# facebook

facebook

# Erlang at Facebook

Eugene Letuchy

Apr 30, 2009

# Agenda

1. Facebook ... and Erlang
2. Story of Facebook Chat
3. Facebook Chat Architecture
4. Key Erlang Features
5. Then and Now

# Facebook … and Erlang

# The Facebook Environment

- The Site
  - More than 200 million active users
  - More than 3.5 billion minutes are spent on Facebook each day
  - Fewer than 900 employees
- The Engineering Team
  - Fast iteration: code gets out to production within a week
  - Polyglot programming: interoperability is key
  - Practical: high-leverage tools win

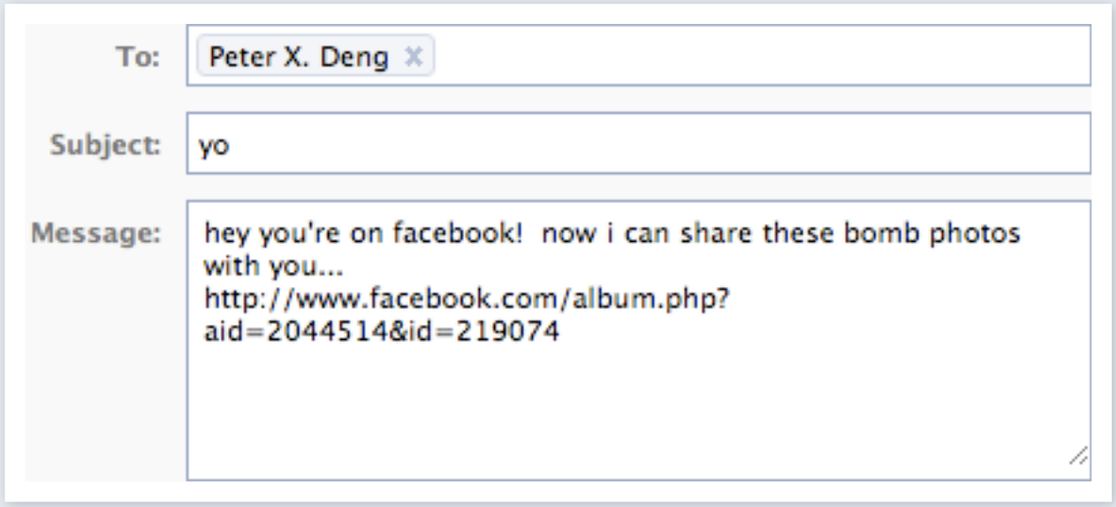
# Erlang Projects

- Chat: the biggest and best known user
- AIM Presence: a JSONP validator
- Chat Jabber support (ejabberd)

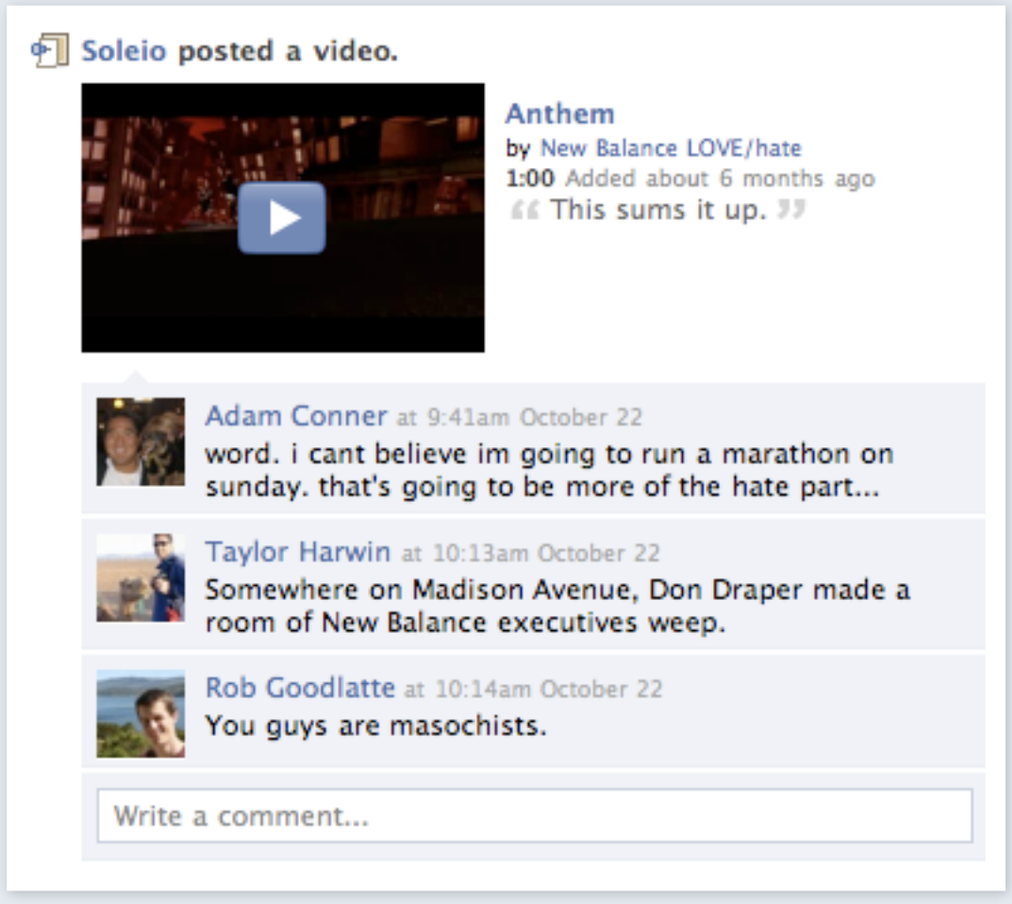
# Facebook Chat

# 2007: Facebook needs Chat

## Messages, Wall, Links aren't enough







# Enter a Hackathon (Jan 2007)

- Chat started in one night of coding
  - Floating conversation windows
  - No buddy list
  - One server (no distribution)
  - Erlang was there!

# Enter Eugene (Feb 2007)

- I joined Facebook after Chat Hackathon
- What is this Erlang?

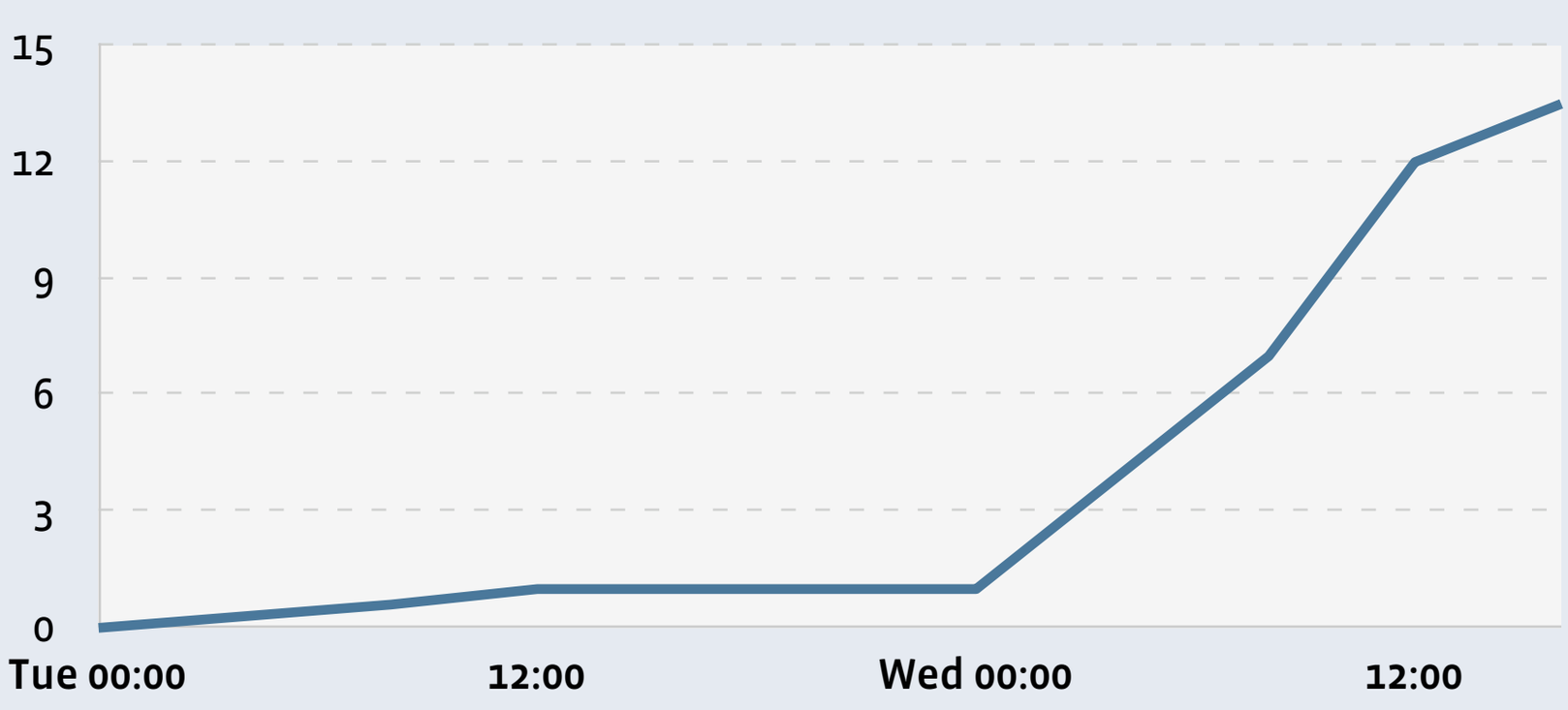
- Spring 2007:
  - Learning Erlang from Joe Armstrong's thesis
  - Lots of prototyping
  - Evaluating infrastructure needs
- Summer 2007:
  - Chris Piro works on Erlang Thrift bindings

# Let's do this!

- Mid-Fall 2007: Chat becomes a "real" project
  - 4 engineers, 0.5 designer
- Infrastructure components get built and improved
- Feb 2008: "Dark launch" testing begins
  - Simulates load on the Erlang servers ... they hold up
- Apr 6, 2008: First real Chat message sent
- Apr 23, 2008: 100% rollout (Facebook has 70M users at the time)

# Launch: April 2008

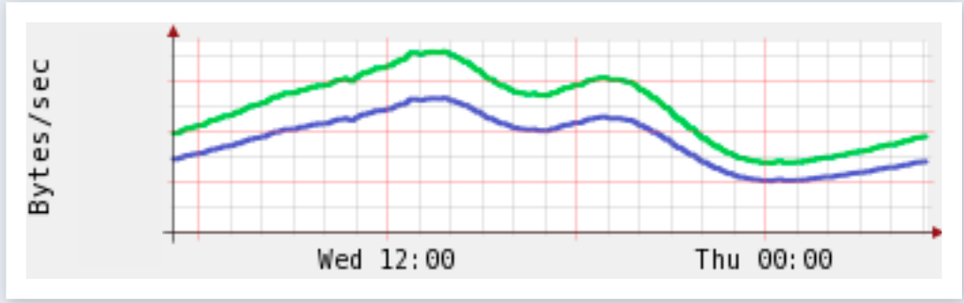
- Apr 6, 2008: gradual live rollout starts
  - First message: "msn chat?"
- Apr 23, 2008: 100% rollout (to Facebook's 70M users)
- Graph of sends in the first days of launch

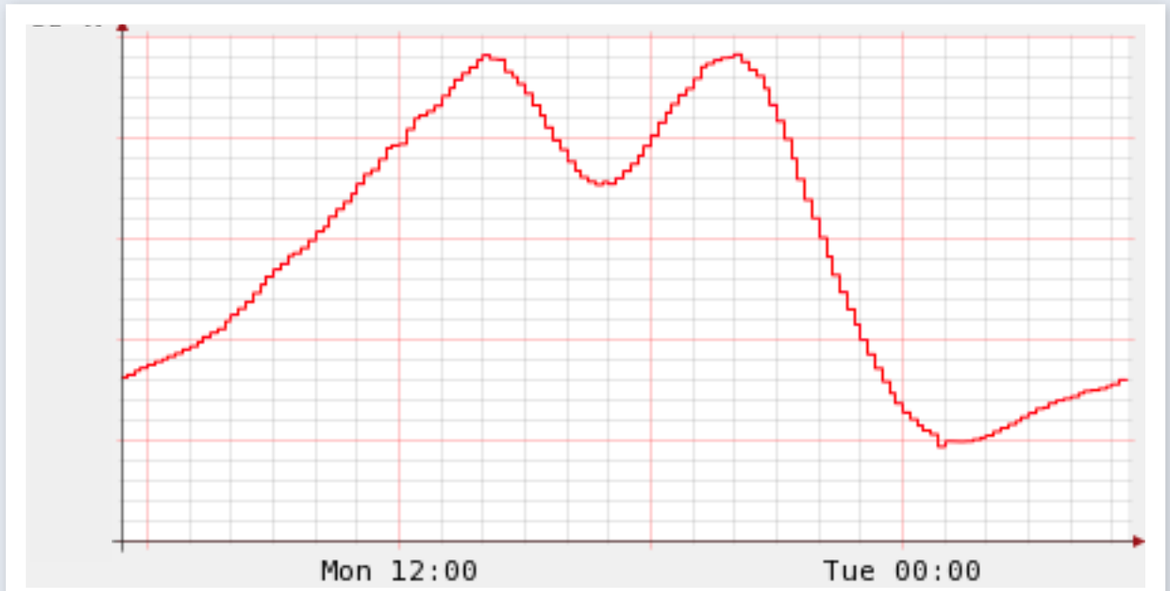


# Chat ... one year later

- Facebook has 200M active users
- 800+ million user messages / day
- 7+ million active channels at peak
- 1GB+ in / sec at peak
- 100+ channel machines
- ~9-10 times the work at launch;

  ~2 as many machines



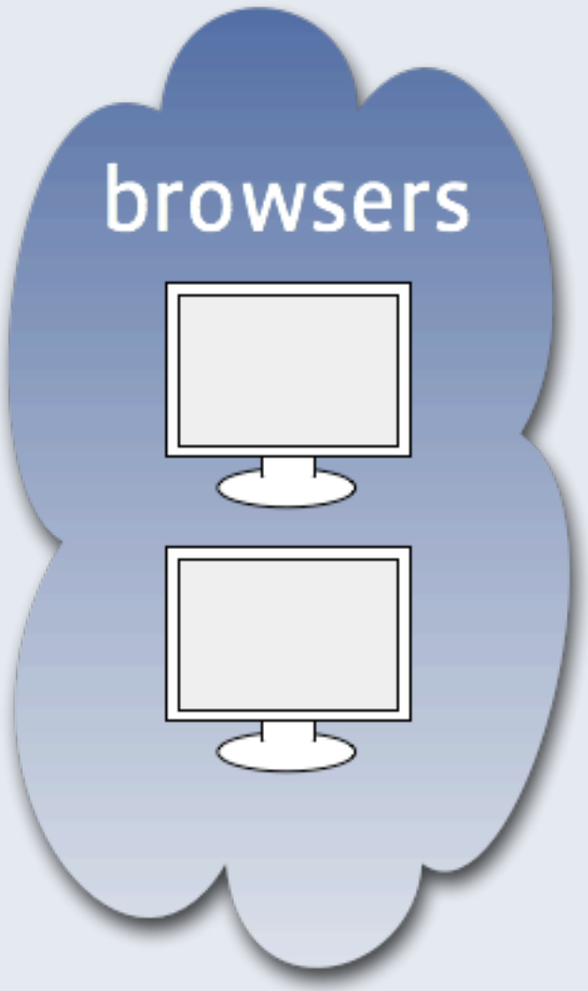



# Chat Architecture

# System challenges

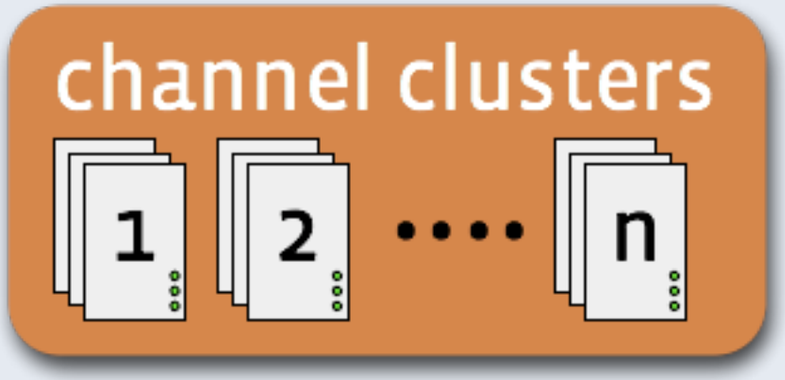
- How does synchronous messaging work on the Web?
- “Presence” is hard to scale
- Need a system to queue and deliver messages
  - Millions of connections, mostly idle
- Need logging, at least between page loads
- Make it work in Facebook’s environment

# System overview









# System overview - User Interface

## Chat in the browser?

- Chat bar affixed to the bottom of each Facebook page

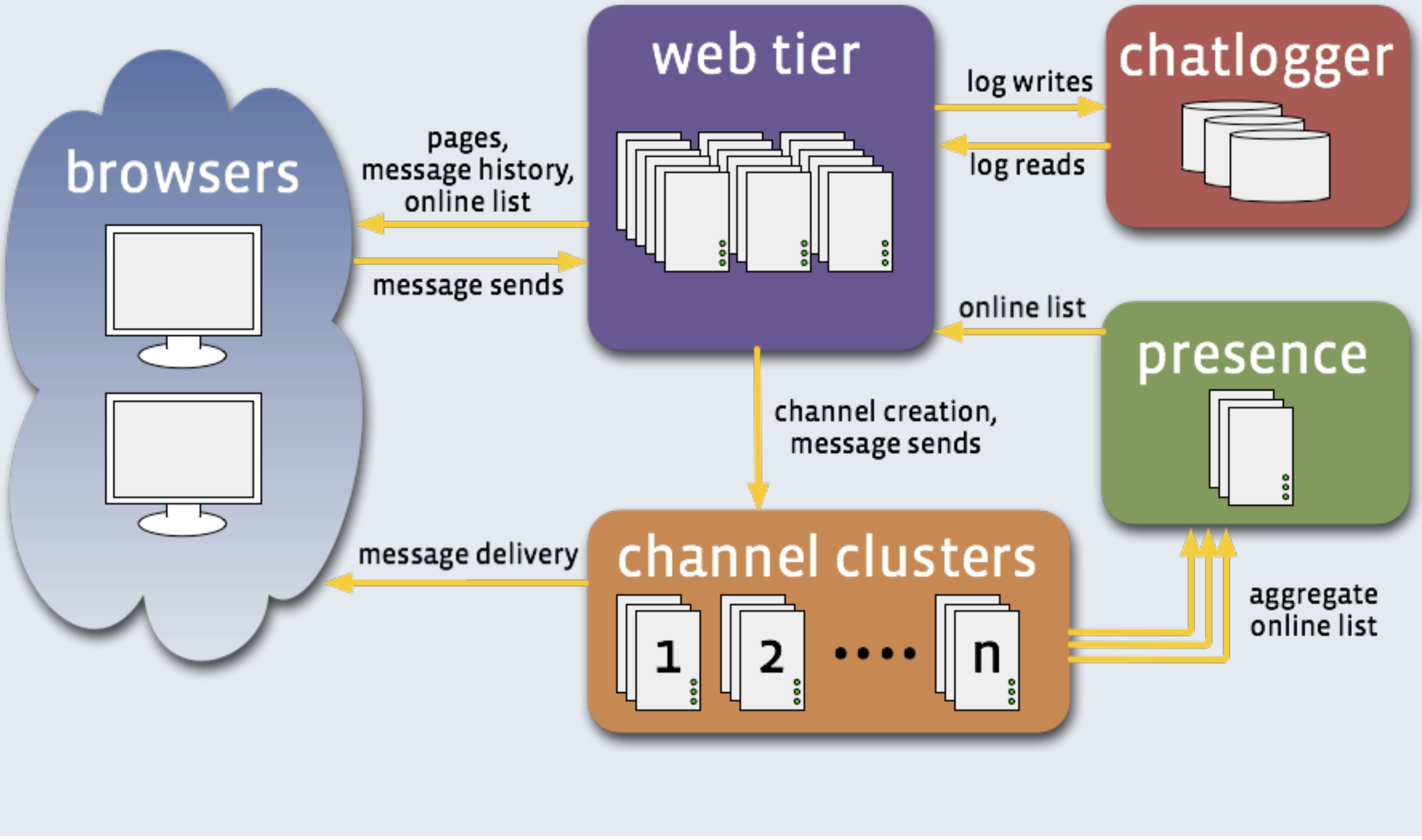

- Mix of client-side Javascript and server-side PHP
- Works around transport errors, browser differences
- Regular AJAX for sending messages, fetching conversation history
- Periodic AJAX polling for list of online friends
- AJAX long-polling for messages (Comet)

# System Overview - Back End

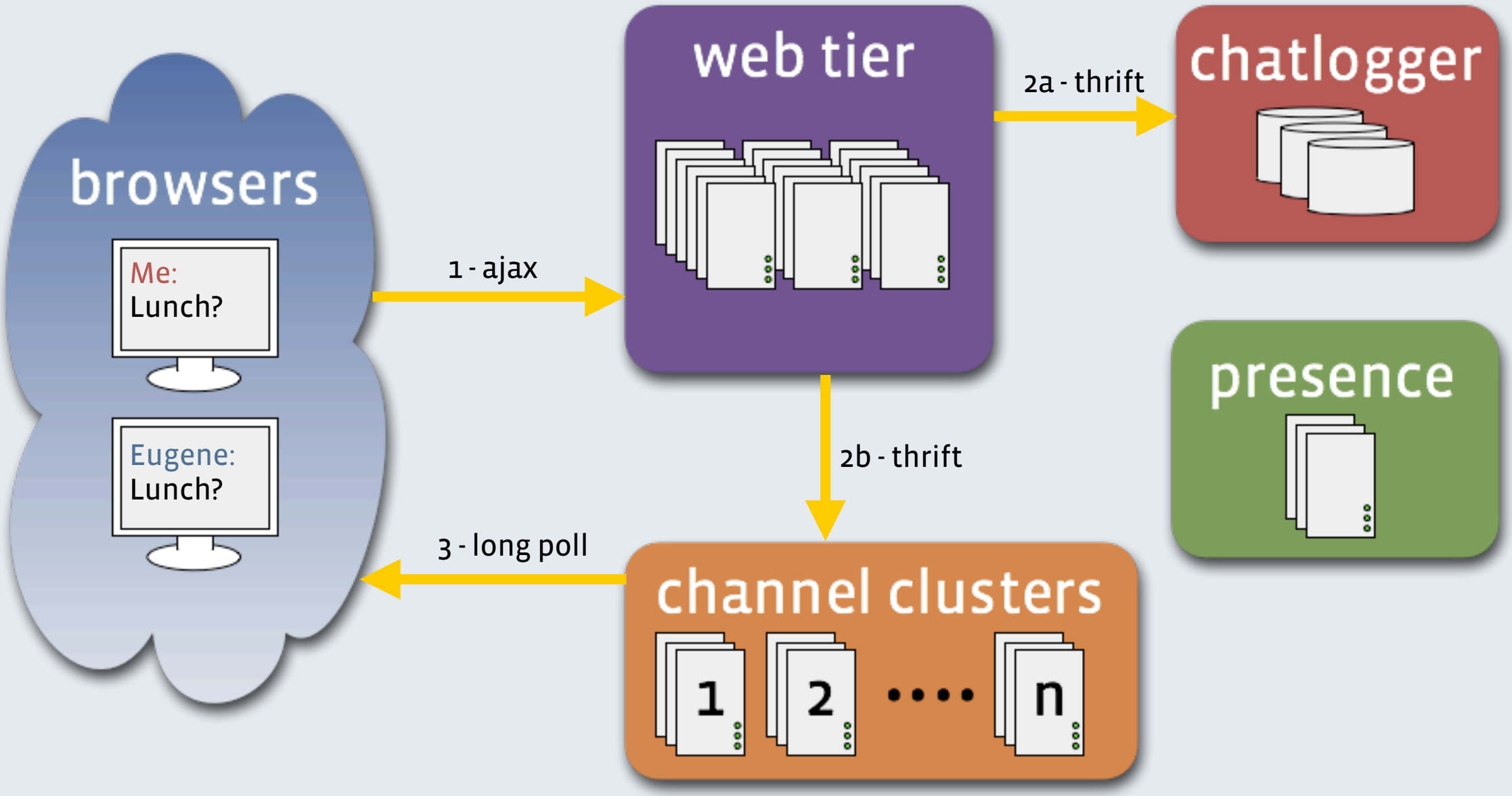
## How does the back end service requests?

- Discrete responsibilities for each service
  - Communicate via Thrift
- Channel (Erlang): message queuing and delivery
  - Queue messages in each user's "channel"
  - Deliver messages as responses to long-polling HTTP requests
- Presence (C++): aggregates online info in memory (pull-based presence)
- Chatlogger (C++): stores conversations between page loads
- Web tier (PHP):  serves our vanilla web requests

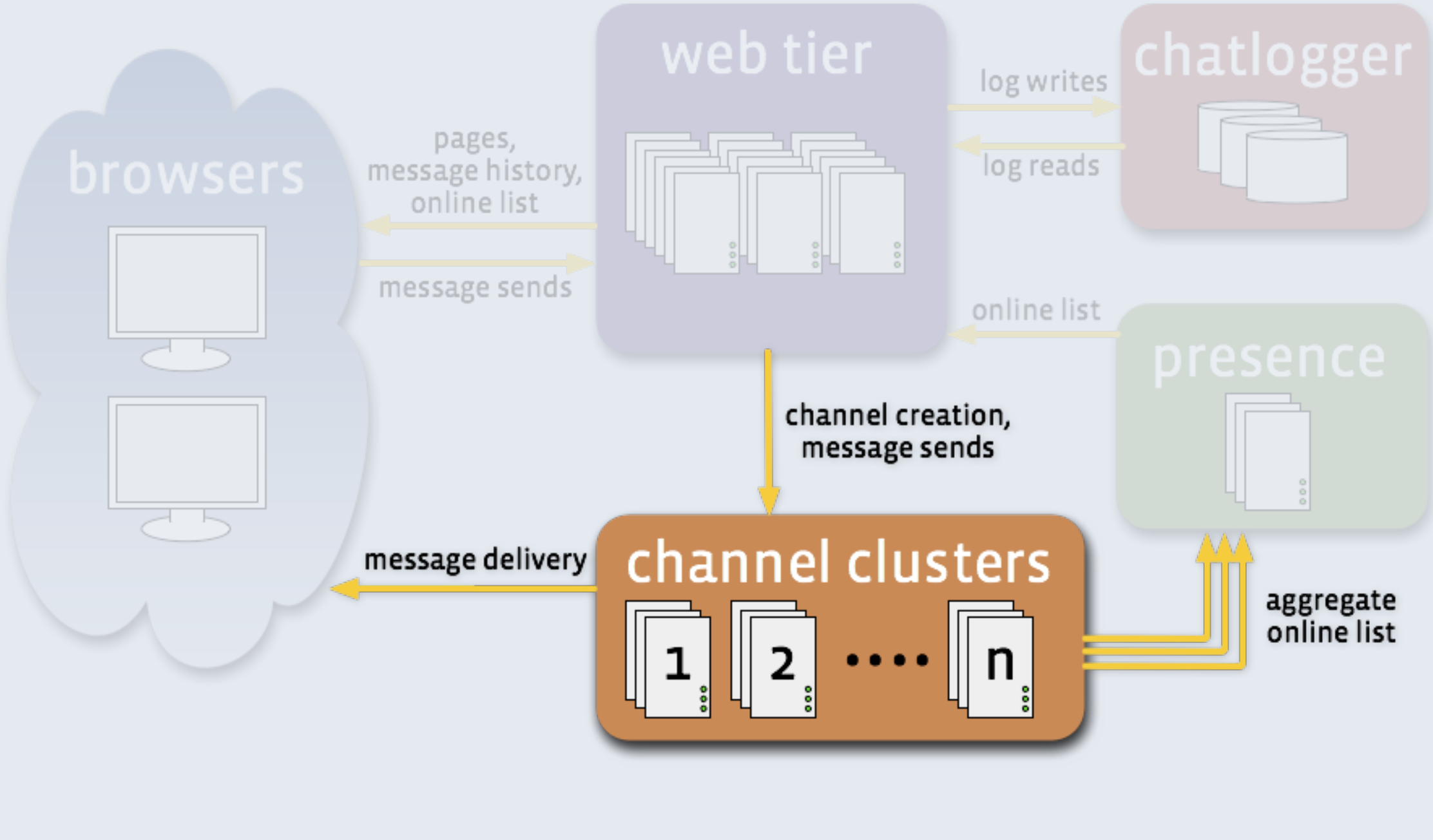
# System overview



# Message send



# Channel servers (Erlang)

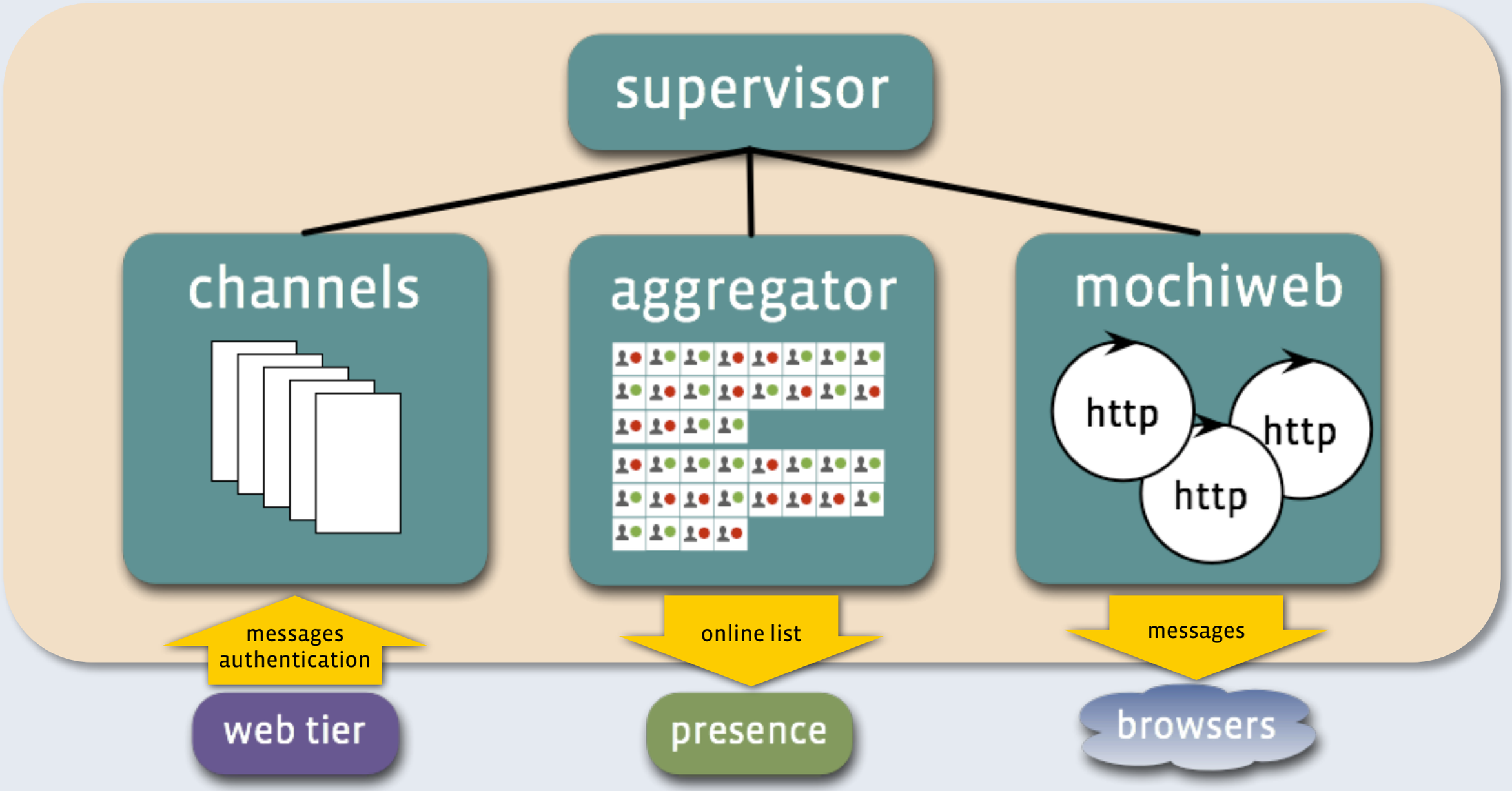


# Channel servers

## Architectural overview

- One channel per user
- Web tier delivers messages for that user
- Channel State: short queue of sequenced messages
- Long poll for streaming (Comet)
  - Clients make an HTTP request
  - Server replies when a message is ready
  - One active request per browser tab

# channel application



# Channel servers

## Architectural details

- Distributed design
  - User id space is partitioned (division of labor)
  - Each partition is serviced by a cluster (availability)
- Presence aggregation
  - Channel servers are authoritative
  - Periodically shipped to presence servers
- Open source:  Erlang, Mochiweb, Thrift, Scribe, fb303, et al.

# Key Erlang Features we love

# Concurrency

- Cheap parallelism at massive scale
- Simplifies modeling concurrent interactions
  - Chat users are independent and concurrent
  - Mapping onto traditional OS threads is unnatural
- Locality of reference

- Bonus: carries over to non-Erlang concurrent programming

# Distribution

- Connected network of nodes
- Remote processes look like local processes
  - Any node in a channel server cluster can route requests
  - Naive load balancing
- Distributed Erlang works out-of-the-box (all nodes are trusted)

# Fault Isolation

- Bugs in the initial versions of Chat:
  - Process leaks in the Thrift bindings
  - Unintended multicasting of messages
  - Bad return state for presence aggregators
- (Horrible) bugs don't kill a mostly functional system:
  - C/C++ segfault takes down the OS process and your server state
  - Erlang badmatch takes down an Erlang process
    - … and notifies linked processes

# Error logging (Crash Reports)

- Any proc_lib-compliant process generates crash reports
- Error reports can be handled out of band (not where generated)
- Stacktraces point the way to bugs (functional languages win big here)
  - … but they could be improved with source line numbers
- Writing error_log handlers is simple:
  - gen_event behavior
  - Allows for massaging of the crash and error messages (binaries!)
  - Thrift client in the error log
- WARNING: error logging can OOM the Erlang node

# Hot code swapping

- Restart-free upgrades are awesome (!)
  - Pushing new functional code for Chat takes ~20 seconds
  - No state is lost
- Test on a running system
- Provides a safety net … rolling back bad code is easy

- NOTE: we don't use the OTP release/upgrade strategies

# Monitoring and Error Recovery

- Supervision hierarchies
  - Organize (and control) processes
  - Organize thoughts
  - Systematize restarts and error recovery
  - simple_one_for_one for dynamic child processes
- net_kernel (Distributed Erlang)
  - sends nodedown, nodeup messages
  - any process can subscribe
- heart: monitors and restarts the OS process

# Remote Shell

- To invoke:
  > erl -name hidden -hidden -remsh ‹node_name› -setcookie ‹cookie›
  Eshell V5.7.1 (abort with ^G)
  (‹node_name›)1›

- Ad-hoc inspection of a running node
- Command-and-control from a console
- Combines with hot code loading

# Erlang top (etop)

- Shows Erlang processes, sorted by reductions, memory and message queue
- OS functionality … for free

```
 Load:  cpu        67              Memory:  total     1276592    binary       8966
        procs   21774                       processes  709872    code         4738
        runq        0                       atom          501    ets          3583

Pid            Name or Initial Func    Time    Reds  Memory    MsgQ Current Function
------------------------------------------------------------------------------------
<4803.74.0>    ch_channel_sup           '-'********12798552       0 gen_server:loop/6
<4803.107.0>   ch_http                  '-'69578012  436504       0 gen_server:loop/6
<4803.6.0>     application_controll     '-'    31905  371960       0 gen_server:loop/6
<4803.47.0>    erlang:apply/2           '-'     4580  263680       0 shell:get_command1/5
<4803.26.0>    code_server              '-'   214718  163256       0 code_server:loop/1
<4803.5.0>     error_logger             '-'11129521  142696       0 gen_event:fetch_msg/
<4803.0.0>     init                     '-'********  142600       0 init:loop/1
<4803.2.0>     erl_prim_loader          '-'   547986  142520       0 erl_prim_loader:loop
**************proc_lib:init_p/5        '-'    16451  110440       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    15275  101160       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    17058   95992       0 erlang:hibernate/3
<4803.3103.369>proc_lib:init_p/5        '-'    14883   91864       0 erlang:hibernate/3
<4803.5808.369>proc_lib:init_p/5        '-'    15160   91032       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    14873   90440       0 erlang:hibernate/3
<4803.3274.369>proc_lib:init_p/5        '-'    13848   89016       0 erlang:hibernate/3
<4803.9448.368>proc_lib:init_p/5        '-'    12719   85688       0 erlang:hibernate/3
<4803.817.368> proc_lib:init_p/5        '-'    12327   85448       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    12245   83352       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    11694   78712       0 erlang:hibernate/3
<4803.4166.368>proc_lib:init_p/5        '-'    12697   76792       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    11531   76216       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    11347   75928       0 erlang:hibernate/3
<4803.8712.368>proc_lib:init_p/5        '-'    13199   75624       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    11201   75272       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    10891   73272       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    11763   72840       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    11711   72712       0 erlang:hibernate/3
**************proc_lib:init_p/5        '-'    10740   72248       0 erlang:hibernate/3
<4803.4577.368>proc_lib:init_p/5        '-'    11679   72136       0 erlang:hibernate/3
<4803.6617.368>proc_lib:init_p/5        '-'    10800   71880       0 erlang:hibernate/3
====================================================================================
```

# Hibernation

- Drastically shrink memory usage with erlang:hibernate/3
  - Throws away the call stack
  - Minimizes the heap
  - Enters a wait state for new messages
  - “Jumps” into a passed-in function for a received message
- Perfect for a long-running, idling HTTP request handler
- But … not compatible with gen_server:call (and gen_server:reply)
  - gen_server:call has its own receive() loop
  - hibernate() doesn’t support have an explicit timeout
  - Fixed with a few hours and a look at gen.erl

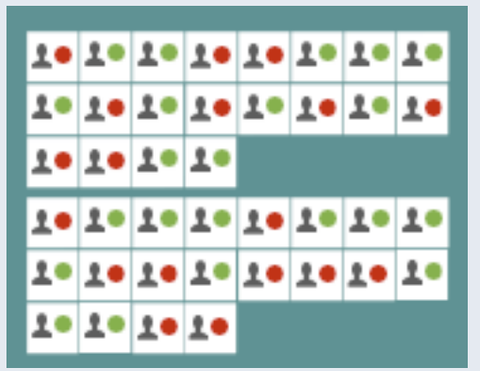
# Symmetric MultiProcessing (SMP)

- Take advantage of multi-core servers
- erl -smp runs multiple scheduler threads inside the node
- SMP is emphasized in recent Erlang development
  - Added to Erlang R11B
  - Erlang R12B-0 through R13B include fixes and perf boosts
  - Smart people have been optimizing our code for a year (!)
  - Upgraded to R13B last night with about 1/3 less load

# hipe_bifs

## Cheating single assignment

- Erlang is opinionated:
  - Destructive assignment is hard because it should be
- hipe_bifs:bytearray_update() allows for destructive array assignment
  - Necessary for aggregating Chat users' presence
  - Don't tell anyone!



# Then and now Erlang in Progress

# Then … a steep learning curve

- Start of 2007:
  - Few industry-focused English-language resources
  - Few blogs (outside of Yariv's and Joel Reymont's)
  - Code examples spread out and disorganized
  - U.S. Erlang community limited in number and visibility

# Now ...

- Programming Erlang (Jun 2007)
- Erlang Programming (upcoming...)
- More blogs and blog aggregators:
  - Planet Erlang, Planet TrapExit
- Erlang Factory aggregates Erlang developments
- More code available:
  - GitHub, CEAN
  - More general-purpose Open Source Libraries
- U.S. -located conference and ErlLounges

# facebook