Building a Web-Scale Search Engine with Perl

Greg Lindahl, CTO, blekko

greg@blekko.com - @glindahl - wumpus

Waah

- This is my first YAPC
- They scheduled me against Rick?!

Agenda

- Personal histories
- The Search Business in 3 minutes
- Why we chose Perl
- Writing NoSQL in Perl
- The search engine app – event-driven progr.
- Continuous Everything
- Updating perl (and CentOS)
- Open Data / Open Source
- Our secret sauce

Personal Histories

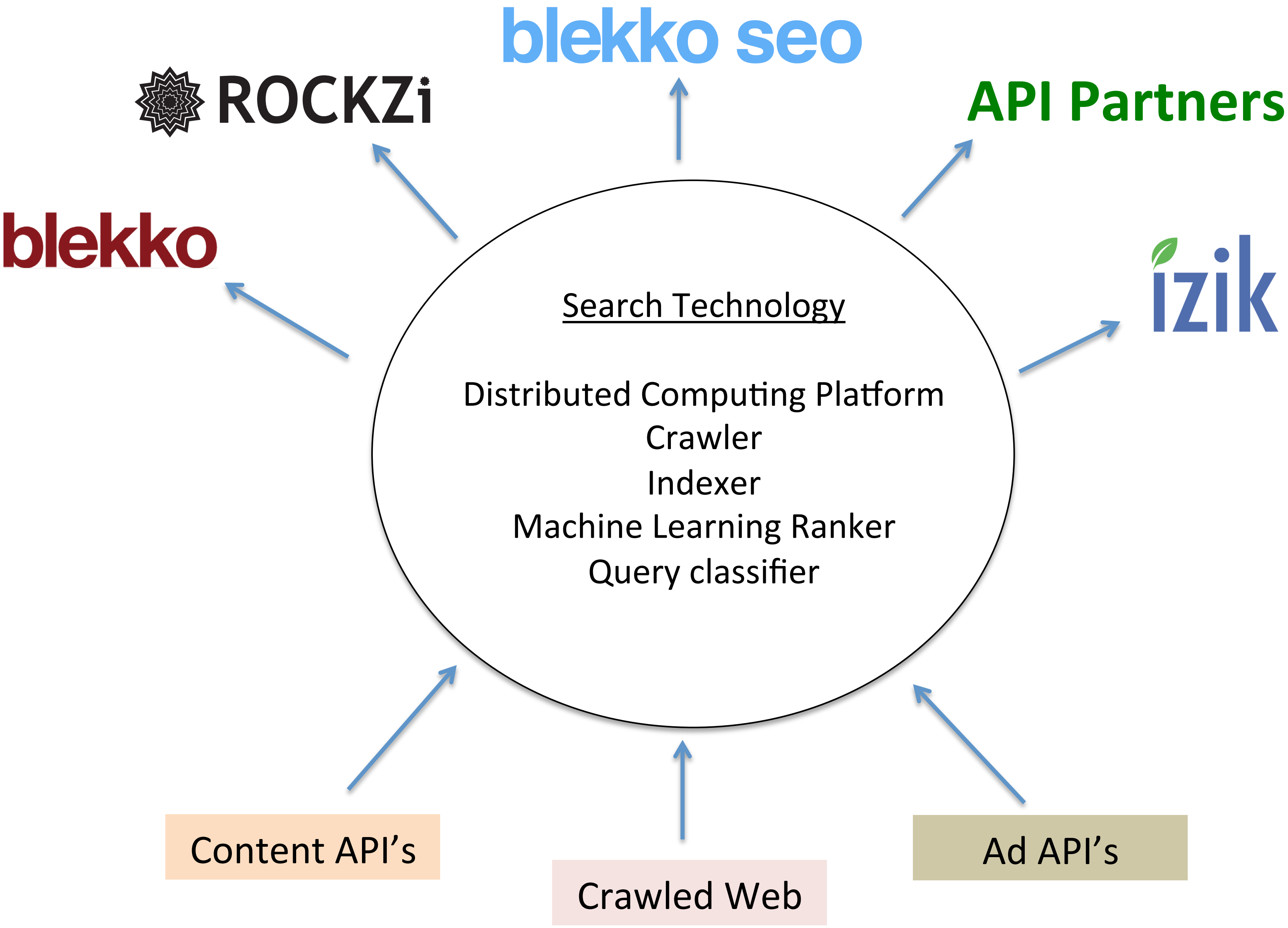
- me: icon, usenet? or irc?, merlyn, camel book
- http://www.pbm.com/~lindahl/
- really a supercomputing guy: Fortran, MPI
- Contrib to ircd (founded efnet), binutils, emacs

- others: GnuHoo => NewHoo => ODP / dmoz
- Netscape => AOL => AOL-TimeWarnerMegaCorp
- Popular open data dataset, inspired Wikipedia
- All of us know a lot of other languages

The Search Business in 3 minutes

- Some people say building a real search engine costs $1 billion
  - and by real, I mean multi-billion-page crawl and index, not using Google/bing’s index
- Recent “real” failures: cuil, SearchMe; they raised ~ $30-40 million, hire 80-100 people
- The only successful new “real” engine since Google is the bing re-write! ($1B loss/year)
- We felt real innovation was possible only if we had our own crawl and index

What our marketing team says we built



blekko seo

ROCKZi

API Partners

blekko

izik

Search Technology

Distributed Computing Platform
Crawler
Indexer
Machine Learning Ranker
Query classifier

Content API’s

Crawled Web

Ad API’s

Our plan

- Build the basics of a search engine
- Try some innovations, most of which will fail
- Don't die on launch day, have a long runway
- Try some innovations, most of which will fail
- …
- Profit!

- (we didn't really have a plan)
- Keep the team small: build an environment that makes programmers efficient

Choosing perl

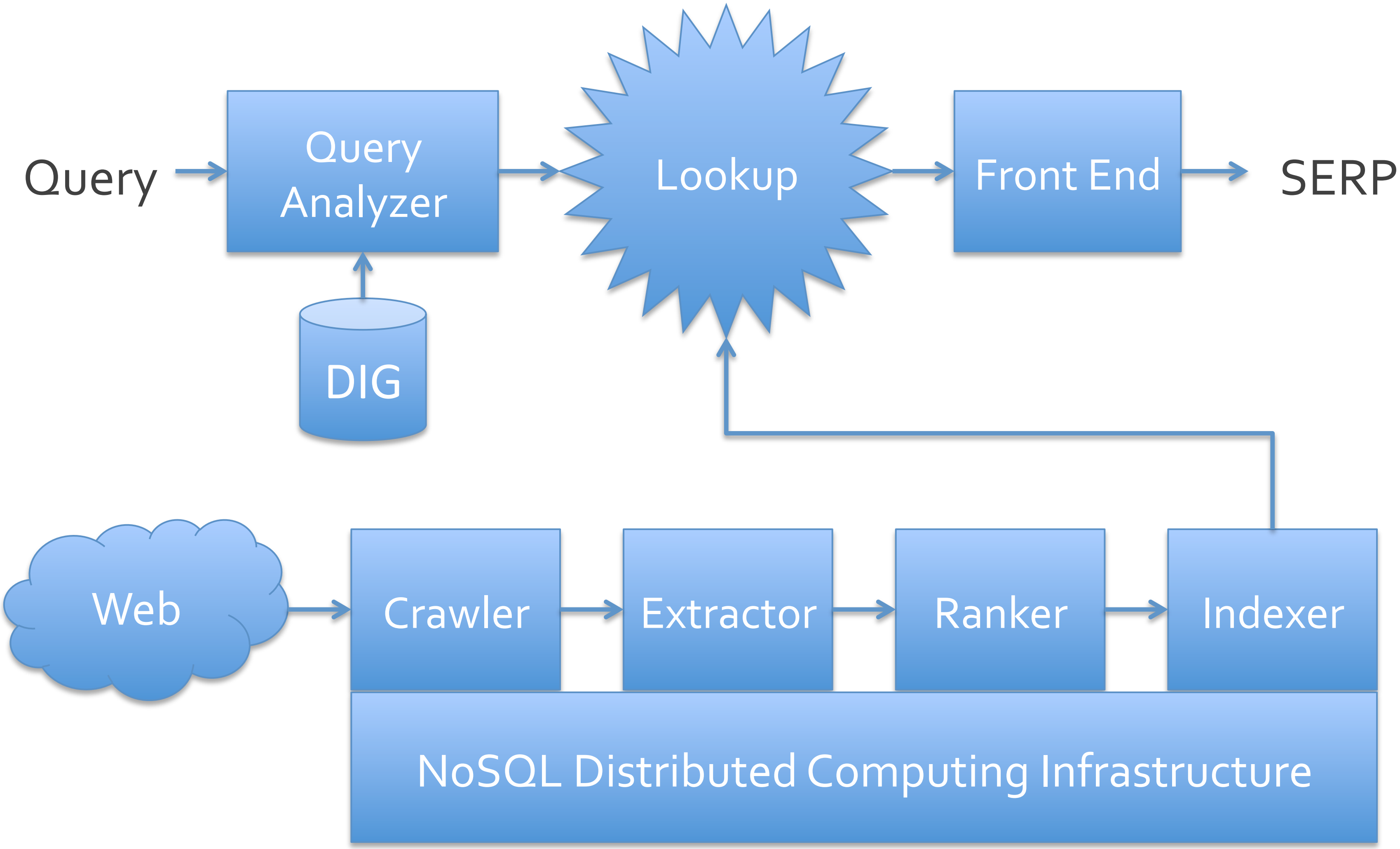
- “Maybe we should switch to Python
  - basically the same language
  - easier to hire”
- Went and bought Python books
- 1 week later: “Anyone read their book?”

- Allrighty, Perl it is.

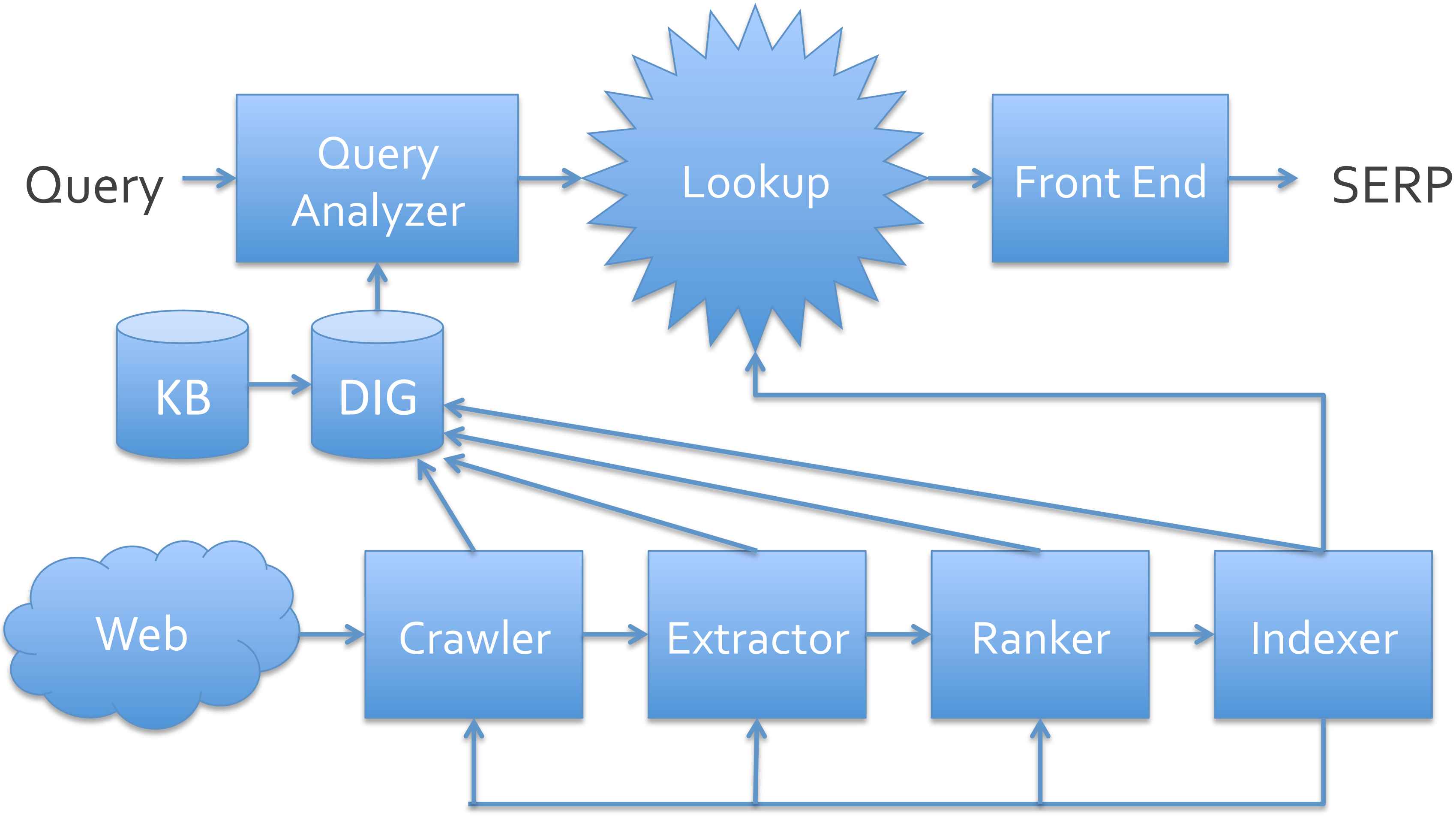
How did that work out?

- Hired some people who loved perl and wanted to work for an awesome startup
- Hired lots of smart people who didn’t know perl, but knew Python/Ruby or Java
- No complaints about “resume damage”
- No problem hiring during the Silicon Valley hiring crunch
- We succeeded in making our programmers very productive

A Search Engine



The Real Diagram – lots of feedback



NoSQL

- “Let me explain. No, there’s not time. Let me sum up.”
- …
- Lots of existing NoSQL alternatives, but
  - We want a bunch of unusual features that other people wouldn’t care about – and we’d be extreme users overall
  - Choice: understand all the code, or write all the code
  - Plus, all that existing stuff is in Java. Screw that.

NoSQL in Perl

- Lots of great building blocks
  - IO::AIO
  - JSON::XS – hacked into our own solution
  - AnyEvent
  - … 397 more CPAN distributions
- Need: Map/Reduce
- Need: a way of expressing efficient transactions

Map/Reduce without the Reduce

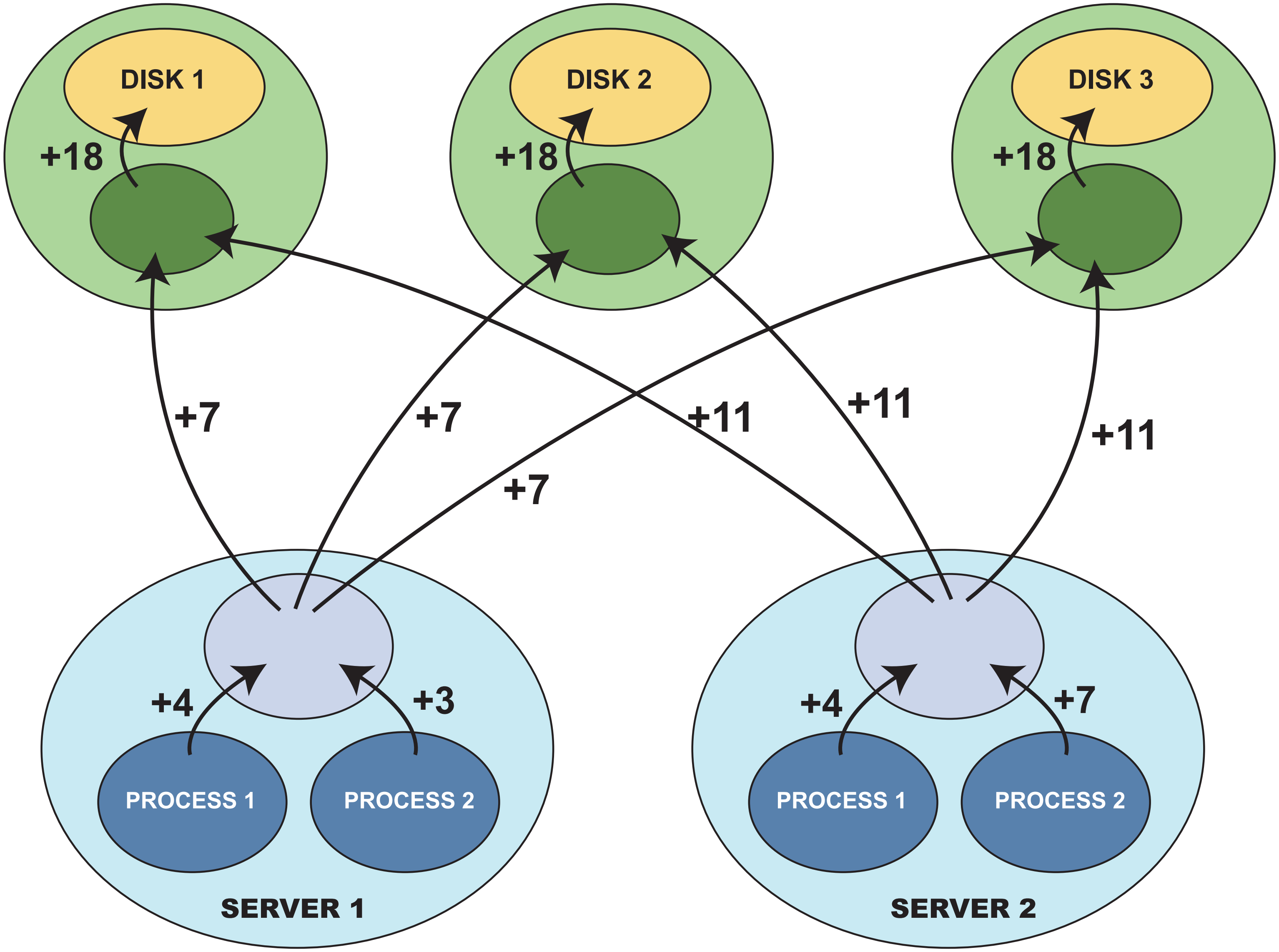
```
sub foo {
  return if $pm->MAPJOB();
  while ( ($url, $doc) =
           $pm->nextrow('/crawl', $url) ) {
      foreach word in doc
        $pm->add("/wc", $word,{ count=>1 } );
  }
}
```

- No reduce phase
- Writes to the database are made through combinators (comb_add in this case)

Combinators?

- Transactional data structures
  - add, min, max, delete, average, …
- Associative: Can combine before delivery
  - In process, before they’re sent to the local write daemon
  - Writer daemon, before delivery to buckets
  - Bucket daemon, before delivery to disk
- Preferably commutative
  - who’s first in a cluster?
- They allow sub-cell atomic updates
- Minimizes operations at the disk

Combinators reduce the total work



TopN combinator

- table: /crawl/32/url
- row: blekko.com/, column: inlinks

rank	key	anchortext
900	nytimes.com	new search engine blekko
540	techcrunch.com	blekko removes spam
1	www.ehow.com/blekko	our enemy

Writing the search engine app

- Sits on top of the NoSQL stuff
- crawl => extract => index
- Needs a lot of event-driven programming
  - because we aren’t using threads, no way
- Needs to be integrated with the NoSQL stuff, network outcalls
- Must allow pretty code

Expressing Dataflow

```
my @work = generate_work();
PM::FrameWork::do_work( \@work, \& first,
                        ( framecount => 5, fps => 1 ) );
sub first {
   my ( $frame, $work ) = @_;
   get_page( $work, $frame->{page} );
   return \& second;
}
sub second {
   my ( $frame ) = @_;
   … do something with $frame->{page} …
   return undef;
}
```

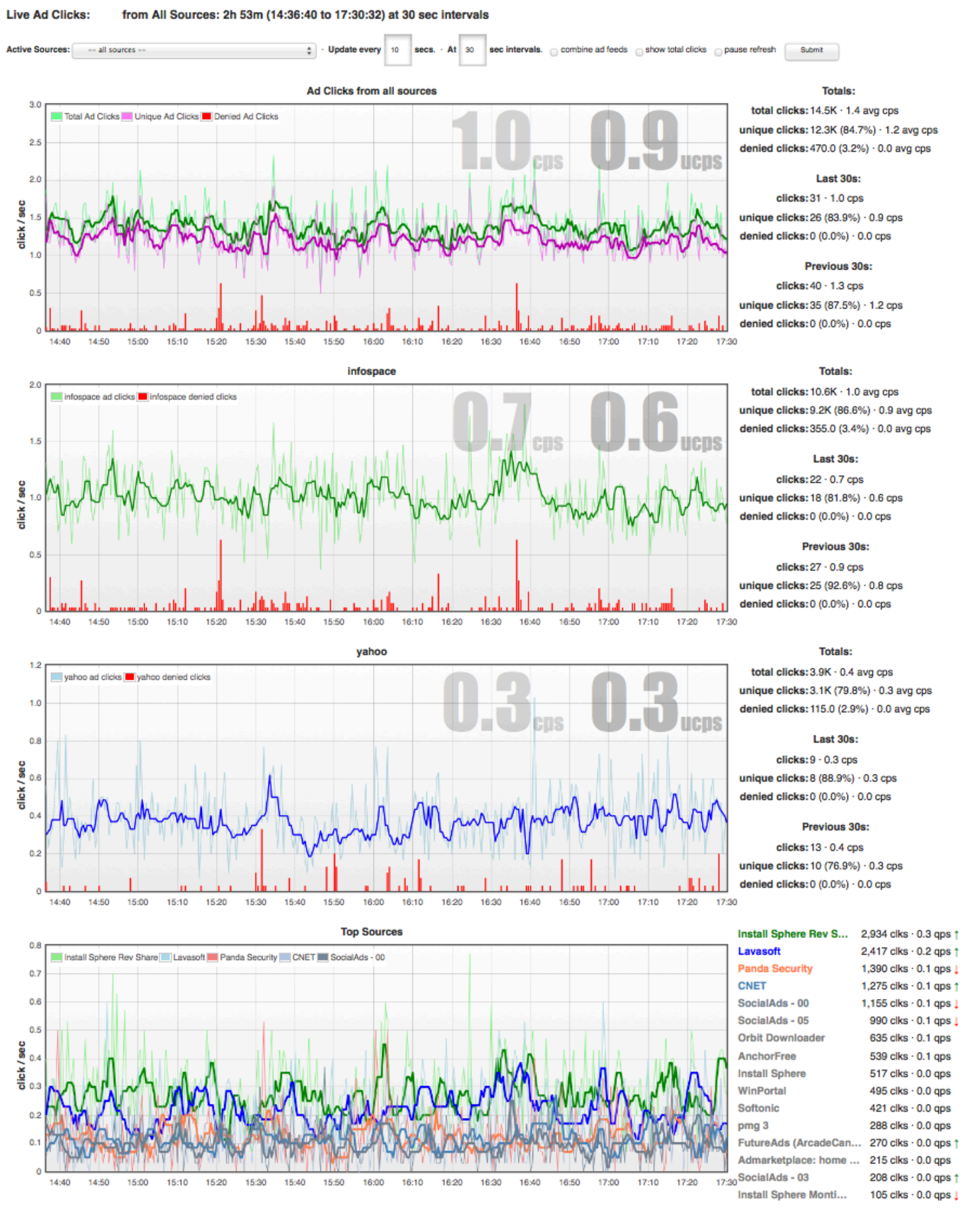
Dataflow good and bad

- Good: very efficient, great cache patterns
  - our crawlers frequently have 1000 active frames per single-threaded process
- Slightly bad: have to run multiple heavy-weight processes to use multiple cores
  - because perl is single-threaded
  - not a big deal for us, all of our daemons are parallel
  - multiple crawlers/box, 1 set of disk daemons for every disk etc.

Continuous Everything

- Continuous Integration
  - was Jenkins, now homegrown
  - end-to-end testing catches most disasters
- Continuous Deployment
  - want to push the webserver multiple times a day
  - want to push the NoSQL daemons every few months
  - no downtime for search users
- Continuous monitoring
  - Nagios and status displays

Show me the money: ad clicks



Live Ad Clicks: **from All Sources: 2h 53m (14:36:40 to 17:30:32) at 30 sec intervals**

Active Sources: -- all sources -- · **Update every** 10 **secs.** · **At** 30 **sec intervals.** □ combine ad feeds □ show total clicks □ pause refresh

Ad Clicks from all sources

Totals:

total clicks: 14.5K · 1.4 avg cps

unique clicks: 12.3K (84.7%) · 1.2 avg cps

denied clicks: 470.0 (3.2%) · 0.0 avg cps

Last 30s:

clicks: 31 · 1.0 cps

unique clicks: 26 (83.9%) · 0.9 cps

denied clicks: 0 (0.0%) · 0.0 cps

Previous 30s:

clicks: 40 · 1.3 cps

unique clicks: 35 (87.5%) · 1.2 cps

denied clicks: 0 (0.0%) · 0.0 cps

infospace

Totals:

total clicks: 10.6K · 1.0 avg cps

unique clicks: 9.2K (86.6%) · 0.9 avg cps

denied clicks: 355.0 (3.4%) · 0.0 avg cps

Last 30s:

clicks: 22 · 0.7 cps

unique clicks: 18 (81.8%) · 0.6 cps

denied clicks: 0 (0.0%) · 0.0 cps

Previous 30s:

clicks: 27 · 0.9 cps

unique clicks: 25 (92.6%) · 0.8 cps

denied clicks: 0 (0.0%) · 0.0 cps

yahoo

Totals:

total clicks: 3.9K · 0.4 avg cps

unique clicks: 3.1K (79.8%) · 0.3 avg cps

denied clicks: 115.0 (2.9%) · 0.0 avg cps

Last 30s:

clicks: 9 · 0.3 cps

unique clicks: 8 (88.9%) · 0.3 cps

denied clicks: 0 (0.0%) · 0.0 cps

Previous 30s:

clicks: 13 · 0.4 cps

unique clicks: 10 (76.9%) · 0.3 cps

denied clicks: 0 (0.0%) · 0.0 cps

Top Sources

Install Sphere Rev S...	2,934 clks · 0.3 qps ↑
Lavasoft	2,417 clks · 0.2 qps ↑
Panda Security	1,390 clks · 0.1 qps ↓
CNET	1,275 clks · 0.1 qps ↑
SocialAds - 00	1,155 clks · 0.1 qps ↓
SocialAds - 05	990 clks · 0.1 qps ↓
Orbit Downloader	635 clks · 0.1 qps
AnchorFree	539 clks · 0.1 qps
Install Sphere	517 clks · 0.0 qps
WinPortal	495 clks · 0.0 qps
Softonic	421 clks · 0.0 qps
pmg 3	288 clks · 0.0 qps
FutureAds (ArcadeCan...	270 clks · 0.0 qps ↑
Admarketplace: home ...	215 clks · 0.0 qps
SocialAds - 03	208 clks · 0.0 qps ↑
Install Sphere Monti...	105 clks · 0.0 qps ↓

What we've got

- Each of our daemons has its own complete set of perl libraries, independently updatable
- Bittorrent-like thingie that manages millions of objects (files)
  - one file changed in 1 snapshot deployed in a few seconds
- build <changeset> && badm restart fe
- Works great! A few rollbacks, but most outages affect a subset of users and we just fix the bug

Updating to 5.16

- First step was hard, because we stopped using the system perl & CPAN from rpms
  - Was needed to facilitate changing out Linux
  - Which we did, CentOS 5 => 6
- We have a lot of XS. Numerous minor changes. Some Cargo Cult in our XS.

- 5.18: a few days work. Bug #118159

  $a = dualvar 1, “”; print $a ? “true” : “false”

  Change was reverted! Our first perl bug ever!

Advice for updating

- Watch out for dual-life modules: upgrades turn into downgrades when you rev Perl
- Yes, updating 400 cpan modules will introduce bugs
  - BSD::Resource::getrlimit stopped returning a hashref in scalar context between 1.28 and 1.2904
- Rerun all your XS memory leak tests
- Read the release notes
  - sv_upgrade => SvUPGRADE
- All your hacks will haunt you
  - failed requires stay in %INC in 5.16, wuh?
- Send out advice to your colleagues
  - "don't use given/when & smartmatch"

Blaming the Bystander

- We're still on IO::AIO 3.3, tip is 4.18
- All of our disk I/O goes through it, even logs
- Significant bug: double-close
  - if you reuse fds fast enough, it closes something
  - our daemons would run for 2 weeks and then close their TCP accept socket, or their UDP socket
- Never reported to author ☹
- Upgraded to CentOS 6: OMG !!! load doubled, disks are 100% busy !!! Our old IO::AIO is incompatible with new kenel ?!?!
- Solved a week later: IO::AIO a bystander

Challenges with perl

- Memory leaks
  - Surprised that a tool to monitor object leaks is not common
  - Built one; turns out almost all of our leaks were actually busted XS
- Crashes in 5.8.8
  - Built an industrial strength stack dumper: C then ugly Perl then pretty Perl
  - Woah. 5.16 got rid of all of our mystery crashes

The Dien Bien Phu of logfile analysis

- perl spews shit to stderr all the time
- Hard to split our warns from perl complaints formatted like warns ("at line N, file foo")
- In order to summarize all those damn warns, I thought I wanted to know the difference
- I wrote a routine with a 720 line regex of messages culled from perldiag
- Also needed to make a list of non-warn-like things perl printed: /^panic:/
- And glibc: /^\Q*** stack smashing detected/

Please

- Someone (with more brain than I) please invent a better methodology for classifying lines in logfiles from long running daemons
- I’ll toss in the “get some backtrace info even if the stack is corrupt” stuff

The body count

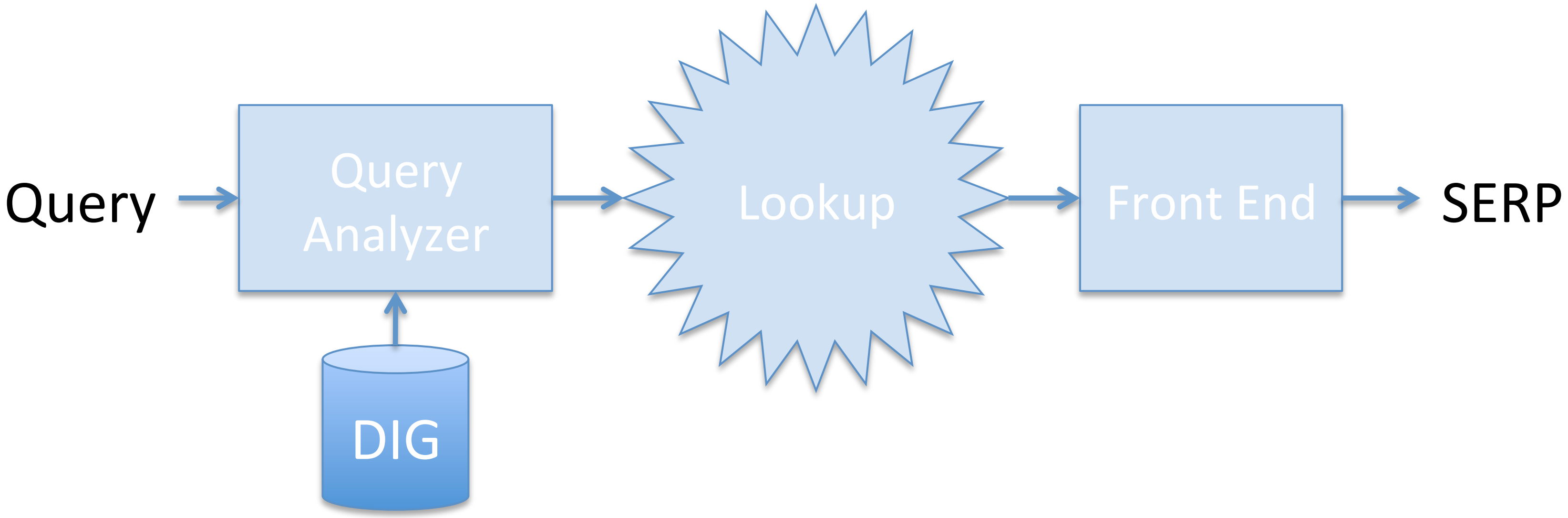
- total perl: 3,500 files, 1.07 million lines
  - NoSQL: 135,000 lines
  - Search engine: 872,000 lines
- total XS: 152 files, 82,000 lines
- total C/C++: 380 files, 326,000 lines

- $53 million, nearly 6 years of our lives, 5 babies, 5 million daily searches, a few gray hairs, our CEO has a tonsure now

Open Data and Open Source

- ODP got a lot of interest in the open source community, & inspired Wikipedia
  - (that was us, yo)
- We’ve gotten little attention for giving away both our curation data (github.com/blekko) and our crawl ranking data (via the Common Crawl Foundation)
- We’re open to Open Sourcing a bunch of our code, but our resources to do it are zilch.

p.s. this is the secret sauce



- 2,000 vertical categories built by our librarians
- Non-keyword-based classification using the web as a dataset
- Runs in ~0.01 seconds
- Many accurate categories for ambiguous queries
- Additional word-based "afterburner" to split big categories like "programming"

Thank you! For more info

- http://bit.ly/yapcna_blekko_2013

- These slides
- High Scalability blog series
- Videos of conference talks
- A few blekko blog postings