COSBench: Cloud Object Storage Benchmark

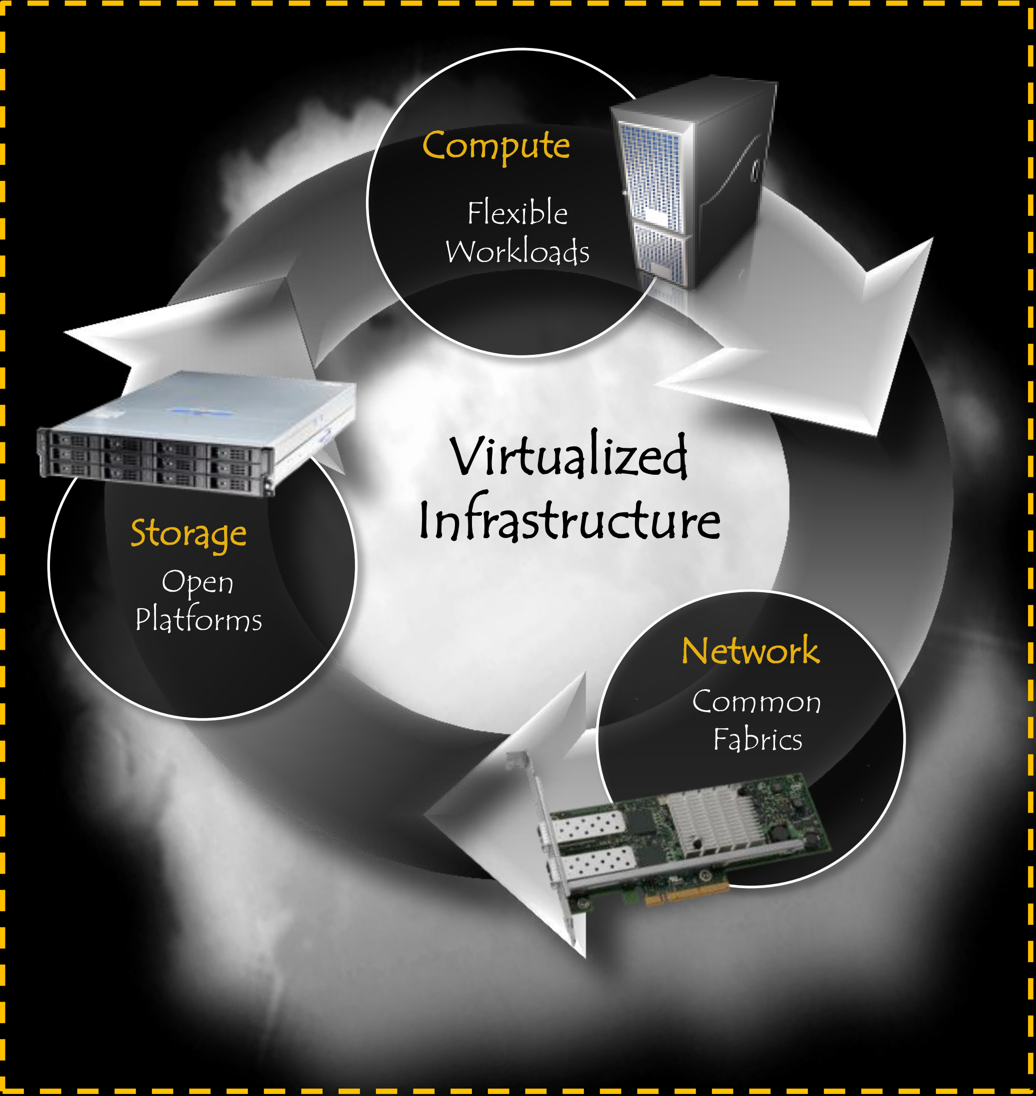
Qing Zheng
Reliable, Scalable, Intelligent Systems Group
School of Software, Shanghai Jiao Tong University
http://reins.se.sjtu.edu.cn

Oct. 12nd, 2012

qzheng2010@hotmail.com

Data Centers are Evolving!

REINS



- Data centers are built upon 3 fundamental pillars:
  - Compute
  - Network
  - Storage
- To achieve high efficiency in performance & utilization
  - A balanced data center is essential!

Storage Capacity Growth

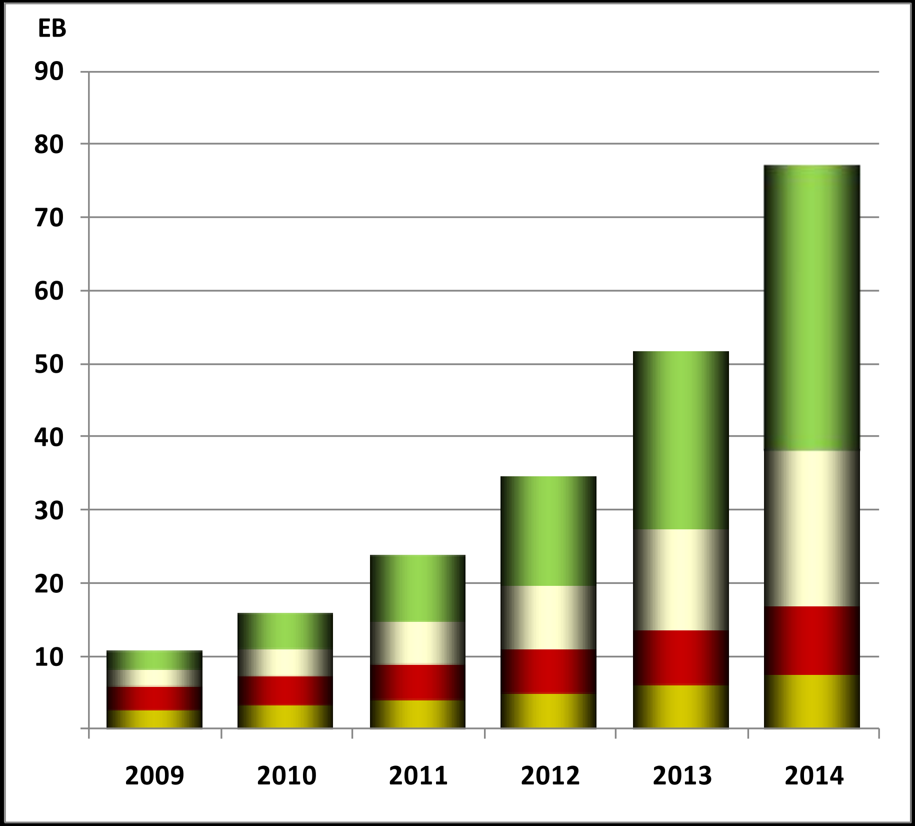
REINS

Structured data (23.6% ↑)
- Traditional enterprise database

Replicated data (24.2% ↑)
- Backups
- Data warehouses

Unstructured data (54.8% ↑)
- Archives

Content Depots (75.6% ↑)
- Web
- Email
- Document sharing
- Social network content (pictures/videos)



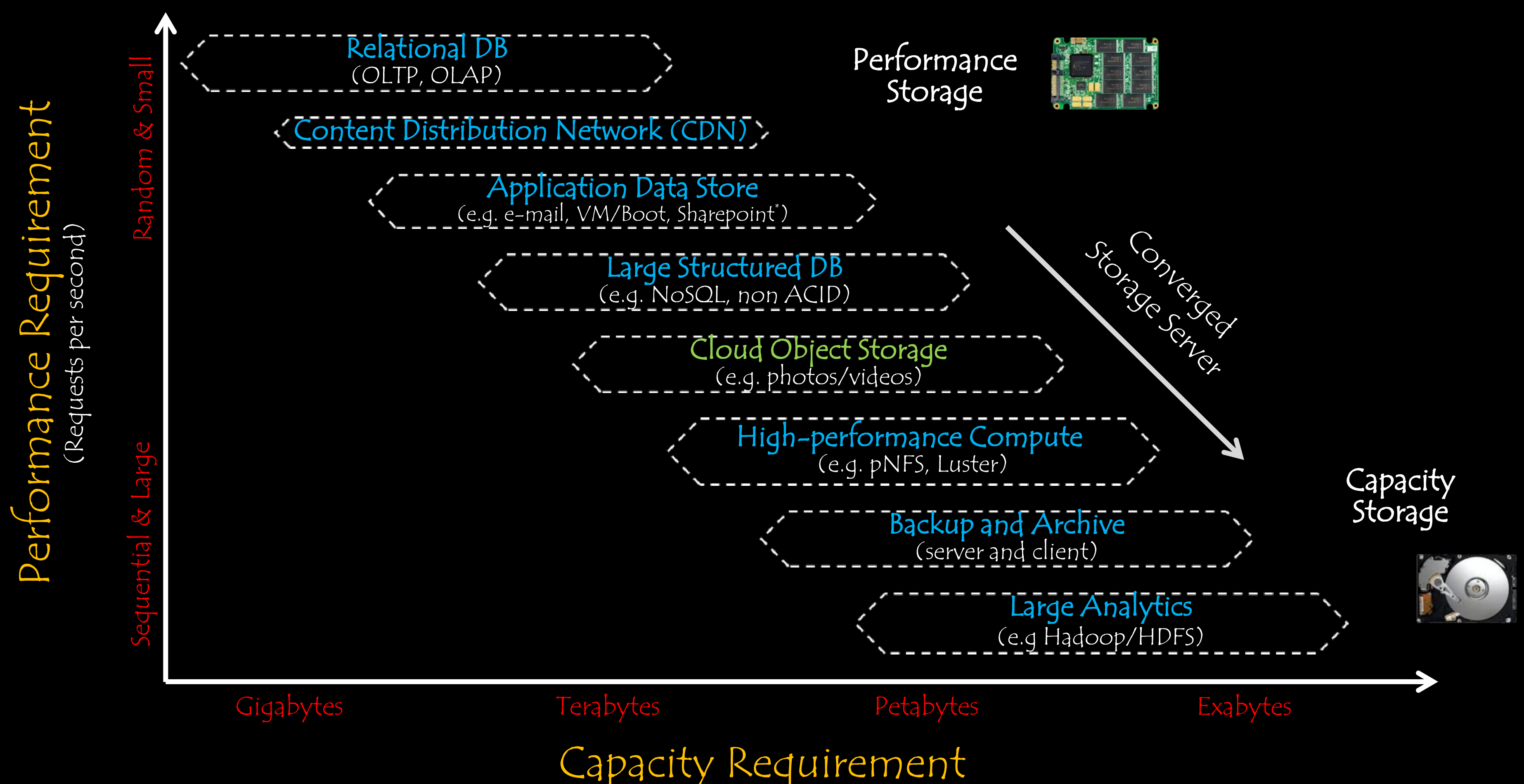
2012 deployment estimation

~ 7.6 million drives

~ 500,000 storage systems

Solutions dictated by Usage Models

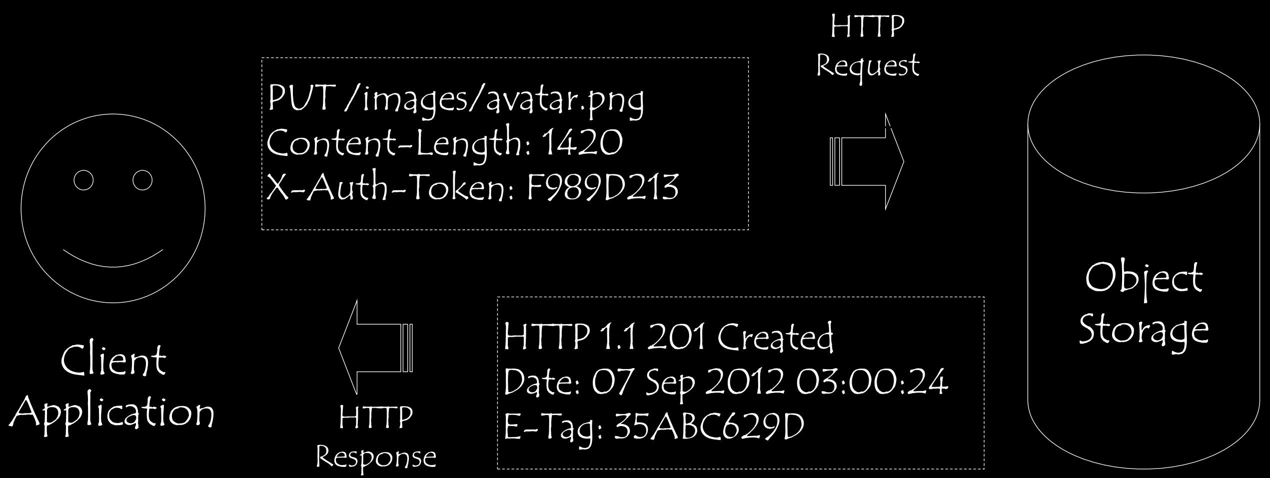
REINS



Different storage usage models create different ecosystems!

REINS

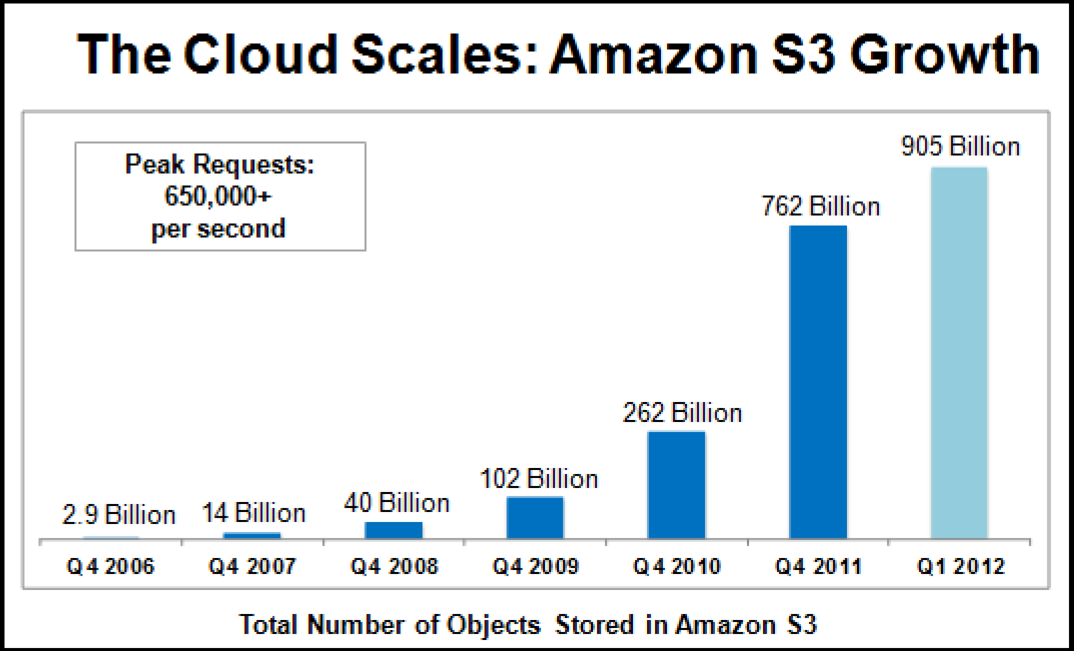
- Storage Interface



HEAD / GET / PUT / POST / DELETE / COPY

Object Storage in Getting Accepted

REINS



Object storage have been increasingly recognized as a right destination for data outsourcing

Challenges …

REINS

- The proliferation of existing offerings
  - Amazon S3, Rackspace Cloud Files, Google Cloud Storage, HP Cloud Object Storage, Windows Azure, EMC ATMOS, Openstack Swift, Ceph, …
  - coupled with a lack of workload modeling object storage apps
  - makes it difficult for one to choose the right infrastructure!

- Tuning systems to their optimal performance is also nontrivial
  - resulted from the complexity in the designs of various available solutions

We need a benchmark tool
dedicated for object storage system!

We present COSBench

REINS

- People can use this tool to evaluate & compare different
  - hardware and software stacks
  - and obtain a better understanding of these offerings

Throughput / Latency / Scalability / Flexibility

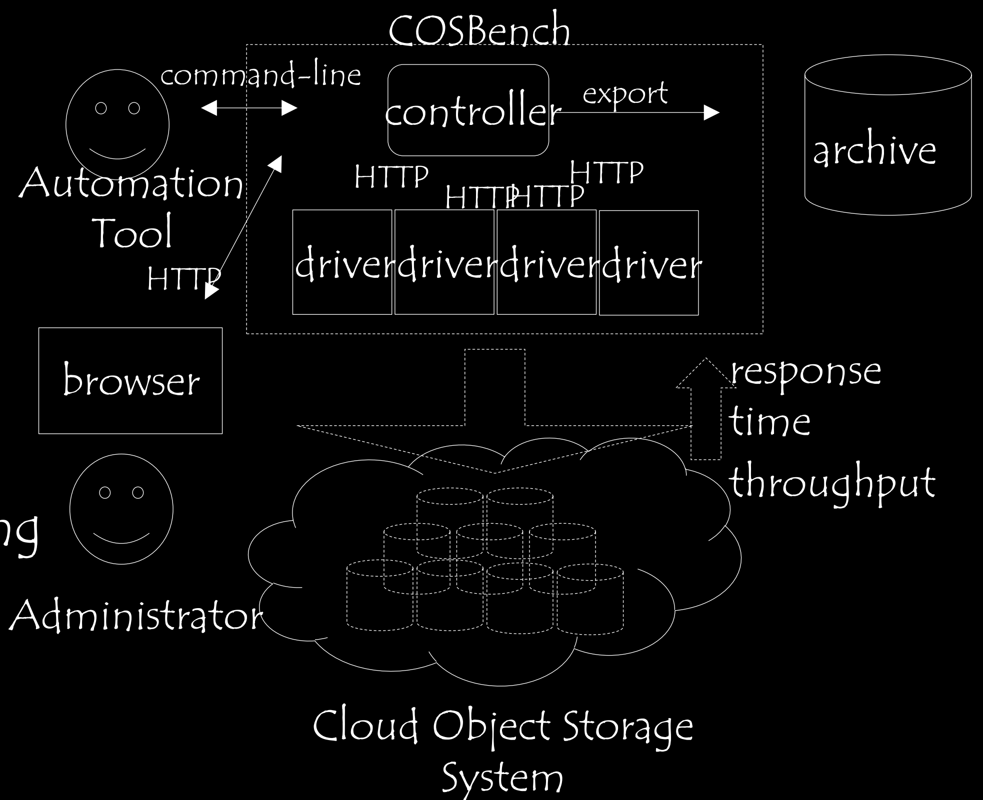
- People can also use this tool to characterize their systems
  - and get insights to guide system designs, tuning, & optimization

Architecture Decisions / Algorithms / Hardware Impacts / Innovation

COSBench Key Component

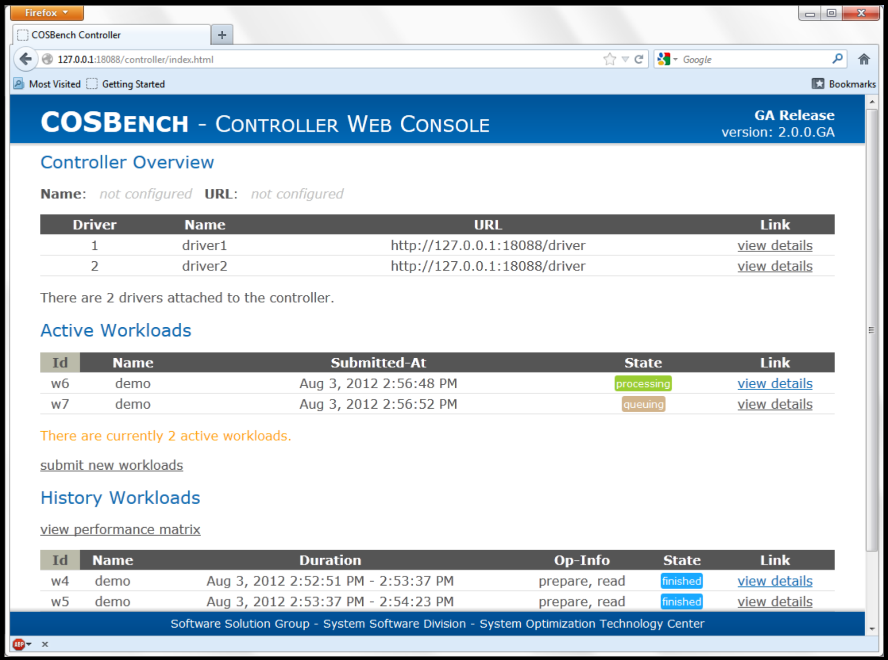
REINS

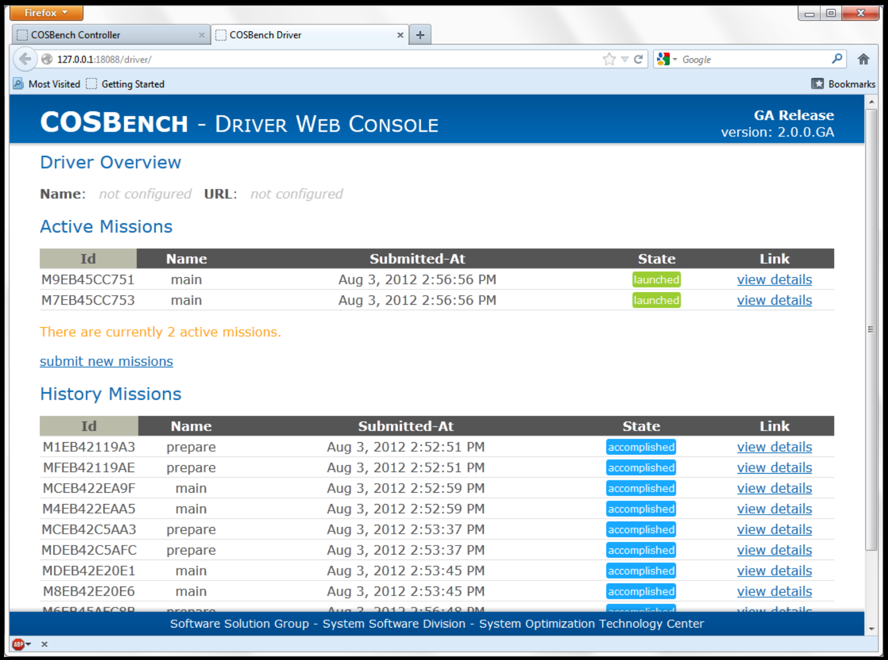
- Config.xml:
  - define workload with flexibility.
- Controller:
  - Control all drivers
  - Collect and aggregate stats.
- Driver:
  - generate workload according to config parameters.
  - can run tests without a supervising controller.
- Web Console:
  - System facade
  - Browse real-time stats
  - HTTP based Communication (RESTful style)



Web Console

REINS





Controller

Driver

DEMO

Workload Configuration

REINS

```
- <workflow>
  - <workstage name="init">
      <work type="init" workers="8" config="containers=r(1,32)" />
    </workstage>
  - <workstage name="prepare">
      <work type="prepare" workers="8" config="containers=r(1,32);objects=r(1,50);sizes=c(64)KB" />
    </workstage>
  - <workstage name="main">
    - <work name="main" workers="8" rampup="100" runtime="300">
        <operation type="read" ratio="80" config="containers=u(1,32);objects=u(1,50)" />
        <operation type="write" ratio="20" config="containers=u(1,32);objects=u(51,100);sizes=c(64)KB" />
      </work>
    </workstage>
  - <workstage name="cleanup">
      <work type="cleanup" workers="8" config="containers=r(1,32);objects=r(1,50)" />
    </workstage>
  - <workstage name="dispose">
      <work type="dispose" workers="8" config="containers=r(1,32)" />
    </workstage>
  </workflow>
</workload>
```

Flexible Load Control

Path/Size Distribution

Read/Write Patterns

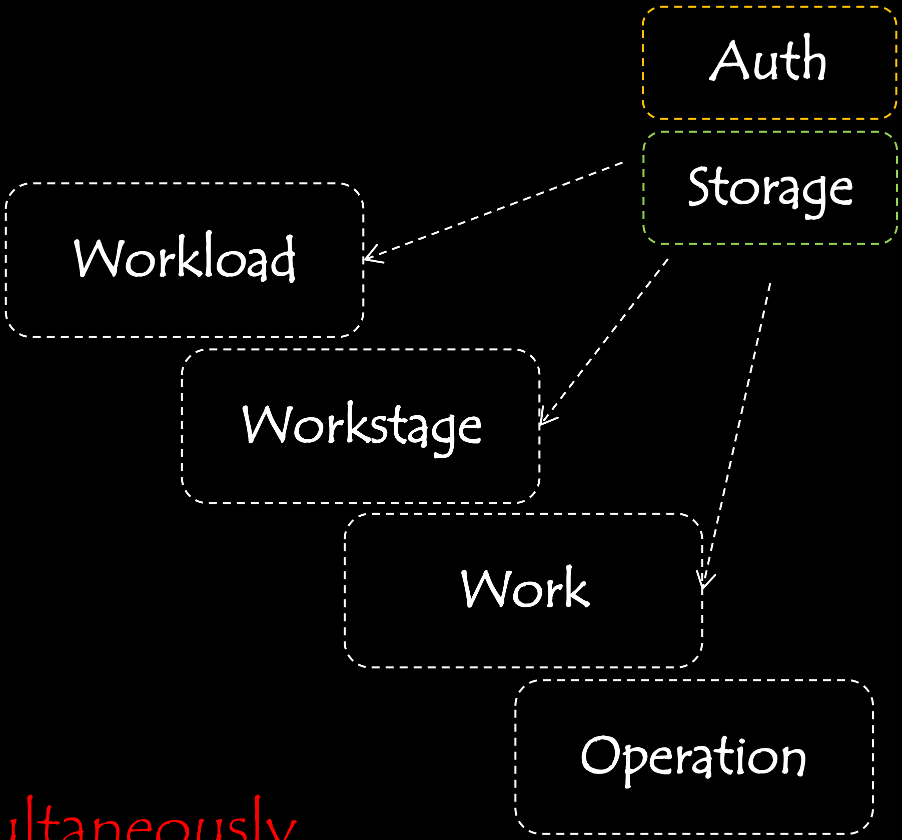
Extensible Parameters

Workflow Model

Flexible configuration which gives birth to diverse usage patterns

Configurable workload

- Mixed operations (GET/PUT…)
- Mixed object sizes
- Multiple stages
- Auth/storage association
- Load control
- Extensible parameters

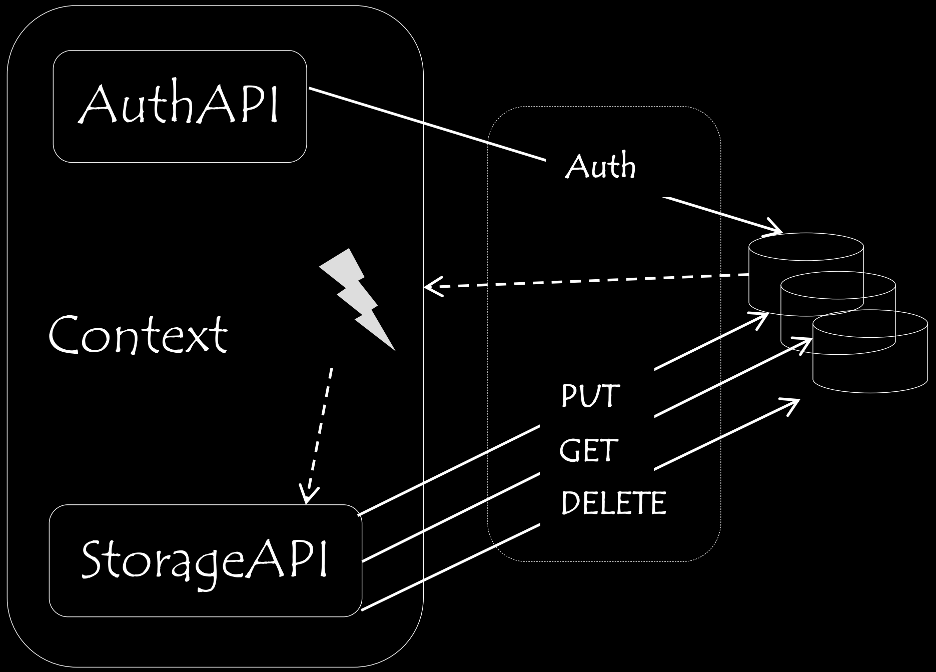


Stress multiple systems simultaneously

Extensible API

REINS

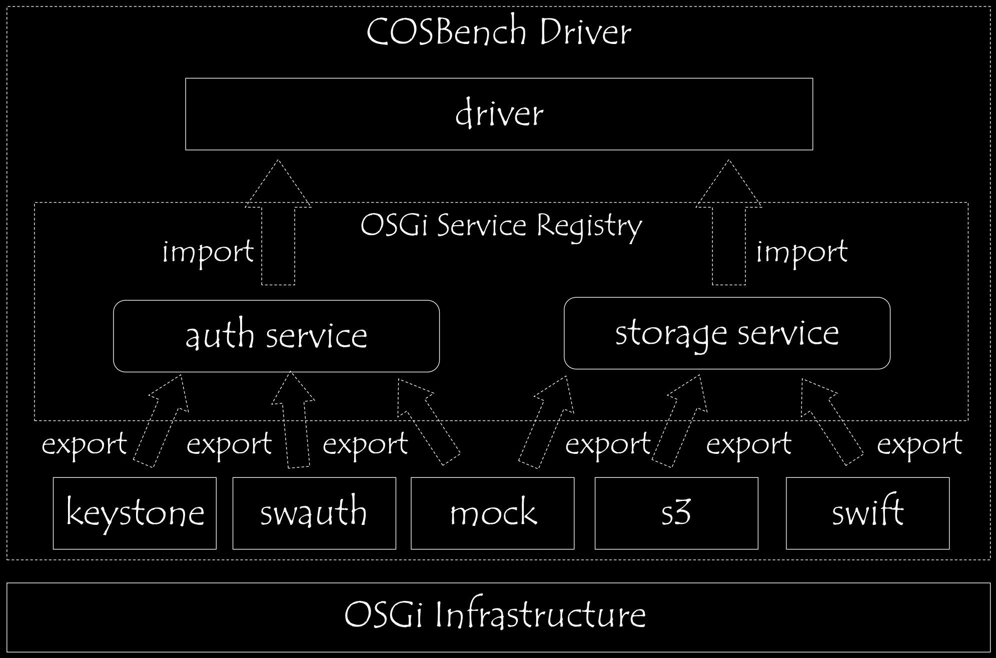
- Separate auth and storage API, so
  - One auth → multiple storages
  - One storage → multiple auths



Extensible API which support various storage systems

Modular Design

REINS



New adaptors can be separately developed, individually configured, and dynamically plugged into the env. without the knowledge of the core system

Performance Metrics

REINS

- Throughput (Operations/s): the operations completed in one second
- Response Time (in ms): the duration between operation initiation and completion.
- Bandwidth (KB/s): the total data in KiB transferred in one second
- Success Ratio (%): the ratio of successful operations

Swift

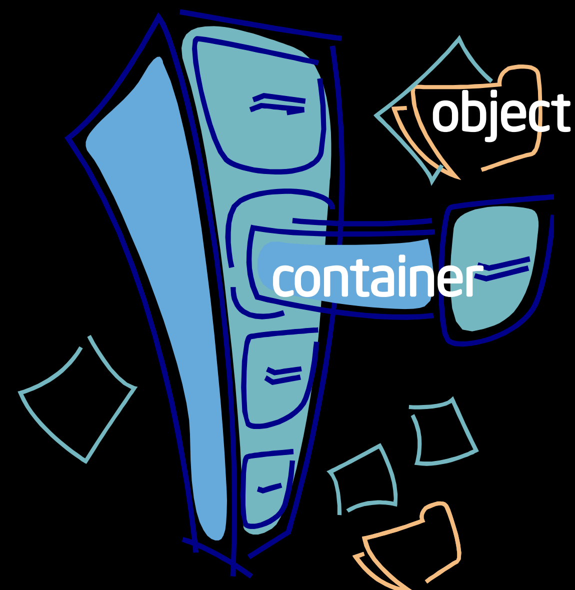
- Object Storage for Openstack

Main Features

REINS

- As an object storage system, Swift:

1. allows users to create containers and to stores data objects in these containers
   - *objects are identified by their paths and have metadata associated with them*
2. can be accessed via "RESTFul" interface
   - *including "GET"/"PUT"/"DELETE"*
3. can be well built upon commodity storage devices and is highly scalable
   - *achieving cost effectiveness*
4. is redundant and is eventually consistent
   - *suitable for long-term storage*

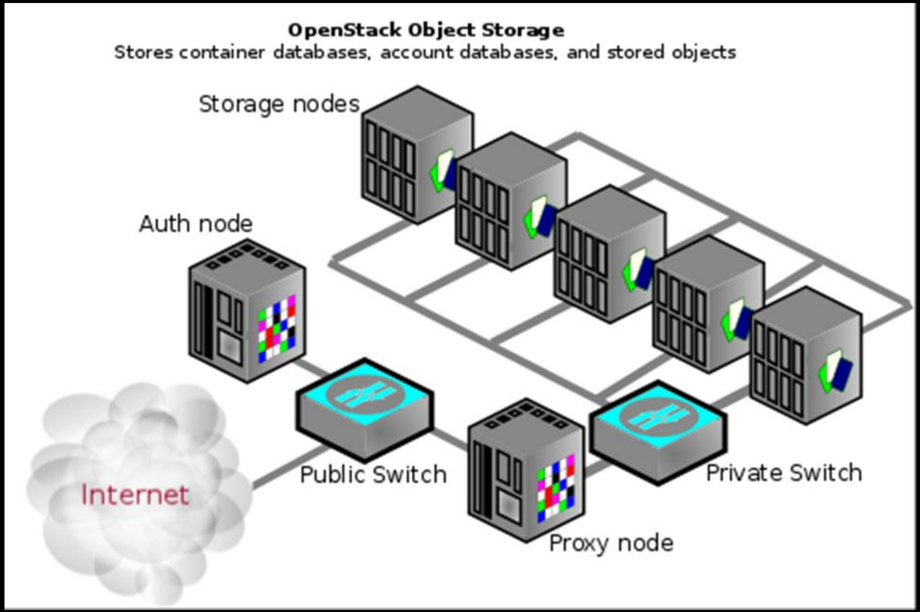


As you see, container cannot be nested

Services on Proxy/Storage Node

REINS

- Proxy Node
  - proxy-server
    - cluster gateway
  - swauth-server
    - authentication & authorization

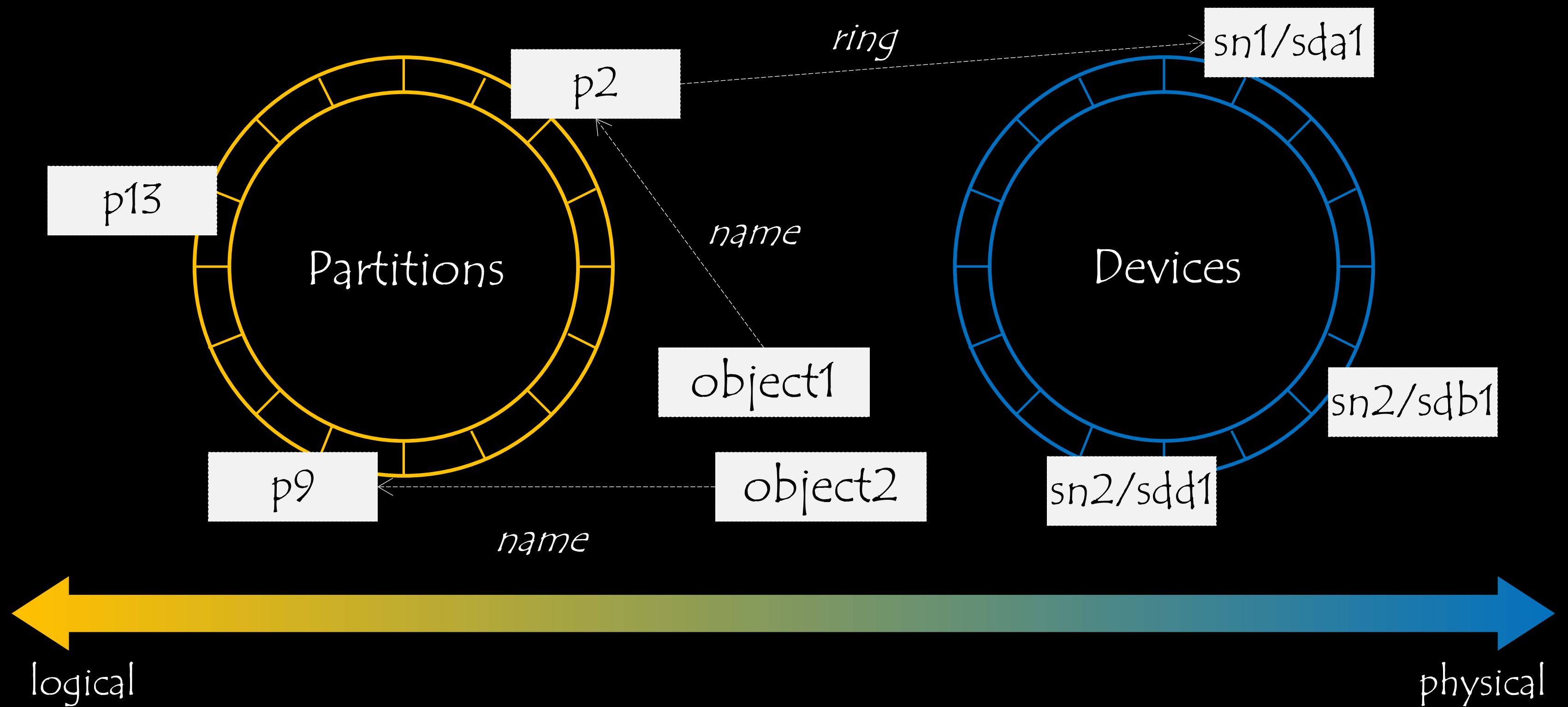


- Storage Node
  - account/container/object - server
    - listing containers
    - listing objects
    - saving/retrieving/removing objects
  - account/container/object - replicator
    - pushing local replicas to other storage nodes should replicas of those nodes are missing
  - account/container/object - auditor
    - quarantining local corrupted data entities
  - container/object -updater
    - updating metadata asynchronously

Load Balance

REINS

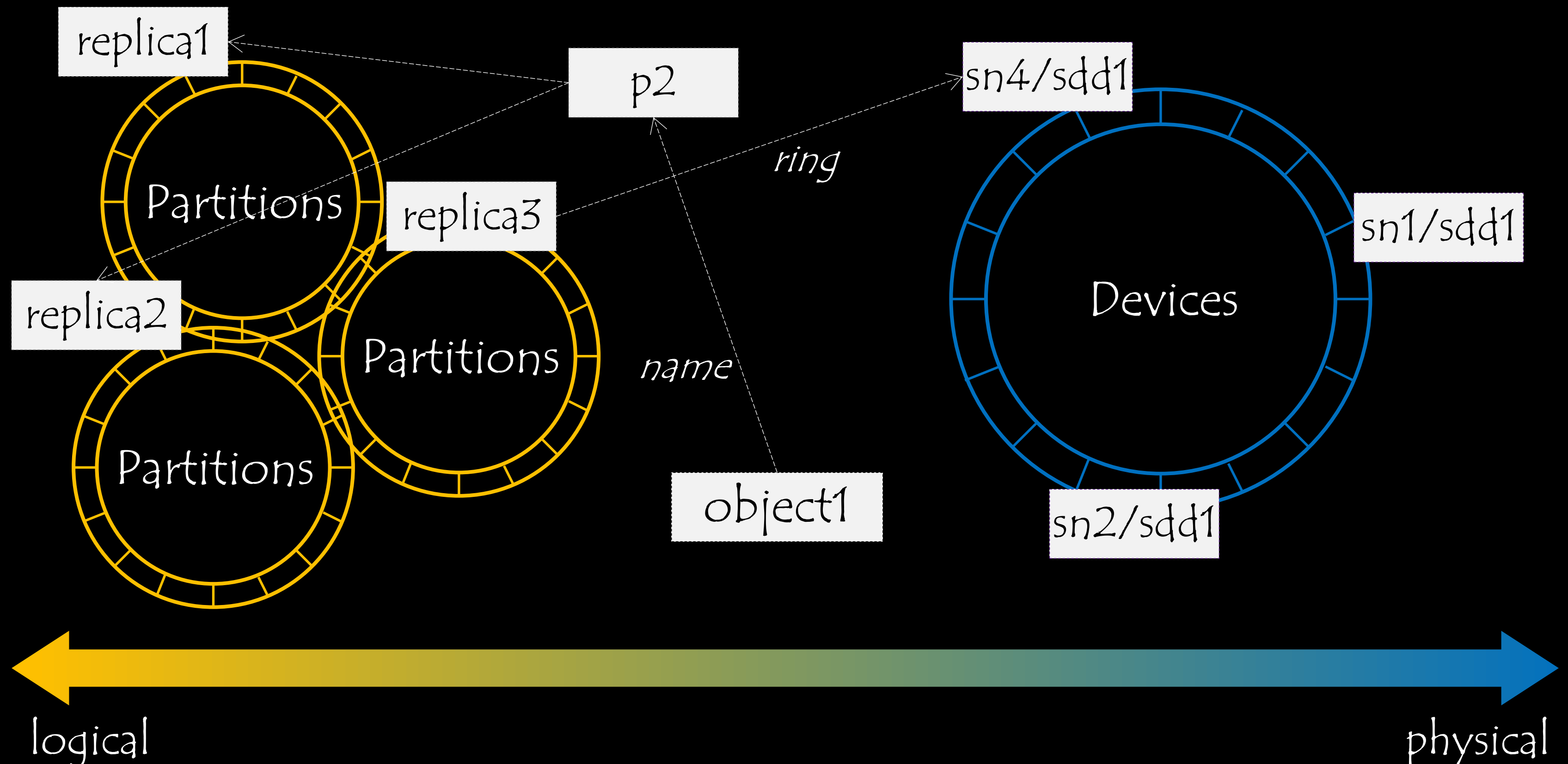
- A ring maintains its mapping using
  - physical perspective: devices
  - logical perspective: partitions



Load Balance

REINS

- A ring maintains its mapping using
  - physical perspective: devices
  - logical perspective: partitions -> replicas



How ring is initialized?

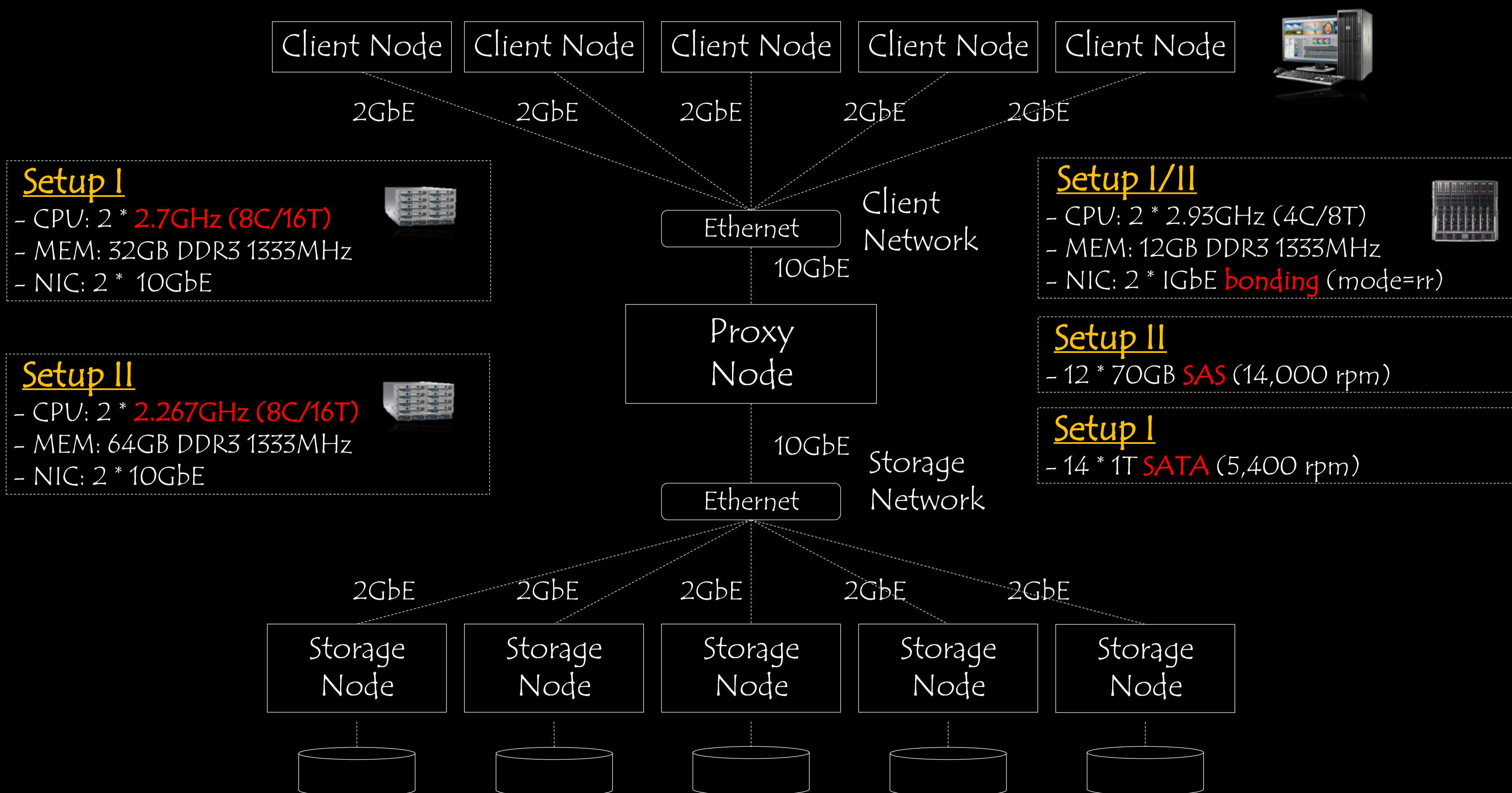
REINS

- In ring, each device is associated with metadata
  - describing device's weight, # replica wanted and # replica assigned
- A ring is built as
  - replicas from each partitions are in turn assigned to
  - the device currently enjoying the highest value of
    - "# replica wanted - # replica assigned"
  - Note that best attempts will be made to assign replicas from a sample partition to different zones or at least different nodes if possible

System Configuration

REINS

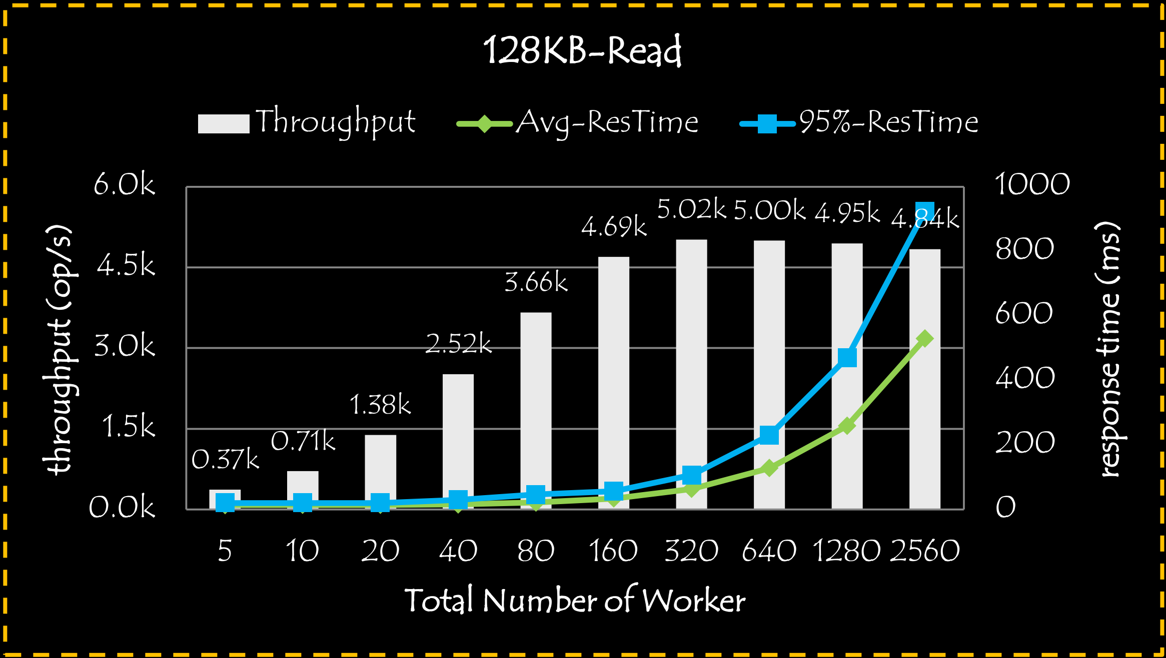
Setup-I had higher CPU power, Setup-II had faster disks



A: 128KB-Read

REINS

- SLA: 200ms + 128KB/1MBps = **325ms**



Workers	95%-ResTime	Throughput
	ms	op/s
5	20.00	369.49
10	20.00	711.24
20	20.00	1383.30
40	30.00	2517.94
80	46.67	3662.71
160	56.67	4693.97
320	106.67	5019.85
640	230.00	4998.13
1280	470.00	4947.15
2560	923.33	4840.19

The **bottleneck** was identified to be the **proxy's CPU**

-- The CPU utilization at that node was **93**%!

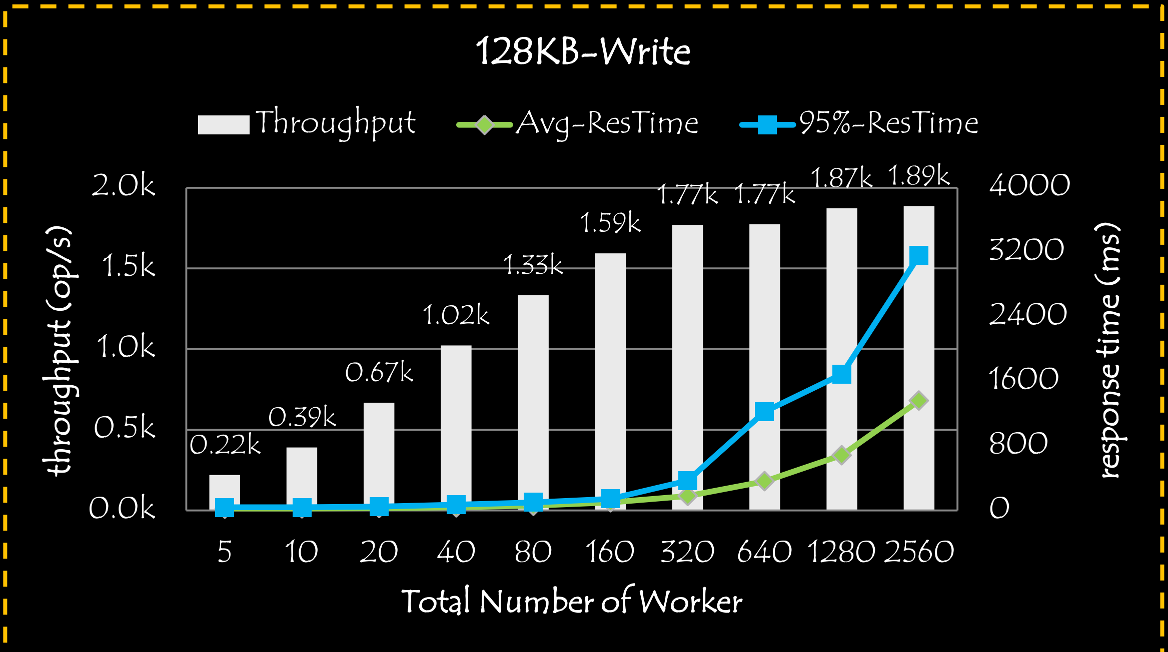
-- The peak throughput for setup-I was **5576** op/s (640 workers)

↑ CPU could ↑ throughput

B: 128KB-Write

REINS

- SLA: 200ms + 128KB/1MBps = 325ms



Workers	95%-ResTime	Throughput
	ms	op/s
5	40.00	219.73
10	40.00	391.14
20	50.00	668.19
40	70.00	1022.07
80	100.00	1333.34
160	143.33	1594.12
320	370.00	1769.55
640	1223.33	1773.12
1280	1690.00	1871.58
2560	3160.00	1886.81

The **Disks** at storage nodes had significant impact on overall throughput

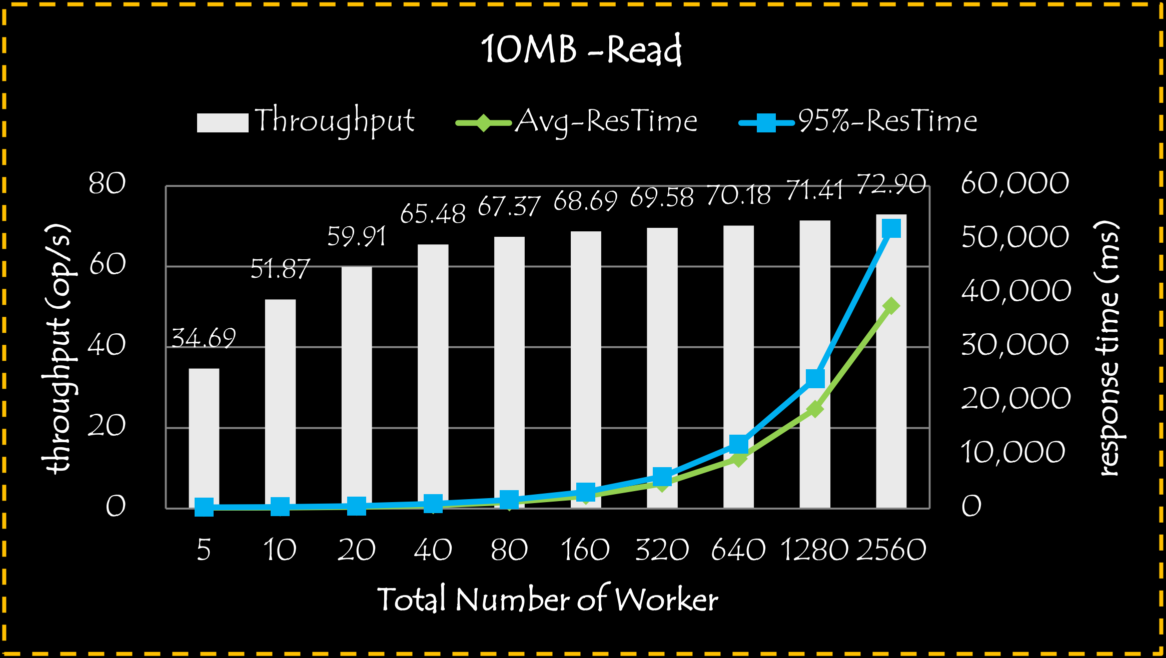
-- The peak throughput for setup-I was only **155** op/s (20 clients)

-- HDD → SSD in setup-I would ↑ throughput to **1621** op/s (320 clients)

storage disks ←→ throughput

C: 10MB-Read

- SLA: 200ms + 10MB/1MBps = **1200ms**



Workers	95%-ResTime	Throughput
	ms	op/s
5	270.00	34.69
10	320.00	51.87
20	480.00	59.91
40	900.00	65.48
80	1636.67	67.37
160	3093.33	68.69
320	5950.00	69.58
640	11906.67	70.18
1280	24090.00	71.41
2560	52090.00	72.90

The **bottleneck** was identified to be the **clients' NICs**

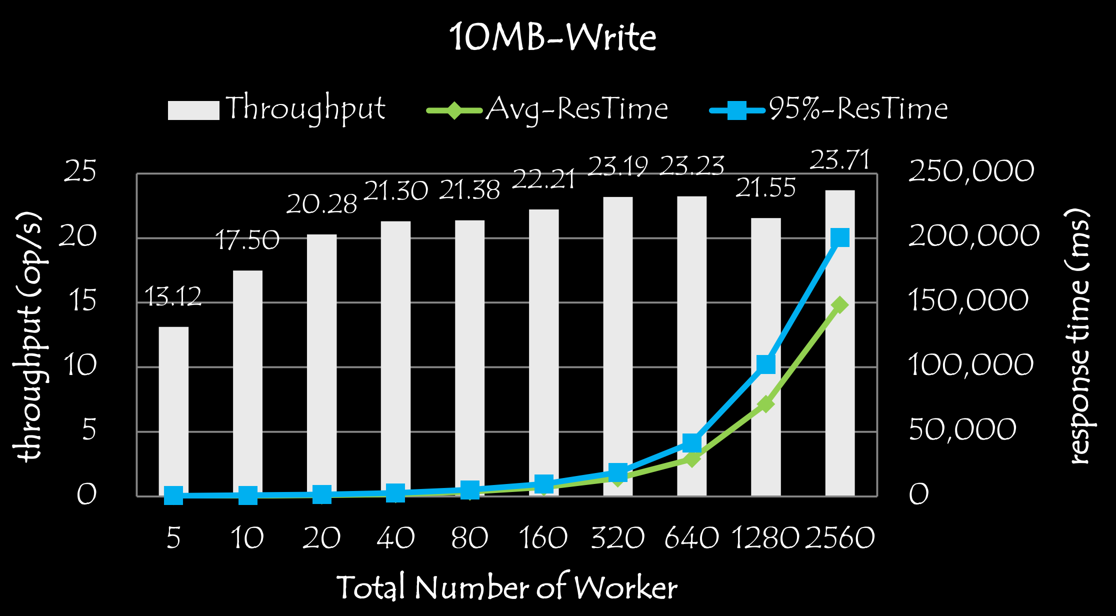
-- in setup-II, adding 5 more clients ↑ throughput to 103 op/s (80 clients)

-- in setup-I, using vClient + SRIOV could achieve similar improvements

D: 10MB-Write

REINS

- SLA: 200ms + 10MB/1MBps = <u>1200ms</u>



Workers	95%-ResTime ms	Throughput op/s
5	536.67	13.12
10	**936.67**	**17.50**
20	1596.67	20.28
40	2786.67	21.30
80	5133.33	21.38
160	9800.00	22.21
320	18623.33	23.19
640	41576.67	23.23
1280	102090.00	21.55
2560	200306.67	23.71

The **bottleneck** might be the **storage nodes' NICs**

-- in setup-I, the peak throughput was 15.74 op/s (10 clients)

-- in both setups, the write performance was 1/3 of the read performance

Thank you!