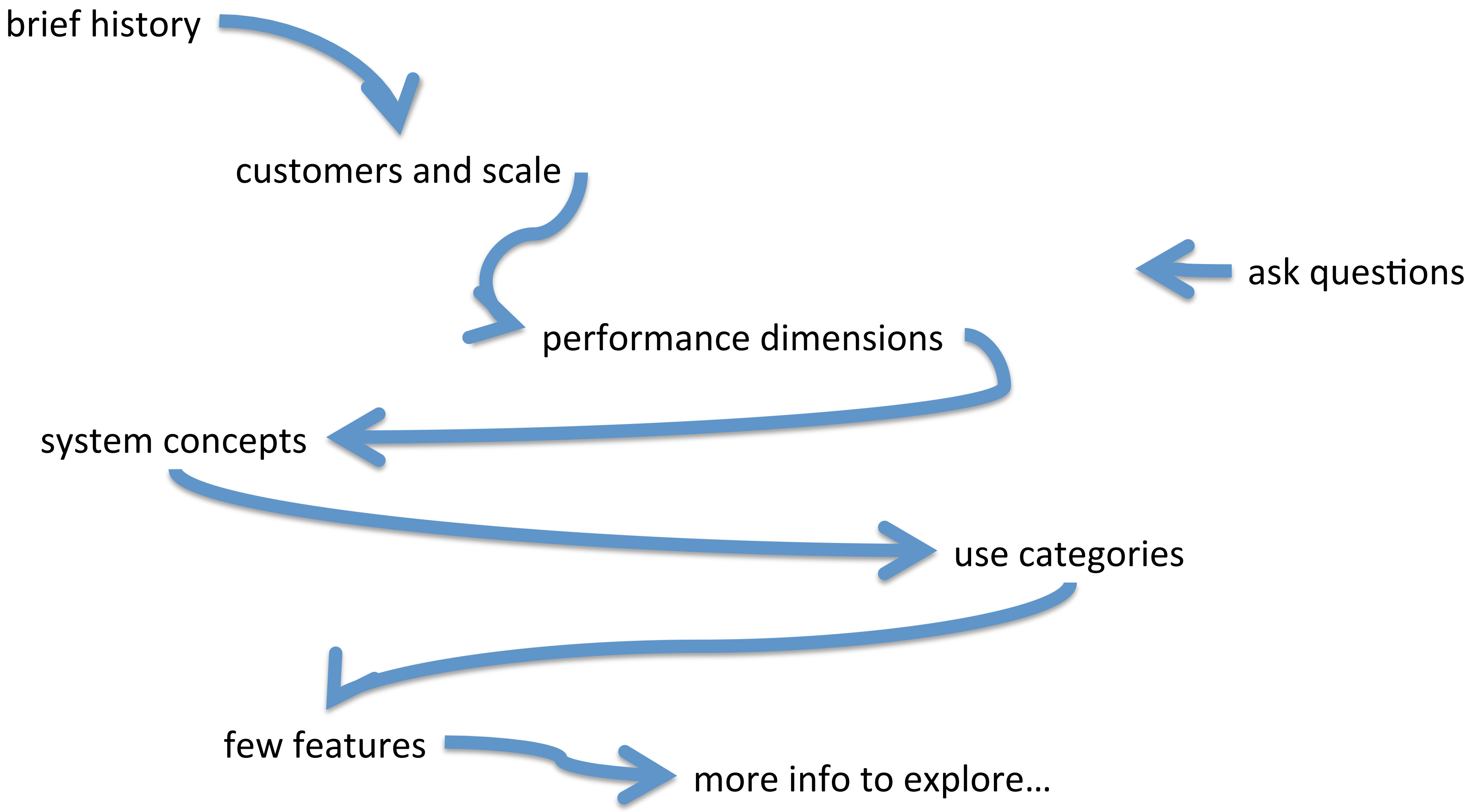
yql techtalk

data processing at scale

20120927 – mirek@yahoo-inc.com

the plan


brief history
customers and scale
performance dimensions
ask questions
system concepts
use categories
few features
more info to explore…

brief history

yql started as an upgrade to pipes (developers asked for cmd version of pipes)

evolved into “select * from internet” – cloud data serving

launched October 2008

added insert update delete – grew its vocabulary

<execute> added

developer console

…lots of features

served 5B queries September 2010

upgraded runtime architecture to scale

introduced tenancy into serving

over 1200 open tables

served 20B queries September 2011

over million of pipes

hundreds of yahoo! tables added

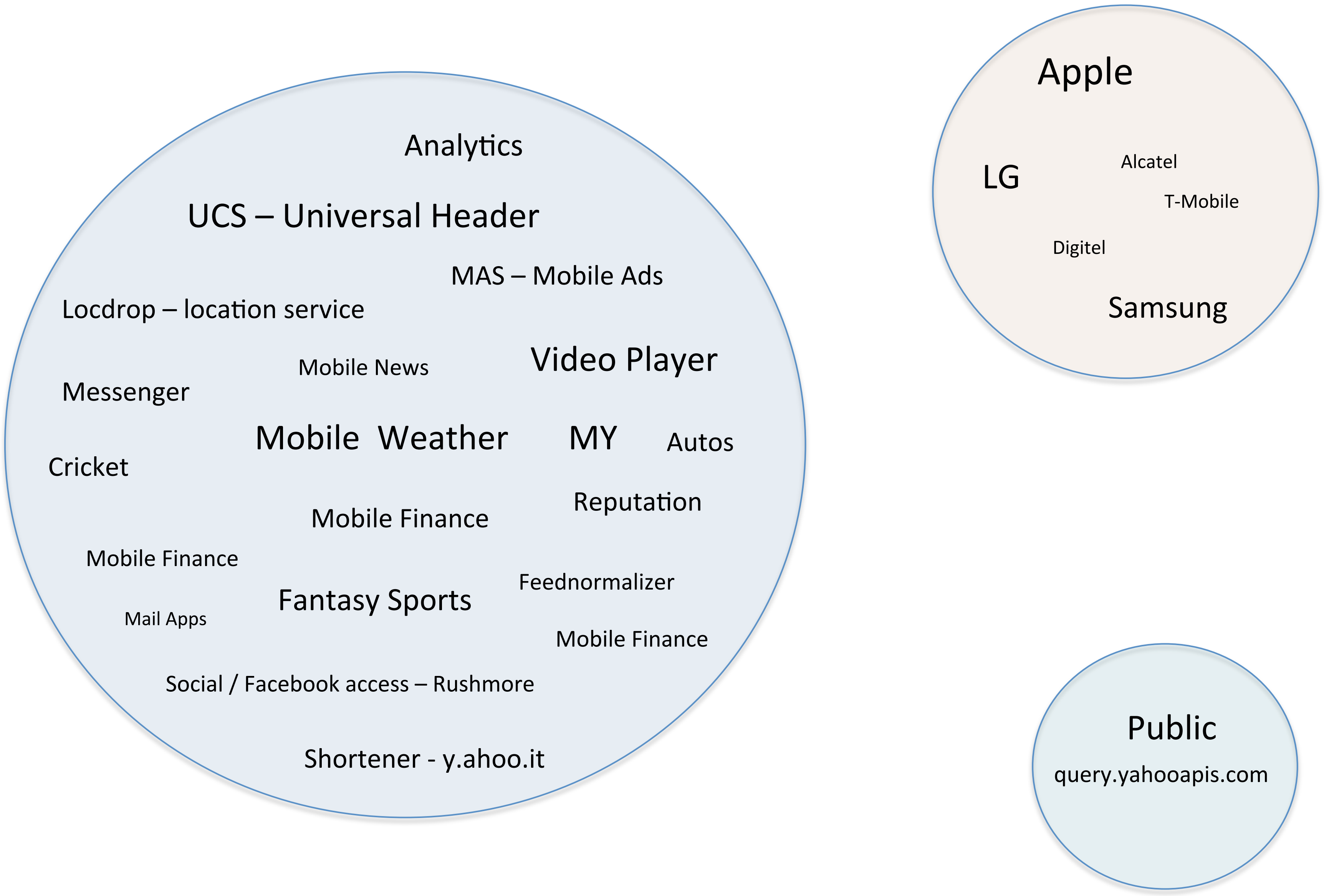
over hundred of on-boarded customers

today

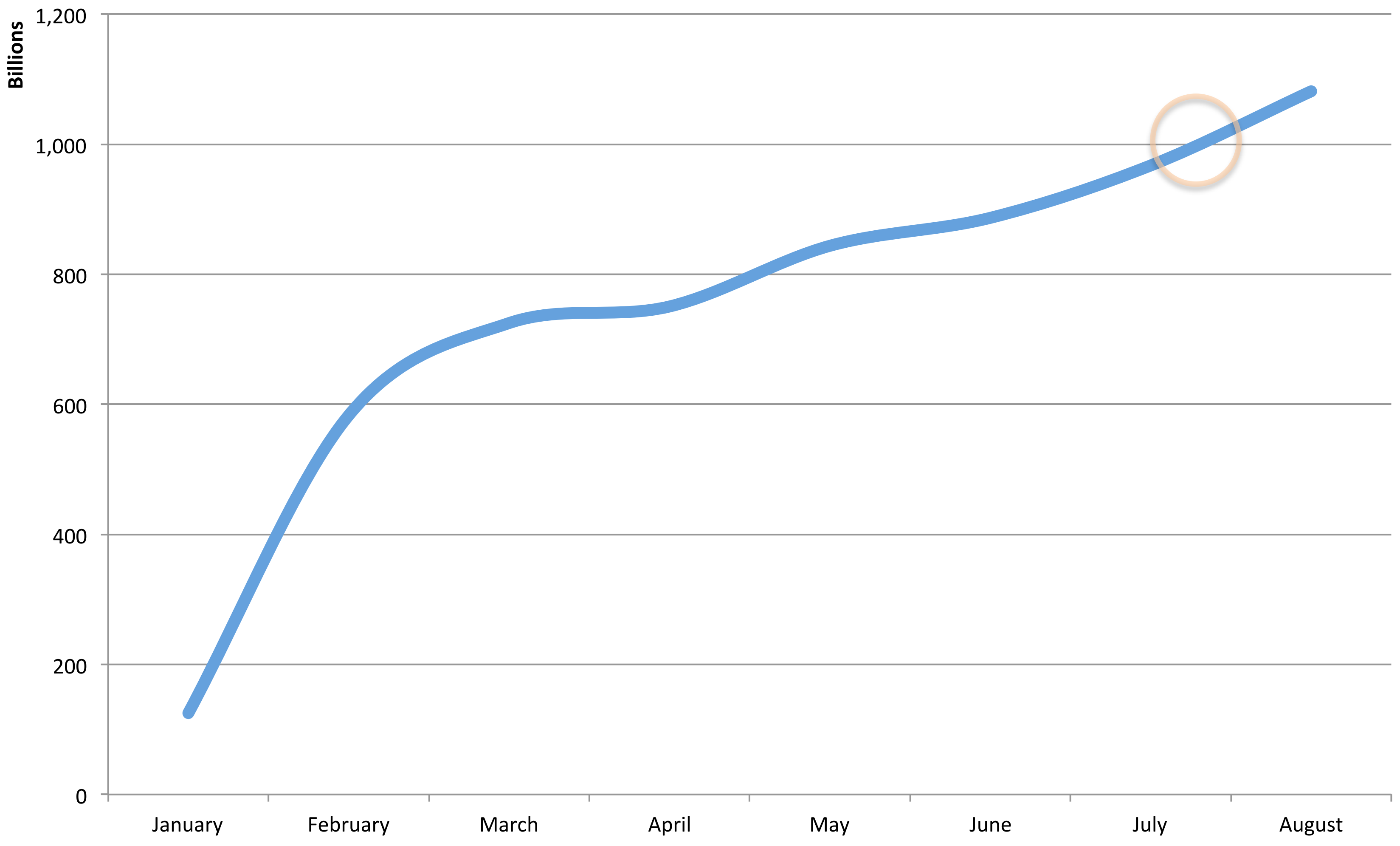
the mission: make data discovery, enrichment and serving fast and easy

customers

three categories of customers

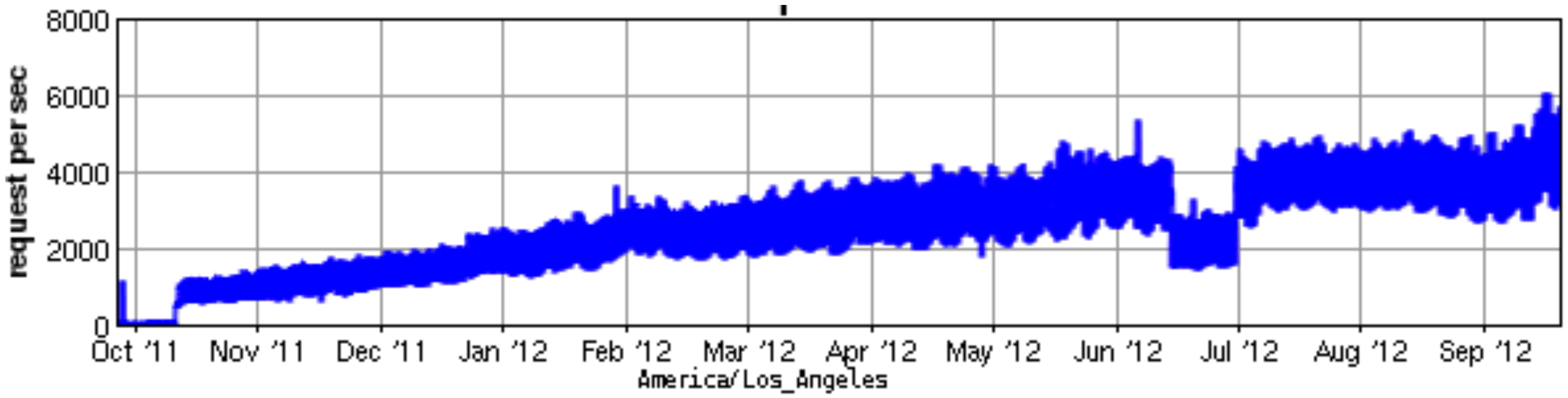


yql query volume in 2012



Apple

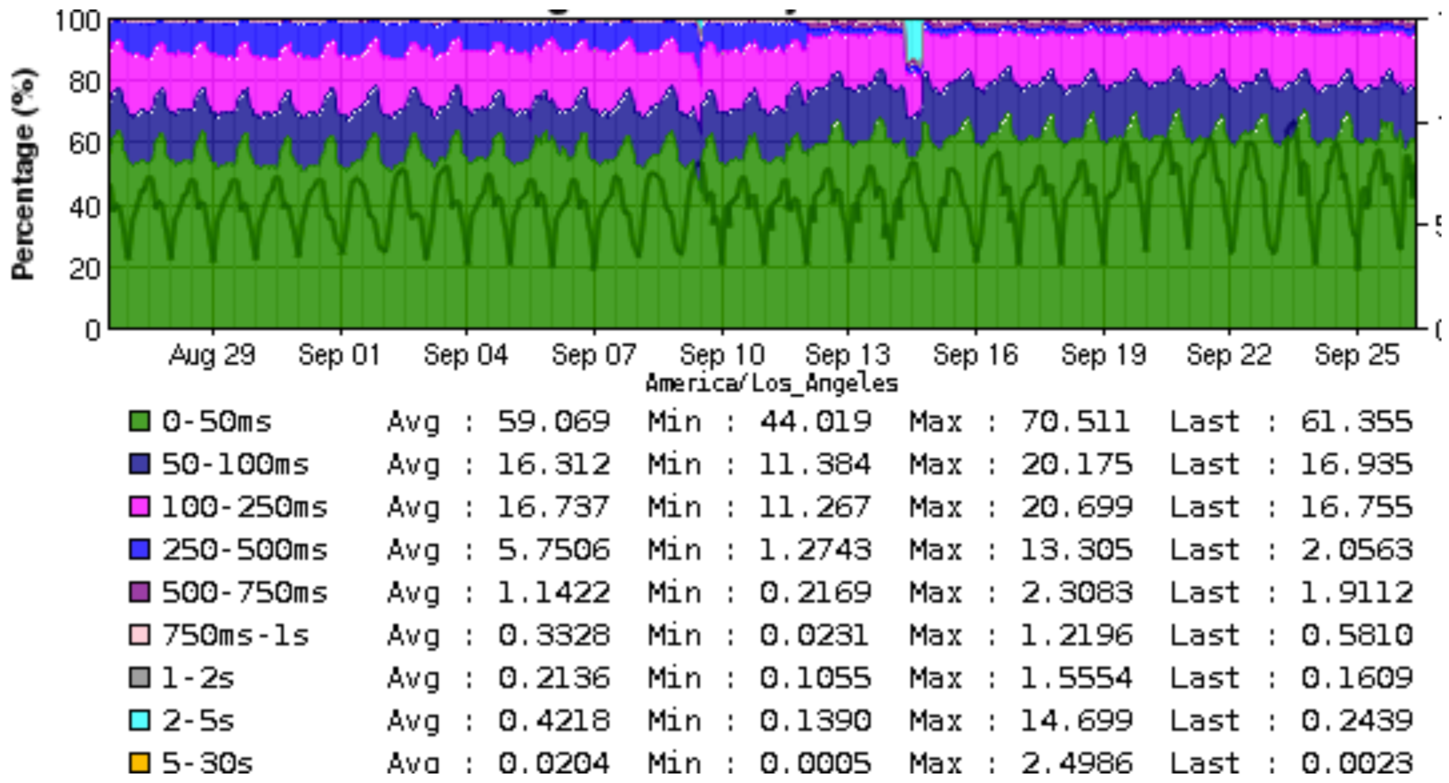
major partner – grows with iOS adoption



predominantly weather / siri

heavily utilizes yql.multi

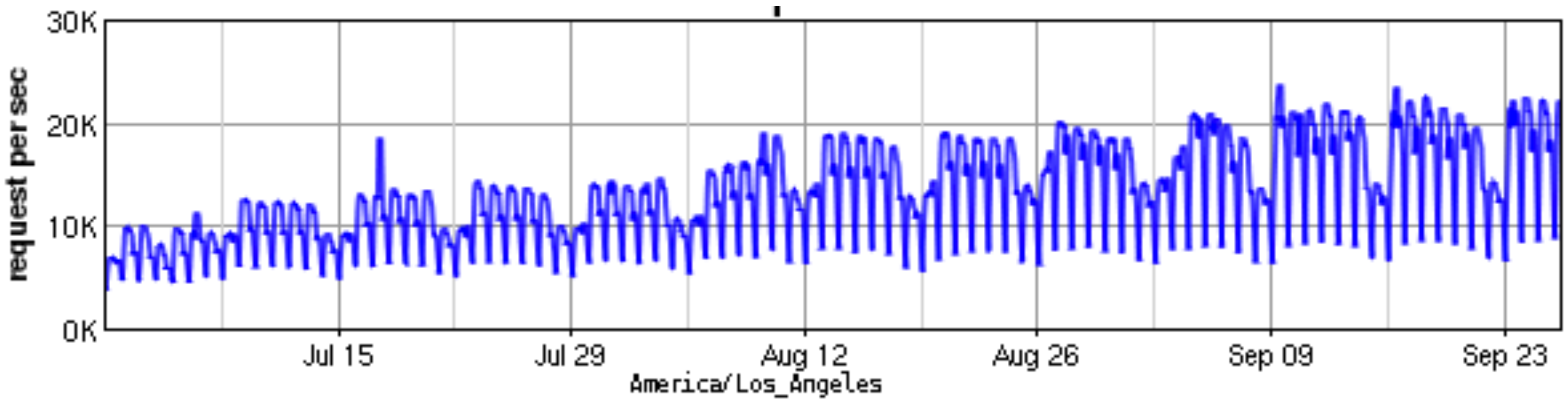
Apple



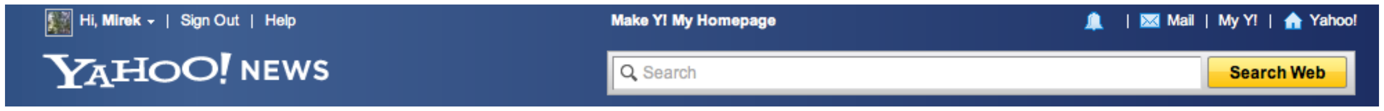
```
🟩 0-50ms       Avg : 59.069  Min : 44.019  Max : 70.511  Last : 61.355
🟦 50-100ms     Avg : 16.312  Min : 11.384  Max : 20.175  Last : 16.935
🟪 100-250ms    Avg : 16.737  Min : 11.267  Max : 20.699  Last : 16.755
🟦 250-500ms    Avg : 5.7506  Min : 1.2743  Max : 13.305  Last : 2.0563
🟪 500-750ms    Avg : 1.1422  Min : 0.2169  Max : 2.3083  Last : 1.9112
⬜ 750ms-1s     Avg : 0.3328  Min : 0.0231  Max : 1.2196  Last : 0.5810
⬜ 1-2s         Avg : 0.2136  Min : 0.1055  Max : 1.5554  Last : 0.1609
🟦 2-5s         Avg : 0.4218  Min : 0.1390  Max : 14.699  Last : 0.2439
🟨 5-30s        Avg : 0.0204  Min : 0.0005  Max : 2.4986  Last : 0.0023
```

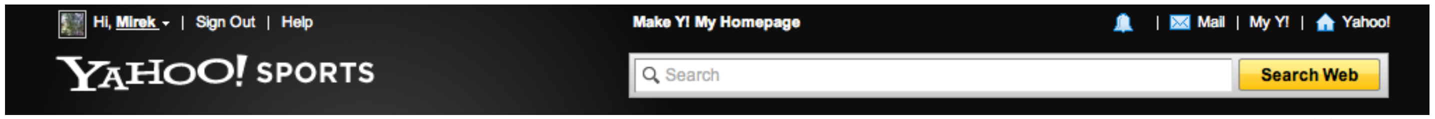
places performance demands on the platform

UCS – Universal Header

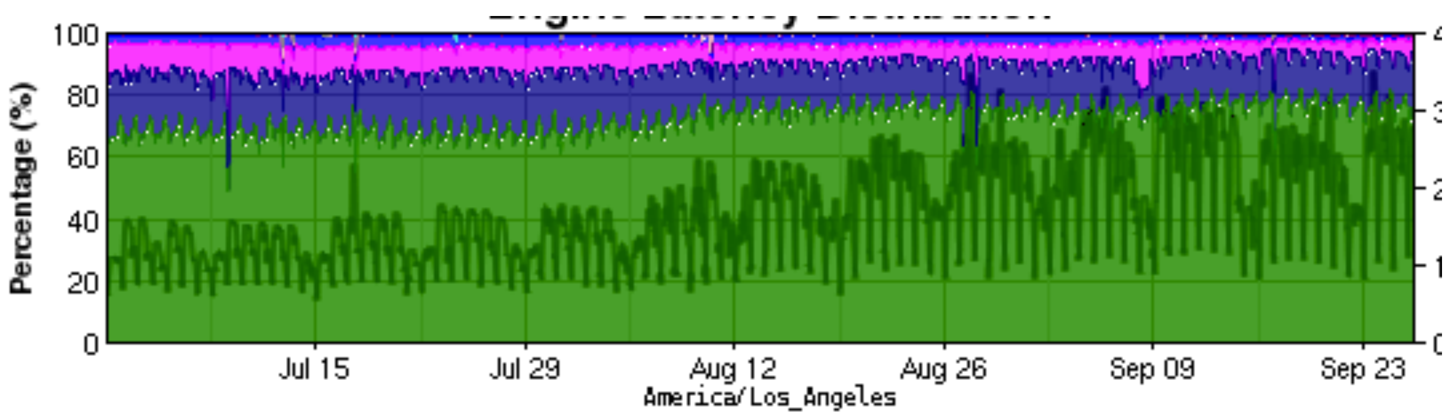


part of nearly every yahoo page





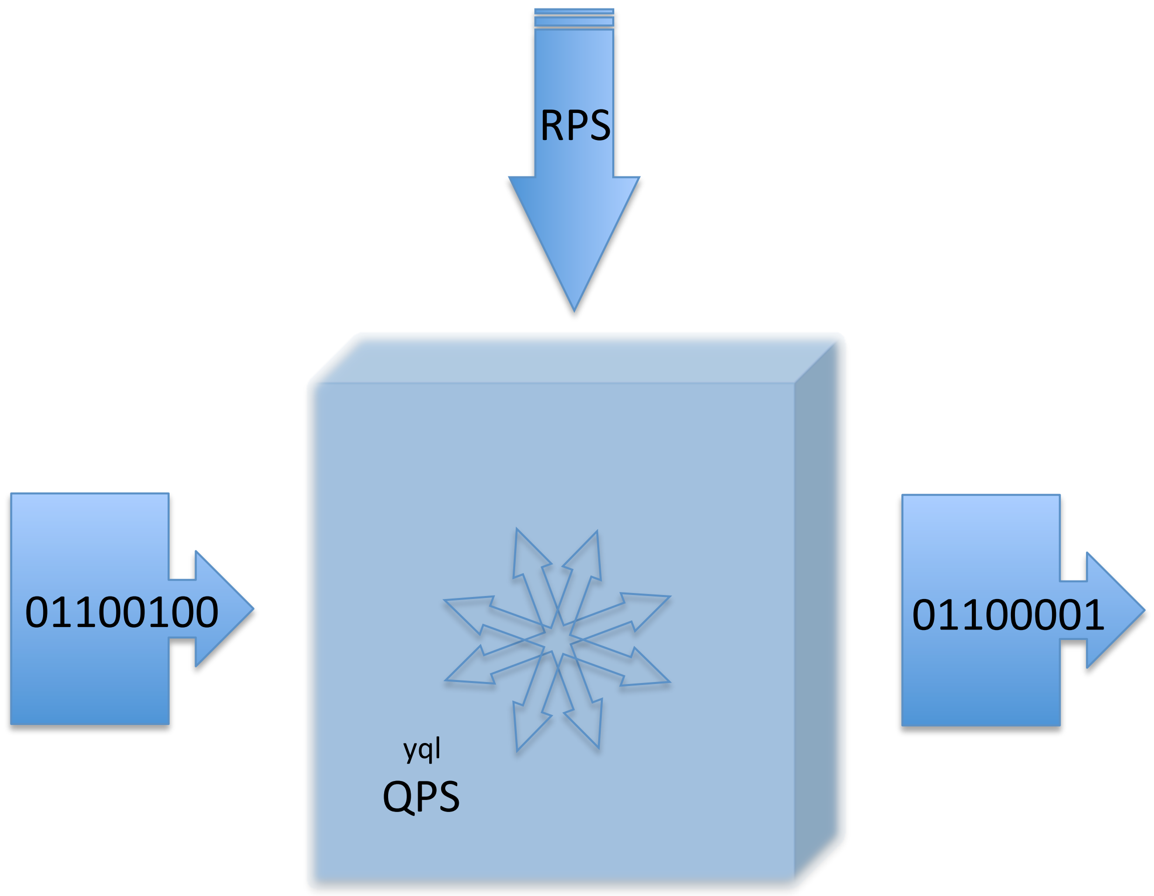
UCS – Universal Header

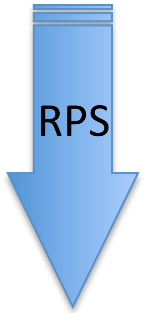


```
🟩 0-50ms      Avg : 72.190   Min : 49.222   Max : 82.203   Last : 76.998
🟦 50-100ms    Avg : 16.624   Min : 6.4365   Max : 24.643   Last : 16.827
🟪 100-250ms   Avg : 7.4759   Min : 2.9352   Max : 38.000   Last : 4.2868
🟦 250-500ms   Avg : 2.9282   Min : 0.7770   Max : 6.6071   Last : 0.8027
🟪 500-750ms   Avg : 0.4408   Min : 0.1904   Max : 1.7318   Last : 0.6795
⬜ 750ms-1s    Avg : 0.1334   Min : 0.0484   Max : 2.9683   Last : 0.2205
⬜ 1-2s        Avg : 0.1637   Min : 0.0589   Max : 6.1371   Last : 0.1690
🟦 2-5s        Avg : 0.0385   Min : 0.0048   Max : 4.1487   Last : 0.0151
🟧 5-30s       Avg : 0.0054   Min : 0.0003   Max : 2.5409   Last : 0.0017
🟨 30-35s      Avg : 6e-06    Min : 0        Max : 0.0047   Last : 0
```

performance

lets look at platform performance as function of three dimensions





requests per second

represents inbound demand for work and product

efficient system does work fast and cheap

~35K average

~60K at peak

few* platforms at y! do this much work over HTTP

what’s in a yql query?

compile statement   gather response data

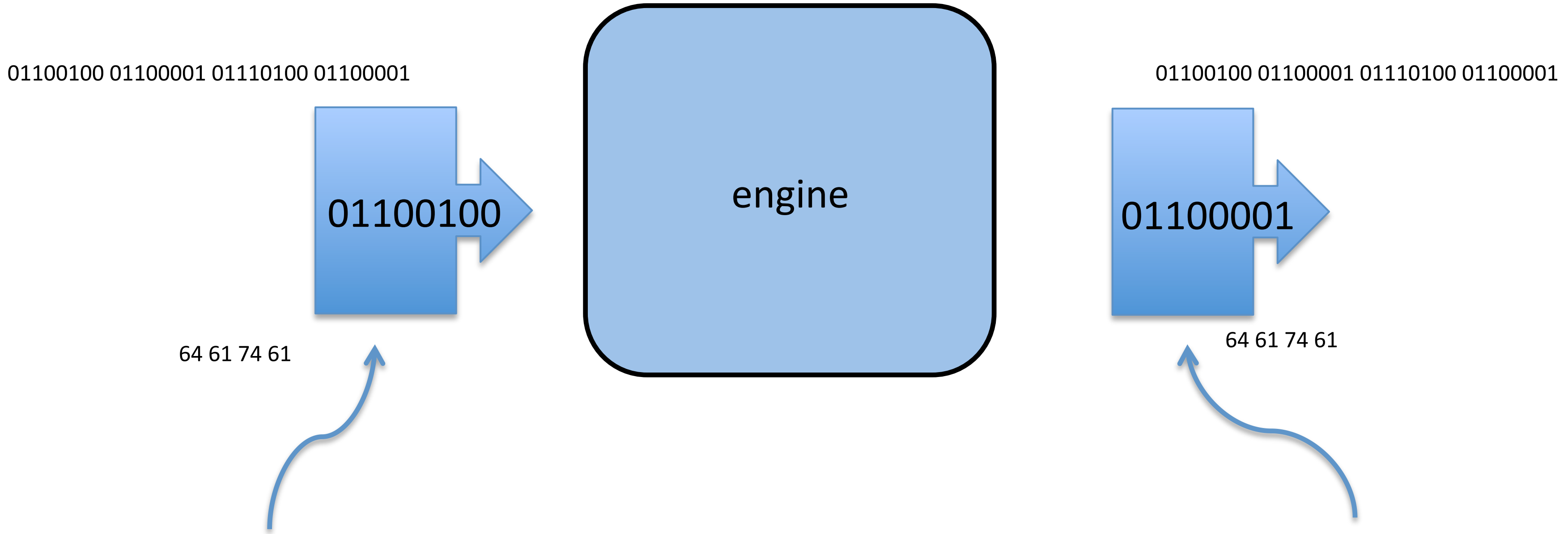
$$q = pr + cs + bep + grd + ppr$$

parse request   build execution pipeline   package product and return

over 400K / sec

yql queries represent work done by the engine

1T / month



inbound over 30TB / day

produces over 10TB / day

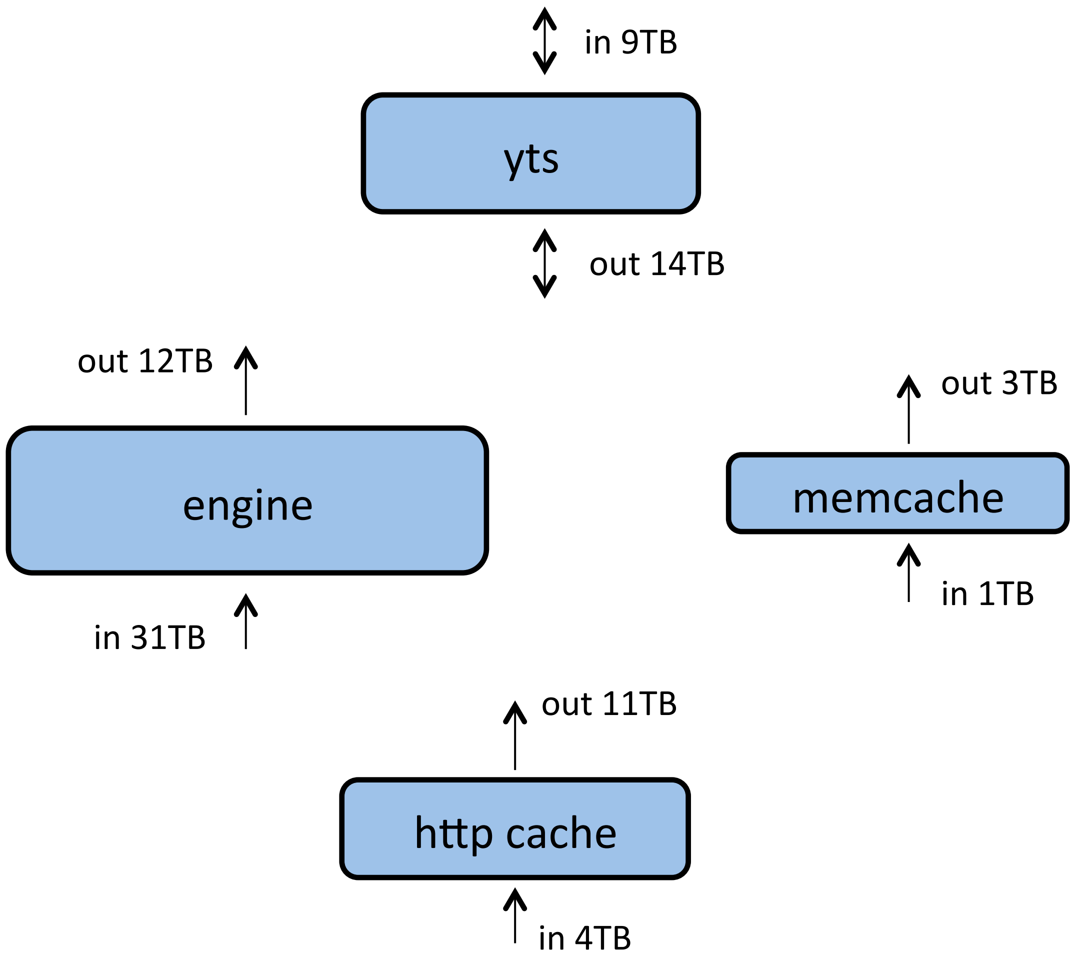
or

about 100M really long articles

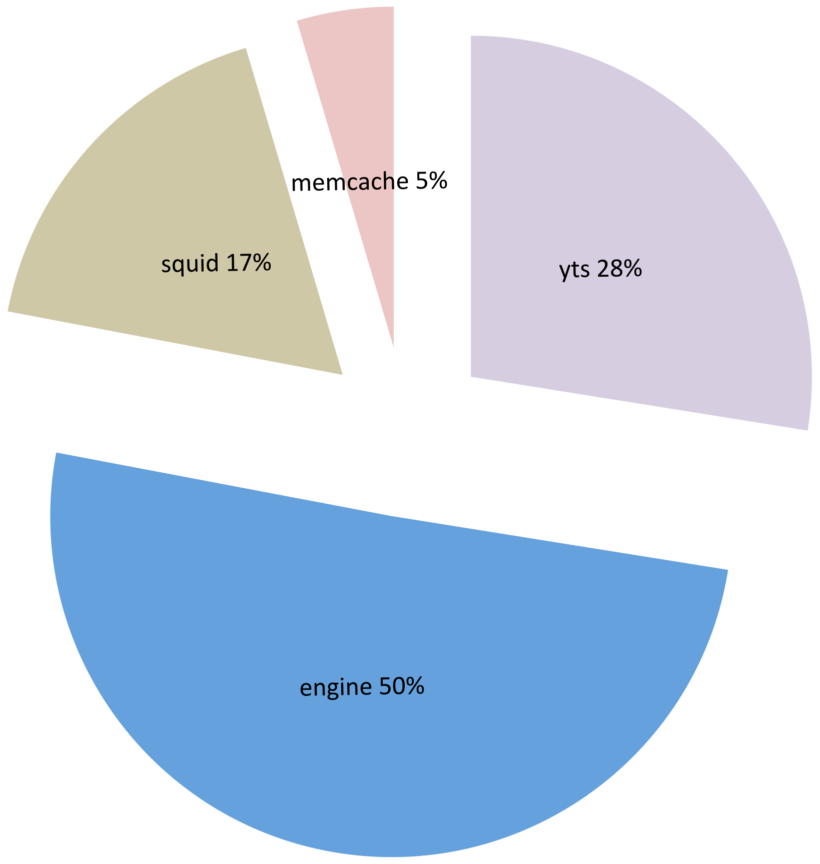
data volume processed by the engines

data flow by platform component per day

45TB in

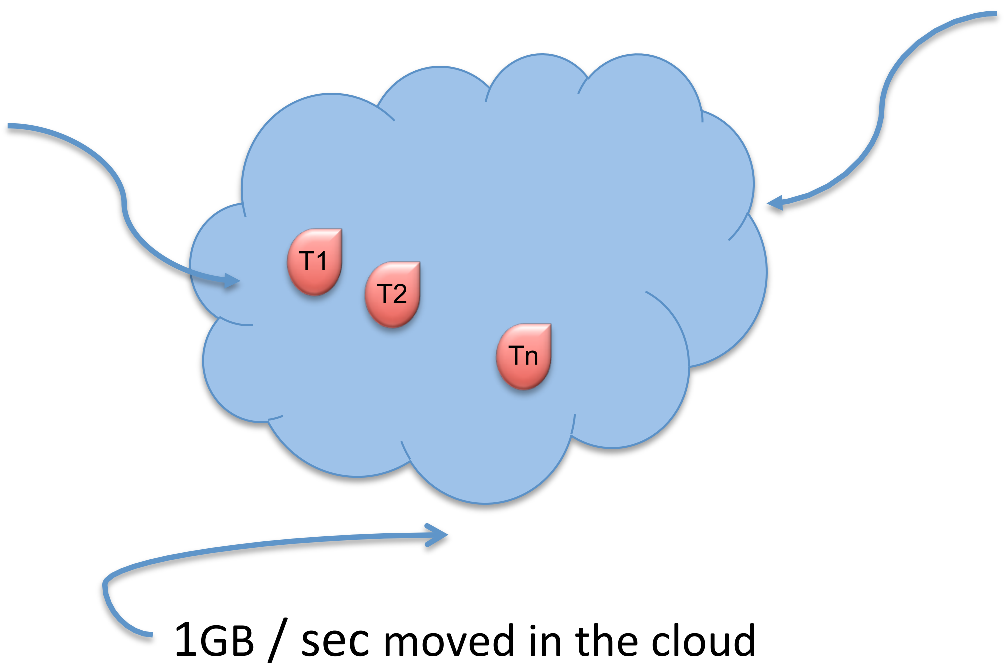


40TB out



data seen by the platform component

yql platform sees over 2.5 PT in a month



30B+ queries/day

3B requests/day

50% < 50ms
80% < 250ms
90% < 500ms

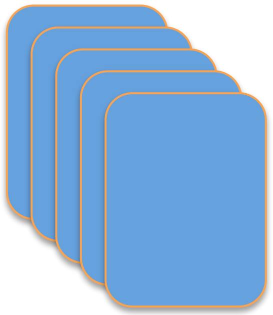
1GB / sec moved in the cloud

yql cloud = all yql system components

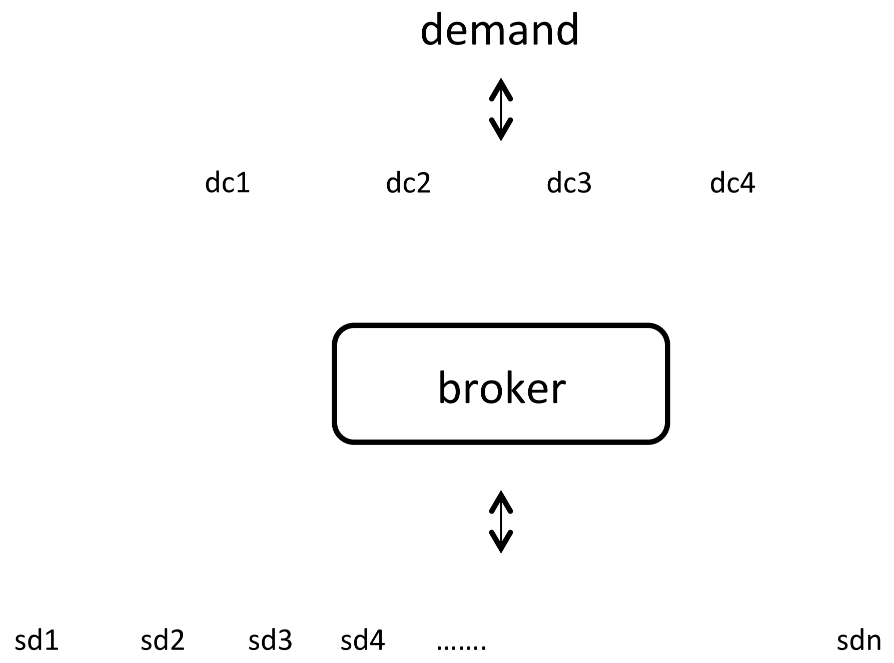
system

- Deployment
  - Active in 8 colos globally
    - US: SP2, MUD, AC4, BF1 (GQ1, NE1) = 70%
    - EU: CH1, UKL, IRD = 15%
    - AP: SG3 = 15%

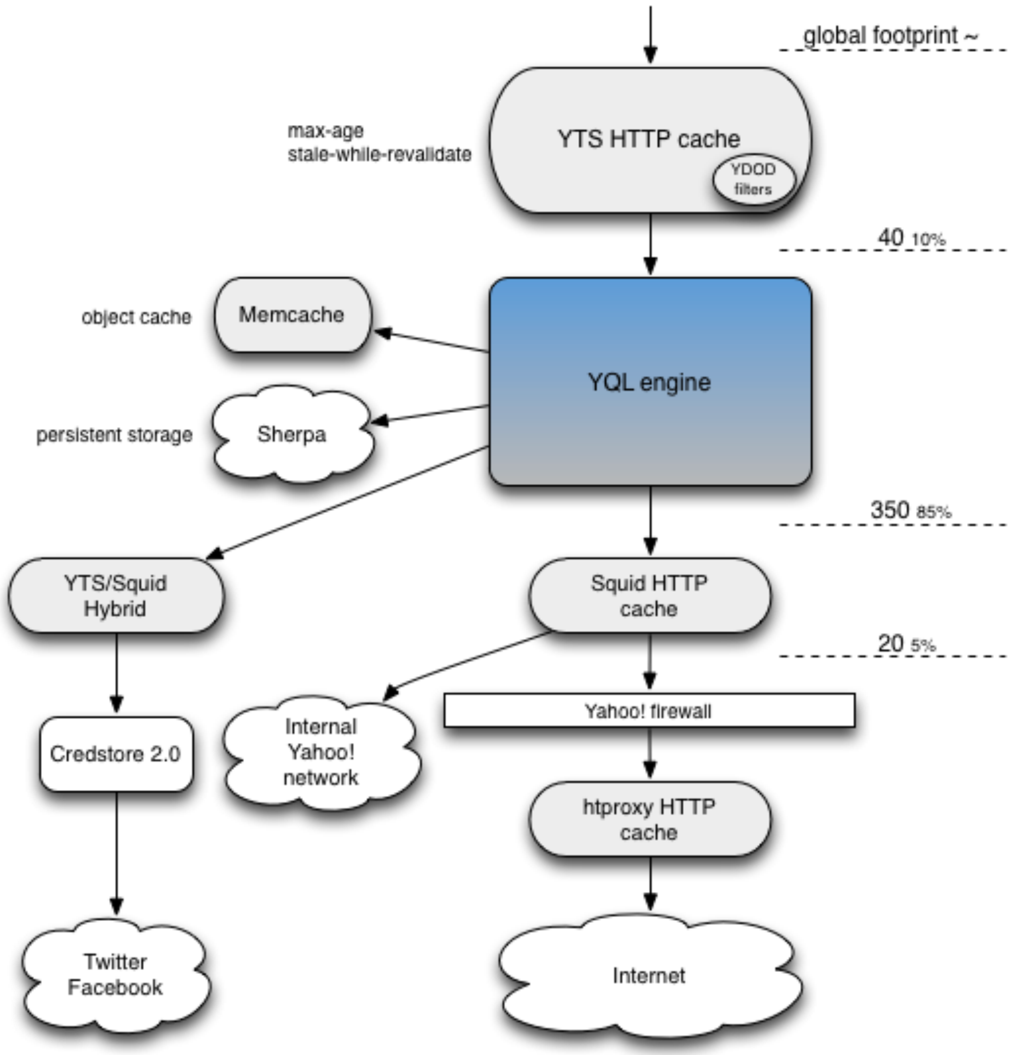
- Scales
  - horizontally
  - vertically



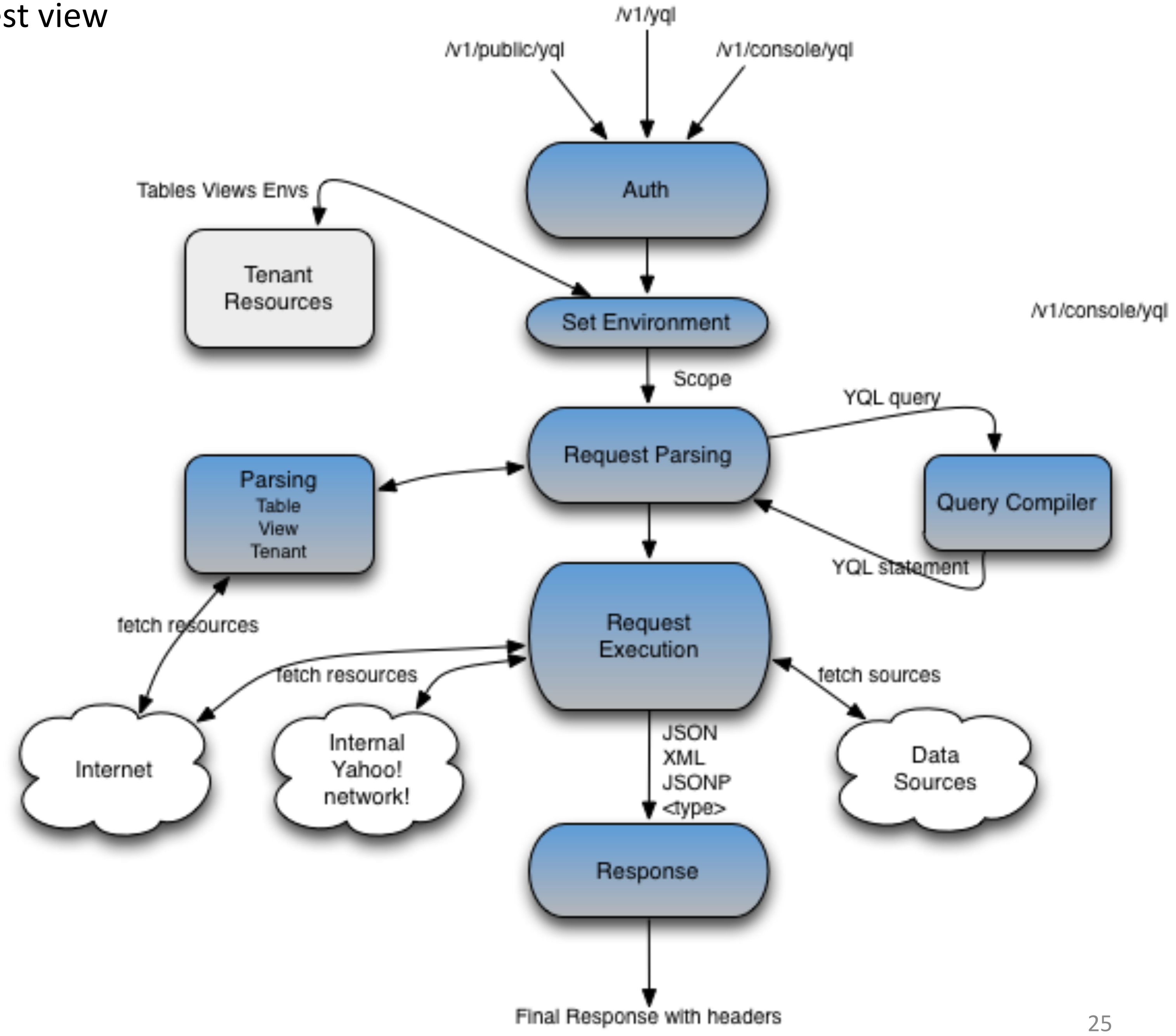
yql platform design reflects data supply and demand model



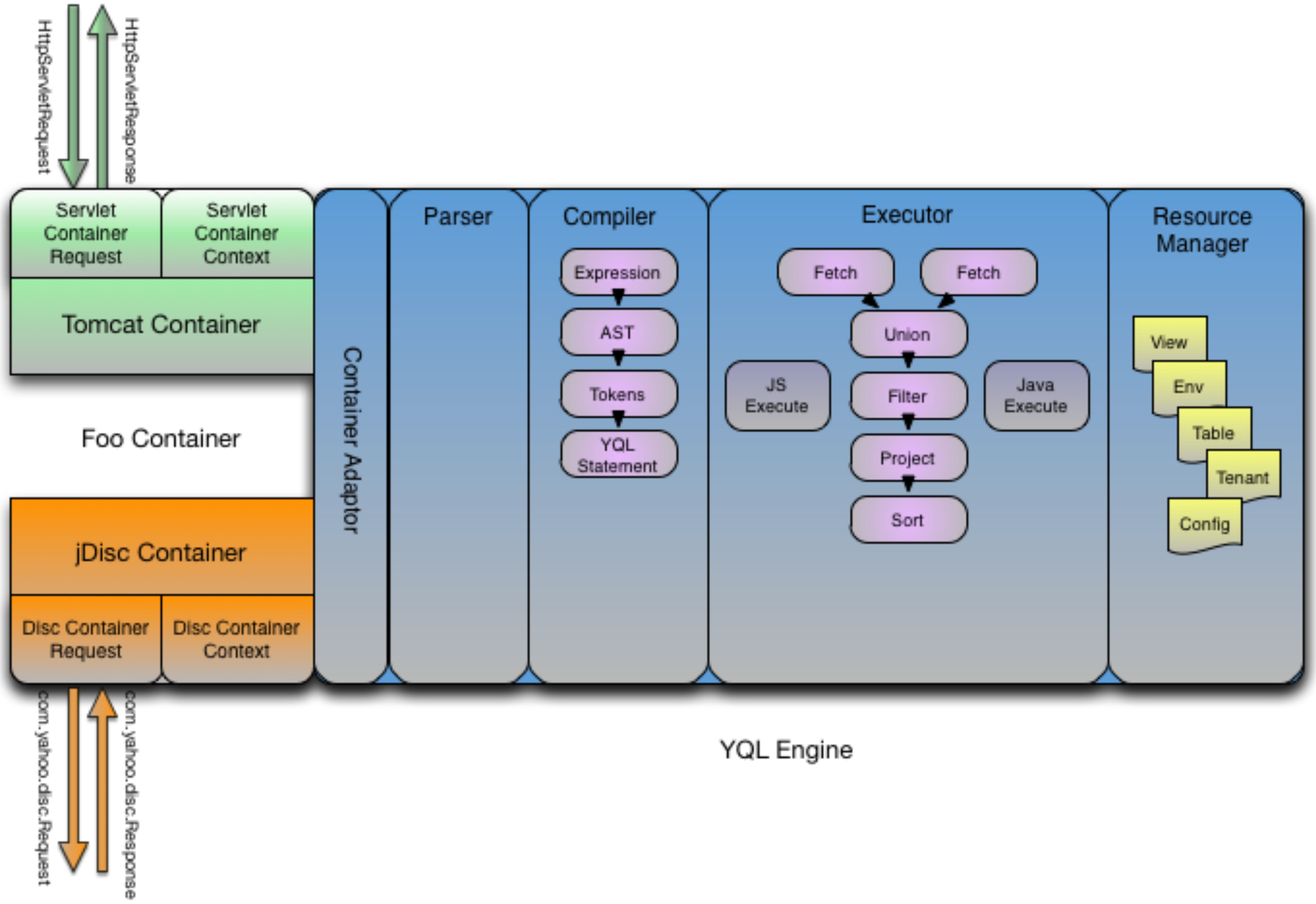
what does the production system look like



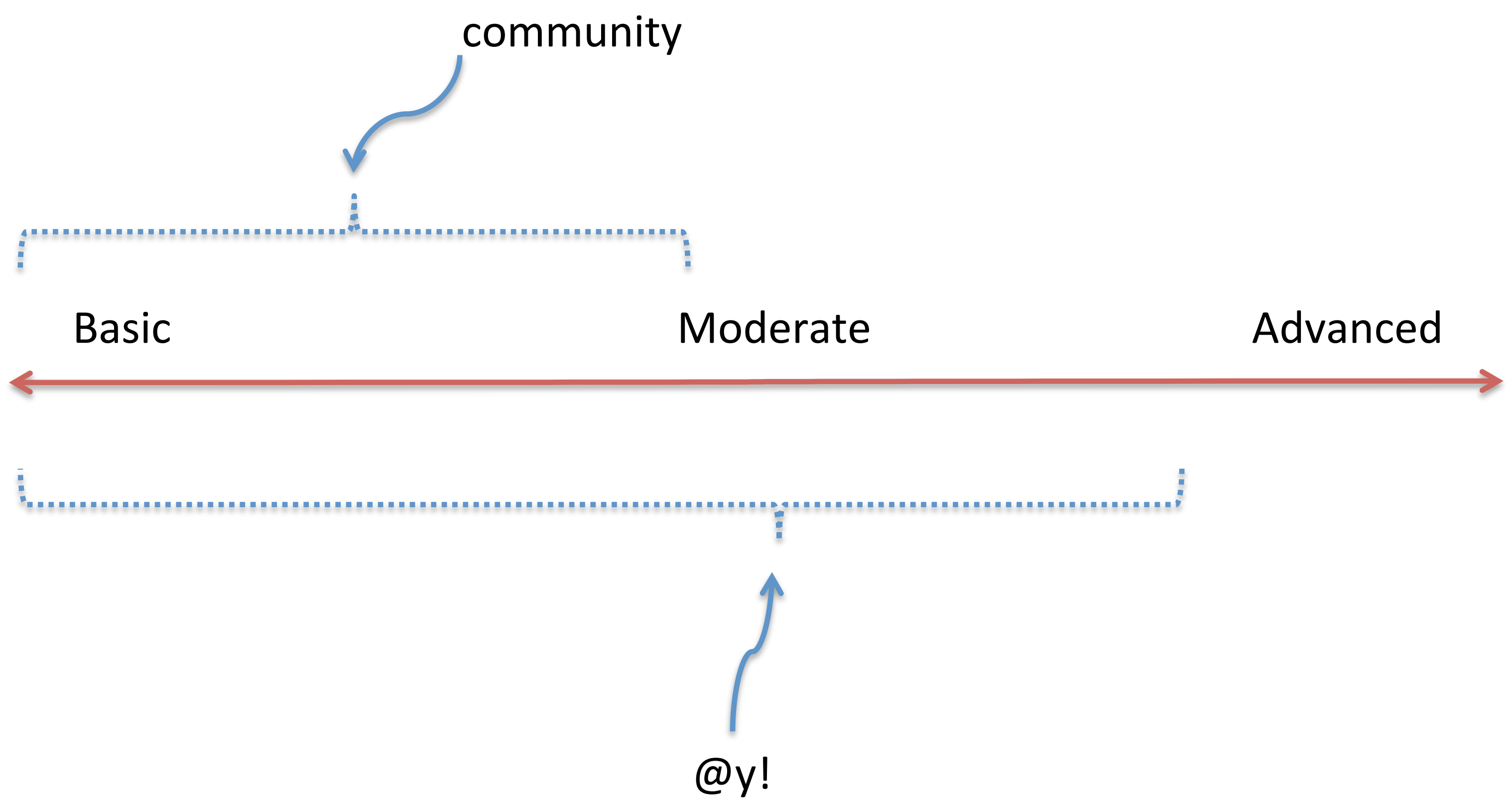
yql engine request view



yql container view



uses


community
Basic
Moderate
Advanced
@y!

Simple – access data source and use syntax features

```
select * from rss where url
        in (select title from atom where url="http://y.ahoo.it/lu5F/")
                and description like "%Wall Street%" limit 10 | unique(field="title")
```

Moderate – use execute to shape data

```
delete from twitter.status
    where id="2108869549" and username=@username and password=@password
```

```
<delete itemPath="" produces="XML">
        <urls>
                <url>http://twitter.com/statuses/destroy/{id}.xml</url>
        </urls>
        <inputs>
                <key id="username" type="xs:string" required="true" paramType="variable"/>
                <key id="password" type="xs:string" required="true" paramType="variable"/>
                <key id="id" type="xs:string" required="true" paramType="path"/>
        </inputs>
        <execute><![CDATA[
                y.include("http://yqlblog.net/samples/base64.js");
                var authheader = "Basic "+Base64.encode(username+":"+password);
                response.object = request.header("Authorization",authheader).del().response;
        ]]></execute>
</delete>
```

Advanced – native implementation of data API

```
insert into yql.storage.admin (url) values ("http://localhost/table.xml")
```

not a use

caching only

(if you use yql for its cache then something went seriously wrong elsewhere in your design and implementation)

features

there are many features

select insert update delete

https streaming charset controls encoding

fxible creation, on demand load and reload of data application

projections

sub-select join y.query

yca internal table – extended schema desc xpath

pipe table view y.rest

oauth developer console proxy selection

y.include

executable business logic in Java or JavaScript

local and remote filter authorization and access

cookies backposting

cache controls

meta services (JSON, XML, JSONP, JSONPX) cross tenant calling

environments

hosted paging flexible tenant configuration batching

store:// easy to integrate with your development and testing flow

query, filter and combine data across Yahoo! and beyond

functions sds

http endpoint table load fast easy async cache refresh

master those

pipe table view

pipe

a workflow

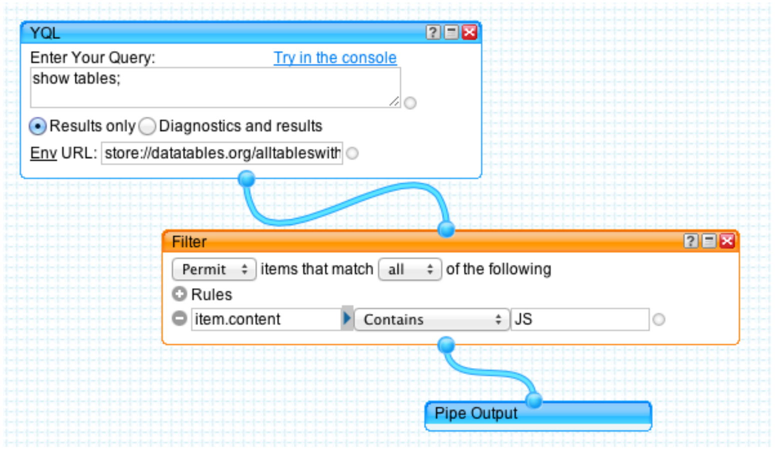
encapsulation of multiple tables and queries

expressed through JSON definition

created through GUI

instruction set bundle with operators for yql engine

available from execute y.pipe("id")



YQL

Enter Your Query: Try in the console

```
show tables;
```

[✓ Results only] Diagnostics and results

Env URL: store://datatables.org/alltableswith

Filter

Permit items that match all of the following

Rules

item.content Contains JS

Pipe Output

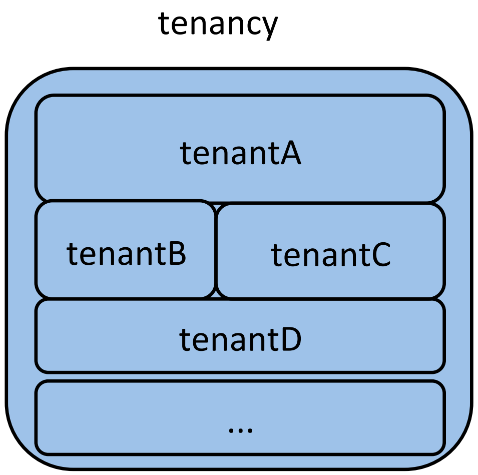
table

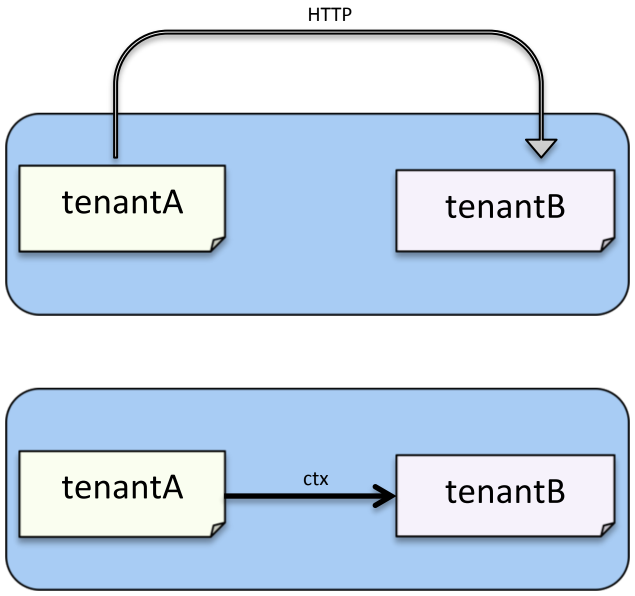
yql data app definition / plugin

runtime instruction container for yql

fundamental way to express source binding

cross tenant calling





- keeps client resources isolated from each other
    - Tables, Views, Envs
    - Secrets (keydb)
    - Certificates (yca)
    - Customized OC TTL
- authorization schemes: public, oauth, cookie/crumb, YQL token
- custom hostnames: **media.query.yahoo.com**
- tables access can be restricted per tenant
- generate default request characteristics (b-cookie, user-agent, cache control)

coming soon…

views

tenant self-provisioning

even more flexible tenant operated resource reload

…..

dblink

remote table execute

recap

yql dna is ...

a pipes effort funded by Jerry and Filo in 2005

grew this investment slowly

multi tenant cloud data serving platform

serving over 1T queries/month
while processing over 2PT

more info...

on-boarding: easy

just search backyard for yql on-boarding

do not launch with public rate limited tenant – **query.yahooapis.com**

mirek@yahoo-inc.com

yql-discuss@yahoo-inc.com

devel.corp.yahoo.com – internal yql docs

know Java well...? yql could be a place to write it – let us know