WebRTC

AWPUG July 24th, 2014

Hello.

I’m Josh Marshall.
I work at uStudio.
I’m a WebRTC enthusiast.

(caveats)

This talk is more **Web** than **Python**.

“This is fun but not very useful.”

Localhost is cheating.

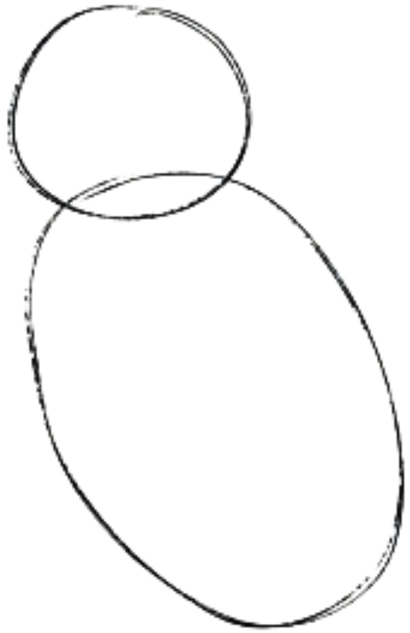


Fig 1. Draw two circles

Fig 2. Draw the rest of the damn Owl

(caveats)

This is step one.

What is WebRTC?

> WebRTC enables web browsers with real time communication capabilities via simple JavaScript APIs.

- webrtc.org

WebRTC provides two things.

Access to media devices.
Peer to peer communication.

(The underlying tech isn’t necessarily new.)

“I could already do that with Flash.”

(you could)

“Cisco was doing this video conferencing stuff in the 90’s.”

(they were)

“P2P, SCTP, ICE… I’ve done all that.”

(see Skype, BitTorrent, the internet…)

(sidebar)

Browser tech doesn’t have to invent.
It just needs to enable.

getUserMedia()
RTCPeerConnection()
RTCSessionDescription()
RTCIceCandidate()

Device(s) getUserMedia()

Connection RTCPeerConnection()

Handshake RTCSessionDescription()

Routing RTCIceCandidate()

Devices

```
var constraints = {…};
navigator.getUserMedia(constraints, success, failure);
//webkitGetUserMedia();
//mozGetUserMedia();
```

```
var constraints = {                    // relaxed
   video: {
      mandatory: {                     // or “optional”
         minWidth: 1280,               // er, recommended…
         minHeight: 720,
         minFrameRate: 30,             // probably don’t
         sourceId: “foobar”            // more in a sec
      }
   }                                   // audio goes here…
};
```

```
var onsuccess = function(stream) {
   var url = URL.createObjectURL(stream);    // blob URL
   video.src = url;                          // have a <video>
   video.play();                             // !important
};
```

Demo 1

Let’s play with the media.

```
var context = swapCanvas.getContext(“2d”);
context.drawImage(video, 0, 0, width, height);
var data = context.getImageData(0, 0, width, height);
// do stuff with data
outContext.putImageData(data, 0, 0);
setTimeout(function() {…}, 0);
```

```
window.audioContext();
// webkitAudioContext
// mozAudioContext
```

Demo 2

(One more just for fun.)

Demo 3

So that’s getting sources.

Let’s make it multiplayer.

Connections

```
var conn = new window.RTCPeerConnection(…);
//do communication stuff();
conn.close();
```

Peer connections are not useful until:

- addresses are negotiated
- offers and answers are shared
- streams / channels are added

Crossing the NAT

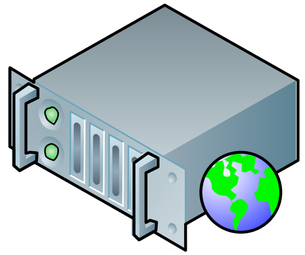
A story of ICE, STUN, and TURN.

Big, fat caveat.

Lots of acronyms incoming.

Not a lot of interface.

(so don’t worry if you get lost)

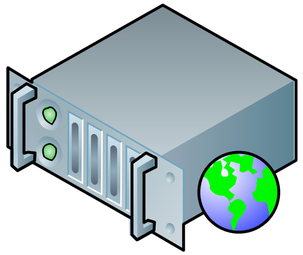


STUN / TURN
server(s)

client one

client two

Two clients, alike in symmetry…



- 127.0.0.1
- 192.168.0.10

- 127.0.0.1
- 10.0.0.5

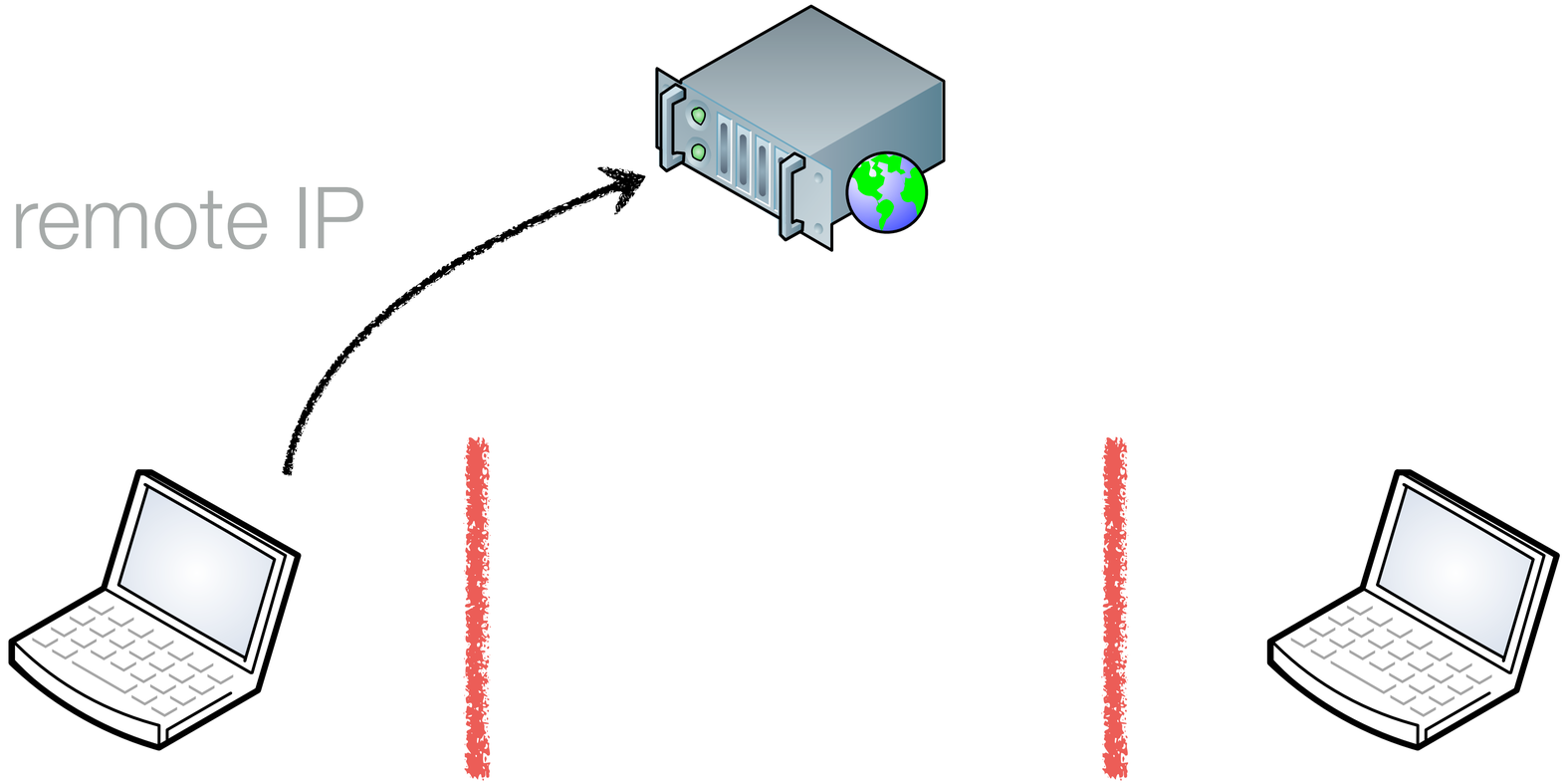
Cannot talk directly to each other

Interactive Connectivity Establishment (ICE)

- RFC 5245 for the standards geeks
- Generates a set of IP addresses
- Uses SIP / SDP, STUN, and TURN

STUN

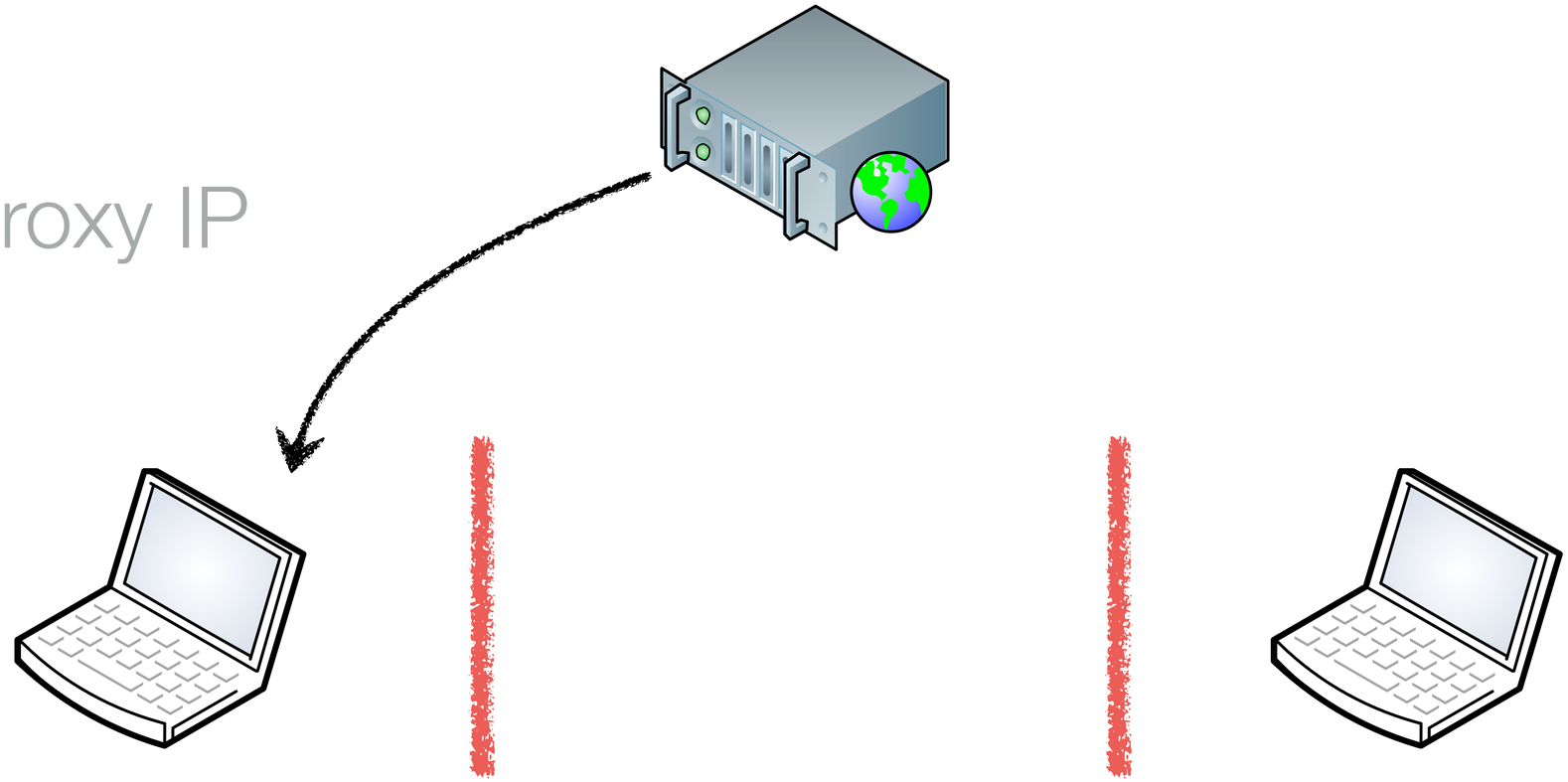
asks for remote IP



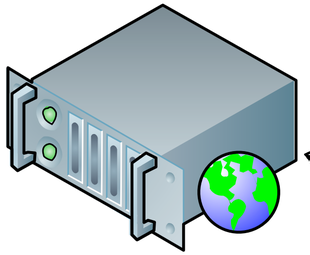
- 127.0.0.1
- 192.168.0.10
- 68.45.3.115

TURN

server proxy IP



- 127.0.0.1
- 192.168.0.10
- 68.45.3.115
- 184.16.20.12



same process

- 127.0.0.1
- 192.168.0.10
- 68.45.3.115
- 184.16.20.12

- 127.0.0.1
- 10.0.0.5
- 43.151.16.3
- 174.121.2.11

Acronym city, but FYI for now

- Simple Traversal of User Datagram - RFC 3489
- Traversal Using Relay NAT - RFC 5766
- Session Initiation Protocol - RFC 3261
- Session Description Protocol - RFC 2327

```
var servers = {
  iceServers: [
    {url: “stun:stun.l.google.com:19302”},
    {url: “turn:turn.s.proxy.com”}
  ]
};
var conn = new window.RTCPeerConnection(servers);
conn.onicecandidate = function(candidate) {…}; // or
conn.addIceCandidate(…);
```

How do we communicate these addresses?
(and media capabilities, protocols, etc.)

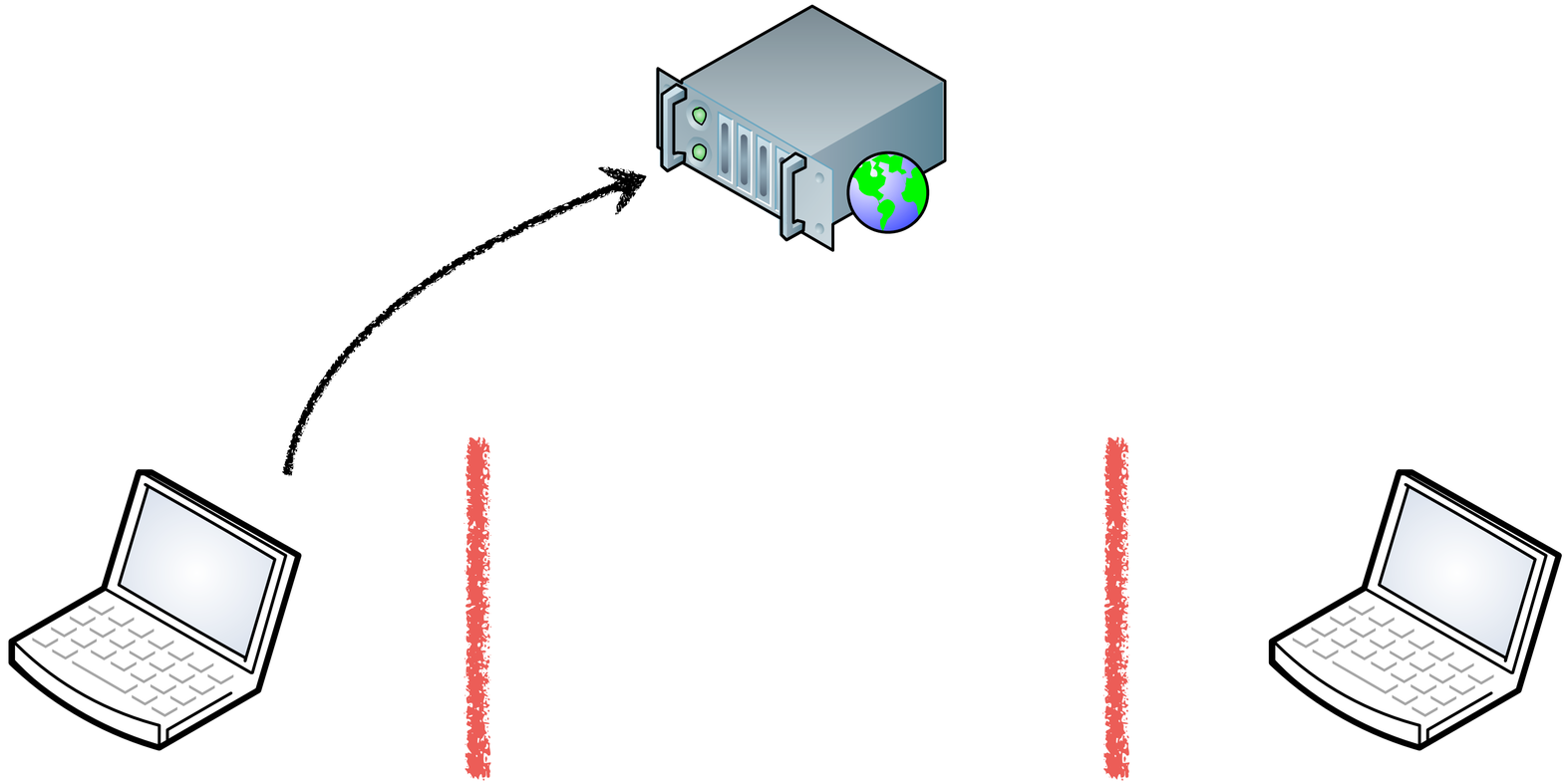
Signaling

Signaling is not part of the WebRTC spec.
(it’s up to you.)

You need a service that:

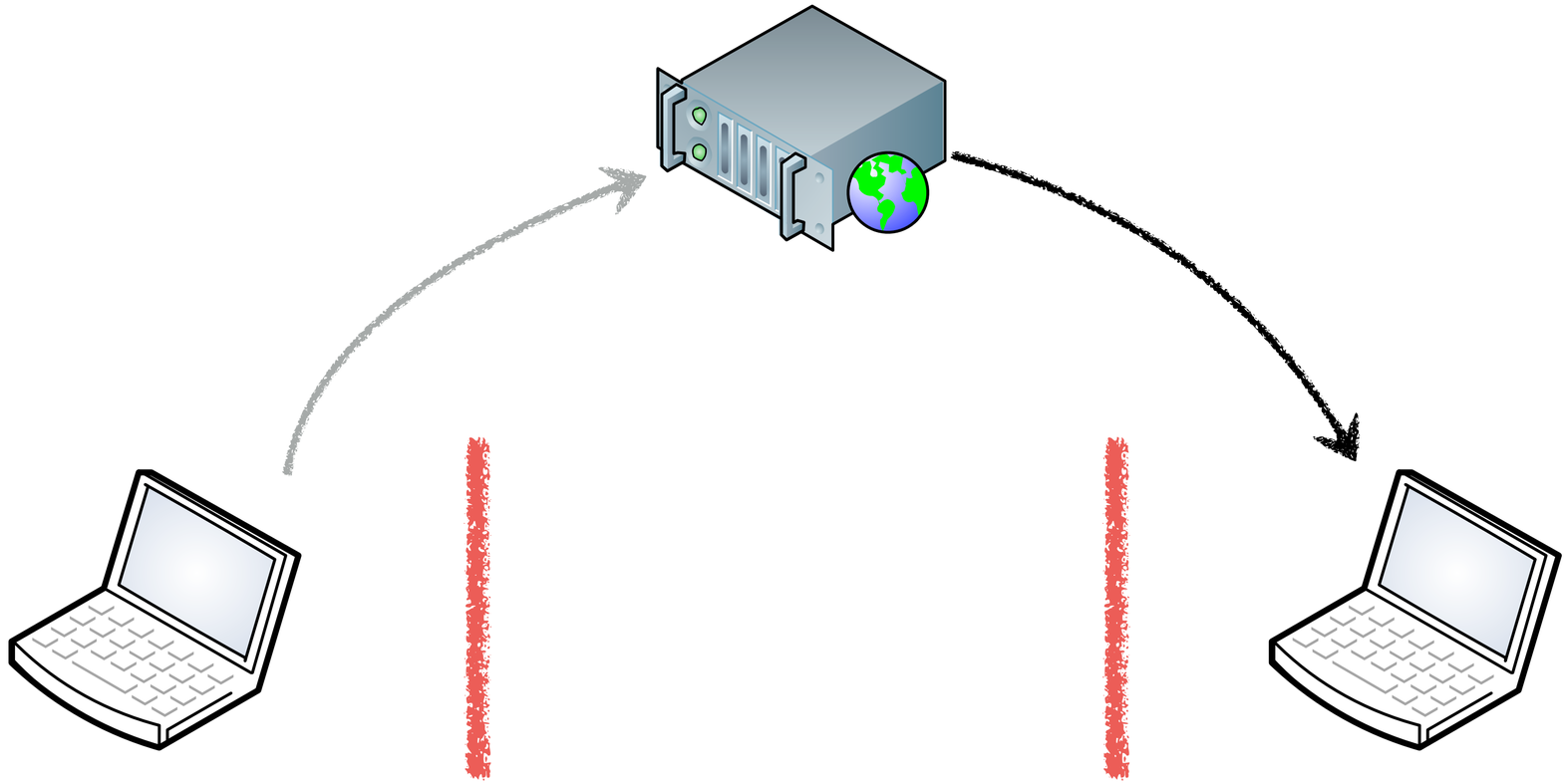
- is visible to all clients
- is lightweight, probably event-based
- can share addresses and offers / answers

This is where Python comes in.



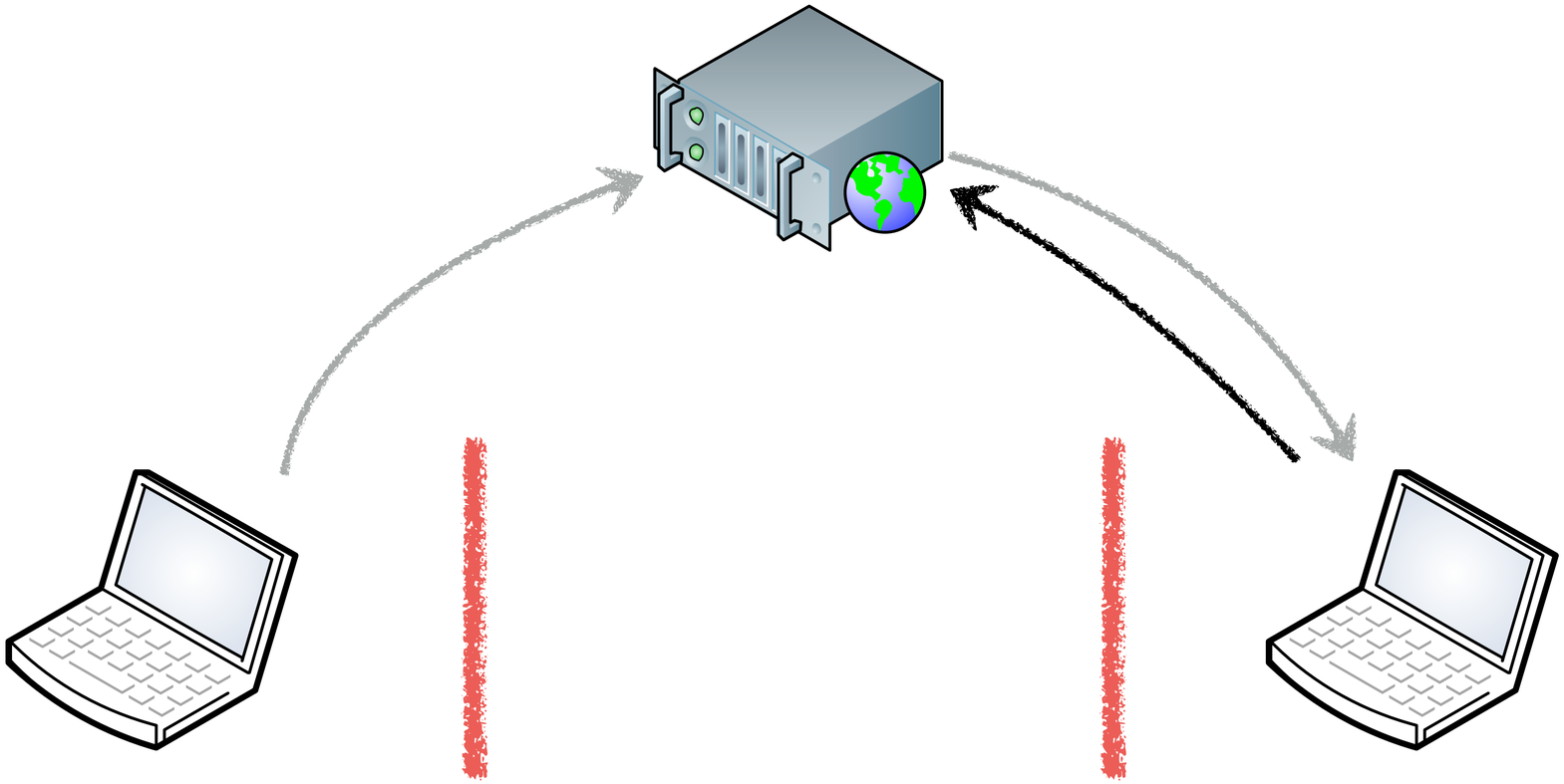
send offer

via signaling



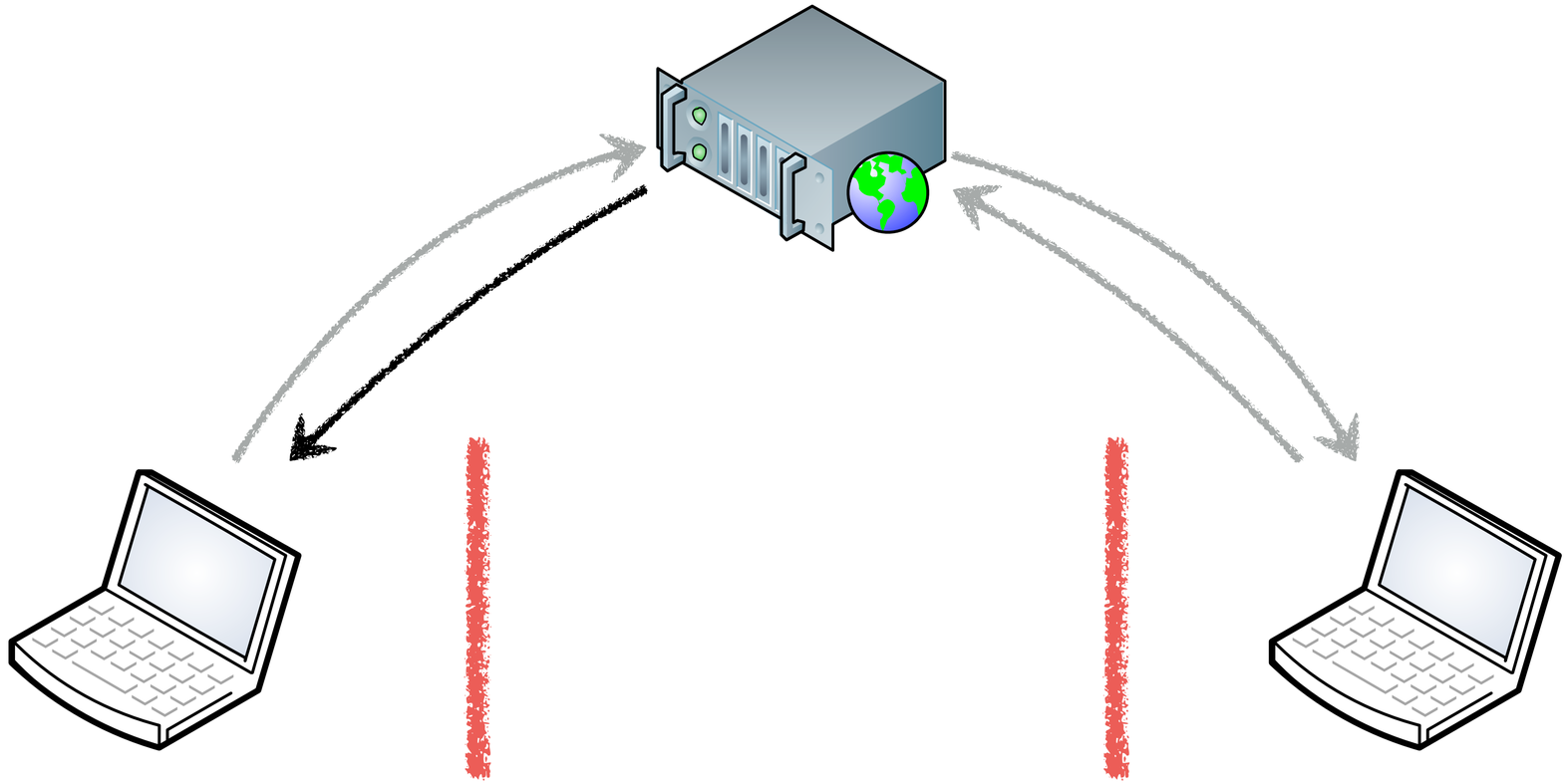
recv offer

via signaling



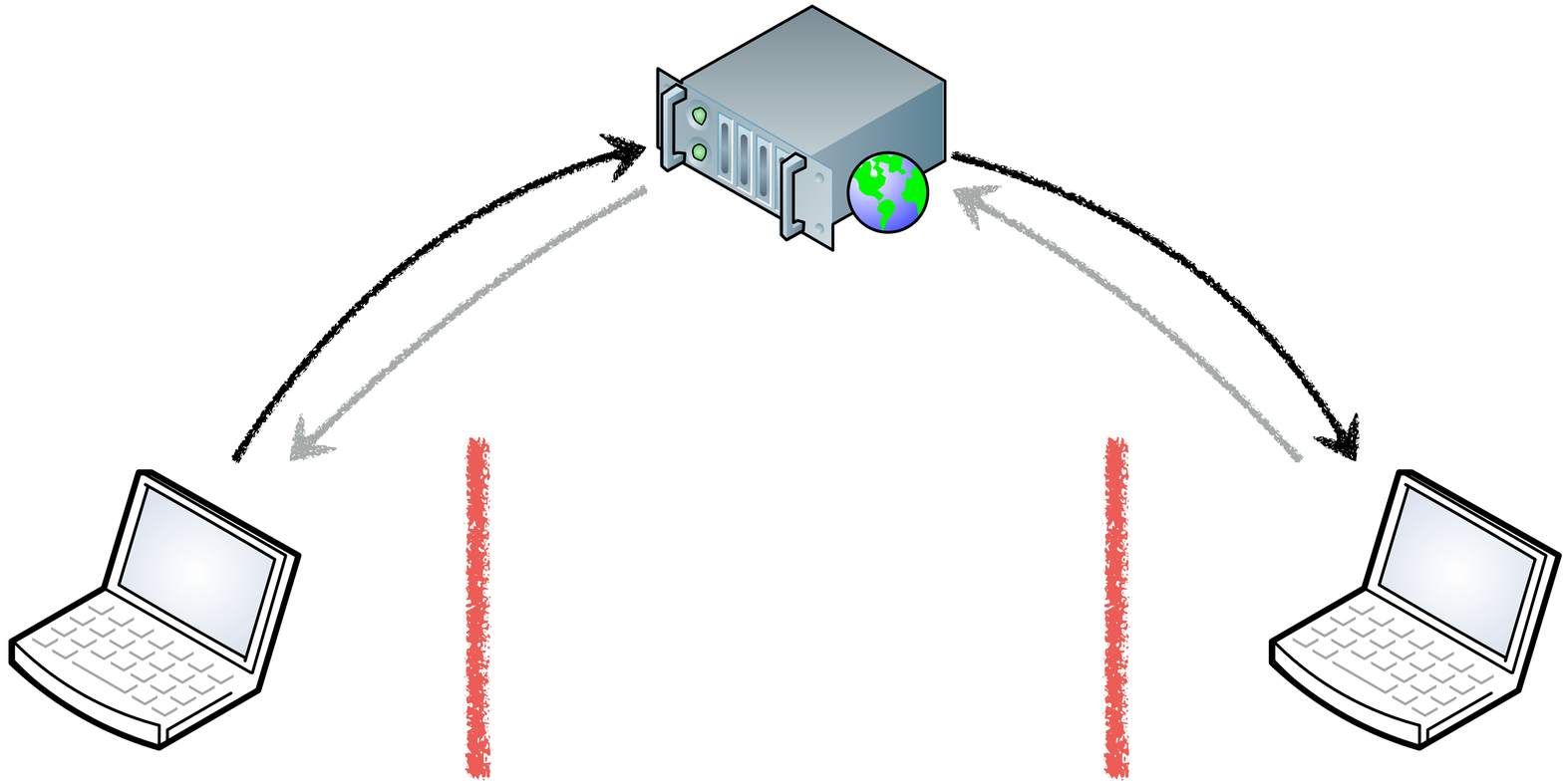
send answer

via signaling



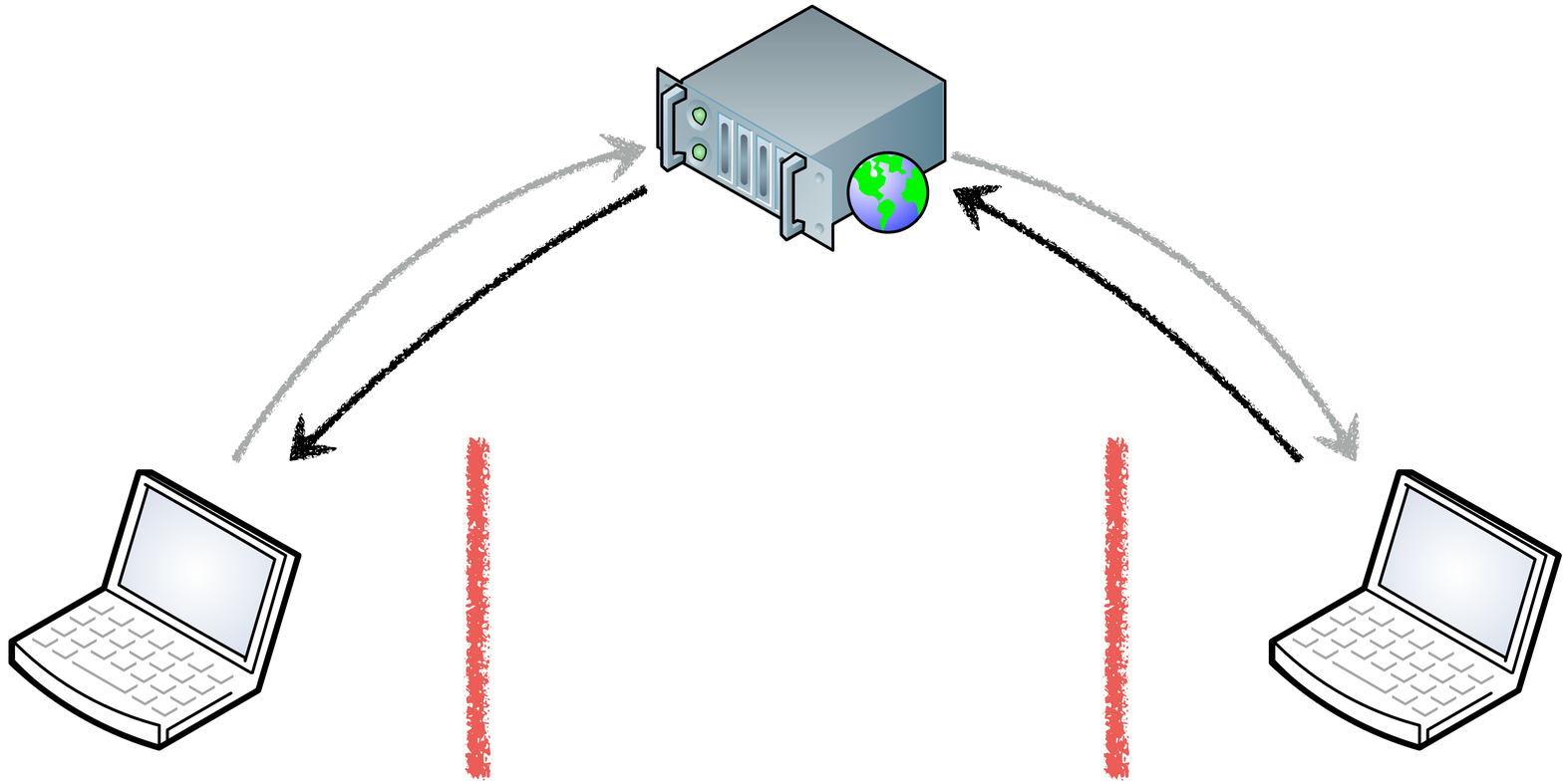
recv answer

(via signaling)

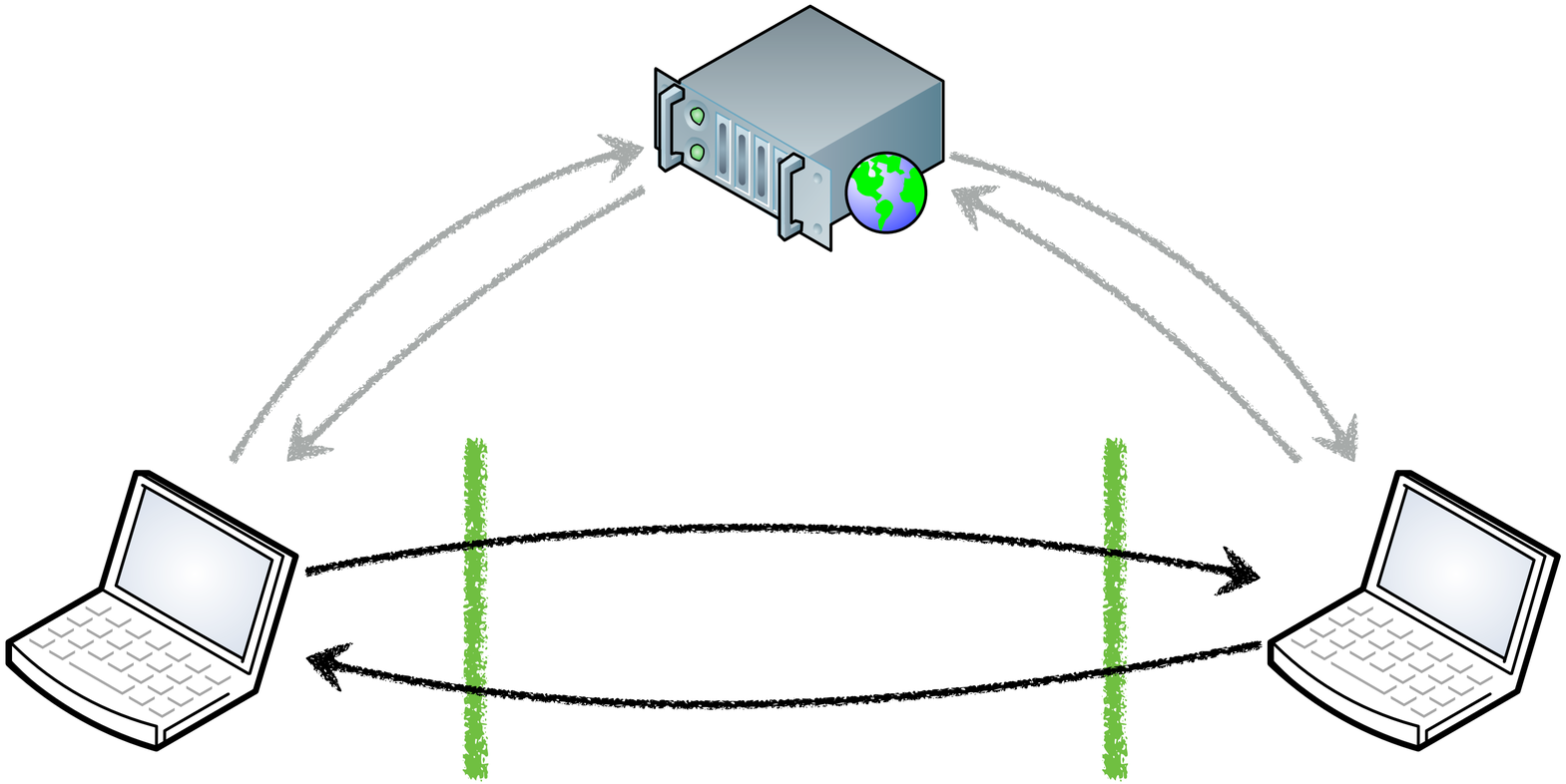


send ice candidates

(via signaling)



return ice candidates
(via signaling)



attempt NAT punchthrough

(and hopefully succeed)

```
conn = new RTCPeerConnection(…);          // everyone does this
conn.createOffer(function(offer) {
   conn.setLocalDescription(…);           // initiator sets offer
   // send offer via signaling            // sends it via websocket
});
// wait for answer                        // recvs via websocket
conn.setRemoteDescription(…);
conn.onicecandidate = function(e) {       // local candidate
   // send candidate via signaling        // send via web socket
};
// wait for candidates
conn.addIceCandidate(…);                  // remote candidates
```

```
conn = new RTCPeerConnection(…);          // everyone does this
// wait for offer                         // offer comes via websocket
conn.setRemoteDescription(…);
conn.createAnswer(function(answer) {
   conn.setLocalDescription(…);           // answer is local
   // send answer via signaling           // sent via websocket
});
conn.onicecandidate = function(e) {       // rest same as initiator
   // send candidate via signaling
};
// wait for candidates
conn.addIceCandidate(…);
```

And that’s just one-to-one.

Streams are added while connecting peers.

- video and audio streams
- data channels for arbitrary messages

Initiator

```
conn = new RTCPeerConnection(…);
// create streams or channels
conn.createOffer(…);
conn.setRemoteDescription(…);
```

Recipient

```
conn = new RTCPeerConnection(…);
conn.setRemoteDescription(…);
// wait for streams or channels
conn.createAnswer(…);
```

Demo 4

```
// media streams
conn.addStream(…);
conn.onstream = function(…);

// data channels
conn.addDataChannel(“name”);
conn.ondatachannel = function(…);
```

// media streams

DTLS-SRTP is the MTI

No video codec MTI

// data channels

DTLS-SCTP for most

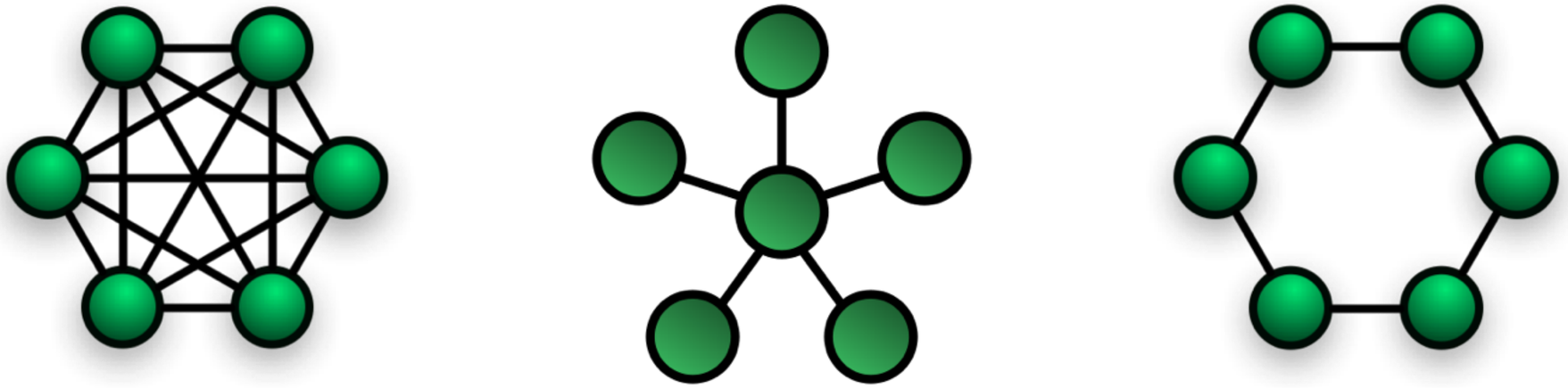
RTP-SAVPF deprecated

There are many examples of one-to-one video and audio chat on the inter web.

Talky.IO

Data channels are (IMO) the sleeper feature.

Where from here?



How do you scale connections?

- fully connected mesh - (falls over)
- star - puts pressure on central servers
- ring - more complicated

MCU (Multipoint Control Unit)

- composites many streams to one
- reduces connections drastically
- requires beefy server
- eliminates P2P
- can do slick stuff (archive, AR, etc.)

Who is using WebRTC today?

- Google Hangouts just went WebRTC
- HipChat uses it in native clients
- Amazon Fire Support / MayDay
- Talky.IO, Sqwiggle, PeerCDN, etc.

Things we didn’t talk about

- SDP versus ORTC
- Native client and server libraries (C++)
- MediaStreamTrack.getSources();
- Echo / noise cancellation
- Synchronizing events
- SCTP and media constraints
- Browser compatibilities (haha) and plugins
- Lots more.

Lots of resources

- http://webrtc.org
- https://github.com/GoogleChrome/webrtc
- http://bloggeek.me/
- http://webrtchacks.com/