ZOTONIC

Simple stuff that works

Making it fast!

Zotonic & performance

Erlang User Conference,
Stockholm, June 14 2013

Arjan Scherpenisse - *arjan@miraclethings.nl*

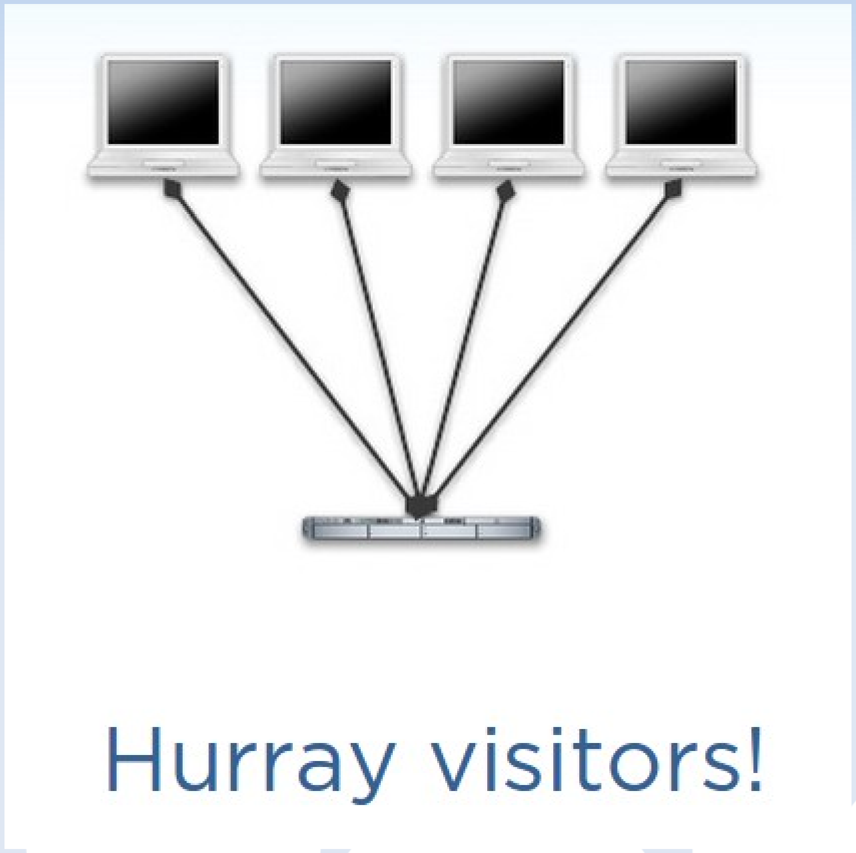
Let's make a website!

I have <? PHP ?>

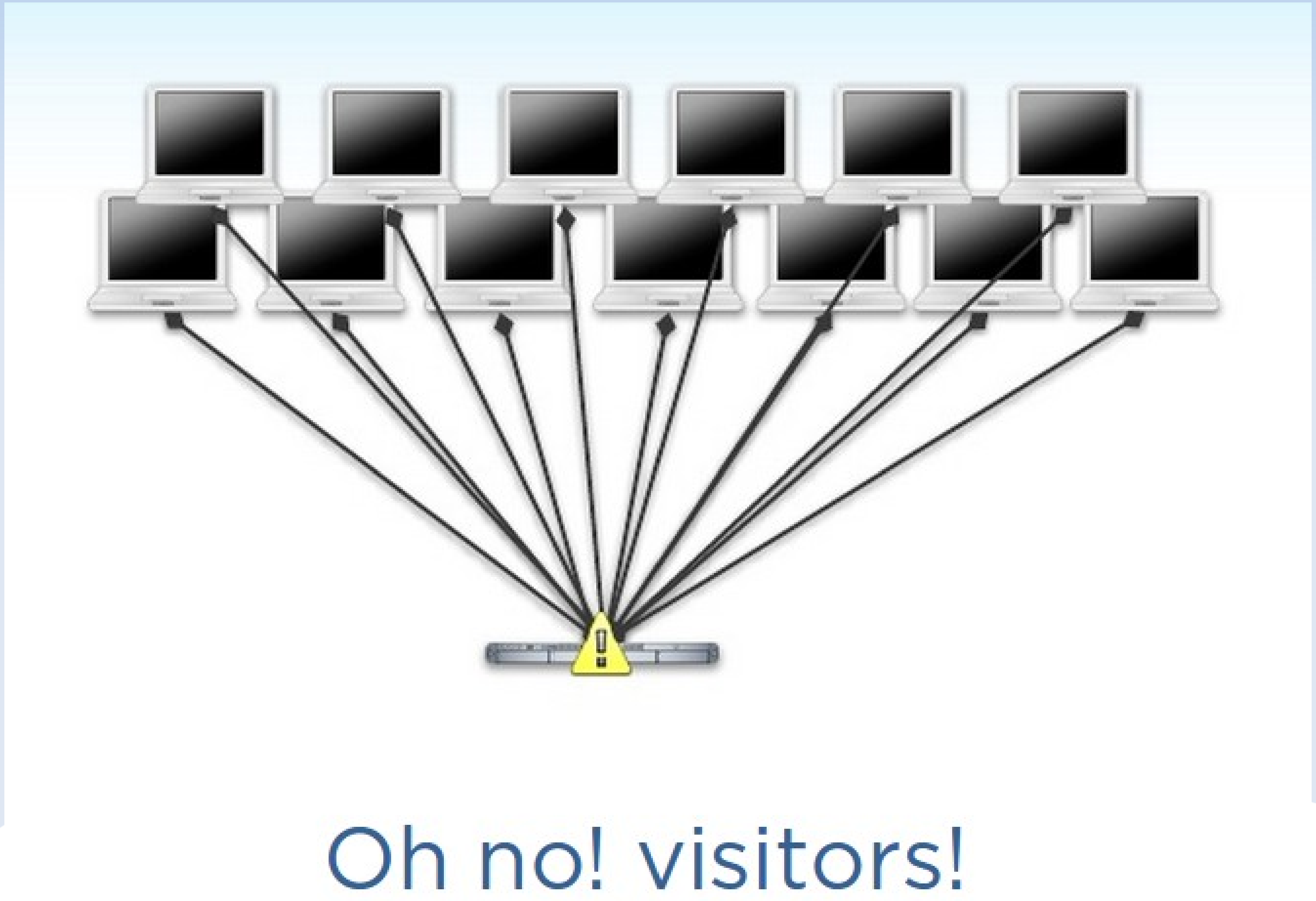
- It is on this machine.
- Everyone uses it.
- So it must be good.
- Let’s use it... (and think later)

I use <? PHP ?>

I use <? PHP ?>



I use <? PHP ?>



What happened?

- I got mentioned on popular blog
- Too many PHP+Apache processes
- Melt down

I can use PHP!

- Of course you can
- Use more hardware
- Use caching proxy
- Use xyz and a bit of abc
- Add complexity
- And keep it all running, all the time

Same for RoR, Django...

- The problem is not that you can’t scale
- The problem is that you need to scale immediately

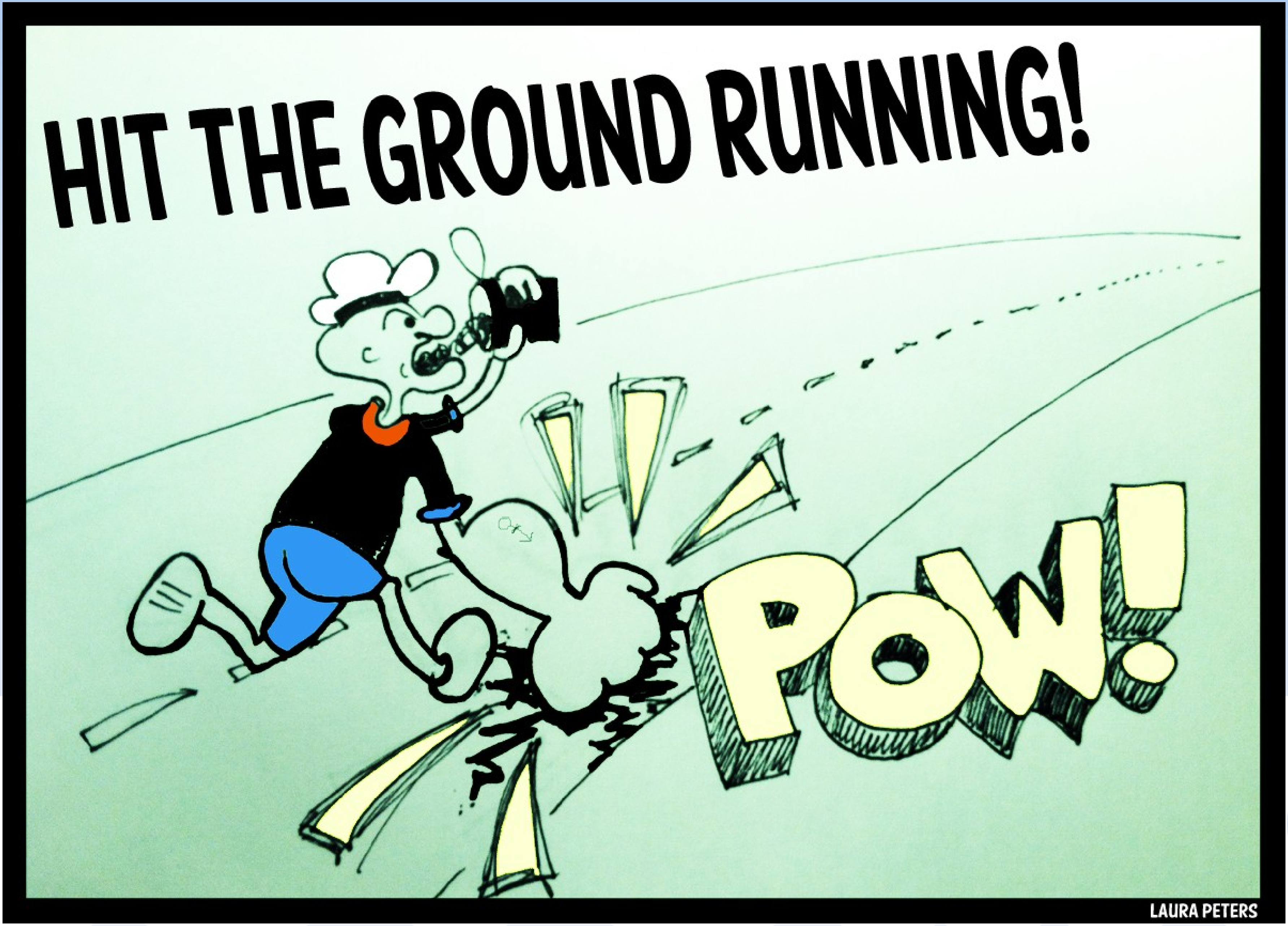
Yur site got /.'ed!

- Many people followed popular link
- A process per request
- Death by too many processes
- ... doing the same thing!

Most websites are...

- quite small
  - e.g. less than a million pages
  - except for a couple of huge ones
- not visited that much
  - e.g. less than 10 pages per second
  - Unless linked to from popular place
  - Relative small set of “hot data”

That's why we are making Zotonic.



Zotonic's goals

- The frontender is in the driver's seat
- Reliable performance
  - A web server should easily handle the load of 99% of all web sites
  - Maximise the use of hardware, do more with less hardware and less watts
- Self-contained, sysadmin friendly
  - No external services, CDN's, caching servers, background workers.., and, *no downtime*

So, what's in the box?

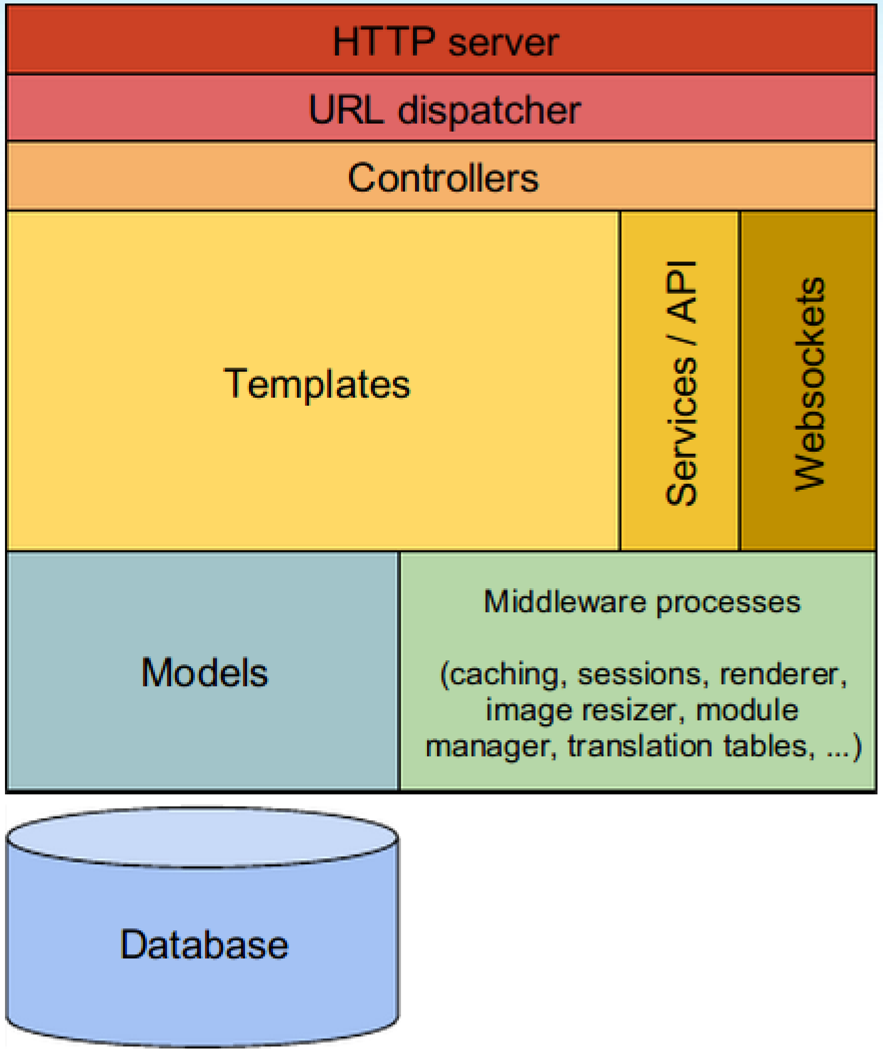
So, what's in the box?

- Well, a lot :-P

So, what's in the box?

- Multiple sites
- Admin interface
- User management
- Page management
- Menu editor
- Commenting
- Image management
- Video embedding
- i18n
- E-mail sending, receiving (!)
- Importer modules
- REST API
- …
- You name it, we (probably) got it :)

The request stack



Steps for a request

- Accept
- Parse
- Dispatch
    - (match host, URL, controller)
- Render template
    - (fetch data)
- Serve the result

Where is the time spent?

- Simple request: 7.5k/sec
- Template rendering request: 10ms
- Lots of content: a lot less :p
- Fetching data & rendering should be optimized

What takes time?

- Fetch data from database
  - Simple query roundtrip takes 1 – 10 ms
- Fetch data from caching server
  - Network roundtrip = 0.5 ms
- So: *do not hit the network or the database*

What *saves* time?

- Don't repeat things that you could have done a long time ago
- HTML escaping
- Content filtering
- (Zotonic stores sanitized / escaped content)

What *saves* time? pt II

- Combine similar (and especially *simultaneous*) actions into one
  - Requests
  - DB results
  - calculations...

Where can we save time

- Client-side caching
- Static files
- Templates
- In-memory caching

Client-side

- Let client (and proxies) cache css, javascript, images etc.
- Combine css and javascript requests:
- `http://example.org/lib/bootstrap/css/bootstrap~bootstrap-responsive~bootstrap-base-site~/css/jquery.loadmask~z.growl~z.modal~site~63523081976.css`

Static files

- File requests are easily cached
- Checks on modification dates
- Cache both compressed and uncompressed version
- Still access control checks for content (images, pdfs etc.)

Templates

- Drive page rendering
- Compiled into Erlang byte code
- Using ErlyDTL
    - Forked; we're merging it back

Template 101

```
Hello, {{ m.rsc[123].title }}

This is the id of your first image:
{{ m.rsc[123].o.depiction[1] }}

Search query:
{% for id in m.search[{query cat='person'}] %}
...
```

- Call the models – models responsible for caching those results

Template caching

```
{% include “_template.tpl” maxage=100 %}

 and

{% cache 3600 vary=z_language %}
  This gets cached per language for an hour
{% endcache %}
```

- Whole and partial caching possible
- Maxage in dispatch rules

```
{page, [“hello”], controller_template,
 [{template, “hello.tpl”}, {maxage, 3600}]}
```

In-memory caching

1) Memo cache in process dictionary of the request process
2) Central shared cache for the whole site (“depcache”)

Memo cache

- In process heap of request handler
- Quick access to often used values
- Resources, ACL checks etc.
- Flushed on writes and when growing too big

Depcache

- Central cache per site
  - ETS based
- Key dependencies for consistency
- Garbage collector thread
  - Simple random eviction
- Sharing non-cached results between processes

```
z_depcache:memo(fun() … end, 0, Context)
```

Erlang VM considerations

- Cheap processes
- Expensive data copying on messages
- Binaries have their own heap
- String processing is expensive
  - (as in any language)

Erlang VM and Zotonic

- Big data structure, #context{}
- Do most work in a single process
- Prune #context{} when messaging
  - z_context:prune_for_{database, template, async}/1
- Messaging binaries is ok

Aside: Webmachine

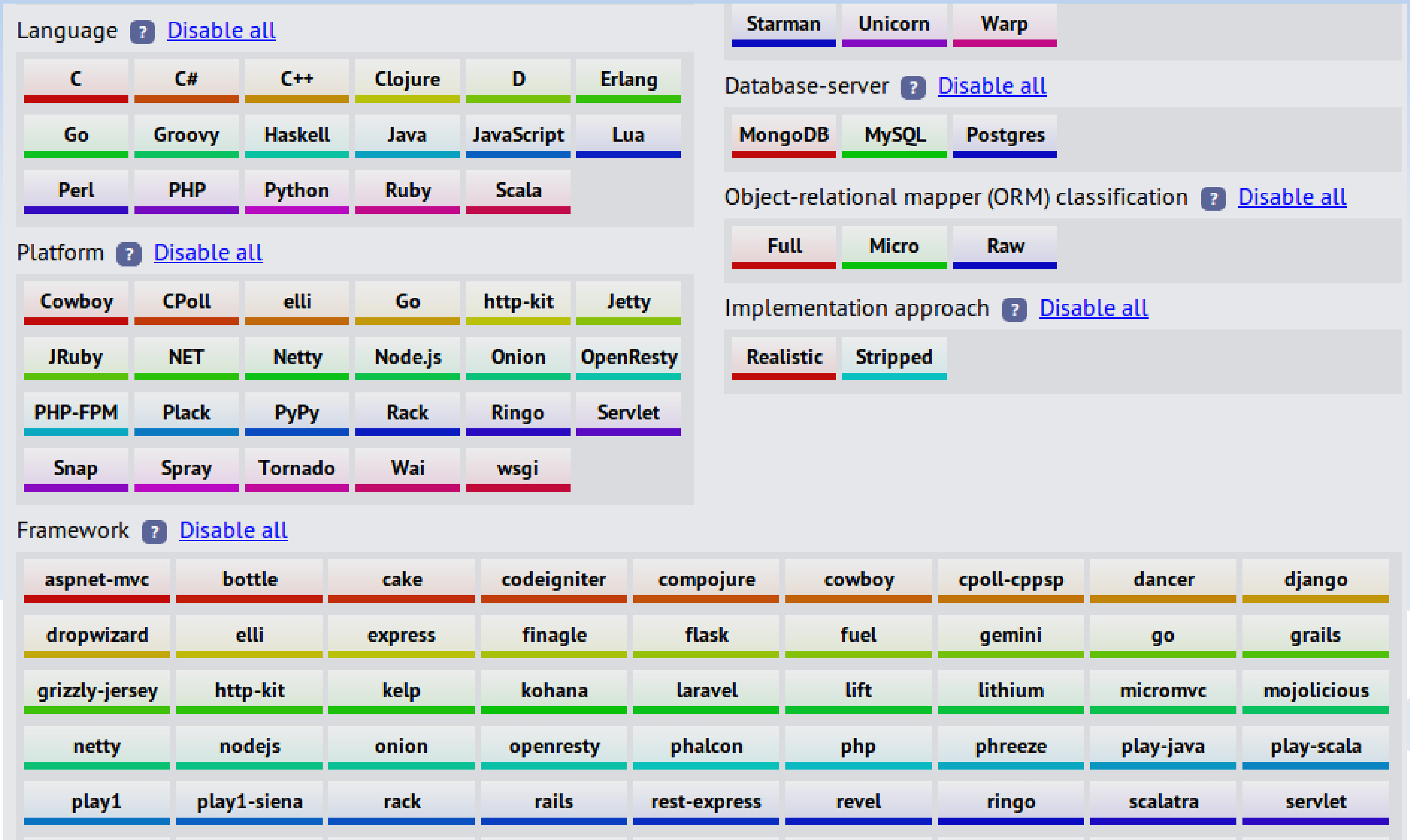
- We created a fork, webZmachine
- No dispatch list copying
- No Pmods
- Memo of some lookups
- Optimizations (process dictionary removal, combine data structures)
- Custom dispatcher (different way of treating vhosts)

Slam dunk protection

- Happens on startup, change of images, templates, memory cache flush etc.
- Let individual requests fail
- Build in artificial bottlenecks
  - Single template compiler process
  - Single image resize process
  - Memo cache – share computations
- mod_failwhale
  - Measure system load, serve 503 page, retry-after

So, what about performance?

http://www.techempower.com/benchmarks/



Language Disable all

C | C# | C++ | Clojure | D | Erlang
Go | Groovy | Haskell | Java | JavaScript | Lua
Perl | PHP | Python | Ruby | Scala

Platform Disable all

Cowboy | CPoll | elli | Go | http-kit | Jetty
JRuby | NET | Netty | Node.js | Onion | OpenResty
PHP-FPM | Plack | PyPy | Rack | Ringo | Servlet
Snap | Spray | Tornado | Wai | wsgi

Starman | Unicorn | Warp

Database-server Disable all

MongoDB | MySQL | Postgres

Object-relational mapper (ORM) classification Disable all

Full | Micro | Raw

Implementation approach Disable all

Realistic | Stripped

Framework Disable all

aspnet-mvc | bottle | cake | codeigniter | compojure | cowboy | cpoll-cppsp | dancer | django
dropwizard | elli | express | finagle | flask | fuel | gemini | go | grails
grizzly-jersey | http-kit | kelp | kohana | laravel | lift | lithium | micromvc | mojolicious
netty | nodejs | onion | openresty | phalcon | php | phreeze | play-java | play-scala
play1 | play1-siena | rack | rails | rest-express | revel | ringo | scalatra | servlet

How important are these, really?

- JSON test
  - Spit out “hello world” in json
- What are you testing?
  - HTTP parsing?
  - JSON encoding?
  - Your TCP/IP stack?

- Well, OK, Zotonic does NOT do so well...

Some numbers

Platform	x1000 req/sec
Node.js	27
Cowboy	31
Elli	38
Zotonic	5.5
Zotonic w/o logging	7.5
Zotonic w/ dispatcher process pool	8.5

i7 quadcore M620 @ 2.67GHz

```
wrk -c 3000 -t 3000 http://localhost:8080/json
```

Techempower conclusions

- We can improve some stuff
  - Compiled dispatch rule / host matching
  - Migrate to webserver that handles binaries (Elli or Cowboy)
  - Merge Webzmachine ReqData/Context params
  - Caching template timestamps – speedup freshness check
- Not every framework implements the same test.
- Pose artificial restrictions on the tests?
  - Zotonic's memory-caching is fast...

A recent project

Oranje Fonds voor sociale initiatieven Kroonappels

Oranje Fonds Over Kroonappels Tentoonstelling

Bekijk hier de mooiste sociale initiatieven van Nederland of bezoek de reizende tentoonstelling.

Oranje Fonds voor sociale initiatieven

De zoektocht naar de mooiste sociale initiatieven van ons land

Kroonappels

- Nation-wide voting weekend
- Client requested 100% availability + high performance
- 100k “votes” in 1 hour

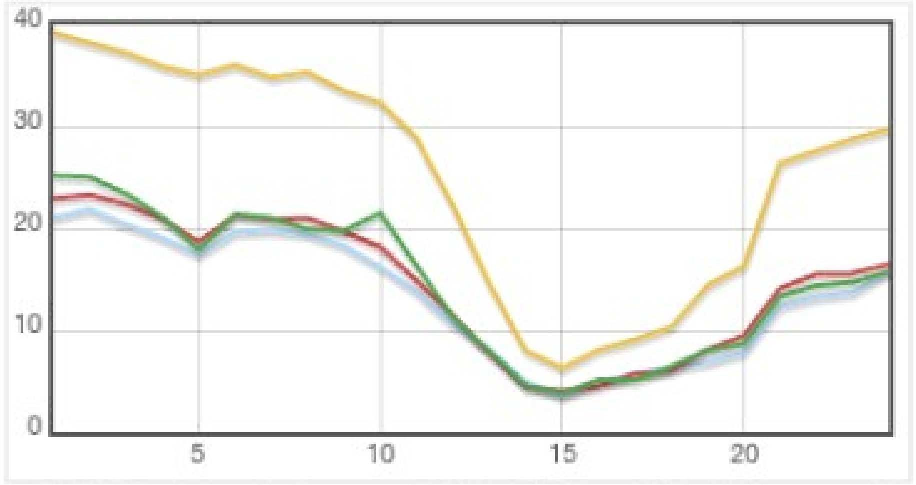
- 3x Rackspace VPS nodes, 2 GB, load balanced

Kroonappels

- 1 vote was about 30 requests
  - Dynamic i18n HTML
  - Ajax
  - Static assets
- Load test needed adjustments
- Did not push to the max
  - Stopped at 500k votes / hr; 1.5M req/hr
  - Customer satisfied :-)

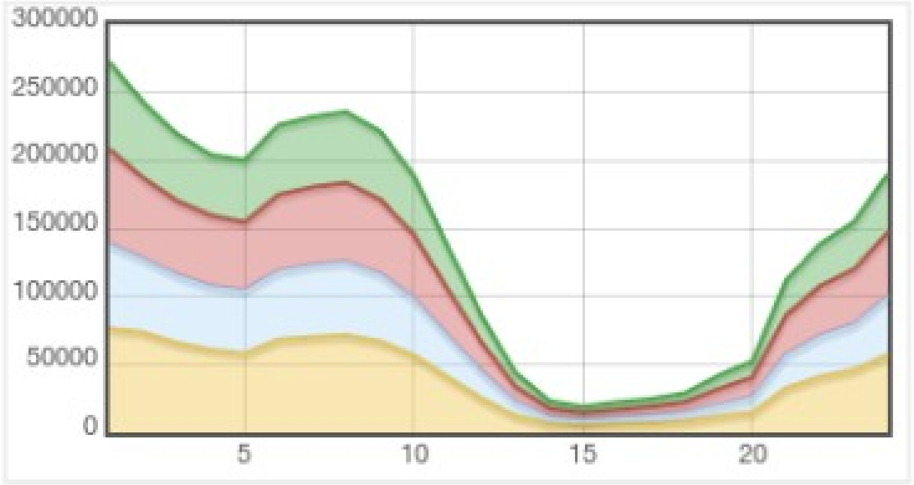
zotonic001@node0.projectx.zynamo.nl zotonic001@node1.projectx.zynamo.nl zotonic001@node2.projectx.zynamo.nl zotonic001@node3.projectx.zynamo.nl

Database - query time

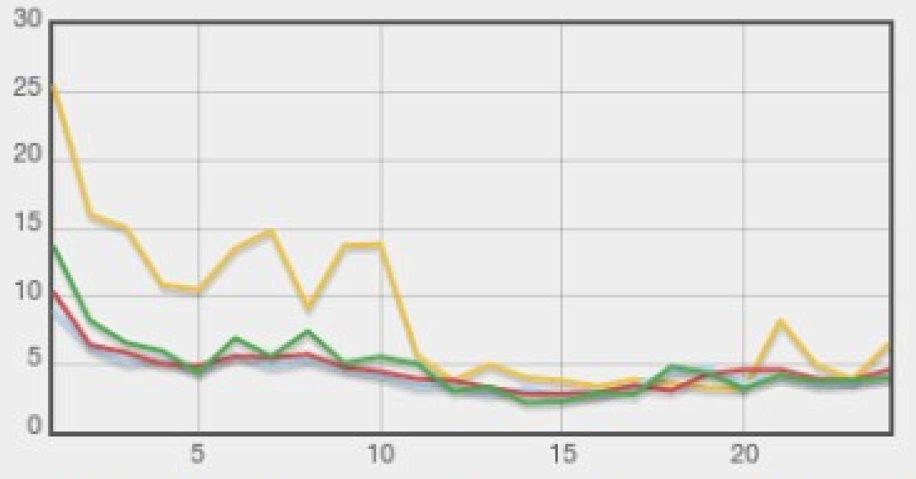


zotonic001@node0.projectx.zynamo.nl: 99% < 350.1 ms, 95% < 197.8 ms, avg: 18.6 ms
zotonic001@node1.projectx.zynamo.nl: 99% < 238.4 ms, 95% < 108.7 ms, avg: 10.8 ms
zotonic001@node2.projectx.zynamo.nl: 99% < 293.6 ms, 95% < 190.9 ms, avg: 11.3 ms
zotonic001@node3.projectx.zynamo.nl: 99% < 242.4 ms, 95% < 17.3 ms, avg: 11.6 ms

Database - nr. of requests

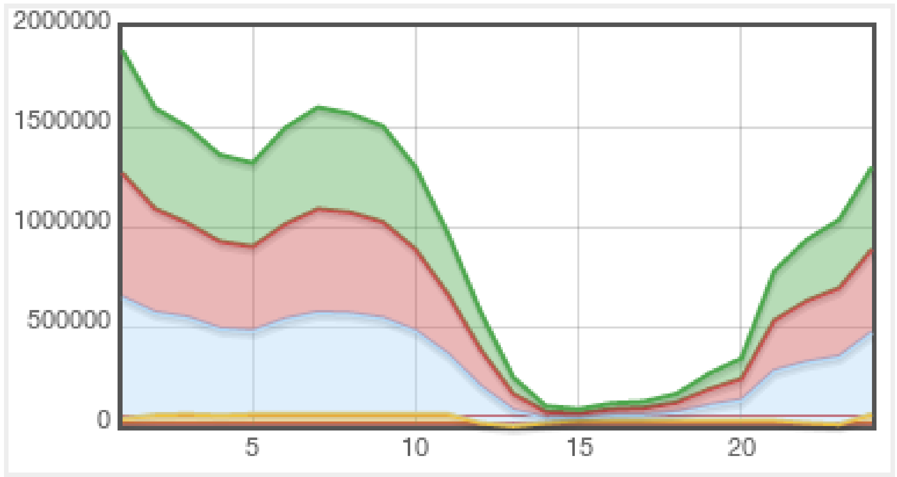


Web server - request time



zotonic001@node0.projectx.zynamo.nl: 99% < 3869.4 ms, 95% < 12.7 ms, avg: 61.5 ms
zotonic001@node1.projectx.zynamo.nl: 99% < 128.7 ms, 95% < 6.3 ms, avg: 5.1 ms
zotonic001@node2.projectx.zynamo.nl: 99% < 139.2 ms, 95% < 6.5 ms, avg: 4.9 ms
zotonic001@node3.projectx.zynamo.nl: 99% < 129.0 ms, 95% < 6.5 ms, avg: 5.4 ms

Web server - Kb out

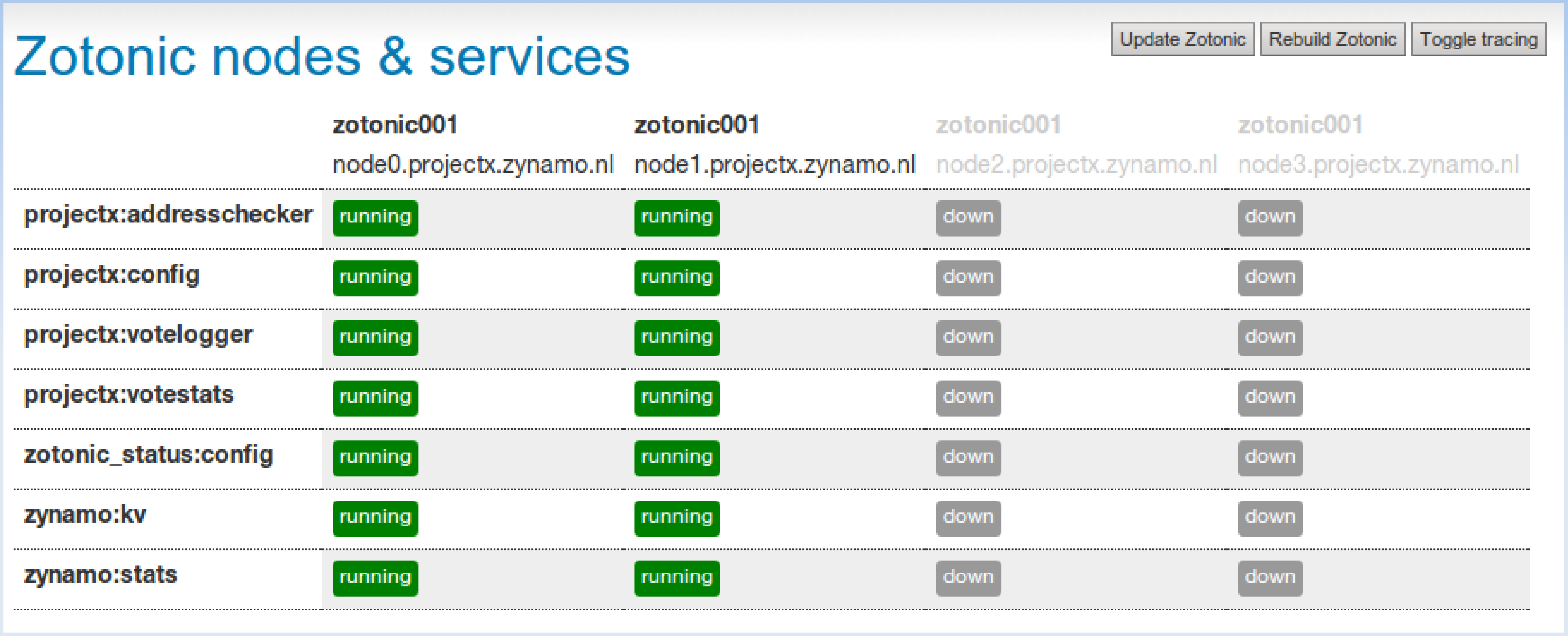


zotonic001@node0.projectx.zynamo.nl: 99% < 83.67 Kb, 95% < 52.15 Kb, avg: 10.79 Kb
zotonic001@node1.projectx.zynamo.nl: 99% < 32.11 Kb, 95% < 12.31 Kb, avg: 1.93 Kb
zotonic001@node2.projectx.zynamo.nl: 99% < 32.11 Kb, 95% < 12.31 Kb, avg: 1.94 Kb
zotonic001@node3.projectx.zynamo.nl: 99% < 32.11 Kb, 95% < 12.31 Kb, avg: 1.93 Kb

Kroonappels – made with Zynamo

- Data layer
  - Distribution ring based on Dynamo principles
  - Consistent hashing, work distribution
  - Service architecture w/ GET/PUT/DELETE semantics
  - Like riak_core without vnodes

Service oriented



Zotonic nodes & services

	zotonic001 node0.projectx.zynamo.nl	zotonic001 node1.projectx.zynamo.nl	zotonic001 node2.projectx.zynamo.nl	zotonic001 node3.projectx.zynamo.nl
projectx:addresschecker	running	running	down	down
projectx:config	running	running	down	down
projectx:votelogger	running	running	down	down
projectx:votestats	running	running	down	down
zotonic_status:config	running	running	down	down
zynamo:kv	running	running	down	down
zynamo:stats	running	running	down	down

Zynamo's downside

- Hard...
  - to maintain,
  - to do caching
  - to write new stuff
  - there are DBMS's that can do this for us

- Got us thinking: Do we really need this scale?

What do we want?

- Multiple machines, but for error recovery
  - Hardware errors
  - Hardware upgrades
- Hot failover

The P2P idea

- Trusted P2P ring of collaborative Zotonic machines
- Reliable messaging / notification
  - *Poldercast* P2P model
- Synced database backups / assets
  - Bittorrent protocol for large files
  - WAL for db delta's
- Sites are vertical, data silo's
- Run our own DNS?

Thank you!

- Book chapter: “The performance of Open Source Applications” coming out soon (http://www.aosabook.org/)
- …and chat with me & Andreas :-)
    - Come get a tshirt!
- Online resources:
    - http://zotonic.com
    - @zotonic - http://twitter.com/zotonic
    - IRC, XMPP, mailing lists
    - Talk slides, tutorial slides, tutorial source code...