Varnish Cache: What it is and why it's cool.

Kristian Lyngstøl
Developer,
Norman Shark AS

Lysaker, May, 2013

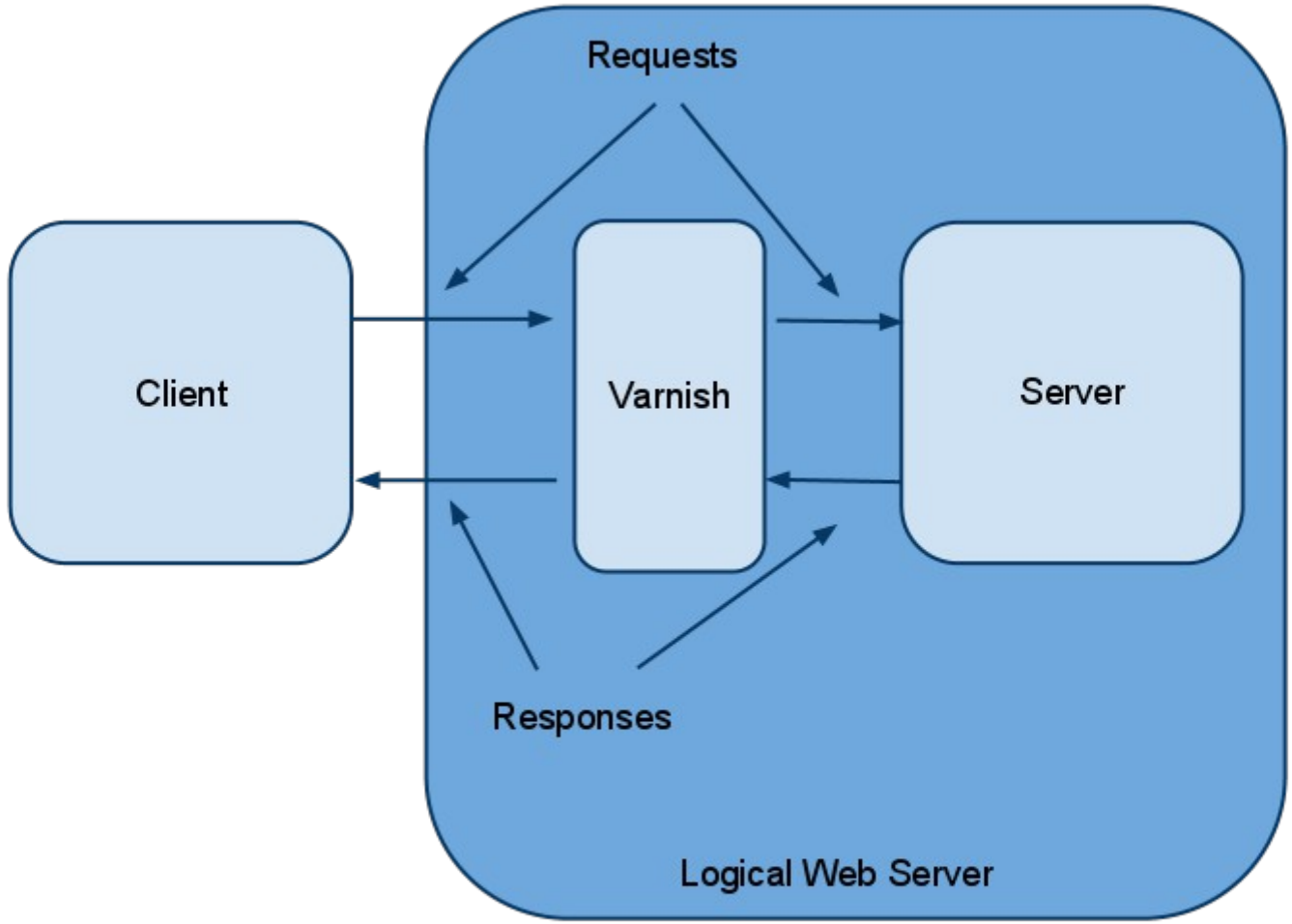
Quick Glance

- Web accelerator for slow CMS systems
- Simple
  - HTTP only
  - No FTP, no SSL, etc.
- High performance
  - Focused on RAM, not disk storage
  - Optimized for 64-bit
  - Optimized for multi-core/thread
  - Modern software design
- Nice HTTP swiss army knife.

Quick Glance

- Runs on: Linux, FreeBSD
  - (and known to run at numerous other UNIXes too, with varying degrees of success).
- Free Software
  - BSD license
- Web site
  - http://www.varnish-cache.org
- Book
  - https://www.varnish-software.com/static/book/

Web application accelerator



or a *caching HTTP proxy*

The briefest of histories

- 2005: Anders Berg, the sole sysadmin of VG.no tire of escalating hardware needs and gets the idea for Varnish Cache, contacts Poul-Henning Kamp (FreeBSD kernel developer) and gets things started.
- 2006: Varnish is released. VG is able to run their site on a single Varnish if needed.
- 2008: Varnish 2.0 is released – first version useful for non-vg.no-sites.
- 2009: First user group meeting gather around 12 people.
- 2012: Sixth user group meeting gather 80+ people from around the world.

Some known users

- Most high-traffic Norwegian sites (Amedia, Schibsted, Aller Internett, etc)
- Facebook, Twitter, Wordpress.com, Wimp (Aspiro), Mercado Libre, Globo, The Hindu, Vimeo, Fastly (Varnish-based CDN), Wikipedia (ish), Manwin (ahem!), Slashdot, BBC, etc, etc.

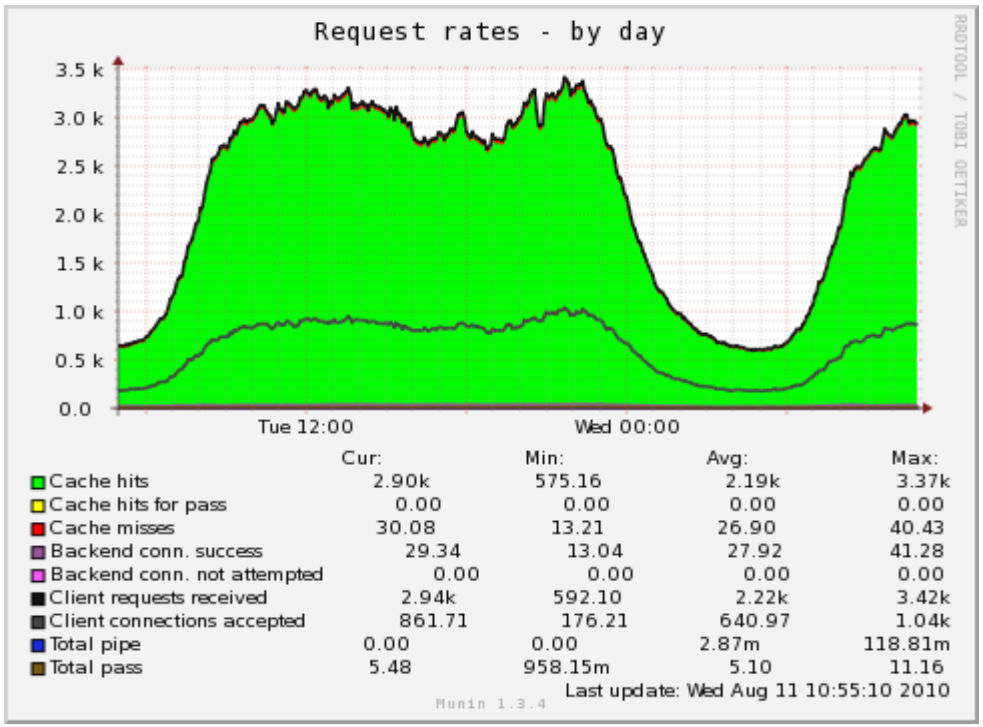
Why do people use Varnish?

It is fast.

It is flexible.

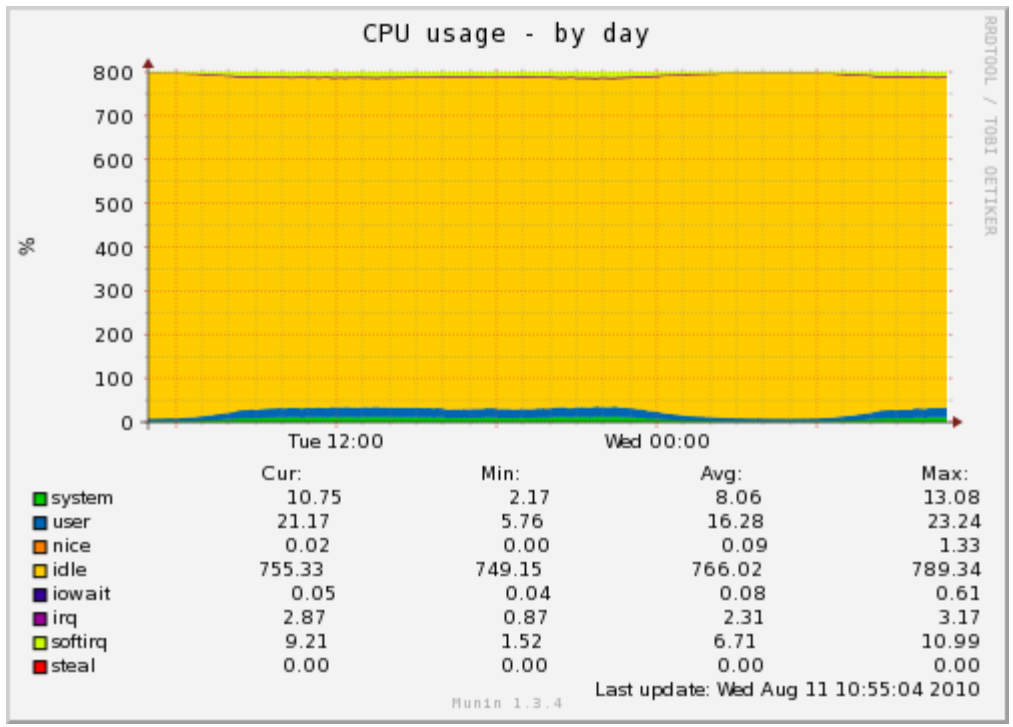
There's a bunny on the home page.

Request- and Connection rate



Peaks at:
3.5k requests/second
1k connections/s

CPU Usage (8 cores)

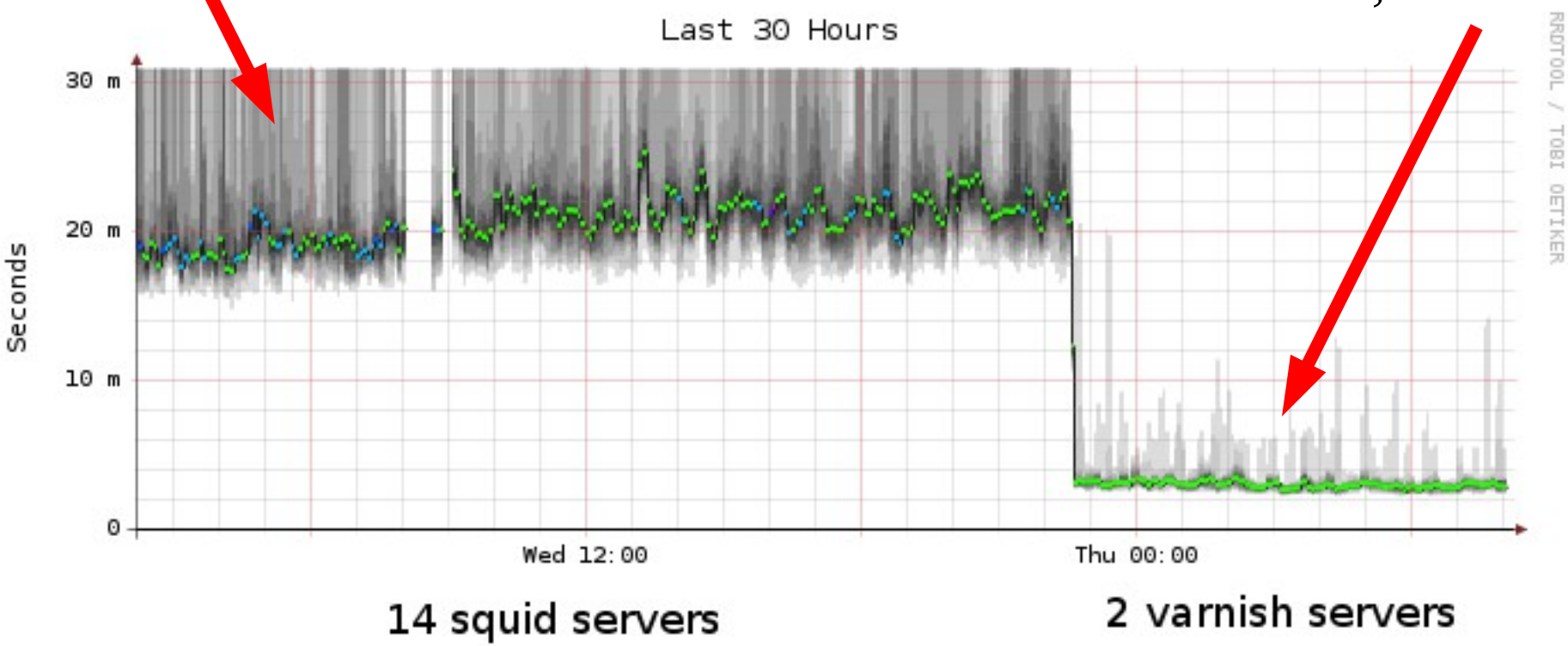


Minimum idle CPU usage:
749.15%
(Yellow is idle)

HTTP Response time, in ms

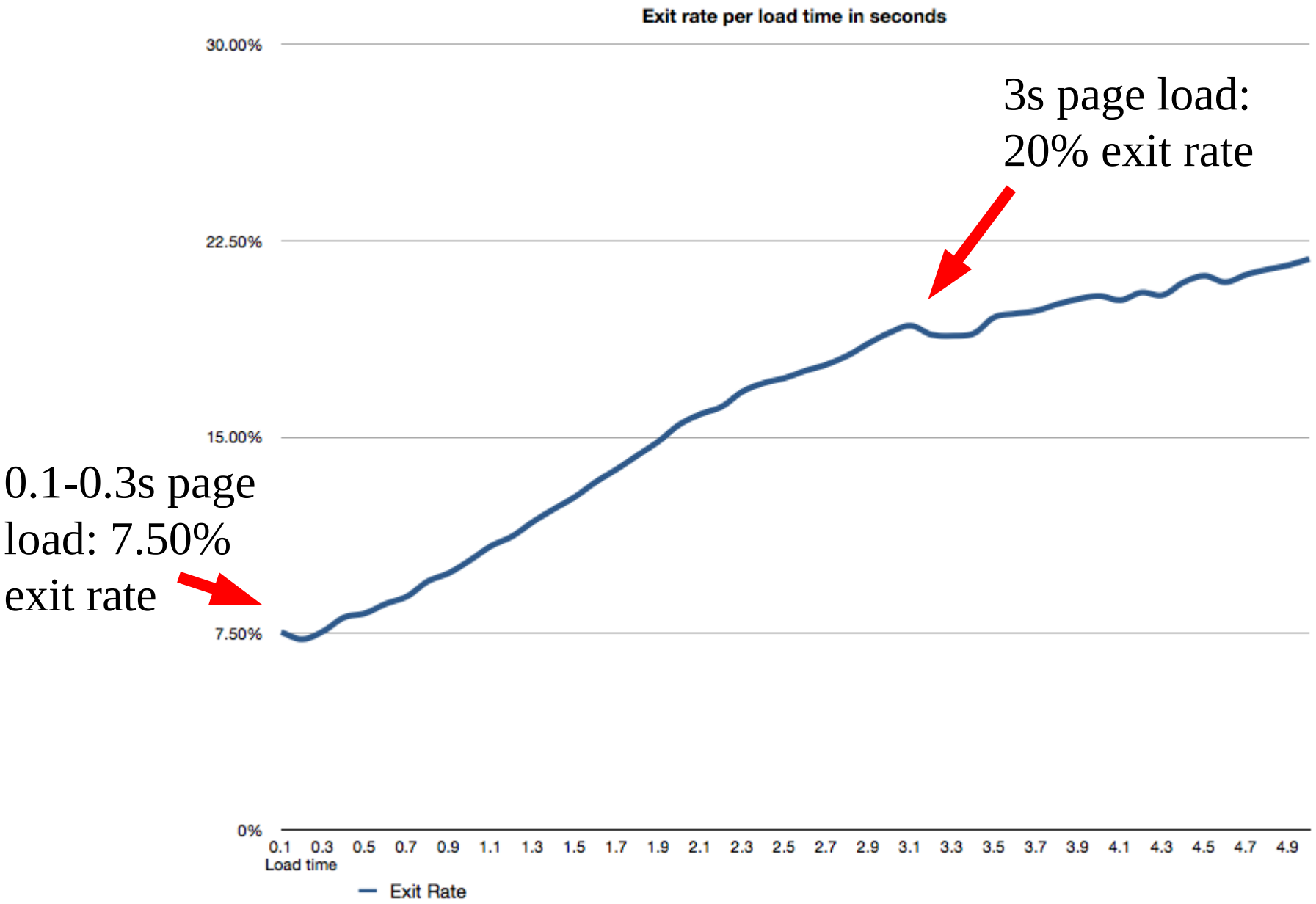
20-50ms, with lost replies

3-10ms, no lost replies



Roughly 4.5k requests/second

Exit rate versus page load time

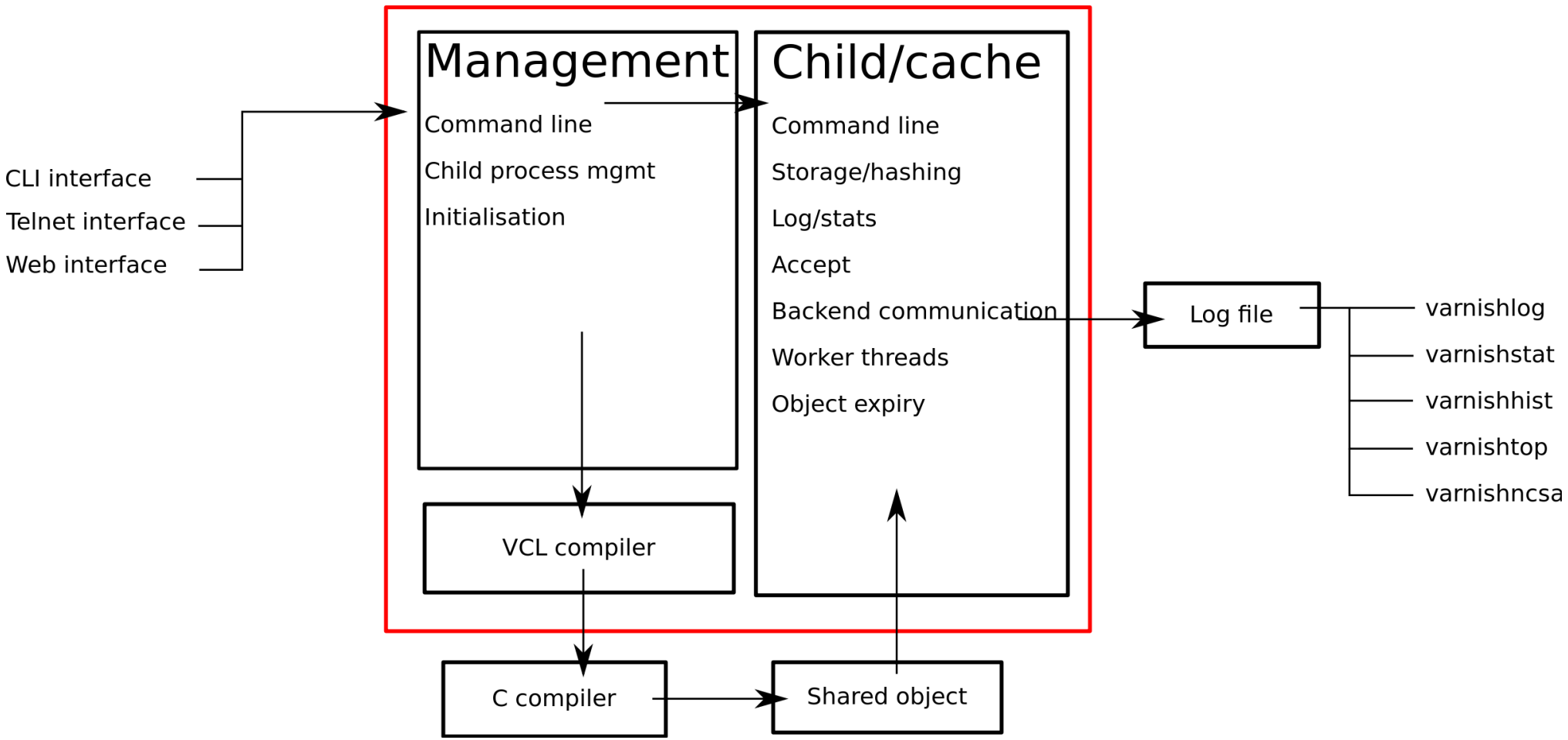


Courtesy of Artur Bergman, http://www.flickr.com/photos/crucially/3716344792/

The Architecture of Varnish

- 64 bit - support 32 bit
- Heavily multi-threaded (1 session ~= 1 thread)
- Work with the OS
- Avoid synchronization operations if possible
- Use thread- and session-specific workspaces to avoid malloc() triggering process wide synchronization

Let the VCL figure out the policy


Management
Command line
Child process mgmt
Initialisation
Child/cache
Command line
Storage/hashing
Log/stats
Accept
Backend communication
Worker threads
Object expiry
CLI interface
Telnet interface
Web interface
VCL compiler
C compiler
Shared object
Log file
varnishlog
varnishstat
varnishhist
varnishtop
varnishncsa

“Everything” is a parameter

```
gzip_level                6 []
gzip_memlevel             8 []
gzip_stack_buffer         32768 [Bytes]
gzip_tmp_space            0 []
gzip_window               15 []
http_gzip_support         on [bool]
http_max_hdr              64 [header lines]
http_range_support        on [bool]
http_req_hdr_len          8192 [bytes]
http_req_size             32768 [bytes]
http_resp_hdr_len         8192 [bytes]
http_resp_size            32768 [bytes]
```

```
acceptor_sleep_decay        0.900000 []
acceptor_sleep_incr         0.001000 [s]
acceptor_sleep_max          0.050000 [s]
auto_restart                on [bool]
ban_dups                    on [bool]
ban_lurker_sleep            0.010000 [s]
between_bytes_timeout       60.000000 [s]
cc_command                  "exec gcc -std=gnu99 -g -O2
  -pthread -fpic -shared -Wl,-x -o %o %s"
cli_buffer                  8192 [bytes]
cli_timeout                 10 [seconds]
clock_skew                  10 [s]
connect_timeout             0.700000 [s]
critbit_cooloff             180.000000 [s]
default_grace               10.000000 [seconds]
default_keep                0.000000 [seconds]
default_ttl                 120.000000 [seconds]
diag_bitmap                 0x0 [bitmap]
esi_syntax                  0 [bitmap]
```

Defensive programming

Around 5% of all code is some sort of assert().

Use magic markers in any long lived memory structure.

“The cost of an assert is negative: Any time spent writing and running it is made up for tenfold when it reveals a bug.”

Magic marker

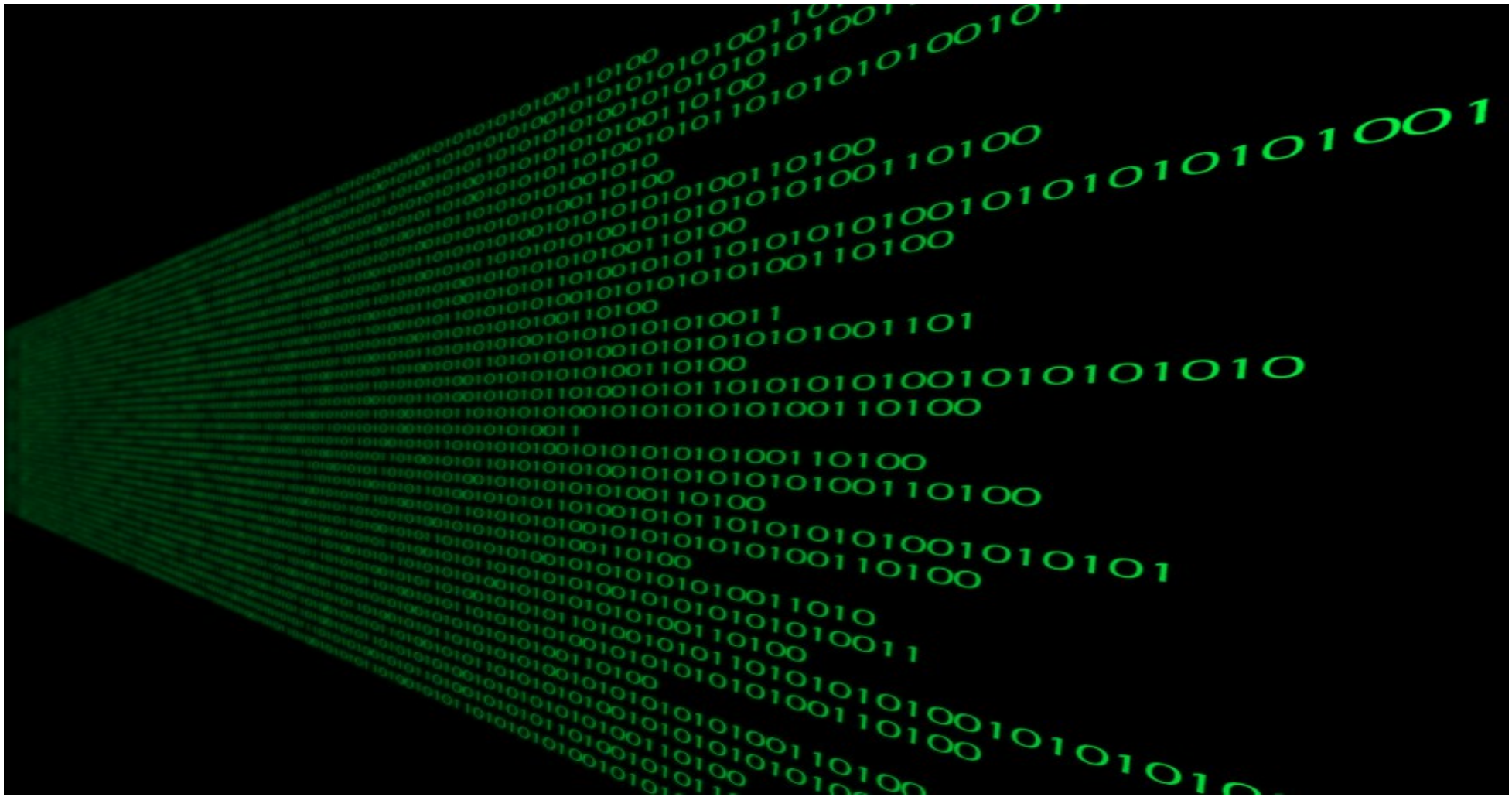
```
struct worker {
        unsigned   magic;
#define WORKER_MAGIC 0x6391adcf
        struct pool *pool;
        struct objhead *nobjhead;
        (...)
```

Magic marker

```
#define CHECK_OBJ_NOTNULL(ptr, type_magic)                              \
        do {                                                            \
                assert((ptr) != NULL);                                  \
                assert((ptr)->magic == type_magic);                     \
        } while (0)
```

```
static void
hsh_prealloc(struct worker *wrk)
{
   /*...*/
   CHECK_OBJ_NOTNULL(wrk, WORKER_MAGIC);
```

Config to Code



Policy based configuration
VCL

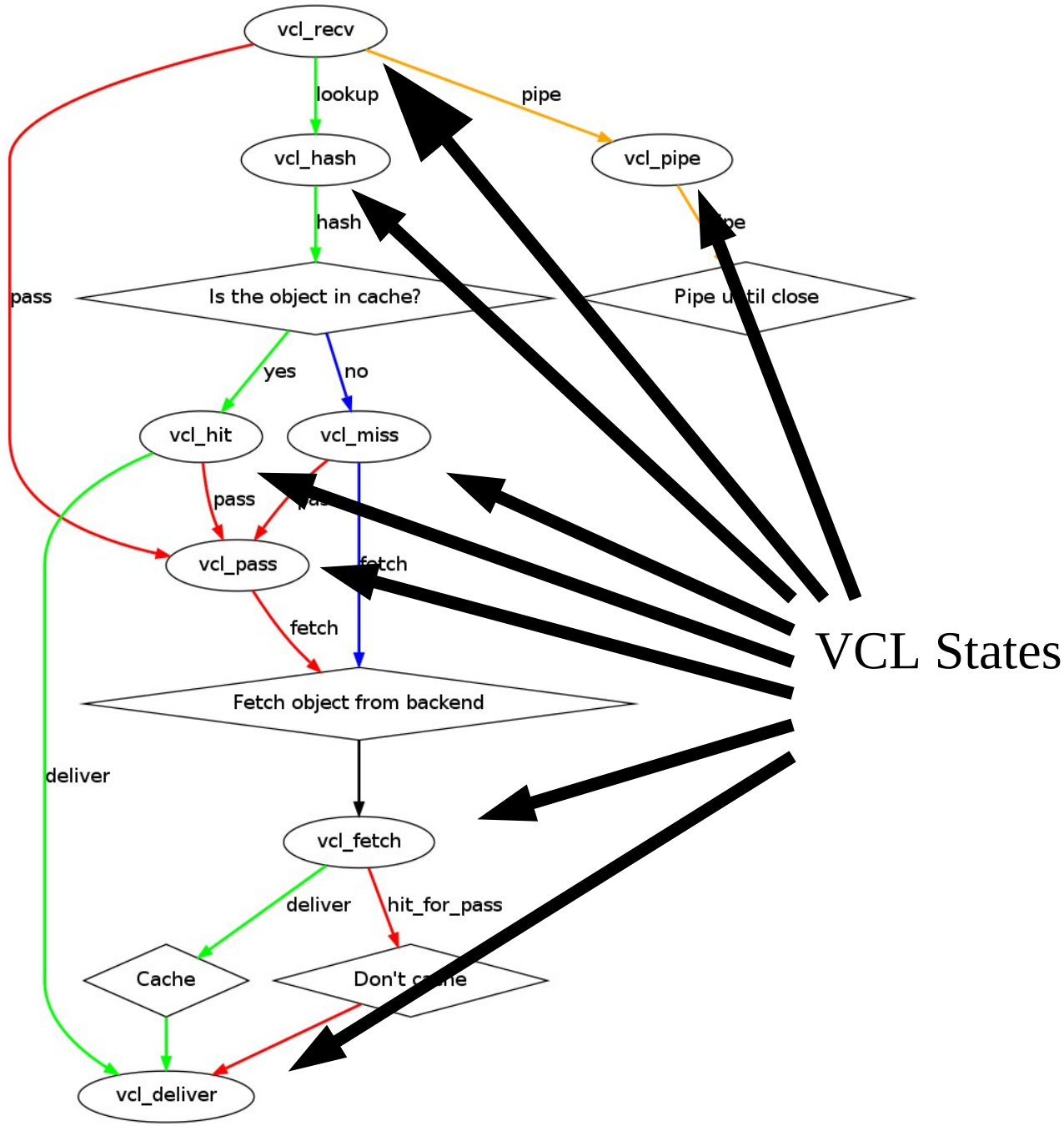
Varnish Configuration Language

- A simple Domain Specific Language to control the request state engine.
- VCL provides mechanisms
- Administrators provide policy

Varnish Configuration Language

- An interface to the inner workings of Varnish
- Transformed to C, compiled and linked in
- Fast run-time configuration switching
- In-line C provides an escape hatch to write C in your configuration file.

VCL: The simple version



VCL

```
if (req.http.user-agent ~ "IE") {
   set req.url = "/ie-page";
}
```

VCL → C

```
if (VRT_re_match(sp, VRT_GetHdr(sp,
    HDR_REQ, "\013user-agent:"),
    VGC_re_1[1]))
        {
          VRT_l_req_url(sp,"/ie-page",
              vrt_magic_string_end);
        }
```

[1]: VGC_re_1: Refers to the pre-compiled regex.

C → VCL → C ?

```
if (req.http.user-agent ~ "IE") {
    C{
     sleep(1);
    }C
  }
```

C → VCL → C

```
if (VRT_re_match(sp,
  VRT_GetHdr(sp, HDR_REQ,
  "\013user-agent:"), VGC_re_1)) {
    {
      sleep(1);
    }
}
```

Varnish Modules

Formal ways to embed in-line C in VCL.

```
import std;
sub vcl_recv {
   // Set the request header x-rand
   // to a random value between 1 and 100
   set req.http.x-rand = std.random(1,100);
}
```

Vmods

- Easy to use
- Relatively easy to write
- Example vmod available
- Regression tests are easy to add

Vmod examples

- std – standard vmod
  - toupper(), random(), log(), fileread(), etc.
- example – template/example vmod, including build system and documentation.
- digest – libmhash wrapper + base64
  - hash_sha256(), hmac_sha1(), base64(), etc
- curl – execute curl requests in VCL
  - e.g: Ask authentication service if a user can access some specific content.
- Various GeoIP look plugins

Varnish *can* save the request

Retry, rewrite, reroute failed TX

How Varnish can save a request

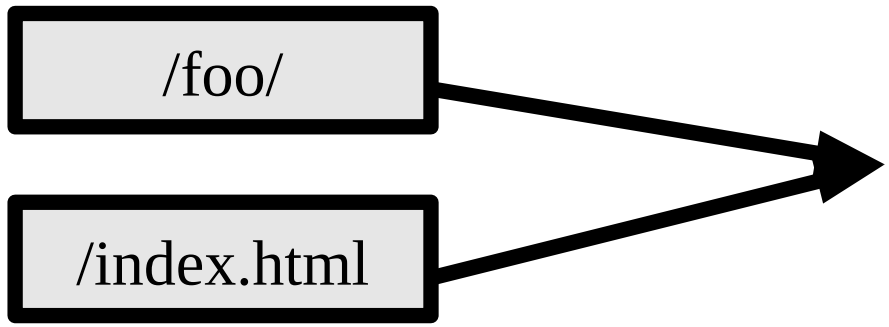
1) Try to fetch the object from app-server #1
2) Rewrite the request, try app-server #2
3) Try to find a stale object in cache and serve it
4) Serve an error

```
sub vcl_recv {
    set req.backend = appserver1;
    if (req.restarts == 1) {
        set req.backend = appserver2;
    }
    set req.grace = 30s;
    if (!req.backend.healthy) {
        set req.grace = 1d;
    }
}
sub vcl_error {
    if (req.restarts == 0) {
        return (restart);
    }
    set obj.http.Content-Type = "text/plain";
    synthetic{"Re-fill your coffee cup and try again."};
    return (deliver);
}
```

Bans

- Invalidate objects
- Instantly added – no blocking
- Ban on (almost) anything!
- Not designed to reclaim memory, but to invalidate content.

How bans work

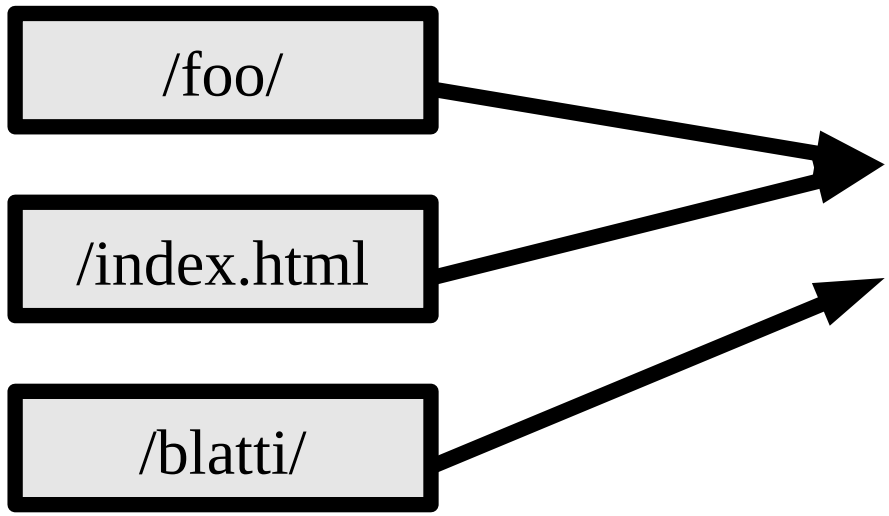


Ban list:

```
END
req.url ~ /bar/
```

```
> ban req.url ~ /bar/
```

How bans work

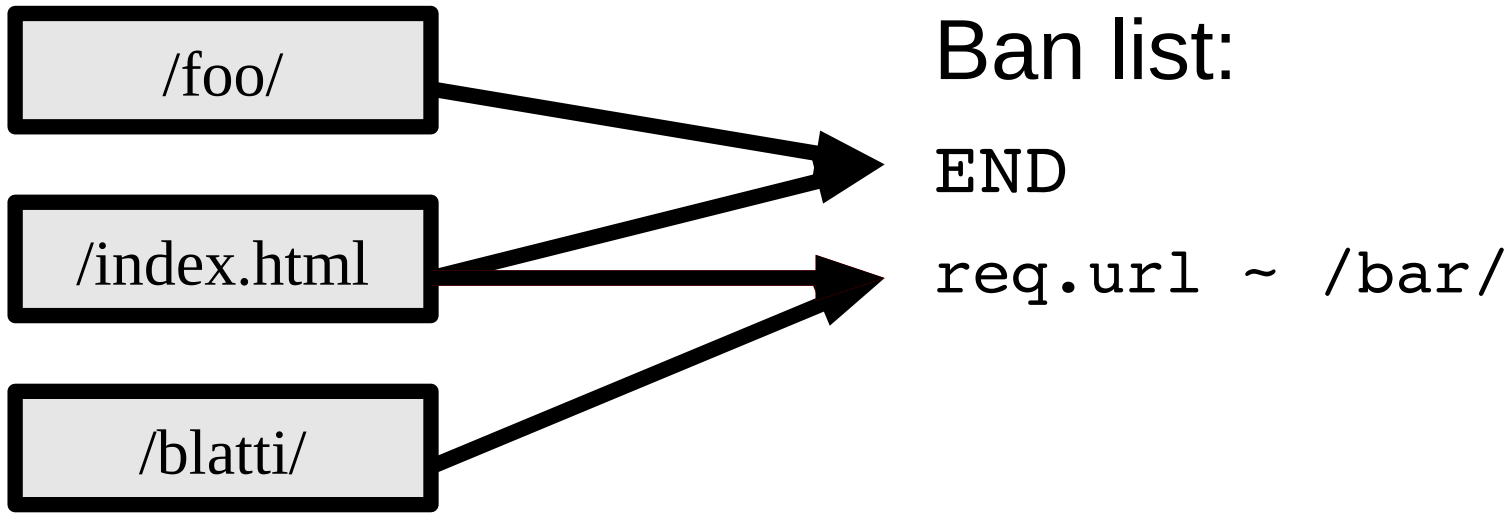


Ban list:

```
END
req.url ~ /foo/
```

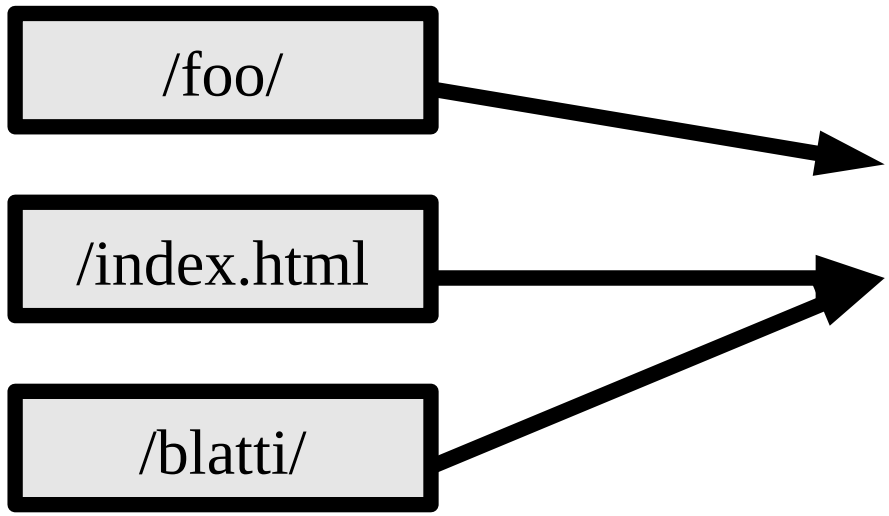
```
> GET /blatti/
```

How bans work




```
GET /index.html
   Test /index.html against newer bans.
   No match? Deliver cache hit!
```

How bans work

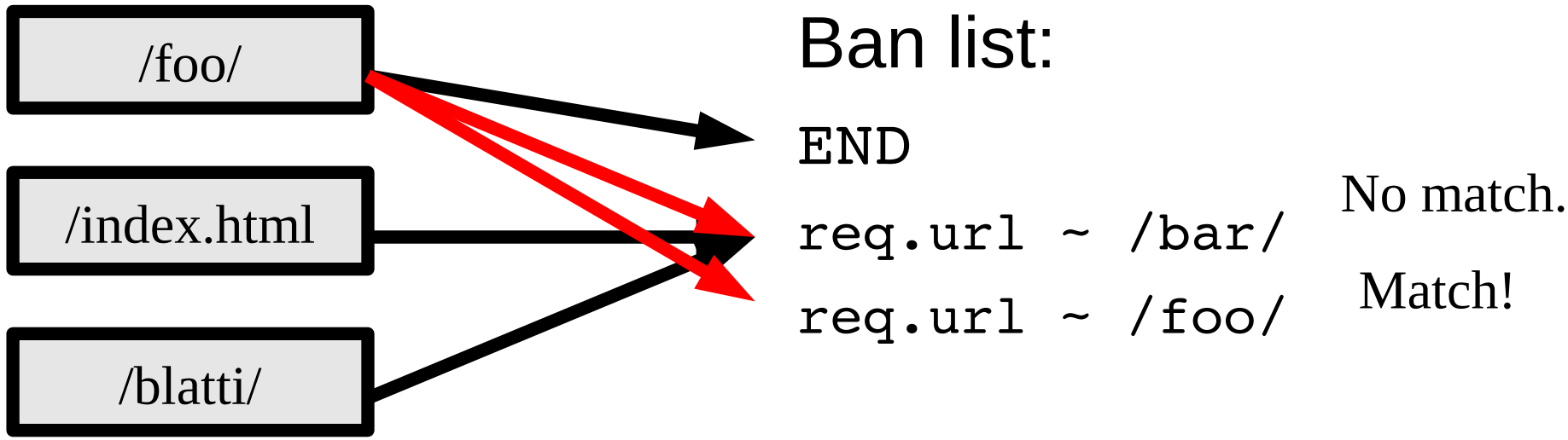


Ban list:

```
END
req.url ~ /bar/
req.url ~ /foo/
```

```
> ban req.url ~ /foo/
```

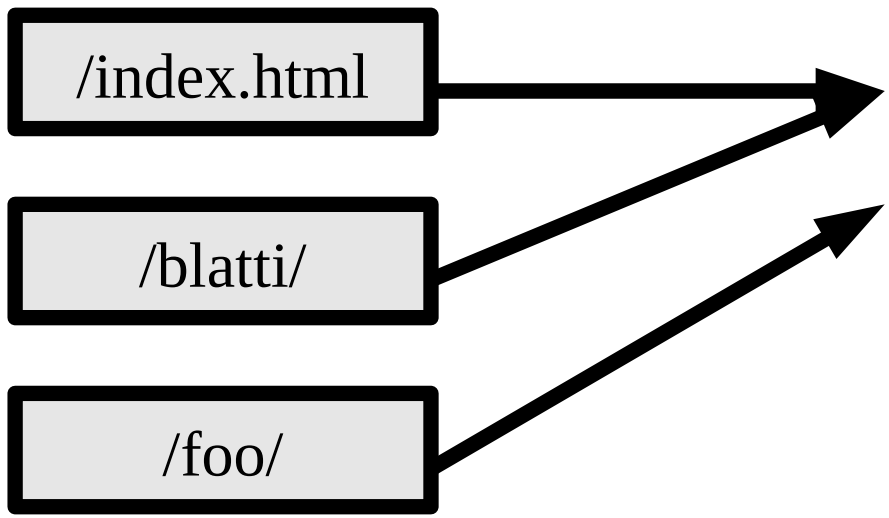
How bans work




```
GET /foo/
   Test /foo/ against newer bans.
   Match! Remove object, return cache miss.
   End of list unreferenced? Remove.
```

How bans work

Ban list:




```
req.url ~ /bar/
req.url ~ /foo/
```

```
# Up to VCL to fetch a new /foo/.
```

Ban examples

```
if (req.request == “BAN”) {
        ban(“obj.http.Cache-Channel ~” + req.url);
        error 200 “Ban added.”
}

if (req.request == “BAN” && client.ip ~ banners) {
     ban(“req.url ~ ” + req.http.x-ban-regex);
     error 200 “Banned “ + req.http.x-ban-regex;
}

$ varnishadm -T localhost:6082 ban req.url == /
```

Fully programmable

Can do awesome tricks!

VCL trick #731

```
sub vcl_recv {
 if (req.http.host ==   “example.com” &&
     req.url ~ “^/fun/” &&
     ( req.http.referer !~
       “^http://example.com/”) ) {
         error 403 “No hotlinking please”;
}
```

Fully manageable

powerful CLI - simple protocol

Logging Varnish

- Most logs are never read
- Logs to shared memory
  - Part 1: Counters
  - Part 2: Log records
- Extensive information – cheap
- Separate tools to analyse the data
- Flexible

Logging tools

- Varnishlog
- Varnishstat
- Varnishhist
- Varnishtop
- Varnishncsa
- Varnishsizes

```
37+13:04:10
Hitrate ratio:       10      100      125
Hitrate avg:     0.9875   0.9858   0.9858

   436566336       247.99       134.58 Client connections accepted
  3738238196      2149.92      1152.41 Client requests received
  3647901706      2113.92      1124.56 Cache hits
     2995698         2.00         0.92 Cache hits for pass
    71691301        25.00        22.10 Cache misses
    83240848        36.00        25.66 Backend connections success
       35457         0.00         0.01 Backend connections failures
    70714372        31.00        21.80 Backend connections reuses
    82628227        35.00        25.47 Backend connections recycles
        2475            .            .   N struct srcaddr
        1097            .            .   N active struct srcaddr
```

Varnish Agent (v2+)

- https://github.com/varnish/vagent2
- Simple REST interface to Varnish.
  - Access logs, stats, VCL and parameters.
- Provides PoC UI
- BSD Licensed
- Plug and play
- (Last thing I wrote for Varnish Software, it's awesome, I promise!)

This is the varnish agent.

GET requests never modify state
POST requests are not idempotent, and can modify state
PUT requests are idempotent, and can modify state
HEAD requests can be performed on all resources that support GET

The following URLs are bound:

- /log	GET			
- /stats	GET			
- /help/ban	GET			
- /ban	GET		POST	
- /help/param	GET			
- /paramjson/	GET			
- /param/	GET	PUT		
- /html/	GET			
- /help/vcl	GET			
- /vcldeploy/		PUT		
- /vcl/	GET	PUT	POST	DELETE
- /vcljson/	GET			
- /package_string	GET			
- /version	GET			
- /help/panic	GET			
- /panic	GET			DELETE
- /start		PUT	POST	
- /stop		PUT	POST	
- /status	GET			
- /echo	GET	PUT	POST	

Active VCL: boot

2req/s

Child in state running

```
Default: 0.900
Value:   0.900000
Unit:

If we run out of resources, such
 as file
descriptors or worker threads, t
he acceptor will
sleep between accepts.
This parameter (multiplicatively
) reduce the
sleep duration for each succesfu
ll accept. (ie:
0.9 = reduce by 10%)

NB: We do not know yet if it is
a good idea to
change this parameter, or if the
 default value is
even sensible.  Caution is advis
ed, and feedback
is most welcome.
```

Parameters

These are run-time parameters of the Varnish Cache daemon. Most of them can be changed on the fly, but some might not take effect for some time (e.g: Changing default_ttl will only apply to new content). Others might require a child restart.

Best practices is to keep your parameters as close to the defaults as possible.

acceptor_sleep_decay ▾

0.900000 Save Default

View non-default parameters

Cache invalidation

The agent only supports banning. For other methods, use VCL.

The agent issues ban commands over the Varnish CLI. It will not free up memory unless you use smart bans.

Keep in mind that if your cache is empty, your ban list will also always be empty. In other words: If you are just testing this and banning something, then get an empty ban list instead of seeing your own ban: make sure the cache has content.

URL to ban. E.g: /foobar.html Ban

List bans

Running

You can stop and start the Varnish child. This clears out all statistics and the cache itself.

While stopped, Varnish does not listen for HTTP connections, but does accept administrative commands (e.g: setting parameters, VCL, and starting it back up again).

For information about panics, see /help/panic. The links below just use this API.

Start Varnish | Stop Varnish | Show panics | Clear panics

Varnishtop

This is a JavaScript implementation of varnishtop. It can help you determine what traffic you should look closer at (e.g: cache misses), what web servers you are using (e.g: Server headers), etc.

Cache misses ▾

```
/  (315 times)
/tomfil  (260 times)
/favicon.ico  (137 times)
/misc/agent-2.0.png  (124 times)
/robots.txt  (122 times)
```

Questions?