Intelligence at the Speed of Light™

SDK Internals and Advanced Topics

Cepton Webinar Series #4

2022-05-04

Table of Contents

 Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page

What type of device is Cepton lidar?

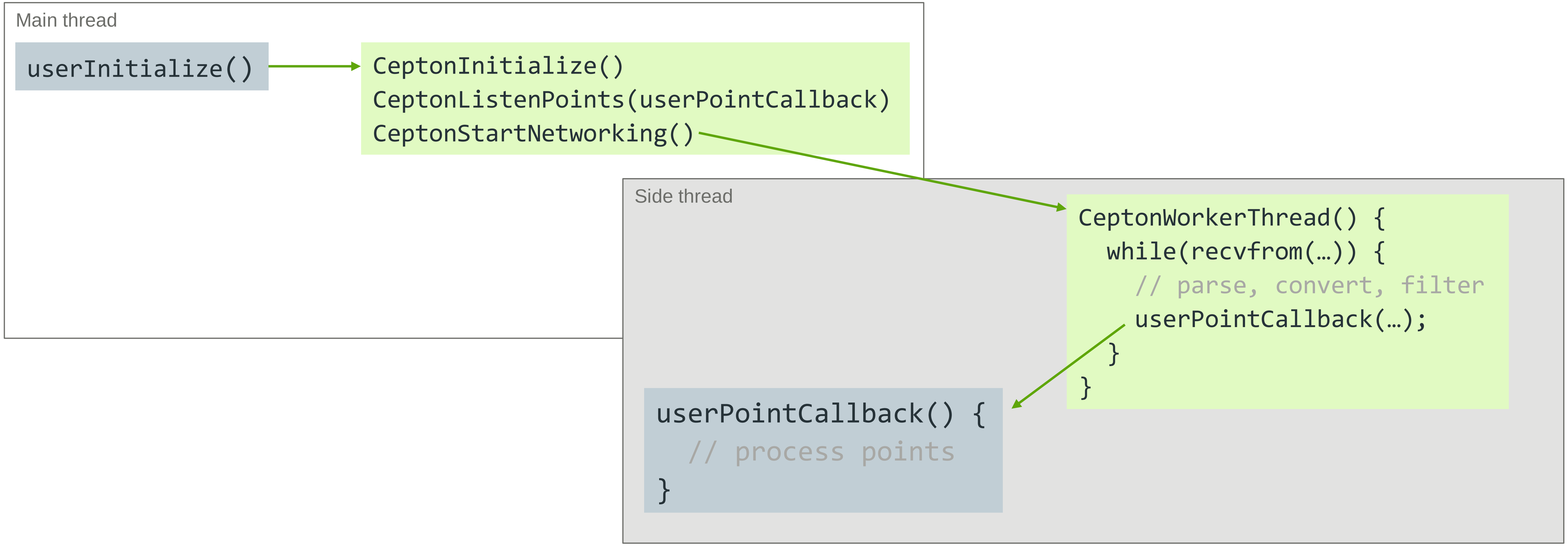
Important aspects of lidar that determine how SDK functions

- Lidars are connected through ethernet cables
  - All data coming from sensor to host are UDP packets
  - Ethernet devices do not need “driver”. User applications can use lidars directly (with SDK)
- Lidars are outgoing-only passive devices
  - SDK only listens, there is no handshake between lidar and host computer.
  - Exception: Device configurations and firmware updates. These are only used during setup or maintenance.
- Things lidar devices do not do:
  - No trigger for action, no dynamic area-of-interest (AOI).
  - No buffer for frames: Frame buffer exist on host computer. Lidars only do streaming-mode.
  - No connection: Lidars default to broadcast mode. Can be changed to multicast or unicast.

How does ethernet programming work?

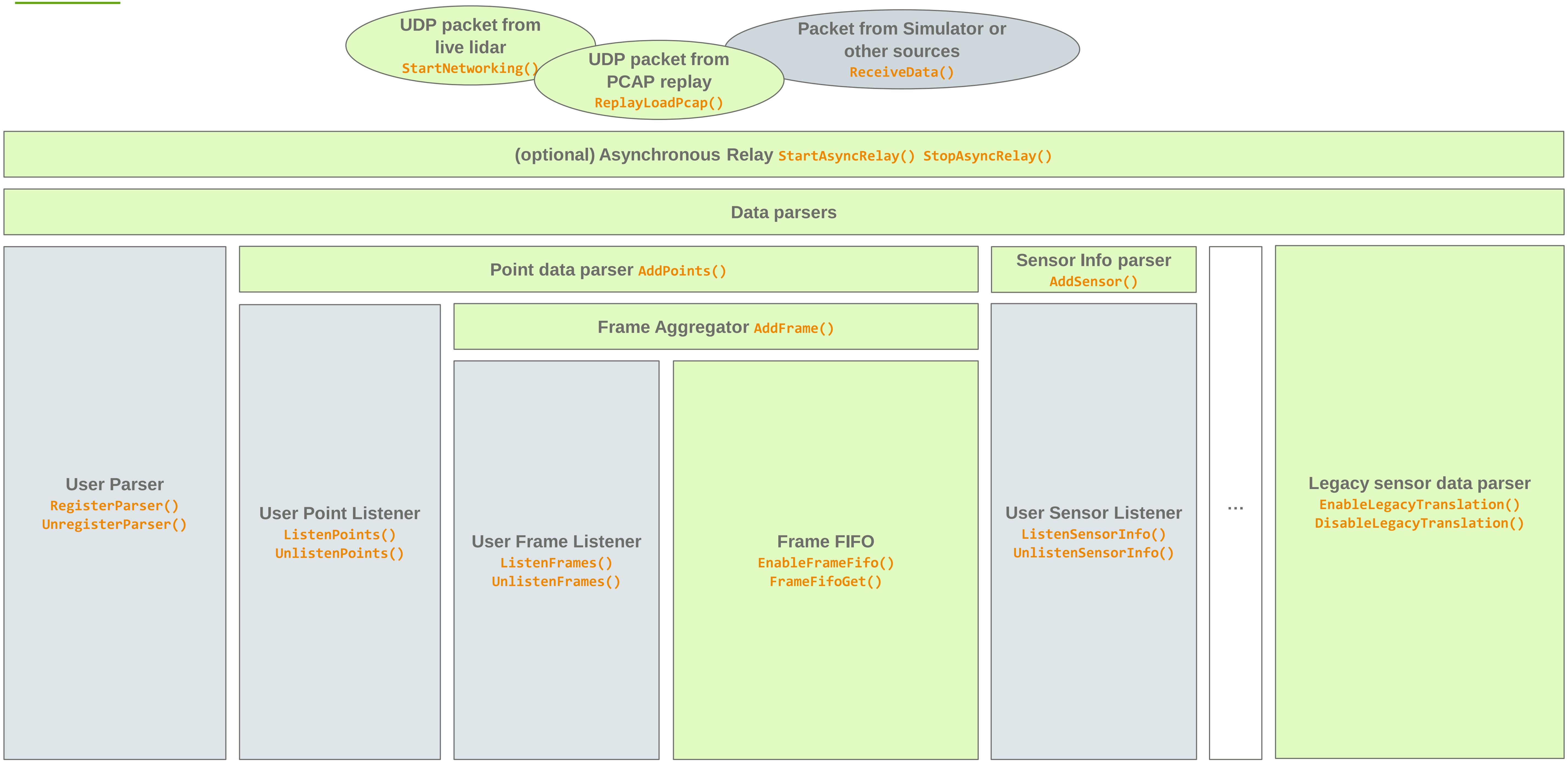
UDP programming: `socket()` → ~~`sendto()`~~`/recvfrom()`

- Simplest UDP program call `recvfrom()` which is blocking. Not suitable for SDK
- In SDK, we use a thread to do the blocking receive. This is standard practice in network programming (e.g., asio)



 Cepton is hiring interns and new college grads. Check out our [LinkedIn] or [Handshake] job page

Cepton SDK Architecture



Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page

Direct Frame FIFO

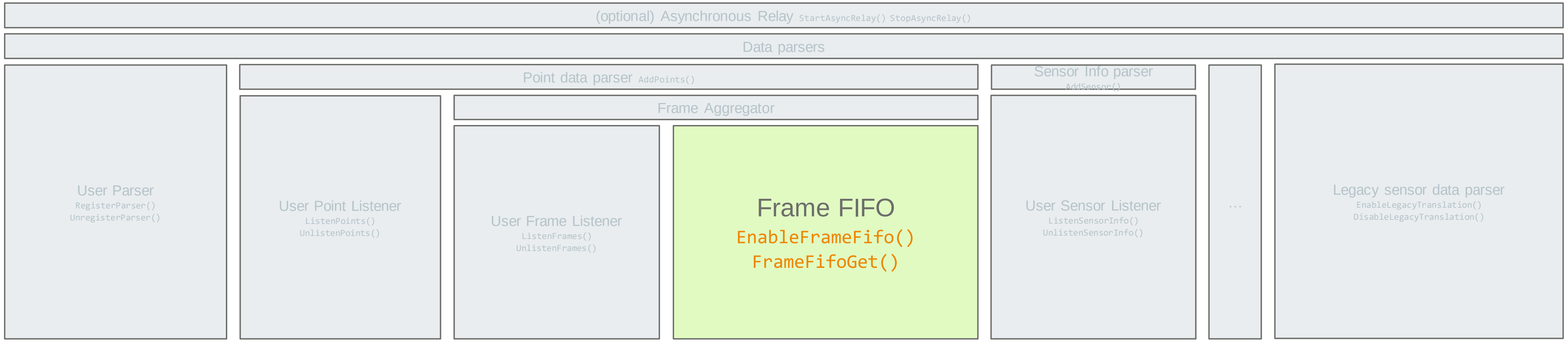
CEPTON

Ideal simple API to work with lidars: `GetFrame()`

- Python and MATLAB APIs are based on this.
- Buffered frame in FIFO: `EnableFrameFifo(…, int nFrames);`
- Frame FIFO are implemented as an independent module using `ListenFrames()`
  - Register itself as a frame listener
  - Allocate FIFO buffers at enable time
- Copy-less and allocation-free design: Concept of “get” and “release”
- Blocking and non-blocking behavior:
  - Both producer and consumer can block when FIFO is full or empty

Python offline processing sample:

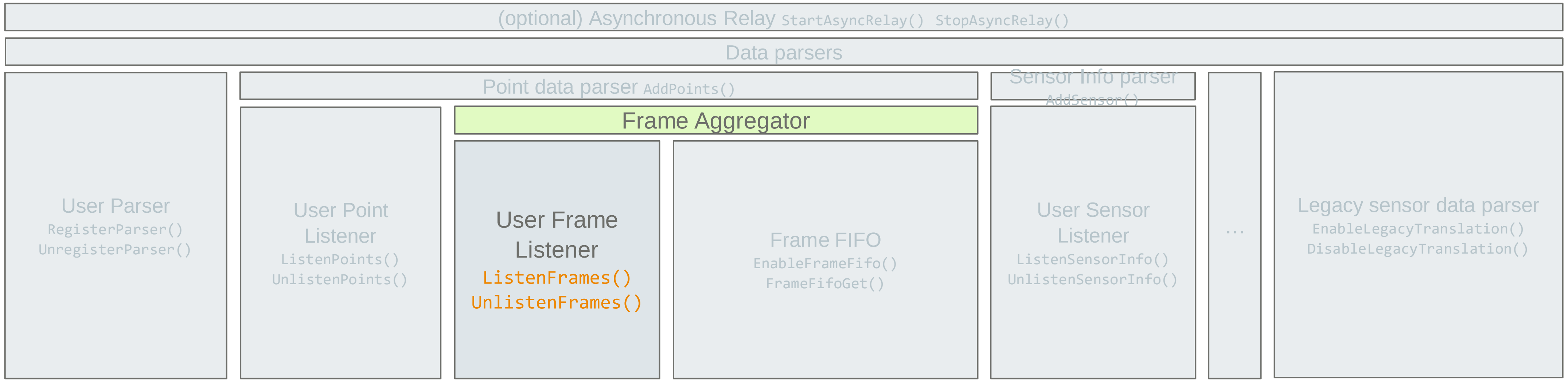
```
# Loop until pcap replay is finished and fifo drained
while not (sdk.ReplayIsFinished() and sdk.FrameFifoEmpty()):
    frame = sdk.FrameFifoGetFrame(timeout=2000) # 2000 ms

    if frame is not None:
        print("get frame: {}, size: {}, start timestamp: {:.6f}s durat
            frame_count, len(frame.flags), frame.timestamps[0]*1e-6, (
        frames.append(frame)
        frame_count += 1
        sdk.FrameFifoRelease()
```



Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page

Frame Aggregation

- What is a frame: A sequence of measurements that covers the entire field of view.
- Sensor hardware knows where a frame starts and ends. It uses per-point *frame parity* bit to indicate even/odd frames.
- SDK keeps all points in a buffer until parity changes. It issues a *frame callback* when that happens.
- SDK-side aggregation mode:
  - Natural mode: Use sensor's frame boundary, variable size, fixed pattern.
  - Timed mode: A fixed time slice of the streaming data. Can use to aggregate for longer or shorter time.
- Frame aggregation is implemented as an independent module using `ListenPoints()`
- Aggregated frames are served through callbacks registered with `ListenFrames()`
- Frame boundary (overlap) in natural mode



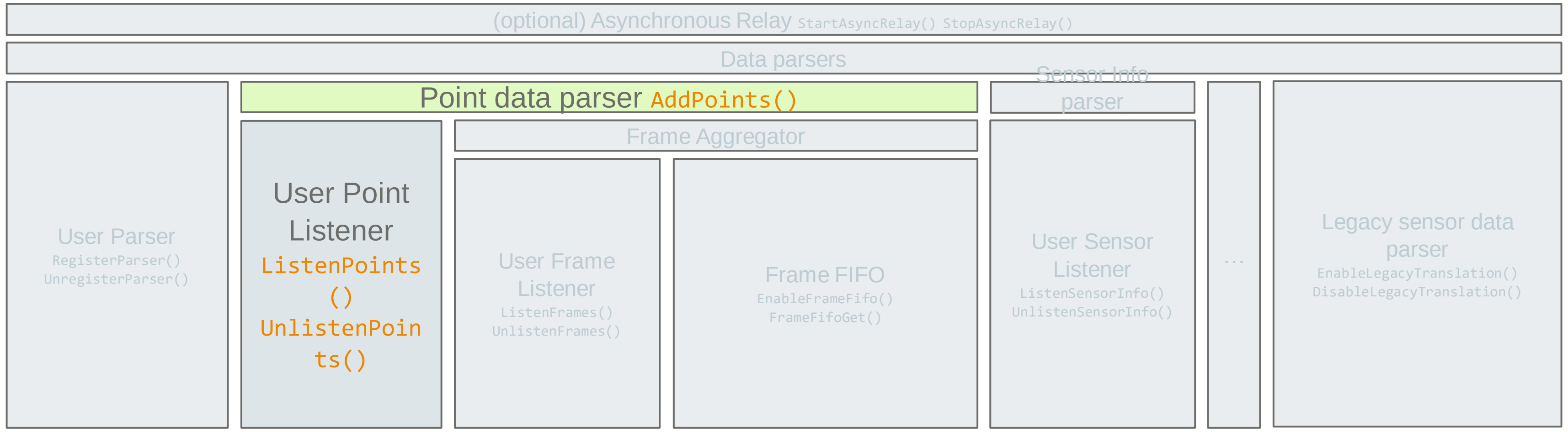
Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page

Point Data

- Point data comes directly from sensor in modern sensors
- Concept of *stride* to be future compatible
- Reflectivity representation: 0-100%, then 101-255 with exponential lookup table
- Legacy sensors or encrypted data goes through different parsers
- Explanation of the point flags
- Point data parser module:
  - Point data parser takes UDP packet and convert them to points.
  - Points are fed into every point listeners.
  - Direct call of `AddPoints()` can be used to inject points into SDK pipeline

SDK's point data structure:

```
enum {
  CEPTON_POINT_SATURATED = 1 << 0,
  CEPTON_POINT_LOW_SNR = 1 << 1,
  CEPTON_POINT_FRAME_PARITY = 1 << 2,
  CEPTON_POINT_FRAME_BOUNDARY = 1 << 3,
  CEPTON_POINT_SECOND_RETURN = 1 << 4,
  CEPTON_POINT_NO_RETURN = 1 << 5,
  CEPTON_POINT_NOISE = 1 << 6,
};

struct CeptonPoint {
  int16_t x;
  uint16_t y;
  int16_t z;
  uint8_t reflectivity;
  uint8_t relative_timestamp;
  uint8_t channel_id;
  uint8_t flags;
};
```



Sensor's point data format:

Offset	Field	Size	Unit	Range	Description
0	X	2	0.5cm	-163.840m to 163.835m	X coordinate
2	Y	2	0.5cm	0 to 327.68m	Y coordinate
4	Z	2	0.5cm	-163.840m to 163.835m	Z coordinate
6	Reflectivity	1	%	0 to 255	Reflectivity
7	Timestamp	1	us	0 to 255	Time difference from the point before
8	Channel	1		0 to 63	Channel ID
9	Flags	1			Flags
10+	Internal				When PointSize > 10, these are internal data

Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page

Packet Parsers

What's going through on the network?

- Point data
  - Cepton point clouds
  - Legacy point clouds (from older sensors)
  - Encrypted or signed point data
- Other data from sensors
  - Sensor information: Status, firmware version, etc.
  - Error reporting
- Other sensors
  - Camera
  - Other vendor's lidars

(Cepton network communication specificiation documents are available for latest sensor generations)

What can you do with SDK's raw packet parser?

- Leverage SDK's networking code with data callback
- Seamlessly handle different data formats
  - This is how `EnableLegacyTranslation()` works
  - Add support for other network devices in the same code base.
- Seamlessly inject data into the perception pipeline
  - Directly call `ReceiveData()` to emulate live sensor
  - This is how PCAP replay is implemented
  - Can integrate with frameworks where networking is not directly exposed. (NVIDIA Driveworks, e.g.)

 Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page

Packet Parsers (continued)

How to use packet parsers?

- This is not common, only advanced applications need to parse packets directly.
- User code and SDK internal modules (like legacy connector) register parser callbacks:
  - `RegisterParser()` `UnregisterParser()`
- Each UDP packet received by SDK is passed into every parser callbacks until it is "consumed" by one.
  - Parsers use return values to indicate if the data should be passed on or not.
- User code and SDK's replay module can use `ReceiveData()` to directly inject data into this parser chain.
- Existing SDK parsers:
  - Point data parser (enabled by default)
  - Sensor info parser (enabled by default)
  - Serial data parser (enabled when `ListenSerialLines()` is called)
  - Legacy connector parser (enabled when `EnableLegacyTranslation()` is called)

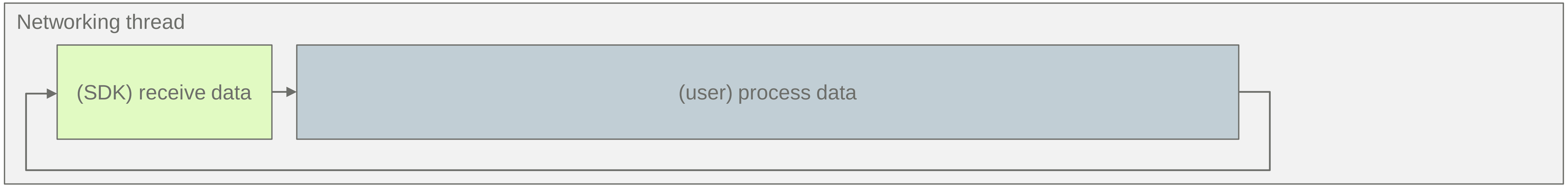
Asynchronous Relay

Problem: UDP is not tolerant of latency.

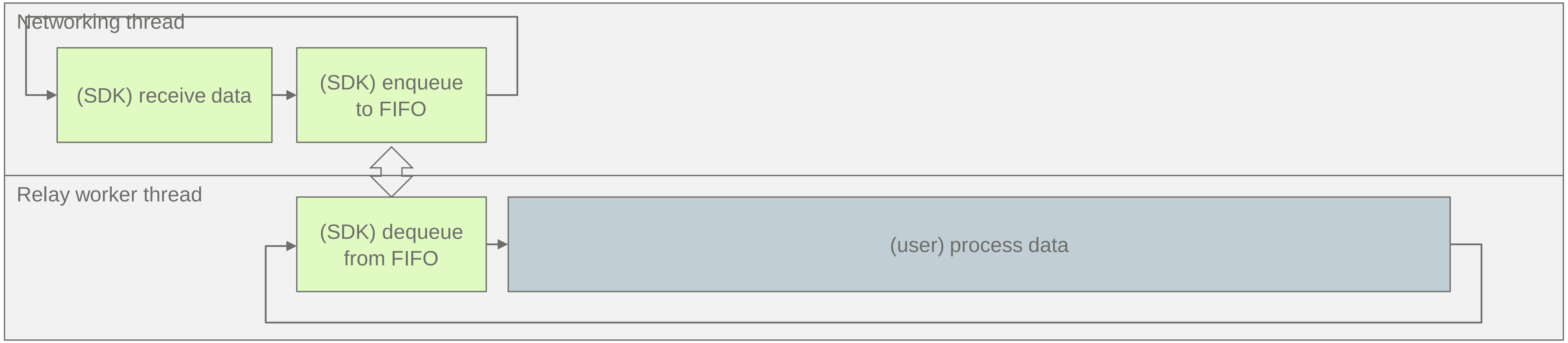
- User callback routines (point or frame) is run directly on networking thread with unbounded delays
- It is not desirable to limit processing time, neither is losing data from sensor.

Solution: Decouple UDP receive code from user processing code with a producer-consumer FIFO:

Before:



After:



Asynchronous Relay (continued)

- SDK’s asynchronous relay is a middleware sits between network data and all the parsers.
- SDK’s asynchronous relay is transparent: You can enable and forget.
- Relay FIFO helps when there are uneven processing times that are fast enough on average.
- When relay FIFO is full, data will still get lost. If average processing time is slower than incoming data, FIFO won’t help.
- Like other features in SDK: asynchronous relay is off by default. It must be explicitly enabled.
- When enabled, asynchronous relay allocates all the FIFO buffers and starts a new thread. There is no other allocation or additional copying during the operation.

Capture Replay Facility

Replay functionalities

- Load and play multiple .pcap files, control each independently: `ReplayLoadPcap()`
- Support pause/resume/step/seek: `ReplayPause()` `ReplaySeek()` etc.
- Replay speed can be controlled: `ReplaySetSpeed()`
- Special “full speed” mode for off-line processing without delay

Replay internals

- Replay facility is implemented as an independent module using `ReceiveData()`
- Each pcap file being replayed creates its own thread for replay
- Indexing of pcap
  - Each pcap file being replayed may optionally create an indexing thread: `ReplayGetIndexPosition()`
  - Index records are not persisted for .pcap files
- Replay can coexist with live sensor or other data sources.

 Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page

Brief Summary

SDK architecture

- Features in SDK are like Lego blocks with all interfaces exposed for customization.
- SDK is doing very little work other than plumbing and buffering.
- SDK is designed to do no copying (other than frame aggregation) and no allocation during perception inner loop.

Interact with SDK at different levels

- Direct frames from FIFO
- Frame callbacks
- Point callbacks
- Raw data callbacks

Ways to use SDK

- Online processing with live sensor
- Offline processing with PCAP
- Mix and match different things with direct data injection

Advanced Topics

- Sensor self announcement
  - Cepton sensor broadcast a “sensor info” data packet regularly (every 0.5-1 second)
  - Sensor info packet contains both static info (serial number, revision, firmware version, features etc.) and diagnostic info (temperatures, voltages etc.)
  - Caveat: Point data can arrive *before* sensor info.
- SDK binding mechanism
  - Operating system native dynamic linking (C/C++/Rust)
  - Python/Matlab integrations and language-native APIs.
  - Foreign function interfaces for JavaScript/Ruby etc.
- How legacy connector works and some FAQs on legacy hardware
  - For Vista-P sensors, CPU is used to perform some calibration calculations that can be heavy.
  - Legacy connector relies on the old SDK dynamic library to function.
- Details in handling multiple sensors
  - `CeptonSensorHandle` is the unique identifier of sensors.
  - For live sensors `CeptonSensorHandle` is the source IP address.

 Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page

Future Topics For Webinar

- Cepton MMT and scan pattern.
- Cepton's perception system and CR file.
- Physics of lidar sensors and common misconceptions

 Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page

Resources

- Developer Center (https://developer.cepton.com coming soon…)
  - This and all other webinars
  - Download SDK package
  - Download Cepton Viewer executable
- Official cepton.com
- JOB postings: LinkedIn and Handshake

Cepton is hiring interns and new college grads. Check out our LinkedIn or Handshake job page