

Ruby & XMPP

- What is XMPP?
- Uses
- Why you care
- Ruby Libraries
- Basic examples
- Advanced example
- ActionMessenger
- Resources

What is XMPP?

"The Extensible Messaging and Presence Protocol (XMPP) is an open technology for real-time communication, which powers a wide range of applications including instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication, and generalized routing of XML data." - xmpp.org/about

What is XMPP?

- Open technology
- Defines protocols for real-time communication
- Allows you to send small pieces of XML between entities
- Invented by Jeremie Miller in 1998
- First server released Jan 1999

What can you do?

XMPP services

- Authentication
- Presence
- Contact lists
- One to one messaging
- Many to many messaging
- Notifications
- Service discovery

What can you do?

XMPP applications

- Sending notifications (replacement for email)
- Instant messaging
- Group chat
- Gaming (chesspark.com)
- Systems control (<http://github.com/engineyard/vertebra>)
- VoIP (Google Talk)

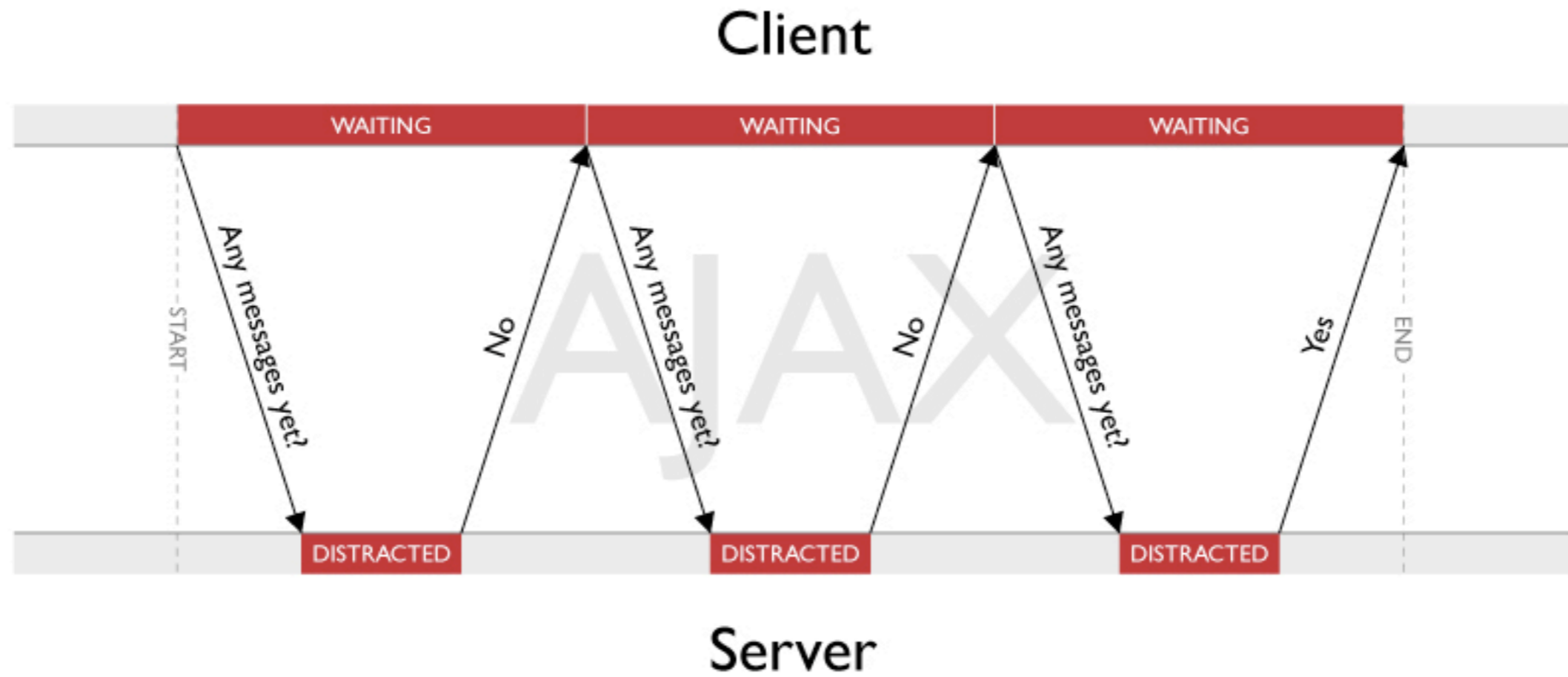
Why you care

- Extensible
- Built for handling large numbers of connections
- Scalable
- Secure
- Real-time

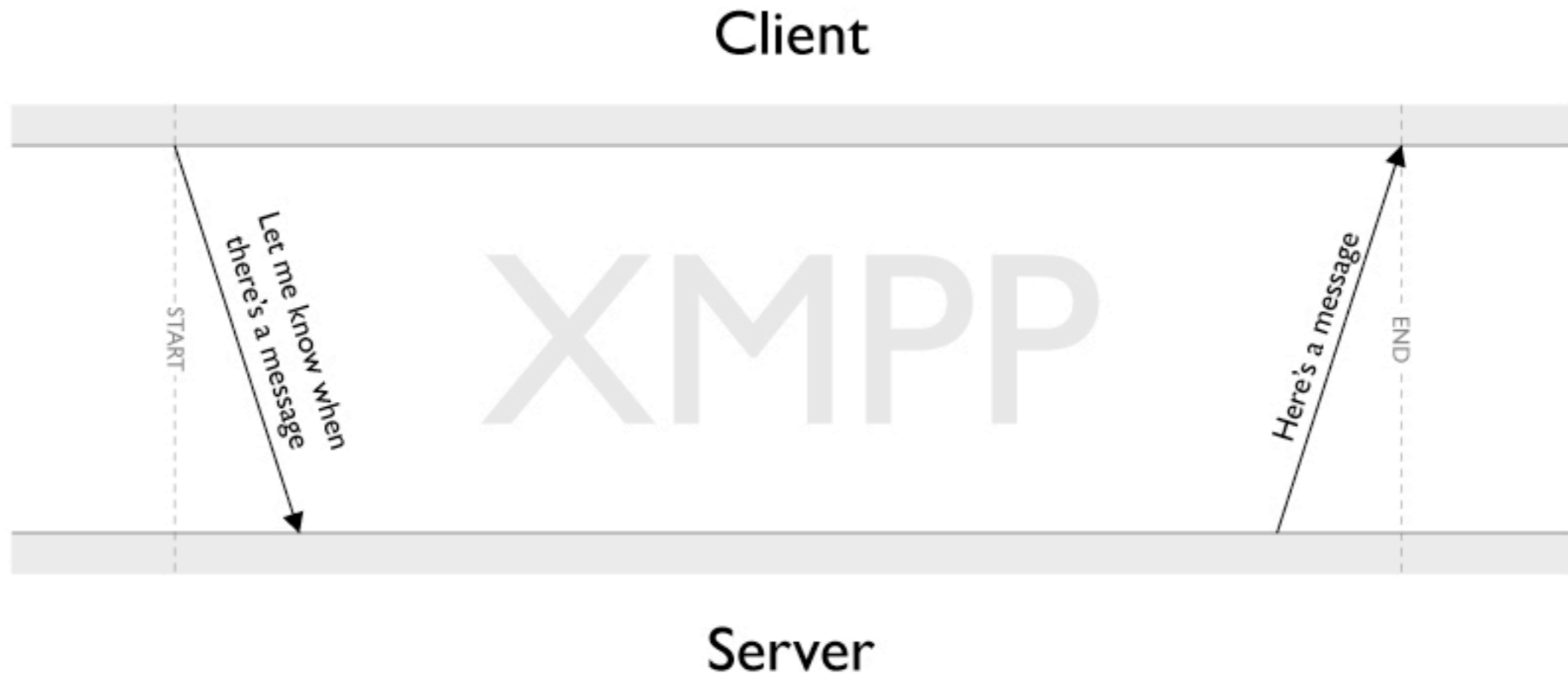
HTTP Vs XMPP

- HTTP is great!
- XMPP is better in some situations...
- The 'social web' has brought problems that are hard to solve via HTTP
- XMPP can often be used for such cases

Why you care



Why you care



From ruby

- XMPP4r
 - XMPP4r-simple

<http://home.gna.org/xmpp4r/>

<http://xmpp4r-simple.rubyforge.org/>

XMPP4r-simple

- Built on top of XMPP4r
- Makes it very easy
- Only exposes common tasks
- Use XMPP4r for anything advanced

XMPP4r-simple

```
require 'rubygems'  
require 'xmpp4r-simple'
```

```
im = Jabber::Simple.new("user@example.com", "password")  
im.deliver("friend@example.com", "Hey there friend!")
```

XMPP4r-simple

```
im.presence_updates  
#Array of presence updates
```

```
im.received_messages { |m| puts m.body }  
#Outputs body of messages
```

```
im.roster  
#Returns your roster
```

```
im.status(:dnd, 'speaking at lrug')  
#Set status
```

XMPP4r - MUC example

Joining the room

```
require 'rubygems'  
require 'xmpp4r'
```

```
@jid = Jabber::JID.new('thomas@localhost')  
@client = Jabber::Client.new(@jid)  
@client.connect  
@client.auth('password')
```

XMPP4r - MUC example

Joining the room

```
require 'xmpp4r/muc'
```

```
@room = Jabber::MUC::MUCClient.new(@client)
```

```
@room.join(Jabber::JID.new('chat@conference.localhost/' +  
@client.jid.node))
```

XMPP4r - MUC example

Interacting with the room

```
@room.add_message_callback do |m|  
  puts m.body  
end
```

```
@room.add_join_callback do |m|  
  @room.send(Jabber::Message.new(m.to, "Hello  
#{m.from}!"))  
end
```

ActionMessenger

- <http://trypticon.org/software/actionmessenger/>
- Like ActionMailer but for im
- Can send & receive
- Simplifies testing message sending
- YAML config, similar to database.yml
- Possibly not maintained??
- Also: http://jamesgolick.com/action_messenger (http://github.com/giraffesoft/action_messenger)
- Both reconnect every time you send...

ActionMessenger

```
class UserMessenger < ActionMessenger::Base
  def ping(user)
    recipients user
    body 'The website is down!'
  end
end
```

#In controller

```
UserMessenger.send_ping('example@example.com')#Send message
```

Things to watch

- Unwise to use XMPP4r directly from your controllers
- Use Drb or similar
- Connection is fairly slow so connect once on app initialization
- Could queue messages if large amounts
- Components are often better for bots
- Watch roster sizes
- Watch data size

More info/resources

- XMPP homepage - <http://xmpp.org>
- Peepcode screencast - <http://peepcode.com/products/xmpp>
- XMPP book - <http://oreilly.com/catalog/9780596157197/>
- XMPP4r - <http://home.gna.org/xmpp4r/>
- XMPP4r-simple - <http://xmpp4r-simple.rubyforge.org/>
- ActionMessenger - <http://trypticon.org/software/actionmessenger/>
- Vertebra - <http://github.com/engineyard/vertebra/tree/master>