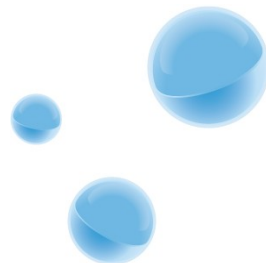
VARNISH

Makes Websites Fly

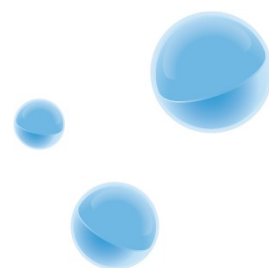
Back to basics

Kristian Lyngstøl
Product Specialist
Varnish Software AS
Twitter: @kristianlyng

Montreal, March 2013

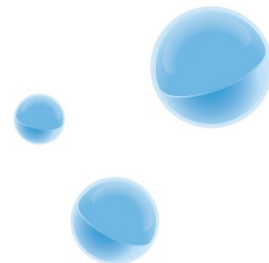


Or: How that new fancy hammer isn't necessarily the solution.

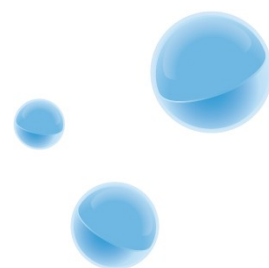


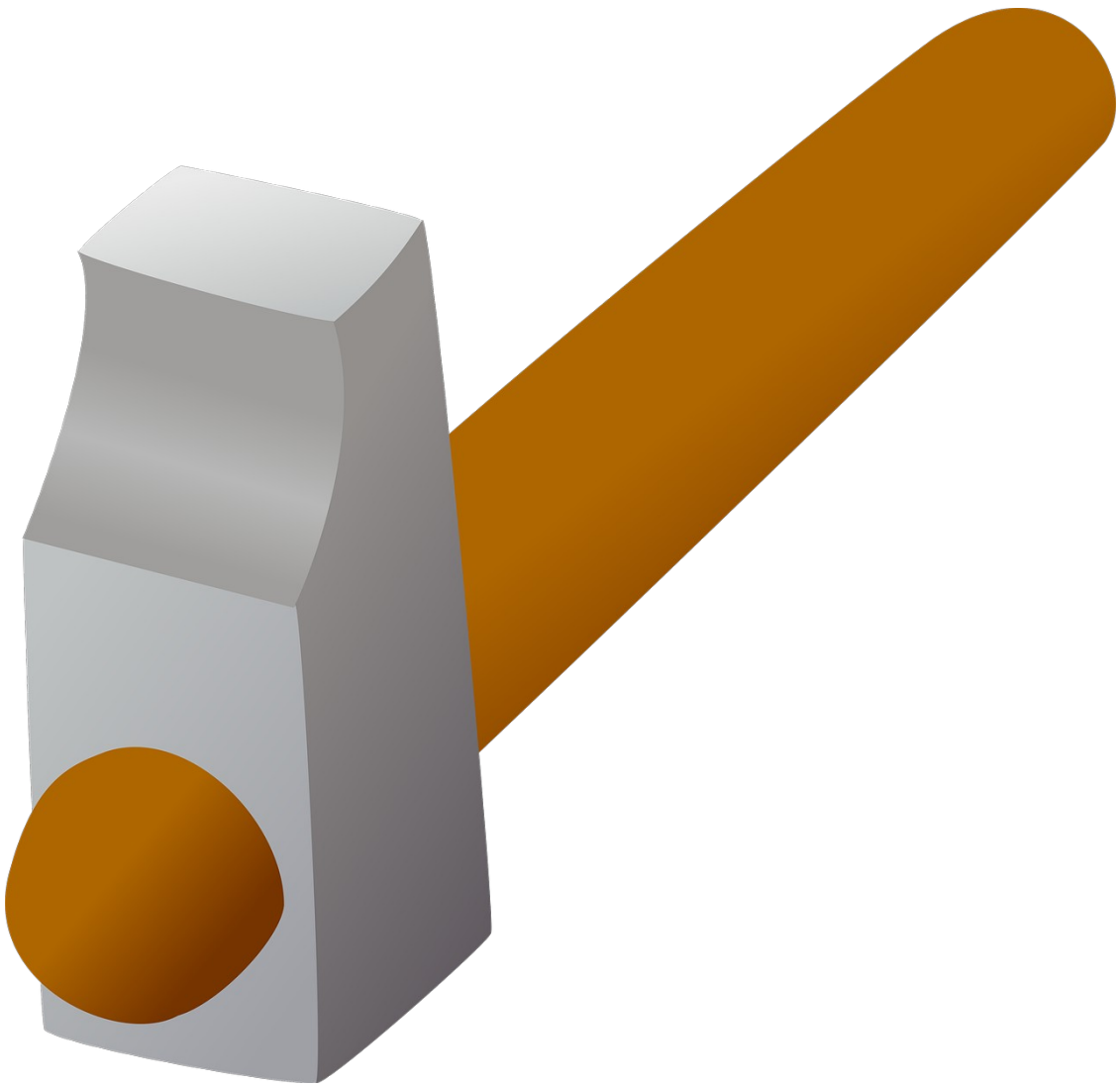
Kristian who?

- GNU/Linux user and developer of many years.
- Mostly work in C and shell-related stuff
- Frequently also work with Python and Java
- Worked on Varnish Cache and Compiz
- Principal author of the Varnish Book.
- More of a system developer than web developer.



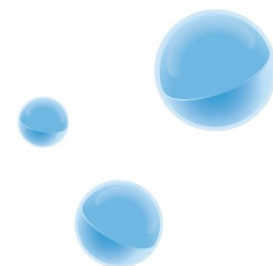
In the beginning, there was the solution.



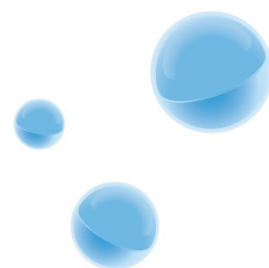


The solution was good. It was new. It was fancy. It was Cool. It was proven to be fast.

People talked about it on hacker news, so it must be awesome.

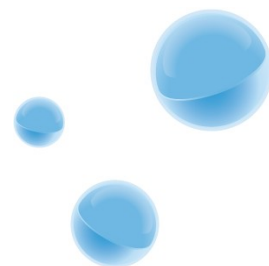


Then along came a project.



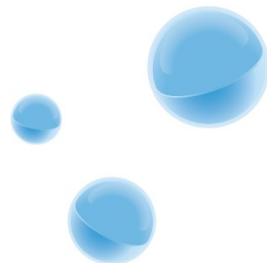
Project Spec

- Purge 2000+ objects per second, across at least 4 different caches.
- Implement retries
- Timeout
- Give status on the success

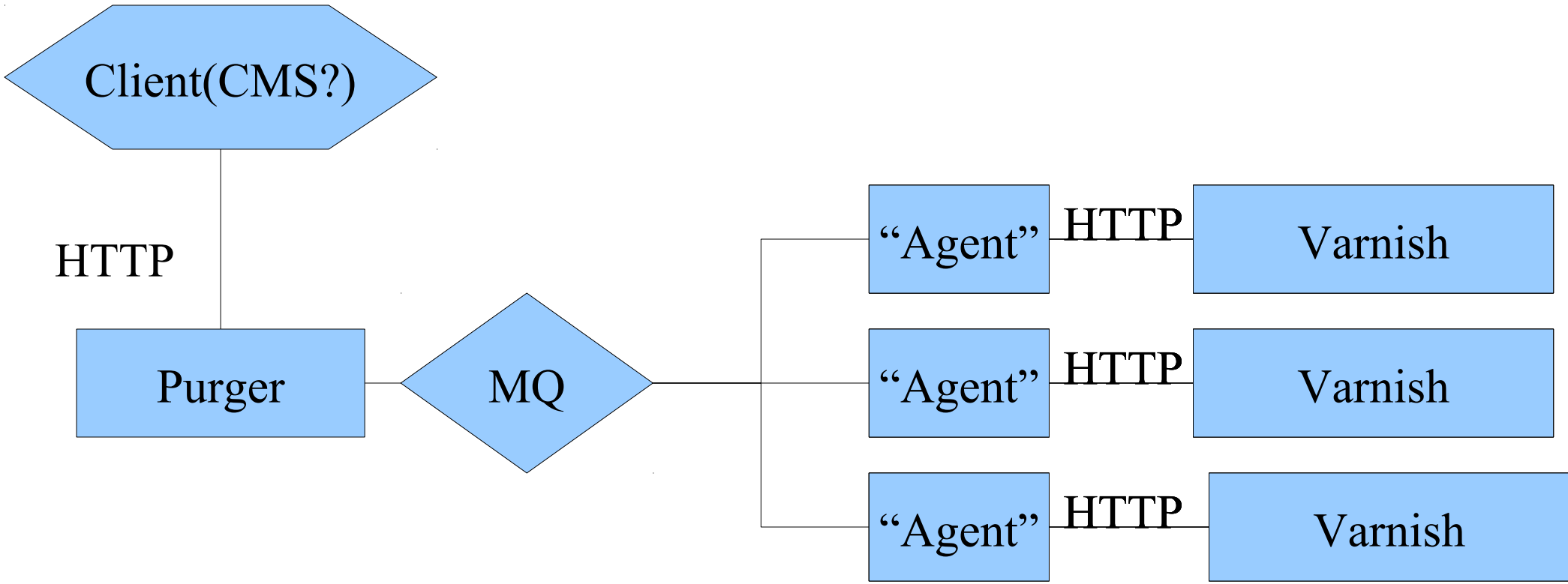


Point of order

- Varnish itself can purge content roughly as fast as it can look them up.
- (In other words: orders of magnitude faster than the spec required)
- (E.g: 100k req/s for this scenario)

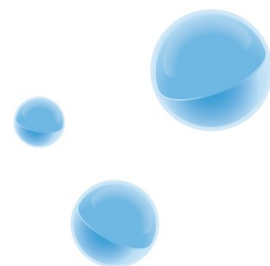


Solution!

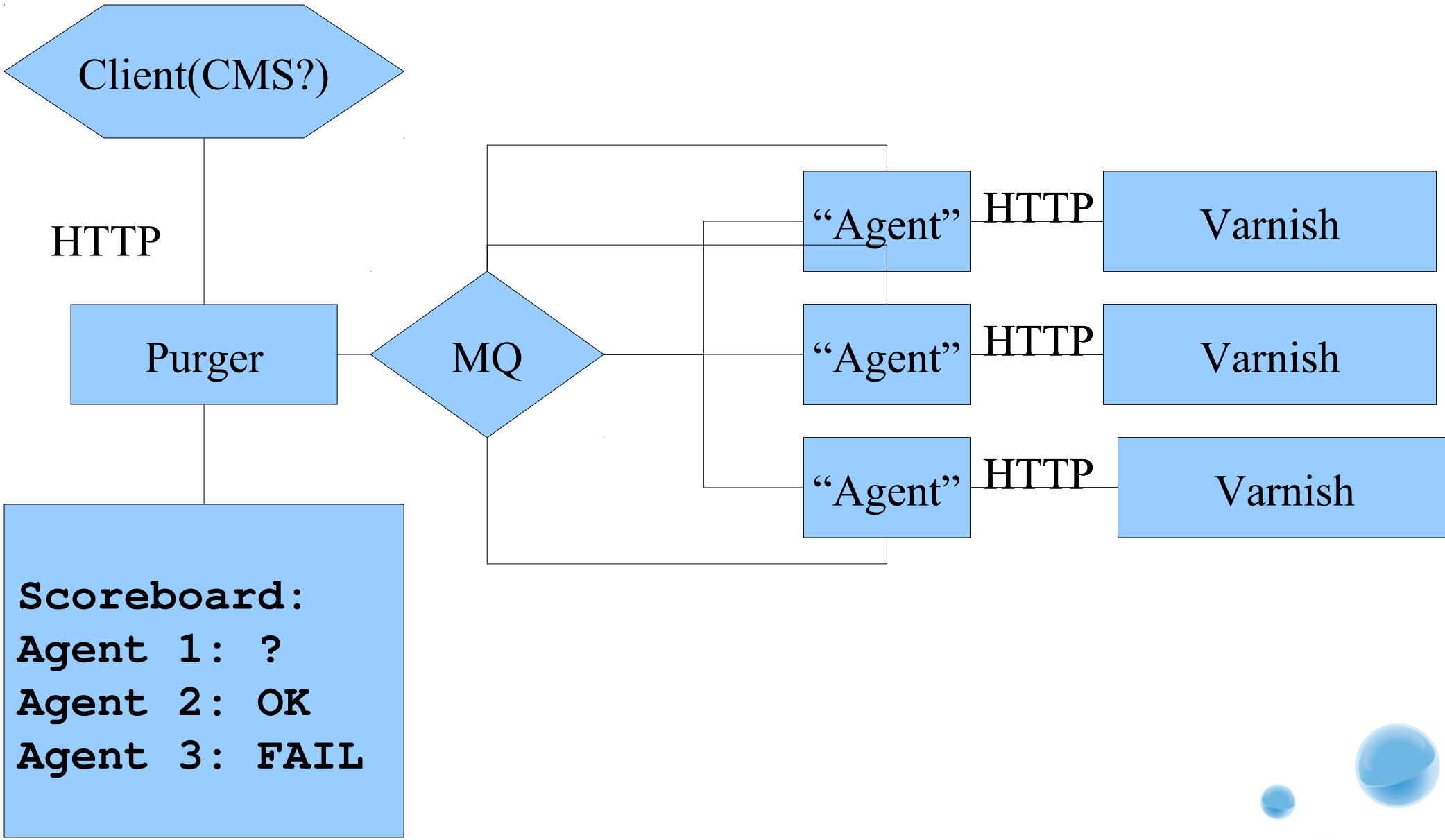

Client(CMS?)
HTTP
Purger
MQ
“Agent”
HTTP
Varnish
“Agent”
HTTP
Varnish
“Agent”
HTTP
Varnish

Solution details

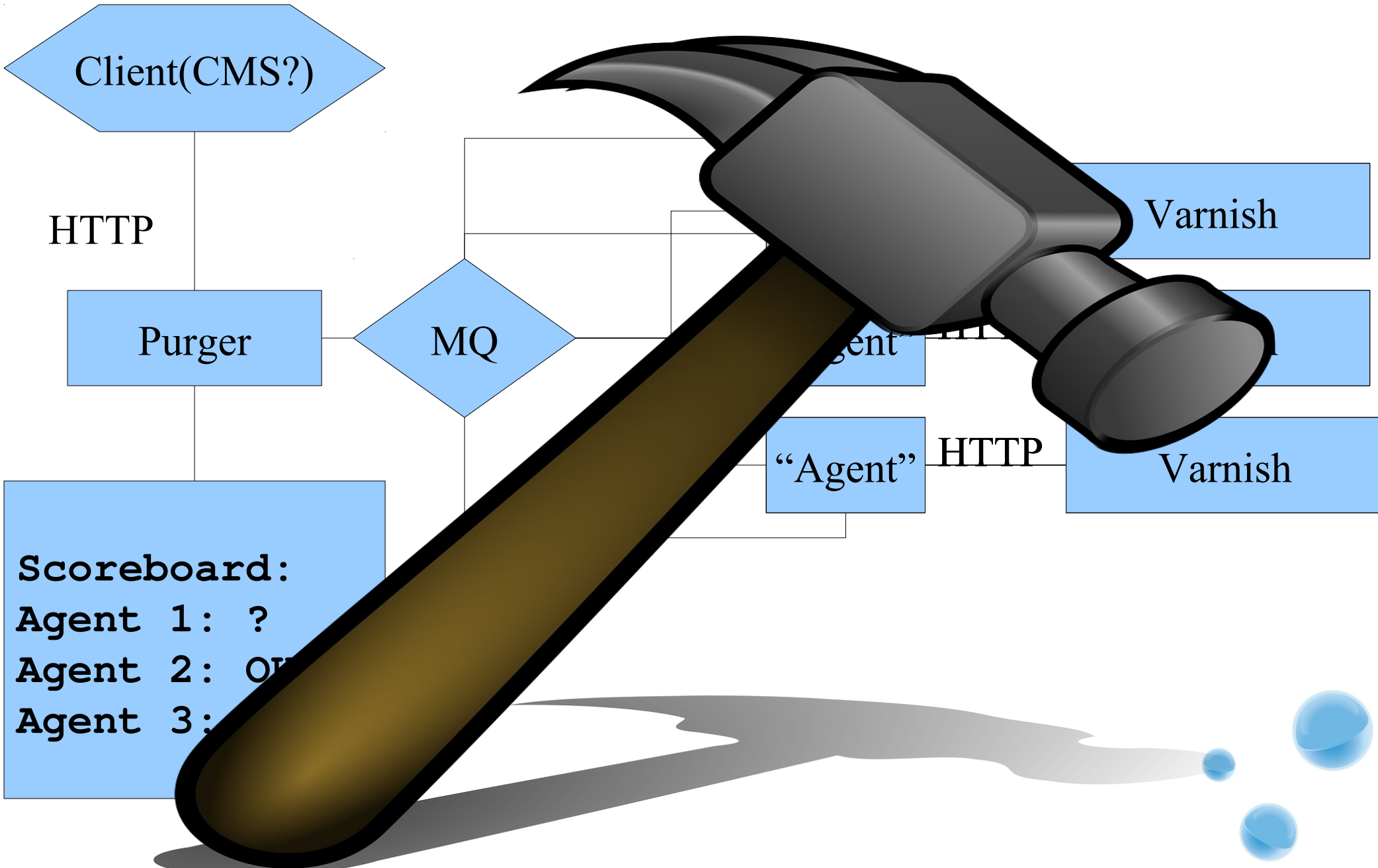
- The MQ was async
- But we needed to report status?
- And retry!
- Solution 2: Scoreboard, and callback when complete.



Solution!

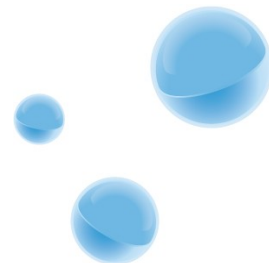

Client(CMS?)
HTTP
Purger
MQ
"Agent"
HTTP
Varnish
"Agent"
HTTP
Varnish
"Agent"
HTTP
Varnish
Scoreboard:
Agent 1: ?
Agent 2: OK
Agent 3: FAIL

Next iteration:



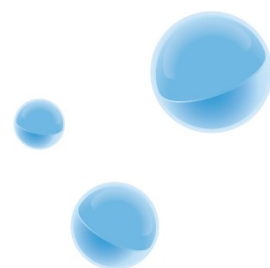
Step 0: WHAT are we trying to do?

- Turn 1 HTTP request into X HTTP requests, possibly retrying on failure.

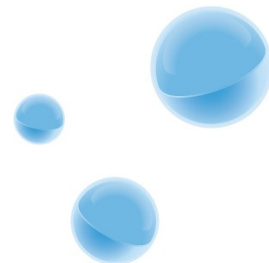


Step 1: What is the **ideal** way to solve this?

At this point, you should pretend you do not know anything about existing solutions.

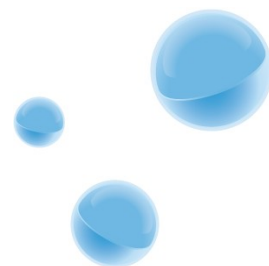


“If I had infinite time, this is how I would do it.”

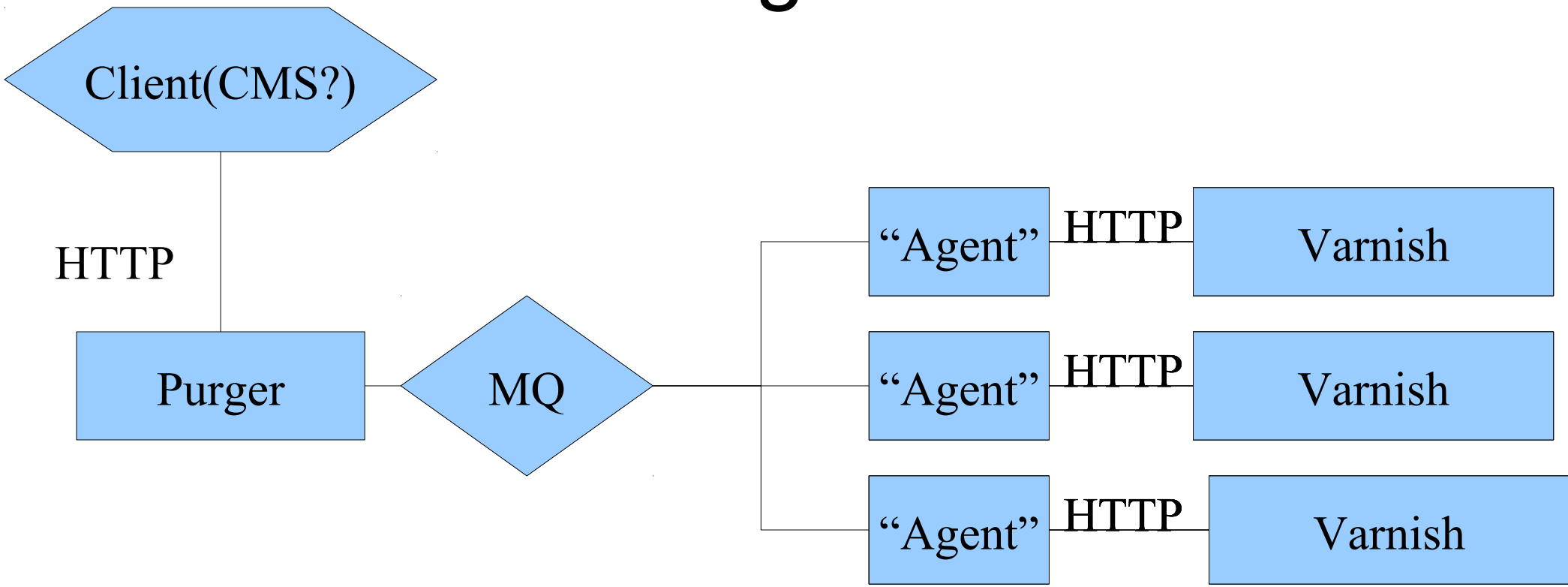


Step 2: Introduce reality

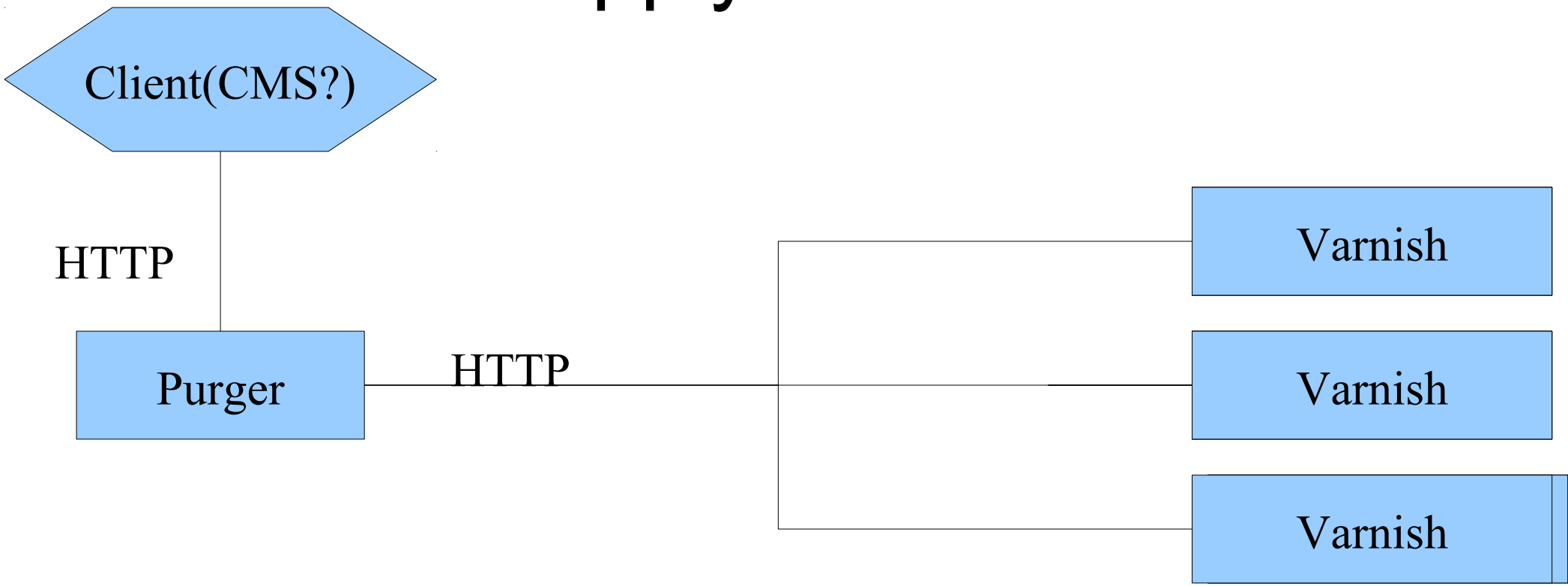
What products exists? What solutions are feasible?



Looking back:

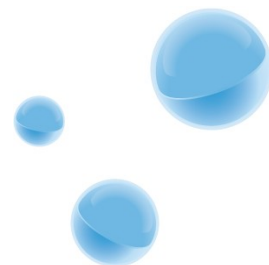


Apply KISS:

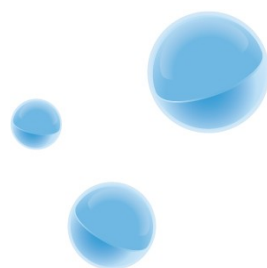


“JDI” solution:

- Speaks HTTP directly to Varnish.
- 1 connection to the purger = 1 connection to each Varnish server.
- 1000 connections to the purger = 1000 connections to the Varnish servers
- Authorization mostly handled by Varnish

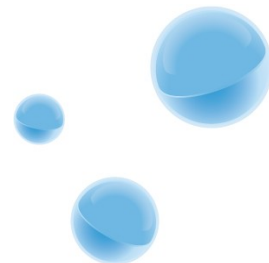


You are not done when there is nothing more to add. You are done when there is nothing more to remove.



Trinkle with magic!

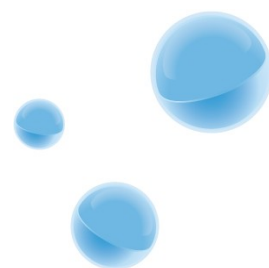
- The code working with varnish is completely isolated from the code talking to clients
- Jetty proved too slow for us. We rolled our own HTTP daemon for this particular purpose (This is on the KISS kill list)
- Test test test test



Lab: 60 000 purges/sec across 4 caches

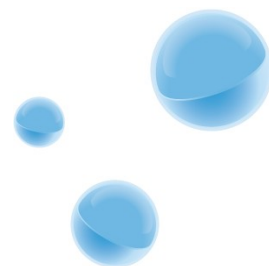
Production: ~600 purges/sec across 4 caches

Why the difference?



Dig down!

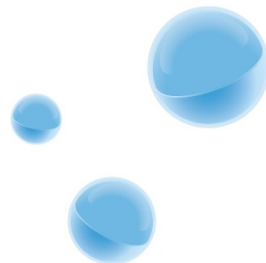
- The solution certainly could be faster.
- Something changed!
- Dig down.



The devil is in the details

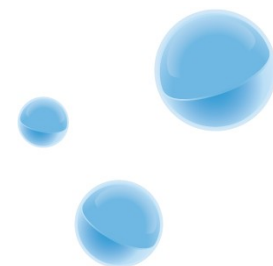
- Lab latency: ~0ms
- Production latency: ~20-30ms
- 25ms RTT can provide 40 synchronous transactions/sec over 1 connection
- Hmm.....

- Our tread pool was hard coded to maximum 16 worker threads.
- 16 * 40 = 640.

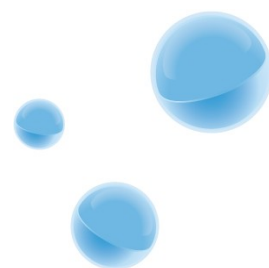


Up the thread pool to 1000 threads. 1000*40 = 40 000 transactions/s theoretically.

We measured about 20k/s successfully without trying very hard.

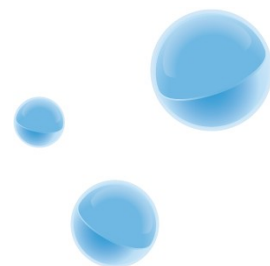


Mastering the “low” level stuff allows you raise the bar on what's possible. If you don't know how to do it yourself, don't use a library or framework.



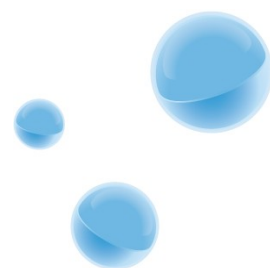
Compete !

- If in doubt, consider implementing a PoC of two or more competing solutions. Set a strict deadline, and evaluate.



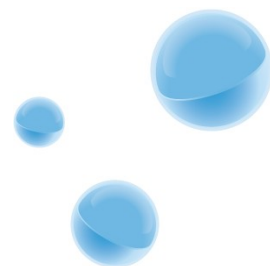
“Bad solution”

- Complex, therefor slow
- Complex, therefor prone to bugs
- Complex, therefor harder to maintain
- Introduced third party external dependency without any clear benefit.
- Not designed to fit the problem, but designed to fit the solution.



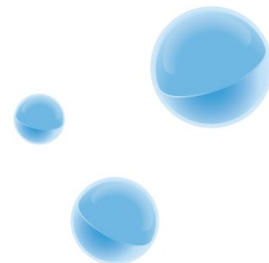
“Better” Solution

- Simple, thus fast
- Does exactly what's needed, but not more
- Designed to fit the problem, not the solution.



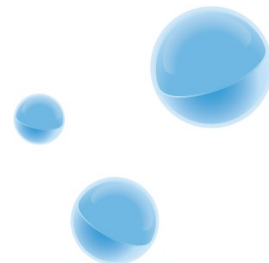
Do one thing and do it well.

NOT just a UNIX philosophy, but programming.
Anything else is scripting.



Unused code is broken code.

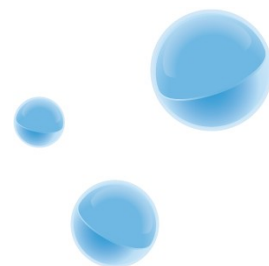
Untested code is subtly broken code.



Example: Varnish Agent 2 has 287 tests for 6k LOC.

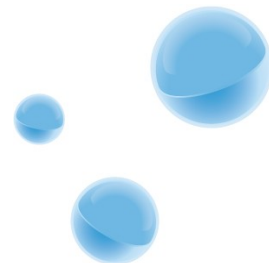
Even the simplest tests are useful.

The cost of writing a test case is negative.



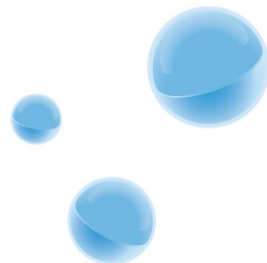
The Rise of “Worse is better”

http://www.jwz.org/doc/worse-is-better.html



Summary

- Analyze the problem without mentioning a solution
- Think up an ideal solution, without picking technology
- Look around for technology that matches or comes close to your ideal solution
- Keep whacking away at your solution until there is nothing more to remove.



Contact information

Kristian Lyngstøl

kristian@bohemians.org

http://kly.no/

@kristianlyng

https://joind.in/7982