AI编译器-系列之前端优化

布局转换

ZOMI

Talk Overview of Frontend Optimizer

I. AI 编译器前端优化

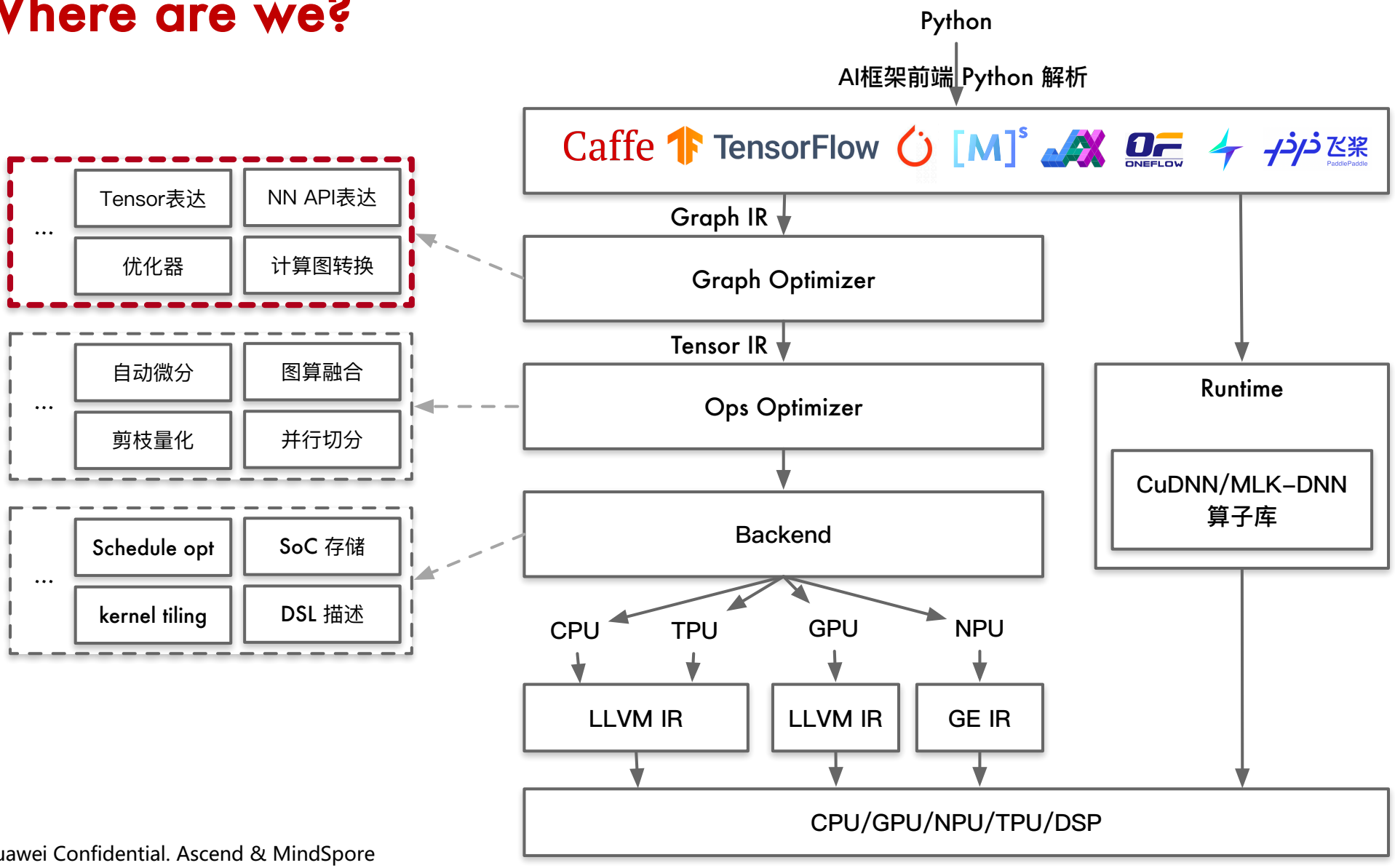
- 图层 - Graph IR
- 算子融合 - OP Fusion
- 布局转换 - Layout Transform
- 内存分配 - Memory Allocation
- 常量折叠 – Constant Fold
- 公共子表达式消除 - CSE
- 死代码消除 - DCE
- 代数简化 - ARM

Talk Overview

Layout Transformation – 布局转换

- 数据内存排布
- 张量数据布局
- NCHW与NHWC
- 华为昇腾数据排布
- 编译布局转换优化

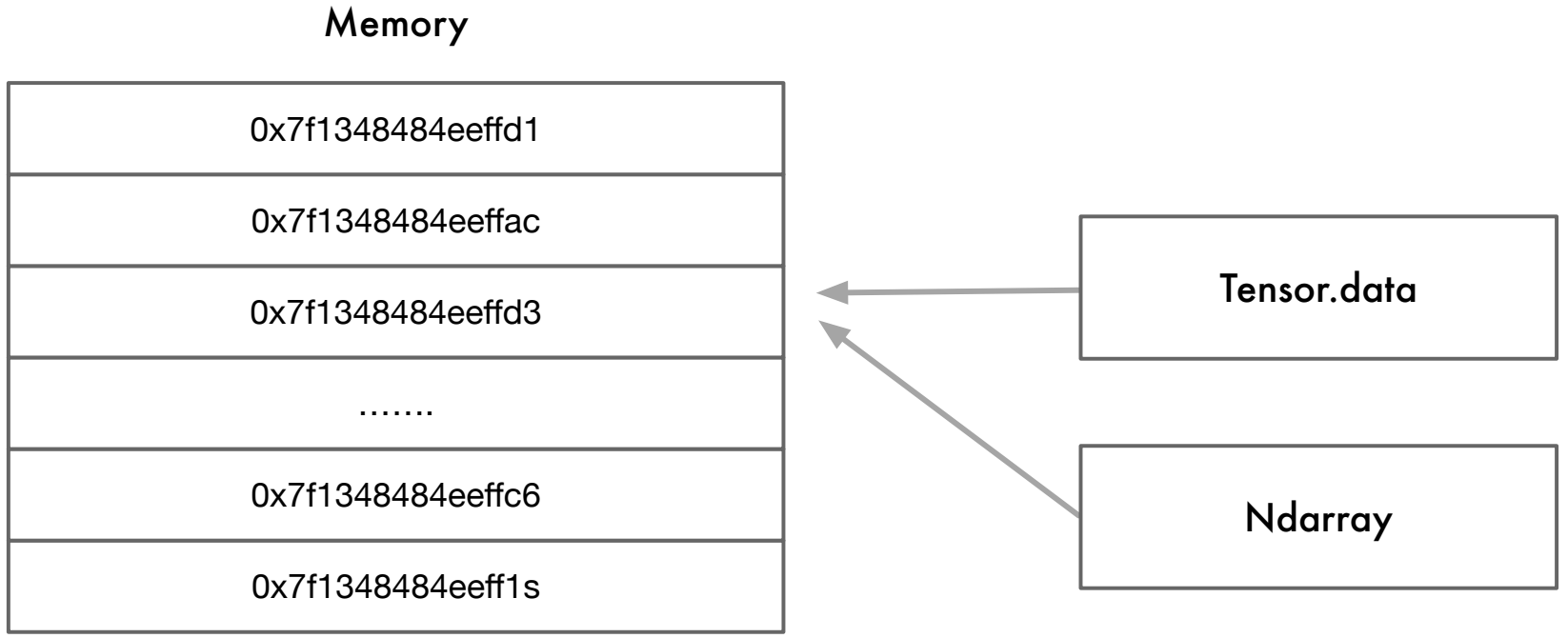
Where are we?



数据内存排布

什么是内存对齐？

- 内存对齐和数据在内存中的位置有关。内存对齐以字节为单位进行，一个变量的内存地址如果正好等于它的长度的整数倍，则称为自然对齐。
- 在32位CPU下，一个u32的内存地址为0x00000004 ，则属于自然对齐。内存空间按照字节进行划分，理论上可以从任意地址开始读取，实际上会要求读取数据的首地址时某一个值的整数倍。

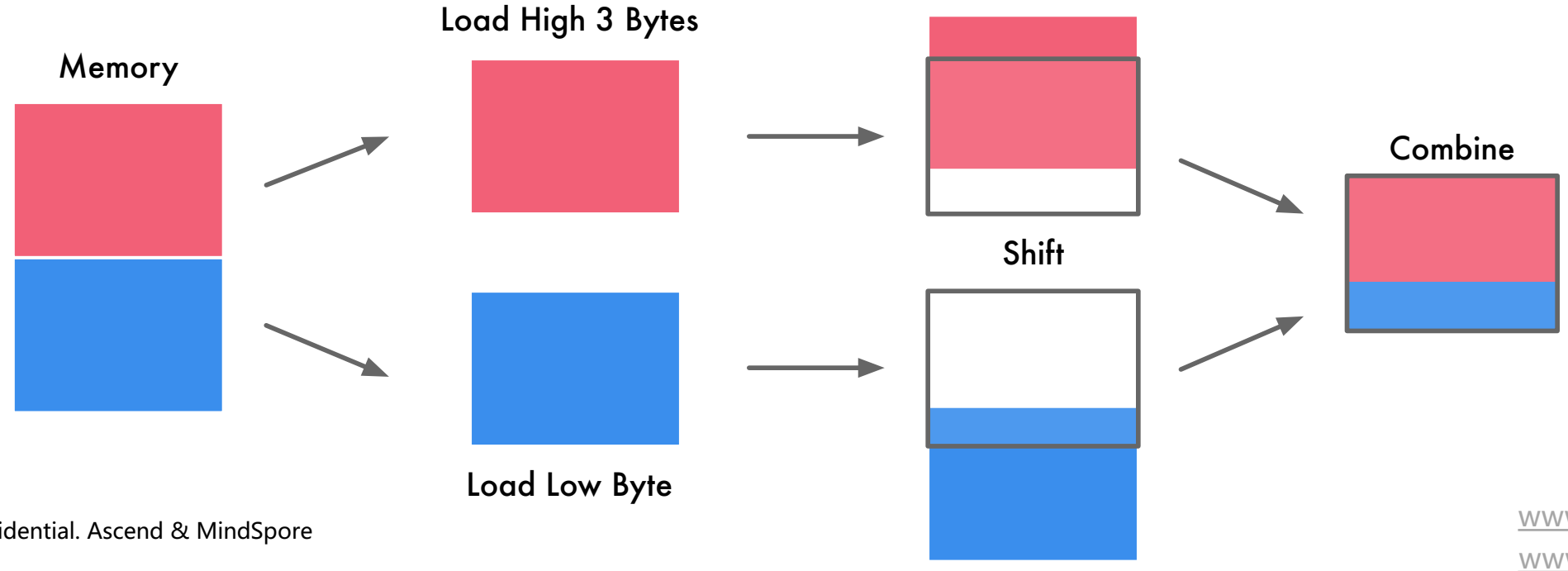


为什么要内存对齐？

- 尽管内存以字节为单位，现代处理器的内存子系统仅限于以字的大小的力度和对齐方式访问，处理器按照字节块的方式读取内存。一般按照2, 4,8, 16 字节为粒度进行内存读取。合理的内存对齐可以高效的利用硬件性能。

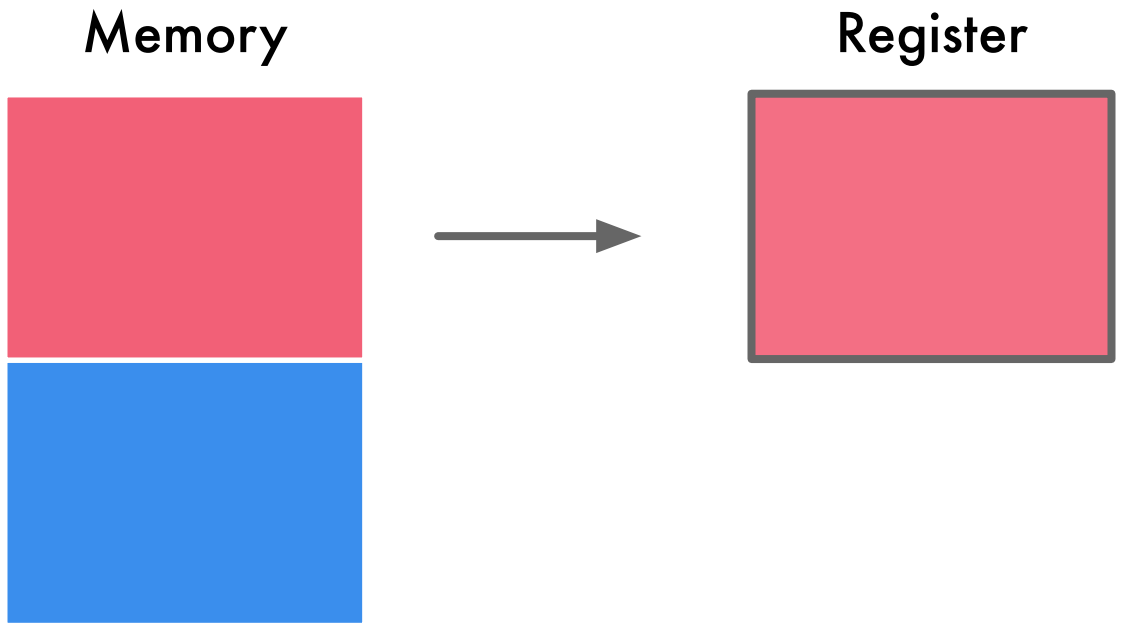
为什么要内存对齐？

- 以4字节存取粒度的处理器为例，读取一个int变量（32bit 系统），处理器只能从4的倍数的地址开始。假如没有内存对齐机制，将一个int放在地址为1的位置。现在读取该int时，需要两次内存访问。第一次从0地址读取，剔除首个字节，第二次从4地址读取，只取首个字节；最后两下的两块数据合并入寄存器，需要大量工作。



为什么要内存对齐？

- 有了严格的内存对齐，int必须按照对其规则进行存储，起始位置必须是4的整数倍，只需要进行一次读取：



以 bit 大小粒度进行内存访存？

访问速度

- 当代处理器具有多个级别的高速缓存，数据必须通过这些高速缓存；支持单字节读取将使内存子系统的吞吐量与执行单元的吞吐量紧密的绑定（也就是CPU吞吐量），消耗大量 CPU 资源的的同时，称为系统性能的瓶颈。可以类比在硬盘读写中 DMA(Direct Memory Access) 性能是如何超越 PIO（Programmed Input/Output）的。
- CPU 总是以其字的大小进行内存读取，进行未对齐的内存访问时，处理器将读取多个字，需要读取变量所跨越内存的所有字，同时进行处理。将导致访问请求数据所需要的内存事务增加2倍。

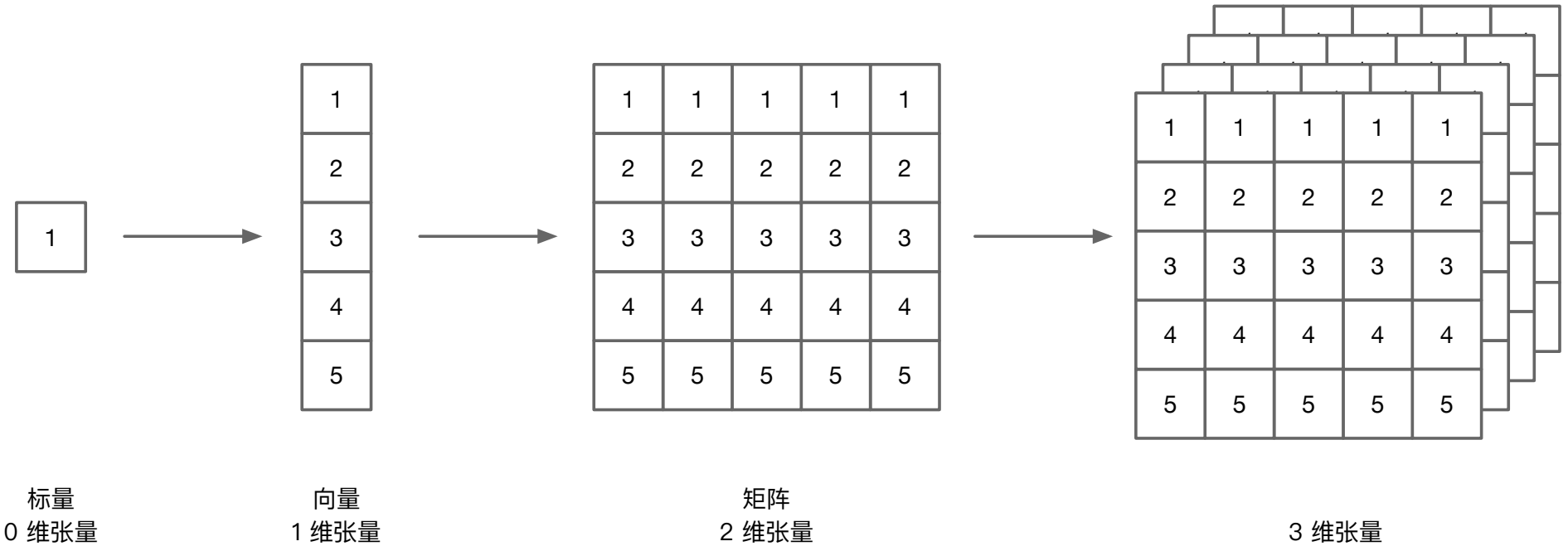
原子性

- CPU 可以在一个对齐的内存字上操作，意味着没有指令可以中断该操作。这对于许多无锁数据结构和其他并发范式的正确性至关重要。

张量数据布局

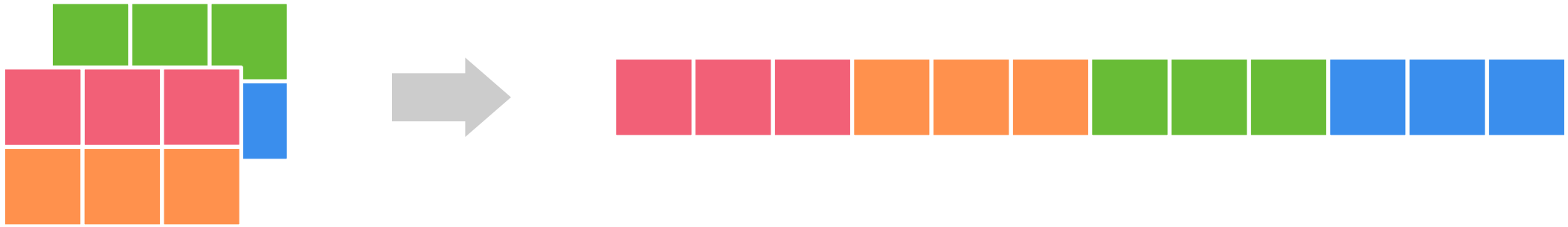
张量与内存排布

- Tensor 是一个多维数组，它是标量、向量、矩阵的高维拓展。标量是一个零维张量，没有方向，是一个数。一维张量只有一个维度，是一行或者一列。二维张量是一个矩阵，有两个维度，灰度图片就是一个二维张量。当图像是彩色图像（RGB）的时候，就得使用三维张量了。



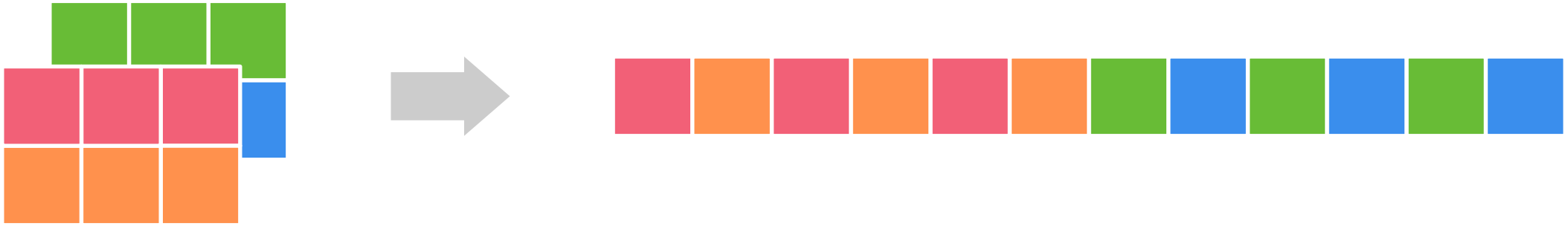
行优先和列优先排布方式

- 形状 (3,2,2) 三维张量，行优先数据布局



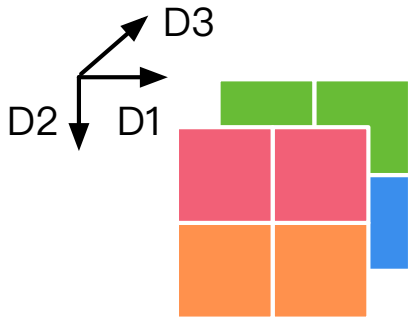
行优先和列优先排布方式

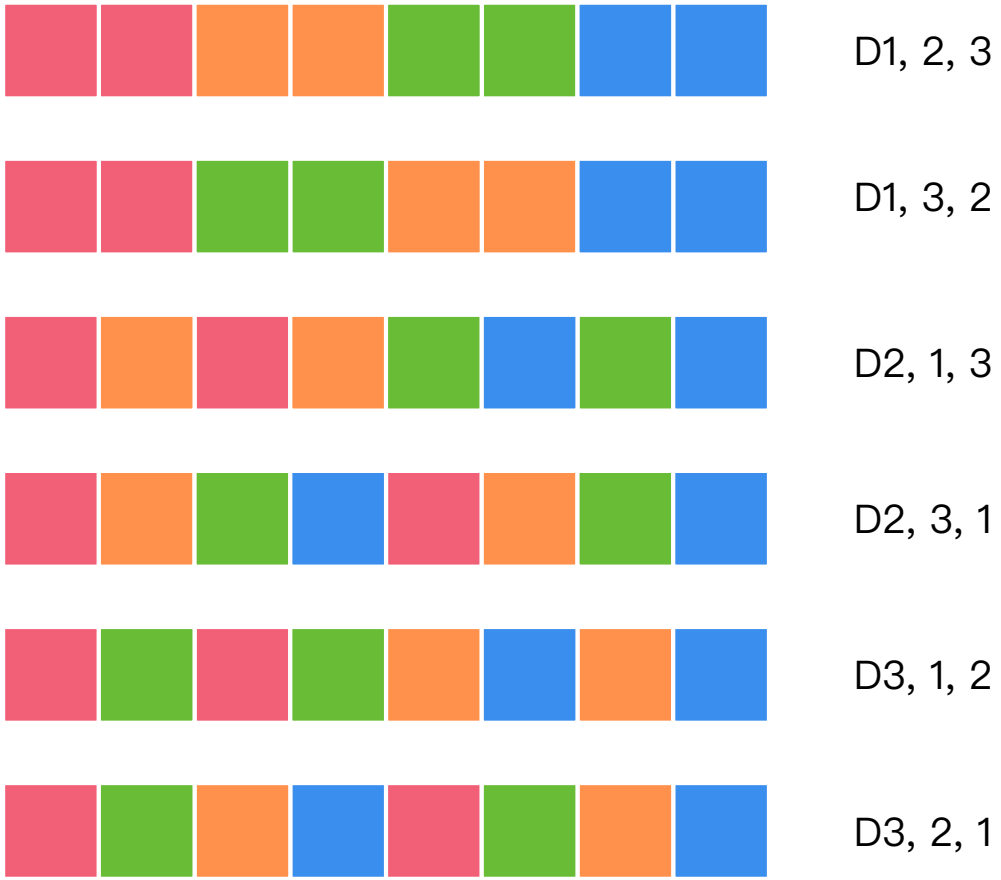
- 形状 (3,2,2) 三维张量，列优先数据布局



行优先和列优先排布方式

- 形状 (2,2,2) 三维张量，数据布局方式有：





NCHW与NHWC

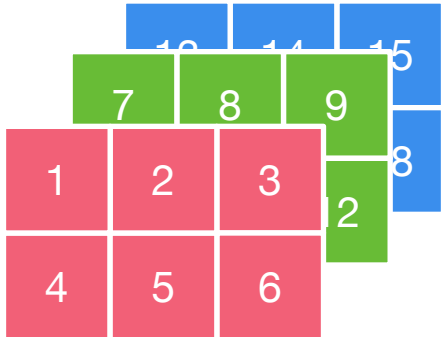
Why

- 尽管存储的数据实际上是一样的，但是不同的顺序会导致数据的访问特性不一致，因此即使进行同样的运算，相应的计算性能也会不一样。

- 在深度学习领域，多维数据通过多维数组存储，比如卷积神经网络的特征图（Feature Map）通常用四维数组保存，即4D，4D格式解释如下：
  1. N：Batch数量，例如图像的数目。
  2. H：Height，特征图高度，即垂直高度方向的像素个数。
  3. W：Width，特征图宽度，即水平宽度方向的像素个数。
  4. C：Channels，特征图通道，例如彩色RGB图像的Channels为3。

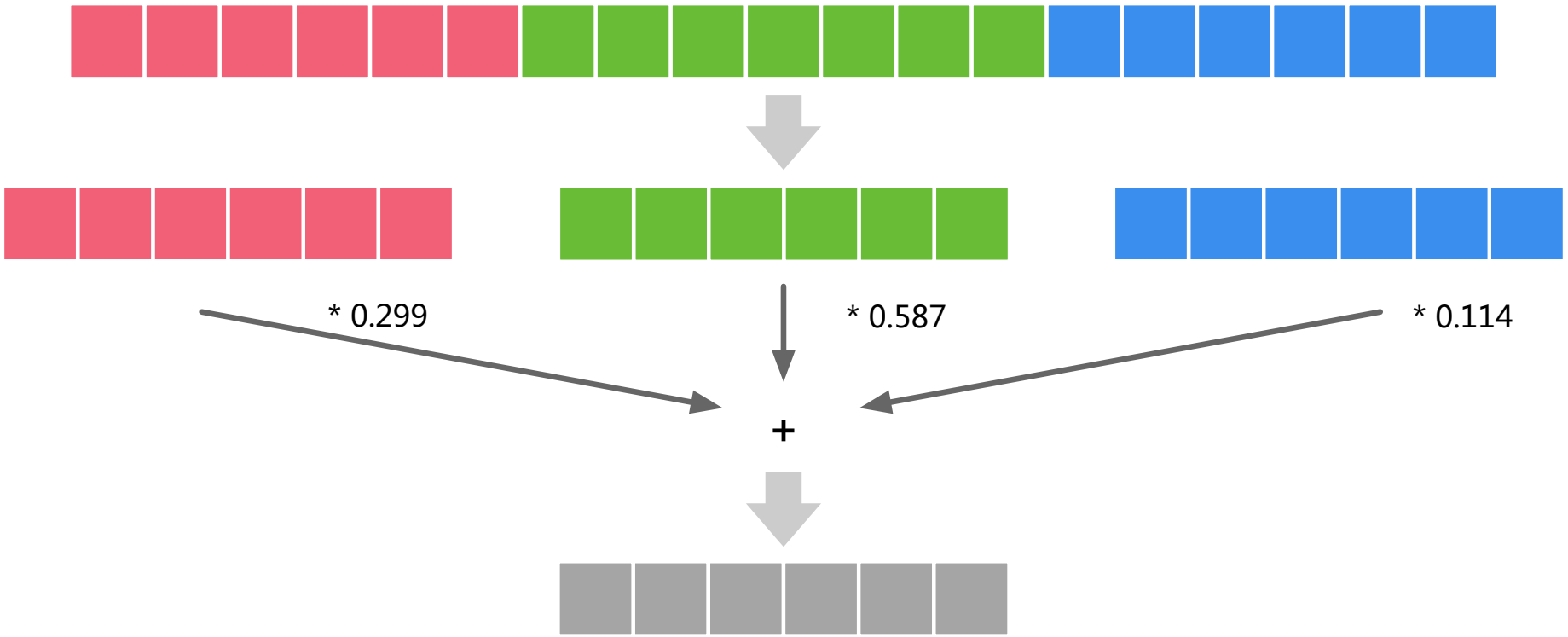
NCHW

- NCHW 同一个通道的数据值连续排布，更适合需要对每个通道单独运算的操作，如 MaxPooling
- NCHW 计算时需要的存储更多，适合GPU运算，利用 GPU 内存带宽较大并且并行性强的特点，其访存与计算的控制逻辑相对简单：



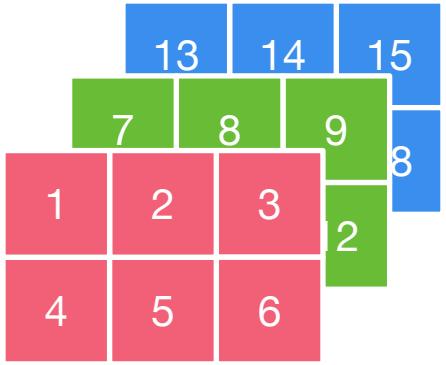
NCHW

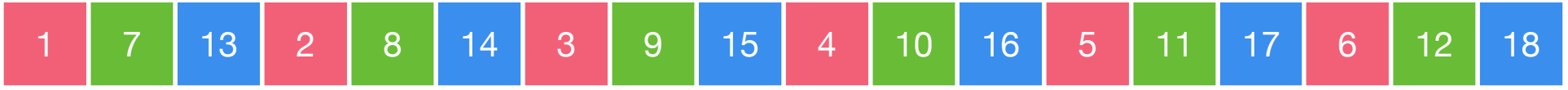
- R 通道所有数据值乘以 0.299，G 通道所有数据值乘以 0.587，B 通道所有数据值乘以 0.114，最后将三个通道结果相加得到灰度值：



NHWC

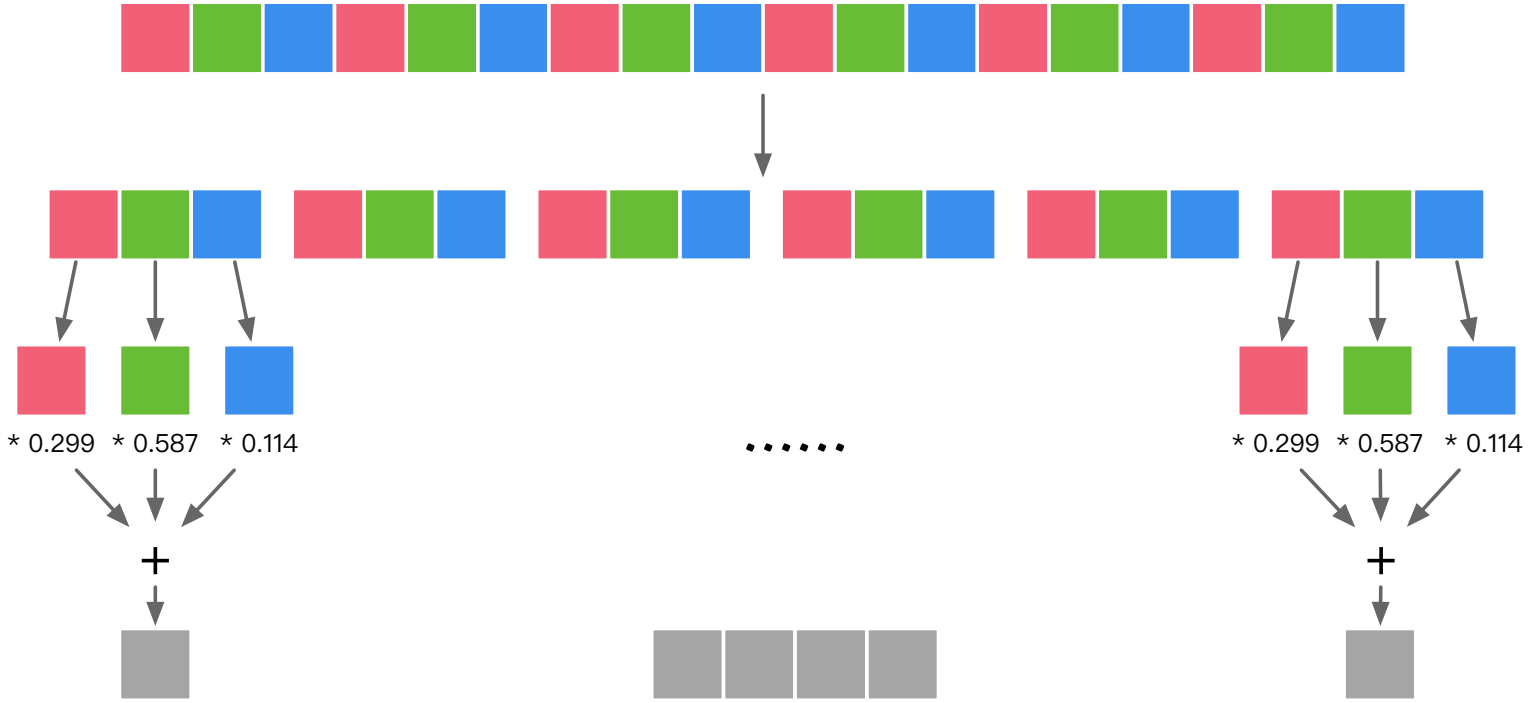
- NHWC 其不同通道中的同一位置元素顺序存储，因此更适合那些需要对不同通道的同一数据做某种运算的操作，比如“Conv1x1”
- NHWC 更适合多核CPU运算，CPU 内存带宽相对较小，每个数据计算的时延较低，临时空间也很小，有时计算机采取异步的方式边读边算来减小访存时间，因此计算控制灵活且复杂





NHWC

- 输入数据分成多个 (R, G, B) 像素组，每个像素组中 R 通道像素值乘以 0.299，G 通道像素值乘以 0.587，B 通道像素值乘以 0.114 后相加得到一个灰度输出像素。将多组结果拼接起来得到所有灰度输出像素：

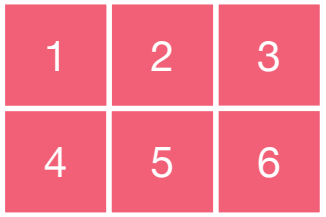


框架的默认选择

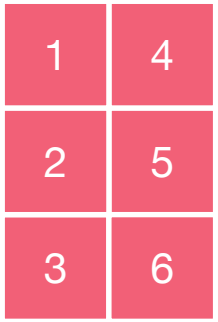
- 由于数据只能线性存储，因此这四个维度有对应的顺序。不同深度学习框架会按照不同的顺序存储特征图数据：
  - 以 NPU/GPU 为基础的 PyTorch 和 MindSpore 框架默认使用 NCHW 格式，排列顺序为[Batch, Channels, Height, Width]
  - Tensorflow 采用了 NHWC，排列顺序为[Batch, Height, Width, Channels]，何面向移动端部署 TFLite 只采用 NHWC 格式

连续与非连续问题

- 连续张量



- 非连续张量



对不连续存储的张量执行连续变换，也就是重新开辟内存，按逻辑结构填入对应的物理结构

Reference

1. CANN V100R020C20 TBE自定义算子开发指南 https://support.huawei.com/enterprise/zh/doc/EDOC1100180762/f96da97d
2. CANN V100R020C20 TBE自定义算子开发指南 (推理) https://support.huawei.com/enterprise/zh/doc/EDOC1100180762/8e6a99eb
3. 深度学习NCHW和NHWC数据格式 https://blog.csdn.net/Dontla/article/details/123141775
4. [DLComplier] The Deep Learning Compiler: A Comprehensive Survey – 3 https://zhuanlan.zhihu.com/p/543187086
5. Pytorch NCHW/NHWC 理解 https://zhuanlan.zhihu.com/p/556222920
6. https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html
7. 深度学习框架zf_谈谈深度学习框架的数据排布 https://blog.csdn.net/weixin_26854555/article/details/112360638
8. 华为Ascend昇腾CANN详细教程（一） https://blog.csdn.net/m0_37605642/article/details/125691134
9. Tensor中数据摆放顺序NC4HW4是什么意思，只知道NCHW格式，能解释以下NC4HW4格式吗？ https://www.zhihu.com/question/337513515/answer/768632471
10. 谈谈深度学习框架的数据排布 https://zhuanlan.zhihu.com/p/149464086
11. 数据排布格式 https://support.huaweicloud.com/TIKopdevgd_beta/tik1.5_10_0005.html
12. 数据布局与内存对齐 https://books.innohub.top/rustinfo/info/alignment
13. https://hughiehao.github.io/2021/10/28/%E5%BC%A0%E9%87%8F%E6%95%B0%E6%8D%AE%E5%AD%98%E5%82%A8%E6%96%B9%E5%BC%8F.html