WebRTC w pigułce #3

WebRTC w praktyce

Mateusz Front

WebRTC w praktyce

- Do czego tego użyć?
- Jak tego użyć?
- Jakie są inne opcje?

Do czego tego użyć?

Zastosowania media streamingu

- Wideokonferencje
- Broadcasting
- VOD

Zastosowania media streamingu

- Wideokonferencje - max 720p, max 30 FPS, latencja max 200ms, ~2,5 Mb/s
- Broadcasting - 720p do 4K, min 24 FPS, latencja 3 do 15s, ~10 Mb/s
- VOD - 1920p i więcej, ~60 FPS, ~30 Mb/s

Zastosowania media streamingu

- Sub-second latency
- > second latency

Zastosowania WebRTC

- Kolejny klon Google Meet
- Video chat zintegrowany z aplikacją
- Videodomofony
- Stream z drona/robota
- Low latency broadcasting

Jak tego użyć?

Przeglądarka

SDP offer-answer

Peer 1

```
const pc = new PeerConnection(configuration);
const offer = await pc.createOffer();
await pc.setLocalDescription(offer);
signaling.sendOffer(offer);
const answer = await signaling.awaitAnswer();
pc.setRemoteDescription(answer);
```

SDP offer-answer

Peer 2

```
const pc = new PeerConnection(configuration);
const offer = await signaling.awaitOffer();
const answer = pc.createAnswer(offer);
await pc.setLocalDescription(answer);
signaling.sendAnswer(answer);
```

ICE

```
pc.onicecandidate = (event) => signaling.sendCandidate(event.candidate);

signaling.oniceandidate = (candidate) => pc.addIceCandidate(candidate);
```

ICE

```
const pc = new PeerConnection(configuration);

pc.onicecandidate = (event) => signaling.sendCandidate(event.candidate);

signaling.oniceandidate = (candidate) => pc.addIceCandidate(candidate);

const offer = ...
```

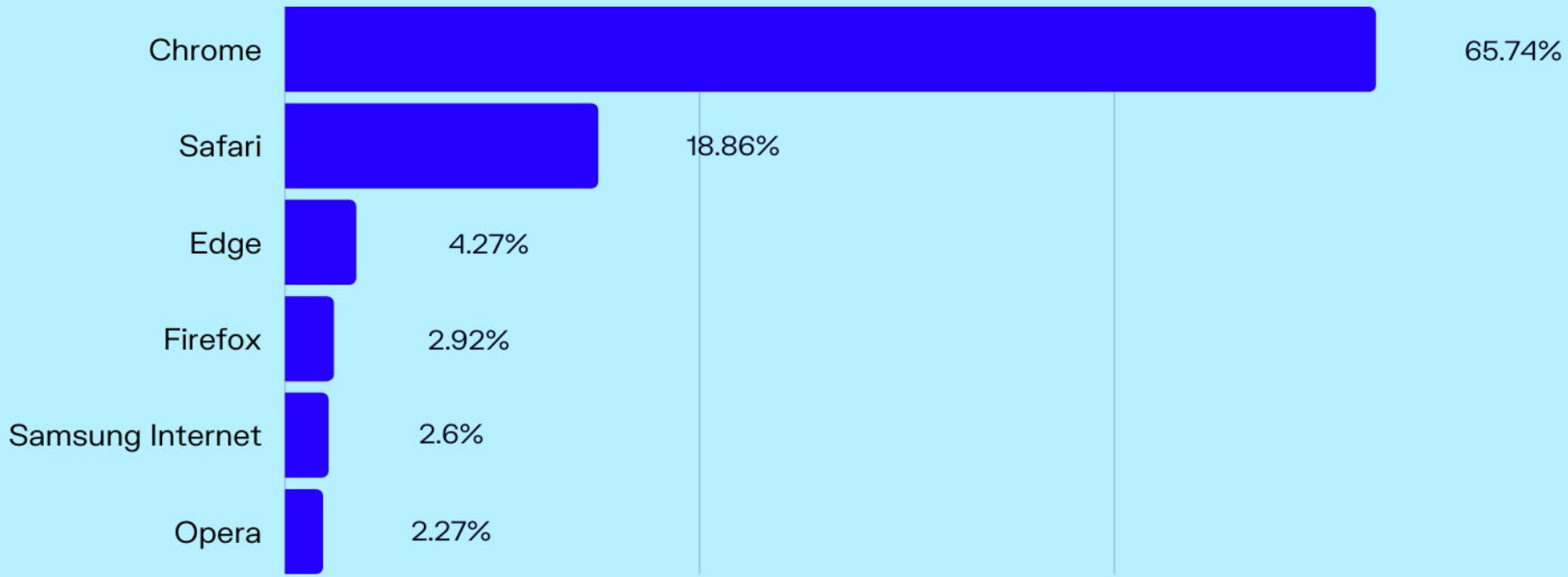
Tracks

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

for (const track of stream.getTracks()) {
  pc.addTrack(track);
}

pc.ontrack = (event) => {
  document.getElementById("video").srcObject = event.streams[0];
}
```

Przeglądarka

- JS API - klasa PeerConnection
- Tryb headless & Playwright
- WebRTC internals

Most Popular Browsers in 2023



Source: Gs.statcounter.com

OBERLO

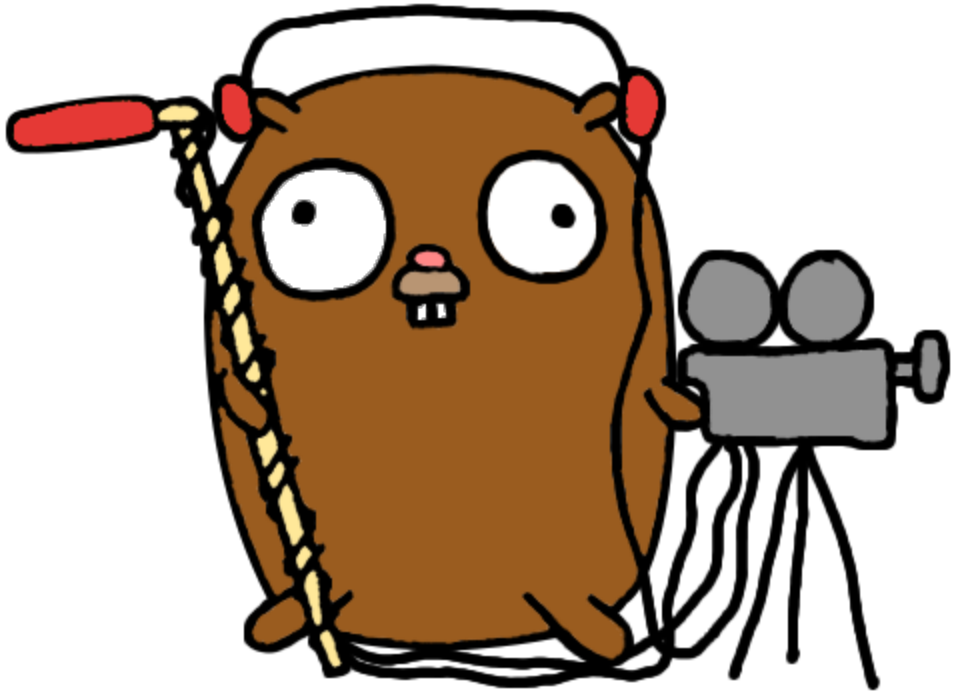
Implementacje WebRTC

LibWebRTC

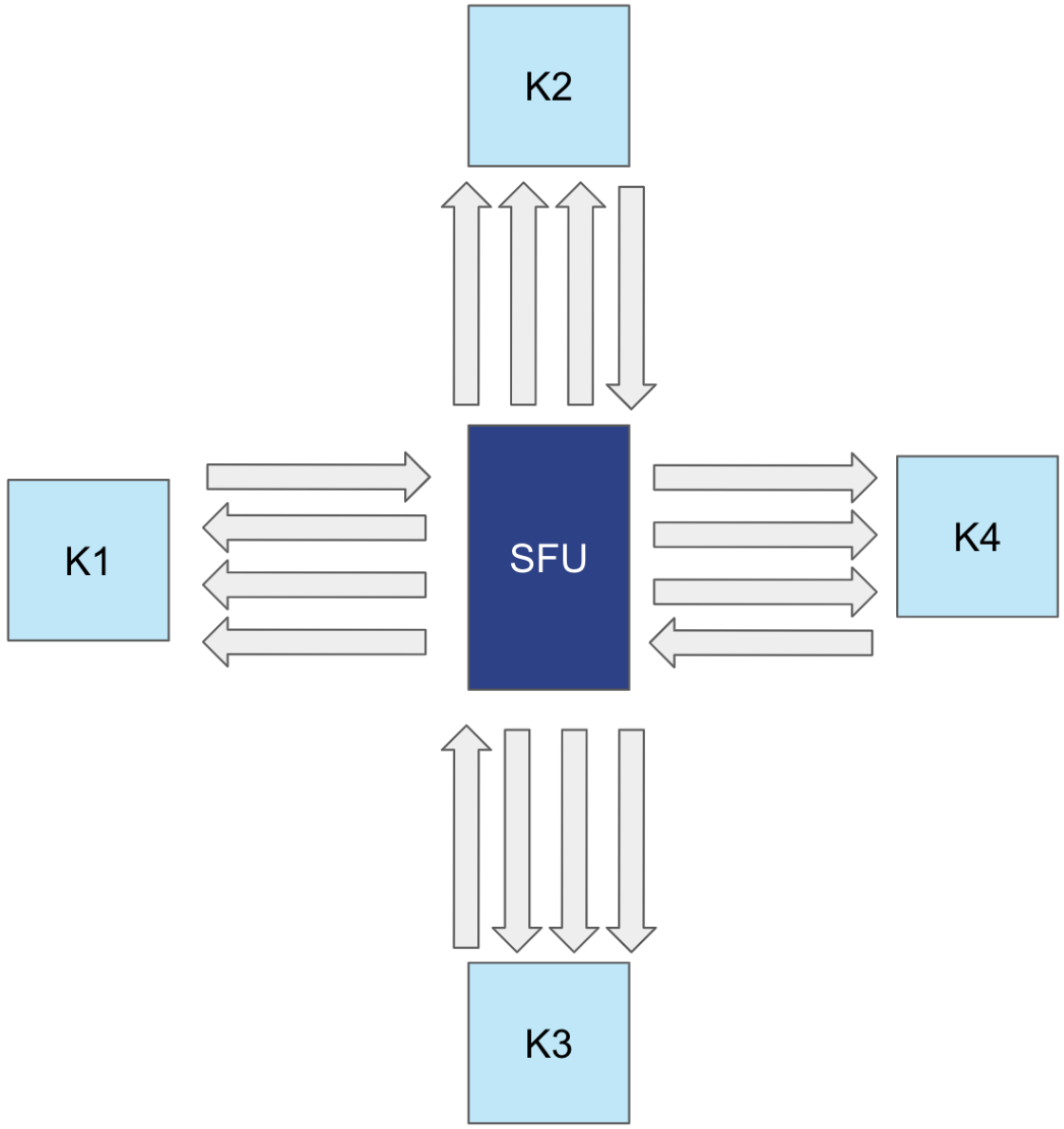
- Implementacja Google używana w Chromium
- Monolit w C++
- De facto standard
- Brak buildów

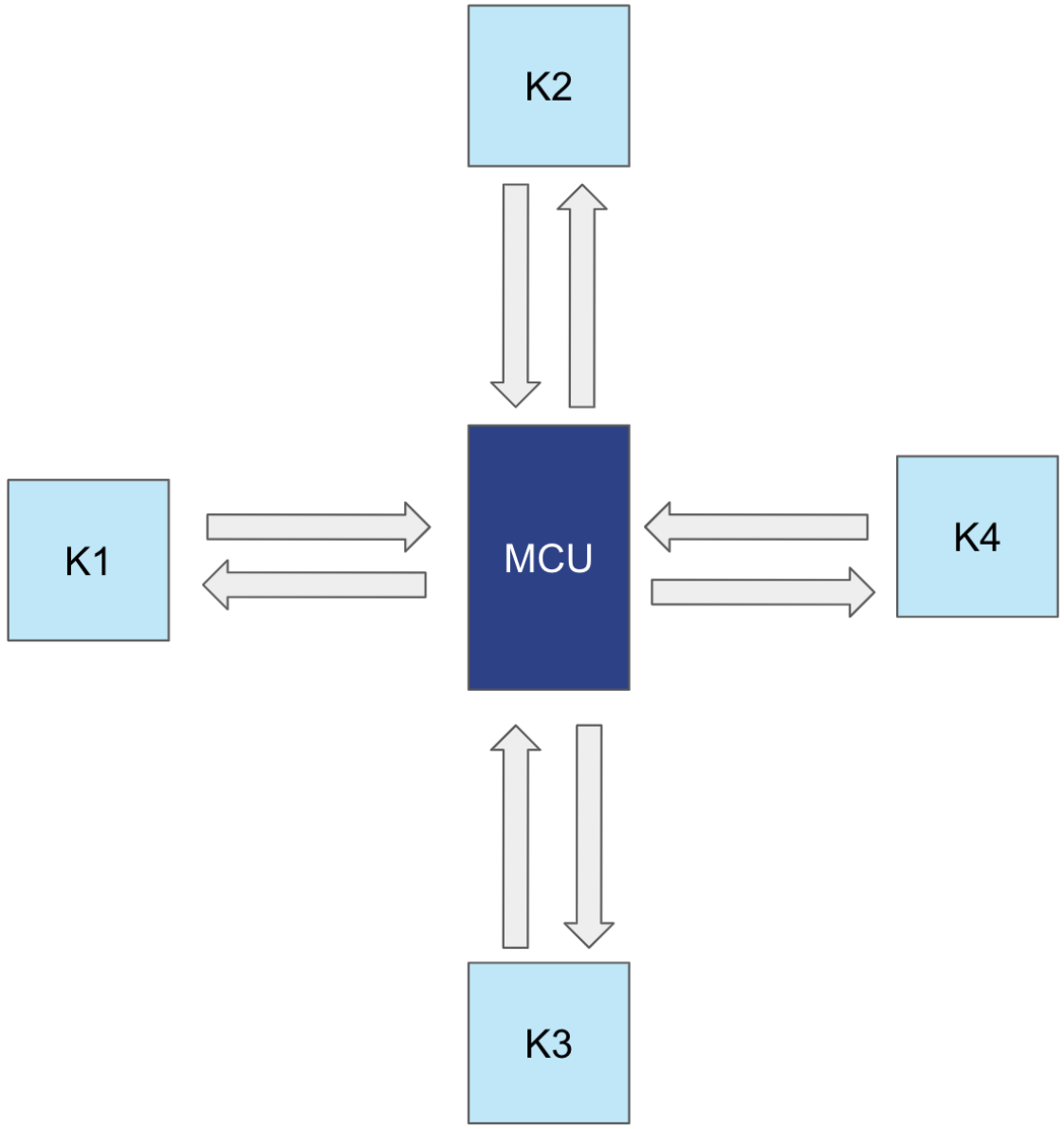
Pion

- W całości w Go
- Feature complete
- Czytelny kod
- Community
- github.com/pion/webrtc



Serwery


K2
K1
SFU
K4
K3


K2
K1
MCU
K4
K3

LiveKit

- Sensowne API
- Wiele SDK
- Optymalizacje E2E (demo)
- Distributed (cloud)
- Czytelny kod
- livekit.io

Jitsi

- W Javie
- W założeniu łatwy w użyciu
- Ograniczone możliwości konfiguracji/interakcji
- Jitsi meet
- jitsi.org

Janus

- W C
- Pierwsza popularna implementacja WebRTC
- Konfigurowalny
- Niska jakość kodu
- Niepraktyczne API
- GPL
- github.com/meetecho/janus-gateway

MediaSoup

- Node, C++
- Kiedyś rozbudowany media serwer
- Dosyć popularny

Jellyfish

- W Elixirze
- Integruje WebRTC z innymi protokołami
- Rozszerzalny
- Oparty o Membrane
- membrane.stream

Wydajność

Comparative Study of WebRTC Open Source SFUs for Video Conferencing

Emmanuel André*, Nicolas Le Breton*§, Augustin Lemesle*§, Ludovic Roux* and Alexandre Gouaillard*
*CoSMo Software, Singapore, Email: {emmanuel.andre, ludovic.roux, alex.gouaillard}@cosmosoftware.io
§CentraleSupélec, France, Email: {nicolas.lebreton, augustin.lemesle}@supelec.fr

***Abstract*—WebRTC capable media servers are ubiquitous, and among them, Selective Forwarding Units (SFU) seem to generate more and more interest, especially as a mandatory component of WebRTC 1.0 Simulcast. The two most represented use cases implemented using a WebRTC SFU are video conferencing and broadcasting. To date, there has not been any scientific comparative study of WebRTC SFUs. We propose here a new approach based on the KITE testing engine. We apply it to the comparative study of five main open-source WebRTC SFUs, used for video conferencing, under load. The results show that such approach is viable, and provide unexpected and refreshingly new insights on the scalability of those SFUs.**

***Index Terms*—WebRTC, Media Server, Load Testing, Real-Time Communications**

I. Introduction

servers, even from frameworks that claim to be media server and signalling agnostic.

In this paper, we will focus on scalability testing of a video conference use case using a single WebRTC SFU media server. The novelty here is the capacity to run exactly the same test scenario in the same conditions against several different media servers installed on the same instance type. To compare the performance of each SFU for this use case, we report the measurements of their bit rates and of their latency all along the test.

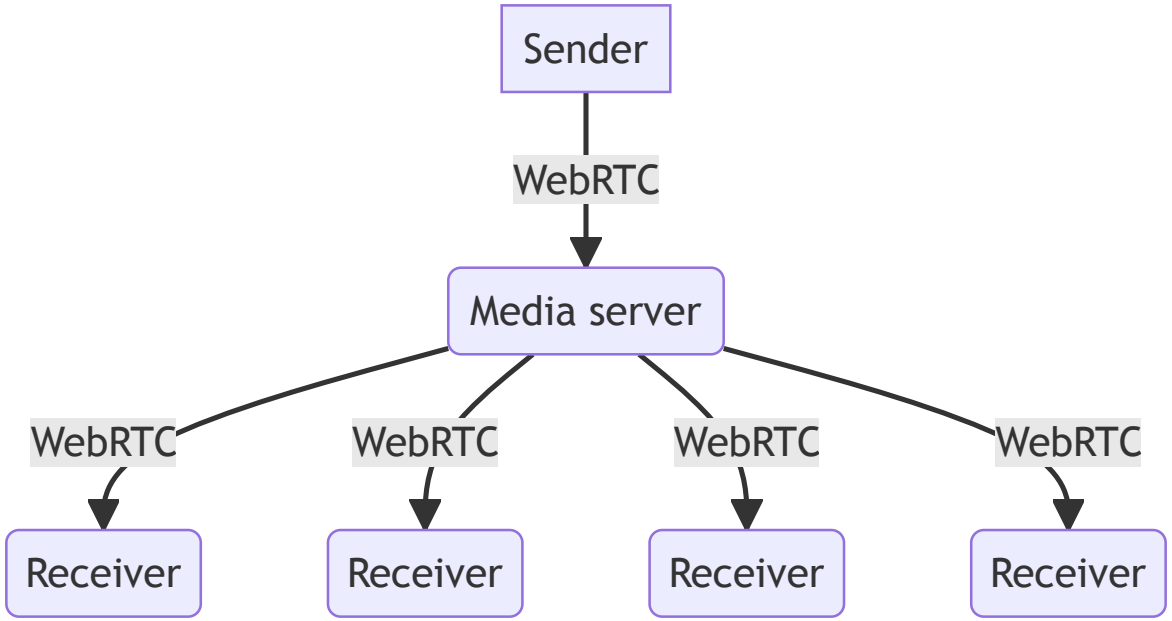
The rest of the paper is structured as follows: section II provides a quick overview of the state of the art of WebRTC testing. Section III describes, in detail, the configurations, metrics, tools and test logic that were used to generate the

https://mediasoup.org/resources/CoSMo_ComparativeStudyOfWebrtcOpenSourceSfusForVideoConferencing.pdf

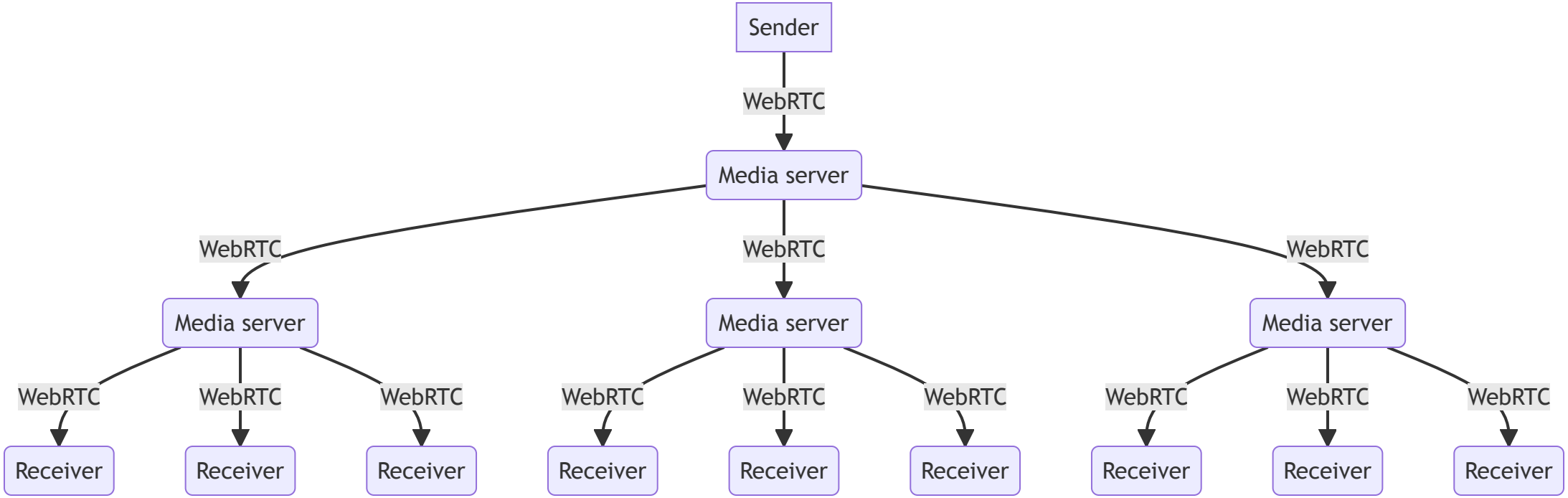
SaaS

- Mux
- Agora
- Livekit
- 8x8 (Jitsi)
- AWS Chime
- Cloudflare

Broadcasting



Broadcasting



Jakie są inne opcje?

Problemy WebRTC

- Wymusza niską latencję
- Wymaga użycia (zbyt?) wielu protokołów
- Wymaga użycia nieadekwatnych protokołów (SDP, JSEP)
- Bałagan w RTP extensions
- Nieustandaryzowany signaling
- Nieustandaryzowane kluczowe funkcjonalności (congestion control)
- libwebrtc jest de facto standardem

Low latency streaming

- ORTC
- QUIC

ORTC

- Określane WebRTC 1.1
- Nie wymusza SDP/JSEP
- ~2016 [*]

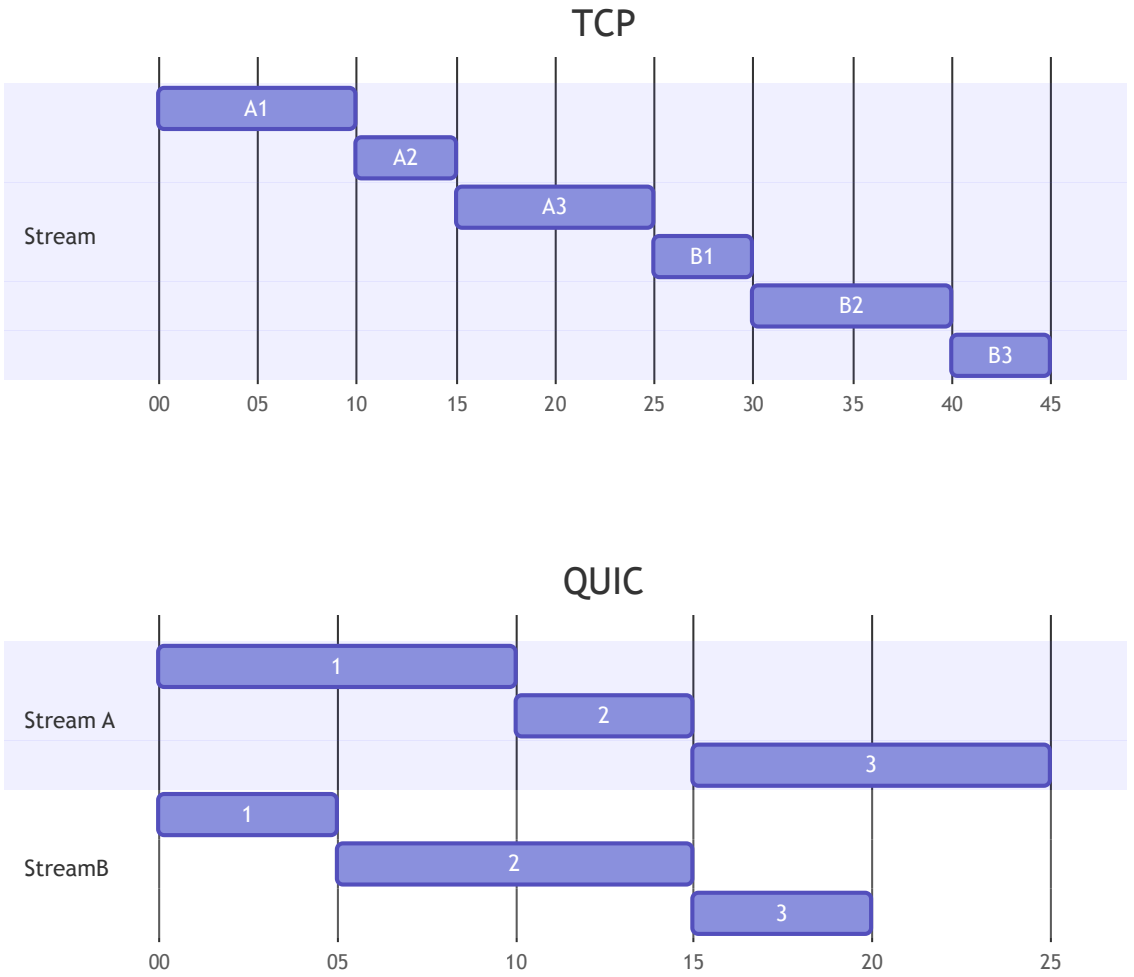
QUIC

- Warstwa IV, enkapsulowany w UDP
- Zaimplementowany w Chromium
- Działa na nim HTTP/3
- WebTransport
- Stream API i datagram API
- Konfigurowalna kontrola przeciążeń

QUIC Stream API

- Niezawodne
- Uporządkowane
- Podlega kontroli przeciążeń

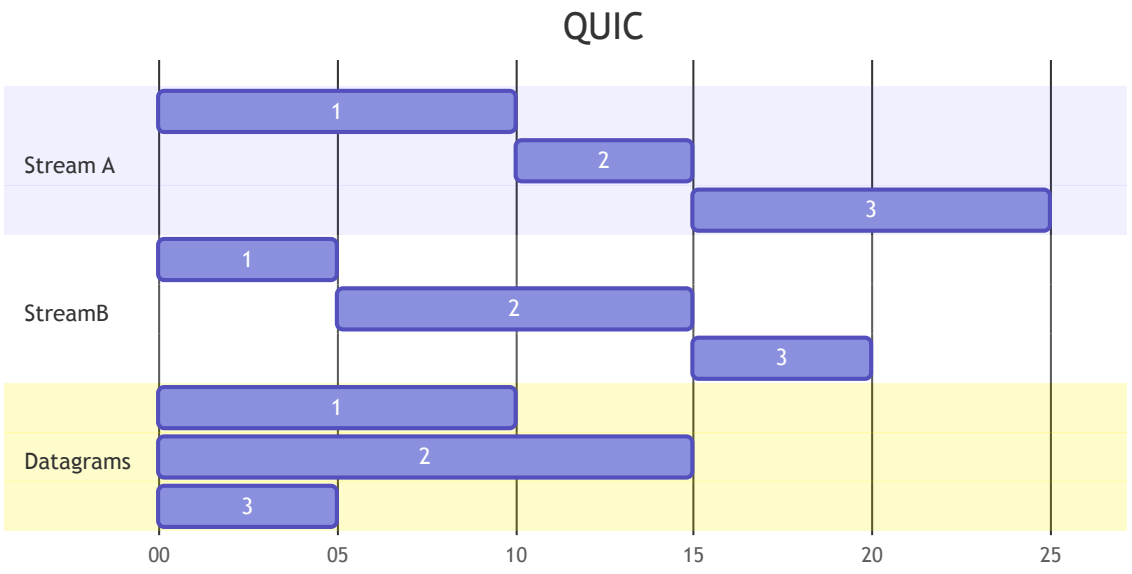
QUIC Stream API



QUIC Datagram API

- Zawodne
- Nieuporządkowane
- Podlega kontroli przeciążeń

QUIC



Media over QUIC

- Uniwersalny protokół
- Problemy z kontrolą przeciążeń
- Problem ze sterowaniem enkoderem - WebCodecs

Low latency streaming

- ni ma ¯_(ツ)_/¯

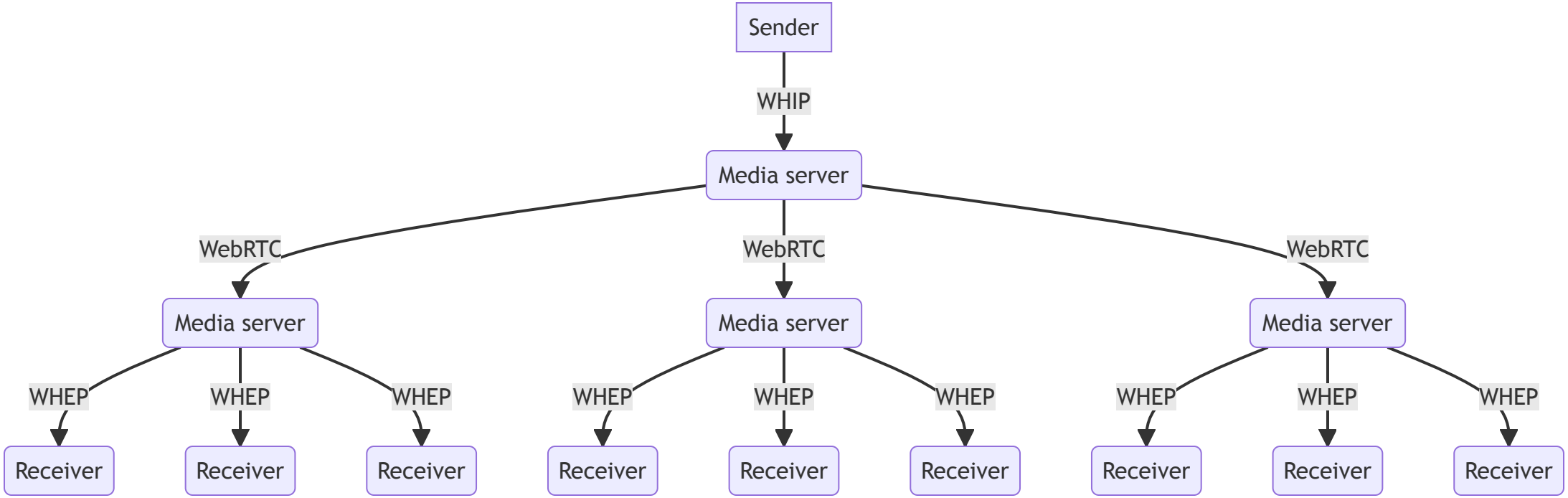
Broadcasting

- WHIP & WHEP
- RTMP & HLS
- RUSH & WARP

WHIP & WHEP

- Standaryzują signaling dla WebRTC
- WHIP ingres, WHEP egres
- Komunikacja między serwerami
- Mało elastyczne

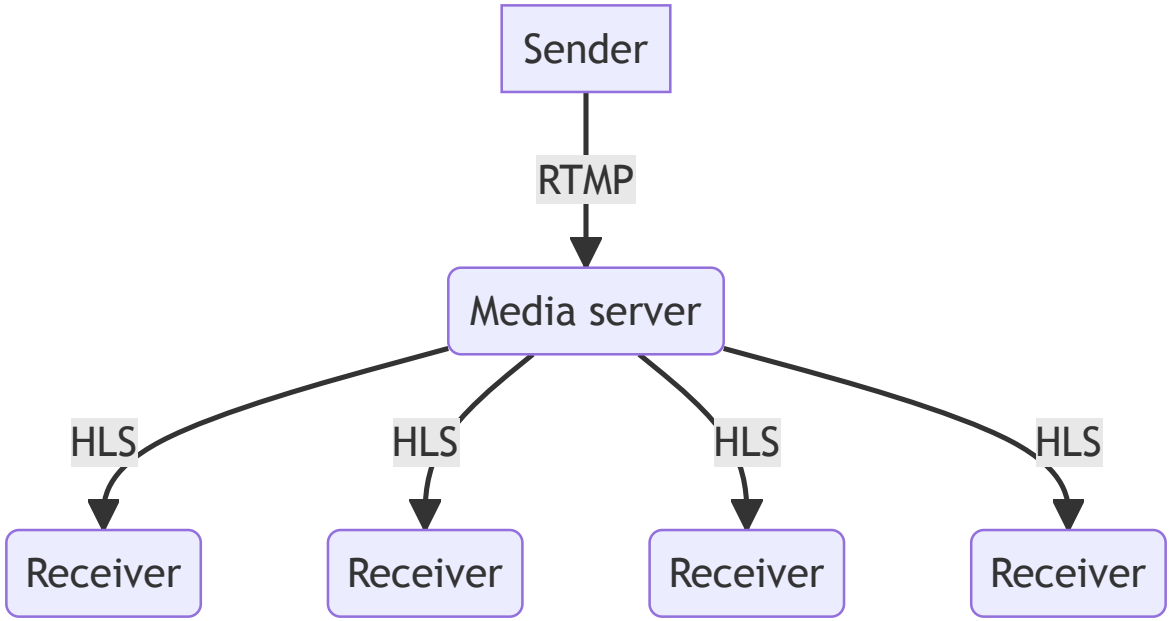
Broadcasting



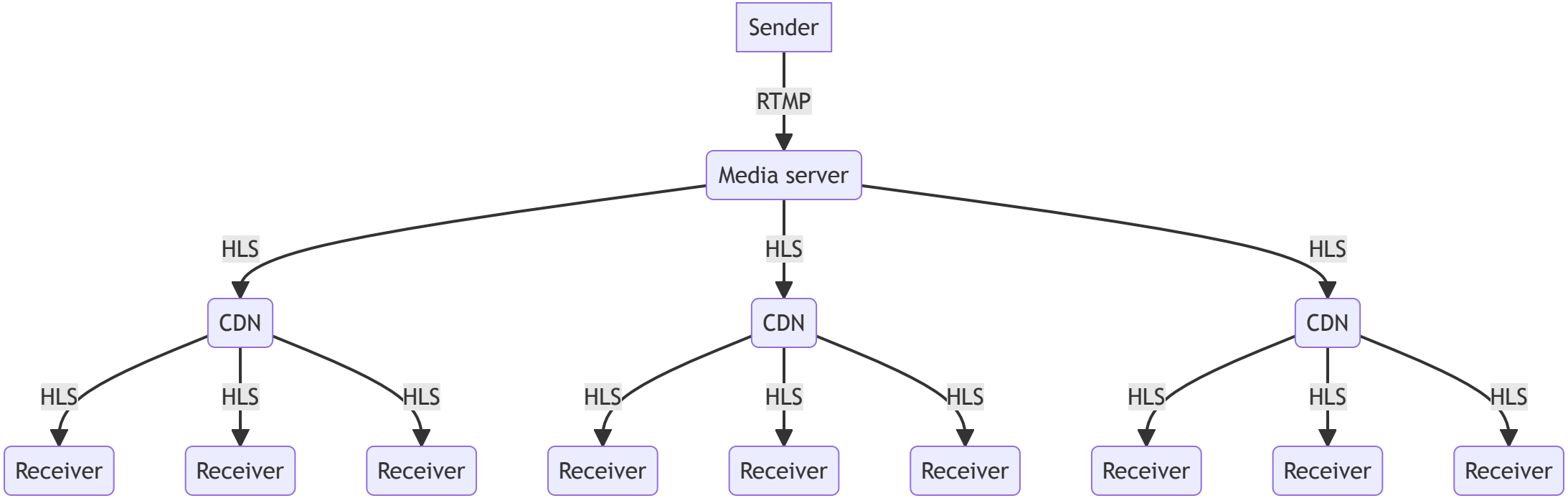
RTMP & HLS

- Standardowy stack
- Działa na TCP
- RTMP
  - popularny, ale porzucony
  - jest
  - SRT & RIST
- HLS
  - dzieli stream na krótkie fragmenty
  - zapisuje je do plików i udostępnia po HTTP

Broadcasting



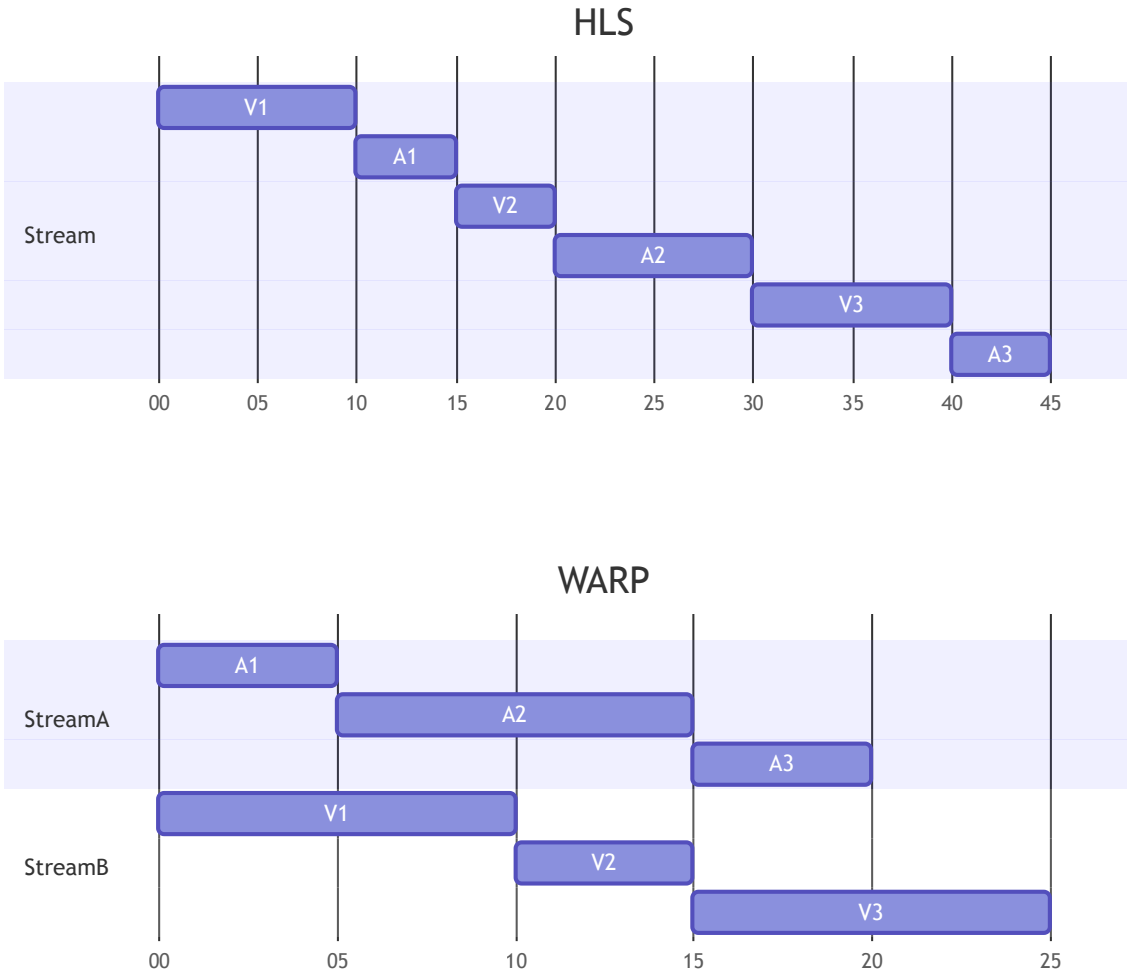
Broadcasting



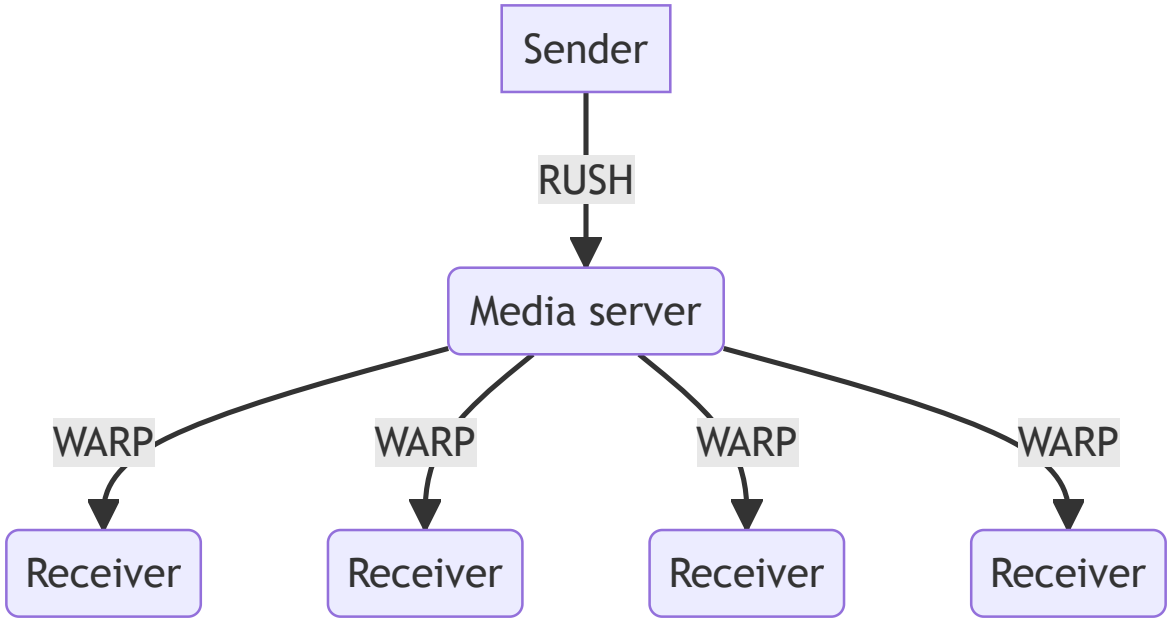
RUSH & WARP

- Problemy z jakością WebRTC
- Oparte o QUIC streams
- RUSH - Meta, WARP - Twitch

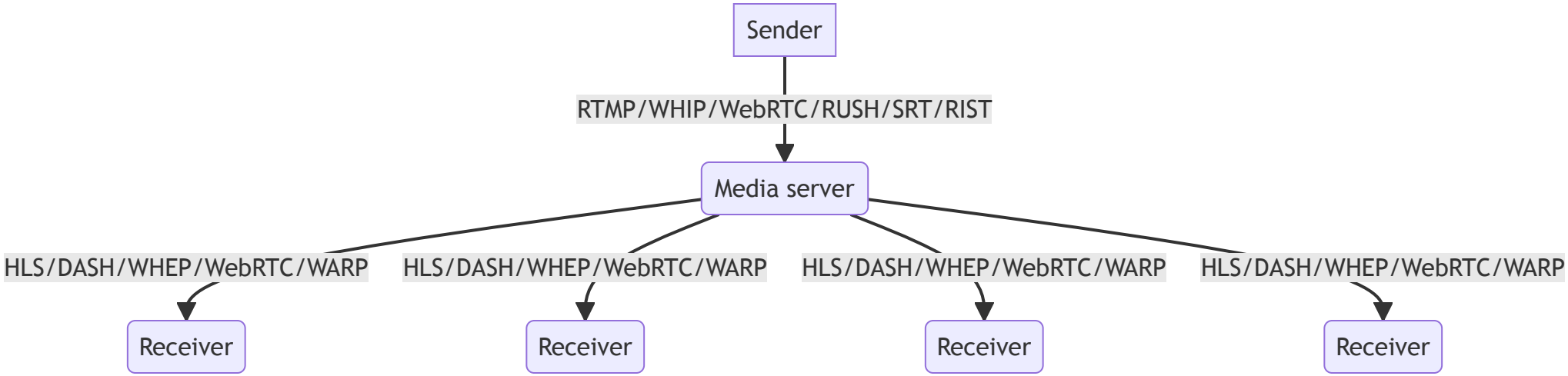
RUSH & WARP


HLS
Stream
V1
A1
V2
A2
V3
A3
00
05
10
15
20
25
30
35
40
45
WARP
StreamA
A1
A2
A3
StreamB
V1
V2
V3
00
05
10
15
20
25

Broadcasting



Broadcasting



Jak to jest pracować w multimediach?

Jak to jest pracować w multimediach?

internship.swmansion.com

Pytania?

Linki

- https://webrtc.github.io/samples/
- https://www.chromium.org/quic/
- https://atscaleconference.com/videos/live-media-over-quic/
- https://bloggeek.me/whip-whep-webrtc-live-streaming/
- https://developer.mozilla.org/en-US/docs/Web/API/WebCodecs_API
- https://github.com/pion/webrtc
- https://webrtc.googlesource.com/src

Linki c.d.

- https://livekit.io
- https://meet.jit.si
- https://jitsi.org
- https://github.com/meetecho/janus-gateway
- https://mediasoup.org
- https://membrane.stream
- https://mediasoup.org/resources/CoSMo_ComparativeStudyOfWebrtcOpenSourceSfusForVideoConferencing.pdf
- https://ortc.org/