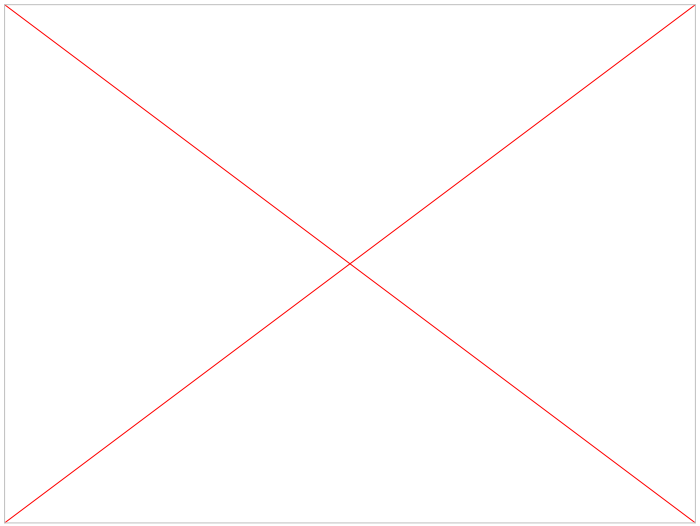
Klarna

Continuous Migration

Reimplementing the
Purchase Taking Capabilities of a
24/7 Financial System

@dklee #EUC2013
Klarna Engineering

About @dklee



Past Life: type-theory researcher

Core Code Grunt at Klarna since 2011

Currently Real-Time Core: Platform

Hobby: cooking big pieces of meat

Klarna

Online Payments provider since 2004

Goal: increase merchant conversions through simpler paying experience

Invoicing: Klarna takes credit risk

Services merchants in Sweden, Norway, Finland, Denmark, Germany, the Netherlands, and Austria

Klarna Engineering

Almost 200 engineers across Stockholm, Uppsala, and Tel Aviv

Stockholm Office:
one of the world's biggest Erlang shops

Pre: 2010 ~10 Erlang developers

Today: ~70 Erlang developers

SFSA Compliance

As an (almost) bank, Klarna is subject to financial regulations from **Swedish Financial Supervisory Authority**

All code changes must be documented in ticket system and reviewed

"Implicit" code changes highly undesirable, i.e. illegal

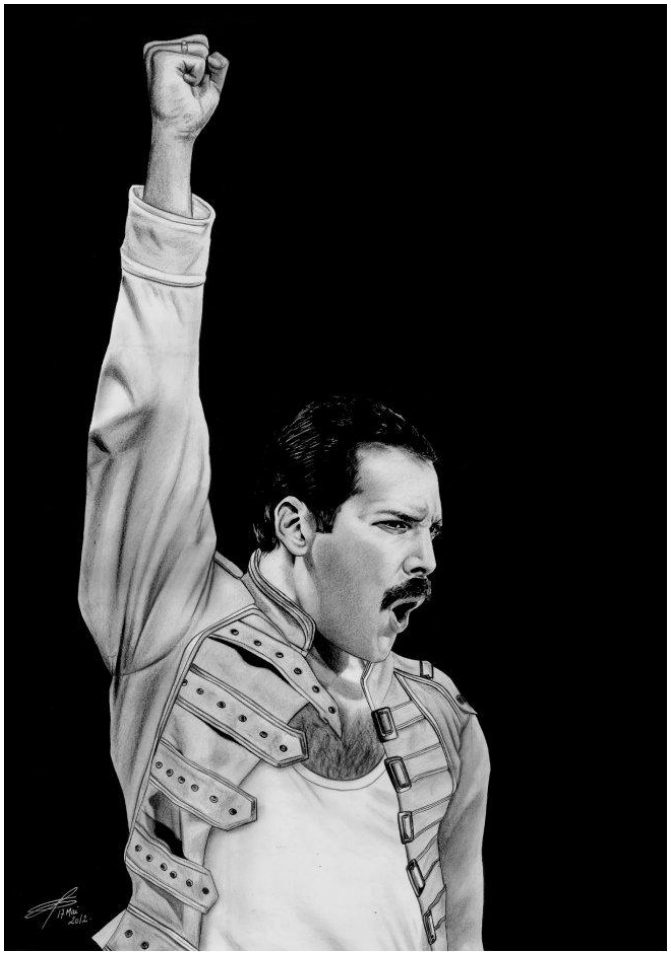
kred -- legacy system

Monolithic system serving almost all business functions (purchase taking, payments, collections, etc)

2011: all business-logic version controlled in single git repo

Multiple on-going projects to split out functionality into new services

FRED -- shiny new system



FRont-end krED

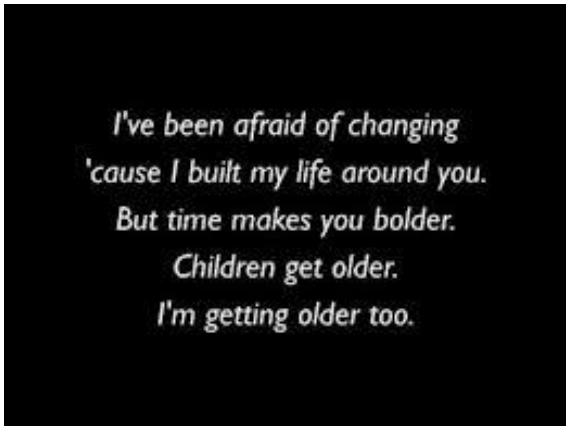
Termination point for estore APIs

Legacy XML-RPC API

Klarna Checkout

See Mats' Cronqvist's talk for architectural details

Legacy Business Logic



- Klarna sells a service that merchants integrate against
- Many special cases in code to please customers and make sales
- Legacy business logic is rather profitable in the short-term
- Removing old cruft is complicated and expensive

Linus Torvalds on Refactoring Legacy Code

"Why don't we write code that just works? Or absent a 'just works' set of patches, why don't we revert to code that has years of testing? This kind of 'I broke things, so now I will jiggle things randomly until they unbreak' is not acceptable. [...] Don't just make random changes. There really are only two acceptable models of development: 'think and analyze' or 'years and years of testing on thousands of machines'. Those two really do work."
-- Linus Torvalds

Shared Code

Migrations are dangerous

New system requires re-writing dependencies out of legacy code

Maintenance, minor feature additions, and bugfixes must still occur

copy-x-paste: BAD

cut-x-paste: not-so-BAD

Continuous Migration = Iterative Growth of New System

Small incremental changes, shipped regularly

Klarna's legacy system releases weekly

Re-organize legacy spaghetti
-> frameworks
-> shared libraries
-> system specific code
-> stubs

General Frameworks

Hollywood Model, code that calls you

behaviours to specify callback interface

Abstract common control flow patterns

Enforce separation of concerns

write-once, change rarely

Shared Libraries

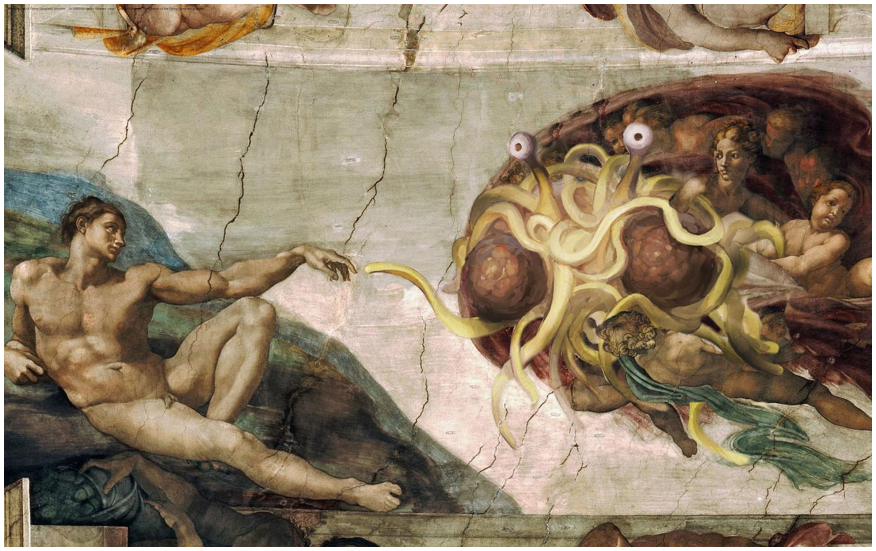
system agnostic utility code (e.g. tulib)

shared business definitions (currency.git)

pure business libraries (pno.git)

shared callbacks to frameworks
(rpc_api_fe.git)

System-Specific Code



un-refactored legacy code-base

database clients

logging and monitoring

system-specific callback modules to frameworks

stubs

minimal implementations of un-ported dependencies

exploit Erlang's weak module system

present module with same name, with stubbed function implementations

new migration related technical debt

version-controlled TODO list

Ship of Theseus Refactoring

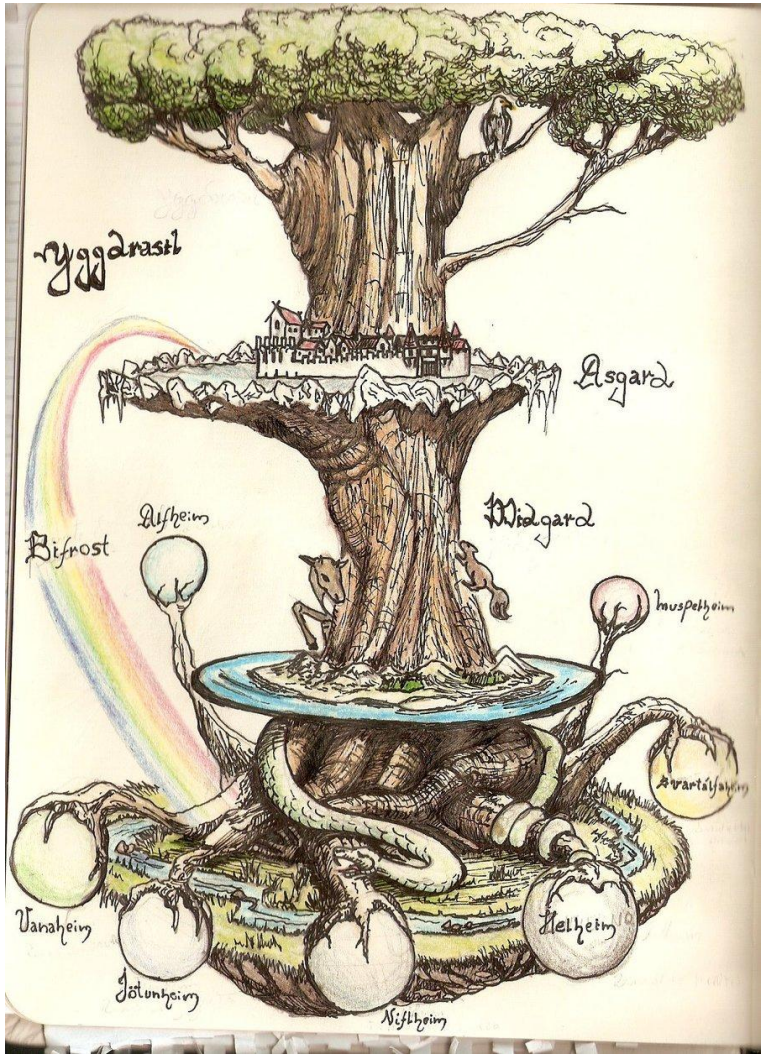
legacy code-base shrinks as shared code ecosystem grows

move logic from legacy code-base into a shared repository
-> ship result to both systems

maintenance in shared repository
-> ship result to both systems

two parallel versions of code :(
-> all maintenance must be done twice

Dependency Management



System repos must include shared code as dependencies

kred.git: lots of legacy code + lots of dependencies

fred.git: no business logic, just points to dependencies and config settings

kred: git submodules

```
dklee@gelth:~/git/klarna/dev(master)$ git diff
--- a/lib/pd
+++ b/lib/pd
@@ -1 +1 @@
-Subproject commit e6014914c1fe226f8c2a4b94f034d62897b10a0f
+Subproject commit 1bb77ce497847db15dbe56b04a7ff2fa9d196a03
```

For historical reasons, kred manages dependencies with git submodules

git submodules has a rather consistent semantics based on SHA-1 hashes of dependencies

git submodules entirely unintuitive if you expect to work with branches and tags

FRED: rebar dependencies

```
deps:
    ./rebar get-deps skip_deps=true
```

New systems use rebar.config for dependency management

Nested dependencies resolve non-deterministically

fred.git: fully-flattened dependencies

Internal fork of rebar to ignore sub-dependencies

Exposes need for really good package manager

tracking master always ~~works~~ screws you eventually

common practice: track github masters of dependencies

does not work for a financial company: unreviewed code changes slip in

full of surprises: build breaks with no code changes

does not work for reproducibility: what were the masters of all my dependencies yesterday?

Semantic Versioning

http://www.semver.org

MAJOR.MINOR.PATCH

major: breaks interface

minor: conservative extension

patch: FIX, no interface change

tagging conventions

be honest!

don't be afraid to bump minor or major!

semver bumps communicate important warnings to users

would be nice if rebar were more semver aware

Testing

rely on Jenkins jobs for master branches
of major systems

test suites with both eunit and CT
-> common_eunit in use

some use of proper

svt

System Verification Tests

multi-system integration test

acceptance testing

multi-host configuration

goal: use cloudstack + vagrant + jenkins to spin up sets of VMs for regression runs

FRED Beta

FRED is in a live Beta for legacy XML-RPC API

currently servicing an identification call (get_addresses)

work ongoing for purchase creating calls (reserve_amount, add_invoice)

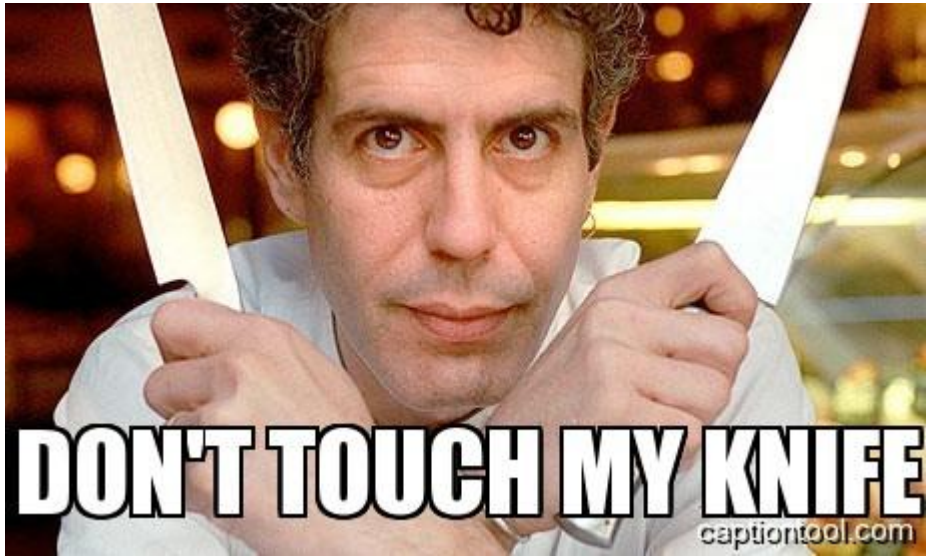
FRED DevOps

FRED is currently developer operated

mechanics in the helicopter

Configuration Management via Chef

Chef



FRED+riak servers, RabbitMQ servers managed via Chef

Chef also important in configuring test machines

Specialize to different scenarios via Chef attributes

ergonaut

JSON DSL for re-writing erlang config files

removes need for config template in Chef

Ruby Hash <-> JSON <-> erlang cfg diff

minimizes cookbook changes even though sys.config grows

Deployment

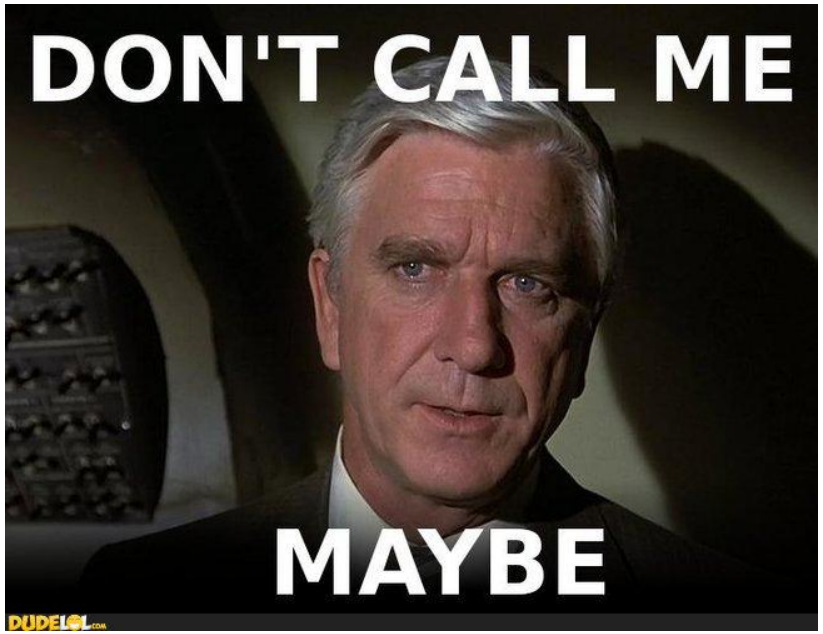
f5 BIG-IP load balancer to manage XML-RPC traffic

FREDs are upgraded independently

FREDs are stateless and redundant: chef-client restarts node

multiple releases a week

Frameworks ftw



We really like frameworks!

Important for code quality

Callback modules allow for dependency injection

Keep layers clean

Enforce separation of concerns

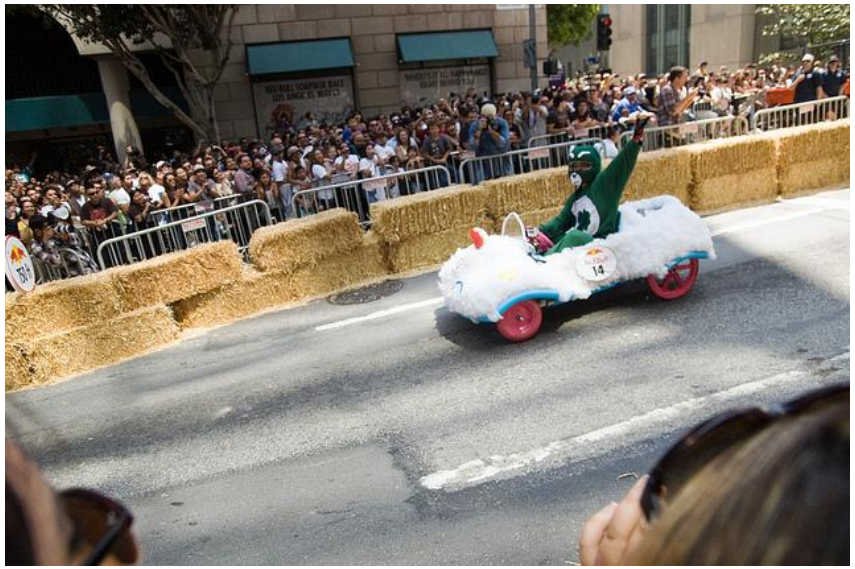
soapbox

open-sourced framework for RPC-style APIs

callback behaviours for input types and methods

separate callbacks for checking arguments and executing call

soapbox method routing



soapbox_method_router, select implementation based on input values

high degree of configurability without modifying existing method callbacks

FRED custom routing

allows us to control whether FRED or kred services API call using inputs to determine whether appropriate functionality is available

gradually service API call on FRED for more merchants as more corner-case features are ported over

sick (TM) use of parameterized modules to eliminate code duplication

lager_smtp_backend

smtp backend for lager

has a callback behaviour for specifying FROM, TO, Subject, Body based on log message

allows different users to format e-mails as desired without having to modify lager_smtp_backend code

descendent: lager_sms_backend

lager_rate_limiter_backend(lager_smtp_backend)

lager_rate_limiter_backend: generic rate limiting back-end that wraps any lager_backend

callback behaviour for how to react when rate-limiting kicks in
-> escalate to higher severity?
-> kill node?

used for both smtp and sms

lager_mq_backend + lawgalog

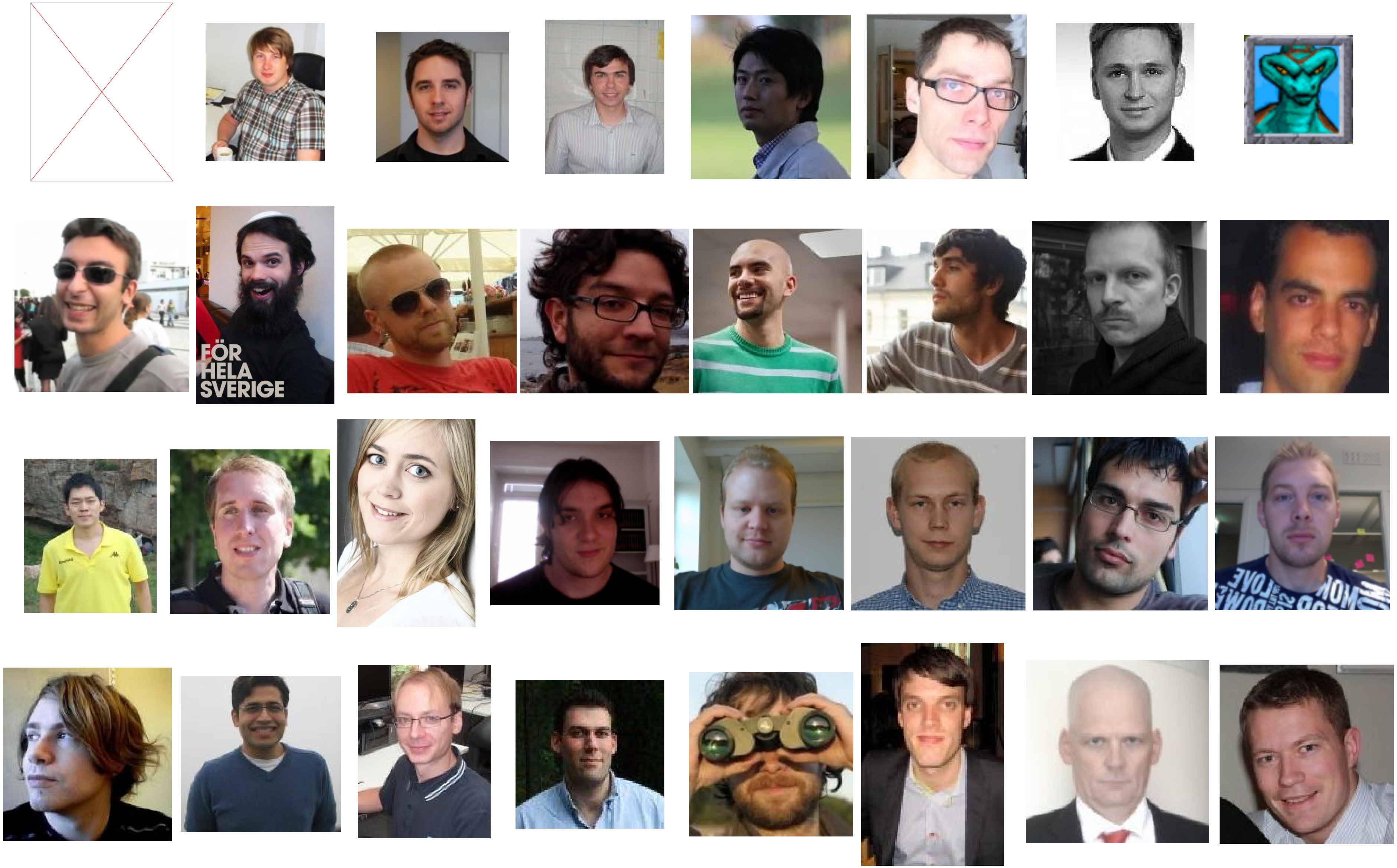
lager_mq_backend: sends lager_msg object over MQ

lawgalog: consumer that aggregates logs from multiple producers

lawgalog used to
- tail all FRED logs real-time
- evaluate experimental lager backends, backends out of place on FRED

official log aggregation/indexing done via Splunk

FRED Developers, Past and Present



Thanks! Questions?