百度统一分布式计算框架Bigflow

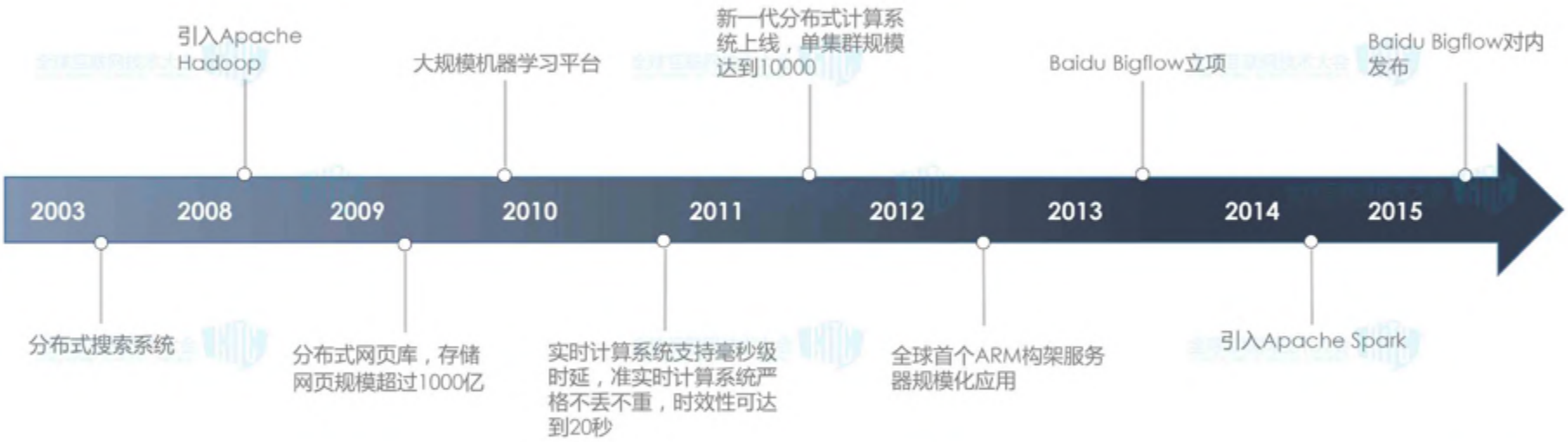
百度基础架构部　张云聪

Baidu INF

Agenda

- 背景
- 基本抽象/接口
- 统一离线/实时计算
- 优化
- 应用实践

背景


2003
2008
2009
2010
2011
2012
2013
2014
2015
分布式搜索系统
引入Apache Hadoop
分布式网页库，存储网页规模超过1000亿
大规模机器学习平台
实时计算系统支持毫秒级时延，准实时计算系统严格不丢不重，时效性可达到20秒
新一代分布式计算系统上线，单集群规模达到10000
全球首个ARM构架服务器规模化应用
Baidu Bigflow立项
引入Apache Spark
Baidu Bigflow对内发布

背景-多引擎并存

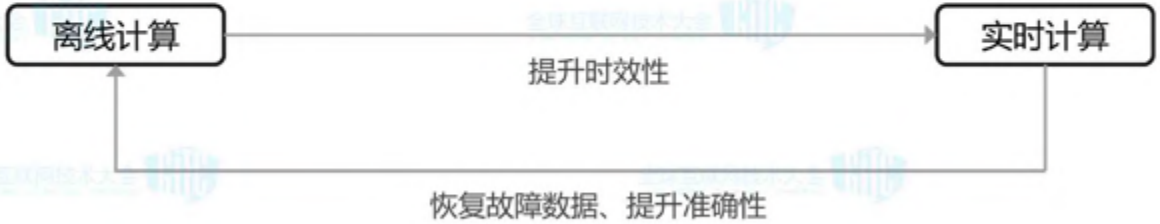
多个计算平台

开发慢 | 使用繁 | 维护难

背景-跨引擎成本高

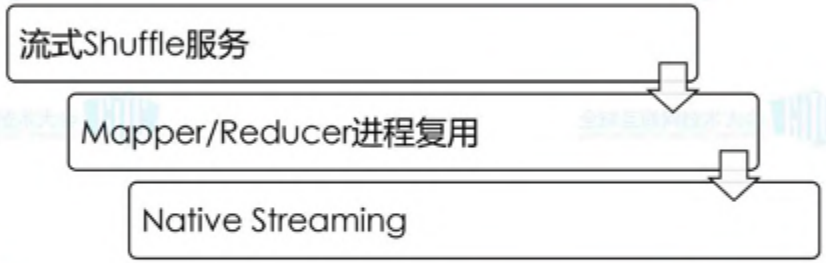
- 一套逻辑，不断重写：

- 一套逻辑，同时维护：



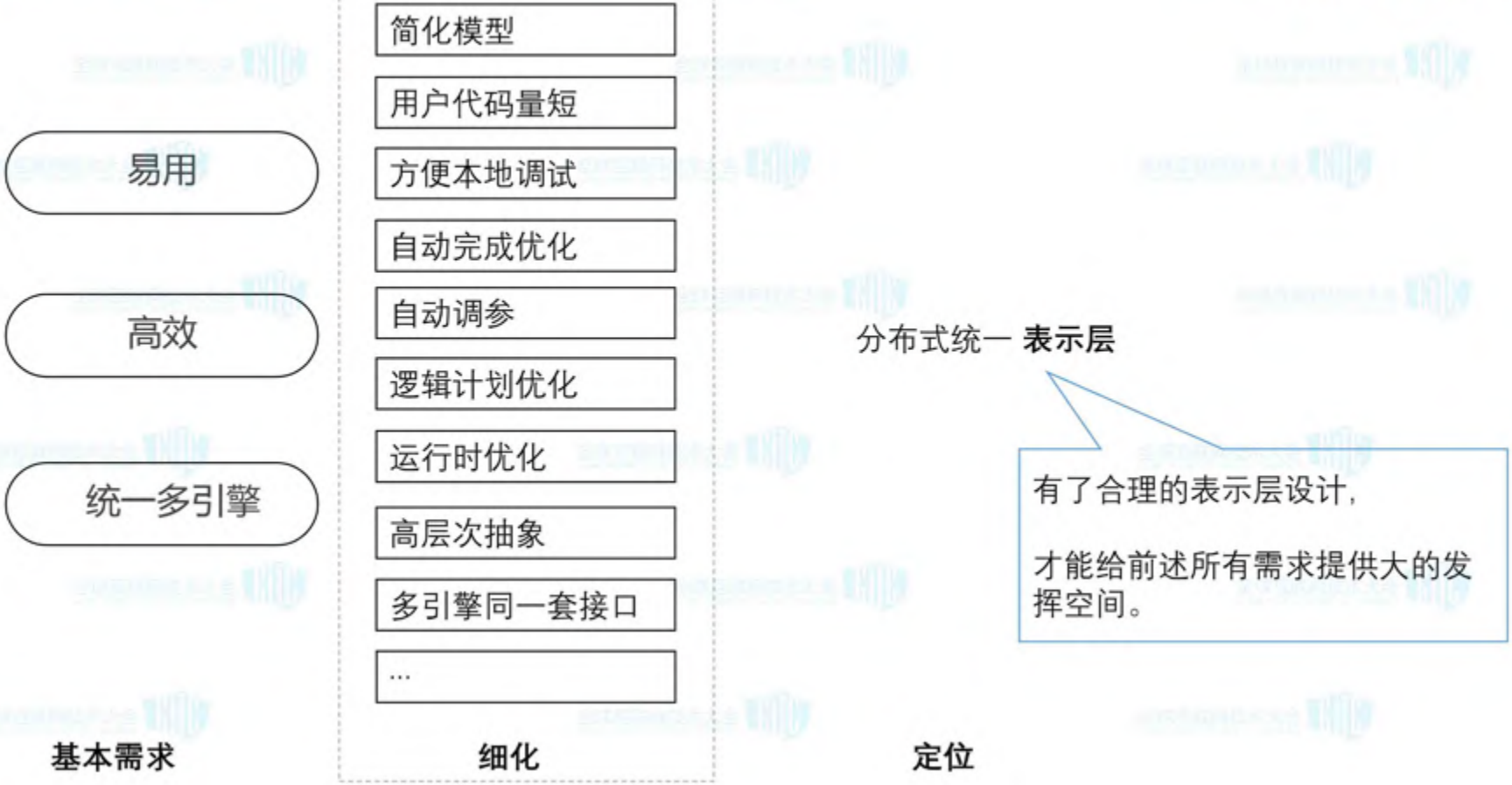
背景-升级困难

- Baidu MapReduce进化：
  - 用户无感知：



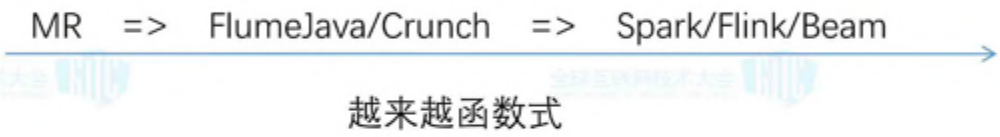
  - 用户感知，推广难度大：

项目目标


易用
高效
统一多引擎
简化模型
用户代码量短
方便本地调试
自动完成优化
自动调参
逻辑计划优化
运行时优化
高层次抽象
多引擎同一套接口
...
基本需求
细化
定位
分布式统一 表示层
有了合理的表示层设计，
才能给前述所有需求提供大的发挥空间。

指导思想

指导思想：**使分布式程序尽可能像单机程序**



函数式的分布式计算接口是社区趋势

指导思想 – 现有抽象的缺陷

指导思想：使分布式程序尽可能像单机程序。

细则一：不应暴露分布式的实现细节给用户。

细则二：所有的抽象应尽可能与单机语言中找到原型。

细则三：不应让用户去根据底层引擎特性来优化作业。

指导思想 – 现有抽象的缺陷

指导思想：使分布式程序尽可能像单机程序。

> An RDD is a read-only, partitioned collection of records.
>
> ——Spark RDD论文

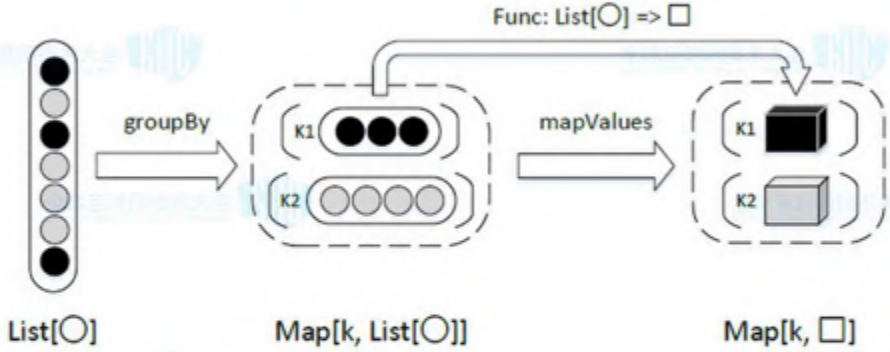
单机用户

什么是partition？

我为什么要关注数据是怎么分片的？

指导思想

指导思想：使分布式程序尽可能像单机程序



单机： groupBy + reduce

现有分布式系统： reduceByKey

为什么我不能先分组，
然后再在每个组上聚合？

单机用户

指导思想

指导思想：**使分布式程序尽可能像单机程序**

单机用户

RDD中存储的元素为什么不能是一个RDD?

[redacted]

hi，咨询一下，我现在想要解决的问题是在map里边做一个list转换为rdd，这时候要用到JavaSparkContext sc.parallelize(list)的这种操作，我把sc作为一个变量传到类里边，

```
JavaPairRDD<String, JavaRDD<LabeledPoint>> groupRdd = groupPointRdd.mapValues(new Function<Iterable<LabeledPo
        {
            private static final long serialVersionUID = 1L;

            @Override
            public JavaRDD<LabeledPoint> call(Iterable<LabeledPoint> v1) throws Exception {
                // TODO Auto-generated method stub
                List<LabeledPoint> list = new ArrayList<LabeledPoint>();
                for (LabeledPoint v : v1) {
                    list.add(v);
                }
                JavaRDD<LabeledPoint> labeledPointRdd = sc.parallelize(list);
                return labeledPointRdd;
            }
        });
```

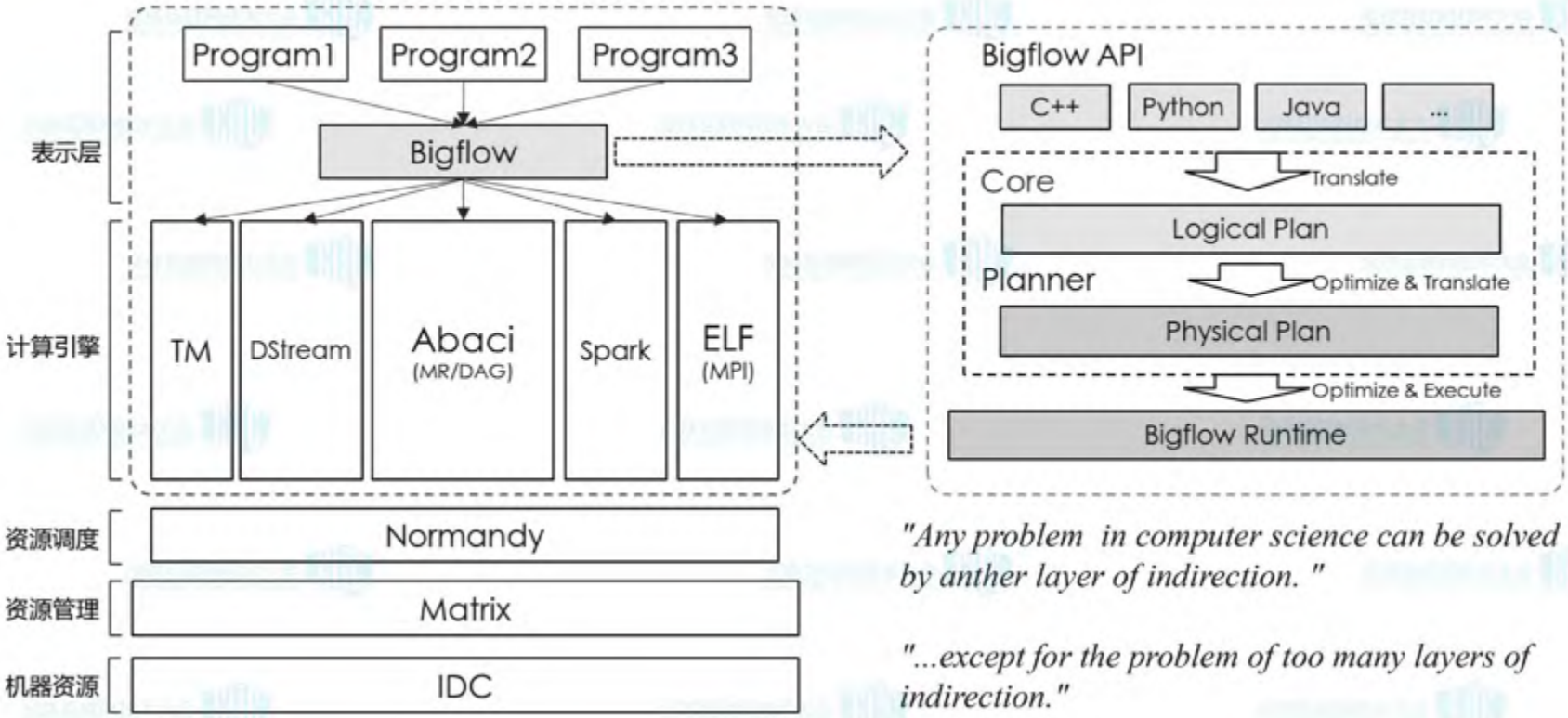
为transient，但我标的话这时候就会报sc空指针，各位老师有啥这方面的解决方案吗？ @张云聪

不支持嵌套，每个组的数据就不能被表达为RDD。
则现有分布式算法无法复用于每个分组。

指导思想

指导思想：**使分布式程序尽可能像单机程序**

Bigflow



"Any problem in computer science can be solved by anther layer of indirection. "

"...except for the problem of too many layers of indirection."

Agenda

- 背景
- 基本抽象/接口
- 统一离线/实时计算
- 优化
- 应用实践

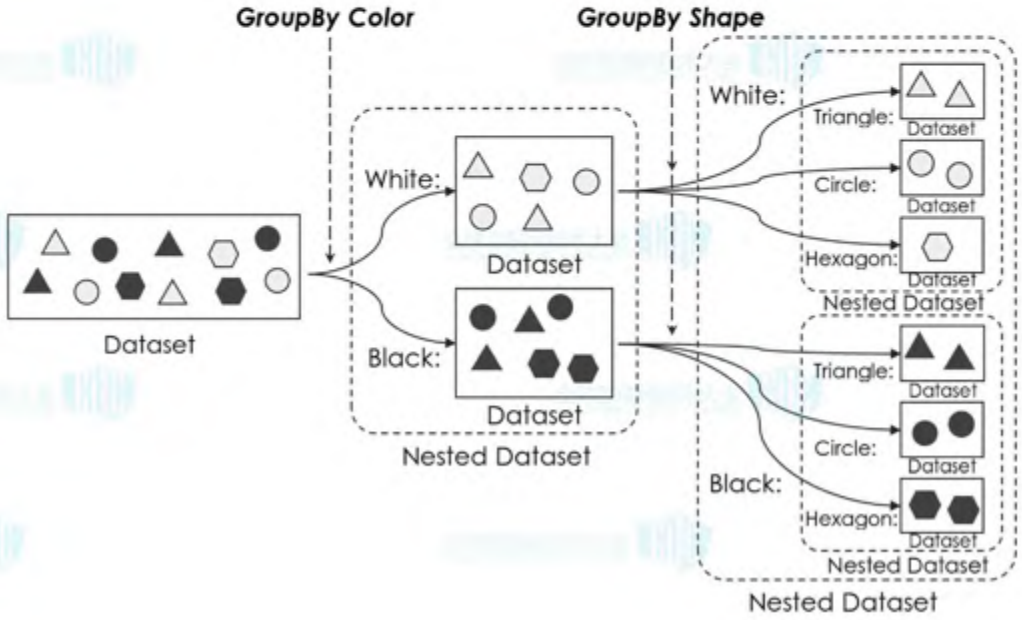
抽象—Nestable Dataset

- Pipeline: 入口 / 分布式作业的抽象
  - MRPipeline/SparkPipeline...
- Dataset: 分布式数据集抽象
  - 有序/无序 by ***Sort***
  - 单元素/任意数量元素 by ***Aggregations***
  - 有穷/无穷 from ***Source/Window***
  - 有/无Schema
  - 扁平/嵌套 by ***Groupings***
- Transformations: 变换
  - 全部惰式计算
  - ***SideInput***

```
p = base.Pipeline.create(ENGINE)

words = p.read(input.TextFile('/a/b/c')) \
    .flat_map(lambda line: line.split('\t'))

per_word_cnt = words \
    .group_by(lambda word: word) \
    .apply_values(count) \
    .flatten()

p.write(word_cnt, output.TextFile('/a/b/e'))
p.run()
```

抽象—Nestable Dataset

- DataSet嵌套分组
  - DataSet<Color, DataSet<Element>> grouped = dataset.**apply**(*GroupBy*, getColor)
  - DataSet<Color, DataSet<Shape, DataSet<Element>>> nested = grouped.**apply_values**(*GroupBy*, getShape)



接口

```
p = base.Pipeline.create(ENGINE)

visitors = p.read(input.TextFile('/a/b/c')) \
    .flat_map(lambda line: line.split('\t'))

def cnt_distinct(visitors):
  ''' @param visitors: DataSet<String> '''
  return visitors \
    .distinct() \
    .count()
num = visitors.apply(cnt_distinct)

p.write(num, output.TextFile('/a/b/d'))
p.run()
```

```
total_visitors = ...

def uv_counting(total_visitors):
  ''' @param page_visitors: DataSet<Pair<String, String>> '''
  return total_visitors \
    .group_by(lambda (visitors, page): page) \
    .apply_values(cnt_distinct) \
    .flatten()

pv_result = total_visitors.apply(uv_counting)

p.write(num, output.TextFile('/a/b/d'))
p.run()
```

变换

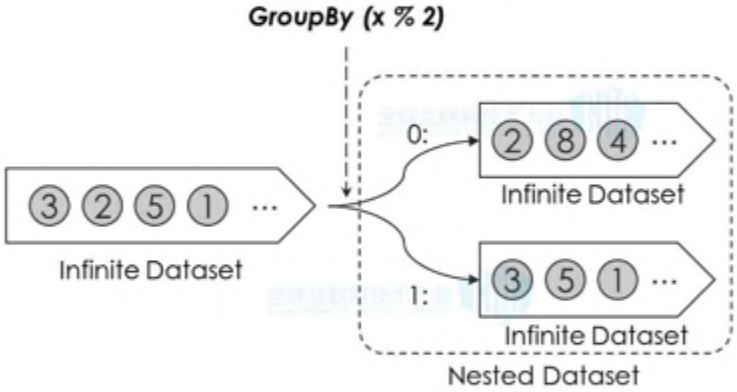
Transformation	UDF	Input => output
map(p, f)	*f: (V) => T*	*[V] => [T]*
filter(p, f)	*f: (V) => bool*	*[V] => [V]*
flat_map(p, f)	*f: (V) => Iterable[T]*	*[V] => [T] / V => [T]*
group_by_key(p)	-	*[(K, V)] => {K: [V]}*
group_by(p, f)	*f: (V) => K*	*[V] => {K: [V]}*
flatten(p)	-	*{K: [V]} => [(K, V)]*
reduce(p, f)	*f: (V, V)=>V*	*[V] => V*
aggregate(p, v, f1, f2)	*v: T or () => T* *f1: (T, V) => T* *f2: (T, T) => T*	*[V] => T*
join(p1, p2)	-	*[(K, V1)], [(K, V2)] => [(K, (V1, V2))]*
cogroup(p1, p2)	-	*[(K, V1)], [(K, V2)] => {K: ([V1], [V2])}]*
cartesian(p1, p2)	-	*[V1], [V2] => [(V1, V2)]*
union(p1, p2)	-	*[V], [V] => [V]*
sum(p)	-	*[V] => V*
count(p)	-	*[V] => Long*
take(p, num)	-	*[V] => [V]*
sort(p, f)	*f: f(V) => K*	*[V] => [V]*

Agenda

- 背景
- 基本抽象/接口
- 统一离线/实时计算
- 优化
- 应用实践

抽象—Nestable Dataset

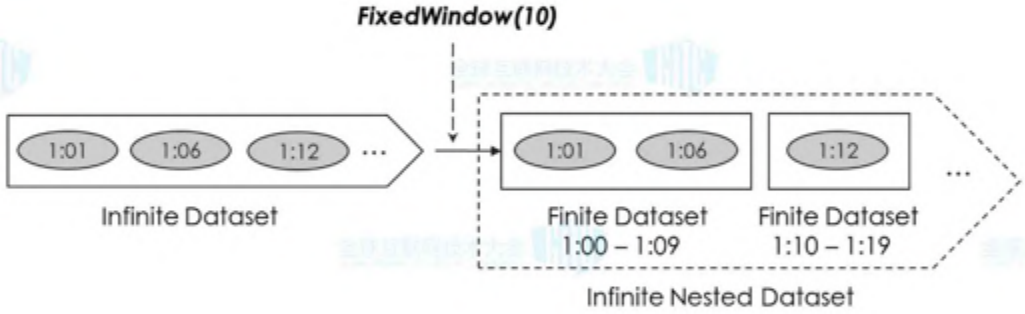
- 无穷DataSet分组



- DataSet<Odevity, DataSet<Int>> grouped = dataset.**apply**(*GroupBy*, modByTwo)

抽象—Nestable Dataset

- 无穷DataSet划分窗口



➢ DataSet<Window, DataSet<Log>> windowed = dataset.**apply**(*WindowInto*, FIXED_WINDOW, getTime)

离线/实时

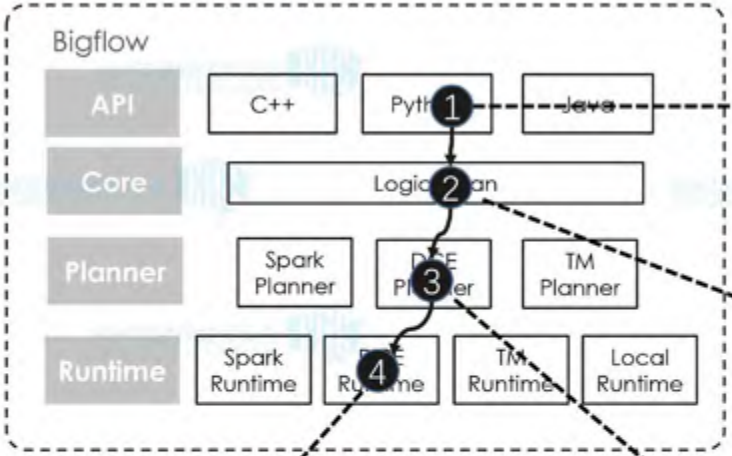
```
total_visitors = ...

def uv_counting(total_visitors):
   return total_visitors \
      .group_by(lambda (visitors, page): page) \
       .apply_values(cnt_distinct)
      .flatten()

pv_result = total_visitors.apply(uv_counting)

p.write(num, output.TextFile('/a/b/d'))
p.run()
```

```
def uv_counting(total_visitors):
   return total_visitors \
      .group_by(lambda (visitors, page): page) \
       .apply_values(cnt_distinct)
      .flatten()

raw_words = p.read(input.Kafka(xxx)) \
      .flat_map(lambda line: line.split())

num = words \
      .window_into(time_extractor, FIXED_WINDOW) \
         .apply_values(uv_counting) \
      .flatten()
p.write(num, ...)
p.run()
```

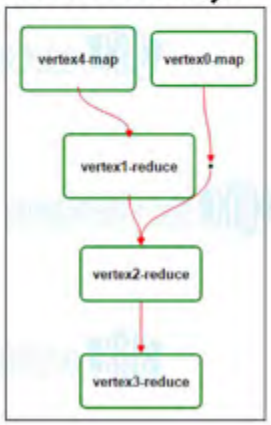
Agenda

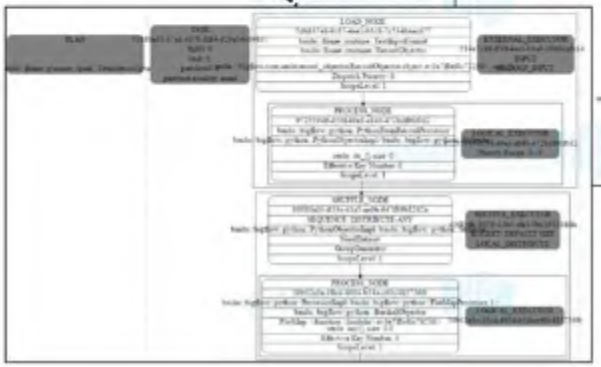
- 背景
- 基本抽象/接口
- 统一离线/实时计算
- 优化
- 应用实践

优化-基本框架

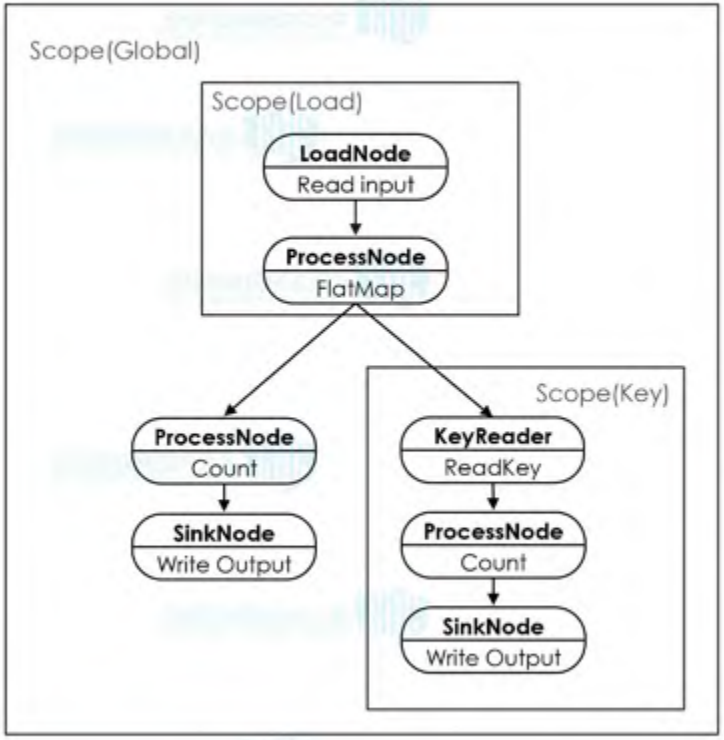


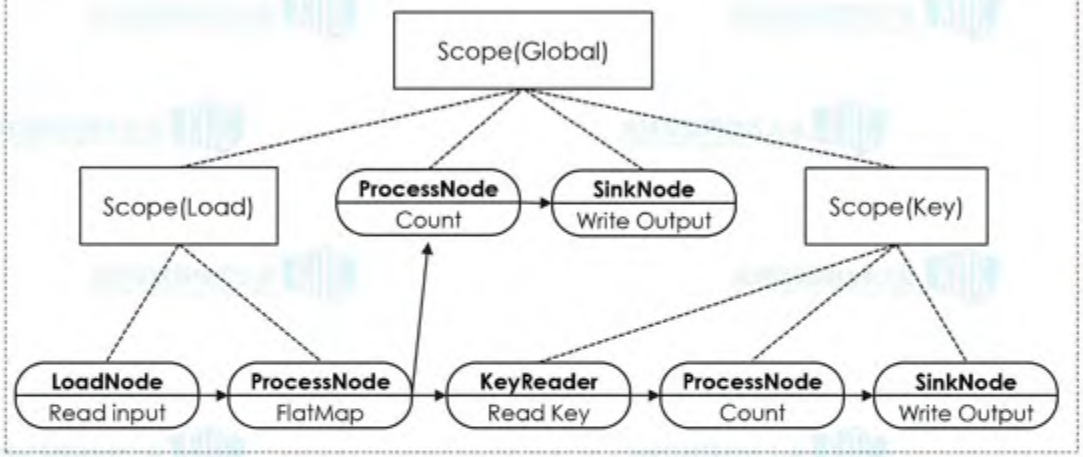

```
p = base.Pipeline.create(ENGINE)

words = p.read(input.TextFile('/a/b/c')) \
        .flat_map(lambda line: line.split('\t'))

def cnt_distinct(words):
    """ @param words: DataSet<String> """
    return words \
        .distinct() \
        .count()
num = user_words.apply(cnt_distinct)
```





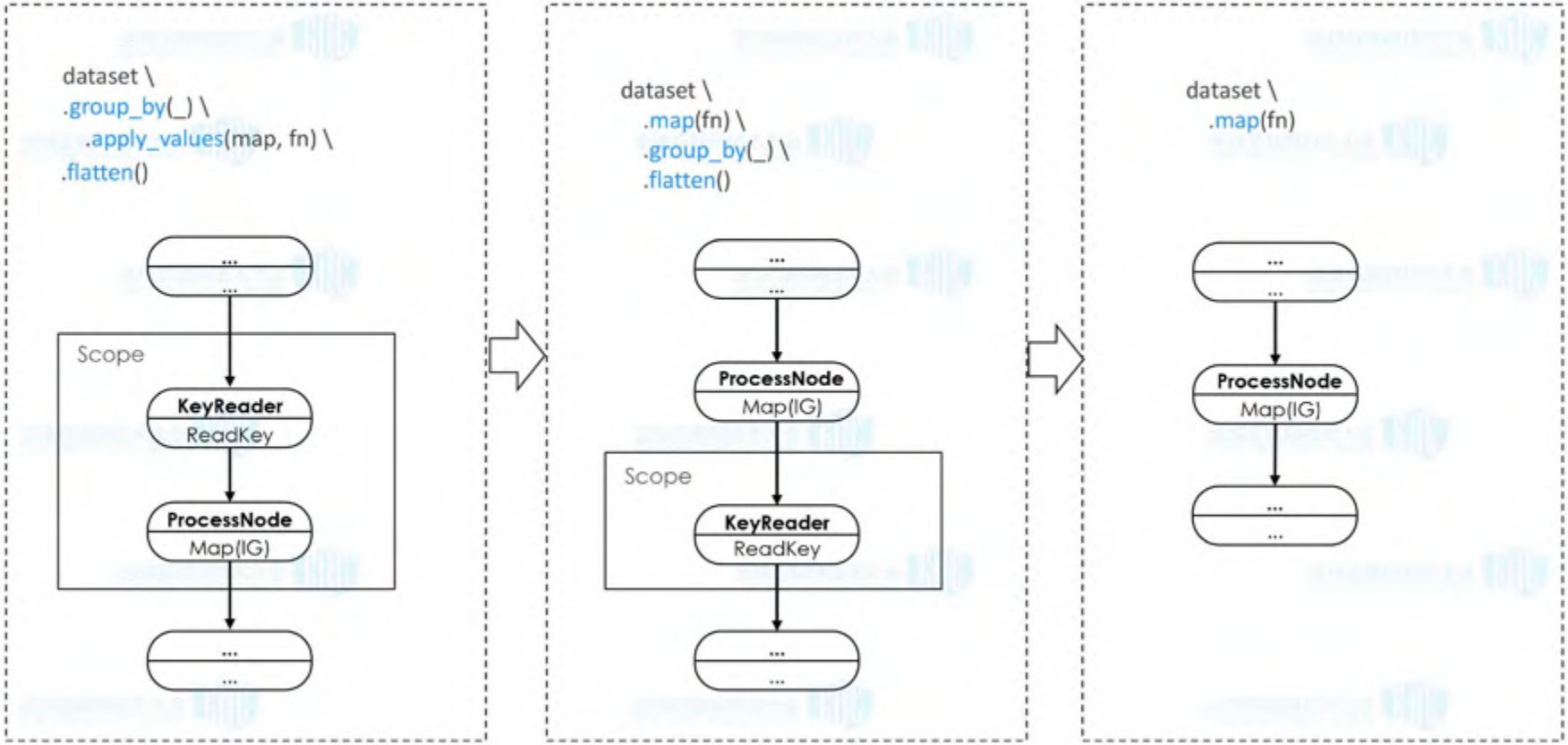
优化-基本框架





- Node: “计算” (DAG)
- Scope: “分布式” (Tree)

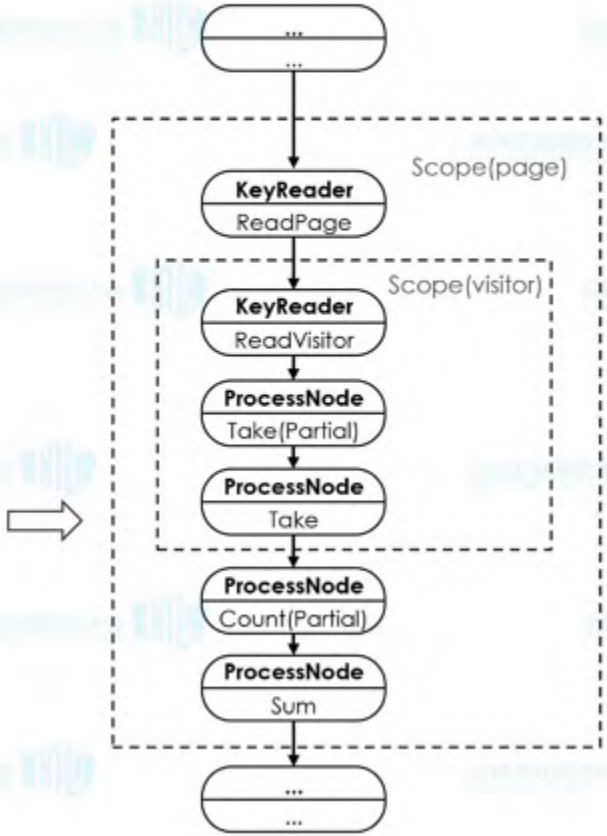
优化-示例(逻辑计划)


dataset \
.group_by(_) \
.apply_values(map, fn) \
.flatten()
...
...
Scope
KeyReader
ReadKey
ProcessNode
Map(IG)
...
...
dataset \
.map(fn) \
.group_by(_) \
.flatten()
...
...
ProcessNode
Map(IG)
Scope
KeyReader
ReadKey
...
...
dataset \
.map(fn)
...
...
ProcessNode
Map(IG)
...
...

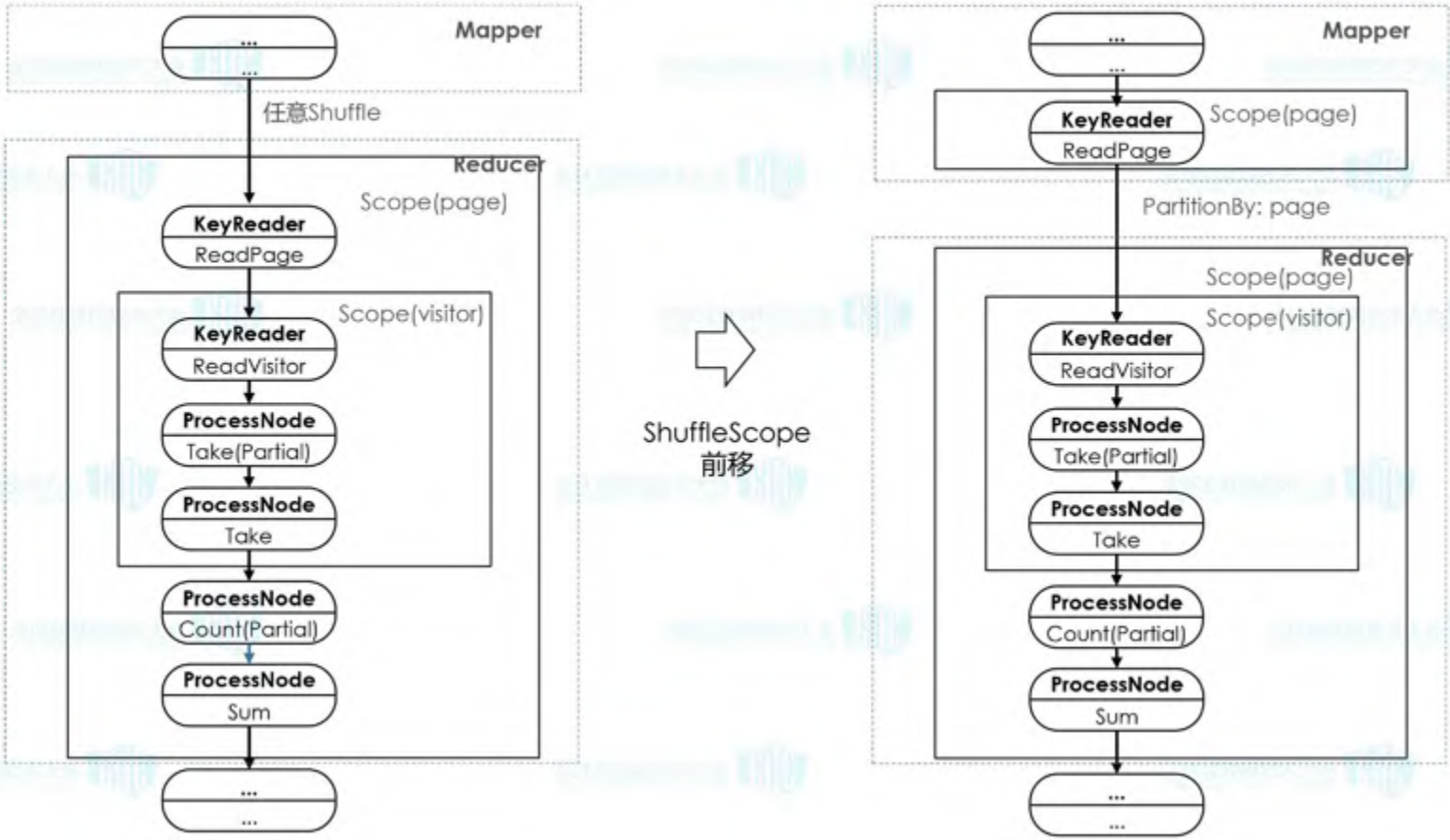
优化-示例(物理计划)

```
dataset \
 .group_by(lambda (visitors, page) : page) \
  .apply_values(cnt_distinct) \
 .flatten()
```

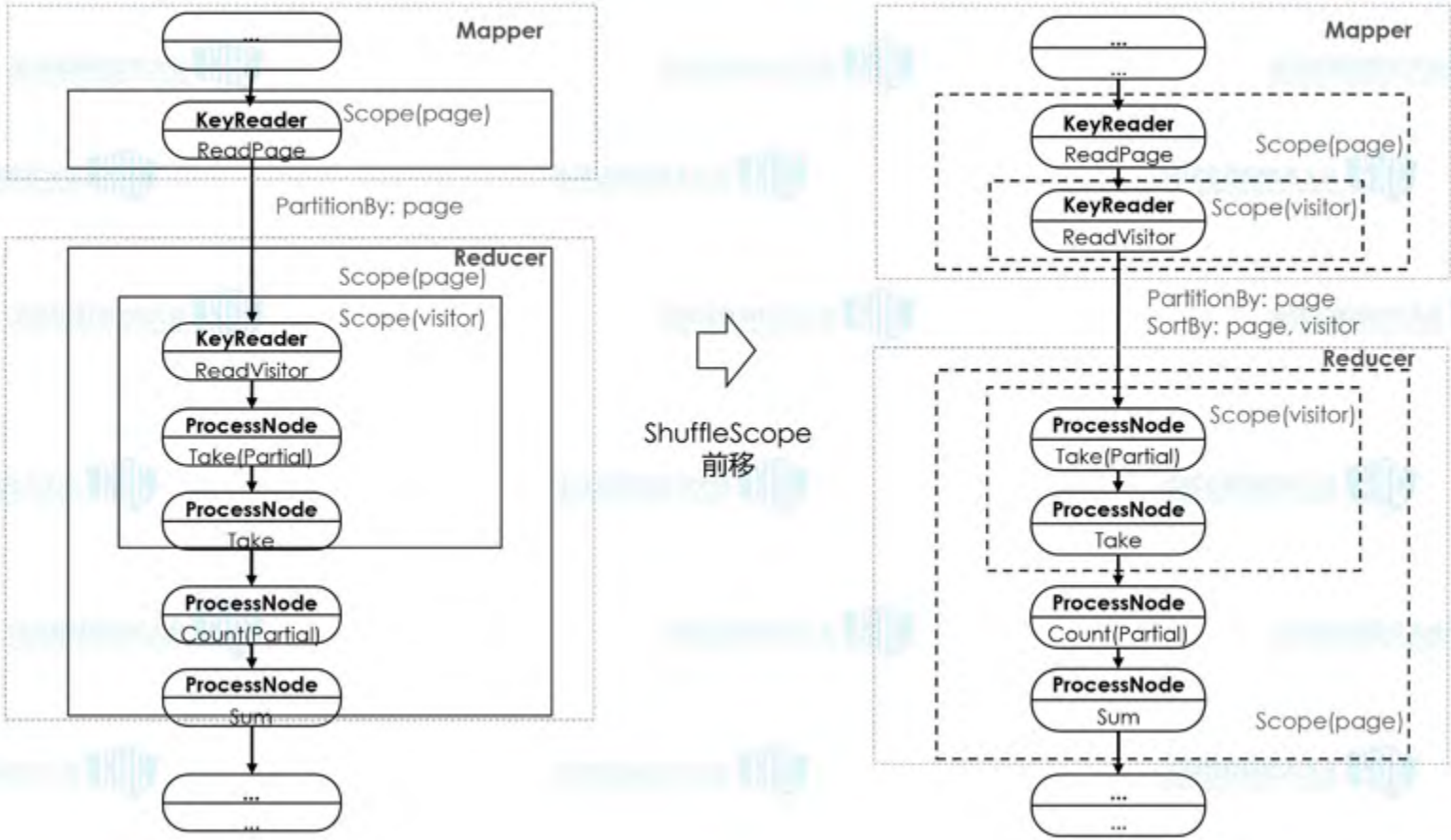
```
dataset \
 .group_by(lambda (visitors, page): page) \
  .apply_values(lambda visitors: visitors \
                    .group_by(visitor) \
                      .apply_values(take, 1)
                    .flatten() \
                    .count())
  .flatten()
```



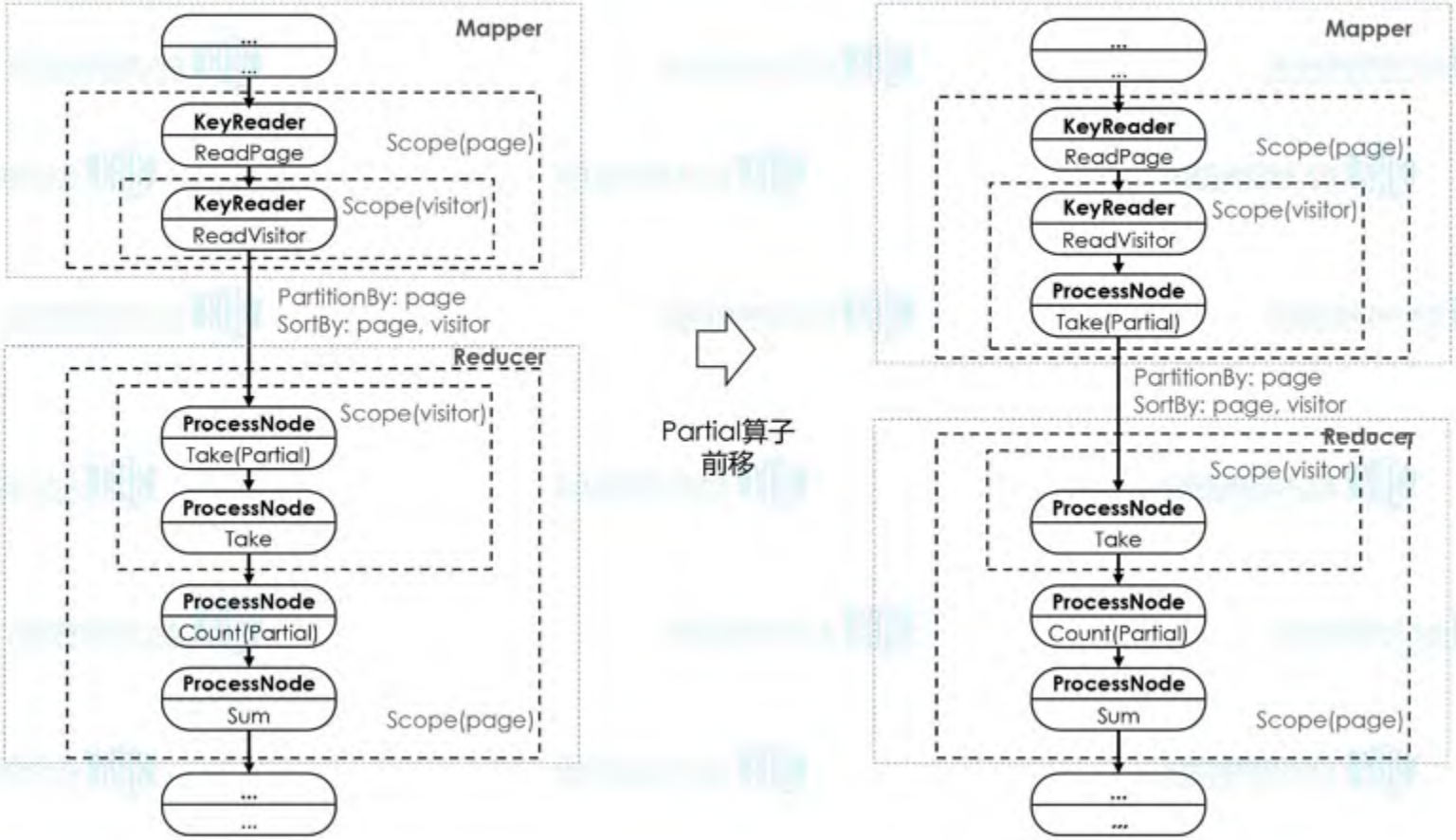
优化-示例(物理计划)


Mapper
...
...
任意Shuffle
Reducer
Scope(page)
KeyReader
ReadPage
Scope(visitor)
KeyReader
ReadVisitor
ProcessNode
Take(Partial)
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
...
...
ShuffleScope
前移
Mapper
...
...
KeyReader
ReadPage
Scope(page)
PartitionBy: page
Reducer
Scope(page)
Scope(visitor)
KeyReader
ReadVisitor
ProcessNode
Take(Partial)
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
...
...

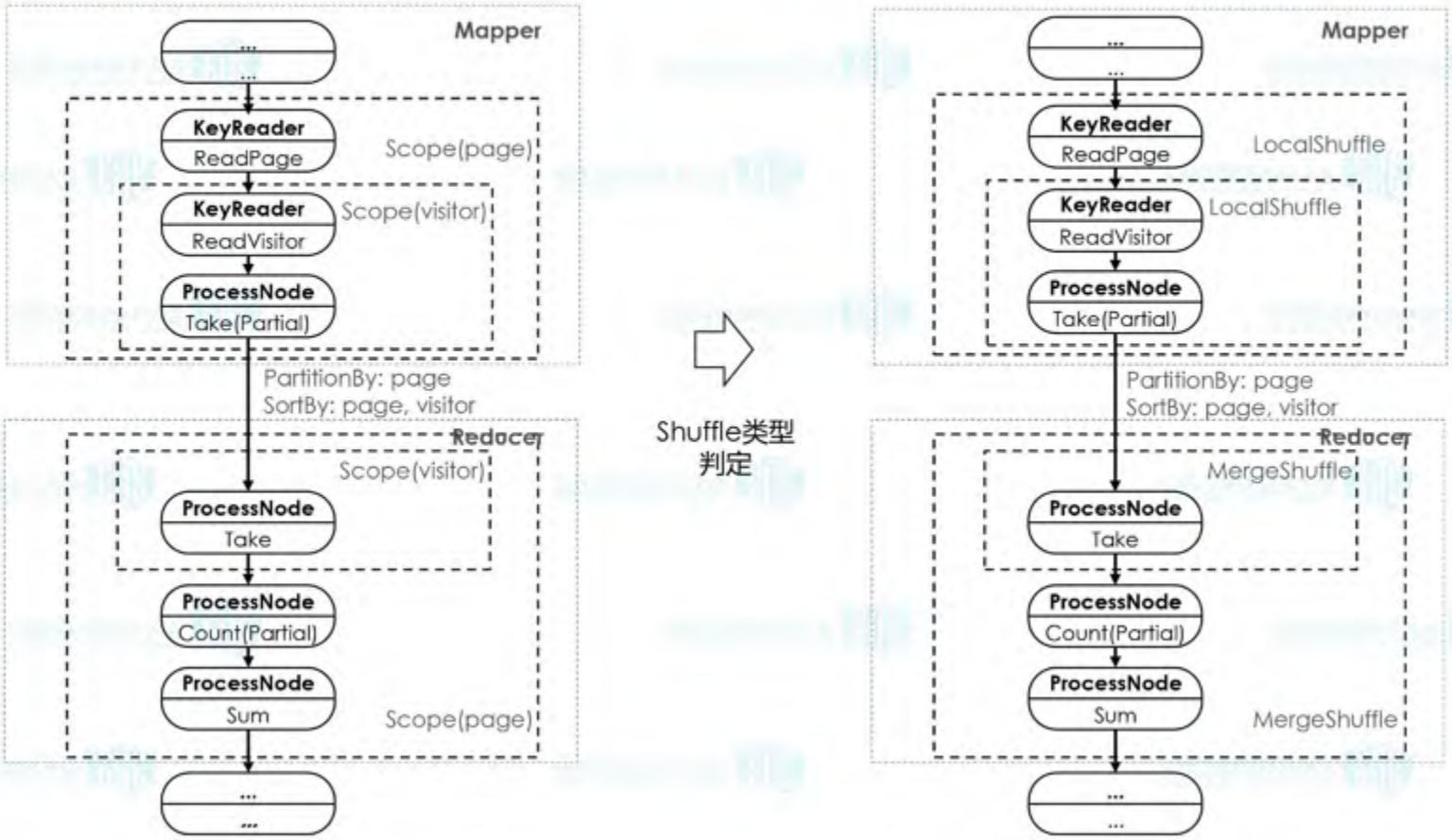
优化-示例(物理计划)


Mapper
...
KeyReader
ReadPage
Scope(page)
PartitionBy: page
Reducer
Scope(page)
Scope(visitor)
KeyReader
ReadVisitor
ProcessNode
Take(Partial)
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
...
ShuffleScope
前移
Mapper
...
KeyReader
ReadPage
Scope(page)
KeyReader
ReadVisitor
Scope(visitor)
PartitionBy: page
SortBy: page, visitor
Reducer
ProcessNode
Take(Partial)
Scope(visitor)
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
Scope(page)
...

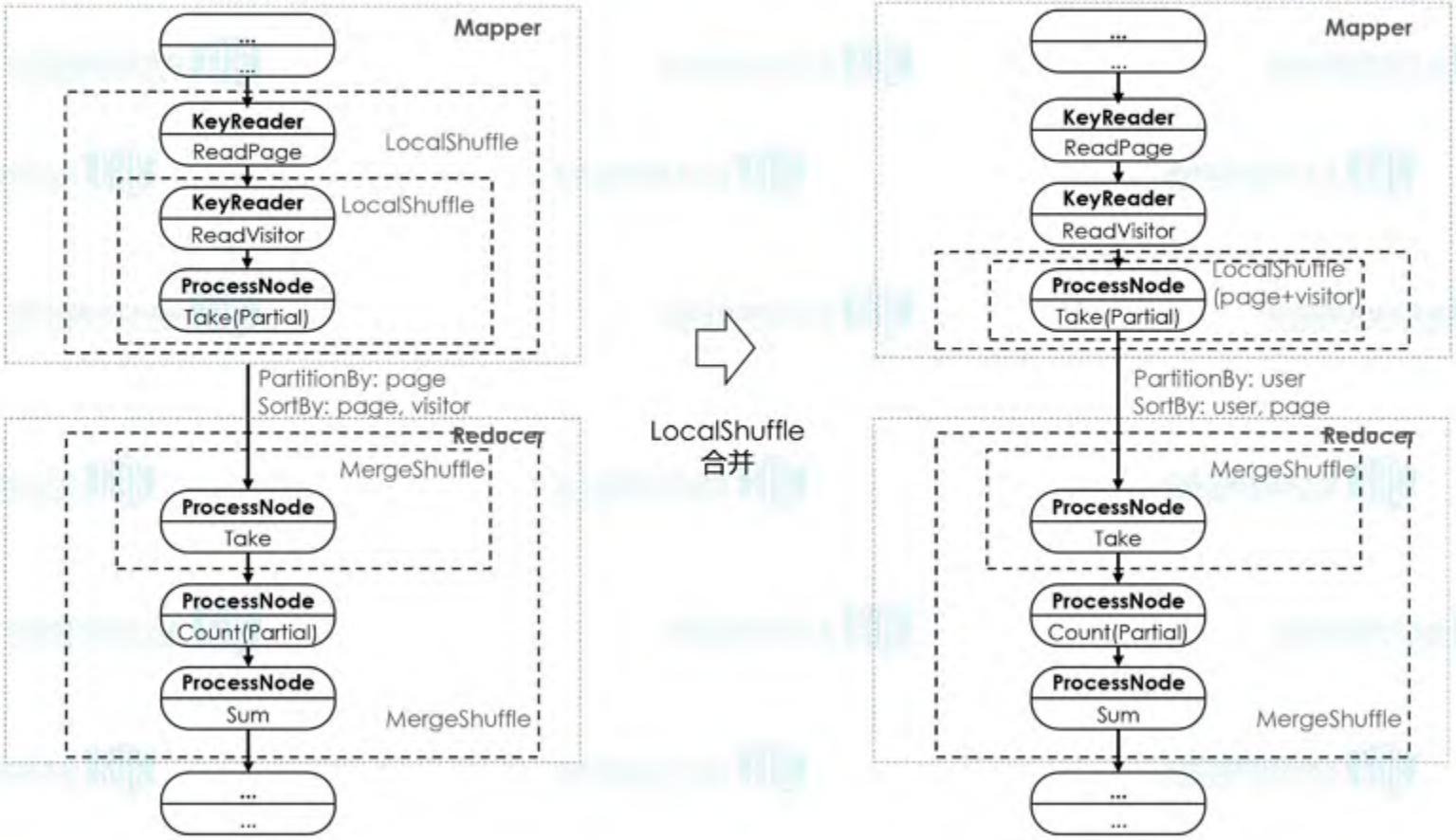
优化-示例(物理计划)


Mapper
...
KeyReader
ReadPage
Scope(page)
KeyReader
ReadVisitor
Scope(visitor)
PartitionBy: page
SortBy: page, visitor
Reducer
ProcessNode
Take(Partial)
Scope(visitor)
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
Scope(page)
...
Partial算子
前移
Mapper
...
KeyReader
ReadPage
Scope(page)
KeyReader
ReadVisitor
Scope(visitor)
ProcessNode
Take(Partial)
PartitionBy: page
SortBy: page, visitor
Reducer
Scope(visitor)
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
Scope(page)
...

优化-示例(物理计划)


Mapper
...
...
KeyReader
ReadPage
Scope(page)
KeyReader
ReadVisitor
Scope(visitor)
ProcessNode
Take(Partial)
PartitionBy: page
SortBy: page, visitor
Reducer
Scope(visitor)
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
Scope(page)
...
...
Shuffle类型
判定
Mapper
...
...
KeyReader
ReadPage
LocalShuffle
KeyReader
ReadVisitor
LocalShuffle
ProcessNode
Take(Partial)
PartitionBy: page
SortBy: page, visitor
Reducer
MergeShuffle
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
MergeShuffle
...
...

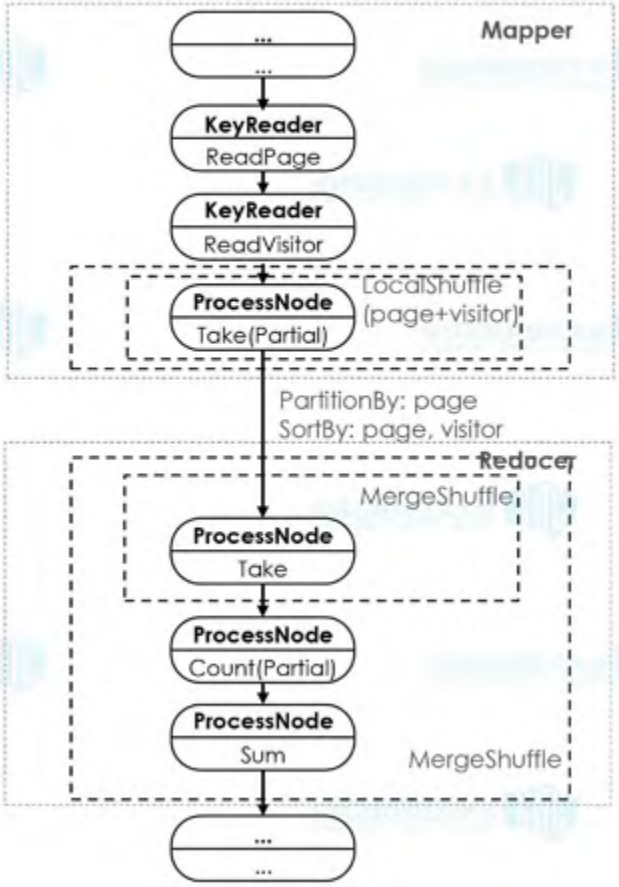
优化-示例(物理计划)


Mapper
...
...
KeyReader
ReadPage
LocalShuffle
KeyReader
ReadVisitor
LocalShuffle
ProcessNode
Take(Partial)
PartitionBy: page
SortBy: page, visitor
Reducer
MergeShuffle
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
MergeShuffle
...
...
LocalShuffle
合并
Mapper
...
...
KeyReader
ReadPage
KeyReader
ReadVisitor
ProcessNode
Take(Partial)
LocalShuffle
(page+visitor)
PartitionBy: user
SortBy: user, page
Reducer
MergeShuffle
ProcessNode
Take
ProcessNode
Count(Partial)
ProcessNode
Sum
MergeShuffle
...
...

优化-示例(物理计划)

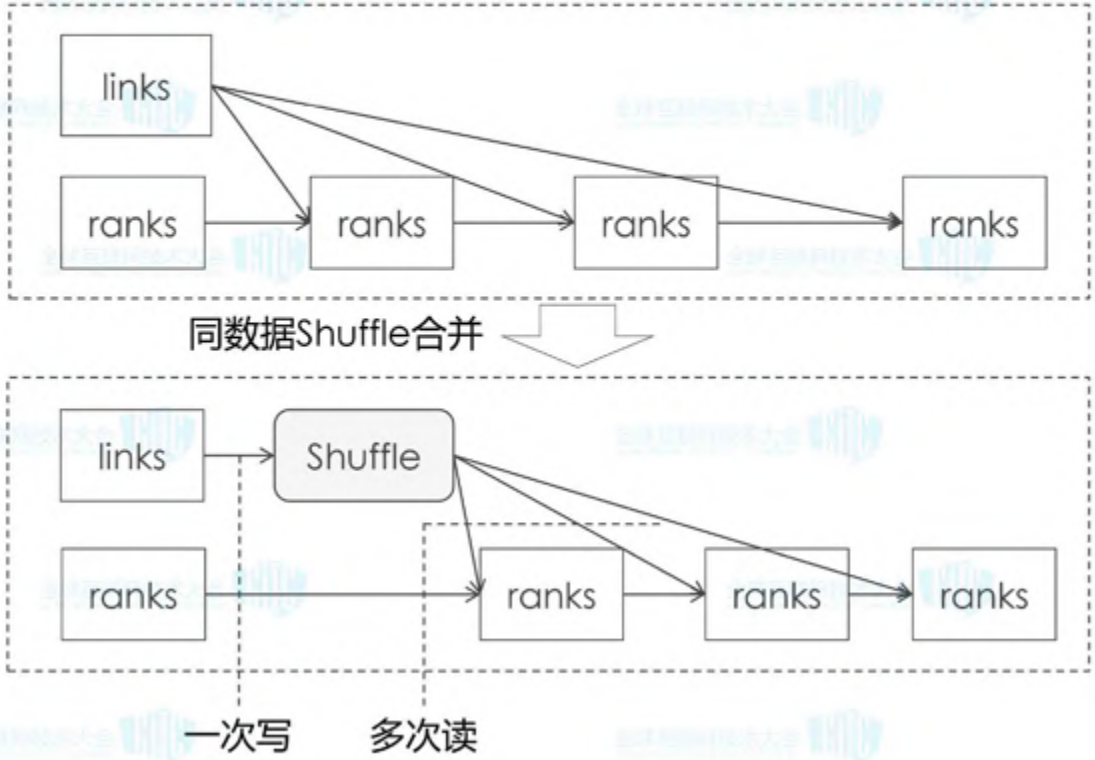
```
dataset \
 .group_by(lambda (visitors, page) : page) \
  .apply_values(cnt_distinct) \
 .flatten()
```

```
dataset \
  .group_by(lambda (visitors, page): page) \
    .apply_values(lambda visitors: visitors \
                        .group_by(visitor) \
                          .apply_values(take, 1)
                        .flatten() \
                        .count())
    .flatten()
```



优化-示例(结合引擎)

- 示例: PageRank



Agenda

- 背景
- 基本抽象/接口
- 统一离线/实时计算
- 优化
- 应用实践

应用实践-收益

日运行作业数3K+

日处理数据量3P+

在百度内部应用于上百产品线

数百活跃用户

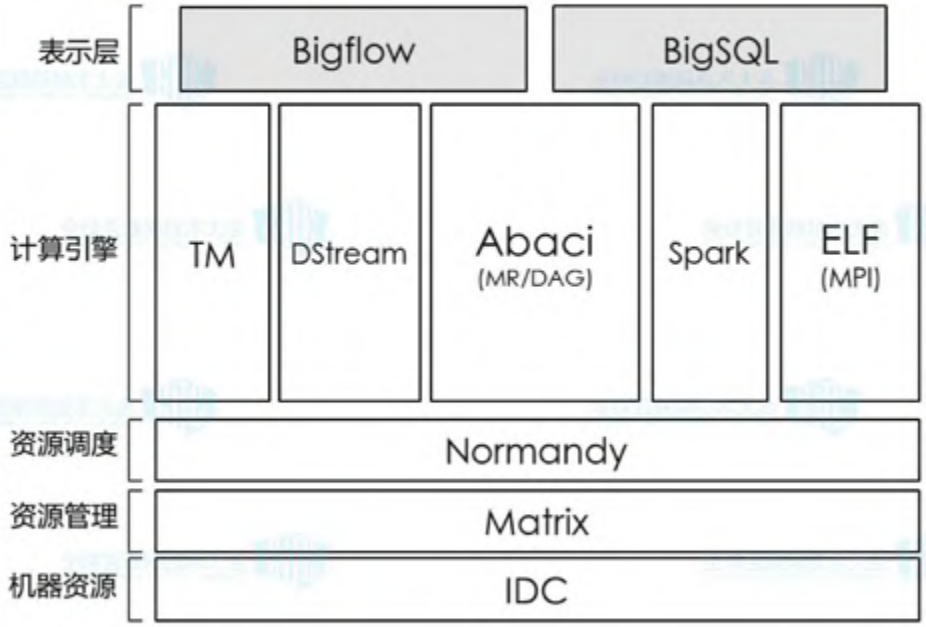
Bigflow作业占比持续稳步提升

应用实践-收益

- 性能提升
  - C++ vs Hadoop Streaming
  - 算子前移: filter/select/初段aggregate
  - Python解释器复用，相同字典只加载一次
- 多引擎支持
  - Hadoop MapReduce -> DAG MapReduce -> Spark
- 实际性能对比结果
  - 单个作业平均20%，DAG作业提升50

应用实践-总结

- 100+ 业务线，300+ 用户
- 旧作业改写：
  - 30%代码量减少
  - 50%性能提升



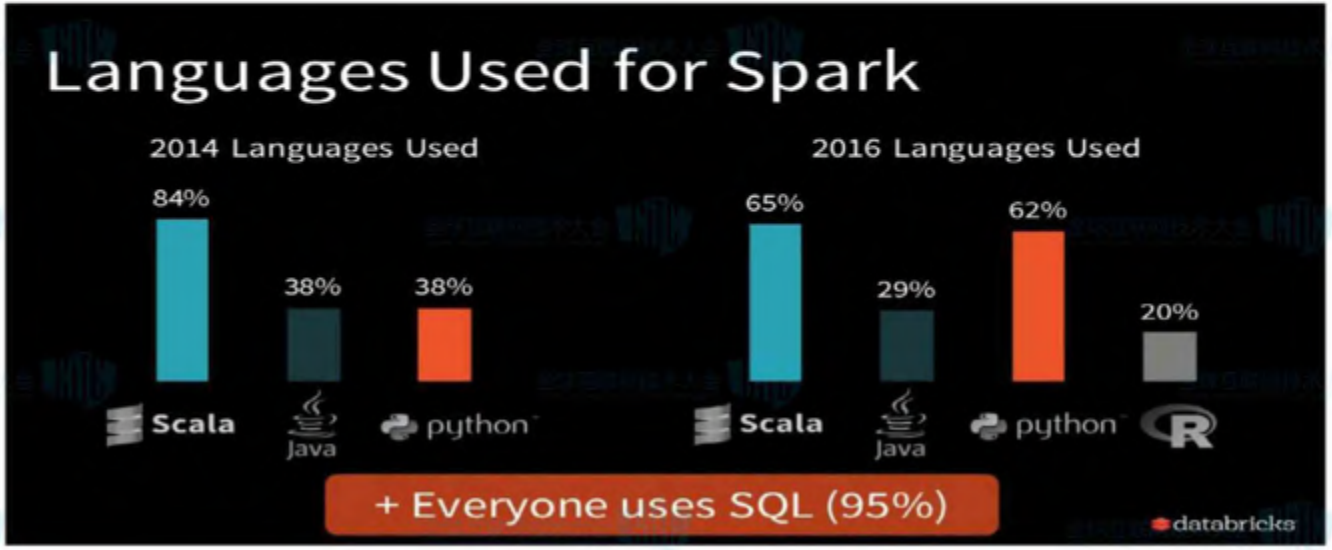
INF-DC Stack

Bigflow开源计划

目标：

2017 Q3 开源Bigflow on Spark

Python接口性能可大大高于PySpark



THANK YOU

Baidu Infrastructure