Making things Work Together

QCon NY June 20, 2012

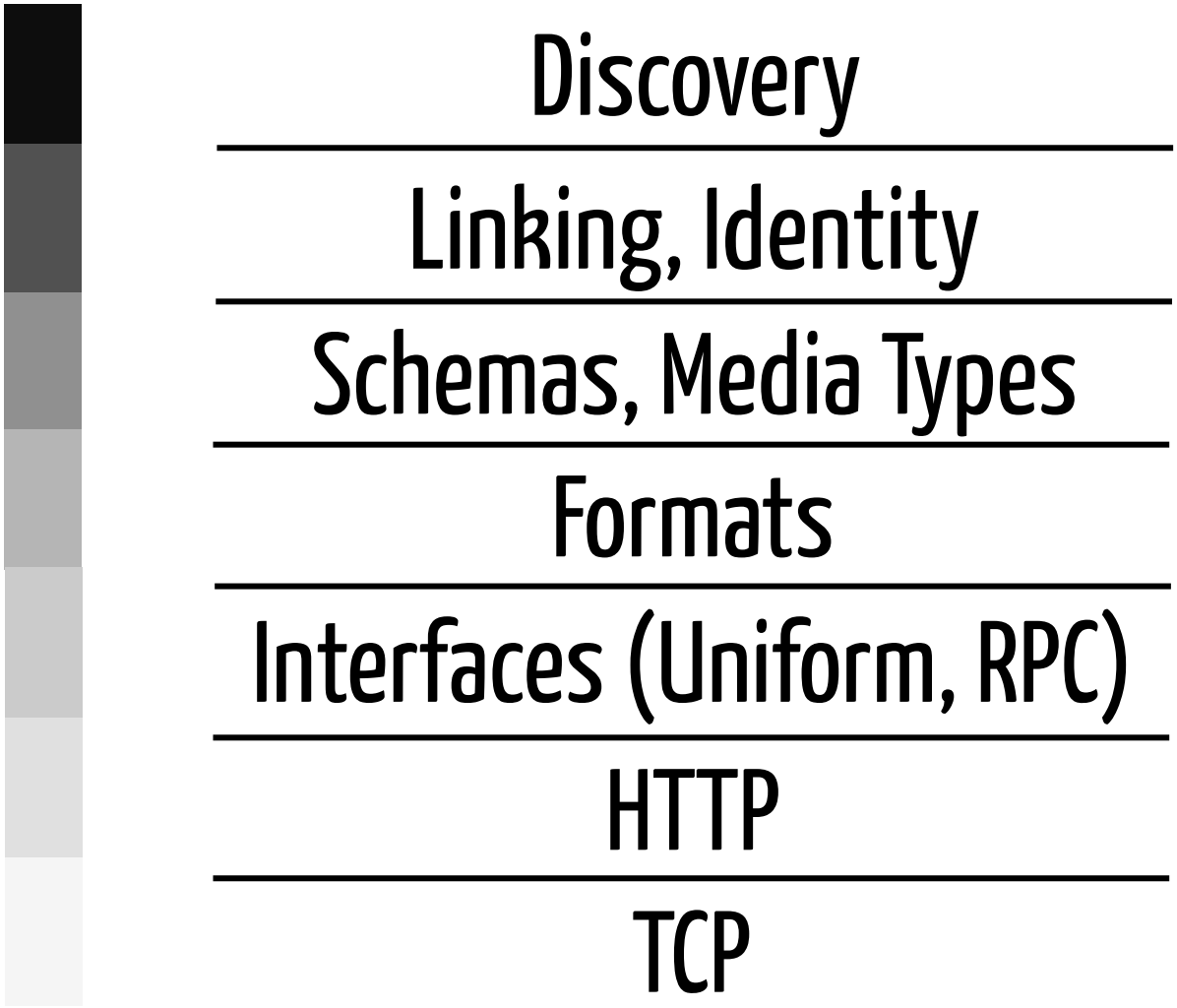
Subbu Allamaraju

@sallamar

github.com/s3u

How to build an interoperable Programmable Web?

Agreements


Discovery
Linking, Identity
Schemas, Media Types
Formats
Interfaces (Uniform, RPC)
HTTP
TCP

Server's point of view

Client's point of view

Several hundred APIs
Different styles
Written by different teams
Over years
Built under different constraints
Built with different goals

Use case:

Find things from A
For each thing, find details from B
For each thing, find more details from C
Merge results

Use case:

Find products
Find dominant categories of products
Look up category info
Merge categories with products

Use case:

Get stuff from A
If A is down, try from B
Annotate stuff with other stuff from C
Ignore some things from the stuff
Join all

Client

[Really important client]

Why don't you give me an API optimized for my use cases?

[Really important producer]

Thanks. Get a number and stand in the line!

Server

Client

[Really important client]

Why don't you give me an API optimized for my use cases?

Who gets to decide the right thing?

[Really important producer]

Thanks. Get a number and stand in the line!

Server

Discovery

Linking, Identity

Schemas, Media Types

Formats

> Interfaces (Uniform, RPC)

HTTP

TCP

Discovery

Linking, Identity

> Schemas, Media Types

Formats

HTTP

TCP

> Discovery

> Linking, Identity

Formats

HTTP

TCP

Formats

HTTP

TCP

Smart Client Code

Latency

Failure Recovery

Orchestration

IO

Formats

HTTP

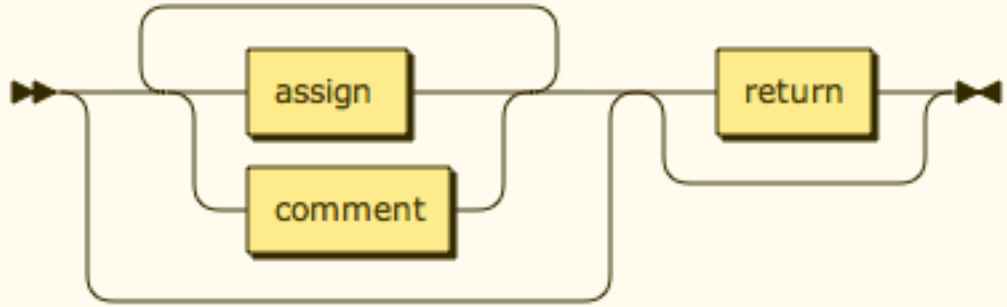
TCP

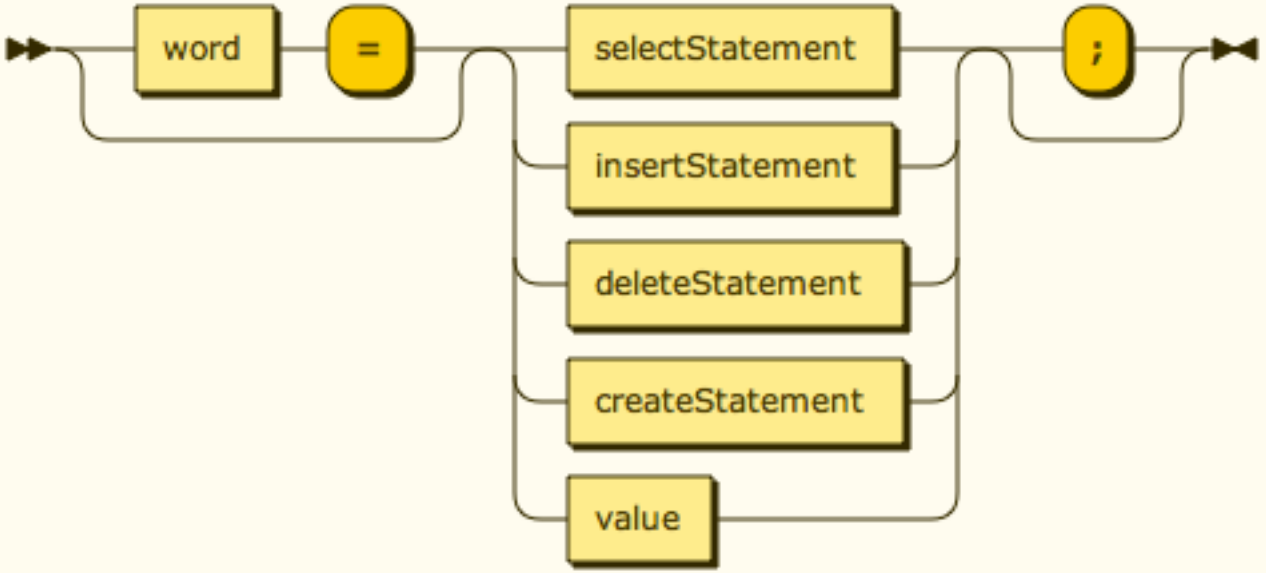
ql.io

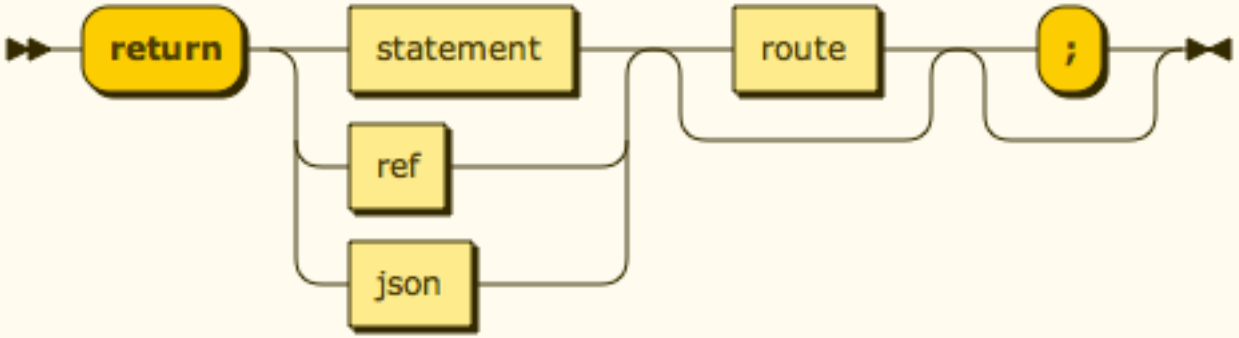
https://github.com/ql-io/ql.io

Platform Engineering

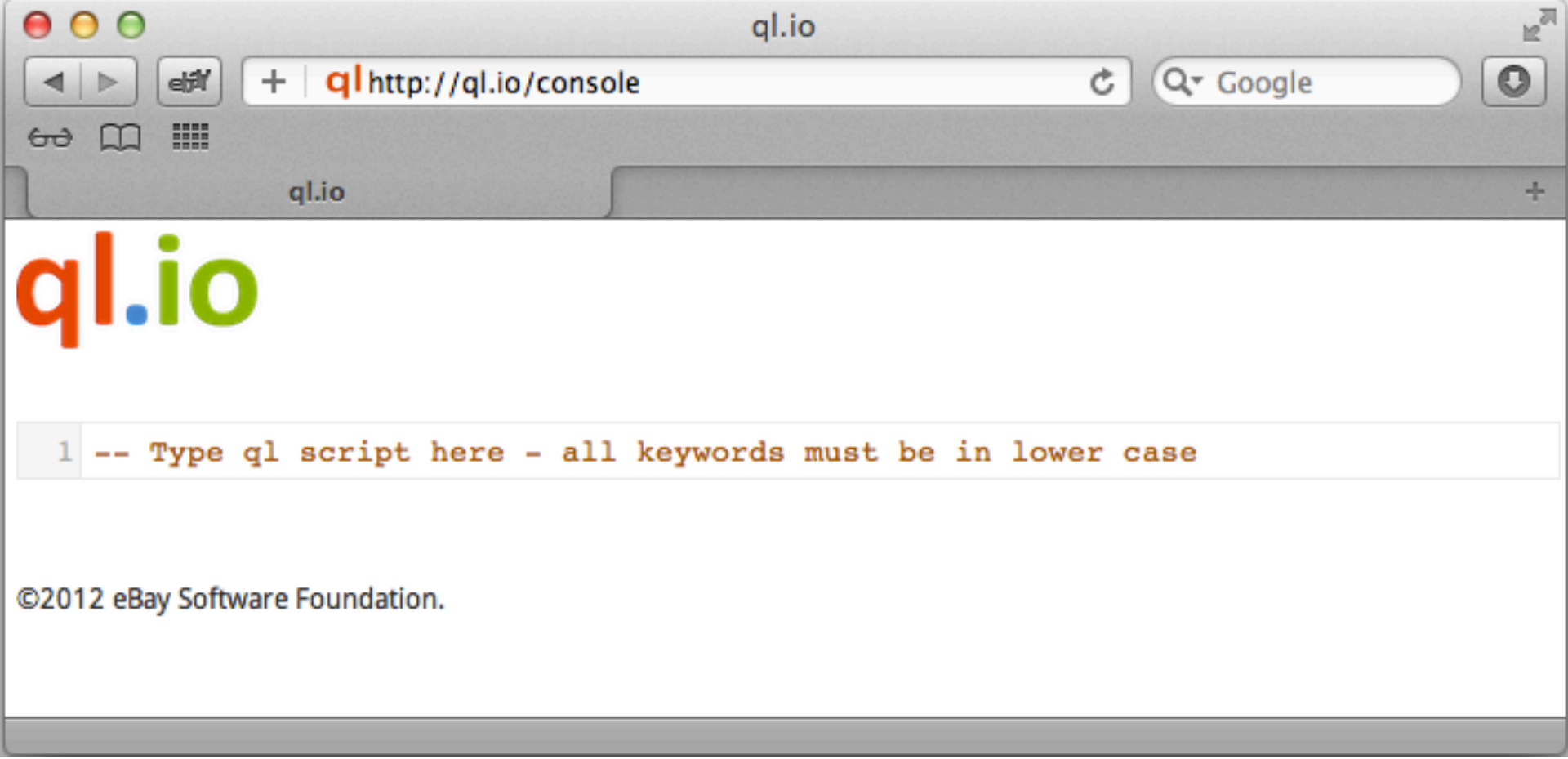
A domain specific language for HTTP client programming


assign
comment
return

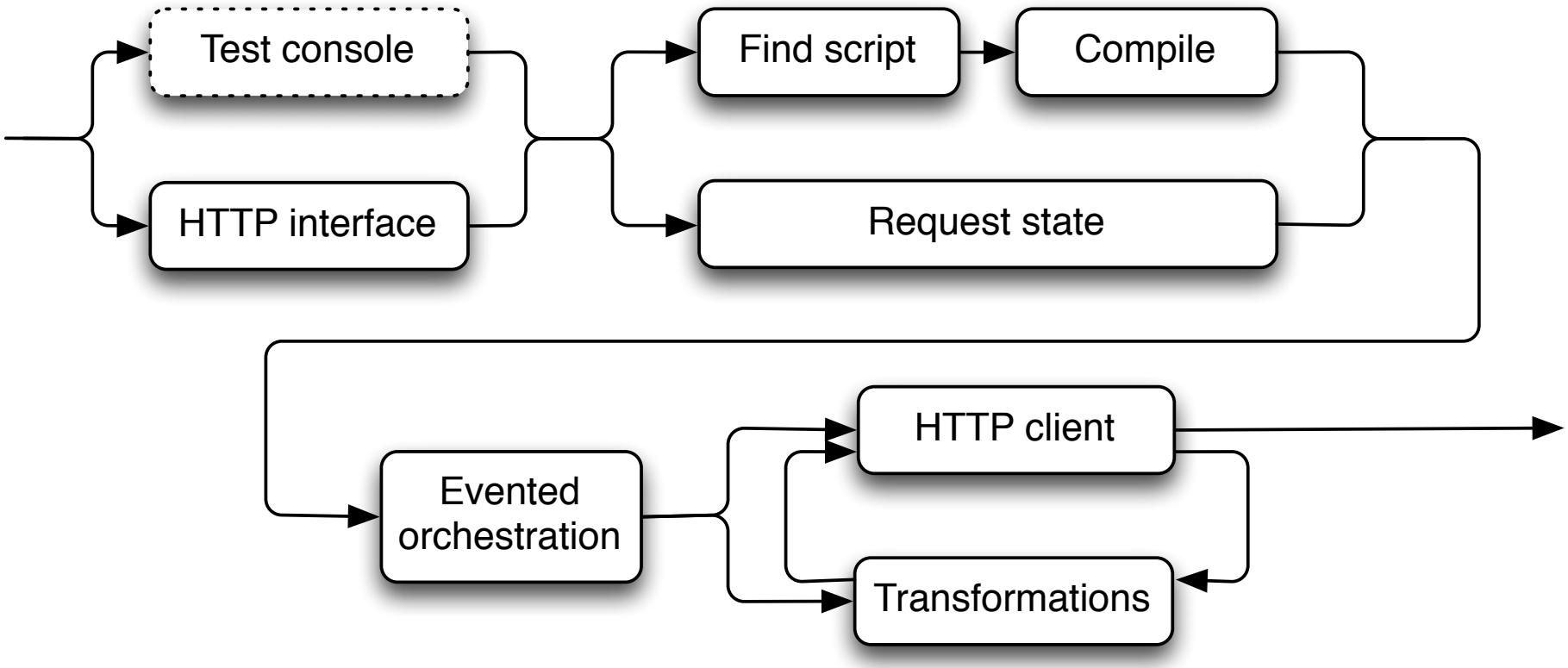

word
=
selectStatement
insertStatement
deleteStatement
createStatement
value
;


return
statement
ref
json
route
;

```
prodid = select ProductID[0].Value from eBay.FindProducts
    where QueryKeywords = '{q}';
details = select * from eBay.ProductDetails
    where ProductID in ('{prodid}') and
    ProductType = 'Reference';
reviews = select * from eBay.ProductReviews
    where ProductID in ('{prodid}') and
    ProductType = 'Reference';

return select d.ProductID[0].Value as id,
                d.Title as title,
                d.ReviewCount as reviewCount,
                r.ReviewDetails.AverageRating as rating
        from details as d, reviews as r
        where d.ProductID[0].Value = r.ProductID.Value
        via route '/myapi' using method get;
```



http://ql.io/console



A runtime designed for IO workloads
Runs on node.js

1. Interop via HTTP

Support the protocol – but don't require style adherence

```
create table mytable1
  on select
  get from 'http://...';

create table mytable2
  on insert
  get from 'http://...&do=create'

create table mytable3
  on delete
  post to 'http://...'
  using headers 'X-Do' = 'Delete'
```

2. Interop using common formats

Agnostic of domain specific type systems

```
/* A table with a body template */
create table bing.soap.search
  on select post to 'http://api.bing.net/
soap.asmx'
  using defaults appid = 'xxx'
  using bodyTemplate 'bing.xml.mu'
  type 'application/xml'
  resultset
'soapenv:Envelope.soapenv:Body.SearchResponse.pa
rameters.Web.Results.WebResult';
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <SearchRequest xmlns="http://schemas.microsoft.com/
LiveSearch/2008/03/Search">
      <parameters>
      {{#params}}
      <Query>{{q}}</Query>
      <AppId>{{appid}}</AppId>
      {{/params}}
      <Sources>
        <SourceType>Web</SourceType>
      </Sources>
      </parameters>
    </SearchRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

```
-- Form encoded body
create table mytable
  on insert post to 'http://...'
  using bodyTemplate 'somefile.txt'
    type 'application/x-www-form-urlencoded'

-- Or JSON
create table mytable
  on update put to 'http://...'
  using bodyTemplate 'file.json' type
'application/json'
```

```
create table {a name}
on {select|insert|update|delete}
    {get from|post to|
     put to|delete|patch}
    {a URI template}
using headers {name}={value}*
using bodyTemplate={a file}
        type={media type}
```

```
create table {a name}
on {select|insert|update|delete}
    {HTTP method st to|
     put to|delete | patch}
    {URI template}
usir Headers s {name}={value}*
usir Body dyTemplate={a file}
       type={r Media type
```

```
-- A table with a monkey patch
create table bitly.shorten
  on insert get from "..."
    using patch 'bitly.js'
  on select get from "..."
    using patch 'bitly.js'

/* bitly.js A Javascript monkey patch */
exports['patch status'] = function(options) {
    return args.body ?
      args.body.status_code : 200;
};
```

3. SQL inspired constructs

CRUD operations,
filtering, joins

```
-- Get the complete response as JSON
select * from google.geocode where
  address = 'Brooklyn, NY'

-- Project fields
select ProductID[0].Value,
  DetailsURL, StockPhotoURL
  from eBay.FindProducts
  where QueryKeywords='mini cooper'
```

```
/* Join datasets without enforcing identity
   across systems */

select e.ItemID as id, e.Title as title,
  e.ViewItemURLForNaturalSearch as url,
  g.geometry.location as latlng
  from details  as e, google.geocode as g
  where e.itemId in (select itemId from
  finditems where keywords = "iPad")
  and g.address = e.Location
```

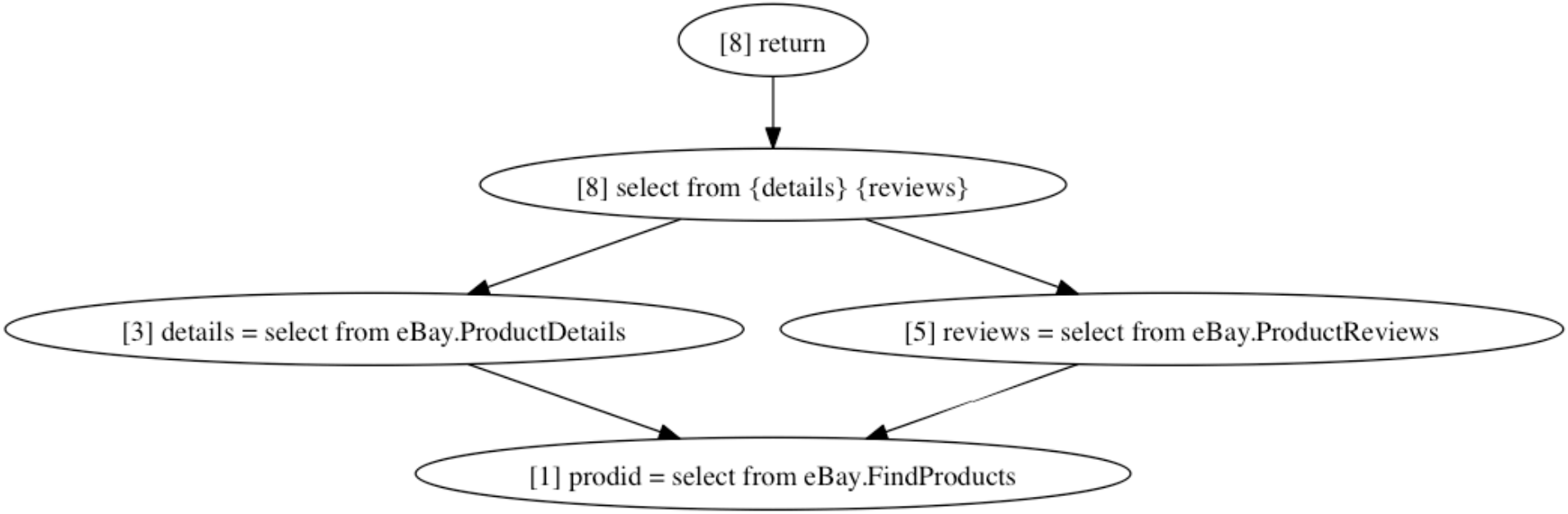
4. Orchestration

Implicit sequential, scatter, parallel, forks and joins

```
-- Sequential
minis = select * from finditems where
    keywords = 'mini cooper' limit 10;
return select PictureURL from details where
    itemId = "{minis.itemId}";

-- Or parallel
keyword = "ql.io";
web = select * from bing.search where q = "{keyword}";
tweets = select id as id, from_user_name as user_name,
    text as text  from twitter.search where q = "ql.io";

return {
  "keyword": "{keyword}",
  "web": "{web}",
  "tweets": "{tweets}"
}
```

```
prodid = select ProductID[0].Value from eBay.FindProducts
    where QueryKeywords = 'macbook pro';

-- Scatters "n" requests
details = select * from eBay.ProductDetails where
    ProductID in ('{prodid}') and
    ProductType = 'Reference';

-- Scatters "n" requests
reviews = select * from eBay.ProductReviews where
    ProductID in ('{prodid}') and
    ProductType = 'Reference';

-- Gather, merge, and join
return select d.ProductID[0].Value as id,
    d.Title as title, d.ReviewCount as reviewCount,
    r.ReviewDetails.AverageRating as rating
  from details as d, reviews as r
    where d.ProductID[0].Value = r.ProductID.Value
```

```
-- An API that takes one ID at a time
create table eBay.ProductDetails
  on select get from "...&...={^ProductID}"
  ...
  resultset 'Product';


-- A batch API – takes upto 20 IDs
create table details
  on select get from "...&ItemID={20|itemId}
  ...
  resultset 'Item';
```


[8] return
[8] select from {details} {reviews}
[3] details = select from eBay.ProductDetails
[5] reviews = select from eBay.ProductReviews
[1] prodid = select from eBay.FindProducts

5. Failure Modes

Timeouts, back-off and recovery

```
-- Timeout on idle sockets
-- Retry once for idempotent statements
select id, img from recommendations
  timeout 100

-- On three successive failures, backoff
-- for 100 – 2000 msec
select id, img from recommendations
  timeout 100 minDelay 100 maxDelay 2000
```

```
-- Use a fallback
select id as id, item_url as img
  from recommendations
  timeout 100
  ||
  select ID as id, deal_url as img
    from deals;
```

6. Interface

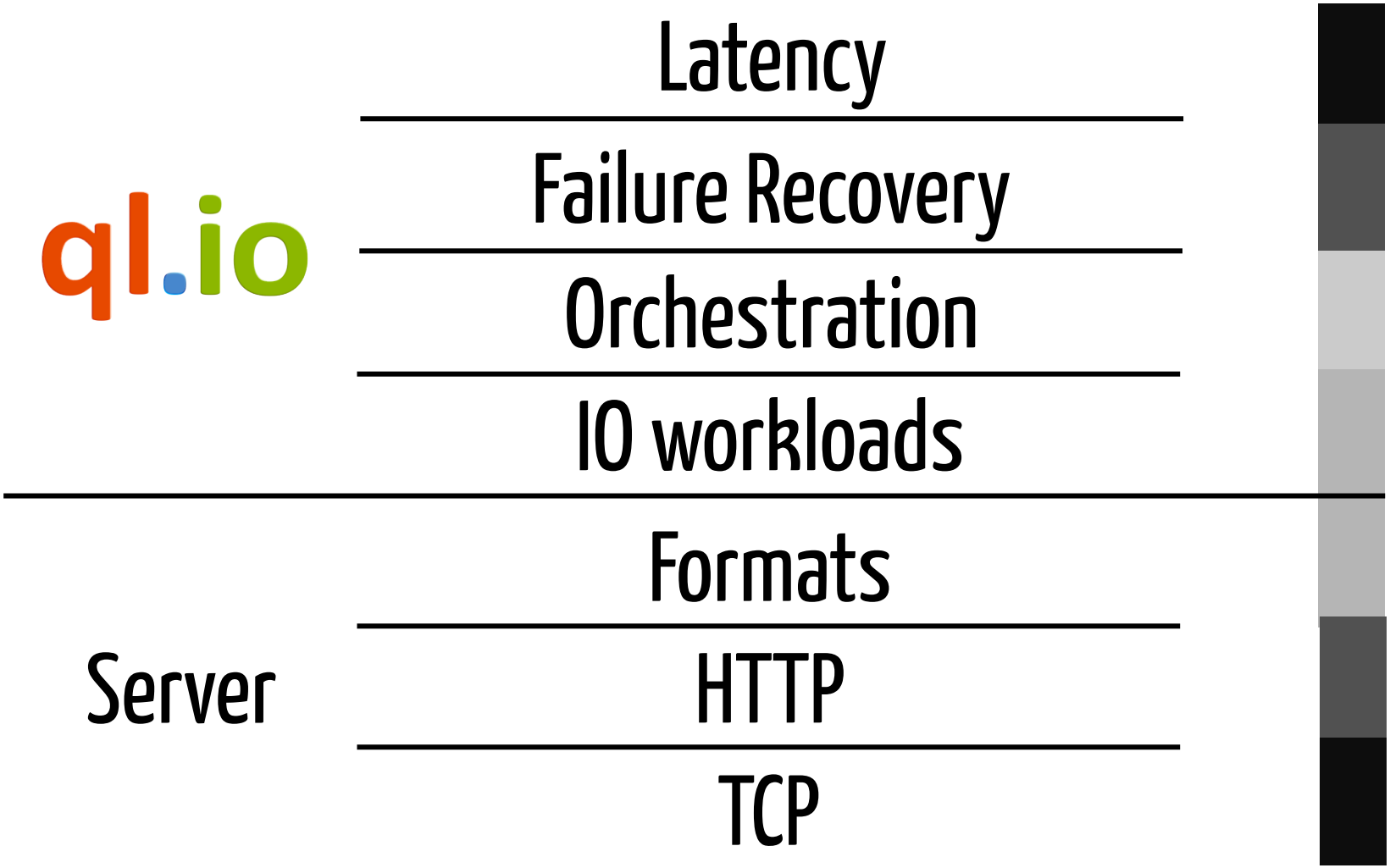
Consumer-specified

```
var Engine = require('ql.io-engine'),
    fs = require('fs');
var engine = new Engine({
    tables : __dirname + '/../tables',
    config: __dirname + '/../config/dev.json'
});
var script = fs.readFileSync(__dirname + '/some.ql', 'UTF-8');

// Exec the script
engine.execute(script, function(emitter) {
    emitter.on('end', function(err, result) {
        result.body.forEach(function(row) {
            console.log(row);
        });
    });
});
```

Push orchestration to the data center

- Reduce HTTP requests
- Reduce bandwidth use

```
/* Shows daily deals - use siteId=0 for US and siteId=3
for UK. */
resp = select * from dailydeals where siteId="{siteId}";
deals = "{resp.$..Item}";
itemDetails = select ItemID as itemId, Title as title,
  GalleryURL as pic, Seller.UserID as sellerUserId,
  Seller.FeedbackScore as rating,
  Seller.PositiveFeedbackPercent as feedback,
  HitCount as hits from details
  where itemId in (deals.ItemID);

return itemDetails via route '/deals/{siteId}' using
method get;
```


ql.io
Latency
Failure Recovery
Orchestration
IO workloads
Server
Formats
HTTP
TCP

code

https://github.com/ql-io/ql.io

docs/demos

http://ql.io

blog

http://ql-io.github.com