Tesla V100： FP32 Peak performance 14Tflops

	m	k	N (Batch=1)	GFLOPS	Ratio to Peak
inception3a_1	96	192	784	3011.18	0.215084
Inception3a_2	16	192	784	416.84	0.029774
Inception3a_3	64	192	784	1497.77	0.106984
Inception3a_4	128	864	784	5193.64	0.370974
Inception3a_5	32	400	256	470.9	0.033636
Inception3a_6	32	192	784	818.79	0.058485
Inception4a_1	128	256	784	3676.79	0.262628
Inception4a_2	32	256	784	1012.76	0.07234
Inception4a_3	128	256	784	3667.59	0.261971
Inception4a_4	192	1152	676	6910.33	0.493595
Inception4a_5	96	800	784	4888.3	0.349164
Inception4a_6	64	256	784	1845.13	0.131795

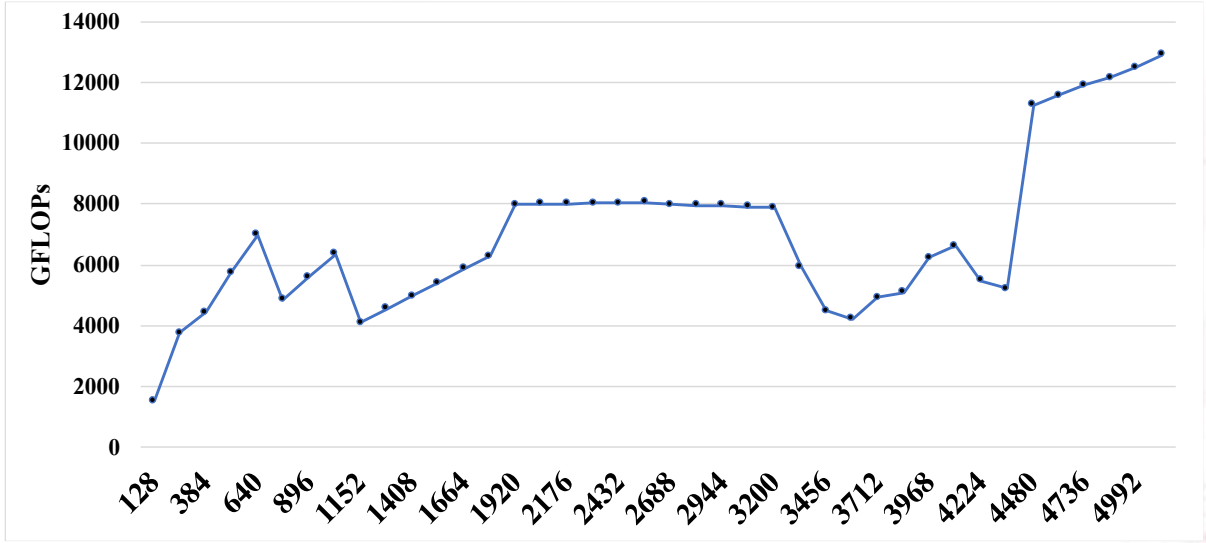
(a) Matrix Size of some convolution



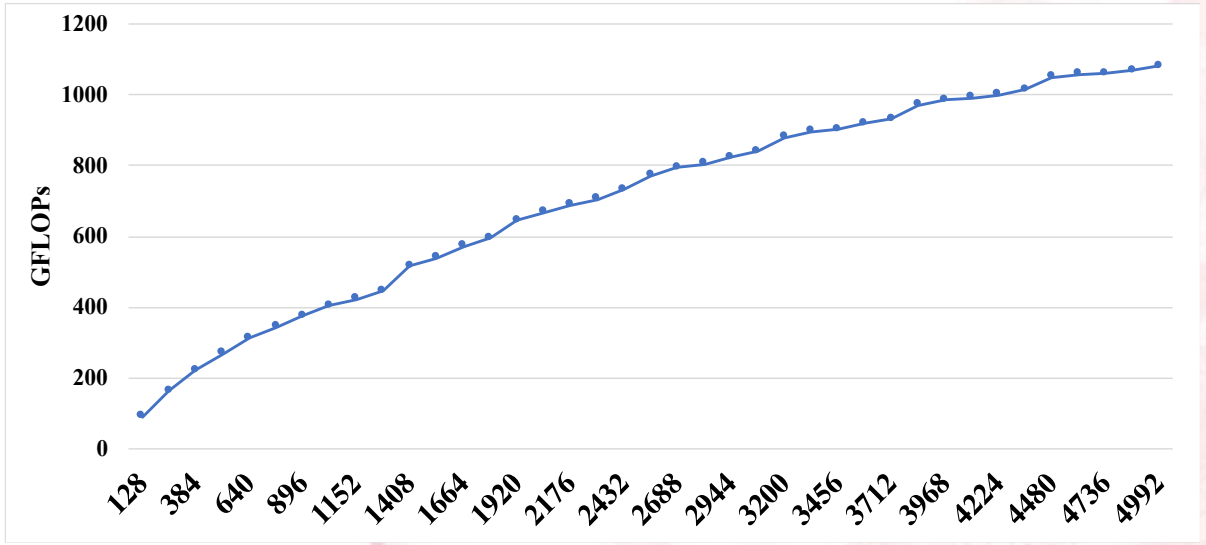
(b) Performance and Normalized Performance

Issues on Small Matrix

- M and N determine tile size and the number of tiles
  - impact the thread-level parallelism **(TLP)**
  - Tiling Strategy
- K determines the workload of each tile
  - Impact on instruction-level parallelism **(ILP)** within a single thread
  - Software Pipelining

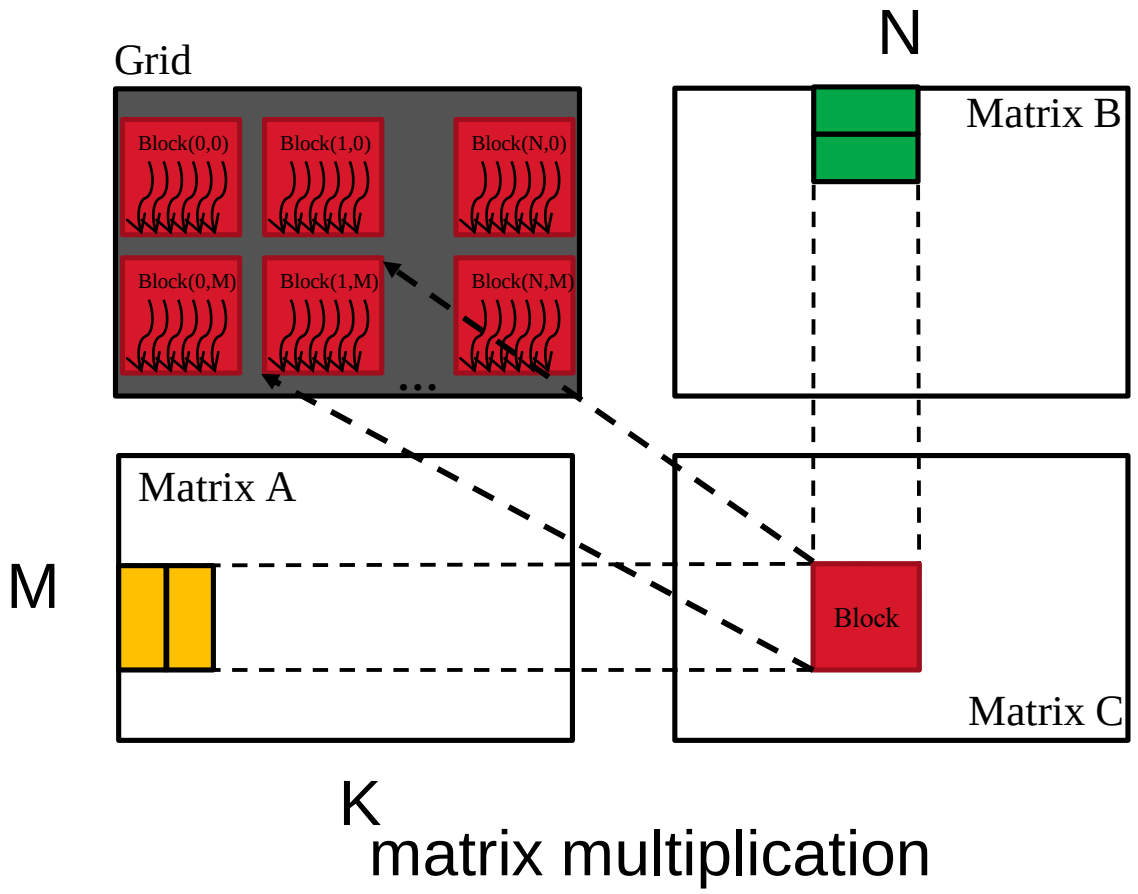


$\mathrm{M}\ (64)\ \times 5120 \times N$



$64 \times K \times 64$

Tiling Strategy

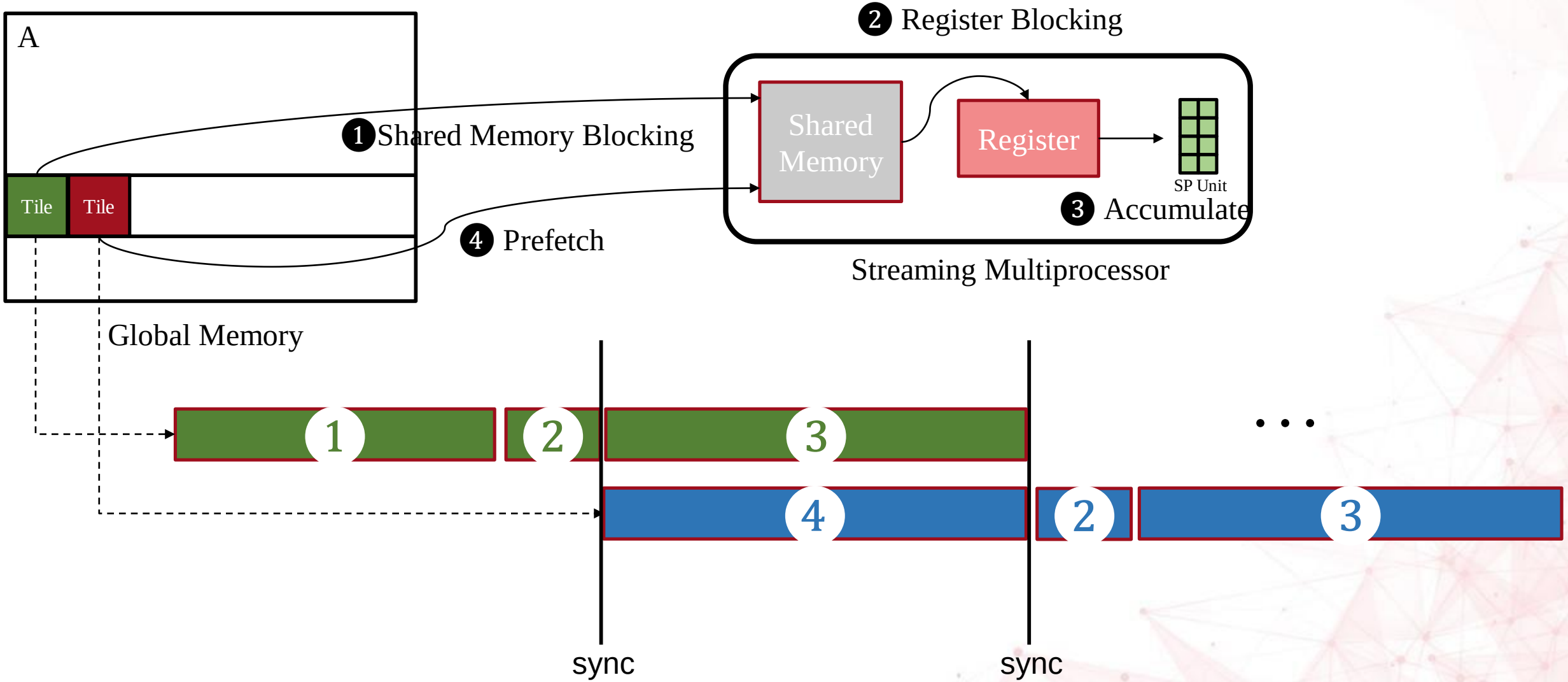


Tiling Strategy	BY	BX	Thread
Small	16	16	32
Medium	32	32	64
Large	64	64	64
Tall	128	64	128
Wide	64	128	128
Huge	128	128	256

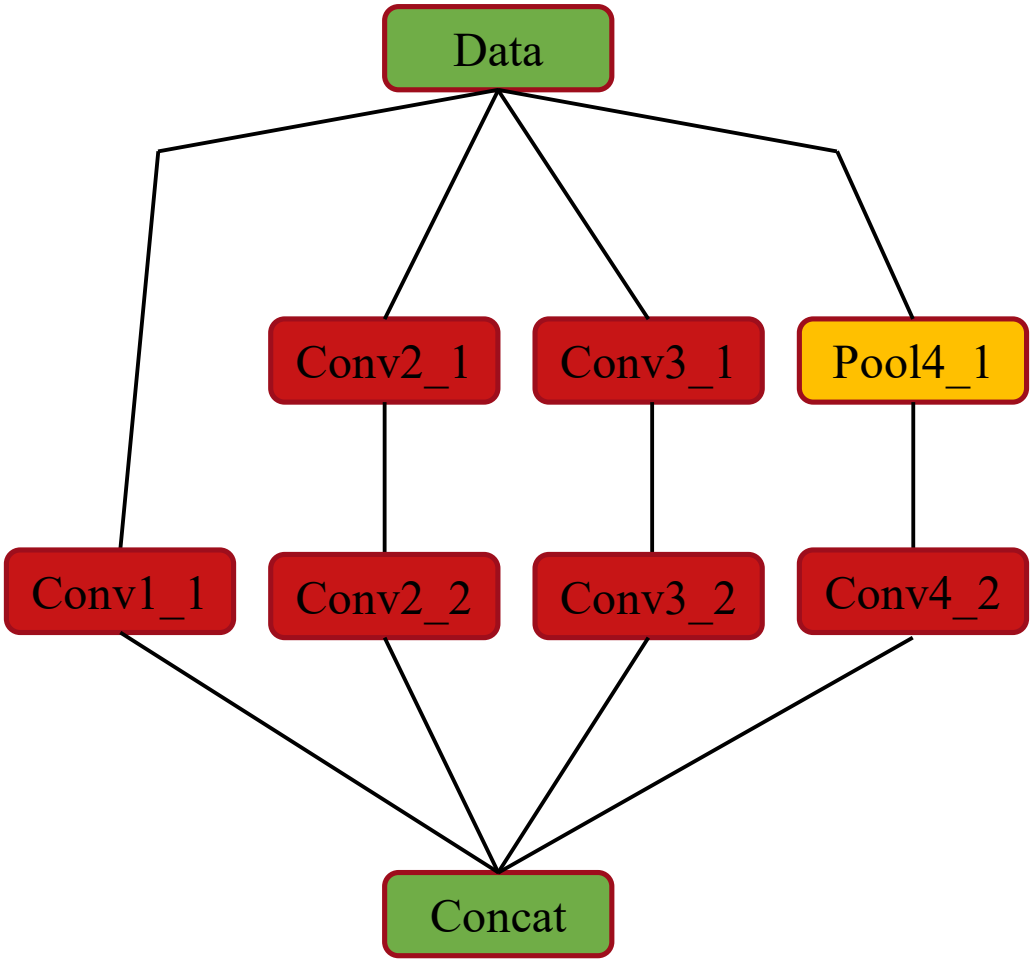
$$\text{计算访存比} = \frac{M*N*K}{(M+N)*K+M*N}$$

Prefers large tile size to hide the memory latency.

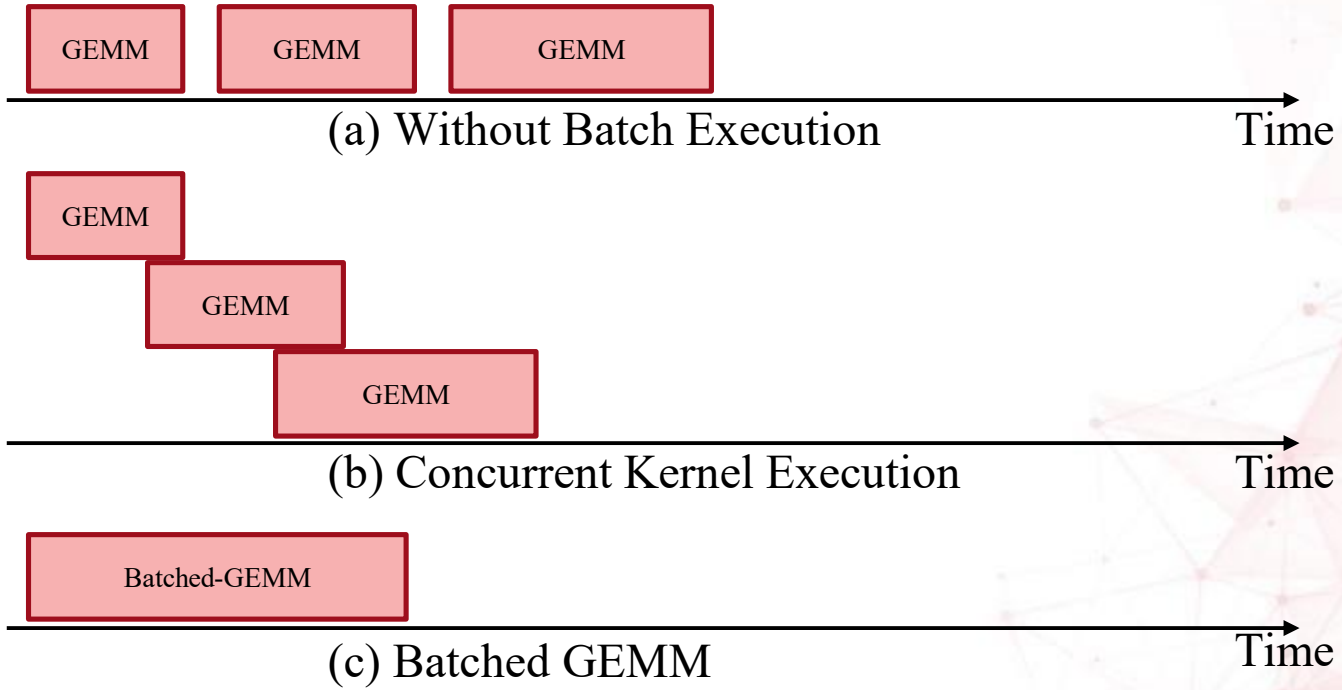
Software Pipelining



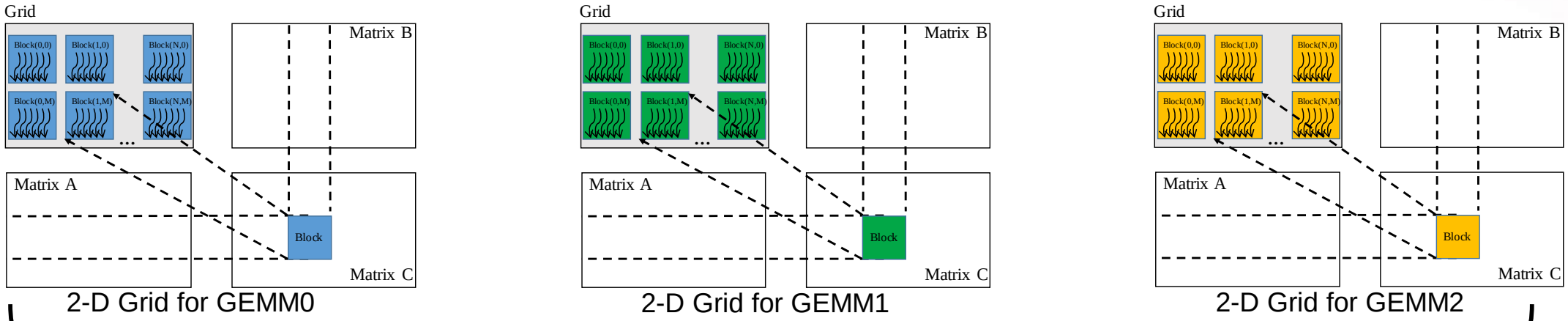
Batched GEMM



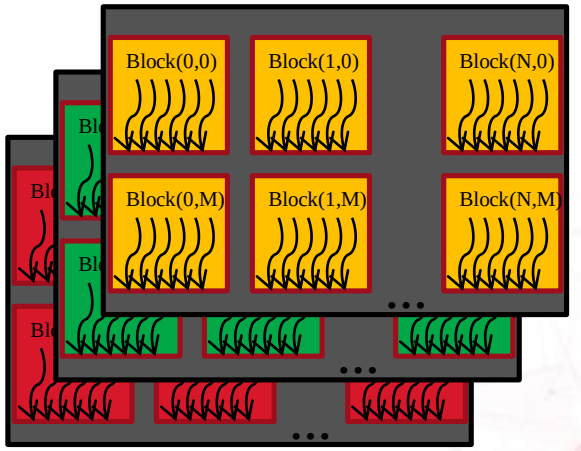
A fan-structure in Google-Net



Batched GEMM



Merge three 2D CUDA kernels into a 3D CUDA kernel

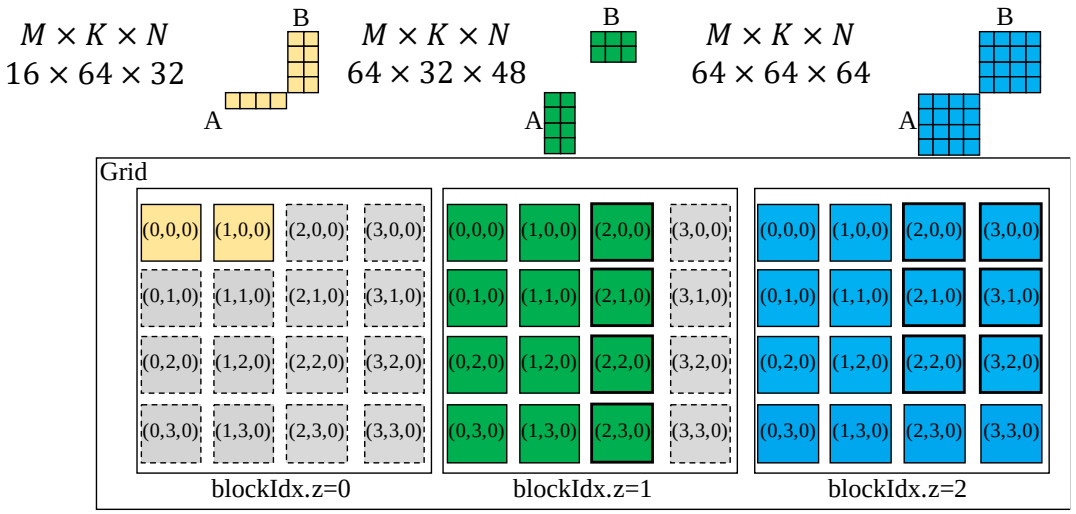


3-D Grid for Batched GEMM

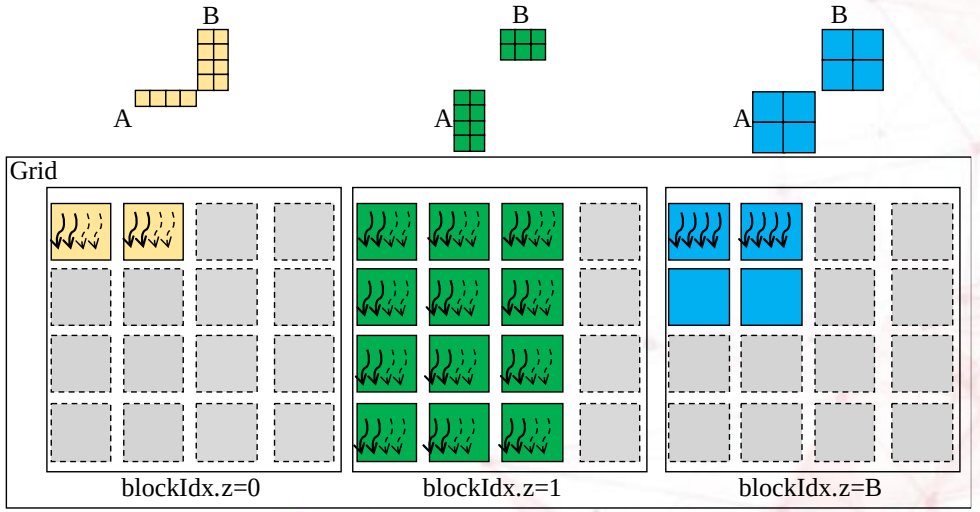
Challenges of Batched GEMM

- How to choose proper tile size for each GEMM?
- Thread Under-utilized?

Tiling Strategy	BY	BX	Thread
Small	16	16	32
Medium	32	32	64
Large	64	64	64
Tall	128	64	128
Wide	64	128	128
Huge	128	128	256



(a) Each GEMM uses the same Tile size 16 × 16.



(b) The third tile size 32 × 32.

Tiling Engine

- Design Tiling Strategy dedicated for Batched GEMM scenario

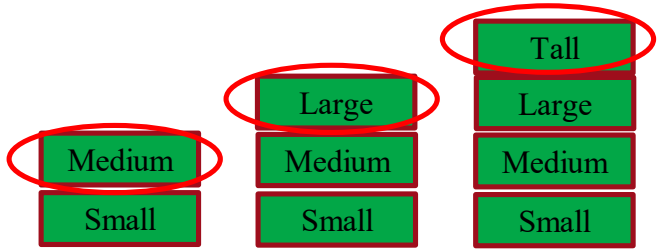
Tiling Strategy	BY	BX	256-Thread (Sub-Tile per Thread)	128-Thread (Sub-Tile per Thread)
Small	16	16	1x1	2x1
Medium	32	32	2x2	4x2
Large	64	64	4x4	8x4
Tall	128	64	8x4	8x8
Wide	64	128	4x8	8x8
Huge	128	128	8x8	16x8

Tiling Engine

Tiling Strategy	BY	BX
Small	16	16
Medium	32	32
Large	64	64
Tall	128	64
Wide	64	128
Huge	128	128

1. Obtain applicable tiling strategies for each GEMM
2. First try the small tiling strategy to guarantee TLP
3. Try the larger tiling strategy
4. The bottom of each bucket is the selected tiling strategies

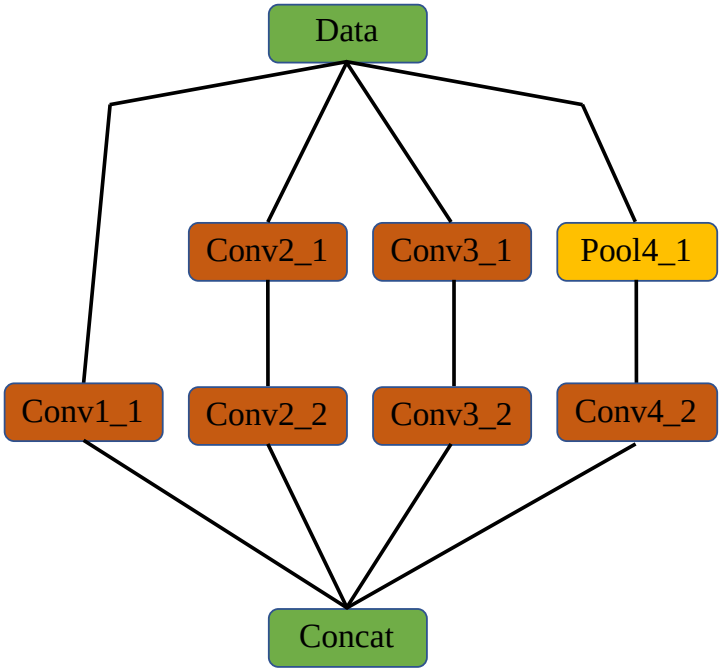
	GEMM 1	GEMM 2	GEMM 3
$M \times N \times K$	$32 \times 64 \times 16$	$64 \times 64 \times 32$	$128 \times 64 \times 32$



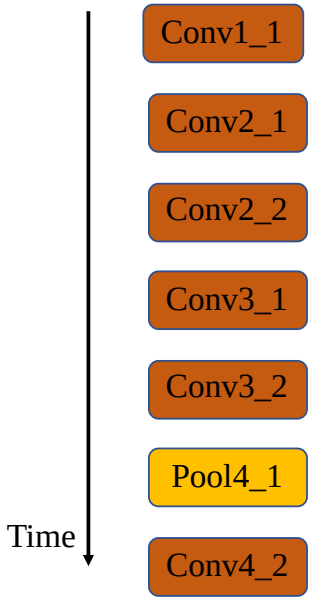
TLP

$TLP > Threshold? Larger: END$

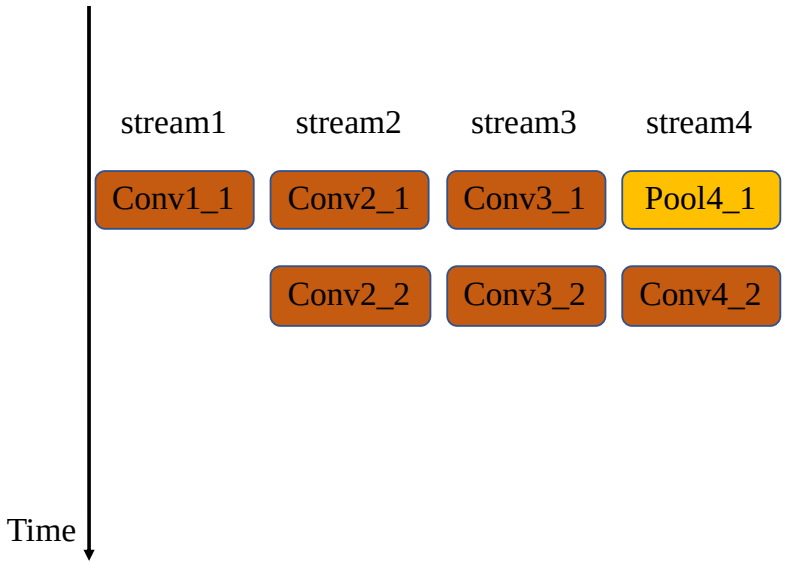
Evaluation – Google-Net



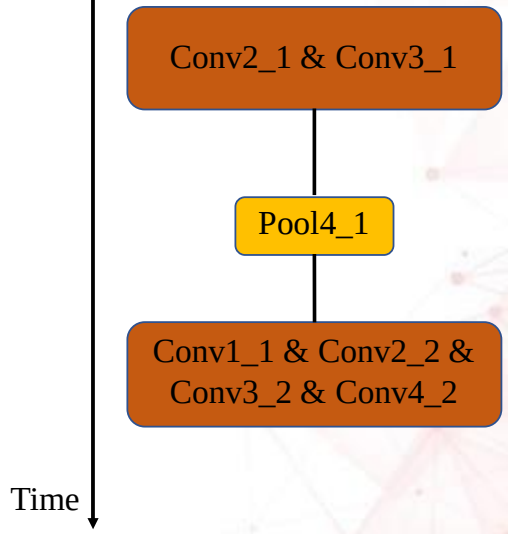
(a) A fan-structure in Google-Net



(b) Default Single-Stream Execution



(c) Multi-Stream Execution



(d) Batched Convolution

Evaluation – Real-world Case Study

- Platform
  - NVIDIA V100 GPU with CUDA 9.0
  - Batch size =1

Implementation	Time (ms)	Speedup
cuDNN without stream	3.18	1X
cuDNN with stream	2.41	1.31X
Batched GEMM	2.01	1.58X

CONTENTS

目录

1 AI复兴：深度学习

2 计算优化：GPU

3 计算以外：内存&访存

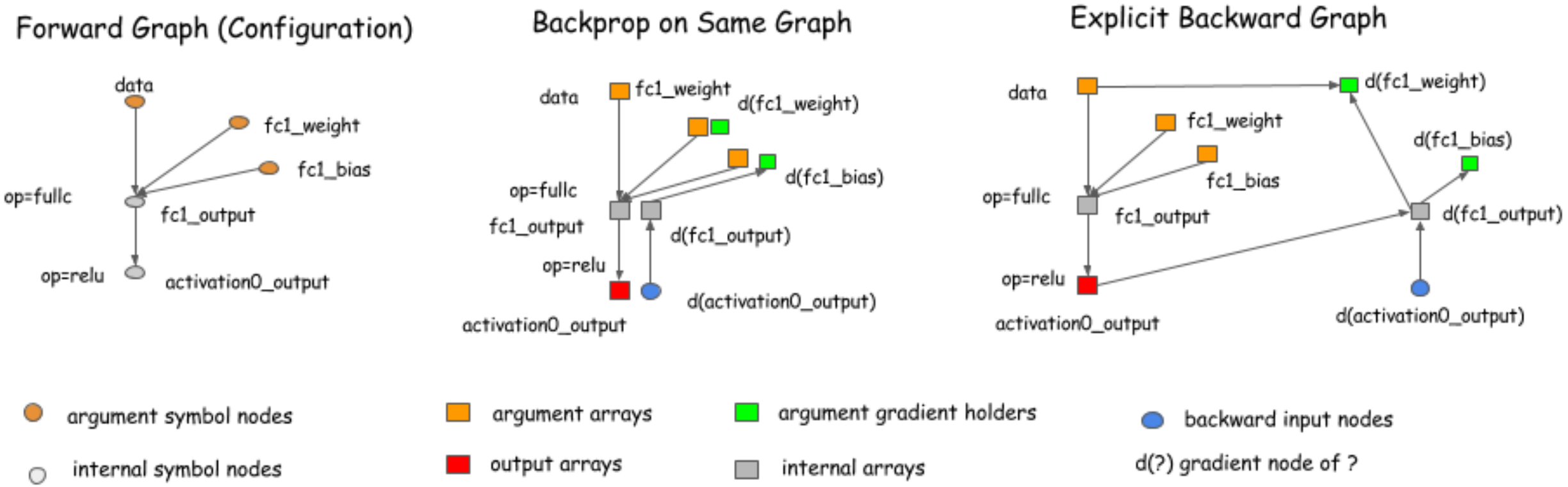
4 携手共进：通信优化

5 无所不能：云上AI

6 轻松一刻：应用案例

计算以外：内存优化

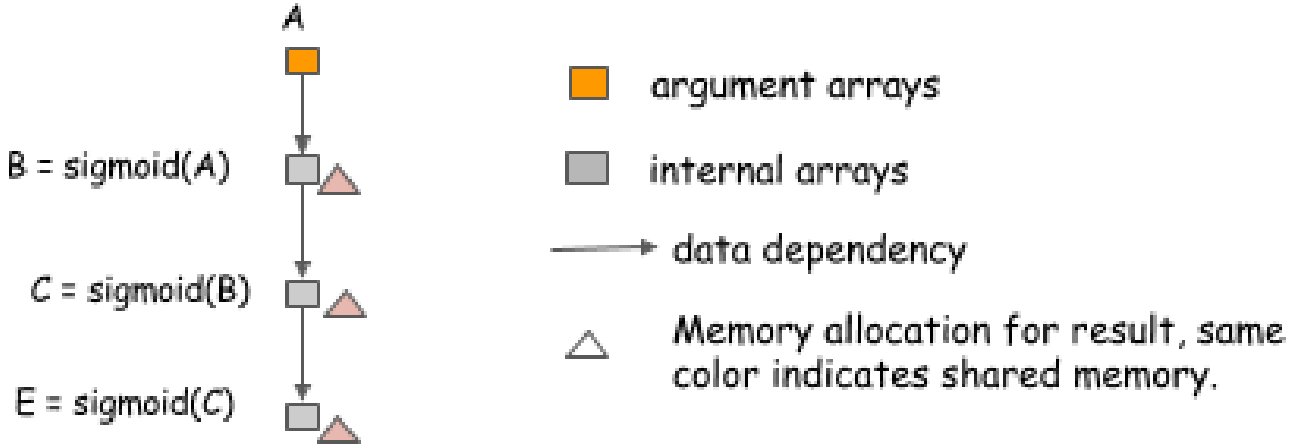
- 深度神经网络的计算过程可以用一个图来进行描述
  - 分为前向计算和反向传播两个过程



参考：https://mxnet.incubator.apache.org/architecture/note_memory.html

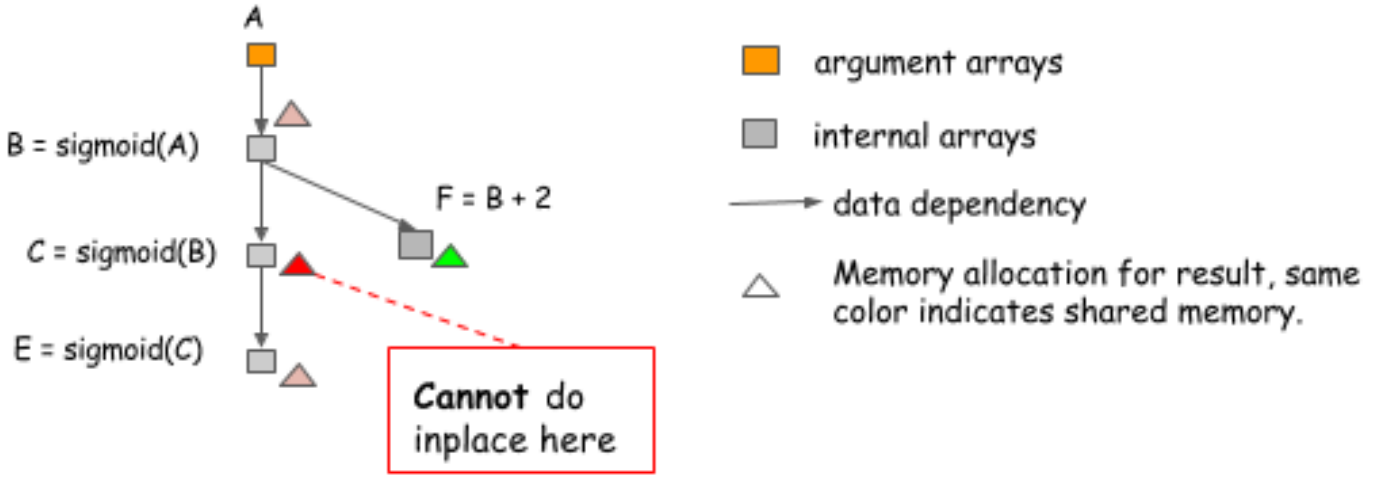
计算以外：内存优化

■ 优化方法1：In-place Operation



- 激活函数层往往可以使用In-Place操作

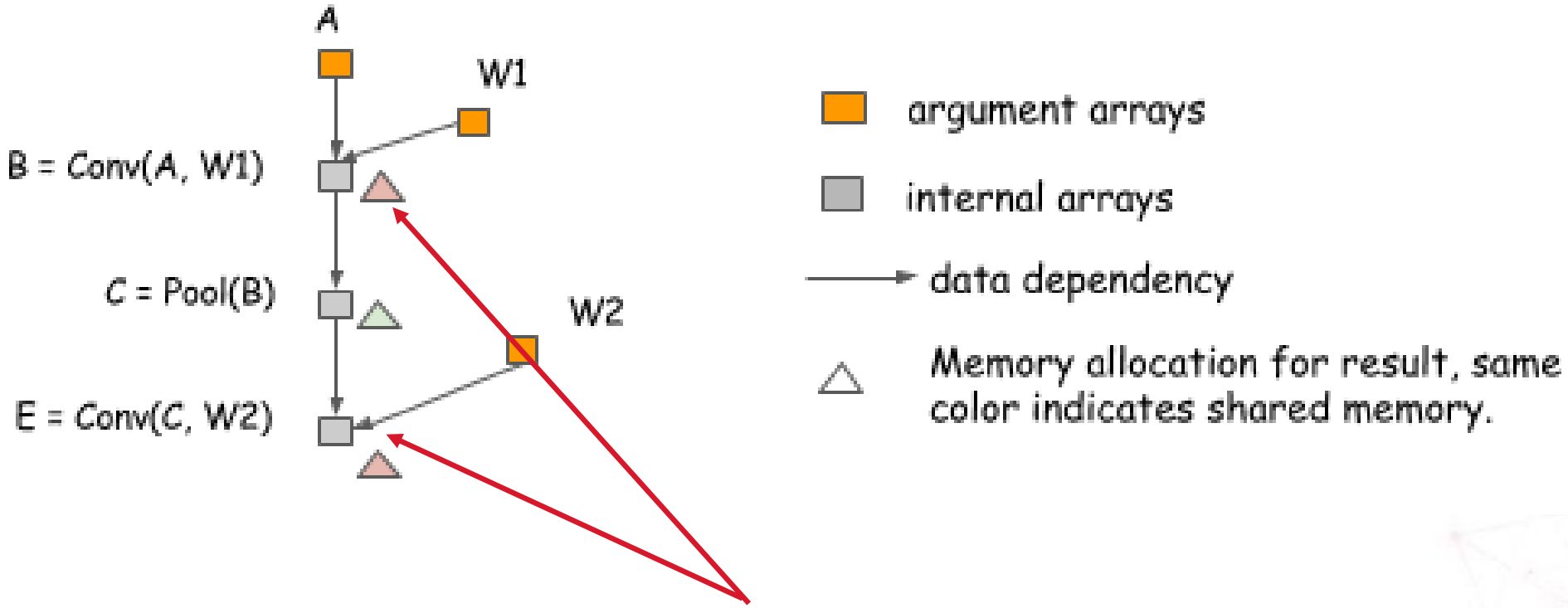
- 带有分支的激活函数不能使用In-Place操作



参考：https://mxnet.incubator.apache.org/architecture/note_memory.html

计算以外：内存优化

优化方法2：Memory Sharing

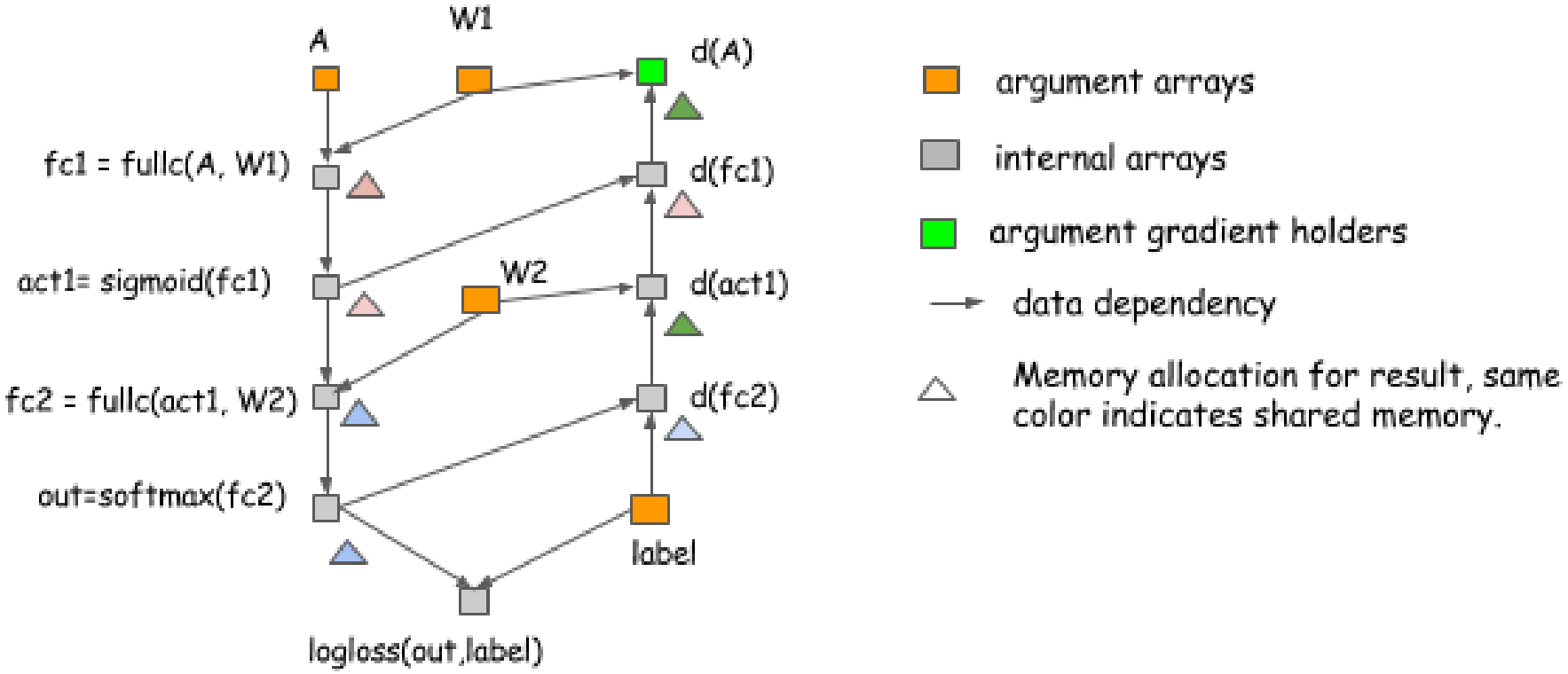


在计算E的时候，B的内存可以释放

参考：https://mxnet.incubator.apache.org/architecture/note_memory.html

计算以外：内存优化

■ 实际案例：In-Place Operation & Memory Sharing



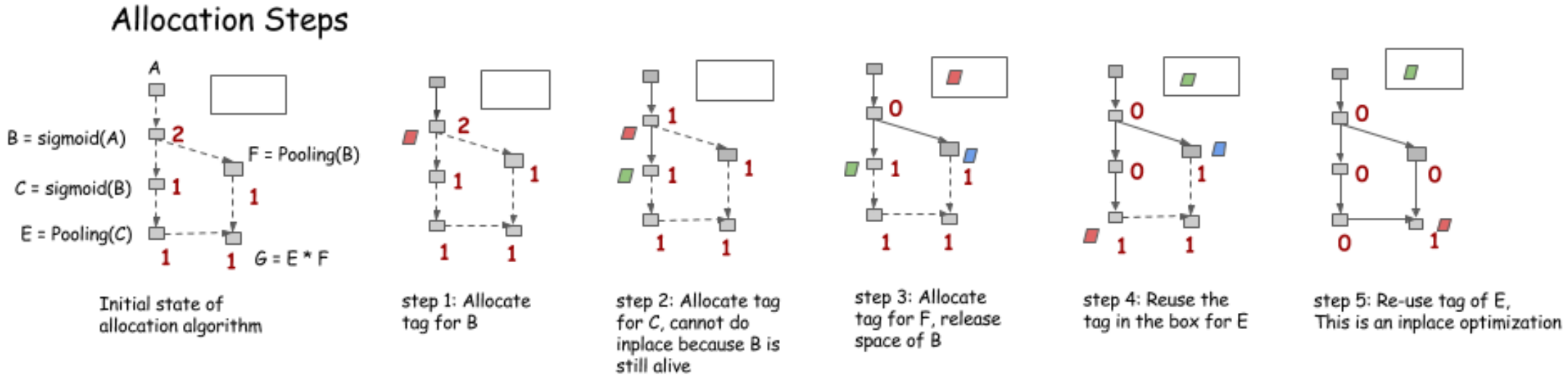
相同颜色的三角形复用内存

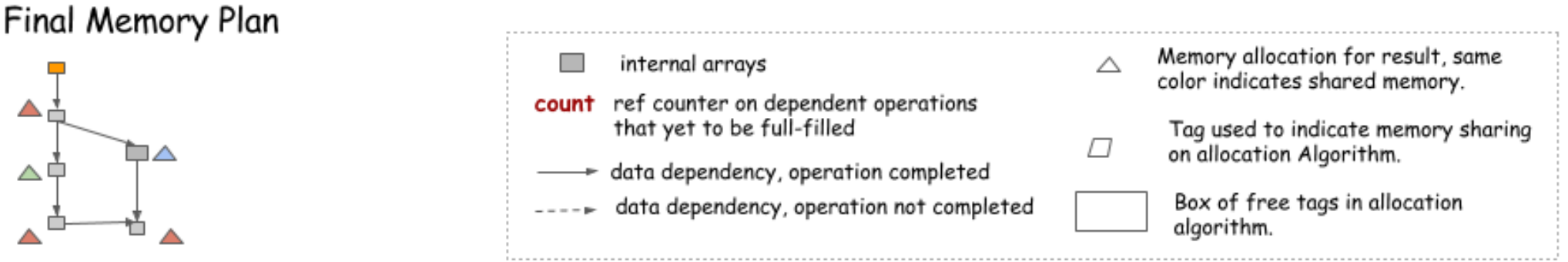
参考：https://mxnet.incubator.apache.org/architecture/note_memory.html

计算以外：内存优化

如何实现？

- 内存申请算法，类似编译器中寄存器分配

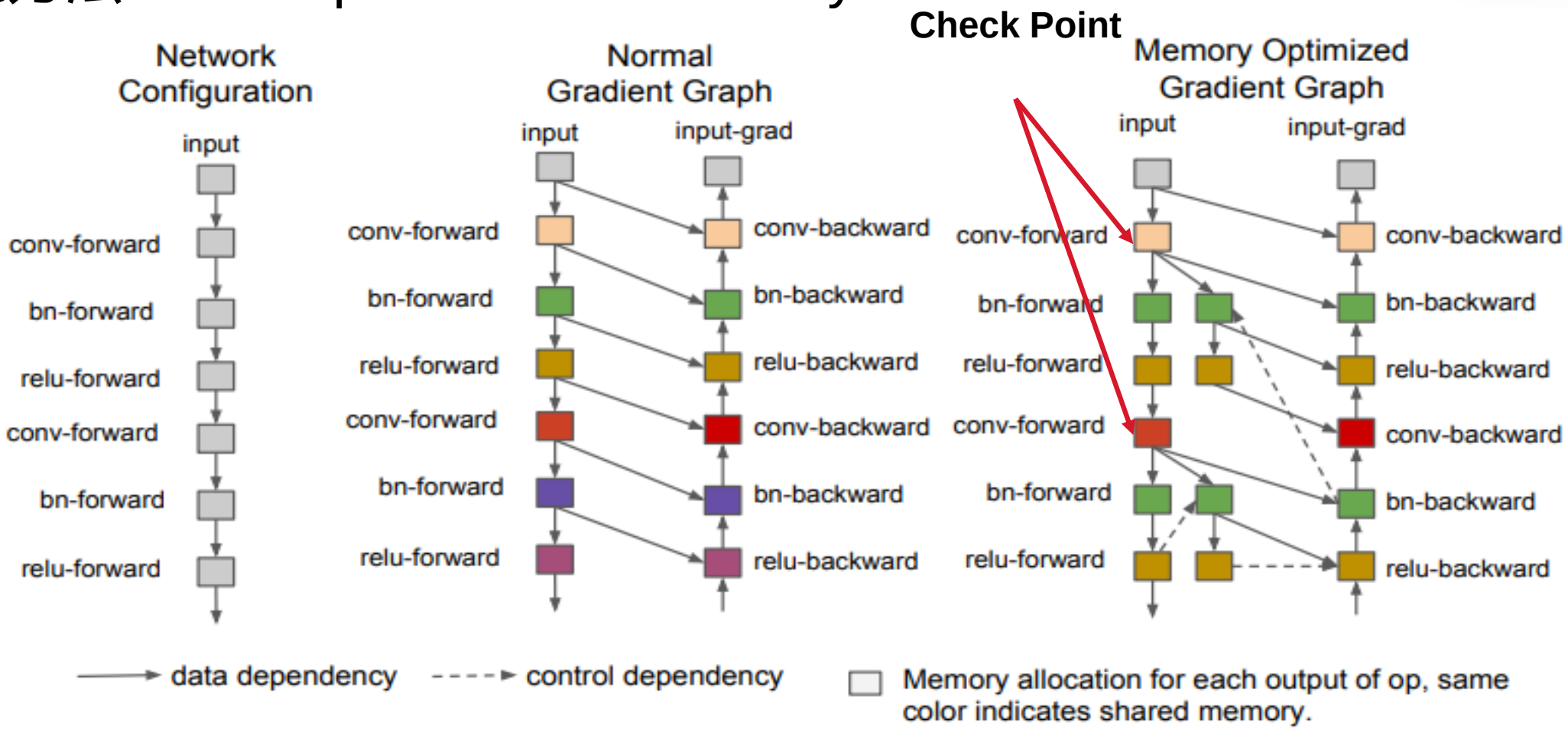




采用简单计数器，标记每一块内存之后需要被用到的次数

计算以外：内存优化

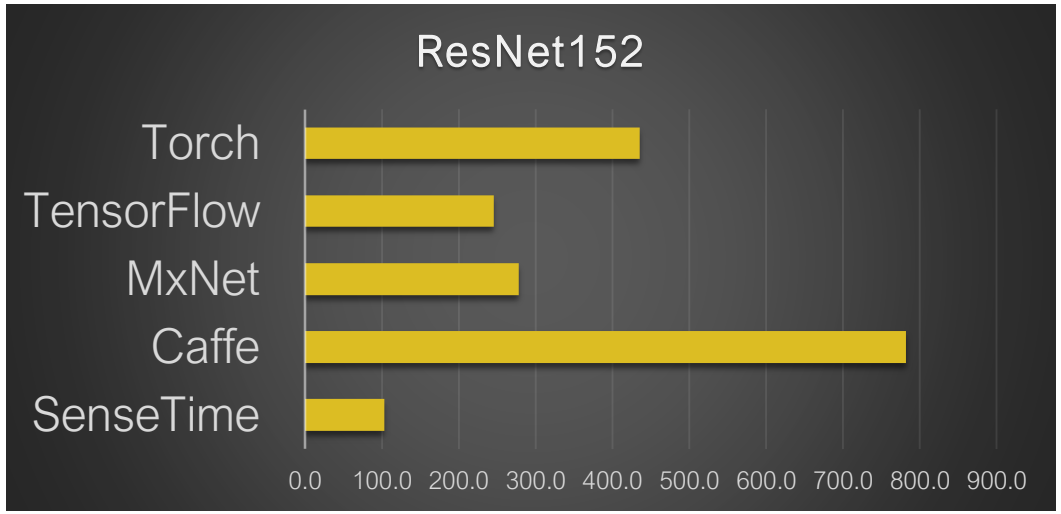
- 优化方法3：Computation Vs. Memory

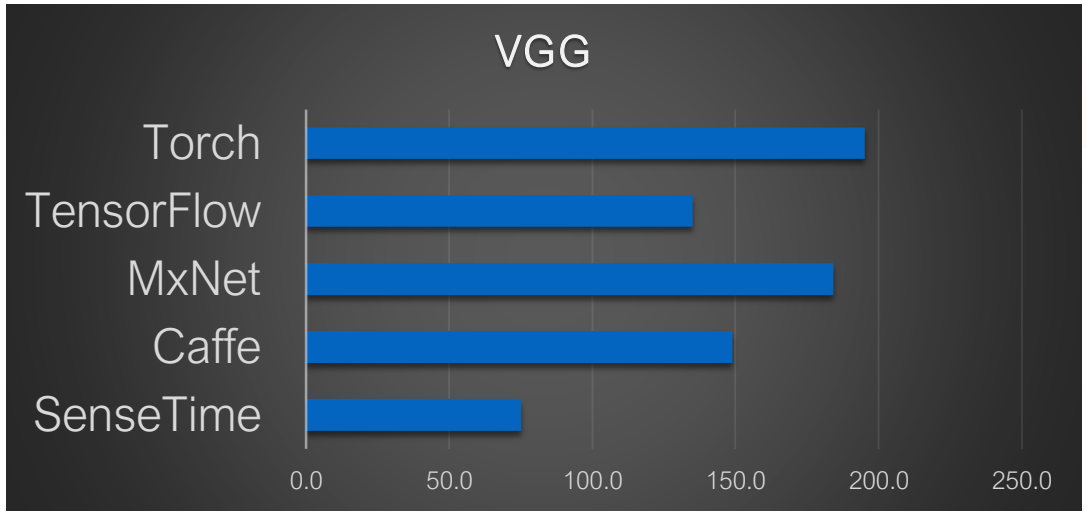


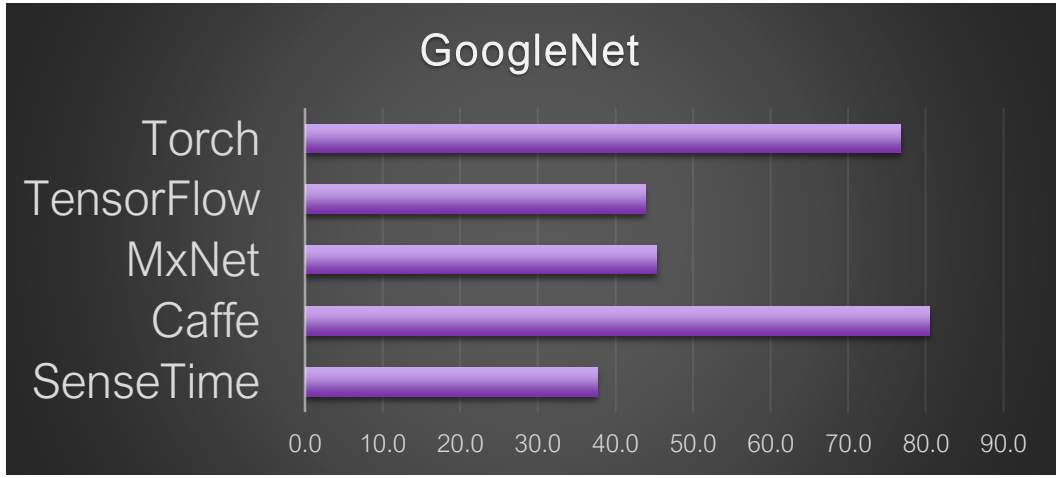
通过重复计算一次Forward，可以获得次线性的内存消耗

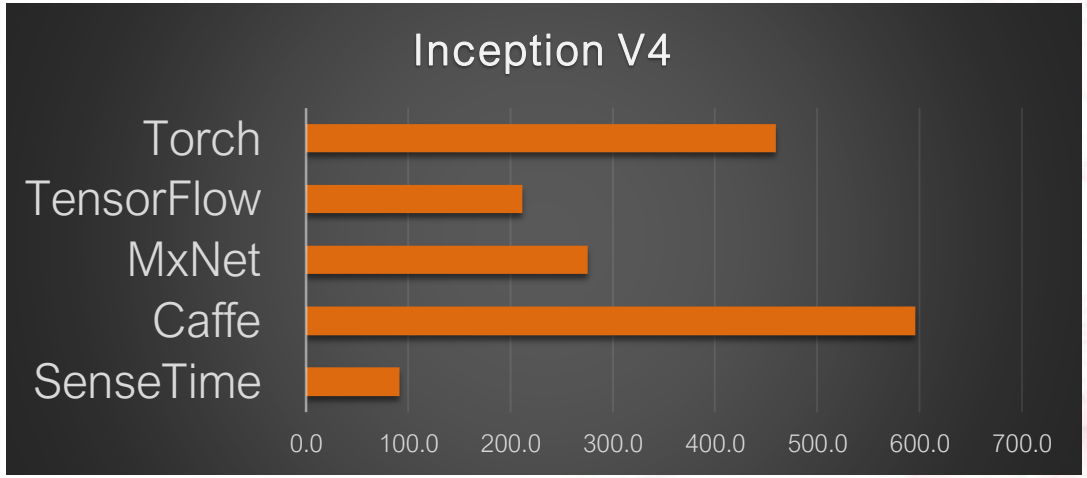
参考：https://arxiv.org/pdf/1604.06174.pdf

计算以外：内存优化



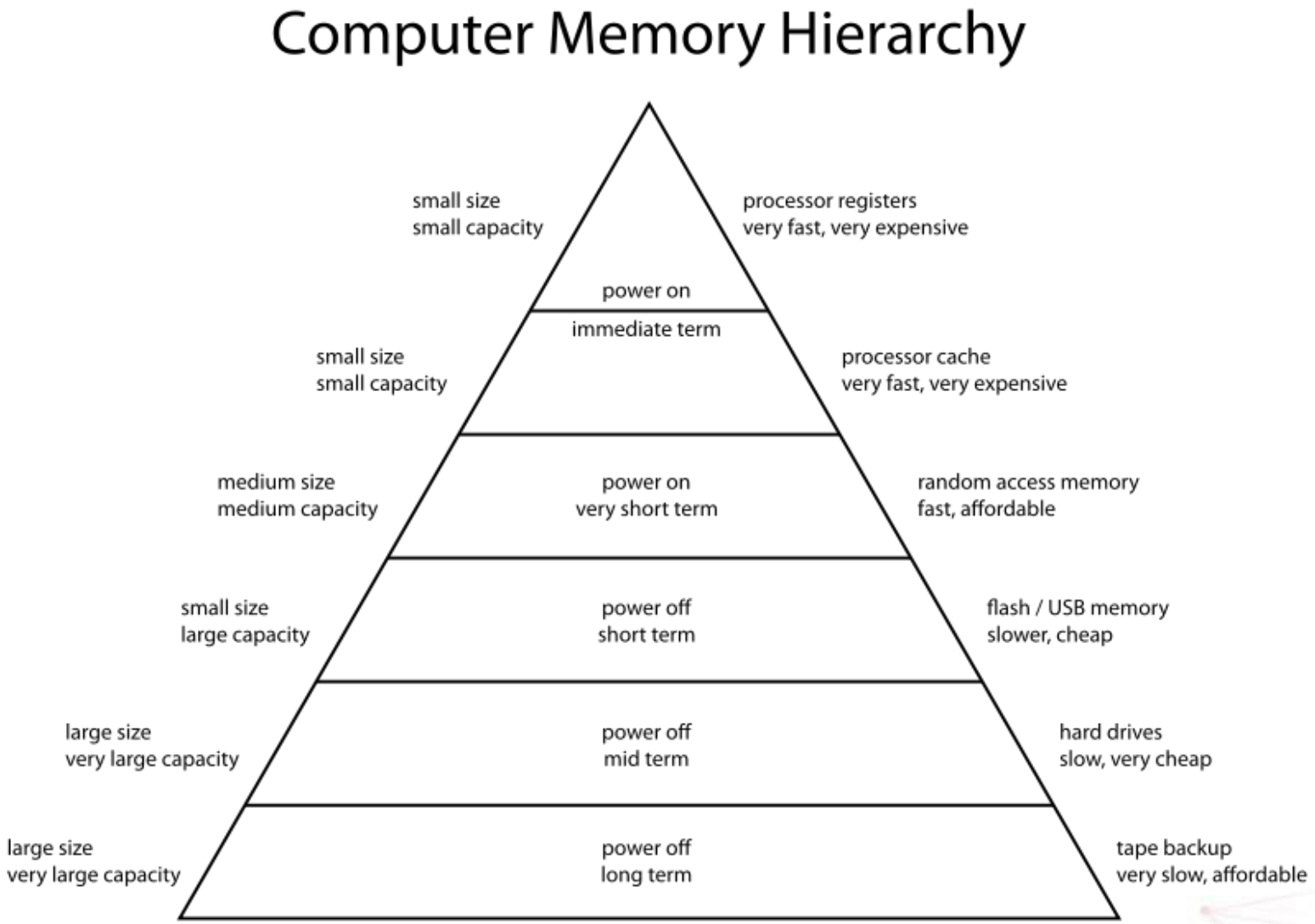






Memory efficiency, lower is better（MxNet不确认是否是最优版本）

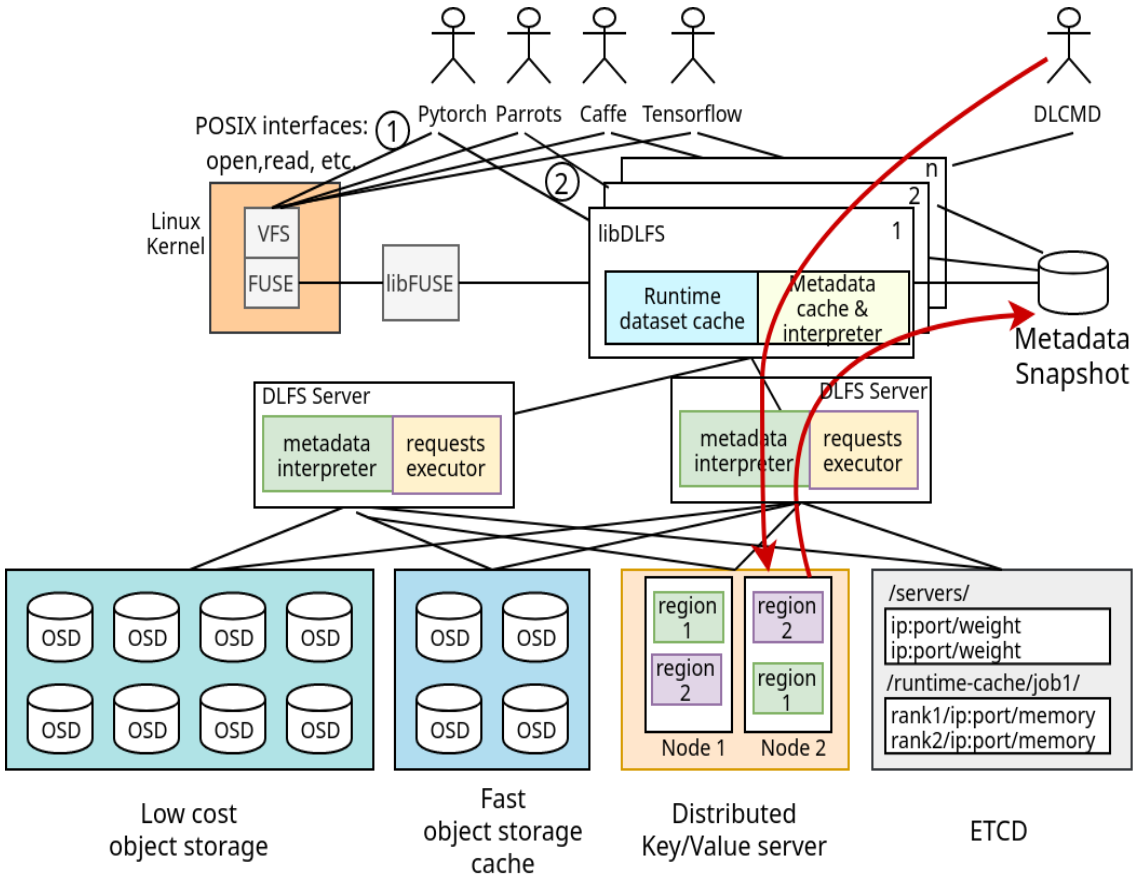
计算以外：访存优化



参考：https://en.wikipedia.org/wiki/Memory_hierarchy

计算以外：访存优化

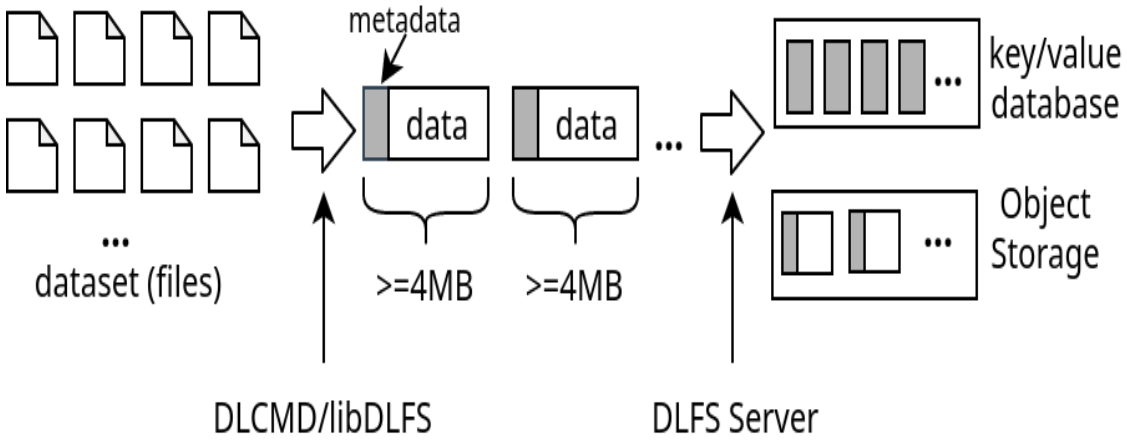
System Overview



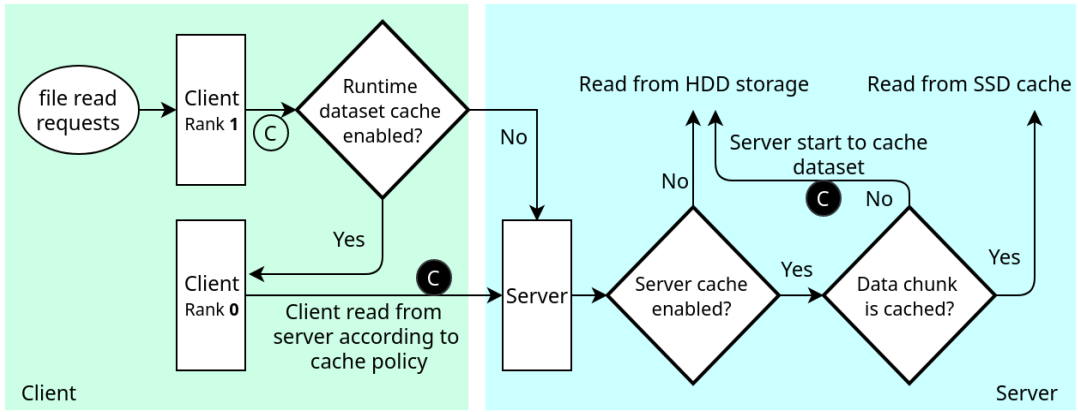
Overview of DLFS System

- Use multiple data storage backends to reduce hardware cost
- Server is stateless and has no dependencies between each other, free to scale up
- Metadata is stored in distributed key/value server. Support two types of key/value server (NVME-based Tikv and memory-based Redis).
- Users have two ways to access the data in our system: proprietary APIs and POSIX (via FUSE - Filesystem in Userspace)
- Client has runtime dataset caching system to cache job dataset among client instances
- Metadata can be materialized (snapshot) locally to bypass the key/value server

Write/Read dataflow from DLFS client to DLFS server



Write flow of DLFS

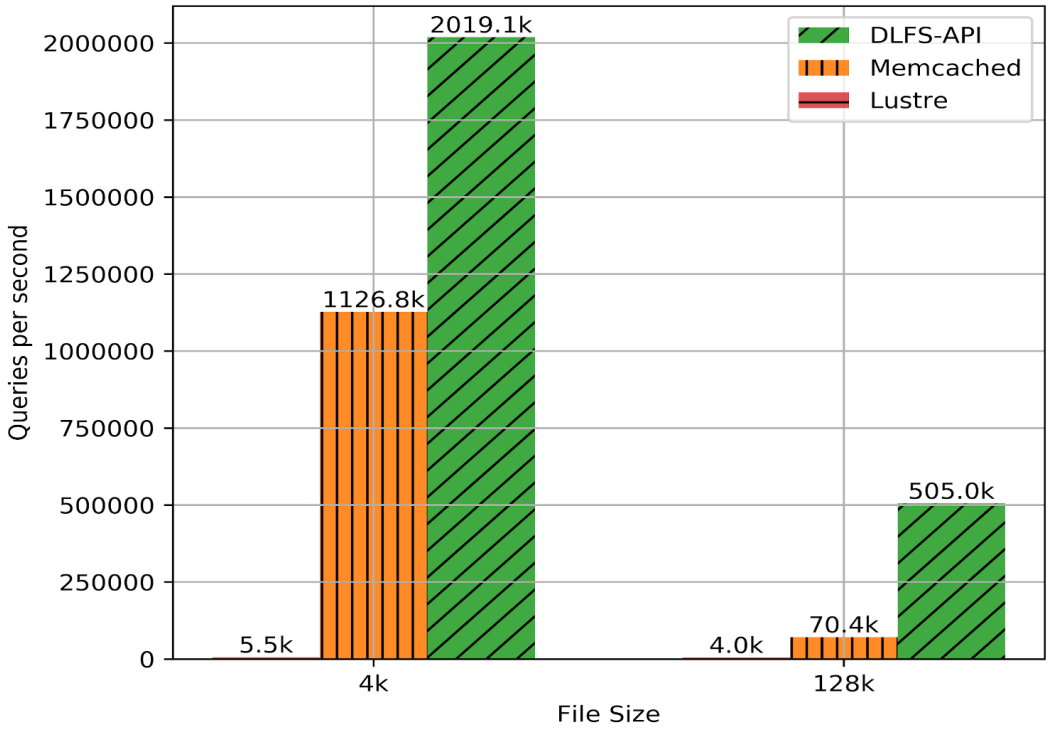


Read flow of DLFS

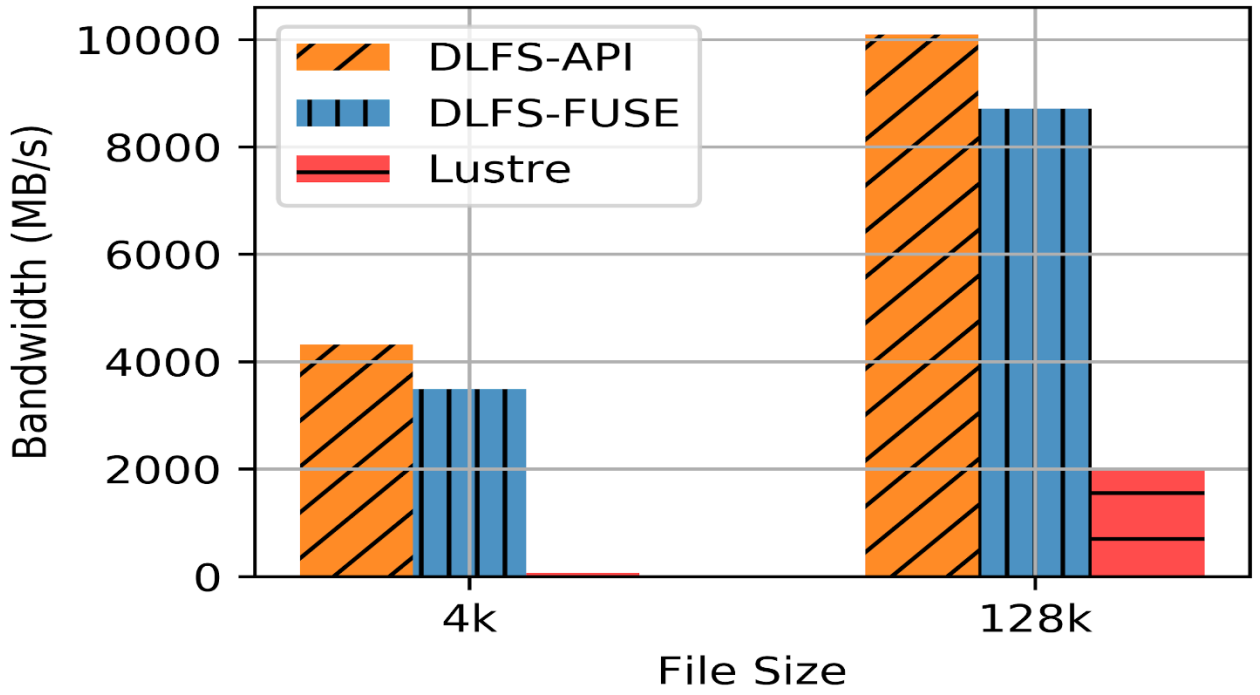
- File writes in batches
- File aggregates to large data chunks with metadata include in the head
- Runtime dataset cache works on metadata snapshots
- There are multiple client instances on a machine. To reduce the number of network connections, we only select one client instance to perform the runtime dataset cache
- The client cache used for block-based shuffle read method and the FUSE interface

高性能存储|Evaluation

Points: DLFS writes in batches, merges small files into large data chunks
DLFS API Vs. Lustre vs. Memcached



QPS with 64 threads (on 4 nodes)



Bandwidth on 4 nodes (higher is better)

CONTENTS

目录

1 AI复兴：深度学习

2 计算优化：GPU

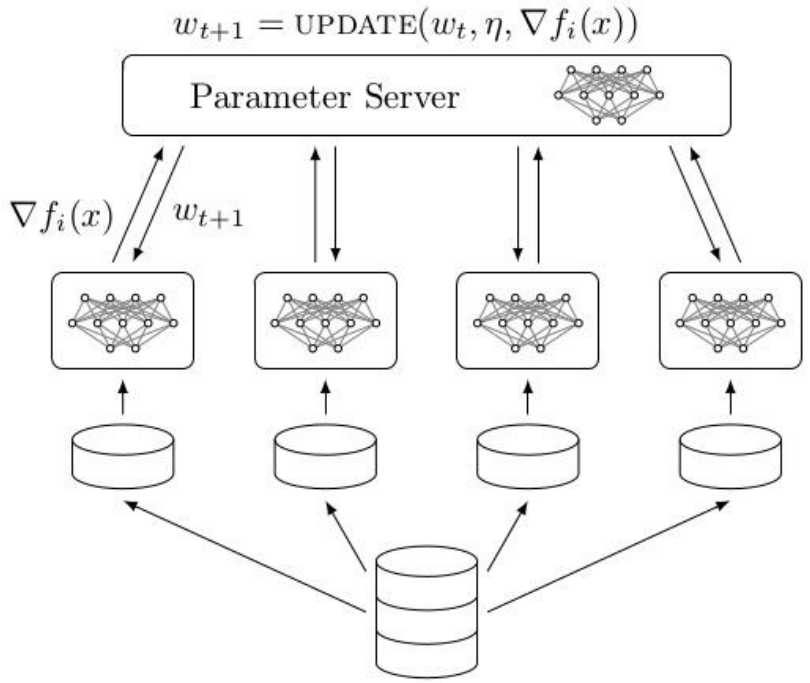
3 计算以外：内存&访存

4 携手共进：通信优化

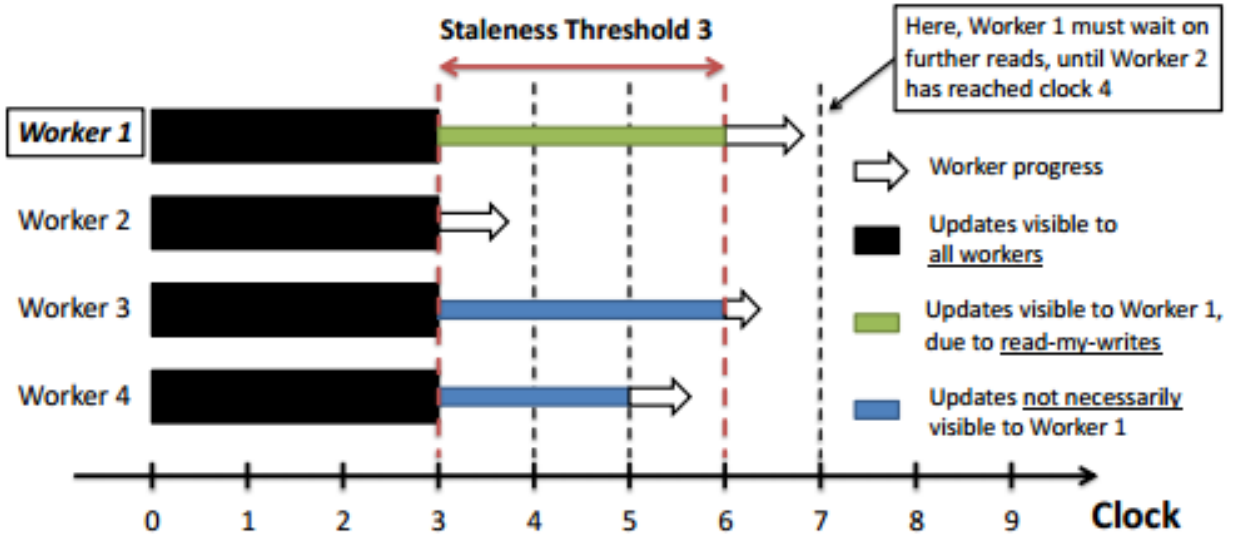
5 无所不能：云上AI

6 轻松一刻：应用案例

模型同步：参数服务器



参数服务器



异步训练

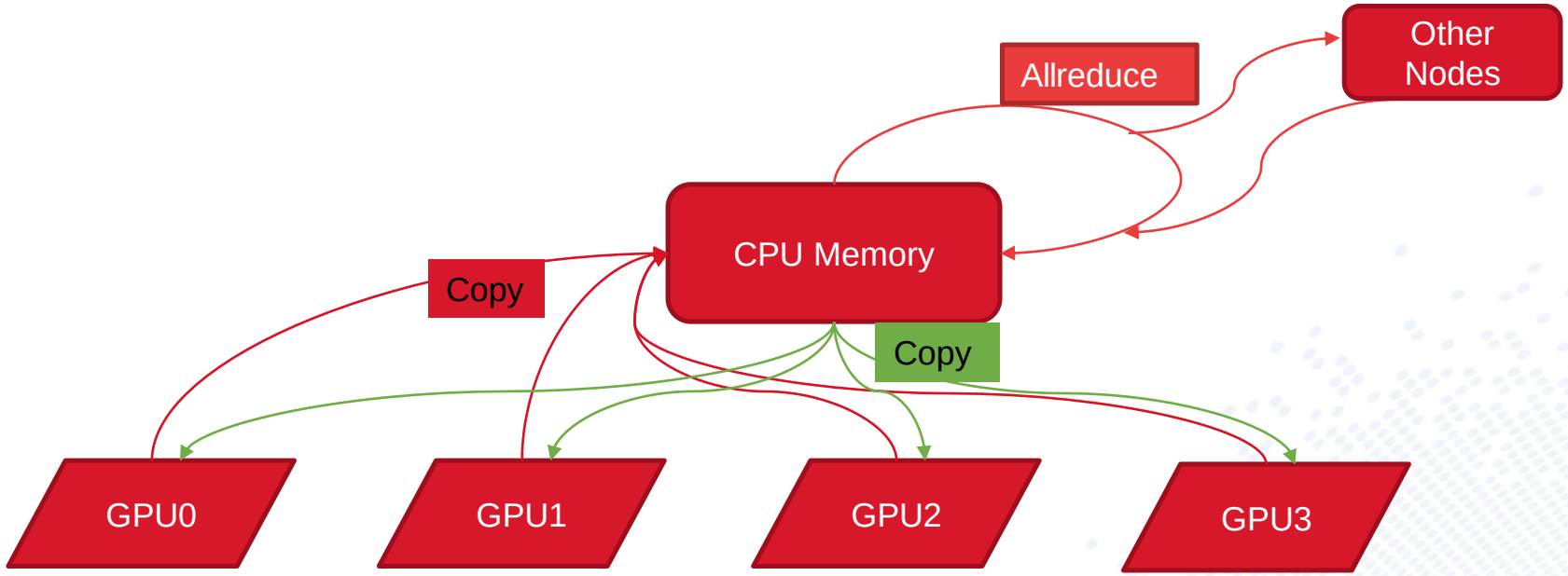
参数服务器的好处：

1.服务器分为Server和Work，二者之间采用Put和Get操作通信，无需显示同步

2. 模型存放在Server上，不受单机内存限制，方便做模型并行，支持大模型

模型同步：Allreduce

- Support Multi-GPUs and Multi-Nodes
- Three procedures: Copy, Allreduce, Copy

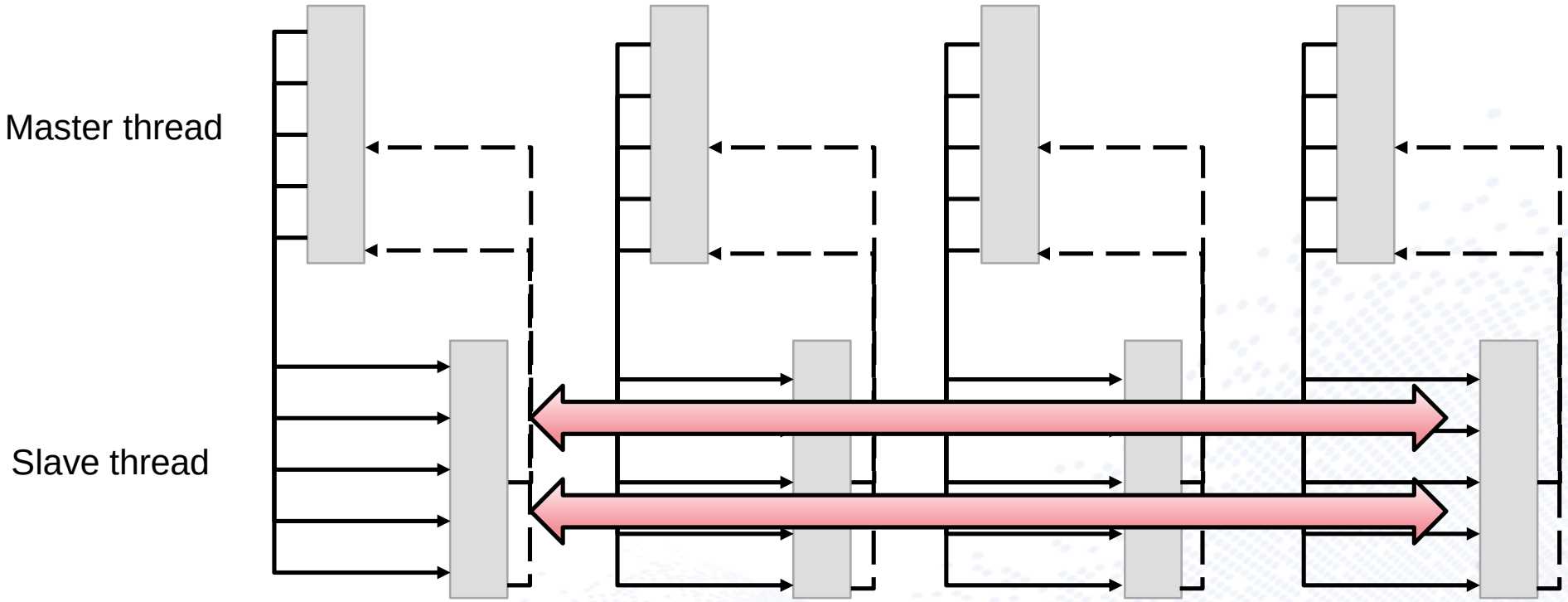


Copy: PCI-E

MPI Allreduce: CPU Allreduce

软件优化-通信优化

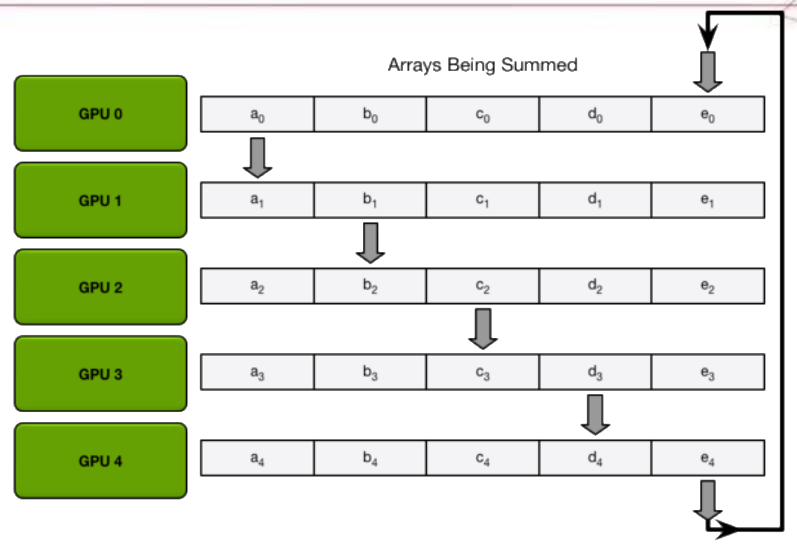
- **Task queue**

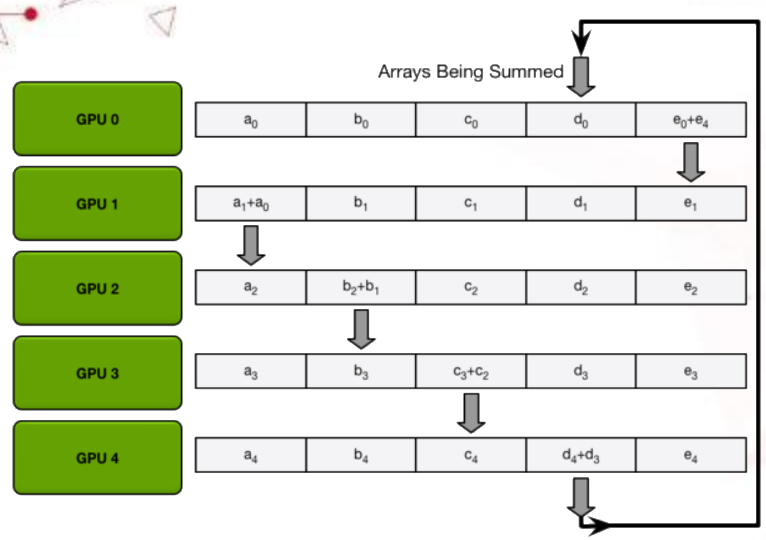


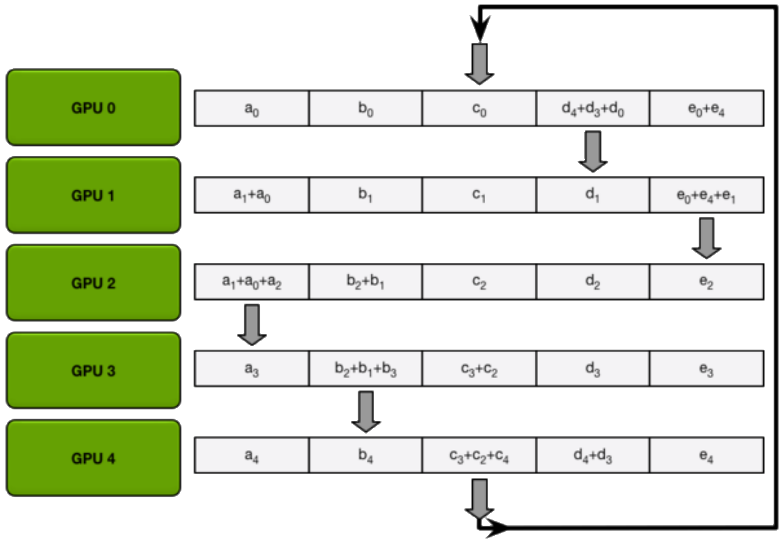
Ring-based Allreduce

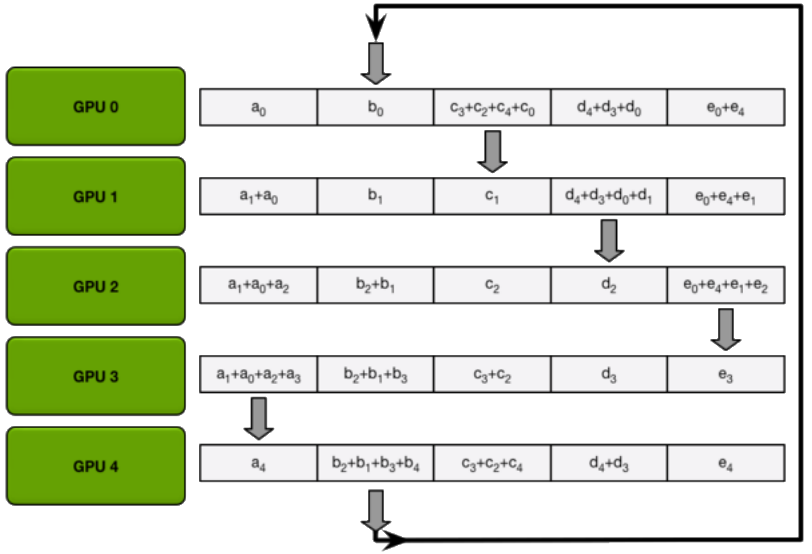
Ring Allreduce流程

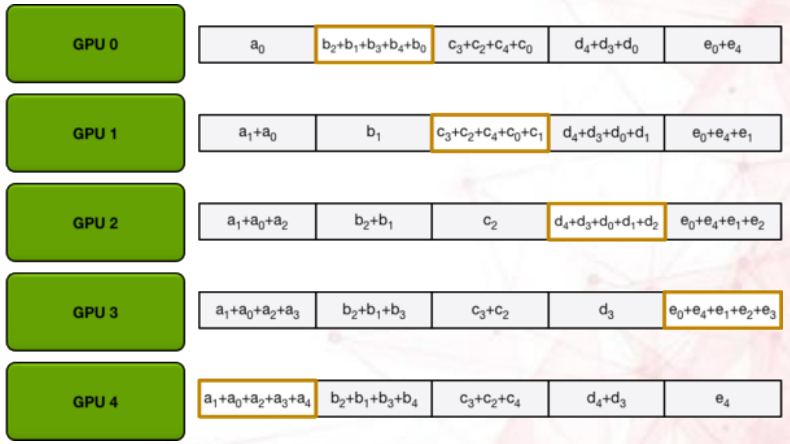
1. scatter-reduce
2. allgather







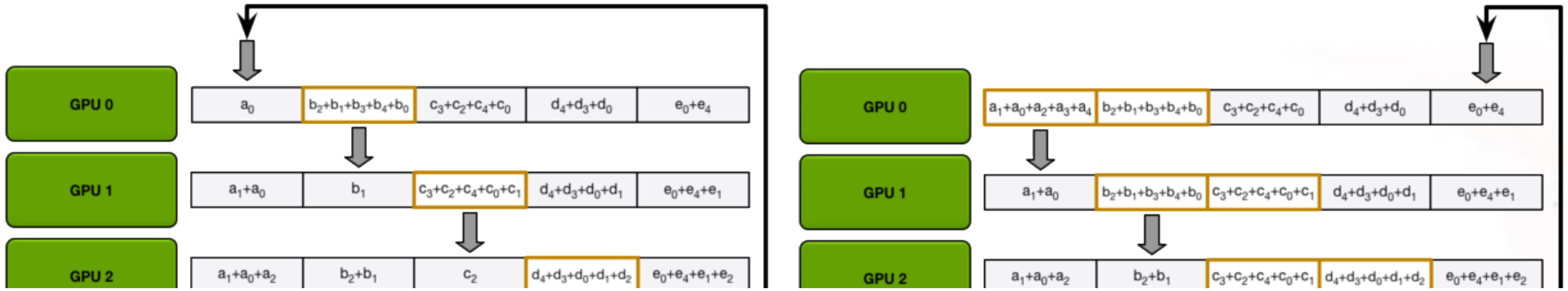




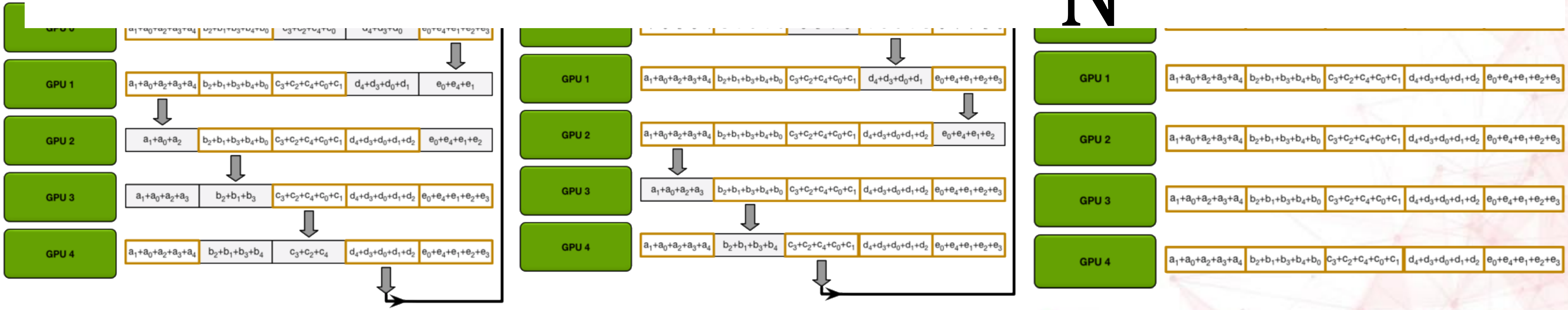
Ring-based Allreduce

Ring Allreduce流程

1. scatter-reduce
2. allgather



$$Data\ transferred = 2\frac{\mathrm{M}}{\mathrm{N}} * (\mathrm{N} - 1)$$



Scale out distributed DL on a cluster with >512 GPUs

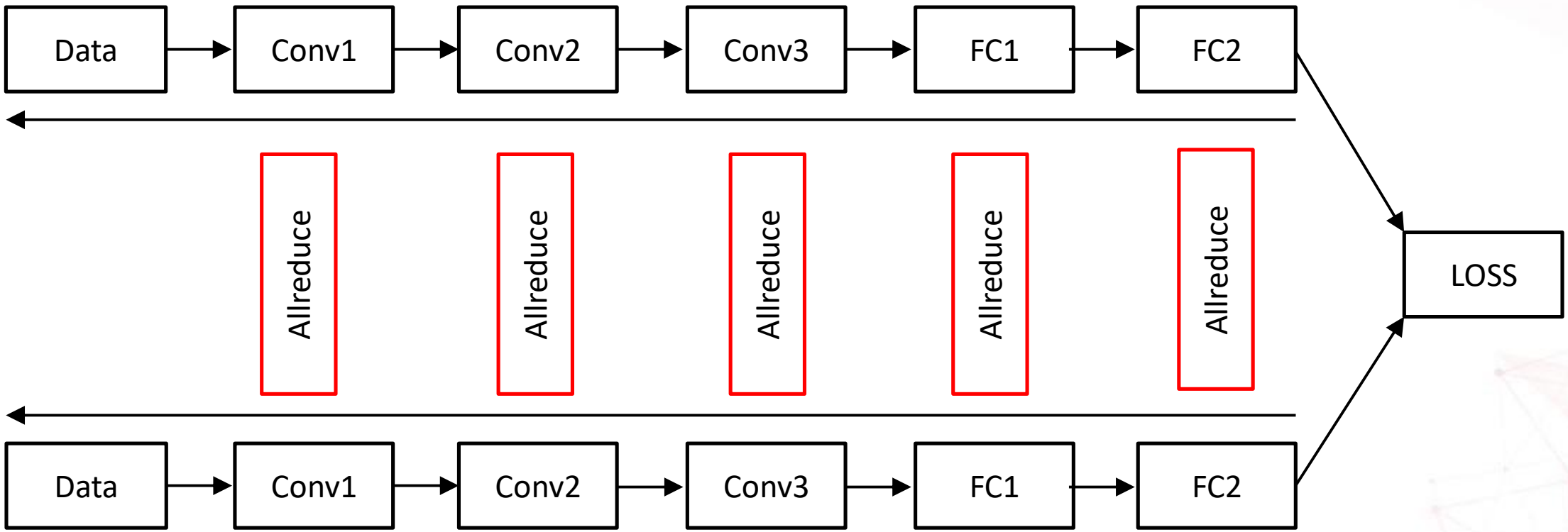
Target: Train AlexNet on ImageNet with 512 GPUs using 64K batch size

Problem: Guarantee the model quality with large batch size

Solutions:

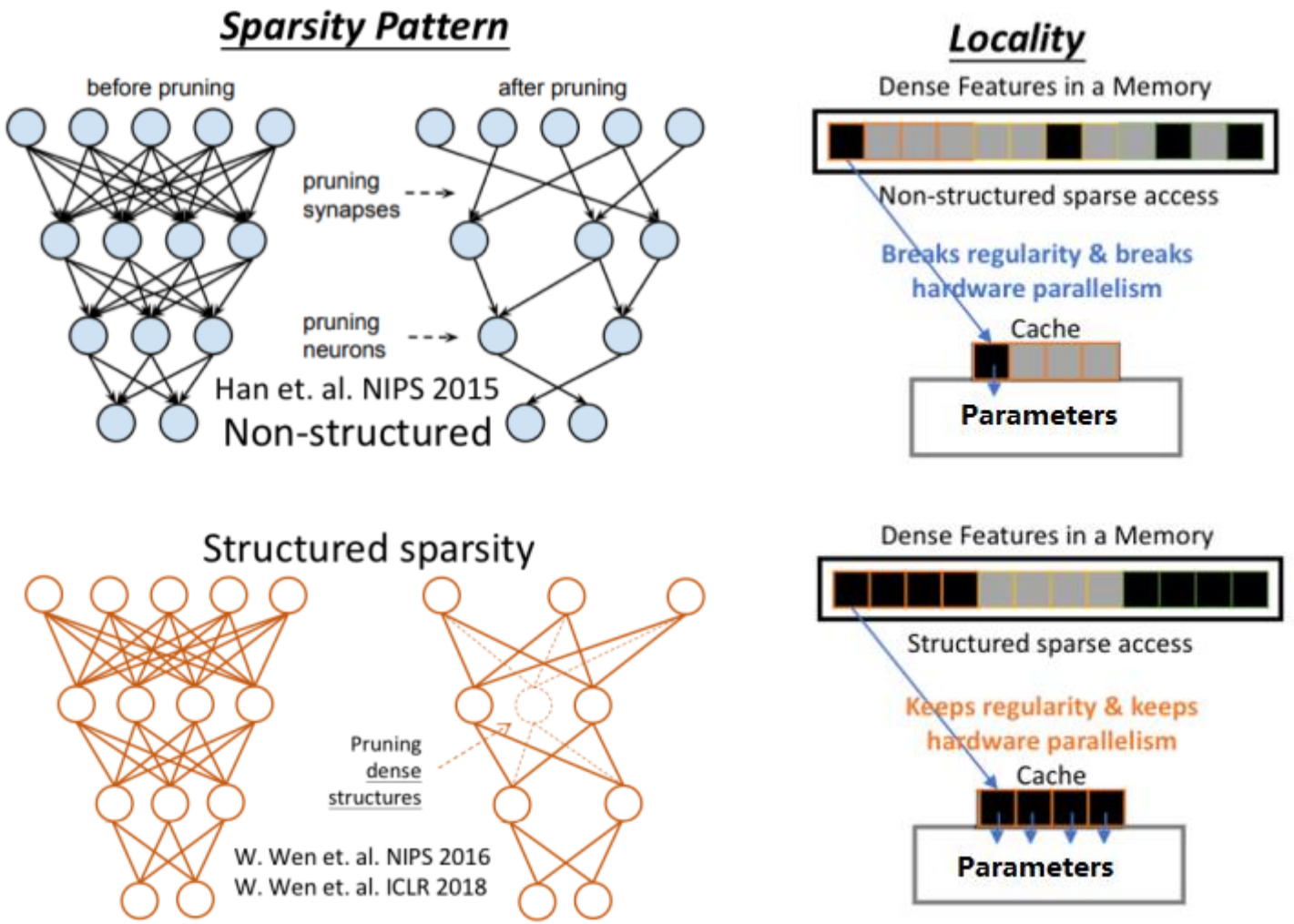
- Linearly scaling up learning rate (LR)
- Warmup LR
- Layer-wise Adaptive Rate Scaling (LARS)
- AlexNet: 8K (2017) -> 64K (2018), > 58.6 accuracy

Improve network performance with block-based communication design

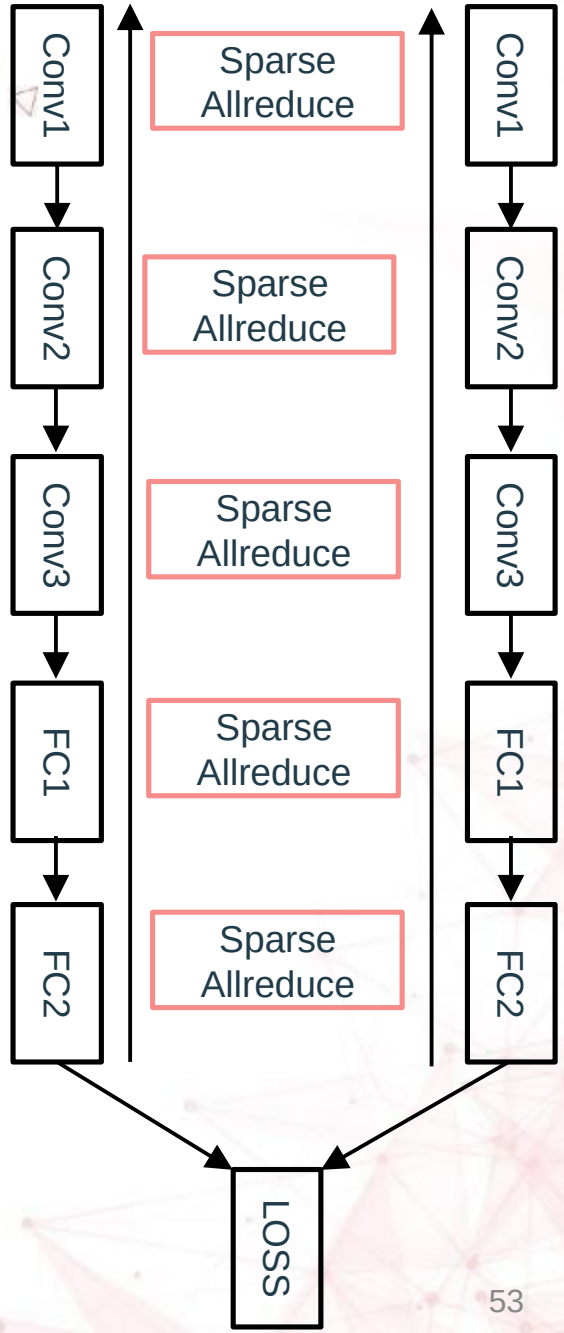


- Layer-based allreduce communication operation

Sparse block-based communication design



结构化Sparse示意图



结构化Sparse后通信的数据

数据类型：单精度和半精度

32 bit = float，single precision

1bit 8bit 23bit

可表示的最大值

3.402823 X 10^38

非规格化最小值

2^-149

16 bit = half，half precision

1bit 5bit 10bit

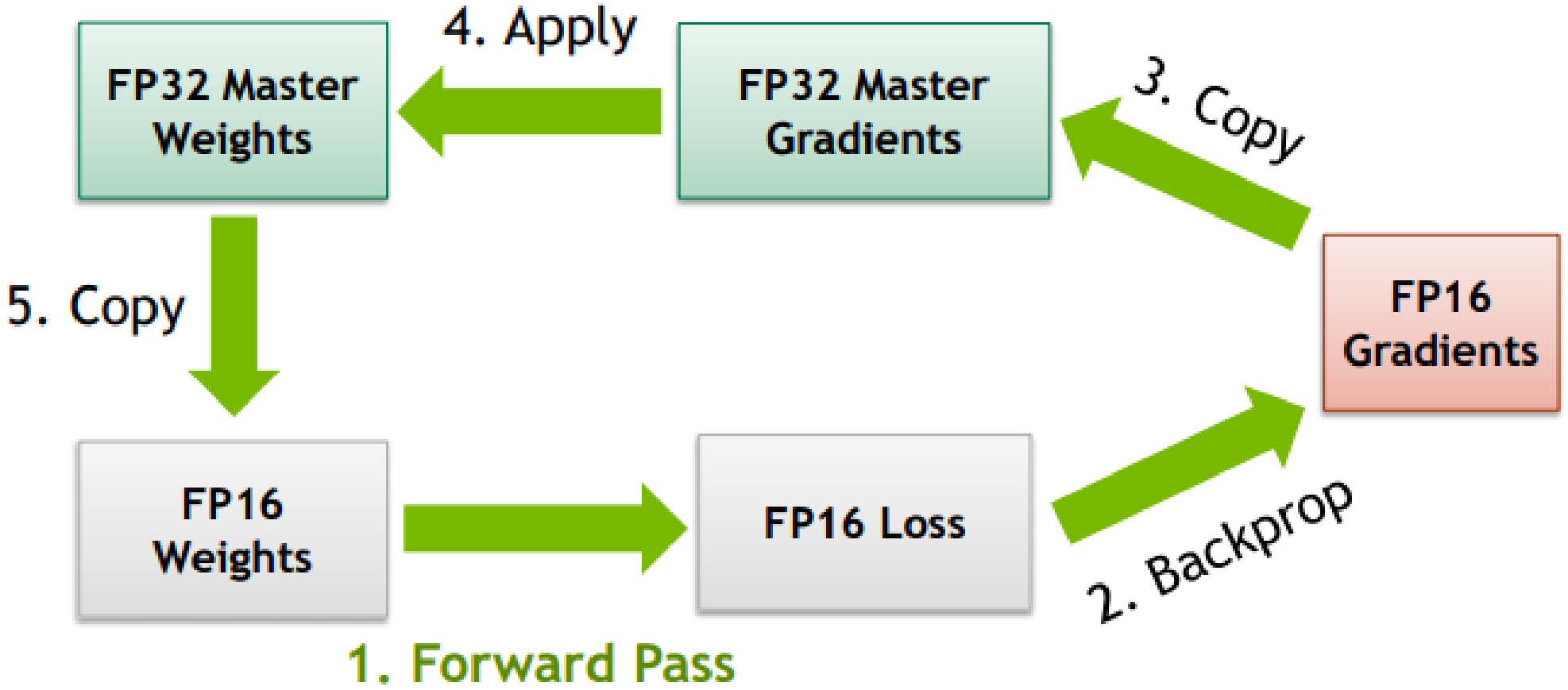
可表示的最大值

65504

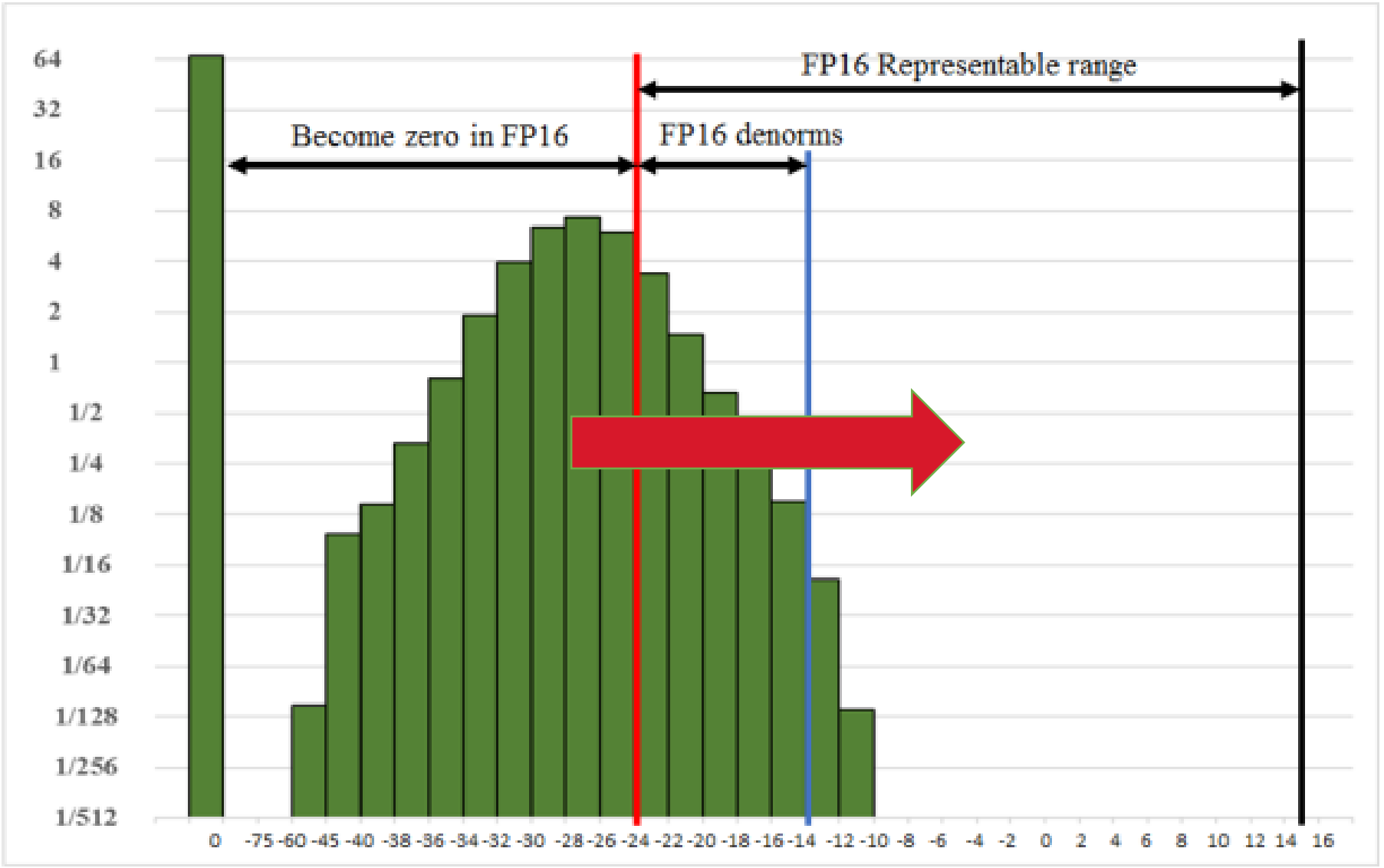
非规格化最小值

2^-24

混合精度解决方案: FP32 MASTER WEIGHTS

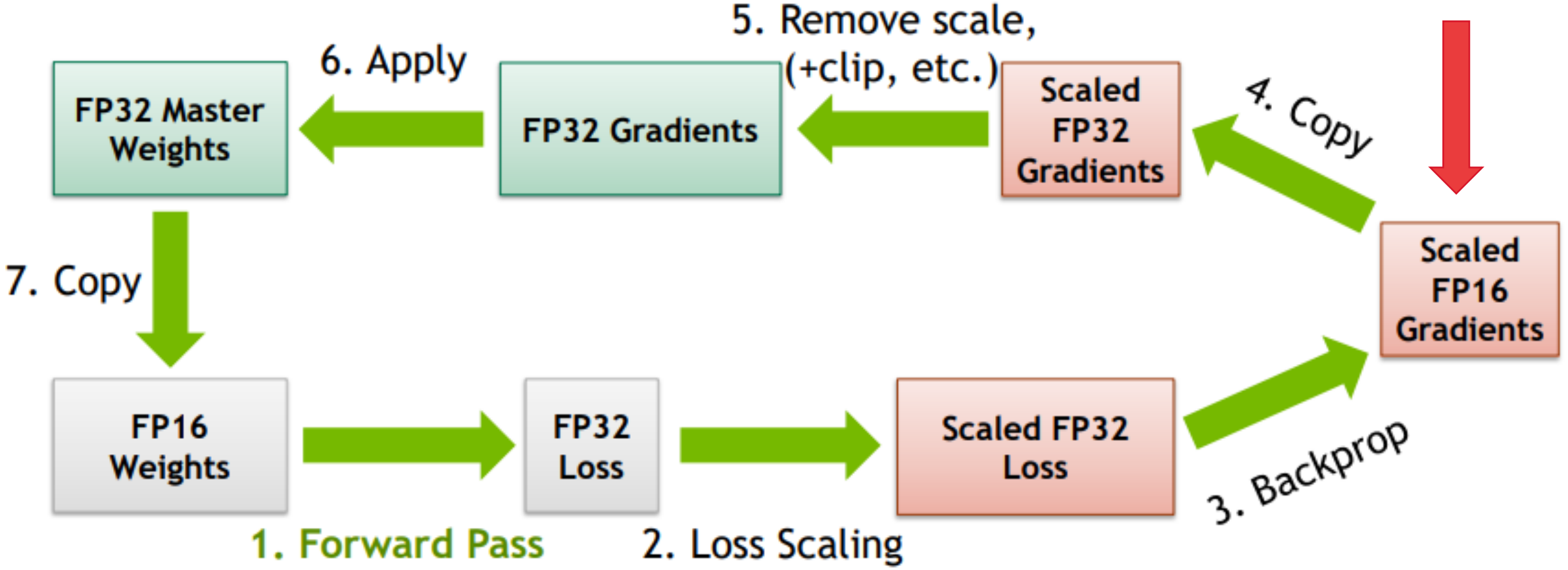


混合精度的问题：导数下溢

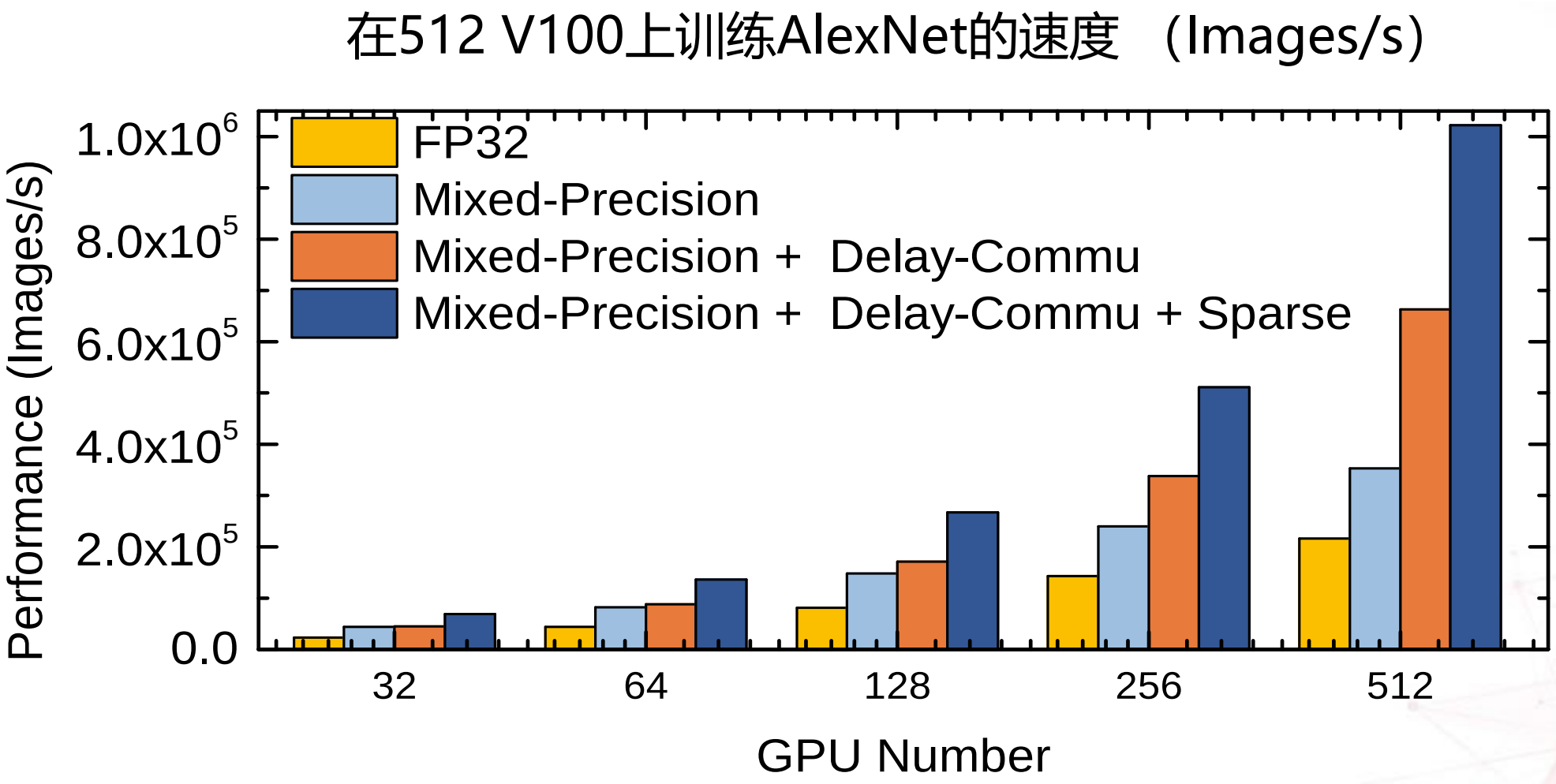


混合精度解决方案： LOSS SCALING

需要通信的部分

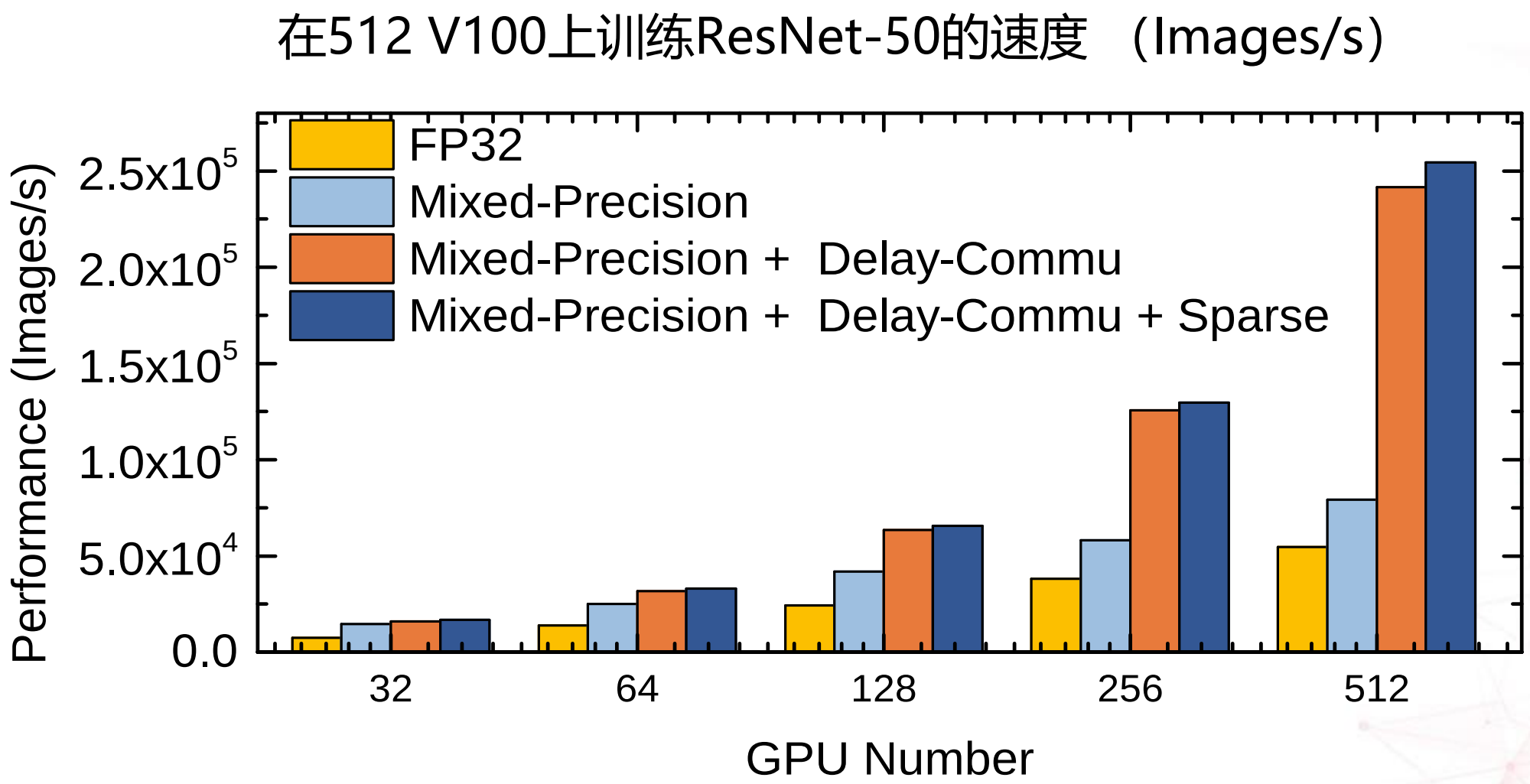


ImageNet training with AlexNet and Resnet-50



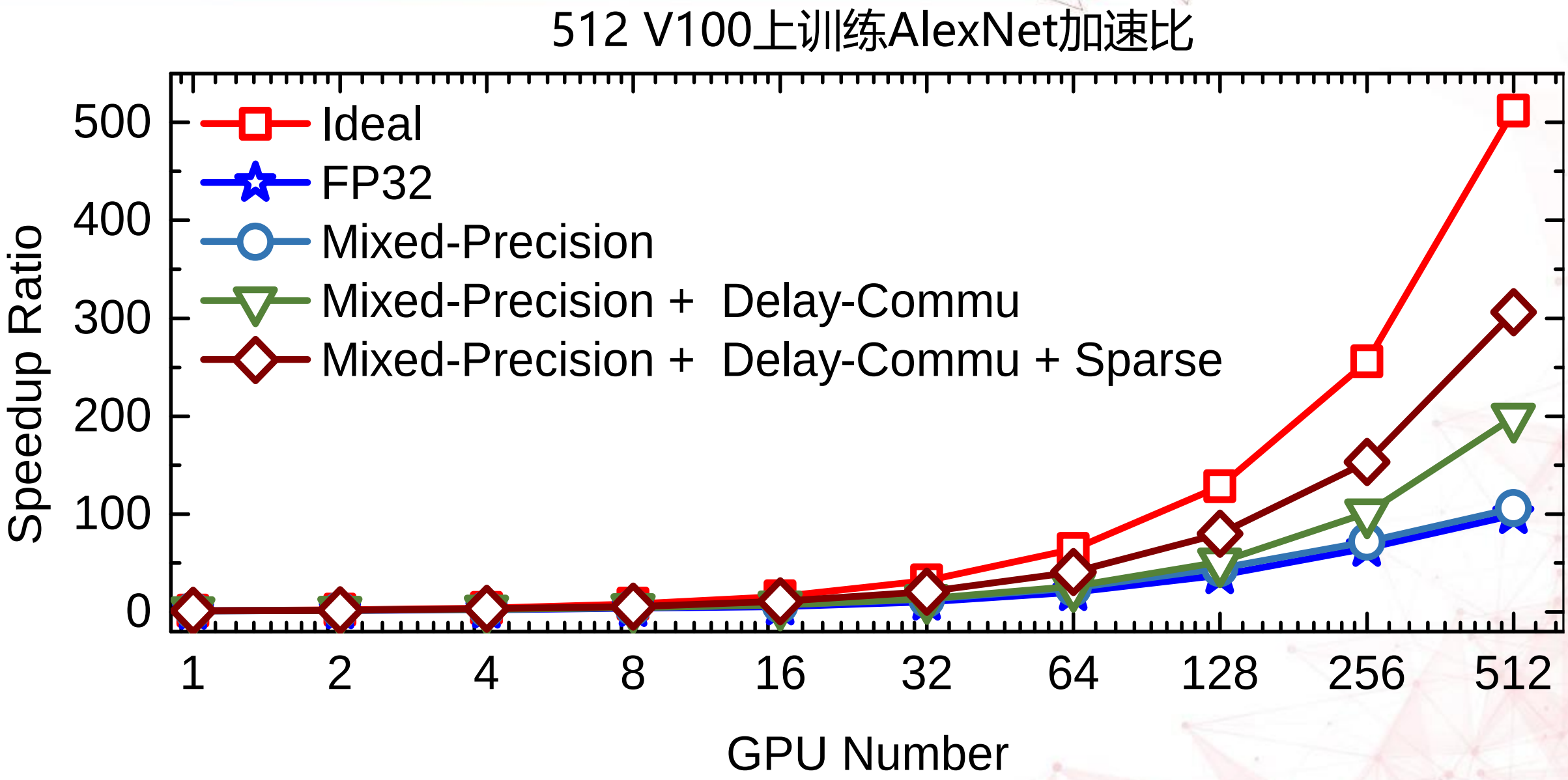
150秒内完成ImageNet/AlexNet 92 Epoch 训练，精度达到58.3%

ImageNet training with AlexNet and Resnet-50

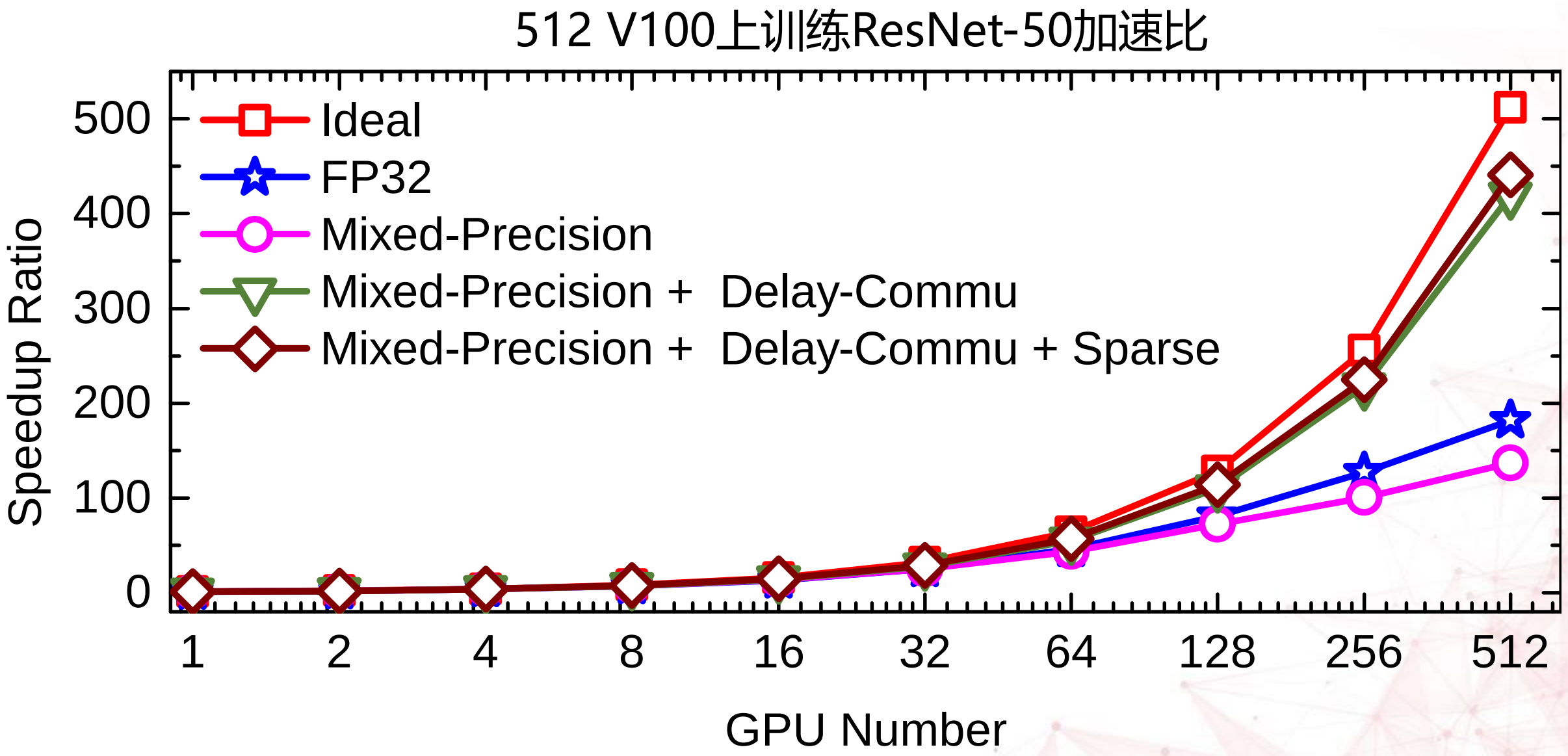


480秒内完成ImageNet/ResNet 90 Epoch 训练，精度达到75%

ImageNet training with AlexNet and Resnet-50



ImageNet training with AlexNet and Resnet-50



CONTENTS

目录

1 AI复兴：深度学习

2 计算优化：GPU

3 计算以外：内存&访存

4 携手共进：通信优化

5 无所不能：云上AI

6 轻松一刻：应用案例

Parrots综合算力平台

Parrots定位

完全商汤自研，提供综合算力平台，支撑AI研发和行业赋能

应用场景

安防 | 互娱 | 手机 | 自动驾驶 | 教育 | 医疗图像 | ...

算法/工具链

检测 | 分类 | 跟踪 | 关键点 | 分割 | 定点化 | ....

算力

综合算力平台

GPU/CPU

NVIDIA intel QUALCOMM

Storage

DELL SUPERMICRO 中科曙光 Sugon HUAWEI

Interconnect

Mellanox TECHNOLOGIES

新一代Parrots

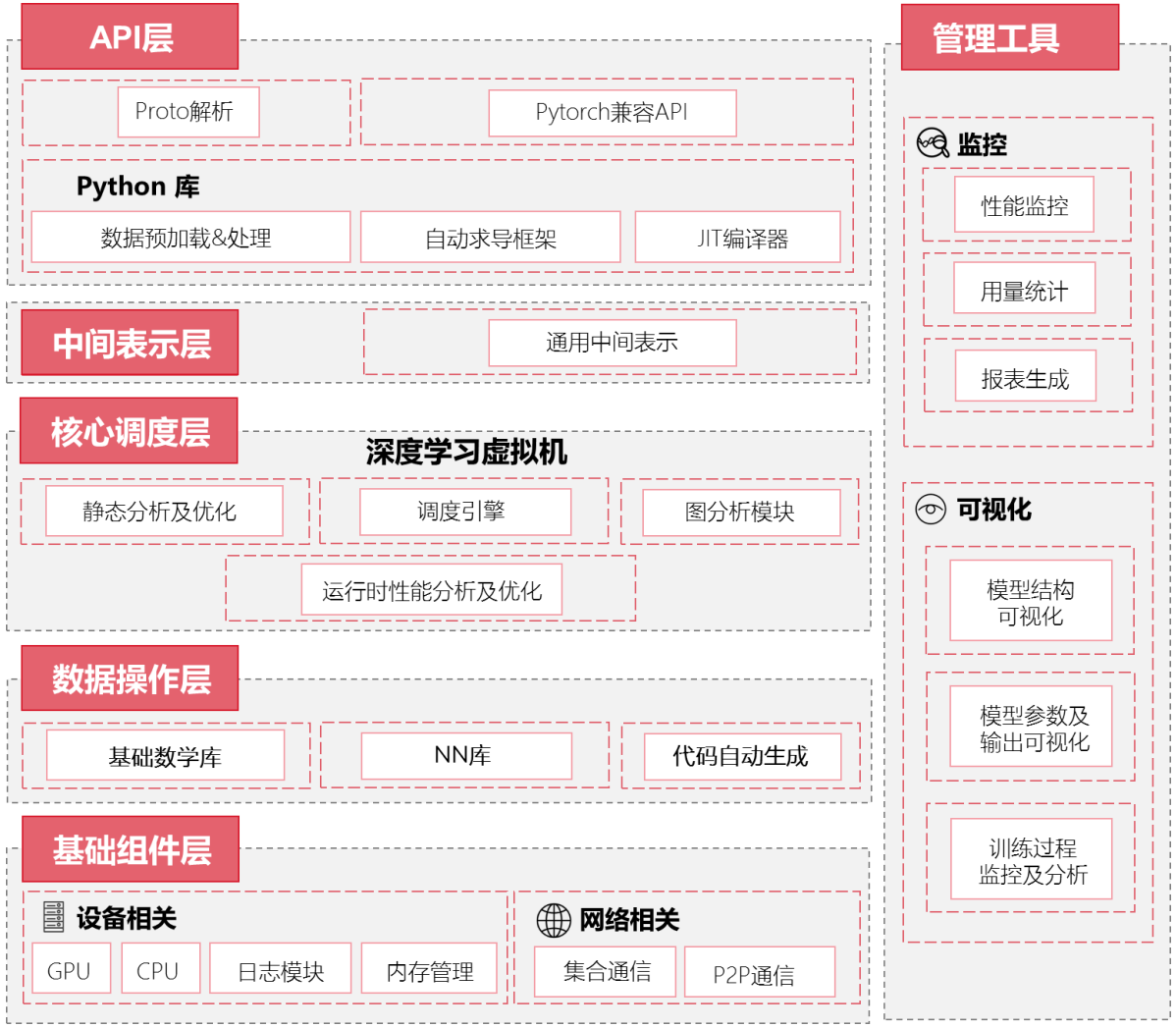
主要竞争对手趋势分析

	Facebook	Google	Baidu
动态图支持	● PyTorch 用作前端语言	● 引入Eager模式	● 自定义Fluid语法，使用python编写Fluid Program
计算性能	● Caffe2 用作底层执行	● 针对常用case优化Eager性能	● 使用c++内核执行Fluid Program
模型部署	● Scripts: 将python代码编译成中间表示	● 使用AutoGraph将python动态逻辑编译成静态图代码	● Fluid Program做为中间表示，支持直接部署
特色/生态	● Glow: 跨平台底层引擎	● Lite：端上部署套件 ● TFX：机器学习pipeline	● Cloud：支持浏览器开发应用 ● EDL：虚拟化部署、调度训练

新一代Parrots

设计目标

新一代Parrots



系统架构

主要特性

兼容Pytorch

一键切换直接source

线性拓展

128卡加速比超过90%

关键技术

动态图并行

支持灵活定义模型结构

JIT编译

自动优化训练脚本

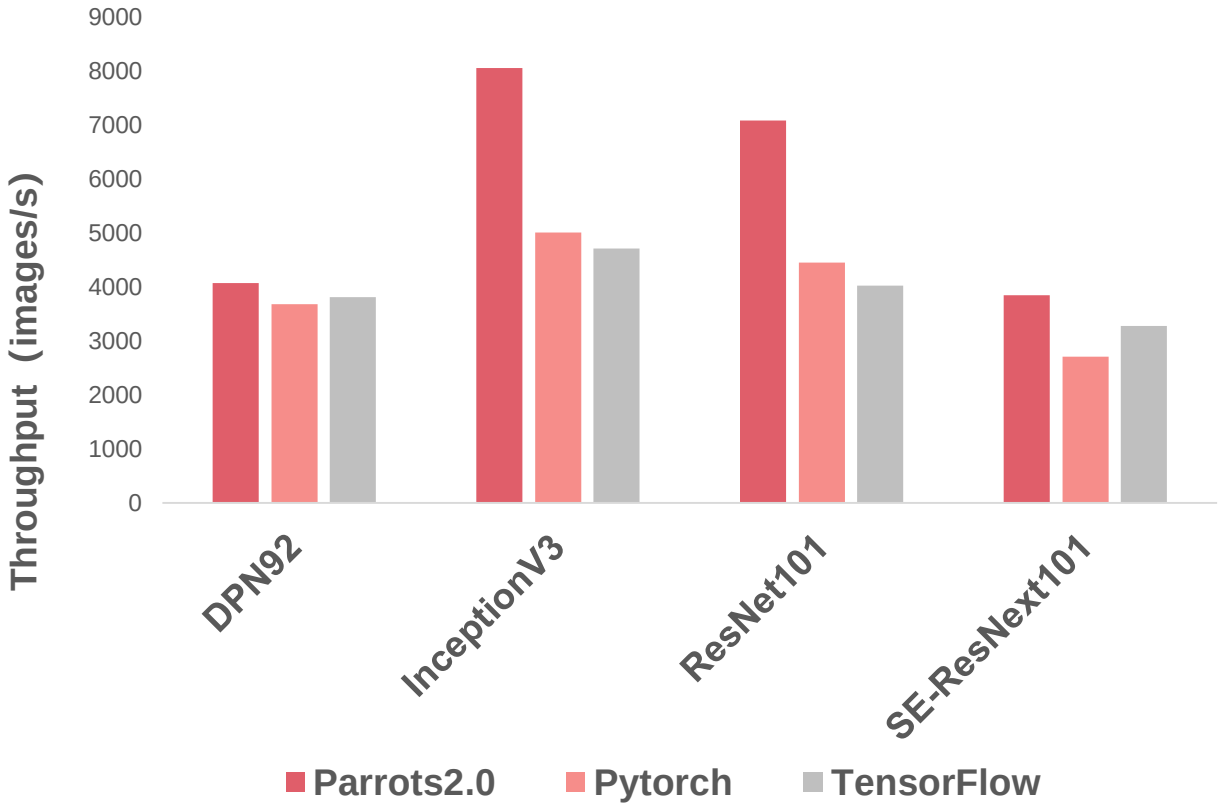
非阻塞执行引擎

运算速度不受限于python

多stream并行

最大限度利用显卡算力

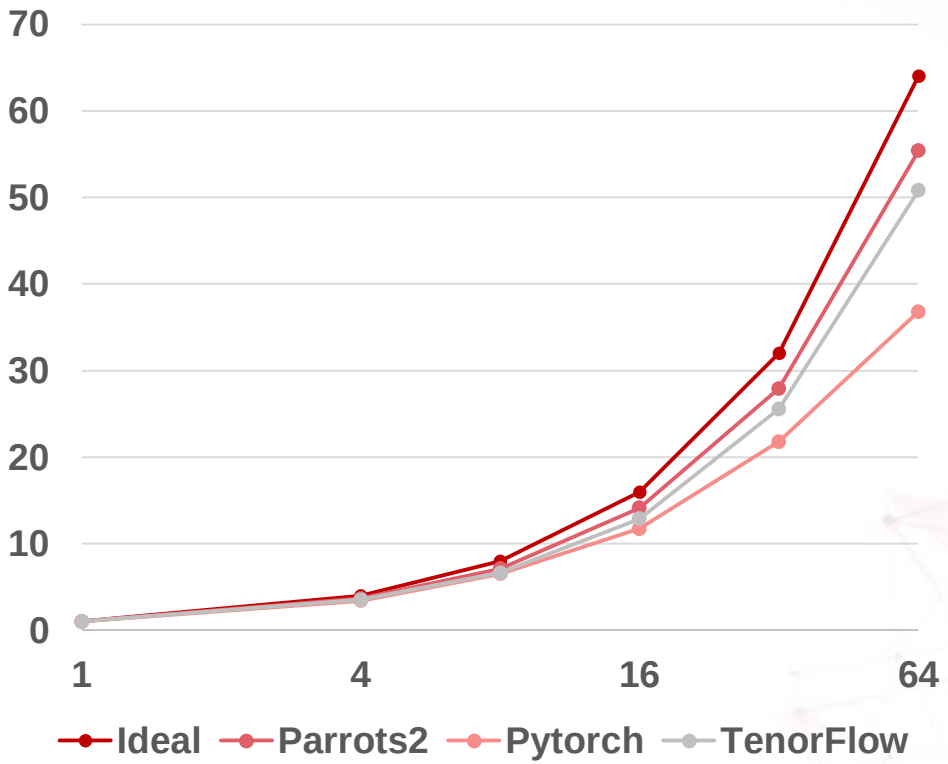
新一代Parrots



Training throughput on 64GPUs (images/s)

CPU: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.40GHz

GPU: NVIDIA Titan XP



Speedup on InceptionResNetV2 (Batchsize = 32)

CPU: Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz

GPU: NVIDIA GeForce GTX 1080 Ti

AI云原型系统概览

应用场景域

安防 | 互娱 | 手机 | 自动驾驶 | 教育 | 医疗图像 | ...

中间服务域

- 工具链：检测 | 分类 | 跟踪 | ...
- 训练框架：前端 | 引擎 | IO库 | ...
- 数据链：接入 | 标注 | 管理 | ...

基础资源域

- 虚拟资源抽象
- 集群资源池基础软件
- 硬件设备

管理域

- 服务管理：管理 | 监控 | 数据安全
- 基础资源管理：管理 | 监控 | 运维

AI云原型系统概览


服务输出
虚拟集群：拥有独立的管理节点、计算节点；计算、管理节点故障隔离；
单个虚拟计算实例：支持GPU、CPU虚拟化输出
存储：虚拟实例对接对象存储、块存储，数据隔离；文件存储进行权限控制
虚拟抽象
虚拟化管理：OpenStack & Kubernetes
KVM & Docker
软件定义网络，RDMA隔离性保障
服务模块化
存储资源
高速对象存储
训练数据、小文件
SSD
文件存储
代码、工作环境
海量对象存储
原始数据、大文件
HDD
存储中间件
SenseAgent
MC
硬件基础
高性能计算硬件
Nvidia GPU
AI协处理器
AMD GPU
CPU
服务器设备
GPU服务器
CPU服务器
网络设备
万兆以太网
40Gb RoCE
100Gb IB
存储设备
磁盘阵列
SATA/PCIe SSD
JBOD
SAS HDD

CONTENTS

目录

1 AI复兴：深度学习

2 计算优化：GPU

3 计算以外：内存&访存

4 携手共进：通信优化

5 无所不能：云上AI

6 轻松一刻：应用案例

轻松一刻：应用案例

Image and Video Editing

1

De-hazing

- Restore pictures impacted by fog or haze to original appearance through industry leading image processing techniques

2

Super-Resolution

- Improve the resolution of photos without impacting quality, allowing clearer and stunning visual effects

3

Restoration (Fast De-noising)

- Achieve fast de-noising of images on mobile devices using leading algorithm for image de-noising

4

Dark Light Enhancement

- Improve overall brightness and definition of images or video being shot under dark light environment

5

Single Picture HDR

- Use one single picture to deliver impressive pro-level HDR effect shooting with mobile devices

6

De-focus Repair

- Leading image processing algorithm to deliver rapid repair and restoration of defocused pictures

7

Video Stabilization

- Get rid of the jitter of the video so that the user can control the magnitude of stabilization

8

Re-focus Ability

- Dual camera-based 3D visual technology to deliver the function of focusing after shooting & aperture after shooting

9

Filtering

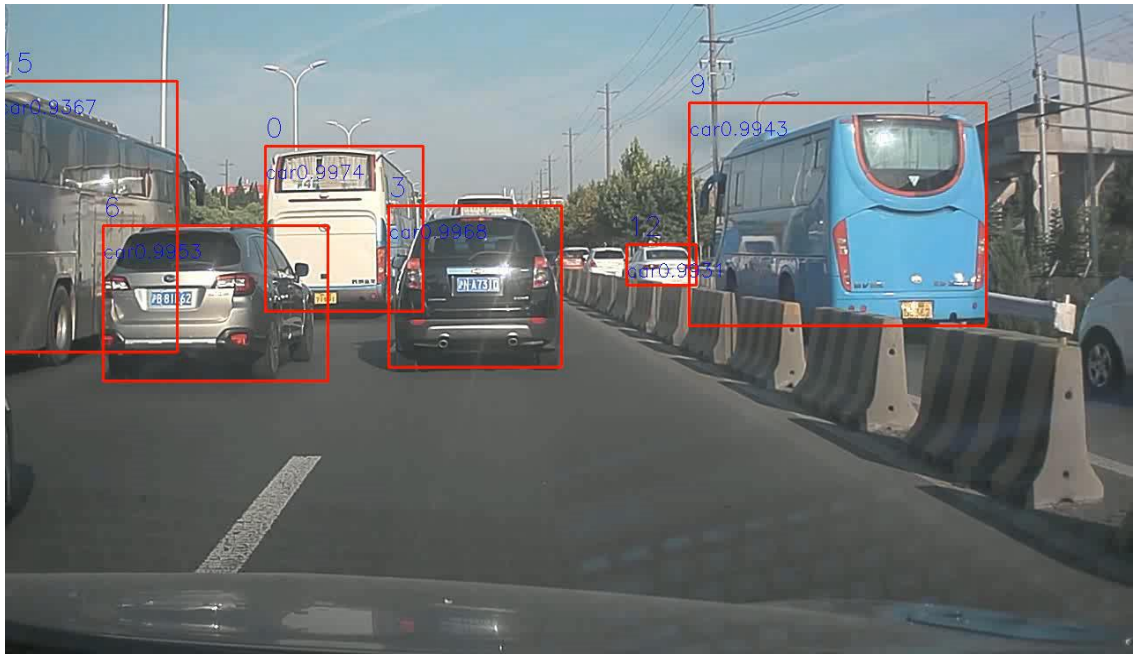
- Deliver various special image effects. Currently capable of delivering over 70 filter effects and 6 artistic filtering effects

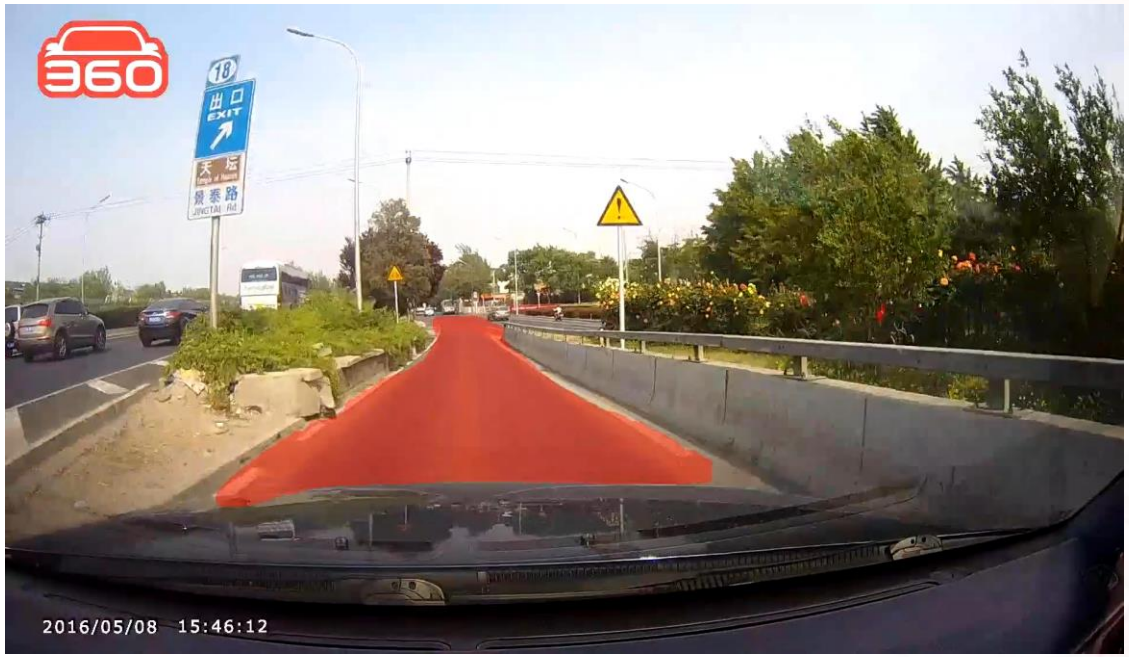
轻松一刻：应用案例


商汤
sensetime
Monitor
History
Task
SenseFace
Target
Statistic
Tools
Apr 3 2016
11:56:16
Task List
Control Task 1
local-video-1
surveillance task1
surveillance task2
surveillance task3
surveillance task4
surveillance task5
surveillance task6
surveillance task7
surveillance task8
surveillance task9
surveillance task10
surveillance task11
surveillance task12
surveillance task13
surveillance task14
surveillance task15
surveillance task16
surveillance task17
surveillance task18
08-13-2015 16:12:52
Captured
Today: 1019
Month: 1019
More
11:56:17
11:56:16
11:56:15
11:56:08
11:56:05
11:55:57
11:55:47
11:55:47
11:55:47
11:55:47
11:55:46
11:55:46
Target
Target Number: 223
More
2016-04-03 11:55:39
local-video-1
Captured
92.6%
Target
chenzhaoj...
2016-04-03 11:54:56
local-video-1
Captured
91.7%
Target
suzhekun
2016-04-03 11:53:51
local-video-1
Captured
93.9%
Target
chenzhaoj...

轻松一刻：应用案例

轻松一刻：应用案例





Enabling cars to sense the surrounding environment and recognizing objects

轻松一刻：应用案例

夜晚、大雾、恶劣天气等困难复杂场景对算法的高要求

轻松一刻：应用案例


Cancle
SwitchSticker
Toggle Camera
Start
iTools.cn


Cartier
iTools.cn


iTools.cn


椰树
29年
鲜榨

谢谢！