Racs-EV-Java

Gary Zibrat (gdz4)

Motivation

- Large organizations want to store data in the cloud (e.g. Library of Congress, Netflix, Reddit)
- Not only does do users pay per byte of data currently in the cloud, but also per byte of data transferred to and from.

Current Prices

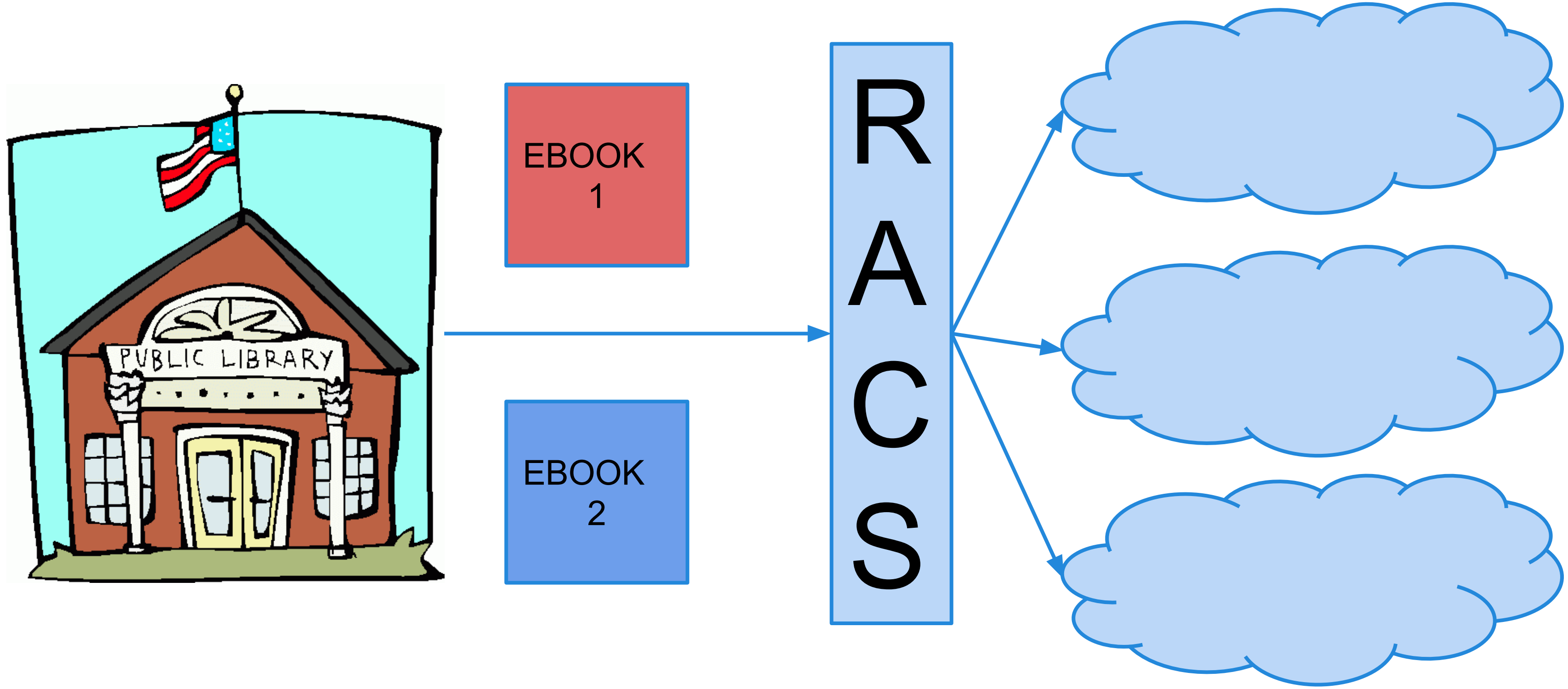
	Storage	Transfer out	Put Request	Get Request
Microsoft	$0.024 per GB/month	$0.080 per GB	$0.000036 per 1,000 transactions	$0.000036 per 1,000 transactions
Amazon	$0.0290 per GB/month	$.080 per GB	$0.005 per 1,000 requests	$0.0004 per 1,000 requests
Google	$0.026 per GB/month (flat rate)	$0.080 per GB	$0.01 per 1,000 requests	$0.001 per 1,000 requests

Current Prices Example (500TB)

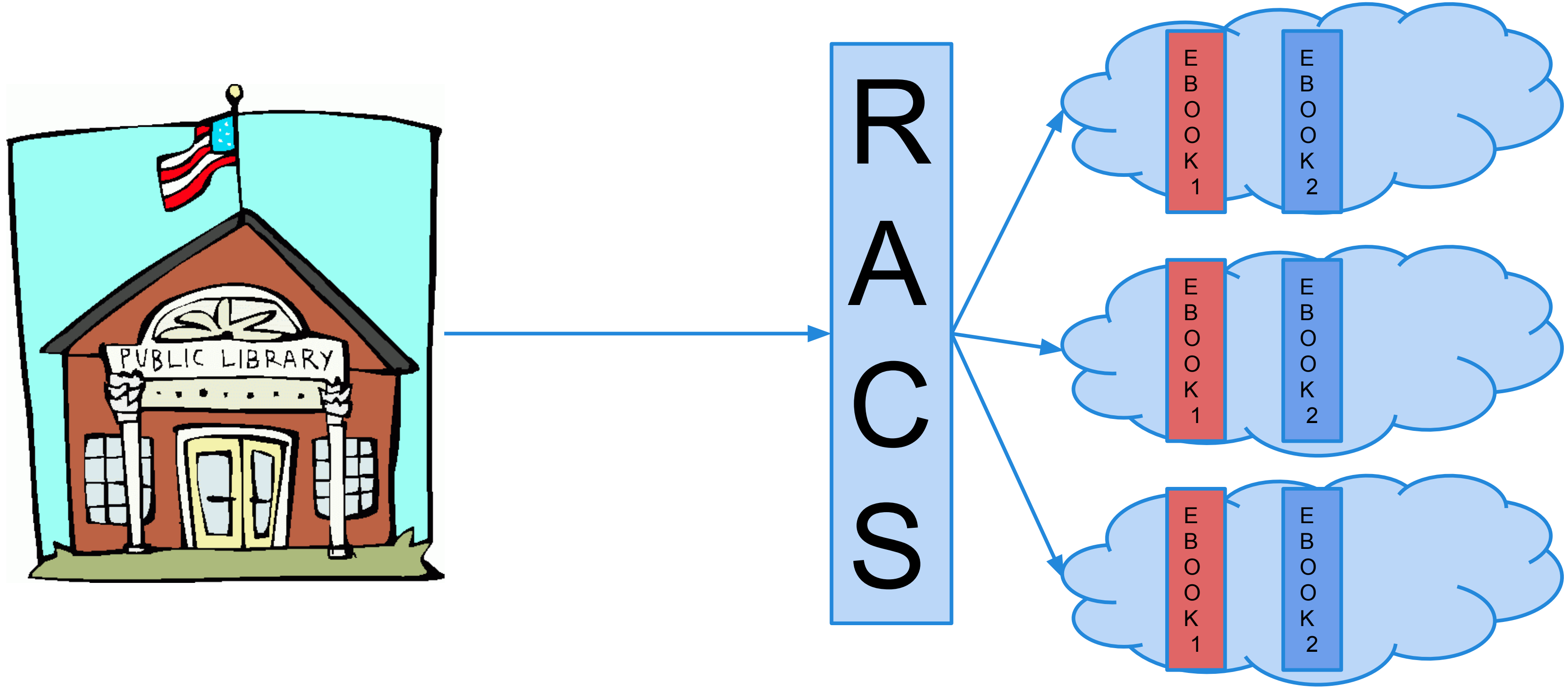
	Storage	Transfer out (all data)	Put Request	Get Request
Microsoft	$147456 per year	$491520	~0	~0
Amazon	$178176 per year	$491520	~0	~0
Google	$159744 per year	$491520	~0	~0

Moving from Amazon to Microsoft would cost roughly 2.75 years worth of storage! Large customers can’t leave due to slight price increases.

Original Racs

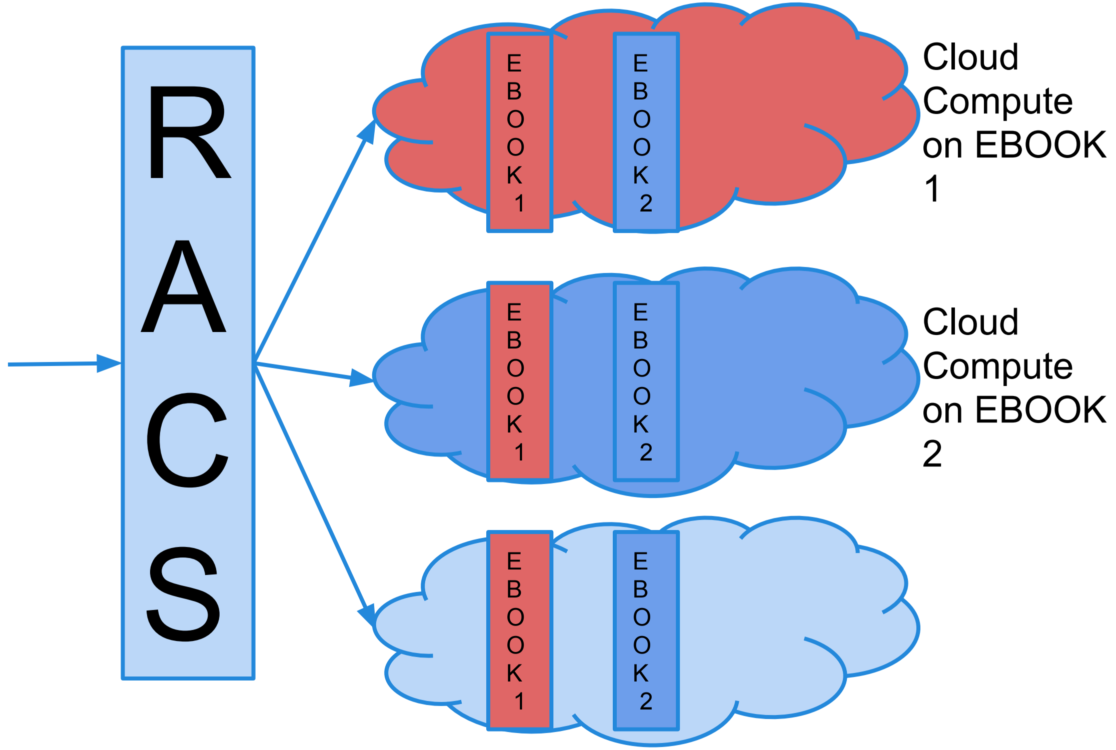


Original Racs

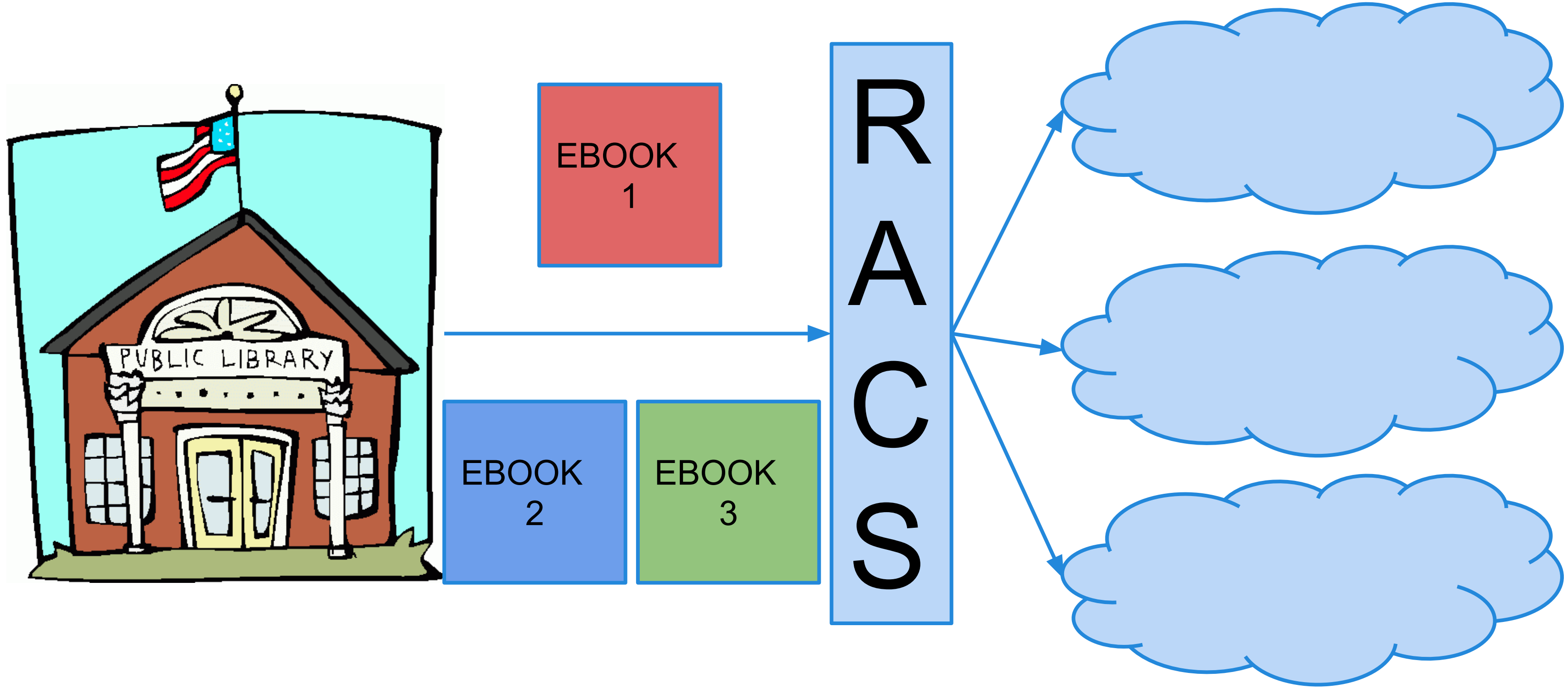


Original Racs

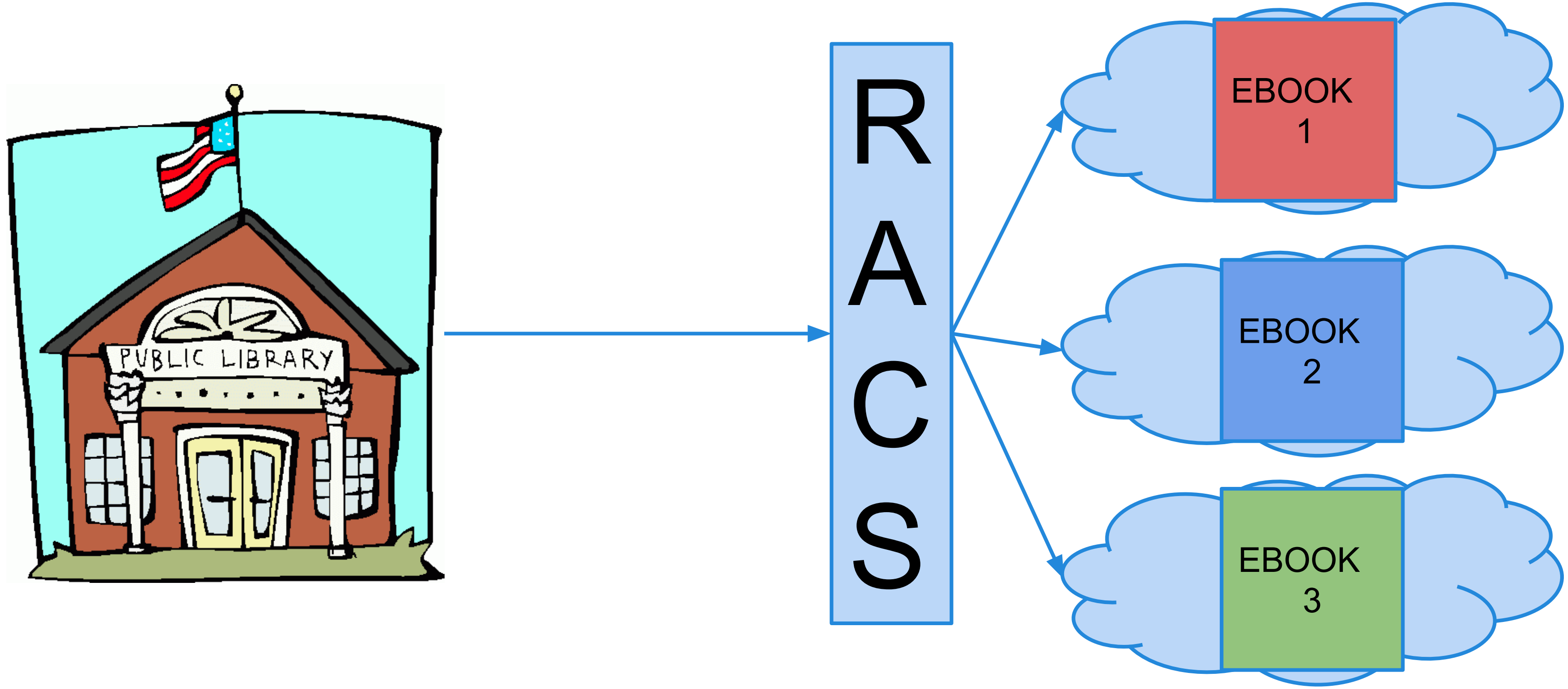
Since files are split up, cloud computation requires reassembling the files. Only part of the file may be in the same provider as where a user wishes to do cloud computation.



Racs-EV

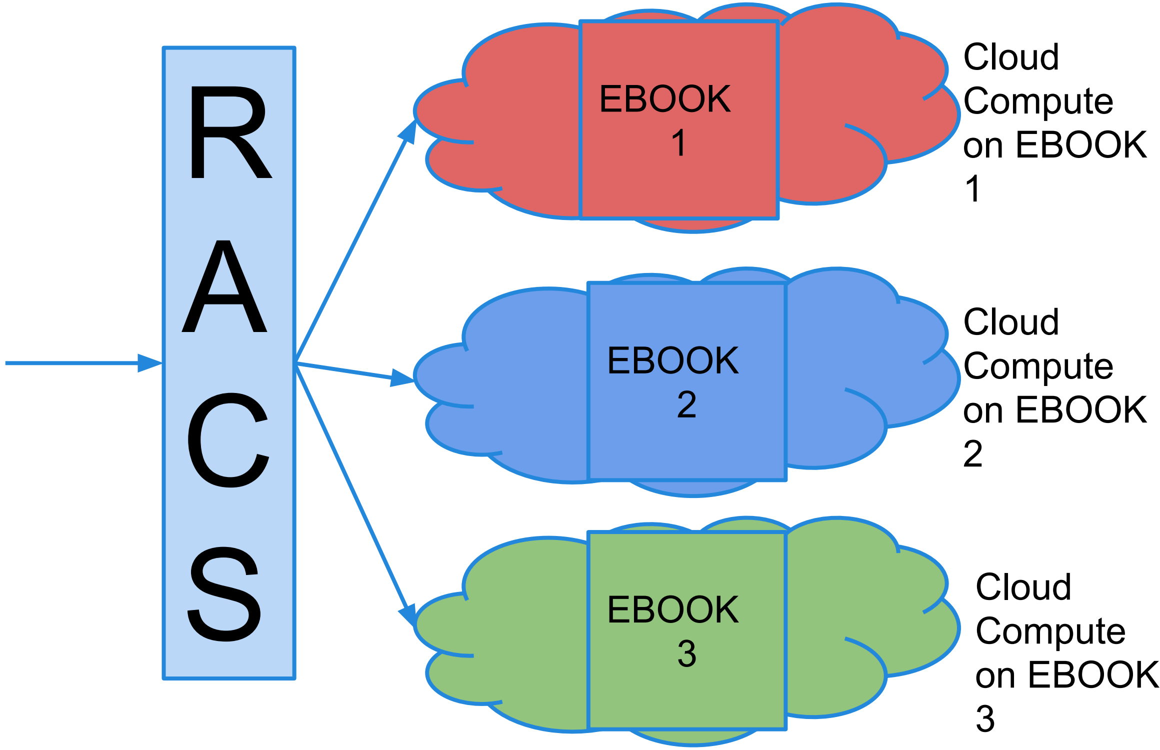


Racs-EV

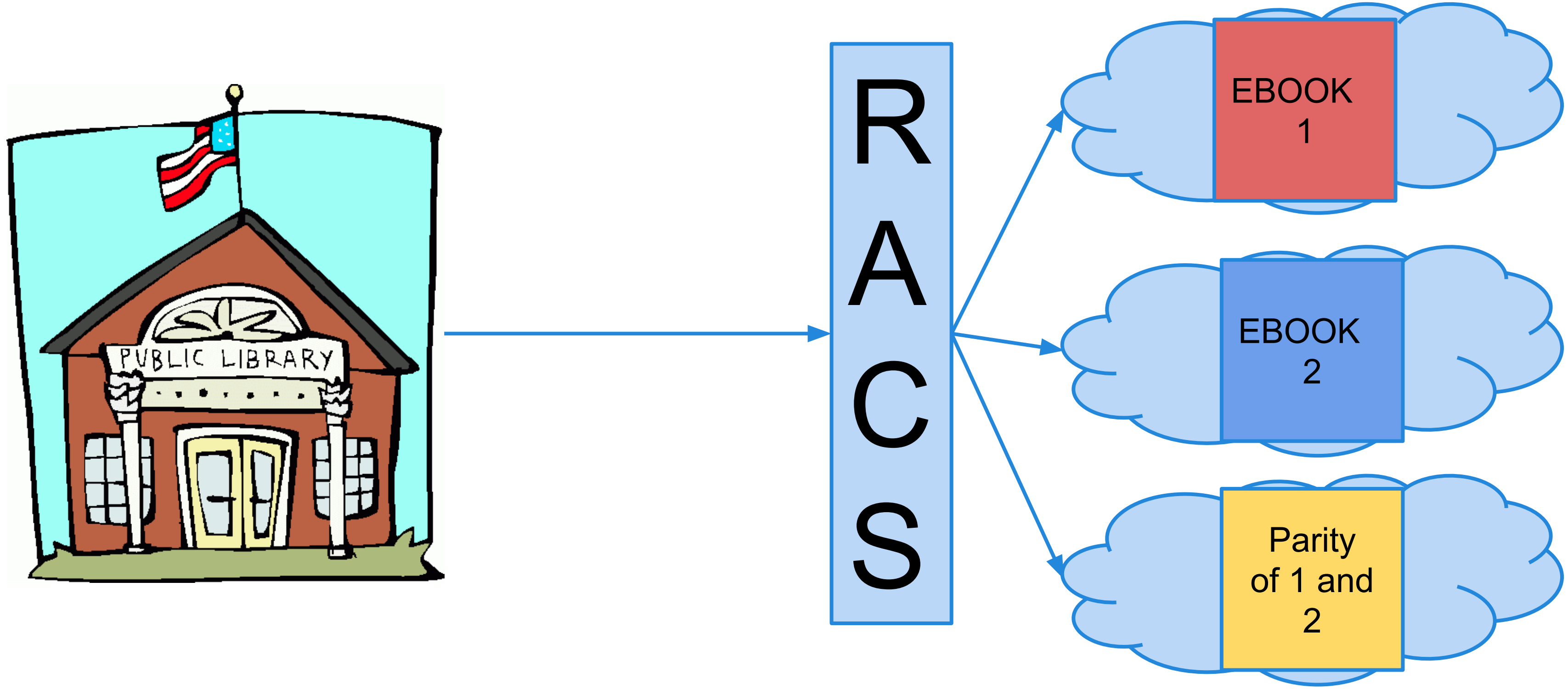


Racs-EV

Now the files are still distributed evenly, but don't need to be reassembled for cloud computation which saves on transfer fees and transfer time.



Racs-EV

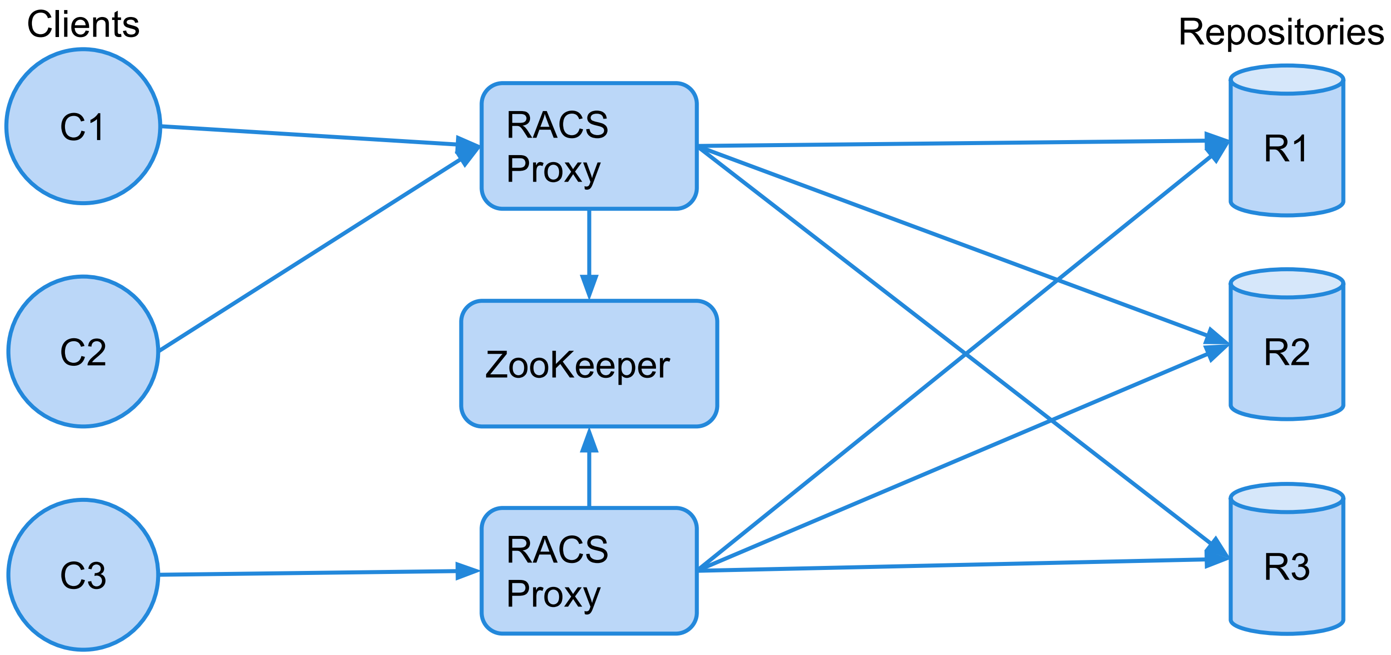


Current Prices Example (625TB)

	Storage	Transfer out (all data)	Storage(625)-Storage(500)
Microsoft	$184320 per year	$491520	$36864 per year
Amazon	$222720 per year	$491520	$44544 per year
Google	$199680 per year	$491520	$39936 per year

Costs are for RACS with 5 providers and the parity file turned on.
Transfer out doesn't include extra 125 TB since parities aren't transfer.

Overview - Multiple Proxies



ZooKeeper is a distributed coordination system. For RACS, it is used to get locks and reliably store meta-data.

RACS-EV API

PUT (Bucket, Key, Data)

GET (Bucket, Key)

PUTAT (Bucket, Key, Data, Repo)

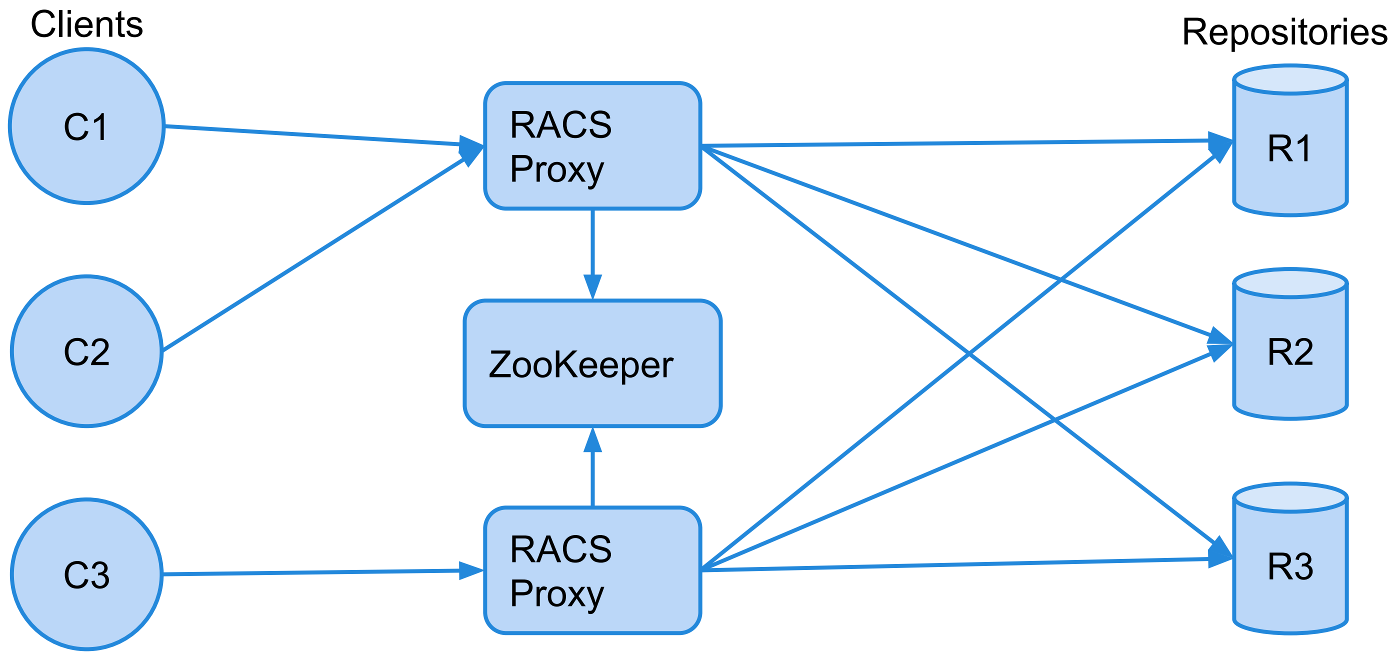
LOCATE(Bucket, Key)

DELETE (Bucket, Key)

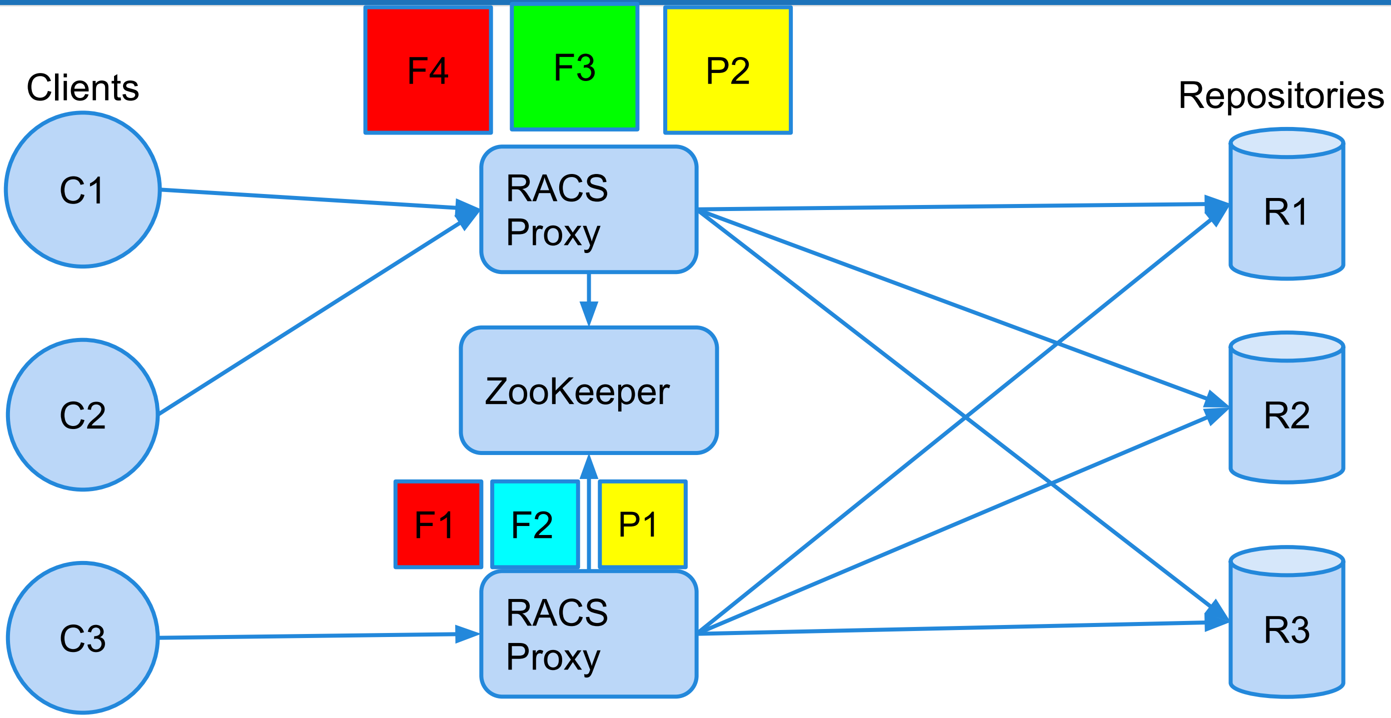
PUTS(Buckets...,Keys…,Datas…)

GETS(Buckets…,Keys…)

Simplified Put

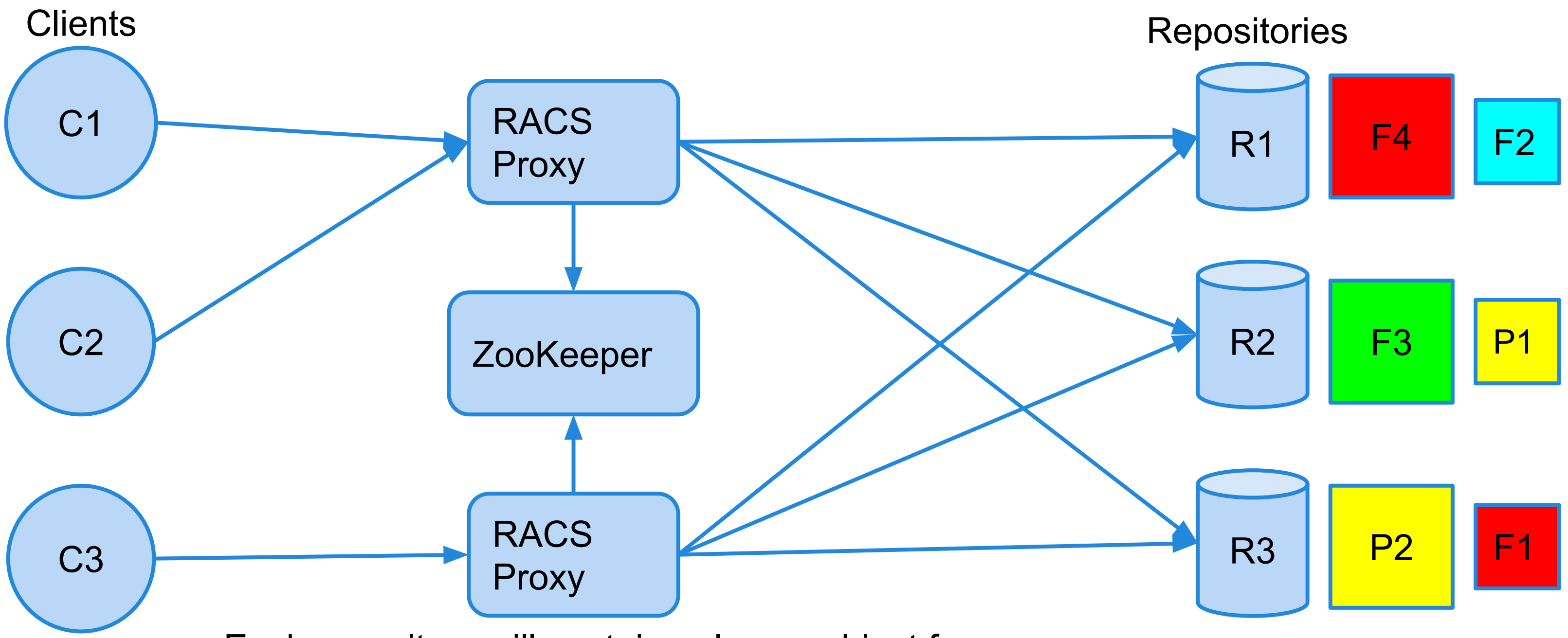


Simplified Put



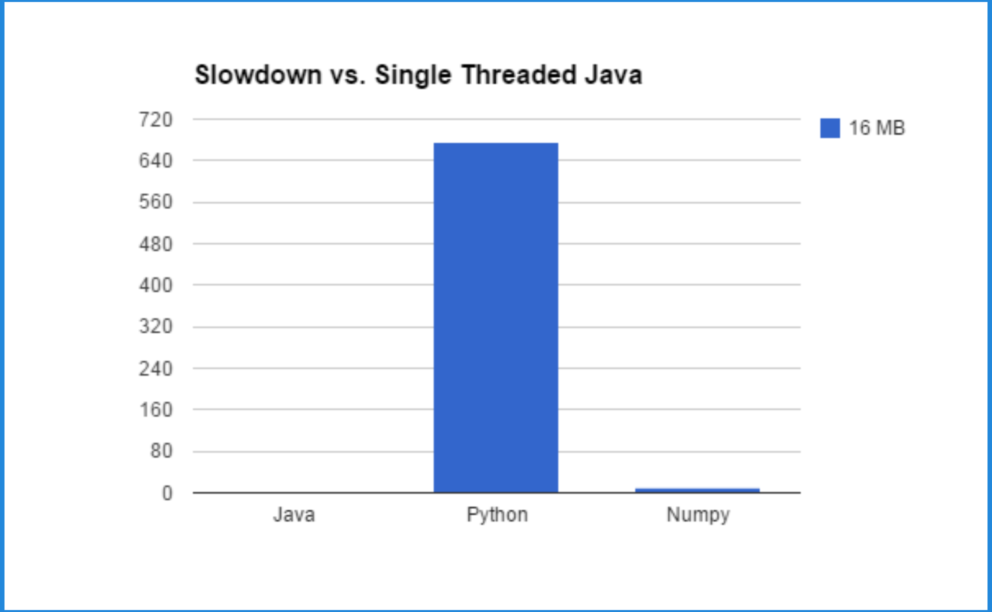
Files are grouped by size (in object groups) to reduce overhead of parity object. Group size is the same as the number of repositories.

Simplified Put



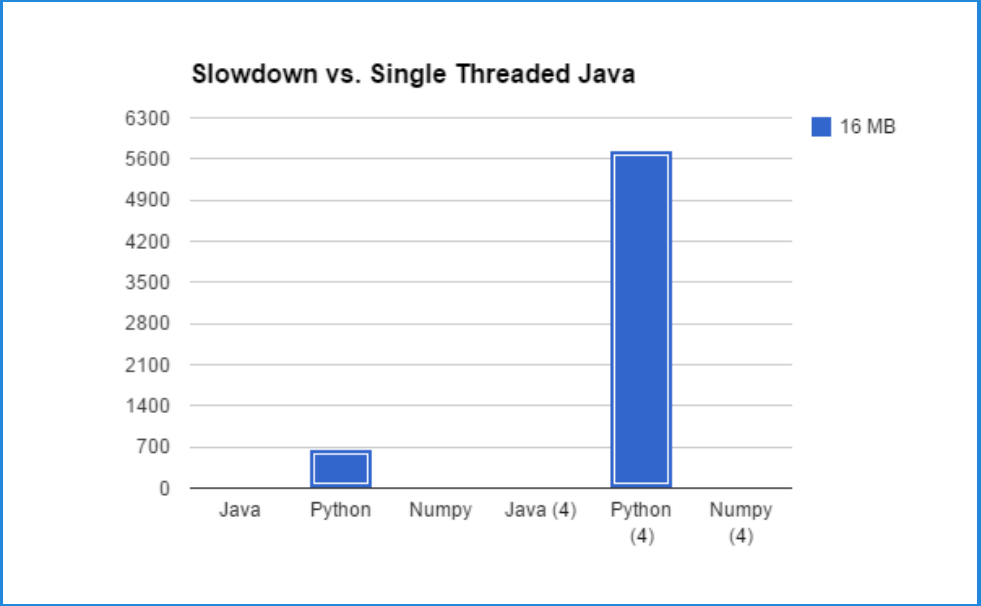
Each repository will contain only one object from each group to allow for fault tolerance.

Problems with RACS(-EV)

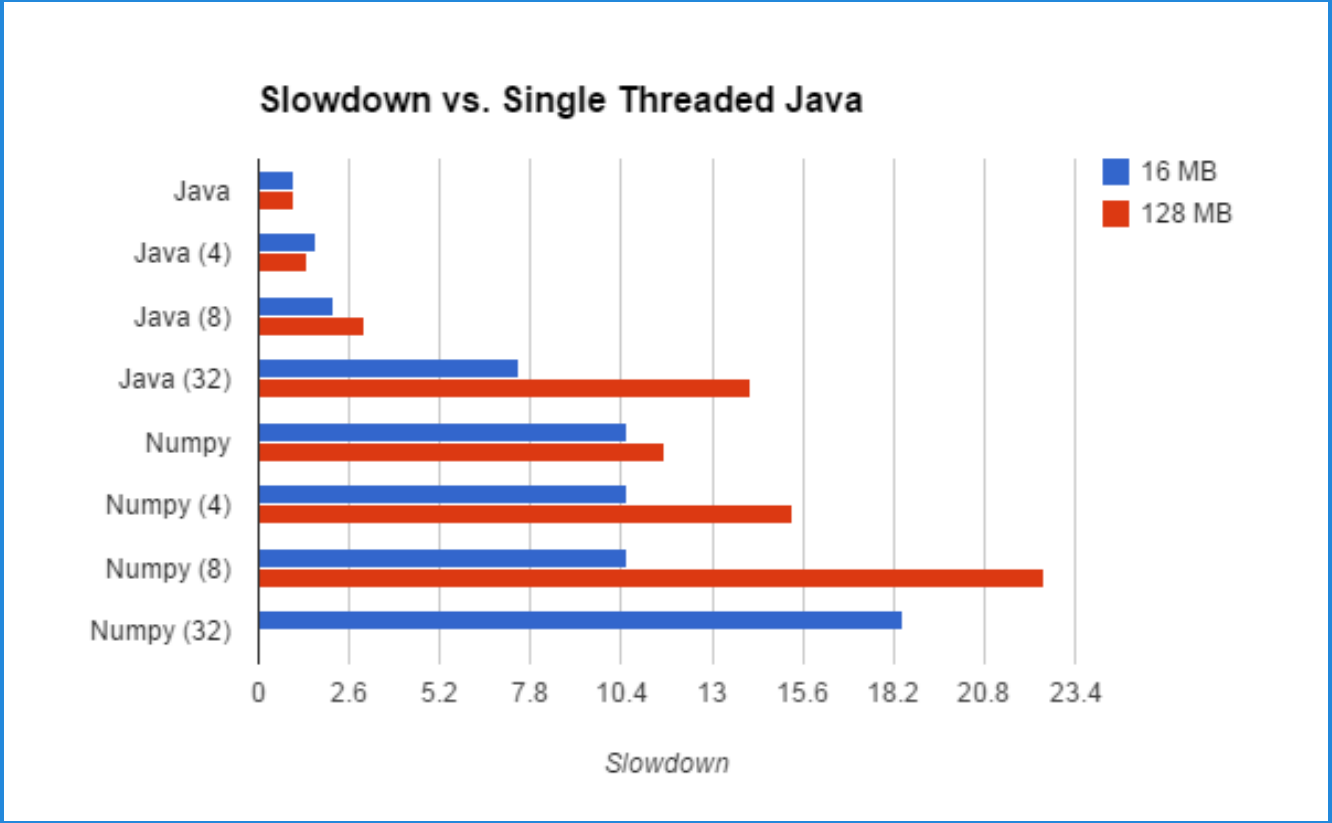


String Xor for 16 MB string

Problems with RACS(-EV)



String Xor for 16 MB string
(x) denotes number of threads

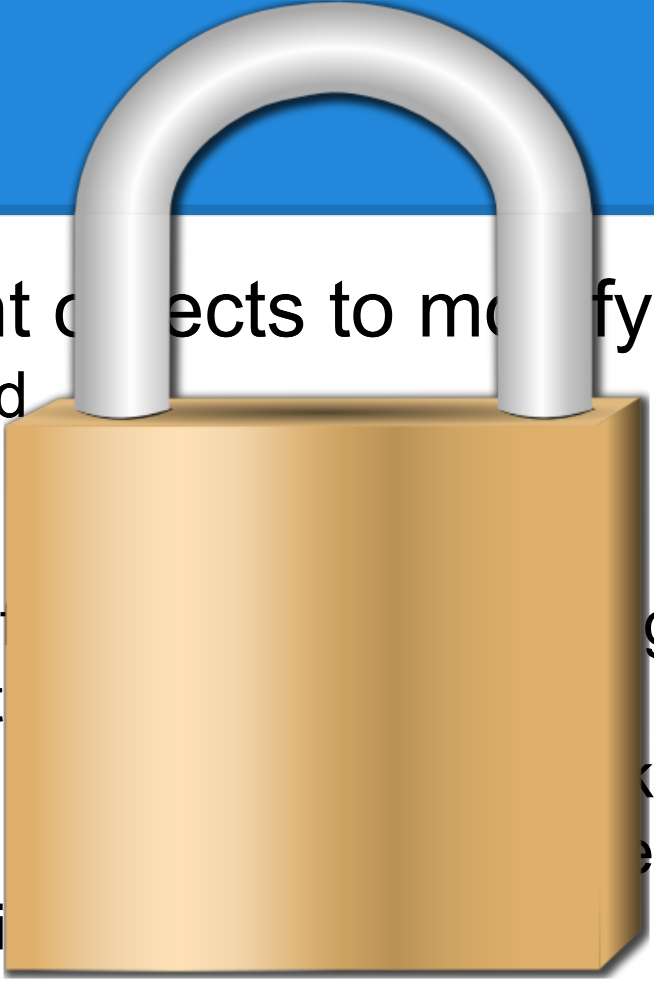


String Xor for 16 MB and 128 MB strings on 4 core 2 threads machine.
Regular Python crashed with memory problems (Numpy did too on 32 cores)

Challenges with RACS-EV

- Eight different objects to modify on put
  - Data on cloud
  - Objectgroup
  - Parity File
  - Objectgroup freelist (keeps track of groups with space)
  - Key to object group mapping
  - Previous key objectgroup (remove key from group)
  - Previous key data on cloud (remove it)
  - Previous parity file

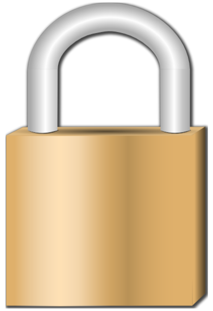
Solution



- Eight different o ects to mo fy on put
    - Data on cloud
    - Objectgroup
    - Parity File
    - Objectgroup groups with space)
    - Key to object
    - Previous key ey from group)
    - Previous key e it)
    - Previous pari

Solution (or not)

- Locking alone doesn’t solve the problems
  - Could lose connection at any point
  - The lock could be lost at any point.
  - if(lock.isAcquired()) then modifyData() isn’t atomic
  - Similar problems to updating hard drive
    - Things have to be done in a particular order



Solutions with RACS-EV

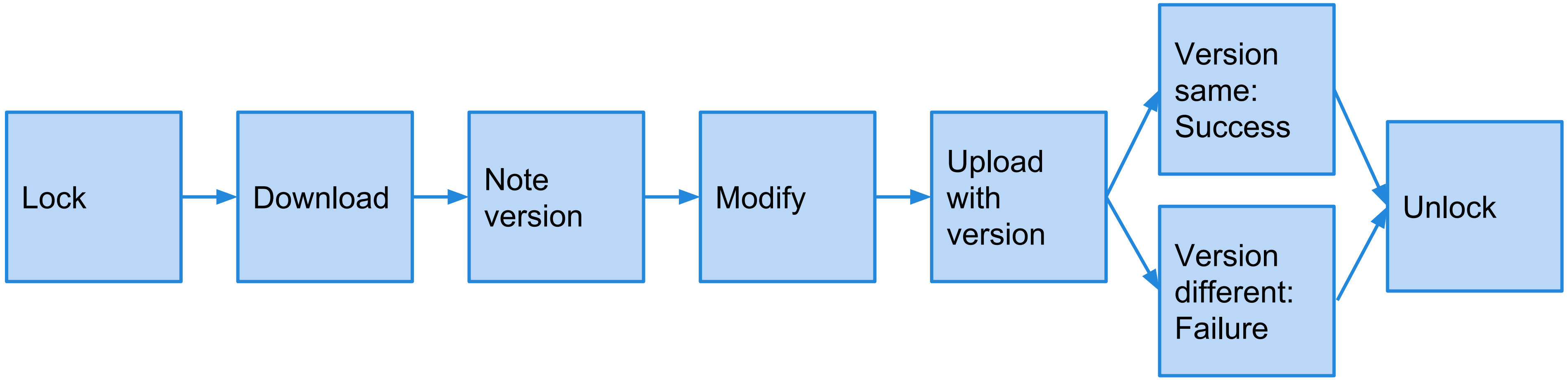
1. Turns out this order is pretty good
   a. Data on cloud
   b. Objectgroup
   c. Parity File
   d. Objectgroup Freelist (keeps track of groups with space)
   e. Key to object group mapping
   f. Previous key objectgroup (remove key from group)
   g. Previous key data on cloud (remove it)
   h. Previous parity file

Solutions with RACS-EV

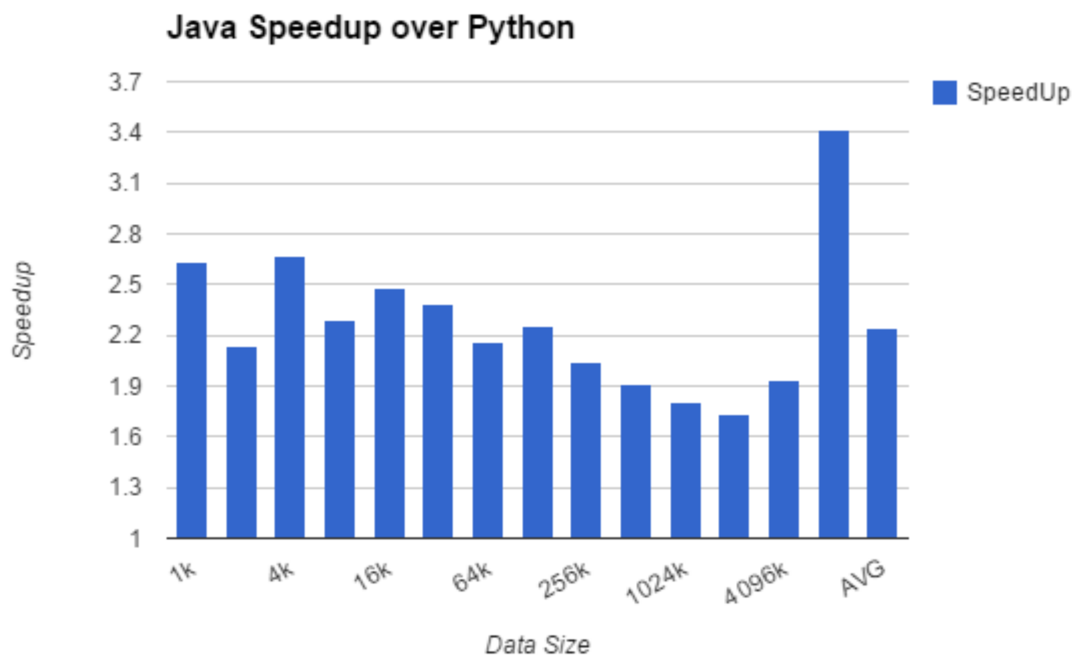
1. Turns out this order is pretty good
        - Register Intent
    b. Data on cloud
    c. Objectgroup
    d. Parity File
    e. Objectgroup Freelist (keeps track of groups with space)
    f. Key to object group mapping
        - (Deregister Intent, Register Intent to delete, and part f) atomically
    g. Previous key objectgroup (remove key from group)
    h. Previous key data on cloud (remove it)
    i. Previous parity file

if(lock.isAcquired()) then ...

- This isn't atomic.
    - Requires versioning

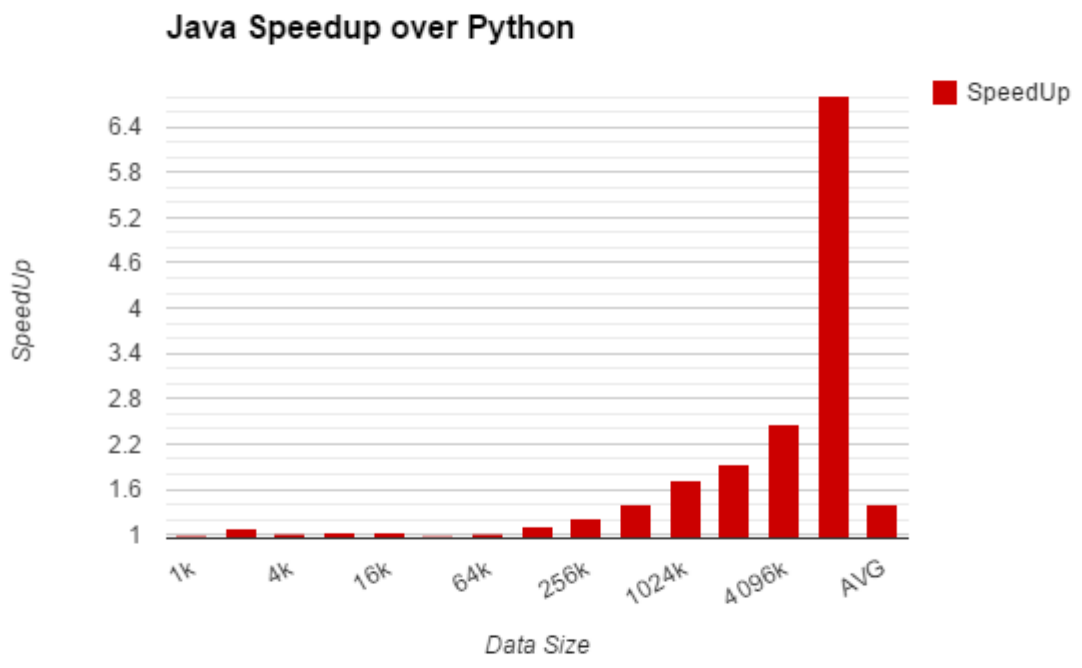


The Data



- 3 Repos
- 3 EC2 Instances
  - 2 cores
  - 2 threads/core
  - 1.7 GHz
  - 8 GB ram
- Clients:
  - 9 Clients
  - 20 Files of each size per client
  - Clients send then wait for response

The Data Part 2



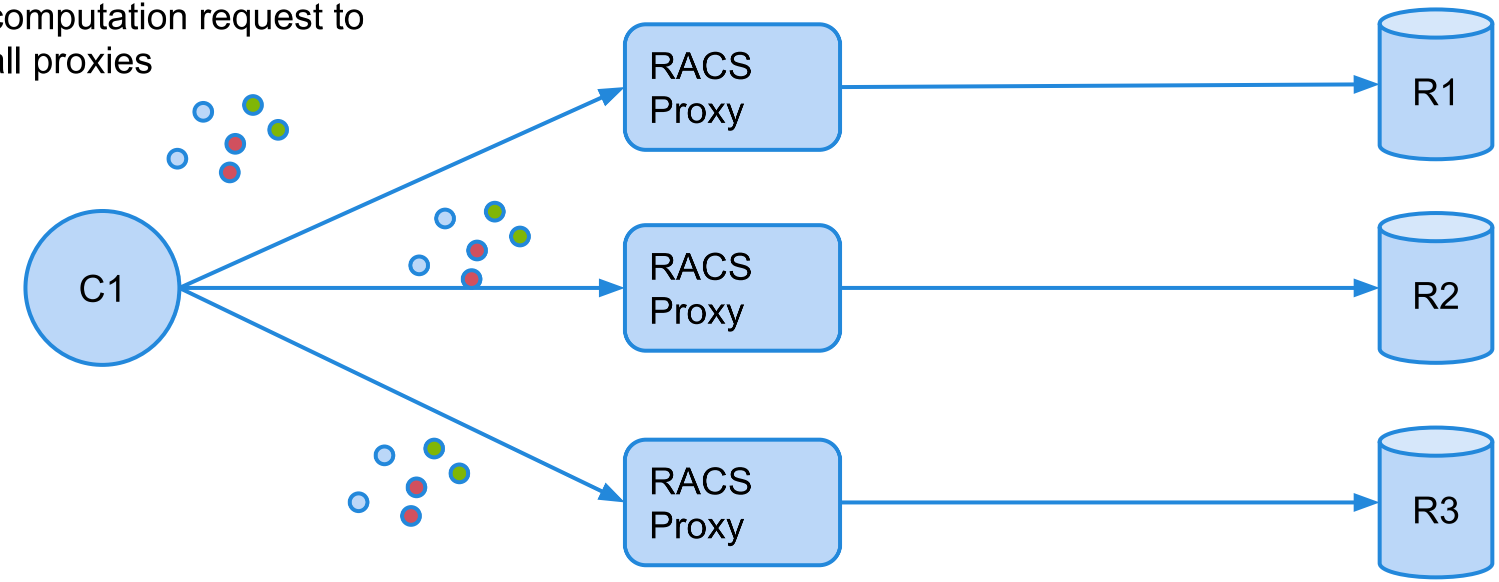
- 3 Repos
- 3 EC2 Instances
  - 2 cores
  - 2 threads/core
  - 1.7 GHz
  - 8 GB ram
- Clients:
  - 18 Clients
  - 20 Files of each size per client
  - Clients send then wait for response

Future Plans

- Always room for optimization to faster
- Cloud computation

Future Plans - Cloud Computation

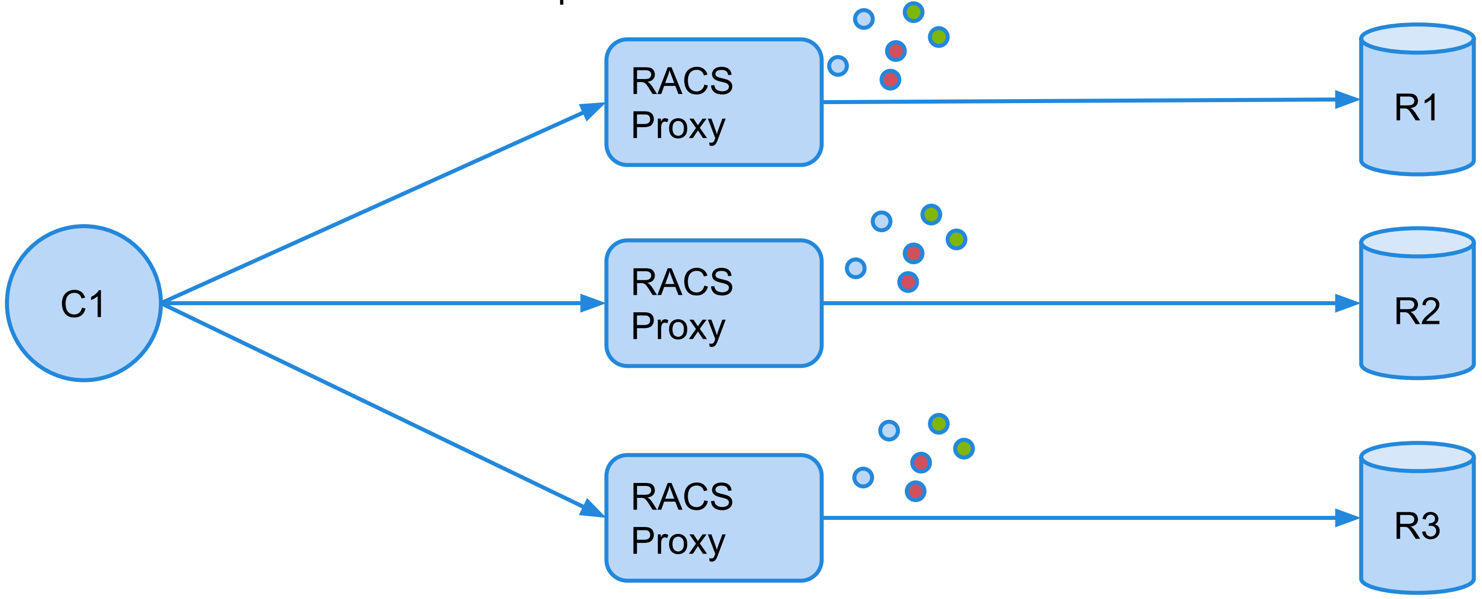
Send same computation request to all proxies



Zookeeper not shown; Connections from each R to each RACS not shown

Future Plans - Cloud Computation

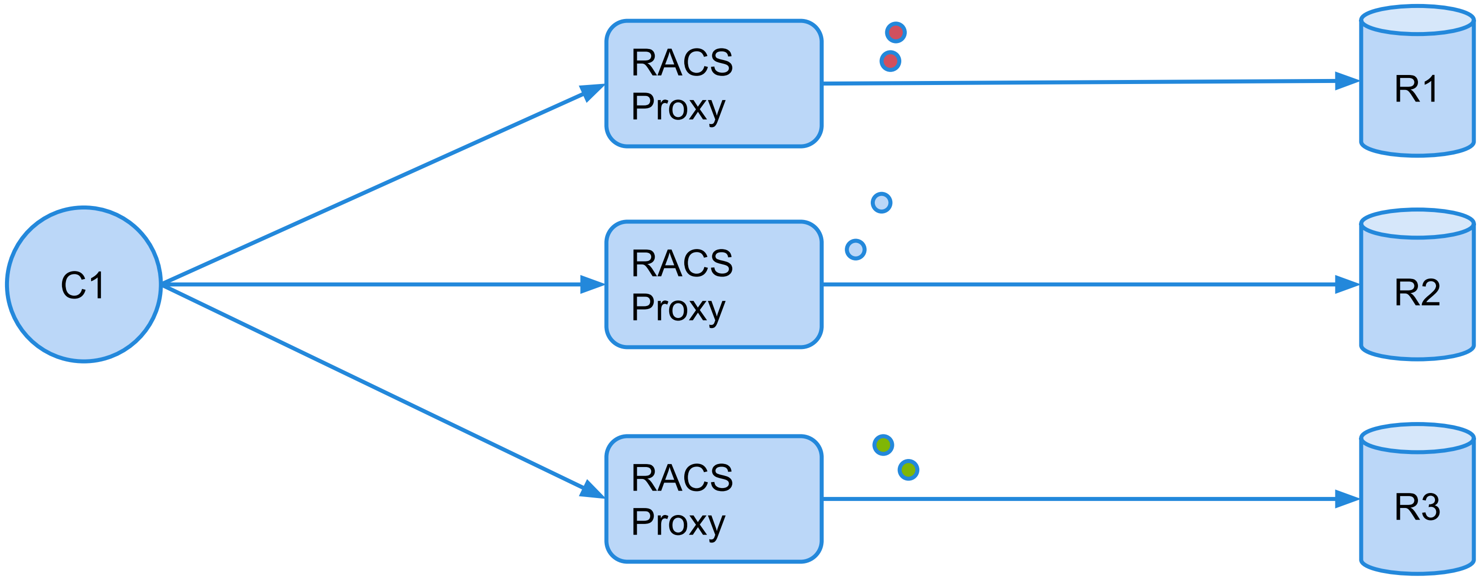
Proxy will see which requests belong to ‘its’ repo



Zookeeper not shown; Connections from each R to each RACS not shown

Future Plans - Cloud Computation

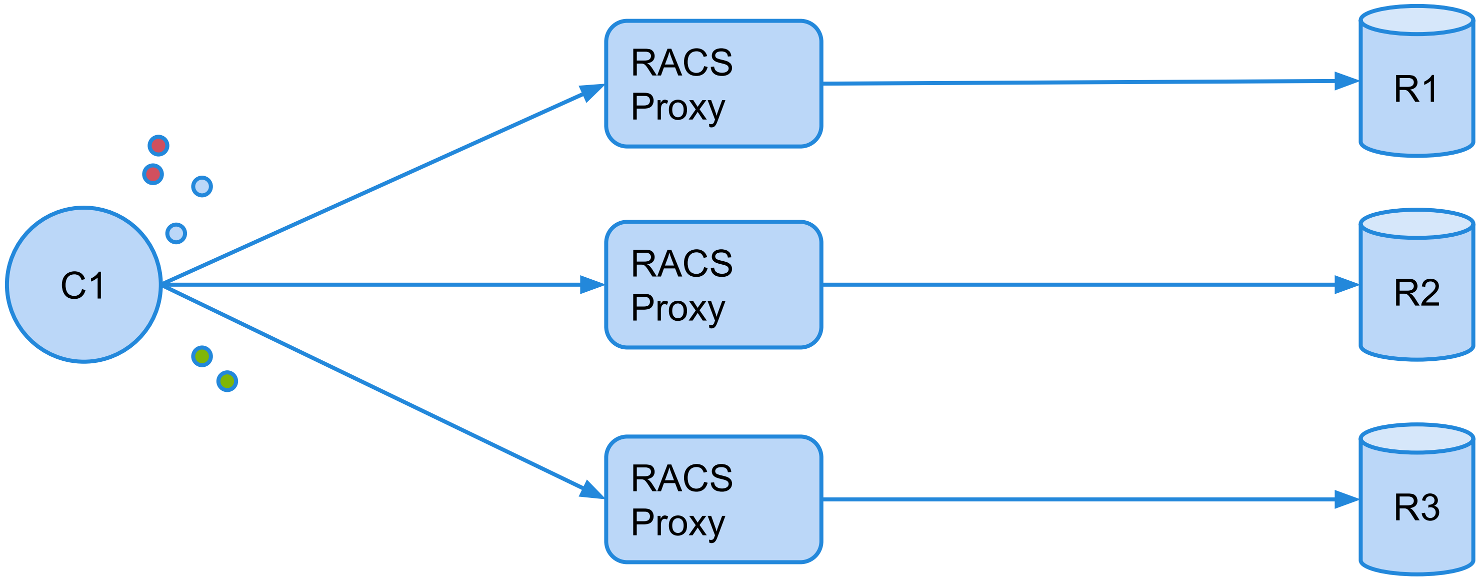
Proxy will see which requests belong to ‘its’ repo



Zookeeper not shown; Connections from each R to each RACS not shown

Future Plans - Cloud Computation

Client will receive back the results



Zookeeper not shown; Connections from each R to each RACS not shown

Demo/Questions