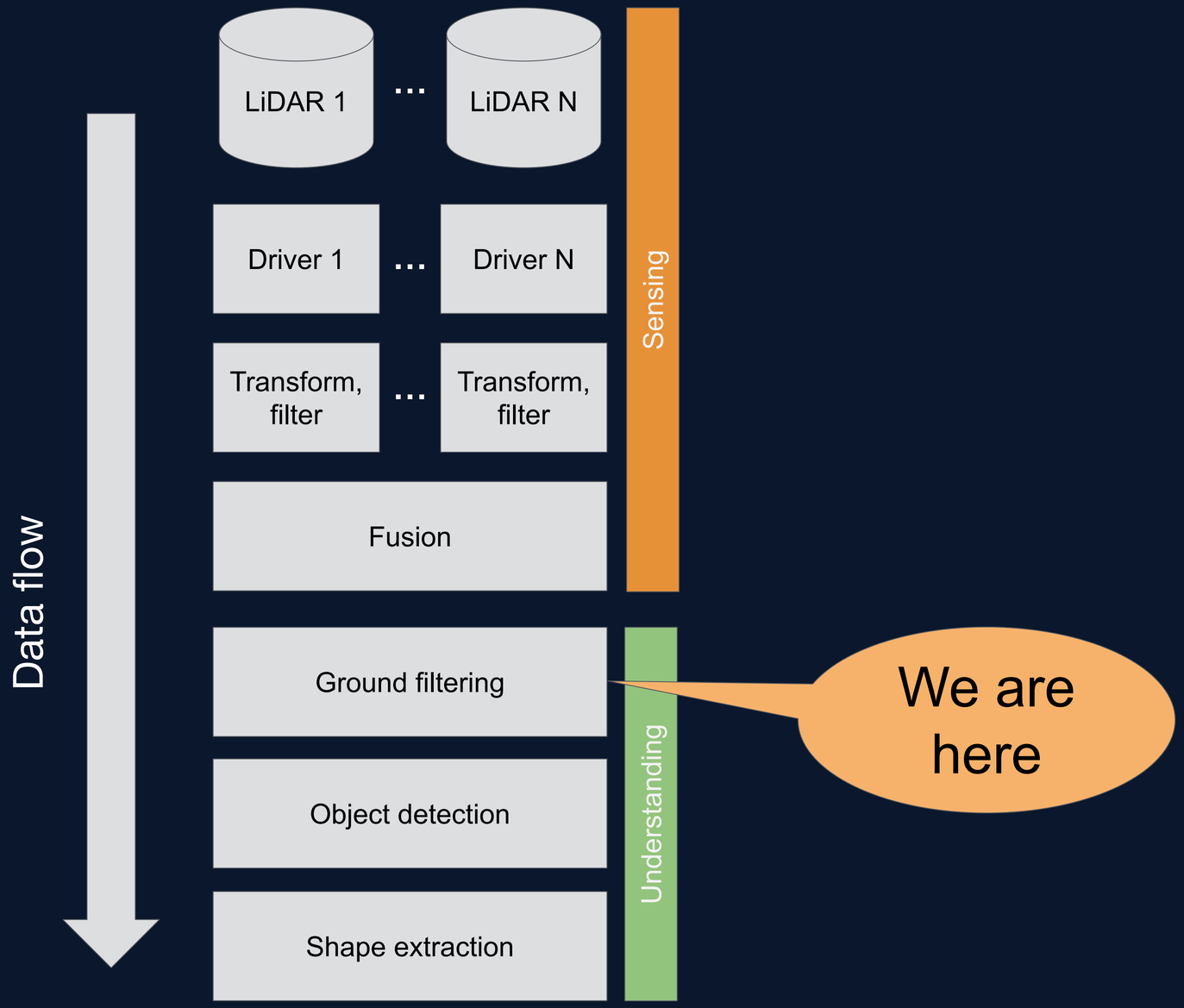
04 / Ground Filtering

Ground Filtering in the Classical LiDAR Processing Stack



The Problem of Ground Filtering

- Primary motivation for object detection is collision detection
- You can’t hit things on the ground
- We want the minimal sufficient information passed into our algorithm
- Ignore or *filter* useless* ground points

*: Useless in the context of object detection

Identifying ground points, a general strategy

1. Look at curvature/normals -> Moosman et al
2. Fit a big fat plane to the scene -> RANSAC
3. Look at rays/columns in depth image -> Petrovskaya & Thrun, Bogoslavskyi, Tier4, Cho et al
4. Other approaches (factor graphs, voxels, etc)

Expensive/slow (but possibly improved by our understanding of *near*)

Fast, nondeterministic, planar model is maybe not the best (i.e. road drainage)

Super fast, possible accuracy issues, generally relies on high vertical resolution

Generally slow, or throws away points

Autoware.Auto uses ray-based ground filtering because it is fast and deterministic

(Standard) Ray Ground Filtering

Petrovskaya & Thrun, Bogoslavskyi:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If it's pretty flat, ground
   b. If there's a big change in angle, non-ground from here out

(Standard) Ray Ground Filtering

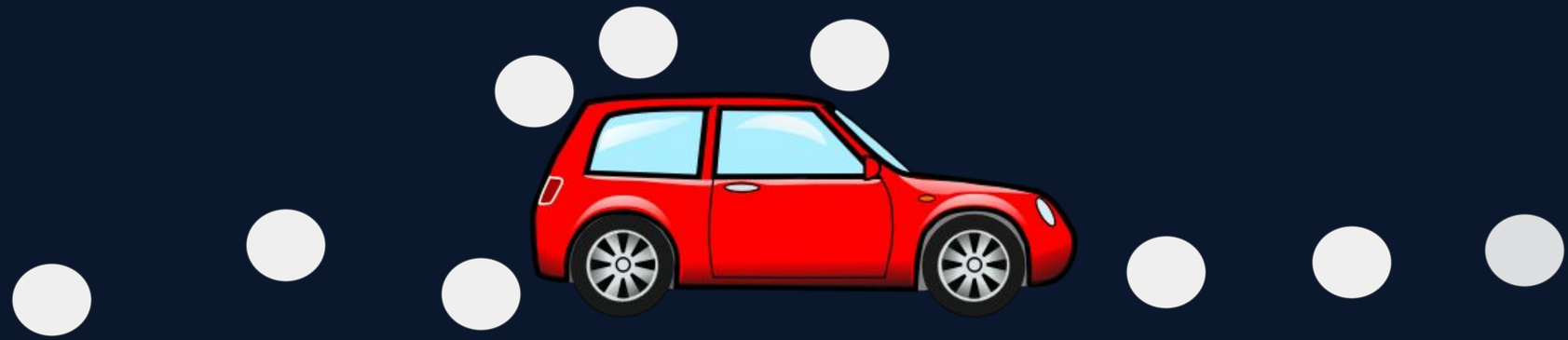
Petrovskaya & Thrun, Bogoslavskyi:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If it's pretty flat, ground
   b. If there's a big change in angle, non-ground from here out

Unclassified

Non-ground

Ground



(Standard) Ray Ground Filtering

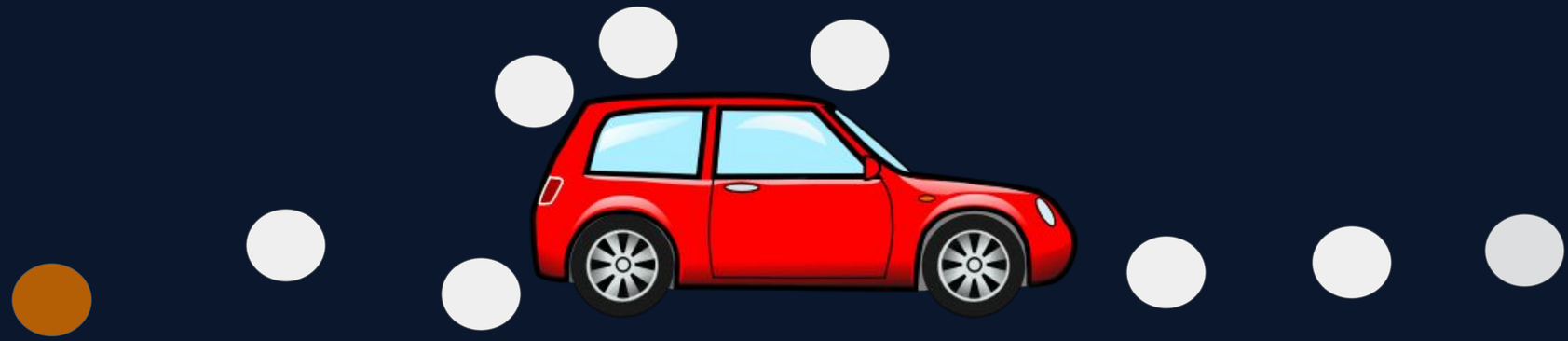
Petrovskaya & Thrun, Bogoslavskyi:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If it's pretty flat, ground
   b. If there's a big change in angle, non-ground from here out

Unclassified

Non-ground

Ground



(Standard) Ray Ground Filtering

Petrovskaya & Thrun, Bogoslavskyi:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If it's pretty flat, ground
   b. If there's a big change in angle, non-ground from here out

Unclassified

Non-ground

Ground

(Standard) Ray Ground Filtering

Petrovskaya & Thrun, Bogoslavskyi:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If it's pretty flat, ground
   b. If there's a big change in angle, non-ground from here out

Unclassified

Non-ground

Ground

(Standard) Ray Ground Filtering

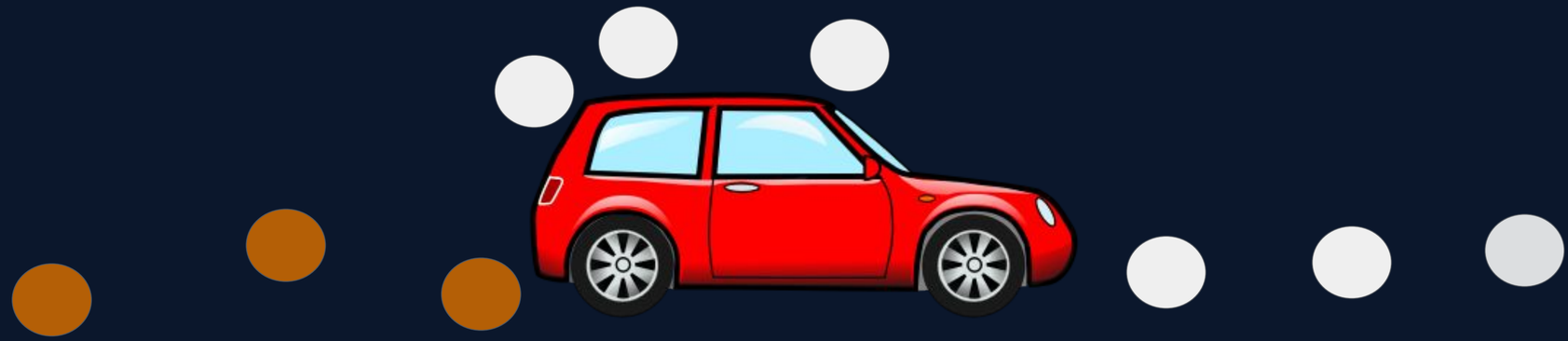
Petrovskaya & Thrun, Bogoslavskyi:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If it's pretty flat, ground
   b. If there's a big change in angle, non-ground from here out

Unclassified

Non-ground

Ground



(Standard) Ray Ground Filtering

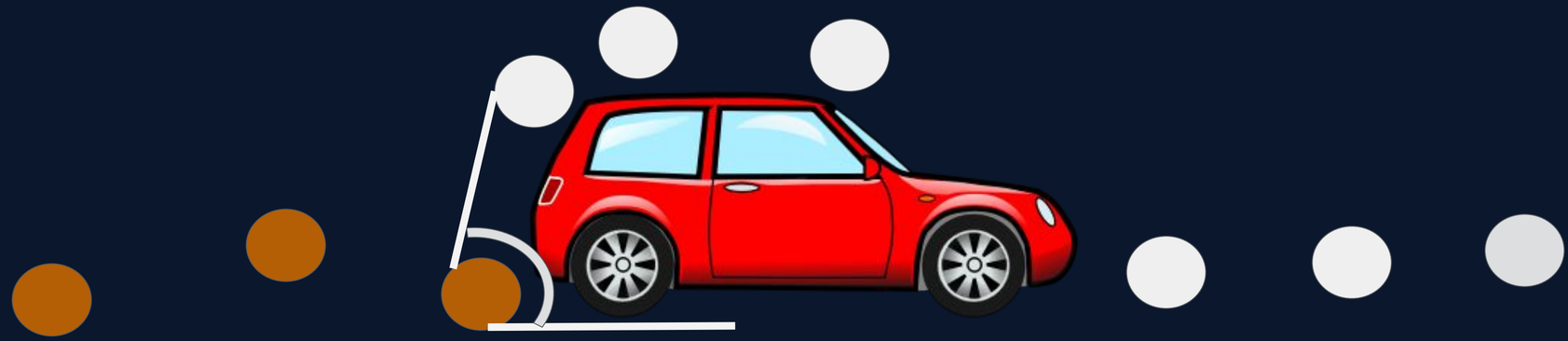
Petrovskaya & Thrun, Bogoslavskyi:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If it's pretty flat, ground
   b. If there's a big change in angle, non-ground from here out

Unclassified

Non-ground

Ground



Θ > thresh

(Standard) Ray Ground Filtering

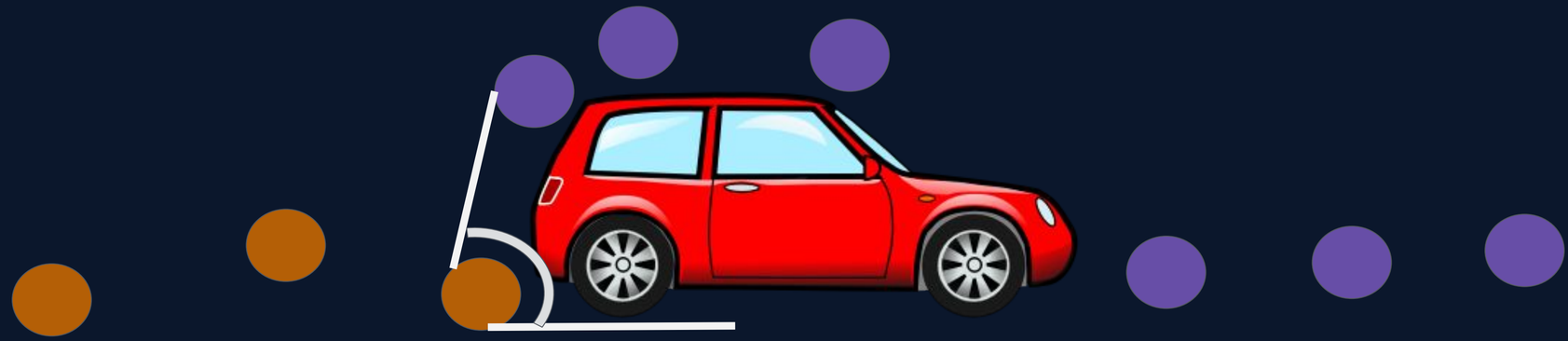
Petrovskaya & Thrun, Bogoslavskyi:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If it's pretty flat, ground
   b. If there's a big change in angle, non-ground from here out

Unclassified

Non-ground

Ground



Θ > thresh

Autoware.AI's Ray Ground Filtering

Cho et al, Tier4:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If close to last point:
      i. Check if point is in local cone WRT last ground point → ground/non-ground
   b. Otherwise, check global cone from local ground

Autoware.AI's Ray Ground Filtering

Cho et al, Tier4:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If close to last point:
      i. Check if point is in local cone WRT last ground point → ground/non-ground
   b. Otherwise, check global cone from local ground

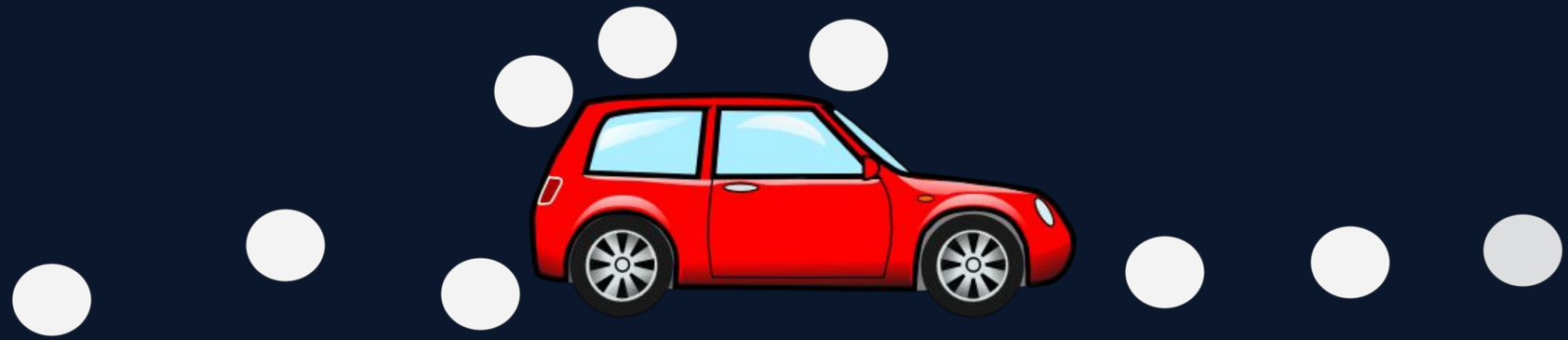
Unclassified

Non-ground

Ground

Global cone

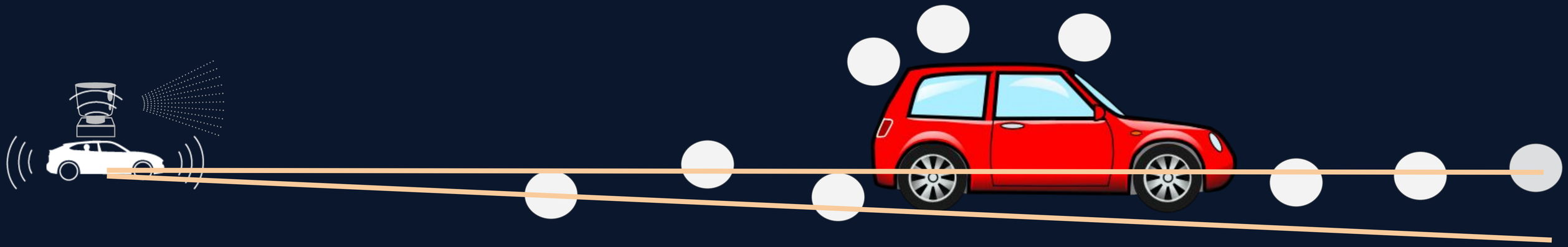
Local cone



Autoware.AI's Ray Ground Filtering

Cho et al, Tier4:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If close to last point:
      i. Check if point is in local cone WRT last ground point → ground/non-ground
   b. Otherwise, check global cone from local ground

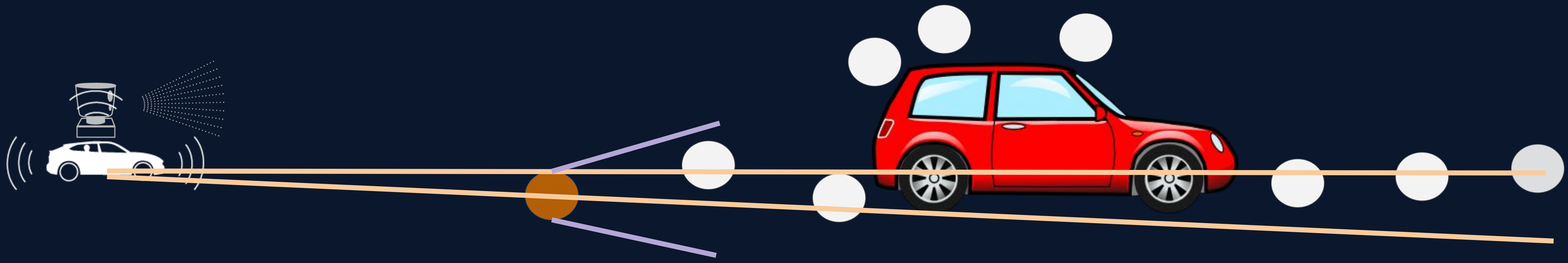


Autoware.AI's Ray Ground Filtering

Cho et al, Tier4:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If close to last point:
      i. Check if point is in local cone WRT last ground point → ground/non-ground
   b. Otherwise, check global cone from local ground

Unclassified
Non-ground
Ground
Global cone
Local cone

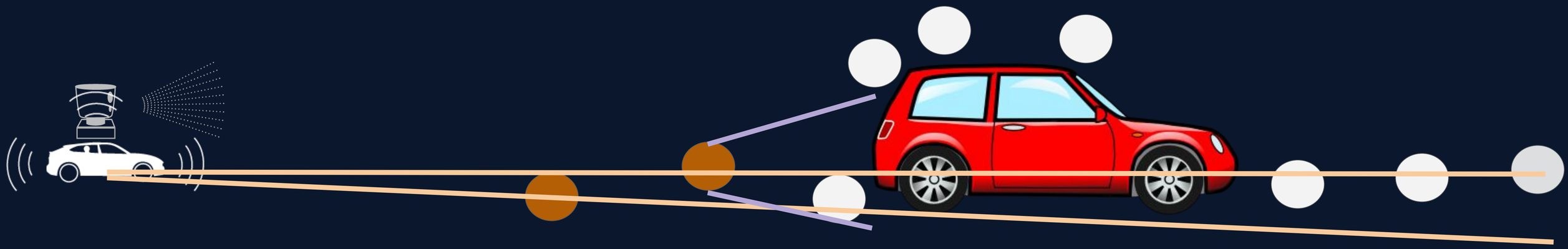


Autoware.AI's Ray Ground Filtering

Cho et al, Tier4:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If close to last point:
      i. Check if point is in local cone WRT last ground point → ground/non-ground
   b. Otherwise, check global cone from local ground

Unclassified
Non-ground
Ground
Global cone
Local cone

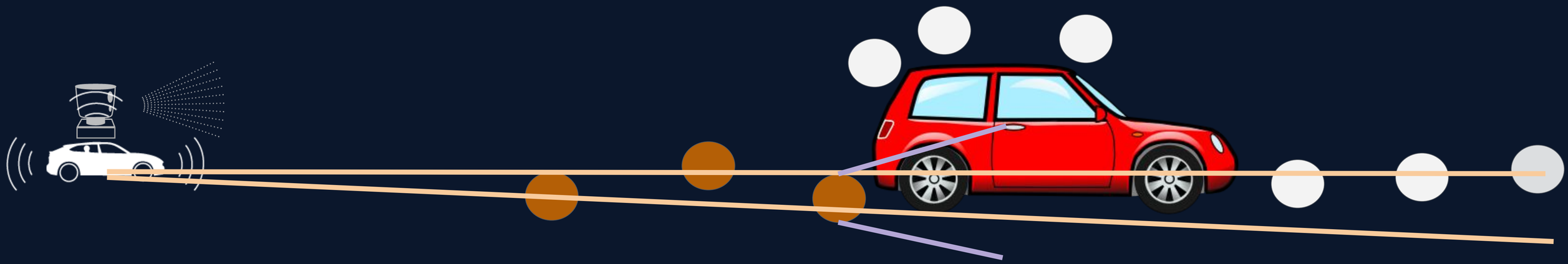


Autoware.AI's Ray Ground Filtering

Cho et al, Tier4:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If close to last point:
      i. Check if point is in local cone WRT last ground point → ground/non-ground
   b. Otherwise, check global cone from local ground

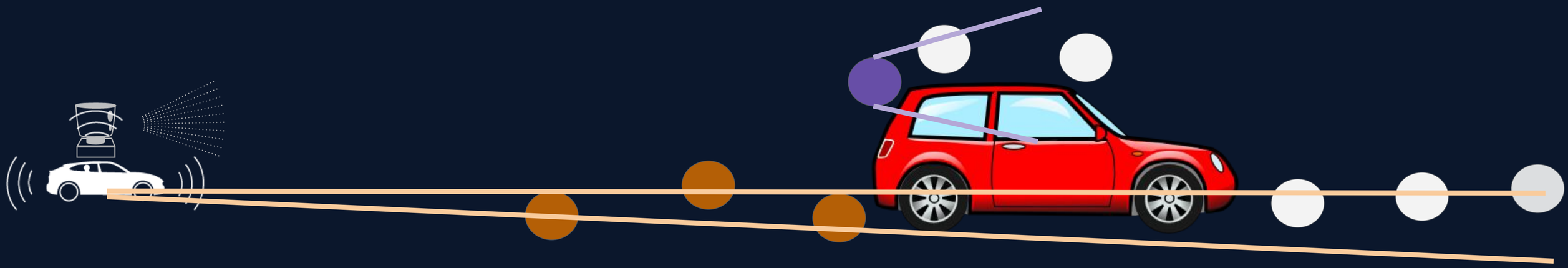
Unclassified
Non-ground
Ground
Global cone
Local cone



Autoware.AI's Ray Ground Filtering

Cho et al, Tier4:

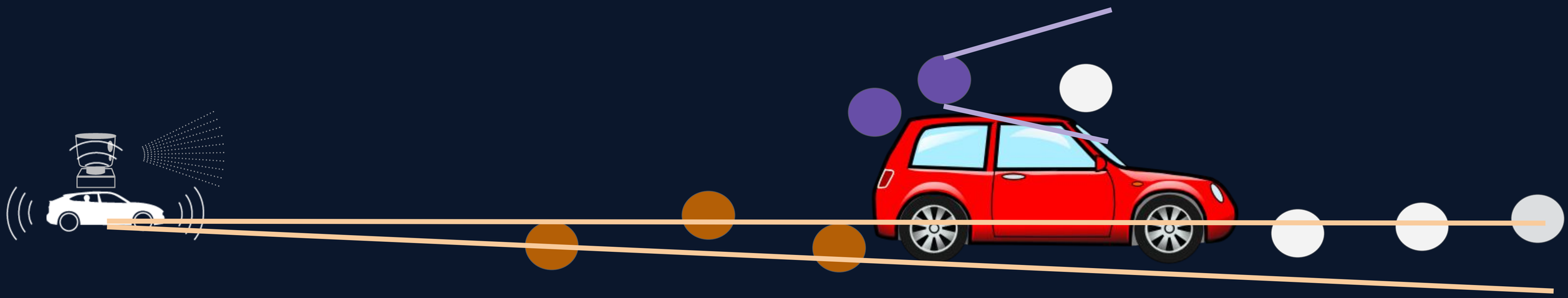
1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If close to last point:
      i. Check if point is in local cone WRT last ground point → ground/non-ground
   b. Otherwise, check global cone from local ground



Autoware.AI's Ray Ground Filtering

Cho et al, Tier4:

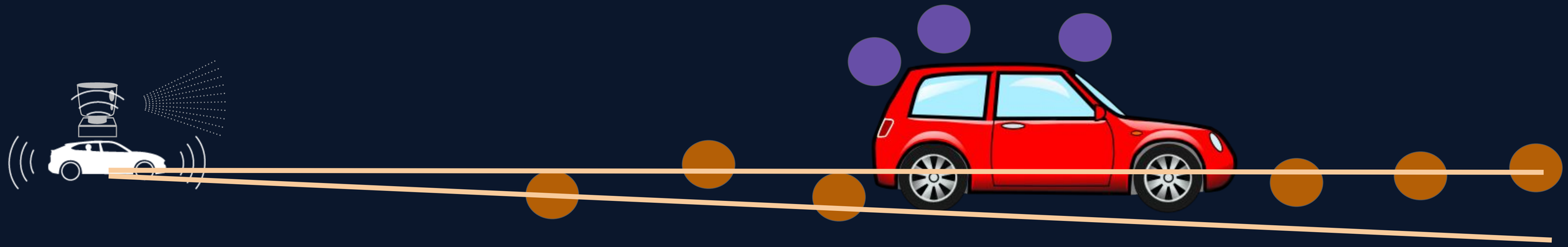
1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If close to last point:
      i. Check if point is in local cone WRT last ground point → ground/non-ground
   b. Otherwise, check global cone from local ground



Autoware.AI's Ray Ground Filtering

Cho et al, Tier4:

1. Build range image or otherwise bin into angle slices
2. Know where the ground is local to the sensor
3. Scan through points in a ray with increasing distance:
   a. If close to last point:
      i. Check if point is in local cone WRT last ground point → ground/non-ground
   b. Otherwise, check global cone from local ground



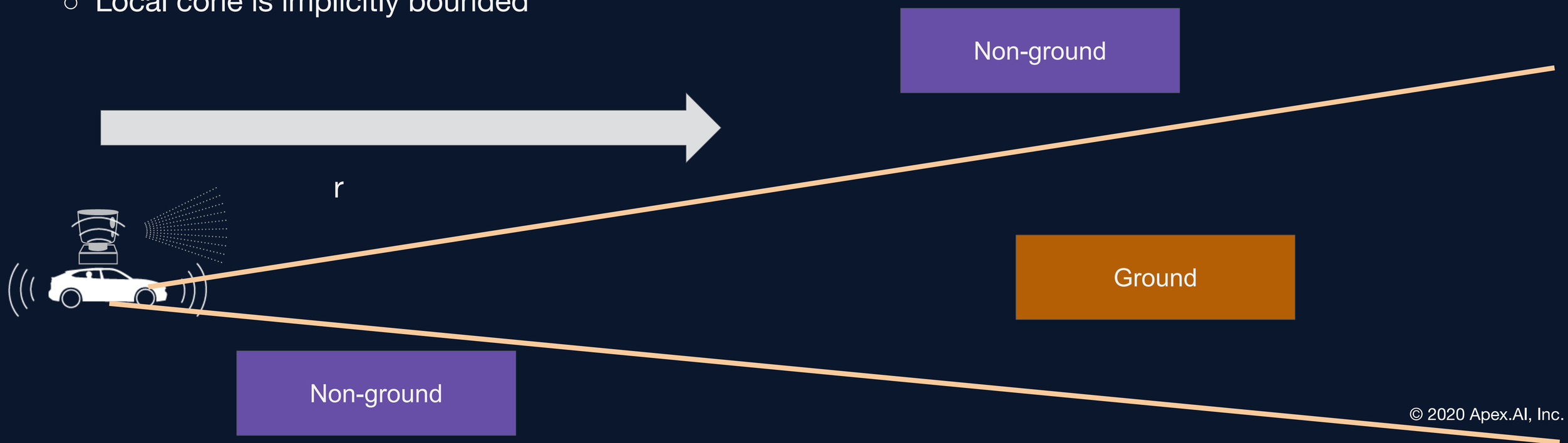
Ground Filtering in Autoware.Auto

Use the Autoware.AI approach with three key improvements:

1. Threshold statistics
2. Use domain knowledge
3. Reinterpret as a factor graph

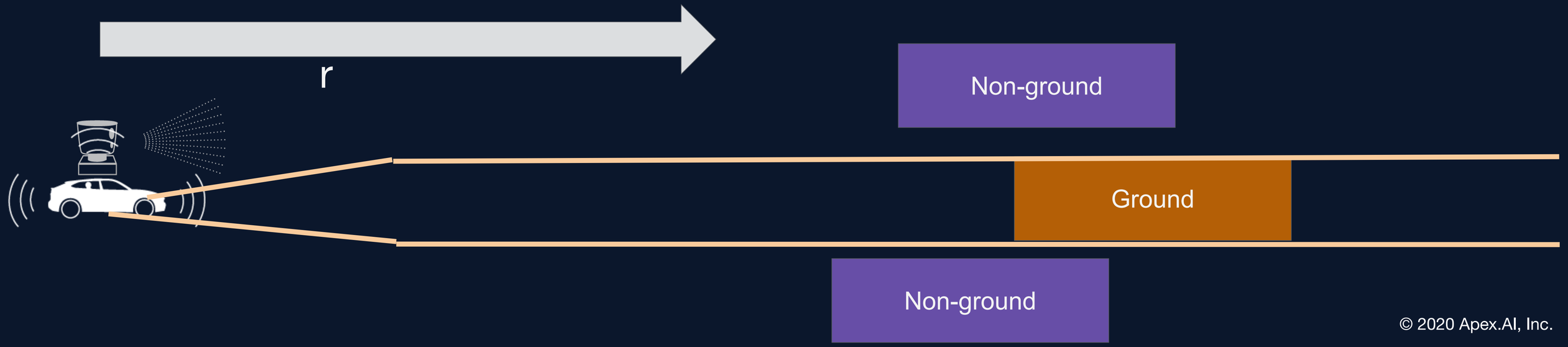
Thresholding Statistics

- Local and global threshold are unbounded in original implementation
  - As r → ∞, all points become ground
  - Bad detection at range
- Threshold global cone statistics
  - Control unbounded growth of thresholds
  - More robust filtering at range
  - Local cone is implicitly bounded



Thresholding Statistics

- Local and global threshold are unbounded in original implementation
  - As r → ∞, all points become ground
  - Bad detection at range
- Threshold global cone statistics
  - Control unbounded growth of thresholds
  - More robust filtering at range
  - Local cone is implicitly bounded



Adding Domain Knowledge (aka heuristics)

An almost message-passing algorithm:

1. Sort points by radial distance
2. Classify point based on immediate information into:
   a. GROUND
   b. PROVISIONAL_GROUND
   c. NONGROUND
   d. RETRO_NONGROUND
   e. NONLOCAL_NONGROUND
3. Pass information *back*: Update previous label based on current label:
   a. RETRO_NONGROUND -> last point becomes nonground
   b. PROVISIONAL_GROUND -> becomes nonground if next point is local nonground
   c. NONLOCAL_NONGROUND -> doesn't cause PROVISIONAL_GROUND to become nonground

- Slightly more computational effort
- But much more expressive

Need as many ground points as possible on distant objects

Handle low objects e.g. in front of buildings -> Better segmentation

RETRO_NONGROUND

- Need as many points as possible on distant objects for clustering
- Send information *back* in ray
- Intended to detect “vertical structure”
  - AKA something you can hit

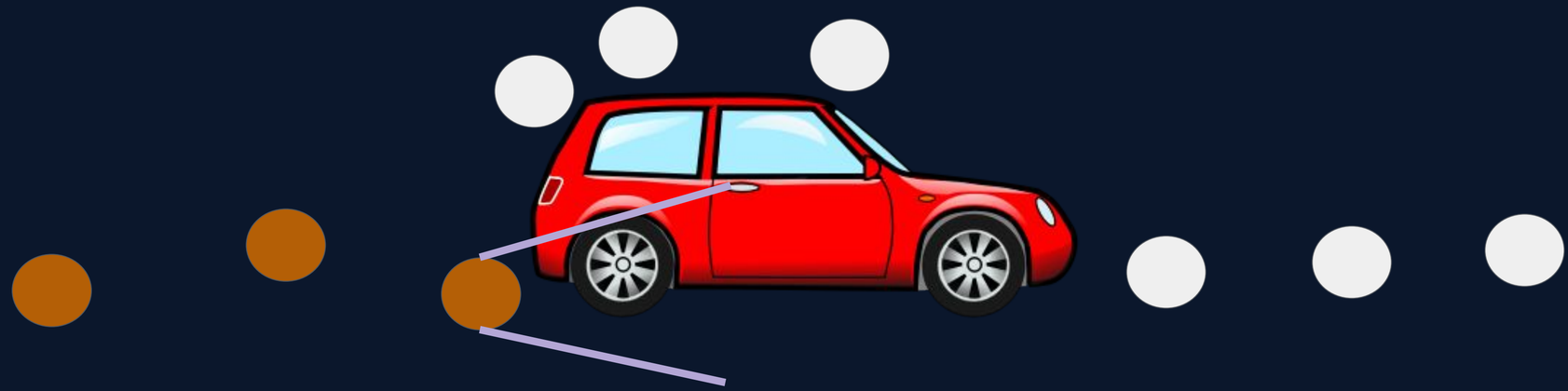
Unclassified

Non-ground

Ground

Global cone

Local cone



RETRO_NONGROUND

- Need as many points as possible on distant objects for clustering
- Send information *back* in ray
- Intended to detect “vertical structure”
  - AKA something you can hit

RETRO_NONGROUND

- Need as many points as possible on distant objects for clustering
- Send information *back* in ray
- Intended to detect “vertical structure”
  - AKA something you can hit

Unclassified

Non-ground

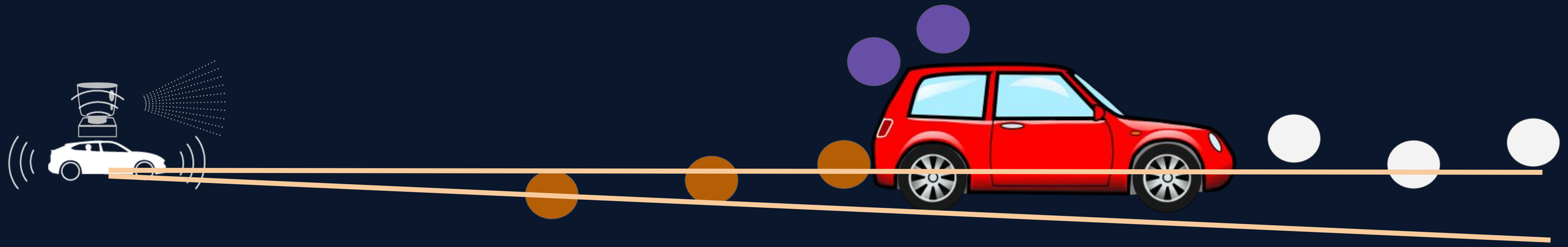
Ground

Global cone

Local cone

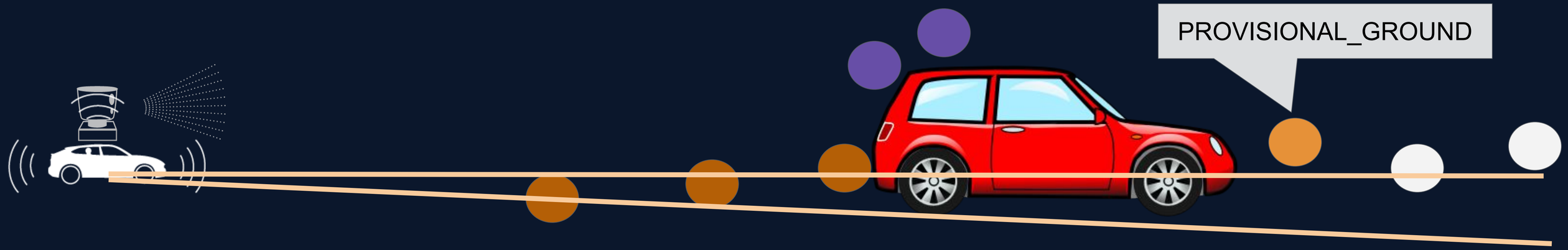
PROVISIONAL_(NON)GROUND

- Adds a “weak” label between using local classification and global classification
  - Local -> strong similarity/label
  - Global -> educated guess
- When moving from one class to another, apply weak label based on vertical proximity
- Label changes based on next (local) point



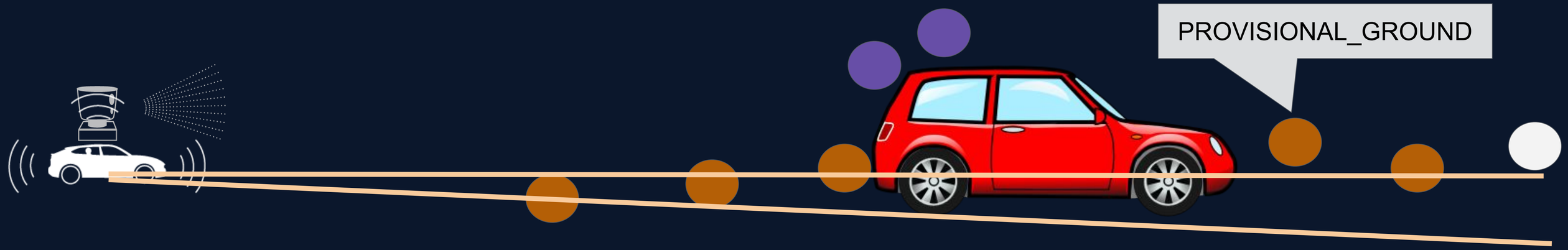
PROVISIONAL_(NON)GROUND

- Adds a “weak” label between using local classification and global classification
  - Local -> strong similarity/label
  - Global -> educated guess
- When moving from one class to another, apply weak label based on vertical proximity
- Label changes based on next (local) point



PROVISIONAL_(NON)GROUND

- Adds a “weak” label between using local classification and global classification
  - Local -> strong similarity/label
  - Global -> educated guess
- When moving from one class to another, apply weak label based on vertical proximity
- Label changes based on next (local) point



Interpretation as constraint satisfaction

Dynamic programming view:

1. Scan through sorted points
2. Classify points based on maintained state

Factor graph/Markov network view

1. Each point in ray has a label
2. Given labels, adjacent points must have some relationship
3. Find joint set of labels which is most likely

See e.g. More Global Matching for another example of reimagining a classic algorithm as a factor graph

Can build probabilistic model for ground filtering:

- Let x = (Label, distance, height)
- Assume conditional independence
  - Unary factors: $P(x_i)$
  - N-ary factors: $P(x_i \mid x_{i-1}, x_{i+1})$
- $P(x1, \ldots., xN) = \Pi_{i=1}^{N} P(x_i)\, P(x_i \mid x_{i-1}, x_{i+1})$
- Solve via message passing algorithms
- Find factors via machine learning

Final Algorithm

...But we didn't do that

1. (Optional) bin point cloud into rays
2. For each ray:
    a. Sort point in ray by radial distance
    b. Last label = GROUND, Last point = (0, 0)
    c. For each point in order:
        i. If close to last point:
            1. Super steep angle -> RETRO_NONGROUND
            2. Close in height -> Same label as last
        ii. Otherwise nonlocal:
            1. Check if local to last ground point -> PROVISIONAL_NONGROUND
            2. Check against global cone -> NONGROUND/PROVISIONAL_GROUND
        iii. Update last label:
            1. If RETRO_NONGROUND -> last = NONGROUND
            2. Update PROVISIONAL_GROUND label -> Takes decayed version of current point

Ground Filtering - Summary

- Ground filtering simplifies the problem of object detection
- There are many approaches:
  - Region-growing
  - Voxel-based
  - Ray-based
- Autoware.Auto uses a ray-based approach

Important optimizations

- Threshold statistics
  - Better classification at range
- Add additional domain knowledge
  - Better accuracy
- Reinterpret problem as factor graph
  - Better at classification
  - Not fully realized--still room for improvement