Why is Varnish Cache neat?

Who am I?

- Per Buer
- CTO @ Varnish Software
- Programmer
- Sysadmin

Varnish Software

- We sell Varnish Plus
  - Products (clustering, scalability, etc)
  - Support
  - Custom development
- Other software built on Varnish

What is Varnish?

- Cache it?
- Transform it?
- Reject?
- Auth? Authz?

Q: Why do we cache?

A: <40µs TTFB (vs 40ms)

Design

- A HTTP server with HTTP backend
- Threaded architecture
- Logs to shared memory - weird, right?

VCL

- Varnish Configuration Language
- Gets compiled into binary code (.so), loaded and run

Varnish doesn't support purging of content … out of the box

SOME ASSEMBLY REQUIRED

Purging content (1/2)

```
sub vcl_recv {
  if (req.method == "PURGE") {
    return (purge);
  }
}
```

Purging content (2/2)

```
acl purge {
  "localhost";
  "192.168.55.0"/24;
}

sub vcl_recv {
  if (req.method == "PURGE") {
    if (!client.ip ~ purge) {
      return(synth(405,"Not allowed."));
    }
  return (purge);
  }
}
```

Adding a “feature” to Varnish

Throttling hot linking

- Hotlinking is unlawfully using resources from other servers in your own content
- In this example we put a cap on the number of times per minute this can happen
- Leverages a VMOD - “vsthrottle” to add throttling

```
import vsthrottle;
(..)
if (req.url ~ "^/assets/" &&
  (req.http.referer !~
“^http://www.example.com/“) &&
vsthrottle.is_denied(req.url, 10, 60s) {
   return(error(403,“Hotlinking prohibited”);
  }
```

Things you should know

- Varnish will not cache content requested with cookies
- Solution: Strip the cookie
- Tip: The cookie VMOD makes this easy

```
import cookie;

sub vcl_recv {
        cookie.parse("cookie1: value1; cookie2: value2");
        cookie.filter_except("cookie1");
        // get_string() will now yield
        // "cookie1: cookie2: value2;";
}
```

More things to know

- Set-Cookie headers deactivate cookies
- Solution: Remove Set-Cookie or fix the backend

Grace mode

Grace mode

- Allows Varnish to server outdated content if new content isn't available
- Content will be refreshed asynchronously from the backend increasing performance

```
sub vcl_backend_response {
    set beresp.grace = 2m;
}
```

Opening the hood

```
sub vcl_hit {
    if (obj.ttl >= 0s) {
        // A pure unadultered hit, deliver it
        return (deliver);
    }
    if (obj.ttl + obj.grace > 0s) {
        // Object is in grace, deliver it
        // Automatically triggers a background fetch
        return (deliver);
    }
    // fetch & deliver once we get the result
    return (fetch);
}
```

Modifying grace semantics

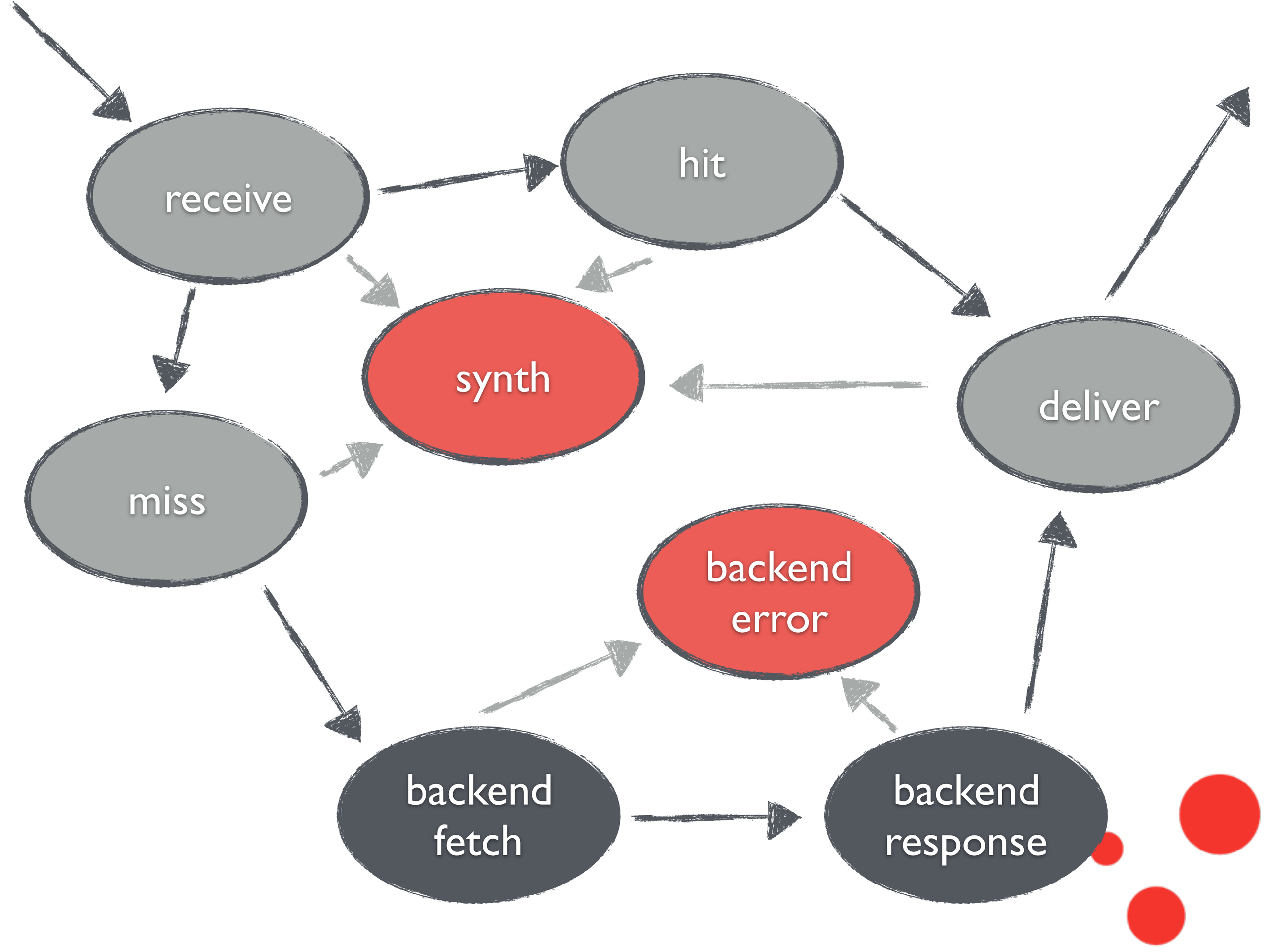
```
sub vcl_hit {
    if (obj.ttl >= 0s) {
        // A pure unadultered hit, deliver it
        return (deliver);
    }
    if (!std.healthy(req.backend_hint) &&
        (obj.ttl + obj.grace > 0s)) {
            return (deliver);
    }
    // fetch & deliver once we get the result
    return (fetch);
}
```

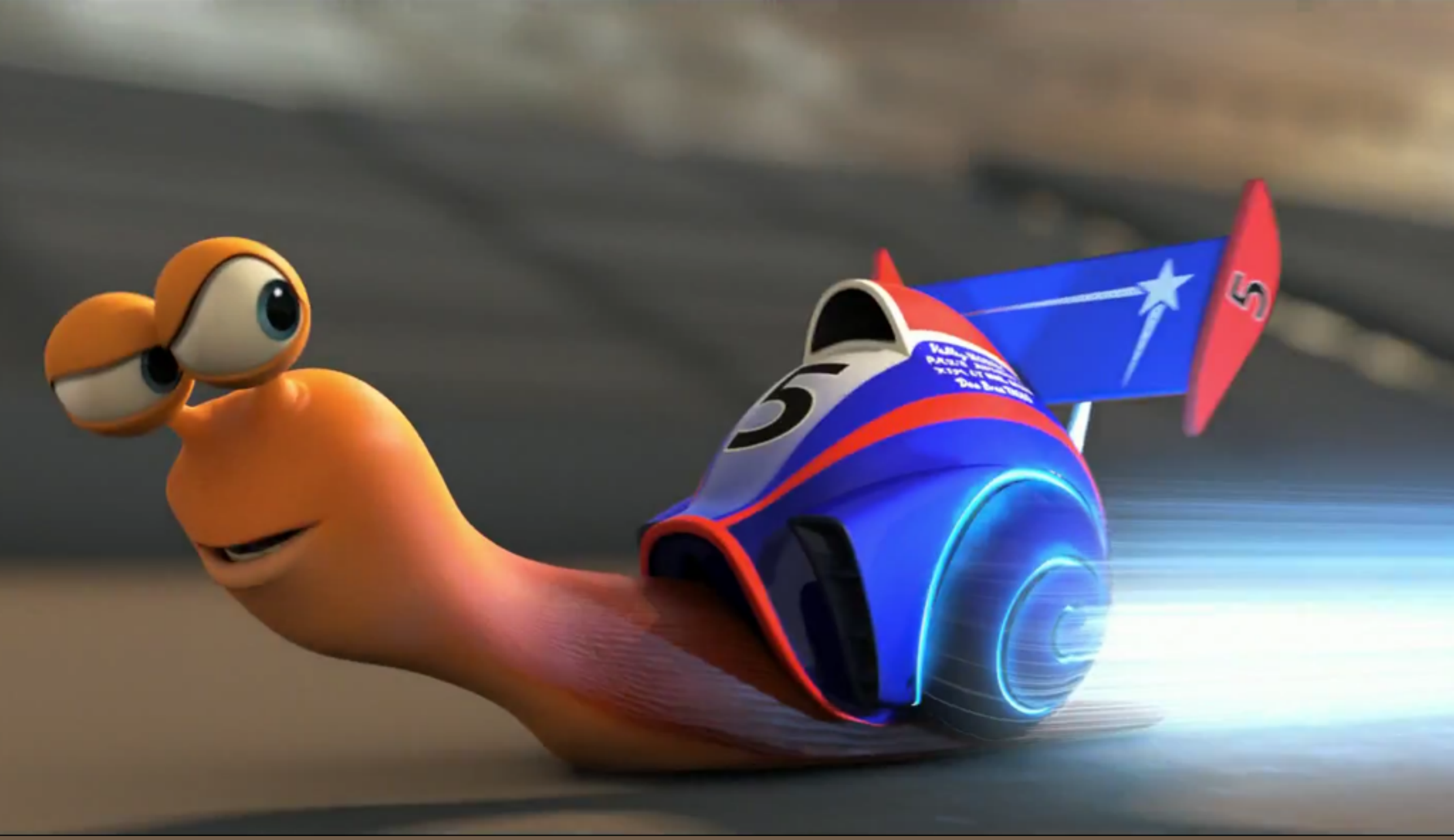
A couple of things you might wonder about…

What's beresp?

- req is the request object - use in vcl_recv
- bereq is the backend request object - use in vcl_backed_fetch
- beresp is the backend response - use in vcl_backend_response
- resp is the response object - use in vcl_deliver
- obj is the original object in memory - use in vcl_hit
- "`man(7) vcl`" for details

The state machine


receive
hit
synth
deliver
miss
backend error
backend fetch
backend response



What about tuning?

Quick guide to tuning on Linux

- Up somaxconn and tcp_max_syn_backlog
- Don’t mess with tcp_tw_recycle
- Be aware of workspaces
- Don’t do connection tracking
- Up the threads - 1req/sec per thread

Bonus content

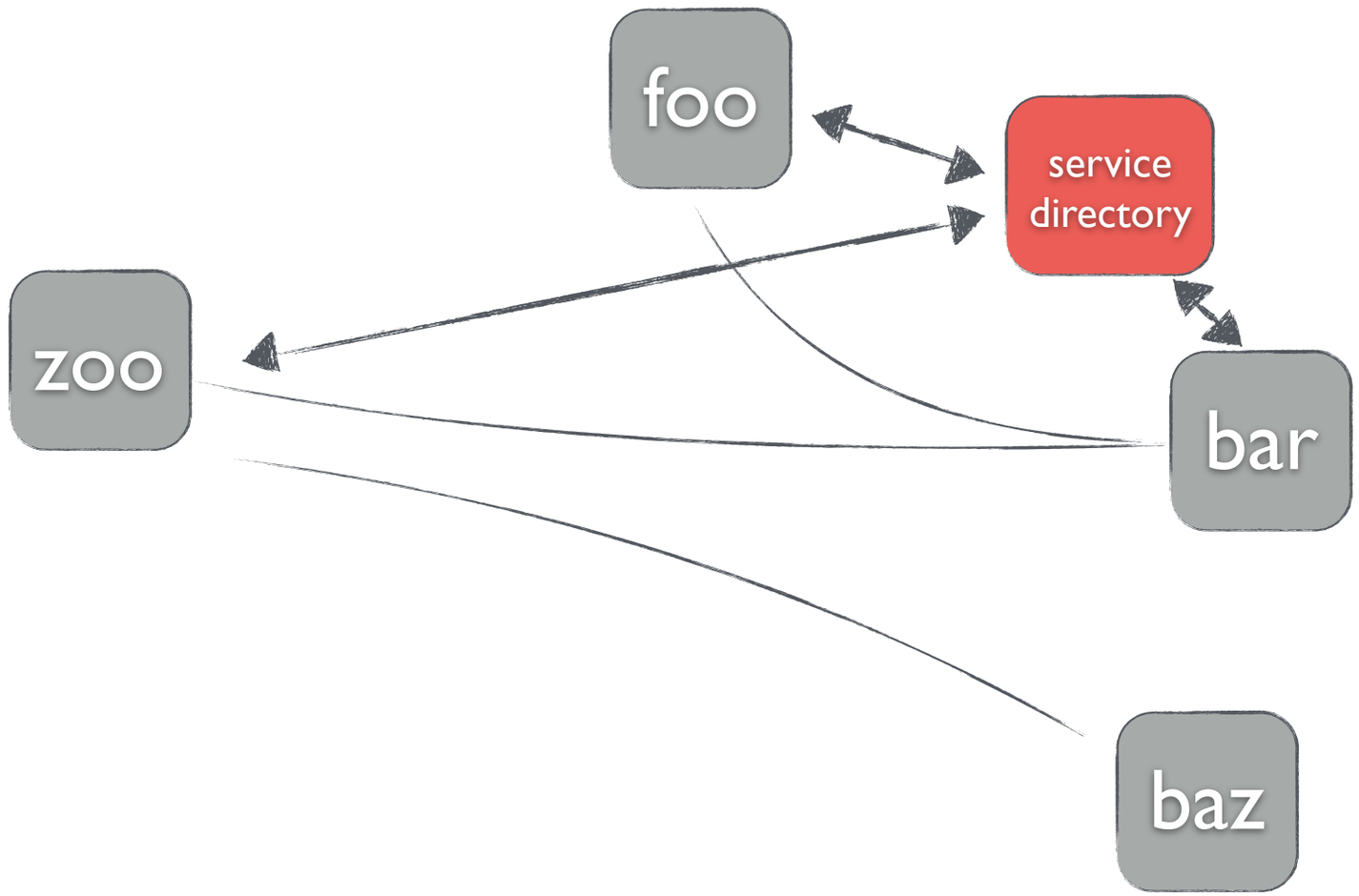
Redirection

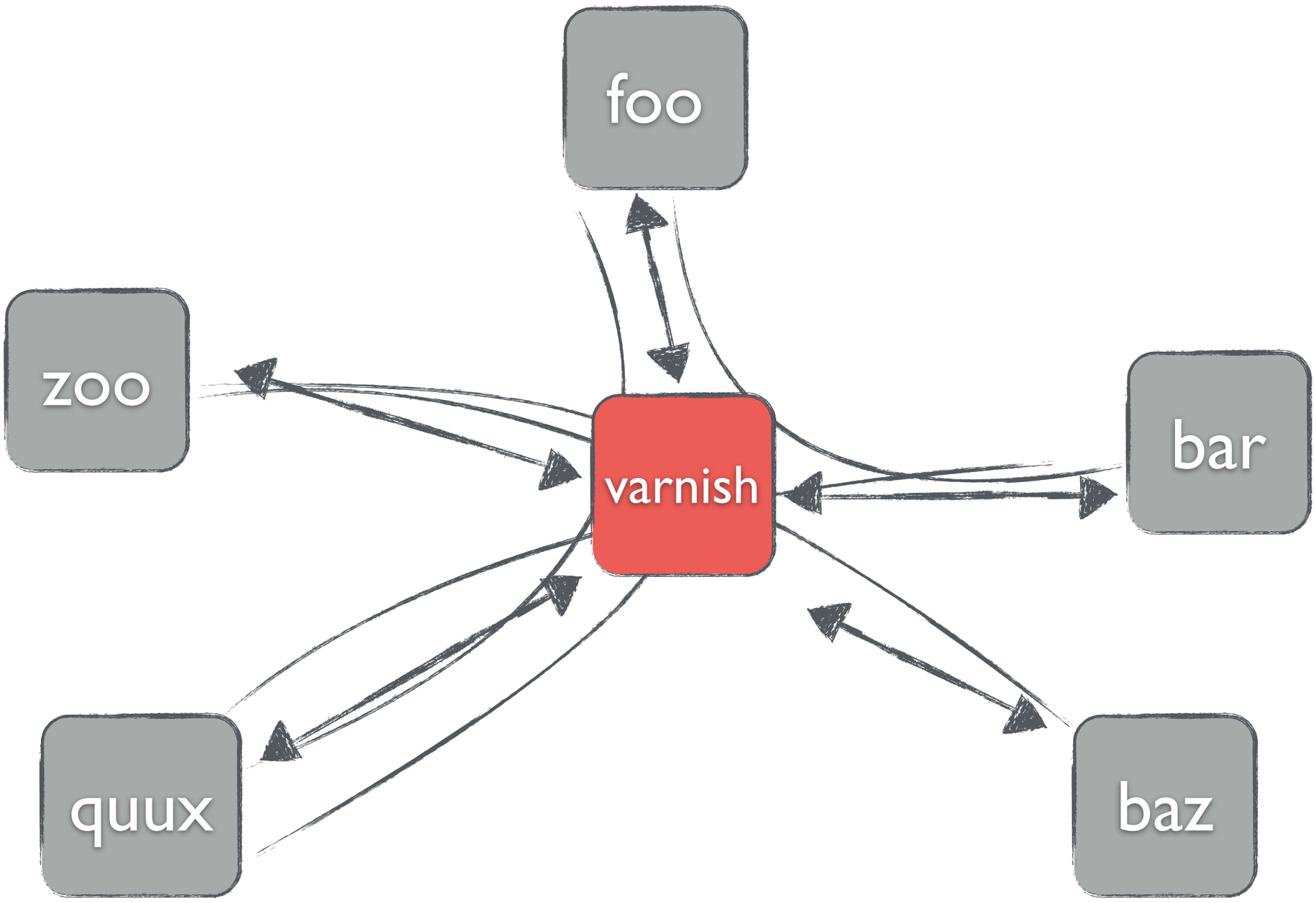
```
sub vcl_synth {
    if (resp.status == 750) {
        set resp.http.Location = "http://"
+ req.http.host + req.url;
        set resp.status = 301;
        return(deliver);
    }
}
```

```
# invoking a redirection
sub vcl_recv {
 if (req.http.host == "dev.example.com") {
   if (req.url ~ "^/archives/") {
     set req.url = regsub(req.url,
             "^/old/(.*)", "/archive/\1");
      set req.http.host = “example.com";
      return(synth(750, "Moved permanently"));
    }
  }
}
```

Ideas not covered in this talk

- shared memory logging in Varnish
- bans: asynchronous filter expressions to mass-invalidate based on arbitrary input
- “soft bans”: invalidate object but retain in memory
- auth/authz in VCL - cryptography
- playing with hashing vs Vary


foo
service directory
zoo
bar
baz


foo
zoo
varnish
bar
quux
baz

Thanks!

@perbu

@varnishcache