November 18-20, 2014 - San Jose Convention Center - San Jose, California

WebRTC V
CONFERENCE & EXPO

Explore Changes in Enterprise Communications

#webrtcexpo | @webrtcexpo

www.webrtcexpo.com

Signaling for Different Applications

Matt Krebs
Kelcor, Inc.

Workshop Leaders

- John Riordan
  - OnSIP, Founder and CEO
- Dr. Thomas Sheffler
  - SightCall,
- Oleg Levy
  - Eyeball Networks,
- Rod Apeldoorn
  - Priologic, EasyRTC Server Lead

onsip

SIGNALING FOR DIFFERENT WEBRTC APPLICATIONS

SIP over WebSocket

John Riordan
WebRTC Conference and Expo
San Jose 2014

SIGNALING OPTIONS

Transport Layer
- WebSocket, Comet

Application Layer
- SIP, XMPP, proprietary, other

Use case dependent
- Can be easy in trivial use case

  Pass JSON between two browsers connected to same URL
- But harder in reality

  Performance, reliability, scaling, interoperation
  Operating signaling network vs operating web servers

HTML5 WEBSOCKET

Provides

- Full-duplex communications channels over a single TCP connection
- Designed to be implemented in web browsers and web servers

Leverages HTTP

- HTTP handshake initiated by client
- HTTP "Upgrade" to WebSocket protocol
- Subprotocols (ie SIP over WebSocket)

Secure

- HTTP/WebSocket unified security model
- As HTTPS is HTTP over TLS…
- WebSocket Secure is WebSocket over TLS

Supported
Firefox 6, Chrome 14,
IE 10, Safari 6…

SIP: SESSION INTERNET PROTOCOL

Pros

- ✓ Mature
- ✓ Federated
- ✓ Interoperable
- ✓ Supports JSON

Cons

- ✗ Not a W3 standard
- ✗ Unfamiliar for web developers
- ✗ Messages hard to parse in JavaScript
- ✗ Bloated, complex, and a lot of extensions

SIP: SESSION INTERNET PROTOCOL

June 2002 - IETF RFC 3261 - https://datatracker.ietf.org/doc/rfc3261/

RFC Abstract

This document describes Session Initiation Protocol (SIP), an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences.

SIP invitations used to create sessions carry session descriptions that allow participants to agree on a set of compatible media types. SIP makes use of elements called proxy servers to help route requests to the user's current location, authenticate and authorize users for services, implement provider call-routing policies, and provide features to users. SIP also provides a registration function that allows users to upload their current locations for use by proxy servers. SIP runs on top of several different transport protocols.

THE WEBSOCKET PROTOCOL AS A TRANSPORT FOR SIP

January 2014 - IETF RFC 7118 - https://datatracker.ietf.org/doc/rfc7118/

RFC Abstract

The WebSocket protocol enables two-way real-time communication between clients and servers in web-based applications. This document specifies a WebSocket subprotocol as a reliable transport mechanism between Session Initiation Protocol (SIP) entities to enable use of SIP in web-oriented deployments.

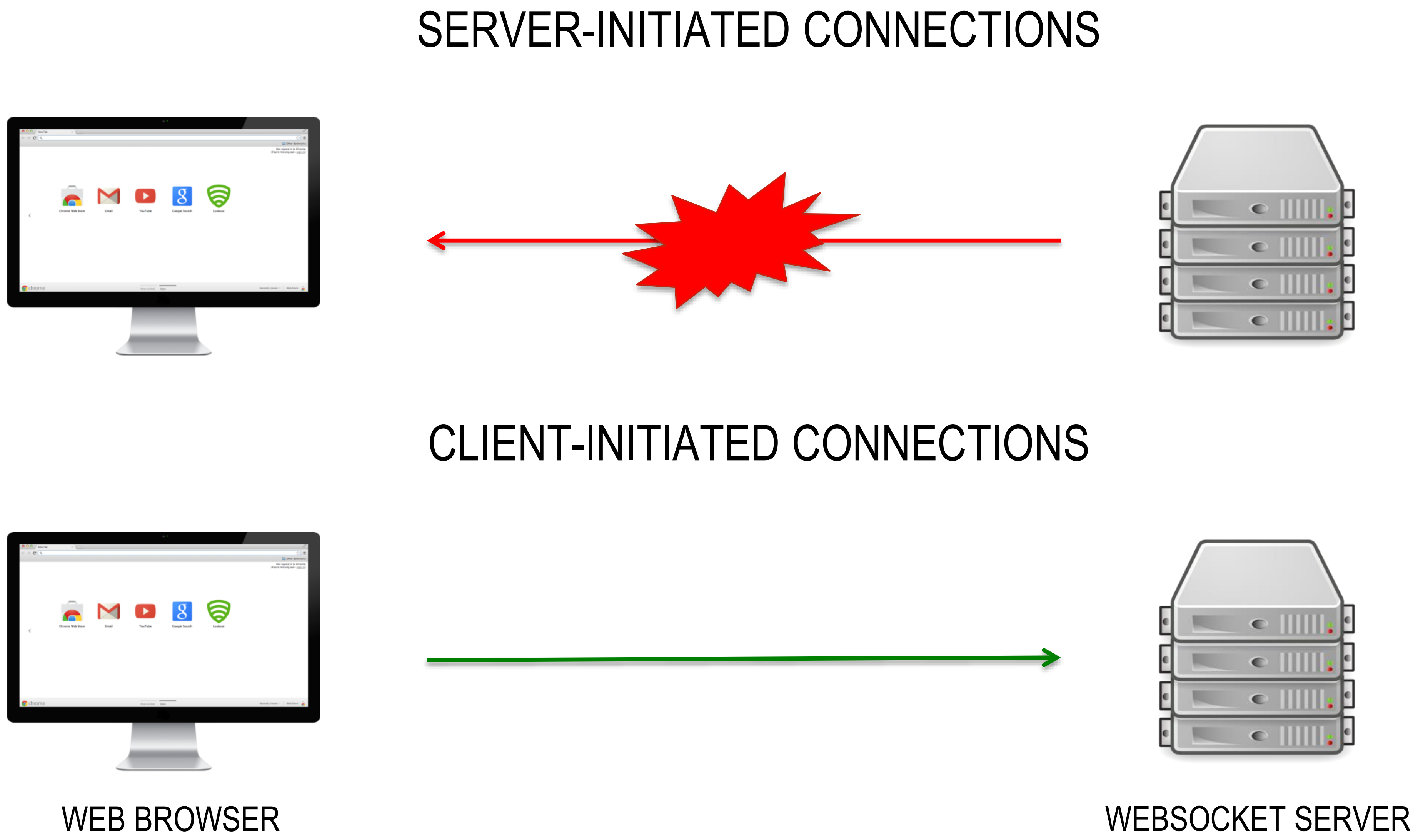
MOBILE WEBRTC APP SIGNALING

Signaling reliability
- Multiple IP networks – ie LTE and WiFi
- Volatile network connectivity

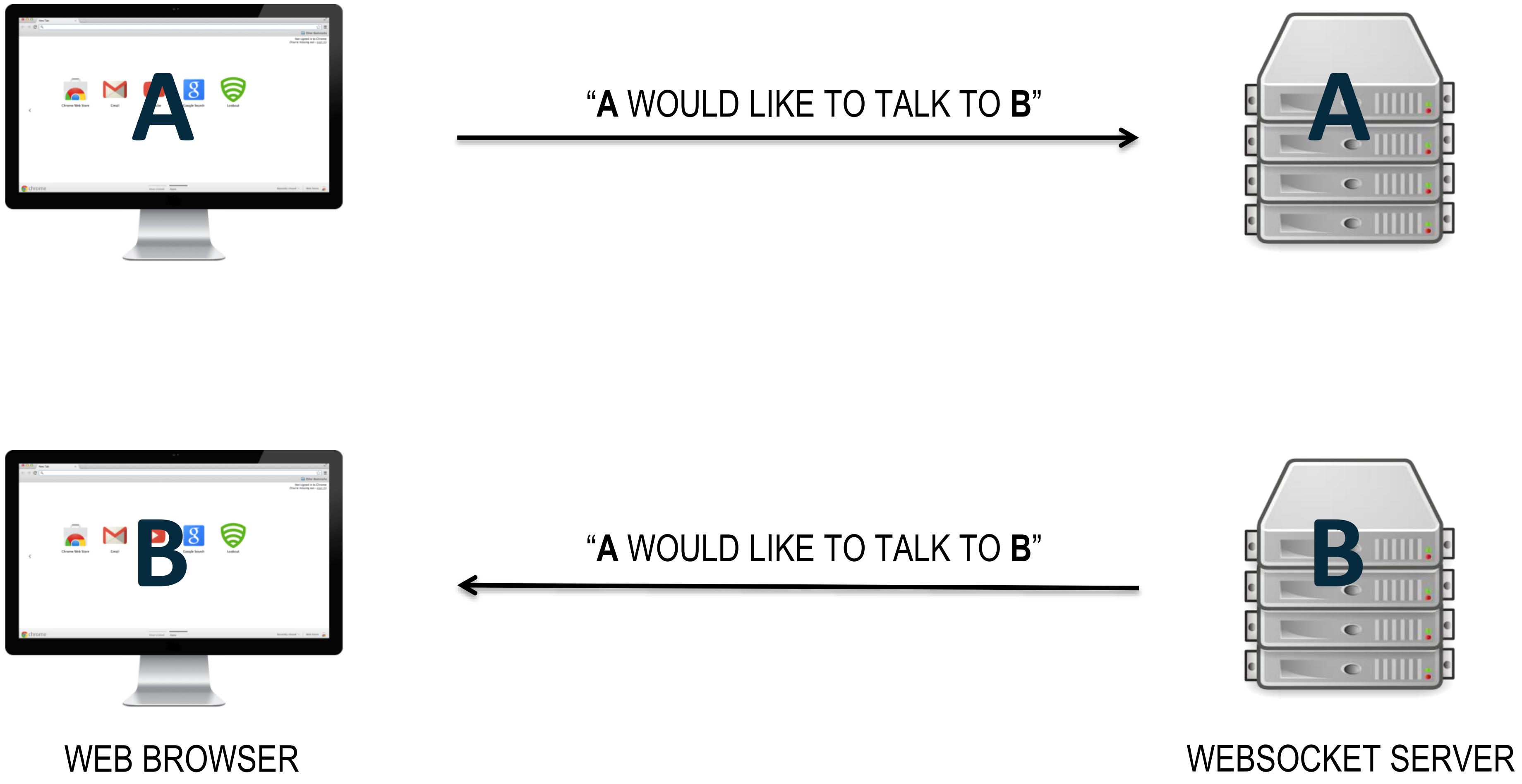
Battery life
- Drained battery impacts application function

iOS Support
- No browser application support
- No native application (WebView) support

SIGNALING CHANNEL IS CLIENT-INITIATED



SIGNALING CHANNEL NEEDS TO BE "UP"



RELIABLE SIGNALING CHANNEL

Must be client initiated
- NATs, Firewalls
- Dynamic network configurations (DHCP)
- No useful or long-term names in DNS

Need low-latency signaling channel for reliable applications
- RTC signals are time sensitive
- Polling signals missed between polls

RELIABLE SIGNALING CHANNEL

Access link failures

- Client needs to reconnect, but might not be aware of link failure
- Multihoming needed to protect against individual access link failure

Avalanche restart problem

- Load on when hosts reconnect simultaneously
- Distribution and load balancing of connections
- Client side exponential back-off warranted

MANAGING CLIENT-INITIATED CONNECTIONS IN SIP

October 2009 - IETF RFC 5626- https://datatracker.ietf.org/doc/rfc5626/

RFC Abstract

The Session Initiation Protocol (SIP) allows proxy servers to initiate TCP connections or to send asynchronous UDP datagrams to User Agents in order to deliver requests. However, in a large number of real deployments, many practical considerations, such as the existence of firewalls and Network Address Translators (NATs) or the use of TLS with server-provided certificates, prevent servers from connecting to User Agents in this way. This specification defines behaviors for User Agents, registrars, and proxy servers that allow requests to be delivered on existing connections established by the User Agent. It also defines keep-alive behaviors needed to keep NAT bindings open and specifies the usage of multiple connections from the User Agent to its registrar.

CLOSING

Thank You

email/sip: john@onsip.com

Please jot down any questions for the end of the session

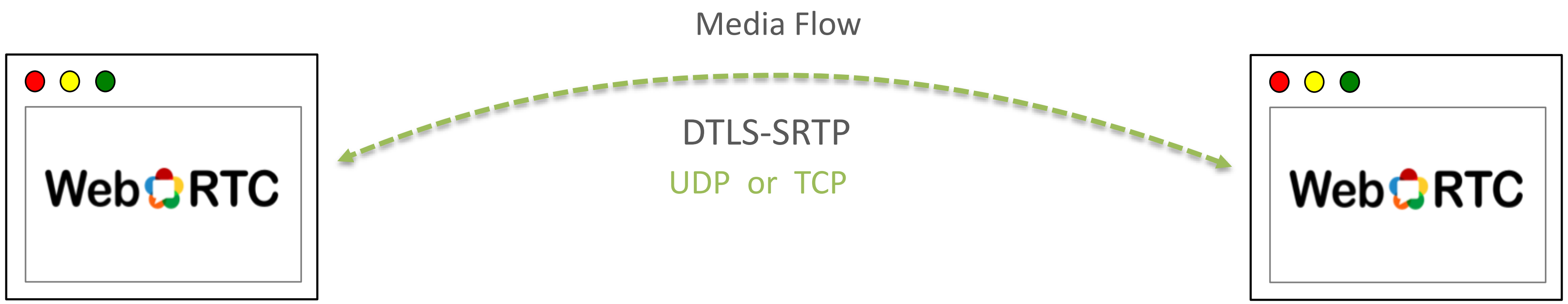
Dr. Thomas Sheffler

SightCall

Signaling Challenges in the Large

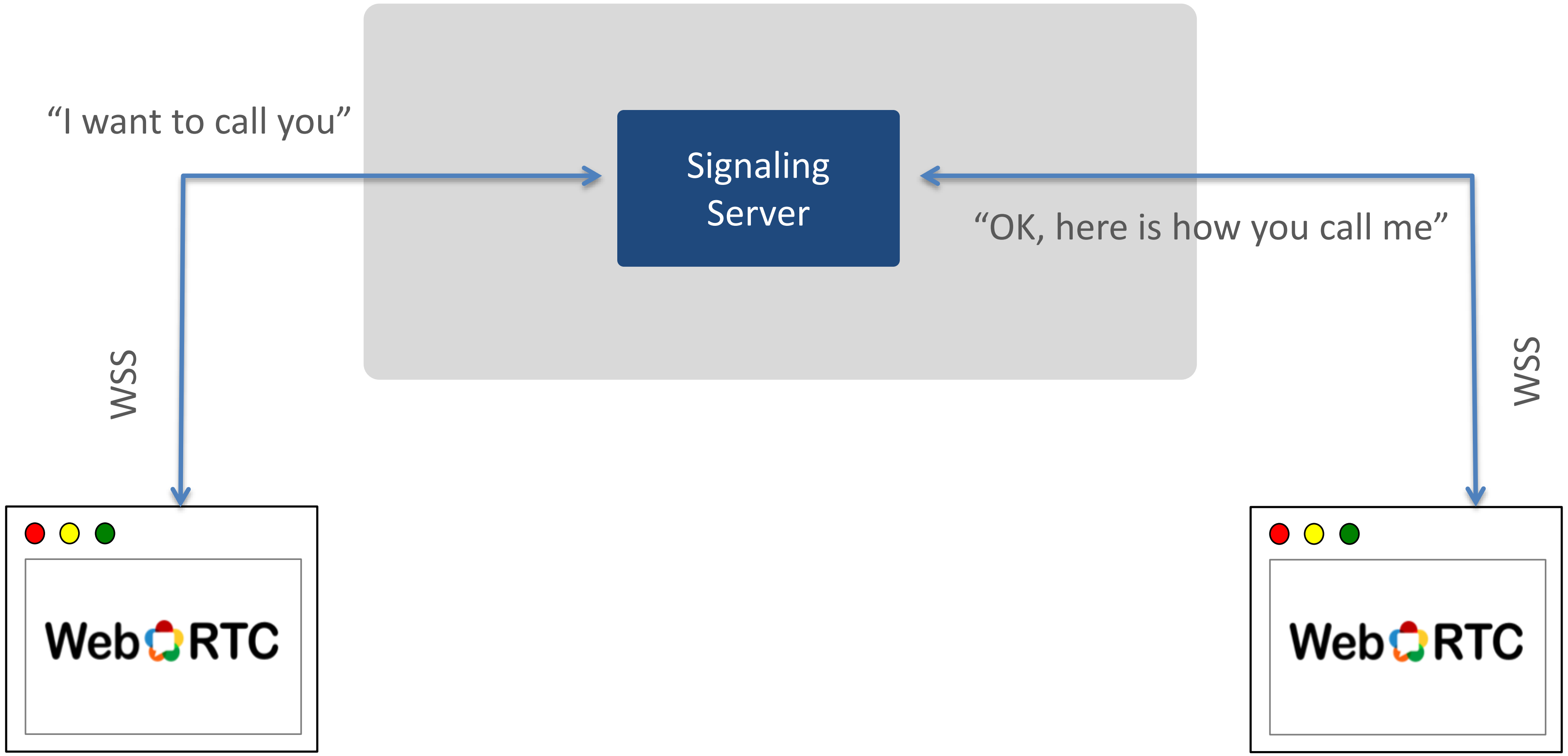
- Signaling – what is it?
- Scaling Issues
- Security Issues
- Mobility Issues

- WebRTC defines the media plane but leaves Signaling undefined

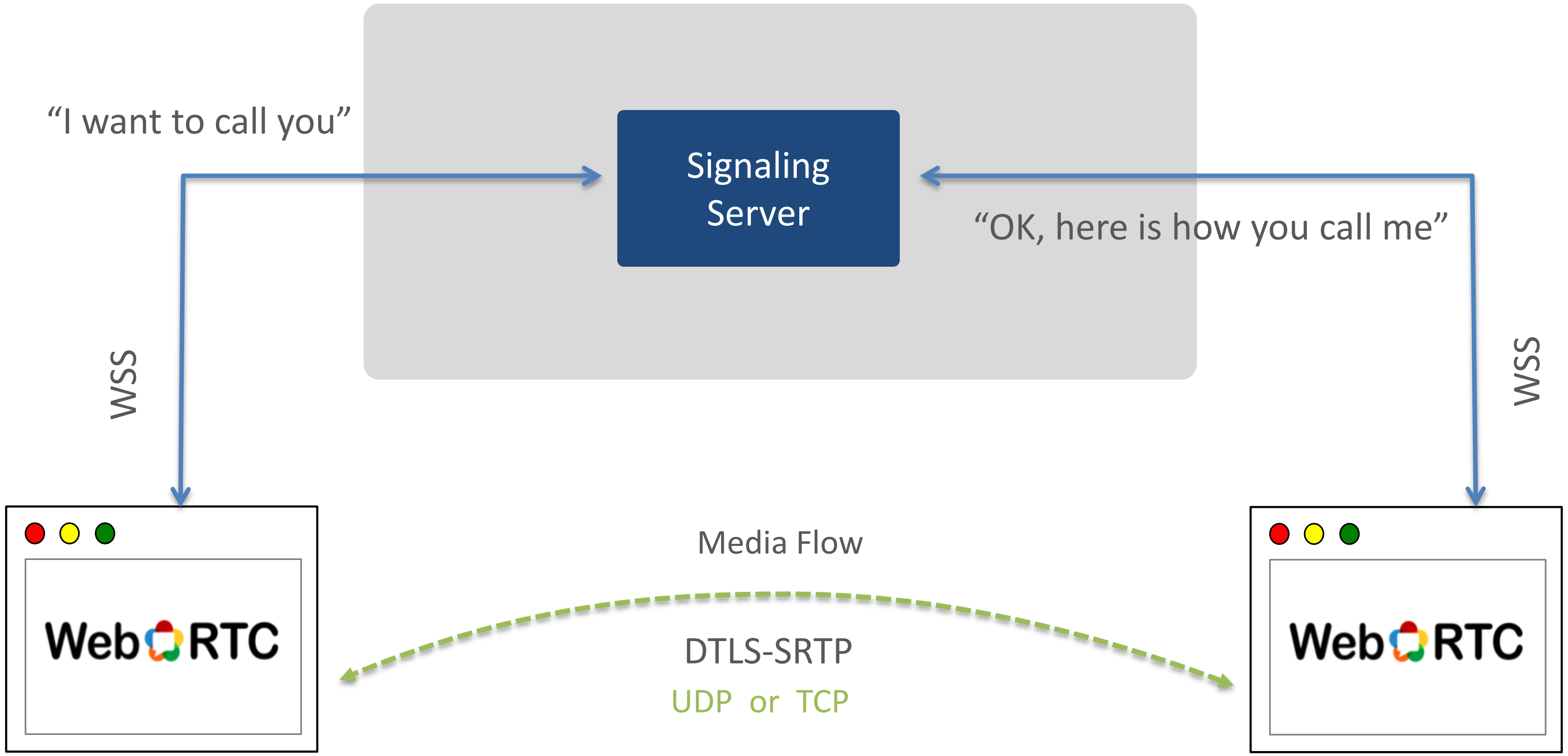


*Web browsers are not servers

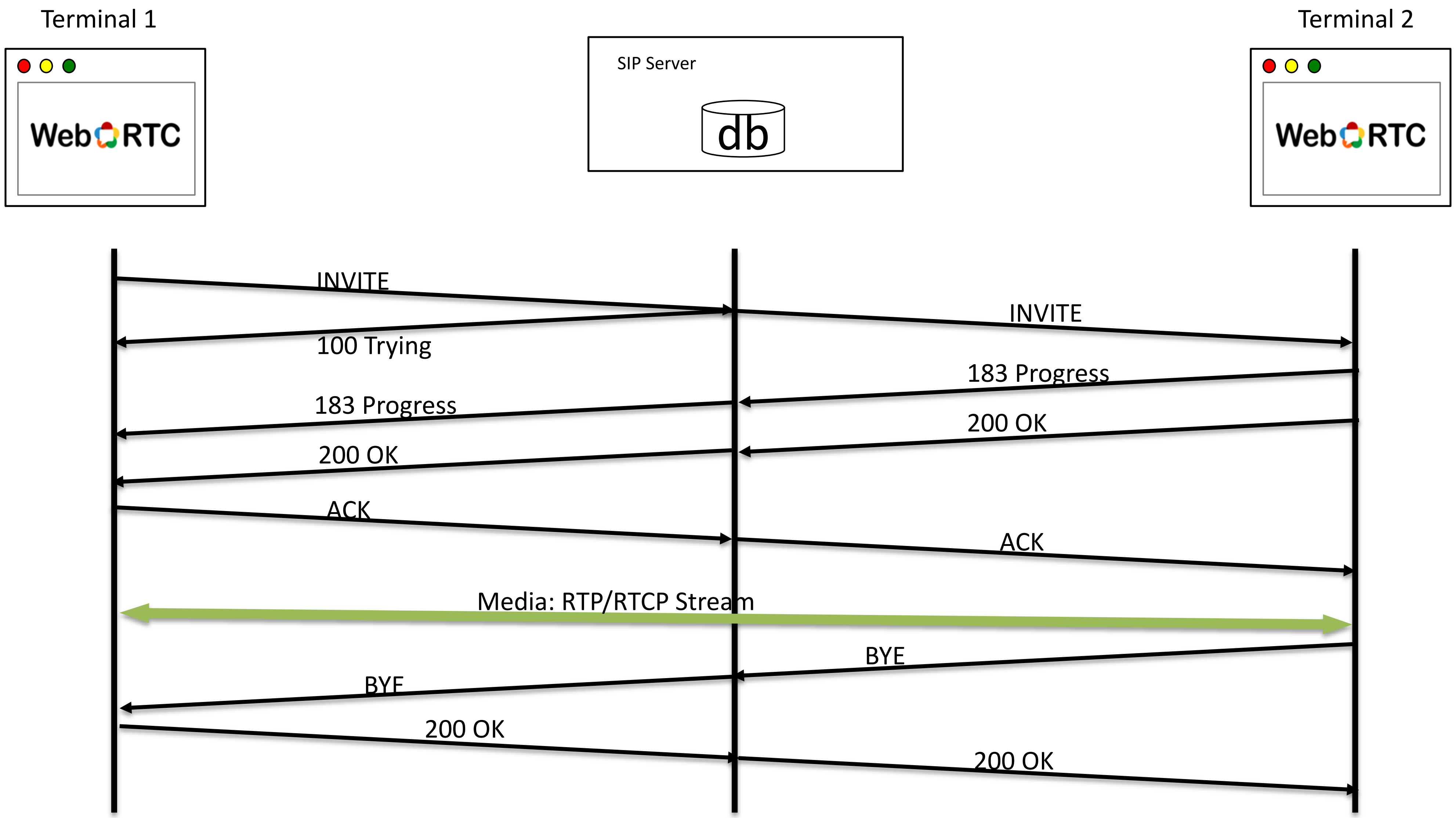
How to send a notification to web page?



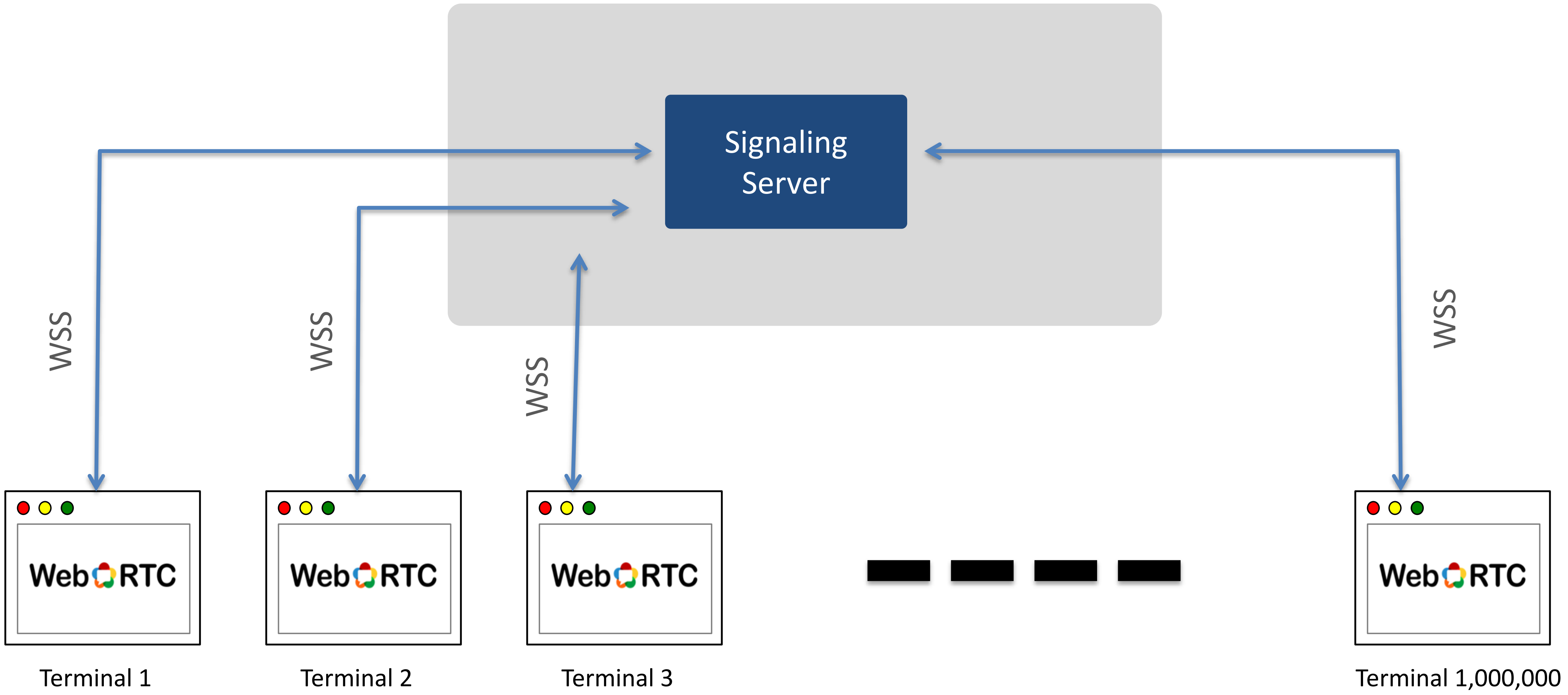
*UDP is not an option


"I want to call you"
Signaling Server
"OK, here is how you call me"
WSS
WSS
Media Flow
DTLS-SRTP
UDP or TCP
WebRTC
WebRTC

Signaling with SIP – an example



Implications of Signaling on Scaling



* millions of open connections

Scaling

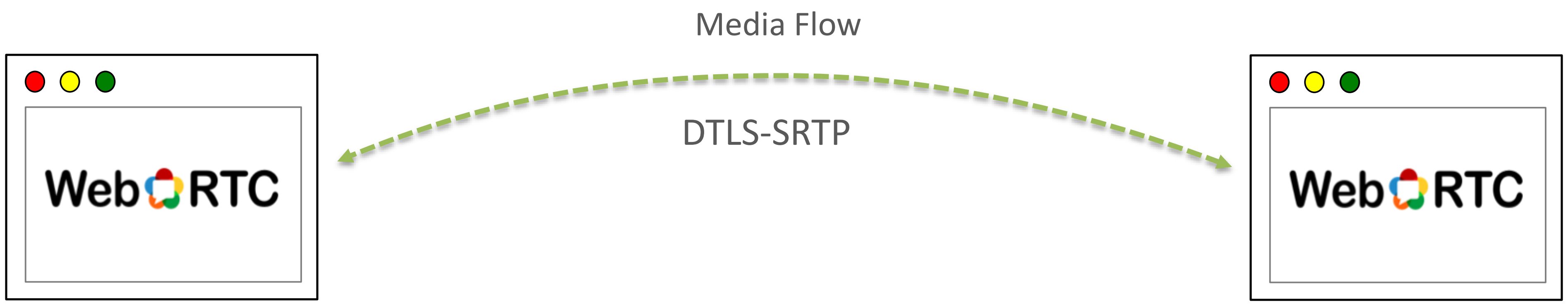
- The Signaling Service must be capable of maintaining millions of open TCP connections.
- A single server cannot do this.
- A distributed architecture is necessary.
  - this is difficult

Security

- WebRTC Security evaluated against three safety objectives
  - Confidentiality - WebRTC
  - Integrity - WebRTC
  - Authenticity - ~~WebRTC~~

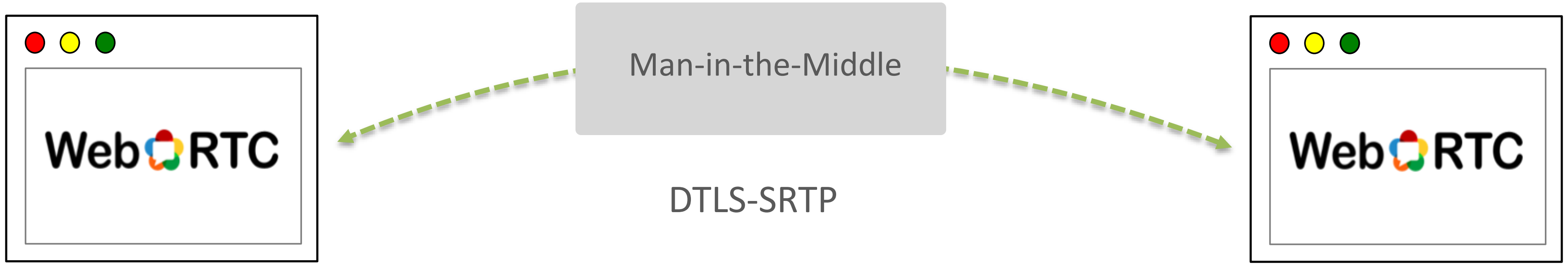
Confidentiality

- Data transferred between two peers does not reach an untrusted third party.
  – handled by encryption



Integrity

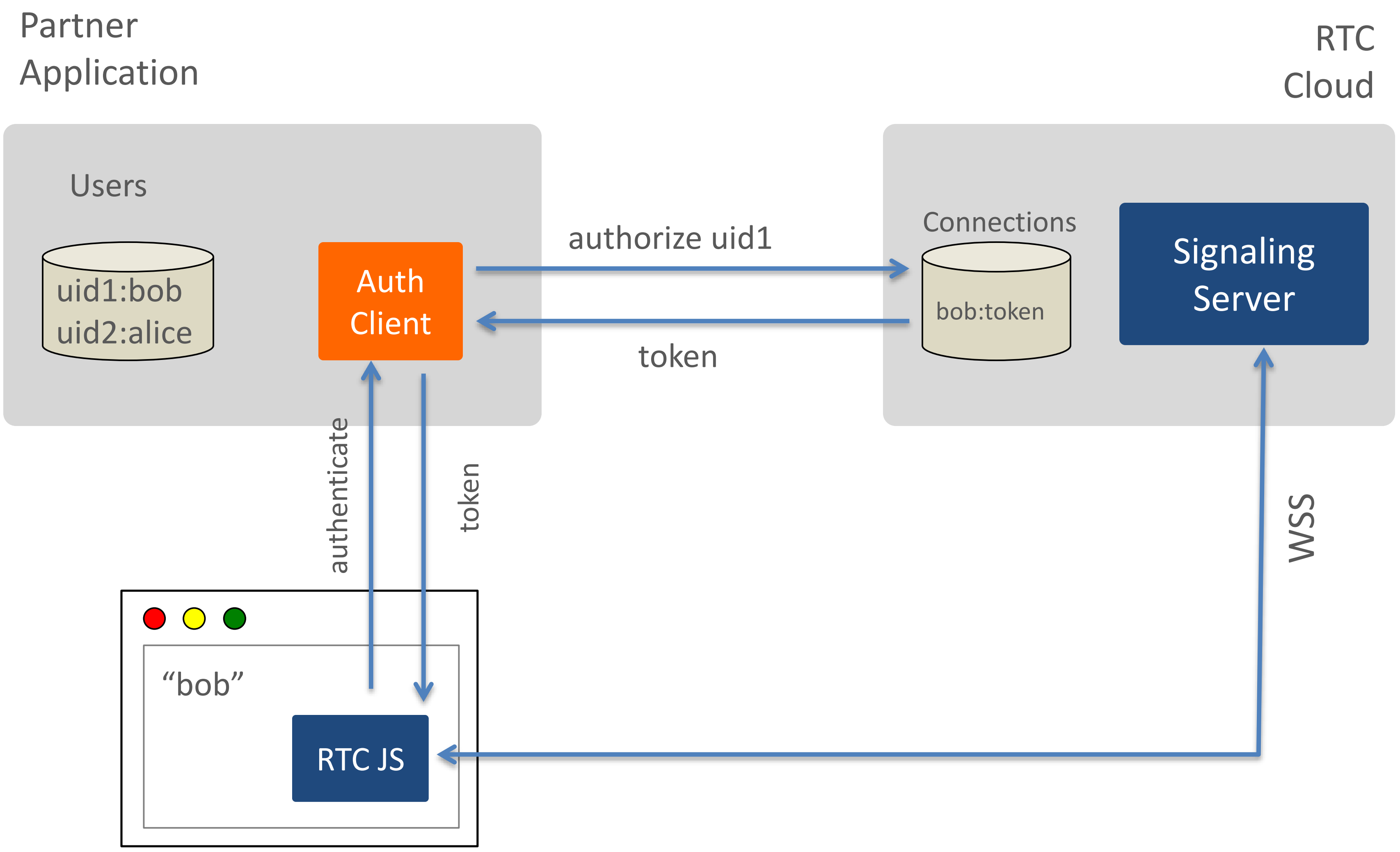
- Data is not modified on the way to the receiver and that the receiver can detect modification.



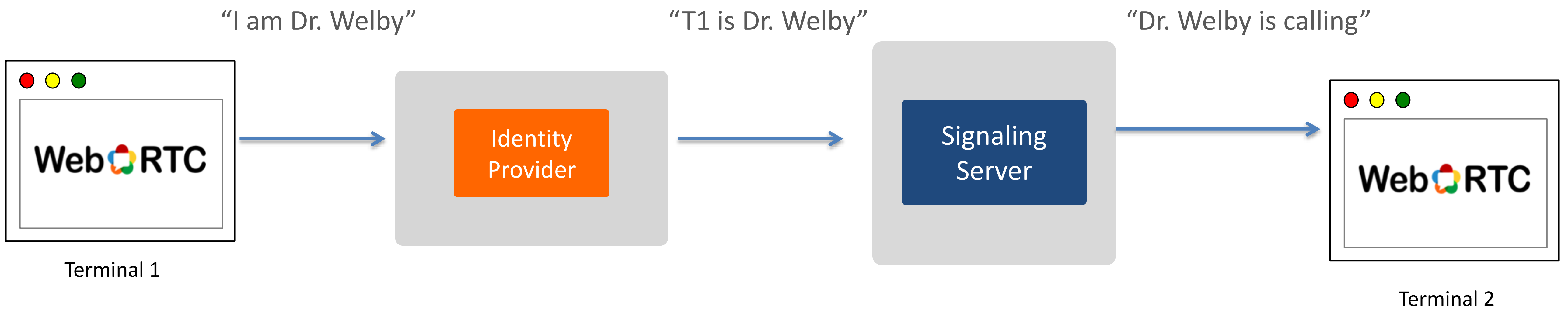
Authenticity

- The claim that the real-time data is really coming from who you think it is.
- WebRTC endpoints are not tied to user identities.
- This becomes an issue of the signaling layer.

Authentication



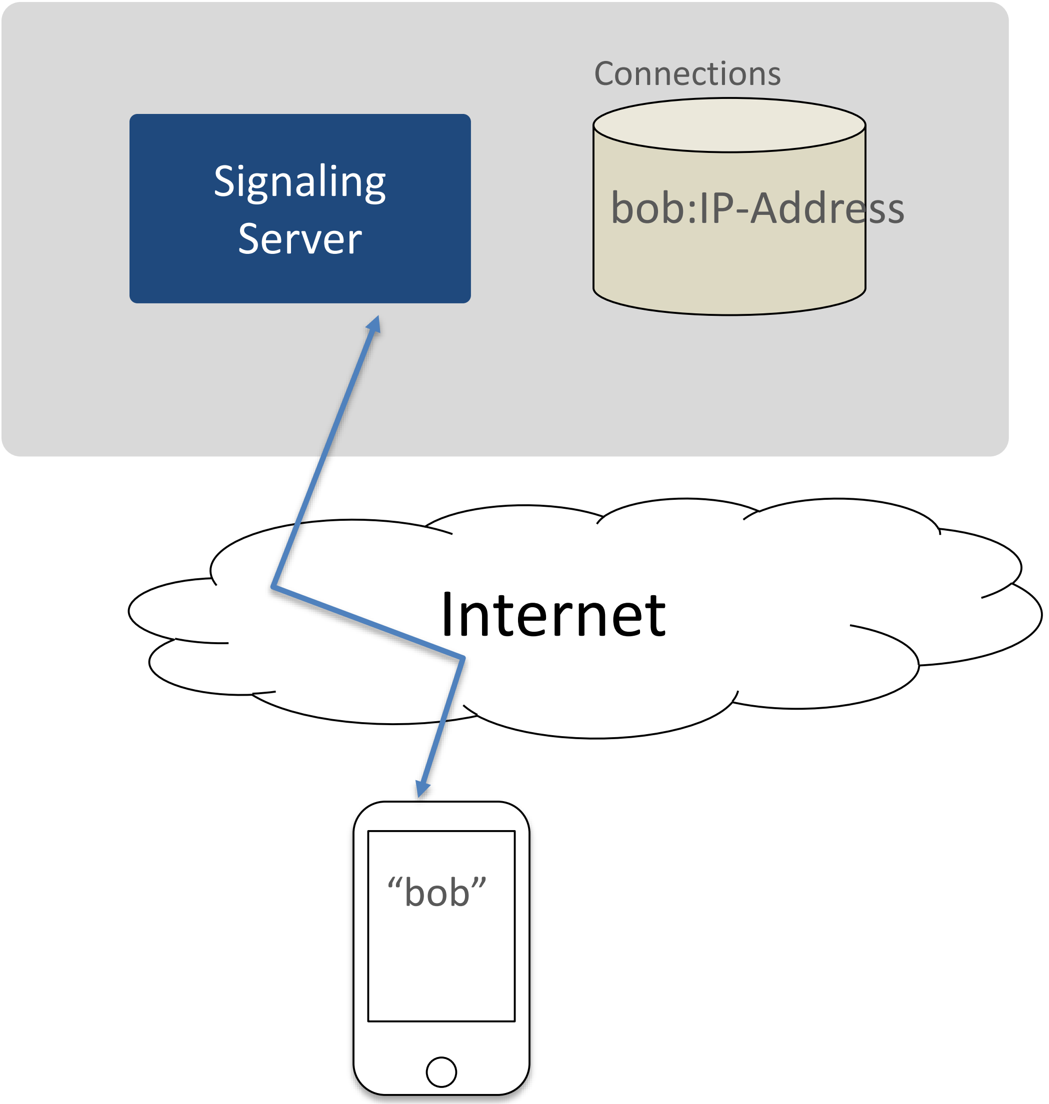
Authenticity: Maintain a Chain Of Trust



Mobility

- mobile devices hop networks (4G <-> WiFi)
  - their IP address changes
- Handoffs between cells affects IP addresses
  – sudden changes in network connectivity [RFC5944: mobilility]
- WebRTC technologies do not really address changing network topologies

When Bob moves his IP address changes



Summary

- WebRTC defines media flow, but leaves signaling undefined
  - This leaves room for lots of innovation
    - SIP over WebSockets, PubNub, Bespoke Protocols
  - Be aware of the challenges
    - Scaling - to millions of open TCP connections
    - Security - ensuring the Authenticity of callers
    - Mobility – signaling in the face of changing network topology

The Future

- ORTC
  - relieves developers from manipulating SDP packets
    - raises the level of abstraction
    - potentially greater interoperability
    - enhanced services easily
      - bypasses limitations of SDP offer/answer
      - for example asymmetry: audio-only endpoint to a/v endpoint

Please jot down any questions for the end of the session

SIGNALING FOR DIFFERENT WEBRTC APPLICATIONS

XMPP and WebRTC

Oleg Levy

Outline

- Quick overview of XMPP
- What can you build with XMPP
- How does XMPP work with WebRTC

e**X**tensible **M**essaging and **P**resence **P**rotocol

- Client-Server
- Simple to work with*
- Secure
- Native support for users with multiple devices
- Presence/Messaging
- XMPP Standards Foundation (XSF)
- Very strong eco-system (batteries included)

What can you build with XMPP?

- Obvious example - Audio/Video/Chat app
  - Multiple logged in endpoints
  - Secure
  - Message archiving
- Smart devices
  - Washing machine control
  - Home alarm system
  - Baby monitor
- Really anything that can be modeled after presence/messaging

So many endpoints, but how can we scale?

- DNS SRV records
  - Part of the spec (unlike HTTP)
  - Users request _xmpp-client._tcp.eyeball.com
  - …and choose the server from the list
- XMPP server farm
  - Requires a load balancer
  - Amazon ELB works nicely

XMPP & Browsers

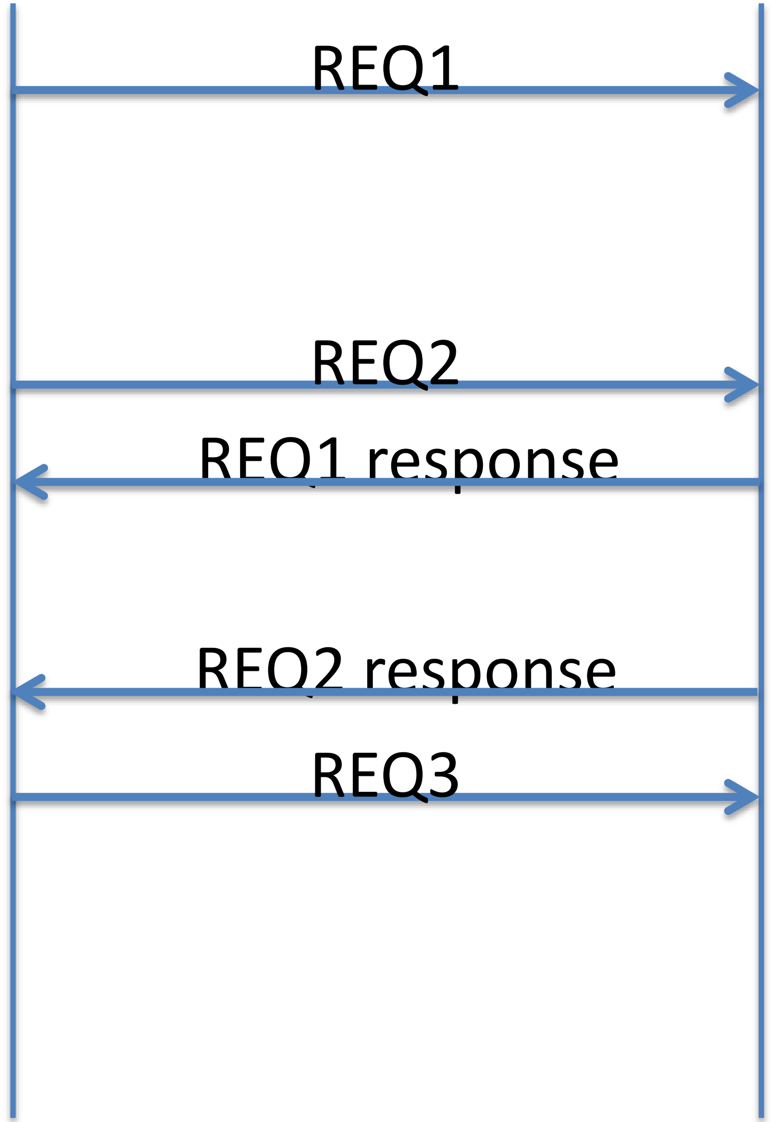
- Native transport is a long lived TCP/TLS connection
  - Try telnetting port 5222
  - **Not** a request/response protocol
- We could try to use WebSockets
  - Load balancing is not trivial
  - At least for now, ELB doesn't support it
- BOSH is the native method

Bidirectional-streams Over Synchronous HTTP

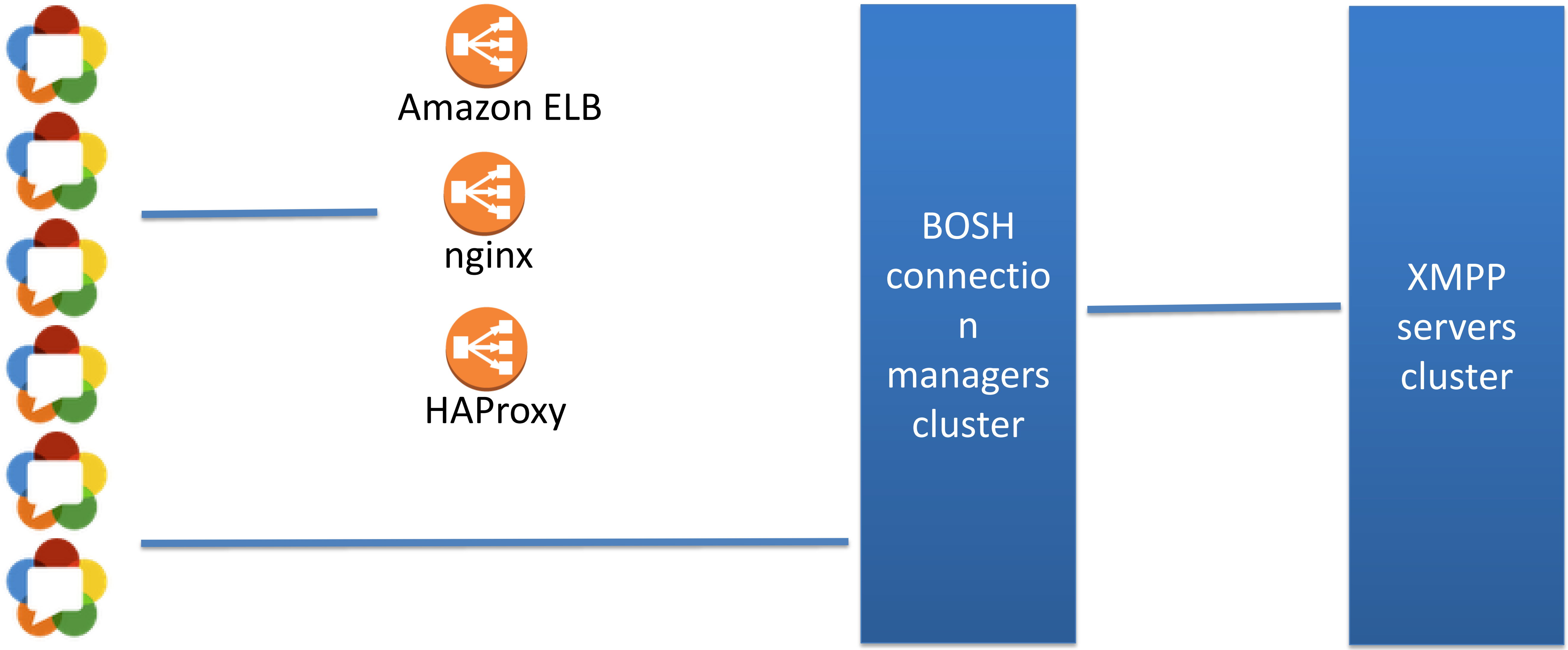
- Efficient method to simulate long-lived connections with HTTP
- Secure
- Compatible with standard HTTP endpoints
  - Proxies, firewalls
  - Load balancers
  - **And the rest of your HTTP infrastructure**
- Doesn't require HTTP/1.1
  - But probably not very important at this point

How does BOSH work?

- Client wants to send data
  - Sends HTTP request
  - Response is **only when** the server has anything to say
- Client wants to send more data
  - Sends a new HTTP request
  - Server responds to the previous request
  - New request is now held open
- Server wants to send data
  - Puts the data into the open HTTP session and responds
  - Client gets the response and immediately opens a new HTTP connection
- One connection is always open
  - At most two



scalable XMPP for WebRTC



Again, please note questions for the end

(it is almost here)

Custom and Data Channel Signaling

Rod Apeldoorn
EasyRTC Server Lead
Priologic Software Inc.

Example Custom Message Types
(from EasyRTC)

WebRTC Core

- candidate
- offer
- answer
- reject

Application Level

- authenticate
- hangup
- getIceConfig
- roomJoin
- roomData
- setPresence
- filesOffer
- Many more…

Why Combine WebRTC Signaling with Application Servers?

- Authentication
- Call logging
- Call control
- Combine with application logic
- Client connects to just one server
  – Why SIP + Presence + Application servers?

Transports

Websockets

- Available in all modern browsers
- Fast + Responsive + Securable
- Maintains open socket
- Servers have to deal with concurrent socket limits

XHR Polling

- AKA "HTTP Long Polling"
- Easy + Securable
- To use:
  - XMLHttpRequest API
  - jquery.ajax()
- Used by Google AppRTC Demo

Transports cont.

JSONP + CORS

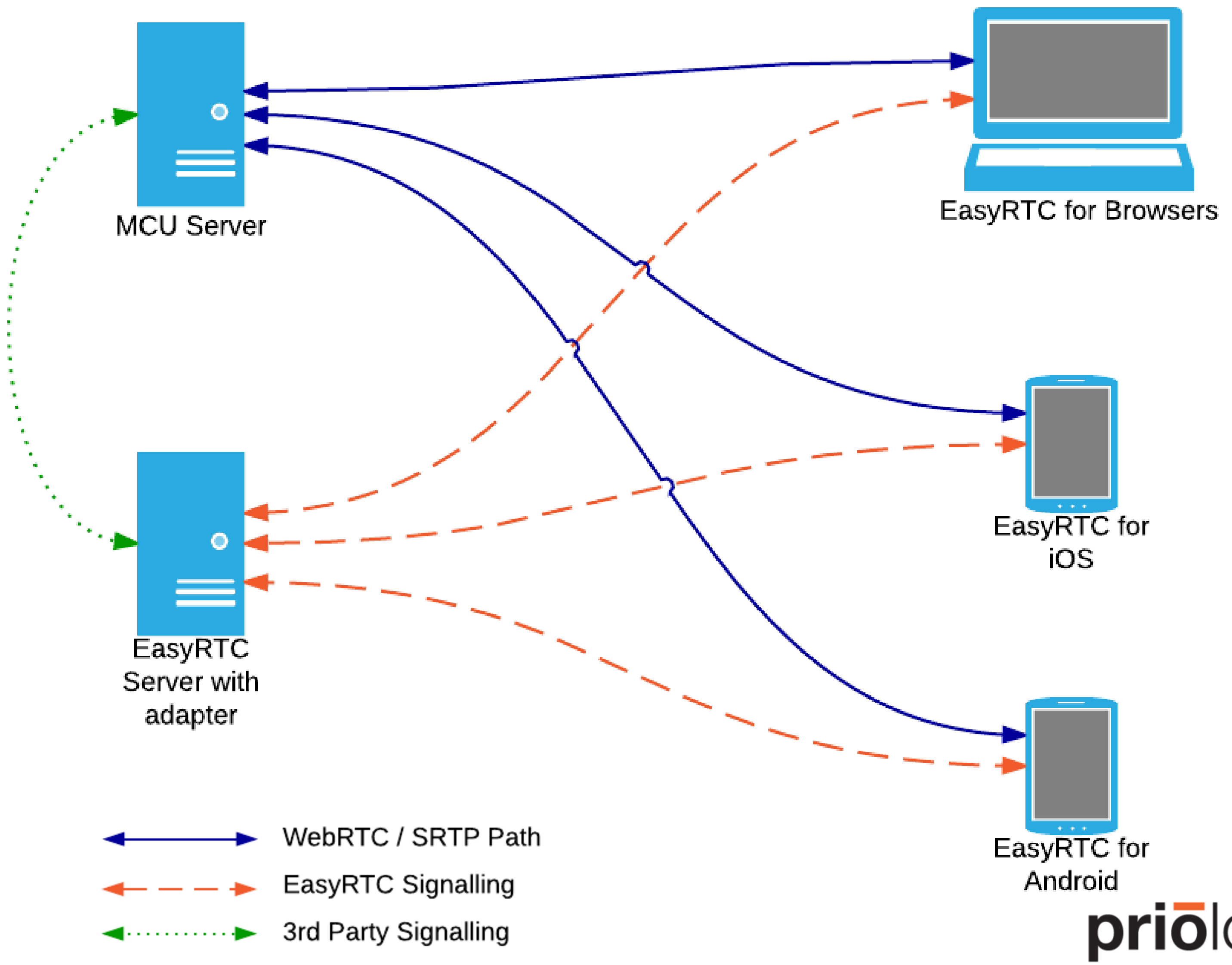
- The original popular method for DHTML
- Cross site scripting issues
- “Cross-Origin Resource Sharing” can be setup
- Still a valid fallback
  - Especially for older browsers

Other

- XMPP (Jabber)
  - Instant messengers
- Local
  - Bluetooth
  - USB / Serial
- WebRTC Data Channels
  - Example coming!

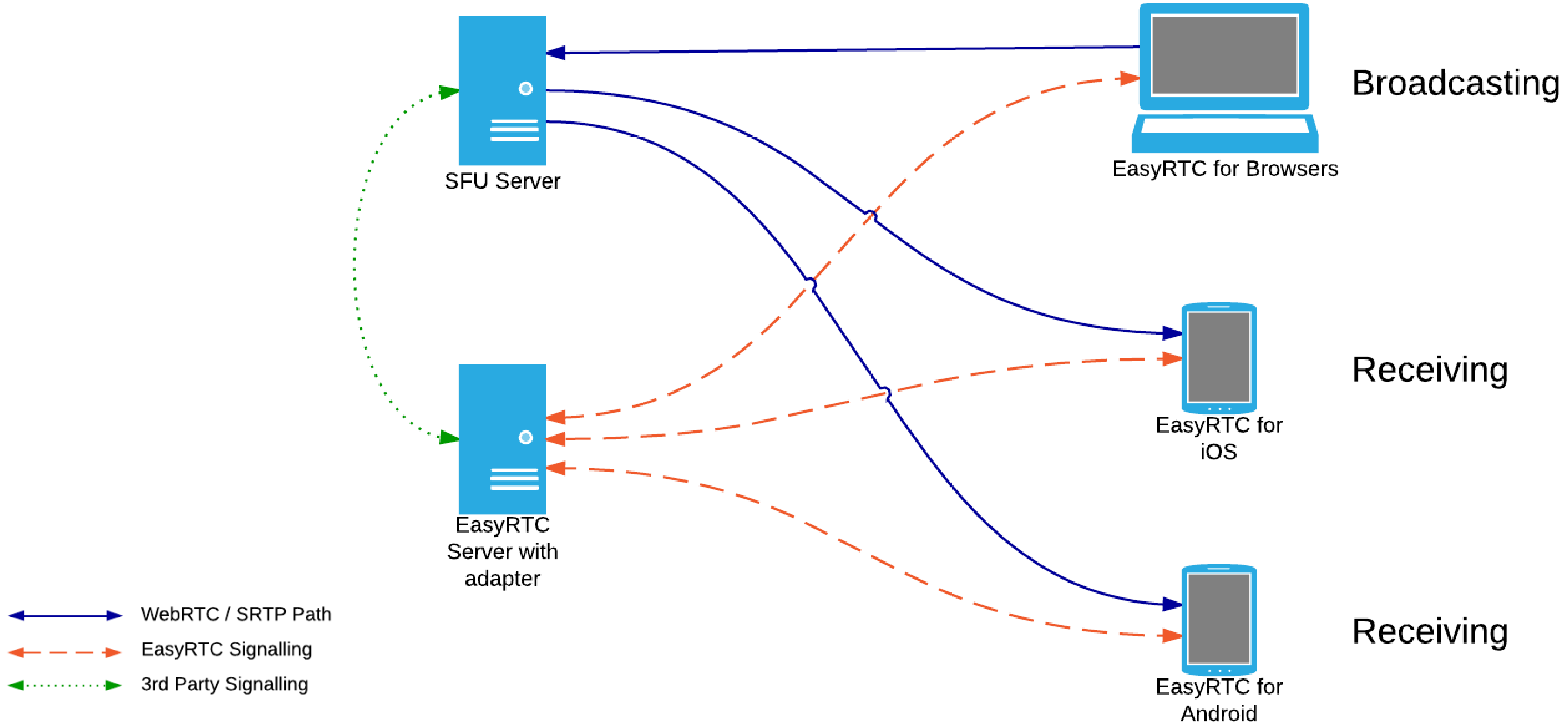
Restful WebRTC Clients

- Conference server (MCU)
- Recording and playback server
- WebRTC / SIP Gateway
- Selective forwarding units
- Test clients



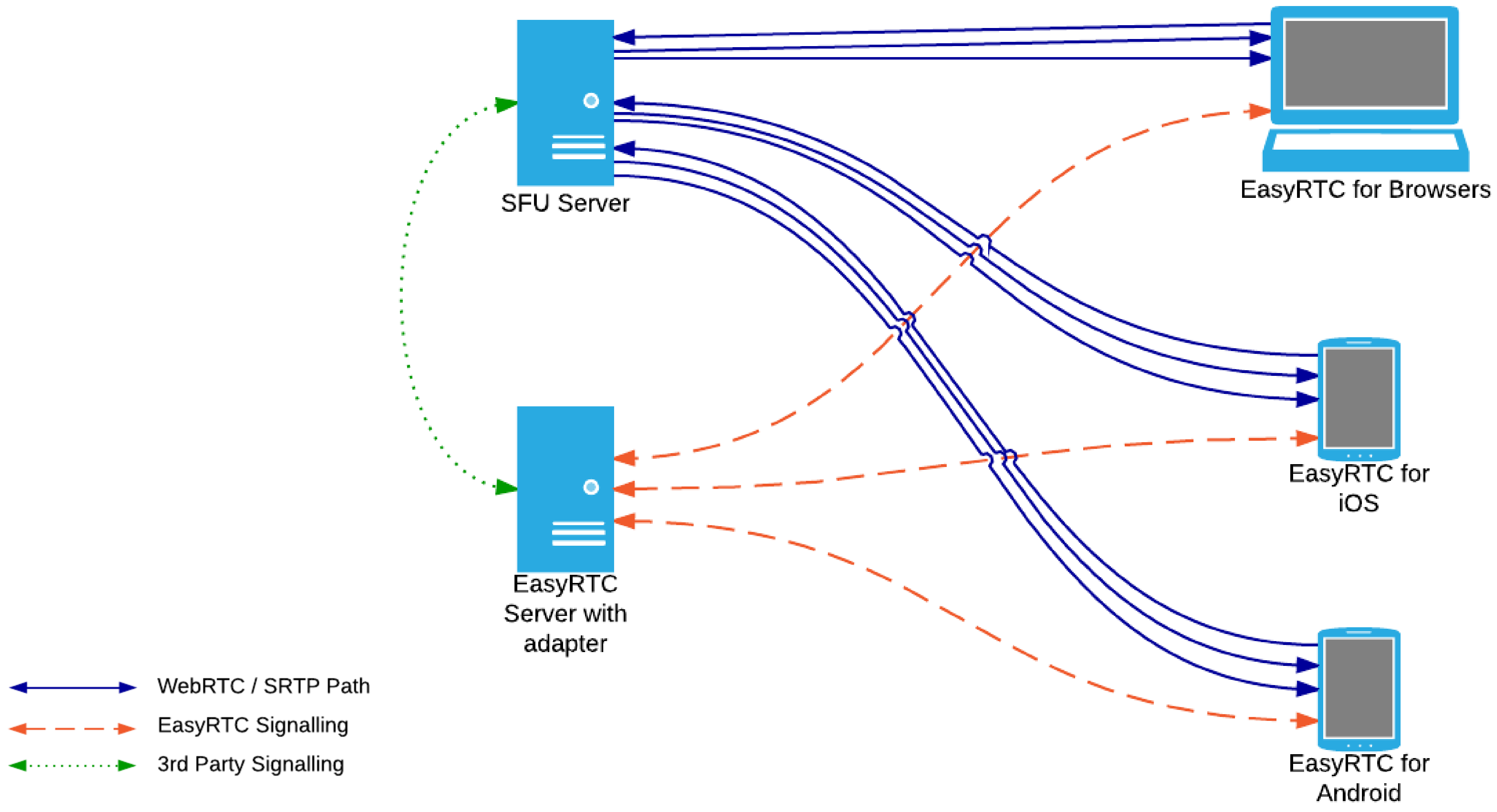
priologic

Broadcast SFU Signaling Path



priologic

Conference SFU Signaling Path



priologic

Data Channel Signaling

What Can Be Done?

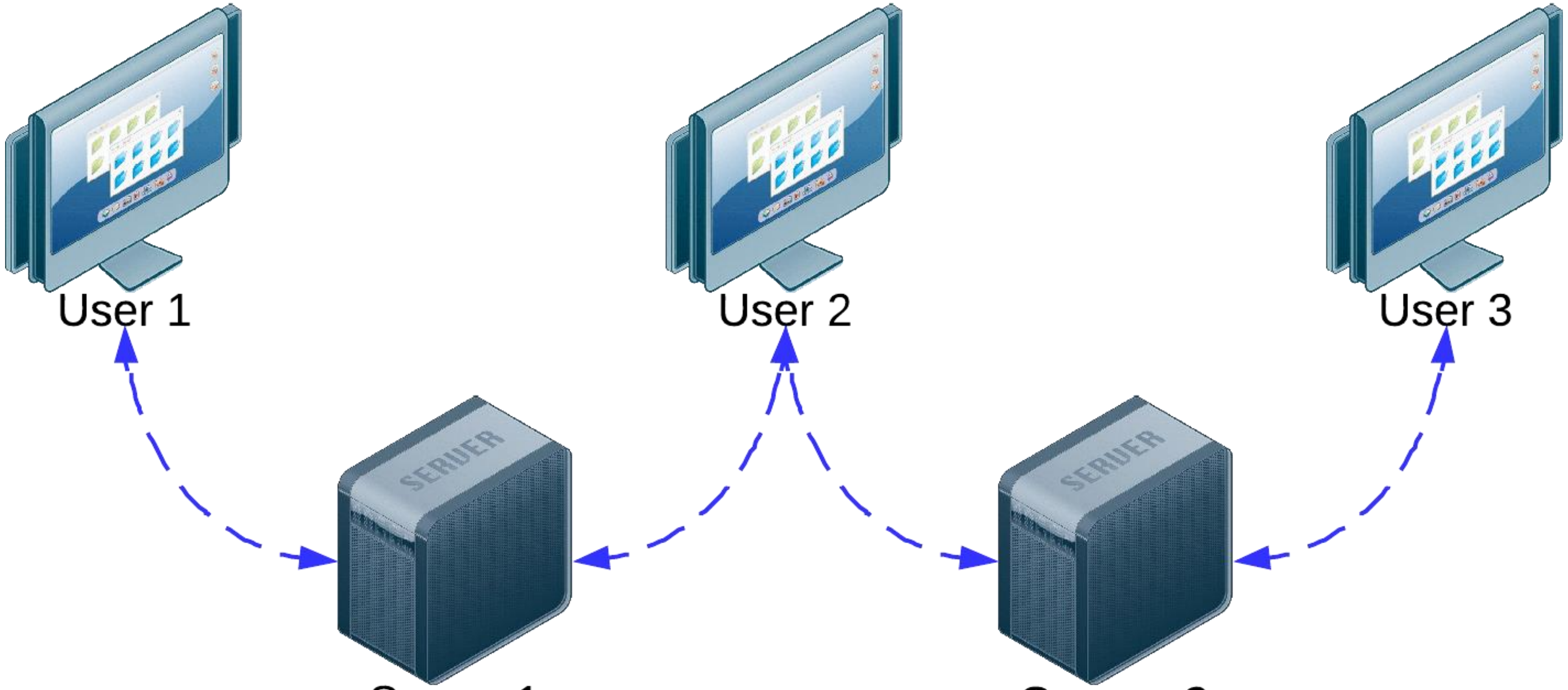
- Text messaging
- File transfers
- Gaming
- Call quality feedback and control
- Private networking

Benefits & Limitations

- Offload requests to central server
- Greater privacy
- Reduced latency
- Greater speed
- Lose server control
  - security
- Inconsistent Data Channel support

Private WebRTC Signaling

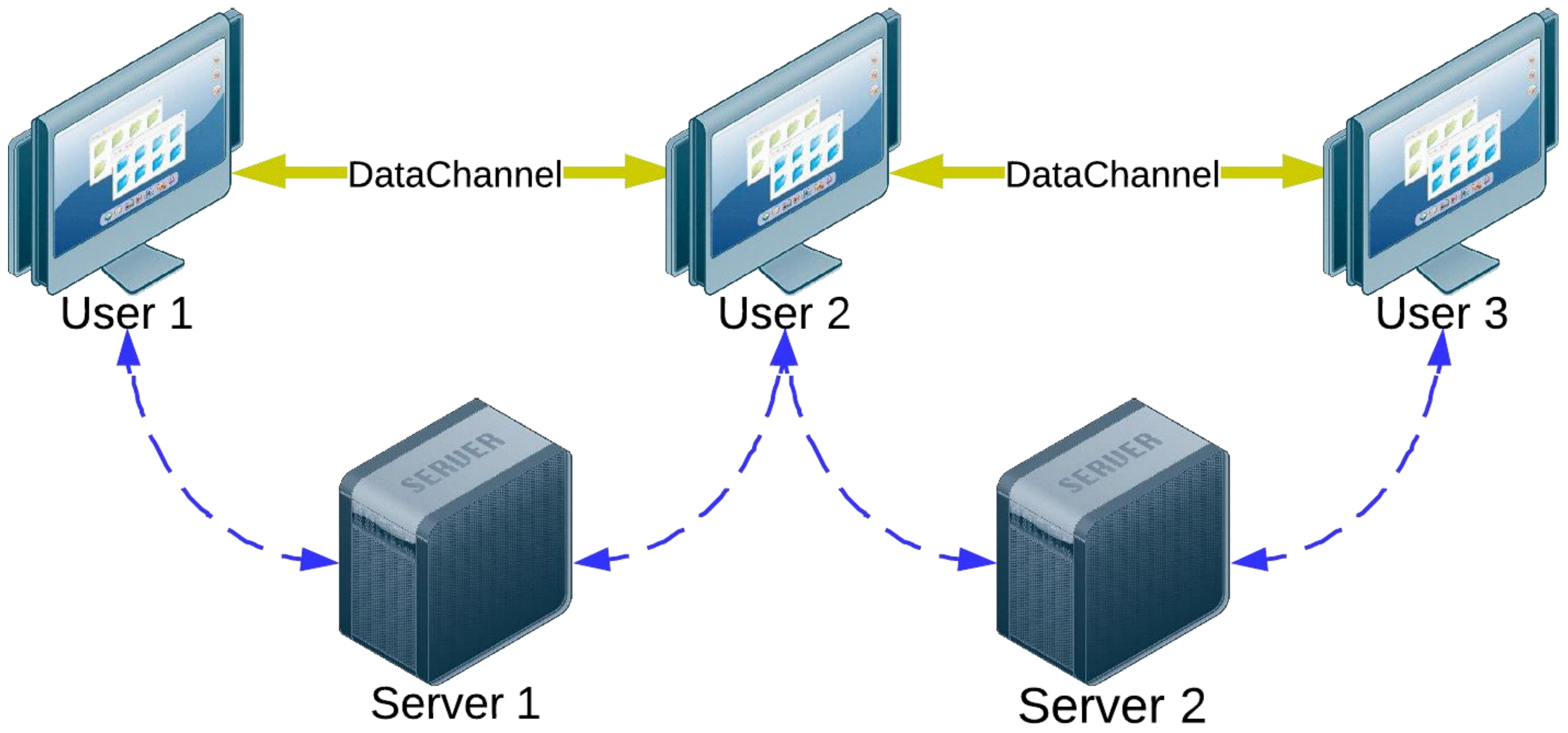
1. Connect users to servers via Websockets
2. Establish DataChannels between users on same servers
3. Establish WebRTC Peer Connection between User 1 and 3
   - Signals sent via DataChannel
   - User 2 acts as a relay
   - Neither server aware of final connection



priologic

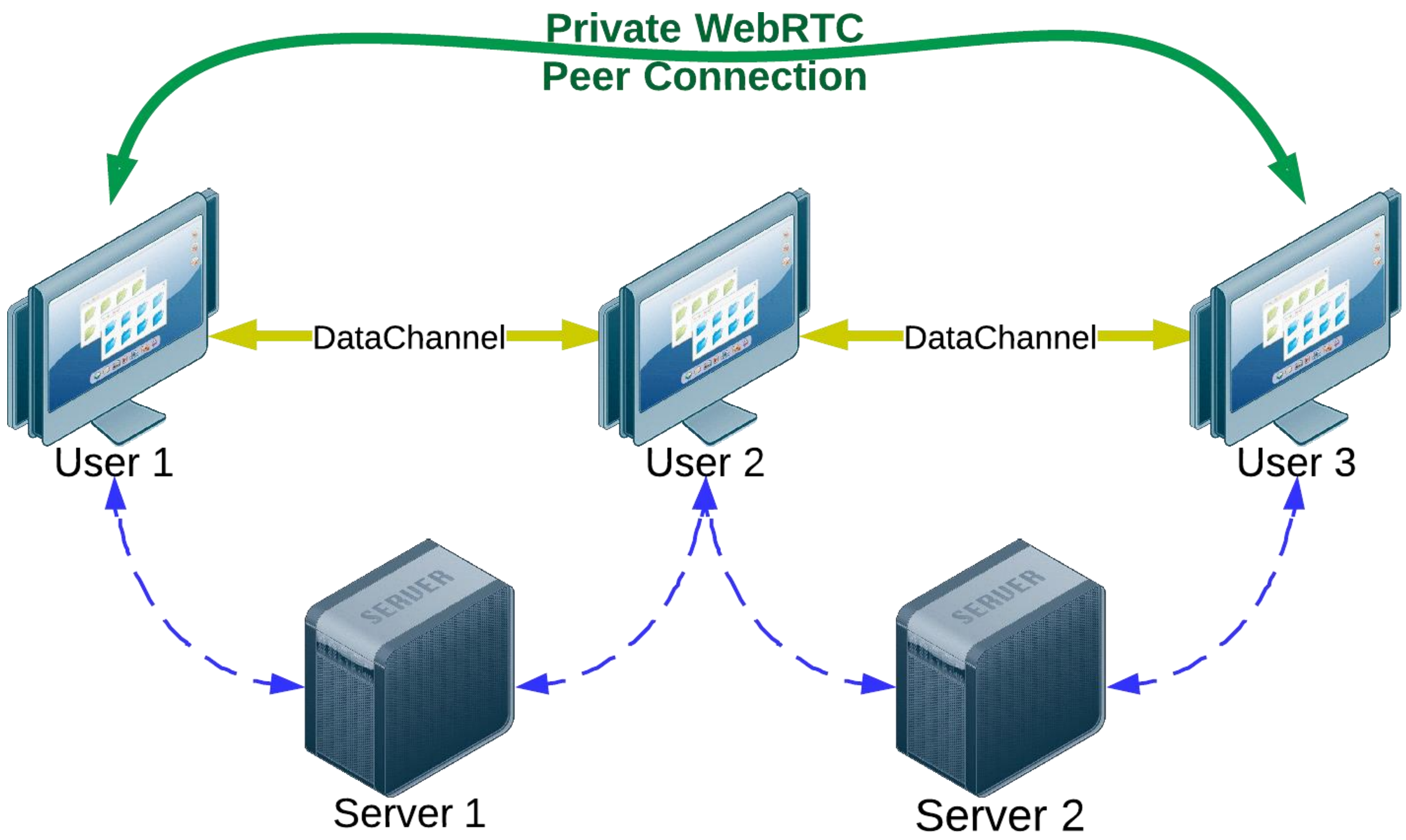
Private WebRTC Signaling

1. Connect users to servers via Websockets
2. Establish DataChannels between users on same servers
3. Establish WebRTC Peer Connection between User 1 and 3
   - Signals sent via DataChannel
   - User 2 acts as a relay
   - Neither server aware of final connection



Private WebRTC Signaling

1. Connect users to servers via Websockets
2. Establish DataChannels between users on same servers
3. Establish WebRTC Peer Connection between User 1 and 3
   - Signals sent via DataChannel
   - User 2 acts as a relay
   - Neither server aware of final connection



priologic

That was the end.

Now: questions?