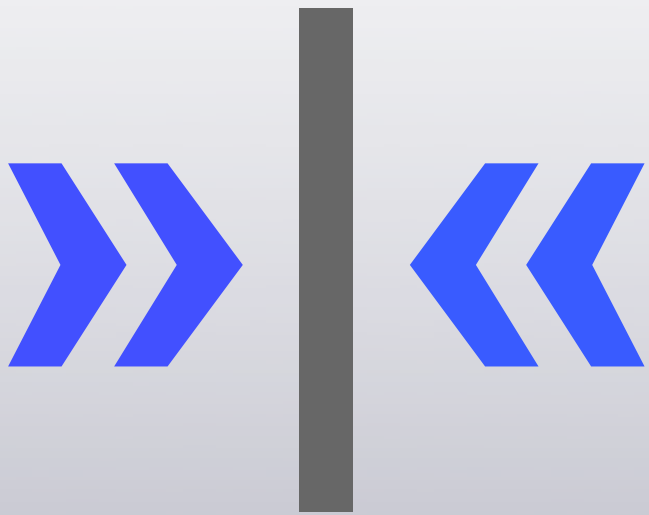
Scary Jifty

Jesse Vincent



Best Practical

Not covering...

Not covering...

- Jifty::DBI

Not covering...

- Jifty::DBI
- Continuations

Not covering...

- Jifty::DBI
- Continuations
- Actions

Not covering...

- Jifty::DBI
- Continuations
- Actions
- Regions and Fragments

Those are all intro-level topics.

Agenda

Agenda

- »|« PubSub

Agenda

- »|« PubSub
- »|« Declarative Templates

Agenda

- »|« PubSub
- »|« Declarative Templates
- »|« I18N

Agenda

- »|« PubSub
- »|« Declarative Templates
- »|« I18N
- »|« Scarier stuff

If you were here this morning, you know...

I'm Crazy

Jifty isn't just me these days...

We have 20+ committers.

Including Audrey

She's Crazy too

My goals

My goals

- ✓ Unease

My goals

- ✓ Unease
- ✓ Fear

My goals

- ✓ Unease
- ✓ Fear
- ✓ Shock

My goals

- ✓ Unease
- ✓ Fear
- ✓ Shock
- ✓ Disgust

My goals

- ✓ Unease
- ✓ Fear
- ✓ Shock
- ✓ Disgust
- ✓ Recruitment

My goals


PubSub


Comet!


Remember 1997?


Pointcast


Marimba


Comet is just Server Push


No-click page updates


How it usually works


Javascript on the client side sends a 'Subscribe to events matching '/foo/bar'


Server registers the subscription


Client connects to a special server


Client waits for server to send events


Server sends events as JSON data...


...or worse


XML


Javascript on the client side builds HTML using the event.


Javascript on client side inserts the HTML into the DOM.


What that *really* means...


You write custom Javascript to send event subscriptions to the server.


You write a custom handler to subscribe to events on behalf of the client.


You write a custom server to dispatch events to the client.


You write a javascript client to poll for events.


You write a Javascript templating engine to deal with formatting results.


You manipulate the DOM by hand to insert your events.


You spend your evenings sobbing quietly into a pint of cheap beer.


The Jifty way
(starting backwards)


Fragments are ideal for this.


Fragments

(A refresher)


On the fly page updates


They react to user input


AHAH

(not AJAX)


HTTP Request

(Javascript or browser)


HTML rendered on server


Server sends back HTML


Client inserts response in DOM


(Without JS, server returns whole transformed pages)


We extended fragments to include placement information


We wrote a "long-poll" routine for Jifty's clientside.


It connects to the built-in fragment server and waits for content.


We wrote a fragment server that takes the client's subscriptions and renders them into fragments.


(It's built into the standard Jifty server)


We wrote a simple Perl syntax for describing subscription operations.


It generates the JS invocations you need and works like the rest of our fragment stuff.


That just left the hard part.


The Message Bus


We had some nasty requirements.


Must not require additional user setup


No required external processes


No Java


(Yes, we're bigoted)


(And it's good for a cheap laugh)


Developer usability is really important

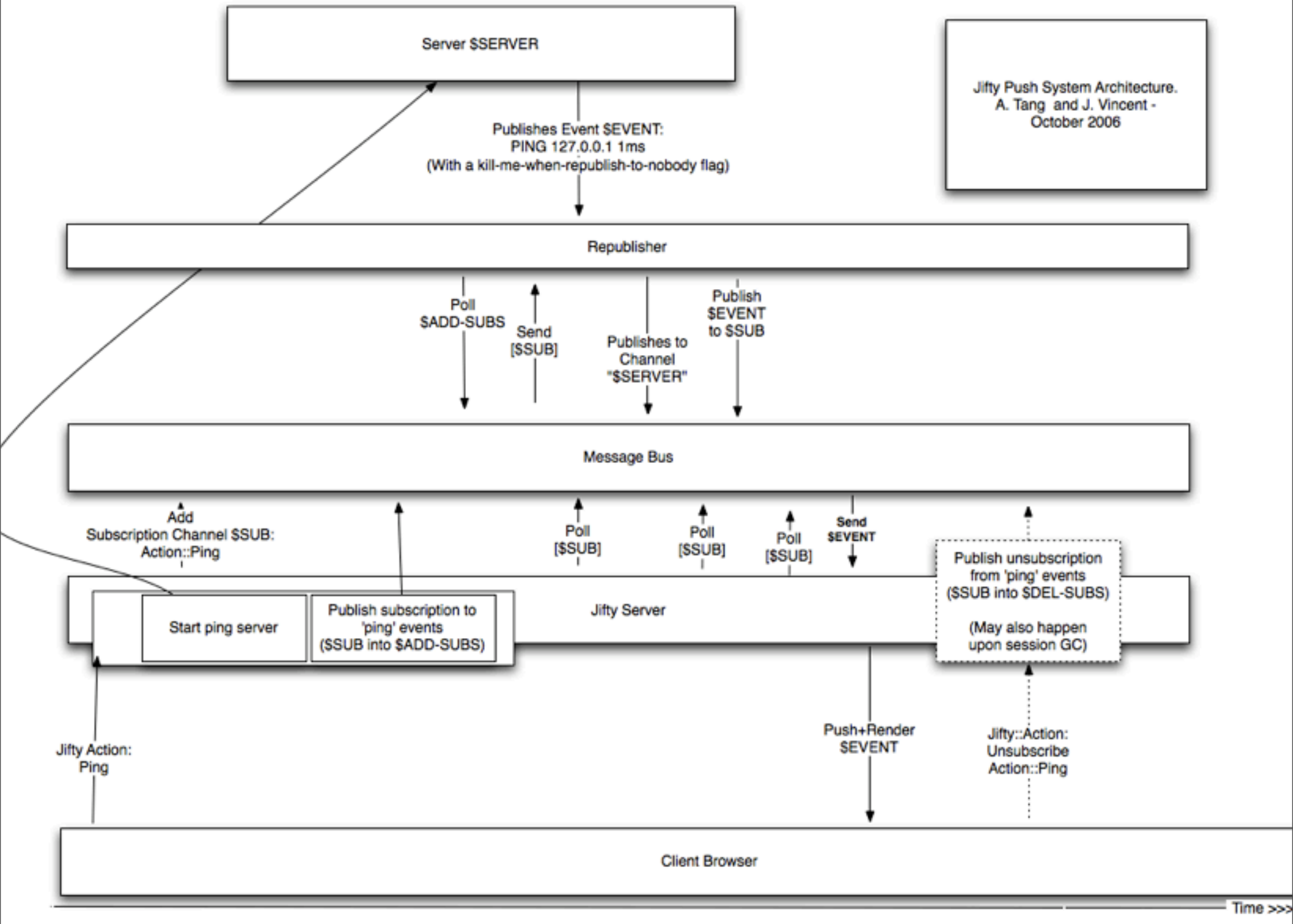

Needs to scale up


Plenty of options at the super-high-end


Needs to scale *down*


The low-end was a vast, barren plain

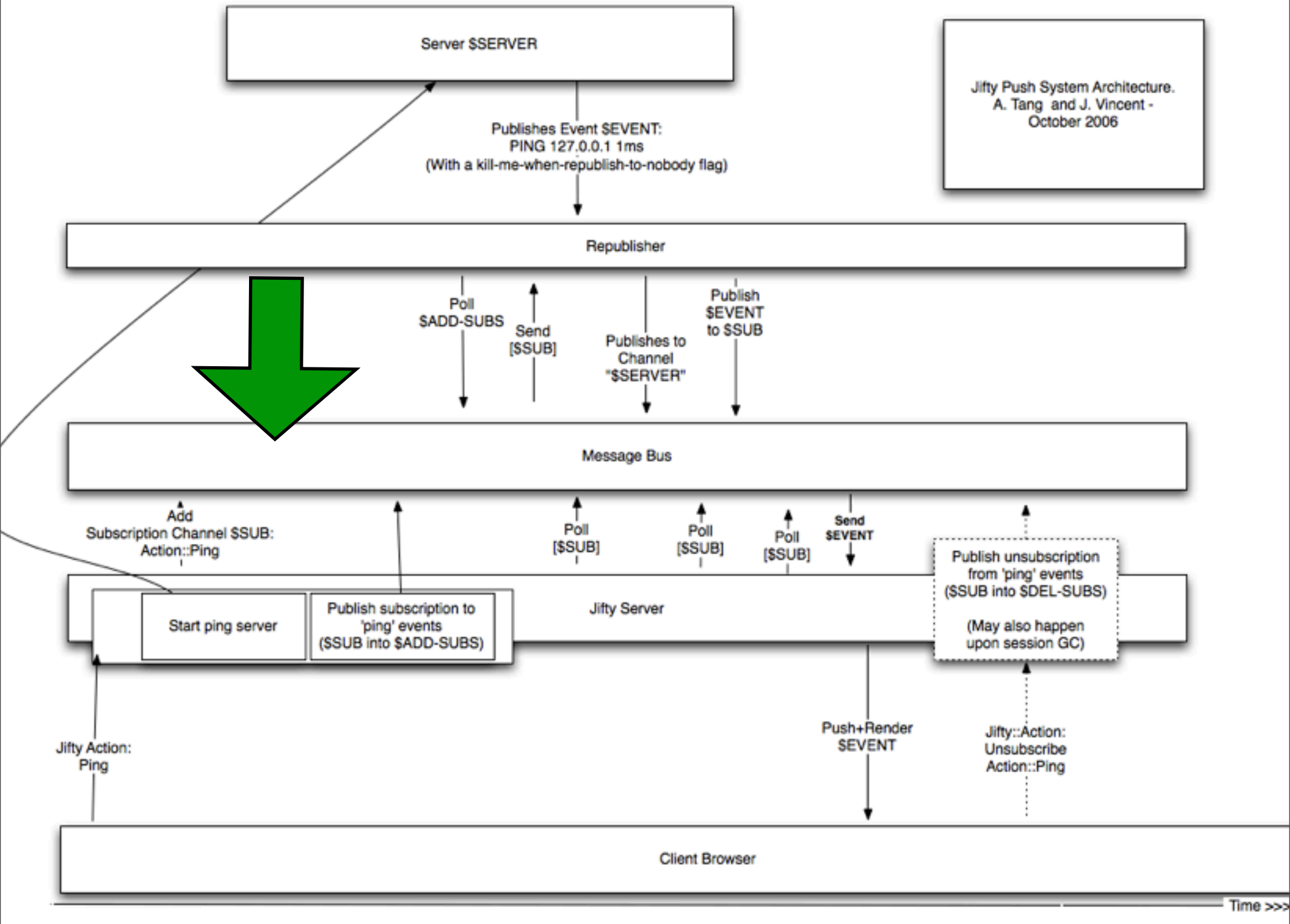

We made IPC::PubSub


Server $SERVER
Jifty Push System Architecture.
A. Tang and J. Vincent -
October 2006
Publishes Event $EVENT:
PING 127.0.0.1 1ms
(With a kill-me-when-republish-to-nobody flag)
Republisher
Poll
$ADD-SUBS
Send
[$SUB]
Publishes to
Channel
"$SERVER"
Publish
$EVENT
to $SUB
Message Bus
Add
Subscription Channel $SUB:
Action::Ping
Poll
[$SUB]
Poll
[$SUB]
Poll
[$SUB]
Send
$EVENT
Publish unsubscription
from 'ping' events
($SUB into $DEL-SUBS)
(May also happen
upon session GC)
Start ping server
Publish subscription to
'ping' events
($SUB into $ADD-SUBS)
Jifty Server
Jifty Action:
Ping
Push+Render
$EVENT
Jifty::Action:
Unsubscribe
Action::Ping
Client Browser
Time >>>

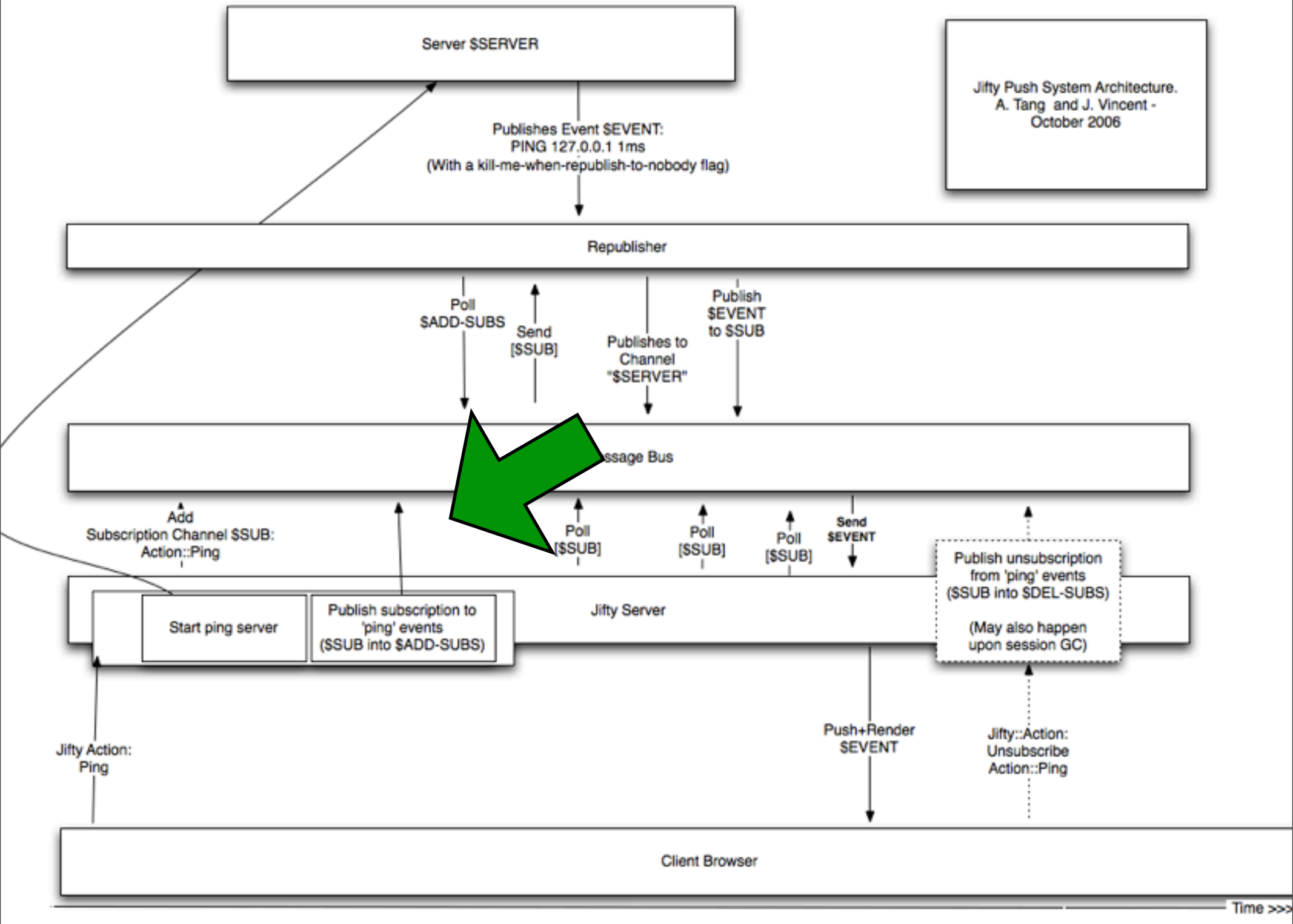

Implemented in Perl 6


Ported to Perl 5

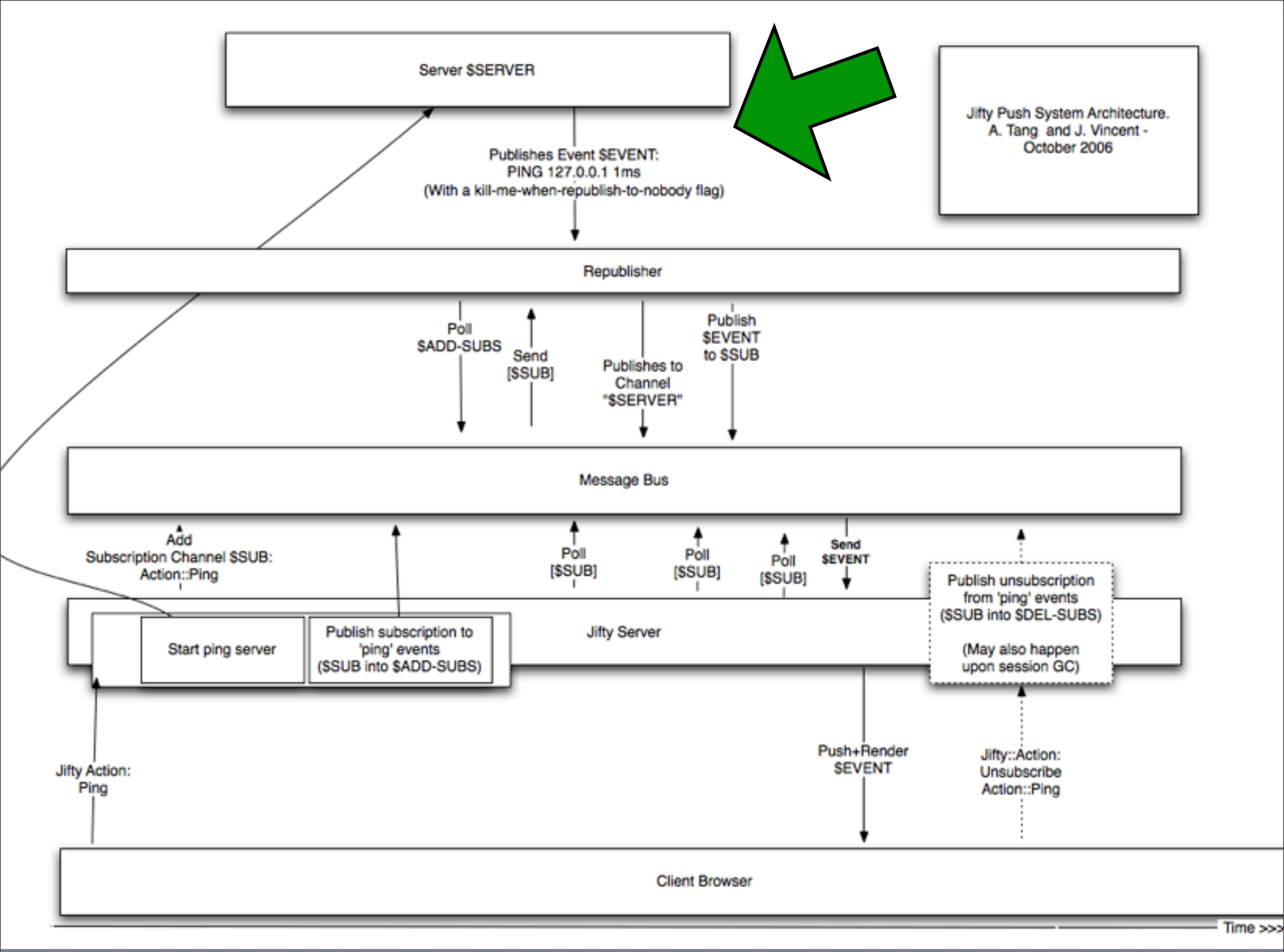

Clients subscribe to Channels of Events


Server $SERVER
Jifty Push System Architecture.
A. Tang and J. Vincent -
October 2006
Publishes Event $EVENT:
PING 127.0.0.1 1ms
(With a kill-me-when-republish-to-nobody flag)
Republisher
Poll
$ADD-SUBS
Send
[$SUB]
Publishes to
Channel
"$SERVER"
Publish
$EVENT
to $SUB
Message Bus
Add
Subscription Channel $SUB:
Action::Ping
Poll
[$SUB]
Poll
[$SUB]
Poll
[$SUB]
Send
$EVENT
Publish unsubscription
from 'ping' events
($SUB into $DEL-SUBS)
(May also happen
upon session GC)
Start ping server
Publish subscription to
'ping' events
($SUB into $ADD-SUBS)
Jifty Server
Jifty Action:
Ping
Push+Render
$EVENT
Jifty::Action:
Unsubscribe
Action::Ping
Client Browser
Time >>>

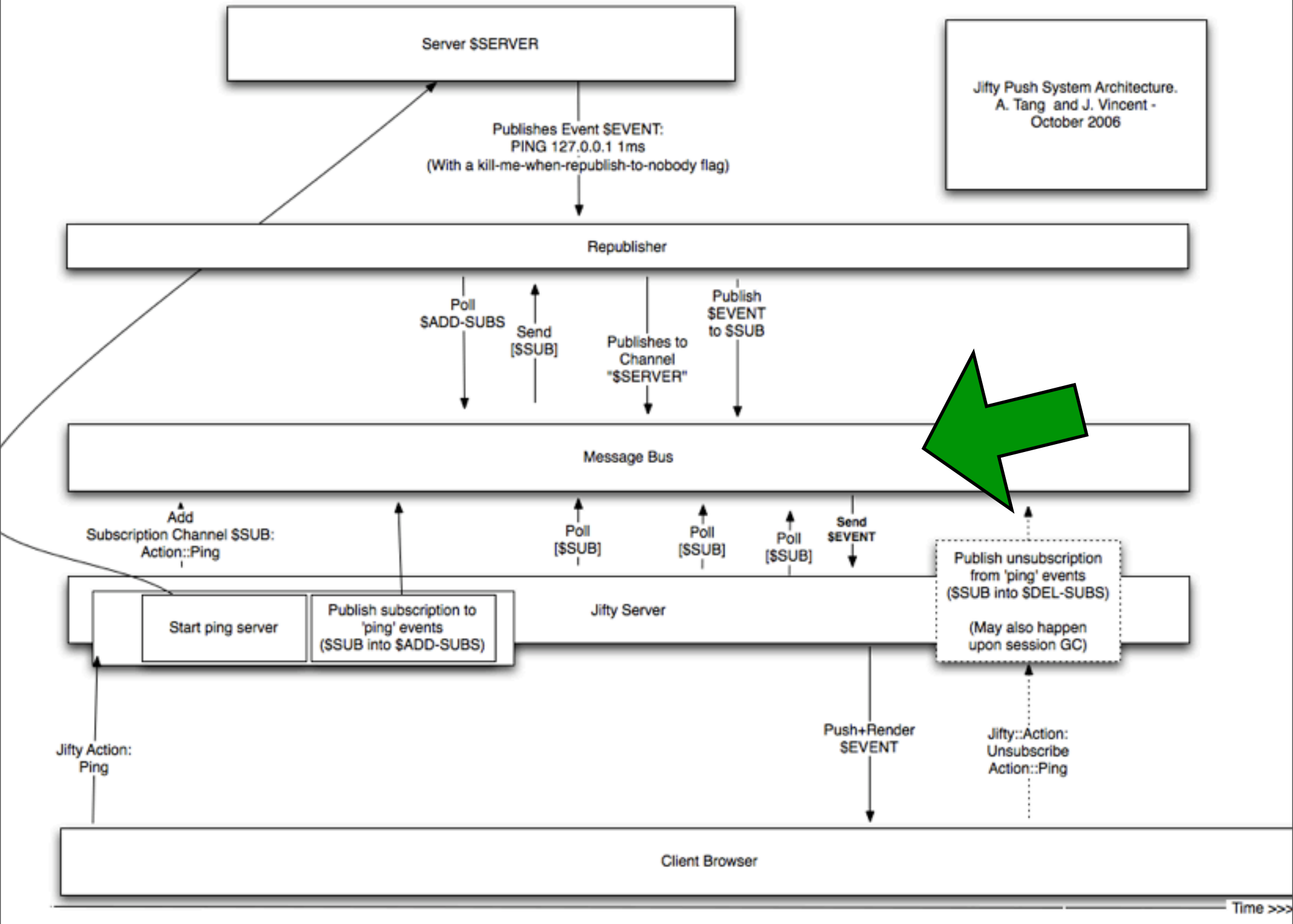

Channels are described by a Class and a Class-Specific “Search”


Server $SERVER
Jifty Push System Architecture.
A. Tang and J. Vincent -
October 2006
Publishes Event $EVENT:
PING 127.0.0.1 1ms
(With a kill-me-when-republish-to-nobody flag)
Republisher
Poll
$ADD-SUBS
Send
[$SUB]
Publishes to
Channel
"$SERVER"
Publish
$EVENT
to $SUB
ssage Bus
Add
Subscription Channel $SUB:
Action::Ping
Poll
[$SUB]
Poll
[$SUB]
Poll
[$SUB]
Send
$EVENT
Publish unsubscription
from 'ping' events
($SUB into $DEL-SUBS)
(May also happen
upon session GC)
Start ping server
Publish subscription to
'ping' events
($SUB into $ADD-SUBS)
Jifty Server
Jifty Action:
Ping
Push+Render
$EVENT
Jifty::Action:
Unsubscribe
Action::Ping
Client Browser
Time >>>

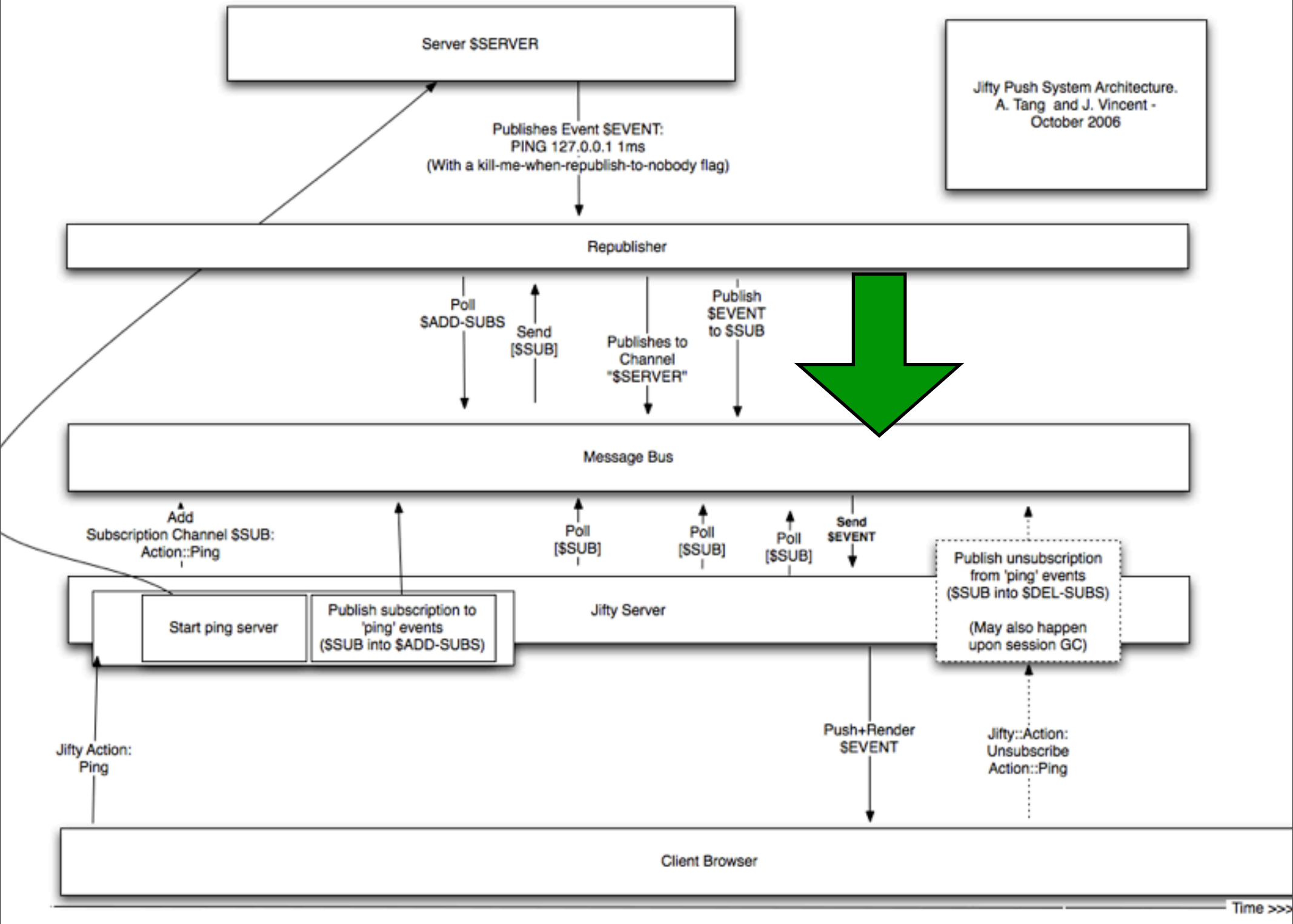

Servers publish Event streams


Server $SERVER
Jifty Push System Architecture.
A. Tang and J. Vincent -
October 2006
Publishes Event $EVENT:
PING 127.0.0.1 1ms
(With a kill-me-when-republish-to-nobody flag)
Republisher
Poll
$ADD-SUBS
Send
[$SUB]
Publishes to
Channel
"$SERVER"
Publish
$EVENT
to $SUB
Message Bus
Add
Subscription Channel $SUB:
Action::Ping
Poll
[$SUB]
Poll
[$SUB]
Poll
[$SUB]
Send
$EVENT
Publish unsubscription
from 'ping' events
($SUB into $DEL-SUBS)
(May also happen
upon session GC)
Start ping server
Publish subscription to
'ping' events
($SUB into $ADD-SUBS)
Jifty Server
Jifty Action:
Ping
Push+Render
$EVENT
Jifty::Action:
Unsubscribe
Action::Ping
Client Browser
Time >>>


“Republishers” place Events in Channels


Server $SERVER
Jifty Push System Architecture.
A. Tang and J. Vincent -
October 2006
Publishes Event $EVENT:
PING 127.0.0.1 1ms
(With a kill-me-when-republish-to-nobody flag)
Republisher
Poll
$ADD-SUBS
Send
[$SUB]
Publishes to
Channel
"$SERVER"
Publish
$EVENT
to $SUB
Message Bus
Add
Subscription Channel $SUB:
Action::Ping
Poll
[$SUB]
Poll
[$SUB]
Poll
[$SUB]
Send
$EVENT
Publish unsubscription
from 'ping' events
($SUB into $DEL-SUBS)
(May also happen
upon session GC)
Start ping server
Publish subscription to
'ping' events
($SUB into $ADD-SUBS)
Jifty Server
Jifty Action:
Ping
Push+Render
$EVENT
Jifty::Action:
Unsubscribe
Action::Ping
Client Browser
Time >>>


When a client process polls, hand it events


Server $SERVER
Jifty Push System Architecture.
A. Tang and J. Vincent -
October 2006
Publishes Event $EVENT:
PING 127.0.0.1 1ms
(With a kill-me-when-republish-to-nobody flag)
Republisher
Poll
$ADD-SUBS
Send
[$SUB]
Publishes to
Channel
"$SERVER"
Publish
$EVENT
to $SUB
Message Bus
Add
Subscription Channel $SUB:
Action::Ping
Poll
[$SUB]
Poll
[$SUB]
Poll
[$SUB]
Send
$EVENT
Publish unsubscription
from 'ping' events
($SUB into $DEL-SUBS)
(May also happen
upon session GC)
Start ping server
Publish subscription to
'ping' events
($SUB into $ADD-SUBS)
Jifty Server
Jifty Action:
Ping
Push+Render
$EVENT
Jifty::Action:
Unsubscribe
Action::Ping
Client Browser
Time >>>


The Backends


Pluggable (of course)


A plain hash


MemCached (of course)


DBM::Deep


Jifty::DBI


That's where it gets weird


Near-realtime notifications for new events


Optional full, queryable archive of historical events


In a Jifty world...


External servers publish events


Jifty::Actions publish events


CRUD operations on models are Jifty::Actions


They publish events


Client apps subscribe to (Class, Search, Renderer)


That's it


How you use it:


In-browser chat


(Because we'd all rather use Firefox than a program designed for chat)


Six files. 50 loc.

```
$ find lib share -type f |xargs wc
      18      49     428 lib/Chat/Action/Send.pm
       6      10      84 lib/Chat/Event/Message.pm
       6      11      92 lib/Chat/Server.pm
       1       3      44 share/web/templates/fragments/message
       7      38     272 share/web/templates/fragments/sender
      12      48     425 share/web/templates/index.html
      50     159    1345 total
```


share/web/templates/fragments/sender

```
<%init>
my $action = Jifty->web->new_action( class => 'Send' );
</%init>
<% Jifty->web->form->start %>
<% $action->form_field('message', focus => 1) %>
<% Jifty->web->form->submit(
    onclick => [ { submit => $action },
                 { refresh_self => 1 } ]) %>
<% Jifty->web->form->end %>
```


share/web/templates/fragments/message

```
<div>
<% ${$ARGS{event}}->{message} %>
</div>
```


share/web/templates/index.html

```
<&| /_elements/wrapper,
     title => "Jifty chat server", subtitle => "" &>

<% Jifty::Web::PageRegion->new(
        name => "message",
        path => '/__jifty/empty' )->render %>
<% Jifty::Web::PageRegion->new(
        name => "sender",
        path => '/fragments/sender' )->render %>
</&>
<%init>
Jifty->subs->add(
    class       => 'Message',
    region      => "message",
    mode        => 'Bottom',
    render_with => '/fragments/message',
);
</%init>
```


lib/Chat/Action/Send.pm

```
package Chat::Action::Send;
use warnings;
use strict;

use Jifty::Param::Schema;
use Jifty::Action schema {
    param message => label is 'Say something witty:';
};

sub take_action {
    my $self = shift;
    my $msg  = $self->argument_value('message');
    $msg = "<$1\@${ENV{'REMOTE_ADDR'}}> $msg"
     if $ENV{HTTP_USER_AGENT} =~ /([^\W\d]+)[\W\d]*$/;
    Chat::Event::Message->new( { message => $msg } )->publish;
}

1;
```


lib/Chat/Event/Message.pm

```
package Chat::Event::Message;
use strict;
use warnings;
use base 'Chat::Event';

1;
```


lib/Chat/Server.pm

```
package Chat::Server;
use base 'Jifty::Server';

sub net_server { 'Net::Server::Fork' }

1;
```


What the heck


I only
live once


A live demo!


Chat!


We have another use for this...


We're going for syncable apps next.


Template::Declare


Jifty uses HTML::Mason


TT doesn't do it for me.


I don't want to learn another language for templating.


Mason's closer.


It's Perl...


...mostly


<mason><is><still>
<full><of>
<angle><brackets>


</brackets></angle>
</of></full>
</still></is></mason>


<html><is><for>
<jedi><knights>


The rest of us are just screwed


So what's the answer?


HTML Templating is a holdover from the days of <blink>


It let designers do your layout


Now they have CSS


There's a chance to do things more sanely


There's a chance to do things less sanely


I like Perl.


I want mixins


I want inheritance


I want introspection


I want CPAN


There's Prior Art


Remember this?

```
print header, start_html('A Simple Example'),
  h1('A Simple Example'),
  start_form, "What's your name? ", textfield('name'),
  p,"What's the combination?", p,
  checkbox_group(
  -name   => 'words',
  -values => [ 'eenie', 'meenie', 'minie', 'moe' ],
  -defaults => [ 'eenie', 'minie' ]
  ),
  p, "What's your favorite color? ",
  popup_menu(
  -name   => 'color',
  -values => [ 'red', 'green', 'blue', 'chartreuse' ]
  ),
  p, submit, end_form, hr;
```


CGI.pm is still sick and wrong...


So are we.


(So are the Ruby guys)


HAML


Markaby


Template::Declare


Template::Declare

```
template ‘/page/edit.html’ => sub { p {
    form {
        div { attr { class => 'form_wrapper' }
            form_next_page( url => '/view/' . $page );
            render_param( $action => 'name');

            div { attr { class => 'inline' }
                render_param( $action => 'content', rows => 30 )
            }
            div { attr { class => 'inline' }
                form_submit( label => 'Create' )
            }
        }
    }
    show('markup');
} };
```


We read perlsub.pod very carefully and are abusing every single line of it.

- A. Tang, December 2006


```
=head1 BUGS

Crawling all over, baby. Be
very, very careful. This
code is so cutting edge, it
can only be fashioned from
carbon nanotubes.
```


The Mundane


Faking up inheritance

```
foreach my $parent (@{"$pkg\::ISA"}) {
  $parent->can('has_template') or next;
  $rv = $parent->has_template($name);
  return $rv if $rv;
}
```


The Obscure


Naming the anonymous

```
sub install_tag {
  my $tag  = lc( $_[0] );
  my $name = $tag;

  push @EXPORT, $tag;

  no strict 'refs';
  no warnings 'redefine';
  *$tag = sub (&;$)
     { local *__ANON__ = $tag; _tag(@_) };
}
```


The Obscure


Optionally gobbling

```
sub install_tag {
   my $tag  = lc( $_[0] );
   my $name = $tag;

   push @EXPORT, $tag;

   no strict 'refs';
   no warnings 'redefine';
   *$tag = sub (&;$)
       { local *__ANON__ = $tag; _tag(@_) };
}
```


...for readability

```
p { b { ‘Basic’ }
    i { ‘Important’ }
    u { ‘Unusual’ }
  }
```

```
p { b { ‘Basic’ };
    i { ‘Important’ };
    u { ‘Unusual’ };
  }
```


The Wrong


Functional syntax, OO Semantics

```
sub show {
    local $self = shift
       if ( $_[0]->isa('Template::Declare') );
    my $template = shift;
    my $buf      = '';

    {
      ...
```

We're not ditching Mason.


The Plan


You can cross the streams


The refactoring should make it *easier* to add TT


But maybe you won't want it anymore.


“T-D is the first thing I'd consider using instead of TT”

- Matt Trout, #jifty, December 2006


nternationalizatio


I18N


Problem: Every string in your code should be localized.


In RT, we create a loc(‘’) method in our base class...in *every* base class.


I guess we could use a crazy Exporter and spew it all over the symbol table.


i18n.pm:

~~“foo bar” is an overloaded i18n object


autobox.pm:
I don't even want to think about it


Jifty::I18N


One exposed method


Same problem as RT?


Nope.


_ is special.


Only Larry gets ‘.’


But everybody gets ‘ ’


It's always in the current package.


_('Cat');
Just works.
Anywhere


There's always a catch.


Other code can get _() confused with....


Some smartass in the audience want to shout out the answer?


Right.

The stat() cache.


It all still works fine

```
my $loc_method = sub {
  # Retain compatibility
  # with "-e _" etc.
  return \*_ unless @_;
  ...
};


{
  no strict 'refs';
  no warnings 'redefine';
  *_ = $loc_method;
}
```

Fixed in 5.10!


I’m not even going to get into:


Blessing an object into package ‘0’


...so ref() returns false.


```
local *is::AUTOLOAD
    = sub {...};
```


to create a ‘=>’ workalike


Or why I let Audrey check in this code...


...that MD5s the call stack

```
use Digest::MD5 qw(md5_hex);
my $frame = 1;
my @stack = (ref($self) || $self);
while (my ($pkg, $filename, $line) = caller($frame++)) {
  push @stack, $pkg, $filename, $line;
}

my $digest = md5_hex("@stack");
```


If I haven't scared you off yet...


If I haven't scared you off yet...

We're hiring


If I haven't scared you off yet...

We're hiring

Unfortunately...

If I haven't scared you off yet...

We're hiring

Unfortunately...

We pay in dollars