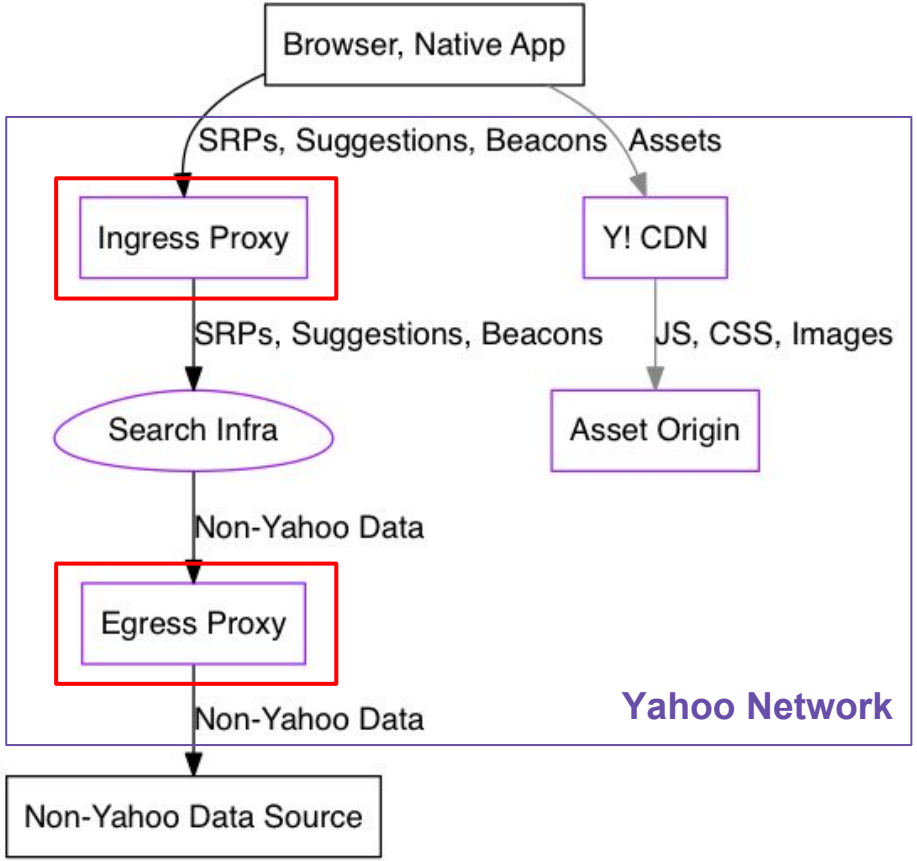
Yahoo Search ATS Plugins

Daniel Morilha and Scott Beardsley

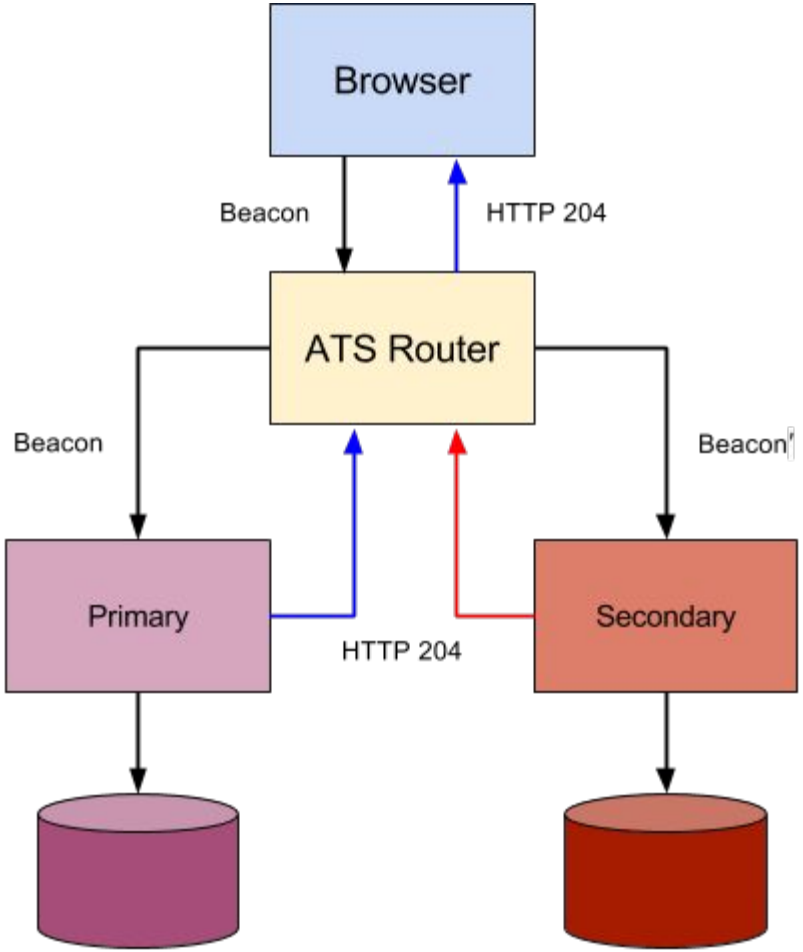
About Us

- We have a **HUGE** team!
- Serves traffic which generates ~40% of Yahoo's $$$
- We run both Search Ingress and Egress
- Maintain around a dozen plugins
- Plugins installed across ATS tiers in Yahoo
- Covering the following plugins today
  - Multiplexer [OS]
  - Where on Earth (WOE) and Sonora [Y]
  - Http Filters [OS]
  - Image Inliner [OS eta Q1 2015]
  - Zeus configuration [OS]



Problem: multiple reporting pipelines

- One request from the browser
- Multiple origin implementations
- Need a way to validate capacity
- Need a way to validate end-to-end reporting
- Drop response from non-primary origin

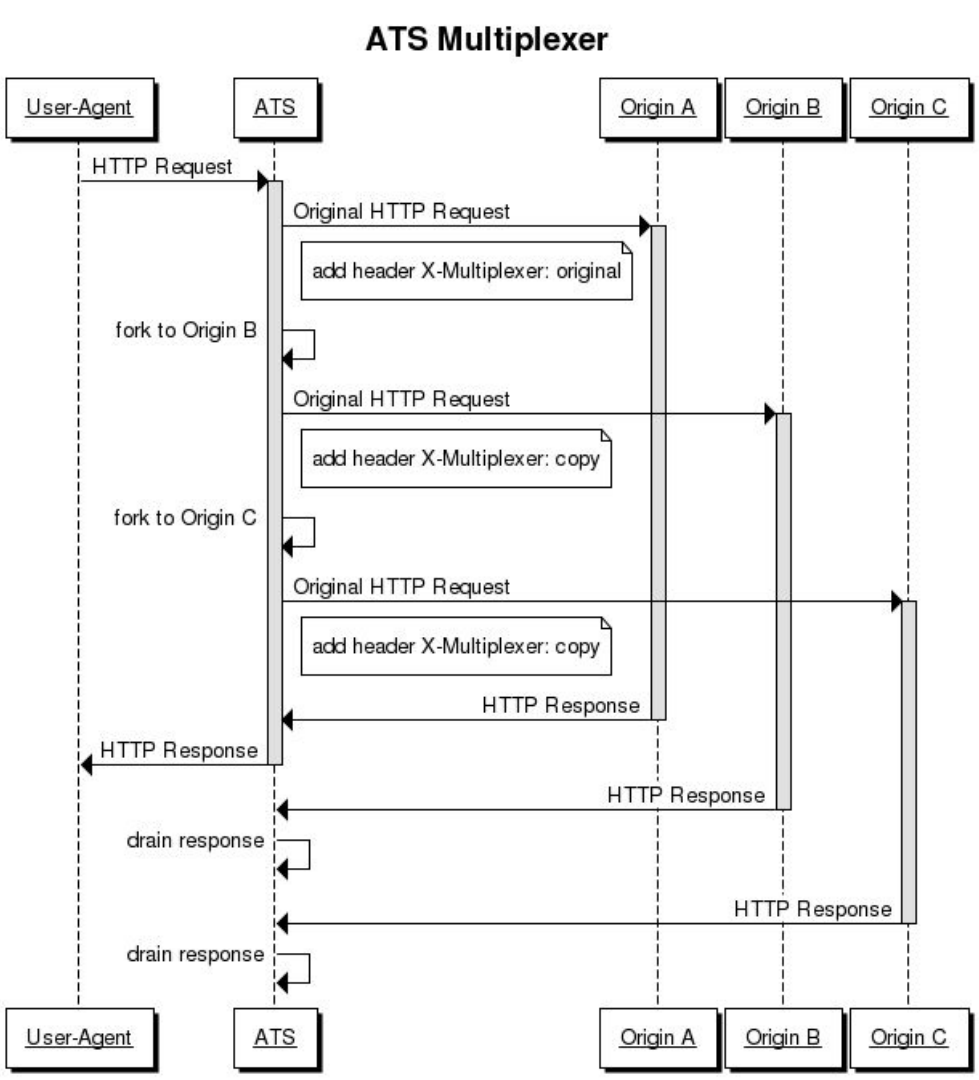


Multiplexer

- Remap plugin
- Multiplex UA requests to multiple origins
- Supported methods: GET, POST
- Env Variable for non-primary origin timeout (ns)
  - `multiplexer__timeout`
  - Defaults to 1 second
- Metrics!

```
$ sudo traffic_line -m multiplexer.*
multiplexer.requests 1998
multiplexer.hits 1998
multiplexer.failures 0
multiplexer.timeouts 0
multiplexer.time 2408 µs
multiplexer.size 1091b
```

- **Open Sourced Oct. 2015**: https://tr.im/multiplexer



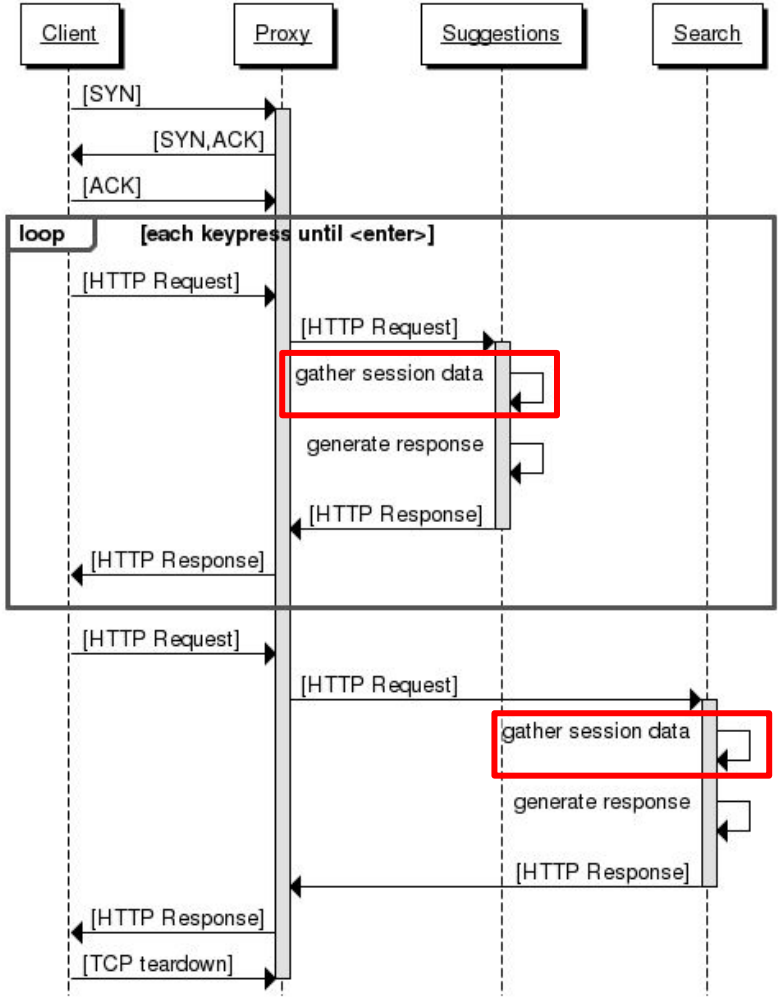
Multiplexer

- Example remap.config:

```
map /click http://a.example.com/clk @plugin=multiplexer.so @pparam=b.example.com @pparam=c.example.com

# internal_only is for remaps which are only needed by ATS plugins
.definefilter internal_only @action=allow @src_ip=127.0.0.1 @src_ip=::1
.activatefilter internal_only

map http://b.example.com/click http://b.example.com/clk
map http://c.example.com/click http://c.example.com/some_other_path @plugin=conf_remap.so @pparam=nopristine.config

.deactivatefilter internal_only
```

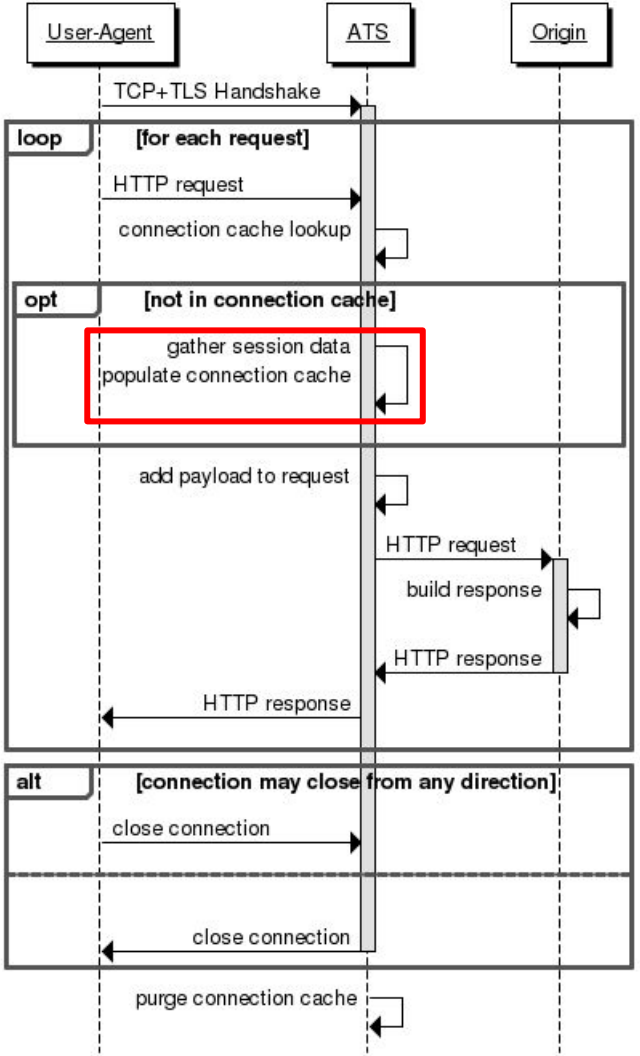
Problem: transient session data

- session == lifetime of a TCP connection
- need to keep session data server-side
- existing solution: repeat session lookups for each request
- multiple session lookup concerns:
  - large dataset distribution
  - consistency
  - latency
- centralized session cache
  - ~~large dataset distribution~~
  - ~~consistency~~
  - latency



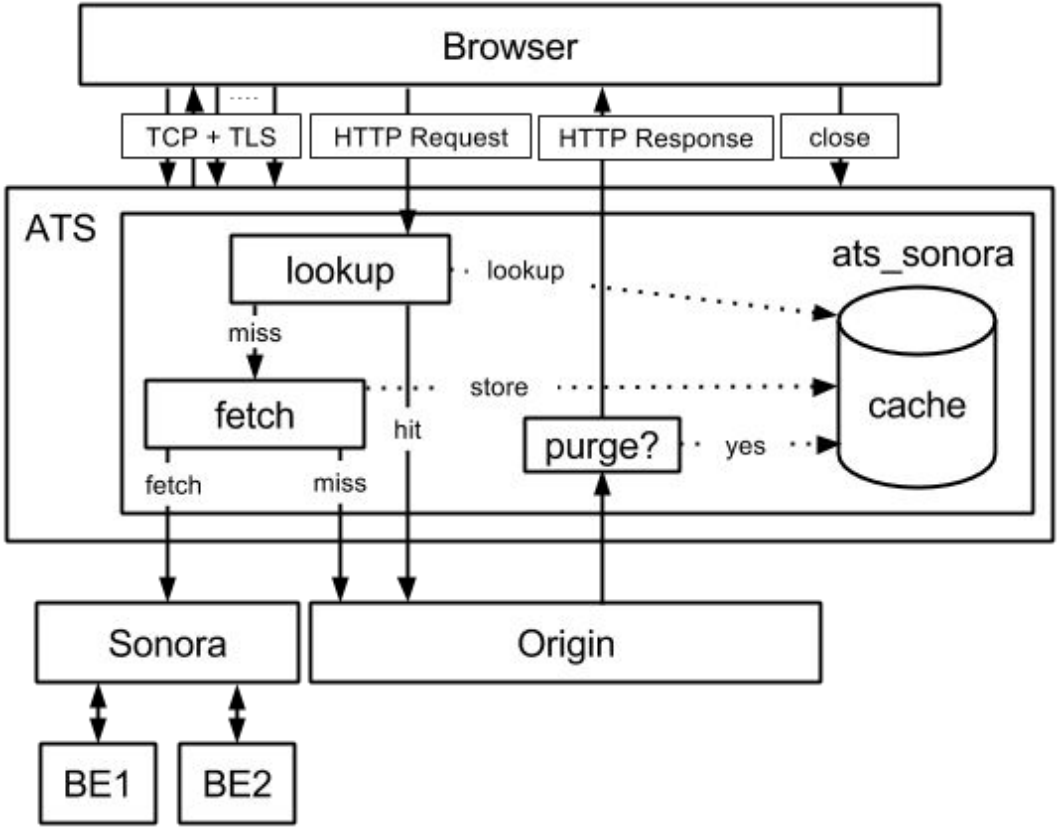
Where on Earth

- Global plugin
- Translate Client IP to a WOE IDs
  - woeids, names, confidence levels
- Introduces a new “connection cache” concept
- WOE database size ~3GB
- IP override via **.ip** query parameter
- Production Statistics (April 2015):
  - Avg lookup time: **80ns**
- TODO
  - X-Forwarded-For parsing?
  - Fix TS-3612 to **improve CHR to 80%**



Sonora

- Almost identical to Where on Earth plugin
- Remote API call instead of local DB lookup
- Production Stats (April 2015):
  - Avg lookup time: **20ms**
- Handles “purge” response header directives
- TODO
  - Improve ACLs
  - Parse Sonora response
    - skip cache insertion on error
    - routing actions with backend data
  - Fix TS-3612 to **improve CHR to 80%**



Problem: generic request classifier

- Determine if request matches a set of criteria (aka filter)
- If request matches a filter remove from experiment
- Use the following as input:
  - HTTP method
  - URL
  - request headers
    - from client
    - from other plugins
  - cookies
- Make decisions fast!
- Allow classification logic to change frequently

```
<filter name="mobile_beta_testers">
  <and>
    <equal header="X-Device-Type" value="smartphone"/>
    <contains header="X-WOE" value="country=US"/>
    <greater-than after="confidence_country=" header="X-WOE" value="75"/>
    <not>
      <or>
        <and>
          <greater-than header="X-Os-Version" value="6"/>
          <equal header="X-Os-Name" value="ios"/>
          <or>
            <equal header="X-Browser-Name" value="safari"/>
            <equal header="X-Browser-Name" value="safari mobile"/>
          </or>
        </and>
        <and>
          <greater-than header="X-Os-Version" value="3"/>
          <equal header="X-Os-Name" value="android"/>
          <or>
            <equal header="X-Browser-Name" value="chrome"/>
            <equal header="X-Browser-Name" value="chrome mobile"/>
          </or>
        </and>
      </or>
    </not>
  </and>
</filter>
```

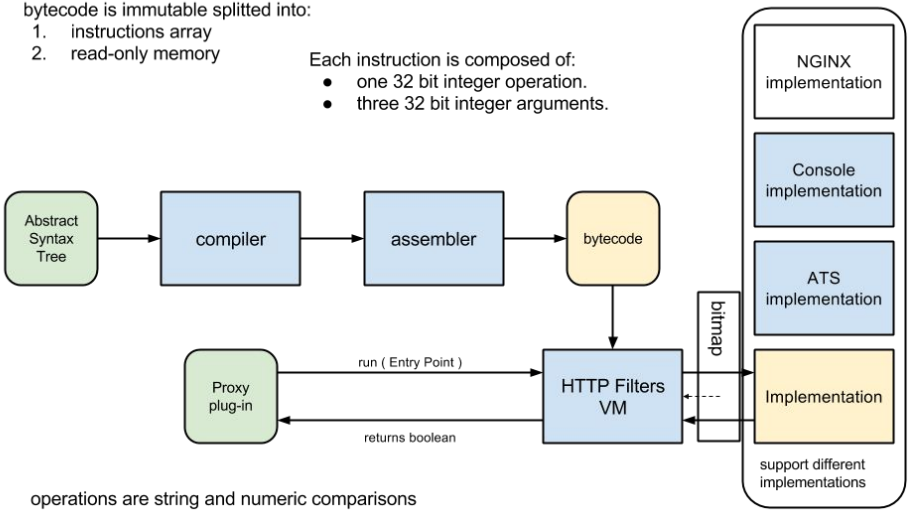
HTTP Filters

- Integrated with ATS-based experimentation classifier
- Boolean expressions are compiled during run-time
- No external dependencies
- Full compilation pipeline starting on Abstract Syntax Tree all the way to interpreted byte-code
- Can be easily ported to other HTTP servers/proxies
- Fast: experiment selection and filtering logic: **<0.5ms**
- **Open Sourced Apr. 2015**: https://github.com/yahoo/http-filters

HTTP Filters

bytecode is immutable splitted into:
1. instructions array
2. read-only memory

Each instruction is composed of:
- one 32 bit integer operation.
- three 32 bit integer arguments.



operations are string and numeric comparisons into the HTTP request:
- method.
- url (scheme, domain, path, query string).
- headers.
- cookies.

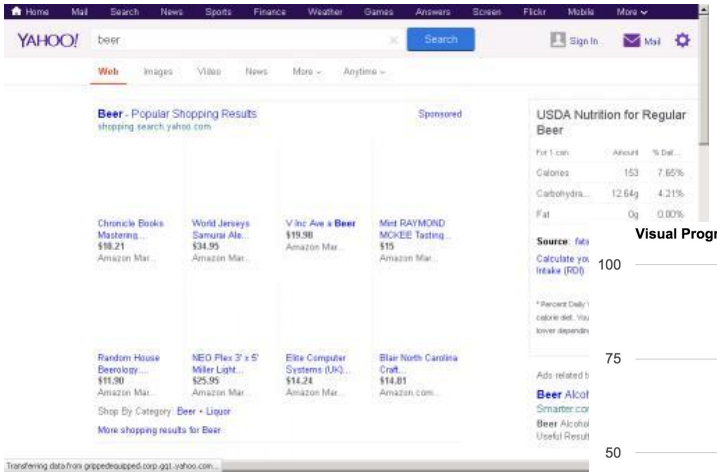
The VM does not change the request.

```
printing vm code
     0 0x000000 kSkip                        0x0       0x0       0x0

 -- Entry 1 --
     1 0x100000 kIsMethod                    0x1       0x3       0x0
 -> "GET"
     2 0x200000 kIsScheme                    0x5       0x4       0x0
 -> "http"
     3 0x300000 kReturn                      0x0       0x0       0x0
     4 0x400000 kExecute                     0x2       0x1       0x0       // kAnd, Entry 1
     5 0x500000 kReturn                      0x0       0x0       0x0
     6 0x600000 kHalt                        0x0       0x0       0x0

 -- Entry 2 --
     7 0x700000 kExistsHeader                0xa       0x0       0x0
 -> "User-Agent"
     8 0x800000 kContainsHeader              0xa       0x15      0x0
 -> "User-Agent"
 -> "Firefox"
     9 0x900000 kReturn                      0x0       0x0       0x0

 -- Entry 3 --
    10 0xa00000 kExistsHeader                0x1d      0x0       0x0
 -> "user-agent"
    11 0xb00000 kContainsHeader              0x1d      0x15      0x0
 -> "user-agent"
 -> "Firefox"
    12 0xc00000 kReturn                      0x0       0x0       0x0

 -- Entry 4 --
    13 0xd00000 kExecute                     0x2       0x7       0x0       // kAnd, Entry 2
    14 0xe00000 kExecute                     0x2       0xa       0x0       // kAnd, Entry 3
    15 0xf00000 kReturn                      0x0       0x0       0x0
    16 0x100000 kExecute                     0x3       0xd       0x0       // kOr, Entry 4
    17 0x110000 kReturn                      0x0       0x0       0x0
    18 0x120000 kHalt                        0x0       0x0       0x0
    19 0x130000 kContainsDomain              0x28      0xa       0x0
 -> ".yahoo.com"
```

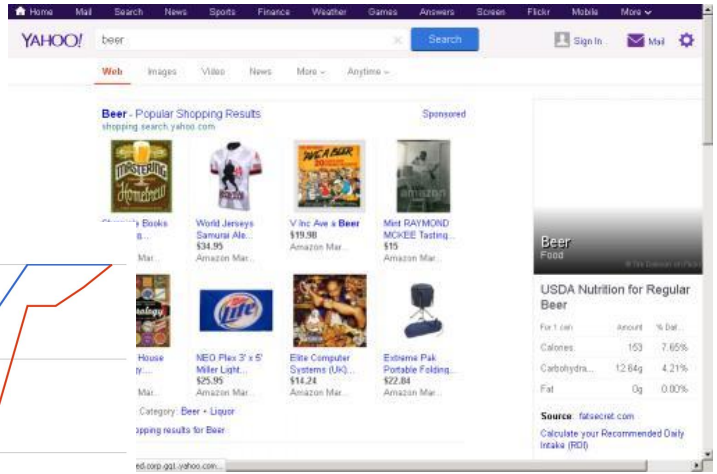
Problem: insert images into HTML

- Goal is to return all content in one response
  - improve render time by **~300ms!**
- RFC 2397 defines data URIs: `data:[<mediatype>][;base64],<data>`
- Works great for non H2/SPDY clients
- For H2/SPDY it is best to use Server Push
- Requires origin changes (server hints `Link:<logo.png>;rel=subresource`)
- Requires streaming origin to know at HTTP response header generation
- Requires origin to have a copy of the images (and as a result a cache)
- Priority should be given to Above-the-Fold images
- Deliver image content at end of page to allow browser to render text
- Ideally image urls would be detected, image content fetched, cached, and injected (or server pushed) at the edge

Before



After



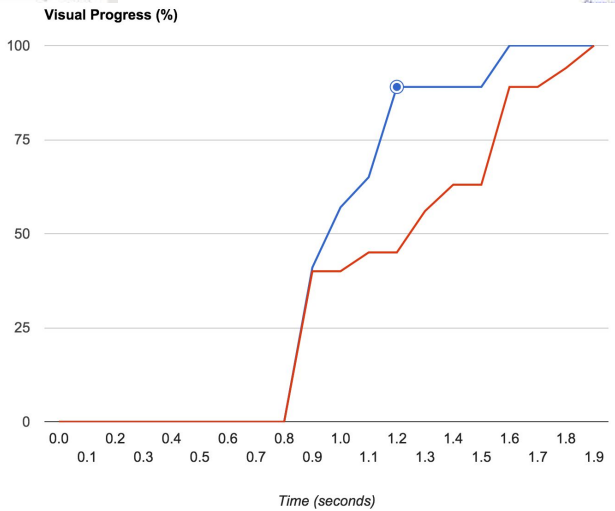
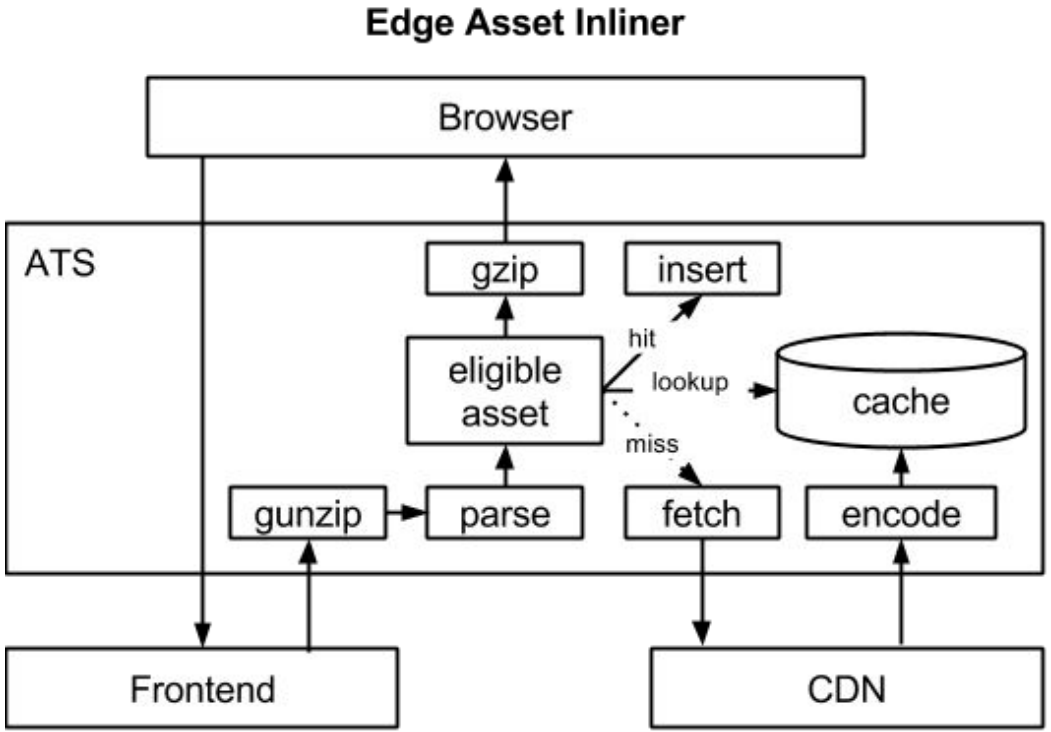


Image Inliner

- Parse response for asset URLs
- Fetch and cache assets from CDN
- Inline images (datauri)
- Lazy-inline mode
  - for cache hits replace with 1x1
  - insert base64 at bottom
- Base64 increases size by 30%
- HTTP/2 Server Push
  - browser can cancel stream if it doesn't need
  - no base64 overhead
  - faster by one roundtrip
  - **Requires new ATS plugin APIs**
- **Open Sourced**: **ETA Q1 2016**



Problem: unified configuration

- Routing configuration not consistent with origins
- Feature flag, application configuration, experimentation definition in many places
- Exponential complexity growth with number of dimensions
  - 16 dimensions = 200ms init cost
  - sparse graph
  - leads to large in-memory caches
- No existing syntax validation of configuration values
- Hard to audit which configurations are used

Zeus Configuration Compiler

- Scalable configuration system
- Goal is to have a unified configuration across all serving tiers
- Multiple configuration distribution options
  - **Option 1: configuration compiled into native code**
    - Languages: C++, Java, JavaScript, PHP, (we accept pull requests)
  - **Option 2: REST API served from ATS via a plugin**
    - REST responses are structured, versioned, and cachable JSON
    - 50kqps; 5ms @99th; 2 x Xeon E5-2430 (sandy-bridge); cache disabled
    - dynamically reload compiled shared library
  - **Option 3: subset of config available via HTTP request header**
    - allows coordination of configuration versions
- **Open Sourced Nov 2015**: https://github.com/yahoo/zeus

Areas of Improvement

- IPCs (Inter **Plugin** Communication)
  - repeated parsing request/responses
  - message passing standards
- Core regressions
- Execution ordering (multiple plug-ins sharing the same hook)
- Unified error handling
  - origin errors
  - internal errors
- Session hook access to SPDY/H2 data TS-3612
- SPDY/H2 server push plugin APIs
- Tracing through various hooks
- Validating plugin is memory leak free