
Google
Developers

WebRTC

Real-time Audio/Video and P2P in HTML5

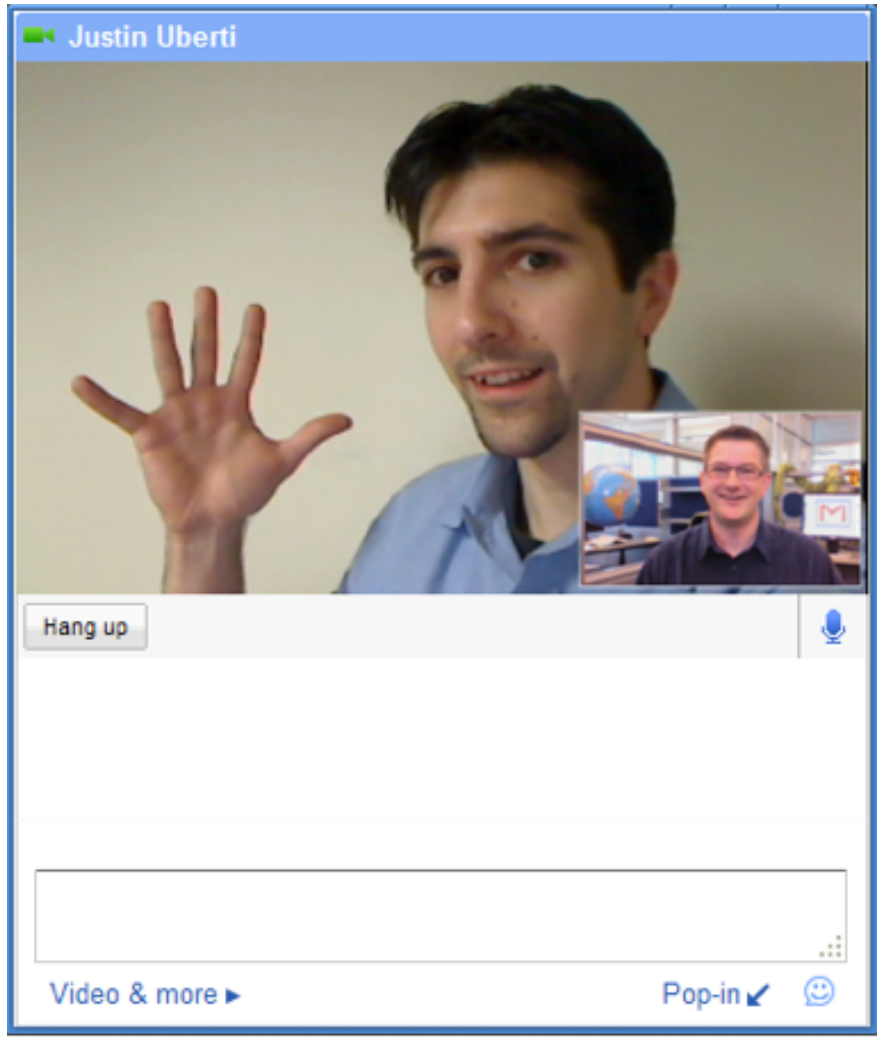
Watch this presentation on YouTube

Justin Uberti
Tech Lead, WebRTC
#io12 #webrtc

Google 12

About Me

- Tech Lead on Chrome WebRTC team
- 10 years of audio/video communications experience
- Co-creator of:
  - Google+ Hangouts
  - Google Video Chat
  - Gmail Call Phone



Audience

How many of you are familiar with:

- WebRTC
- HTML5
- WebSockets
- AppEngine
- SIP
- H.323

What is WebRTC?

- **Real Time Communications** meets the web
- A **state-of-the-art** audio/video communication stack in your web browser
- A cross-industry effort to create a **new communications platform**

WebRTC Vision

- The communications industry, at web speed
- To allow real-time communication to be a feature of apps
- To create a common platform for real-time communication - so that your PC, your phone, your TV can all communicate

“WebRTC and HTML5 could enable the same transformation for real time that the original browser did for information.”

Phil Edholm
NoJitter

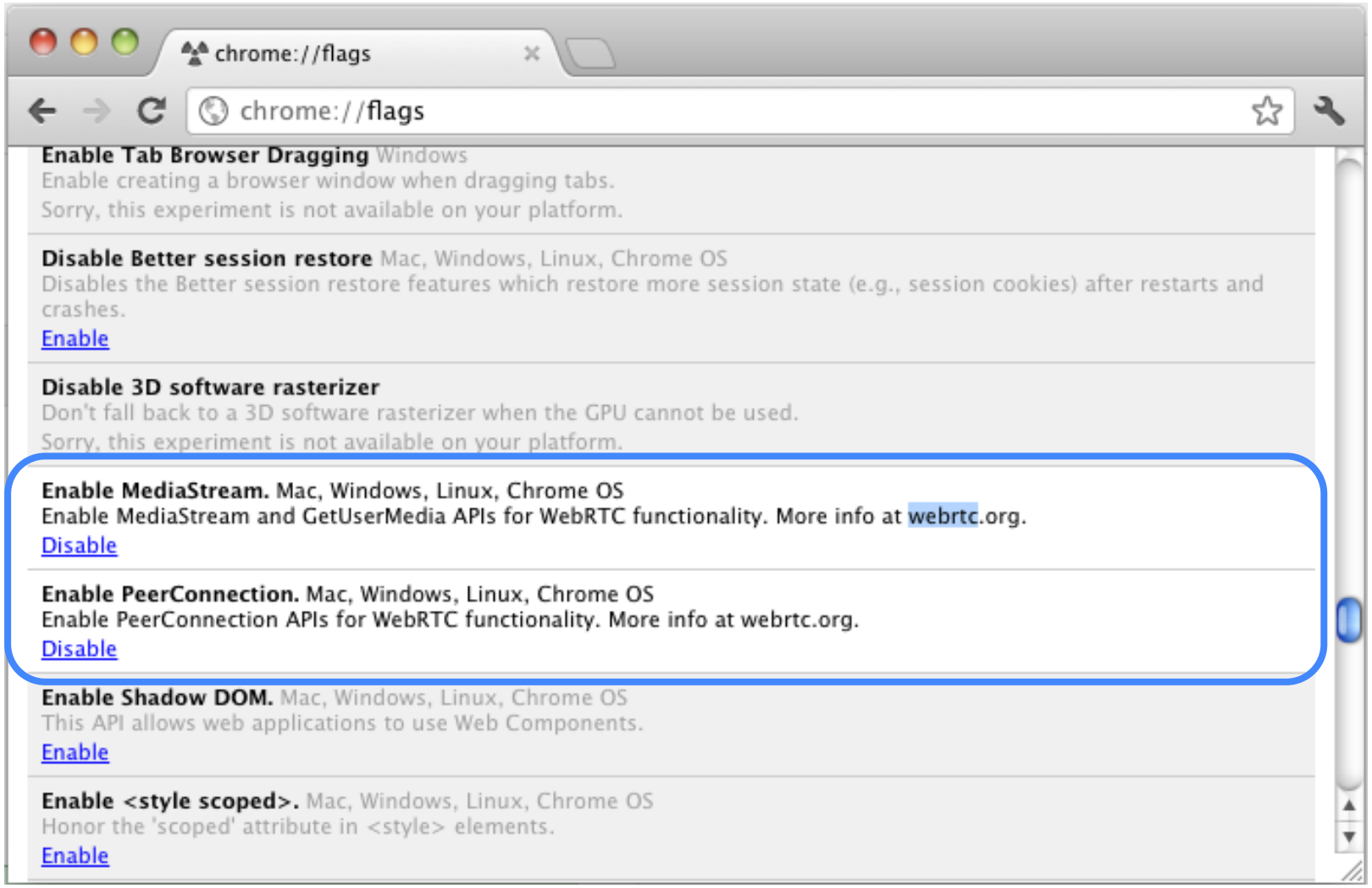
WebRTC Support

- WebRTC coming to **almost all desktop browsers** by EOY 2012
  - Chrome 21
  - Opera 12
  - Firefox 17
  - IE (via ChromeFrame)
- **Mobile browser** support will follow
- **Native C++ versions** of WebRTC stack also available

Key Features

- **MediaStreams** - access to the user's camera and mic
- **PeerConnection** - easy audio/video calls
- **DataChannels** - p2p application data transfer

Chrome Flags



MediaStreams

Getting access to audio and video

MediaStreams

- A **MediaStream** represents a media source, containing one or more synchronized **MediaStreamTracks**
- A **MediaStream** can be converted to an **object URL**, and passed to a **<video/> element**.
- Use the **getUserMedia** API to get a **MediaStream** for the webcam/mic (prompts user for consent)

getUserMedia Sample

http://webrtc-demos.appspot.com/html/gum1.html

```
<script type='text/javascript'>
navigator.webkitGetUserMedia({video:true}, onGotStream, onFailedStream);

onGotStream = function(stream) {
  var url = webkitURL.createObjectURL(stream);
  video.src = url;
}
</script>
<video id="video" autoplay="autoplay" />
```

getUserMedia + <canvas/>

http://webrtc-demos.appspot.com/html/gum2.html

```
<script type='text/javascript'>
function onClick() {
  snapshot.getContext("2d").drawImage(video, 0, 0, canvas.width, canvas.height);
}
</script>
<video id="video" autoplay="autoplay"/>
<canvas id="canvas"></canvas>
<button onclick="onClick()">Snapshot</button>
```

getUserMedia + CSS

http://webrtc-demos.appspot.com/html/gum3.html

```
<style>
.grayscale {
  -webkit-filter: grayscale();
}
</style>
<script type='text/javascript'>
function onClick() {
  video.classList.add("grayscale");
}
</script>
```

Webcam Toy Demo

http://www.neave.com/webcam/html5

PeerConnection

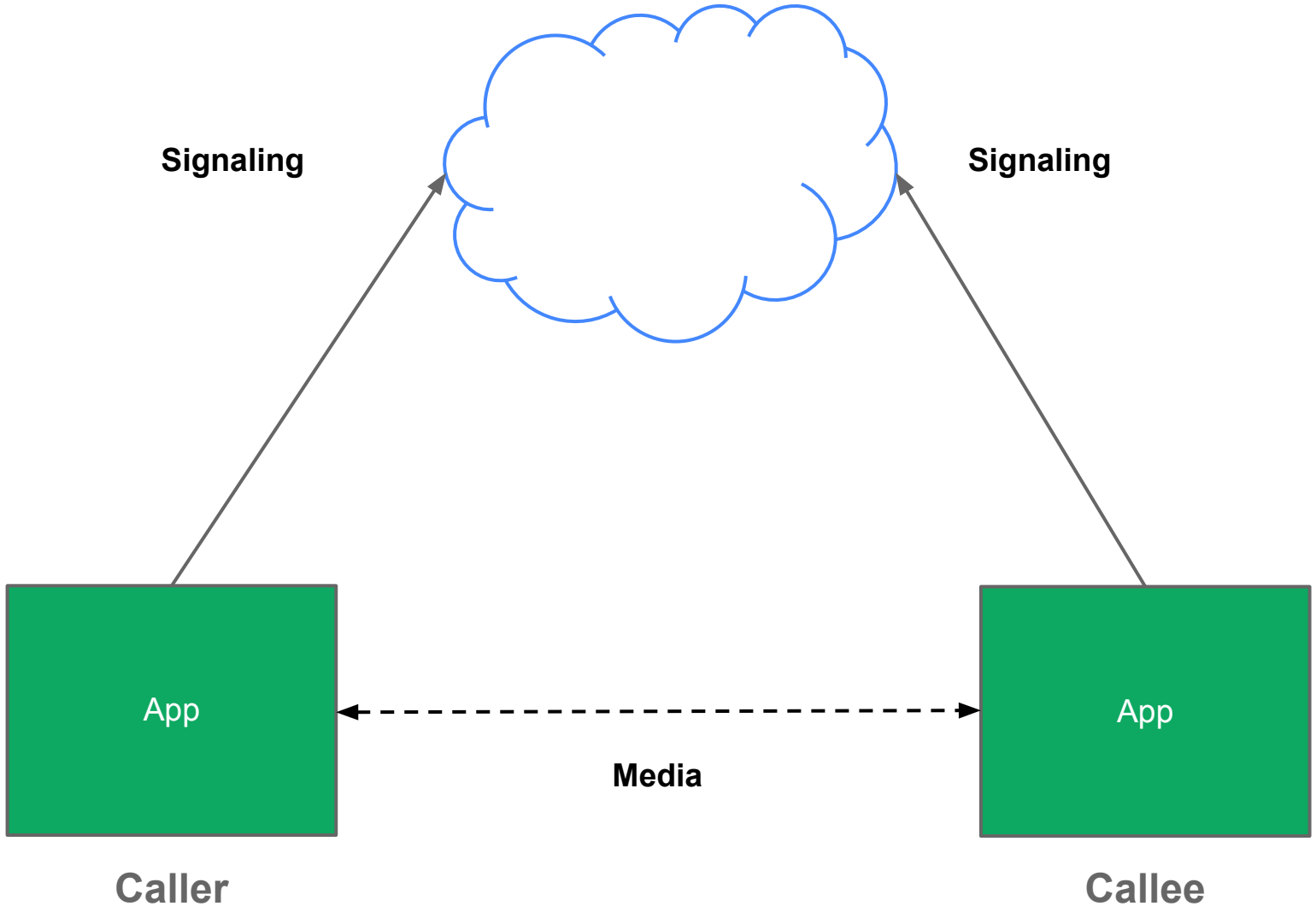
Live audio/video sessions

PeerConnection Basics

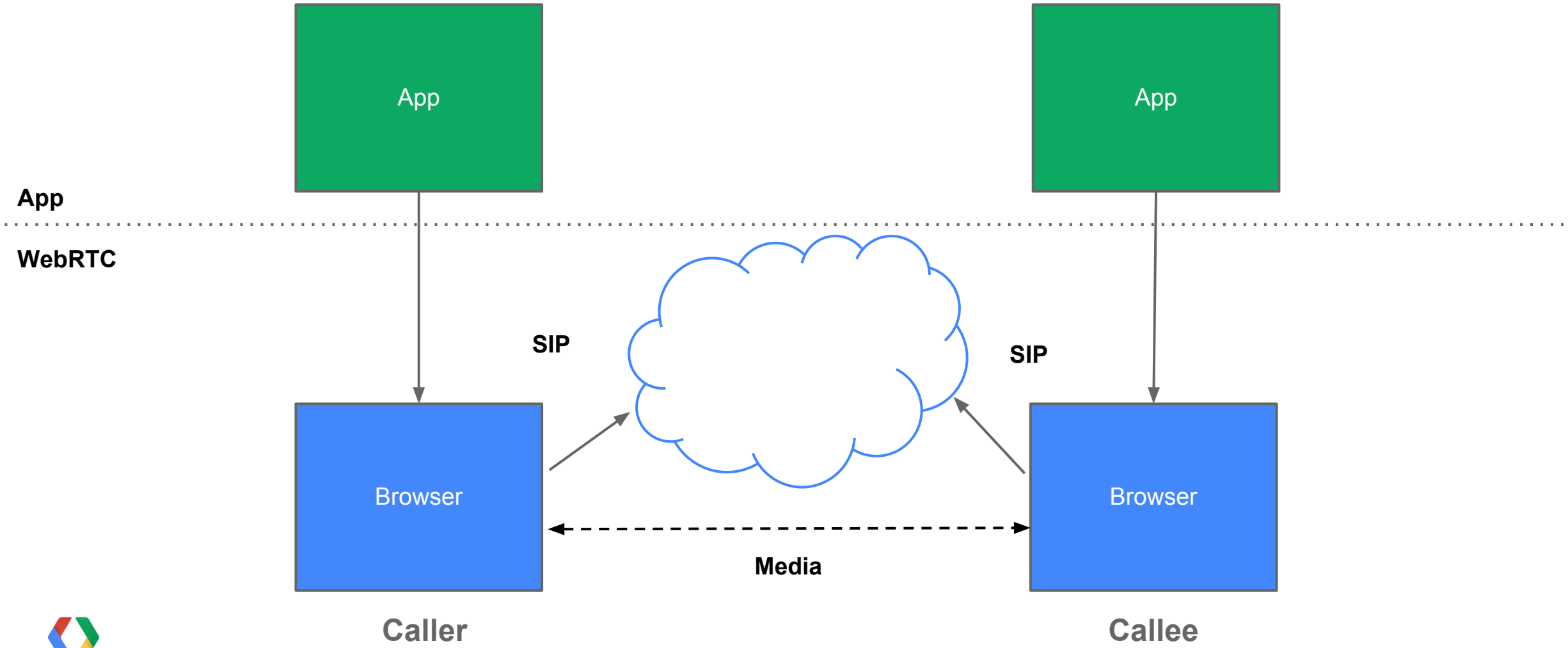
- API for establishing audio/video calls ("sessions")

- Built-in:
  - Peer-to-peer
  - Codec control
  - Encryption
  - Bandwidth Management

Desktop Client Architecture

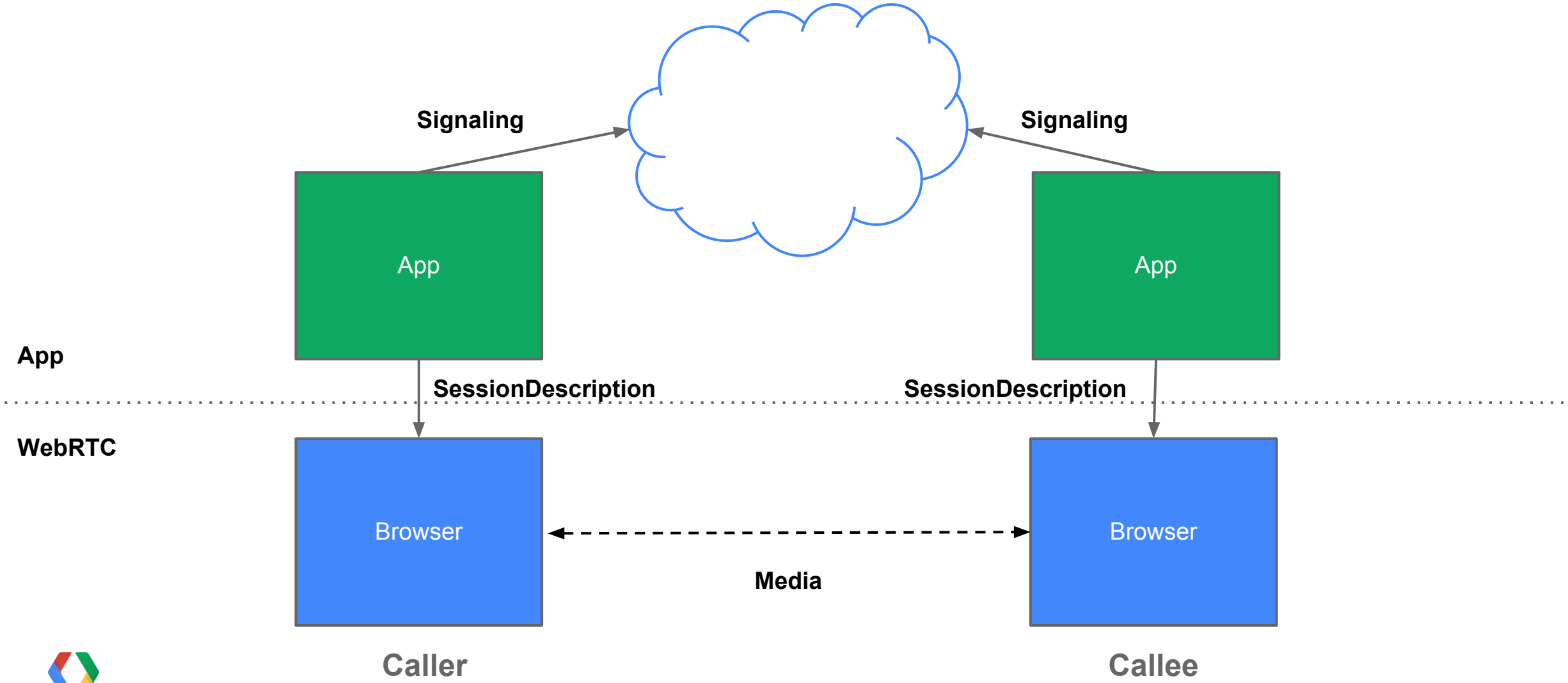


WebRTC: Approach Not Taken



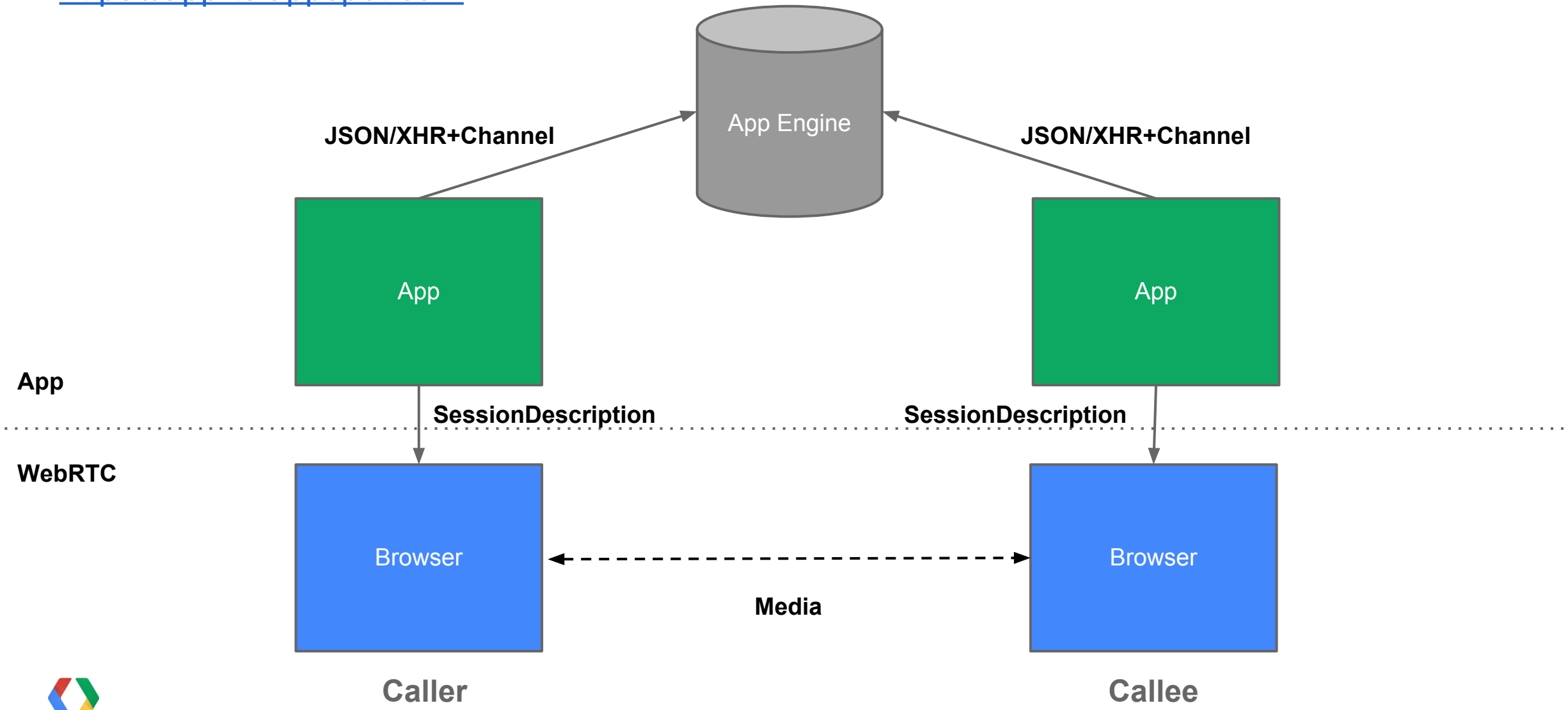
WebRTC: JSEP Approach

Signaling vs Media



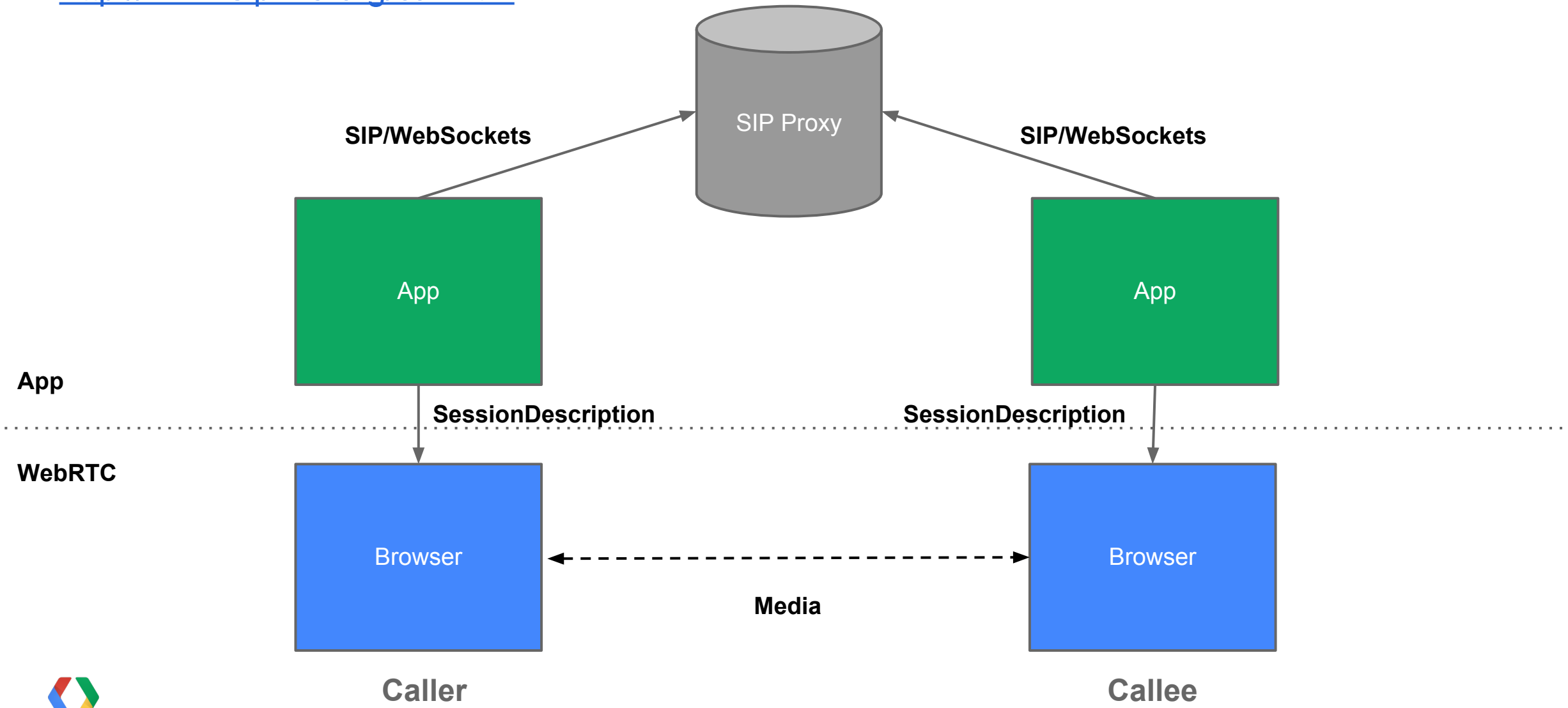
App Engine Example

https://apprtc.appspot.com



SIP Example

http://www.sipml5.org/call.htm



Setting Up a Session

To start a session, a client needs:

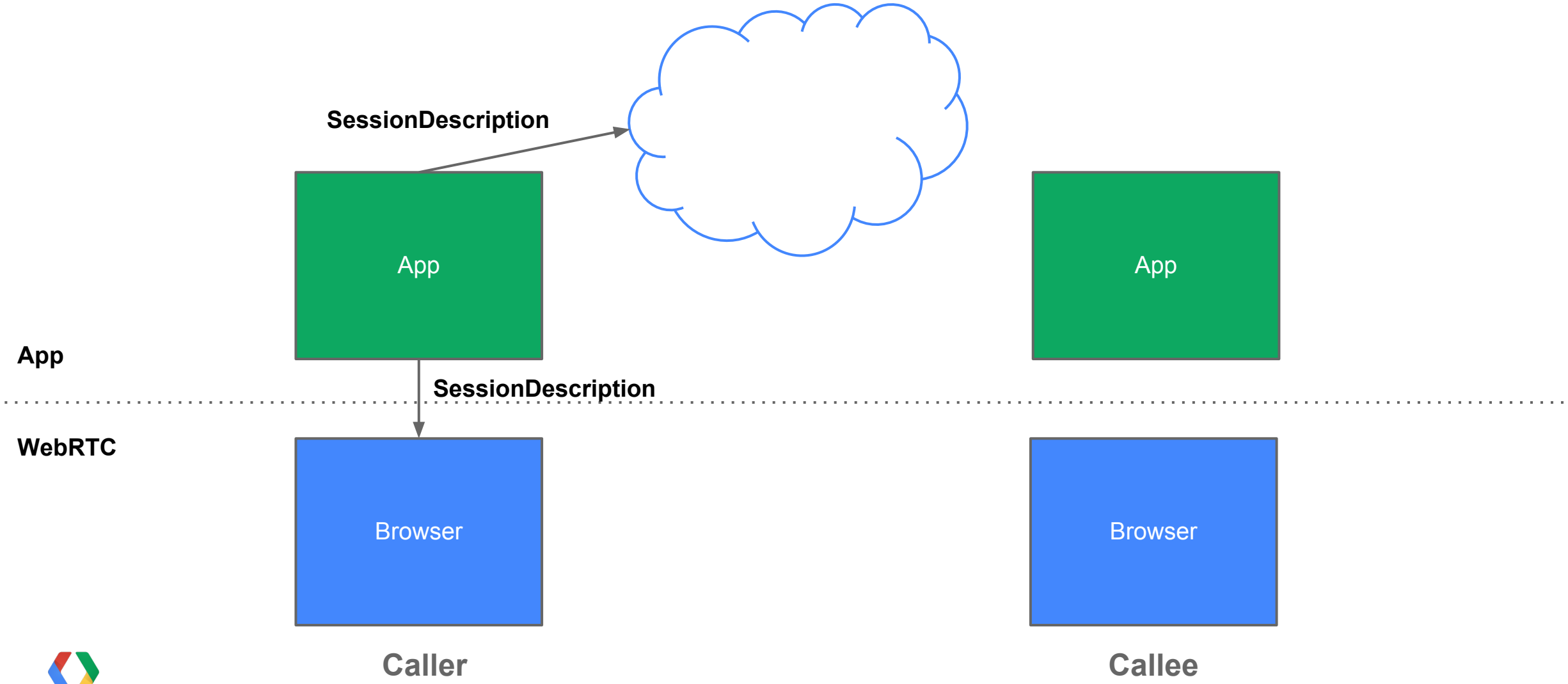
- **Local Session Description** (describes the configuration of the local side)
- **Remote Session Description** (describes the configuration of the remote side)
- **Remote Transport Candidates** (describes how to connect to the remote side)

These parameters are exchanged via signaling, and communicated to the browser via the **PeerConnection** API.

The initial session description sent by the caller is called an **offer**, and the response from the callee is called an **answer**.

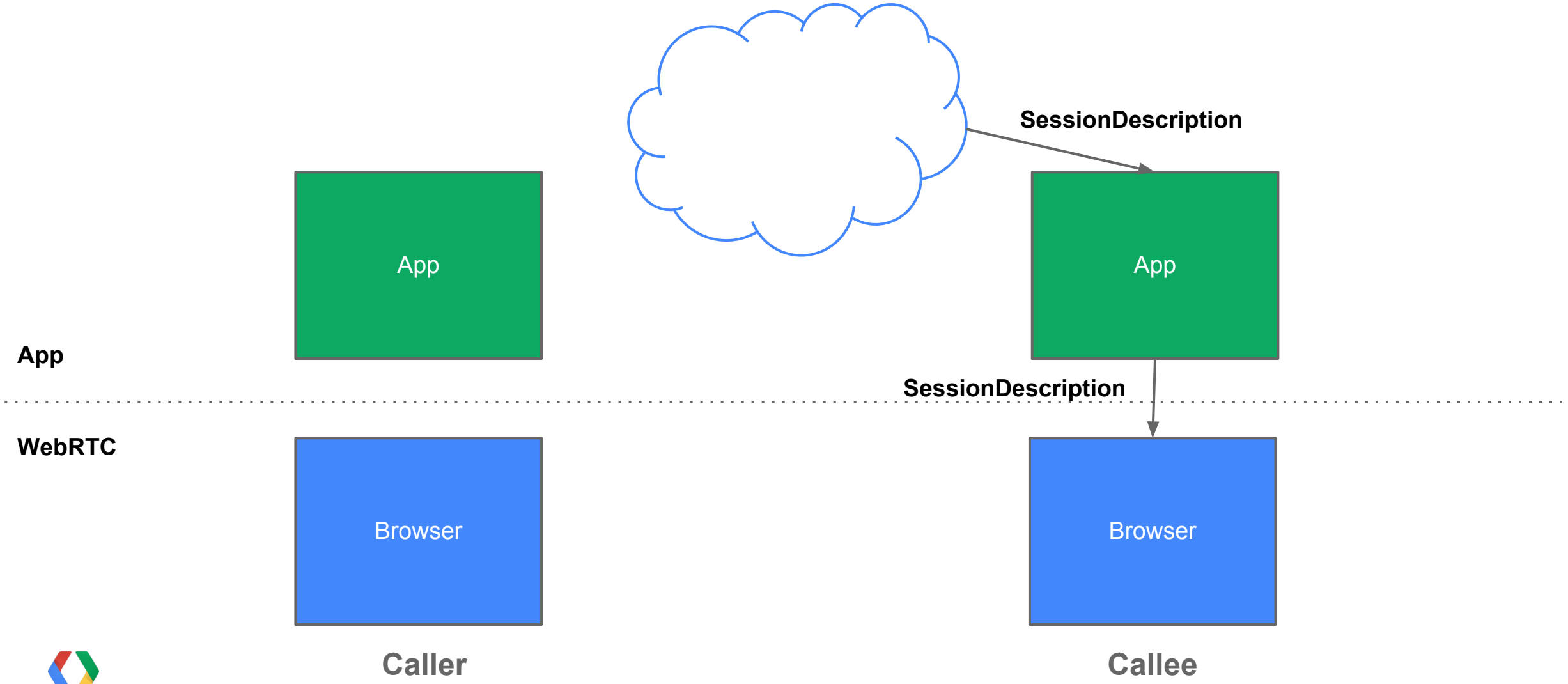
Signaling Flow: #1

Caller Sends Offer



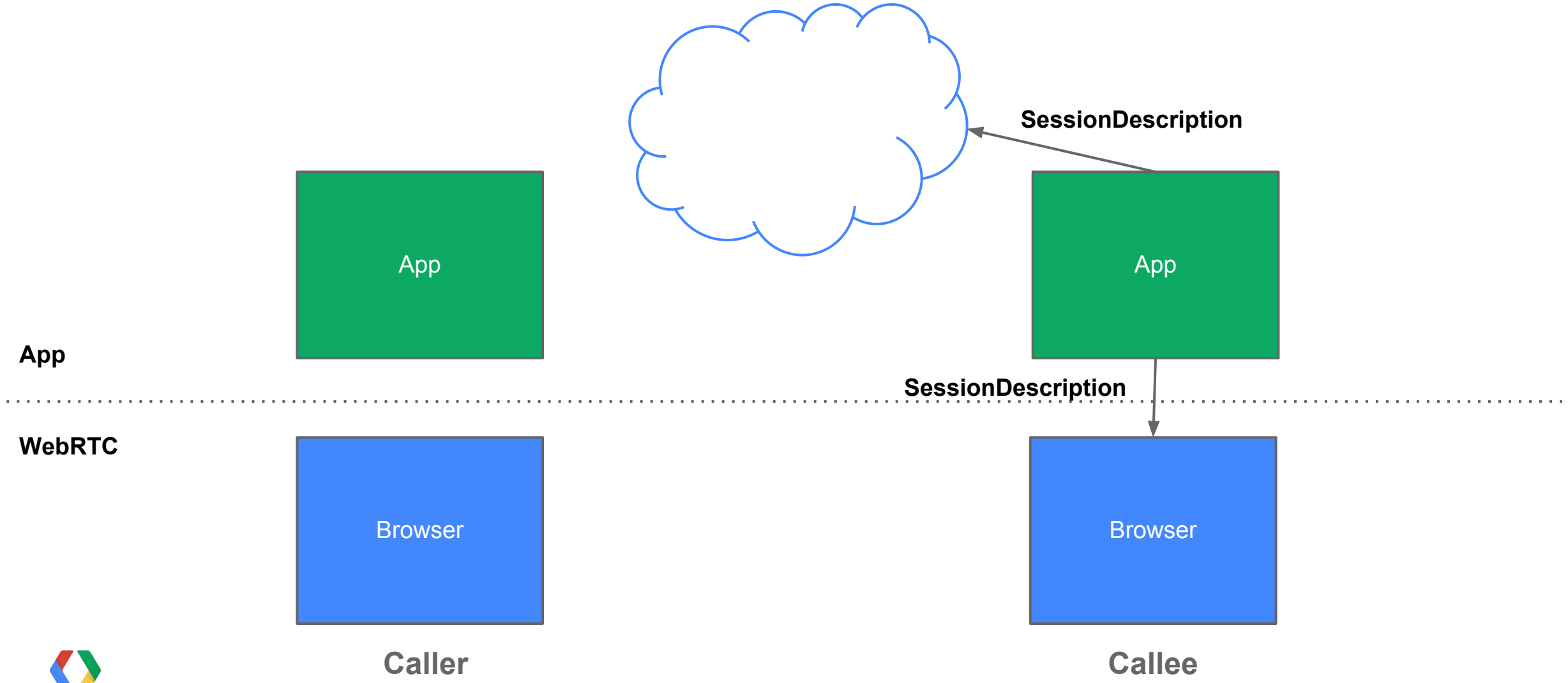
Signaling Flow: #2

Callee Receives Offer



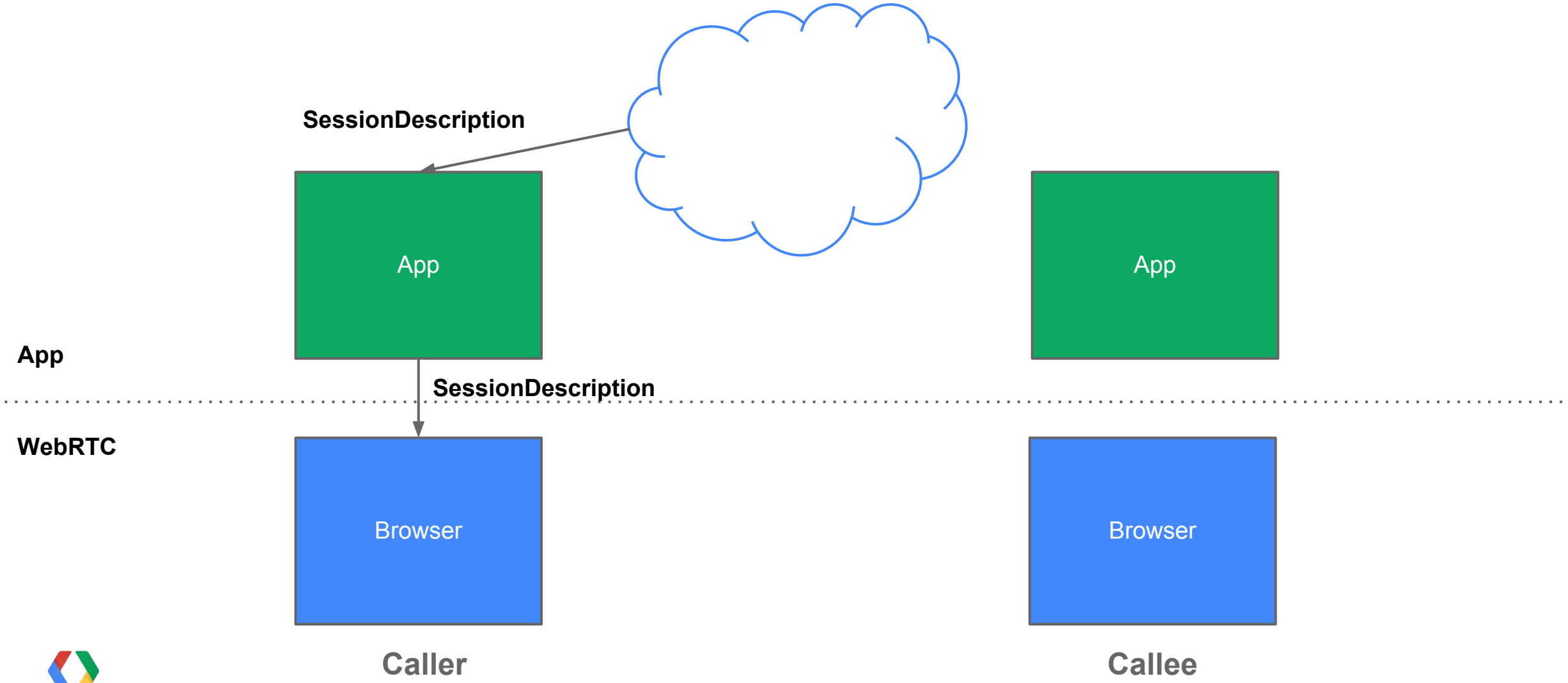
Signaling Flow: #3

Callee Sends Answer



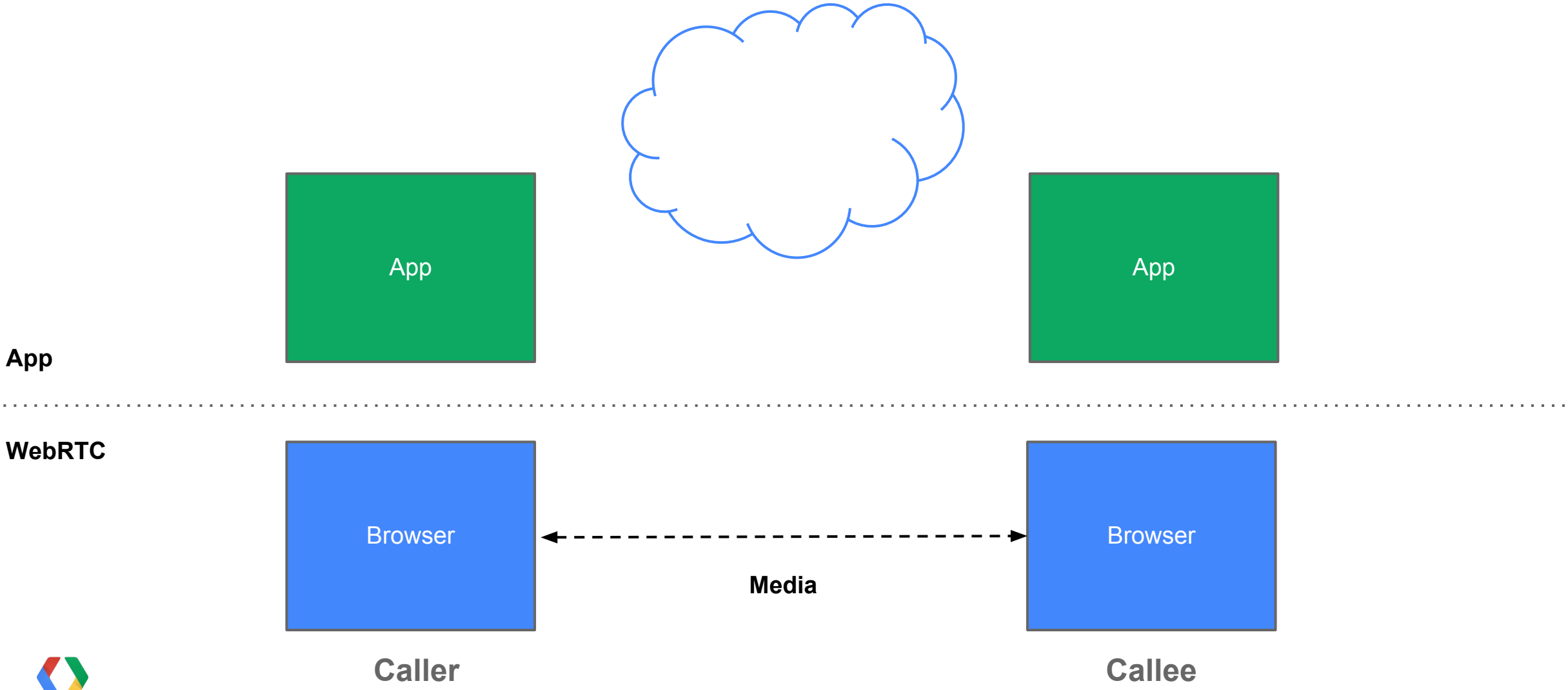
Signaling Flow: #4

Caller Receives Answer



Signaling Flow: #5

Media Flows



PeerConnection API

Caller side

Create a new PeerConnection and add stream:
PeerConnection(config, iceCallback)
addStream(stream)

Create local SessionDescription and apply it:
createOffer(hints)
setLocalDescription(type, desc)
startIce()

<wait for response from callee>

Receive remote SessionDescription and use it:
setRemoteDescription(type, desc)

Callee side

<receive call from caller>

Create PeerConnection and set description:
PeerConnection(config, iceCallback)
setRemoteDescription(type, desc)

Create local SessionDescription and apply it:
createAnswer(offer, hints)
setLocalDescription(type, desc)
startIce()

PeerConnection Sample

http://webrtc-demos.appspot.com/html/pc1.html

```
<script type='text/javascript'>
  pc1 = new webkitPeerConnection00(null, onIceCandidate1);  // create the "sending" PeerConnection
  pc2 = new webkitPeerConnection00(null, onIceCandidate2);  // create the "receiving" PeerConnection
  pc2.onaddstream = onRemoteStream;
  pc1.addStream(localStream);                                  // add the local stream to the sending PeerConnection
  var offer = pc1.createOffer(null);                           // create an offer, with the local stream
  pc1.setLocalDescription(pc1.SDP_OFFER, offer);               // set it on the sending and receiving PeerConnection
  pc2.setRemoteDescription(pc2.SDP_OFFER, offer);
  var answer = pc2.createAnswer(offer.toSdp(), null);          // create an answer
  pc2.setLocalDescription(pc2.SDP_ANSWER, answer);         // set it on the receiving and sending PeerConnection
  pc1.setRemoteDescription(pc1.SDP_ANSWER, answer);
  pc1.startIce();   pc2.startIce();                            // start the connection process
</script>
```

Backend Services

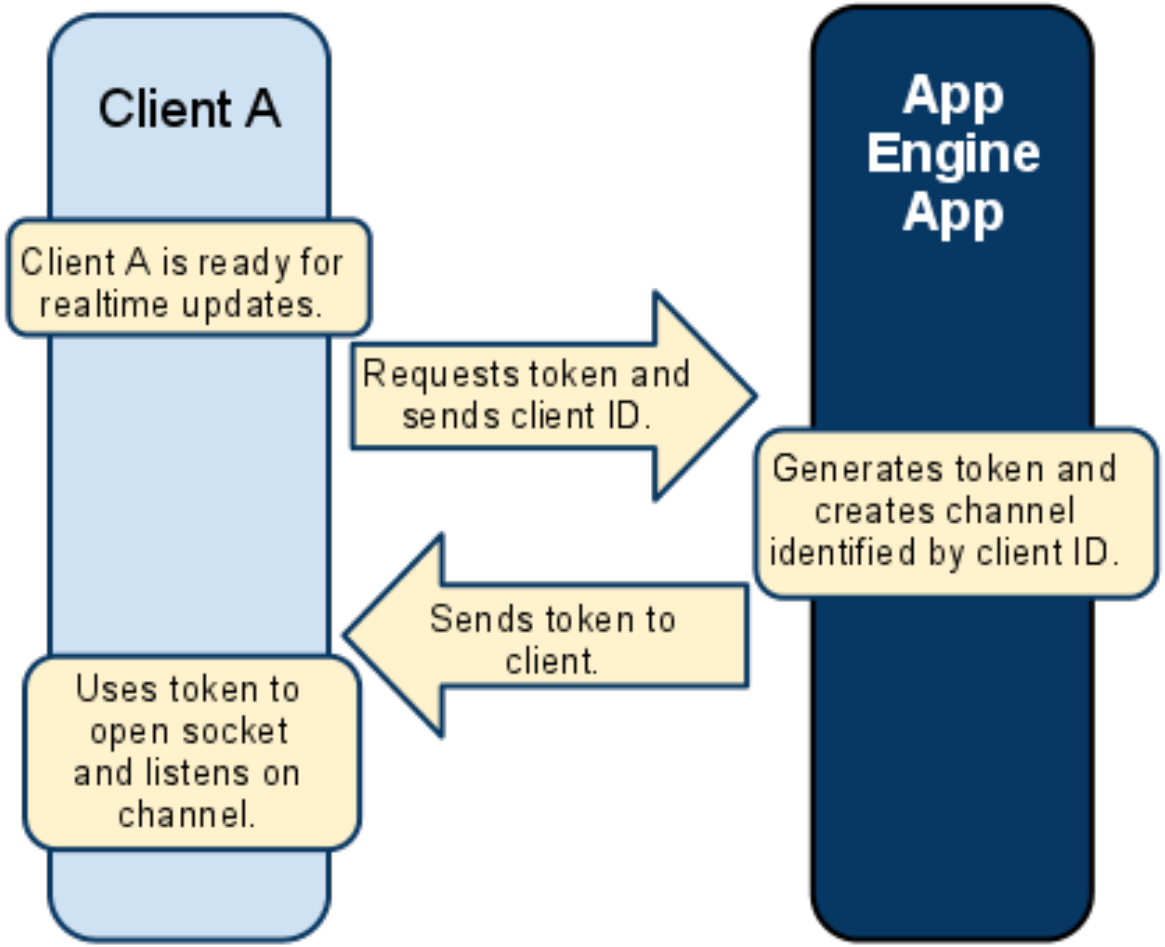
Connecting WebRTC endpoints

WebRTC Signaling Channel

- XMLHttpRequest works great for sending requests, but receiving them isn't as easy.
- This is a real issue when trying to set up a video call.
- App Engine's **Channel API** provides this server->client messaging path.

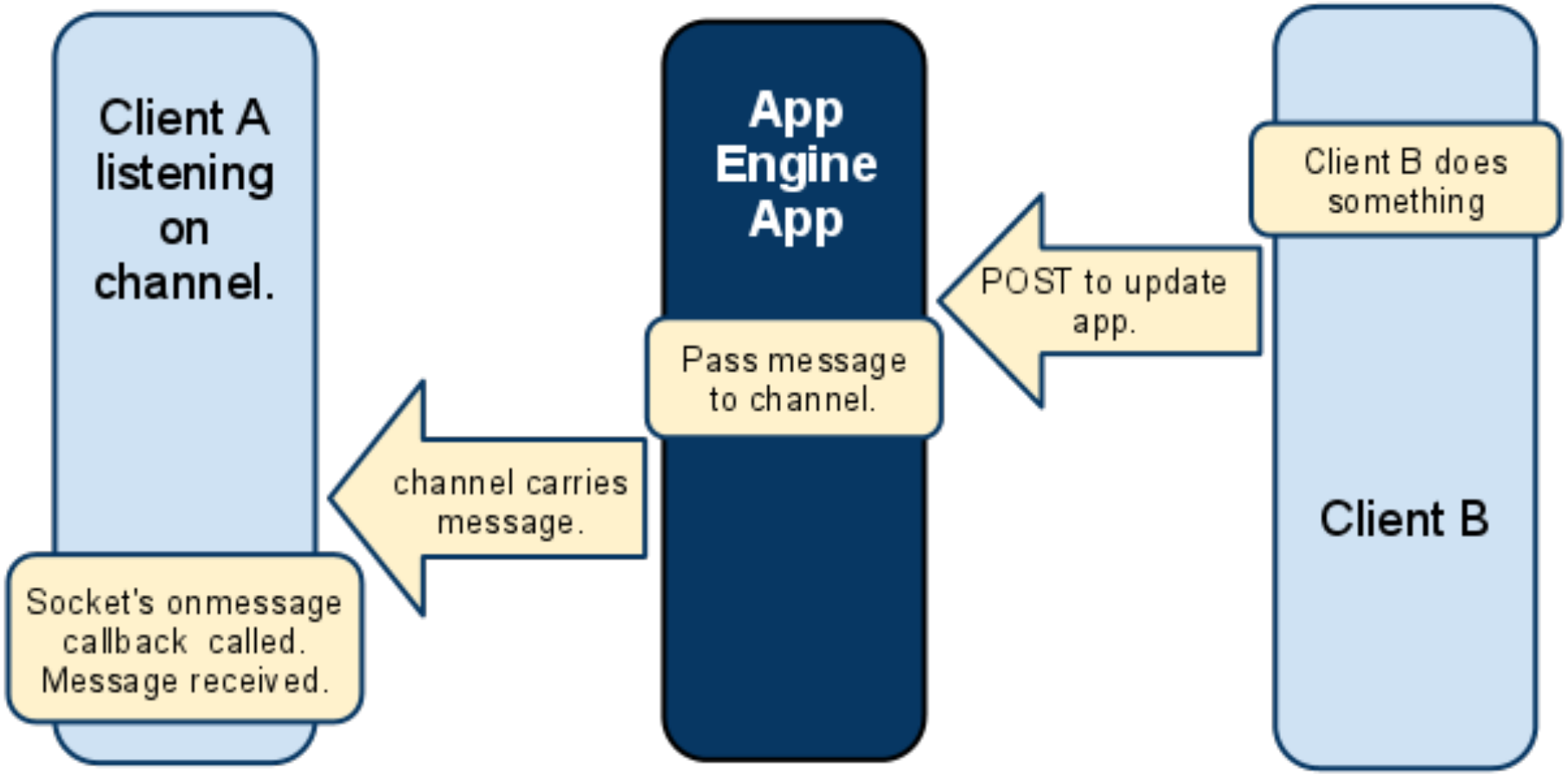
App Engine Channel API

Establishing a Channel



App Engine Channel API

Sending a Message



PeerConnection + Channel API

http://webrtc-demos.appspot.com/html/pc2.html

```
<script type='text/javascript'>
  channel = new goog.appengine.Channel(<token>);
  handler = { "onmessage": onChannelMessage };
  socket = channel.open(handler);

  function onChannelMessage(msg) {
    offer = new SessionDescription(msg.data);
    pc2.setRemoteDescription(pc2.SDP_OFFER, offer);
    answer =  pc2.createAnswer(offer, null);
    pc2.setLocalDescription(pc2.SDP_ANSWER, answer);
    xhr = new XMLHttpRequest();
    xhr.open('POST', path, true);
    xhr.send(answer.toSdp());
  }
```

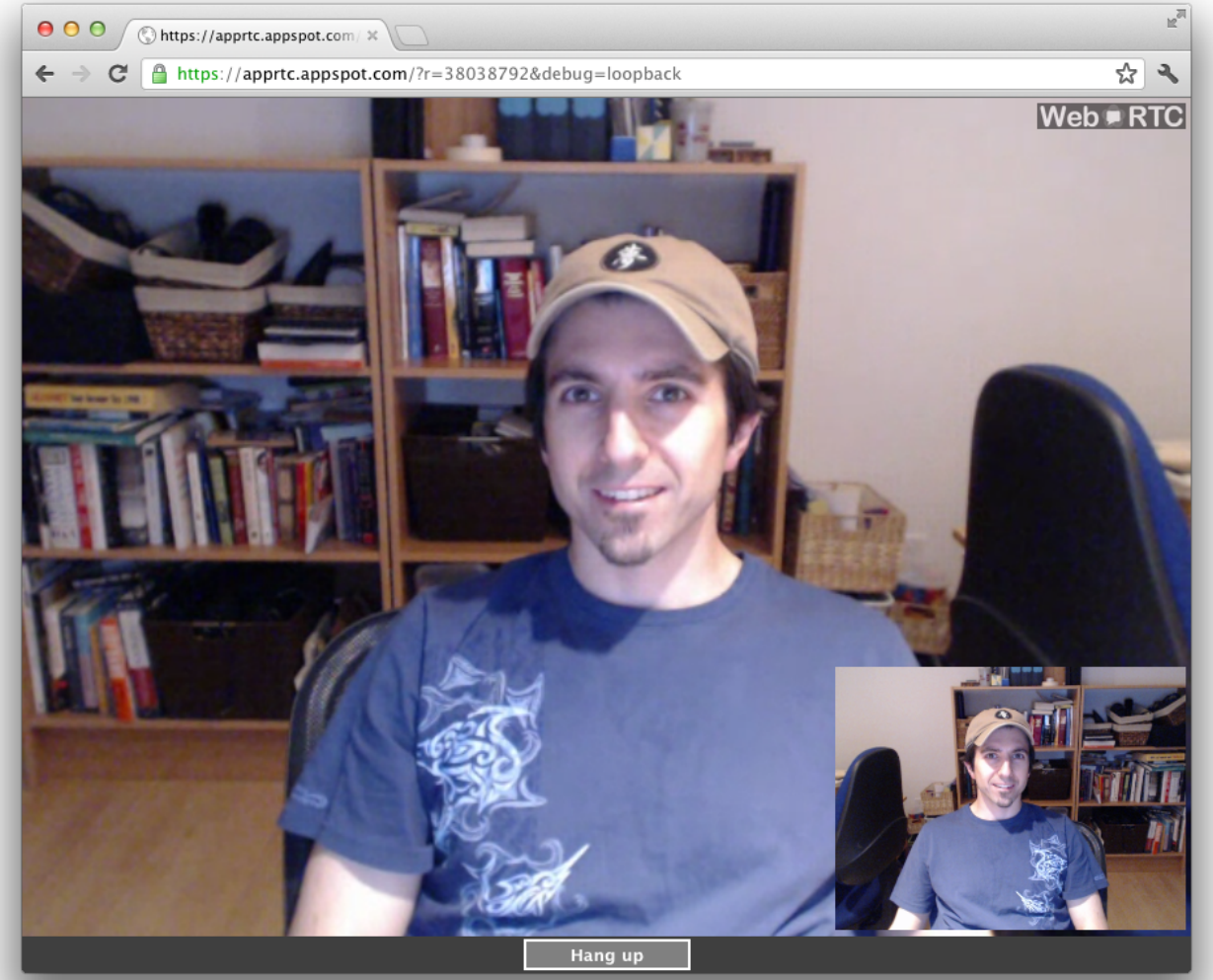
AppRTC

https://apprtc.appspot.com

- Fully functional video calling application
- Built on App Engine infrastructure
  - Channel API
  - Datastore
- Shows how to use WebRTC features
  - PeerConnection API
  - Fast call setup
  - NAT Traversal
  - Encryption

AppRTC Demo

https://apprtc.appspot.com



DataChannels

P2P data in a browser

DataChannel Introduction

Peer-to-peer exchange of arbitrary **application data**

- Low latency
- High message rate/throughput
- Optional unreliable semantics

Many real-world use cases

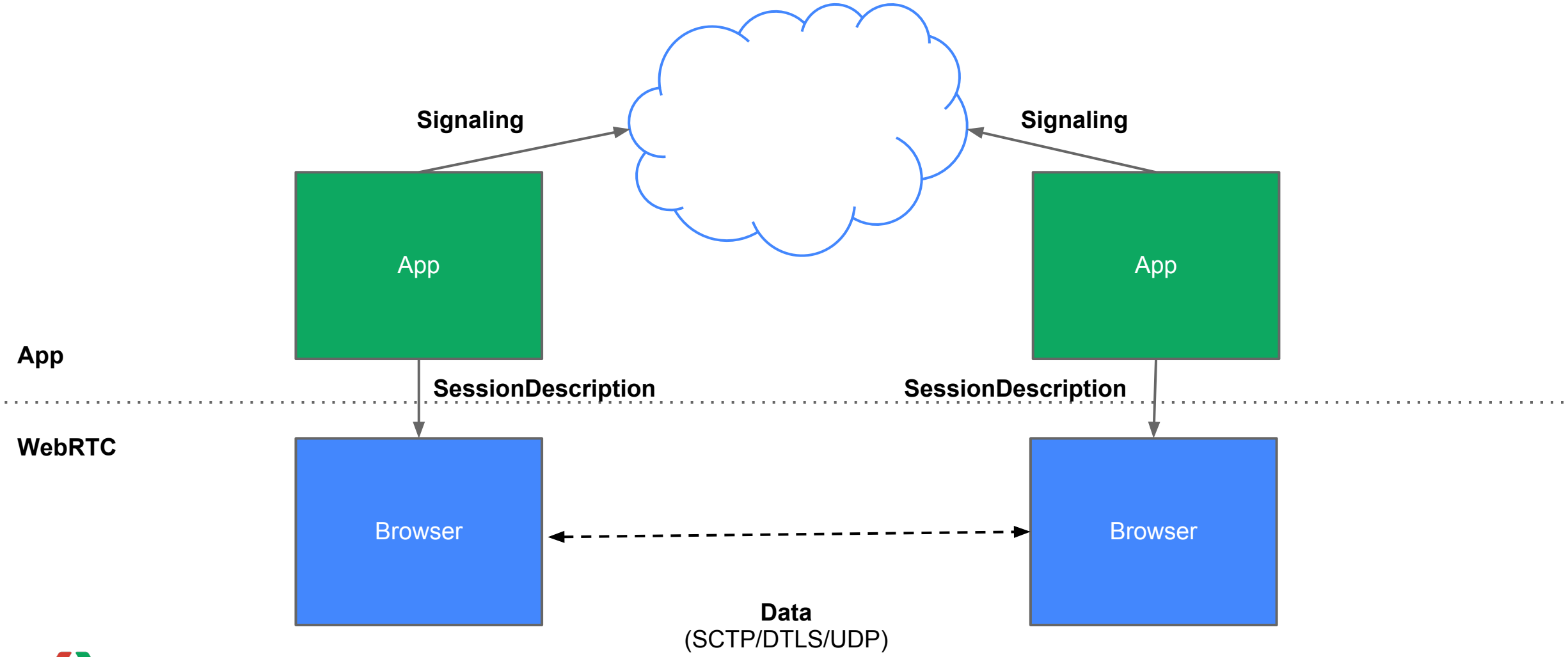
- Gaming
- Remote desktop applications
- Real-time text
- File transfer
- Decentralized networks

DataChannel Key Features

- Leverages PeerConnection session setup
- **Multiple simultaneous channels**, with prioritization
- **Reliable and unreliable** delivery semantics
- Built-in **security** (DTLS) and **congestion control**
- Can be used with or without audio/video
- Similar API to **WebSockets**
  - send(), onmessage()

DataChannel

Basic Topology



DataChannel Sample

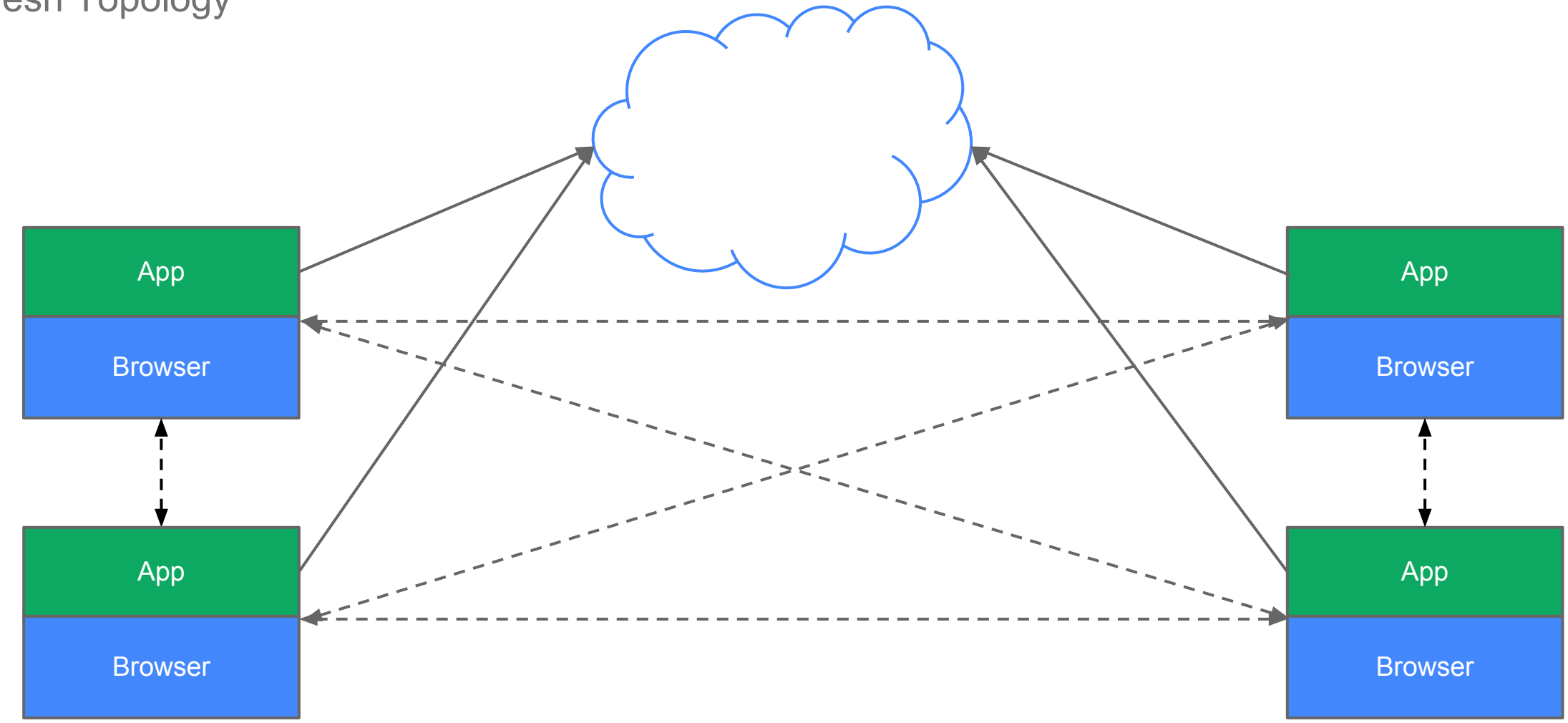
http://webrtc-demos.appspot.com/html/dc1.html

```
<script type='text/javascript'>
  // standard PeerConnection setup and offer-answer exchange omitted
  dc1 = pc1.createDataChannel("mylabel");  // create the sending DataChannel (reliable mode)
  dc2 = pc2.createDataChannel("mylabel");  // create the receiving DataChannel (reliable mode)

  // put received DataChannel messages into a textarea
  dc2.onmessage = function(evt) {
    textarea2.value += evt.data
  };
   // send the contents of the local text field over the DataChannel
  function onSend() {
    dc1.send(input1.value);
  }
</script>
```

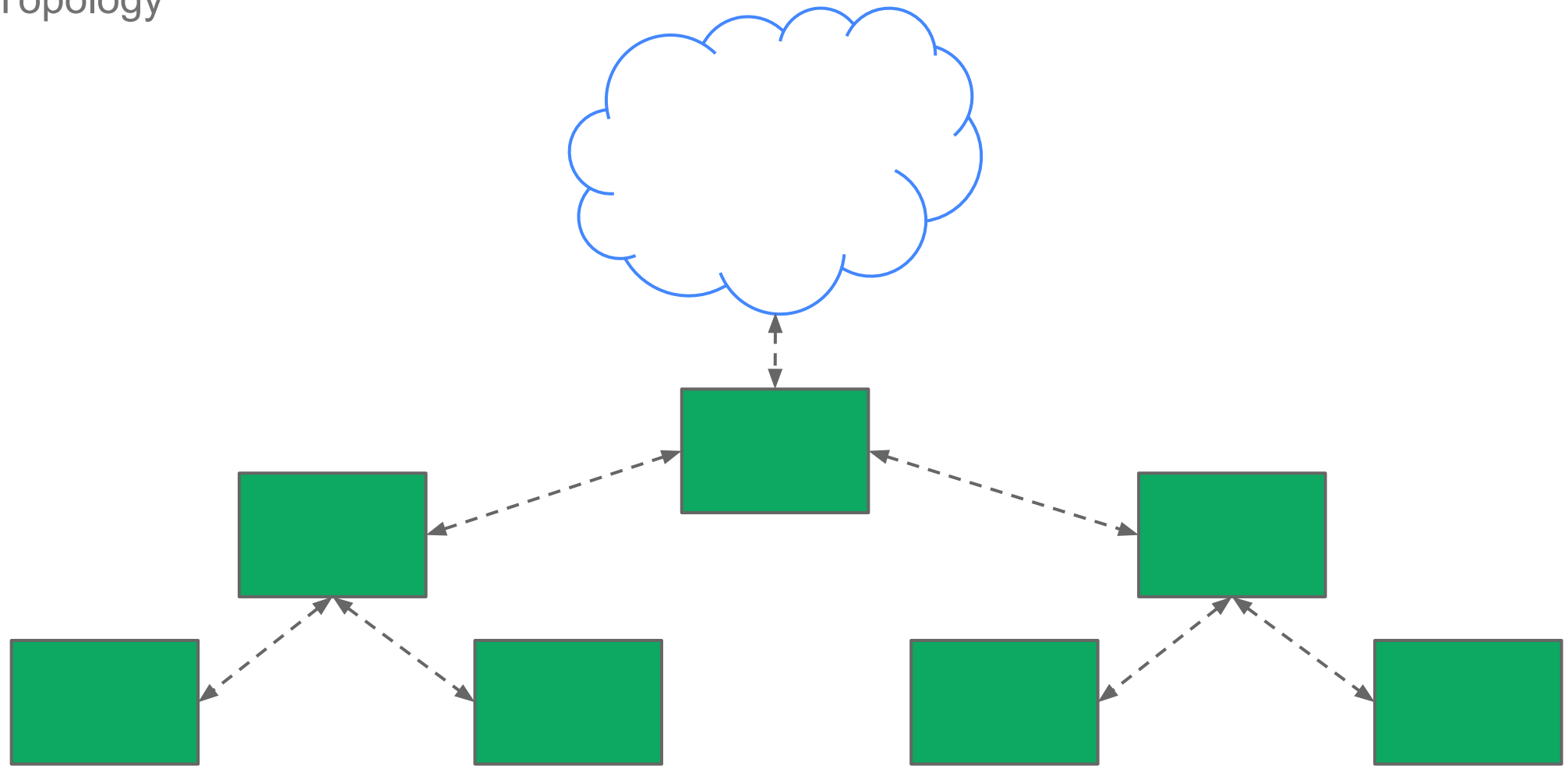
DataChannel

Mesh Topology



DataChannel

DAG Topology



Recap

Summary

- State-of-the-art communications APIs now in the browser
- Fills in a key missing piece in the web platform
- Amazing apps will be built on this

WebRTC Availability

- **Chrome + ChromeFrame**
  - Chrome 21: MediaStreams
  - Chrome 22 (target): PeerConnection
  - Q4 2012: DataChannels

- **Opera**
  - Opera 12: MediaStreams

WebRTC Availability

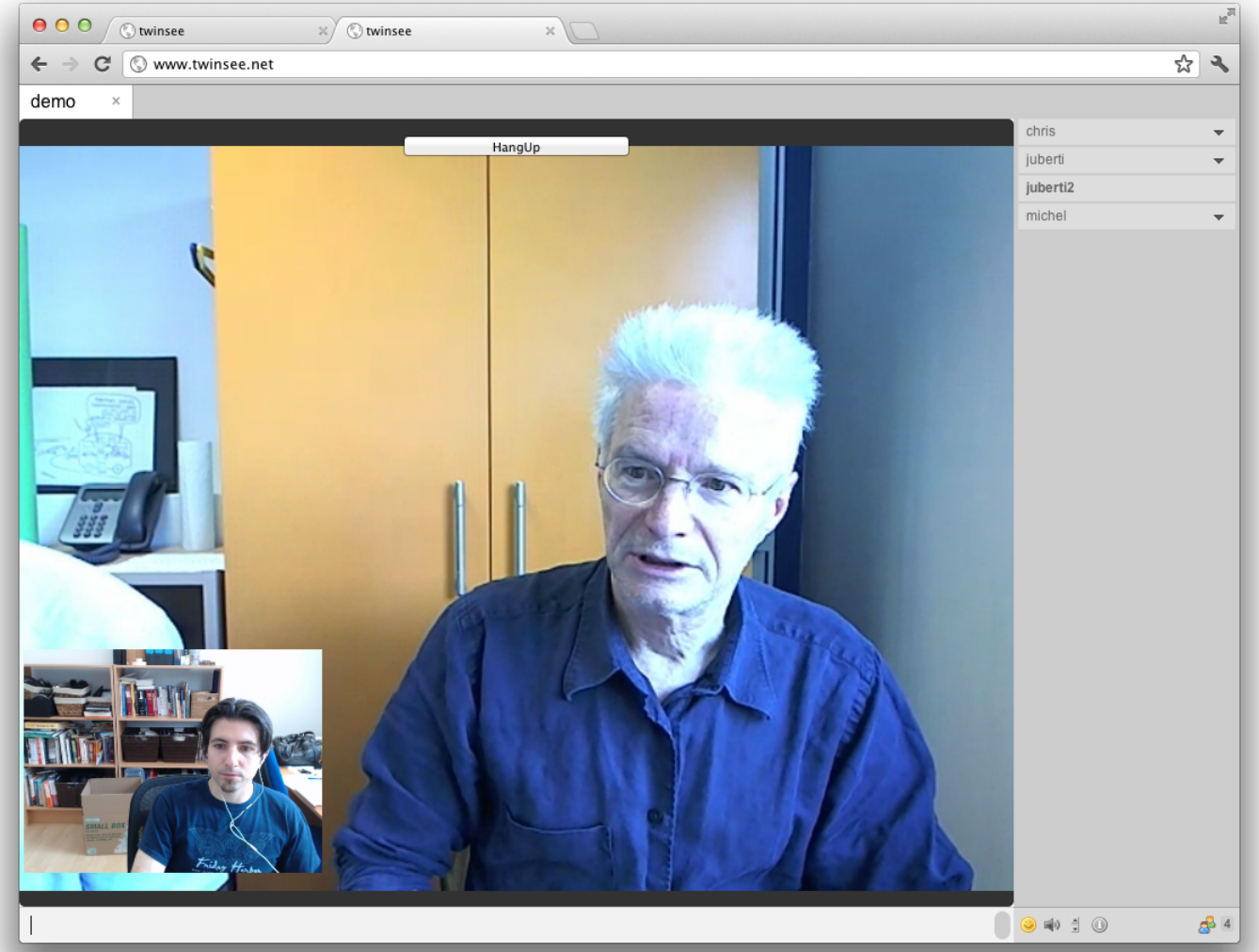
- **Firefox**
  - Early Q4 2012: MediaStreams
  - Late Q4 2012: PeerConnection, DataChannels

- **Other**
  - Internet Explorer support via ChromeFrame
  - Mobile browser support in progress
  - Native C++ implementations also available

WebRTC in ChromeFrame

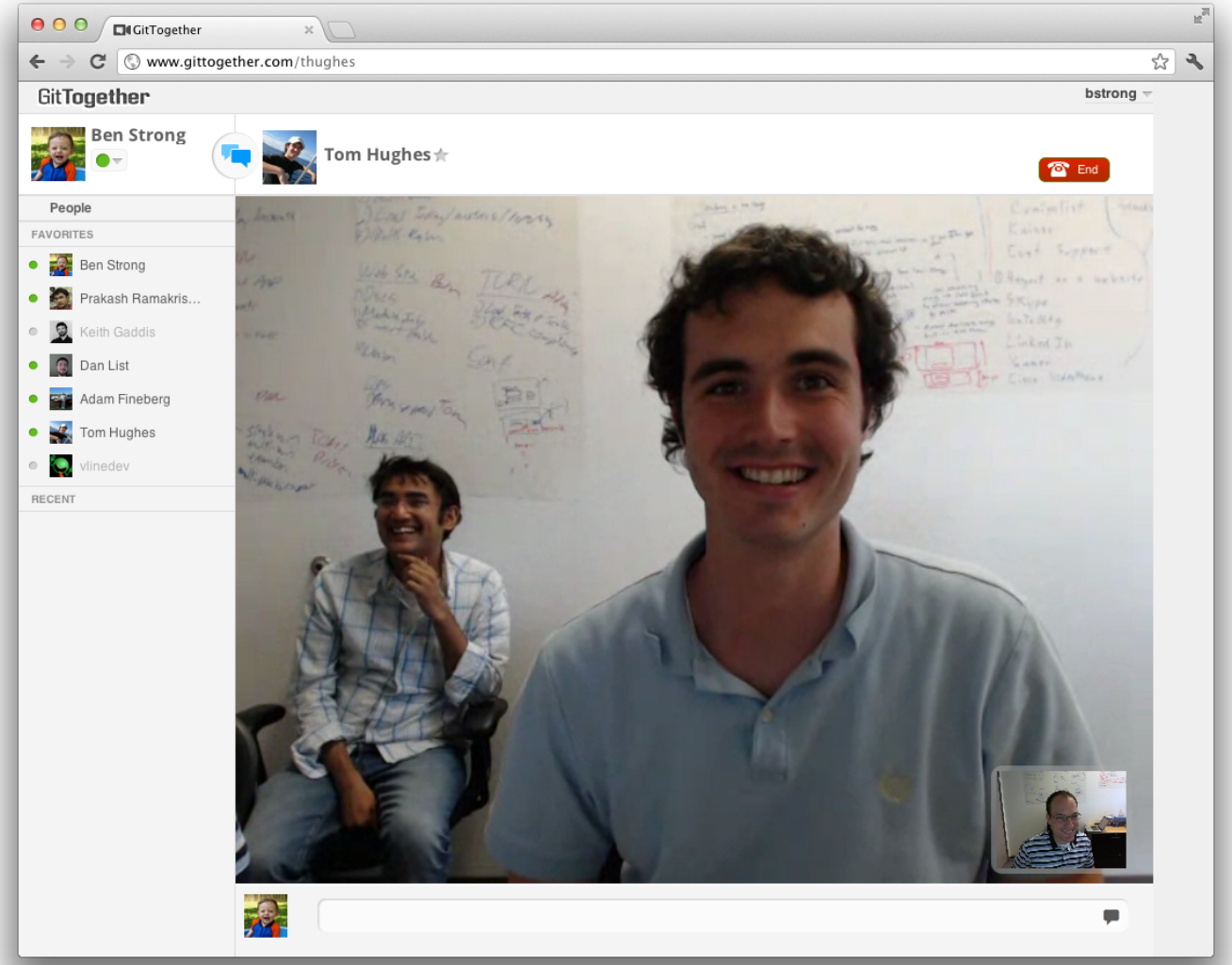
http://www.google.com/chromeframe

TwinSee Demo



GitTogether Demo

http://www.gittogether.com



<Thank You!/>

For more about WebRTC, visit **www.webrtc.org**

juberti@google.com
@juberti
+Justin Uberti


Google
Developers