Probabilistic data structures

CS524: Big Data Visualization & Analytics

Fabio Miranda

https://fmiranda.me

Overview

- Introduction:
  - High-dimensional data
  - Random projections
  - Hash functions & hash tables
- Probabilistic data structures
  - Bloom filter
  - Linear counter
  - Loglog counter
  - Count-min sketch

High-dimensional data

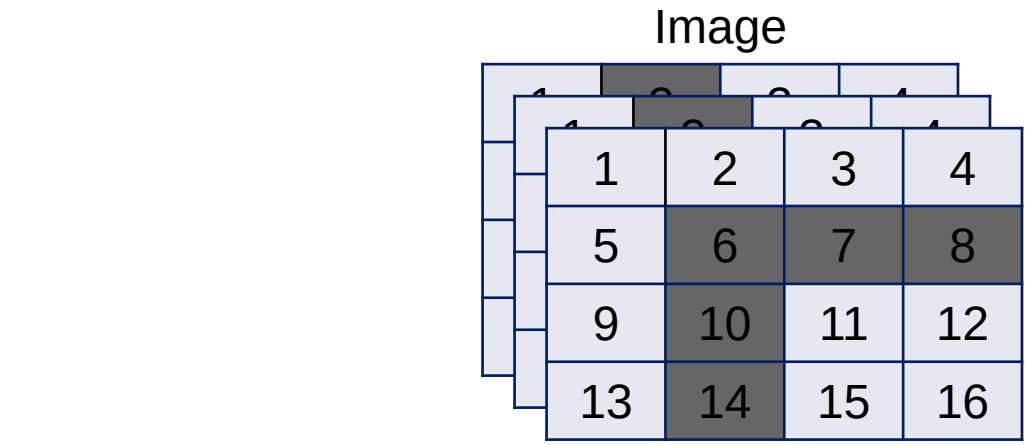
Image

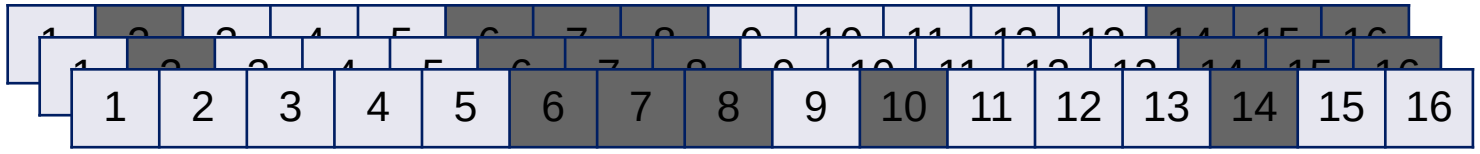
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Pixels

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

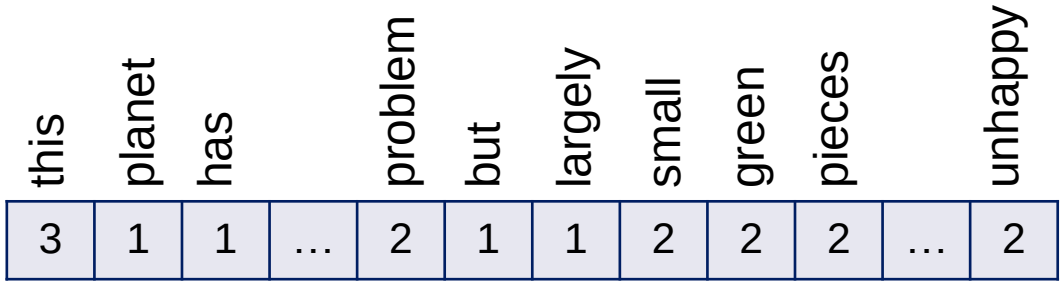
High-dimensional data





High-dimensional data

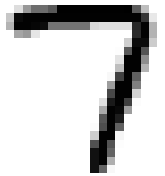
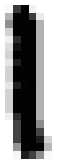
“This planet has - or rather had - a problem, which was this: most of the people living on it were unhappy for pretty much of the time. Many solutions were suggested for this problem, but most of these were largely concerned with the movement of small green pieces of paper, which was odd because on the whole it wasn't the small green pieces of paper that were unhappy.”



MNIST example

- Dataset with images with size 28 x 28 pixels (784 dimensions).
- Defining distance in terms of pixels:

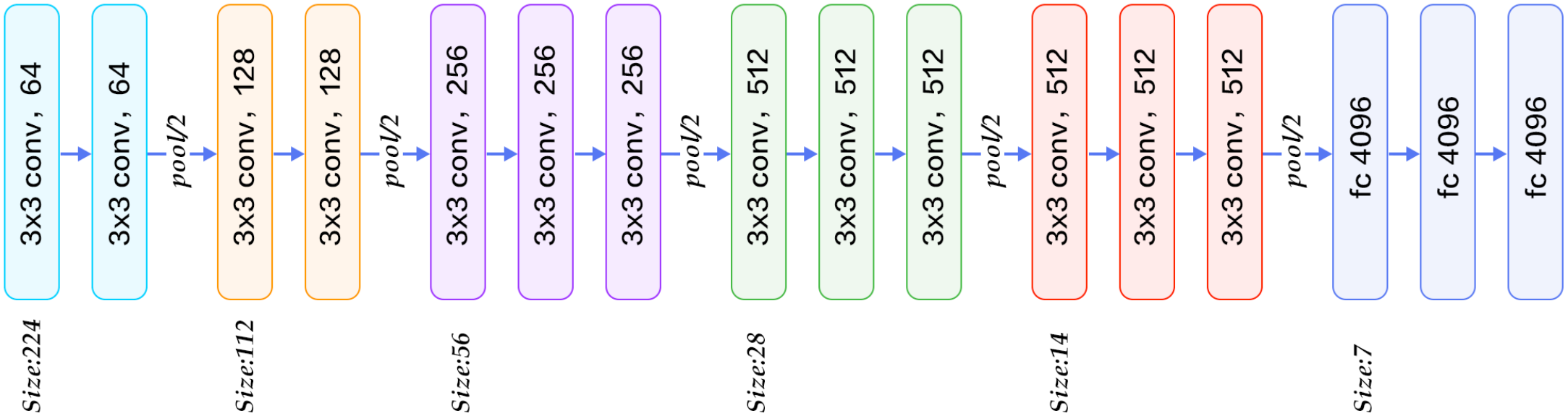
$$d(a,b) = \sum_{1 \le i,j \le 28} \left(a_{ij} - b_{ij}\right)^2$$

$d($  ,  $) = 53.64$

Improving: dimensionality reduction

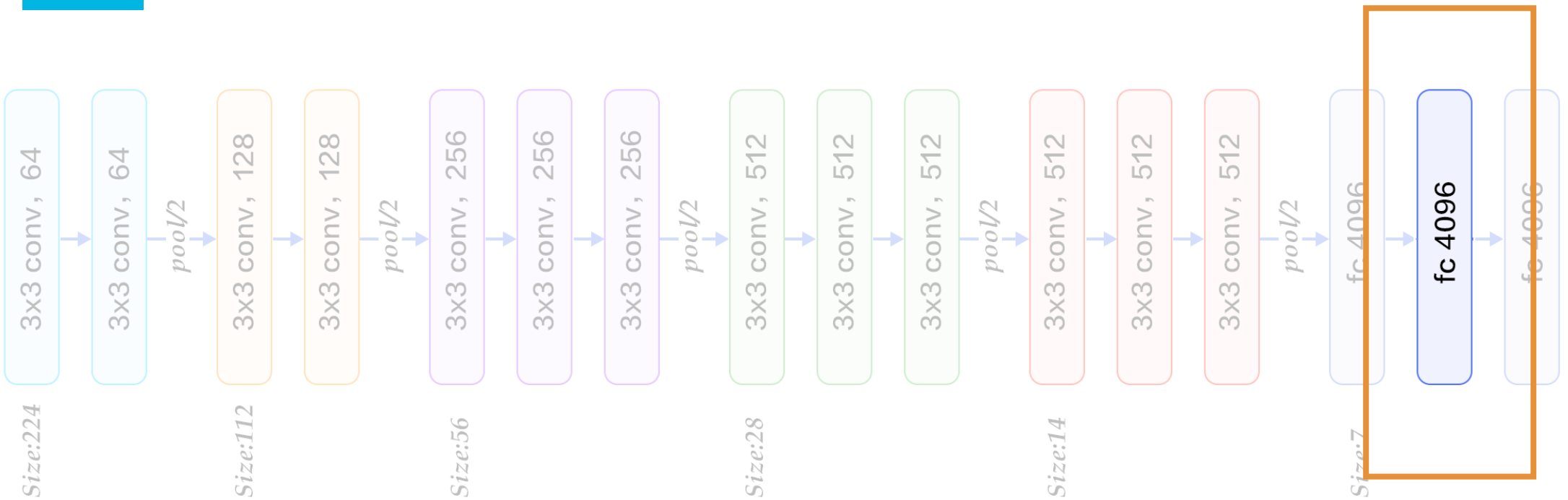
1. Start with high-dimensional data
2. Reduce number of dimensions to 10-1000 dimensions.
3. Operate in that lower dimensional space.

Image embeddings



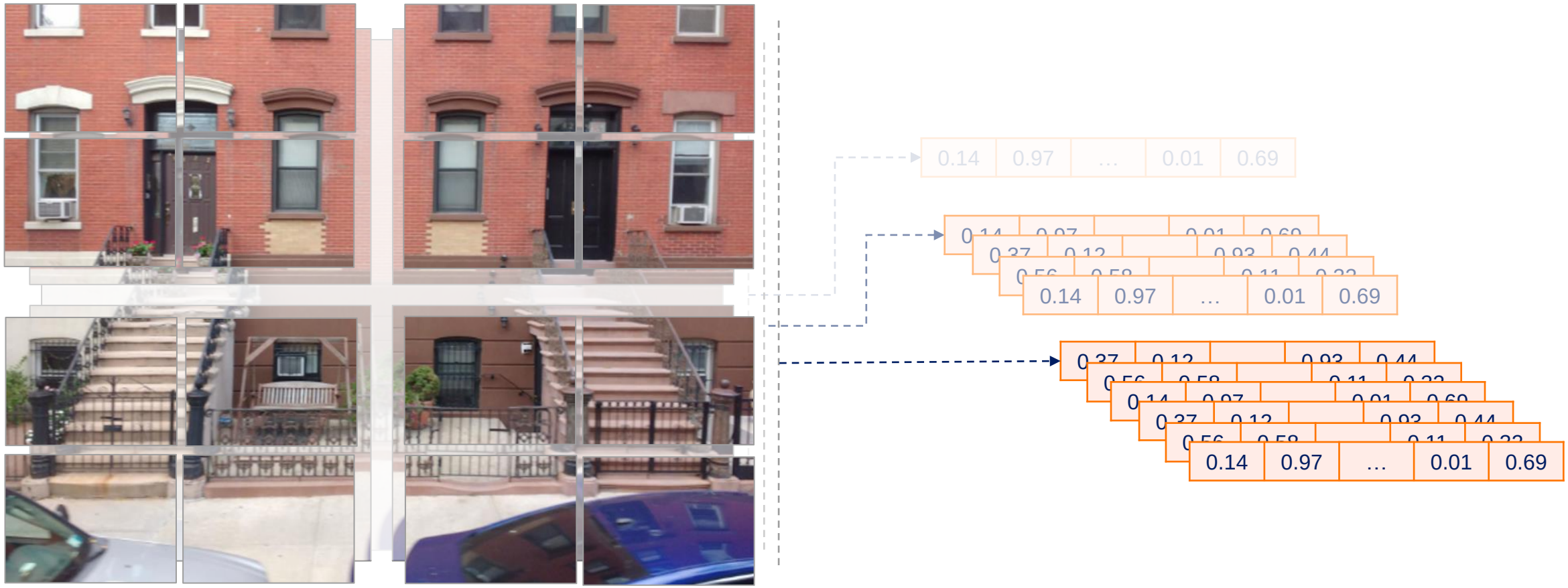
VGG

Image embeddings



VGG

Image embeddings



Distance in low-dimensional space

1. Run image through network.
2. Use one of the layers as an image descriptor.
3. Cosine distance in the reduced space.

$$\text{f}(\ \text{  }\ ) = \mathbf{v}, \mathbf{v} \in \mathbb{R}^{4096}$$

$$d(a,b) = \left(\frac{\mathbf{u}}{|\mathbf{u}|} - \frac{\mathbf{v}}{|\mathbf{v}|}\right)^2$$

High-dimensional data

- What's the point?
  - Vector models are everywhere.
  - Applications in language processing, computer vision, graphics, visualization, …

Dimensionality reduction

Given an object, how to search for its nearest neighbors in an efficient manner?

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

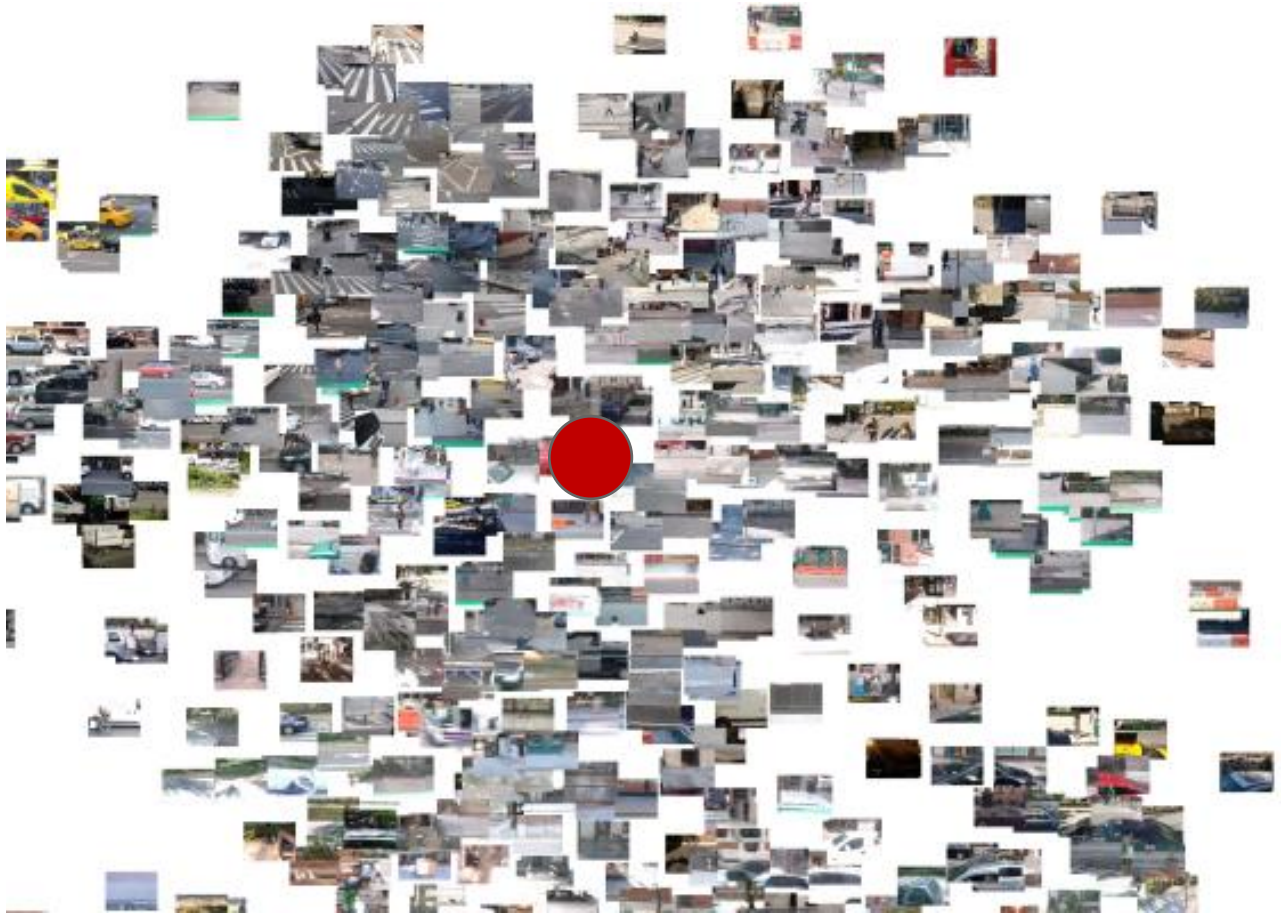
Dimensionality reduction

Given an object, how to search for its nearest neighbors in an efficient manner?

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

Naïve approach: simply compare query object with every single object in the dataset.

Dimensionality reduction



Nearest neighbor problem: find the closest point.

Dimensionality reduction

Nearest neighbor problem: find the closest point.
Approximation: neighbors are in a circle with small radius.

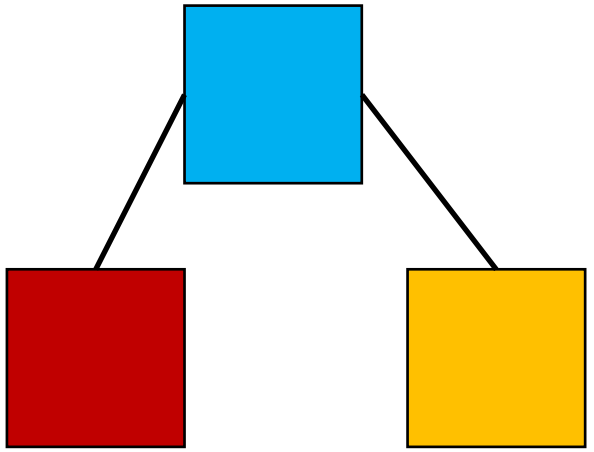
Dimensionality reduction

Nearest neighbor problem: find the closest point.
Approximation: neighbors are in a circle with small radius.

Random projections

Core idea: partition space into smaller spaces.

Random projections



High-dimensional data represented as a matrix $\mathbf{D}_{nxm}$, it can be projected onto a lower dimensional space $\mathbf{P}$ with k dimensions:

$$[Projected(\mathbf{P})]_{kxn} = [Random(\mathbf{R})]_{kxd}[Original(\mathbf{D})]_{dxn}$$

Matrix $\mathbf{R}$ is called random matrix, with elements drawn from gaussian distribution.

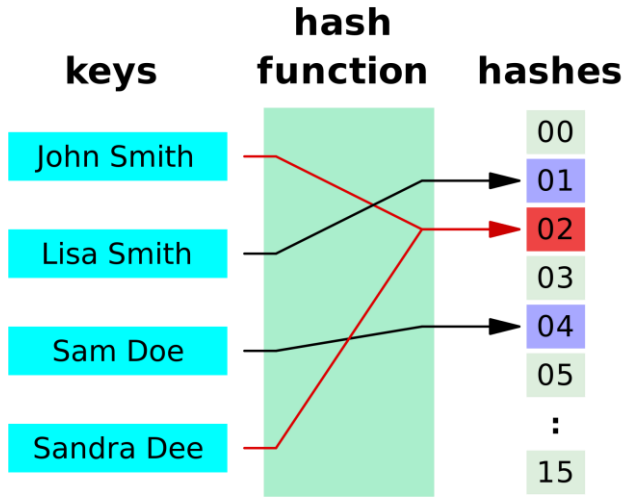
Annoy

- Python library for nearest neighbor search using random projections.

```
from annoy import AnnoyIndex
import random

f = 40
t = AnnoyIndex(f, 'angular')  # Length of item vector that will be indexed
for i in range(1000):
    v = [random.gauss(0, 1) for z in range(f)]
    t.add_item(i, v)

t.build(10) # 10 trees
t.save('test.ann')

u = AnnoyIndex(f, 'angular')
u.load('test.ann') # super fast, will just mmap the file
print(u.get_nns_by_item(0, 1000)) # will find the 1000 nearest neighbors
```

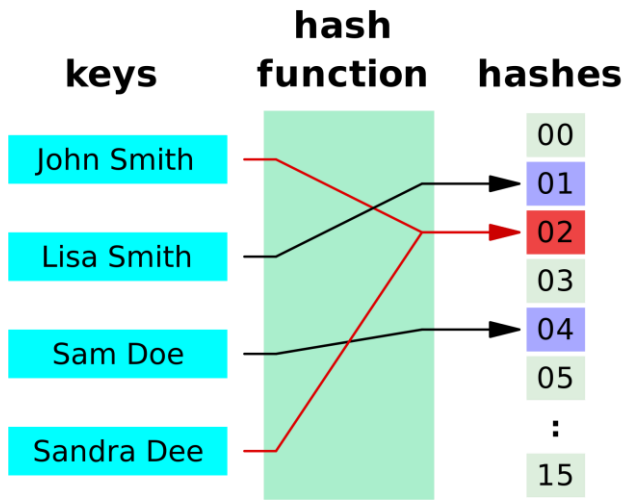
Hash functions

- Convert any input to a fixed-length, deterministic output.
  - Potential side effect: small change in input leads to drastically different hash.
- Hash functions are one-way functions: computationally impractical to reverse.
- Properties of a good hash function:
  - Irreversibility
  - Speed
  - Collision resistance



Hash tables

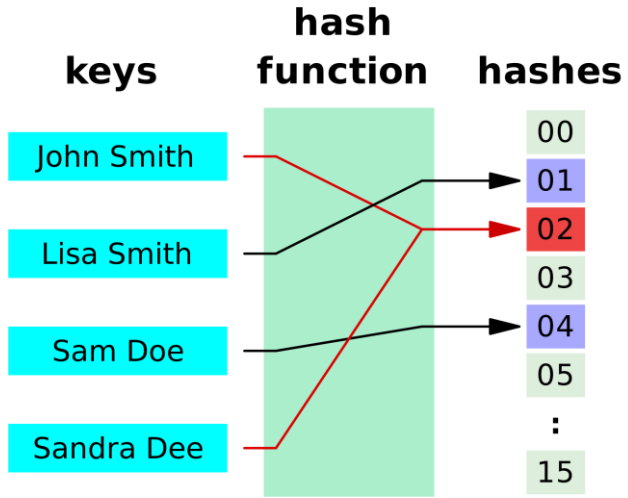
- Hash functions are used in conjunction with hash tables to store and retrieve data items.
- A hash function maps items to array positions.
- Why hash?



Hash tables

- Hash functions are used in conjunction with hash tables to store and retrieve data items.
- A hash function maps items to array positions.
- Why hash?

O(1) look ups



Probabilistic data structures

- Family of approaches that are optimized to use fixed or sublinear memory; often based on hashing.
- Disadvantage: cannot provide exact answers.
- Trade off between error and performance.

Counting example

- Estimate the cardinality of a set $\mathbf{X}$.

$$\{1, 5, 13, 14, 5829, 3, \ldots, 42, 148901\}$$

Counting example

- Estimate the cardinality of a set $\mathbf{X}$.

$$\{1, 5, 13, 14, 5829, 3, \ldots, 42, 148901\}$$

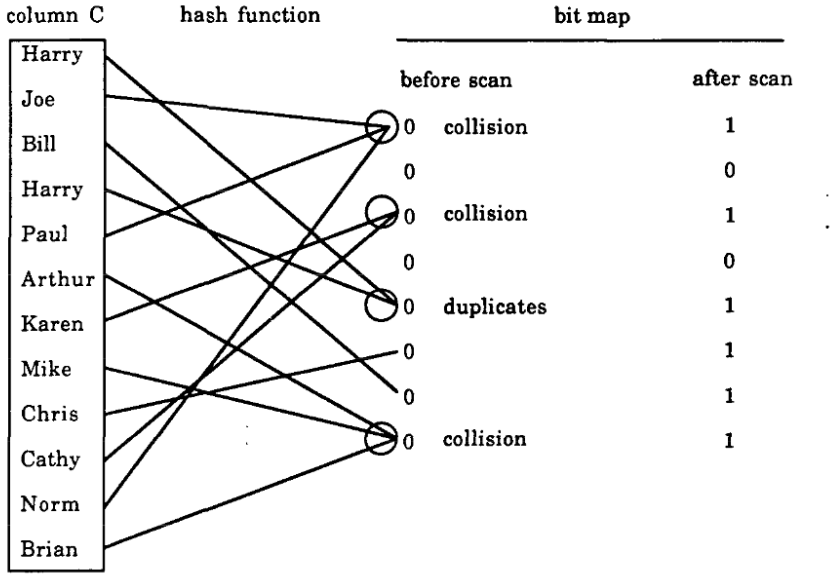
- Naïve solution:
  - Build a list of all unique elements we saw thus far.
  - To avoid storing repeated elements, store them in a sorted form and search on insert.
  - Disadvantage?

Linear counting

- Estimate the cardinality of a set $\mathbf{X}$.

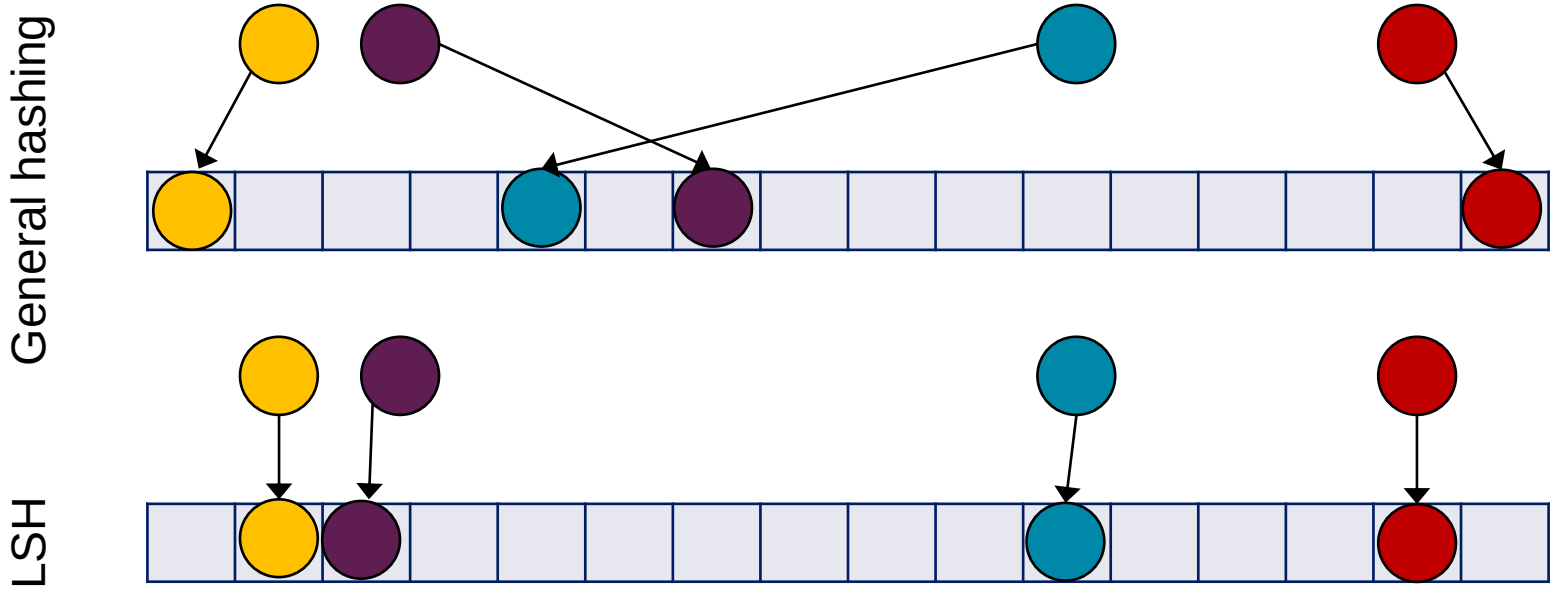
$$\{1, 5, 13, 14, 5829, 3, \ldots, 42, 148901\}$$

- Solution:
  1. Initialize a bit array $\mathbf{D}$ with zeros.
  2. Compute hash of $x \in \mathbf{X}$ to a location in $\mathbf{D}$, set bits to 1.
     - Naturally handles identical items.
  3. $\hat{n} = -m\ lnV_n$



Locality sensitive hashing

- LSH hash data points into buckets so that data points near each other are located in the same buckets with high probability.

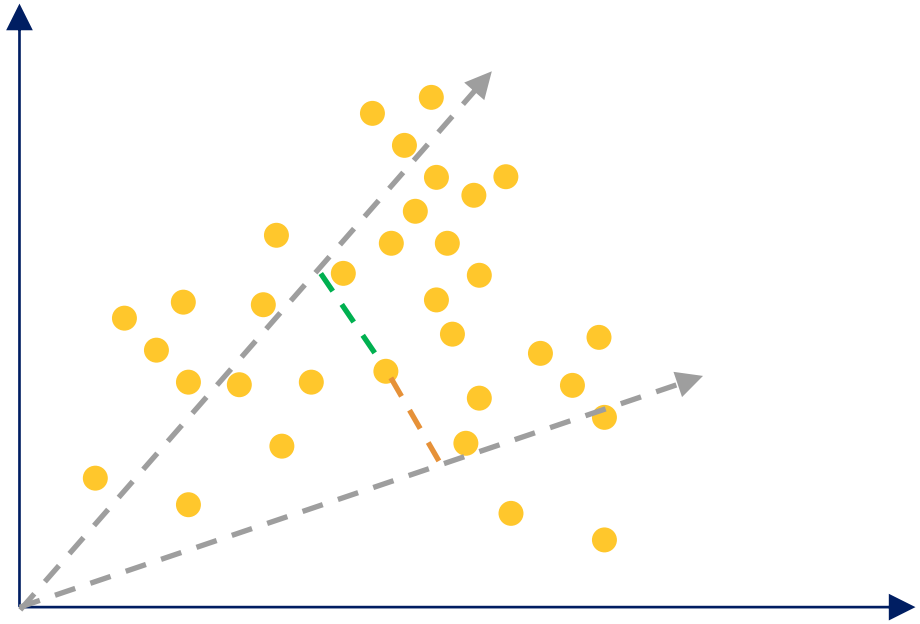


LSH using random projection

1. Create k random vectors.
2. For each random vector, compute the dot product of the random vector and the observations.
    1. Positive result: assign bit value 1.
    2. Negative result: bit value 0.
3. Concatenate all bit values computed for k dot products.

Locality sensitive hashing

- Locality sensitive hashing:



0.37	0.12	…	0.93	0.44

4 kbytes per embedding

0	0	…	1	0

64 bytes per embedding

$$\alpha_{1,2} = \cos^{-1}\left(\frac{\overrightarrow{v_1} \cdot \overrightarrow{v_2}}{|\overrightarrow{v_1}||\overrightarrow{v_2}|}\right)$$

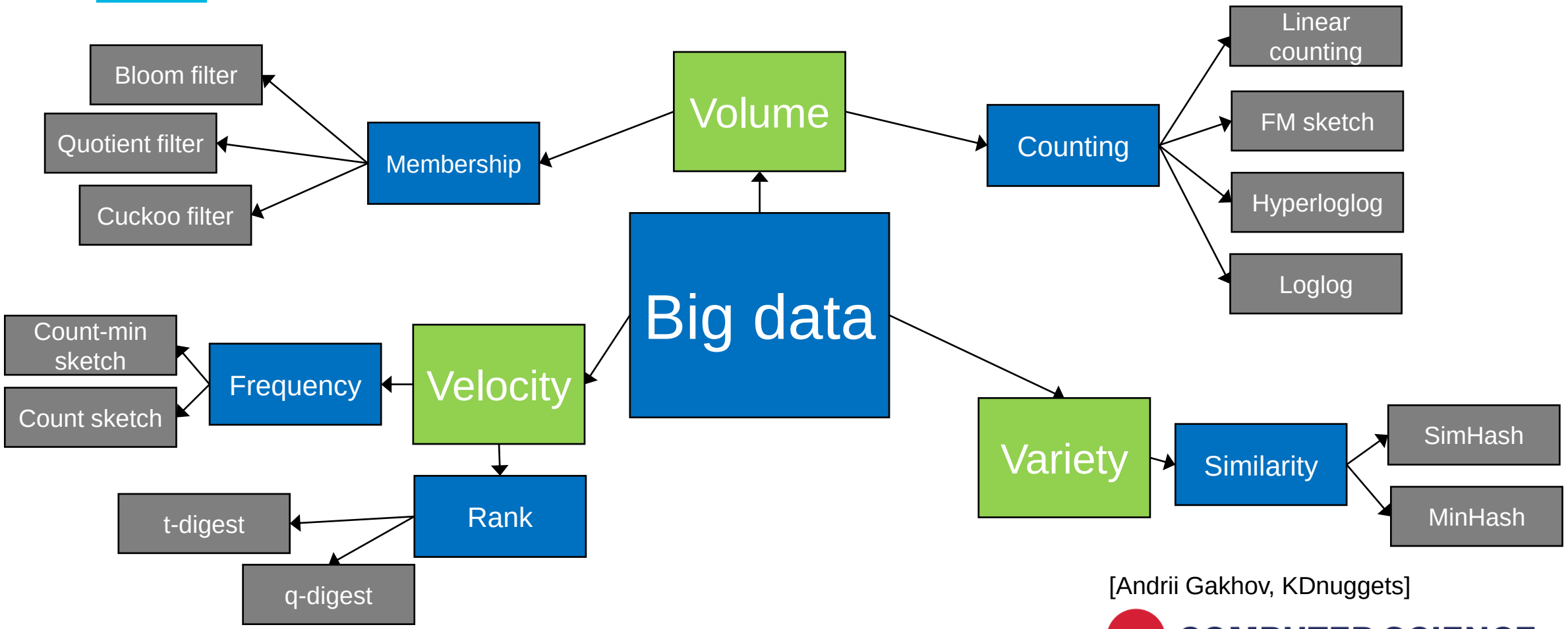
Types of problems

- Membership: test whether an element is a member of a set.
- Frequency: frequency table of events.
- Rank: rank-based statistics (e.g., quantiles).
- Similarity: estimate how similar two sets are.
- Counting: count distinct elements.

Hyperloglog

- Probabilistic data structure for the count-distinct problem.

Probabilistic data structures



[Andrii Gakhov, KDnuggets]

Datasketch

- Python library with implementation of probabilistic data structures.

```
from datasketch.hyperloglog import HyperLogLog

h1 = HyperLogLog()
for d in data1:
    h1.update(d.encode('utf8'))
for d in data2:
    h2.update(d.encode('utf8'))
u = HyperLogLog.union(h1, h2)
```

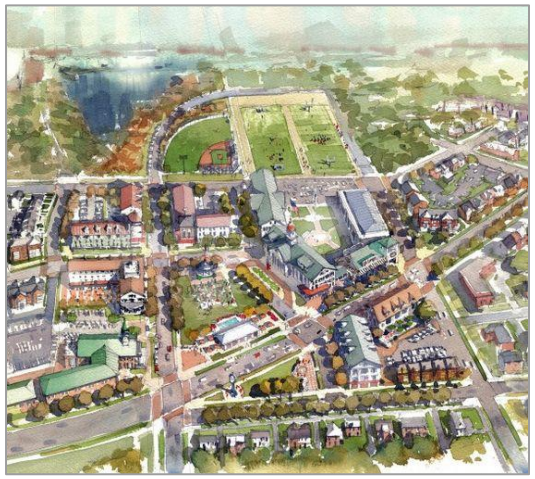
City at the microscale

Urban Mosaic: Visual Exploration of Streetscapes Using Large-Scale Image Data

Miranda et al.

2020 CHI Conference on Human Factors in Computing Systems

Scales of the city



macro

Suburbanization
Transport infrastructure
Population density

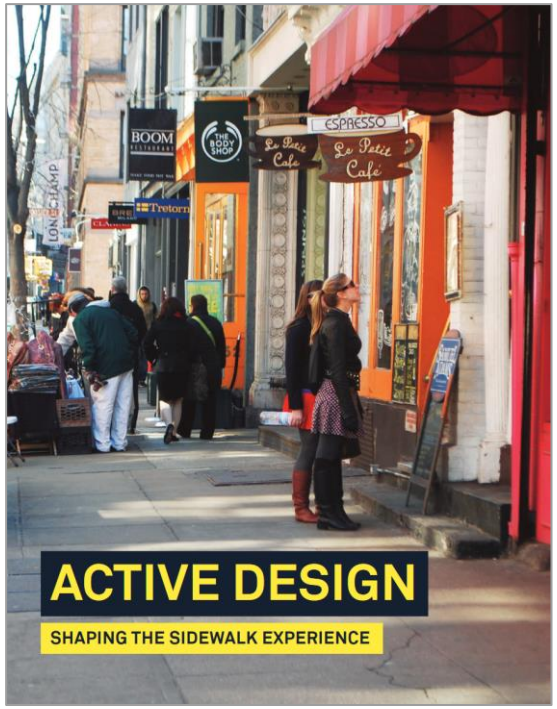
meso

Access to amenities
Green space
Street width

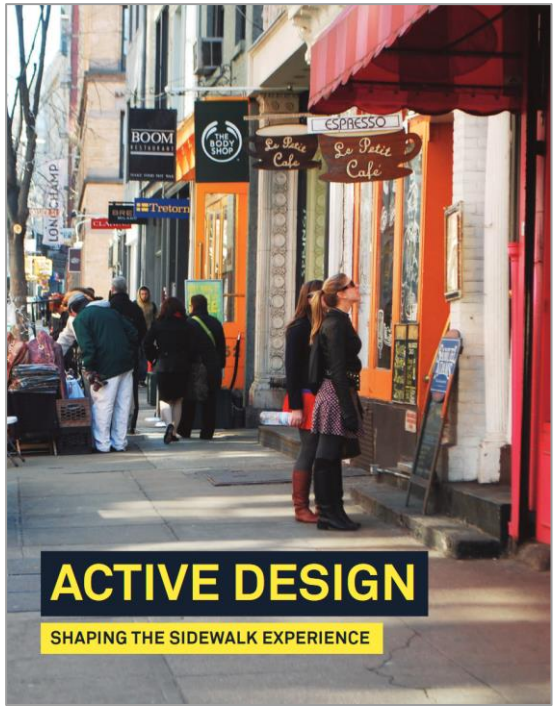
micro

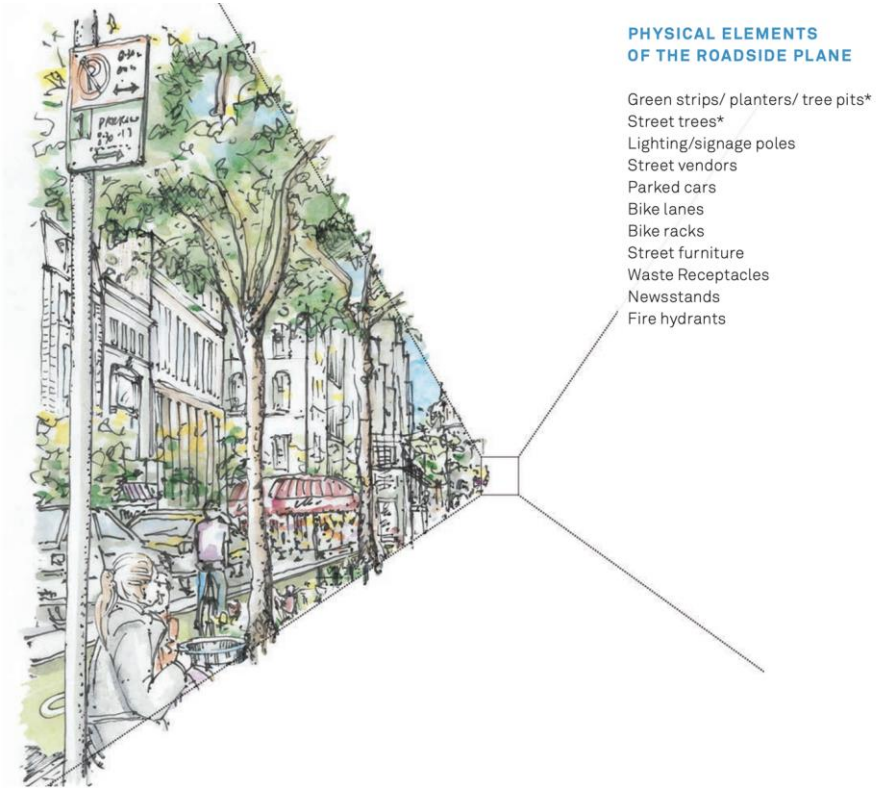
Open-front buildings
Building facades
Sidewalk

City at the microscale



City at the microscale

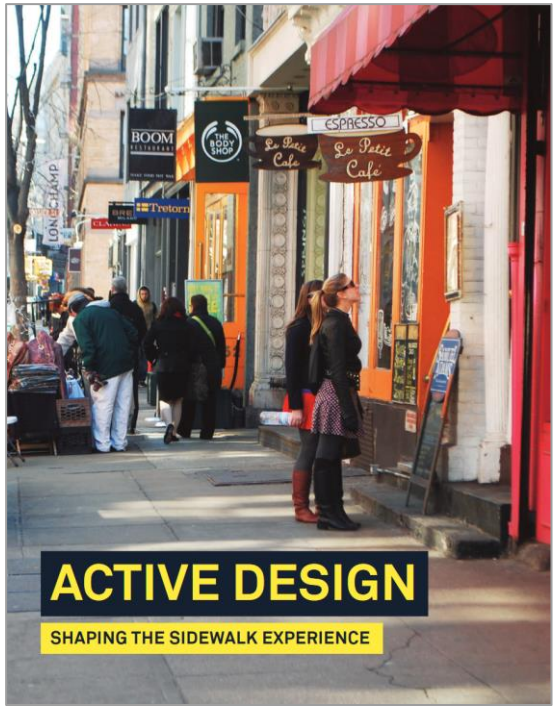




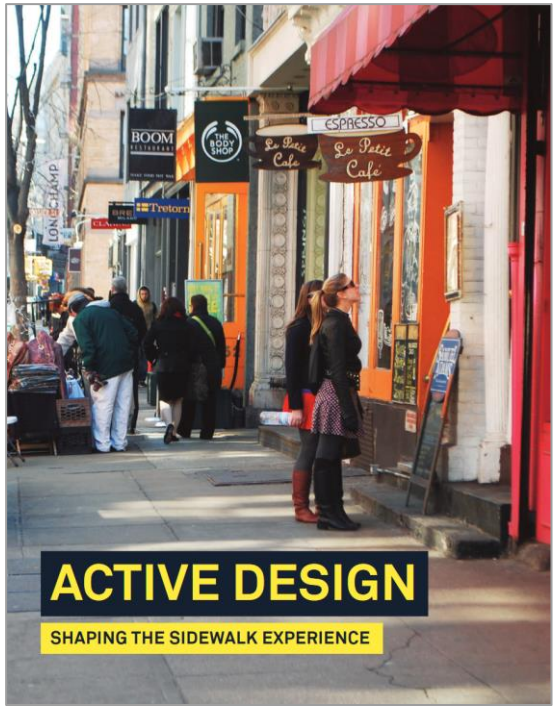
PHYSICAL ELEMENTS OF THE ROADSIDE PLANE

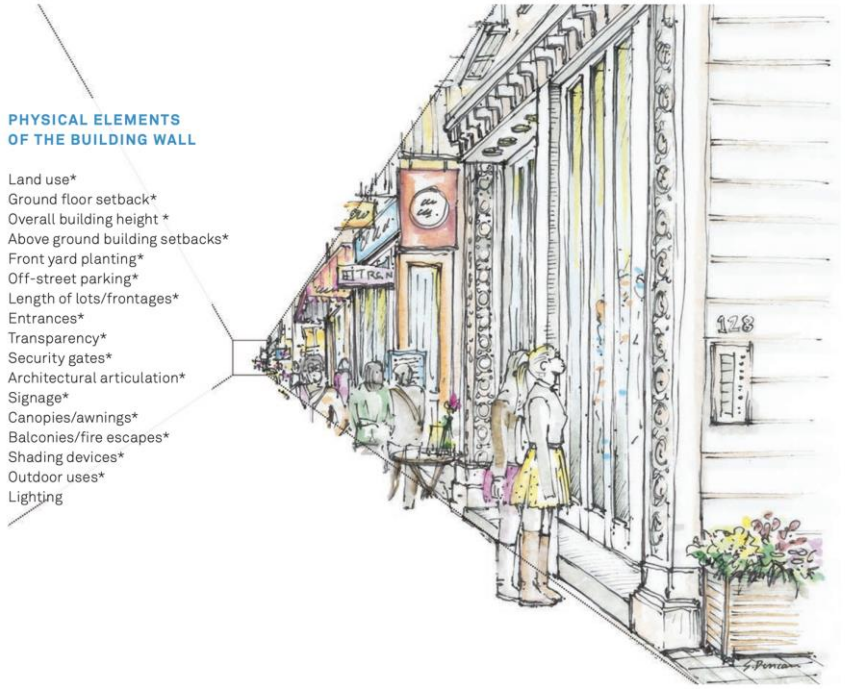
Green strips/ planters/ tree pits*
Street trees*
Lighting/signage poles
Street vendors
Parked cars
Bike lanes
Bike racks
Street furniture
Waste Receptacles
Newsstands
Fire hydrants

City at the microscale

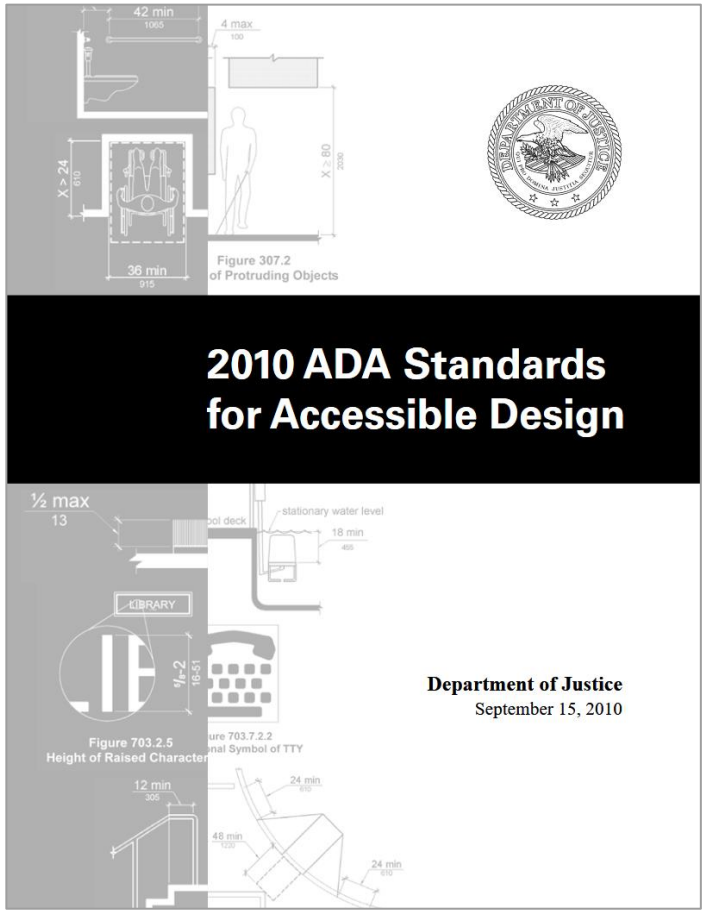


City at the microscale





City at the microscale



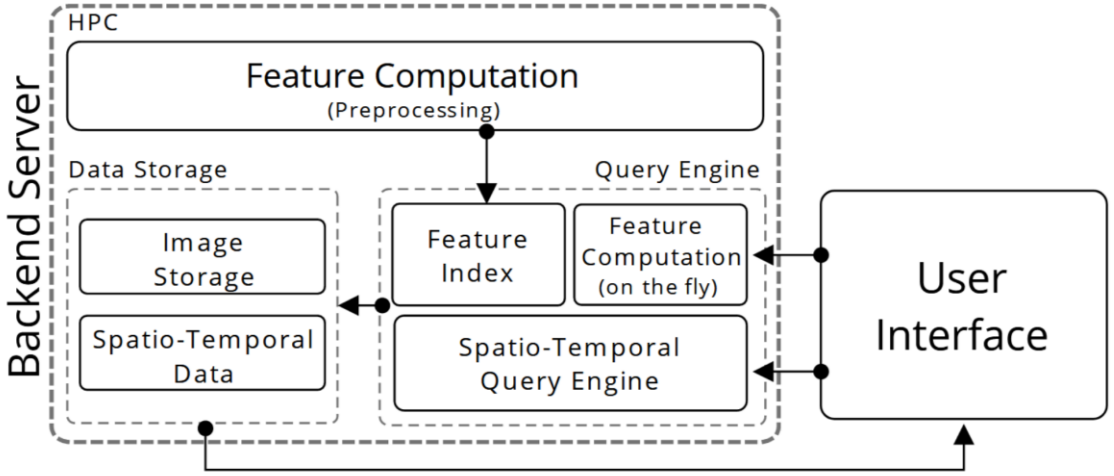
"Newly constructed or altered streets, roads, and highways must contains curb ramps or other sloped areas at intersections to streets, roads, or highways."

Urban Mosaic

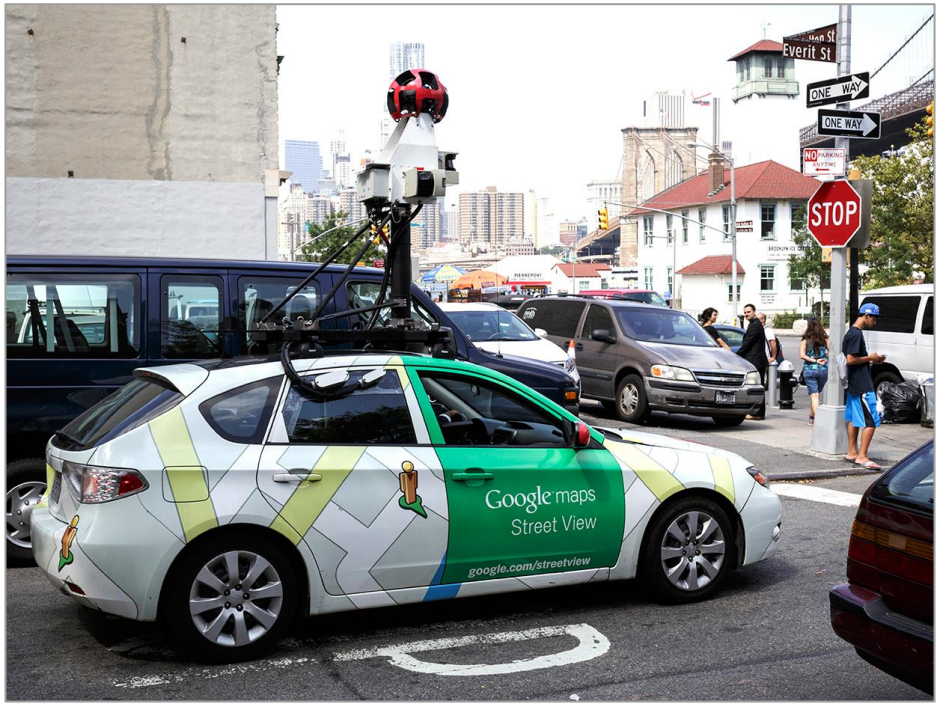
A tool for the exploration of the urban fabric

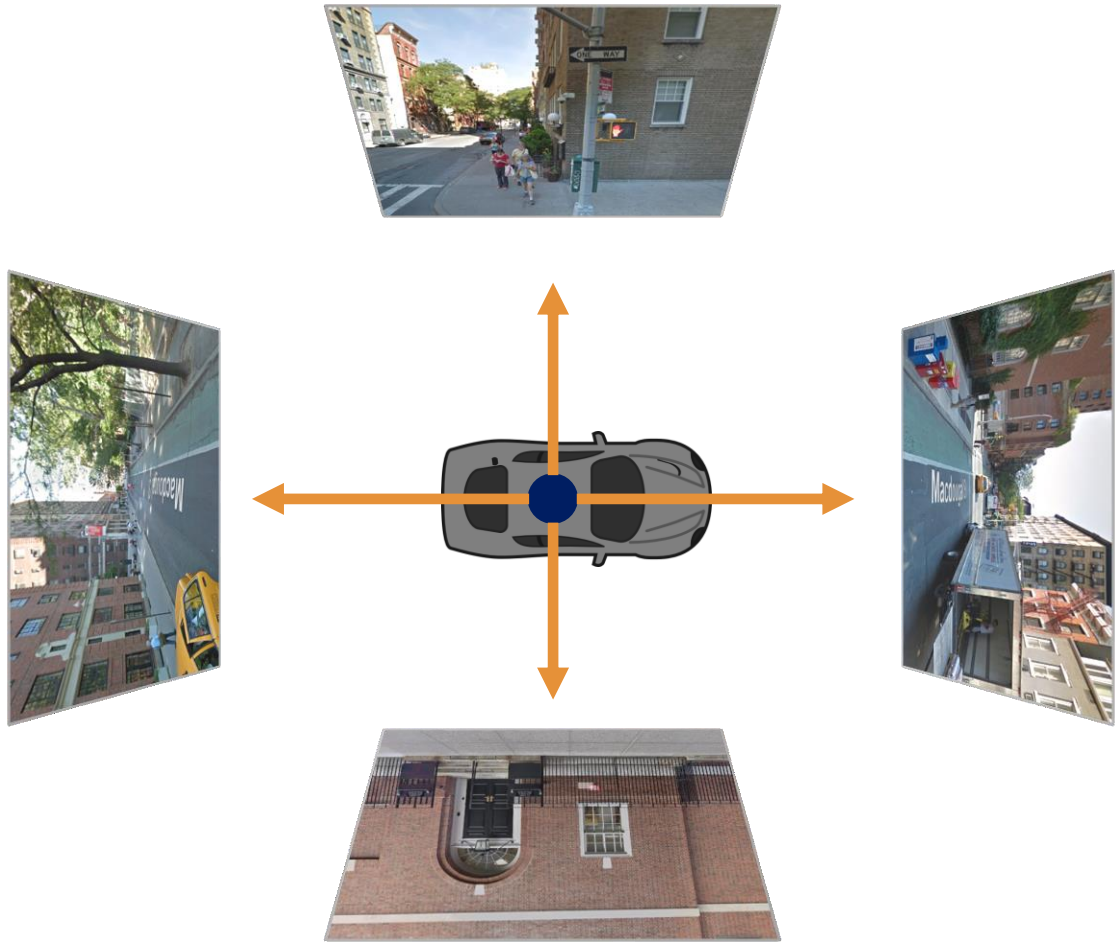
Visual comparison of geographically distant areas

Temporal analysis of unfolding urban developments

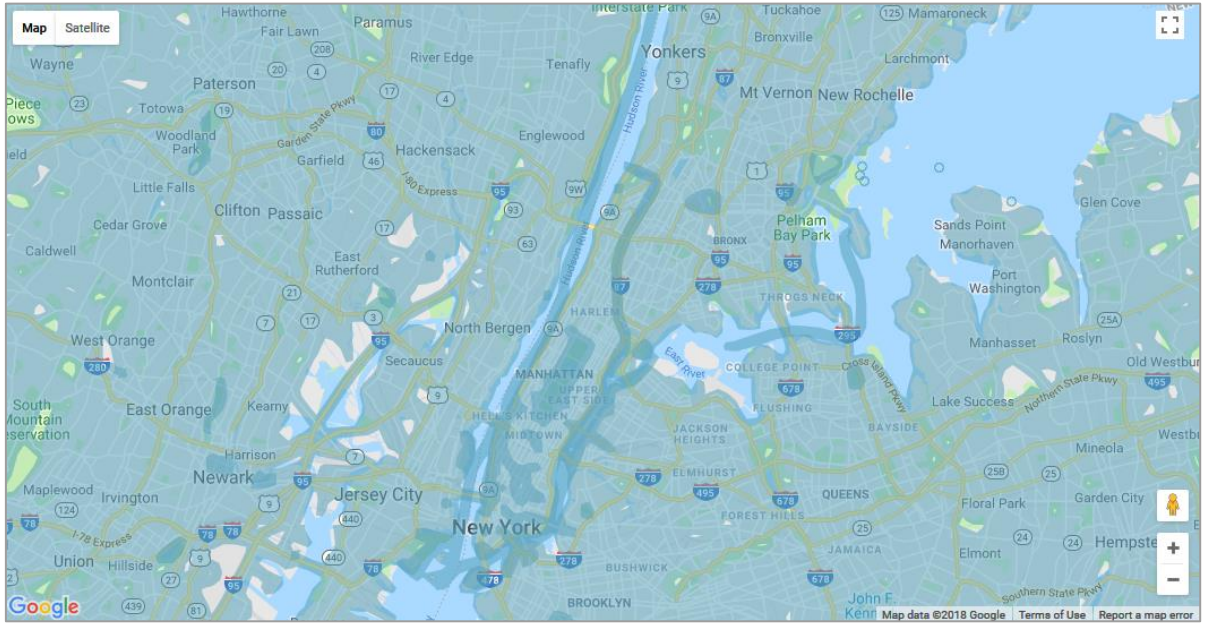


Street-level Images





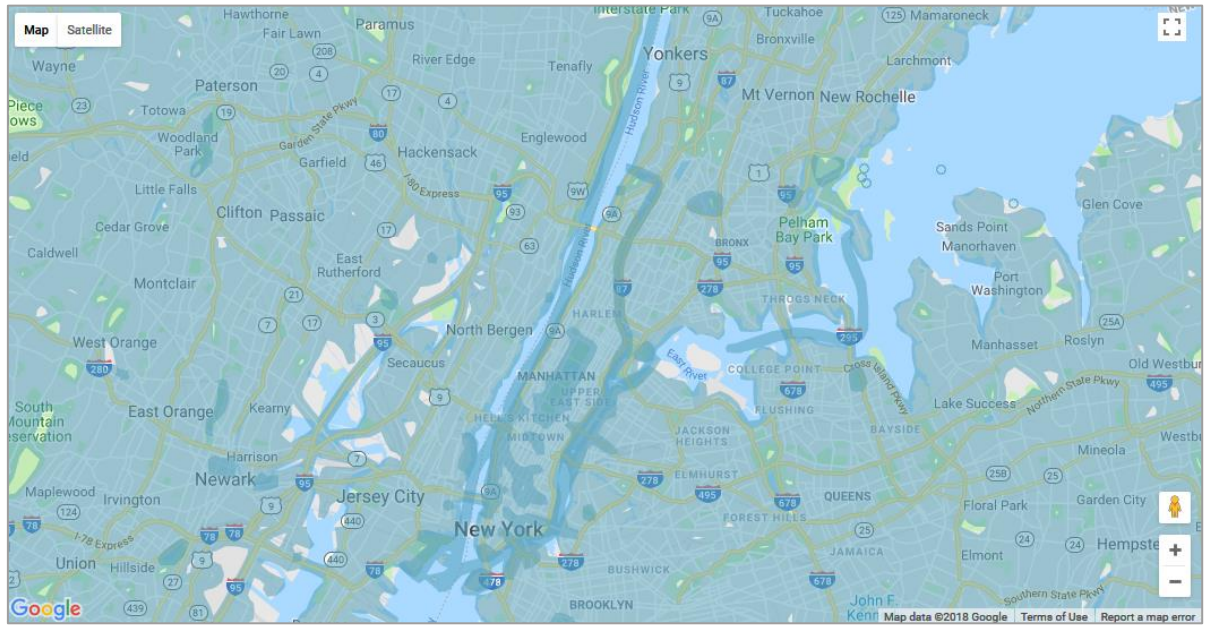
Google Street View



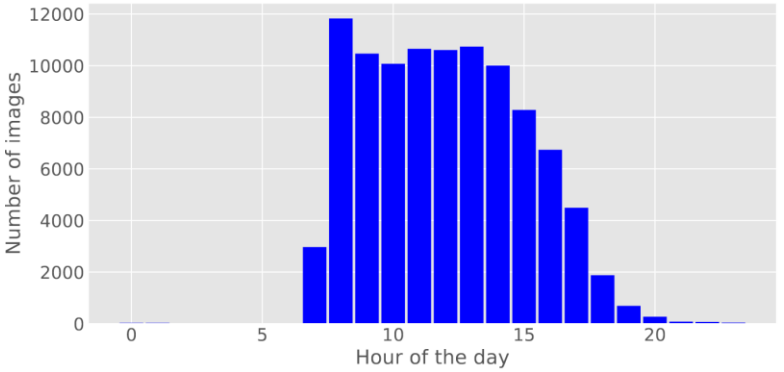
Spatially dense

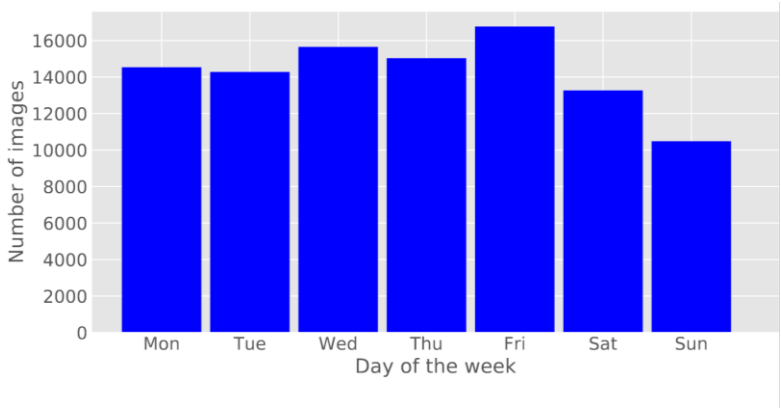
Temporally sparse

Temporally-dense street-level images



Spatially dense



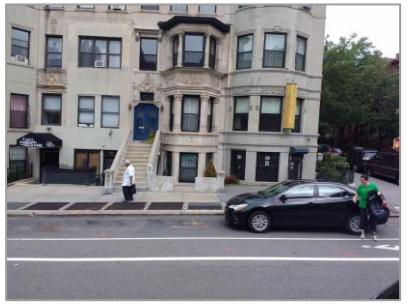


More than 7.7 million images
4 TB worth of image data

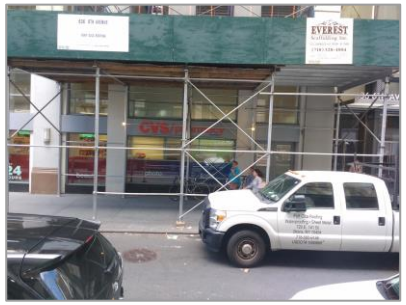
Temporally-dense street-level images

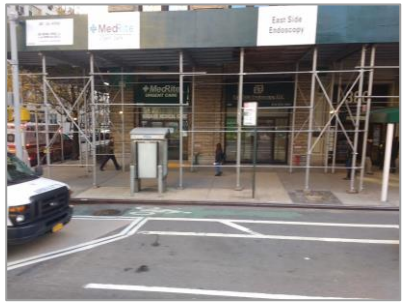
Objectives

Support the interactive analysis of the city at the micro scale, over geographically distant regions.



Comparison of the urban fabric in different regions

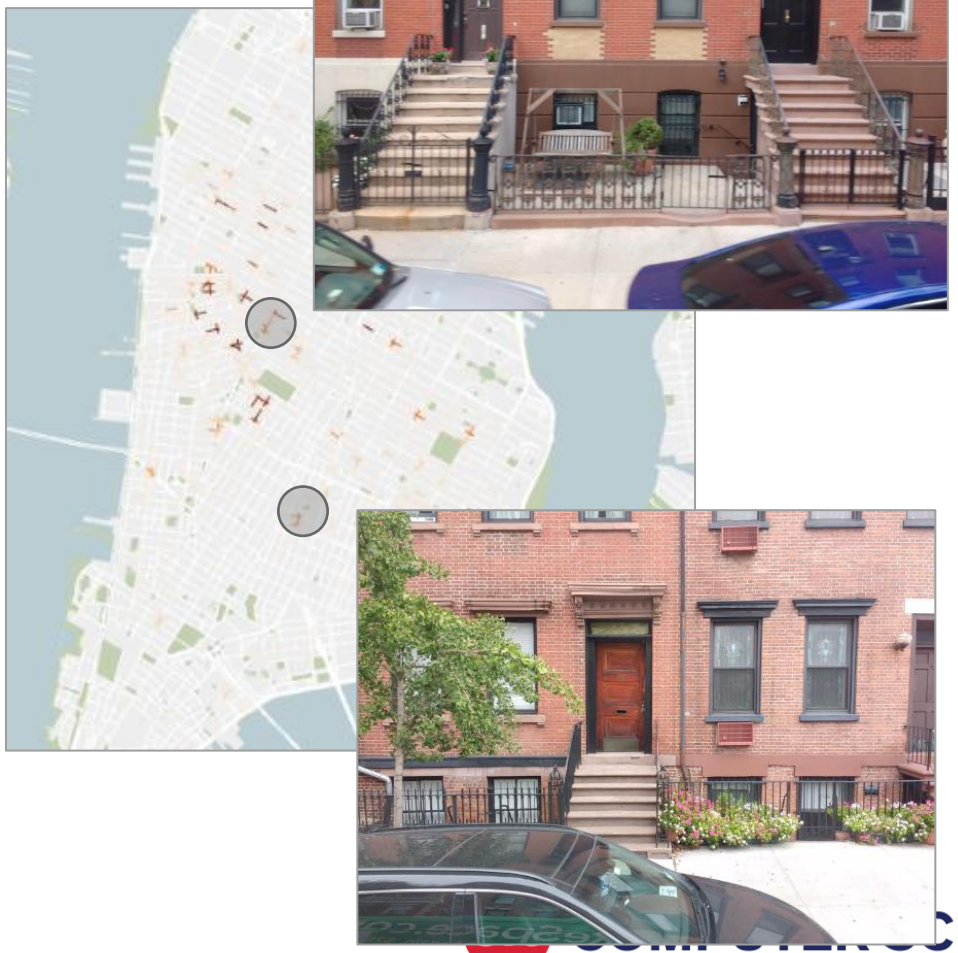


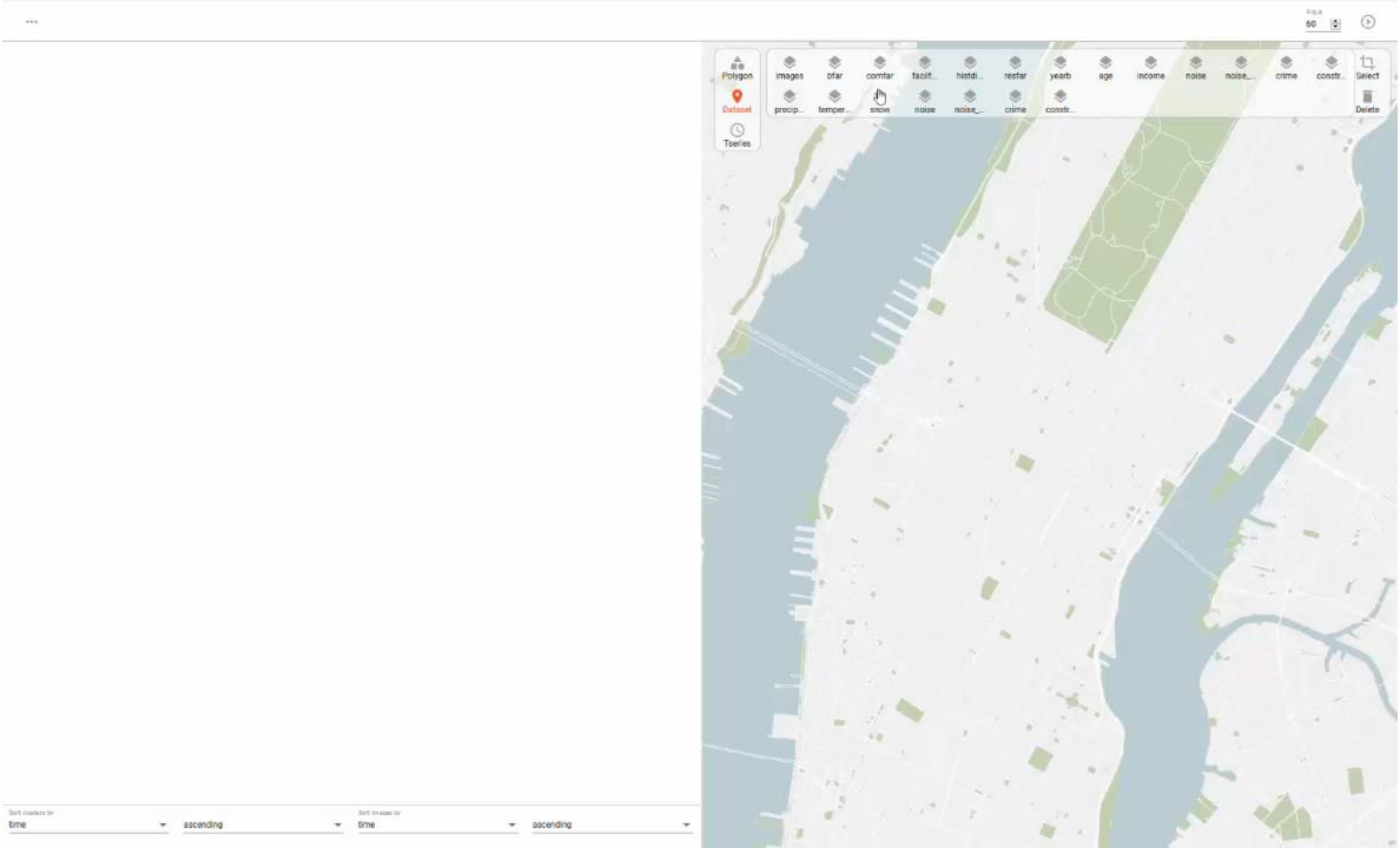


Assessment of features in the built environment

Assessment of walkability and accessibility

Image query composition




Polygon
Dataset
Tseries
images
bfar
comfar
facilit...
histdi...
resfar
yearb
age
income
noise
noise_...
crime
constr...
precip...
temper...
snow
noise
noise_...
crime
constr...
Select
Delete
60
time
ascending
time
ascending

Use cases

- Provided Urban Mosaic to domain experts:
  - Architects from Draw Brooklyn
    - Urban planning
    - Retrofitting
  - Occupational therapist
    - Accessibility and walkability older adults

Accessibility for older adults

Focuses on interventions that enable older adults to “Age in Place” and prevent outdoor fall

Impact of the neighborhood environment on falls

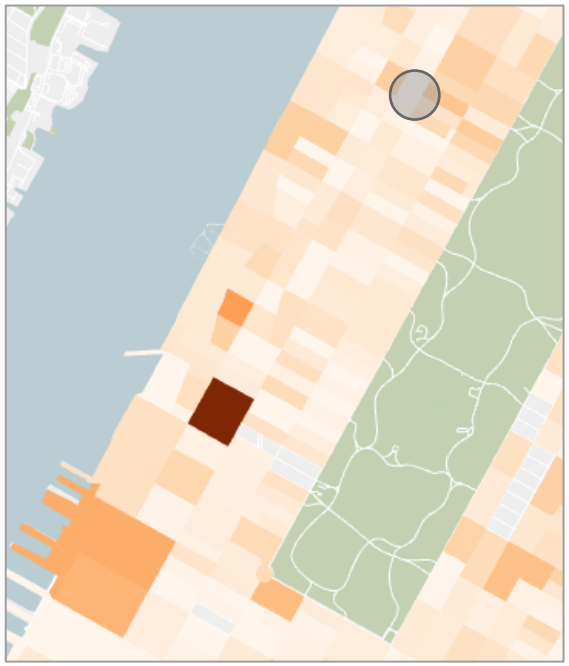
Accessibility: installation of tactile pavings

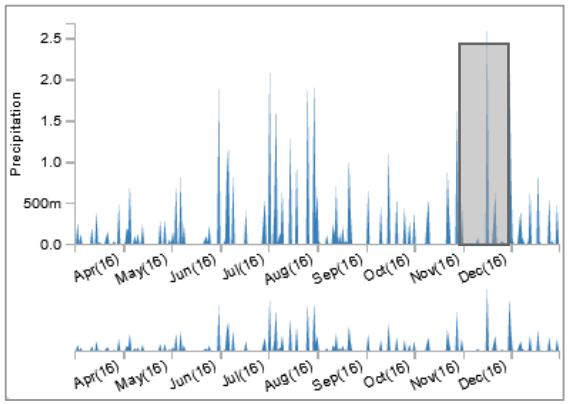
June

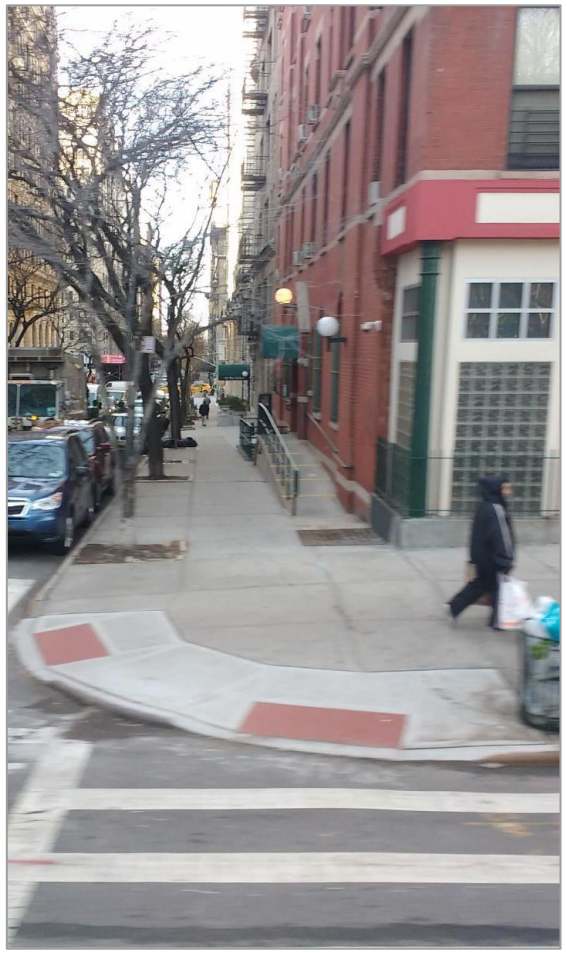
August

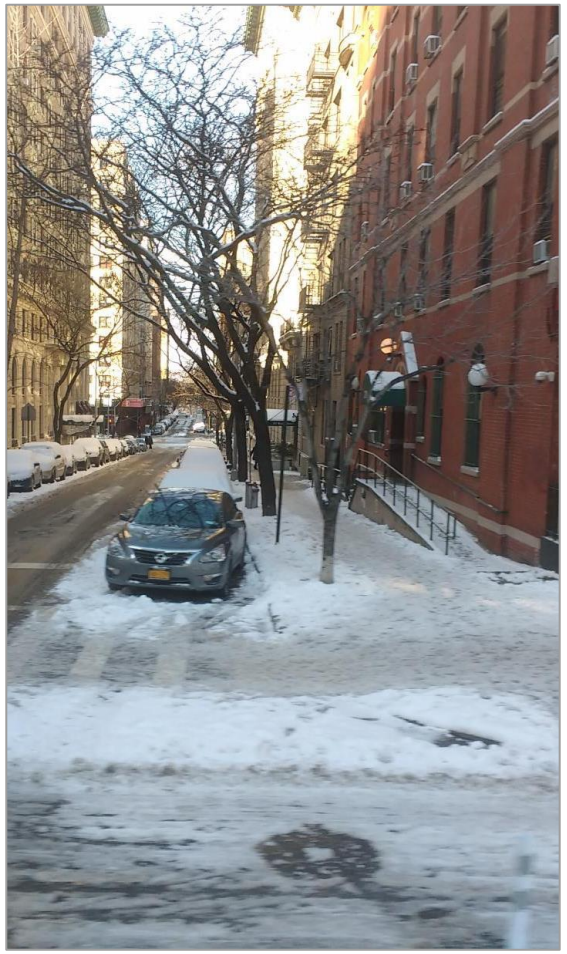
October

Accessibility: assessing hazards for older adults









Practitioners perspective

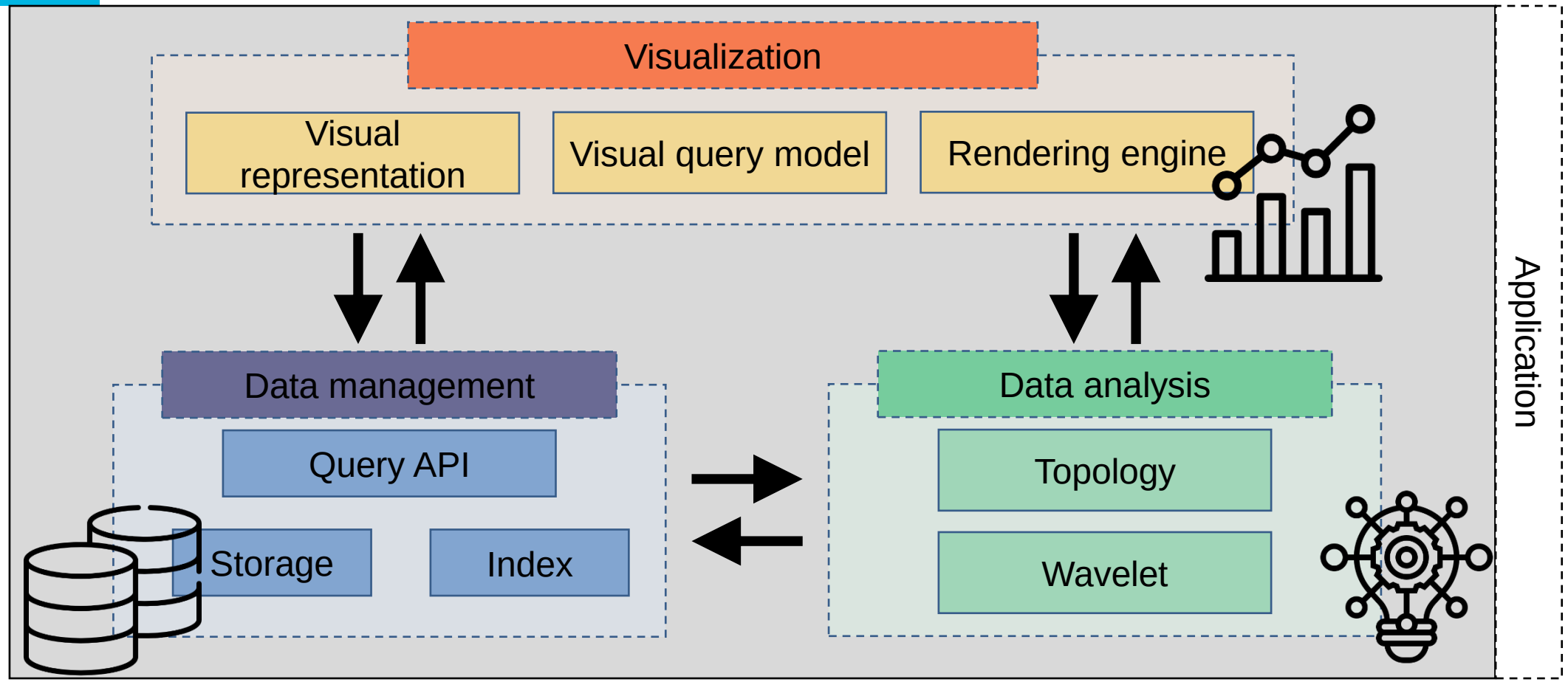
“What I like about the tool is that you can define what the main problem is, e.g. inclement weather or obstruction, and these are the conditions we’re are going to help the seniors identify and be safe around”.

Tracy Chippendale, Occupational Therapist at NYU

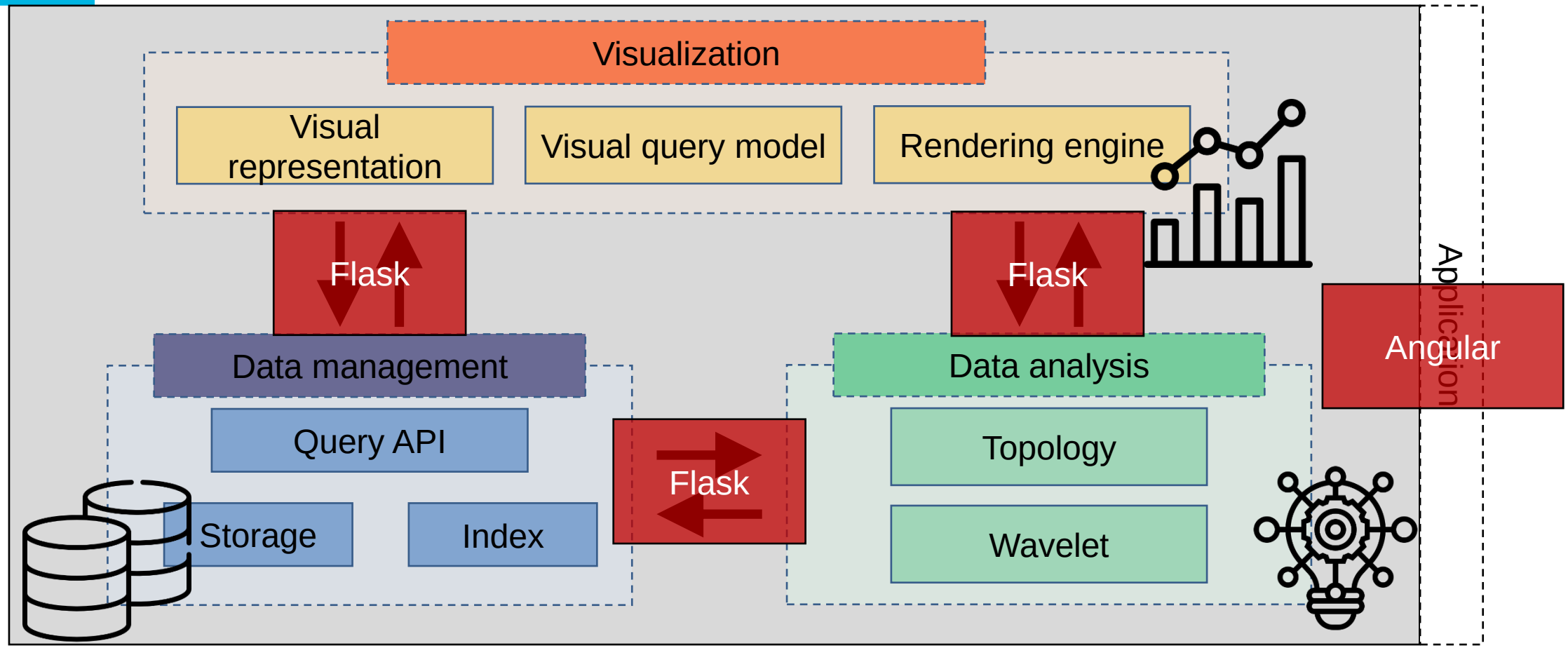
“Approaches like this can dramatically transform the way cities are planned and operated”.

Alexandros Washburn, former Chief Urban Designer of NYC

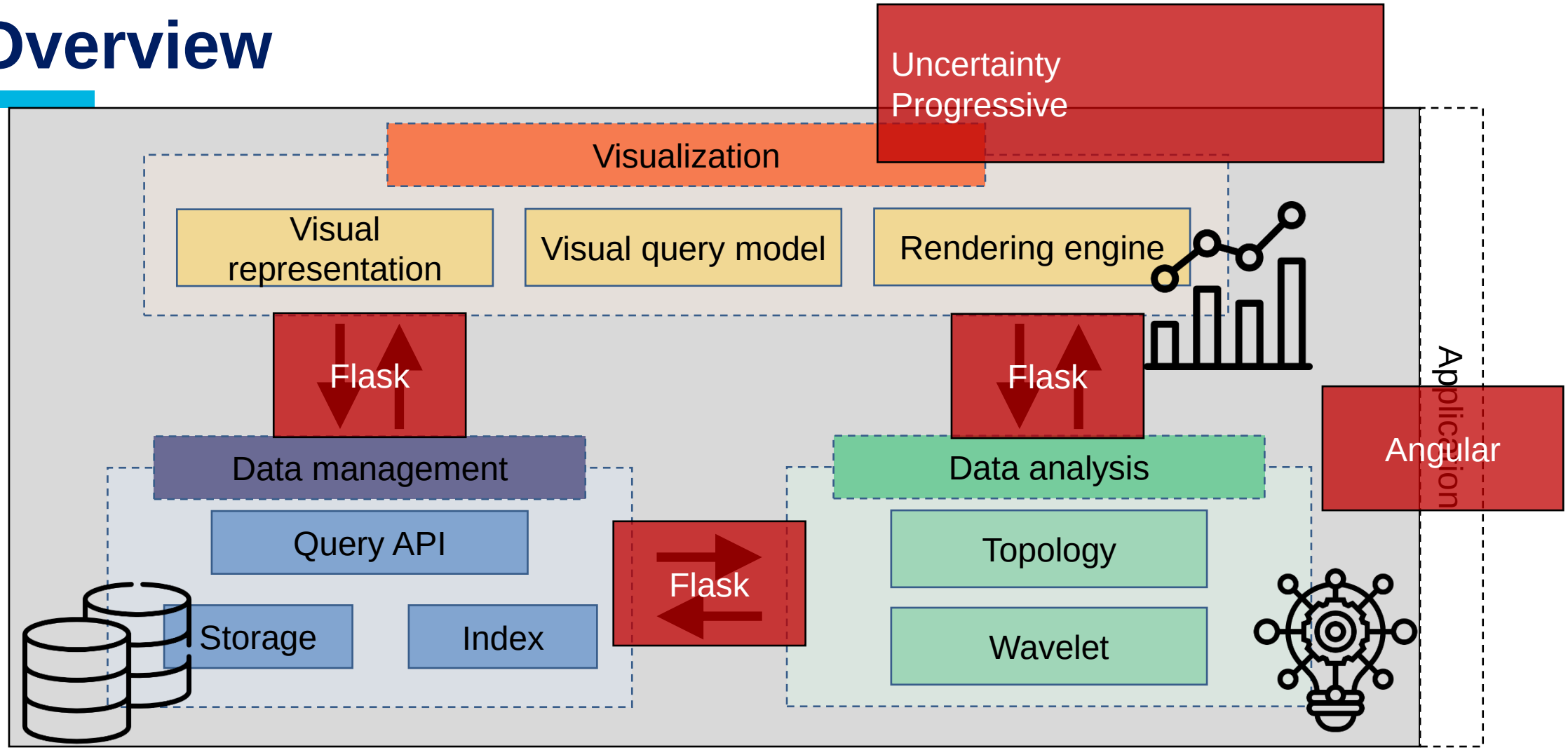
Overview


Visualization
Visual representation
Visual query model
Rendering engine
Application
Data management
Query API
Storage
Index
Data analysis
Topology
Wavelet

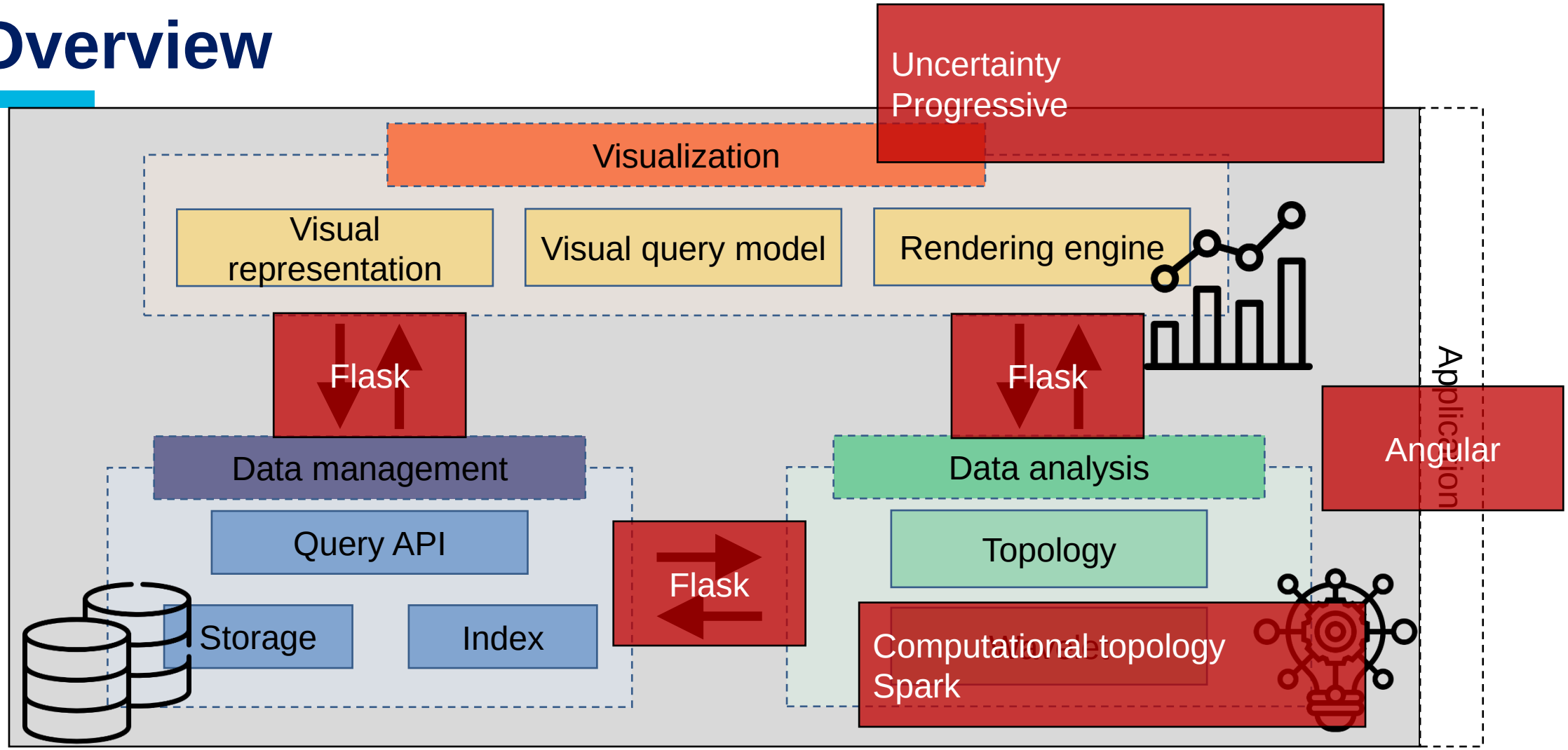
Overview


Visualization
Visual representation
Visual query model
Rendering engine
Flask
Flask
Application
Angular
Data management
Data analysis
Query API
Topology
Flask
Storage
Index
Wavelet

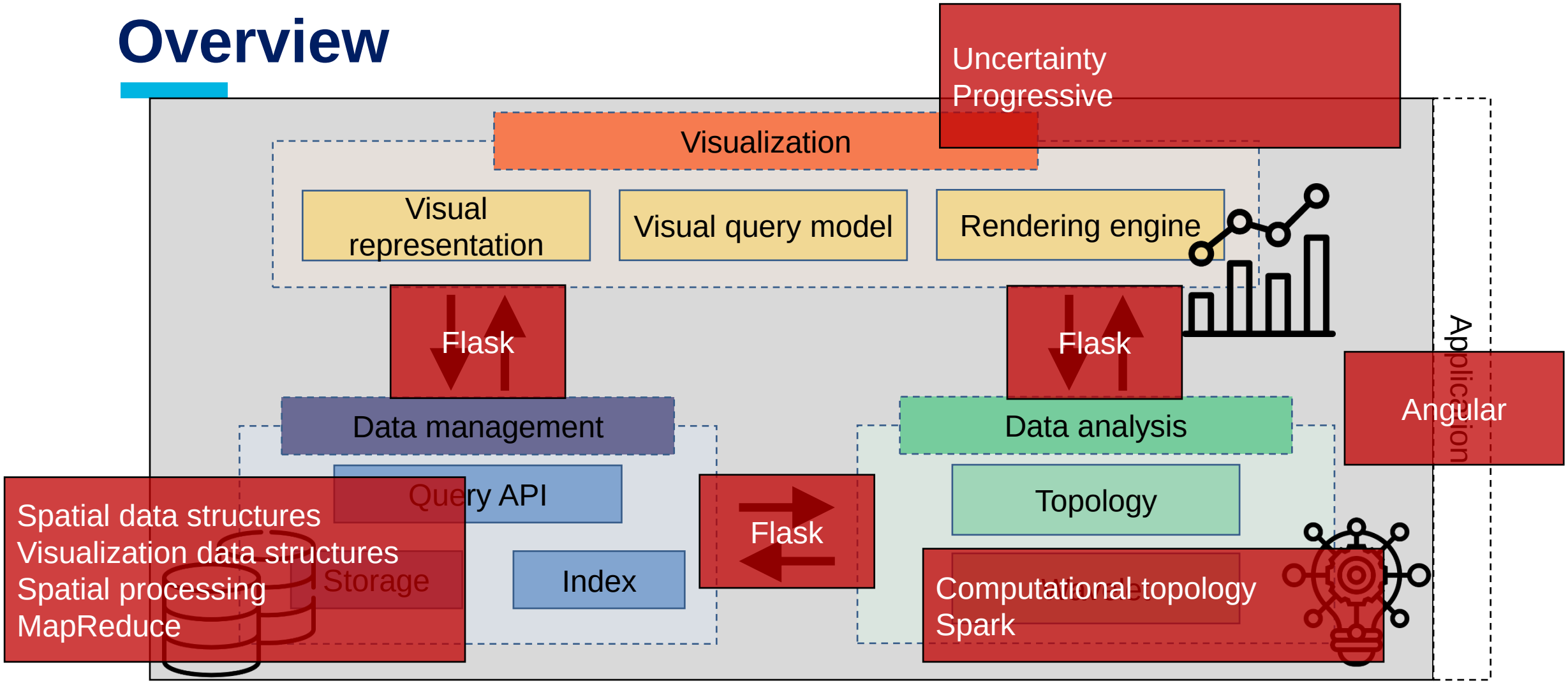
Overview


Uncertainty
Progressive
Visualization
Visual representation
Visual query model
Rendering engine
Flask
Flask
Application
Angular
Data management
Data analysis
Query API
Topology
Flask
Storage
Index
Wavelet

Overview


Uncertainty
Progressive
Visualization
Visual representation
Visual query model
Rendering engine
Flask
Flask
Application
Angular
Data management
Data analysis
Query API
Topology
Flask
Storage
Index
Computational topology
Spark

Overview


Uncertainty
Progressive
Visualization
Visual representation
Visual query model
Rendering engine
Flask
Flask
Data management
Data analysis
Application
Angular
Query API
Topology
Flask
Spatial data structures
Visualization data structures
Spatial processing
MapReduce
Storage
Index
Computational topology
Spark