Sitos: Proyecto BitTorrent

75.42 / 95.08 Taller de Programación I
1C 2022 - Ing. Pablo Deymonnaz

¿Quienes Somos?

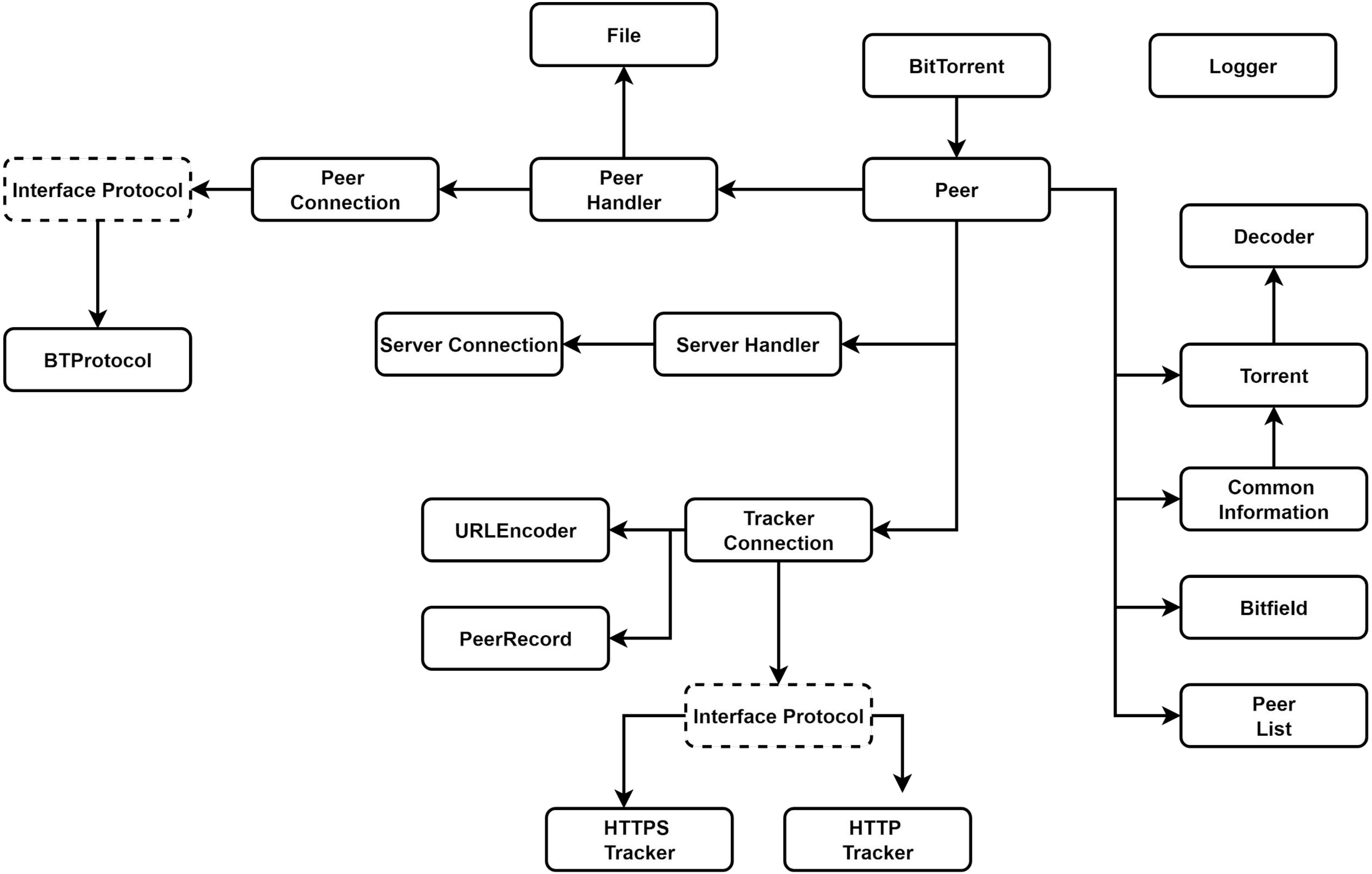
Lautaro
Goijman

Sofia
Carbon Posse

Martin
Veiga

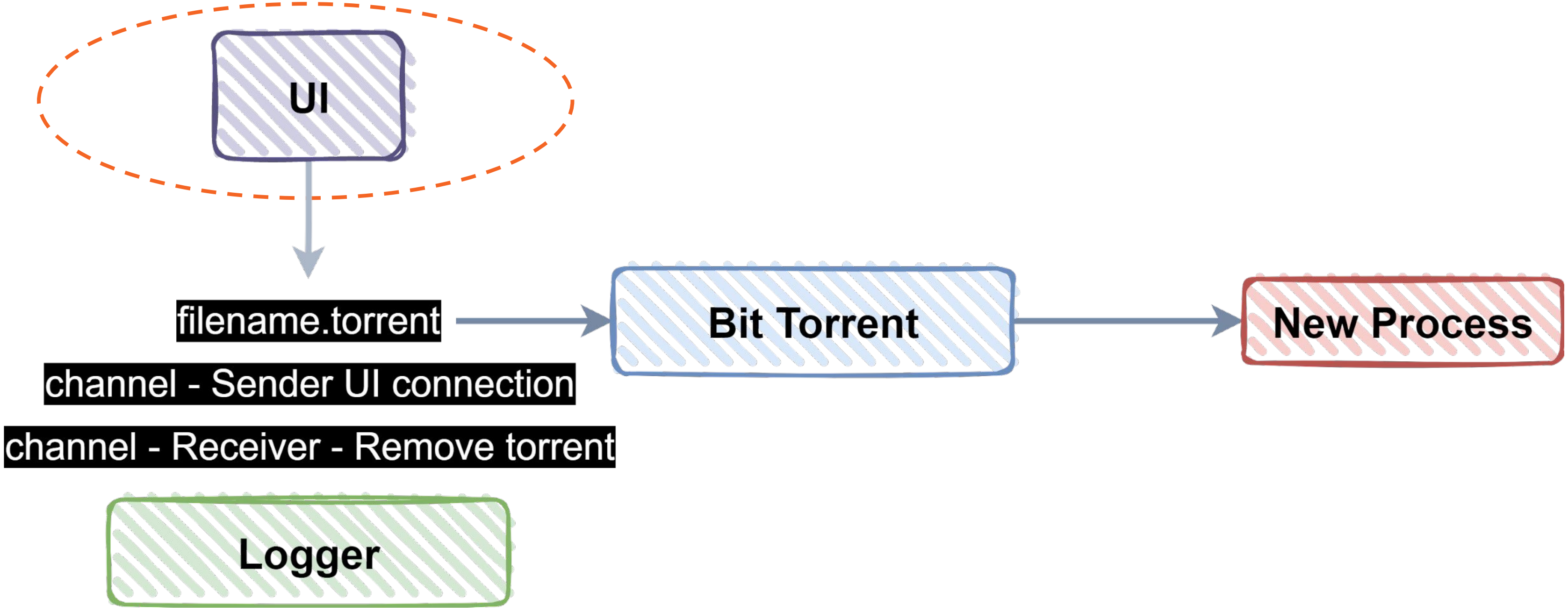
Julian
Garcia

Arquitectura del Programa

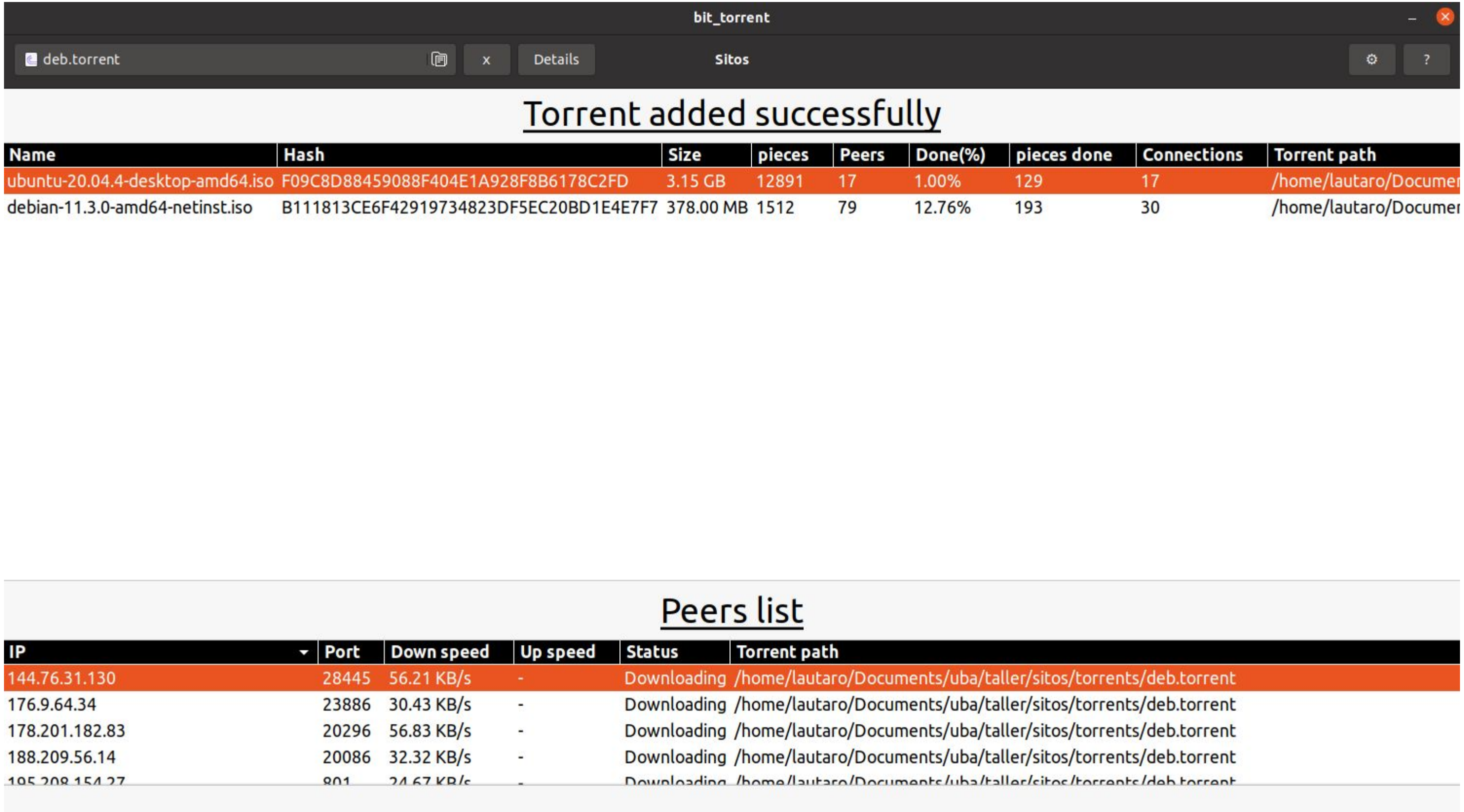

File
BitTorrent
Logger
Interface Protocol
Peer Connection
Peer Handler
Peer
Decoder
BTProtocol
Server Connection
Server Handler
Torrent
Common Information
URLEncoder
Tracker Connection
PeerRecord
Bitfield
Interface Protocol
Peer List
HTTPS Tracker
HTTP Tracker

¿Sobre qué estructuras pasa el flujo de programa cuando queremos descargar un archivo?

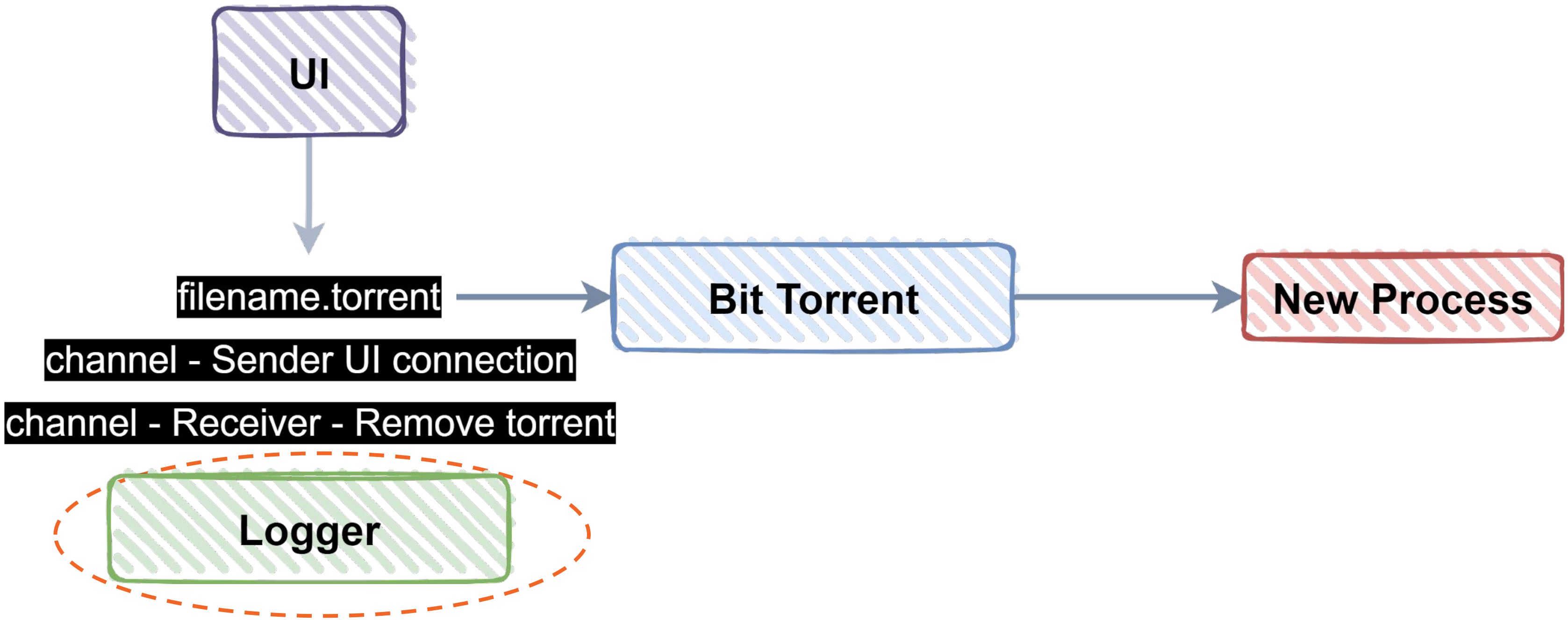
Flujo


UI
filename.torrent
channel - Sender UI connection
channel - Receiver - Remove torrent
Bit Torrent
New Process
Logger

UI


bit_torrent
deb.torrent
x
Details
Sitos
?
Torrent added successfully
Name | Hash | Size | pieces | Peers | Done(%) | pieces done | Connections | Torrent path
ubuntu-20.04.4-desktop-amd64.iso | F09C8D88459088F404E1A928F8B6178C2FD | 3.15 GB | 12891 | 17 | 1.00% | 129 | 17 | /home/lautaro/Documen
debian-11.3.0-amd64-netinst.iso | B111813CE6F42919734823DF5EC20BD1E4E7F7 | 378.00 MB | 1512 | 79 | 12.76% | 193 | 30 | /home/lautaro/Documen
Peers list
IP | Port | Down speed | Up speed | Status | Torrent path
144.76.31.130 | 28445 | 56.21 KB/s | - | Downloading | /home/lautaro/Documents/uba/taller/sitos/torrents/deb.torrent
176.9.64.34 | 23886 | 30.43 KB/s | - | Downloading | /home/lautaro/Documents/uba/taller/sitos/torrents/deb.torrent
178.201.182.83 | 20296 | 56.83 KB/s | - | Downloading | /home/lautaro/Documents/uba/taller/sitos/torrents/deb.torrent
188.209.56.14 | 20086 | 32.32 KB/s | - | Downloading | /home/lautaro/Documents/uba/taller/sitos/torrents/deb.torrent

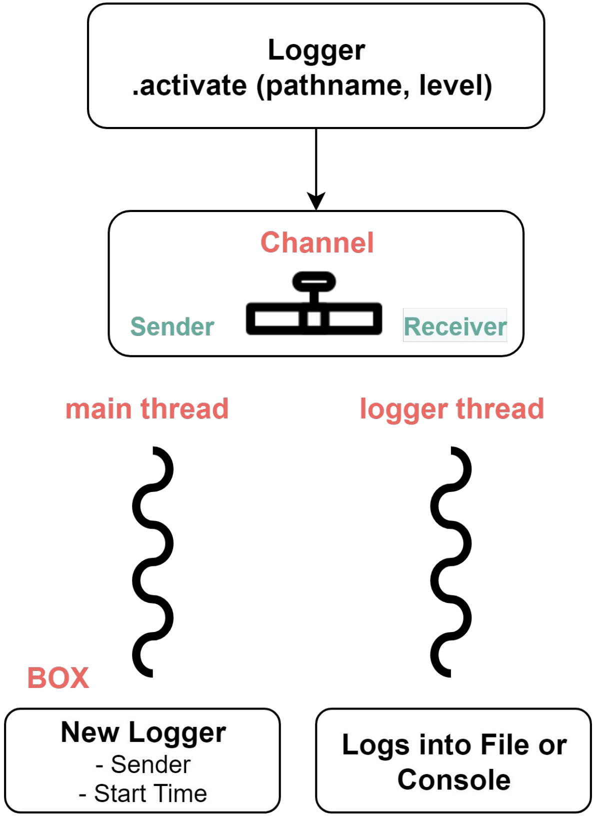
Flujo


UI
filename.torrent
channel - Sender UI connection
channel - Receiver - Remove torrent
Logger
Bit Torrent
New Process

Logger

- Es un **SINGLETON:** una única instancia puede ser creada.
- Loggea por consola o en un archivo específico.
- *.activate* (pathname, level)

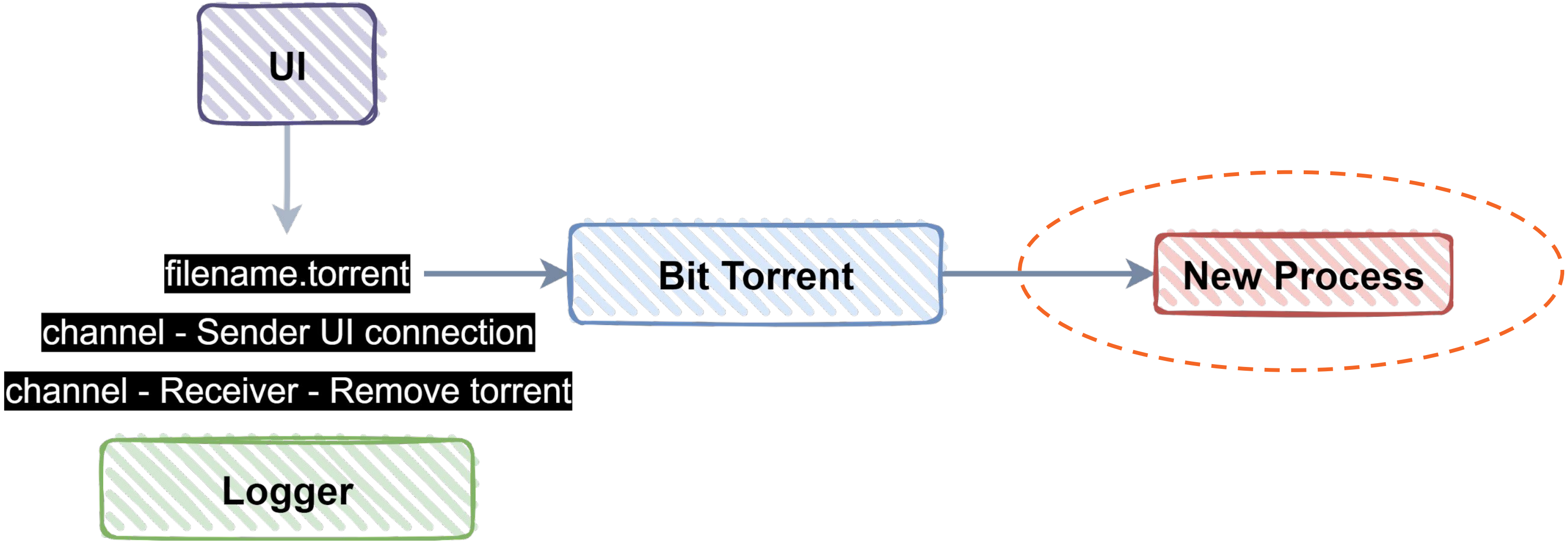
Logger



Formato con el que logea:

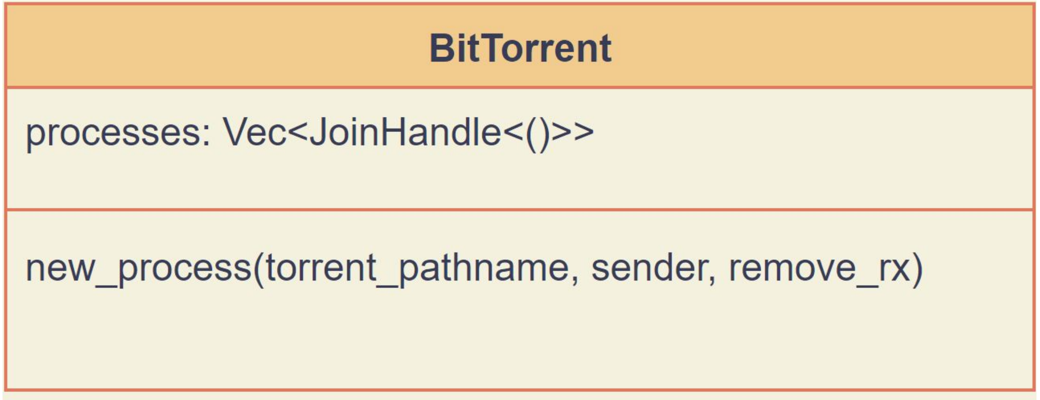
```
[00:00:00.000] ERROR  [sitos::logger::index::tests:167] Ups! This is an Error
[00:00:01.000] WARN   [sitos::logger::index::tests:169] This is a Warn
[00:00:02.004] INFO   [sitos::logger::index::tests:171] Info available here
[00:00:03.005] DEBUG  [sitos::logger::index::tests:173] Debug mode
[00:00:04.005] TRACE  [sitos::logger::index::tests:175] Oh no! Trace!
```

Flujo


UI
filename.torrent
channel - Sender UI connection
channel - Receiver - Remove torrent
Bit Torrent
New Process
Logger

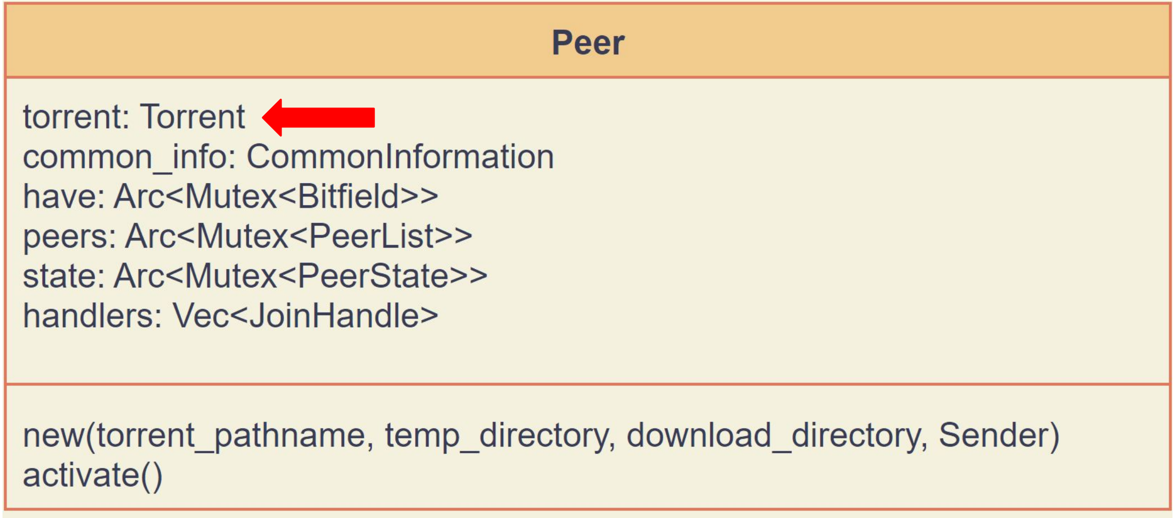
– BitTorrent

- Contiene un vector con **JOIN HANDLE** que permite tener control de los threads.
- Pushea al vector un Peer activado.

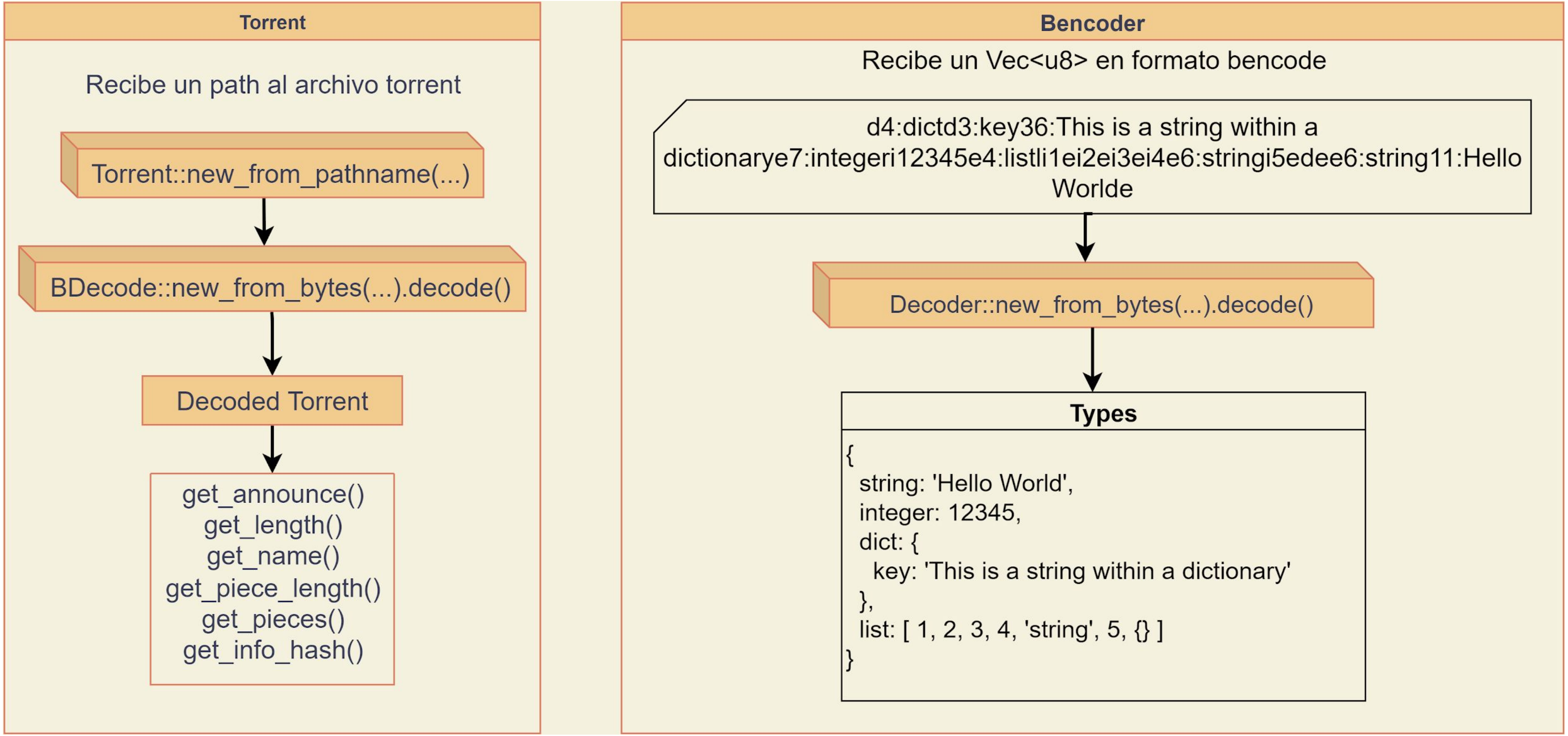


Veamos en detalle el Peer...

Peer


Peer
torrent: Torrent
common_info: CommonInformation
have: Arc<Mutex<Bitfield>>
peers: Arc<Mutex<PeerList>>
state: Arc<Mutex<PeerState>>
handlers: Vec<JoinHandle>
new(torrent_pathname, temp_directory, download_directory, Sender)
activate()

Torrent


Torrent
Recibe un path al archivo torrent
Torrent::new_from_pathname(...)
BDecode::new_from_bytes(...).decode()
Decoded Torrent
get_announce()
get_length()
get_name()
get_piece_length()
get_pieces()
get_info_hash()
Bencoder
Recibe un Vec<u8> en formato bencode
d4:dictd3:key36:This is a string within a dictionarye7:integeri12345e4:listli1ei2ei3ei4e6:stringi5edee6:string11:Hello Worlde
Decoder::new_from_bytes(...).decode()
Types
{
string: 'Hello World',
integer: 12345,
dict: {
key: 'This is a string within a dictionary'
},
list: [1, 2, 3, 4, 'string', 5, {}]
}

Peer

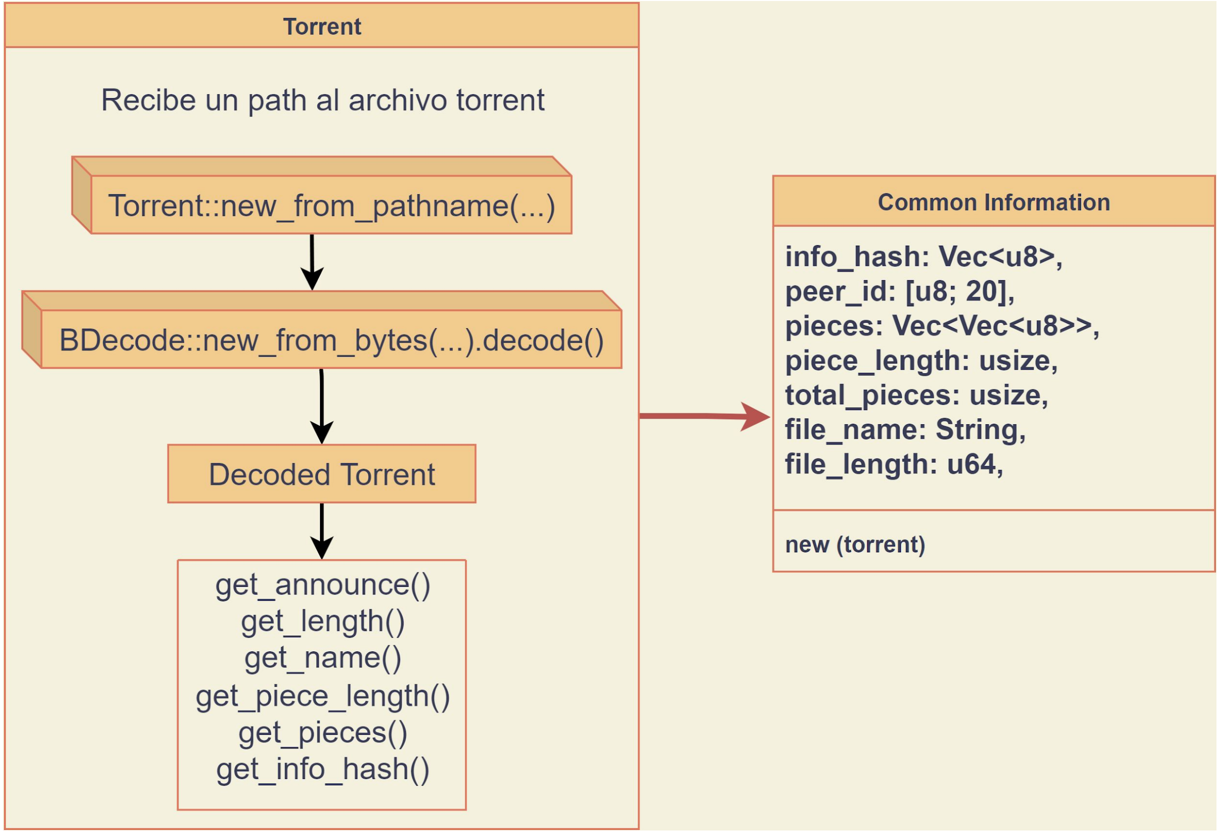
Peer

torrent: Torrent
common_info: CommonInformation
have: Arc<Mutex<Bitfield>>
peers: Arc<Mutex<PeerList>>
state: Arc<Mutex<PeerState>>
handlers: Vec<JoinHandle>

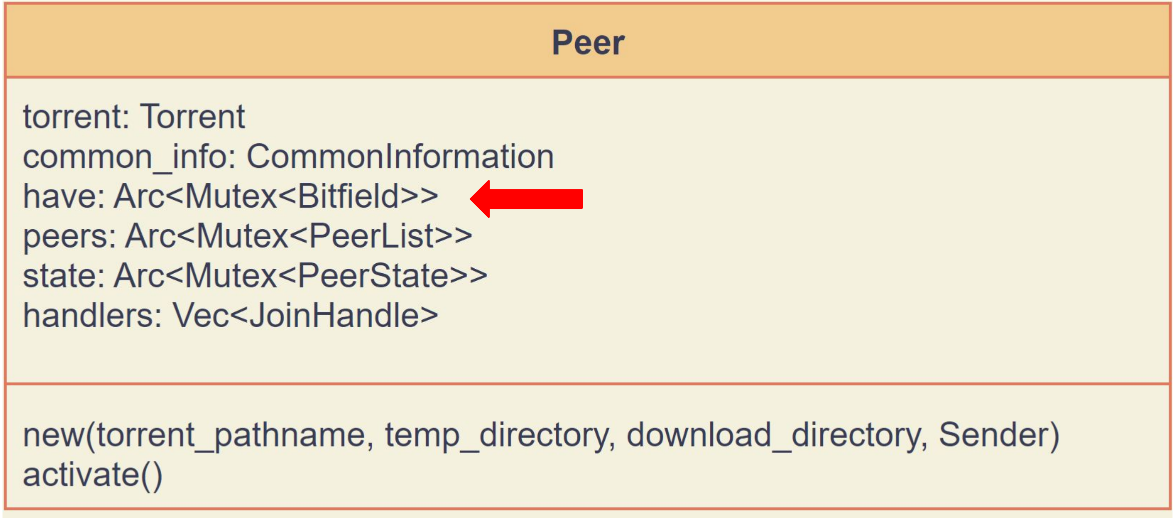
new(torrent_pathname, temp_directory, download_directory, Sender)
activate()

_ Common Information

Contenedor de información que se comparte entre todas las partes del proceso.



Peer


Peer
torrent: Torrent
common_info: CommonInformation
have: Arc<Mutex<Bitfield>>
peers: Arc<Mutex<PeerList>>
state: Arc<Mutex<PeerState>>
handlers: Vec<JoinHandle>
new(torrent_pathname, temp_directory, download_directory, Sender)
activate()

Bitfield

have:

- *1 -> pieces obtenidas*
- *0 -> pieces faltantes*

downloading:

- *1 -> pieces downloading*
- *0 -> pieces not downloading*

*Esta estructura está dentro de un **ARC - MUTEX** debido a que esta información debe ser compartida entre los distintos threads.*

Bitfield

total_pieces: usize,
have: Vec<u8>,
downloading: Vec<u8>,

new (total_pieces)
new_from_vec (have, total_pieces)
is_complete ()
index_from_bytes (bytes)
set (index)
set_downloading (index)
has (index)
is_downloading (index)
status ()
first_needed_available_piece (bitfield)

Peer

Peer

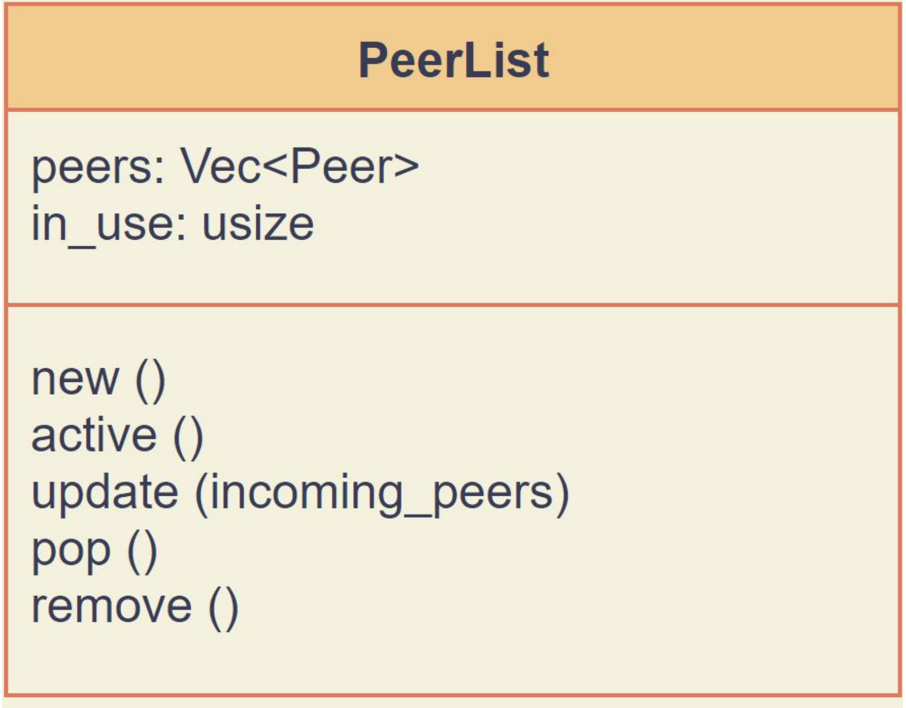
torrent: Torrent
common_info: CommonInformation
have: Arc<Mutex<Bitfield>>
peers: Arc<Mutex<PeerList>>
state: Arc<Mutex<PeerState>>
handlers: Vec<JoinHandle>

new(torrent_pathname, temp_directory, download_directory, Sender)
activate()

– Peer List

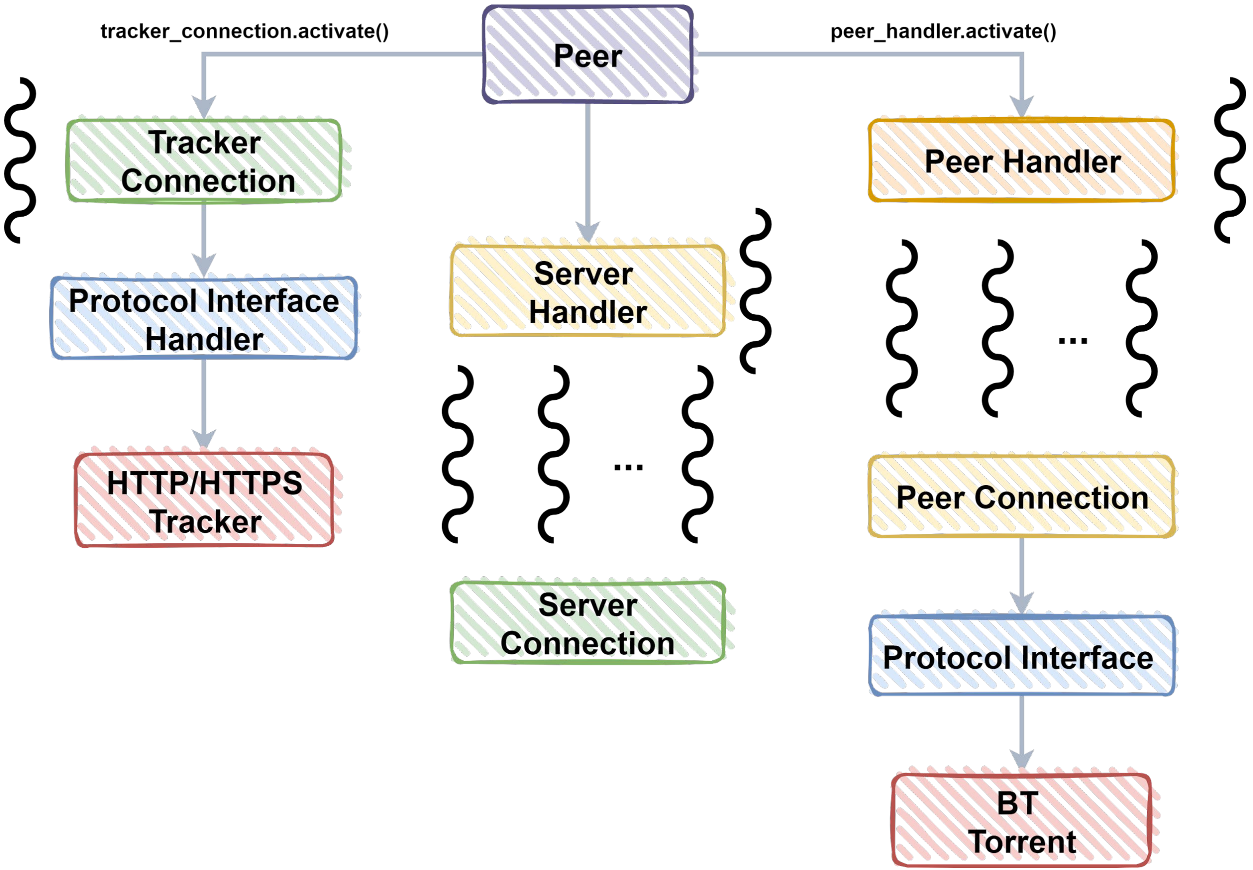
Almacena en un Vector los peers que nos devuelve el Tracker.

Esta estructura está dentro de un ***ARC - MUTEX*** *debido a que esta información debe ser compartida entre los distintos threads.*



¿Qué sucede cuando activamos el Peer?

Flujo



tracker_connection.activate()
Peer
peer_handler.activate()
Tracker Connection
Peer Handler
Protocol Interface Handler
Server Handler
HTTP/HTTPS Tracker
...
...
Peer Connection
Server Connection
Protocol Interface
BT Torrent

Tracker Connection

Realiza la conexión con el tracker

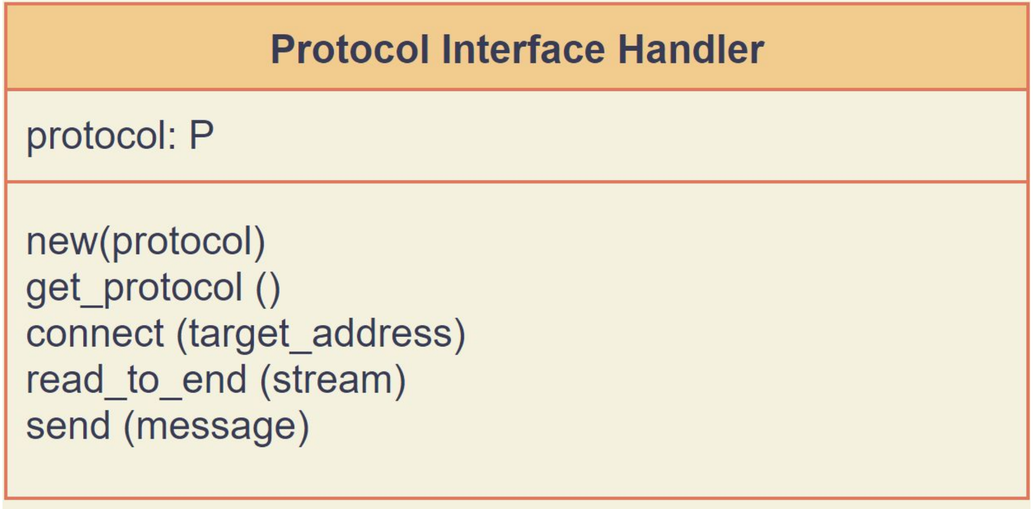
TrackerConnection

protocol_interface: ProtocolInterfaceHandler
bitfield: <Arc<Mutex<Bitfield>>
peers: <Arc<Mutex<PeerList>>
common_info: CommonInformation
tracker_address: String
sleep: u64
state: Arc<Mutex<PeerState>>

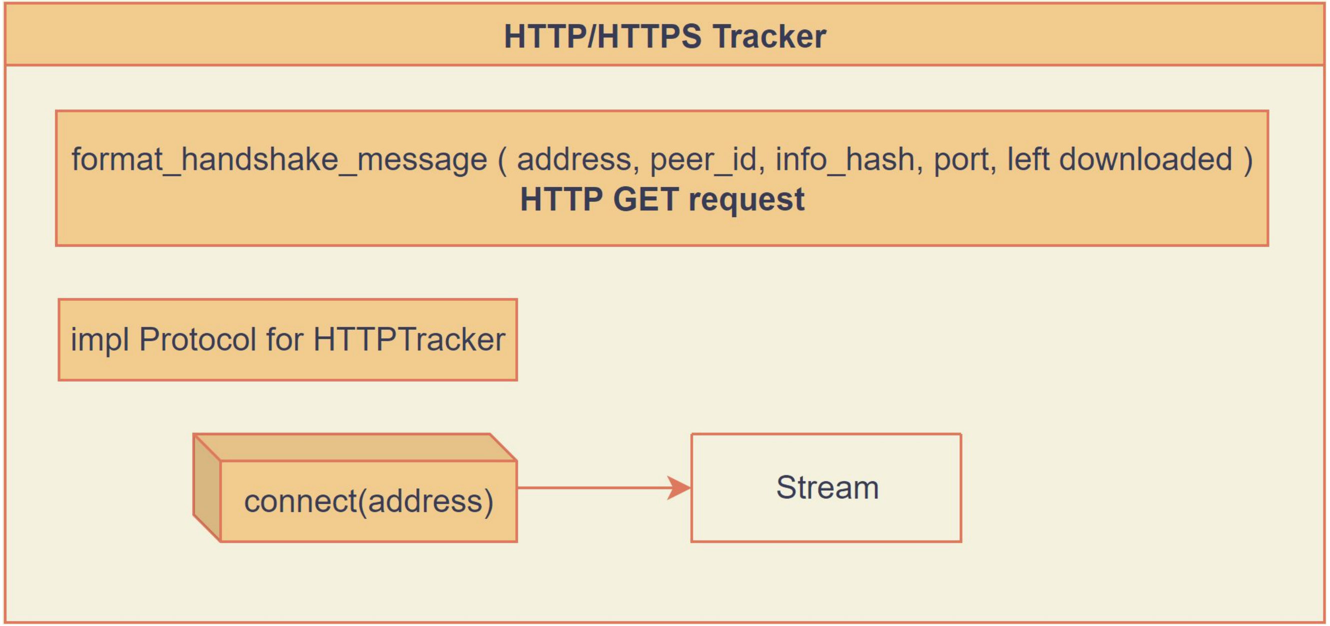
new (announce, bitfield, peers, common_information)
activate (listening_port)

– Protocol Interface Handler

Es un enumerate que maneja el request al Tracker dependiendo del protocolo.



Protocol
HTTP/HTTPS Tracker



Peer Record

Por cada peer obtenido se crea una instancia de *Peer Record* que almacena la información importante de cada peer y actualiza la PeerList.

Protocol Interface

Interface Protocol

protocol: P

new(protocol)
get_protocol ()
connect (target_address)
read_to_end (stream)
send (message)

Peer Handler

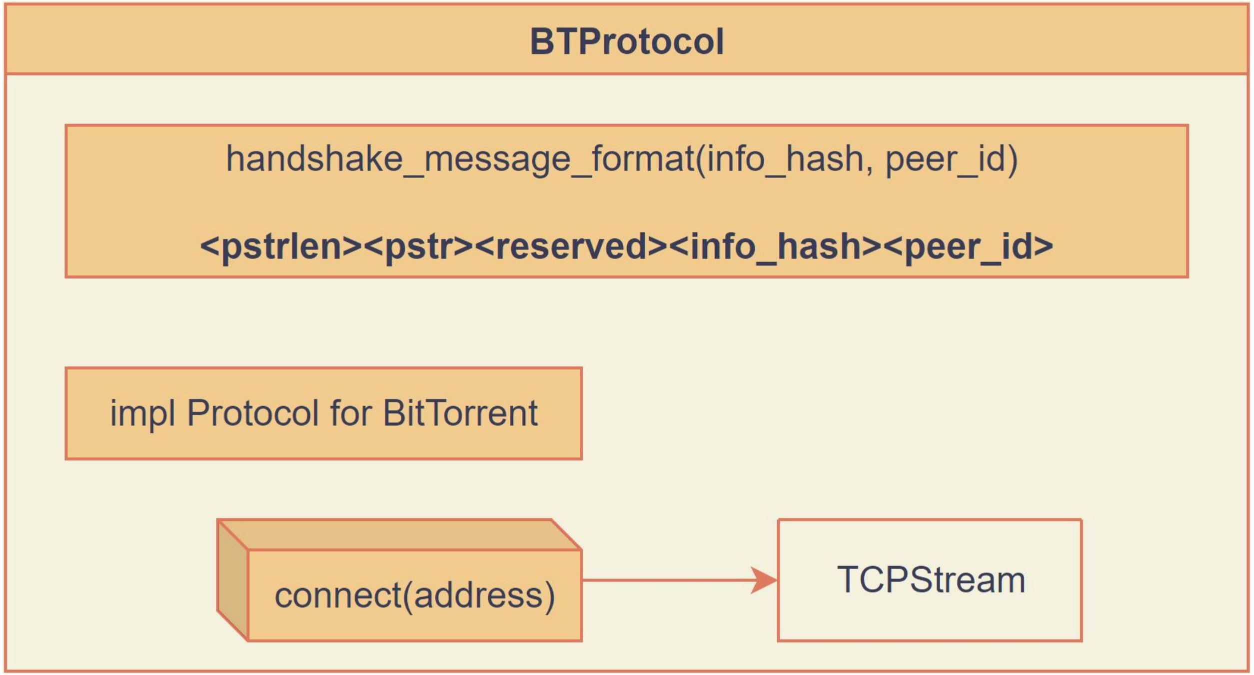
Peer Connection

PeerConnection

peer: PeerRecord
protocol_interface: ProtocolInterfaceProtocol<BTProtocol>>
bitfield: <Arc<Muex<Bitfiled>>
peers: <Arc<Muex<PeerList>>
common_information: CommonInformation
stream: <BTProtocol as Protocol>::Stream
state: State
peer_state: <Arc<Muex<PeerState>>

new (bitfield, peers, common_info, peer)
active (bitfield, peers, common_information, peer, connections)
greet ()
process_handshake_response ()
unchoke ()
download_piece ()

Protocol BitTorrent



File

File
pathname: String handler: std::fs::File (Handler)
open_file (pathname) new (pathname) with_contents (pathname) get_contentes (pathname, contents) get_block (piece_index, piece_size, block_size, block_offset) new_file_from_pieces (piece, pathname) concat (piece) join_pieces (path_from, path_to)

Server Handler

ServerHandler

bitfield: Arc<Mutex<Bitfield>>
peer_state: Arc<Mutex<PeerState>>
common_information: CommonInformation
ip: String
port: u16
socket: TCPListener

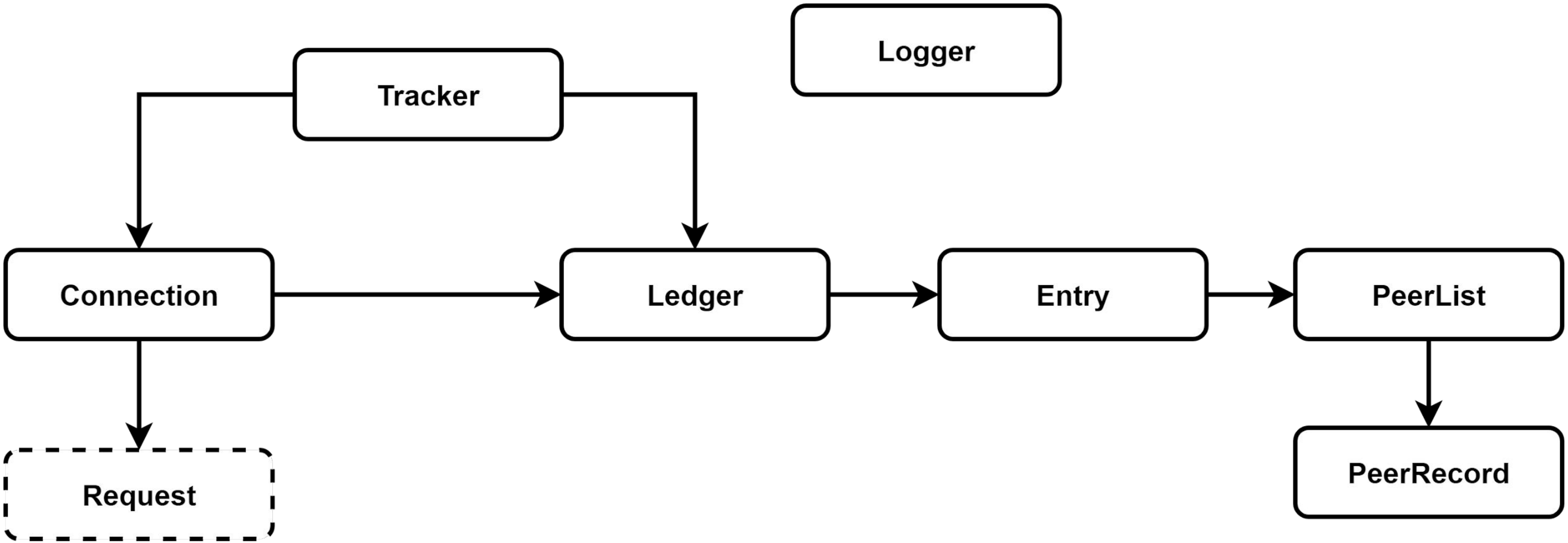
get_clients_ip ()
get_port ()
get_ip ()
new (bitfield, common_information, peer_state)
activate ()

Server Connection

ServerConnection
bitfield: Arc<Mutex<Bitfield>> peer_state: Arc<Mutex<PeerState>> common_information: CommonInformation state: UploadState
new (bitfield, common_information, peer_state) activate (bitfield, common_information, peer_state, stream) serve_file (stream) send_handshake_response (stream) validate_peer (stream, own_peer_id)

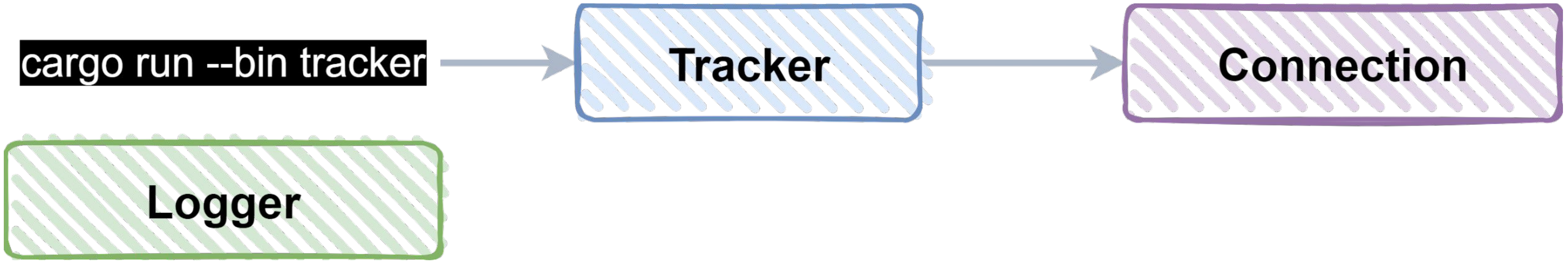
¡Ahora sí!
Vamos a la DEMO…

Agregado:
Tracker BitTorrent

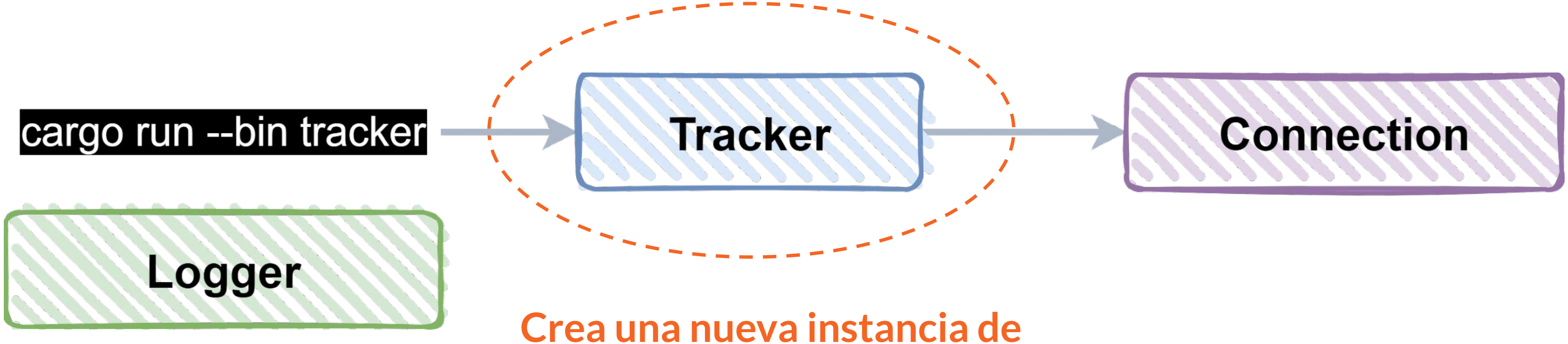


Logger
Tracker
Connection
Ledger
Entry
PeerList
Request
PeerRecord

Flujo



Flujo



Crea una nueva instancia de Tracker y la activa..

Veamos en detalle el Tracker...

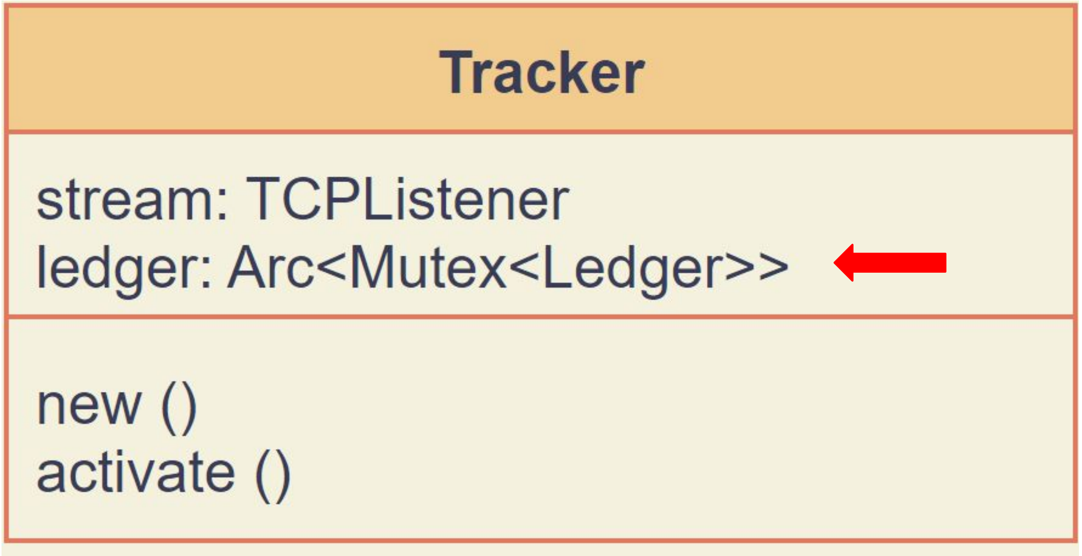
Tracker


Tracker
listener: TCPListener
ledger: Arc<Mutex<Ledger>>
new ()
activate ()

Tracker

El listener se queda esperando conexiones entrantes de los distintos peers..

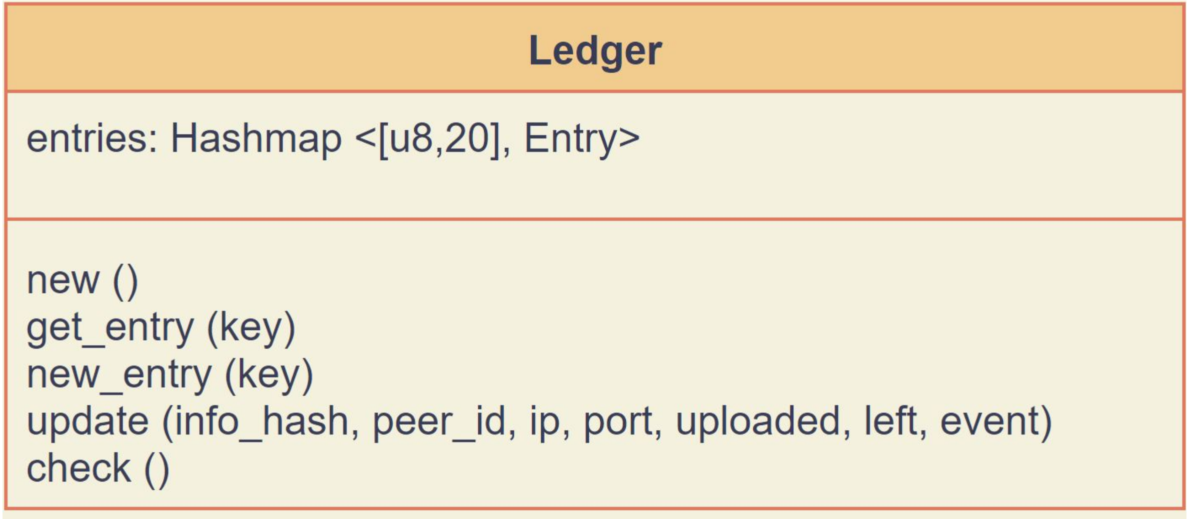
Tracker


Tracker
stream: TCPListener
ledger: Arc<Mutex<Ledger>>
new ()
activate ()

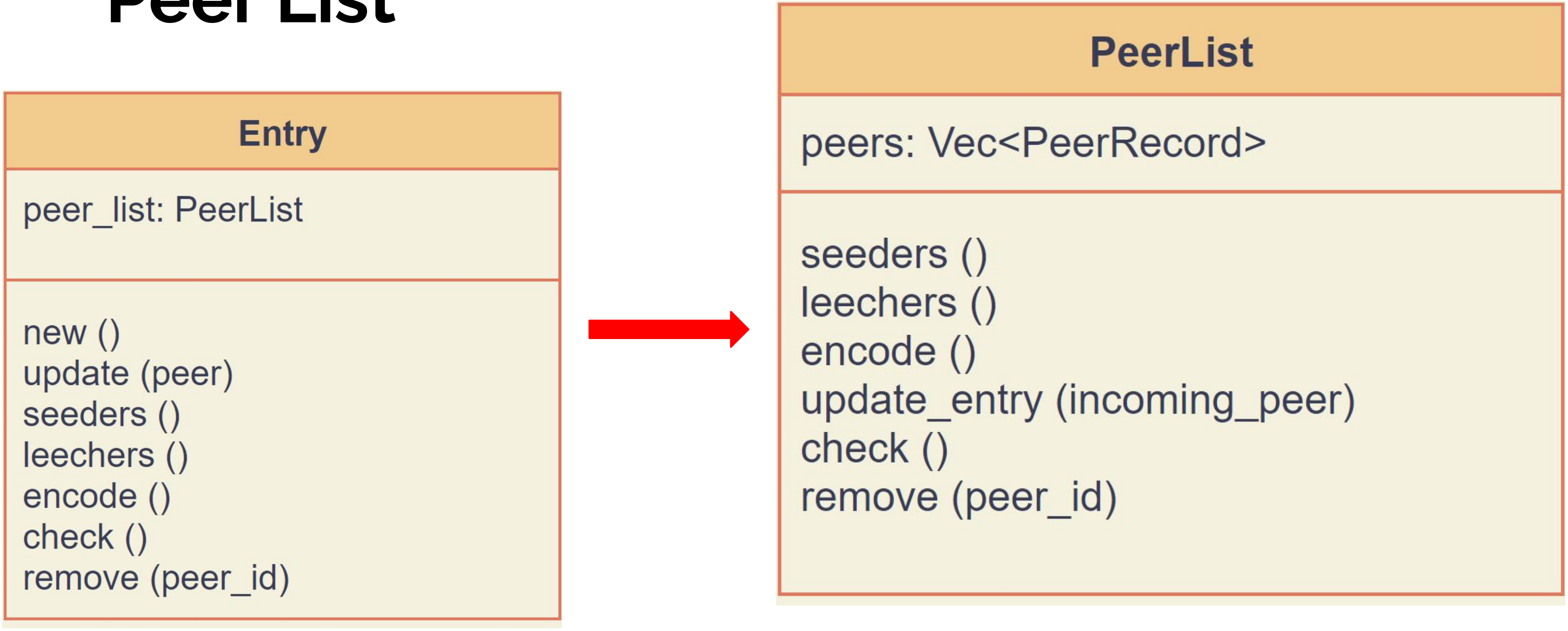
Ledger

Lleva un registro de todos los torrents solicitados:

{info_hash: Peer List}

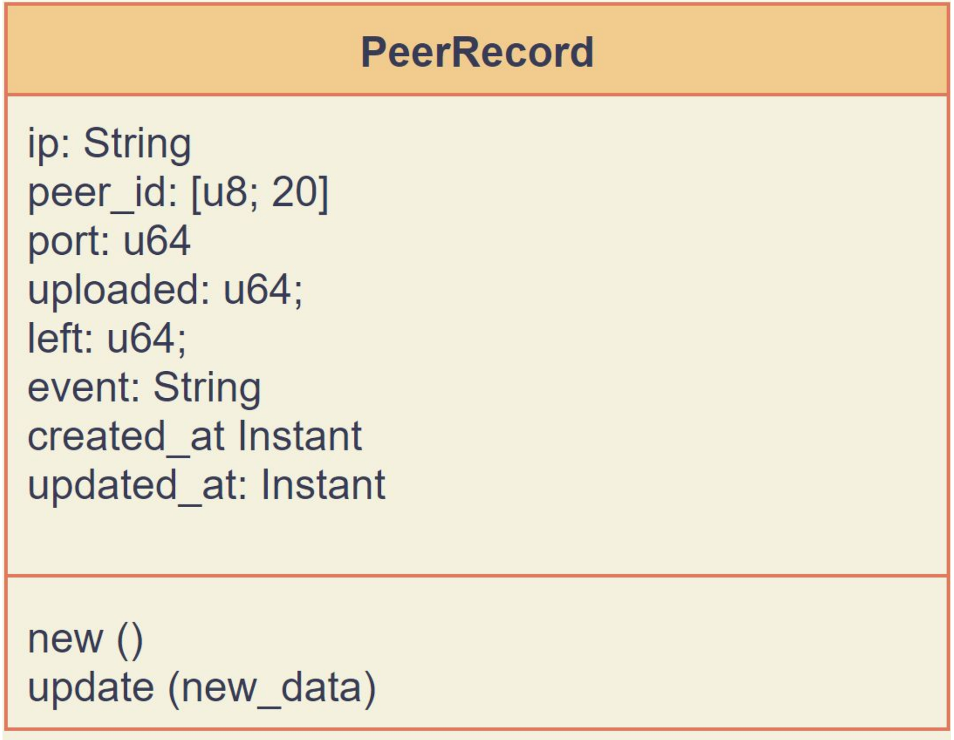


Entry & Peer List



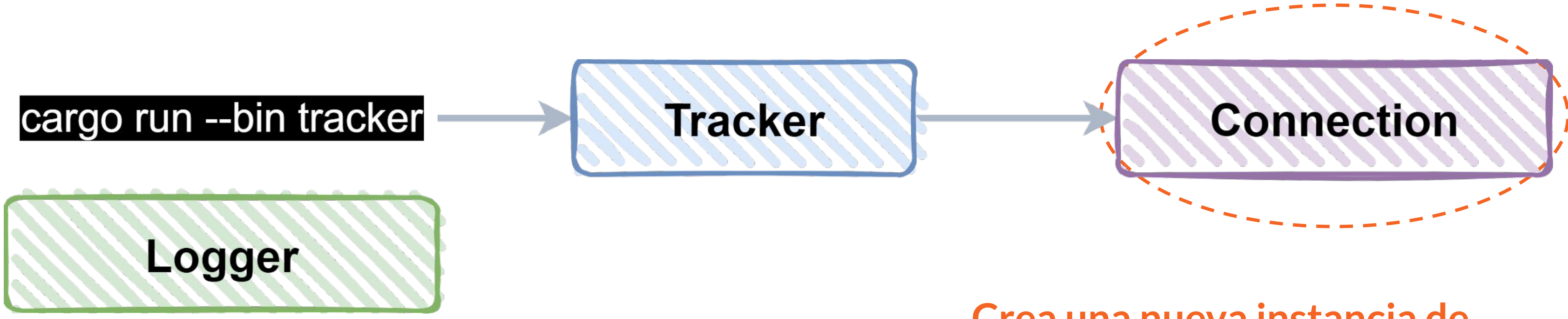
Peer Record

Al igual que antes, por cada peer obtenido se crea una instancia de *Peer Record* que almacena la información importante de cada peer.



¿Qué sucede cuando activamos el Tracker?

Flujo



Crea una nueva instancia de Connection y la activa dentro del vector connections

Connection

Cuando hacemos **activate()**:

- Leemos del stream
- Analizamos qué tipo de request recibimos

Connection

stream: TcpStream
ledger: Arc<Mutex<Ledger>>

new ()
activate ()
manage_cross_origin(allow, origin)
manage_stat ()
manage_announce (info_hash, peer_id, ip,port, uploaded, left, event)

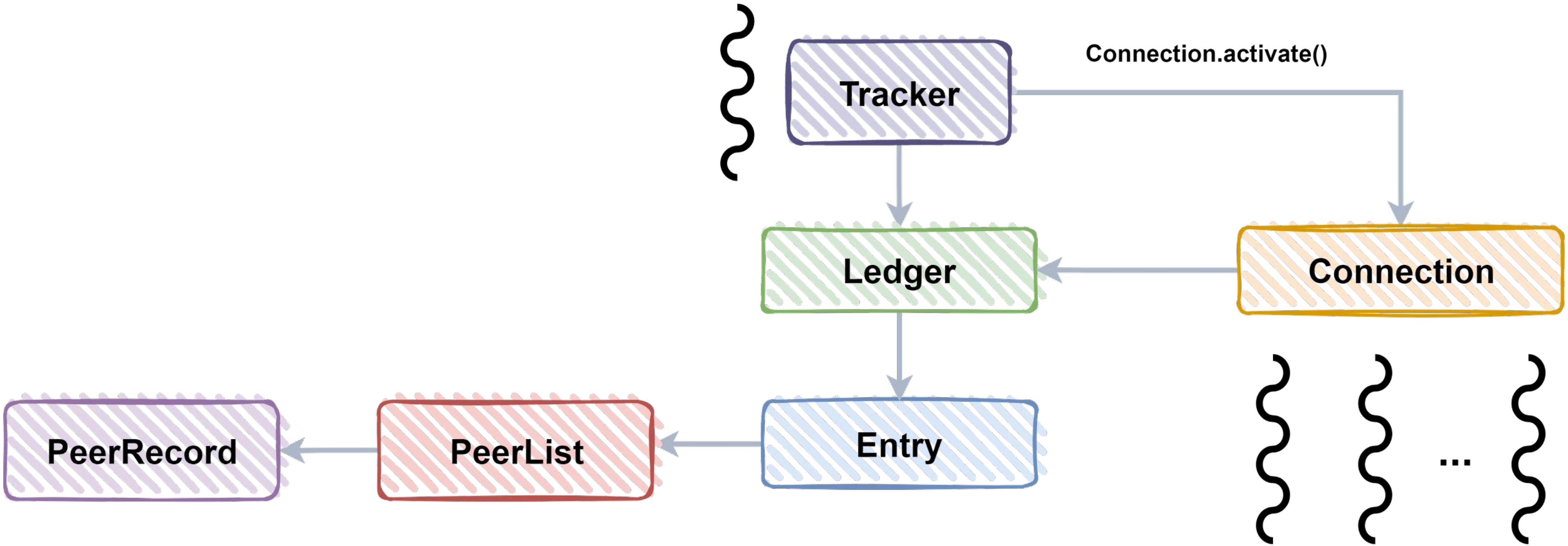
Request

- Es un enumerate de distintos tipos de Requests:
  Announce, Stats, Cors

- Splitea el request recibido y devuelve el enum correspondiente con la información necesaria.

```rust
pub enum Request {
    Announce {
        peer_id: [u8; 20],
        info_hash: [u8; 20],
        port: u16,
        uploaded: u64,
        left: u64,
        event: String,
        ip: String,
    },
    Stats,
    Cors {
        origin: String,
        allow: String,
    },
}
```

Flujo



Connection.activate()
Tracker
Ledger
Connection
Entry
PeerList
PeerRecord
...

¡Ahora sí!
Vamos a la DEMO…

¡GRACIAS!
¿PREGUNTAS?