Cache on delivery

marco@sensepost.com

whoami

Scalable applications / Cloud?

Essential characteristics

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

Service models

- Cloud Software as a Service (SaaS)
- Cloud Platform as a Service (PaaS)
- Cloud Infrastructure as a Service (IaaS)

Deployment models

- Private cloud
- Community cloud
- Public cloud
- Hybrid cloud

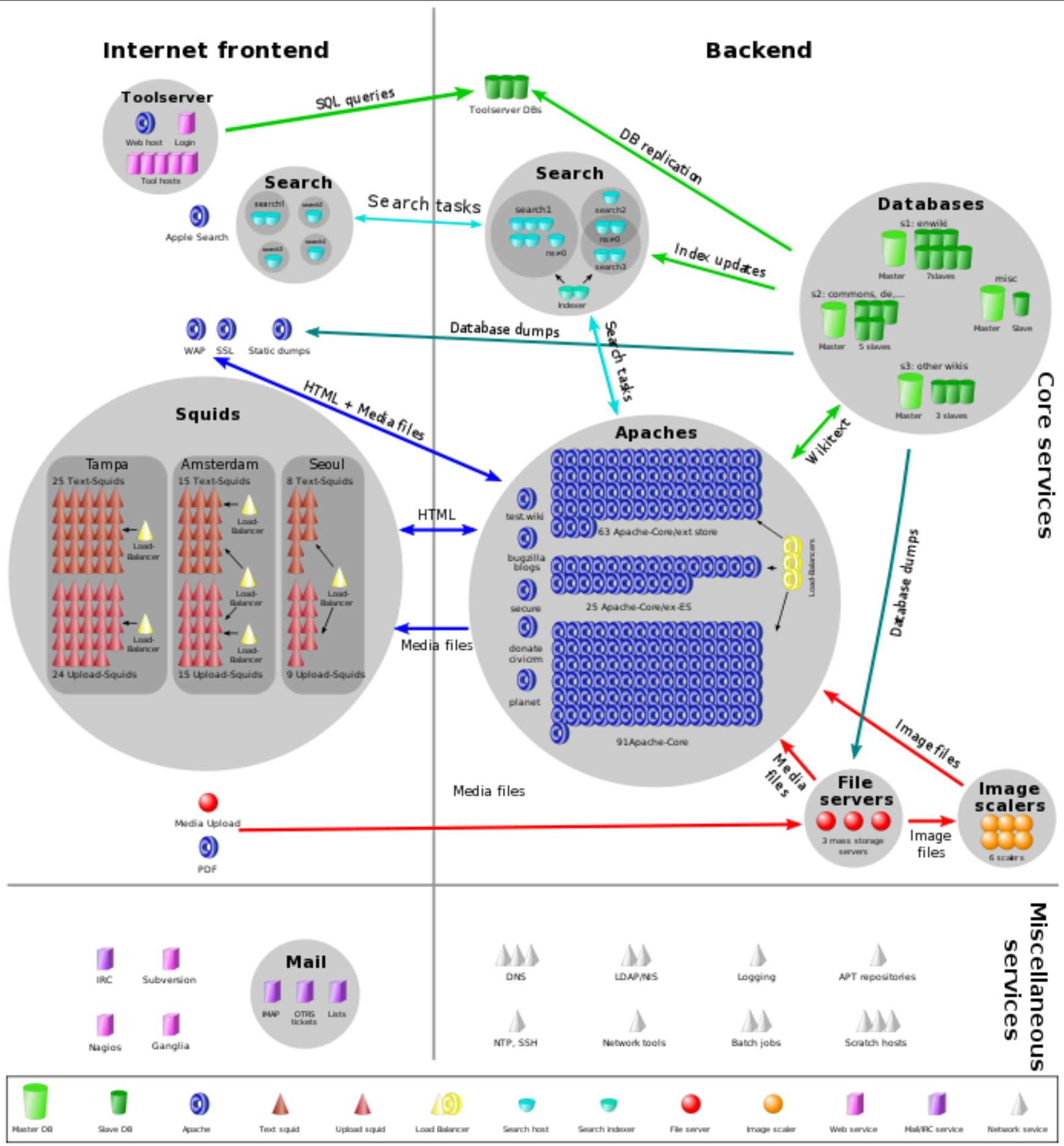
http://csrc.nist.gov/groups/SNS/cloud-computing/

Cloud options

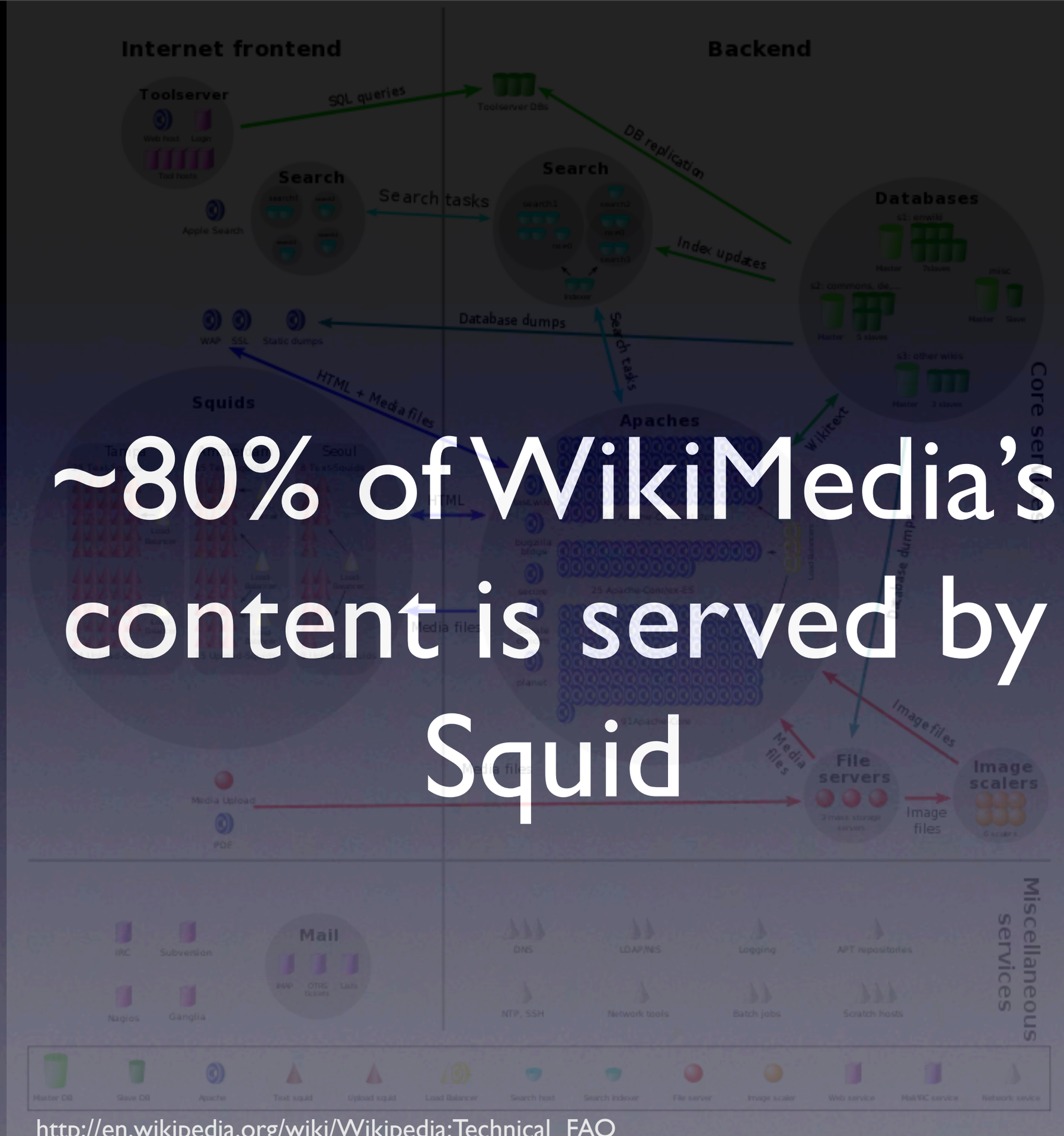
http://www.flickr.com/photos/eli_k_hayasaka/764416130/

The need for caching

- Large percentage of data remains relatively constant
  - Wikipedia page contents
  - Youtube video links
  - FB Profile data
- Poorly designed solutions regenerate data on each request
- Don't regenerate, rather regurgitate
- Caching!=buffering



http://upload.wikimedia.org/wikipedia/commons/4/4f/Wikimedia-servers-2009-04-05.svg



~80% of WikiMedia's content is served by Squid

http://en.wikipedia.org/wiki/Wikipedia:Technical_FAQ

Caching solutions

At all layers, there are caches

Hard disk cache	< 64MB
CPU Cache	< 32MB
Caching proxies	GBs-TBs
Cached scripts/pages	MBs-GBs
Cached database queries / computations	MBs-GBs
Browser caches	MBs-GBs

Caching solutions

Let's focus on the application layer (too many options)

Redis	Persistent KV Store
Ehcache	Persistent Store
Memcache	KV Store
MemcacheDB	Persistent KV Store
Websphere eXtreme Scale	Obj Store
Oracle Coherence	Obj Store
Google BigTable	Persistent Store

Caching solutions

Let's focu
application layer (too many
options)

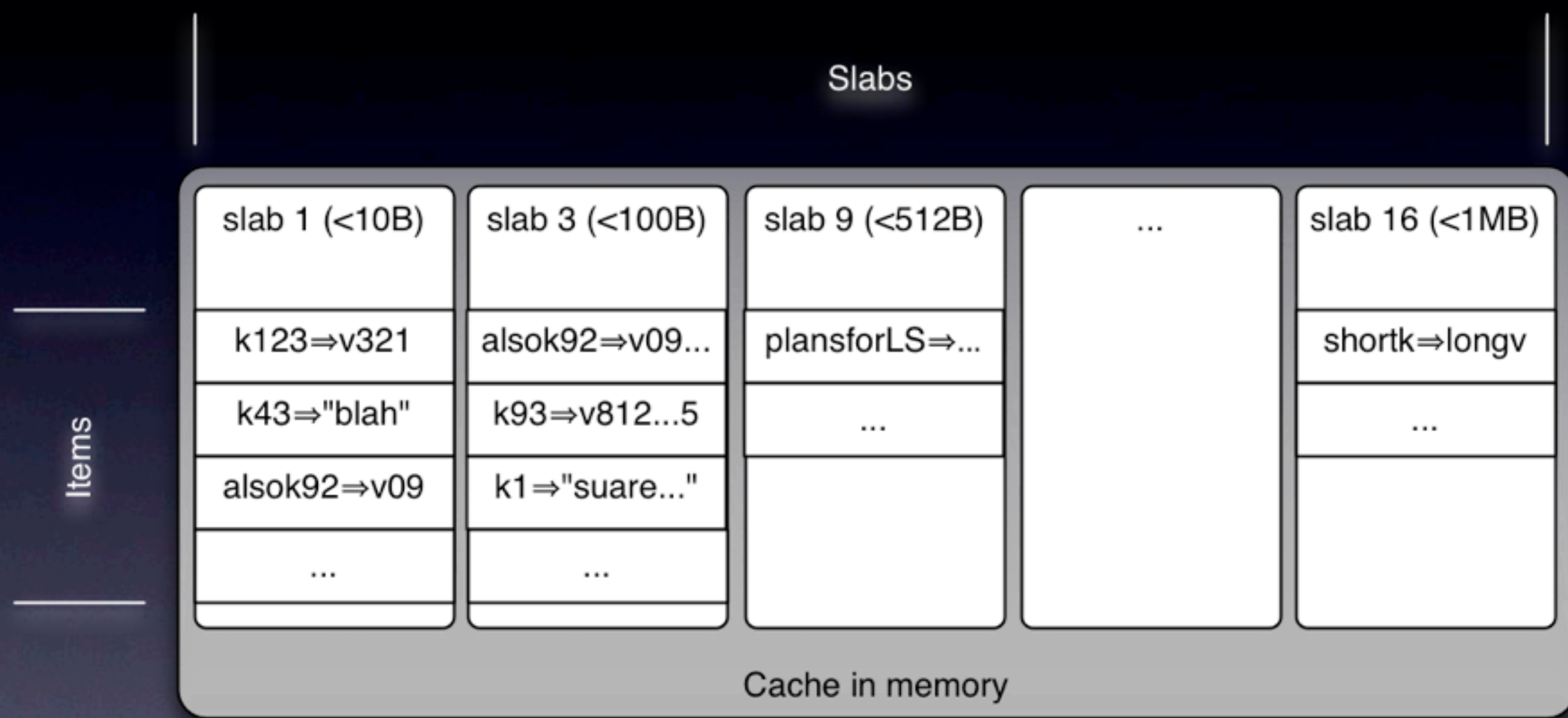
Redis	Persistent KV Store
Ehcache	Persistent Store
Memcache	KV Store
MemcacheDB	ersistent KV Store
Websphere eXtreme Scale	Obj Store
Oracle Coherence	Obj Store
Google BigTable	Persistent Store

Memcache

- memcache.org
- Written for early LJ
- Non-persistent network-based KV store
- [setup+usage demo]

LiveJournal
Wikipedia
Flickr
Bebo
Twitter
Typepad
Yellowbot
Youtube
Digg
Wordpress

Basic KV



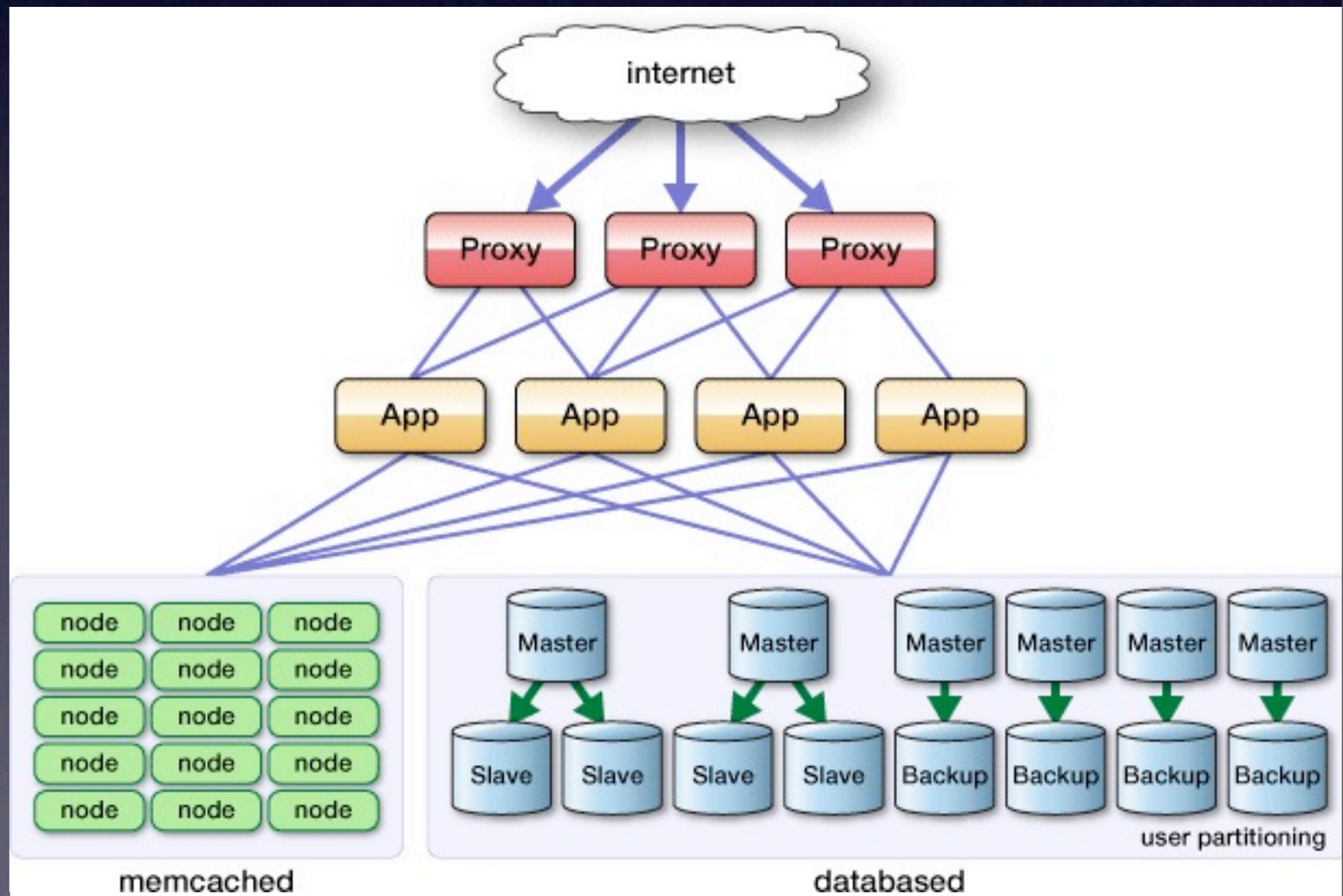
- Slabs are fixed size
- Dst slab determined by *value* size
- Users don't care about slabs
- Miners care about slabs

Application Integration

```
function get_foo(foo_id)
    foo = memcached_get("foo:" . foo_id)
    return foo if defined foo

    foo = fetch_foo_from_database(foo_id)
    memcached_set("foo:" . foo_id, foo)
    return foo
end
```

http://memcached.org/



http://pic001.cnblogs.com/img/dudu/200809/2008092817263955.png

Trivial protocol

- ASCII-based
- Long-lived
- Tiny command set

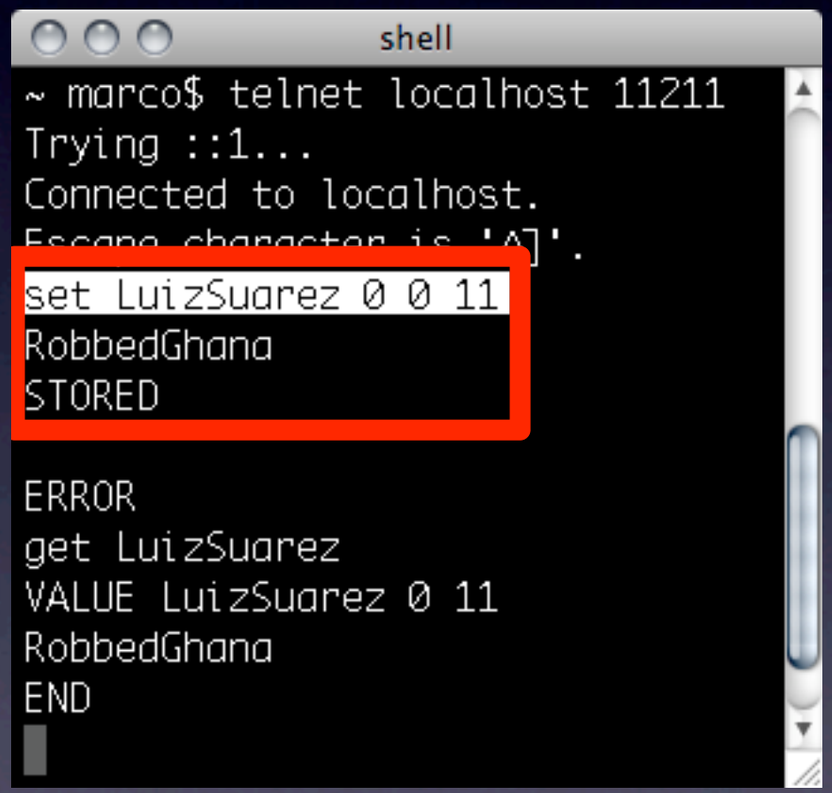
- ????
- set
- get
- stats
- ...

Binary and UDP protocols also exist, these were not touched.

Trivial protocol

- ASCII-based
- Long-lived
- Tiny command set

- ????
- set
- get
- stats
- ...

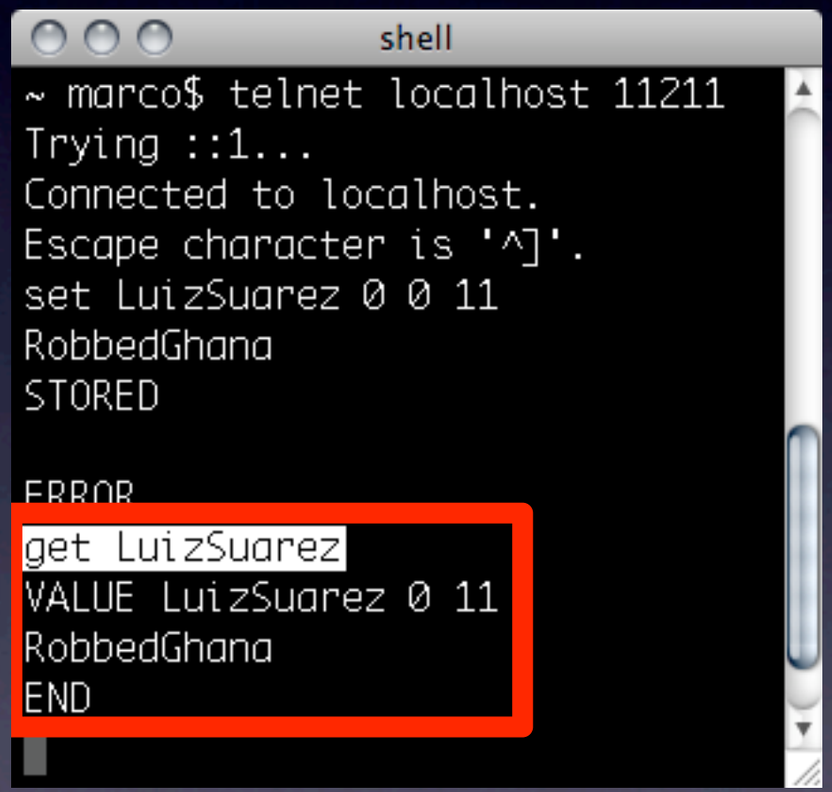


Binary and UDP protocols also exist, these were not touched.

Trivial protocol

- ASCII-based
- Long-lived
- Tiny command set

- ????
- set
- get
- stats
- ...



Binary and UDP protocols also exist, these were not touched.

Trivial protocol

- ASCII-based
- Long-lived
- Tiny command set

- ????
- set
- get
- stats
- ...

```
stats
STAT pid 99519
STAT uptime 496945
STAT time 1279563052
STAT version 1.4.5
STAT pointer_size 32
STAT rusage_user 9.454329
STAT rusage_system 22.089662
STAT curr_connections 10
STAT total_connections 215
STAT connection_structures 12
STAT cmd_get 442
STAT cmd_set 102779
STAT cmd_flush 0
```

Binary and UDP protocols also exist, these were not touched.

Trivial protocol

- ASCII-based
- Long-lived
- Tiny command set

- ????
- set
- get
- stats
- ...

Binary and UDP protocols also exist, these were not touched.

Goals

- Connect to memcached
- Find all slabs
- Retrieve keynames from each slab
- Retrieve each key
-

Lies, damn lies, and stats

- stats cmd has subcmds
  - items
  - slabs
  - ...

```
stats slabs
STAT 1:chunk_size 80
<...>
STAT 2:chunk_size 104
<...>
STAT 3:chunk_size 136
<...>
STAT 4:chunk_size 176
<...>
STAT 6:chunk_size 280
<...>
STAT 8:chunk_size 440
<...>
STAT 9:chunk_size 552
<...>
STAT 9:cas_badval 0
STAT active_slabs 7
```

This gets us the slabs_ids

Retrieving key names

Rely on two
{poorly|
un}documented
features

Retrieving key names

Feature #1:

Remote enabling of debug mode

```
shell
memcached-1.4.5 marco$ ./memcached -h
memcached 1.4.5
-p <num>      TCP port number to listen on (default: 11211)
-U <num>      UDP port number to listen on (default: 11211, 0 is off)
-s <file>     UNIX socket path to listen on (disables network support)
              the memory page size could reduce the number of TLB misses
              and improve the performance. In order to get large pages
              from the OS, memcached will allocate the total item-cache
              in one large chunk.
-D <char>     Use <char> as the delimiter between key prefixes and IDs.
              This is used for per-prefix stats reporting. The default is
              ":" (colon). If this option is specified, stats collection
              is turned on                                , then it may be turned on
              by sending the "stats detail on"  ommand to the server.
-t <num>      number of thr                                4)
-R            Maximum number of requests per event, limits the number of
              requests process for a given connection to prevent
              starvation (default: 20)
```

Retrieving key names

Feature #2:

“stats cachedump”

Retrieving key names

Feature #2:

“stats cachedump”

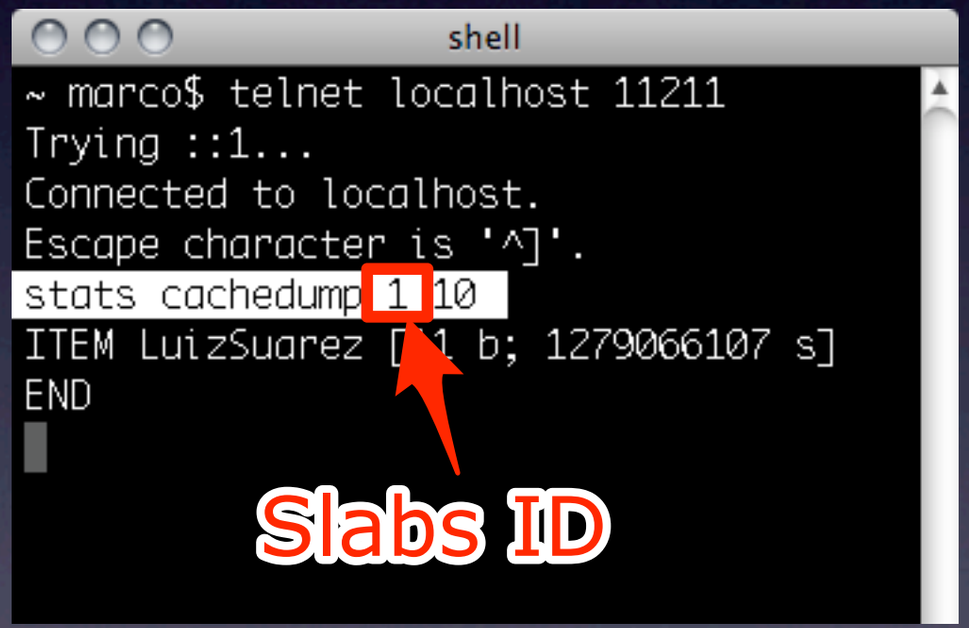
shell

```
~ marco$ telnet localhost 11211
Trying ::1...
Connected to localhost.
Escape character is '^]'.
stats cachedump 1 10
ITEM LuizSuarez [11 b; 1279066107 s]
END
```

Retrieving key names

Feature #2:

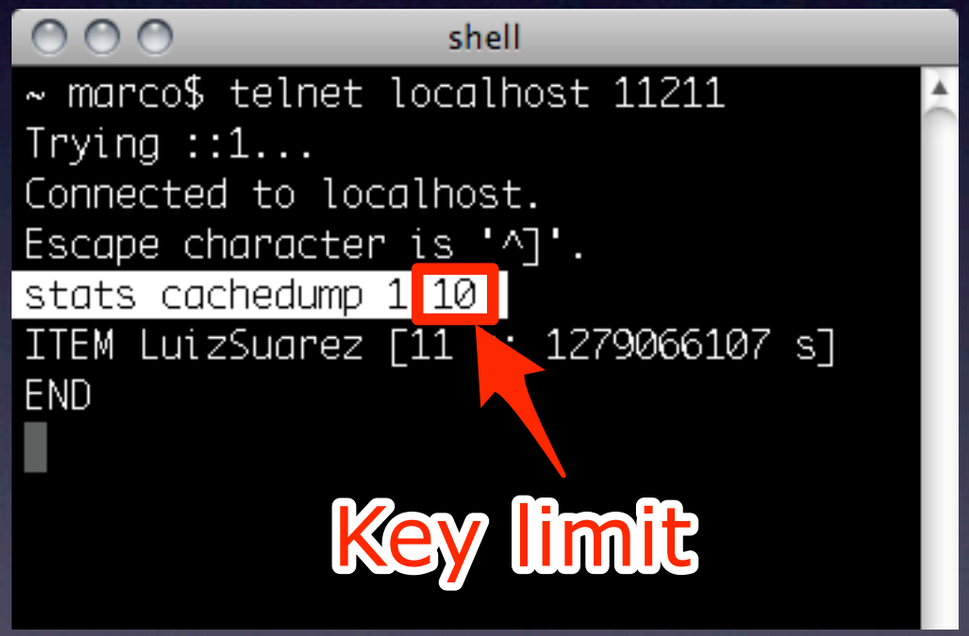
“stats cachedump”



Retrieving key names

Feature #2:

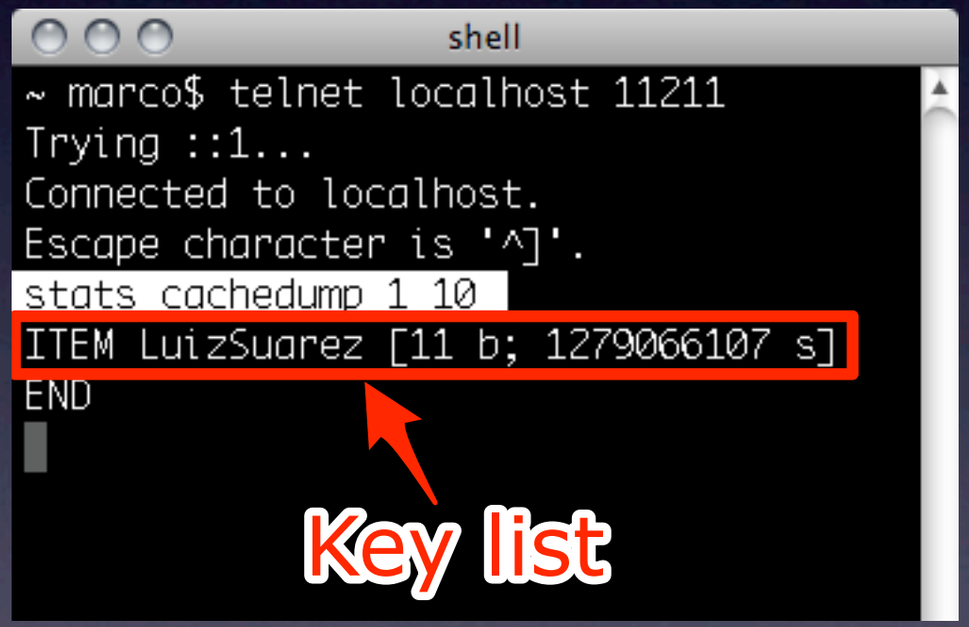
“stats cachedump”



Retrieving key names

Feature #2:

“stats cachedump”



Retrieving key names

Feature #2:

“stats cachedump”

This gets us key names

And this gets us?

- No need for complex hacks. Memcached serves up all its data for us.
- What to do in an exposed cache?
  - Mine
  - Overwrite

Mining the cache

- go-derper.rb – memcached miner
  - Retrieves up to *k* keys from each slab and their contents, store on disk
  - Applies regexes and filters matches in a *hits* file
  - Supports easy overwriting of cache entries
- [demo]

Finding memcaches

- Again with the simple approach
  - Pick an EC2 subnet, scan for memcaches
    Port 11211 and mod'ed .nse
- Who's %#^&ing cache is this?
- Where's the good stuff?
- Is it live?

Results

- Objects found
  - Serialized Java
  - Pickled Python
  - Ruby ActiveRecord
  - .Net Object
  - JSON

Results: Actual Sites

- [screenshots in the talk]

Fixes?

- FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW.
- Hack code to disable stats facility (but doesn’t prevent key brute-force)
- Hack code to disable remote enabling of debug features
- Switch to SASL
  - Requires binary protocol
  - Not supported by a number of memcached libs
- Also, FW.

Places to keep looking

- Improve data detection/sifting/filtering
- Spread the search past a single EC2 subnet
- Caching providers (?!?!)
- Other cache software

Questions?

sensepost.com/blog