lamar.ethz.ch

LaMAR Tutorial
4. Practical guide & conclusion

Mihai Dusmanu

ETH zürich

ECCV TEL AVIV 2022

Microsoft

Overview

a) Benchmarking Pipeline
b) Customizing the Pipeline
c) Data format
d) Additional tools
e) Conclusion

a) Benchmarking Pipeline

Benchmarking pipeline – Mapping

feature extraction
global features
local features

Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

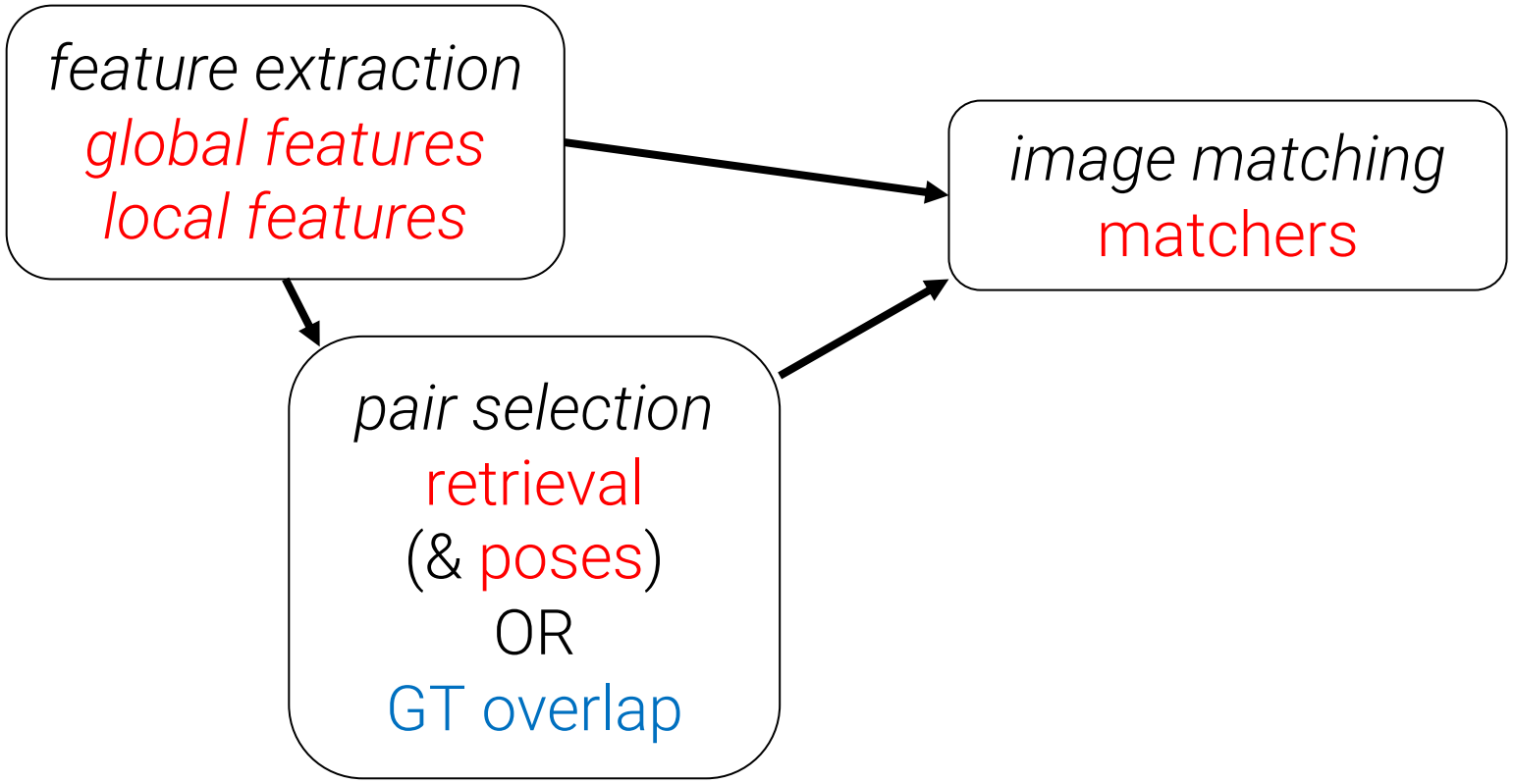
Benchmarking pipeline – Mapping

feature extraction
global features
local features

pair selection
retrieval
(& poses)
OR
GT overlap

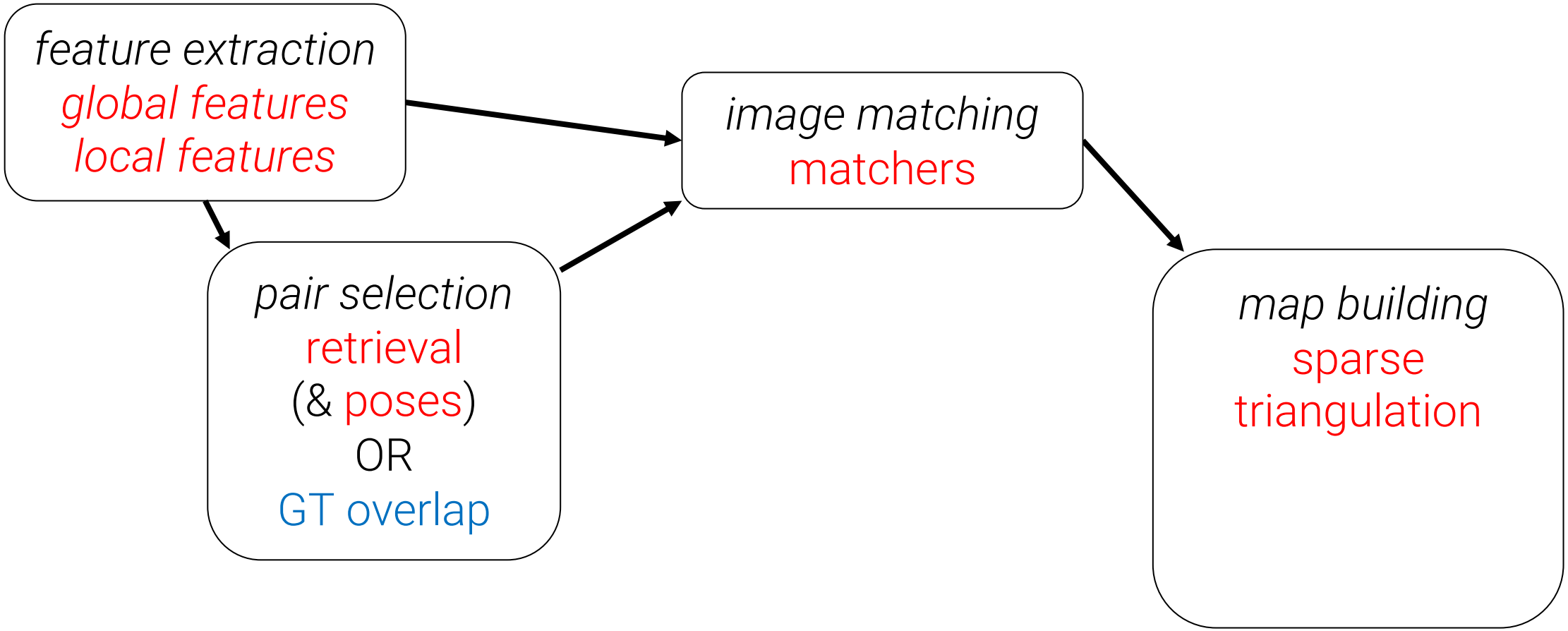
Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

Benchmarking pipeline – Mapping



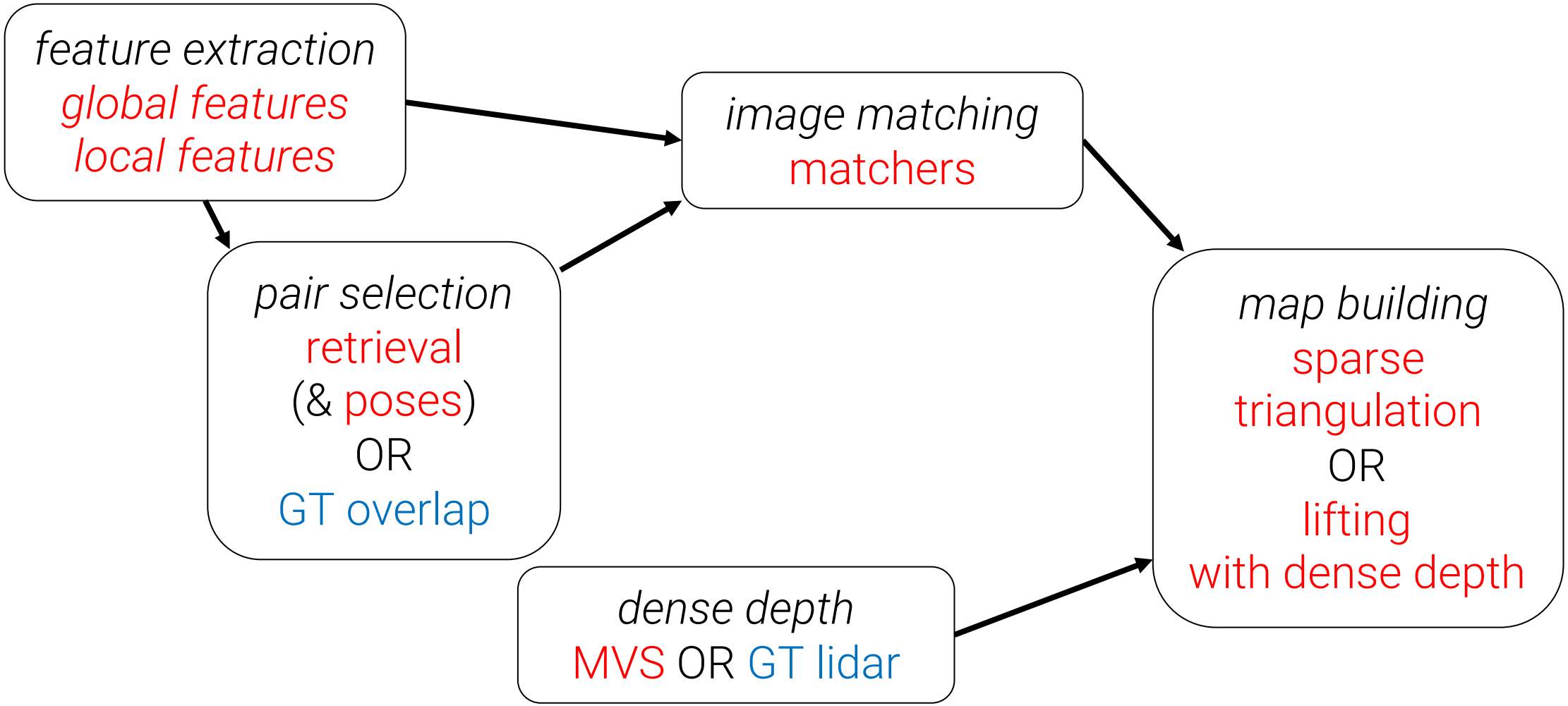
Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

Benchmarking pipeline – Mapping



Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

Benchmarking pipeline – Mapping



Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

Benchmarking pipeline – Localization

feature extraction
global features
local features

map building
sparse
triangulation
OR
lifting
with dense depth

Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

Benchmarking pipeline – Localization

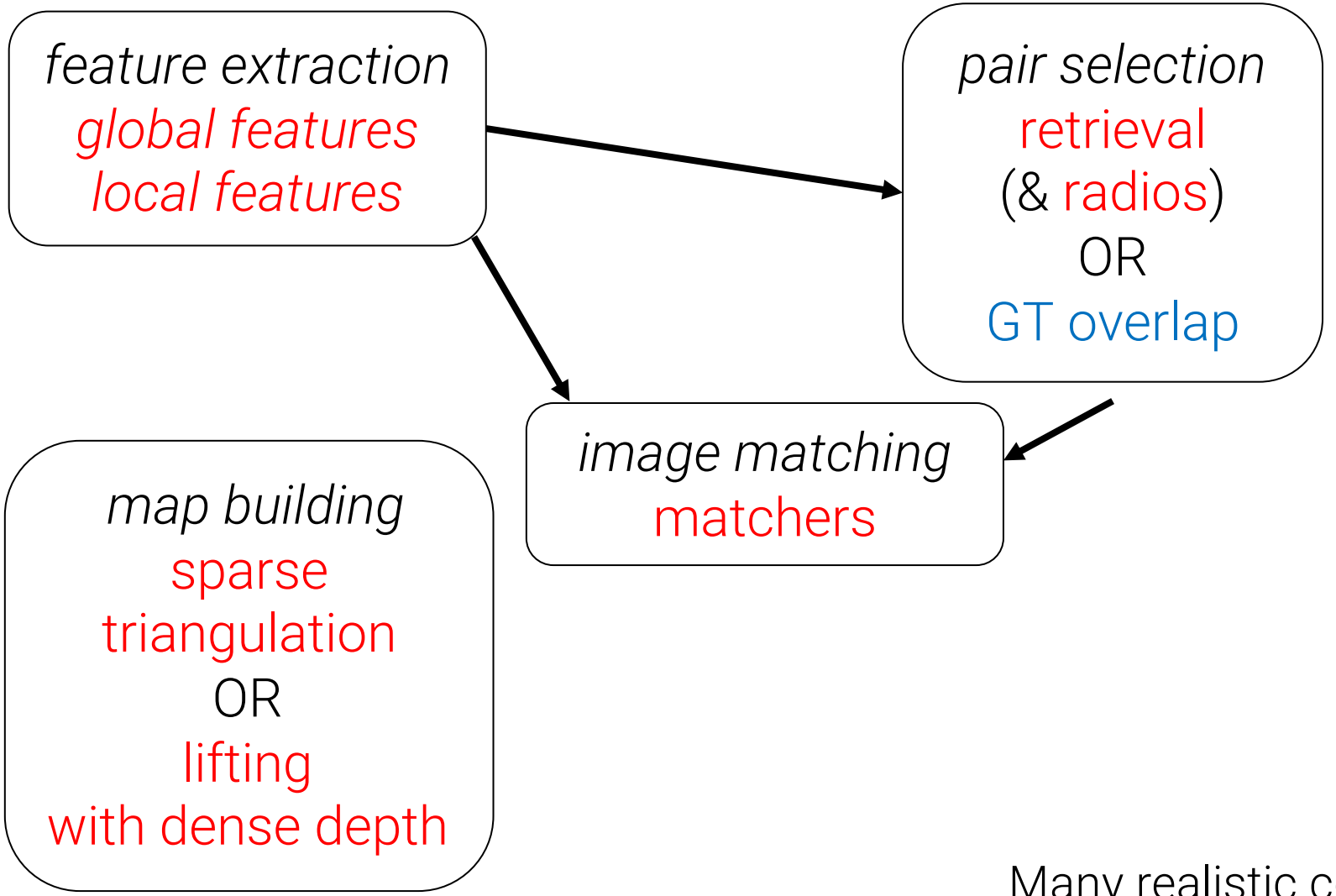
feature extraction
global features
local features

pair selection
retrieval
(& radios)
OR
GT overlap

map building
sparse
triangulation
OR
lifting
with dense depth

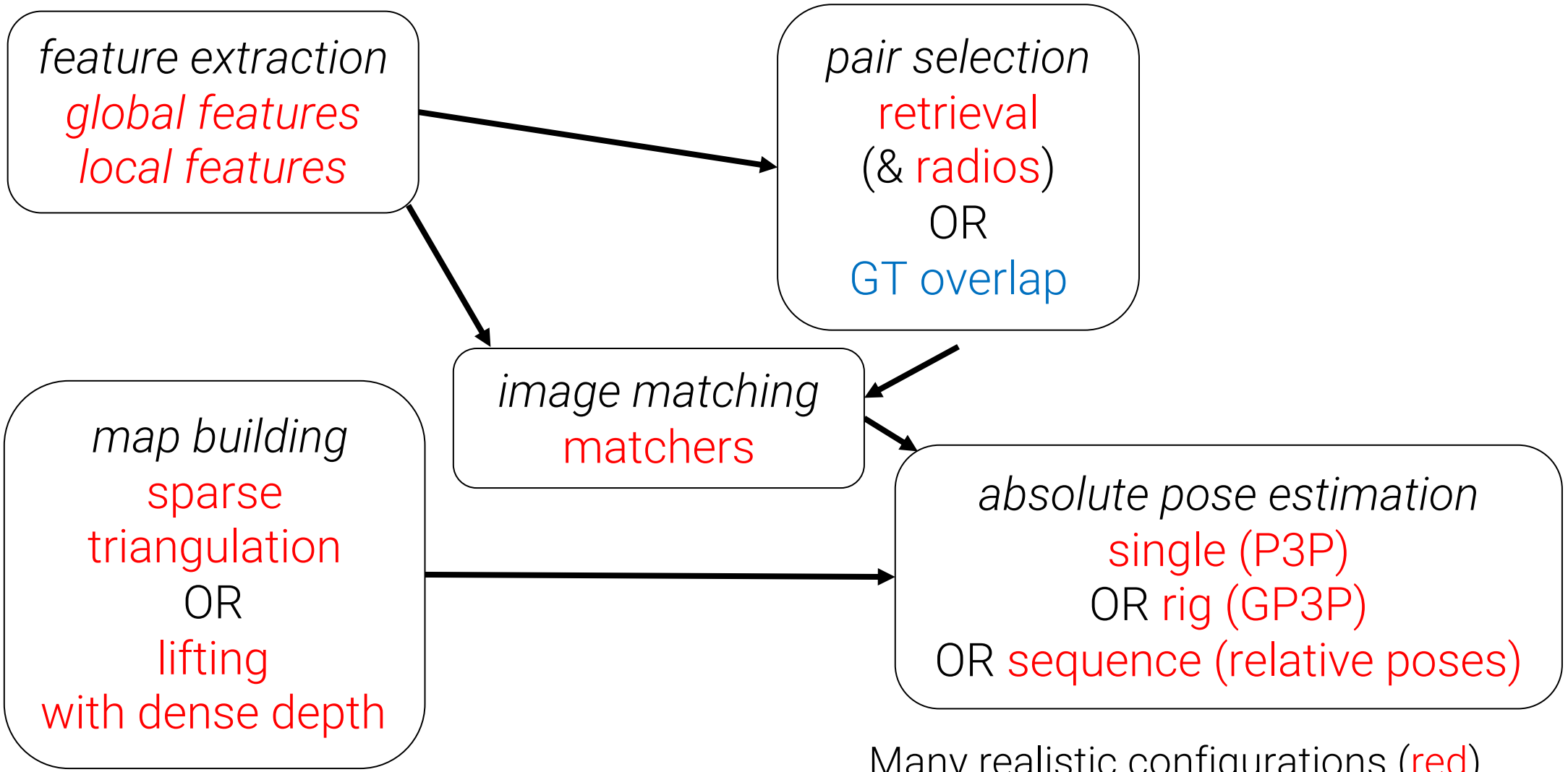
Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

Benchmarking pipeline – Localization



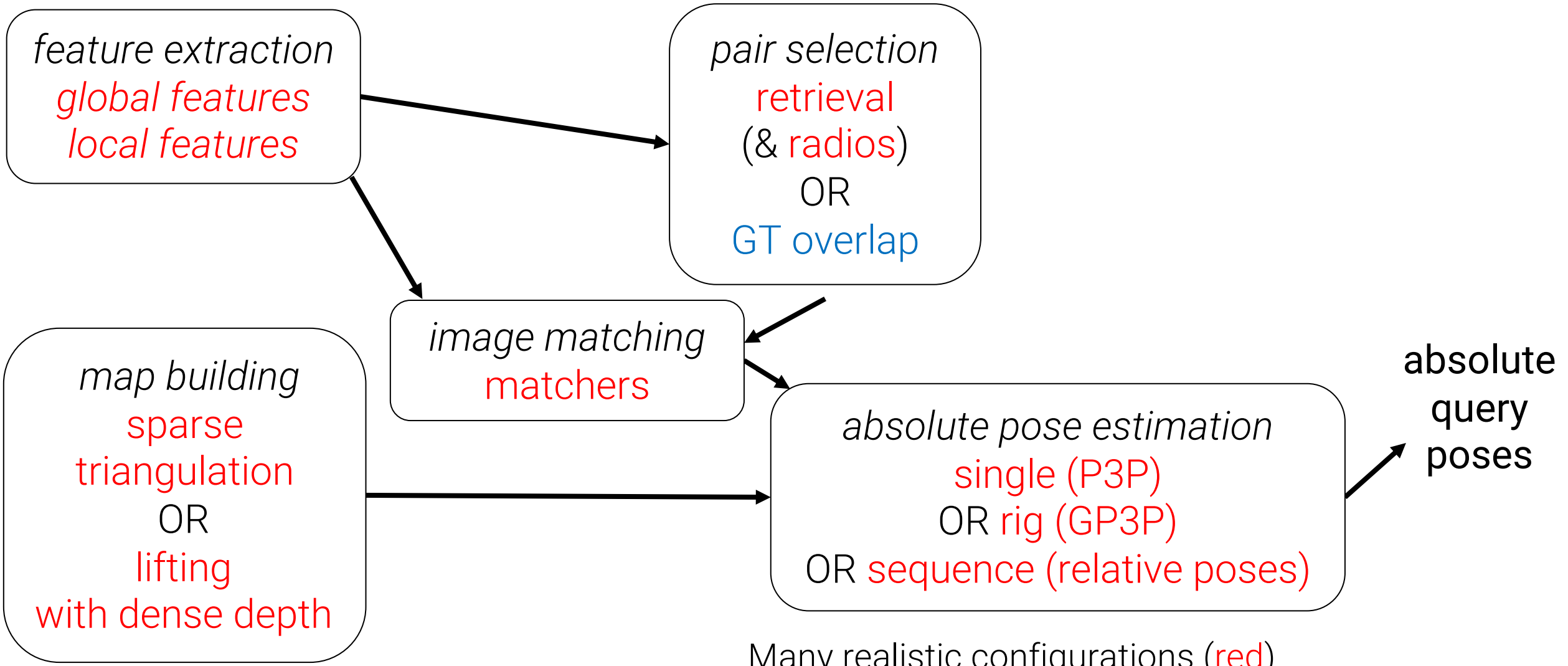
Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

Benchmarking pipeline – Localization



Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

Benchmarking pipeline – Localization

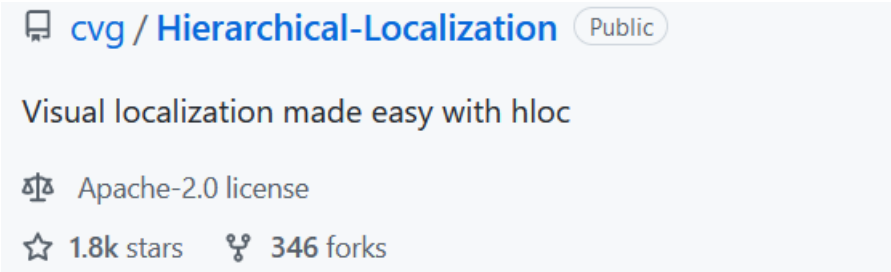


Many realistic configurations (red)
Swapable with GT quantities (blue) derived from lidar

b) Customizing the Pipeline

Feature Extraction / Matching

- Builds upon Hierarchical-Localization (hloc) repository



- Minimal changes required in the benchmarking repository

Adding a new feature extractor

- Add the method to hloc
  - New class: hloc/extractors/my-extractor.py
  - Inputs: image
  - Outputs:
    - Global features: global descriptor
    - Local features: keypoints, descriptors

```python
class MyGlobalFeature(BaseModel):
    default_conf = {
        'model_name': 'MyGlobalFeature',
    }
    required_inputs = ['image']

    def _init(self, conf):
        pass

    def _forward(self, data):
        image = data['image']

        desc = np.zeros(512)

        return {
            'global_descriptor': desc
        }
```

Adding a new feature extractor

- Add the method to hloc
  - New class: hloc/extractors/my-extractor.py
  - Inputs: image
  - Outputs:
    - Global features: global descriptor
    - Local features: keypoints, descriptors
- New config in the benchmark repo
  - tasks/feature_extraction.py

```python
class MyGlobalFeature(BaseModel):
    default_conf = {
        'model_name': 'MyGlobalFeature',
    }
    required_inputs = ['image']

    def _init(self, conf):
        pass

    def _forward(self, data):
        image = data['image']

        desc = np.zeros(512)

        return {
            'global_descriptor': desc
        }
```

```python
class RetrievalFeatureExtraction(FeatureExtraction):
    methods = {
        'netvlad': {
            'name': 'netvlad',
            'hloc': {
                'model': {'name': 'netvlad'},
                'preprocessing': {'resize_max': 640},
            },
        },
        'my-global-feature': {
            'name': 'my-global-feature',
            'hloc': {
                'model': {'name': 'MyGlobalFeature'},
                'preprocessing': {'resize_max': 640},
            },
        },
    }
```

Adding a new feature matcher

- Add the method to hloc
  - New class: hloc/matchers/my-extractor.py
  - Inputs: keypoints & descriptors for two images
  - Outputs: matches

Adding a new feature matcher

- Add the method to hloc
  - New class: hloc/matchers/my-extractor.py
  - Inputs: keypoints & descriptors for two images
  - Outputs: matches

- New config in the benchmark repo
  - tasks/feature_matching.py

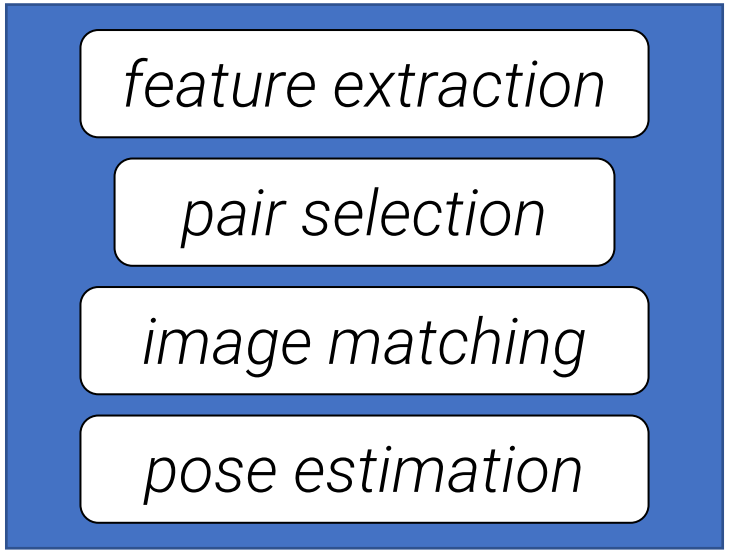
Mapping algorithm

- Create your own mapping class
- “run” method
  - Input: Capture object
  - Runs the reconstruction
- “get_points3D” method
  - Input:
    - image ID (key)
    - keypoint indices
  - Output:
    - valid indices
    - 3D coordinates, 3D point ids

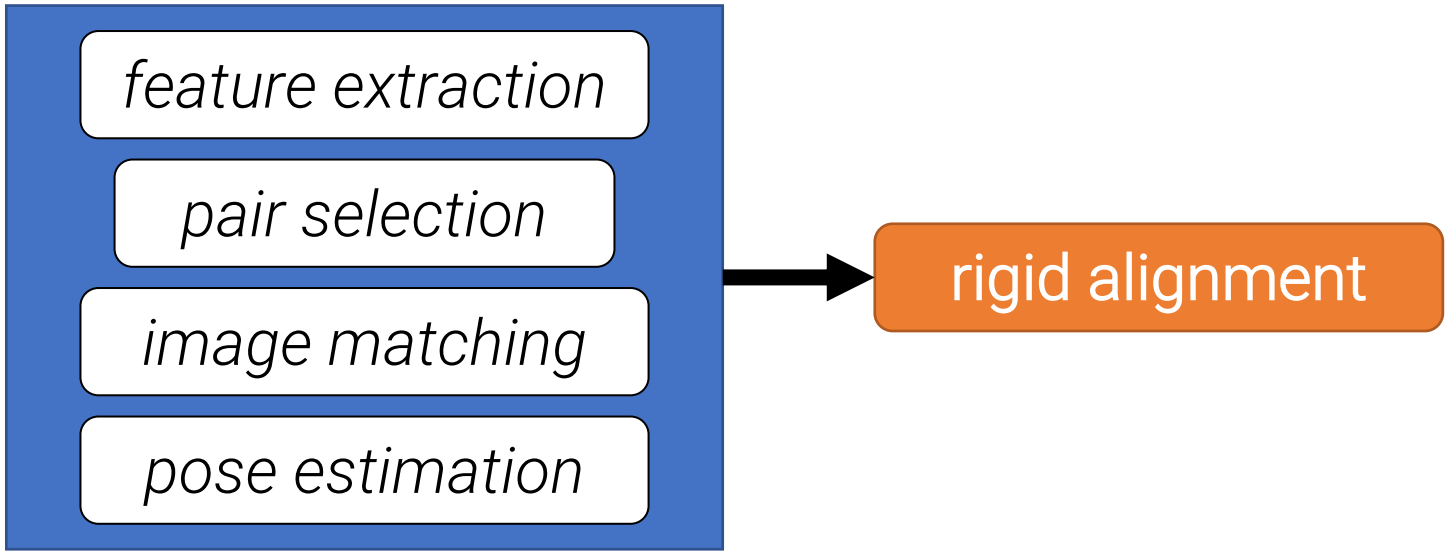
```
class Triangulation(Mapping):
    method = {
        'name': 'triangulation',
        # some COLMAP parameters and thresholds
    }

    def __init__(self, config, outputs, capture, session_id,···

    def run(self, capture):···

    def get_points3D(self, key, point2D_indices):···
```

```
def get_points3D(self, key, point2D_indices):
    image = self.reconstruction.images[self.key2imageid[key]]
    valid = []
    xyz = []
    ids = []
    if len(image.points2D) > 0:
        for idx in point2D_indices:
            p = image.points2D[idx]
            valid.append(p.has_point3D())
            if valid[-1]:
                ids.append(p.point3D_id)
                xyz.append(self.reconstruction.points3D[ids[-1]].xyz)
    return np.array(valid, np.bool), xyz, ids
```

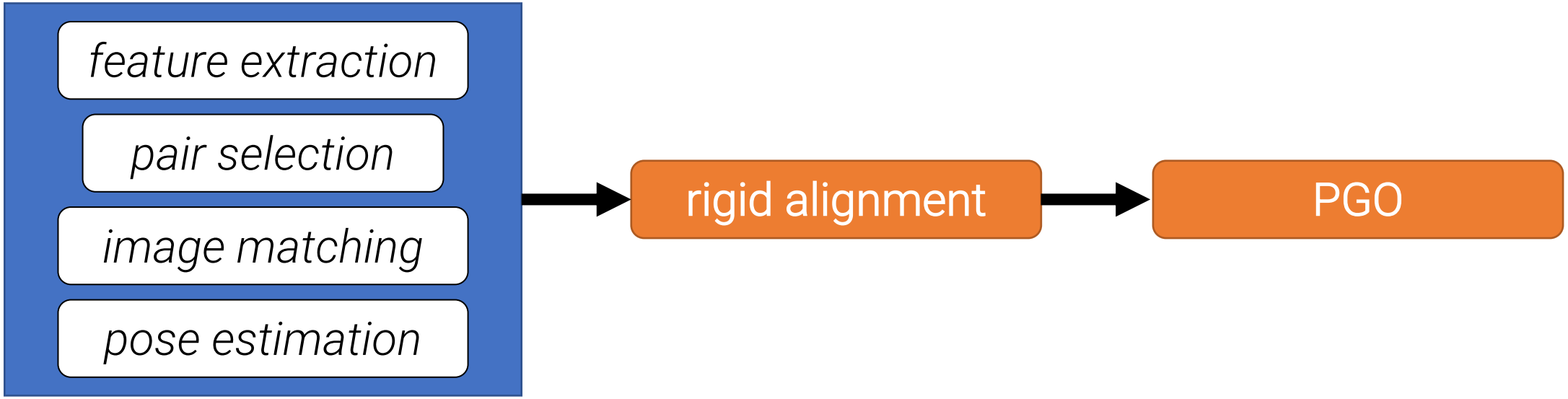
Sequence Baseline Overview



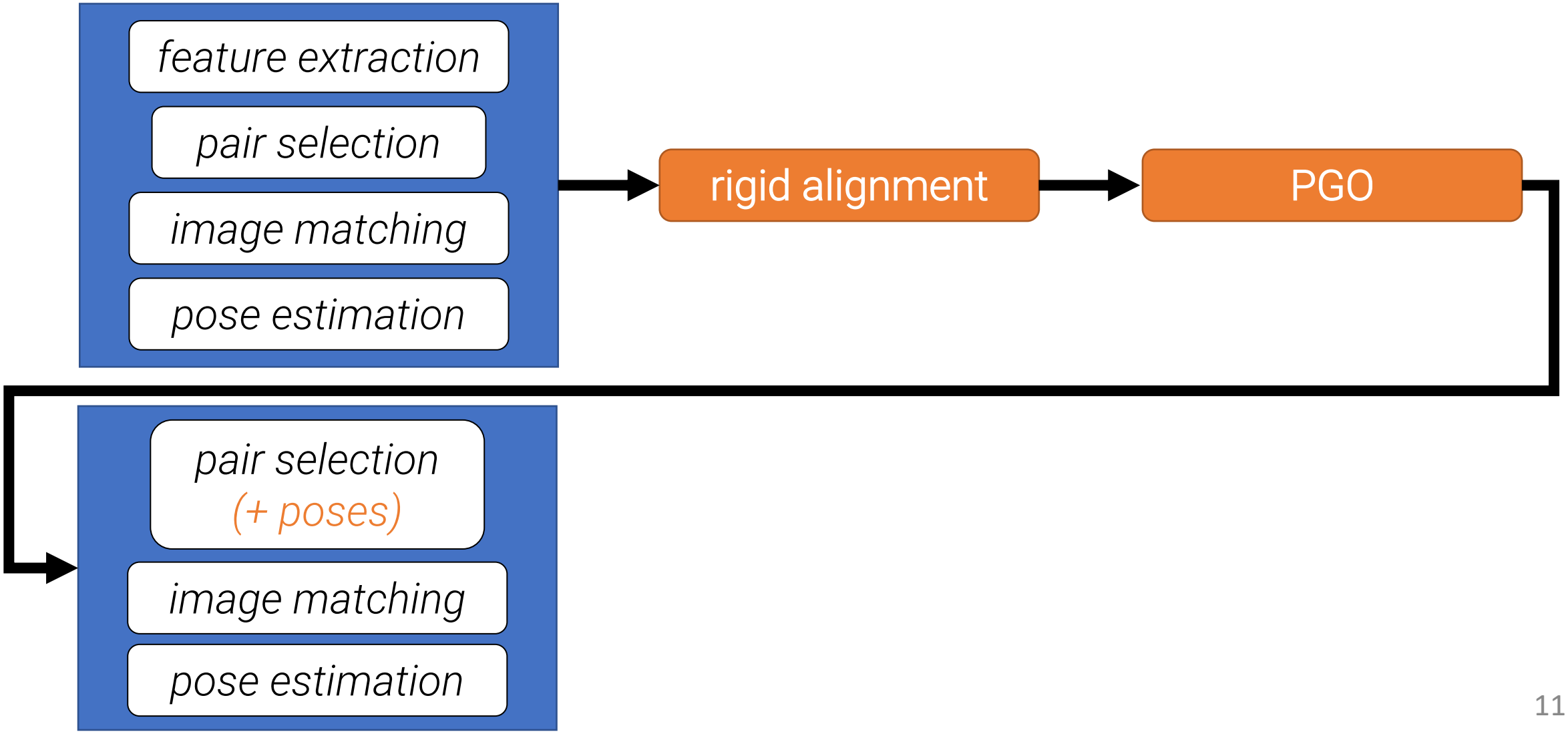
Sequence Baseline Overview



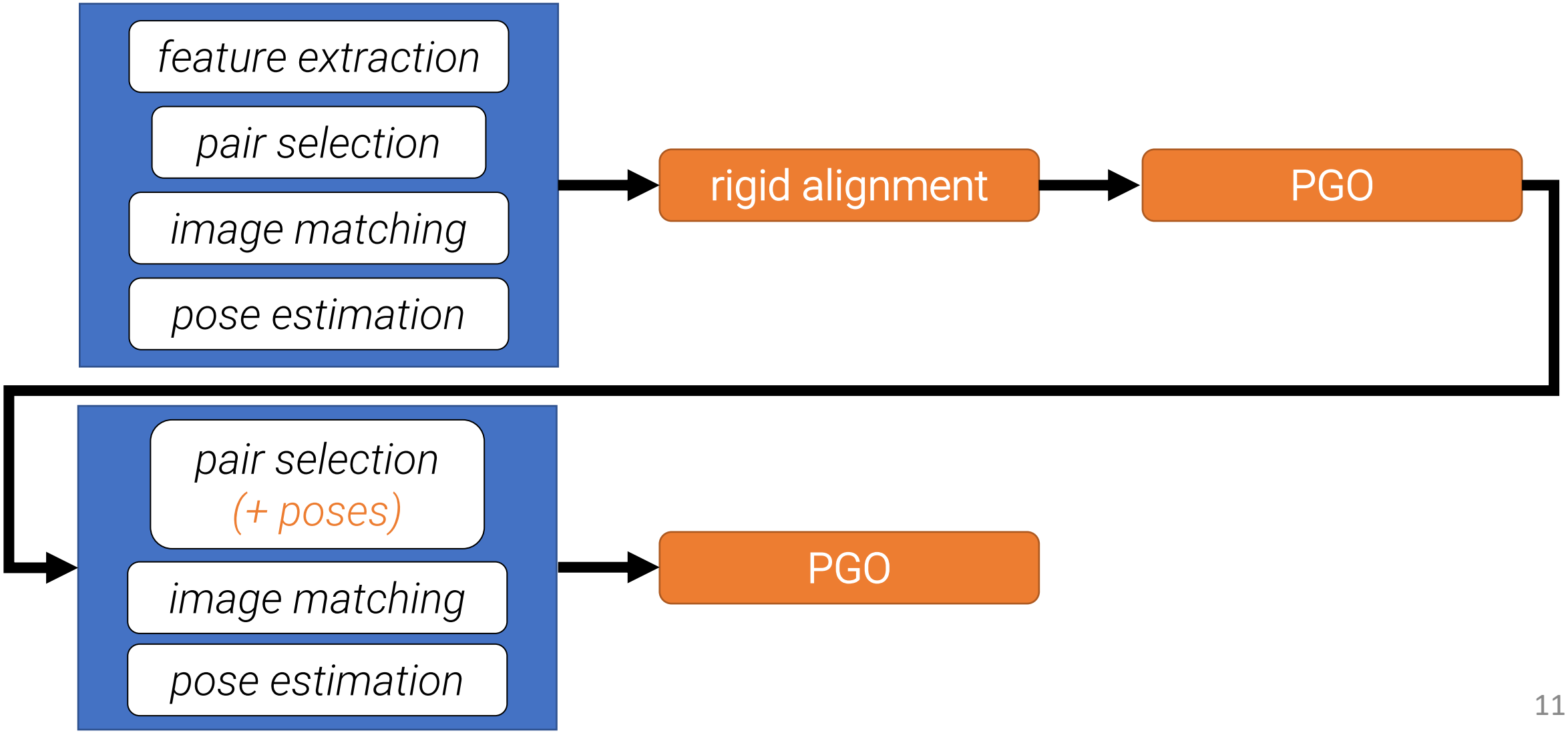
Sequence Baseline Overview



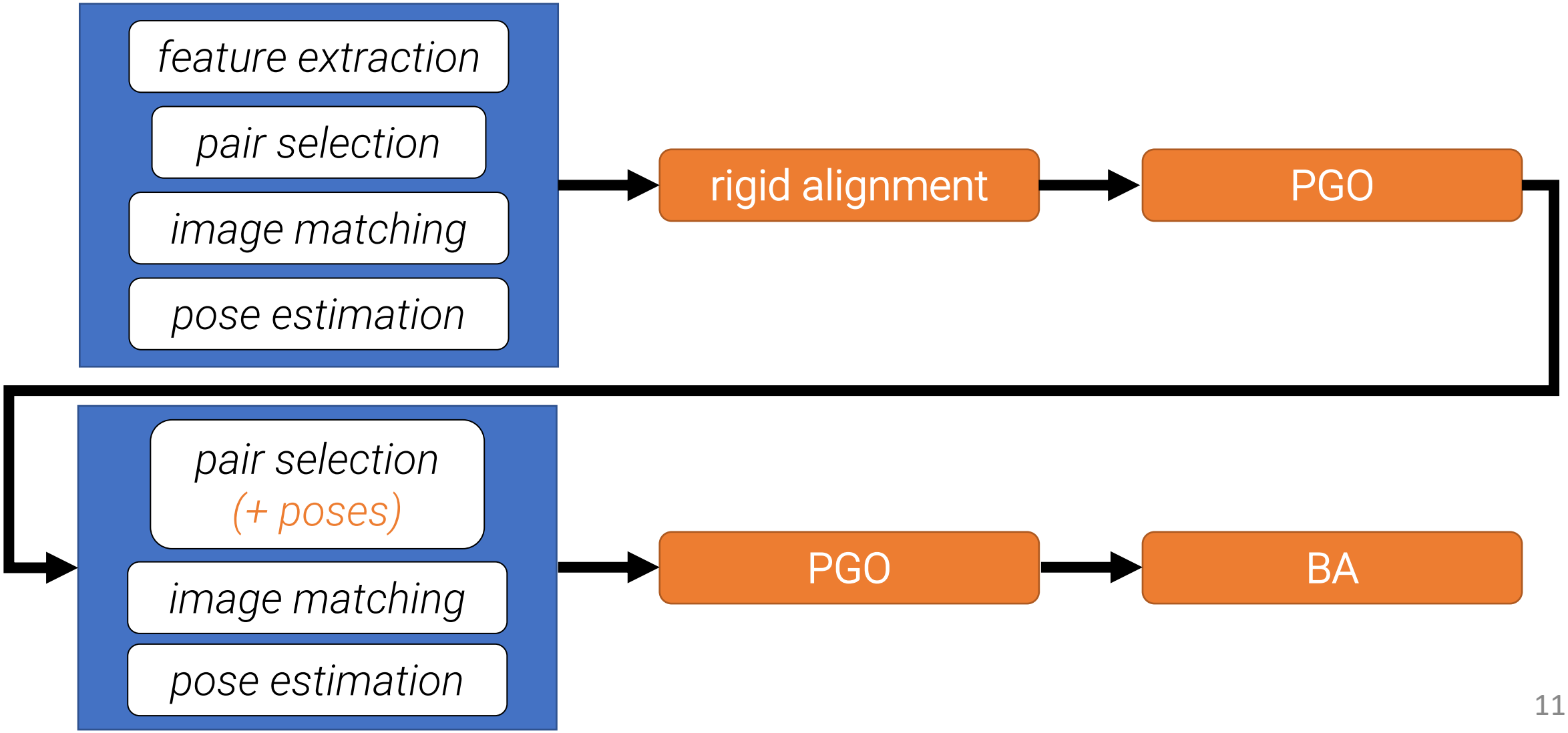
Sequence Baseline Overview



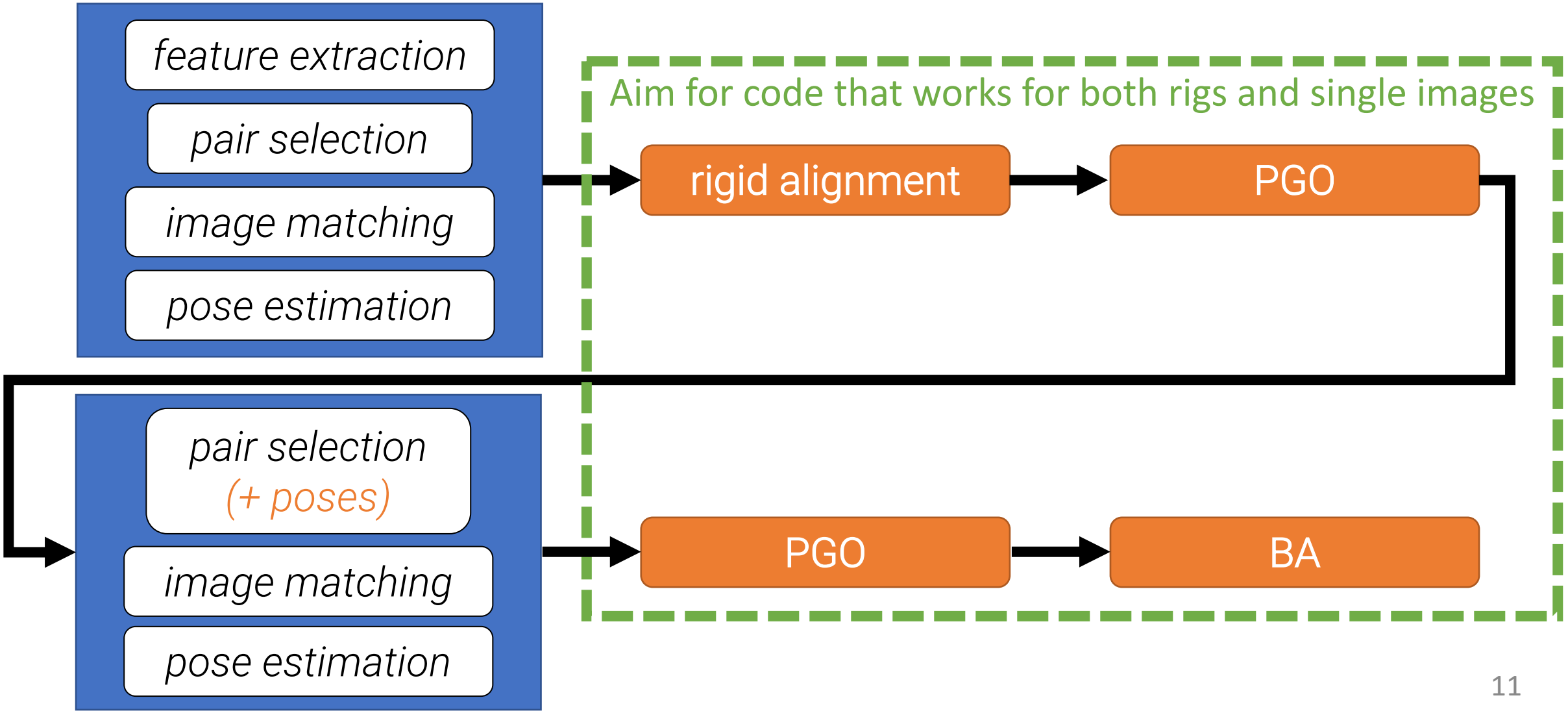
Sequence Baseline Overview



Sequence Baseline Overview



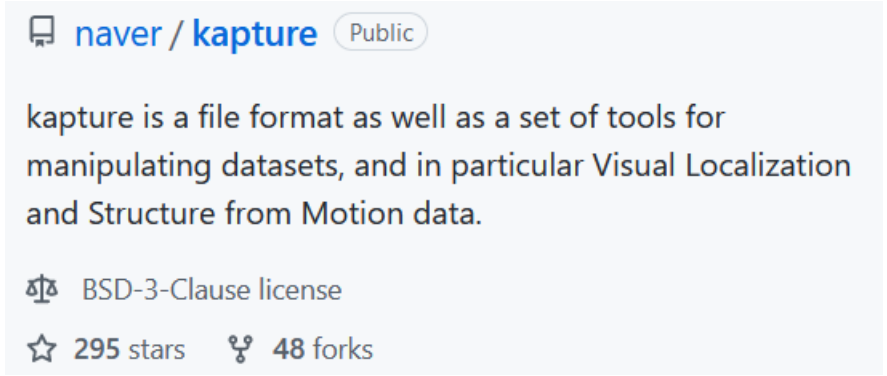
Sequence Baseline Overview



c) Data format

Existing data format

- *Kapture* format
  - Great format for Vis-Loc / SfM
  - Contains most common sensors
  - Support for features
  - Unified format for all datasets – great effort!
- Our requirements → new *Capture* format
  - Native multi-session datasets
  - Support for processed data (meshes, renderings, different trajectories)
  - Separated (raw / processed) data from features / reconstructions
- Very easy conversion between *Capture* and *Kapture*

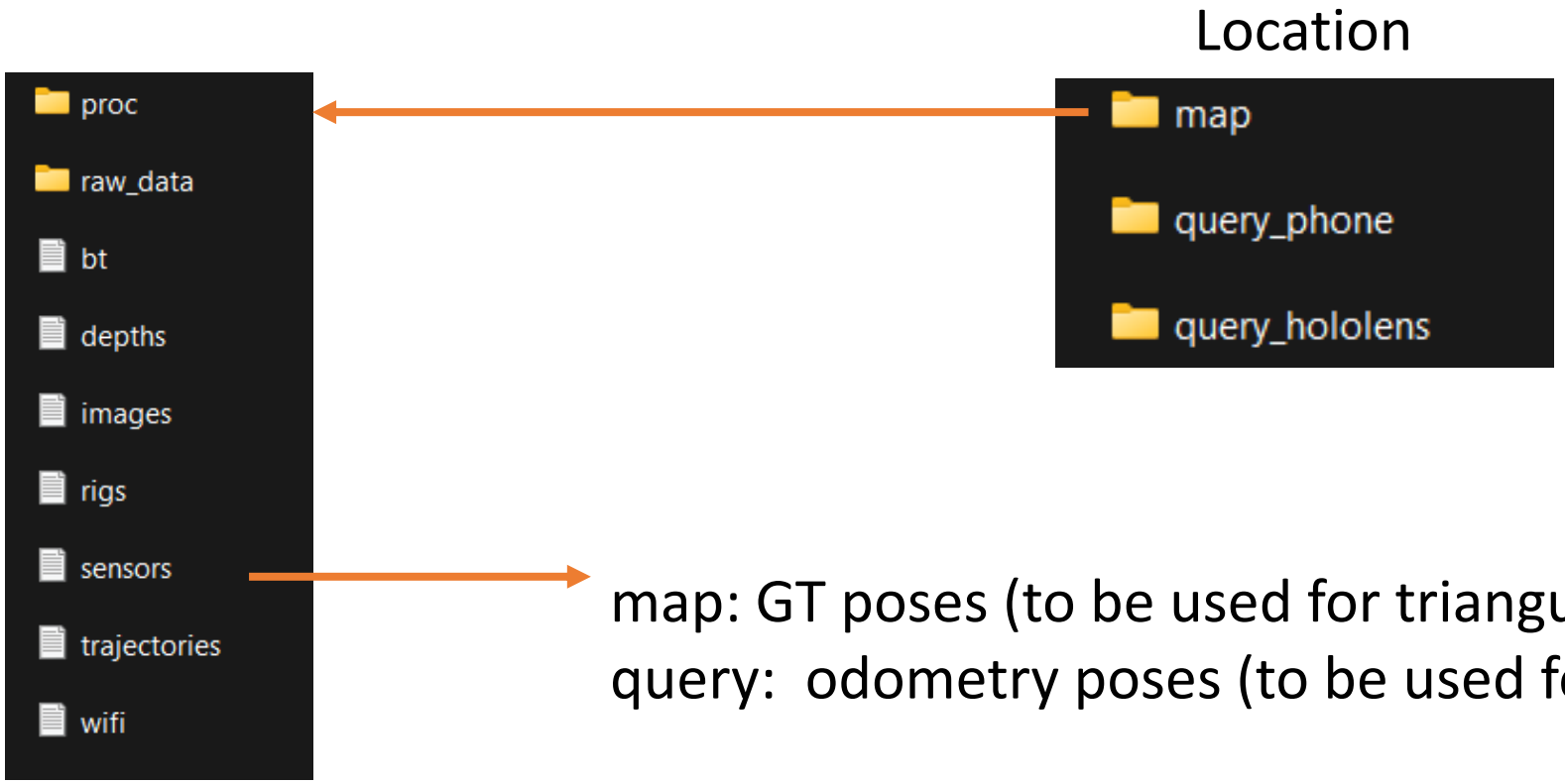


Overview of the capture format

- Images, depth maps, point-clouds
- Camera intrinsics, rigs
- Trajectories (from odometry)
- Radios (WiFi, Bluetooth)
- Coming soon: GPS, IMU, Gravity
- Natively multi-session
- Support for processed data

Benchmarking Data

- Keyframed data (images only)
  - 2.5FPS for database, 1FPS for query

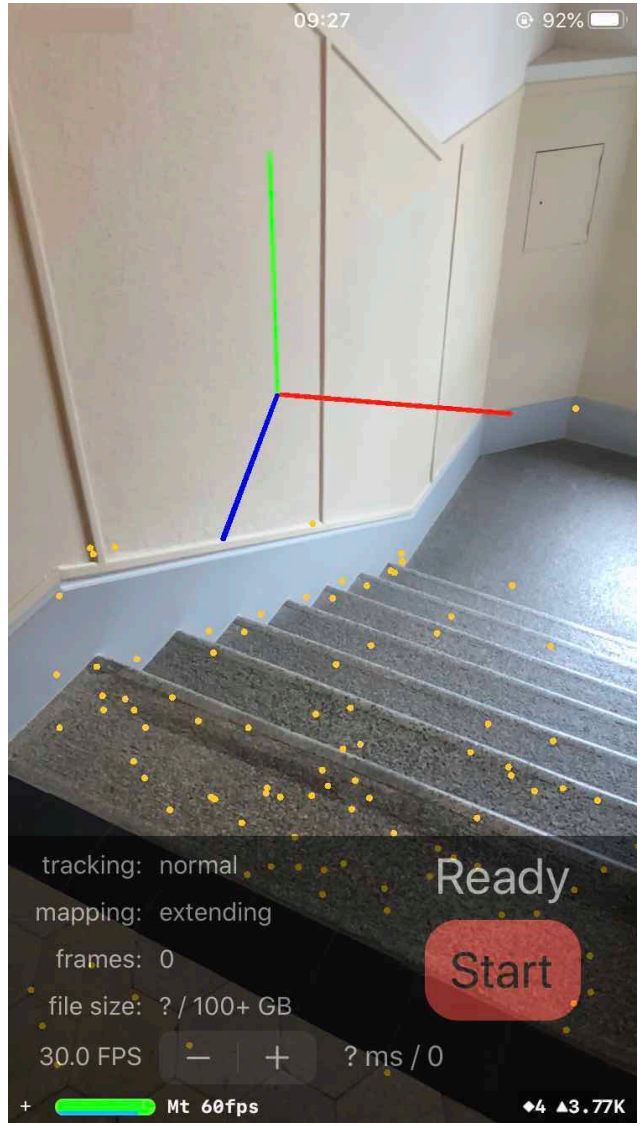


map: GT poses (to be used for triangulation)
query: odometry poses (to be used for seq. evaluation)

d) Additional tools

iOS Capture App

- iPhone / iPad capture app recording
  - Video stream (variable framerate)
  - Depth maps
  - ARKit poses & camera intrinsics
  - IMU (filtered & unfiltered)
  - GPS, Bluetooth (anonymized)



PyCOLMAP

colmap / pycolmap Public

Python bindings for COLMAP

BSD-3-Clause license

417 stars 65 forks

- Python bindings for COLMAP
  - Estimators (P3P, GP3P, Homography, Essential / Fundamental + LO-RANSAC)
  - Reconstruction object (images, cameras, points)
  - SIFT (VLFeat / GPUSIFT) feature extraction
  - Reconstruction, triangulation, MVS pipelines
- Contributions
  - Bindings for parts of COLMAP you're using!
  - Modularize each pipeline to be able to costumize
- Thanks to Philipp Lindenberger 

PyCeres

cvg / pyceres Public

Factor graphs with Ceres in Python

82 stars 11 forks

- Factor graphs backed by Ceres from Python
  - Relative pose constraints
  - Absolute pose constraints
  - Bundle cost
  - Floor-plan alignment cost
- Contributions
  - Gravity constraints
  - IMU with preintegration
  - GPS cost
- Thanks to Philipp Lindenberger 

pcdMeshing

cvg / pcdmeshing Public

Point cloud meshing with CGAL

Apache-2.0 license

12 stars 1 fork

- CGAL meshing for large dense point-clouds
  - Advancing Front algorithm

pcdMeshing

cvg / pcdmeshing Public

Point cloud meshing with CGAL

Apache-2.0 license

12 stars 1 fork

- CGAL meshing for large dense point-clouds
  - Advancing Front algorithm

pcdMeshing

cvg / pcdmeshing Public

Point cloud meshing with CGAL

Apache-2.0 license

12 stars 1 fork

- CGAL meshing for large dense point-clouds
  - Advancing Front algorithm
- Contributions
  - Free-space filtering with sensor positions
  - Simplification and UV texturing

RayBender

cvg / raybender Public

Fast CPU rendering in Python using the Intel® Embree backend

BSD-3-Clause license

26 stars 5 forks

- CPU ray tracing with Intel Embree from Python

```
scene = rb.create_scene()
geometry_id = rb.add_triangle_mesh(
    scene, vertices, triangles)

ray_origins, ray_directions = rbutils.compute_rays_for_simple_pinhole_camera(
    R, tvec, intrinsics)
geom_ids, bcoords = rb.ray_scene_intersection(
    scene, ray_origins, ray_directions)

tri_ids, bcoords, valid = rbutils.filter_intersections(
    geom_ids, bcoords)
rgb, depth = rbutils.interpolate_rgbd_from_geometry(
    triangles, vertices, vertex_colors,
    tri_ids, bcoords, valid,
    R, tvec, w, h)
```

RayBender

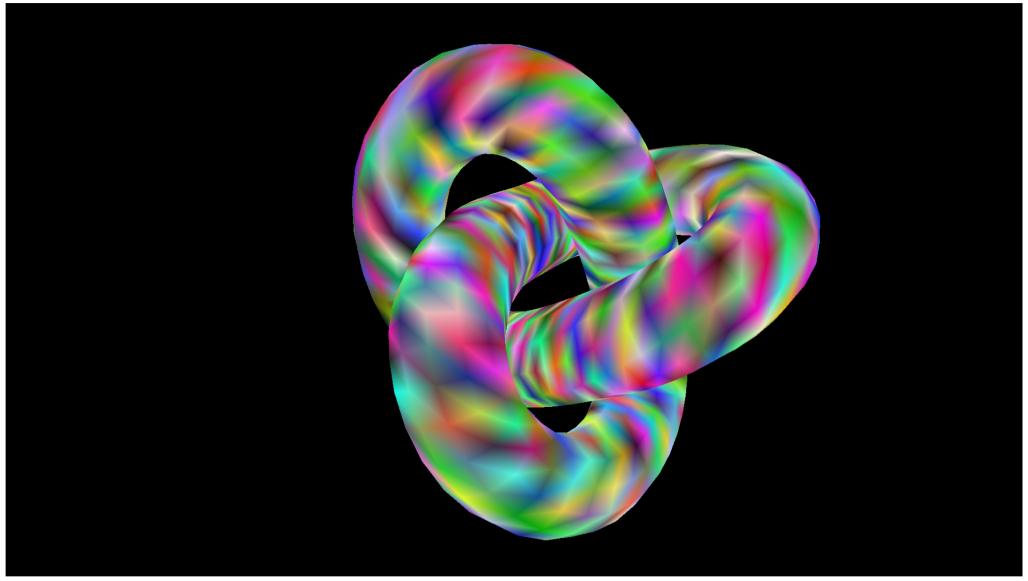
cvg / raybender Public

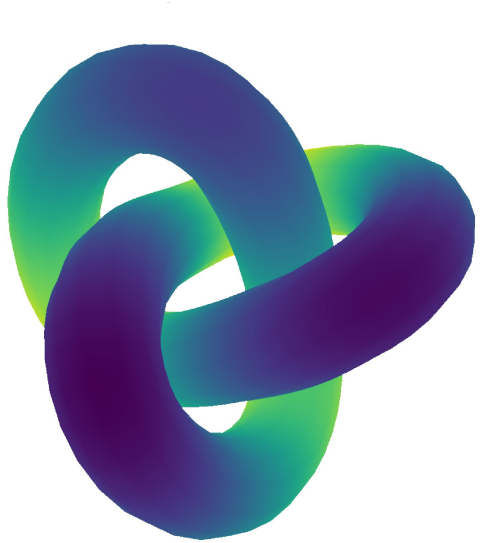
Fast CPU rendering in Python using the Intel® Embree backend

BSD-3-Clause license

26 stars 5 forks

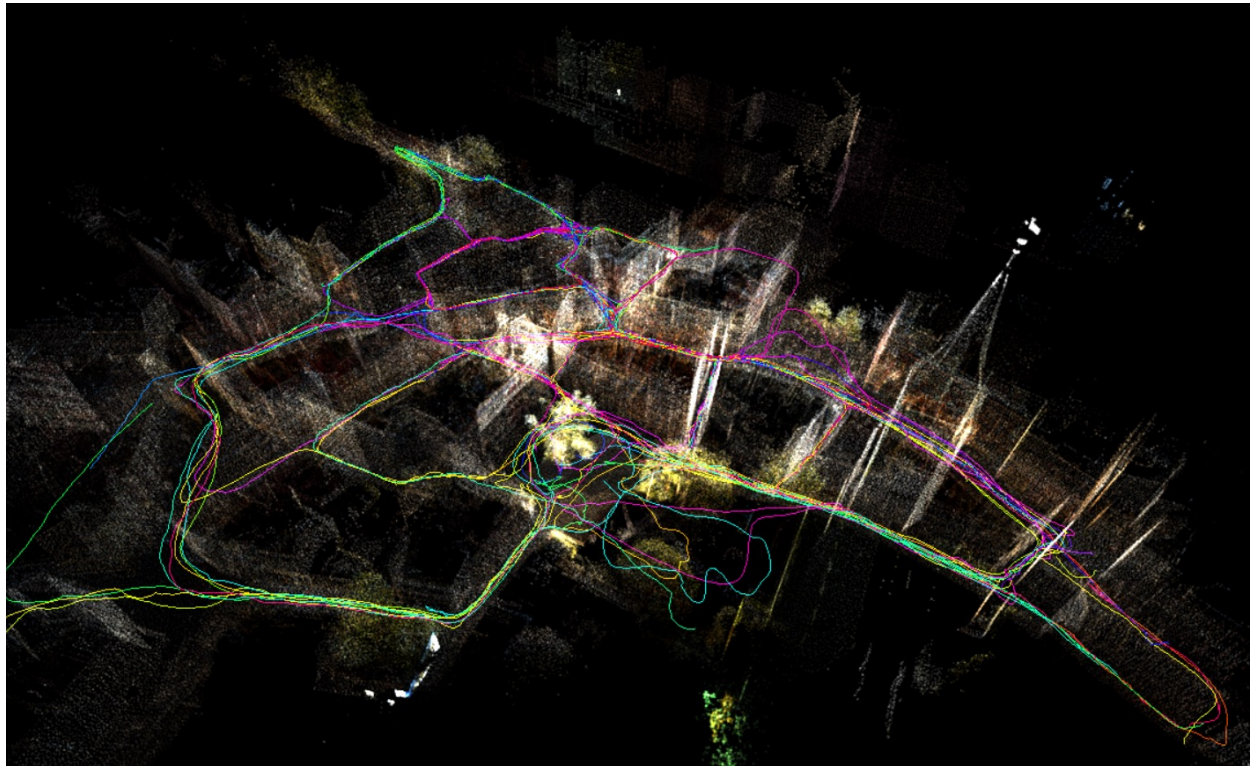
- CPU ray tracing with Intel Embree from Python
- Contributions
  - Support for multiple meshes in the same scene
  - Multi-ray single-operation tracing (AVX-512)





CVisG

- Browser visualizer (TypeScript + ThreeJS) supporting
  - Adaptive pointcloud rendering (PoTree)
  - Trajectories, frustums, images



e) Conclusions

Conclusions

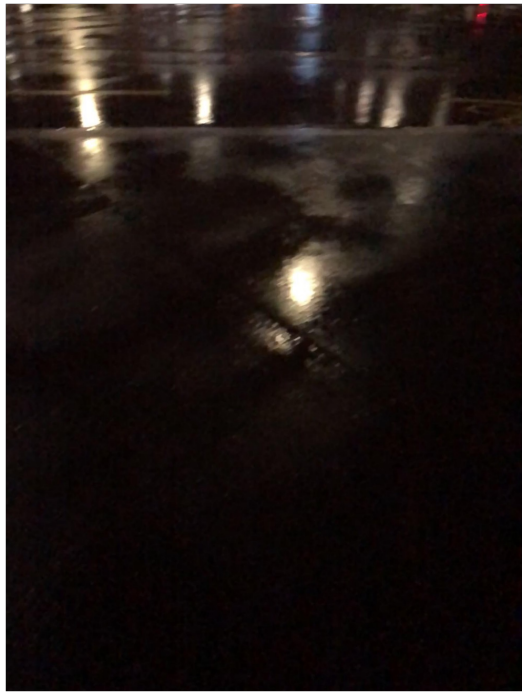
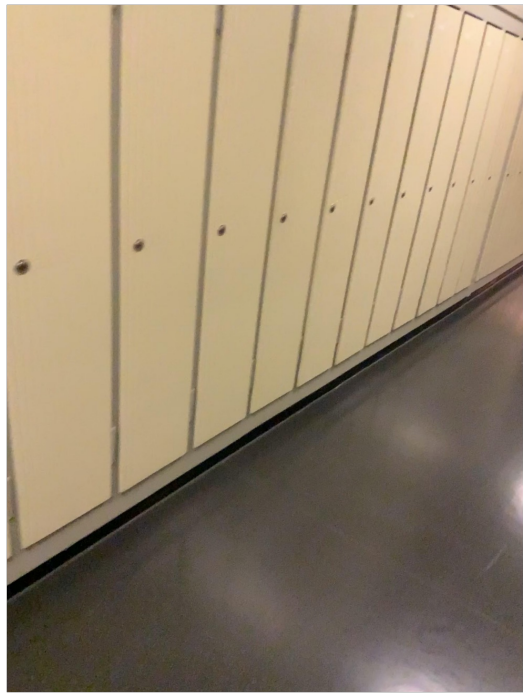
- Making this kind of dataset is a long-term investment (2+ years)
  - Requires good hardware (and preferably good vendor support)
  - Figure out how to get best performance out of your GT sensors
  - Pipelines designed with real-world constraints in mind
  - Constant feedback loop between data and code
  - Set realistic goals, constantly revisit them based on progress
    - Focused on 3 locations, but prioritized coverage / GT quality
    - Still some trade-off between scale and (automated) GT accuracy

Conclusions

- Benchmarking is a combinatorial problem
  - A lot of features, matchers, solvers, RANSAC variants
  - Difficult to include everything
    - Lack of open source implementations
    - Effort to interface everything with the benchmarking pipeline
  - Need to make some hard decision on what to prioritize
    - Representative baselines of large-scale localization methods
  - Benchmark – crowd-source this!
- Very curious what you will come up with!

Conclusions

- For AR, single-image localization is not realistic
  - Makes it harder (or even impossible)
  - Minor improvements might not translate to practical applications
  - Scalability in mind – some methods required 100+GB RAM for triangulation


Q&A