分布式系统

缓存设计浅析

朋春
pengchun@taobao.com

为什么要做这个分享?

缓存系统（2011.6）

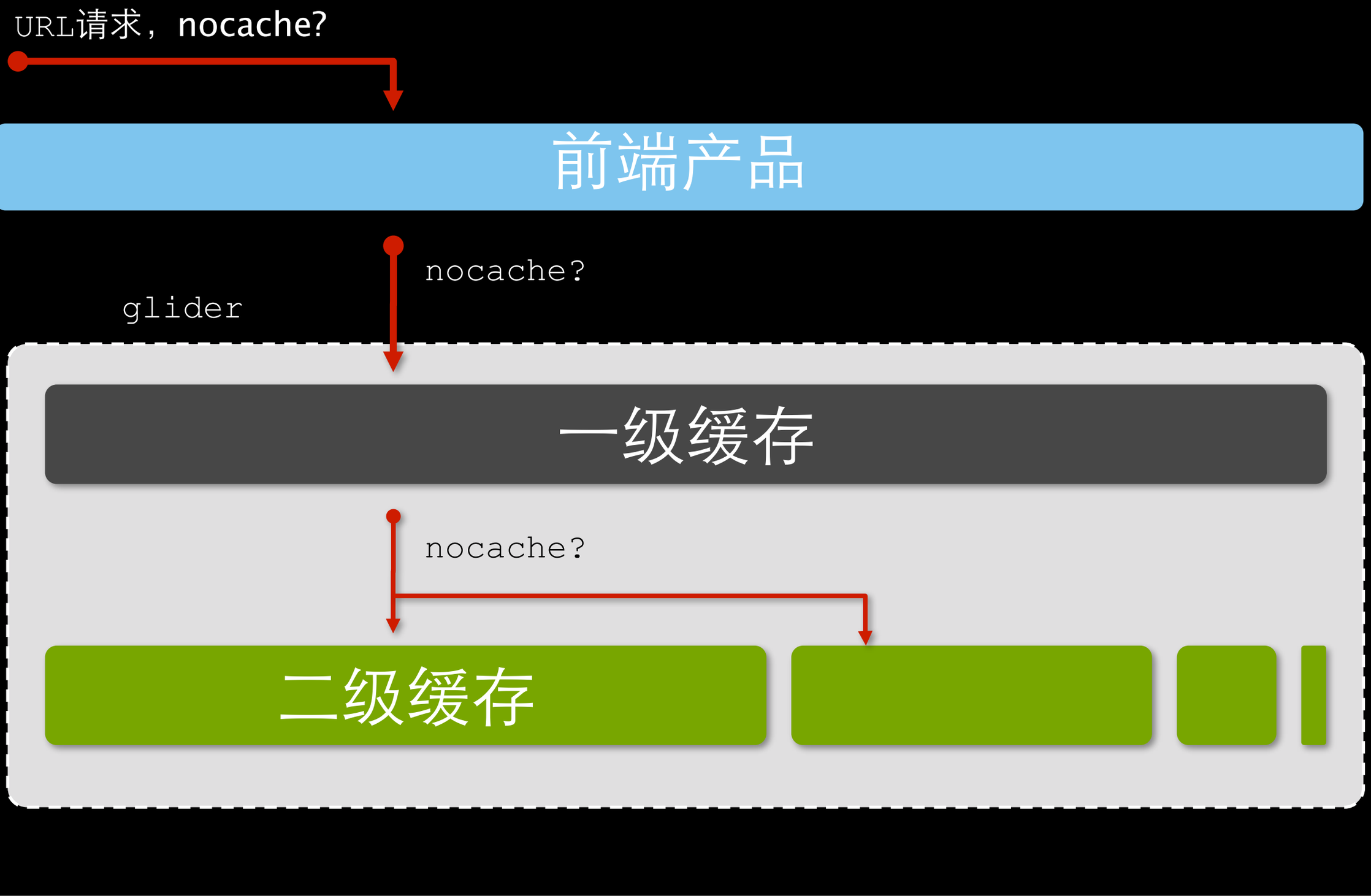
前端产品

glider

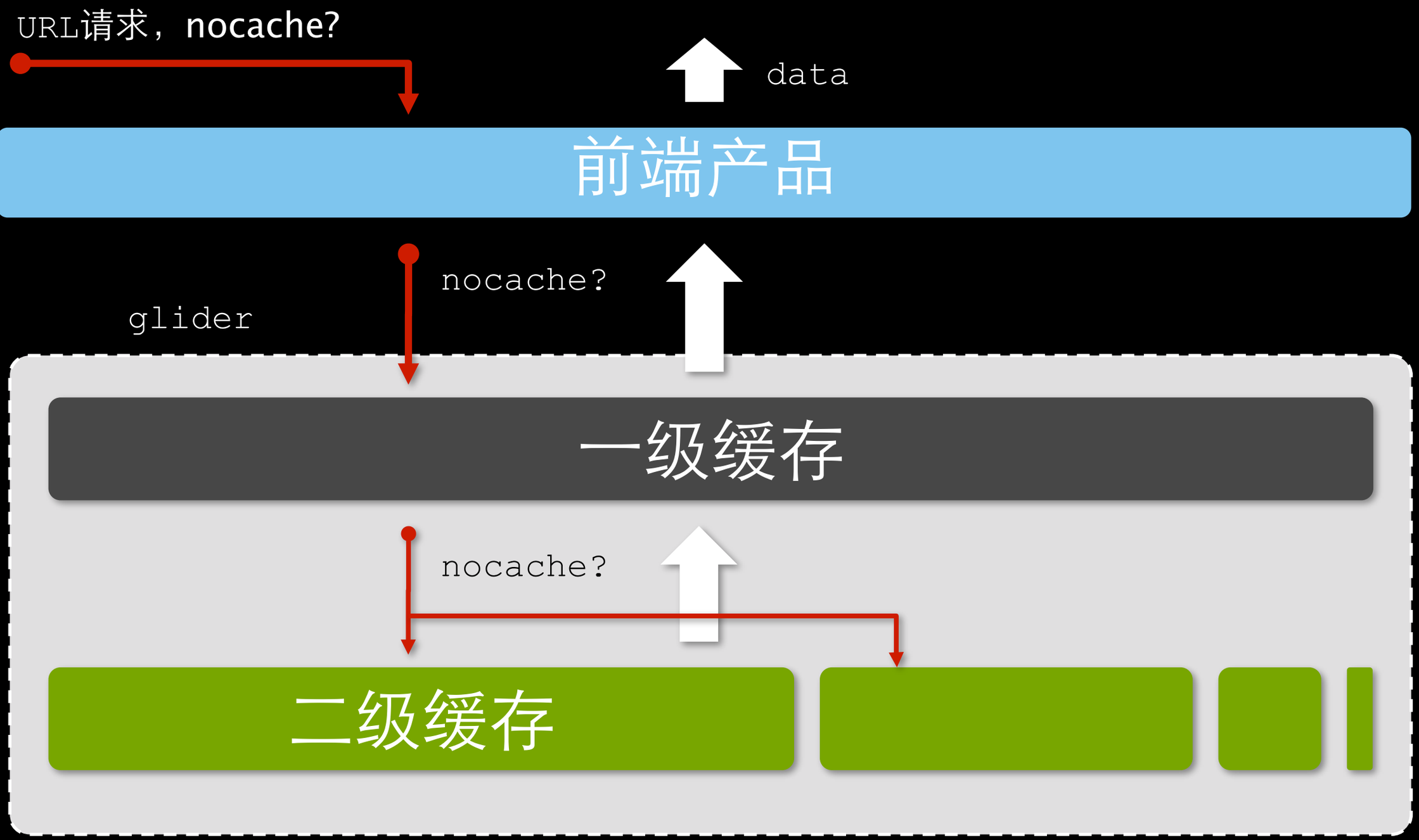
一级缓存

二级缓存

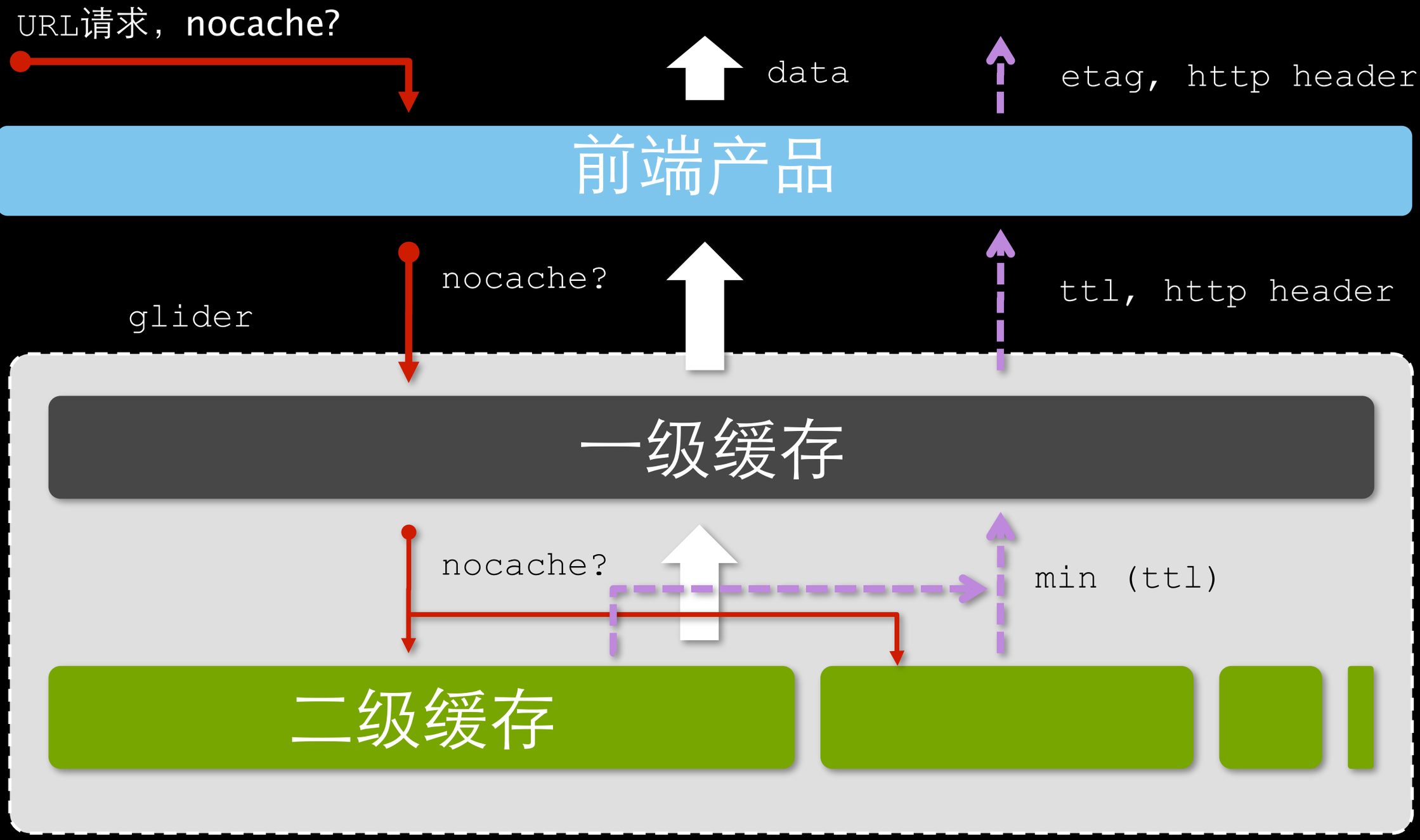
缓存系统（2011.6）



缓存系统（2011.6）



缓存系统（2011.6）


URL请求，nocache?
data
etag, http header
前端产品
nocache?
glider
ttl, http header
一级缓存
nocache?
min (ttl)
二级缓存

时代变了

- 容灾考虑，缓存多实例，多写单读
- 实时的数据更新
- 更多更复杂的底层系统

缓存系统的三个问题

- 数据一致性问题
- ~~缓存雪崩问题~~
- ~~缓存穿透问题~~

数据一致性

- 缓存与底层数据的一致性
- 有继承关系的缓存之间的一致性
- 多个缓存副本之间的一致性

缓存数据的淘汰

- “预估”失效时间
- 单调递增的数据版本号
- 提供清理接口

精细化管理

- 只淘汰该淘汰的

itier的设计（通用）

- 根据tag进行缓存管理
- 时间戳做数据版本号
- 提供清理API
- 版本号共享协作

```
var me = Cache.create(...);
me.set(key, value, ttl, tags);
me.get(key);
me.tagrm(tag, offset, flush);
```

缓存的数据结构

```
var data = {
  'i':now,          /** 数据写入时间戳 */
  'e':now + ttl,/** 预期过期时间 */
  'k':key,          /** 原始key */
  'v':value,        /** 原始值  */
  't':tags          /** tag列表 */
};
```

tagrm (cleanByTag)

```
/**
 * @定时同步
 */
var __taginfo = {};
tagrm (tag                    ) {
  __taginfo[tag] = now()              ;
  // flush to zookeepr or mysql ...
}
```

itier中的tag定义

- __global__
- 数据来源driver名字
- SQL中的表名

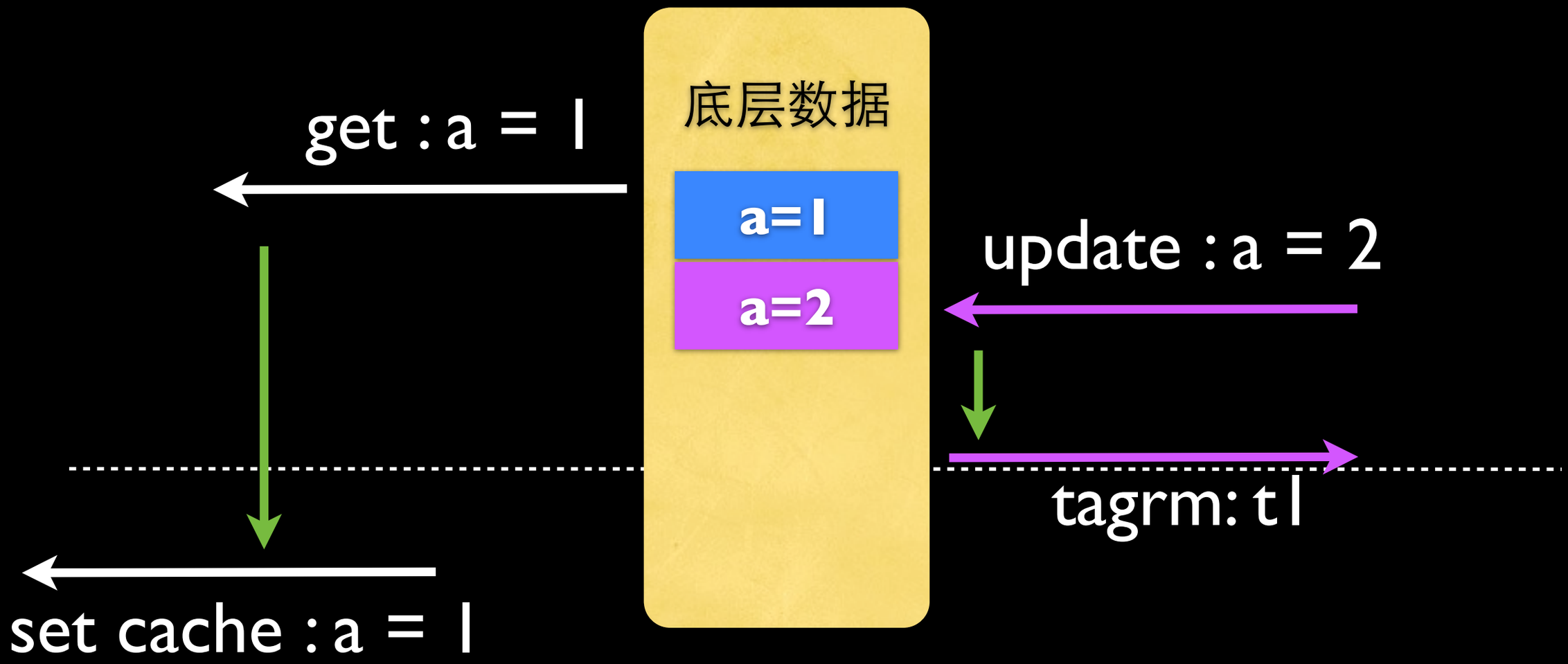
一致性目标

- 缓存的tag信息一路继承，取并集
- 缓存的ttl信息一路继承，取最小值
- 秒量级保证一致性

但是，

简单的事情从来不简单

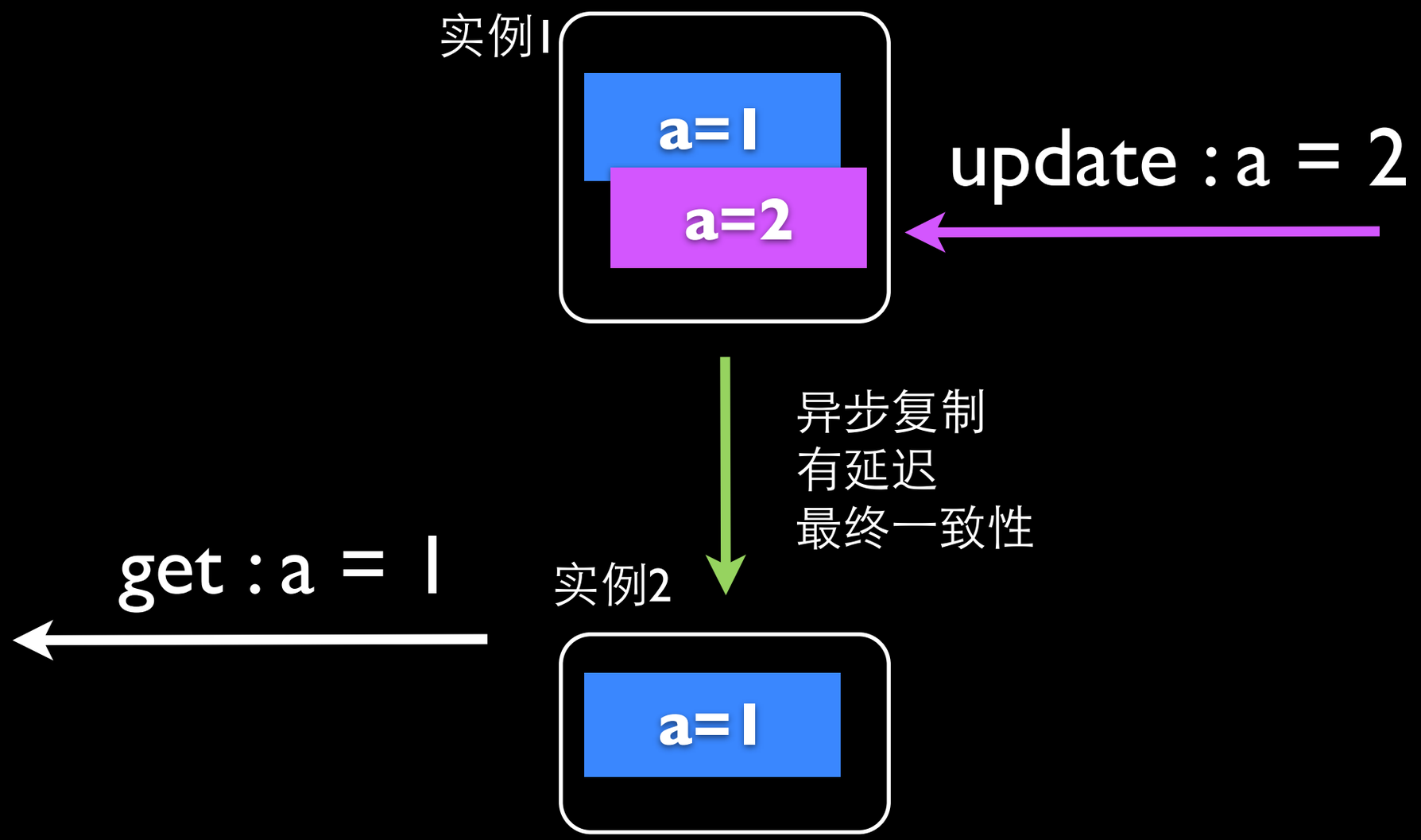
case1： 网络延迟



case2：多机时差

- tagrm在机器1上调用，取机器1上的时间
- 写缓存在机器2上，取机器2上的时间
- 机器1和机器2在ms级别时间不同步

case3： 异步复制



tagrm (cleanByTag)

```
/**
 * @定时同步
 */
var __taginfo = {};
tagrm (tag, offset, flush) {
 __taginfo[tag] = now() + offset;
 // flush to zookeepr or mysql ...
}
```

一些基本理念

- 缓存永远是锦上添花的部分
- 保证不了一致性就拉到吧
- 不要为了一致性加锁，得不偿失
- 不要过度依赖缓存
- 关注你的缓存命中率（tag粒度）

代码

- https://github.com/aleafs/node-shark/blob/master/lib/cache.js

- https://github.com/aleafs/node-shark/blob/master/test/unit/CacheTest.js