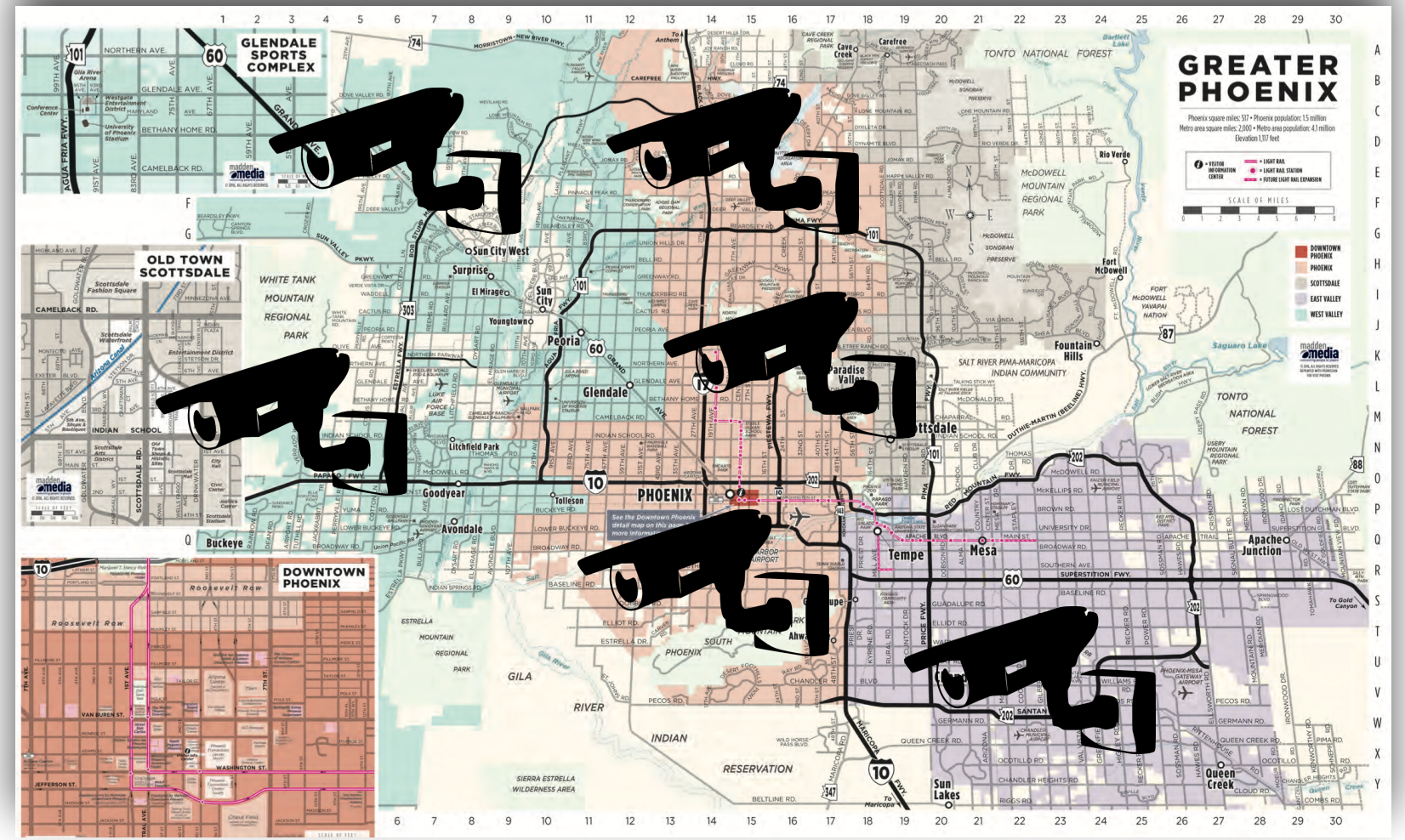
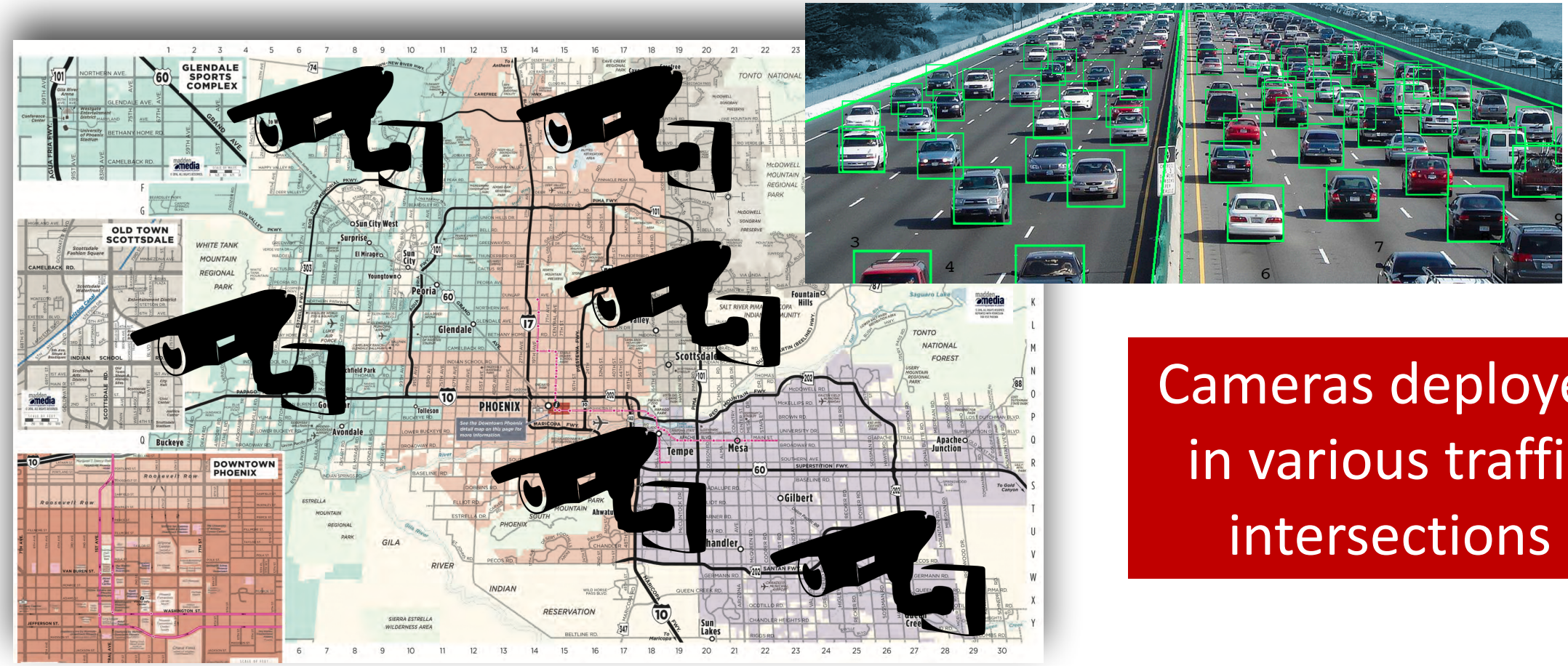
IoT Data Management


GLENDALE SPORTS COMPLEX
OLD TOWN SCOTTSDALE
DOWNTOWN PHOENIX
GREATER PHOENIX
Phoenix square miles: 517 • Phoenix population: 1.5 million
Metro area square miles: 2,000 • Metro area population: 4.1 million
Elevation 1,117 feet
DOWNTOWN PHOENIX
PHOENIX
SCOTTSDALE
EAST VALLEY
WEST VALLEY
TONTO NATIONAL FOREST
McDOWELL MOUNTAIN REGIONAL PARK
WHITE TANK MOUNTAIN REGIONAL PARK
SALT RIVER PIMA-MARICOPA INDIAN COMMUNITY
FORT McDOWELL YAVAPAI NATION
ESTRELLA MOUNTAIN REGIONAL PARK
GILA RIVER INDIAN RESERVATION
SIERRA ESTRELLA WILDERNESS AREA
Carefree
Cave Creek
Rio Verde
Fort McDowell
Fountain Hills
Saguaro Lake
Sun City West
Surprise
El Mirage
Sun City
Youngtown
Peoria
Glendale
Paradise Valley
Litchfield Park
Goodyear
Tolleson
Avondale
Buckeye
PHOENIX
Tempe
Mesa
Apache Junction
Queen Creek
Sun Lakes
Ahwatukee
To Gold Canyon
To Anthem
To Maricopa

Lifecycle of IoT Data

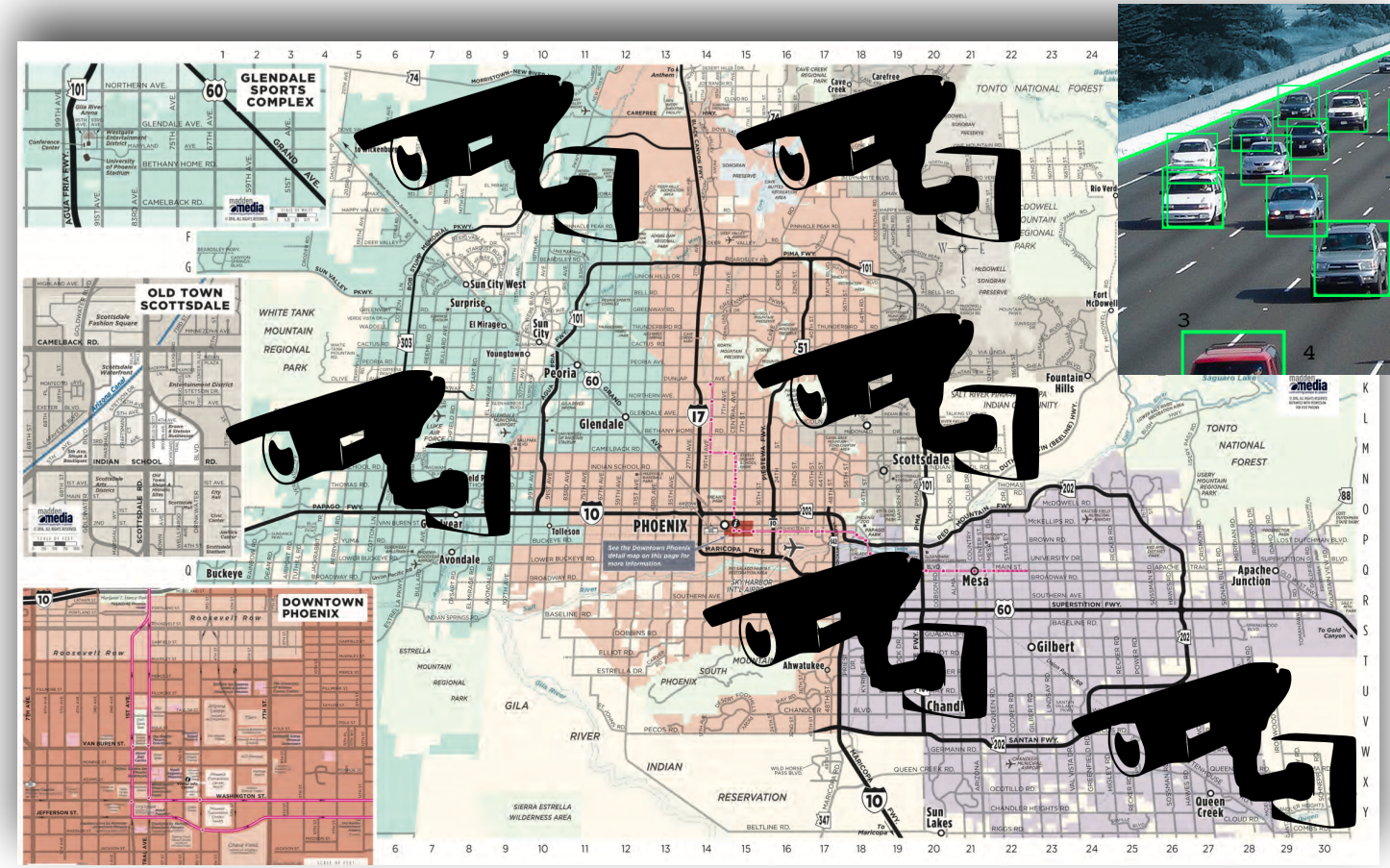
IoT Data

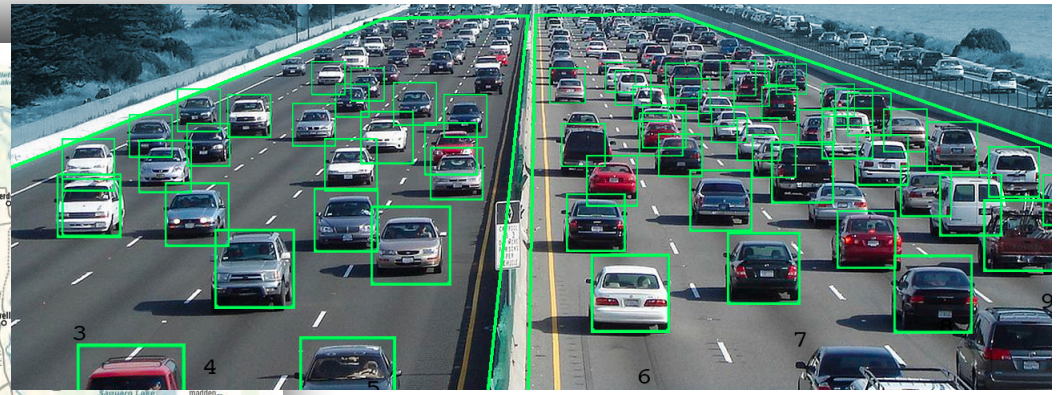
- Step I: Data Generated by various sensors (cameras)



IoT Data

- Step II: The data is first processed at the device (the edge)

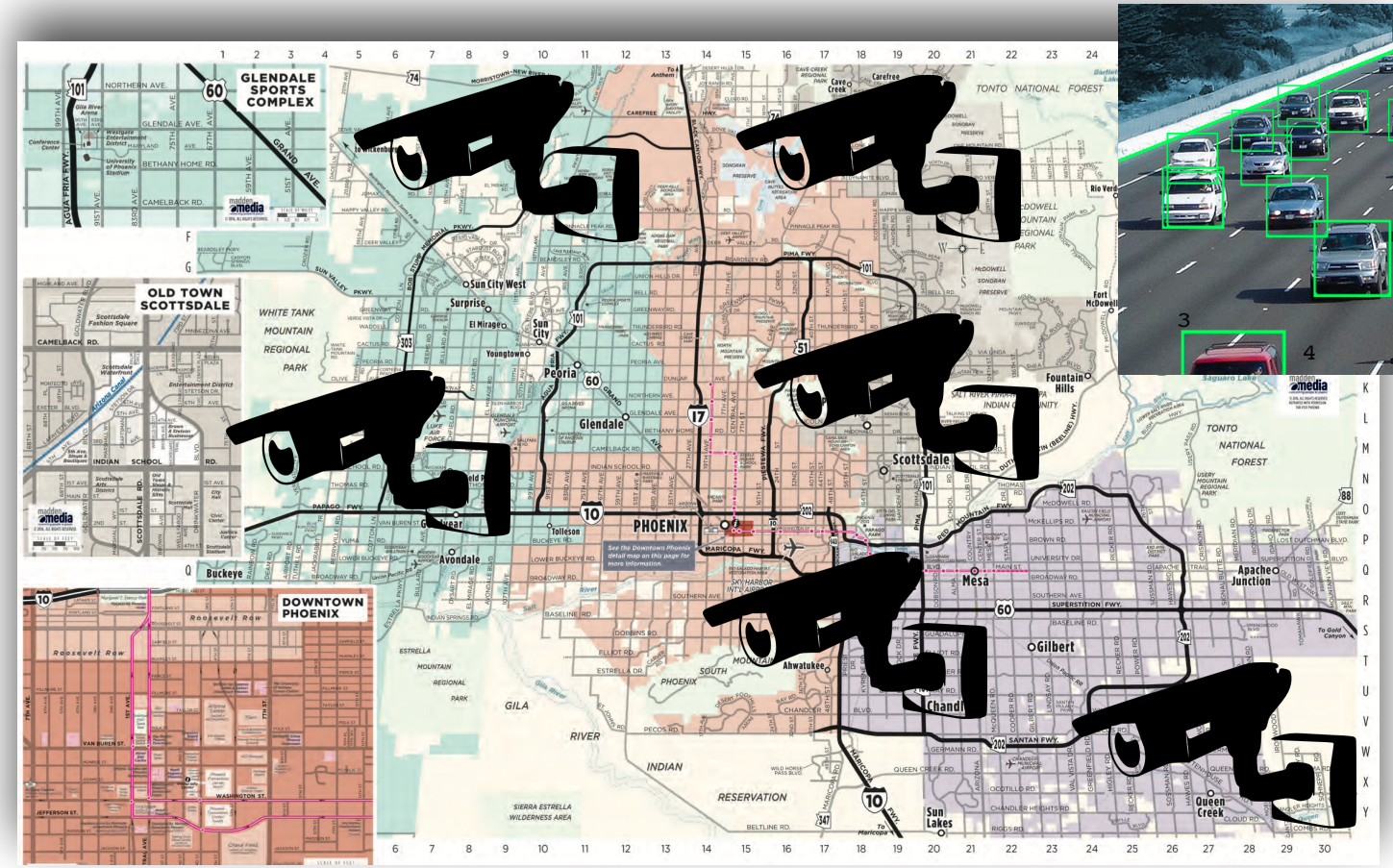


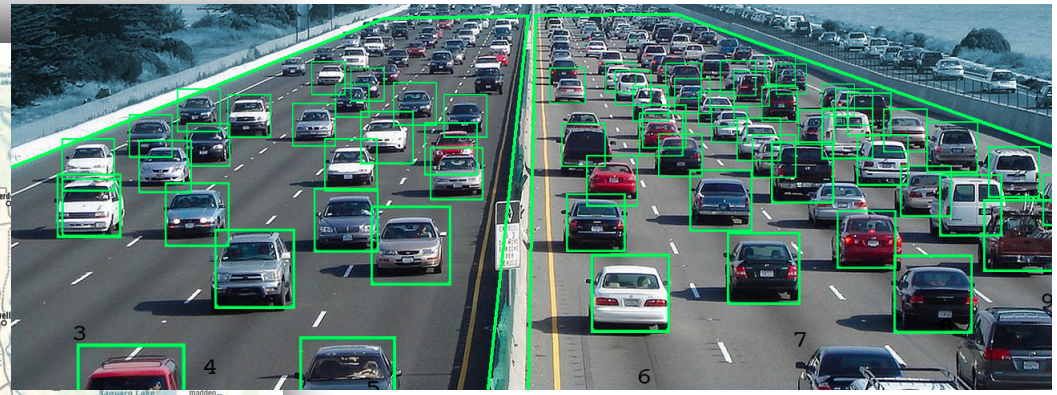


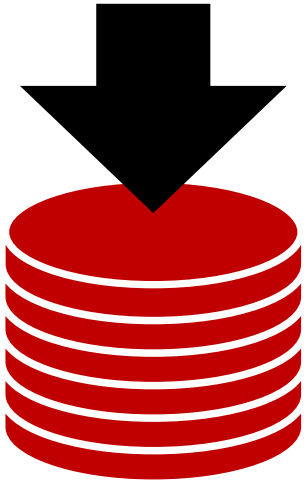
Camera sensors process data to Detect Objects (e.g, Cars)

IoT Data

- Step III: Processed data sent to the central system (i.e., the cloud)

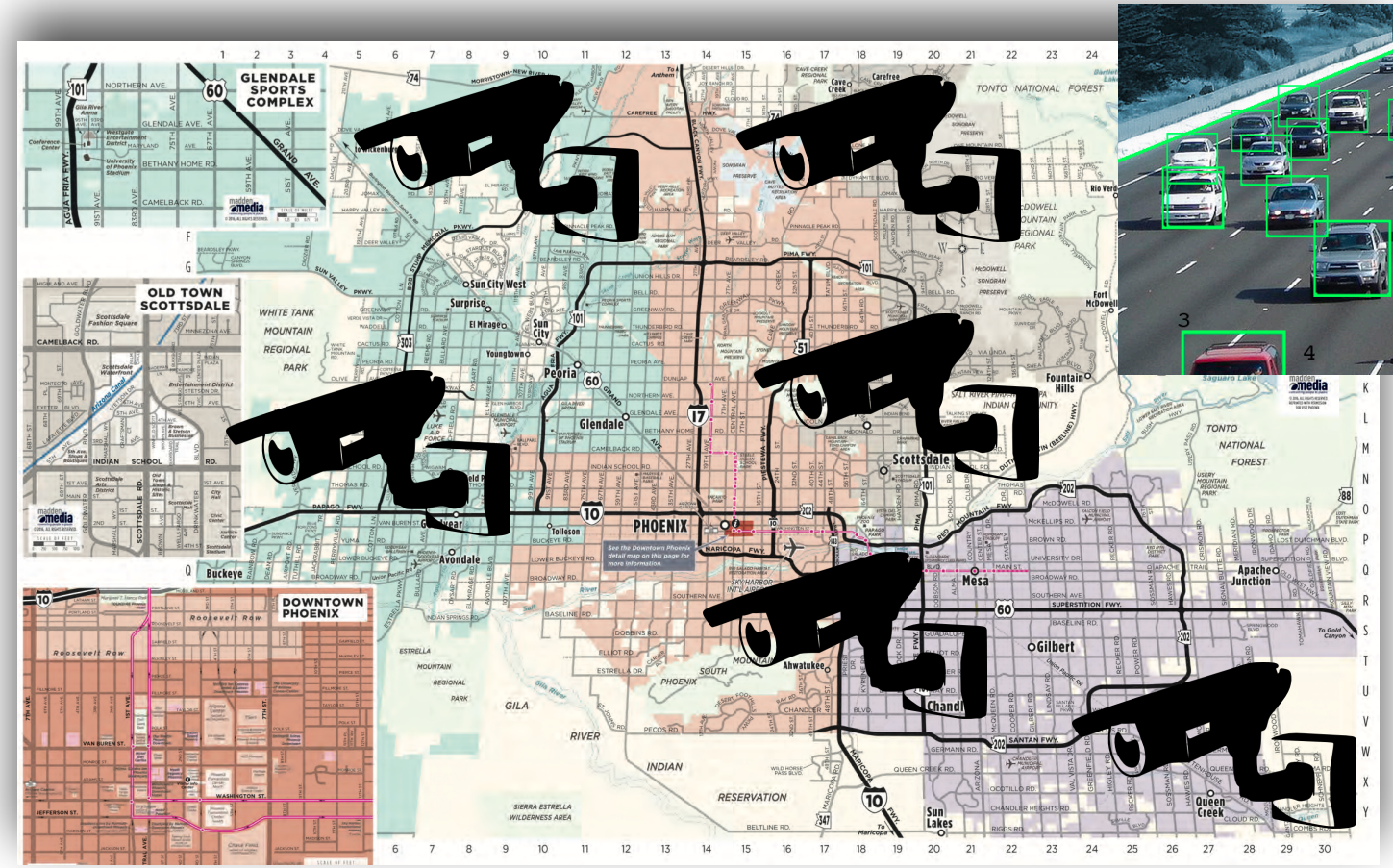


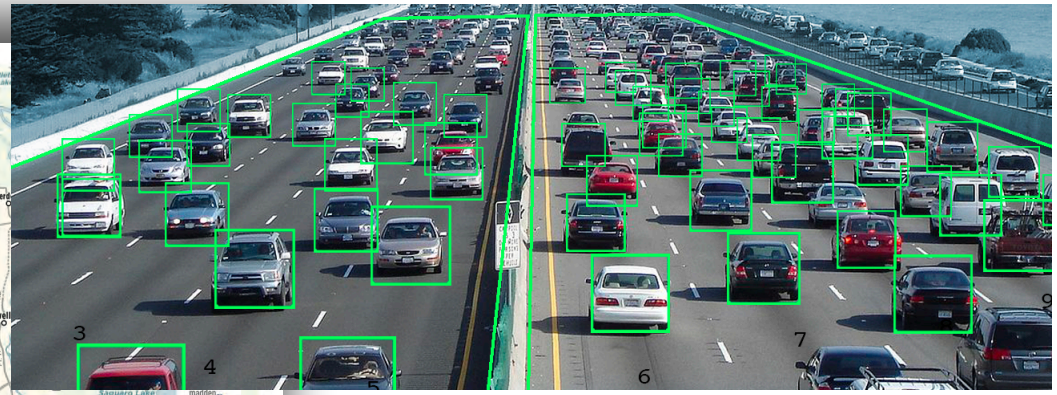


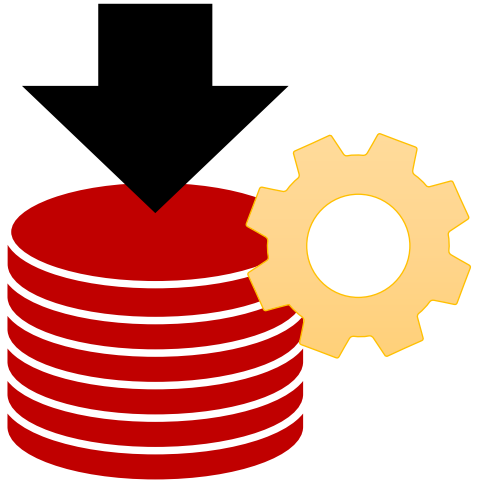


IoT Data

- Step IV: Further processing can happen at the central system



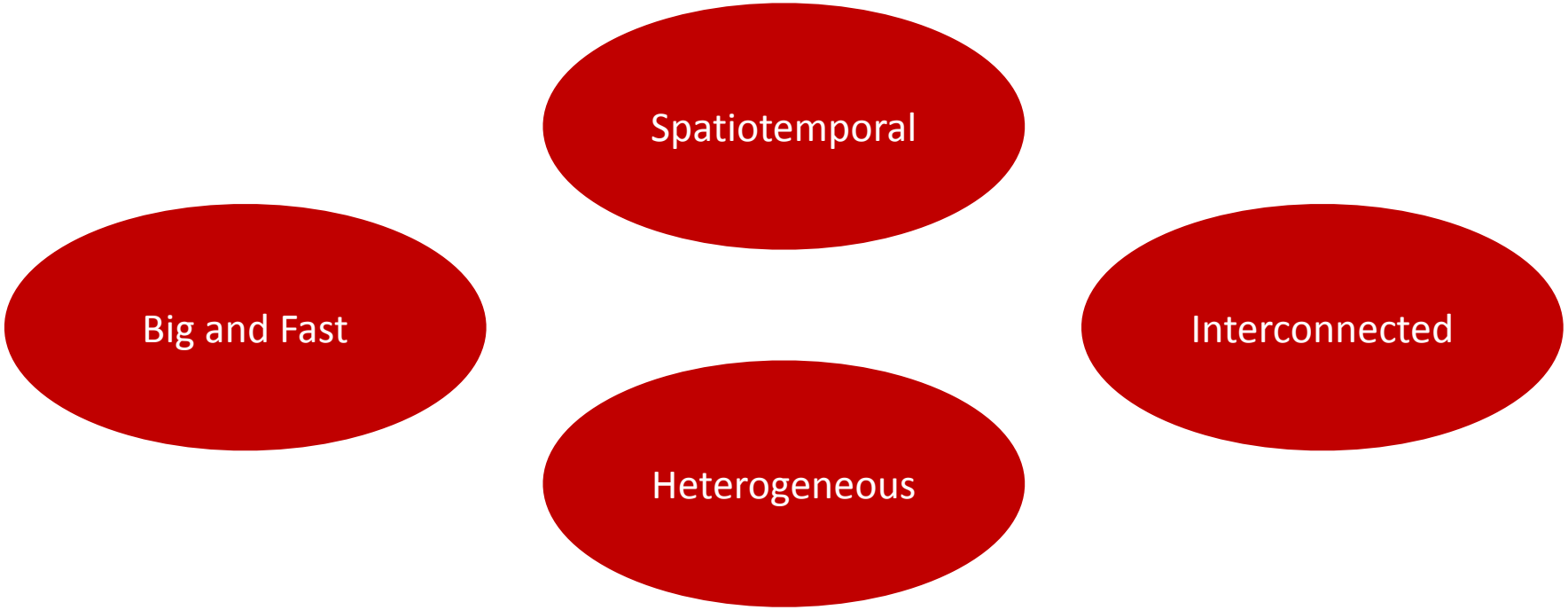


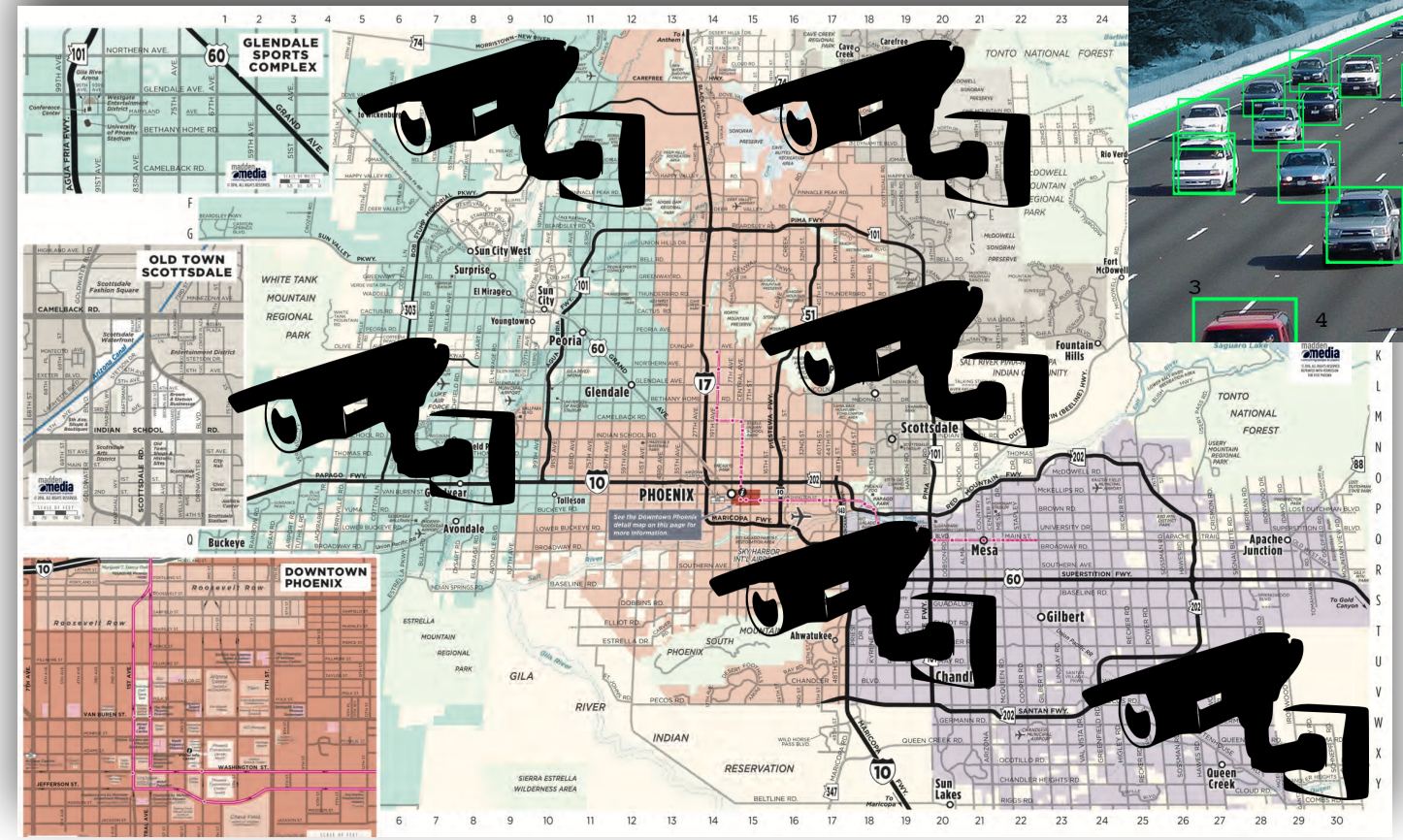


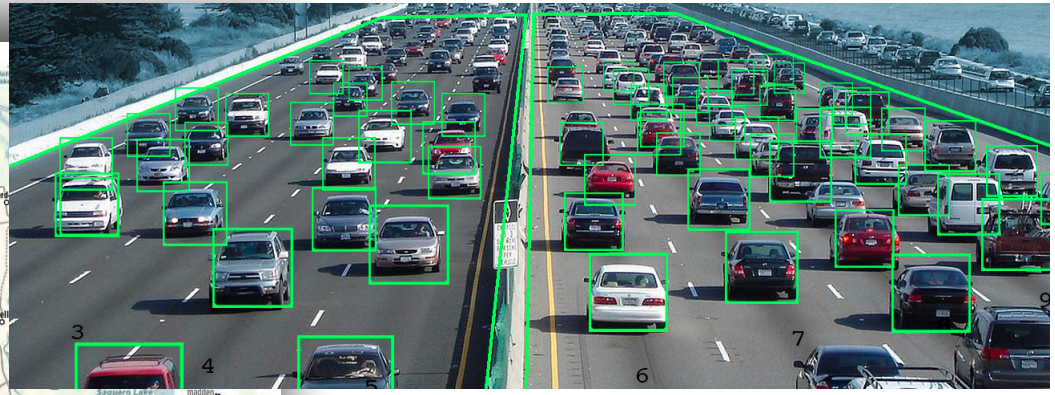
Where to Process IoT Data

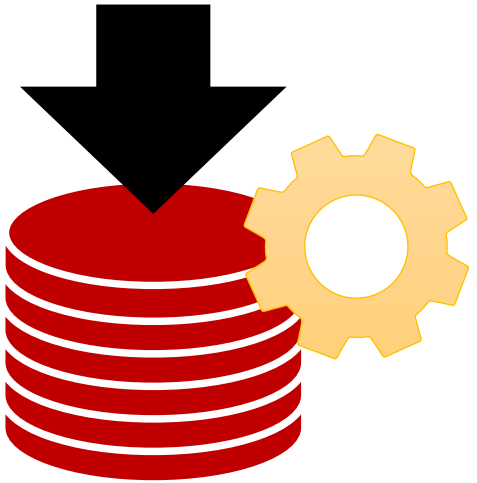
Cloud Vs. Edge

IoT Data: Properties









ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…		
2	Motorcycle		…		
…	…		…	…	
…	…		…	…	

IoT Data: Spatial and Temporal

- Every piece of data has a geospatial location and a time stamp
  - A blue BMW Car spotted at the
    Mill Ave / University Drive intersection
    11:00 am

IoT Data: Spatial and Temporal

- Some of these things are mobile (e.g., Smart car)
  - Moving car (with camera sensors) detects other cars in different locations at different times
  - Moving person (with smart watch) records different heart rates at different locations and different times

IoT Data: Big and Fast

- Imagine that you have a 1000 cams
- Each cam detects only100 objects per second
- **That means the central system receives 100K detected objects every second**

IoT Data: Heterogeneous

- **Heterogeneity on steroids**
  - IoT devices have totally different HW and SW.
  - Shall we process the data on the device, edge/fog, or in the cloud?
  - Each device generate different physical measurements (e.g., Temperature, LiDAR, Humidity, GPS, Glass Break Sensor, Camera Sensor).
  - Each device sends data back to the central data system at different rates

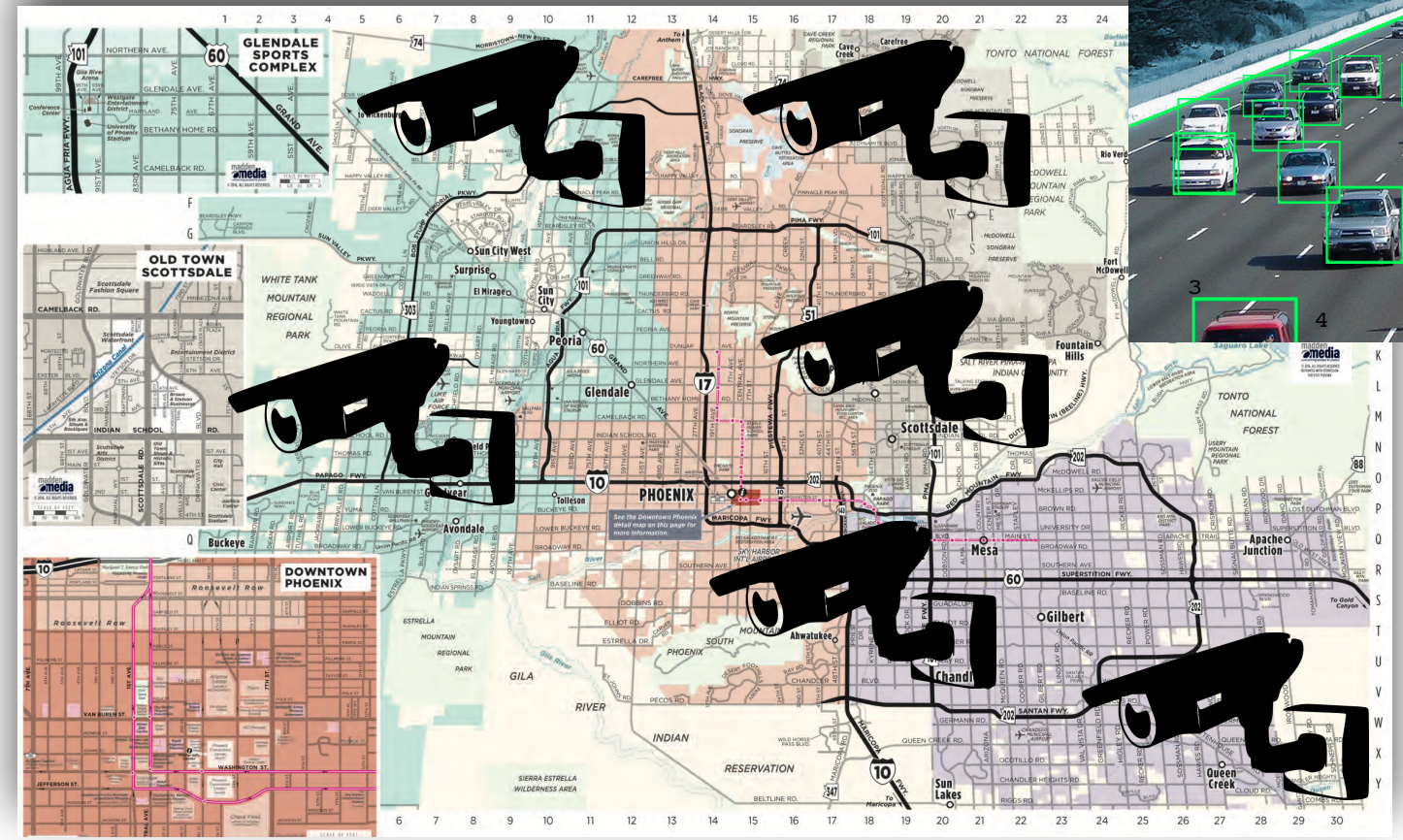
IoT Data: Interconnected

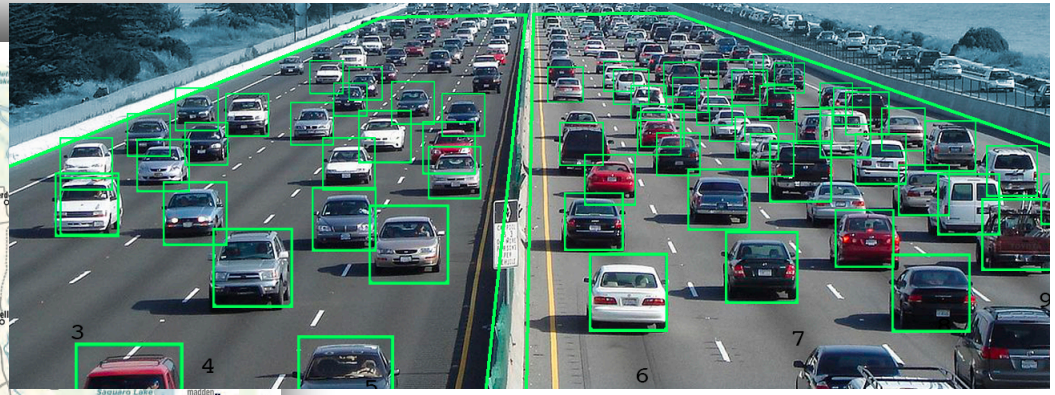
- The data collected from various sensors are related to each other
  - Traffic cams detect similar objects at different times
  - Data collected from various sensors can be fused to serve the application purposes

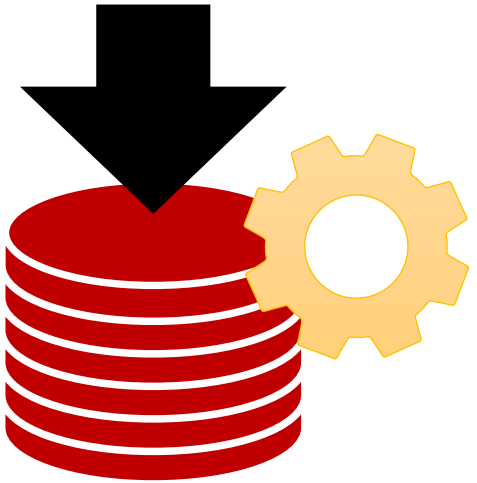
Agenda

- Storing IoT Data
- Handling the Geospatial Aspect of IoT Data
- Handling the Interconnected Aspect of IoT Data
- Handling the Big / Fast aspect of IoT Data

Storing IoT in data







ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…		
2	Motorcycle		…		
…	…		…	…	
…	…		…	…	

Storing IoT Data in a Relational Database

```
CREATE TABLE TrafficCams (
ObjectID INT,
ObjectType VARCHAR(20),
Properties VARCHAR(100),
…
Location VARCHAR(100)
Time Timestamp)
```

ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	Mill/Univ	12/1/2020 11:00 am
2	Motorcycle	Red	…	Riggs/Val	12/1/2020 11:30 am
…	…		…	…	
…	…		…	…	

Basic SQL Query

```
SELECT    [DISTINCT]  target-list
FROM      relation-list
WHERE     qualification
```

- relation-list A list of relation names (possibly with a range-variable after each name).
- target-list A list of attributes of relations in relation-list
- qualification Comparisons (Attr op const or Attr1 op Attr2, where op is one of **<, >, =, ≤, ≥, ≠**) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are not eliminated!

Querying IoT Data in a Relational Database

Find all detected car records

ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	Mill/Univ	12/1/2020 11:00 am
2	Motorcycle	Red	…	Riggs/Val	12/1/2020 11:30 am
…	…		…	…	
…	…		…	…	

```
SELECT *
FROM TrafficCams
WHERE ObjectType = 'Car'
```

Querying IoT Data in a Relational Database

Find all detected BMW car records

ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	Mill/Univ	12/1/2020 11:00 am
2	Motorcycle	Red	…	Riggs/Val	12/1/2020 11:30 am
…	…		…	…	
…	…		…	…	

```
SELECT *
FROM  TrafficCams
WHERE ObjectType = 'Car'
AND Properties LIKE %BMW%
```

Querying IoT Data in a Relational Database

Find the ObjectID of detected cars

ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	Mill/Univ	12/1/2020 11:00 am
2	Motorcycle	Red	…	Riggs/Val	12/1/2020 11:30 am
…	…		…	…	
…	…		…	…	

```
SELECT ObjectID
FROM TrafficCams
WHERE ObjectType = 'Car'
```

Querying IoT Data in a Relational Database

Count all detected car records

ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	Mill/Univ	12/1/2020 11:00 am
2	Motorcycle	Red	…	Riggs/Val	12/1/2020 11:30 am
…	…		…	…	
…	…		…	…	

```
SELECT Count(*)
FROM TrafficCams
WHERE ObjectType = 'Car'
```

Querying IoT Data in a Relational Database

Count all detected objets in every Object Type

ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	Mill/Univ	12/1/2020 11:00 am
2	Motorcycle	Red	…	Riggs/Val	12/1/2020 11:30 am
…	…		…	…	
…	…		…	…	

```
SELECT ObjectType, Count(*)
FROM TrafficCams
GROUP BY ObjectType
```

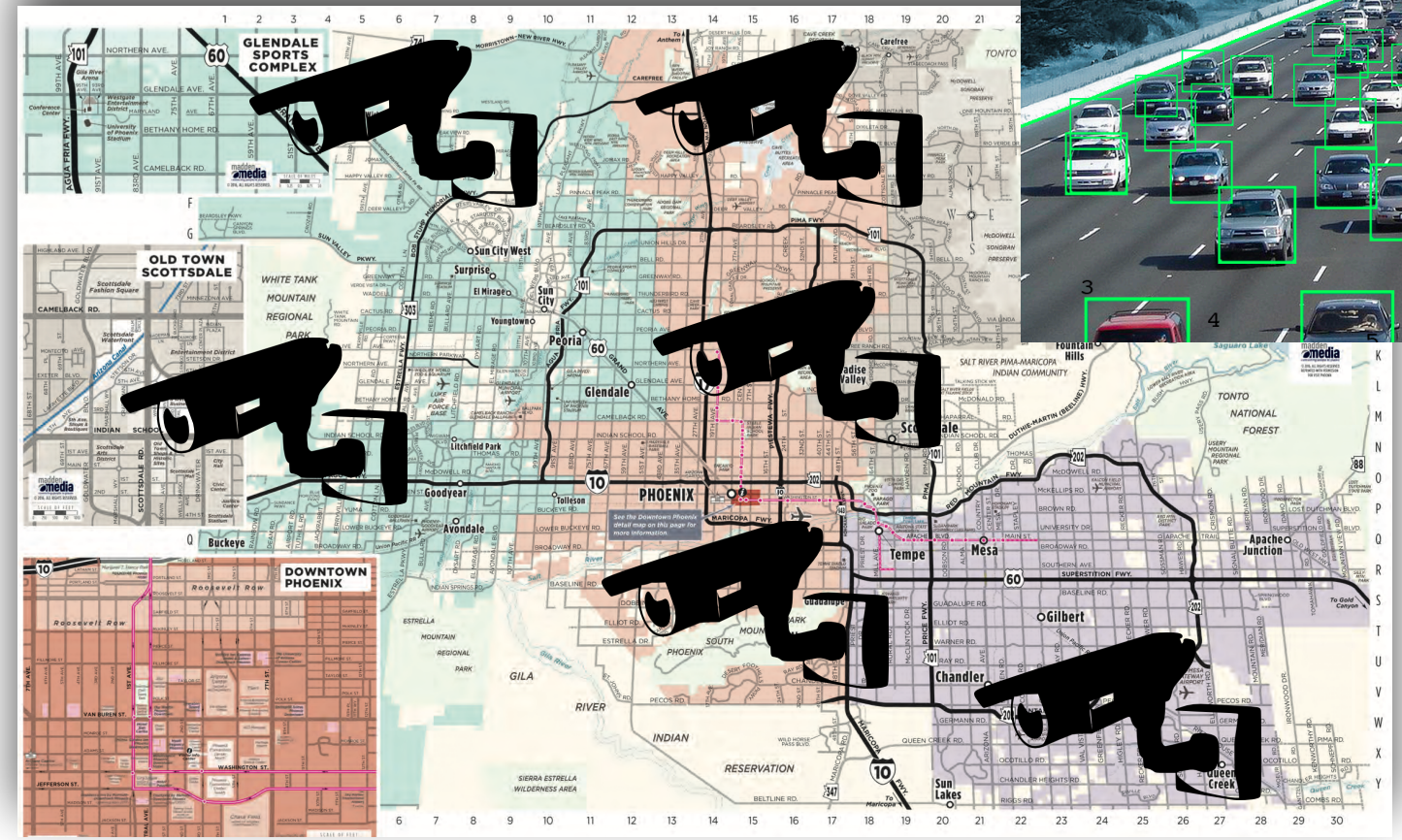
Inserting IoT Data in a Relational Database

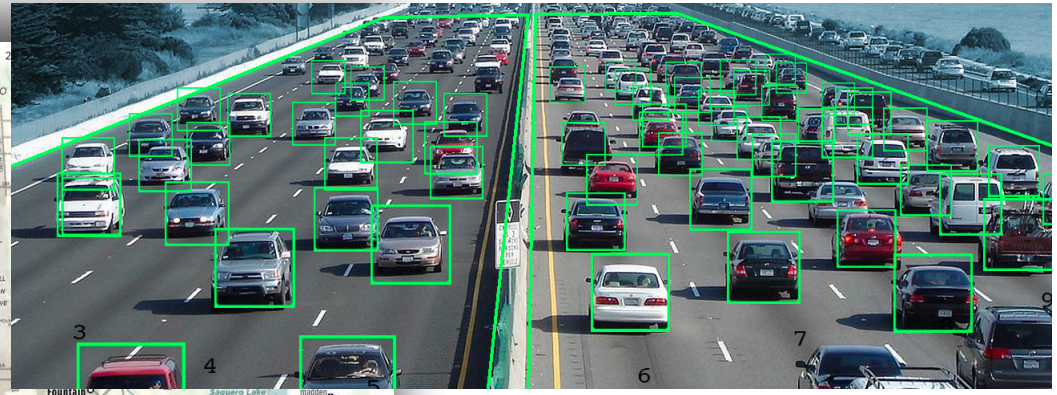
```
INSERT INTO TrafCams
VALUES (3, 'CAR', 'Red', …)
```

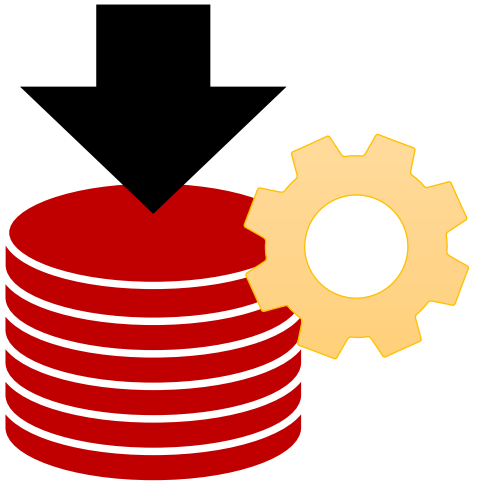
```
INSERT INTO TrafCams
VALUES (4, 'Truck', '18 Wheeler',
…)
```

ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	Mill/Univ	12/1/2020 11:00 am
2	Motorcycle	Red	…	Riggs/Val	12/1/2020 11:30 am
…	…		…	…	
…	…		…	…	

Handling the Geospatial aspect of IoT Data



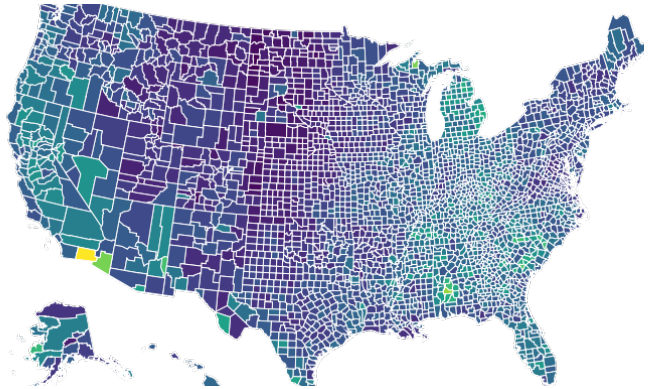




ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…		
2	Motorcycle		…		
…	…		…	…	
…	…		…	…	

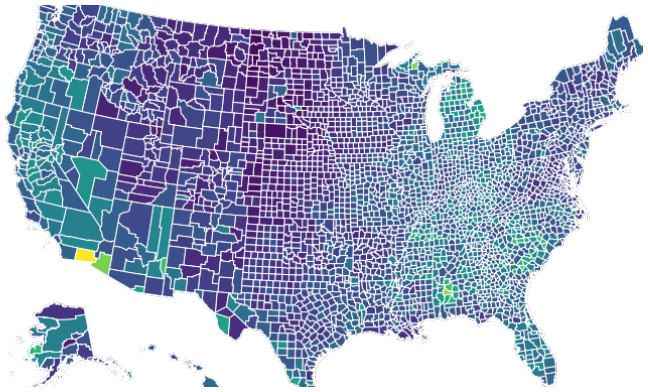
IoT Data: Spatial and Temporal

- Properties:
  - More than one dimension
  - Geometrical properties

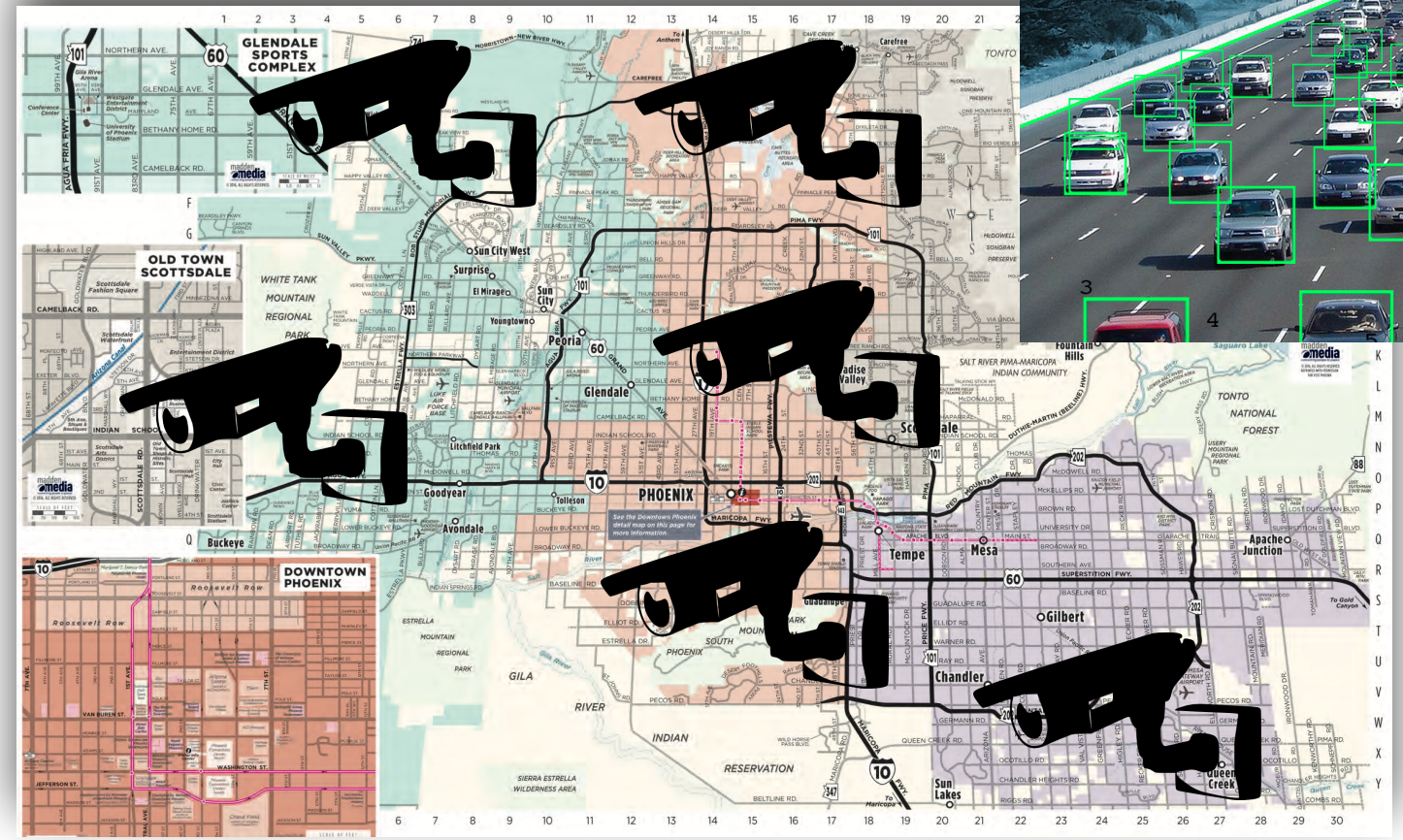


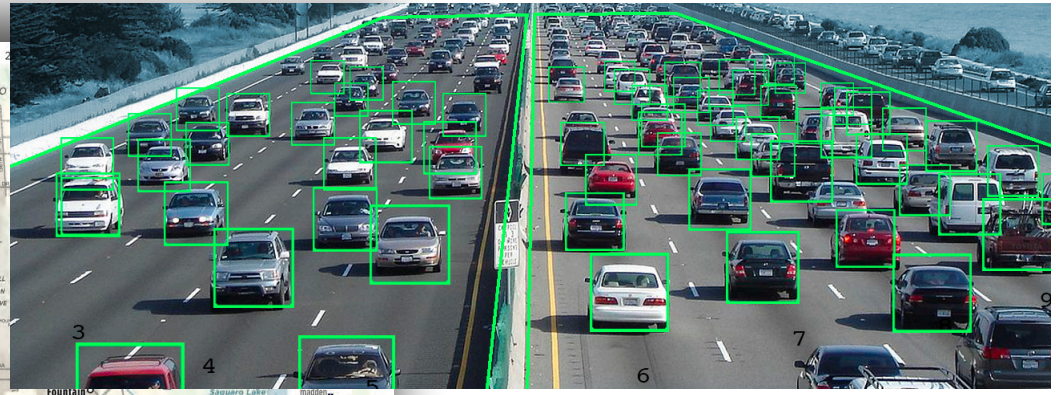
IoT Data: Spatial and Temporal

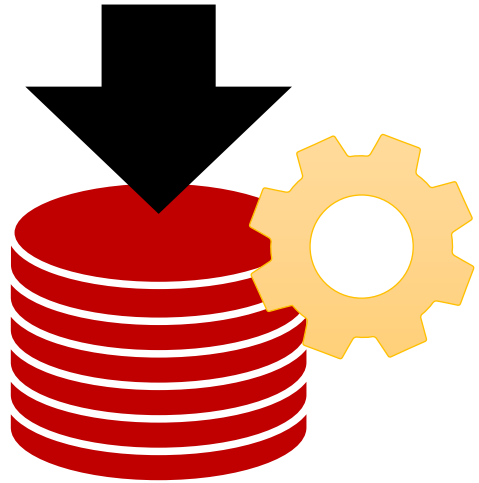
- Properties:
  - More than one dimension
  - Geometrical properties



- **Both Data Intensive and Compute Intensive**
- **Sorting is difficult**

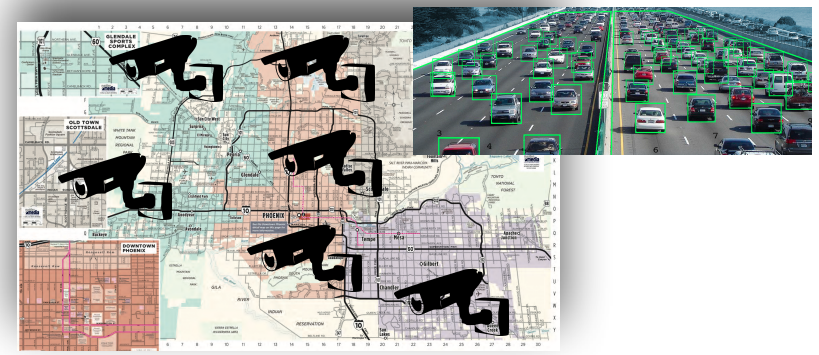






Report the count of **Cars Detected**

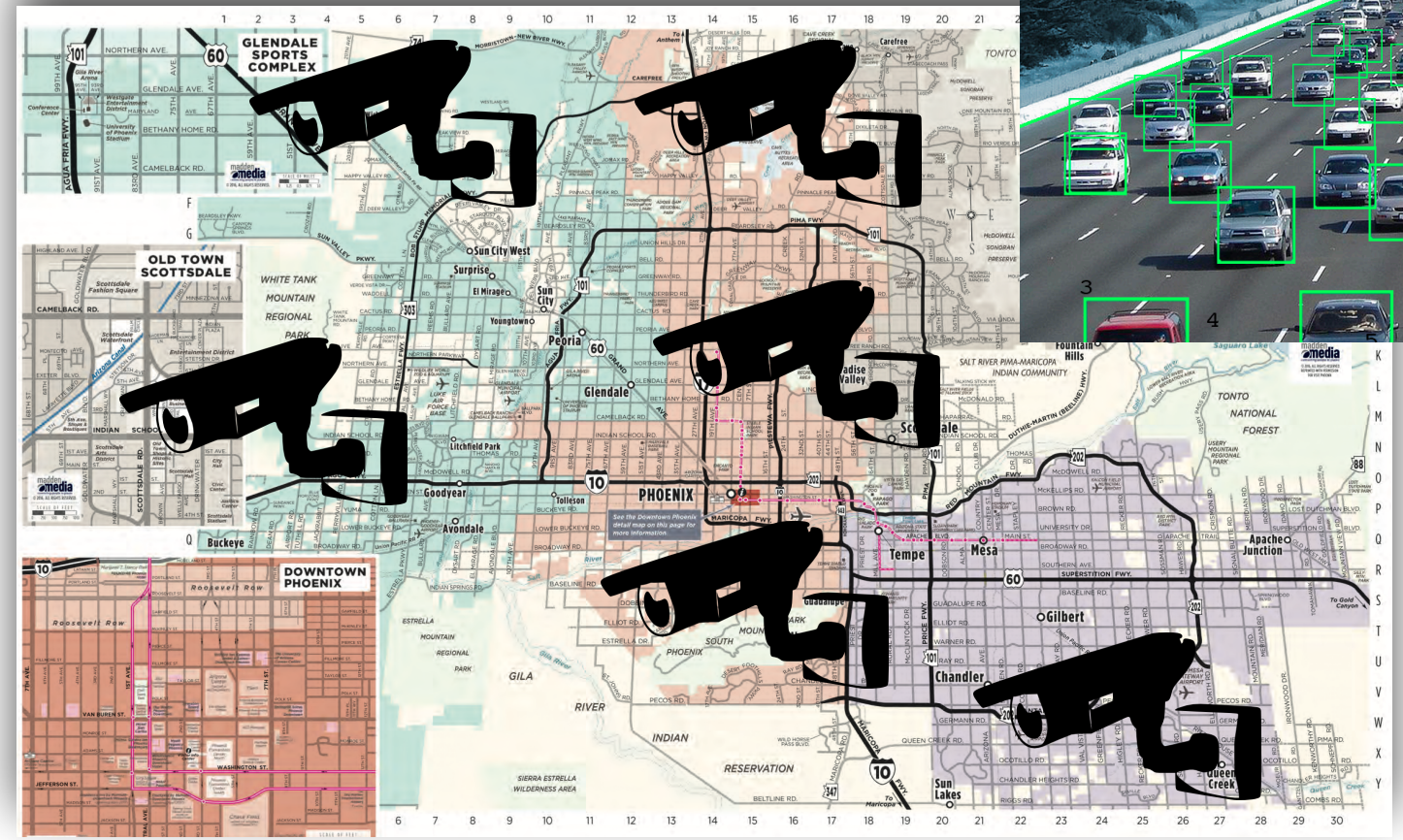
in the city of Tempe area

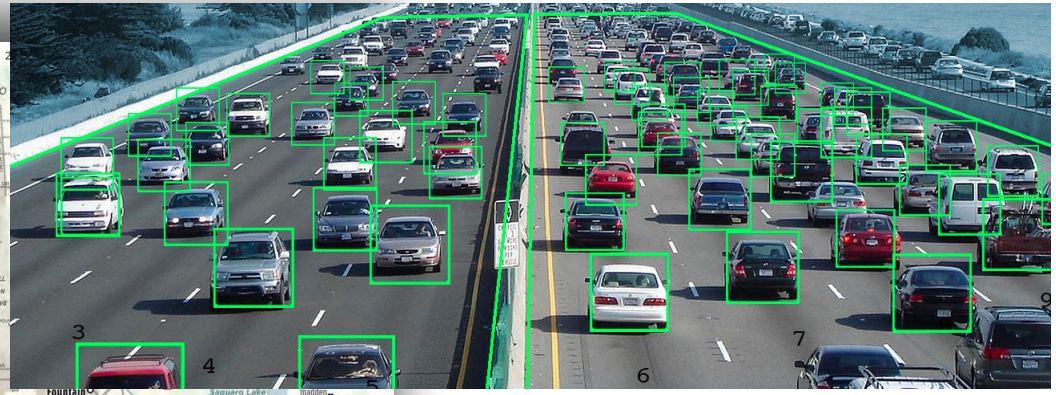


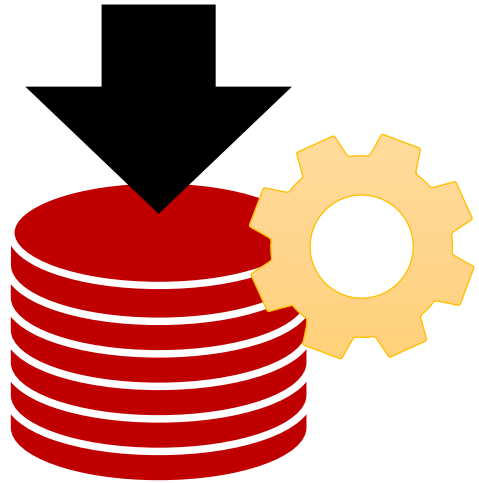
ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	<x1,y1>	
2	Motorcycle		…	<x2,y2>	
…	…		…	…	
…	…		…	…	

CityID	CityName	CityBoundaries	….
1	Phoenix	Polygon representing the city boundaries	…
2	Tempe		…
…	…		…
…	…		…

```
SELECT Count(*)  FROM   City, TrafficCams T
WHERE ST_Contains (city.geom, T.location)
    AND city.name = 'Tempe'
    AND T.ObjectType = 'Car'
```



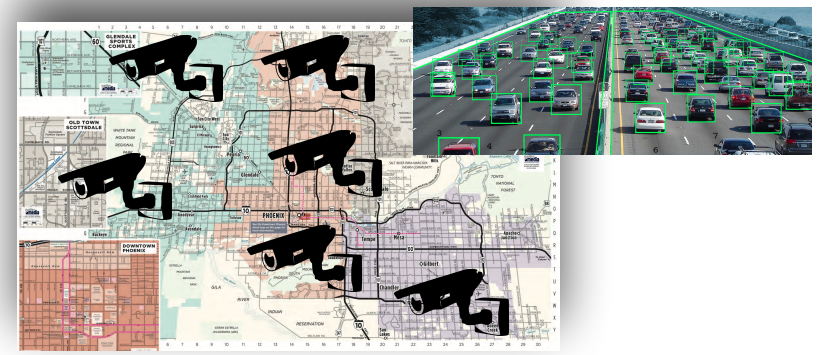




Report the count of **Cars Detected**

in the city of Tempe area

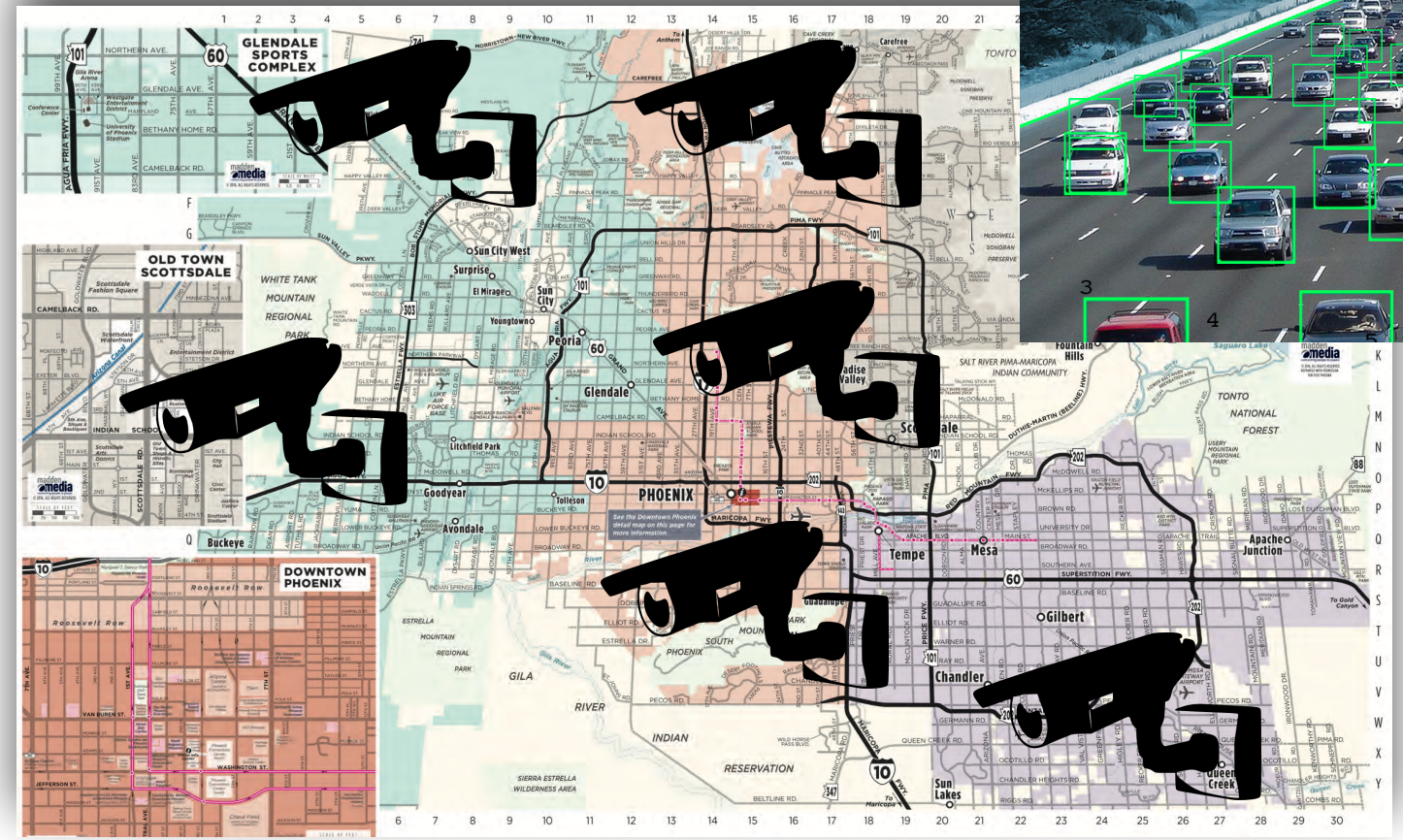
between 13:45 and 14:00 pm on 2018/03/01

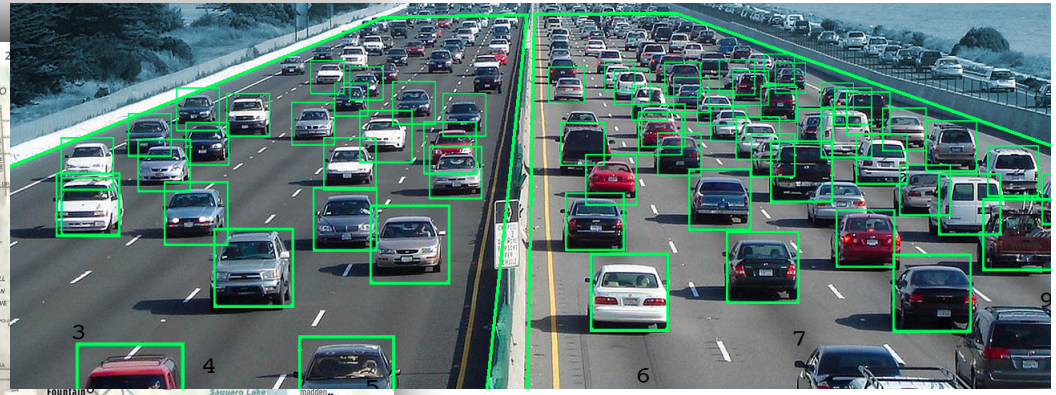


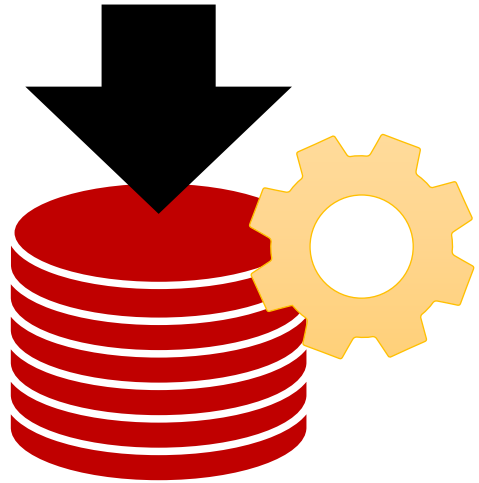
ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	<x1,y1>	
2	Motorcycle		…	<x2,y2>	
…	…		…	…	
…	…		…	…	

CityID	CityName	CityBoundaries	….
1	Phoenix	Polygon representing the city boundaries	…
2	Tempe		…
…	…		…
…	…		…

SELECT Count(*) FROM City, TrafficCams T
WHERE ST_Contains (city.geom, T.location)
AND city.name = 'Tempe'
AND T.ObjectType = 'Car'
AND T.Time > "13:45 2018/03/01" AND T.Time < 14:00 pm on 2018/03/01



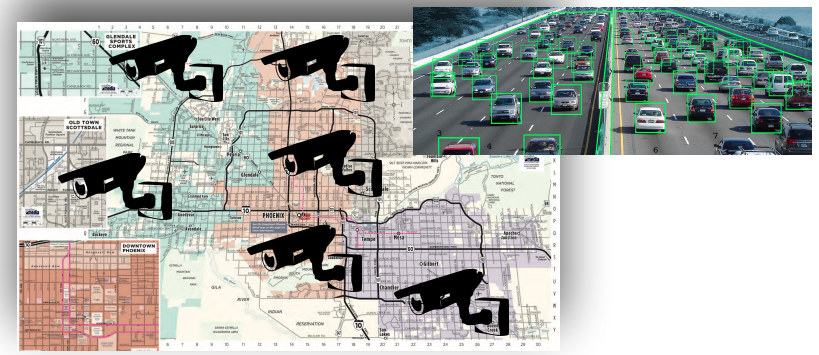




Report the **Cars Detected**

within a 10 meters distance from ASU

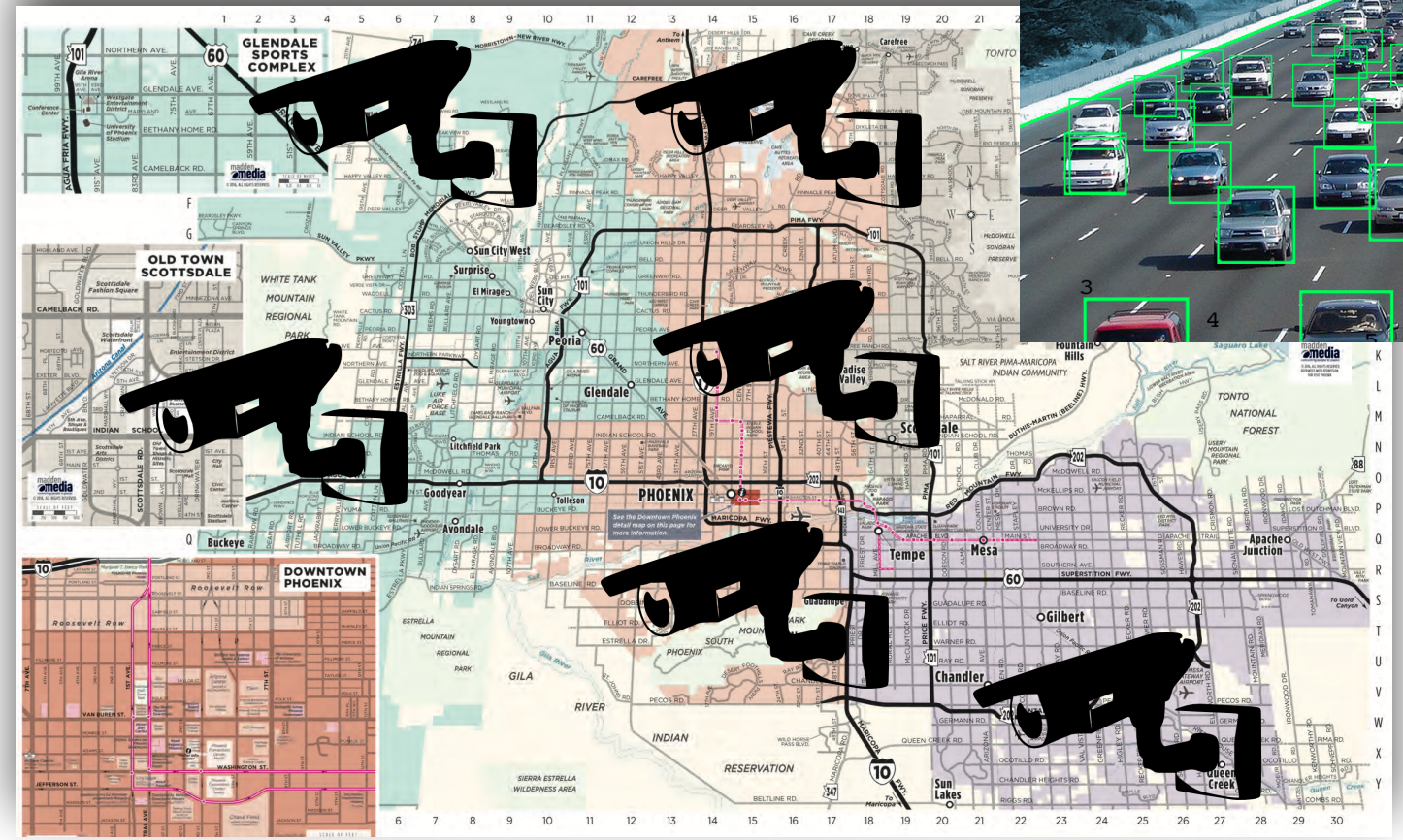
between 13:45 and 14:00 pm on 2018/03/01

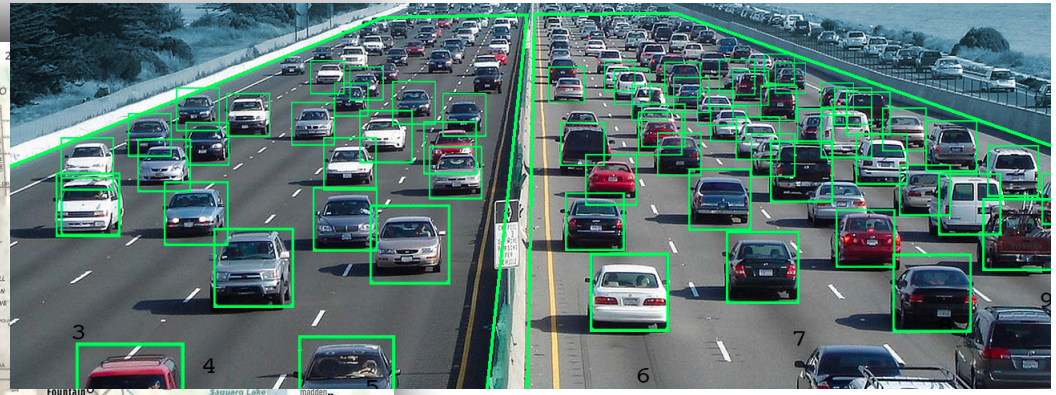


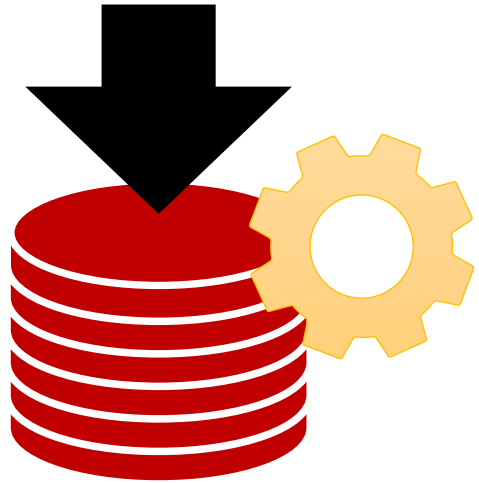
ObjectID	ObjectType	Properties	….	Location	Time
1	Car	Blue BMW	…	<x1,y1>	
2	Motorcycle		…	<x2,y2>	
…	…		…	…	
…	…		…	…	

PID	PNAME	CityBoundaries	….
1	ASU	Point representing the ASU boundaries	…
2	Stadium		…
…	…		…
…	…		…

SELECT T.* FROM PointOfInterests P, TrafficCams T
WHERE STD_WITHIN (P, T.location,10)
___AND city.name = 'Tempe'
AND T.ObjectType = 'Car'
AND T.Time > "13:45 2018/03/01" AND T.Time < 14:00 pm on 2018/03/01





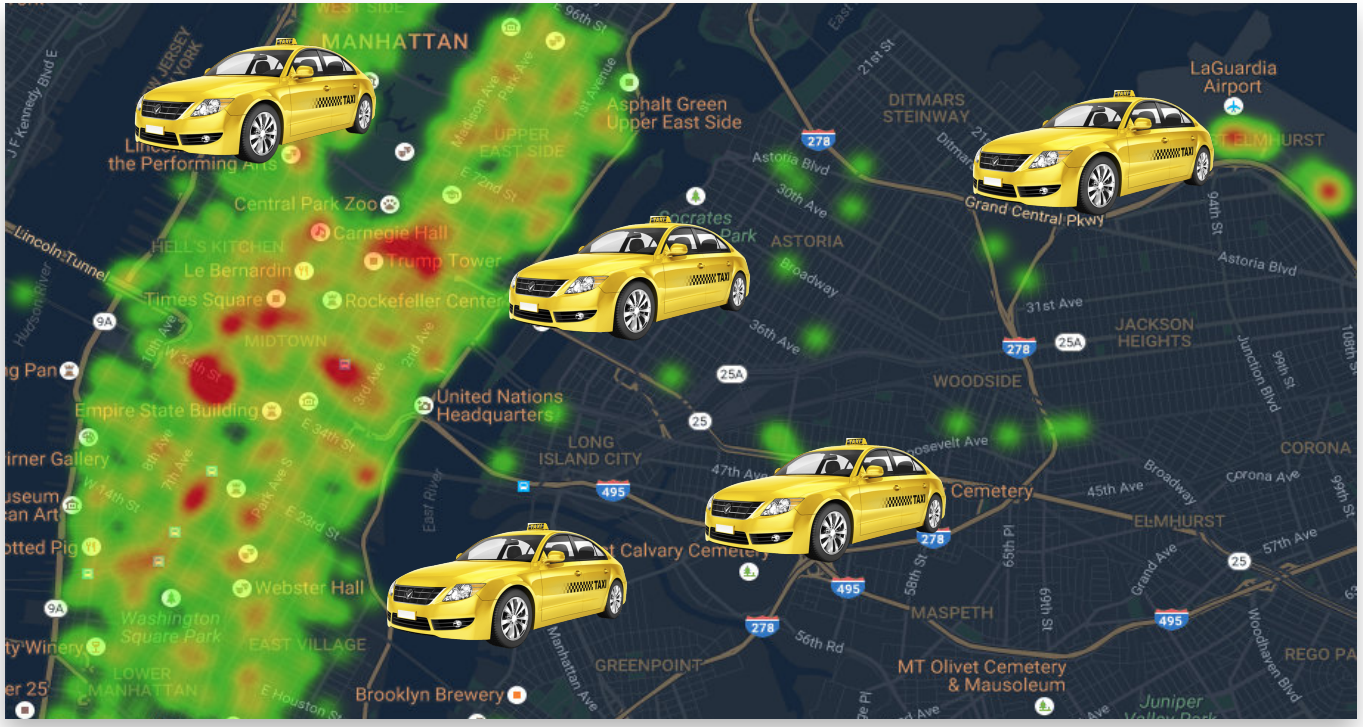


Report **Cars Detected**
[in Downtown Tempe area]
between 13:45 and 14:00 pm on 2018/03/01

Report **Cars Detected**
[within the City of Tempe area]
In the last ten minutes time window

Report **Cars Detected**
[Nearby (e.g., within 0.1 mi) a School in the City of Tempe]

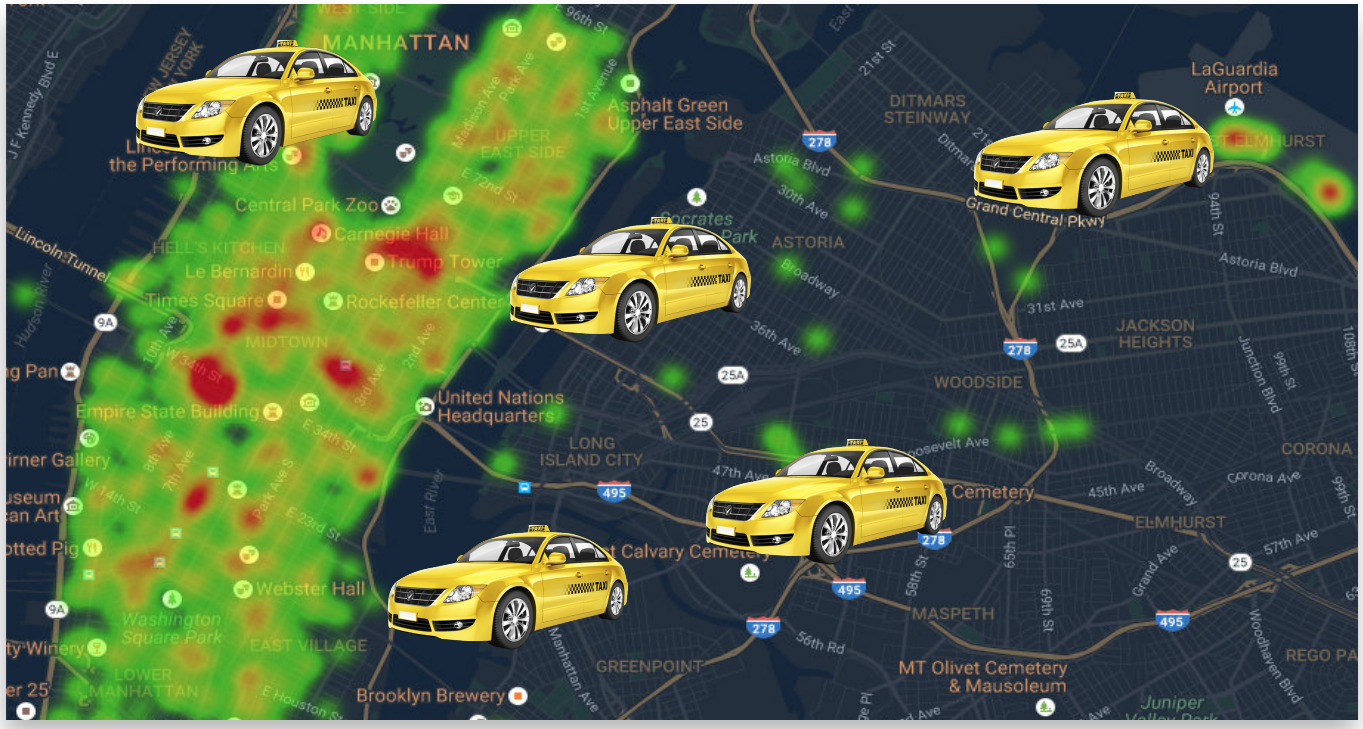
Monitor **Cars Detected**
[Nearby (e.g., within 0.1 mi) a School in the City of Temp]



NYC Taxi Trips GPS Traces

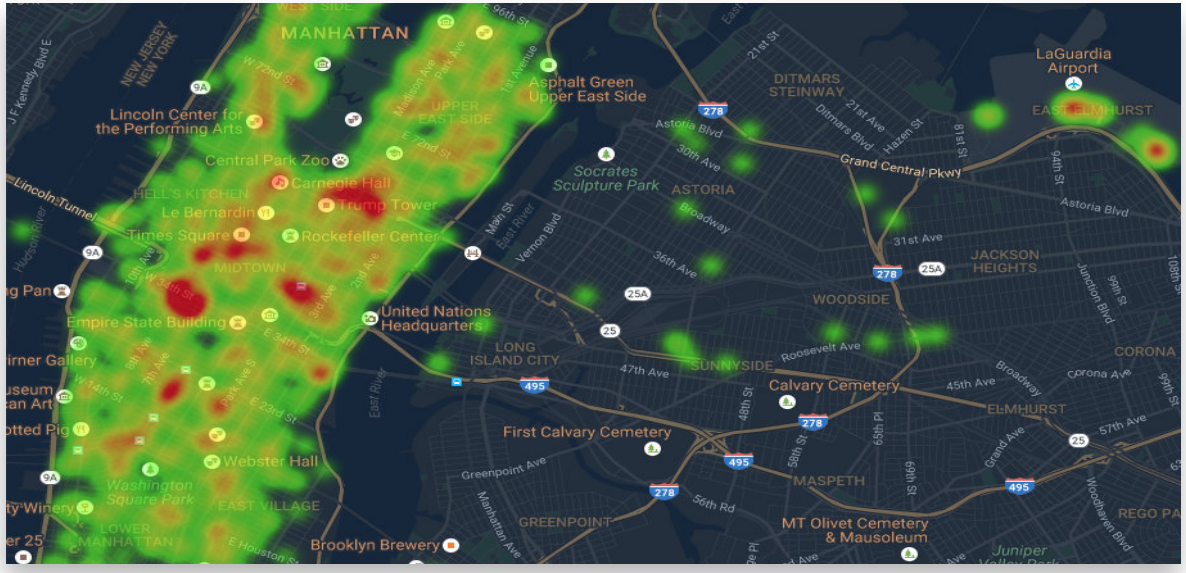
NYC taxi and limousine services released data for over a billion taxi trips

Spatial Database Systems



Trip Time Stamp	Pick up location	….	Fare amount
1	Laguardia	…	$40
2	JFK	…	$30
3	<lat,long>	…	$…
…	…	…	…

SELECT * FROM city, TaxiTrip
WHERE ST_Contains (city.geom, TaxiTrip.geom)
AND city.name = 'Manhattan'



Pick Up Time	Pick up location	….	Fare amount
1	Laguardia	…	$40
2	JFK	…	$30
…	…	…	…
…	…	…	…

Heap File

1	Laguardia	…	40
2	JFK	…	30
			.

Data Page 1

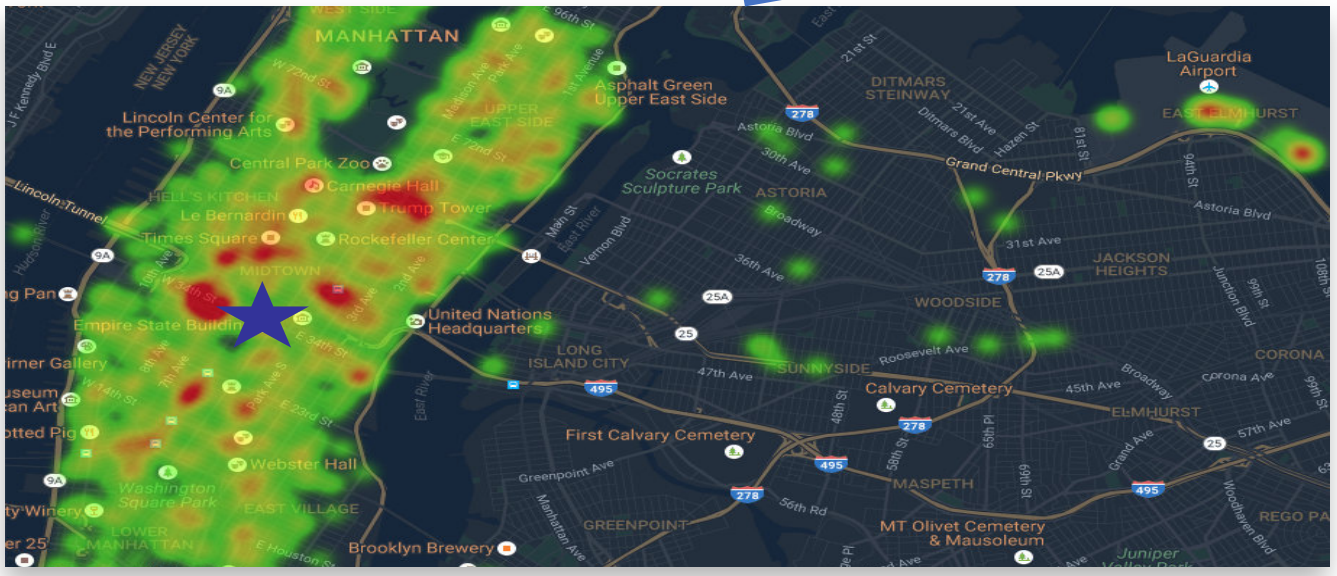
101	Times	…	10
102	ESB	…	31
			.

Data Page 2

…

1001	MET	…	15
1002	Park	…	20
			.

Data Page N



Pick Up Time	Pick up location	….	Fare amount
1	Laguardia	…	$40
2	JFK	…	$30
102	ESB	…	$31
…	…	…	…

Heap File

1	Laguardia	…	40
2	JFK	…	30
.			

Data Page 1

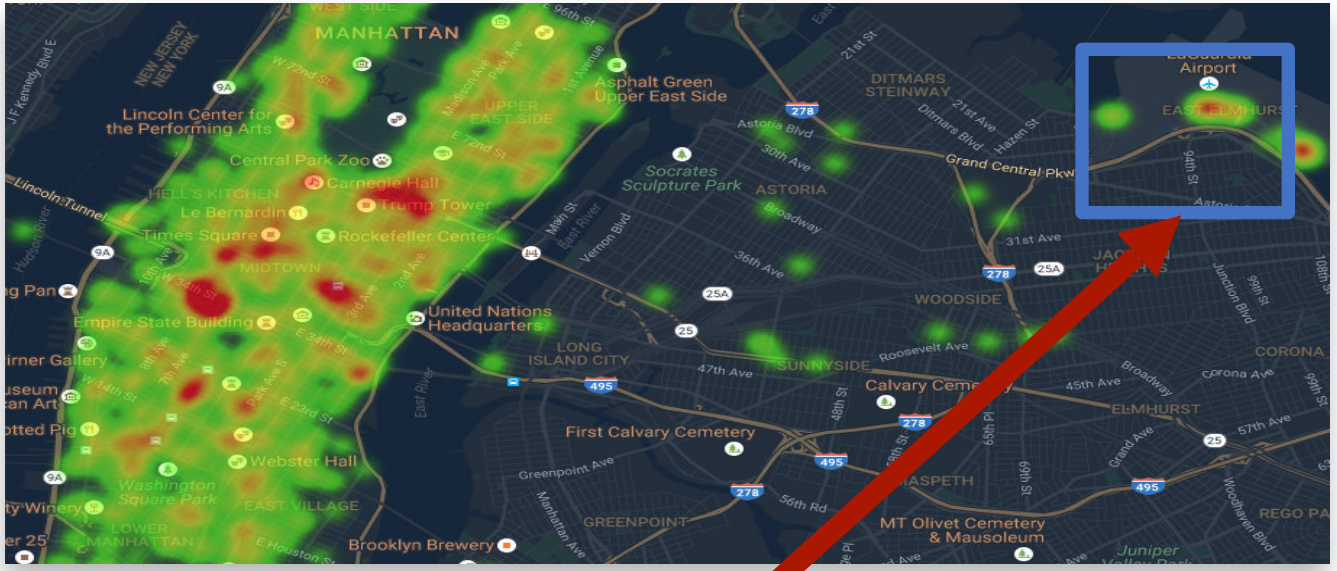
101	Times	…	10
102	ESB	…	31
.			

Data Page 2

…

1001	MET	…	15
1002	Park	…	20
.			

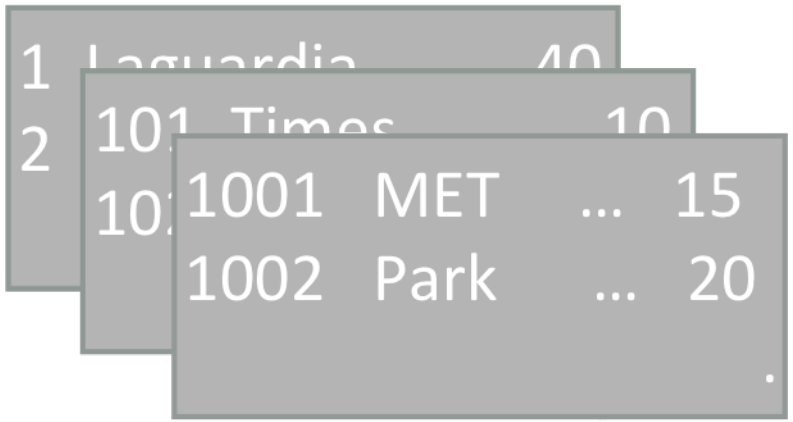
Data Page N



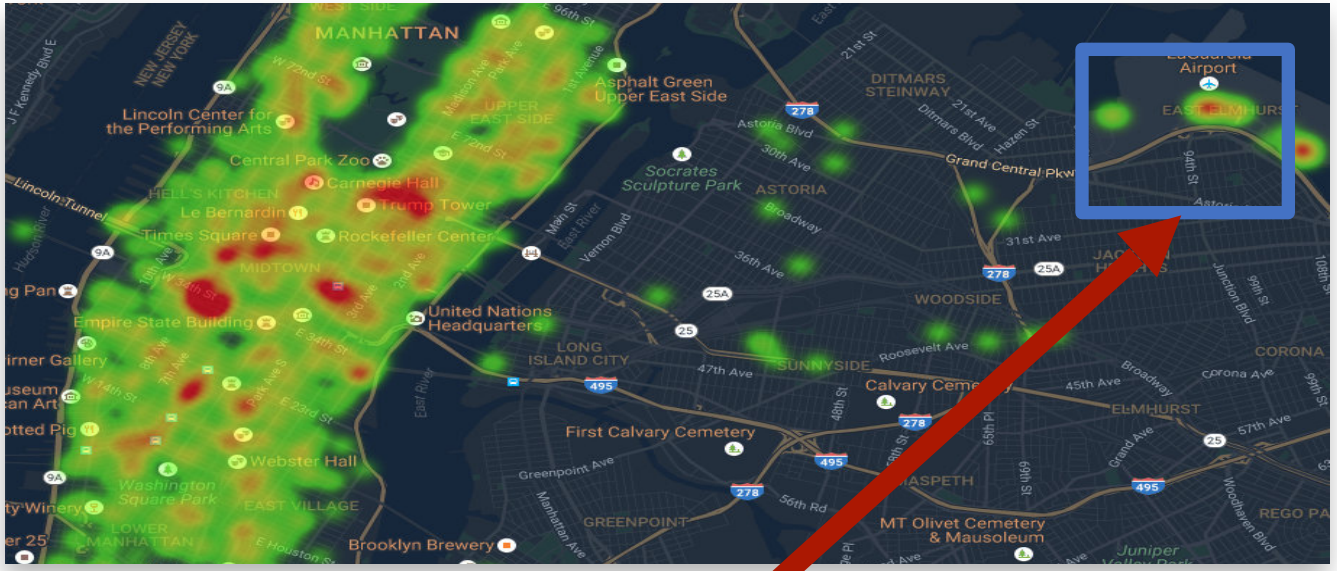
Trip Time Stamp	Pick up location	….	Fare amount
1	Laguardia	…	$40
2	JFK	…	$30
3	<lat,long>	…	$…
…	…	…	…

Range Query

Return all the NYC taxi trips for which the pick up location is within the input range



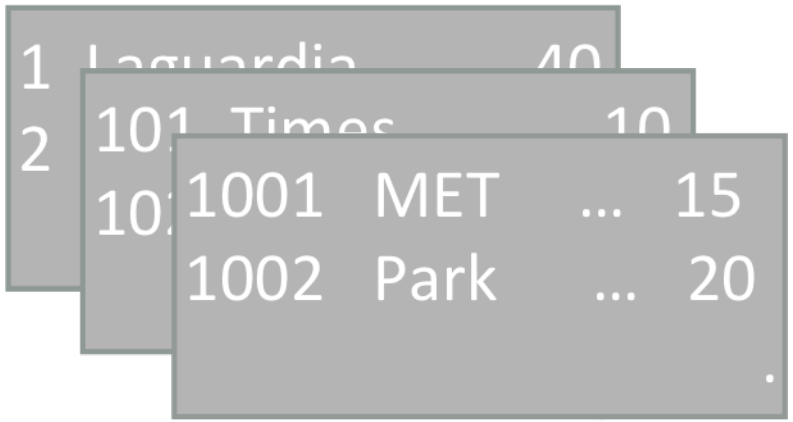
Data Pages



Trip Time Stamp	Pick up location	….	Fare amount
1	Laguardia	…	$40
2	JFK	…	$30
3	<lat,long>	…	$…
…	…	…	…

KNN Query

Return the **K** taxi trips for which the pick up location is the closest to JFK airport



Data Pages

From Databases to Data Streams

Traditional DBMS makes several assumptions:

- persistent data storage
- relatively static records
- (typically) no predefined notion of time
- complex one-off queries

From Databases to Data Streams

Some applications have very different requirements:

- data arrives in real-time
- data is ordered (implicitly by arrival time or explicitly by timestamp)
- too much data to store!
- data never stops coming
- ongoing analysis of rapidly changing data

One-Time versus Continuous Queries

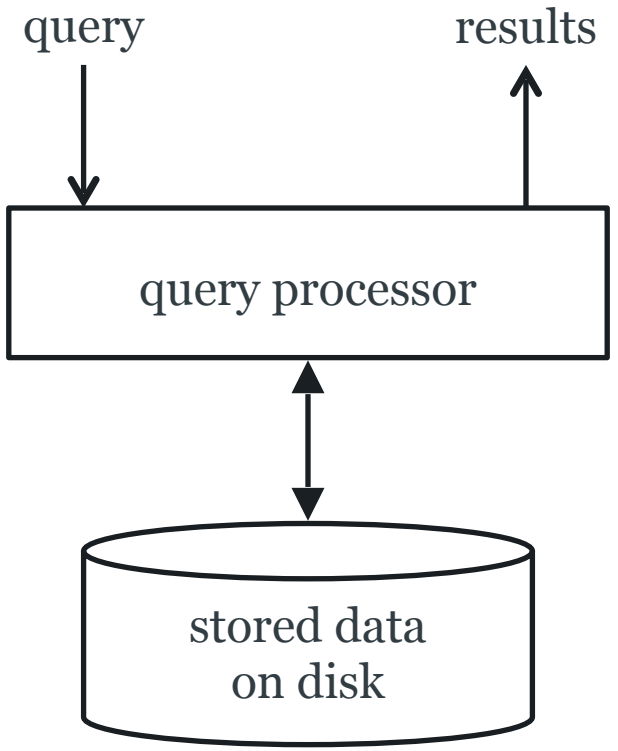
One-time queries

- Run once to completion over the current data set

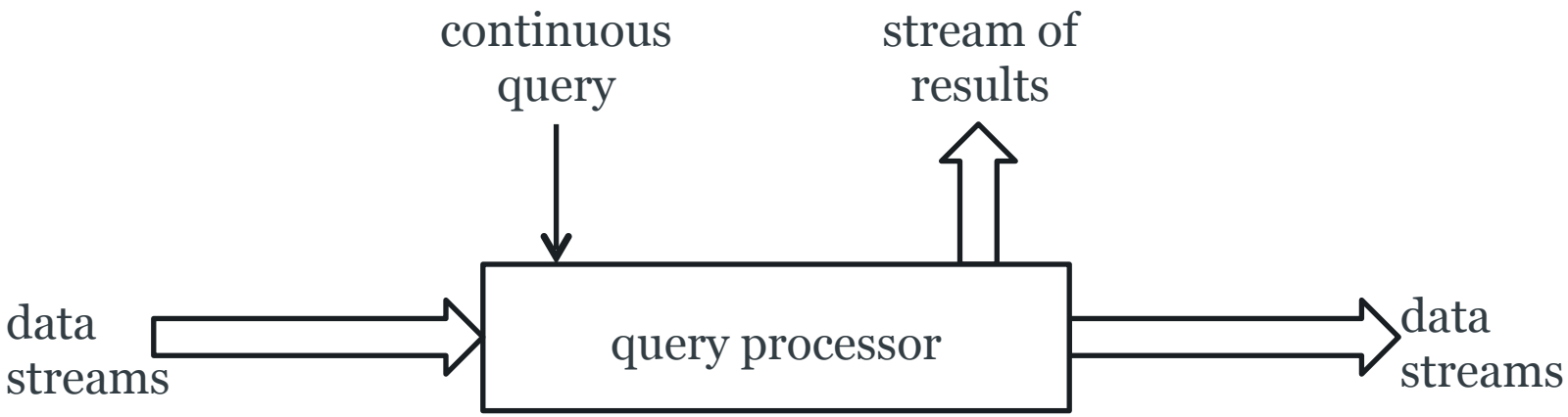
Continuous queries

- Issued once and then continuously evaluated over a data stream
  - "Notify me when the temperature drops below X"
  - "Tell me when prices of stock Y > 300"

Database Management System



Data Stream Management System (DSMS)



DBMS versus DSMS

DBMS

- Persistent relations (relatively static, stored)
- One-time queries
- Random access
- "Unbounded" disk store
- Only current state matters

DSMS

- Transient streams (on-line analysis)
- Continuous queries (CQs)
- Sequential access
- Bounded main memory
- Historical data is important

DBMS versus DSMS

DBMS

- No real-time services
- Relatively low update rate
- Data at any granularity
- Assume precise data
- Access plan determined by query processor, physical DB design

DSMS

- Real-time requirements
- Possibly multi-GB arrival rate
- Data at fine granularity
- Data stale/imprecise
- Unpredictable/variable data arrival and characteristics

A Motivation for Stream Processing

Over the past twenty-five years:

- CPU performance has increased by a factor of >1,000,000
- Typical RAM capacity increased by a factor of >1,000,000
- RAM access time has decreased by a factor of >50,000
- Typical HD capacity increased by a factor of >50,000

- HD access time has decreased by a factor of ~10

Architectural Issues

DBMS

- Resource (memory, disk, per-tuple computation) rich
- Extremely sophisticated query processing, analysis
- Useful to audit query results of data stream systems.
- Query Evaluation: Arbitrary
- Query Plan: Fixed.

DSMS

- Resource (memory, per-tuple computation) limited
- Reasonably complex, near real time, query processing
- Useful to identify what data to populate in database
- Query Evaluation: One pass
- Query Plan: Adaptive

Example: Continuous Query Language

Queries produce/refer to relations and streams

Based on SQL, with the addition of:

- Streams as new data type
- Continuous instead of one-time semantics
- Windows on streams (derived from SQL-99)
- Sampling on streams (basic)

Query Processing

Construct query plan based on relational operators, as in an RDBMS

- Selection
- Projection
- Join
- Aggregation (group by)

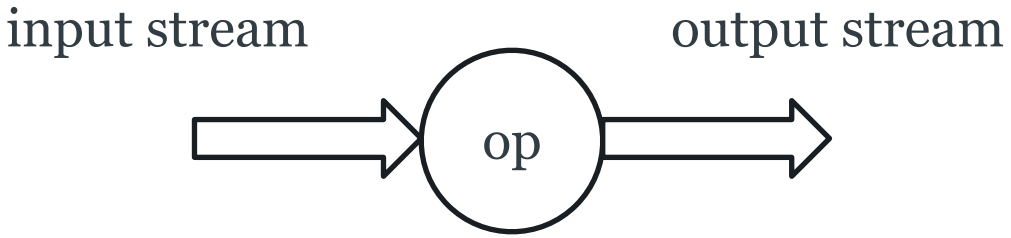
Combine plans from continuous queries (reduce redundancy)

Stream tuples through the resulting network of operators

Tuple-at-a-time Operators

Evaluation requires consideration of only one tuple at a time

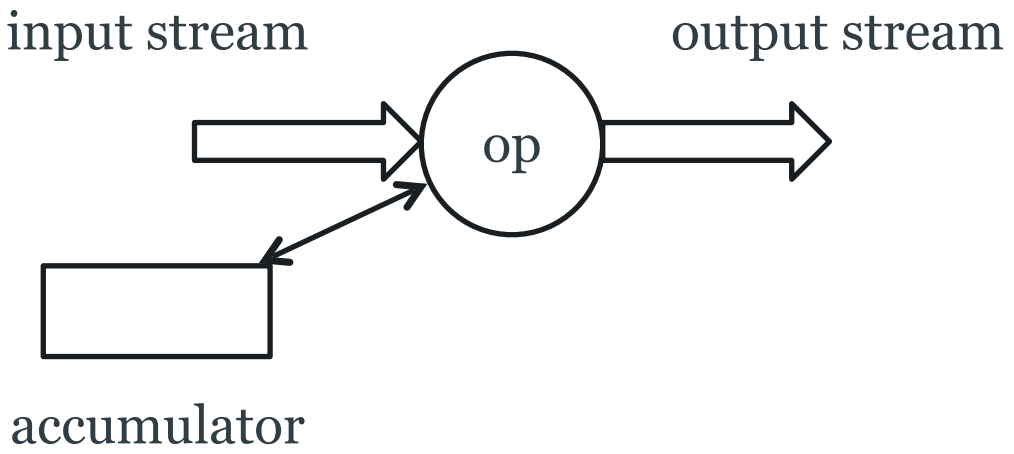
- Selection and projection



Full Relation Operators

Some full relation operators can work on a tuple at a time

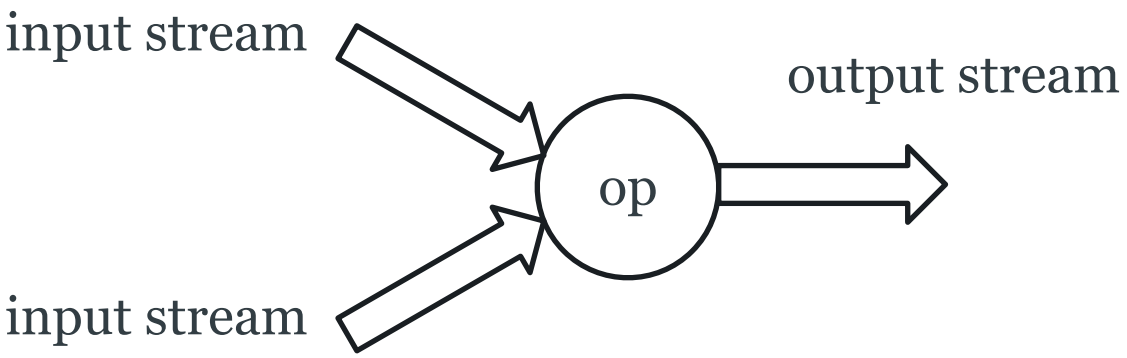
- Count, sum, average, max, min (even with group by)
- (order by, however, can't)



Full Relation Operators

Other (binary) full relation operators can't

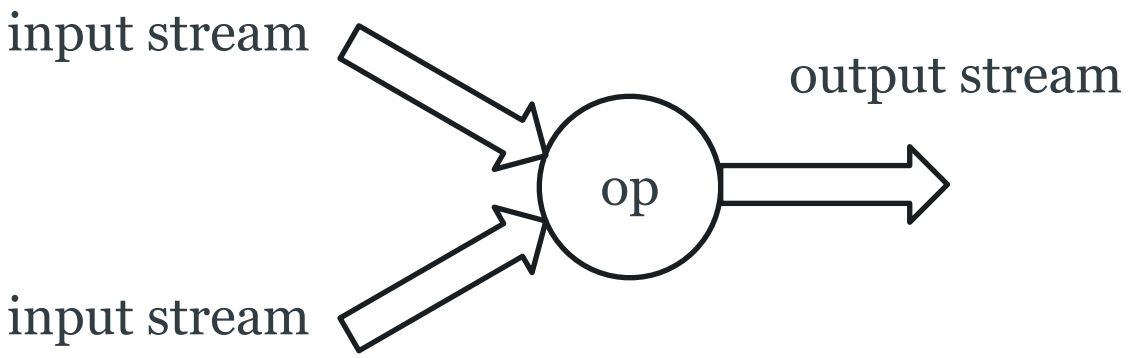
- Intersection, difference, product, join
- (union, however, can be evaluated tuple-by-tuple)



Full Relation Operators

May block when applied to streams

- no output until entire input seen, but streams are unbounded
- joins may need to join tuples that are arbitrarily far apart



Relation/Stream Translation

Some relational operators can work directly on streams

- Selection, projection, union, some aggregates

Some relational operators need to work on relations

- Join, product, difference, intersection, other aggregates

Stream-to-relation operators

- Windows

Relation-to-stream operators

- Istream, Dstream, Rstream

Windows

Mechanism for extracting a finite relation (synopsis) from an infinite stream

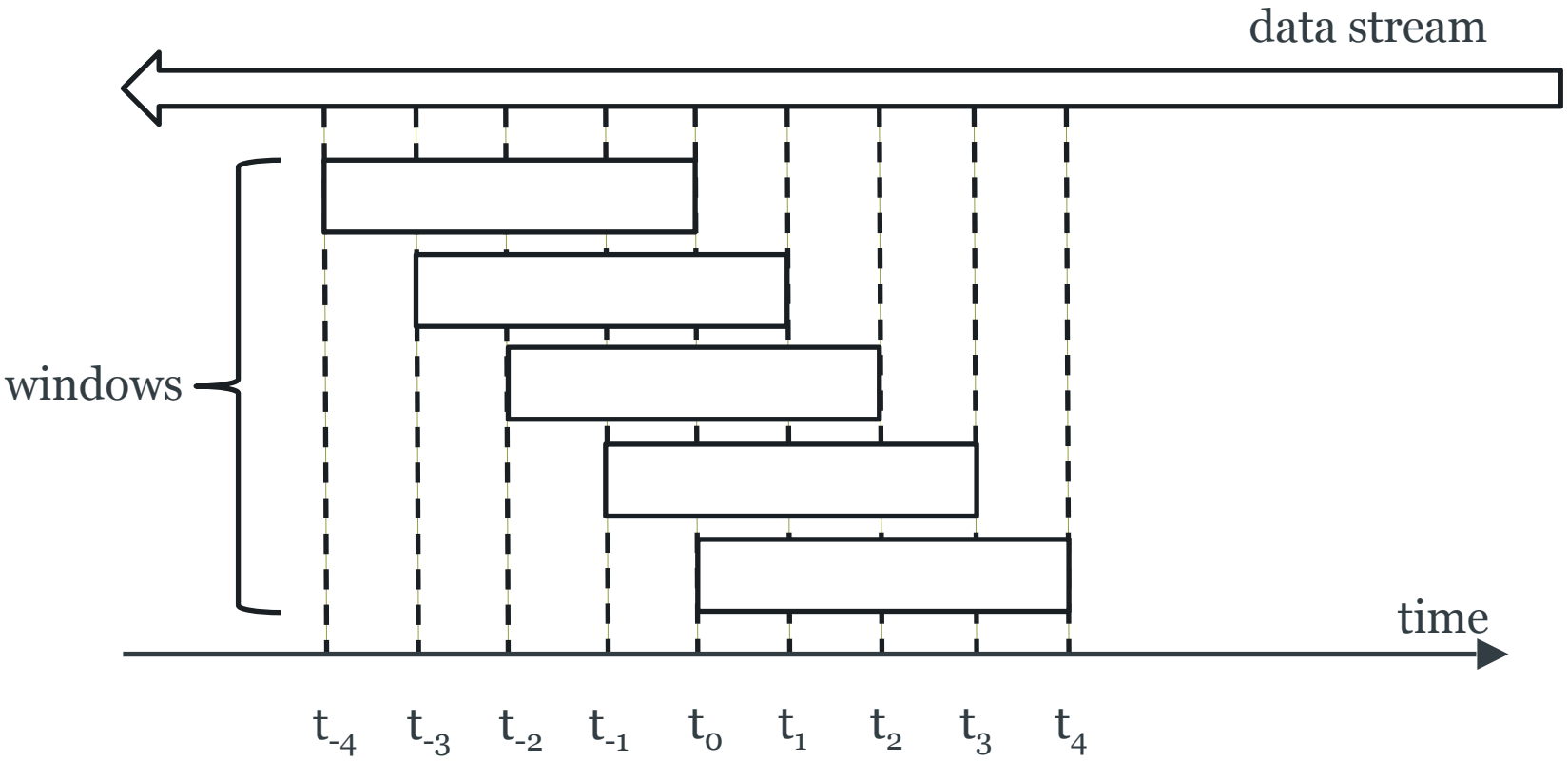
Various window proposals for restricting operator scope.

- Windows based on ordering attribute (e.g. last 5 minutes of tuples)
- Windows based on tuple counts (e.g. last 1000 tuples)
- Windows based on explicit markers (e.g. punctuations)
- Variants (e.g., partitioning tuples in a window)

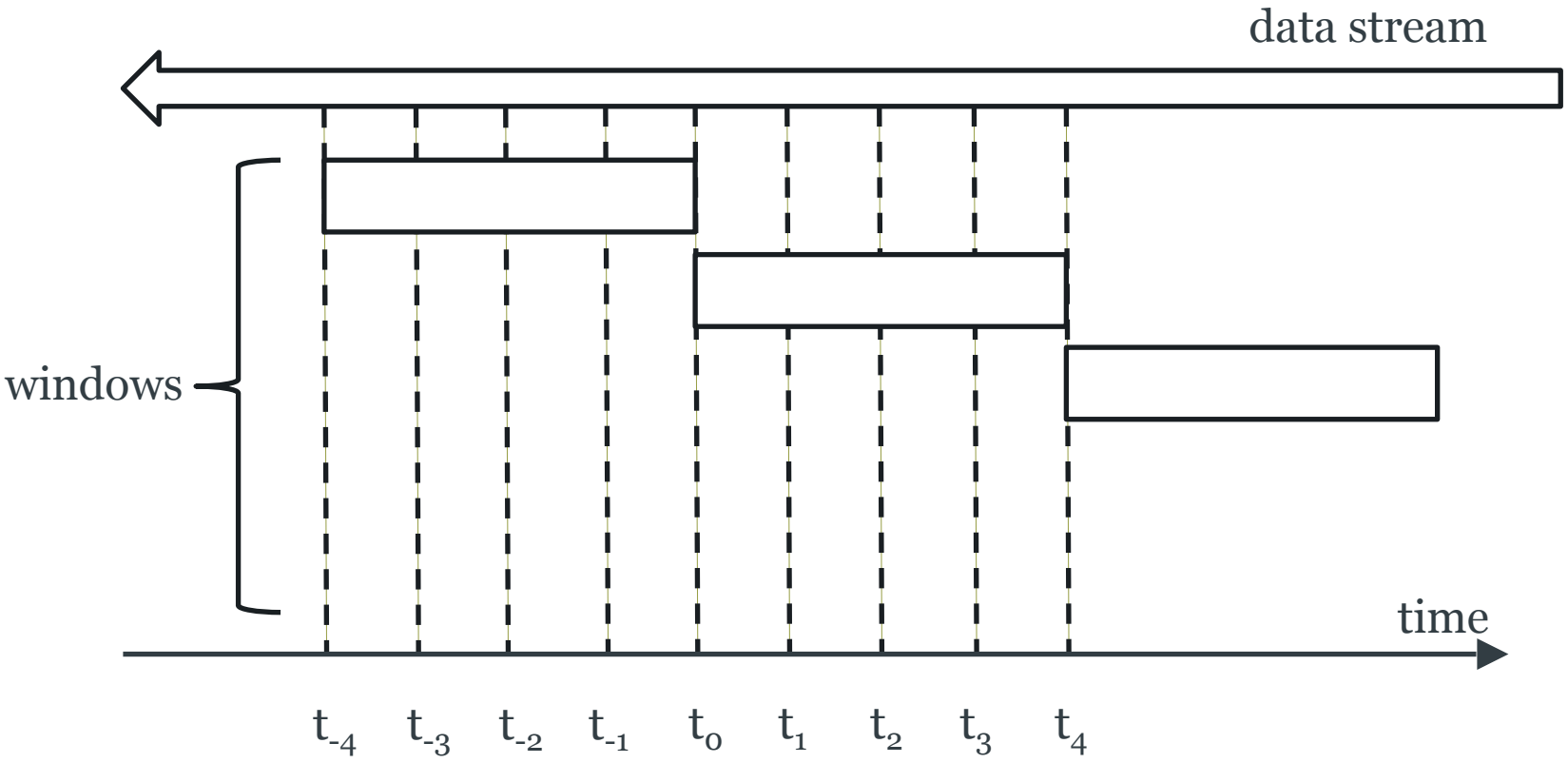
Various window behaviours

- Sliding, tumbling

Sliding Windows



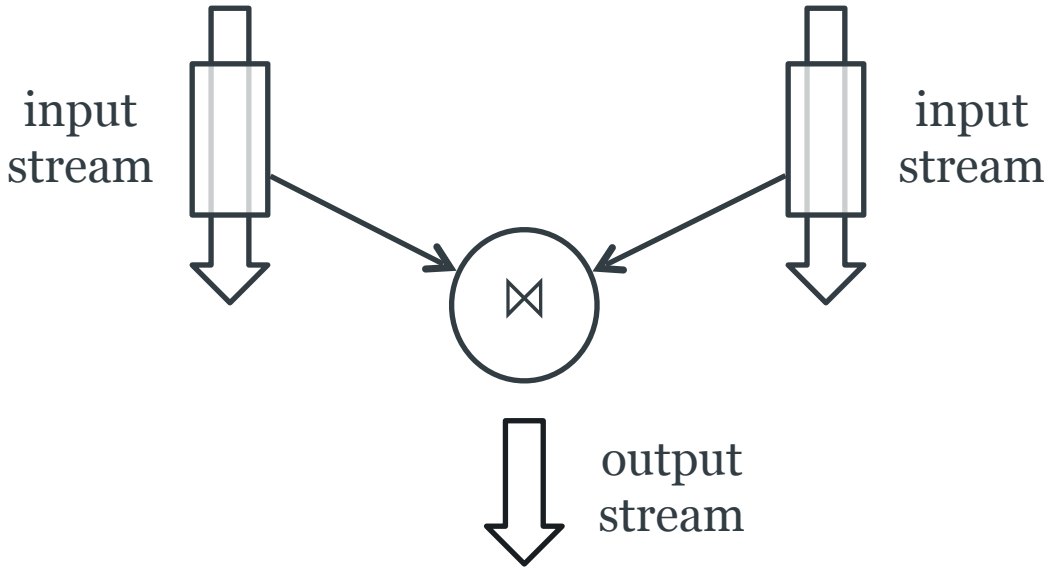
Tumbling Windows



Join Evaluation

Consider a stream-based join operation:

- a conventional join over a pair of windows on the input streams
- outputs a stream of tuples joined from the input streams



Scalability and Completeness

DBMS deals with finite relations

- query evaluation should produce all results for a given query

DSMS deals with unbounded data streams

- may not be possible to return all results for a given query
- trade-off between resource use and completeness of result set
- size of buffers used for windows is one example of a parameter that affects resource use and completeness
- can further reduce resource use by randomly sampling from streams

Relation-to-Stream Operators

Insert Stream (Istream)

- Whenever a tuple is inserted into the relation, emit it on the stream

Delete Stream (Dstream)

- Whenever a tuple is deleted from the relation, emit it on the stream

Relation Stream (Rstream)

- At every time instant, emit every tuple in relation on the stream

Example CQL Query

```
SELECT Istream(*)
FROM S [rows unbounded]
WHERE S.A > 10
```

S is converted into a relation (of unbounded size!)

Resulting relation is converted back to a stream via Istream

Example CQL Query

```
SELECT *
FROM S
WHERE S.A > 10
```

S is a stream – query plan involves only selection, so window is now unnecessary

Example CQL Query

```
SELECT *
FROM   S1 [rows 1000],
       S2 [range 2 minutes]
WHERE S1.A = S2.A AND S1.A > 10
```

Windows specified on streams

- Tuple-based sliding window – [rows 1000]
- Time-based sliding window – [range 2 minutes]

Example CQL Query

SELECT Rstream(S.A, R.B)
FROM S [now], R
WHERE S.A = R.A

Query probes a stored table R based on each tuple in stream S and streams the result

- [now] – time-based sliding window containing tuples received in last time step

Query Optimisation

Traditionally relation cardinalities used in query optimiser

- Minimize the size of intermediate results.

Problematic in a streaming environment

- All streams are unbounded = infinite size!

Query Optimisation

Need novel optimisation objectives that are relevant when input sources are streams

- Stream rate based (e.g. NiagaraCQ)
- Resource-based (e.g. STREAM)
- QoS based (e.g. Aurora)

Continuous adaptive optimisation