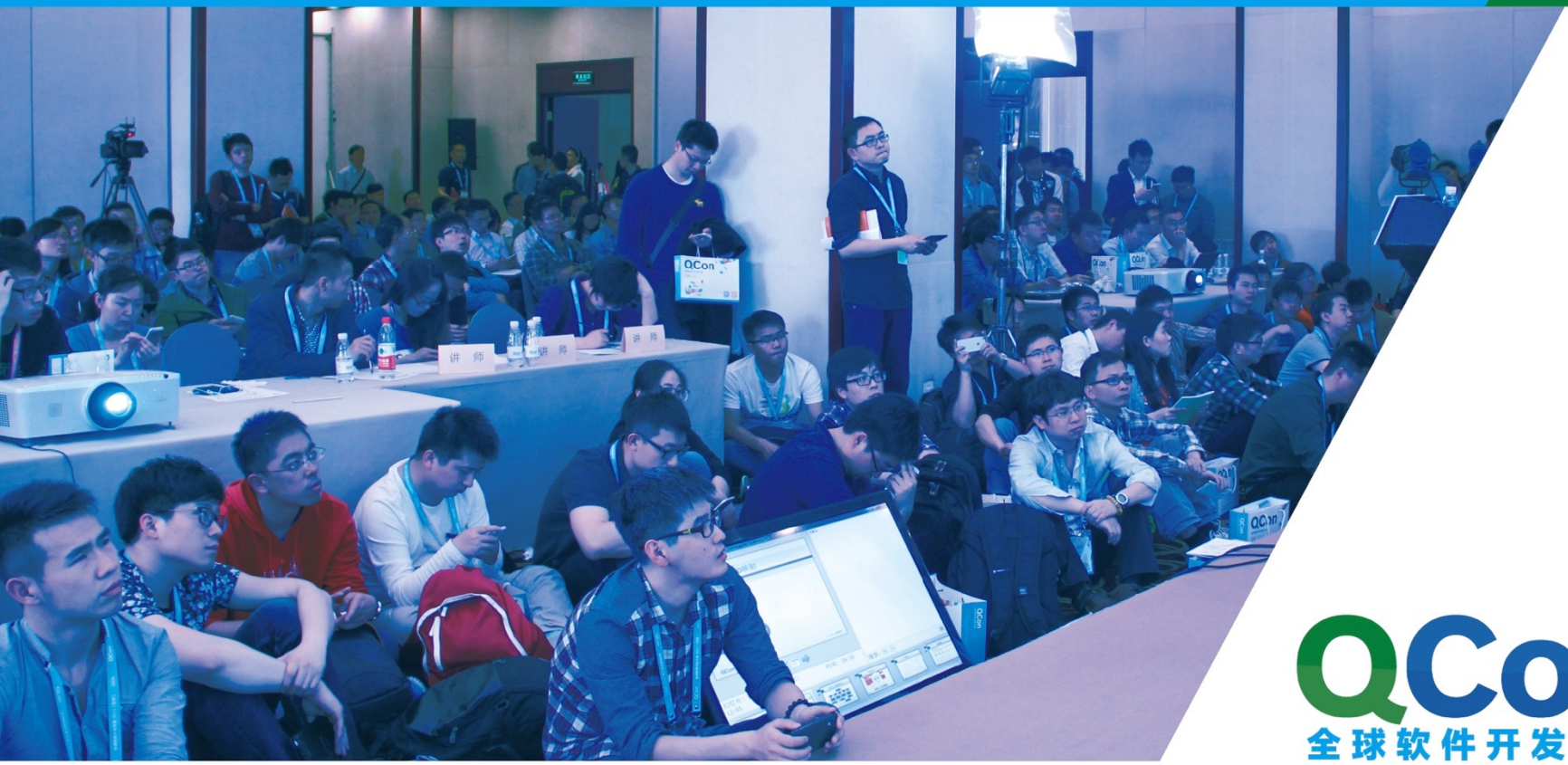
QCon全球软件开发大会

International Software Development Conference

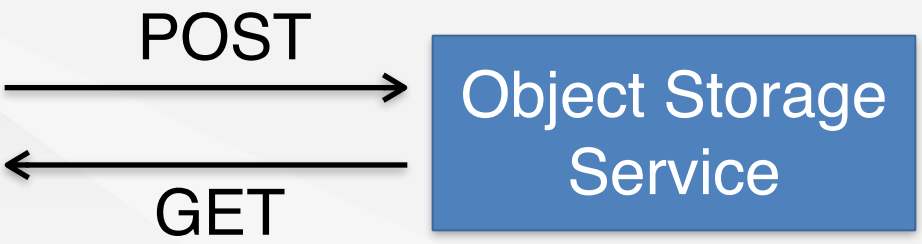


QCon
全球软件开发大会

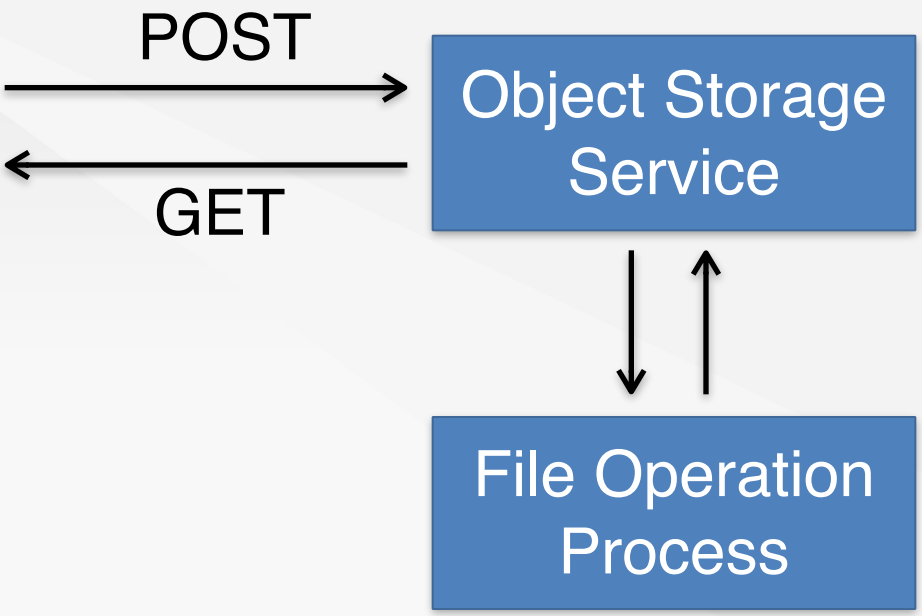
基于连接与组合的微服务架构剖析

xuli@qiniu.com

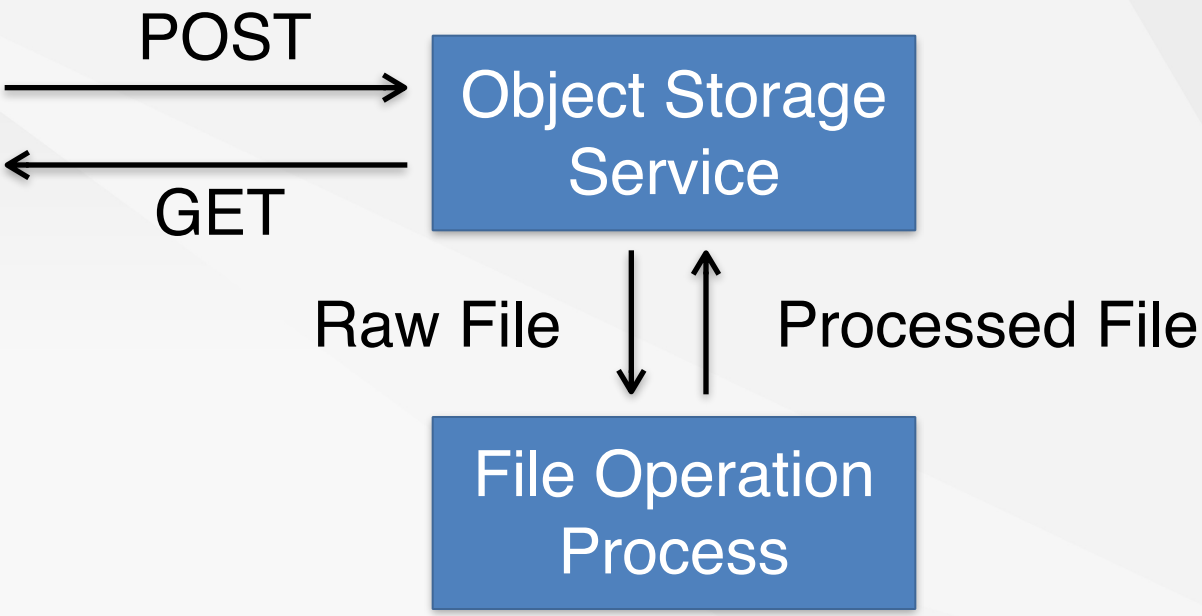
Qiniu 1.0



Qiniu 2.0



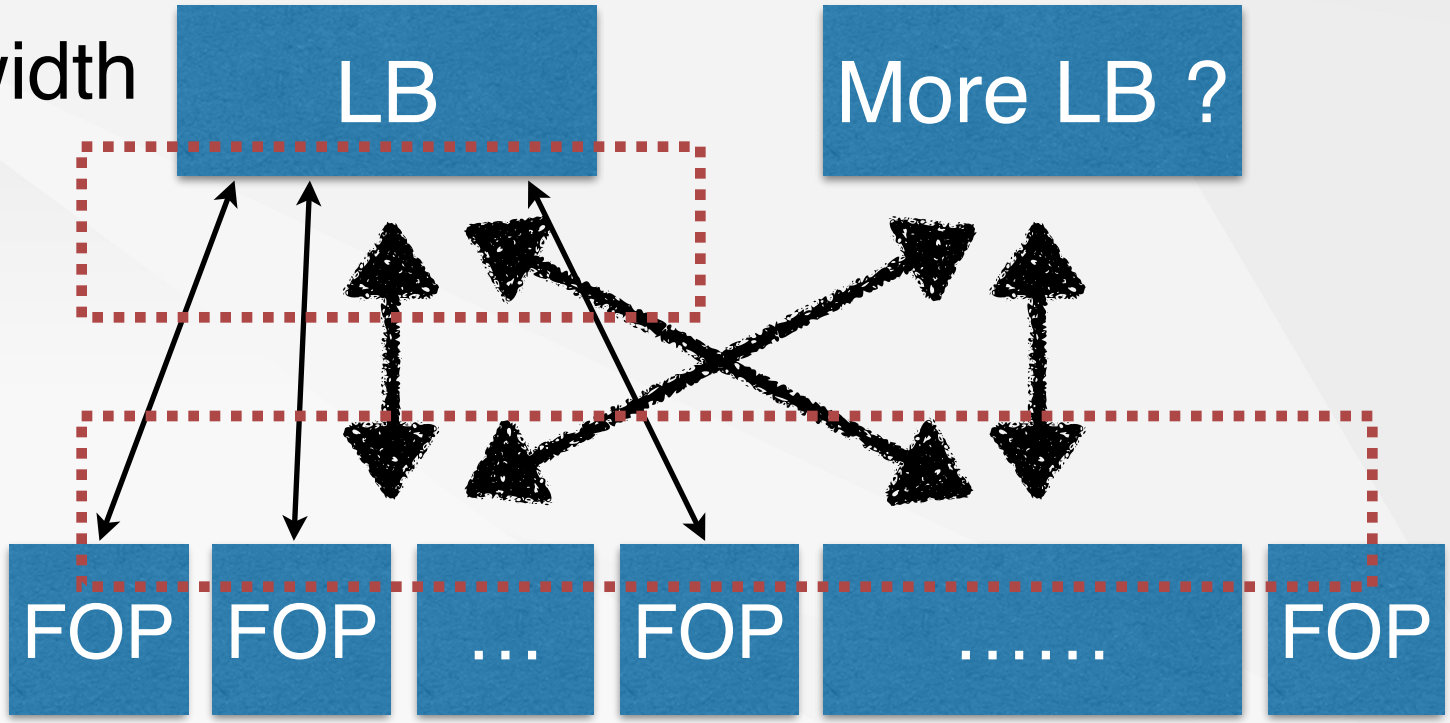
Qiniu 2.0 / FOP



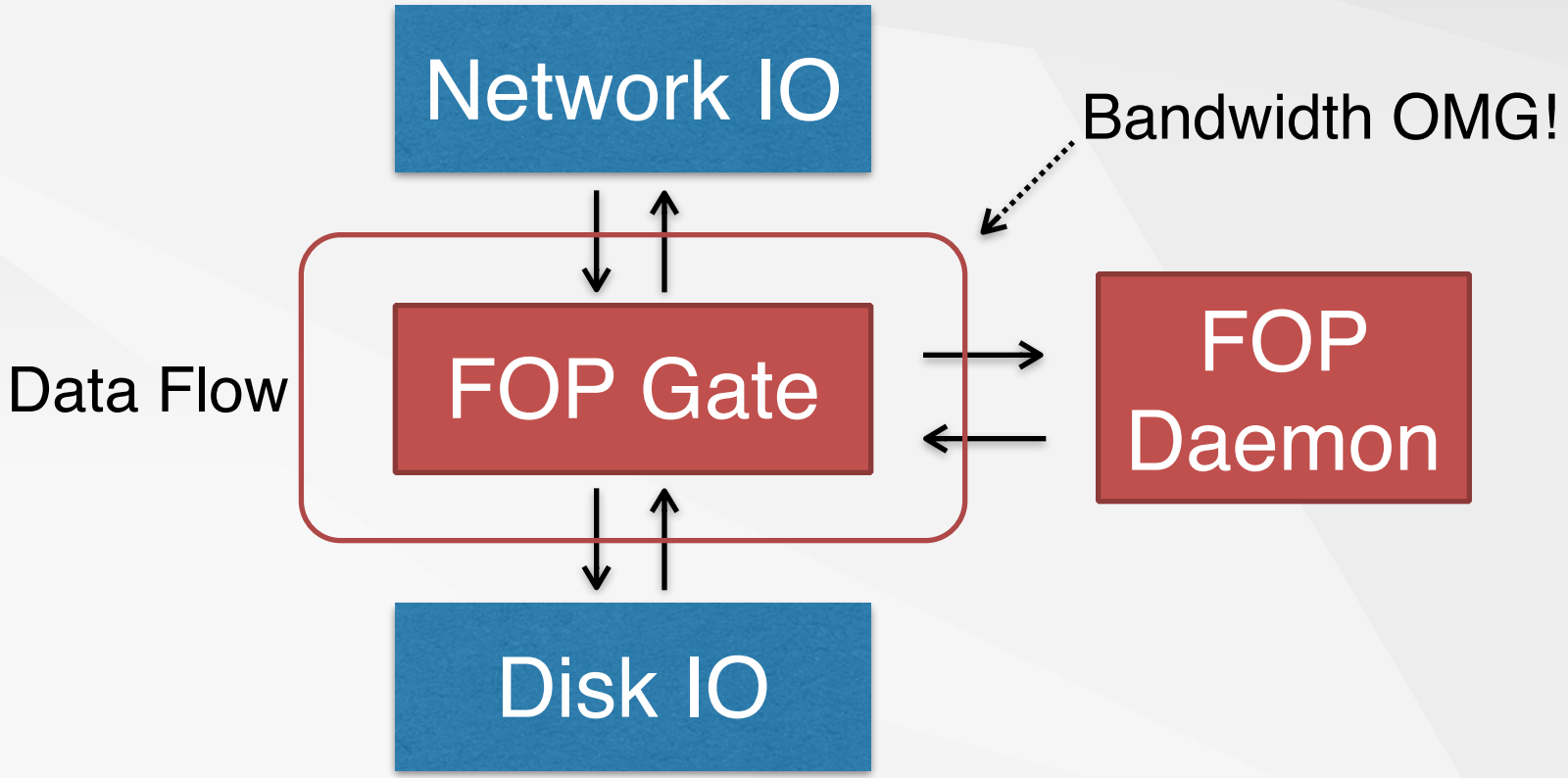
FOP (File Operation Process)

early FOP

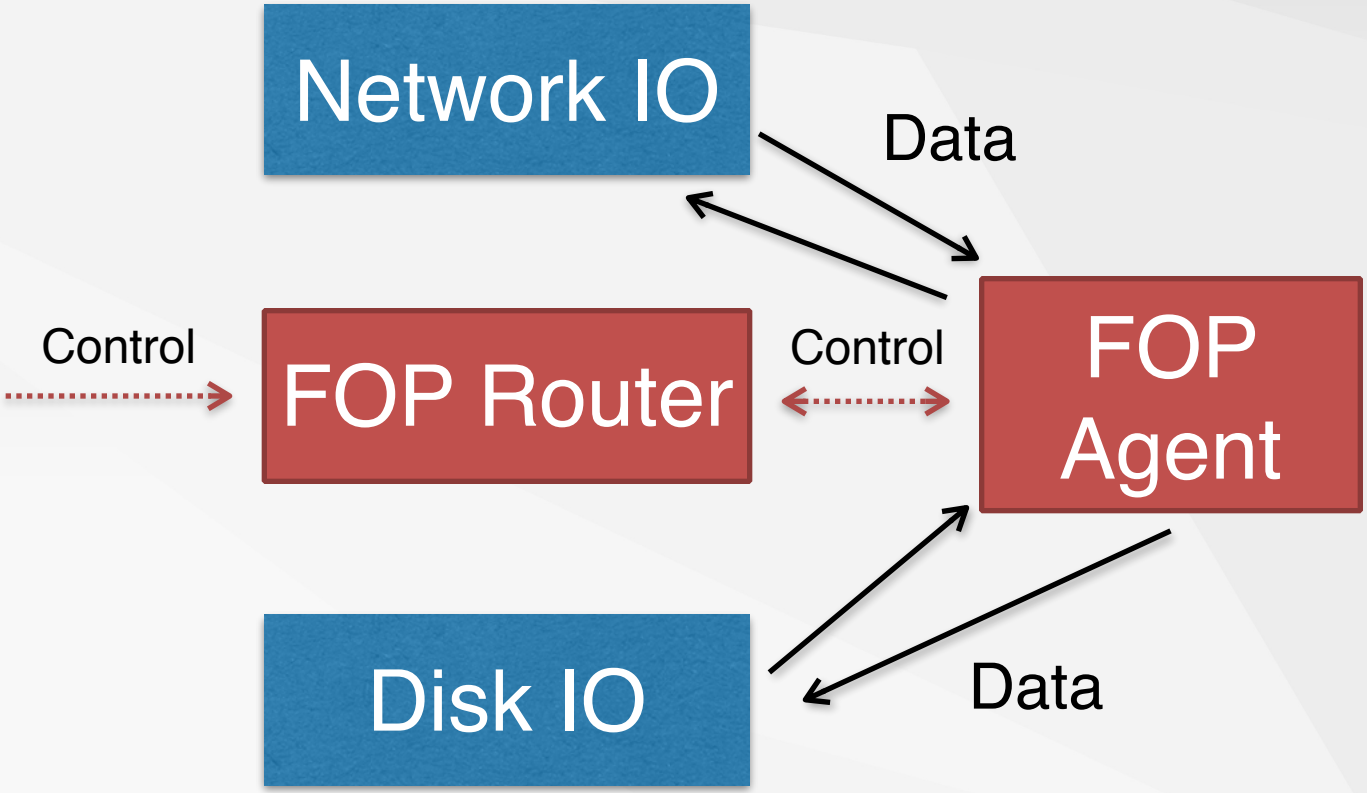
Bandwidth



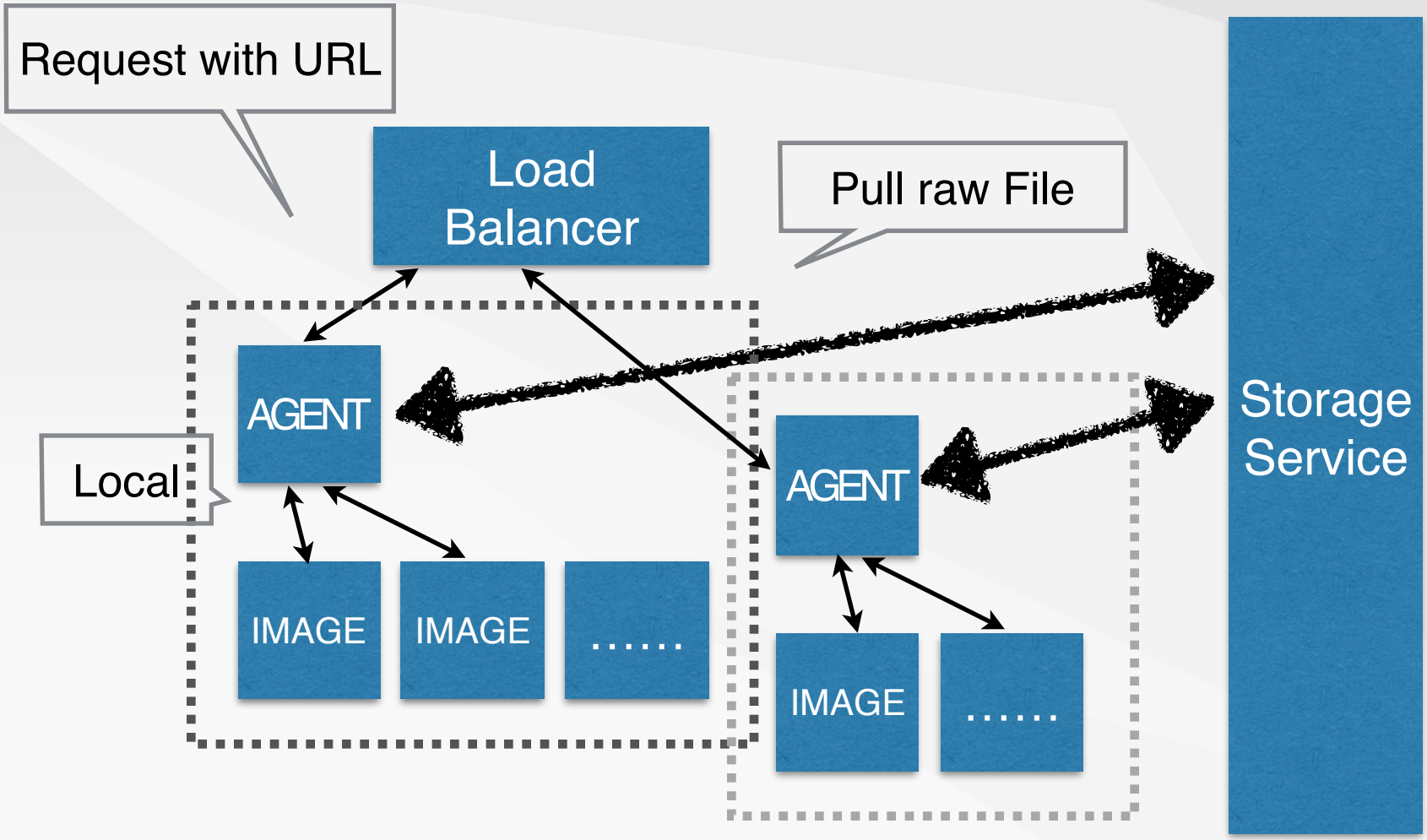
early FOP Bottlenecks

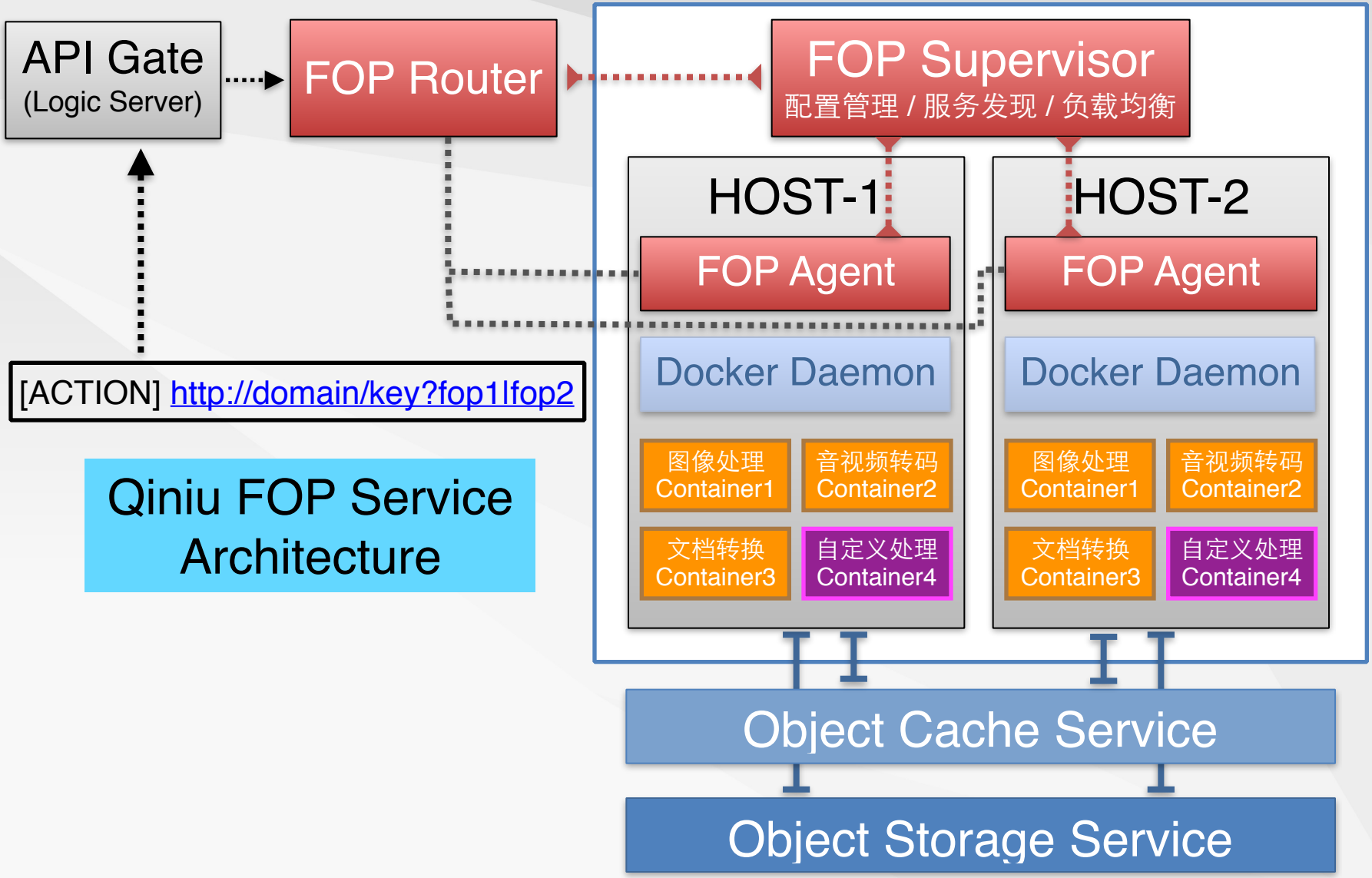


early FOP Bottlenecks Fix

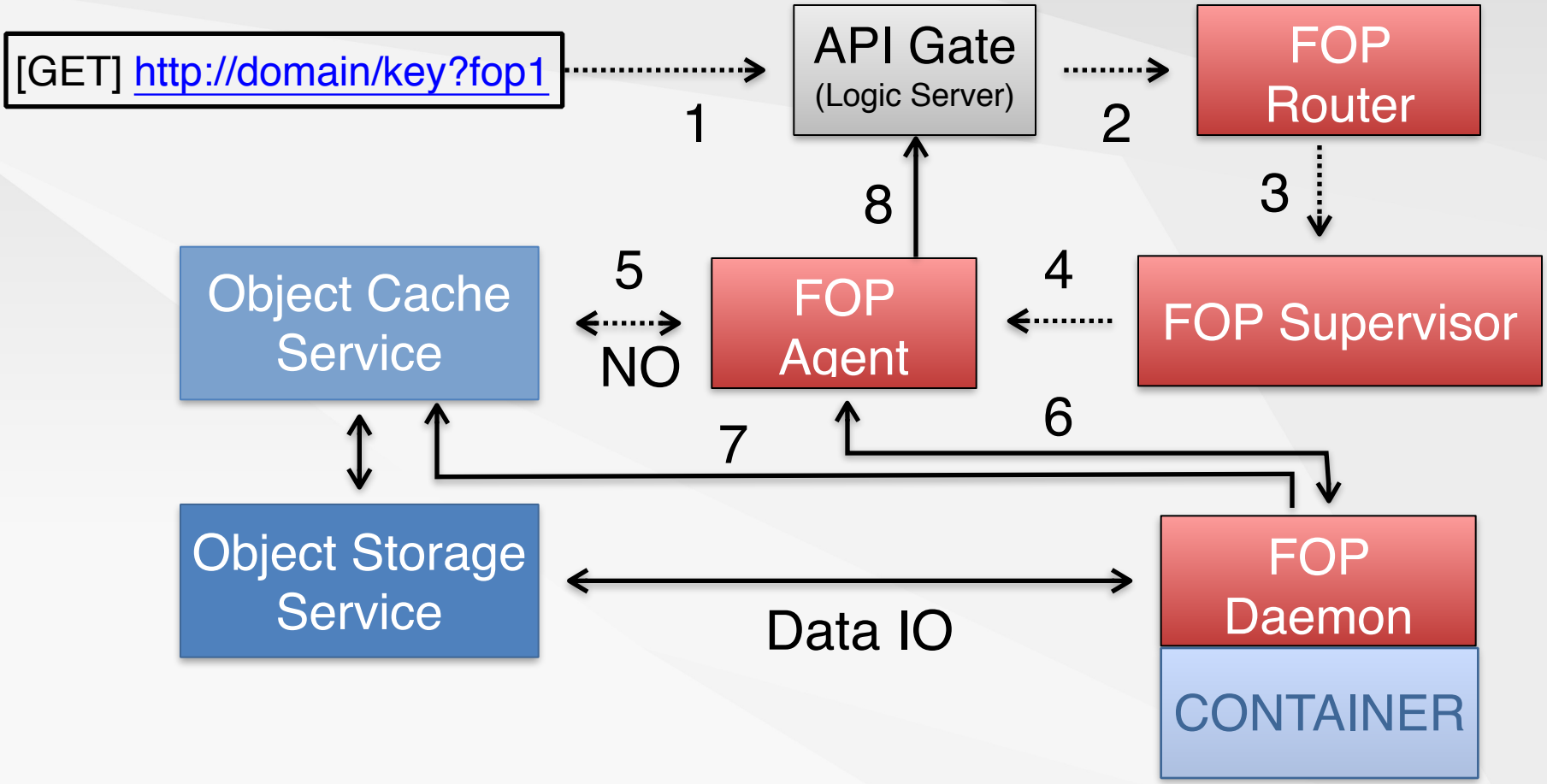


“nanoservices”




API Gate
(Logic Server)
FOP Router
FOP Supervisor
配置管理 / 服务发现 / 负载均衡
HOST-1
HOST-2
FOP Agent
FOP Agent
Docker Daemon
Docker Daemon
图像处理
Container1
音视频转码
Container2
文档转换
Container3
自定义处理
Container4
图像处理
Container1
音视频转码
Container2
文档转换
Container3
自定义处理
Container4
[ACTION] http://domain/key?fop1|fop2
Qiniu FOP Service
Architecture
Object Cache Service
Object Storage Service

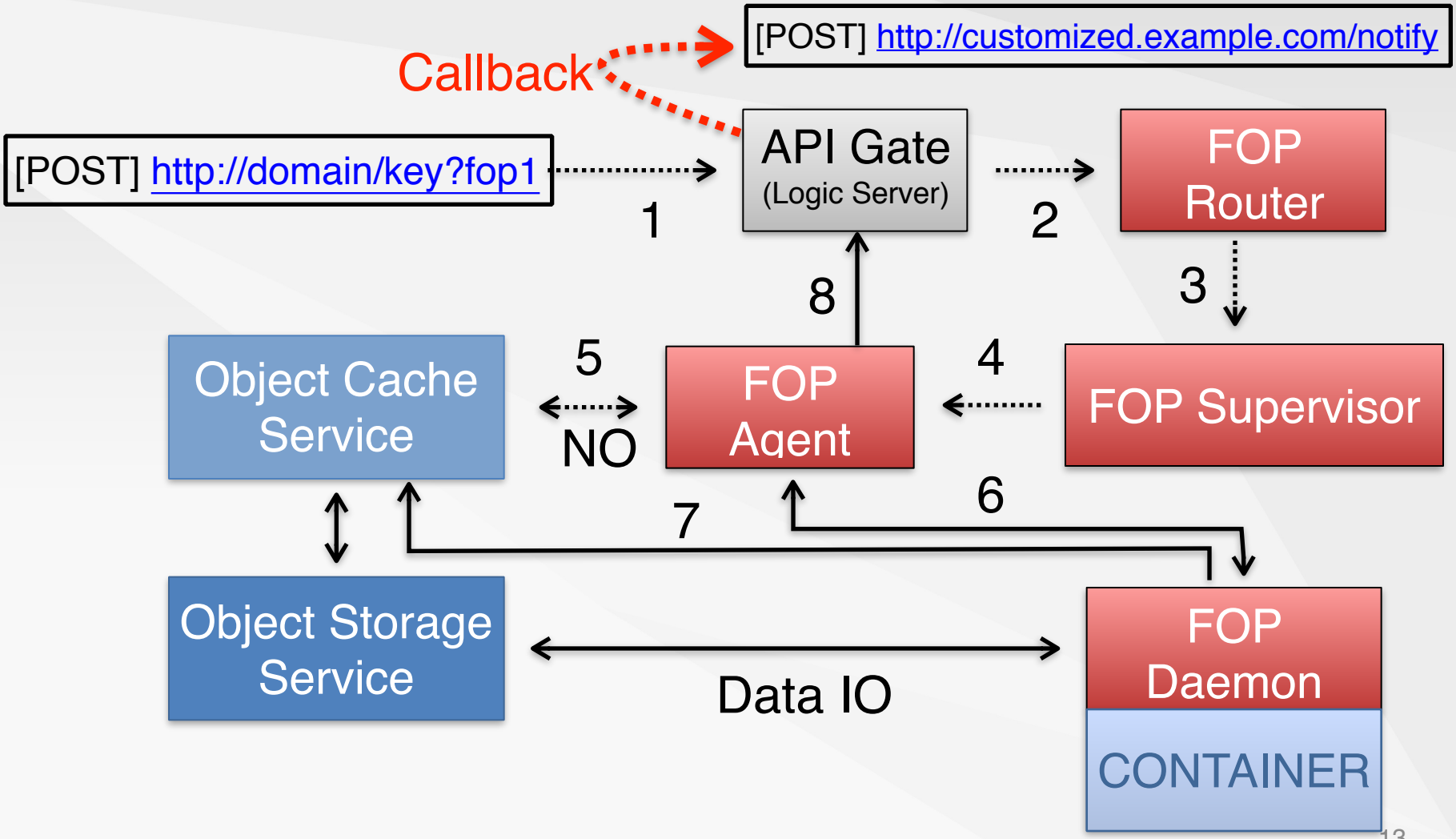
How FOP Works?



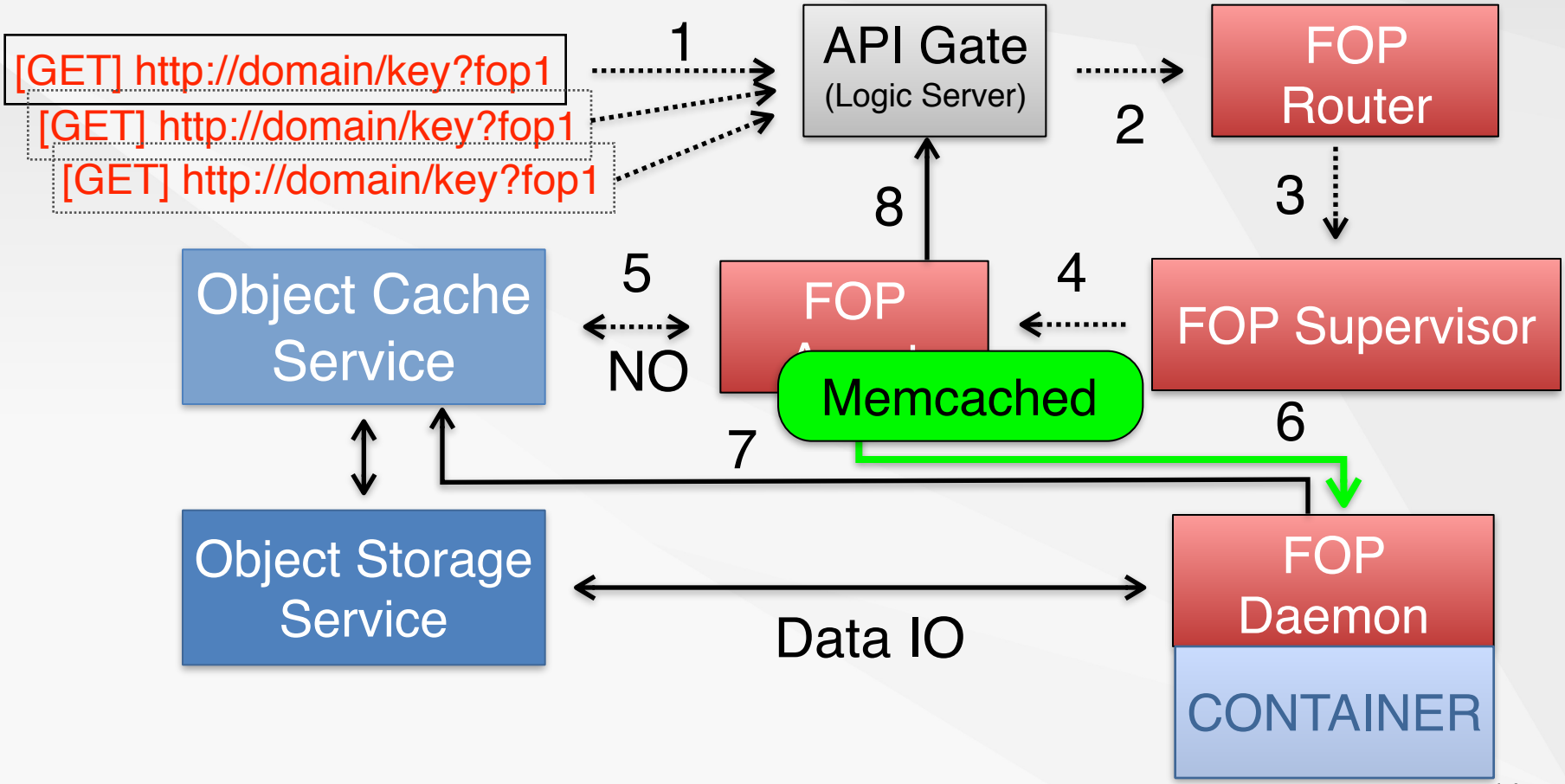
Questions

» FOP 离线计算怎么做?
» 同一个 FOP 并发请求怎么办?
» 多个 FOP 组合，工作流如何处理?
» FOP 程序（IMAGE）可以自定义吗?

How FOP Offline Works?



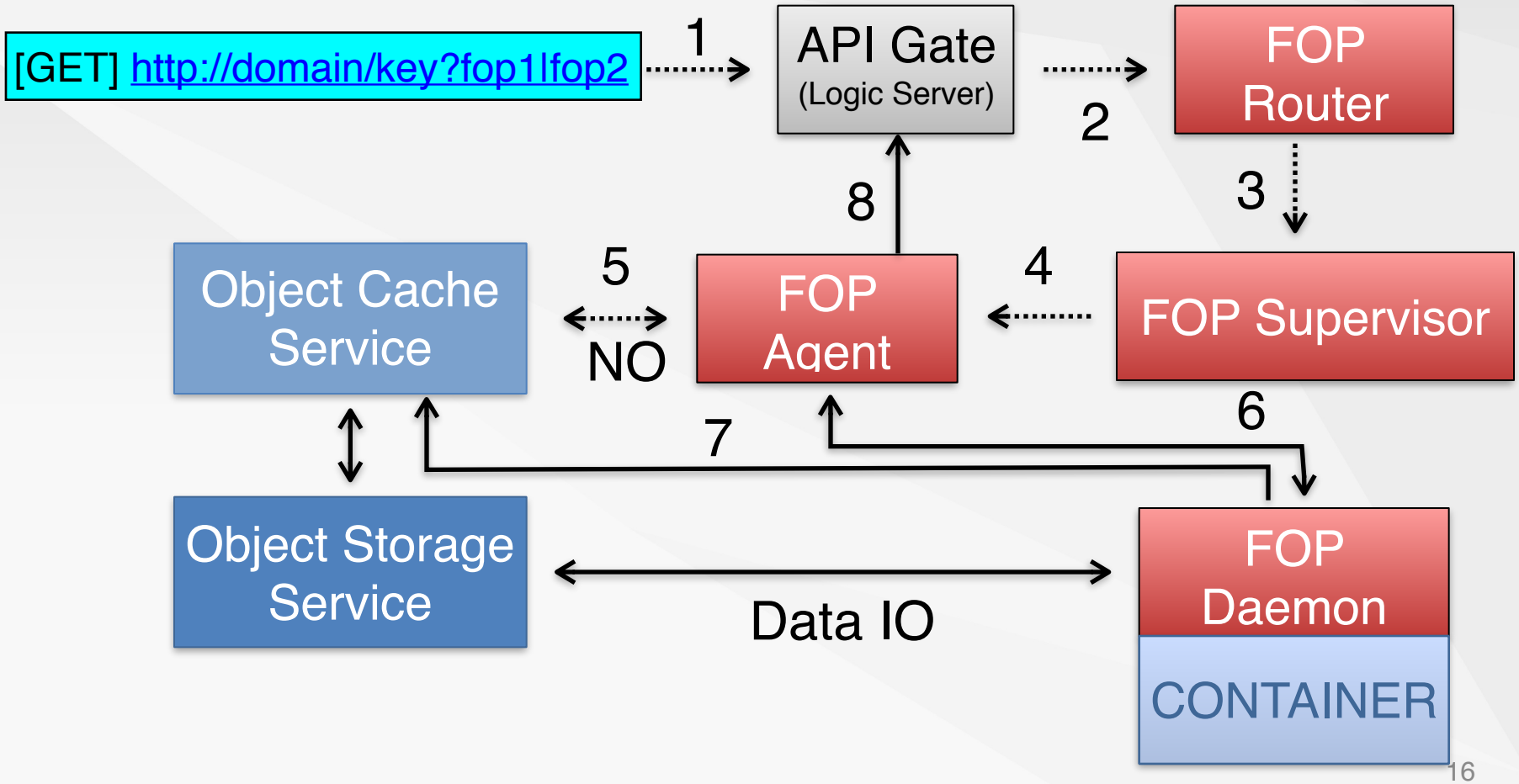
How FOP Concurrency Works?



Waiting for results

```
done := make(chan Result, 1)
go func() {
    ...
    done <- Result{}
}()
...
select {
case result := <-done:
    do(result)
case <- time.After(3 * time.Second):
    fmt.Println("timeout")
}
```

How Multi-FOP Works?



What is Multi-FOPs?

[ACTION] http://domain/key?fop1|fop2|fop3|...

从 Unix 谈起

- Unix 的连接和组合
  - app1 params1 | app2 params2
- App 接口
  - 输入： stdin, params
  - 输出： stdout
  - 协议： text (data stream)
- Pipeline
  - 将一个 app 的输出(stdout) 转为另一个 app 的输入 (stdin)

论 awk 编程与 Unix 管道艺术

200 GET ...
500 POST ...
200 GET ...
500 GET ...

HTTP CODE COUNTING

```
awk '{h[$1]++} END {for(k in h) print h[k], k}' test.log
```

```
cut -d' ' -f1 test.log | sort | uniq -c
```

Unix Pipelines

awk - pattern-directed scanning and processing language

cut -- cut out selected portions of each line of a file

sort - sort lines of text files

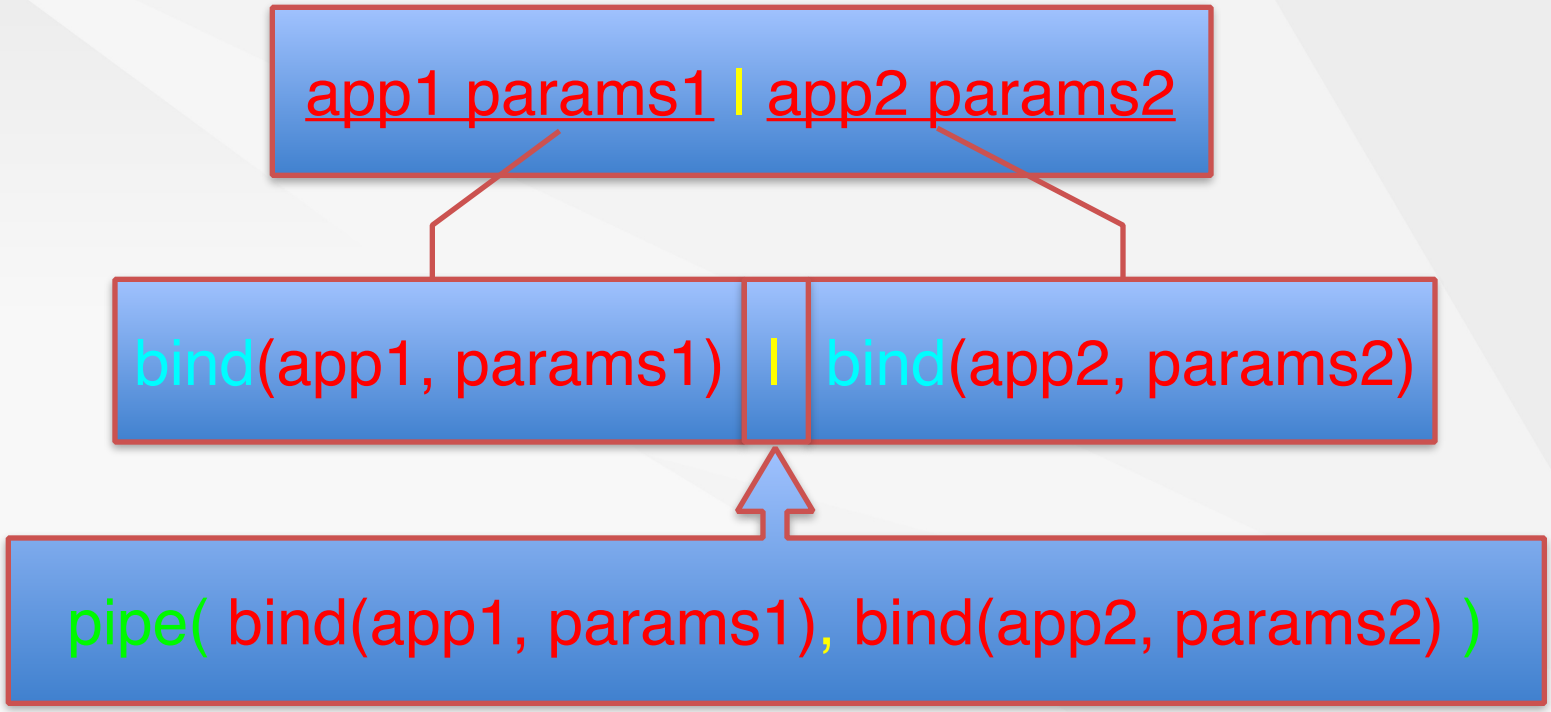
uniq -- report or filter out repeated lines in a file

Pipeline 关键点

- 多个 app 是并行执行的
  - 上游每产生一段output，会立即交由下游处理。
- app 间的协议是松散耦合的
  - 上游 app 的 ouput 是 xml 还是 json，下游 app 需要知晓，但是属于一种松散的耦合关系，并无任何强制的约束。

Go 对 Unix Pipeline 的仿真

- App
  - func(in io.Reader, out io.Writer, args []string)



Go 对 Unix Pipeline 的仿真

```
func bind(
    app func(in io.Reader, out io.Writer, args []string),
    args []string
) func(in io.Reader, out io.Writer) {

    return func(in io.Reader, out io.Writer) {
        app(in, out, args)
    }
}
```

Go 对 Unix Pipeline 的仿真

```
func pipe(
    app1 func(in io.Reader, out io.Writer),
    app2 func(in io.Reader, out io.Writer)
) func(in io.Reader, out io.Writer) {

    return func(in io.Reader, out io.Writer) {
        pr, pw := io.Pipe()
        defer pw.Close()
        go func() {
            defer pr.Close()
            app2(pr, out)
        }()
        app1(in, pw)
    }
}
```

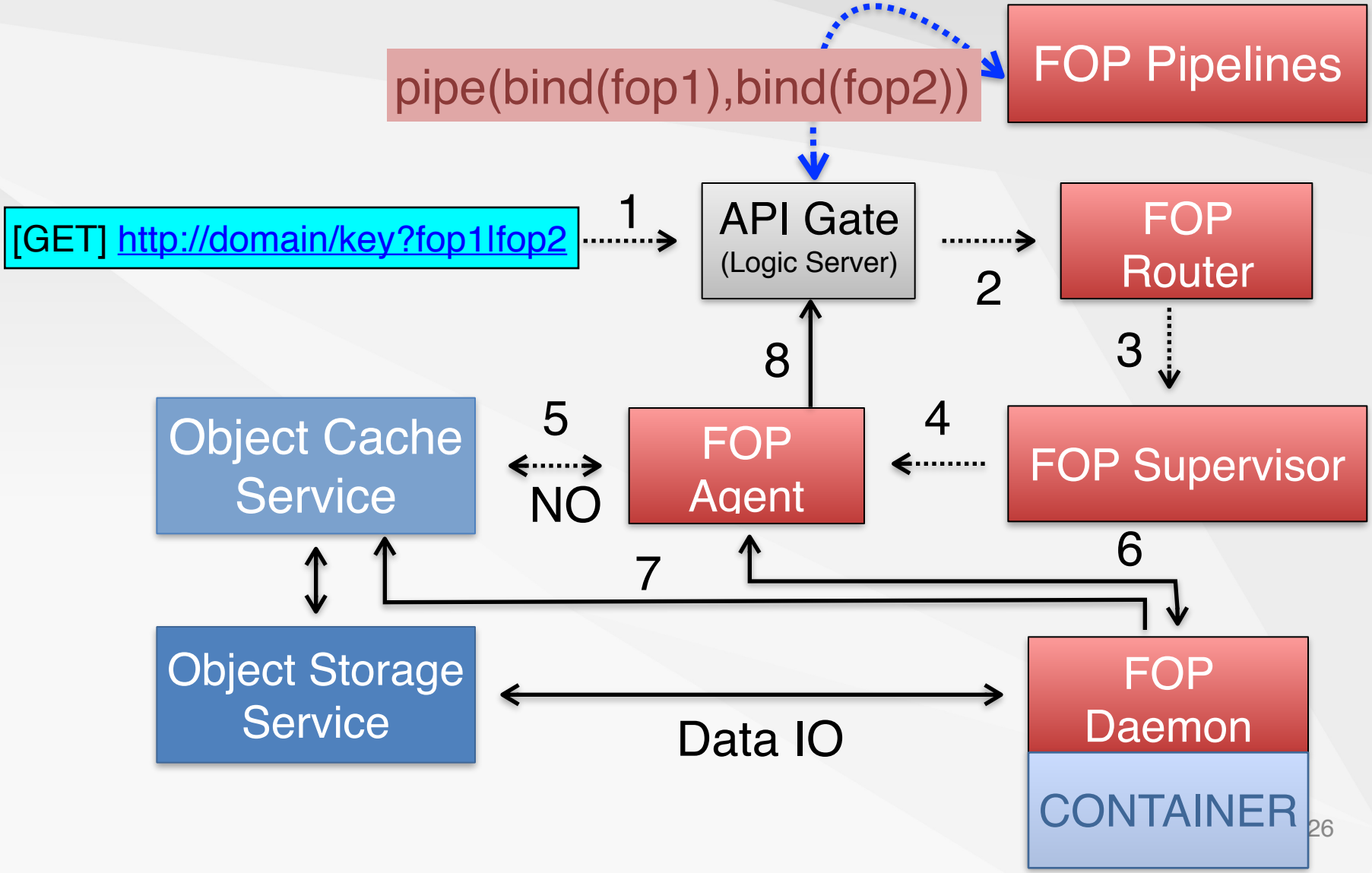
Go 对 Unix Pipeline(s) 的仿真

```
func pipe(apps ...func(in io.Reader, out io.Writer)) func(in io.Reader, out io.Writer) {

        if len(apps) == 0 { return nil }
        app := apps[0]
        for i := 1; i < len(apps); i++ {
                app1, app2 := app, apps[i]
                app = func(in io.Reader, out io.Writer) {
                        pr, pw := io.Pipe()
                        defer pw.Close()
                        go func() {
                                defer pr.Close()
                                app2(pr, out)
                        }()
                        app1(in, pw)
                }
        }
        return app
}
```

难以想象的优雅!

Multi-FOP Works



FOP 程序可以自定义吗?

- Pipelined FOP services
- User-defined FOP service (Docker)

自定义数据处理 UFOP

- UFOP：“用户自定义的数据处理进程”
- 基于 Docker
- 面向对象存储做就近计算处理
- UFOP 程序可以是自行部署，也可以选用七牛 UFOP 公共平台上已有的服务程序。
- Docker 为部署自定义 Image 提供了可能性

UFOP 案例

https://hd.qiniu.com/nrop2015/

图普科技

鉴黄服务

广州图普网络科技有限公司

图片鉴黄服务能够帮您有效判断保存在七牛云的图片是否为不良图片

价格：5.5123 元/千次 起

开始使用

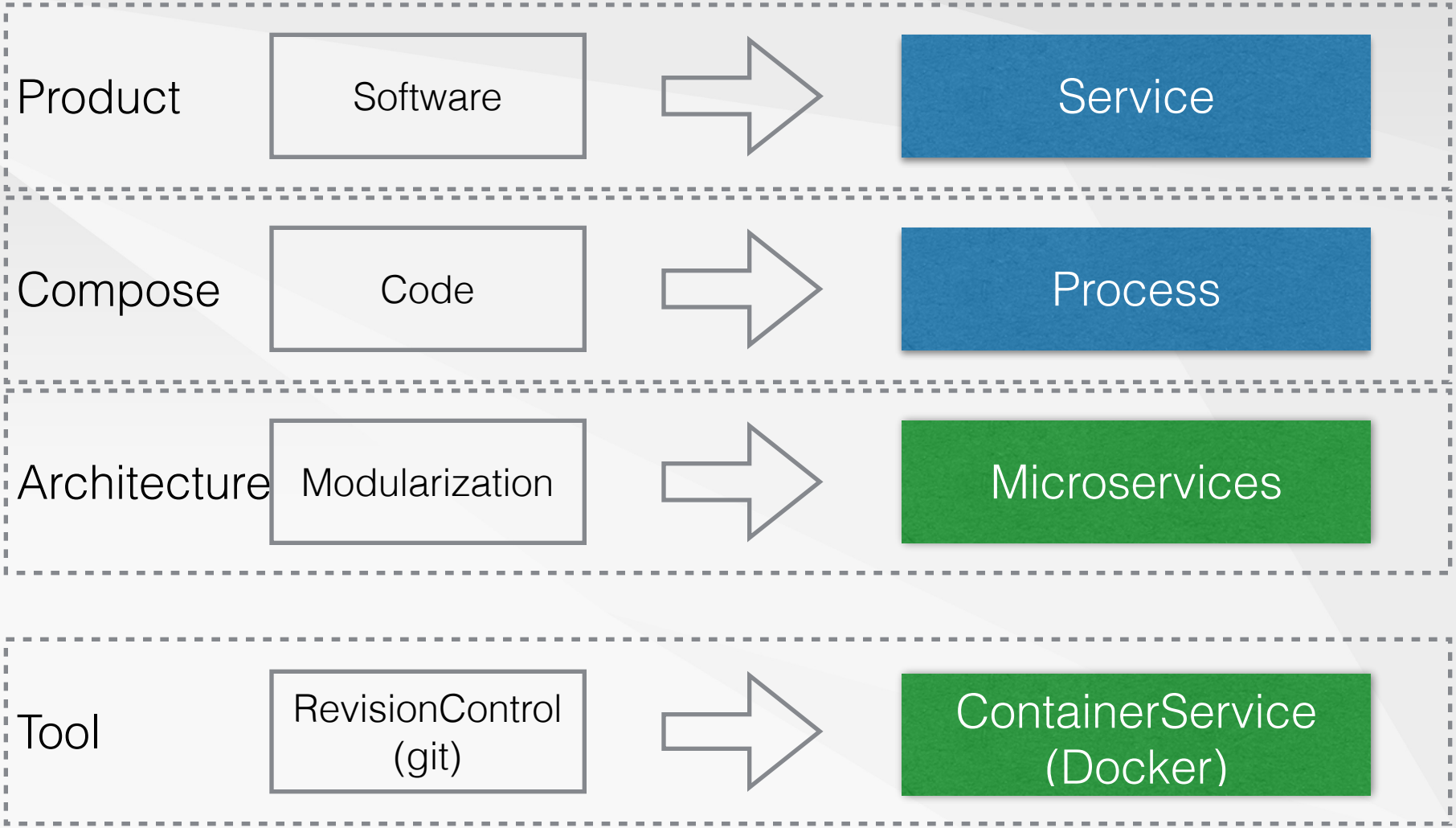
+

我要加入七牛服务市场

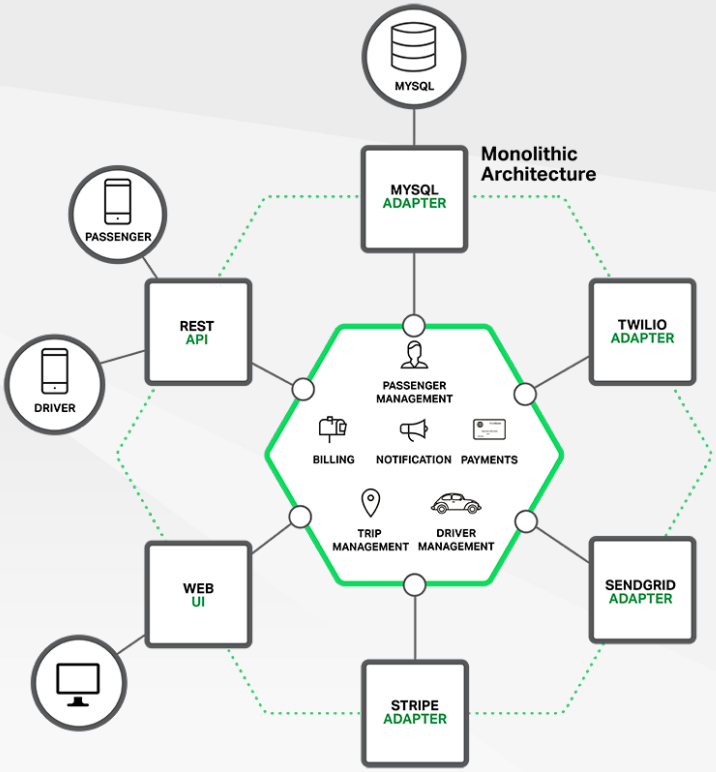
UFOP 不可以做什么

- 不可以作为网站主机。因为 UFOP 没有 IP:PORT，只能通过 URL?ufop_api_args 形式调用。其中 ufop_api_args 由 UFOP 程序自定义。
- 只要数据存在七牛，针对该数据的各种特征计算处理，都可以使用 UFOP 实现。
- UFOP 也可与 七牛云 平台上的其他 FOP 无缝集成，管道链式处理: fop1 | fop2 | app3 | appN | …

Why Docker

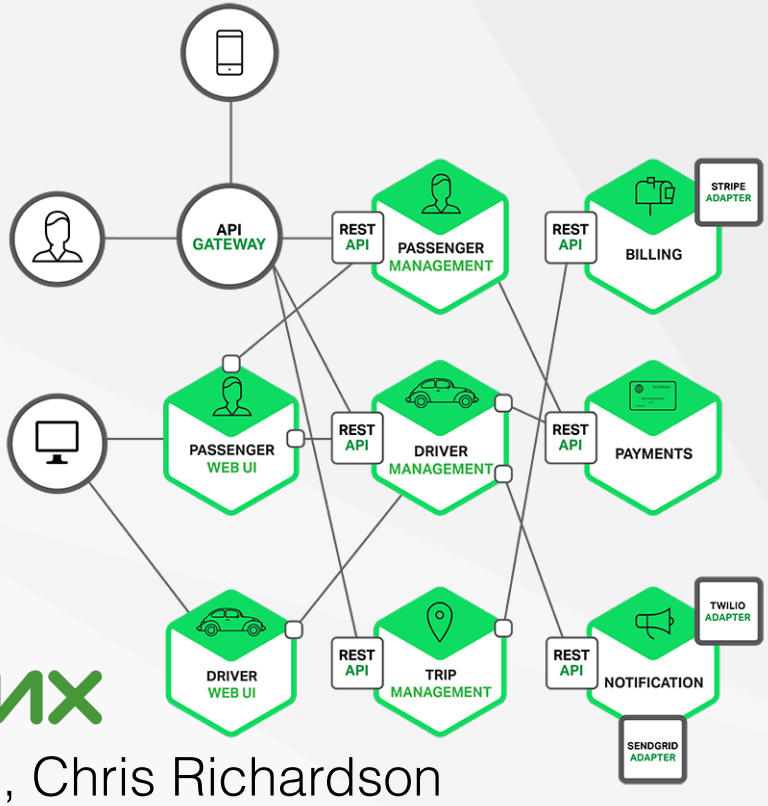

Product
Software
Service
Compose
Code
Process
Architecture
Modularization
Microservices
Tool
RevisionControl
(git)
ContainerService
(Docker)

软件架构变迁



Monolithic Application

Microservices



NGINX

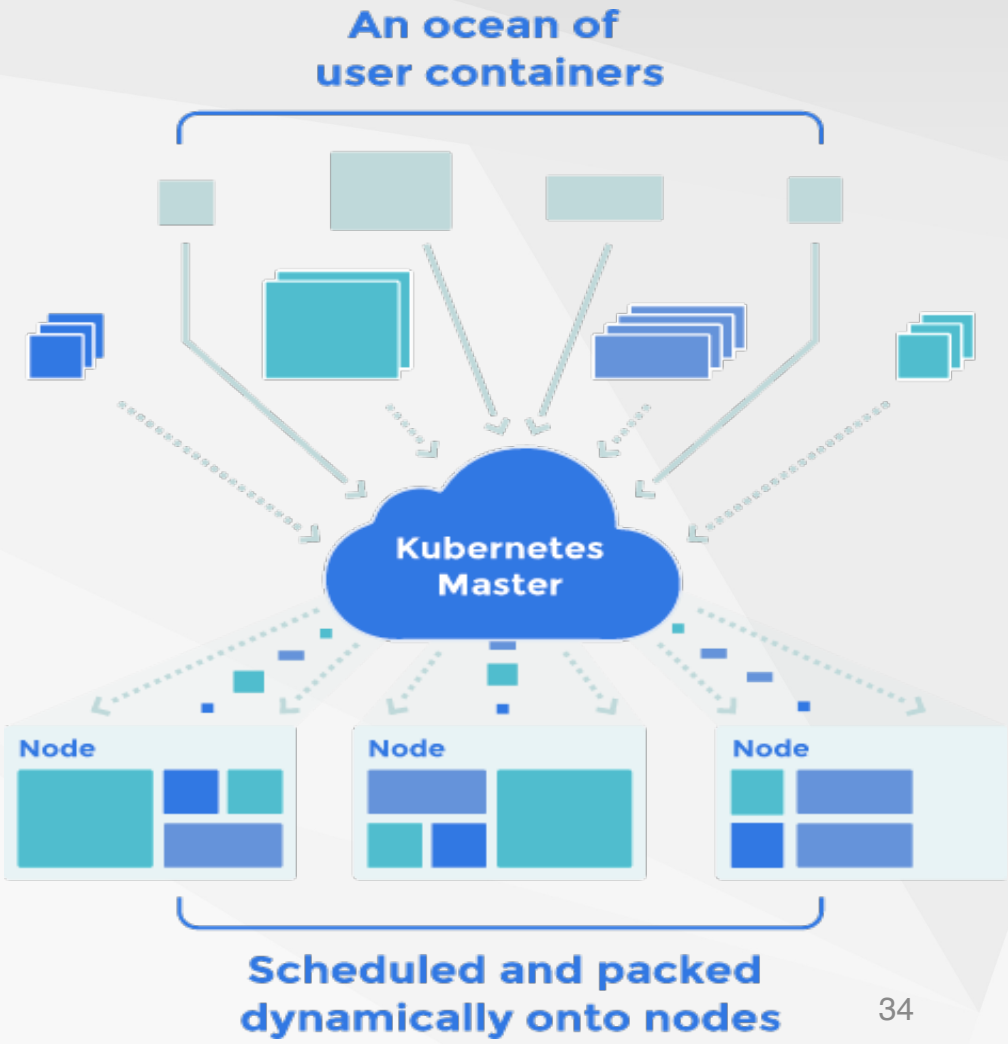
– Introduction to Microservices, Chris Richardson

Why not Docker

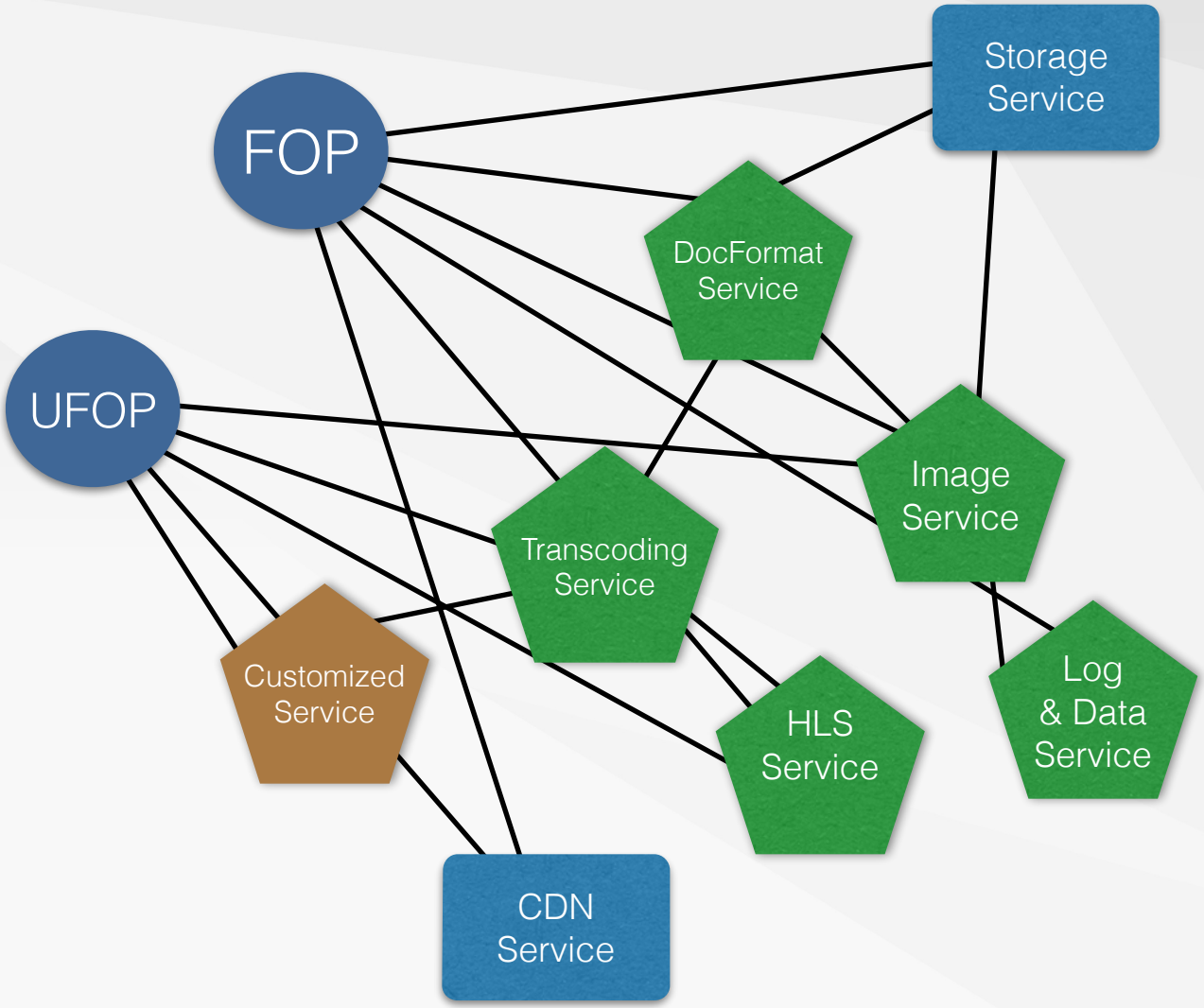
- 容器技术提供的隔离性，物理边界机制和微服务的理念是契合的。
- 拆细/封装了环境和实现。不同的技术，不需要知道里面是怎样的，只关心接口。
- 然而还不够，没有解决服务运维问题。是以出现官方的工具链。
- 没有运维的解决方案，如同空中楼阁。
- Kubernetes, 谷歌去年公开的容器编排系统。领先十年，omega/borg 论文分量。
- 旨在解决容器运维问题。定义状态，而非关心过程。
- 其他问题：存储、网络 硬伤。

Kubernetes

- Master
- Minion(Kubelet)
- Pods
- Replication Controller
- Labels
- Service - Proxy



FOP as-nanoservices



THANKS