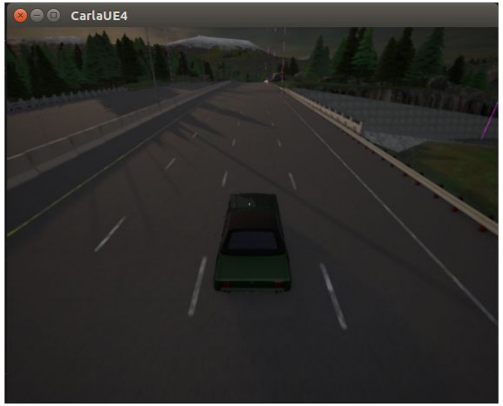
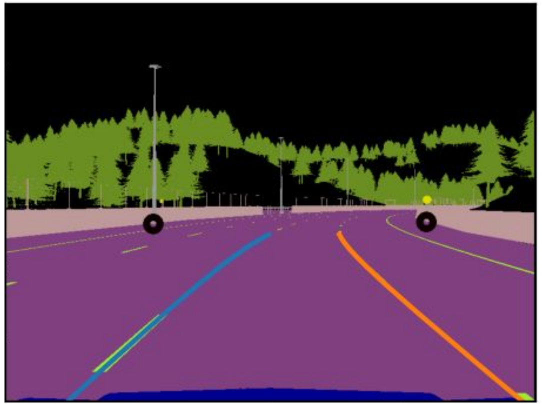
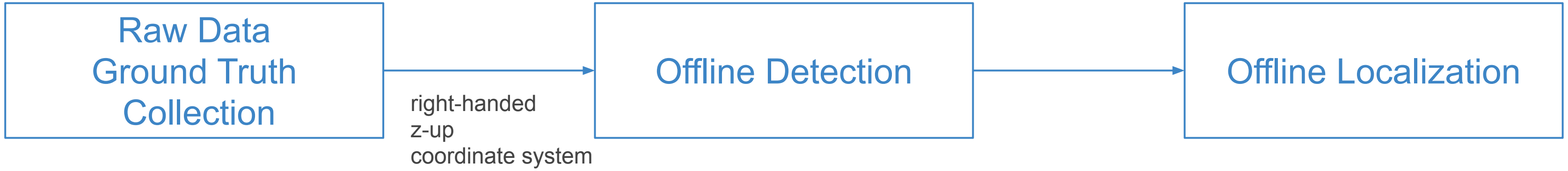
Scheme


CarlaUE4




Raw Data
Ground Truth
Collection
Offline Detection
Offline Localization
right-handed
z-up
coordinate system

Raw Data Collection

- Sensor Data:
  - GNSS
  - IMU (Odom)
  - Semantic/Depth Image
- Ground Truth:
  - Static:
    - Traffic Sign Actors*
  - Sequential:
    - Pose
    - Lane**

Offline Detection

- Pole
- Lane
- Stop Line (STOP Sign)

- By-product
  - Pole Map*

carlasim Package

data_collect.py

- class Geo2Location
- class World

- class CarlaSensor
- class IMU
- class GNSS
- class SemanticCamera
- class DepthCamera

groundtruth.py

- class GroundTruthExtractor
- class PoseGTExtractor
- class LaneGTExtractor (twofold)

record.py

- class Recorder

CarlaSensor

name:	String
_parent_world:	World
_parent:	Carla.Actor
sensor:	Carla.Actor
data:	dict
_queue:	Queue

update()

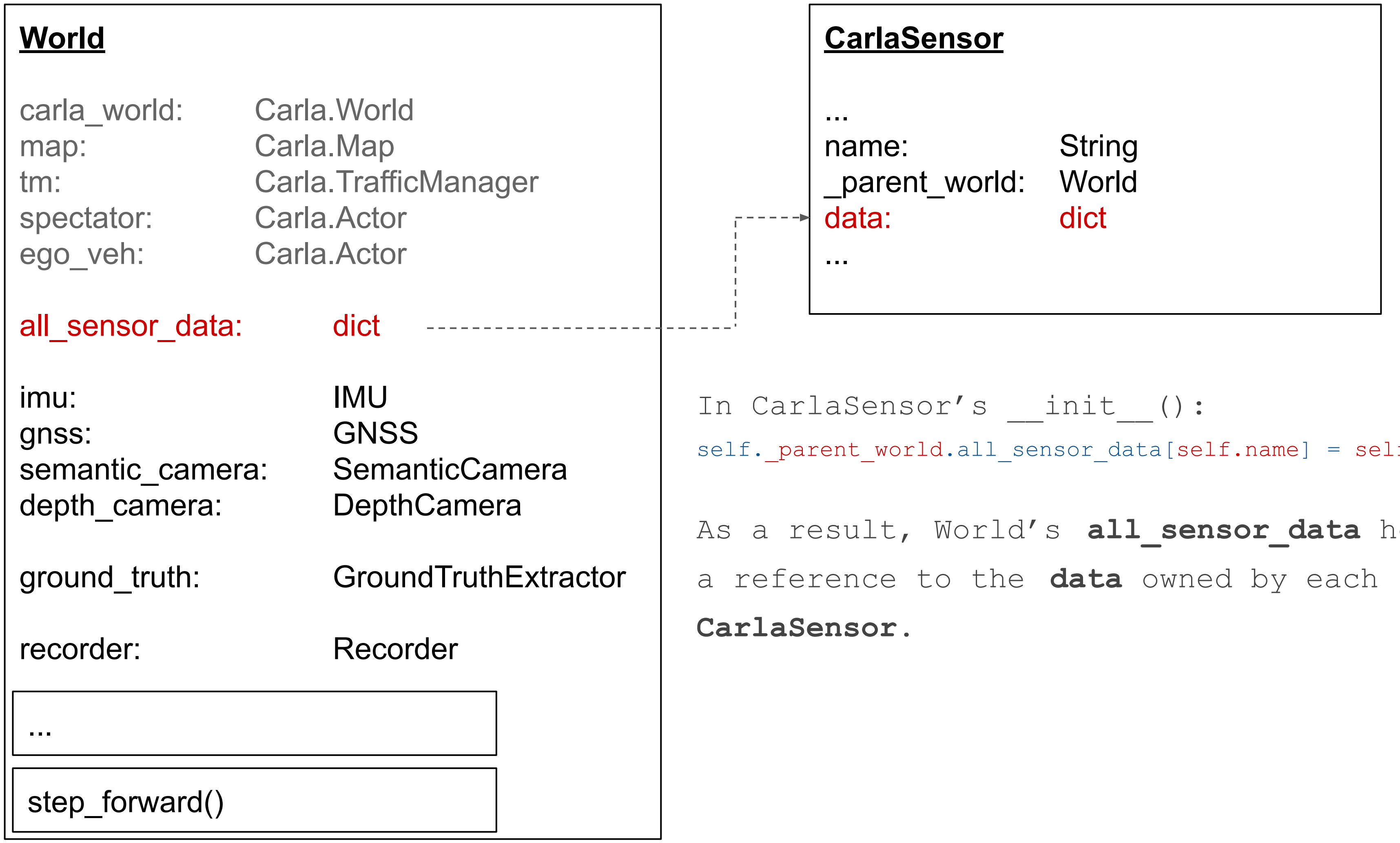
destroy()

Spawn sensor in __init__()

```
self.sensor = carla_world.spawn_actor(...)
self.sensor.listen(lambda event: self._queue.put(event))
```

Update data in update()

```
event = self._queue.get()
self.data['timestamp'] = event.timestamp
self.data['accel_x'] = event.accelerometer.x
self.data['accel_y'] = - event.accelerometer.y
self.data['accel_z'] = event.accelerometer.z
self.data['gyro_x'] = event.gyroscope.x
self.data['gyro_y'] = - event.gyroscope.y
self.data['gyro_z'] = - event.gyroscope.z
...
```



In CarlaSensor's __init__():

```
self._parent_world.all_sensor_data[self.name] = self.data
```

As a result, World's **all_sensor_data** holds a reference to the **data** owned by each **CarlaSensor.**

World

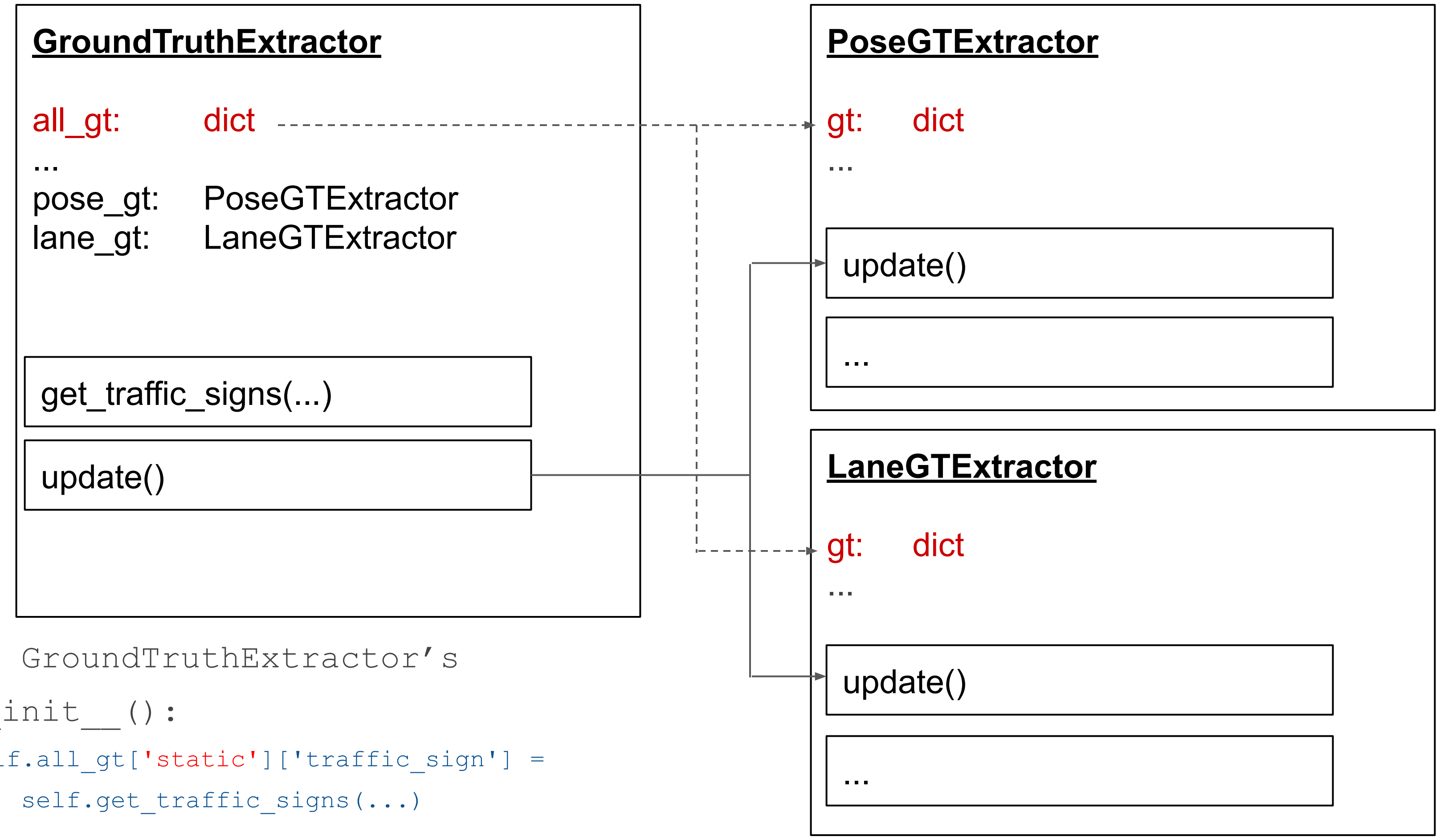
carla_world:	Carla.World
map:	Carla.Map
tm:	Carla.TrafficManager
spectator:	Carla.Actor
ego_veh:	Carla.Actor
all_sensor_data:	dict
imu:	IMU
gnss:	GNSS
semantic_camera:	SemanticCamera
depth_camera:	DepthCamera
ground_truth:	GroundTruthExtractor
recorder:	Recorder

...

step_forward()

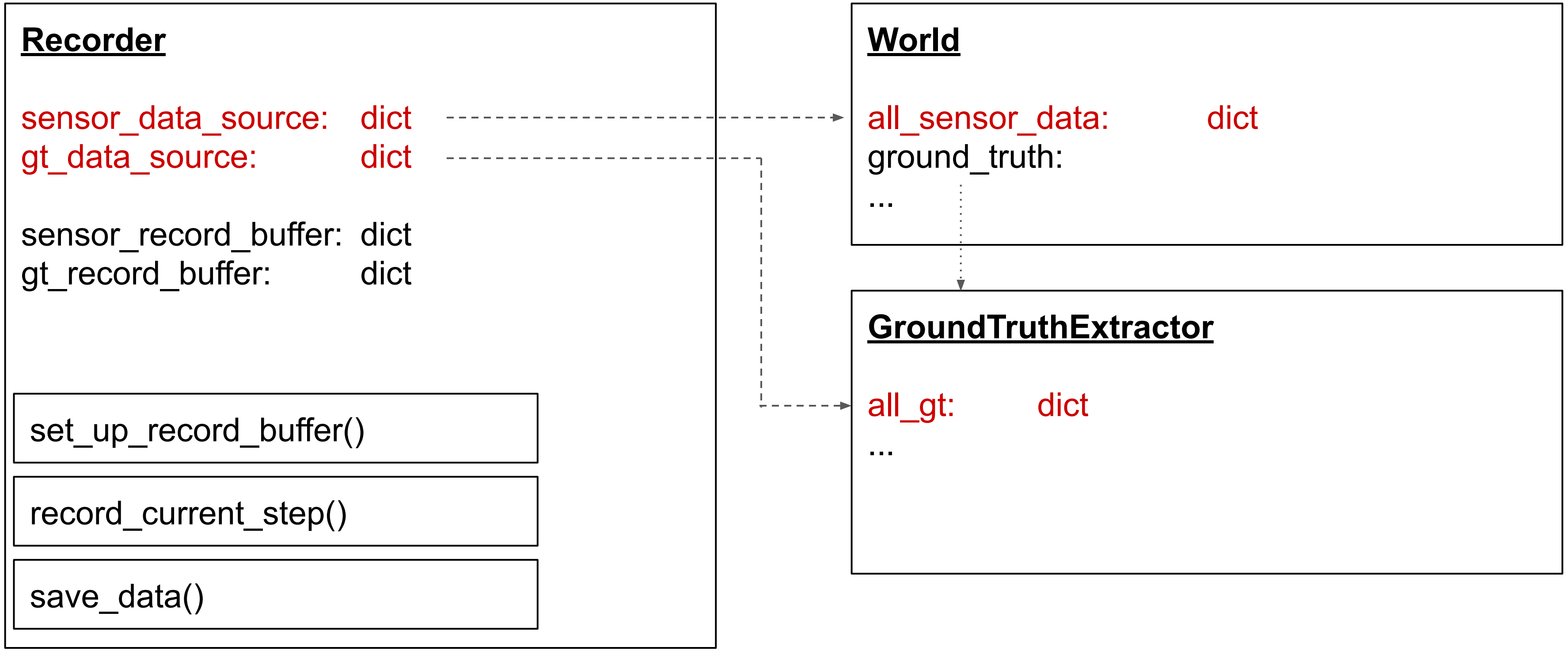
```
step_forward():
self.carla_world.tick()
self.imu.update()
self.gnss.update()
self.semantic_camera.update()
self.depth_camera.update()

self.ground_truth.update()

if self.activate_recorder:
    self.recorder.record_current_step()
```

all_sensor_data gets updated automatically since it holds just pointers to **data** that are updated by each individual **CarlaSensor**.



In GroundTruthExtractor's `__init__()`:

```
self.all_gt['static']['traffic_sign'] =
    self.get_traffic_signs(...)
...
self.all_gt['seq']['pose'] = self.pose_gt.gt
```


Recorder
sensor_data_source: dict
gt_data_source: dict
sensor_record_buffer: dict
gt_record_buffer: dict
set_up_record_buffer()
record_current_step()
save_data()
World
all_sensor_data: dict
ground_truth:
...
GroundTruthExtractor
all_gt: dict
...

Recorder

sensor_data_source: dict
gt_data_source: dict

sensor_record_buffer: dict
gt_record_buffer: dict

set_up_record_buffer()

record_current_step()

save_data()

carlasim.yaml:

```yaml
...
imu:
    timestamp: On
    # Velocities
    vx: On
    vy: On
    # Angular velocities
    gyro_x: Off
    gyro_y: Off
    gyro_z: On
    # Accelerations
    accel_x: Off
    accel_y: Off
    accel_z: Off
...
```

In set_up_record_buffer():

```python
sensor_record_buffer['imu']['vx'] = []
sensor_record_buffer['imu']['vy'] = []
sensor_record_buffer['imu']['gyro_z'] = []
```

Recorder

sensor_data_source:	dict
gt_data_source:	dict
sensor_record_buffer:	dict
gt_record_buffer:	dict

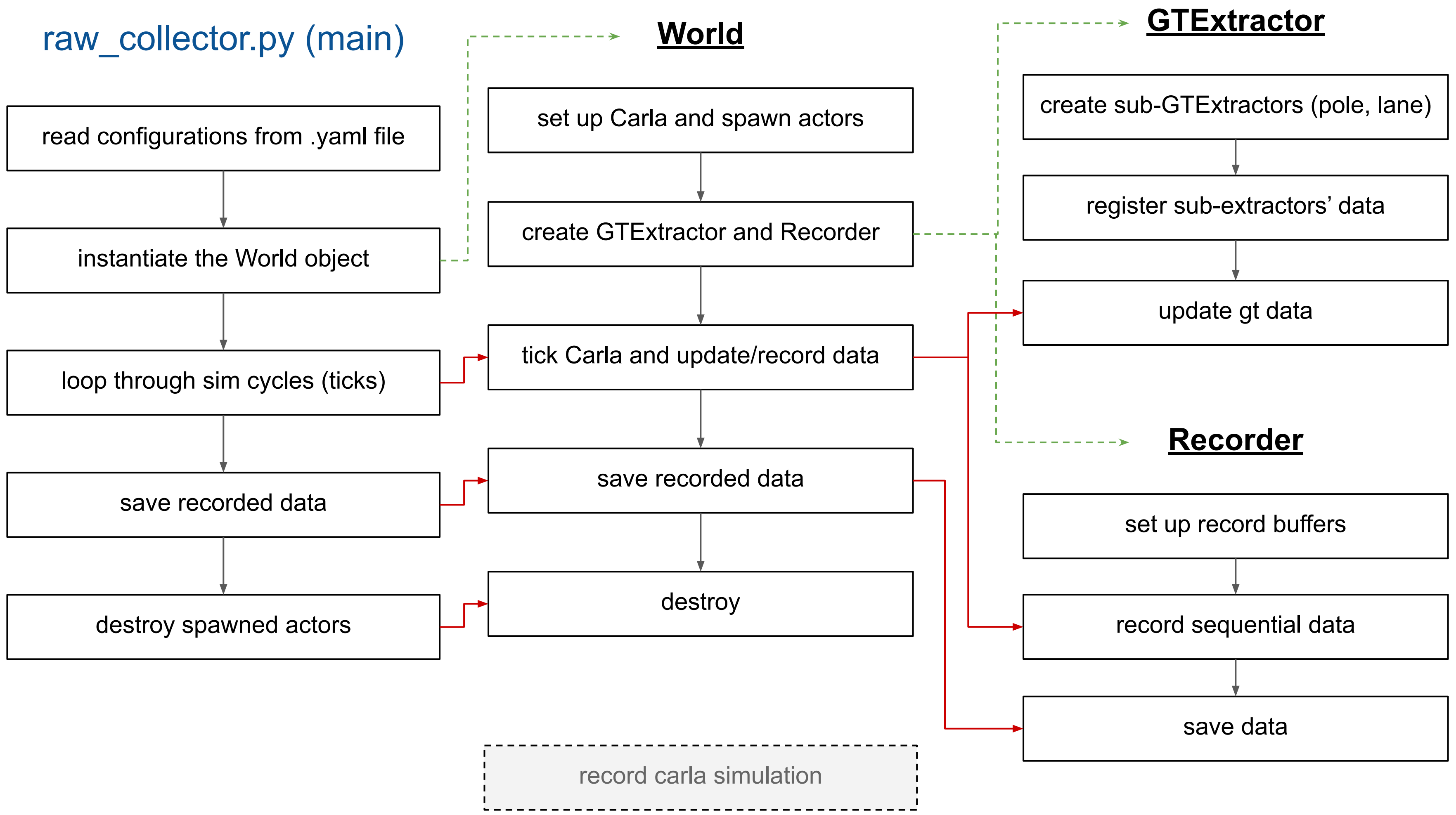
set_up_record_buffer()

record_current_step()

save_data()

At each time step (tick):

```
sensor_record_buffer['imu']['vx'].append(
      sensor_data_source['imu']['vx'])

sensor_record_buffer['imu']['vy'].append(
      sensor_data_source['imu']['vy'])

sensor_record_buffer['imu']['gyro_z'].append(
      sensor_data_source['imu']['gyro_z'])
```



raw_collector.py (main)
read configurations from .yaml file
instantiate the World object
loop through sim cycles (ticks)
save recorded data
destroy spawned actors
World
set up Carla and spawn actors
create GTExtractor and Recorder
tick Carla and update/record data
save recorded data
destroy
record carla simulation
GTExtractor
create sub-GTExtractors (pole, lane)
register sub-extractors' data
update gt data
Recorder
set up record buffers
record sequential data
save data