









# FAC 9 WebRTC Workshop

Peter Wilson, IpCortex


# Making the web 'real-time'

- Audio calls, Video calls, Phone calls (via gateway)
- Using 'standard' technologies
  - Vanilla browser
  - No proprietary plugins (Flash)
  - No proprietary protocols (Skype)
- The standardised framework is WebRTC
  - Web Real Time Communication
- Most browser are adopting this now

# Is it all hype?

								
	Canary	Chrome	Opera	Nightly	Firefox	Bowser	Edge	Safari
PeerConnection API	Green	Green	Green	Green	Green	Green	Yellow	Red
getUserMedia	Green	Green	Green	Green	Green	Green	Green	Red
dataChannels	Green	Green	Green	Green	Green	Green	Red	Red
TURN support	Green	Green	Green	Green	Green	Green	Green	Red
Echo cancellation	Green	Green	Green	Green	Green	Green	Green	Red
MediaStream API	Green	Green	Green	Green	Green	Green	Green	Red
mediaConstraints	Yellow	Yellow	Yellow	Green	Green	Yellow	Yellow	Red
Multiple Streams	Yellow	Yellow	Yellow	Green	Green	Green	Green	Red
Simulcast	Yellow	Yellow	Yellow	Yellow	Yellow	Red	Yellow	Red
Screen Sharing	Yellow	Yellow	Yellow	Yellow	Yellow	Red	Red	Red
Stream re-broadcasting	Green	Green	Green	Green	Green	Red	Red	Red
getStats API	Yellow	Yellow	Yellow	Green	Green	Red	Green	Red
ORTC API	Red	Red	Red	Red	Red	Red	Green	Red
H.264 video	Green	Green	Green	Green	Green	Green	Green	Red
VP8 video	Green	Green	Green	Green	Green	Green	Red	Red
Solid interoperability	Green	Green	Green	Green	Green	Green	Yellow	Red
srcObject in media element	Green	Green	Green	Green	Green	Red	Green	Red
Promise based getUserMedia	Green	Green	Green	Green	Green	Green	Green	Red
Promise based PeerConnection API	Green	Green	Green	Green	Green	Green	Yellow	Red
WebAudio Integration	Green	Yellow	Yellow	Green	Green	Red	Yellow	Red
MediaRecorder Integration	Green	Green	Green	Green	Green	Red	Red	Red
Canvas Integration	Green	Green	Green	Green	Green	Red	Red	Red
Test support	Green	Green	Green	Green	Green	Red	Green	Red

# Apple Safari



WebKit  
Open Source Web Browser Engine

BlogDownloadsFeature StatusReporting BugsContribute ▾

## WebKit Feature Status

### Filters

webrtc

☐ Done☐ In Development☐ Removed☐ Under Consideration☐ Partial Support

### Features

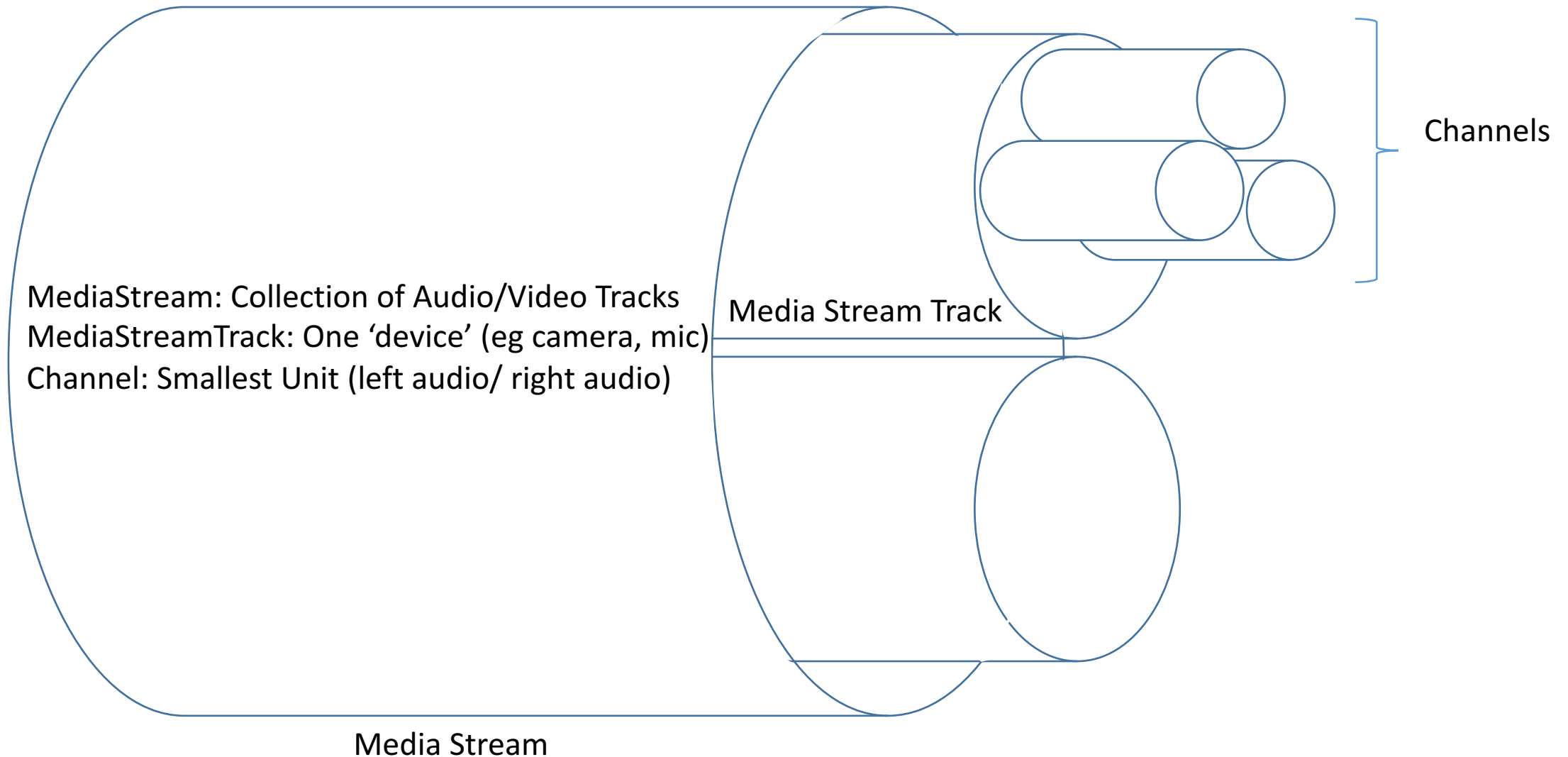
WebRTC

In Development ▾

Cannot find something? You can contact [@webkit](#) on Twitter or contact the [webkit-help](#) mailing list for questions.

You can also [contribute to features](#) directly, the entire project is Open Source. To report bugs on existing features or check existing bug reports, see <https://bugs.webkit.org>.

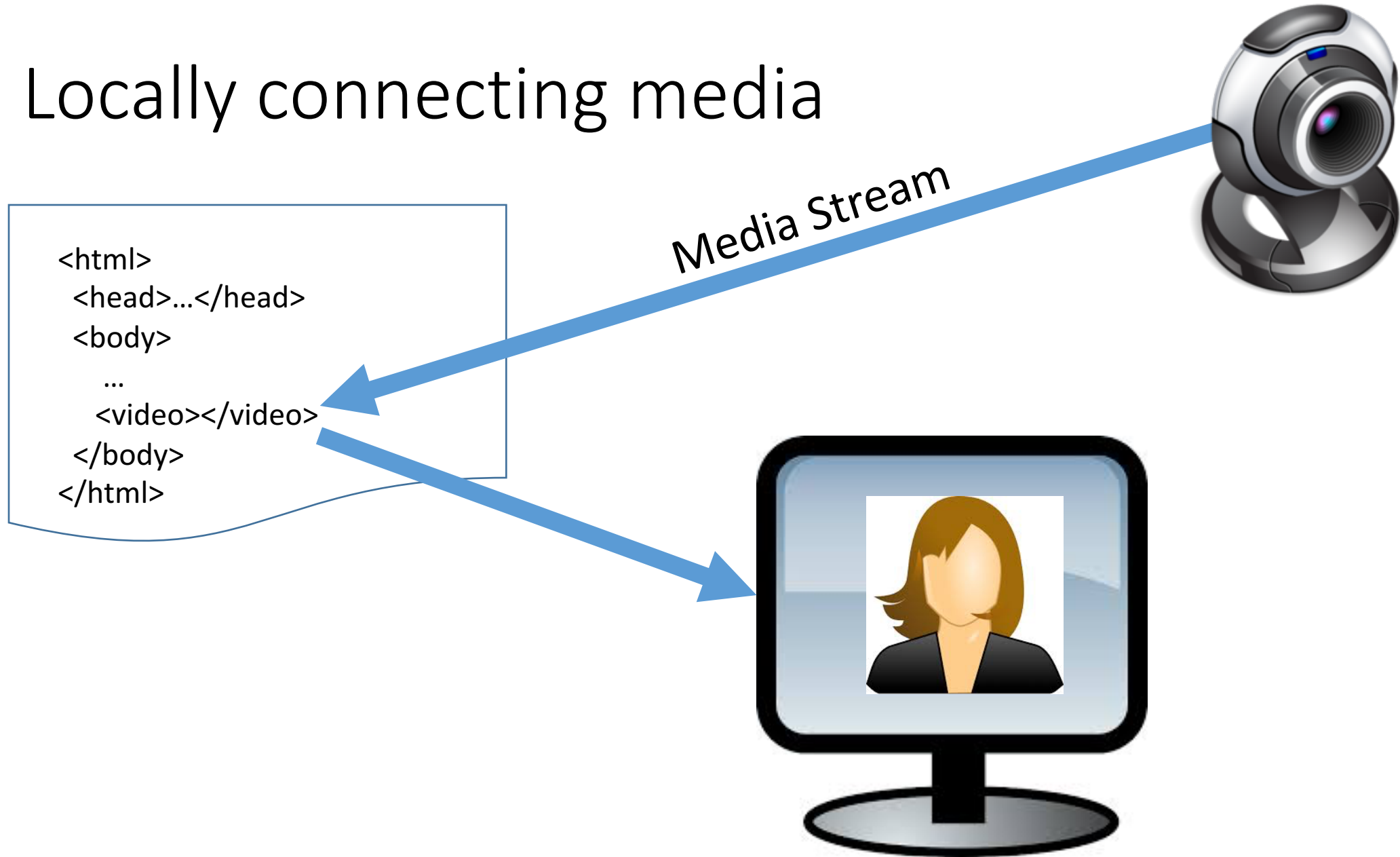
# Media Streams



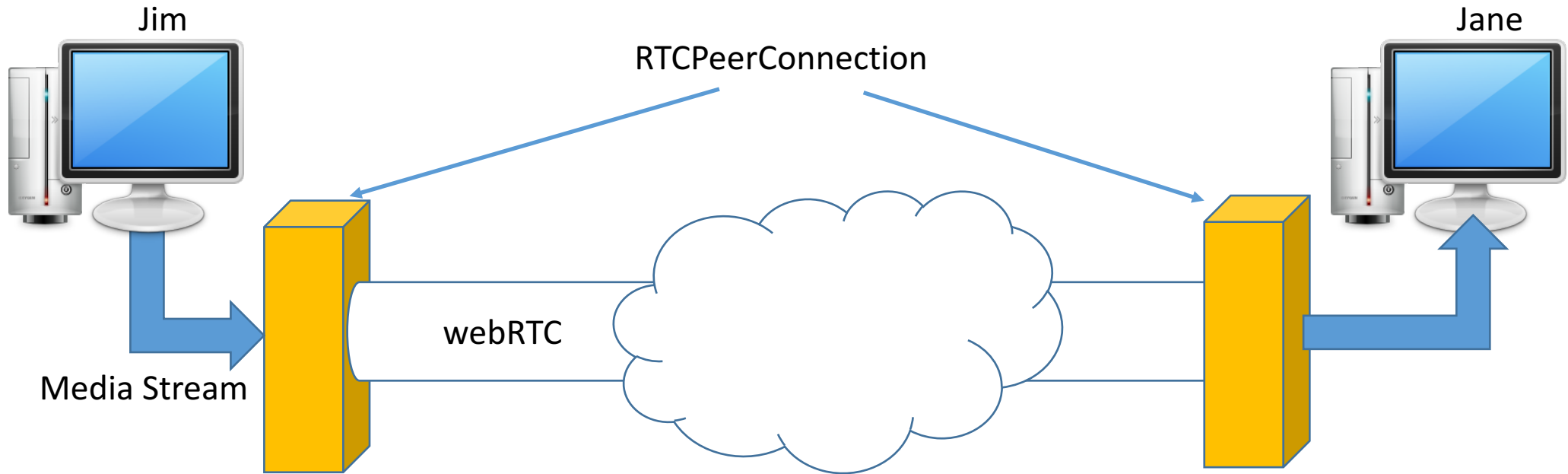
# Media Stream

- Unidirectional
- Has one input and out output
- Local inputs:
  - Microphone
  - Camera
  - *RTCPeerConnection*
- Outputs:
  - <video> tag
  - *RTCPeerConnection*

# Locally connecting media



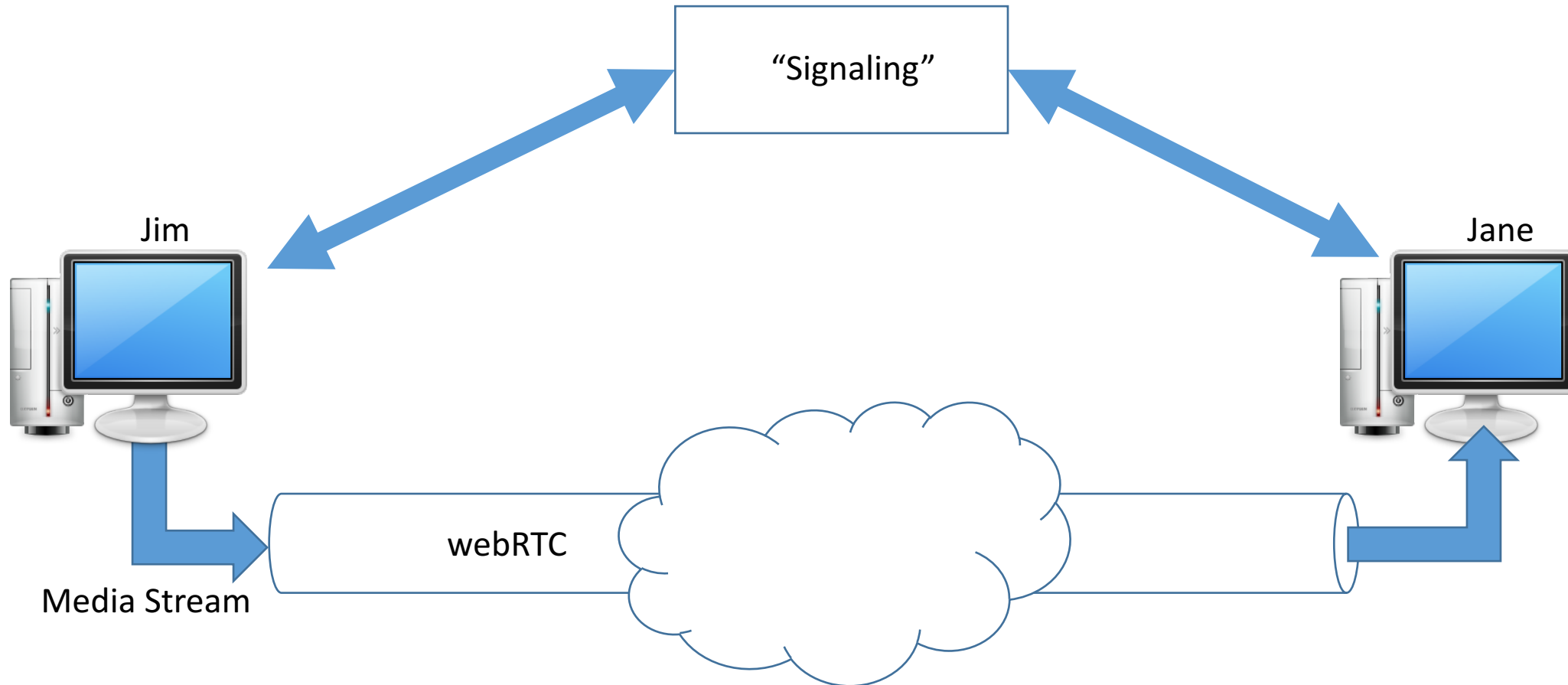
# Connecting between devices



- How do Jim and Jane find each other?
- How is their traffic routed across the Internet through corporate firewalls?
- How is permission asked to accept the call at the receiving end?



# Signaling



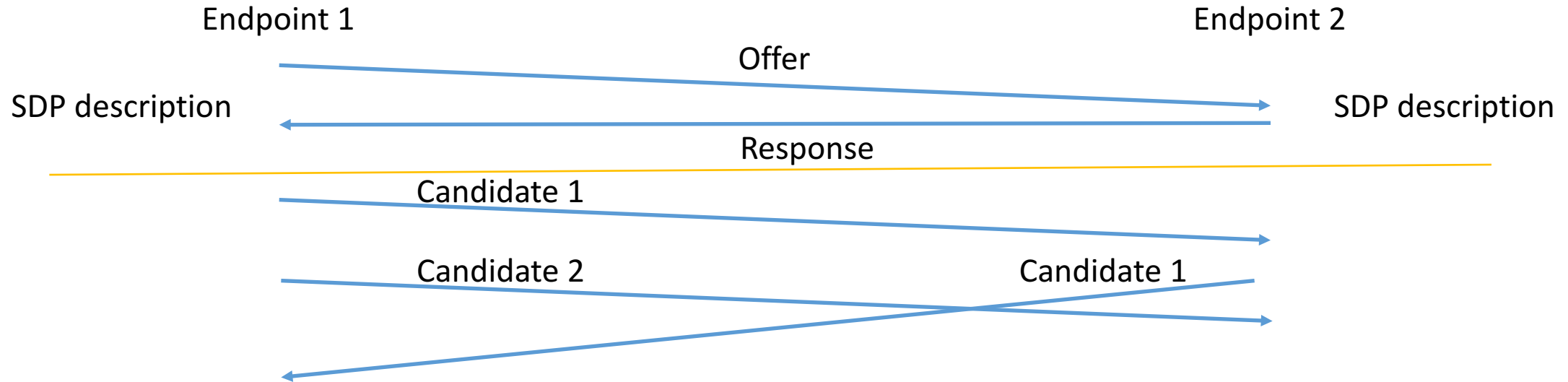
# Signaling

- Allows two ends of an RTCPeerConnections to find each other
- NOT standardised
  - Not necessary and much of this is application specific
- Examples
  - IpCortex PABX – signaling, directory, presence and PTSN gateway (phone calls)
  - webRTC.io – one of the first libraries
- Must be accessible to both parties
- Provide a means of relaying information between the two parties
- Should encrypt all communications
- *Usually* a separate server

# What is signaling used for?

- Very simple WebRTC requirements
  - Transfer Javascript Objects between endpoints (eg serialised as JSON)
  - Send media 'offers' and 'responses' between endpoints
  - Swap communication 'candidates' between endpoints
- Other signaling usually required for an application
  - Endpoint discovery/name mapping (I'm Fred, I want to talk to Jane)
  - Presence (I'm Fred and I'm 'online')

# Typical signaling exchange



- Offers are sent as Session Description Protocol messages
- Candidates give options for how to connect end points across a network
  - Accomplished via the Interactive Connectivity Establishment (ICE) framework
  - Multiple 'candidates' are tested concurrently with the first (fastest) used

# Signaling Overhead

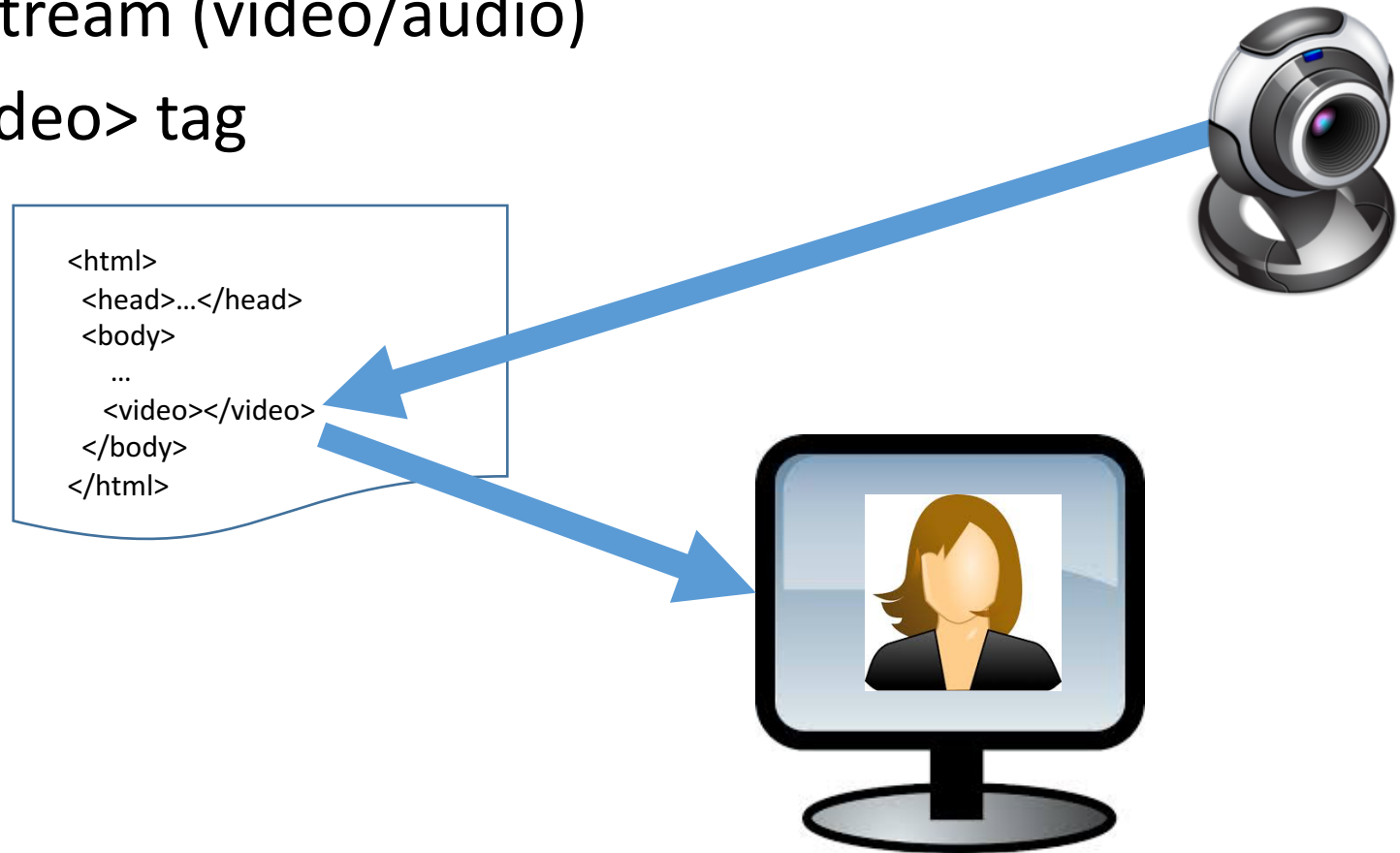
- Intentionally very low overhead
- Typically 24 exchanges per WebRTC session
- ~10K data exchanged
- Many techniques available
  - REST Polling
  - HTTP 'Long Poll'
  - REST to the signaling server/ EventSource distribution to clients
  - WebSocket bi-directional 'pipes'
- Only requires text transfer
- Not just for setup though – media can change during a call

# Workshop – what we’re going to do...

1. Configure simple HTTPS server to serve scripts
2. Using streams locally  
Create a <video> tag in a static page  
Request media (camera and microphone)  
Attach media to video tag
3. Local Peer Connection and Signaling
4. Using streams between devices over a network
5. *Building a simple remote ‘Presentation’ application using webRTC*

## 2. Local stream

- Request local media stream (video/audio)
- Attach to browser `<video>` tag



## 2. Local media streams

```
var promise = navigator.mediaDevices.getUserMedia({  
  video: true,  
  audio: true  
});
```

```
promise.then((avStream) => {  
  // Find my video tag...  
  video = document.createElement('video');  
  
  attachMediaStream(video, stream);  
  video.play();  
  
  // Add video tag to DOM  
  videoContainer.append(v);  
}).catch(() => {...});
```



# 3. Local Peer Connection

```
<html>
  <head>...</head>
  <body>
    ...
    <h2>Person 1</h2>
    <video id="person_1"></video>
    <button id="Start Call">Call</button>

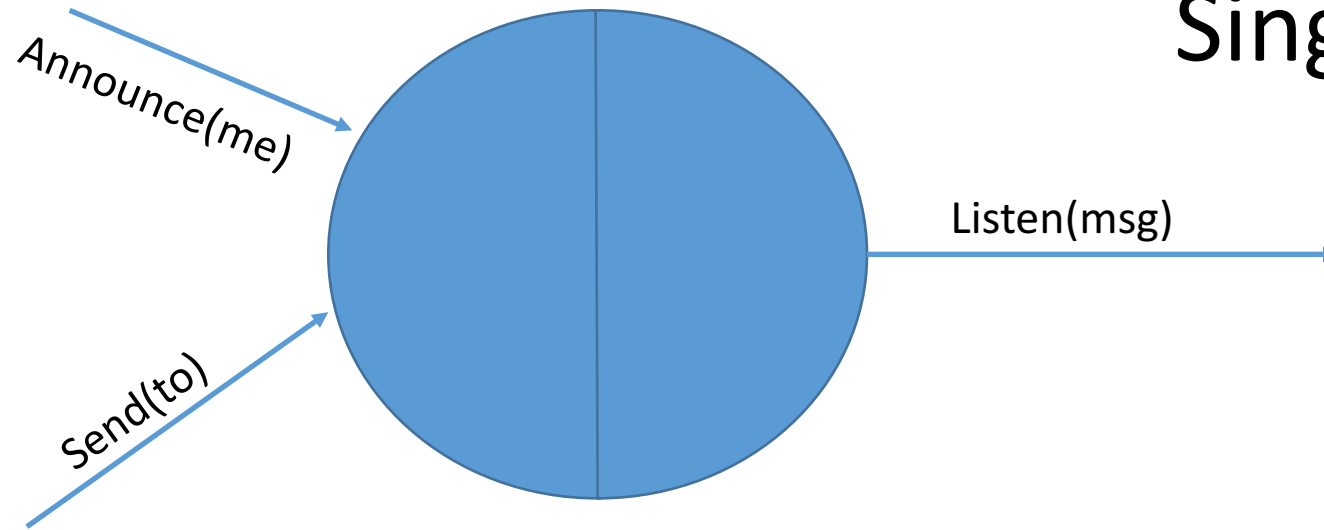
    <h2>Person 2</h2>
    <video id="person_2"></video>

  </body>
</html>
```

- Connecting camera/mic to a local video tag THROUGH a peer connector
- Implement our own local signaling
- Shows the basic structure of how to connect streams to each other remotely without network complexity

### 3. Local signaling

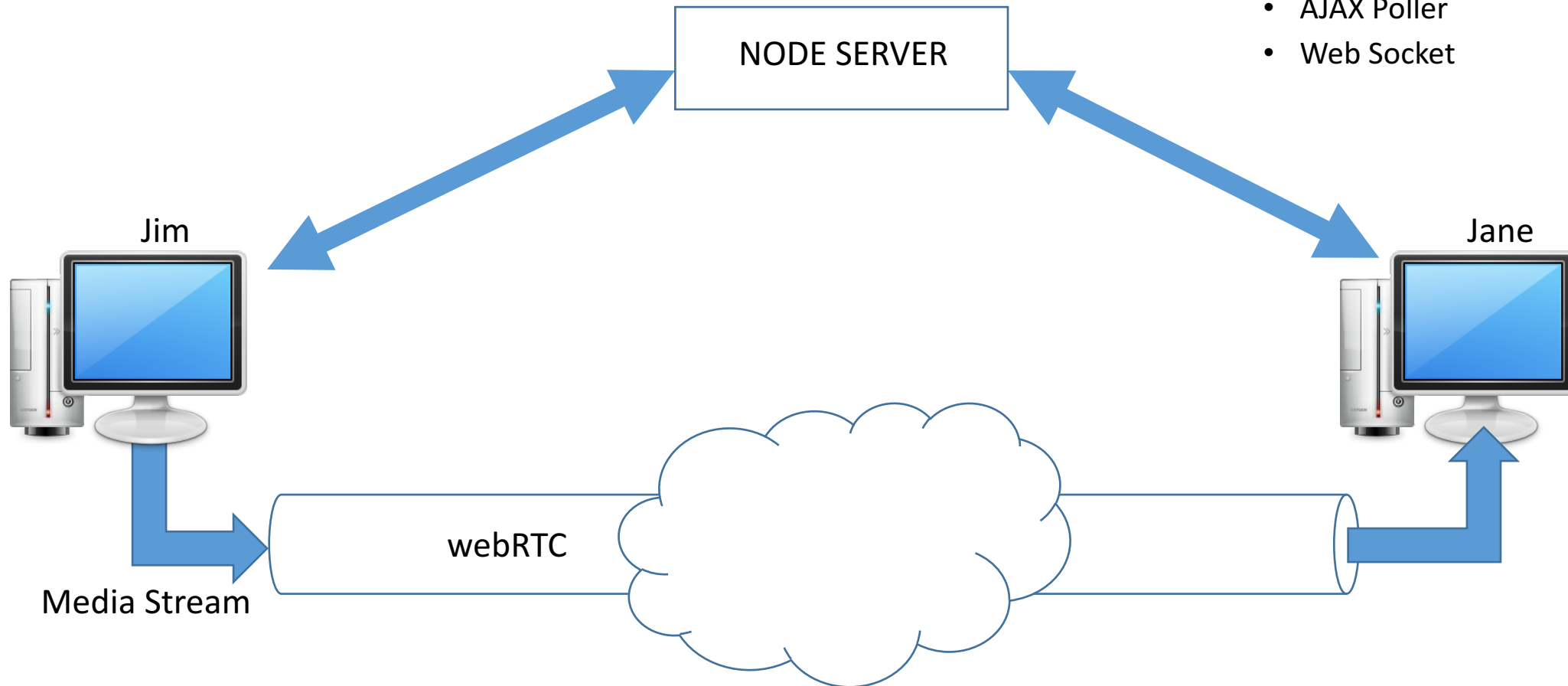
- Create a 'signaling' abstraction:
  - Announce(me), send(to), listen
  - Completely local



Single web page!

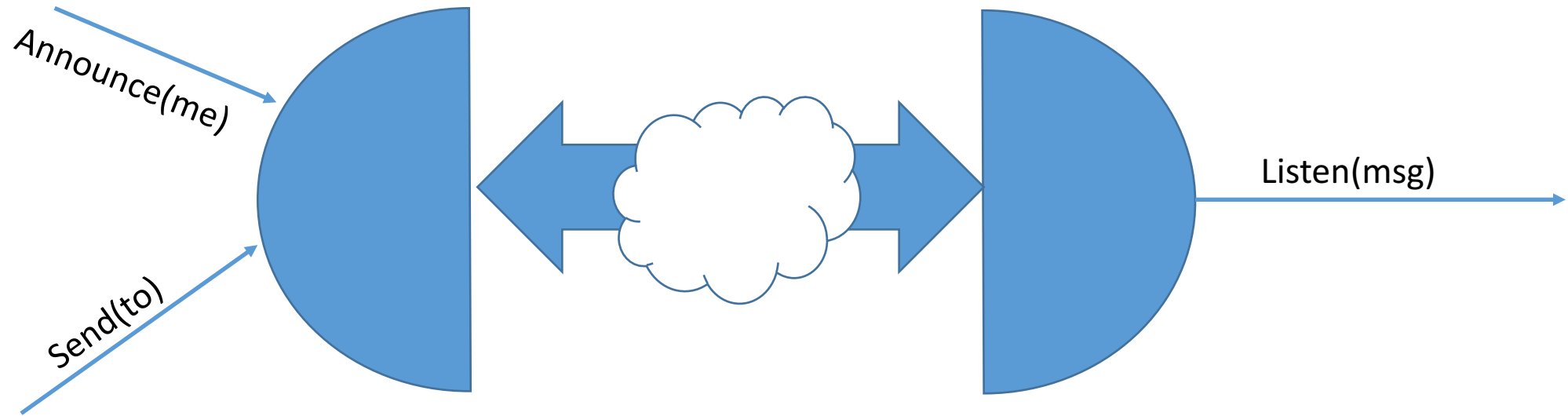
## 4. Remote Peer Connections

- Replace local signaling with node server proxy.
- Options for transferring signalling:
  - AJAX Poller
  - Web Socket

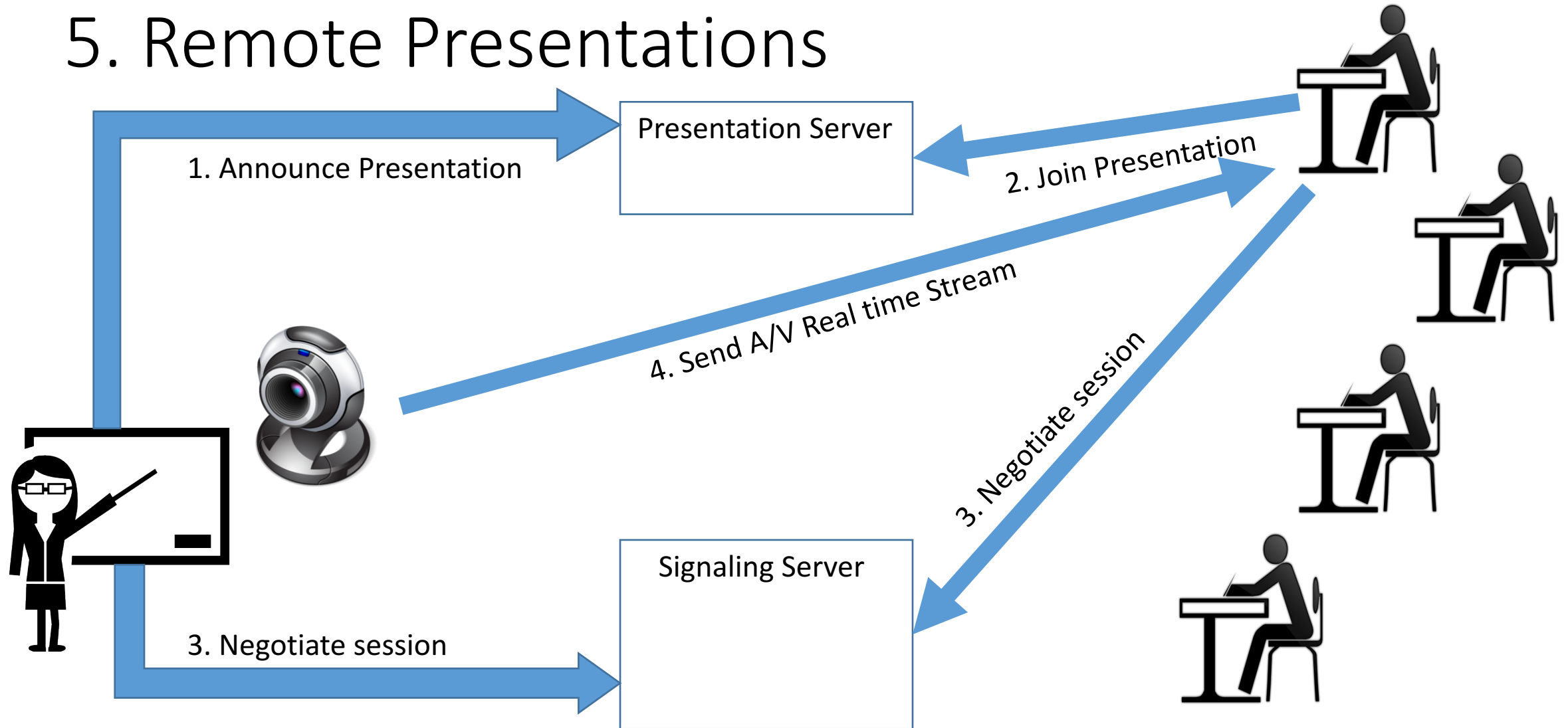


## 4. Signaling across a network

- Split your signaling into two parts:
  - Carry information across the local network
  - Modify the application to have one end point per browser



# 5. Remote Presentations



Signaling: EITHER using IpCortex api OR modified signaling from previous task

# Signaling for Remote Presentations: Two Options

- Evolution of simple signaling from previous example
  - Should work on a local LAN
  - Won't work across the Internet without TURN/STUN servers (complexity)
- IPCortex API
  - Covers all the routing across the Internet
  - More complex to configure/run

# References

- MDN WebRTC
  - [https://developer.mozilla.org/en-US/docs/Web/API/WebRTC\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API)
- WebRTC.org – Getting Started
  - <https://webrtc.org/start/>
- HTML 5 Rocks – Getting started with WebRTC (2012)
  - Good overview
  - Illustrates local signaling *but* not is a portable way
  - <https://www.html5rocks.com/en/tutorials/webrtc/basics/>
- HTML 5 Rocks – WebRTC Infrastructure
  - Great overview of signalling – everything you need to know!
  - <https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/>
- adapter.js
  - Shim to isolate applications from browser incompatibilities
  - <https://github.com/webrtc/adapter>