AndroAR

github.com/alexdamian/AandroAR

Alexandru Damian

prof. dr. ing. Adina Magda Florea
as. drd. ing. Andrei Ismail

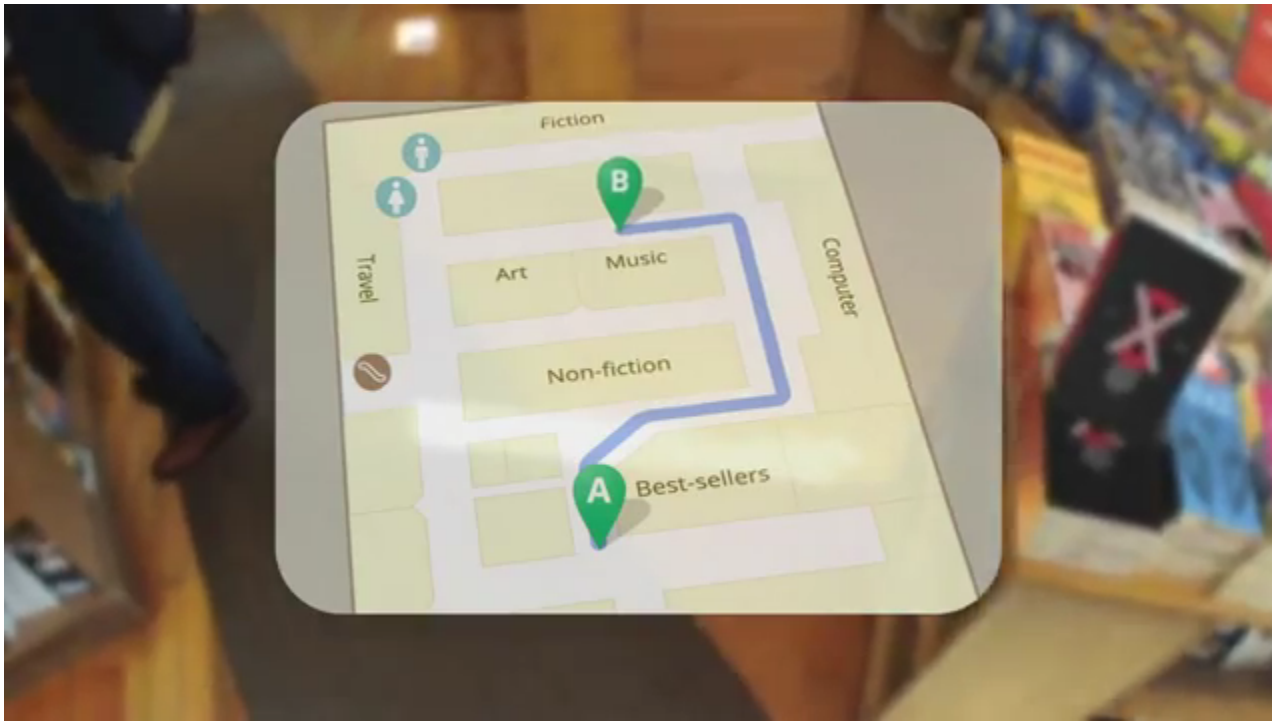
What is it? Scenario

- system (with front-end mobile application) that allows navigation through a potentially unknown environment and displays information for landmarks that are visible on the camera's phone feed

- tourists travelling through foreign cities
- freshmen getting around campus

Google Glasses

- Google IO 2012 conference
- Augmented Reality w/o smartphone



Approaches

System will use: **camera feed** and **database of object information**

And typically **one** of:

- sensors (GPS, compass): Street View

- image recognition software: Layar

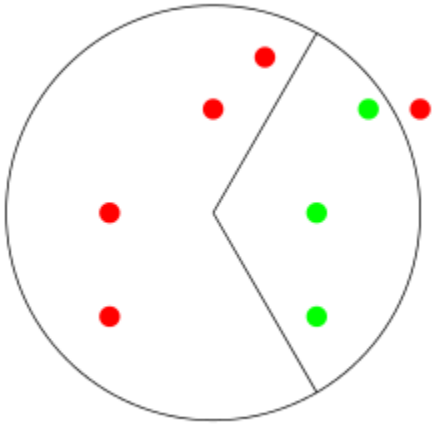
Example

- sensor generated AR

junaio Augmented Reality app

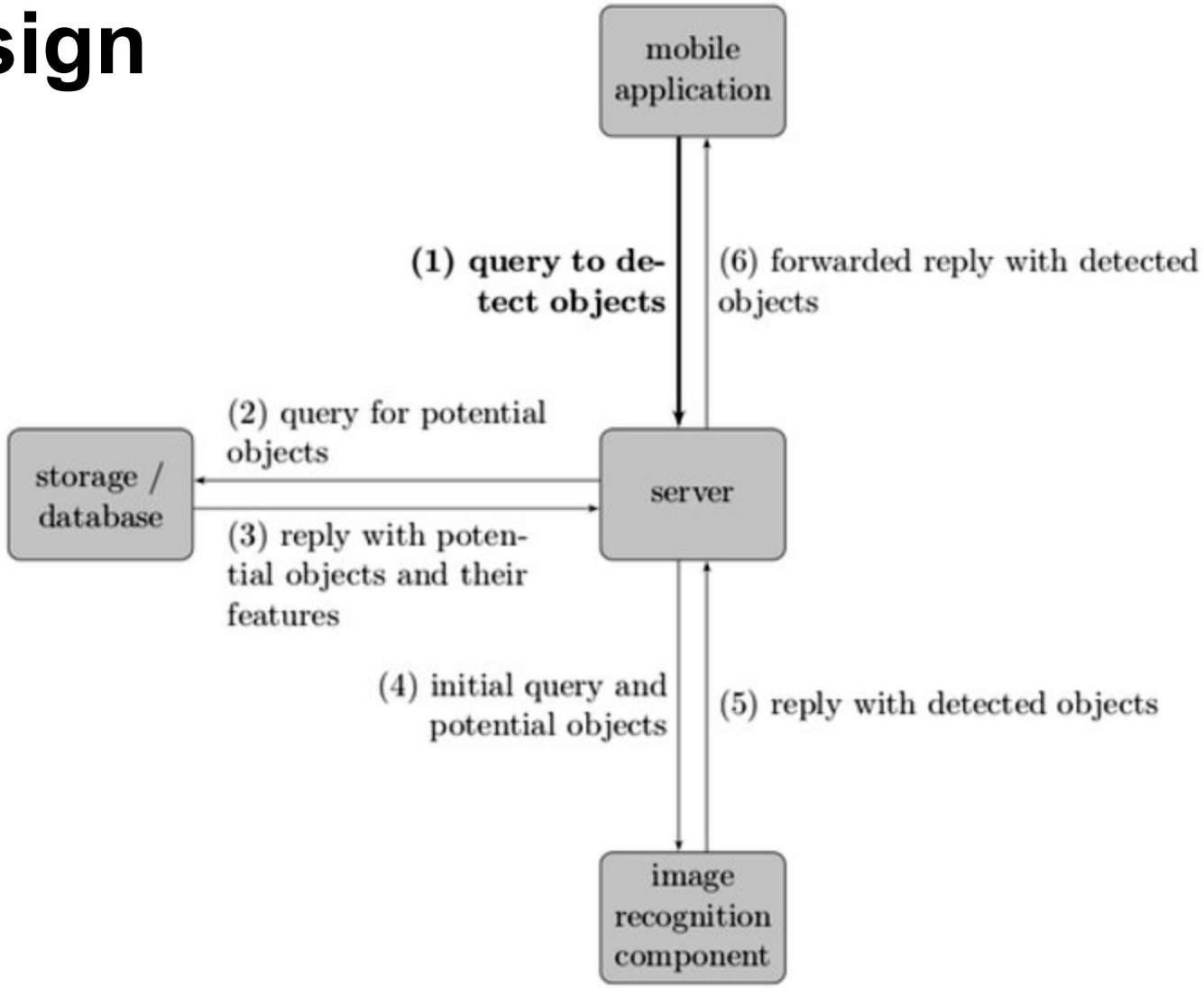
Our approach

- Combination of
  - **GPS positioning** (objects in the user's line of sight)

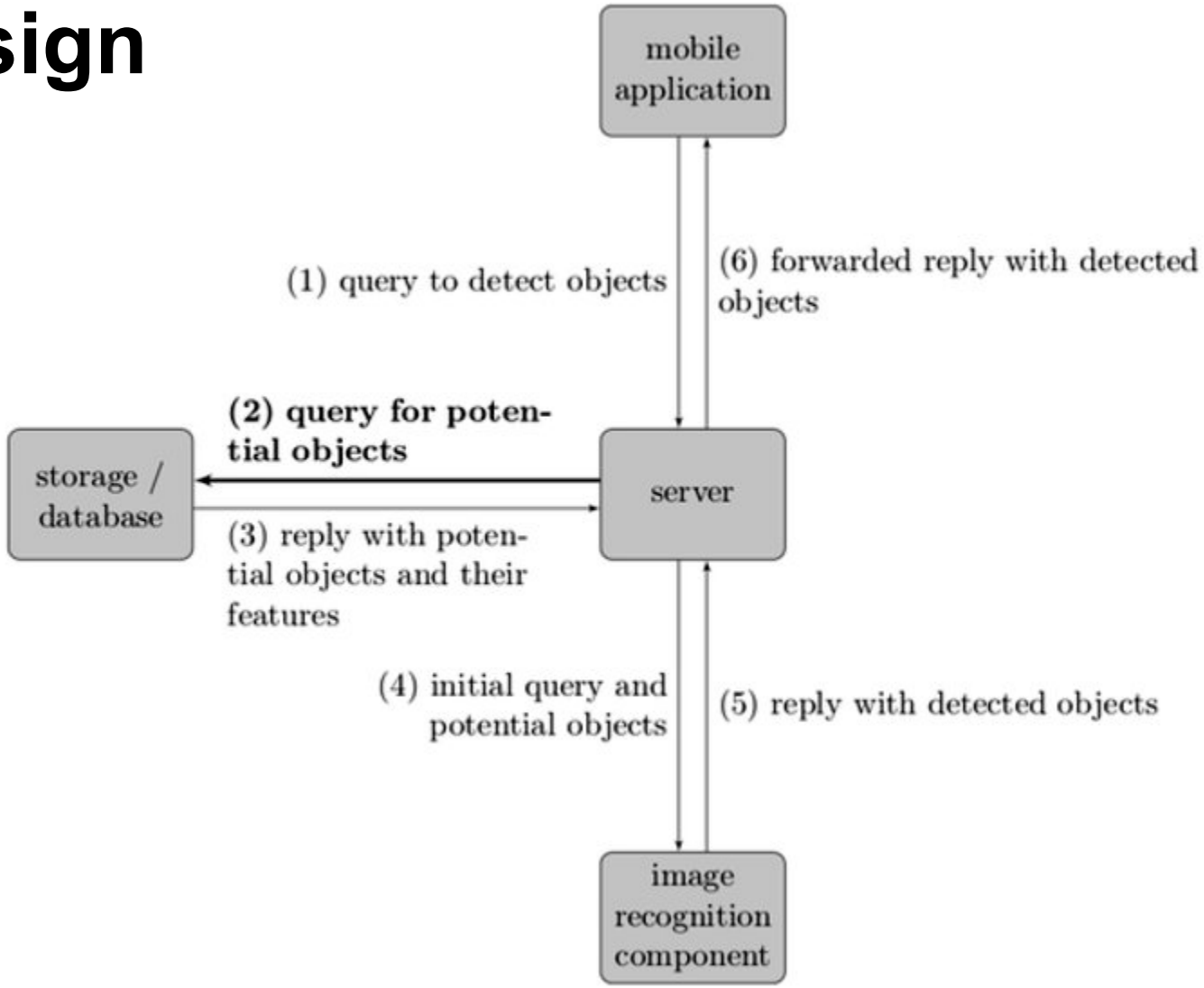


  - **Image recognition** (removal of false positives)

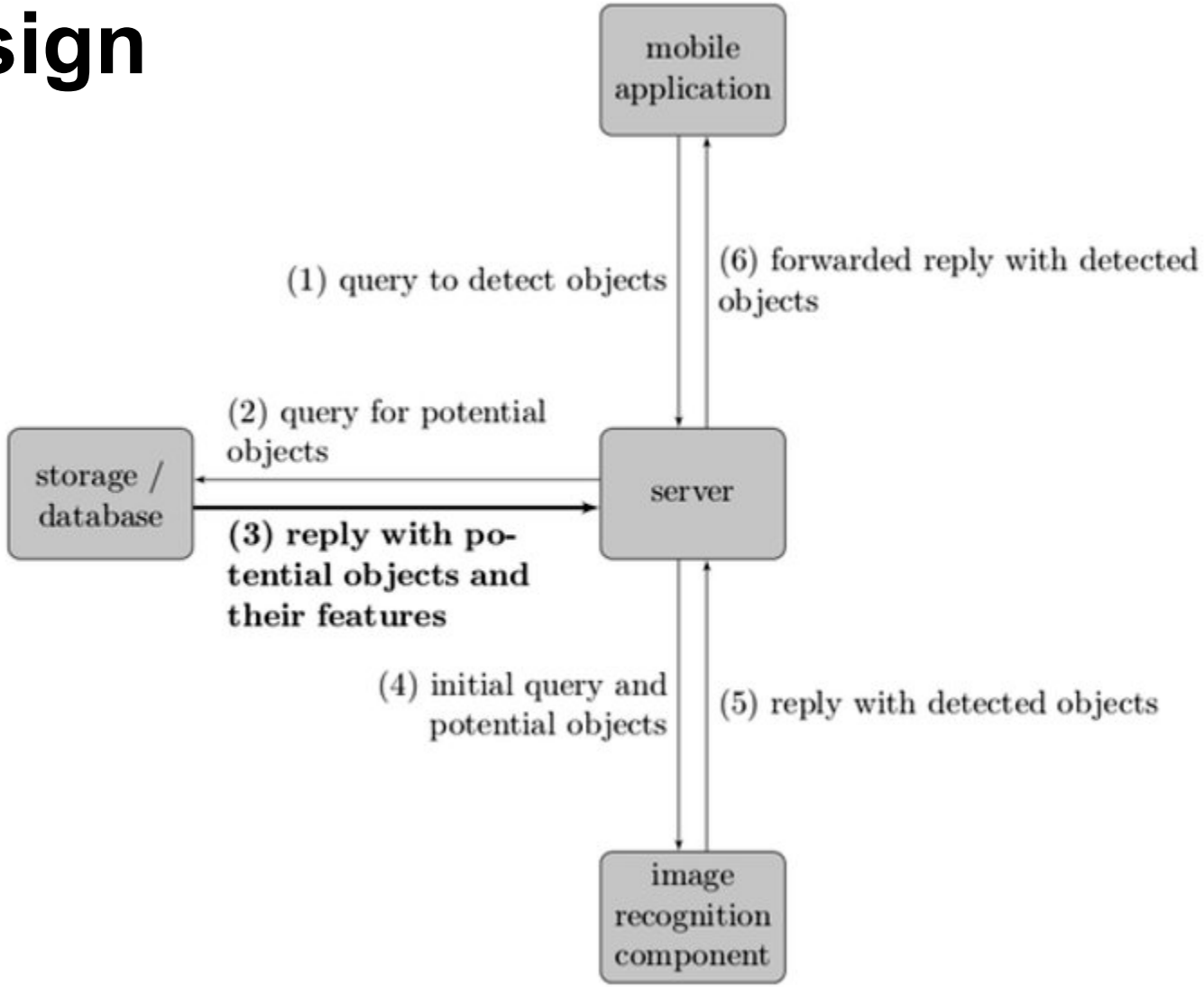
Design


mobile application
(1) query to detect objects
(6) forwarded reply with detected objects
(2) query for potential objects
storage / database
server
(3) reply with potential objects and their features
(4) initial query and potential objects
(5) reply with detected objects
image recognition component

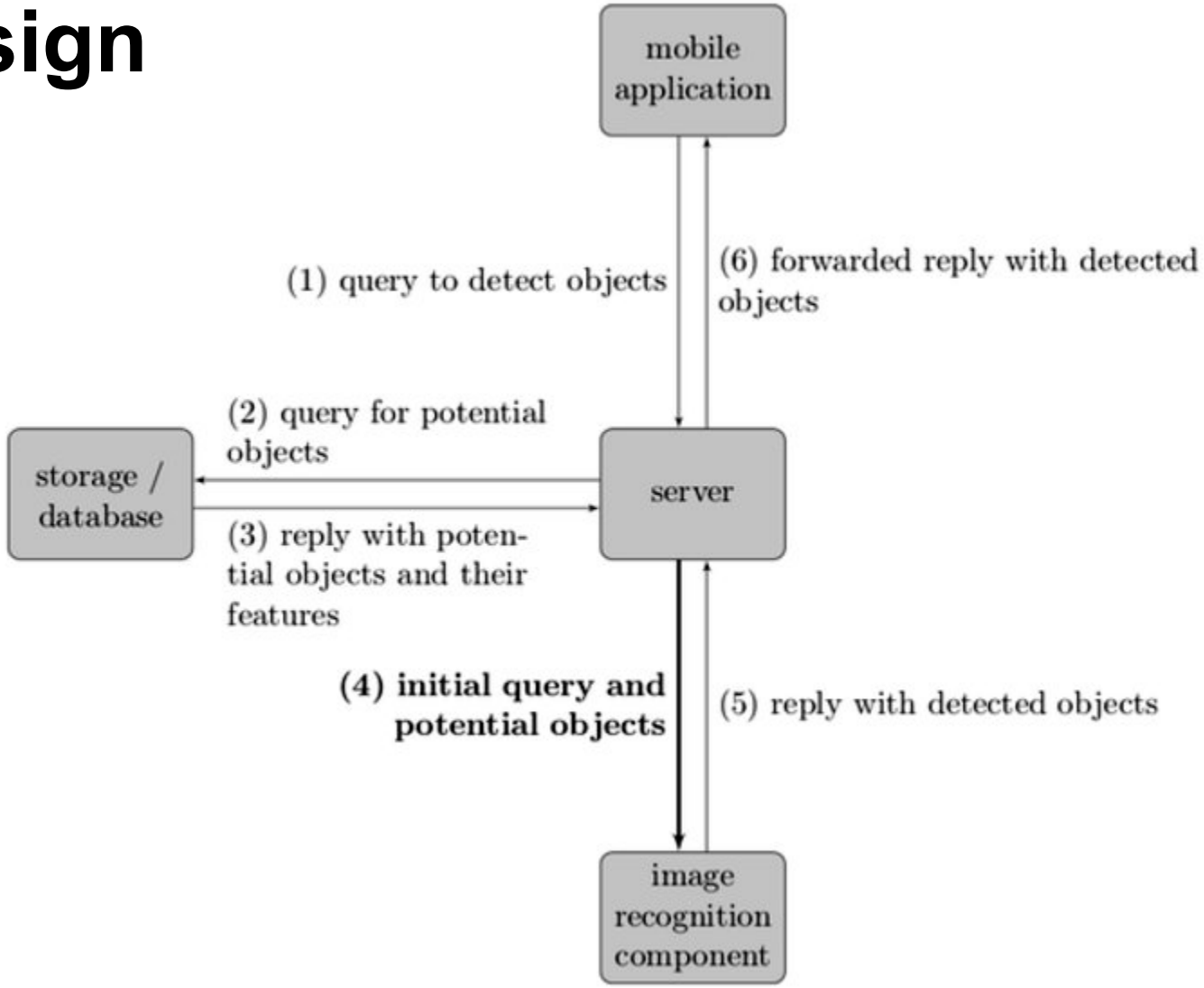
Design


mobile application
(1) query to detect objects
(6) forwarded reply with detected objects
(2) query for potential objects
storage / database
server
(3) reply with potential objects and their features
(4) initial query and potential objects
(5) reply with detected objects
image recognition component

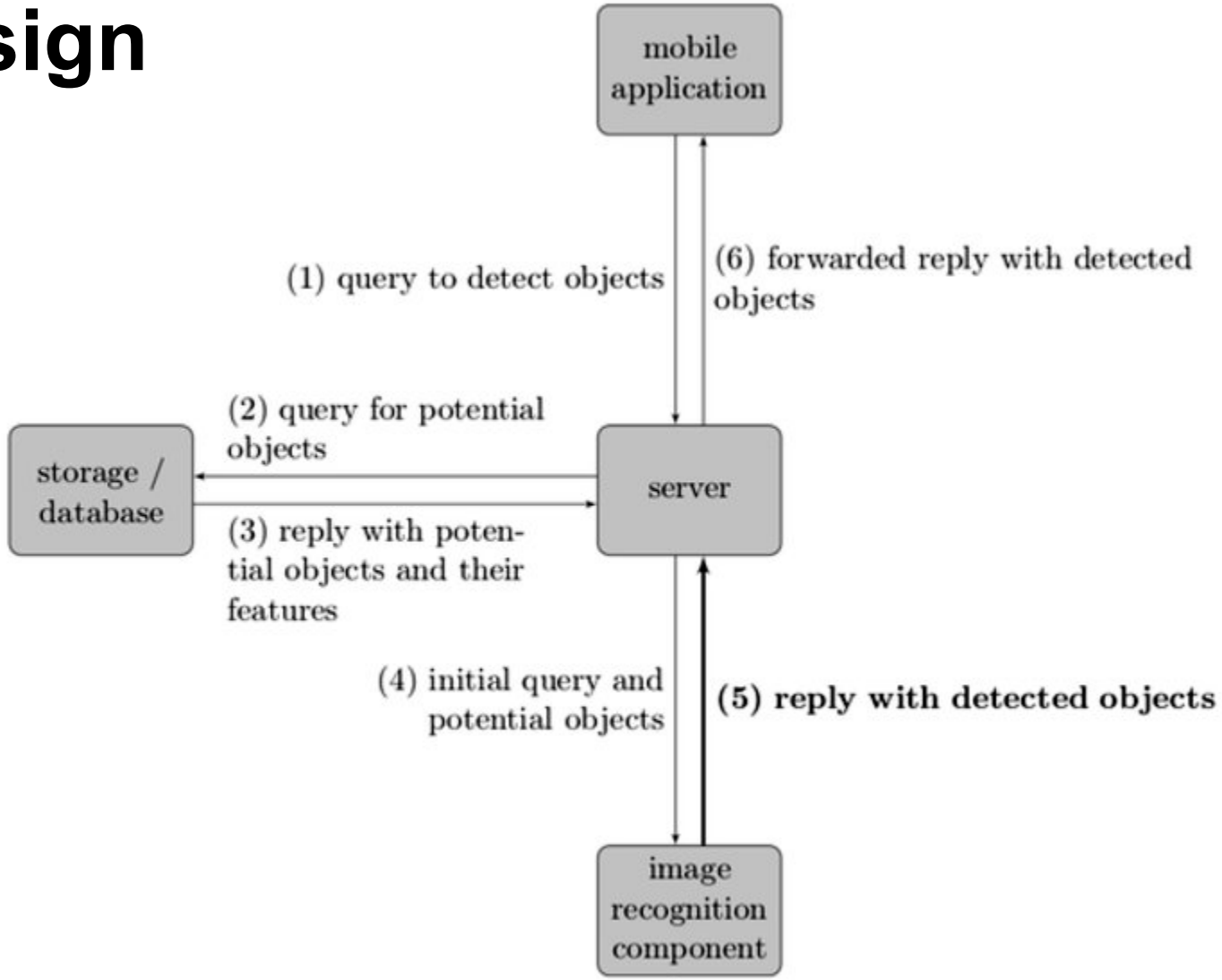
Design


mobile application
(1) query to detect objects
(6) forwarded reply with detected objects
(2) query for potential objects
storage / database
server
(3) reply with potential objects and their features
(4) initial query and potential objects
(5) reply with detected objects
image recognition component

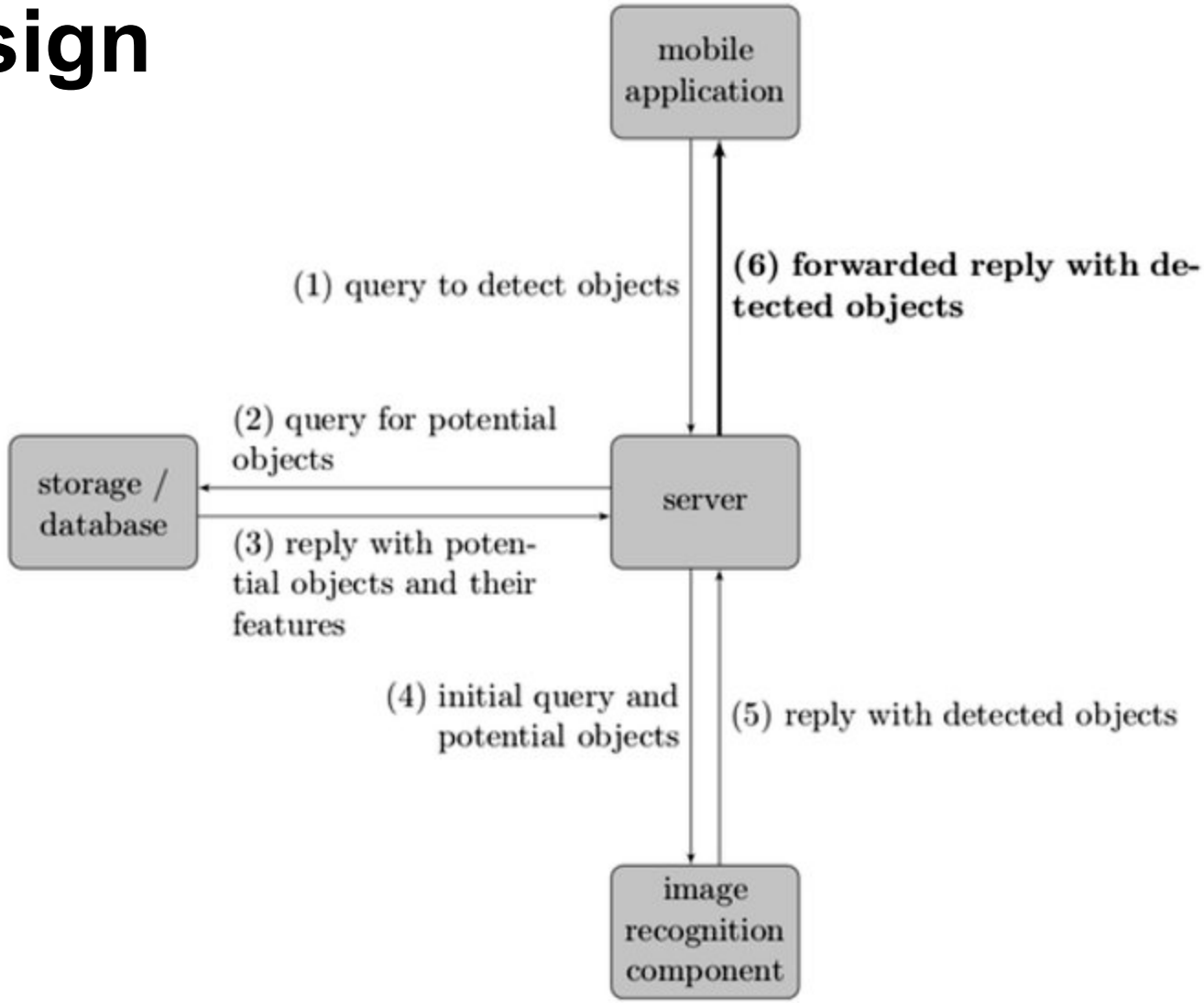
Design


mobile application
(1) query to detect objects
(6) forwarded reply with detected objects
(2) query for potential objects
storage / database
server
(3) reply with potential objects and their features
(4) initial query and potential objects
(5) reply with detected objects
image recognition component

Design


mobile application
(1) query to detect objects
(6) forwarded reply with detected objects
(2) query for potential objects
storage / database
server
(3) reply with potential objects and their features
(4) initial query and potential objects
(5) reply with detected objects
image recognition component

Design


mobile application
(1) query to detect objects
(6) forwarded reply with detected objects
(2) query for potential objects
storage / database
server
(3) reply with potential objects and their features
(4) initial query and potential objects
(5) reply with detected objects
image recognition component

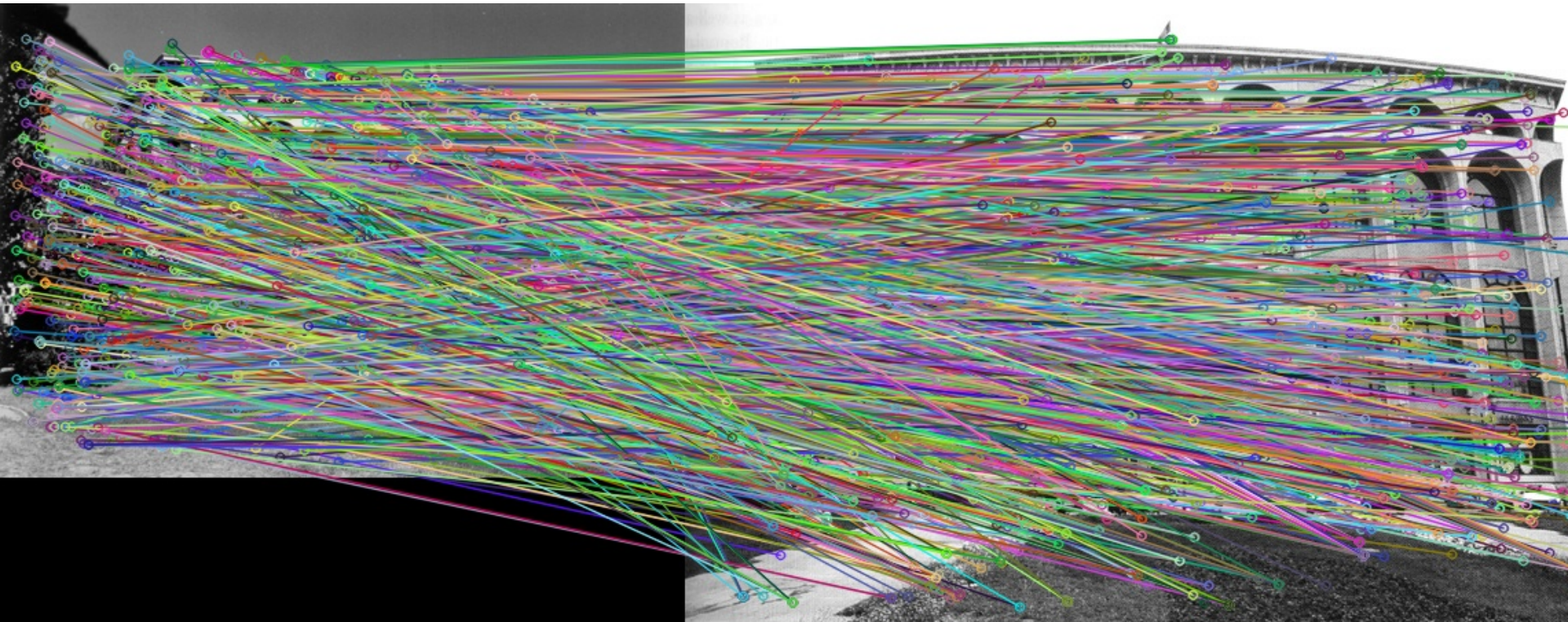
Database

- Stores: **images, features, objects' metadata**
- **2** types of queries: **query** and **store request**
- **easily and optimally distributed**
- **schema-less**

- Netflix & Cassandra: 3.3 million writes/sec, w/ 288 machines

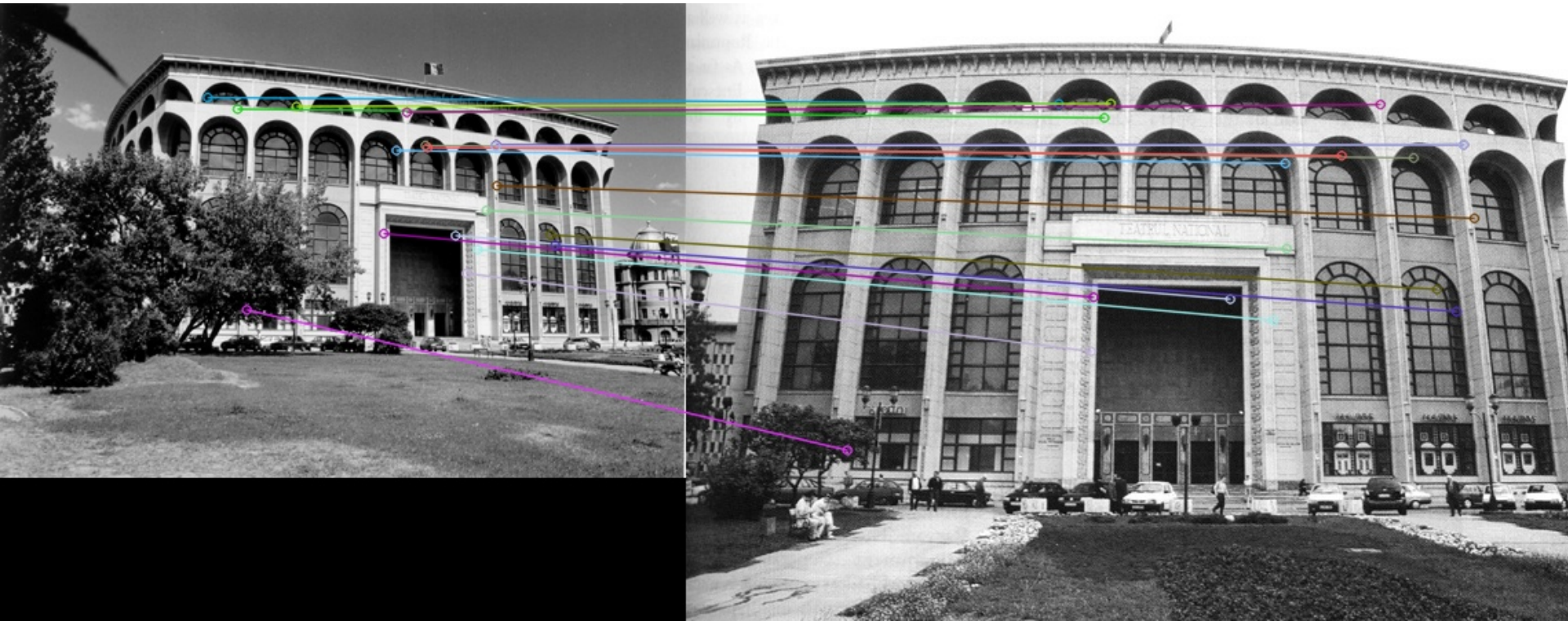
OpenCV

- most well-known library
- SIFT, SURF
- FLANN-based matcher



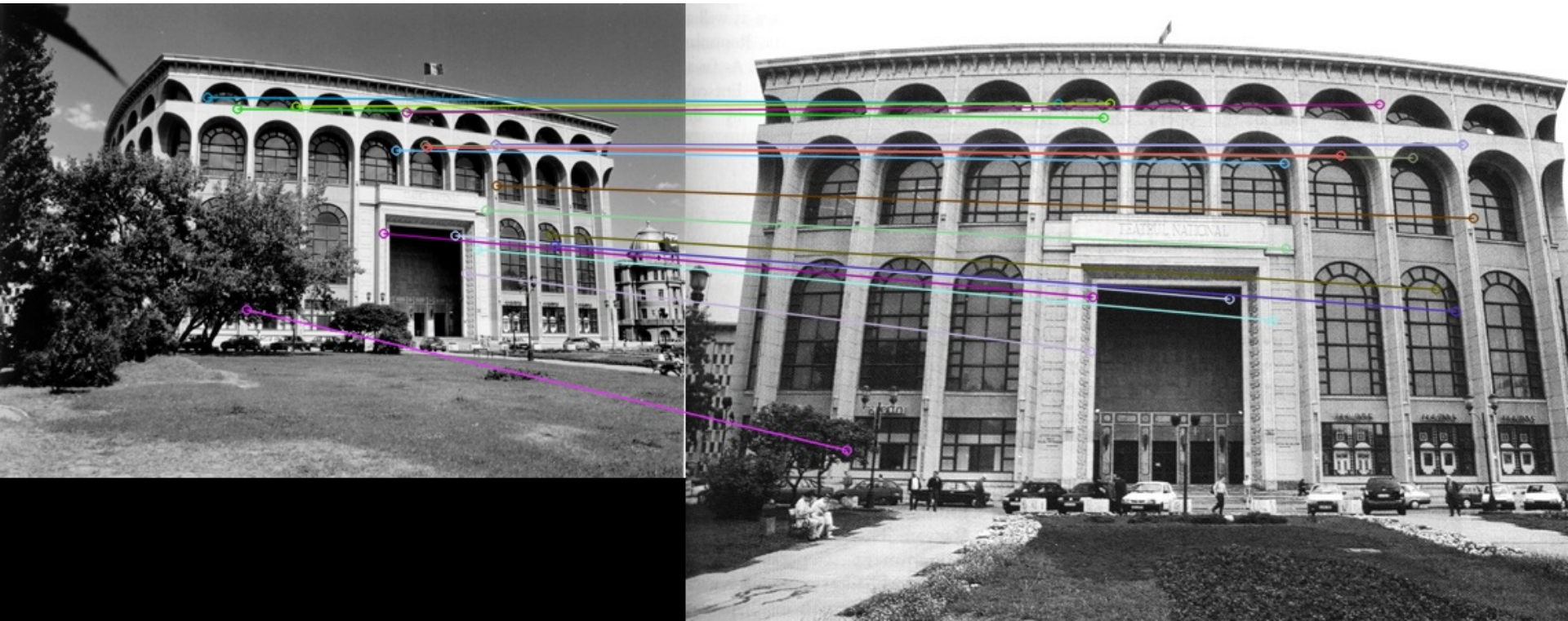
OpenCV(2)

- How it should look like:



Purgers - problems

1. **repetitive features** (*i.e. windows*)
2. **features matched incorrectly**



Purgers - solutions (1)

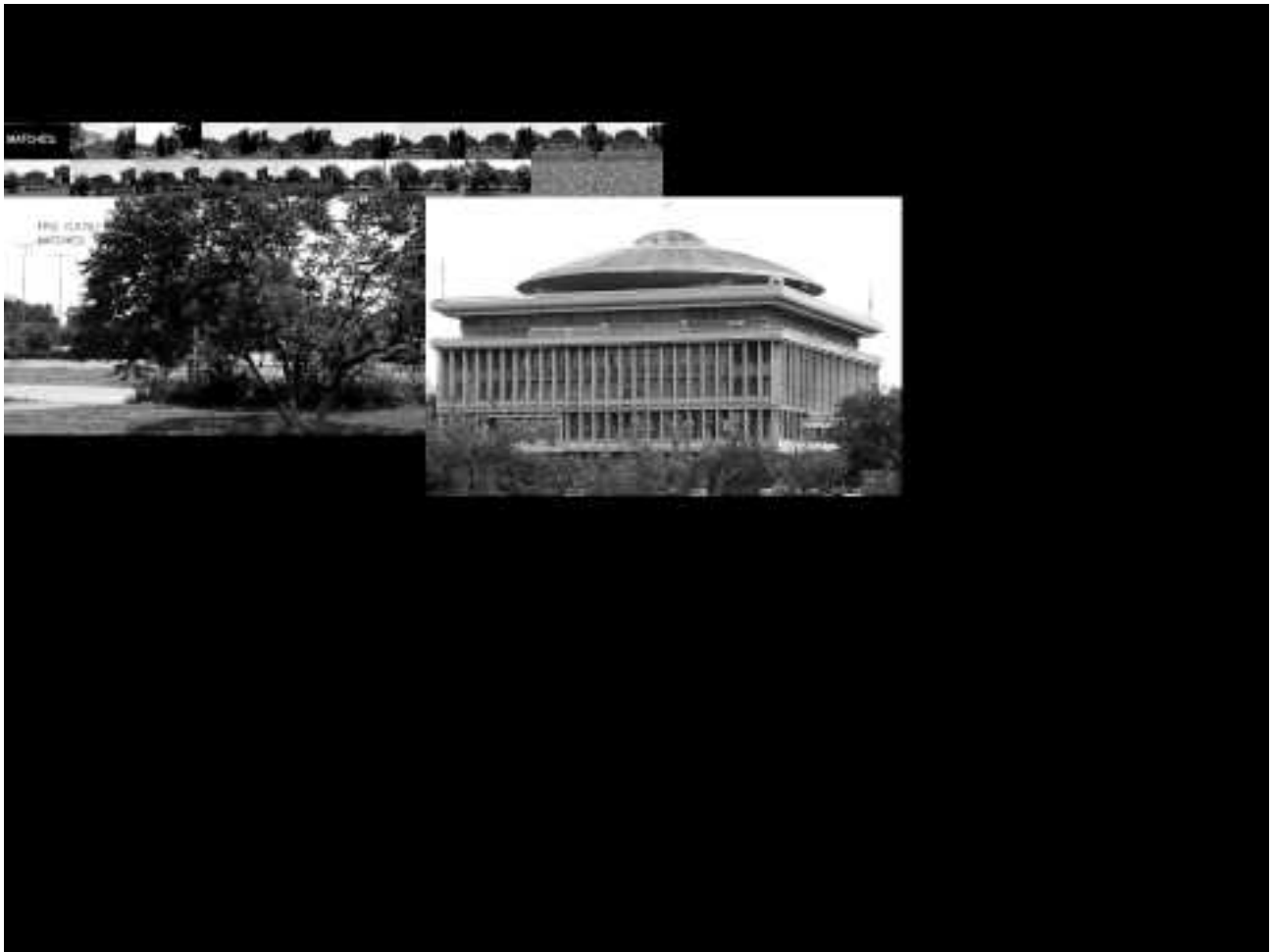
- Problem: **repetitive features** (*i.e. windows*)
  - we're matching query image **(Q)** to existing image **(E)**;
- Solution:
  - for any feature in **(Q)**, get the top 2 choices for matches in **(E)**;
  - if they have roughly the same certainty of being valid, drop the match;
  - swap **(Q)** with **(E)** and repeat;
  - keep matches that are symmetric.

Purgers - solutions (2)

- Problem: **features matched incorrectly**
  - we're matching query image **(Q)** to existing image **(E)**;
- Observations:
  - **(Q)** and **(E)** are *looking* at the same object, from different viewports (**stereovision**);
  - matches lie on the **epipolar line;**
- Solution:
  - compute the **fundamental matrix (F):**
    (Q) = (F) * (E) using **RANSAC;**
  - remove all matches **(Q) - (E)** that do not have the same epipolar lines.

End-result

- simulation of the mobile app behaviour:



Crowdsourcing

- user input (**store requests**)
- **passive example generation**
  - from queries;
  - from storage requests.

purple - query frame; green - valid passive examples; red - invalid examples

Future work

- sparse queries, backed by feature tracking on the mobile application;
- passive example generation;
- better inference of localization features;

- database improvements (sharding);
- security

Thanks

- Thanks to **Andrei Petre (yr. 2)** for implementing the **Android application**.

Questions