Efficient HTTP Apis

A walk through http/2 via okhttp

adrian

engineer at Square

founded apache jclouds

focus on (small) libraries

* Can be blamed for the http/2 defects in okhttp!

okhttp

`HttpURLConnection` compatible

designed for java and android

BDFL: Jesse Wilson from square

https://github.com/square/okhttp

json apis are so simple

```
$ curl https://apihost/things \
-H 'SecurityToken: b08c85073c1a2d02' \
-H 'Accept: application/json'
[
  {
    "id": 1,
    "owner_id": 0,
    "name": "Able"
  },
...
  {
    "id": 26,
    "owner_id": 0,
    "name": "Zest"
  }
]
```

```
$ curl https://apihost/things/2 \
-H 'SecurityToken: b08c85073c1a2d02' \
-H 'Accept: application/json'
{
  "id": 2,
  "owner_id": 0,
  "name": "Beatific"
}
```

```
$ curl -X POST https://apihost/things \
-H 'SecurityToken: b08c85073c1a2d02' \
-H 'Content-Type: application/json' -d
'{
  "name": "Open-minded"
}'
```

```
$ curl -X DELETE https://apihost/things/2 \
-H 'SecurityToken: b08c85073c1a2d02'
```

so many bytes?!

```
$ curl https://apihost/things \
-H ‘SecurityToken: b08c85073c1a2d02’ \
-H ‘Accept: application/json’
[
  {
    "id": 1,
    "owner_id": 0,
    "name": "Able"
  },
...
  {
    "id": 26,
    "owner_id": 0,
    "name": "Zest"
  }
]
```

1642!

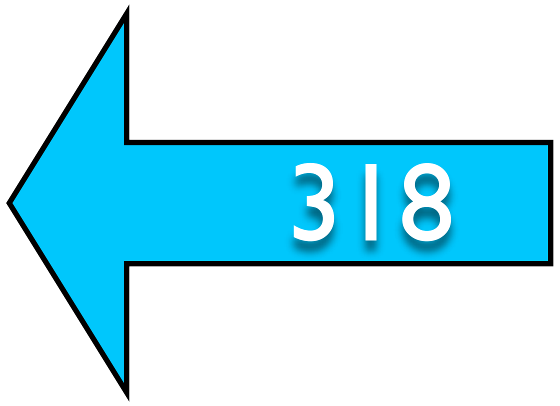
```
$ curl https://apihost/things/2 \
-H ‘SecurityToken: b08c85073c1a2d02’ \
-H ‘Accept: application/json’
{
  "id": 2,
  "owner_id": 0,
  "name": "Beatific"
}
```

```
$ curl -X POST https://apihost/things \
-H ‘SecurityToken: b08c85073c1a2d02’ \
-H ‘Content-Type: application/json’ -d
’{ "name": "Open-minded" }’
```

```
$ curl -X DELETE https://apihost/things/2 \
-H ‘SecurityToken: b08c85073c1a2d02’
```

now with gzip

```
$ curl --compress https://apihost/things \
-H ‘SecurityToken: b08c85073c1a2d02’ \
-H ‘Accept: application/json’
-H ‘Accept-Encoding: gzip’
[
  {
    "id": 1,
    "owner_id": 0,
    "name": "Able"
  },
...
  {
    "id": 26,
    "owner_id": 0,
    "name": "Zest"
  }
]
```



```
$ curl https://apihost/things/2 \
-H ‘SecurityToken: b08c85073c1a2d02’ \
-H ‘Accept: application/json’
{
  "id": 2,
  "owner_id": 0,
  "name": "Beatific"
}
```

```
$ curl -X POST https://apihost/things \
-H ‘SecurityToken: b08c85073c1a2d02’ \
-H ‘Content-Type: application/json’ -d
’{ "name": "Open-minded" }’
```

```
$ curl -X DELETE https://apihost/things/2 \
-H ‘SecurityToken: b08c85073c1a2d02’
```

java + gson

```
url = new URL(“https://apihost/things”);
connection = (HttpURLConnection) url.openConnection();
connection.setRequestProperty(“SecurityToken”, “b08c85073c1a2d02”);
connection.setRequestProperty(“Accept”, “application/json”);
connection.setRequestProperty(“Accept-Encoding”, “gzip”);
is = connection.getInputStream();

isr = new InputStreamReader(new GZIPInputStream(is));
things = new Gson().fromJson(isr, new TypeToken<List<Thing>>(){});
```

okhttp + gson

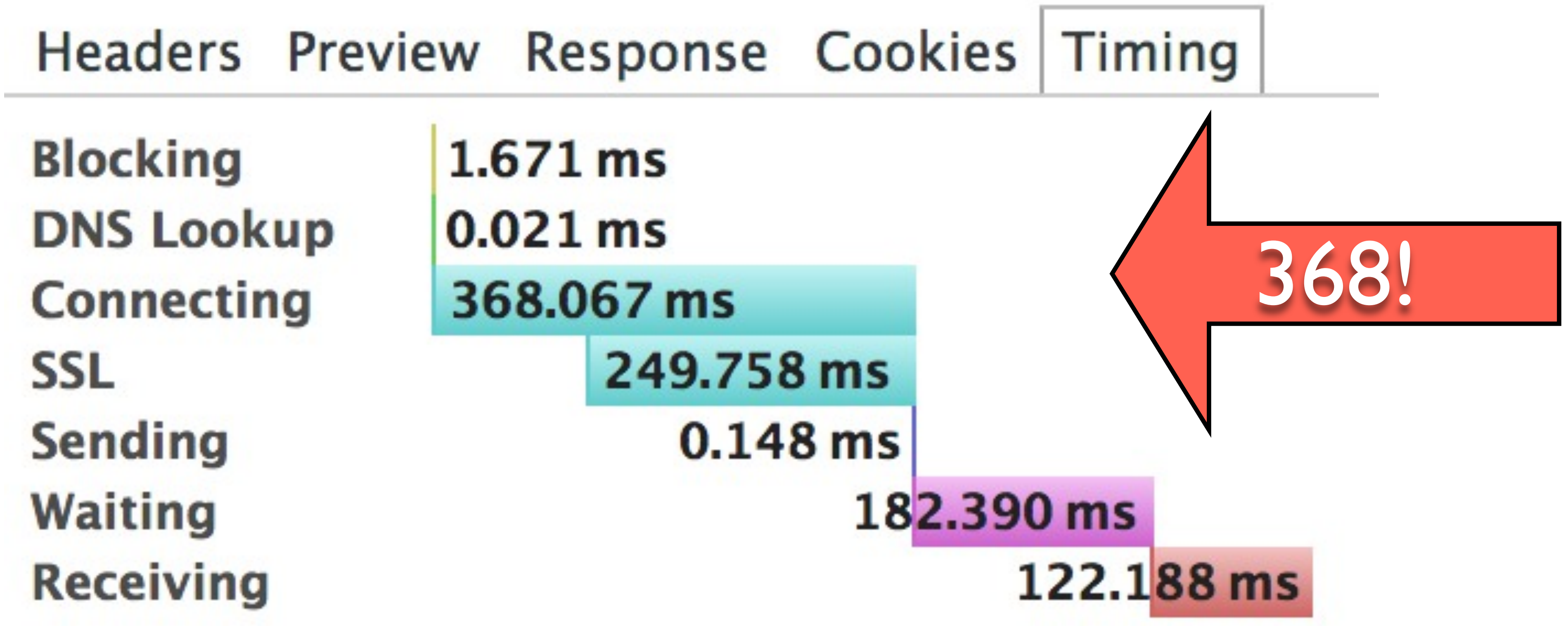
```
client = new OkHttpClient();
url = new URL(“https://apihost/things”);
connection = client.open(url);
connection.setRequestProperty(“SecurityToken”, “b08c85073c1a2d02”);
connection.setRequestProperty(“Accept”, “application/json”);
connection.setRequestProperty(“Accept-Encoding”, “gzip”);
is = connection.getInputStream();

isr = new InputStreamReader(is); // automatic gunzip
things = new Gson().fromJson(isr, new TypeToken<List<Thing>>(){});
```

OkHttp has its own api, but for portability, use java.net.HttpUrlConnection!

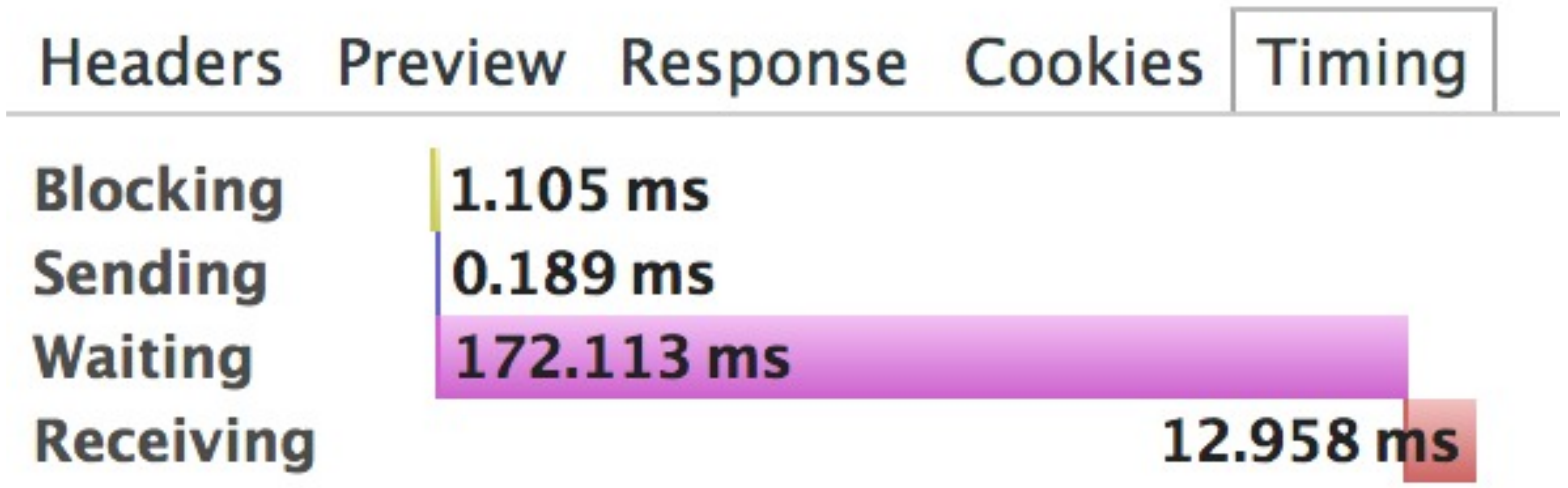
We won!

- List body reduced from 1642 to 318 bytes!
- We saved some lines using OkHttp
- Concession: cpu for compression, curl is a little verbose.

Hey.. List #1 is slow!



Blocking	1.671 ms
DNS Lookup	0.021 ms
Connecting	368.067 ms
SSL	249.758 ms
Sending	0.148 ms
Waiting	182.390 ms
Receiving	122.188 ms



Blocking	1.105 ms
Sending	0.189 ms
Waiting	172.113 ms
Receiving	12.958 ms

Ask Ilya why!

- TCP connections need 3-way handshake.
- TLS requires up to 2 more round-trips.
- Read High Performance Browser Networking

http://chimera.labs.oreilly.com/books/1230000000545

HttpUrlConnection

- `http.keepAlive` - should connections should be pooled at all? Default is true.
- `http.maxConnections` - maximum number of idle connections to each host in the pool. Default is 5.

- `get[Input|Error]Stream().close()` - recycles the connection, if fully read.
- `disconnect()` - removes the connection.

Don’t forget to read!

```
...

is = tryGetInputStream(connection);

isr = new InputStreamReader(is);
things = new Gson().fromJson(isr, new TypeToken<List<Thing>>(){});
...

InputStream tryGetInputStream(HttpUrlConnection connection)
  throws IOException {
try {
  return connection.getInputStream();
} catch (IOException e) {
  InputStream err = connection.getErrorStream();
  while (in.read() != -1); // skip
  err.close();
  throw e;
}
```

We won!

- Recycled socket requests are much faster and have less impact on the server.
- Concessions: must read responses, concurrency is still bounded by sockets.

Let's make List #2 free

Step 1: add a little magic to your server response

```
Cache-Control: private, max-age=60, s-maxage=0
Vary: SecurityToken
```

Step 2: make sure you are using a cache!

```
client.setOkResponseCache(new HttpResponseCache(dir, 10_MB));
...
connection.setDefaultUseCaches(true);
```

Step 3: pray your developers don't get clever

```
// TODO: stupid server sends stale data without this
connection = new URLConnection("https://apihost/things?time="
+ currentTimeMillis());
```

We won again!

- No time or bandwidth used for cached responses
- No application-specific cache ~~bugs~~ code.
- Concessions: only supports "safe methods", caching needs to be tuned.

http/1.1

rfc2616 - June 1999

text-based framing

defined semantics of the web

head-of-line blocking

http://www.w3.org/Protocols/rfc2616/rfc2616.html

spdy/3.1

google - Sep 2013

binary framing

retains http/1.1 semantics

concurrent multiplexed streams

http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3-1

http/2

ietf draft 09 - Dec 2013

binary framing

retains http/1.1 semantics

concurrent multiplexed streams

https://github.com/http2/http2-spec

http/2 headline features

- multiplexing
- header compression
- flow control
- priority
- server push

http/2 headline features

- multiplexing
- priority
- flow control
- header compression
- server push

Looking at the whole message

- Request: request line, headers, and body
- Response: status line, headers, and body

http/1.1 round-trip

GET /things HTTP/1.1

```
Host: apihost
SecurityToken: b08c85073c1a2d02
Accept: application/json
Accept-Encoding: gzip
```

HTTP/1.1 200 OK

```
Content-Length: 318
Cache-Control: private, max-age=60,
s-maxage=0
Vary: SecurityToken
Date: Sun, 02 Feb 2014 20:30:38 GMT
Content-Type: application/json
Content-Encoding: gzip
```

```
GZIPPED DATA
```

http/2 round-trip

HEADERS
Stream: 3
Flags: END_HEADERS, END_STREAM

```
:method: GET
:authority: apihost
:path: /things
securitytoken: b08c85073c1a2d02
accept: application/json
accept-encoding: gzip
```

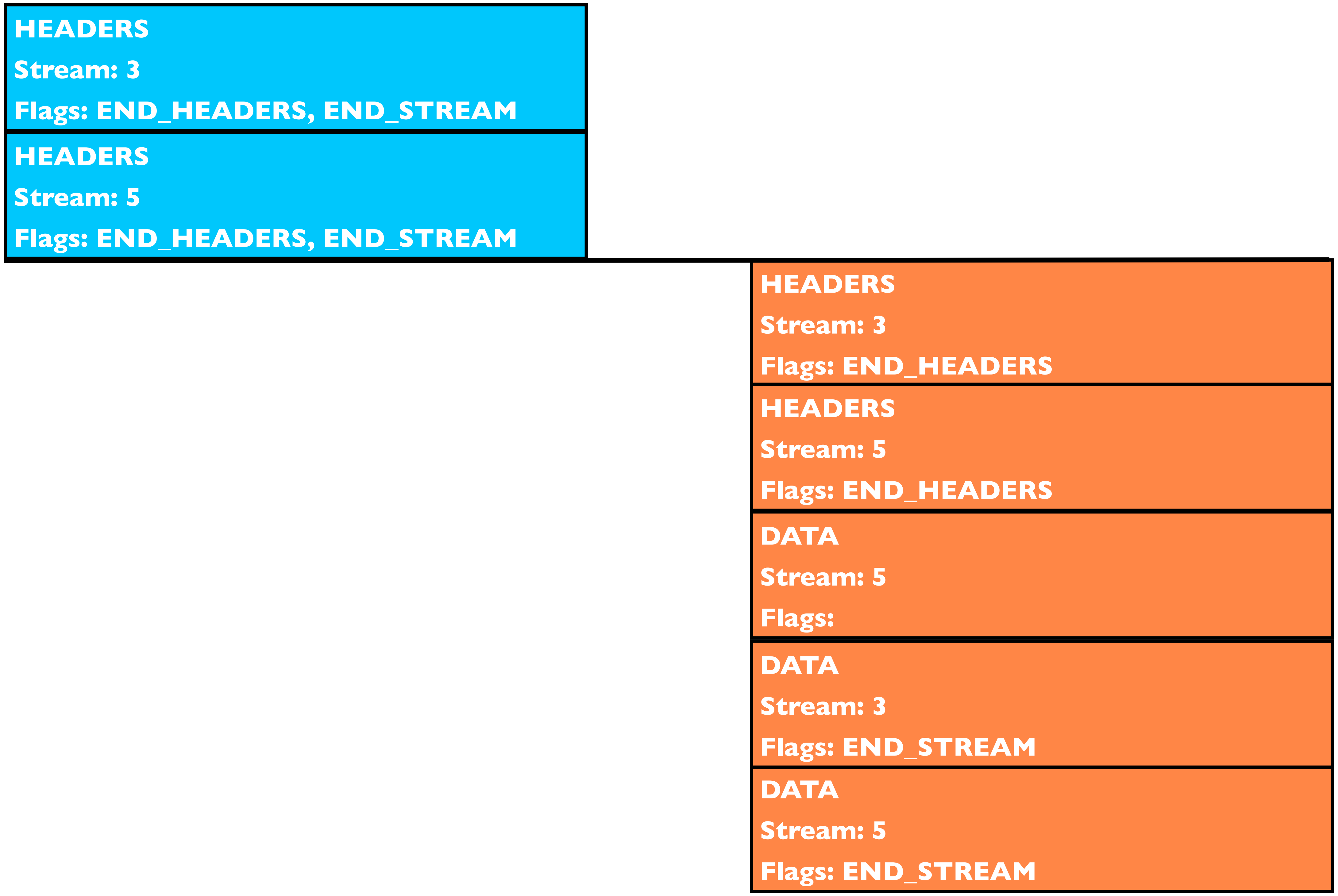
HEADERS
Stream: 3
Flags: END_HEADERS

```
:status: 200
content-length: 318
cache-control: private, max-age=60,
s-maxage=0
vary: SecurityToken
date: Sun, 02 Feb 2014 20:30:38 GMT
content-type: application/json
content-encoding: gzip
```

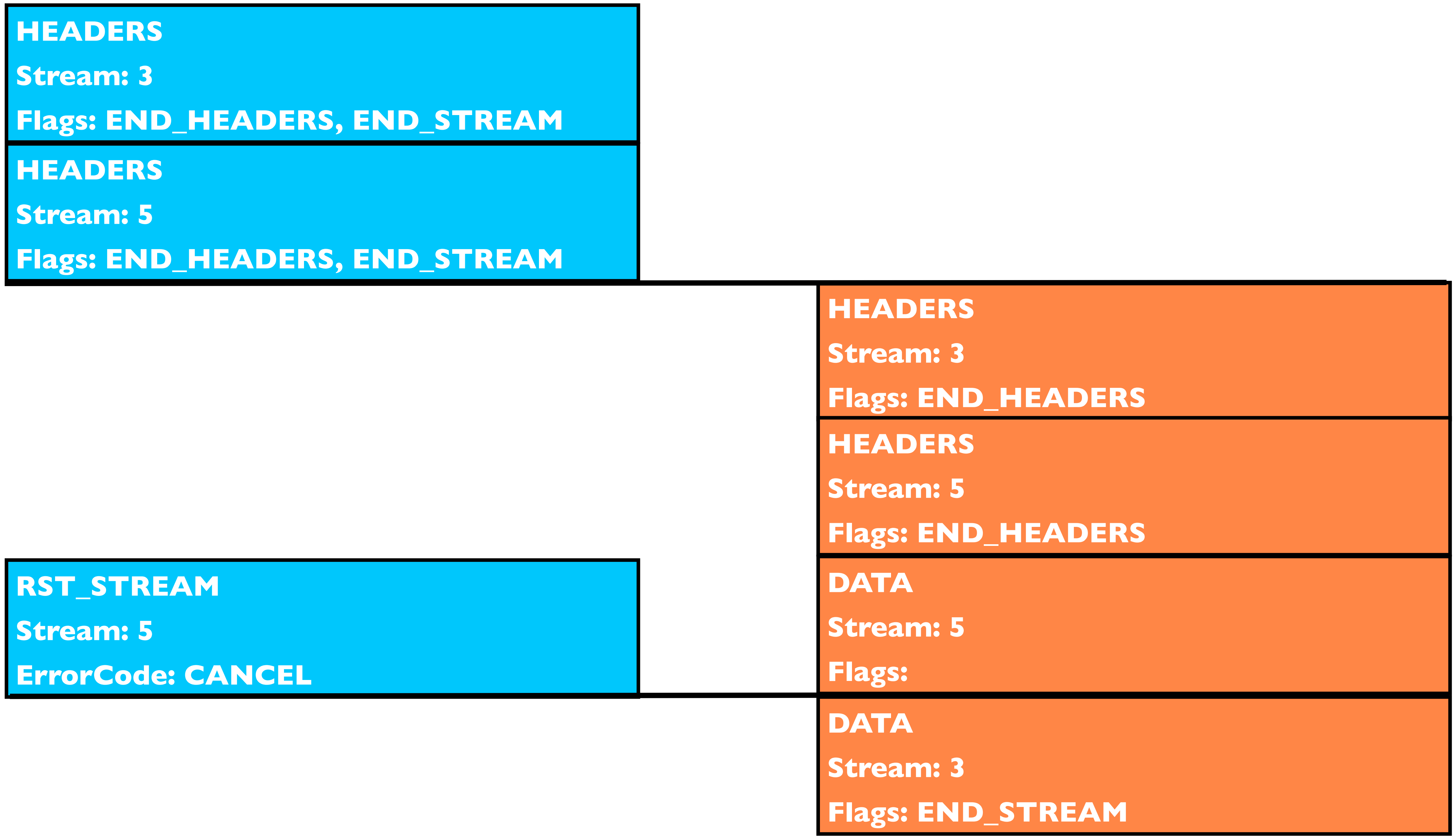
DATA
Stream: 3
Flags: END_STREAM

```
GZIPPED DATA
```

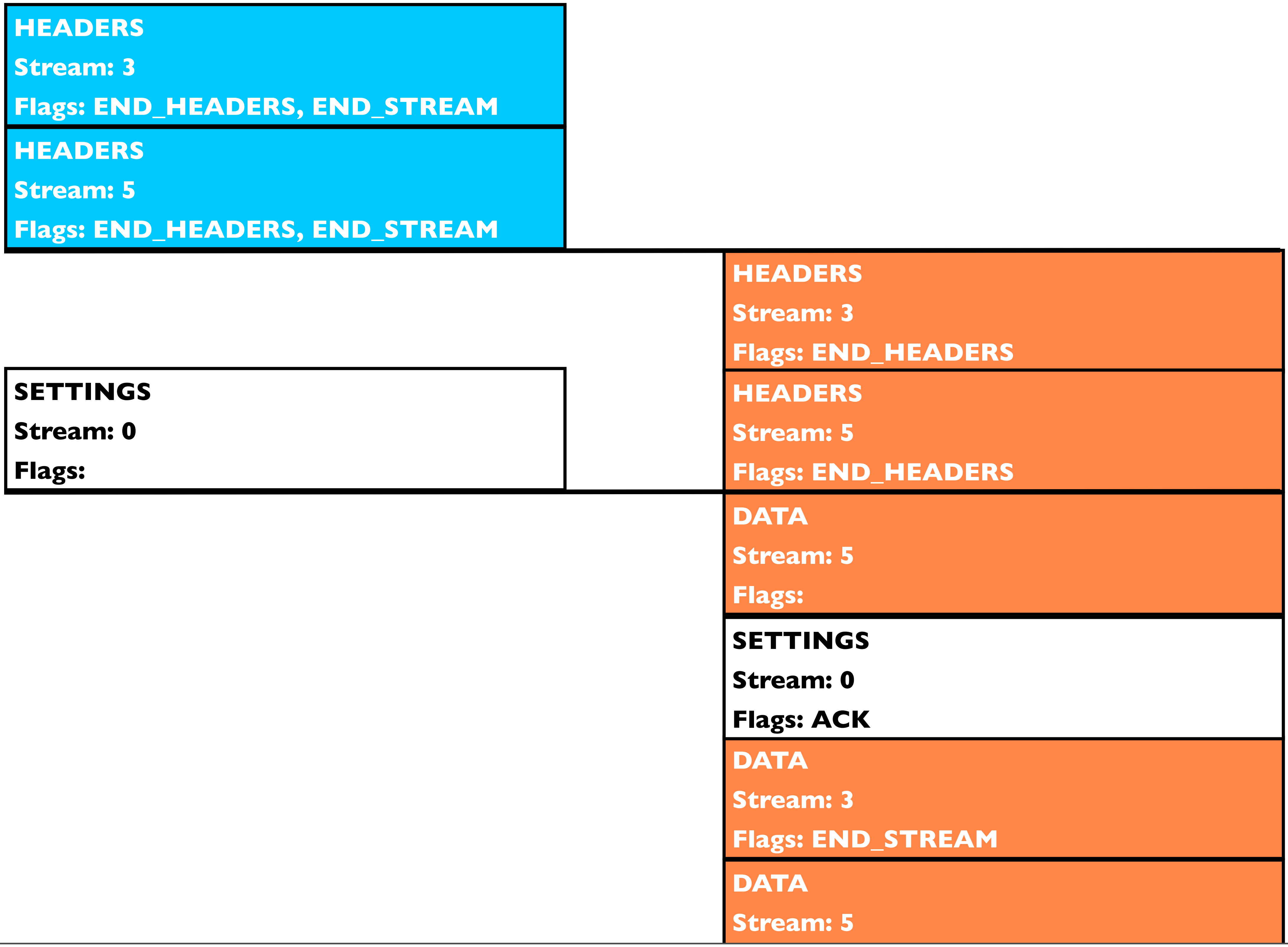
interleaving



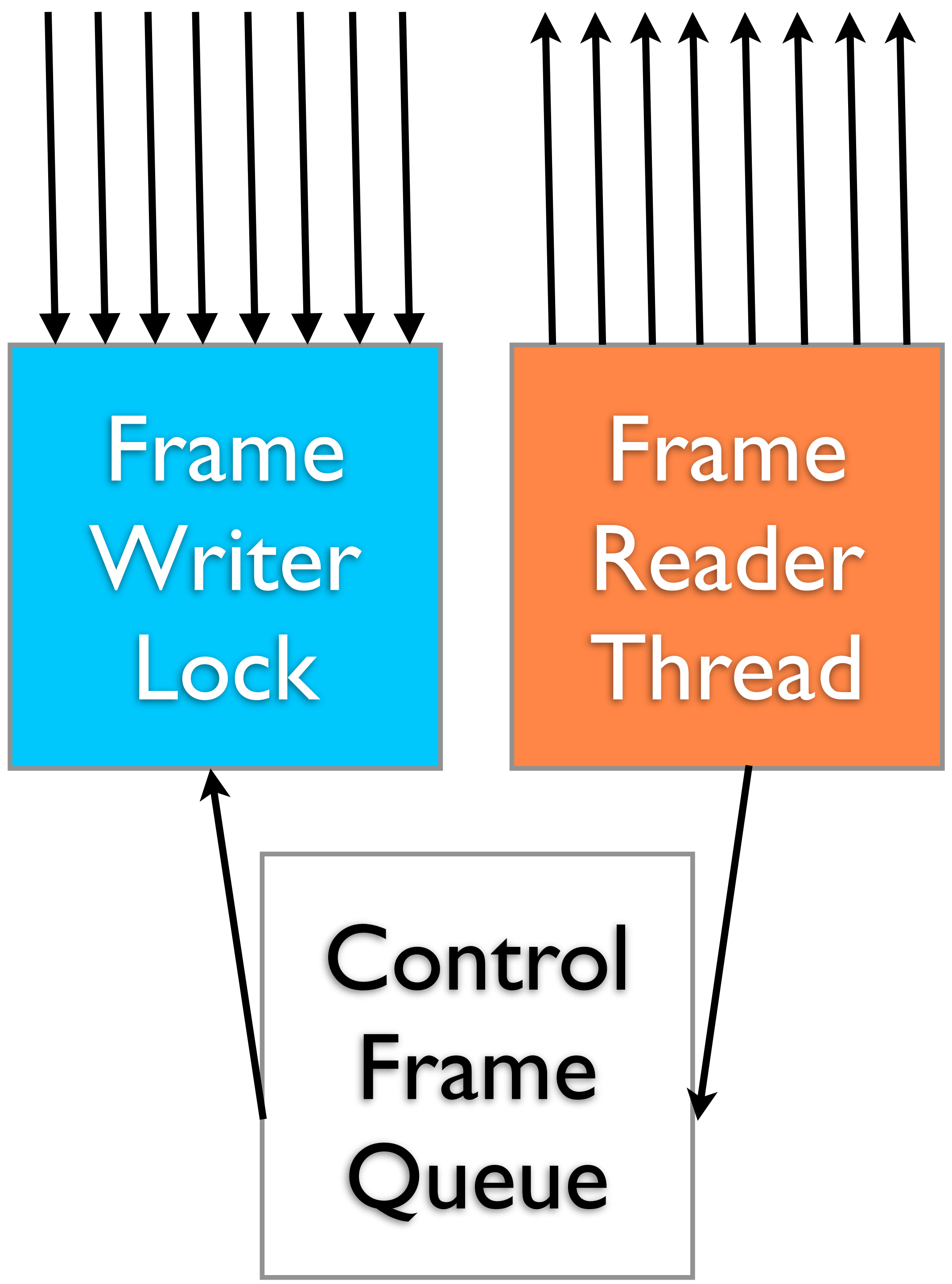
Canceling a Stream



control frames



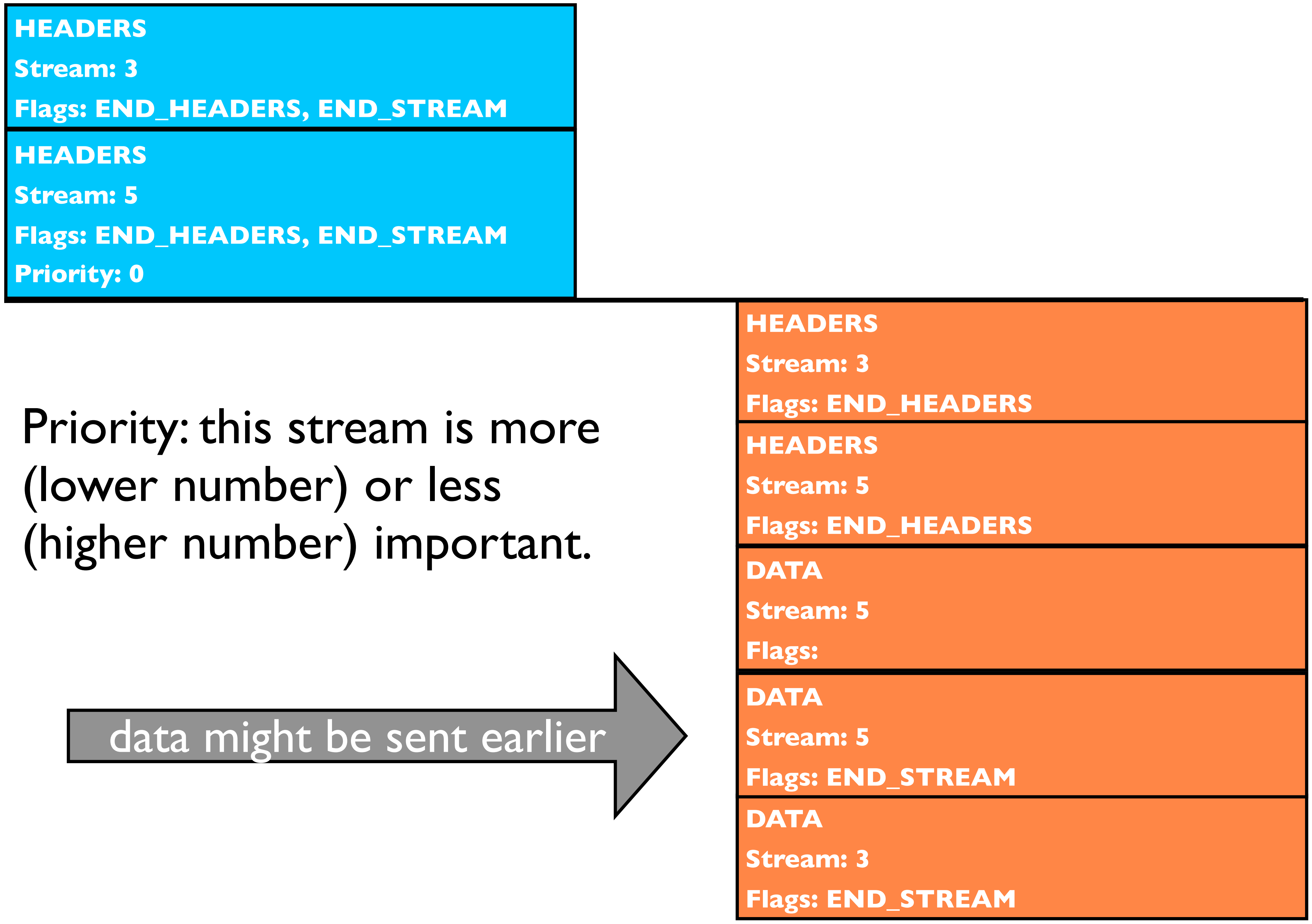
OkHttp/2 Architecture



http/2 headline features

multiplexing
priority
flow control
header compression
server push

priority



HEADERS
Stream: 3
Flags: END_HEADERS, END_STREAM

HEADERS
Stream: 5
Flags: END_HEADERS, END_STREAM
Priority: 0

HEADERS
Stream: 3
Flags: END_HEADERS

HEADERS
Stream: 5
Flags: END_HEADERS

DATA
Stream: 5
Flags:

DATA
Stream: 5
Flags: END_STREAM

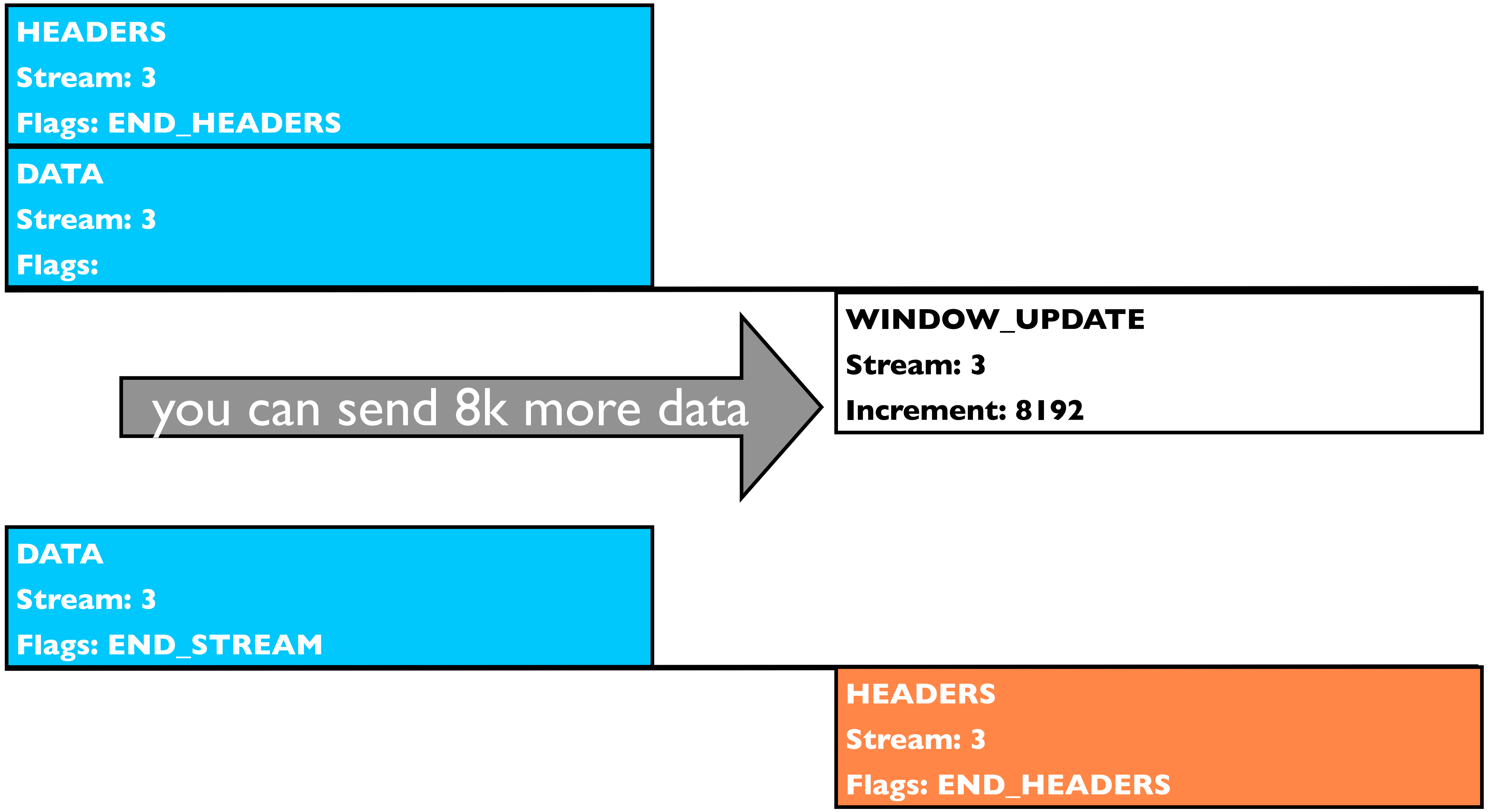
DATA
Stream: 3
Flags: END_STREAM

Priority: this stream is more (lower number) or less (higher number) important.

http/2 headline features

- multiplexing
- priority
- flow control
- header compression
- server push

flow control

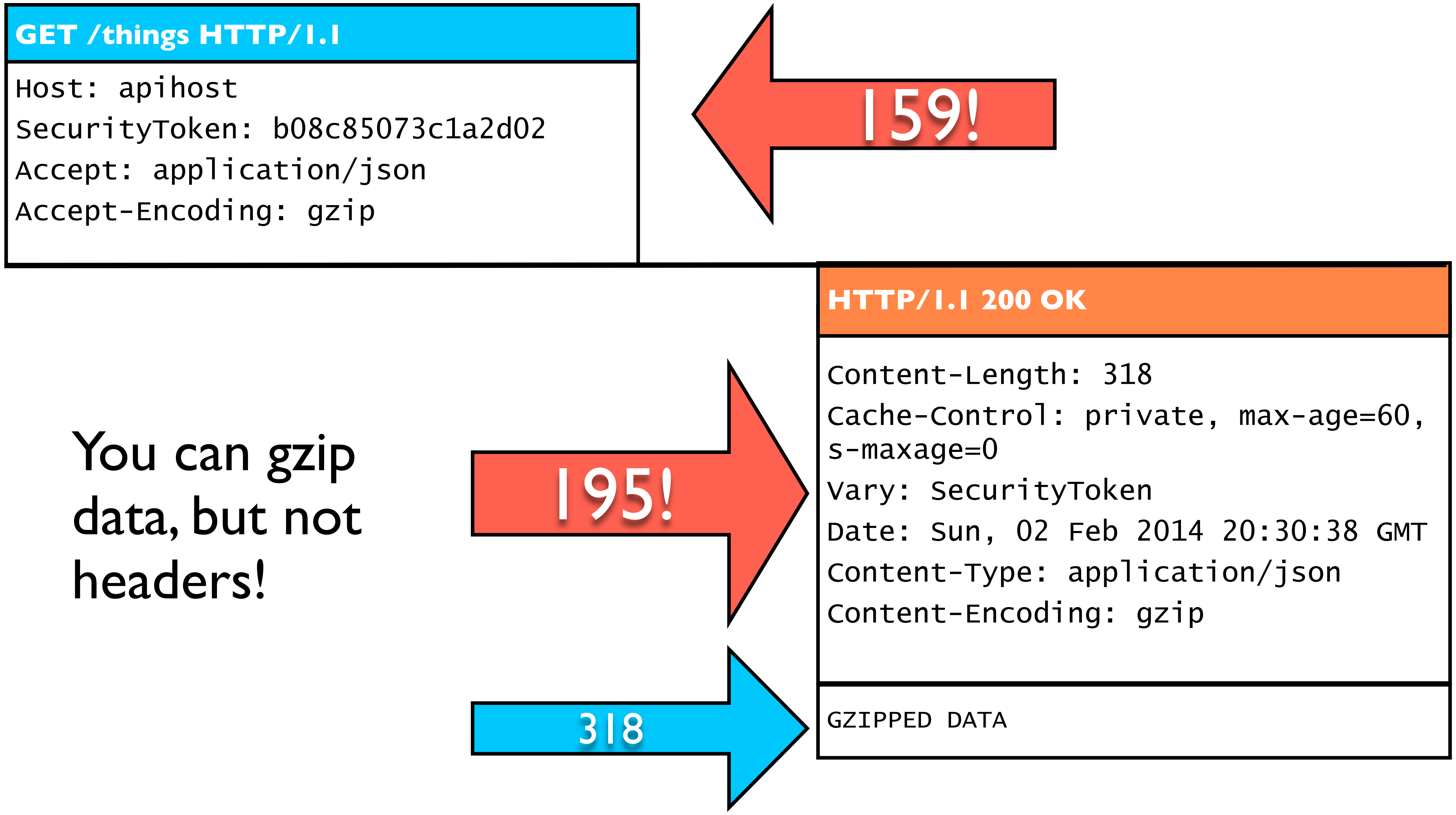


flow control: send up to the lesser of stream window and connection window (stream 0)

http/2 headline features

multiplexing
priority
flow control
header compression
server push

http/1.1 headers

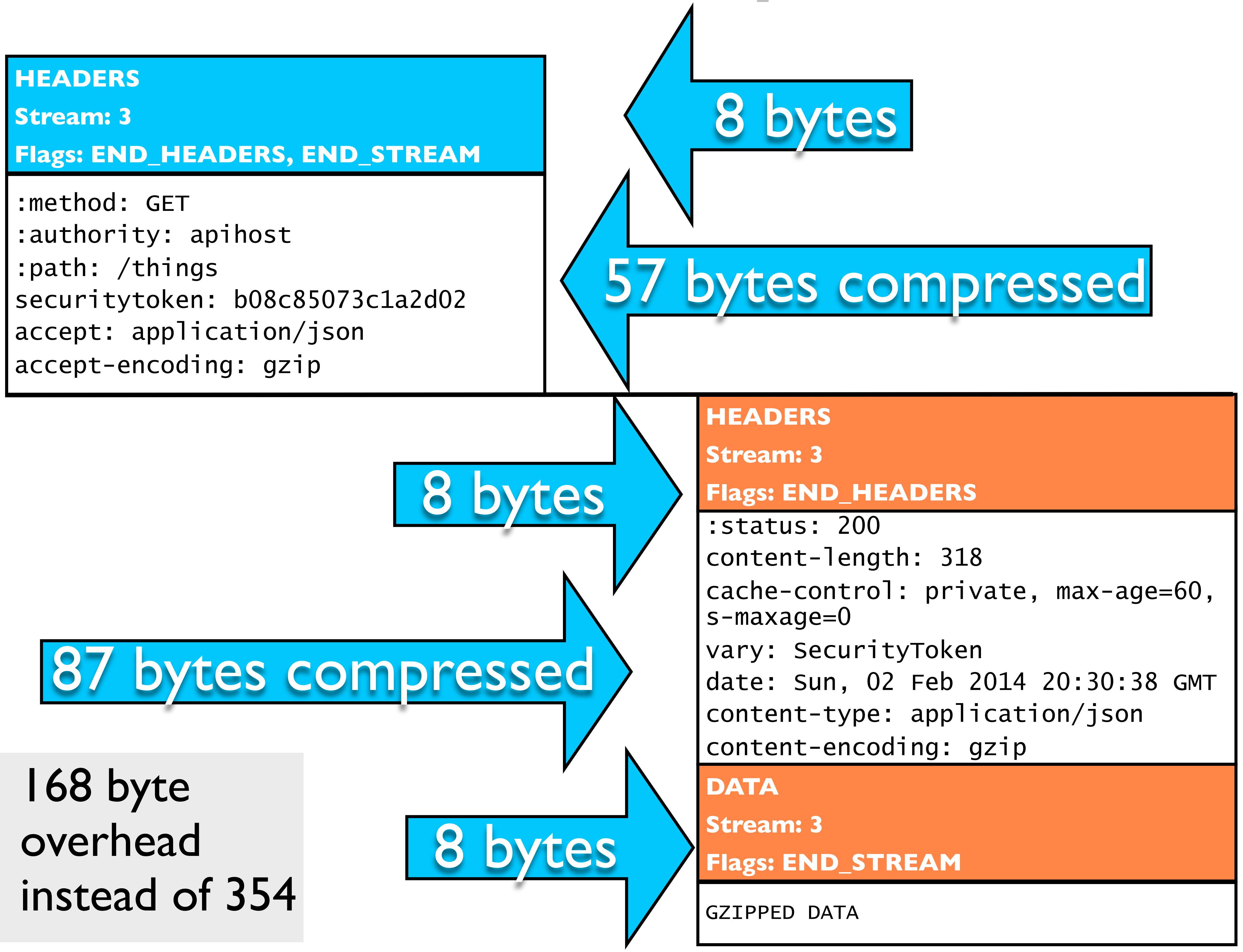


```
GET /things HTTP/1.1
Host: apihost
SecurityToken: b08c85073c1a2d02
Accept: application/json
Accept-Encoding: gzip
```

```
HTTP/1.1 200 OK
Content-Length: 318
Cache-Control: private, max-age=60,
s-maxage=0
Vary: SecurityToken
Date: Sun, 02 Feb 2014 20:30:38 GMT
Content-Type: application/json
Content-Encoding: gzip

GZIPPED DATA
```

You can gzip data, but not headers!

header compression



HEADERS
Stream: 3
Flags: END_HEADERS, END_STREAM

```
:method: GET
:authority: apihost
:path: /things
securitytoken: b08c85073c1a2d02
accept: application/json
accept-encoding: gzip
```

HEADERS
Stream: 3
Flags: END_HEADERS

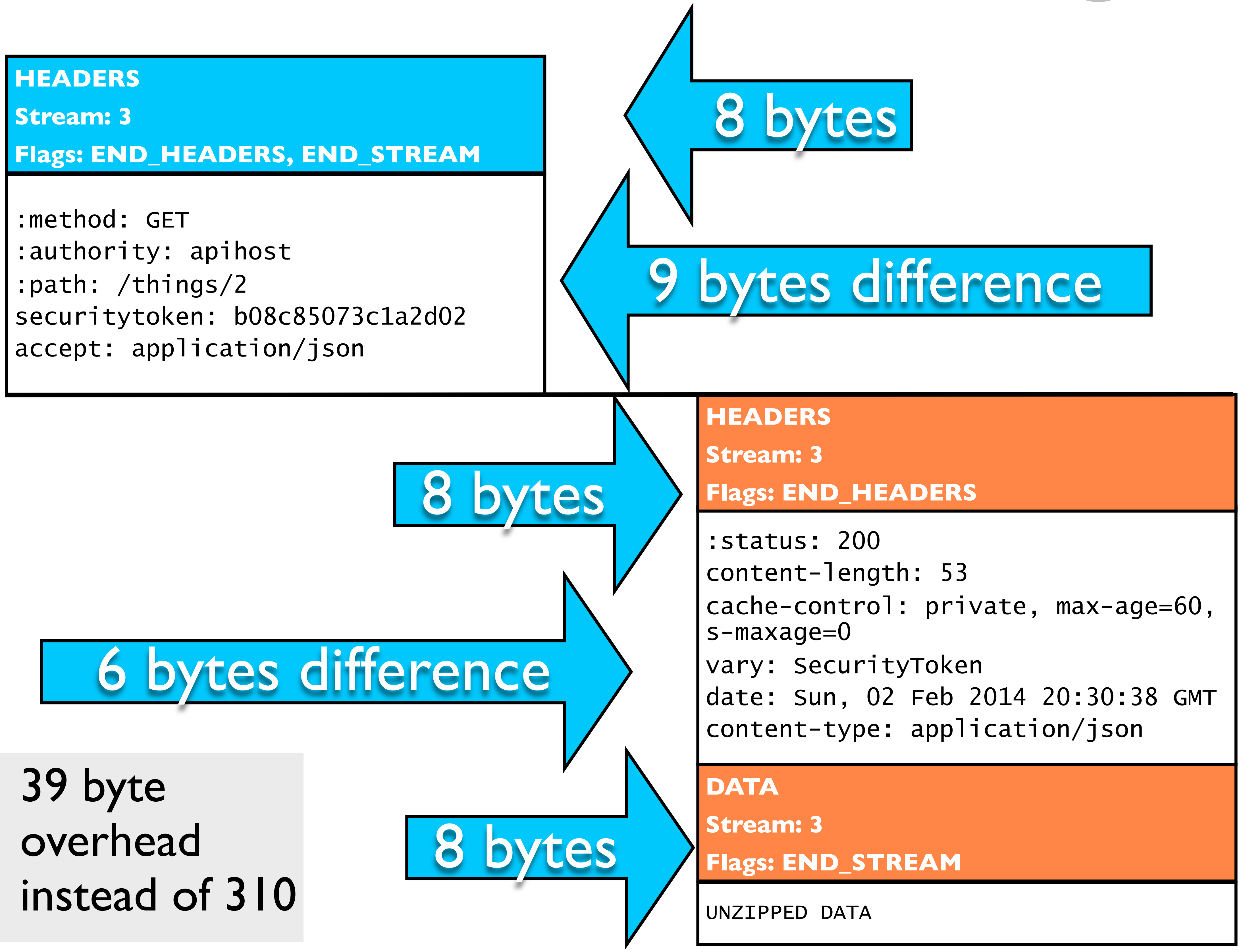
```
:status: 200
content-length: 318
cache-control: private, max-age=60,
s-maxage=0
vary: SecurityToken
date: Sun, 02 Feb 2014 20:30:38 GMT
content-type: application/json
content-encoding: gzip
```

DATA
Stream: 3
Flags: END_STREAM

```
GZIPPED DATA
```

- 168 byte overhead instead of 354

reference tracking



HEADERS
Stream: 3
Flags: END_HEADERS, END_STREAM

```
:method: GET
:authority: apihost
:path: /things/2
securitytoken: b08c85073c1a2d02
accept: application/json
```

HEADERS
Stream: 3
Flags: END_HEADERS

```
:status: 200
content-length: 53
cache-control: private, max-age=60,
s-maxage=0
vary: SecurityToken
date: Sun, 02 Feb 2014 20:30:38 GMT
content-type: application/json
```

DATA
Stream: 3
Flags: END_STREAM

```
UNZIPPED DATA
```

- 39 byte overhead instead of 310

hpack

ietf draft 05 - Dec 2013

static reference table

reference tracking

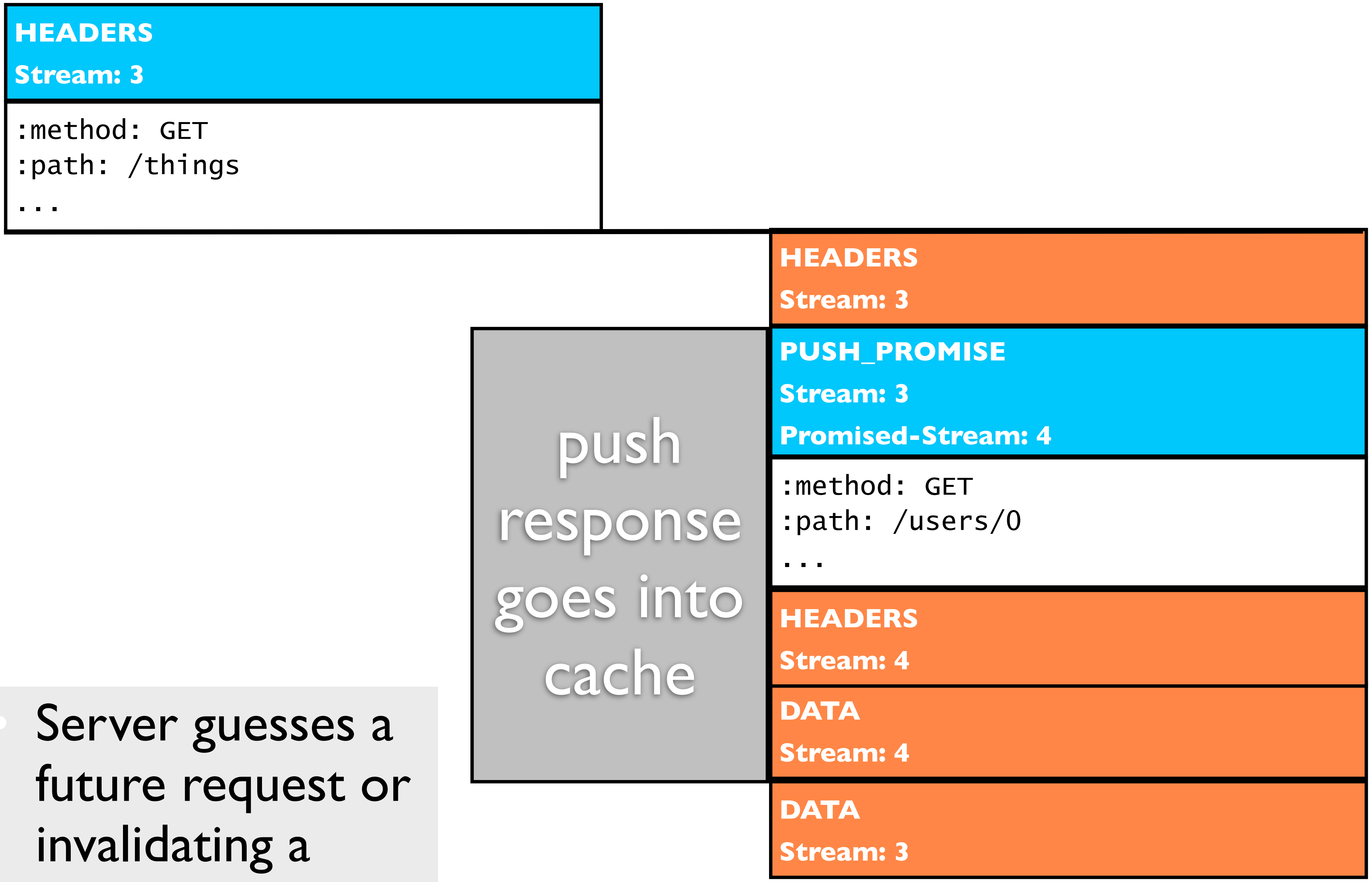
huffman encoding

http://tools.ietf.org/html/draft-ietf-httpbis-header-compression-05

http/2 headline features

- multiplexing
- priority
- flow control
- header compression
- server push

push promise



- Server guesses a future request or invalidating a cached resource

okhttp + http/2

OkHttp 2.0 supports http/2
on ssl connections.

Works out of the box on
Android.

For Java, add jetty's NPN
to your bootclasspath.

```
java -Xbootclasspath/p:/path/to/npn-boot-8.1.2.v20120308.jar ...
```

We won!

- 1 socket for 100+ concurrent streams.
- Advanced features like flow control, cancellation and cache push.
- Dramatically less header overhead.

Getting started

Connect to twitter or use OkHttp’s MockWebServer.

Note: OkHttp does not yet implement cache push or enforce priority settings.

https://github.com/square/okhttp

Engage!

- Implement http/2 in JVM web containers.
- Spread the word and get involved in http/2.
- Work on OkHttp with us!

https://github.com/http2/http2-spec/wiki/Implementations

https://github.com/square/okhttp

Thank you!

```
squareup.com/careers
corner.squareup.com
```

```
github square/okhttp
        @adrianfcole
```