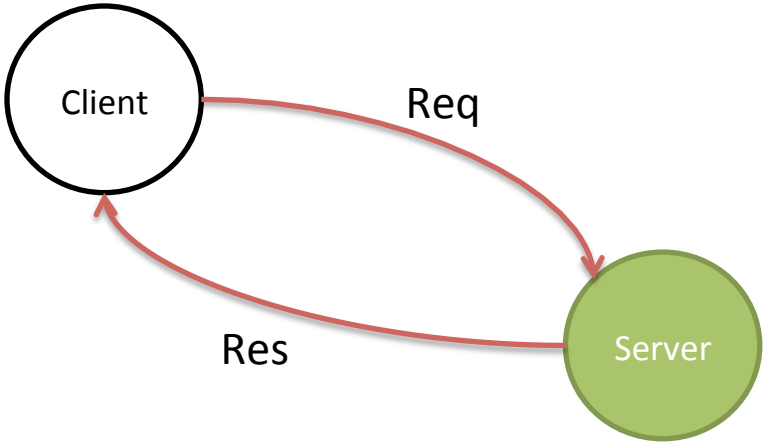
WebSockets


1-1/4 CR-MO S64H1-14L
1-1/8 CR-MO S64H1-18L
1-1/16 CR-MO S64H1-116L
1" CR-MO S64H1L
7/8 CR-MO S64H78L
15/16 CR-MO S64H11516L
11/16 CR-MO S64H11116L
13/16 CR-MO S64H1316L
3/4 CR-MO S64H34L
1/2 CR-MO S64H12L
5/8 CR-MO S64H58L
7/16 CR-MO S64H716L
9/16 CR-MO S64H916L

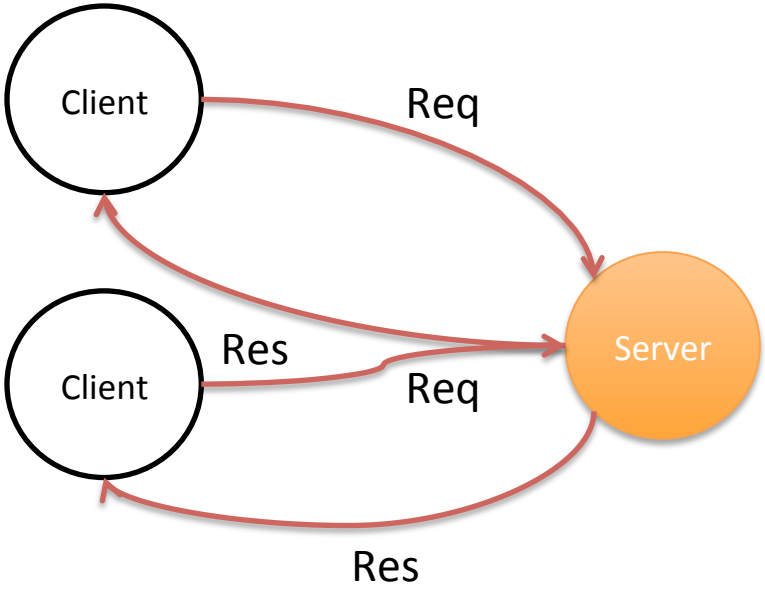
Problem with Ajax?

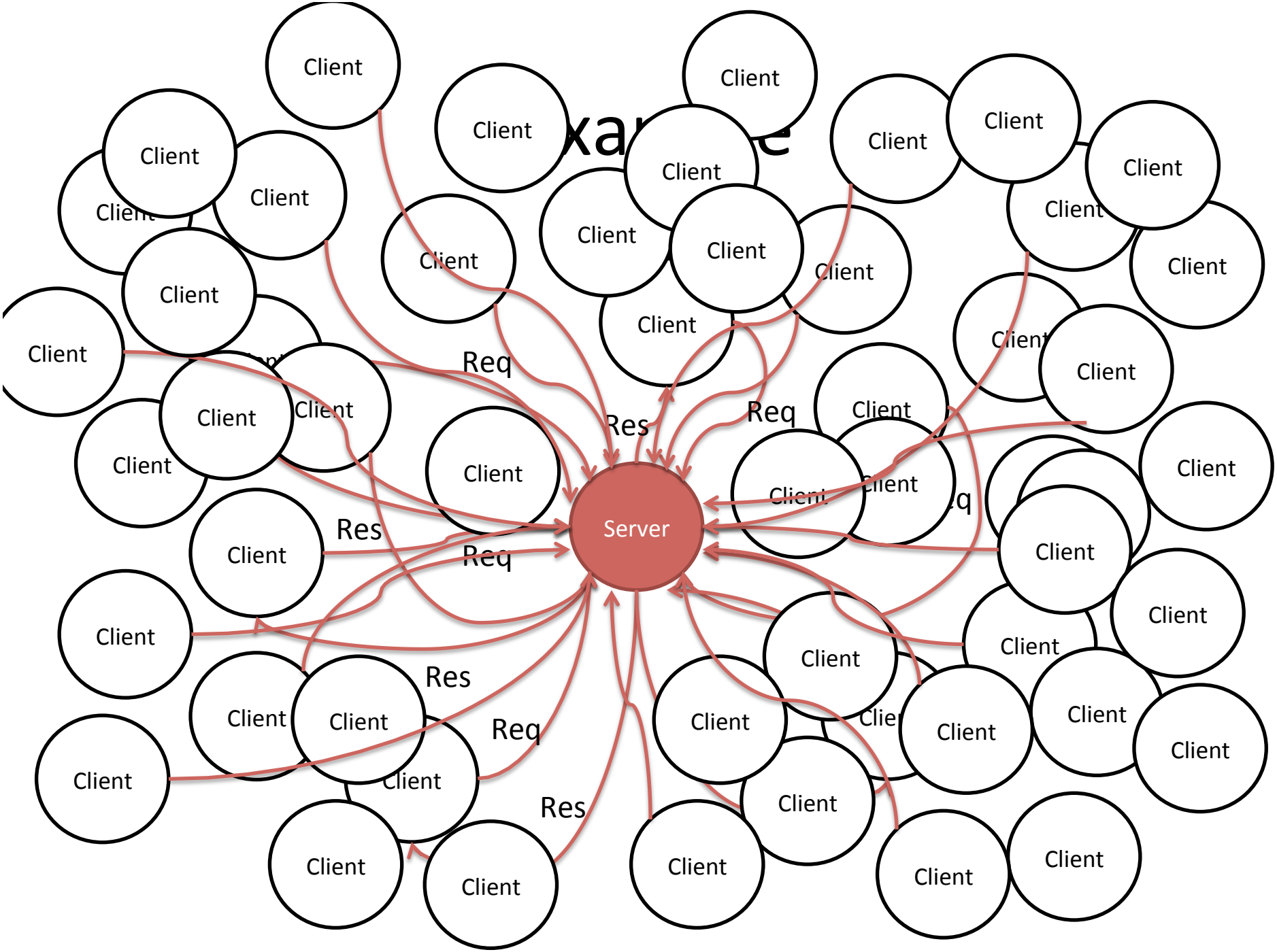
- **Ajax: client must *pull* from the server**
  - This works for many applications
  - What type of application does this work for?
- **But, what about real-time apps?**
  - They require notification **immediately** when state changes on the server!
  - Constant polling can lead to unnecessary overhead that can bog down the server and client

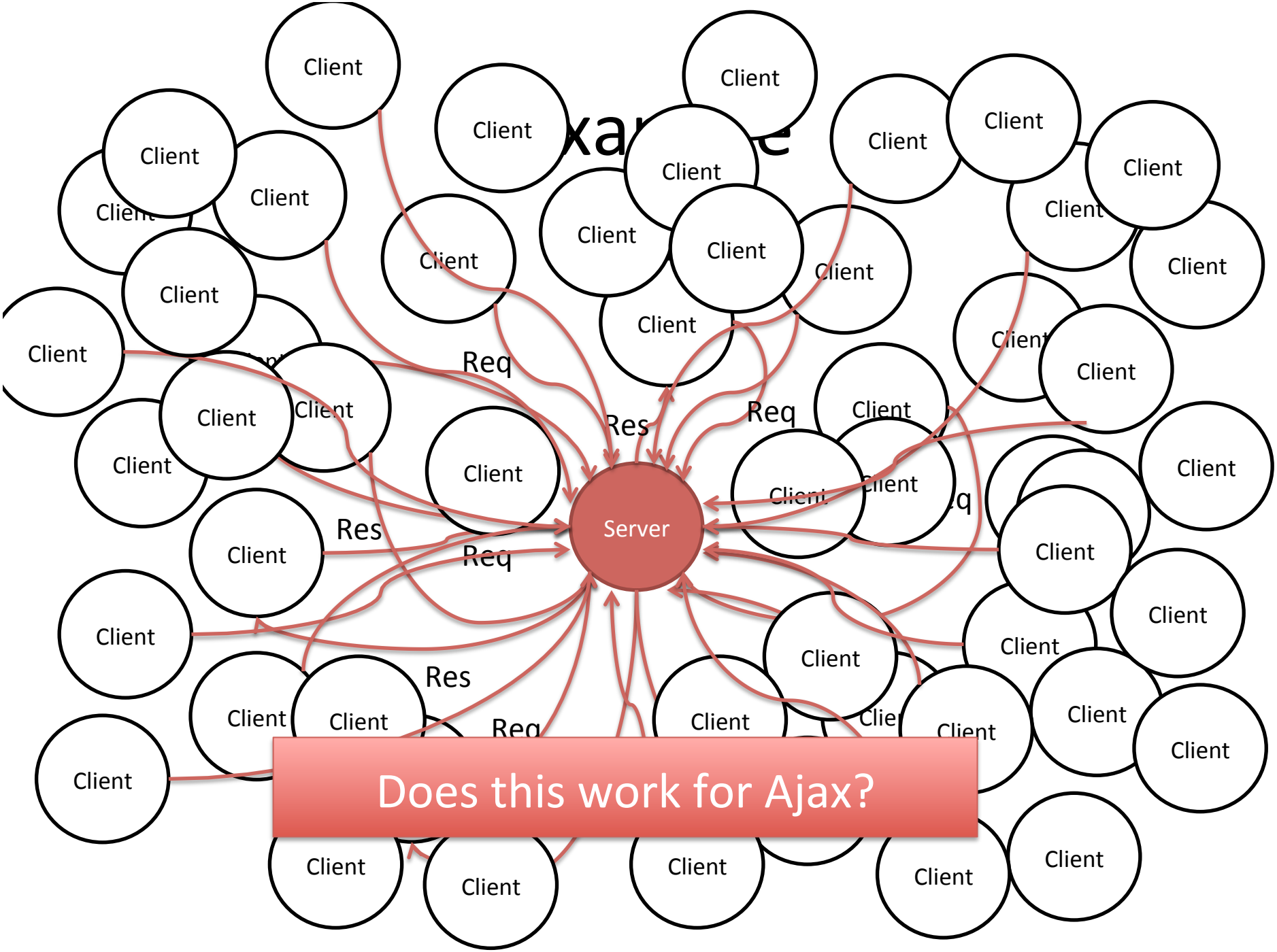
This works fine…



We are still ok, but…




Client
Server
Req
Res
Req
Res
Req
Res

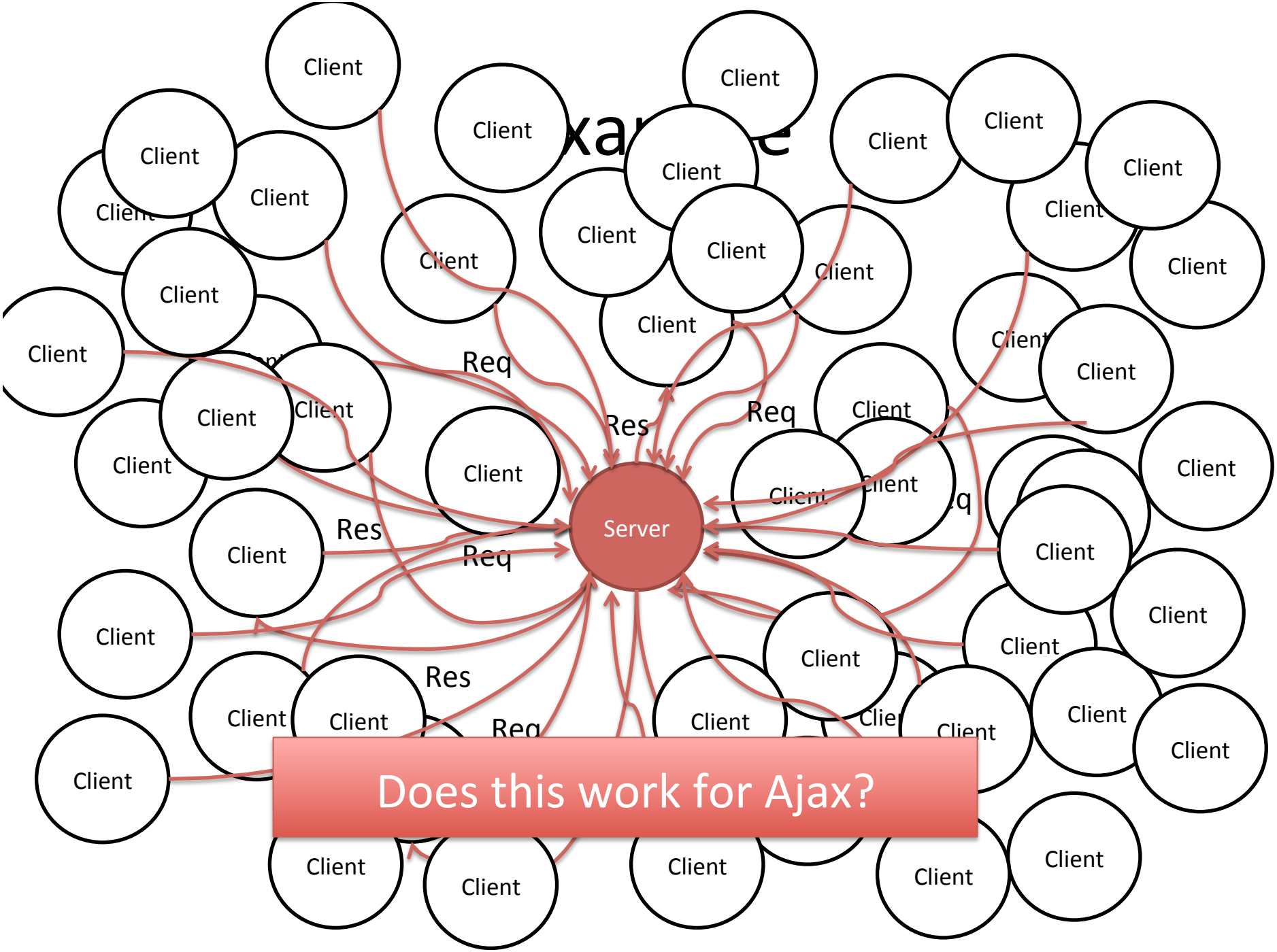

Client
Server
Req
Res
Req
Res
Req
Res
Req
Does this work for Ajax?

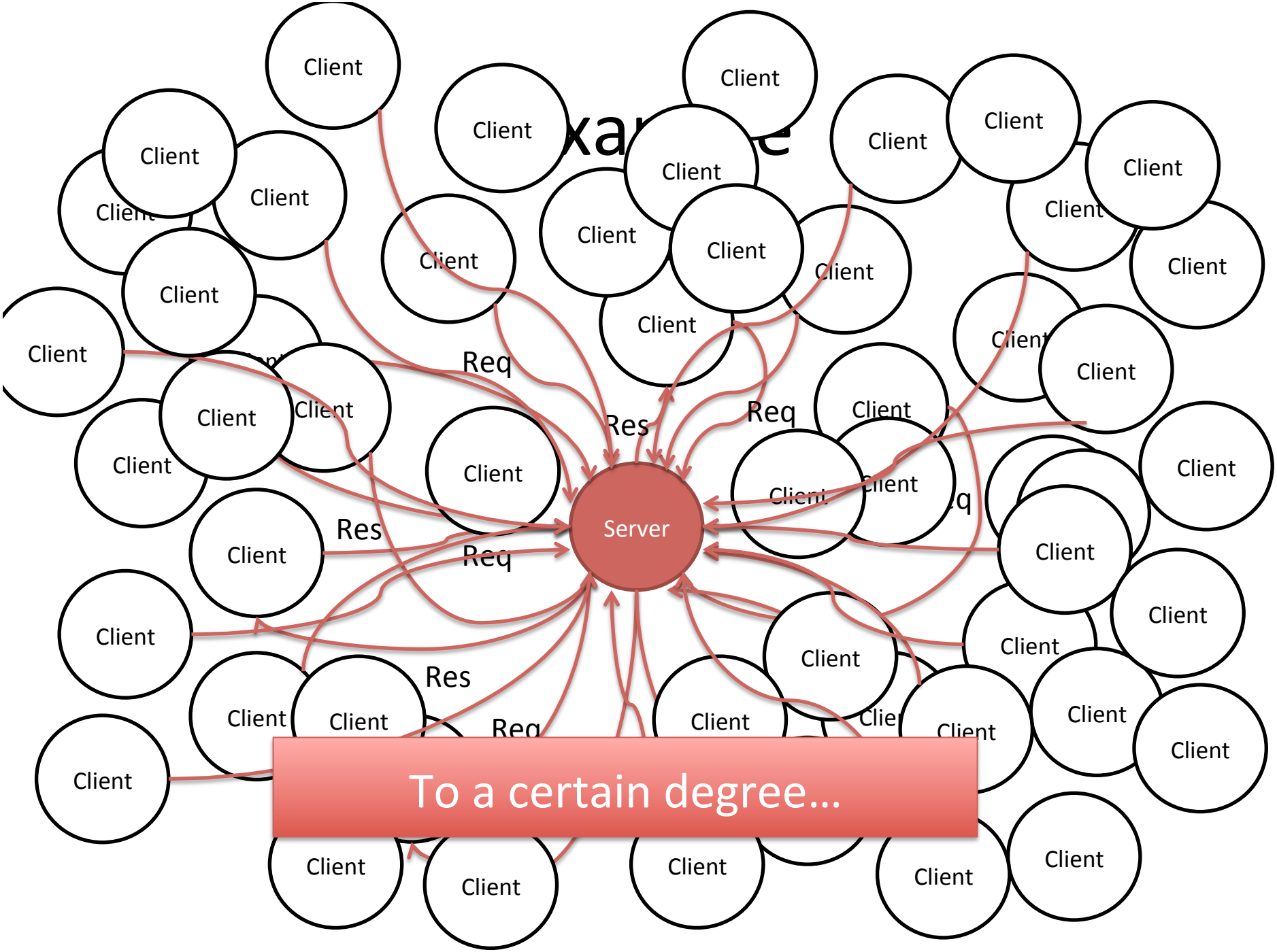
Communication Behavior

- **Behavior of Client Communication**
  - How does the client communicate with the server?
  - What is required?
- **Observations**
  - Frequent communication?
  - Real-time communication?
  - How many clients are communicating at once?
  - Scalability?

Ajax Application Properties

- **Communication**
  - Communication with the server does not have real-time constraints, it is not frequent
- **Scalability**
  - Can support many client connections at any given time as the connection is short-lived
  - Server does not need to maintain **many** live connections at once (sort of)
  - Ajax-style applications are scalable to a limit, the overhead of HTTP is substantial


Client
Server
Req
Res
Req
Res
Req
Res
Req
Does this work for Ajax?


Client
Server
Req
Res
Req
Res
Req
Res
To a certain degree…

Long Polling

- **What if we want real-time –like properties?**
- **Long Polling**
  - Send Ajax request
  - Server does not respond immediately, holds open the connection for a certain amount of time
  - Replies when data is available
- **Pro: reduces computing/network overhead**
- **Con: Still lots of overhead**
- **This is the approach before new techniques**

WebSockets

- **New Approach**
  - Relatively, 2011
- **Provides**
  - *Bi-directional, full-duplex communication*
    - Much like tradition sockets, but not quite
  - *Minimal communication overhead*
    - Binary protocol, no HTTP overhead
  - *Super Scalability*
    - Very Low Overhead
    - Open connections: limited by machine/OS configuration

How does it work?

Initial WebSocket Handshake Request:

```
GET /mychat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Response indicates the protocol switch:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

How does it work?

Initial WebSocket Handshake Request:

```
GET /mychat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Response indicates the protocol switch:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

After this, messages can be sent/received from client/server

WebSocket API

- **We will use:**
  - Socket.IO
  - http://socket.io
- **Works with Node.js & Express**
  - Other frameworks have similar libraries

Client API: Connecting

io.connect(URL)

```
var socket = io.connect('http://localhost');
```

Client API: Disconnecting

socket.disconnect()

```
var socket = io.connect('http://localhost');
socket.disconnect();
```

Client API: Receiving

socket.on(event, callback)

```
socket.on('chat-post', function (data) {
  console.log(data);
});
```

Client API: Sending

socket.emit(event, data)

```
socket.on('chat-post', function (data) {
  console.log(data);
  socket.emit('chat-post-rvd', {
    status : 'received'
  });
});
```

Client API: Sending

socket.emit(event, data)

```
socket.emit('chat-post-send', {
  msg : $('input#message').text();
});
```

Server API: Listening

socket_io.listen(server)

```
var socket_io = require('socket.io');
var io = socket_io.listen(server);
```

Server API: Connect/Disconnect

io.sockets.on(event, callback)

event = ‘connection’ | ‘disconnect’

```
var socket_io = require(‘socket.io’);
var io = socket_io.listen(app);

io.sockets.on(‘connection’,
  function (socket) {
    // Now we have a websocket: socket
  });
io.sockets.on(‘disconnect’, function () { });
```

Server API: Receiving

socket.on(channel, callback)

```
var socket_io = require('socket.io');
var io = socket_io.listen(app);

io.sockets.on('connection',
  function (socket) {
    socket.on('chat-post-send', function (data) {
      console.log(data.msg);
    });
  });
```

Server API: Sending

socket.emit(channel, callback)

```
var socket_io = require('socket.io');
var io = socket_io.listen(app);

io.sockets.on('connection',
  function (socket) {
    socket.on('chat-post-send', function (data) {
      socket.emit('chat-post-rcd', { status: 'ok' });
    });
  });
```

Server API: Broadcasting

socket.emit(channel, callback)

```
var socket_io = require('socket.io');
var io = socket_io.listen(app);

io.sockets.on('connection',
  function (socket) {
    socket.broadcast.emit('all', 'user connected');
    // OR
    io.sockets.emit('all', 'user connected');
  });
```

Server API: Session Support

socket.set(key, value, callback)

```
io.sockets.on('connection',
  function (socket) {
    socket.on('set user', function (user) {
      socket.set('user', user, function () {});
    });
  });
```

Server API: Session Support

socket.get(key, callback)

callback : function (err, value)

```
io.sockets.on('connection',
  function (socket) {
    socket.on('get user', function () {
      socket.get('user', function (err, user) {
        socket.emit('user name', user);
      });
    });
  });
```