

---

# Data Collection and Aggregation

# Challenges: data

---

- Data type: numerical sensor readings.
- **Rich and massive data**, spatially distributed and correlated.
- **Data dynamics**: data streaming and aging.
- **Uncertainty**, noise, erroneous data, outliers.
- **Semantics**. Raw data → knowledge.

# Challenges: query variability

---

- **Data-centric query**: search for “car detection”, instead of sensor node ID.
- **Geographical query**: report values near the lake.
- **Real-time detection & control**: intruder detection.
- **Multi-dimensional query**: spatial, temporal and attribute range.
- **Query interface**: fixed base station or mobile hand held devices.

# Data processing

---

- In-network aggregation
- In-network storage
- Distributed data management
- Statistical modeling
- Intelligent reasoning

# In-network data aggregation

---

- Communication is expensive, bandwidth is precious.
  - “In-network processing”: process raw data before transmit.
- Single sensor reading may not hold much value.
  - Inherently unreliable, outlier readings.
  - Users are often interested in the hidden patterns or the global picture.
- Data compression and knowledge discovery.
  - Save storage; generate semantic report.

# Distributed In-network Storage

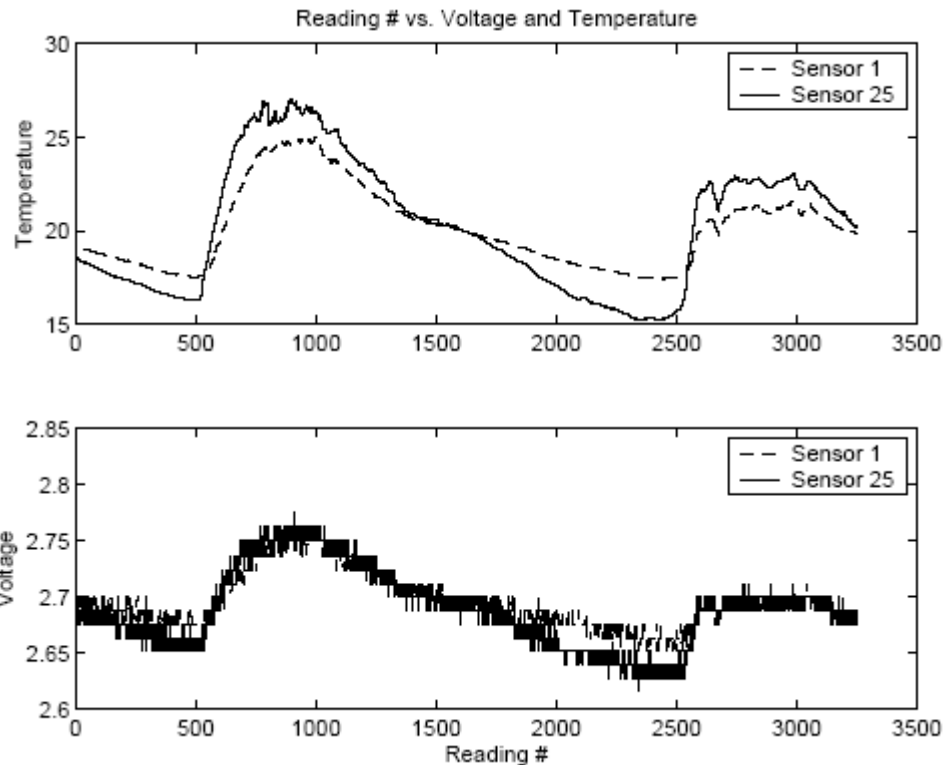
---

- Flash drive, etc. enables distributed in-network storage
- Challenges
  - Distributed indexing for fast query dissemination
  - Explore storage locality to benefit data retrieval.
  - Resilience to node or link failures.
  - Graceful adaptation to data skews.
  - Alleviate the “hot spot” problem created by popular data.

# Sound statistical models

---

- Raw data may misrepresent the physical world.
  - Sensors sample at discrete times. Sensors may be faulty. Packets may be lost.
  - Most sensor data may not improve the answer quality to the query. Data can be compressed.
  - Correlation between nearby sensors or different attributes of the same sensor.



# Model-based query

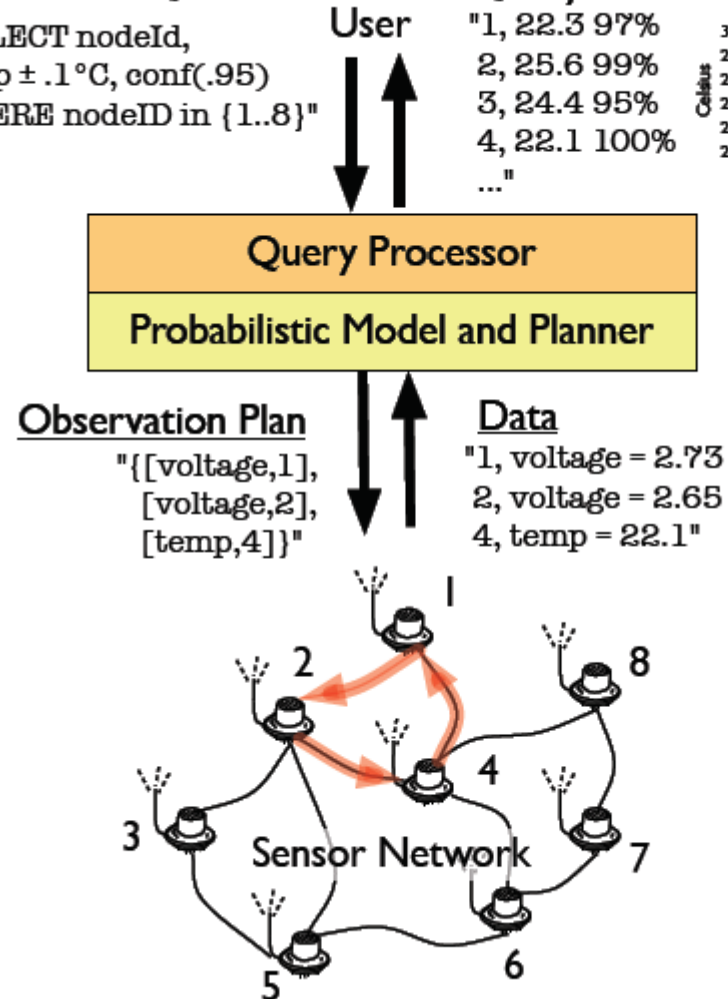
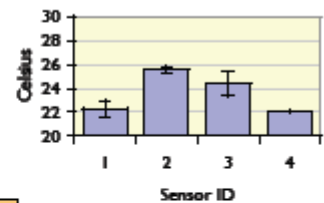
- Build statistical models on the sensor readings.
  - Generates observation plan to improve model accuracy.
  - Answers query results.
- Pros:
  - Improve data robustness.
  - Explore correlation
  - Decrease communication cost.
  - Provide prediction of the future.
  - Easier to extract data abstraction.

## Probabilistic Queries

```
"SELECT nodeId,  
temp ± .1 °C, conf(.95)  
WHERE nodeId in {1..8}"
```

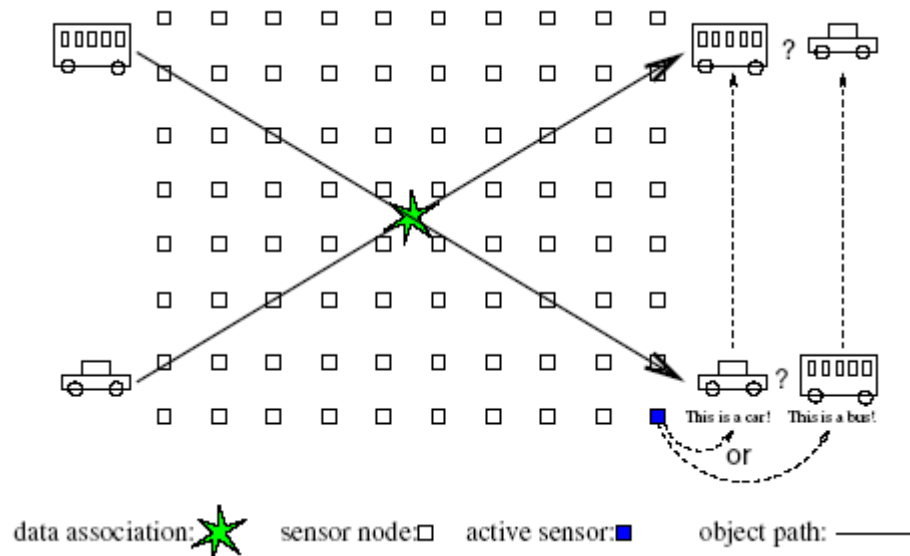
## Query Results

```
"1, 22.3 97%  
2, 25.6 99%  
3, 24.4 95%  
4, 22.1 100%  
..."
```



# Reasoning and control

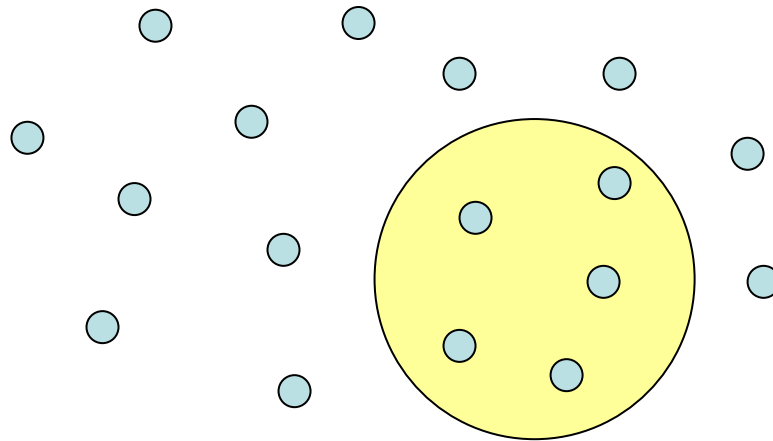
- Reason from raw sensor readings for high-level semantic events.
  - Fire detection.
- Events triggered reaction, sensor tasking and control.
  - Turn on fire alarm. Direct people to closest exits.



# Data privacy, fault tolerance and security

---

- Under what format should data be stored?
- What if a sensor die? Can we recover its data?
- What information is revealed if a sensor is compromised?
- Adversary injects false reports and false alarms.



# Approximation and randomization

---

- Connection to **streaming data** model:
  - No way to store the raw data.
  - Scan the data sequentially.
  - Maintain sketches of massive amount of data.
  - One more challenge in sensor network: the streaming data is spatially distributed and communication is expensive.
- Approximations, sampling, randomization.

# Papers

---

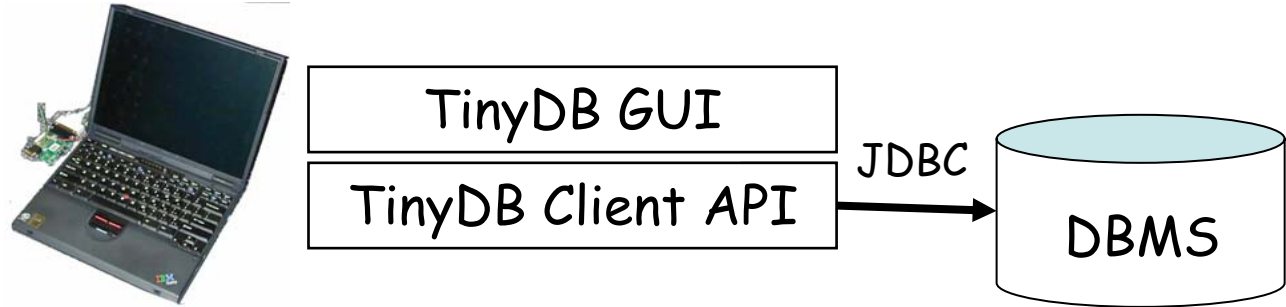
- **[Madden02]** Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. [TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks](#). OSDI, December 2002. *Aggregation with a tree.*
- **[Shrivastava04]** Nisheeth Shrivastava, Chiranjeev Buragohain, Divy Agrawal, Subhash Suri, [Medians and Beyond: New Aggregation Techniques for Sensor Networks](#), ACM SenSys '04, Nov. 3-5, Baltimore, MD. *Approximate answer to medians, reduce storage and message size.*
- **[Nath04]** Suman Nath, Phillip B. Gibbons, Zachary Anderson, and Srinivasan Seshan, [Synopsis Diffusion for Robust Aggregation in Sensor Networks](#)". In proceedings of ACM SenSys'04. *Use multipath routing to improve routing robustness. Order and duplicate insensitive synopsis needs to be used to prevent one data value to be aggregated multiple times.*

# TinyDB

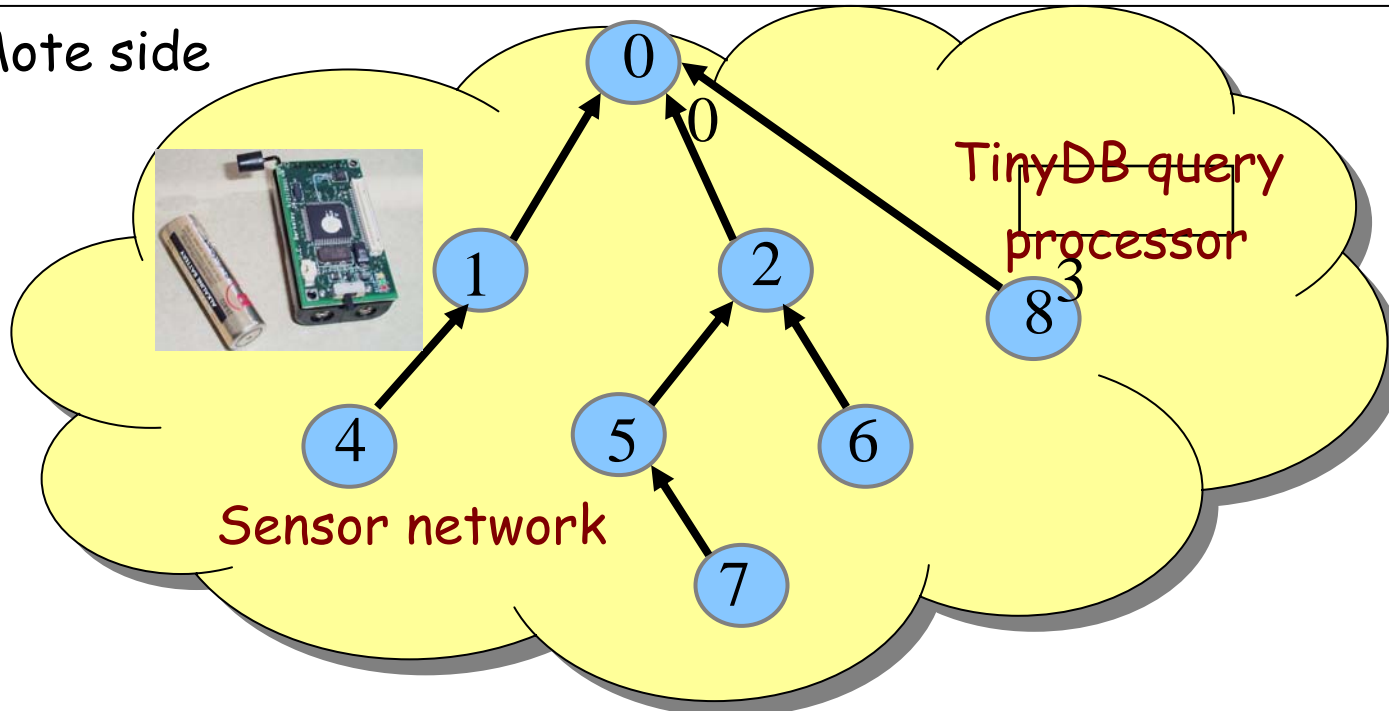
---

- Philosophy:
  - Sensor network = distributed database.
  - Data are stored locally.
  - Networking structure: tree-based routing.
  - Top-down SQL query.
  - Results aggregated back to the query node.
  - Most intelligence outside the network.

# TinyDB Architecture



Mote side



# Query Language (TinySQL)

---

**SELECT** <aggregates>, <attributes>  
[**FROM** {sensors | <buffer>}]  
[**WHERE** <predicates>]  
[**GROUP BY** <exprs>]  
[**SAMPLE PERIOD** <const> | **ONCE**]  
[**INTO** <buffer>]  
[**TRIGGER ACTION** <command>]

# TinySQL Examples

"Find the sensors in bright nests."



1

```
SELECT nodeid, nestNo, light
FROM sensors
WHERE light > 400
EPOCH DURATION 1s
```

Sensors

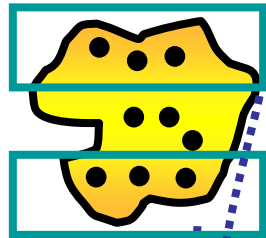
Epoch	Nodeid	nestNo	Light
0	1	17	455
0	2	25	389
1	1	17	422
1	2	25	405

# TinySQL Examples (cont.)

② `SELECT AVG(sound)`  
`FROM sensors`  
`EPOCH DURATION 10s`

"Count the number occupied nests in each loud region of the island."

③ `SELECT region, CNT(occupied)`  
`AVG(sound)`  
`FROM sensors`  
`GROUP BY region`  
`HAVING AVG(sound) > 200`  
`EPOCH DURATION 10s`



Epoch	region	CNT(...)	AVG(...)
0	North	3	360
0	South	3	520
1	North	3	370
1	South	3	520

Regions w/ `AVG(sound) > 200`

# Data Model

---

- Entire sensor network as one single, infinitely-long logical table: *sensors*
- Columns consist of all the *attributes* defined in the network
- Typical attributes:
  - Sensor readings
  - Meta-data: node id, location, etc.
  - Internal states: routing tree parent, timestamp, queue length, etc.
- Nodes return NULL for unknown attributes

# Query over Stored Data

---

- Named buffers in Flash memory
- Store query results in buffers
- Query over named buffers
- Analogous to materialized views
- Example:
  - CREATE BUFFER name SIZE x (field1 type1, field2 type2, ...)
  - SELECT a1, a2 FROM sensors SAMPLE PERIOD d INTO name
  - SELECT field1, field2, ... FROM name SAMPLE PERIOD d

# Event-based Queries

---

- ON event SELECT ...
- Run query only when interesting events happens
- Event examples
  - Button pushed
  - Message arrival
  - Bird enters nest
- Analogous to triggers but events are user-defined

# TAG: Tiny Aggregation

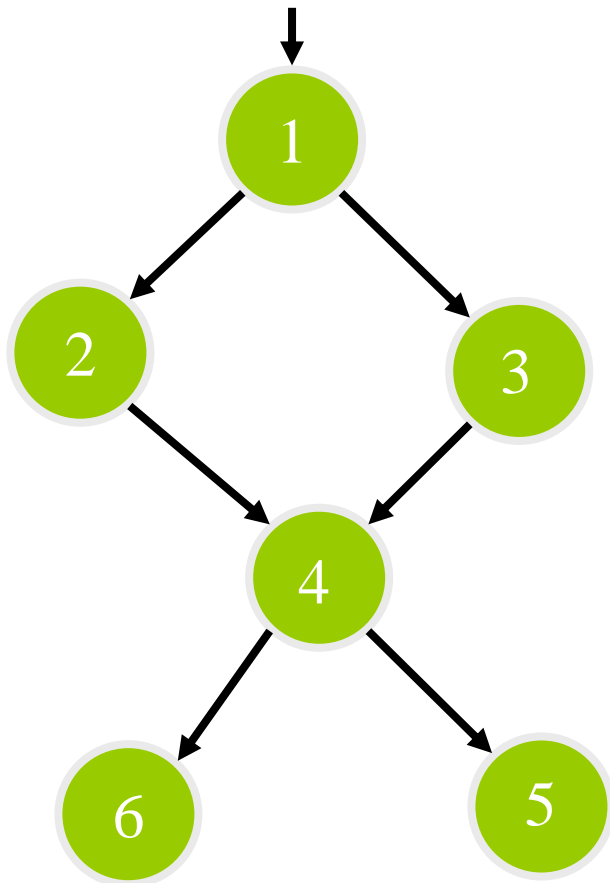
---

- **Query Distribution:** aggregate queries are pushed down the network to construct a spanning tree.
  - Root broadcasts the query, each node hearing the query broadcasts.
  - Each node selects a parent. The routing structure is a spanning tree rooted at the query node.
- **Data Collection:** aggregate values are routed up the tree.
  - Internal node aggregates the partial data received from its subtree.

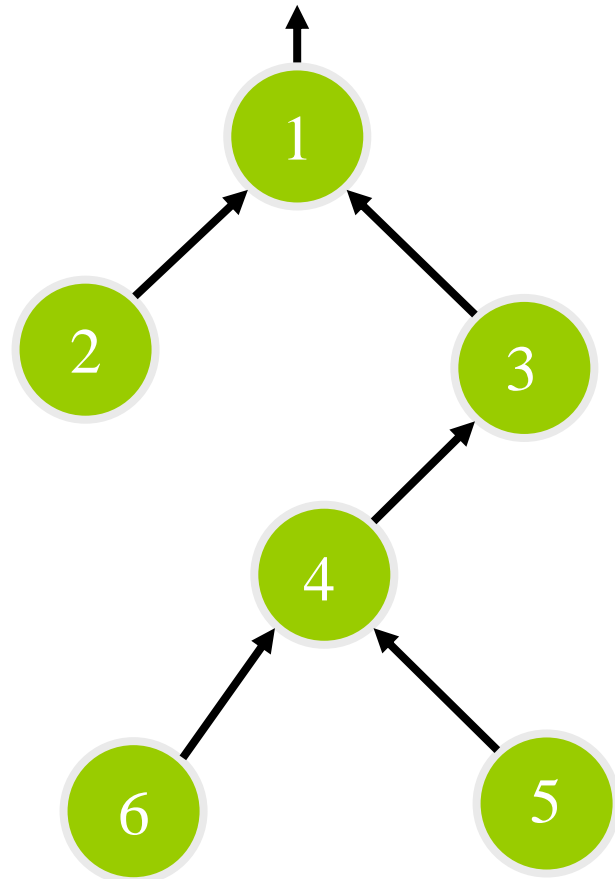
# TAG example

---

Query distribution



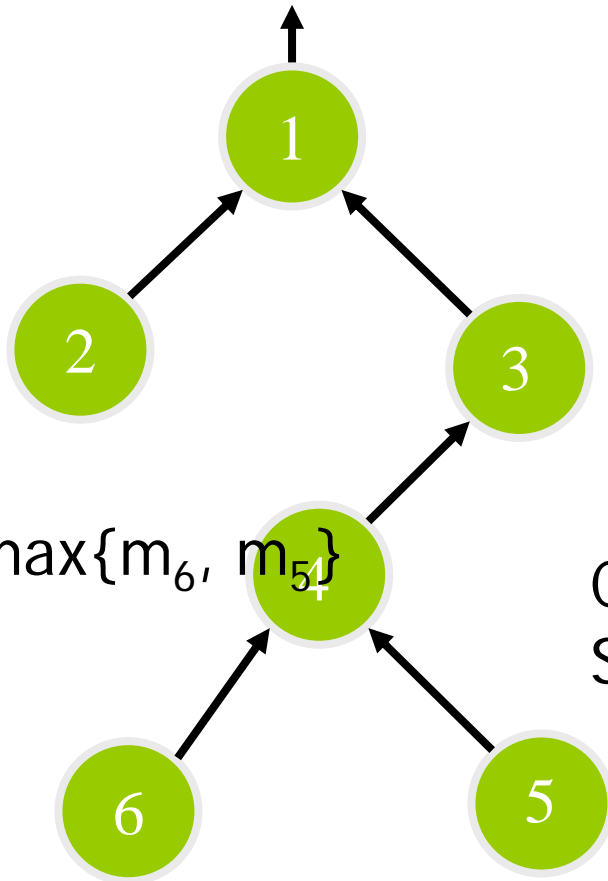
Query collection



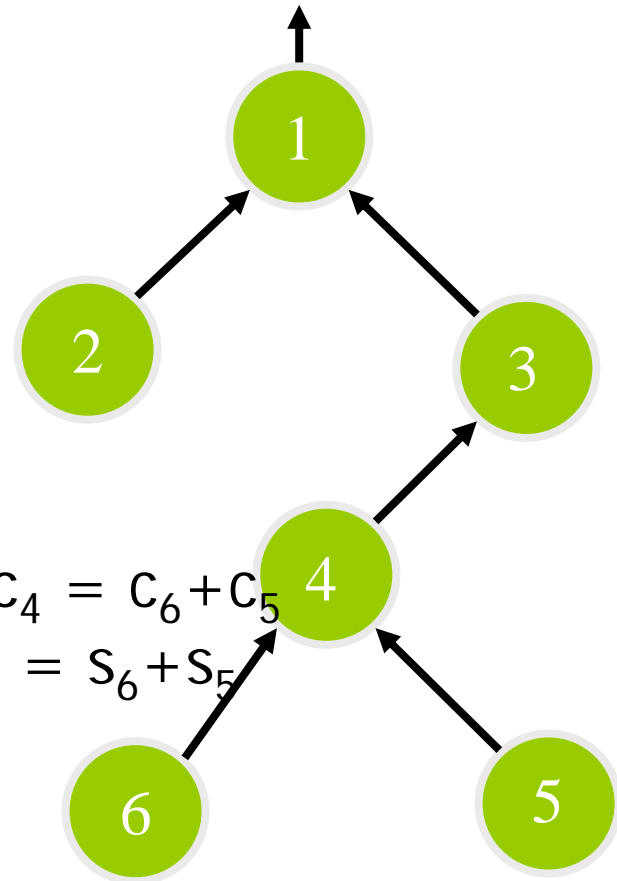
# TAG example

---

MAX



AVERAGE



# Considerations about aggregations

---

- Packet loss?
  - Acknowledgement and re-transmit?
  - Robust routing?
- Packets arriving out of order or in duplicates?
  - Double count?
- Size of the aggregates?
  - Message size growth?

# Classes of aggregations

---

- **Exemplary** aggregates return one or more representative values from the set of all values; **summary** aggregates compute some properties over all values.
  - MAX, MIN: exemplary; SUM, AVERAGE: summary.
  - Exemplary aggregates are prone to packet loss and not amendable to sampling.
  - Summary aggregates of random samples can be treated as a robust estimation.

# Classes of aggregations

---

- **Duplicate insensitive** aggregates are unaffected by duplicate readings.
  - Examples: MAX, MIN.
  - Independent of routing topology.
  - Combine with robust routing (multi-path).

# Classes of aggregations

---

- **Monotonic** aggregates: when two partial records  $s_1$  and  $s_2$  are combined to  $s$ , either  $e(s) \geq \max\{e(s_1), e(s_2)\}$  or  $e(s) \leq \min\{e(s_1), e(s_2)\}$ .
  - Examples: MAX, MIN.
  - Certain predicates (such as HAVING) can be applied early in the network to reduce the communication cost.

# Classes of aggregations

---

- Partial state of the aggregates:

**Good** → 

- **Distributive**: the partial state is simply the aggregate for the partial data. The size is the same with the size of the final aggregate. Example: MAX, MIN, SUM

- **Algebraic**: partial records are of constant size. Example: AVERAGE.

**worst** → 

- **Holistic**: the partial state records are proportional in size to the partial data. Example: MEDIAN.

- **Unique**: partial state is proportional to the number of distinct values. Example: COUNT DISTINCT.

**bad** → 

- **Content-sensitive**: partial state is proportional to some (statistical) properties of the data. Example: fixed-size bucket histogram, wavelet, etc.

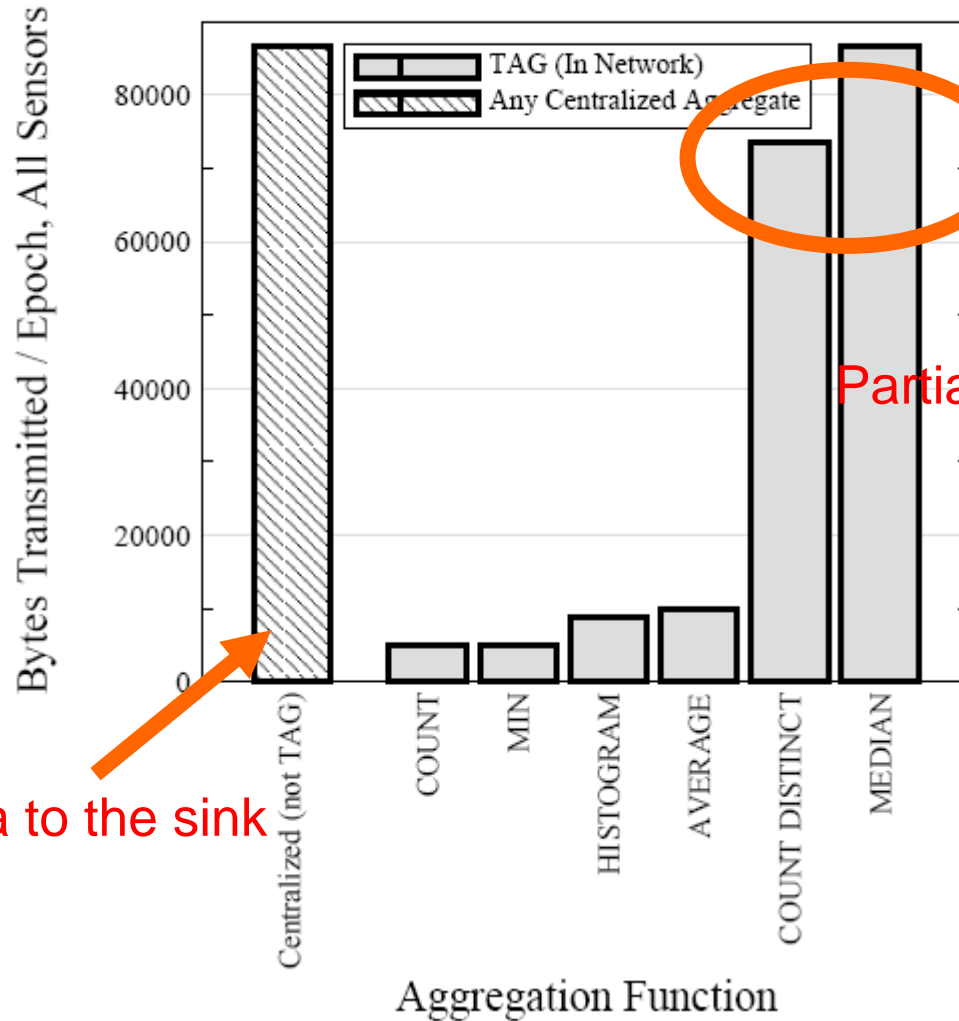
# Classes of aggregates

	Duplicate sensitive	Exemplary, Summary	Monotonic	Partial State
MAX, MIN	No	E	Yes	Distributive
COUNT, SUM	Yes	S	Yes	Distributive
AVERAGE	Yes	S	No	Algebraic
MEDIAN	Yes	E	No	Holistic
COUNT DISTINCT	No	S	Yes	Unique
HISTOGRAM	Yes	S	No	Content-sensitive

# Communication cost

In-network vs. Centralized Aggregation

Network Diameter = 50, No Loss



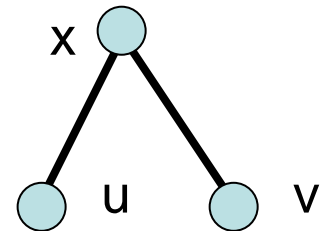
Send all data to the sink

Partial states too large!

# Problem with median

---

- Computing average is simple on an aggregation tree.
  - Each node  $x$  stores the average  $a(x)$  and the number of nodes in its subtree  $n(x)$ .
  - The average of a node  $x$  can be computed from its children  $u, v$ .  $n(x)=n(u)+n(v)$ .  $a(x)=(a(u)n(u)+a(v)n(v))/n(x)$ .
- Computing the median with a fixed amount of message is hard.
  - We do not know the rank of  $u$ 's median in  $v$ 's dataset.
  - We resort to approximations.



# Deal with computing median

---

- Resort to approximation.
  - Random sampling approach.
  - A deterministic approach.

# Approach I: Random sampling

---

- Problem: compute the median  $a$  of  $n$  unsorted elements  $\{a_i\}$ .
- Solution: Take a random sample of  $k$  elements  $K$ . Compute the median  $x$  of  $K$ .
- Claim:  $x$  has rank within  $(\frac{1}{2}+\varepsilon)n$  and  $(\frac{1}{2}-\varepsilon)n$  with probability at least  $1-2/\exp\{2k\varepsilon^2\}$ . (Proof left as an exercise.)
- Choose  $k=\ln(2/\delta)/(2\varepsilon^2)$ , then  $x$  is an approximate median with probability  $1-\delta$ .

# Approach II: Quantile digest (q-digest)

---

- A data structure that answers
  - Approximate quantile query: median, the  $k$ th largest reading.
  - Range queries: the  $k$ th to  $l$ th largest readings.
  - Most frequent items.
  - Histograms.
- Properties:
  - Deterministic algorithm.
  - Error-memory trade-off.
  - Confidence factor.
  - Support multiple queries.

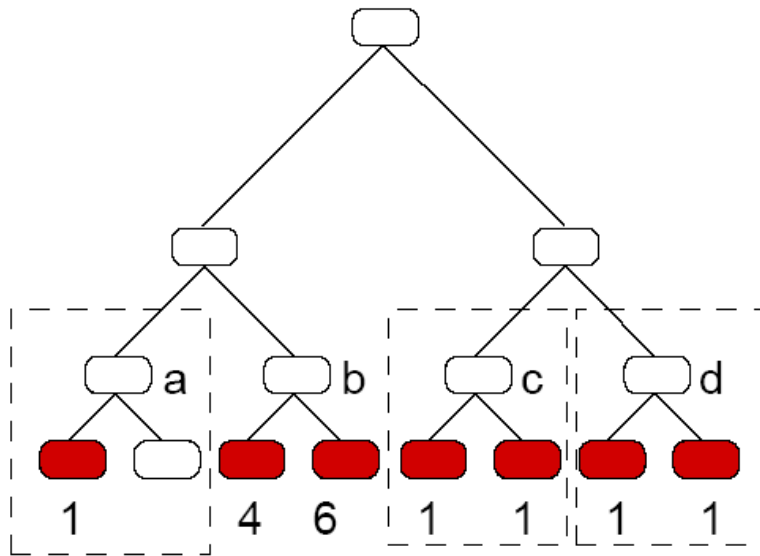
# Q-digest

---

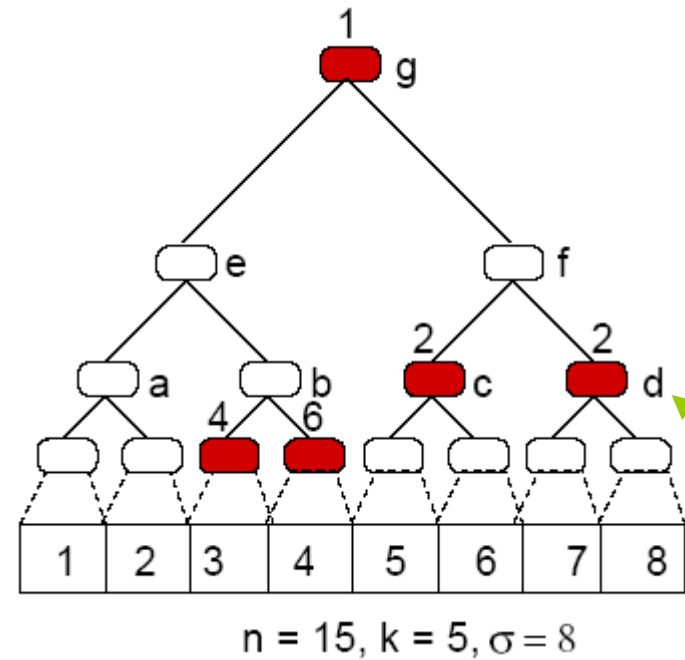
- Input data: frequency of data value  $\{f_1, f_2, \dots, f_\sigma\}$ .
- Compress the data:
  - detailed information concerning frequent data are preserved;
  - less frequently occurring values are lumped into larger buckets resulting in information loss.
- Buckets: the nodes in a **binary partition** of the range  $[1, \sigma]$ . Each bucket  $v$  has range  $[v.min, v.max]$ .
- Only store non-zero buckets.

# Example

Input data bucketed



Q-digest




Information loss


# Q-digest properties

---

- Store values in buckets.
  1.  $\text{Count}(v) \leq n/k$ . (except leaf)
    - Control information loss.
  2.  $\text{Count}(v) + \text{Count}(p) + \text{Count}(s) > n/k$ .  
(except root)



parent

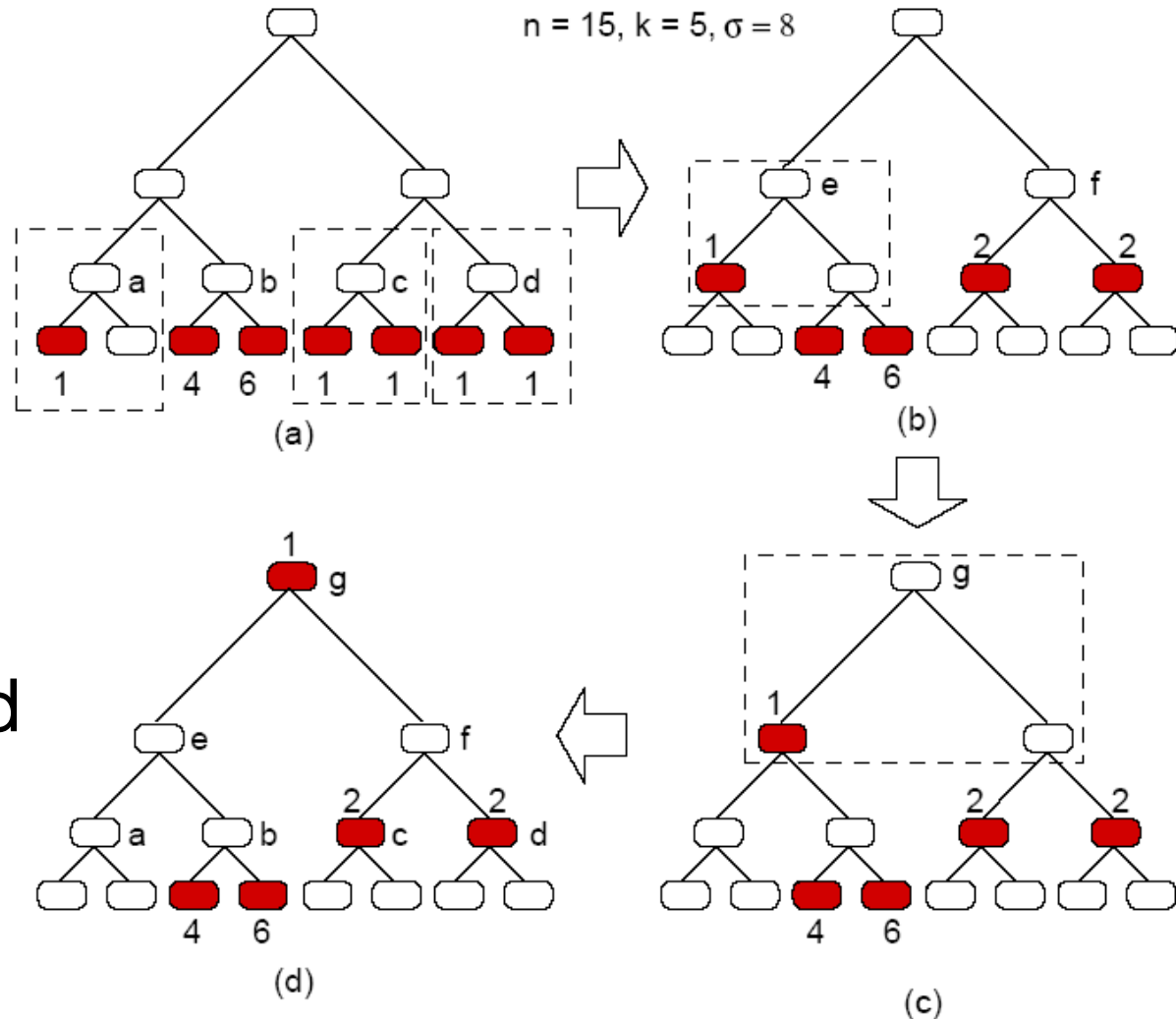


sibling

    - Ensure sufficient compression.
    - $K$ : compression parameter.

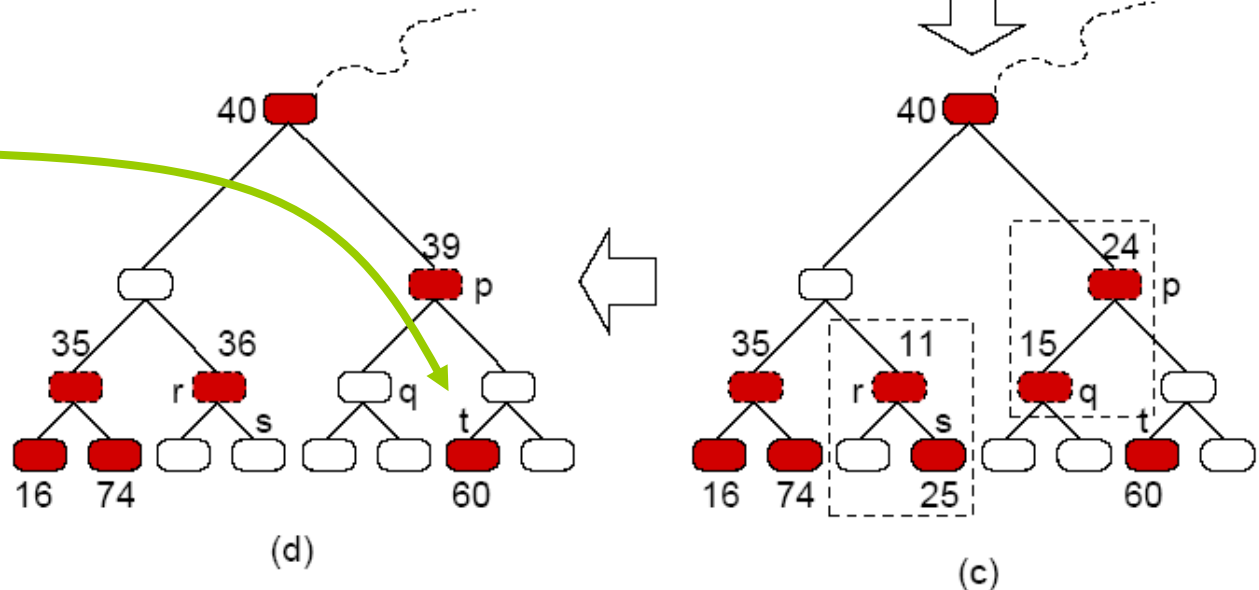
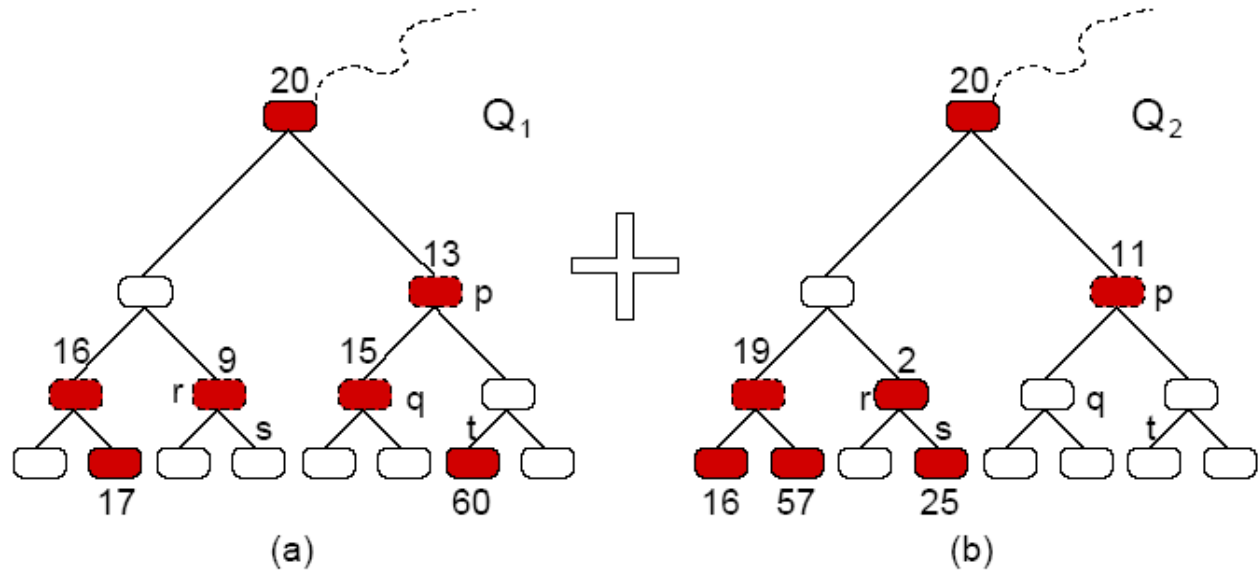
# Construct a q-digest

- Each sensor constructs a q-digest based on its value.
- Check the digest property bottom up: two “small” children’s count are added up and moved to the parent.



# Merging two q-digests

- Merge q-digests from two children
- Add up the values in buckets
- Re-evaluate the digest property bottom up.



Information loss: t undercounts since some of its value appears on ancestors.

# Space complexity

---

Claim: A q-digest with compression parameter  $k$  has at most  $3k$  buckets.

- By property 2, for all buckets  $v$  in  $Q$ ,
  - $\sum_{v \in Q} [\text{Count}(v) + \text{Count}(p) + \text{Count}(s)] > |Q| n/k.$
  - $\sum_{v \in Q} [\text{Count}(v) + \text{Count}(p) + \text{Count}(s)] \leq 3$   
 $[\sum_{v \in Q} \text{Count}(v)] = 3n.$
  - $|Q| < 3k.$

# Paper presentation on 3/24

---

- **[Vieira08]** Luiz F. M. Vieira, Uichin Lee, Mario Gerla, "**Phero-Trail: a Bio-inspired Location Service for Mobile Underwater Sensors**"  
*WUWNet'08, Location service for mobile underwater 3D networks.*
- **[Przydatek03]** Bartosz Przydatek, Dawn Song, Adrian Perrig, "**SIA: Secure Information Aggregation in Sensor Networks**", Sensys'03.  
*Secure computation of median and average.*