软件架构解耦

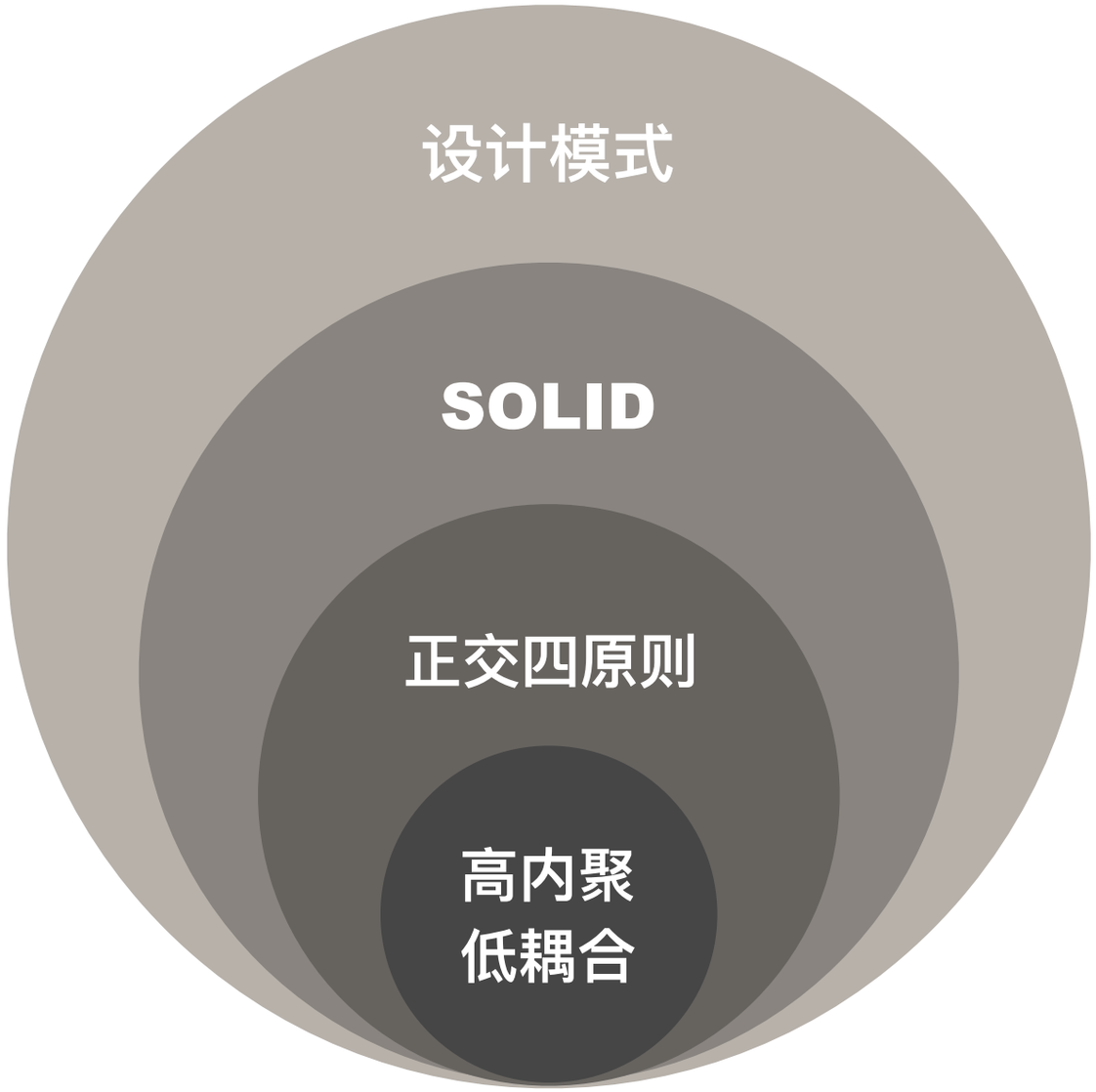
王博

内容

- 正交设计原则
- 服务间解耦设计
- 服务内解耦设计
- 演进式设计建议

正交设计原则

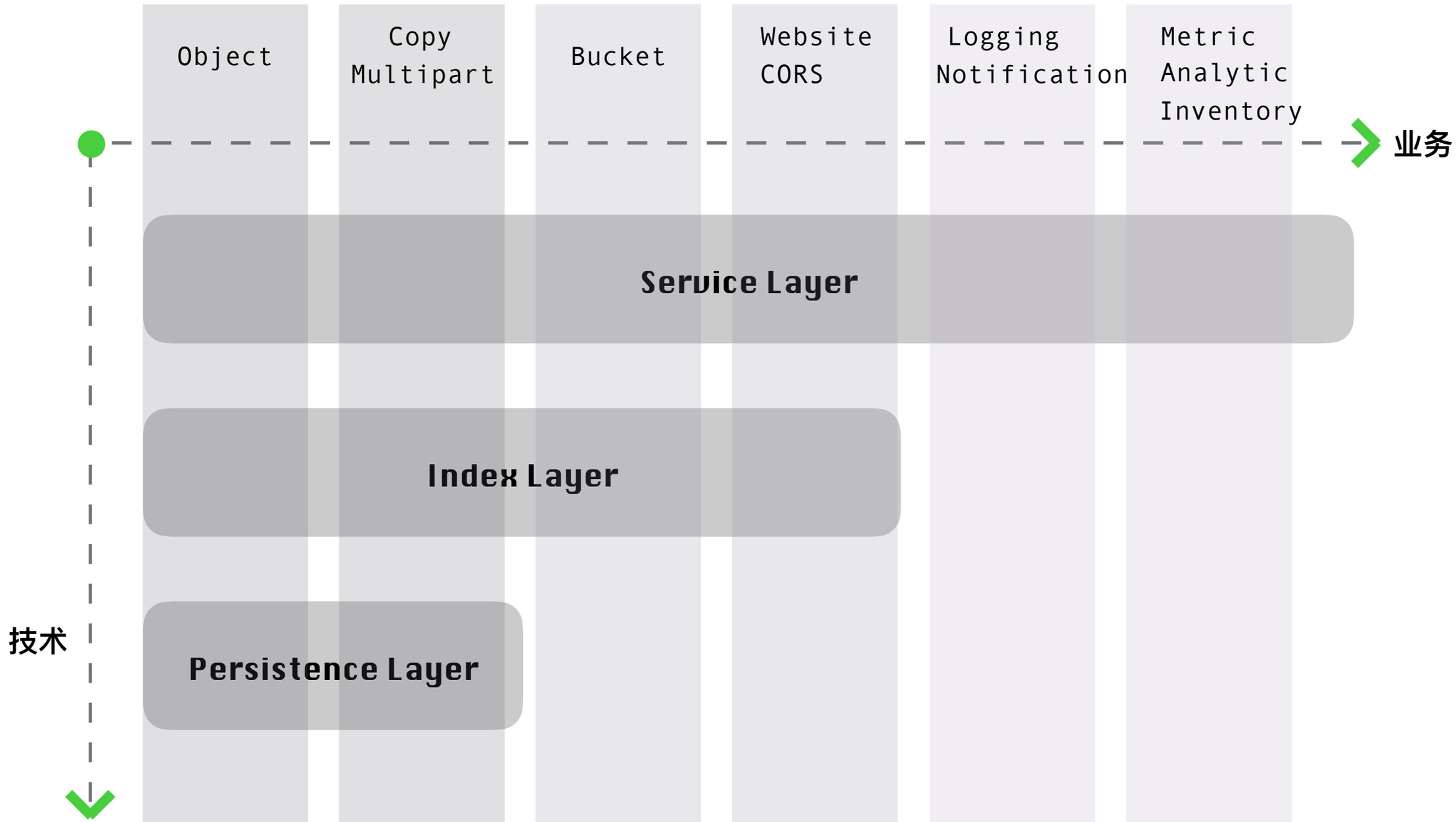
设计原则



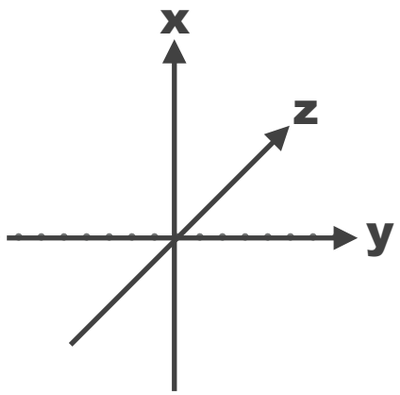
“Design is there to enable you to keep changing the software easily in the long term”

—— Kent Beck

变化方向


Object
Copy
Multipart
Bucket
Website
CORS
Logging
Notification
Metric
Analytic
Inventory
业务
Service Layer
Index Layer
Persistence Layer
技术

正交设计原则



分离变化方向

消除重复

缩小依赖范围

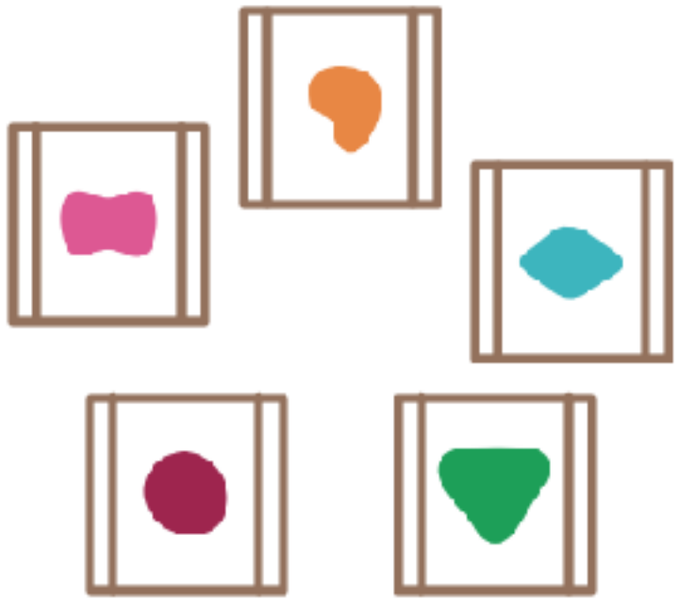
向稳定的方向依赖

1个被动策略

3个主动策略

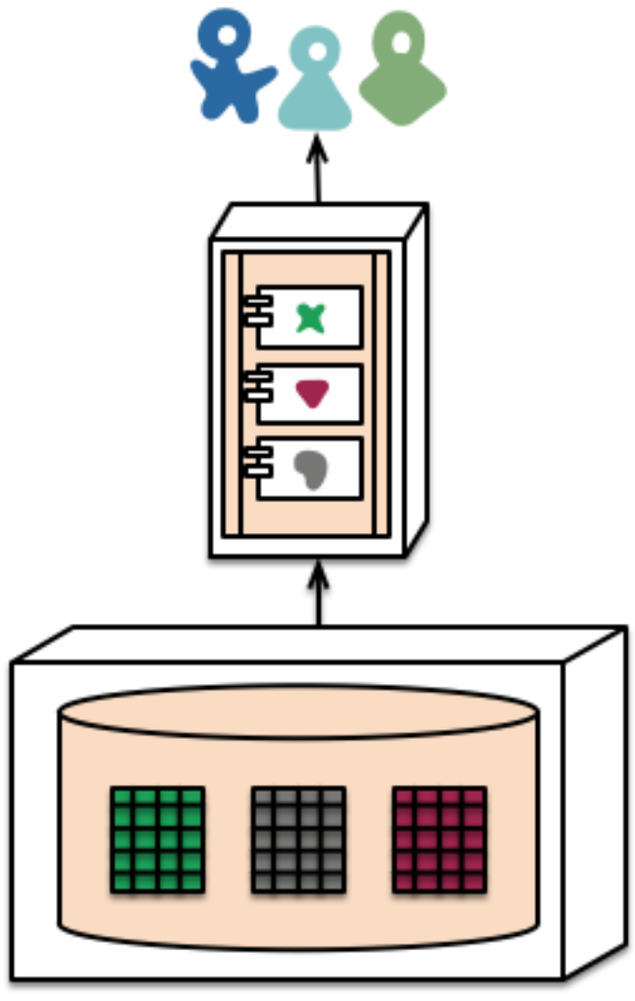
Microservices

common characteristics of this architectural style

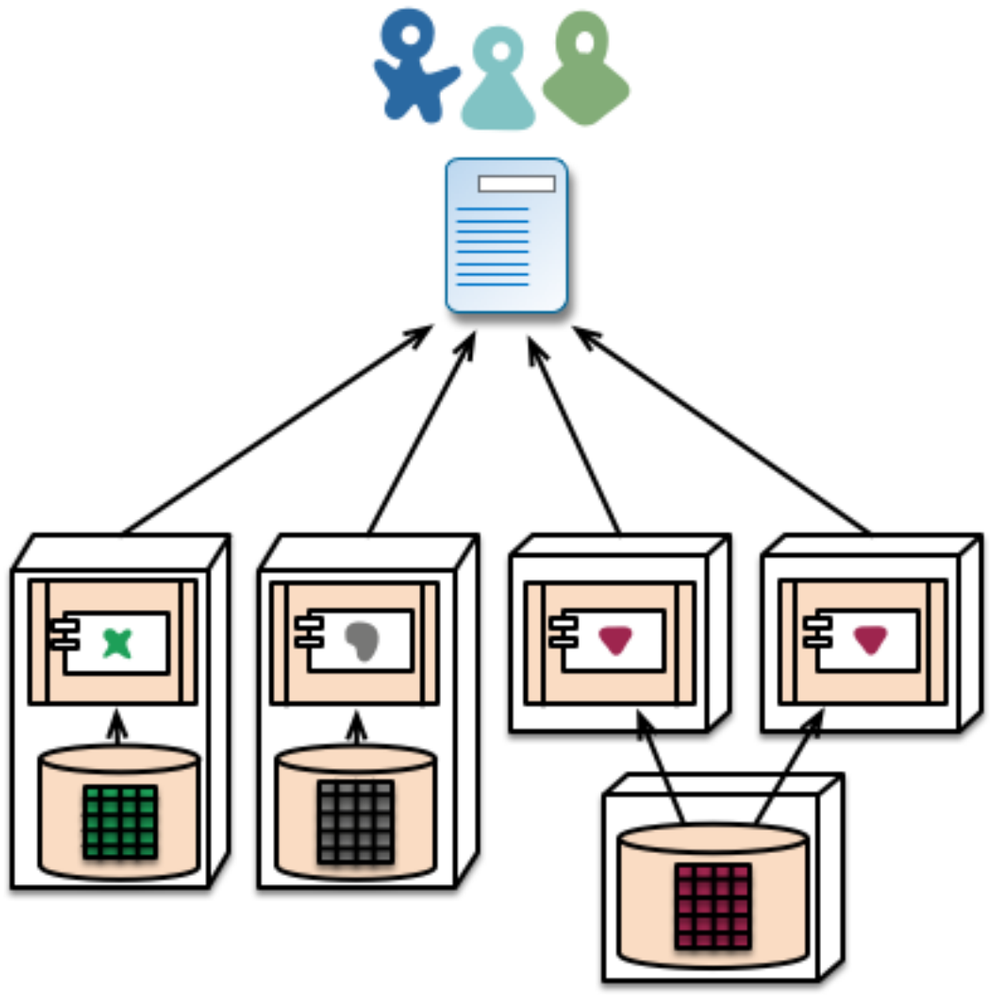


by James Lewis and Martin Fowler

MICROSERVICE

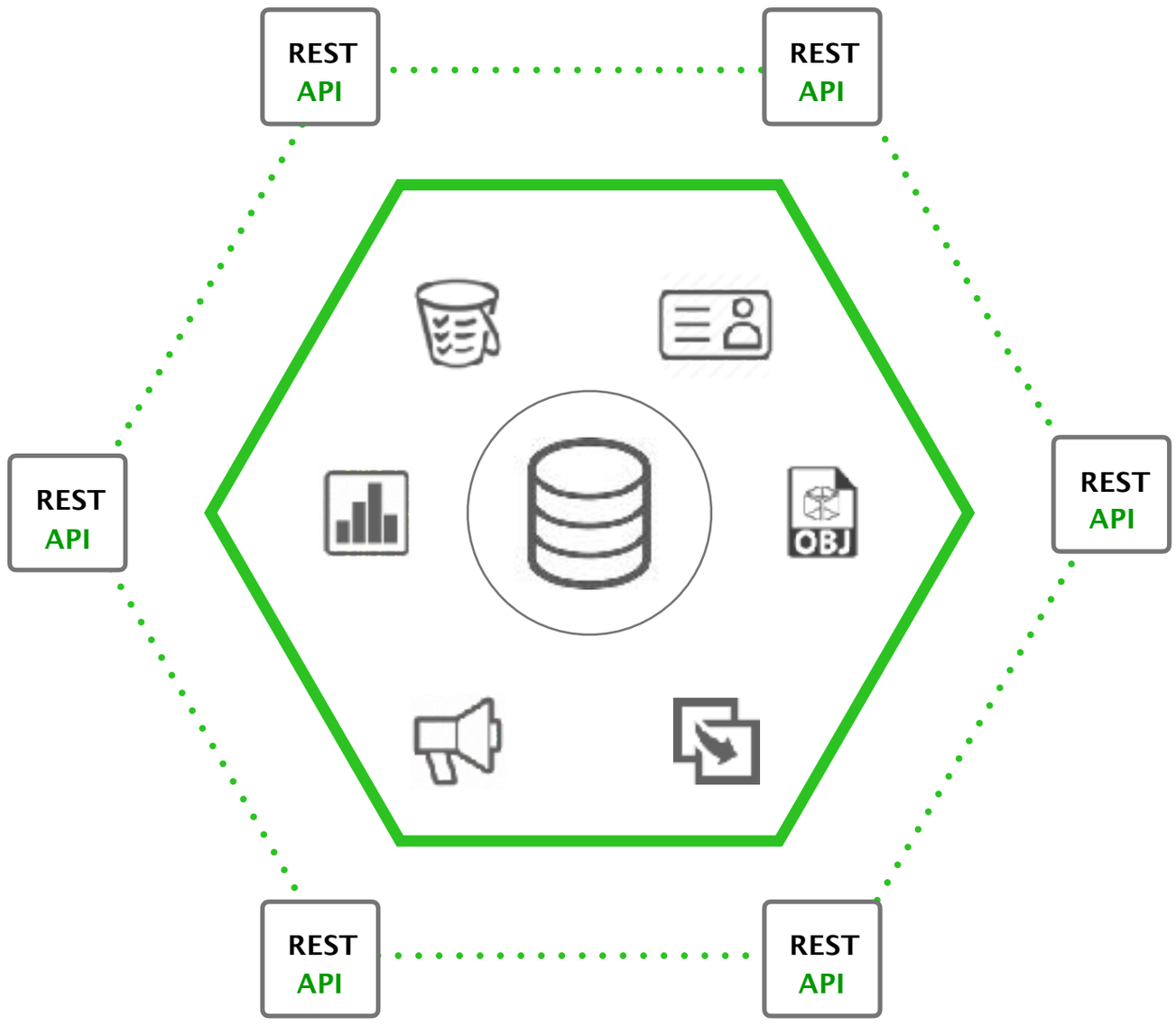


monolith - single database

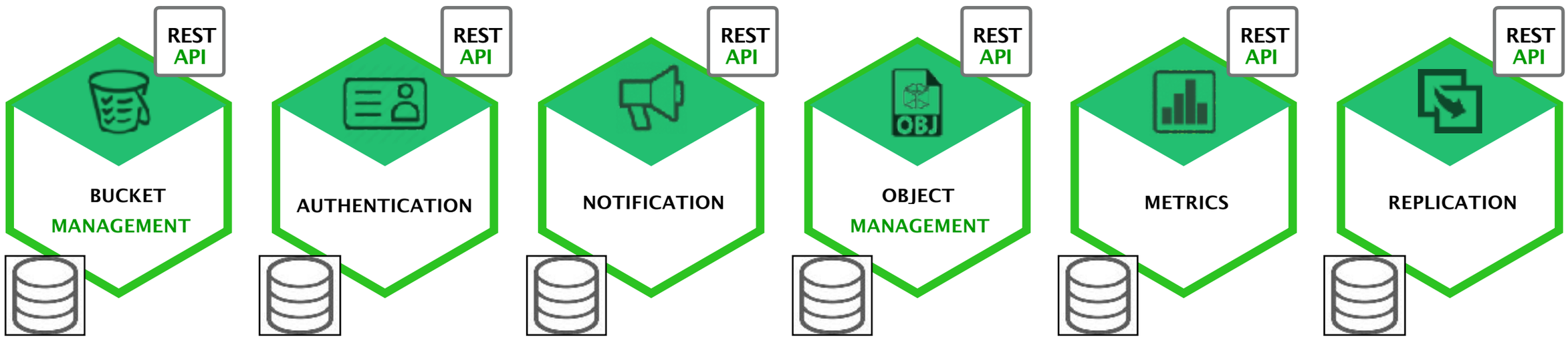


microservices - application databases

MONOLITHS



SERVICE CHARACTERS



- *Present customer value preferentially*
- *Cohesive for independence*
- *Data splitting is critical*
- *Reduce interaction*
- *Concern consistency requirements*
- *Treat performance issues reasonably*

DATA SPLITTING

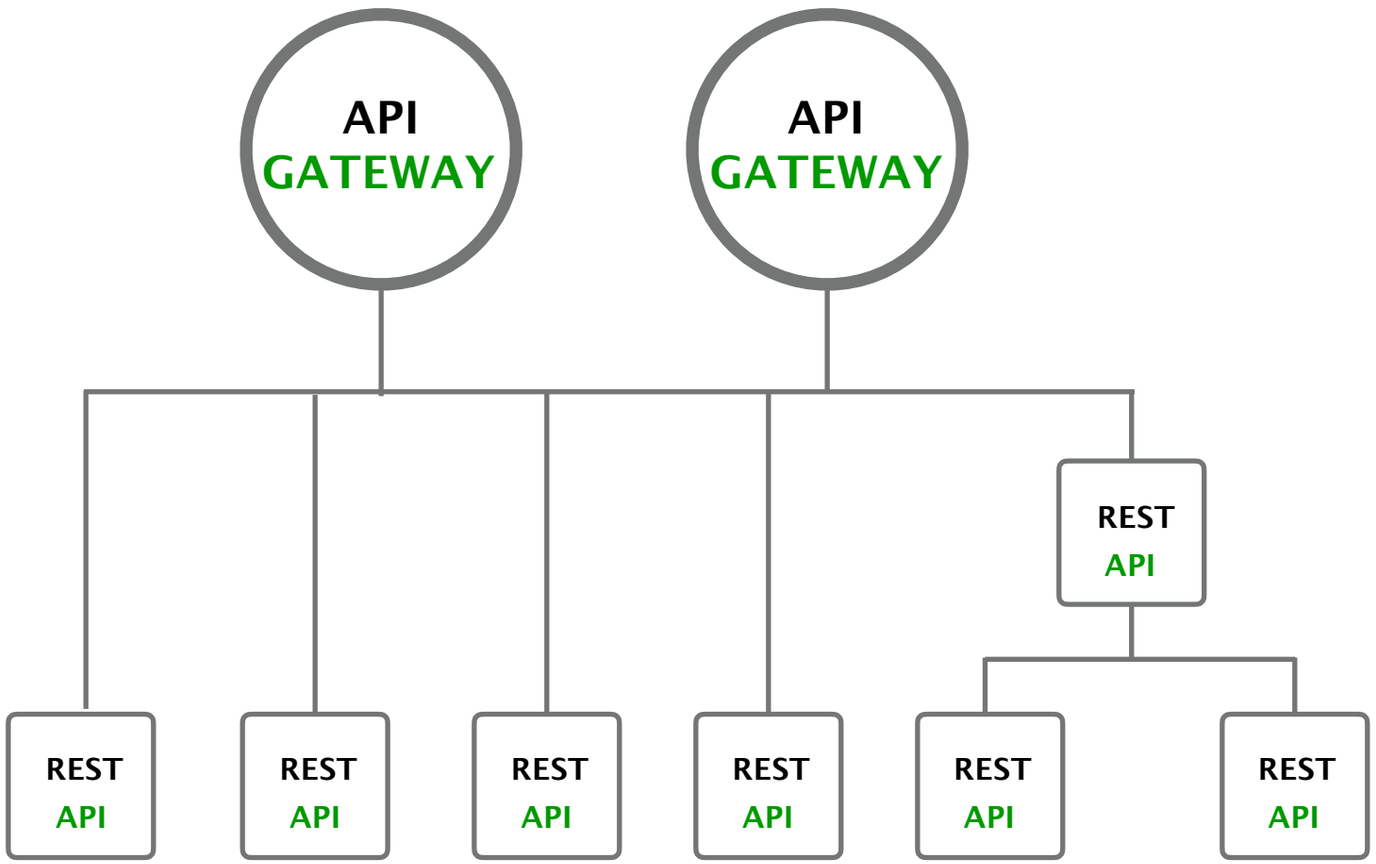
BUCKET MANAGEMENT									
	LIFECYCLE	AUTHENTICATION	WEBSITE	CORS	LOGGING	METRICS	ANALYTIC		
OBJECT MANAGEMENT	CRR								

```
Bucket :  {name, versioning, lifecycle, CRR, policy, ACL, website, CORS, logging, metrics, analytic,…}

Object :  {name, versionId, PLOG list, expire, CRR status, ACL,…}
```

- *Orthogonal Design Rules*
- *Concern consistency requirements*

API DESIGN

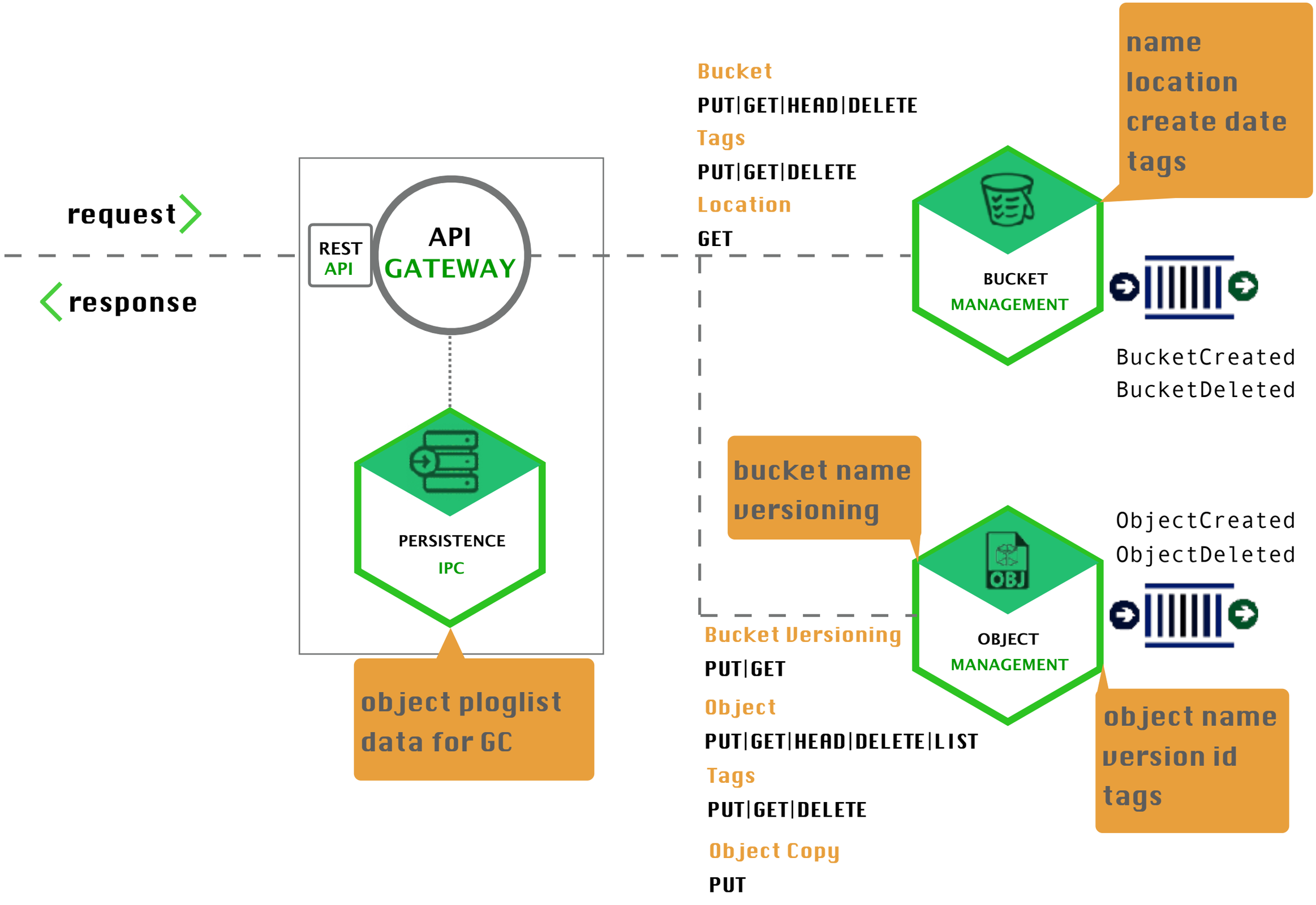


- *One size does not fit all*
- *Interface isolate principle*
- *Use facade pattern to convenient different users*
- *REST is not the only choice*
- *SYNC vs ASYNC*
- *P2P vs PUB/SUB*
- *Postel principle*
- *Idempotent design*
- *Semantic version*
- *...*

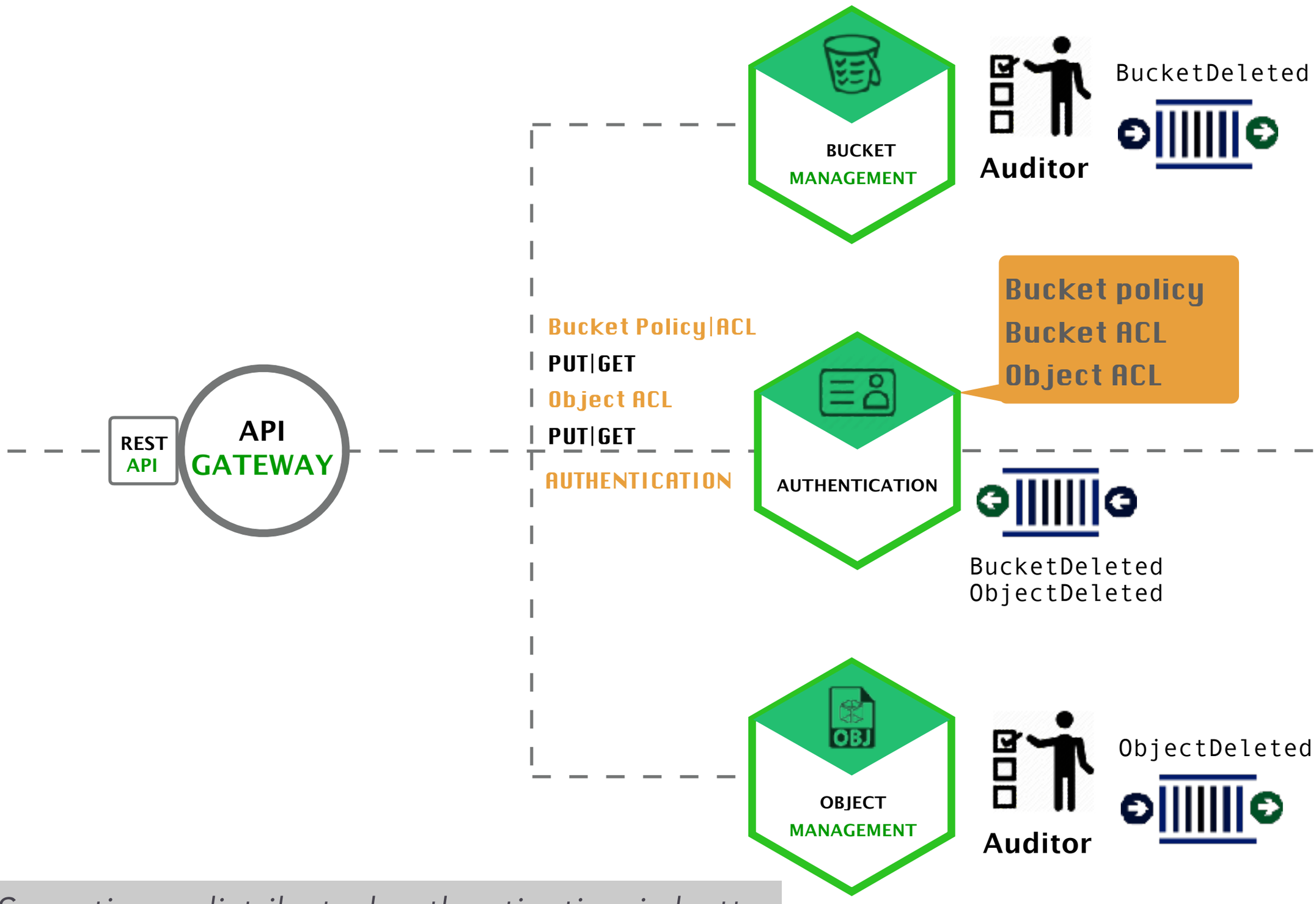
AWS S3 API

- Operations on Buckets
  - DELETE Bucket
  - DELETE Bucket analytics
  - DELETE Bucket cors
  - DELETE Bucket inventory
  - DELETE Bucket lifecycle
  - DELETE Bucket metrics
  - DELETE Bucket policy
  - DELETE Bucket replication
  - DELETE Bucket tagging
  - DELETE Bucket website
  - GET Bucket (List Objects) Version 2
  - GET Bucket accelerate
  - GET Bucket acl
  - GET Bucket analytics
  - GET Bucket cors
  - GET Bucket inventory
  - GET Bucket lifecycle
  - GET Bucket location
  - GET Bucket logging
  - GET Bucket metrics
  - GET Bucket notification
  - GET Bucket Object versions
  - GET Bucket policy
  - GET Bucket replication
  - GET Bucket requestPayment
  - GET Bucket tagging
  - GET Bucket versioning
  - GET Bucket website
- HEAD Bucket
- List Bucket Analytics Configurations
- List Bucket Inventory Configurations
- List Bucket Metrics Configurations
- List Multipart Uploads
- PUT Bucket
- PUT Bucket accelerate
- PUT Bucket acl
- PUT Bucket analytics
- PUT Bucket cors
- PUT Bucket inventory
- PUT Bucket lifecycle
- PUT Bucket logging
- PUT Bucket metrics
- PUT Bucket notification
- PUT Bucket policy
- PUT Bucket replication
- PUT Bucket requestPayment
- PUT Bucket tagging
- PUT Bucket versioning
- PUT Bucket website
- Operations on Objects
  - Delete Multiple Objects
  - DELETE Object
  - DELETE Object tagging
  - GET Object
  - GET Object ACL
  - GET Object tagging
  - GET Object torrent
  - HEAD Object
  - OPTIONS object
  - POST Object
  - POST Object restore
  - PUT Object
  - PUT Object - Copy
  - PUT Object acl
  - PUT Object tagging
  - Abort Multipart Upload
  - Complete Multipart Upload
  - Initiate Multipart Upload
  - List Parts
  - Upload Part
  - Upload Part - Copy

BASE SKETCH

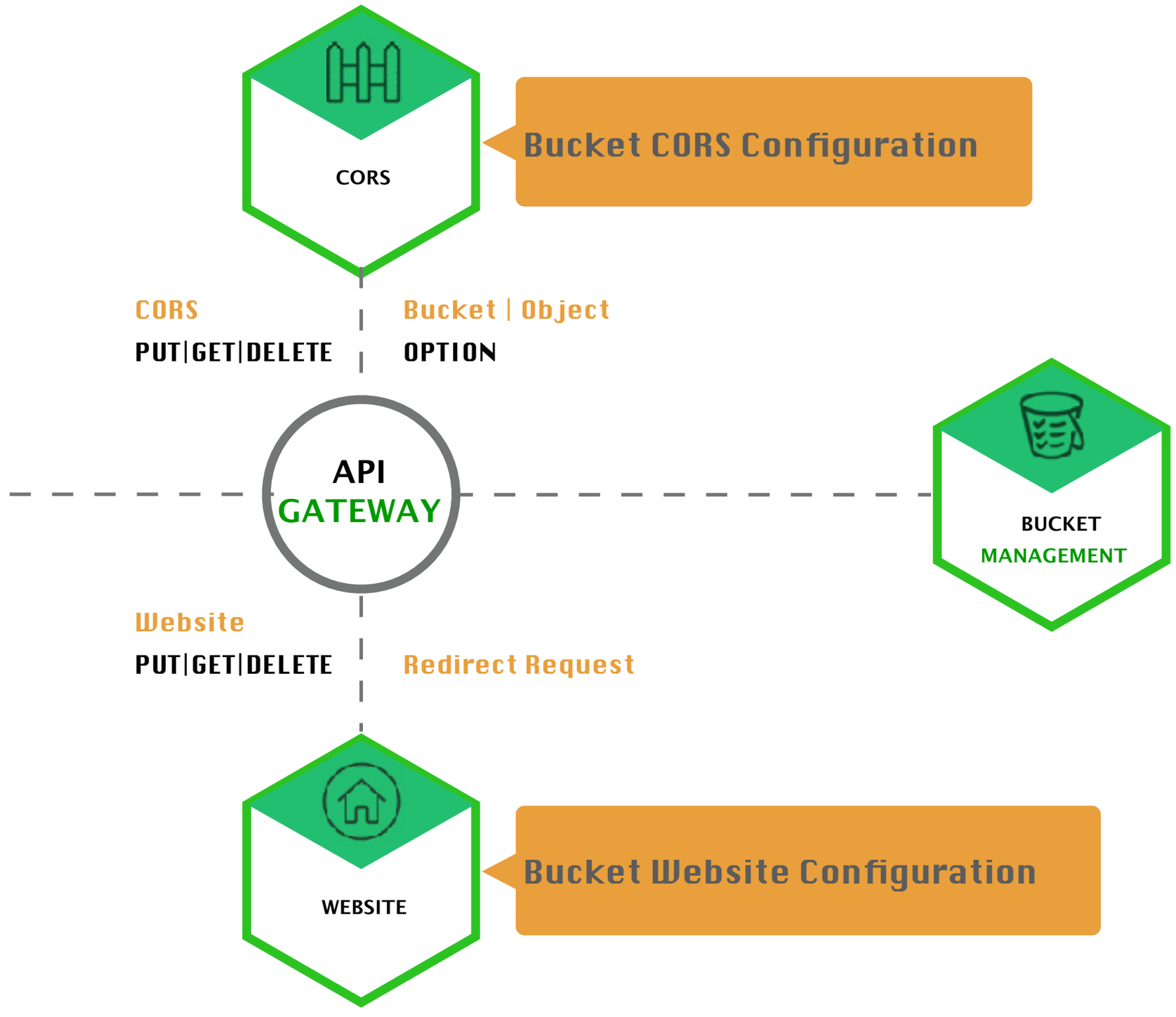

request
response
REST API
API GATEWAY
PERSISTENCE
IPC
object ploglist
data for GC
Bucket
PUT|GET|HEAD|DELETE
Tags
PUT|GET|DELETE
Location
GET
name
location
create date
tags
BUCKET
MANAGEMENT
BucketCreated
BucketDeleted
bucket name
versioning
OBJ
OBJECT
MANAGEMENT
ObjectCreated
ObjectDeleted
Bucket Versioning
PUT|GET
Object
PUT|GET|HEAD|DELETE|LIST
Tags
PUT|GET|DELETE
Object Copy
PUT
object name
version id
tags

AUTHENTICATION

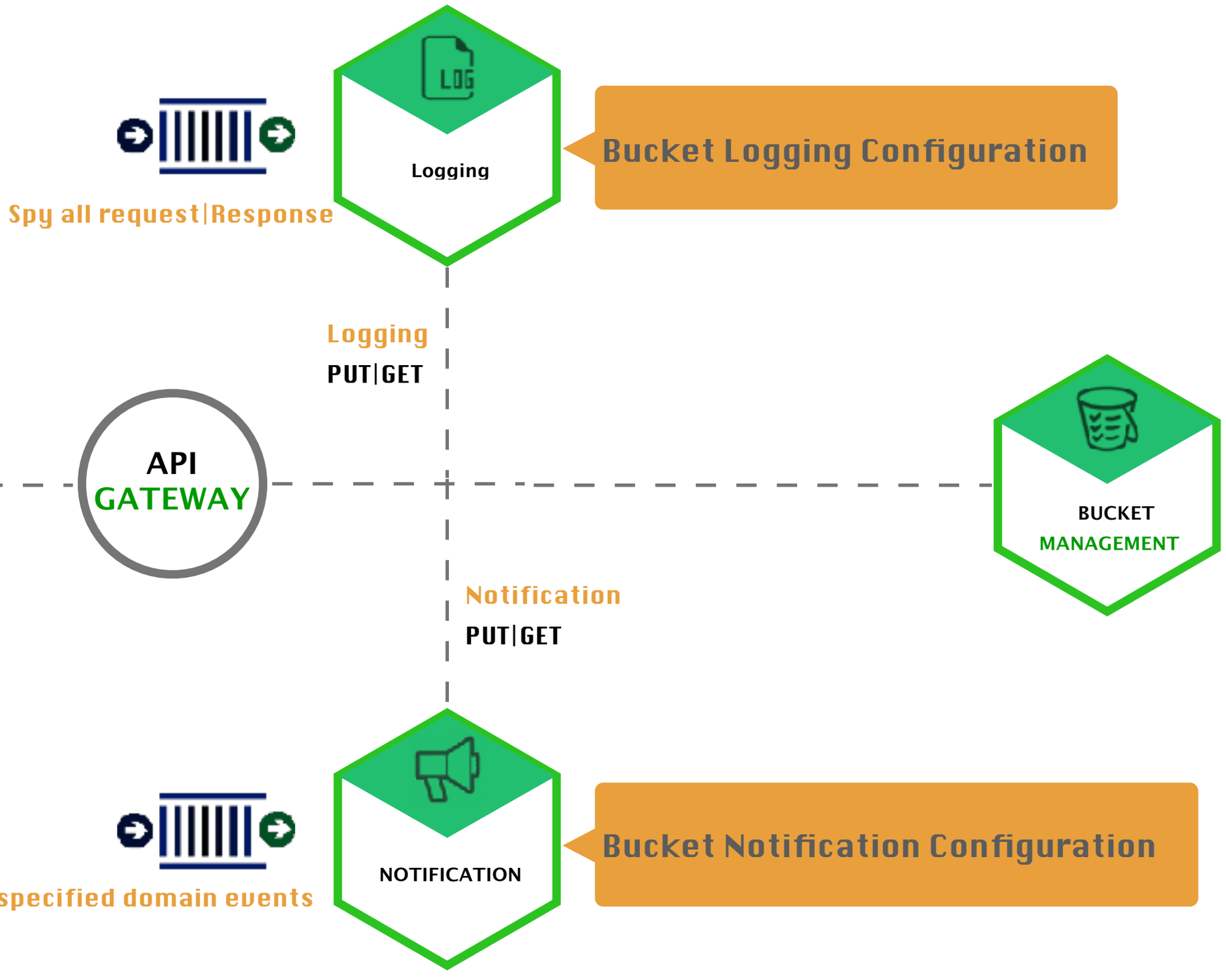


➤ *Sometimes distributed authentication is better*

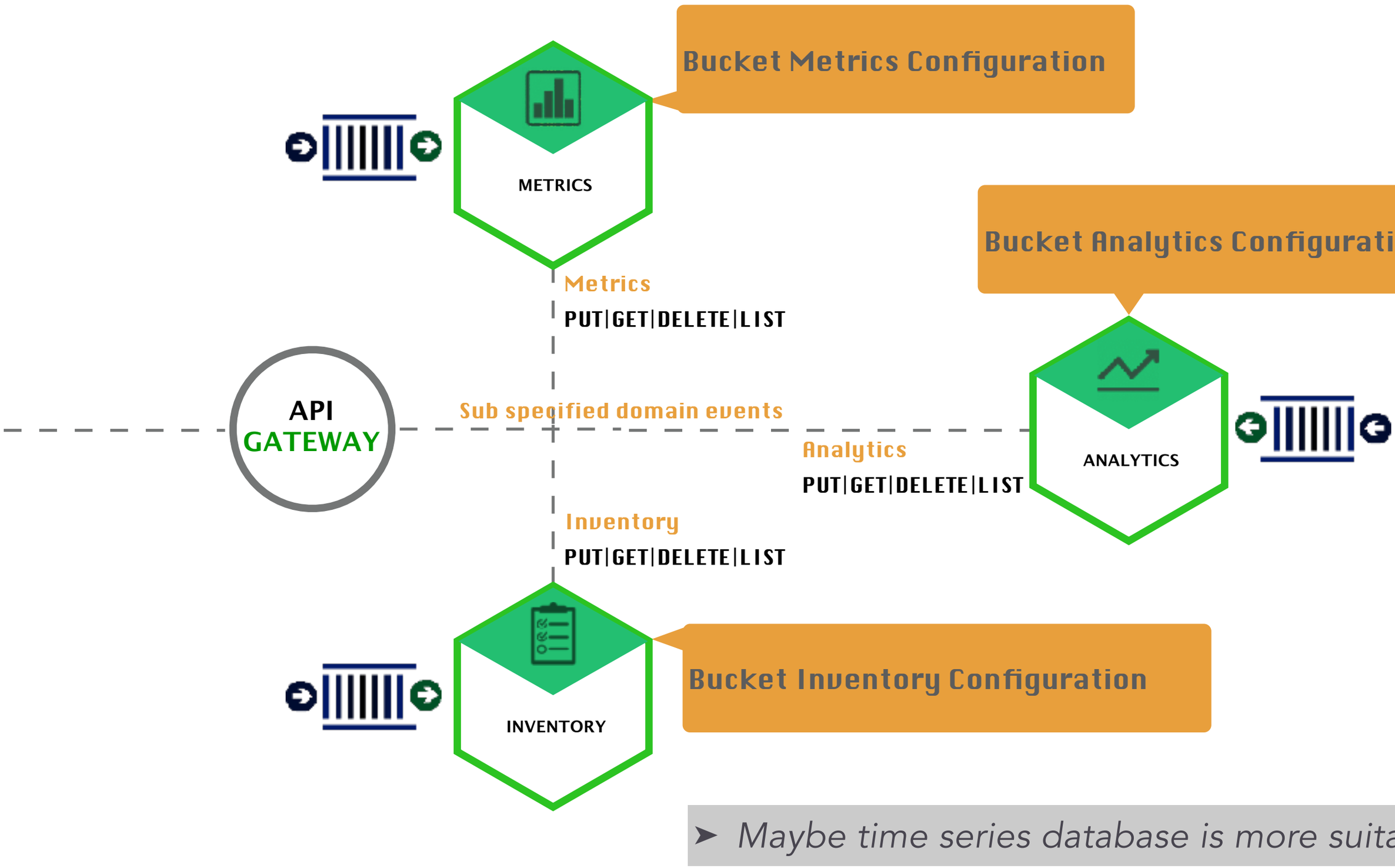
WEBSITE AND CORS



LOGGING AND NOTIFICATION

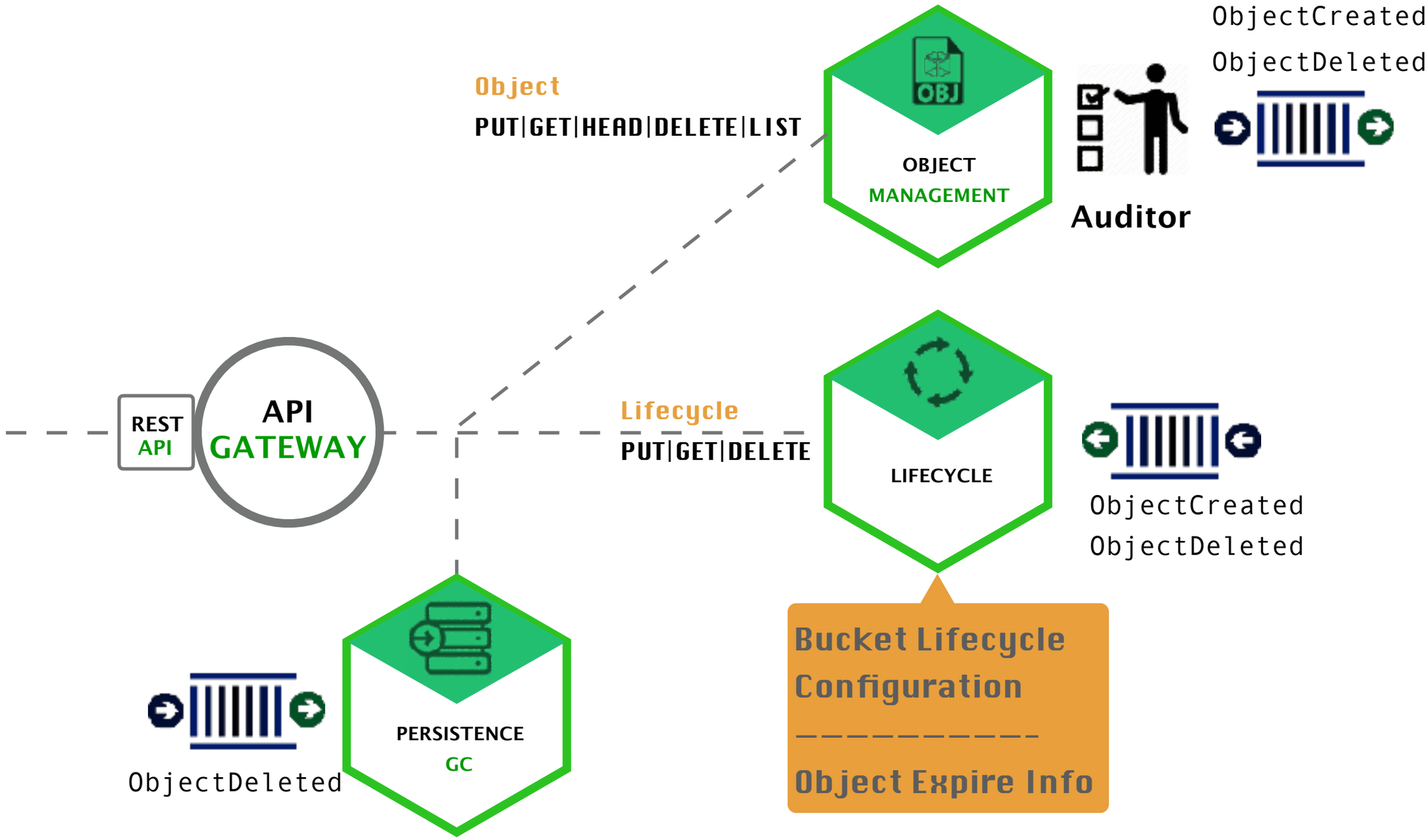


STATISTICS AND ANALYTICS



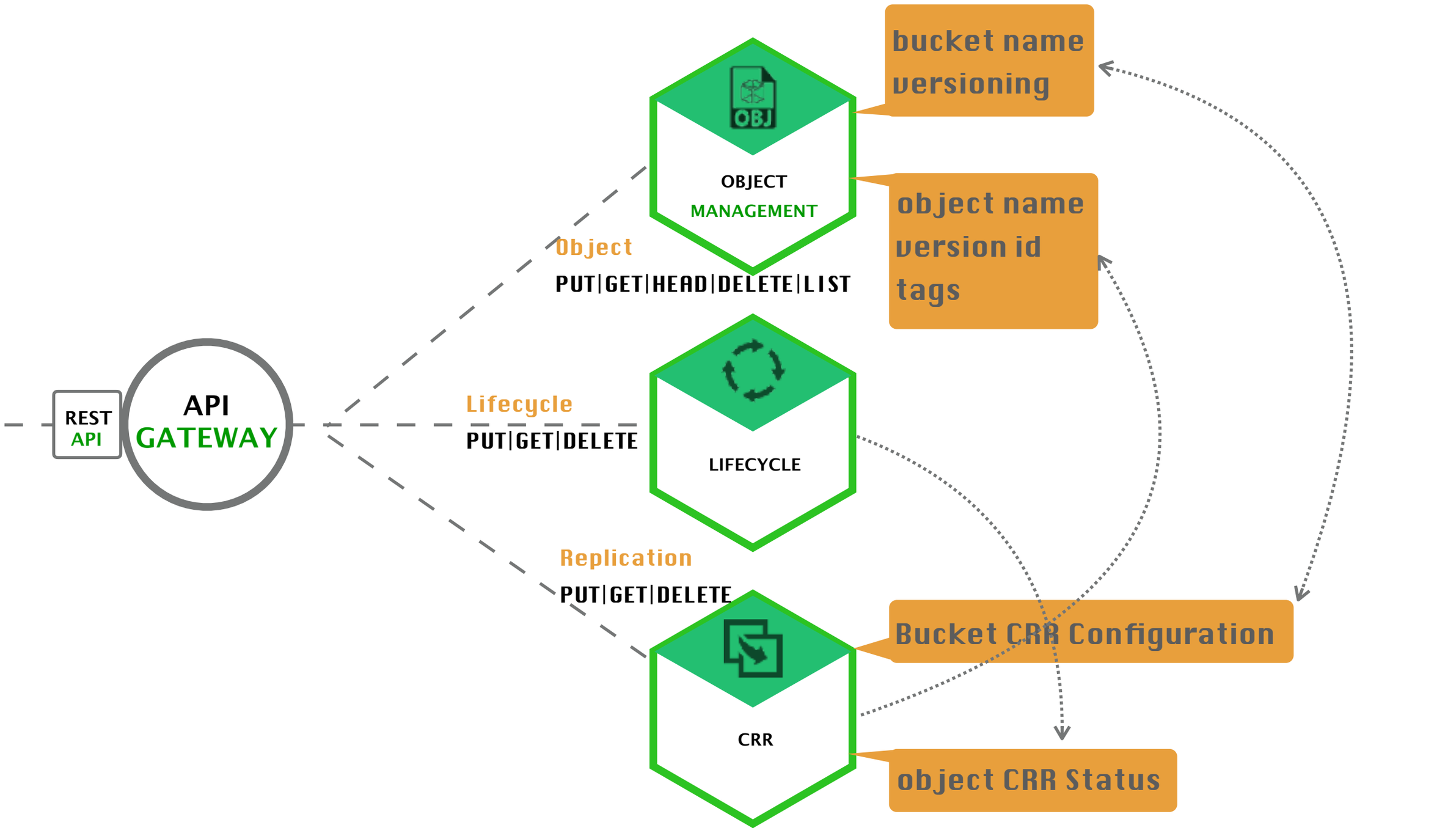
➤ *Maybe time series database is more suitable*

LIFECYCLE



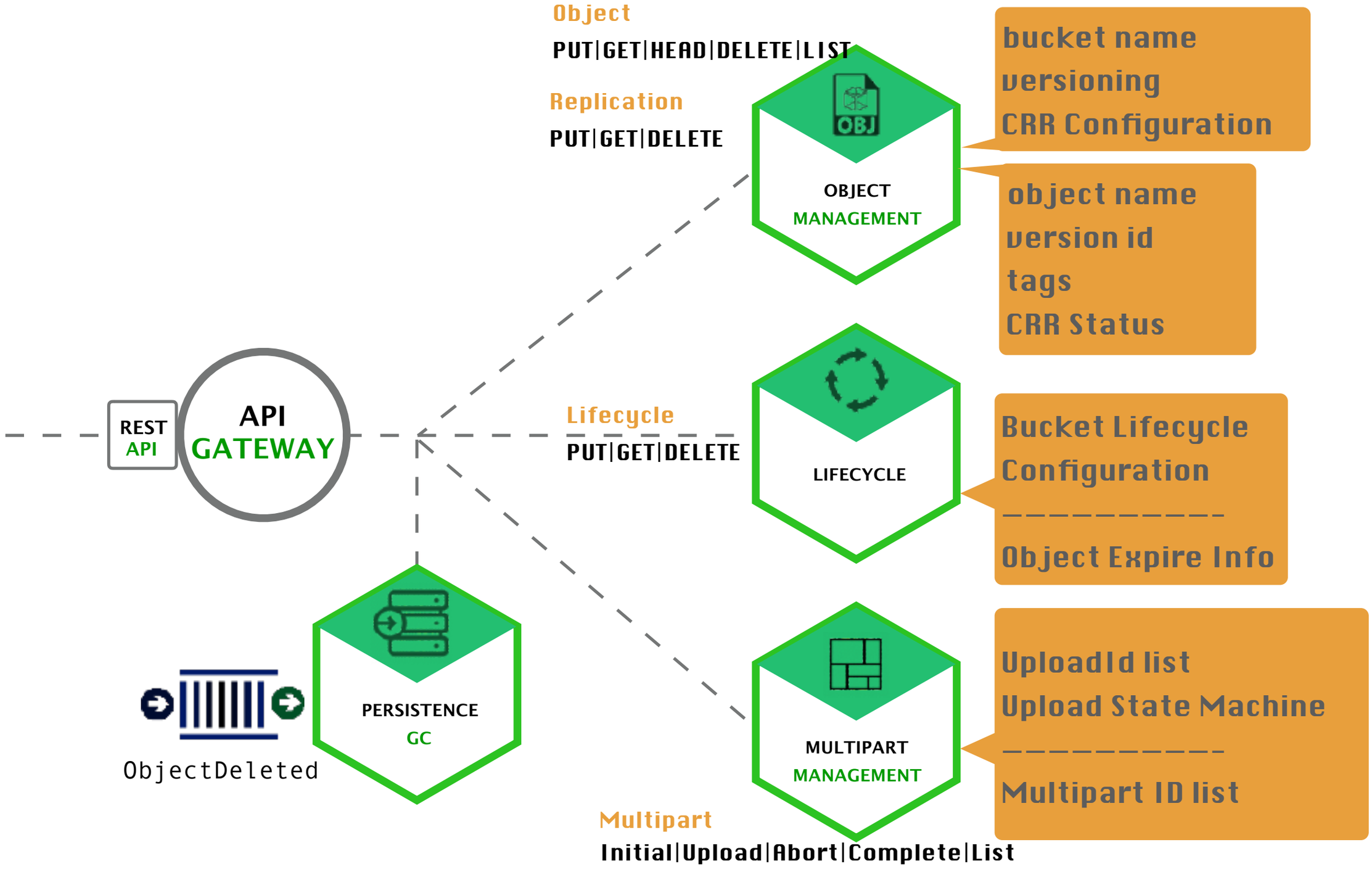
➤ *Composite design: service level reuse*

CRR



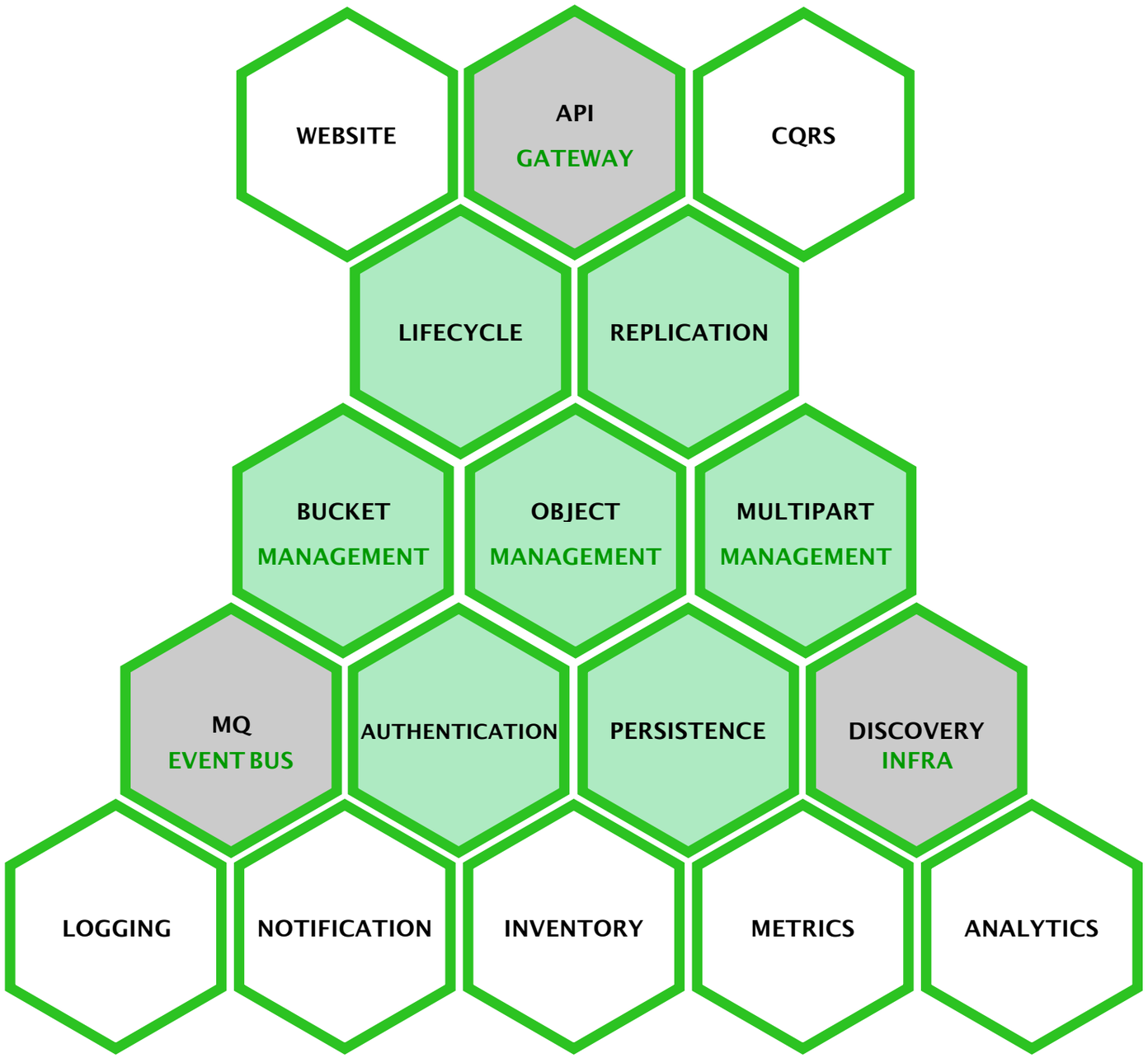
- *CRR是否适合拆分为独立的服务?*

CRITICAL PATH



➤ 事实上，数据面的每个*feature*都需要更谨慎细致的分析

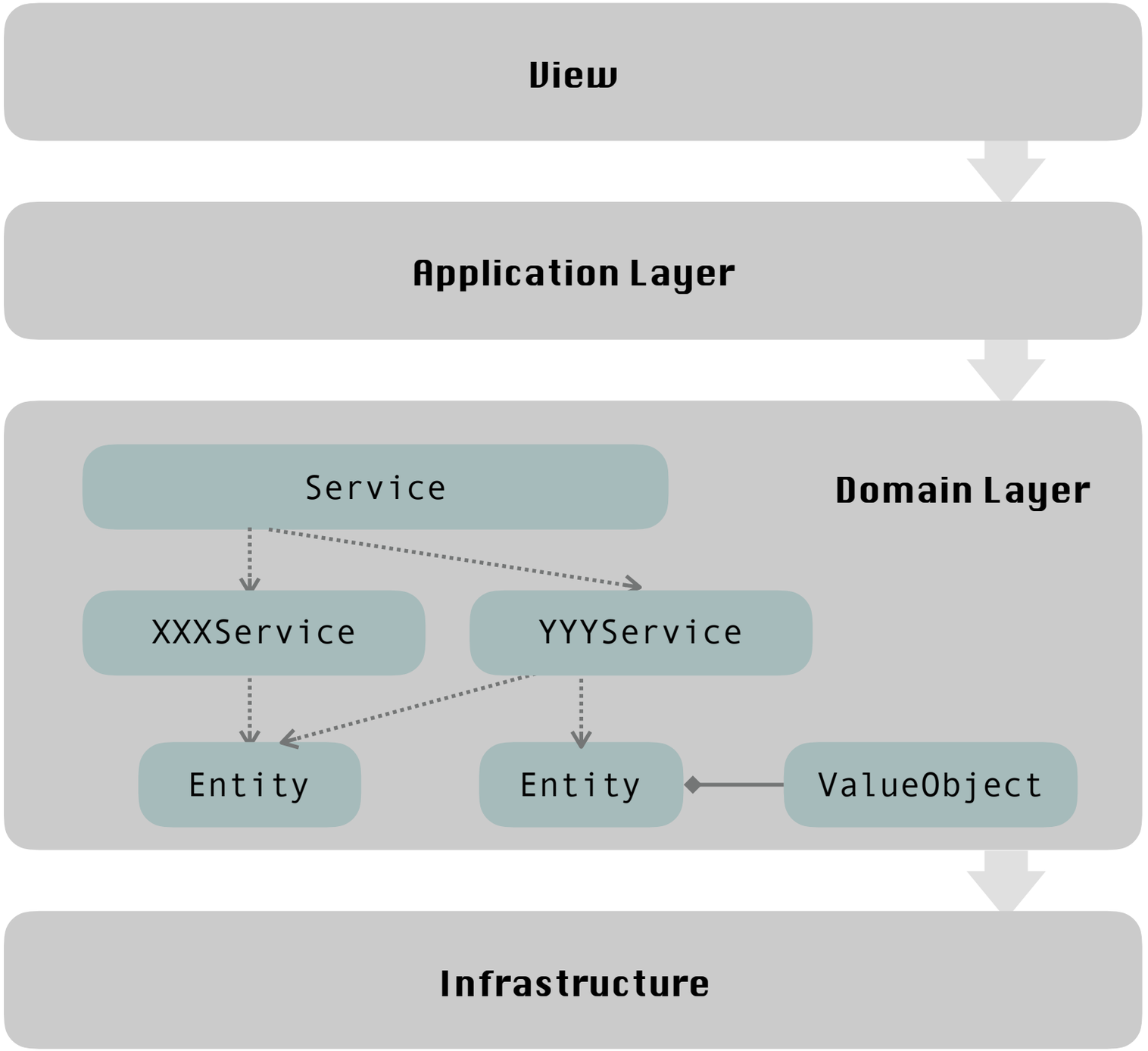
CONCLUSION



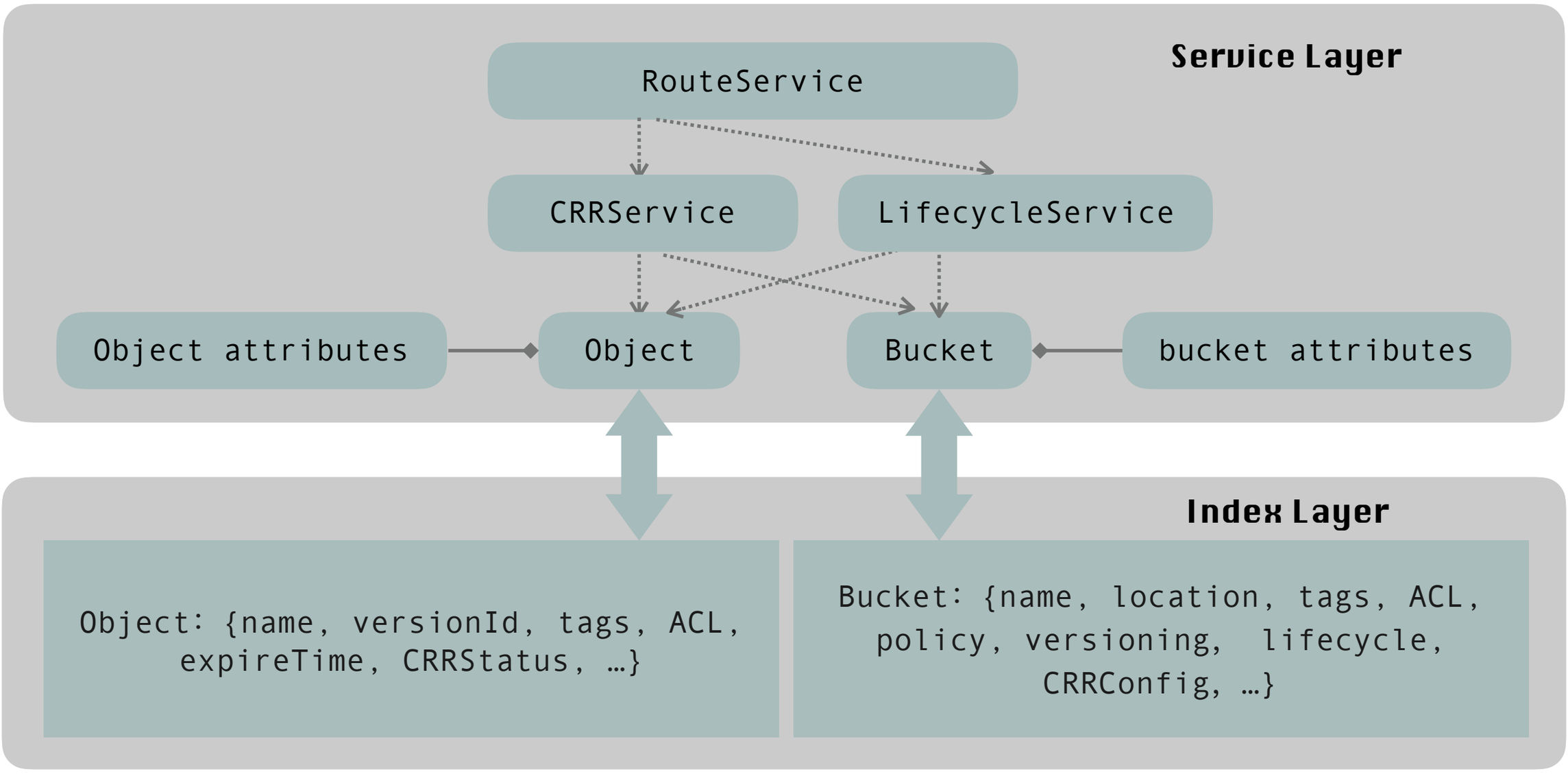
- 从前端和后端较容易的开始拆分
- 性能关键的和强一致性的不要拆分
- 耦合较大，不确定的暂时不要拆分
- 依赖服务稳定的接口
- 通过演进式拆分逐步完善基础设施
- 新写的功能逐一分析

服务内解耦设计

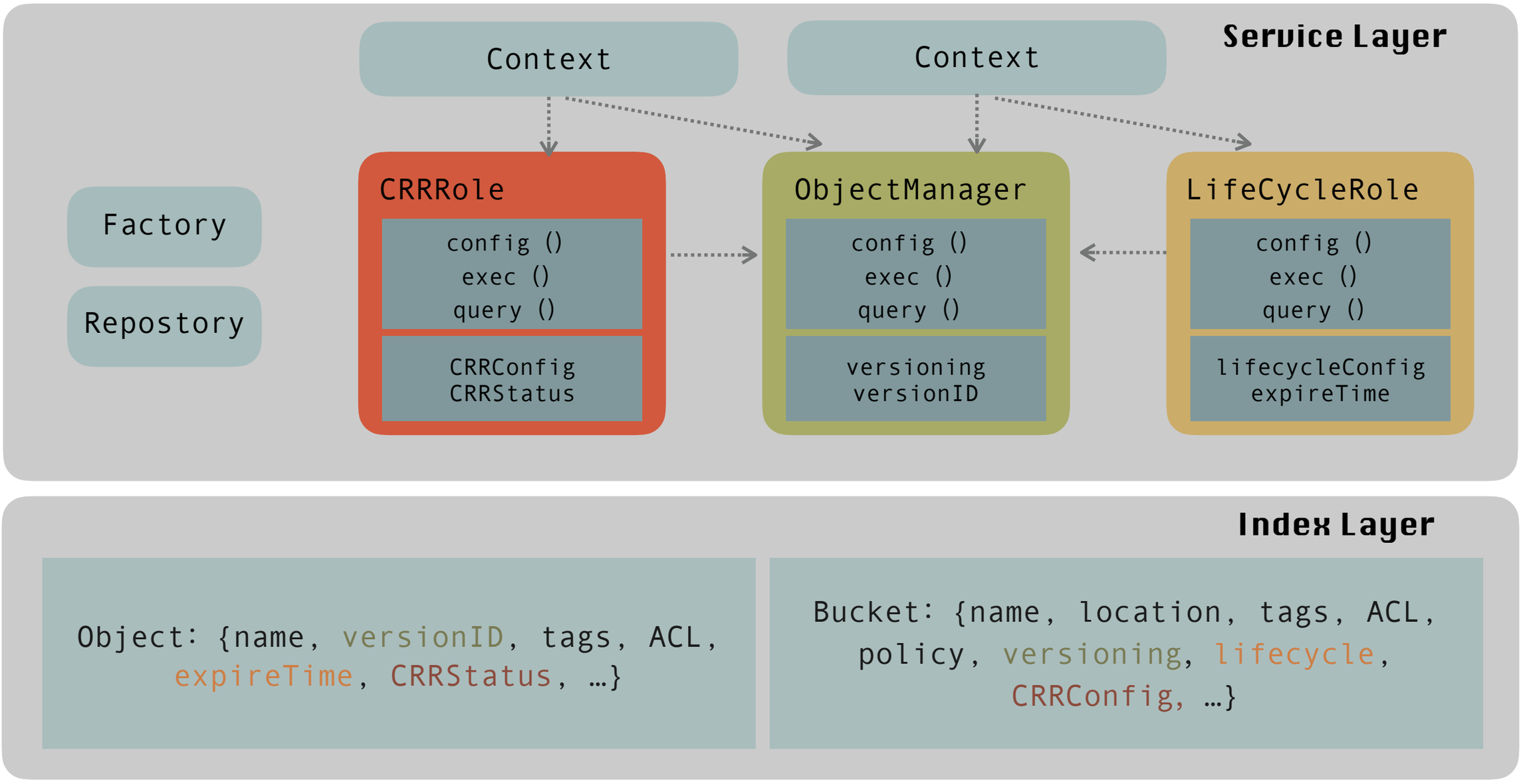
领域驱动设计


View
Application Layer
Service
Domain Layer
XXXService
YYYService
Entity
Entity
ValueObject
Infrastructure

避免贫血模型

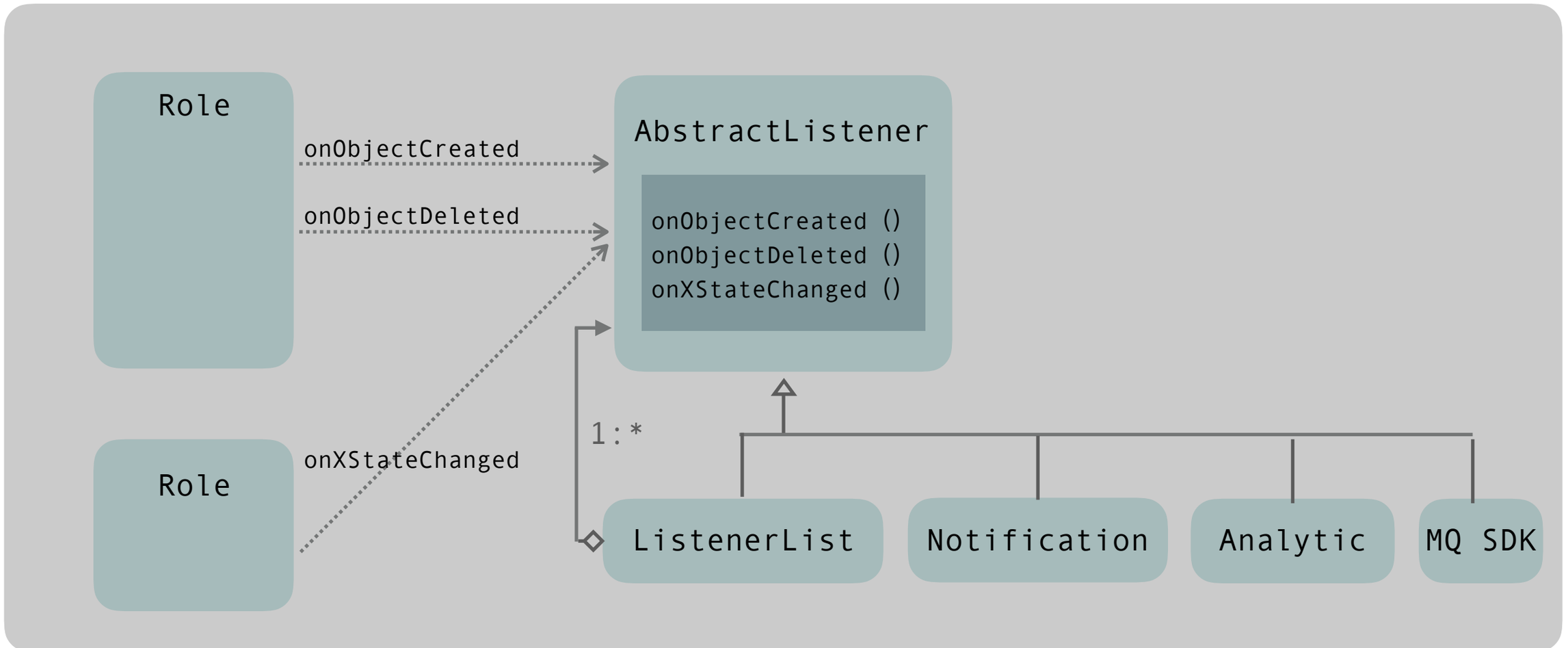

Service Layer
RouteService
CRRService
LifecycleService
Object attributes
Object
Bucket
bucket attributes
Index Layer
Object: {name, versionId, tags, ACL, expireTime, CRRStatus, …}
Bucket: {name, location, tags, ACL, policy, versioning, lifecycle, CRRConfig, …}

模块化设计


Service Layer
Context
Context
Factory
Repostory
CRRRole
config ()
exec ()
query ()
CRRConfig
CRRStatus
ObjectManager
config ()
exec ()
query ()
versioning
versionID
LifeCycleRole
config ()
exec ()
query ()
lifecycleConfig
expireTime
Index Layer
Object: {name, versionID, tags, ACL, expireTime, CRRStatus, …}
Bucket: {name, location, tags, ACL, policy, versioning, lifecycle, CRRConfig, …}

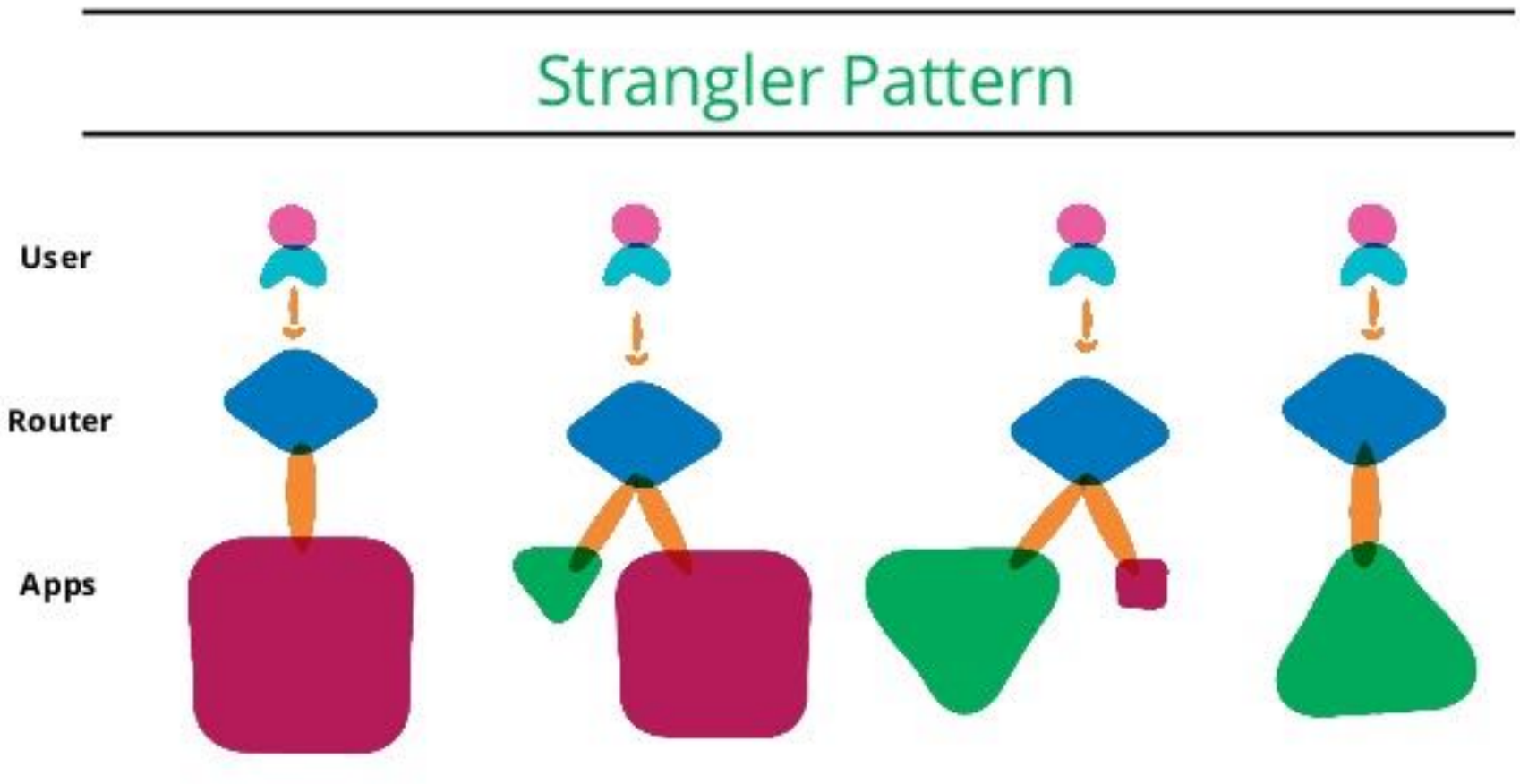
示意图

组合式设计


Role
onObjectCreated
onObjectDeleted
AbstractListener
onObjectCreated ()
onObjectDeleted ()
onXStateChanged ()
Role
onXStateChanged
1:*
ListenerList
Notification
Analytic
MQ SDK

演进式设计及建议

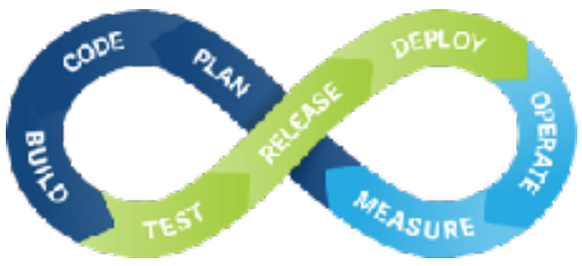
EVOLUTION WAY



➤ *Visualize the whole process, statistics can tell you more!*

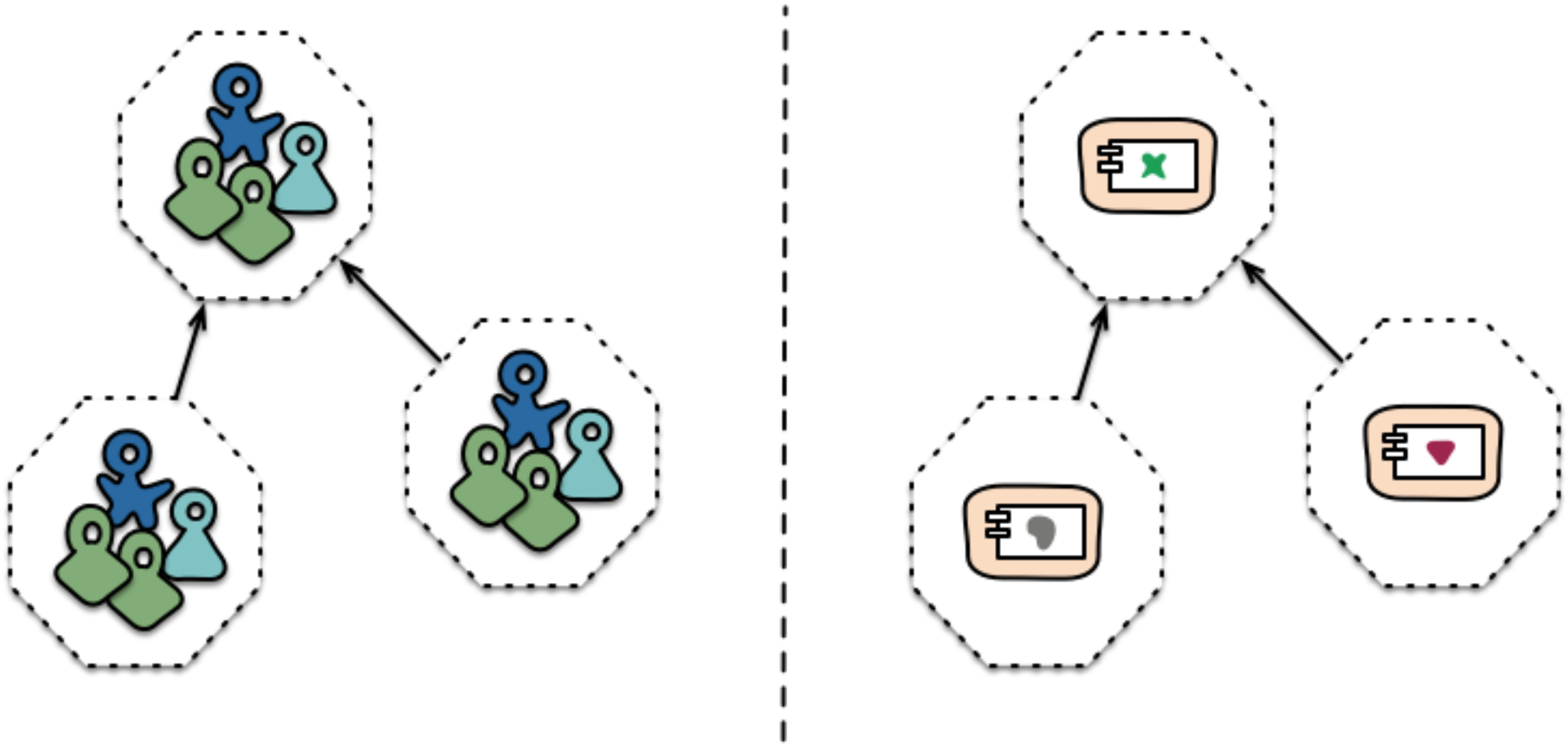
IMPROVEMENTS

- Skills of design, coding and test
  - Domain Driven Design
  - Orthogonal design
  - TDD, Refactoring…



- Process on continuous delivery pipeline
  - Customer driven contract test
  - integrating speed
  - DevOps

ADAPTIVE ORGANIZATION STRUCTURE



Cross-functional teams...

... organised around capabilities
Because Conway's Law

Questions?

ThoughtWorks

THANKS

e.wangbo@gmail.com