Lourens Naudé, Trade2Win

# Embracing Events

Stop polling … react.

RailsWayCon

# Bio

- http://github.com/methodmissing
  - Ruby contractor
  - General misfit for the current economy
  - Ruby, C#, Java, admin 500 nodes, CSS * deferred payment
  - Currently building out a consumer forex platform @ Trade2Win Slippage matter.
- creator
  - scrooge, mysqlplus_adapter, mri_instrumentation
- contributor
  - Rails, mysqlplus, query_memcached, thinking-sphinx, em-spec

RailsWayCon

# e-vent [i-vent]

something that happens or is regarded as happening; an occurrence, esp. one of some importance.

# A change in state …

- MySQL server sends results down the wire
  - Connection file descriptor becomes readable
    - Process results
    - Render results
- Ignorance is bliss
  - Never be flooded
  - Only handle events that matter

# Talk#to_a

- Context switches
- Scheduling
- Programming models
- Frameworks
  - Eventmachine
  - Neverblock
- Operating System
- Lessons from massively multiplayer online games
- Testing
- Q & A ?

# Getting Fibers (coroutines) and Threads

# Scenario we all can relate to ...

- Quiet weekend day, erratic thoughts from the week before comes together
- Sharing space with your better half ... who's being chatty
- You're being interrupted at random
- Let's assume an average "in the zone" startup threshold of 2 minutes per interruption
- Deadlock's not an option ...

# Multithreading

- Let the initial pattern of thought and the interruption be distinct Threads
- The interruption's effective right away
- The context switch is 'expensive' : coding VS whichever chore's been neglected
- Preemptive scheduling == constant interruption

# Fiber / coroutine pool

- Two distinct Fibered contexts
- Able to defer the interruption
- Cheap switching : being thick skinned buys time
- You#resume the interruption if and when there's time to handle it

# Thread

- Scheduling
  - Handled by the VM – timesliced @ 10ms by MRI
  - Executes as soon as CPU resources become available
- Availability
  - MRI 18: Maps to a single native thread
  - MRI 19: 1 x native thread for each Ruby thread - GIL
  - JRuby: 1 x Java thread for each Ruby thread
- Overheads
  - Large initial stack size
  - MRI uses a SIGVTALRM to signal a context switch

# Fiber

- Scheduling
  - No concurrency – scheduled by application code
  - Never executes right away
- Availability
  - MRI 18: Patch by Aman Gupta && Joe Damato
  - MRI 19: Core Fiber class
  - JRuby: Poor Man's Fibers
- Overheads
  - Small initial stack size @ 4k
  - Context switches exclusively in user space – very fast

# Why Fibers matter

- Computes a partial result - generator
- Yields back to the caller
- Saves execution state
- Caller resumes
- Data exchange
  - Accept arguments through Fiber#resume
  - Return values through Fiber.yield

# Threads as Coroutines

- Through a voodo combination of
  - Thread priorities
  - Thread.pass
  - Thread.stop
  - Thread#run (yields CPU) or Thread#wakeup (no CPU)
- Poor Man's Fibers piggy backs off a Queue
- MRI 18 Fibers = Threads – preemptive scheduling, much like continuations, but with a smaller stack.
- Performant, but not as lightweight as Fibers

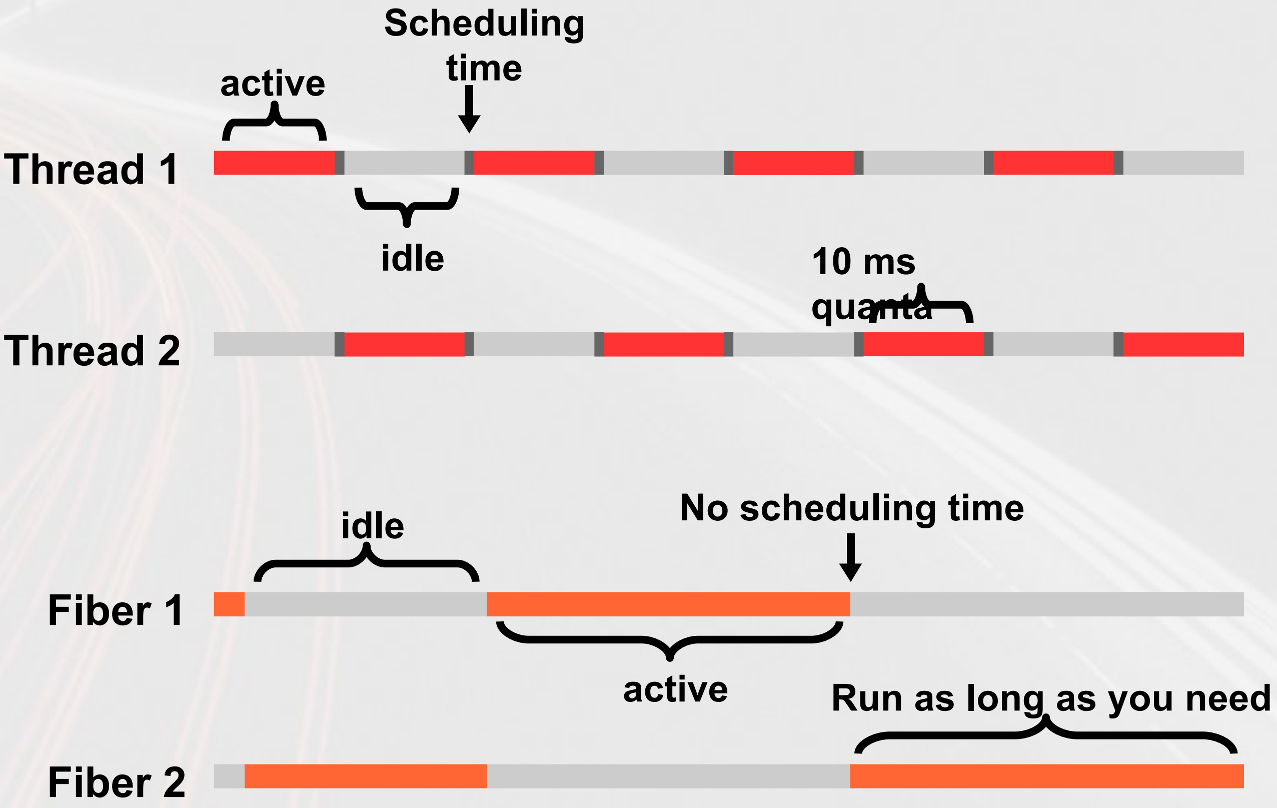
# Cooperative scheduling

- Request / response with evented DB access
  - HTTP request wrapped in a Fiber
  - => mysql->fd : SELECT * FROM users WHERE id = 1
  - Execution paused
  - <= mysql->fd : #<User id: 1 uname: methodmissing>
  - Execution resumed
  - => mysql->fd: SELECT * FROM wishlist WHERE user_id = 1
  - Execution paused
  - <= mysql->fd: #<Wishlist id: 20 user_id 1>

# Cooperative VS preemptive

- Fibered dispatch, query = *, result = +, int = &
  - <GET /orders>          *    + *+*+*+*+*+*+
  - <POST /checkout>     * + * +  *+*+
  - <DELETE /sessions>   *+

- Threaded dispatch, interupted every 10ms
  - <GET /orders>          *  & + *+*+&*+*+*&+*+
  - <POST /checkout>     * + * +&  *+*&+
  - <DELETE /sessions>   &*+



Cooperative VS Preemptive

RailsWayCon

# Event-driven Programming

- Program flow determined by events
- Characterized by an event handler, main loop or reactor
- The reactor typically 'owns' the process
- loop { button.on(:click).dispatch }
- Binds event handlers to events
- Deep nesting of callbacks and constructs

  eg. Twisted for Python

# Batch programming

- wysiwig
- Flow determined by the developer
- Read → process → output

Event-driven execution, synchronous API.

# Database ...

```
class FiberedMysqlConnection
  def query( sql )
     send_query sql
     Fiber.yield
     get_result
  end
end

EM.run{ conn.query( 'SELECT SLEEP 1' ) }
```

# … app server

```
class UsersController < AC::Base
  def index
      @users = User.paginate( :page => 1 )
   end
end

Thin::Server.start('0.0.0.0', 3000) do

  Fiber.new{ AC::Dispatcher.call( env ) }

end
```

# Frameworks

# Eventmachine

- Implementation of the Reactor pattern
- C++ with Ruby Bindings
- JRuby compatible through jeventmachine
- High performance event driven IO : epoll on Linux, kqueue on BSD and derivatives

# The Reactor

- Maintains a global current loop time
- Quantum is just under 100ms … adjustable
- Verifies descriptors through a heartbeat
- Runs timer events
- New descriptors dropped onto a different queue
- Modified descriptors processed outside loop
- Stopped with special Loop Break Descriptor

# Bindings and Callbacks

- A portable signature registry / list, implemented as UUID identifiers
- Callback triggers :
  - timer fired
  - connection read
  - connection completed
  - connection accepted
  - connection unbound

# Eventable File Descriptors

- Nonblocking
- Notify of read and writability
- Closed in 3 ways
  - hard
  - immediate ( when possible, next tick )
  - after writing ( protocol handlers, serve HTTP page, close connection

# Deferrables

- A callback ... specifically a callback for :success or :failure
- Immediately pushed to the background and scheduled for future execution
- Deferrables should have an associated timeout

# Timers

- Periodic or oneshot
- Set a max timer count to avoid flooding the reactor
- Can be cancelled

# Files

- Very fast transfer of < 32k files with EM#send_file_data
- File watch API
  - Notified when modified, deleted or moved
- fastfilereader extension
  - Memory <=> disk transfer with mmap

# Processes

- Supports deferrable child processes through pipes
- EM#system
- Process watch API
  - Notified when forked or exits

# Client / Server

```
module Challenge
  def post_init
    send_data 'psw?'
  end

  def receive_data( data )
    ( @data ||= '' ) << data
  end
end
EM.run{ EM.connect 'localhost', 80, Challenge }
```

# Observing the Reactor

( Interactive dtrace session )

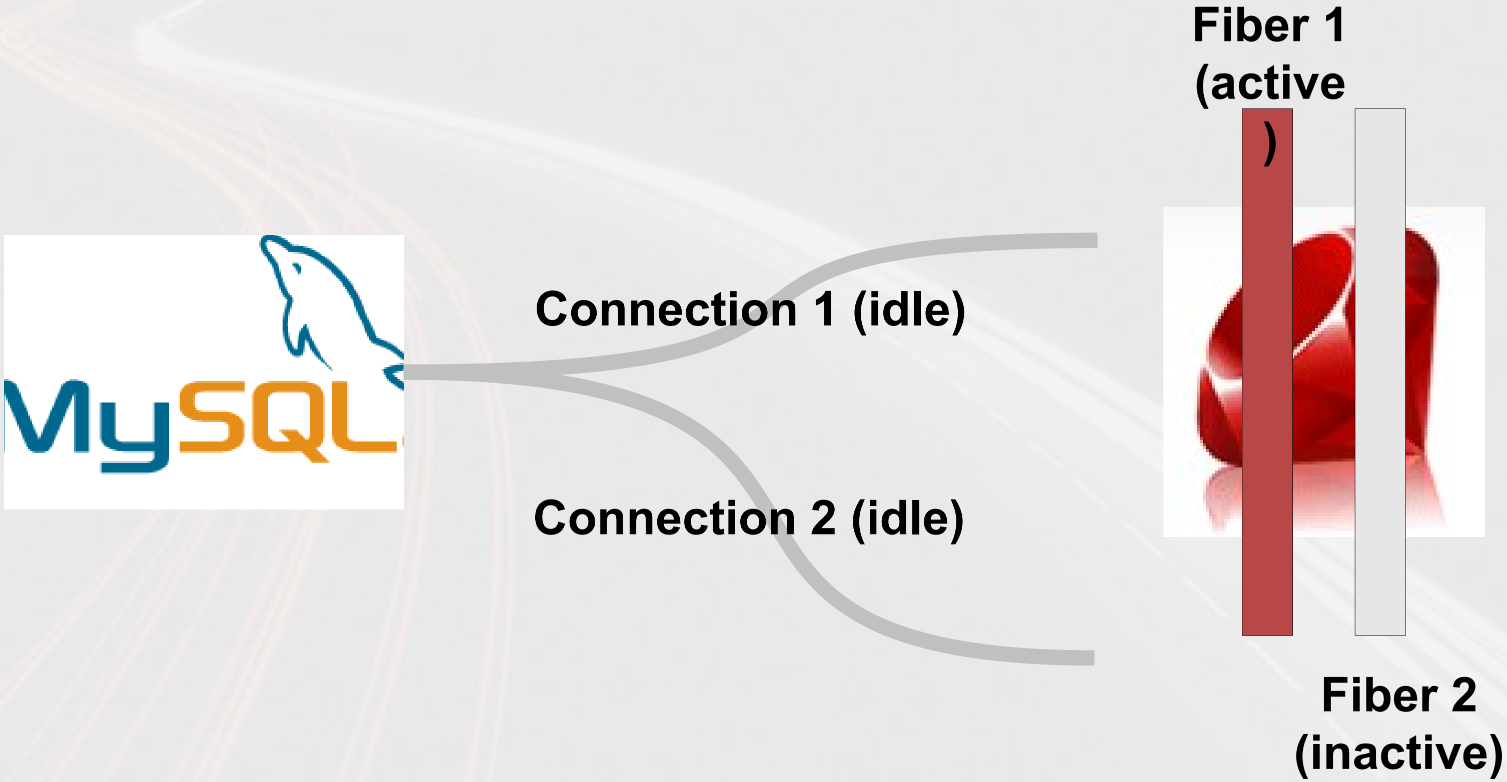
# Neverblock

# Why ?

- Most web apps block on IO
- Ruby processes has 'high' memory overheads
- 60MB RSS blocking 30ms on IO == $$$
- Allocated in slabs – 80MB -> 120MB
- Significantly reduces capacity for concurrent requests
- eSpace stepped in

# How ?

- Fibered connection pool – typically 8 to 12
- Attaches to either EM or Rev as main loop
- Patches for Evented Mongrel && Thin to wrap requests in Fibers
- Fibered DB connections for postgres && mysql ( with mysqlplus )
- Respects transactions - BEGIN ... COMMIT required to be on the same connection



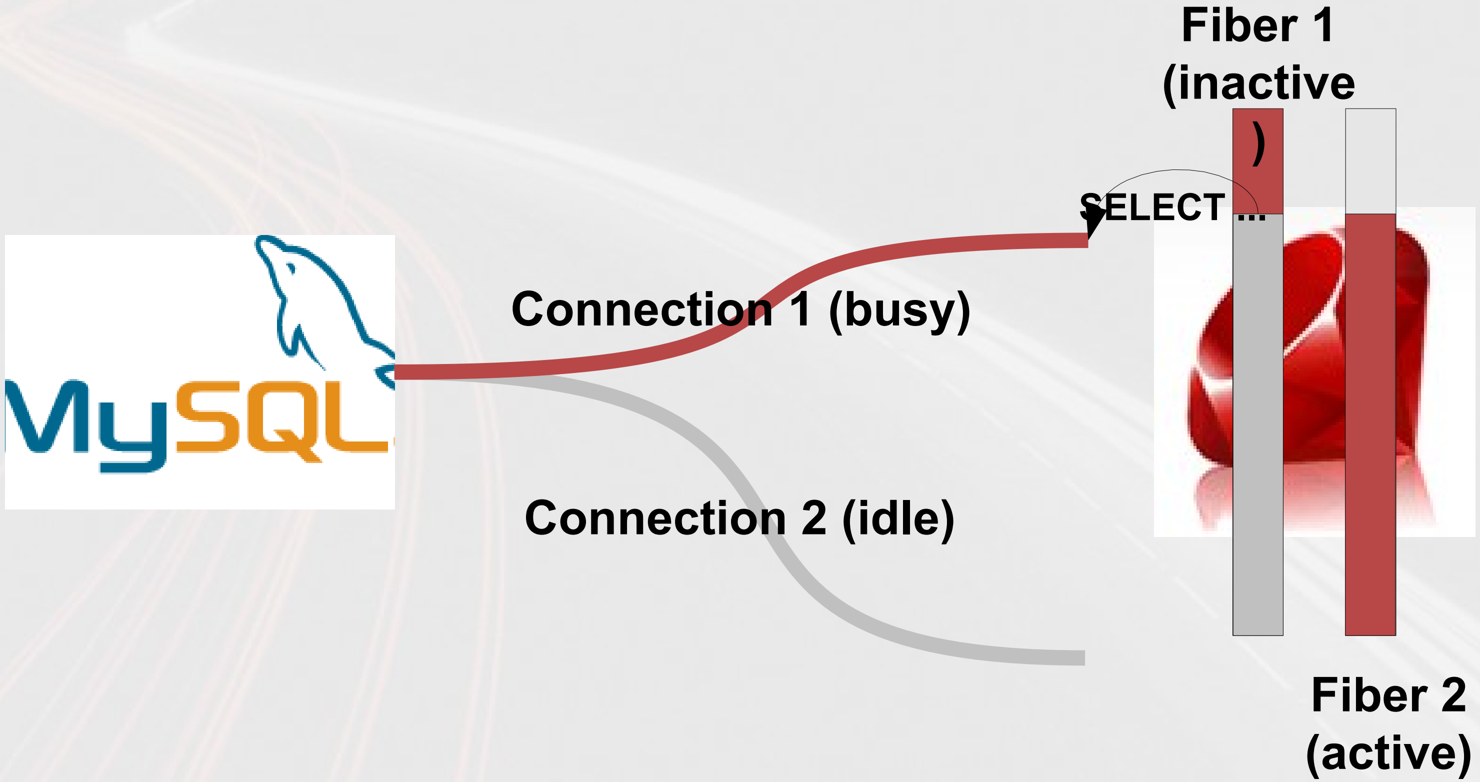
Fiber 1<br>(active<br>)

Connection 1 (idle)

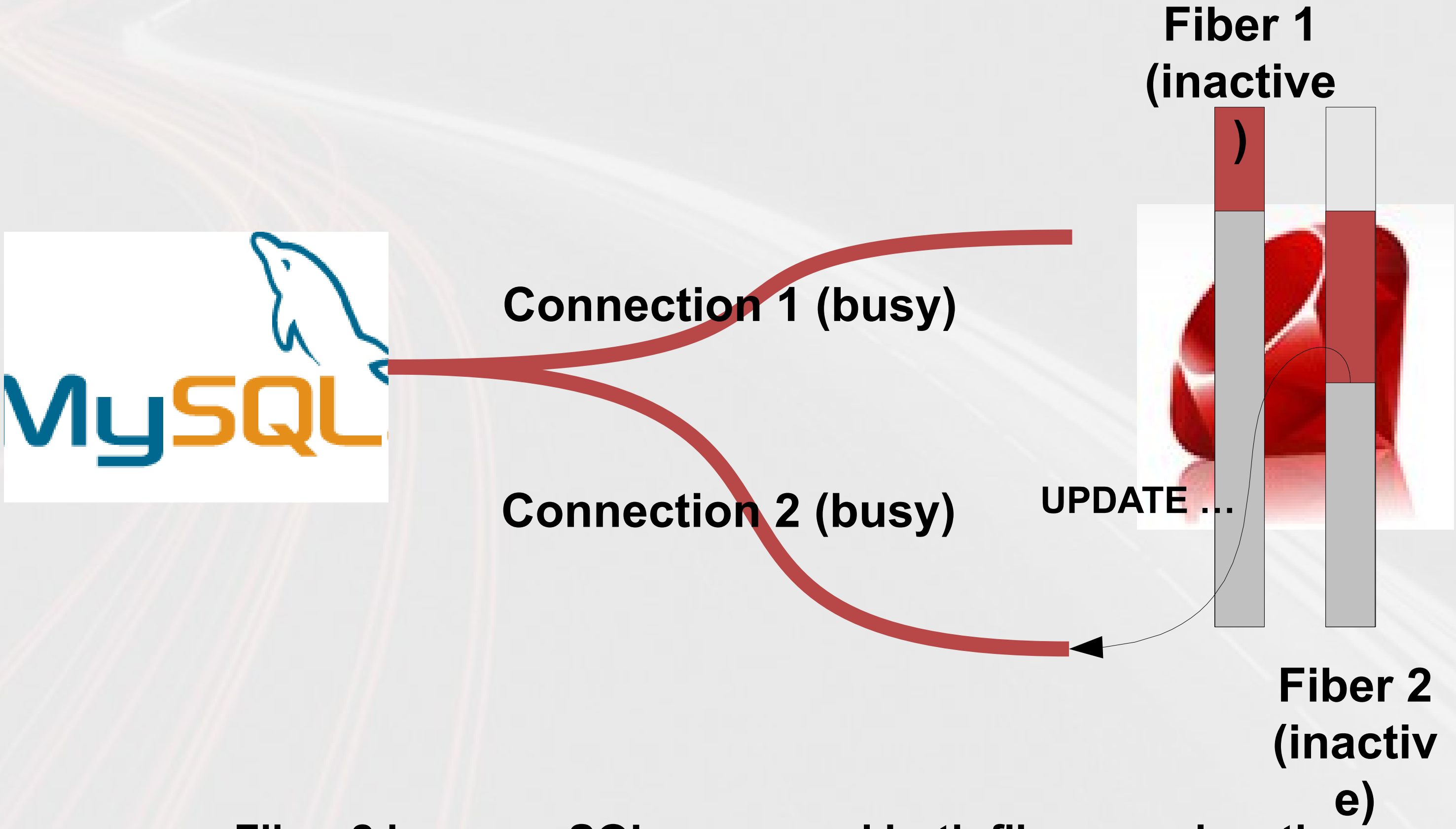
Connection 2 (idle)

Fiber 2<br>(inactive)

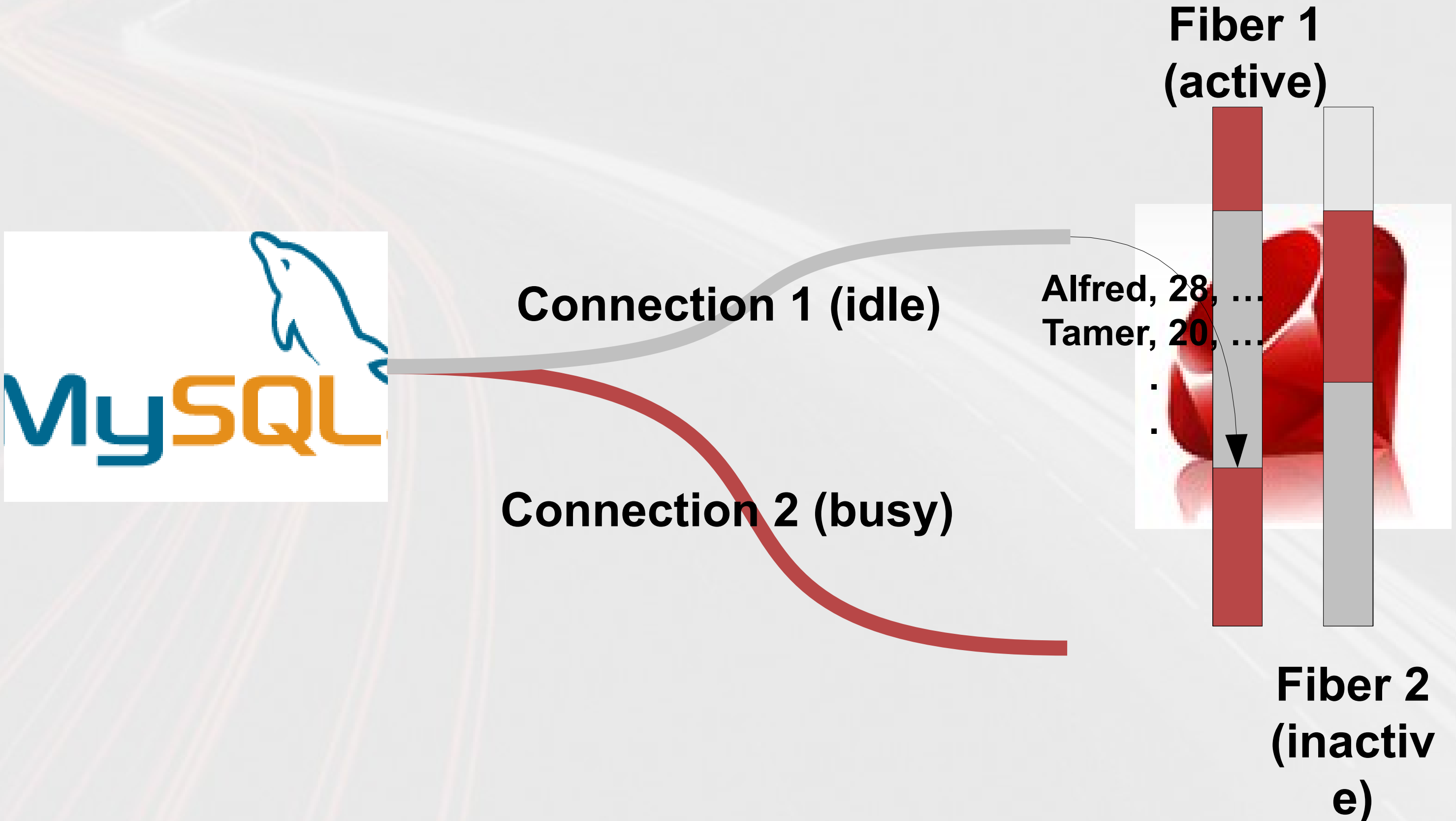
Fiber 1 is executing



Fiber 1 issues a SQL query and passes control to Fiber 2



Fiber 2 issues a SQL query and both fibers are inactive



Fiber 1 (active)

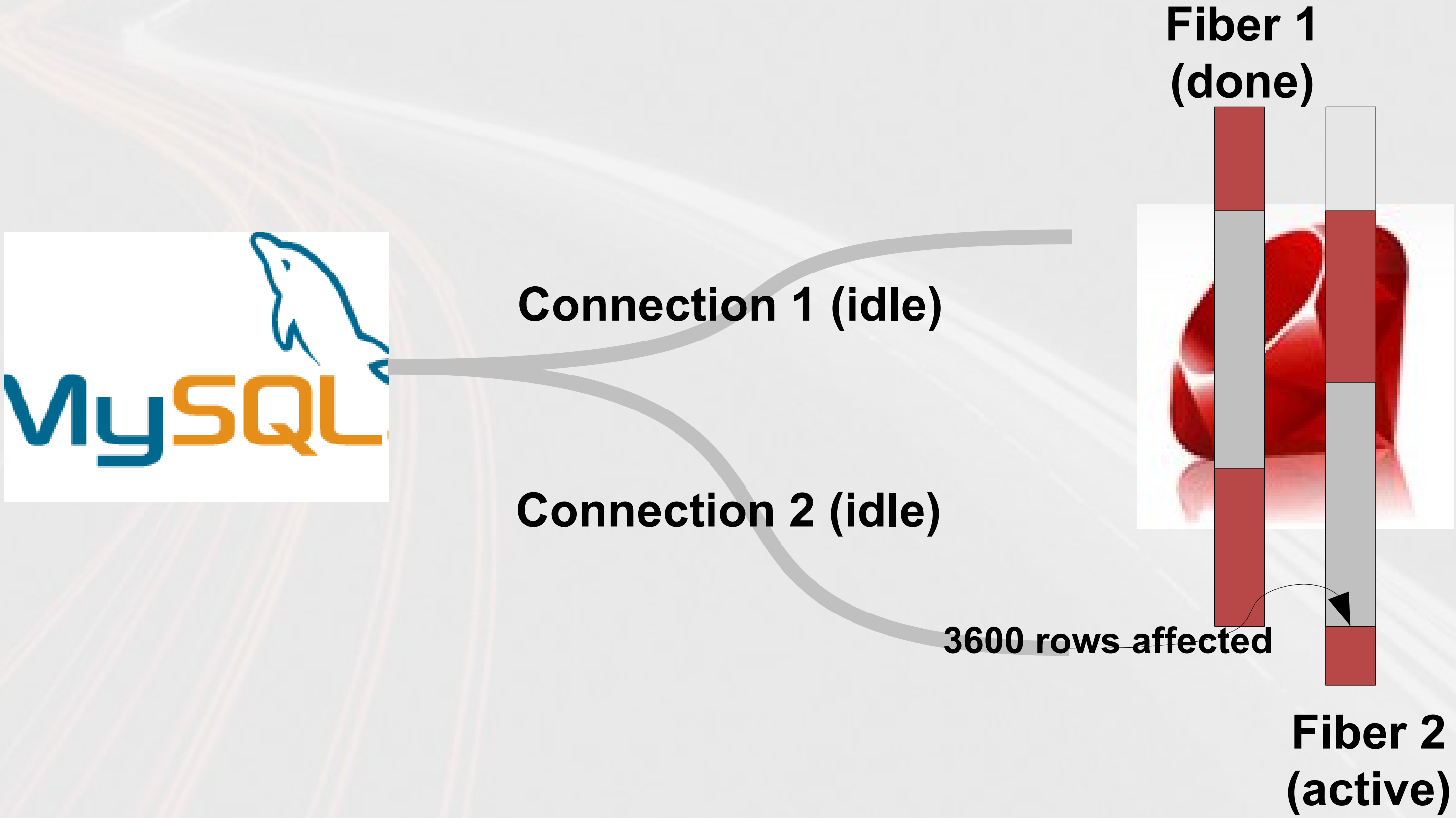
Connection 1 (idle)

Alfred, 28, …
Tamer, 20, …
.
.

Connection 2 (busy)

Fiber 2 (inactive)

Reactor notices first connection is finished so it resumes Fiber 1



Fiber 1 is done and reactor notices second connection is finished so it resumes Fiber 2

# Patches to Ruby IO

- Clever redirection to non-blocking methods
  - IO#read* => IO#read_nonblock
  - IO#write* => IO#write_nonblock
- Catches IO errors
  - Errno::EWOULDBLOCK
  - Errno::EAGAIN
  - Errno::EINTR
- Rescue attaches to the event loop
  - Requests notification of read / writability
  - Detach on success

# Operating System Events

# Signals

- Software interrupts
- Portable
  - Signal.list supported on all host systems
- Unobtrusive
  - Caught Signal.trap( 'IO' ){ puts 'readable' }
  - Sent with Process.kill( 'IO', $ )
- Can be scoped to a single process or a process group
- Daemons: SIGHUP && SIGTERM

# Handling Signals

- Ignore them – except KILL && STOP
- Catch with trap('IO'){}
- Let the default action apply
- SIGVTALRM is reserved for threading
- Signals the process every 10ms
- Flags that it's time to schedule
- Green threading maps to 1 OS thread, 1 per process, signals scoped per process

# Processes

- Process groups
  - Leader ? Process#pid == process group identifier
  - Group persist when leader exists as long as there's some members

# Multi-process Quality of Service

- Signals to enforce QOS
  - URG: Slow down
  - CONT: Resume normal operation
  - XCPU: Crawl
- Monitoring agent sends URG during load spike
- Sends CONT when load drops
- XCPU during extremely high load, effectively pausing the worker(s)

# POSIX Realtime Extensions

- Well supported, but implementations differ
  - Linux Kernel 2.6
  - FreeBSD / Darwin
  - Solaris
- Kernel level asynchronous IO
- Supports thread, signal based or no-op callbacks
- Reduces syscall overheads in IO bound applications

# How it works

- Control block
  - Struct: fd, buffer, notification, offset, number of bytes
- aio_read( '/tmp/file' ) # signal notification with SIGIO
- Returns right away, faster than O_NONBLOCK
- Kernel signals with SIGIO on completion
  - Handler notification contains pointer to the the original control block

# Why it matters

- Parallel execution of aio_read && aio_write
  - lio_listio( LIO_WRITE, list_of_cbs, 90, &list_sig )
- A single user to kernel space switch
- 90 operations, a single signal notification
- Experimental rb_aio extension
- AIO.write( { :filename => 'buffer' } )

# Project Darkstar

http://www.projectdarkstar.com

# Gaming @ a Rails conf ?

- Rails Rack compatibility
- Middleware + ESI
- SOA architecture with thin processes + layers
- Russian doll pattern

# Architecture Overview

- Communications
  - Pubsub && direct channels suported
- Execution Kernel
  - Stateless execution of tasks
- Data storage
  - Transactional storage of serialized objects

# Task Execution

- Spawned and managed by a task manager
- Appears monothreaded, executes in parallel
- Immediate, delayed or periodic
- Parent <=> child relationships
- Guaranteed to execute in order
- Children inherits priority of parent

# Task Lifetime

- Max execution time is 100ms - configurable
- Split long running tasks into multiple smaller tasks

# Managed Objects && References

- Managed object == a persitable entity
- References
  - Encapsulates all interactions with the data store
- Retrieval strategies
  - Get with the intention of modifying state
  - Get for read
  - Object#mark_for_update to promote a read only instance

# Listeners

- Application listener
  - Initialize: setup any game play items when first booted
  - Login callback: setup per player game state
- Client listener
  - Defines methods to repond to client events eg. logout

# Testing

RailsWayCon

# Test strategies

- Define environments : test / production
- Ability to switch between an evented and batched programming model depending on the environment
- A desirable side effect of loose coupling
- Very well suited for integration testing

# Block in testing mode

- Inject correlation middleware when in the test environment
- extend BlockInTest
- A single event may spawn *data
  - @broker.collateral_report( 'ACCXXXX' )
  - => [#<Report:0x202948>, #<Report:0x202234>]
- A single event may spawn many others
  - @broker.collateral_report( 'ACCXXXX' )

    => [:acknowledge_reports, :reports]

# Event => expectation

- Mapping Event X to Reaction *Y : Event IDXXXXXXX => { IDXXXXXXX => [Y,Y,Y] }
- Mapping Event X to Data Elements *Z : Event IDXXXXXXX => { IDXXXXXXX => [Z,Z,Z] }
- Backbone of integration tests

# Existing specs ?

- github.com/tmm1/em-spec
- Bacon and Rspec backends
- A reactor / event loop for context
- How ?
  - Context executes in a Fiber, each expectation yields
- Unobtrusive
  - describe 'This is a context'
  - Em.spec 'This is an evented context

# Questions ?

# Slides + Code

http://github.com/methodmissing/railswaycon_events

# Thanks! Fork away.