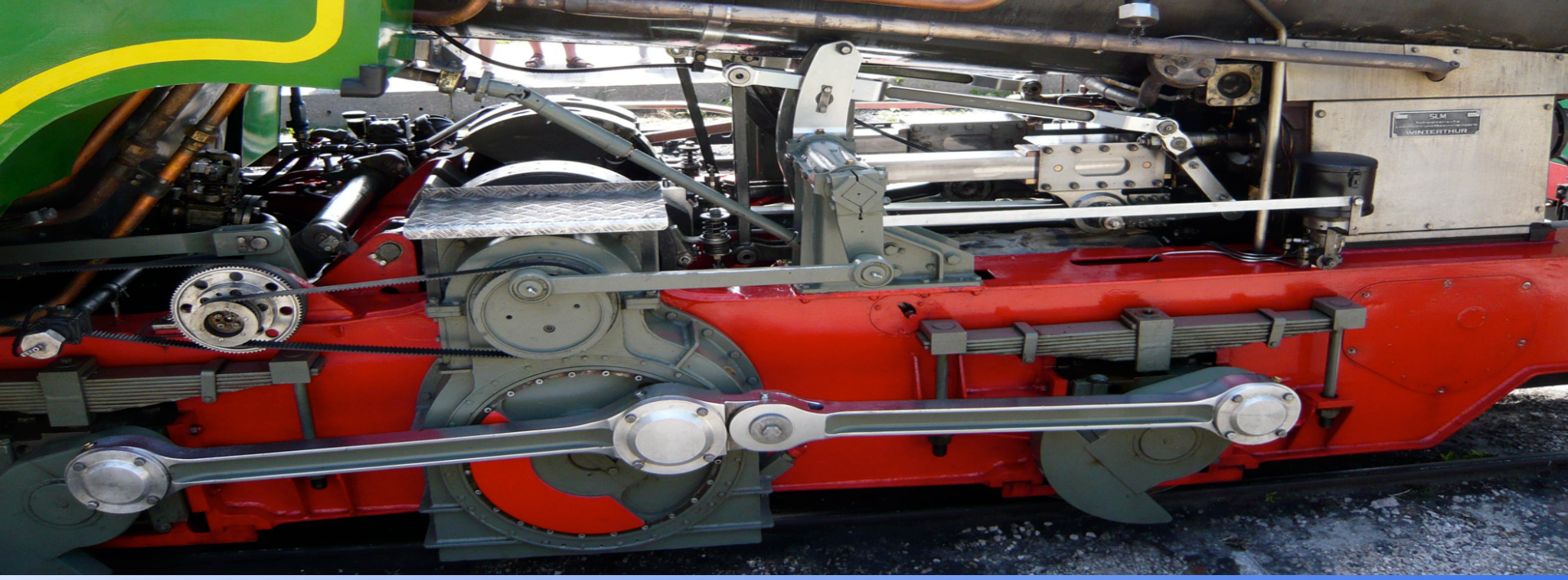


RestFS Internals

Fabrizio Manfredi Furuholmen
Federico Mosca

Beolink.org

Agenda

RestFS

- Introduction
  - Goals
  - Principals
- RestFS
  - Architecture
  - Internals
  - Sub project
- Conclusion
  - Developments

Zetabyte

1000^7 bytes

10^{21} bytes

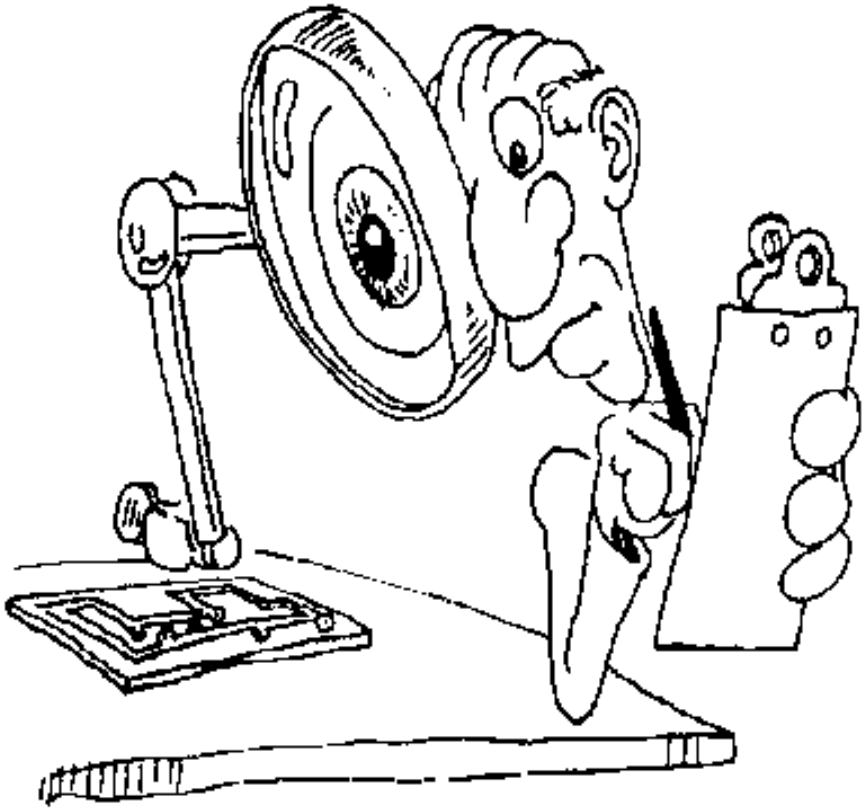
1,000,000,000,000,000,000,000 bytes

All of the data on Earth today 150GB of data per person

2% of the data on Earth in 2020

Create a free available Cloud Storage Software

Create a framework for testing a new technologies and paradigm



“Moving Computation is Cheaper than Moving Data”

“There is always a failure waiting around the corner”

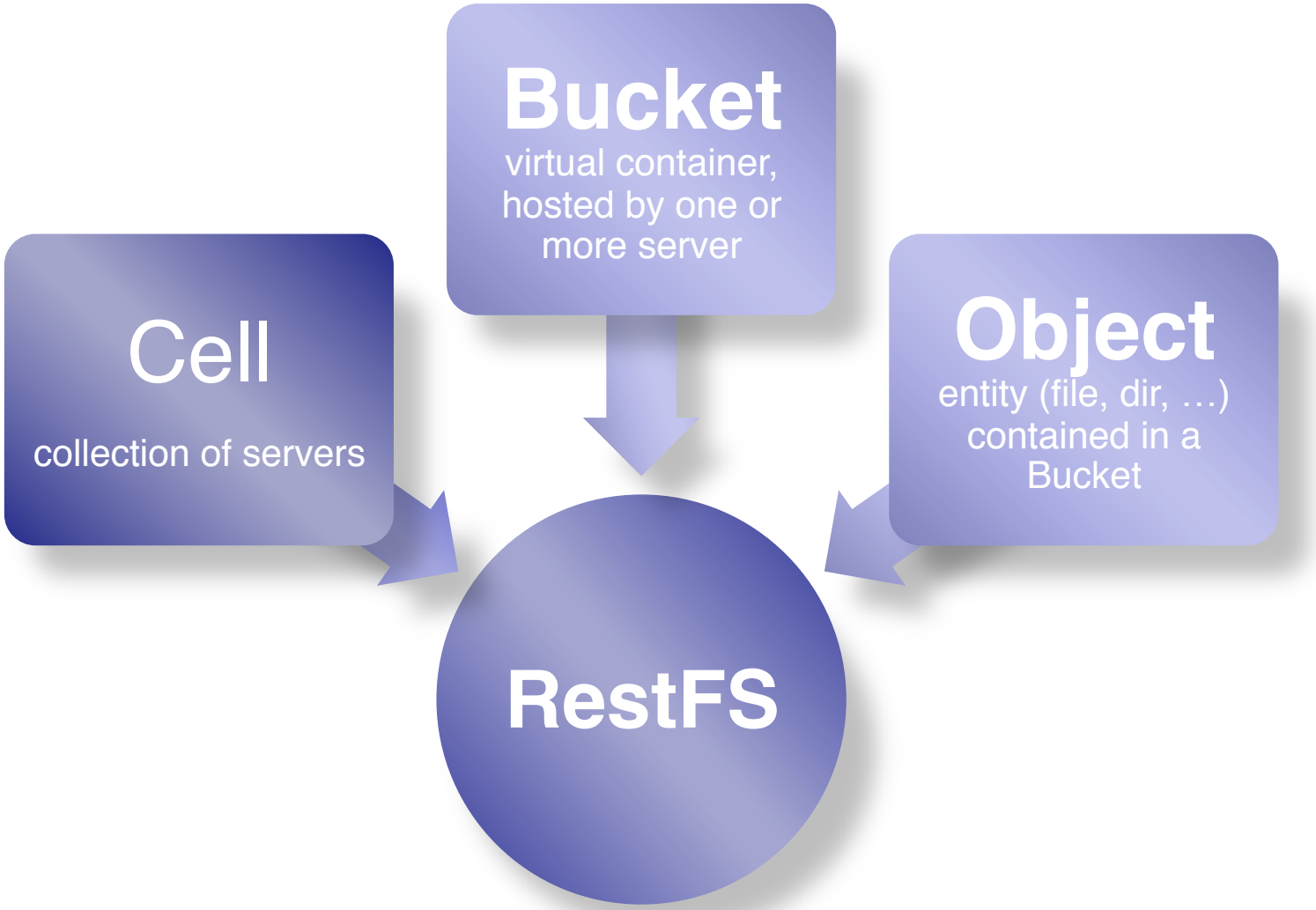
*Werner Vogel

“Decompose into small loosely coupled, stateless building blocks”

*’ Leaving a Legacy System Revisited’ Chad Fowler


RestFS

RestFS Key Words



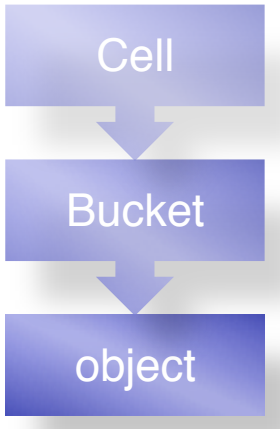
S3:

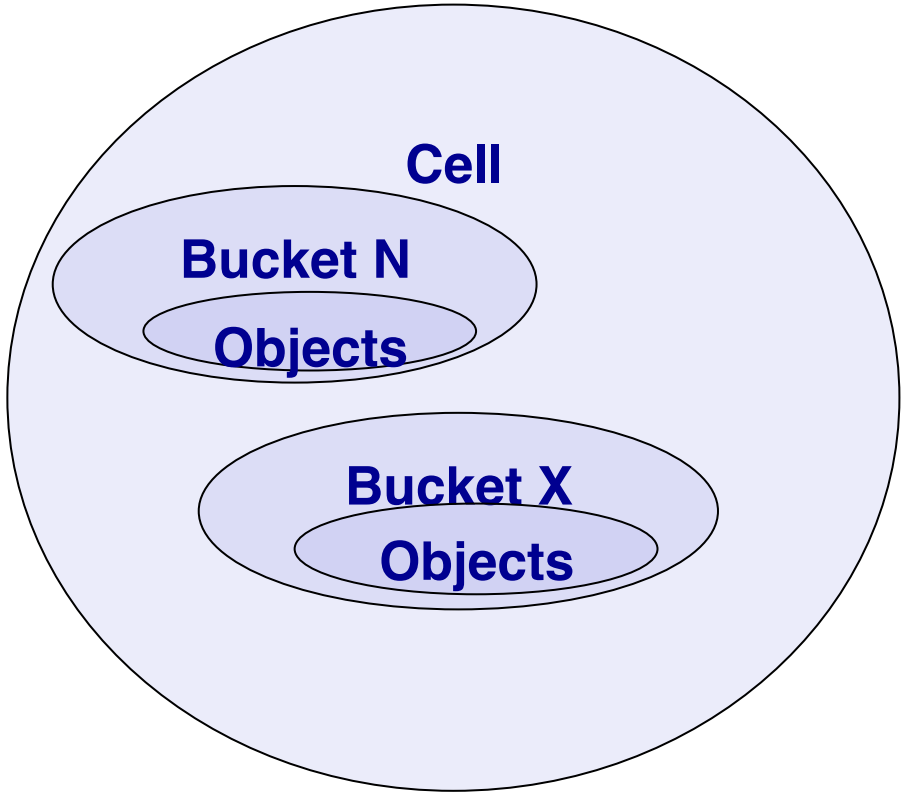
bucket_name.mydomain.com/object_name

RestFS:

bucket_name.mydomain.com

Rpc :
bucket_name
object_name





Five main areas

Beolink.org

Objects

- Separation btw data and metadata
- Each element is marked with a revision
- Each element is marked with an hash.

Cache

- Client side
- Callback/ Notify
- Persistent

Transmission

- Parallel operation
- Http like protocol
- Compression
- Transfer by difference

Distribution

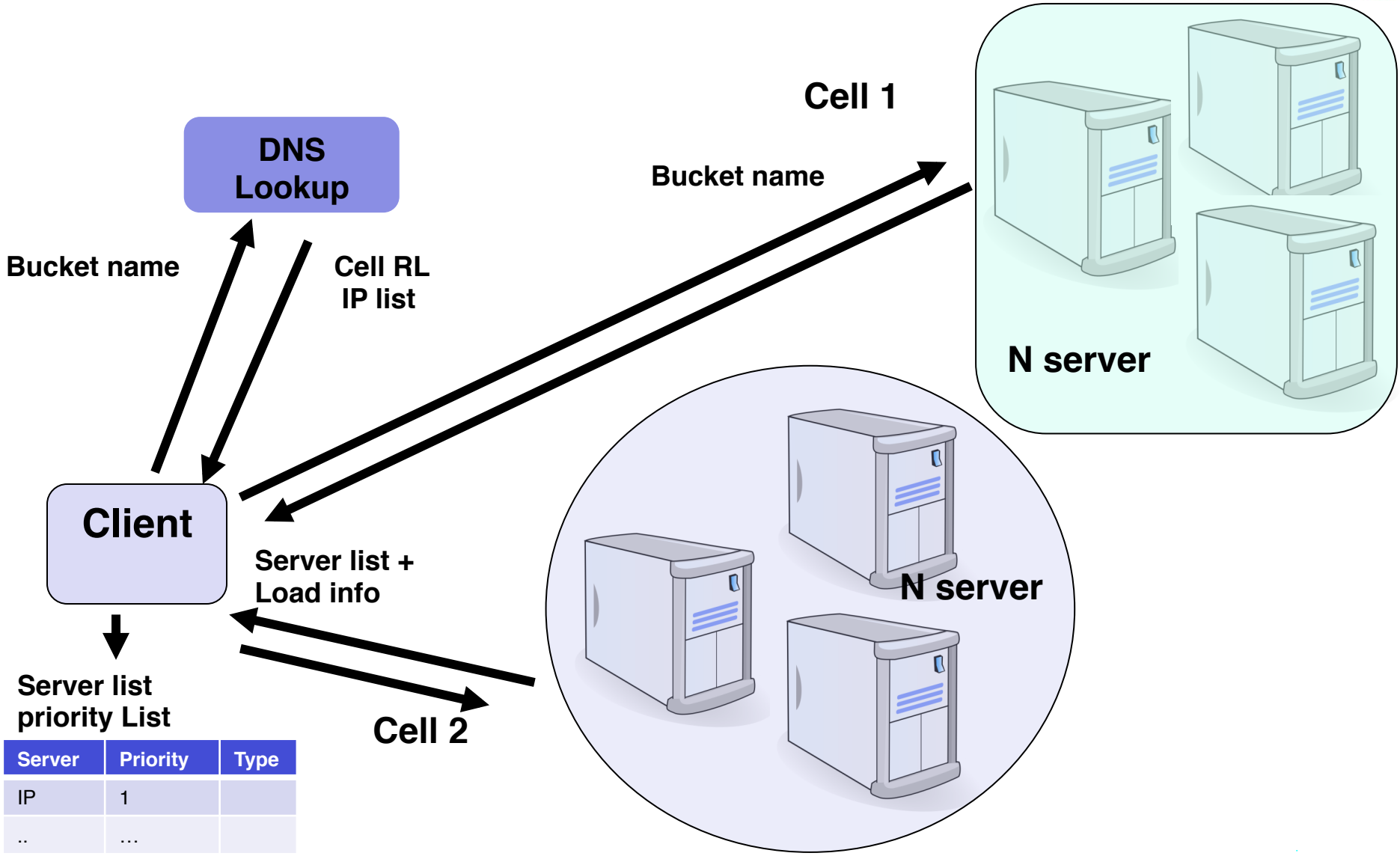
- Resource discovery by DNS
- Data spread on multi node cluster
- Decentralize
- Independents cluster
- Data Replication

Security

- Secure connection
- Encryption client side,
- Extend ACL
- Delegation/ Federation
- Admin Delegation

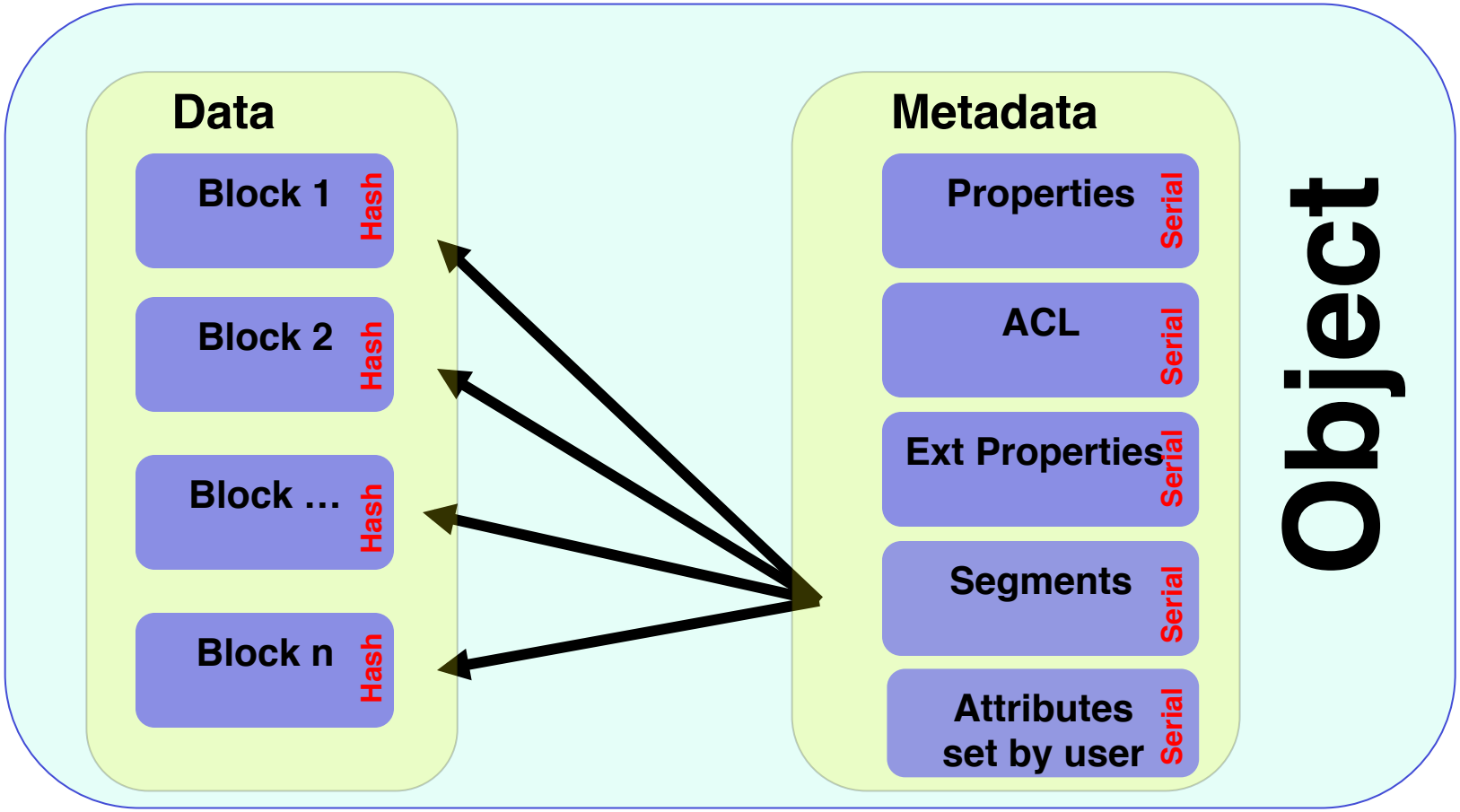
Bucket Discovery

Beolink.org

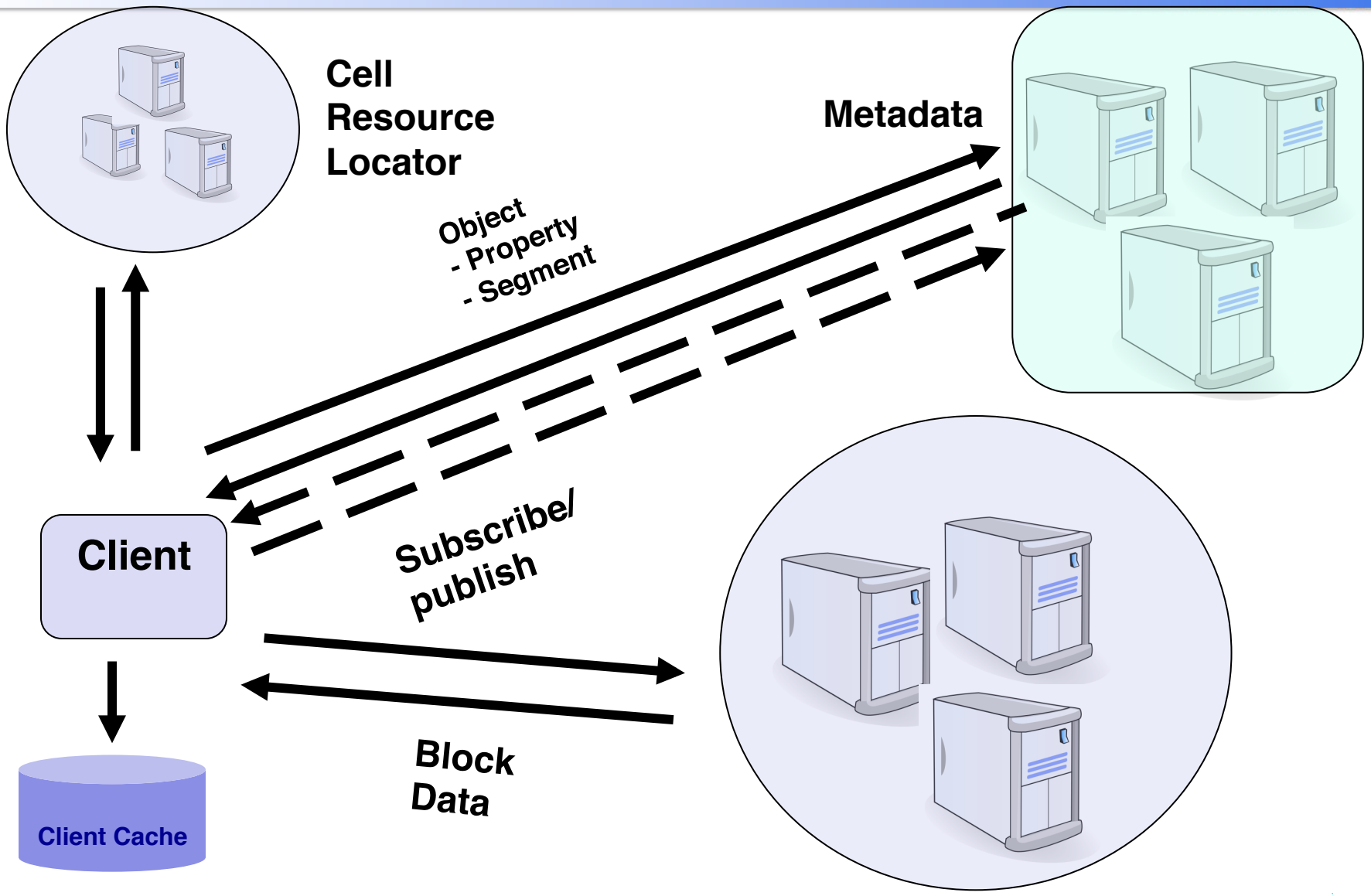


Server	Priority	Type
IP	1	
..	…	

Object

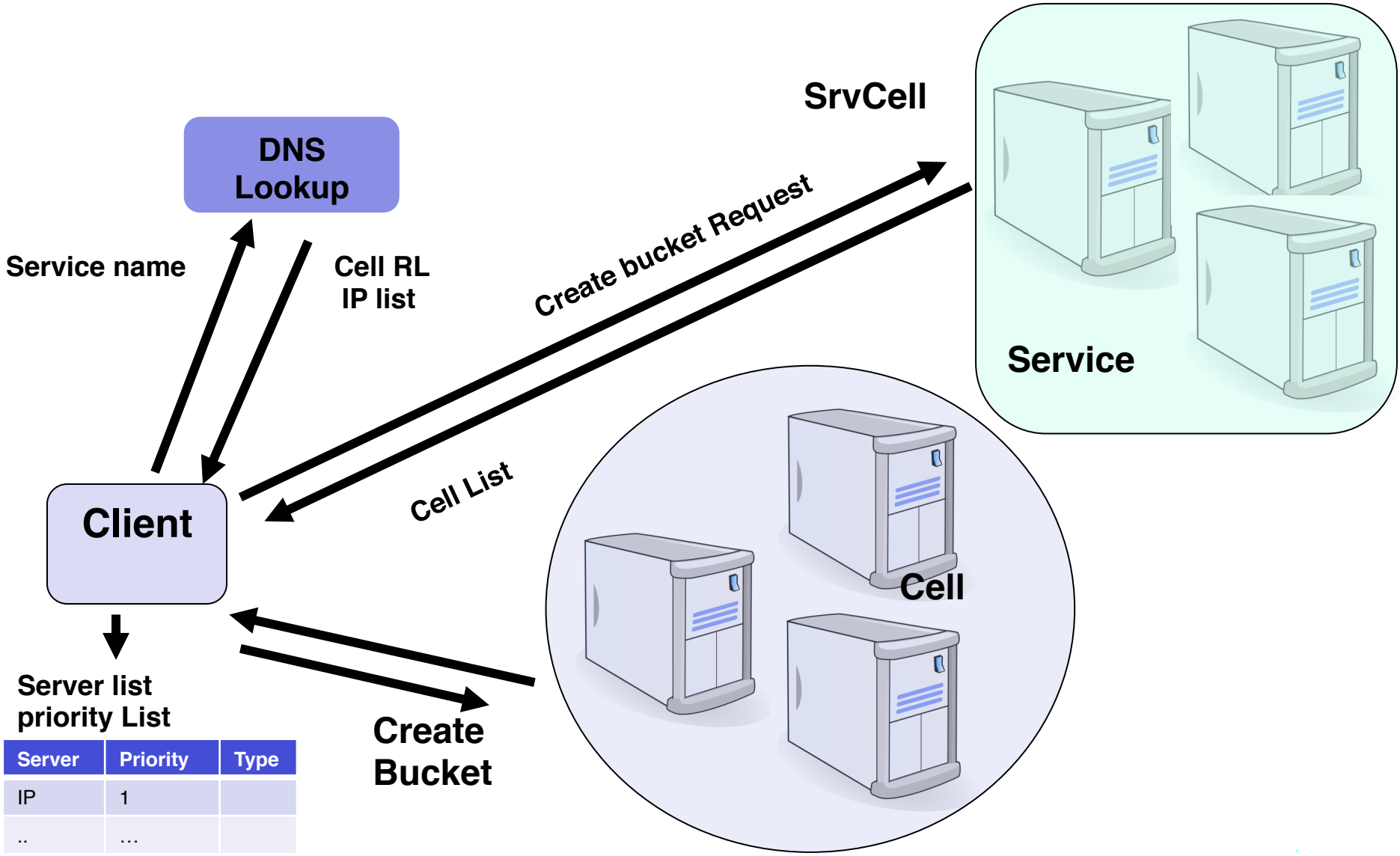

Data
Block 1
Hash
Block 2
Hash
Block ...
Hash
Block n
Hash
Metadata
Properties
Serial
ACL
Serial
Ext Properties
Serial
Segments
Serial
Attributes set by user
Serial
Object

Cell Interaction



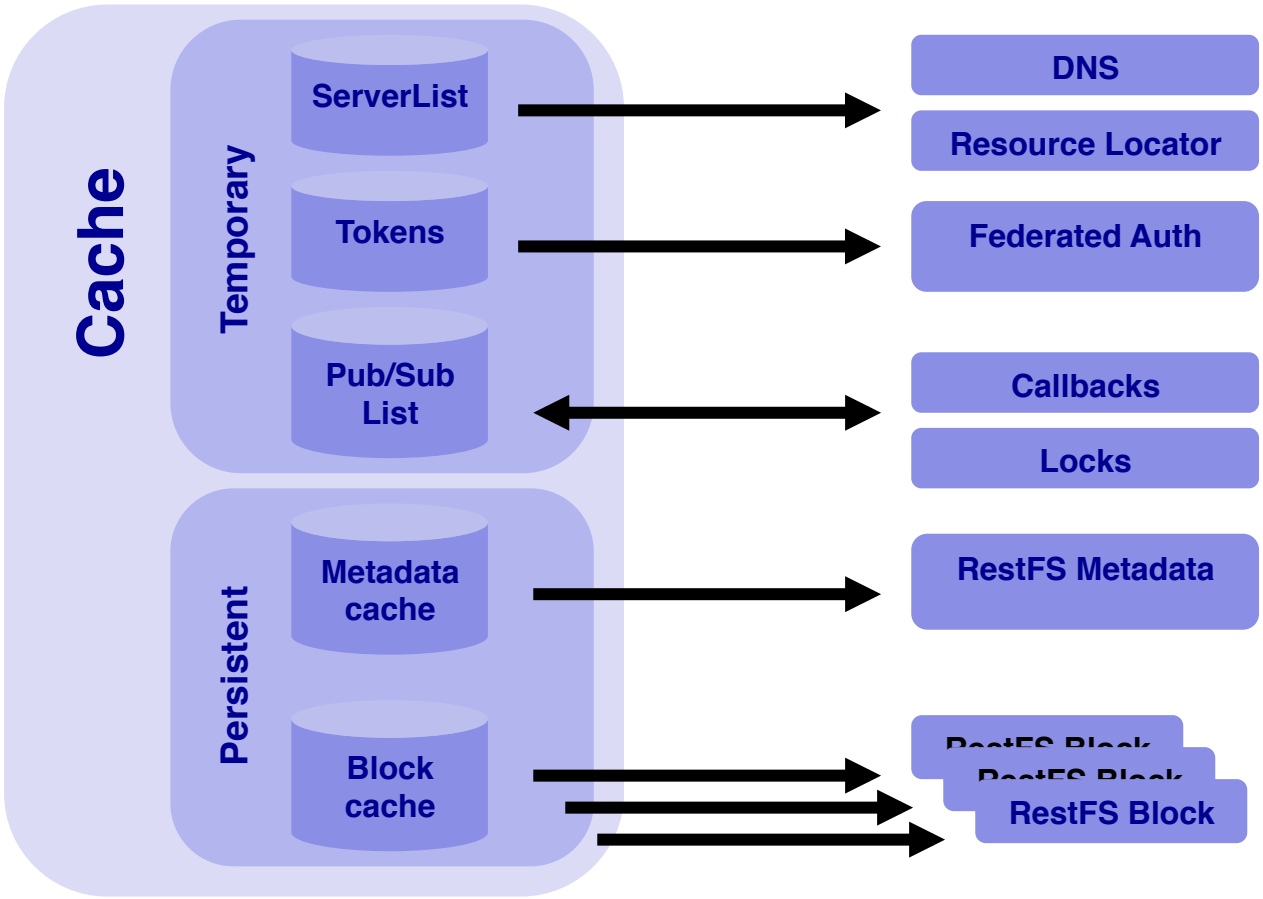
Central Service for Domain

Beolink.org



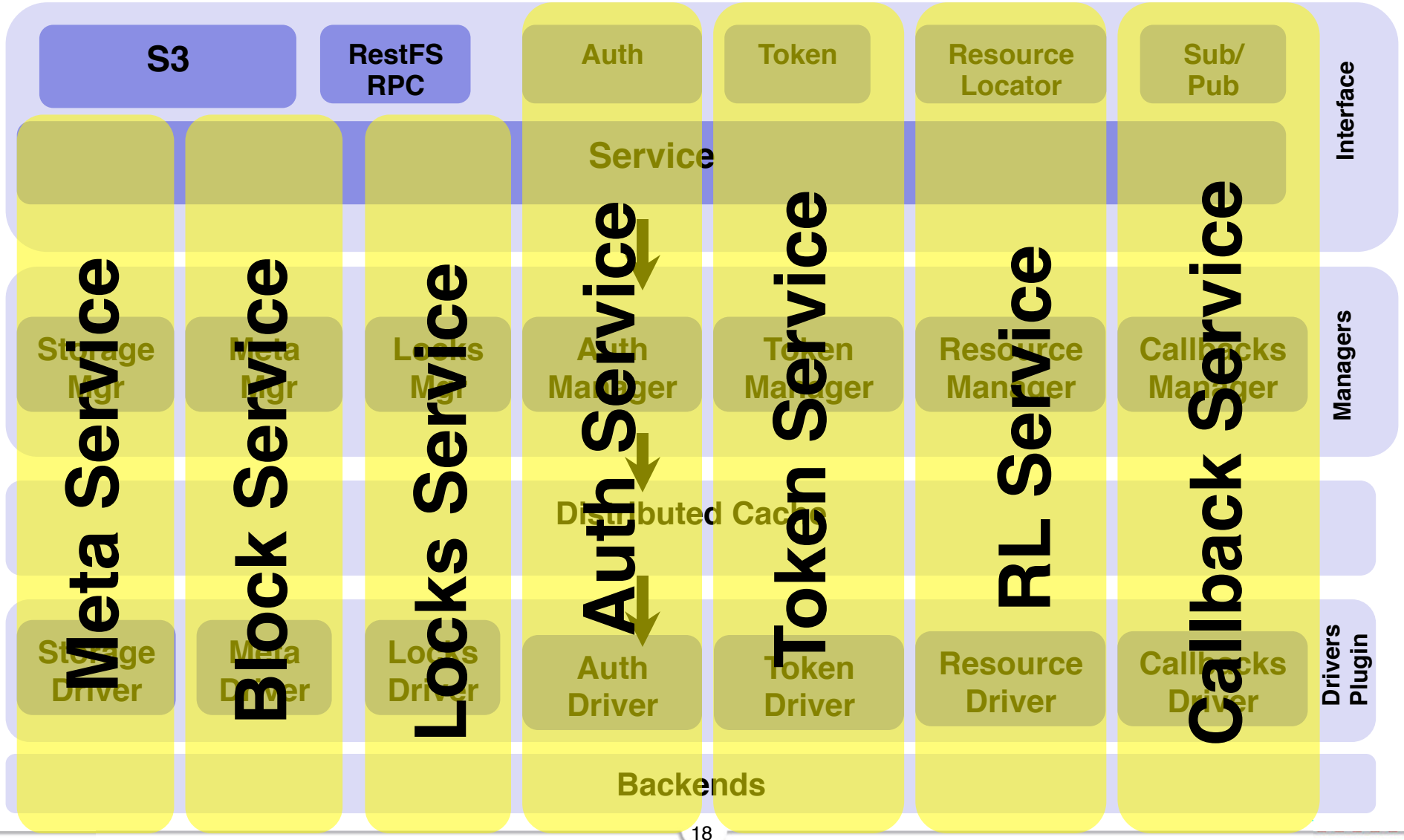
Server	Priority	Type
IP	1	
..	…	

Cache client side

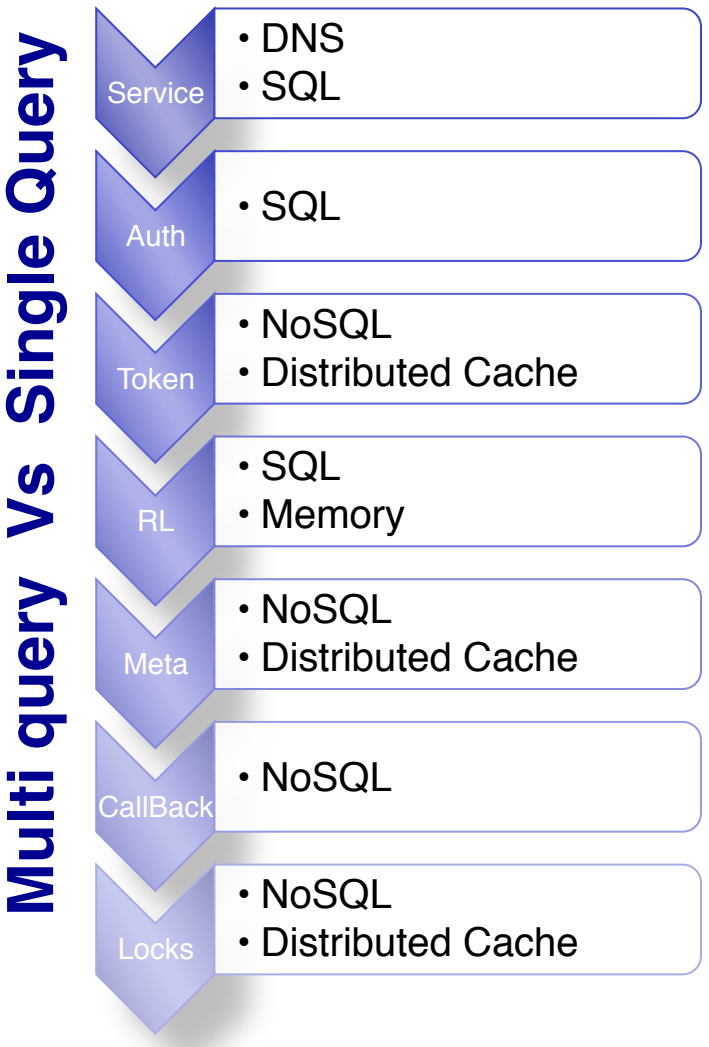


Server Architecture

Beolink.org



- Dedicated storage infrastructure per Service
- Distributed Memory cache
- One or more DB per Driver



Bucket

❑ Cell Ips

The bucket is stored in the DNS for lookup, the ip address returned by DNS are the Cell RL address

❑ Property

The property element is a collection of object information, with this element you can retrieve the default value for the bucket (logging level, security level, ect).

- Property
- Property Ext
- Property ACL
- Property Stats

Bucket Name
zebra

Property
segment_size= 512
block_size = 16k
max_read'=1000
storage_class=STANDARD
compression= none
…

Object Serialized

Backends agnostic on information stored

Bucket Type

- **Filesystm**
  The bucket is used as a filesystem
- **Logging**
  Logging operation done on the specific Bucket
- **Replica RO**
  Bucket shadow replication
- **...**
  Custom definition

Objects

Object

- Object Property
- Object Property Ext
- Object Stats
- Object ACL
- Segments

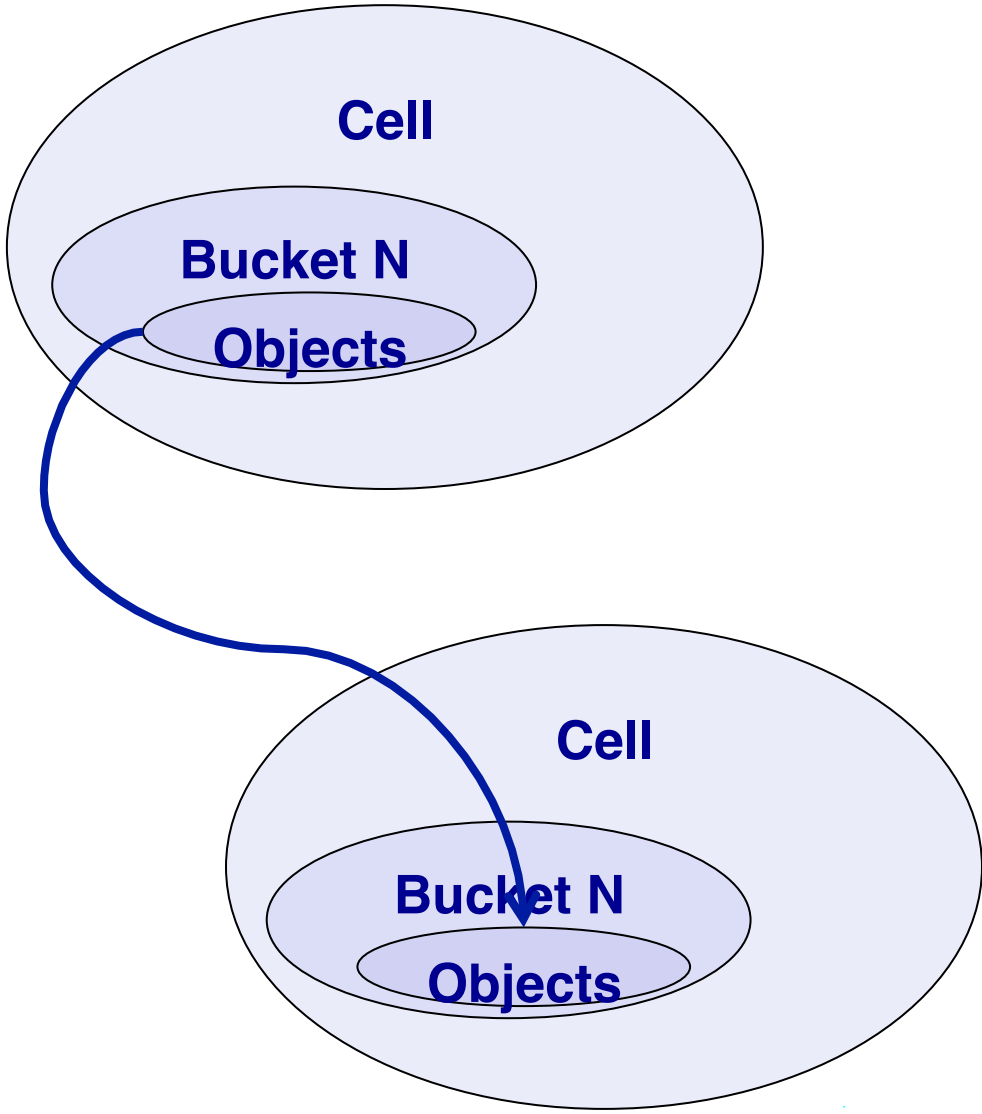
Object
zebra.c1d2197420bd41ef24fc665f228e2c76e98da247

Segment-id
1:16db0420c9cc29a9d89ff89cd191bd2045e47378
2:9bcf720b1d5aa9b78eb1bcdbf3d14c353517986c
3:158aa47df63f79fd5bc227d32d52a97e1451828c
4:1ee794c0785c7991f986afc199a6eee1fa4
5:c3c662928ac93e206e025a1b08b14ad02e77b29d
…
vers:1335519328.091779

Property
segment_size= 512
block_size = 16k
content_type =
md5=ab86d732d11beb65ed0183d6a87b9b0
max_read'=1000
storage_class=STANDARD
compression= none
…

Object Type

- **Data**
  Contains files
- **Folder**
  Special object that contain others objects
- **Mount point**
  Contains the name of the buckets
- **Link**
  Contains the name of the objects
- **Immutable**
  Gold image
- **Custom**
  Defined by the users



Object Properties

Key Value Pair

Key for everything
- Metadata: BUCKET_NAME.UUID
- Block: BUCKET_NAME.UUID

Serial

Each element has a version which is identified by a serial.

Object Serialized

Backends agnostic on information stored

Default Root Object

For each bucket is defined a default root object with object id ROOT

Nosql Storage

Key: serialized object

Object

zebra.c1d2197420bd41ef24fc665f228e2c76e98da247

Segment-id

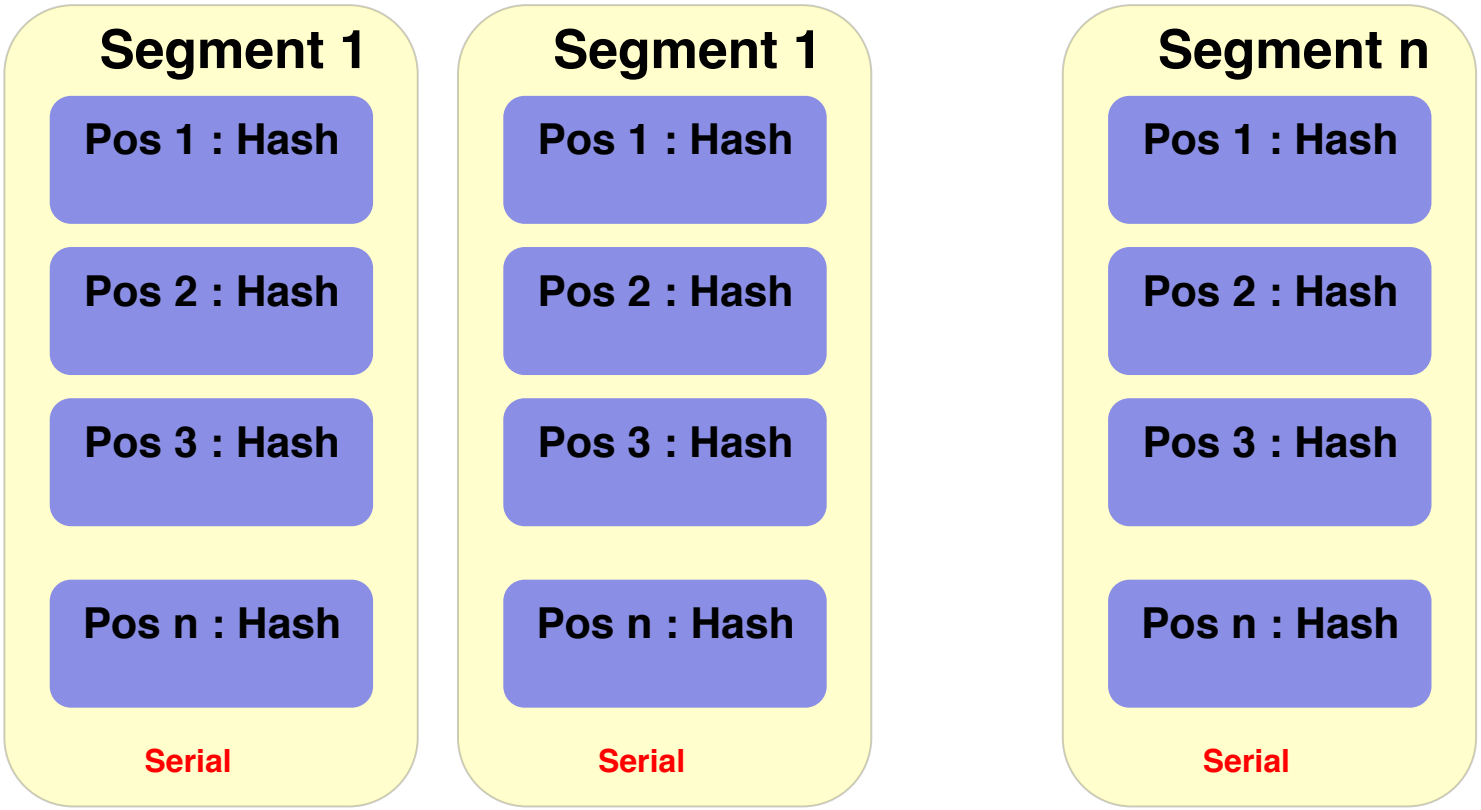
1:16db0420c9cc29a9d89ff89cd191bd2045e47378
2:9bcf720b1d5aa9b78eb1bcdbf3d14c353517986c
3:158aa47df63f79fd5bc227d32d52a97e1451828c
4:1ee794c0785c7991f986afc199a6eee1fa4
5:c3c662928ac93e206e025a1b08b14ad02e77b29d
…
vers:1335519328.091779

Property

segment_size= 512
block_size = 16k
content_type =
md5=ab86d732d11beb65ed0183d6a87b9b0
max_read'=1000
storage_class=STANDARD
compression= none
…

Segments

Segments

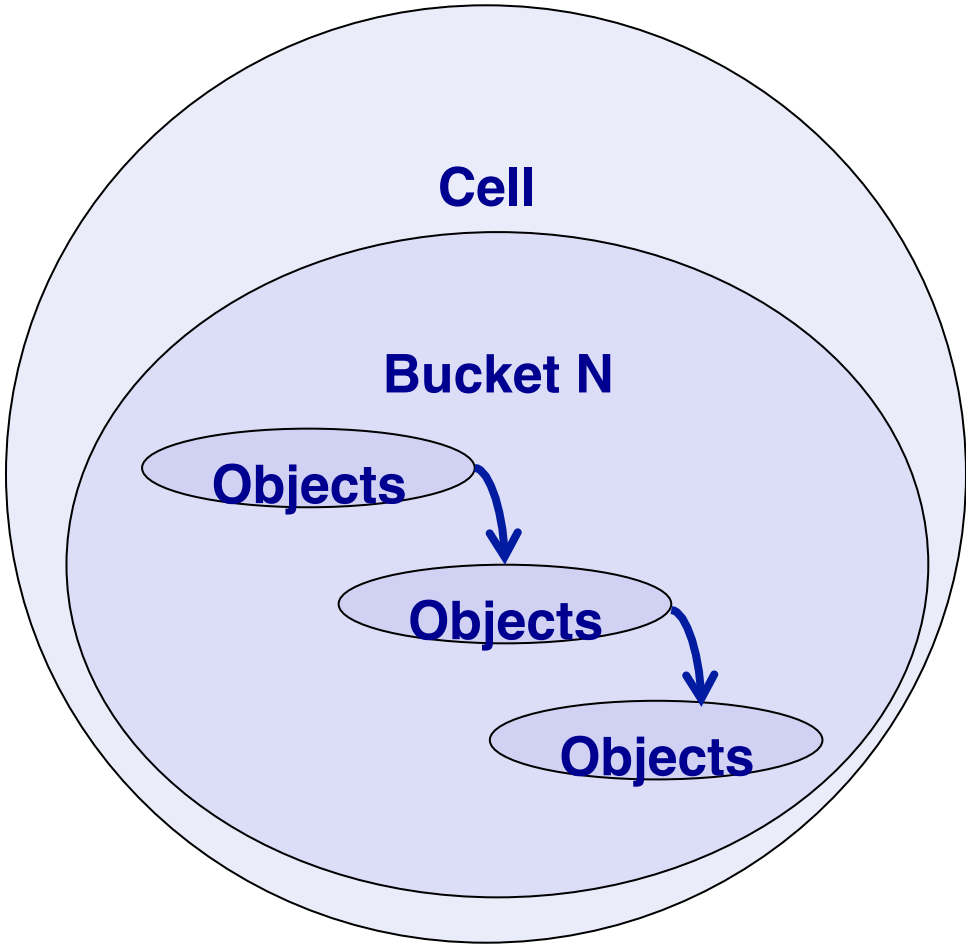


N Segment = (Object Size/block size)/segment size

“Serial vs Hash”

Object Versioning

- **Object Pointer**
  Object point to the previous one
- **Only CreateBlock**
  Client has to use only createBlock operation
- **New ID for the old Object**
  Segment Difference



Protocols

- DNS
- RestFS
- S3 Interface

RestFS Protocol

WebSocket

is a web technology for multiplexing bi-directional, full-duplex communications channels over a single TCP connection.

This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open…

```
GET /mychat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

JSON-RPC

is lightweight remote procedure call protocol similar to XML-RPC. It's designed to be simple

```
--> { "method": ”readBlock", "params": [”…"], "id": 1}
<-- { "result": [..], "error": null, "id": 1}
```

BSON

short for Binary JSON,
is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays.
BSON can be compared to binary interchange formats

```
{"hello": "world"}
→
"\x16\x00\x00\x00\x02hello\x00
 \x06\x00\x00\x00world\x00\x00"
```

Only Primitives

No objects or list

❑ Meta Operation

- Bucket
- Object id
- Operation
- List elements

❑ Operation Packets

Collect operation to single segment

❑ Parallel

- ❑ Single channel to meta data server
- ❑ Parallel channel to block storage (one per segment)

Locks

Locks vs No consistency

- **OpLocks**
- **Time base**
  Token base
- **Ordering**
  Conflict are management with the serial property

Cache

Publish–subscribe

"… is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Published messages are characterized into classes, without knowledge of what, if any, subscribers there may be. Subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what, if any, publishers there are… " Wikipedia

Pattern matching

Clients may subscribe to glob-style patterns in order to receive all the messages sent to channel names matching a given pattern.

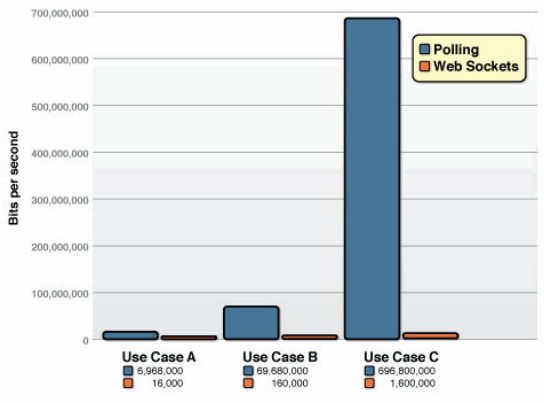
Distributed Cache Server Side

For server side the server share information over distributed cache to reduce the use of backend

Client Cache

Pre allocated block with circular cache
write-through cache

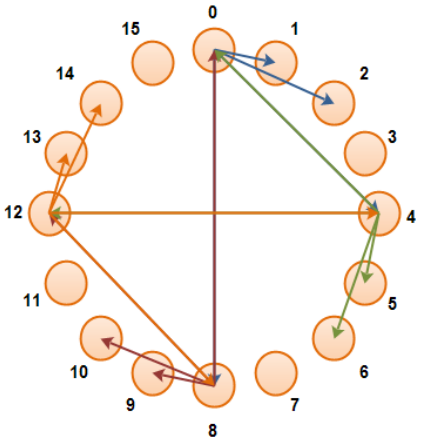
Demo http://www.websocket.org/echo.html



Use case A: 1,000
Use case B: 10,000
Use case C: 100,000

Block Storage

Backend: Storage



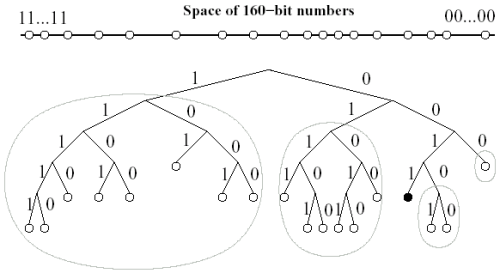


Fig. 1: Kademlia binary tree. The black dot shows the loation of node 0011... in the tree. Grey ovals show subtrees in which node 0011... must have a contact.

Kademlia's XOR distance is easier to calculate.

Kademlia's routing tables makes routing table management a bit easier.

Each node in the network keeps contact information for only log n other nodes

Kademlia implements a "least recently seen" eviction policy, removing contacts that have not been heard from for the longest period of time.

Key/value pair is stored on the node whose 160-bit nodeID is closest to the key

Closest node, send a copy to neighbor

Code

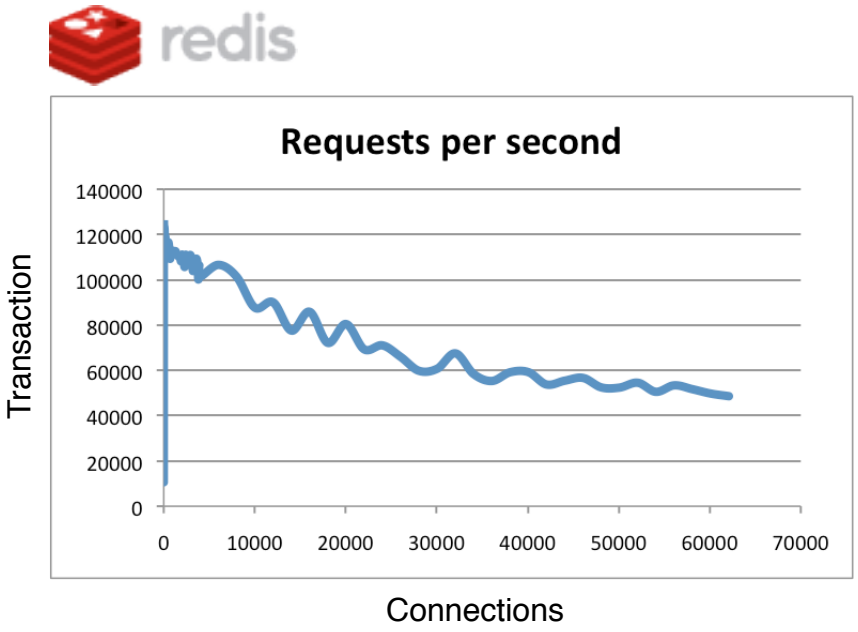
Pluggable

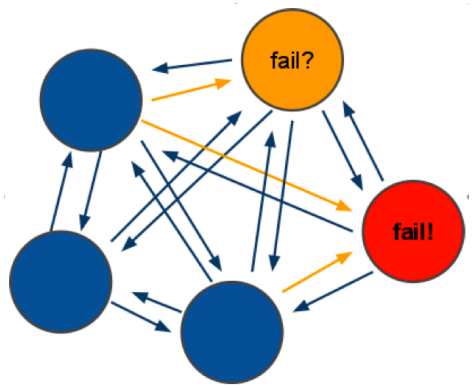
Protocol
- Connection Handler
- Data transcoding

Service
- High level Operations across multiple functions (like locking)
- Integrity operations/transaction

Manager
- Operations handler for specific area (ex. metadata)
- Split info in sub info

Driver
- Read and write operation to storage system, agnostic operation





Key Value

- Key in memory
- Value on disc

Example of benchmark result

The test was done with 50 simultaneous clients performing 100000 requests.
The value SET and GET is a 256 bytes string.
The Linux box is running Linux 2.6, it's Xeon X3320 2.5 GHz.
Text executed using the loopback interface (127.0.0.1).

Cluster

Multi-master
Auto recovery

What we are using

Module	Software
Storage	Filesystem, DHT (kademlia, Pastry*)
Metadata	SQL(mysql,sqlite), Nosql (Redis)
Auth	Oauth(google, twitter, facebook), kerberos*, internal
Protocol	Websocket
Message Format	JSON-RPC 2.0, Amazon S3
Encoding	Plain, bson
CallBack	Subscribe/Publish Websocket/Redis, Async I/O TornadoWeb, ZeroMQ*
HASH	Sha-XXX, MD5-XXX, AES
Encryption	SSL, ciphers supported by crypto++
Discovery	DNS, file base

* are planned

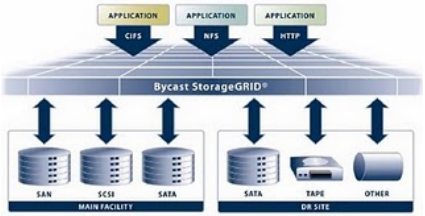
What is it good for ?

User

- Home directory
- Remote/Internet disks

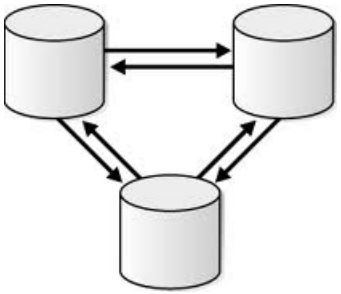
Application

- Object storage
- Shared space
- Virtual Machine



Distribution

- CDN (Multimedia)
- Data replication
- Disaster Recovery



Advantages

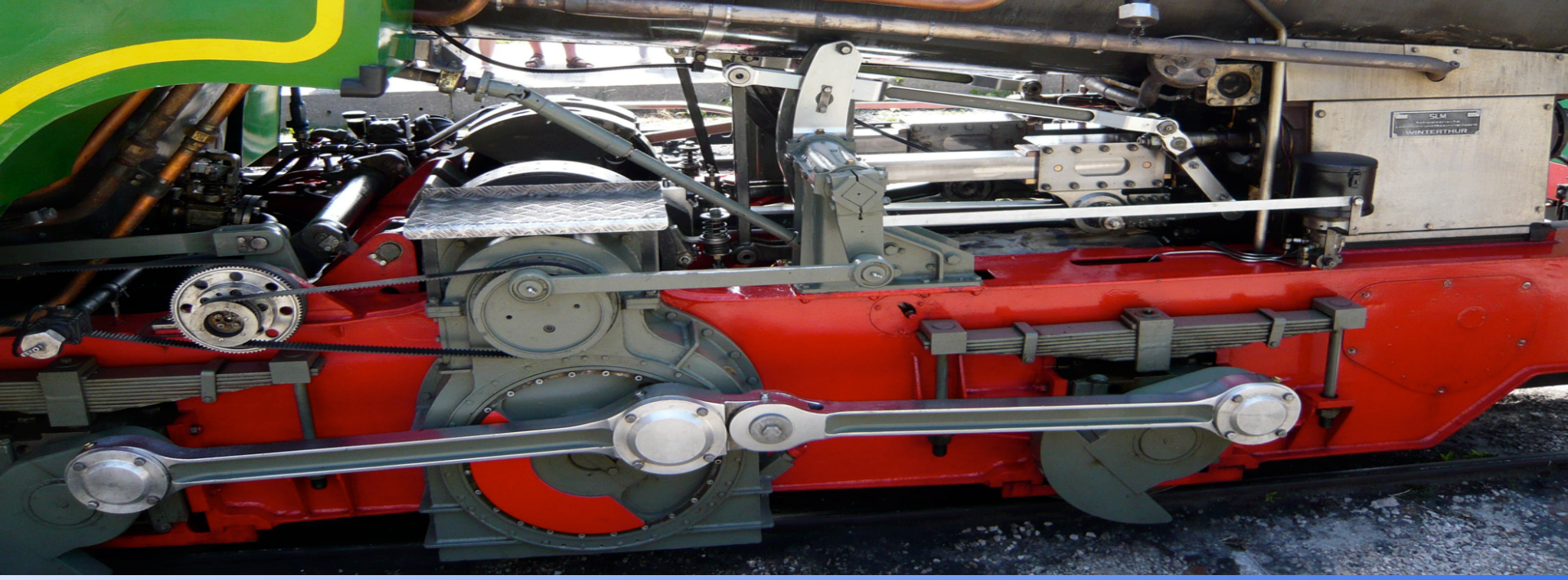
High reliability	Distributed
	Decentralized
	Data replication
Nearly unlimited scalability	Horizontal scalability
	Multi tier scalability
Cost-efficient	Cheap HW
	Optimized resource usage
Flexible	User Property
	Extended values and info
Enhanced security	Extended ACL
	OAUTH / Federation
	Encryption
	Token for single device
Simple to Extend	Plugin
	Bricks

Roadmap

- **0.1**
  - Single server on storage (No DHT)
  - S3 Interface
  - Federated Authentication
- **0.2 Release (coming soon)**
  - DHT on storage
  - Storage Encryption and compression
  - FUSE
- **0.3 Release TBD (codename WorstFS++)**
  - Deduplication
  - pub/sub
  - ACL
  - …
- **Next**

Disconnected operation, Logging, Locks, Dlocks, Bucket automate provisioning, Distribution algorithms, Load balancing, samba module, more async i/o, block replication control, negative cache, index, user defined index

Support



Thank you

http://restfs.beolink.org

manfred.furuholmen@gmail.com
fege85@gmail.com

Beolink.org