Передача видео без потери качества в WebRTC

опыт VK Звонков

Николай Васильчук

Николай Васильчук

@vasilchuk

- 12 лет с видео
- 8 лет в VK
- 4 года со звонками
- не знаю React

О чём этот доклад?

- Как работают звонки и что такое WebRTC
- Демонстрация экрана в звонке
- Как получать, кодировать и декодировать кадры в браузере
- WebCodecs, libvpx, WASM
- Как отправлять данные через датаканал
- Как устроен контейнер WebM

О чём этот доклад?

- Как демонстрировать экран в высоком качестве

VK Звонки

6 млн

звонков в день

20 млн

пользователей в месяц

20%

групповых звонков
с демонстрацией экрана

2012	2018	05.2020	09.2020	02.2021
Start: web	mobile	8 group calls	128	

07.2021	08.2021	10.2021	11.2021	04.2022
4K	VK Звонки desktop 2048	SDK	∞ участников	VK Звонки mobile app

Как устроены звонки

WebRTC — Web Real Time Communication

W3C home > Mailing lists > Public > public-webrtc@w3.org > May 2011

Google release of WebRTC source code

This message: [Message body] [Respond] [More options]
Related messages: [Previous message]

From: Harald Alvestrand <hta@google.com>
Date: Wed, 1 Jun 2011 01:04:28 +0200
Message-ID: <BANLkTinXEJypAc2tVvAySzUGOtG9kLZNXA@mail.gmail.com>
To: public-webrtc@w3.org

Today, Google made available WebRTC, an open source software package for real-time voice and video on the web that we will be integrating in Chrome. This is our initial contribution to achieve the mission of making audio and video available in all browsers, through a uniform standard set of APIs. This initial release will provide the functionality we envision for WebRTC/RTCWEB, as detailed at https://sites.google.com/site/webrtc/. Working with the browser community and working groups like this, our goal is to expand the available APIs over the next few months for developers to create voice and video applications on the web.

The underlying components we're releasing are stable and the interfaces for this initial release are consistent with the discussions in this working group. We will continue to provide working implementations for consideration and feedback to collectively ensure stable standards are finalized. Google is committed to fully supporting these standards and we look forward to your input in the coming months.

Harald, speaking for Google.

https://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html

WebRTC — Web Real Time Communication

WebRTC	
Original author(s)	Justin Uberti Peter Thatcher
Initial release	2011; 11 years ago
Stable release	1.0[1] / May 4, 2018; 4 years ago
Repository	webrtc.googlesource.com
Written in	C++,[2] JavaScript
Standard(s)	w3.org/TR/webrtc/
License	BSD license
Website	webrtc.org

https://en.wikipedia.org/wiki/WebRTC

WebRTC — Web Real Time Communication

10 Years after Inception, WebRTC Becomes an Official Web Standard

APR 07, 2021 • 3 MIN READ

Web Real-Time Communications (WebRTC) recently became a World Wide Web Consortium (W3C) recommendation and Internet Engineering Task Force (IETF) standard. This is a major milestone on a long journey for WebRTC that started in 2011 with Google open-sourcing key communication technologies and Ericsson implementing the ConnectionPeer API. The new standard will continue to evolve as the WebRTC Working Group strives to integrate new use cases — live processing of audio and video feeds, Internet of Things use cases, and more.

https://www.infoq.com/news/2021/04/webrtc-official-web-standard/

WebRTC Peer-to-peer connections

- WD

Usage % of all users ?

Global	94.27% + 0.13% =	94.41%
unprefixed:	90.41% + 0.13% =	90.54%

Method of allowing two users to communicate directly, browser to browser using the RTCPeerConnection API.

Current aligned | Usage relative | Date relative | Filtered | All

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
						12.5				
						14.4				
						14.8				
	99	98	99	14.1		15.1				
	100	99	100	15.4	85	15.3				
11	101	100	101	15.5	86	15.4	all	101	12.12	16.0
		101	102	TP	87					
		102	103							
			104							

MediaStream

MediaStreamTrack

MediaStream

RTCPeerConnection

MediaStreamTrack

MediaStream

RTCPeerConnection

MediaStreamTrack

RTCSessionDescription

RTCIceCandidate

MediaStream

RTCPeerConnection

MediaStreamTrack

RTCSessionDescription

RTCIceCandidate

RTCDataChannel

MediaStream

RTCPeerConnection

MediaStreamTrack

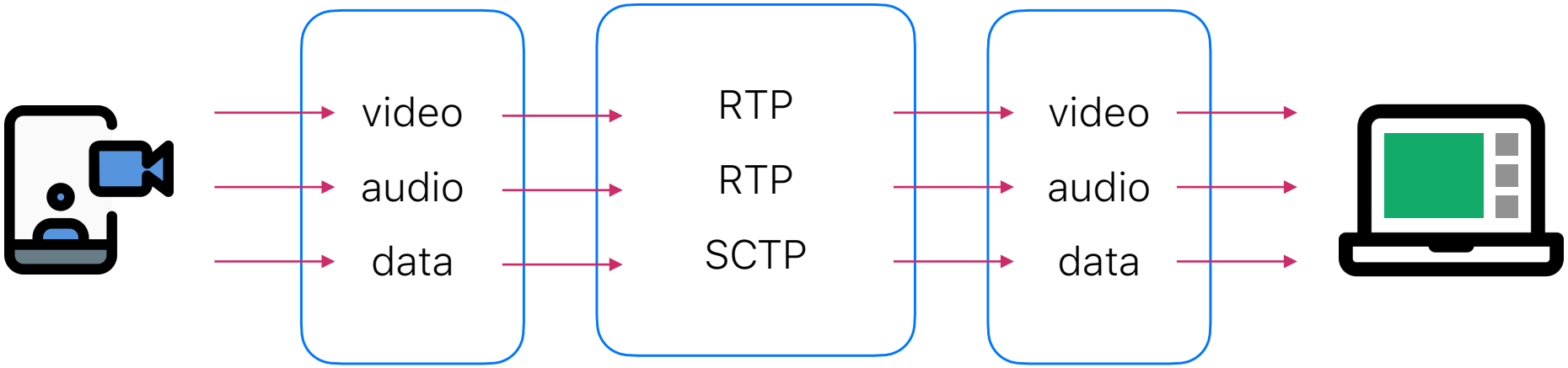
RTCSessionDescription

RTCRtpReceiver

RTCIceCandidate

RTCRtpSender

RTCDataChannel



Real-time Transport Protocol

Stream Control Transmission Protocol

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

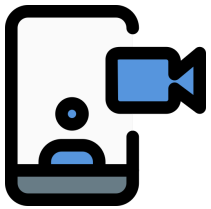
```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

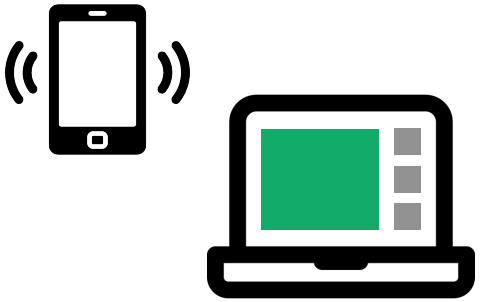
```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

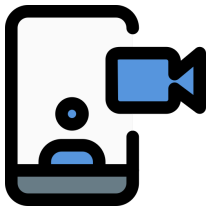
```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

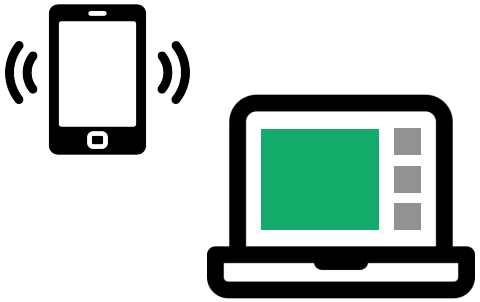
offer

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```







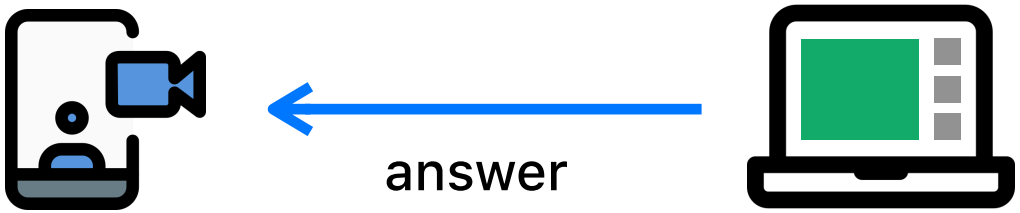


```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```


answer

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```

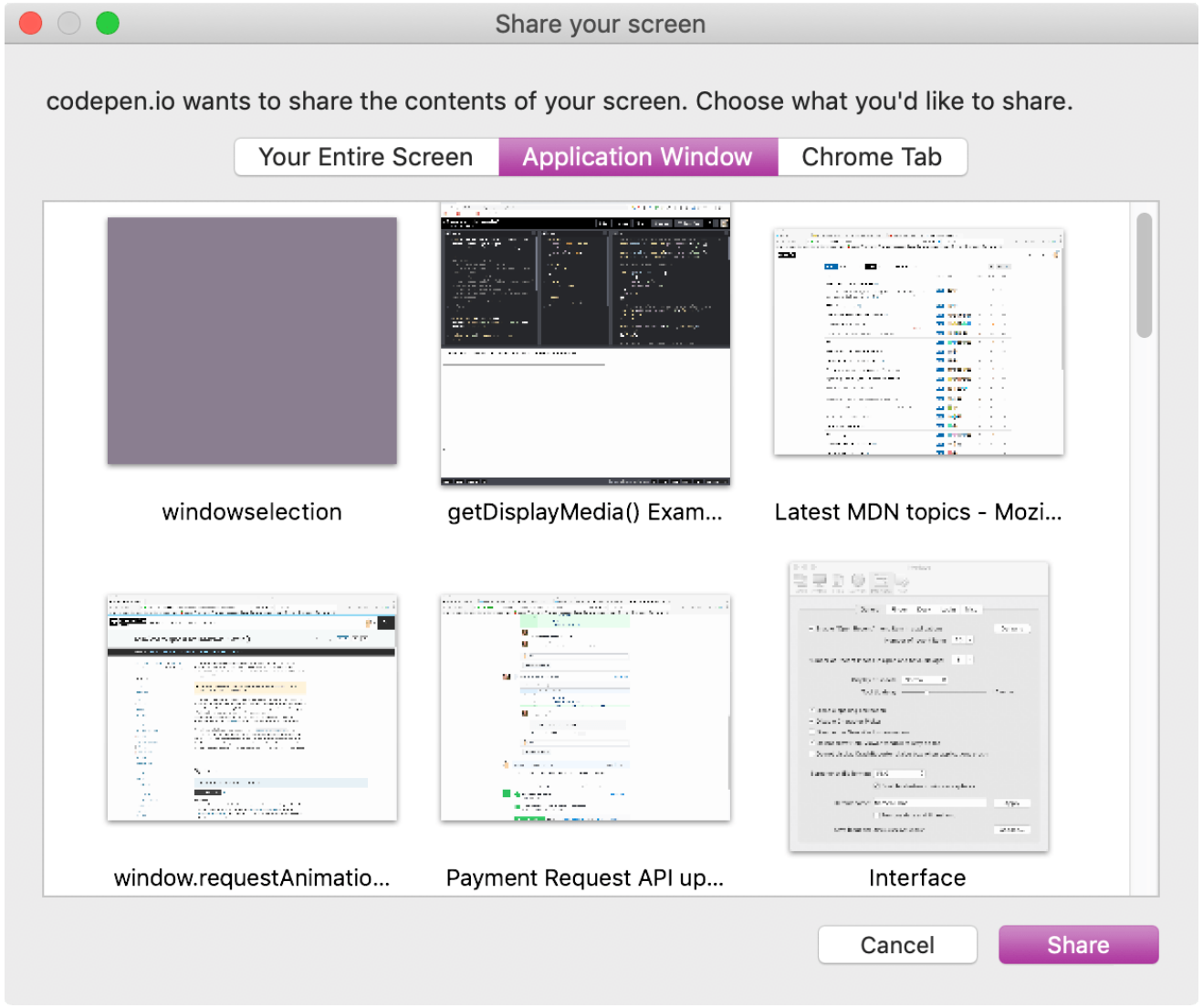
Захват экрана

```
navigator.mediaDevices.getUserMedia()
```

https://developer.mozilla.org/ru/docs/Web/API/MediaDevices/getUserMedia

```
navigator.mediaDevices.getDisplayMedia()
```

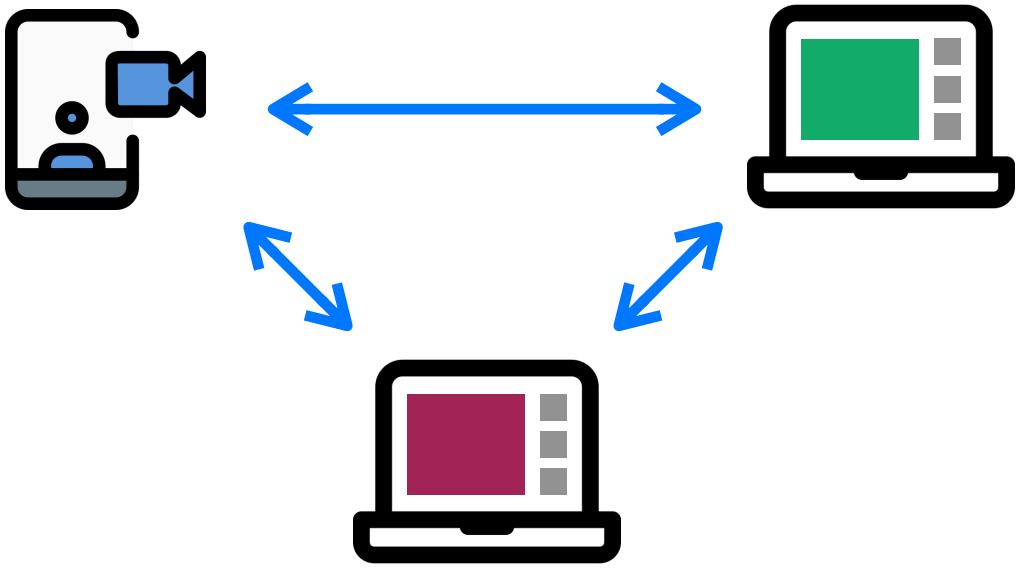
https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getDisplayMedia

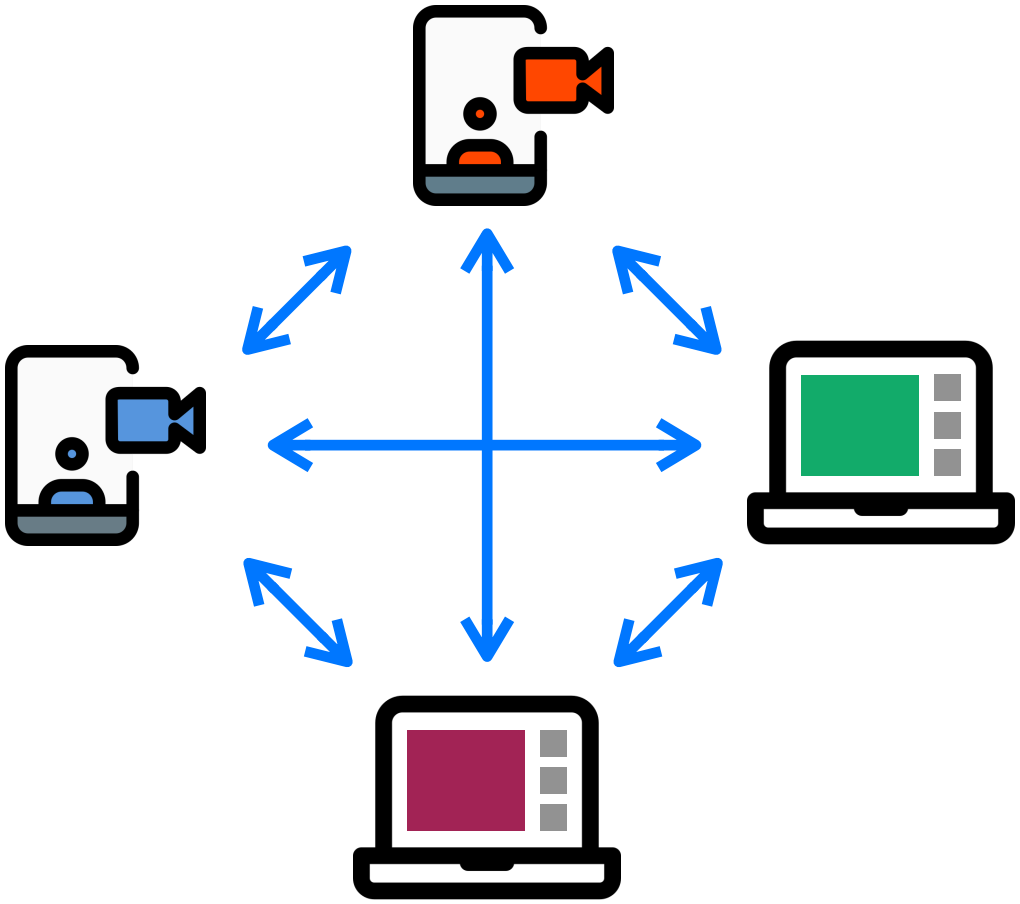

Share your screen
codepen.io wants to share the contents of your screen. Choose what you'd like to share.
Your Entire Screen
Application Window
Chrome Tab
windowselection
getDisplayMedia() Exam...
Latest MDN topics - Mozi...
window.requestAnimatio...
Payment Request API up...
Interface
Cancel
Share

```
const stream = await navigator.mediaDevices.getDisplayMedia({
    video: true,
});

const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));

// or
const [sender] = pc.getSenders();
const [track] = stream.getVideoTracks();
await sender.replaceTrack(newTrack);
```

```
const stream = await navigator.mediaDevices.getDisplayMedia({
    video: true,
});

const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));

// or
const [sender] = pc.getSenders();
const [track] = stream.getVideoTracks();
await sender.replaceTrack(newTrack);
```

```
const stream = await navigator.mediaDevices.getDisplayMedia({
    video: true,
});

const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));

// or
const [sender] = pc.getSenders();
const [track] = stream.getVideoTracks();
await sender.replaceTrack(newTrack);
```





2012	2018	05.2020	09.2020	02.2021
Start: web	mobile	**8** group calls	**128**	

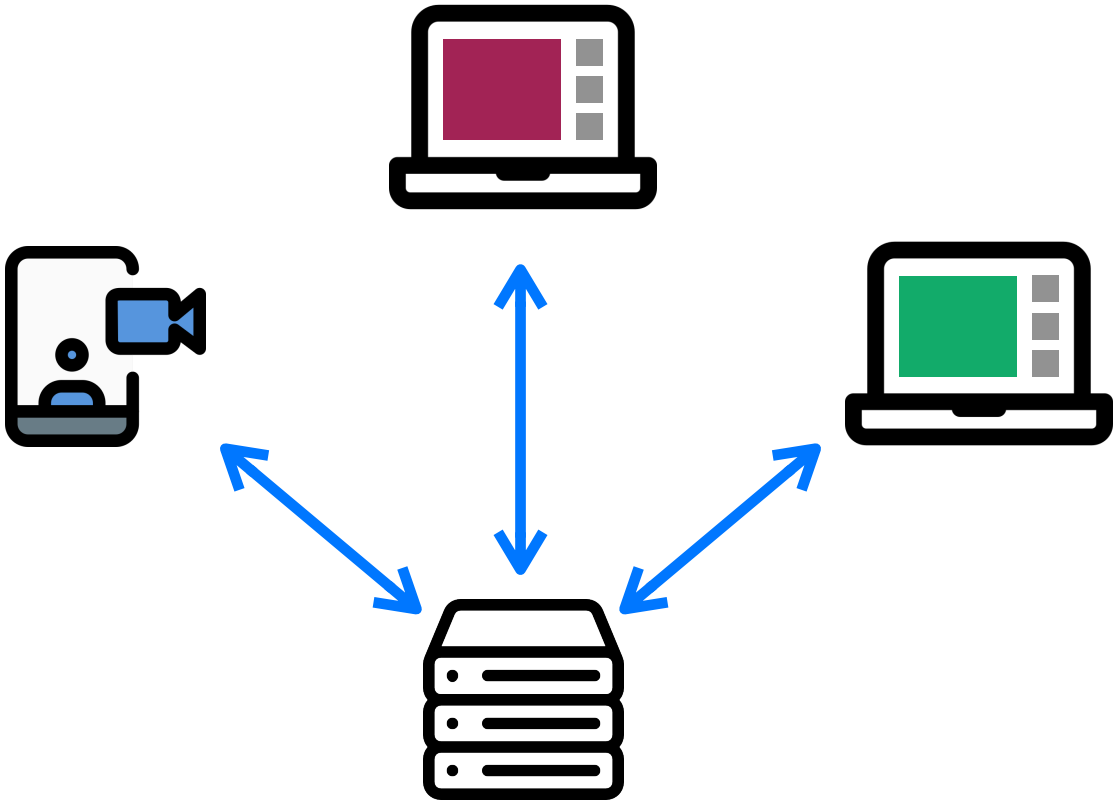
07.2021	08.2021	10.2021	11.2021	04.2022
4K	**VK Звонки** desktop **2048**	**SDK**	∞ участников	**VK Звонки** mobile app

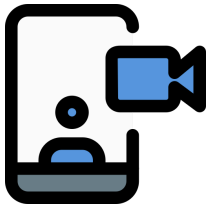
05.2020

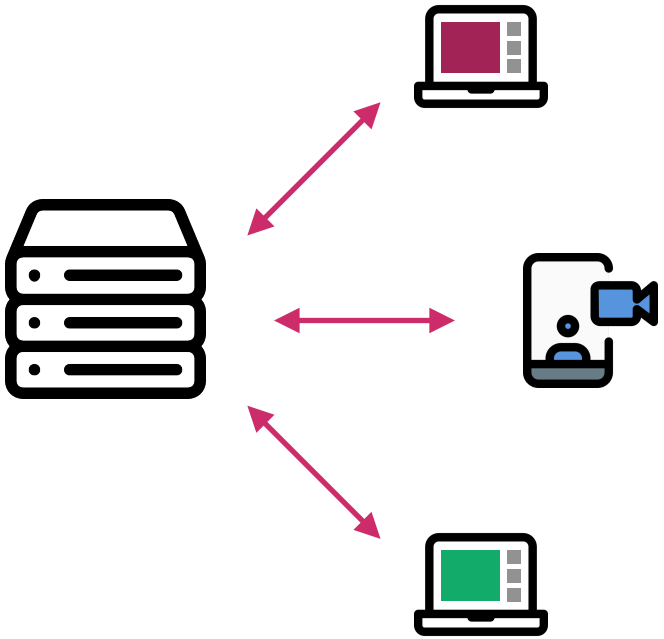
8 group calls

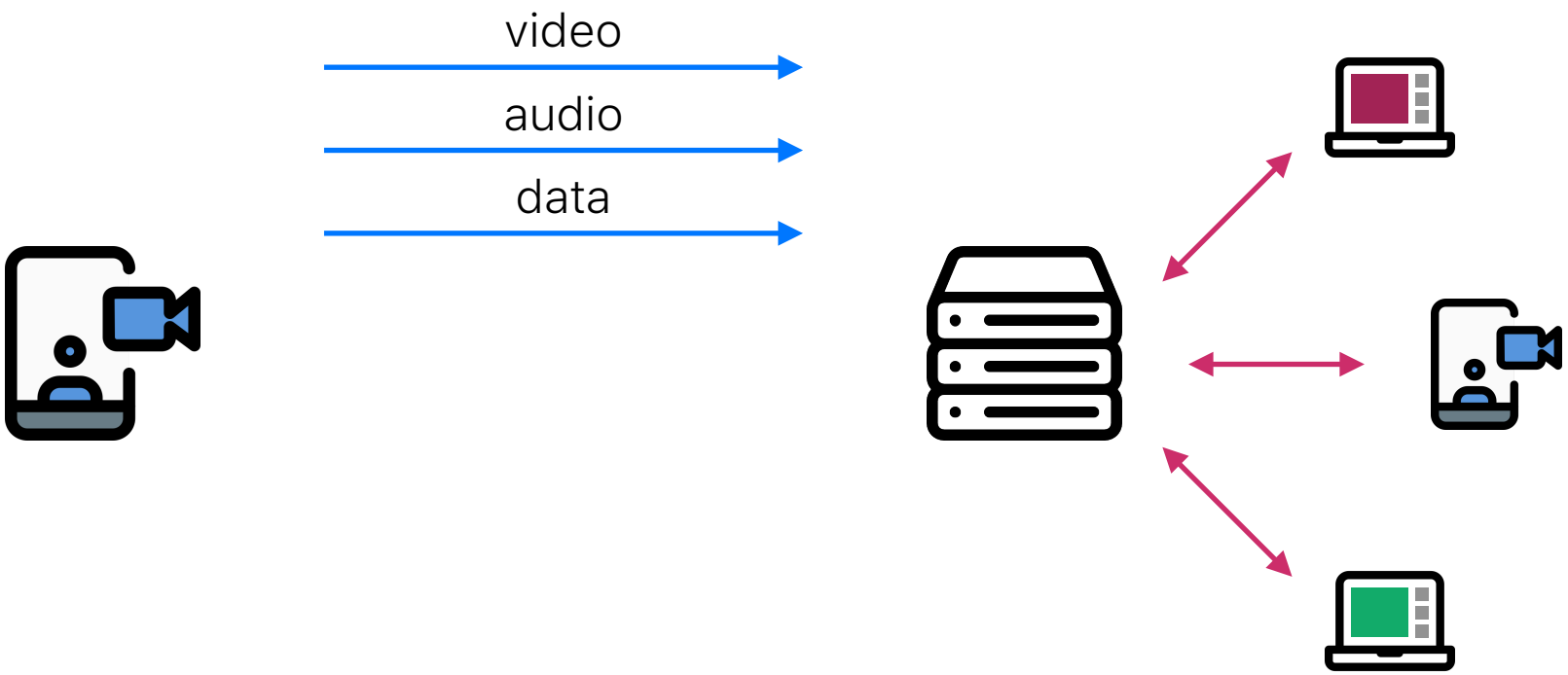
09.2020

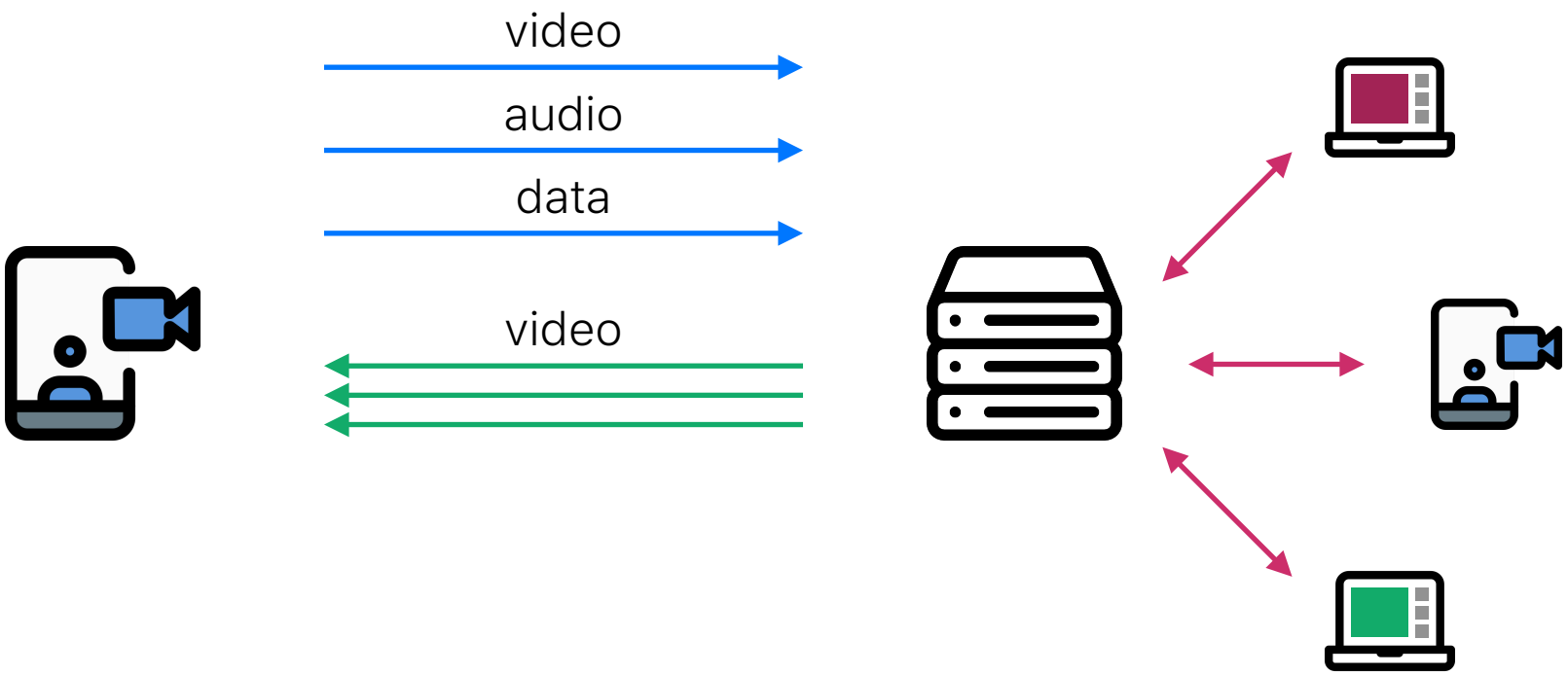
128

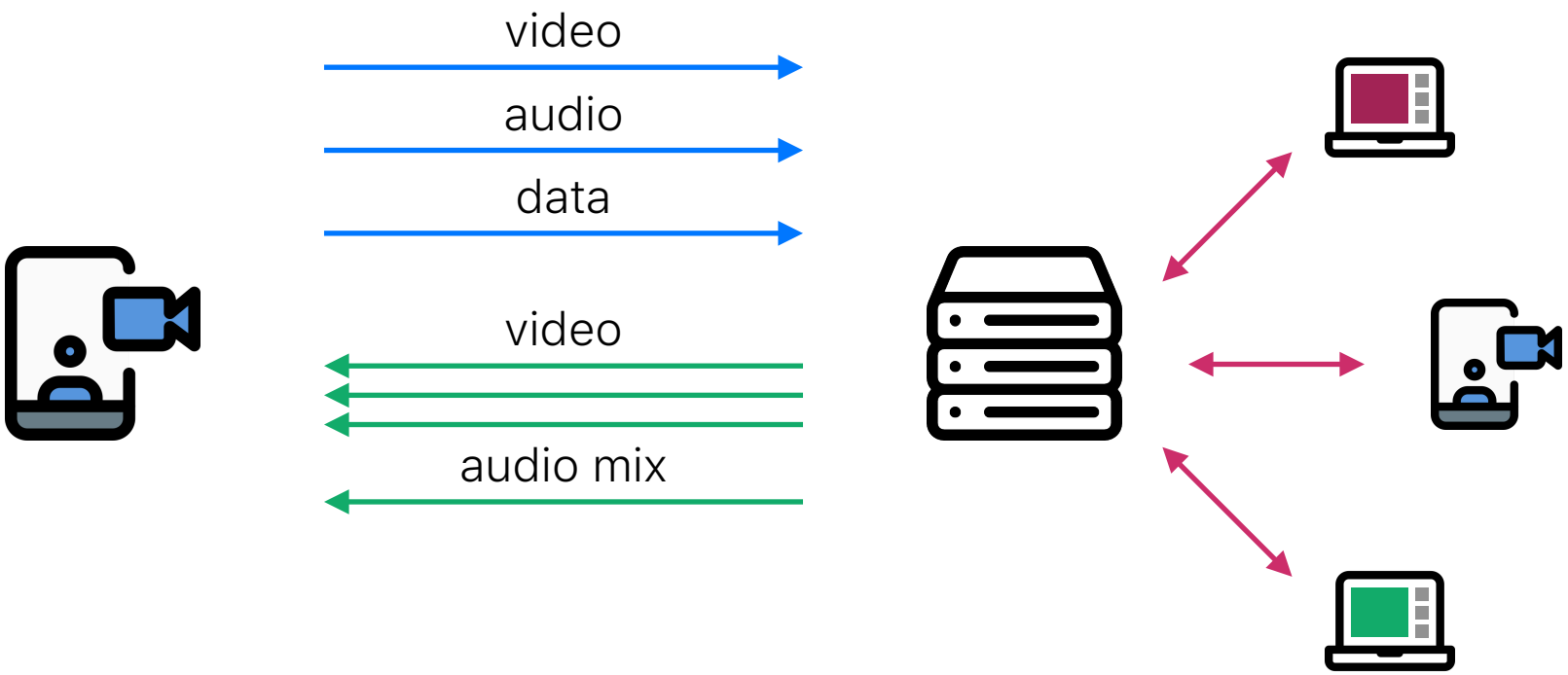


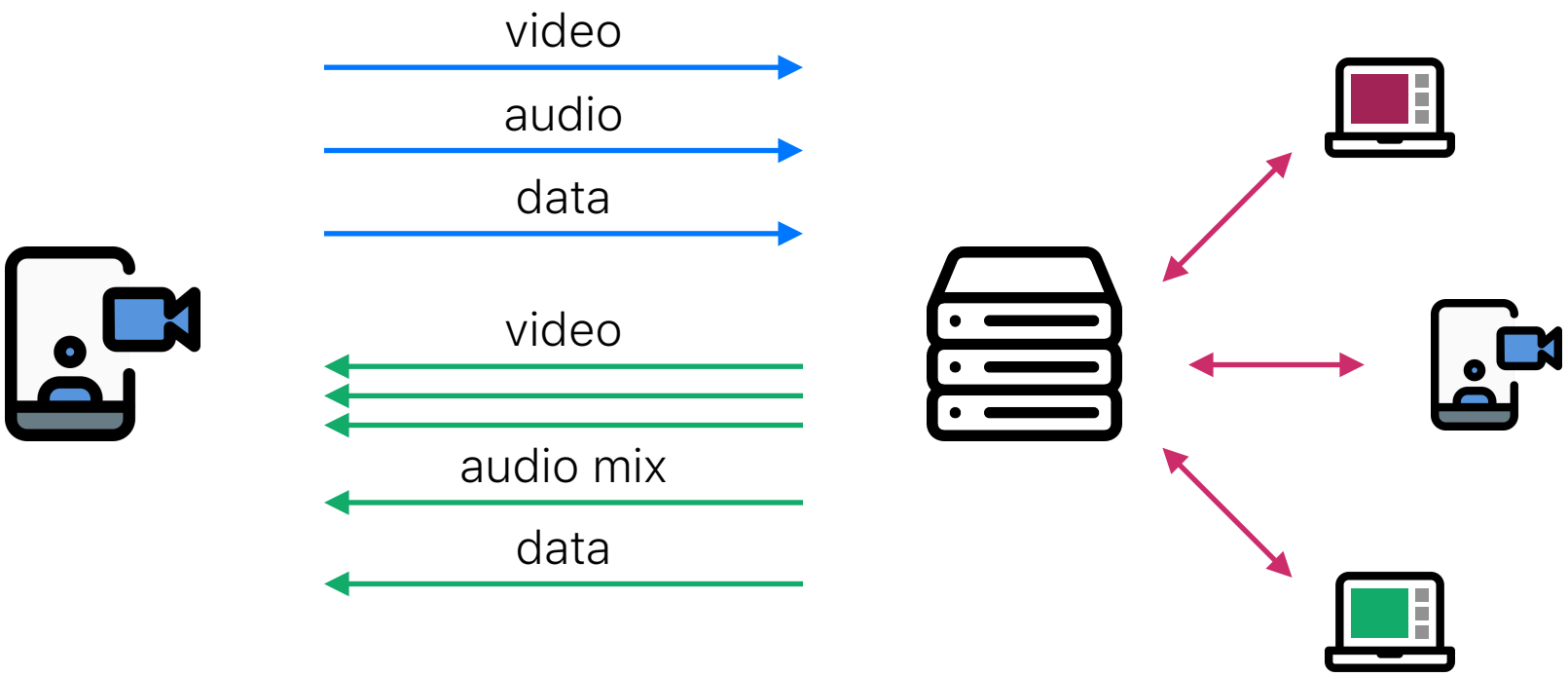





video
audio
data


video
audio
data
video


video
audio
data
video
audio mix


video
audio
data
video
audio mix
data

Что мы узнали?

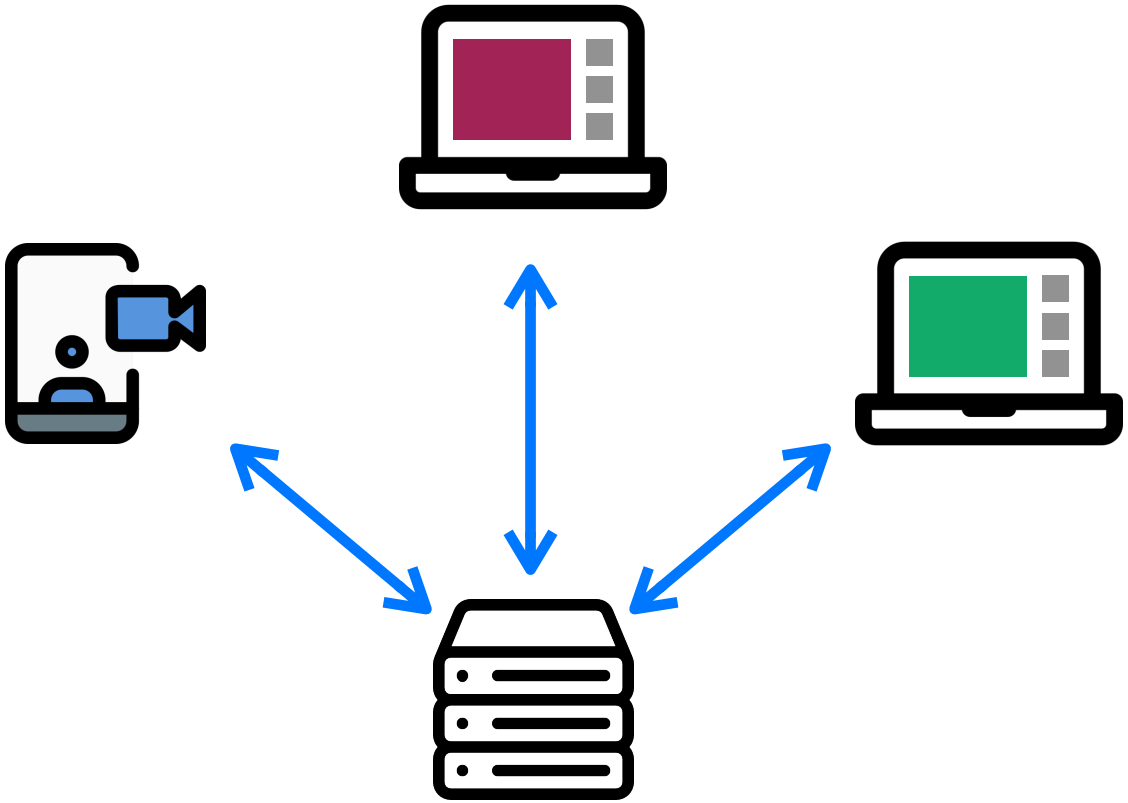
- WebRTC – это просто
- WebRTC – это кроссбраузерно
- WebRTC – это про P2P
- WebRTC – это про медиа

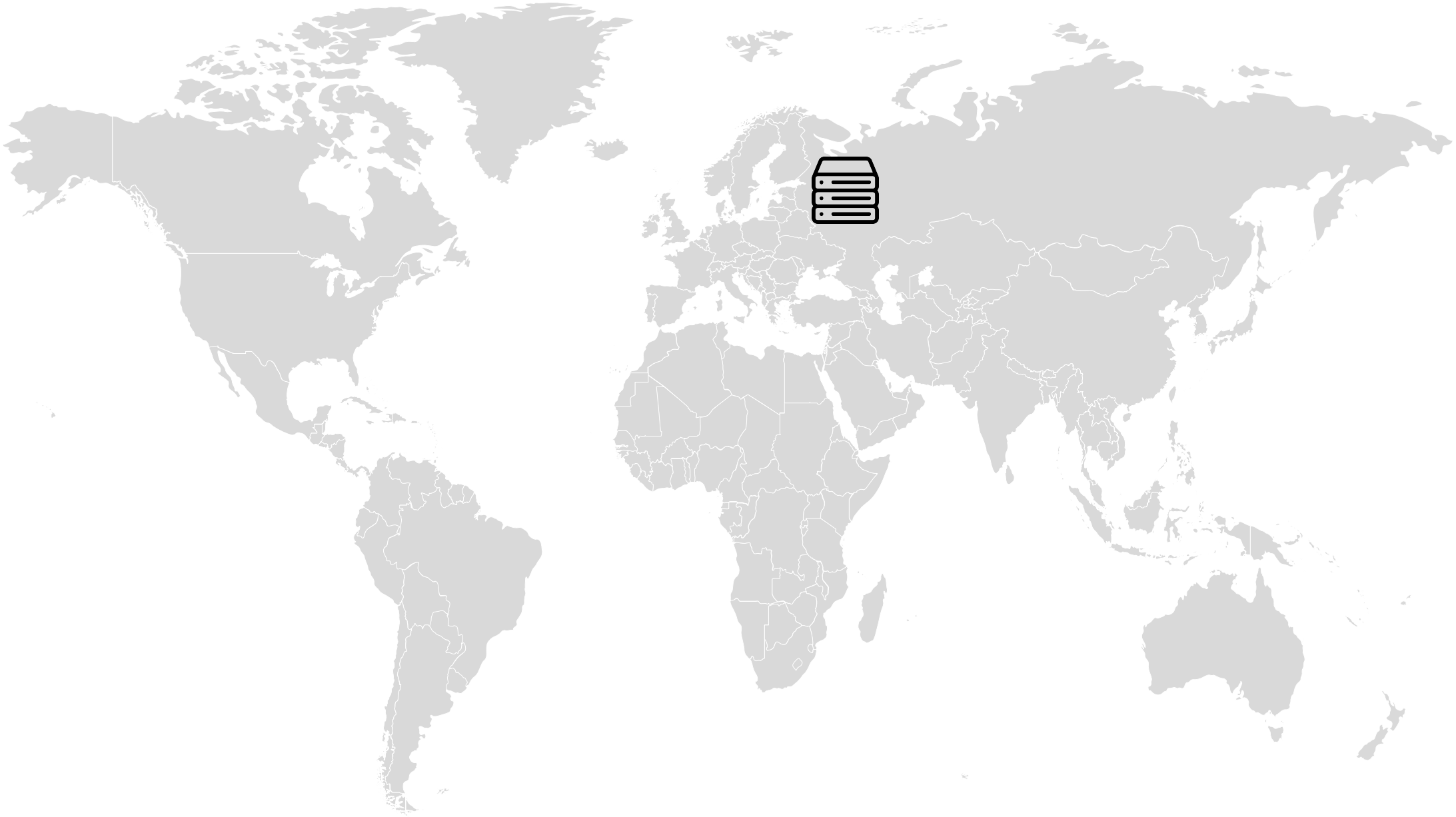
Что мы узнали?

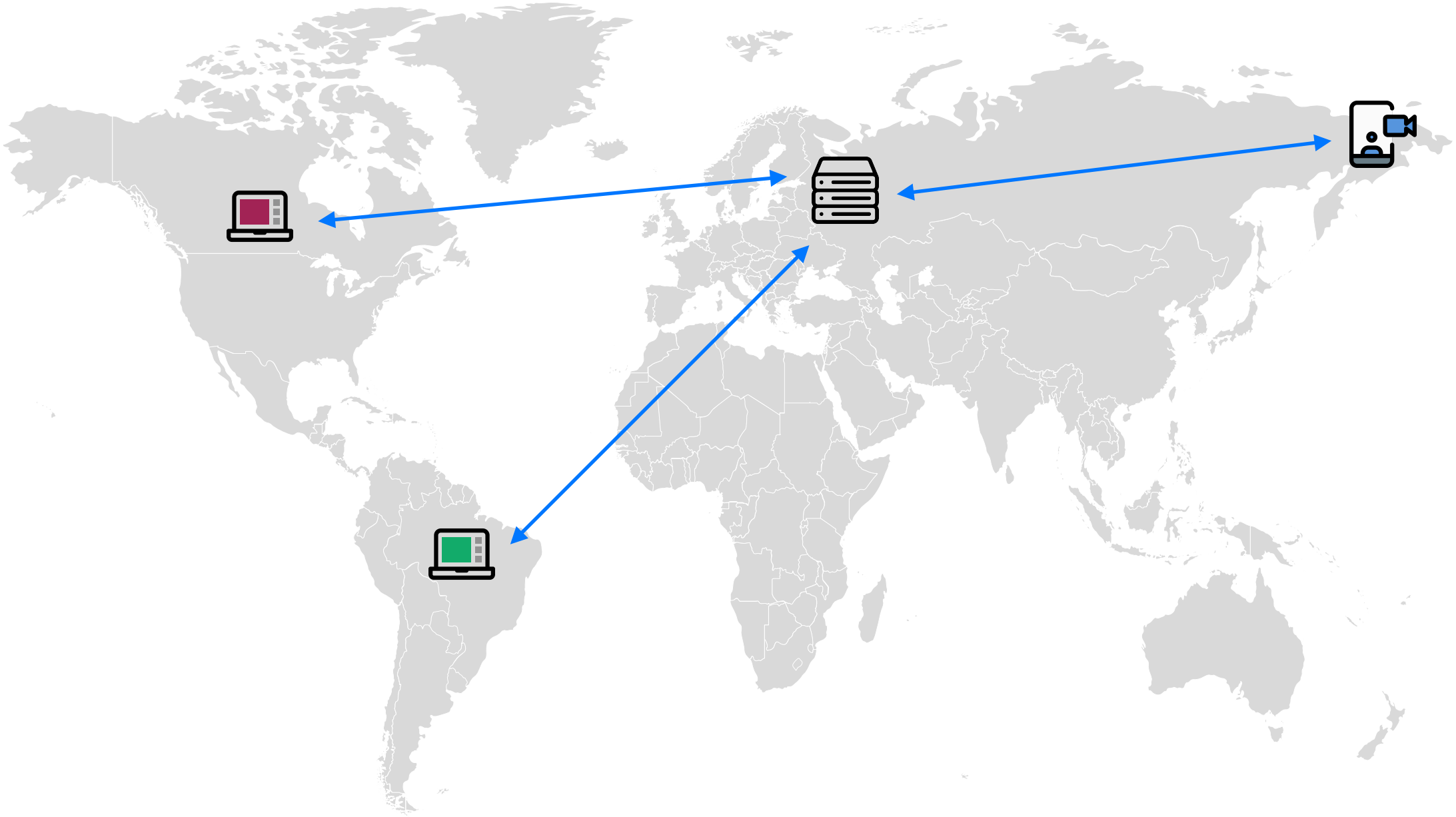
- WebRTC – это просто
- WebRTC – это кроссбраузерно
- WebRTC – это про P2P
- WebRTC – это про медиа
- Если участников много – нужен сервер

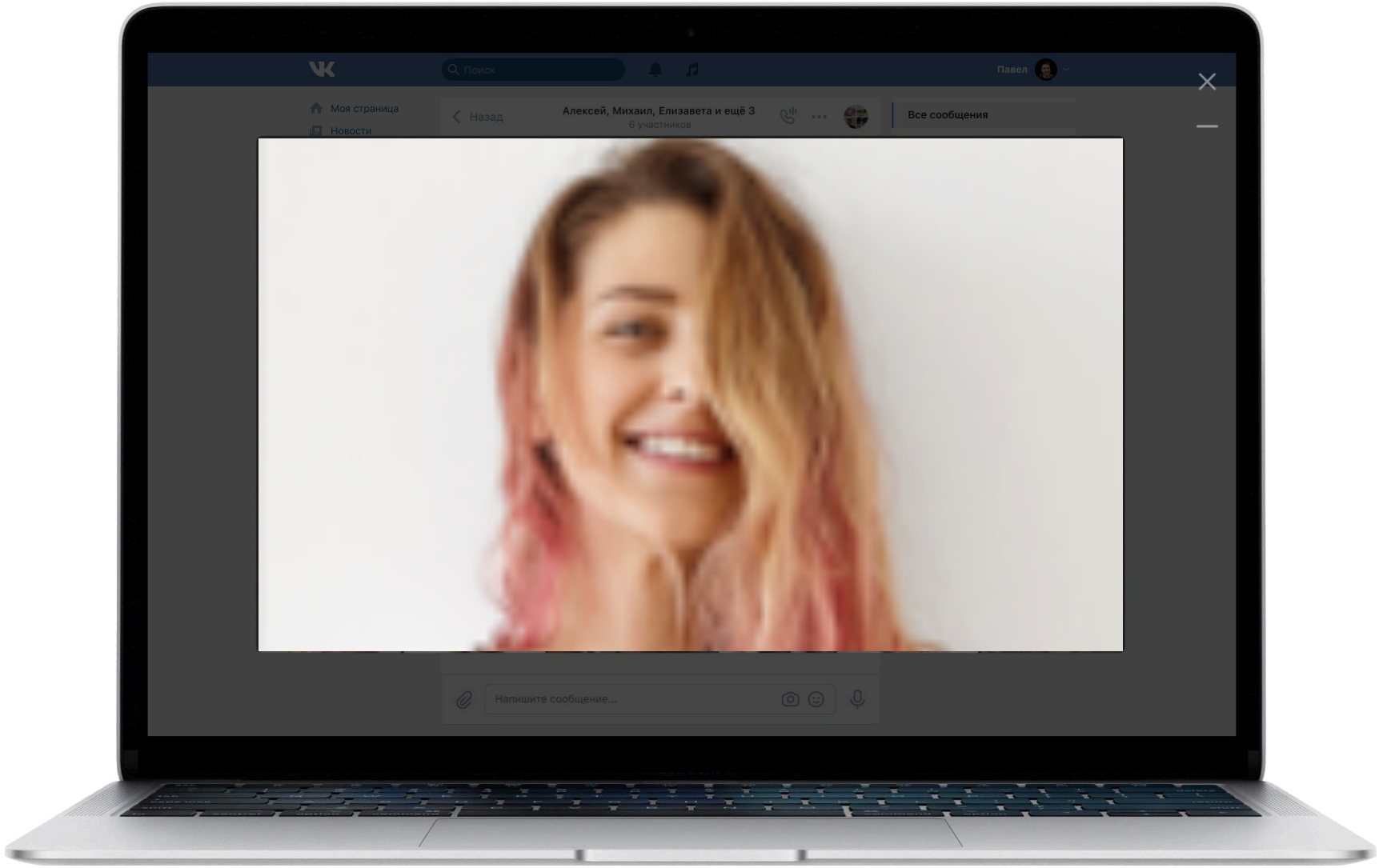
Плохая сеть

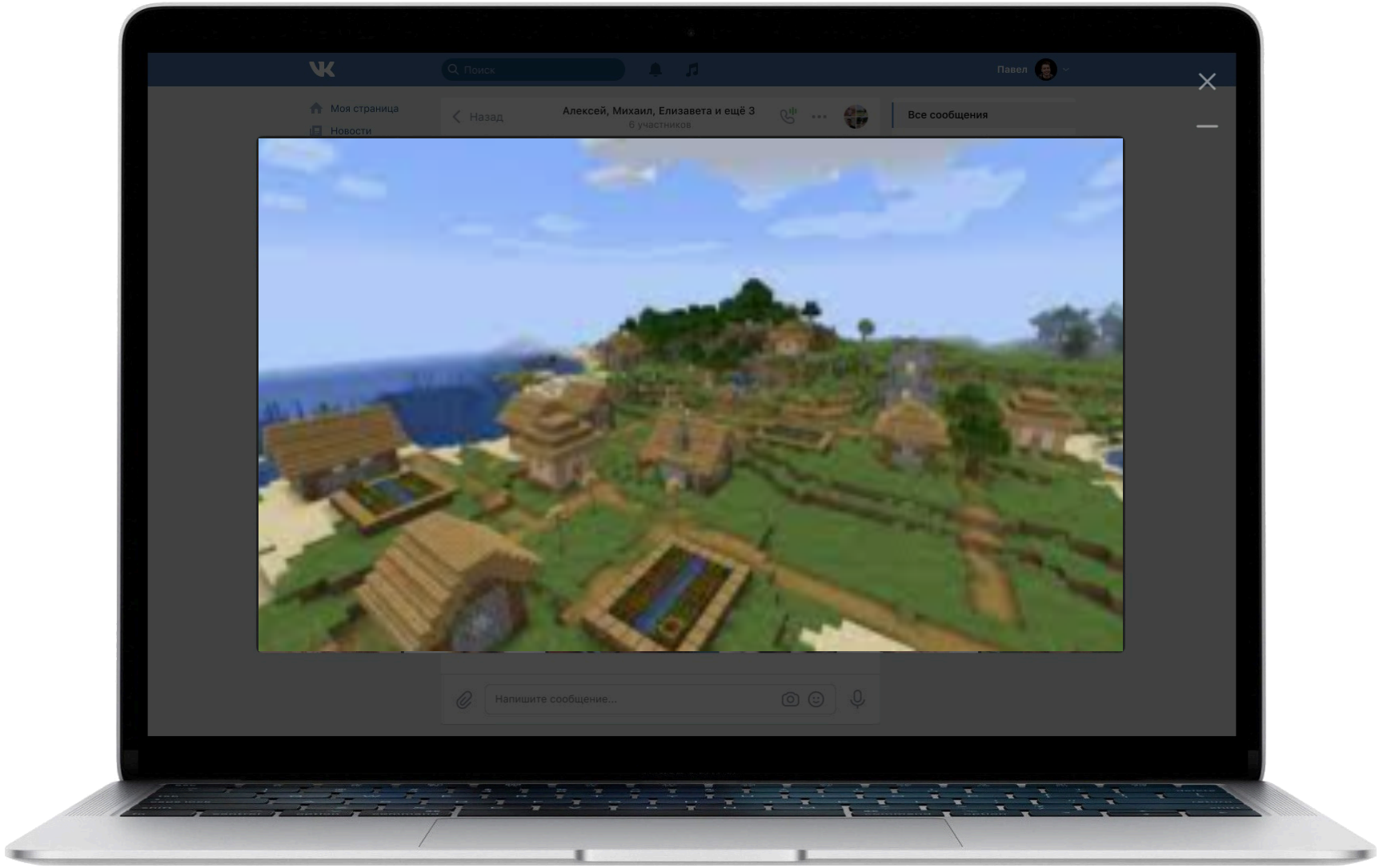
вконтакте
Павел
Встреча в ВК
1:15:37 · 128 участников
Говорят Павел, Арина
financial highlights
Вы
Наташа
Арина
Ян
12:48
1:15:37

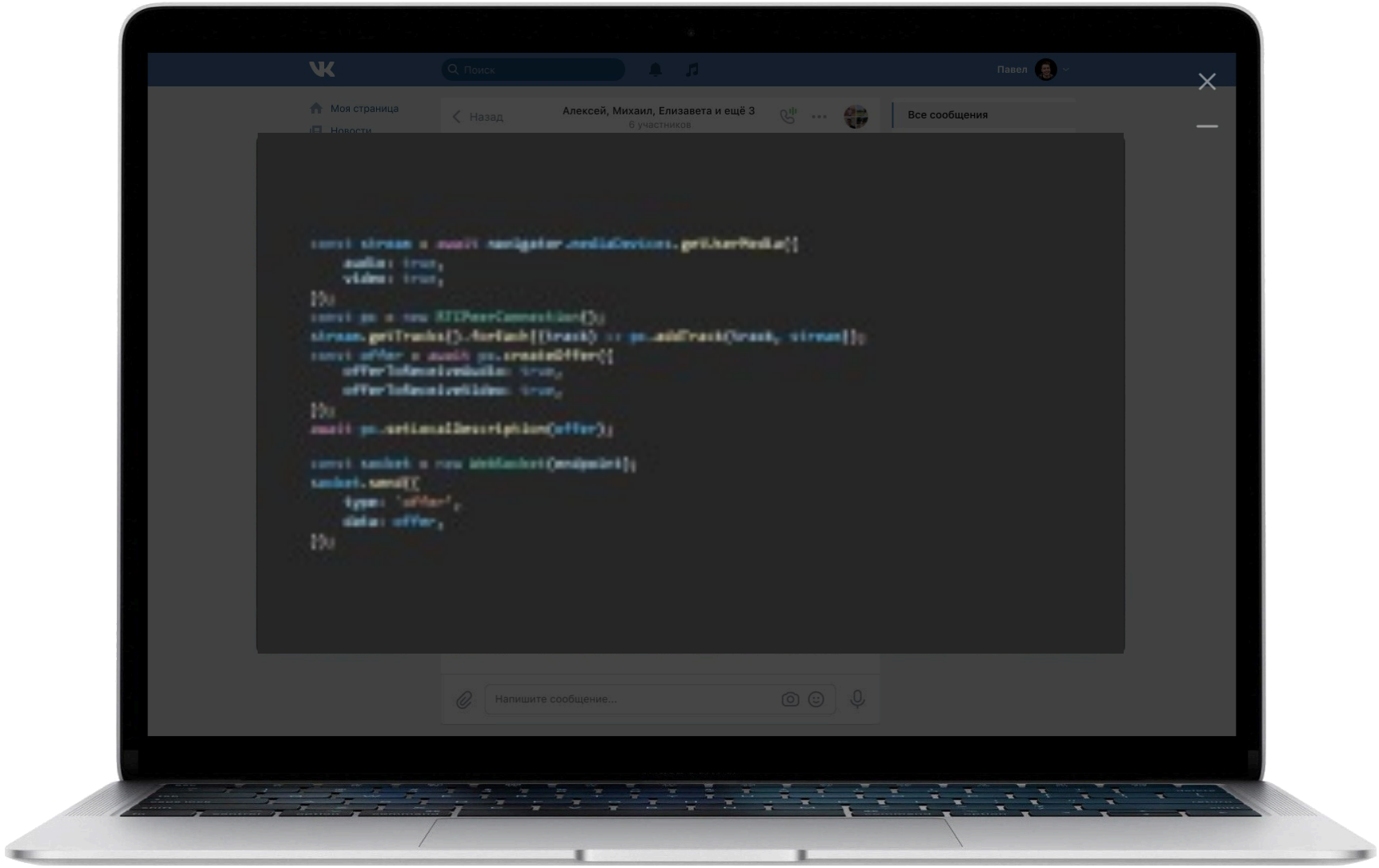




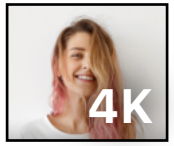


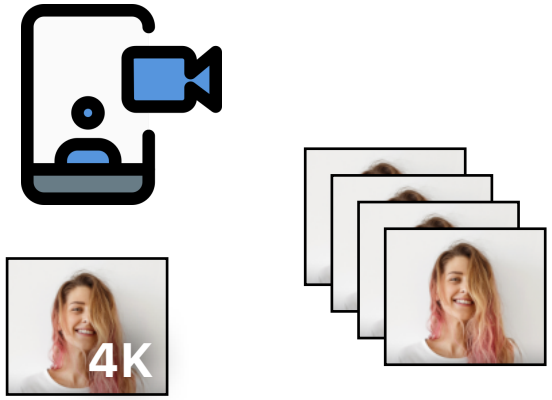

Поиск
Павел
Моя страница
Новости
Назад
Алексей, Михаил, Елизавета и ещё 3
6 участников
Все сообщения
Напишите сообщение...

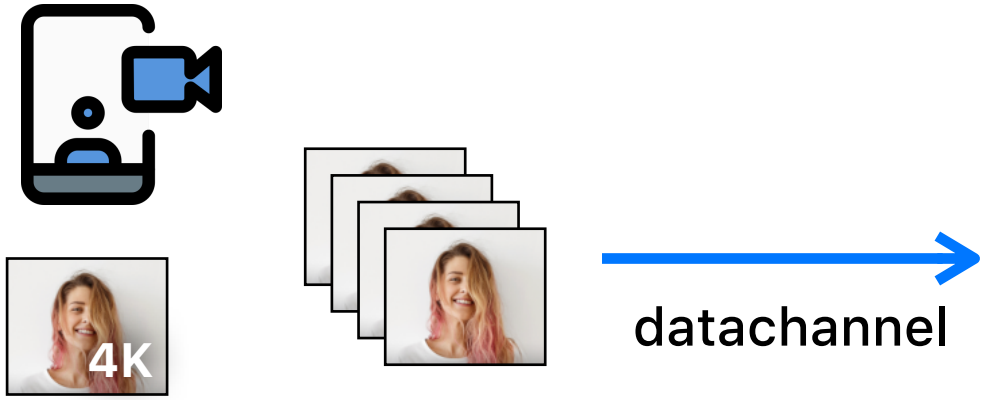

Поиск
Павел
Моя страница
Новости
Назад
Алексей, Михаил, Елизавета и ещё 3
6 участников
Все сообщения
Напишите сообщение...

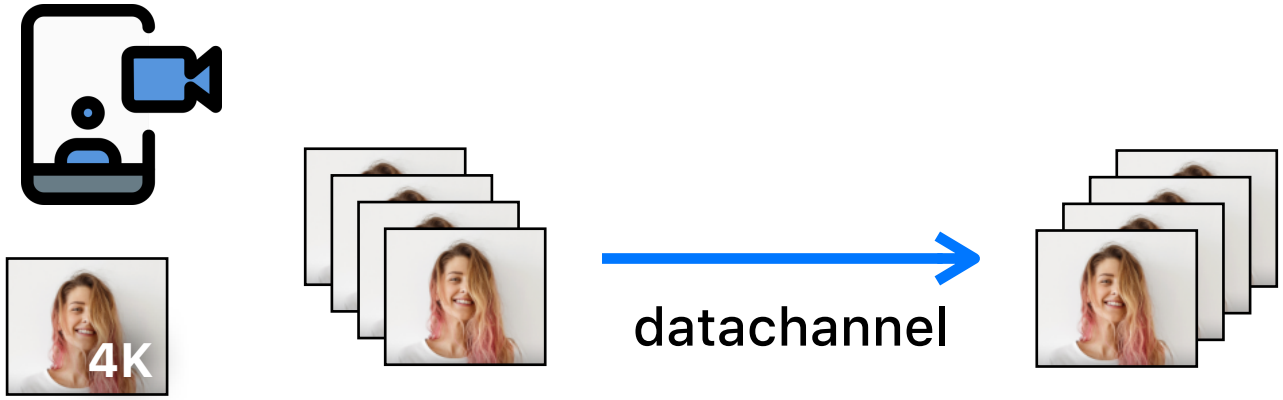

Поиск
Павел
Моя страница
Новости
Назад
Алексей, Михаил, Елизавета и ещё 3
6 участников
Все сообщения
const stream = await navigator.mediaDevices.getUserMedia({
audio: true,
video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
offerToReceiveAudio: true,
offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);
const socket = new WebSocket(endpoint);
socket.send({
type: 'offer',
data: offer,
});
Напишите сообщение...

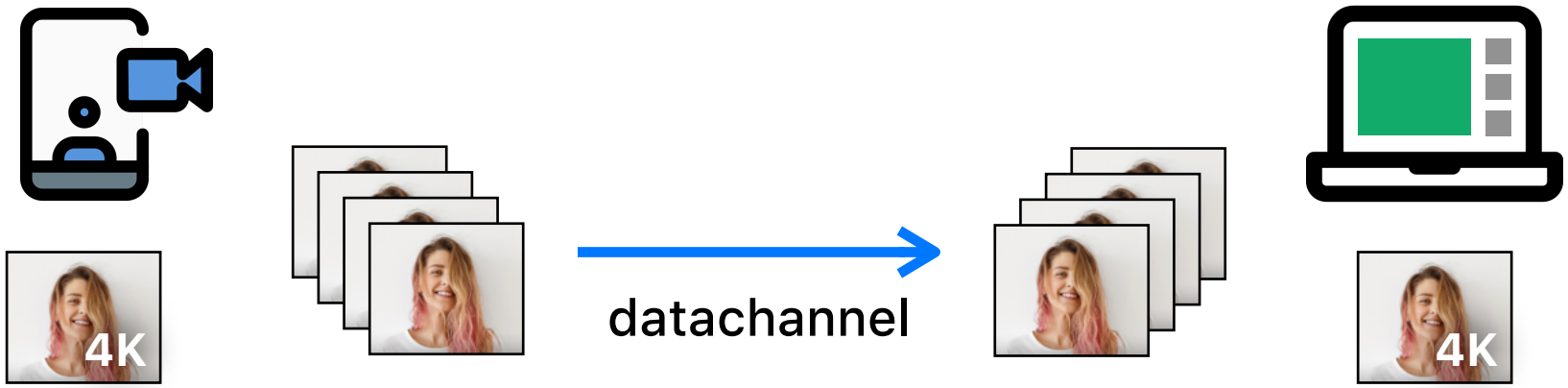
```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(wssUrl);
socket.send({
    type: "offer",
    data: offer,
});
```


4K


4K


4K
datachannel


4K
datachannel


4K
datachannel
4K

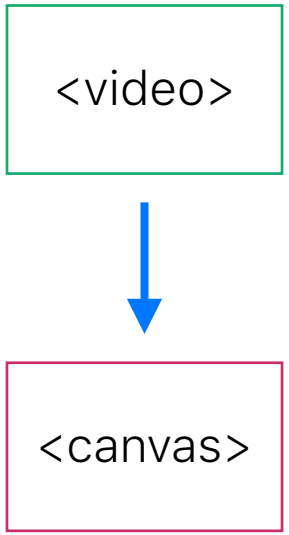
<video>

```
const video = document.querySelector('video');
video.srcObject = new MediaStream([track]);
```

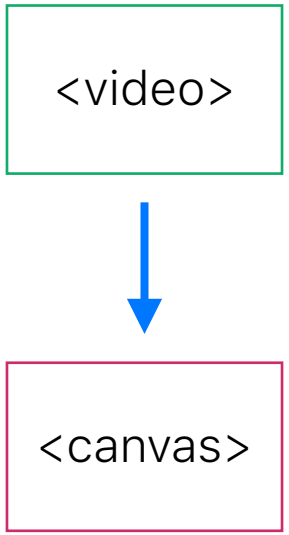
<video>

<canvas>

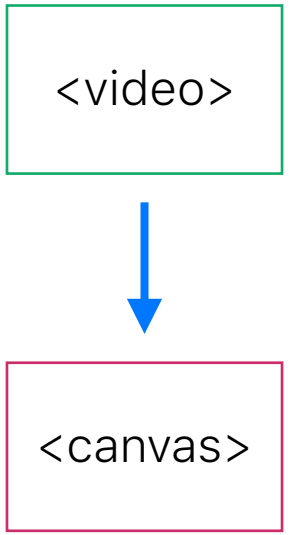
```
const video = document.querySelector('video');
video.srcObject = new MediaStream([track]);

const canvas = document.querySelector('canvas');
const canvasCtx = canvas.getContext('2d');
```



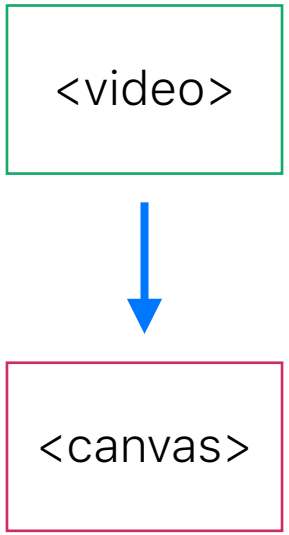

```
const video = document.querySelector('video');
video.srcObject = new MediaStream([track]);

const canvas = document.querySelector('canvas');
const canvasCtx = canvas.getContext('2d');

canvasCtx.drawImage(video, 0, 0, width, height);
```



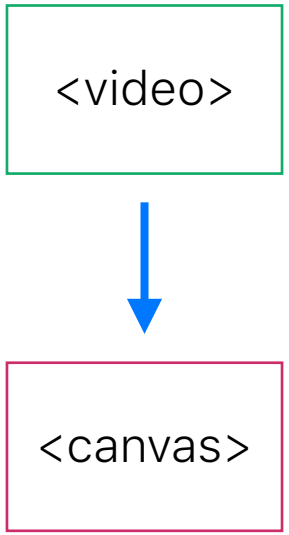
```
const video = document.querySelector('video');
video.srcObject = new MediaStream([track]);

const canvas = document.querySelector('canvas');
const canvasCtx = canvas.getContext('2d');

canvasCtx.drawImage(video, 0, 0, width, height);

const imageData = canvasCtx.getImageData(0, 0, width, height);
```



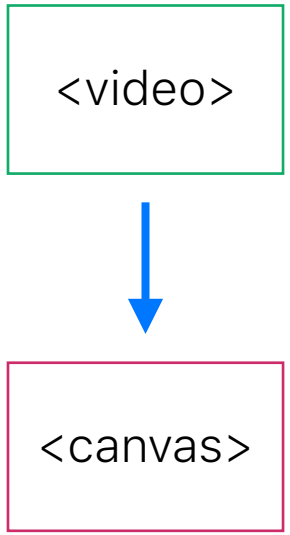
```
const video = document.querySelector('video');
video.srcObject = new MediaStream([track]);

const canvas = document.querySelector('canvas');
const canvasCtx = canvas.getContext('2d');

canvasCtx.drawImage(video, 0, 0, width, height);

const imageData = canvasCtx.getImageData(0, 0, width, height);
```

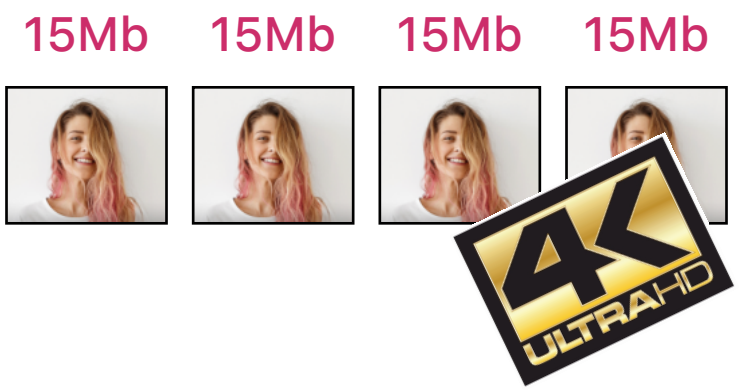


```
const video = document.querySelector('video');
video.srcObject = new MediaStream([track]);

const canvas = document.querySelector('canvas');
const canvasCtx = canvas.getContext('2d');

canvasCtx.drawImage(video, 0, 0, width, height);

const imageData = canvasCtx.getImageData(0, 0, width, height);
```

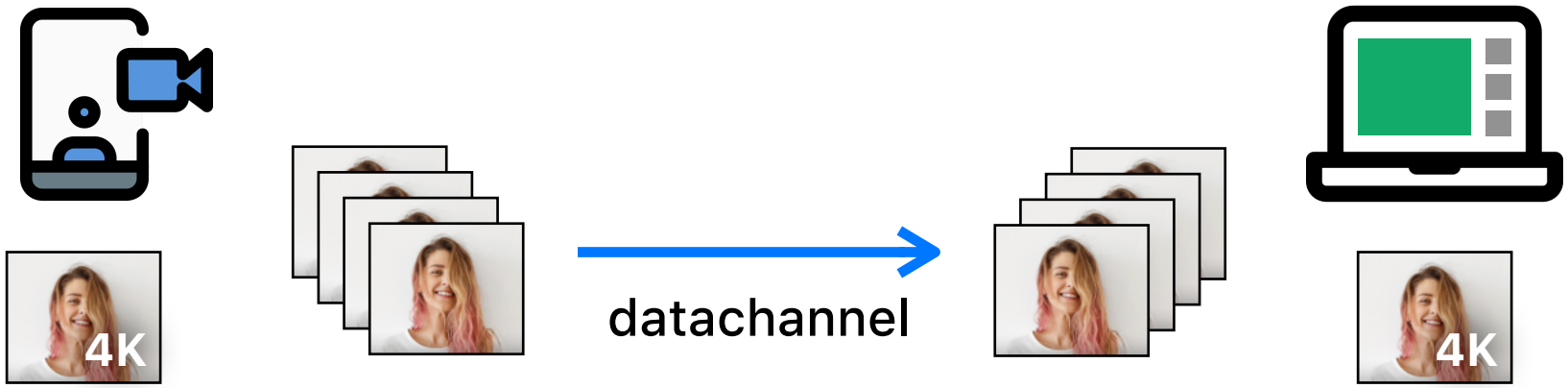


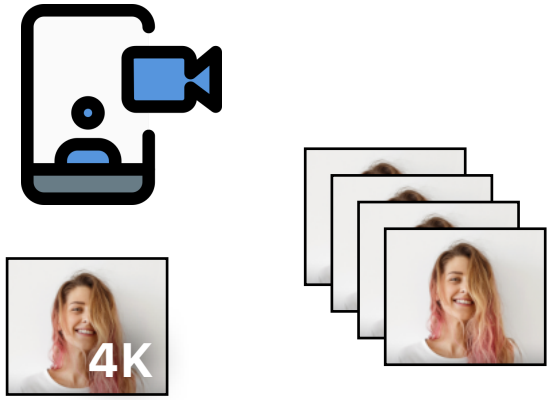
```
const video = document.querySelector('video');
video.srcObject = new MediaStream([track]);

const canvas = document.querySelector('canvas');
const canvasCtx = canvas.getContext('2d');

canvasCtx.drawImage(video, 0, 0, width, height);

const imageData = canvasCtx.getImageData(0, 0, width, height);
```

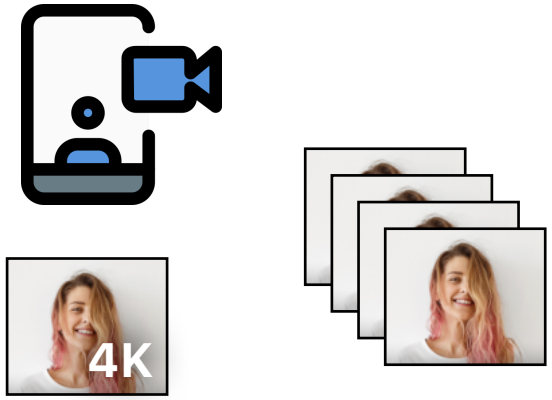




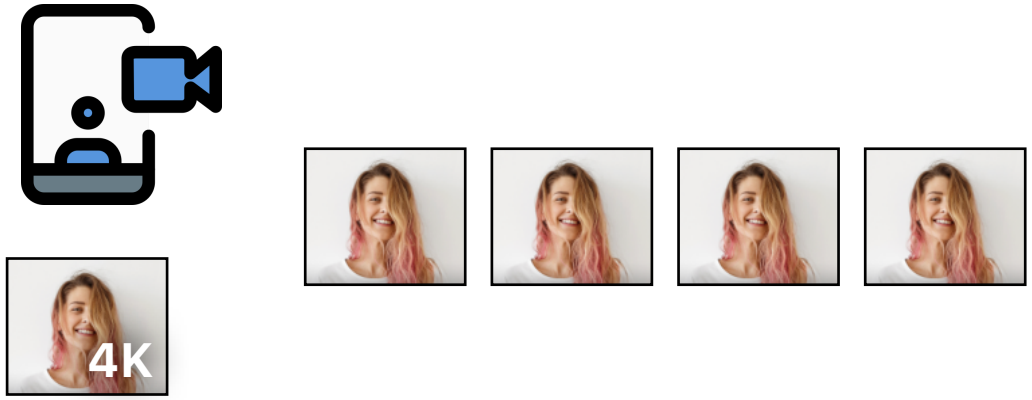

```
const video = document.querySelector('video');
video.srcObject = new MediaStream([track]);

const canvas = document.querySelector('canvas');
const canvasCtx = canvas.getContext('2d');

canvasCtx.drawImage(video, 0, 0, width, height);

const imageData = canvasCtx.getImageData(0, 0, width, height);
```


4K
datachannel
4K

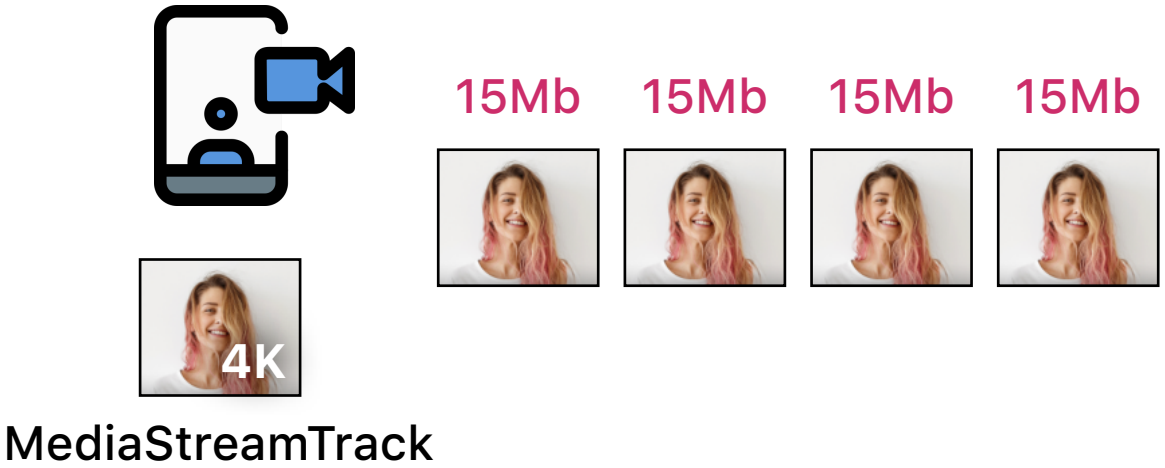

4K

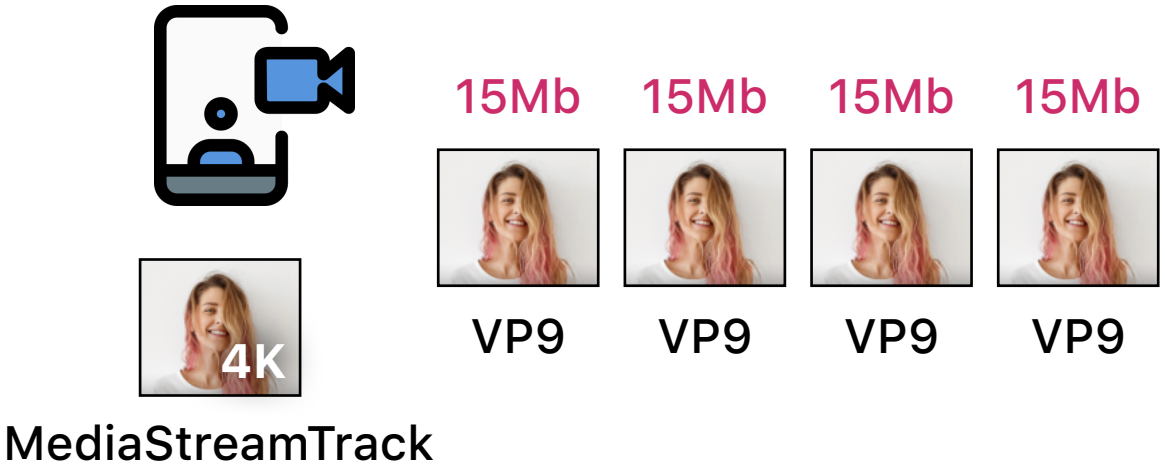


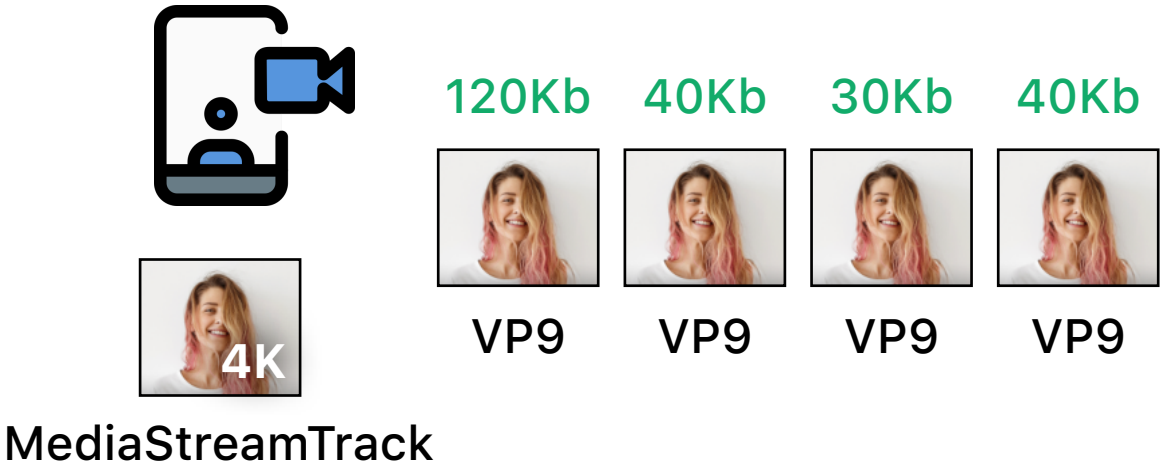
MediaStreamTrack

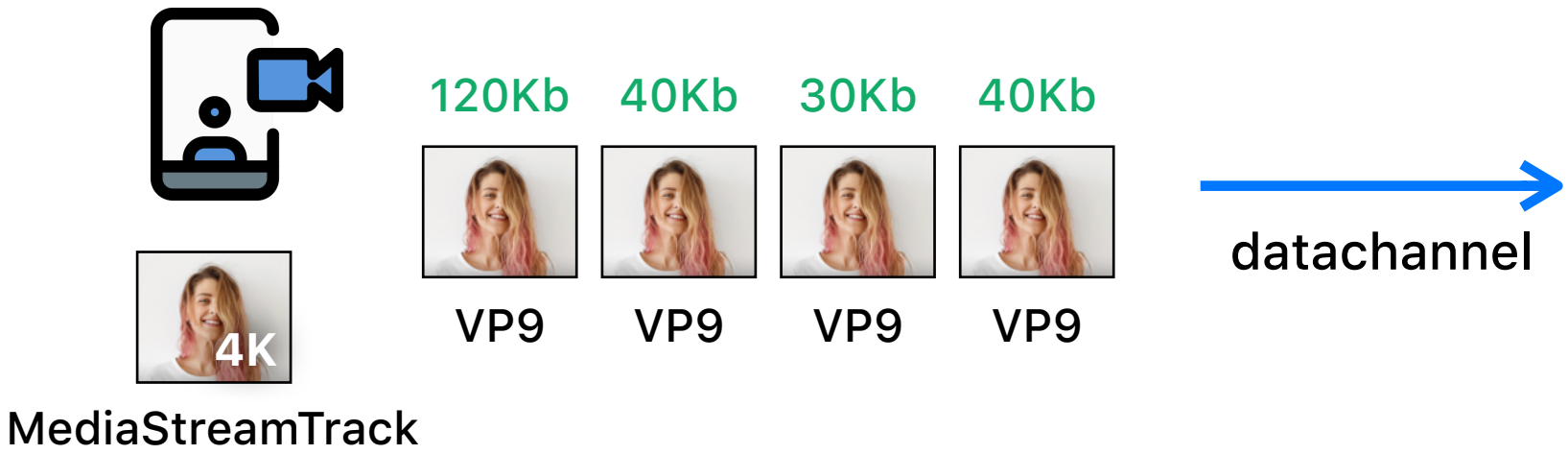


MediaStreamTrack


15Mb
15Mb
15Mb
15Mb
4K
MediaStreamTrack


15Mb
15Mb
15Mb
15Mb
4K
VP9
VP9
VP9
VP9
MediaStreamTrack


120Kb
40Kb
30Kb
40Kb
4K
VP9
VP9
VP9
VP9
MediaStreamTrack


120Kb
40Kb
30Kb
40Kb
4K
VP9
VP9
VP9
VP9
datachannel
MediaStreamTrack

Кодеки

	H.264	VP8	H.265	VP9	AV1
Поддержка браузерами					
Потребление ресурсов	Низкое	Низкое	Среднее	Среднее	Высокое
Эффективность сжатия	Низкая	Низкая	Средняя	Средняя	Высокая
Аппаратное ускорение	Почти везде	Местами	Местами	Местами	Большая редкость

Кодеки

	H.264	VP8	H.265	VP9	AV1
Поддержка браузерами					
Потребление ресурсов	Низкое	Низкое	Среднее	Среднее	Высокое
Эффективность сжатия	Низкая	Низкая	Средняя	Средняя	Высокая
Аппаратное ускорение	Почти везде	Местами	Местами	Местами	Большая редкость

Кодеки

	H.264	VP8	H.265	VP9	AV1
Поддержка браузерами					
Потребление ресурсов	Низкое	Низкое	Среднее	Среднее	Высокое
Эффективность сжатия	Низкая	Низкая	Средняя	Средняя	Высокая
Аппаратное ускорение	Почти везде	Местами	Местами	Местами	Большая редкость

WebCodecs

https://developer.mozilla.org/en-US/docs/Web/API/WebCodecs_API

https://www.w3.org/TR/webcodecs/

WebCodecs

MediaStreamTrackProcessor

MediaStreamTrackGenerator

WebCodecs

MediaStreamTrackProcessor

MediaStreamTrackGenerator

VideoEncoder

VideoDecoder

WebCodecs

MediaStreamTrackProcessor

MediaStreamTrackGenerator

VideoEncoder

VideoDecoder

AudioEncoder

AudioDecoder

WebCodecs

MediaStreamTrackProcessor

MediaStreamTrackGenerator

VideoEncoder

VideoDecoder

AudioEncoder

AudioDecoder

WebCodecs

VideoFrame

AudioData

MediaStreamTrackProcessor

MediaStreamTrackGenerator

VideoEncoder

VideoDecoder

AudioEncoder

AudioDecoder

WebCodecs

VideoFrame

AudioData

EncodedVideoChunk

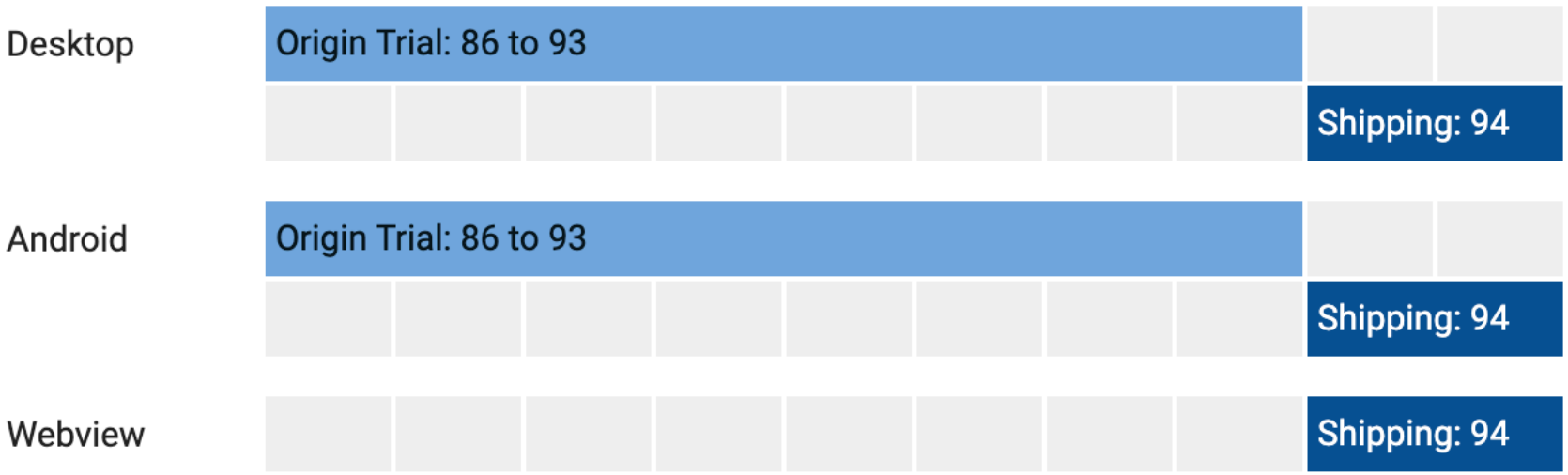
EncodedAudioChunk

Status in Chromium

Blink components: Blink>Media

Implementation status: **Enabled by default** (tracking bug)

Estimated milestones:



Status in Chromium

Blink components: Blink>Media

Implementation status: **Enabled by default** (tracking bug)

Estimated milestones:

Desktop	Origin Trial: 86 to 93	
	Chrome 86: October 6, 2020	Shipping: 94
Android	Origin Trial: 86 to 93	
		Shipping: 94
Webview		Shipping: 94

Status in Chromium

Blink components: Blink>Media

Implementation status: **Enabled by default** (tracking bug)

Estimated milestones:

Chrome 94: September 21, 2021

Desktop	Origin Trial: 86 to 93	
	Chrome 86: October 6, 2020	Shipping: 94
Android	Origin Trial: 86 to 93	
		Shipping: 94
Webview		Shipping: 94

Chrome Origin Trials

https://developer.chrome.com/origintrials/

```html
<meta http-equiv="origin-trial" content="TOKEN"></meta>
```

Trial for WebCodecs

COMPLETE

Provides efficient, low-level access to built-in (software and hardware) media encoders and decoders.
Learn more

Available	Chrome 86 to 93
End date	2 дек. 2021 г.

Resources

Chrome Platform Status Feature Feedback Intent to Experiment

2012	2018	05.2020	09.2020	02.2021
Start: web	mobile	8 group calls	128	

07.2021	08.2021	10.2021	11.2021	04.2022
4K	VK Звонки desktop 2048	SDK	∞ участников	VK Звонки mobile app

09.2020

128

02.2021

07.2021

4K

```javascript
const trackProcessor = new MediaStreamTrackProcessor(track);
const frameReader = trackProcessor.readable.getReader();

const encoder = new VideoEncoder({
    output: (chunk) => {
        const data = new ArrayBuffer(chunk.byteLength);
        chunk.copyTo(data);
        send(data);
    },
    error: (error) => {},
});
encoder.configure({
    codec: 'vp09.00.50.08',
    width: track.getSettings().width,
    height: track.getSettings().height,
    latencyMode: 'realtime',
});

const {done, value} = await frameReader.read();
encoder.encode(frame);
```

```
const trackProcessor = new MediaStreamTrackProcessor(track);
const frameReader = trackProcessor.readable.getReader();

const encoder = new VideoEncoder({
    output: (chunk) => {
        const data = new ArrayBuffer(chunk.byteLength);
        chunk.copyTo(data);
        send(data);
    },
    error: (error) => {},
});
encoder.configure({
    codec: 'vp09.00.50.08',
    width: track.getSettings().width,
    height: track.getSettings().height,
    latencyMode: 'realtime',
});

const {done, value} = await frameReader.read();
encoder.encode(frame);
```

```
const trackProcessor = new MediaStreamTrackProcessor(track);
const frameReader = trackProcessor.readable.getReader();

const encoder = new VideoEncoder({
    output: (chunk) => {
        const data = new ArrayBuffer(chunk.byteLength);
        chunk.copyTo(data);
        send(data);
    },
    error: (error) => {},
});
encoder.configure({
    codec: 'vp09.00.50.08',
    width: track.getSettings().width,
    height: track.getSettings().height,
    latencyMode: 'realtime',
});

const {done, value} = await frameReader.read();
encoder.encode(frame);
```

```
const trackProcessor = new MediaStreamTrackProcessor(track);
const frameReader = trackProcessor.readable.getReader();

const encoder = new VideoEncoder({
    output: (chunk) => {
        const data = new ArrayBuffer(chunk.byteLength);
        chunk.copyTo(data);
        send(data);
    },
    error: (error) => {},
});
encoder.configure({
    codec: 'vp09.00.50.08',
    width: track.getSettings().width,
    height: track.getSettings().height,
    latencyMode: 'realtime',
});

const {done, value} = await frameReader.read();
encoder.encode(frame);
```

```
const trackProcessor = new MediaStreamTrackProcessor(track);
const frameReader = trackProcessor.readable.getReader();

const encoder = new VideoEncoder({
    output: (chunk) => {
        const data = new ArrayBuffer(chunk.byteLength);
        chunk.copyTo(data);
        send(data);
    },
    error: (error) => {},
});
encoder.configure({
    codec: 'vp09.00.50.08',
    width: track.getSettings().width,
    height: track.getSettings().height,
    latencyMode: 'realtime',
});

const {done, value} = await frameReader.read();
encoder.encode(frame);
```

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
`VideoEncoder`	✓ 94	✓ 94	✗ No	✗ No	✓ 80	✗ No	✓ 94	✗ No	✓ 66	✗ No	17.0	✓ 94
`VideoEncoder()` constructor	✓ 94	✓ 94	✗ No	✗ No	✓ 80	✗ No	✓ 94	✗ No	✓ 66	✗ No	17.0	✓ 94
`close`	✓ 94	✓ 94	✗ No	✗ No	✓ 80	✗ No	✓ 94	✗ No	✓ 66	✗ No	17.0	✓ 94
`configure`	✓ 94	✓ 94	✗ No	✗ No	✓ 80	✗ No	✓ 94	✗ No	✓ 66	✗ No	17.0	✓ 94
`encode`	✓ 94	✓ 94	✗ No	✗ No	✓ 80	✗ No	✓ 94	✗ No	✓ 66	✗ No	17.0	✓ 94

libvpx

https://chromium.googlesource.com/webm/libvpx

emscripten

https://emscripten.org/

WEBASSEMBLY

https://webassembly.org/

libvpx.js

libvpx.wasm

libvpx.d.ts

```typescript
interface VpxEncoder {
    init(type: VpxType): boolean;
    debug(enabled: boolean): void;
    setMaxQuantizer(quantizer: number): void;
    setTargetBitrate(bitrate: number): void;
    allocateImage(width: number, height: number): Uint8Array;
    encode(pts: number, duration: number, keyframe: boolean): boolean;
    readFrame(): Uint8Array;
    delete(): void;
}

interface VpxDecoder {
    init(type: VpxType): boolean;
    debug(enabled: boolean): void;
    allocateBuffer(size: number): Uint8Array;
    decode(): boolean;
    nextImage(): boolean;
    getImageWidth(): number;
    getImageHeight(): number;
    getImageBuffer(): Uint8Array;
}
```

```
const canvas = document.createElement('canvas');
const canvasCtx = canvas.getContext('2d');
const video = document.createElement('video');
video.autoplay = true;
video.srcObject = new MediaStream([track]);

canvasCtx.clearRect(0, 0, width, height);
canvasCtx.drawImage(video, 0, 0, width, height);
const imageData = canvasCtx.getImageData(0, 0, width, height);

const encoder = new VpxEncoder();
encoder.init(VpxType.VP9);

const buffer = encoder.allocateImage(width, height);
buffer.set(imageData.data);
encoder.encode(pts, duration, keyFrame);

const frameData = encoder.readFrame();
send(frameData);
```

```
const canvas = document.createElement('canvas');
const canvasCtx = canvas.getContext('2d');
const video = document.createElement('video');
video.autoplay = true;
video.srcObject = new MediaStream([track]);

canvasCtx.clearRect(0, 0, width, height);
canvasCtx.drawImage(video, 0, 0, width, height);
const imageData = canvasCtx.getImageData(0, 0, width, height);

const encoder = new VpxEncoder();
encoder.init(VpxType.VP9);

const buffer = encoder.allocateImage(width, height);
buffer.set(imageData.data);
encoder.encode(pts, duration, keyFrame);

const frameData = encoder.readFrame();
send(frameData);
```

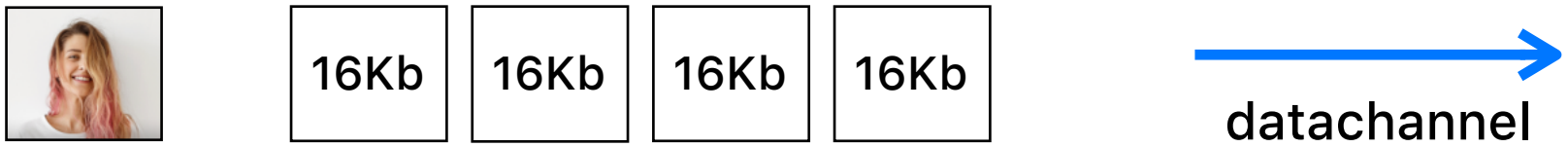
```
const canvas = document.createElement('canvas');
const canvasCtx = canvas.getContext('2d');
const video = document.createElement('video');
video.autoplay = true;
video.srcObject = new MediaStream([track]);

canvasCtx.clearRect(0, 0, width, height);
canvasCtx.drawImage(video, 0, 0, width, height);
const imageData = canvasCtx.getImageData(0, 0, width, height);

const encoder = new VpxEncoder();
encoder.init(VpxType.VP9);

const buffer = encoder.allocateImage(width, height);
buffer.set(imageData.data);
encoder.encode(pts, duration, keyFrame);

const frameData = encoder.readFrame();
send(frameData);
```

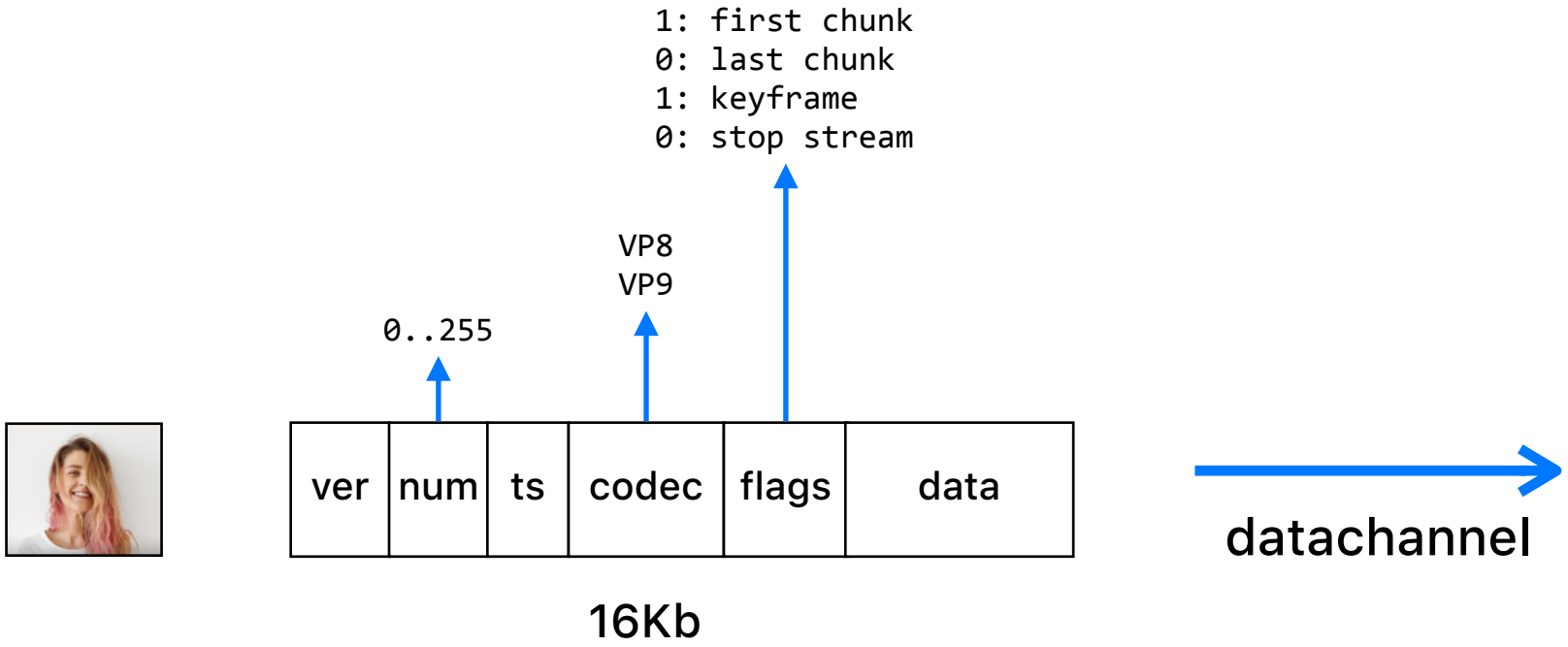
```
const canvas = document.createElement('canvas');
const canvasCtx = canvas.getContext('2d');
const video = document.createElement('video');
video.autoplay = true;
video.srcObject = new MediaStream([track]);

canvasCtx.clearRect(0, 0, width, height);
canvasCtx.drawImage(video, 0, 0, width, height);
const imageData = canvasCtx.getImageData(0, 0, width, height);

const encoder = new VpxEncoder();
encoder.init(VpxType.VP9);

const buffer = encoder.allocateImage(width, height);
buffer.set(imageData.data);
encoder.encode(pts, duration, keyFrame);

const frameData = encoder.readFrame();
send(frameData);
```

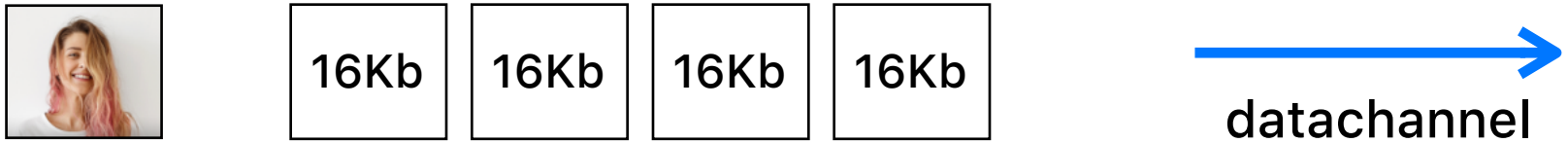
```
const canvas = document.createElement('canvas');
const canvasCtx = canvas.getContext('2d');
const video = document.createElement('video');
video.autoplay = true;
video.srcObject = new MediaStream([track]);

canvasCtx.clearRect(0, 0, width, height);
canvasCtx.drawImage(video, 0, 0, width, height);
const imageData = canvasCtx.getImageData(0, 0, width, height);

const encoder = new VpxEncoder();
encoder.init(VpxType.VP9);

const buffer = encoder.allocateImage(width, height);
buffer.set(imageData.data);
encoder.encode(pts, duration, keyFrame);

const frameData = encoder.readFrame();
send(frameData);
```

```
const canvas = document.createElement('canvas');
const canvasCtx = canvas.getContext('2d');
const video = document.createElement('video');
video.autoplay = true;
video.srcObject = new MediaStream([track]);

canvasCtx.clearRect(0, 0, width, height);
canvasCtx.drawImage(video, 0, 0, width, height);
const imageData = canvasCtx.getImageData(0, 0, width, height);

const encoder = new VpxEncoder();
encoder.init(VpxType.VP9);

const buffer = encoder.allocateImage(width, height);
buffer.set(imageData.data);
encoder.encode(pts, duration, keyFrame);

const frameData = encoder.readFrame();
send(frameData);
```

datachannel


16Kb
16Kb
16Kb
16Kb
datachannel


1: first chunk
0: last chunk
1: keyframe
0: stop stream
VP8
VP9
0..255
ver
num
ts
codec
flags
data
16Kb
datachannel


16Kb
16Kb
16Kb
16Kb
datachannel

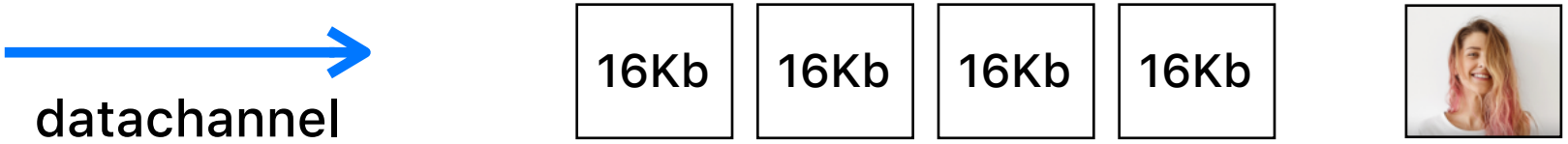
datachannel

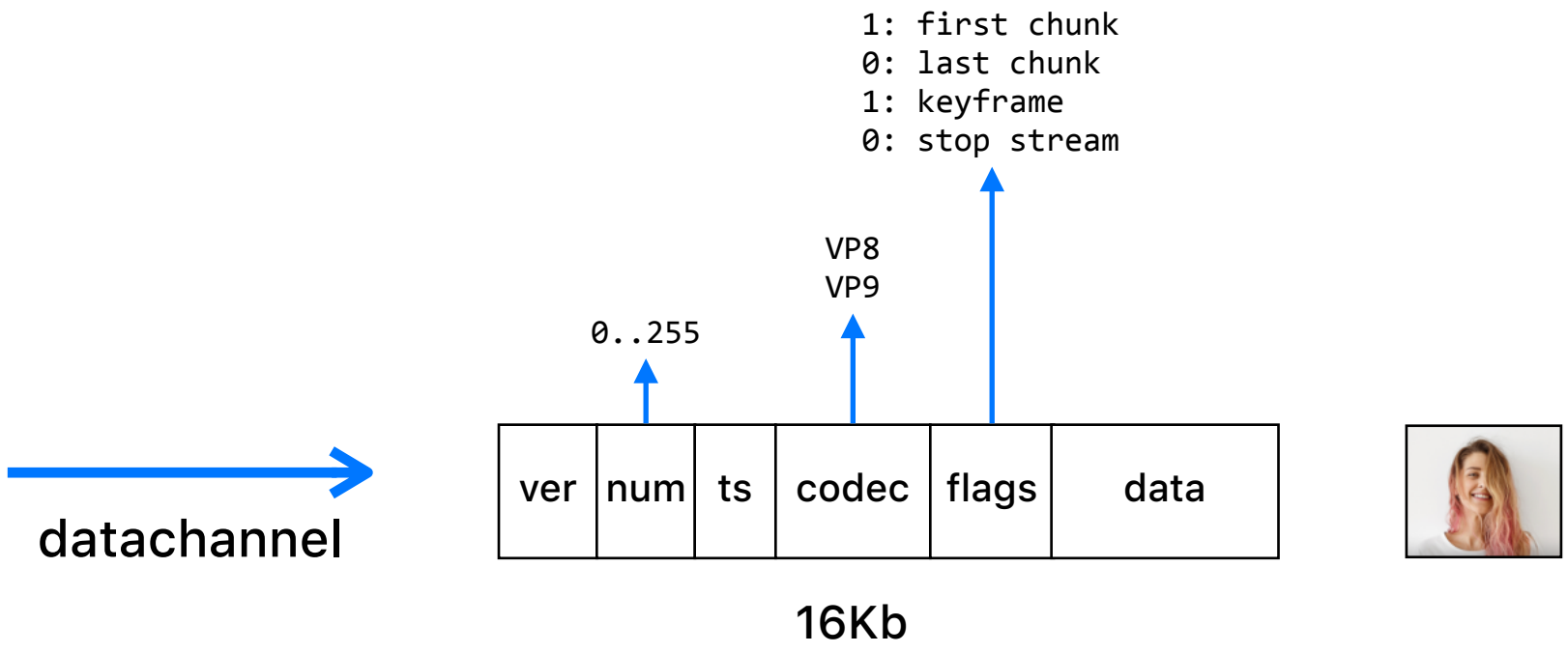
```
Uncaught Error: Failed to execute 'send' on 'RTCDataChannel': RTCDataChannel.readyState is not 'open'
    at RTCDataChannelWeb.send$1 (main.dart.js:106103)
    at main.dart.js:107225
    at _wrapJsFunctionForAsync_closure.$protected (main.dart.js:8751)
    at _wrapJsFunctionForAsync_closure.call$2 (main.dart.js:47579)
    at Object._asyncStartSync (main.dart.js:8715)
    at _DataChannelSampleState._handleDataChannelTest$body$_DataChannelSampleState (main.dart.js:107236)
    at _DataChannelSampleState._handleDataChannelTest$1 (main.dart.js:107205)
    at Object.eval (eval at Closure_forwardCallTo (main.dart.js:5859), <anonymous>:3:41)
    at _TimerImpl$periodic_closure.call$0 (main.dart.js:47533)
    at invokeClosure (main.dart.js:5700)
```

```
datachannel.onbufferedamountlow = () => {
    if (datachannel.bufferedAmount <= datachannel.bufferedAmountLowThreshold) {
        sendNextChunk();
    }
};
```

Что мы узнали?

- WebCodecs – это круто
- WASM – мощный волчара
- Не все кодеки одинаково полезны
- DataChannel – привередливый


datachannel
16Kb
16Kb
16Kb
16Kb


1: first chunk
0: last chunk
1: keyframe
0: stop stream
VP8
VP9
0..255
datachannel
ver
num
ts
codec
flags
data
16Kb

datachannel
VP9
VP9
VP9
VP9

MediaStreamTrack

```javascript
const decoder = new VideoDecoder({
    output: async (frame) => {
        const image = await createImageBitmap(frame);
        render(image);
        image.close();
        frame.close();
    },
    error: (e) => {},
});
decoder.configure({
    codec: 'vp09.00.50.08',
});

const chunk = new EncodedVideoChunk({
    type: keyFrame ? 'key' : 'delta',
    timestamp,
    data,
});
decoder.decode(chunk);
```

```
const decoder = new VideoDecoder({
    output: async (frame) => {
        const image = await createImageBitmap(frame);
        render(image);
        image.close();
        frame.close();
    },
    error: (e) => {},
});
decoder.configure({
    codec: 'vp09.00.50.08',
});

const chunk = new EncodedVideoChunk({
    type: keyFrame ? 'key' : 'delta',
    timestamp,
    data,
});
decoder.decode(chunk);
```

```
const decoder = new VideoDecoder({
    output: async (frame) => {
        const image = await createImageBitmap(frame);
        render(image);
        image.close();
        frame.close();
    },
    error: (e) => {},
});
decoder.configure({
    codec: 'vp09.00.50.08',
});

const chunk = new EncodedVideoChunk({
    type: keyFrame ? 'key' : 'delta',
    timestamp,
    data,
});
decoder.decode(chunk);
```

```
const decoder = new VideoDecoder({
    output: async (frame) => {
        const image = await createImageBitmap(frame);
        render(image);
        image.close();
        frame.close();
    },
    error: (e) => {},
});
decoder.configure({
    codec: 'vp09.00.50.08',
});

const chunk = new EncodedVideoChunk({
    type: keyFrame ? 'key' : 'delta',
    timestamp,
    data,
});
decoder.decode(chunk);
```

```
const decoder = new VpxDecoder();
decoder.init(VpxType.VP9);

const buffer = decoder.allocateBuffer(data.byteLength);
buffer.set(new Uint8Array(data));

decoder.decode();
const frameData = decoder.getImageBuffer();

const result = new Uint8ClampedArray(frameData.byteLength);
result.set(frameData);

const imageData = new ImageData(result, width, height);
const imageBitmap = await createImageBitmap(imageData, 0, 0, width, height);
render(imageBitmap);
```

```
const decoder = new VpxDecoder();
decoder.init(VpxType.VP9);

const buffer = decoder.allocateBuffer(data.byteLength);
buffer.set(new Uint8Array(data));

decoder.decode();
const frameData = decoder.getImageBuffer();

const result = new Uint8ClampedArray(frameData.byteLength);
result.set(frameData);

const imageData = new ImageData(result, width, height);
const imageBitmap = await createImageBitmap(imageData, 0, 0, width, height);
render(imageBitmap);
```

```
const decoder = new VpxDecoder();
decoder.init(VpxType.VP9);

const buffer = decoder.allocateBuffer(data.byteLength);
buffer.set(new Uint8Array(data));

decoder.decode();
const frameData = decoder.getImageBuffer();

const result = new Uint8ClampedArray(frameData.byteLength);
result.set(frameData);

const imageData = new ImageData(result, width, height);
const imageBitmap = await createImageBitmap(imageData, 0, 0, width, height);
render(imageBitmap);
```

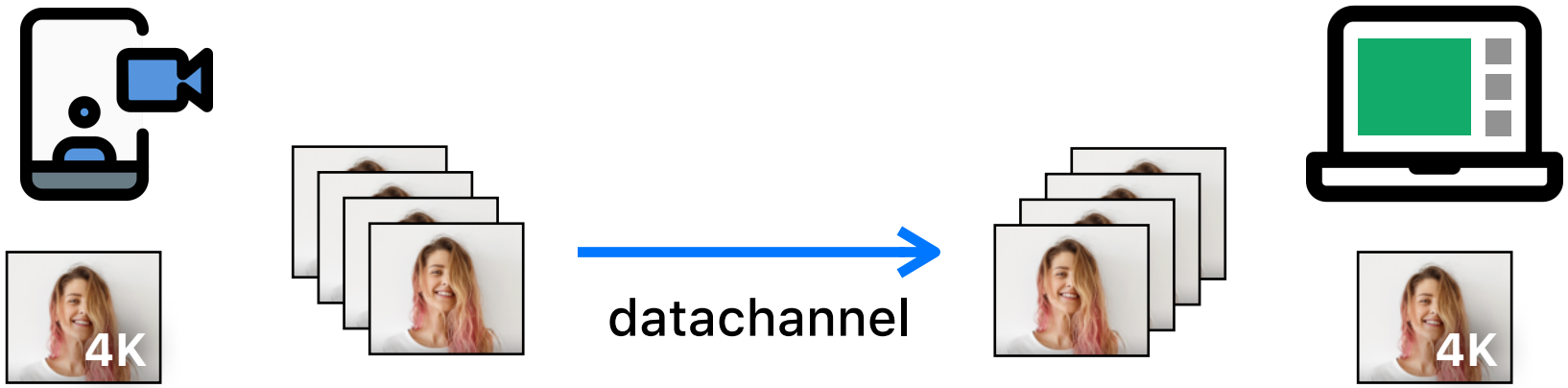
```
const decoder = new VpxDecoder();
decoder.init(VpxType.VP9);

const buffer = decoder.allocateBuffer(data.byteLength);
buffer.set(new Uint8Array(data));

decoder.decode();
const frameData = decoder.getImageBuffer();

const result = new Uint8ClampedArray(frameData.byteLength);
result.set(frameData);

const imageData = new ImageData(result, width, height);
const imageBitmap = await createImageBitmap(imageData, 0, 0, width, height);
render(imageBitmap);
```

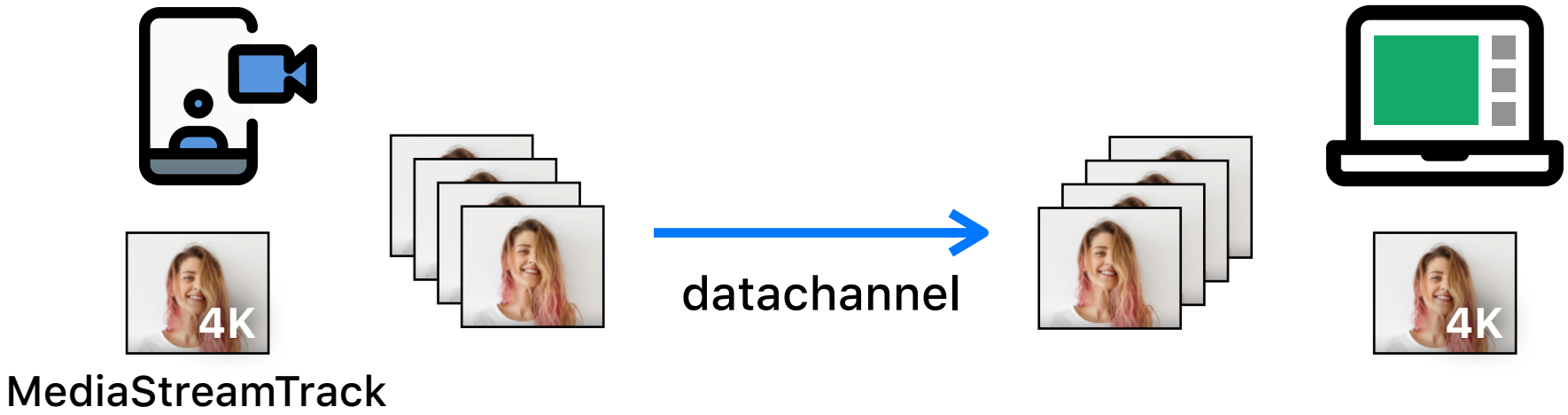
```
const decoder = new VpxDecoder();
decoder.init(VpxType.VP9);

const buffer = decoder.allocateBuffer(data.byteLength);
buffer.set(new Uint8Array(data));

decoder.decode();
const frameData = decoder.getImageBuffer();

const result = new Uint8ClampedArray(frameData.byteLength);
result.set(frameData);

const imageData = new ImageData(result, width, height);
const imageBitmap = await createImageBitmap(imageData, 0, 0, width, height);
render(imageBitmap);
```

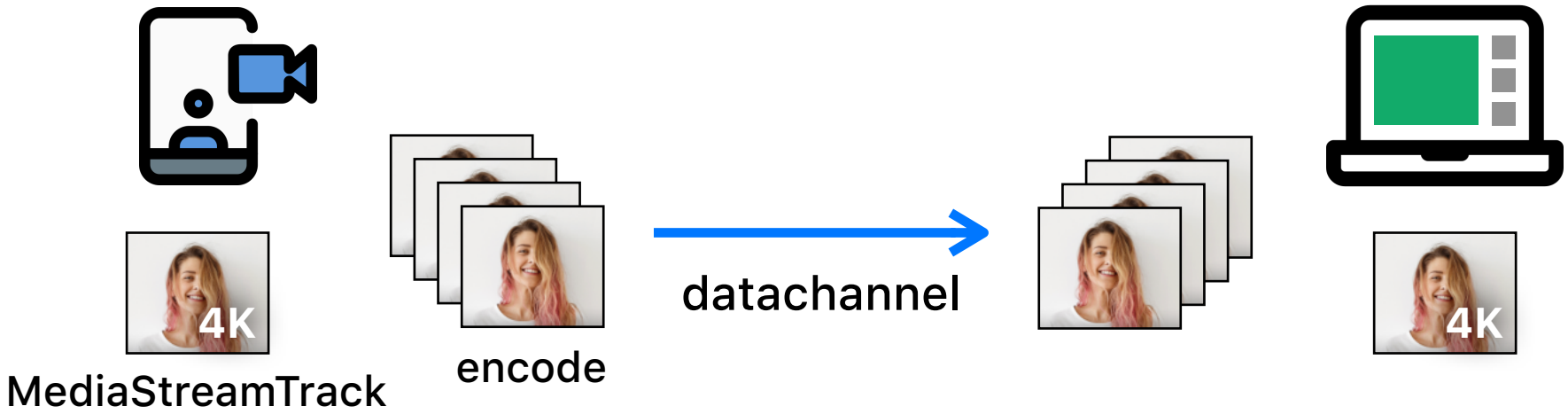
```
const canvas = document.createElement('canvas');
canvas.width = width;
canvas.height = height;

const stream = canvas.captureStream(25);

const canvasContext = canvas.getContext('bitmaprenderer');
canvasContext.transferFromImageBitmap(image);
```

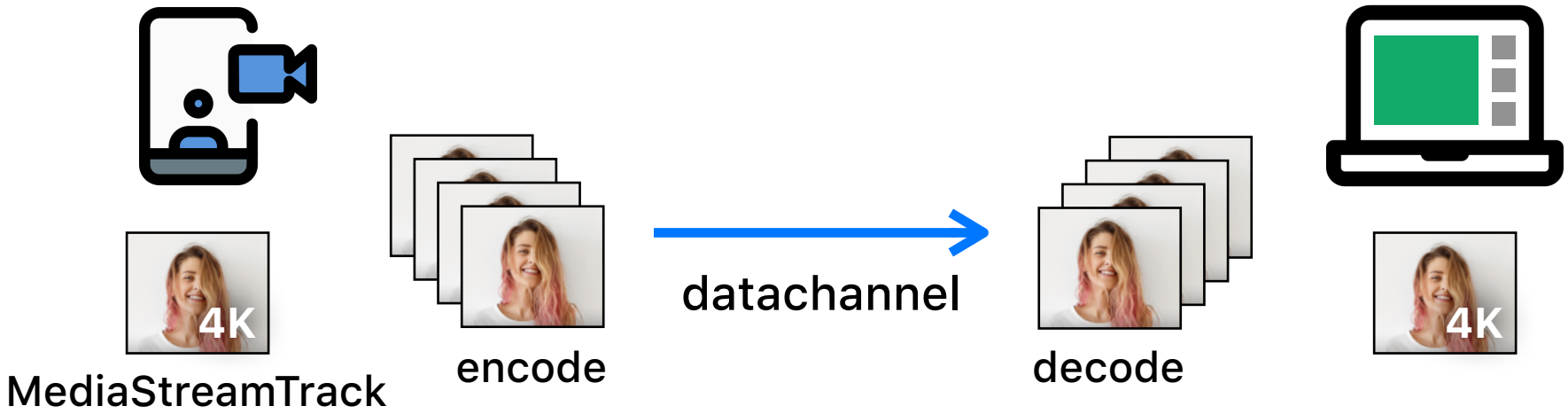
```
const canvas = document.createElement('canvas');
canvas.width = width;
canvas.height = height;

const stream = canvas.captureStream(25);

const canvasContext = canvas.getContext('bitmaprenderer');
canvasContext.transferFromImageBitmap(image);
```

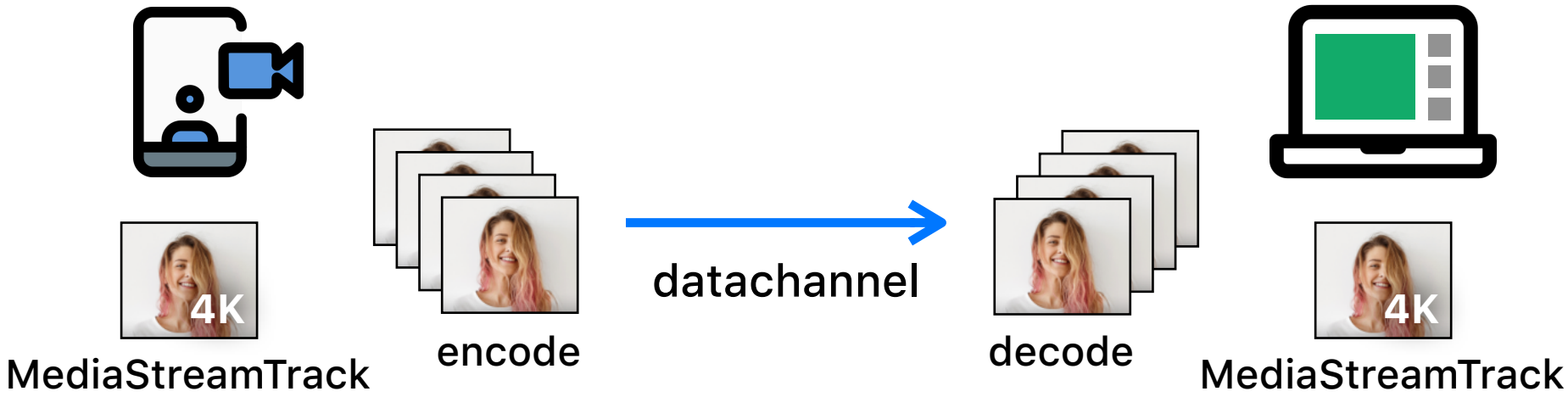
```
const canvas = document.createElement('canvas');
canvas.width = width;
canvas.height = height;

const stream = canvas.captureStream(25);

const canvasContext = canvas.getContext('bitmaprenderer');
canvasContext.transferFromImageBitmap(image);
```

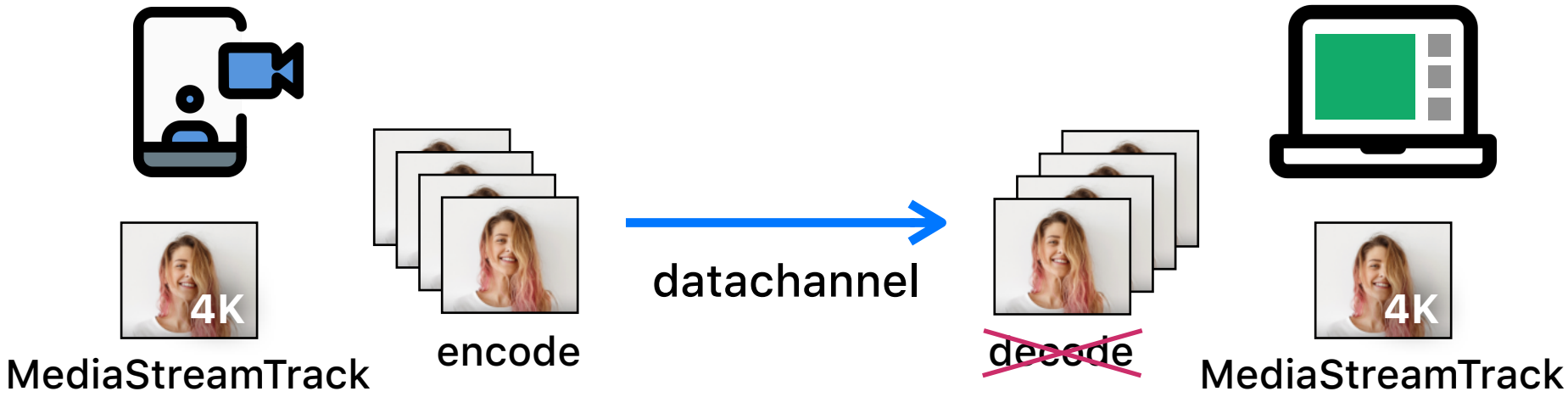

4K
datachannel
4K


4K
MediaStreamTrack
datachannel
4K


4K
MediaStreamTrack
encode
datachannel
4K


4K
MediaStreamTrack
encode
datachannel
decode
4K

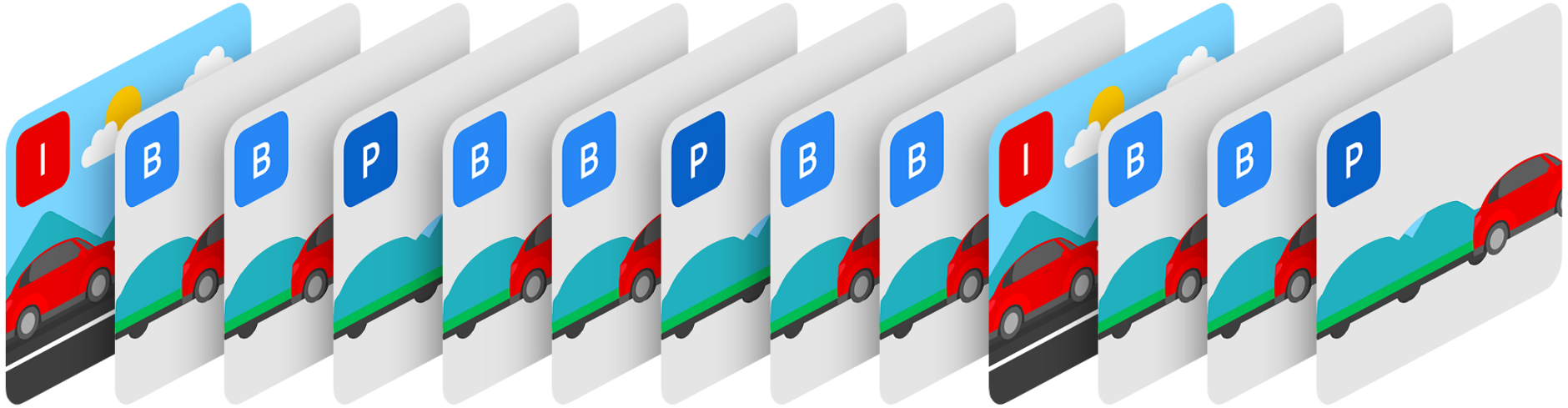

4K
MediaStreamTrack
encode
datachannel
decode
4K
MediaStreamTrack


4K
MediaStreamTrack
encode
datachannel
decode
4K
MediaStreamTrack

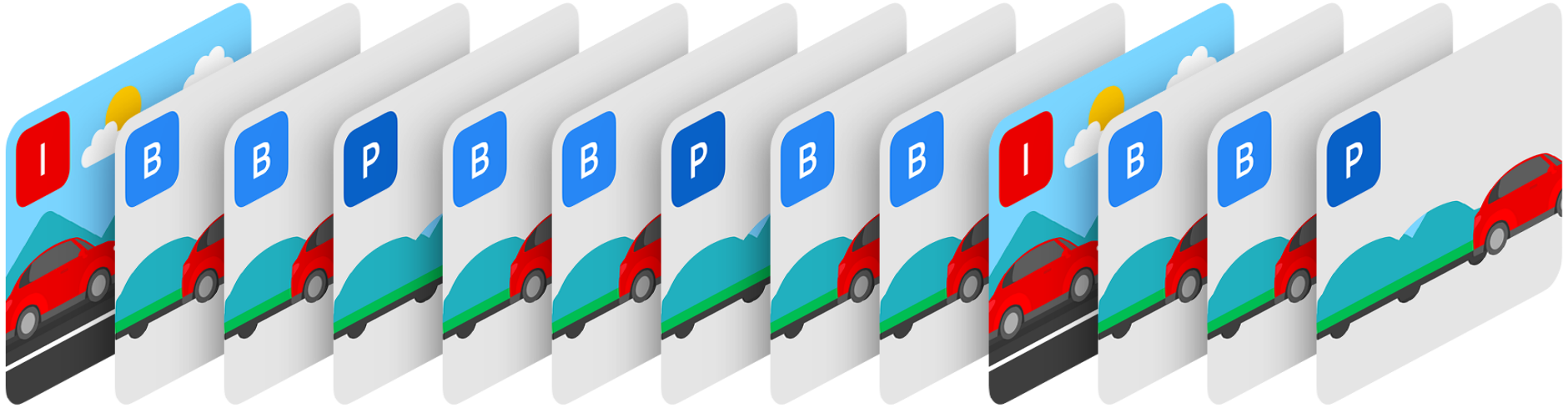
WebM

вконтакте
Павел
1:15:37 · 128 участников
Говорят Павел, Арина
financial highlights
Вы
Наташа
Арина
Ян
12:48
1:15:37

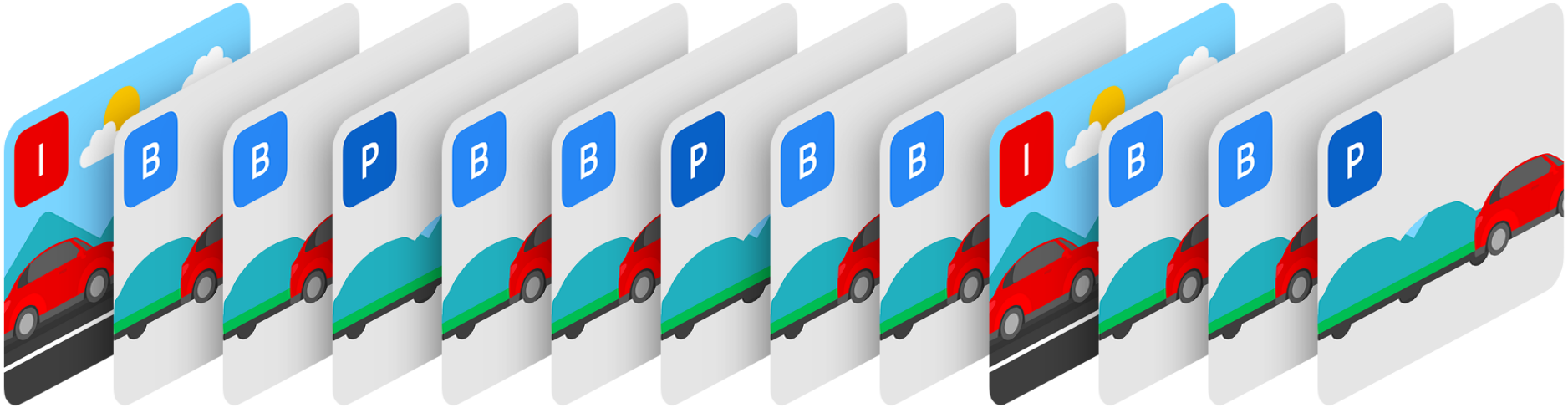
0:51 / 9:56



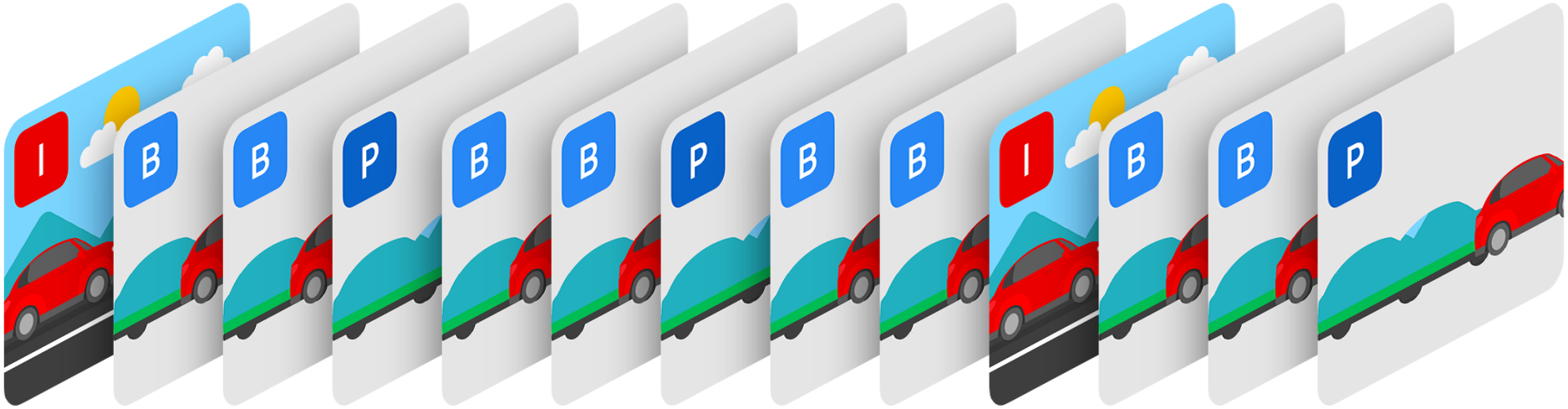
H264 VP8 VP9



H264 VP8 VP9



H264 VP8 VP9



H264 VP8 VP9

WebM

https://ru.wikipedia.org/wiki/WebM

- Открытый
- VP8, VP9, AV1
- Простой
- Поддерживается браузерами *

Matröška

https://ru.wikipedia.org/wiki/Matroska

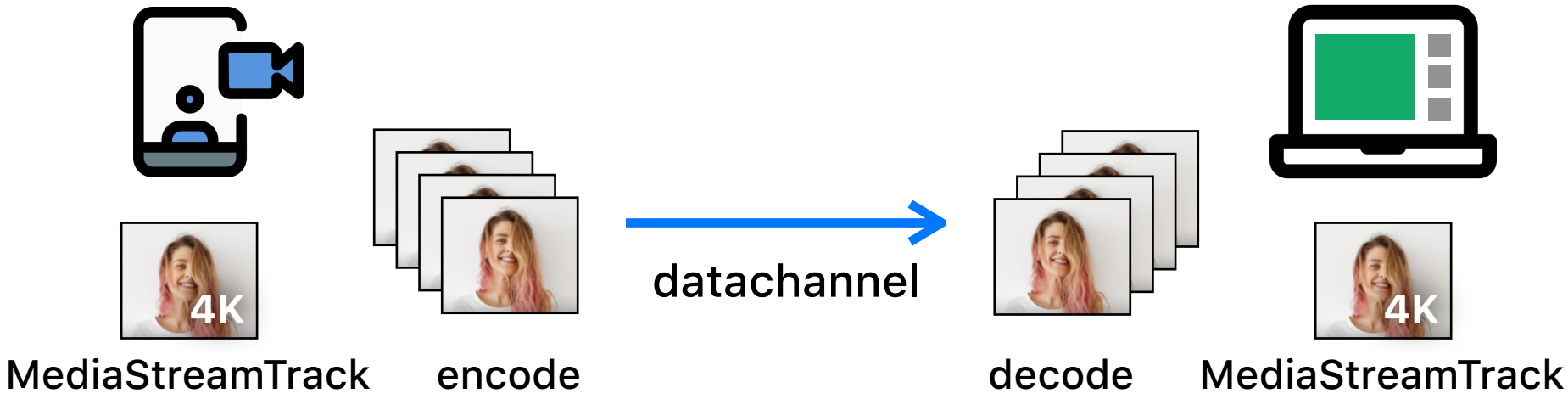
WebM video format - OTHER

Multimedia format designed to provide a royalty-free, high-quality open video compression format for use with HTML5 video. WebM supports the video codec VP8 and VP9.

Usage	% of all users ?
Global	77.41% + 18.43% = 95.84%

Current aligned | Usage relative | Date relative | Filtered | All

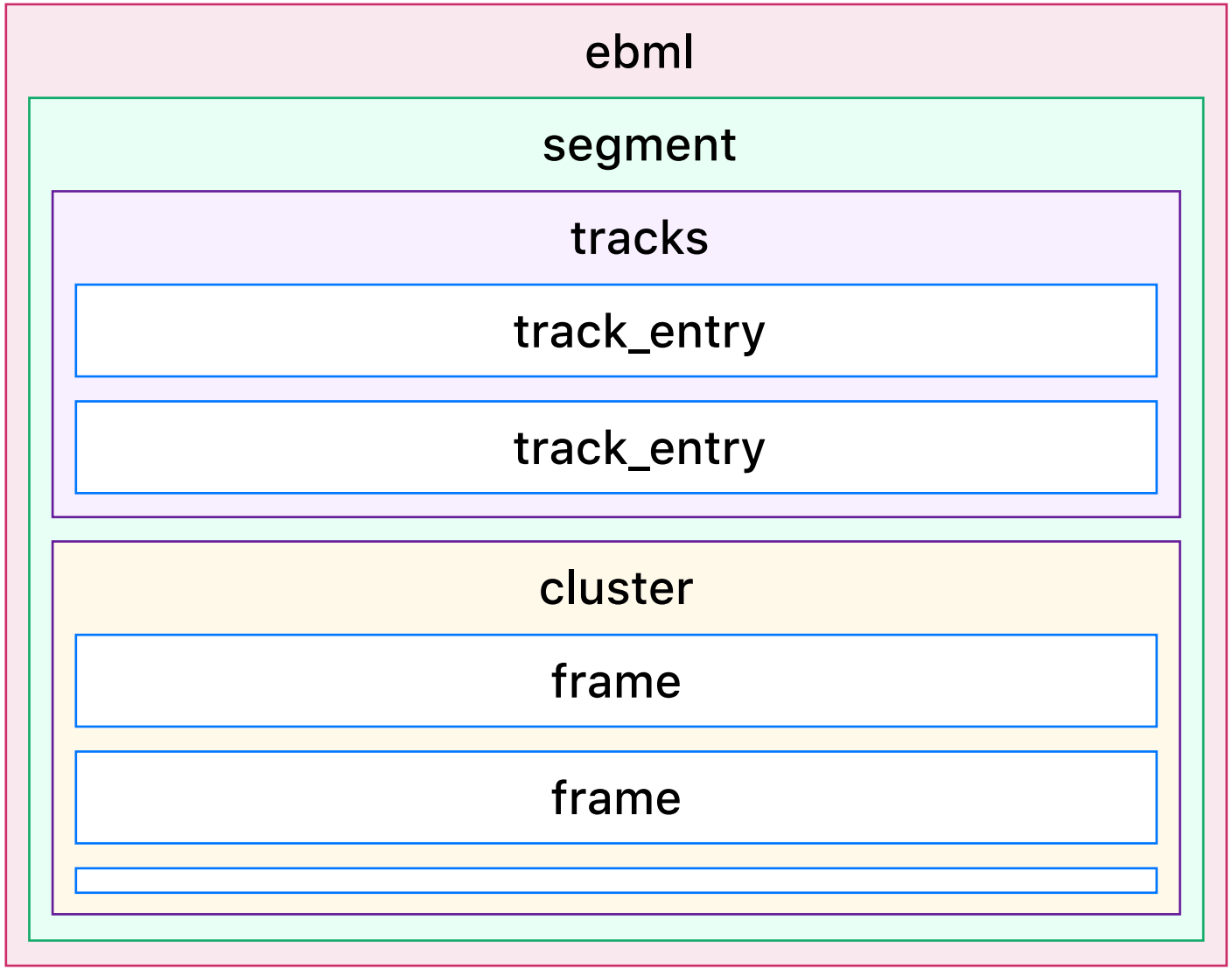
IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
						[4] 12.5				
						[4 5] 14.4				
						[4 5] 14.8				
	99	98	99	[7] 14.1		[4 5] 15.1				
	100	99	100	[7] 15.4	85	[4 5] 15.3				
[3] 11	101	100	101	[7] 15.5	86	[4 5] 15.4	all	101	12.12	16.0
		101	102	[7] TP	87					
		102	103							
			104							



* в браузерах без поддержки WebM

EBML – Extensible Binary Meta Language

https://ru.wikipedia.org/wiki/EBML


ebml
segment
tracks
track_entry
track_entry
cluster
frame
frame

simple-ebml-builder

https://www.npmjs.com/package/simple-ebml-builder

```
EBML.element(EBML.ID.EBML, [
    EBML.element(EBML.ID.EBMLVersion, EBML.number(1)),
    EBML.element(EBML.ID.EBMLReadVersion, EBML.number(1)),
    EBML.element(EBML.ID.EBMLMaxIDLength, EBML.number(4)),
    EBML.element(EBML.ID.EBMLMaxSizeLength, EBML.number(8)),
    EBML.element(EBML.ID.DocType, EBML.string('webm')),
    EBML.element(EBML.ID.DocTypeVersion, EBML.number(2)),
    EBML.element(EBML.ID.DocTypeReadVersion, EBML.number(2)),
]);
```

Media Source Extensions

https://developer.mozilla.org/en-US/docs/Web/API/Media_Source_Extensions_API

Media Source API

Media Source API

MediaSource

```
video.src = URL.createObjectURL(mediaSource);
```

Media Source API

MediaSource

```
video.src = URL.createObjectURL(mediaSource);
```

Media Source API

SourceBuffer

Media Source API

MediaSource

```
video.src = URL.createObjectURL(mediaSource);
```

SourceBuffer

```
sourceBuffer.appendBuffer(data);
```

Media Source Extensions - REC

API allowing media data to be accessed from HTML `video` and `audio` elements.

Usage	% of all users ?			
Global	80.48%	+ 14.63%	=	95.11%
unprefixed:	80.43%	+ 14.63%	=	95.07%

Current aligned | Usage relative | Date relative | Filtered | All

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
						12.5				
						[2] 14.4				
						[2] 14.8				
	99	98	99	14.1		[2] 15.1				
	100	99	100	15.4	85	[2] 15.3				
[1] 11	101	100	101	15.5	86	[2] 15.4	all	101	12.12	16.0
		101	102	TP	87					
		102	103							
			104							

```
const video = document.createElement('video');
video.autoplay = true;

const mediaSource = new MediaSource();
const sourceBuffer = mediaSource.addSourceBuffer('video/webm; codecs="vp9"');
sourceBuffer.mode = 'sequence';

video.src = URL.createObjectURL(mediaSource);

const webmHeader = genWebmHeader();
sourceBuffer.appendBuffer(webmHeader);
const segmentHeader = genSegmentHeader(frame.width, frame.height, codec);
sourceBuffer.appendBuffer(segmentHeader);
const cluster = genClusterHeader();
sourceBuffer.appendBuffer(cluster);
const frameBlock = genFrame(data);
sourceBuffer.appendBuffer(frameBlock);
```

```
const video = document.createElement('video');
video.autoplay = true;

const mediaSource = new MediaSource();
const sourceBuffer = mediaSource.addSourceBuffer('video/webm; codecs="vp9"');
sourceBuffer.mode = 'sequence';

video.src = URL.createObjectURL(mediaSource);

const webmHeader = genWebmHeader();
sourceBuffer.appendBuffer(webmHeader);
const segmentHeader = genSegmentHeader(frame.width, frame.height, codec);
sourceBuffer.appendBuffer(segmentHeader);
const cluster = genClusterHeader();
sourceBuffer.appendBuffer(cluster);
const frameBlock = genFrame(data);
sourceBuffer.appendBuffer(frameBlock);
```

```
const video = document.createElement('video');
video.autoplay = true;

const mediaSource = new MediaSource();
const sourceBuffer = mediaSource.addSourceBuffer('video/webm; codecs="vp9"');
sourceBuffer.mode = 'sequence';

video.src = URL.createObjectURL(mediaSource);

const webmHeader = genWebmHeader();
sourceBuffer.appendBuffer(webmHeader);
const segmentHeader = genSegmentHeader(frame.width, frame.height, codec);
sourceBuffer.appendBuffer(segmentHeader);
const cluster = genClusterHeader();
sourceBuffer.appendBuffer(cluster);
const frameBlock = genFrame(data);
sourceBuffer.appendBuffer(frameBlock);
```

```
const video = document.createElement('video');
video.autoplay = true;

const mediaSource = new MediaSource();
const sourceBuffer = mediaSource.addSourceBuffer('video/webm; codecs="vp9"');
sourceBuffer.mode = 'sequence';

video.src = URL.createObjectURL(mediaSource);

const webmHeader = genWebmHeader();
sourceBuffer.appendBuffer(webmHeader);
const segmentHeader = genSegmentHeader(frame.width, frame.height, codec);
sourceBuffer.appendBuffer(segmentHeader);
const cluster = genClusterHeader();
sourceBuffer.appendBuffer(cluster);
const frameBlock = genFrame(data);
sourceBuffer.appendBuffer(frameBlock);
```

```
const video = document.createElement('video');
video.autoplay = true;

const mediaSource = new MediaSource();
const sourceBuffer = mediaSource.addSourceBuffer('video/webm; codecs="vp9"');
sourceBuffer.mode = 'sequence';

video.src = URL.createObjectURL(mediaSource);

const webmHeader = genWebmHeader();
sourceBuffer.appendBuffer(webmHeader);
const segmentHeader = genSegmentHeader(frame.width, frame.height, codec);
sourceBuffer.appendBuffer(segmentHeader);
const cluster = genClusterHeader();
sourceBuffer.appendBuffer(cluster);
const frameBlock = genFrame(data);
sourceBuffer.appendBuffer(frameBlock);
```

```
const video = document.createElement('video');
video.autoplay = true;

const mediaSource = new MediaSource();
const sourceBuffer = mediaSource.addSourceBuffer('video/webm; codecs="vp9"');
sourceBuffer.mode = 'sequence';

video.src = URL.createObjectURL(mediaSource);

const webmHeader = genWebmHeader();
sourceBuffer.appendBuffer(webmHeader);
const segmentHeader = genSegmentHeader(frame.width, frame.height, codec);
sourceBuffer.appendBuffer(segmentHeader);
const cluster = genClusterHeader();
sourceBuffer.appendBuffer(cluster);
const frameBlock = genFrame(data);
sourceBuffer.appendBuffer(frameBlock);
```

```
const video = document.createElement('video');
video.autoplay = true;

const mediaSource = new MediaSource();
const sourceBuffer = mediaSource.addSourceBuffer('video/webm; codecs="vp9"');
sourceBuffer.mode = 'sequence';

video.src = URL.createObjectURL(mediaSource);

const webmHeader = genWebmHeader();
sourceBuffer.appendBuffer(webmHeader);
const segmentHeader = genSegmentHeader(frame.width, frame.height, codec);
sourceBuffer.appendBuffer(segmentHeader);
const cluster = genClusterHeader();
sourceBuffer.appendBuffer(cluster);
const frameBlock = genFrame(data);
sourceBuffer.appendBuffer(frameBlock);
```

Что мы узнали?

- WebM – простой
- WebM – кроссбраузерный *

Итоги

вконтакте
Павел
1:15:37 · 128 участников
Говорят Павел, Арина
financial highlights
Вы
Наташа
Арина
Ян
12:48
1:15:37

- WebCodecs
- DataChannel
- WebM
- MSE

- libvpx
- DataChannel
- canvas

- RTP

2012	2018	05.2020	09.2020	02.2021
Start: web	mobile	8 group calls	128	

07.2021	08.2021	10.2021	11.2021	04.2022
4K	VK Звонки desktop 2048	SDK	∞ участников	VK Звонки mobile app

07.2021

4K

11.2021

∞ участников

Александр Тоболь

Платформа видеоконференций ВКонтакте — сделано удалённо

https://youtu.be/ae101YtVpC0

Иван Григорьев

VK Звонки: платформа видеоконференций без ограничений на количество участников звонка

https://youtu.be/M_1EHbjwII4

Вадим Горбачев

VK Звонки: выходя за лимиты браузера

https://youtu.be/OTtAJUcR1QY

Иван Шафран, Никита Разумный

VK Звонки: соединяем тысячи людей с Android и iOS по WebRTC

https://youtu.be/PlOnMxWKaSY

Павел Мацула

VK Звонки: десктопный клиент для видеозвонков без лимита на число участников — с нуля за 365 дней

https://youtu.be/ZvOAs8iWN-8

Всё

Николай Василькук
@vasilchuk

Всё

Николай Васильчук
@vasilchuk

@vasilchuk

VK Team

Хабр