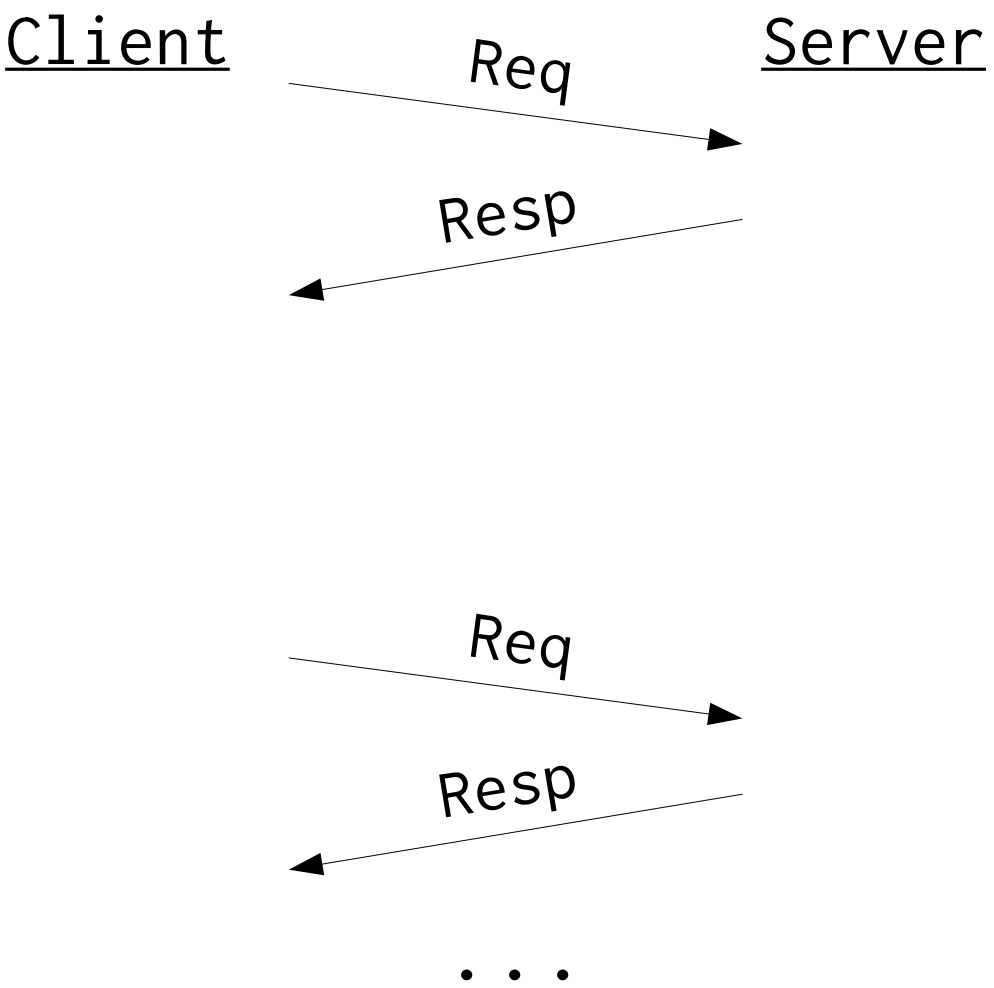
How I spent my summer vacation

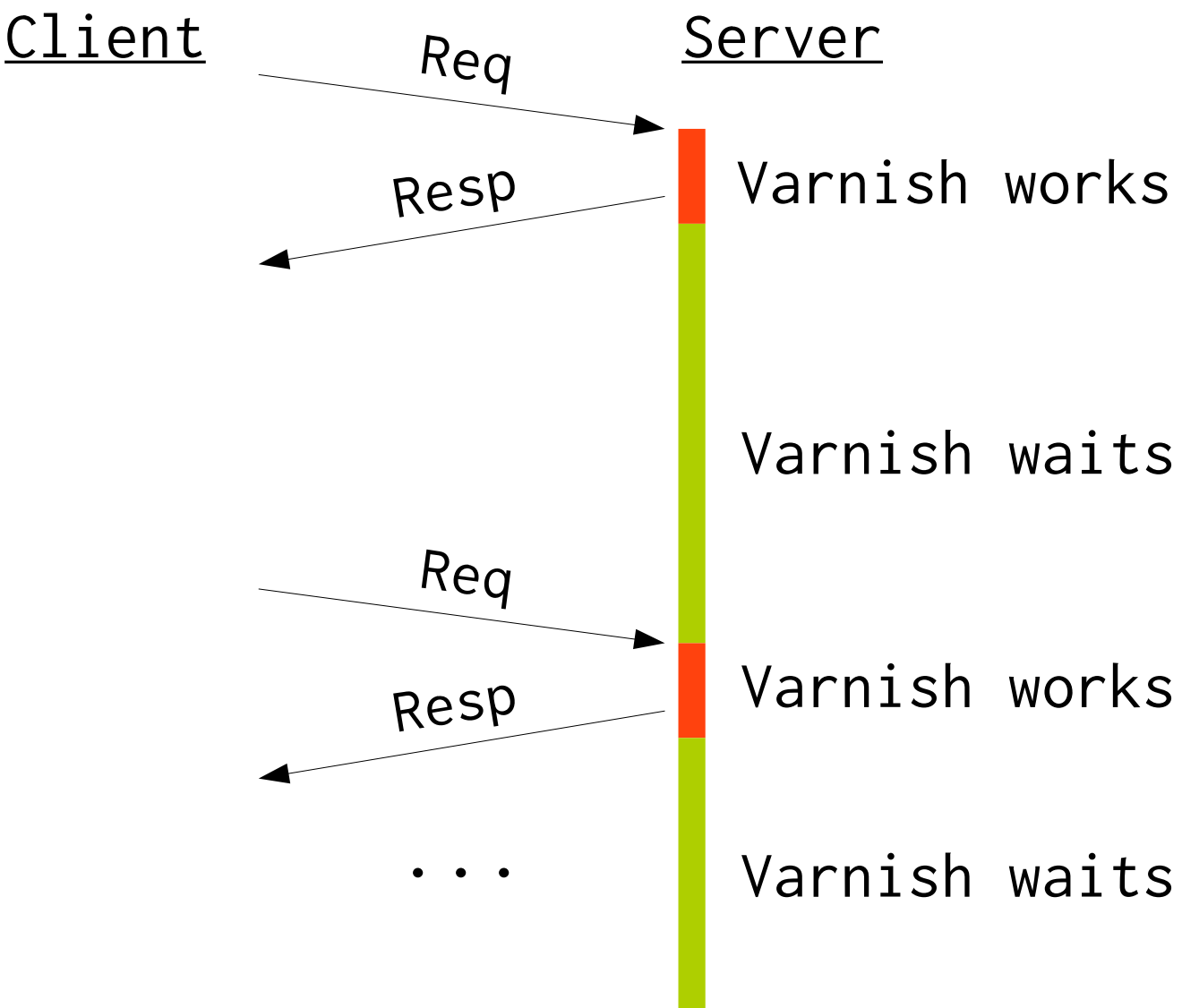
Poul-Henning Kamp

phk@FreeBSD.org

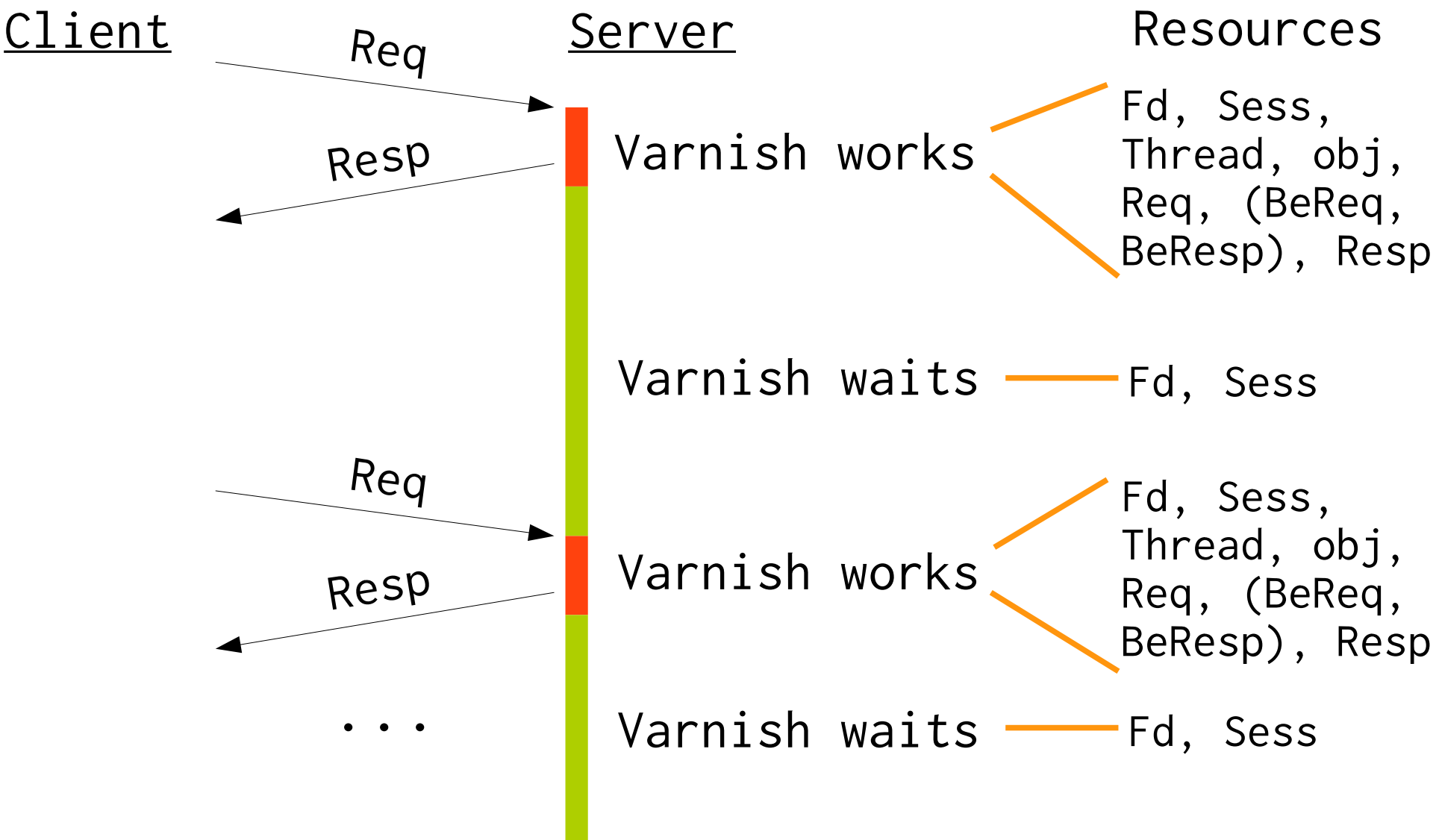
”HTTP is pure request->response”

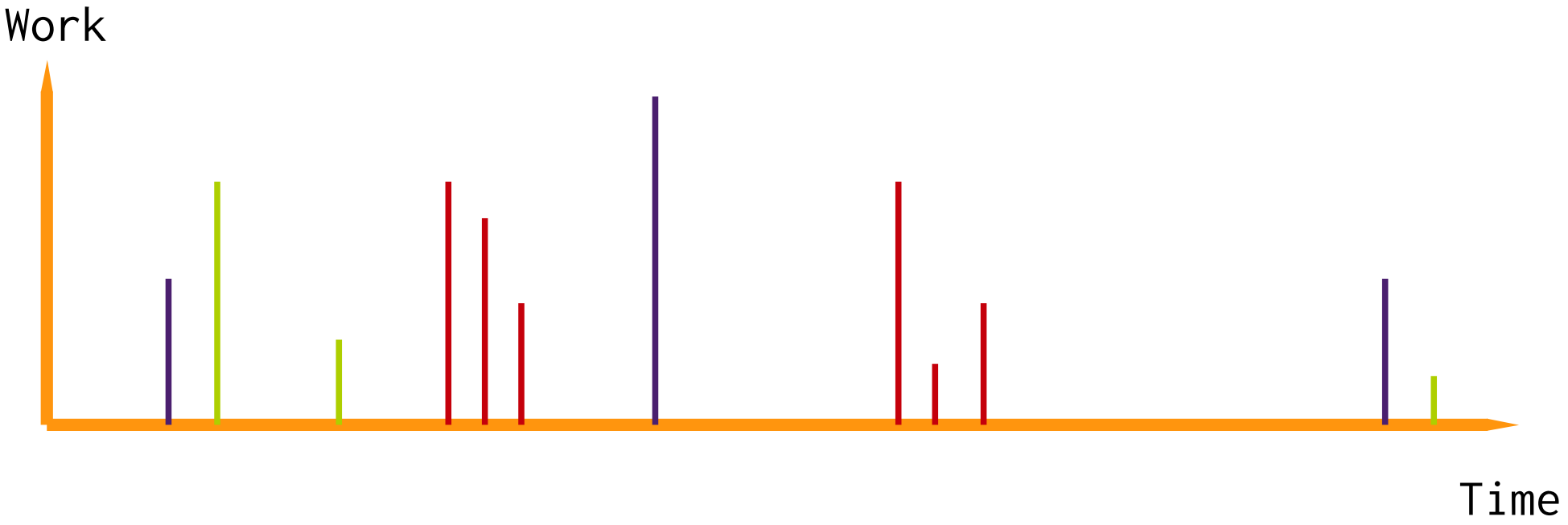


”HTTP is pure request->response”

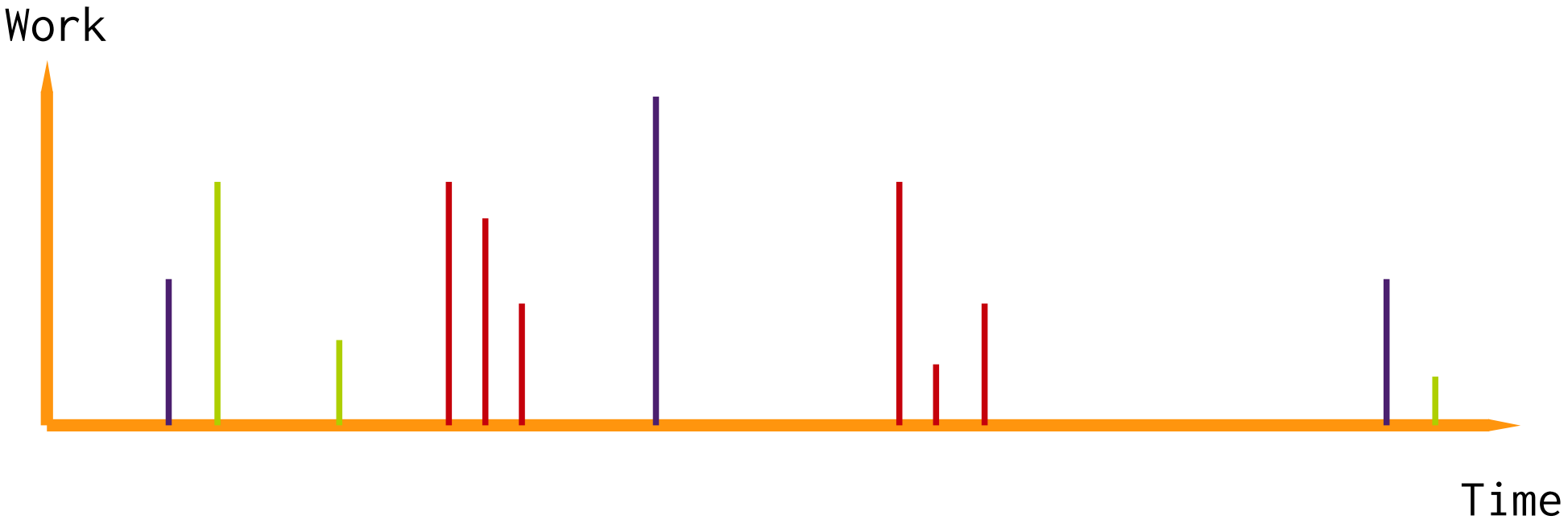


”HTTP is pure request->response”

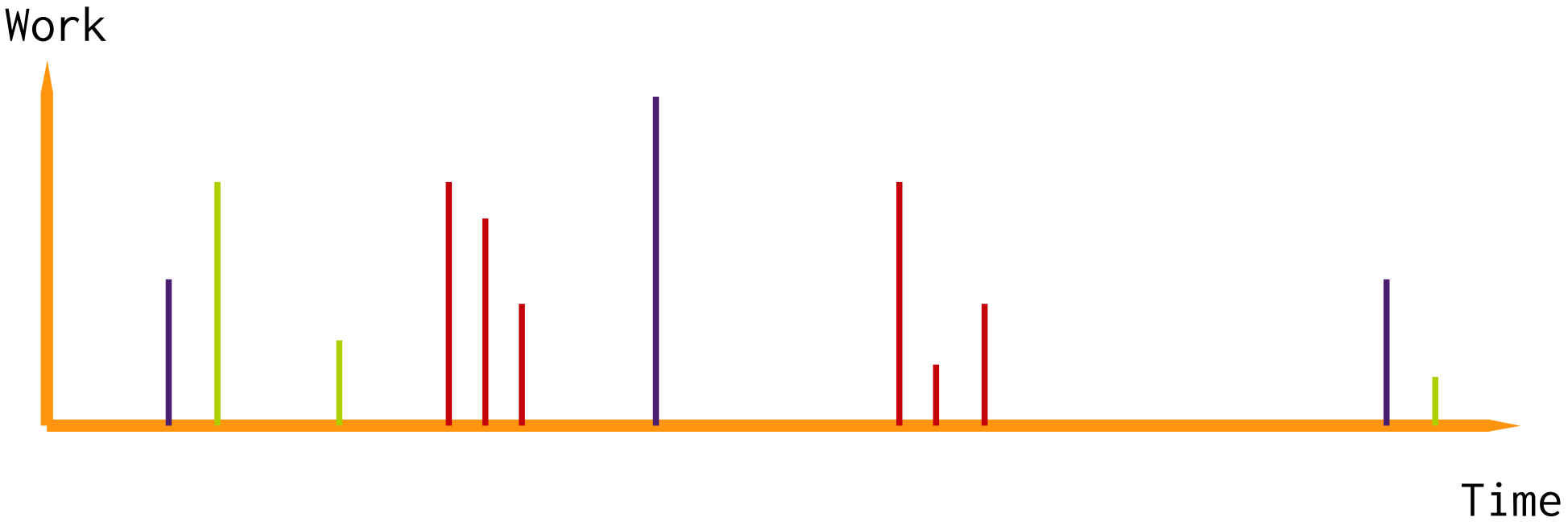



Work
Time

”High aspect ratio, farm friendly job”



”High aspect ratio, farm friendly job”



”On average, Varnish doesn't do s**t”
— sky

HTTP design scenario

```
Open TCP
GET dilbert.com::index.html
200 <some bytes>
Close TCP
<Render page>

Open TCP
GET dilbert.com::dilbert.19940622.gif
200 <some bytes>
Close TCP
<ReRender page>
```

HTTP todays scenario

```
Open TCP
GET dilbert.com::index.html
200 <some bytes>
GET dilbert.com::favicon.ico
200 <some bytes>
GET dilbert.com::bg_body.jpg
200 <some bytes>
GET dilbert.com::ico_new.png
200 <some bytes>
GET dilbert.com::bg_strip_header_featured.gif
200 <some bytes>
[42 other objects ]
```

HTTP todays scenario

	Cookies	½RTT's
Open TCP		2
GET dilbert.com::index.html	1	
200 <some bytes>		1
GET dilbert.com::favicon.ico	1	1
200 <some bytes>		1
GET dilbert.com::bg_body.jpg	1	1
200 <some bytes>		1
GET dilbert.com::ico_new.png	1	1
200 <some bytes>		1
GET dilbert.com::bg_strip_heac	1	1
200 <some bytes>		1
[42 other objects]	42	2*42

HTTP creativity boundary

Page render time is dominated by speed of light

→ 299,792,458 m/s: It's the law.

Supposedly unchangeable invariants:

→ TCP/IP
→ TCP port 80 (& 443)
→ HTML/CSS/JS model

HTTP/2.0 pipeline/mux

	Cookies	½RTT's
Open TCP		2
GET dilbert.com::index.html	1	
200 <some bytes>		1
GET dilbert.com::favicon.ico		1
GET dilbert.com::bg_body.jpg		
GET dilbert.com::ico_new.png		
GET dilbert.com::bg_strip_heac		ıre
[42 more gets]		
220 favicon.ico <some bytes>		1
220 ico_new.png <some bytes>		
220 bg_body.jpg <some bytes>		
220 bg_strip_header_featured.g		; b
[42 more 220's]		

HTTP/2.0 server push

	Cookies	½RTT's
Open TCP		2
GET dilbert.com::index.html	1	
200 <some bytes>		1
220 favicon.ico <some bytes>		
220 ico_new.png <some bytes>		
220 bg_body.jpg <some bytes>		
220 bg_strip_header_featured.g e b		
[42 other objects]		

HTTP/2.0 political wishlist

Mandatory SSL/TLS
   + End-to-end privacy
   - Kills all caches
   - Law mandates inspection (jails, kids …)
   → Killed, but a recurring zombie issue

Session-IDs
   → Trying to avoid cookies

Proxy HTTPS handling
   → How to do inspection on HTTPS

HTTP/2.0 technical wishlist

Compression
→ Really about cookies

Better protocol serialization
→ Faster, safer processing

UniCode
→ Really about avoiding encodings

HTTP/2.0 appearantly not about

HTTP → HTTPS upgrade cost
  → Connection upgrade would have been nice

Proxy mux efficiency
  → Mux secured and unsecured traffic

HTTP routers
  → Routable easily accessible envelope

Handling realistic future bandwidths
  → 10 years → 10Gb/s on laptops
  → 20 years → 100Gb/s on laptops

HTTP/2.0 — Because IPv6 was such a success

[RANT]

Shooting fish in a waterfall

HTTP/2.0 is an uncertain target

- → No realistic timeline
- → Very unclear featureset

- → May not even catch on when done

What does SPDY do in the meantime ?

What should Varnish do ?

Wouldn't it be nice...

What *could* Varnish do with/for HTTP/2.0 ?

→ Perfect for protocol migration
1.0 backend, 2.0 client or vice versa

→ Add server-push to HTTP/1.0 backend
VCL good for expressing policy & logic

```
sub vcl_recv {
   if (req.proto == ”HTTP/2.0” && req.url == ”/”) {
      push(”style.css”);
      push(”favicon.ico”);
      push(”fp.js”);
   }
}
```

Where are we ?

Varnish ≤ 3 heavily optimized for HTTP/1.1

Technical

- → Threads, pools, workspaces

Architectural

- → VCL request flow
- → VSL log record contents

Where do we want to end up ?

Multiple concurrent requests per session

Multiprotocol: HTTP/1.1, SPDY, HTTP/2.0 …

Decouple client & backend transactions

Separate thread does backend transaction

Introduce rendez-vous between threads

→ refcount/copy req.* ?
→ Original session may be gone before beresp.*
→ How does this look in VCL ?

Not a lot of work, but some pitfalls & tarpits

Decouple client & backend transactions

Separate thread does backend transaction

HTTP/1.1 Payoff:

+ cleaner streaming

+ parallel ESI fetches

+ no-delay grace mode

Multiprotocol

The protocol is what defines a session

→ VCL not protocol agnostic
   ie: setting ”Connection: Close” in vcl_pipe{}
   What does ”pipe” mean for HTTP/2.0 anyway ?

→ Per protocol VCL ?

→ Or VCL just does ”abstract” request ?

   Where does session trickery go then ?

Multiprotocol

The protocol is what defines a session

HTTP/1.1 session/protocol code:

- → accept connections (VCA)
- → receive & validate requests (HTTP)
- → feed in requests to be handled (HTTP1_FSM)
- → responses in abstract form (RSP)
- → serializes response onto connection (WRW)

… All this must be objectified
= Design OO-API

Multiprotocol

The protocol is what defines a session

HTTP/2.0 session/protocol code:

- → accept connections (VCA)
  Reuse mandatory (due to UPGRADE)
- → receive & validate requests (HTTP2)
  May be best with per-session rx-thread
- → feed in requests to be handled (HTTP2_FSM)
- → responses in abstract form (RSP)
- → serializes response onto connection (WRW)
  Probably better with a per-session thread
  rather than per-object thread.

Multiprotocol

The protocol is what defines a session

HTTP/1.1 Payoff:

… none really

Lots of work

But show stopper for multiprotocol support

… We might as well start early

VCL language design is the hard part

VCL is a ”domain specific language”

→ Strong expression of domain mechanics

→ Hide/Handle invariant details

→ Minimize boilerplate code

VCL language design is the hard part

VCL is also where users invest resources

→ Changes not welcome, unless they are very good

But VCL is why Varnish is popular

→ Must stay relevant and on top of game

VCL language vs. magic buttons

We are trending towards magic buttons:

do_gzip, do_gunzip, do_esi, always_miss

N² complexity of interaction
→ Hard to define
→ Hard for users to remember

Prefer programmatic model in VCL
→ Unfortunately leads to more VCL code

PURGE with magic buttons

```
sub vcl_recv {
   if (req.request == ”PURGE && client.ip ~ pacl) {
      set req.do_purge_all_variants;
   }
}
```

PURGE with semi-programmatic model:

```
sub vcl_recv {
   if (req.request == ”PURGE && client.ip ~ pacl) {
      return (lookup);
   }
}
sub vcl_lookup {
   if (req.request == ”PURGE) {
      purge_all_variants;
      return(synth(200, ”Purge done”));
   }
}
```

PURGE with full programmatic model:

```
sub vcl_lookup {
   if (req.request == ”PURGE) {
      for_variants {
         if (backend.default.healthy) {
            purge(obj);
         } else {
            set obj.ttl = 0s;
            set obj.grace = 10m;
         }
      }
      return(synth(200, ”Purge done”));
   }
}
```

And yeah, I'm not asking you to decide...

”One of each, please ?”

Finally:

A big thanks to the Varnish Moral Licensees:

- Varnish Software
- Globo
- UPLEX
- FaceBook

They pay me for working on Varnish