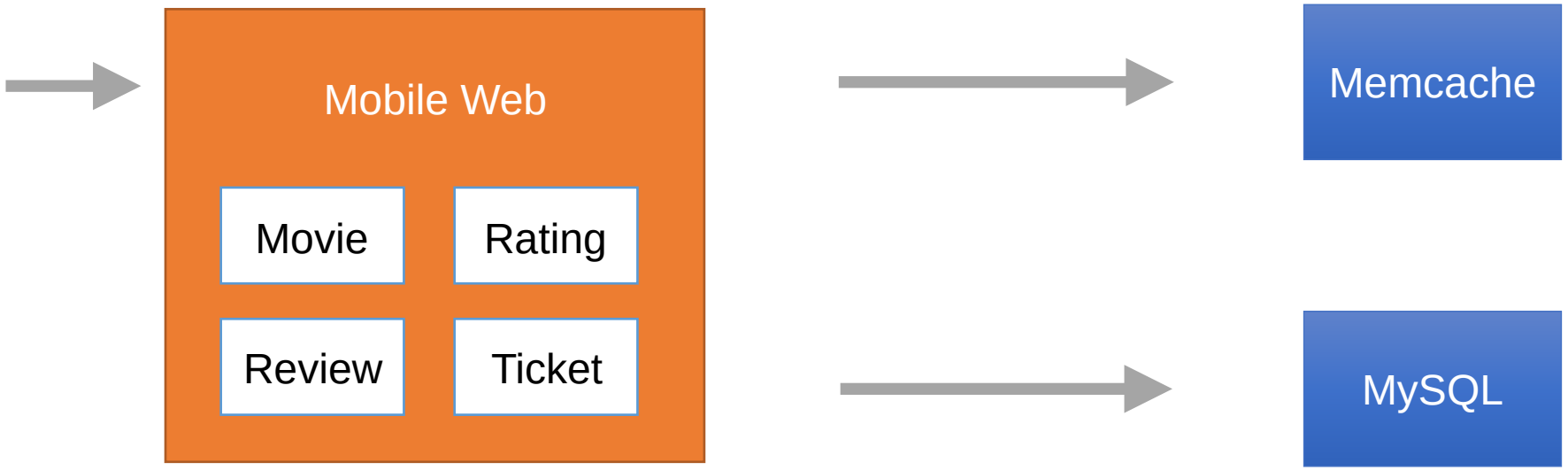
使用 Python 构建服务化基础设施

张华翼

豆瓣

- 背景
- 服务化的问题
- 如何定位性能问题
- 如何优化性能
- 总结

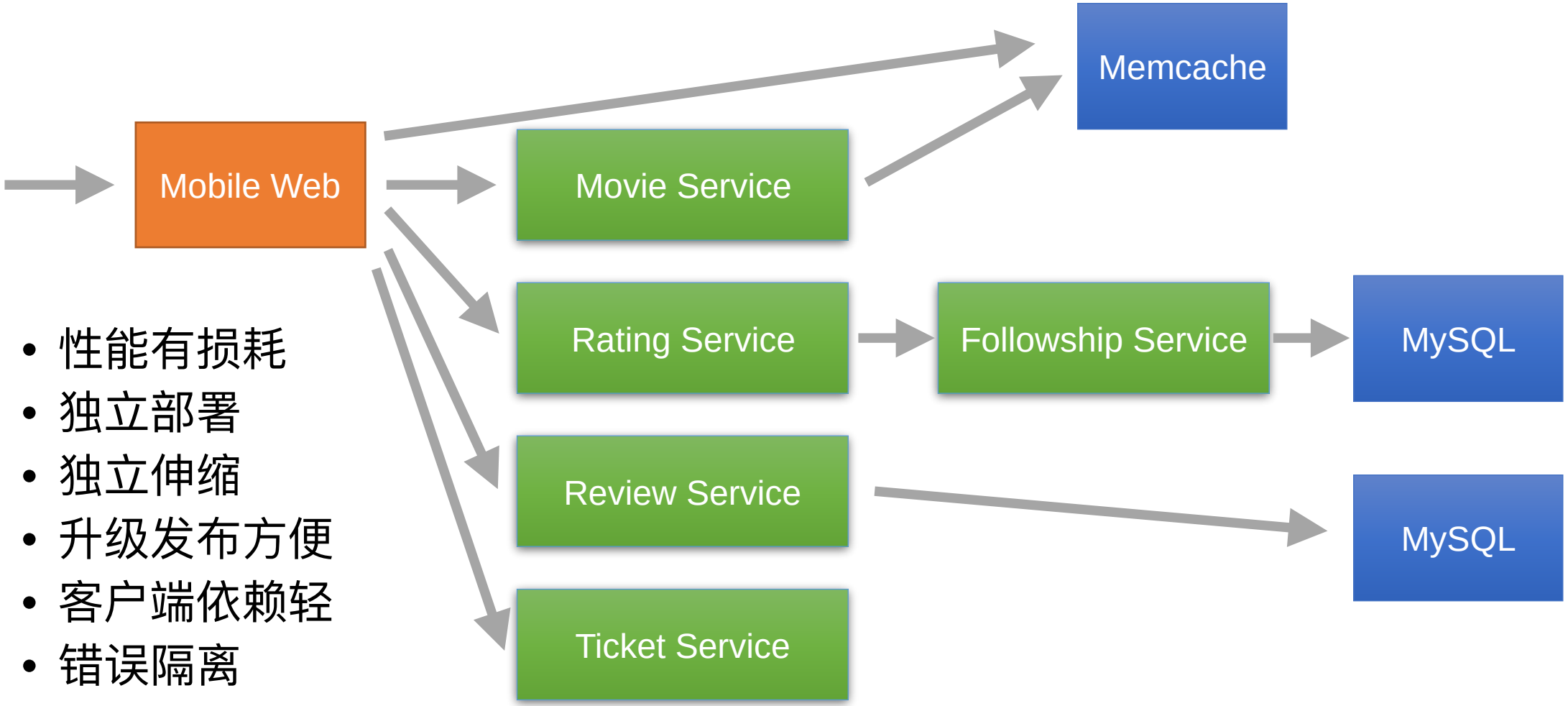
一体化架构



- 性能无损耗
- 无法独立上线
- 无法独立伸缩

- 依赖混乱
- 局部错误导致全站不可用

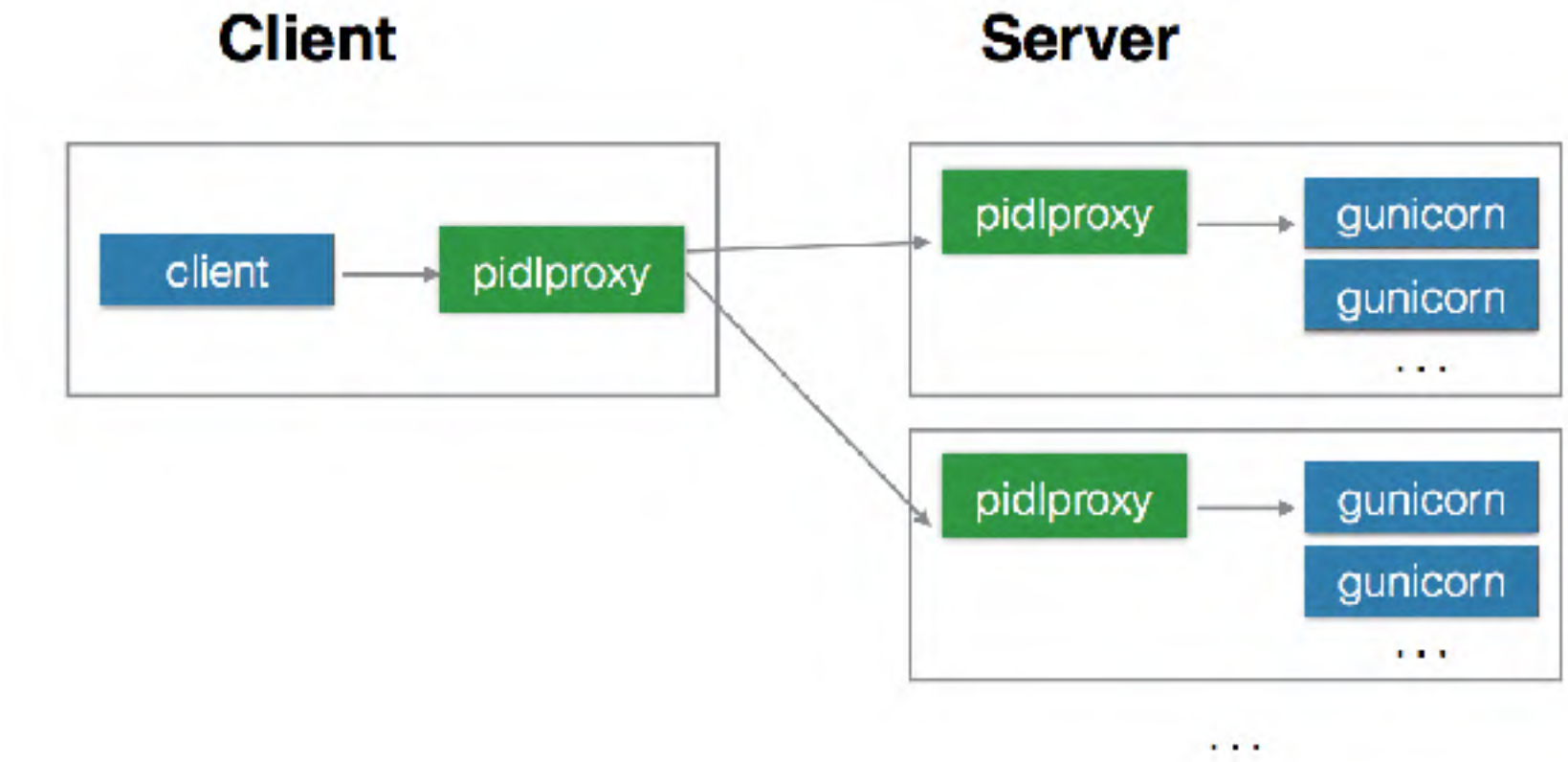
服务化架构



- 性能有损耗
- 独立部署
- 独立伸缩
- 升级发布方便
- 客户端依赖轻
- 错误隔离
- 跨语言

- **服务特点**

- 分布式
- 无状态
- 长连接



- **网站变慢了怎么办?**

- 一体化架构

  一次请求在一个进程内完成

  工具：cProfile, pstats, gprof2dot, pyflame

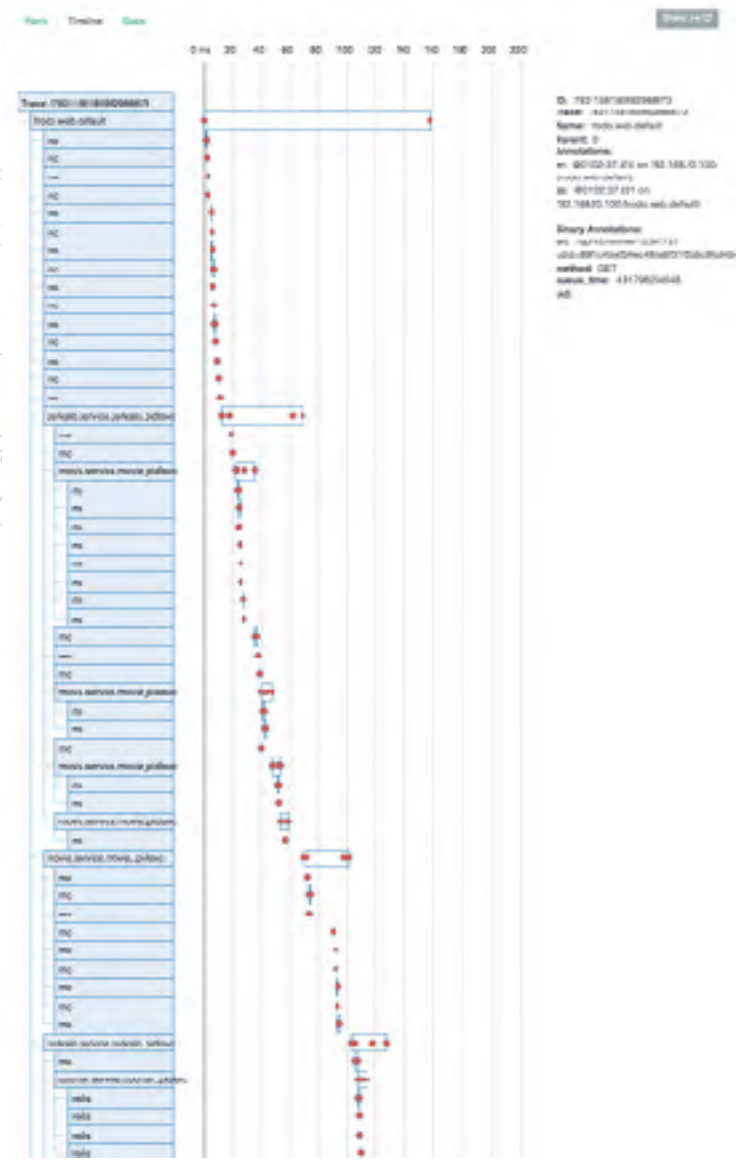
- 服务化架构

  分布式系统难以追踪，调用链复杂

  如何定位性能问题?

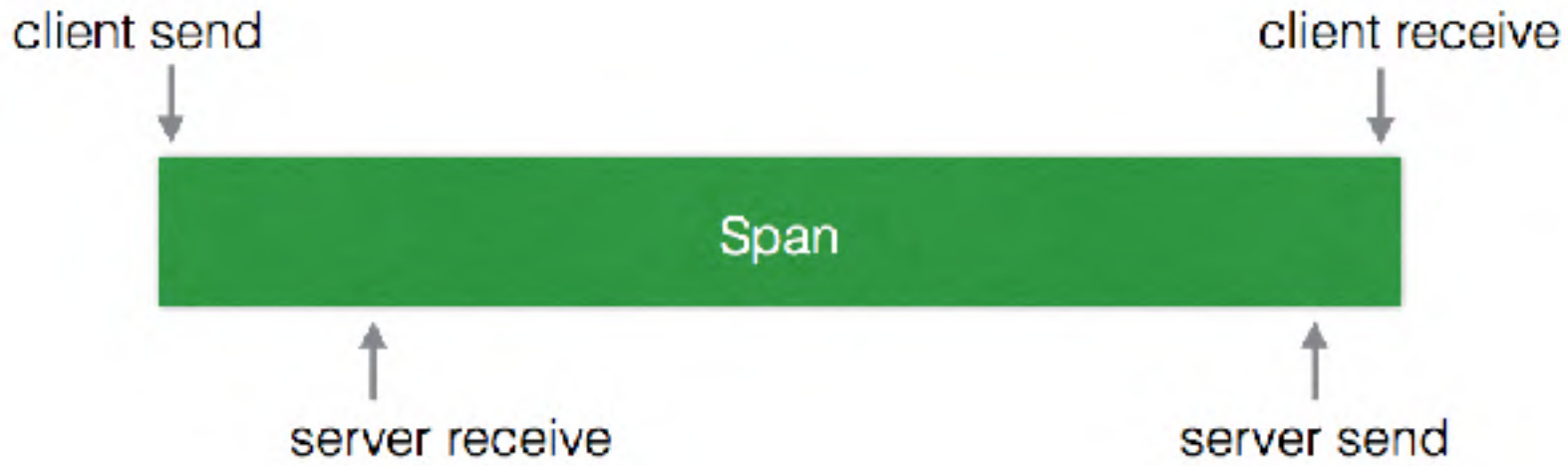
- **Shuai**

- 分布式请求追踪系统
- Google Dapper, Twitter Zipkin
- 采集调用信息，0.1%采样率
- 可视化统计分析性能瓶颈



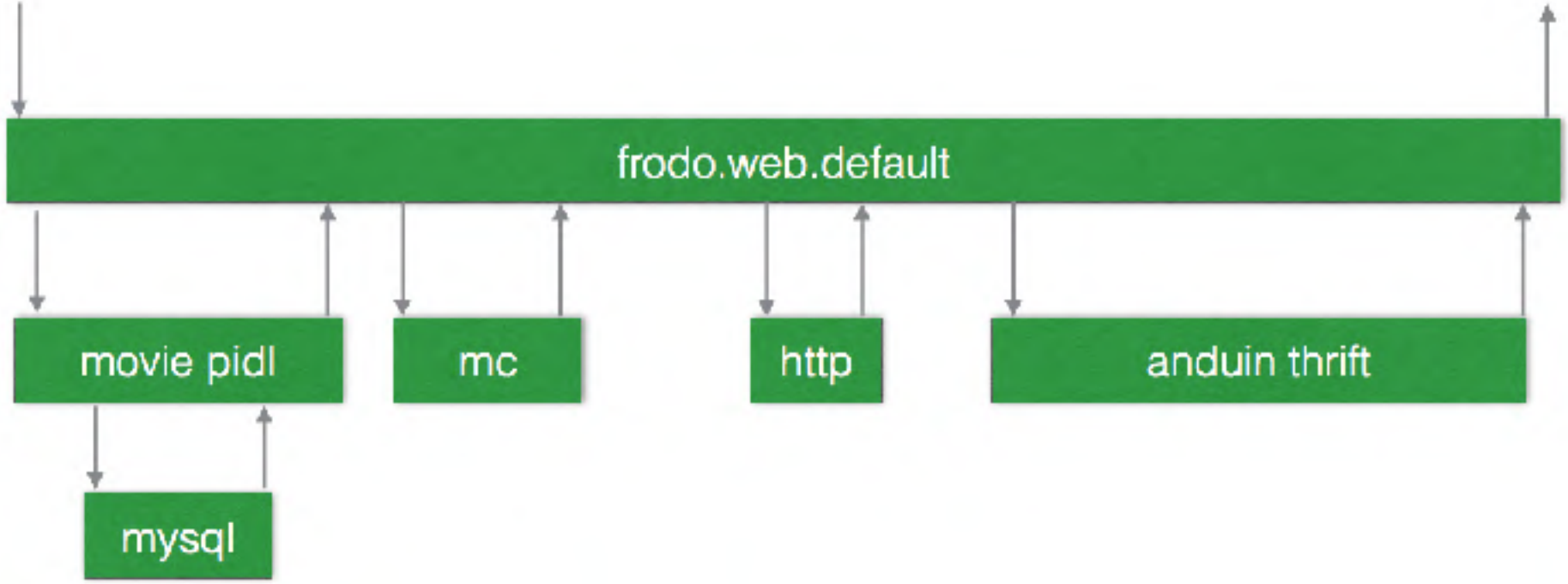
- **Span**

每个请求对应一个 Trace
每个服务调用对应一个 Span
将 timestamp, ip, pid 等信息编码为 int 作为 SpanId



- **Trace**

通过 TraceId, ParentId 将 Span 串起来得到完整的树状调用链



- **TraceId, ParentId 信息如何传递**

- Thrift: zipkin, finagle, upgrade protocol
- PIDL: plugin
- HTTP: headers
- 基础服务（MySQL, Memcache, Redis 等）

- **实现目标**

  - 健壮，对应用无影响
  - 开销低
  - 对开发者透明
  - 实时查询

- **如何实现透明：monkey patch**

python2: httplib.HTTPConnection

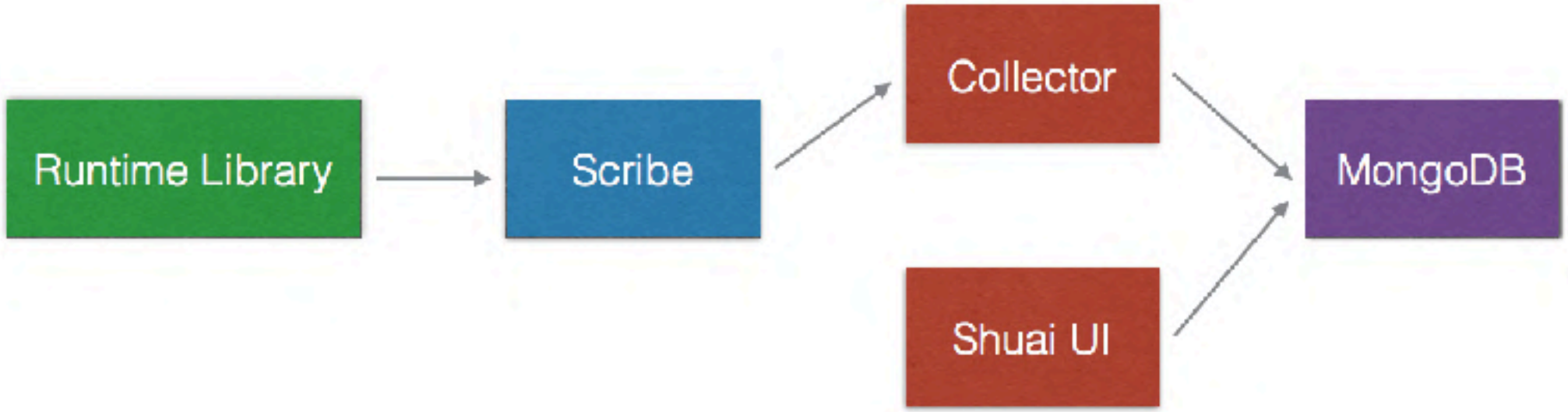
python3: http.client.HTTPConnection

HTTPConnection.endheaders, HTTPConnection.getresponse

基础服务: 重写 client 的 get, execute, __getattr__ 等方法

统一的私有云部署：Gunicorn hook

- **Shuai Architecture**

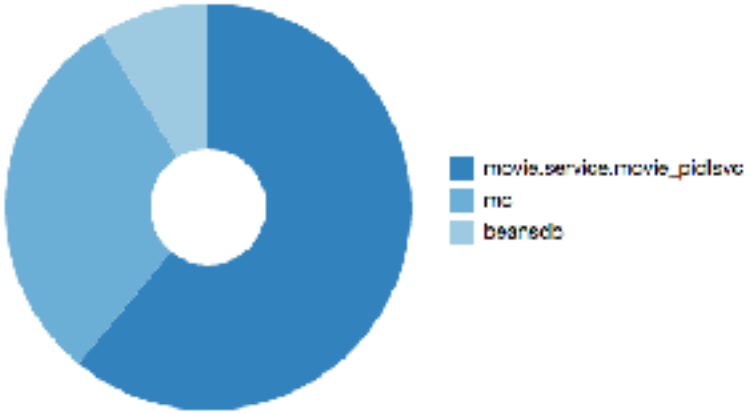


- 一个请求的统计

Count Stats

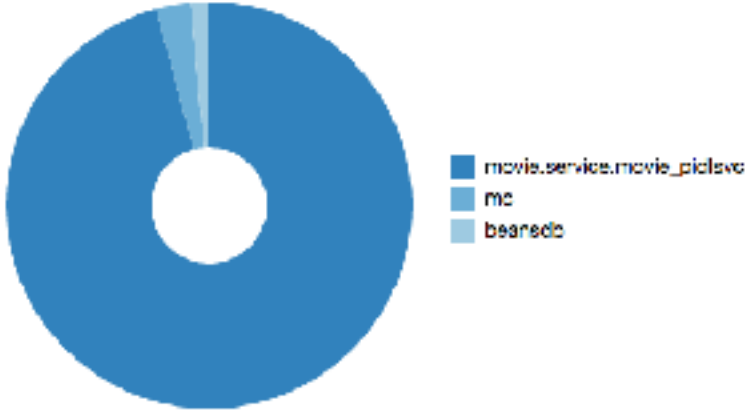
#	Span Type	Count
1	movie.service.movie_pidsvc	160
2	mc	69
3	beansdb	26

Direct Child Count: 295



Duration Stats

#	Span Type	Duration (ms)
1	movie.service.movie_pidsvc	1515
2	mc	47
3	beansdb	24



- **性能下降的主要原因**

- 函数调用 -> RPC 调用
- 网络请求：连接，传输
- 协议，编码解码
- 重新创建对象（无状态服务）
- 频繁 RPC 调用

- **性能下降的主要原因**

```
movies = get_movies(ids)                -> server1
return [{‘title’: m.title(),            -> server2
         ‘rating’: m.rating()}          -> server3
         for m in movies]
```

- n * k 次 RPC 调用
- 为这样的需求新增接口?

- **Redeye**

- 声明式的动态数据接口
- 将 client 对数据的操作转为 AST 在 server 执行
- Control Flow -> Data Flow
- Pull Data vs Push Computation?
- REST vs GraphQL?

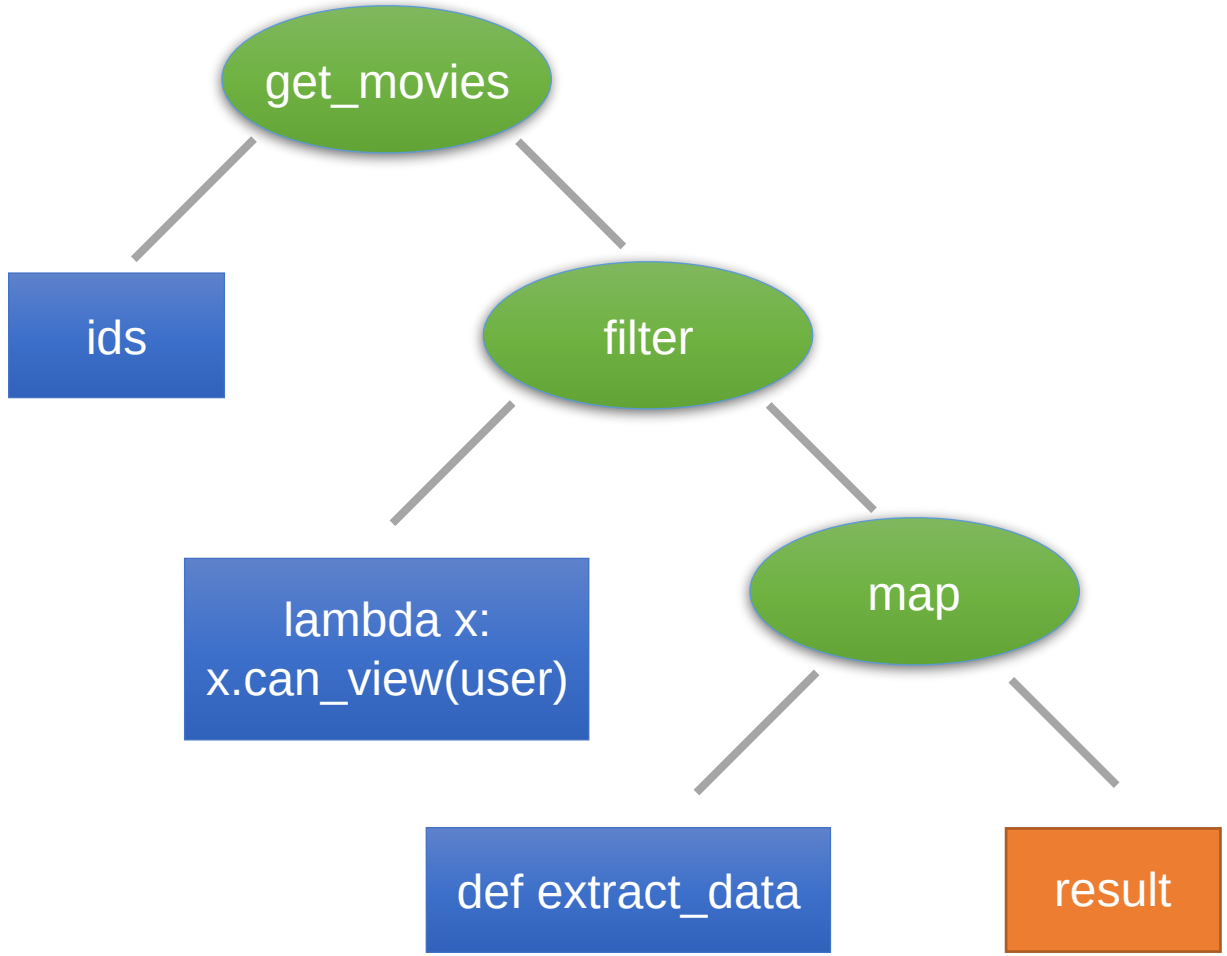
- **Data Flow**

```
query(movie_client)\
    .get_movies(ids)\
    .filter(lambda x: x.can_view(user))\
    .map(lambda x: {‘id’: x.id,
                    ‘title’: x.title(),
                    ‘rating’: x.rating()})\
    .execute()
```

只有 1 次 RPC 调用！ 800ms -> 100ms

- **构建抽象语法树**

limit, filter, map,
reduce, attr, call,
validate, join, cache,
warm, page, length,
…



- **序列化一个函数**

```
code = func.__code__
closure = func.__closure__
globals = func.__globals__
defaults = func.__defaults__
```

pickle recursively

marshal

```
f = types.FunctionType(
    code, globals, “redeye_closure”, defaults, closure)
```

- **重建闭包**

```
closure = tuple(cell.cell_contents
                for cell in func.__closure__)

def make_cell(v):
    return (lambda x: v).__closure__[0]
c = tuple(make_cell(v) for v in closure)
```

- **Sandbox 保证封装性**

```
__builtins__ + closure + interfaces

_blacklist = {
    '__import__', 'compile', 'eval', 'execfile',
    'exit', 'file', 'open', 'quit', 'raw_input', 'reload',
    'repr'
}
```

- **与服务熔断保护结合**

以文件名+行号标识新接口

```
current_frame = sys._getframe()
calling_f = current_frame.f_back.f_back
fn = calling_f.f_code.co_filename
lineno = calling_f.f_lineno
path_identifier = os.path.abspath(fn)

key = ‘{0}_{1}’.format(path_identifier, lineno)
```

总结

- Shuai 分布式追踪系统
- Redeye 动态数据接口
- 不只是追踪和性能
- 服务依赖关系，指标统计，监控，报警，…
- 由一组原子接口灵活组装出新接口

谢谢观看

irachex

zhy@douban.com