

Database Driven Cache Invalidation

JDCon-West, 2010
San Francisco, CA

Magnus Hagander
magnus@hagander.net



DrakeCon2010

www.drakecon.org

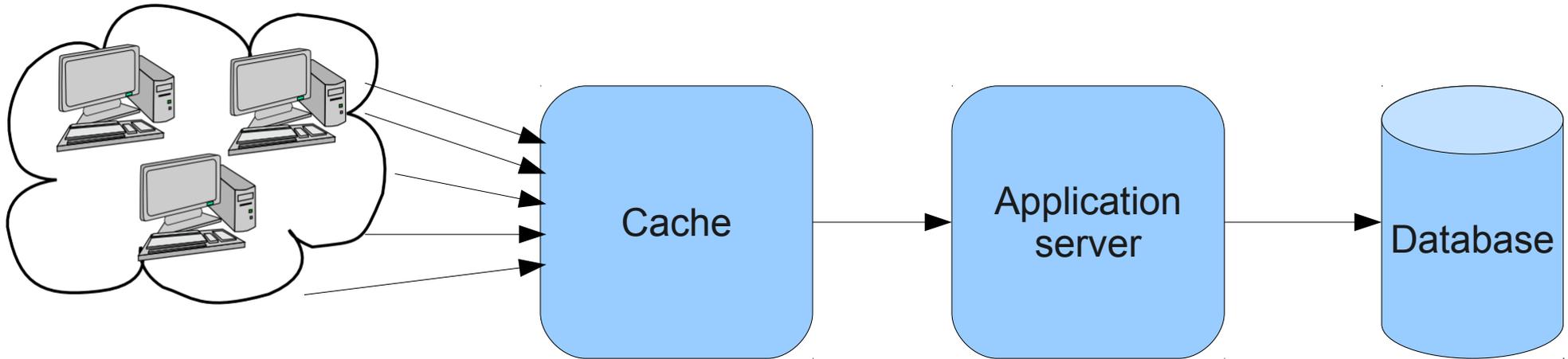
What's this all about

- We like to build advanced websites
- We want them to be popular
- Hopefully they become popular
- They fall over
 - So we add caching
 - Which adds new issues

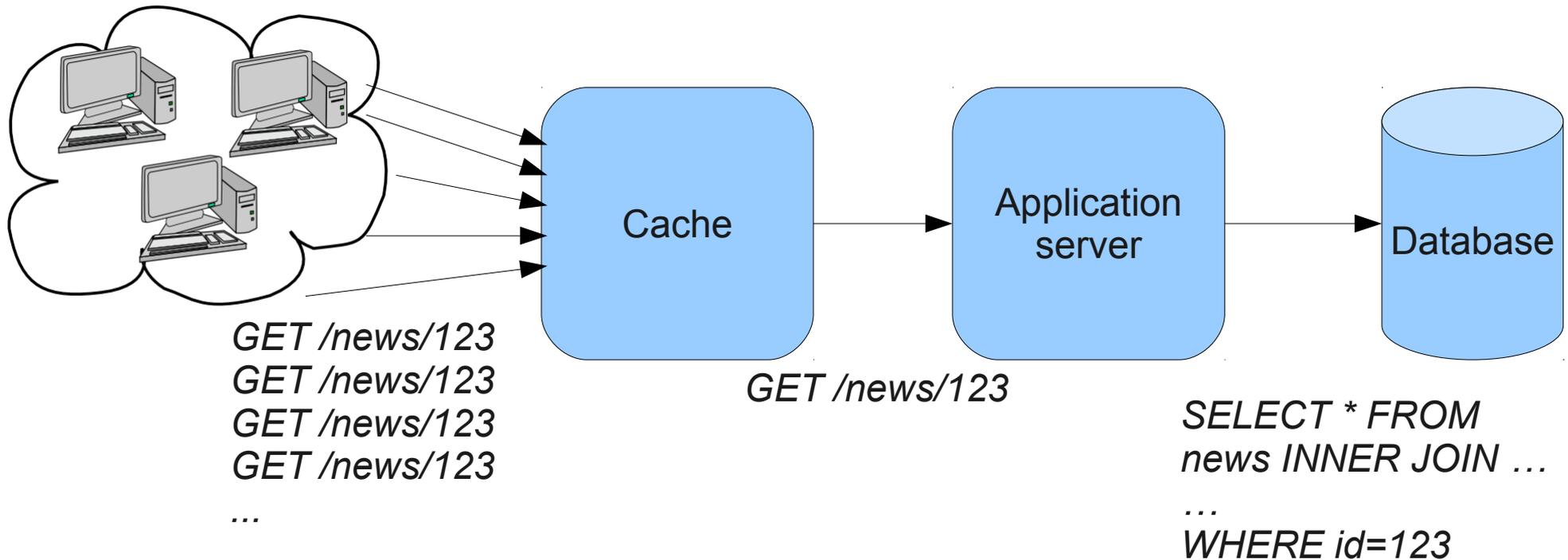
Where to cache

- Many different locations to cache:
 - In the database server (buffer cache)
 - In the database server (“query cache”)
 - In the application server
 - Before the application server (“http cache”)
 - In the client
- The closer to the client, the more efficient

Typical architecture



Typical architecture



In this presentation

- Let's build a blog!
- A really simple one!
- But we're popular! (right?)

Intro to the application

- We'll use PostgreSQL (d'uh)
- We'll use *django*
 - Theory is framework independent
 - Django-admin makes life easy...
- We'll use Varnish

Let's create the application

```
$ django-admin startproject demo
```

```
$ manage.py startapp blog
```

- Configure our database connection
- Enable the django admin site
- Add our application

Create a simple model

- We're just going to hold blog posts
- Title, date/time, and contents
- And add it to the admin site
- Sync the database



Create simple views

- One view to list all blog posts
 - Served up as /
- One view to show the posts
 - Served up as /<blog post id>/
- (let's ignore nice URLs and CSS for now)



Try it out

- Using local django web server
\$./manage.py runserver
- Each click causes appserver call
- Which in turn causes db query



Add varnish to the mix

- Very simple default install:
 - Relay all URLs to localhost:8000
 - Cache all pages for 1 hour
 - Ignore cookie-related issues
 - (this is not a varnish session, after all...)



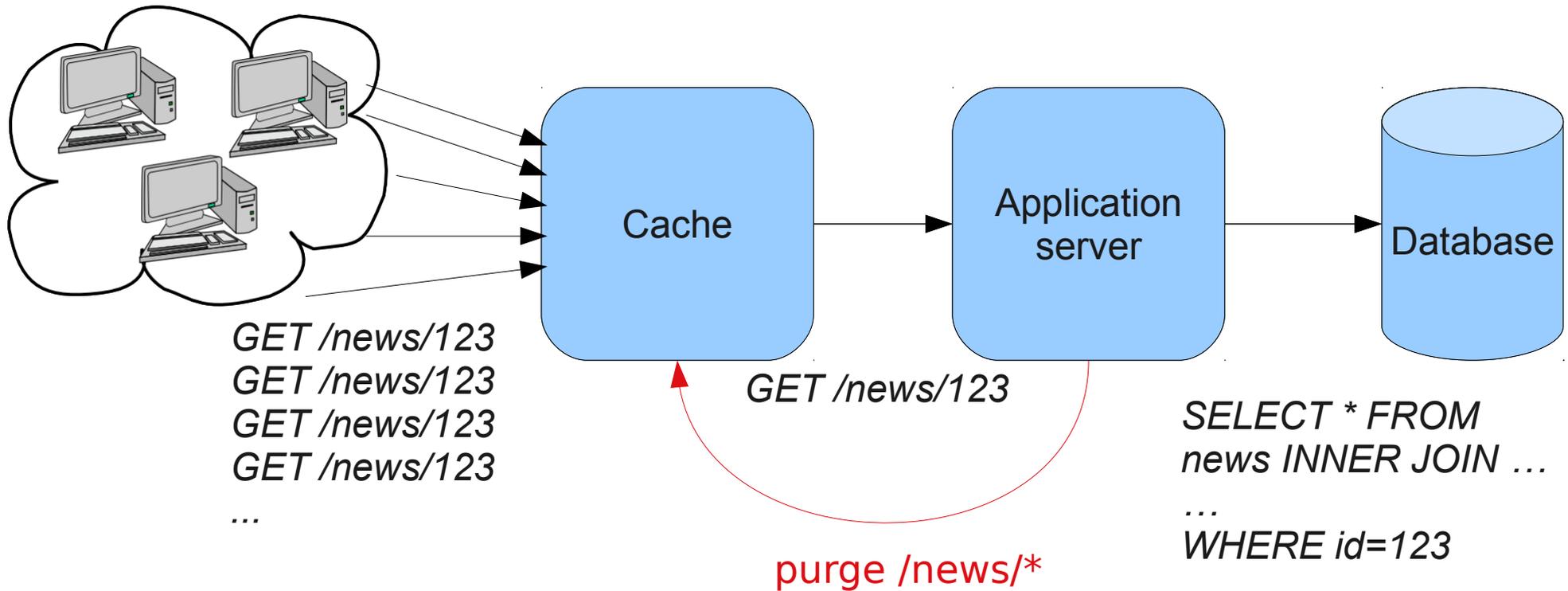
Issues with caching

- Some content rapidly go stale
 - Solution: cache only a short time!
- Some content is very long-lived
 - Solution: cache a long time!
- Sometimes unpredictable
 - Per-url or per-request cache values become sub-optimal

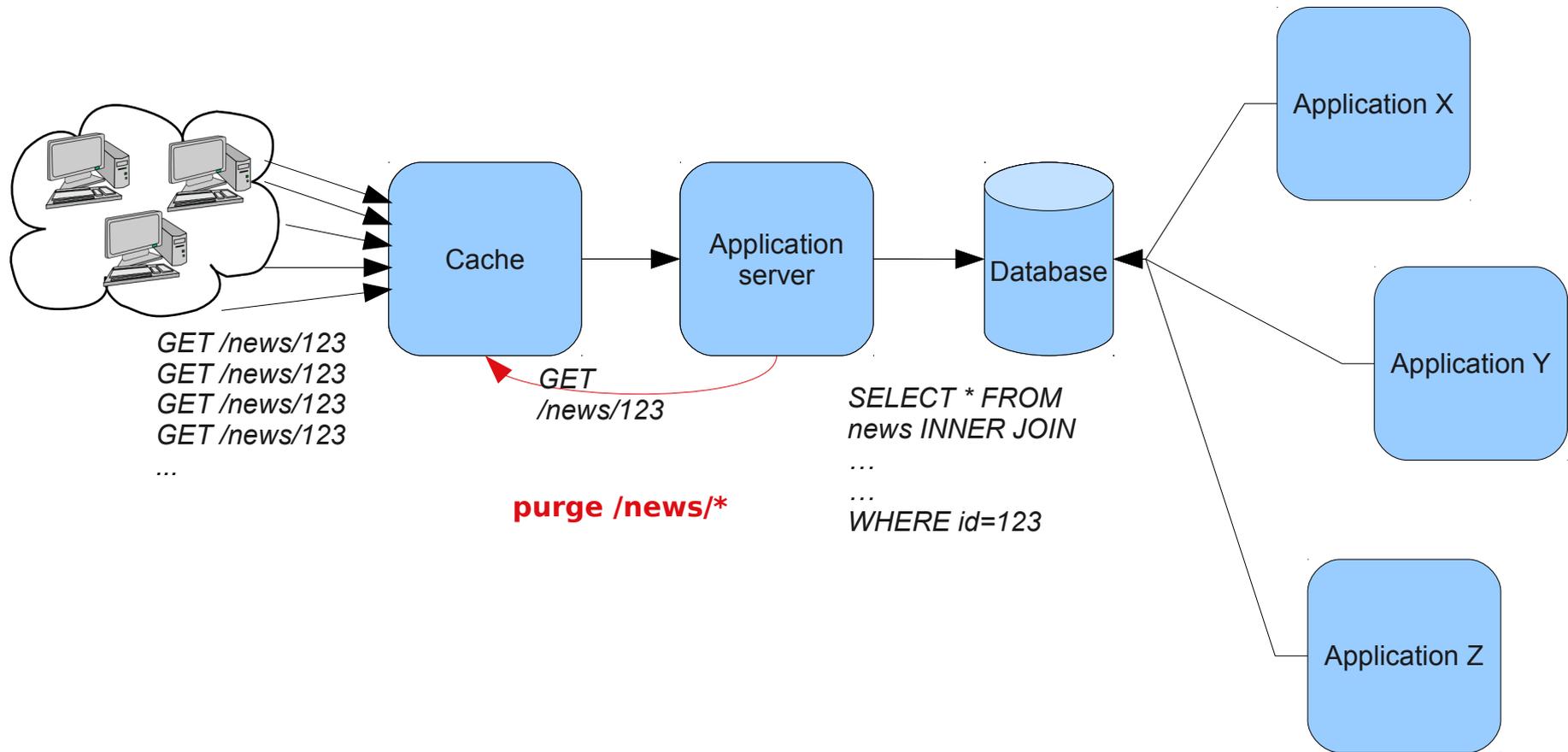
On-demand invalidation

- Cache most objects for a long time
- Explicitly remove them from the cache
 - When object modified in app
 - When dependent object modified in app
 - ...

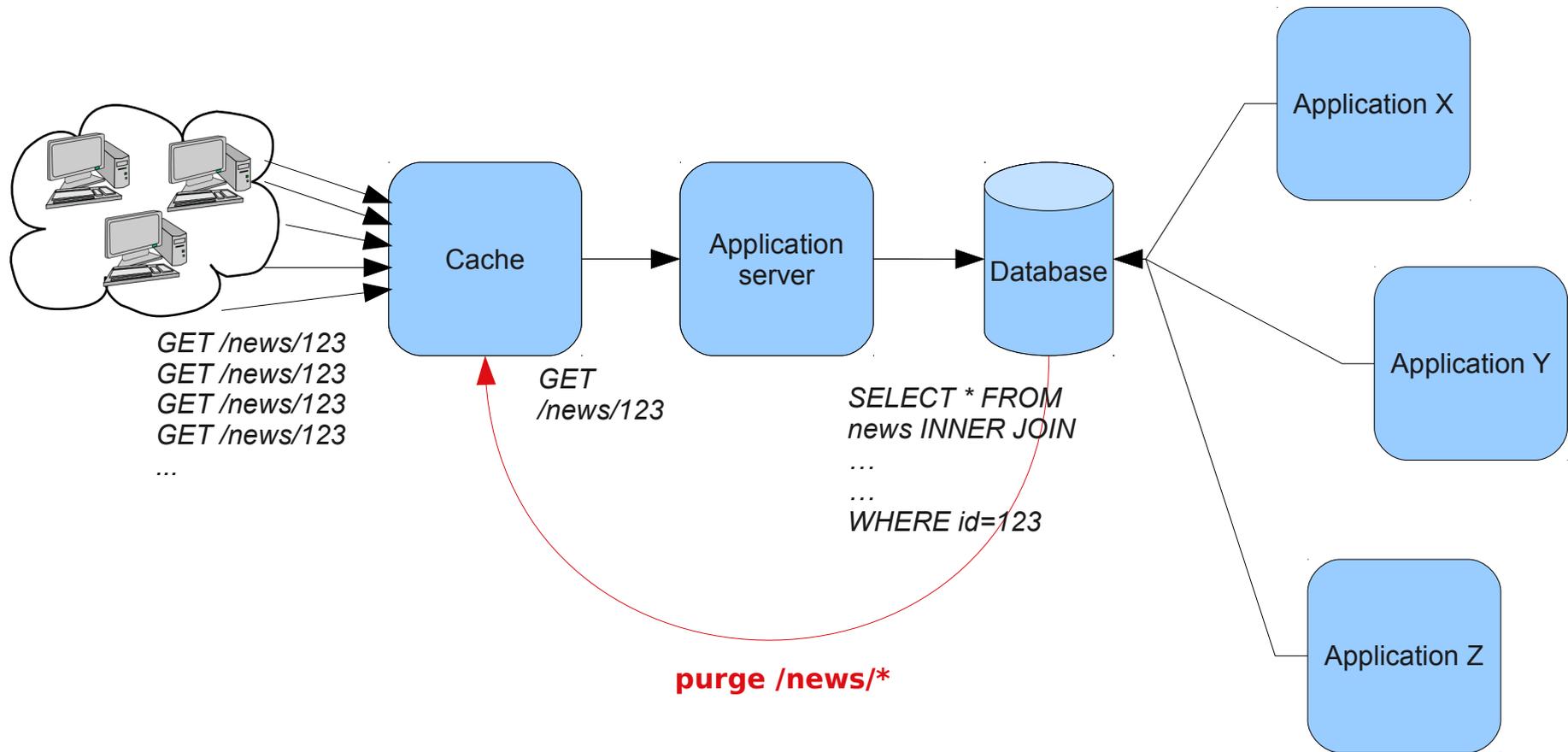
Cache invalidation



What about other apps!



Invalidate from the db!



Ups and Downs

- Upside
 - Cache invalidation when relevant *data* changes
 - Regardless of origin of change
- Downside
 - Database needs to gain URL knowledge
 - Breaks “clean abstraction model”

Invalidation: method 1

- Create a trigger on the blog_post table
- Have it send off a varnish purge request



Invalidation: method 2

- Create a trigger on the blog_post table
- Trigger inserts the request in a queue table
- Trigger on the queue table fires a NOTIFY
- Daemon listens to notifies and sends purge requests

Invalidation: method 3

- We just created a trivial queue
- There are ready-made queues out there
- Pq!
- High performance and efficient
- Offload work of dealing with multiple caches etc

Install pgq

- Install pgq in the database
- Create our queue
- Start a “ticker” process



Write simple consumer

- Consumes events from pgq
- Generates varnish purge requests
- Run one consumer for each varnish server



Pgq hand-holding

- Needs almost nothing
- But don't forget to monitor it!
- E.g. munin plugin

Final words

- Keep setting cache expiry times
 - s-maxage etc
- Then use purging only when needed
- You don't want thousands per second

Thank you!

Questions?

Twitter: @magnushagander
<http://blog.hagander.net/>
magnus@hagander.net



DrakeCon2010

www.drakecon.org