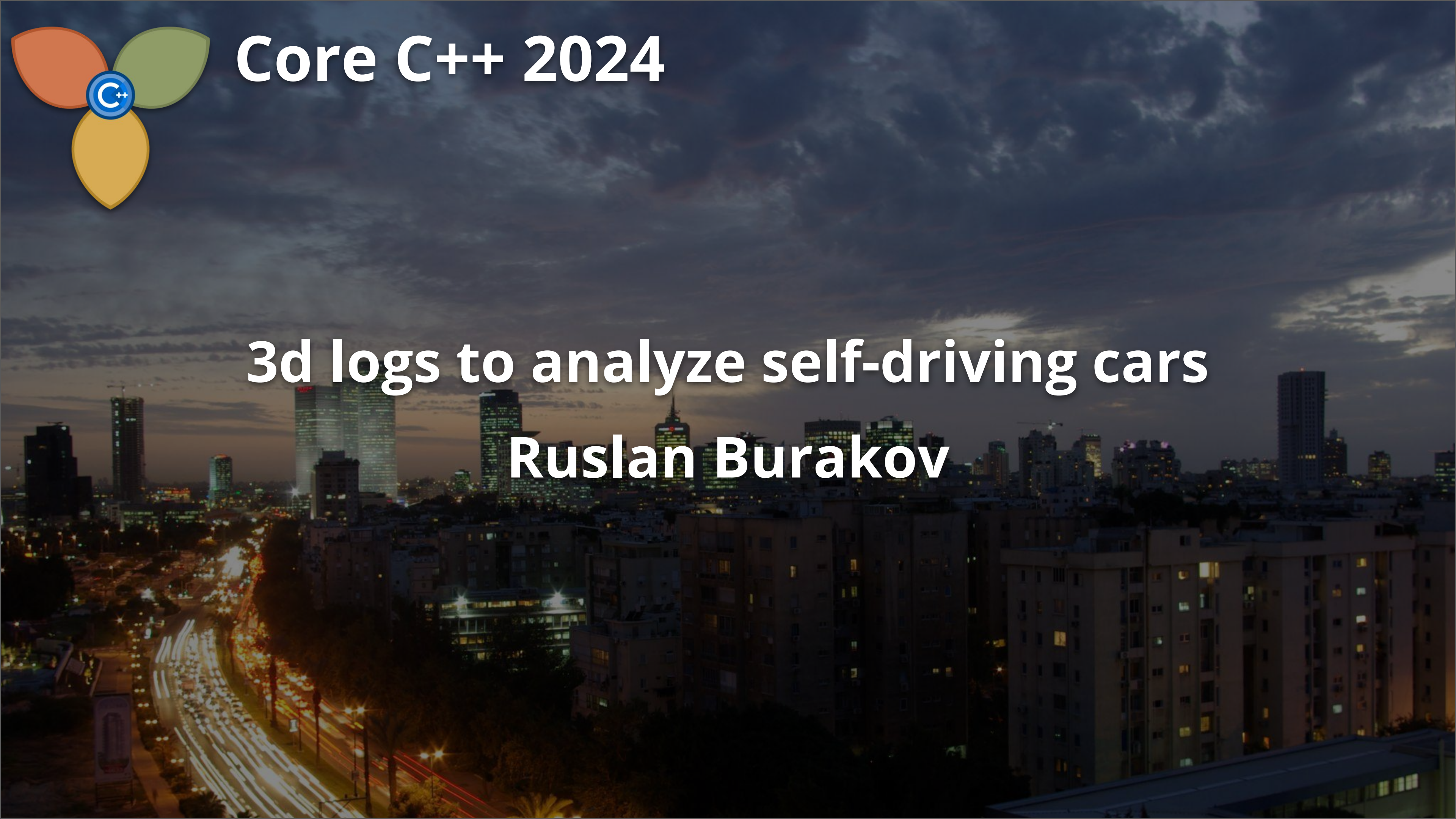
Core C++ 2024



3d logs to analyze self-driving cars

Ruslan Burakov

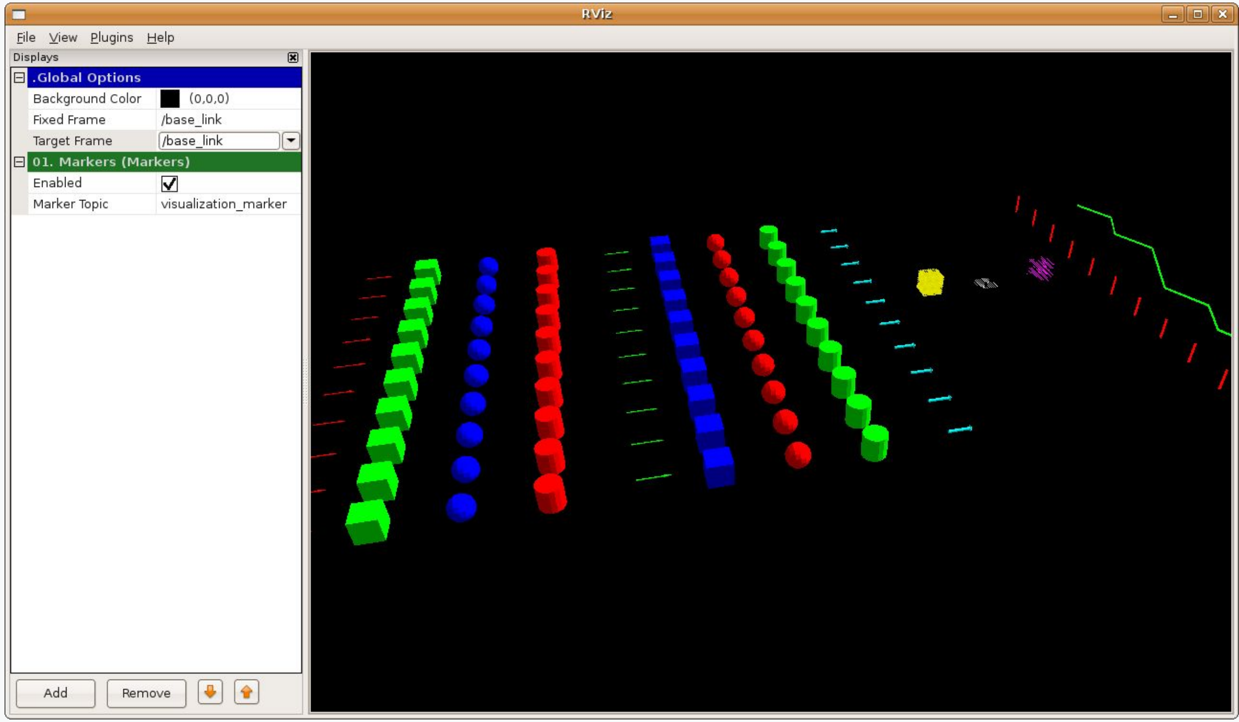
AVRIDE

01
Visualization for self-driving ride

Visualization for self-driving ride

Rviz

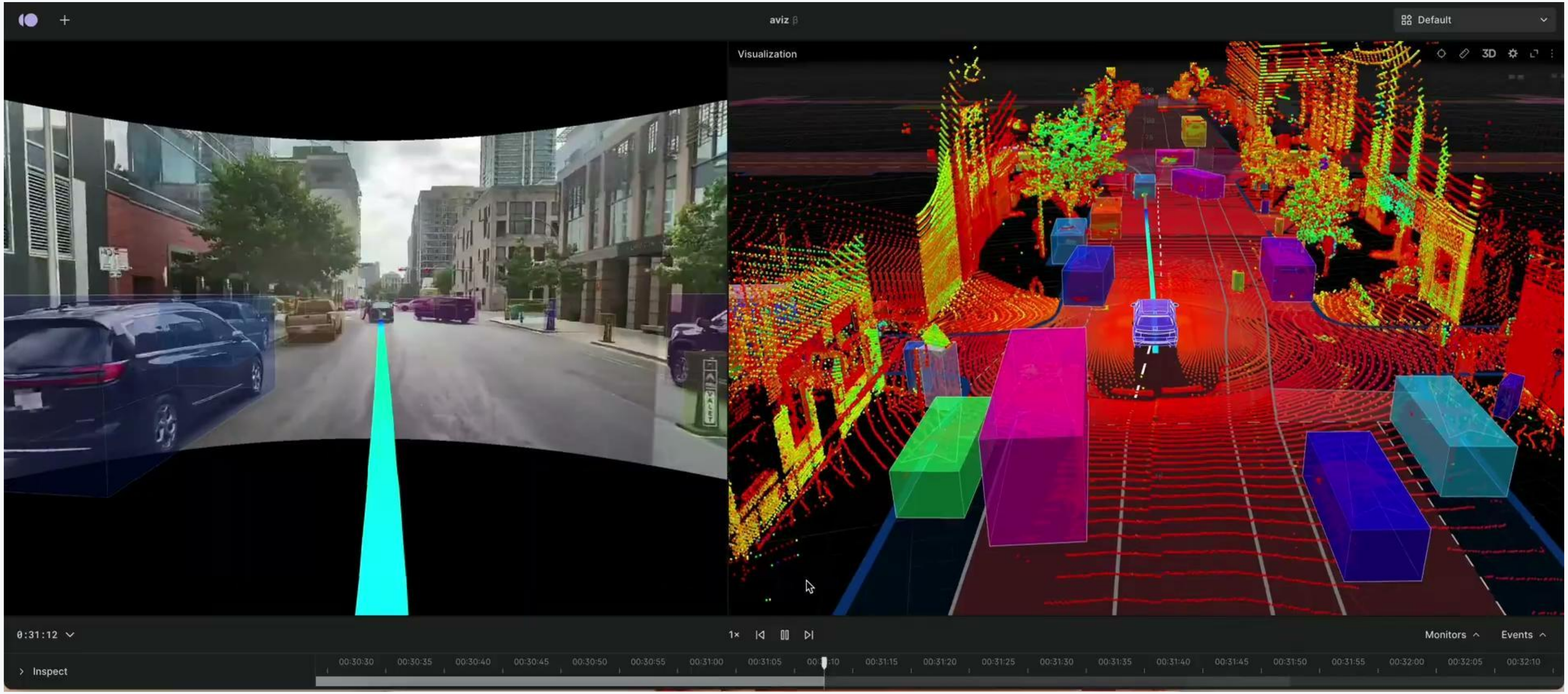
- Desktop
- Performance issues
- No cloud



.Global Options
Background Color (0,0,0)
Fixed Frame /base_link
Target Frame /base_link
01. Markers (Markers)
Enabled ✓
Marker Topic visualization_marker

Rviz: image courtesy of Open Robotics

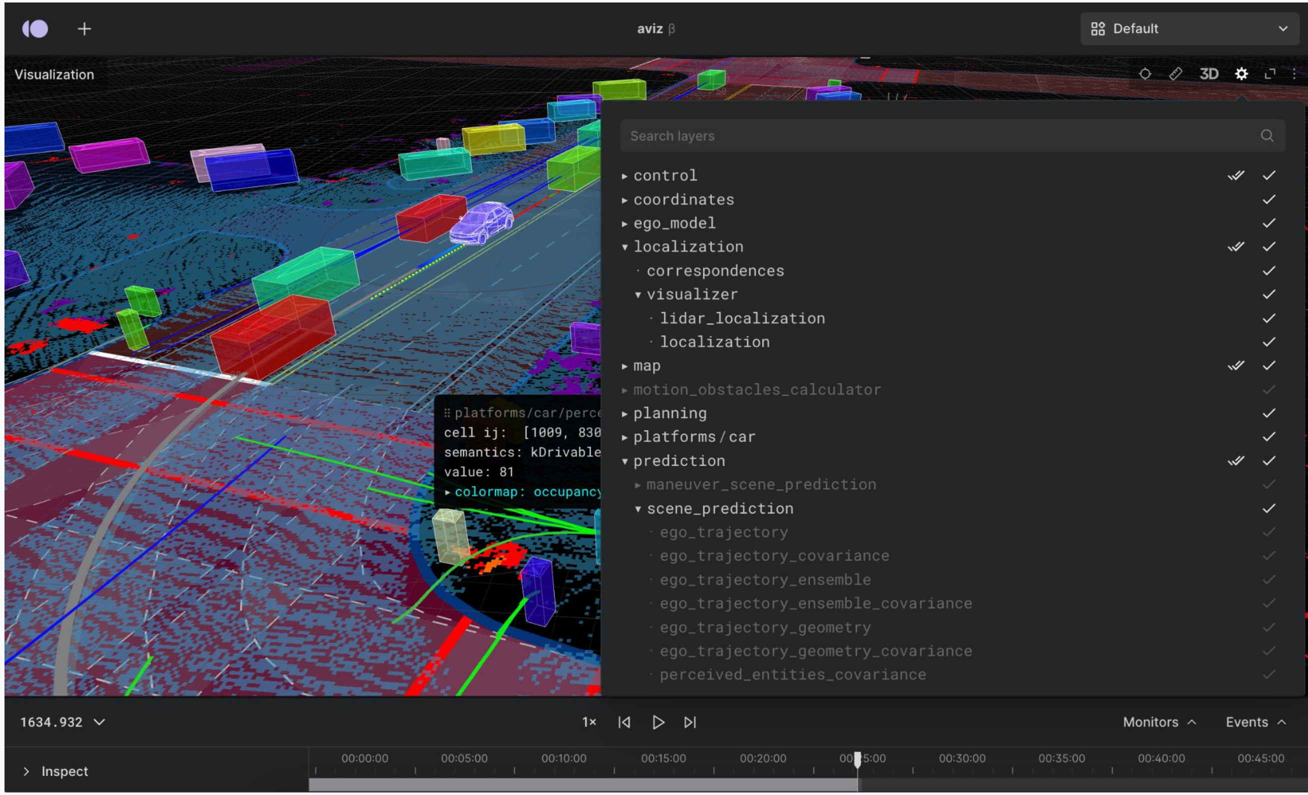
Visualization for self-driving ride



Visualization for self-driving ride

Aviz

- Web
- Cloud Streaming
- Read optimization
- Write optimization



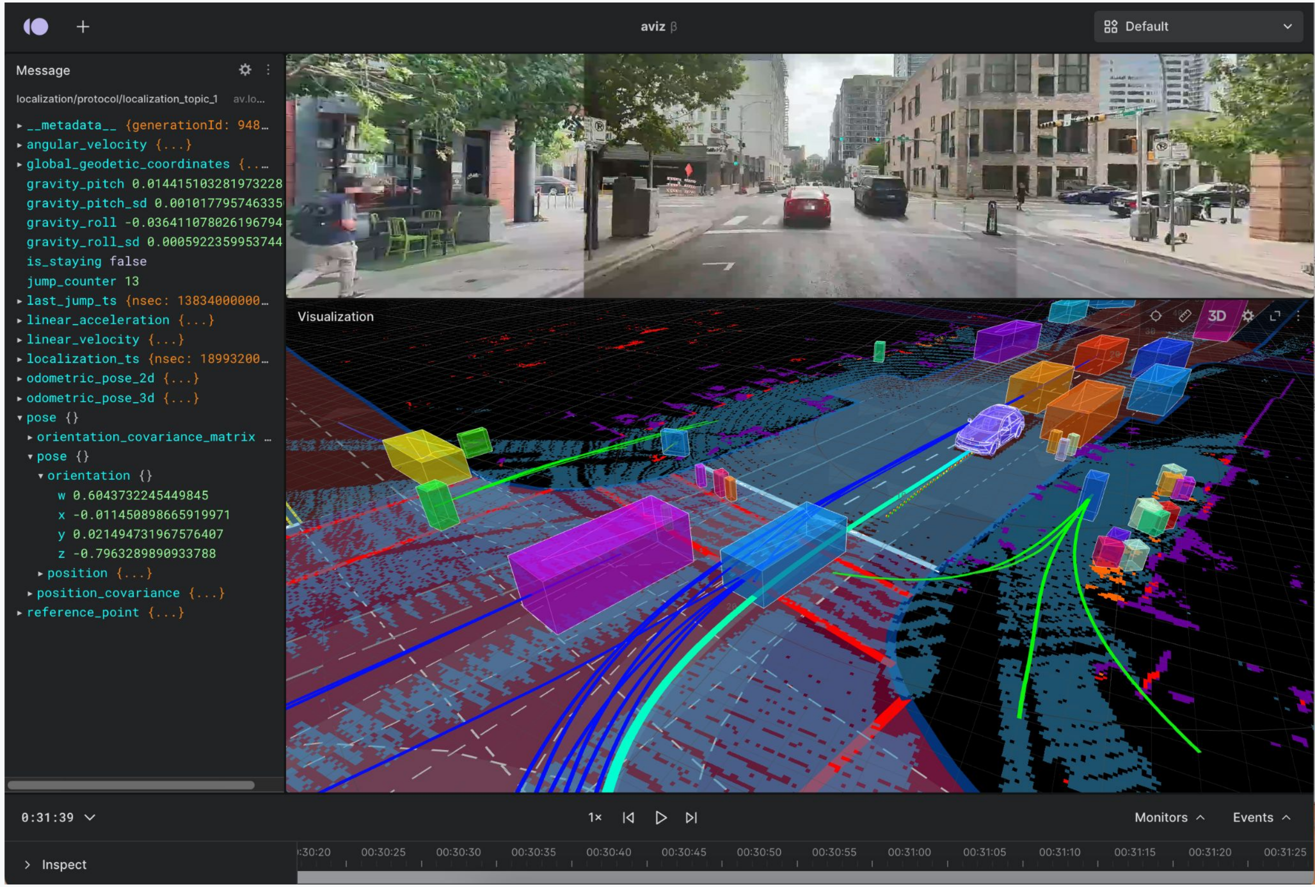
```
platforms/car/perce
cell ij:  [1009, 830
semantics: kDrivable
value: 81
▸ colormap: occupancy
```

02
Read Optimization

Read Optimization

Read Requirements

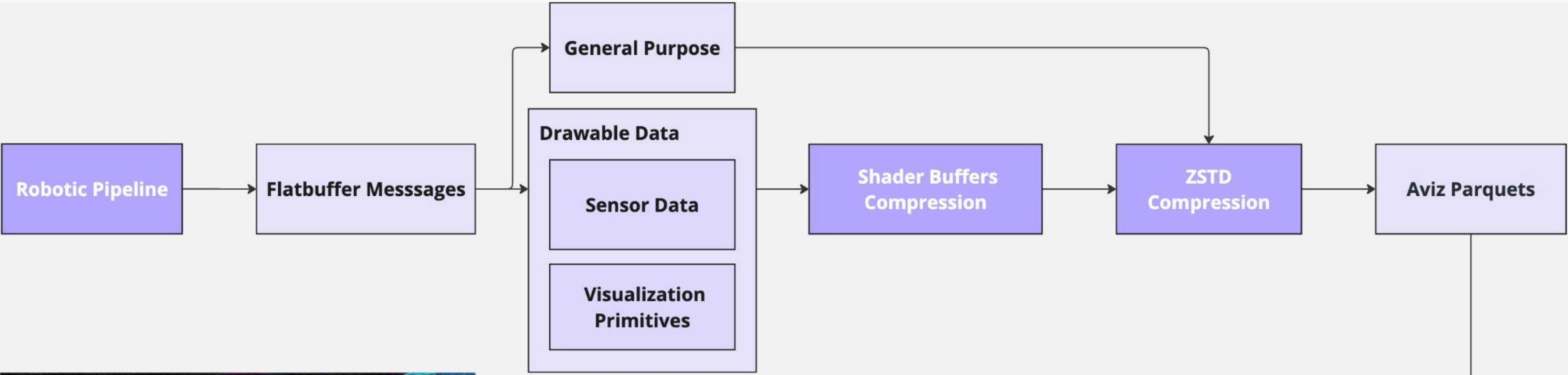
- Huge data volume
- Low latency
- High throughput
- High performance

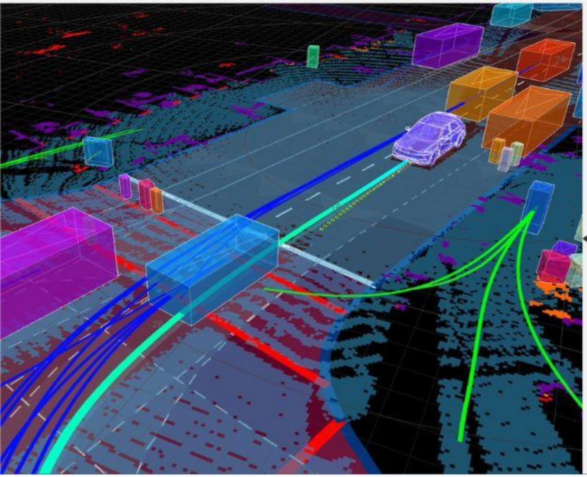


localization/protocol/localization_topic_1 av.lo…

```
▸ __metadata__ {generationId: 948…
▸ angular_velocity {...}
▸ global_geodetic_coordinates {.…
  gravity_pitch 0.014415103281973228
  gravity_pitch_sd 0.001017795746335
  gravity_roll -0.036411078026196794
  gravity_roll_sd 0.0005922359953744
  is_staying false
  jump_counter 13
▸ last_jump_ts {nsec: 13834000000…
▸ linear_acceleration {...}
▸ linear_velocity {...}
▸ localization_ts {nsec: 18993200…
▸ odometric_pose_2d {...}
▸ odometric_pose_3d {...}
▾ pose {}
  ▸ orientation_covariance_matrix …
  ▾ pose {}
    ▾ orientation {}
        w 0.6043732245449845
        x -0.011450898665919971
        y 0.021494731967576407
        z -0.7963289890933788
    ▸ position {...}
  ▸ position_covariance {...}
▸ reference_point {...}
```

Read Optimization





Draw Calls Reduction

- Struct of vectors instead of vector of structs
- Concatenate primitives attributes
- Merge topics draw calls further through 1D texture mask

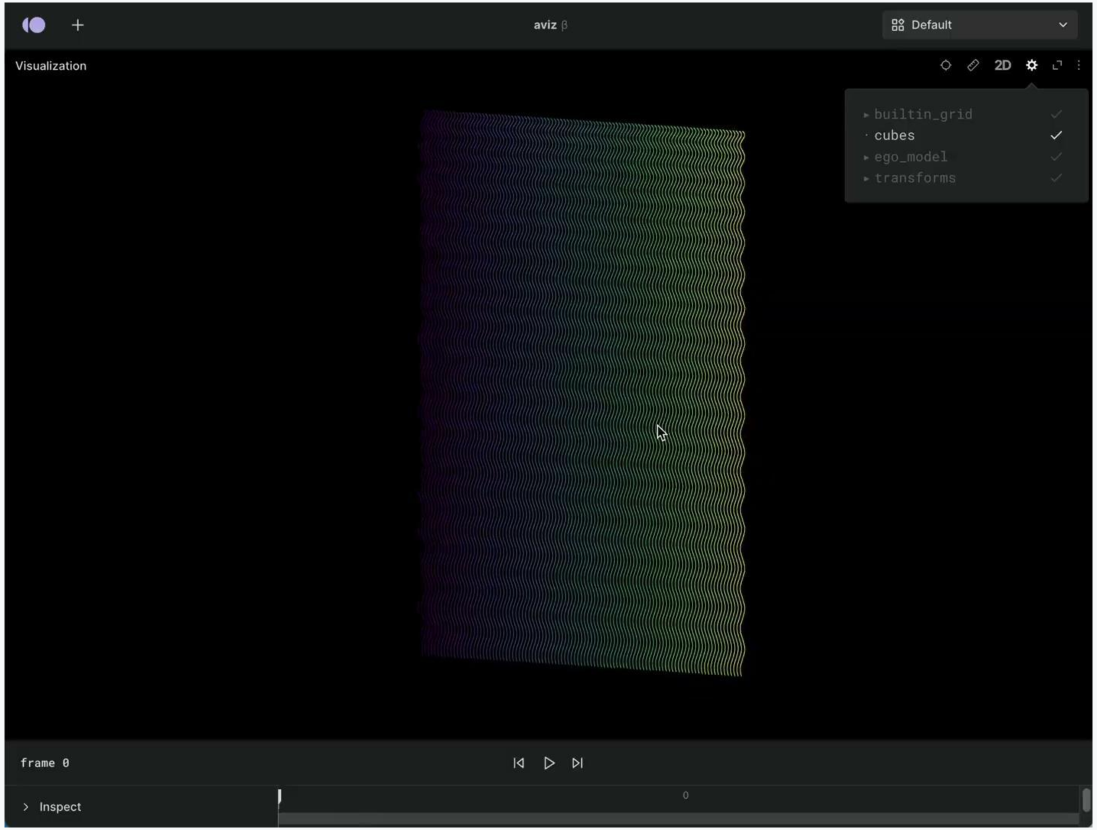
```
// many draw calls - expensive
// vector of structs
for (const auto& cube : cubes) {
    BufferDataToGPU(cube.x);
    BufferDataToGPU(cube.y);
    BufferDataToGPU(cube.z);
    DrawCall();
}

// single draw call - efficient
// struct of vectors
auto buffer = Buffer{
    .xs = Concatenate(cubes, &Cube::x),
    .ys = Concatenate(cubes, &Cube::y),
    .zs = Concatenate(cubes, &Cube::z)
};
BufferDataToGPU(buffer.xs);
BufferDataToGPU(buffer.ys);
BufferDataToGPU(buffer.zs);
DrawCall();
```

buffers pseudocode

Read Optimization

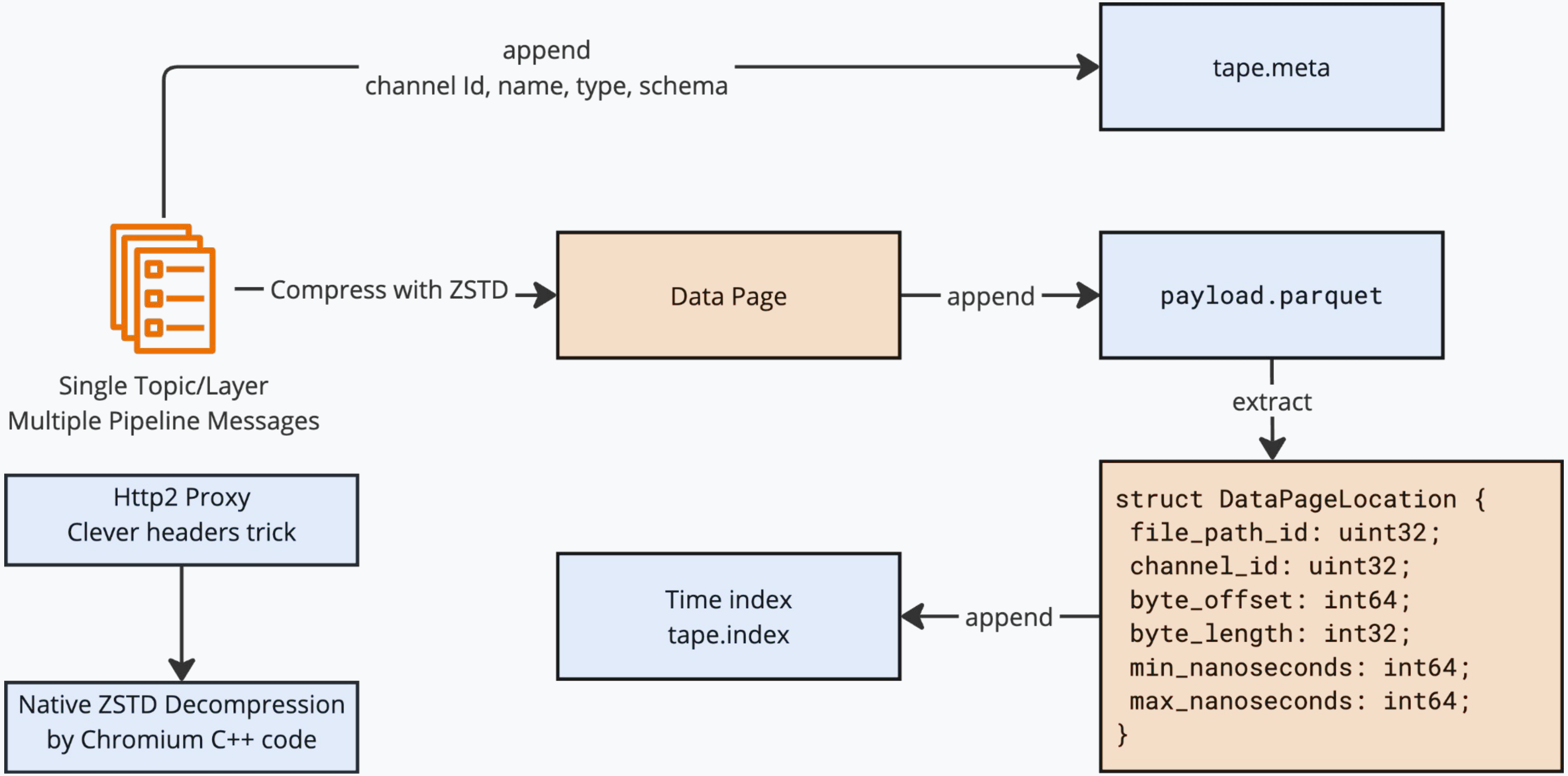
1 000 000 cubes - 60+ fps



Buffer Size Reduction

- Center all attributes around primitives center
- Quantize the attributes, for e.g. to int16_t or int8_t
- Collapse close data into the constant
- Struct of vector leads to better zstd compression
- Compressed data takes **~6.8x** less storage than ROS

Aviz Parquet/Tape Format



```
struct DataPageLocation {
 file_path_id: uint32;
 channel_id: uint32;
 byte_offset: int64;
 byte_length: int32;
 min_nanoseconds: int64;
 max_nanoseconds: int64;
}
```

03
Write Optimization

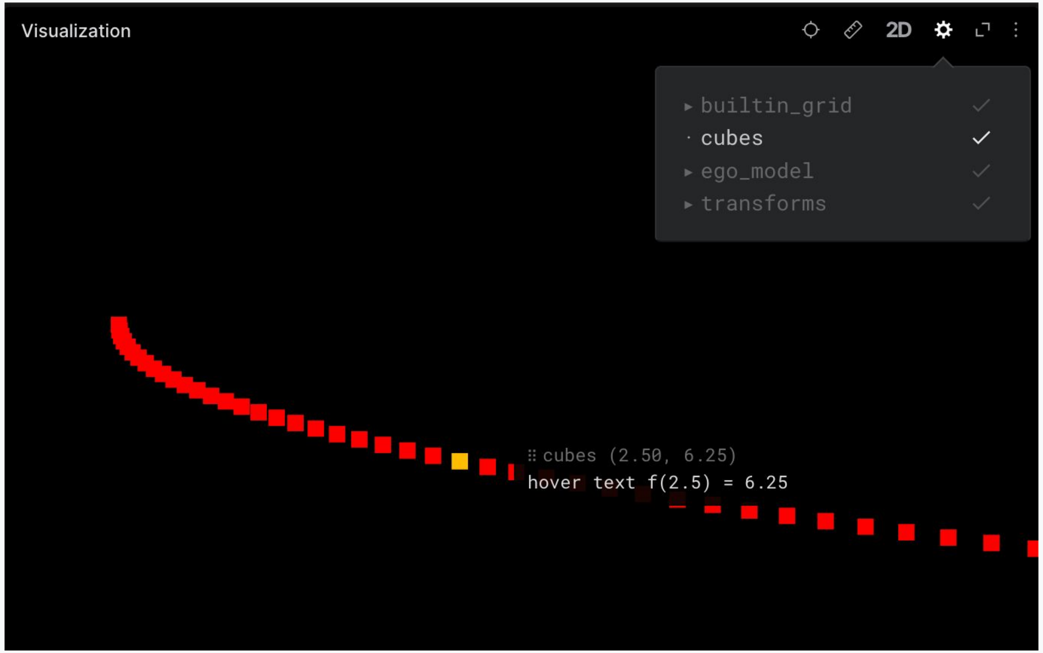
Old way of adding visualization

- Declare visualization topic
- Drill it to the place you want to draw
- Draw your primitives
- Update ~8 places to record it
- Debug your issues and discover that you don’t need these primitives anymore

When team lead asks you to *“add just one more visualization topic”*

Debug Visualization == Debug printing in 3d

```
AV_DRAW("/cubes", []() -> Markers {
  return {
    Cube{
      .position = {0, 0, 0},
      .color=palette::kRed,
      .text="f(0) = 0"
    },
    Cube{
      .position = {2, 4, 0},
      .color=palette::kRed,
      .text="f(2) = 4"
    },
    // …
  };
}());
```



```
cubes (2.50, 6.25)
hover text f(2.5) = 6.25
```

Minimal set of primitives

LLVM kinda approach

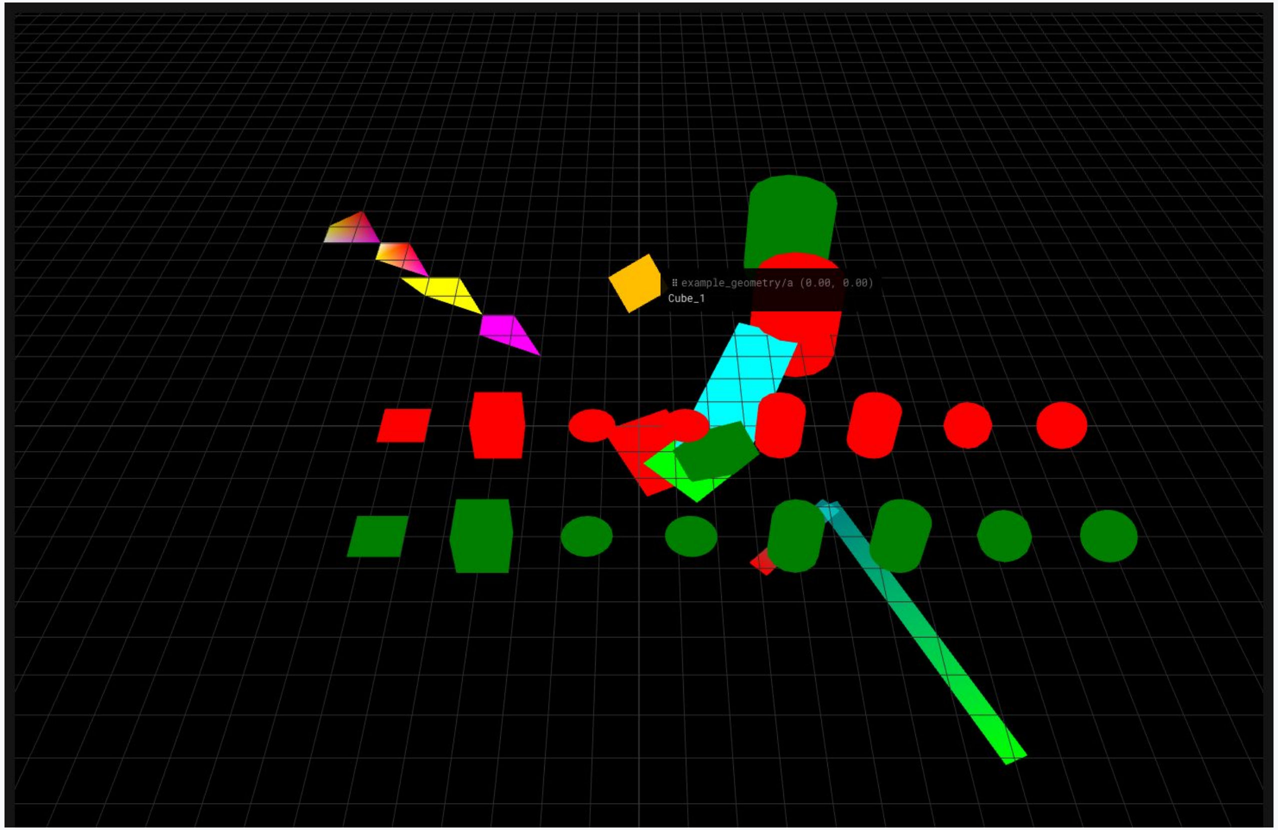
A. API can create composite shapes like occupancy grids.
B. Has low level internal geometry format. Draws few primitives, but can render lots of them.

```
// merge into one to reduce draw calls
Markers{
    LineList{
        .points = {{1, 0, 0}, {5, 0, 0}},
        .palette = palette::kRed,
        .sublayer  = “First LineList”

    },
    LineList{
        .points = {{10, 0, 0}, {15, 0, 0}},
        .palette = palette::kRed,
       .sublayer = “Second LineList”
    }
};
```

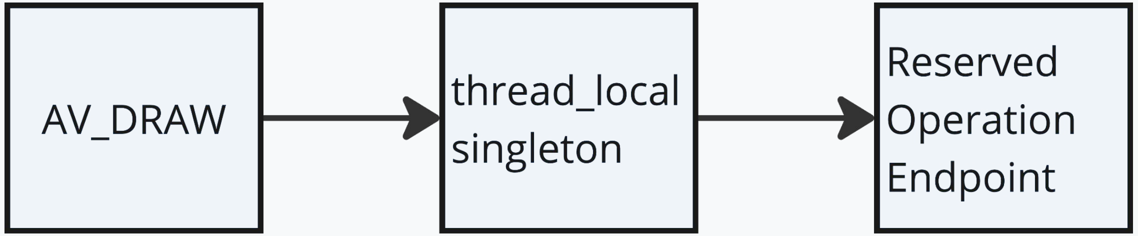
```
"payload": {
  "drawables_type": ["LineDrawable"],
  "drawables": [{
      "type": "LineList",
      "frame_id": 2,
      "pos_xy": {
        "array_type": "I16Array",
        "array": {
          "data": [-1166, 0, -500, 0, 333, 0, 1166, 0]
        },
        "scale": 0.006000000052154064,
        "origin": [8.0, 0.0],
        "count": 8
      }
}]
```

Minimal set of primitives

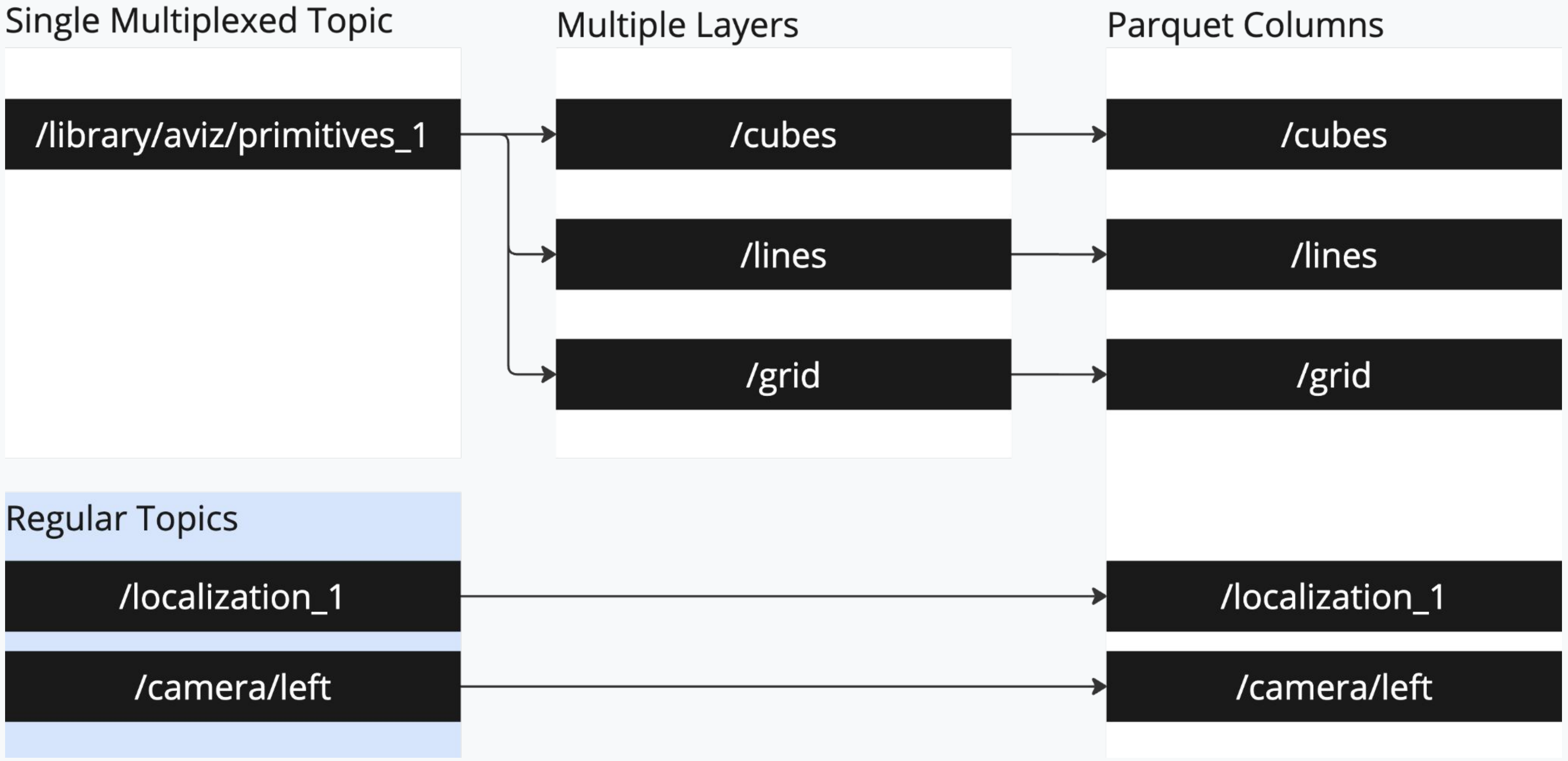


Integration with Pipeline Operation

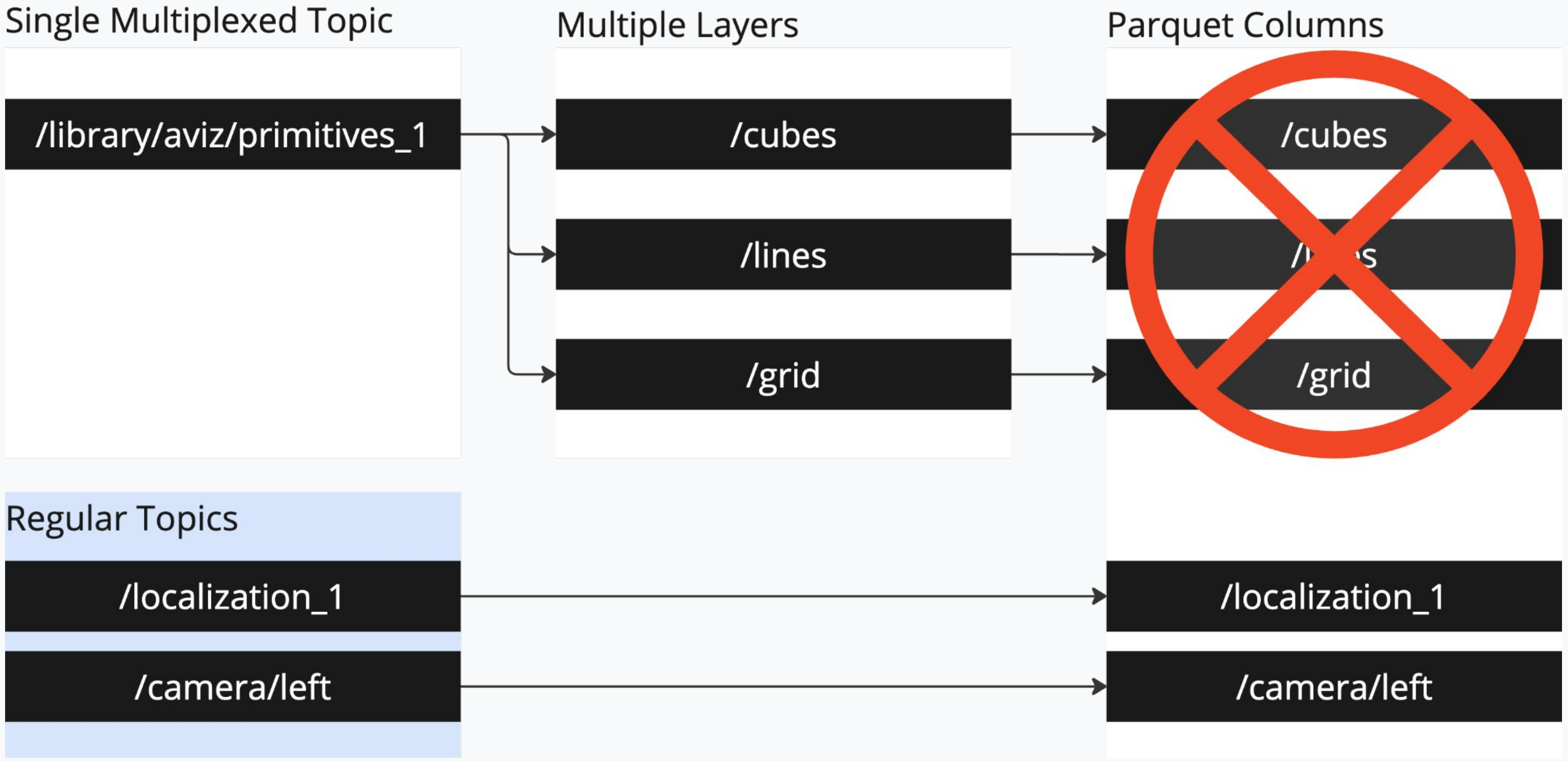
```
AV_DRAW("/cubes", []() -> Markers {
  return {
    Cube{
      .position = {0, 0, 0},
      .color=palette::kRed,
      .text="f(0) = 0"
    },
    Cube{
      .position = {2, 4, 0},
      .color=palette::kRed,
      .text="f(2) = 4"
    },
    // …
  };
}());
```



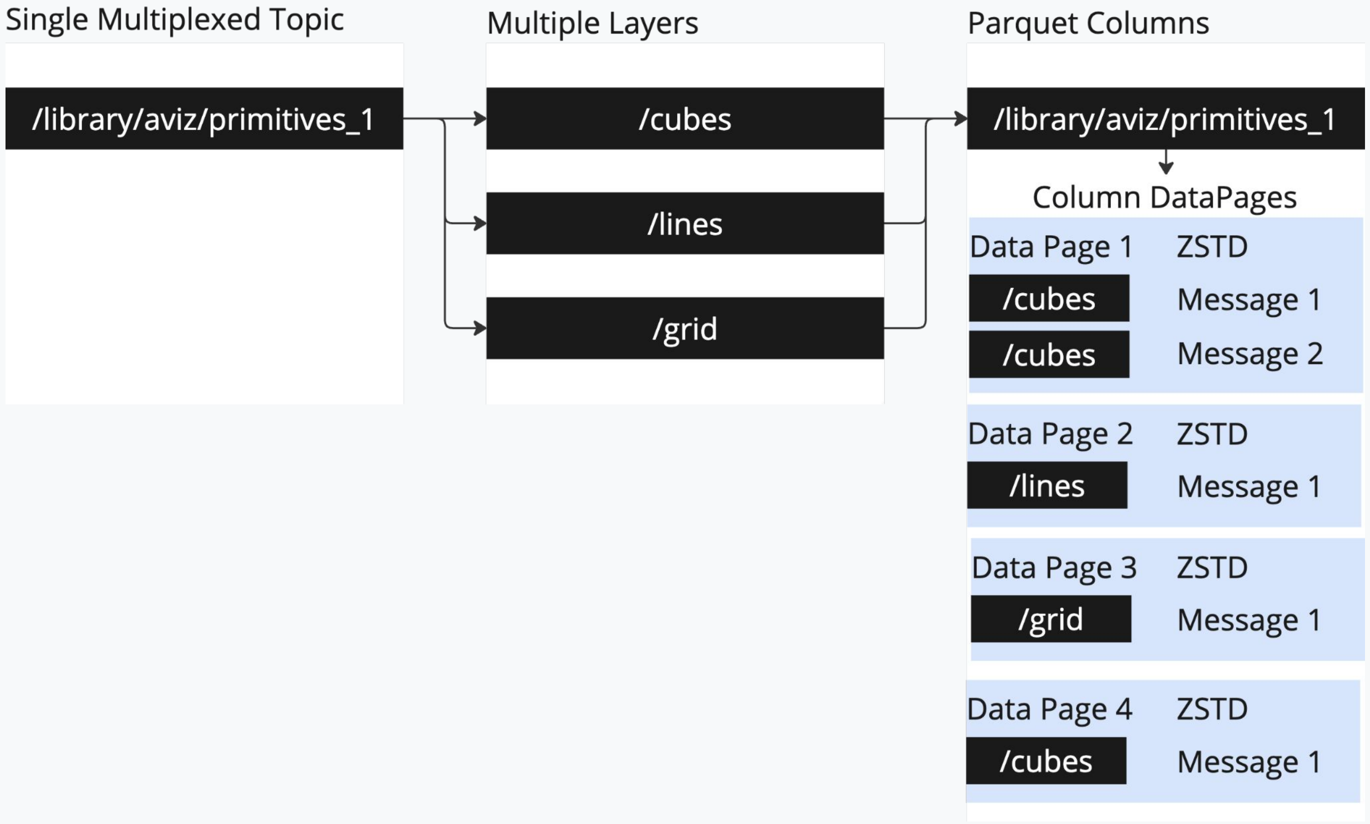
Tape Format + Parquet Version 0



Tape Format + Parquet Version 0



Tape Format + Parquet Version 1



Thank you!

Q&A