大规模分布式机器学习系统设计与应用经验分享

涂威威

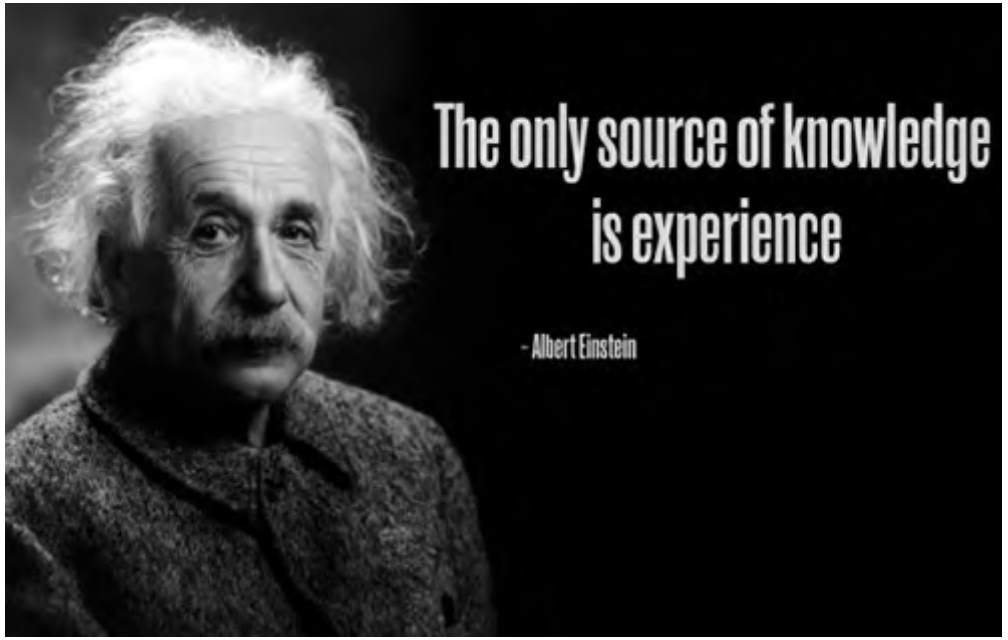
目录

机器学习系统

- 机器学习的经典定义
- 典型的机器学习应用过程
- 机器学习的核心系统

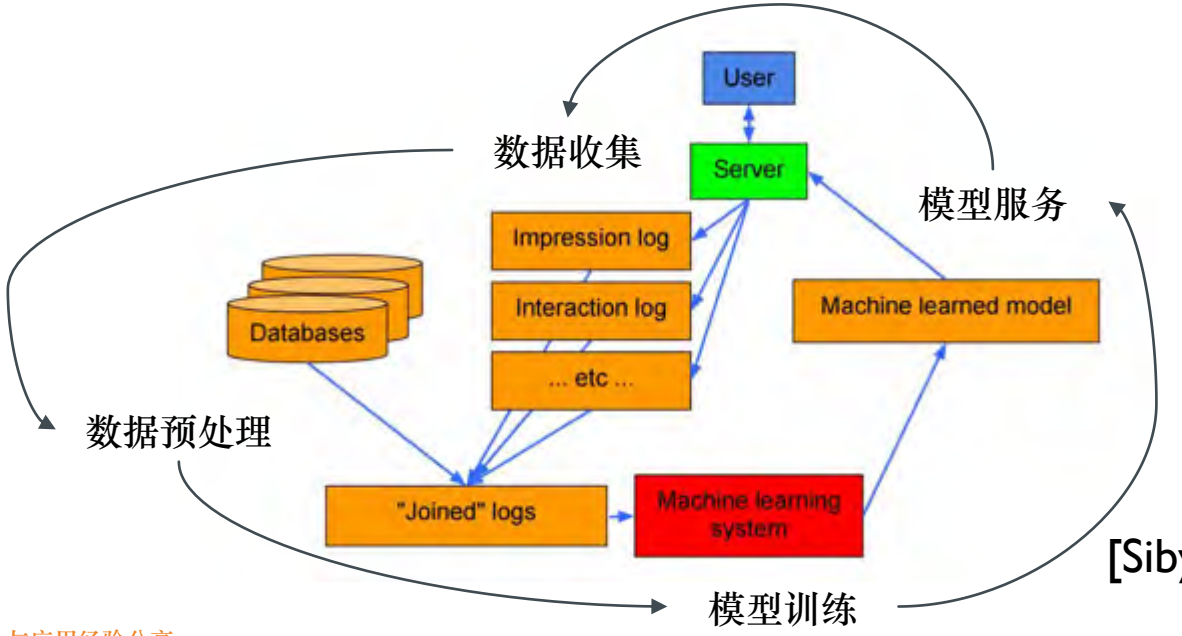
机器学习的经典定义

- 利用**经验**改善系统性能
- 经验 → 数据
- 机器学习无处不在
  - 搜索与推荐
  - 生物特征识别
  - 自动驾驶
  - 军事决策助手（DARPA）
  - …



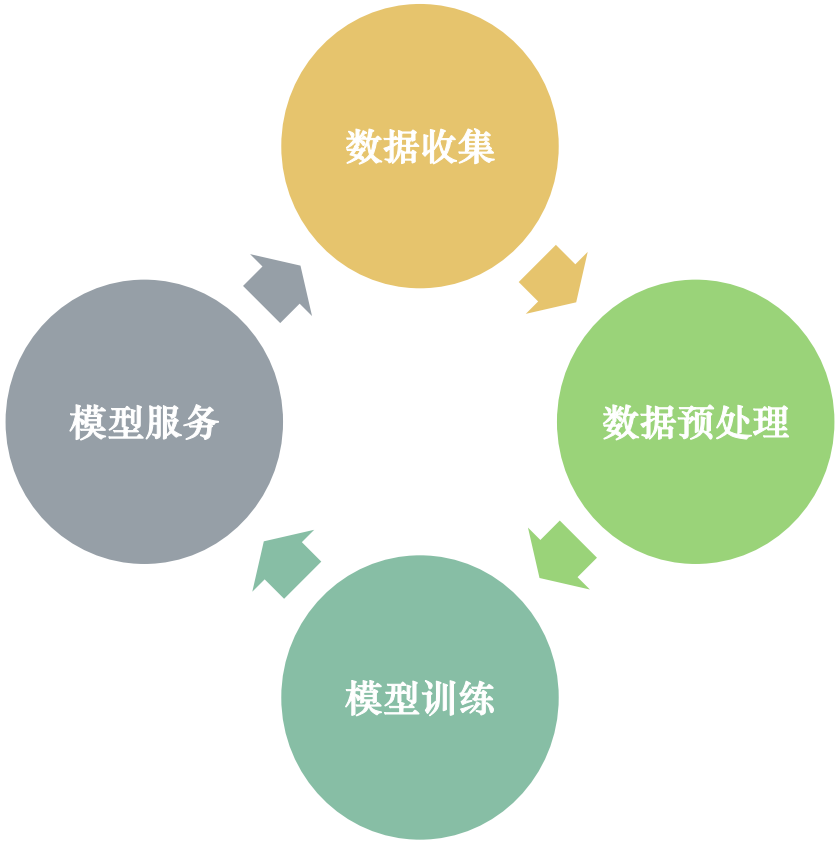
典型的机器学习应用过程（广告点击率预估系统为例）

- 问题定义：收入 = 平均每次点击价格 * 点击率 * 广告展现量
- 应用过程：数据收集→数据预处理→模型训练→模型服务



[Sibyl, Google]

机器学习的核心系统

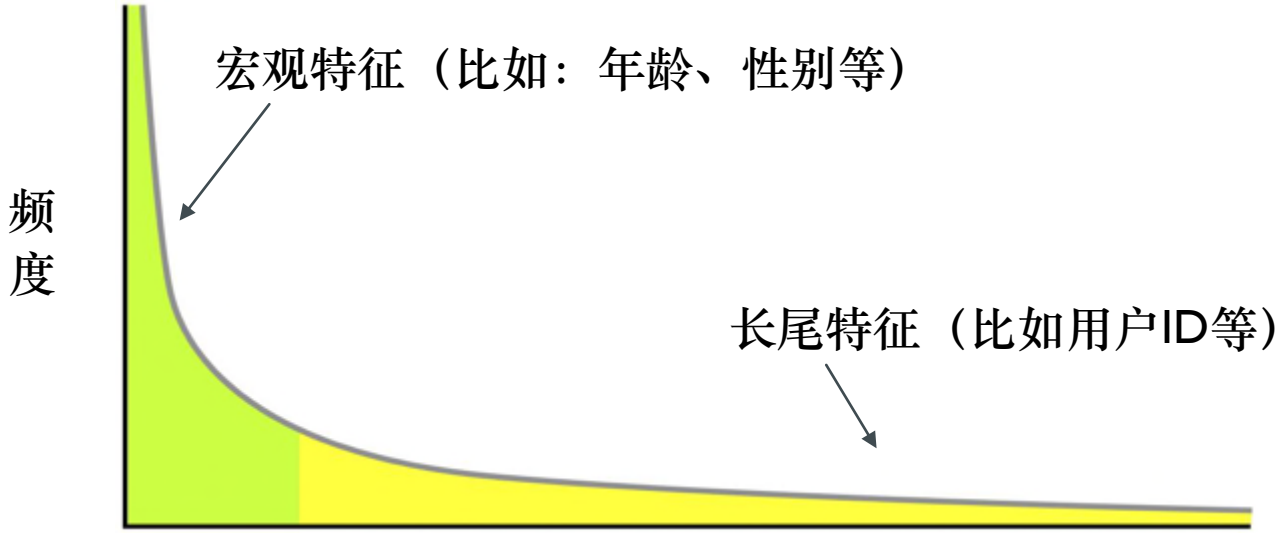


大规模分布式模型训练框架设计

- 工业应用模型训练框架设计目标
- 开发效率：计算模型和编程模型的选择
- 执行效率：计算优化举例

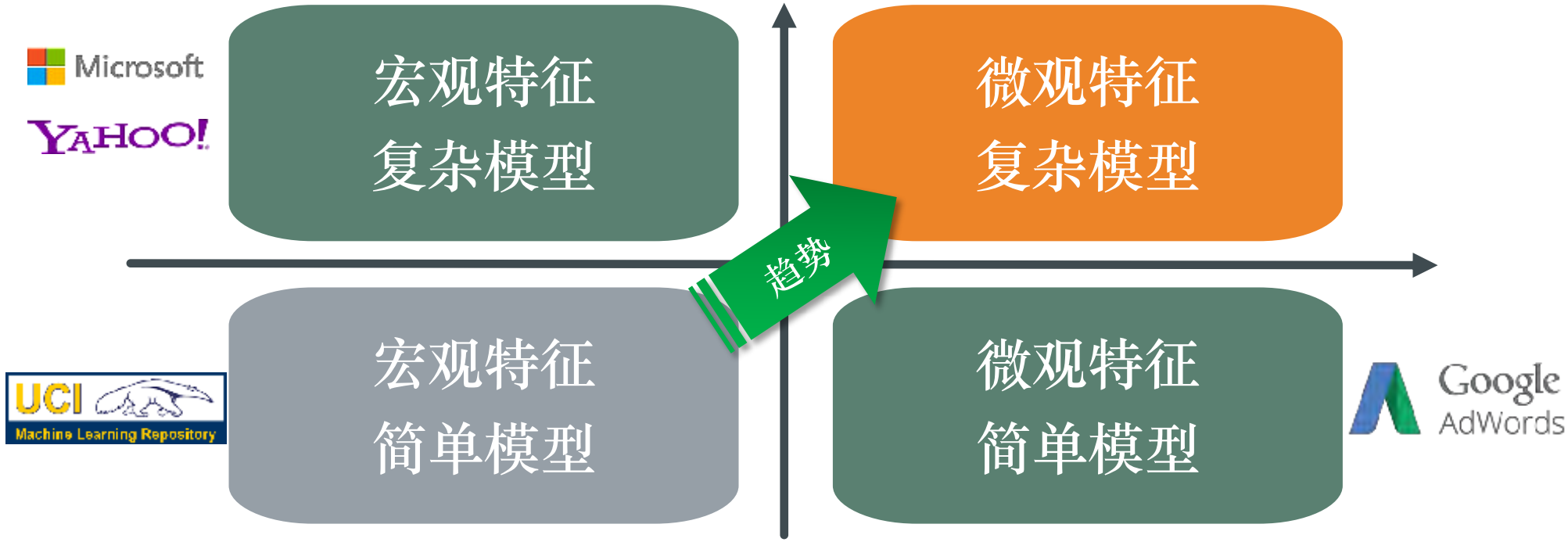
机器学习在工业应用中的发展趋势

- 有效数据的增长
  - 数据量：10^4 → 10^{10}~10^{12}
- 特征维度的增长
  - 宏观特征（10^3）→ 微观特征（ 10^{10}~10^{12}）



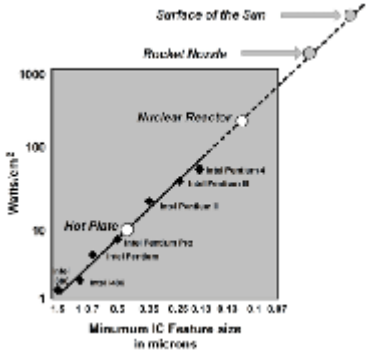
机器学习在工业应用中的发展趋势

机器学习算法在工业应用中的四个象限



第四象限的需求

- 摩尔定律失效
  - 能耗墙（Power Wall）
  - 延迟墙（Latency Wall）
- 单机能力有限
  - IO、存储、计算有限
- 目前提升计算能力的主流方式
  - 并行化：降低执行延迟→提升**吞吐**
  - 但是，Amdahl定律



[Power Wall]

[Latency Wall]

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

[Amdahl定律]

目标

需要**分布式并行**

机器学习算法的NO FREE LUNCH

- No Free Lunch定理：[Wolpert and Macready 1997]任意两个算法a_1和a_2，

$$\sum_f P(d_m^y|f,m,a_1) = \sum_f P(d_m^y|f,m,a_2)$$

- 任意算法（包括随机算法）在**所有问题**上的期望性能一样
  - **不存在通用算法**
  - 但在具体的实际问题上，有可能存在比其他算法好的算法
    - 需要针对不同的实际问题，研究开发不同的机器学习算法

目标

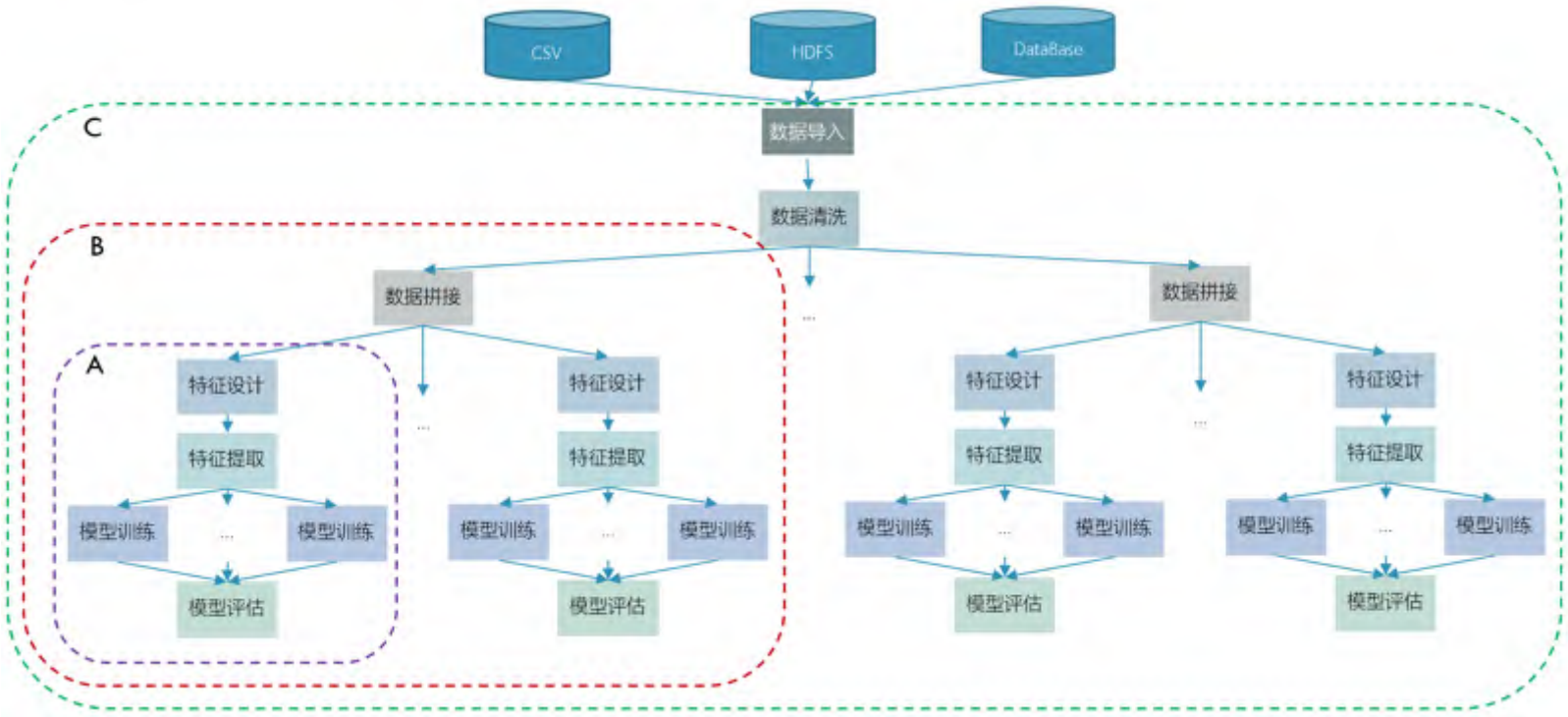
需要**高开发效率**

典型的机器学习建模过程

- 根据No Free Lunch定理：
  - 实际问题需要很多的尝试
    - 不同的数据
    - 不同的特征表达
    - 不同的模型、模型不同的参数
- 模型训练是被重复最多的模块

目标

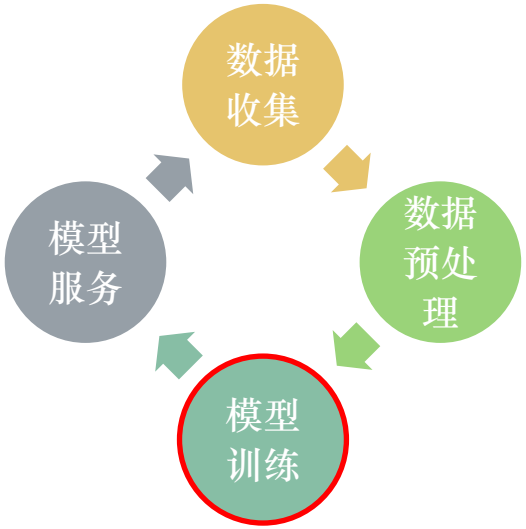
需要**高执行效率**



底层框架的NO FREE LUNCH

没有在**所有问题**上最好的架构，只有最适合实际问题的

系统	计算密集度	存储		通讯		一致性要求
		读写量	性能要求	通讯量	性能要求	
数据收集	✦	✦✦✦	✦	✦✦✦	✦	✦✦✦
数据预处理	✦✦	✦✦	✦✦	✦✦	✦✦	✦✦✦
模型训练系统	✦✦✦	✦✦✦	✦✦✦	✦✦✦	✦✦✦	✦✦
模型服务系统	✦✦	✦	✦✦✦	✦✦	✦✦✦	✦✦✦



目标

需要针对机器学习的兼顾开发效率和执行效率的大规模分布式并行计算框架

开发效率

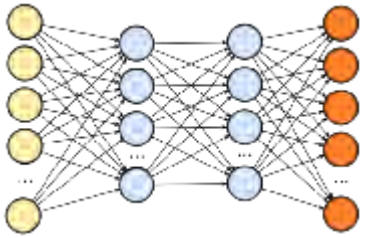
- 计算和编程模型的选择
- 编程语言选择

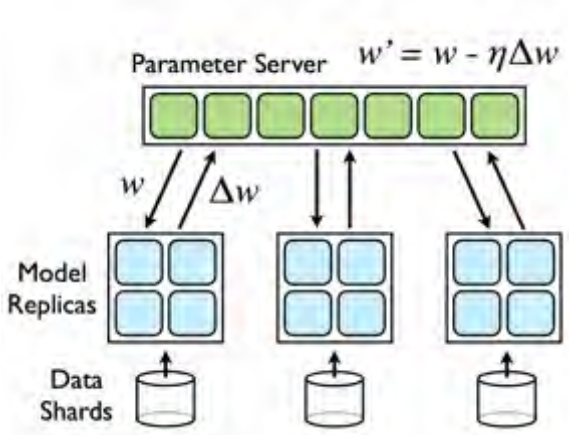
开发效率：分布式并行模型

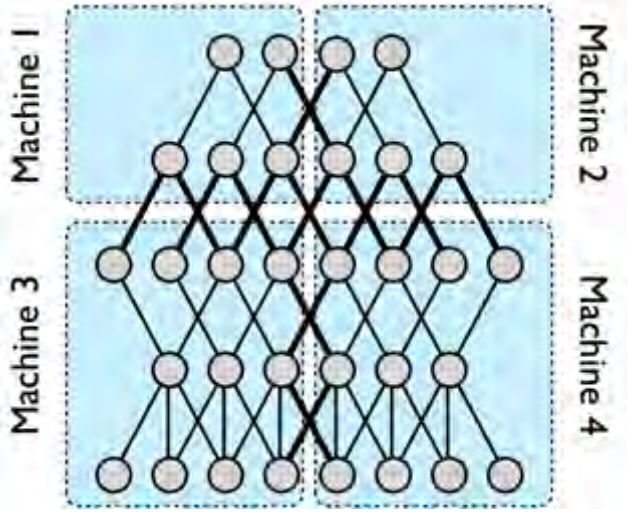
- 数据分布式和模型分布式

训练数据

模型参数

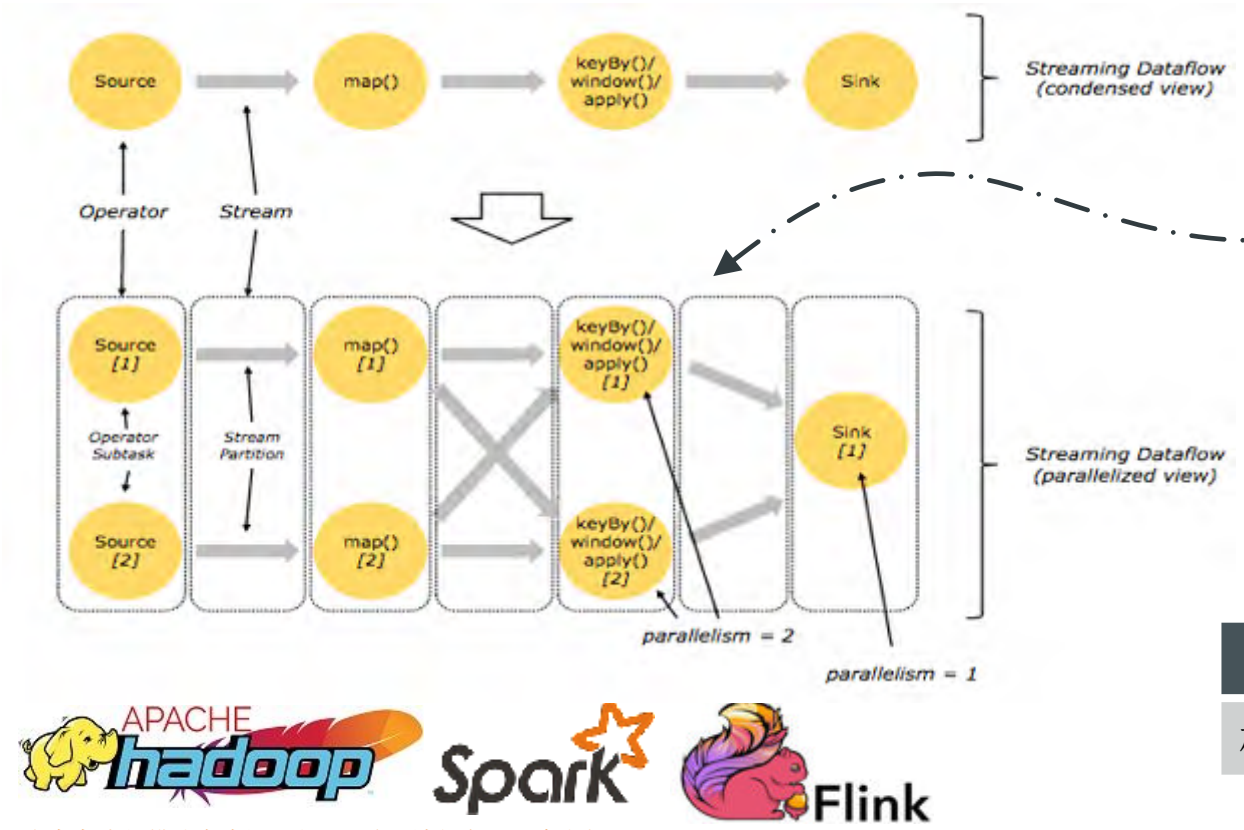


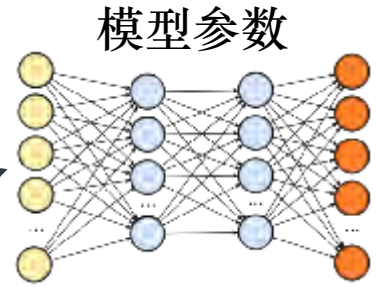




[Large Scale Distributed Deep Networks, Google]

典型计算模型：数据流





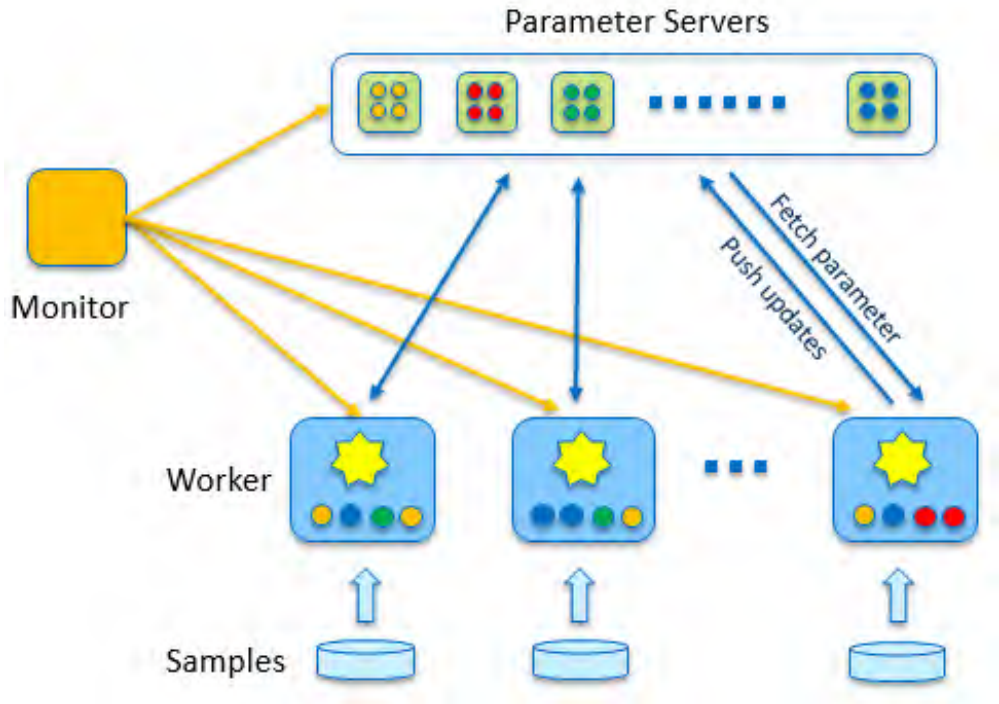
典型机器学习模型优化过程：

- 查取原模型w^{old}
- 根据原模型计算更新Δw
- 更新模型$w^{new} = w^{old} + \Delta w$

问题

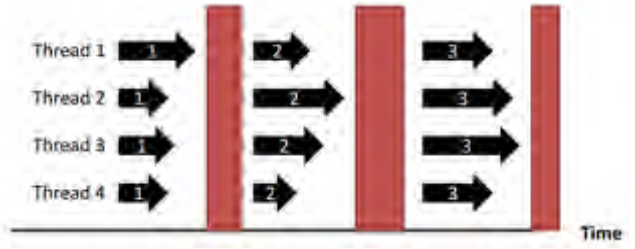
模型参数w是一个所有计算共享的中间状态

典型计算模型：参数服务器

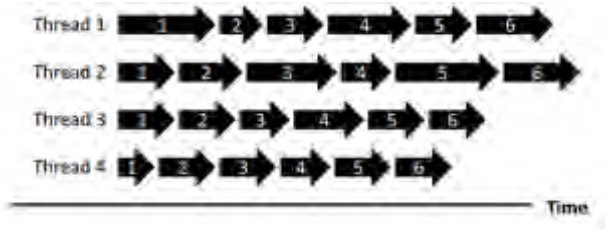


Google DMLC PETUUM

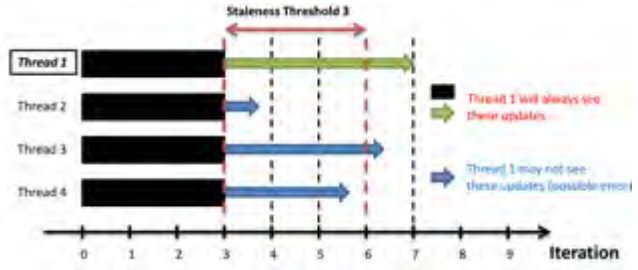
一致性模型



[BSP]

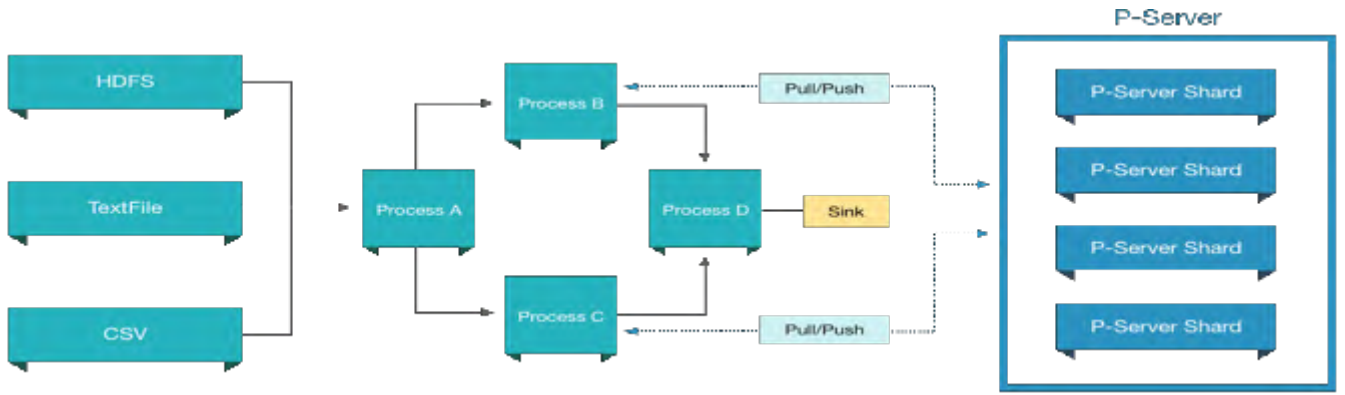


[ASP]



[SSP, Xing]

趋势：数据流 + 参数服务器


P-Server
HDFS
TextFile
CSV
Process A
Process B
Process C
Process D
Pull/Push
Pull/Push
Sink
P-Server Shard
P-Server Shard
P-Server Shard
P-Server Shard

开发效率：命令式和声明式编程

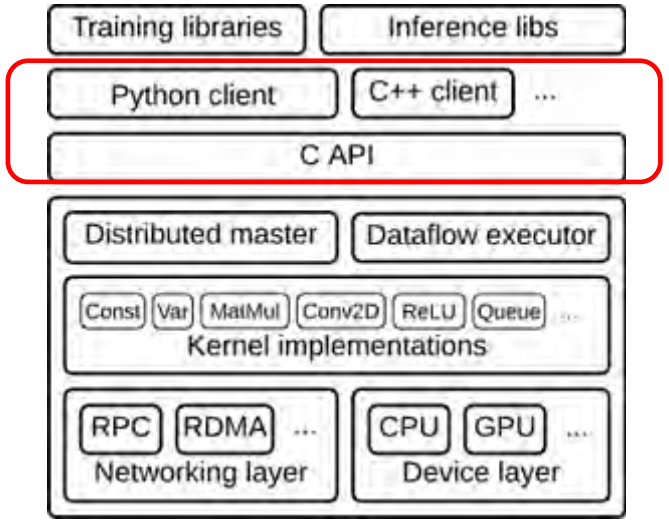
- 命令式更灵活
  - 语言举例：Fortran、C/C++
  - 告诉机器具体的执行流
- 声明式可能更高效
  - 语言举例：DSL（e.g. SQL、Prolog）
  - 只定义任务目标，不指定具体执行
- 实际的系统可能是混合的

命令式	**var** numbers = [1,2,3,4,5]
	var total = 0
	
	for(**var** i = 0; i < numbers.length; i++) {
	  total += numbers[i]
	}

声明式	**var** numbers = [1,2,3,4,5] **var** total = numbers.**reduce**(**function**(sum, n) { **return** sum + n }, 0);

开发效率：编程语言的选择

- 兼顾运行效率和易用性：前后端分离
  - C/C++、Java/Scala作为后端
  - Python、R等作为前端
- Java vs. C/C++
  - 生态：Java生态优于C/C++
  - 可移植性：JVM可移植性高于C/C++
  - 内存管理：Java GC大数据、同步下情况更严重
  - 抽象: C++模板 vs. Java泛型



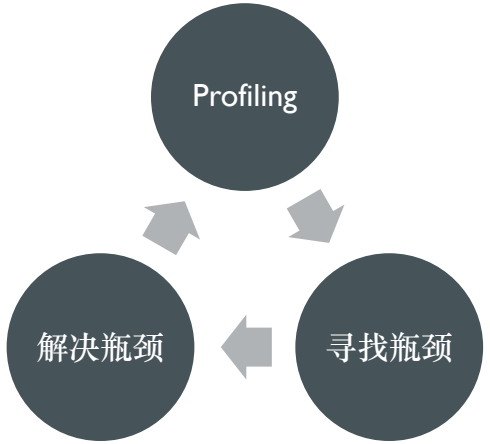
[Tensorflow]

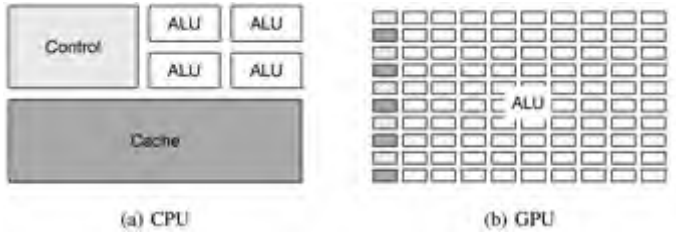
执行效率优化举例

- 计算
- 存储
- 通讯
- 容错

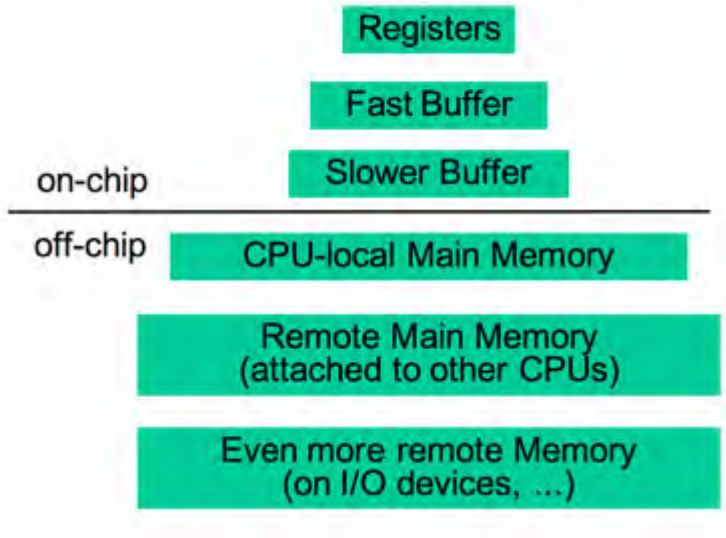
执行效率优化举例 – 计算

- 均衡
  - 负载均衡（不同机器之间、不同资源之间）
  - 任务分解，异步竞争
- 单机、分布式分离优化
  - 分布式overhead很大
  - **不能为了分布式而分布式**
- 利用不同硬件的优势
  - CPU：复杂指令，分支预测，大缓存，任务并行
  - GPU：简单指令，小缓存，粗粒度数据并行
  - FPGA：低峰值、低功耗（vs. GPU）

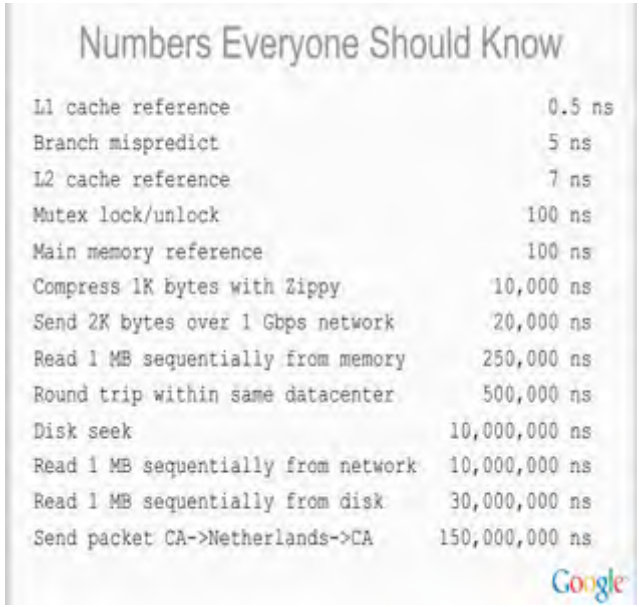




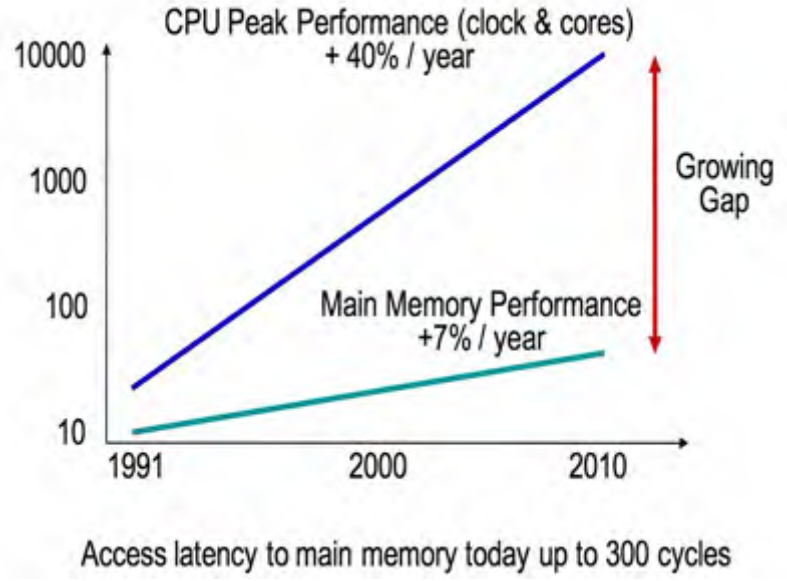
执行效率优化举例 – 存储



[The Memory Hierachy]



[Jeff Dean]



[The Memory Wall]

https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

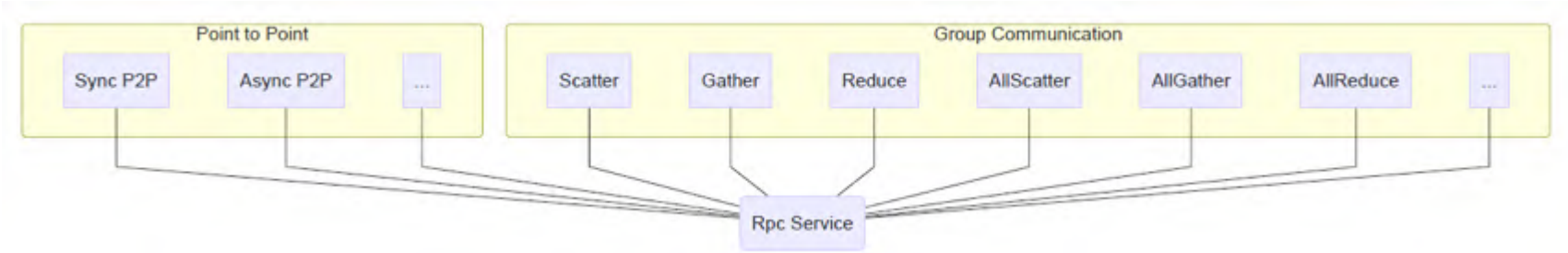
执行效率优化举例 – 存储

- 数据本地化
  - 迭代之间数据本地缓存
- 数据结构优化
  - 数据访问重新排序
  - 调整数据布局
    - 更紧凑的数据结构
      - `Instance: std::vector<float>`
      - `InstanceBlock: std::vector<Instance>` → `std::vector<float>`
    - 冷热分离
  - 数据预取

[Z秩序(曲线)]

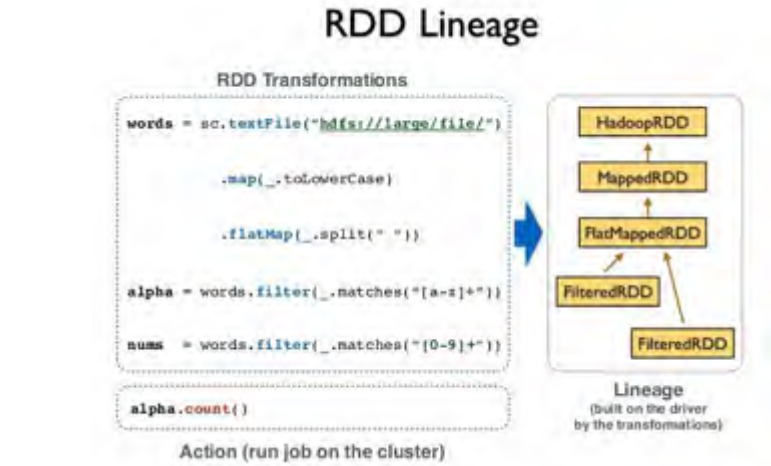
执行效率优化举例 – 通讯

- 通讯模式
  - 点对点
    - RPC
  - 组通讯
    - Reduce、Gather、Scatter
- 软件优化
  - 序列化框架优化
  - 通讯压缩：CPU和带宽互换
  - 应用层：请求合并、缓存
- 硬件优化
  - 多网卡
  - Infini-Band

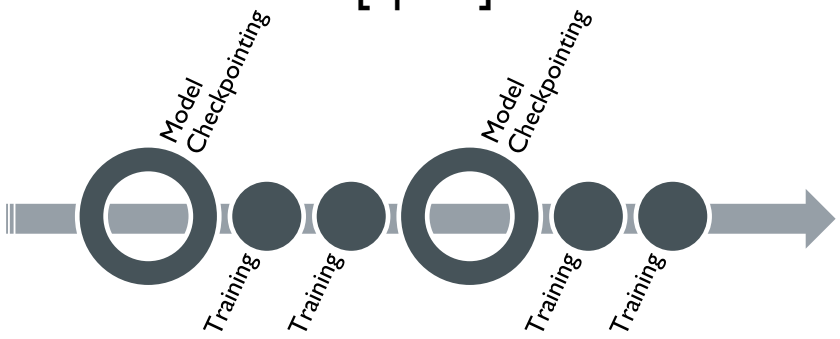


执行效率优化举例 – 容错

- 选择最适合的Tradeoff
  - 机器学习任务迭代式计算，中间状态多
  - 模型参数是最重要的状态
  - 机器学习训练不需要强一致性
- Data Lineage vs. Checkpointing
  - Data Lineage (Spark…)
    - 可选择不同粒度
  - Checkpointing(Tensorflow、MxNet…)
    - 对迭代式的机器学习模型参数和简单的任务状态做粗粒度的Checkpointing
- 冗余
  - Cold / Warm / Hot Standby
  - Hybrid



[Spark]



[Tensorflow]

机器学习实际应用的常见陷阱

- 一致性
- 开放世界
- 依赖管理
- 可理解性/可调试性

一致性

- 训练/预估一致性
  - 特征表达不一致
    - 逻辑错误
    - 表达方式不一
    - “穿越”
    - …
  - 目标含义不一致
    - 比如搜索引擎里 是否点击→是否满意
- 字段含义时间一致性
  - 随着时间的推移，某些特征含义会发生变化

训练时

特征值	特征含义
0	男
1	女

预测时

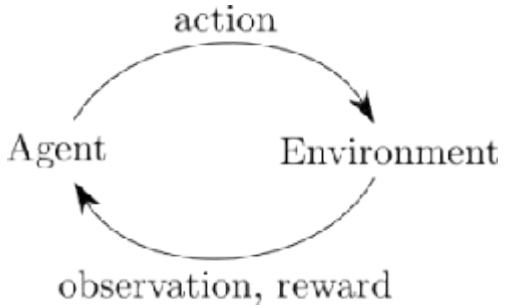
特征值	特征含义
0	女
1	男

[特征“穿越”]

开放世界

- 幸存者偏差
  - 当前模型影响下一次模型的训练数据

[World War II, Abraham Wald]

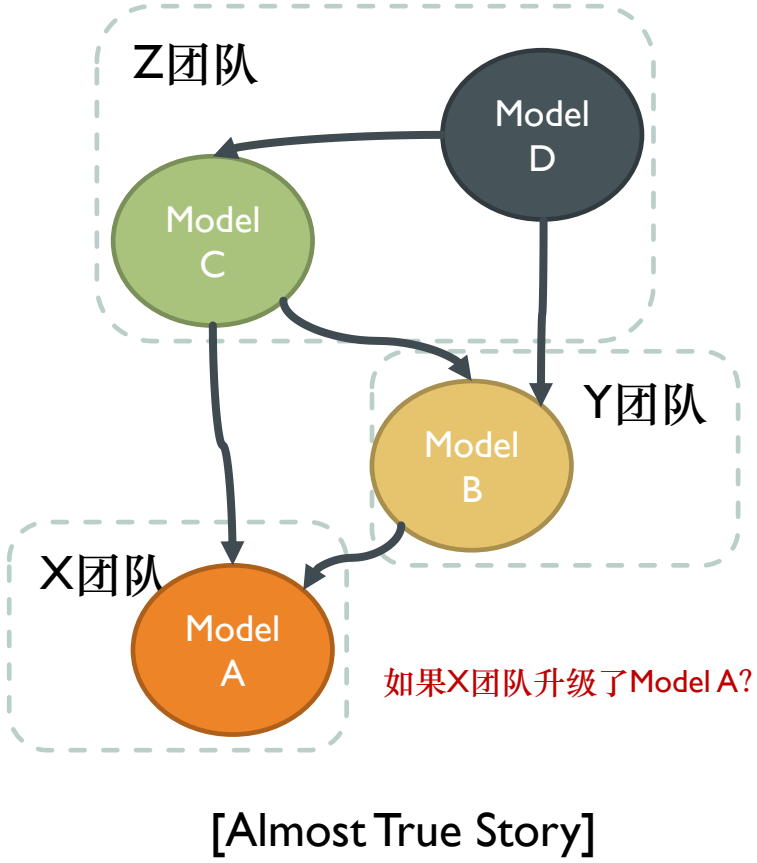


[强化学习]

- 与其他系统配合
  - 模型输出发生着变化，但是系统中的阈值固定不变

依赖管理

- 数据依赖
  - 数据依赖比代码模块依赖更可怕
  - 与传统软件不同，机器学习系统的表现依赖外部数据
- 模型之间依赖
  - 有时为了求快，容易依赖其他团队模型的输出
    - 如果依赖的团队升级的模型?
- 隐性依赖
  - 可能会有一些特征字段会被模型自己改变
    - 比如：用户点击推荐文章的次数 这个特征会随着推荐模型的升级而发生改变



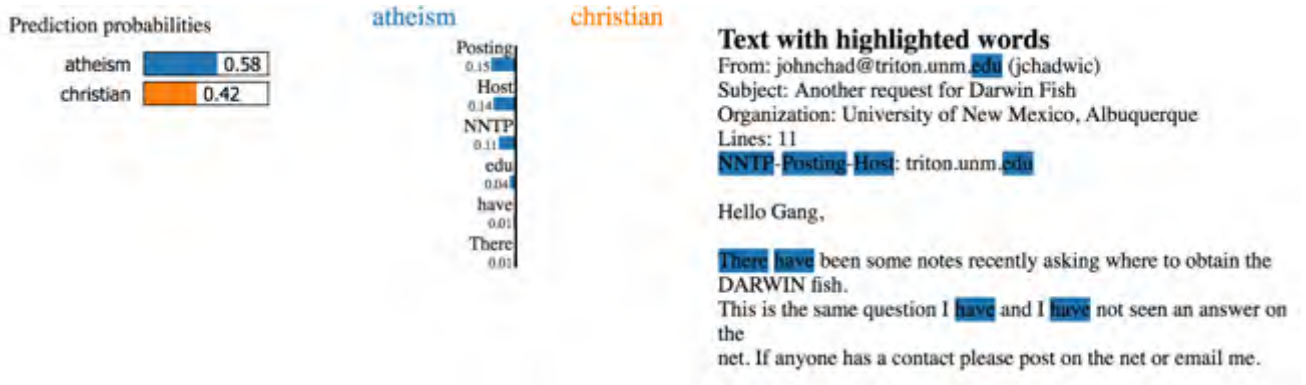
[Almost True Story]

可理解性/可调试性

- 某些业务可理解性很重要
  - 医疗、银行审计、…
  - 模型转换：Twice Learning [Zhou, 2004]
  - 解释：LIME [Ribeiro, 2016]
- 可调试性容易被忽视
  - 模型几乎不会100%正确
  - 非常复杂的特征、非常复杂的模型很难调试
  - 可调试性好可帮助分析Bad Case，提升性能

[Tensorflow DNN]

[Decision Tree]



[LIME, 2016]

总结

- 机器学习模型训练框架
  - 兼顾开发效率和运行效率
  - No Free Lunch，选择最适合问题的
  - 均衡计算资源
- 小心陷阱
  - 各方面的一致性
  - 机器学习系统面对的是开放世界
  - 依赖管理需要非常小心
  - 可理解性/可调试性很重要

谢谢大家