

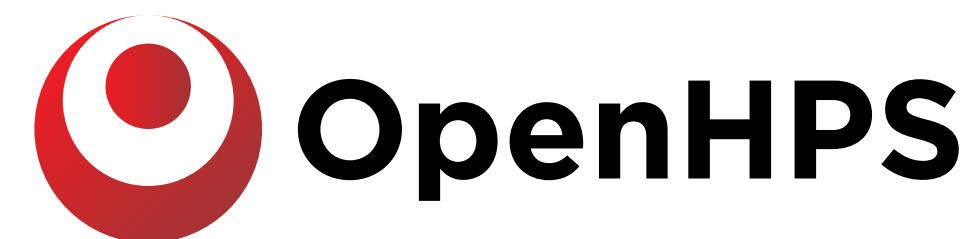
Rapid Prototyping of a Positioning System Using the OpenHPS Framework

Maxim Van de Wynckel

*Web & Information Systems Engineering Lab
Vrije Universiteit Brussel*



WEB & INFORMATION
SYSTEMS ENGINEERING



Positioning System



"A positioning system is a mechanism for determining the position of an object in space."

- Wikipedia (2022)

Positioning System



*"A positioning system is a mechanism for determining the position of an **object** in space."*

- Wikipedia (2022)

Object

What are you tracking? A person, an asset or a phone?

Space

Outdoor, indoor, under water or on a table?

Use Cases



- ▶ **Navigation**

Navigate a person from point A to point B

- ▶ **Tracking**

Asset tracking, customer tracking, tracking items on a table

- ▶ **Location Awareness**

Trigger an action whenever a specific person is in a room

- ▶ **Mapping**

Geospatial mapping of an environment

Technologies

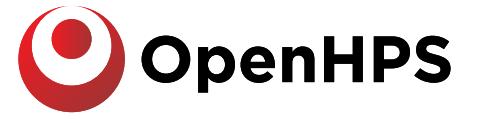


Technologies used to obtain sensor data for positioning

- ▶ Camera (stereoscopic, monocular, omnidirectional¹⁾)
- ▶ Beacons (ultrawideband, Bluetooth, ultrasound)
- ▶ LIDAR
- ▶ Inertial measurement unit (IMU)
- ▶ Visible light communication
- ▶ ...



Algorithms



Algorithms used to process sensor data

- ▶ Lateration
 - ▶ Proximity positioning
 - ▶ Signal propagation
 - ▶ Fingerprinting
 - ▶ Computer vision
 - ▶ Dead reckoning
 - ▶ Sensor fusion
 - ▶ ...



Open Source Solutions

- ▶ AnyPlace <https://anyplace.cs.ucy.ac.cy/>
- ▶ FIND <https://github.com/schollz/find3>
- ▶ IndoorLocation <https://github.com/IndoorLocation>
- ▶ Navigine <https://github.com/Navigine>
- ▶ RedPin <http://redpin.org/>
- ▶ Traccar <https://github.com/traccar>
- ▶ TraceMeNow <https://isislab-unisa.github.io/trace-me-now>

An Open Source Hybrid Positioning System

 OpenHPS

DOCS BLOG GITHUB

Documentation

- Introduction
- Installation
- Modules

Basic Concepts

- Data Object
- Data Frame**
 - Creating data frames
 - Creating a custom data frame
- Standard Units
- Position and Orientation
- Reference Space
- Positioning Model
- Source Node
- Processing Node
- Sink Node
- Services

Advanced Concepts

- Remote Service
- Threading

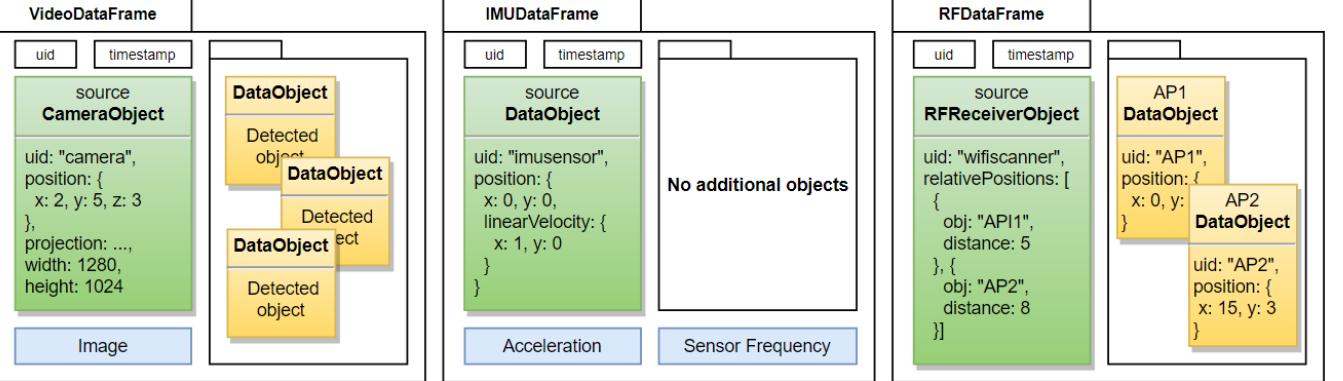
Miscellaneous

- Examples

Data Frame

Data frames are envelopes that are transmitted and processed through a positioning model. These frames are created by source nodes (e.g. sensors) and contain one or more data objects needed to process the frame.

A frame should contain a single reading of a sensor (such as an image of a video stream or current acceleration) and not permanent or calculated information.



Creating data frames

OpenHPS is a framework that processes sensor information to retrieve a position for one or more data objects. These objects are contained within an envelope called a data frame.

```
import { DataObject, DataFrame } from '@openhps/core';
const myObject = new DataObject("bsigner", "Beat Signer");
const frame = new DataFrame();
frame.addObject(myObject);
(method) DataFrame.addObject(object: DataObject): void
```

A basic data frame supports the addition of objects. Extended versions of this basic data frame also add additional sensor data.

Creating a custom data frame

Similar to data objects, decorators have to be used to indicate a serializable data frame.

```
import {
  DataFrame,
  SerializableObject,
  SerializableMember
} from '@openhps/core';

@SerializableObject()
export class QRDataFrame extends DataFrame {
  public rawImage: any = undefined;
}
```

License Apache 2.0

Built With TypeScript

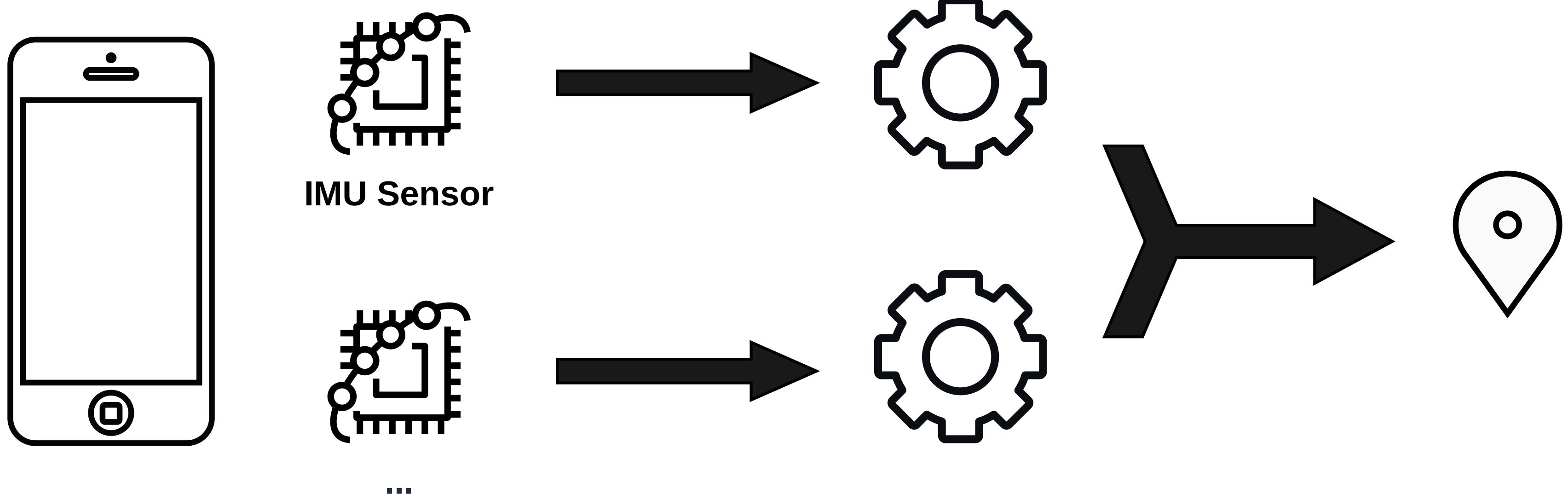
Website openhps.org



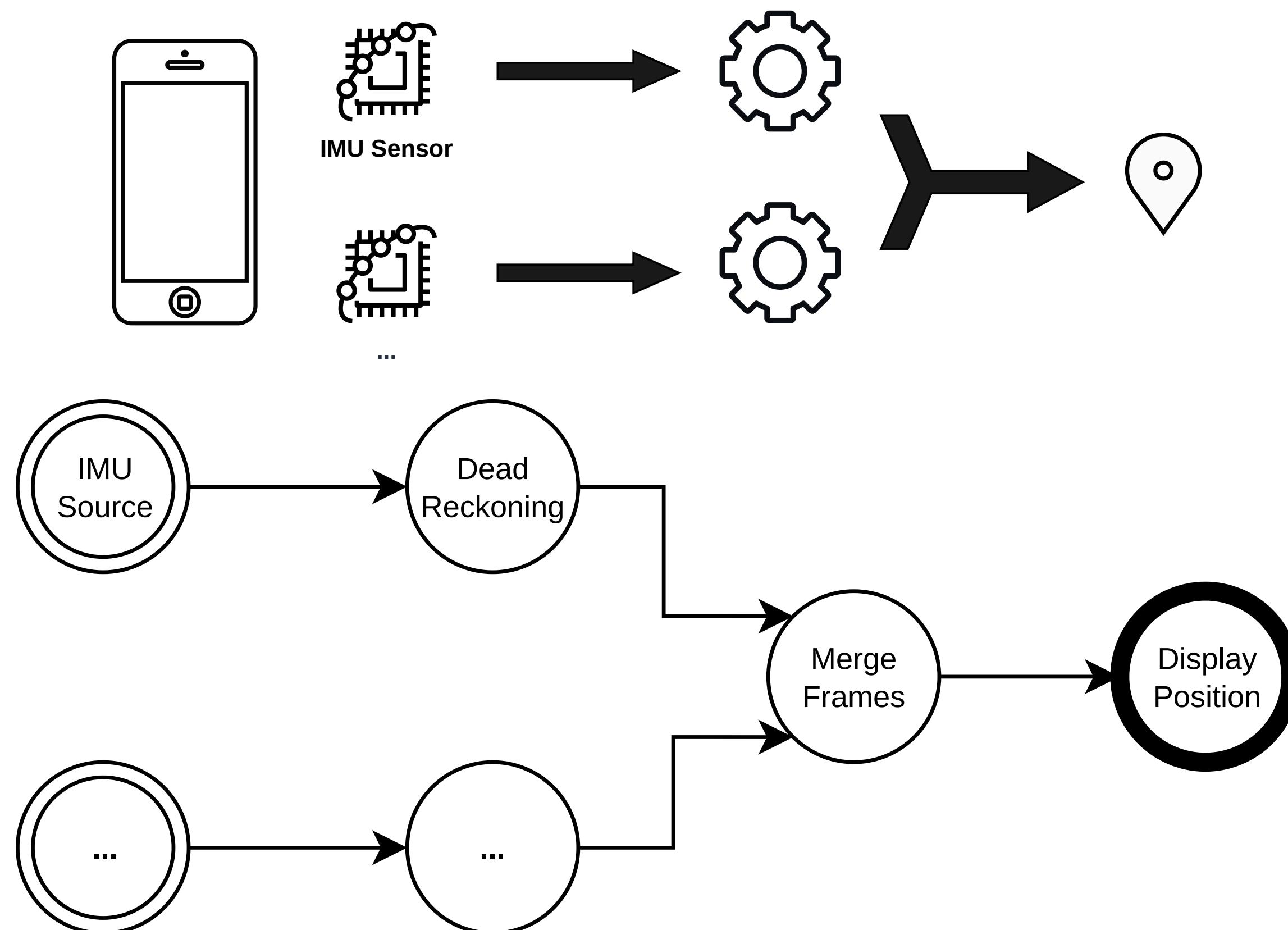
An Open Source Hybrid Positioning System

- ▶ Any technology
- ▶ Any algorithm
- ▶ Various use cases
- ▶ Flexible processing and output
 - Prefer accuracy over battery consumption, reliability, ...
- ▶ Aimed towards developers and researchers

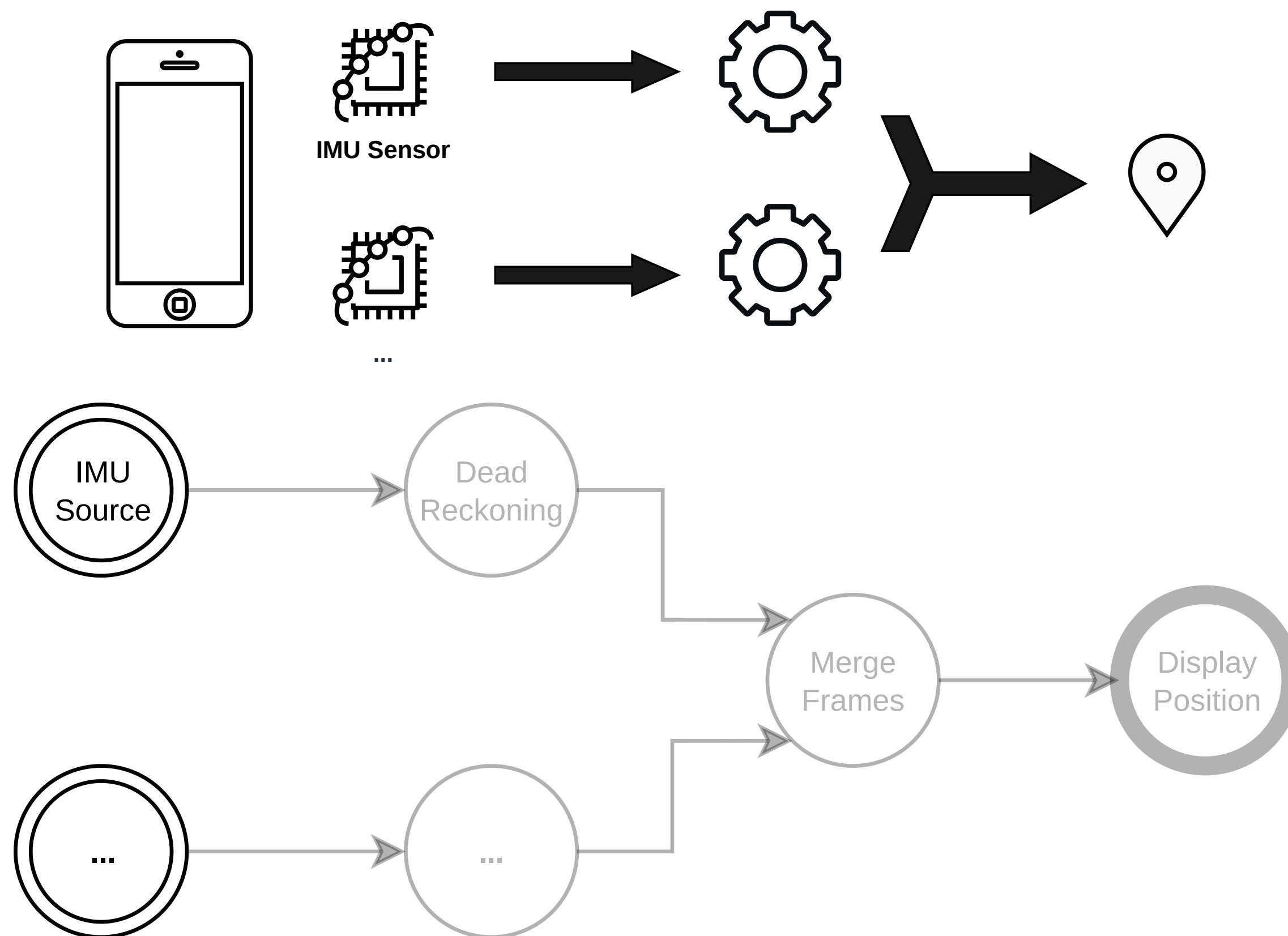
Process Network Design



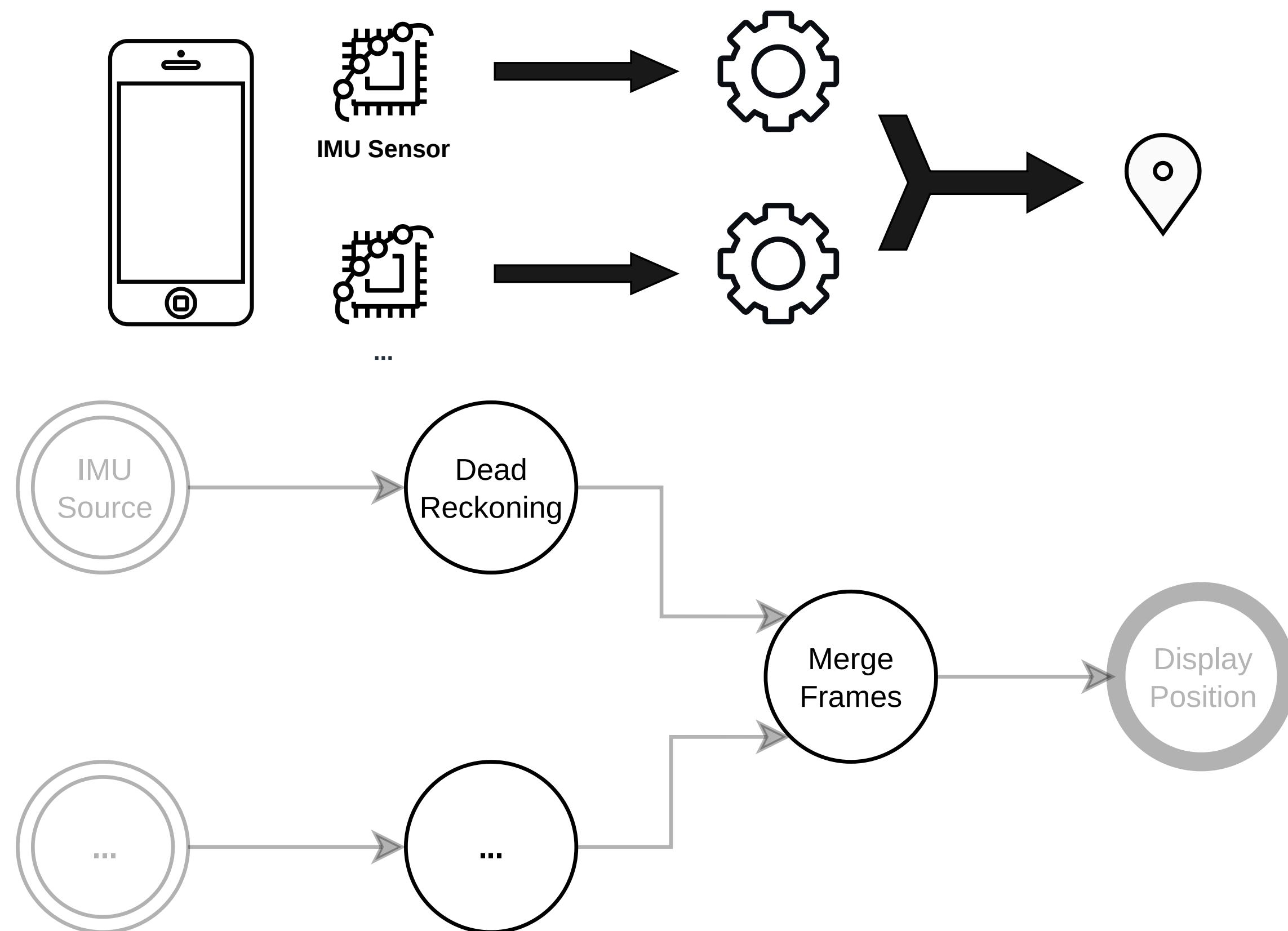
Process Network Design ...



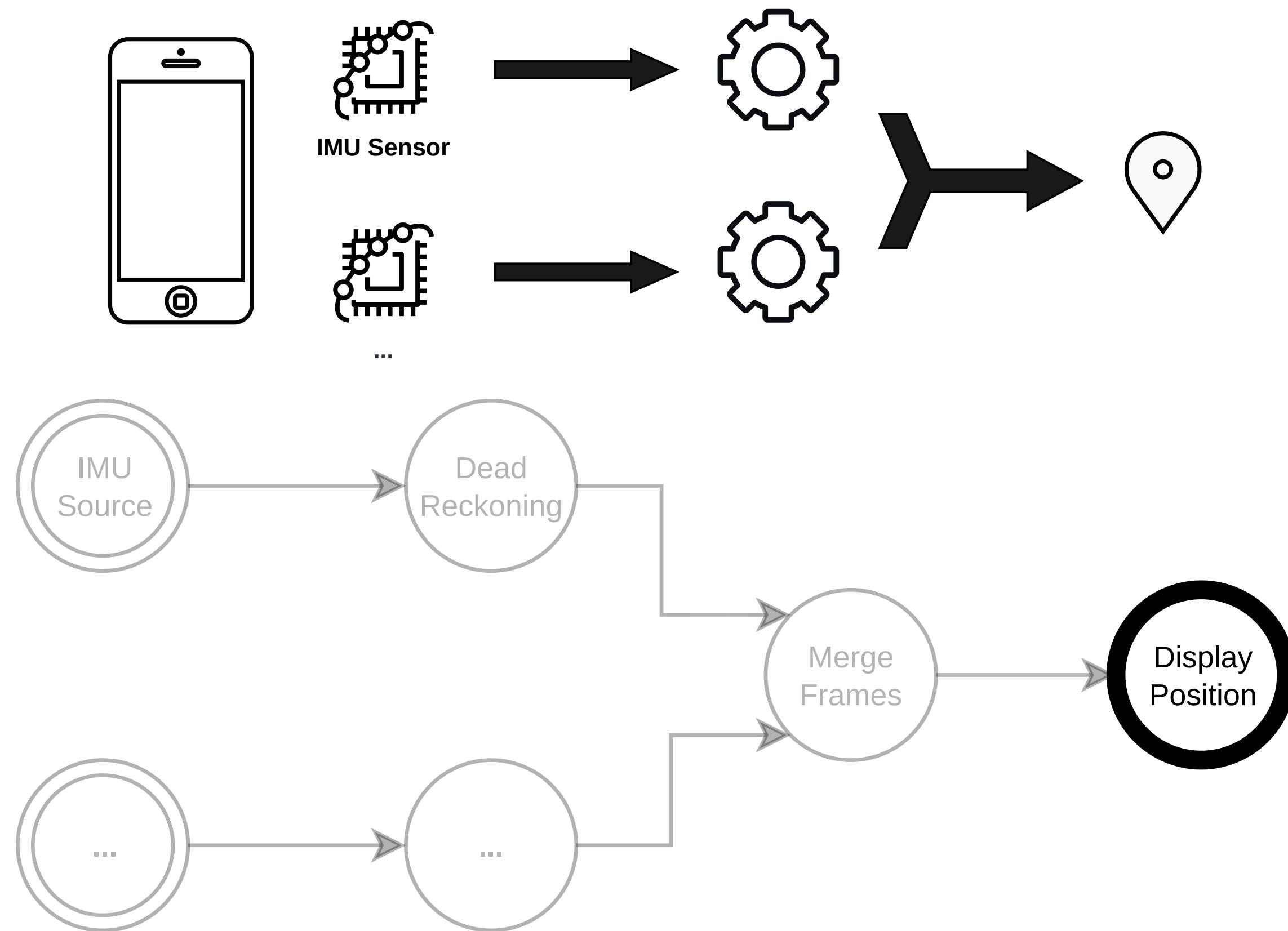
Process Network Design ...



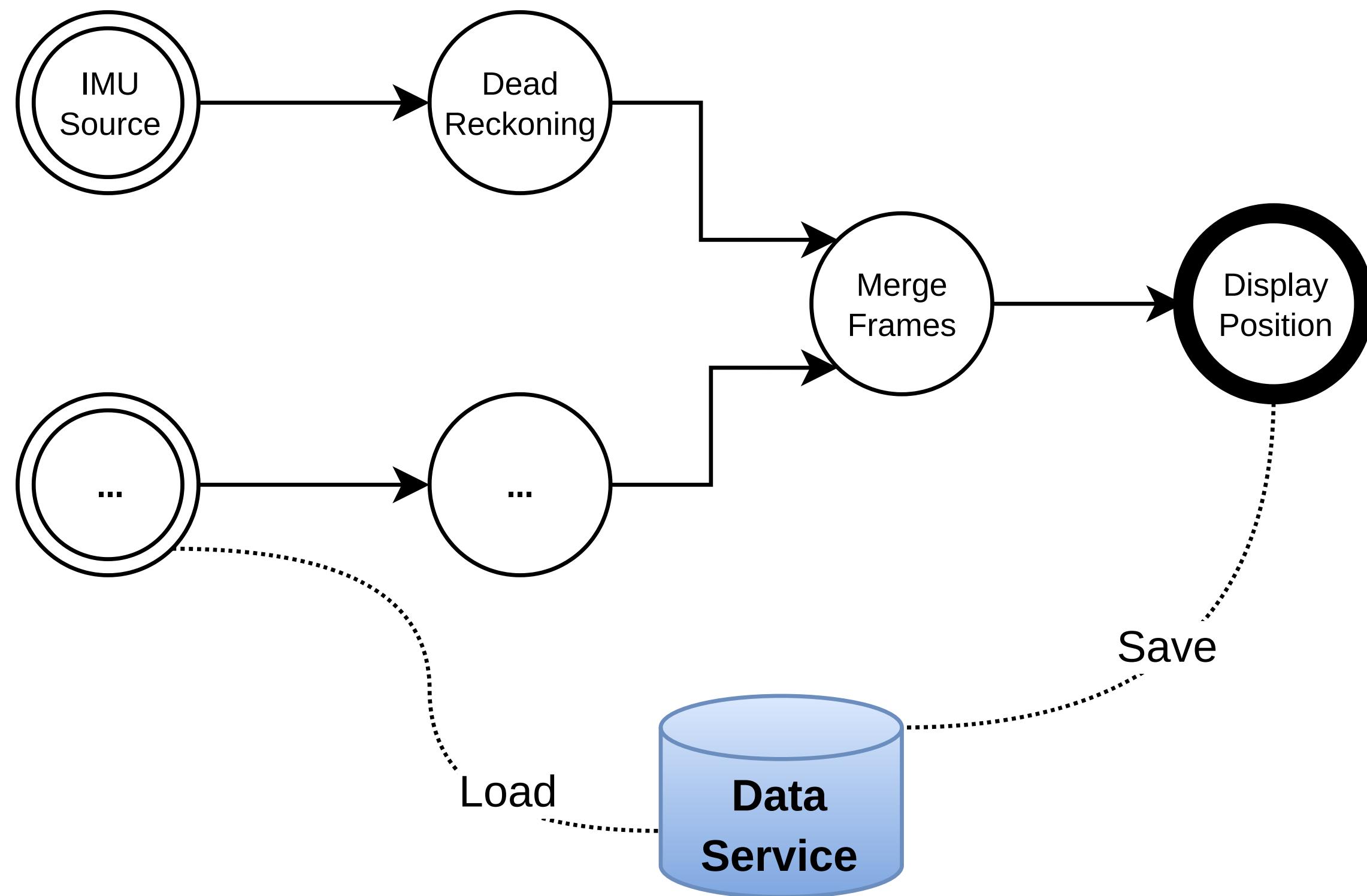
Process Network Design ...



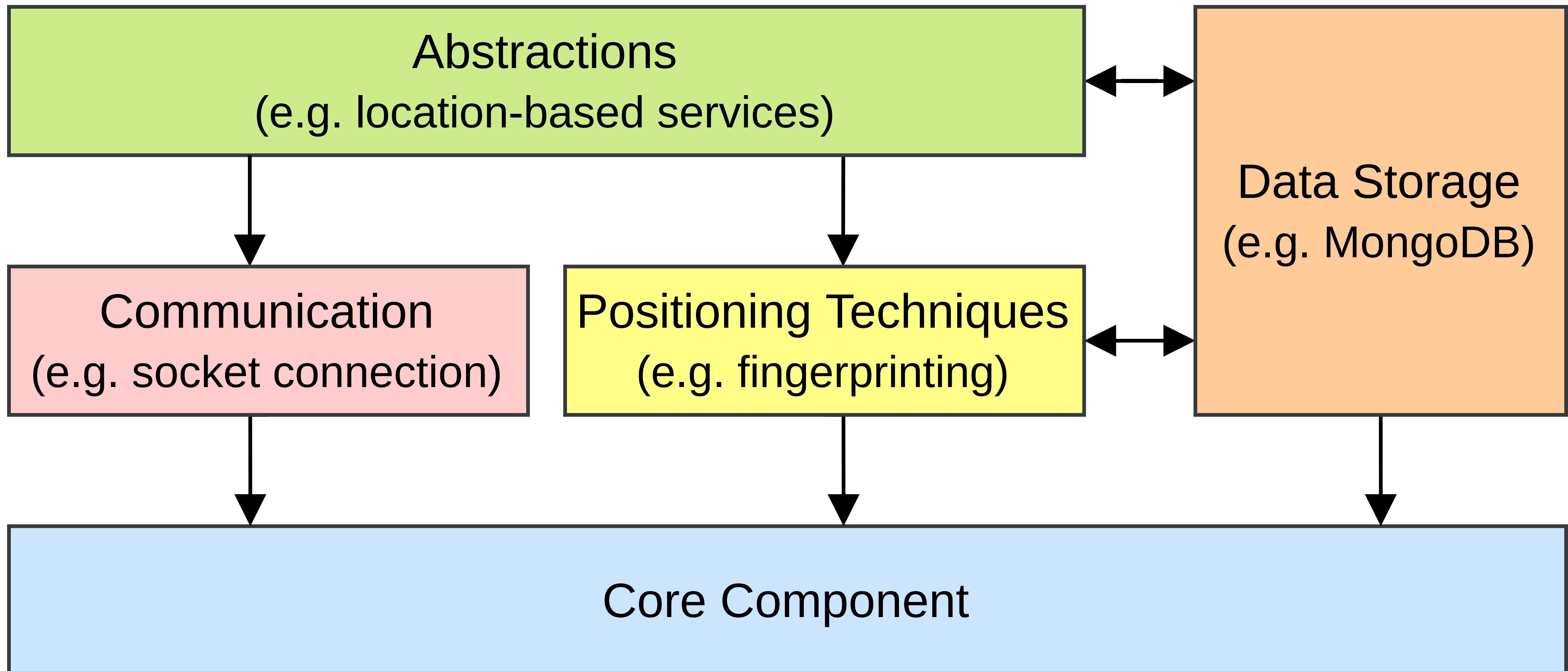
Process Network Design ...



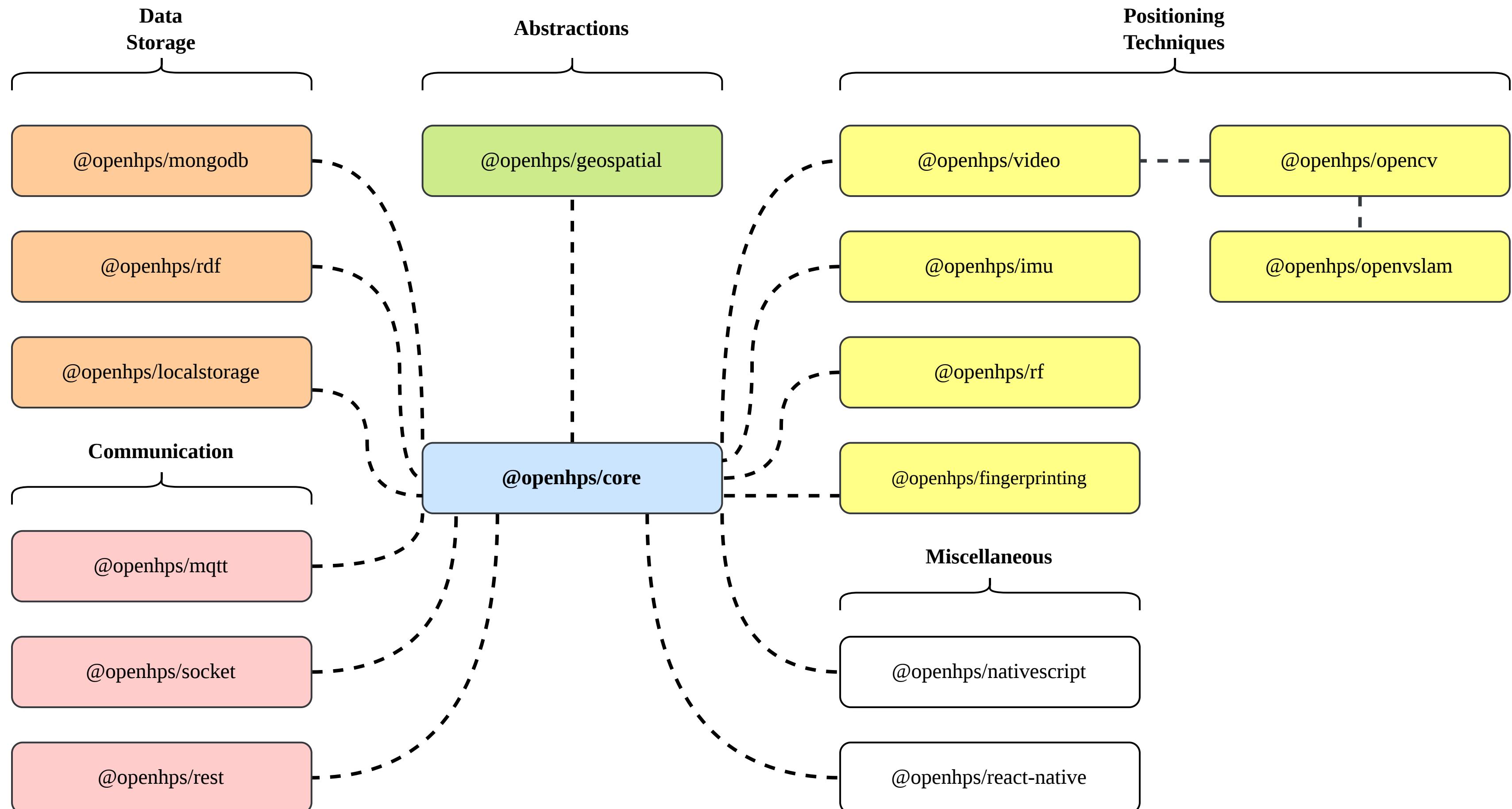
Process Network Design ...



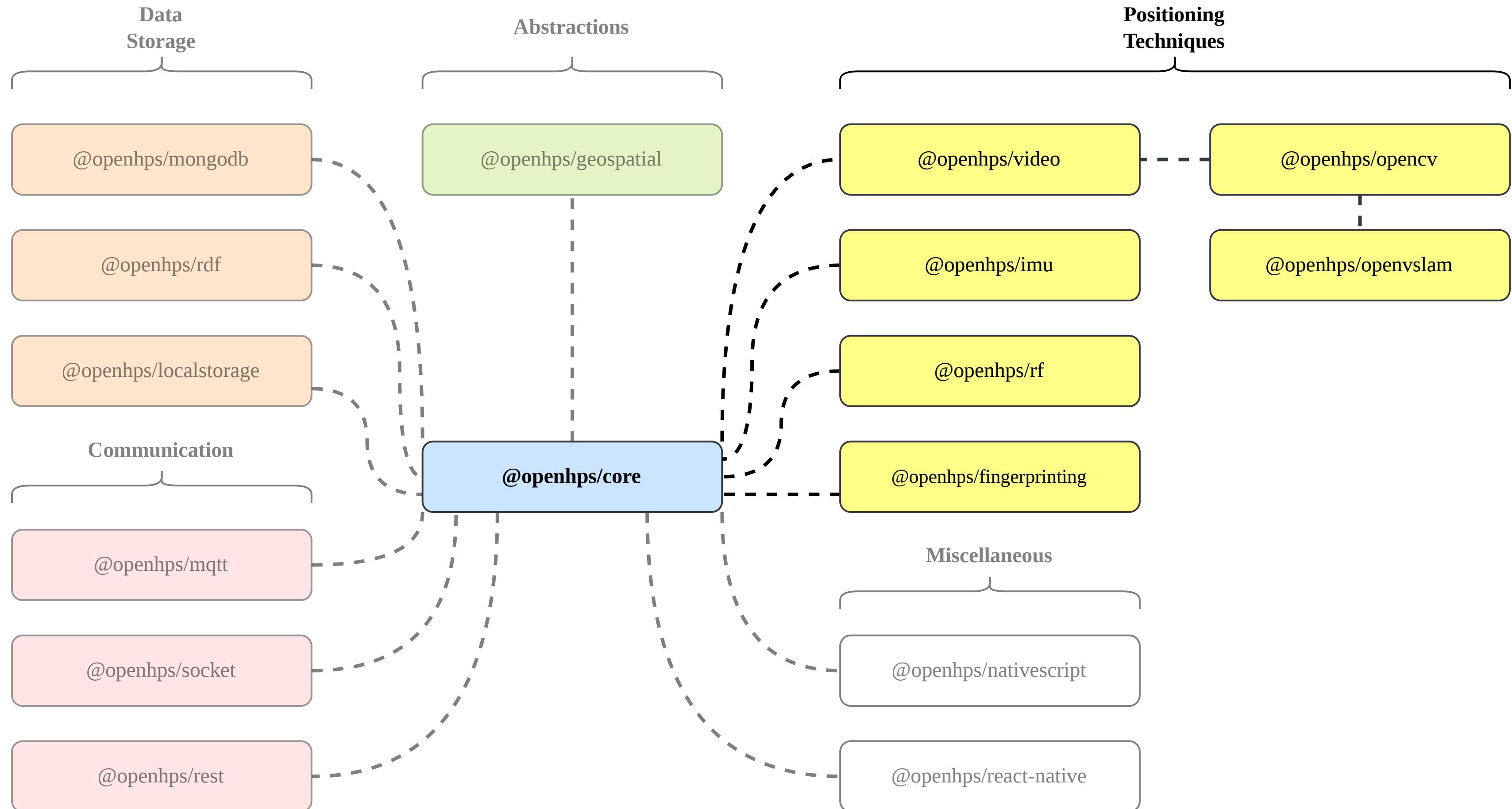
Modularity



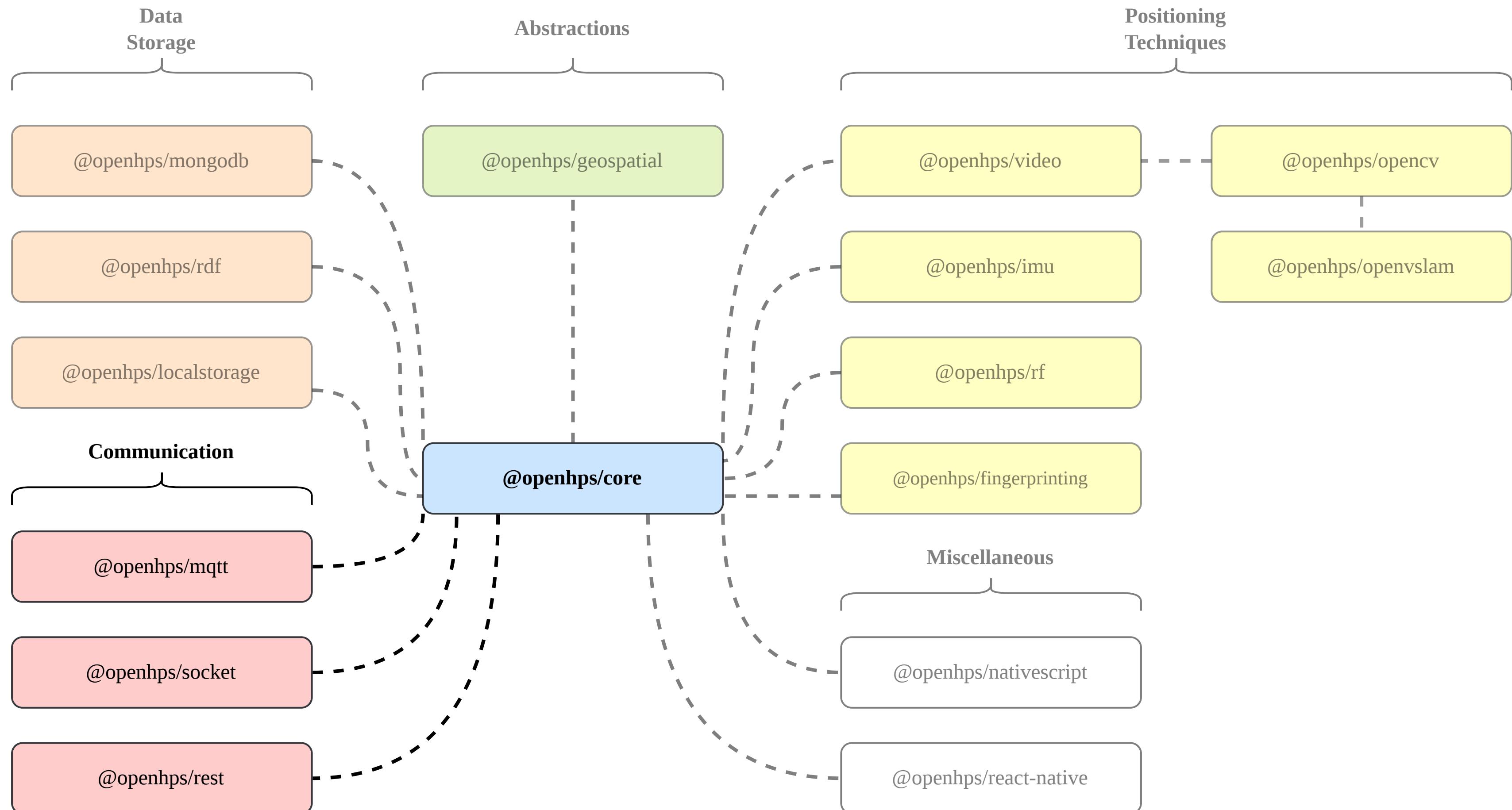
Modularity ...



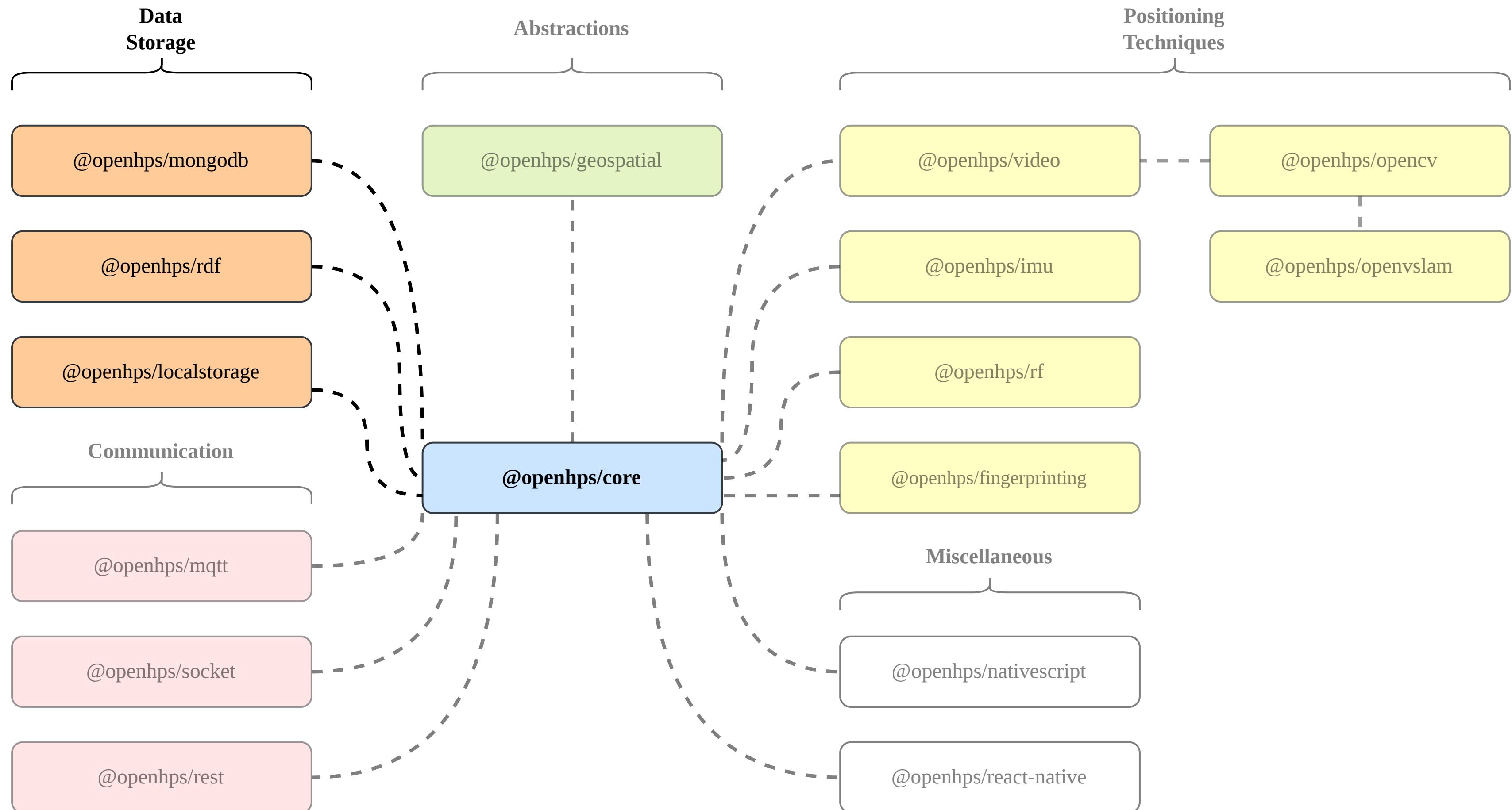
Modularity ...



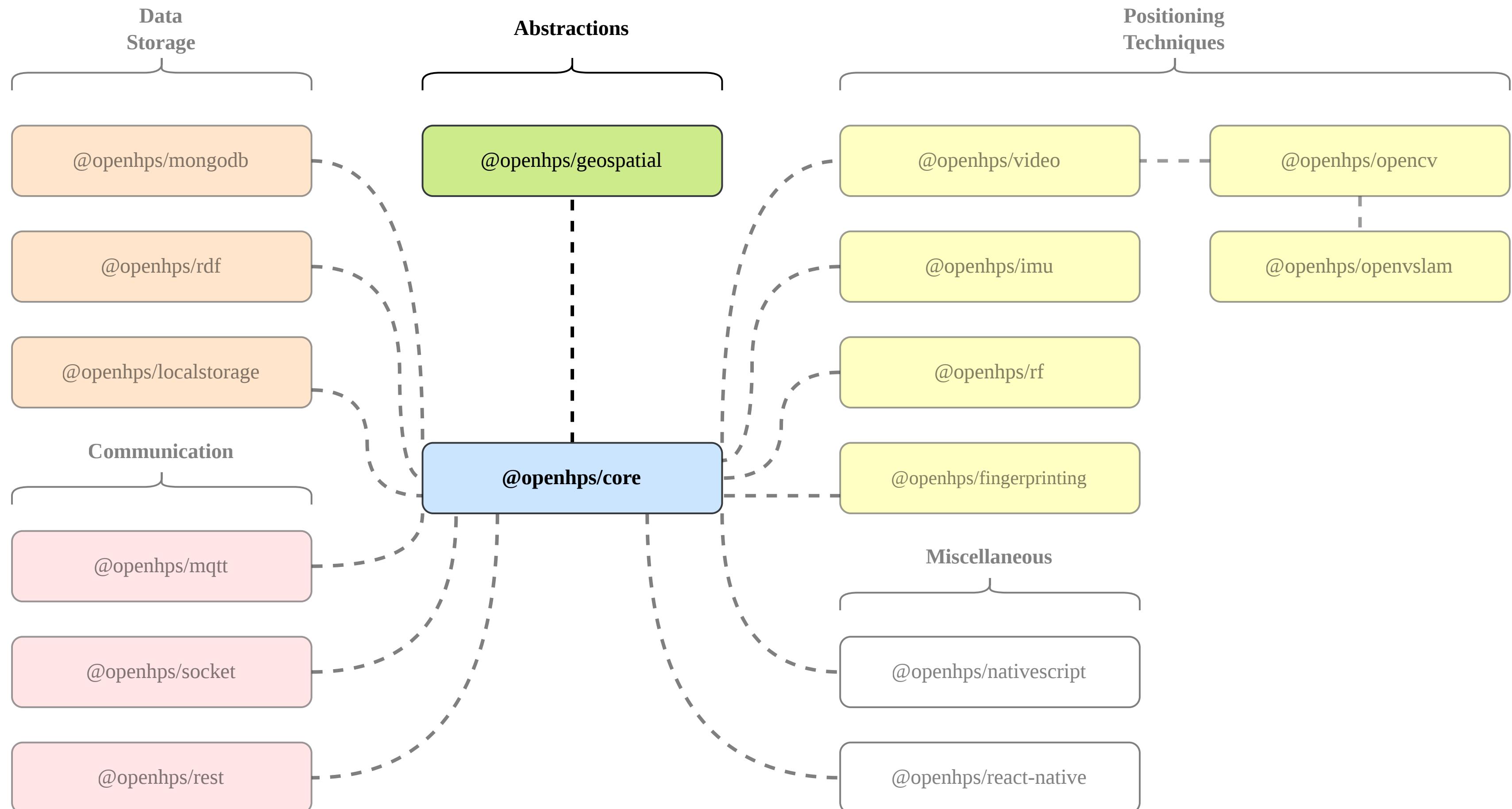
Modularity ...



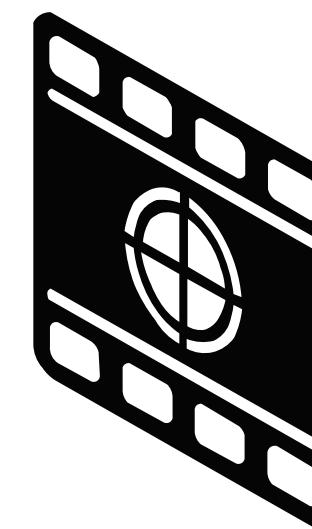
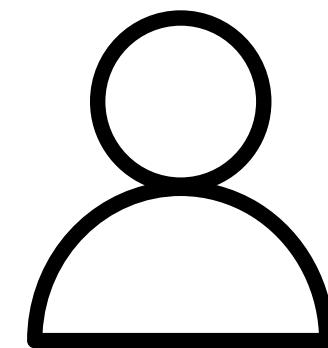
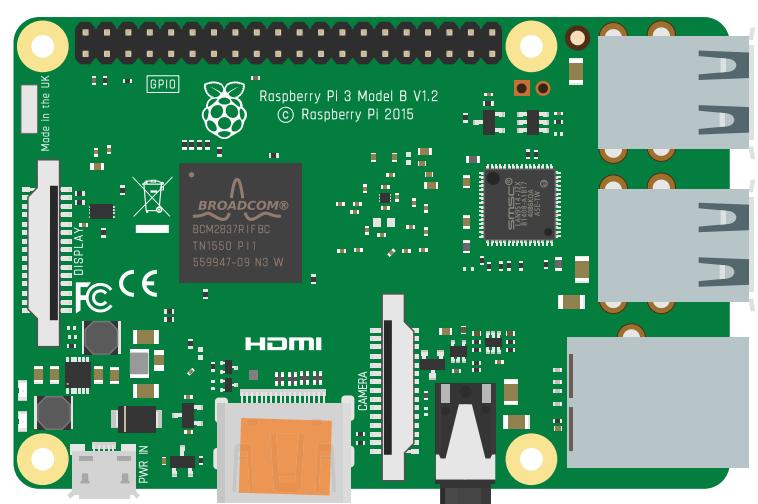
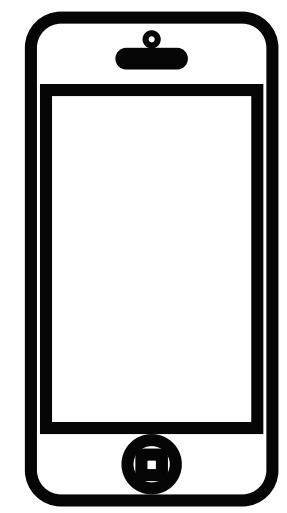
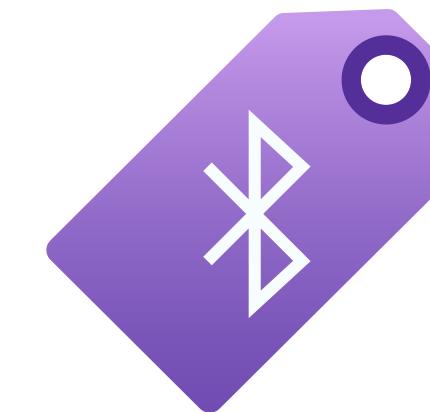
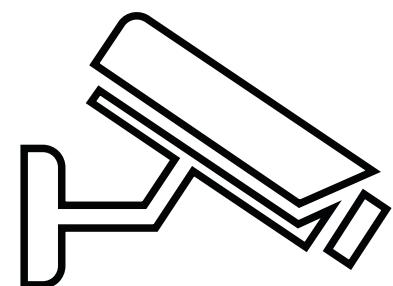
Modularity ...



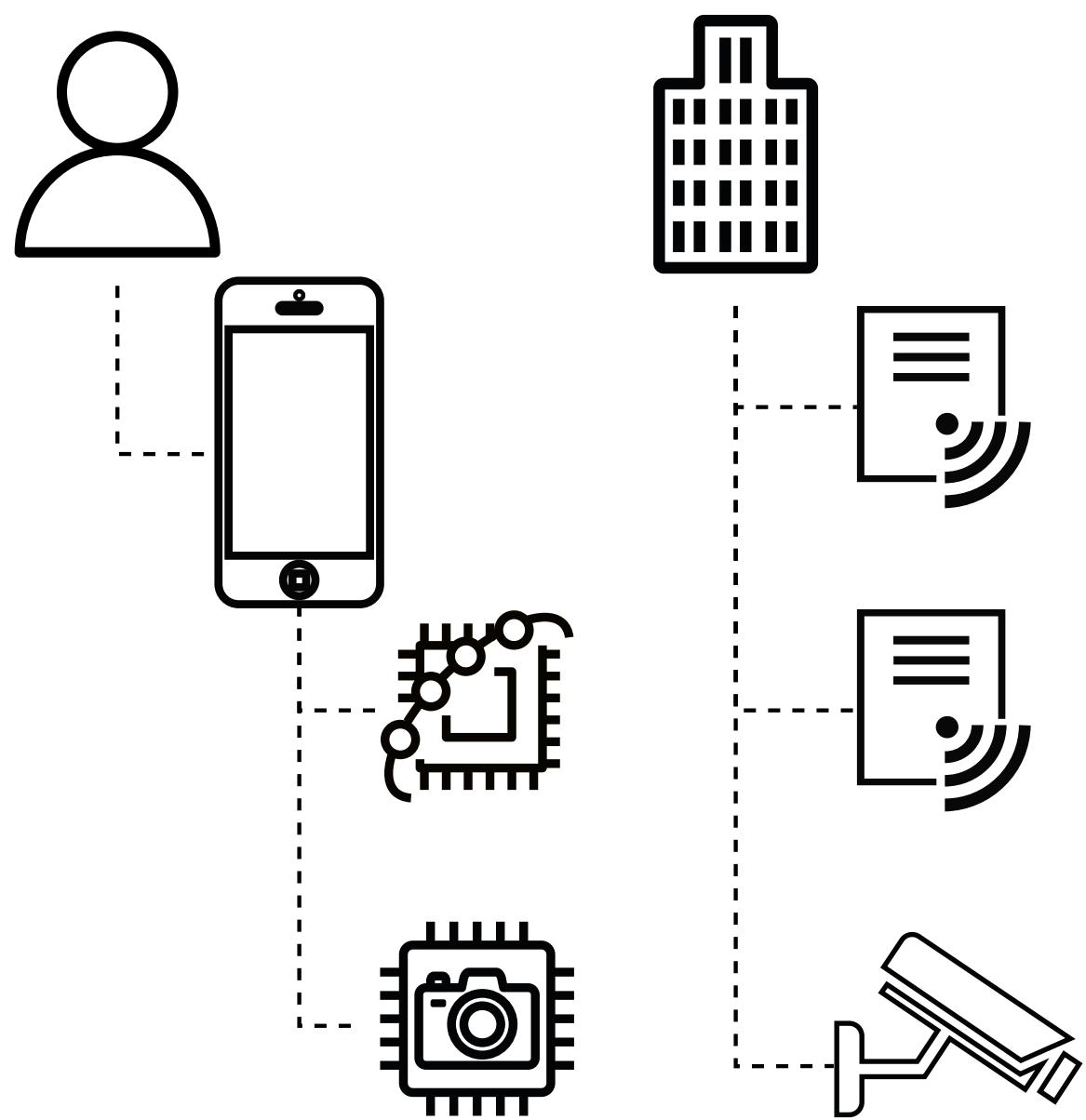
Modularity ...



Data Processing



DataObject



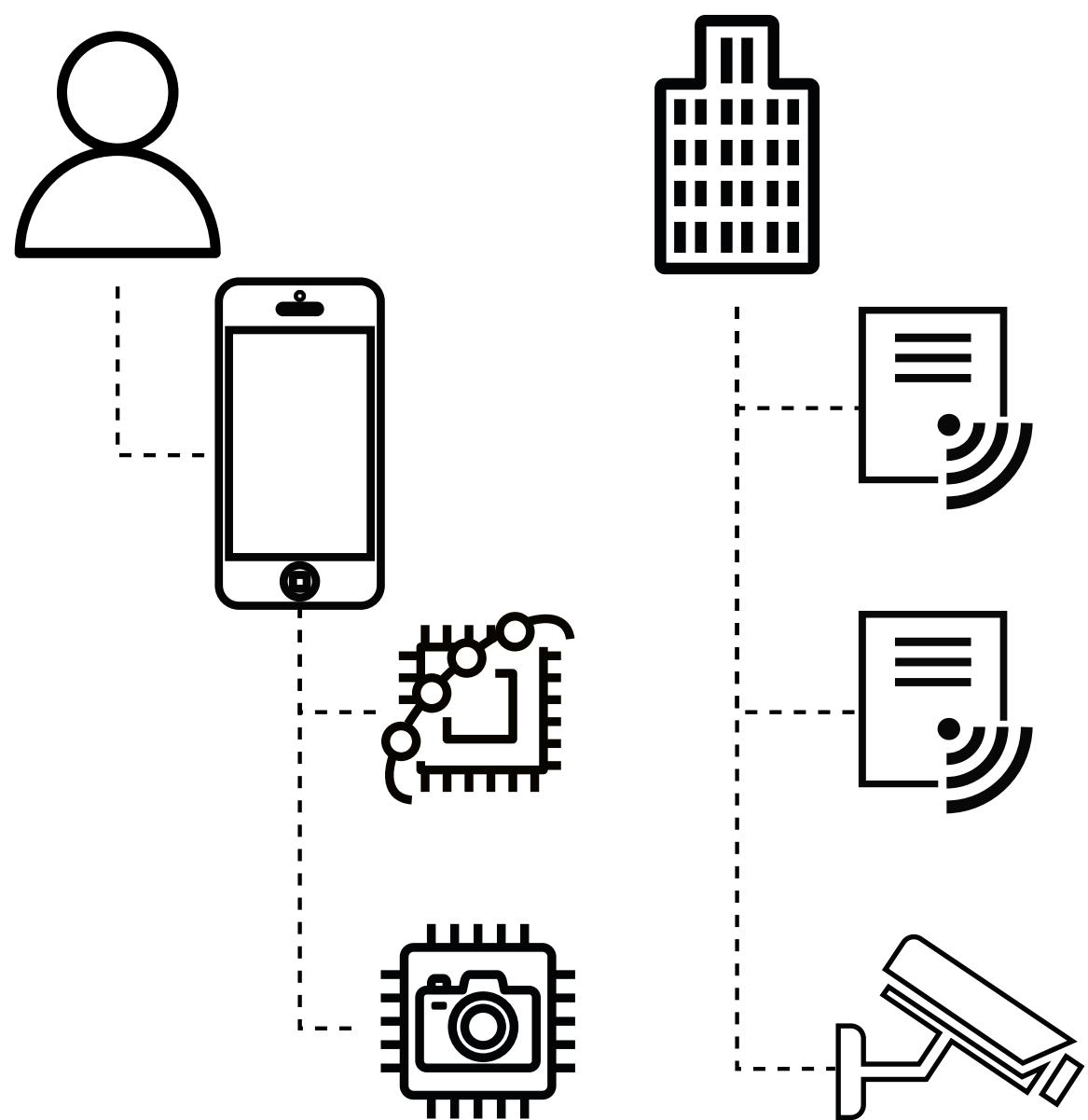
```
// Data object for the person we are tracking
const me = new DataObject("mvdewync@vub.be");
me.displayName = "Maxim Van de Wynckel";

// Phone belonging to the person
const phone = new DataObject()
phone.displayName = "Maxim's Phone";
phone.setParent(me);

// Watch belonging to the person
const watch = new DataObject();
watch.displayName = "Maxim's Android Watch";
watch.setParent(me);

// IP camera identified by MAC
const camera = new CameraObject("80:bb:7c:37:0e:02");
camera.width = 1980;
camera.height = 1024;
camera.fps = 30;
camera.setPosition(/* ... */);
```

DataObject



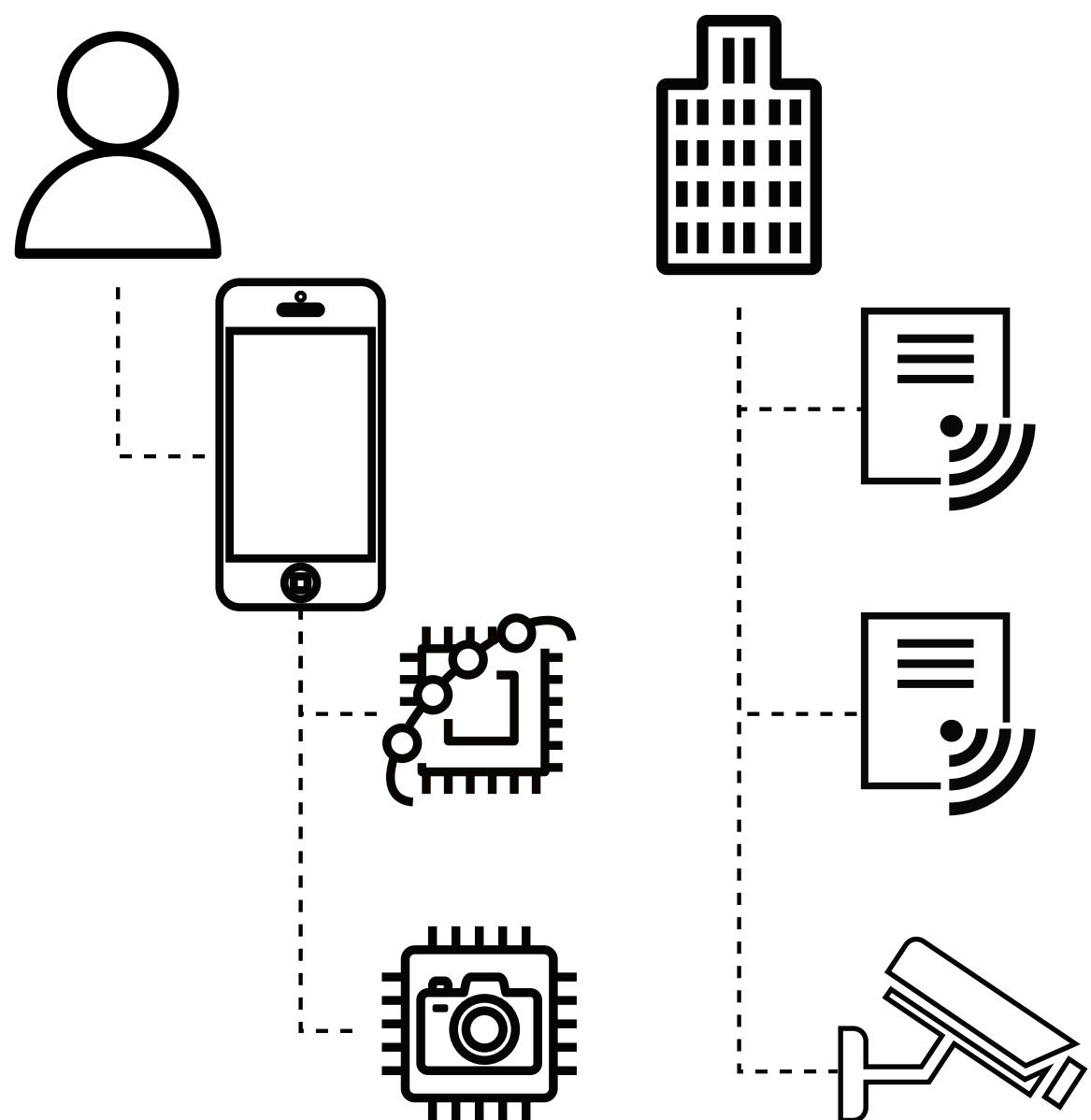
```
// Data object for the person we are tracking
const me = new DataObject("mvdewync@vub.be");
me.displayName = "Maxim Van de Wynckel";

// Phone belonging to the person
const phone = new DataObject();
phone.displayName = "Maxim's Phone";
phone.setParent(me);

// Watch belonging to the person
const watch = new DataObject();
watch.displayName = "Maxim's Android Watch";
watch.setParent(me);

// IP camera identified by MAC
const camera = new CameraObject("80:bb:7c:37:0e:02");
camera.width = 1980;
camera.height = 1024;
camera.fps = 30;
camera.setPosition(/* ... */);
```

DataObject



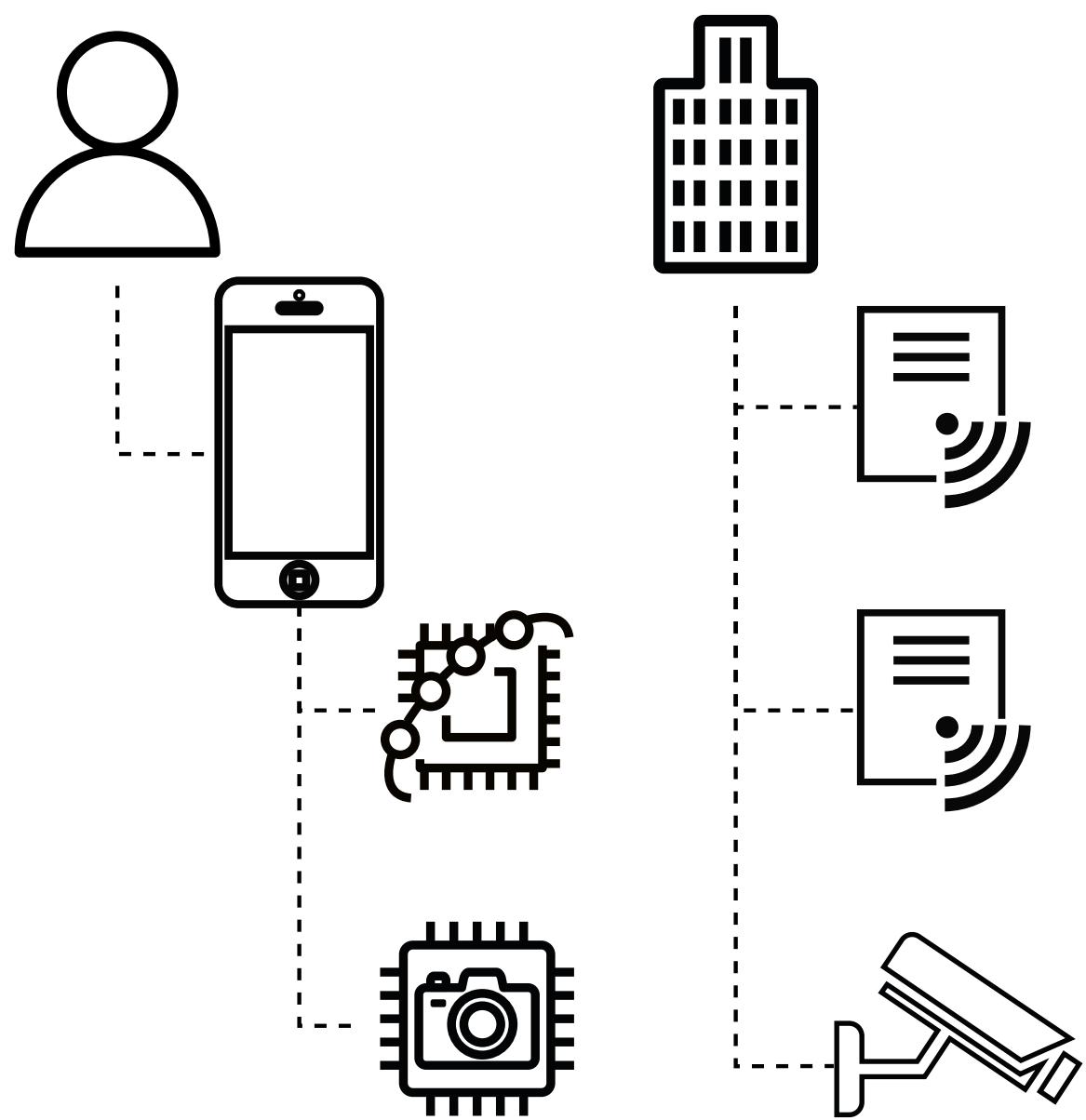
```
// Data object for the person we are tracking
const me = new DataObject("mvdewync@vub.be");
me.displayName = "Maxim Van de Wynckel";

// Phone belonging to the person
const phone = new DataObject();
phone.displayName = "Maxim's Phone";
phone.setParent(me);

// Watch belonging to the person
const watch = new DataObject();
watch.displayName = "Maxim's Android Watch";
watch.setParent(me);

// IP camera identified by MAC
const camera = new CameraObject("80:bb:7c:37:0e:02");
camera.width = 1980;
camera.height = 1024;
camera.fps = 30;
camera.setPosition(/* ... */);
```

DataObject



```
// Data object for the person we are tracking
const me = new DataObject("mvdewync@vub.be");
me.displayName = "Maxim Van de Wynckel";

// Phone belonging to the person
const phone = new DataObject();
phone.displayName = "Maxim's Phone";
phone.setParent(me);

// Watch belonging to the person
const watch = new DataObject();
watch.displayName = "Maxim's Android Watch";
watch.setParent(me);

// IP camera identified by MAC
const camera = new CameraObject("80:bb:7c:37:0e:02");
camera.width = 1980;
camera.height = 1024;
camera.fps = 30;
camera.setPosition(/* ... */);
```

Absolute and Relative Positions



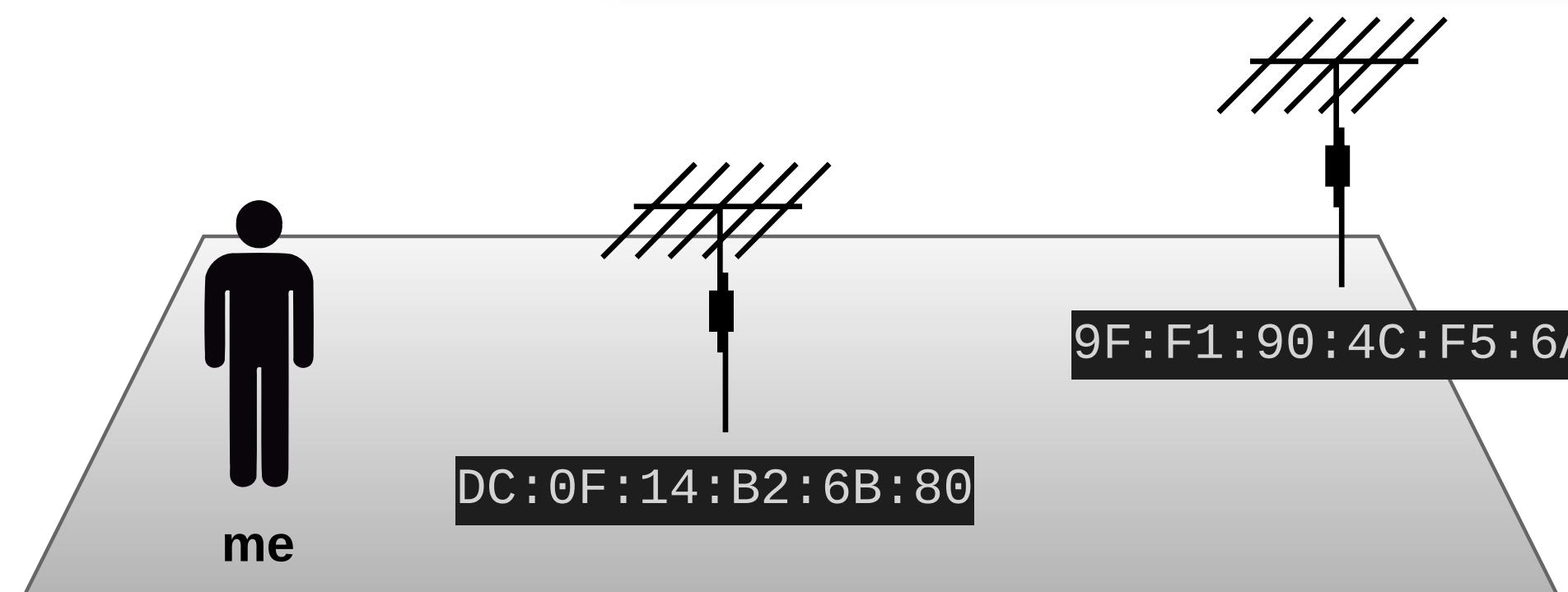
Absolute

- ▶ 2D, 3D, geographical, ...
- ▶ Within a reference space

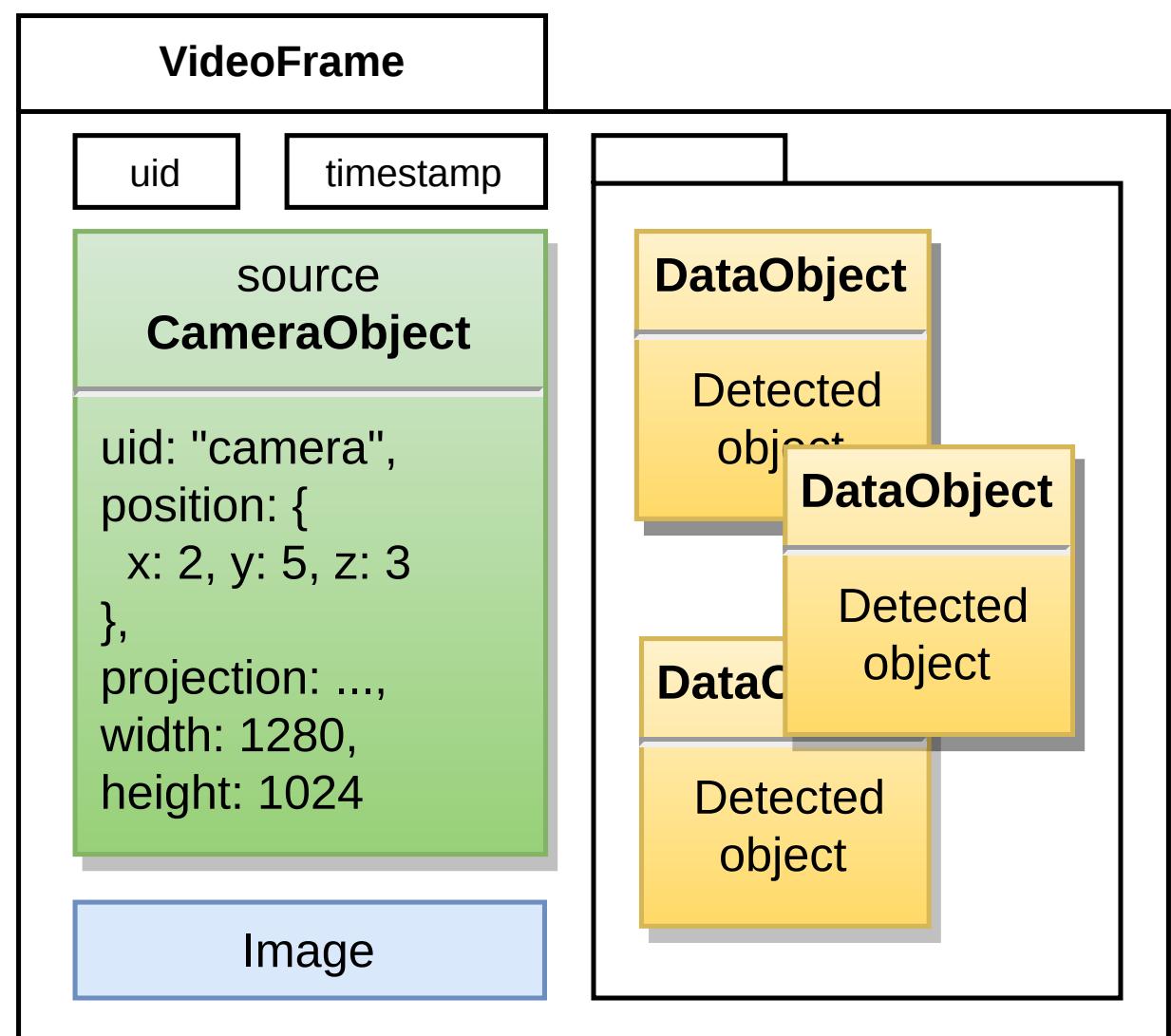
Relative

- ▶ Distance, angle, velocity, ...
- ▶ Relative to another *object*

```
// Absolute geographical position  
me.setPosition(new GeographicalPosition(  
    50.8204, 4.3921  
));  
  
// Relative position(s) to another object  
me.addRelativePosition(new RelativeDistance(  
    "9F:F1:90:4C:F5:6A", 5.2, LengthUnit.METER  
));  
me.addRelativePosition(new RelativeDistance(  
    "DC:0F:14:B2:6B:80", 1.4, LengthUnit.METER  
));
```



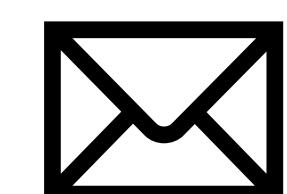
DataFrame



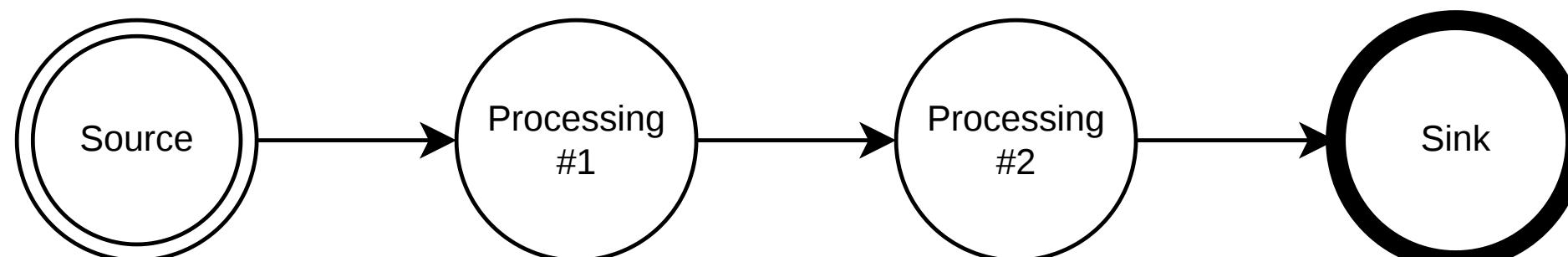
```
// Sensor that captured the frame
const camera = new CameraObject();

// Create a new frame
const frame = new VideoFrame();
frame.source = camera;
frame.image = myImage;

// Add detected objects to frame
frame.addObject(/* ... */);
frame.addObject(/* ... */);
frame.addObject(/* ... */);
```



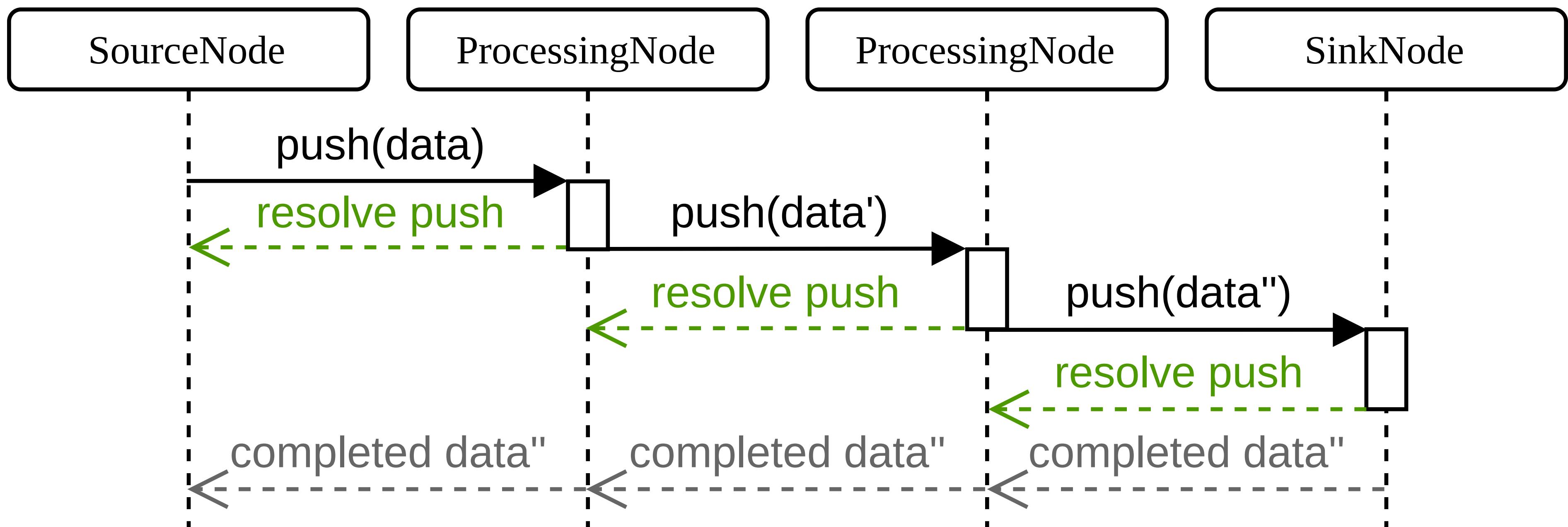
DataFrame



DataFrame ...



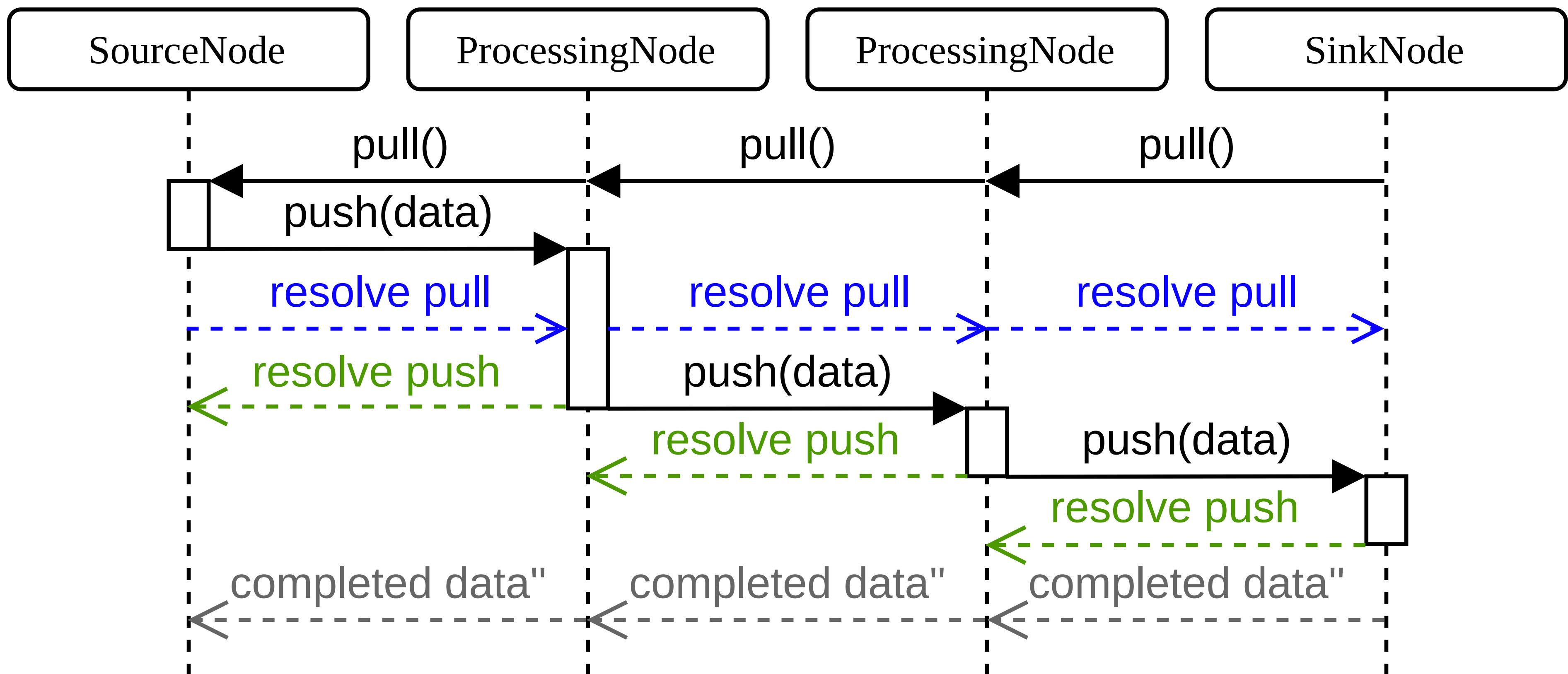
Pushing Data



DataFrame ...



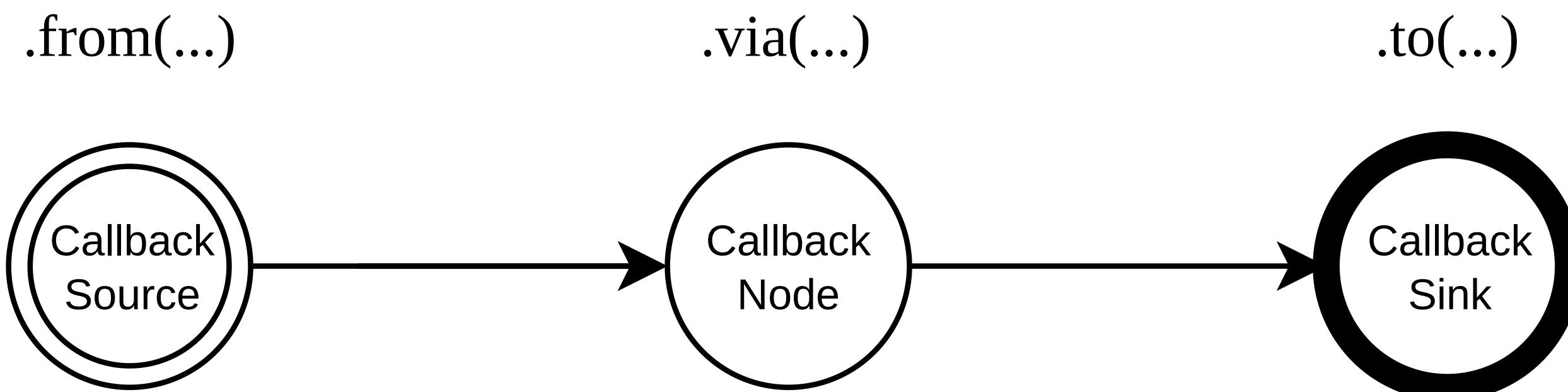
Pulling Data



Positioning Model



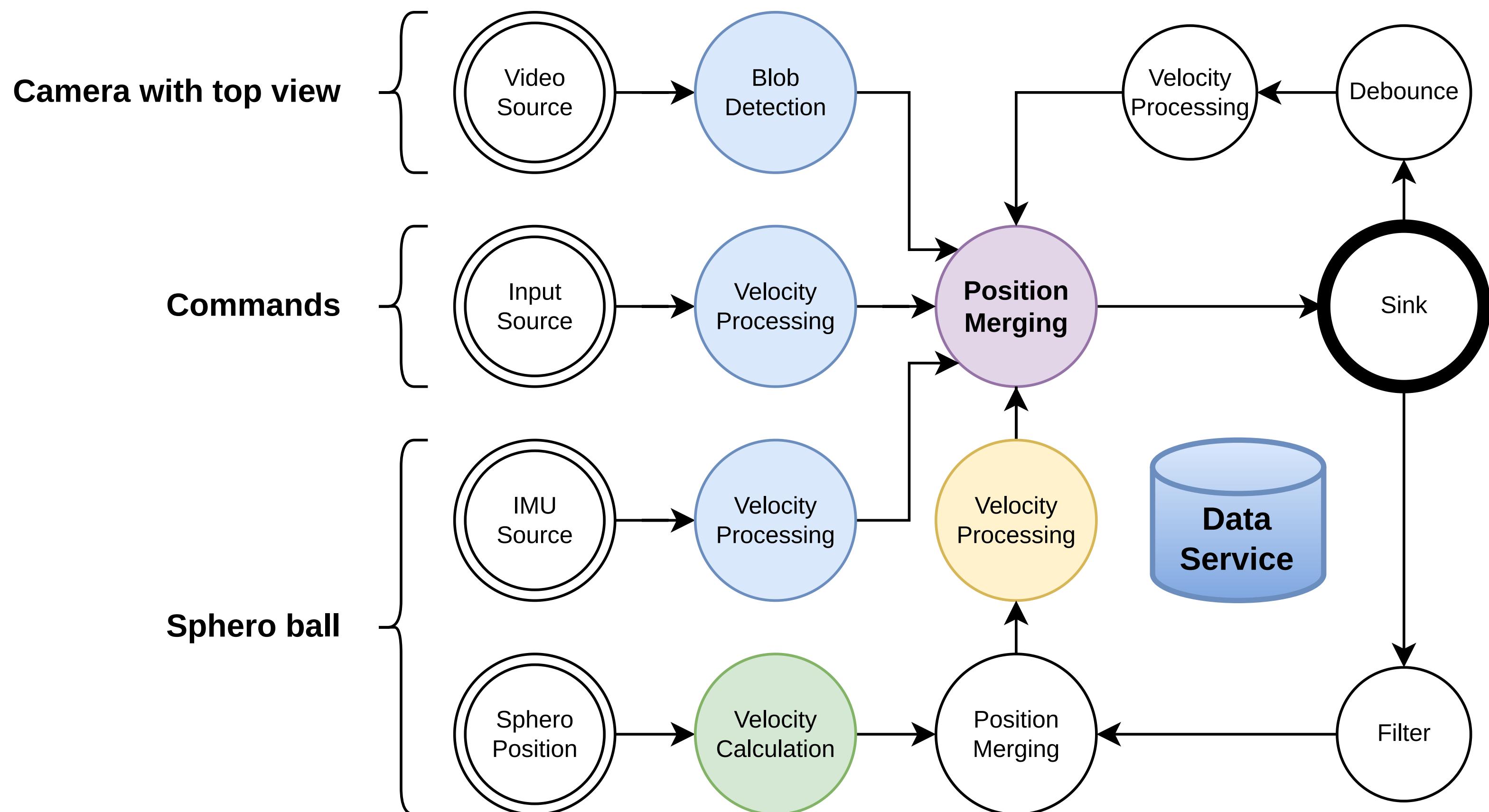
```
ModelBuilder.create()
  .from(new CallbackSourceNode(() => {
    const myObject = new DataObject("mvdewync");
    const frame = new DataFrame();
    frame.addObject(myObject);
    return frame;
  )))
  .via(new CallbackNode((frame: DataFrame) => { /* ... */ }))
  .to(new CallbackSinkNode((frame: DataFrame) => { /* ... */ }))
  .build().then((model: Model) => { /* ... */});
```



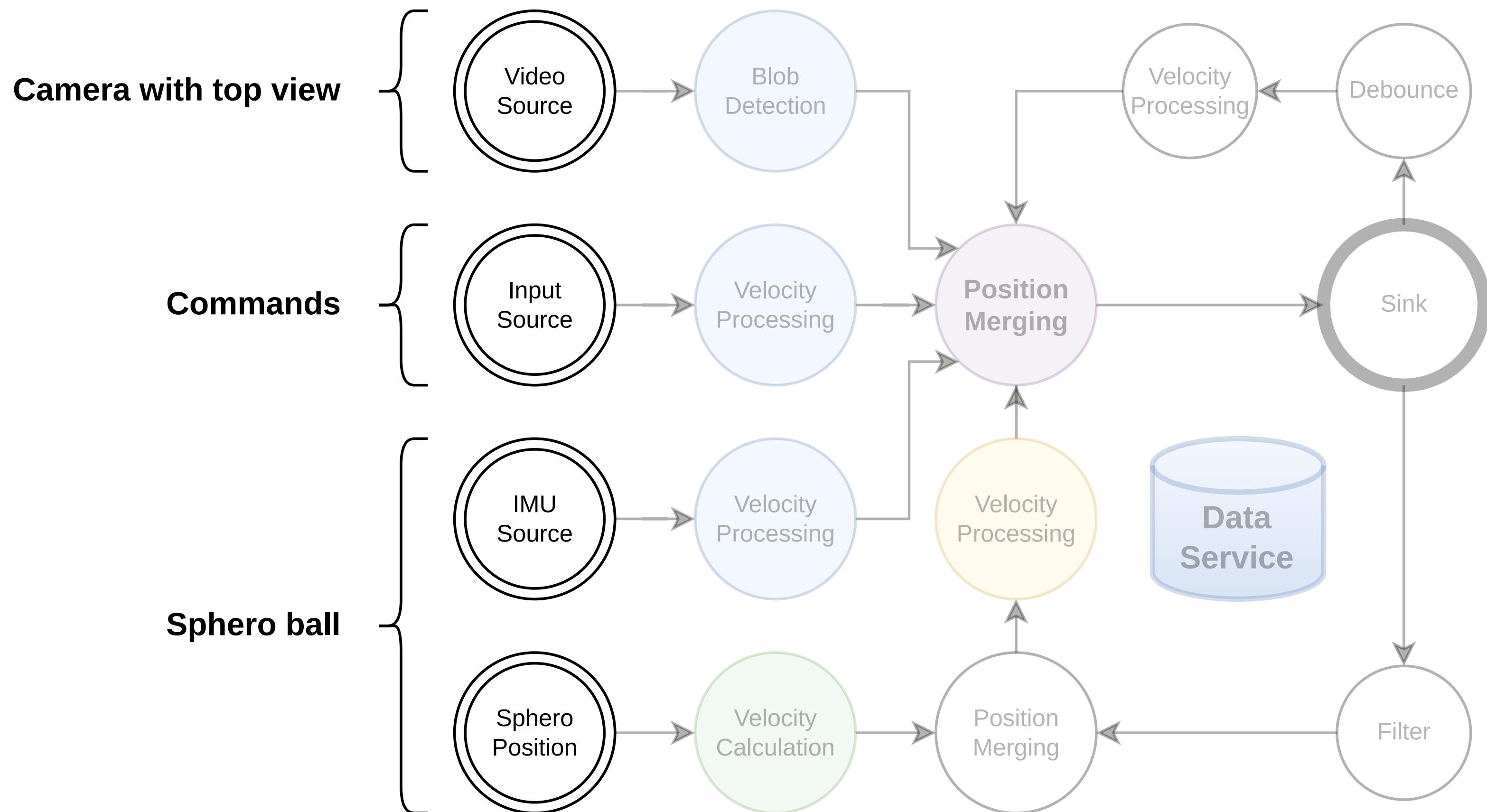
Example



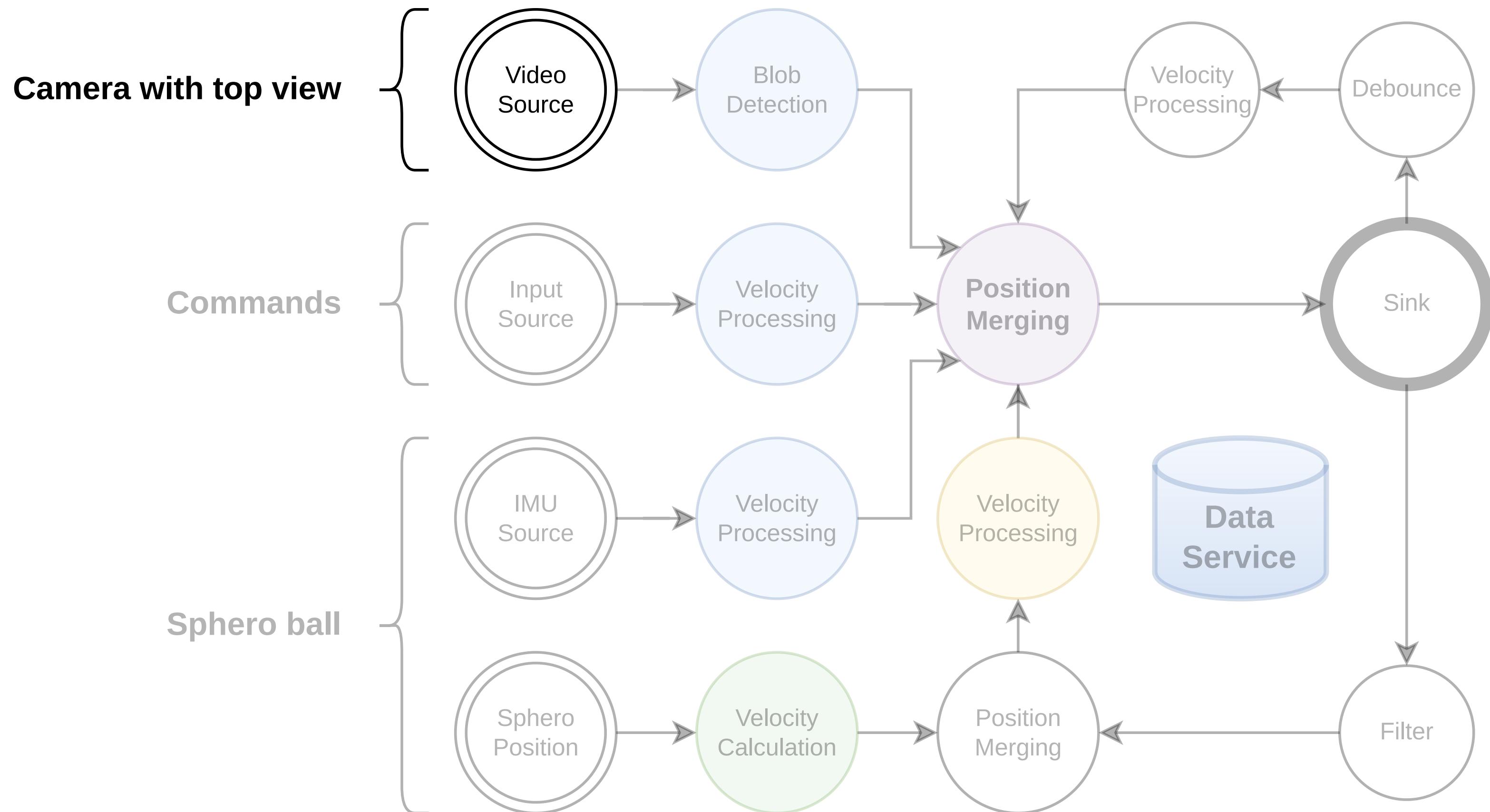
Example ...



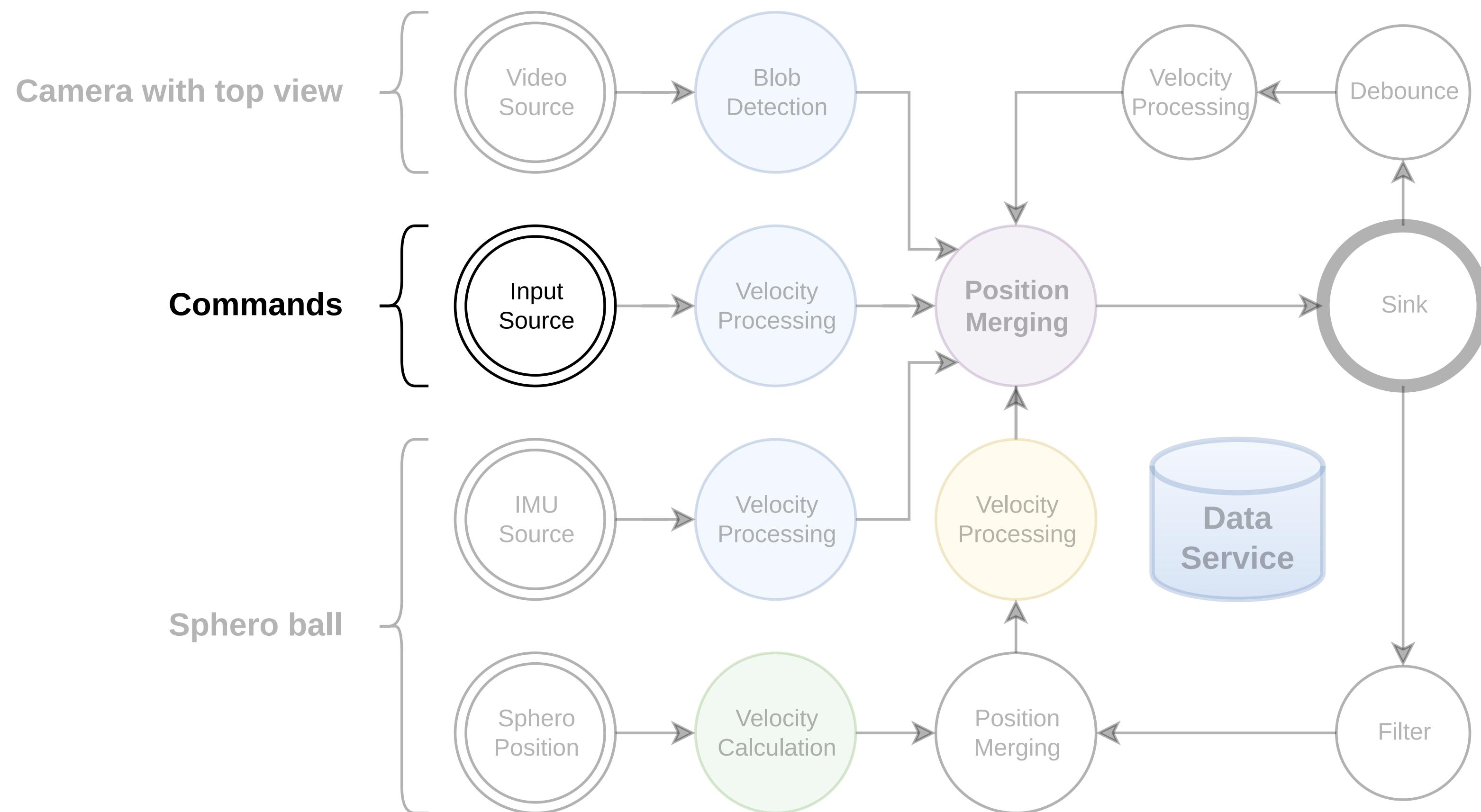
Example ...



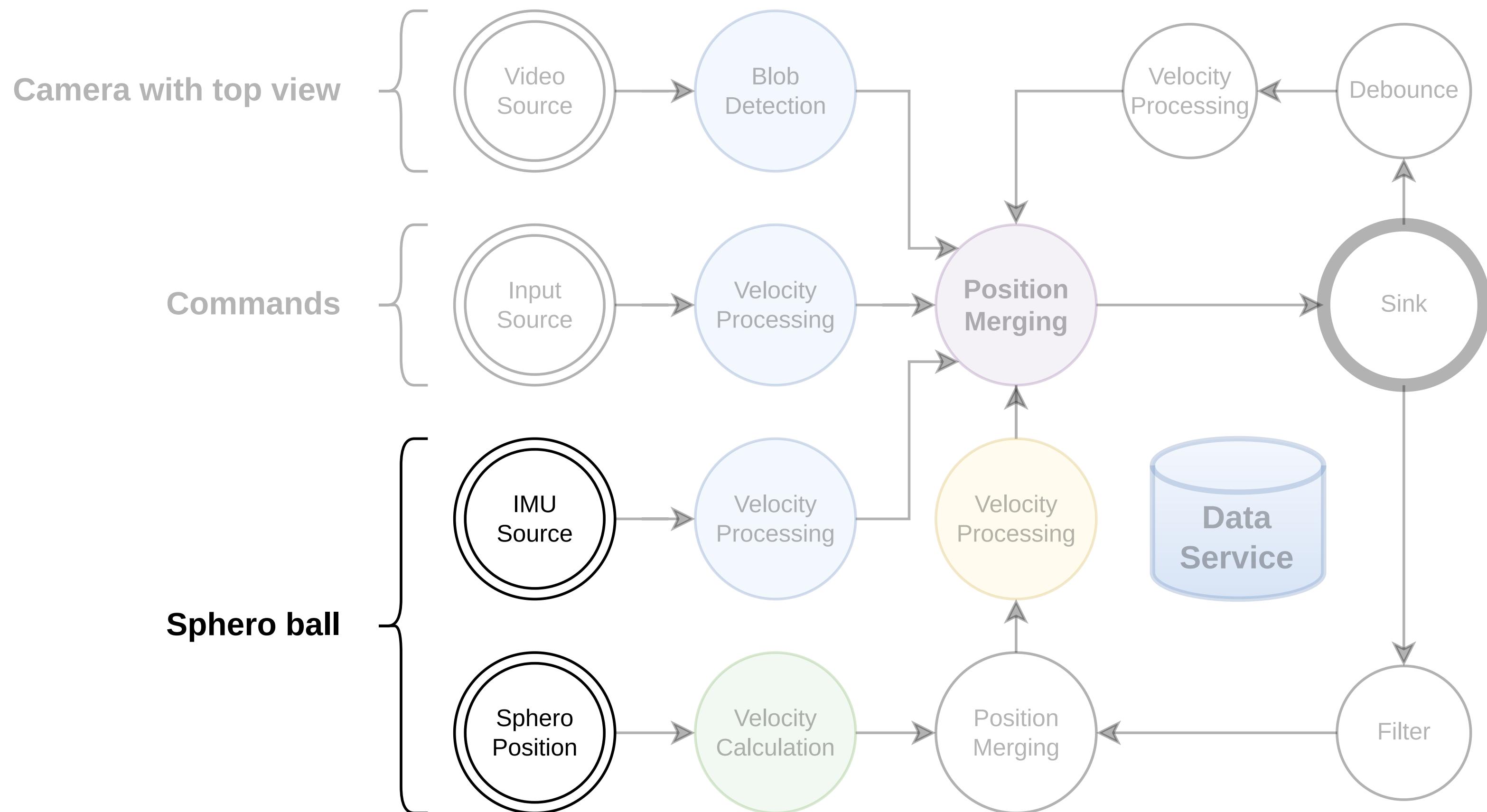
Example ...



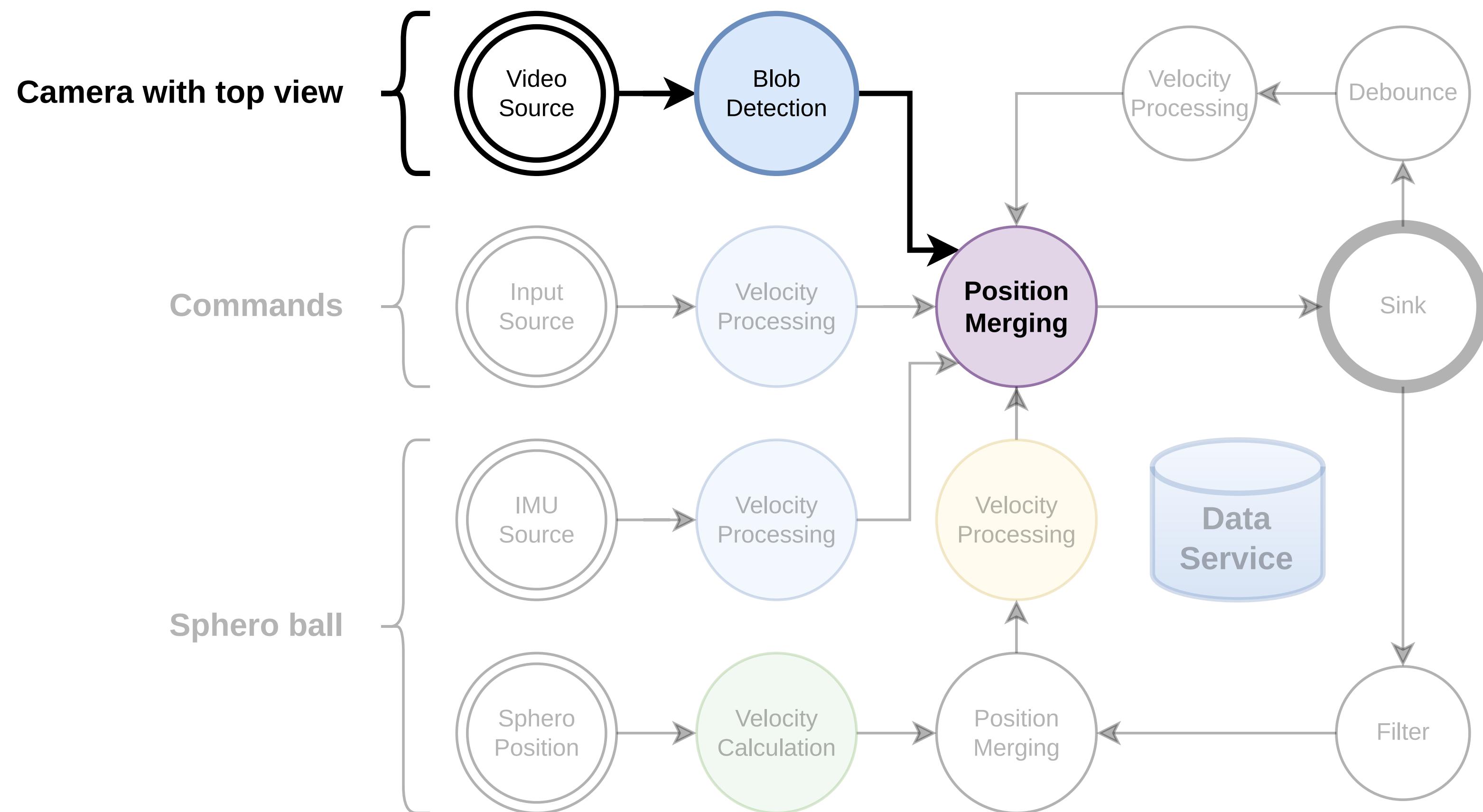
Example ...



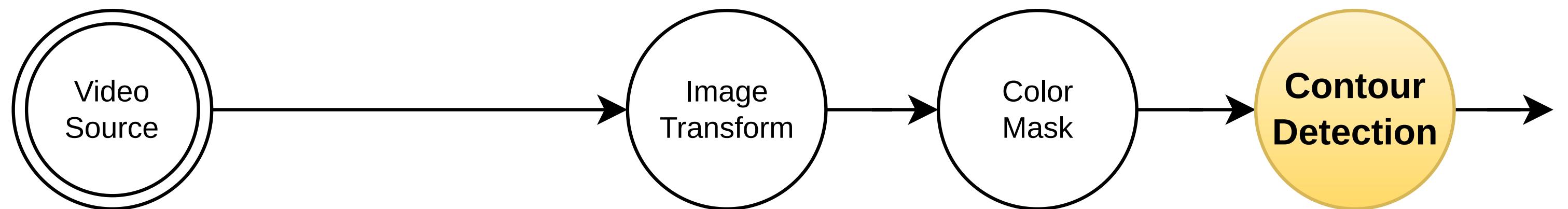
Example ...



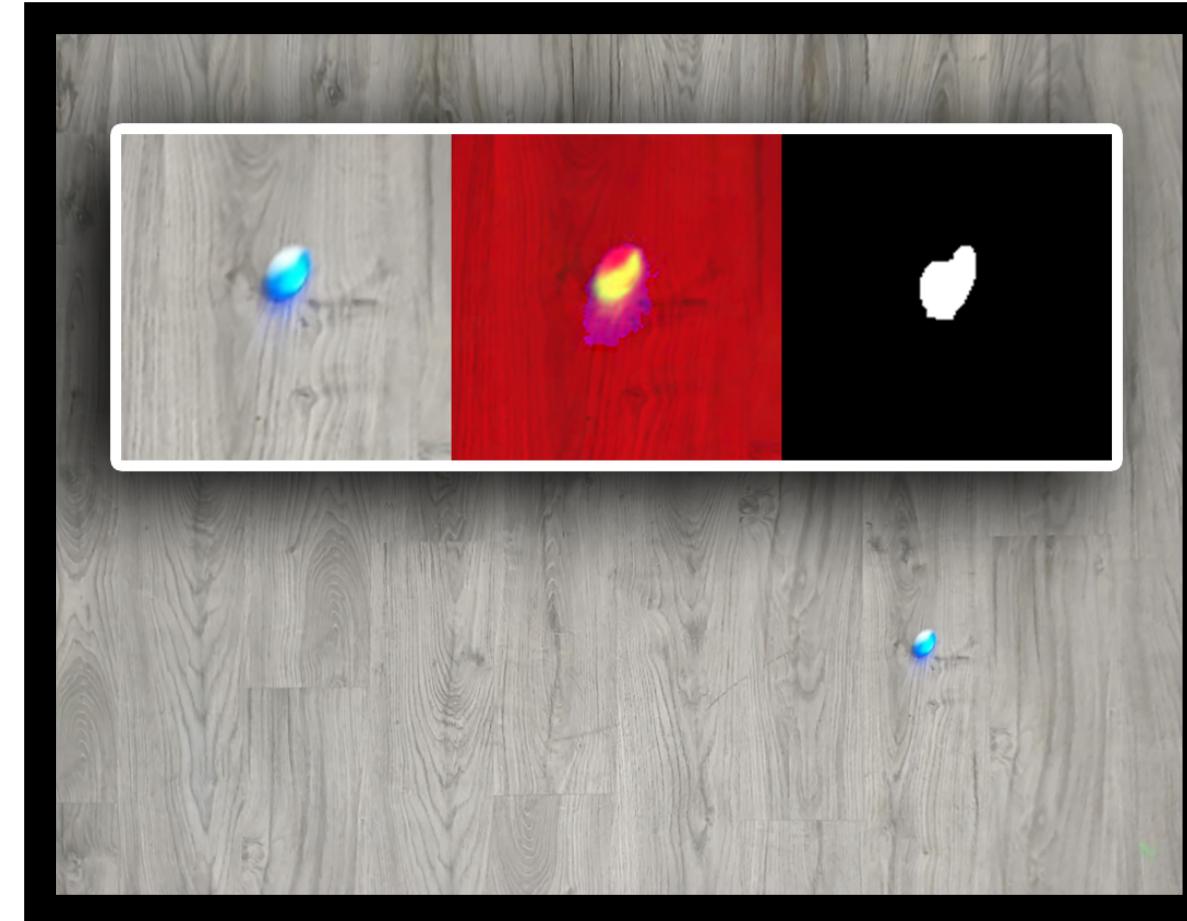
Example ...



Example ...

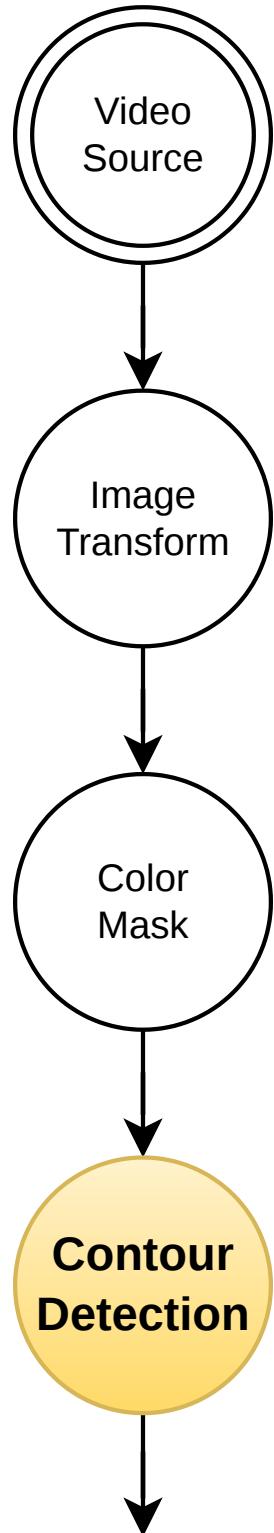


Original



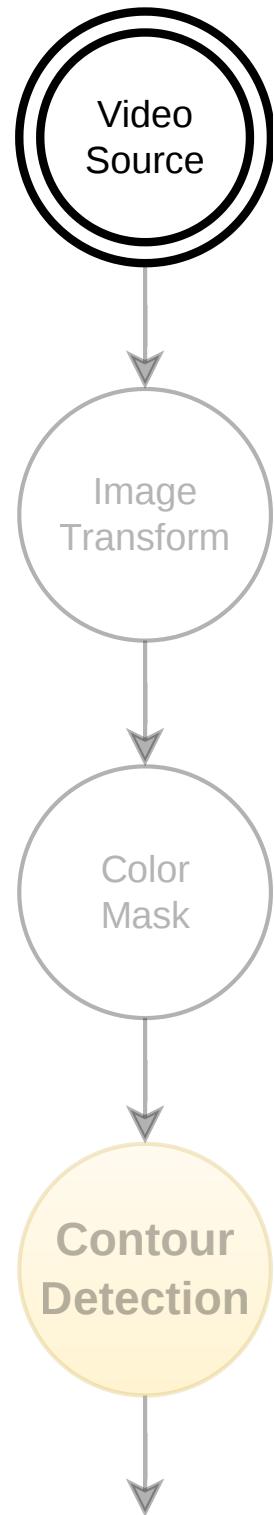
Transformed
& Mask

Example ...



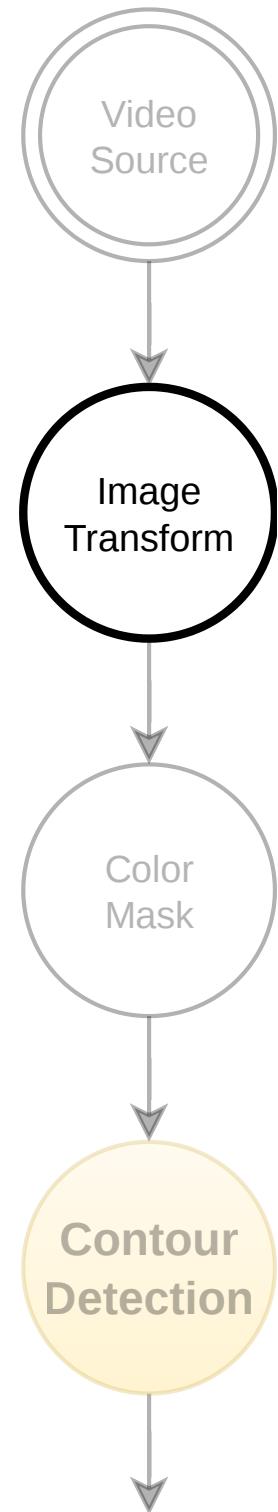
```
GraphBuilder.create()
    .from(new VideoSource({
        autoPlay: true,
        fps: 30,
        throttleRead: true,
        source: new CameraObject("sphero_video")
    }).load("/dev/video2"))
    .via(new ImageTransformNode({
        src: [
            new OpenCV.Point2(307, 120),
            new OpenCV.Point2(1473, 87),
            new OpenCV.Point2(1899, 891),
            new OpenCV.Point2(20, 1024),
        ],
        height: 800,
        width: 1040
    }))
    .via(new ColorMaskProcessing({
        minRange: [90, 50, 50],
        maxRange: [140, 255, 255]
    }))
    .via(new ContourDetectionNode()) // Custom
    .to();
```

Example ...



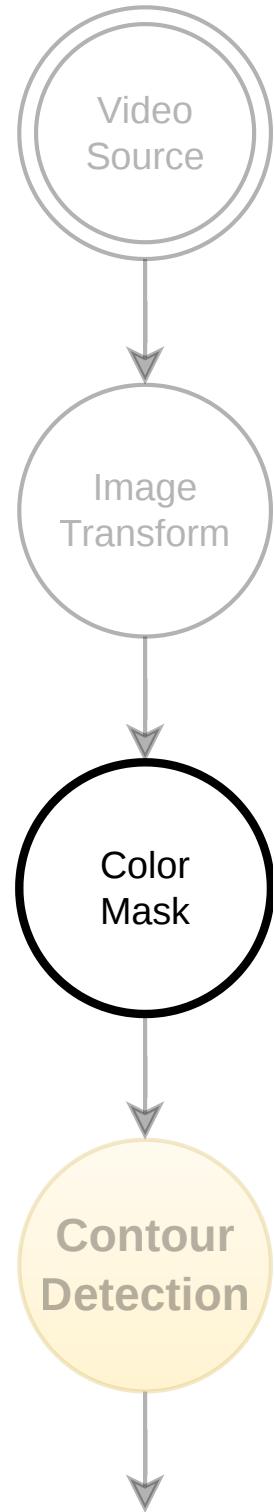
```
GraphBuilder.create()
    .from(new VideoSource({
        autoplay: true,
        fps: 30,
        throttleRead: true,
        source: new CameraObject("sphero_video")
    }).load("/dev/video2"))
    .via(new ImageTransformNode({
        src: [
            new OpenCV.Point2(307, 120),
            new OpenCV.Point2(1473, 87),
            new OpenCV.Point2(1899, 891),
            new OpenCV.Point2(20, 1024),
        ],
        height: 800,
        width: 1040
    }))
    .via(new ColorMaskProcessing({
        minRange: [90, 50, 50],
        maxRange: [140, 255, 255]
    }))
    .via(new ContourDetectionNode()) // Custom
    .to();
```

Example ...



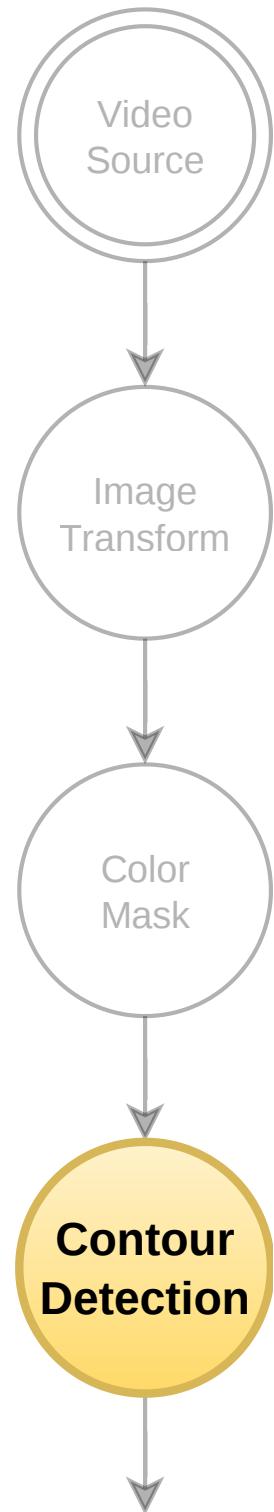
```
GraphBuilder.create()
    .from(new VideoSource({
        autoPlay: true,
        fps: 30,
        throttleRead: true,
        source: new CameraObject("sphero_video")
    }).load("/dev/video2"))
    .via(new ImageTransformNode({
        src: [
            new OpenCV.Point2(307, 120),
            new OpenCV.Point2(1473, 87),
            new OpenCV.Point2(1899, 891),
            new OpenCV.Point2(20, 1024),
        ],
        height: 800,
        width: 1040
    }))
    .via(new ColorMaskProcessing({
        minRange: [90, 50, 50],
        maxRange: [140, 255, 255]
    }))
    .via(new ContourDetectionNode()) // Custom
    .to();
```

Example ...



```
GraphBuilder.create()
    .from(new VideoSource({
        autoPlay: true,
        fps: 30,
        throttleRead: true,
        source: new CameraObject("sphero_video")
    }).load("/dev/video2"))
    .via(new ImageTransformNode({
        src: [
            new OpenCV.Point2(307, 120),
            new OpenCV.Point2(1473, 87),
            new OpenCV.Point2(1899, 891),
            new OpenCV.Point2(20, 1024),
        ],
        height: 800,
        width: 1040
    }))
    .via(new ColorMaskProcessing({
        minRange: [90, 50, 50],
        maxRange: [140, 255, 255]
    }))
    .via(new ContourDetectionNode()) // Custom
    .to();
```

Example ...



```
GraphBuilder.create()
    .from(new VideoSource({
        autoPlay: true,
        fps: 30,
        throttleRead: true,
        source: new CameraObject("sphero_video")
    }).load("/dev/video2"))
    .via(new ImageTransformNode({
        src: [
            new OpenCV.Point2(307, 120),
            new OpenCV.Point2(1473, 87),
            new OpenCV.Point2(1899, 891),
            new OpenCV.Point2(20, 1024),
        ],
        height: 800,
        width: 1040
    }))
    .via(new ColorMaskProcessing({
        minRange: [90, 50, 50],
        maxRange: [140, 255, 255]
    }))
    .via(new ContourDetectionNode()) // Custom
    .to();
```

OpenCV: Contour Detection



```
class ContourDetectionNode extends ProcessingNode<VideoFrame> {
    process(frame: VideoFrame): Promise<VideoFrame> {
        return new Promise((resolve) => {
            let contours = frame.image.findContours(
                OpenCV.RETR_EXTERNAL, OpenCV.CHAIN_APPROX_SIMPLE);
            if (contours.length >= 1) {
                // Sort contours by area and select largest area as 'ball'
                contours = contours.sort((a, b) => a.area - b.area);
                const m = contours[0].moments();
                const center = new OpenCV.Vec2(m.m10 / m.m00, m.m01 / m.m00);
                // Use the center as the 2D pixel position
                const position = new Absolute2DPosition(center.x, center.y);
                position.unit = LengthUnit.CENTIMETER;
                position.accuracy = Math.sqrt(contours[0].area);
                frame addObject(new DataObject("ball").setPosition(position));
            }
            resolve(frame);
        });
    }
}
```

OpenCV: Contour Detection

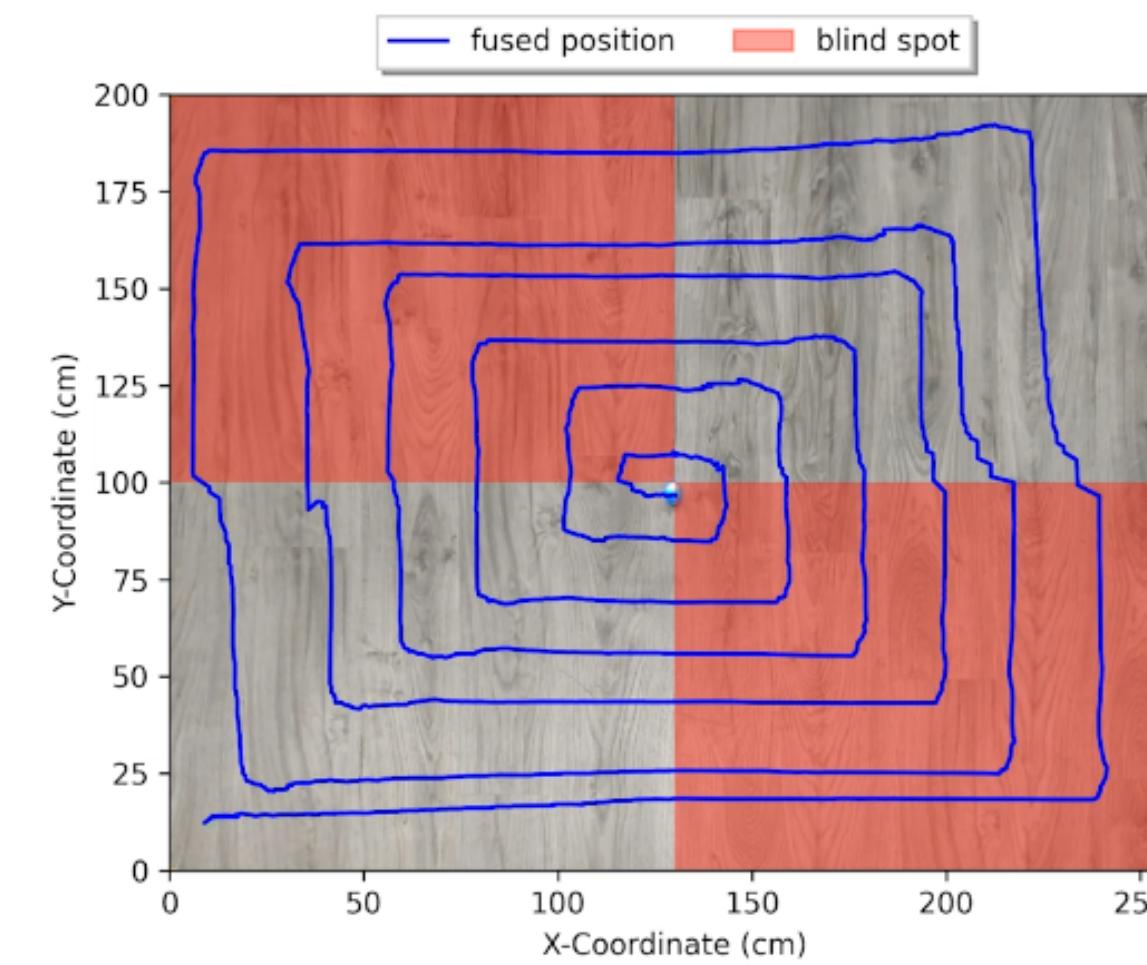
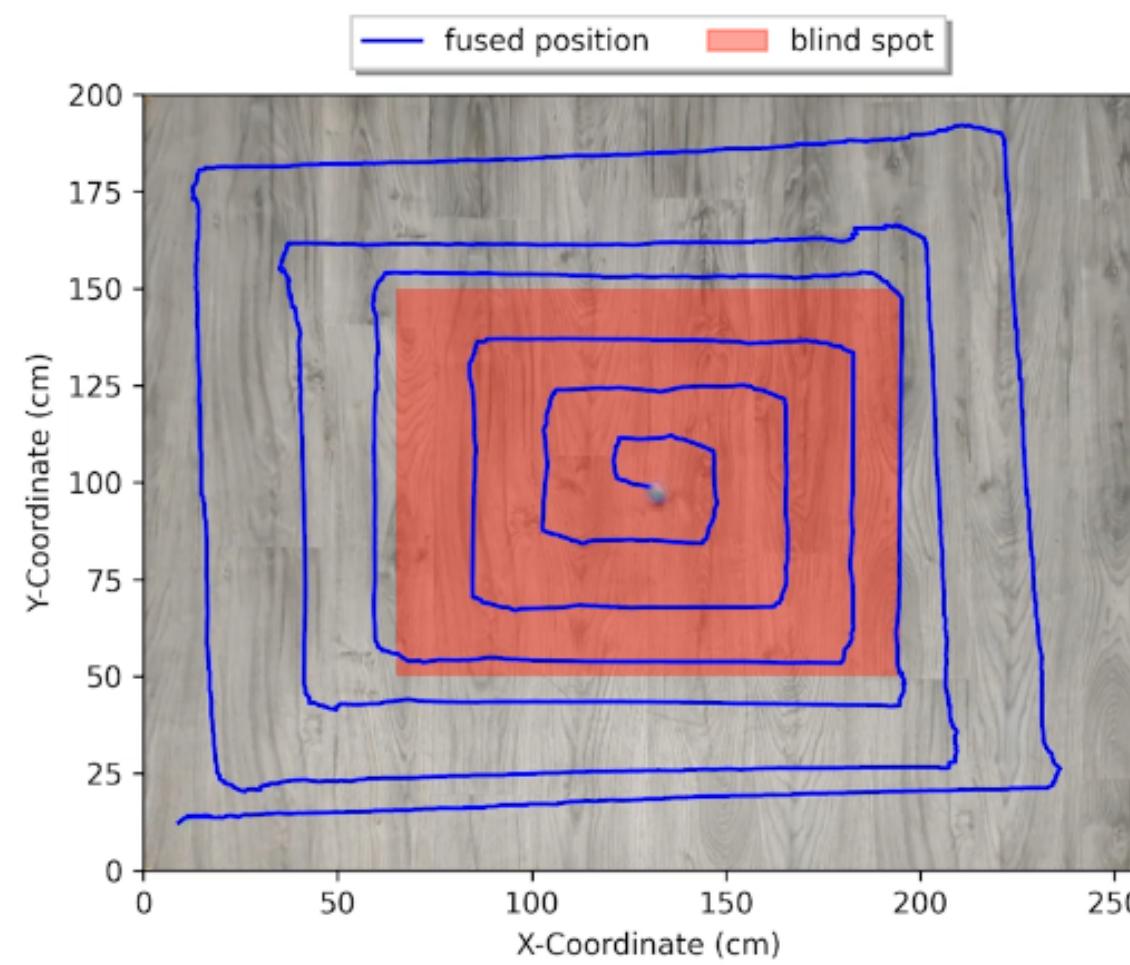
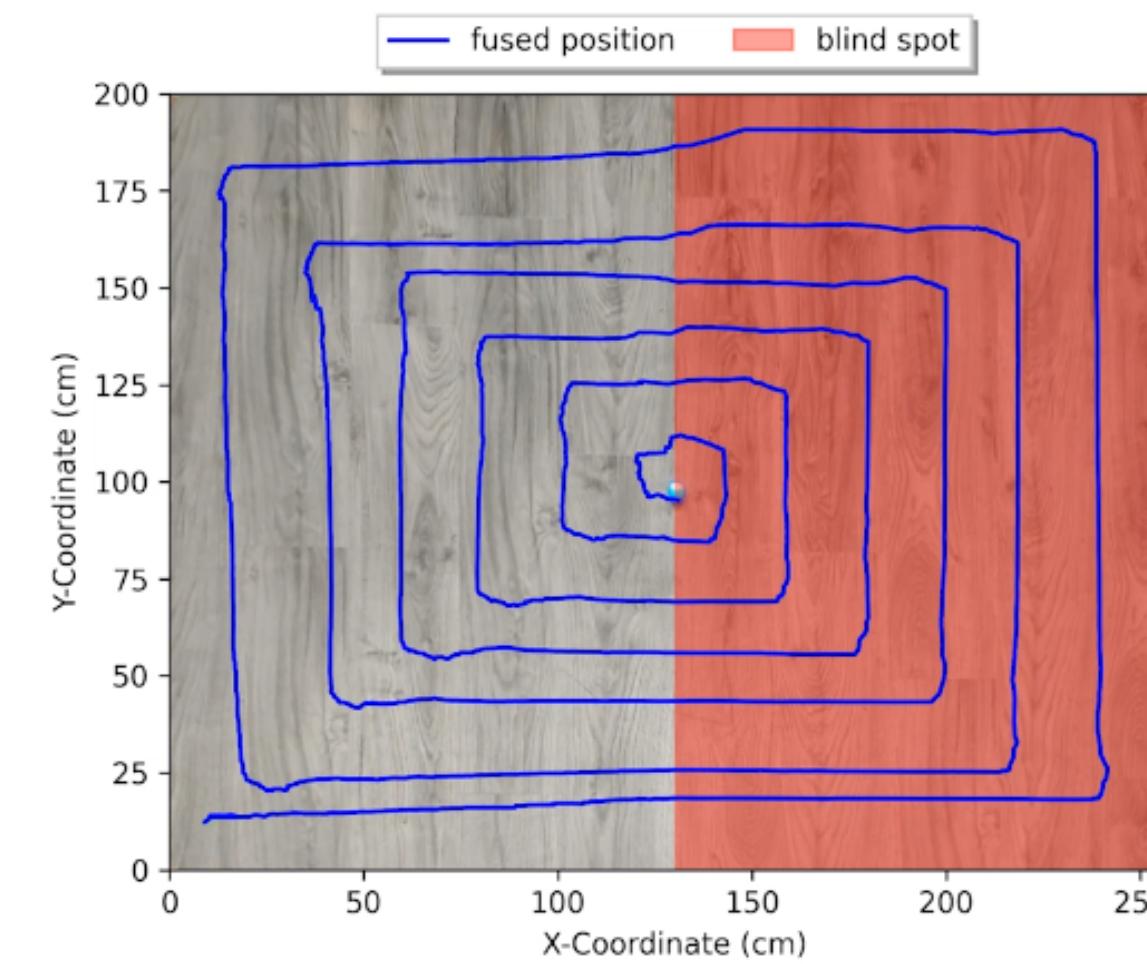
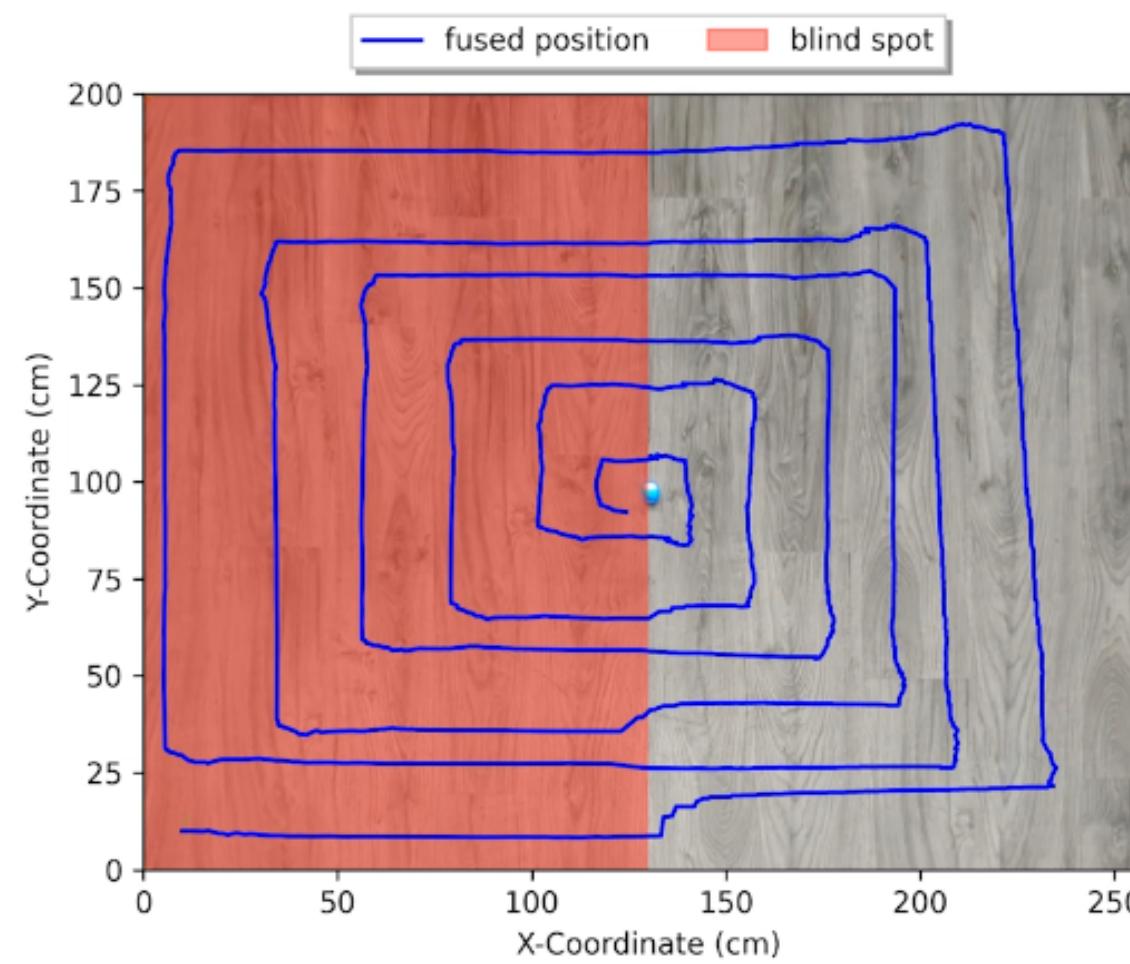


```
class ContourDetectionNode extends ProcessingNode<VideoFrame> {
    process(frame: VideoFrame): Promise<VideoFrame> {
        return new Promise((resolve) => {
            let contours = frame.image.findContours(
                OpenCV.RETR_EXTERNAL, OpenCV.CHAIN_APPROX_SIMPLE);
            if (contours.length >= 1) {
                // Sort contours by area and select largest area as 'ball'
                contours = contours.sort((a, b) => a.area - b.area);
                const m = contours[0].moments();
                const center = new OpenCV.Vec2(m.m10 / m.m00, m.m01 / m.m00);
                // Use the center as the 2D pixel position
                const position = new Absolute2DPosition(center.x, center.y);
                position.unit = LengthUnit.CENTIMETER;
                position.accuracy = Math.sqrt(contours[0].area);
                frame addObject(new DataObject("ball").setPosition(position));
            }
            resolve(frame);
        });
    }
}
```

OpenCV: Contour Detection



```
class ContourDetectionNode extends ProcessingNode<VideoFrame> {
    process(frame: VideoFrame): Promise<VideoFrame> {
        return new Promise((resolve) => {
            let contours = frame.image.findContours(
                OpenCV.RETR_EXTERNAL, OpenCV.CHAIN_APPROX_SIMPLE);
            if (contours.length >= 1) {
                // Sort contours by area and select largest area as 'ball'
                contours = contours.sort((a, b) => a.area - b.area);
                const m = contours[0].moments();
                const center = new OpenCV.Vec2(m.m10 / m.m00, m.m01 / m.m00);
                // Use the center as the 2D pixel position
                const position = new Absolute2DPosition(center.x, center.y);
                position.unit = LengthUnit.CENTIMETER; // Convert later
                position.accuracy = Math.sqrt(contours[0].area);
                frame addObject(new DataObject("ball").setPosition(position));
            }
            resolve(frame);
        });
    }
}
```



Contributing and Future Work

- ▶ Positioning algorithms
- ▶ Process network communication
- ▶ Bindings to other systems
- ▶ (UI) abstractions for end-user authoring
- ▶ Documentation and examples
- ▶ Calibration and set-up utilities

Resources and Links



-  <https://openhps.org>
-  <https://github.com/OpenHPS>
-  <https://npmjs.com/org/openhps>
-  <https://twitter.com/OpenHPS>



Maxim Van de Wynckel
<mvdewync@vub.be>