Using Varnish
or
VCL for webmasters

Poul-Henning Kamp

<phk@FreeBSD.org>

VCL Basics

Comments are 'C', 'C++' or Shell style:

```
# This is a comment

// So is this

/*
 * And this is also a comment
 */
```

VCL Basics

Strings are in ”...” and use %-escapes

```
”Hello World!%0a”

”\.jpg$”  // notice no \ to escape \

”strings”
   ”can be”
   ”continued”;  // (most places)
```

Inline C-code

Can be added between, and in functions.

Requires C-clue.

Can do almost anything.

```
C{
  printf(”Hello World\n”);
}C
```

Backends

Definition of servers to get content from

```
backend b1 { .host = ”10.0.0.1”; }

backend b2 { .host = ”10.0.0.2”; }

sub vcl_recv {
    // Default is first backend
    set req.backend b2;
}
```

Directors

Policy choice of backend.

```
backend b3 = { .host = ”b3”; }

director b2 random {
    { .backend = { .host = ”b1”; }
      .weight = 7; }
    { .backend = b3;
      .weight = 2; }
}
```

Using VCL code

At startup:

Specify backend (-b webhost1)

or

Specify VCL code (-f conf1.vcl)

Using (more) VCL code

From CLI:

vcl.load *config filename*
vcl.inline *config ”vcl program”*
vcl.use *config*
vcl.discard *config*
vcl.list
vcl.show *config*

Using (more) VCL code

Multiple VCL loaded at the same time.

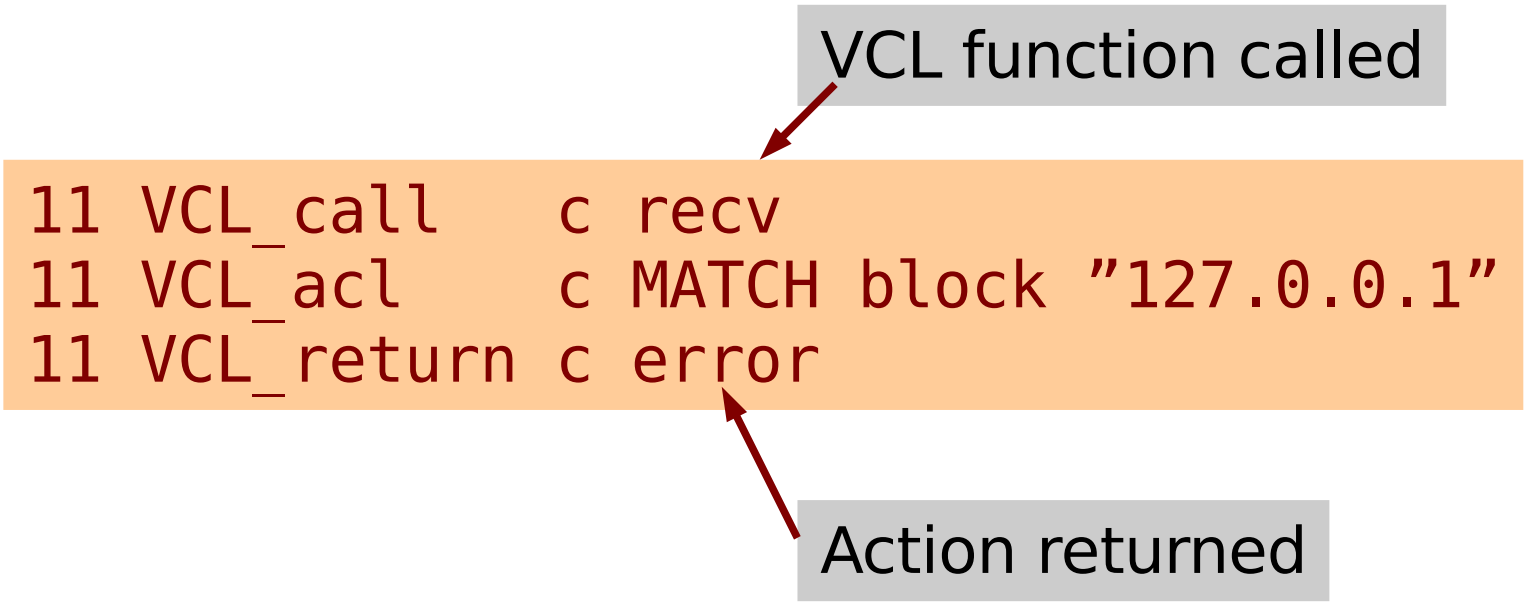
Change of VCL (vcl.use):
-> Instant
-> Does not invalidate cache
-> Affects only all new requests.

Situation/Load mitigation:

```
vcl.use emergency
vcl.use weekend
vcl.use damn_cnn
```

Debugging VCL code

Check varnishlog records.

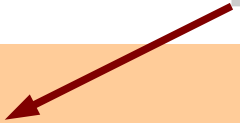


Debugging VCL code

Extra tracing controlled by parameter:

```
param.set vcl_trace on
```

Code-block = 1
Src-Line = 47
Src-Char = 14



```
11 VCL_call   c recv
11 VCL_trace  c 1 47.14
11 VCL_trace  c 2 48.13
11 VCL_acl    c MATCH block ”127.0.0.1”
11 VCL_trace  c 3.48.32
11 VCL_return c error
```

Debugging VCL code

If all else fails: study the compiled C-code:

varnishd -d -f foo.vcl -C

```
static int
VGC_function_vcl_recv (struct sess *sp)
{
  {
    {
      VRT_count(sp, 1);
      if
        (
          (
            (
              VRT_count(sp, 2),
              (
                VRT_strcmp(VRT_r_req_request(sp), "GET")
              )
            ) && (
              VRT_count(sp, 3),
              (
                VRT_strcmp(VRT_r_req_request(sp), "HEAD")
              )
[...]
```

```
sub vcl_recv {
    if (req.request != "GET" &&
        req.request != "HEAD" &&
[...]
```

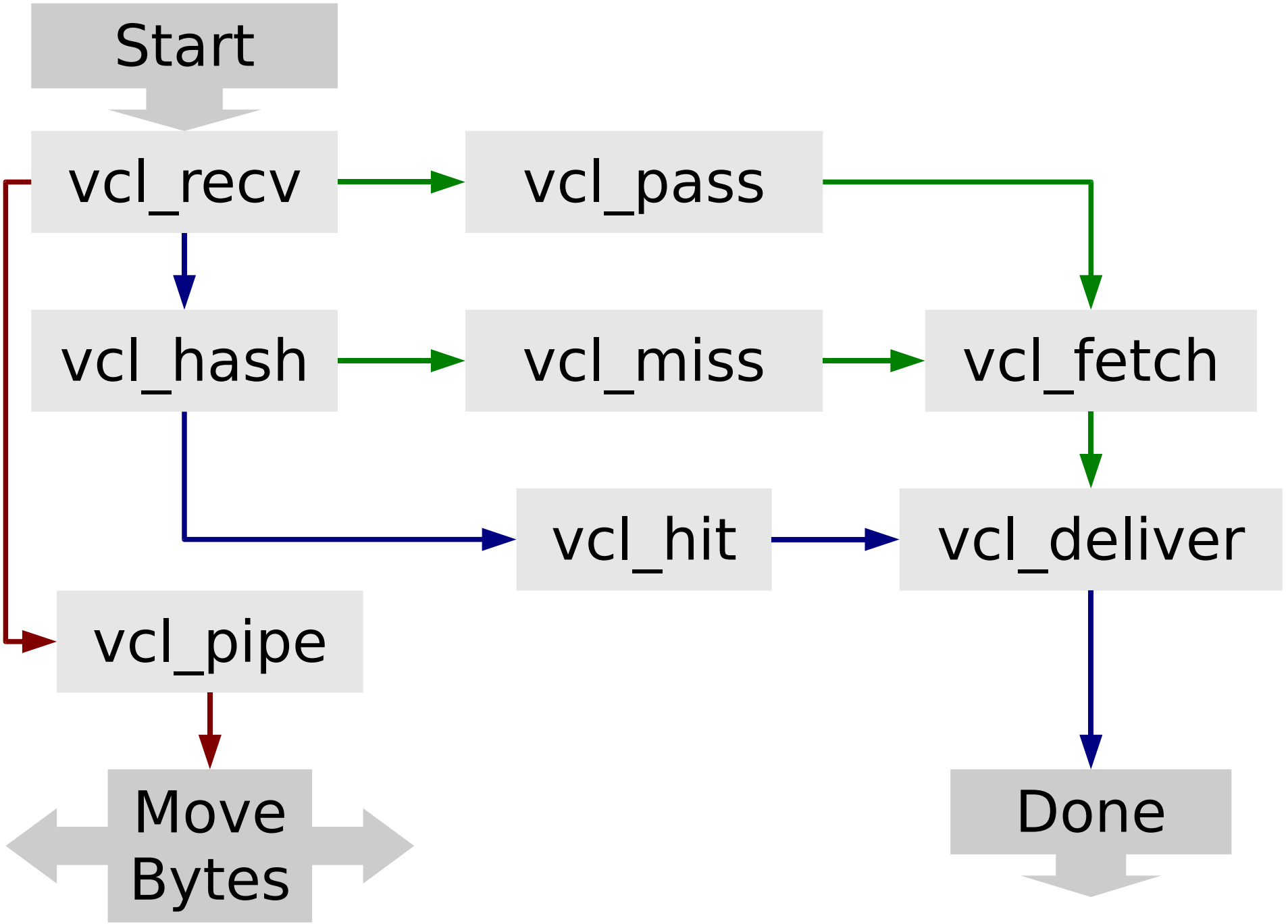
Debugging VCL code

If all else fails: study the compiled C-code:

varnishd -d -f foo.vcl -C

```
static int
VGC_function_vcl_recv (struct sess *sp)
{
  {
    {
      VRT_count(sp, 1);
      if
        (
          (
            (
              VRT_count(sp, 2),
              (
                VRT_strcmp(VRT_r_req_request(sp), "GET")
              )
            ) && (
              VRT_count(sp, 3),
              (
                VRT_strcmp(VRT_r_req_request(sp), "HEAD")
              )
[...]
```

```
sub vcl_recv {[1]
    if ([2]req.request != "GET" &&
      [3]req.request != "HEAD" &&
[...]
```


Start
vcl_recv
vcl_pass
vcl_hash
vcl_miss
vcl_fetch
vcl_hit
vcl_deliver
vcl_pipe
Move
Bytes
Done

vcl_recv – wash, clean and judge

```
sub vcl_recv {
    if (req.request != ”GET” &&
        req.request != ”HEAD” &&
        req.request != ”PUT” &&
        req.request != ”POST” &&
        req.request != ”TRACE” &&
        req.request != ”OPTIONS” &&
        req.request != ”DELETE”) {
            // Non-RFC2616 or CONNECT
            pipe;
    }
[...]
```

vcl_recv – wash, clean and decide

```
[...]
    if (req.http.Expect) {
            // Too hard
            pipe;
    }
    if (req.request != ”GET” &&
        req.request != ”HEAD”) {
            // Not cacheable by default
            pass;
    }
[...]
```

vcl_recv – wash, clean and decide

```
[...]
    if (req.http.Authenticate ||
        req.http.Cookie) {
            // Not cacheable
            pass;
    }
    lookup;
}
```

vcl_hash – what does ”object” mean

```
sub vcl_hash {
    set req.hash += req.url;
    if (req.http.host) {
        set req.hash += req.http.host;
    } else {
        set req.hash += server.ip;
    }
    hash;
}
```

Elements terminated by '#' character in final hash string, thus: ”/#myhost.com#”.

This is important to know for purge_hash().

vcl_fetch – what we got

```
sub vcl_fetch {
    if (!obj.valid) {
        error obj.status;
    }
    if (!obj.cacheable) {
        pass;
    }
    if (obj.http.Set-Cookie) {
        pass;
    }
    insert;
}
```

vcl_hit – what now ?

```
sub vcl_hit {
    if (!obj.cacheable) {
        pass;
    }
    deliver;
}
```

We can cache the fact that we can not cache a given object.

This disables the ”only one at a time” queue on subsequent accesses.

vcl_xxx ? -- the rest

```
sub vcl_miss { fetch; }

sub vcl_pipe { pipe; }

sub vcl_pass { pass; }

sub vcl_deliver { deliver; }

sub vcl_discard { discard; }

sub vcl_timeout { discard; }
```

Prepend and/or replace vcl code

```
acl inhouse { 10.0.0.0/8; }

sub vcl_recv {
    if (client.ip ~ inhouse) {
        pass;
    }
    if (req.url ~ ”[.]jpg$”) {
        unset req.http.cookie;
    }
}
```

Pass is an action, so execution stops here.

No action here, continue into default vcl_recv{}

Access Control Lists

```
acl myfriends {
    10.0.0.4;              // A single host

    ! 10.1.0.1;            // Not this host

    10.1.0.0/24;           // A network

    (our.net/24);          // optional DNS + mask

    !some.host.com;        // A DNS name

    include ”long_list_of_ips.txt”;
}
```

Fixing a mistake

```
sub vcl_recv {
    if (req.url == ”index.hmtl”) {
        set req.url = ”index.html”;
    }
}
```

Stopping robots

```
sub vcl_miss {
    if (req.http.user-agent ~ ”spider”) {
        error 503 ”Not presently in cache”;
    }
}
```

Prevent google and other spiders from pulling 10 years worth of old articles into cache.

Deleting a tracking argument

```
sub vcl_hash {
    vcl.hash += regsub(req.url, ”\?.*”, ””);
    hash;
}
```

regsub(string, pattern, replacement)

Purging, squid style

```
sub vcl_recv {
    if (req.request == ”PURGE”) {
        if (client.ip ~ ournet) {
            lookup;
        } else {
            error 405 ”Not allowed”
        }
    }
}
```

NB: Squid style purging only works if you have the exact URL you want to purge.

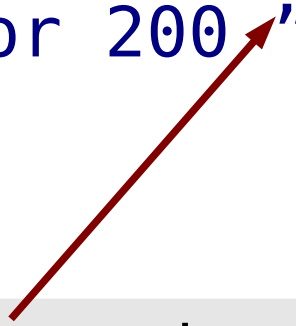
Purging, squid style

```
sub vcl_hit {
    if (req.request == ”PURGE”) {
        set obj.ttl = 0s;
        error 200 ”Purged”
    }
}

sub vcl_miss {
    if (req.request == ”PURGE”) {
        error 404 ”Not in cache”
    }
}
```

Purging, varnish style

```
sub vcl_recv {
    if (req.request == ”PURGE”) {
        if (client.ip ~ our_net) {
            purge_url(req.url);
            error 200 ”Purged”;
        }
    }
}
```

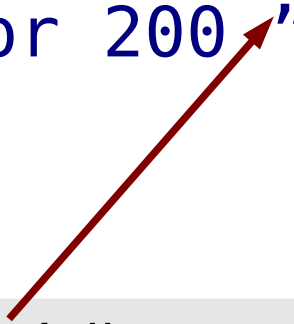


The argument is a regular expression, which is evaluated on demand only.

Because it is on demand, it is possible to instantly purge ”\.jpg$”, even though that may be 1mio objects in a 100mio cache.

Purging, varnish style

```
sub vcl_recv {
    if (req.request == ”PURGE”) {
        if (client.ip ~ our_net) {
            purge_hash(req.http.purgestring);
            error 200 ”Purged”;
        }
    }
}
```



purge_hash() matches against the hash-string built by vcl_hash, thus also taking hostname into account.

Selecting a backend

```
sub vcl_recv {
    if (req.url ~ ”\.(gif|jpg|swf|css|j)$”) {
        unset req.http.cookie;
        unset req.http.authenticate;
        set req.backend = b1;
    } else {
        set req.backend = b2;
    }
}
```

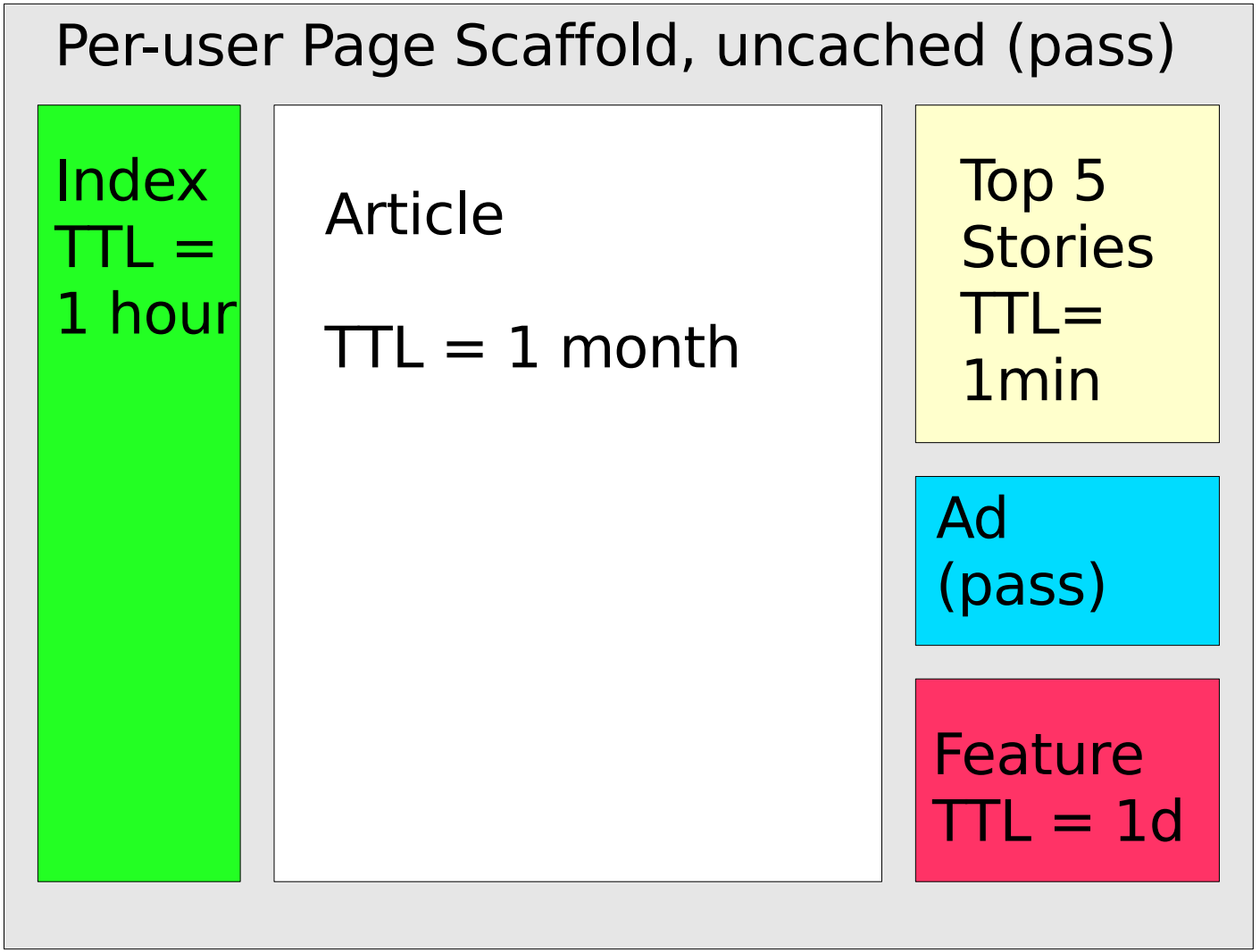
Trying another backend

```
sub vcl_recv {
    if (req.restarts == 0) {
        set req.backend = b1;
    } else {
        set req.backend = b2;
    }
}

sub vcl_fetch {
    if (obj.status != 200) {
        restart;
    }
}
```

Go to vcl_recv{} and try request again.
Parameter limits max number of restarts for each request.

Edge-Side-Includes (”ESI”)



ESI web document

```
[...]
<TABLE>
<TR><esi:include src=”row1.html”/></TR>
[...]
```

<esi:include.../> is replaced by the object ”row1.html”

”row1.html” is treated as separate cache-object.
-> Has it's own TTL & Expiry
-> Can be fetched from a different backend
-> Can be ESI processed (max depth is a parameter)

Enabling ESI processing

```
sub vcl_fetch {
    if (req.url ~ ”\.html$”) {
        esi;
    }
}
```

ESI processing takes CPU time, only enable on relevant document (classes).

ESI can be used also on binary data, but make sure you know what you are doing.