

Open APIs
for Open
Minds

Kurento Real Time Media Stream Processing

Juan Ángel Fuentes

Software Developer. Stream Oriented GE

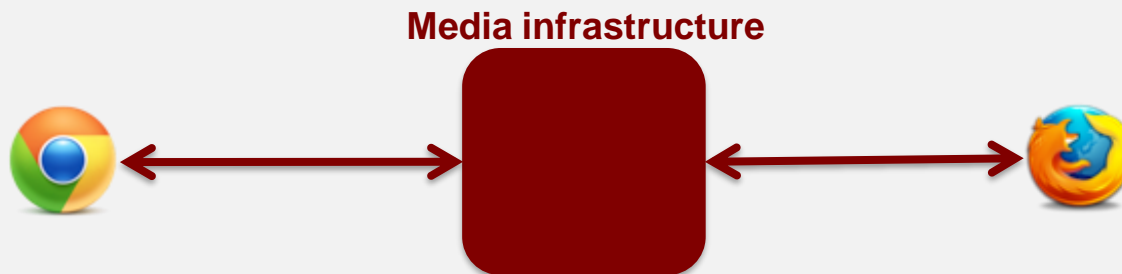
jafuentes@naevatec.com

Introducing multimedia infrastructures

Peer-to-Peer Application (without media infrastructure)

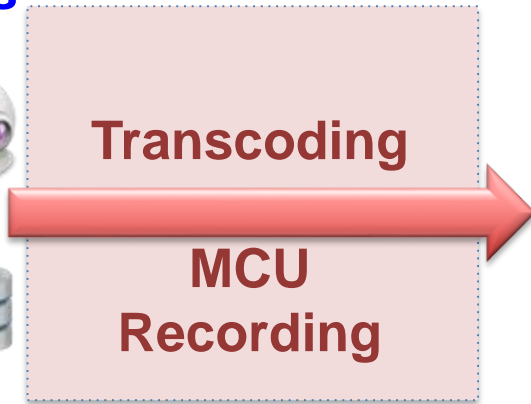


Application with media infrastructure



Multimedia infrastructures for the Future Internet

Media is here



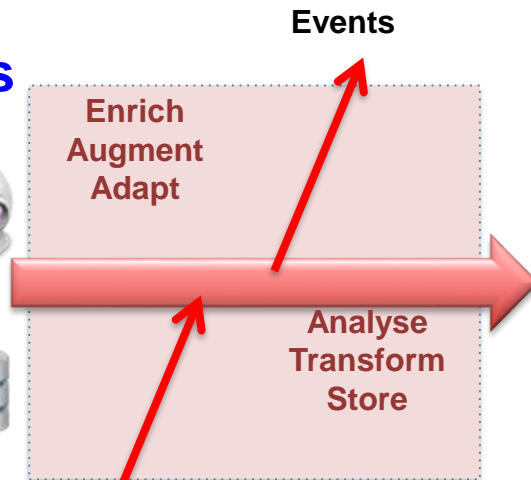
Media got there



Traditional infrastructures

Future Internet infrastructures

Media is here



Events

Media got there



Sensors
Context

The Stream Oriented Generic Enabler – SOGEE

Multimedia infrastructure

- Interoperable media exchange (multiplatform/multiprotocol)
 - WebRTC, RTP, HTTP (video tag), etc.
- Standard capabilities
 - Transcoding, MCU, recording
- Advanced capabilities
 - Computer vision, augmented reality, mixing, blending, etc.

APIs

- REST API
- JavaScript API
- Java API

Is distributed through a flexible FOSS license

- LGPL 2.1

Kurento: The equation

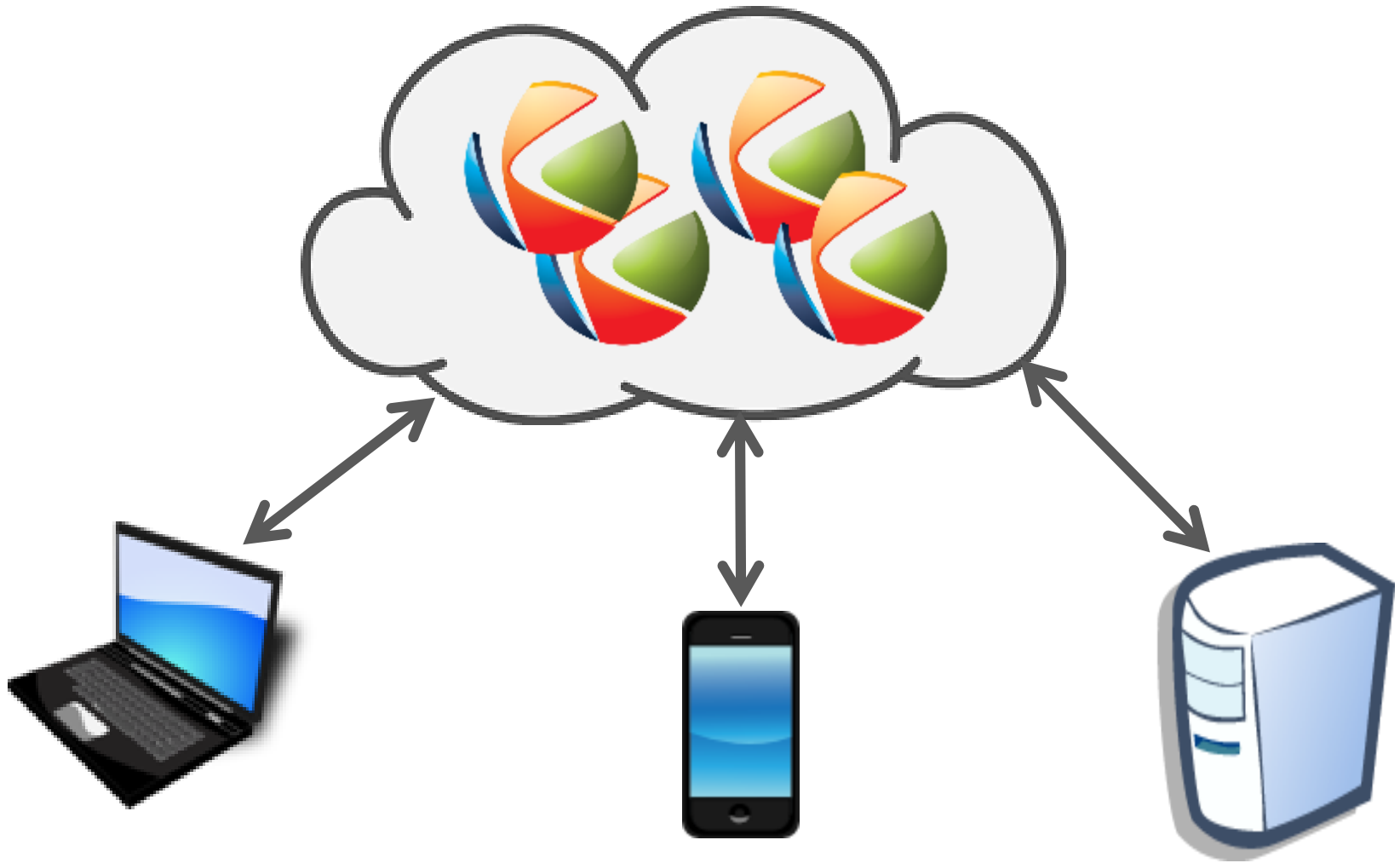
Future
Internet
Multimedia
Infrastructure



Simple
Development
APIs



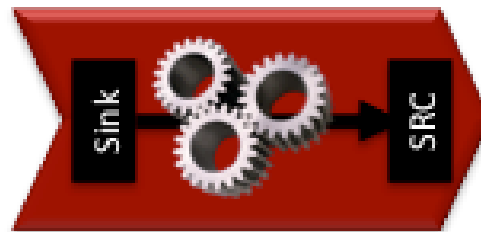
General Architecture





Key Concepts: Media Elements

- Functional unit performing a **specific action** on a media stream
- Developers **abstract** from the low level implementation
- Able to **receive media** form other elements
- Able to **send media** to other elements

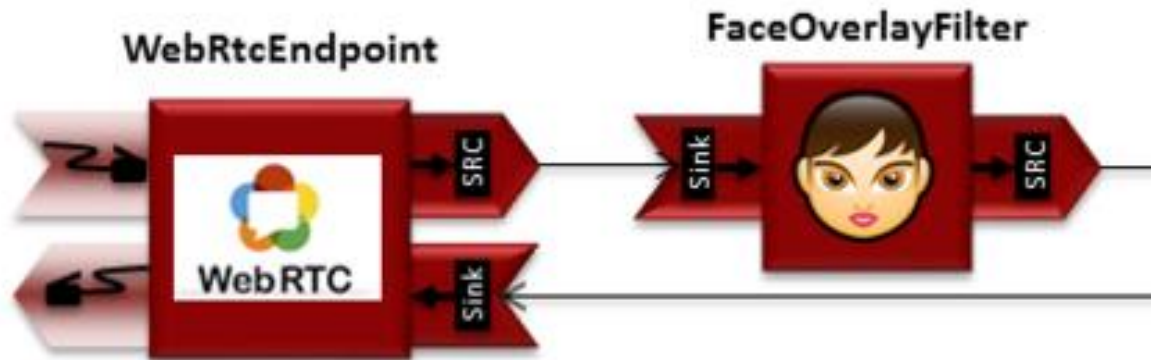


Key Concepts: Types of Media Elements

- **Input Endpoints:** injects media
- **Filters:** Transform or analyze media
- **Hubs:** Manage multiple media flows
- **Output Endpoints:** take the media stream out

Key Concepts: Media Pipeline

- **Media Pipeline:** is a chain of media elements, where the output stream generated by one element (source) is fed into one or more other elements input streams (sinks)

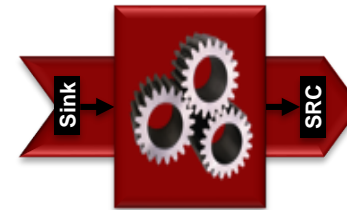


Key concepts: media elements and pipelines

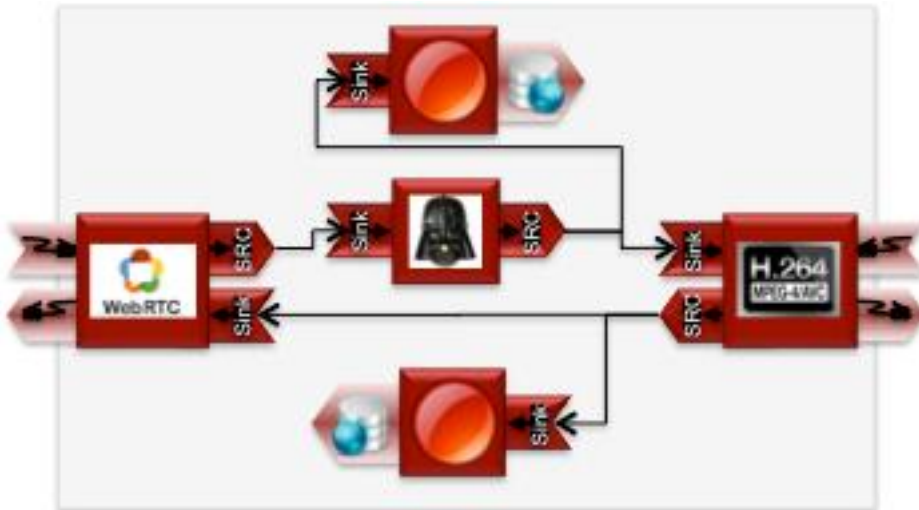
■ Media Element

- Provides a specific media functionality
 - › Send/receive media. These are the Endpoints
 - › Process media
 - › Transform media
- Ready to be used
- New media elements can be added

Media Element

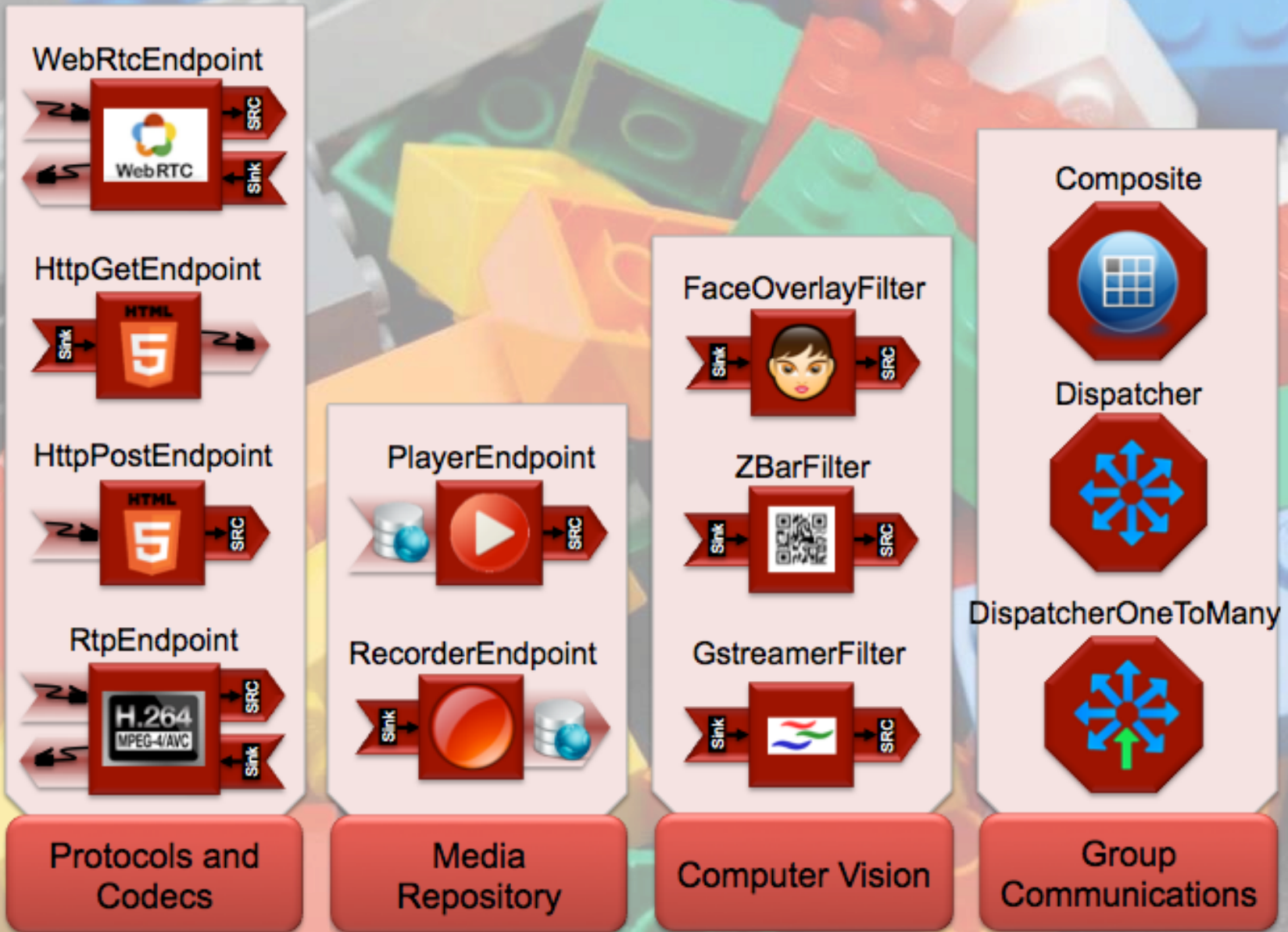


Media Pipeline



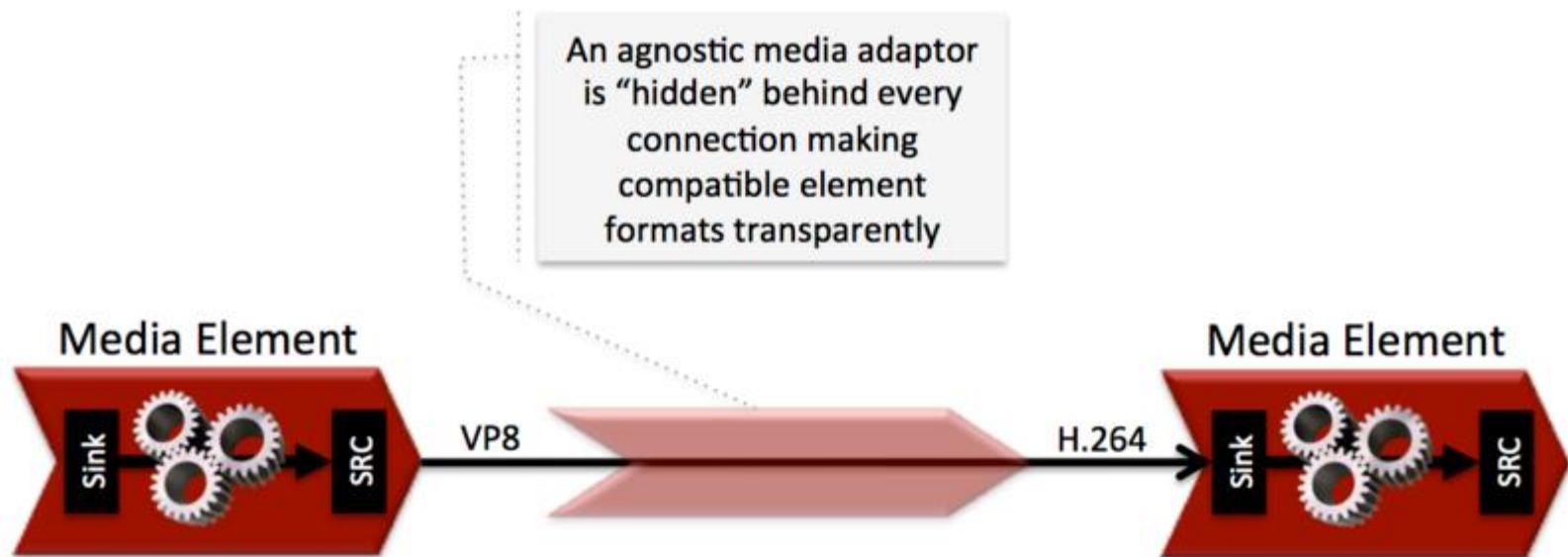
■ Media pipeline

- Chain of media elements implementing the desired media logic
- The Media Server provides the capability of creating media pipelines by joining media elements of the toolbox

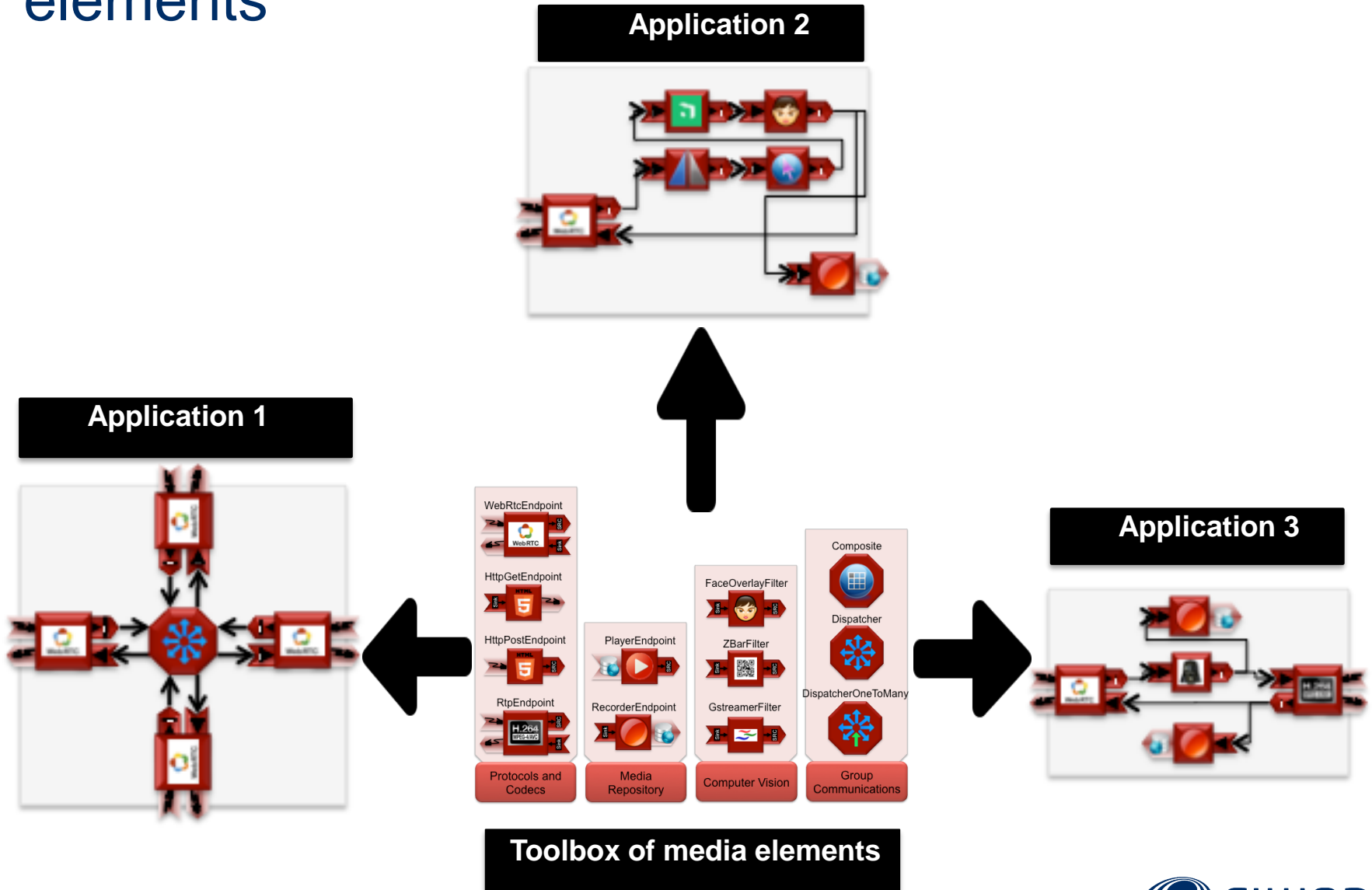


Agnostic Media adapter

- In charge of making possible that Stream Oriented GE APIS **allow developers to combine media elements** to create the desired pipeline
- Fully abstracts all the complexities of media codecs and formats

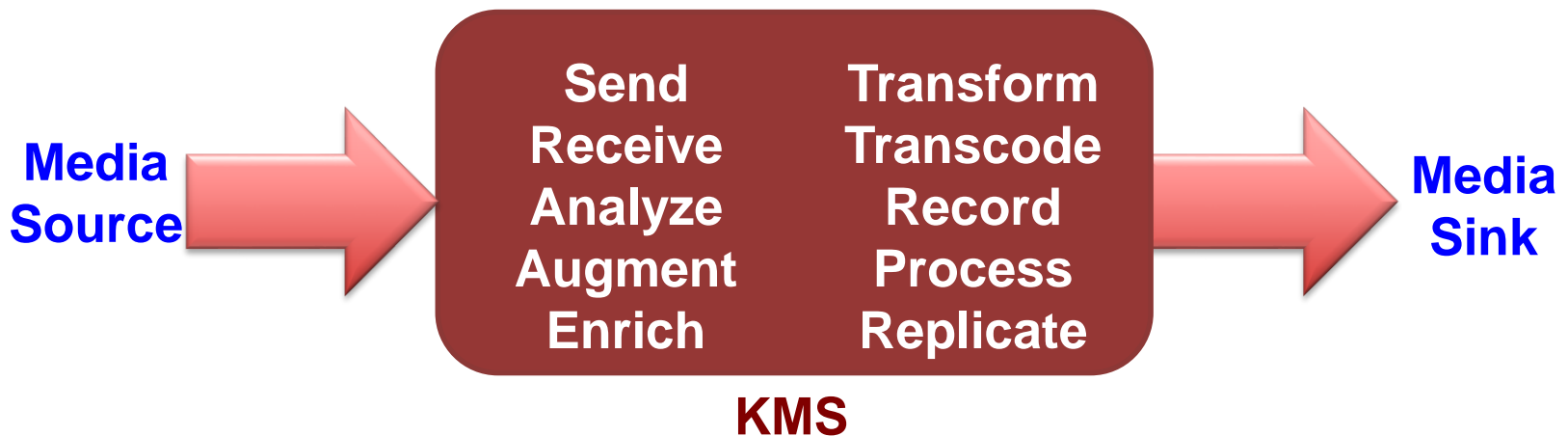


Developers create applications just connecting elements

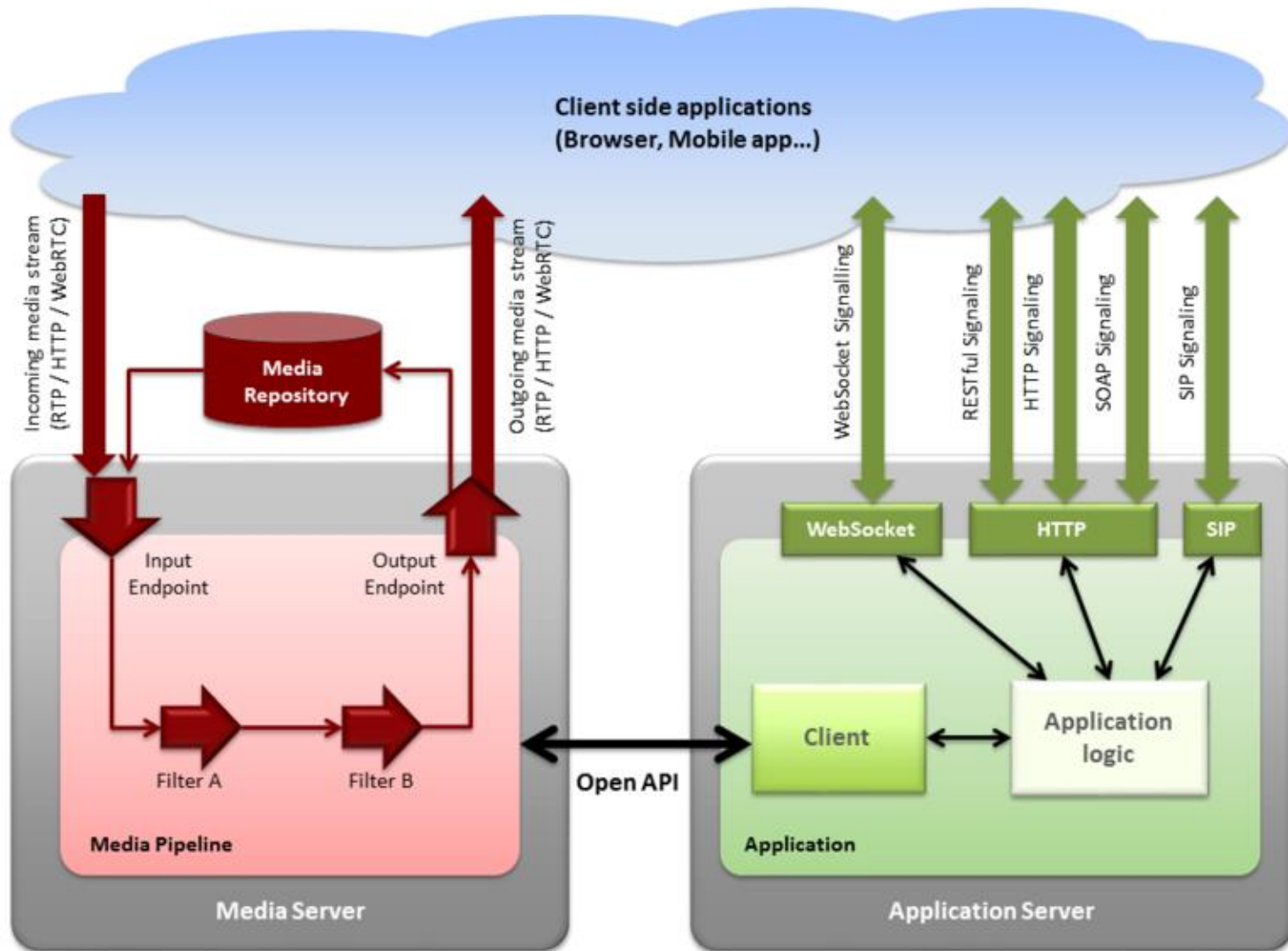


Media Server (KMS): The nucleus of Kurento

- **KMS is a middleware for media streams:**
- **Receives the stream**
- **Process the stream**
- **Issues the stream**

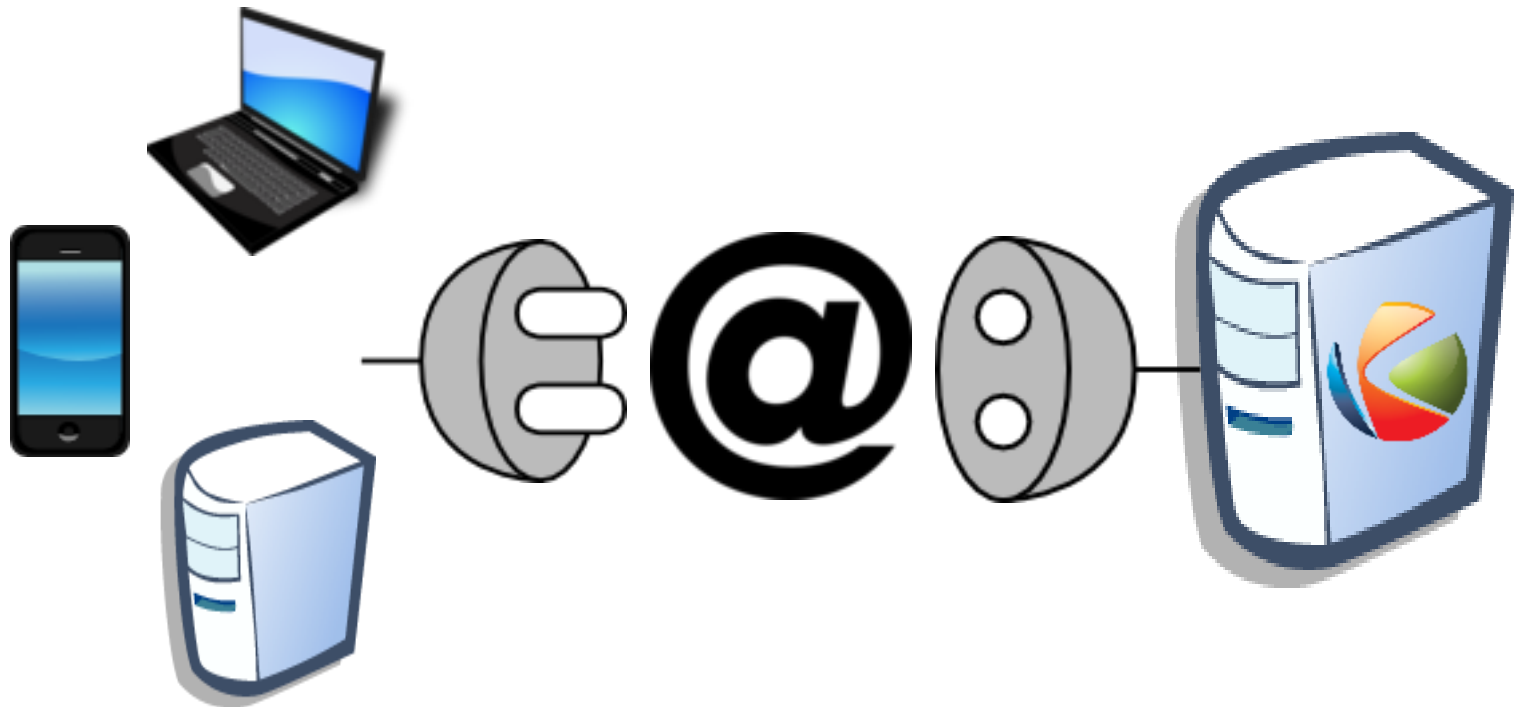


Media plane and Signaling plane



How to control the media server?

- REST API
- JSON RPC
- Websocket interface in KMS



API Implementations

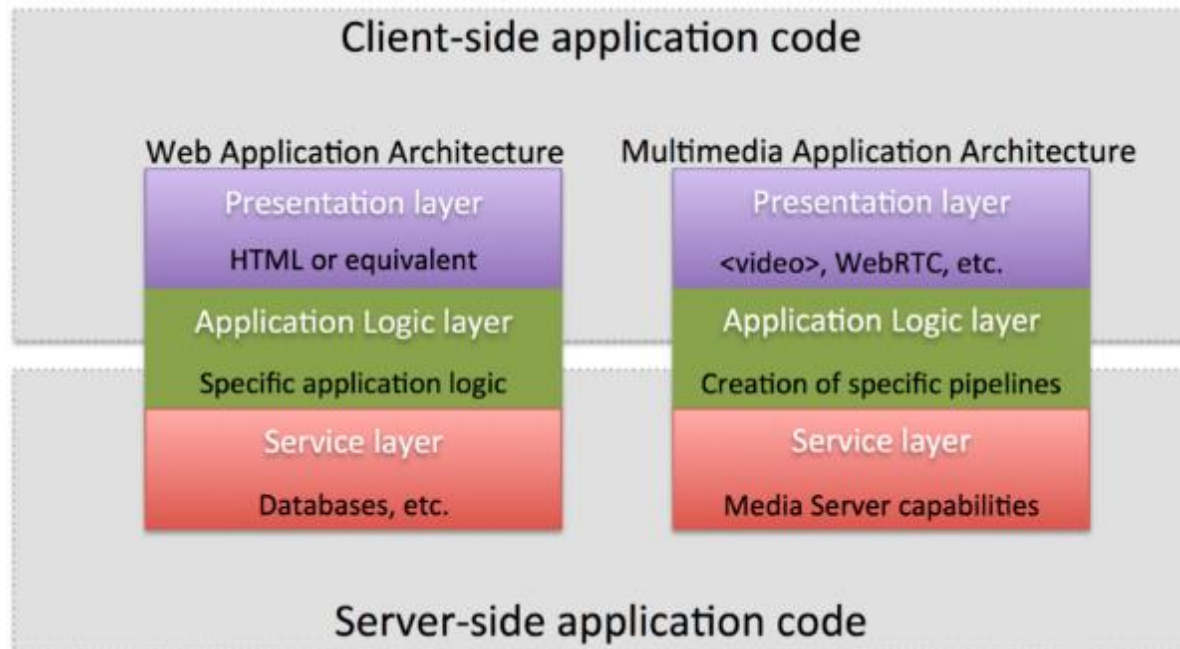


For now...

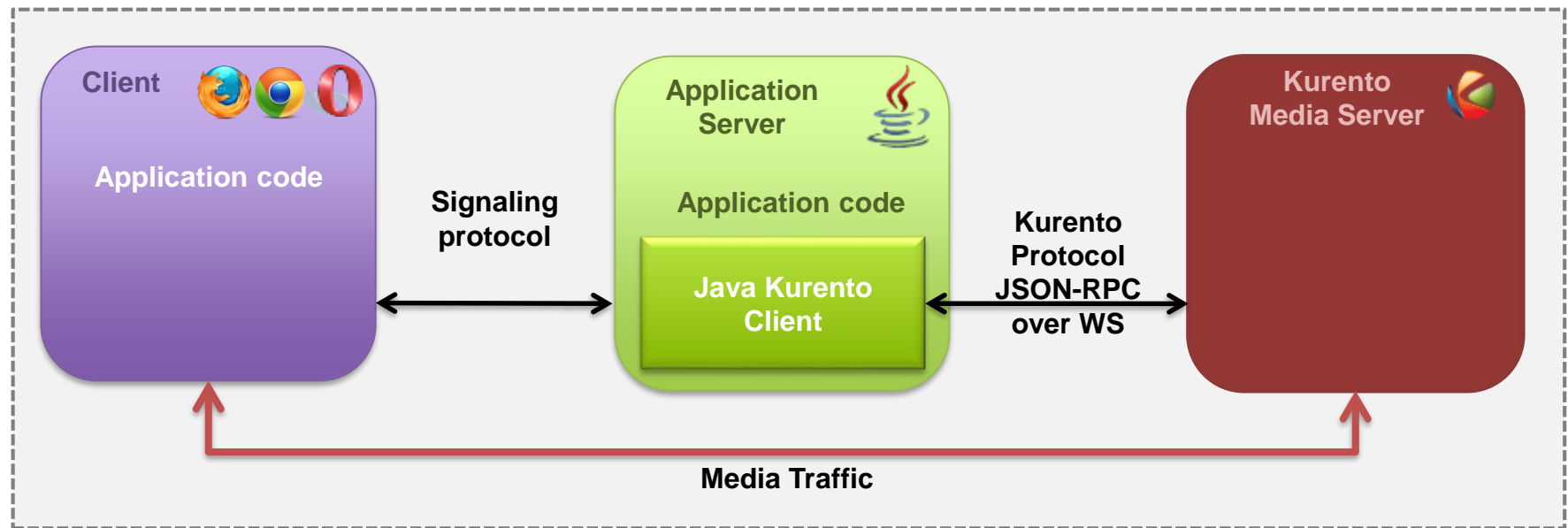
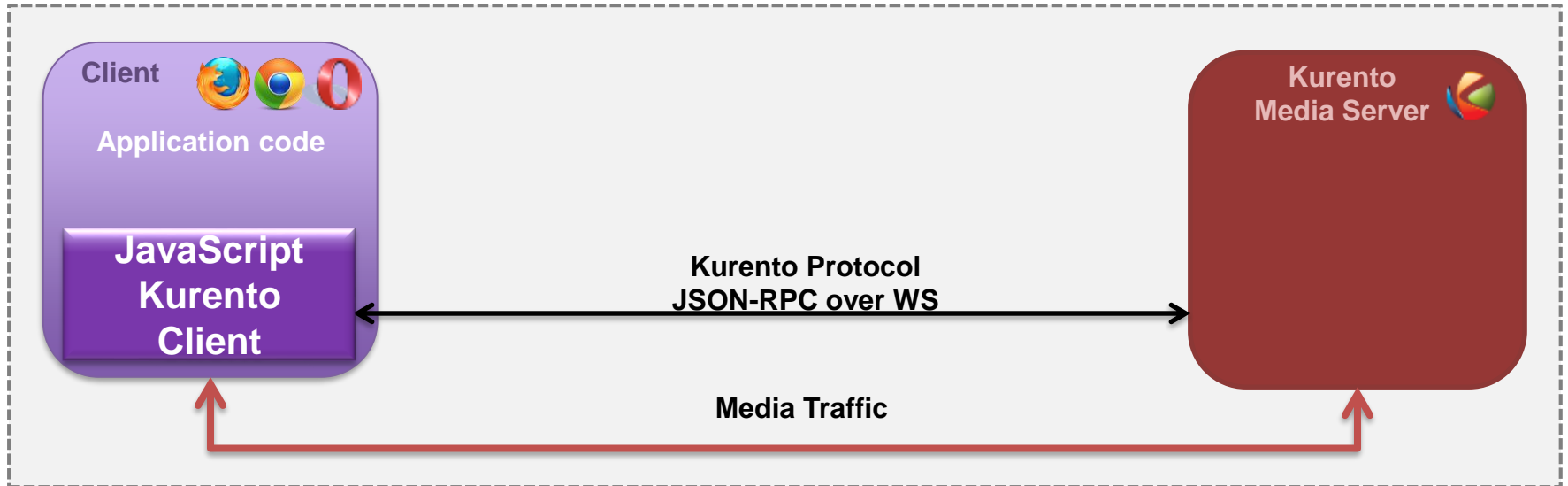
Multimedia Application Architecture in SO GE

Web application three layers parallelism:

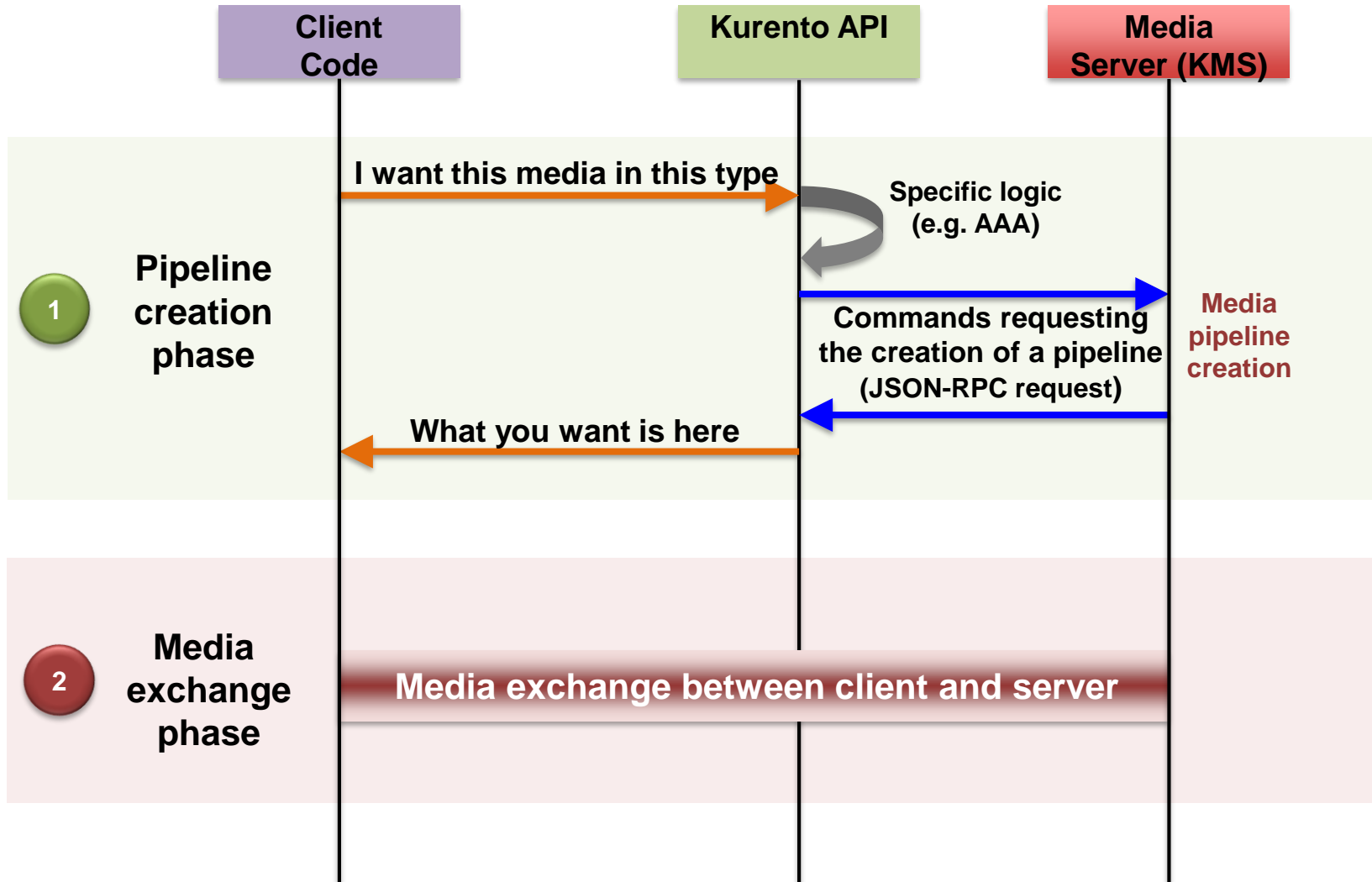
- **Presentation layer:** Multimedia presentation and capturing
- **Application logic:** Multimedia logic in charge of building the pipeline
- **Service Layer:** Kurento Media Server



Kurento Applications



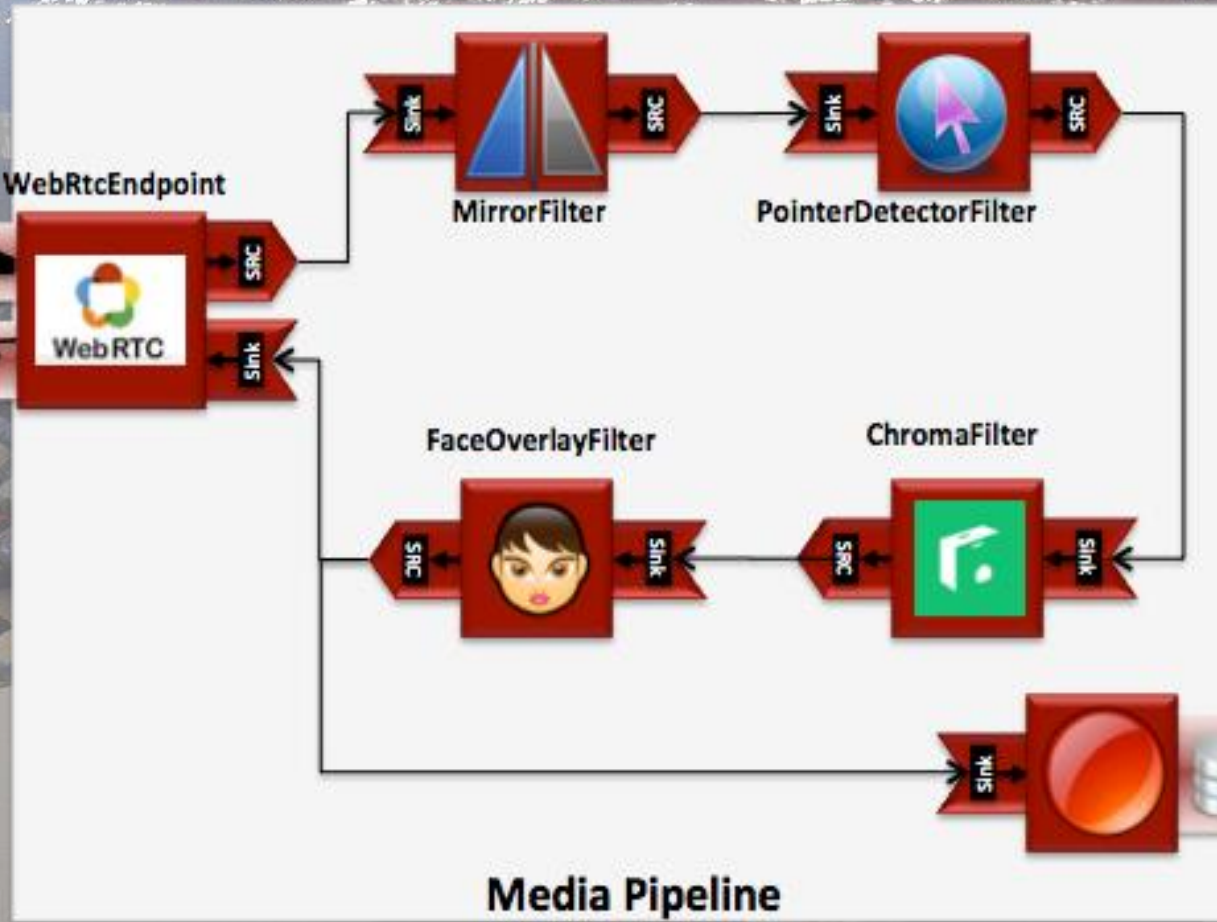
Application execution flow



Campus Party Brazil



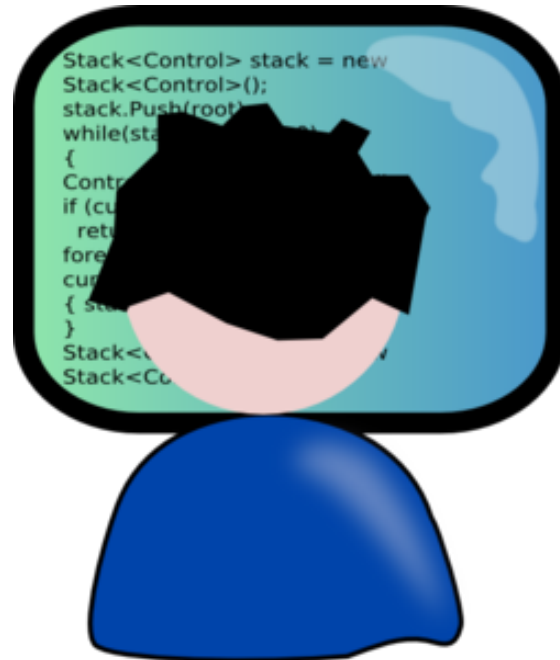
Campus Party Brazil





Let's get started!

Tutorials

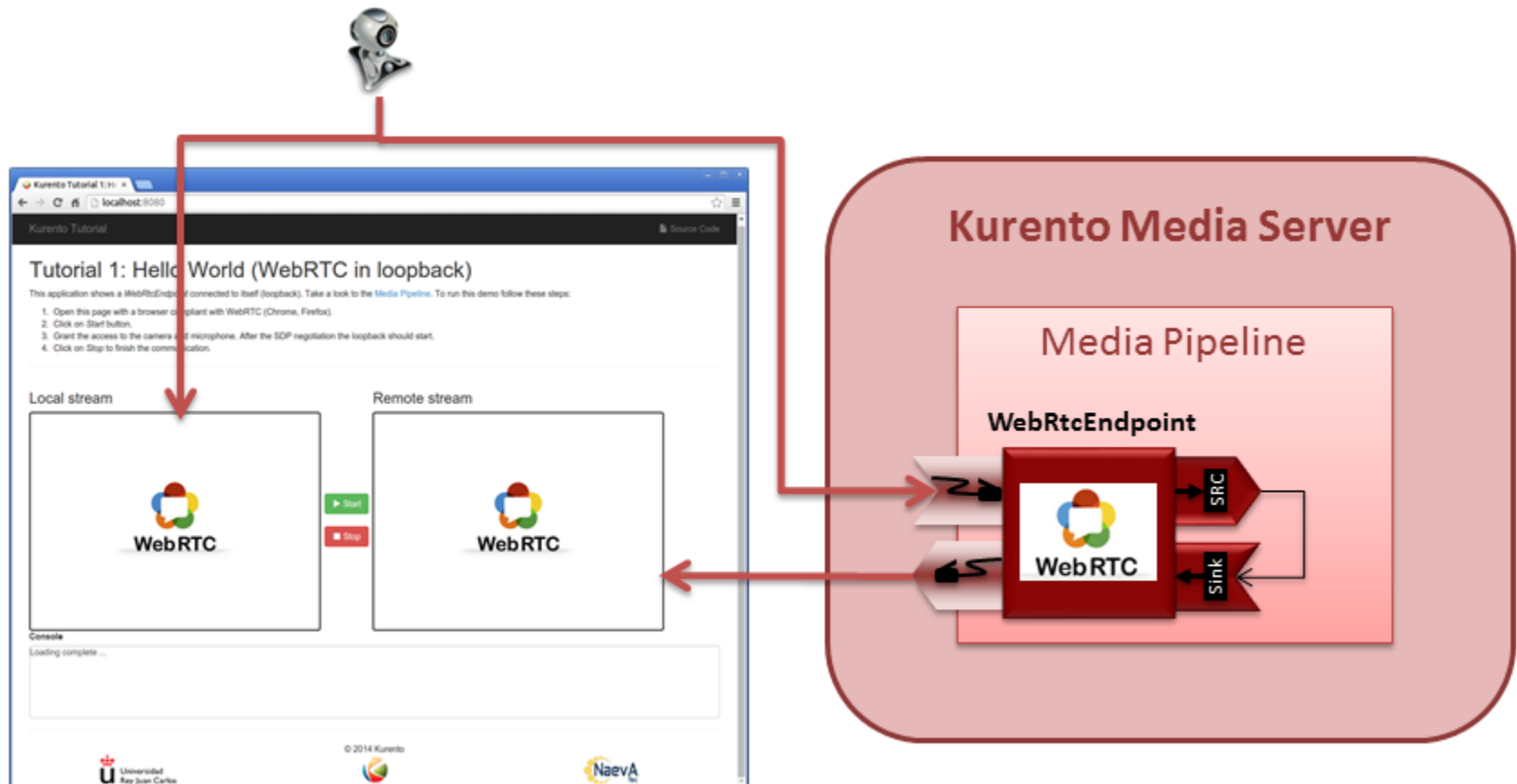


<http://doc-kubernetes.readthedocs.io/en/stable/tutorials.html>

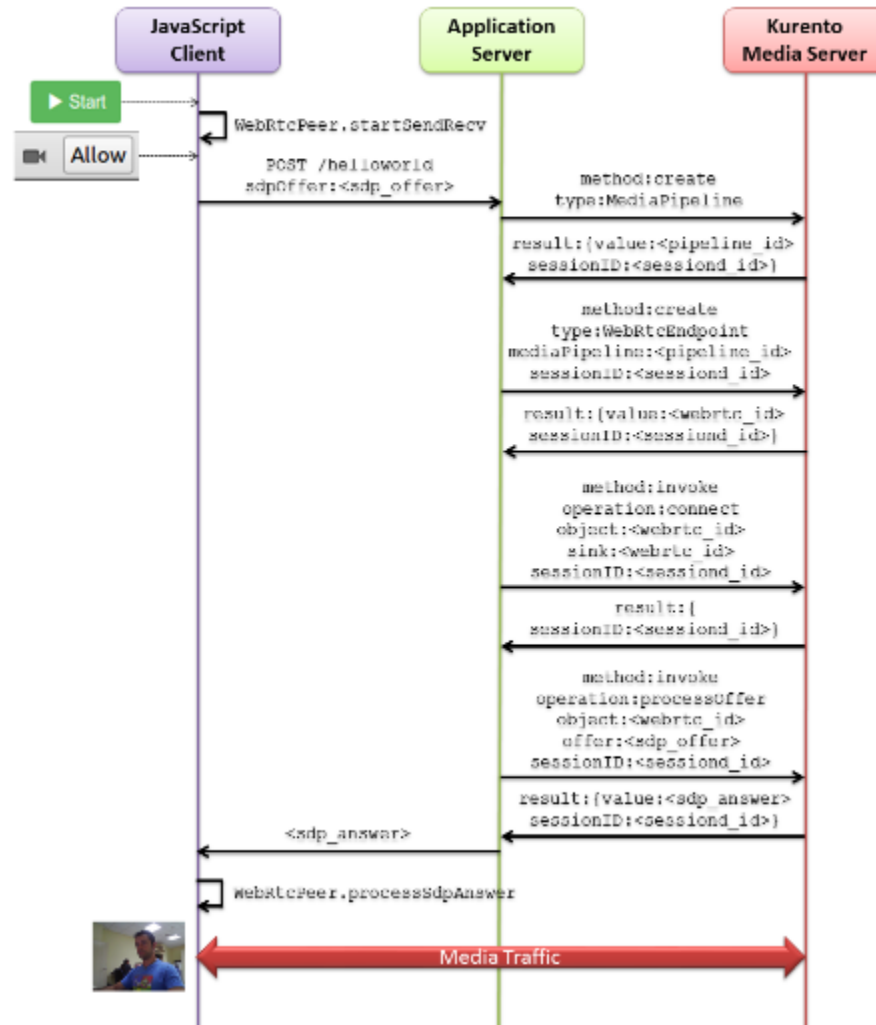
What do we need?

- **An instance of the Kurento Media Server**
 - **FIWARE-LAB**
 - **Local installation**
- **Maven**
- **NPM**
- **Bower**
- **Your favorite IDE**

Tutorial 1 - Hello world



Tutorial 1 - Hello world



Tutorial 1 - Hello world, server

```
@RestController
public class HelloWorldController {

    @Autowired
    private KurentoClient kurento;

    @RequestMapping(value = "/helloworld", method = RequestMethod.POST)
    private String processRequest(@RequestBody String sdpOffer)
        throws IOException {

        // Media Logic
        MediaPipeline pipeline = kurento.createMediaPipeline();
        WebRtcEndpoint webRtcEndpoint = new WebRtcEndpoint.Builder(pipeline)
            .build();
        webRtcEndpoint.connect(webRtcEndpoint);

        // SDP negotiation (offer and answer)
        String responseSdp = webRtcEndpoint.processOffer(sdpOffer);
        return responseSdp;
    }
}
```

Tutorial 1 - Hello world, client

```
var webRtcPeer;

function start() {
  console.log("Starting video call ...");
  showSpinner(videoInput, videoOutput);
  webRtcPeer = kurentoUtils.WebRtcPeer.startSendRecv(videoInput, videoOutput, onOffer, onError);
}

function onOffer(sdpOffer) {
  console.info('Invoking SDP offer callback function ' + location.host);
  $.ajax({
    url : location.protocol + '/helloworld',
    type : 'POST',
    dataType : 'text',
    contentType : 'application/sdp',
    data : sdpOffer,
    success : function(sdpAnswer) {
      console.log("Received sdpAnswer from server. Processing ...");
      webRtcPeer.processSdpAnswer(sdpAnswer);
    },
    error : function(jqXHR, textStatus, error) {
      onError(error);
    }
  });
}

function onError(error) {
  console.error(error);
}
```

Tutorial 1 - Hello world (Node.js)

```
var kurento = require('kurento-client');

//...

const ws_uri = "ws://localhost:8888/kurento";

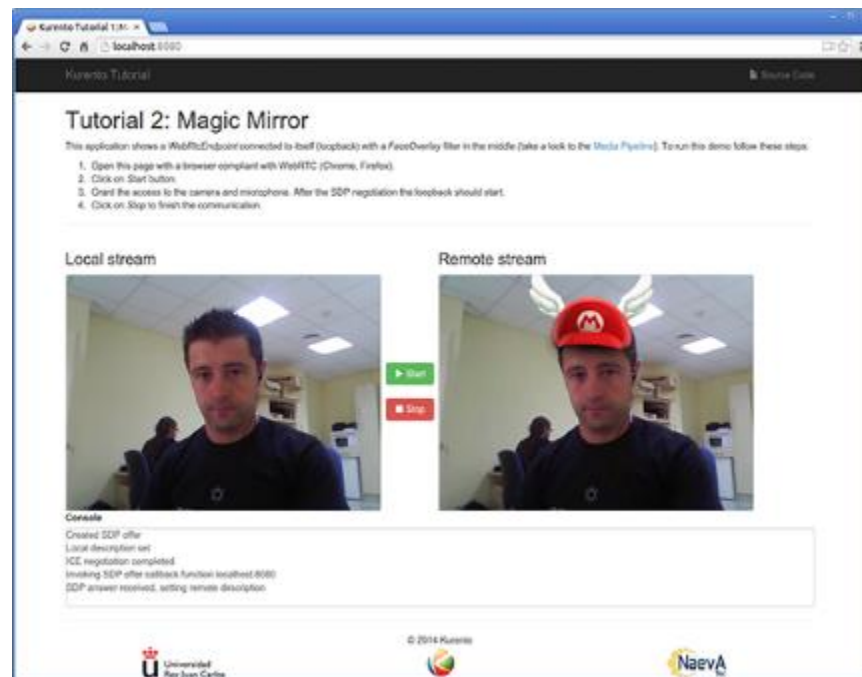
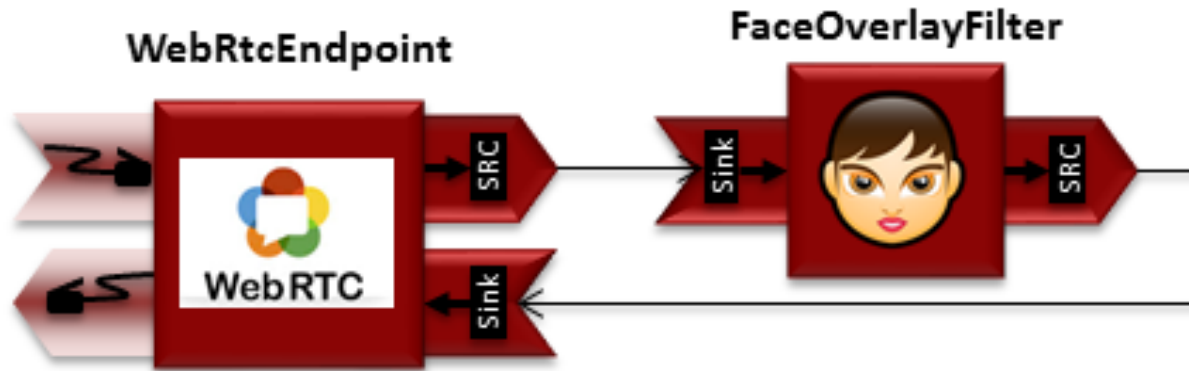
//...

kurento(ws_uri, function(error, kurentoClient) {
  if (error) {
    return callback(error);
  }
  kurentoClient.create('MediaPipeline', function(error, _pipeline) {
    if (error) {
      return callback(error);
    }
    pipeline = _pipeline;
    return callback(null, pipeline);
  });
});
```

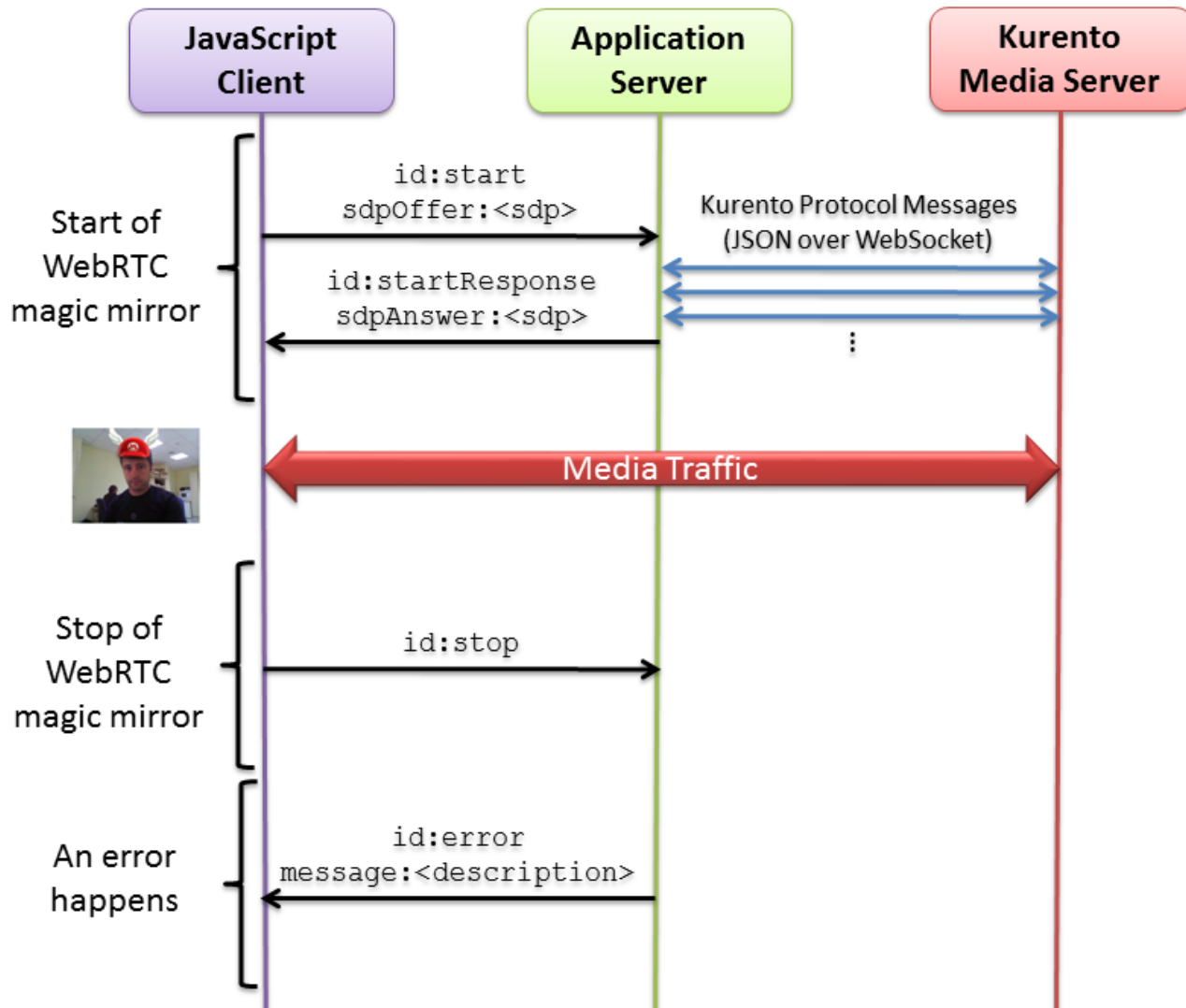

Tutorial 1 - Hello world (Node.js)

```
app.post('/helloworld', function(req, res) {
  var sdpOffer = req.body;
  getPipeline(function(error, pipeline) {
    pipeline.create('WebRtcEndpoint', function(error, webRtcEndpoint) {
      webRtcEndpoint.processOffer(sdpOffer, function(error, sdpAnswer) {
        webRtcEndpoint.connect(webRtcEndpoint, function(error) {
          res.type('application/sdp');
          res.send(sdpAnswer);
        });
      });
    });
  });
});
});
});
```

Tutorial 2 - WebRTC magic mirror



Tutorial 2 - WebRTC magic mirror



Tutorial 2 - WebRTC magic mirror

```
private void start(WebSocketSession session, JsonObject jsonMessage) {
    try {
        // Media Logic (Media Pipeline and Elements)
        MediaPipeline pipeline = kurento.createMediaPipeline();
        pipelines.put(session.getId(), pipeline);

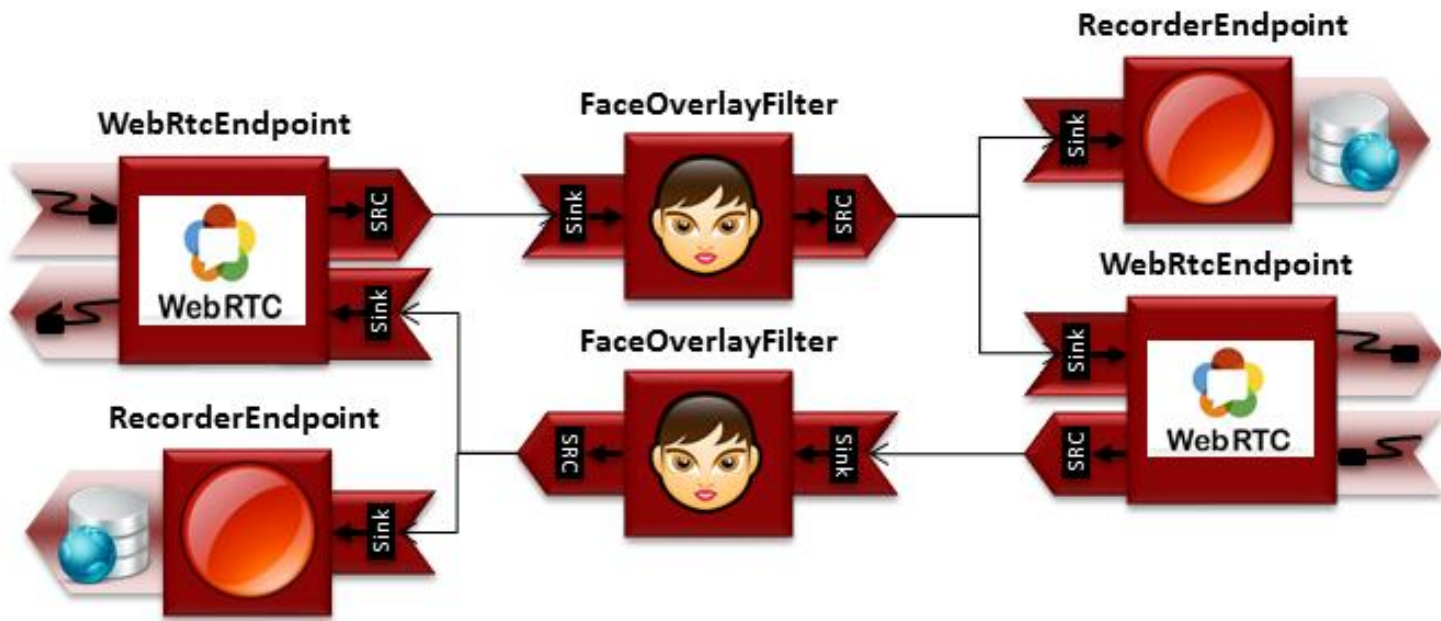
        WebRtcEndpoint webRtcEndpoint = new WebRtcEndpoint.Builder(pipeline)
            .build();
        FaceOverlayFilter faceOverlayFilter = new FaceOverlayFilter.Builder(
            pipeline).build();
        faceOverlayFilter.setOverlaidImage(
            "http://files.kurento.org/imgs/mario-wings.png", -0.35F,
            -1.2F, 1.6F, 1.6F);

        webRtcEndpoint.connect(faceOverlayFilter);
        faceOverlayFilter.connect(webRtcEndpoint);

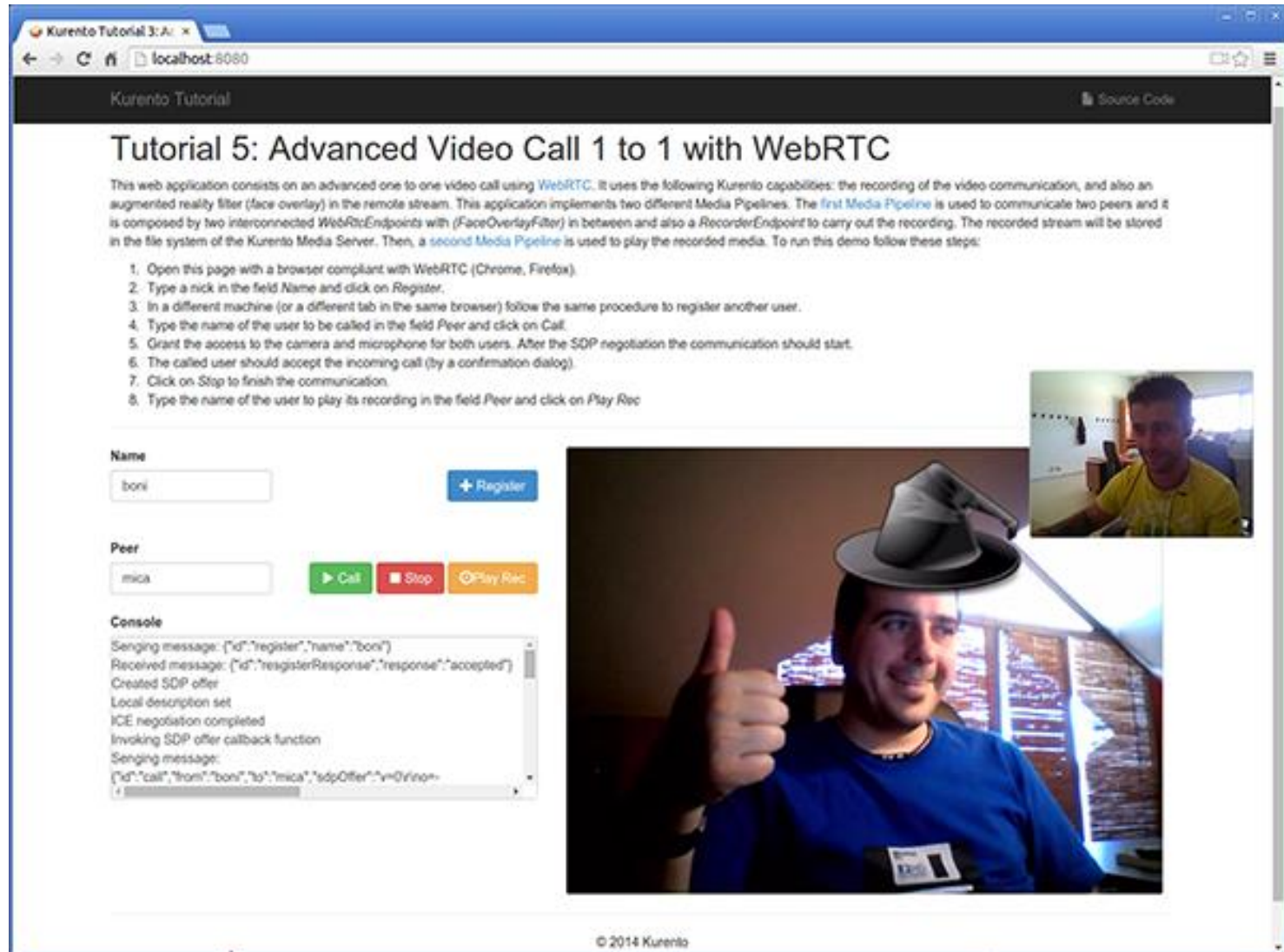
        // SDP negotiation (offer and answer)
        String sdpOffer = jsonMessage.get("sdpOffer").getAsString();
        String sdpAnswer = webRtcEndpoint.processOffer(sdpOffer);

        // Sending response back to client
        JsonObject response = new JsonObject();
        response.addProperty("id", "startResponse");
        response.addProperty("sdpAnswer", sdpAnswer);
        session.sendMessage(new TextMessage(response.toString()));
    } catch (Throwable t) {
        sendError(session, t.getMessage());
    }
}
```

Tutorial 3 - WebRTC one-to-one video call with recording and filtering



Tutorial 3 - WebRTC one-to-one video call with recording and filtering



The screenshot shows a web browser window titled "Kurento Tutorial 3: A" with the address bar at "localhost:8080". The page content includes:

Tutorial 5: Advanced Video Call 1 to 1 with WebRTC

This web application consists on an advanced one to one video call using WebRTC. It uses the following Kurento capabilities: the recording of the video communication, and also an augmented reality filter (face overlay) in the remote stream. This application implements two different Media Pipelines. The first Media Pipeline is used to communicate two peers and it is composed by two interconnected WebRtcEndpoints with (FaceOverlayFilter) in between and also a RecorderEndpoint to carry out the recording. The recorded stream will be stored in the file system of the Kurento Media Server. Then, a second Media Pipeline is used to play the recorded media. To run this demo follow these steps:

1. Open this page with a browser compliant with WebRTC (Chrome, Firefox).
2. Type a nick in the field Name and click on Register.
3. In a different machine (or a different tab in the same browser) follow the same procedure to register another user.
4. Type the name of the user to be called in the field Peer and click on Call.
5. Grant the access to the camera and microphone for both users. After the SDP negotiation the communication should start.
6. The called user should accept the incoming call (by a confirmation dialog).
7. Click on Stop to finish the communication.
8. Type the name of the user to play its recording in the field Peer and click on Play Rec

The interface features a "Name" field with the value "boni" and a "+ Register" button. Below it is a "Peer" field with the value "mica" and three buttons: "Call" (green), "Stop" (red), and "Play Rec" (orange). A "Console" window shows the following log messages:

```
Sending message: {"id": "register", "name": "boni"}
Received message: {"id": "registerResponse", "response": "accepted"}
Created SDP offer
Local description set
ICE negotiation completed
Invoking SDP offer callback function
Sending message: {"id": "call", "from": "boni", "to": "mica", "sdpOffer": "v=0/rtp="}
```

The main video area displays a large video feed of a man in a blue shirt giving a thumbs up, with a virtual hat and microphone overlay. A smaller inset video shows another man in a yellow shirt. At the bottom, it says "© 2014 Kurento".

Tutorial 3 - WebRTC one-to-one video call with recording and filtering

```
public CallMediaPipeline(KurentoClient kurento, String from, String to) {
    // Media pipeline
    MediaPipeline pipeline = kurento.createMediaPipeline();

    // Media Elements (WebRtcEndpoint, RecorderEndpoint, FaceOverlayFilter)
    webRtcCaller = new WebRtcEndpoint.Builder(pipeline).build();
    webRtcCallee = new WebRtcEndpoint.Builder(pipeline).build();

    recorderCaller = new RecorderEndpoint.Builder(pipeline, RECORDING_PATH
        + from + RECORDING_EXT).build();
    recorderCallee = new RecorderEndpoint.Builder(pipeline, RECORDING_PATH
        + to + RECORDING_EXT).build();

    FaceOverlayFilter faceOverlayFilterCaller = new FaceOverlayFilter.Builder(
        pipeline).build();
    faceOverlayFilterCaller.setOverlaidImage(
        "http://files.kurento.org/imgs/mario-wings.png", -0.35F, -1.2F,
        1.6F, 1.6F);

    FaceOverlayFilter faceOverlayFilterCallee = new FaceOverlayFilter.Builder(
        pipeline).build();
    faceOverlayFilterCallee.setOverlaidImage(
        "http://files.kurento.org/imgs/Hat.png", -0.2F, -1.35F, 1.5F,
        1.5F);

    // Connections
    webRtcCaller.connect(faceOverlayFilterCaller);
    faceOverlayFilterCaller.connect(webRtcCallee);
    faceOverlayFilterCaller.connect(recorderCaller);

    webRtcCallee.connect(faceOverlayFilterCallee);
    faceOverlayFilterCallee.connect(webRtcCaller);
    faceOverlayFilterCallee.connect(recorderCallee);
}
```

Kubernetes and the FIWARE-LAB

- FIWARE-LAB
 - Working instance of FI-WARE enabling free experimentation with technology
 - <http://lab.fiware.org>
- Creating a Kubernetes instance from an image
 - Use latest version of Kubernetes images.
- Creating a Kubernetes instance using recipes
 - Use Ubuntu 14.04 LTS clear image
 - Use latest version of Kubernetes recipes

Kubernetes and the FIWARE-LAB

Images

Name ▾	Status ▾
BoINC	active
CentOS-6.2-chef	active
CentOS-6.3-sdc	active
CentOS-6.3-x86_64	active
LPCI-internal	active
Ubuntu12.04-server-x86_64	active
cdvo-image-r2.3	active
cep-image-r2.3	active
datahandling-ppl	active
dbanonymizer-dba	active
kubernetes-image-4.0.0	active
kubernetes-image-r3.3	active

Launch Instances

1. Details 2. Access & Security 3. Post-Creation 4. Summary

Instance Name *

Flavor

Instance Count *

Description
Specify the details for launching an instance. The chart below shows the resources used by this project in relation to the project's quotas.

Flavor Details

Name	m1.small
VCPUs	1
Root Disk	10 GB
Ephemeral Disk	20 GB
Total Disk	30 GB
RAM	2048 MB

Project Quotas

Instance Count (1)	2 Available
VCPUs (1)	5 Available
Disk (10 GB)	990 GB Available
Memory (2048 MB)	22952 MB Available

* Mandatory fields.

Cancel **Next**

To learn more...

| Thank you!

<http://fiware.org>

Follow @FIWARE on Twitter

