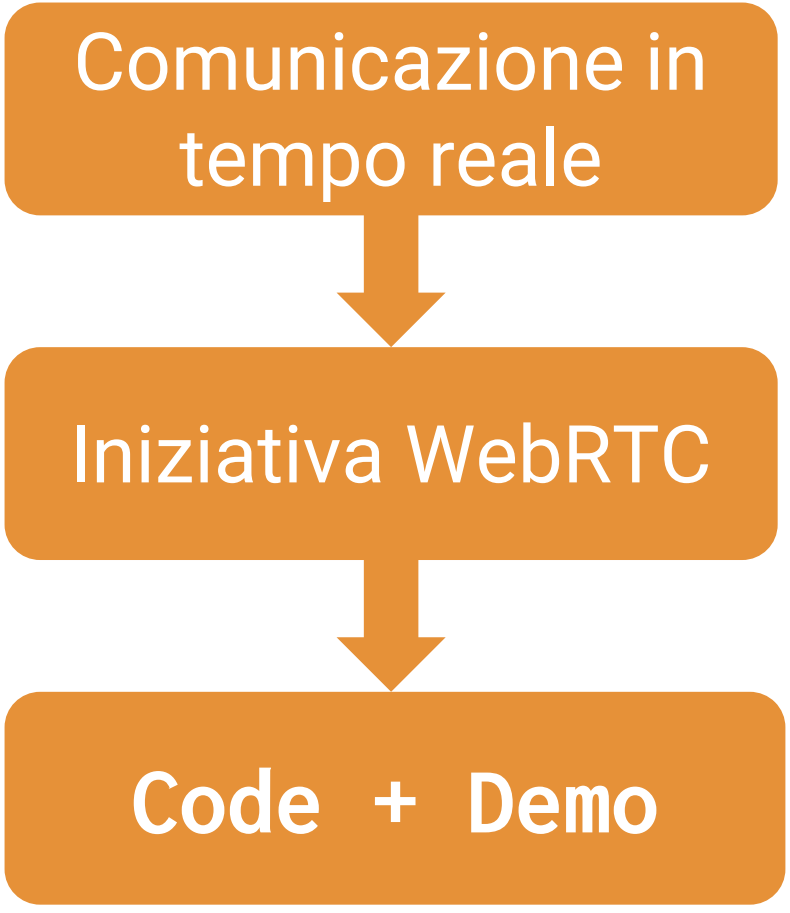
Politecnico di Bari
Dipartimento di Ingegneria Elettrica e dell'Informazione

C3LAB
Control of Computing and Communication Systems Lab

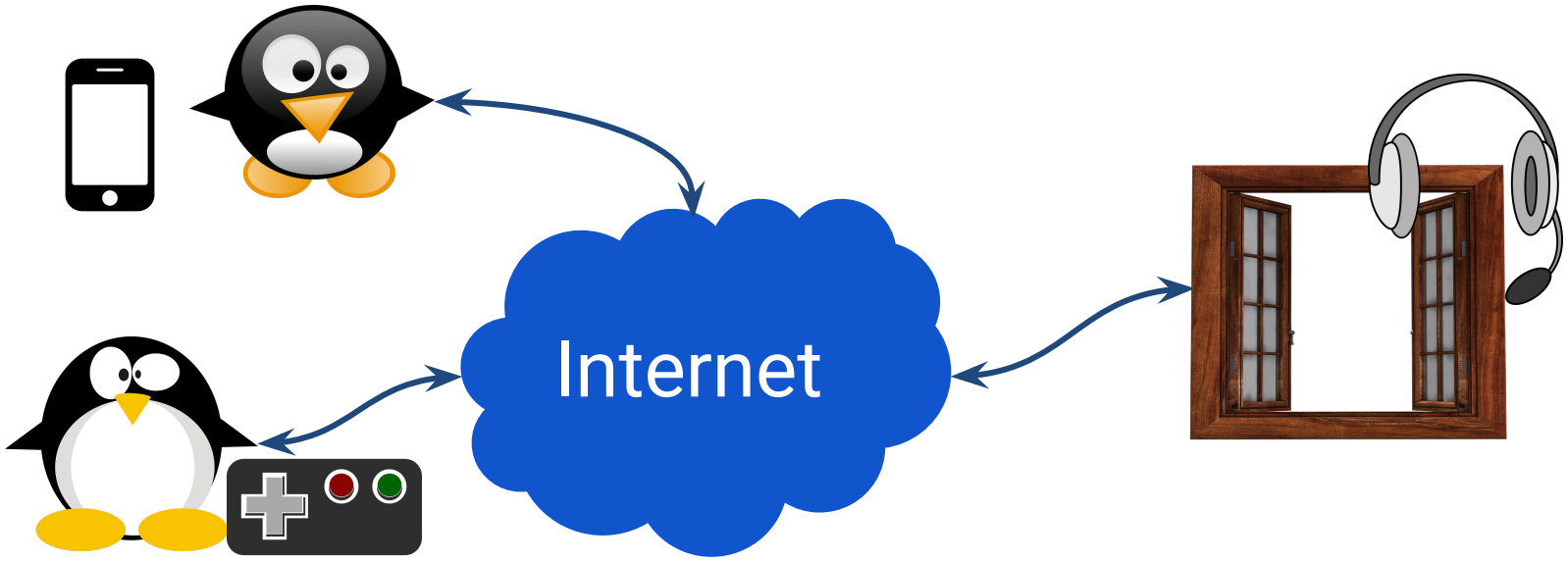
Comunicazione Real-Time con WebRTC

Luca De Cicco
luca.decicco@poliba.it

AGENDA



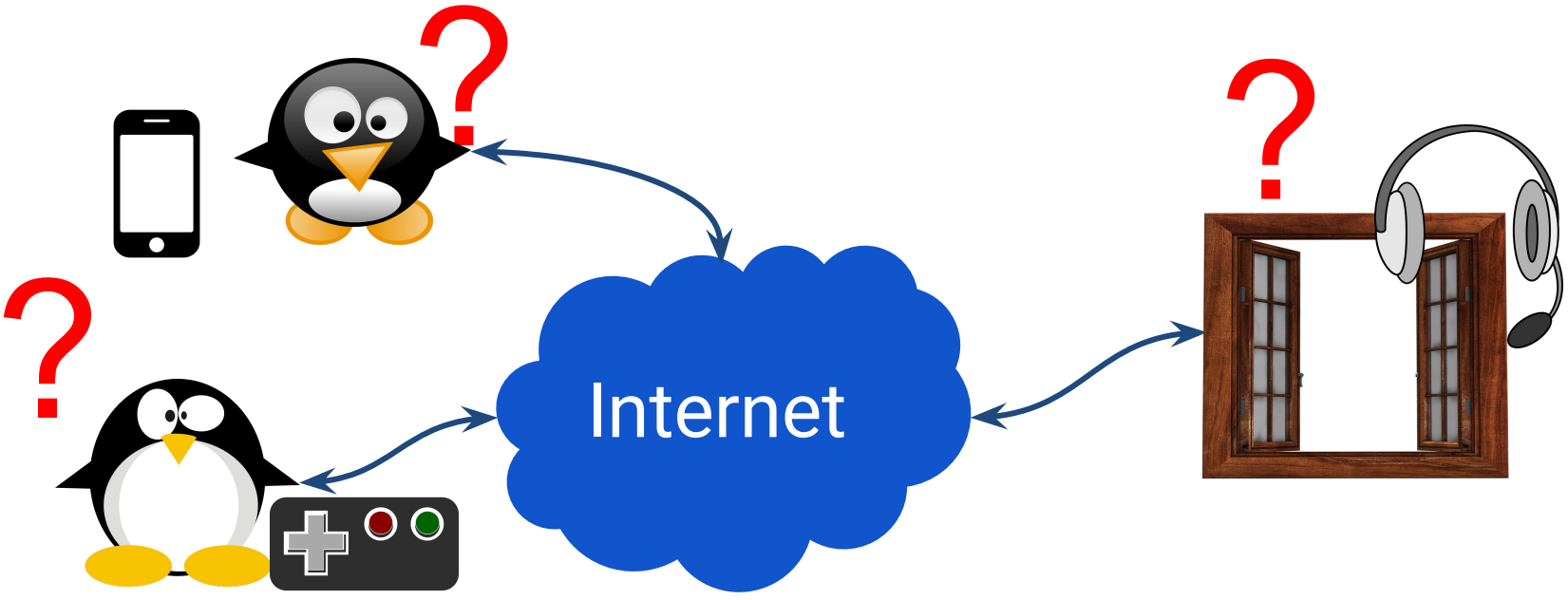
COMUNICAZIONE IN TEMPO REALE



Due o più utenti connessi che vogliono

- Instaurare una sessione di videoconferenza
- Scambiare dati (gaming, testo, IM, foto)

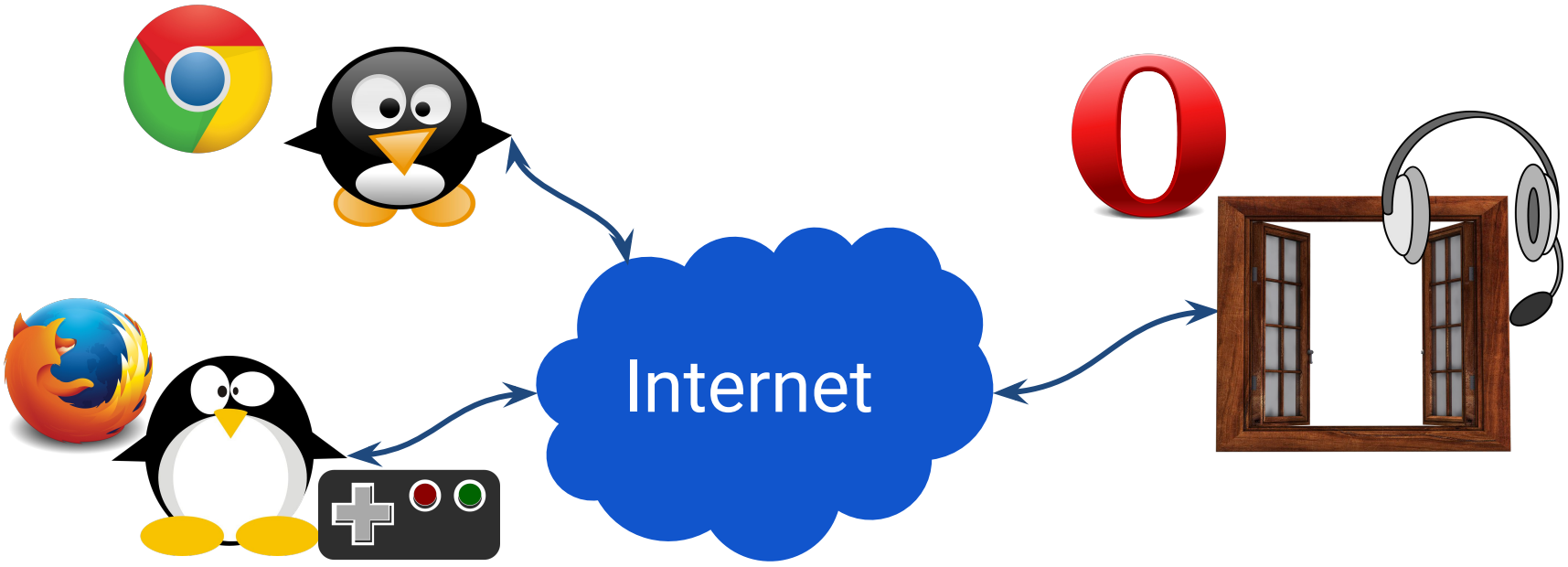
PROBLEMA



Che software usare?

- Deve essere supportato da qualunque O.S.
- Deve funzionare su qualunque dispositivo e rete

SOLUZIONE



Usare i browser come piattaforma unificatrice

L’INIZIATIVA WEBRTC

“La voce sarà semplicemente un’altra applicazione JavaScript”*

Henning Schulzrinne, CTO FCC, inventore SIP

“WebRTC rappresenta il nuovo fronte nella lunga guerra per un web aperto e libero”

Brendan Eich, inventore di JavaScript

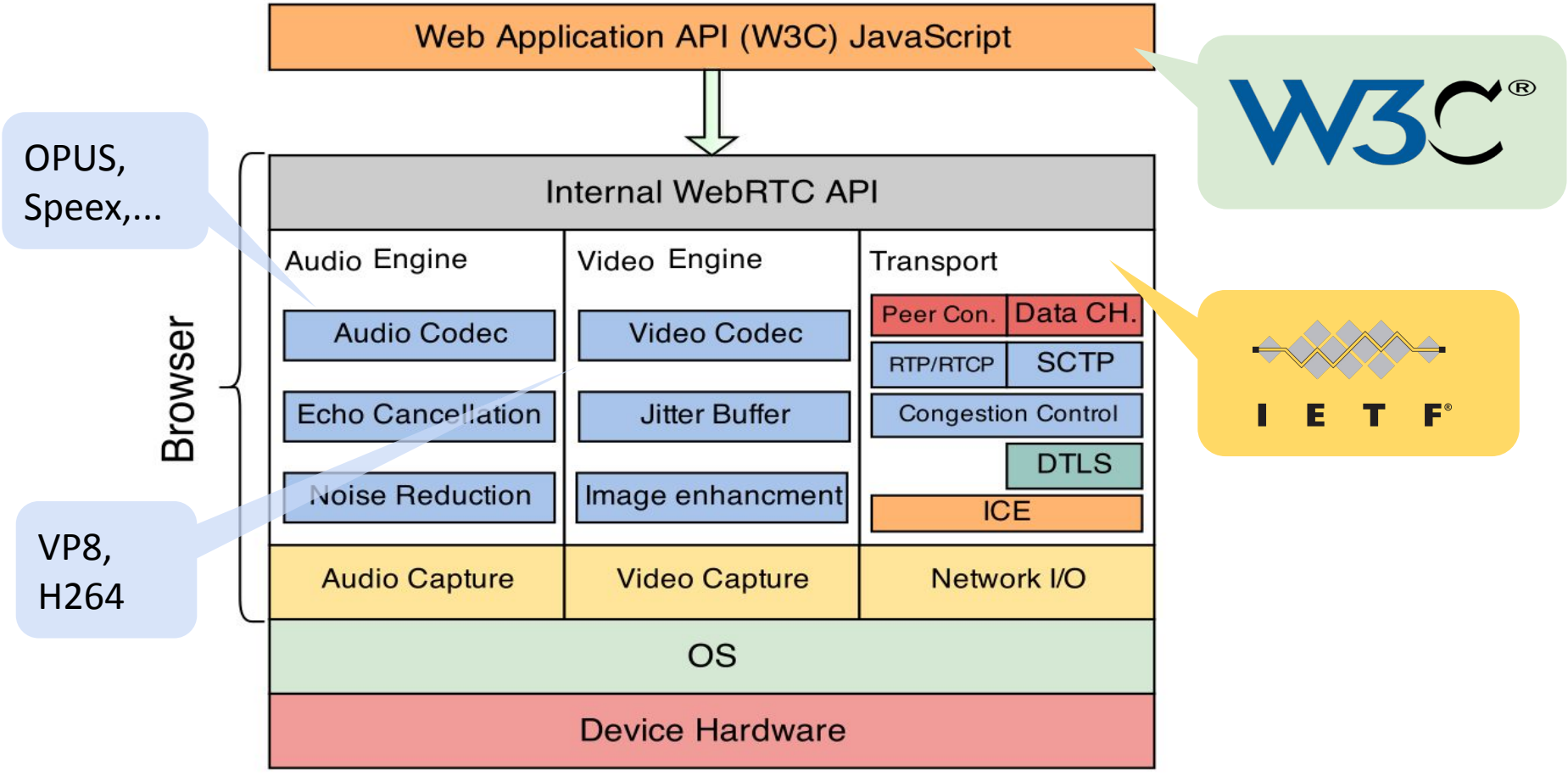
(*) Anche il video

Standardizza protocolli e architetture per nat traversal, controllo di congestione, sicurezza

Standardizza API per Web Browsers

Il browser implementa gli standard e molto altro software...

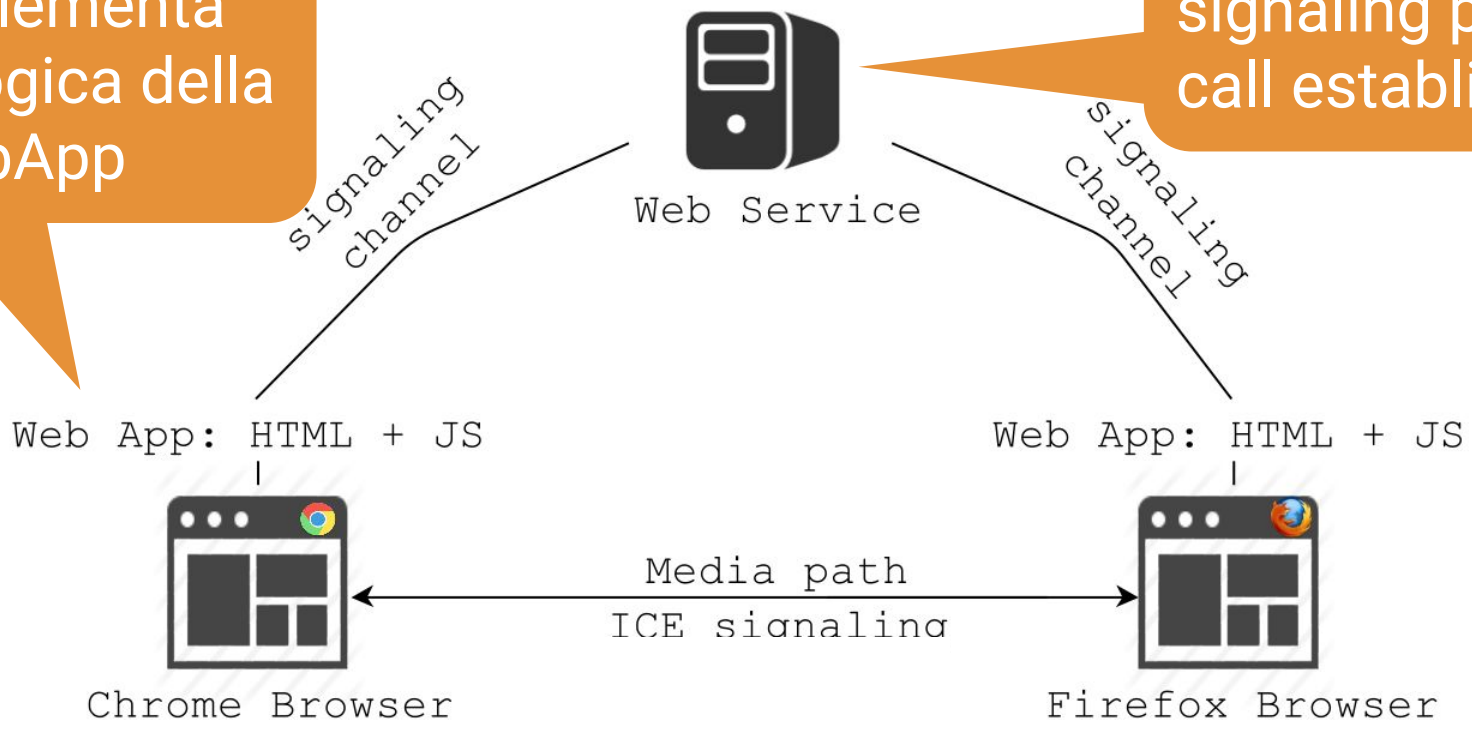
THE BIG PICTURE: NON SOLO PROTOCOLLI ED API



COME FUNZIONA (ARCHITETTURA): IL TRIANGOLO WEBRTC

Implementa la logica della WebApp

Gestisce il signaling per call establish.



Mario crea stanza su https://appr.tc

Nicola si connette sulla stanza creata da Mario

Mario e Nicola parlano!

CODE!

Video conferenza

MediaStream

Accesso a mic. e videocamera

PeerConnection

Gestione signaling di inizializzazione

DataChannel

API per creare canale dati p2p

Supporto

Trasporto dati

VIDEO CONFERENZA

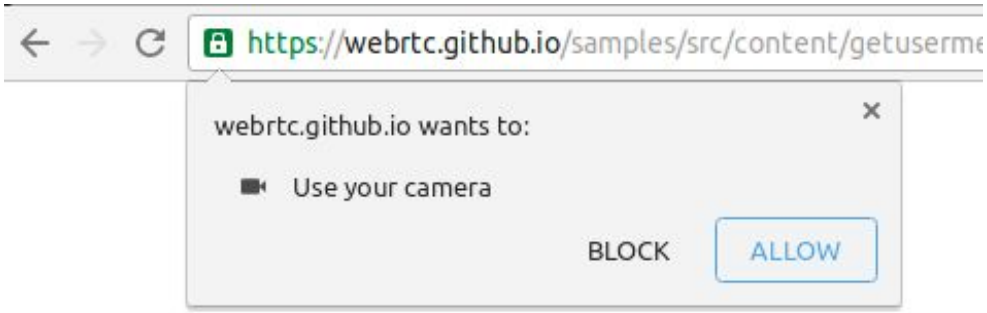
HTML: index.html

```
<html>
<head>...</head>
<body>
  <div id="container">
    <video id="local" autoplay></video>
    <video id="remote" autoplay></video>
    <div id="errorMsg"></div>
    <div id="okMsg"></div>
  </div>
  <script src="js/main.js"></script>
</body>
</html>
```

Elemento Video dove sara' mostrata la webcam locale

Elemento Video dove sara' mostrato il video dell'altro user

Logica applicazione



Le applicazioni devono chiedere permesso per usare mic/webcam

Json con restrizioni da imporre (risoluzione video, no video, no audio...)

Callback chiamata quando lo stream e' stato creato

```
navigator.getUserMedia(constraints, onSuccess, onError);
```

Callback chiamata quando si e' verificato un errore

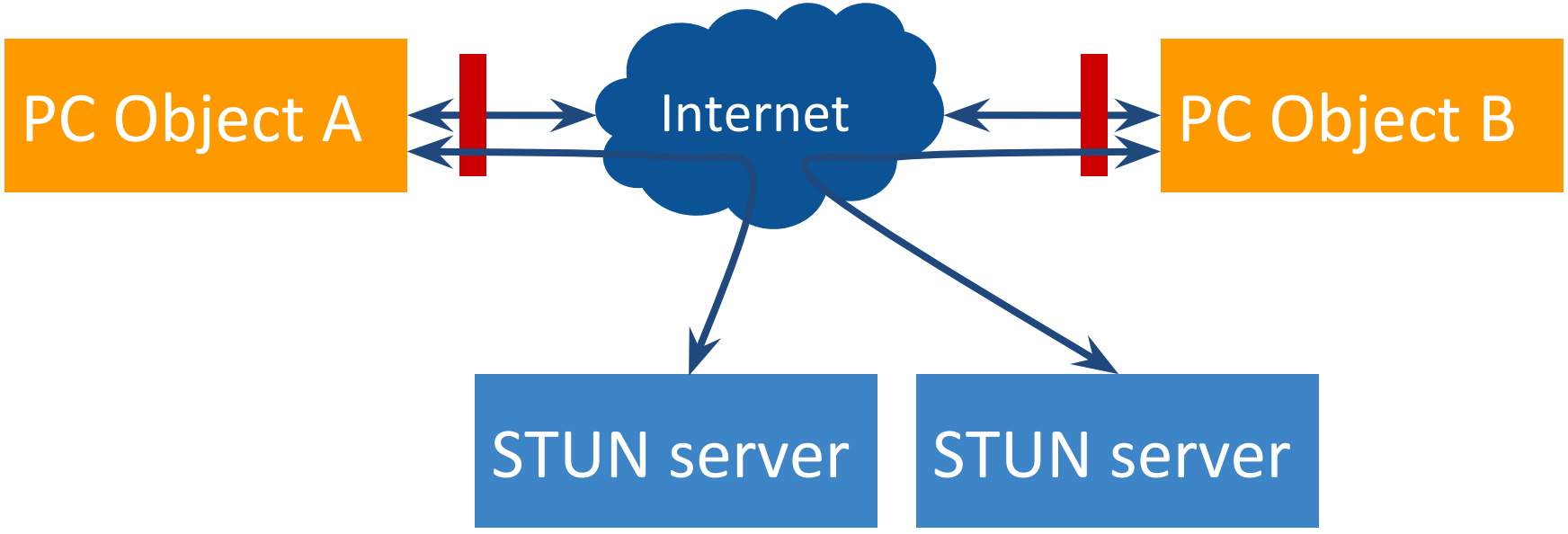
```
onGumSuccess = function (stream) {
   var video = document.getElementById('local');
   video.src = window.webkitURL.createObjectURL(stream);
   var msgdiv = document.getElementById('okMsg');
   msgdiv.innerHTML = '<p> OK MESSAGE!</p>';
};

onError = function (error) {
   console.log('getUserMedia error!', error);
   errordiv = document.getElementById('errorMsg');
   errordiv.innerHTML = '<p> Errore! ' + error.name + '</p>' ;
};

var constraints = { video: true, audio: true};
navigator.webkitGetUserMedia(constraints,onGumSuccess,onError);
```

MEDIASOURCE DEMO

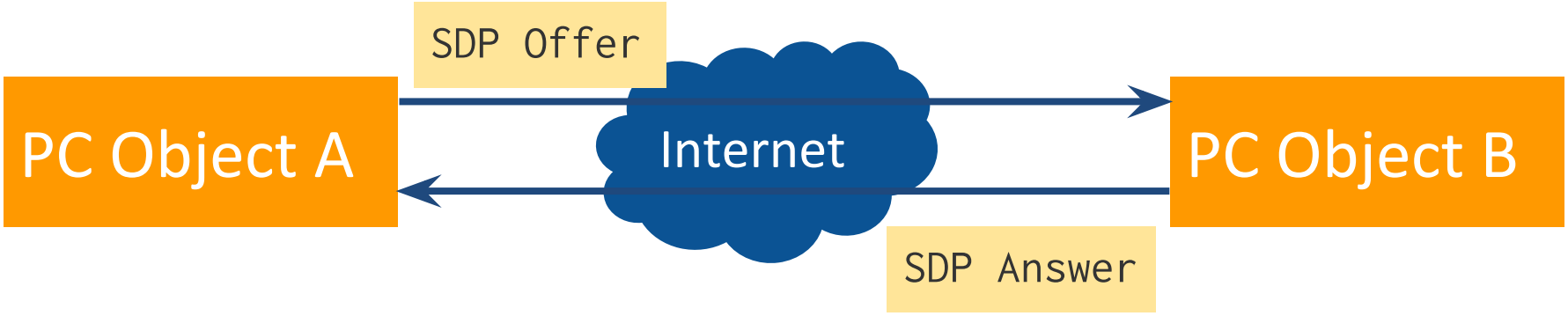
PEERCONNECTION: CONNESSIONE FRA PEER

- APIs per stabilire/gestire una connessione tra peer
- Gestisce il workflow **ICE** per il **NAT traversal**
- Mantiene traccia degli stream locali e remoti
- Gestisce la rinegoziazione automatica degli stream

- In pratica i peers si trovano dietro un NAT
- STUN permette di scoprire l'IP pubblico di un peer
- **ICE** (RFC5245) automatizza il processo



- PeerConnection usa *SDP* per la negoziazione e l'inizializzazione della connessione
- Cosa serve negoziare?
  - Codecs: video, audio
  - Protocolli: congestion control
  - ICE candidates per NAT traversal
  - ...



Chiamante

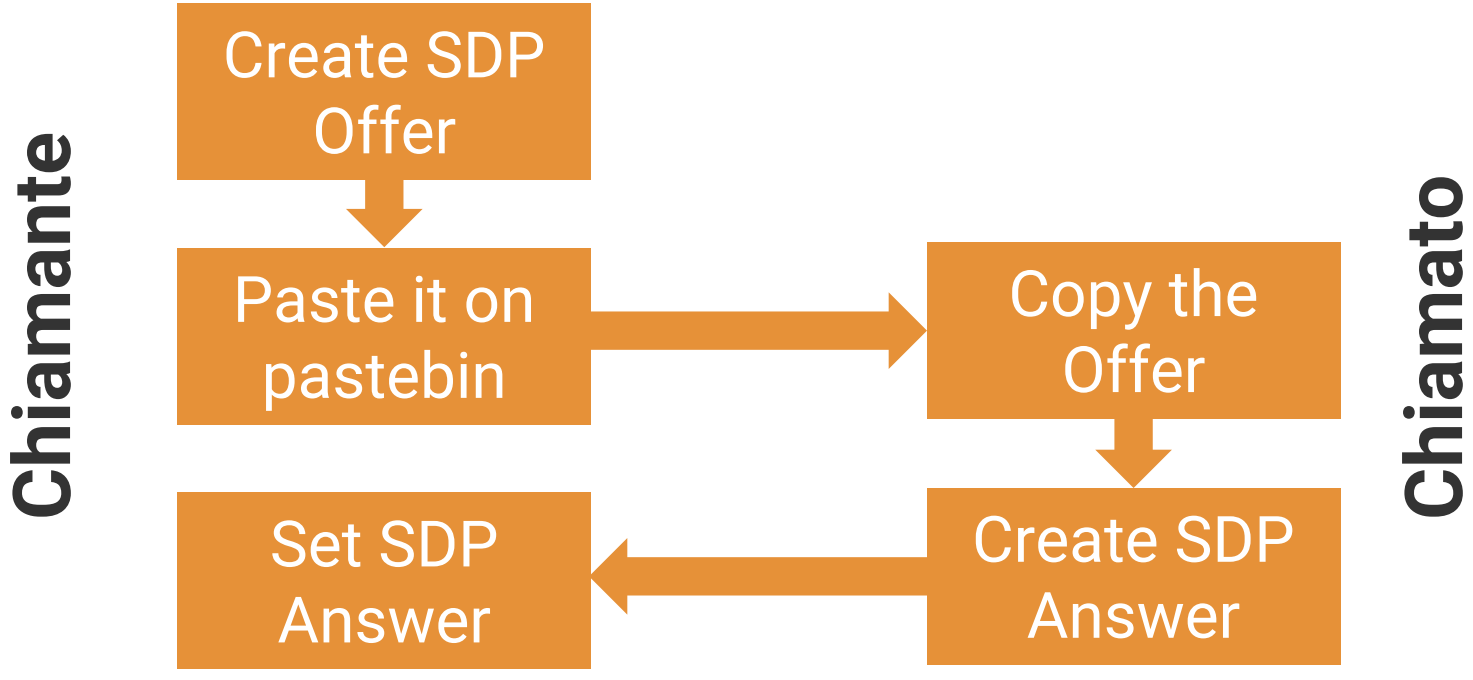
1. Creo PeerConnection
2. Aggiungo alla PC lo stream creato con getUserMedia
3. Creo una SDP Offer
4. Imposto l'SDP Offer
5. Invio la SDP Offer *
10. Imposto la SDP Answer sulla PC

Chiamato

1. Creo PeerConnection
6. Ricevo la *SDP Offer* dal chiamante
7. Imposto l'*SDP Offer*
8. Creo una *SDP Answer*
9. Imposto la *SDP Answer* sulla PC e la invio

* WebRTC non specifica come inviare il signaling

- Normalmente si usano dei server di segnalazione per creare *stanze* (appr.tc)
- In questo esempio non faremo uso di alcun server di segnalazione, useremo pastebin.com



INIZIALIZZAZIONE PEERCONNECTION (CALLER/CALLEE)

```
function initPeerConnection() {
  navigator.webkitGetUserMedia(constraints,
     function (stream) { //OnSuccess
        localStream = stream;
        var source = window.webkitURL.createObjectURL(stream);
        localVideo.src = source; //Set the stream as <video> source
        var servers = {'iceServers': [{'url': 'stun:stun.l.google.com:19302'}]};
        pc = new webkitRTCPeerConnection(servers); //1. Create PC
        pc.onicecandidate = function(e) { //Called on ICE candidate arrival
          onIceCandidate(pc, e);
        };
        pc.onaddstream = gotRemoteStream; //When the connection is ready call this!
        pc.addStream(localStream); //2. Set Local Stream
     },
     function (error) { //OnError
        console.log('getUserMedia error! Got this error: ', error);
     }
  );
}
```

LOGICA CHIAMANTE

```
//Create the SDP Offer
function call() {
    pc.createOffer(offerOptions).then( //3. Create SDP offer
    function (desc) { //on success
       pc.setLocalDescription(desc).then(//4. Set SDP offer
          function() { }, //on success do nothing
          onSetSessionDescriptionError //on Error
       );
   }, onCreateSessionDescriptionError //on error
  );
}

//Show the SDP offer in the webpage (update every time a new candidate is received)
function onIceCandidate(pc, event) {
    msgdiv.innerHTML = '<pre>' + pc.localDescription.sdp + '</pre>';
}

//Set the SDP answer on the PC (received no matter how)
function setSdpAnswer() {
  var sdpAnswer = new RTCSessionDescription({type: 'answer', sdp: SdpText.value});
  pc.setRemoteDescription(sdpAnswer).then( //10. Set the received SDP answer
    function() { }, //on success do nothing
    onSetSessionDescriptionError //on error
  );
}
```

LOGICA CHIAMATO

```
//To be called once the SDP offer has been received (no matter how)
function setSdpOffer() {
  //Create the sdp offer by taking it from a textarea (SdpText)
  sdpOffer =  new RTCSessionDescription({type: 'offer', sdp: SdpText.value});
  pc.setRemoteDescription(sdpOffer).then( //7. Set the SDP offer
    function() { // on Success (setRemoteDescription)
      pc.createAnswer().then( //8. Create SDP answer
        function (desc) { //on Success (createAnswer)
          msgdiv.innerHTML = '<pre>'+desc.sdp+'</pre>'; //set the answer in a div
          pc.setLocalDescription(desc).then( //9. Set the answer on PC
            function() {}, //do nothing on success (setLocalDescription)
            onSetSessionDescriptionError //on error (setLocalDescription)
          );
        }, onCreateSessionDescriptionError //on error (createAnswer)
      );
    },
    onSetSessionDescriptionError //on error (setRemoteDescrition)
  );
}
```

PEERCONNECTION DEMO

chrome://webrtc-internals

WebRTC Internals

chrome://webrtc-internals

▶ Create Dump

https://apprtc.appspot.com/r/155739826 [6515-1]

GetUserMedia Requests

https://apprtc.appspot.com/r/155739826, { servers: [turn:64.233.165.127:19305?transport=udp, turn:[2A00:1450:4010:C08::7F]:19305?transport=udp, turn:64.233.165.127:443?transport=tcp, turn:[2A00:1450:4010:C08::7F]:443?transport=tcp, stun:stun.l.google.com:19302], iceTransportType: all, bundlePolicy: max-bundle, rtcpMuxPolicy: require },

Time	Event
9/22/2016, 10:54:38 AM	▶ addStream
9/22/2016, 10:54:38 AM	▶ createOffer
9/22/2016, 10:54:38 AM	onRenegotiationNeeded
9/22/2016, 10:54:38 AM	▶ createOfferOnSuccess
9/22/2016, 10:54:38 AM	▶ setLocalDescription
9/22/2016, 10:54:38 AM	▶ signalingStateChange
9/22/2016, 10:54:38 AM	setLocalDescriptionOnSuccess
9/22/2016, 10:54:38 AM	▶ iceGatheringStateChange
9/22/2016, 10:54:38 AM	▶ onIceCandidate
9/22/2016, 10:54:38 AM	▶ onIceCandidate

PeerConnection Events

DATA CHANNELS

INIZIALIZZAZIONE PEERCONNECTION (CALLER/CALLEE)

```
function initPeerConnection() {
   var servers = {'iceServers': [{'urls': 'stun:stun.l.google.com:19302'}];
   var  pcConstraints = {'mandatory':{'OfferToReceiveAudio': false,
                                      'OfferToReceiveVideo': false }};//no audio/video
   pc = new webkitRTCPeerConnection(servers, pcConstraint);
   pc.onicecandidate = function(e) {
      onIceCandidate(pc, e);
   };
   pc.ondatachannel = dataChannelCallback;
   pc.oniceconnectionstatechange = function(e) {
     onIceStateChange(pc, e);
   };
}

//Called when the DC becomes ready
function dataChannelCallback(event) {
        if (dataChannel === null) { //Callee
                dataChannel = event.channel; //Set the DC = to the event channel
                dataChannel.onmessage = messageCallback;
        }
}
```

LOGICA CHIAMANTE

```
//Create the SDP Offer
function call() {
   dataConstraint = null;
   dataChannel = pc.createDataChannel('LinuxDayRules', dataConstraint);
   dataChannel.onmessage = messageCallback; //When data arrives!
   dataChannel.onopen = function (e) {console.log("Data Channel open!", e);}
   pc.createOffer().then(
    function (desc) { //on success
        pc.setLocalDescription(desc).then(
          function() { console.log("Create offer success"); },
          onSetSessionDescriptionError
        );
   }, function (e) { console.log("Error", e); } //on error
  );
}
//Show the SDP offer in the webpage (update every time a new candidate is received)
function onIceCandidate(pc, event) {
    msgdiv.innerHTML = '<pre>' + pc.localDescription.sdp + '</pre>';
}

//Set the SDP answer on the PC (received no matter how) identical to the PC case
```

LOGICA CHIAMATO: IDENTICA A PEERCONNECTION

```
//To be called once the SDP offer has been received (no
matter how)
function setSdpOffer() {
   //Exactly like the PC case
}
```

Quando il canale e' pronto (**`dataChannel.onopen`** e' stata chiamata) e' possibile inviare dati

```
dataChannel.send(data)
```

Quando si riceve un messaggio viene chiamata la callback **`dataChannel.onmessage`**

```
function messageCallback(event) {
        console.log("Got data", event);
}
```

DATACHANNEL DEMO

GESTIRE LE DIFFERENTI IMPLEMENTAZIONI DELLE API

W3C Standard	Chrome	Firefox
getUserMedia	webkitGetUserMedia	mozGetUserMedia
RTCPeerConnection	webkitRTCPeerConnection	RTCPeerConnection
RTCSessionDescription	RTCSessionDescription	RTCSessionDescription
RTCIceCandidate	RTCIceCandidate	RTCIceCandidate

Soluzione: utilizzare uno ***shim layer***

Repo git: https://github.com/webrtc/adapter
Shim layer: http://webrtc.github.io/adapter/adapter-latest.js

Abbiamo scalfito la superficie

- Accedere ai device di cattura: **getUserMedia**
- Creare una sessione di video conferenza
- Creare una video chat con **DataChannel**

Slides disponibili su:

http://c3lab.poliba.it/LDC

Codice disponibile su:

https://github.com/ldecicco/webrtc-demos

Come andare avanti?

https://github.com/webrtc/samples

http://webrtc.org

Politecnico di Bari
Dipartimento di Ingegneria Elettrica e dell'Informazione

C3LAB
Control of Computing and Communication Systems Lab

HAPPY HACKING!

Luca De Cicco
luca.decicco@poliba.it