O'REILLY®

oscon

open source convention

oscon.com

Performance vs Scalability

Blocking or non-blocking code, code complexity and architectural blocks to avoid code complexity

$whoami

work: http://locaweb.com

site: http://7co.cc

blog: http://zenmachine.wordpress.com

code: http://github.com/gleicon

RestMQ: http://restmq.com

twitter: @gleicon

Required Listening

JETHRO TULL
FEATURE ON PAGE 3

The St. Cleve Chronicle & Linwell Advertiser

Friday January 7th 1972 — No. 1003 — Price 3p Weekly

THICK AS A BRICK

JUDGES DISQUALIFY "LITTLE MILTON" IN LAST MINUTE RUMPUS

THE SOCIETY FOR LITERARY ADVANCEMENT AND GESTATION, (SLAG), announced their decision late last night to disqualify eight year old prizewinner Gerald (Little Milton) Bostock following the hundreds of protests and threats received after the reading of his epic poem "Thick as a Brick" on B.B.C. Television last Monday night.

A hastily reconvened panel of Judges accepted the decision by four leading child psychiatrists that the boy's mind was seriously unbalanced and that his work was a product of an "extremely unwholesome attitude towards life, his God and Country". Bostock was recommended for psychiatric treatment following examination "without delay". The first prize will now be presented to runner up Mary Whiteyard (aged 12) for her essay on Christian ethics entitled, "He died to save the little Children".

The Literary Competition, which was for children aged from 7 to 16 years of age, was sponsored by leading national newspapers and received thousands of entries from schools all over Britain. Mr. Humphrey Martin, the Headmaster of Moordale Primary School said Gerald, nicknamed "Little Milton" by his English master because of his poetic ability, was mentally advanced for his age, although inclined on occasions to obscure and verbose assertions which led him to being somewhat un-popular with his schoolmates. He went on to say that without doubt the child had a great future academically and that his progress was unsurpassed in the history of Moordale Primary. Gerald and his parents moved to St. Cleve four years ago from Manchester when Mr. Bostock decided for health reasons to live away from the City. David Bostock now does occasional gardening work while his wife Daphne is well known to the Congregation of St. Cleve Parish Church for her activities in social work and her wonderful buffet luncheon at the fete last Saturday. Well done, Daphne! Mr. Bostock said this morning of "Little Milton's" disqualification, "We are heartbroken at the way the Judges changed their minds, and the loss of the prize money and scholarship means we shall find difficulty in paying the instalments on Gerald's Encyclopaedia Britannica. I shall have to do Dr. Munson's roses next week after all." When he heard of the decision against him, Gerald went to his room and locked the door. "Mrs. Bostock and I are sorely vexed at the way this has turned out", said Mr. Bostock of No. 6 Pollitt Close, St. Cleve.

Many local residents are also annoyed and hurt by the news and as some consolation to Gerald and his parents the St. Cleve Chronicle prints the full text of the disqualified poem this week on page 7.

G— —r

Many of the viewers who heard Gerald read his work on the "Young Arts" programme on B.B.C. 2 felt that it was not one poem but a series of separate poems put together merely to appear impressive. Many of the viewers' complaints were centred around "Little Milton's" use of a four-letter

(Continued on Page 4, Col. 6)

Flashback to last week's presentation dinner held in Gerald's honour by the Committee of the St. Cleve District Art and Literary Society at the Parrit Rooms. Left to right: Lord Clive 'Polly' Parritt, Mr. and Mrs. Bostock, Gerald Bostock, Lady Parrit, Julia, Gerald's chum with whom he writes poems.

FLUTE CONCERT AT PARRIT ROOMS

A concert by the Hyde Close Flute and Cello Ensemble was held in the Parrit Rooms, Flood Street, St. Cleve, on Tuesday night. A sensitive and, dare one say, passionate, performance was rendered by the five musicians but was spoilt at times by an exceedingly restless and shifyt audience. The ensemble is well known and liked for its spirited and extroverted style, but some of the quieter and more meaningful passages were lost on an otherwise enthusiastic audience. The music included two sections of the Tycho Asavrick Suite in F major by Jeffrey Teller and a contemporary piece by Ena Sanderone. J.S-M.

HEAD INJURY

Fifty-two year old Sarah Pickles of the High Street, St. Cleve, cut her head when she tripped over while walking in the High Street, St. Cleve.

"But your new shoes are worn at the heels
and your suntan does rapidly peel
and your wise men don't know how it feels
to be thick as a brick."

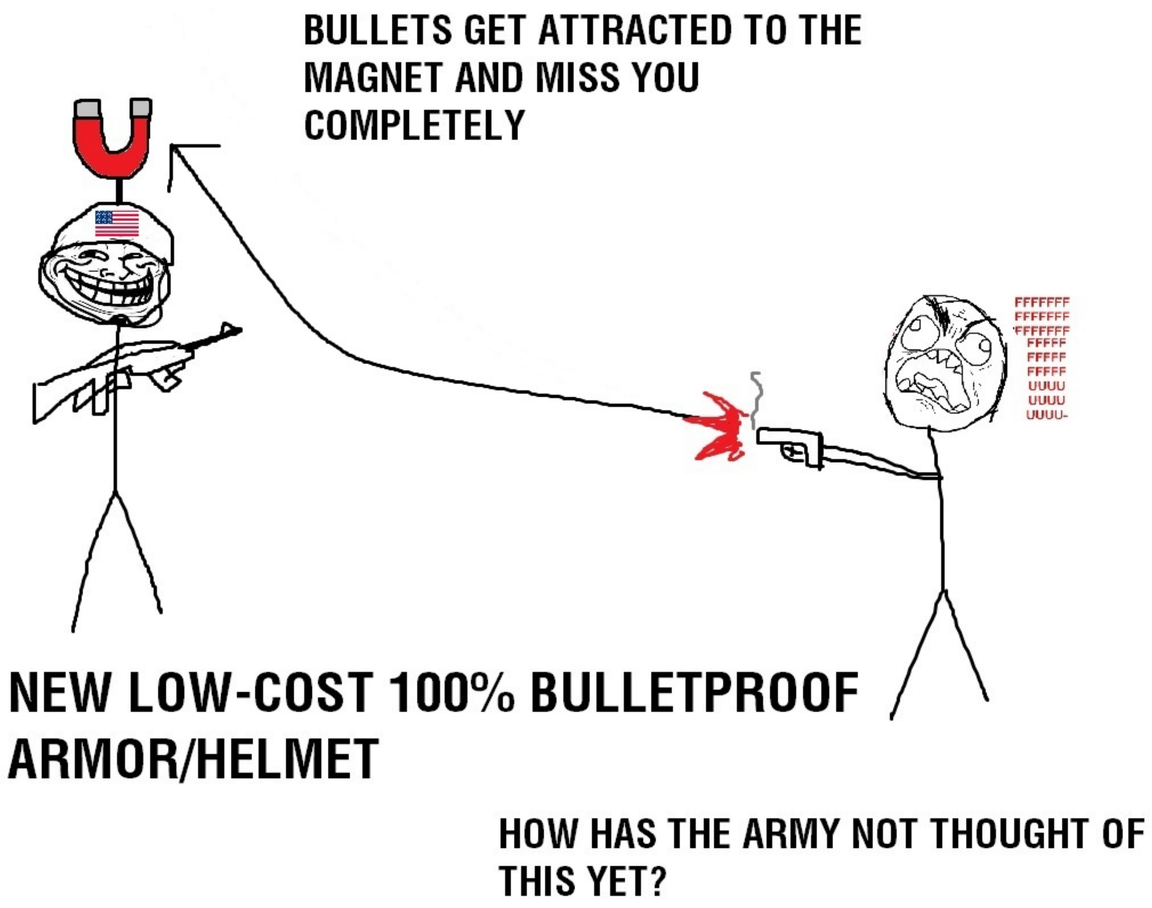
Objective

- To discuss how scalability sometimes get confused with performance when speaking about non-blocking I/O
- To show how most of languages have frameworks to deal with non-blocking I/O
- Examine an example in different languages and the basic blocks of these frameworks
- Discuss that this programming model is mandatory, not even migrate from Blocking to non-blocking I/O is needed for most cases. The best solution may not lie in code complexity, but in the right architecture.
- Discuss a bit about Message Queues, Cache and Redis

Basics

- Asynchronous I/O - Posix aio_*
- Non-Blocking I/O - Select, Poll, EPoll, Kqueue
- Reactor - Pattern for multiplexing I/O
- Future Programming – Generators and Deferreds
- Event Emitters
- The c10k problem - http://www.kegel.com/c10k.html

Scalability, not performance



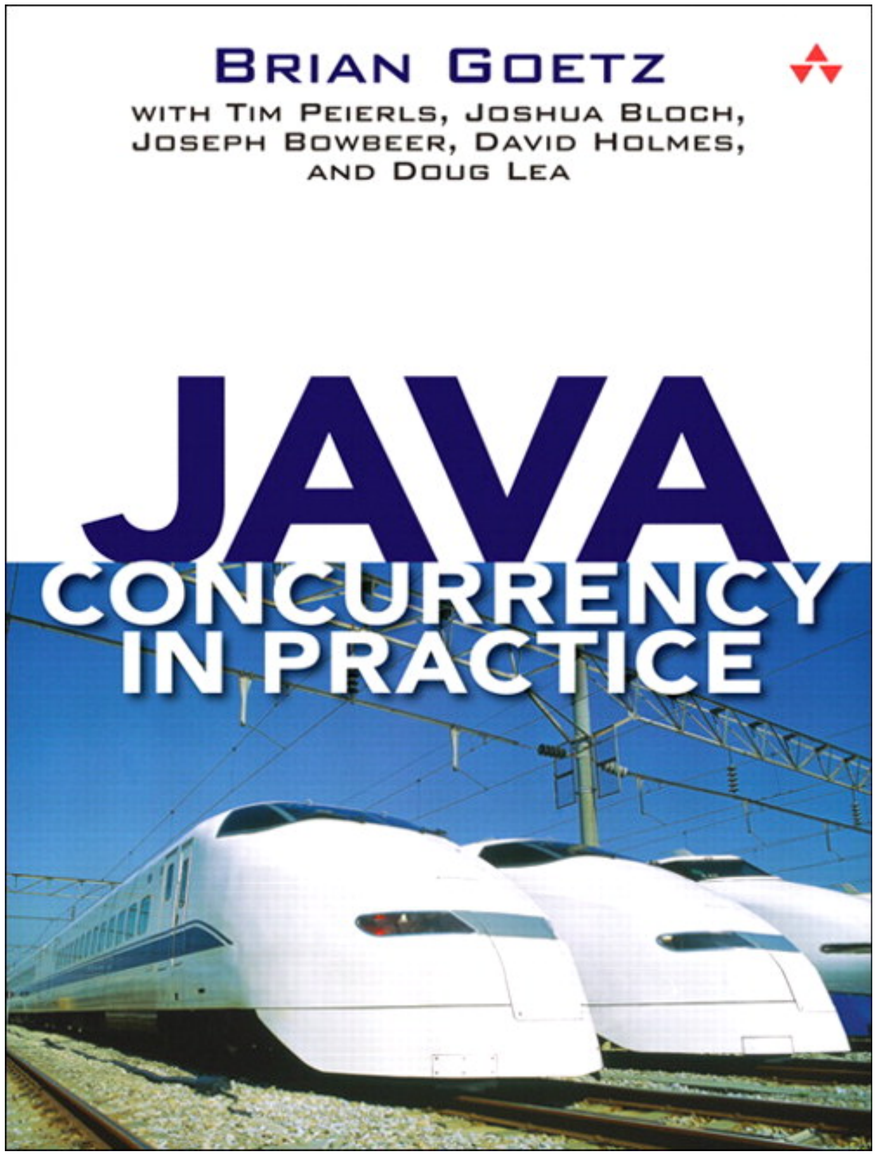
Scalability, not performance

1. Performance: *same work with less resources*
2. Scalability: *more work with added resources*
3. Scientific measures for performance: *How much and How fast*
4. Tricks to improve performance can hurt scalability.
5. Non-Blocking I/O is not an optimization for Performance
6. The time spent waiting for an external resource can be used to other tasks, given a known introduced complexity in the code.
7. Most of time architectural changes can avoid deeper code changes

(1,2,3,4 - Goetz et al, Java Concurrency in Practice)

(5,6,7 - Lucindo, R. http://slidesha.re/aYz4Mk, wording on 6 is mine)

JAVA concurrency in practice



Blocking code example - Download a list of URLs

Ruby

```ruby
require 'open-uri'

urls = ['http://www.google.com', 'http://yahoo.com']
urls.each { |u| puts open(u).read }
```

Python

```python
import urllib2

urls = ['http://www.google.com', 'http://www.yahoo.com']
for u in urls: print urllib2.urlopen(u).read()
```

Benchmark

- Compare blocking and non-blocking Ruby code to download a list of URLs
- A naive example to examine the effects of both paradigms on the final code
- The ‘algorithm’ to download an URL is (should be) the same in both versions
- If we had more cores, or more powerful cores, it should not affect the performance of a single download (I/O bound operation)

Benchmark - Blocking

```
require 'open-uri'

urls = open(ARGV.first).read.split
urls.each { |u|
  begin
    puts open(u).read
  rescue
    puts '404 - #{u}'
  end
}
```

Benchmark – Blocking + Threads

```
require 'open-uri'

threads = []
urls = open(ARGV.first).read.split
urls.each { |u|
  threads << Thread.new(u) { |t|
    begin
      puts open(t).read
    rescue
      puts '404 - #{t}'
    end
    }

}

threads.each { |t| t.join() }
```

Benchmark - Non-Blocking (Eventmachine based)

```ruby
require 'rubygems'
require 'eventmachine'
require 'em-http'

urls = open(ARGV.first).read.split
pending = urls.size

EventMachine::run do
  urls.each do |u|
    defer = EM::HttpRequest.new(u).get :redirects => 1, :timeout => 10

    defer.callback do |r|
      puts u + ': ' + r.response
      pending -= 1
      EM.stop if pending < 1
    end

    defer.errback do |msg, error|
      puts defer.error
      pending -= 1
      EM.stop if pending < 1
    end
  end
end
```

Benchmark - Results

2 urls
Blocking I/O: 0m4.061s
Blocking I/O + Threads: 0m2.914s
Non-blocking I/O: 0m1.027s

100 urls
Blocking I/O: 2m46.769s
Blocking I/O + Threads: 0m33.722s
Non-blocking I/O: 0m11.495s

Python and Twisted

```
from twisted.internet import reactor, defer
from twisted.web.client import getPage
from twisted.internet.defer import DeferredList

def listCallback(results):
    print results

def finish(ign):
    reactor.stop()

def test():
    d1 = getPage('http://www.google.com')
    d2 = getPage('http://yahoo.com')
    dl = DeferredList([d1, d2])
    dl.addCallback(listCallback)
    dl.addCallback(finish)

@defer.inlineCallbacks
def test2():
    d1 = yield getPage('http://www.google.com')
    d2 = yield getPage('http://yahoo.com')
    print '%s\n%s' % (d1, d2)

test()
test2()
reactor.run()
```

Python and Twisted – callback styles

```
from twisted.internet import reactor, defer
from twisted.web.client import getPage
from twisted.internet.defer import DeferredList

def listCallback(results):
    print results

def finish(ign):
    reactor.stop()

def test():
    d1 = getPage('http://www.google.com')
    d2 = getPage('http://yahoo.com')
    dl = DeferredList([d1, d2])
    dl.addCallback(listCallback)
    dl.addCallback(finish)

@defer.inlineCallbacks
def test2():
    d1 = yield getPage('http://www.google.com')
    d2 = yield getPage('http://yahoo.com')
    print '%s\n%s' % (d1, d2)

test()
test2()
reactor.run()
```

Explicit

Inline

Python and Twisted – Generators

```
from twisted.internet import reactor, defer
from twisted.web.client import getPage
from twisted.internet.defer import DeferredList

def listCallback(results):
    print results

def finish(ign):
    reactor.stop()

def test():
    d1 = getPage('http://www.google.com')
    d2 = getPage('http://yahoo.com')
    dl = DeferredList([d1, d2])
    dl.addCallback(listCallback)
    dl.addCallback(finish)

@defer.inlineCallbacks
def test2():
    d1 = yield getPage('http://www.google.com')
    d2 = yield getPage('http://yahoo.com')
    print '%s\n%s' % (d1, d2)

test()
test2()
reactor.run()
```

Generators

Python and Twisted – Deferreds

Deferreds

Not Deferreds

```
from twisted.internet import reactor, defer
from twisted.web.client import getPage
from twisted.internet.defer import DeferredList

def listCallback(results):
    print results

def finish(ign):
    reactor.stop()

def test():
    d1 = getPage('http://www.google.com')
    d2 = getPage('http://yahoo.com')
    dl = DeferredList([d1, d2])
    dl.addCallback(listCallback)
    dl.addCallback(finish)

@defer.inlineCallbacks
def test2():
    d1 = yield getPage('http://www.google.com')
    d2 = yield getPage('http://yahoo.com')
    print '%s\n%s' % (d1, d2)

test()
test2()
reactor.run()
```

Python and Twisted – Callbacks

Callbacks

```
from twisted.internet import reactor, defer
from twisted.web.client import getPage
from twisted.internet.defer import DeferredList

def listCallback(results):
    print results

def finish(ign):
    reactor.stop()

def test():
    d1 = getPage('http://www.google.com')
    d2 = getPage('http://yahoo.com')
    dl = DeferredList([d1, d2])
    dl.addCallback(listCallback)
    dl.addCallback(finish)

@defer.inlineCallbacks
def test2():
    d1 = yield getPage('http://www.google.com')
    d2 = yield getPage('http://yahoo.com')
    print '%s\n%s' % (d1, d2)

test()
test2()
reactor.run()
```

Python and Twisted – Inline callbacks

```
from twisted.internet import reactor, defer
from twisted.web.client import getPage
from twisted.internet.defer import DeferredList

def listCallback(results):
    print results

def finish(ign):
    reactor.stop()

def test():
    d1 = getPage('http://www.google.com')
    d2 = getPage('http://yahoo.com')
    dl = DeferredList([d1, d2])
    dl.addCallback(listCallback)
    dl.addCallback(finish)

@defer.inlineCallbacks
def test2():
    d1 = yield getPage('http://www.google.com')
    d2 = yield getPage('http://yahoo.com')
    print '%s\n%s' % (d1, d2)

test()
test2()
reactor.run()
```

Results from getPage

Python and GEvent

```
import gevent
from gevent import monkey

monkey.patch_all()

urls = ['http://www.google.com','http://www.yahoo.com','http://reallyslowsite.com/10']

import urllib2

def fetch(url):
    data = urllib2.urlopen(url).read()
    print '%s: %s' % (url, data)

jobs = [gevent.spawn(fetch, url) for url in urls]

gevent.joinall(jobs)
```

Monkey patch the socket module

Python and GEvent

```
import gevent
from gevent import monkey

monkey.patch_all()

urls = ['http://www.google.com','http://www.yahoo.com','http://reallyslowsite.com/10']

import urllib2

def fetch(url):
    data = urllib2.urlopen(url).read()
    print '%s: %s' % (url, data)
```

```
jobs = [gevent.spawn(fetch, url) for url in urls]

gevent.joinall(jobs)
```

Create and start greenlets

Ruby and EventMachine

```ruby
require 'rubygems'
require 'eventmachine'
require 'em-http'

urls = open(ARGV.first).read.split
pending = urls.size

EventMachine::run do
  urls.each do |u|
    defer = EM::HttpRequest.new(u).get :redirects => 1, :timeout => 10

    defer.callback do |r|
      puts u + ': ' + r.response
      pending -= 1
      EM.stop if pending < 1
    end

    defer.errback do |msg, error|
      puts defer.error
      pending -= 1
      EM.stop if pending < 1
    end
  end
end
```

Ruby and EventMachine - Generators

Generator

```
require 'rubygems'
require 'eventmachine'
require 'em-http'

urls = open(ARGV.first).read.split
pending = urls.size

EventMachine::run do
  urls.each do |u|
    defer = EM::HttpRequest.new(u).get :redirects => 1, :timeout => 10

    defer.callback do |r|
      puts u + ': ' + r.response
      pending -= 1
      EM.stop if pending < 1
    end

    defer.errback do |msg, error|
      puts defer.error
      pending -= 1
      EM.stop if pending < 1
    end
  end
end
```

Ruby and EventMachine - Deferreds

Sort of deferred

```ruby
require 'rubygems'
require 'eventmachine'
require 'em-http'

urls = open(ARGV.first).read.split
pending = urls.size

EventMachine::run do
  urls.each do |u|
    defer = EM::HttpRequest.new(u).get :redirects => 1, :timeout => 10

    defer.callback do |r|
      puts u + ': ' + r.response
      pending -= 1
      EM.stop if pending < 1
    end

    defer.errback do |msg, error|
      puts defer.error
      pending -= 1
      EM.stop if pending < 1
    end
  end
end
```

Ruby and EventMachine - Callbacks

Success and error callbacks

```
require 'rubygems'
require 'eventmachine'
require 'em-http'

urls = open(ARGV.first).read.split
pending = urls.size

EventMachine::run do
  urls.each do |u|
    defer = EM::HttpRequest.new(u).get :redirects => 1, :timeout => 10

    defer.callback do |r|
      puts u + ': ' + r.response
      pending -= 1
      EM.stop if pending < 1
    end

    defer.errback do |msg, error|
      puts defer.error
      pending -= 1
      EM.stop if pending < 1
    end
  end
end
```

Ruby, EventMachine and EM-Synchrony

```ruby
require 'em-synchrony'
require 'em-synchrony/em-http'

EM.synchrony do

  d1 = EM::HttpRequest.new("http://www.google.com").get
  d2 = EM::HttpRequest.new("http://www.yahoo.com").get

  puts 'Google: ' + d1.response
  puts 'Yahoo ' + d2.response

  EM.stop

end
```

Ruby, EventMachine and EM-Synchrony

```ruby
require 'em-synchrony'
require 'em-synchrony/em-http'

EM.synchrony do

  d1 = EM::HttpRequest.new("http://www.google.com").get
  d2 = EM::HttpRequest.new("http://www.yahoo.com").get

  puts 'Google: ' + d1.response
  puts 'Yahoo ' + d2.response

  EM.stop

end
```

Inline results

Node.js

```
var http = require('http');

var d1 = http.createClient(80, 'www.google.com')
      .request('GET', '/', {'host': 'www.google.com'});

var d2 = http.createClient(80, 'www.yahoo.com')
      .request('GET', '/', {'host': 'www.yahoo.com'});

var g_content;
var y_content;

d1.on('response', function (res) {
  res.on('data', function (chunk) { g_content = g_content + chunk; });
  res.on('end', function() { console.log("Google: " + g_content); });
});

d2.on('response', function (res) {
  res.on('data', function (chunk) { y_content = y_content + chunk; });
  res.on('end', function() { console.log("Yahoo: " + y_content);});
});

d1.end();
d2.end();
```

Node.js – Creating and ending the requests

Start requests

```
var http = require('http');

var d1 = http.createClient(80, 'www.google.com')
      .request('GET', '/', {'host': 'www.google.com'});

var d2 = http.createClient(80, 'www.yahoo.com')
      .request('GET', '/', {'host': 'www.yahoo.com'});

var g_content;
var y_content;

d1.on('response', function (res) {
  res.on('data', function (chunk) { g_content = g_content + chunk; });
  res.on('end', function() { console.log("Google: " + g_content); });
});

d2.on('response', function (res) {
  res.on('data', function (chunk) { y_content = y_content + chunk; });
  res.on('end', function() { console.log("Yahoo: " + y_content);});
});
```

End requests

```
d1.end();
d2.end();
```

Node.js – Events handlers and callbacks

Response event

```
var http = require('http');

var d1 = http.createClient(80, 'www.google.com')
      .request('GET', '/', {'host': 'www.google.com'});

var d2 = http.createClient(80, 'www.yahoo.com')
      .request('GET', '/', {'host': 'www.yahoo.com'});

var g_content;
var y_content;

d1.on('response', function (res) {
  res.on('data', function (chunk) { g_content = g_content + chunk; });
  res.on('end', function() { console.log("Google: " + g_content); });
});

d2.on('response', function (res) {
  res.on('data', function (chunk) { y_content = y_content + chunk; });
  res.on('end', function() { console.log("Yahoo: " + y_content);});
});

d1.end();
d2.end();
```

Erlang

```
-module(erlang_httpclient_example).
-export([start/0, stop/0, getpage/1]).

start() ->
    io:format("Parent pid: ~w~n", [self()]),
    inets:start(),
    Urls = ["http://www.google.com", "http://www.yahoo.com"],
    Pids = lists:map(fun(U)-> spawn(fun() -> getpage(U) end) end, Urls).

stop() ->
    inets:stop().

getpage(Url) ->
    io:format("Url: ~s (pid: ~w)~n", [Url, self()]),
    {ok, {{_Vers, 200, _Rson}, _Hdrs, B}} = httpc:request(get, {Url, []}, [], []),
    io:format("Body: ~s~n", [B]).
```

Erlang – Spawning one download process per url

```
-module(erlang_httpclient_example).
-export([start/0, stop/0, getpage/1]).

start() ->
    io:format("Parent pid: ~w~n", [self()]),
    inets:start(),
    Urls = ["http://www.google.com", "http://www.yahoo.com"],
    Pids = lists:map(fun(U)-> spawn(fun() -> getpage(U) end) end, Urls).

stop() ->
    inets:stop().

getpage(Url) ->
    io:format("Url: ~s (pid: ~w)~n", [Url, self()]),
    {ok, {{_Vers, 200, _Rson}, _Hdrs, B}} = httpc:request(get, {Url, []}, [], []),
    io:format("Body: ~s~n", [B]).
```

A reminder of what we wanted to do

Blocking code example - Download a list of URLs

Ruby

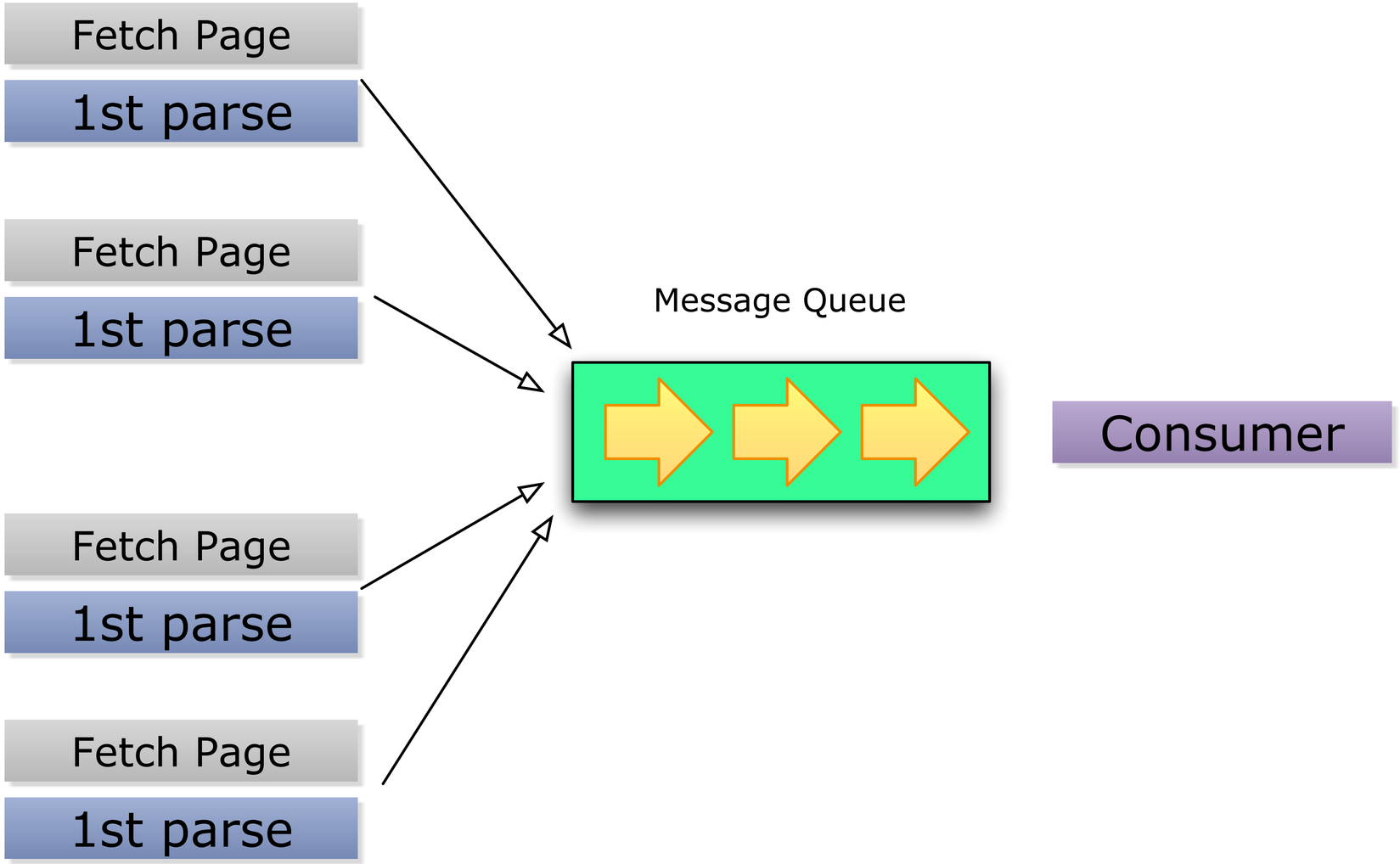
```ruby
require 'open-uri'

urls = ['http://www.google.com', 'http://yahoo.com']
urls.each { |u| puts open(u).read }
```

Python

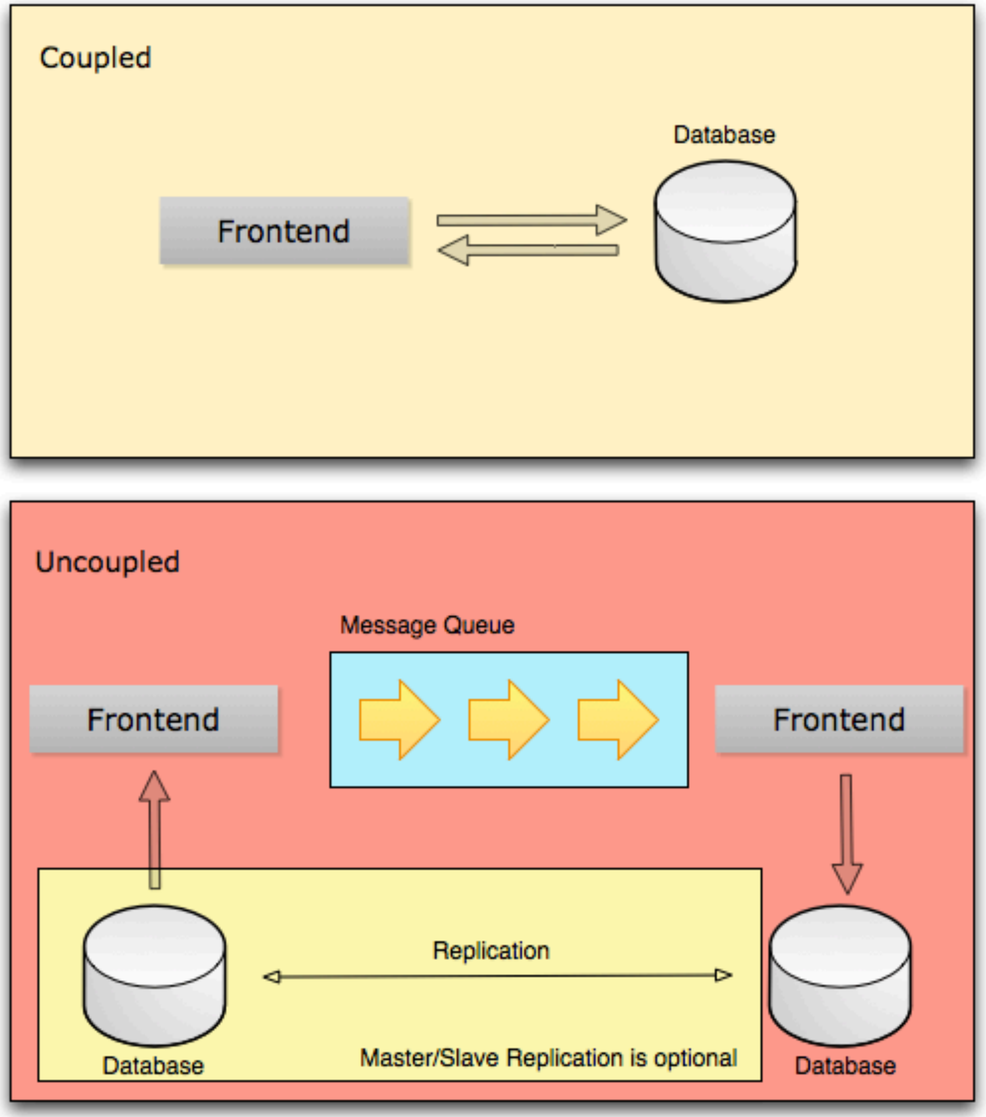
```python
import urllib2

urls = ['http://www.google.com', 'http://www.yahoo.com']
for u in urls: print urllib2.urlopen(u).read()
```

Architectural building blocks

Message Queues – decoupling downloads



Message Queues – decoupling db writes on a webservice



Coupled comments webservice

```ruby
require 'rubygems'
require 'sinatra'
require 'mongo'
require 'json'

include Mongo


DB = Connection.new('localhost').db 'comments'
CC = DB.collection 'oscon11'

get '/' do
  st = CC.find
  st.to_a.to_json
end
```

Receive and write to db

```ruby
post '/comment' do
    body = params[:body]
    dt = Time.now.ctime
    objid = CC.insert "date" => dt, "body" => body
end
```

Message Queue – Comment producer

```
require 'rubygems'
require 'sinatra'
require 'mongo'
require 'json'
require 'net/http'
require 'uri'

include Mongo

DB = Connection.new('localhost').db 'comments'
CC = DB.collection 'oscon11'

get '/' do
  st = CC.find
  st.to_a.to_json
end
```

```
post '/comment' do
    phash = Hash.new
    phash[:body] = params[:body]
    phash[:date] = Time.now.ctime
    res = Net::HTTP.post_form(
                    URI.parse('http://localhost:8888/q/comments'),
                              {'value'=>phash.to_json})
    res.body
end
```

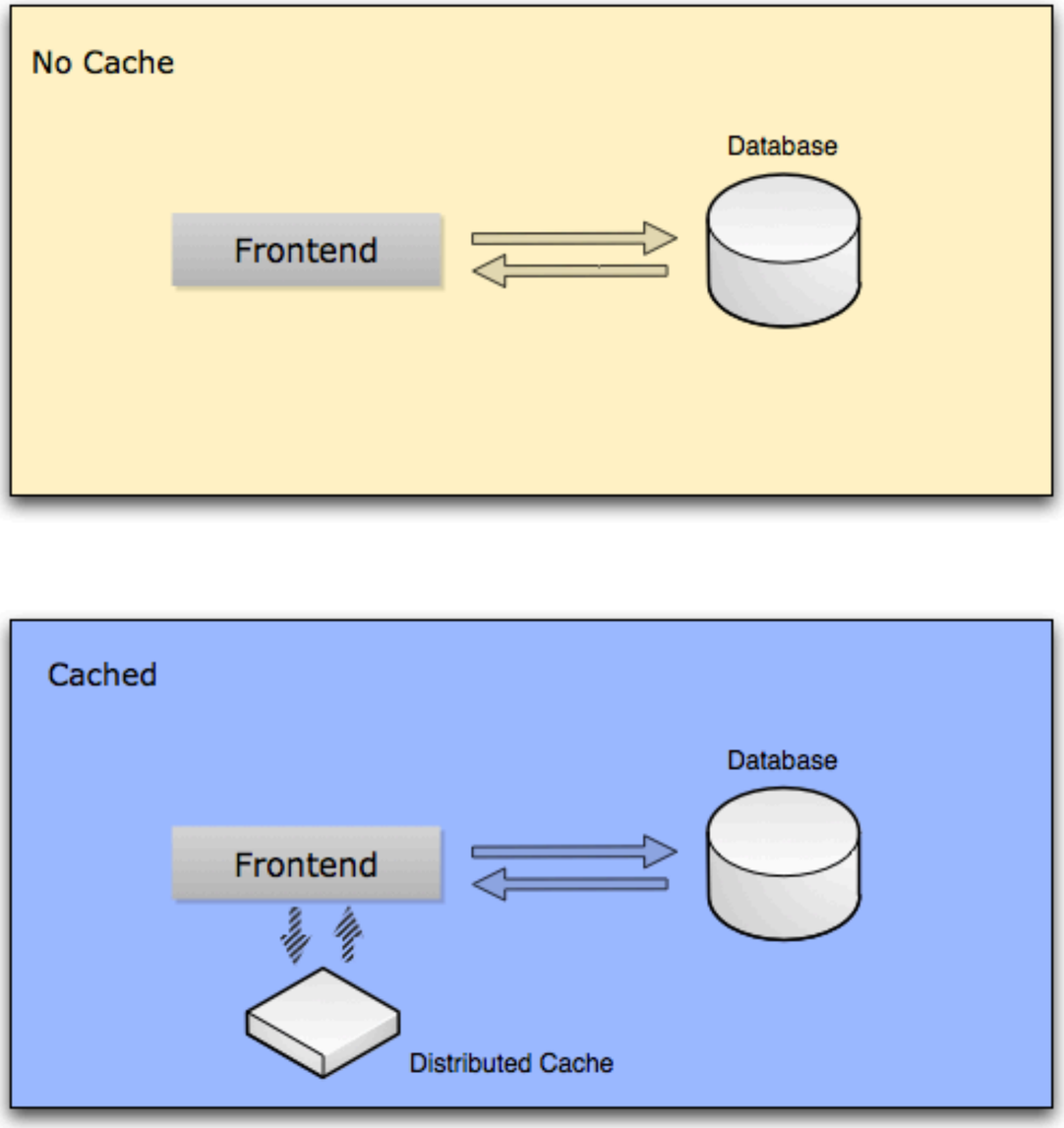
Receive and write to queue, release connection

Message Queue – Comment consumer

```ruby
require 'rubygems'
require 'mongo'
require 'json'
require 'uri'
require 'net/http'

include Mongo

DB = Connection.new('localhost').db 'comments'
CC = DB.collection 'oscon11'

url = URI.parse('http://localhost:8888')

res = Net::HTTP.start(url.host, url.port) do |http|
  http.request_get('/c/comments') do |response|
        response.read_body do |str|
        jp = JSON::parse str
        jv = JSON::parse jp['value']
        objid = CC.insert "date" => jv['date'], "body" => jv['body']
        puts objid
    end
  end
end
```

Take from queue and write to db

Cache


No Cache
Database
Frontend
Cached
Database
Frontend
Distributed Cache

No cache – code from http://github.com/gleicon/vuvuzelr

```
def filter_and_replace(url)
  doc = Nokogiri::HTML open url
  eb = Nokogiri::XML::Node.new 'embed', doc
  eb['src'] = 'http://vuvuzelr.7co.cc/derp.swf'
  eb['width'] = '1'
  eb['height'] = '1'
  eb['wmode'] = 'transparent'
  body = doc.at_css 'body'
  body << eb
  doc.search('img').each do |l| l['src']= 'http://vuvuzelr.7co.cc/vuvuzela.jpg' end
  doc.to_s
end
```

Cache – code from http://github.com/gleicon/vuvuzelr

```
def filter_and_replace(url)
  reddo = Redis.new
  u_hash = Digest::MD5.hexdigest url
  t = reddo.get u_hash
  return t if t != nil
  doc = Nokogiri::HTML open url
  eb = Nokogiri::XML::Node.new 'embed', doc
  eb['src'] = 'http://vuvuzelr.7co.cc/derp.swf'
  eb['width'] = '1'
  eb['height'] = '1'
  eb['wmode'] = 'transparent'
  body = doc.at_css 'body'
  body << eb
  doc.search('img').each do |l| l['src']= 'http://vuvuzelr.7co.cc/vuvuzela.jpg' end
  meh = doc.to_s
  reddo.set u_hash, meh
  reddo.expire u_hash, 180
  meh
end
```

Check if it's on cache
Return if it is

Feed the cache and set expiration

Redis

- Key/Value store
- Key is a string, Value can be string, hash, list, set, scored set
- Atomic operations
- Publish/Subscription channels
- Set/Scored Sets operations (union, diff, intersection)
- UUID Generator, Distributed Cache, Message Queue (RestMQ, Resque), Serialized object storage, throttle control and more at http://rediscookbook.org/

Q & A

Thanks !