|  |  |
| --- | --- |
| Không có mô tả. | **Personal Cloud Computing (first stage)** |
| (v2.0) June 3rd, 2021 | **Project Engineering Documentation** |

# General description

## 1.1 General information

Project leader: Do Huy Hoang  
Email: [huyhoangdo0205@gmail.com](mailto:huyhoangdo0205@gmail.com)

Member:

* Cao Gia Bao
* Vu Truong Giang
* Le Van Thien

[GitHub](https://github.com/pigeatgarlic/personal-cloud-computing/tree/master)

## 1.2 Document structure

Document structures determine how the technical solution will be present in the document. The document structure will be divided into four main layers, each layer will represent one abstraction level of the system: the higher layer it is, the more it related to user, and the lower it is, the more it is related to operation manager.

**Design language:**  How user perceive the platform

[**Architecture**:](#_Communication_layer) Function of each device in system and their connection

**Service**: Distinctive service (both business and engineer) and their principle in term of call procedure.

**Project directory structure and toolset:**  like its name

**Appendix:**

1. Terminology
2. Resources

# Design language

* Design language of Thinkmay - personal cloud computing in the first stage will be copied from Google stadia – a cloud gaming platform.
* The focus of personal cloud computing will be the simplicity.

## Main page

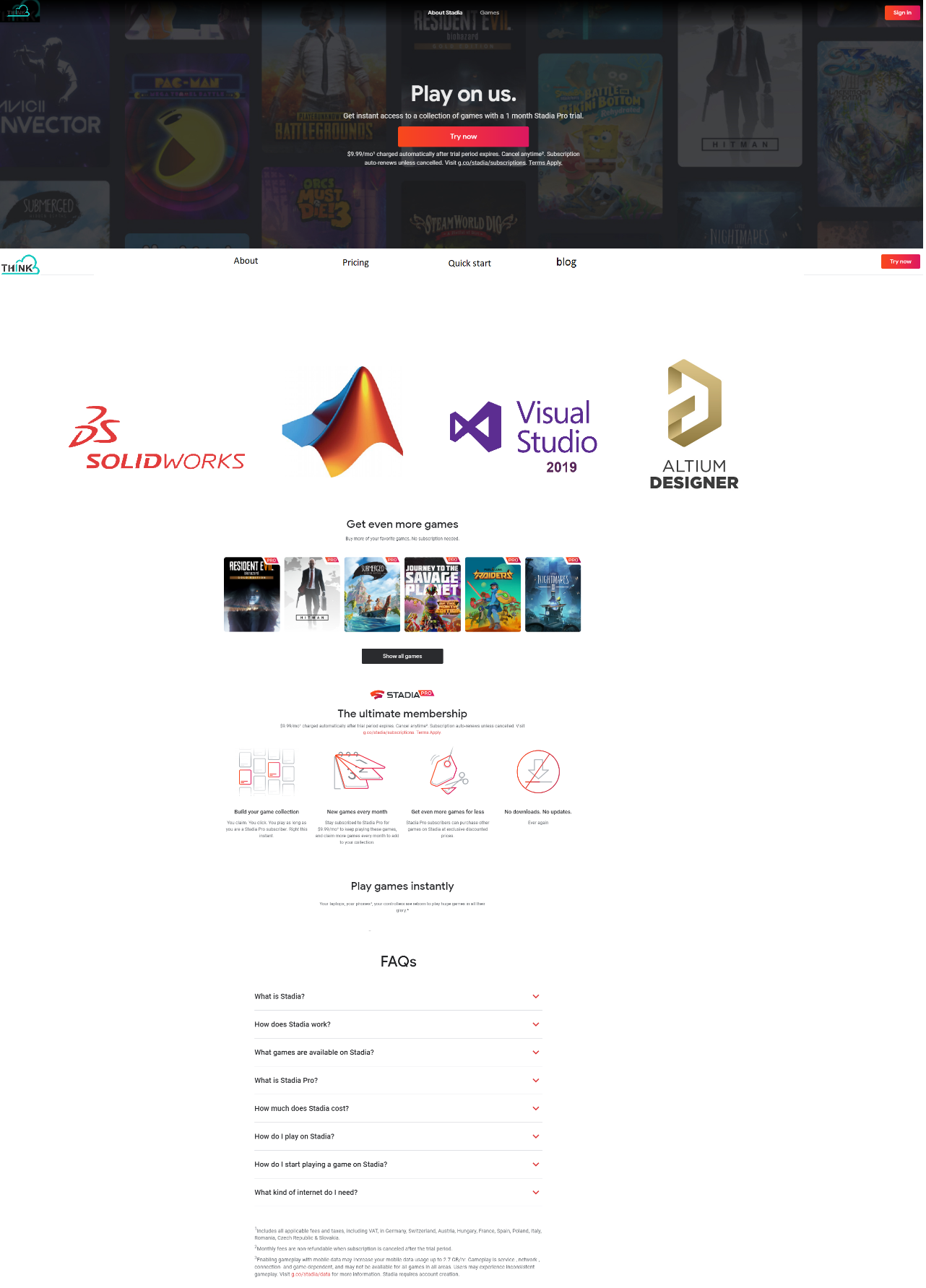


Figure 1 First sight of the web app

After signing in into Thinkmay personal cloud computing website, this is the first thing users will see.

## Service page

The page where users are ready to create session and check available slave device.

## In Session

When user click the remote desktop button, a session will be initialized.

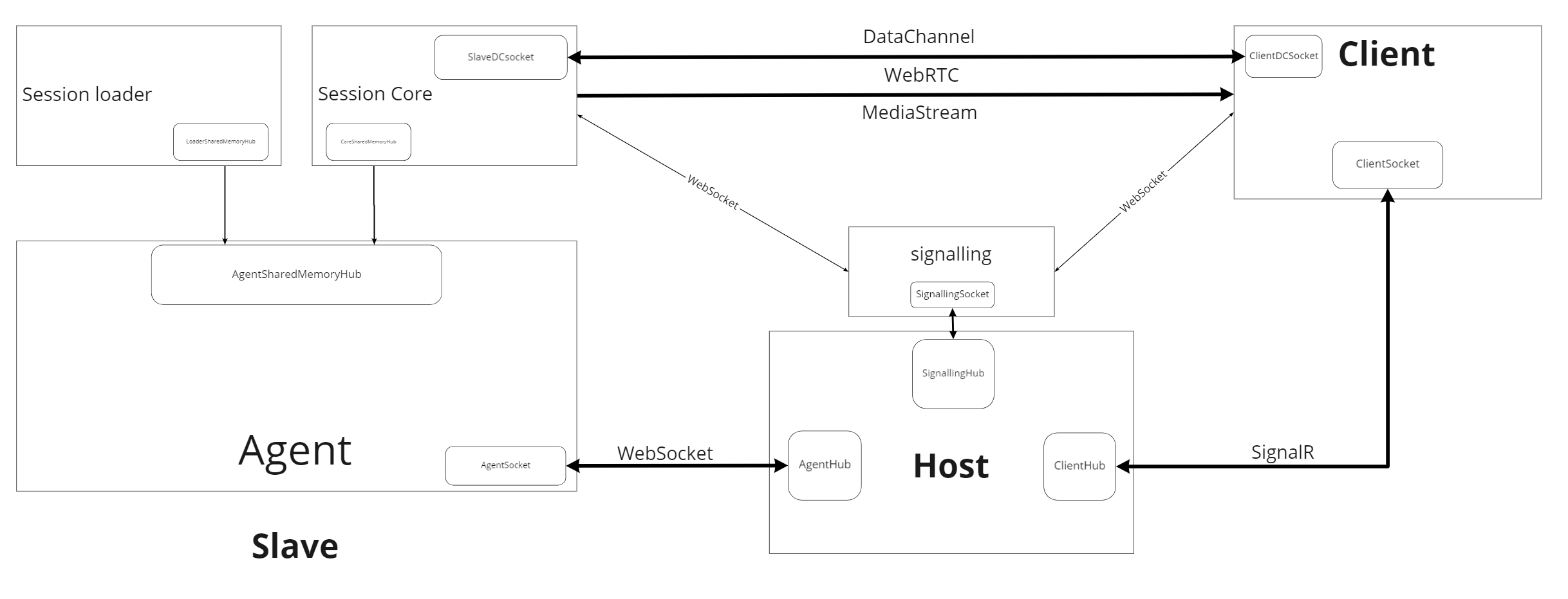


While maintaining the session, user will have option to customize the QoS (quality of service) of the session: resolution optimized, latency optimized, framerate optimized or automatically optimized.

If user extend the session screen into full-size mode, the output resolution will automatically be updated to fit the user screen resolution, the monitor bar will be hided and only can turned back by a predefined key combination.

# Software architecture

Function of each device in system and their connection



## Client

(Browser) - The device uses use to create session and remote the slave devices.

Client serves three main functions:

Stream handling: responsible for receiving the streaming packet from master, decode and display the stream on client’s screen. Display pointer as an independent object.

Message handler: responsible for transport user data to slave included:

* Clipboard
* File Transfer
* HID: mouse, keyboard

Session management: **Manage session included:**

* **Session establishment**
* **Close/reconnect remote control.**
* **Manage QoE (quality of experience): stream resolution, bitrate, latency….**

Storage management: Communicate with host to monitor the personal storage, when user request file transfer, receive http file transfer from master device. (This function will be implemented later when remote control function is finished)

## Slave device

Slave devices bypass the computational heavy workload from user.

There are three distinctive programs responsible for slave management: Agent, Session loader, Session Core.

### Agent

Agent is a program that always run in the background of slave device to keep the connection between Slave and Host.

* Agent is a simple program written on C# and don’t have a User Interface. During the connection with host, Agent keep the slave information service, Update management (will be implemented later) alive.
* When a session request is come from host, Agent responsible for initializing the Session loader which initialize the remote connection.
* AgentPipeSrc is sub block inside Agent program responsible for communication with Session loader using named pipe.
* AgentSocket is a subblock inside Agent program responsible for communication with Host using WebSocket (SignalR)

### Session loader

Session loader is a program that run during the user’s session and responsible for initializing session core and management of user messages.

* Session loader is initialized by agent using command line parsing and responsible for initializing session using the same method.
* LoaderPipeSink is a subblock inside Session Loader program responsible for communication with Agent using Named pipe.
* LoaderPipeSrc is a subblock inside Session Loader program responsible for communication with Session Core using Named pipe.
* Session loader is written in C# and have GUI in form of XAML application.
* Some additional service of Session loader included clipboard sender, Download – Upload file, Input handling.

### Session Core

Session Core responsible for the only function is managing user’s remote control and message transfer using WebRTC protocol.

* Session Core is initialized by Session Loader using command line.
* Session Core is written in C++ based on Gstreamer framework (which based on Glib), some other dependencies of this program are Json-glib, Libsoup (which all depend on Glib).
* Session Core should keep as compact as possible, the purpose of session core is to enable hardware acceleration (which GStreamer support).
* Session Core responsible for data transfer between Slave and Client using WebRTCDataChannel.
* All data related to Session Loader’s service is come from on\_message\_data and transferred directly through CorePipeSink.
* All data related to Session Core’s service is come from on\_message\_string be processed locally.

## Host

Host responsible for orchestrating all session request from all client/slave devices.

* Host (Server in regular term) responsible for all client request outside the session included website hosting, User database.
* Beside basic function of a server, host also has a subblock called signaling responsible for establishing webrtc connection between Client and Slave.

### Signaling

Signaling block responsible for registering/deregistering session and establish remote control.

* Signalling block hold a session keypair ID for each session, one is on client’s side, one is on slave’s side. During WebRTC handshake, if sessionID from client and sessionID from Slave is in one pair. The connection will be established.
* Because of that, Session active time will be counted from the creation to the termination of ID pair.
* Signalling will be written on python. And deployed as a separate module.

Connection Hub

Connection hub responsible for maintaining message transfer with each device in system.

* There are 3 connection hubs corresponding to three connection sockets in the system.

+ Signalling hub: maintain WebSocket connection with signalling service.

+ Agent hub: maintain WebSocket connection with Agent.

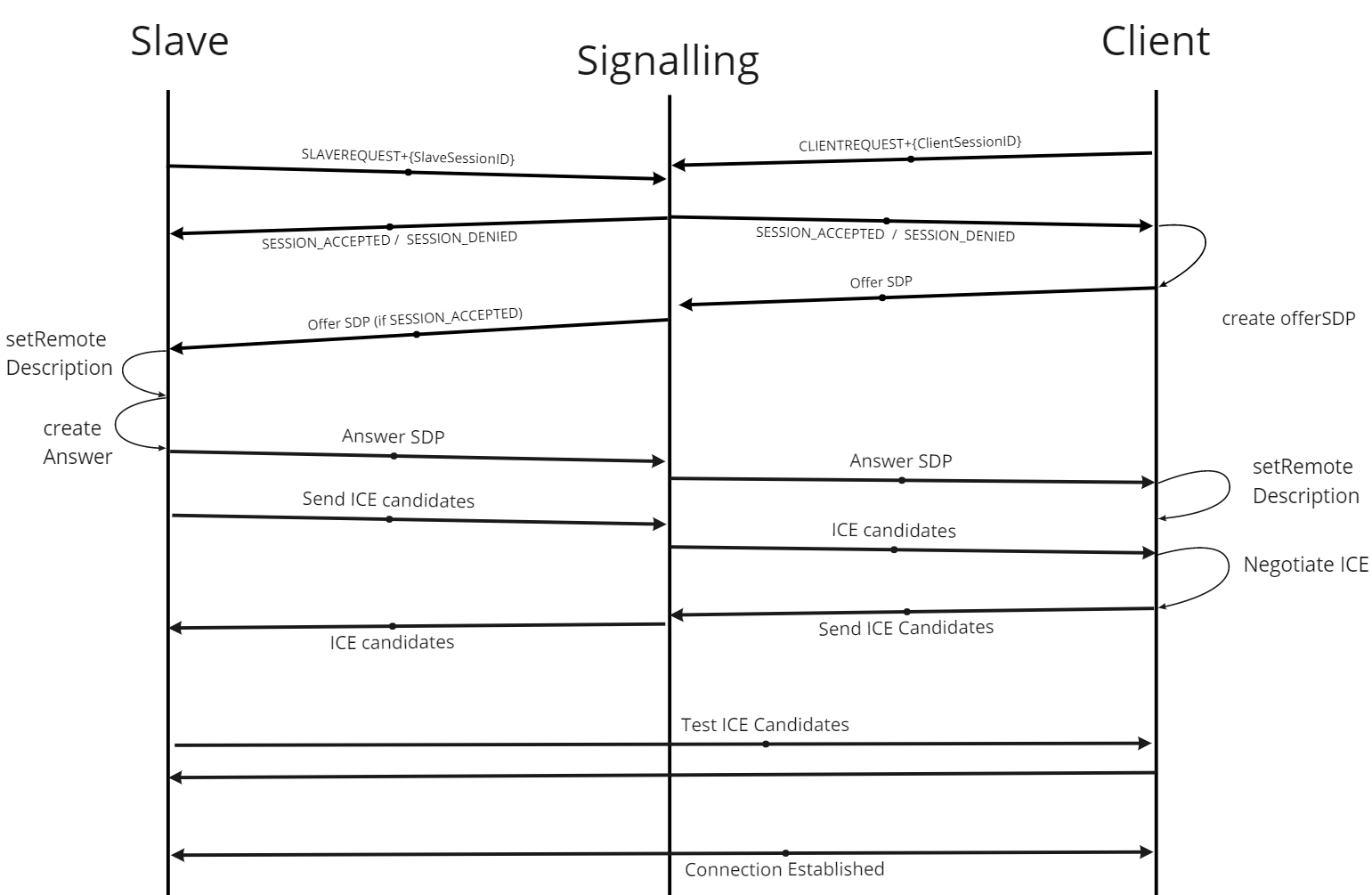
+ Client hub: maintain WebSocket connection with Client (browser).

# **Service**

Distinctive service and their principle in term of call procedure.

## **Establish new session.**

Session establishment is based on WebRTC connection handshake establishment in this [document](https://cloud.google.com/architecture/gpu-accelerated-streaming-using-webrtc).



**Datagram Transport Layer Security (DTLS)**

An implementation of the Transport Layer Security specification that can be used over User Datagram Protocol (UDP). WebRTC requires all data to be encrypted in transit and uses DTLS to secure data transmission.

**Interactive Connectivity Establishment (ICE)**

Method used by WebRTC to discover the optimal way to create a peer-to-peer connection. Peers exchange ICE candidates that are negotiated and prioritized until a common connection method is agreed upon.

**RTCPeerConnection**

The JavaScript object used to create a WebRTC connection. The [WebRTC adapter JavaScript](https://bloggeek.me/webrtc-adapter-js/) source provides a standard interface so that you don't have to create custom code for each browser-specific implementation.

**Session Description Protocol (SDP)**

Media and connection configuration and capabilities exchanged by peers during connection establishment.

**Session Traversal Utilities for NAT (STUN)**

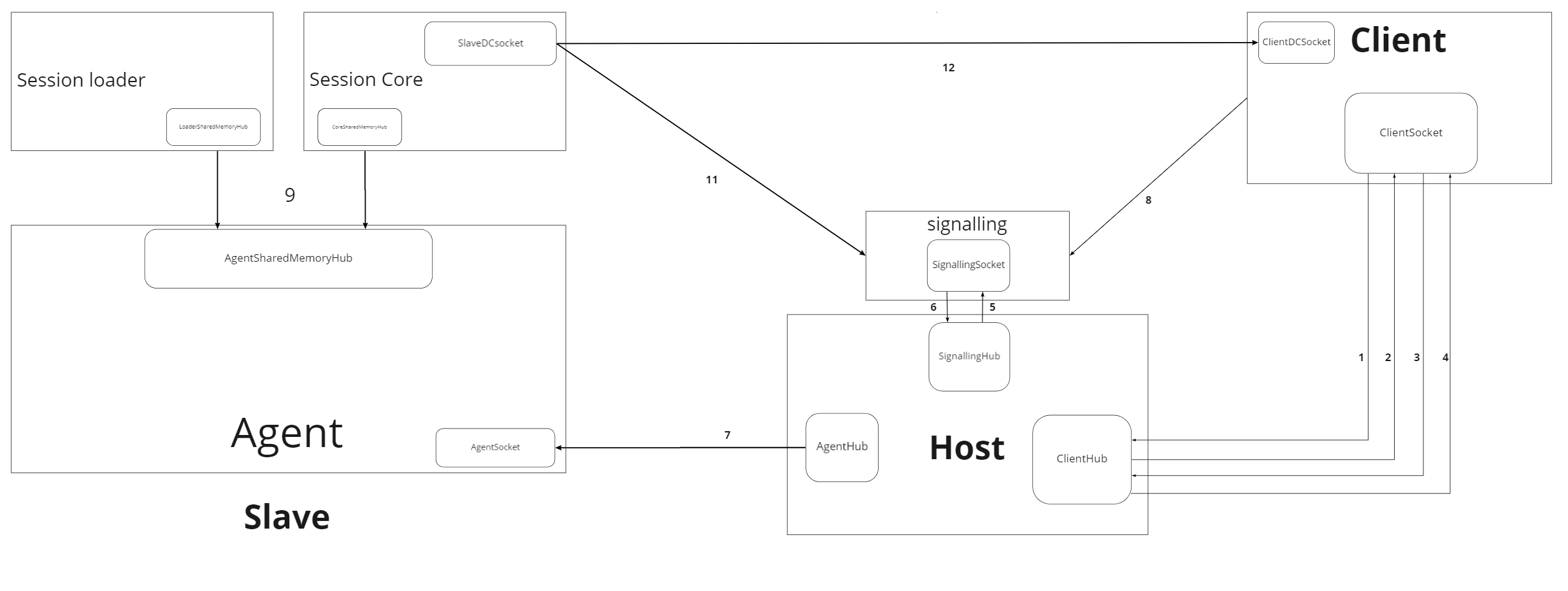
An external service used by peers to discover their real external IP address if they are behind a firewall or NAT gateway.

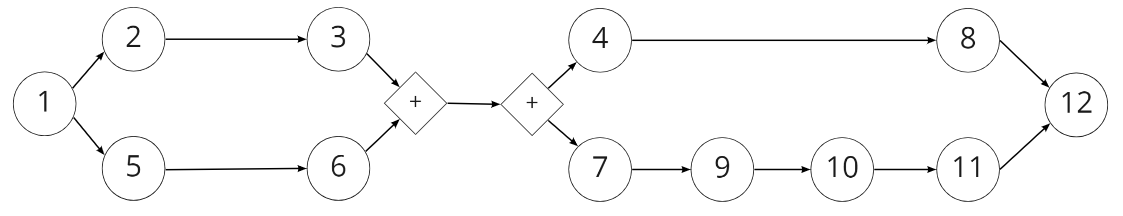
**signaling**

An external service used by peer connections that is not included in the WebRTC specification but is required for connection establishment. Although there is no formal specification for signaling, it is common to use a [WebSocket](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API) or [Extensible Messaging and Presence Protocol (XMPP).](https://xmpp.org/uses/webrtc.html)

**Traversal Using Relays around NAT (TURN)**

An external service used as a relay by peers if no direct peer-to-peer connection method can be discovered during ICE candidate negotiation.

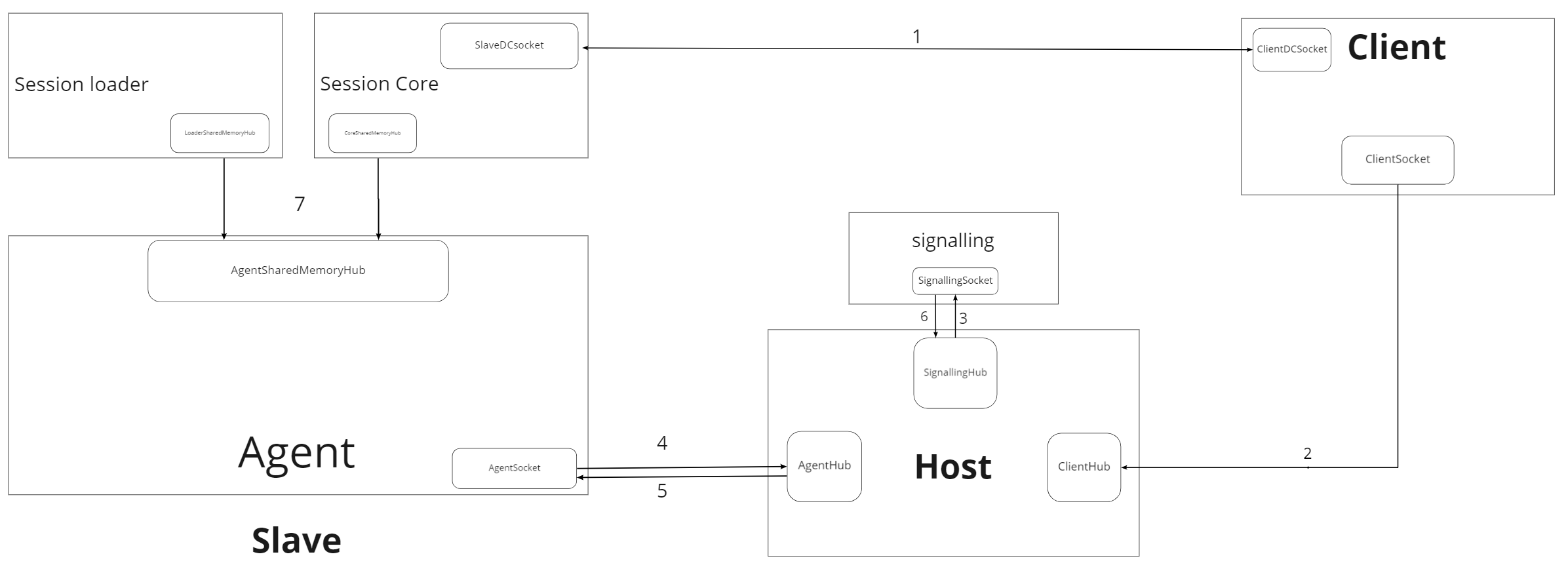




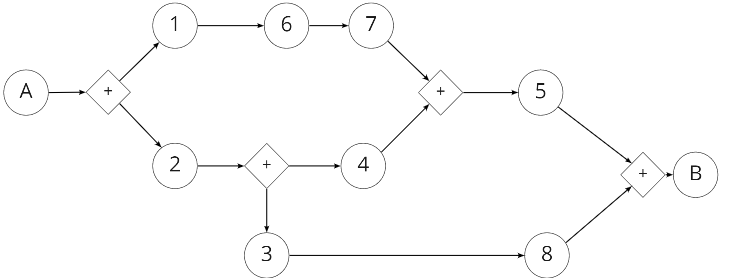
1. Client provides session request.
2. Host generate arbitrary ID pair sends back SessionClientID
3. Client confirmed SessionClientID
4. Host allow Client to implement the handshake.
5. Host sends SessionID pair to signalling server.
6. Signalling server confirmed the SessionID pair.
7. Host sends SessionSlaveID to slave.
8. Client registers to signalling server.
9. Agent initializes Session loader.
10. Session loader initializes session core.
11. Session core implements the handshake procedure.
12. WebRTC session established.

## **Session termination**

Session termination – the function is as simple as its name.



* Session termination process includes two processes: close remote control and terminating session ID pair in signalling server.

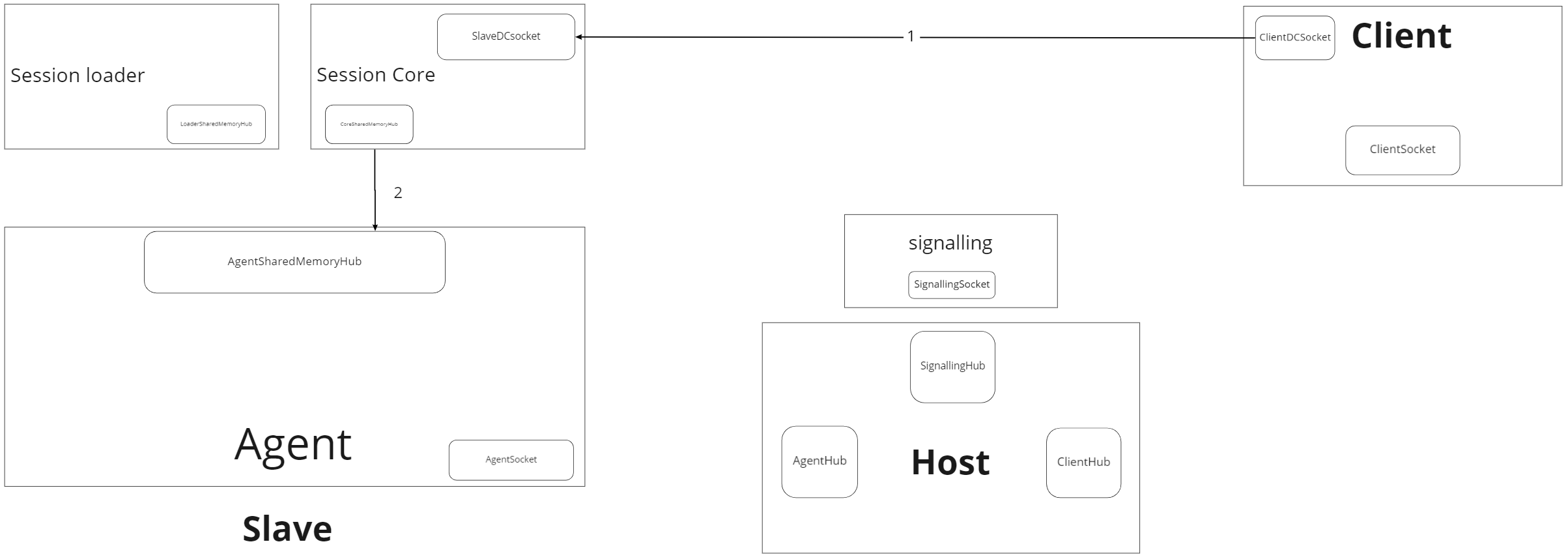


1. Client sends close remote-control signal over DataChannel.
2. Client sends close session signal to Host.
3. Host sends terminate SessionID pair signal to signalling server.
4. Host sends end session signal to Agent.
5. Agent confirmed session close.
6. Session Core sends remote control closed signal to Session loader.
7. Session Loader sends remote control closed signal to Agent.
8. Signalling server confirm SessionID pair closed.

A. User click Session terminate signal.

B. Host confirmed session closed.

## Close Remote Control (without closing session)

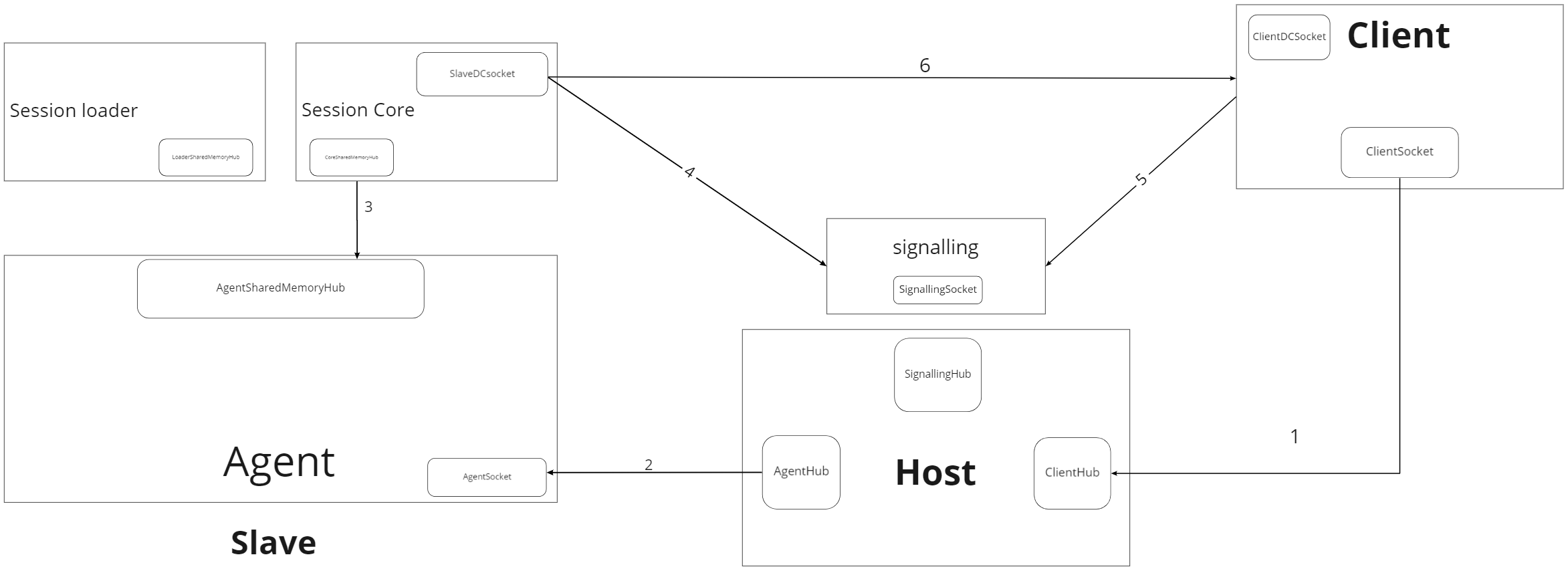
Close remote control while session ID pair still be saved in signalling server.

1. Data\_channel\_on\_message\_close signal
2. Session core termination signal send to session loader.

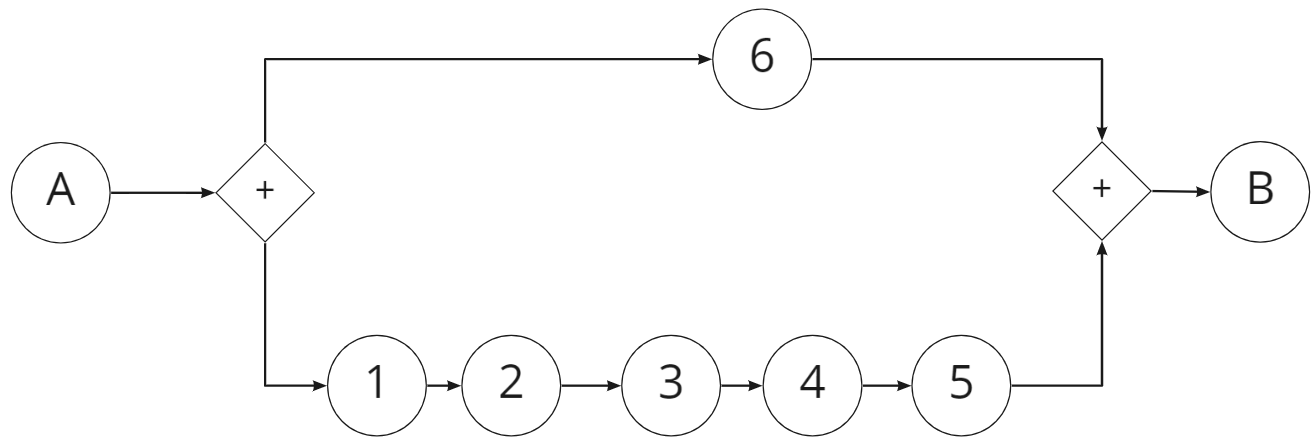
Closing remote control without closing session is only involve sending on\_close signal to slave devices without terminating session ID pair in signalling server.

## **Reconnect Remote Control**

Reconnect remote control after close remote control during a session.



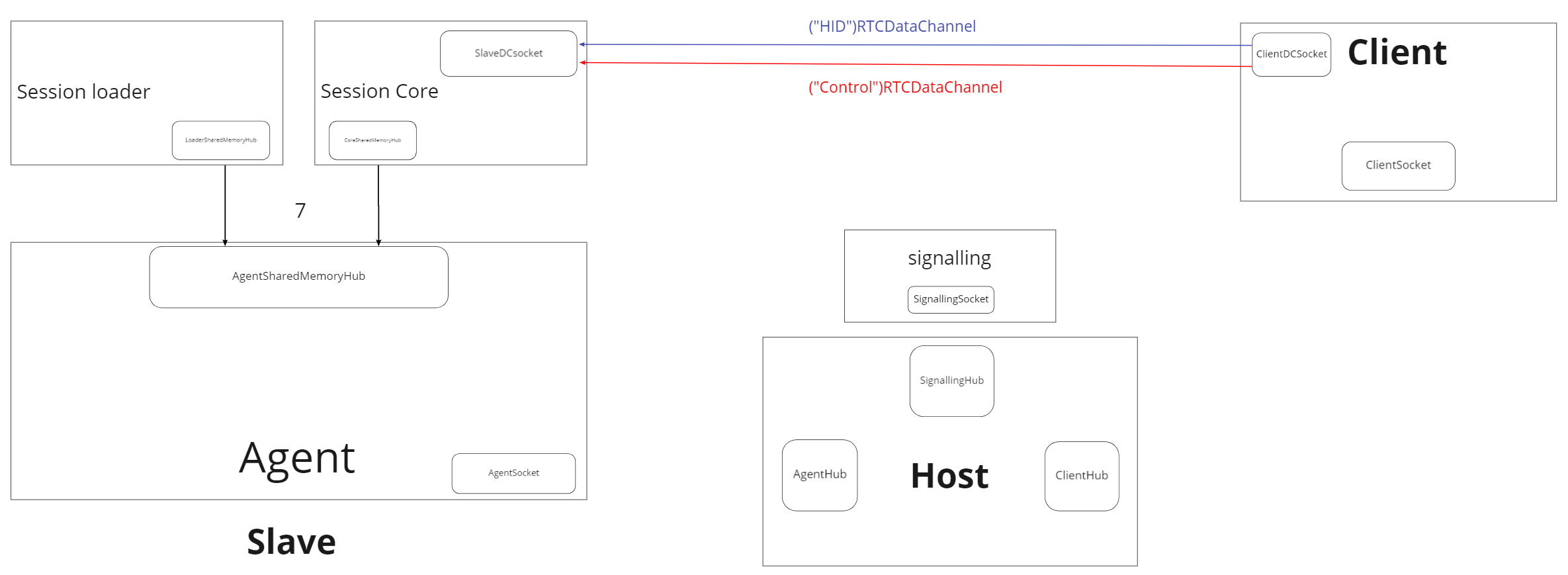
* Reconnecting remote control after closing remote control during remote session is involve sending reconnection signal to host without host sending session ID pair to signalling server, then session loader will initialize session core to process WebRTC handshake.



1. Clients send reconnecting signal to host.
2. Host send reconnecting signal to Agent.
3. Agent sends reconnecting signal to Session Loader.
4. Session loader initializes session core.
5. Slave process WebRTC handshake with signalling server.
6. Client process WebRTC handshake with signalling server.
7. WebRTC connection established.

## **Client message transfer**

Direct message transfer between client and Slave using WebRTCDataChannel.

￼

Data transfer between client and slave will be done through WebRTCDataChannel, information about this standard will be included [here.](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Using_data_channels)

Data transfer between Client and Slave will be divide into two categories:

+ Session Core control data managing information related to WebRTC stream and will be hidden from user. This data category will be passed through data\_channel\_on\_message\_string()

+ Session Loader service data managing data related to remote control’s services: HID input sends, clipboard service... This data categories will be passed through data\_channel\_on\_message\_data()

# Project directory structure and toolset

Agreement on used toolset and project directory's structure.

* Each block listed in Architecture correspond to one sub-project in project working directory.
* The naming convention for function in project is provided [here.](https://vinuniversity-my.sharepoint.com/:x:/g/personal/20hoang_dh_vinuni_edu_vn/ETiPNCbU4_tEp_61fEWOyV0B-pBExYJheWK13SI2gNz1cw?e=eVJ68W)

**Session Core**

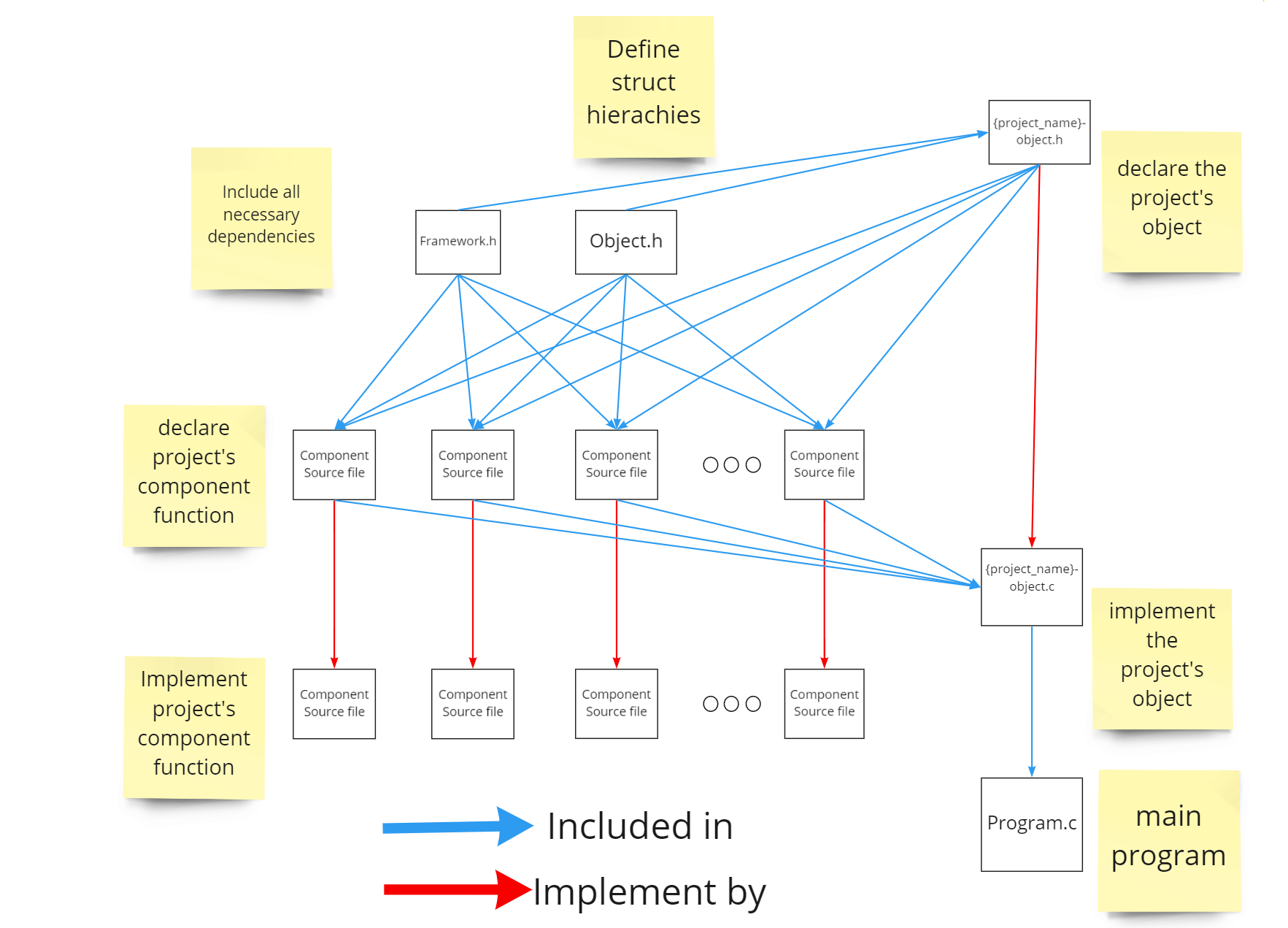
* Contain source code of Session Core program, based on Gstreamer.

## Agent

* Contain source code of Agent program, written in C.
* Signalling Server
* Contain source code of Signalling Server

## **Session Loader**

Contain source code of Session Loader.



Project structure implementation in SessionCore, SessionLoader and Agent.

.

## Signalling Server

Contain source code of Signalling Server

### Ground up

* Reference source code for signalling server are provided in form of python project packed in docker container, link to the project can be found [here.](https://github.com/centricular/gstwebrtc-demos/blob/master/signalling/simple_server.py)
* Developer in charge of signalling server will have three options of deploying signalling server:

+ Modify source python code, add session pair and time management service.

+ Write signalling server using ASP.NET core and dockerized.

+ **Write signalling server using ASP.NET core without dockerized.**

* Signalling server will communicate with SessionCore and Client using WebSocket protocol.

### Service

In short, signalling server will have following service:

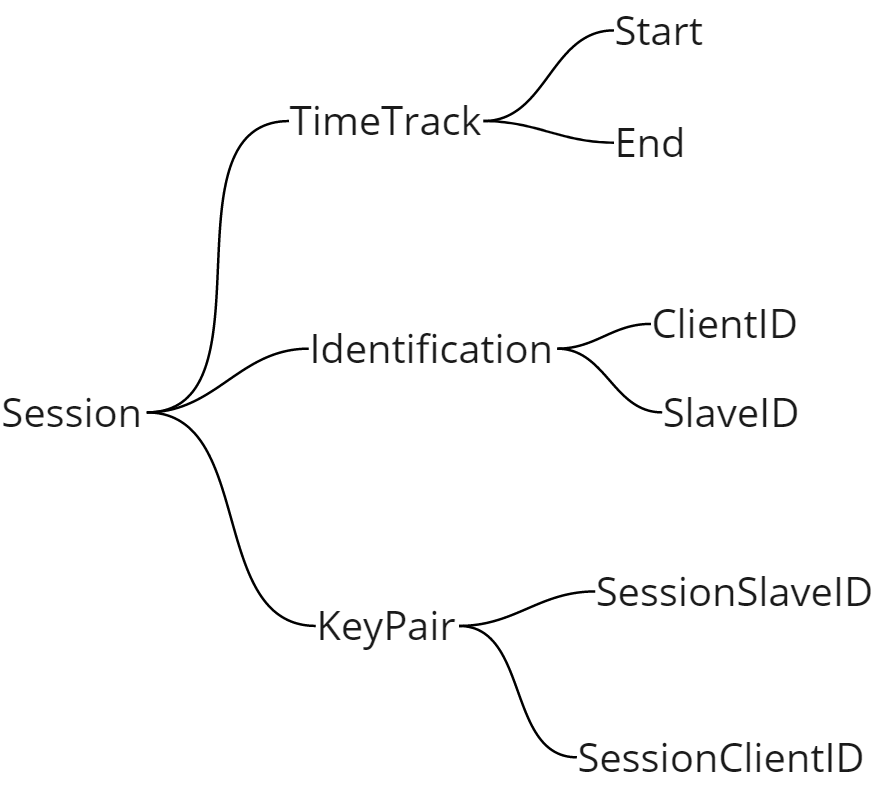
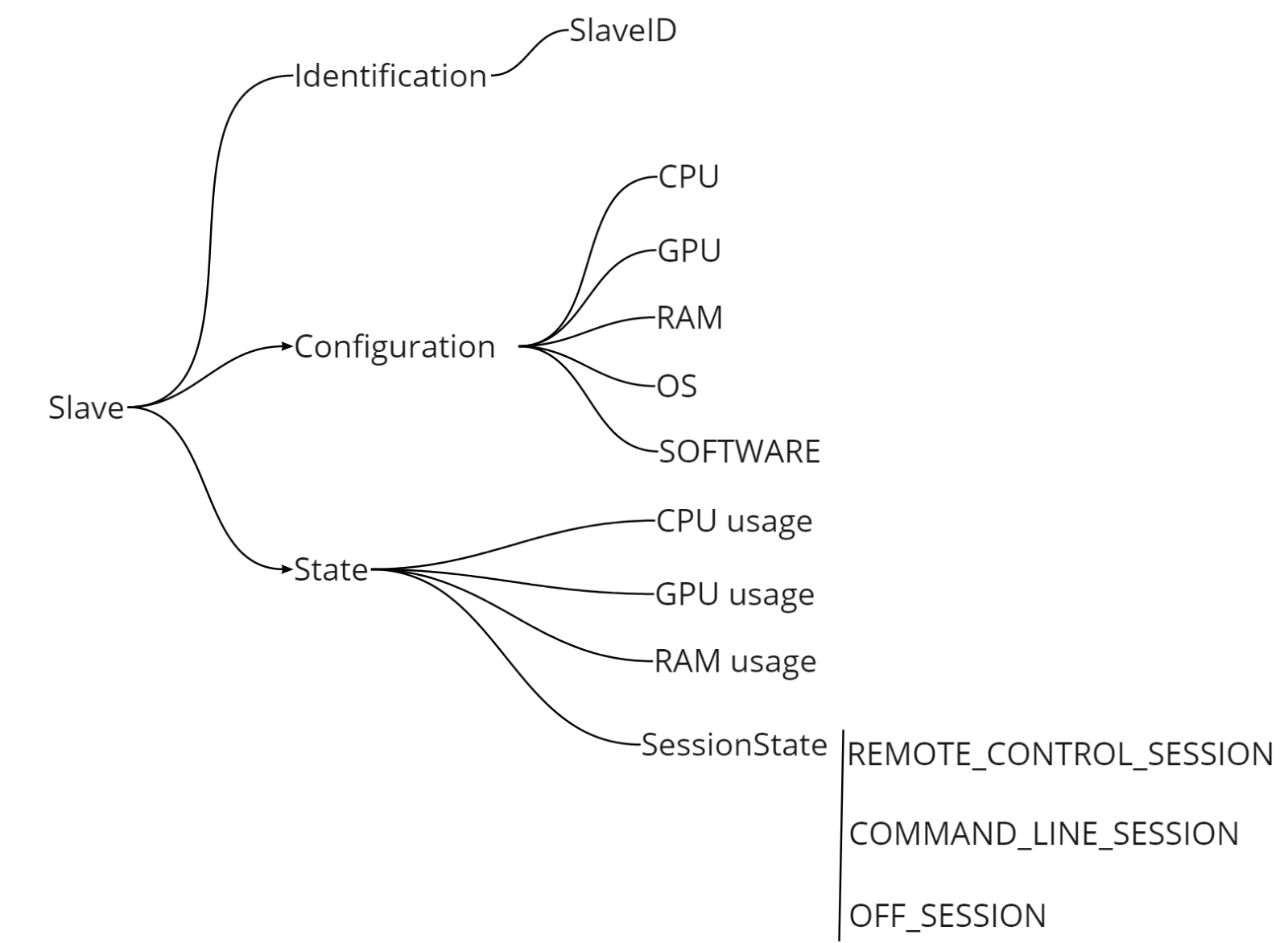
* Implement basic webrtc handshake between client and slave, the detailed initializing process are provided in previous section.
* Storing SessionID pair to process session handshake procedure.
* Inform Host for any SessionID pair change.

Requirement

## Server

Contain source code of Host device.

### Object



# **Appendix**

## **Terminology**

### Session

Session is a pair of one client – one slave in which, client have the right to establish remote control to slave and freely to control slave device.

* Each client can obtain more than one slave, however, at one time, only one client can obtain one slave device.
* Each session is corresponding to one SessionID pair with one SessionSlaveID and one SessionClientID. When a device registers the remote control, signalling server will search for the corresponding ID in SessionID pair pool, if it’s unavailable, remote connection will be denied.
* Session active time is the interval between one Session ID pair is on the queue of signalling server.

### Remote Control (abbreviation: RC)

Remote control is a sub-concept of session in which, Client is receiving monitor stream from slave device and be able to send control signal to slave device.

* During one session, one remote control can be establish/close without generation/termination of SessionID pair.
* Session Core is only active during the remote control.
* In remote control close process, client send data\_channel\_on\_close signal through WebRTCdatachannel.
* In reconnect remote control process, client register remote control to signalling server, and send reconnect remote control to Host.

### SignalR

ASP.NET Core SignalR is an open-source library that simplifies adding real-time web functionality to apps.

* In personal cloud computing project, signalR is used as communication mechanism between client and host.
* Information about signalR can be found in [this page.](https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-5.0)
* SignalR is managed in ClientSocket, ClientHub.

### Named Pipe

A named pipe is a named, one-way or duplex pipe for communication between the pipe server and one or more pipe clients.

* In personal cloud computing project, named pipe is used as IPC (inter-process communication) mechanism between Agent, Session Loader and Session Core.
* Information about named pipe creation, read and write can be found in [this pavge.](https://docs.microsoft.com/en-us/windows/win32/ipc/named-pipes)
* Pipe is managed in CorePipeSink, LoaderPipeSrc, LoaderPipeSink, AgentPipeSrc file.

### WebSocket

WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection.

* In personal cloud computing project, WebSocket is used to maintain connection between Client, Slave and Signalling server, Agent and Host.
* Information about WebSocket standard can be found in [this page.](https://en.wikipedia.org/wiki/WebSocket#:~:text=WebSocket%20is%20a%20computer%20communications,WebSocket%20is%20distinct%20from%20HTTP.)
* In Agent and Session Core sub-project, WebSocket is handled by [libsoup](https://wiki.gnome.org/Projects/libsoup) library.

### Gstreamer

GStreamer is a framework for creating streaming media applications.

* In personal cloud computing project, Gstreamer is the main framework used for capturing, encoding, packetize video and audio and WebRTC management for Slave’s side.
* The reason why we choose Gstreamer for stream handling is the openness of gstreamers allow us to enable hardware acceleration in form of gstreamer’s plugins.
* Gstreamer is based on Glib project. Thus, when working with Session Core, developer should follow the coding convention of Glib project.
* Information about WebSocket standard can be found in [this page.](https://gstreamer.freedesktop.org/documentation/index.html?gi-language=c)

### ASP.NET

.NET is a developer platform made up of tools, programming languages, and libraries for building many different types of applications.

* In personal cloud computing project, ASP.NET is the main framework used for Host side in managing Slave devices as well as user request control.
* Information about ASP.NET standard can be found in [this page.](https://dotnet.microsoft.com/apps/aspnet)

### WebRTC

WebRTC (Web Real-Time Communication) is a free, open-source project providing web browsers and mobile applications with real-time communication (RTC)

* In personal cloud computing project, WebRTC is the main standard used for communication between Client and Slave including video and audio streaming as well as input control signal streaming.
* Information about WebRTC standard can be found in [this page.](https://en.wikipedia.org/wiki/WebRTC)
* WebRTC at Slave’s side is managed by Session Core using [webrtcbin](https://gstreamer.freedesktop.org/documentation/webrtc/index.html#webrtcbin-page) - a gstreamer plugins.

### WebRTCDataChannel - slave’s side

The RTCDataChannel represents a bi-directional data channel between two peers in WebRTC connection.

* In personal cloud computing project, WebRTCDataChannel is the main method used for communication between Client and Slave including input control signal and remote-control management.
* Information about WebRTCDataChannel can be found in [this page.](https://www.w3.org/TR/webrtc/#rtcdatachannel)
* WebRTCDataChannel in PCC is managed by webrtcbin and triggered by on-message-string and on-message-data signal. Further information about this method can be found [here.](https://gstreamer.freedesktop.org/documentation/webrtclib/gstwebrtc-datachannel.html?gi-language=c#GstWebRTCDataChannel)

### 