NORTHWESTERN
UNIVERSITY

Querying the Sensor Network

TinyDb

Acquisitional Query Processing

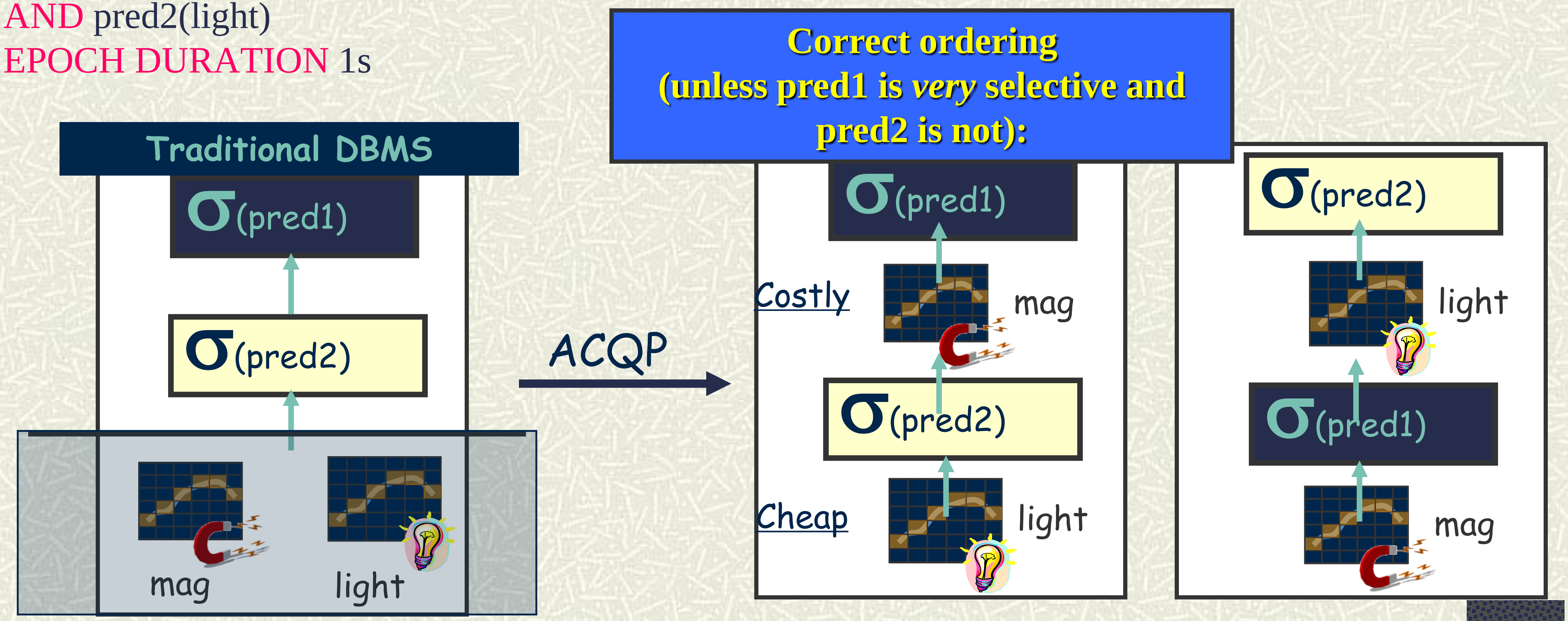
- How does the user control acquisition?
  - Rates or lifetimes
  - Event-based triggers
- How should the query be processed?
  - Sampling as an operator, Power-optimal ordering
  - Frequent events as joins
- Which nodes have relevant data?
  - Semantic Routing Tree for effective pruning
    - Nodes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

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!

- 1500 uJ vs. 90 uJ



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

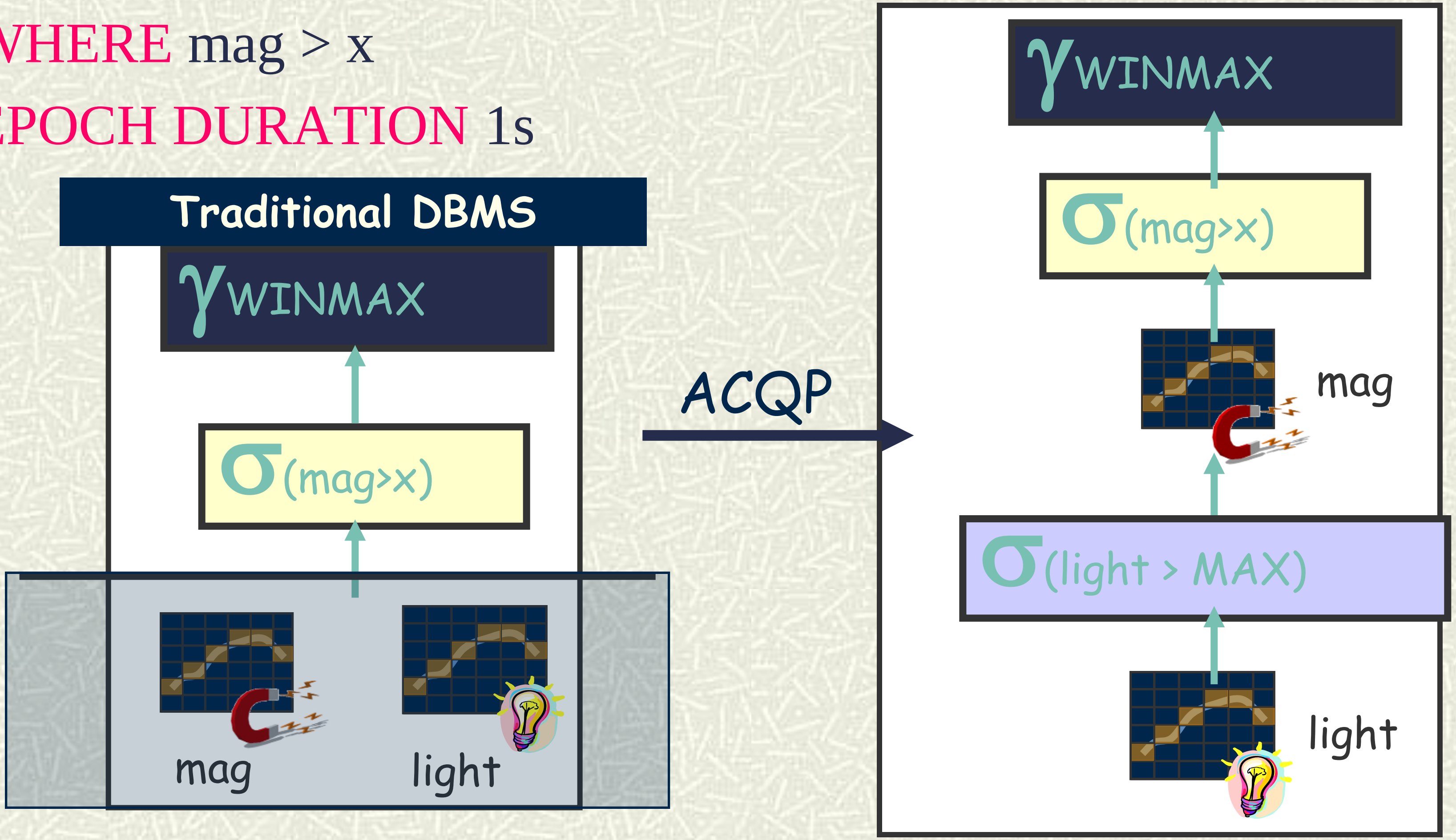
Exemplary Aggregate Pushdown

SELECT WINMAX(light,30s,8s)

FROM sensors

WHERE mag > x

EPOCH DURATION 1s



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

Event Query Batching

```
ON EVENT  E(nodeid)
SELECT a
FROM sensors AS s
WHERE s.nodeid = e.nodeid
SAMPLE INTERVAL d FOR k
```

Problem: Multiple outstanding queries (lots of samples)

Solution: Rewrite as a sliding window join between sensors and the last k seconds of detected events:

```
SELECT s.a
FROM   sensors AS s, events AS e
WHERE s.nodeid = e.nodeid
AND e.type = E AND s.time – e.time <= k AND s.time > e.time
SAMPLE INTERVAL d
```

If events are frequent, use join approach…

Timing issues

- When batching, what if instances of different queries start at different times?
- If we order sampling and predicates sequentially, we can no longer take readings synchronously
- When joining a storage point and a stream, what if their sampling points don’t align?

☞*Tension between continuous signals and discrete events*

Acquisitional Query Processing

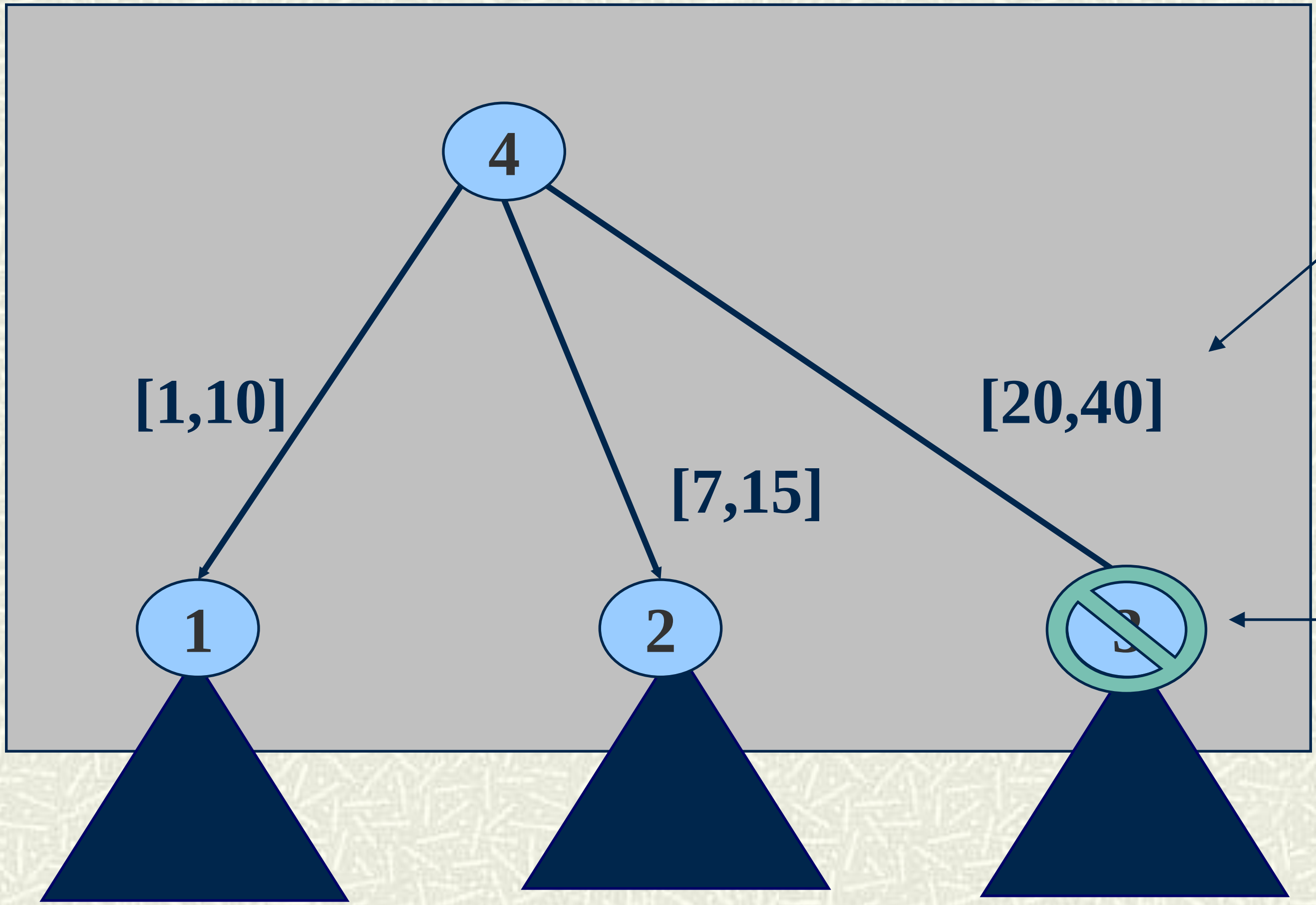
- How does the user control acquisition?
  - Rates or lifetimes
  - Event-based triggers
- How should the query be processed?
  - Sampling as an operator, Power-optimal ordering
  - Frequent events as joins
- Which nodes have relevant data?
  - Semantic Routing Tree for effective pruning
    - Nodes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

Attribute Driven Topology Selection

- Observation: internal queries often over local area
  - Or some other subset of the network
    - E.g. regions with light value in [10,20]
- Idea: build topology for those queries based on values of range-selected attributes
  - For range queries
  - Relatively static trees
    - Maintenance Cost

Attribute Driven Query Propagation

```
SELECT …
WHERE a > 5 AND a < 12
```

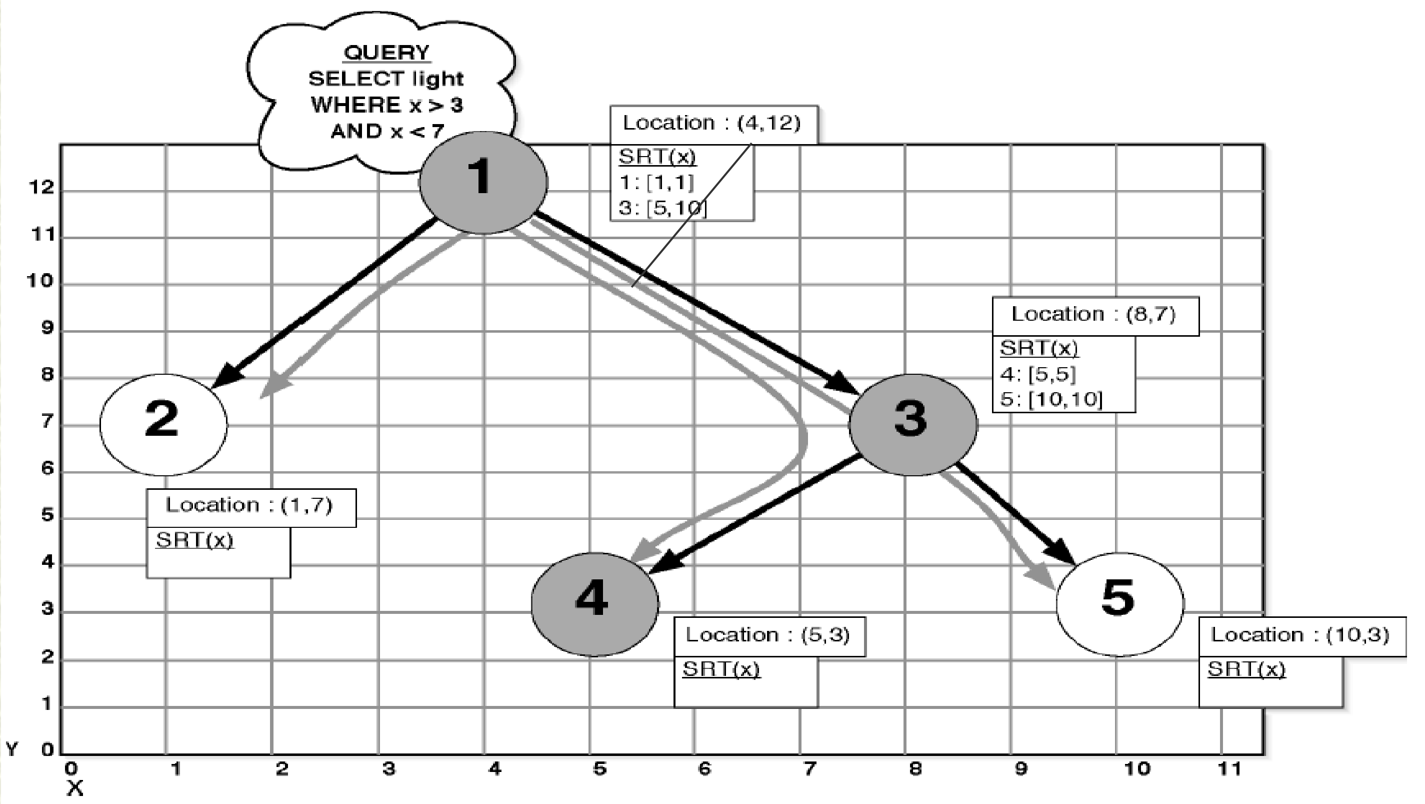


Precomputed intervals = Semantic Routing Tree (SRT)

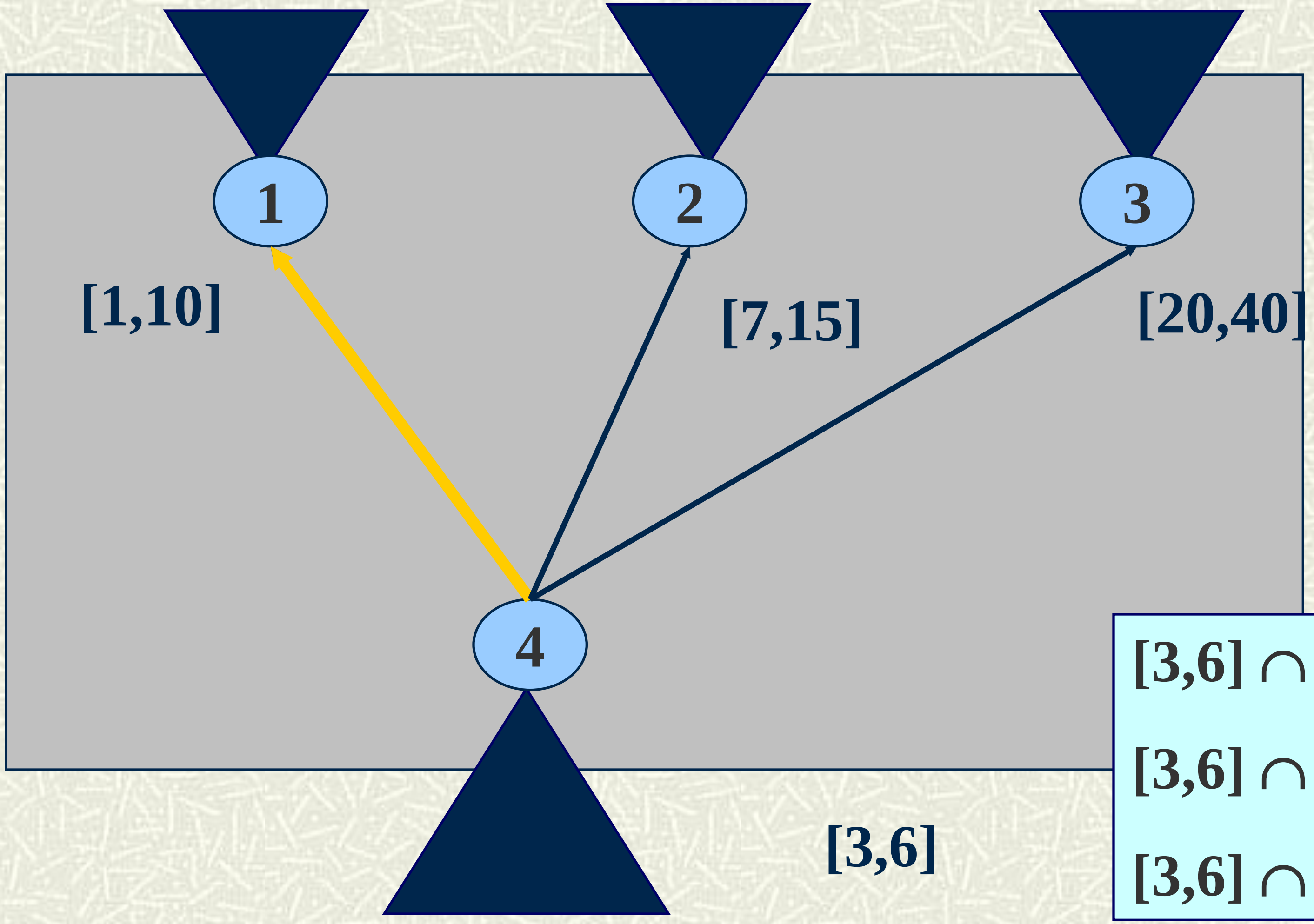
Early pruning

An “index”: semantic routing tree

- SELECT … FROM Sensors WHERE *A in range*...
  - Not sure which sensors have these *A* values?
  - Need to probe the entire network
- *Use an index*
  - Search tree = routing tree
  - Intermediate nodes store bounding boxes for subtrees
- *What’s different from DB search trees?*



Attribute Driven Parent Selection



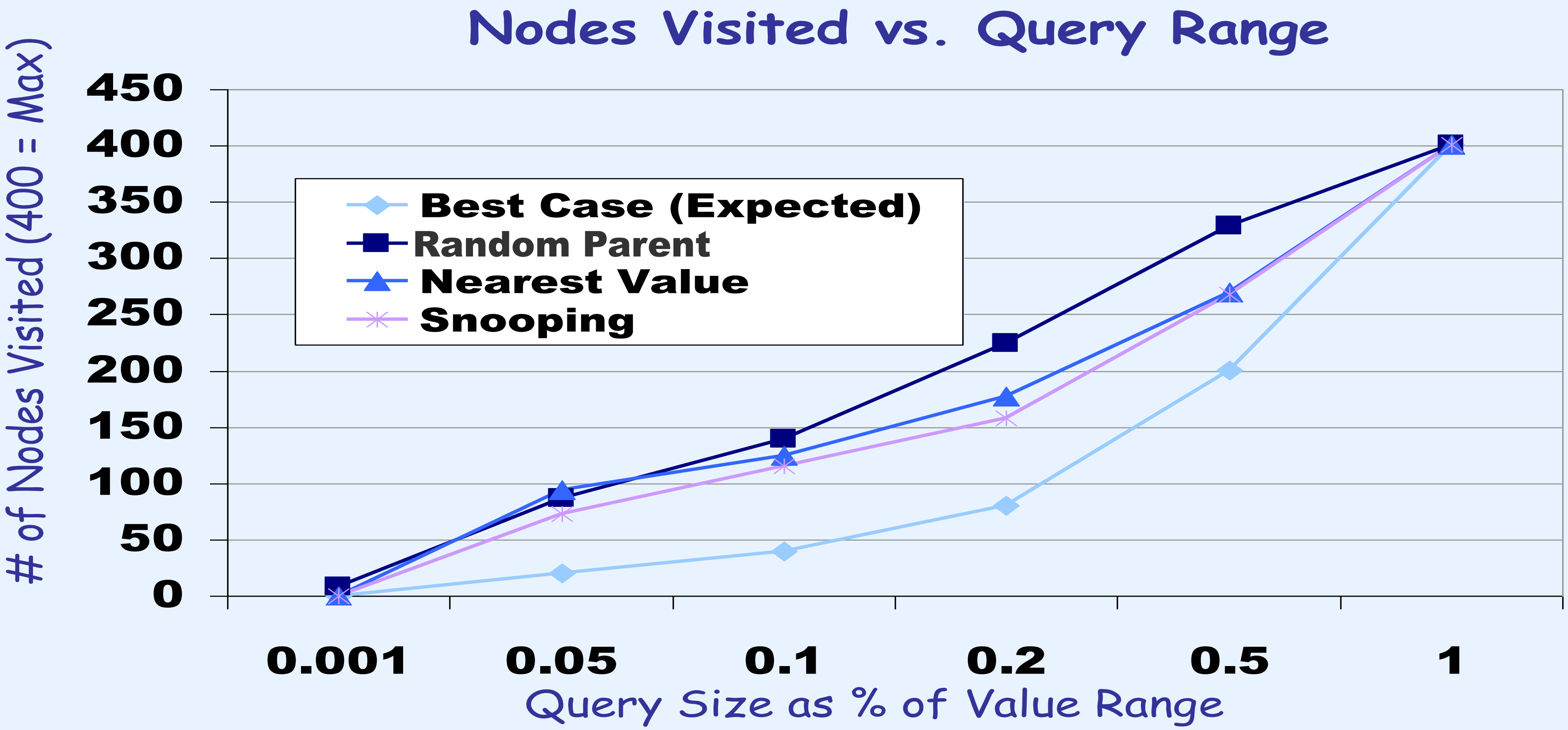
Even without known intervals, expect that choosing the parent with *closest* value will help

[3,6] ∩ [1,10] = [3,6] ★

[3,6] ∩ [7,15] = ø

[3,6] ∩ [20,40] = ø

Simulation Result

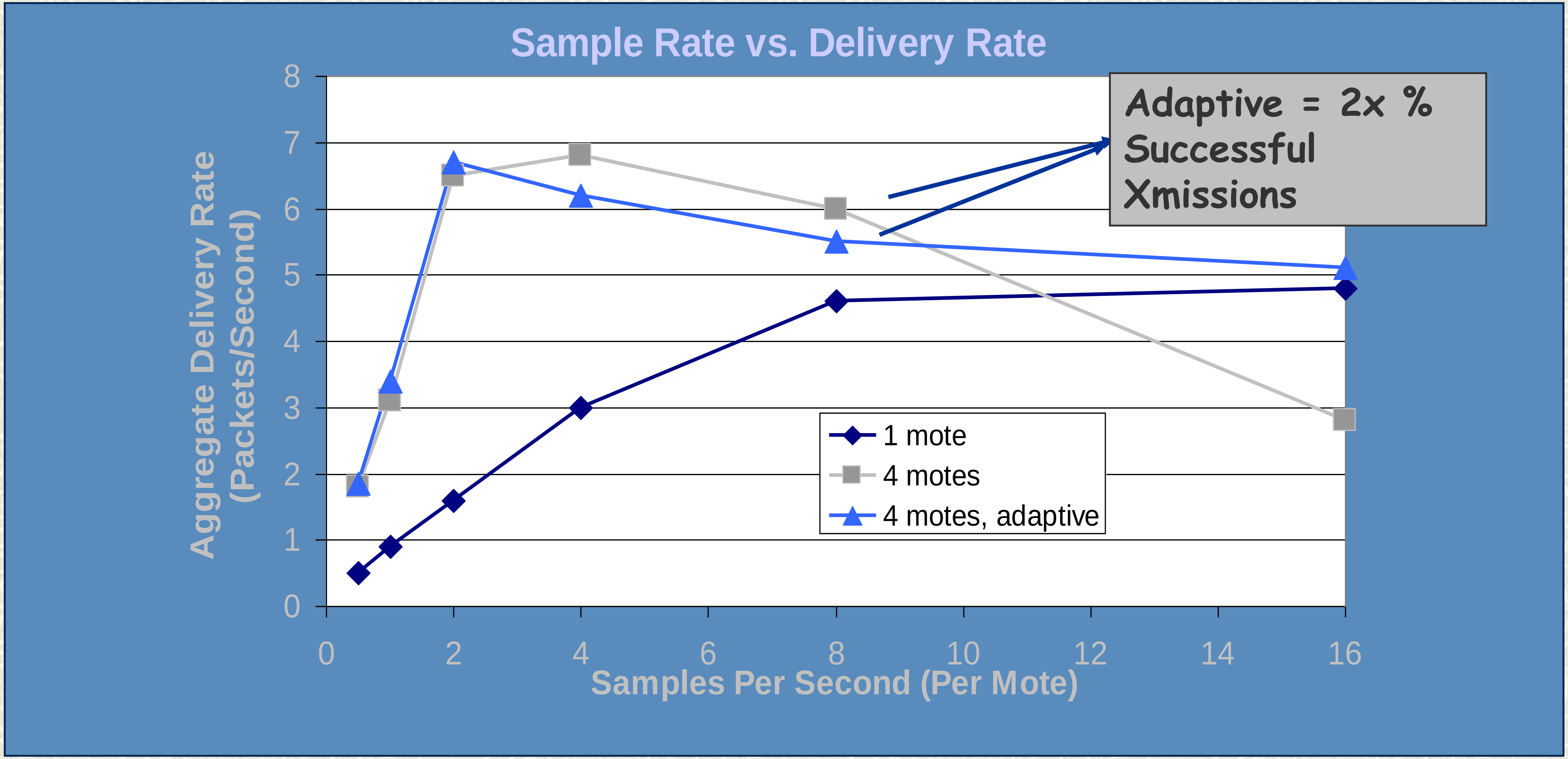


(Random value distribution, 20x20 grid, ideal connectivity to (8) neighbors)

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

Adaptive Transmission Rates

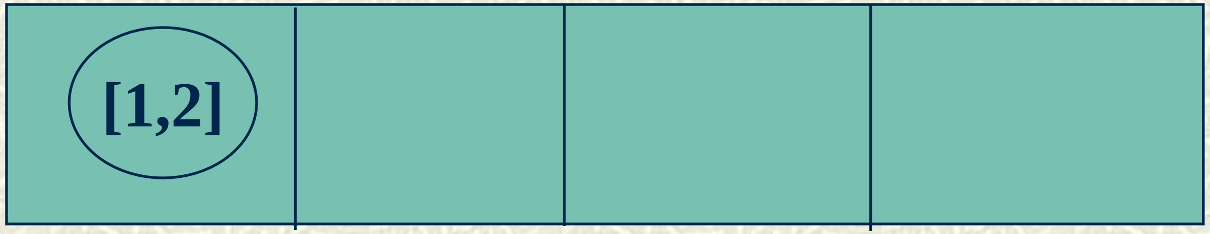


Adaptive = 2x % Successful Xmissions

TinyDB monitors channel contention & backs-off as needed

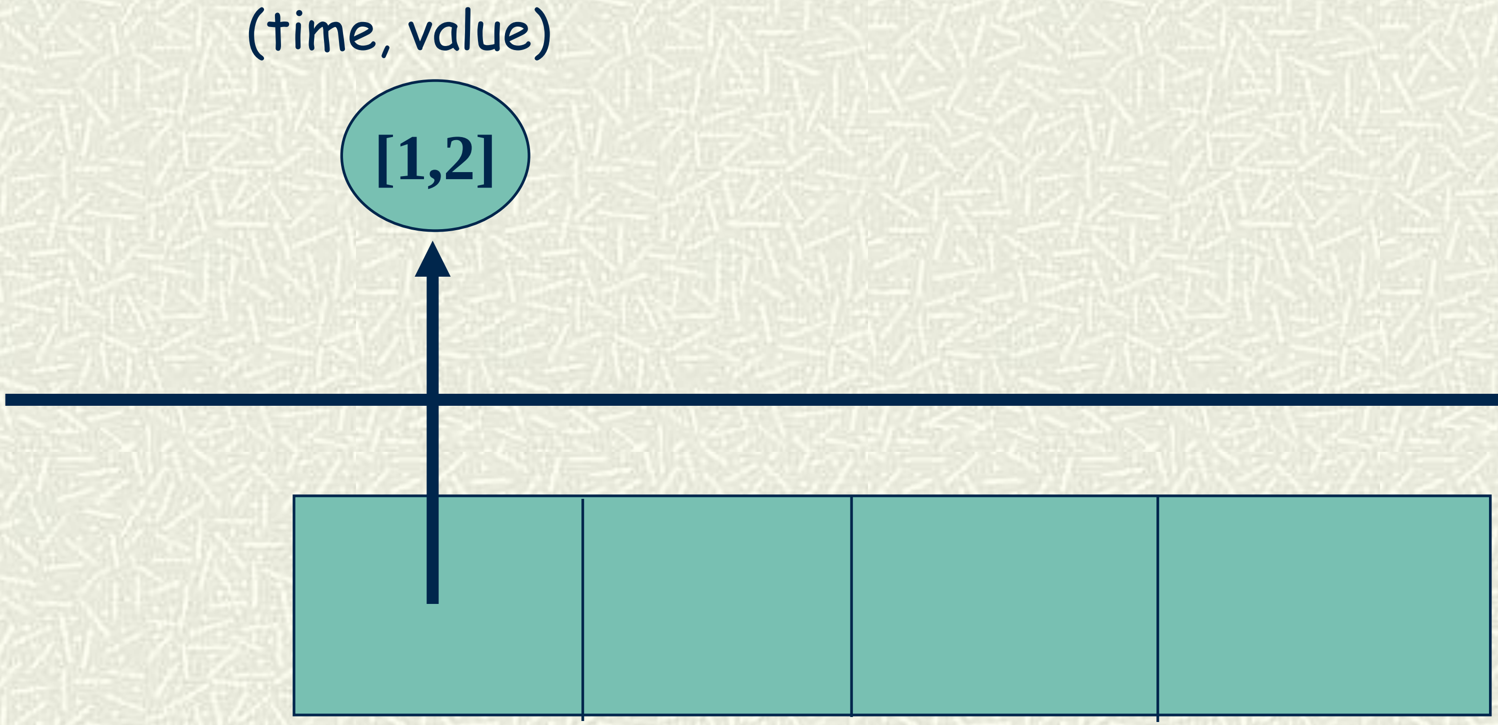
Prioritizing Data Delivery

- Score each item
- Send largest score
  - Out of order -> Priority Queue
- Discard or aggregate when buffer is full

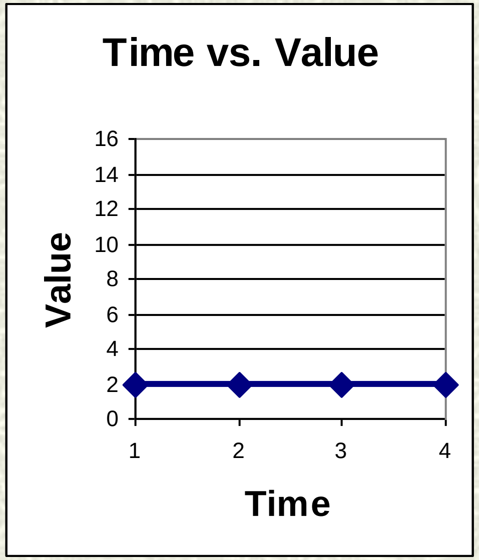


Choosing Data To Send

Delta encoding

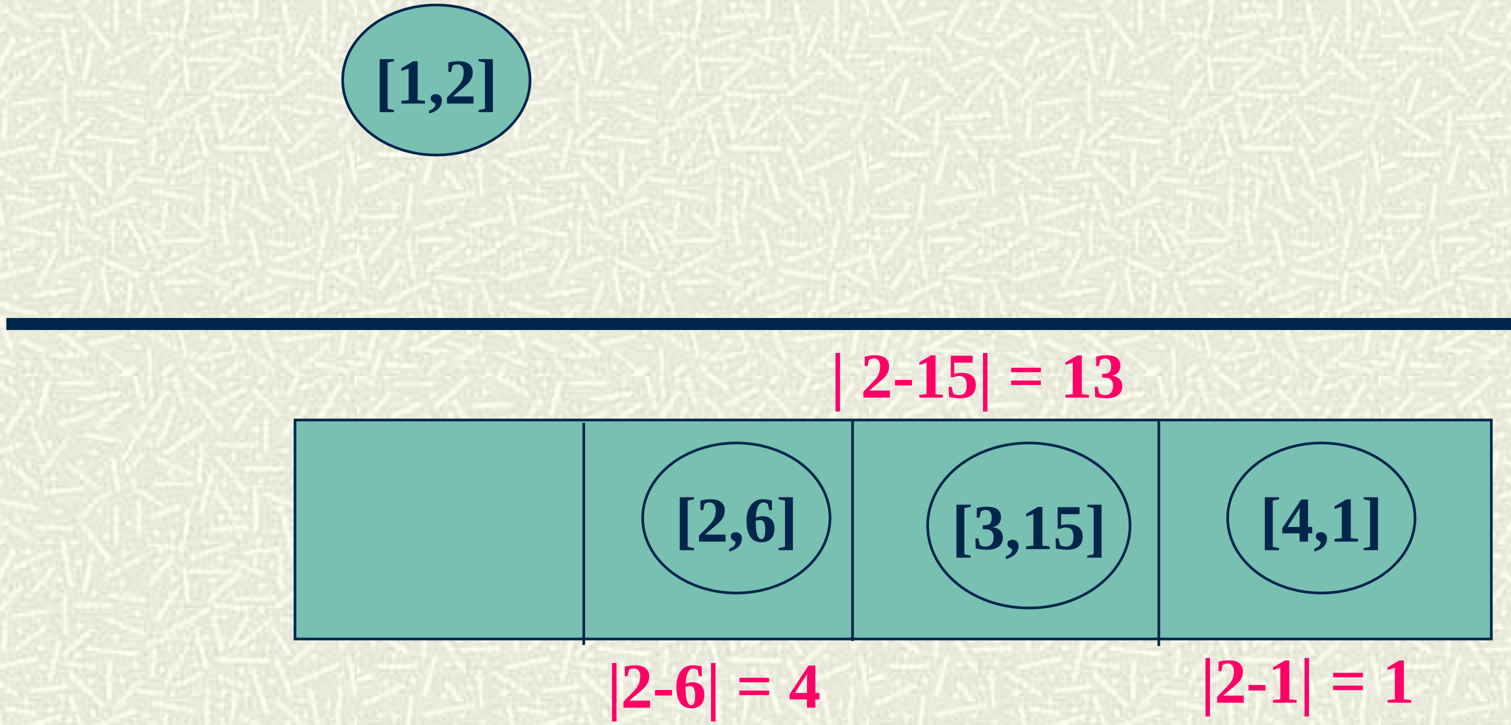

(time, value)

[1,2]



Choosing Data To Send

Delta encoding



[1,2]

| 2-15| = 13

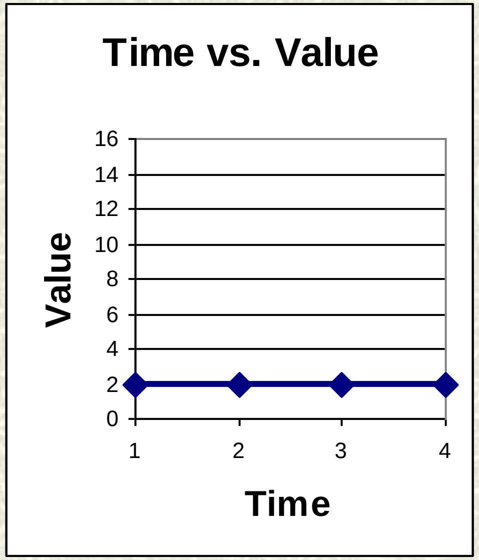
[2,6]

[3,15]

[4,1]

|2-6| = 4

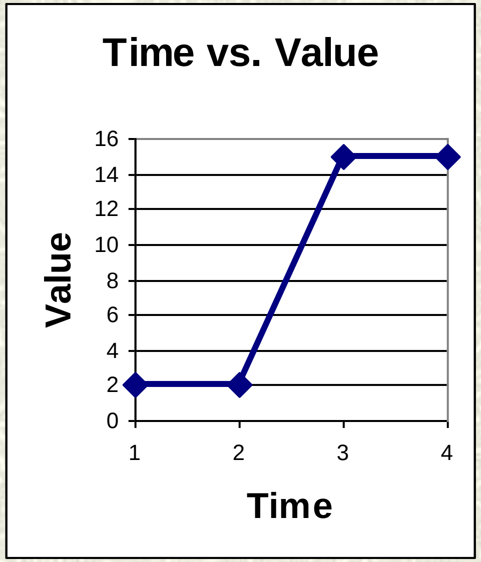
|2-1| = 1

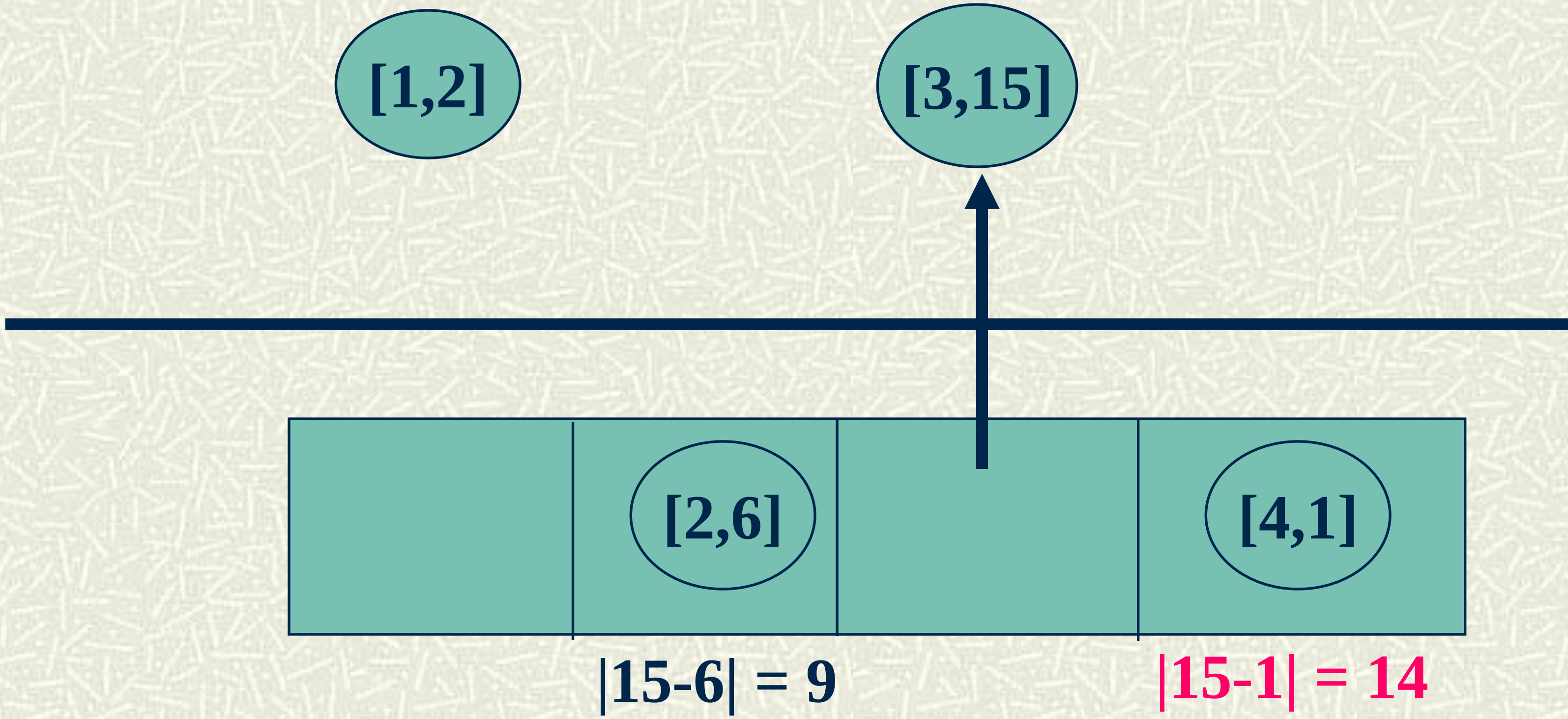


Select which of
the 3 to send

Choosing Data To Send

Delta encoding




[1,2]

[3,15]

[2,6]

[4,1]

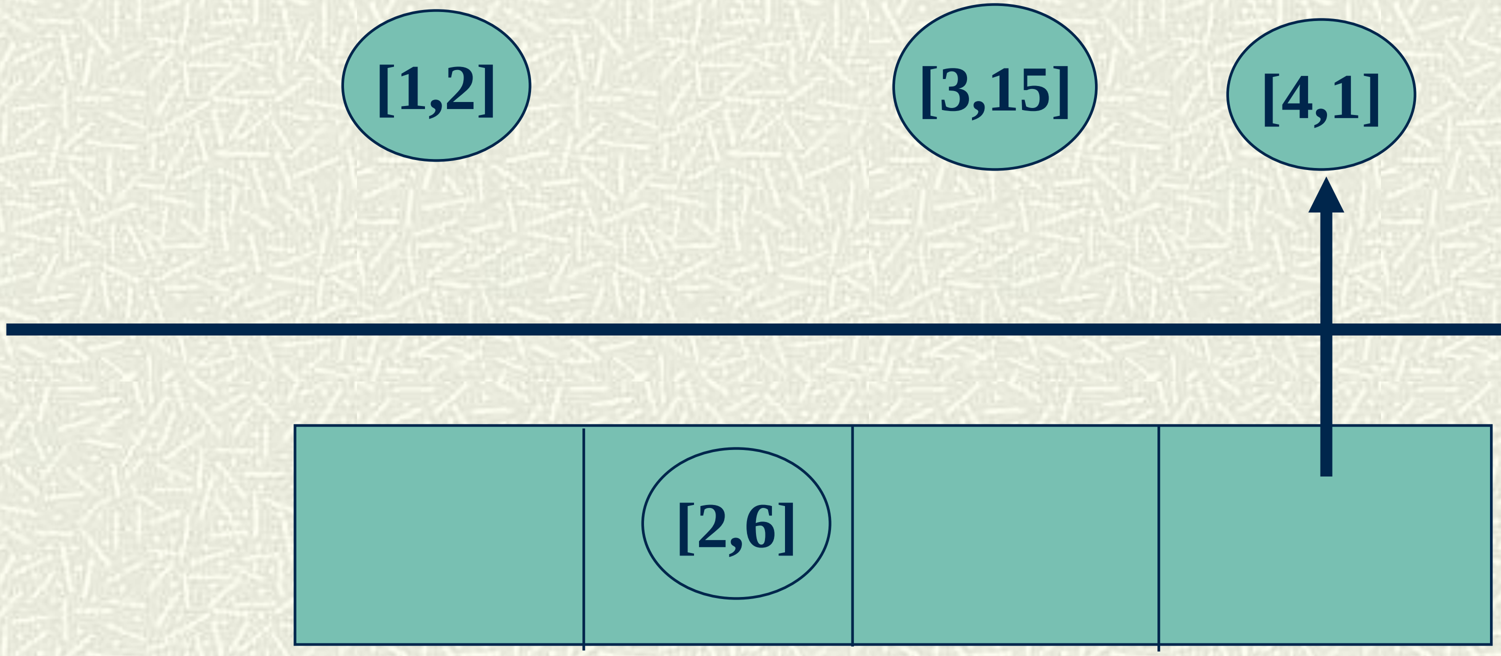
|15-6| = 9

|15-1| = 14

Keep selecting until hit max delivery rate

Choosing Data To Send

Delta encoding

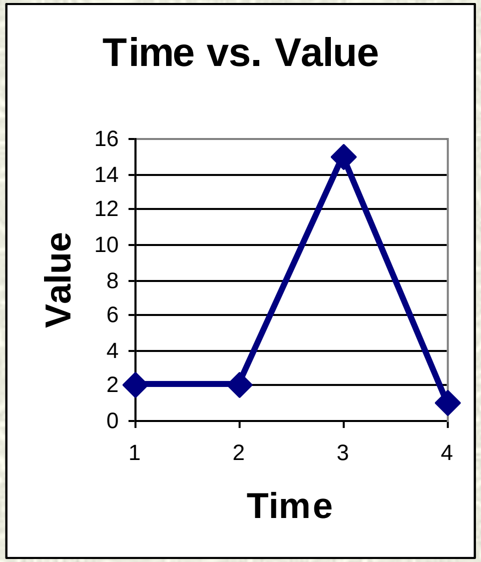


[1,2]

[3,15]

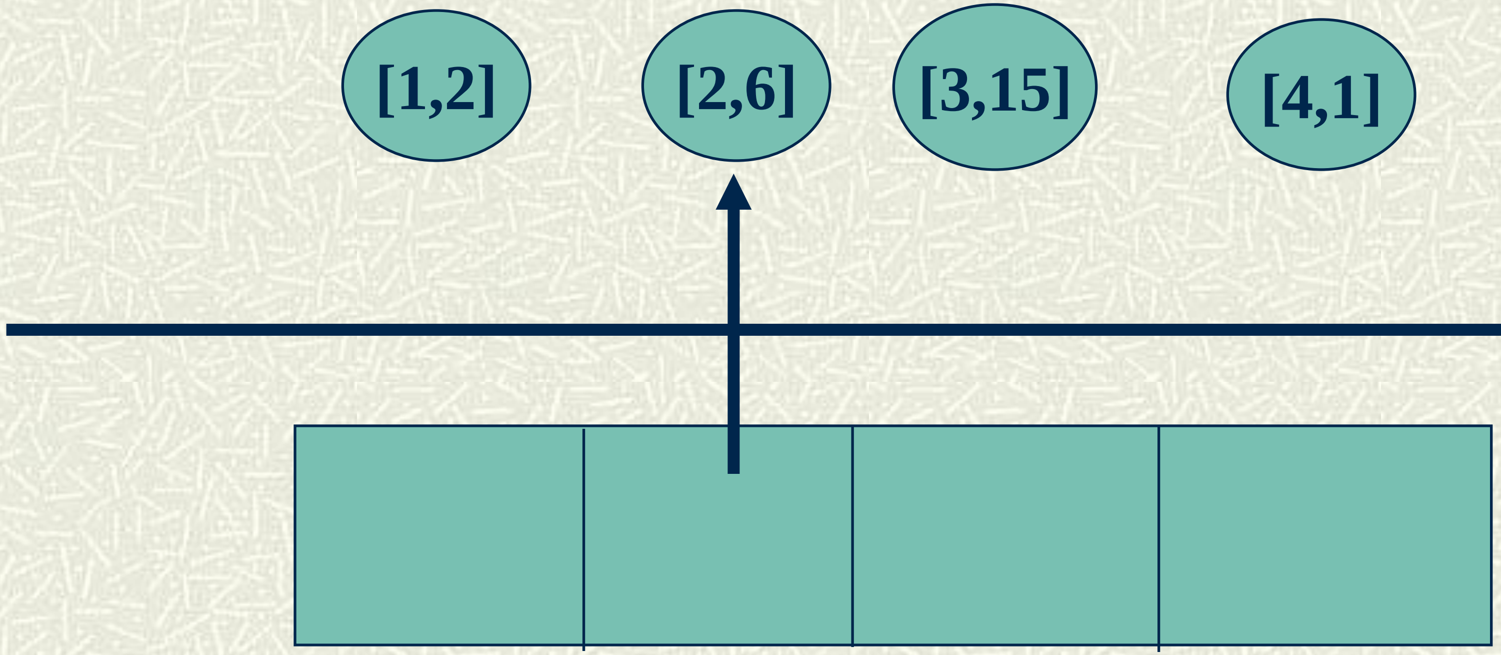
[4,1]

[2,6]



Choosing Data To Send

Delta encoding

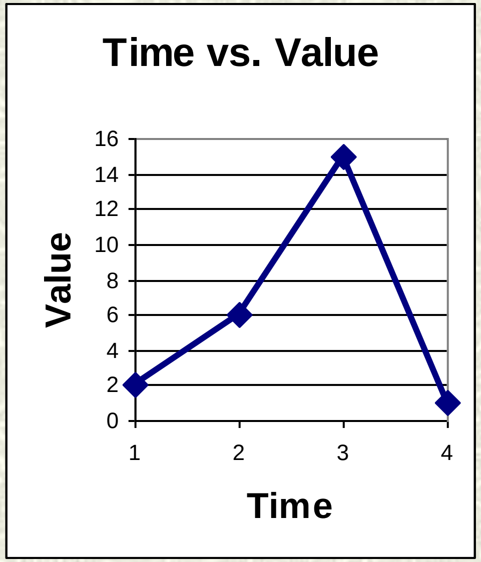


[1,2]

[2,6]

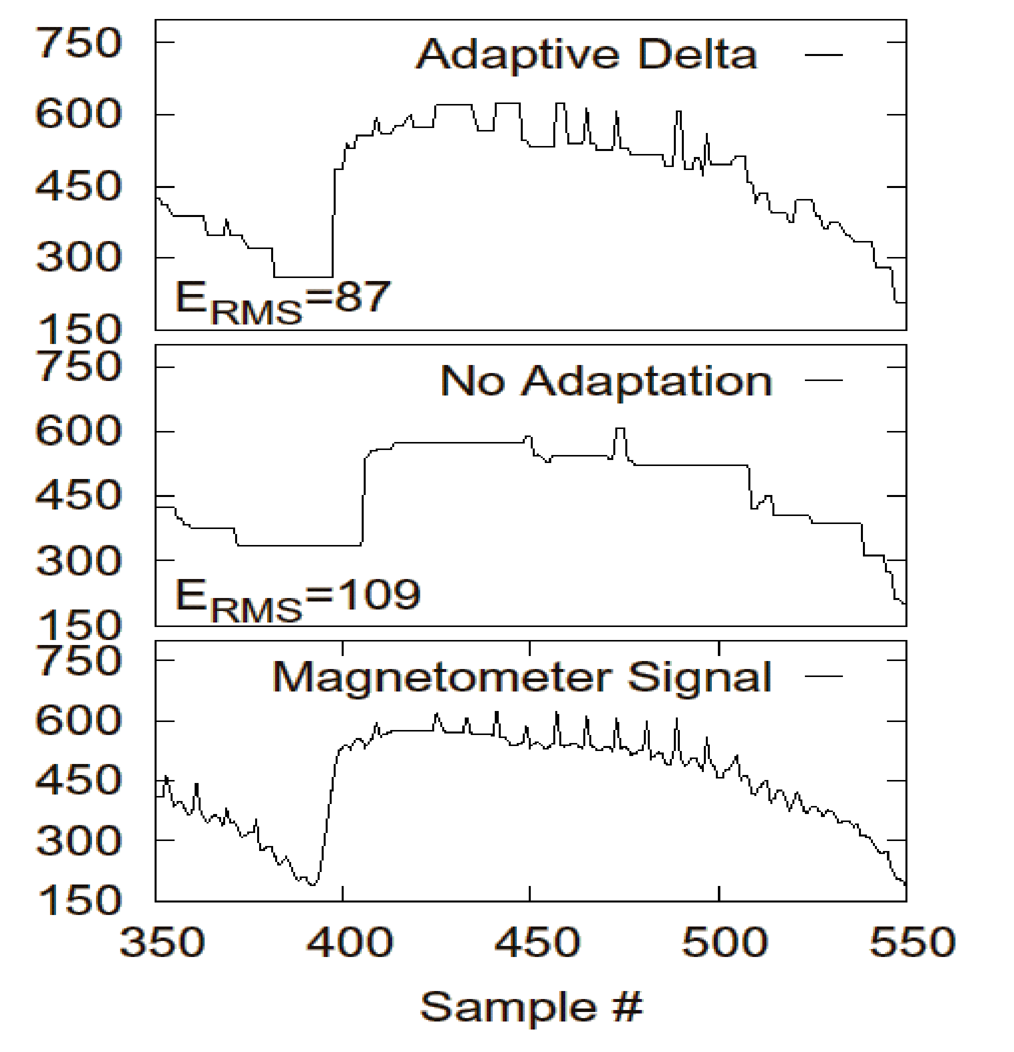
[3,15]

[4,1]



If manage
to send all

Delta + Adaptivity



- 8 element queue
- 4 motes transmitting different signals
- 8 samples /sec / mote