Data Processing and Storage

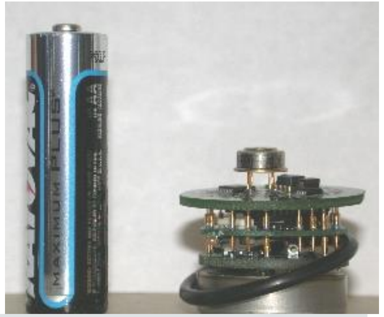
Samuel Madden
Professor, MIT EECS

Computer Science and Artificial Intelligence Laboratory (CSAIL)
Massachusetts Institute of Technology

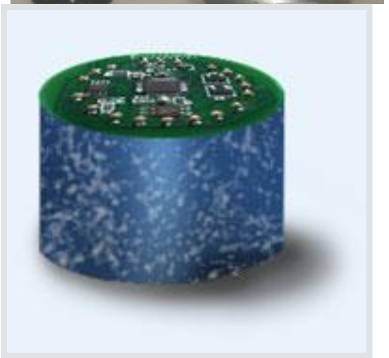
MIT PROFESSIONAL EDUCATION

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

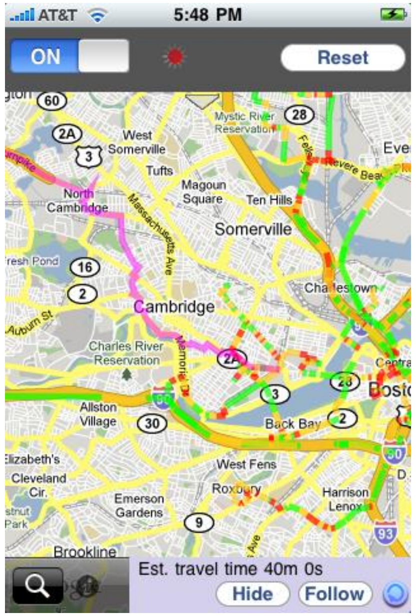
MY IOT EXPERIENCE



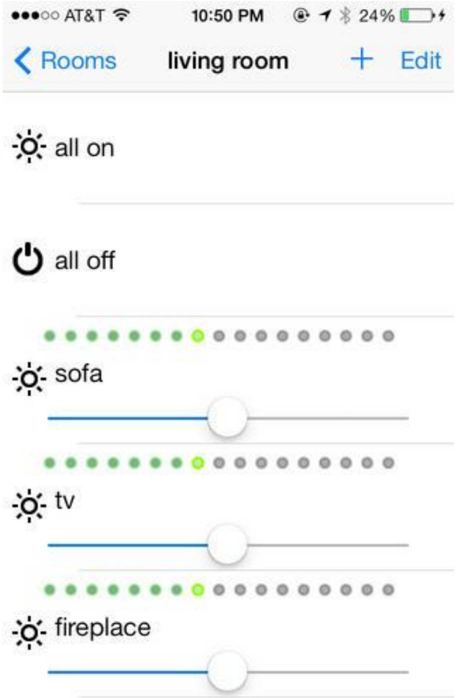
TinyOS: operating system for embedded sensors



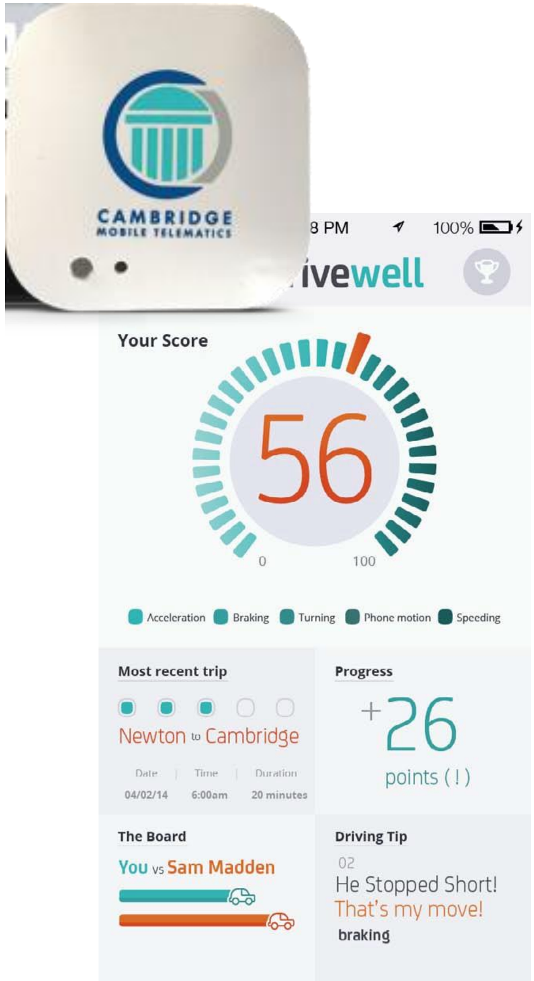
TinyDB: The Sensornet is the Database



iCarTel crowdsourced traffic aware routing app



Lutron Light Control app for controlling lutron lighting systems from iPhone



DriveWell safe driving app and BTLE accident-detection device

PROFESSIONAL EDUCATION

CSAIL

IOT AND THE NEED FOR DATA

Many IoT applications are about data:

- **Infrastructure monitoring**
  - Homes
  - Pipes
  - Power plants
  - Medical patients
- **Fleet & vehicle tracking**
- **Security**
- **Precision agriculture**
- …

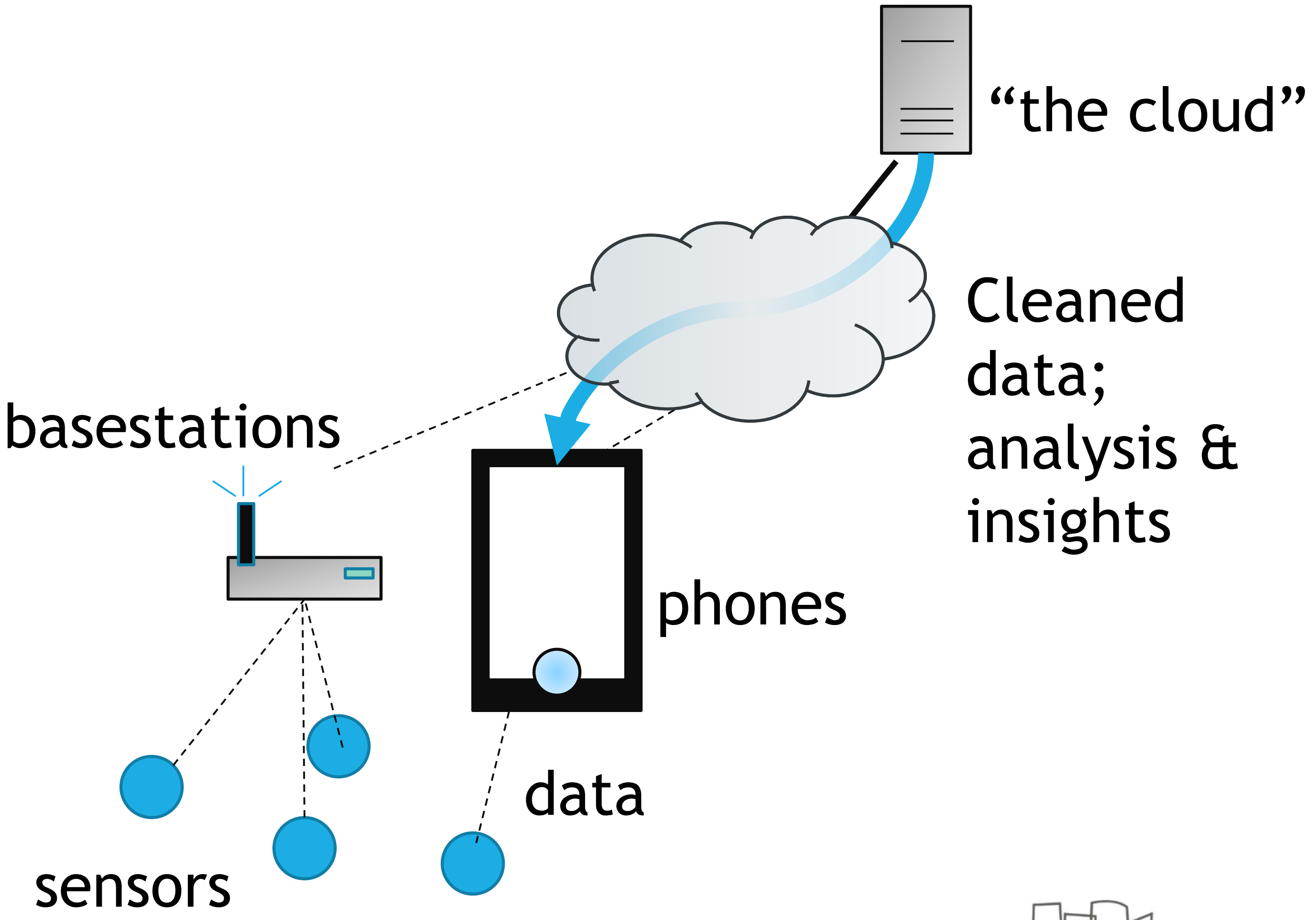
TYPICAL IOT DATA PROCESSING SCENARIO

Data path: sensors ➔ phones/basestations ➔ cloud

Sensors use low-power (BTLE, Zigbee) wireless

Phones and basestations use WiFi, cellular, or wired Internet links

Processing happens on sensors, basestations, phones, and cloud



CHALLENGES OF IOT DATA

IoT data is fundamentally different than other types of data, because it is *sampled*, and comes from devices that have failures and experience noise.

IoT applications must deal with three key data challenges:

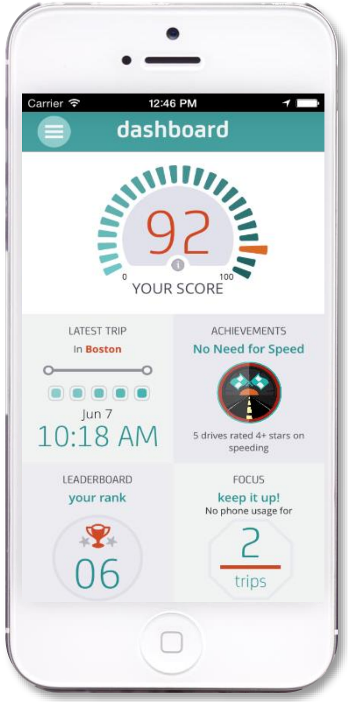
1. Limited resources (power, bandwidth, storage)
2. Missing and noisy data
3. Outliers and anomalies

CASE STUDY: DRIVEWELL + TAG

Key capabilities: “safety score”, end-to-end collision alerting facility

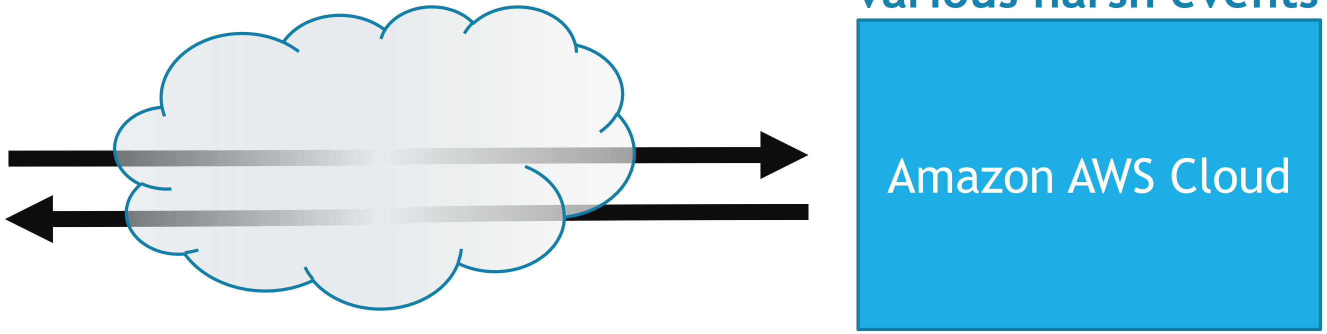
Requirement 1:
3+ years battery life

Acceleration Data
Impacts
Trip starts
(Over BTLE)



Trip data:
Acceleration
Gyroscope
Position

Requirement 2:
< 5% battery drain / hour *when driving*



Requirement 3: 10 second end-to-end notification of accidents

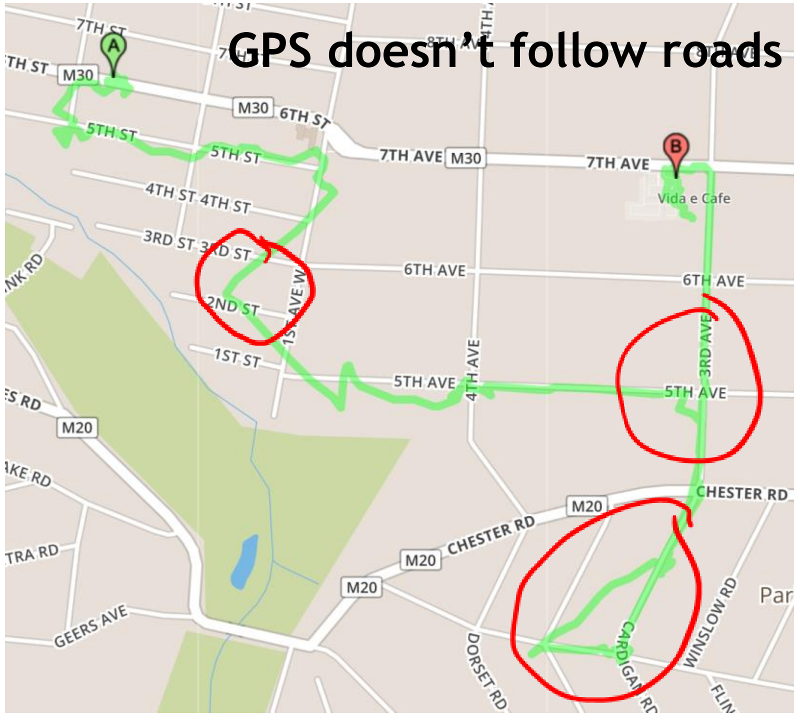
Requirement 5:
Accurately measure mileage and detect various harsh events

Requirement 4:
Real time trip feedback in a few minutes

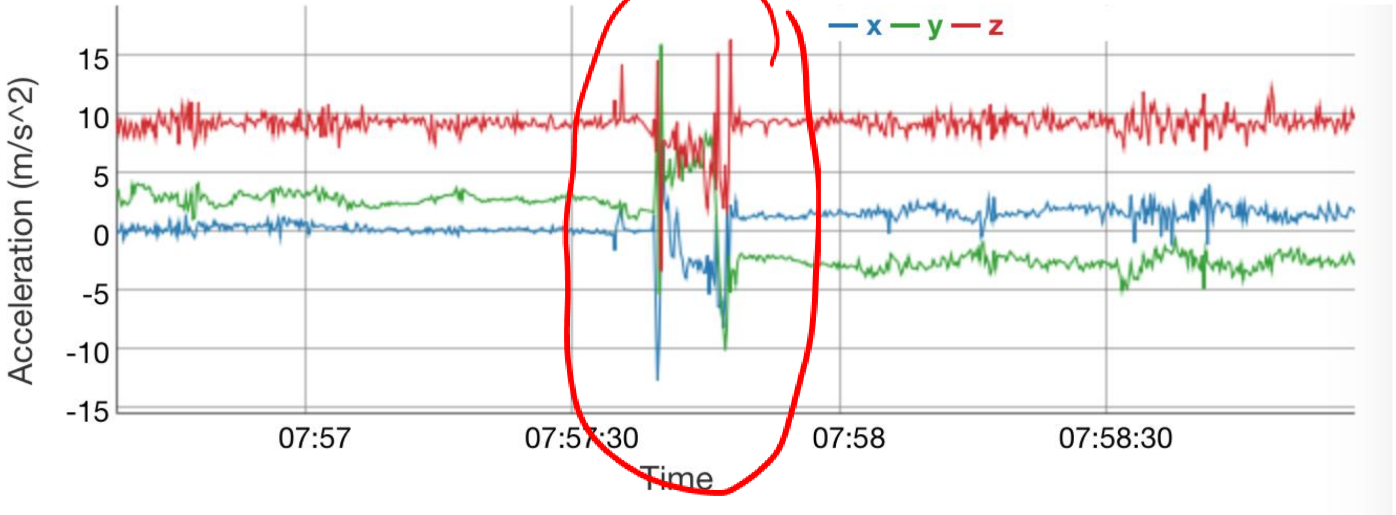
MIT PROFESSIONAL EDUCATION

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

DRIVEWELL DATA CHALLENGES



Users move phone while driving



Certain classes of devices experience failures

Discover CBCharacteristic for CBService misses a few characteristics

442 Views 15 Replies Latest reply: Sep 29, 2015 2:05 AM by masakazu

REST OF THE MODULE

1. Limited resources (power, bandwidth, storage)

2. Missing and noisy data

3. Outliers and anomalies

HANDLING LIMITED RESOURCES

ARCHITECTS VS DATA SCIENTISTS

Data analysts always want more data

IoT system architects limited by *constraints*

E.g.,

- Battery powered devices need to last X years
- Radios can transmit at Y bytes per second
- Device can store Z hours of audio

THE THREE B'S

Battery

Bandwidth

Bytes

BATTERY

Power consumption often limits what you can collect

Some technologies (e.g., 3G radios, GPS) use lots of energy

QUICK PHYSICS RECAP

SI Unit of Energy = Joule (J)

SI Unit of Power = Watt (W) = Joules / second

Wattage of a device is Amperage (A) x Voltage (V)

Wattage determines power consumption of devices (milliwatts, or mW)

Battery *capacity* is its stored energy; measured in milliamp-hours (mAh)

Example: iPhone 6 has 1800 mAh battery; LTE radio uses about 1700 mW when transmitting @ 1 Mbit/sec

iPhone is 3.8V, so 1800 mAh = 6840 mWh @ 3.8V; 6840/1700 = 4 hours

➔ iPhone (doing nothing else) can transmit for about 4 hours on LTE

POWER USED BY SOME COMMON COMPONENTS

Component	Approximate Power Consumption
LTE Radio (transmit @ 1 Mb/s)	1700 mW
3G Radio (transmit @ 1 Mb/s)	1700 mW
WiFi (transmit @ 1Mb / s)	400 mW
ARM+RAM uProc (100% cpu)	2000 mW
ARM+RAM uProc (idle)	70 mW
Smartphone Screen (full brightness)	850 mW
GPS (once lock is acquired)	100-150 mW
Accelerometer (@10 Hz)	75 uW
Image sensor (@1080p/30Hz)	270 mW (Sony IMX206CQC)

Collecting the data is cheap; displays & radios & processing are expensive

SOME CAVEATS

Startup and shutdown times

- E.g., LTE radio 10 second shutdown

CPU power vs processing efficiency

- Faster processor ➔ less processing time

Radio power vs bandwidth

- Higher bandwidth ➔ more power
- Higher bandwidth ➔ less time ➔ less power

Uplink vs downlink

POWER CONSIDERATIONS IN IOT APPLICATIONS

More data ➔ more sensing, more processing, more transmission

➔ Collect what you need!

On device processing can reduce data transmission, but processing is also expensive

WiFi/BluetoothLE use MUCH less energy than 3G/LTE

ILLUSTRATION: IN-NETWORK DATA PROCESSING IN TINYDB

Multihop data collection

-Divide sample period into short time *intervals*

-Assign each node to an interval according to its depth in the tree

Key idea: combine data as it is transmitted in the network

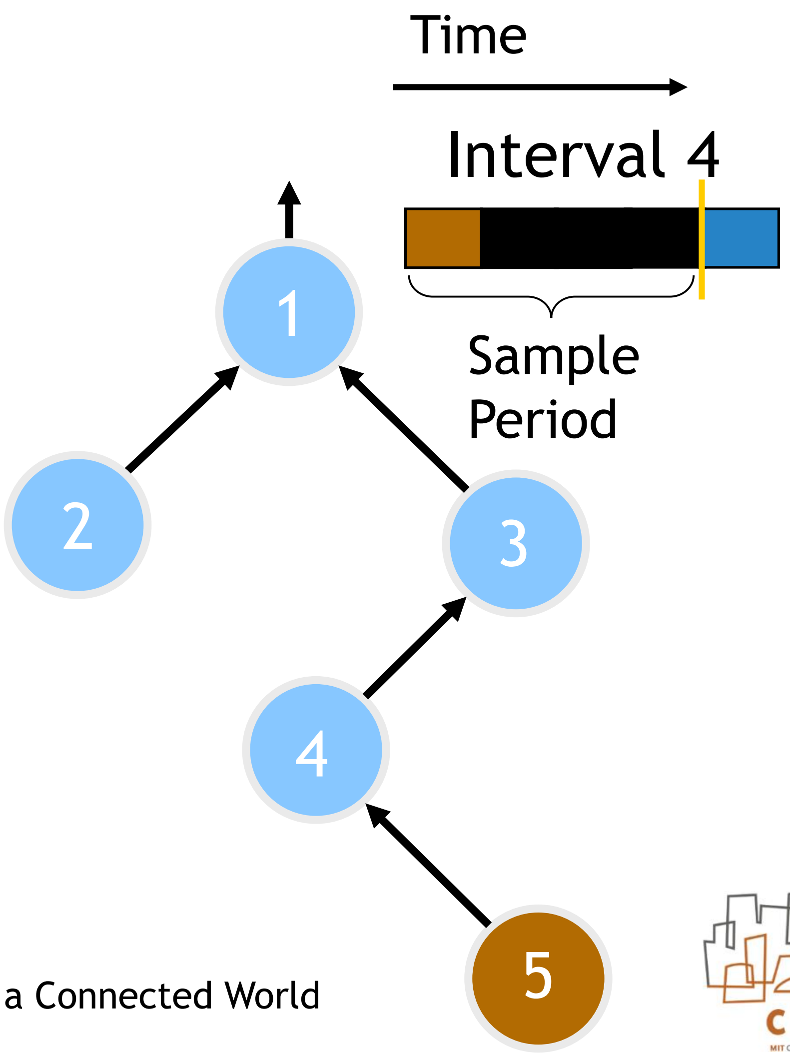


ILLUSTRATION: IN-NETWORK DATA PROCESSING

SELECT COUNT(*) FROM sensors

Sensor #

Interval #

	1	2	3	4	5
4					1
3				2	
2					
1					
4					

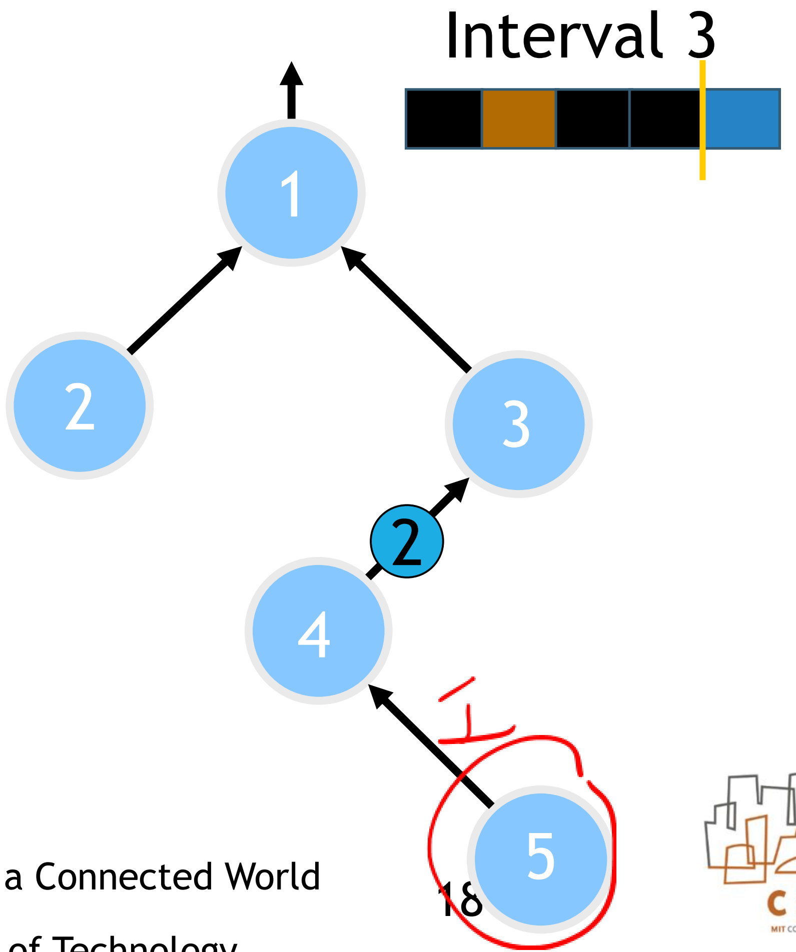
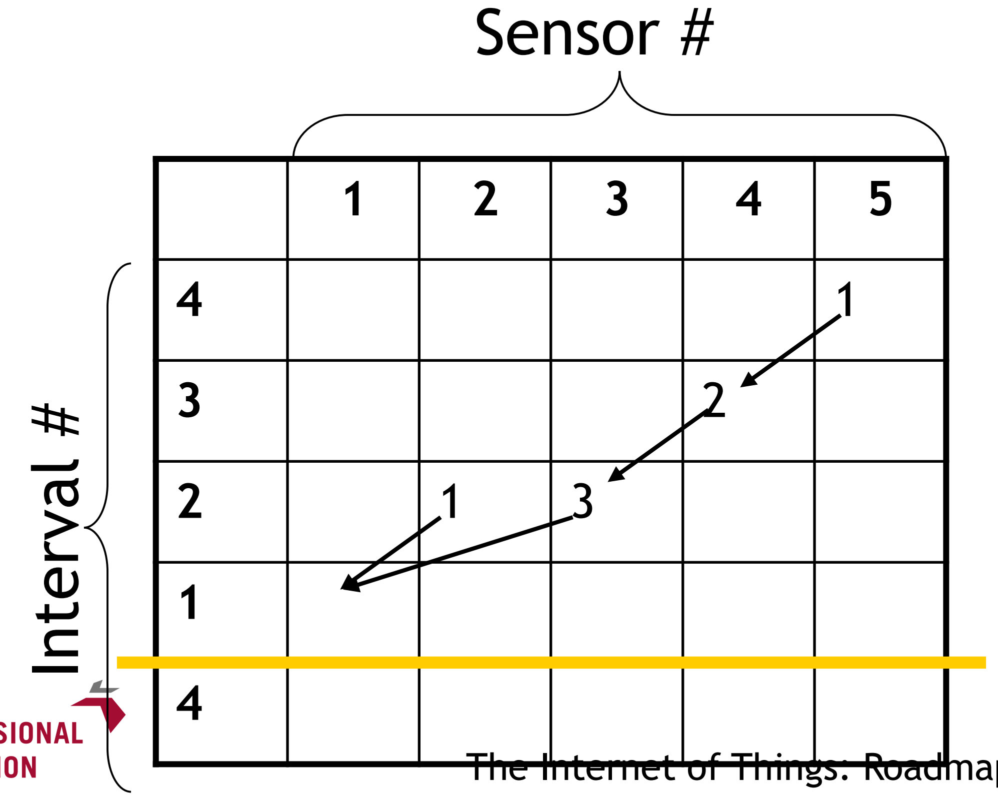


ILLUSTRATION: IN-NETWORK DATA PROCESSING

SELECT COUNT(*) FROM sensors



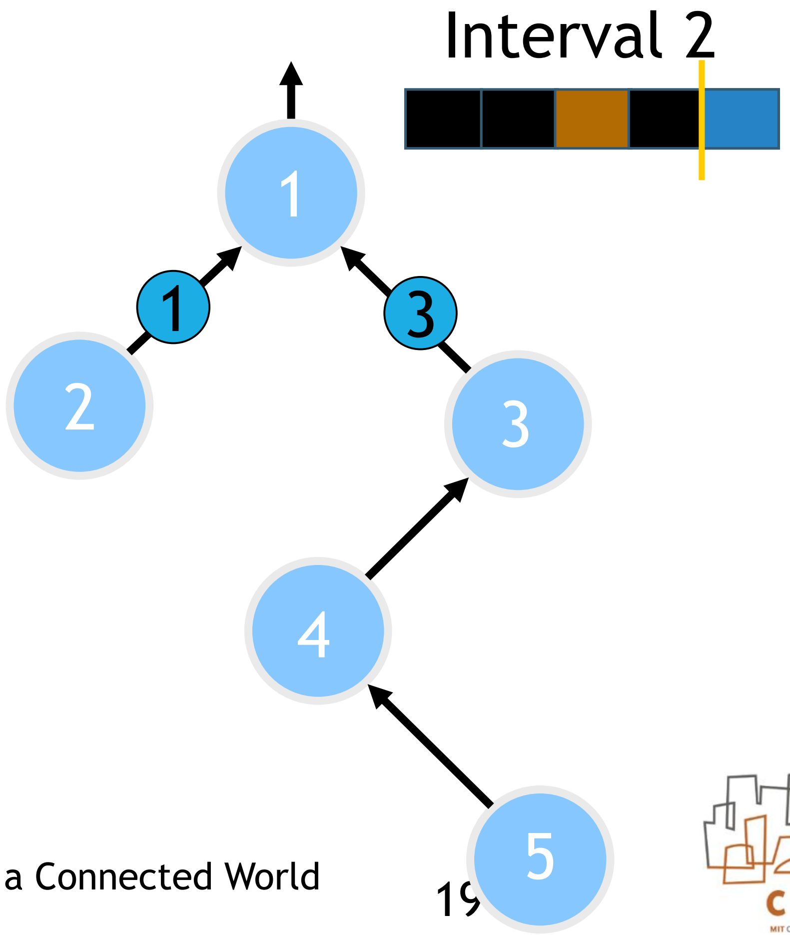


ILLUSTRATION: IN-NETWORK DATA PROCESSING

SELECT COUNT(*) FROM sensors

Sensor #

Interval #	1	2	3	4	5
4					1
3				2	
2		1	3		
1	5				
4					

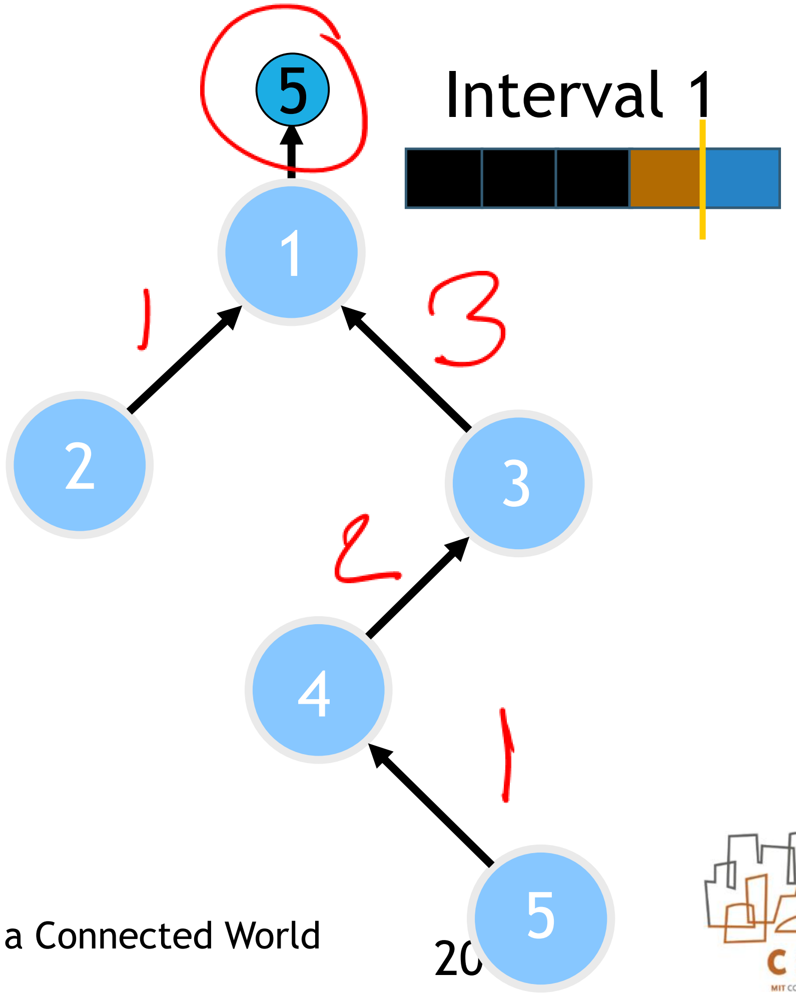


ILLUSTRATION: IN-NETWORK DATA PROCESSING

SELECT COUNT(*) FROM sensors

Sensor #

Interval #

	1	2	3	4	5
4					1
3				2	
2		1	3		
1	5				
4					1

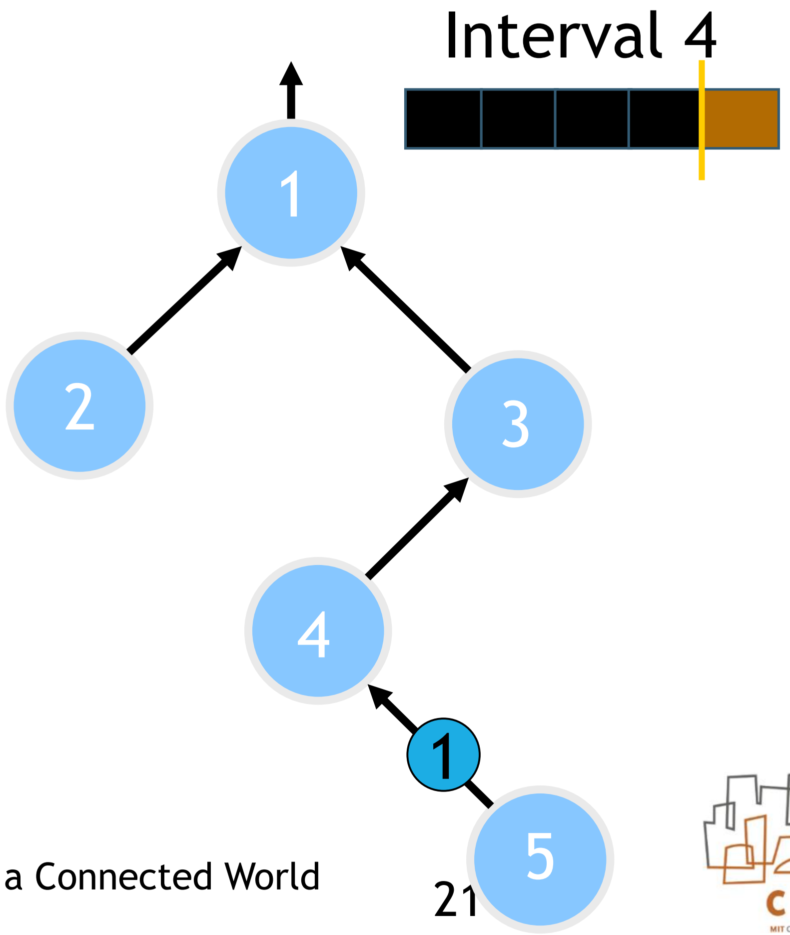


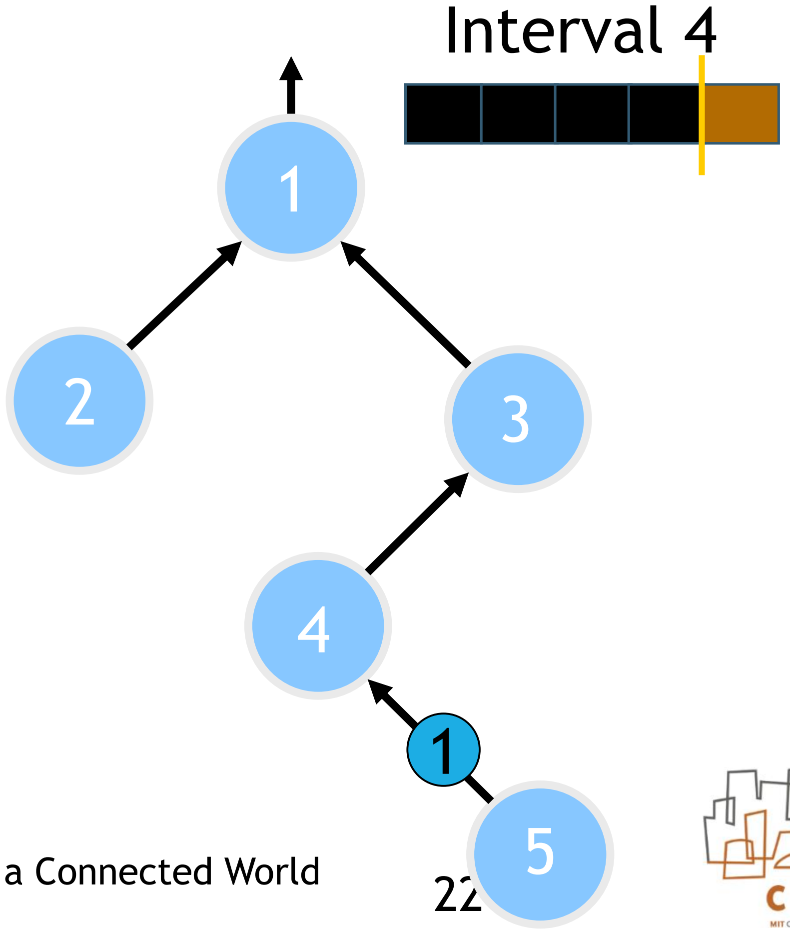
ILLUSTRATION: IN-NETWORK DATA PROCESSING

Nodes can sleep most of the time
Each node transmits only one COUNT

SELECT COUNT(*) FROM sensors

Sensor #

Interval #	1	2	3	4	5
4	zzz	zzz	zzz		1
3	zzz	zzz		2	zzz
2		1	3	zzz	zzz
1	5	zzz	zzz	zzz	zzz
4	zzz	zzz	zzz		1



BANDWIDTH

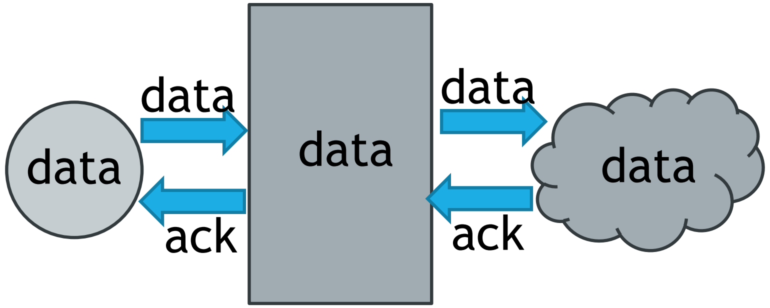
Intermittent or low-rate (e.g., Bluetooth LE) radios limit what you can collect

(See previous module on radio technologies)

Considerations

1. Continuous monitoring vs alerting
2. Buffering to mitigate rate variations, disconnectivity

HANDLING INTERMITTENCY



Radio connectivity can drop out

➔ Local buffering, end-to-end acknowledgement

End-to-end principle: data should only be removed from devices when it has been delivered to permanent storage

EXAMPLE: DRIVEWELL + TAG APPLICATION

Application collects driving metrics (hundreds of KB/hour) and does real time impact alerting

Buffers metrics to files on WiFi, uploads when WiFi is available

- Data only removed from tag once processed data reaches phone

Immediately relays impacts to server via 3G

- Need to be buffered as well

This design:

1. *Limits overall power & 3G bandwidth consumption*
2. *Meets application requirements*

STORAGE

Flash storage is (usually abundant)

Main reason to limit stored data is to reduce power consumption & bandwidth

Example: In DriveWell, we were able to obtain a 4x reduction in stored data using on-device processing, compression, and judicious dropping of data without affecting quality

RESOURCE LIMITATION SUMMARY

Energy and bandwidth restrict what can be collected

IoT design is about engineering tradeoffs, e.g.,

- battery life vs quantity of data
- latency vs resolution

Must plan for edge cases, e.g., disconnection, battery failure

MANAGING MISSING AND NOISY DATA

IOT DATA IS APPROXIMATE

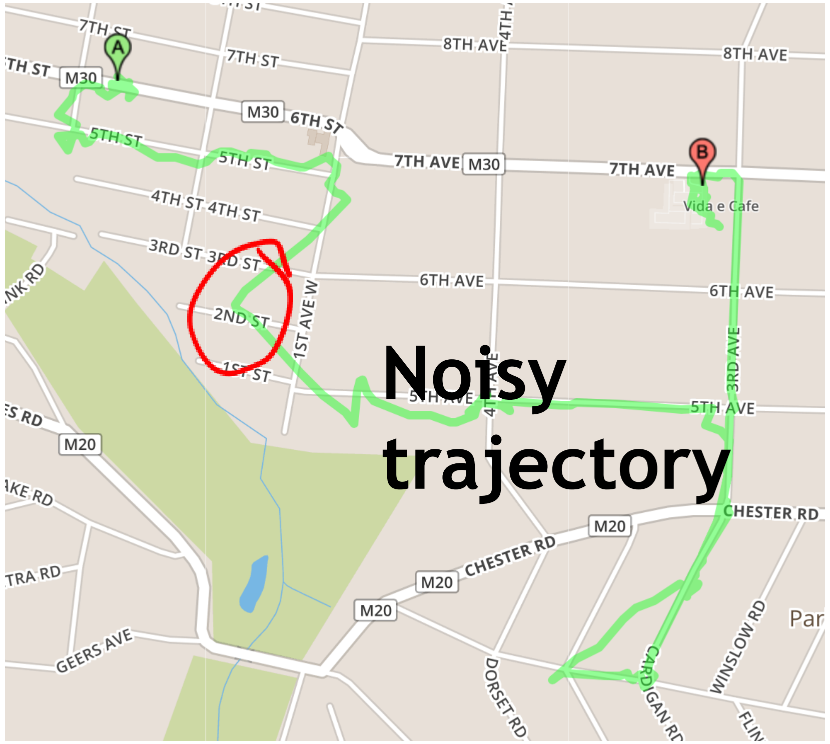
Data arrives at discrete times

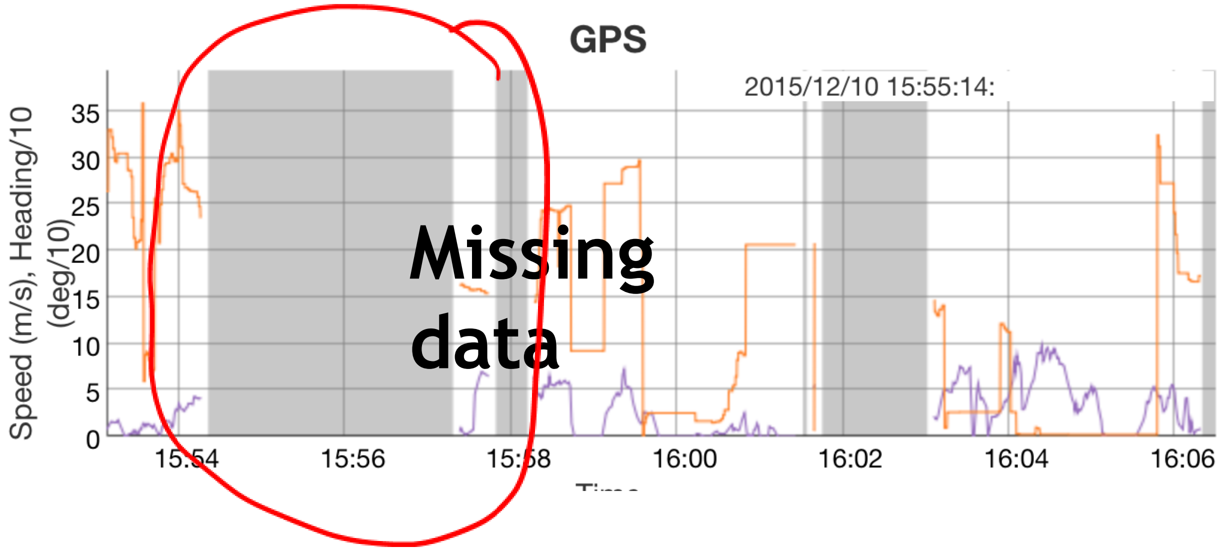
Data is of limited precision

Data can be wrong

EXAMPLE: GPS DATA

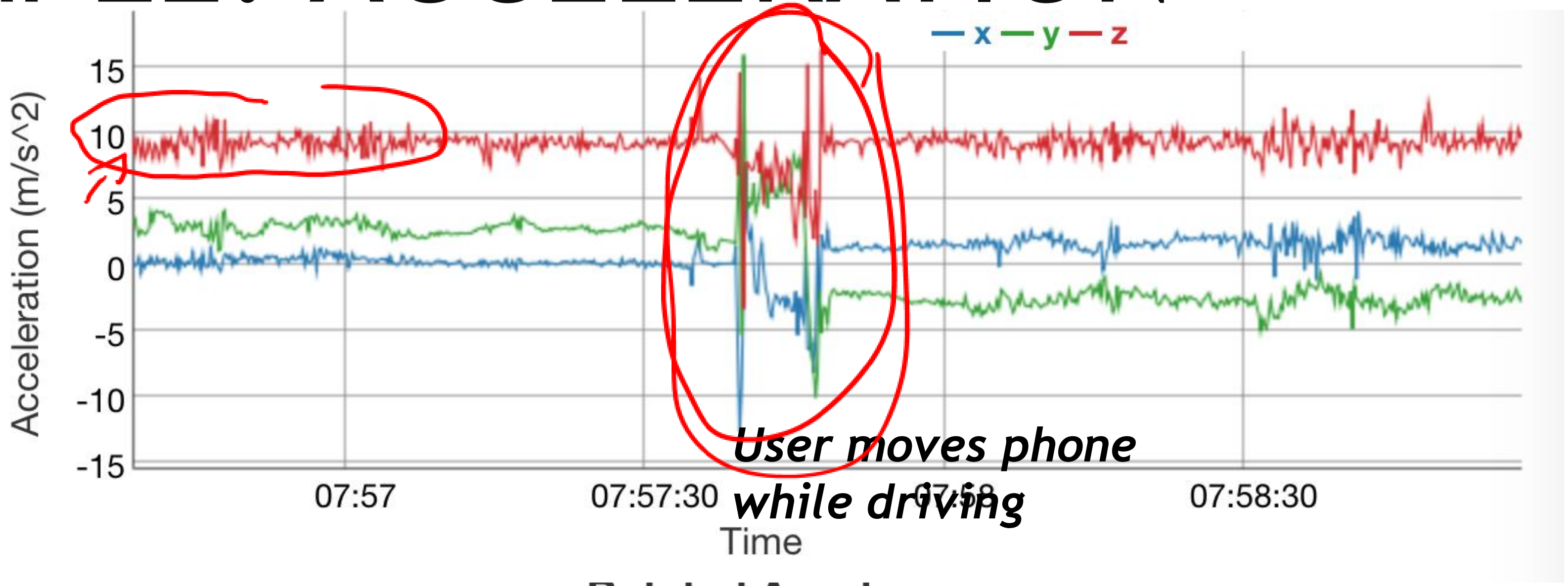
What roads did this device travel on?



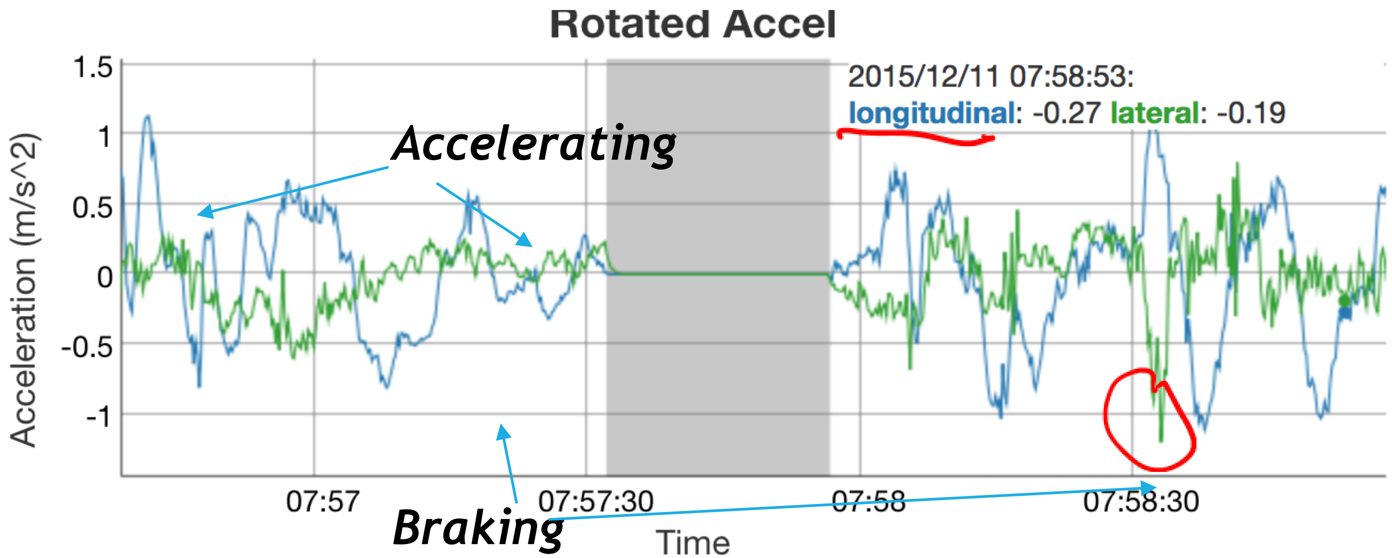


EXAMPLE: ACCELERATION

Raw Signal



Cleaned Signal



BASIC TECHNIQUES FOR DEALING WITH NOISE & MISSING DATA

Interpolation, extrapolation

- E.g., regression

Smoothing

- E.g., curve fitting, low-pass filters

Alignment

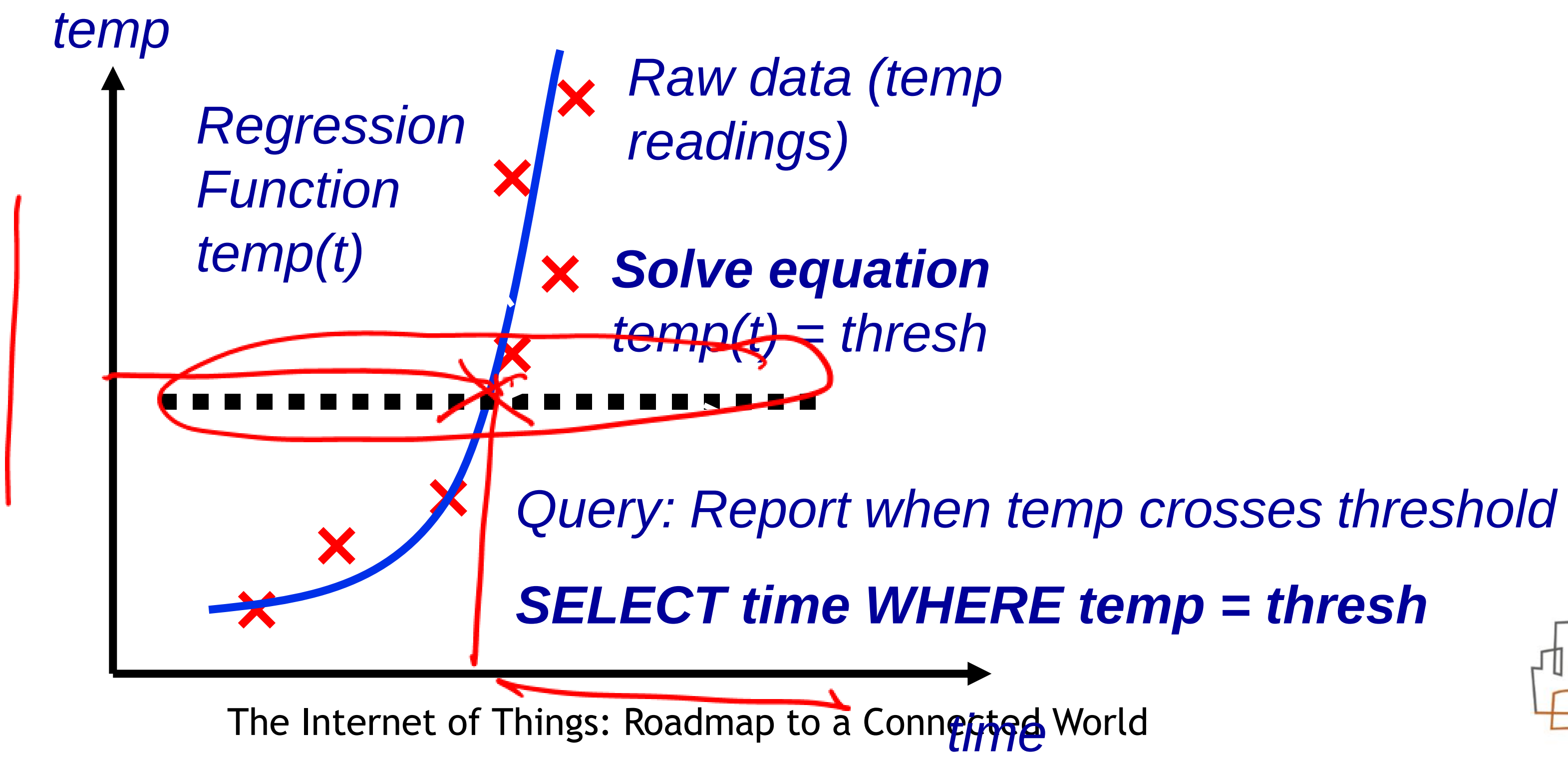
- E.g., auto-correlation, dynamic time warping

Many possible methods!

EXAMPLE 1: FUNCTIONDB

Thiagarajan et al, "Querying Continuous Functions in a Database System"

Database that allows users to fit *continuous functions* to raw data, query data represented by these functions using a SQL-like interface



FUNCTIONDB: SYSTEM ARCHITECTURE

User Query

Query Result

Algebraic Query Processor (Operates on algebraic representation)

Raw Data

X	Y
1	1.1
3	3.1
4	3.9
…	…

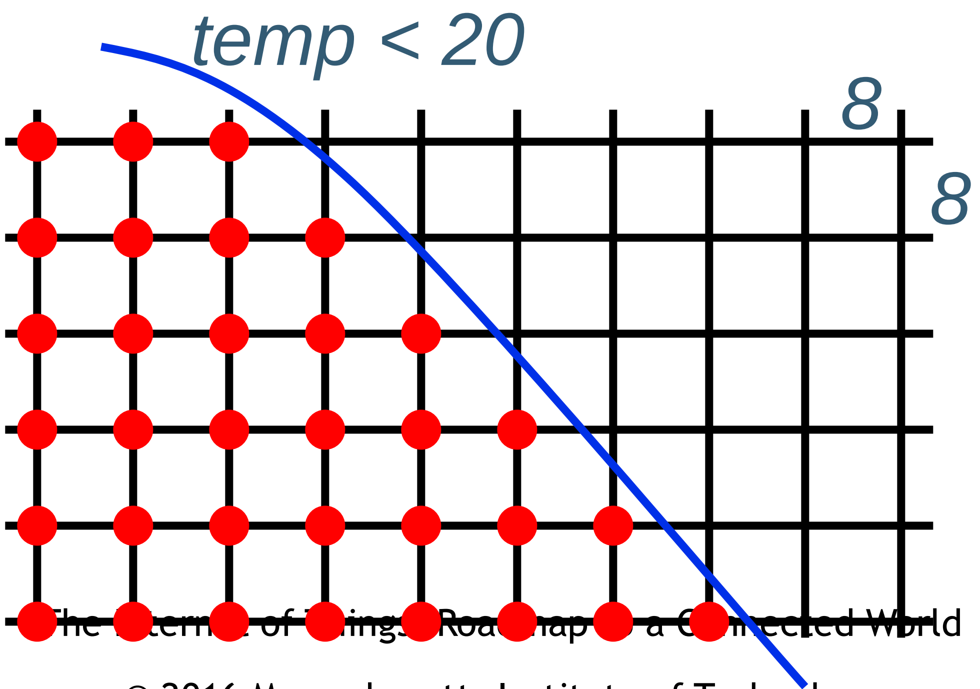
User fits or imports functions

"Function Table" (Storing Piecewise Model)

Start	End	Slope	Intercept
1	8	1	0
8	12	2	-8

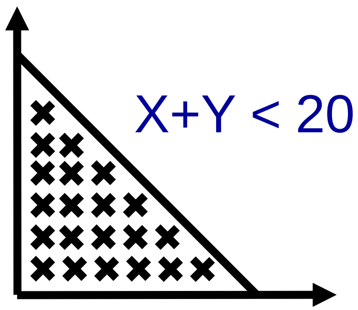
QUERY RESULTS

- *Grid semantics: all queries yield discrete points sampled at user-specified interval ("grid size")*
- *SELECT x,y WHERE temp < 20 GRID x 8, y 8*



SELECT * WHERE TEMP < 20 GRID X 8, Y 8
EFFICIENT ALGEBRAIC IMPLEMENTATION

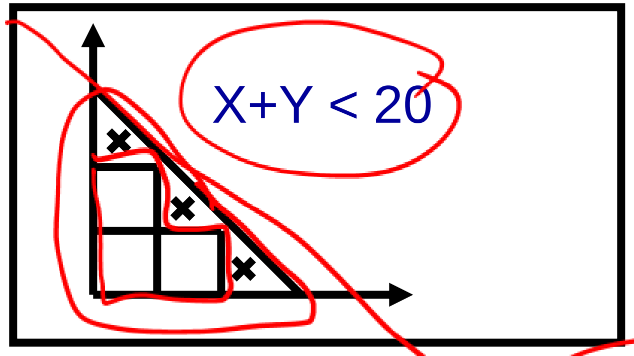
Gridded Result



User

Grid

Hypercubes+Boundary Points



Approx

X:[0,20) , Y:[0,20) , Temp = X+Y , X+Y-20 < 0

Substitute

X:[0,20) , Y:[0,20) , Temp = X+Y , Temp < 20

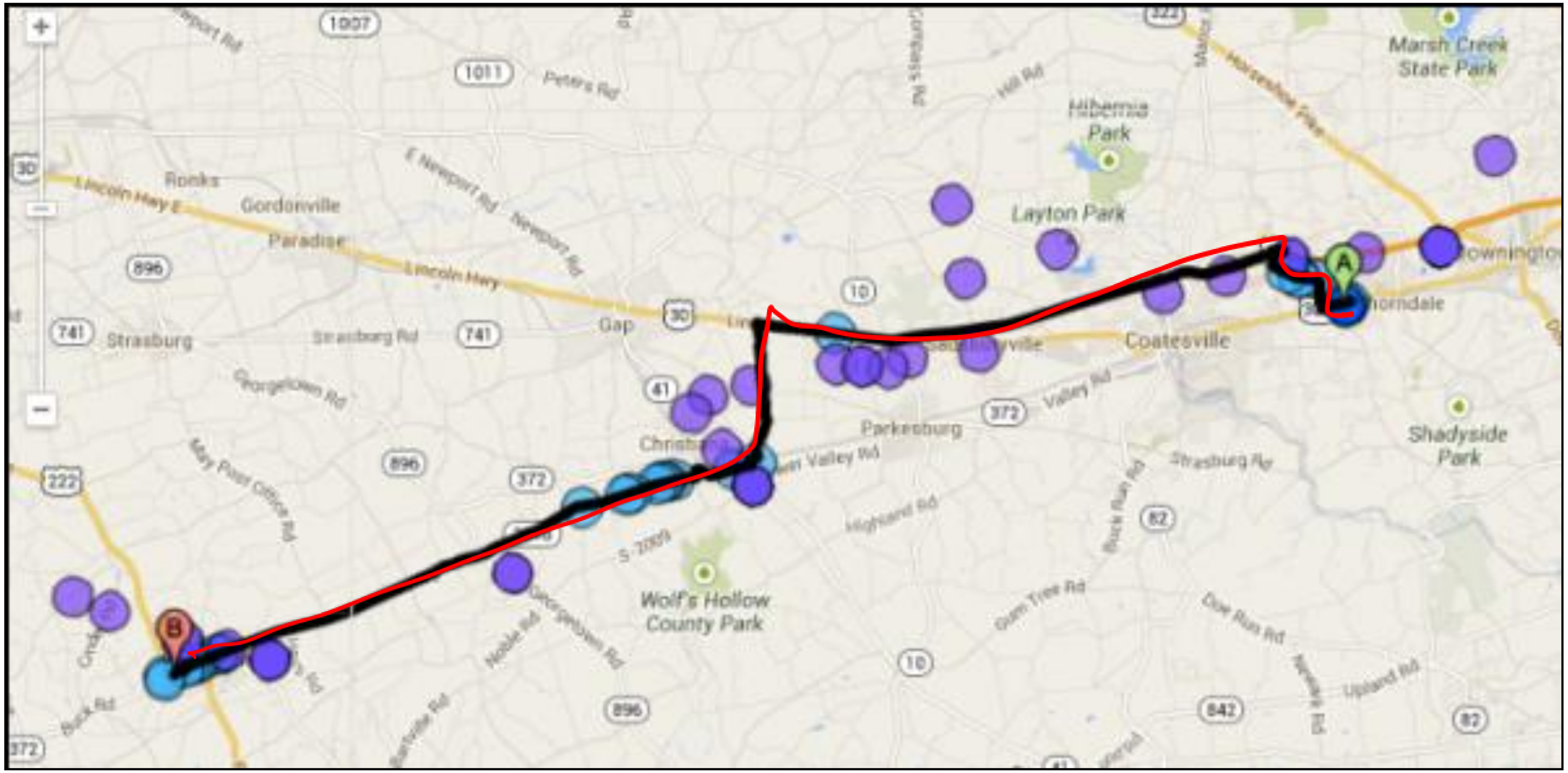
Symbolic Filter (Temp< 20)

X:[0,20) , Y:[0,20) , Temp = X+Y

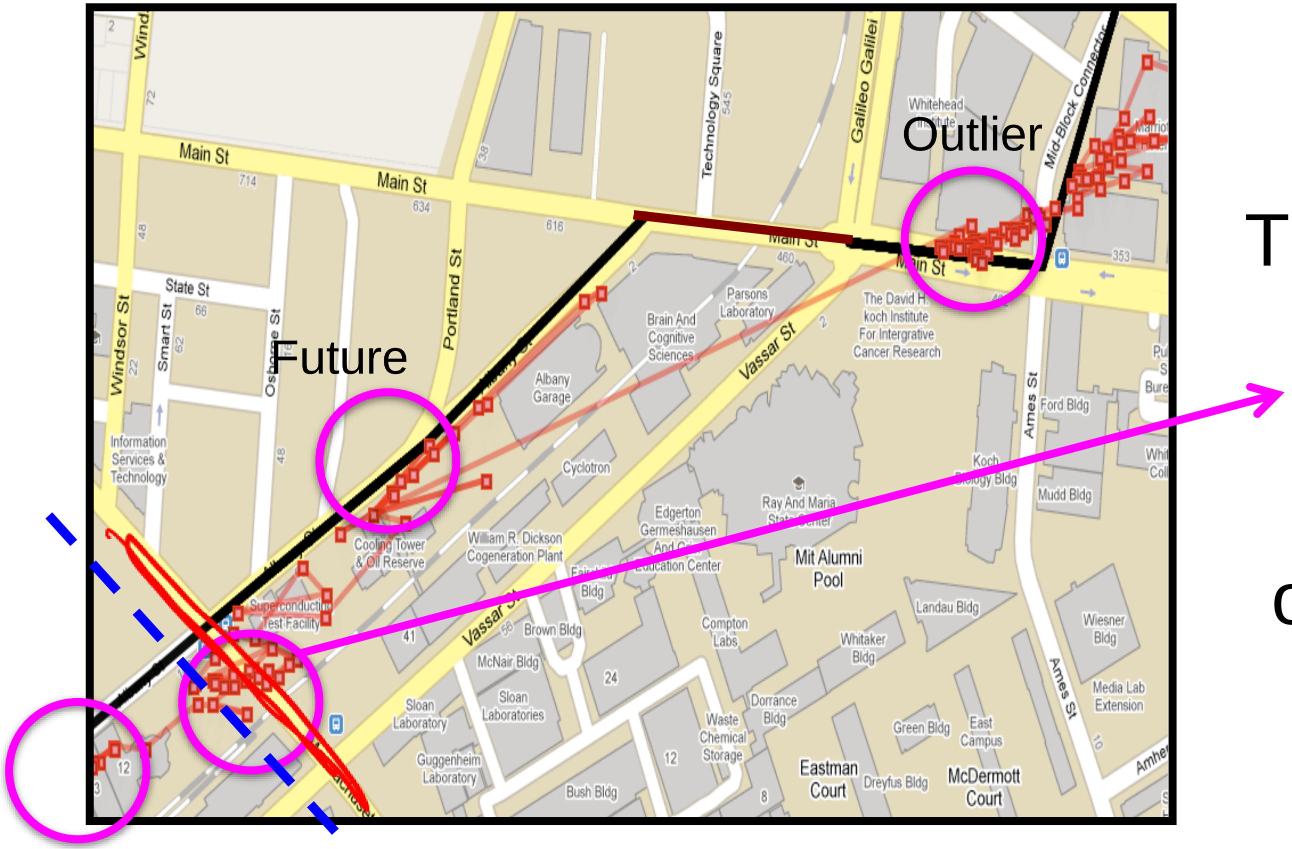
Function Table Temp = F(X,Y)

EXAMPLE 2: MAPMATCHING

Thiagarajan et al, “VTrack: Accurate, Energy-aware Road Traffic Delay Estimation Using Mobile Phones”



MAPMATCHING INTUITION



Past

The closest road to a position sample is *not* where it originally came from

- Exploit both previous and future location info
- Don’t overly weight *any one* location sample

PROFESSIONAL EDUCATION

CSAIL

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

REAL WORLD PROBLEMS

Sporadic

Inaccurate, with varying accuracy

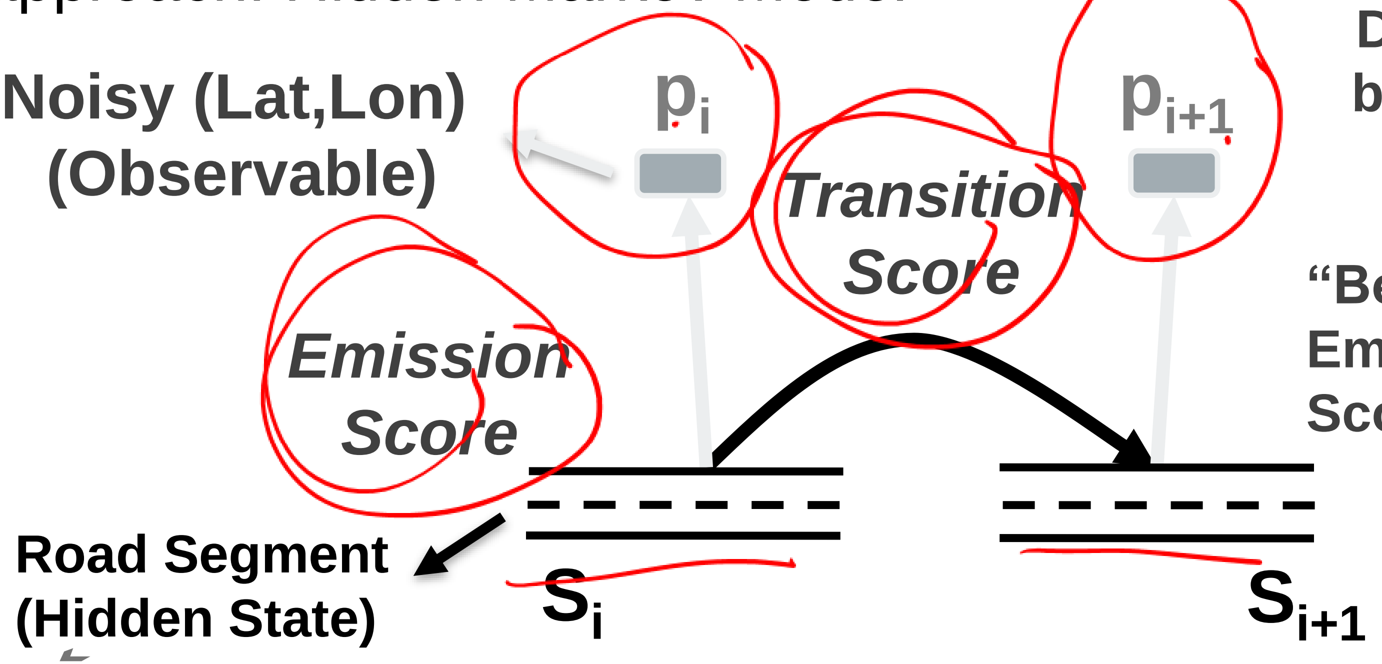
Clustered occasionally

***Real-world*: Non-Markovian**

Observations in clusters

Sporadic data => loops in output

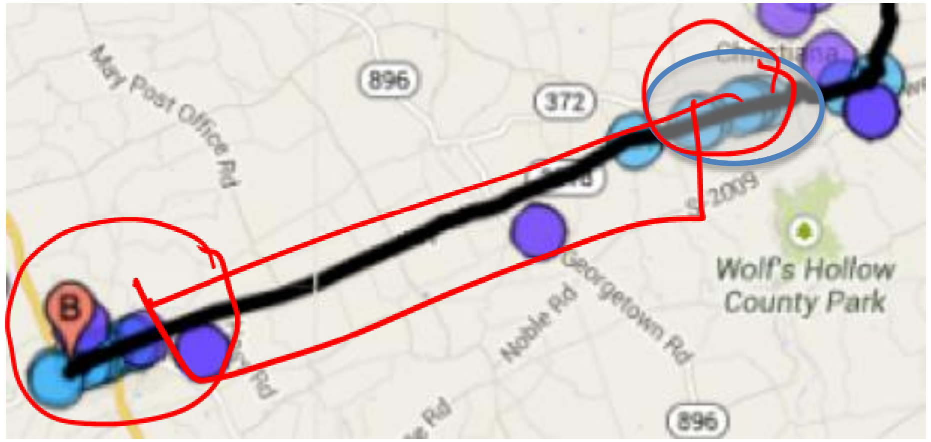
Usual approach: Hidden Markov Model



Dynamic program finds best state sequence (cf. Viterbi)

"Best" ➔ Max Product of Emission and Transition Scores

NEW METHOD: FULL PATHS, THEN RANK



Clustered observations not independent!

Input observations → Cluster → Sequence of feasible segment sets

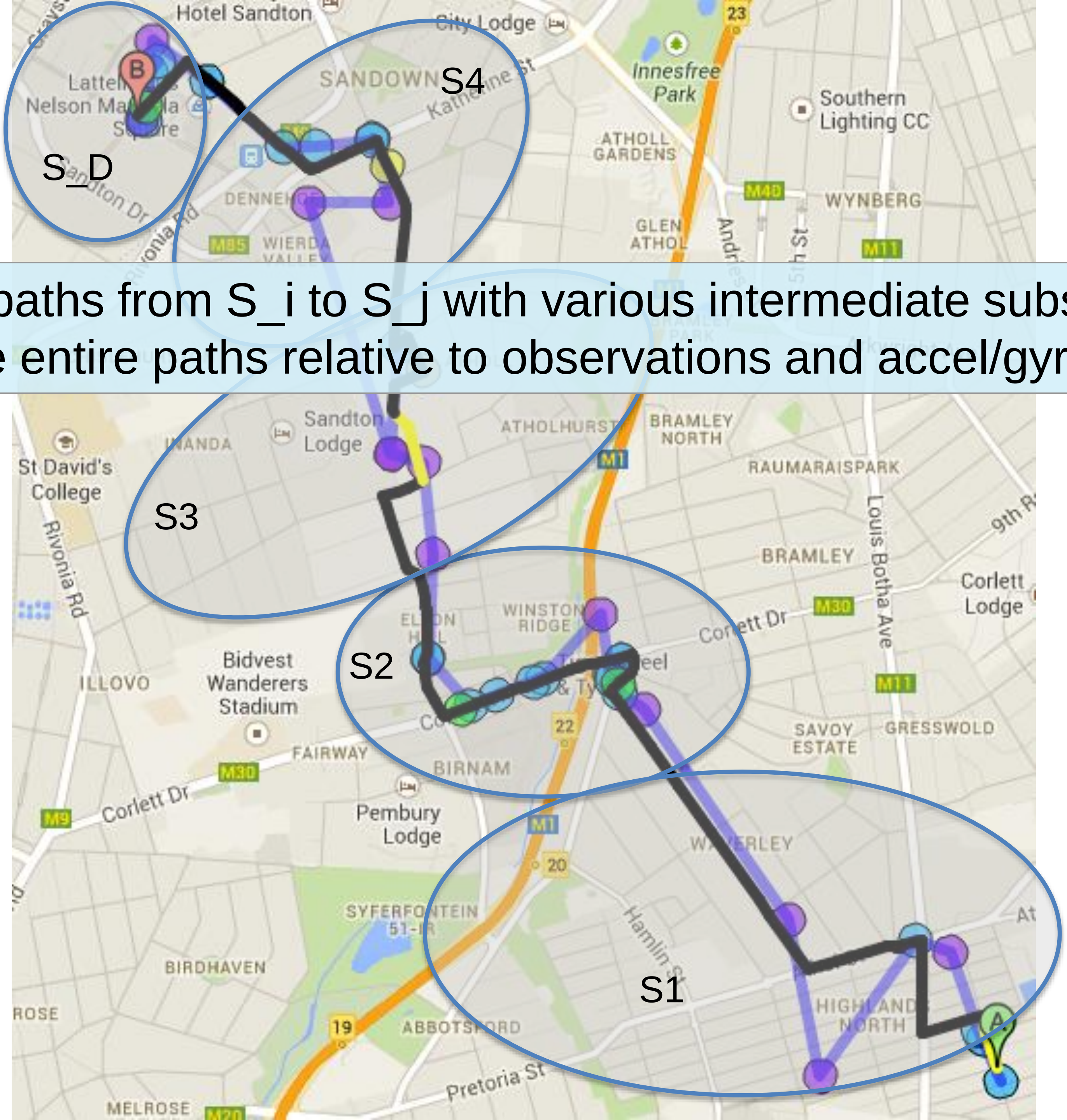
Insight: use a "holistic" method, don't build path incrementally

Idea 1: Score entire path relative to observations

Idea 2: Given segment sets S_i and S_j, compute paths between each segment in S_i and each segment in S_j, traversing various subsets of the intermediate segment sets (dynamic programming)

Idea 3: Use accel and gyro data for rest & turn hints

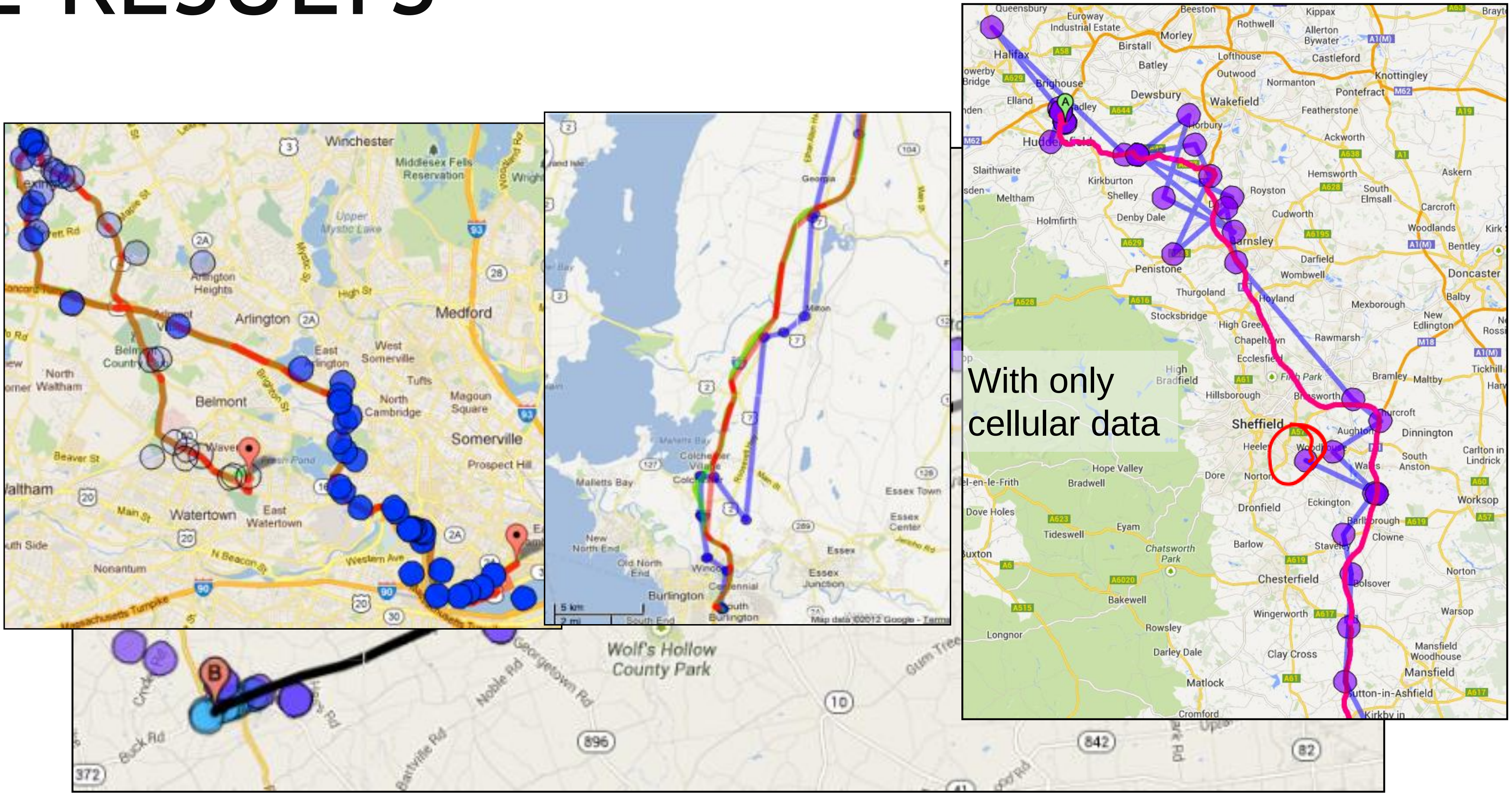
1. Find paths from S_i to S_j with various intermediate subsets
2. Score entire paths relative to observations and accel/gyro data



PROFESSIONAL EDUCATION

CSAIL

SOME RESULTS



MISSING & NOISY DATA SUMMARY

IoT data processing often involves removing noise, predicting missing values

Many methods for doing this:

- Interpolation
- Extrapolation
- Smoothing

Two use cases:

- FunctionDB for answering queries over functions
- Vtrack for noisy position data

DETECTING OUTLIERS AND ANOMALIES

IOT IS ABOUT UNUSUAL EVENTS

For example

- Rapid detection of equipment failure or degradation
    - Pipes, data centers, medial equipment, etc
- Physical tampering in some space
    - E.g., doors, windows, other security apps
- Monitoring of people or behavior
    - I.e., a medical patient stopped breathing

These are outliers, or anomalies

ANOMALY DETECTION DESIDERATA

Automatically flag outliers

- e.g., devices 1,14, and 27 are acting weird

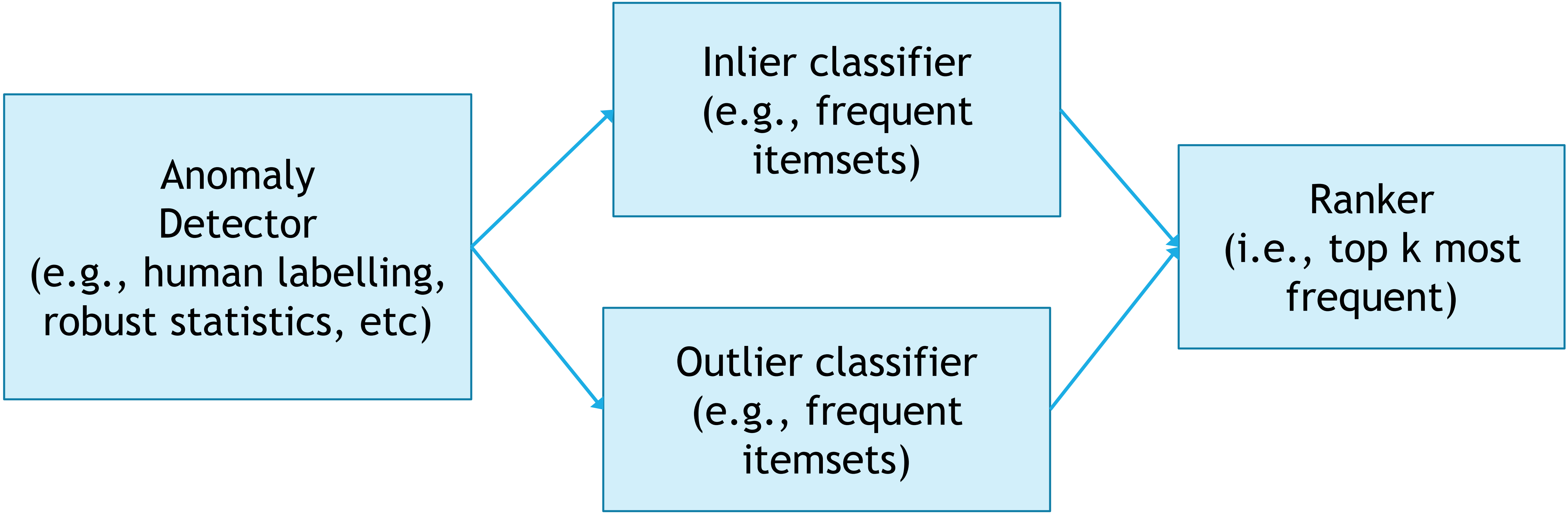
Identify common properties of outliers

- e.g., weird devices are all running Android 4.5

Rank & triage outlier classes

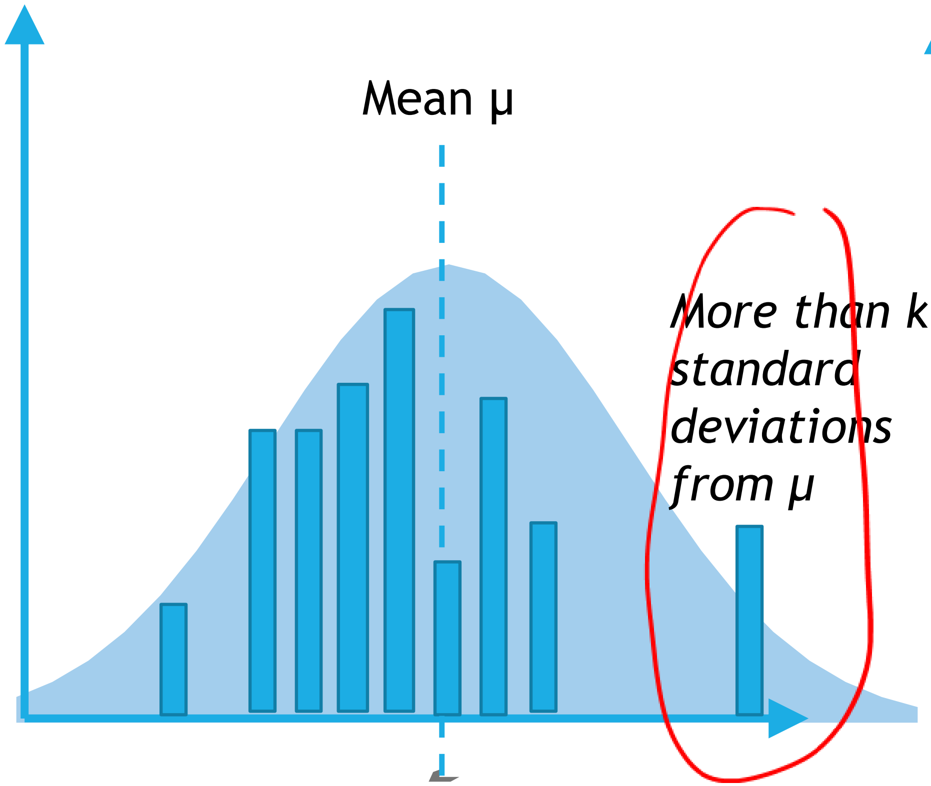
- e.g., by severity & number of affected users

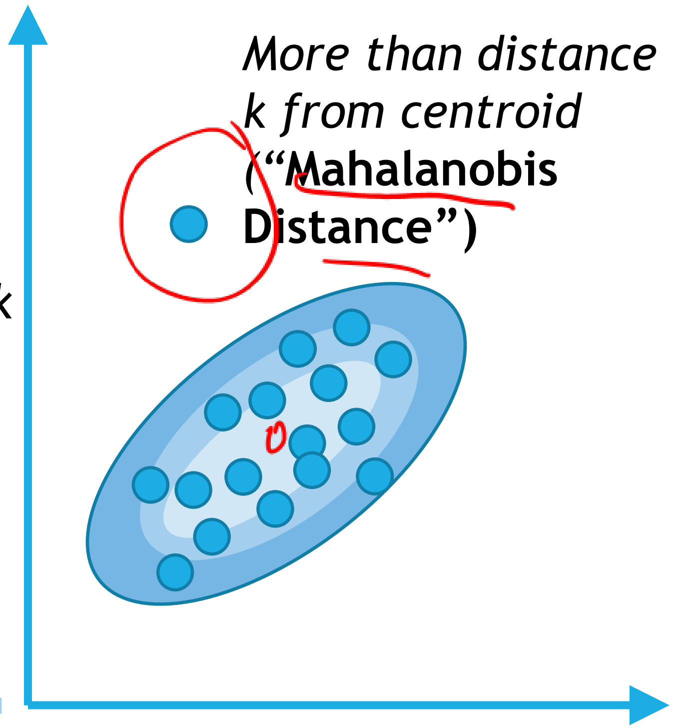
ANOMALY DETECTION & EXPLANATION WORKFLOW



ANOMALY DETECTION

Many different ways to detect anomalies



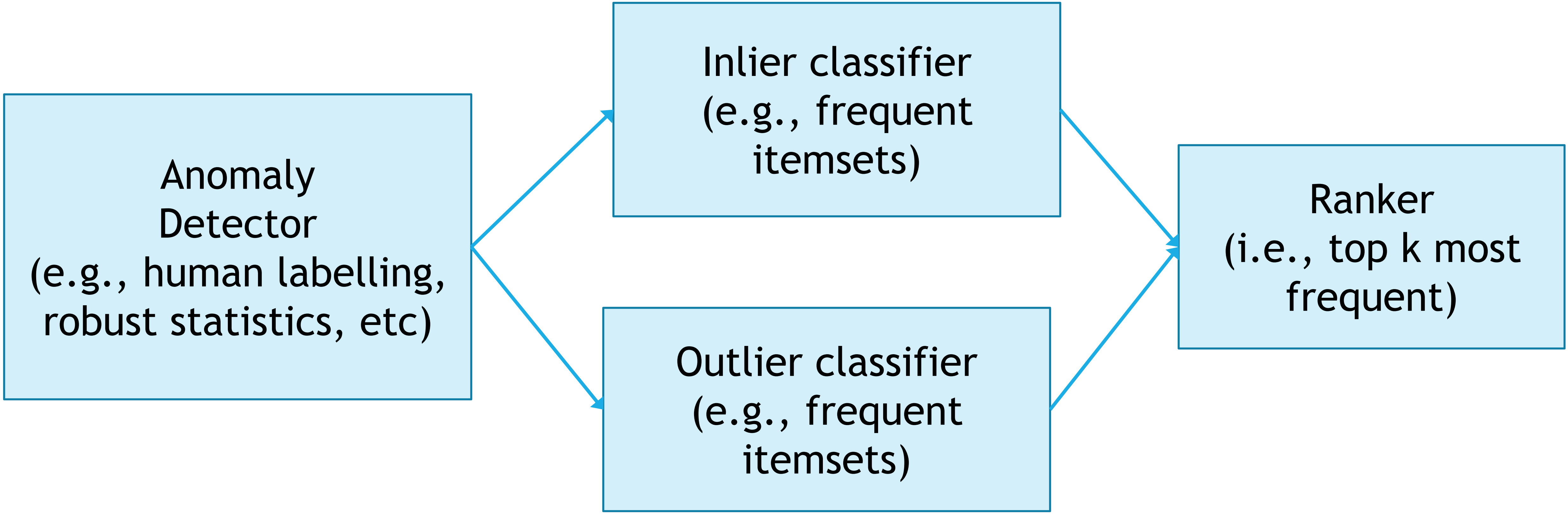


<u>Rules:</u>

Temperature < 100°C

No employee makes more than his manager

ANOMALY EXPLANATION



ANOMALY EXPLANATION

Goal of explanation is to find a description of anomalous records

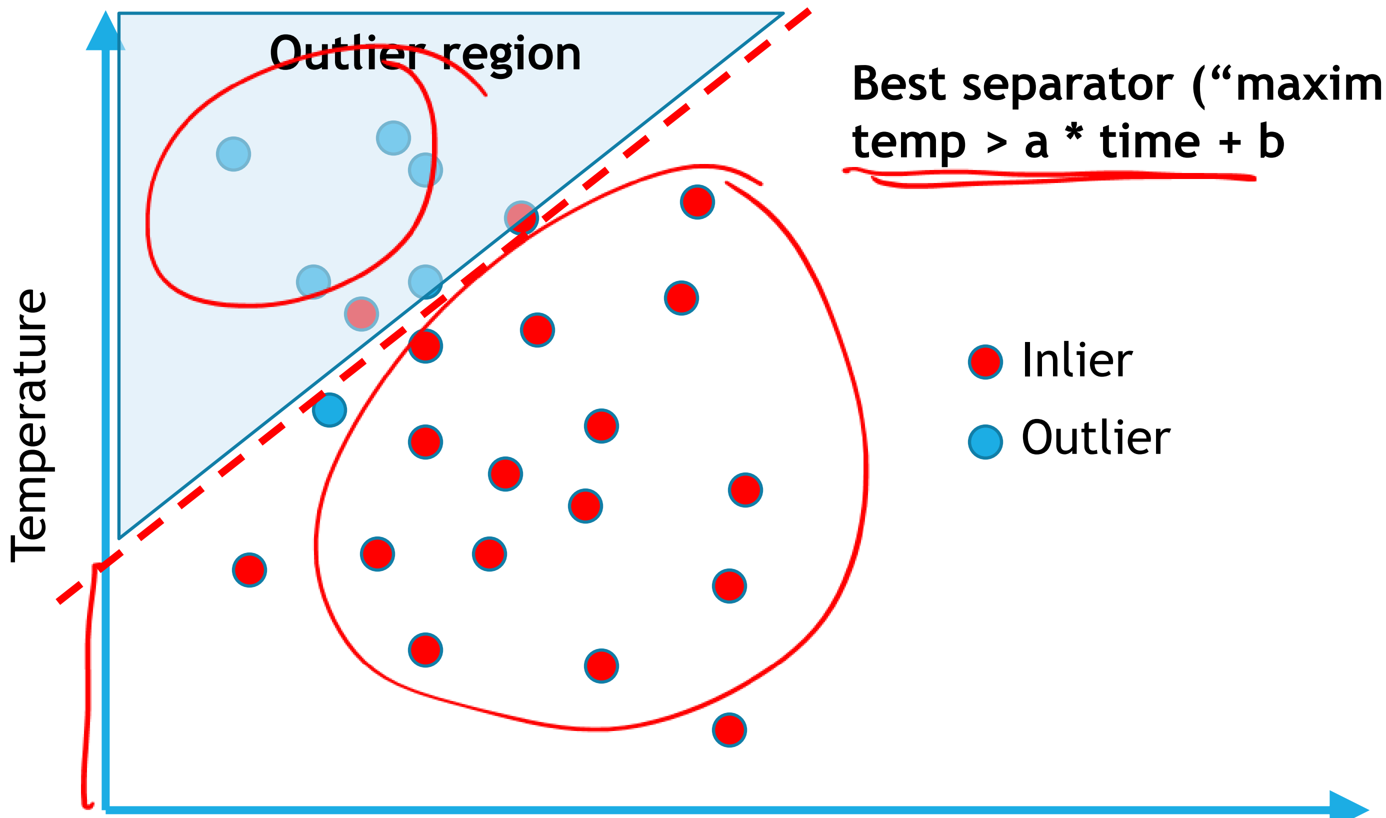
In complex data sets, every record has tens or hundreds of attributes

Two methods:

1. Classifiers: e.g., Decision trees / SVM
2. Frequent item sets

uploadtime
deviceid
driveid
state
dest_server
companyid
hardware_manufacturer
hardware_model
hardware_bootloader
hardware_build
hardware_carrier
android_fw_version
android_api_version
android_codename
android_baseband
raw_hardware_string
raw_os_string
utc_offset_with_dst
app_version
file_format
start_reason
stop_reason
previous_driveid
userid
tag_mac_address
tag_trip_number
primary_driver_app_user_id
tag_last_connection_number
gps_points_lsh_key_1
gps_points_lsh_key_2
gps_points_lsh_key_3
hidden_by_support
dataset_id
uploadtime
runtime
trip_start

TECHNIQUE 1: CLASSIFICATION (VIA SUPPORT VECTOR MACHINES)



Best separator ("maximum margin hyperplane")
temp > a * time + b

TECHNIQUE 2: FREQUENT ITEMSET MINING

Works for categorical data, or binned continuous data; example:

Outliers

{iPhone6, Canada}
{iPhone6, USA}
{iPhone5, Canada}
{iPhone6, USA}
{iPhone5, Canada}

Inliers

{iPhone6, USA}
{iPhone6, USA}
{iPhone5, USA}
{iPhone6, USA}
{iPhone5, USA}
{iPhone6, USA}
{iPhone6, USA}
{iPhone5, USA}

Looks like Canada may have a problem!

TECHNIQUE 2: FREQUENT ITEMSET MINING

Outliers
{iPhone6, USA}
{iPhone6, Canada}
{iPhone5, Canada}
{iPhone6, USA}
{iPhone5, Canada}
{iPhone5, Canada}

Inliers
{iPhone6, USA}
{iPhone6, USA}
{iPhone5, USA}
{iPhone6, USA}
{iPhone5, USA}
{iPhone6, USA}
{iPhone6, USA}
{iPhone5, USA}

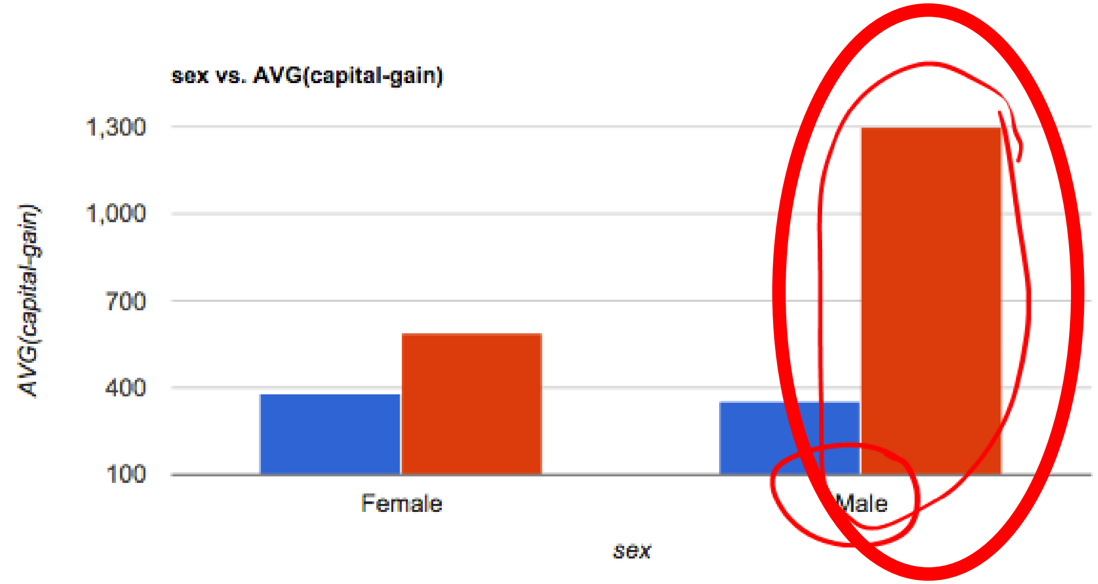
Outliers with support > 2
{iPhone6} (3)
{Canada} (4)
{iPhone5} (3)
{iPhone5, Canada} (3)

Inliers with support > 2
{iPhone6} (5)
{iPhone5} (3)
{iPhone5,USA} (3)
{iPhone6,USA} (5)

Outliers - Inliers = {Canada}, {iPhone5, Canada}

SCORPION: OUTLIER EXPLANATION TOOL

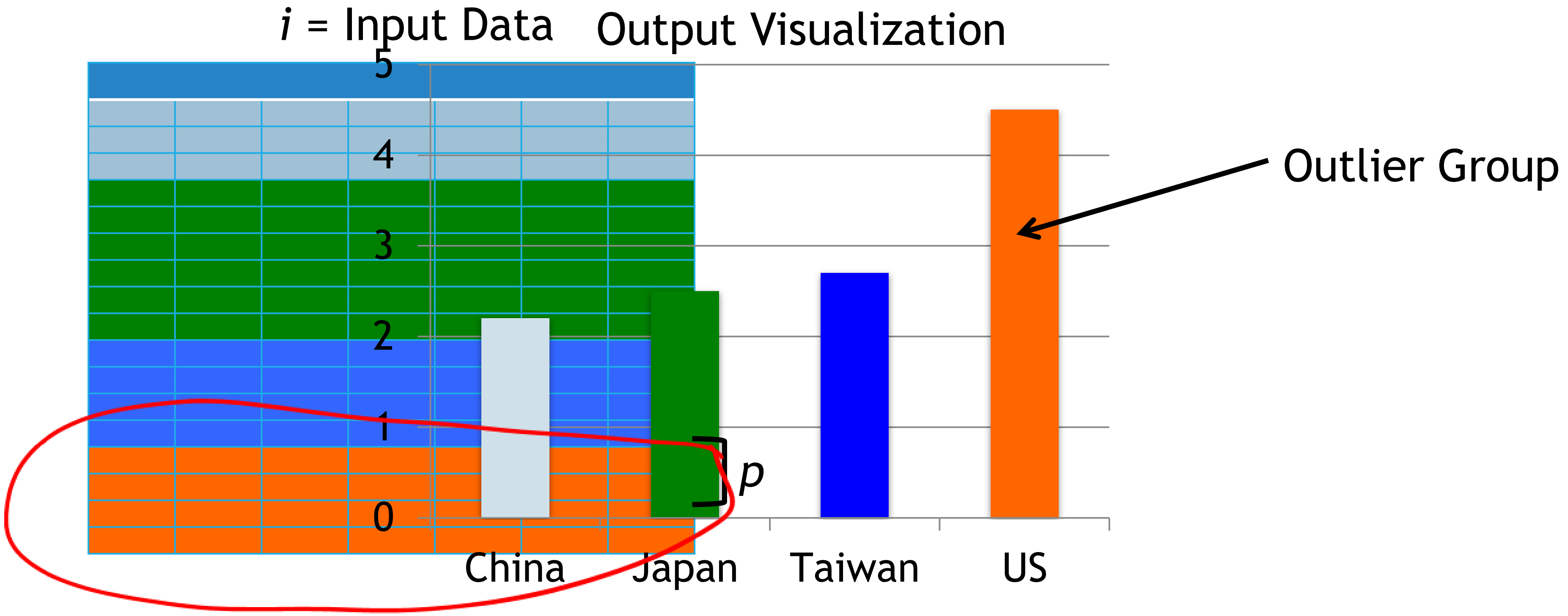
Given an outlier:



Find common properties of points that contributed to outlier to explain *why* outliers exist

DEFINITION OF WHY

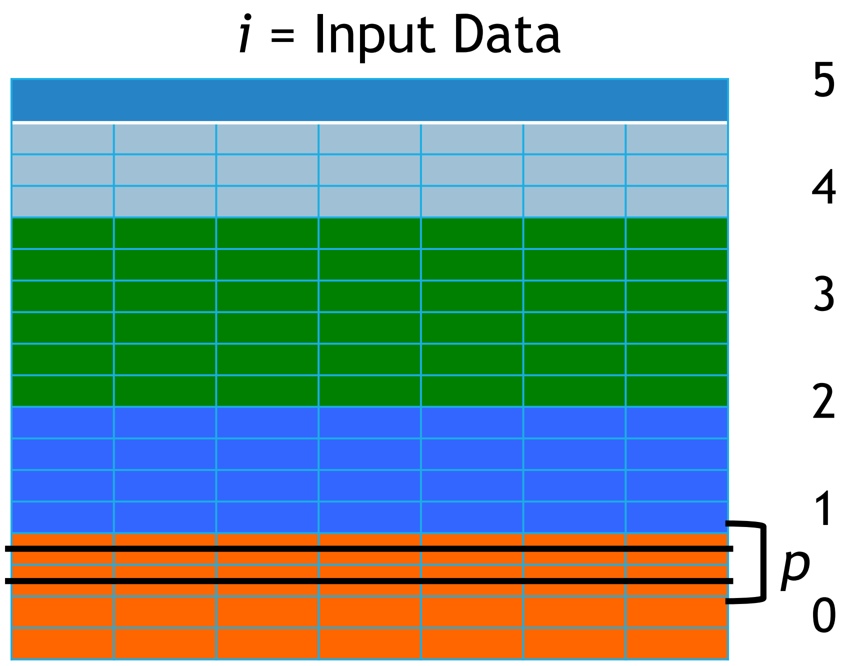
Given an outlier group, find a *predicate* over the inputs that makes the output no longer an outlier.

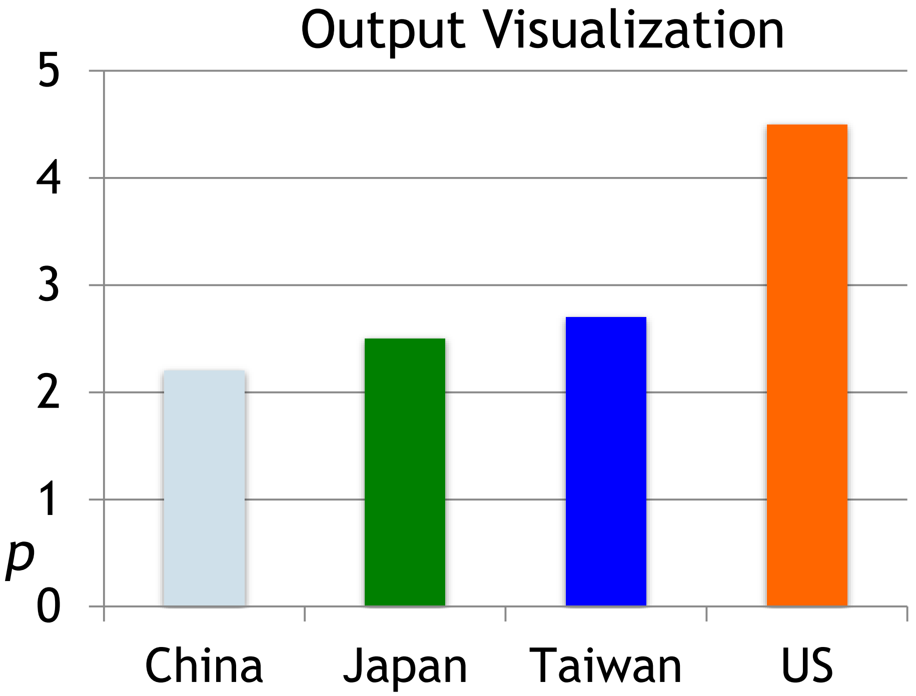


p = predicate

DEFINITION OF WHY

Given an outlier group, find a *predicate* over the inputs that makes the output no longer an outlier.



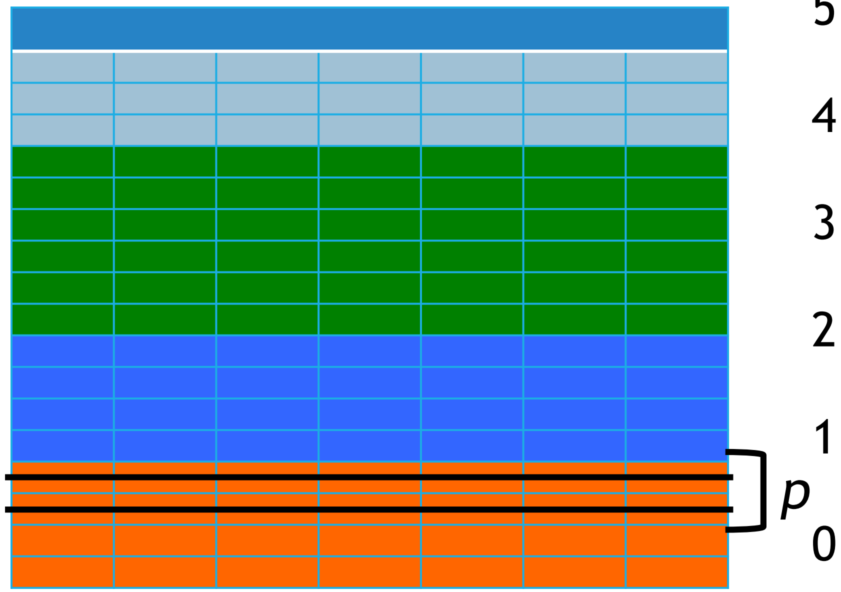


p = predicate

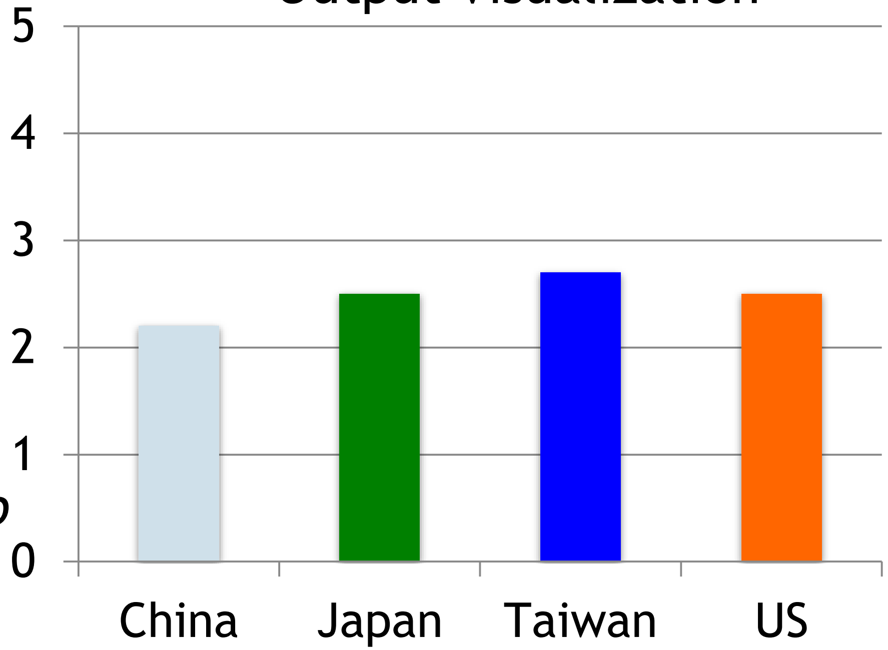
DEFINITION OF WHY

Given an outlier group, find a *predicate* over the inputs that makes the output no longer an outlier.

i = Input Data



Output Visualization

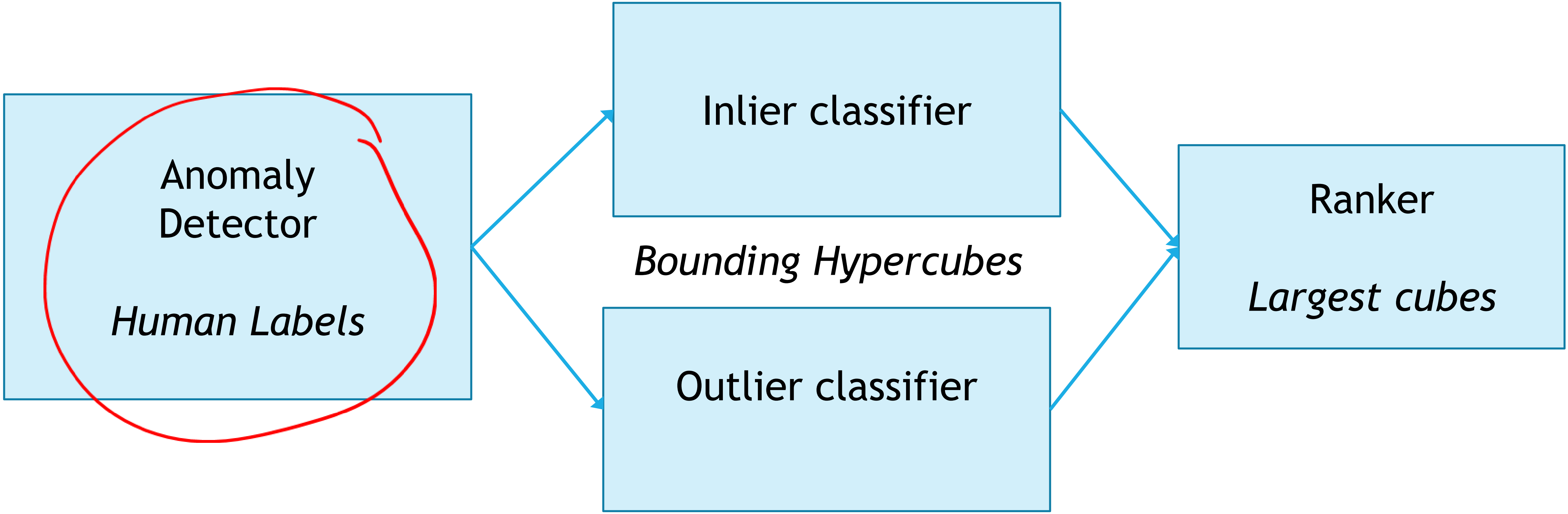


Removing the predicate makes US no longer an outlier

What are common properties of those records? *{Warren Buffet, Tim Cook}*

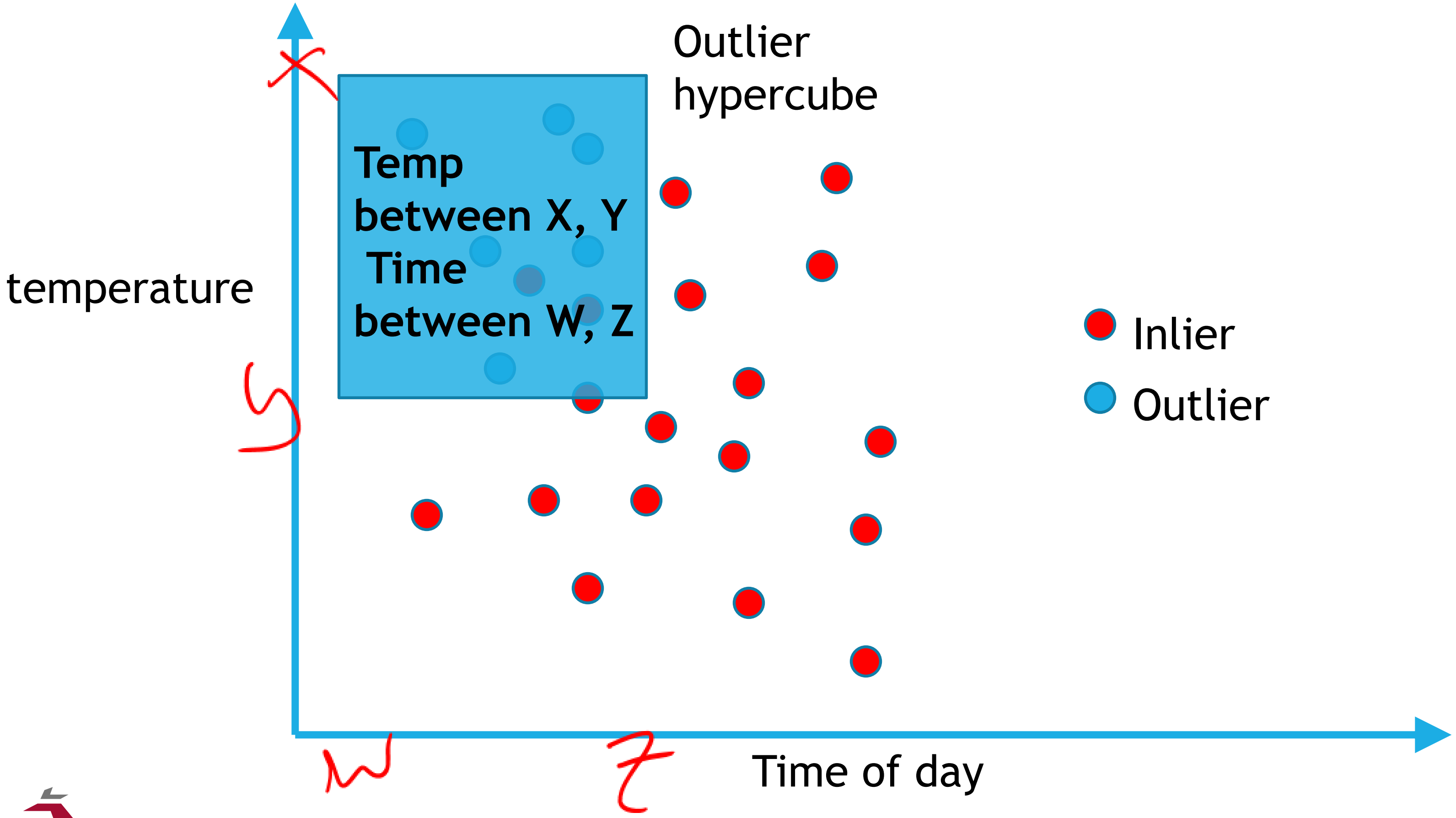
p: Job = CEO

SCORPION ANOMALY DETECTION & EXPLANATION WORKFLOW



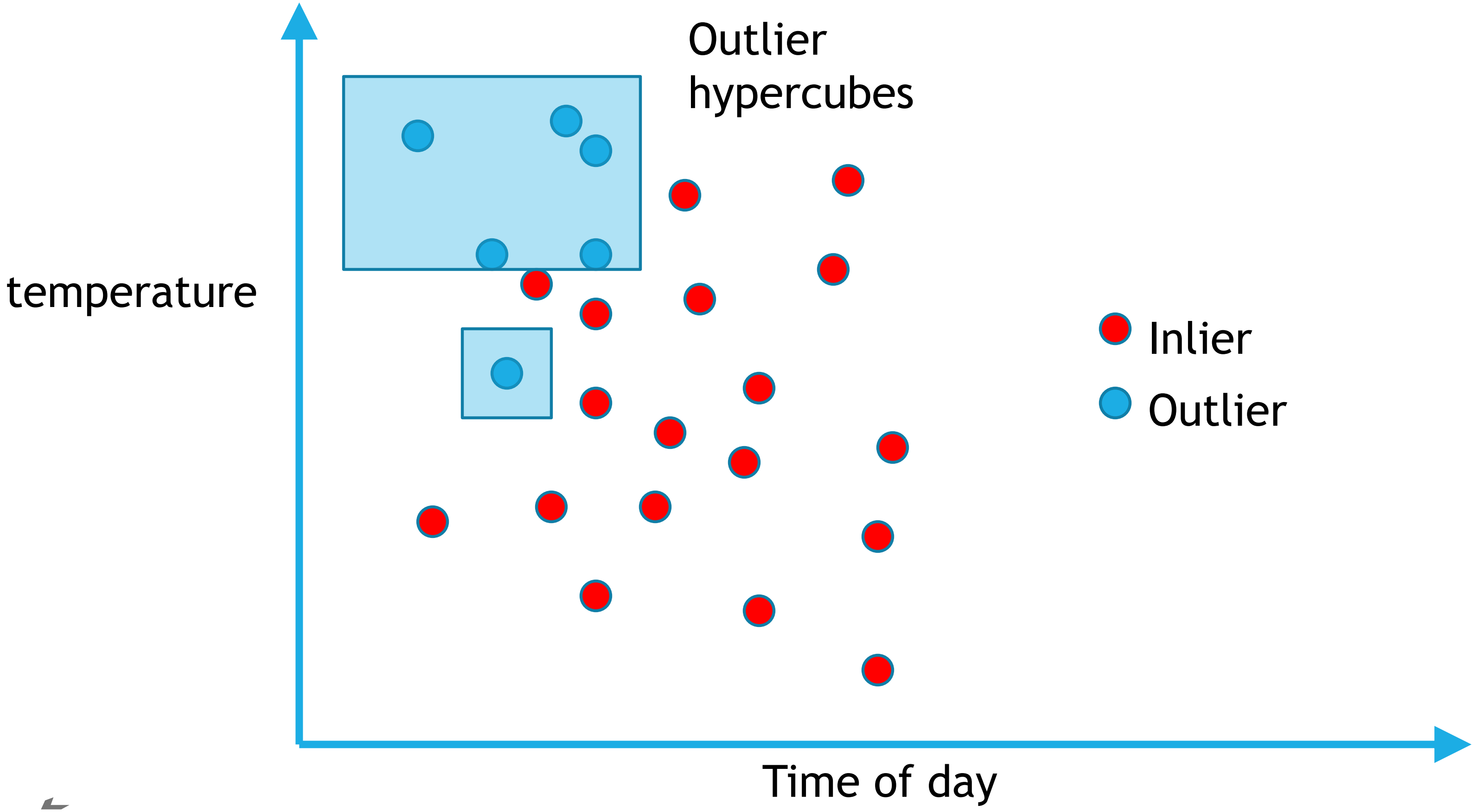
SCORPION EXPLANATION APPROACH

Find hypercubes that contain outliers



VARIABLE SPECIFICITY

Many possible answers depending on tightness of fit



OUTLIER CONCLUSION

Many IoT applications are fundamentally about finding anomalies in a timely fashion

Important to go beyond *finding* outliers to also *explain* them

Explanation is a complex process, as data is often very high dimensional

MODULE SUMMARY

IoT data is quite different than traditional data sources, because it is *noisy* and *approximate*.

Further, IoT systems have special requirements, especially *resource limitations* and a need for *outlier* and *anomaly detection*

THANK YOU!

Samuel Madden

Professor, MIT EECS

Computer Science and Artificial Intelligence Laboratory (CSAIL)
Massachusetts Institute of Technology