INFINUM

Action Cable

NIKA JUKIĆ

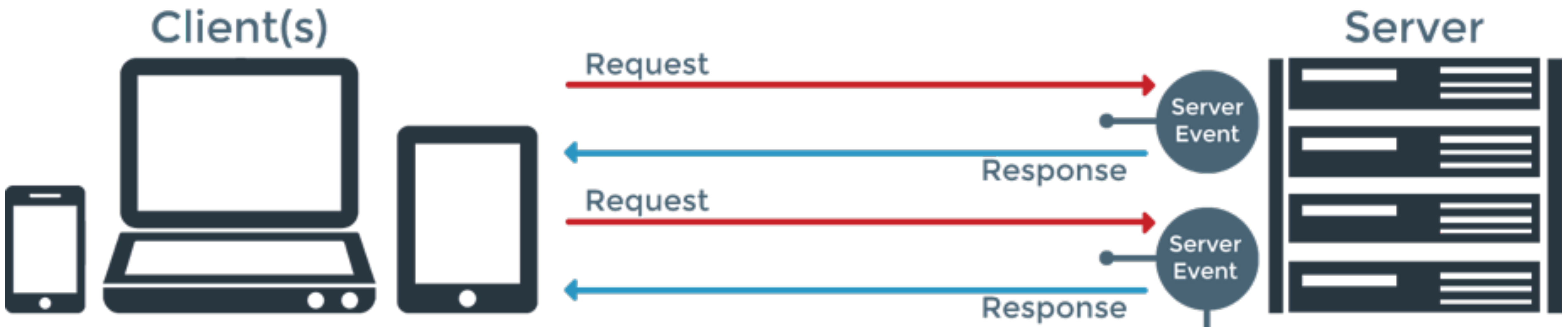
REAL-TIME WEB

System in which users receive new information from the server as soon as it is available

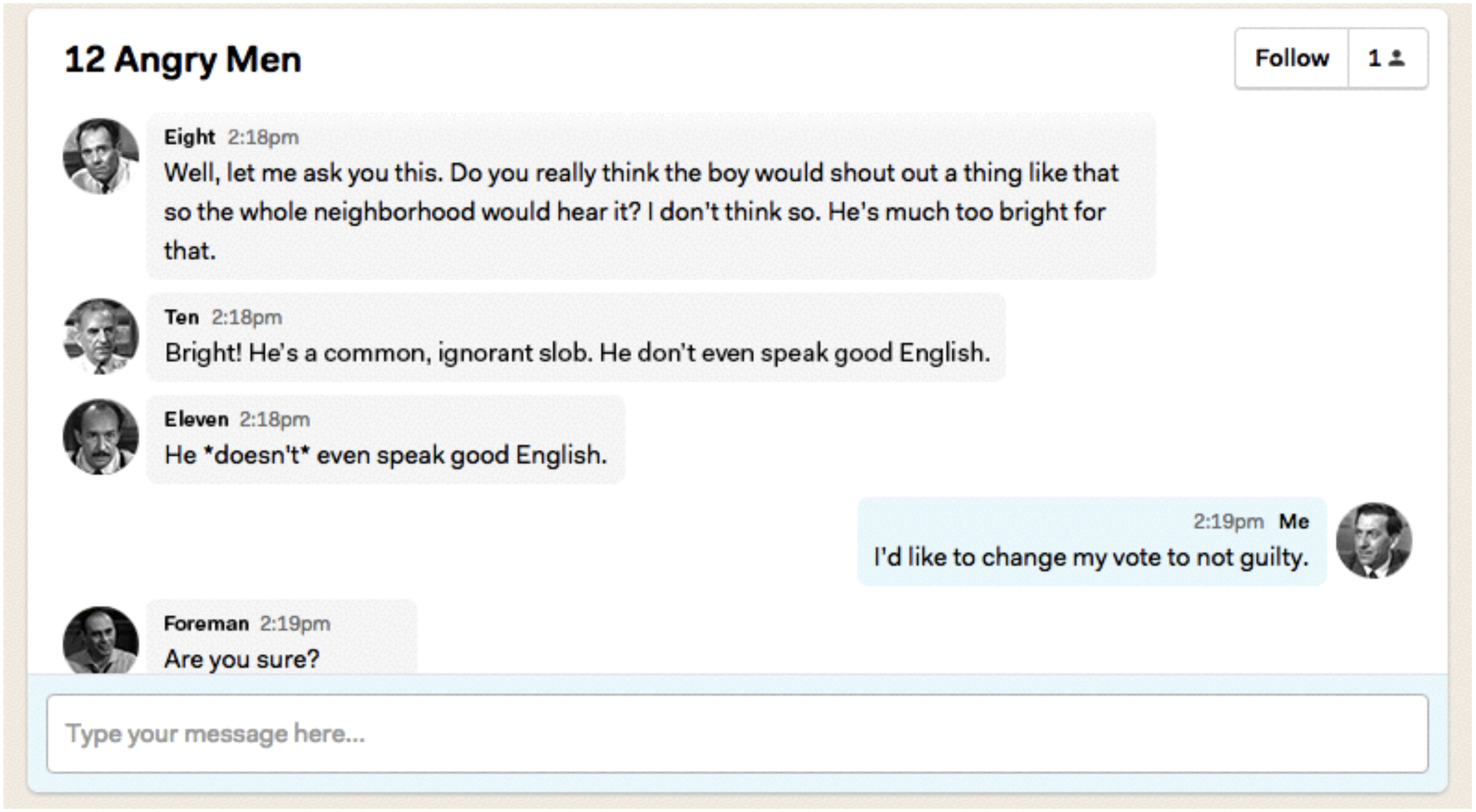
No request required!

“SOLUTIONS”

- polling and long polling
- server load
- scalability?



BASECAMP – CAMPFIRE



12 Angry Men

Eight 2:18pm
Well, let me ask you this. Do you really think the boy would shout out a thing like that so the whole neighborhood would hear it? I don't think so. He's much too bright for that.

Ten 2:18pm
Bright! He's a common, ignorant slob. He don't even speak good English.

Eleven 2:18pm
He *doesn't* even speak good English.

2:19pm **Me**
I'd like to change my vote to not guilty.

Foreman 2:19pm
Are you sure?

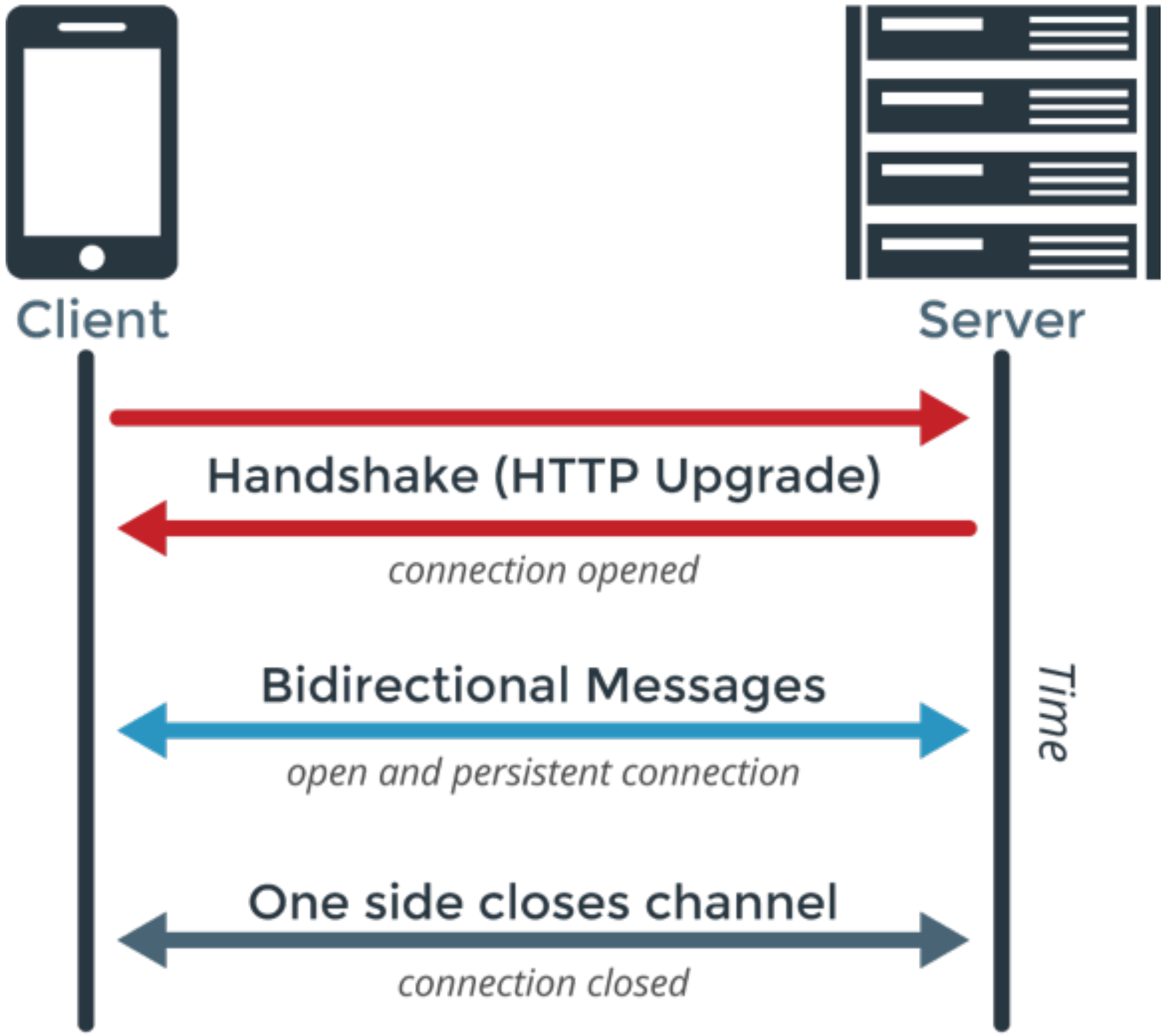
If you can make WebSockets even less work than polling, why wouldn't you do it?

– DHH

WHAT ARE WEBSOCKETS?

- communications protocol
- bidirectional
- full-duplex
- single TCP connection

WEBSOCKET PROTOCOL



HANDSHAKE – HTTP UPGRADE

```
GET /cable HTTP/1.1
Host: localhost:3000
Upgrade: websocket
Connection: Upgrade
```

WHAT IS ACTION CABLE?

Framework for real–time communication over websockets

REAL-TIME COMMUNICATION

- Chat
- Notifications

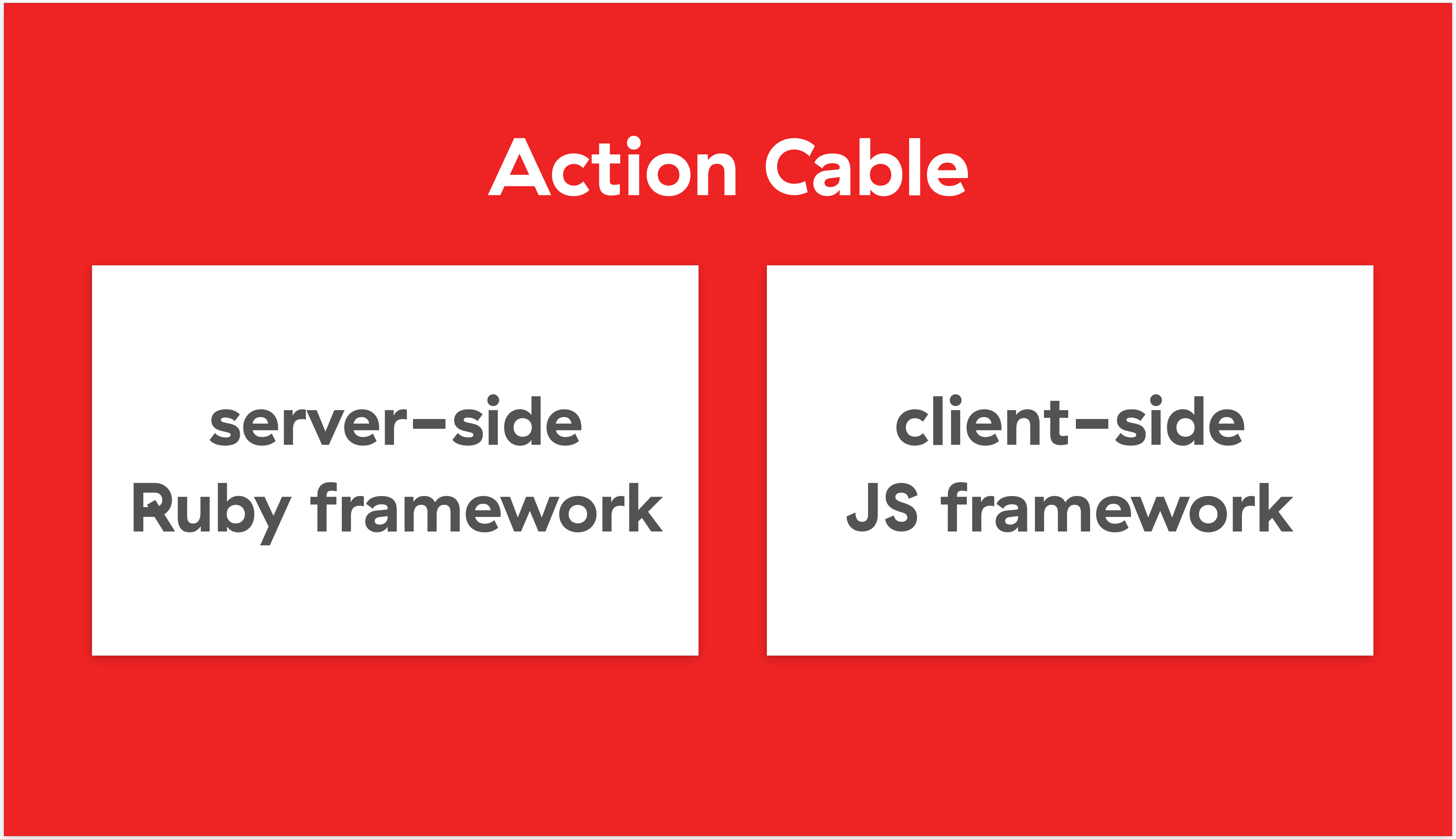
- NO MORE POLLING!

ABOUT ACTION CABLE

- Rails 5
- real-time features written in Ruby

- layer on top of Rails architecture
- ActiveRecord access

FULL-STACK FRAMEWORK



ACTION CABLE SERVER

- stand-alone server
- process withing the main application server

ACTION CABLE SERVER

- Rack socket hijacking API
- multithreaded pattern

- Unicorn, Puma, Passenger

01

TERMINOLOGY

CONNECTION

- Foundation of client–server relationship

- Action Cable server handle multiple connection instances

One connection per WebSocket connection

=

One connection per tab／window／device

CONSUMER

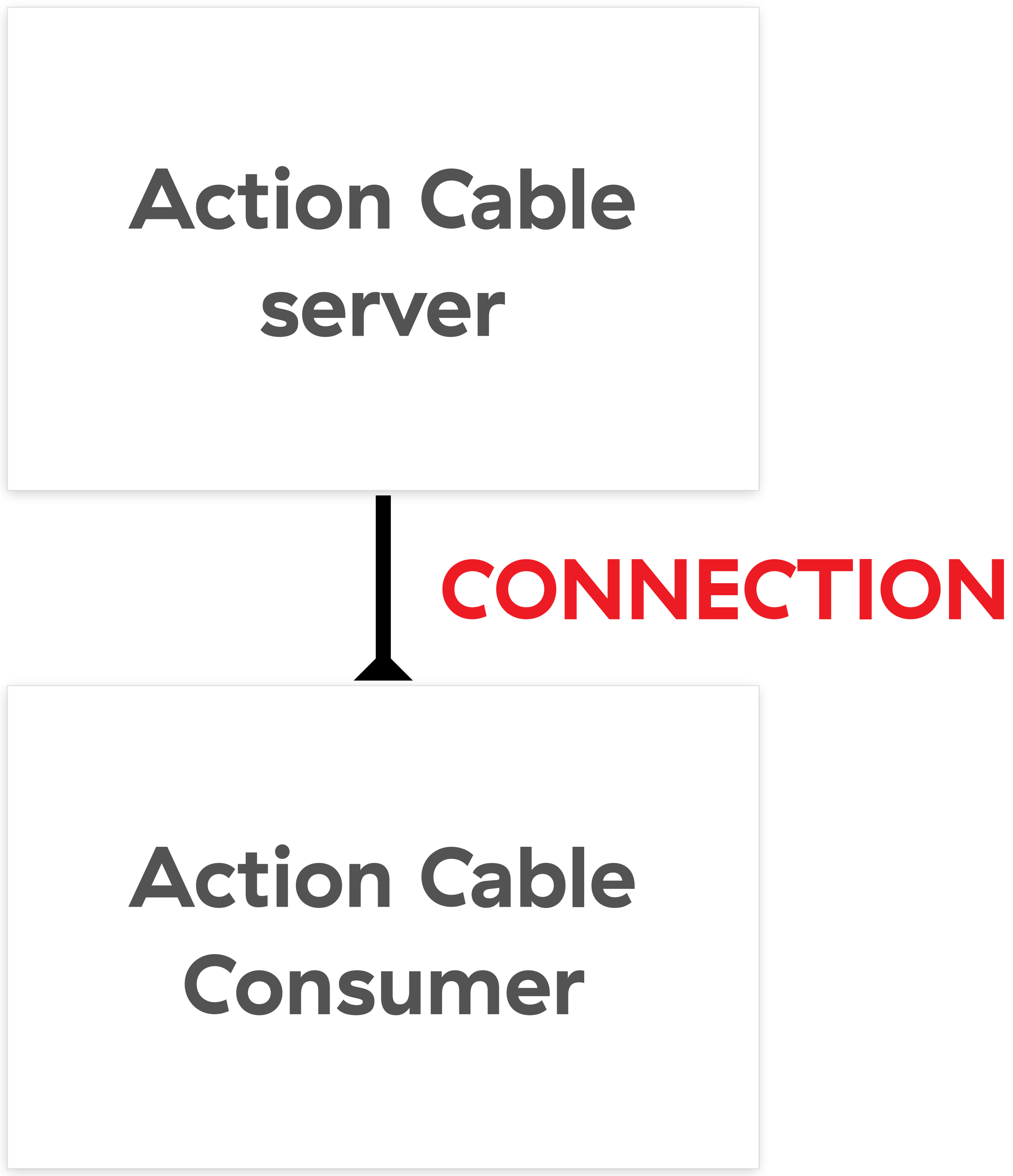
- Client of WebSocket connection
- One consumer–connection pair per tab/window/device

CHANNEL

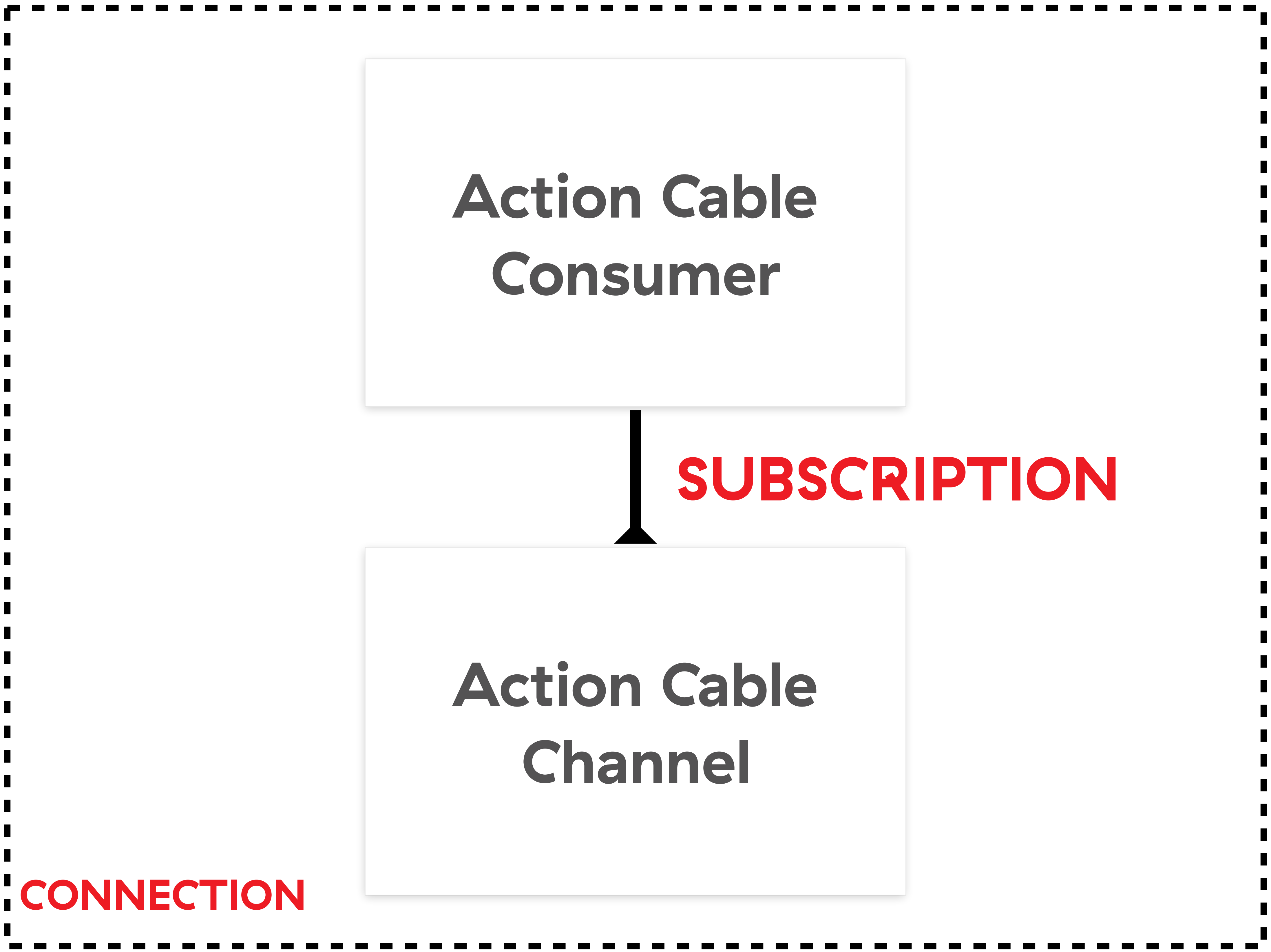
- Logical unit of work
- Similar to MVC controller

- Has many subscribers

CONNECTION – CONSUMER



SUBSCRIPTIONS



PUB/SUB

- message queuing paradigm
- publishers → data → subscribers
- broadcasting

02

CONFIGURATION

MOUNTING THE SERVER

config/environment.rb

```
config.action_cable.mount_path = '/websocket'
```

OTHER CONFIGURATION

- Adapters – pub/sub queues
  - Redis
  - Async

- Allowed request origins

CREATING THE CONSUMER

config/environment.rb

```
config.action_cable.mount_path = ‘/websocket'
```

layouts/application.html

```
= action_cable_meta_tag
```

assets/javascripts/cable.js

```
App.cable = ActionCable.createConsumer();
```

CONNECTION SETUP

- authorizing incoming connection

```
module ApplicationCable
  class Connection < ActionCable::Connection::Base
    identified_by :current_user

    def connect
      self.current_user = find_verified_user
    end

    protected

    def find_verified_user
      if verified_user = env['warden'].user
        verified_user
      else
        reject_unauthorized_connection
      end
    end
  end
end
```

03

CREATING SUBSCRIPTIONS

CLIENT SIDE

assets/javascripts/rooms.js

```
App.chat = App.cable.subscriptions.create(
      {
        channel: "RoomsChannel"
       },

      {
        subscribed: function(data) {
          ...
        },

        unsubscribed: function(data) {
          ...
        },

         received: function(data) {
          ...
        },
      });
```

SERVER SIDE

app/channels/rooms_channel.rb

```ruby
class RoomsChannel < ApplicationCable::Channel
  def subscribed
    stream_from "rooms_channel"
  end
end
```

CLIENT SIDE

- sending additional params

```
App.chat = App.cable.subscriptions.create(
      {
        channel: “RoomsChannel”,
        room_id: messages.data('room-id')
       },

      {
        subscribed: function(data) {
          ...
        },

        unsubscribed: function(data) {
          ...
        }
      });
```

SERVER SIDE

- receiving additional params

```
class RoomsChannel < ApplicationCable::Channel
  def subscribed
    stream_from “rooms_#{params[:room_id]}_channel”
  end
end
```

04

SENDING DATA TO SERVER

CLIENT SIDE

- send(data)

```
App.chat.send(
{
  message: textField.val(),
  room_id: messages.data('room-id')
});
```

SERVER SIDE

- def receive(data)

```
class RoomsChannel < ApplicationCable::Channel
  def receive(data)
    current_user.messages.create(
      room_id: data[‘room_id’],
      content: data[‘message’])
  end
end
```

CLIENT SIDE

- perform('method_name', data)

```
App.chat.perform('send_message', {
  message: textField.val(),
  room_id: messages.data('room-id')
});
```

SERVER SIDE

- def method_name(data)

```
class RoomsChannel < ApplicationCable::Channel
  def send_message(data)
    current_user.messages.create(
       room_id: data['room_id'],
       content: data['message'])
  end
end
```

05

BROADCASTING

CLIENT SIDE

```
App.chat = App.cable.subscriptions.create(
   received: function(data) {
     messages.append(data['message']);
  }
});
```

SERVER SIDE

```
class RoomsChannel < ApplicationCable::Channel
  def receive(data)
    ActionCable.server.broadcast “rooms_channel",
    message: message_partial(data[‘message’])
  end
end
```

RENDERING PARTIALS

```
class RoomsChannel < ApplicationCable::Channel
  def receive(data)
    ActionCable.server.broadcast “rooms_channel",
    message: message_partial(data[‘message’])
  end

  def message_partial(message)
     ApplicationController.renderer.render(
        partial: ‘rooms/message',
        locals: { message: message })
  end
end
```

06

DHH CHAT EXAMPLE

ROOM

```
.messages#js-messages data-room-id="#{@room.id}"
  - @room.messages.each do |message|
    = render 'message', message: message
```

SUBSCRIBING TO A ROOM CHAT

```
$(document).on('turbolinks:load', function() {

  var messages = $(‘#js-messages');

  if (messages.length > 0) {

    App.chat = App.cable.subscriptions.create({
      channel: "RoomsChannel",
      room_id: messages.data('room-id')
    },{
      received: function(data) {
        messages.append(data['message']);
      }
    });
  }
```

SUBSCRIBING TO A ROOM CHAT

```ruby
class RoomsChannel < ApplicationCable::Channel
  def subscribed
    stream_from "rooms_#{params['room_id']}_channel"
  end
end
```

MESSAGE FORM

```
  = simple_form_for @room.messages.new, url: "#", html: { id:
'js-new-message' } do |f|
    = f.input :content, input_html: { id: 'js-message-content' }
    = f.submit 'Send', class: 'js-send-message'
```

SENDING A MESSAGE

```
$('#js-new-message').submit(function(e) {
    var textField = $(this).find('#js-message-content');

    App.chat.send({
      message: textField.val(),
      room_id: messages.data('room-id')
    });

    textField.val('');
    e.preventDefault();
    return false;
  });
```

RECEIVING A MESSAGE ON SERVER

```ruby
class RoomsChannel < ApplicationCable::Channel
  def subscribed
    stream_from "rooms_#{params['room_id']}_channel"
  end

  def receive(data)
    current_user.messages.create(
      room_id: data[‘room_id'],
      content: data['message'])
  end
end
```

BROADCASTING A MESSAGE

```
class Message < ApplicationRecord
  after_create_commit { MessageJob.perform_later(self) }
end


class MessageJob < ApplicationJob
  queue_as :default

  def perform(message)
    ActionCable.server.broadcast "rooms_#{message.room.id}_channel",
                                 message: message_partial(message)
  end

  private

  def message_partial(message)
    ApplicationController.renderer.render(
      partial: ‘rooms/message',
      locals: { message: message })
  end
end
```

RECEIVING A MESSAGE

```
$(document).on('turbolinks:load', function() {

  var messages = $(‘#js-messages');

  if ($('#js-messages').length > 0) {

    App.chat = App.cable.subscriptions.create({
      channel: "RoomsChannel",
      room_id: messages.data('room-id')
    }, {
      received: function(data) {
        messages.append(data['message']);
      }
    });
  }
```

CONCLUSION

- full–stack WebSocket framework
- real–time communication
- intuitive
- many use cases

- still very new technology