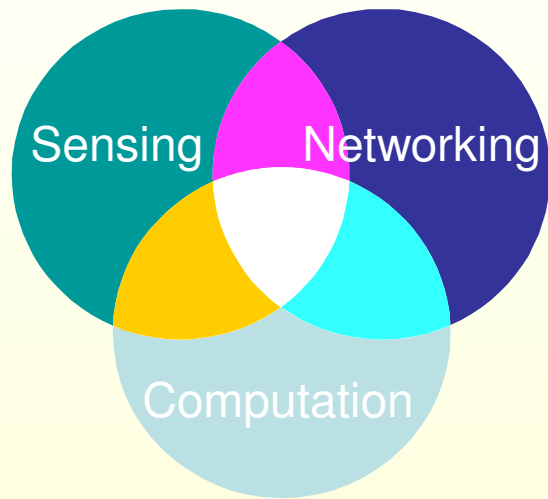
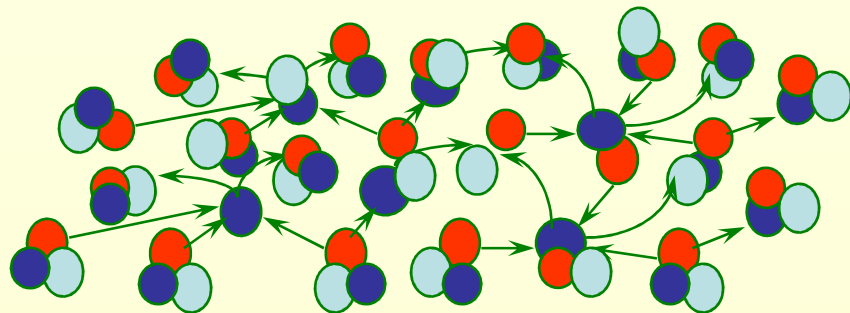


CS321: Information Aggregation, Sensor Data Bases; TinyDB



Leonidas Guibas
Computer Science Dept.
Stanford University

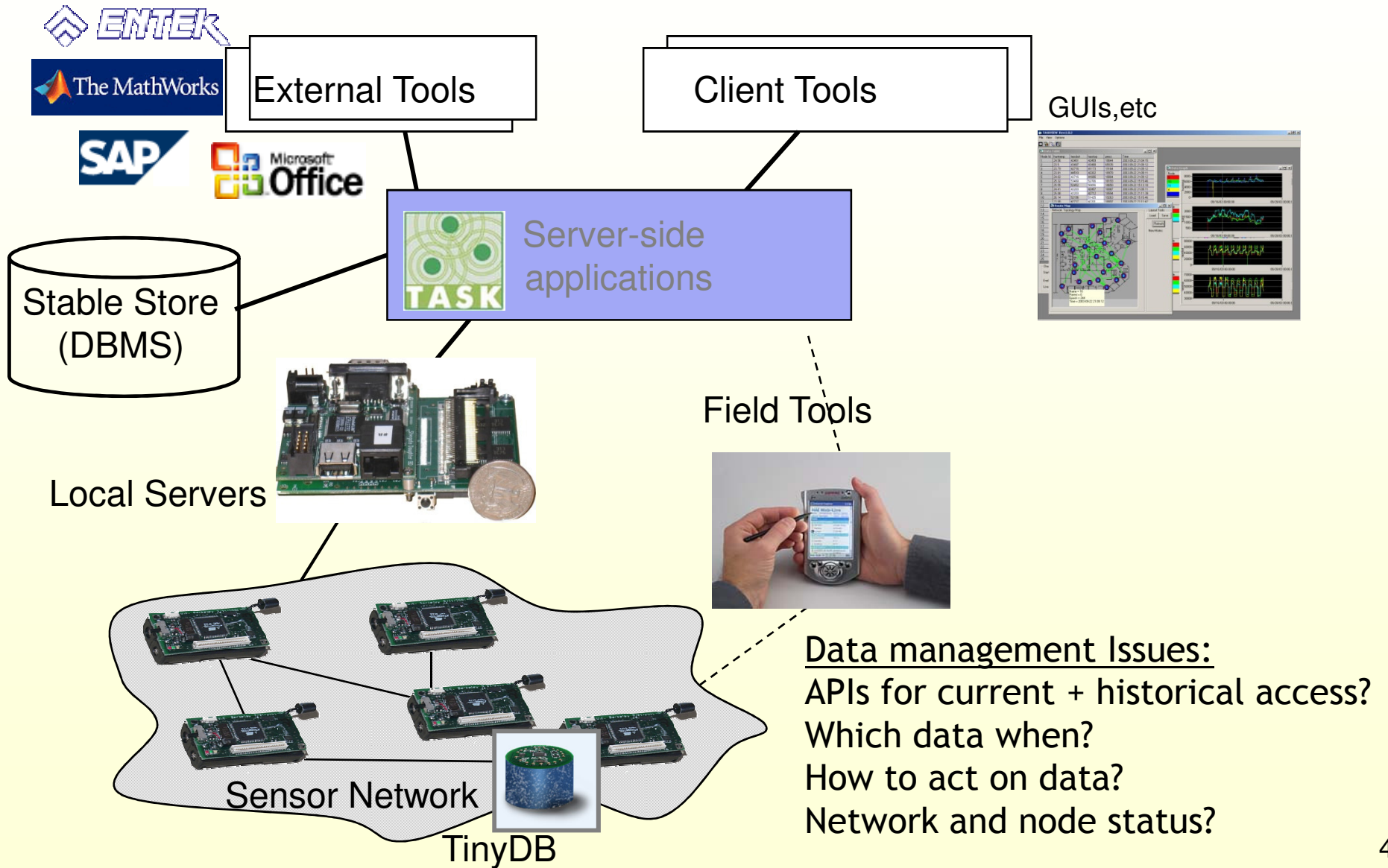


Sensor Systems as Storage Systems and as DBs

Sensor Systems as Storage Systems and as DBs

- Sensor networks:
 - collect measurements from the physical world
 - organize and store these measurements over time
 - serve continuous or single shot queries about current or past events
- So sensor networks can be thought of as **distributed storage systems** over these physical measurements

Data Management Landscape



Logical vs. Physical Data Access

- A sensor net DB organization builds an **abstract reference mechanism** for the sensor net data
- It allows queries to be expressed at a level close to the **application semantics** – just like in a traditional DB
- This allows the system to hide physical layer details, like where the data is stored, replication for robustness, network volatility, and so on.
- Of course, this increased convenience comes at a loss of efficiency.

High Level (Query-Based) Interfaces are Good

- Programming apps is hard
 - Limited power budget
 - Lossy, low bandwidth communication
 - Require long-lived, zero admin deployments
 - Distributed algorithms
 - Limited tools, debugging interfaces
- Queries abstract away much of the complexity
 - Burden on the database developers
 - Users get:
 - Safe, optimizable programs
 - Freedom to think about apps instead of details

TinyDB

- Programming sensor nets is hard
- Declarative queries are easy
 - **TinyDB**: In-network processing via declarative queries
- Example:
 - Vehicle tracking application
 - Custom code
 - 1-2 weeks to develop
 - Hundreds of lines of C
 - TinyDB query (on right):
 - 2 minutes to develop
 - Comparable functionality



```
SELECT nodeid
FROM sensors
WHERE mag > thresh
EPOCH DURATION 64ms
```

[Madden *et. al.*, '03]

A SensorNet Dilemma

- SensorNets still packaged like HeathKits
 - Not easy to cope with out of the box
- Bare metal encourages one-off applications
 - Inhibits reuse
- Deployment not intuitive
 - No configuration/monitoring tools
- SensorNet PhD Factor
 - Today ~2.5 PhDs needed to deploy a SensorNet
 - Needs to be Zero



Sensor Network DataBases

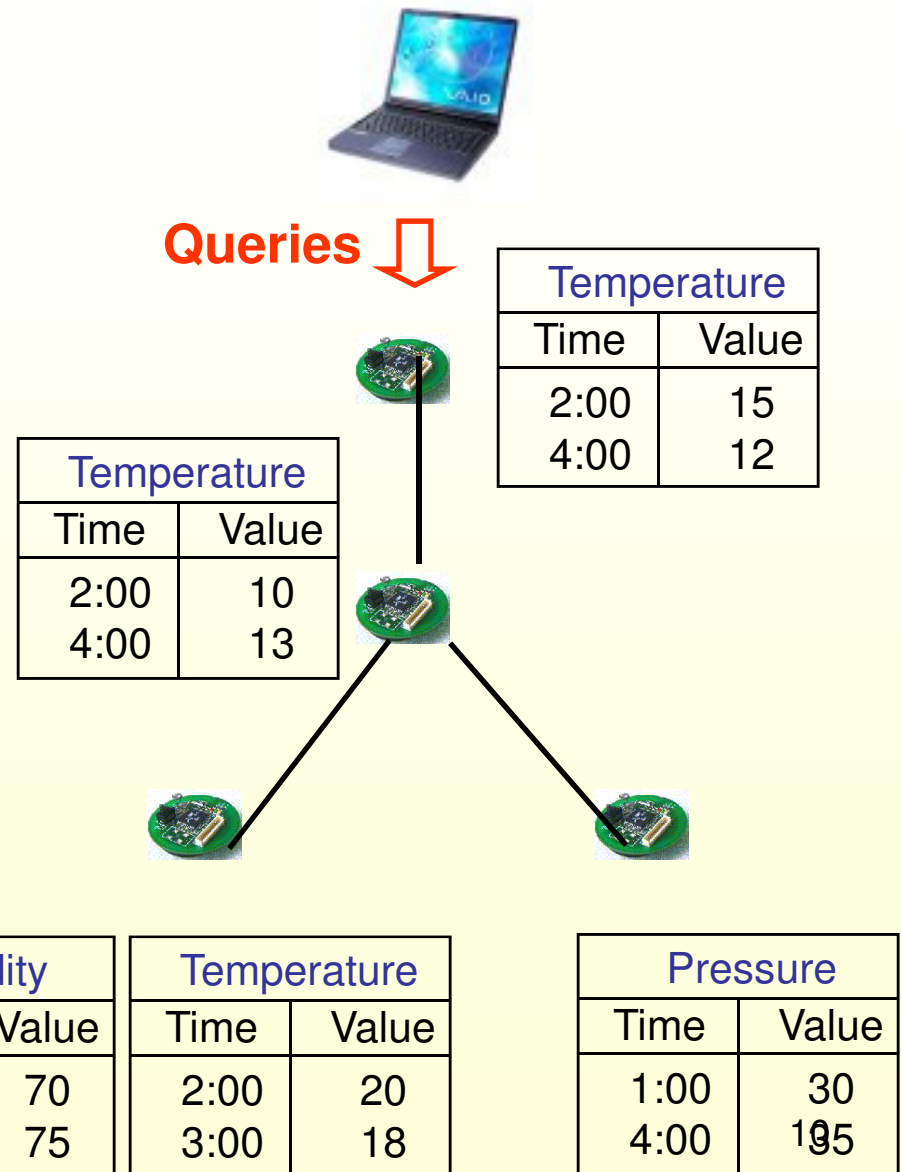
The DB View of Sensor Networks

Traditional SN Programs

Procedural addressing of individual sensor nodes; user specifies how task is executed; data may be processed centrally.

DB Approach

Declarative querying; user isolated from “how the network works”; exploit in-network distributed processing.



Sensor Net Data Types: Observations/Events/Queries

● Observations

- Low-level output(s) from sensors
- E.g., temperature, pressure, acoustic signal

● Events

- Constellations of low-level observations, interpreted as higher-level events or activities
- E.g. fire, intruder

● Clients use **Queries** to elicit event information from the sensor network

- E.g.: Locations of fires in the network
- E.g.: Images of intruders detected

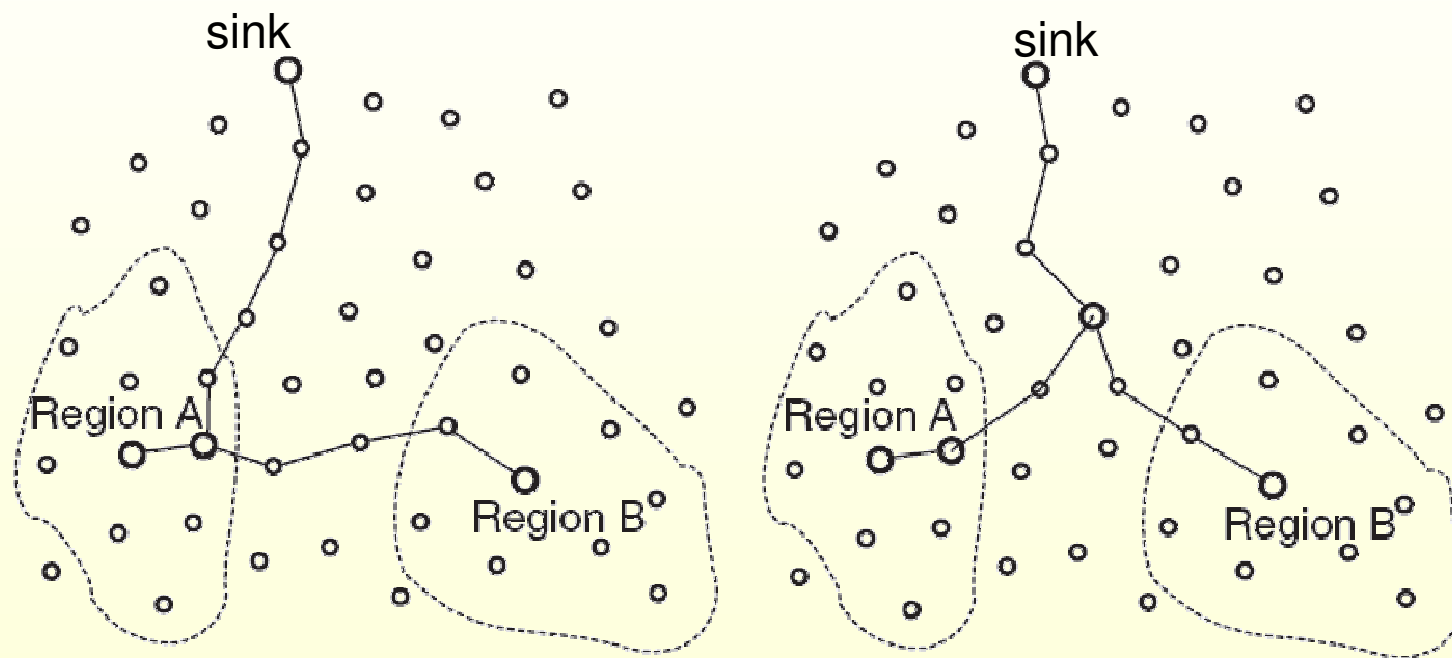
Sensor Network DB Challenges

- These days disks used in DB systems are essentially free; sensor nodes instead have to deal with small node memories – **so data summarization, aggregation, and aging is essential** (but **flash memories** will change that)
- In a sensor network links (and nodes) come and go – **the stored information and access to it must be protected from this physical volatility**
- **Data is not always pre-acquired** – potentially present data must be materialized before being queried

Challenges, Cont'd

- Continuous, rather than single shot queries, will be the norm – thus **query optimization is important for saving energy**
- Latencies in access to data can be highly variable; thus **query execution plans must continuously adapt to the network state**
- Query executions can interact and cause conflicts and **resource contention in sensor tasking**

An Example of Query Adaptation



Many A detections,
little correlation with B

Roughly the same A and B
detections, highly correlated

The Data is Different, Too

- Sensor net data inherently contains errors – so exact data comparisons are of little value
- A distinction has to be made between data that could potentially be acquired, and data that actually has been acquired – resource contention and other issues could prevent the capture of potential data
- The relational view of large tables whose entries can be modified is not realistic; may need to work with append-only relations, or small windows over large tables

What Should Queries be Like?

SQL Query Language

- SQL = Structured Query Language
- Many Variants ...
- Operates on Relations (Relational DBs)

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

- Many web tutorials:
<http://dev.mysql.com/doc/refman/5.0/en/tutorial.html>
- Certain additions are useful for sensor networks

SQL-Like Query Examples

Snapshot (single shot) queries:

- How many empty bird nests are in the northeastern quadrant of the forest?

```
SELECT          SUM(s)
FROM            SensorData s
WHERE           s.nest = empty and s.loc in (0,50,0,50)
```

Long-running (continuous) queries:

- Notify me over the next hour whenever the number of empty nests in an area exceeds a threshold.

```
SELECT          s.area, SUM(s)
FROM            SensorData s
WHERE           s.nest = empty
GROUP BY       s.area
HAVING         SUM(s) > T
DURATION       (now, now+60)
EVERY          5
```



What is New in SN Queries?

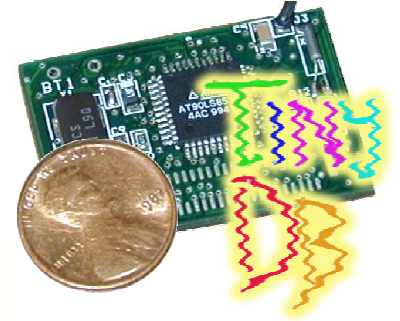
- **Duration** for continuous queries
- **Sampling rates**
- **New data types**, to account for data uncertainty
 - ranges
 - parametric distributions (e.g., Gaussians)
 - operations for computing probabilities, equality likelihood, ...

Using Distributions Instead of Values

- Properly reflects the uncertainty in all sensor measurements
- Answers computed by the sensor net can be given a confidence
- But even simple arithmetic operations can become very costly

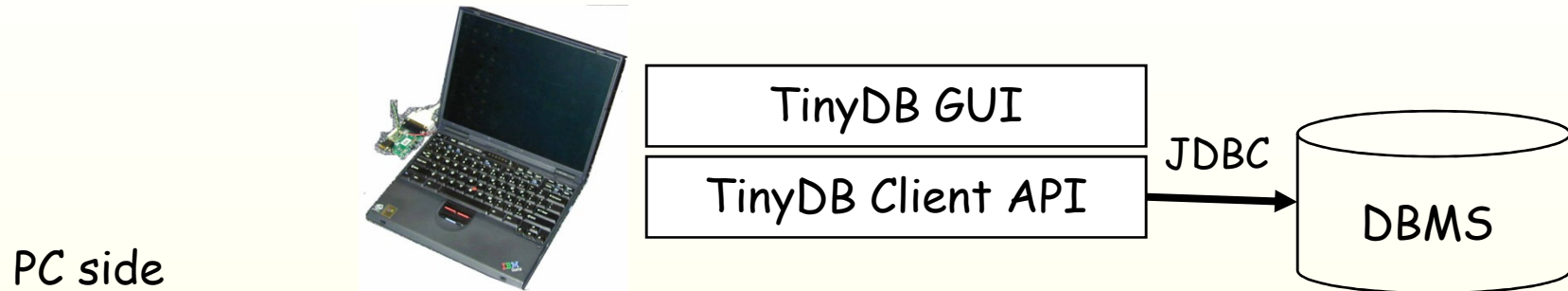
A Sensor Network
Database Example:
TinyDB from UC Berkeley
[Madden, Franklin, Hellerstein, Hong, '03]

Using Declarative Queries

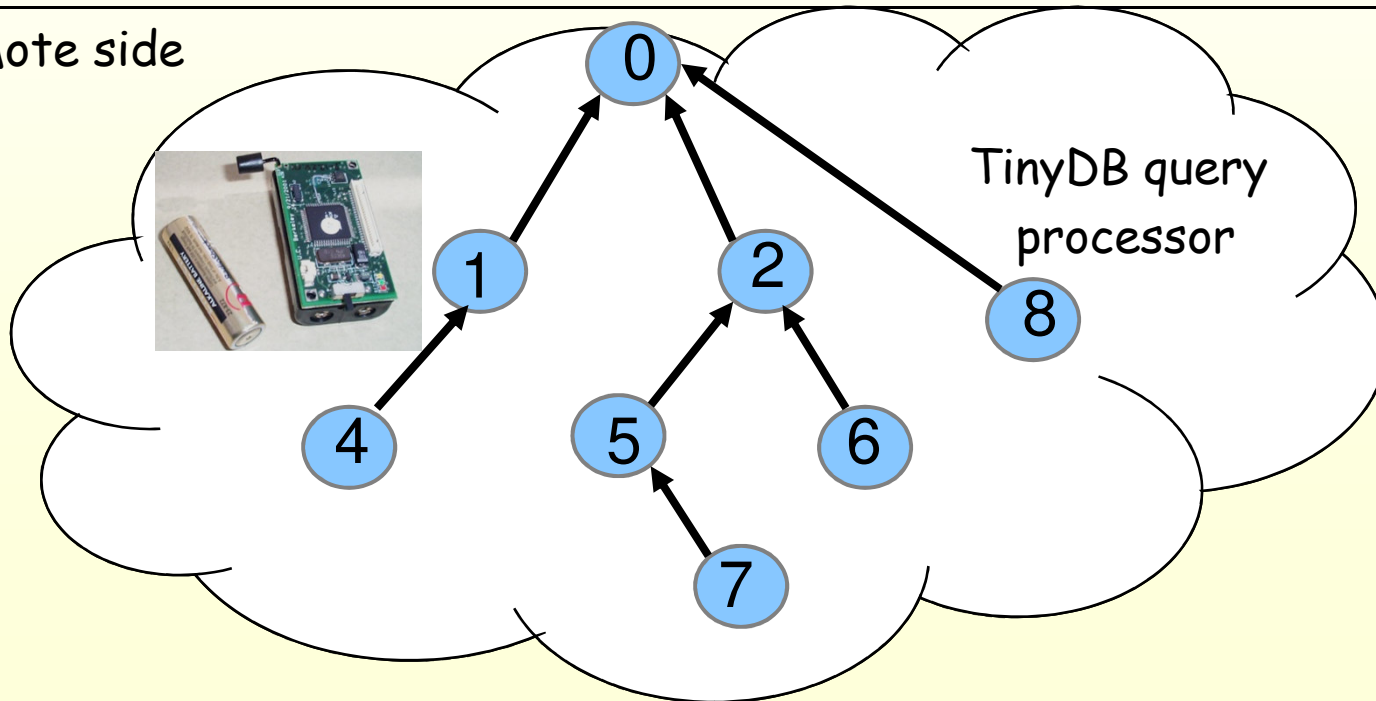


- Users specify the data they want
 - Simple, intuitive, SQL-like queries
 - Using user predicates, not specific node addresses
- Challenge is to provide:
 - Expressive and easy-to-use DB interface
 - High-level operators
 - With well-defined interactions
 - With **transparent optimizations** that many programmers would miss
 - Sensor-net specific techniques
 - Power efficient execution framework

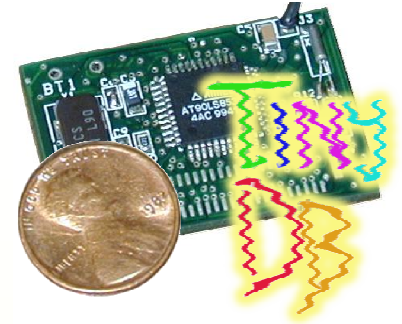
Architecture



Mote side



TinyDB Architecture



```
SELECT  
AVG(temp)  
WHERE  
light > 400
```



Queries



Results

```
T:1, AVG: 225  
T:2, AVG: 250
```



Schema:

• "Catalog" of commands & attributes

~10,000 Lines Embedded C Code
~5,000 Lines (PC-Side) Java
~3200 Bytes RAM (w/ 768 byte heap)
~58 kB compiled code
(3x larger than 2nd largest TinyOS Program)

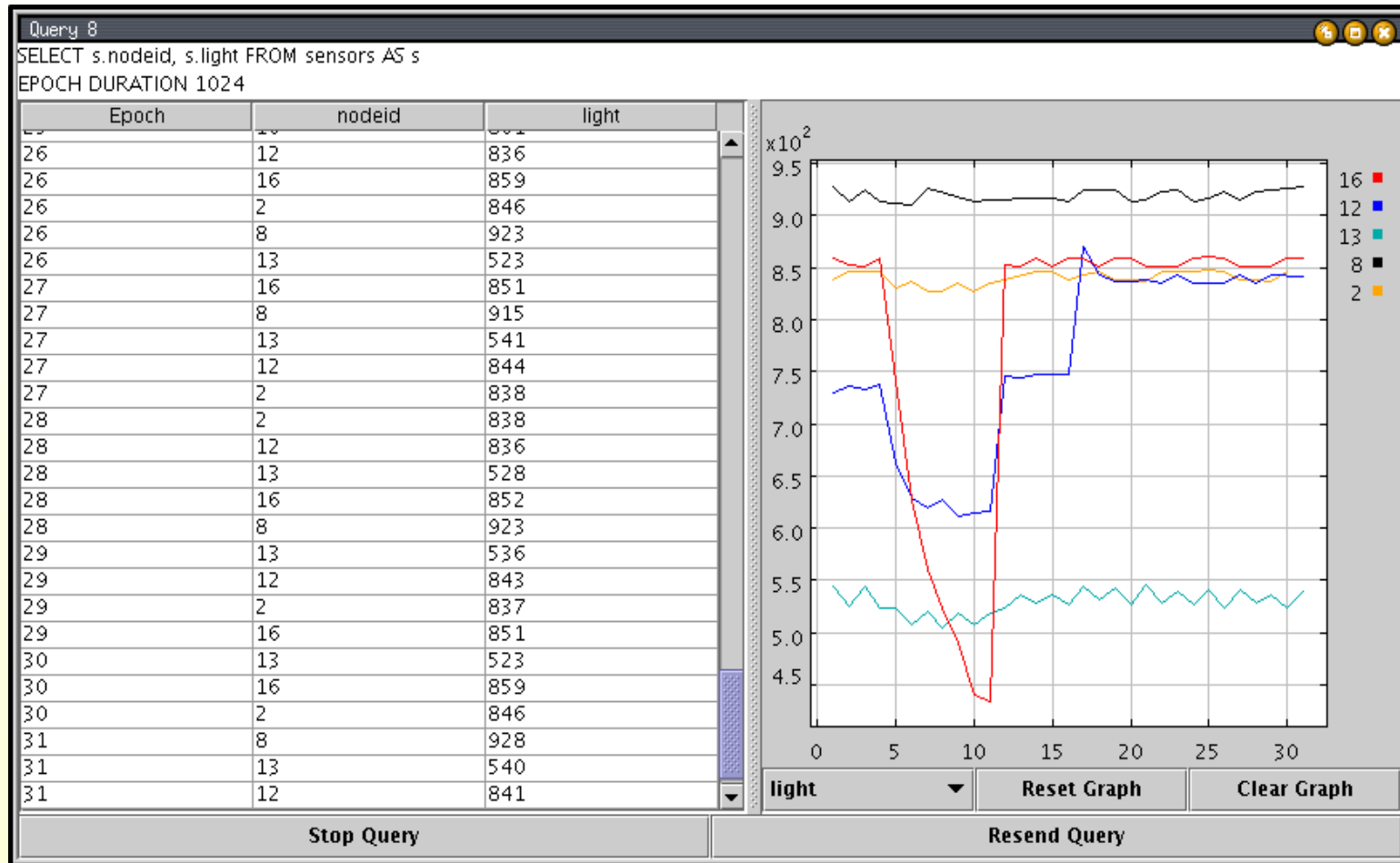


() ...

Data Model

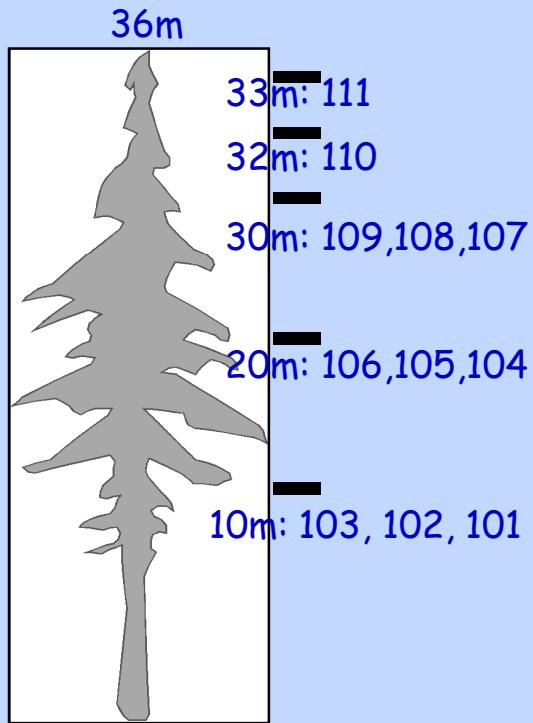
- Entire sensor network as one single, logical table: *sensors*
- Columns consist of all the *attributes* defined in the network
- Typical attributes:
 - Sensor readings
 - Meta-data: node id, location, etc.
 - Internal states: routing tree parent, timestamp, queue length, etc.
- Nodes return NULL for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

TinyDB Interface

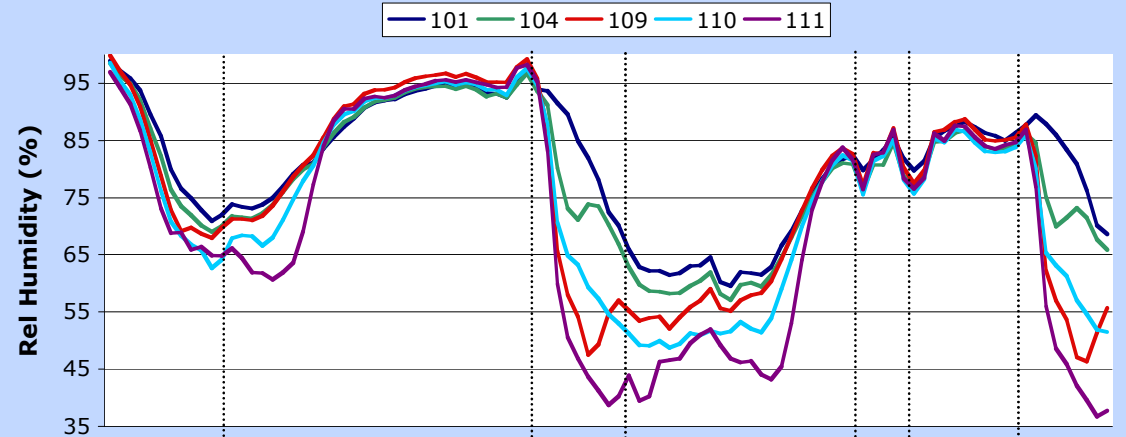


Tables, Graphs

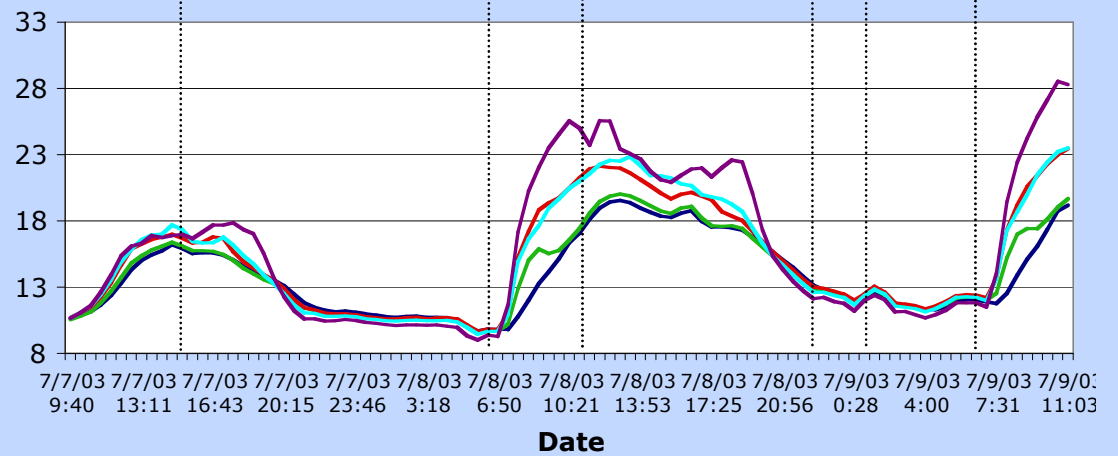
Data from Redwoods



Humidity vs. Time



Temperature vs. Time



Overview

- TinyDB: Queries for Sensor Nets
- Processing Aggregate Queries (TAG)
- Taxonomy and Experiments
- Acquisitional Query Processing (ACQP)

Declarative Queries for Sensor Networks

"Find the sensors in bright nests."



1 Examples:

```
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

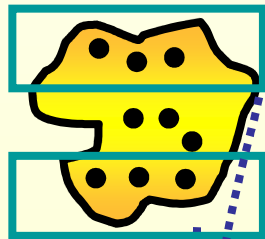


Aggregation Queries

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

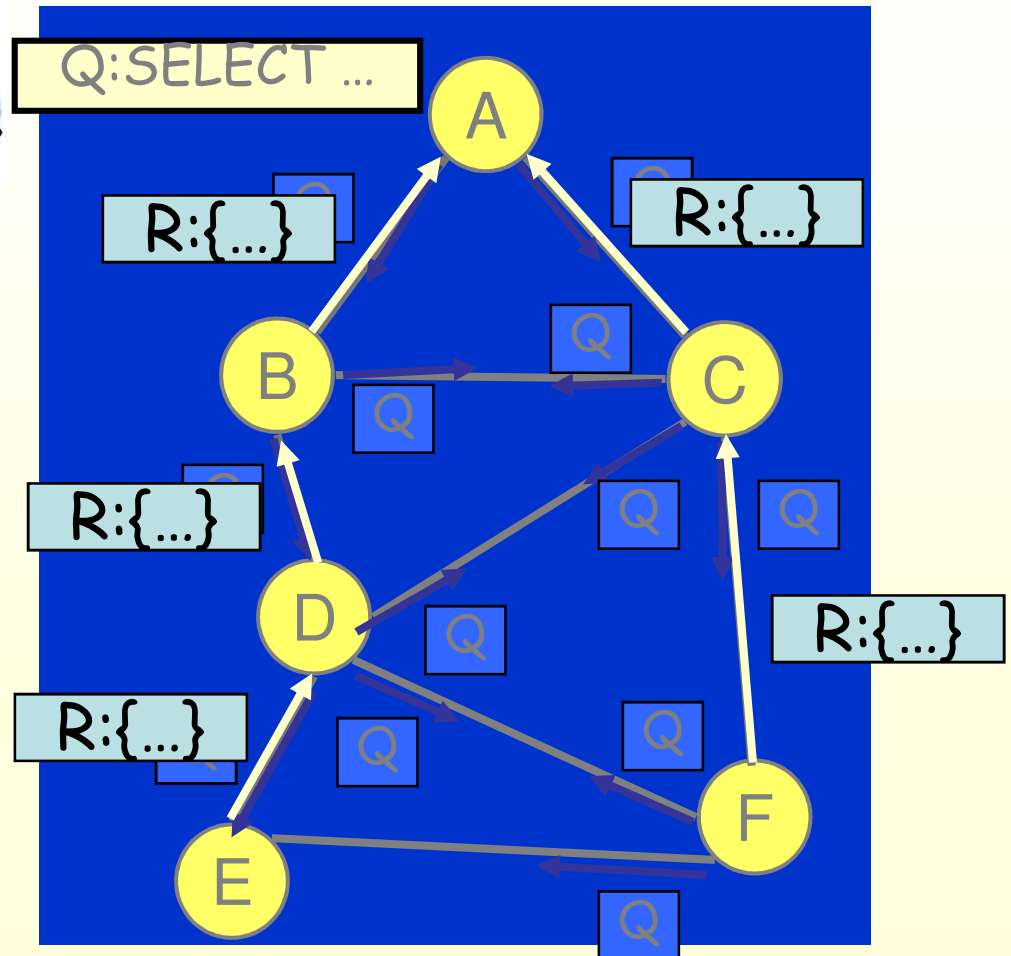
TinyDB Internals

Tiny Aggregation (TAG)

- **In-network processing** of aggregates
 - Common data analysis operation
 - Aka *gather* operation or *reduction* in parallel programming
 - Communication reduction
 - Operator dependent benefit
 - Across nodes during same epoch
- Exploit query semantics to improve efficiency!

Query Propagation Via Tree-Based Routing

- Tree-based routing
 - Used in:
 - Query delivery
 - Data collection
 - Topology selection is important;
 - Continuous process
 - Mitigates failures



Basic Aggregation

- In each epoch (= system sampling period):
 - Each node samples local sensors once
 - Generates **partial state record (PSR)**
 - own local readings
 - readings from children
 - Outputs PSR during assigned **communication interval**
- At end of epoch, PSR for whole network output at root
- New result at each successive epoch
- Extras:
 - Predicate-based partitioning via GROUP BY

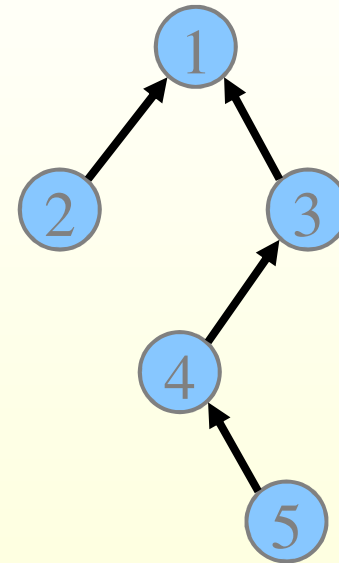


Illustration: Aggregation

```
SELECT COUNT(*)  
FROM sensors
```

Sensor #

	1	2	3	4	5
4					1
3					
2					
1					
4					

Interval #

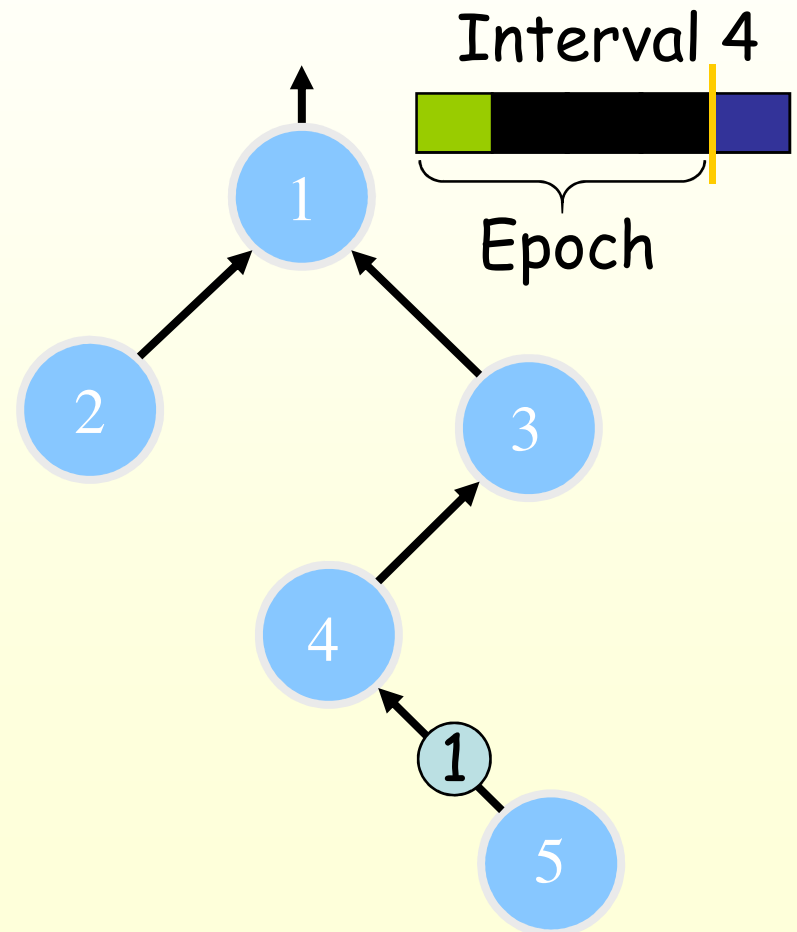


Illustration: Aggregation

```
SELECT COUNT(*)  
FROM sensors
```

Sensor #

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

Interval #

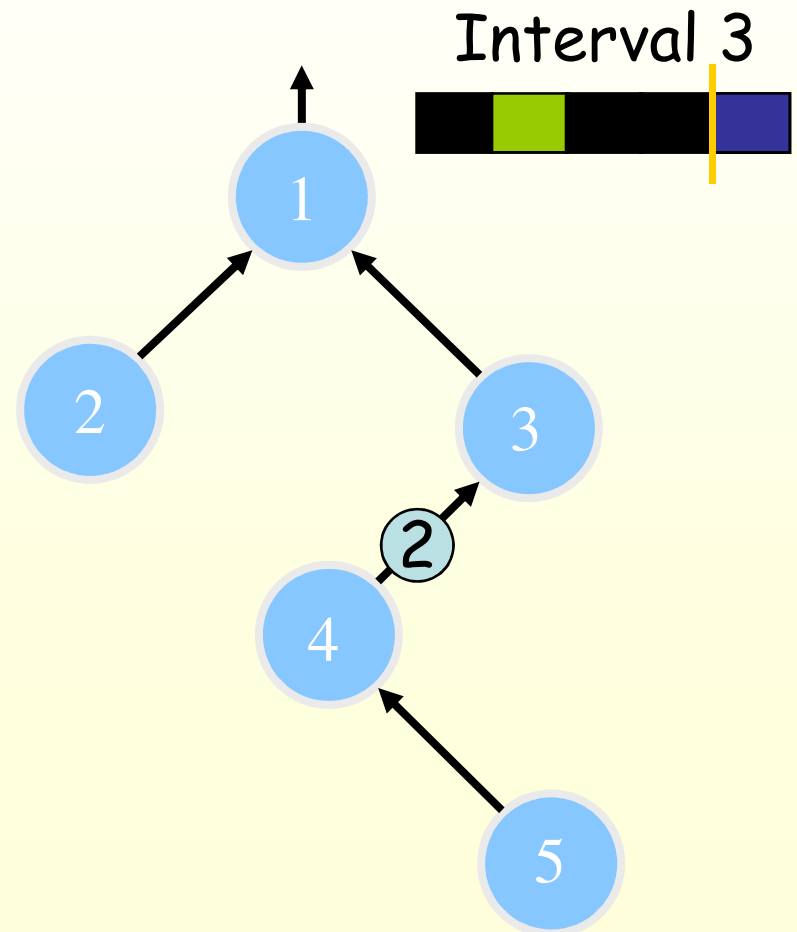


Illustration: Aggregation

```
SELECT COUNT(*)  
FROM sensors
```

Sensor #

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

Interval #

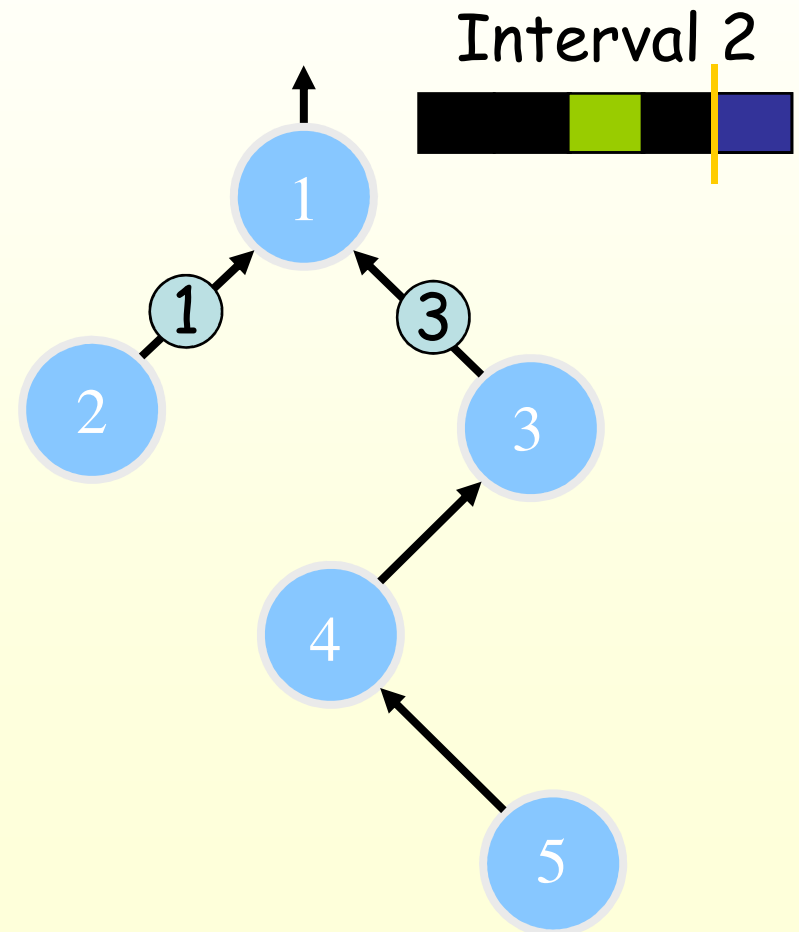


Illustration: Aggregation

```
SELECT COUNT(*)  
FROM sensors
```

Sensor #

Interval #

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

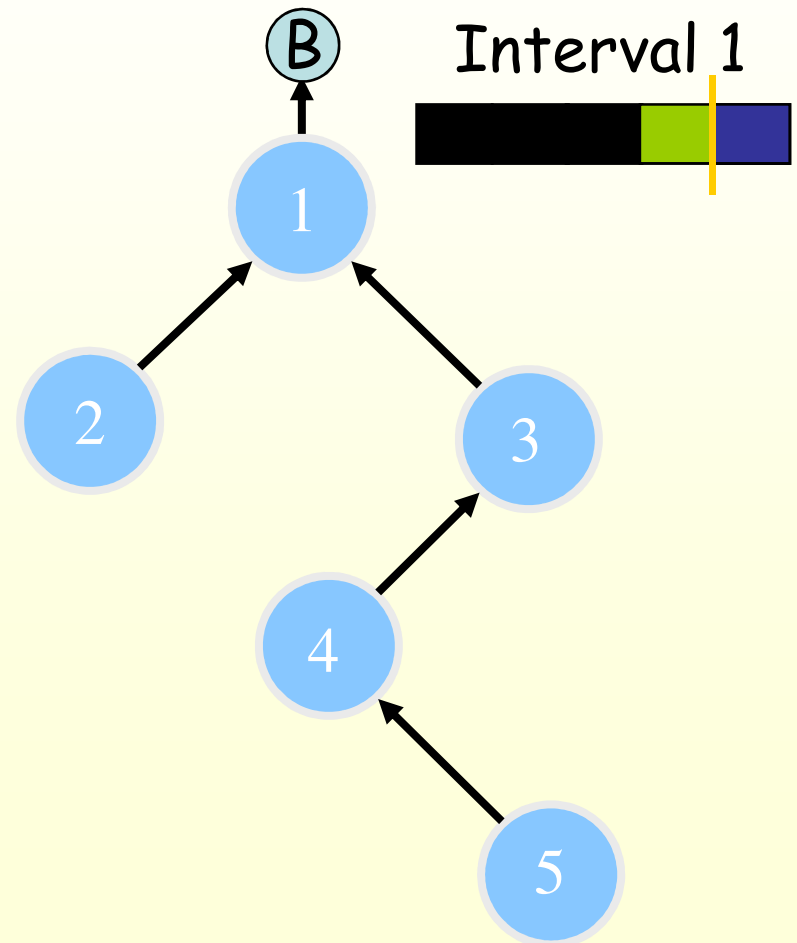


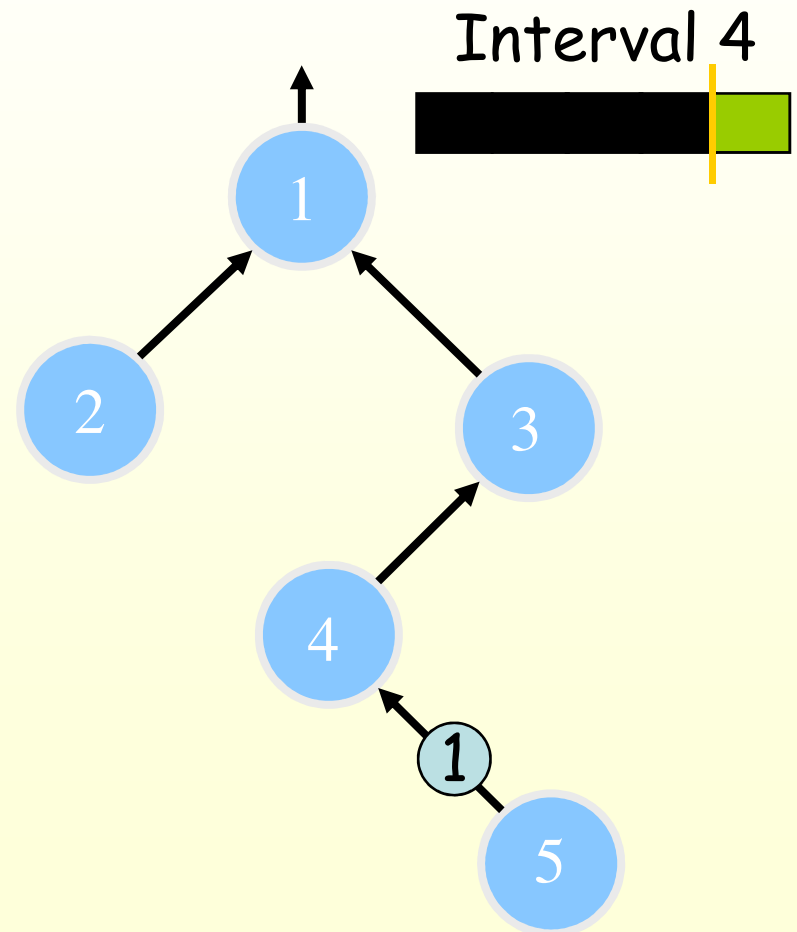
Illustration: Aggregation

```
SELECT COUNT(*)  
FROM sensors
```

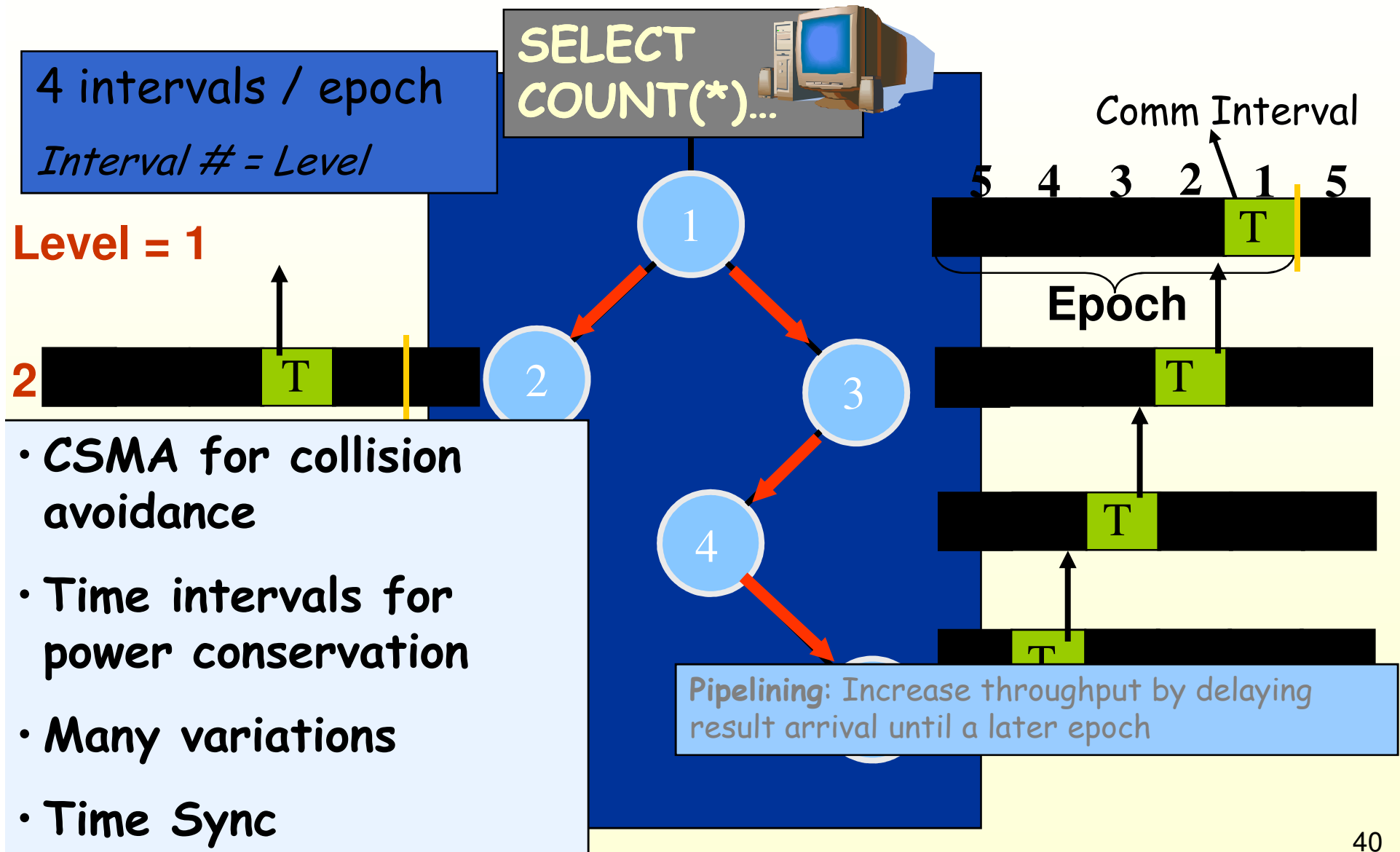
Sensor #

Interval #

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



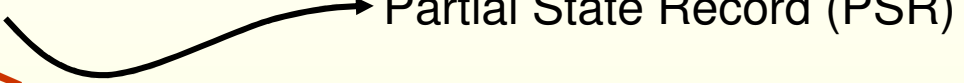
Interval Assignment: An Approach



Aggregation Framework

- As in extensible databases, TinyDB supports any aggregation function conforming to:

Agg_n = {f_{init}, f_{merge}, f_{evaluate}}

F_{init} {a₀} → **<a₀>**  Partial State Record (PSR)

F_{merge} {<a₁>, <a₂>} → **<a₁₂>**

F_{evaluate} {<a₁>} → **aggregate value**

Example: Average

AVG_{init} {v} → **<v, 1>**

AVG_{merge} {<S₁, C₁>, <S₂, C₂>} → **<S₁ + S₂, C₁ + C₂>**

AVG_{evaluate} {<S, C>} → **S/C**

Restriction: Merge must be associative, commutative

Types of Aggregates

- SQL supports MIN, MAX, SUM, COUNT, AVERAGE
- Any function over a set *can* be computed via TAG
- In-network has benefits for many operations
 - E.g. standard deviation, top/bottom n , spatial union/intersection, histograms, etc.
 - Compactness of PSR

Partial State

- Growth of PSR vs. number of aggregated values (n)
 - Algebraic: $|PSR| = 1$ (e.g. MIN)
 - Distributive: $|PSR| = c$ (e.g. AVG)
 - Holistic: $|PSR| = n$ (e.g. MEDIAN)
 - Unique: $|PSR| = d$ (e.g. COUNT DISTINCT)
 - $d = \#$ of distinct values
 - Content Sensitive: $|PSR| < n$ (e.g. HISTOGRAM)

<u>Property</u>	<u>Examples</u>	<u>Affects</u>
Partial State	MEDIAN : unbounded, MAX : 1 record	Effectiveness of TAG

Simulation Environment

- Evaluated TAG via simulation
- Coarse grained event based simulator
 - Sensors arranged on a grid
 - Two communication models
 - Lossless: All neighbors hear all messages
 - Lossy: Messages lost with probability that increases with distance
- Communication (message counts) as performance metric

Benefit of In-Network Processing

Simulation Results

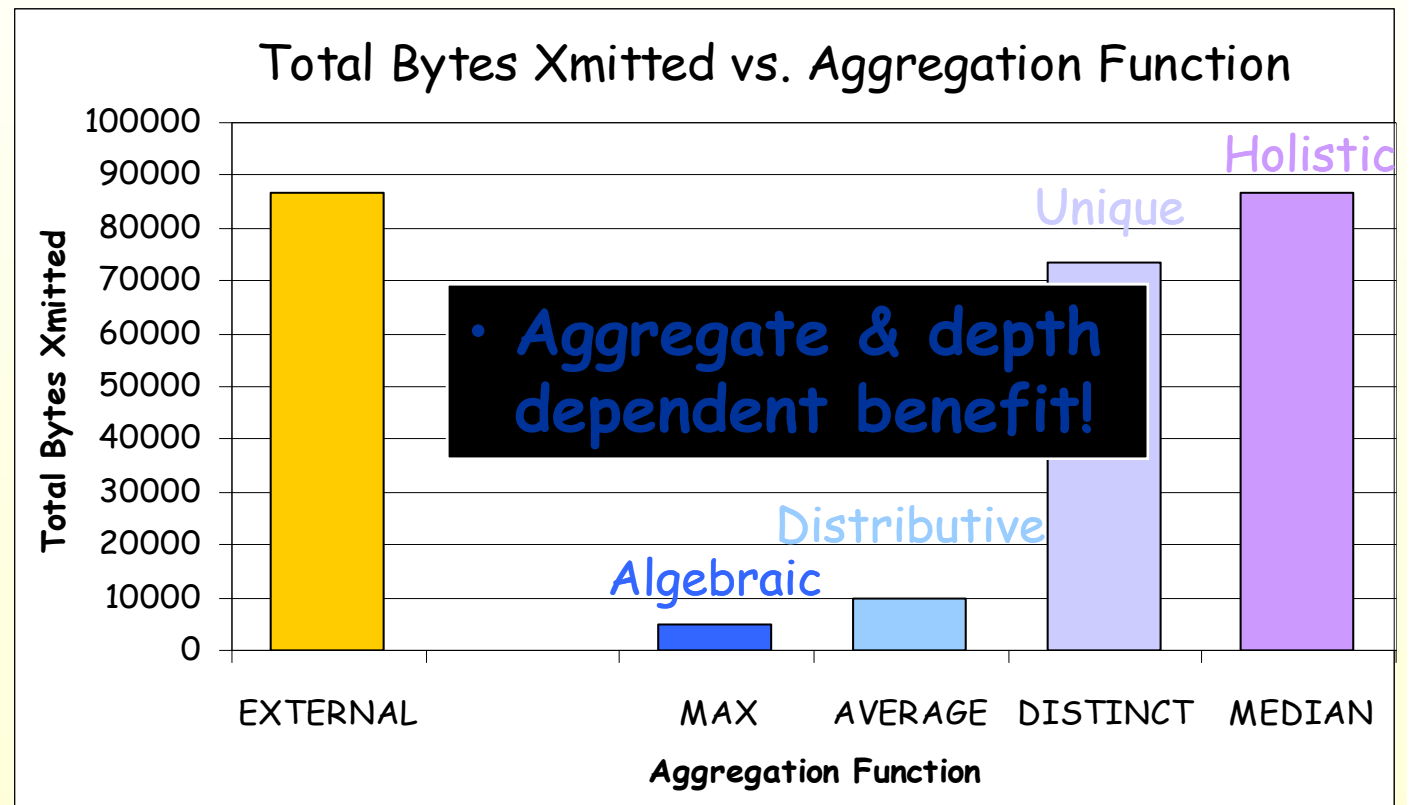
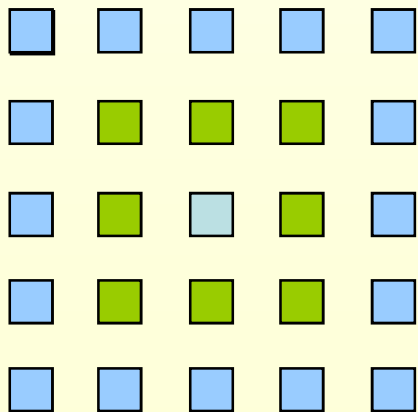
2500 Nodes

50x50 Grid

Depth = ~10

Neighbors = ~20

Uniform Dist.



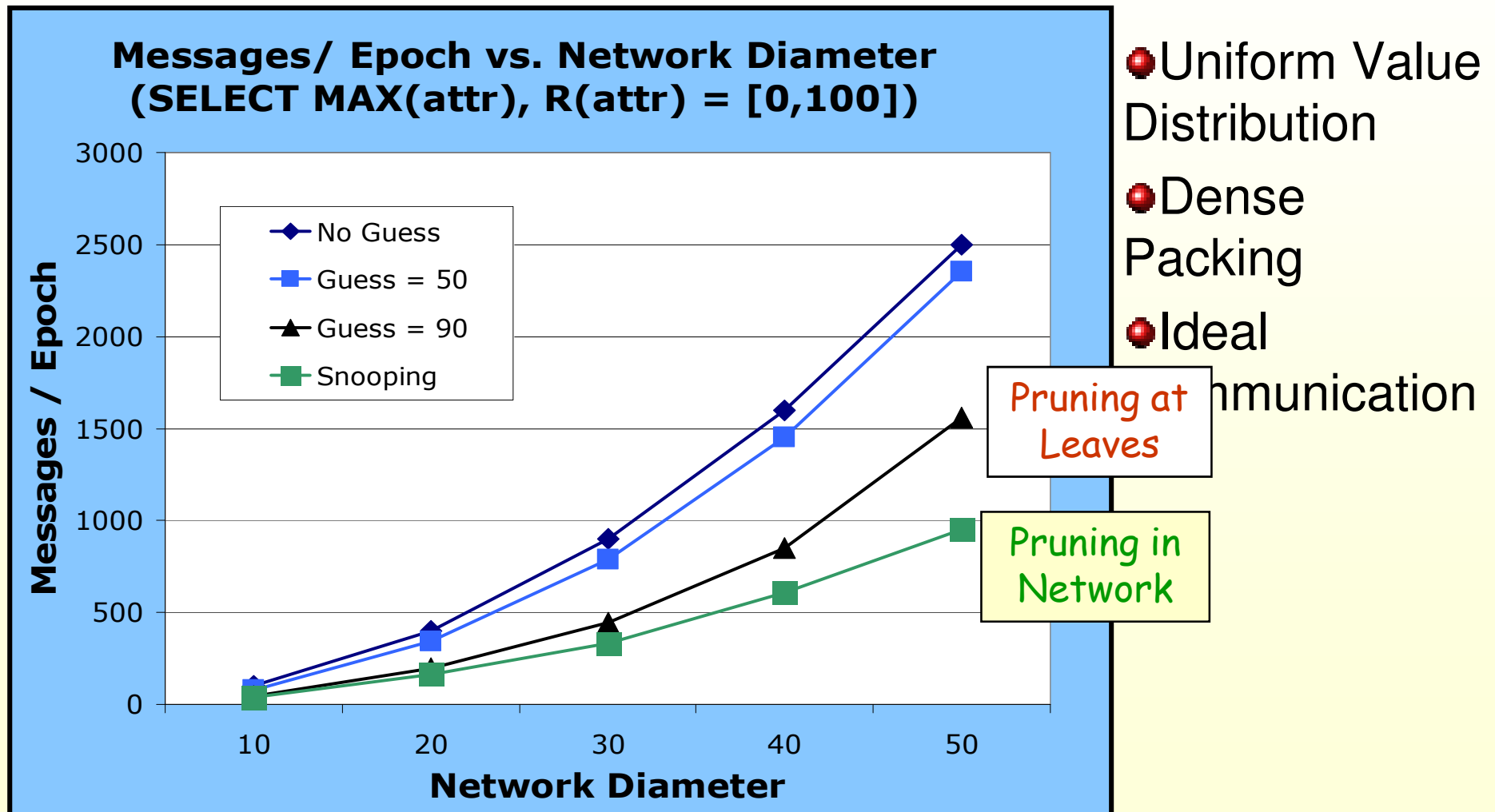
Optimization: Channel Sharing ("Snooping")

- Insight: Shared channel can reduce communication
- Suppress messages that won't affect aggregate
 - E.g., MAX
 - Applies to all **exemplary, monotonic** aggregates
- Only snoop in listen/transmit slots
 - Future work: explore snooping/listening tradeoffs

Optimization: Hypothesis Testing

- Insight: Guess from root can be used for message suppression
 - E.g. 'MAX > 100'
 - Works for **monotonic** & **exemplary** aggregates
 - Also **summary**, if imprecision allowed
- How is hypothesis computed?
 - Blind or statistically informed guess
 - Observation over network subset

Experiment: Snooping vs. Hypothesis Testing



Duplicate Sensitivity

<u>Property</u>	<u>Examples</u>	<u>Affects</u>
Partial State	MEDIAN : unbounded, MAX : 1 record	Effectiveness of TAG
Monotonicity	COUNT : monotonic AVG : non-monotonic	Hypothesis Testing, Snooping
Exemplary vs. Summary	MAX : exemplary COUNT: summary	Applicability of Sampling, Effect of Loss
Duplicate Sensitivity	MIN : dup. insensitive, AVG : dup. sensitive	Routing Redundancy

Using Multiple Parents

- Use graph structure
 - Increase delivery probability with no communication overhead
- For **duplicate insensitive** aggregates, or
- Aggs expressible as sum of parts
 - Send (part of) aggregate to all parents
 - In just one message, via multicast
 - Assuming independence, decreases variance

$$P(\text{link xmit successful}) = p$$

$$P(\text{success from } A \rightarrow R) = p^2$$

$$E(\text{cnt}) = c * p^2$$

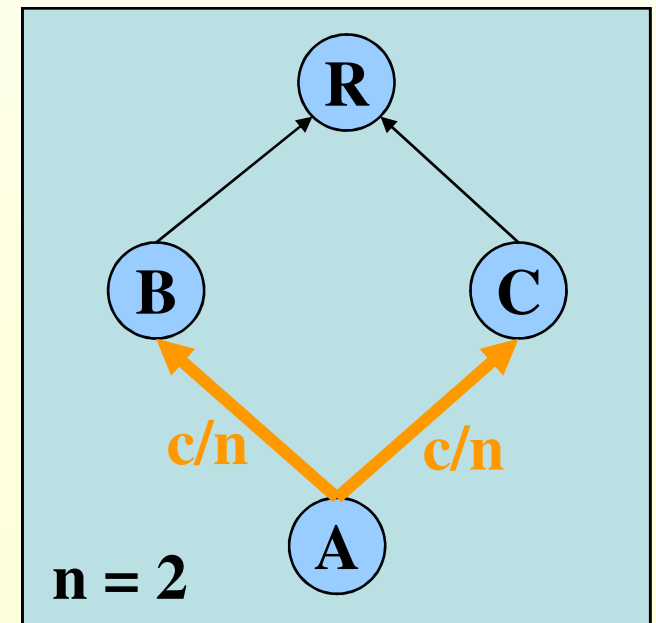
$$\text{Var}(\text{cnt}) = c^2 * p^2 * (1 - p^2) \\ \equiv \underline{V}$$

$$\# \text{ of parents} = n$$

$$E(\text{cnt}) = n * (c/n * p^2)$$

$$\text{Var}(\text{cnt}) = n * (c/n)^2 * \\ p^2 * (1 - p^2) = \underline{V/n}$$

SELECT COUNT(*)



Making Aggregates Duplicate Insensitive

- Possible to convert exact duplicate sensitive aggregates to inexact duplicate insensitive aggregates
- Idea based on Flajolet + Martin's counting sketches
 - Estimate no. values in set of size n in $O(\log n)$ bits
- Considine et al (ICDE 2004)
- Nath et al (SenSys 2004)

Will be covered next Monday, 10/29

Acquisitional Query Processing (ACQP)

- Closed world assumption does not hold
 - Could generate an infinite number of samples
- An acquisitional query processor **controls**
 - **when,**
 - **where,**
 - **and with what frequency data is collected**
- Versus traditional systems where data is provided *a priori*

ACQP: What is Different?

- How should the query be processed?
 - Sampling as a first class operation
 - Event – join duality
- How does the user control acquisition?
 - Rates or lifetimes
 - Event-based triggers
- Which nodes have relevant data?
 - Index-like data structures
- Which samples should be transmitted?
 - Prioritization, summary, and rate control

Event-Based Processing

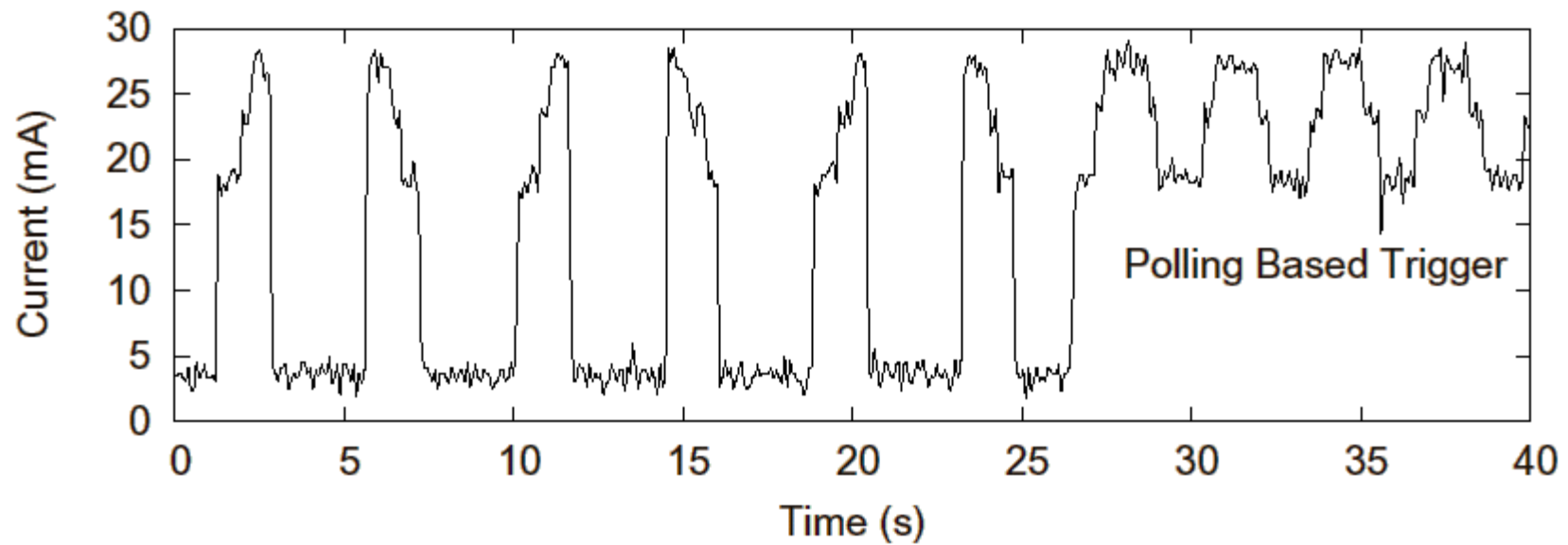
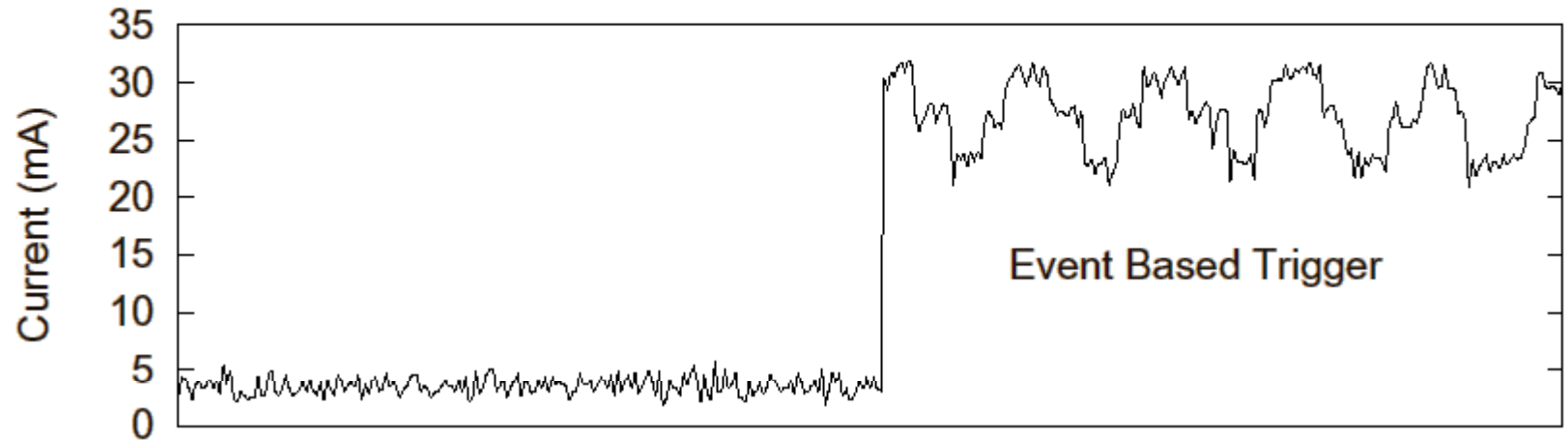
- ACQP – want to initiate queries in response to detected events

```
ON EVENT bird_detect(loc) AS bd
  SELECT AVG(s.light), AVG(s.temp)
  FROM sensors AS s
  WHERE dist(bd.loc,s.loc) < 10m
  SAMPLE PERIOD 1s for 10
```



Event Based Processing

Time v. Current Draw

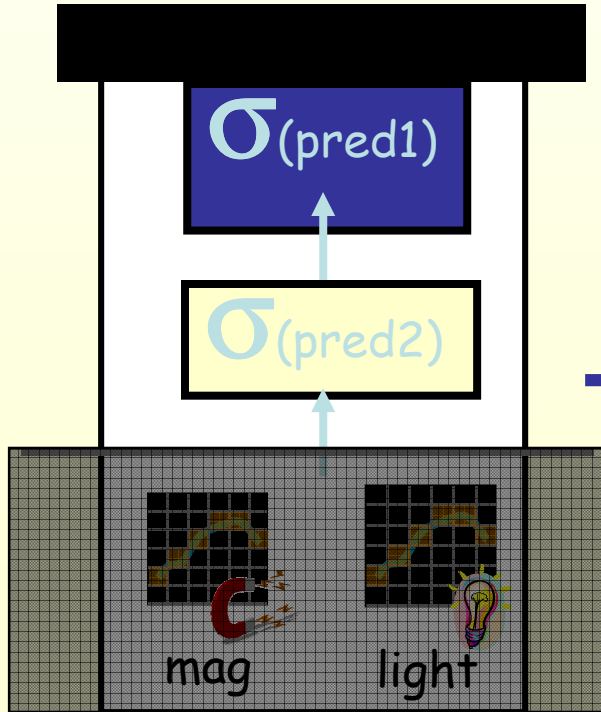


Operator Ordering: Interleave Sampling + Selection

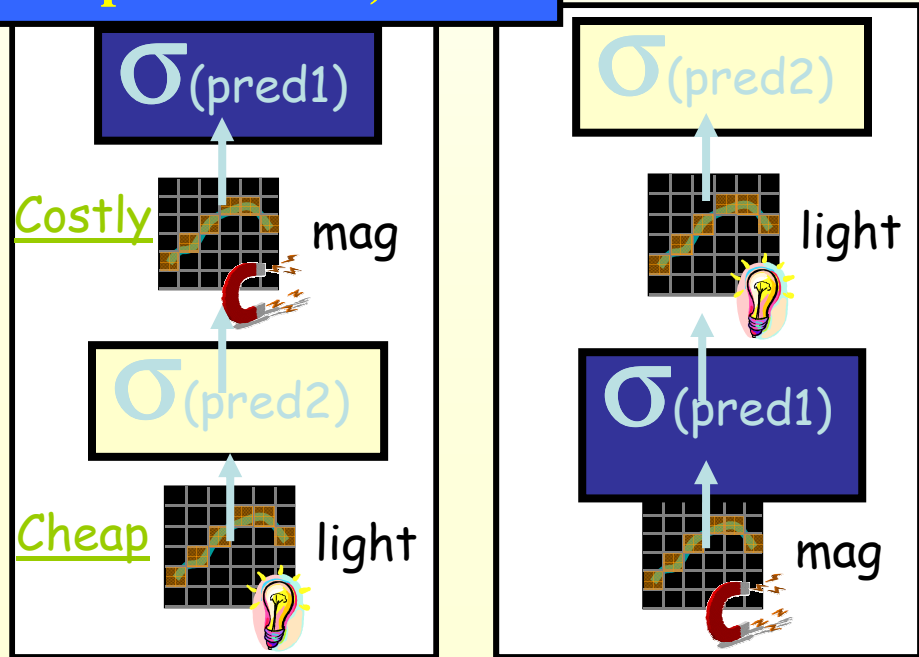
SELECT light, mag
 FROM sensors
 WHERE pred1(mag)
 AND pred2(light)
 EPOCH DURATION 1s

At 1 sample / sec, total power savings could be as much as 3.5mW → Comparable to processor!

Correct ordering
 (unless pred1 is very selective and pred2 is not):



ACQP →



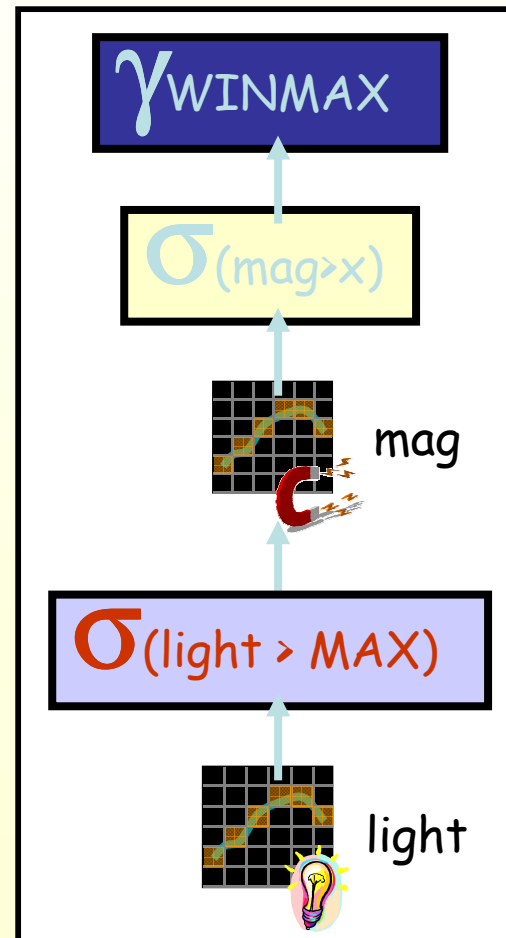
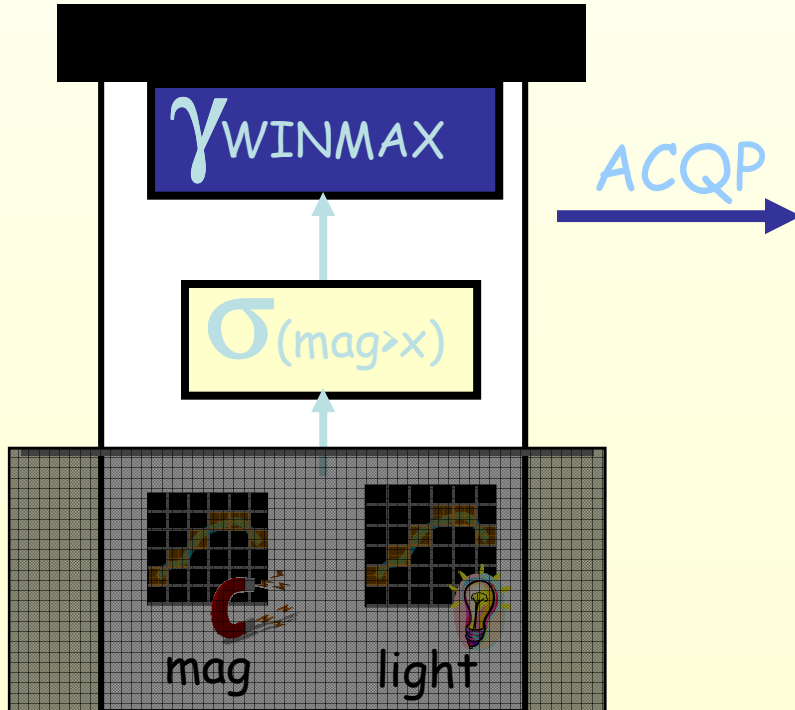
Exemplary Aggregate Pushdown

SELECT WINMAX(light,8s,8s)

FROM sensors

WHERE mag > x

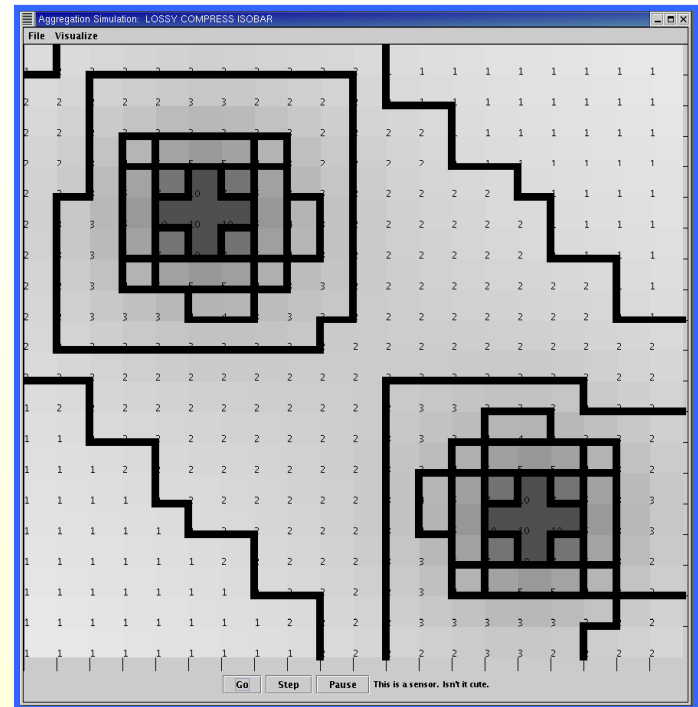
EPOCH DURATION 1s



- Novel, general pushdown technique
- Mag sampling is the most expensive operation!

Sensor Network Challenge DB Problems

- Temporal aggregates
- Sophisticated, sensor network specific aggregates
 - Isobar Finding
 - Vehicle Tracking
 - Lossy compression
 - Wavelets

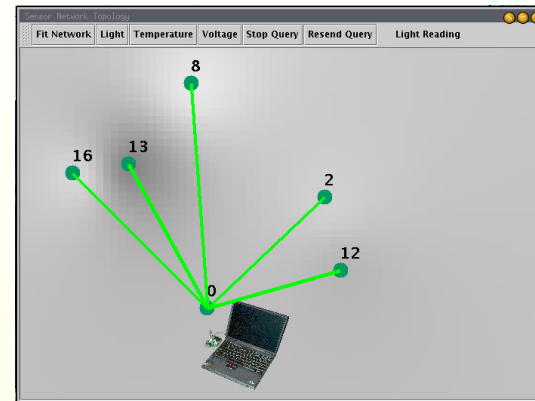


"Isobar Finding"

TinyDB Deployments

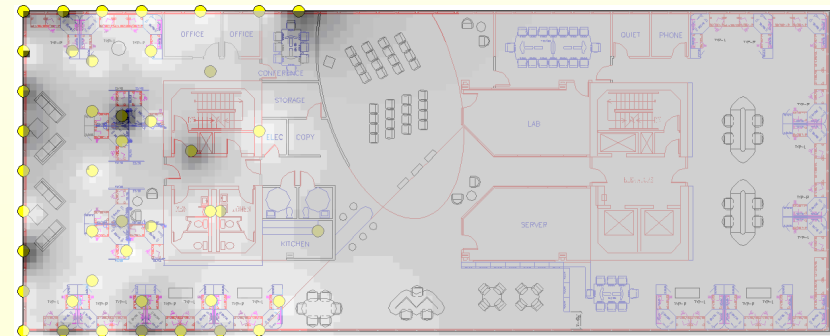
- Initial efforts:

- Network monitoring
- Vehicle tracking



- Ongoing deployments:

- Environmental monitoring
- Generic Sensor Kit
- Building Monitoring
- Golden Gate Bridge



TAG Contributions

- Simple but powerful data collection language
 - Vehicle tracking:

```
SELECT ONEMAX(mag,nodeid)  
EPOCH DURATION 50ms
```

- Distributed algorithm for in-network aggregation
 - Communication Reducing
 - Power Aware
 - Integration of sleeping, computation
 - Predicate-based grouping
- Taxonomy driven API
 - Enables transparent application of techniques to
 - Improve quality (parent splitting)
 - Reduce communication (snooping, hypo. testing)

Summary

- Declarative queries are the right interface for data collection in sensor nets!
 - Easier, faster, & more robust
- Acquisitional Query Processing
 - Framework for addresses many new issues that arise in sensor networks, e.g.
 - Order of sampling and selection
 - Languages, indices, approximations that give user control over *which data enters the system*

Range Queries, Distributed Storage

Range Queries

- Range queries ask for attribute readings with data values in certain ranges, e.g., temperature $T \in [-15 \text{ C}, +15 \text{ C}]$
- They are well-suited to data with uncertainty, such as sensor readings
- Usually multiple attributes are involved
- Typically, the number of records satisfying the query is small compared to the total number of records

Data-Base Indices

- When repeated queries are made on the same data, it makes sense to preprocess the database so as to make the query processing faster
- The auxiliary structures we build to facilitate this processing are called **indices**
- A large body of literature exists on building indices for one-dimensional attributes

Metrics for Evaluating Indices

- For a data base of n records, the relevant metrics are
 - the index size, $S(n)$
 - the preprocessing time required to build the index, $P(n)$
 - the query cost the index enables, $Q(n)$
 - the update cost to allow for record insertions and deletions to the database, $U(n)$

Distributed Range Searching

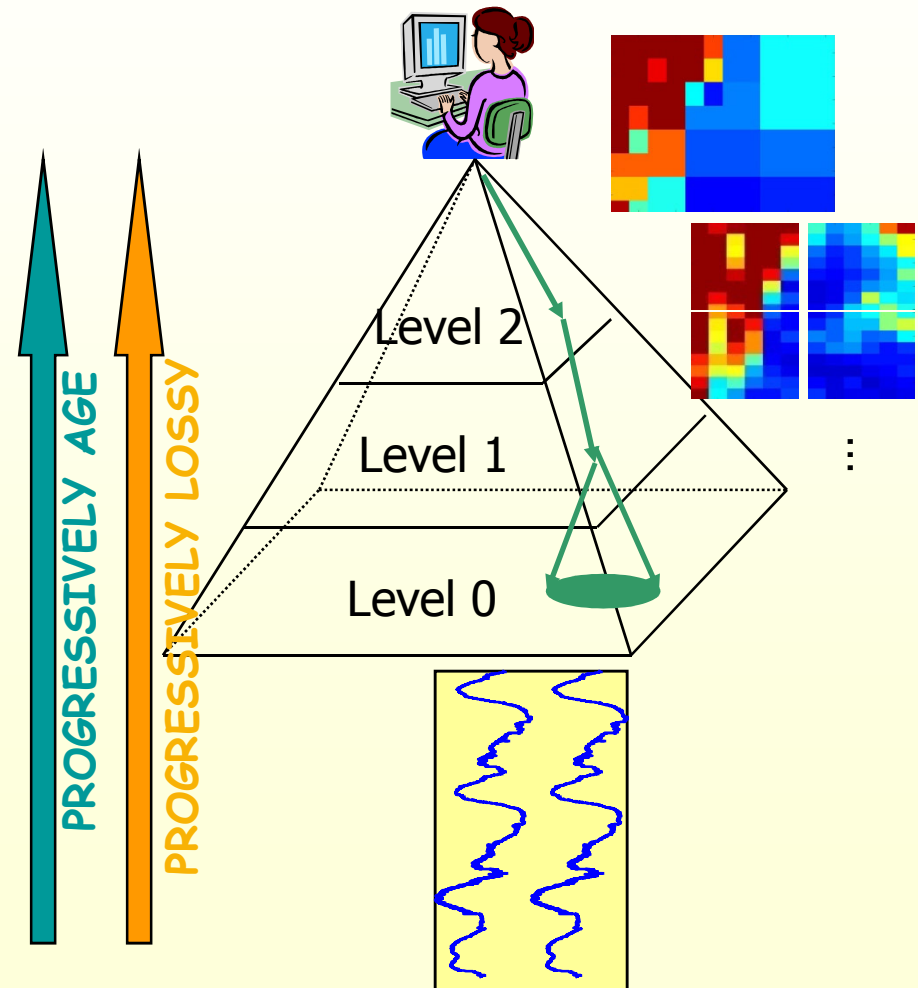
- All structures we saw so far are **hierarchical** – but in a distributed setting nodes that hold data close to the root are likely to be overloaded
- We discuss one sensor network range searching approaches
 - The DIMENSIONS system from UCLA

Some Issues to Consider

- How is information aggregated spatially and temporally?
- How does the system decide where to store information?
- How are queries routed to the correct nodes?
- What steps does the system take to reduce energy use?

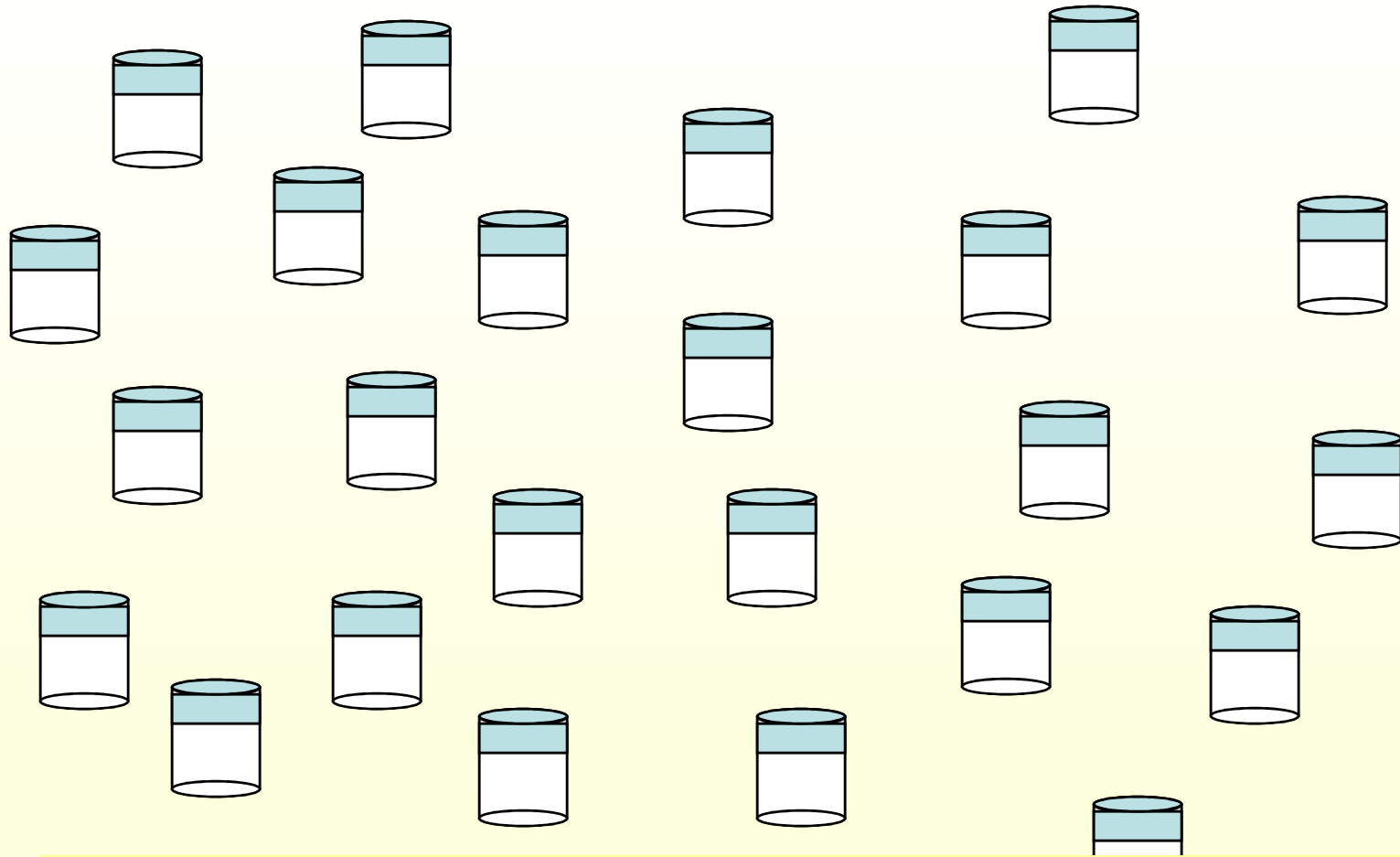
DIMENSIONS System: Key Ideas

- Construct distributed load-balanced quad-tree hierarchy of *lossy wavelet-compressed summaries* corresponding to different resolutions and spatio-temporal scales.
- Queries *drill-down* from root of hierarchy to *focus search* on small portions of the network.
- *Progressively age* summaries for long-term storage and graceful degradation of query quality over time.



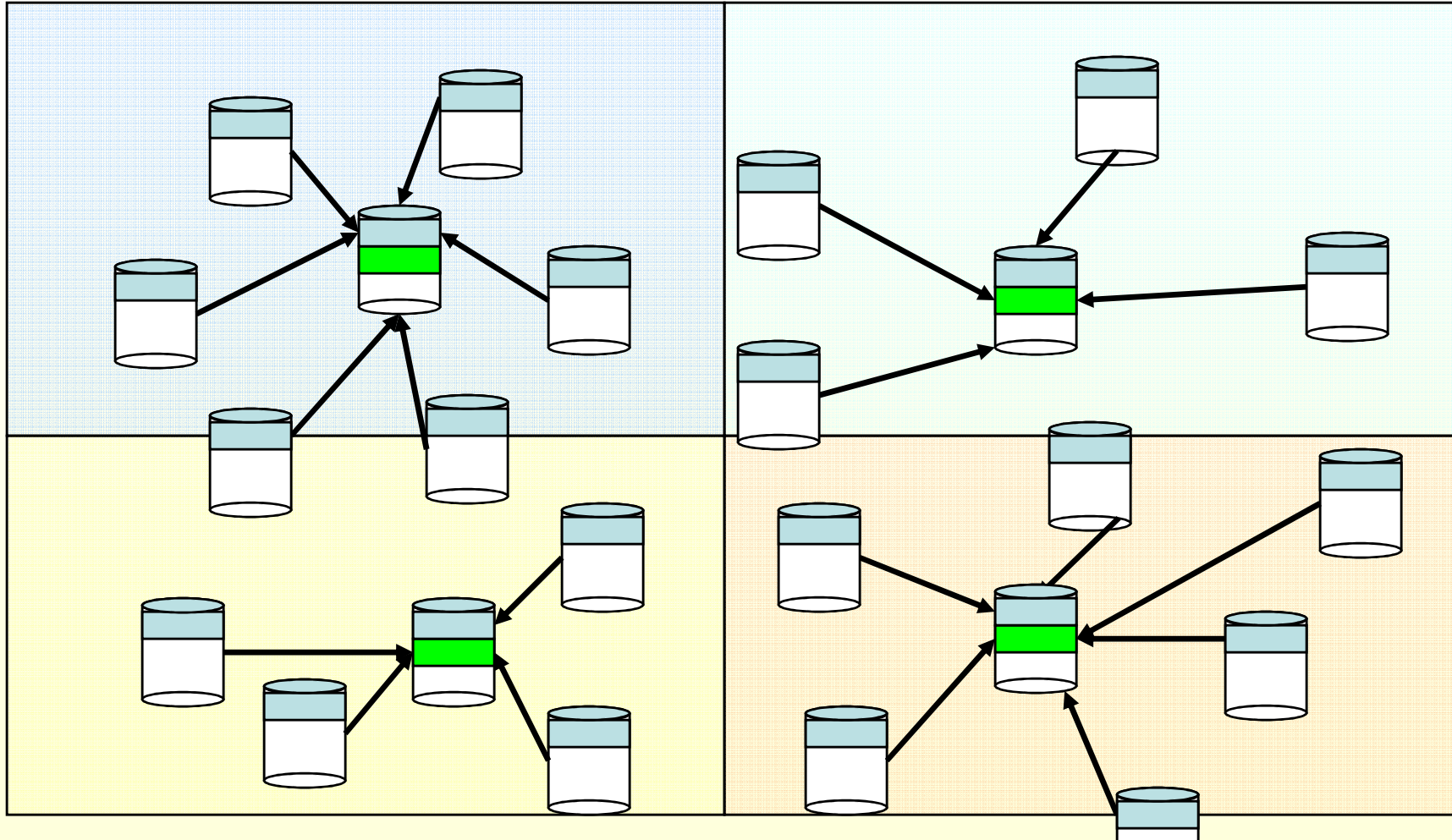
[From Ganesan, et., al., 2003]

Constructing the Hierarchy



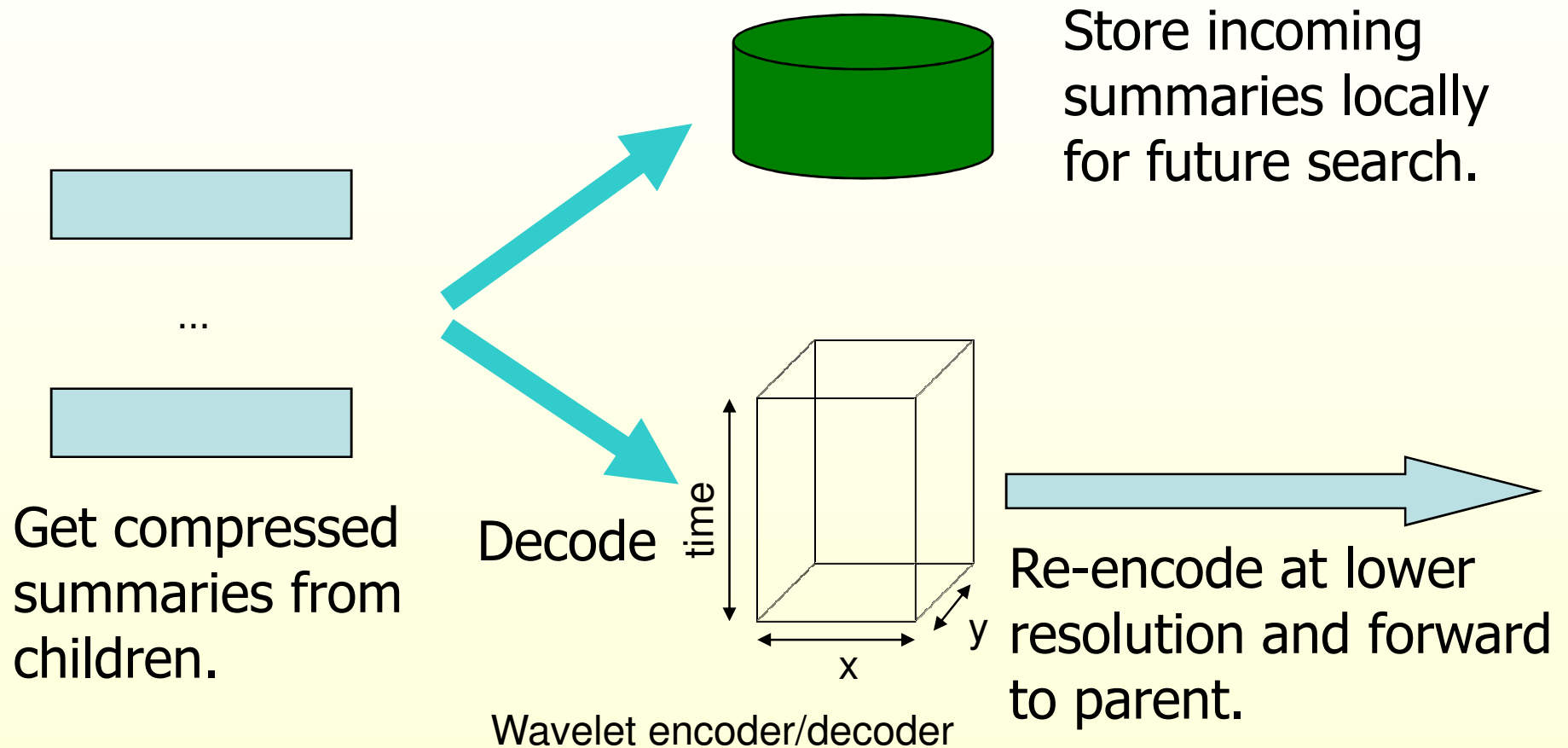
Initially, nodes fill up their own storage with raw sampled data.

Constructing the Hierarchy

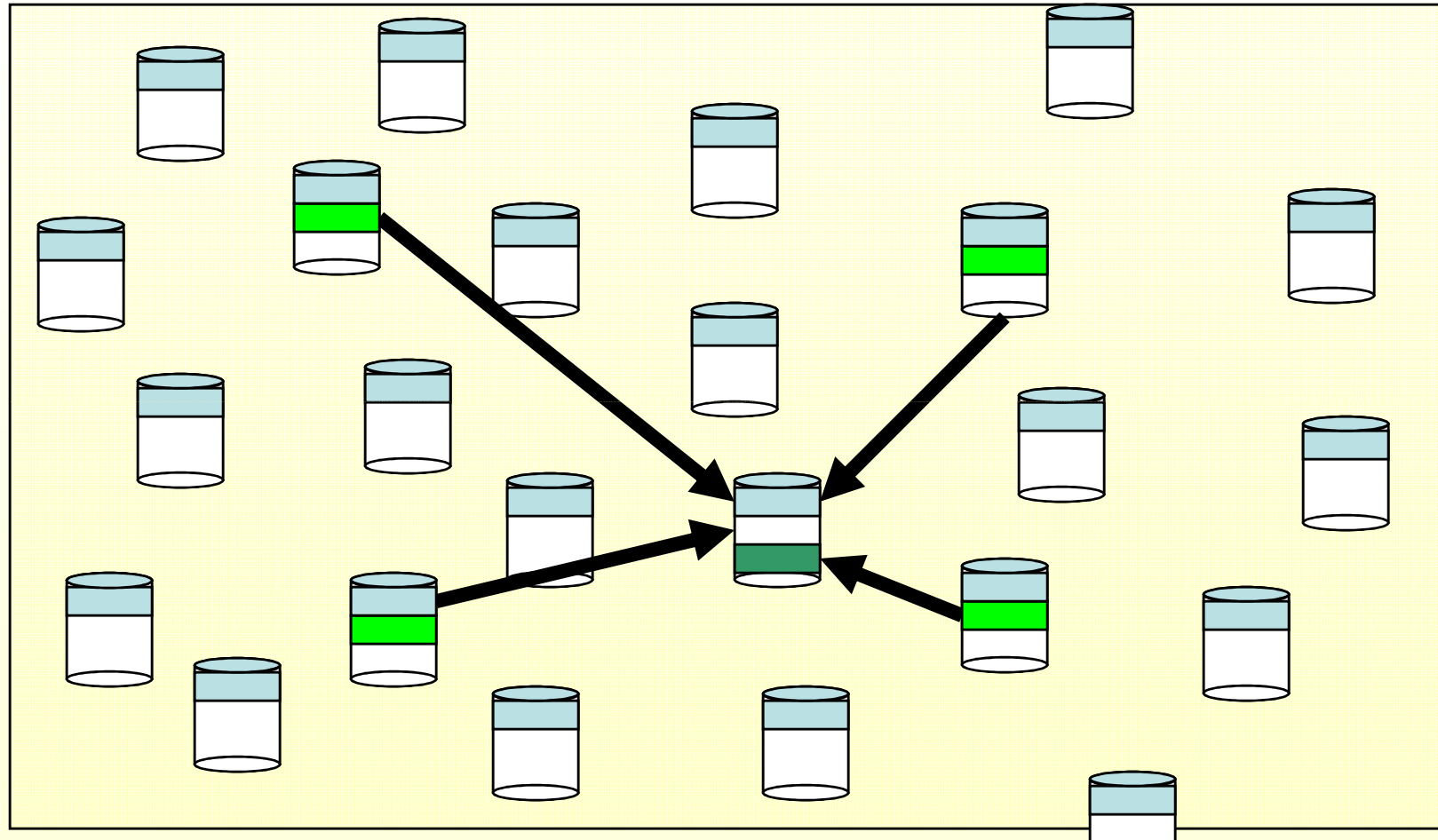


- Tessellate the network space into a grid; use hashing in each cell to determine location of clusterhead (ref: DCS).
- Send wavelet-compressed local time-series to clusterhead.

Processing at Each Level

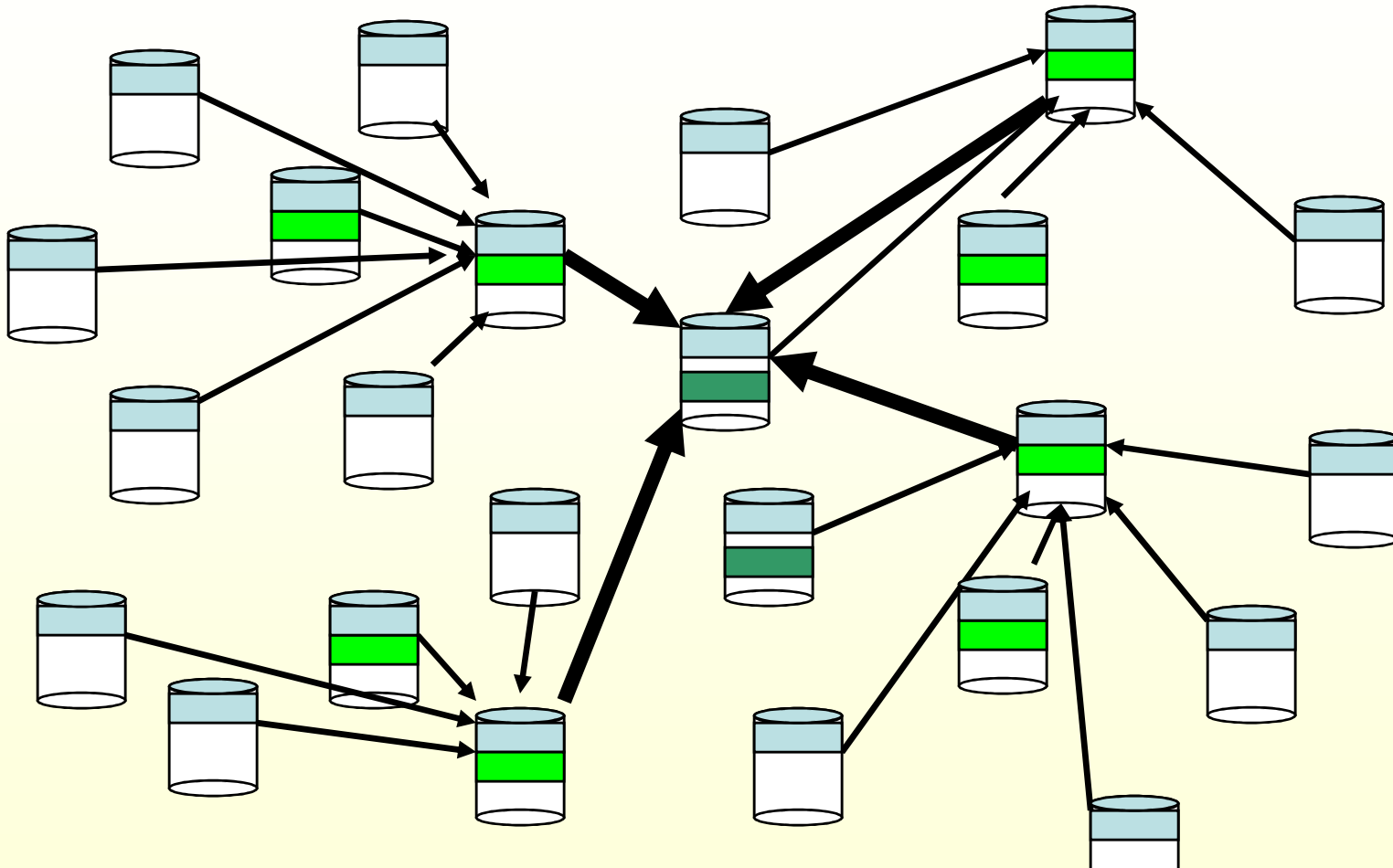


Constructing the Hierarchy



Recursively send data to higher levels of the hierarchy.

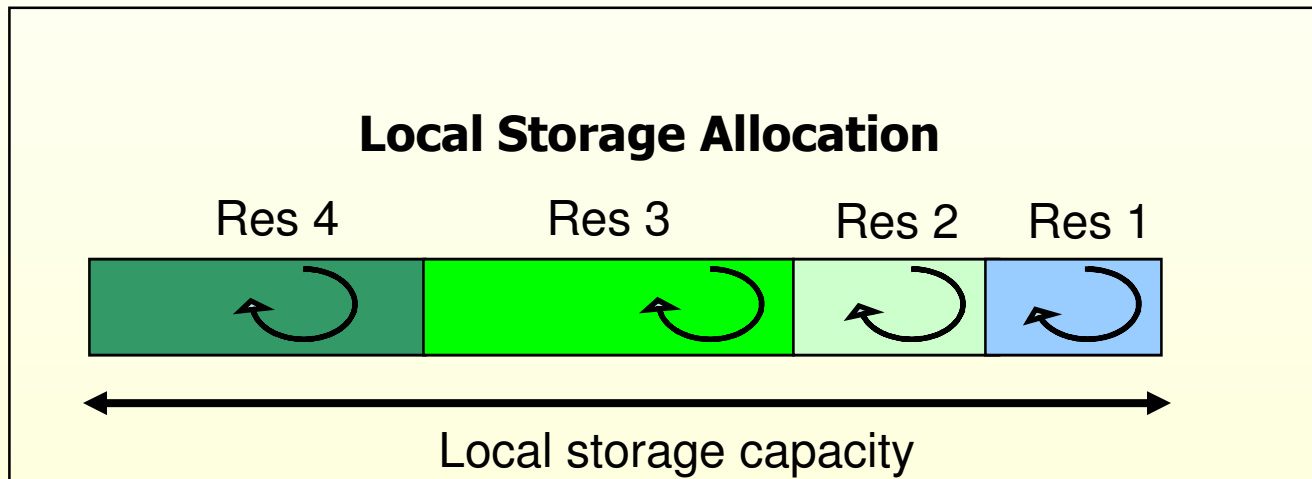
Distributing the Storage Load



Hash to different locations over time to distribute load among nodes in the network.

What Happens when Storage Fills Up?

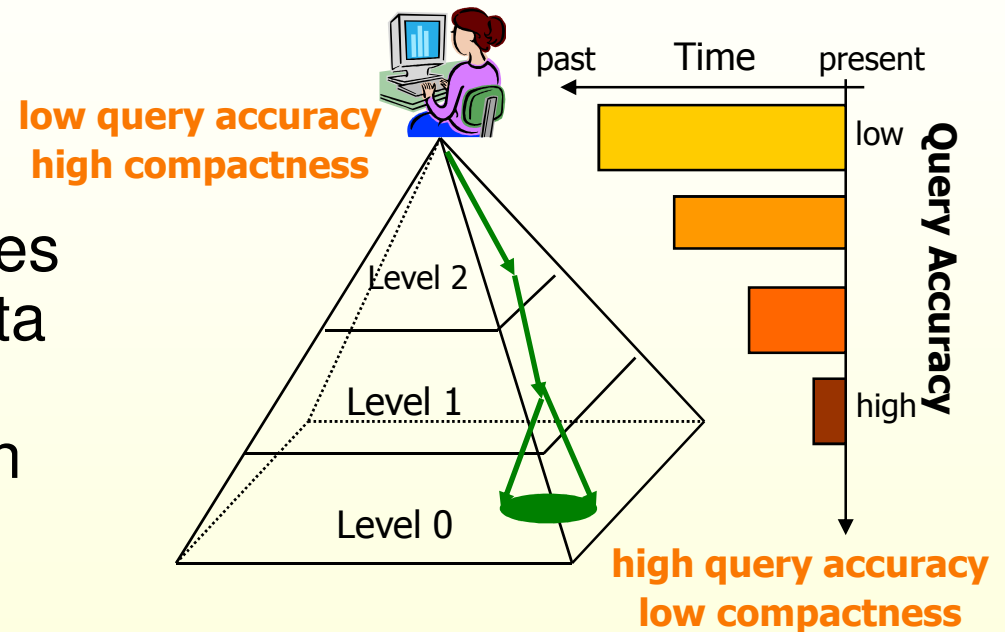
- Eventually, all available storage gets filled, and we have to decide when and how to drop summaries.



- Allocate storage to each resolution and use each allocated storage block as a circular buffer.

Tradeoff Between Age and Storage Requirements for Summary

- *Graceful Query Degradation*: Provide more accurate responses to queries on recent data and less accurate responses to queries on older data.

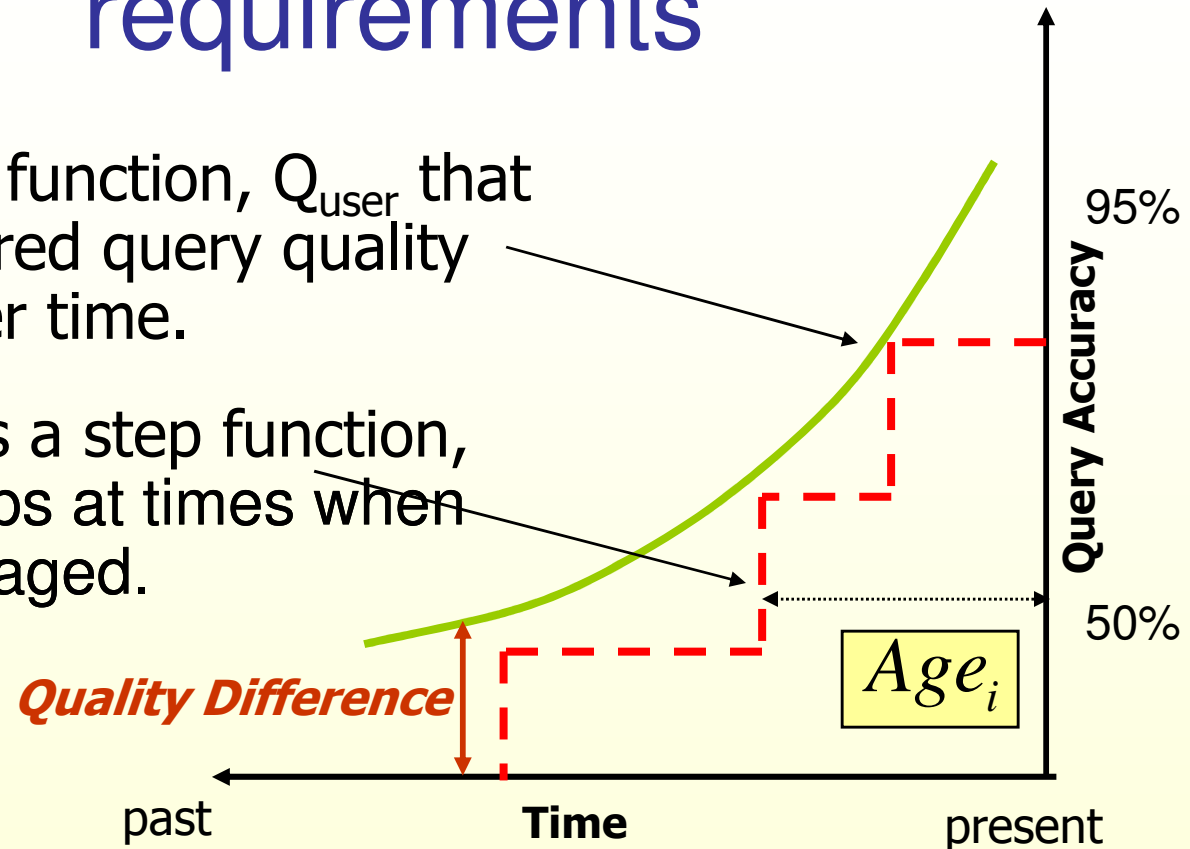


How do we allocate storage at each node to summaries at different resolutions to provide gracefully degrading storage and search capability?

Match system performance to user requirements

User provides a function, Q_{user} that represents desired query quality degradation over time.

System provides a step function, Q_{system} , with steps at times when summaries are aged.



Objective: Minimize worst case difference between user-desired query quality (green curve) and query quality that the system can provide (red step function).

What Do We Know?

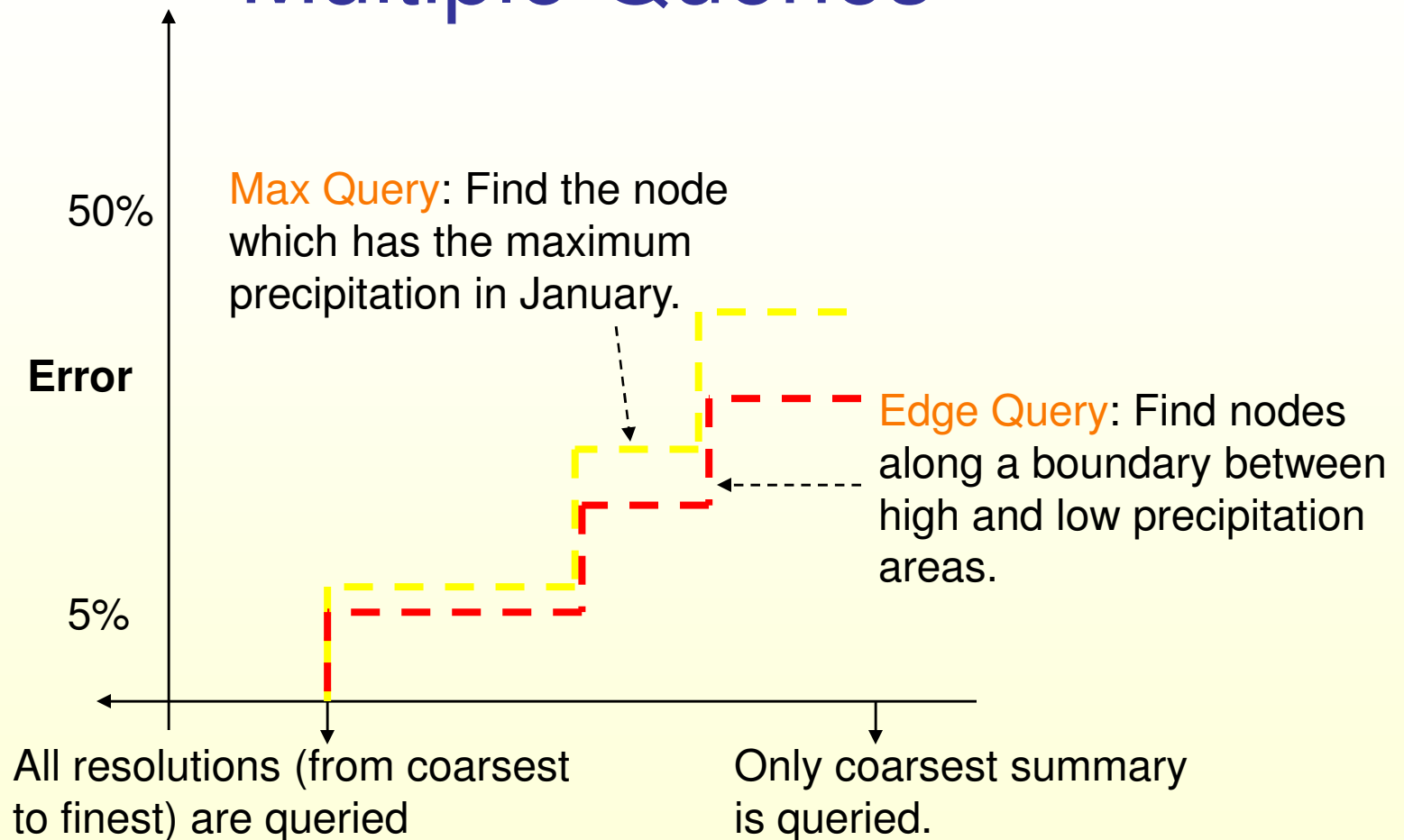
Given

- N sensor nodes.
- Each node has storage capacity, S .
- Data is generated at resolution i at rate R_i .
- Q_{user} - User-desired quality degradation.

We might be provided

- a set of typical queries, T , that the user provides.
- $D(q,k)$ – Query error when drilldown for query q terminates at level k .

Determining Query Quality from Multiple Queries



We need to translate the performance of different drill-down queries to a single “query quality” metric.

Definition: Query Quality

- Given:
 - T = set of typical queries.
 - $D(q,k)$ = Query error when drill-down for query q in set T terminates at resolution k .
- The query quality for queries that refer to data at time t in the past, $Q_{system}(t)$, if k is the finest available resolution is:

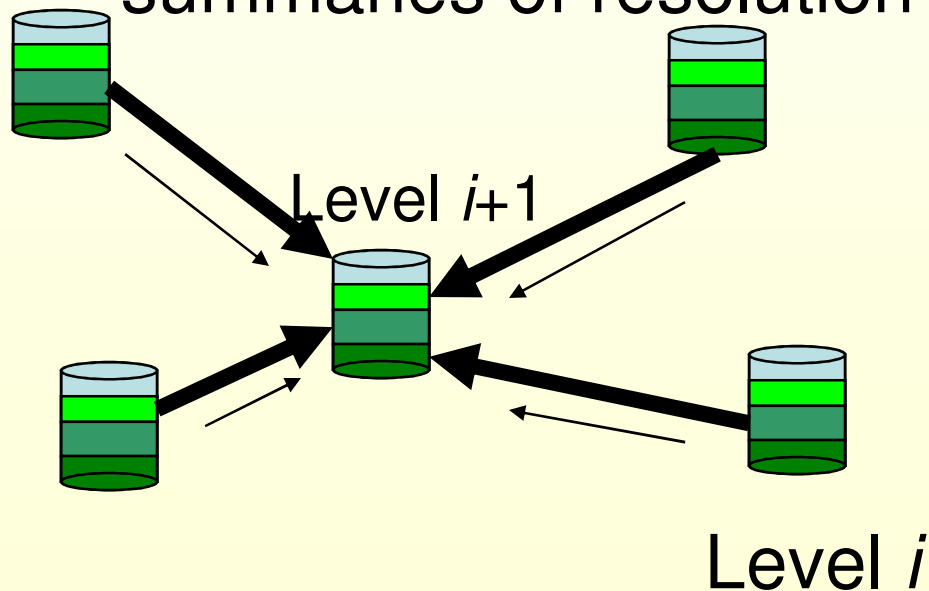
$$Q_{system}(t) = \frac{1}{|T|} \sum_{q \in T} D(q, k)$$

How Many Levels of Resolution k are Available at Time t ?

Given:

R_i = Total transmitted data rate from level i clusterheads to level $i+1$ clusterheads.

Define s_i = storage allocated at each node for summaries of resolution i .



$$Age_i = \frac{Ns_i}{R_i}$$

Storage Allocation: Constraint-Optimization problem

- **Objective:** Find $\{s_i\}$, $i=1..\log_4 N$ that:

$$\min_{t=-\infty..0} \max Q_{user}(t) - Q_{system}(t)$$

- **Given constraints:**

- **Storage constraint:** Each node cannot store any greater than its storage limit.

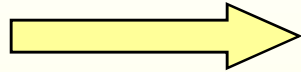
$$\sum_{i=1}^{\log_4 N} s_i \leq S$$

- **Drill-down constraint:** It is not useful to store finer resolution data if coarser resolutions of the same data is not present.

$$Age_{i+1} \geq Age_i$$

Determining Rates and Drilldown Query Errors

$$R_i$$



How do we determine communication rates to bound query error?

Assume: Rates are fixed a-priori by communication constraints.

$$D(q, k)$$



How do we determine the drill-down query error when prior information about deployment and data is limited?

Prior Information about Sampled Data

full a priori information

Omniscient Strategy

Baseline. Use all data to decide optimal allocation.

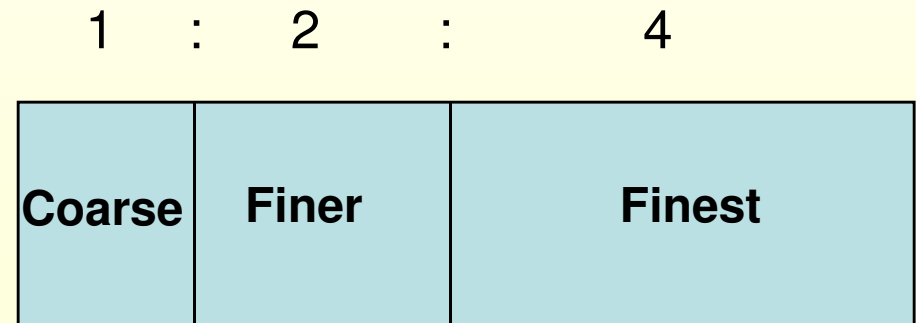
Training Strategy

(can be used when small training dataset from initial deployment).

Greedy Strategy

(when no data is available, use a simple weighted allocation to summaries).

No a priori information

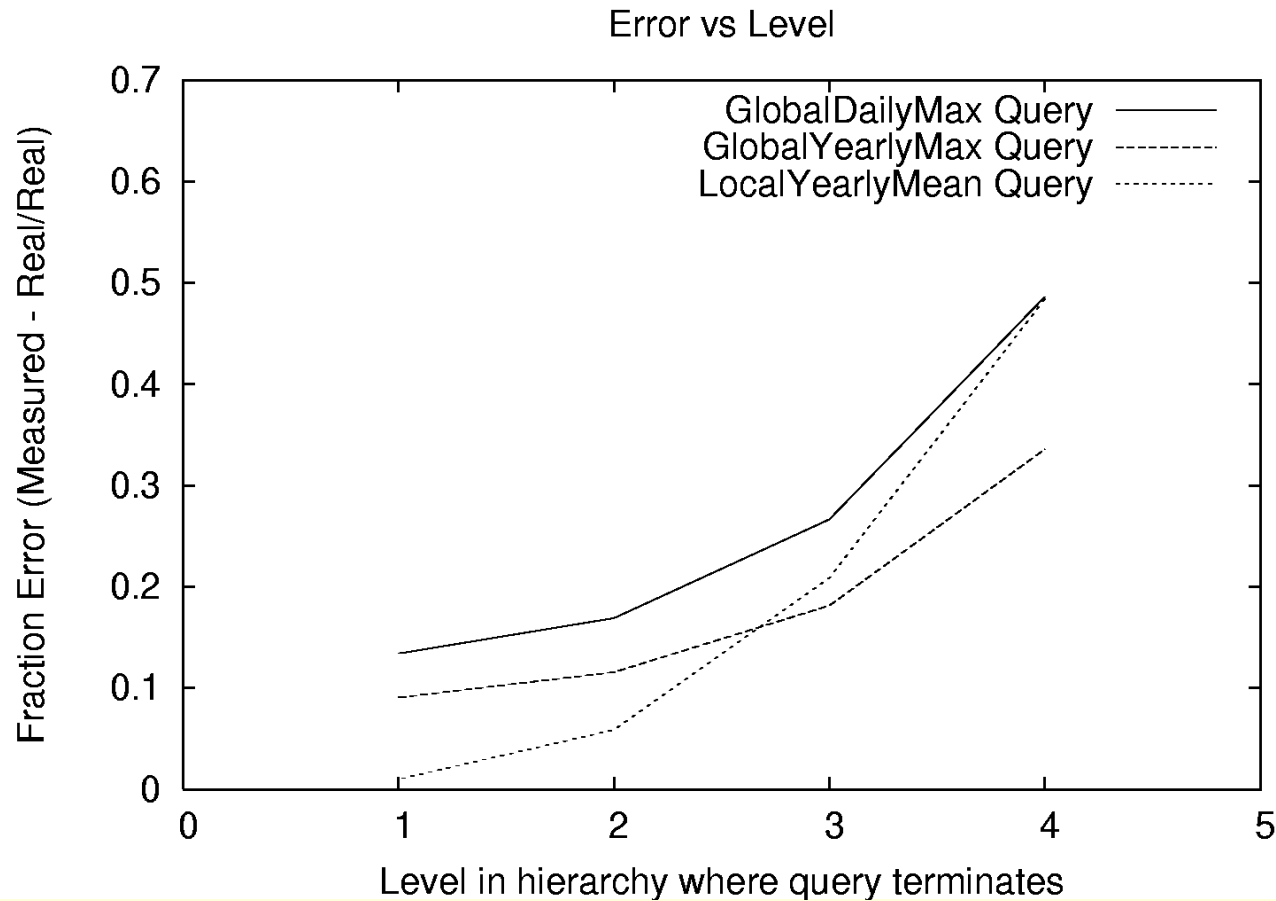


Distributed Trace-Driven Implementation

- Linux implementation for ipaq-class nodes
 - uses Emstar (J. Elson et al), a Linux-based emulator/simulator for sensor networks.
 - 3D Wavelet codec based on freeware by Geoff Davis available at: <http://www.geoffdavis.net>.
 - Query processing in Matlab.
- Geo-spatial precipitation dataset
 - 15x12 grid (50km edge) of precipitation data from 1949-1994, from Pacific Northwest[†]. (Caveat: Not real sensor data).
- System parameters
 - compression ratio: 6:12:24:48.
 - Training set: 6% of total dataset.

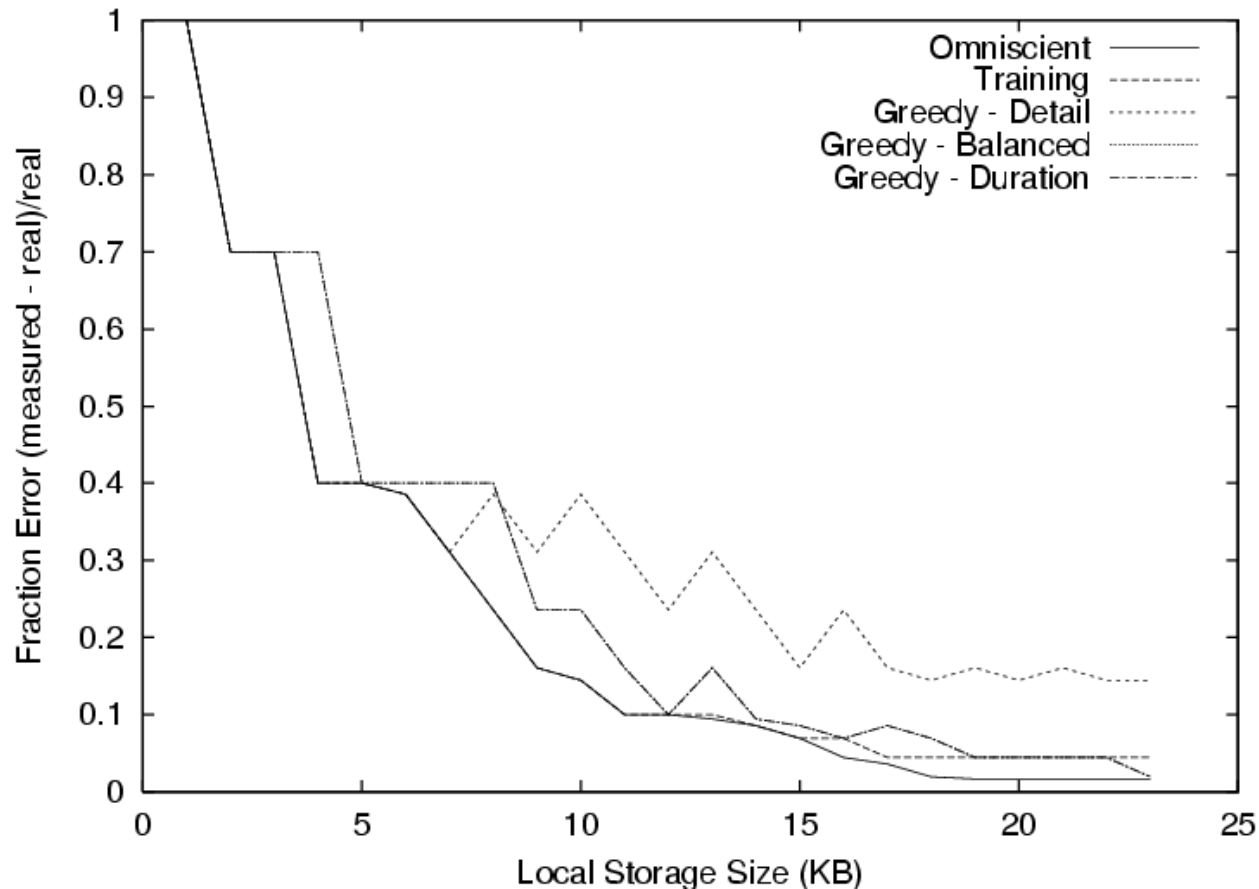
[†] M. Widmann and C. Bretherton. 50 km resolution daily precipitation for the Pacific Northwest, 1949-94.

How Efficient is The Search?



Search is very efficient (<5% of network queried) and accurate for different queries studied.

Comparing Aging Strategies



Training performs within 1% to optimal . Careful selection of parameters for the greedy algorithm can provide surprisingly good results (within 2-5% of optimal).

Conclusion

- Range searching is an important capability for sensor networks
- To allow efficient query processing, data aggregation over space and time is required
- Many methods employ hierarchical structures
- New communication problems arise in how to avoid overloading nodes high in the hierarchy
- Limited node memory implies that data ageing issues have to be addressed

The End