Querying the Sensor Network
TinyDB/TAG

NORTHWESTERN UNIVERSITY

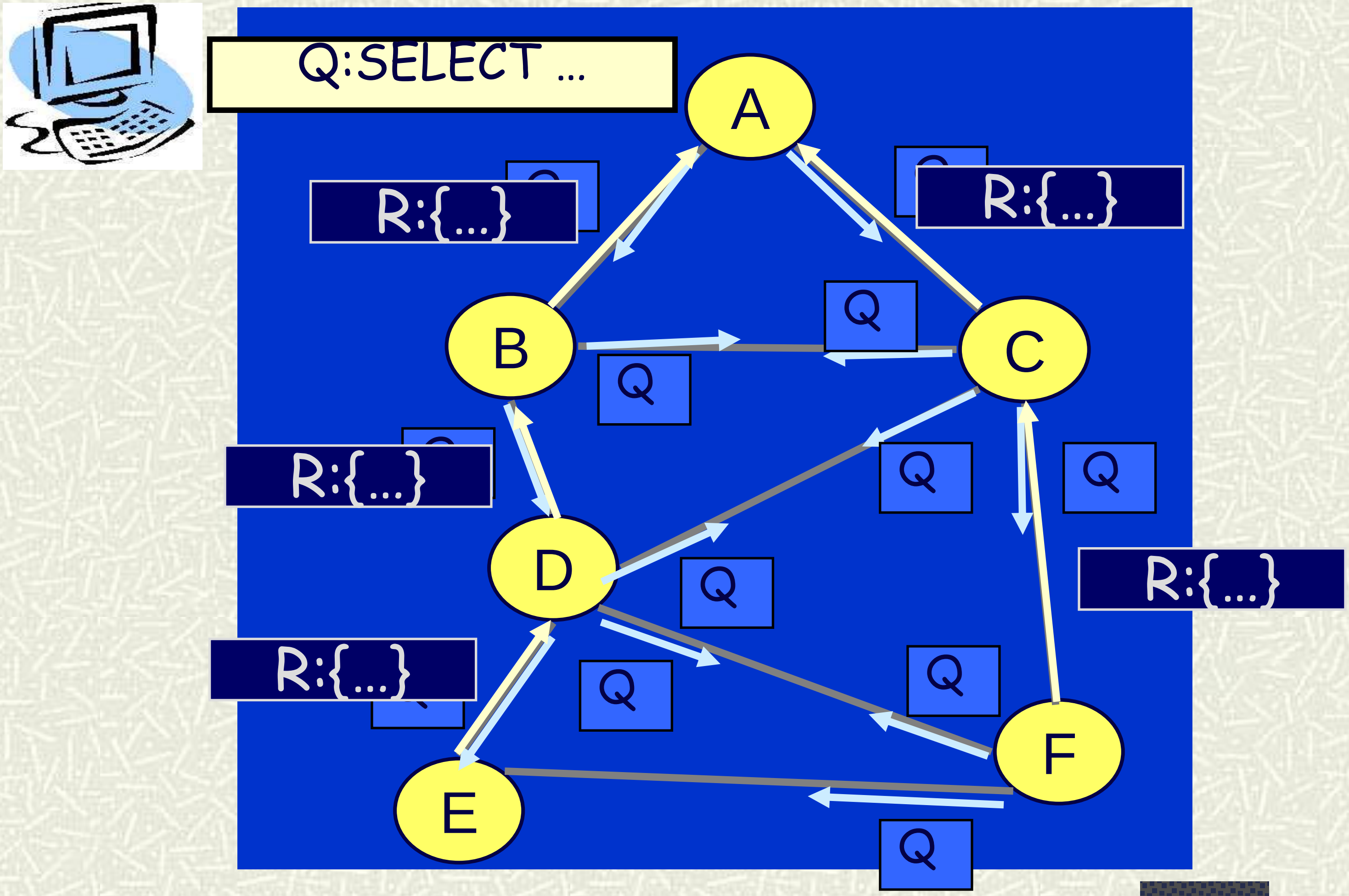
TAG: Tiny Aggregation

- Query Distribution: aggregate queries are pushed down the network to construct a spanning tree.
  - Root broadcasts the query and specifies its *level l*
  - Each node that hears message assigns its own level to be *l+1* and chooses as parent a node with smallest level.
  - Each node rebroadcasts message until all nodes have received it
  - Resulting 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.

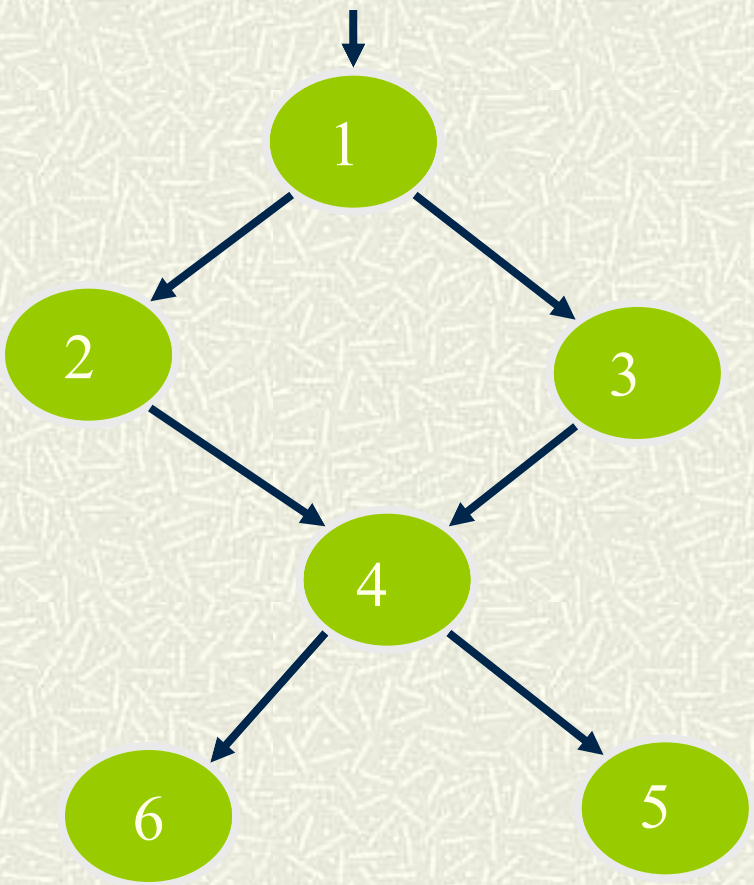
Tree-based Routing

- Tree-based routing
  - Used in:
    - Query delivery
    - Data collection
    - In-network aggregation

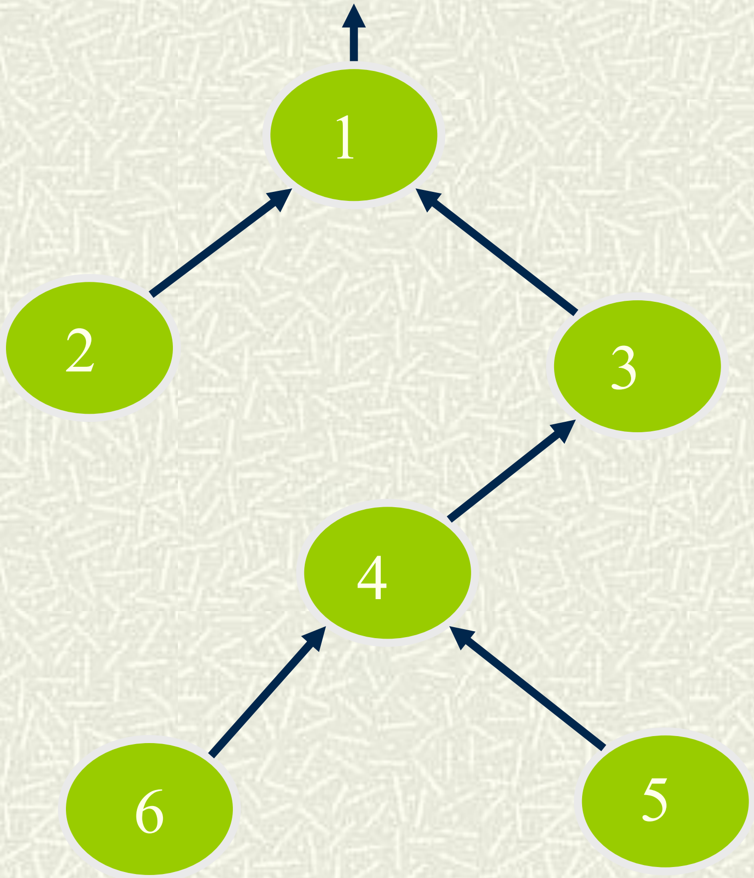


TAG example

Query distribution



Query collection



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
- On server, all attributes are defined in catalog.xml
- Discussion: other alternative data models?

Query Language (TinySQL)

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

Comparison with SQL

- Single table in FROM clause (exception: storage points…)
- Only conjunctive comparison predicates in WHERE and HAVING
- No subqueries
- No column alias in SELECT clause
- Arithmetic expressions limited to *column op constant*
- Only fundamental difference: SAMPLE PERIOD clause

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

NORTHWESTERN UNIVERSITY

TinySQL Examples (cont.)

(2) SELECT AVG(sound)

FROM sensors

EPOCH DURATION 10s

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

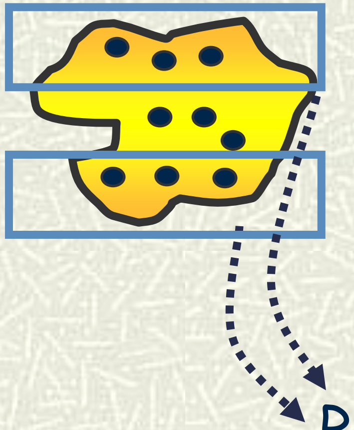
(3) SELECT region, CNT(occupied) AVG(sound)

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

NORTHWESTERN UNIVERSITY

Basic Aggregation

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

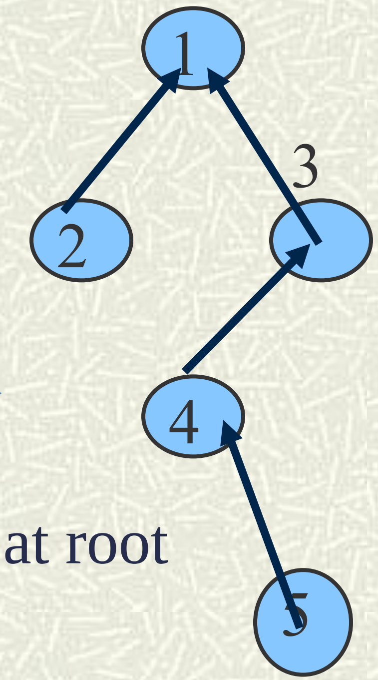


Illustration: Aggregation

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

Sensor #

Interval #

	1	2	3	4	5
4					1
3					
2					
1					
4					

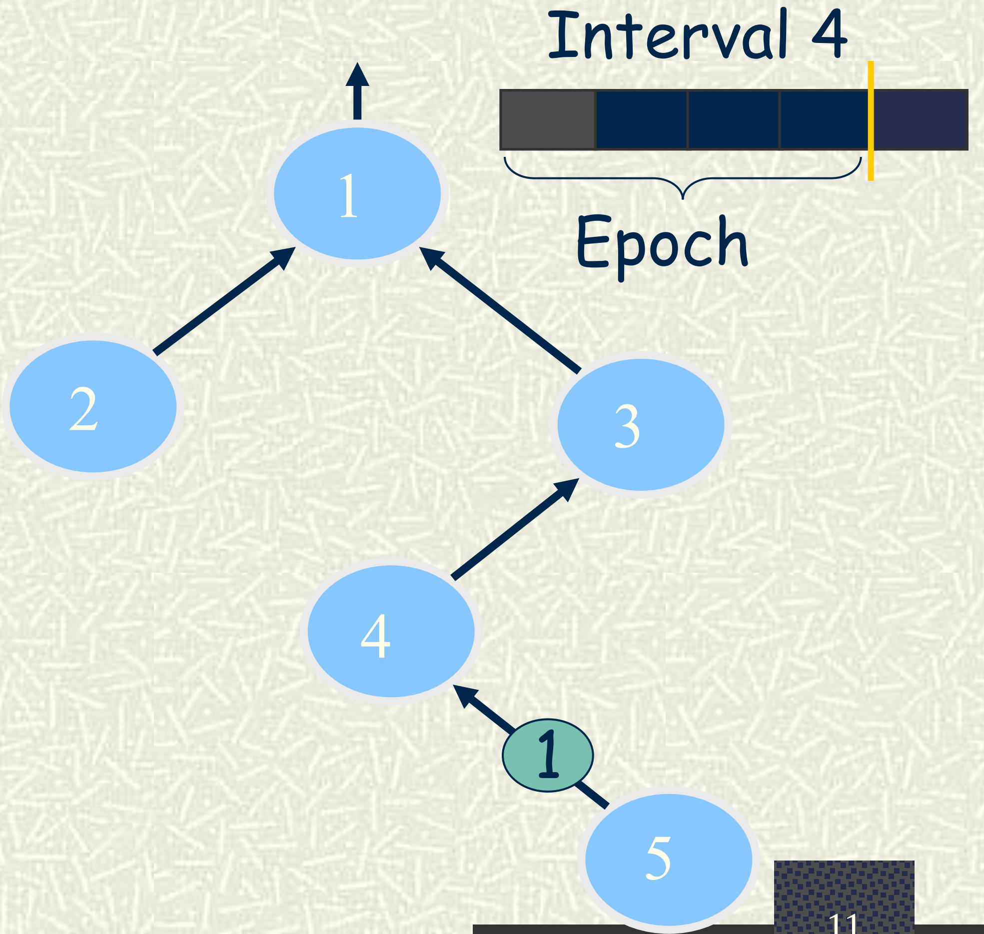
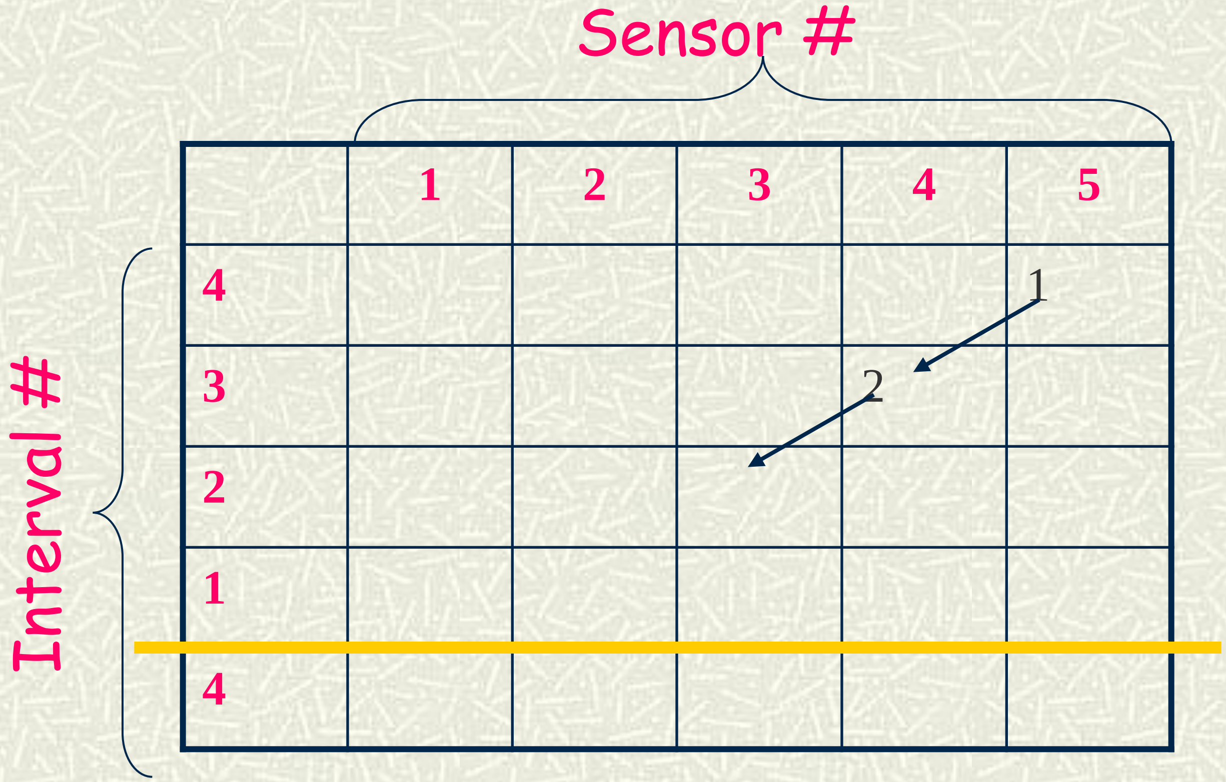


Illustration: Aggregation

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



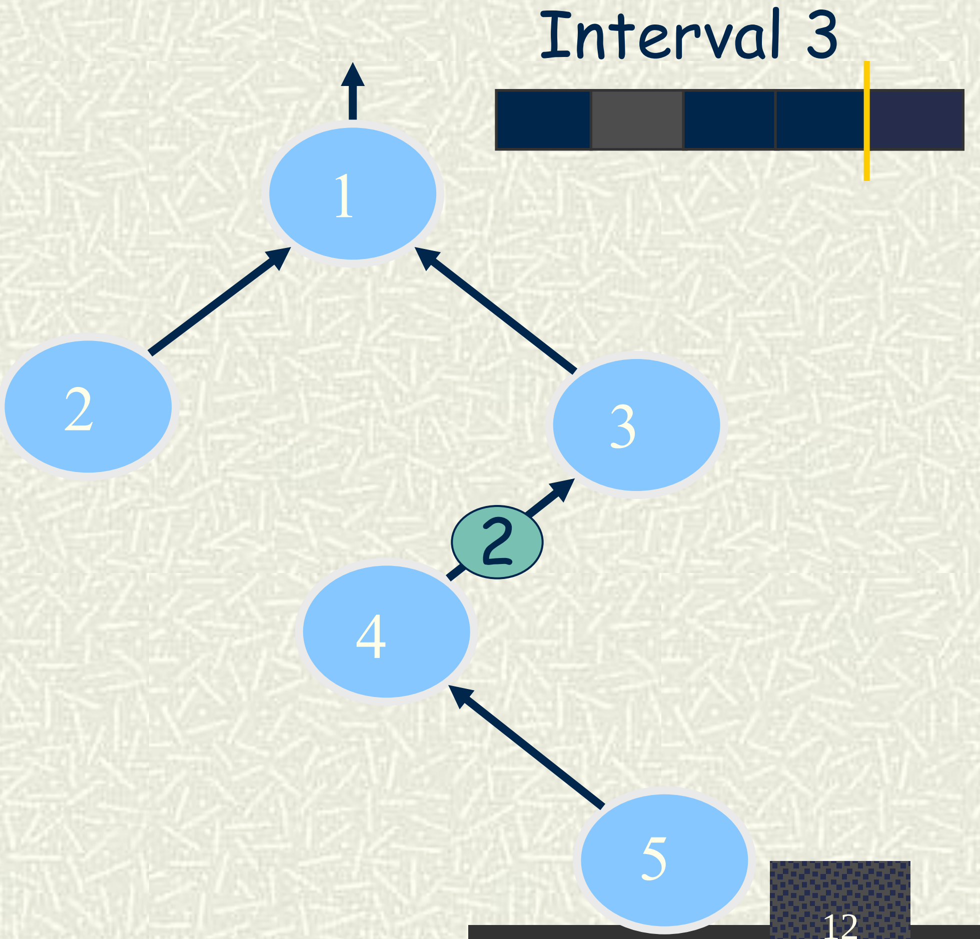
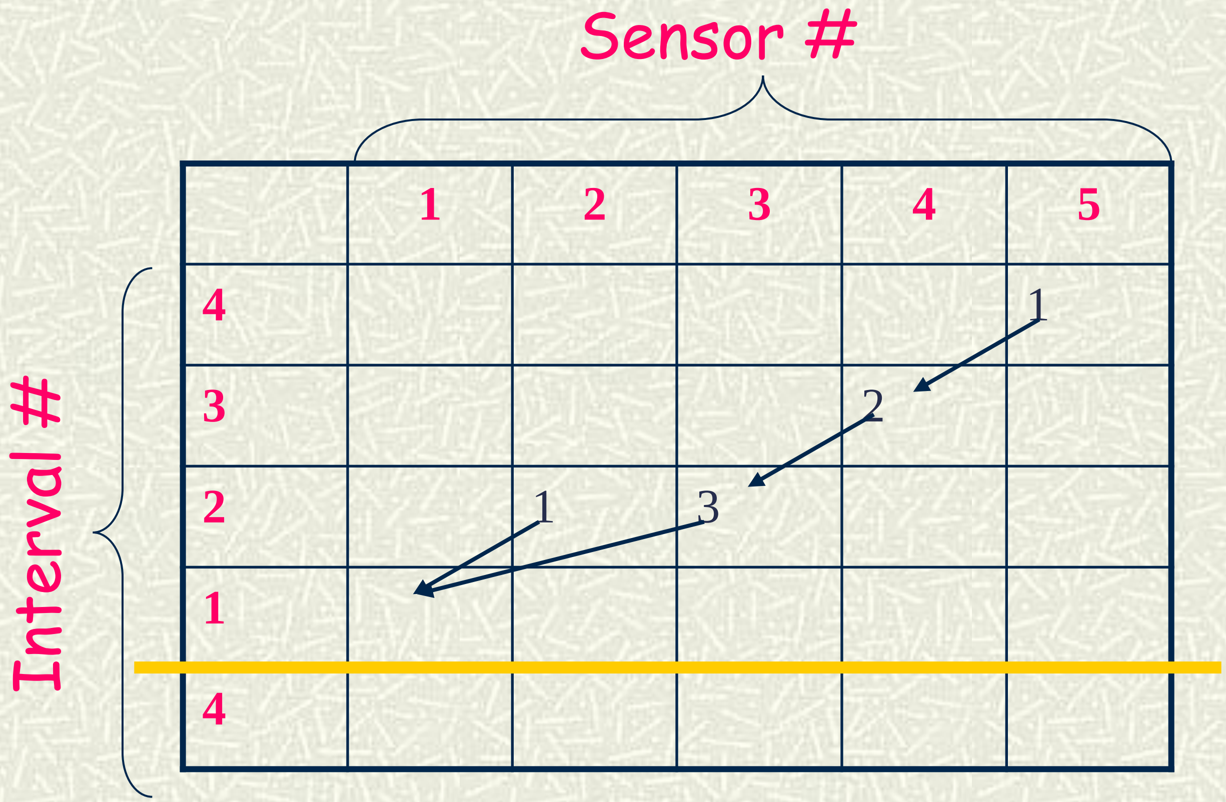


Illustration: Aggregation

SELECT COUNT(*) FROM sensors



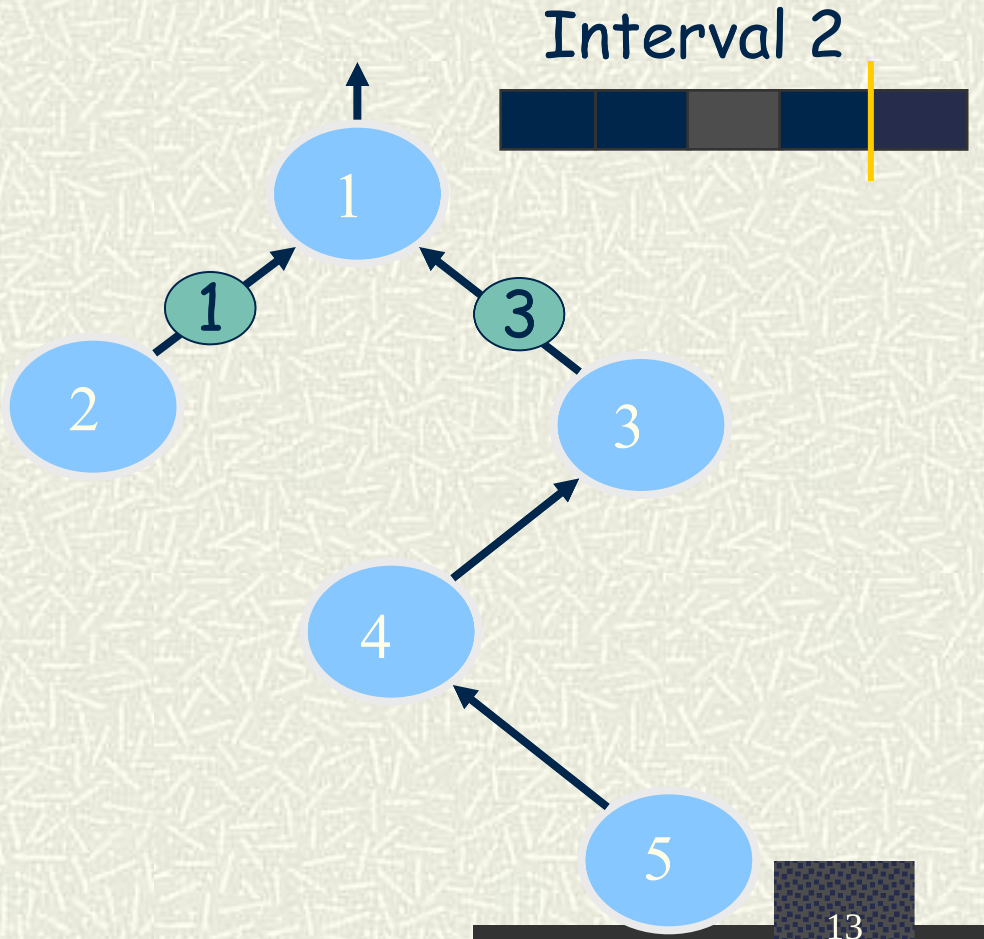


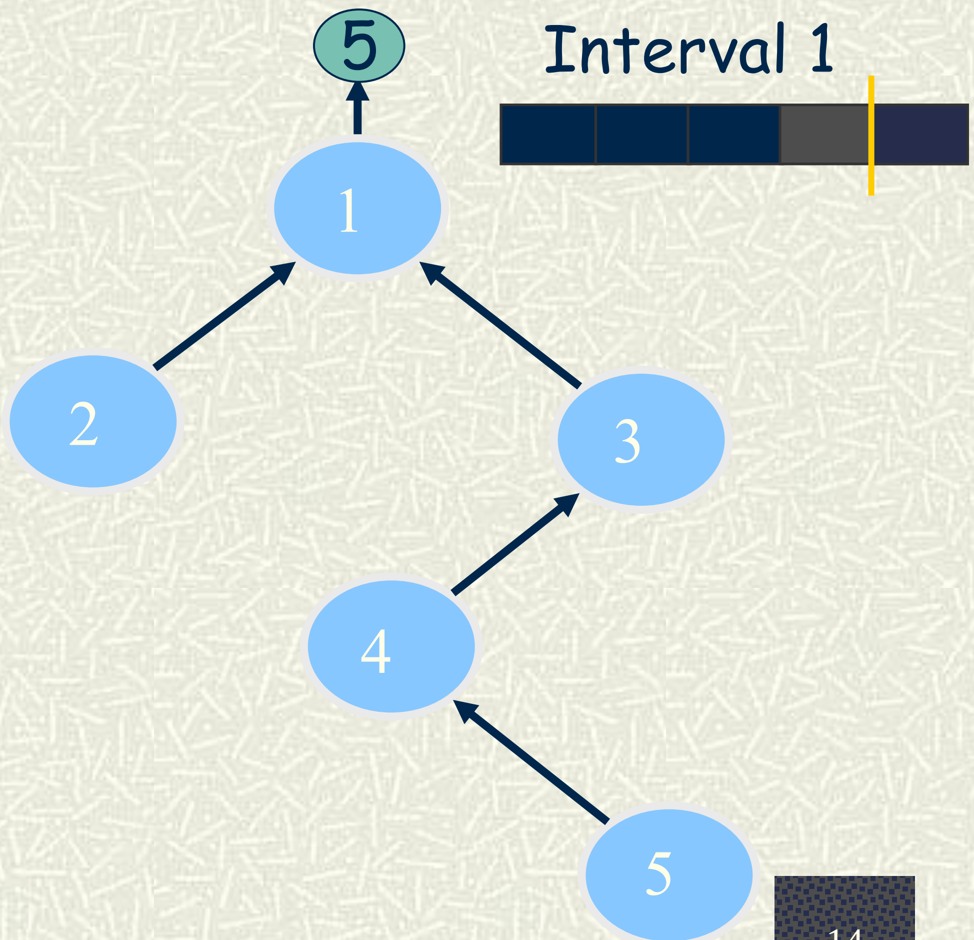
Illustration: Aggregation

SELECT COUNT(*) FROM sensors

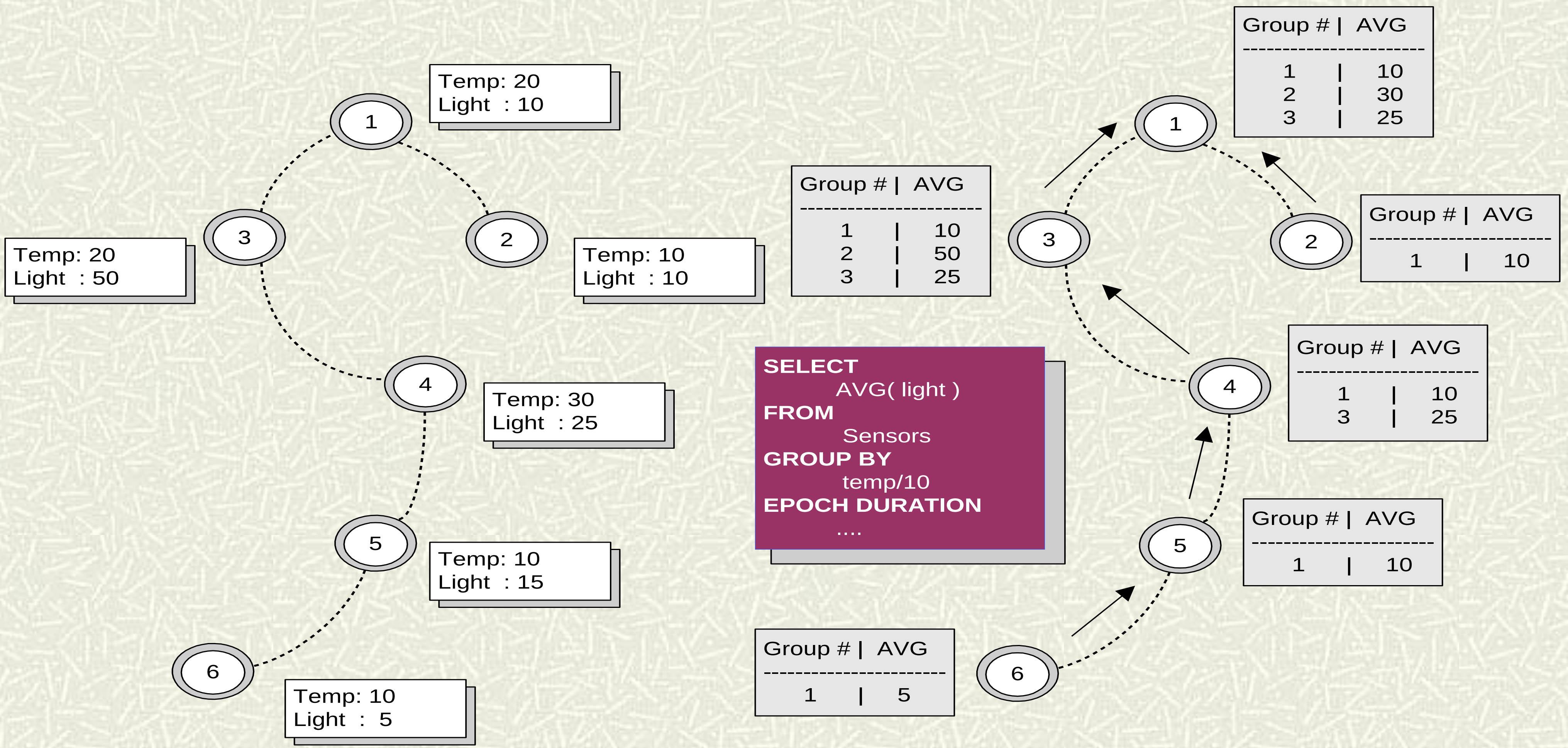
Sensor #

Interval #

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



TAG Algorithm w/ GROUP-ing



Sensor measurements within one epoch

Aggregation state progress during one epoch

Aggregation Framework

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

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

$$F_{init}\,\{a_0\} \rightarrow \langle a_0 \rangle$$

$$F_{merge}\,\{\langle a_1\rangle,\langle a_2\rangle\} \rightarrow \langle a_{12}\rangle$$

$$F_{evaluate}\,\{\langle a_1\rangle\} \rightarrow \text{aggregate value}$$

Partial State Record (PSR)

Example: Average

$$AVG_{init}\ \{v\} \rightarrow \langle v,1\rangle$$

$$AVG_{merge}\ \{\langle S_1, C_1\rangle, \langle S_2, C_2\rangle\} \rightarrow \langle S_1 + S_2 , C_1 + C_2\rangle$$

$$AVG_{evaluate}\{\langle S, C\rangle\} \rightarrow S/C$$

Restriction: Merge associative, commutative

NORTHWESTERN UNIVERSITY

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.

NORTHWESTERN UNIVERSITY

Classes of aggregations

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

NORTHWESTERN UNIVERSITY

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.

NORTHWESTERN UNIVERSITY

Classes of aggregations

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

Good → (Distributive)

worst → (Holistic)

bad → (Unique)

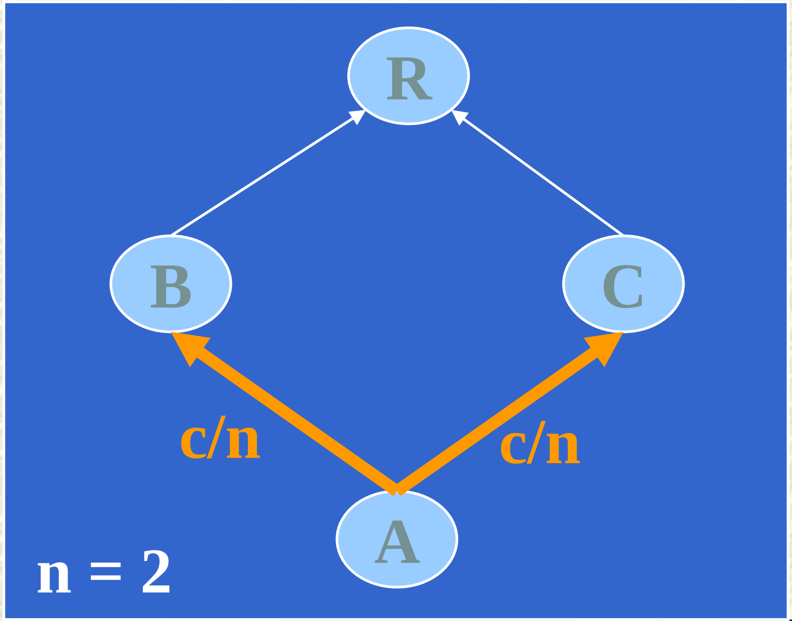
Classes of aggregates

	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

Use 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

SELECT COUNT(*)



P(link xmit successful) = p

P(success from A->R) = p^2

$E(cnt) = c * p^2$

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

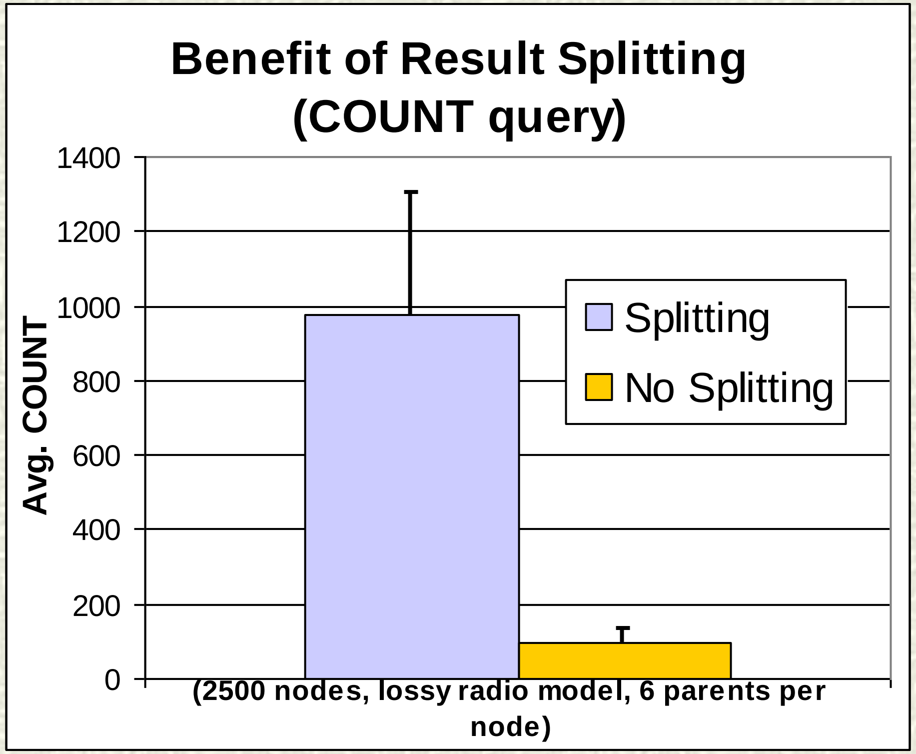
# of parents = n

$E(cnt) = n * (c/n * p^2)$

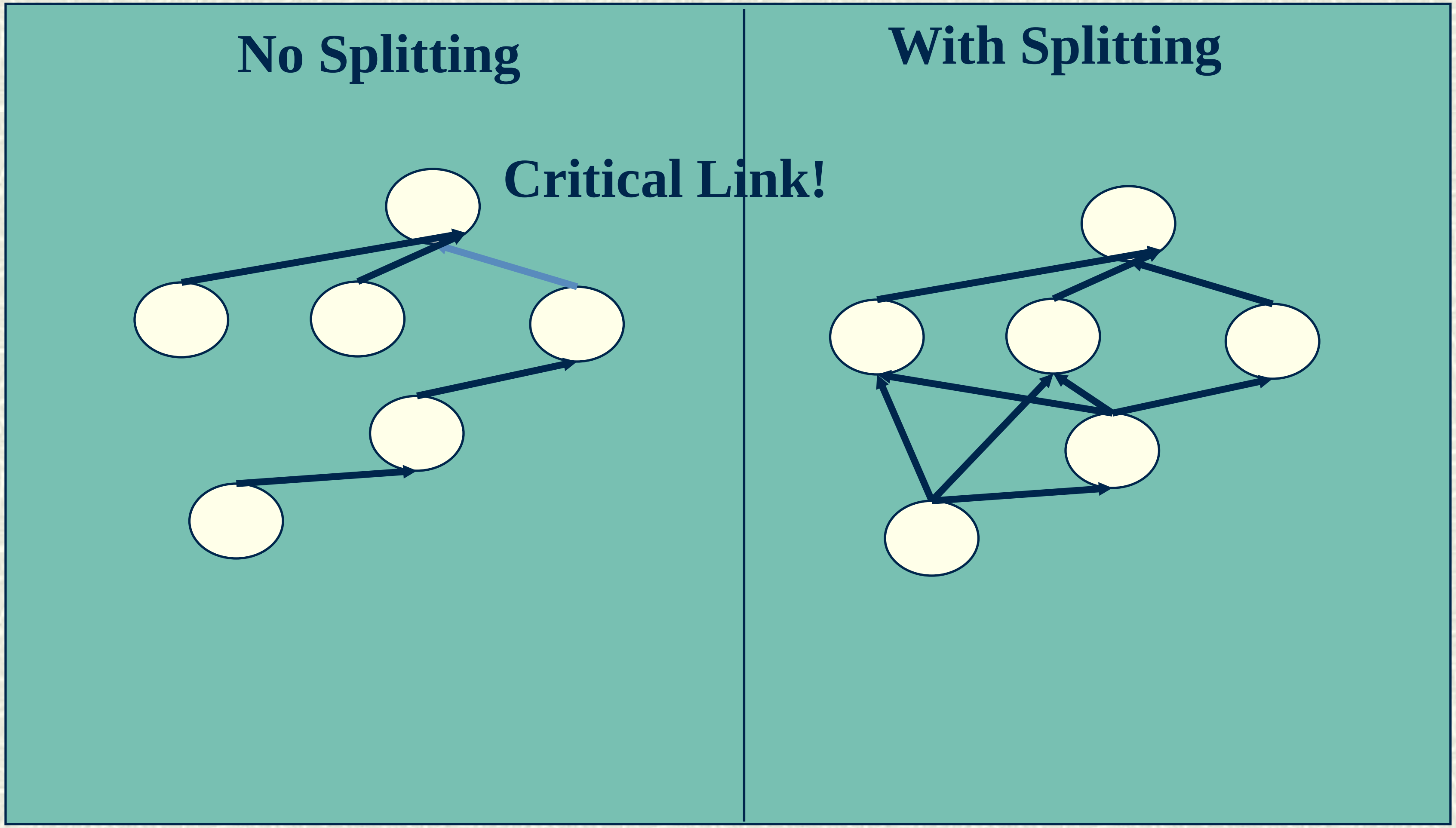
$Var(cnt) = n * (c/n)^2 * p^2 * (1 - p^2) = \underline{V/n}$

Multiple Parents Results

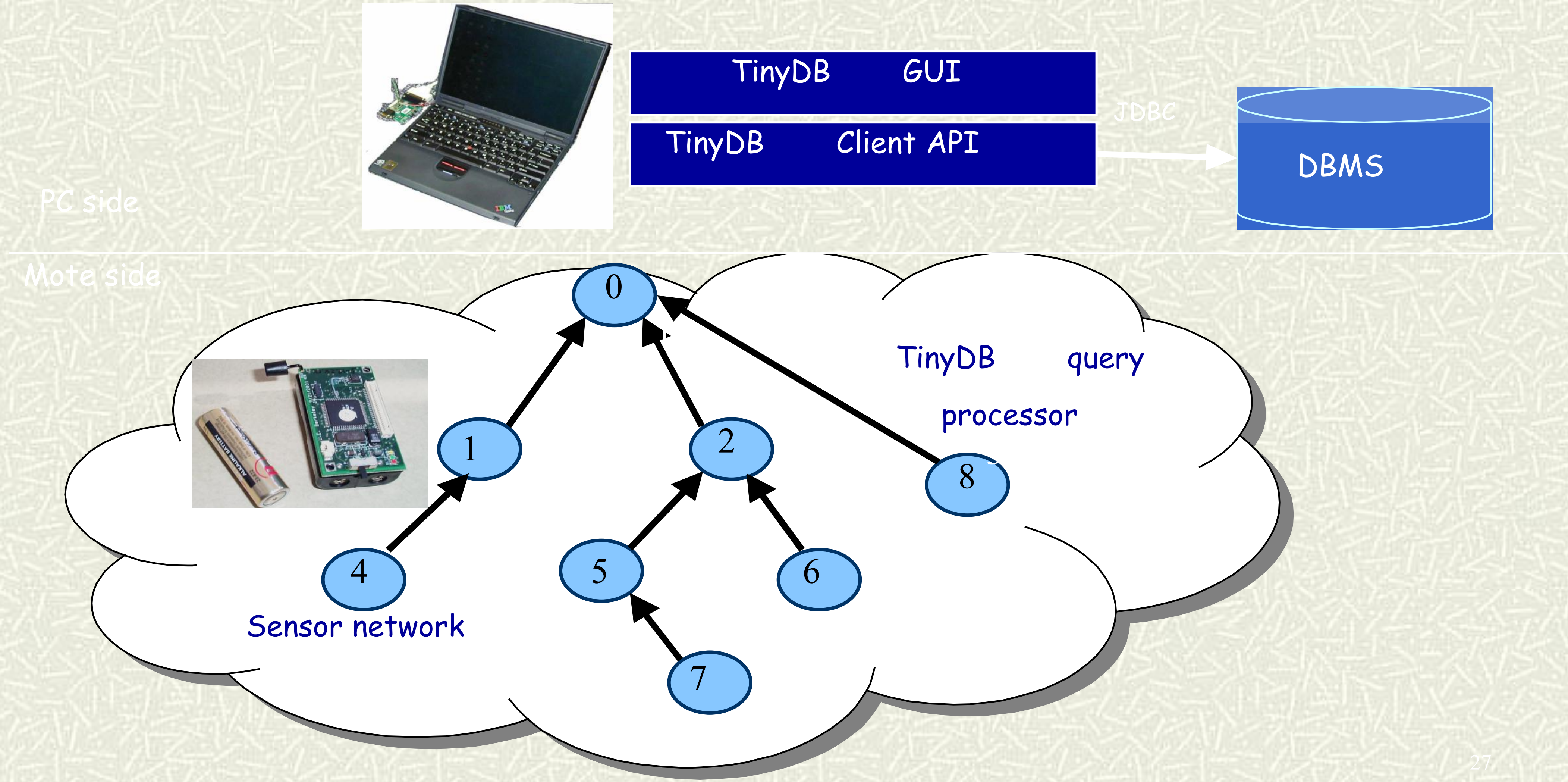
- Better than previous analysis expected!
- Losses aren't independent!
- Insight: spreads data over many links



Multiple Parents Results



TinyDB Architecture



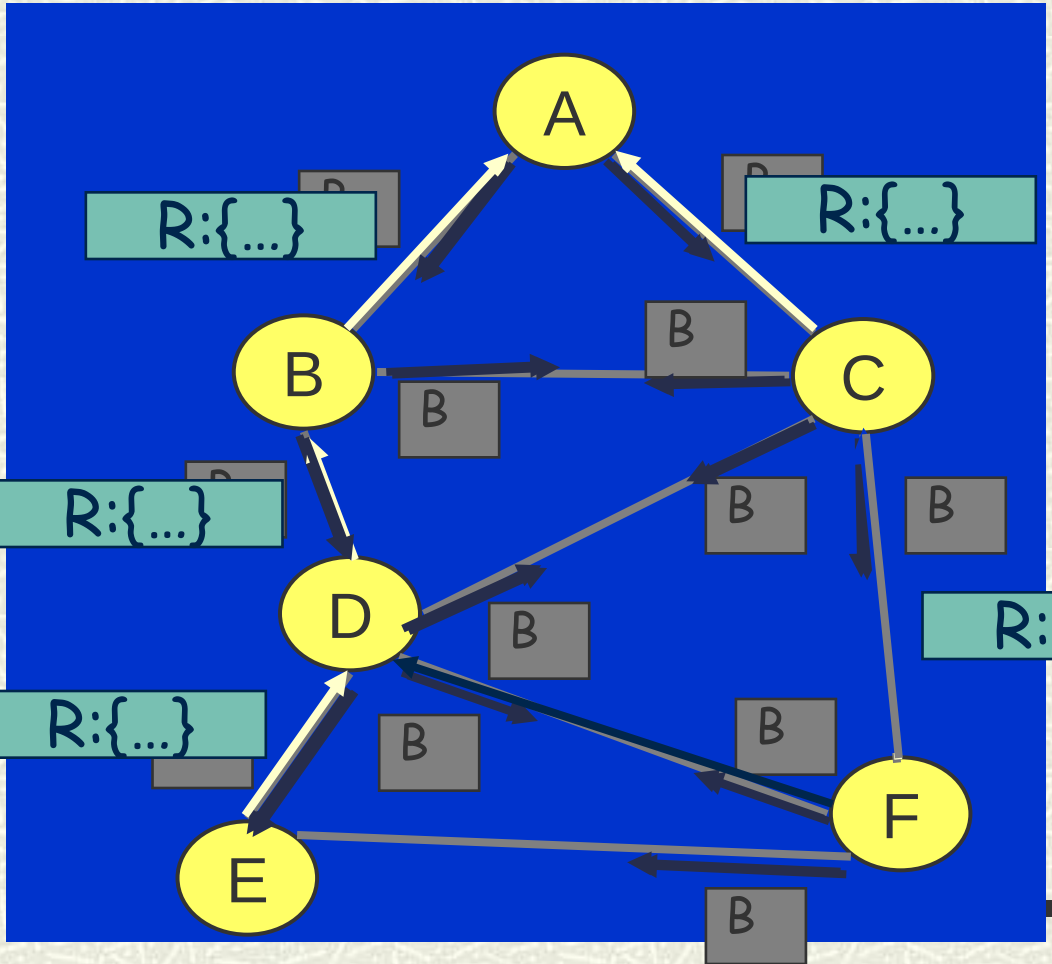
Multihop Networking

- Revised implementation of "tree based routing"

Parent Selection:
Use parent with best Quality link

Node D	
Neigh	Qual
B	.75
C	.66
E	.45
F	.82

Node C	
Neigh	Qual
A	.5
B	.44
D	.53
F	.35



NORTHWESTERN UNIVERSITY

Data model—revisited

- A single, append-only table
  Sensors (*nodeid*, *time*, *light*, *temp*, …)
- Just a conceptual view for posing queries; in reality:
  - Data is not already there at query time
    - Traditional database: queries independent of acquisition
    - Here: queries drive acquisition
      - Didn't ask for light? Then it won't be sampled!
  - Data may not be at one place
    - Like a distributed database, but here nodes/network are much less powerful/reliable
  - Data won't be around forever
    - Similar to stream data processing

Acquisitional Query Processing

- What's really new & different about databases on (mote-based) sensor networks?
- TinyDB's answer:
  - Long running queries on physically embedded devices that control when and where and with what frequency data is collected
  - Versus traditional DBMS where data is provided *a priori*
- For a distributed, embedded sensing environment, ACQP provides a framework for addressing issues of
  - When, where, and how often data is sensed/sampled
  - Which data is delivered

NORTHWESTERN UNIVERSITY

Acquisitional Query Processing

- How does the user control acquisition?
  - Rates or lifetimes
  - Event-based triggers
- How should the query be processed?
  - Sampling as an operator, Power-optimal ordering
  - Frequent events as joins
- Which nodes have relevant data?
  - Semantic Routing Tree for effective pruning
    - Nodes that are queried together route together
- Which samples should be transmitted?
  - Pick most “valuable”?
  - Adaptive transmission & sampling rates

Rate & Lifetime Queries

- Rate query

  SELECT nodeid, light, temp

  FROM sensors

  SAMPLE INTERVAL 1s FOR 10s

- Lifetime query

  SELECT …

  LIFETIME 30 days

A

Estimate sampling rate that achieves this

SELECT …

LIFETIME 10 days

MIN SAMPLE INTERVAL 1s

May not be able to transmit all the data

NORTHWESTERN UNIVERSITY

Processing Lifetimes: Issues

- Provide formulas for estimating power consumption: set maximum per-node sampling rates
- What makes this difficult?
  - estimating the selectivity of predicates
  - amount transmitted by a node varies widely
  - root is a bottleneck: all nodes rates must correspond to it
  - aggregation vs. sending individual values
  - multiple sensing types (temp, accel) with different drain
  - conditions change: multiple queries, burstiness, message losses
- What to do when can't transmit all the data

NORTHWESTERN UNIVERSITY

Storage points

- `CREATE STORAGE POINT recentLight SIZE 8 AS (SELECT nodeid, light FROM Sensors SAMPLE PERIOD 10s);`
  - A sliding window of recent readings, materialized locally

B

- Joining with the *Sensors* stream
  - ```
    SELECT COUNT(*)
    FROM Sensors s, recentLight rl
    WHERE rl.nodeid = s.nodeid AND s.light <
    rl.light
    SAMPLE PERIOD 10s;
    ```

C

☞ *TinyDB only allows joining a stream with a storage point !*
    ```

NORTHWESTERN UNIVERSITY

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

NORTHWESTERN UNIVERSITY

Event Based Processing

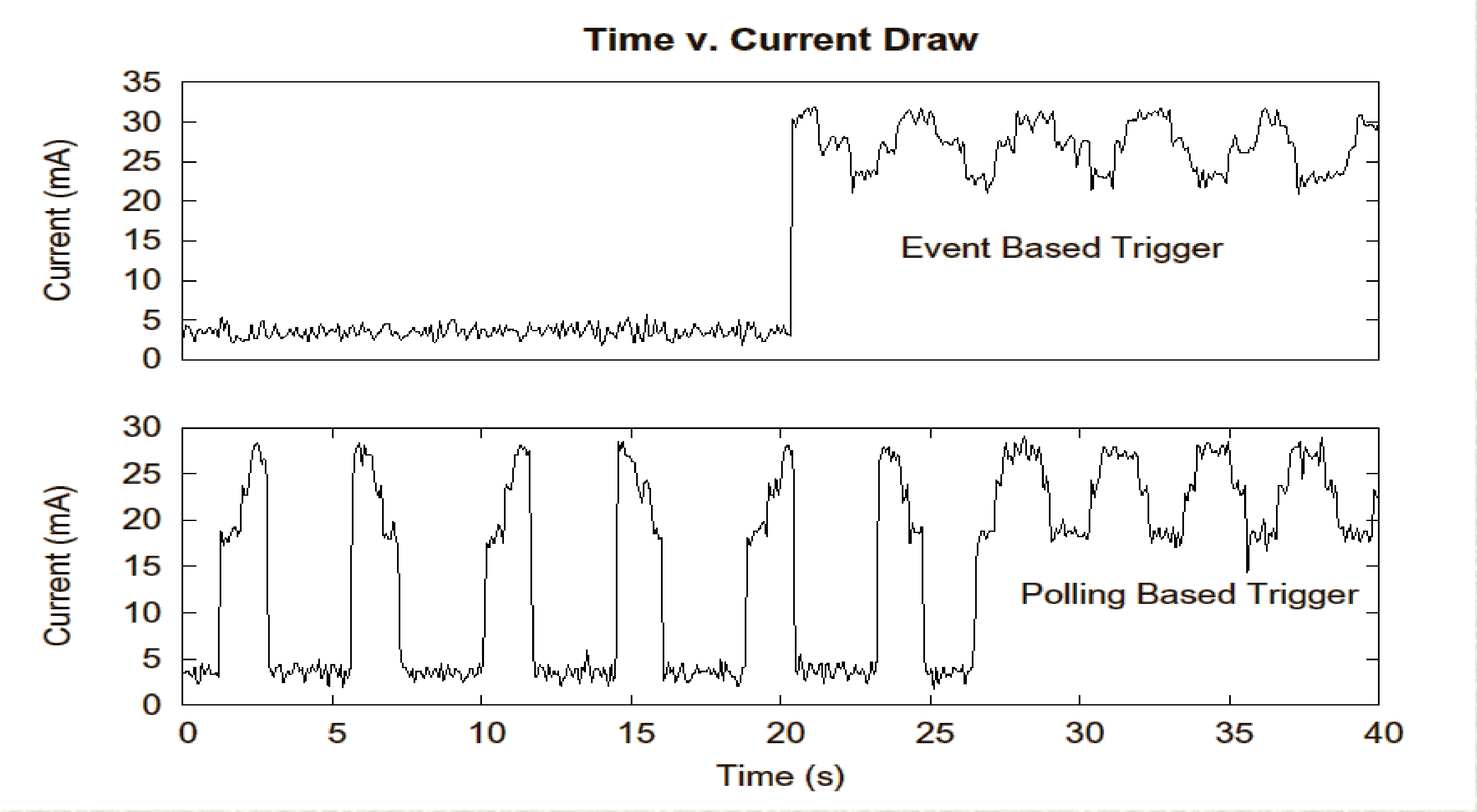
- ACQP – want to initiate queries in response to events

```
ON EVENT bird-detect(loc):
    SELECT AVG(s.light), AVG(s.temp), event.loc
    FROM sensors AS s
    WHERE dist(s.loc, event.loc) < 10m
    SAMPLE PERIOD 2s FOR 30s
```

Reports the average light and temperature level at sensors near a bird nest where a bird has been detected

E.g., New query instance generated for as long as bird is there

Event Based Processing



Single external interrupt