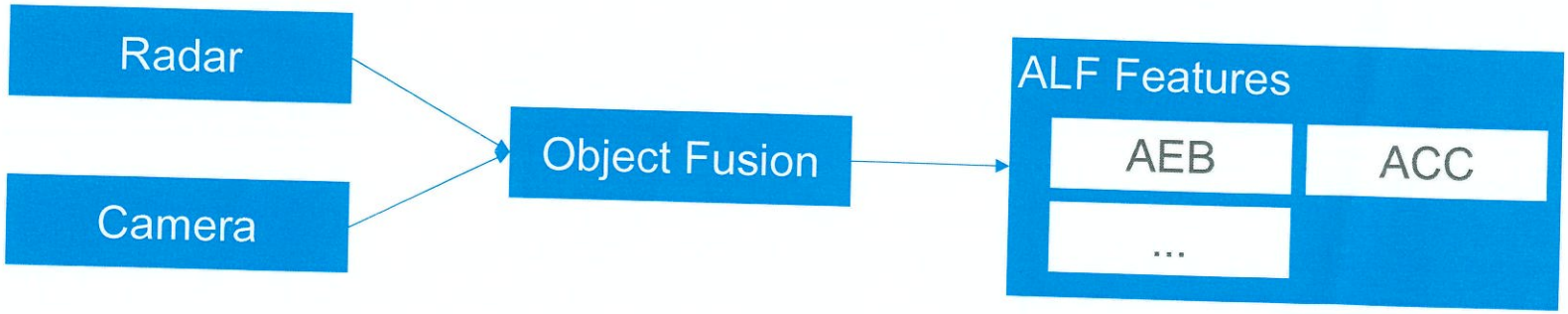
Topics

- Scope / Goals
- Basic Principle
- Functional Review:
  - Input Adapter
  - Ego Motion
  - Time Synchronization / Alignment
  - Association
  - Filtering
  - Track Management
  - Output Converter
  - RaCam Converter

Scope of Object Fusion

- Function supporting Advanced Driver Assistance Systems (ADAS)
- In Geely context, providing ALF Features with information of the environment / objects surrounding the ego vehicle
- Input from camera and radar is on object level
- Object Fusion itself is independent of sensor type



Difference to Information Fusion

- Object Fusion only takes object reported by sensors into account and does not require lane information
- Inside ALF, there are information Fusion AAUs combining e.g. lane information

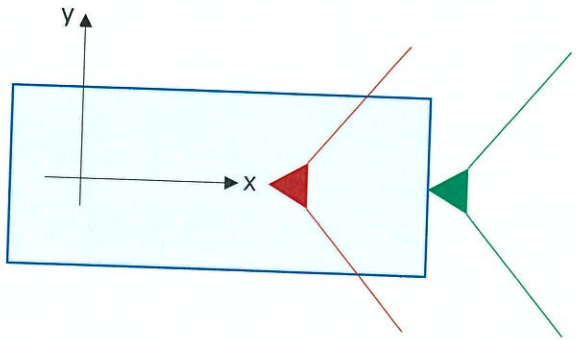
Goals of Object Fusion

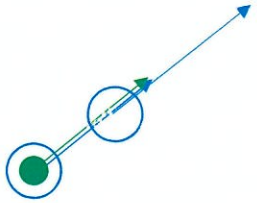
- Combining object information from sensors in an intelligent way taking individual sensor capabilities and limitations into account
  - E.g. measurement uncertainty, classification performance etc.
- Fusion improves
  - field of view
  - availability
  - accuracy
  - stability
  - type of information (e.g. classification from camera, motion type from radar)+
- Providing confidence using redundancy of two sensors (e.g. for AEB)

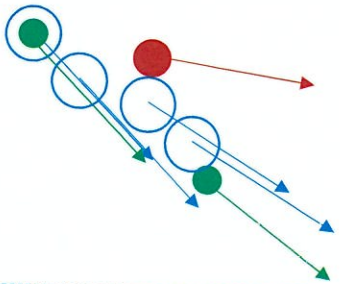
Fundamental Principal of Object Fusion

- Object Fusion always has an internal list with all objects, since objects are tracked over time this list is called *Track List*
- Object received from sensors are directly fused with this internal Track List and not with the objects of another sensor
  - camera and radar are never directly fused!
- Fusion is based on an ordinary Kalman Filter with constant acceleration model and Mahalanobis distance as association metric
- The internal *Track List* is in rear-axle coordinate system and the motion states are over-ground (absolute)
  - Movement of the host vehicle is corrected (no additional dynamic in Kalman Filter required)
- Fusion has no global timestamp, but sensors provide a latency from taking the measurement until Fusion has access to it (e.g. by CAN)

Example of a Fusion sequence







object of internal Track List

radar object

camera object

velocity vector

Radar measurement is received

Camera measurement is received

Fusion is predicted

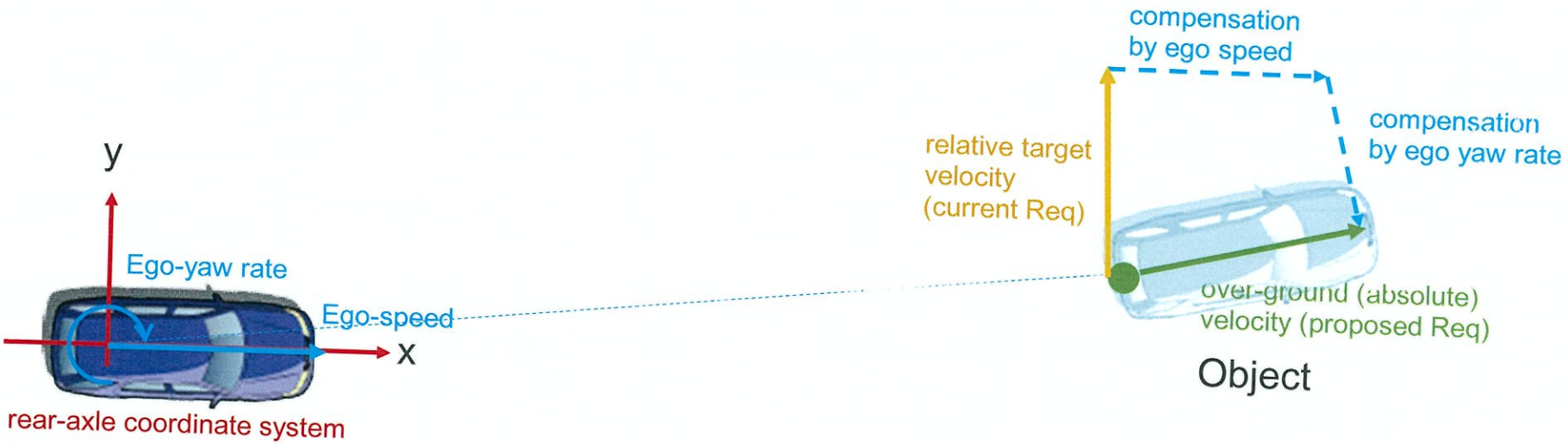
time

internal Track List, time reference

radar measurement

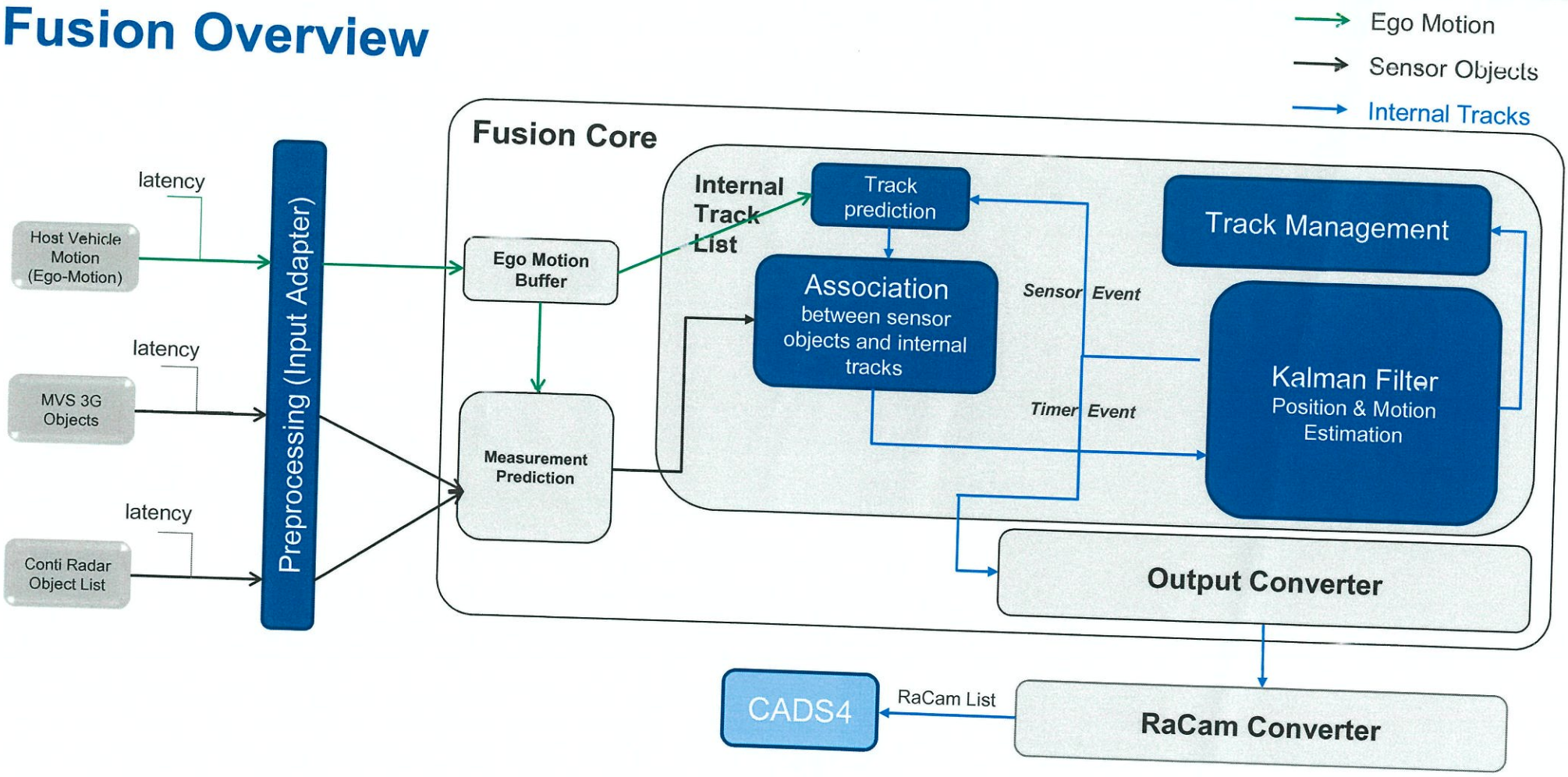
camera measurement

Coordinate System Object Fusion

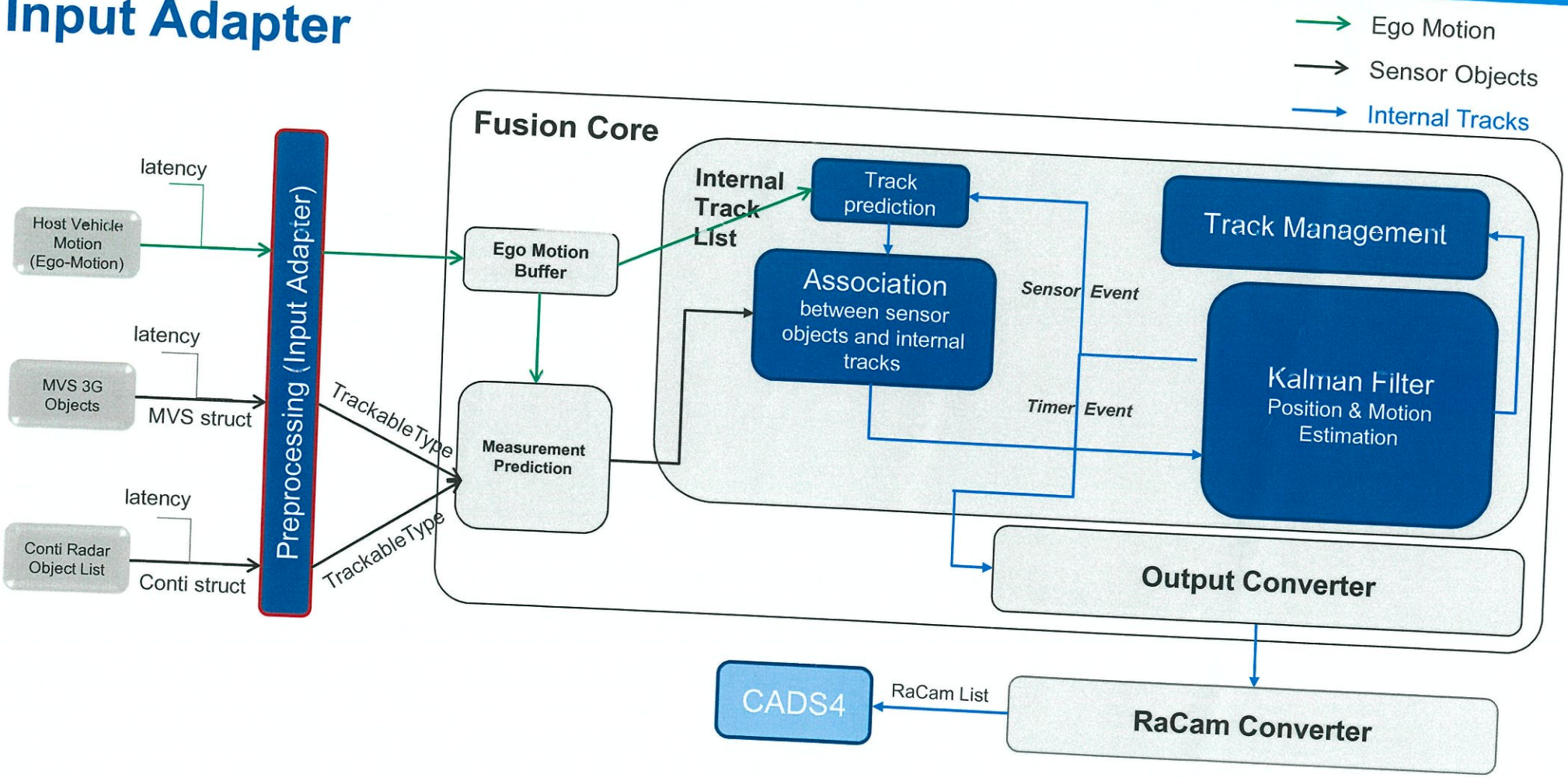


This example shows only the velocity, this is valid as well for acceleration. Instead of storing the relative velocity track are stored in over-ground (absolute) velocity, which results when the relative velocity is compensated by the ego-speed and ego yaw rate (projected on the object position). It is essential that the rear-axle coordinate system is used, since this is the rotation center of the ego vehicle in normal driving situations.

Fusion Overview



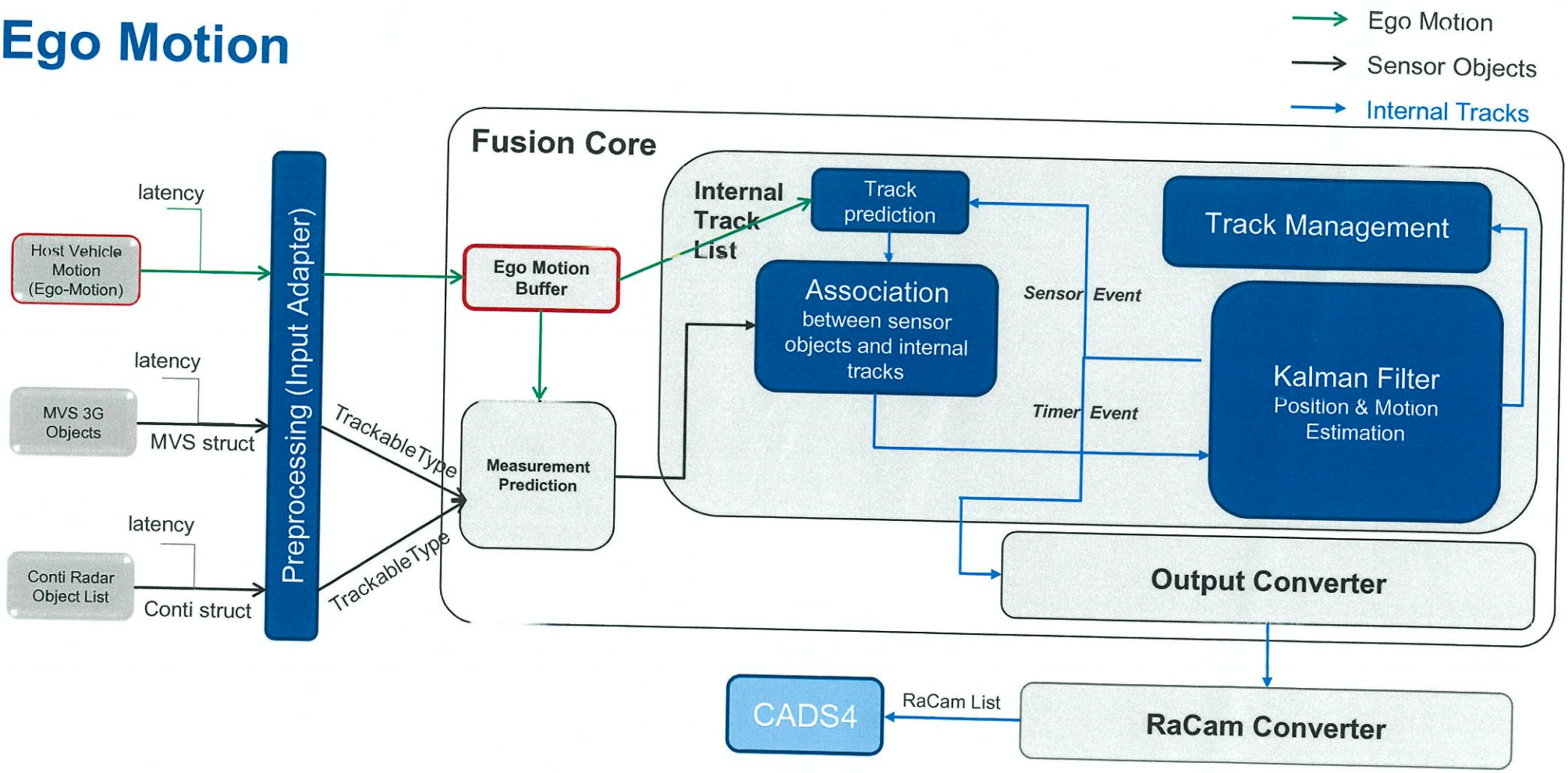
Input Adapter



Input Adapter

- Sensors provide different data structures
  - Fusion has a generic interface (TrackableList)
  - Mapping of sensor structure fields on Object Fusion input structure e.g.:
    - Mapping of class type
    - Mapping of motion type
    - ...
- Sensors provide data in different coordinate systems e.g. front bumper, camera coordinate system, rear axle etc.
  - Coordinate transformation necessary to bring sensor data into Object Fusion input format and frame of reference

Ego Motion


Ego Motion
Sensor Objects
Internal Tracks
Fusion Core
latency
Host Vehicle Motion (Ego-Motion)
latency
MVS 3G Objects
MVS struct
latency
Conti Radar Object List
Conti struct
Preprocessing (Input Adapter)
TrackableType
TrackableType
Ego Motion Buffer
Measurement Prediction
Internal Track List
Track prediction
Association between sensor objects and internal tracks
Sensor Event
Timer Event
Track Management
Kalman Filter Position & Motion Estimation
Output Converter
CADS4
RaCam List
RaCam Converter

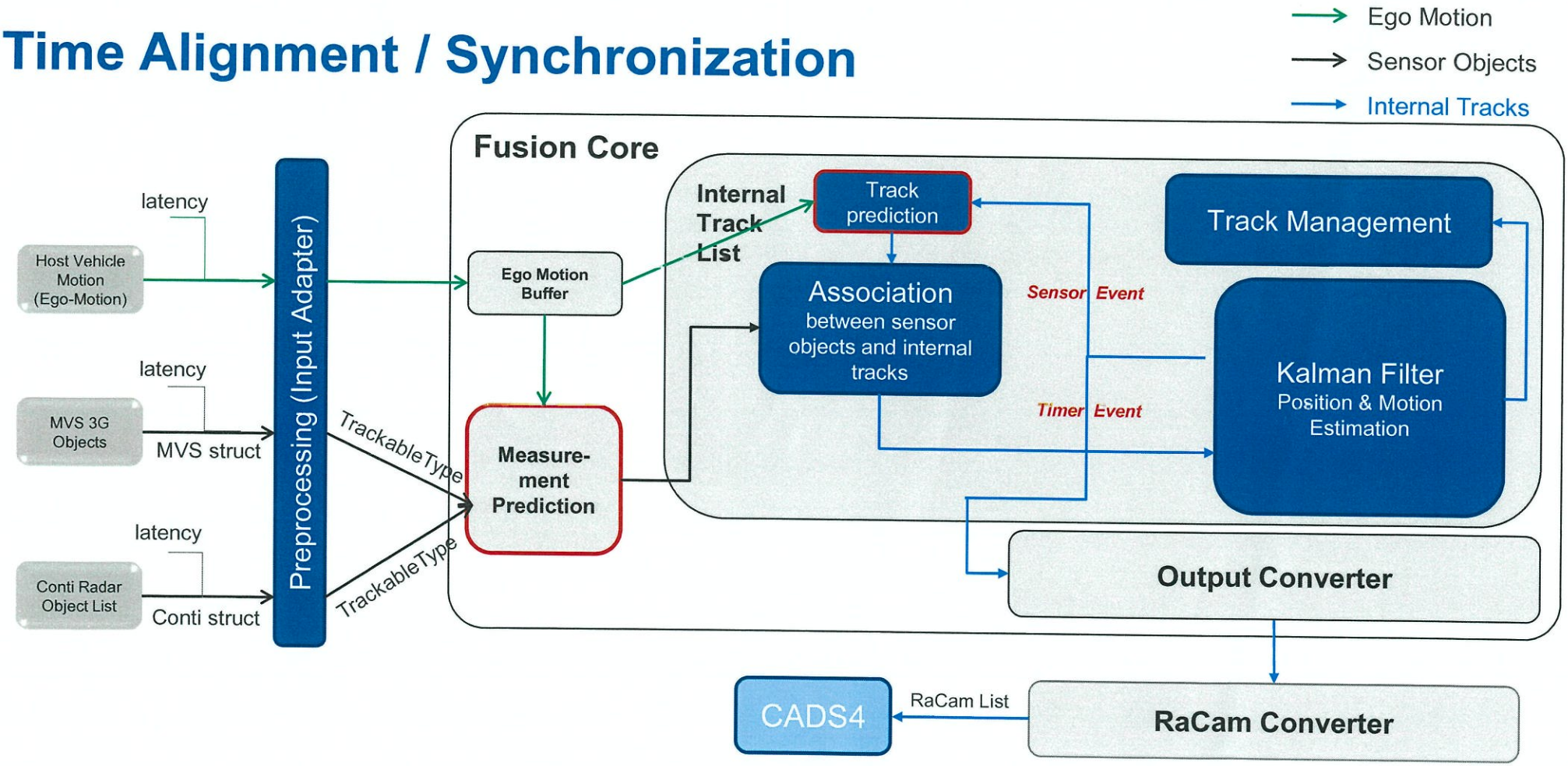
Ego - Motion

- Ego Motion (Host-vehicle motion state) is required for:
  - Compensation of ego motion (only target dynamics are filtered in Kalman filter)
  - Compensation of ego movement / change of heading between measurement and Fusion processing (History required)
  - Calculation of motion type (stationary, oncoming, ego direction / stopped / reversing)
- Signals required (reference point is center of rear axle)
  - Speed
  - Acceleration
  - Yaw Rate
- Signal requirements
  - Signals shall be offset compensated (specially yaw rate)
  - Signals shall not contain a scaling error (specially speed / acceleration due to e.g. wheel diameter mismatch)
  - Signals shall be continuous (no jumps, specially for stopping vehicles)

Ego - Motion History

- For transformation of objects from sensors, but as well internal Track List an Ego Motion history is required for:
  - Position correction: The path (Δx, Δy) the host vehicle moves over ground in a time **dt**, all objects move relative to the host vehicle in the opposite way
  - Motion compensation: If the measurement was taken e.g. 100ms before, the Ego Motion at this point in time (-**dLatency**) has to be used to convert relative motion of sensor objects to over-ground movement
  - Output Prediction: Fusion internal state is in the past, for the motion state at the current point in time, objects has to be predicted using the host movement from ($\mathbf{-dt_{Fusion}}$) to current point in time [see concept for time alignment]

Time Alignment / Synchronization



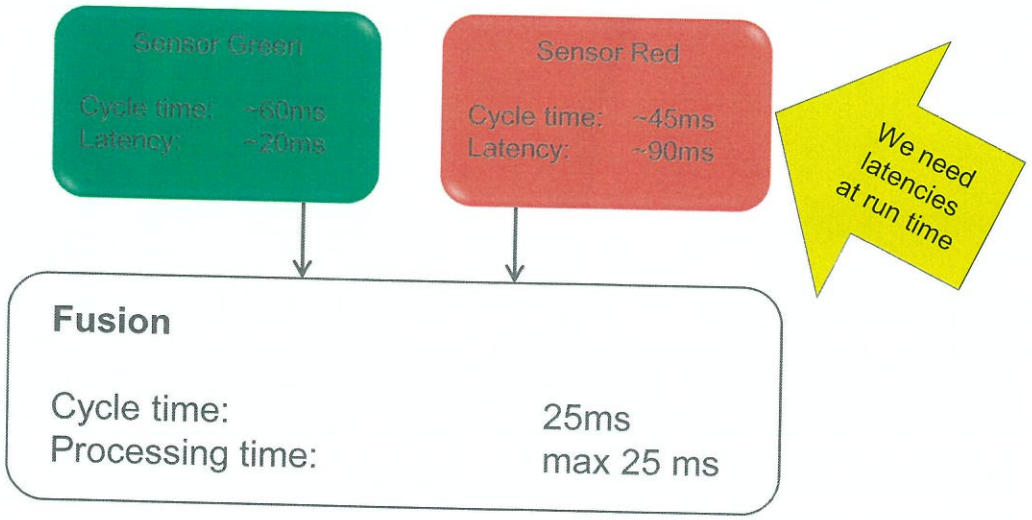
Time Alignment
Latency Concept

- Fusion is executed time-triggered with 25ms, independent if a new measurement has arrived or not
- Sensor Latency is the time from physical sensor measurement until Fusion process this measurement
- Fusion Latency (Age) is the time the internal Track List refers to, to the current point in time
- Fusion has no global timestamp, but works with latencies (ages)

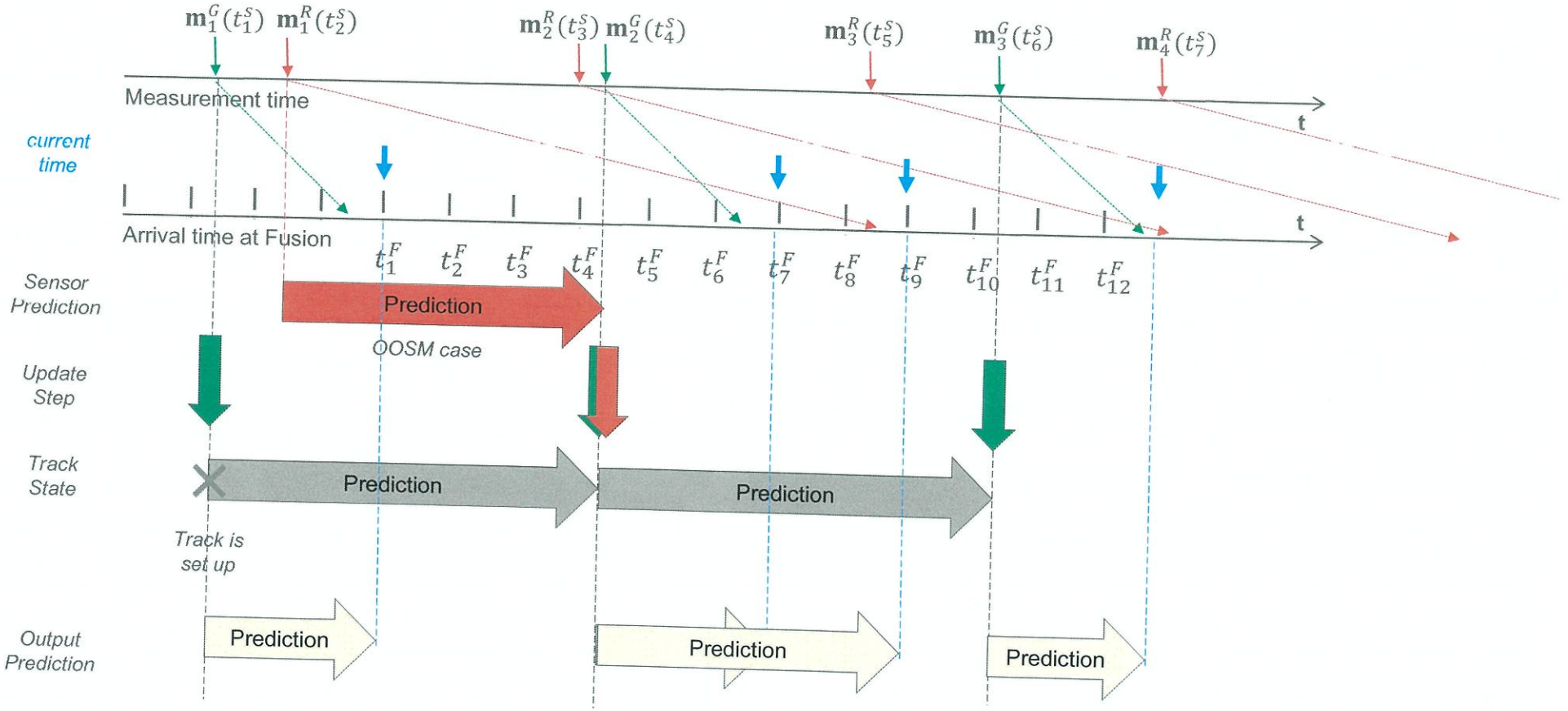
Time Alignment
Latency Concept

- Latency (Age) of internal Track List is always at the time of most recent measurement received from both sensors
- If a new measurement arrives:
  - which was measured after the current time of the internal track list
    → predict internal track list to time of measurement
  - which was measured before the current time of the internal track list
    → predict sensor object list to time of Fusion
    this is called OOSM (Out-Of-Sequence Method)

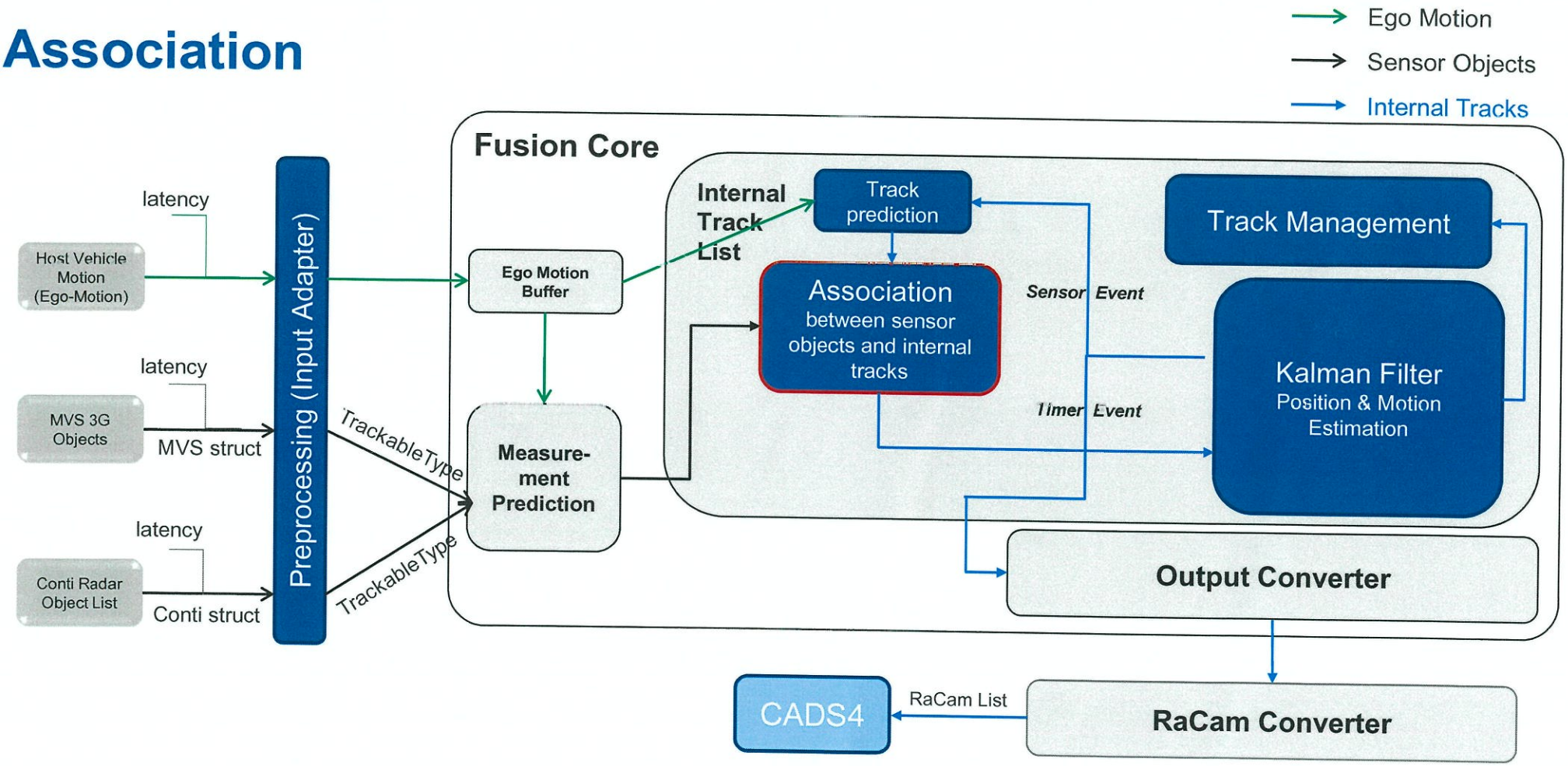
Example



Fusion Timing - Example



Association



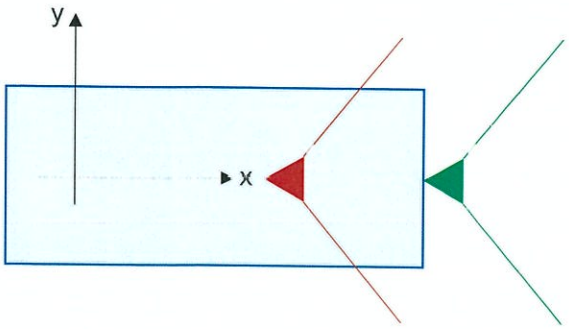
Association Concept

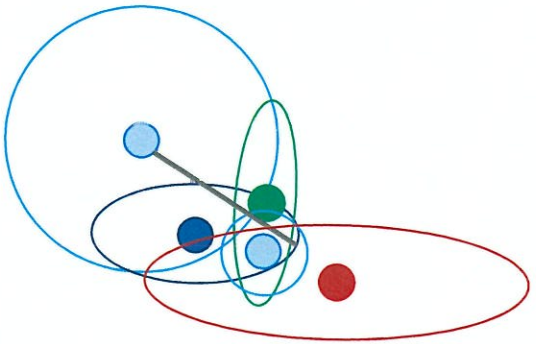
- 1-to-1 object association, based on two inputs:
  sensor object list and internal Fusion tracks
- Calculation of distance matrix between all sensor inputs and Fusion tracks for association based on Mahalanobis distance
- Run association algorithm (local min) based on distance calculation with an appropriate gate

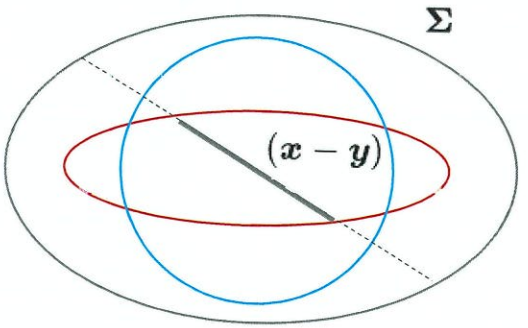
Results:

- Assignment of Fusion tracks with assigned sensor input
  → Fusion tracks are updated with sensor objects in Kalman Filter
- Unassigned senor object
  → Fusion set up new tracks based on this sensor objects

Mahalanobis - Distance
Basic Implementation

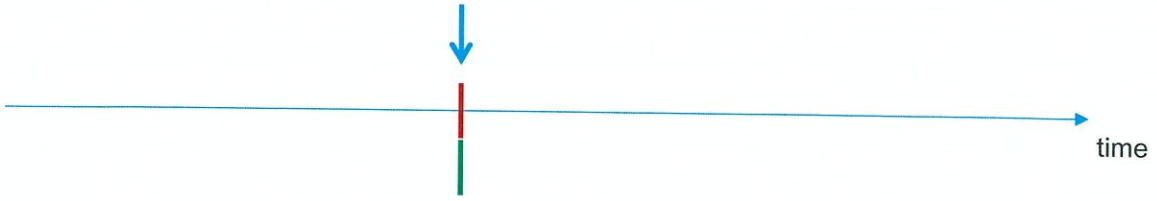






Association based on Mahalanobis distance

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{y})}$$

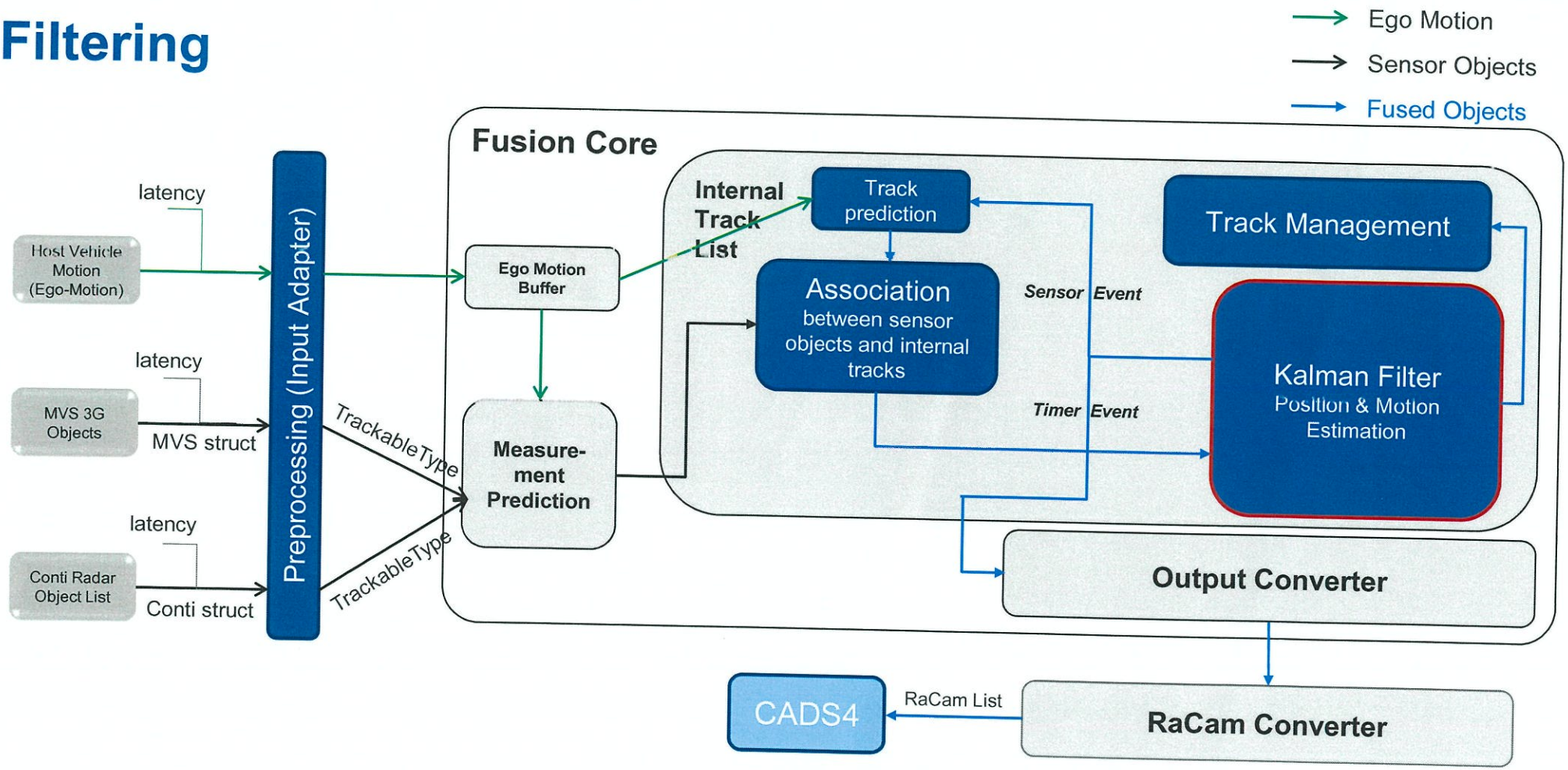


Association
Parametrization / Evaluation

Parameterization concept:

- Critical factor is match gate (Mahalanobis distance, when track and sensor object are associated, otherwise a new track is set up)
- Define scenarios and label scenarios for 1-2 objects
  - small match gate is required (e.g. two vehicles drive parallel, a vehicle drives slowly close to stationary objects, pedestrian is close to stationary vehicle, NCAP occluded child etc.)
  - large match gate is preferred (mainly in cases where sensor's have higher measurement uncertainty e.g. vehicle far away, in a curve, with high dynamic, NCAP other scenarios)
- Automatic sweep of match gate, find a good compromise between small match gate scenarios and large match gate scenarios

Filtering


Ego Motion
Sensor Objects
Fused Objects
Fusion Core
latency
Host Vehicle Motion (Ego-Motion)
latency
MVS 3G Objects
MVS struct
latency
Conti Radar Object List
Conti struct
Preprocessing (Input Adapter)
TrackableType
TrackableType
Ego Motion Buffer
Measure-ment Prediction
Internal Track List
Track prediction
Association between sensor objects and internal tracks
Sensor Event
Timer Event
Track Management
Kalman Filter Position & Motion Estimation
Output Converter
CADS4
RaCam List
RaCam Converter

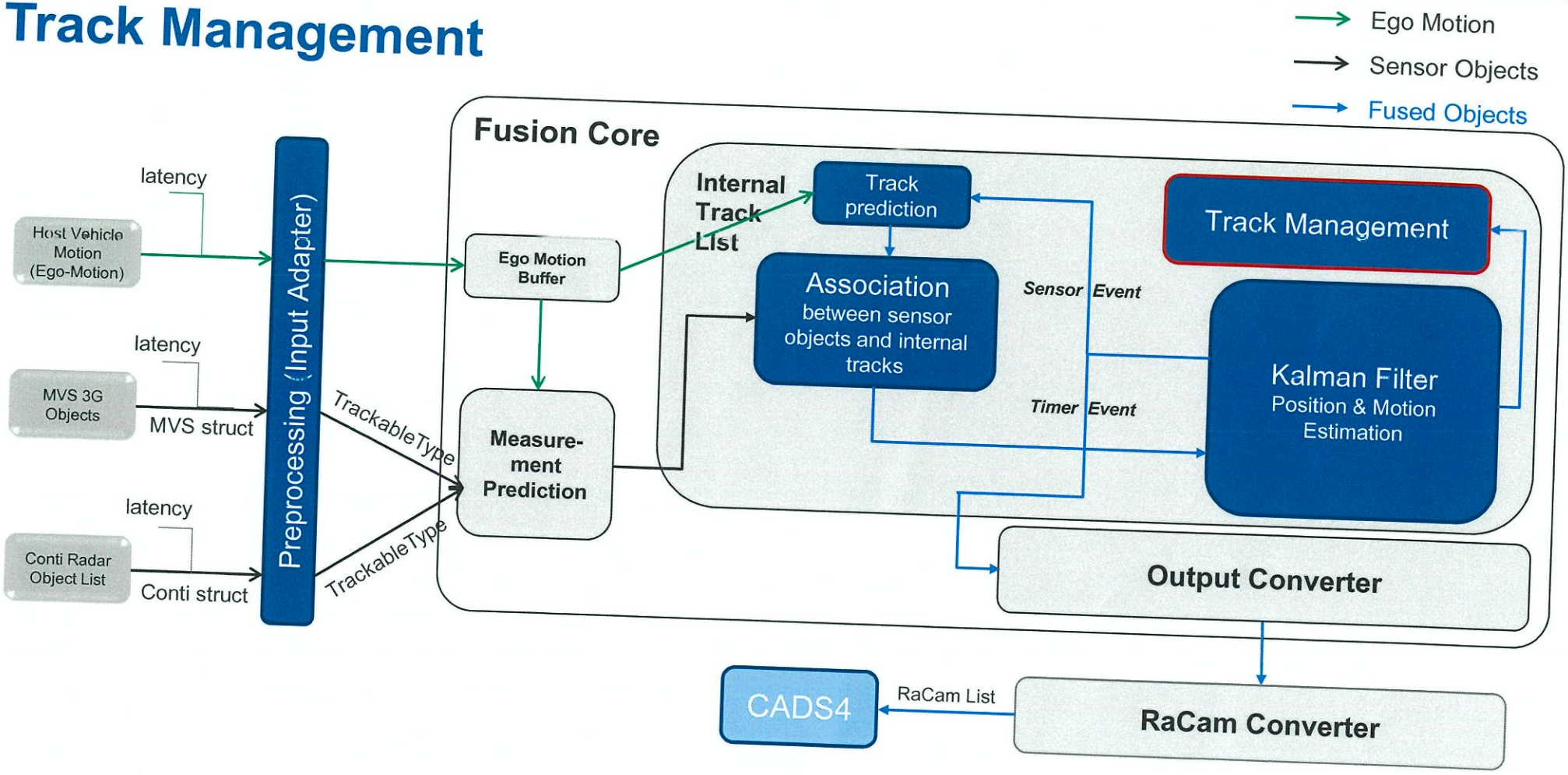
Filtering
Concept & Implementation

- Utilizing the Kalman filter for object fusion in order to describe consistent and smooth trajectories
- The Kalman filter enables modeling object motion with different kinematic models
- The Kalman filter supports modeling input noise as Gaussian
- Implementation:
  - Linear Kalman Filter based on constant acceleration model
  - States are in Cartesian coordinates with a x- and y- component for relative position and over ground velocity and acceleration
  - Objects are treated as point objects
- In addition a second Kalman Filter calculates Heading and Yaw Rate based on the coordinated turn model (constant yaw rate)

Filtering
Parametrization / Evaluation

- Process noise for the kinematic models places the most important parameter to optimize, since we do not get (co)variances from all sensors, the measurement noise might also become subject of an optimization
- Similar approach as for the association:
  - Define a test catalogue
  - Use simulated data to get a baseline for process (and measurement) noise parameters
  - Use real data to derive the actual process (and measurement) noise parameters
- Smooth output vs. filter reaction time:
  - To ensure smooth object trajectories for comfort features and still enable a fast reaction time for safety features the optimization data set must contain a good balance of
    - Standard scenarios that occur all the time (e.g. vehicles driving straight in front of ego vehicle)
    - Highly dynamic scenarios that occur very rarely (e.g. vehicle in front braking with full deceleration)

Track Management



Track Management
ID Provider

Concept description:

- ID provider manages ID pool
- Track management requests an ID on track initialization
- Track management 'returns' ID on track deletion/drop
- ID provider ensures uniqueness of IDs

Current implementation:

- Large ID space (currently 60001) to avoid fast reuse of IDs

Track Management
Lifecycle Management

- Tracks have no initialization phase since we get established tracks from sensors
- Keep/drop:
  - Two fields control keeping/dropping of tracks:
    - Lifespan: Free slot, dying track, updated/new track, coasted track
    - Sensor pattern: logical concatenation of sensor bit codes.
      - Current Sensor Pattern: Zero if no sensor sees the object anymore
      - Sensor Pattern History: Once Sensor pattern is set, it will never be deleted
  - Lifespan will be decreased at the beginning of each cycle and increased with each sensor update
- Currently Coasting for 10 cycles (CADS4 requirement)
  - only fused objects are coasted (but parameter available to coast radar / camera only)

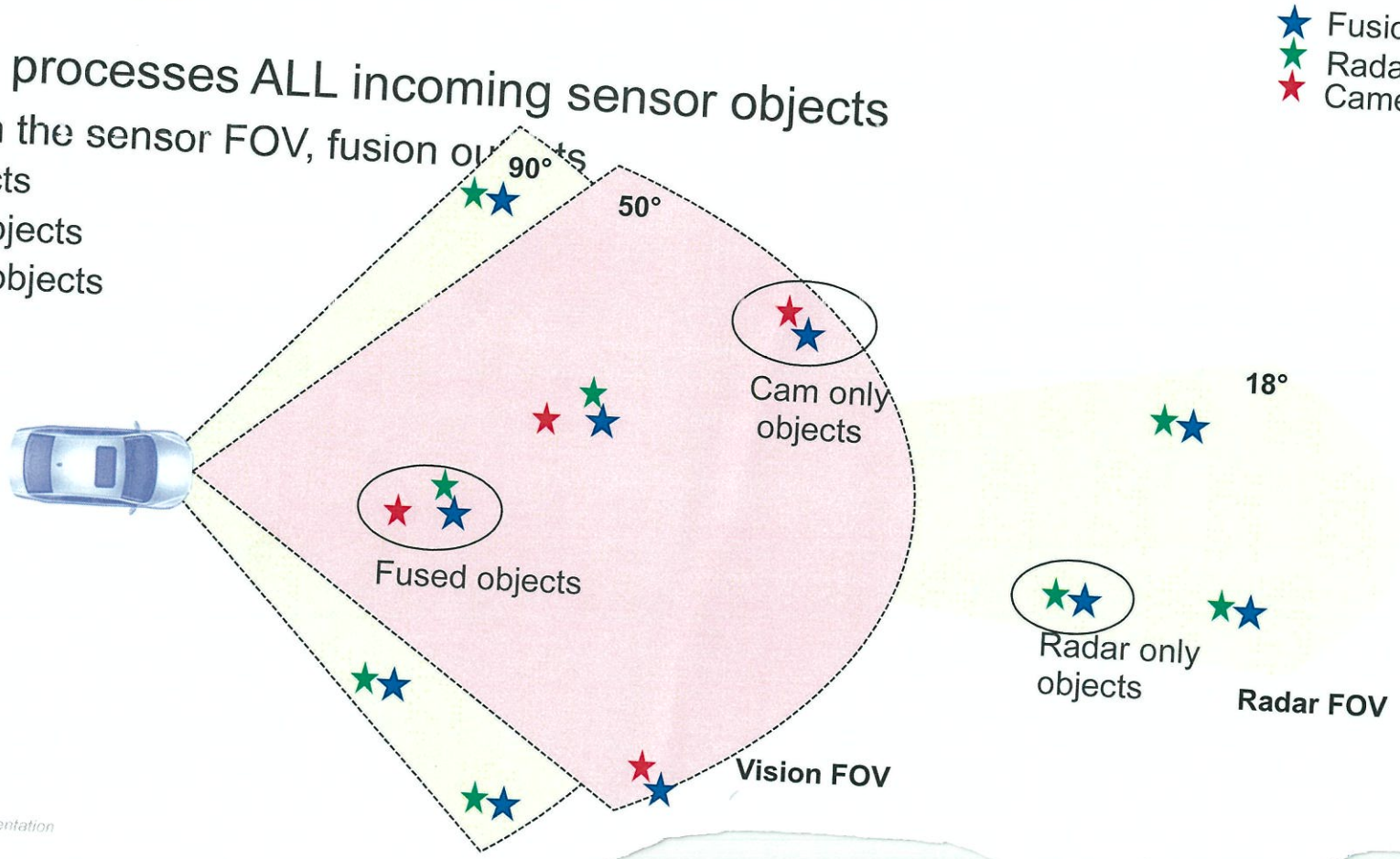
Track Management
Object Selection

Object Selection, prioritize objects based on CADS4 requirements (e.g. ROI, sensor visibility) for:

- Output: maximal number of objects for CADS4 input is 32 - Fusion internally track list has a size for 122 ((40 radar + 21 camera)* 2) → Selection for Output
- Internal: Internal maximal number of tracks is 122, so if there are coasted objects size could be too small (rarely happens) → Selection would deleted coasted objects based on CADS4 requirements

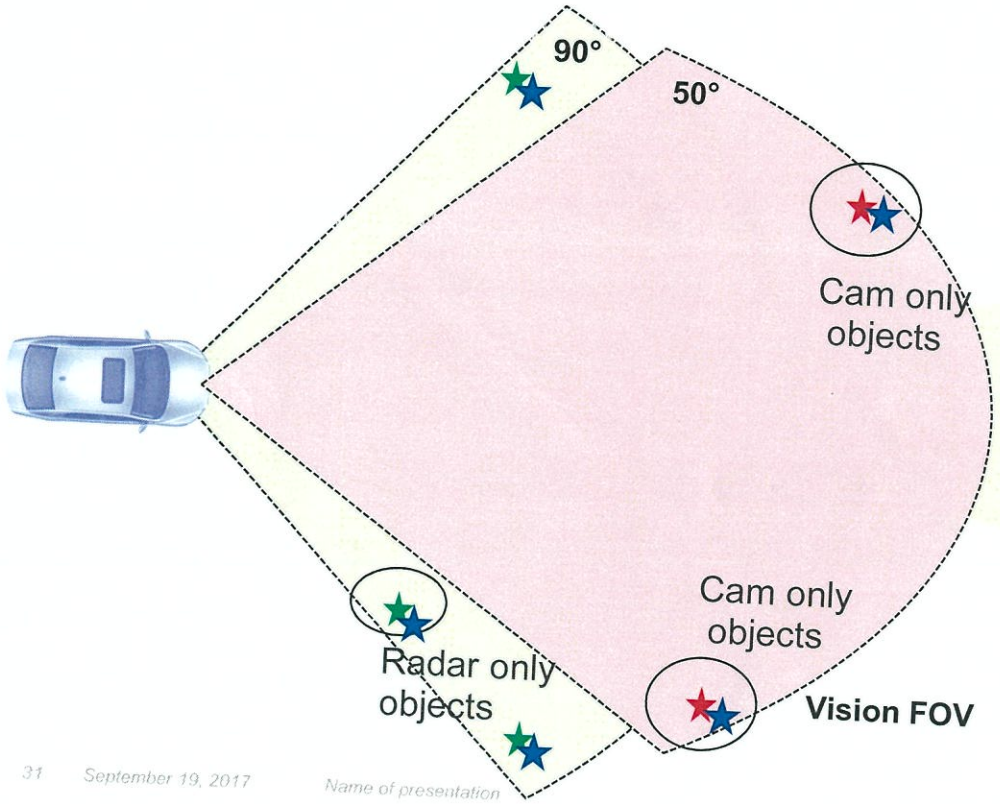
Track Management
Object Selection

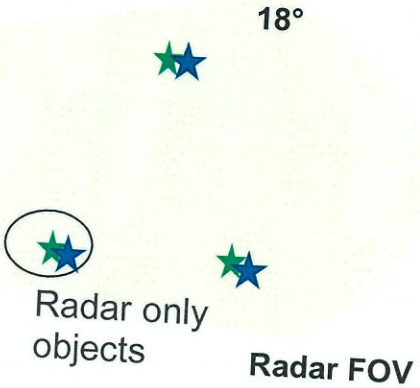
- Object Fusion processes ALL incoming sensor objects
  - Depending on the sensor FOV, fusion outputs
    - Fused objects
    - Cam only objects
    - Radar only objects



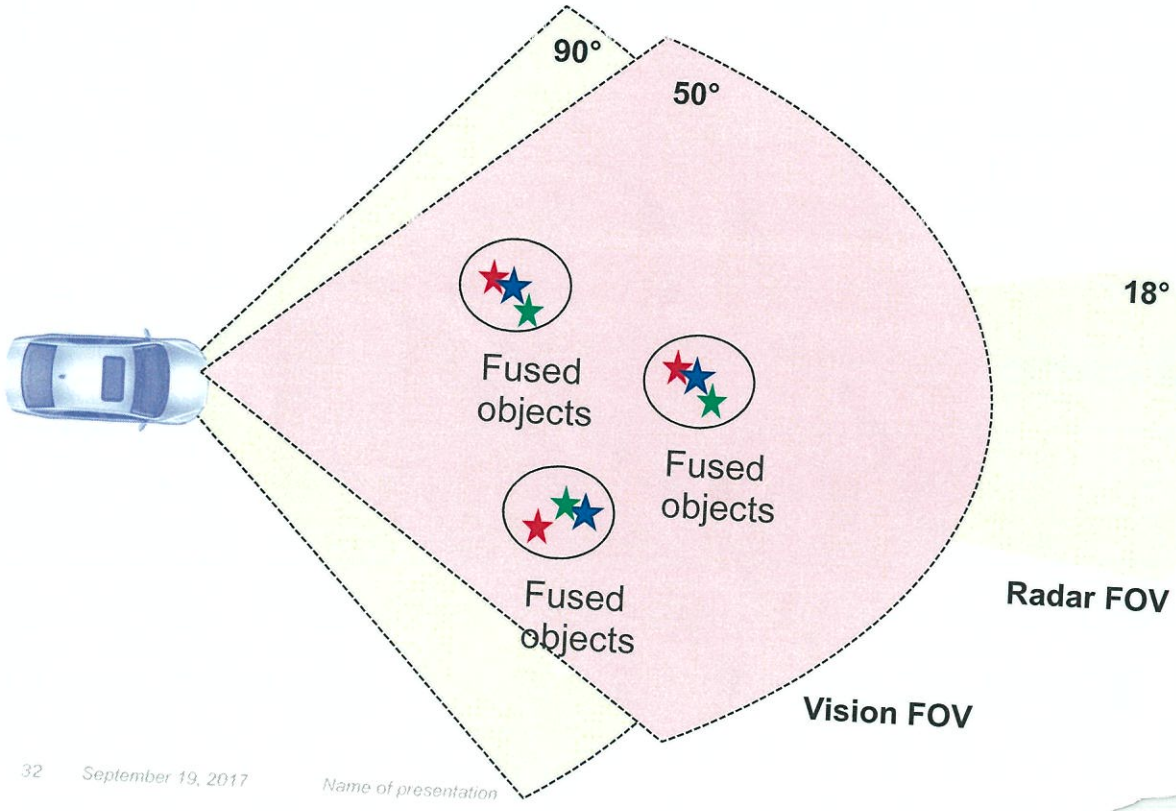
Object Selection
Sensor-only Objects

- Sensors send objects within each sensor FOV to fusion.
- Object fusion outputs fusion objects ★ based on sensor input ★ or ★
- If sensor input ★ or ★ drops, fusion objects drop ★ as well



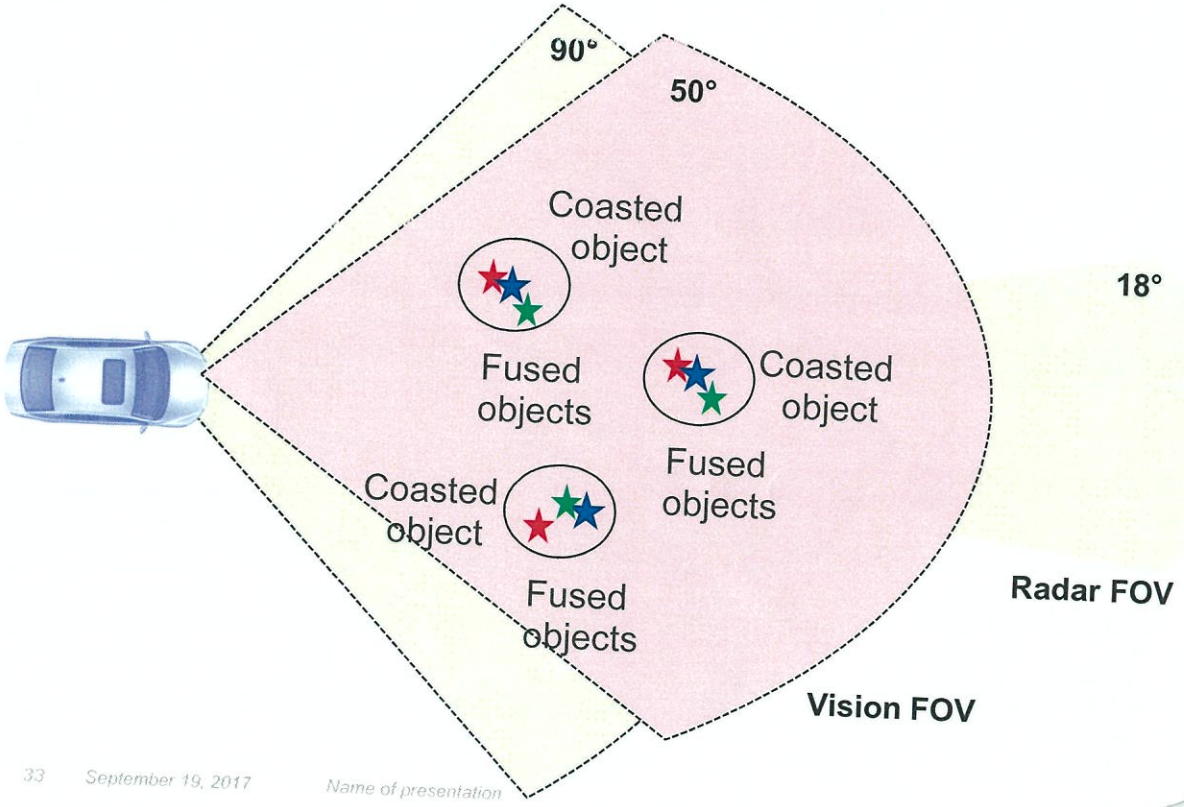


Object Selection
Objects with Overlapping FOV



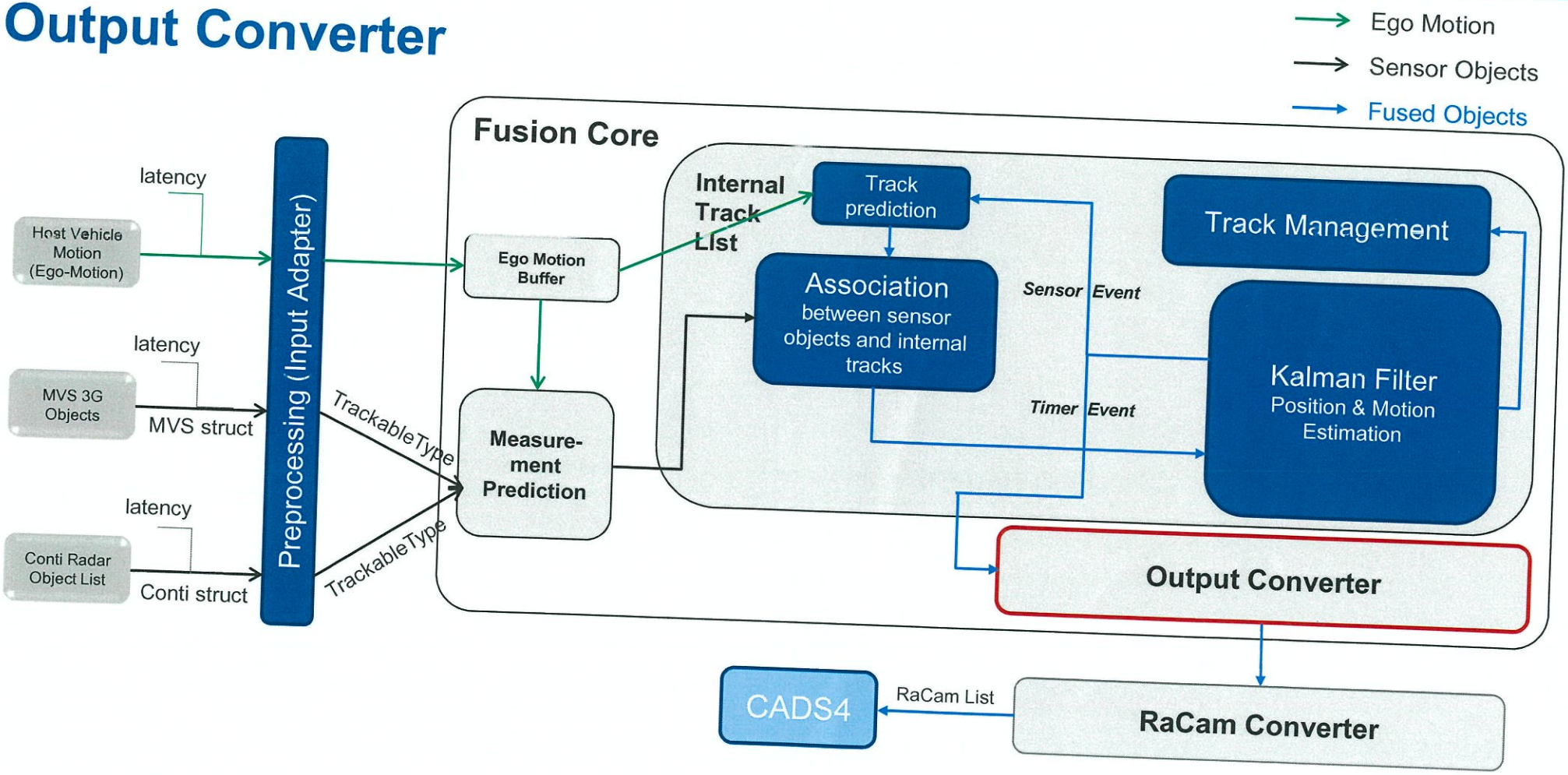
- Sensors send objects within each sensor FOV to fusion.
- Object fusion outputs fusion objects ★ based on sensor input ★ or ★
- If sensor input ★ or ★ is associated to existing object, ★ fusion objects are updated with new sensor object.

Object Selection
Objects with Overlapping FOV



- If one sensor object drops, fusion objects is updated by other sensor object
- Object fusion outputs fusion objects ★ based on the remaining sensor input
- If remaining sensor input drops as well, fused object is coasted/predicted (tuning parameter) for a tunable number of fusion cycles
- Object is still available for feature input

Output Converter



Output Converter
Output Prediction

Latency (Age) of internal Track List is always at the time of most recent measurement received from both sensors

CADS4 requires object list at current point in time, so max. 32 objects are selected from internal track list and predicted to current point in time

Output Converter
Motion Type

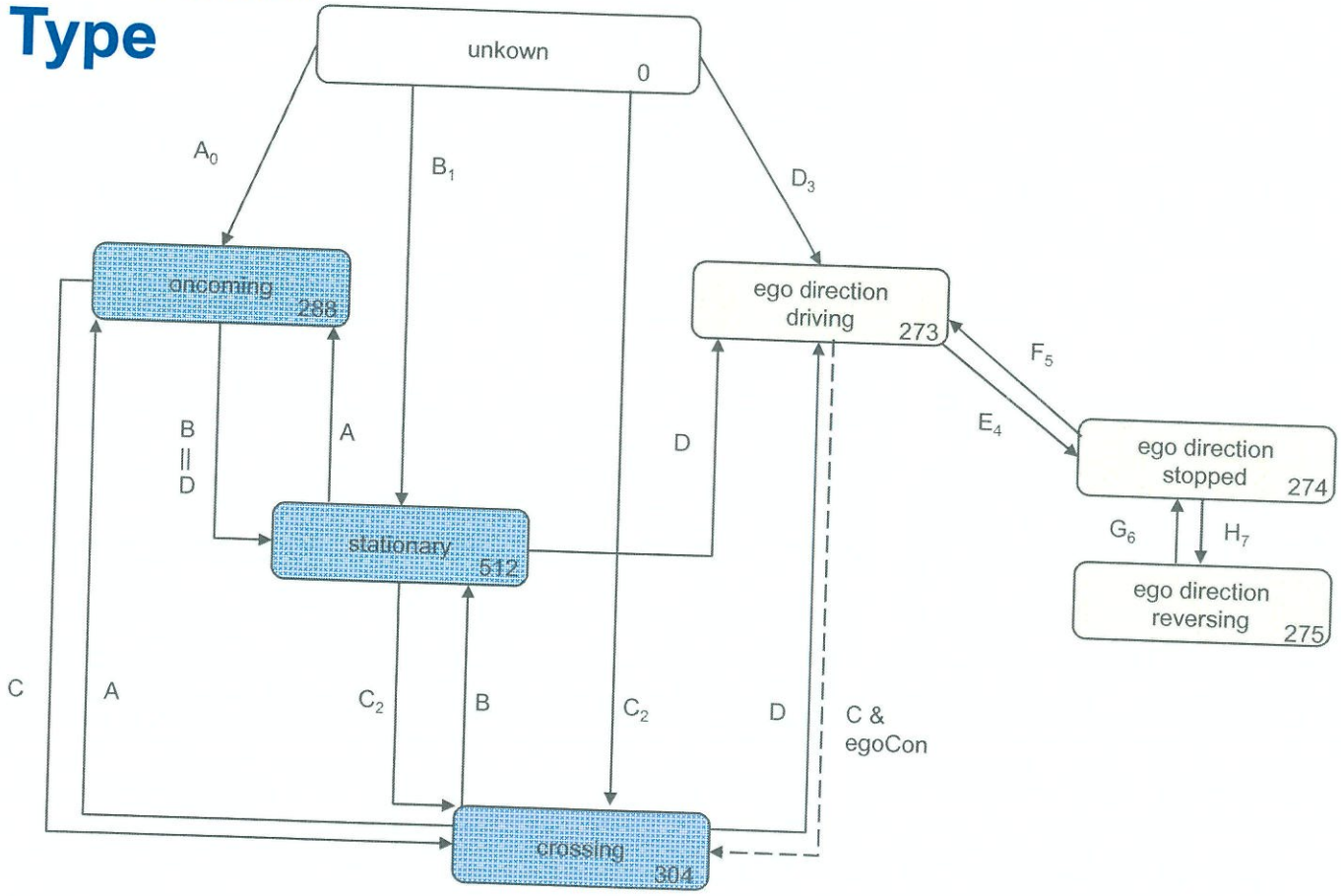
Concept description:

- Calculate Motion type of objects depending on past and current object motion
  Sensor independent calculation of motion type using absolute kinematic equations (sensor motion type is at the moment ignored).

Current implementation:

- Motion type is calculated for each track on the basis of a state machine in the LKF tracker.
- The motion type is calculated based on the heading, lateral velocity and longitudinal velocity of the object.
- At the time of creation of a new track the motion type is initialized as UNKOWN.

Output Converter Motion Type

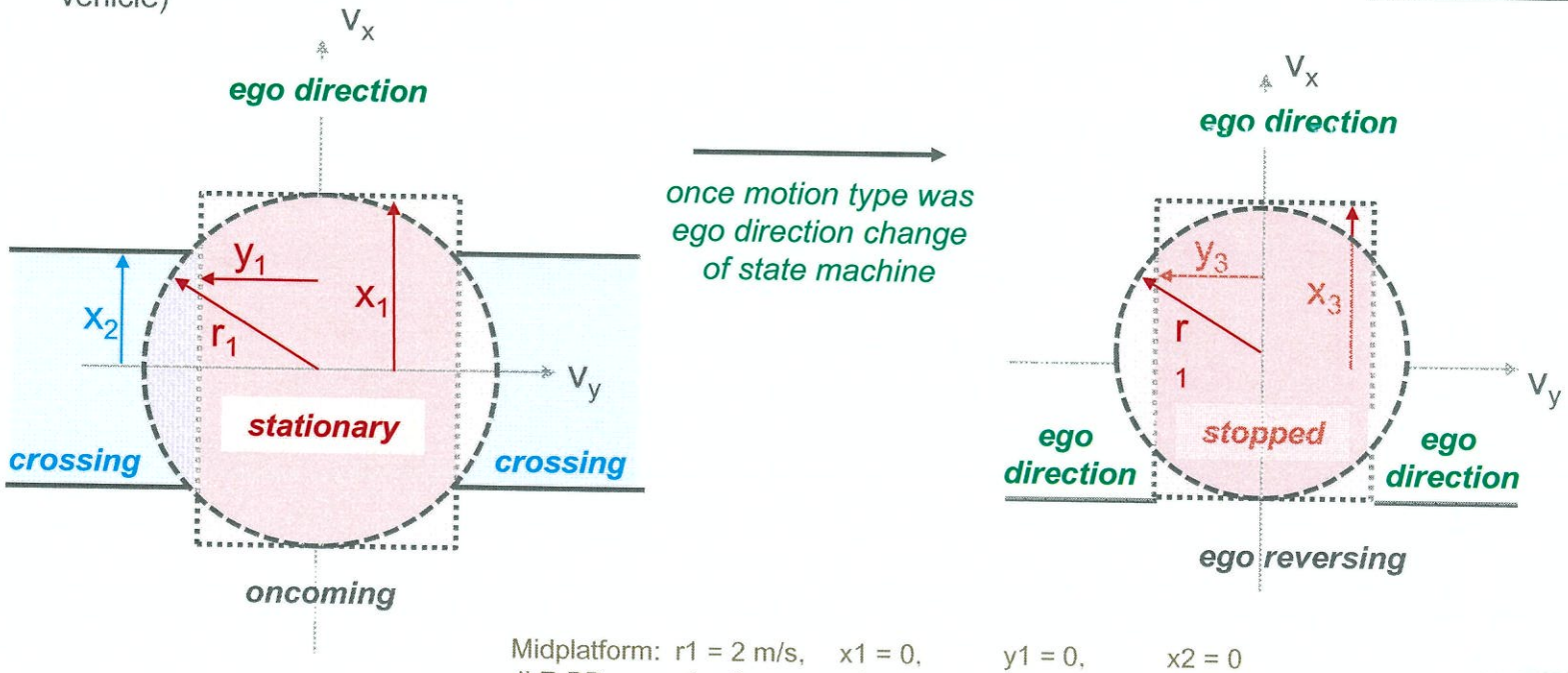


Output Converter Motion Type

Motion Types
UNKNOWN
MOVING_UNKNOWN
MOVING_EGODIRECTION_UNKNOWN
MOVING_EGODIRECTION_DRIVING
MOVING_EGODIRECTION_STOPPED
MOVING_EGODIRECTION_REVERSING
MOVING_ONCOMING
MOVING_CROSSING
STATIONARY

Current implementation:

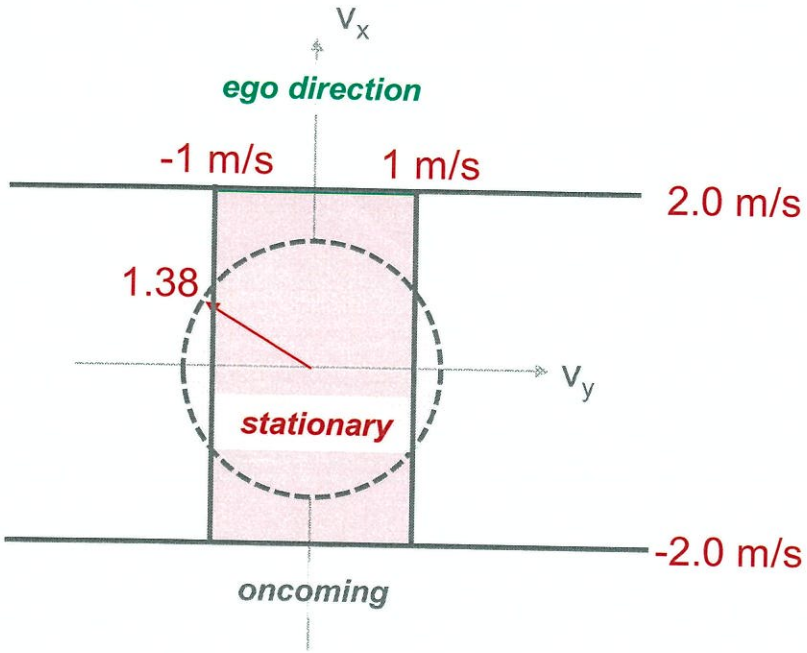
- The motion types are declared depending on the velocity over ground (relative to the host vehicle)



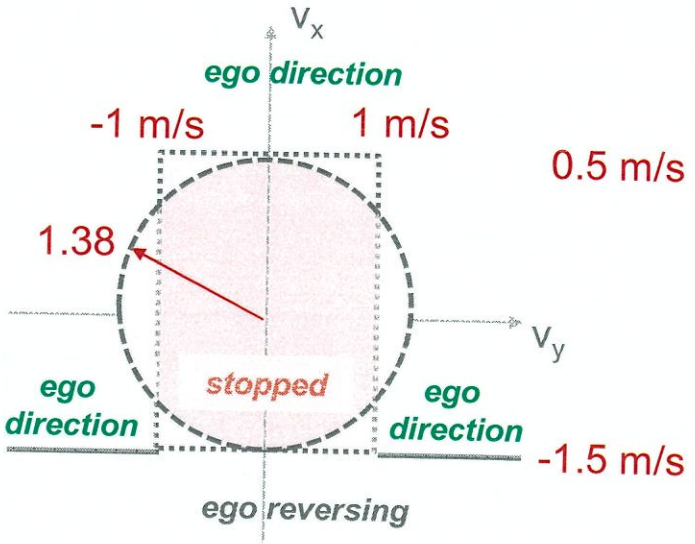
Midplatform: r1 = 2 m/s, x1 = 0, y1 = 0, x2 = 0
JLR BB: r1 = 0, x1 = 2 m/s y1 = 3 m/s x2 = 0

Output Converter Motion Type

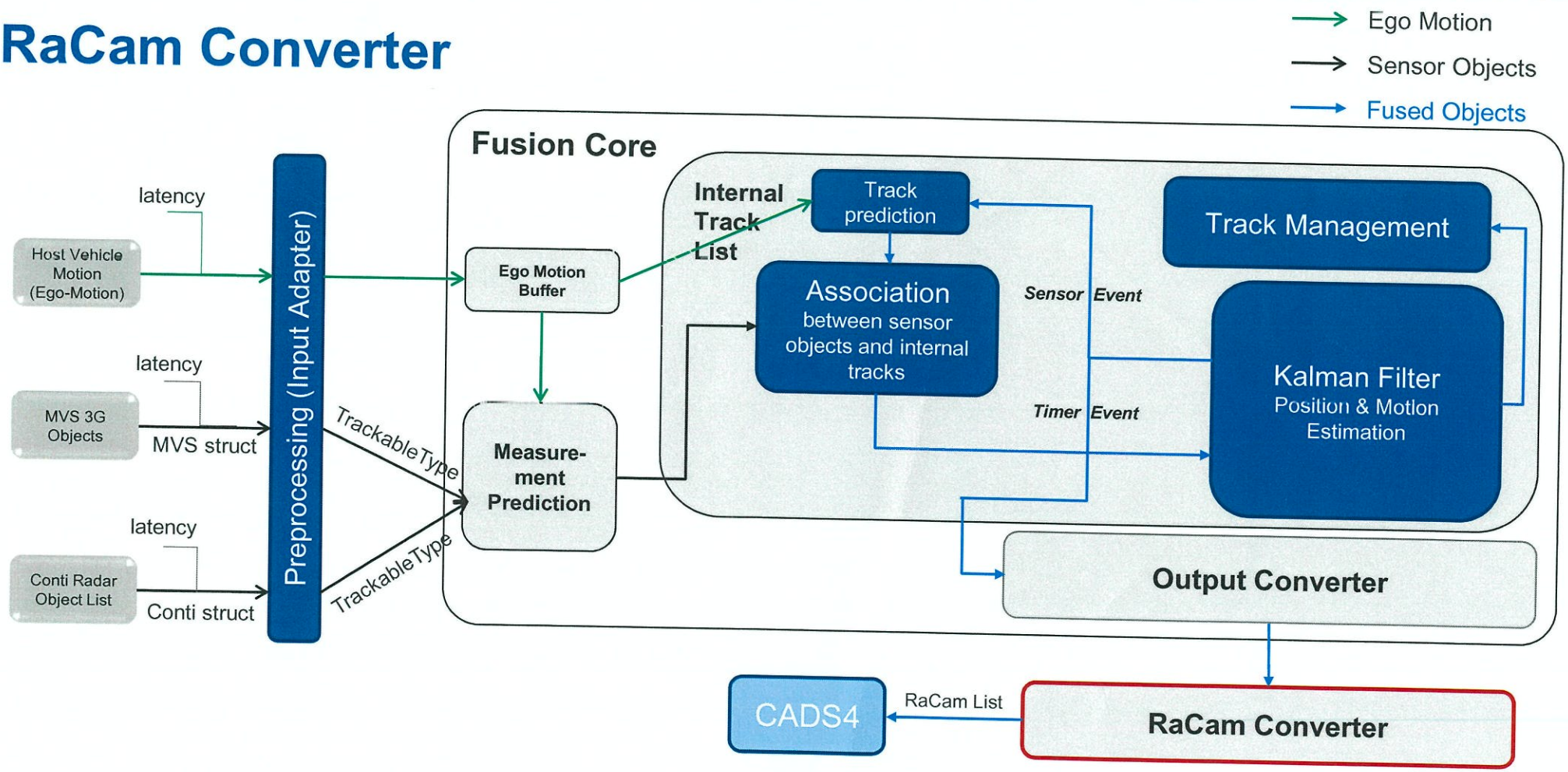
Current parametrization in Geely



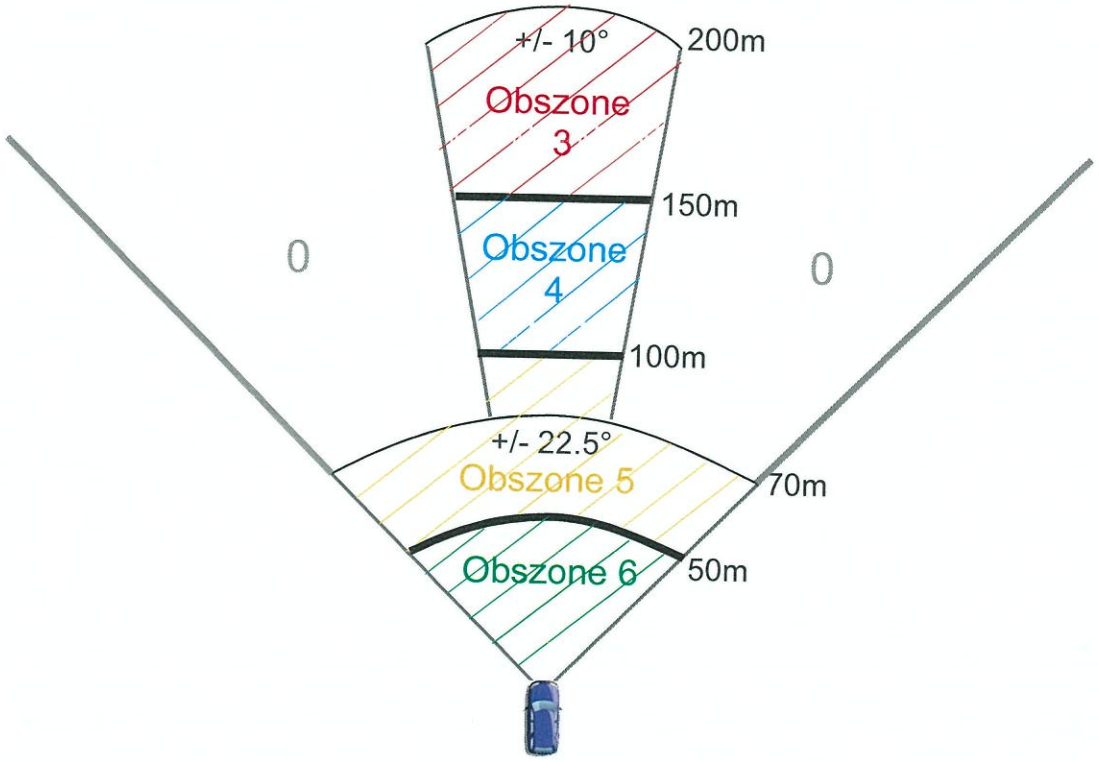
once motion type was ego direction change of state machine



RaCam Converter



Object Selection



Object Selection

- Remove objects with bucketing
  - Iterate through object list and allocate a bucket number to each object
  - Count number of objects in each bucket
  - Start to remove all objects from bucket (based on bucket priorities) if there are more objects than maxOutputObjects.

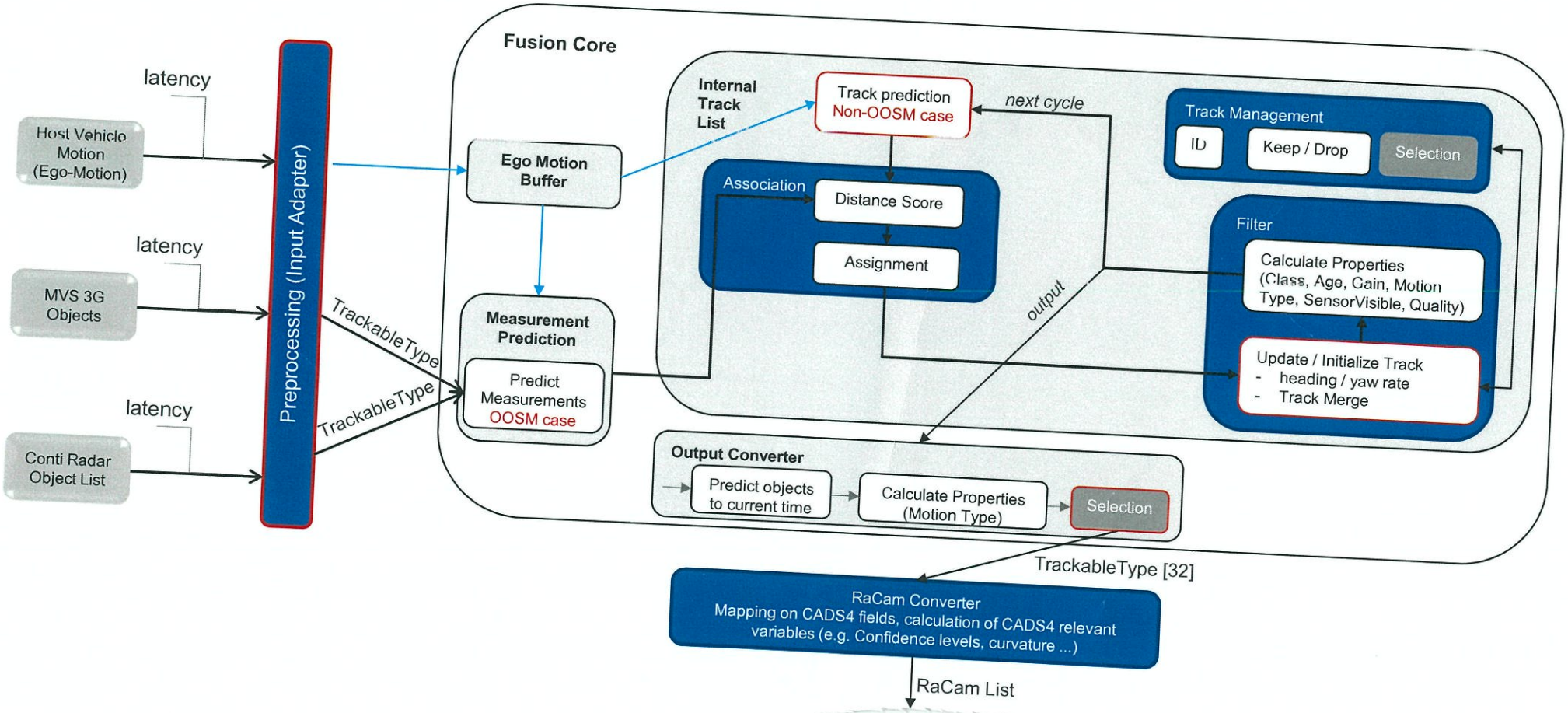
- Object Bucket Priorities
  - 1. Objects outside of MPF feature observation zone
  - 2. Stationary radar only objects
  - 3. **Coasted** sensor only objects with range > ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE4
  - 4. **Coasted** sensor only objects with range > ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE5
  - 5. **Coasted** sensor only objects with range > ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE6
  - 6. **Coasted** sensor only objects with range < ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE6
  - 7. **Coasted fused** objects with range > ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE4
  - 8. **Coasted fused** objects with range > ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE5
  - 9. **Coasted fused** objects with range > ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE6
  - 10. **Coasted fused** objects with range < ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE6
  - 11. **Non-coasted** objects with range > ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE4
  - 12. **Non-coasted** objects with range > ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE5
  - 13. **Non-coasted** objects with range > ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE6
  - 14. **Non-coasted** objects with range < ALV_PRV_OBJECTSELECTION_RANGE_THRESH_OBSZONE6

RaCam Converter

Maps generic Fusion output onto RaCam Structure ALF requires as input and calculates ALF specific signals

- AEB confidence values for each object
  - mainly which sensor reports the object and for how long
  - how stable is the position / velocity from the object
  - is the ID / class reported by sensor constant
- Specific motion parameters (ALF requires curvature (not yaw rate) / heading)
- Set ID from 0...31, map objects to fixed position and generate flag for new objects

Fusion Overview in detail



Object Fusion Example For AEB