Building Video Chat with Elixir and Phoenix

Anil Wadghule
@anildigital

ElixirConfEU 2018

About me

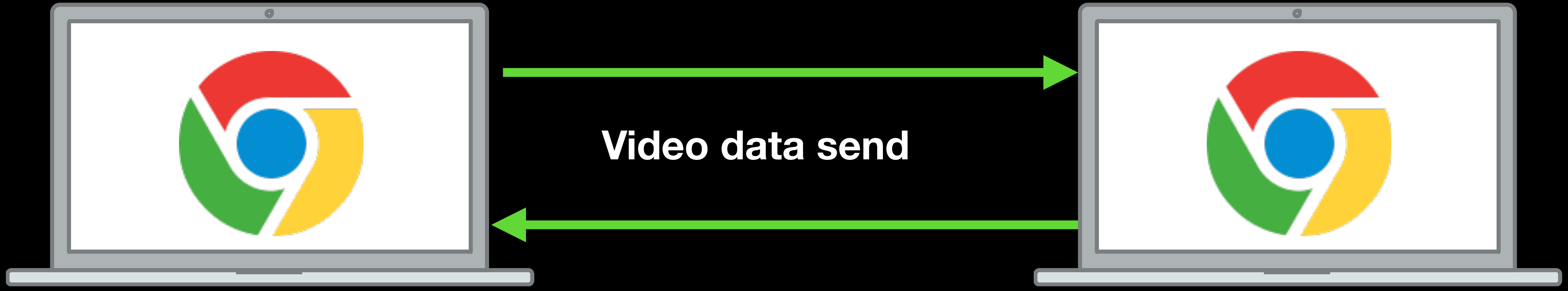
❤️☕ 💚✈️

💙🇯🇵 💜♬

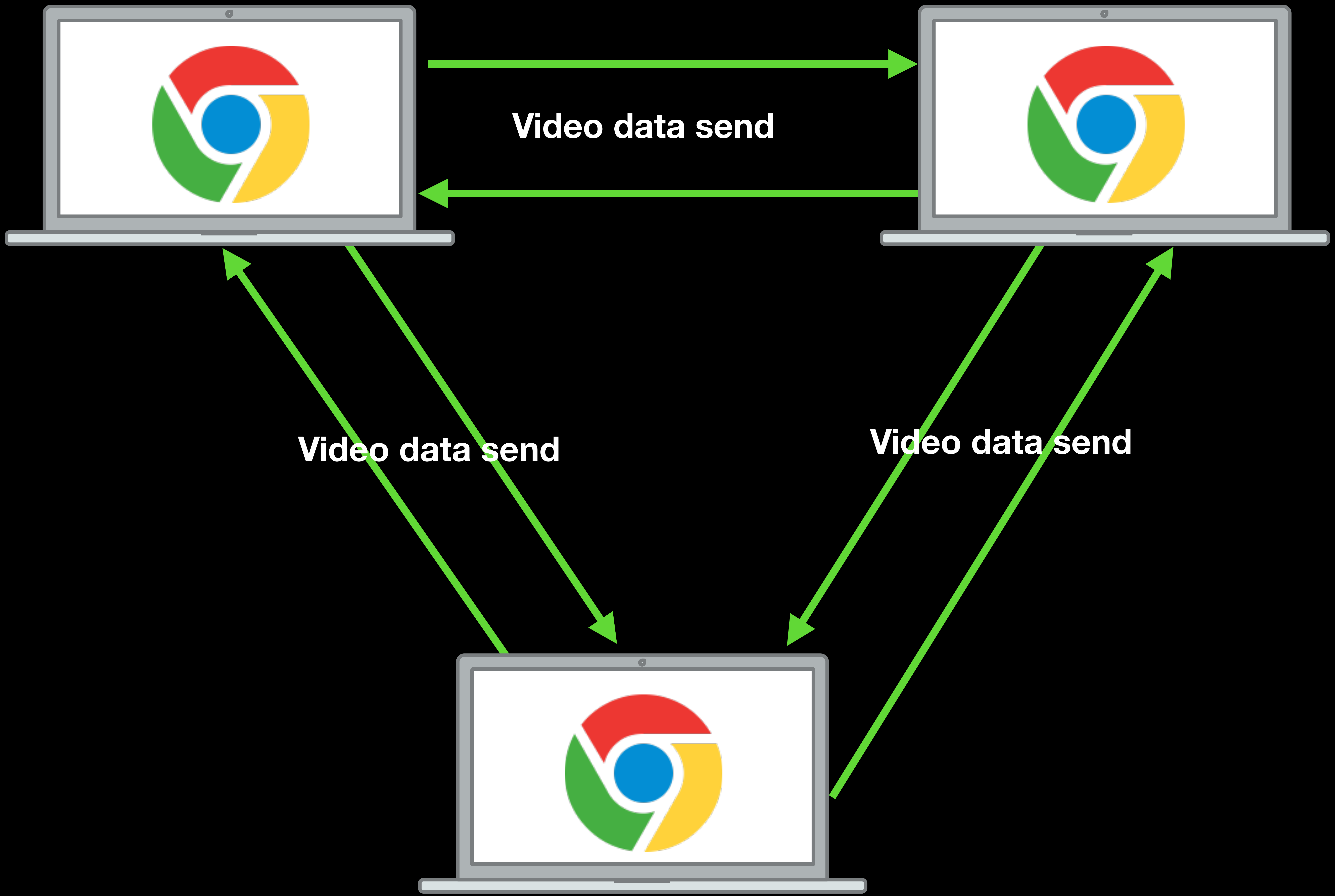
https://skatter.me

How a Video Chat works?

WebRTC peer to peer

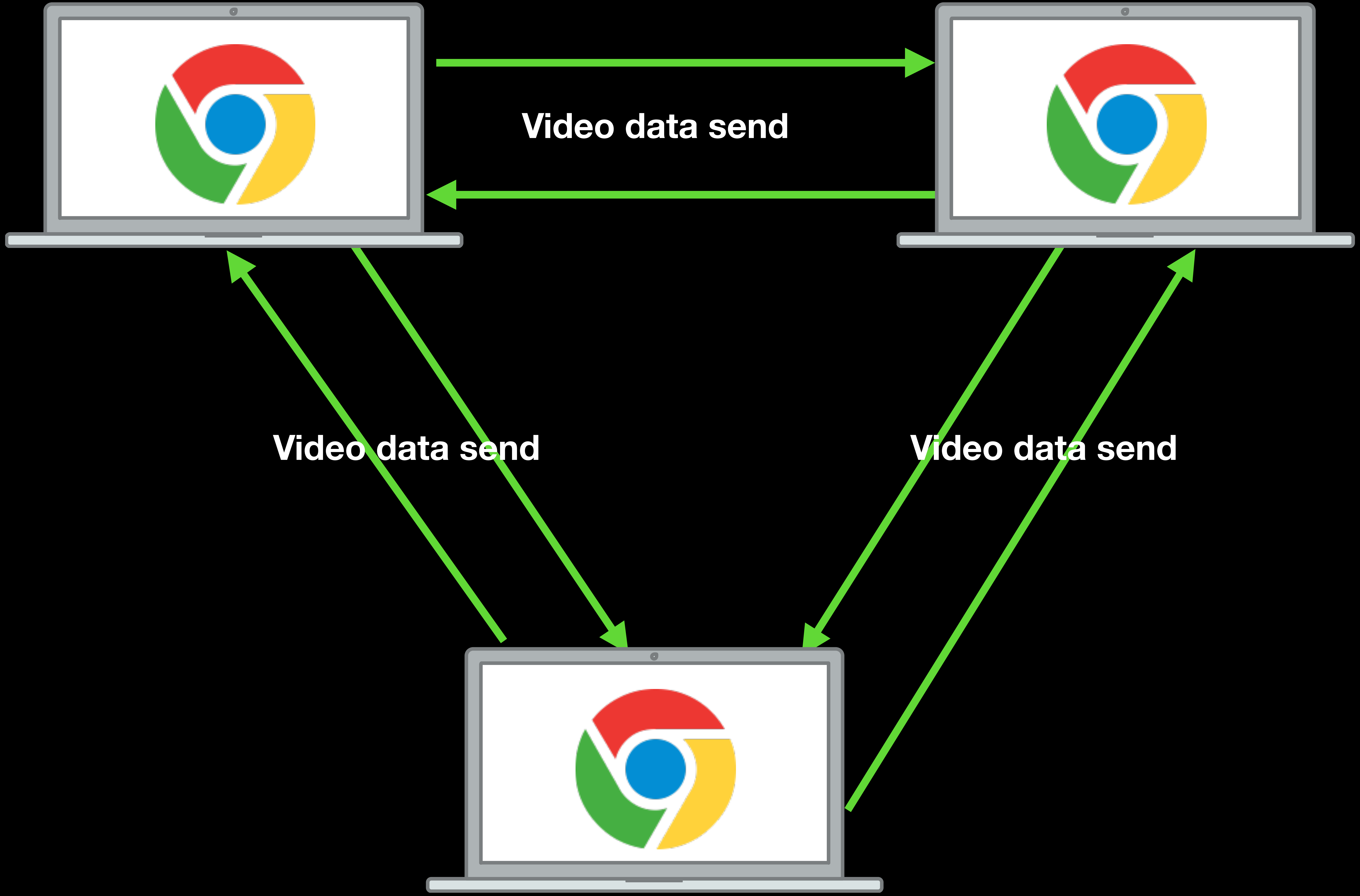


WebRTC peer to peer

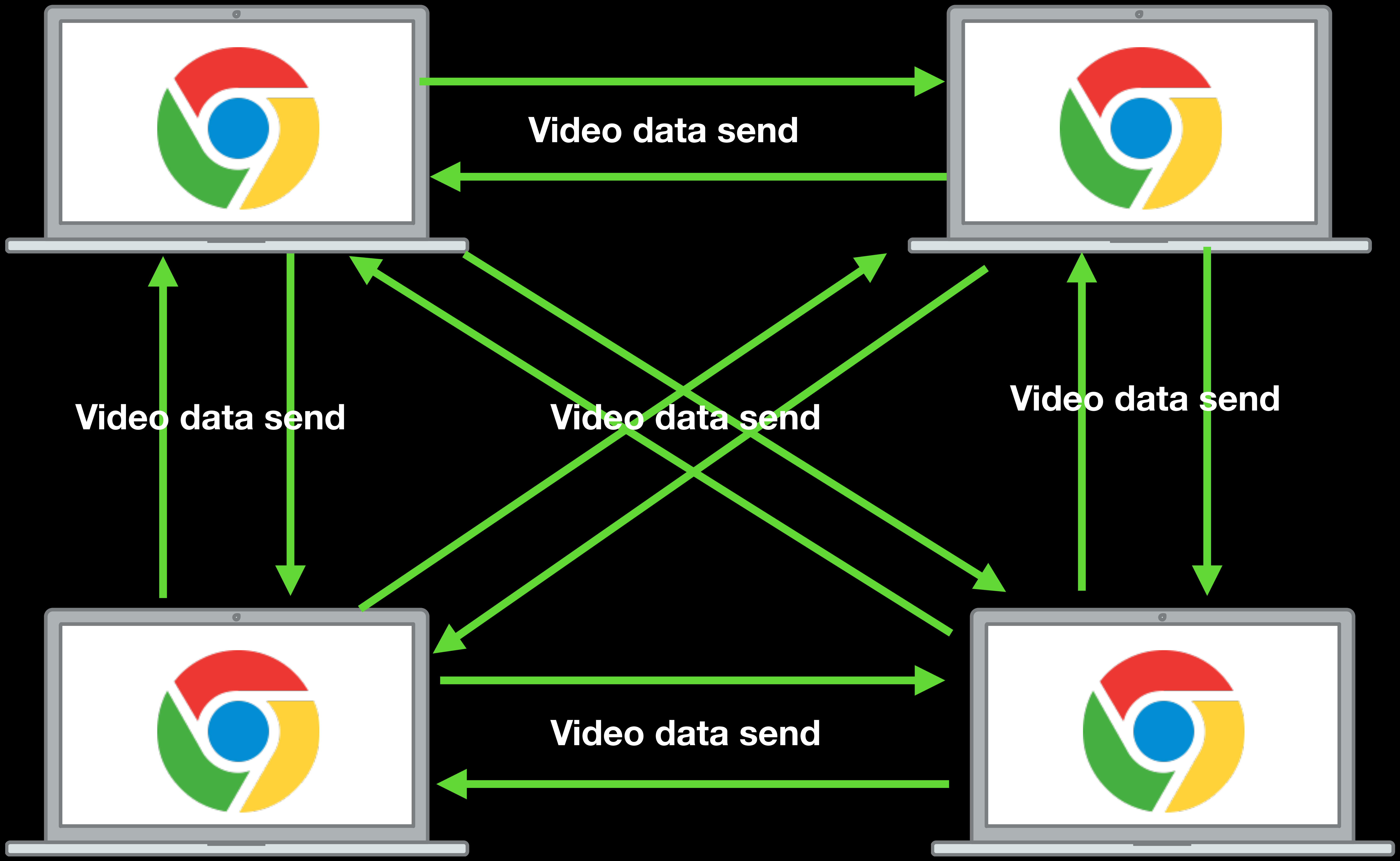


WebRTC peer to peer

What if fourth user joins video chat?



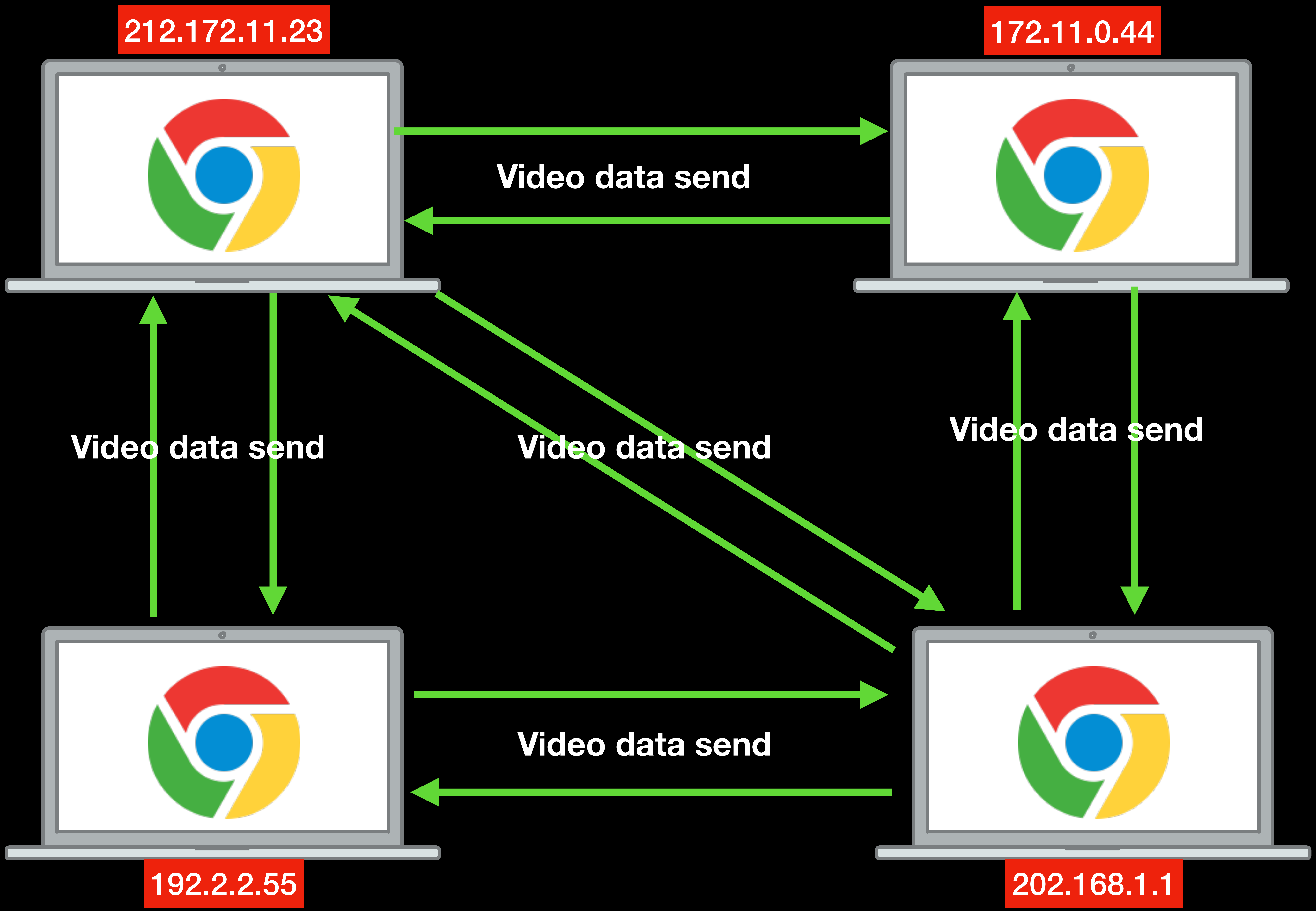
WebRTC peer to peer



WebRTC peer to peer

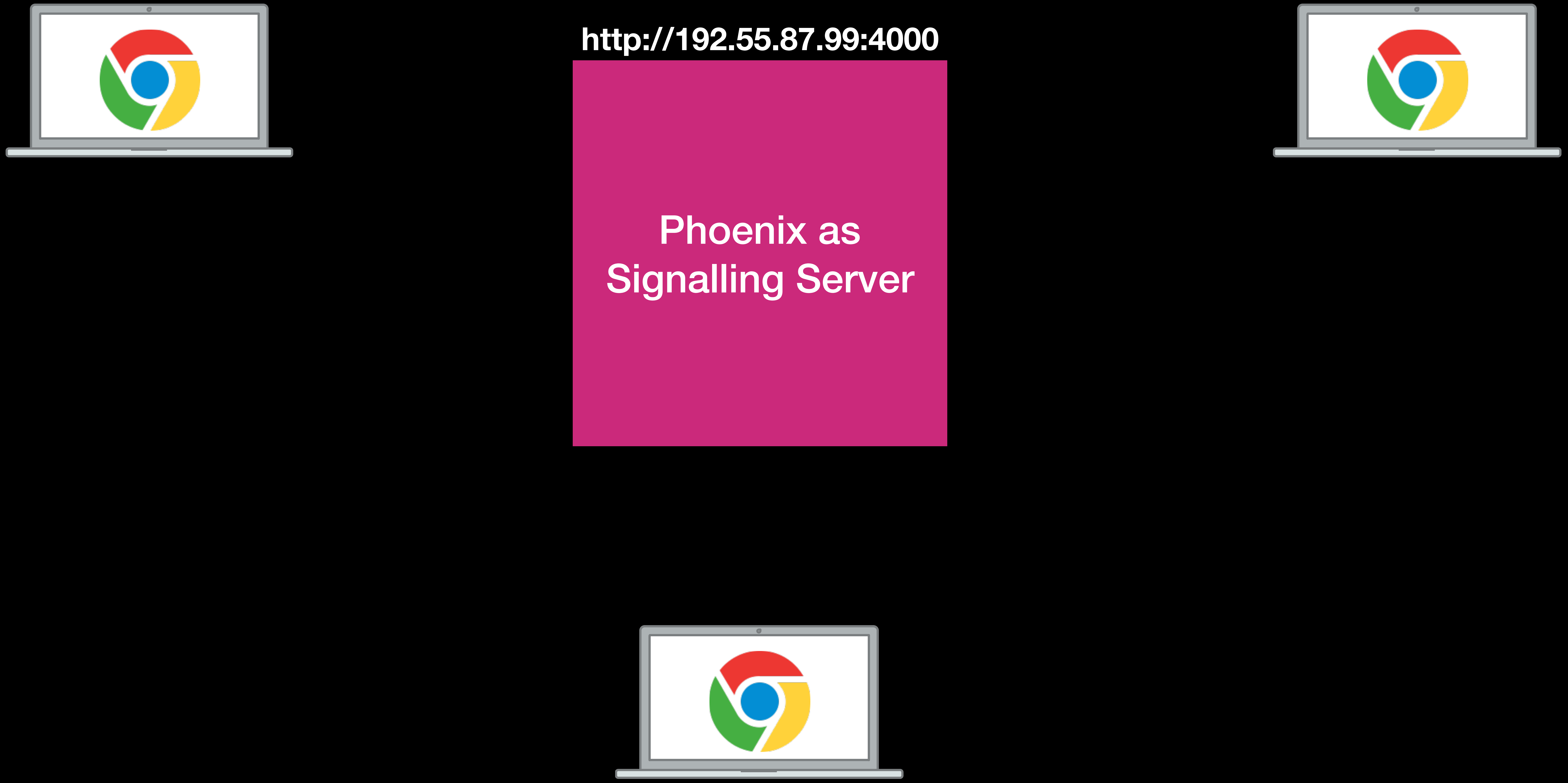
How to communicate?

- Hardcode IP addresses?

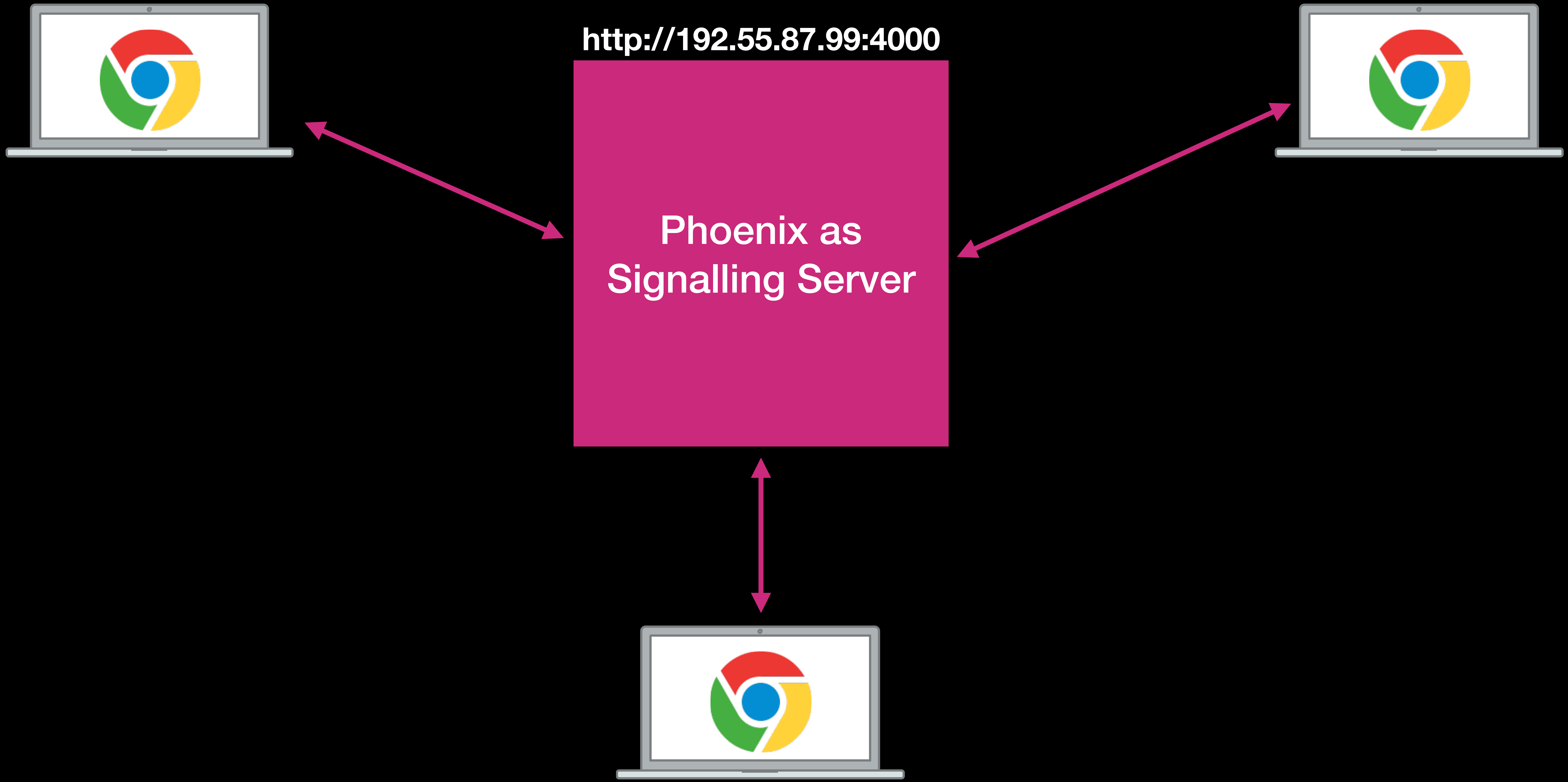


WebRTC peer to peer

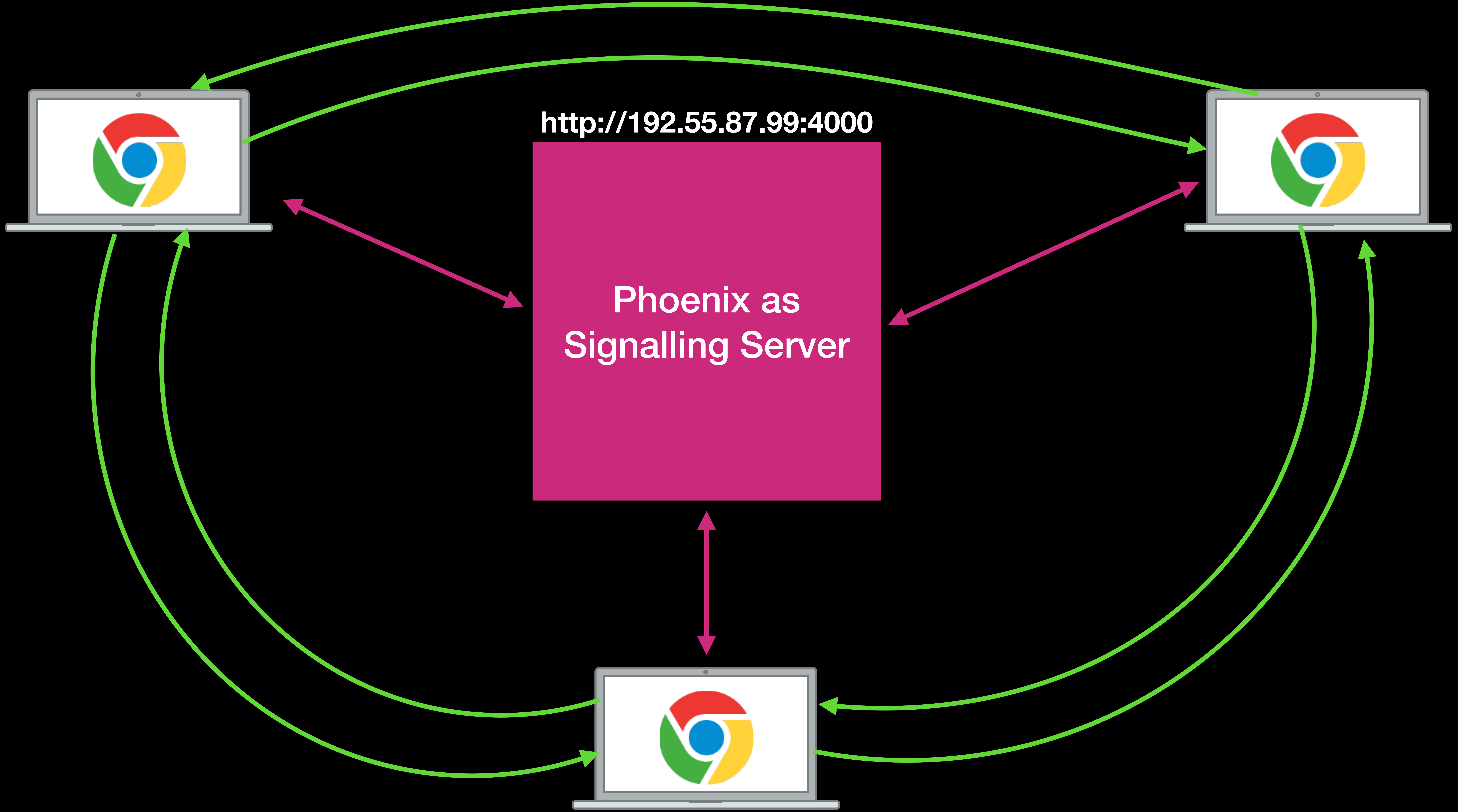
Need of Signalling Server



Signalling Server

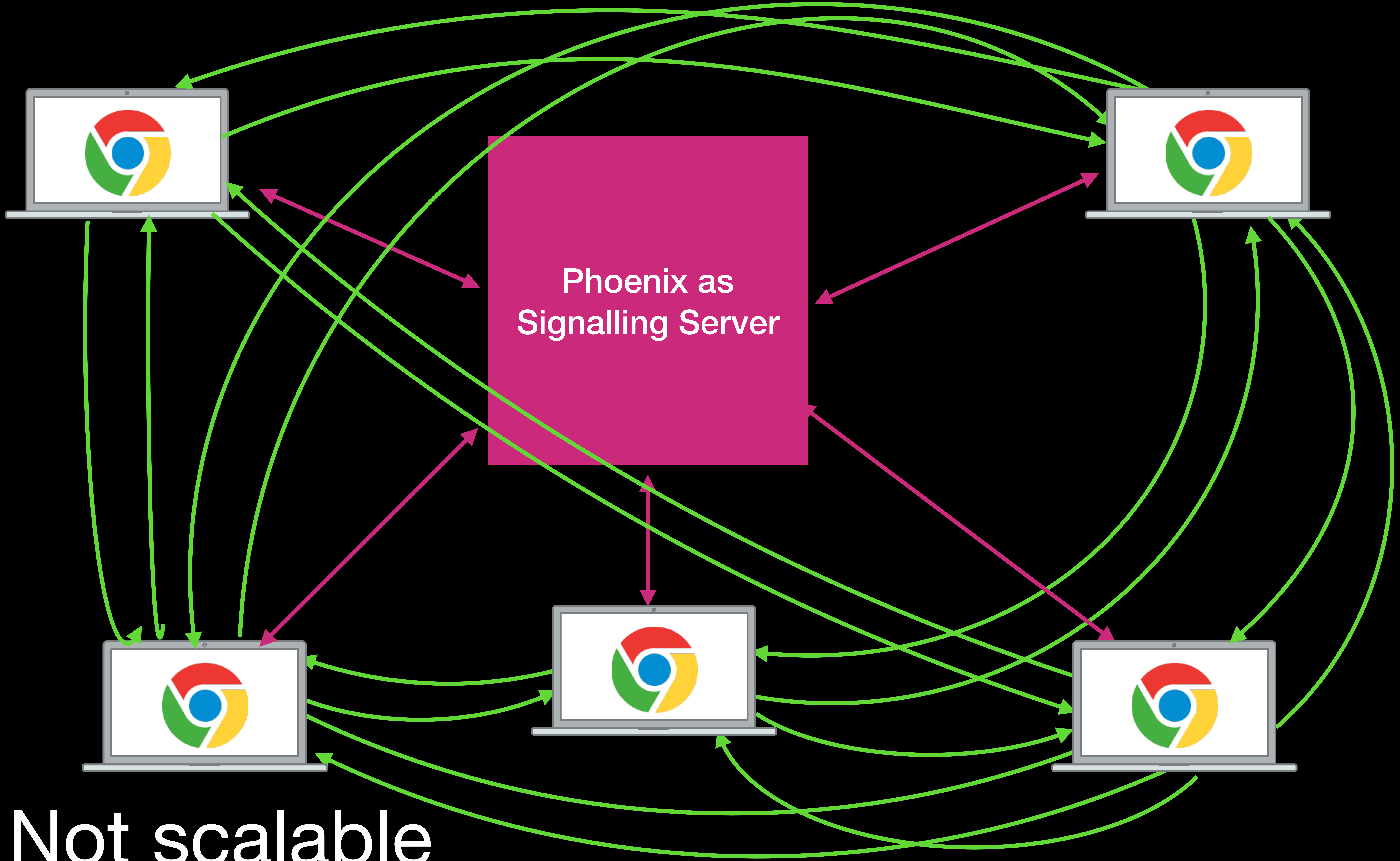


Signalling Server



Signalling Server

WebRTC Peer to Peer is not scalable

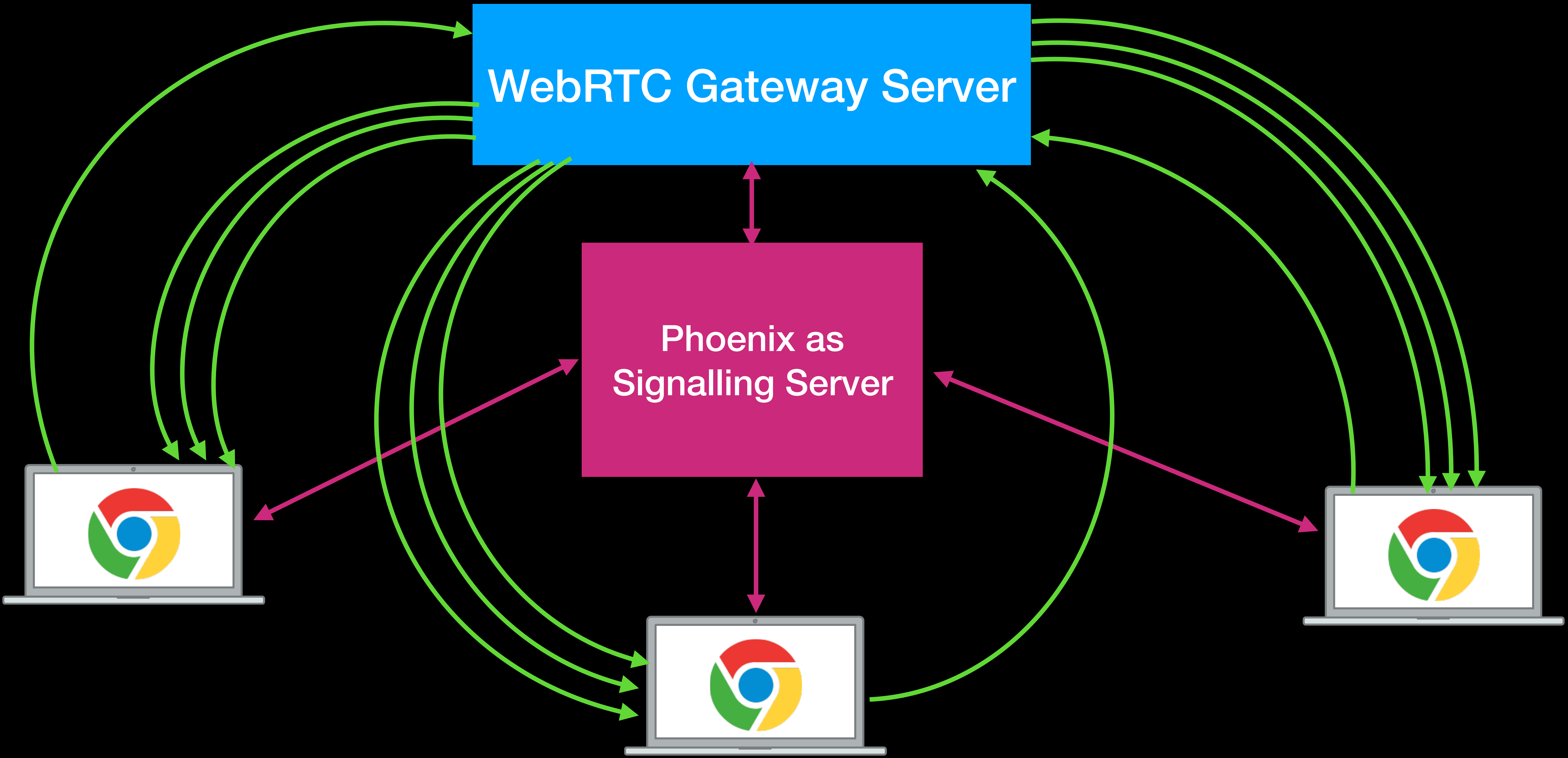


Not scalable

Problem?

- Uplink: 4 UDP streams
- Downlink: 4 UDP streams

Solution - WebRTC Gateway Server



WebRTC Gateway Server

Now

- Uplink: 1 UDP stream
- Downlink: 2 UDP streams

Now

- Uplink: 1 UDP stream
- Downlink: X UDP streams

Why Elixir & Phoenix?

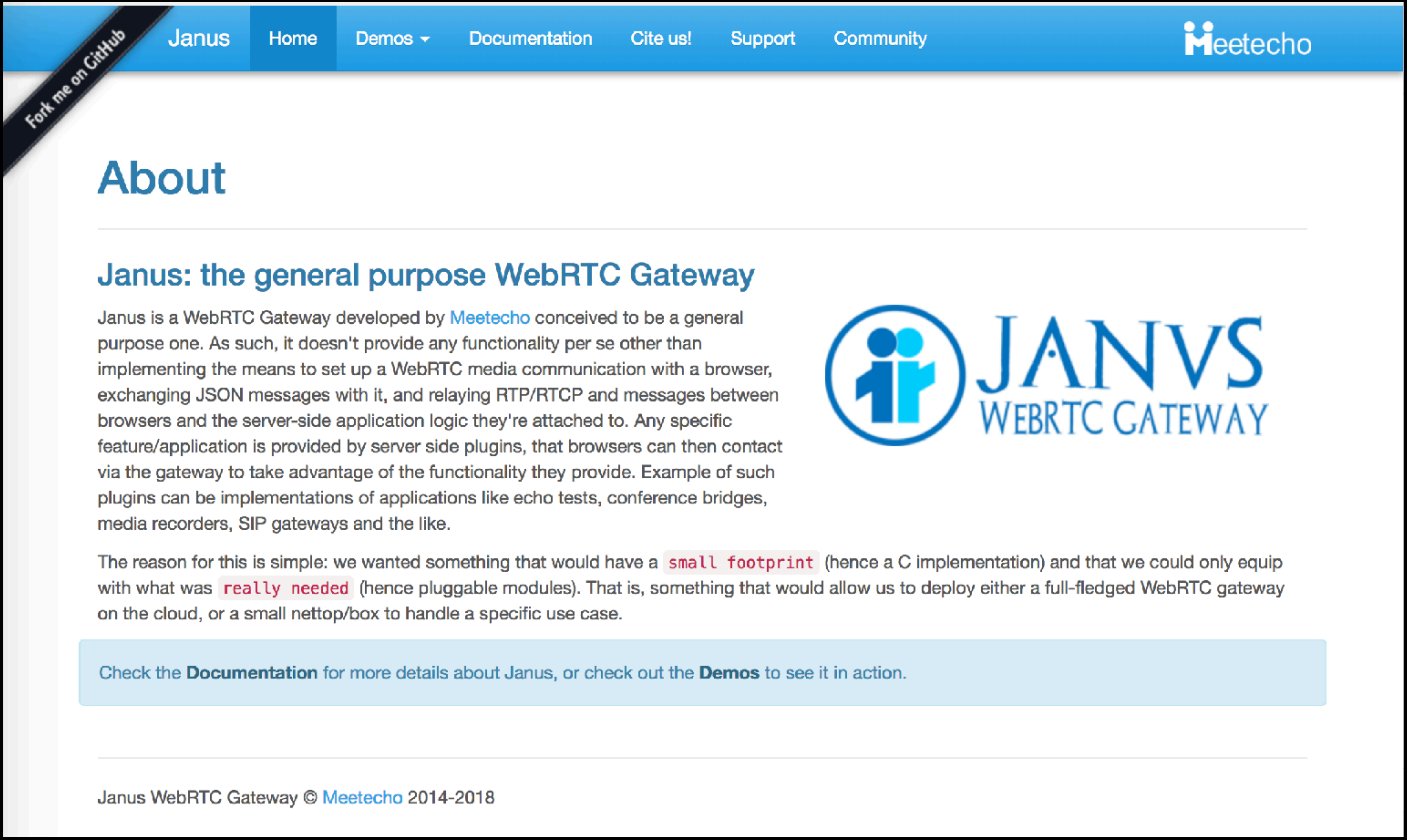
Why Elixir & Phoenix?

- Elixir
  - OTP features - GenServer, Agent, GenEvent, GenStage, Supervisor
- Phoenix
  - Phoenix for channels (signalling), web app basics
  - Authentication
  - Libraries

Why Elixir & Phoenix?

- Actor model
- Battle tested OTP abstractions
- Fan out
- Fault Tolerant
- Soft realtime

Janus WebRTC Gateway Server



About

Janus: the general purpose WebRTC Gateway

Janus is a WebRTC Gateway developed by Meetecho conceived to be a general purpose one. As such, it doesn't provide any functionality per se other than implementing the means to set up a WebRTC media communication with a browser, exchanging JSON messages with it, and relaying RTP/RTCP and messages between browsers and the server-side application logic they're attached to. Any specific feature/application is provided by server side plugins, that browsers can then contact via the gateway to take advantage of the functionality they provide. Example of such plugins can be implementations of applications like echo tests, conference bridges, media recorders, SIP gateways and the like.

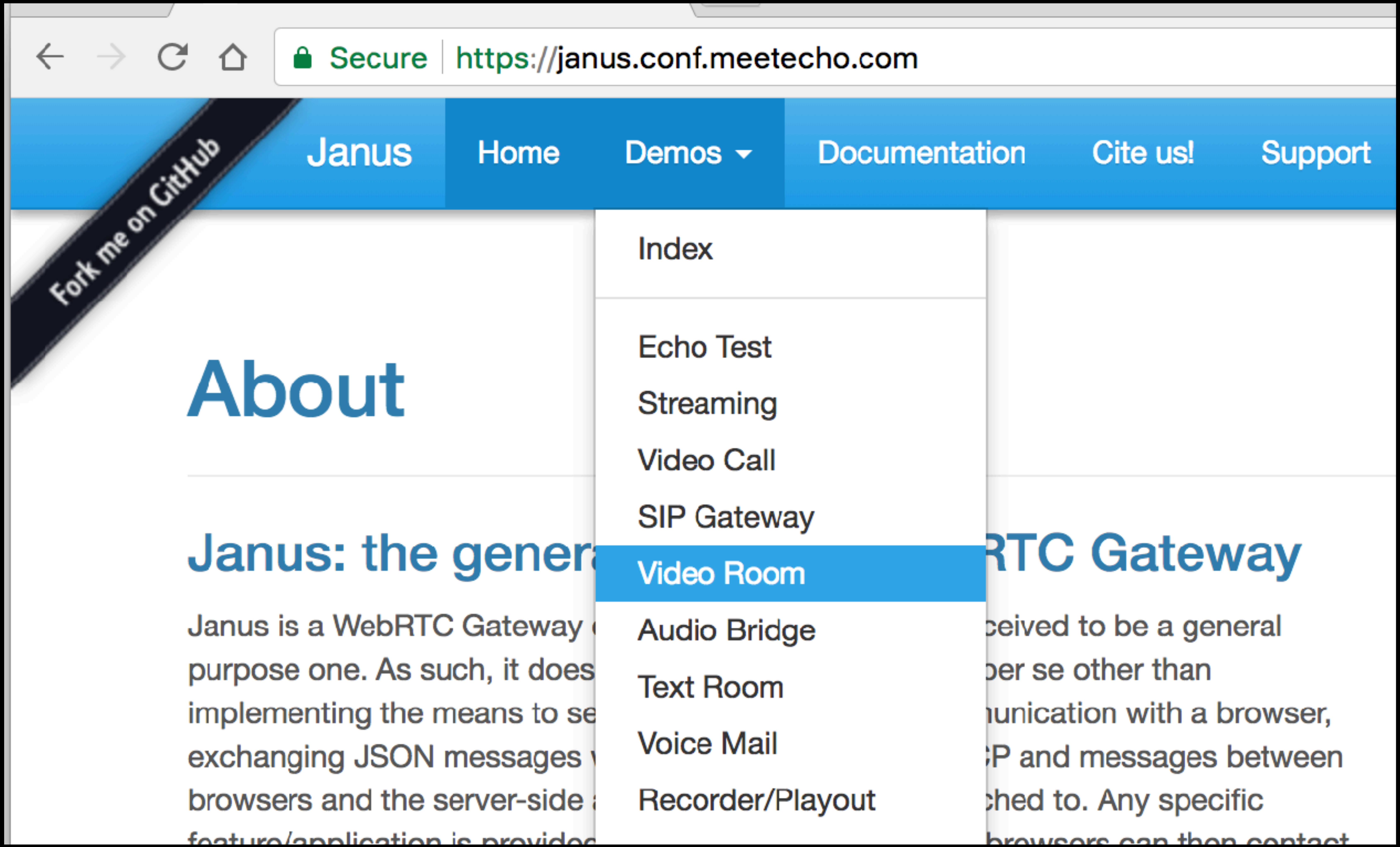
The reason for this is simple: we wanted something that would have a `small footprint` (hence a C implementation) and that we could only equip with what was `really needed` (hence pluggable modules). That is, something that would allow us to deploy either a full-fledged WebRTC gateway on the cloud, or a small nettop/box to handle a specific use case.

Check the **Documentation** for more details about Janus, or check out the **Demos** to see it in action.

Janus WebRTC Gateway © Meetecho 2014-2018

Janus - WebRTC Gateway

- Open source
- Small footprint (C implementation)
- Pluggable modules



About

Janus: the gener RTC Gateway

Janus is a WebRTC Gateway ceived to be a general
purpose one. As such, it does per se other than
implementing the means to se unication with a browser,
exchanging JSON messages P and messages between
browsers and the server-side ched to. Any specific
feature/application is provided browsers can then contact

Janus Plugins

Janus APIs

- RESTful (HTTP)
- WebSockets
- RabbitMQ
- MQTT
- UnixSockets

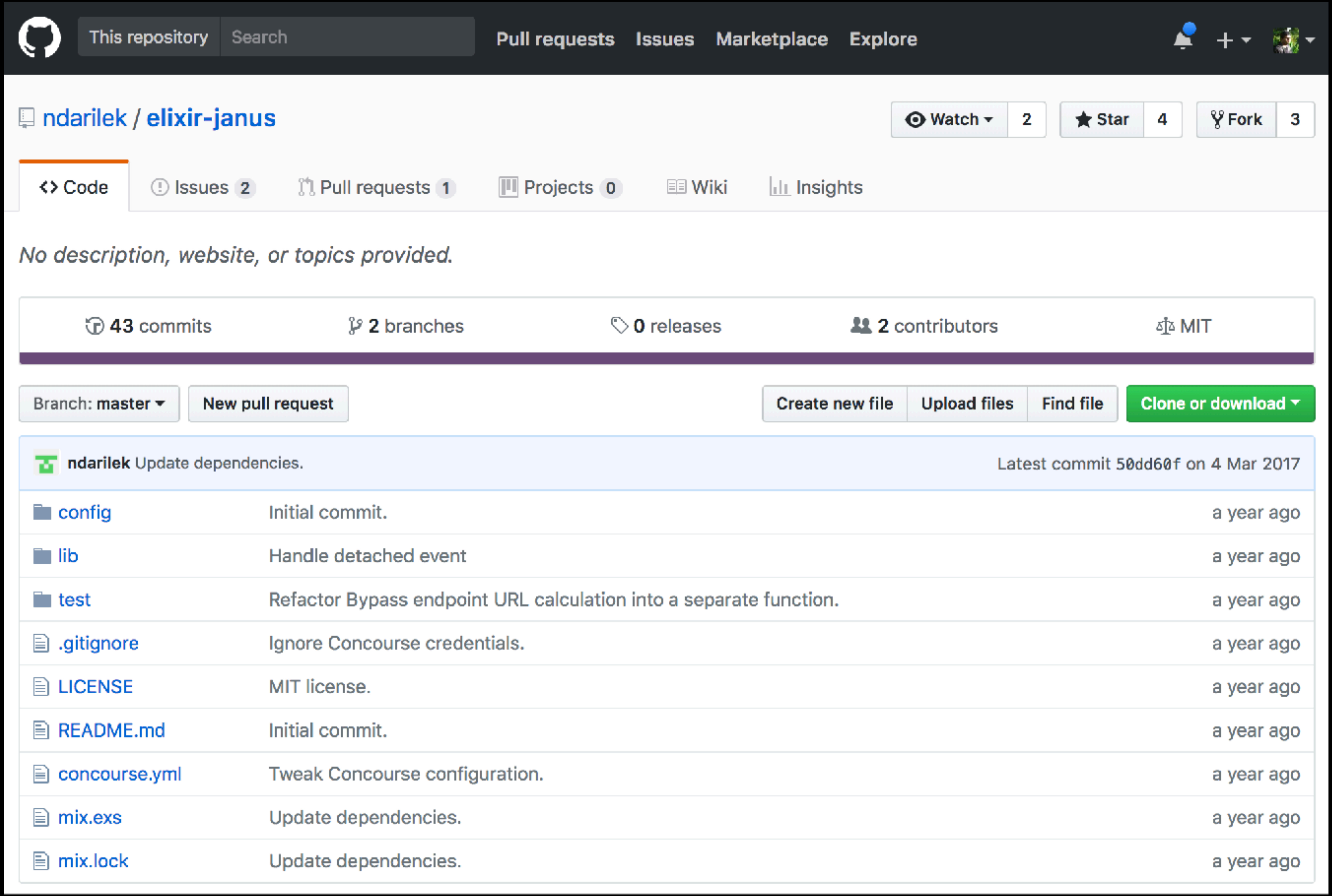
Using Janus RESTful/HTTP APIs

- `POST /janus` (to create Janus Session)
- `POST Session` (attach plugin to session)
- `POST Plugin` (for other requests)
- `GET Session` (for listening for events)

Janus Video Room Events

- slowlink
- configured
- talking (true/false)
- publishers
- leaving
- unpublished
- webrtcup
- media (true/false)
- hangup
- …

Talking with Janus with Elixir



ndarilek / elixir-janus

No description, website, or topics provided.

43 commits · 2 branches · 0 releases · 2 contributors · MIT

ndarilek Update dependencies. Latest commit 50dd60f on 4 Mar 2017

config	Initial commit.	a year ago
lib	Handle detached event	a year ago
test	Refactor Bypass endpoint URL calculation into a separate function.	a year ago
.gitignore	Ignore Concourse credentials.	a year ago
LICENSE	MIT license.	a year ago
README.md	Initial commit.	a year ago
concourse.yml	Tweak Concourse configuration.	a year ago
mix.exs	Update dependencies.	a year ago
mix.lock	Update dependencies.	a year ago

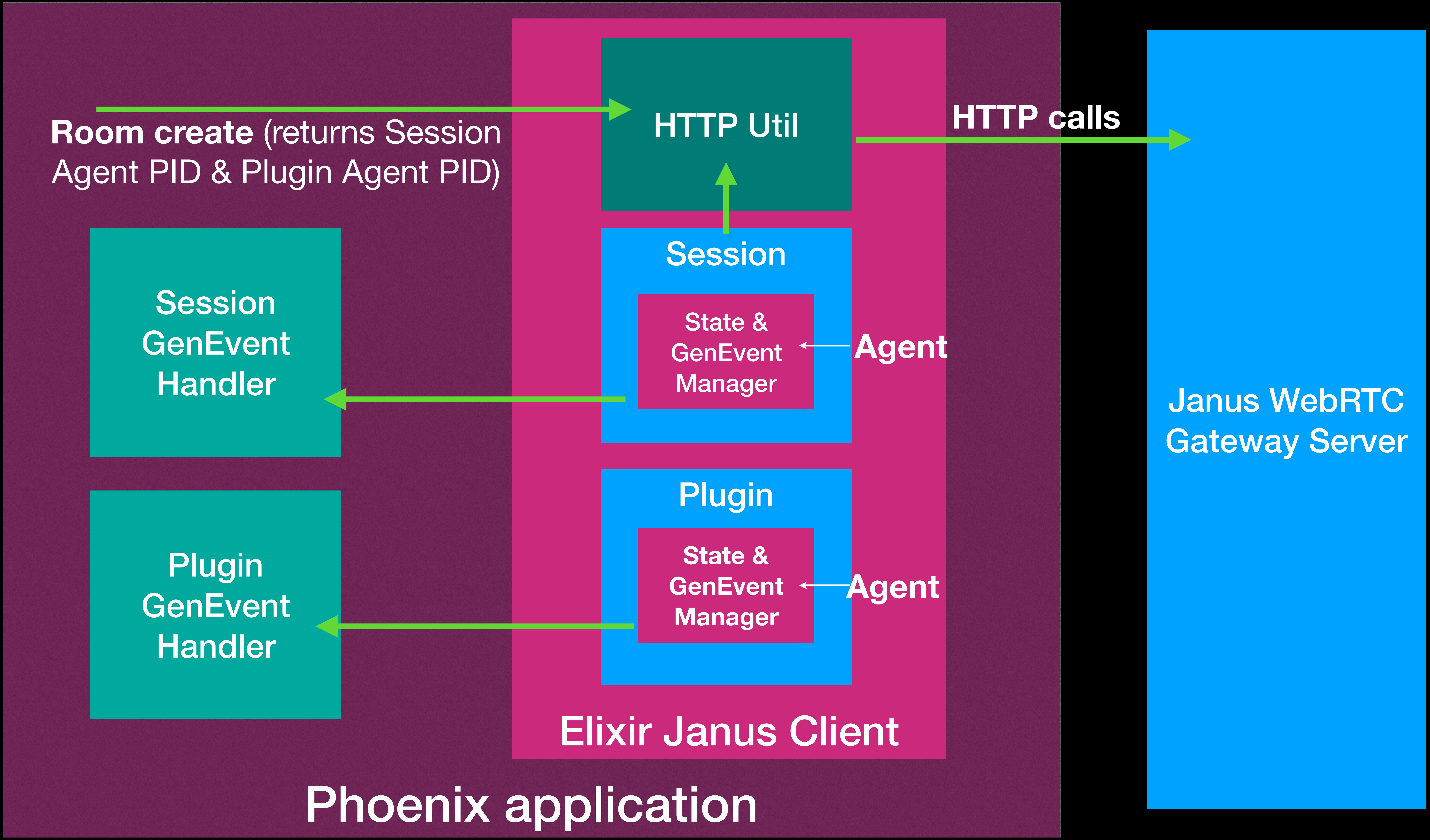
elixir-janus client

https://github.com/ndarilek/elixir-janus

- HTTP client
- State
- Events

elixir-janus client structure

- Session module
- Plugin module
- Util module


Room create (returns Session Agent PID & Plugin Agent PID)
HTTP Util
HTTP calls
Session
State & GenEvent Manager
Agent
Session GenEvent Handler
Plugin
State & GenEvent Manager
Agent
Plugin GenEvent Handler
Elixir Janus Client
Phoenix application
Janus WebRTC Gateway Server

```elixir
import Janus.Util

defmodule Janus.Session do

  @enforce_keys [:id, :base_url, :event_manager]
  defstruct [
    :id,
    :base_url,
    :event_manager,
    handles: %{}
  ]
…
```

```elixir
import Janus.Util

defmodule Janus.Plugin do

  @enforce_keys [:id, :base_url, :event_manager]
  defstruct [
    :id,
    :base_url,
    :event_manager
  ]
…
```

```
defmodule Janus.Session do
  …
  …
  def start(url) do
    case post(url, %{janus: :create}) do
```

```
defmodule Janus.Session do
  …
  …
  def start(url) do
    case post(url, %{janus: :create}) do
      {:ok, body} ->
        id = body.data.id
        {:ok, event_manager} = GenEvent.start_link()
        session = %Janus.Session{
          id: id,
          base_url: "#{url}/#{id}",
          event_manager: event_manager
        }
```

```
defmodule Janus.Session do
  …
  …
  def start(url) do
    case post(url, %{janus: :create}) do
      {:ok, body} ->
        id = body.data.id
        {:ok, event_manager} = GenEvent.start_link()
        session = %Janus.Session{
          id: id,
          base_url: "#{url}/#{id}",
          event_manager: event_manager
        }
        Agent.start(fn -> session end)
      v -> v
    end
  end
```

```elixir
defmodule Janus.Session do
  …
  def attach_plugin(pid, id) do
```

```
defmodule Janus.Session do
  …
  def attach_plugin(pid, id) do
    base_url = Agent.get(pid, &(&1.base_url))
    v = case post(base_url, %{janus: :attach, plugin: id}) do
      {:ok, body} ->
```

```elixir
defmodule Janus.Session do
  …
  def attach_plugin(pid, id) do
    base_url = Agent.get(pid, &(&1.base_url))
    v = case post(base_url, %{janus: :attach, plugin: id}) do
      {:ok, body} ->
        id = body.data.id
        {:ok, event_manager} = GenEvent.start_link()
        plugin = %Janus.Plugin{
          id: id,
          base_url: "#{base_url}/#{id}",
          event_manager: event_manager
        }
```

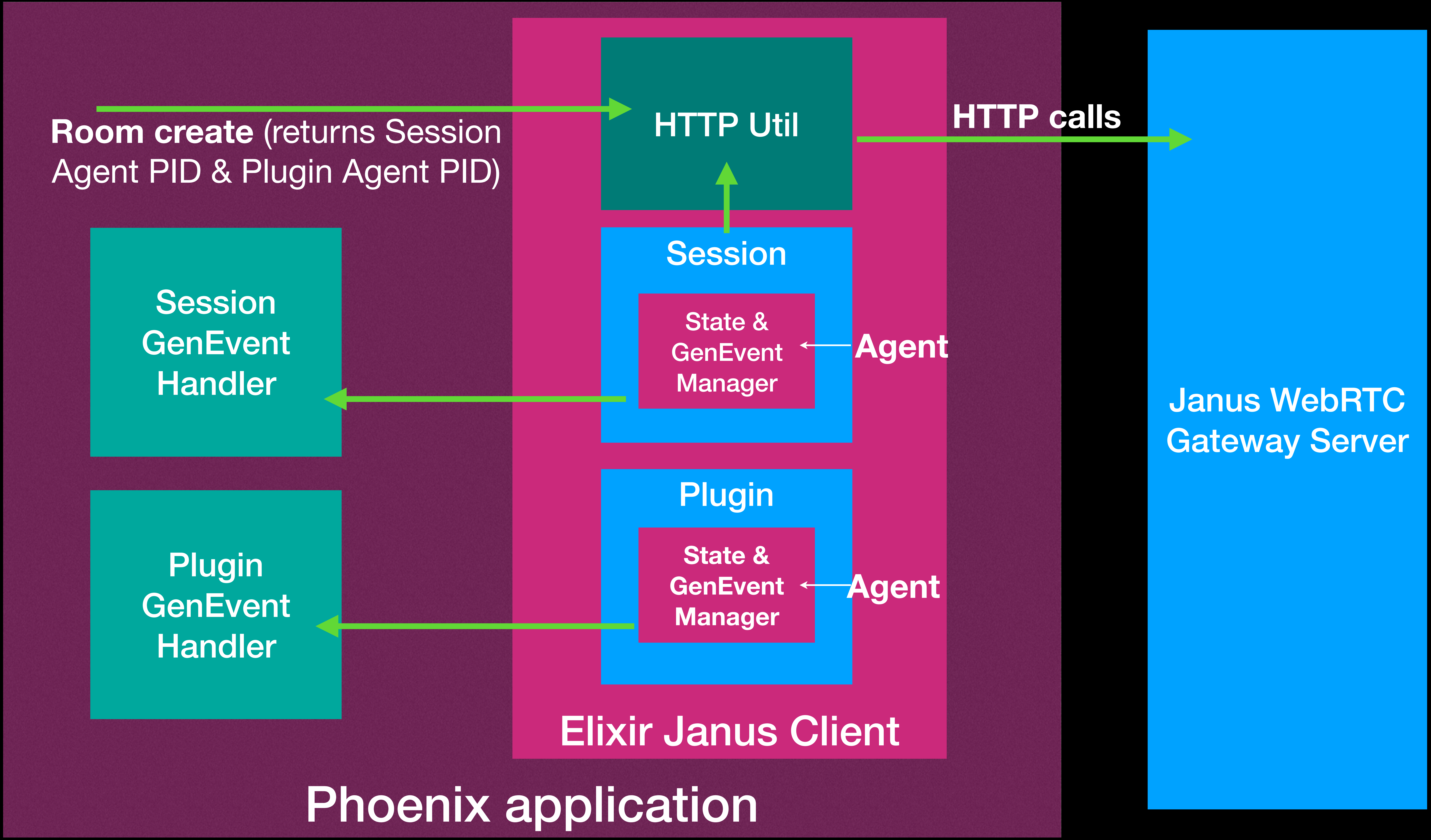
```elixir
defmodule Janus.Session do
  …
  def attach_plugin(pid, id) do
    base_url = Agent.get(pid, &(&1.base_url))
    v = case post(base_url, %{janus: :attach, plugin: id}) do
      {:ok, body} ->
        id = body.data.id
        {:ok, event_manager} = GenEvent.start_link()
        plugin = %Janus.Plugin{
          id: id,
          base_url: "#{base_url}/#{id}",
          event_manager: event_manager
        }
        {:ok, plugin_pid} = Agent.start(fn -> plugin end)
        Agent.update pid, fn(session) ->
          new_handles = Map.put(session.handles, id, plugin_pid)
          %{ session | handles: new_handles}
        end
```

```
defmodule Janus.Session do
  …

  def add_handler(session, handler, args) do
     Agent.get session, &(GenEvent.add_handler(&1.event_manager,
                                               handler, args))
  end
```

```elixir
defp poll(pid) do
  session = Agent.get pid, &(&1)
```

```
defp poll(pid) do
  session = Agent.get pid, &(&1)
  spawn fn ->
    case get(session.base_url) do
      {:ok, data} ->
        event_manager = session.event_manager
  defp poll(pid) do
    session = Agent.get pid, &(&1)
    spawn fn ->
      case get(session.base_url) do
        {:ok, data} ->
          event_manager = session.event_manager
          case data do
            %{janus: "keepalive"} -> GenEvent.notify(event_manager,
{:keepalive, pid})
```

```
  defp poll(pid) do
    session = Agent.get pid, &(&1)
    spawn fn ->
      case get(session.base_url) do
        {:ok, data} ->
          event_manager = session.event_manager
          case data do
            %{janus: "keepalive"} -> GenEvent.notify(event_manager,
{:keepalive, pid})
            %{sender: sender} ->
             plugin_pid = session.handles[sender]
  defp poll(pid) do
    session = Agent.get pid, &(&1)
    spawn fn ->
      case get(session.base_url) do
        {:ok, data} ->
          event_manager = session.event_manager
          case data do
            %{janus: "keepalive"} -> GenEvent.notify(event_manager,
{:keepalive, pid})
            %{sender: sender} ->
             plugin_pid = session.handles[sender]
              if plugin_pid do
                case data do
                  %{janus: "event", plugindata: plugindata} ->
                    jsep = data[:jsep]
                    Agent.get plugin_pid,
&(GenEvent.notify(&1.event_manager, {:event, pid, plugin_pid,
plugindata.data, jsep}))
```


Room create (returns Session Agent PID & Plugin Agent PID)
HTTP Util
HTTP calls
Session
State & GenEvent Manager
Agent
Session GenEvent Handler
Plugin
State & GenEvent Manager
Agent
Plugin GenEvent Handler
Elixir Janus Client
Phoenix application
Janus WebRTC Gateway Server

Video Room start sequence

- Create Janus Room (HTTP Call)

Video Room start sequence

- Create Janus Room (HTTP Call)
- Init Session Agent PID and Plugin Agent PID

Video Room start sequence

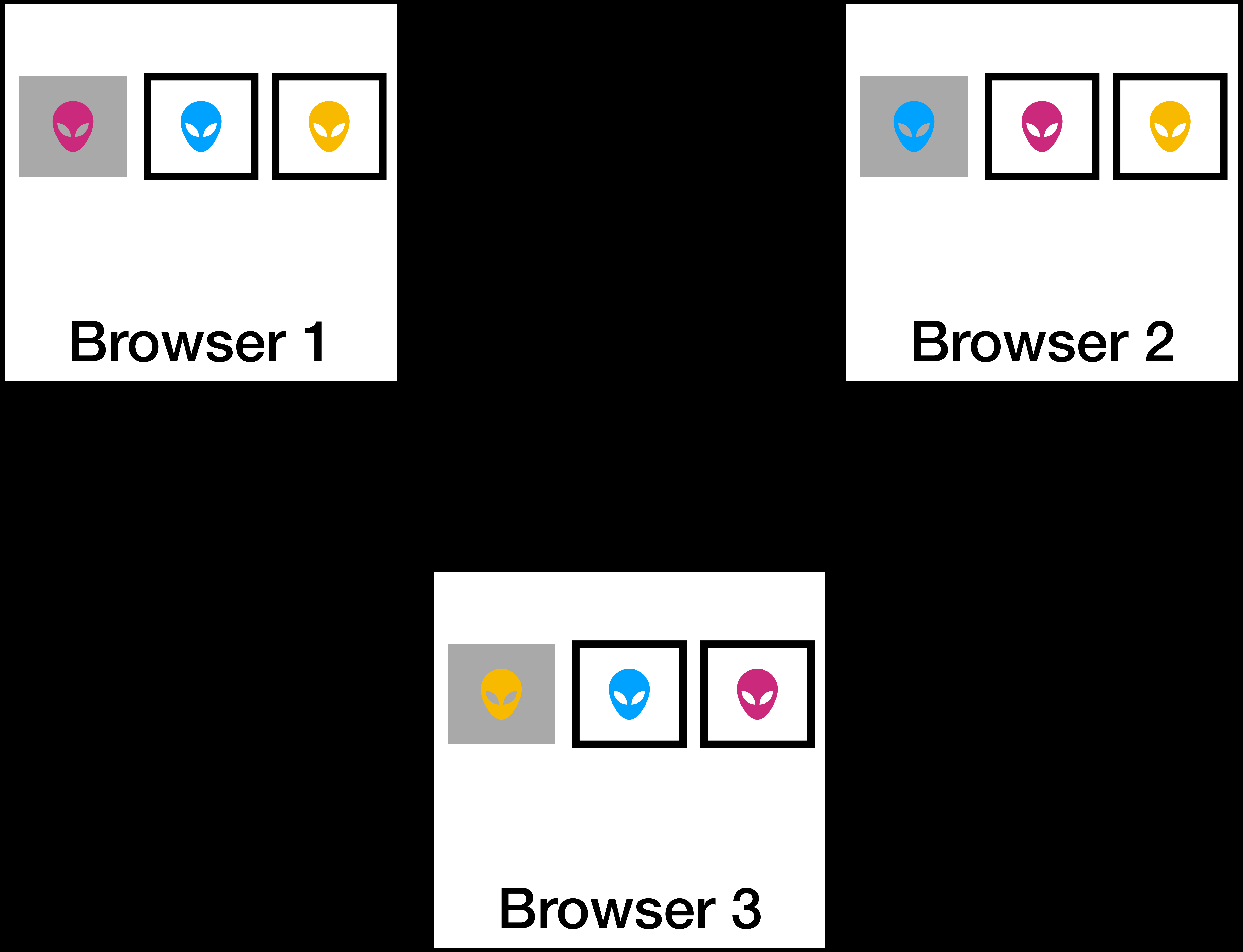
- Create Janus Room (HTTP Call)
- Init Session Agent PID and Plugin Agent PID
- Store these Agents PIDs in Cache (Cache is GenServer based)

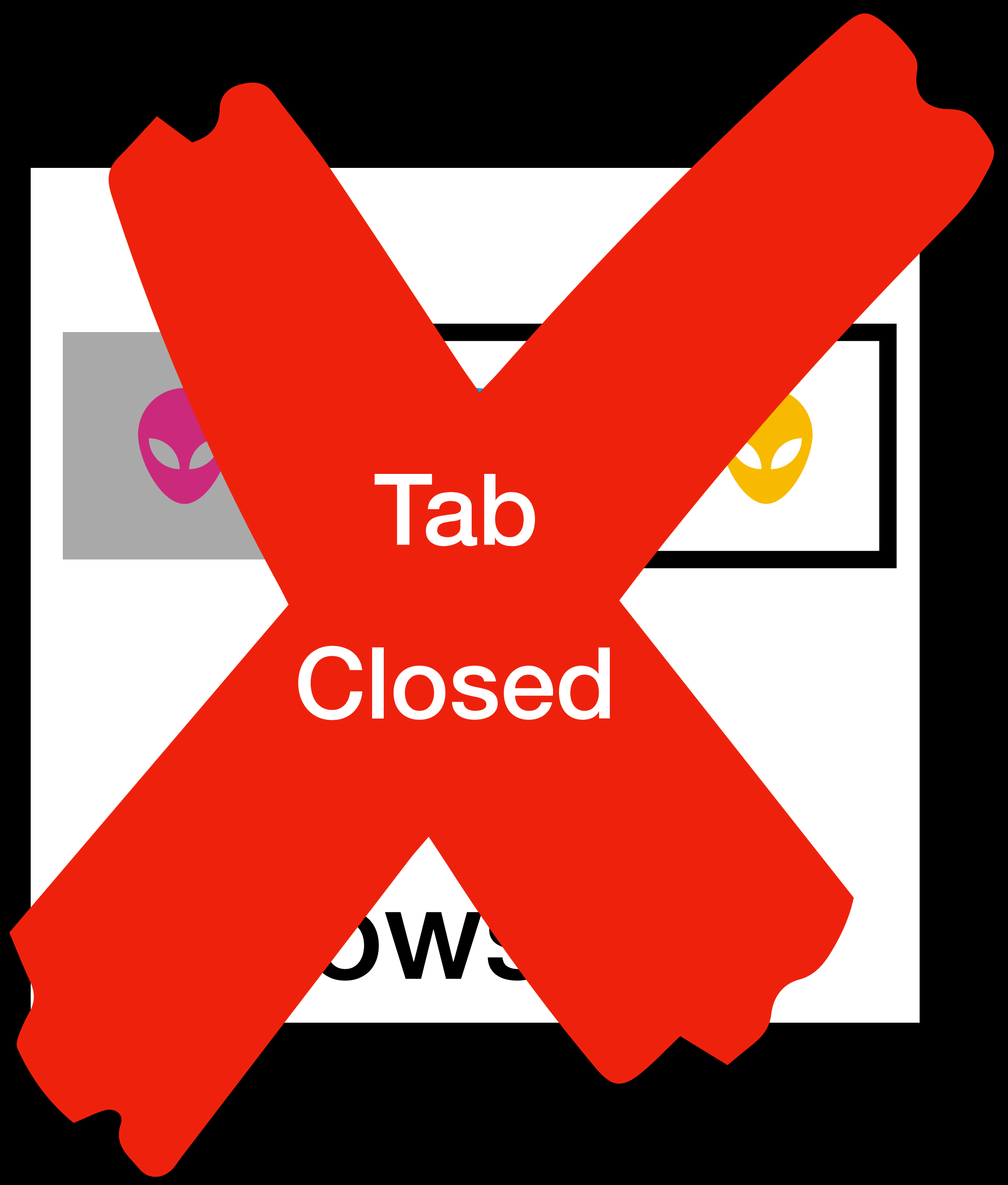
Video Room start sequence

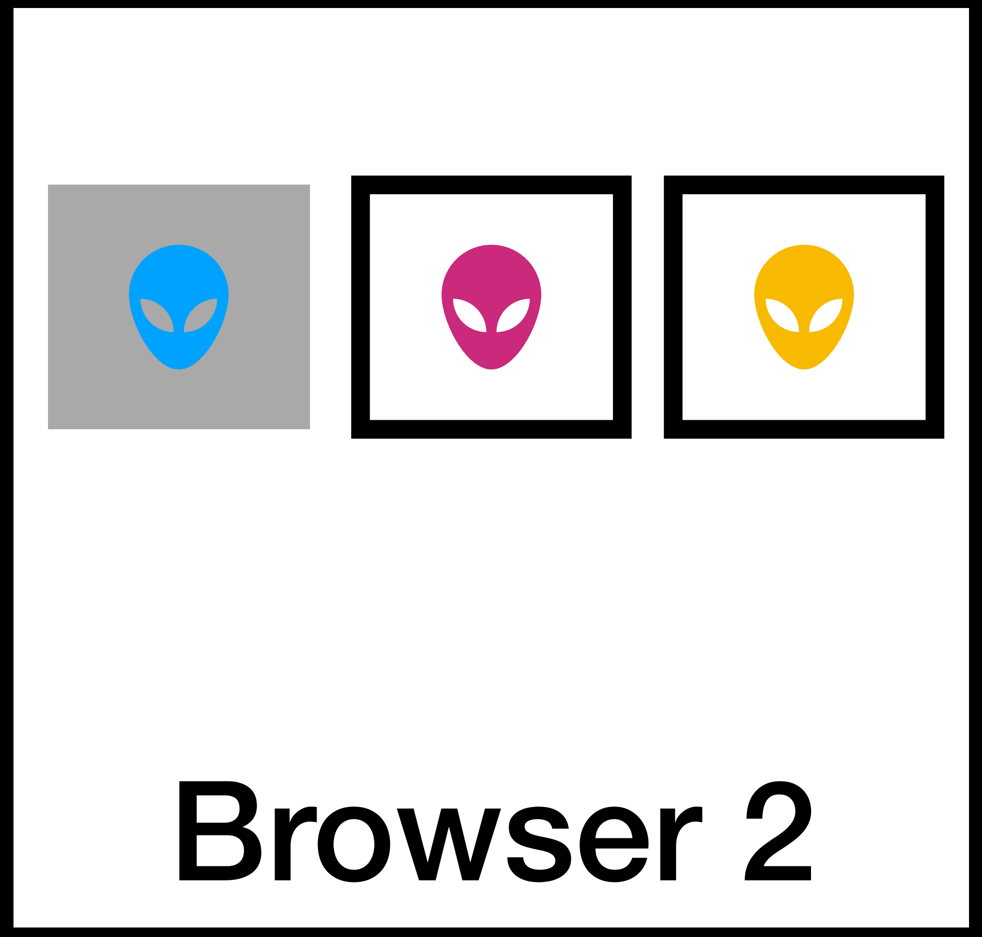
- Create Janus Room (HTTP Call)
- Init Session Agent PID and Plugin Agent PID
- Store these Agents PIDs in Cache (Cache is GenServer based)
- PIDs to further communicate with Janus

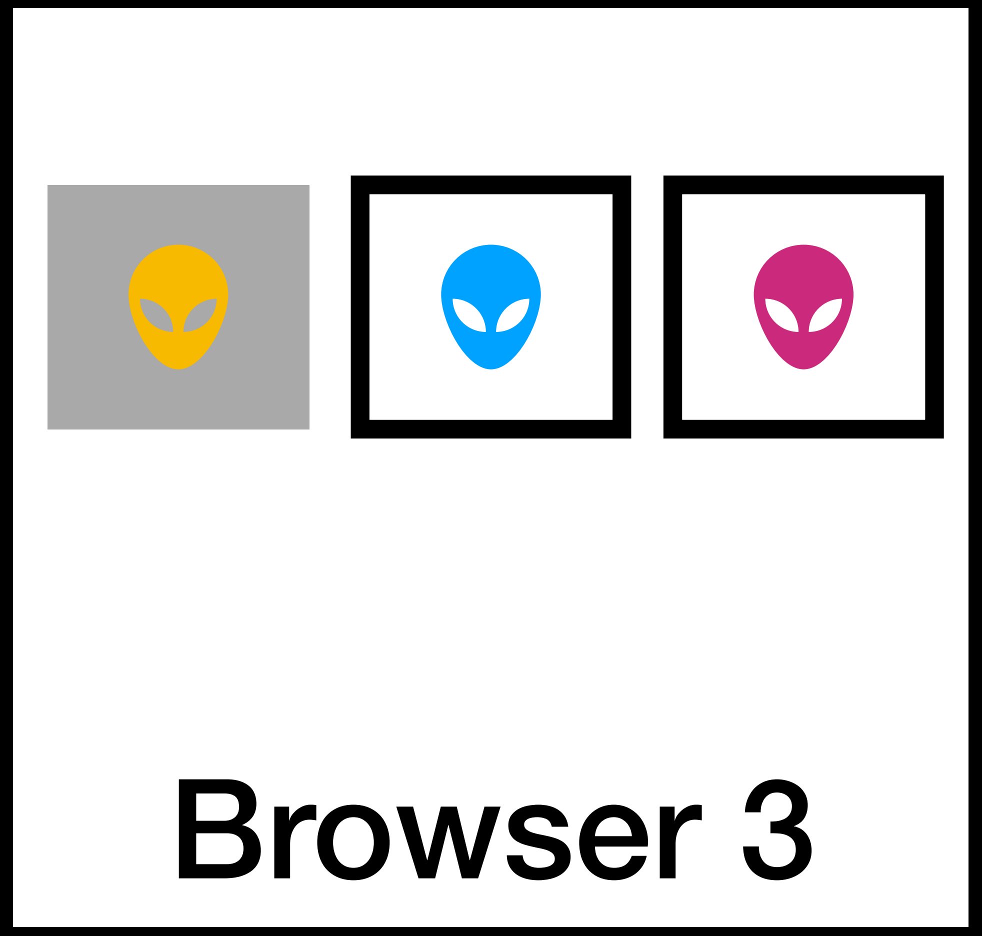
Problems solved with Elixir

Abrupt browser/tab close

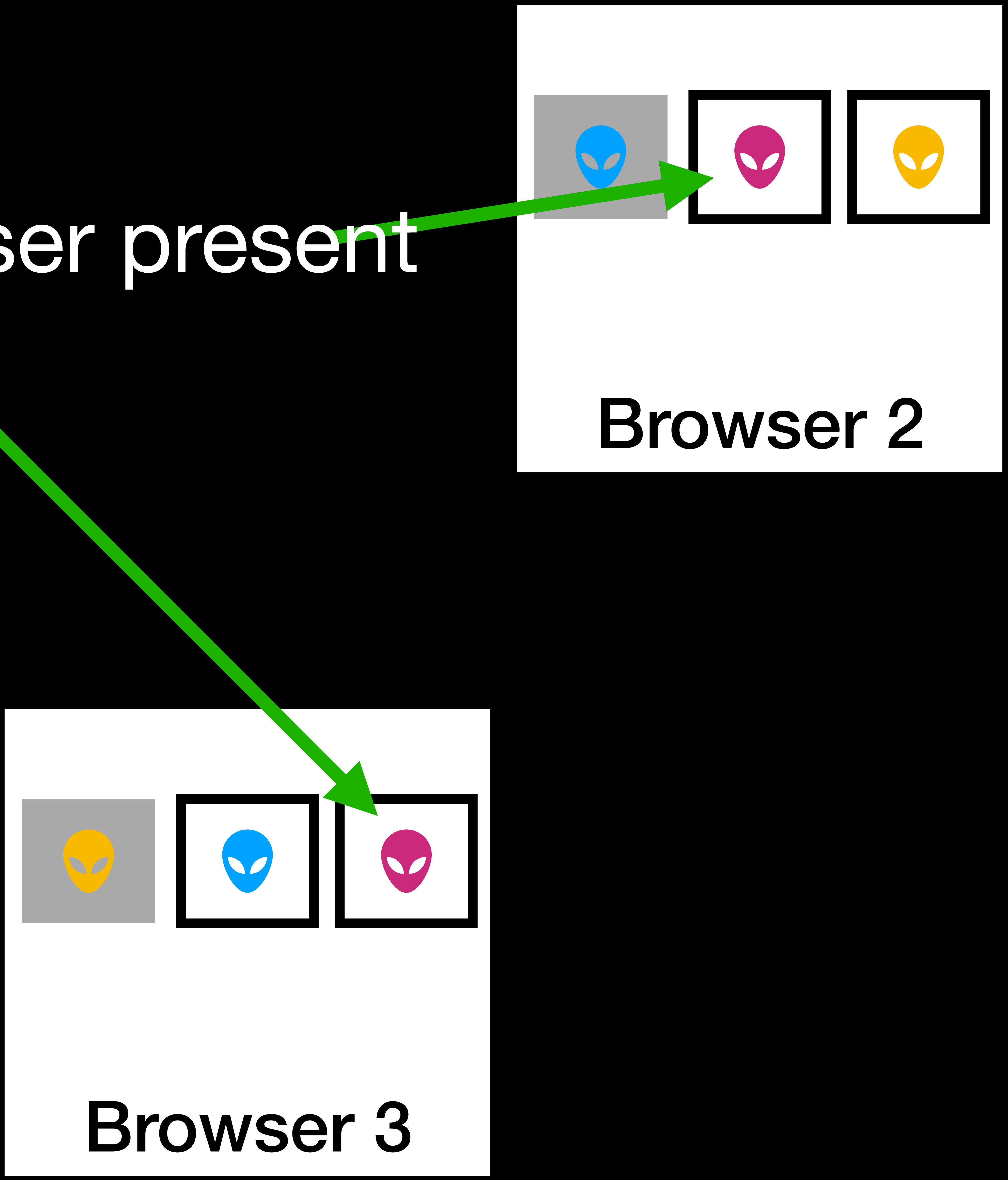

Browser 1
Browser 2
Browser 3


Tab
Closed
owse

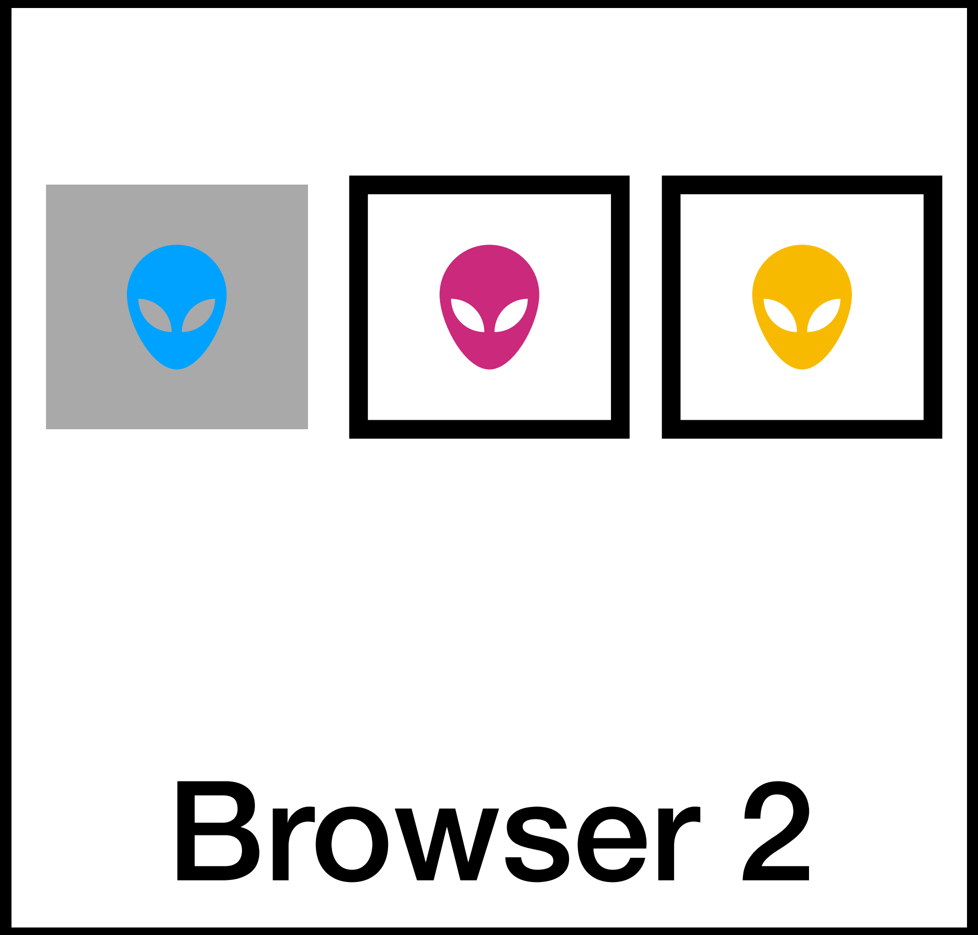

Browser 2

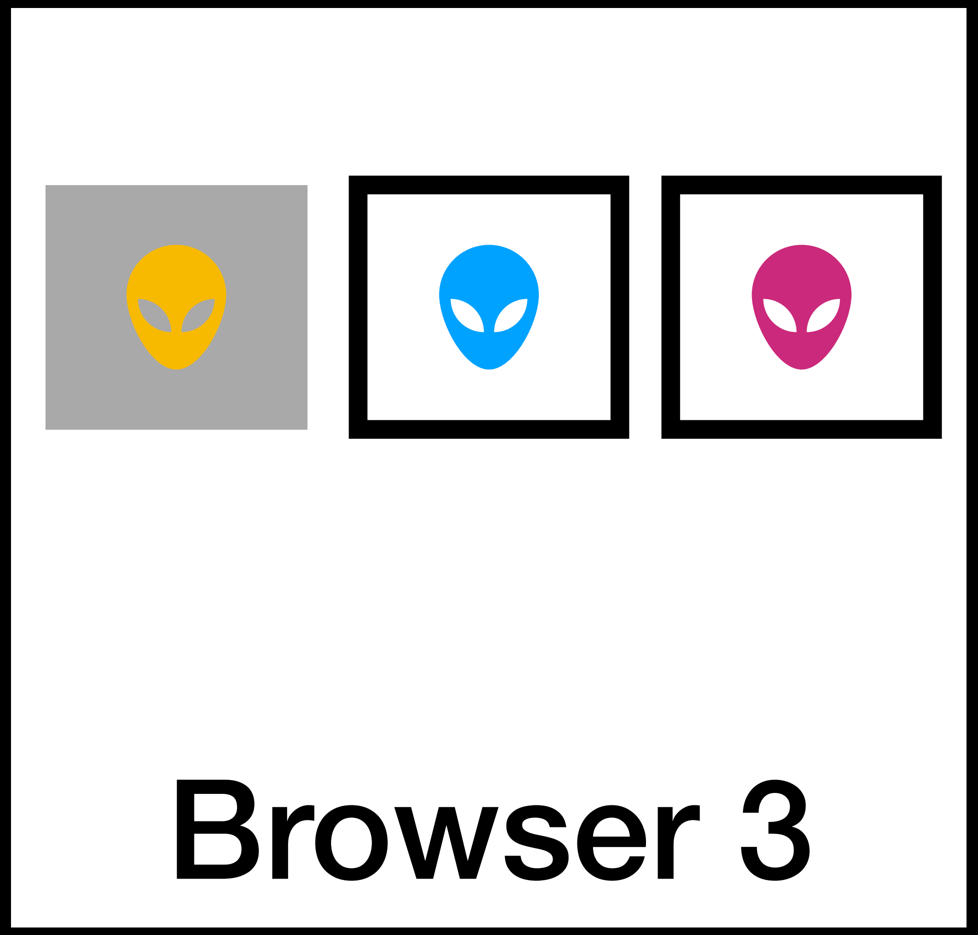

Browser 3

Others still see the user present

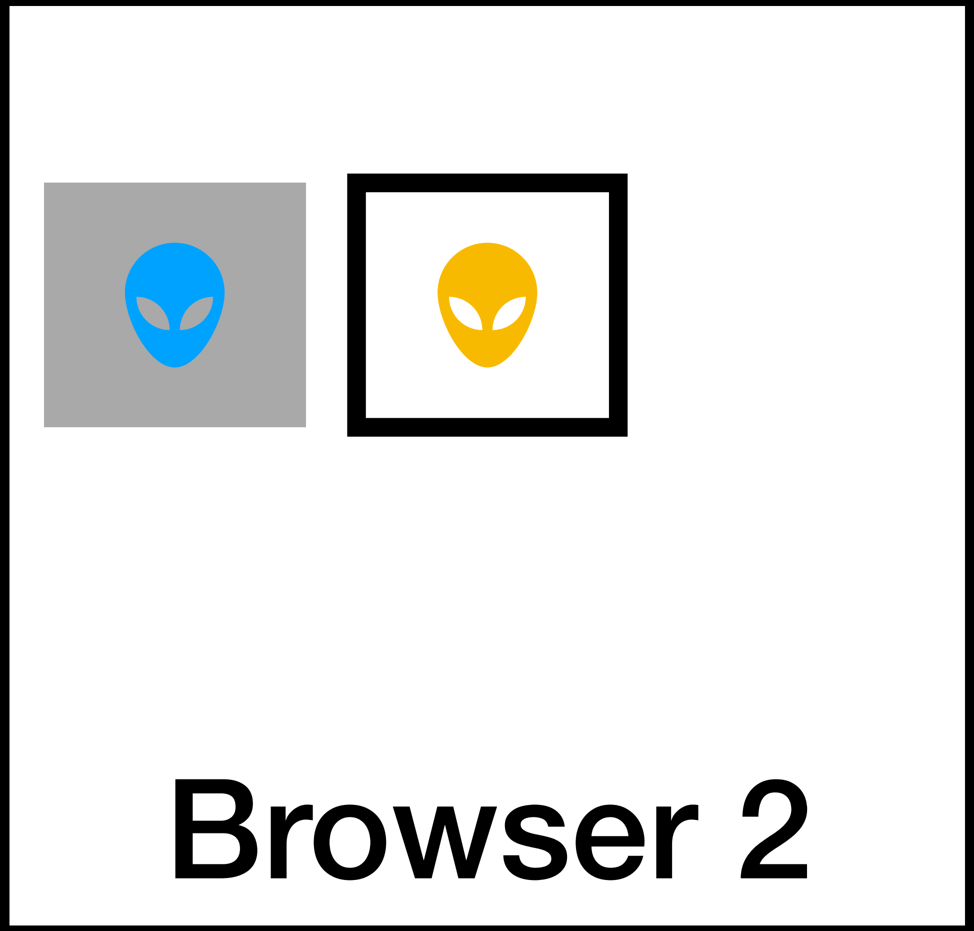


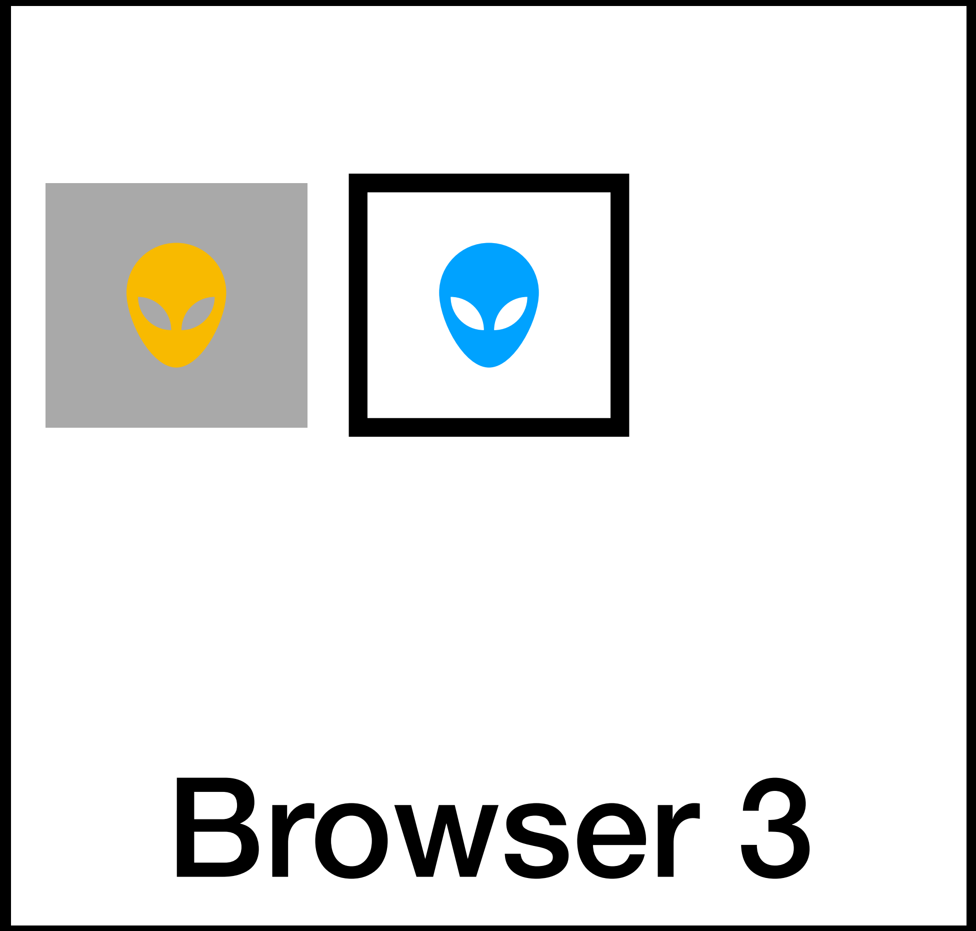
Appears there till minute





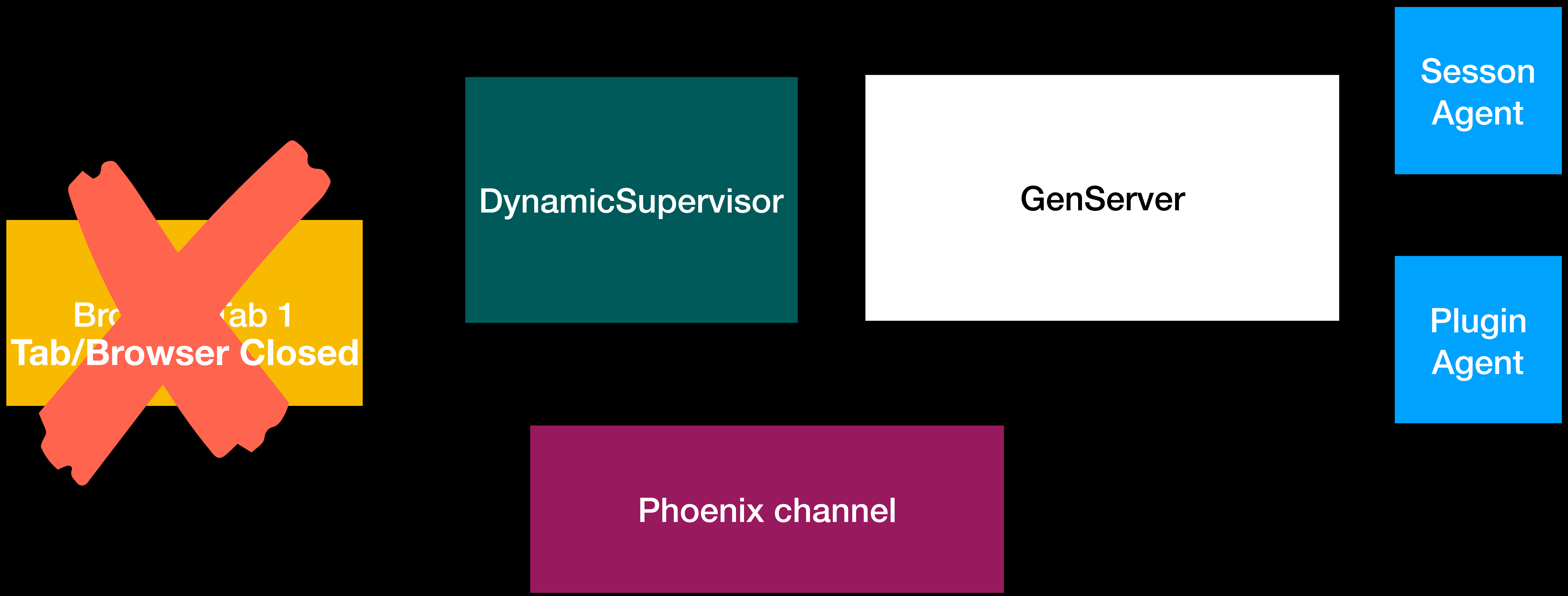
After one minute

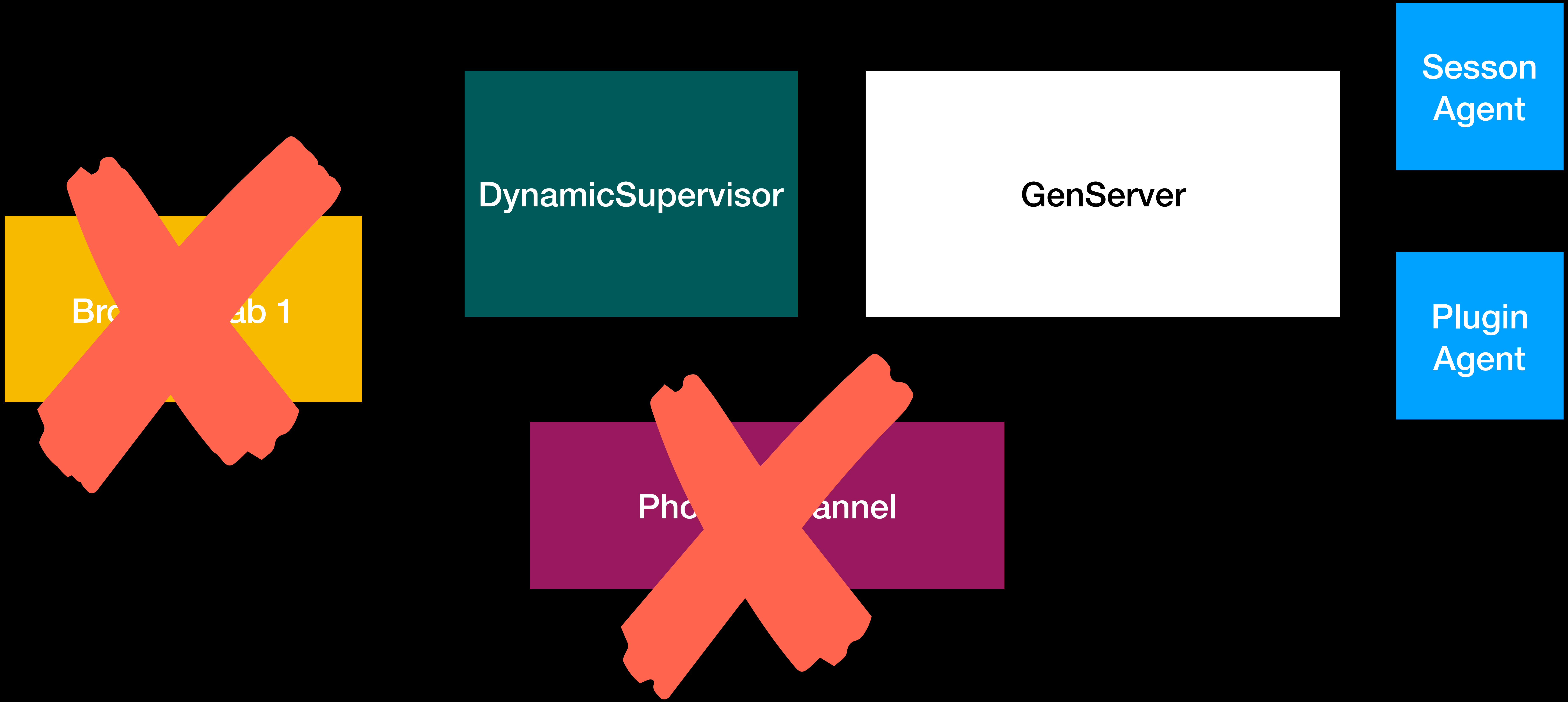


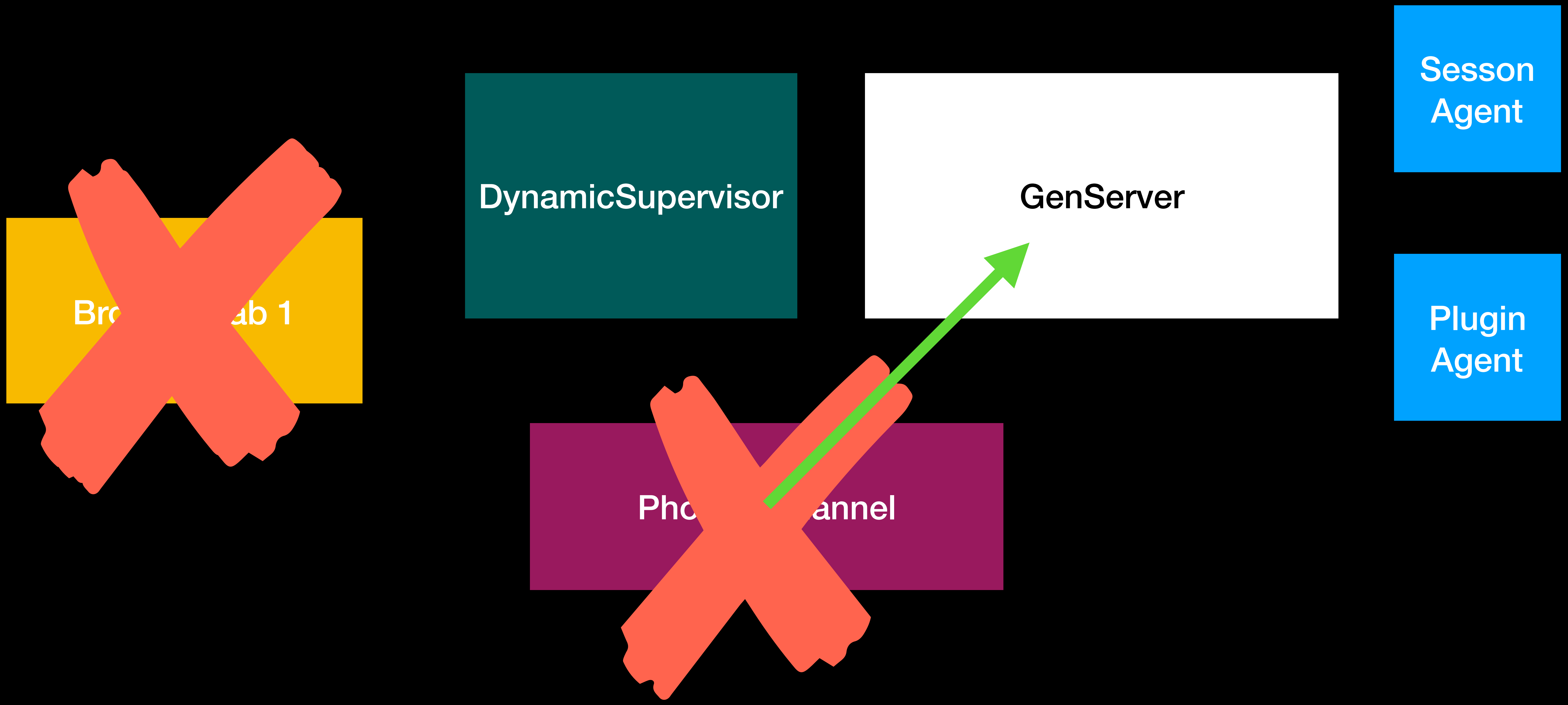


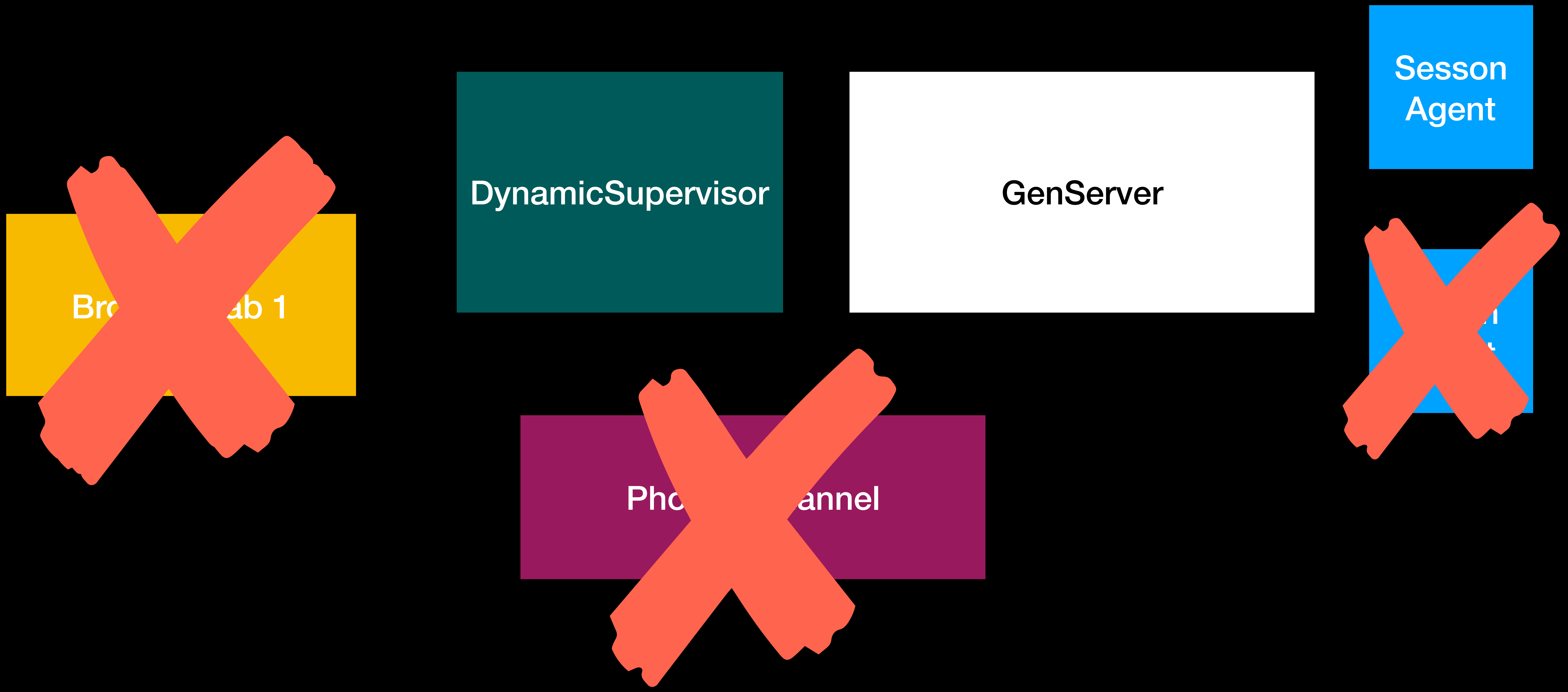
How to solve this problem?

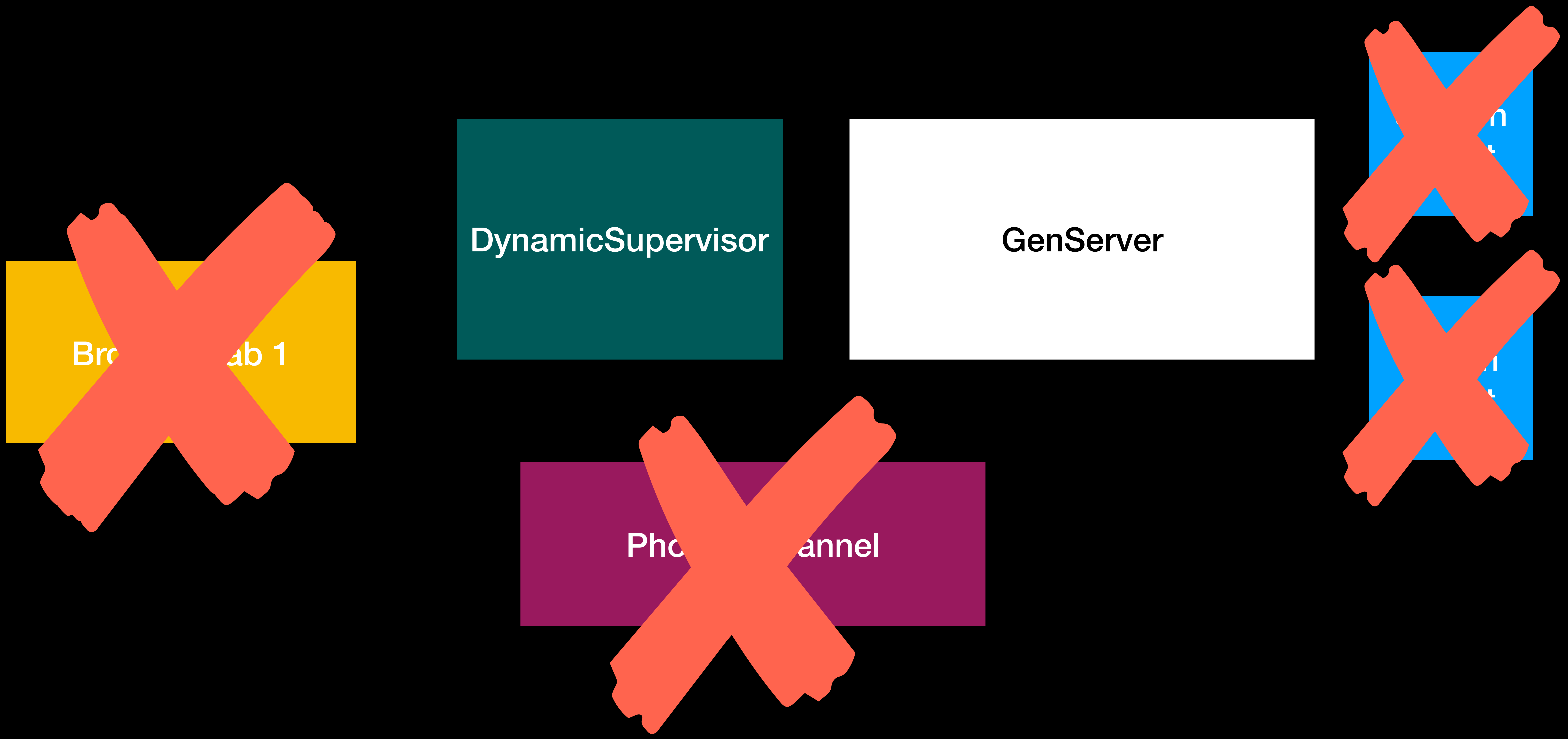
DynamicSupervisor, GenServers & Monitors

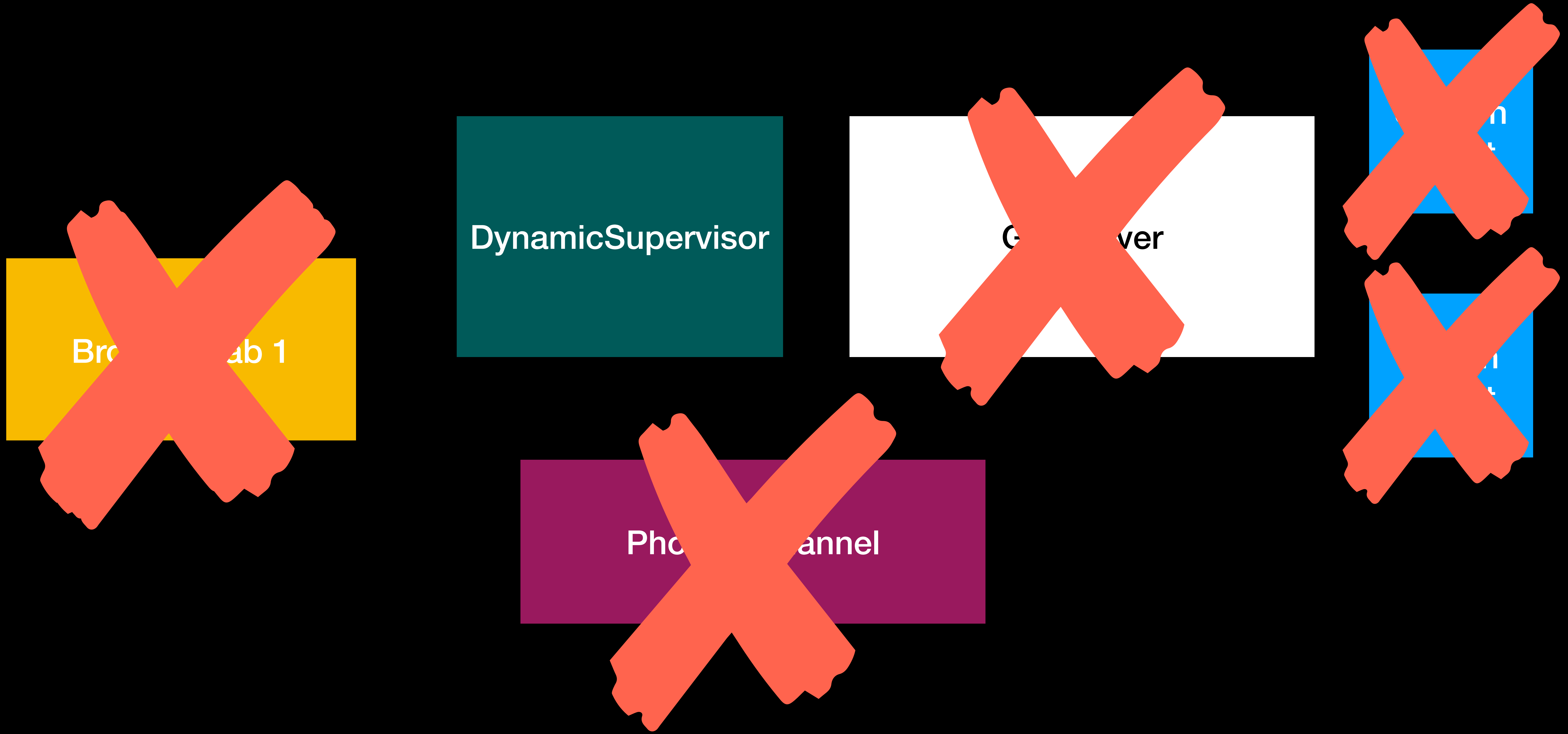

Bro Tab 1
Tab/Browser Closed
DynamicSupervisor
GenServer
Sesson Agent
Plugin Agent
Phoenix channel


Bro
ab 1
DynamicSupervisor
GenServer
Sesson Agent
Plugin Agent
Pho
annel


Bro ab 1
DynamicSupervisor
GenServer
Sesson Agent
Plugin Agent
Pho annel


Bro ab 1
DynamicSupervisor
GenServer
Sesson Agent
Pho annel


Br ab 1
DynamicSupervisor
GenServer
Pho annel


Bro ab 1
DynamicSupervisor
G ver
Pho annel

```elixir
defmodule Janus.Session.GenServer do
  use GenServer

  def init(state) do
    %{url: _url, session: _session, handle: _handle, channel_pid:
channel_pid} = state
    Process.monitor(channel_pid)
    send(self(), :setup)
    {:ok, state}
  end
```

```
defmodule Janus.Session.GenServer do
  use GenServer

  def init(state) do
    %{url: _url, session: _session, handle: _handle, channel_pid:
channel_pid} = state
    Process.monitor(channel_pid)
    send(self(), :setup)
    {:ok, state}
  end
```

```elixir
defmodule Janus.Session.GenServer do
  use GenServer
  …

  def handle_info({:DOWN, ref, :process, other_pid, _reason}, state) do
    %{url: _url, session: session, handle: handle, channel_pid:
_channel_pid} = state
    cleanup(state)
    {:noreply, state}
  end
```

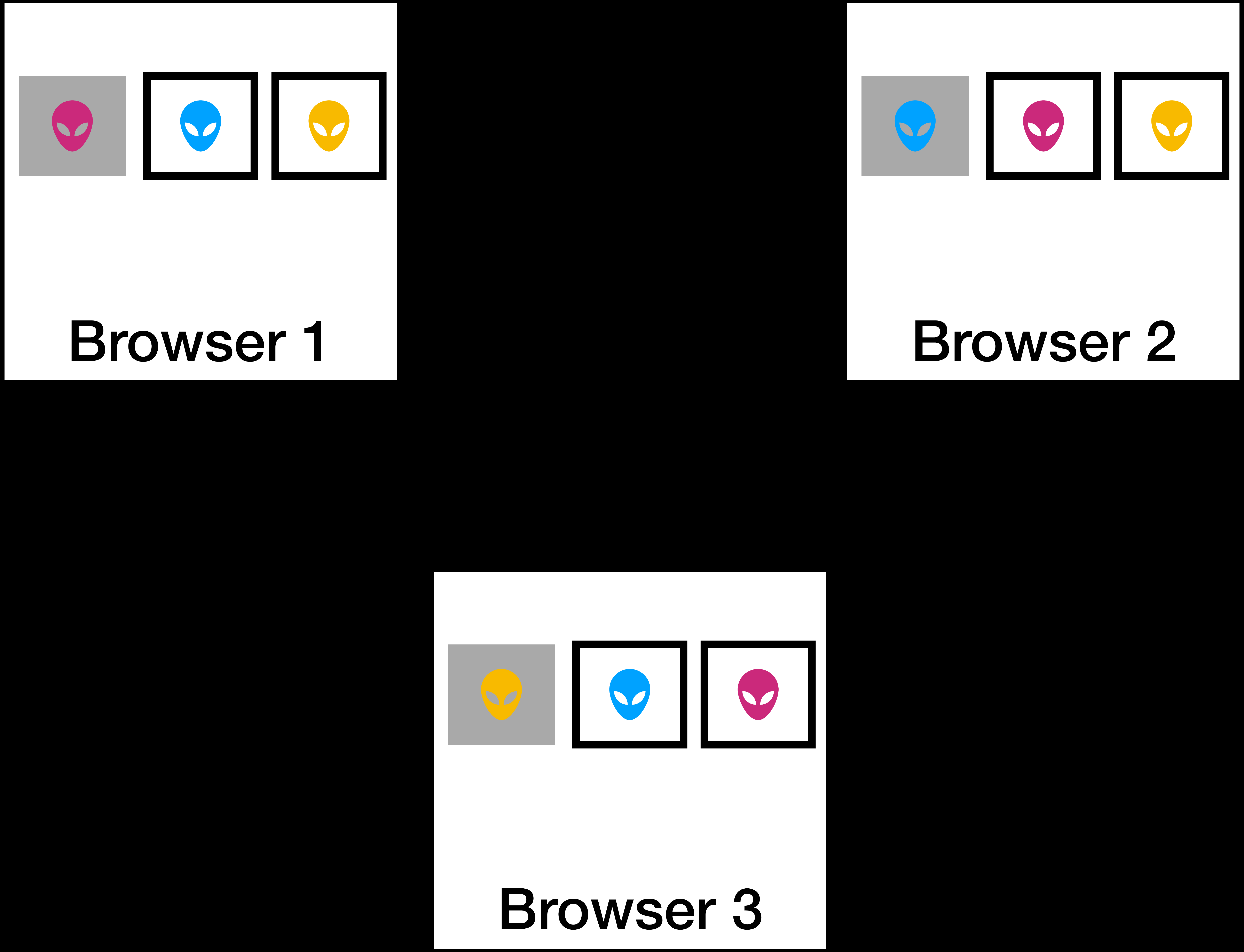
```
defmodule Janus.Session.GenServer do
  use GenServer
  …

  def handle_info({:DOWN, ref, :process, other_pid, _reason}, state) do
    %{url: _url, session: session, handle: handle, channel_pid:
_channel_pid} = state
    cleanup(state)
    {:noreply, state}
  end

  def cleanup(state) do
    %{url: _url, session: session, handle: handle, channel_pid:
_channel_pid} = state
    VideoroomCall.stop(session, handle)
    Process.exit(self(), :kill)
    state
  end
```

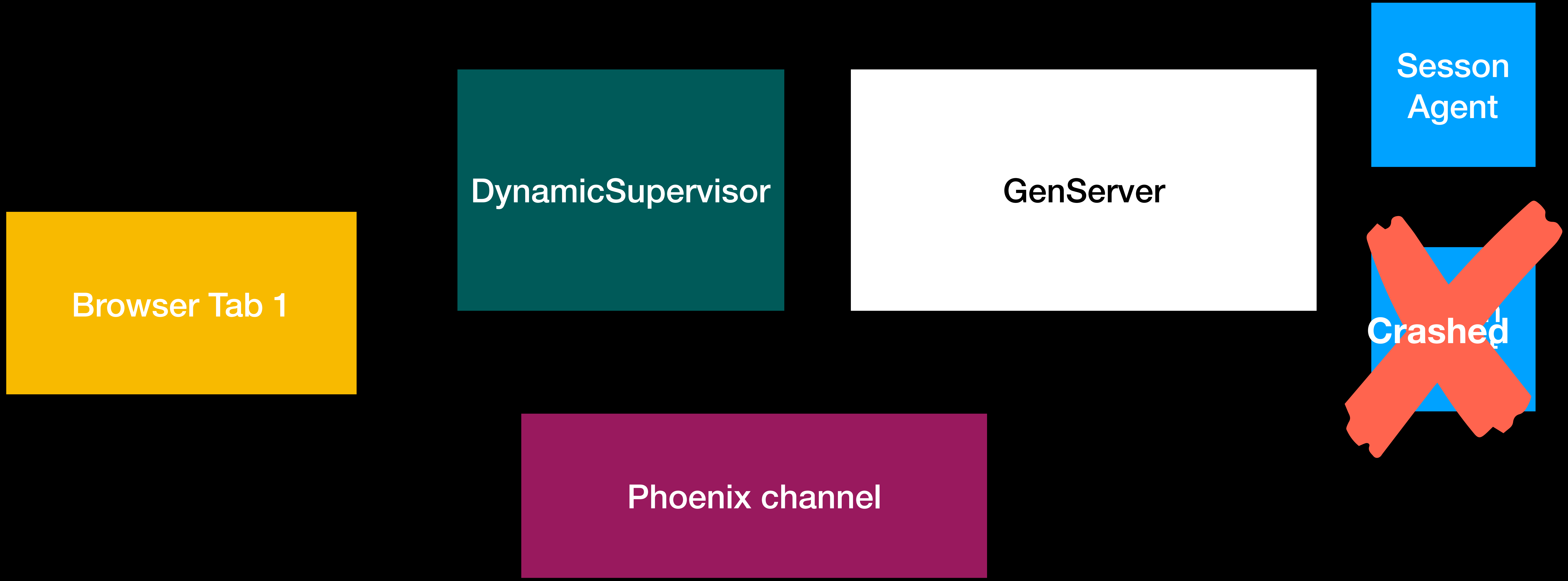
Demo

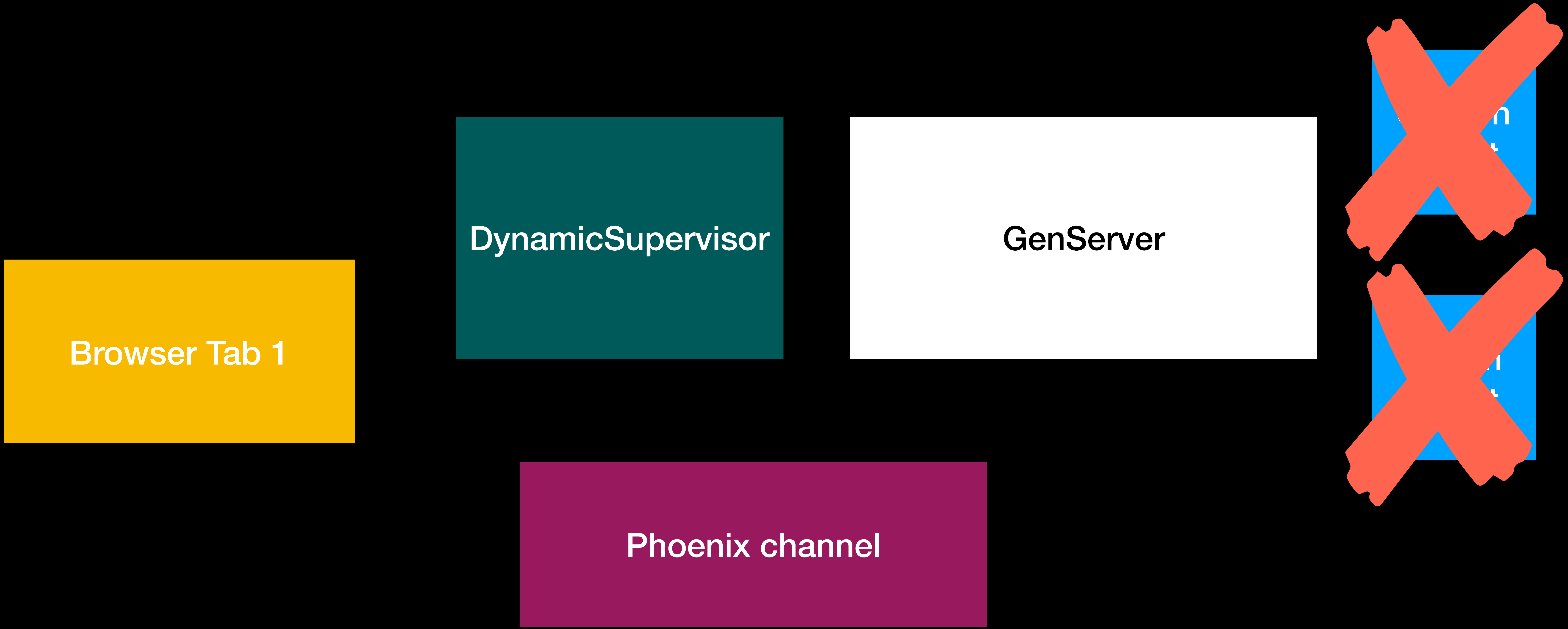
What if Agent crashes?

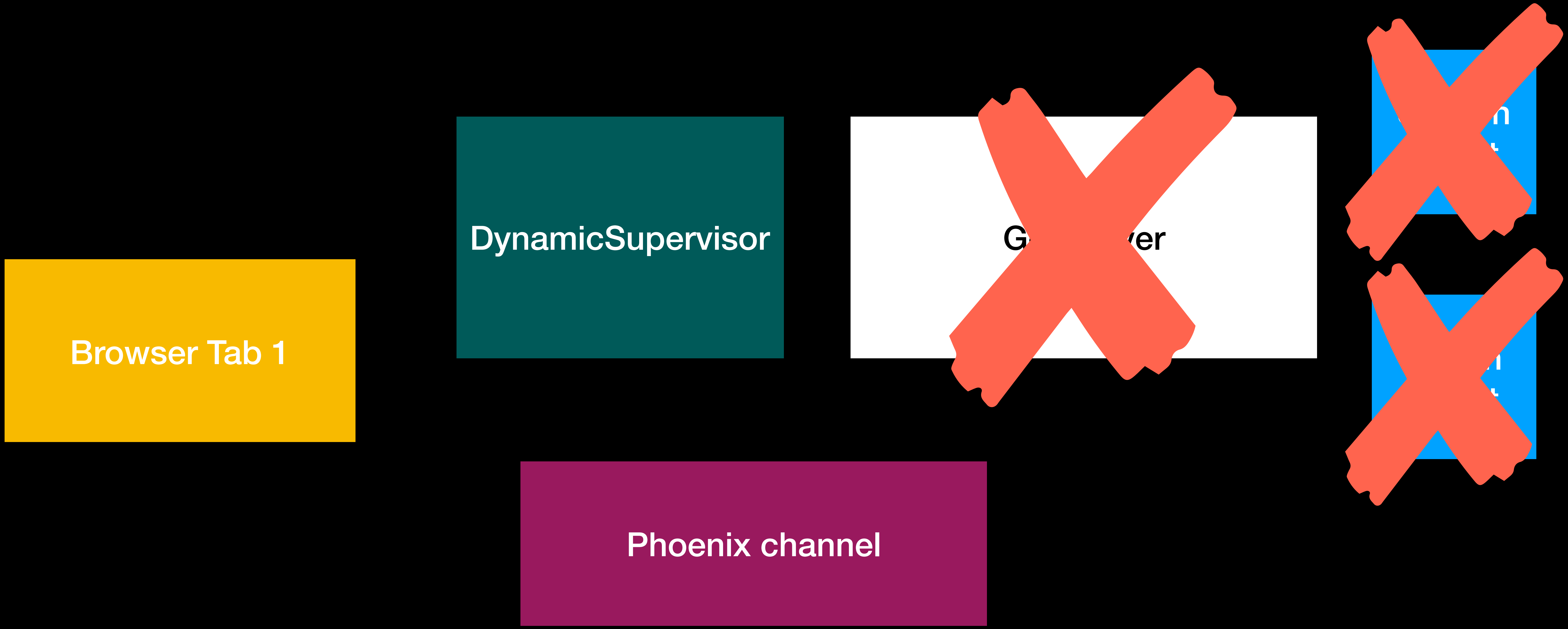
Agents can crash because

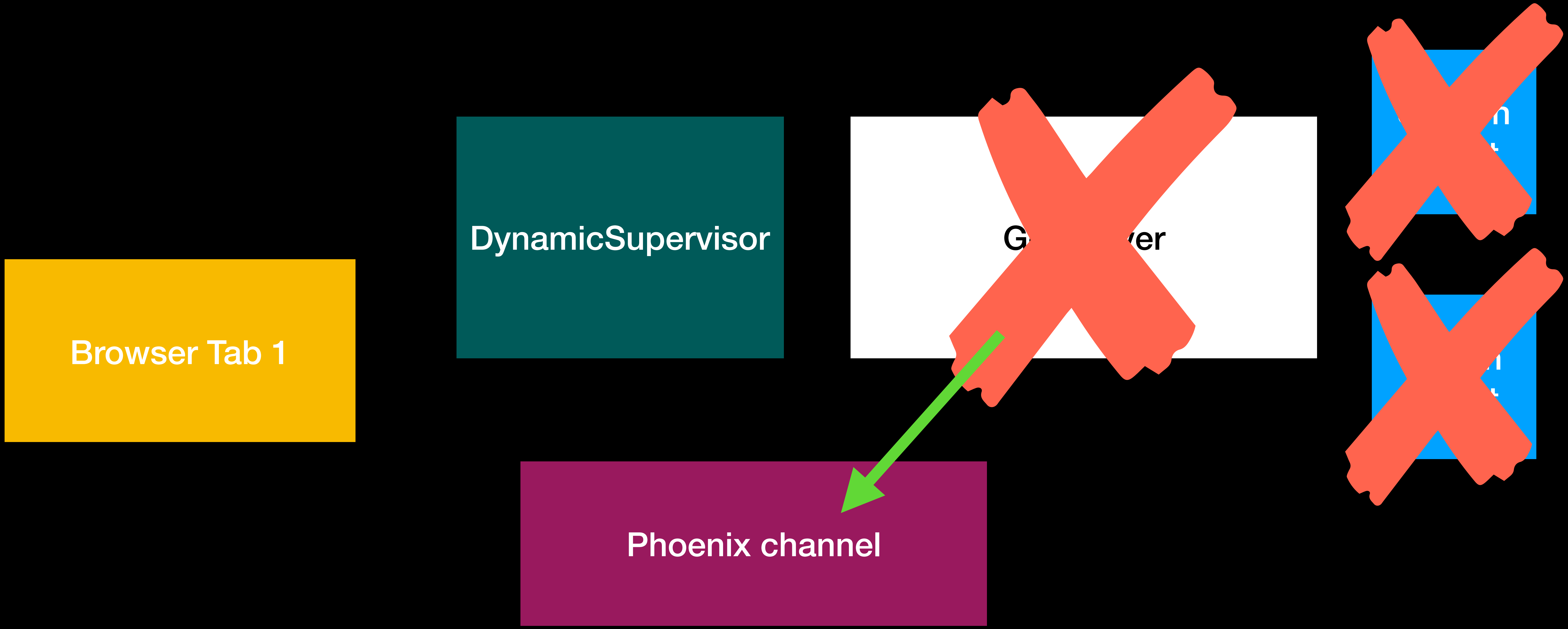
- HTTP call fails
- Exception in Event Handler code


Browser 1
Browser 2
Browser 3

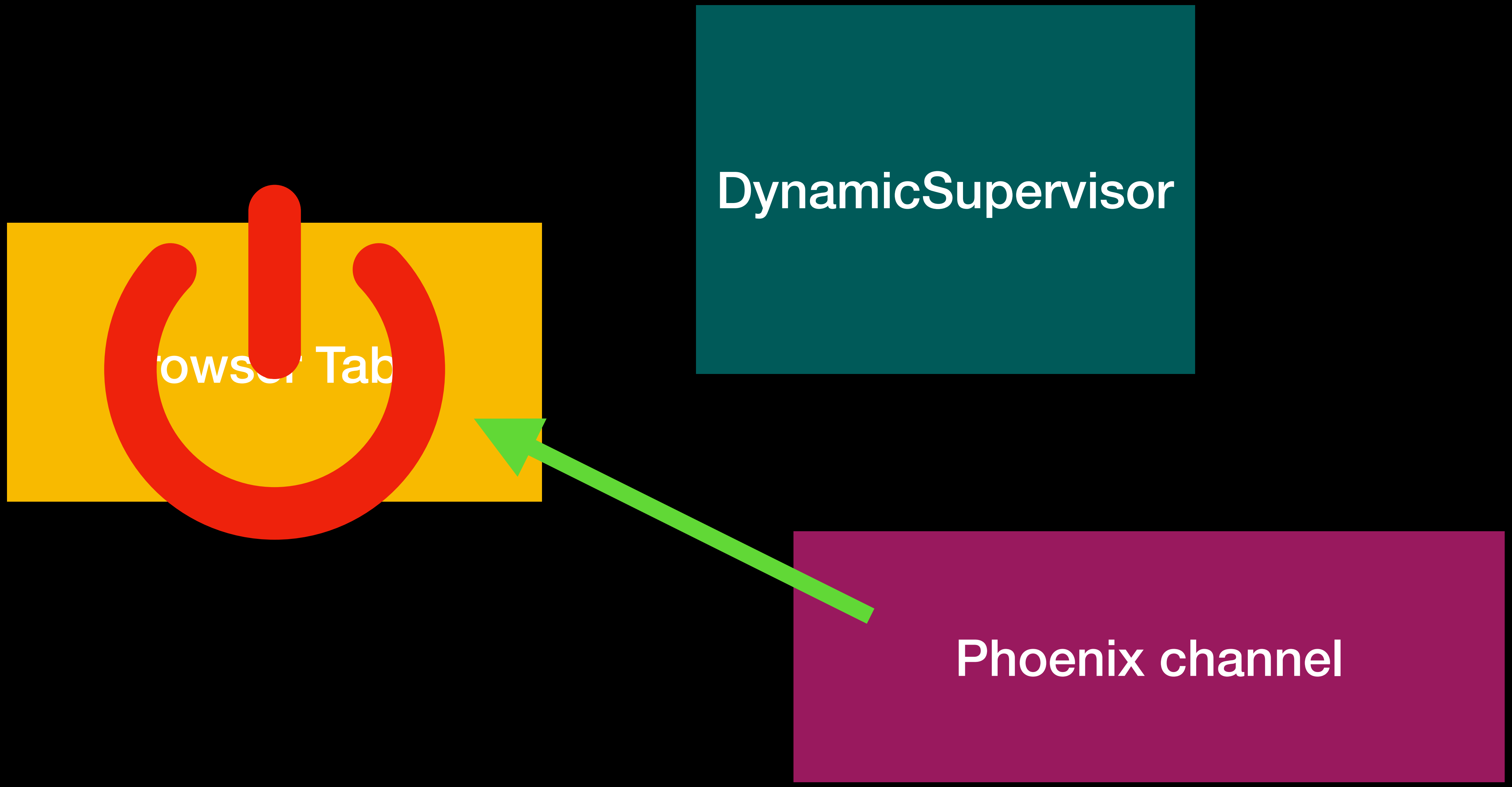

Browser Tab 1
DynamicSupervisor
GenServer
Sesson Agent
Crashed
Phoenix channel


Browser Tab 1
DynamicSupervisor
GenServer
Phoenix channel

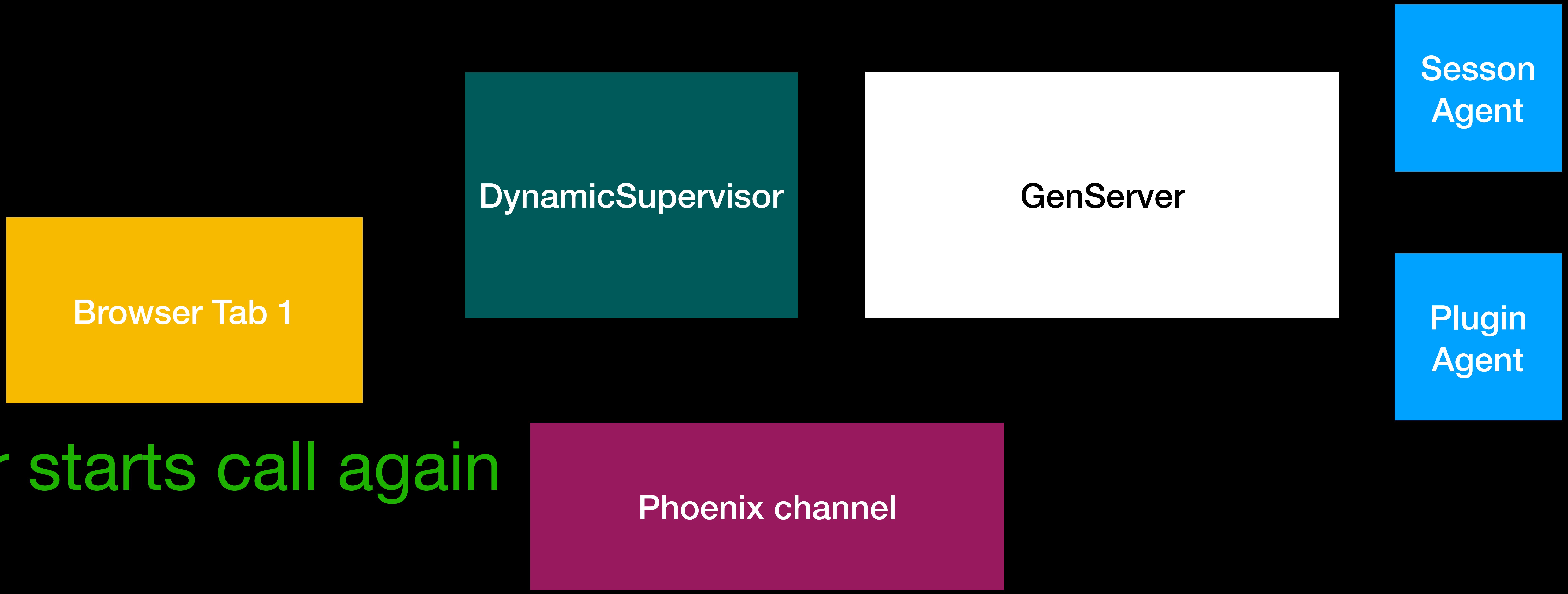

Browser Tab 1
DynamicSupervisor
G ver
Phoenix channel



Phoenix channel receives DOWN message



Send JavaScript ‘Stop Call’ message



User starts call again

```elixir
defmodule SkatterWeb.SkatterRoomChannel do
  use SkatterWeb, :channel
  …
  defp init_janus_room(jsep, room_id, recording) do
    room = Rooms.find_by_id(room_id)
```

```
defmodule SkatterWeb.SkatterRoomChannel do
  use SkatterWeb, :channel
  …
  defp init_janus_room(jsep, room_id, recording) do
    room = Rooms.find_by_id(room_id)

    {:ok, session_server} =
DynamicSupervisor.start_child(Janus.Supervisor,
    Janus.Session.GenServer.child_spec([{:channel_pid,
self()}]))

    ref = Process.monitor(session_server)
```

```
defmodule SkatterWeb.SkatterRoomChannel do
  use SkatterWeb, :channel
  …
  defp init_janus_room(jsep, room_id, recording) do
    room = Rooms.find_by_id(room_id)

    {:ok, session_server} =
DynamicSupervisor.start_child(Janus.Supervisor,
    Janus.Session.GenServer.child_spec([{:channel_pid,
self()}]))

    ref = Process.monitor(session_server)  <-----------

   {session, plugin_pid} =
Janus.Session.GenServer.start_session(session_server,
room_name)
  …
```

```elixir
defmodule SkatterWeb.SkatterRoomChannel do
  use SkatterWeb, :channel
```

```elixir
defmodule SkatterWeb.SkatterRoomChannel do
  use SkatterWeb, :channel
  …
  def handle_info({:DOWN, ref, :process, _pid, _reason}, socket) do
```

```elixir
defmodule SkatterWeb.SkatterRoomChannel do
  use SkatterWeb, :channel
  …
  def handle_info({:DOWN, ref, :process, _pid, _reason}, socket) do

    room_name = get_room_name(socket)

    SkatterWeb.Endpoint.broadcast(room_name, "data", %{
          type: "stop_call"})

    {:noreply, socket}
  end
  …
```

Demo

Benefits of using Elixir

- Useful abstractions
- Control
- Robust
- Clarity

Thank you!

Questions?

@anildigital

anil@anilwadghule.com

https://skatter.me