

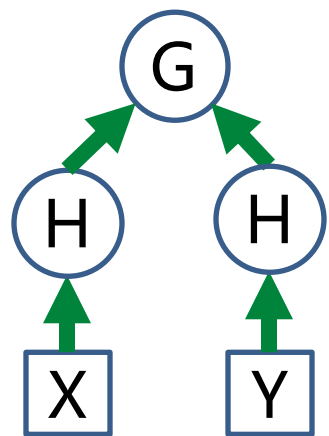
图执行引擎与代码组织

黎江 (lijiang01@baidu.com)

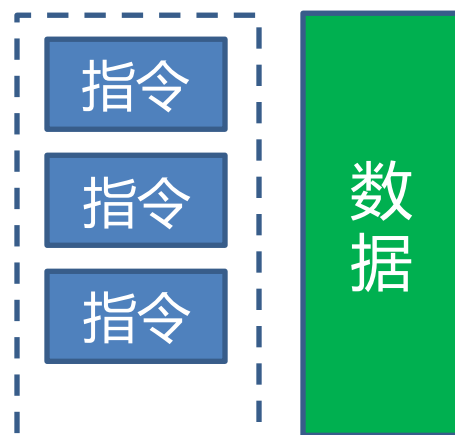
图执行引擎：先来谈一谈函数编程

- 通过**组装**和**应用**函数来构建应用的编程方法
 - 树状组装：通过将子函数组织成**树**，来组装定义一个新函数
 - 无副作用：函数应用在同样的输入上，**恒定**产出同样的结果

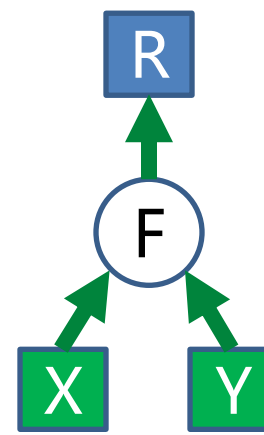
$$F(X, Y) = G(H(X), H(Y))$$



$$F(X, Y) = \{...\}$$



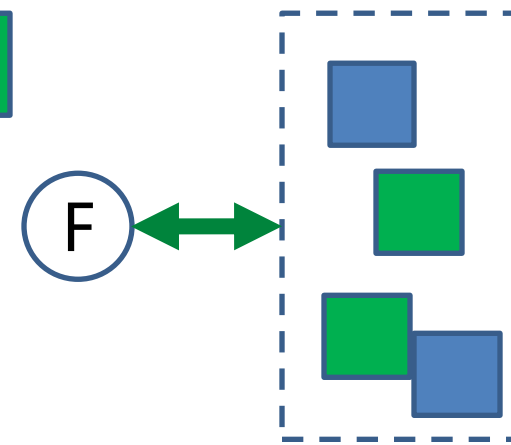
无副作用



可写

可读

有副作用



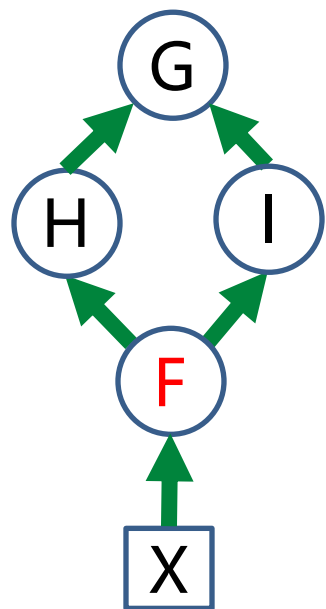
图执行引擎：函数编程又有什么优势？

- 天然可并发

- 并发的难点在竞争
- 纯函数编程是无竞争的

$T = F(X)$

$G(H(T), I(T))$

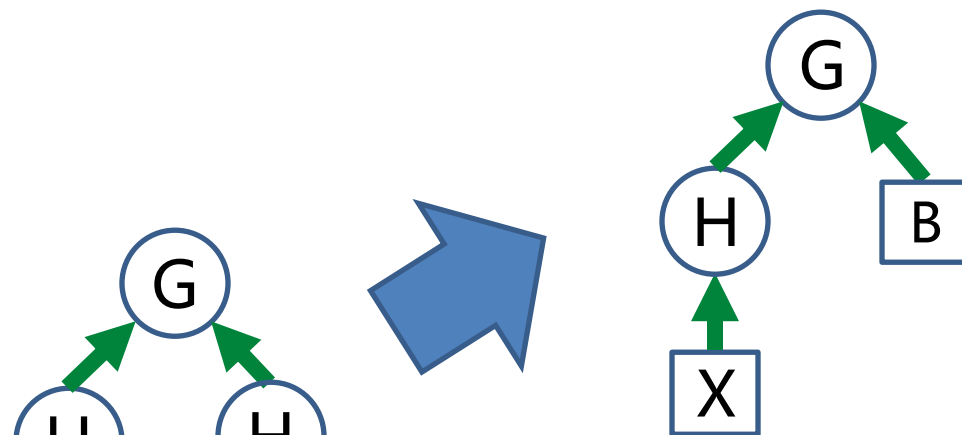


INVOKE F, X -> T
WAIT T
INVOKE H, T -> _T1
INVOKE I, T -> _T2
WAIT _T1, _T2
INVOKE G, _T1, _T2 -> R
WAIT R

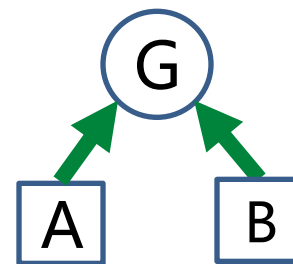
- 执行可分拆

- 输出一样，那么就是一样的

SPARK RDD快速恢复



局部函数可单测



图执行引擎：近在身边的函数编程



- 并行编译

- 使用者描述依赖关系 (a.o: a.cc common.h)
- 使用者实现函数功能 (g++ -c a.cc -o a.o)
- 使用make exe -jn运行，自然完成无依赖并发

- 增量编译

- 进行部分修改 (a.cc)
- 使用make，基于不可变性跳过部分执行 (b.o)

```
exe: a.o
    g++ a.o b.o -o exe

a.o: a.cc common.h
    g++ -c a.cc -o a.o

b.o: b.cc common.h
    g++ -c b.cc -o b.o
```

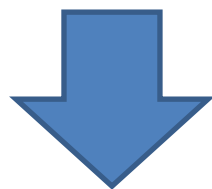
图执行引擎：一个C++函数编程框架



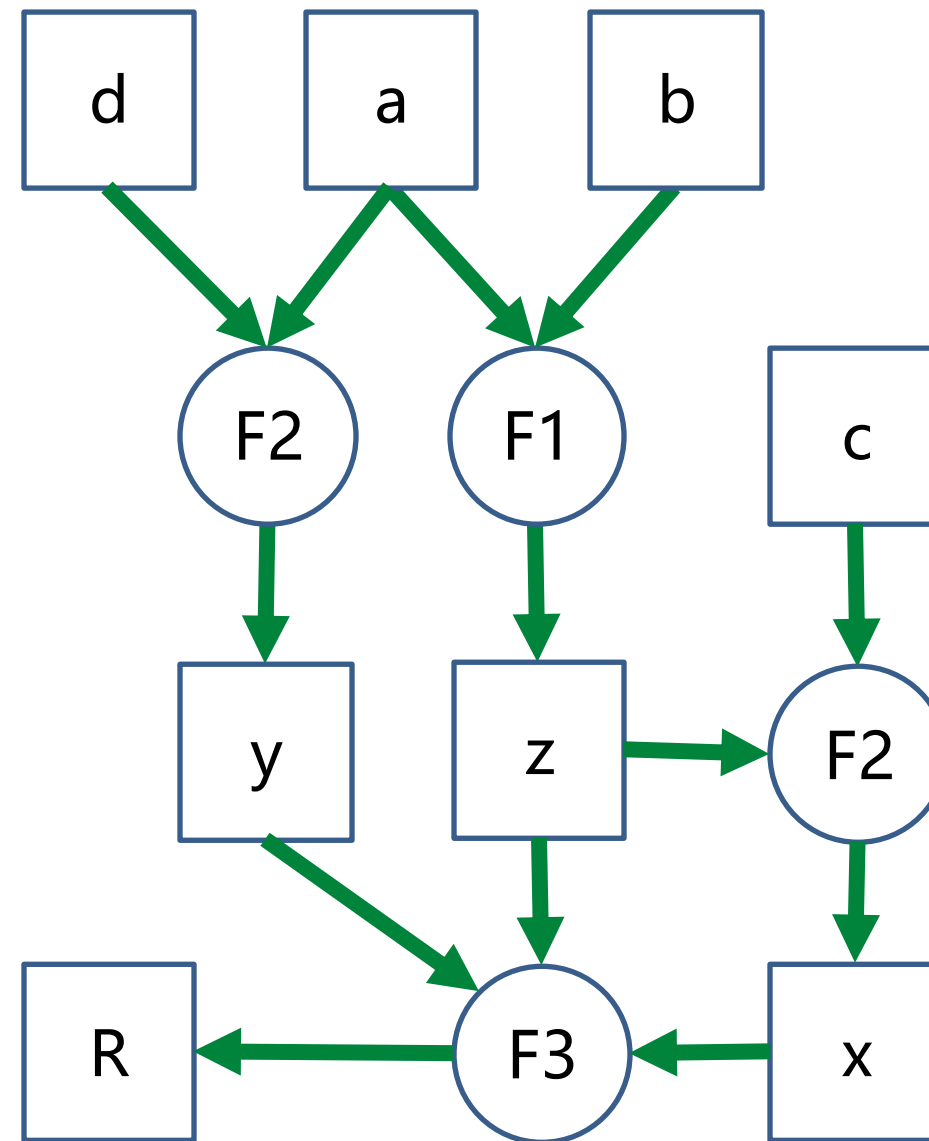
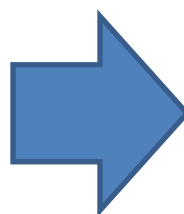
- 项目地址： baidu/haokan-rec/graph-engine
- 整体思路：
 - 使用C++原生指令编程开发基础函数，控制函数间无副作用影响
 - 框架通过组图API的方式采用函数编程语义完成函数组装
 - 采用多种优化手段规避函数编程地不可变性带来地性能损耗
- 预期：
 - IO和复杂计算逻辑交给指令编程模式开发，保证实现效率
 - 函数间组装和应用关系按照函数编程实现，保证并发和隔离

图执行引擎：用一张图来表达函数编程

- $F(a, b, c, d) = F3(F2(F1(a, b), c), F1(a, b), F2(a, d))$

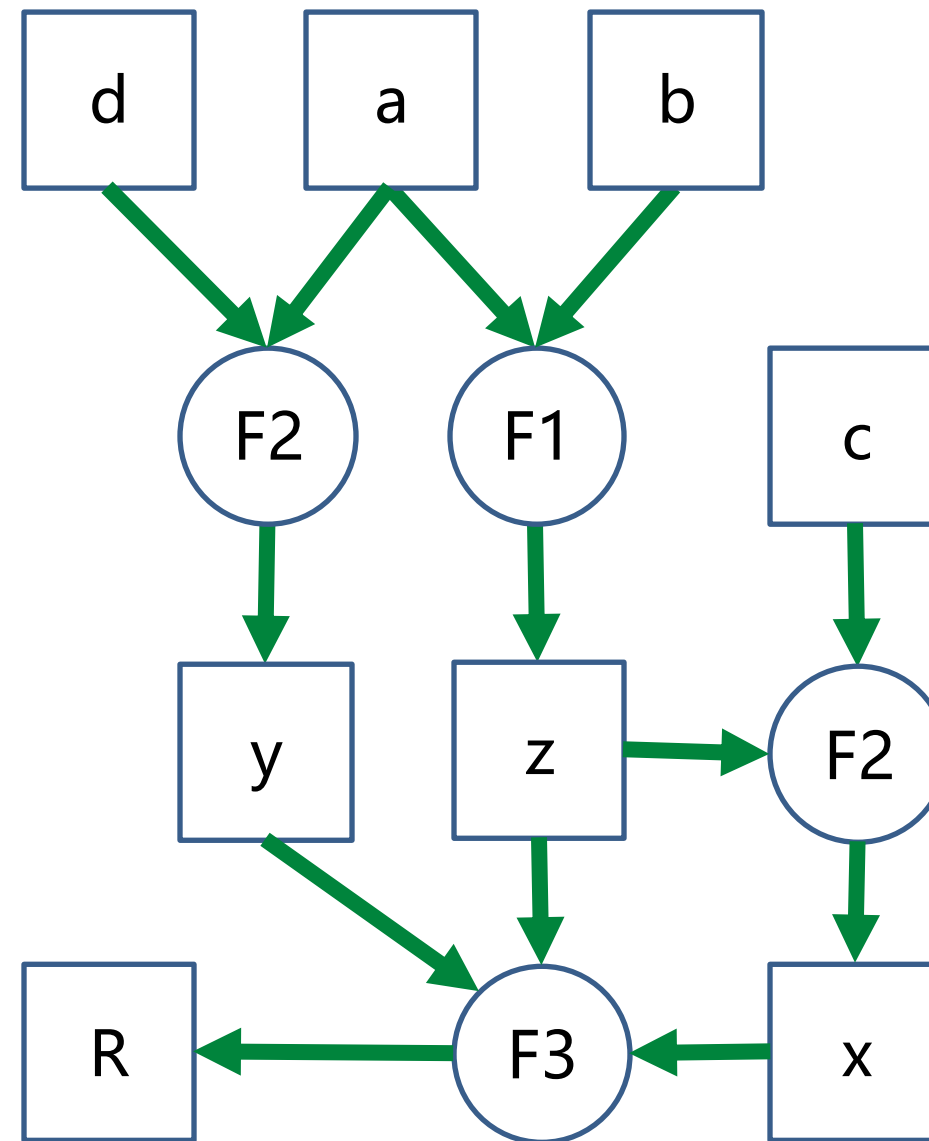


- $z = F1(a, b)$
- $y = F2(a, d)$
- $x = F2(z, c)$
- $R = F3(x, z, y)$



图执行引擎：用一张图来表达函数编程

- 函数
 - -> 计算节点
- 不可变输入&计算结果
 - -> 数据节点
- 高阶函数定义
 - -> 组图
- 函数输入输出
 - -> 节点间依赖
- 函数应用
 - -> DAG求解



图执行引擎：用一张图来表达函数编程

- 构建

```
vertex = builder.add_vertex(F1)
vertex.depend(a);
vertex.depend(b);
vertex.emit(z);
```

.....

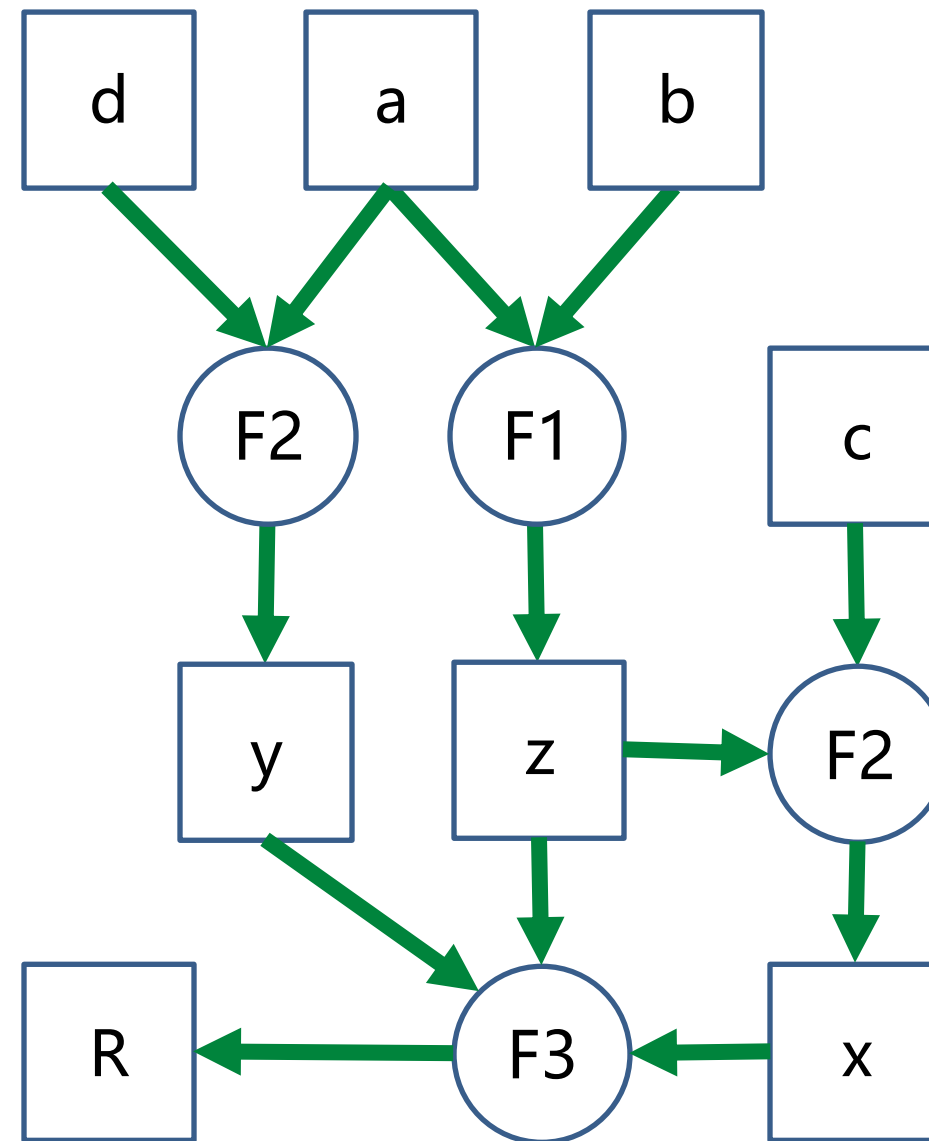
```
graph = builder.build();
```

- 运行

```
graph.find_data(a).set_value(...)
```

.....

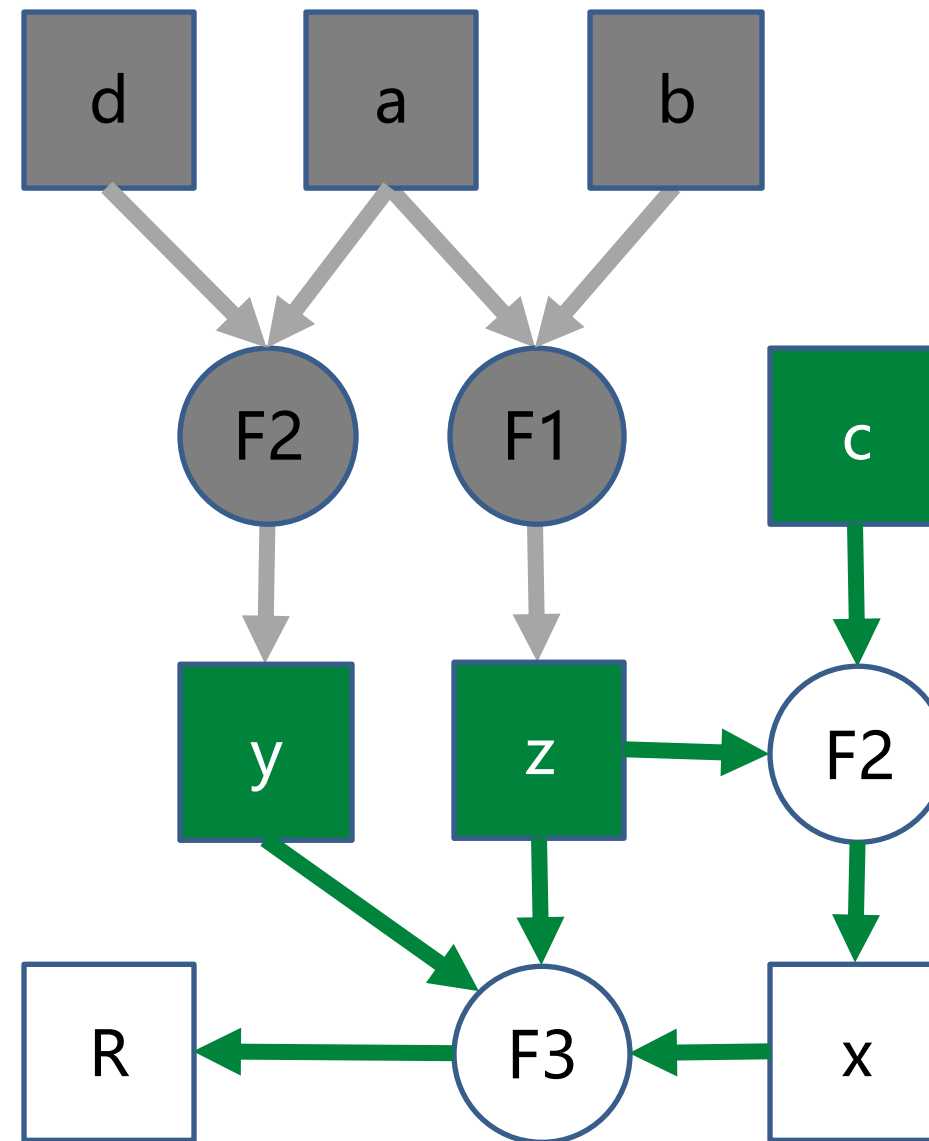
```
graph.run(R)
```



图执行引擎：局部运行

- 运行

```
graph.find_data(z).set_value(...)
graph.find_data(y).set_value(...)
graph.find_data(c).set_value(...)
graph.run(R)
```



- 构建

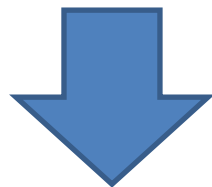
vertex.option(...) // 将任意数据注入函数作为闭包携带

- 初始化

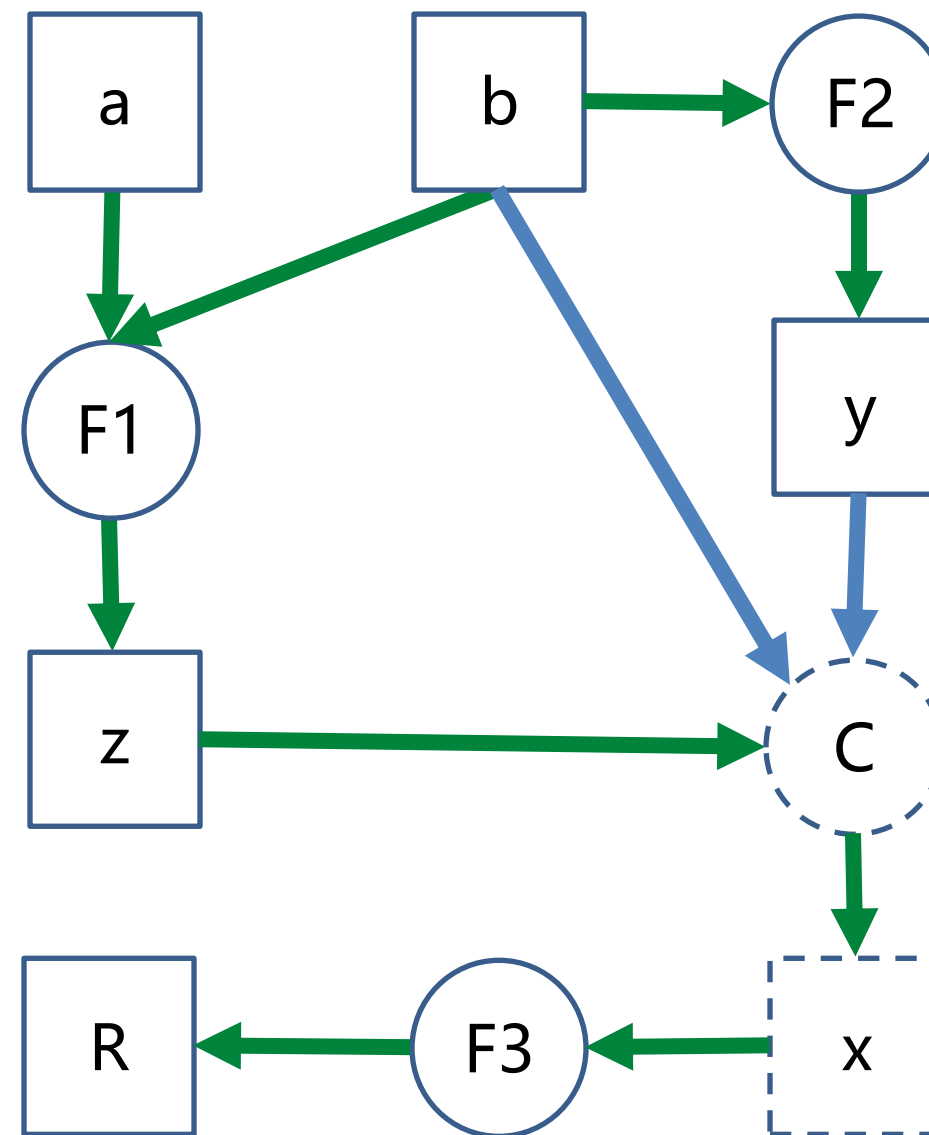
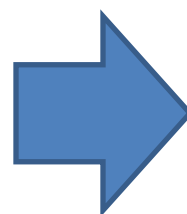
```
int Function::setup(const Any& o) {  
    ... option = o.get<...>()  
    // 依据option初始化，运行时表现不同的行为  
    .....  
    // 初始化一些运行时缓冲区等工作环境，多次运行可复用  
}
```

图执行引擎：惰性求值 -> 条件依赖

- $F(a, b) = F3(F1(a, b) ? F2(b) : b)$



- $z = F1(a, b)$
- $y = F2(b)$
- $x = \textcolor{red}{C}(z, y, b)$
- $R = F3(x)$



图执行引擎：惰性求值 -> 条件依赖

- 构建

builder.add_vertex(F1)

...

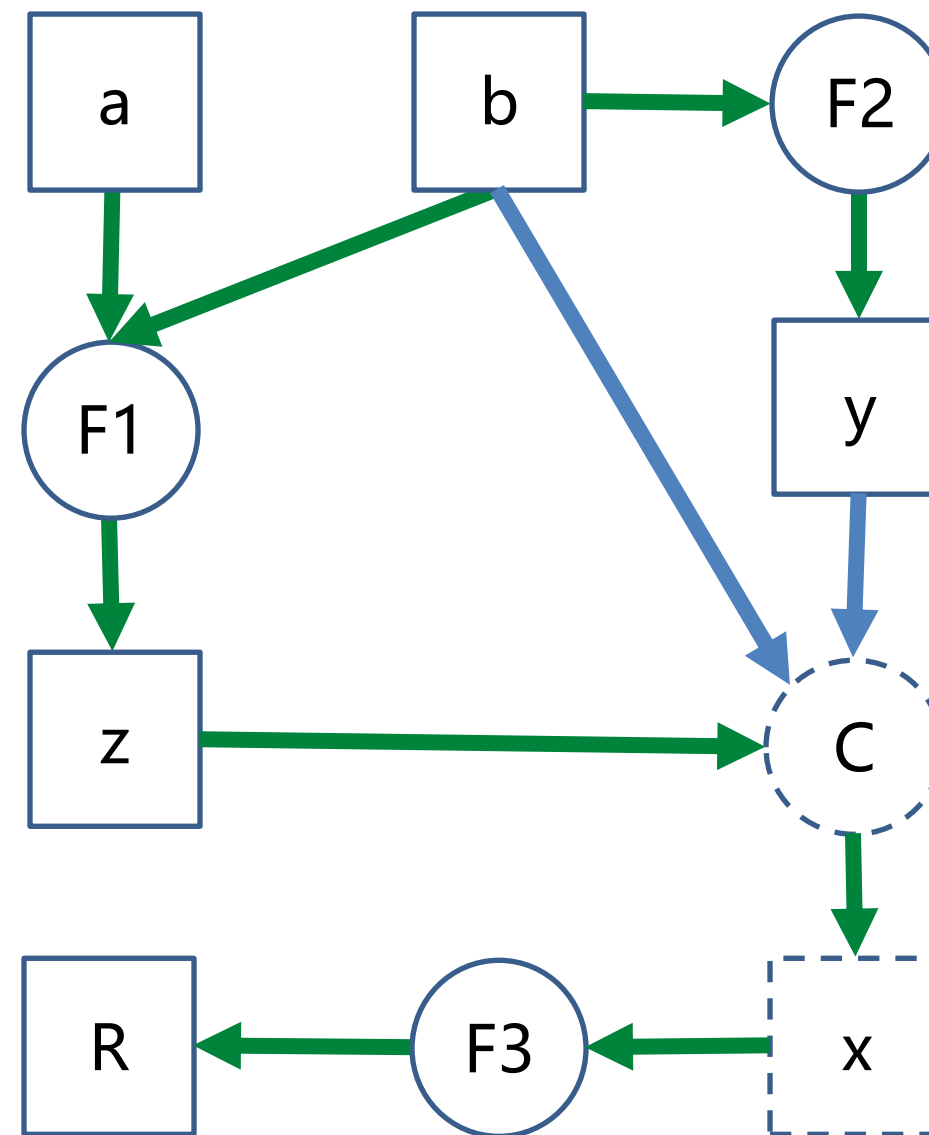
builder.add_vertex(F2)

...

vertex = builder.add_vertex(F3)

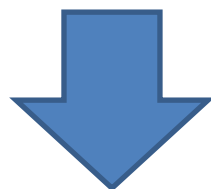
vertex.depend(z ? y : b)

.....

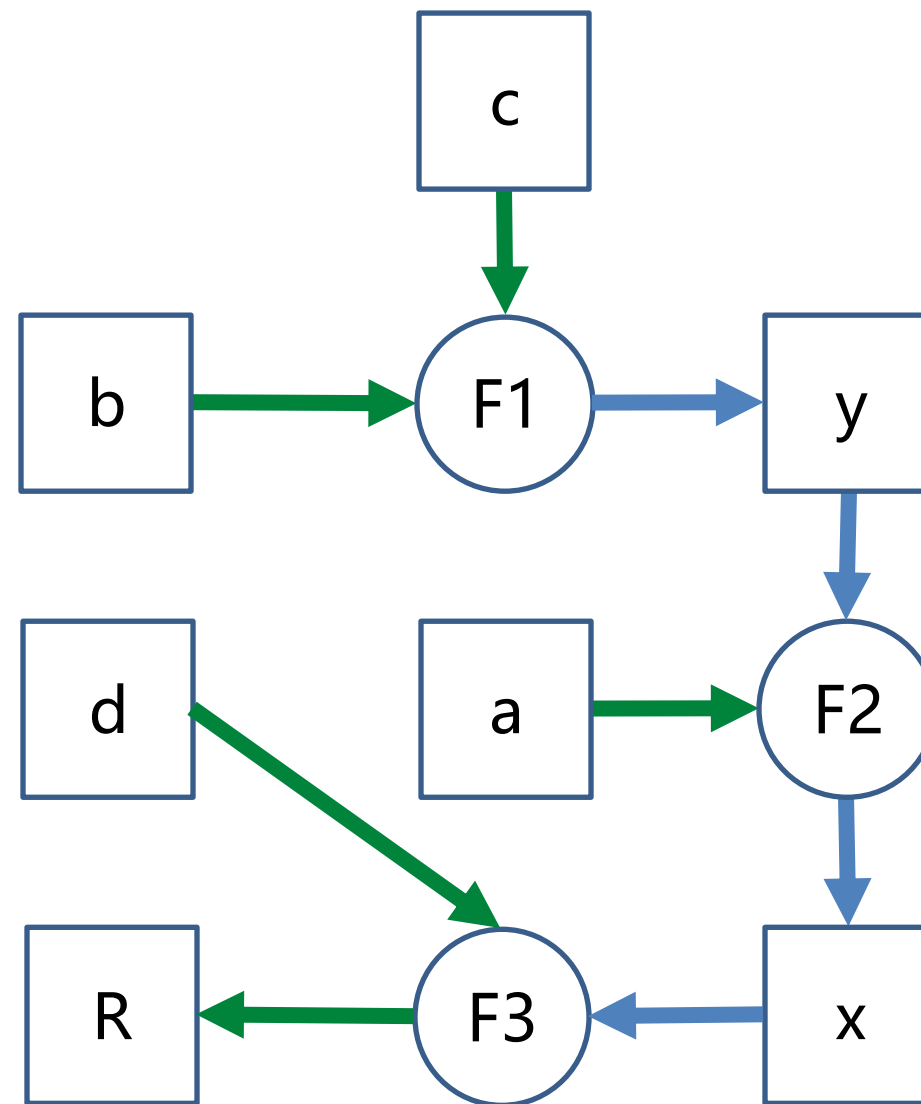
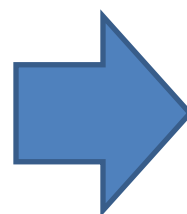


图执行引擎：局部流水线

- $F(a, b, c, d) = F3(F2(a, F1(b, c)), d)$



- channel $y = F1(b, c)$
- channel $x = F2(a, y)$
- $R = F3(d, x)$



图执行引擎：局部流水线



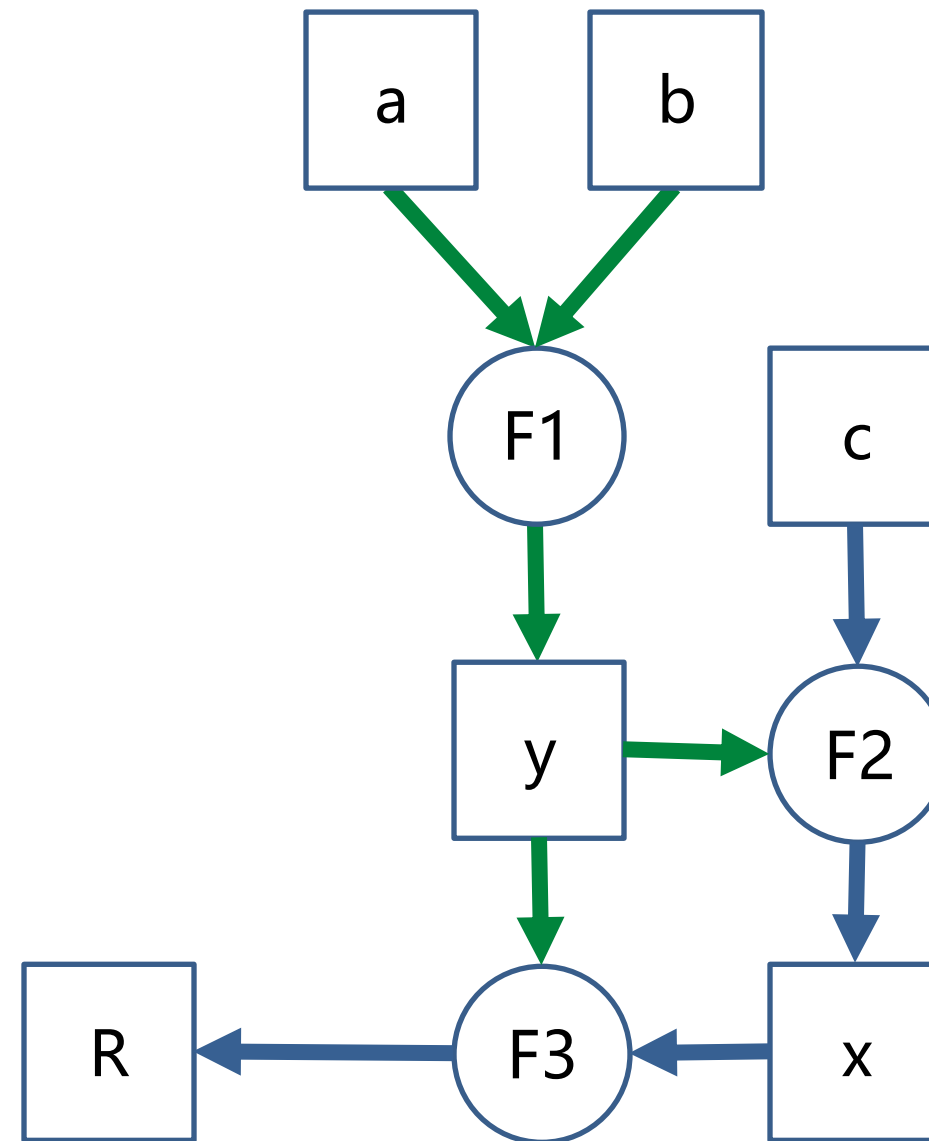
```
int F1::process() {  
    ....  
    {  
        publisher = y.open()  
        // F2被唤起  
        publisher->publish(...)  
    } // 通知F2关闭  
    ....  
}
```

```
int F2::process() {  
    // a产出&y.open之后开始执行  
    ....  
    consumer = y.subscribe()  
    ... = consumer.consume()  
    // 阻塞等待一个发布  
    // 消费完成&上游关闭后收到null  
    ....  
}
```

图执行引擎：引用输出和可变依赖优化

- 可变依赖

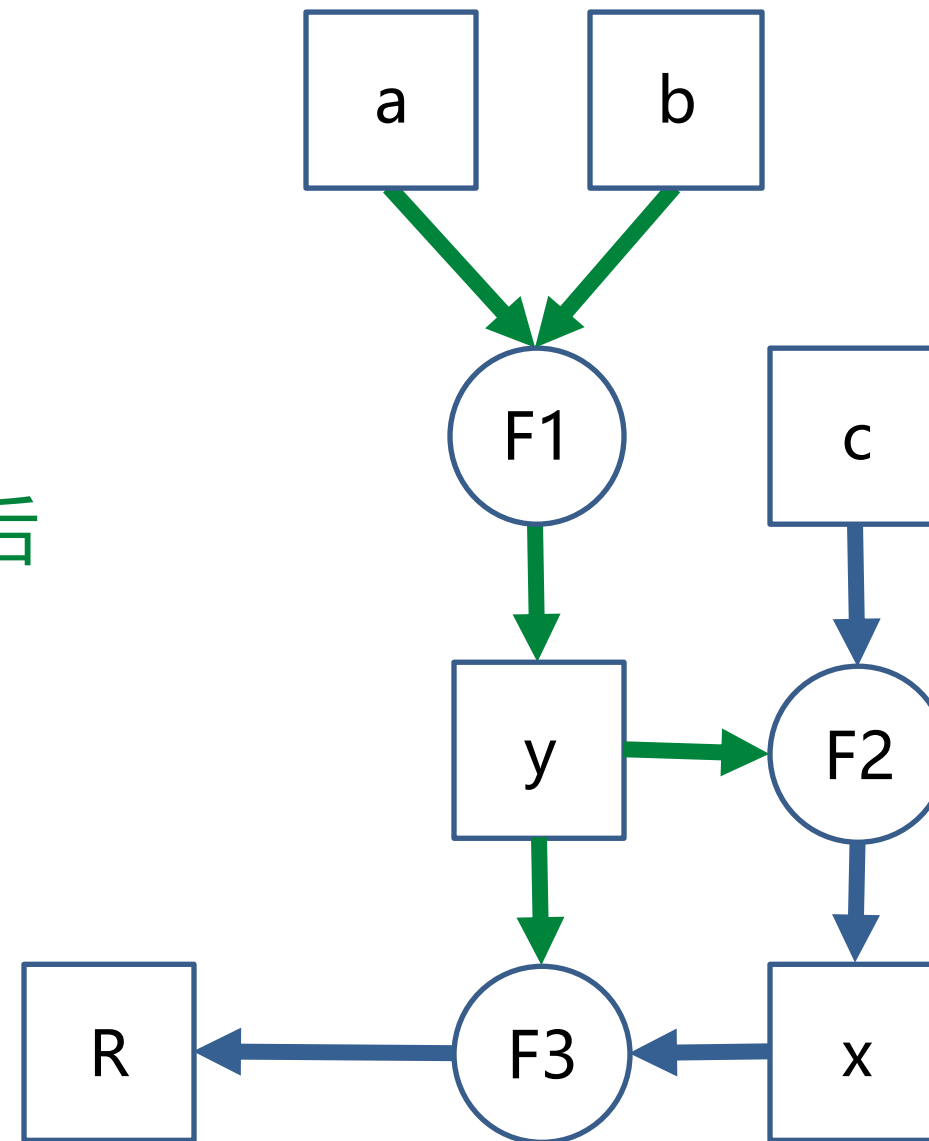
```
int F2::setup(const Any& o) {  
    c.declare_mutable()  
    // 声明需要可变输入  
}  
  
int F2::process() {  
    ... = c.mutable_value()  
    // 声明后可以取得可变指针  
    // 如果框架发现多依赖，则报错  
}
```



图执行引擎：引用输出和可变依赖优化

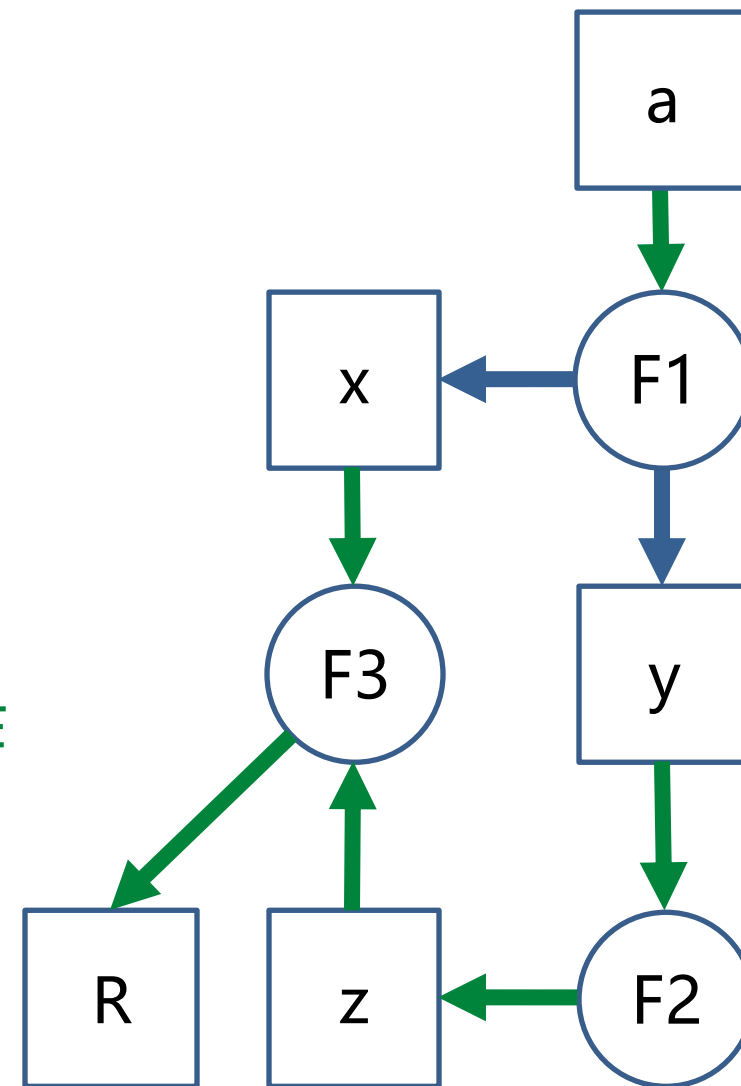
- 引用输出

```
int F2::process() {  
    ... = c.mutable_value()  
    .....  
    x.forward(c) // 直接转发输入，修改后  
    x.emit().ref(...) // 引用输出任意值  
}
```



图执行引擎：分阶段多输出优化

```
int F1::process() {  
    y.emit()  
    // F2已经可以开始执行了  
    ..... // 继续计算x, 可能耗时较久  
    x.emit()  
    // F3可以开始执行了  
    ..... // 可以继续做些打复杂日志等后置操作  
}
```



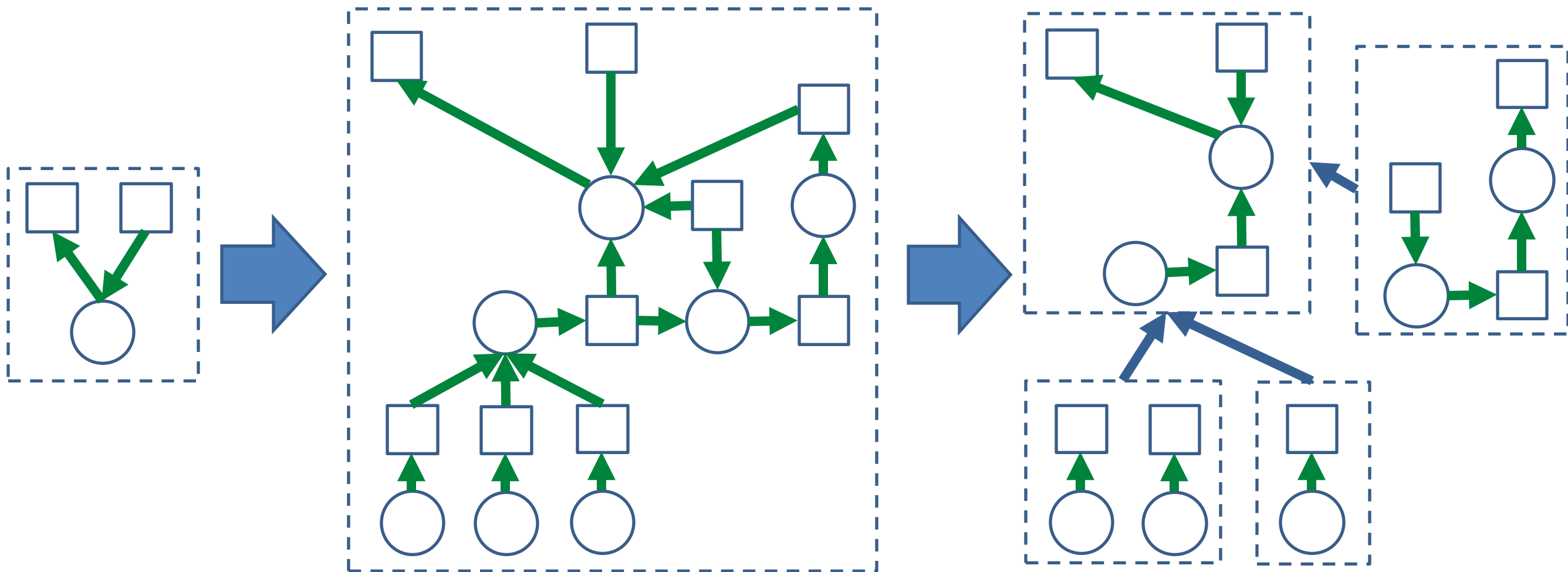
图执行引擎：目前应用案例

- FEED在线特征和模型排序子系统
 - 预估推理***qps、***kps
- FEED矩阵APP推荐系统
 - 全民&好看推荐业务系统等***qps
- 组外应用
 - 图搜特征服务

图执行引擎展望：FEED是一个函数

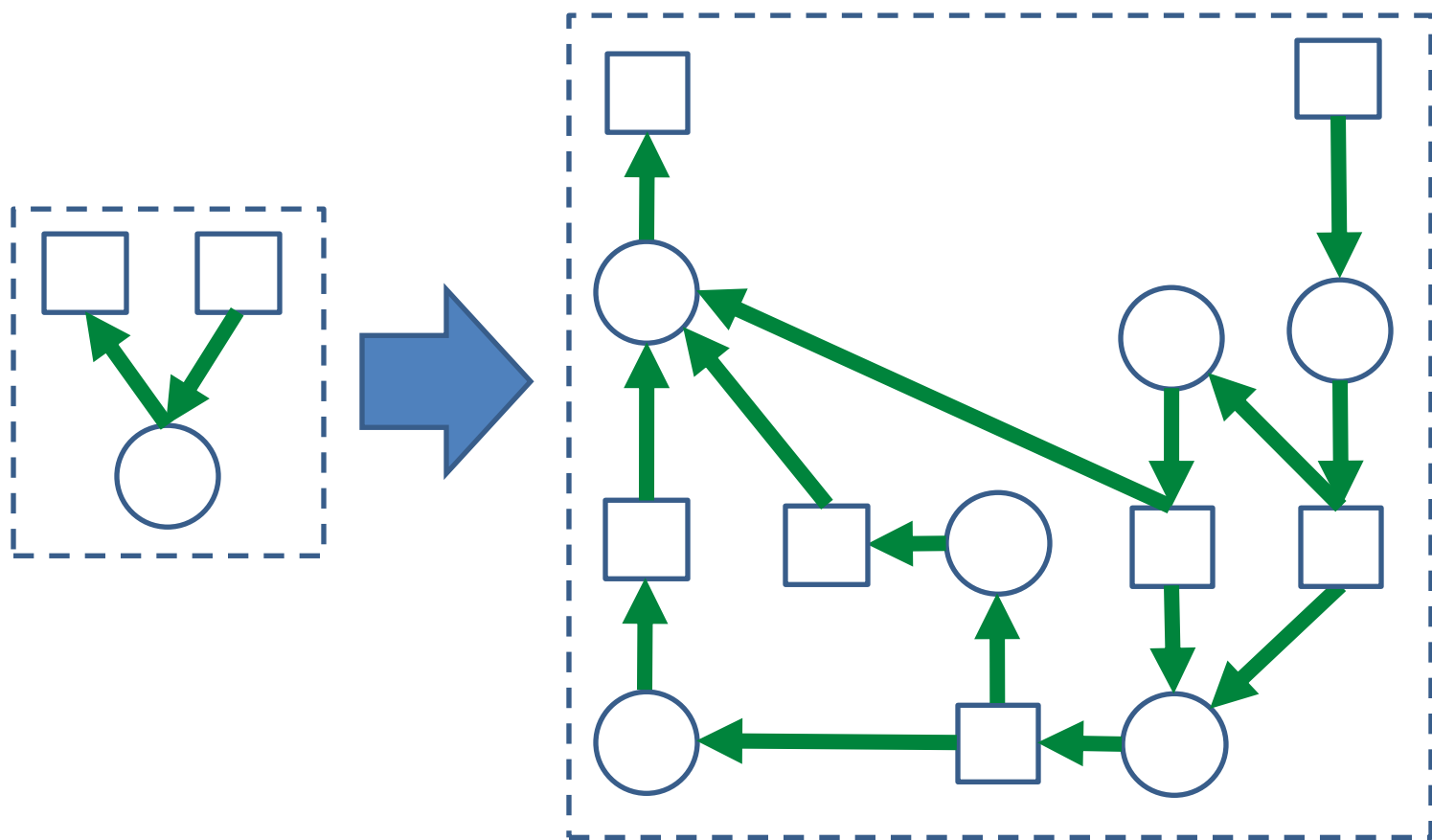
- $LIST = REC (USER)$

- 服务分割&RPC连接



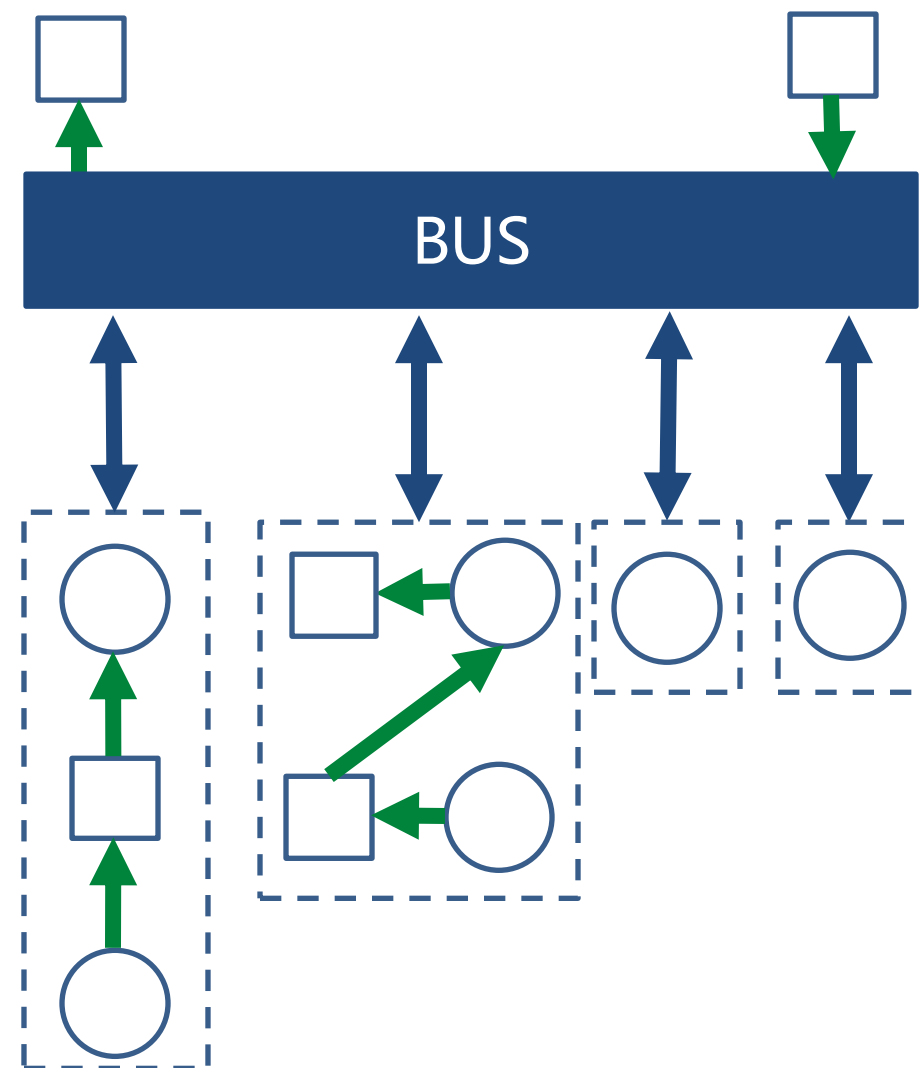
图执行引擎展望：SPIDER是一个函数

- DOC = SPIDER (URL)



内部资料，请勿外传

- 服务分割&消息连接

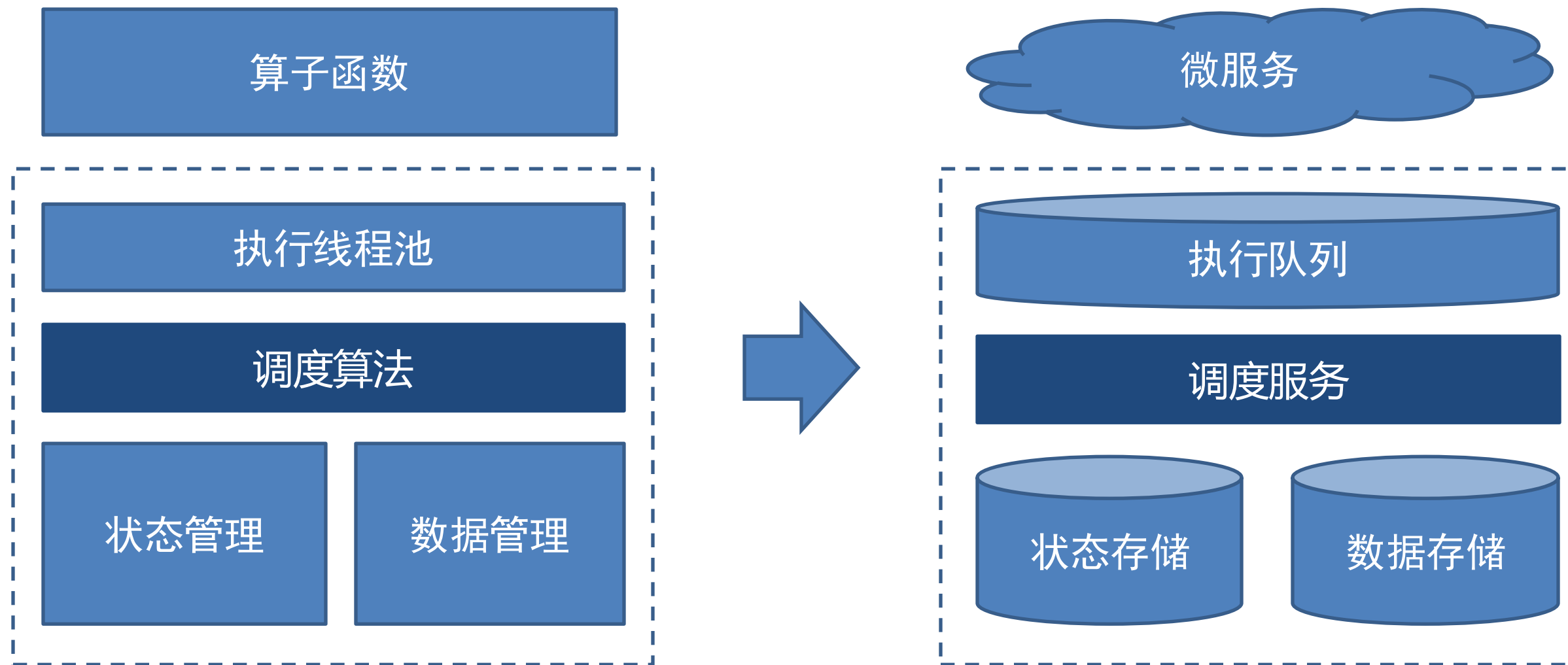


图执行引擎展望：何处是我们的星辰大海



- Focus On Function
 - Assemble As Graph
 - Split To Service
 - Connect By Message
- ~ = Function As A Service

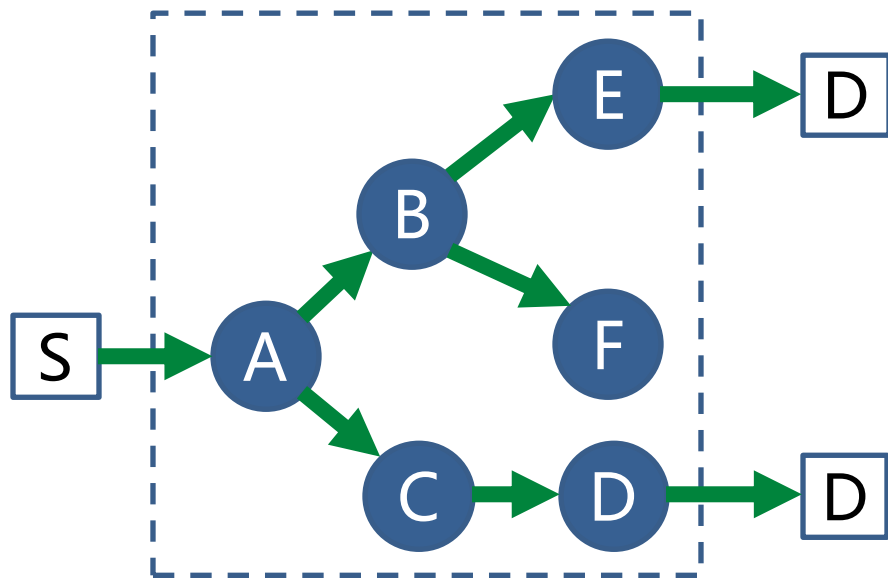
图执行引擎展望：单进程推广到分布式



图执行引擎展望：流式计算到数据流图

流式计算

- 数据传递执行流程耦合
- 没有标准的汇聚支持



数据流图

