Data management in Wireless Sensor Networks (WSN)

Giuseppe Amato

ISTI-CNR

giuseppe.amato@isti.cnr.it

Outline

- Data management in WSN
- Query processing in WSN
- State of the art
- Future research directions

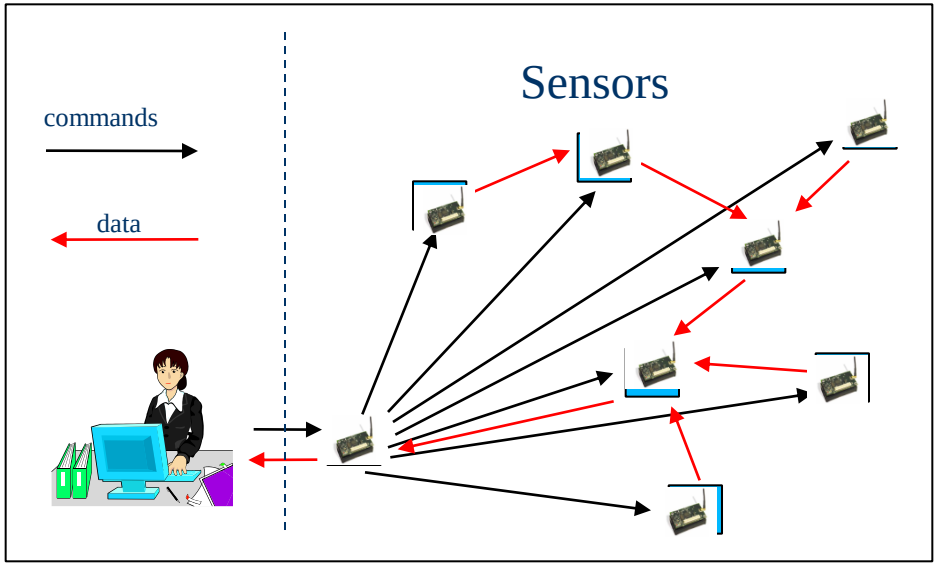
Outline

- Data management in WSN
  - ***WSN applications***
  - Data models
- Query processing in WSN
- State of the art
- Future research directions

Wireless Sensor Networks (WSN)

- A WSN is composed of a set of nodes that
  - Are small as a coin or a credit card
  - That communicate through wireless interfaces
  - Have a set of transducers to acquire environmental data
  - Have a microprocessor and a memory
  - Can run simple software programs
  - Are battery powered

Wireless Sensor Networks (WSN)

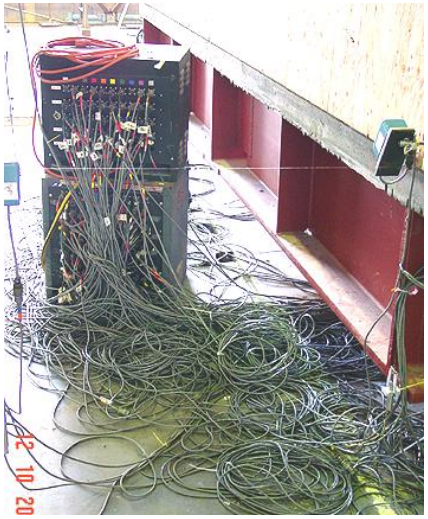


Sensor Network Applications

Habitat Monitoring: Storm petrels on Great Duck Island, microclimates on James Reserve.

Earthquake monitoring in shake-test sites.

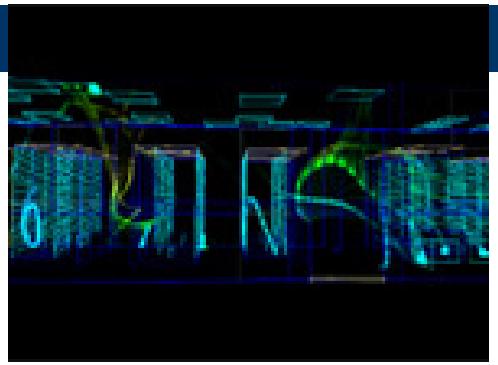
Vehicle detection: sensors along a road, collect data about passing vehicles.



Traditional monitoring apparatus.

Some Sensornet Apps

microclimate monitoring



smart cooling in data centers (hp/intel)

http://www.hpl.hp.com/research/dca/smart_cooling/

condition-based maintenance (intel/BP)

structural integrity (ucb/ggbd)

And More...

- Homeland security
  - Container monitoring
- Mobile environmental apps
  - Bird tracking
  - Zebranet
- Home automation
- Etc!

Peculiarities of WSN applications

- Several applications on WSN produce and process huge amount of data
  - Data are continuously produced
  - Data produced by different sensors might need to be compared/matched
  - Behaviour of sensors might need to be adjusted/refined over time
  - Environmental situation can change so new strategies might need to be used
  - Use of gathered data is not always known a priori

Declarative Queries

- Programming WSN Applications is Hard
  - Limited power budget
  - Lossy, low bandwidth communication
  - Require long-lived, zero admin deployments
  - Distributed Algorithms
  - Limited tools, debugging interfaces
- The database paradigm abstract away much of the complexity
  - Programming complexity is left to database developers
  - Users of the database get:
    - Safe, optimizable programs expressed in terms of queries
    - Freedom to think about apps instead of low-level programming details
    - Reprogramming the WSN remotely by sending new queries

Outline

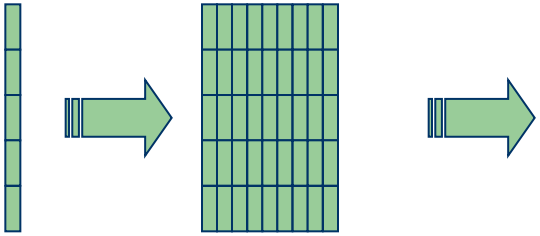
- Data management in WSN
  - WSN applications
  - ***Data models***
- Query processing in WSN
- State of the art
- Future research directions

Data model in wireless sensor networks

- Relational model is widely used in traditional databases
  - SQL databases are everywhere
- It can also be adopted in WSN databases
  - Output of sensors can be seen as infinitely-long logical tables
    - *data streams*
  - Columns consists of attributes defined in the network as
    - Sensor readings
    - Node_id, location, Time_stamps, …
    - User defined attributes
  - Use:
    - A data stream can be associated to every node (a group of transducers)
    - A data stream can be associated to every transducer
    - Some proposals consider just one single global data stream where all nodes put values in

Data Streams

- Each data stream consists of relational tuples
- The stream can be modeled as an append-only relation
- But repetitions are allowed and order is very important!



Data Streams - timestamps

- Data streams are (basically) ordered according to their timestamps
- Several constructs are based on timestamps:
  - temporal windows
  - unions
- Timestamps can be External and Explicit
  - Injected by data source
  - Models real-world event represented by tuples
  - Tuples may be out-of-order, but if near-ordered can reorder with small buffers
- Internal
  - Introduced as special field by Nodes
  - Arrival time in system
  - Can be explicit (I.e., seen by the queries) or implicit.

Query languages

- Declarative languages derived from SQL:

```
select nodeId, timestamp, temp, light
from sensors
where light > 10
Sampling interval 1s
```

- In this example:
  - One single global stream “sensors”
  - The query returns the nodes, timestamp and transducer readings where light is greater than 10

Outline

- Data management in WSN
- Query processing in WSN
  - ***From traditional databases to WSN databases***
  - Data Stream query processing issues
  - Query processing/optimisation issues in WSN
- State of the art
- Future research directions

Query processing

- High level query languages are translated in lower level formalisms
  - Relational algebra is the most used formalism
    - Its abstraction level is a good compromise between low level data access and expressiveness
    - It can be used/extended to support query processing in wireless sensor networks

A relation in Wireless Sensor Networks (stream)

Timestamp	NodeID	Light	Temp	Humidity
13	2	24	22	70
13	3	25	22	70
14	2	25	22	71
14	3	25	23	70
…	…	…	…	…

Relational Algebra applied to WSN

- Role of operators in WSN
  - Select
    - Can be used to filter useful readings and to detect alarms
  - Project
    - Can be used to reduce size of tuples to cope with small size memory of nodes and to reduce cost of sending data through the network
  - Join
    - Can be used to relate data acquired by different nodes and to relate historical data
  - Aggregation
    - In-network aggregation can be used to reduce the amount of data to be transmitted and to abstract over groups of nodes

Select Operation in WSN

$$\sigma_{pred}\,(S)$$

- takes
  - a stream S
  - a predicate *pred*
- returns a new stream containing rows of S that satisfy predicate *pred*
- Suppose *pred* is used to encode an alarm (for instance Temperature > 100)
  - The selection does not produce tuples until an alarm occurs (temperature is above 100)

Project Operation in WSN

$$\Pi_{a1,\ldots,an}(S)$$

- takes
  - a stream S
  - a set of fields *a1,…an* of S
- returns a new stream containing columns of S corresponding to attributes *a1,…an*
- Memory resources of nodes are limited; sending data among nodes has an high cost
  - Projection can be used to eliminate unwanted fields to be able to process queries with small size memory and to save energy when sending data

Natural Join in WSN

$S1 \bowtie S2$

- takes
  - two streams S1 and S2
- returns a new relation obtained as
  - $\Pi_{a1,...,am}(\sigma_{R.a1=S.a1,...,R.an=S.an}(S1xS2))$
  - where *a1,…,an* are common attributes of S1 and S2
  - *a1,…,am* is the union of attributes of S1 and S2
  - S1xS2 is the Cartesian product of the two streams
- Given that Streams are potentially infinite, Cartesian product is a problem
  - The finite set of tuples that participate in the join should be identified
- Join can be used to relate data simultaneously acquired by different nodes
- Join can be used to relate current events with others that happened in the past

Aggregate Functions and Operations in WSN

- An **aggregation function** takes a set of values and returns a single value.

  avg: average value **min**: minimum value
  max: maximum value **sum**: sum of values
  count: number of values

- In WSN it can be useful to aggregate
  - data acquired by different nodes
    - E.g. the average temperature measured in a large room
  - data acquired at different timestamps
    - E.g the average temperature measured during the day

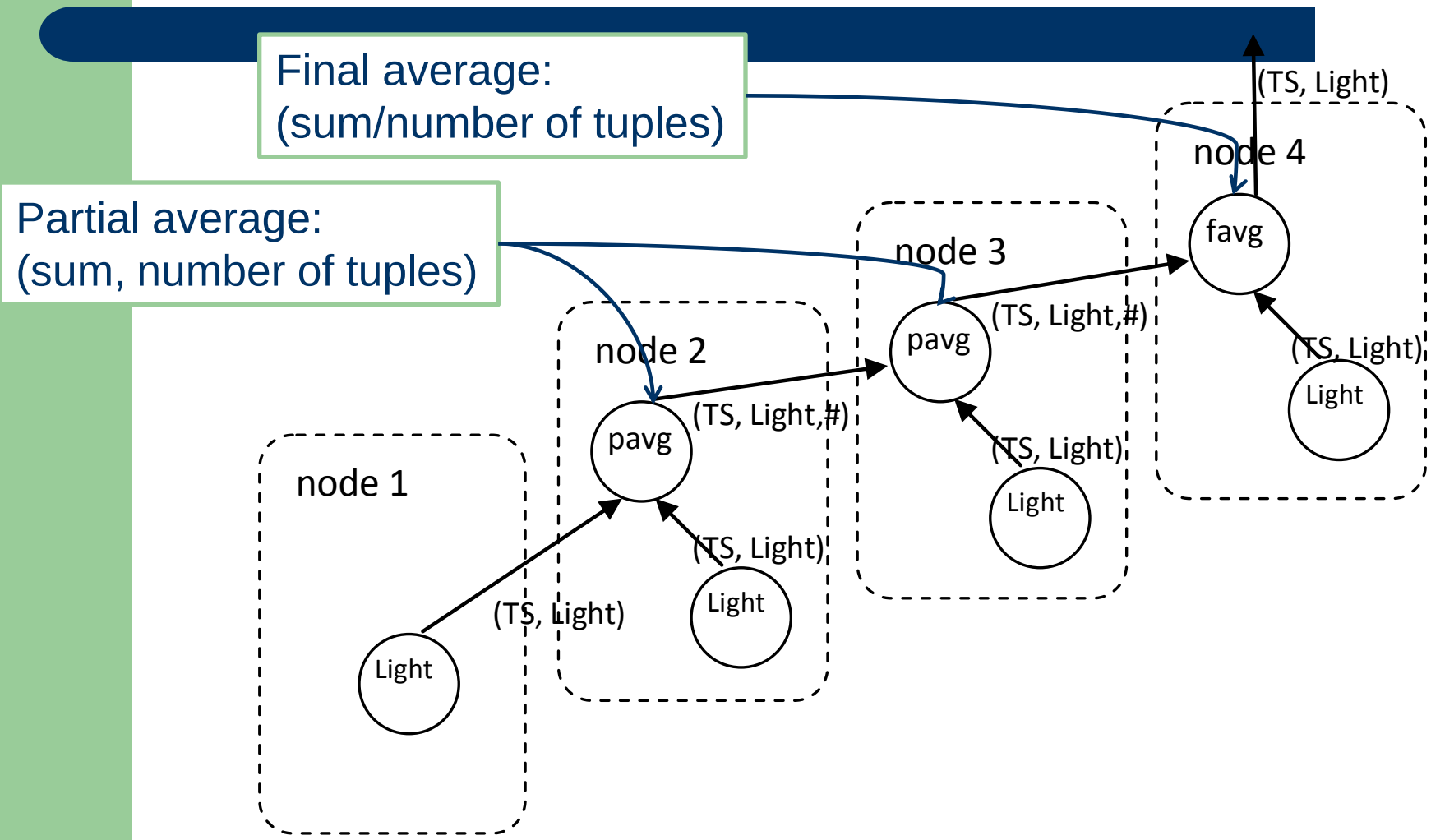
Aggregating data acquired by different nodes (1)

- Trivial solution: centralized aggregations
  - All nodes send acquired data to a node that computes the aggregation
  - It creates a bottleneck
    - Computation is done by one single node that can prematurely consume its energy
  - It has high communication cost
    - It is difficult to exploit proximity between nodes to save transmission energy

Aggregating data acquired by different nodes (2)

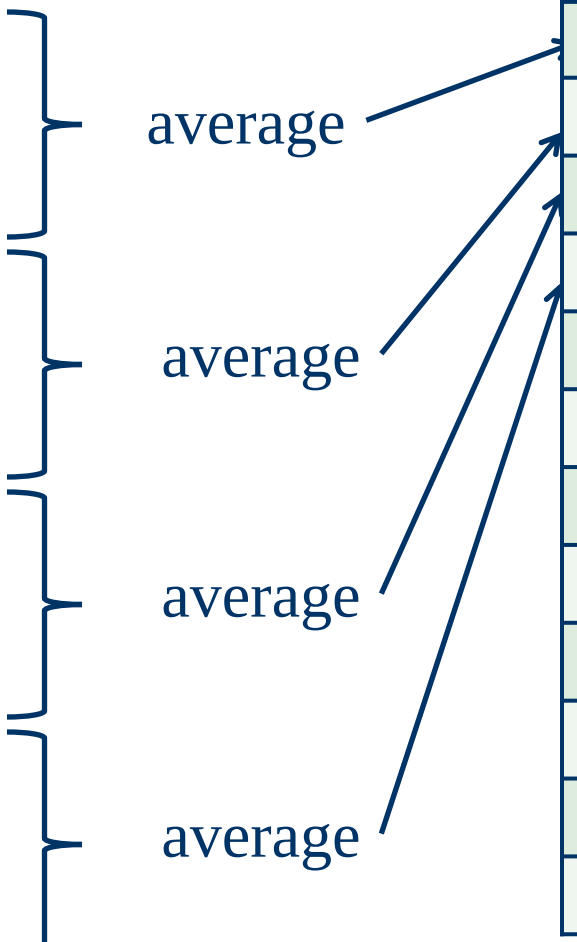
- Distributed computation of aggregation
  - Many relevant aggregation functions can be factorized into simpler functions
  - Different nodes can simultaneously execute the simple functions to contribute to the computation of the overall aggregation

Distributed computation of the average of data acquired by different nodes



Aggregation of data in a time window

Timestamp	Light
11	24
12	25
13	26
14	27
15	28
16	29
17	30
18	31
19	32
20	33
21	34
22	35



Timestamp	Light
13	25
16	28
19	31
22	34
…	…

Outline

- Data management in WSN
- Query processing in WSN
  - From traditional databases to WSN databases
  - ***Data Stream query processing issues***
  - Query processing/optimisation issues in WSN
- State of the art
- Future research directions

Issues in Data Stream Query Processing

- Continuous queries
  - Given that streams are potentially infinite, queries may run forever continuosly
- Blocking Operators
  - Some operators just work when relations are finite

Blocking Query Operators

- No output at all until entire input seen
- Streams – input never ends: only non-blocking operators are allowed
- Traditional SQL aggregates are blocking
  - Cannot determine the “max” until the entire relation is seen
- Many other SQL operators are have a blocking implementation in RDBMS
  - But they are not intrinsically blocking: group by, join
    - Large buffers might be required to store pending records
      - **Example**: in case of join operator, when two records match they can be delivered, however all records should be kept given that new additional matching records can come later

Avoiding Blocking Behavior

- Using Windows
  - aggregates on a limited size window are approximate and non-blocking
- Punctuations
  - Aggregates until some agreed mark (a portion of a stream is considered)
- Assertion about future stream contents
  - Unblocks operators, reduces state

Relational Query Operators on Streams

- Selection and project: no problem.
  - Record can be processed and possibly delivered as they come
- Ordering: not possible
  - The entire relation should be seen before delivering a single record
- Joins:
  - General case problematic on streams
    - May need to join arbitrarily far apart stream tuples
  - Natural join on stream-ordered attributes is tractable
    - but not always usable
  - A solution can be to join one stream and a window specified on another stream(also multiple windows)

```
Select A.value, B.value
from Source1 A  [window T], Source2 B
where A.ID = B.ID
```

Multi-way Sliding Window Joins

- Evaluation of n-way sliding window joins queries
  - n streams with associated sliding windows
  - continuously evaluate the joins between all n windows
- Two natural joins strategies for this
  - eager: join is evaluated each time a new tuple arrives in any of the input streams
  - lazy: join is evaluated with some pre-specified frequency, e.g., every t time units

Aggregation

- Grouping with aggregate operations are blocking
- Example

```
select avg(temp), floor
from sensors
group by floor
```

- (sliding) windows is a widely used solution

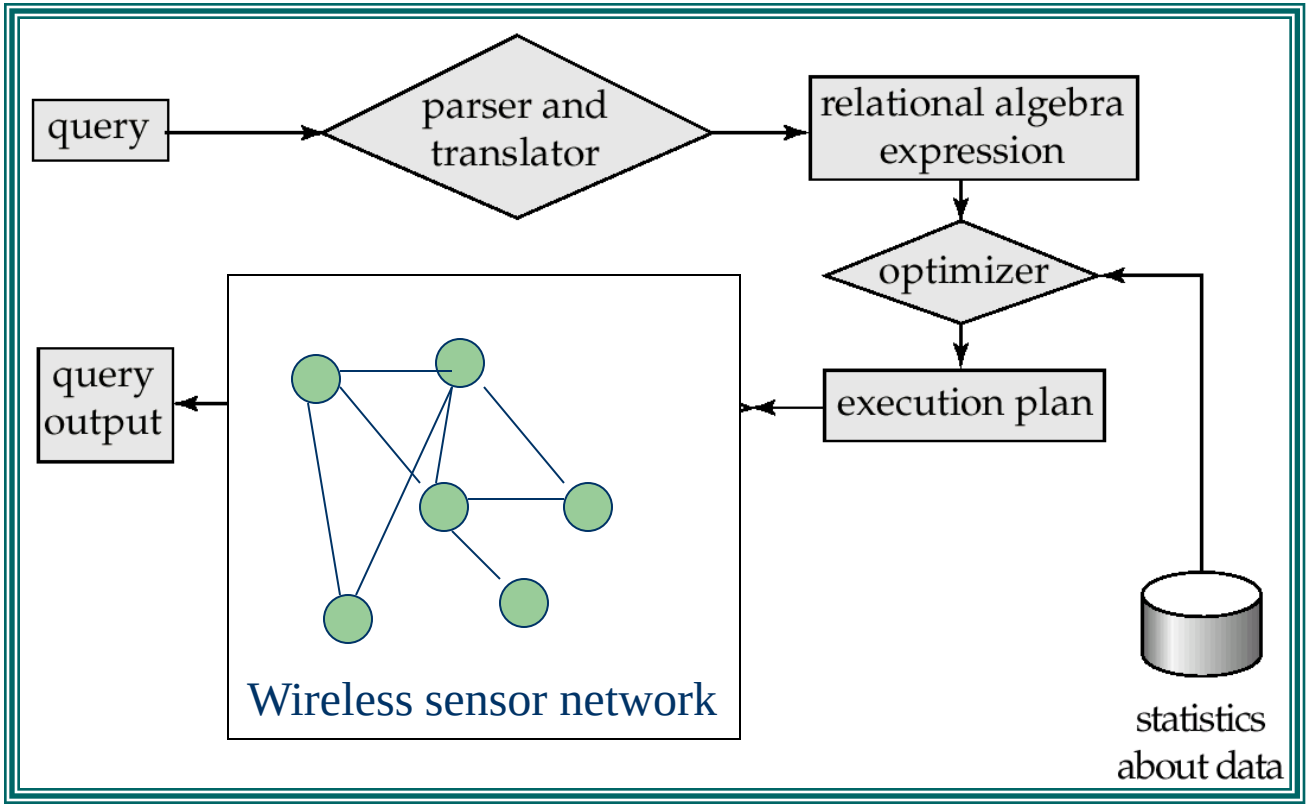
Aggregation with Approximation

- When aggregates cannot be computed exactly in limited storage, approximation may be possible and acceptable
  - Statistics are used to estimate results
- Examples:
  - select G, median(A) from S group by G
  - select G, count(distinct A) from S group by G
    - Can be estimated by using summary structures
      - samples, histograms, sketches …

Outline

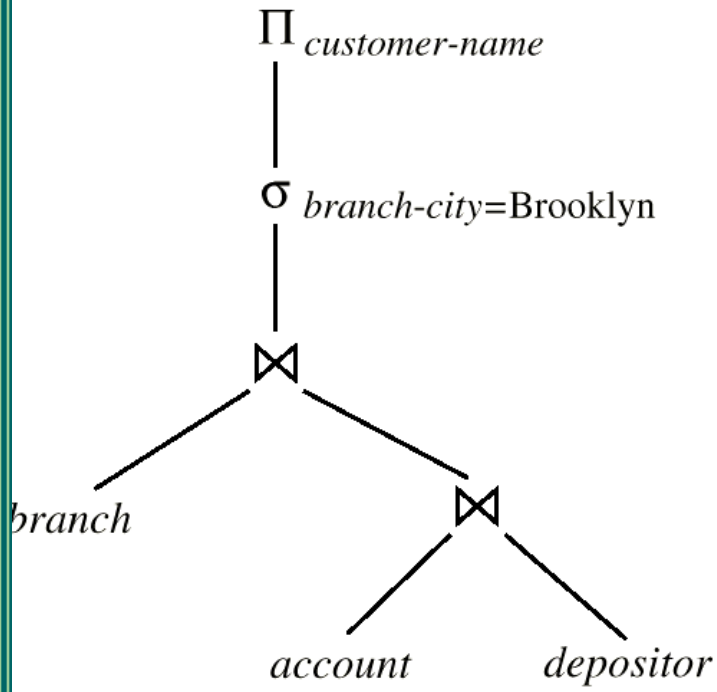
- Data management in WSN
- Query processing in WSN
  - From traditional databases to WSN databases
  - Data Stream query processing issues
  - ***Query processing/optimisation issues in WSN***
- State of the art
- Future research directions

Basic Steps in Query Processing

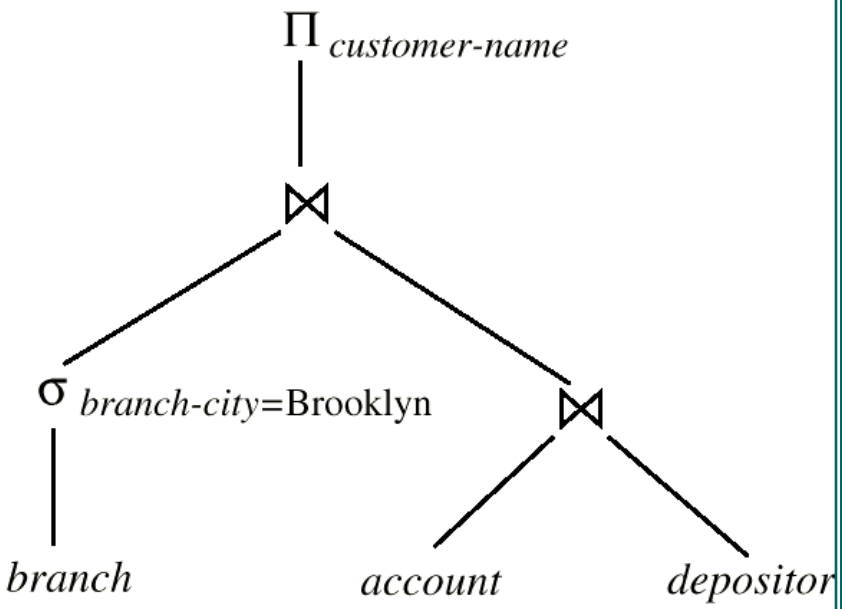


Optimization

Relations generated by two equivalent expressions have the same set of attributes and contain the same set of tuples, although their attributes may be ordered differently.



(a) Initial Expression Tree



(b) Transformed Expression Tree

Optimization

- Generation of query-evaluation plans for an expression involves several steps:
  1. Generating logically equivalent expressions
     - Use **equivalence rules** to transform an expression into an equivalent one.
  2. Annotating resultant expressions to get alternative query plans
  3. Choosing the cheapest plan based on **estimated cost**
- The overall process is called **cost based optimization.**

Heuristic Optimization in traditional DB

- Cost-based optimization is expensive
- Systems may use heuristics to reduce the number of choices that must be made in a cost-based fashion.
- Heuristic optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improve execution performance:
  - Perform selection early (reduces the number of tuples)
  - Perform projection early (reduces the number of attributes)
  - Perform most restrictive selection and join operations before other similar operations.
  - Some systems use only heuristics, others combine heuristics with partial cost-based optimization.

Steps in Typical Heuristic Optimization in traditional DB

- 1. Deconstruct conjunctive selections into a sequence of single selection operations
- 2. Move selection operations down the query tree for the earliest possible execution
- 3. Execute first those selection and join operations that will produce the smallest relations
- 4. Replace Cartesian product operations that are followed by a selection condition by join operations
- 5. Deconstruct and move as far down the tree as possible lists of projection attributes, creating new projections where needed
- 6. Identify those subtrees whose operations can be pipelined, and execute them using pipelining).

Structure of Query Optimizers

- Some query optimizers integrate heuristic selection and the generation of alternative access plans.
- Even with the use of heuristics, cost-based query optimization imposes a substantial overhead.
- This expense is usually more than offset by savings at query-execution time, particularly by reducing the number of slow disk accesses.

Query execution cost in WSN

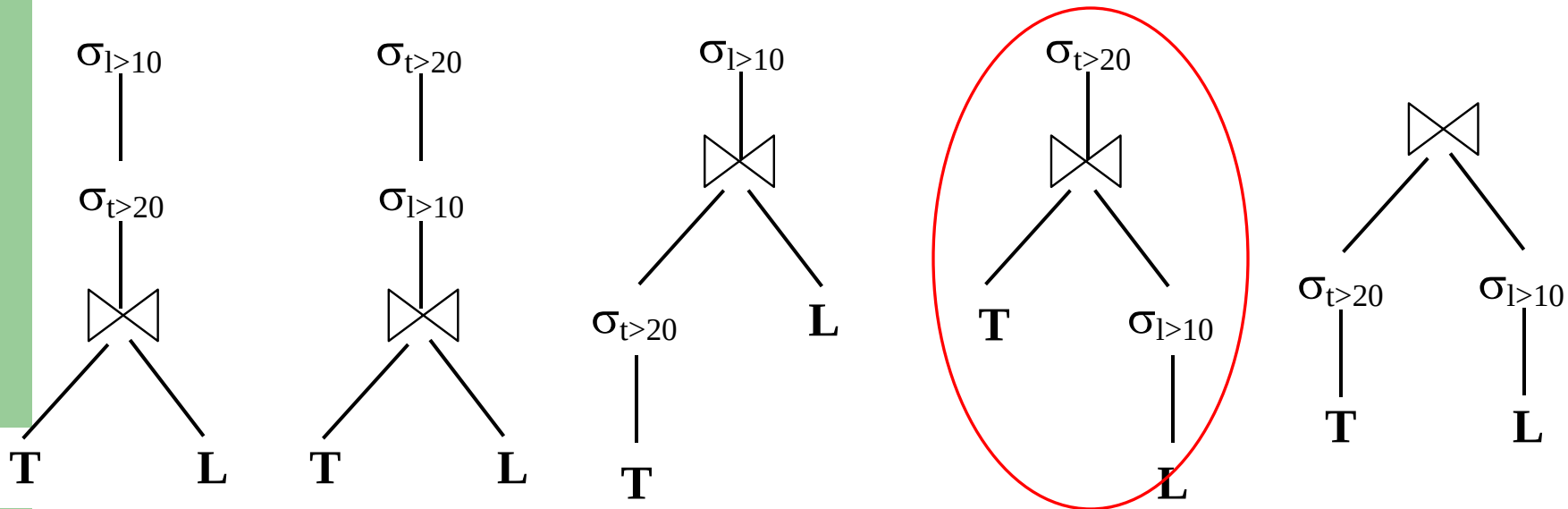
- In traditional databases query processing cost is typically estimated in terms of disc accesses
  - Number of record reads, size of temporary results, …
  - The optimisation objective is high throughput query execution
- In WSN cost is estimated in terms of energy consumed
  - The objective is increasing autonomy of nodes
- Activity that consumes energy is data access
  - Data acquisition from a transducer
    - (different transducers have different activation costs)
  - Accessing data in a remote node
    - (during distributed query processing nodes have to exchange data)
  - Accessing data stored locally
    - (processor and main memory usage)
- The above three cases should be taken into account

Simple query example

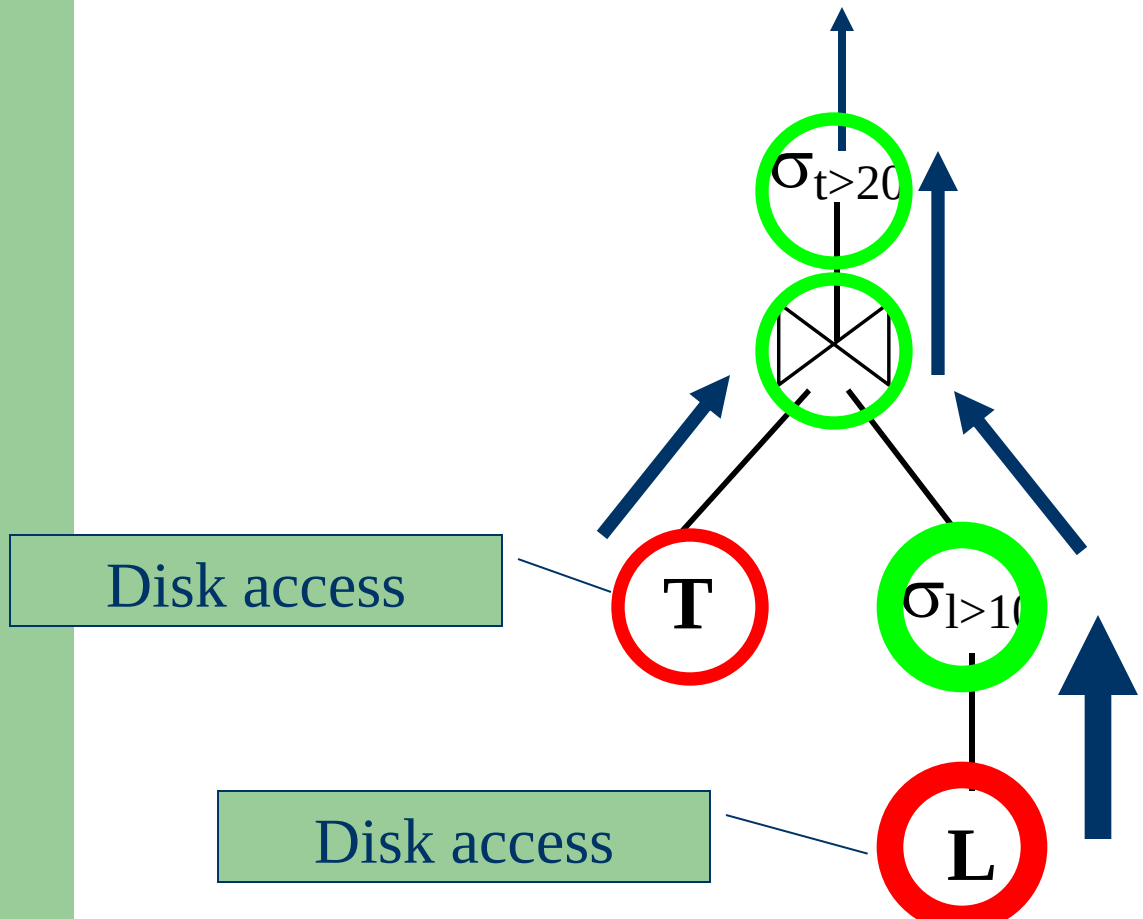
```
select t, l
from T, L
Where T.t_s=L.t_s
      t>20 and
      l > 10
```

T		
t_s	t	…
0	10	..
1	15	…
…	…	…

L		
t_s	l	…
0	5	..
1	7	…
…	…	…



Traditional DB execution

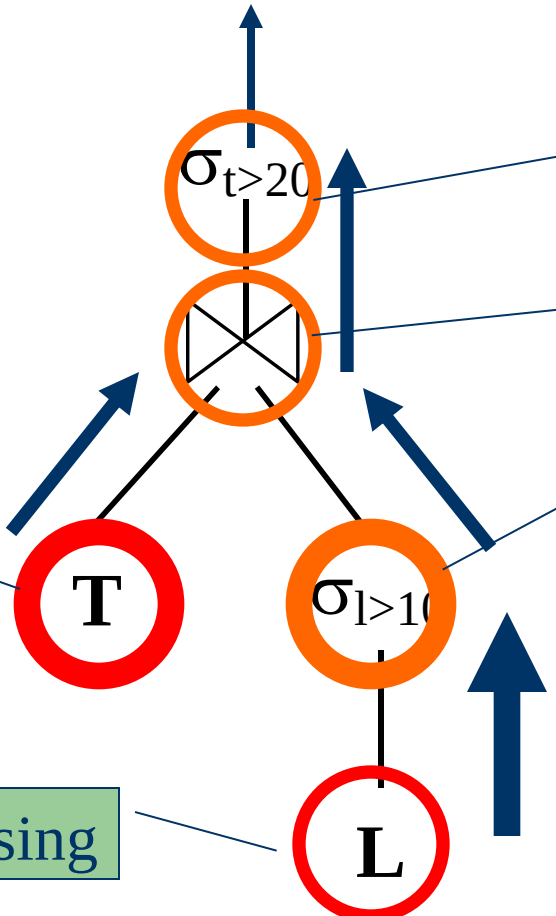


Power aware query optimisation

- Cost of a query is estimated considering energy consumption
  - Processing data consumes energy
  - Sensing data consumes energy
    - Different sensors have different energy consumpiton
  - Transmitting data consumes energy
  - Receiving data consumes energy

Execution on a single node of a WSN

Cost:
(Cl+Cpr)*n+
P(l>10)*(Cpr+Ct)*n+
P(l>10)*Cpr*n



Data Processing

Data Processing

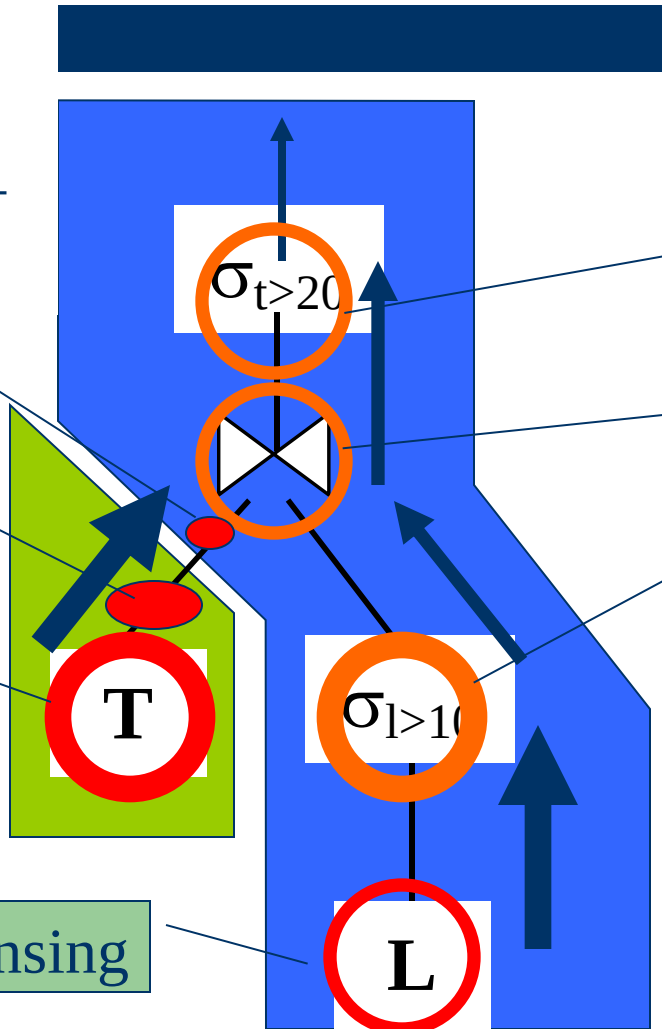
Data Processing

Expansive sensing

Low energy sensing

Execution on two nodes of a WSN

Cost:(Cl+Cpr)*n+
(Ctx+Ct+
P(l>10)*(Cpr+Crx))*n+
P(l>10)*Cpr*n



Receive what needed

Transmit all

Expansive sensing

Low energy sensing

Data Processing

Data Processing

Data Processing

```
SELECT t, l
FROM T, L
WHERE T.t_s=L.t_s
    t>20 AND
    l>10
```

	P(t>20)	0.2	Cost of sensor t: Ct	12	Cost of transmission: Ctx	10
	P(l>10)	0.7	Cost of sensor l: Cl	2	Cost for receiving: Crx	6
			Cost of processing:Cpr	1		

Ex. Plans	Traditional DB		Single node		Multiple node	
$R1= T \bowtie L$	2 * n		(Ct+Cl+Cpr)*n		(Ct+Cl+Cpr+Ctx+Crx)*n	
$R2=\sigma_{t>20}(R1)$	n	**3.2**	Cpr*n	**16**	Cpr*n	**32.2**
$R3=\sigma_{l>10}(R2)$	P(t>20)*n		P(t>20)*Cpr*n		P(t>20)*Cpr*n	
$R1= T \bowtie L$	2* n		(Ct+Cl+Cpr)*n		(Ct+Cl+Cpr+Ctx+Crx)*n	
$R2=\sigma_{l>10}(R1)$	n	**3.7**	Cpr*n	**17**	Cpr*n	**32.7**
$R3=\sigma_{t>20}(R2)$	P(l>10)*n		P(l>10)*Cpr*n		P(l>10)*Cpr*n	
$R1= \sigma_{t>20}(T)$	n		(Ct+Cpr)*n		(Ct+Cpr+P(t>20)*Ctx)*n	
$R2= R1 \bowtie L$	P(t>20)*2*n	**1.6**	P(t>20)*(Cpr+Cl)*n	**14**	(Crx+P(t>20)*(Cpr+Cl))*n	**21.8**
$R3=\sigma_{l>10}(R2)$	P(t>20)*n		P(t>20)*Cpr*n		P(t>20)*Cpr*n	
$R1= \sigma_{l>10}(L)$	n		(Cl+Cpr)*n		(Cl+Cpr)*n	
$R2= R1 \bowtie T$	P(l>10)*2*n	**3.1**	P(l>10)*(Cpr+Ct)*n	**13**	(Ctx+Ct+P(l>10)*(Cpr+Crx))*n	**30.6**
$R3=\sigma_{t>20}(R2)$	P(l>10)*n		P(l>10)*Cpr*n		P(l>10)*Cpr*n	
$R1= \sigma_{t>20}(T)$	n		(Ct+Cpr)*n		(Ct+Cpr+P(t>20)*Ctx)*n	
$R2= \sigma_{l>10}(L)$	n	**2.9**	(Cl+Cpr)*n	**17**	(Cl+Cpr)*n	**23.1**
$R3= R1 \bowtie R2$	(P(t>20)+P(l>19))*n		(P(t>20)+P(l>19))*Cpr*n		((P(t>20)+P(l>10))*Cpr+P(l>10)*Crx)*n	

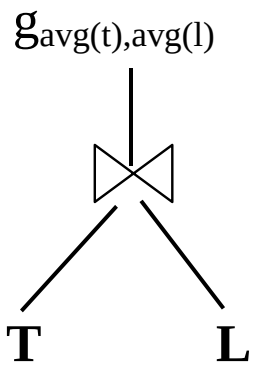
Aggregation query example

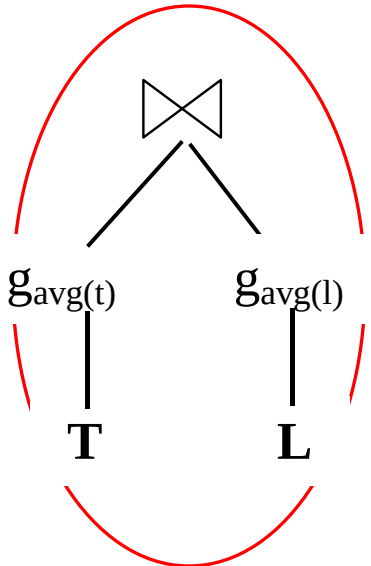
```
select avg(t),avg(l)
from T, L
where T.t_s=L.t_s
```

T		
t_s	t	…
0	10	..
1	15	…
…	…	…

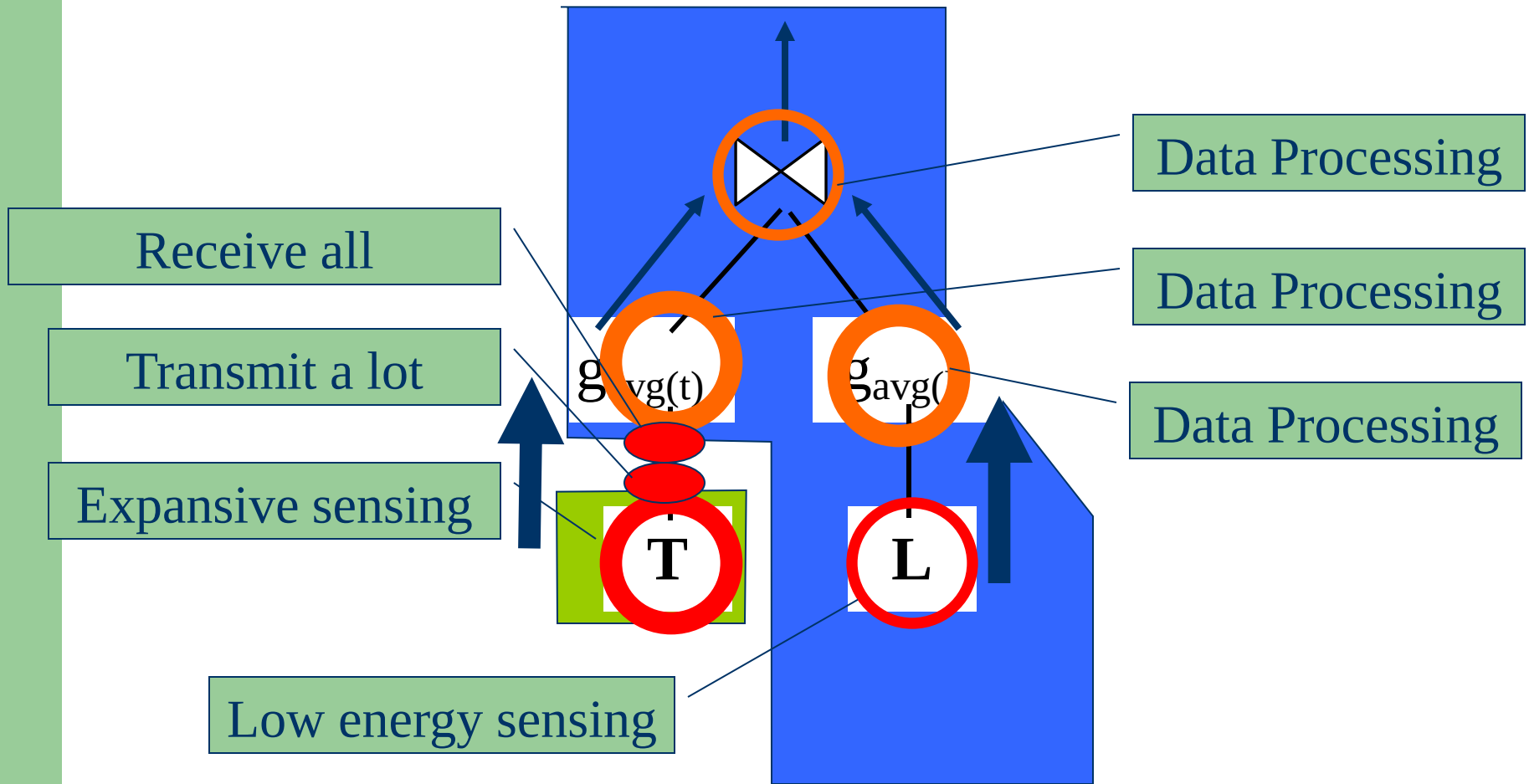
L		
t_s	l	…
0	5	..
1	7	…
…	…	…

They are equivalent just if t_s is unique!

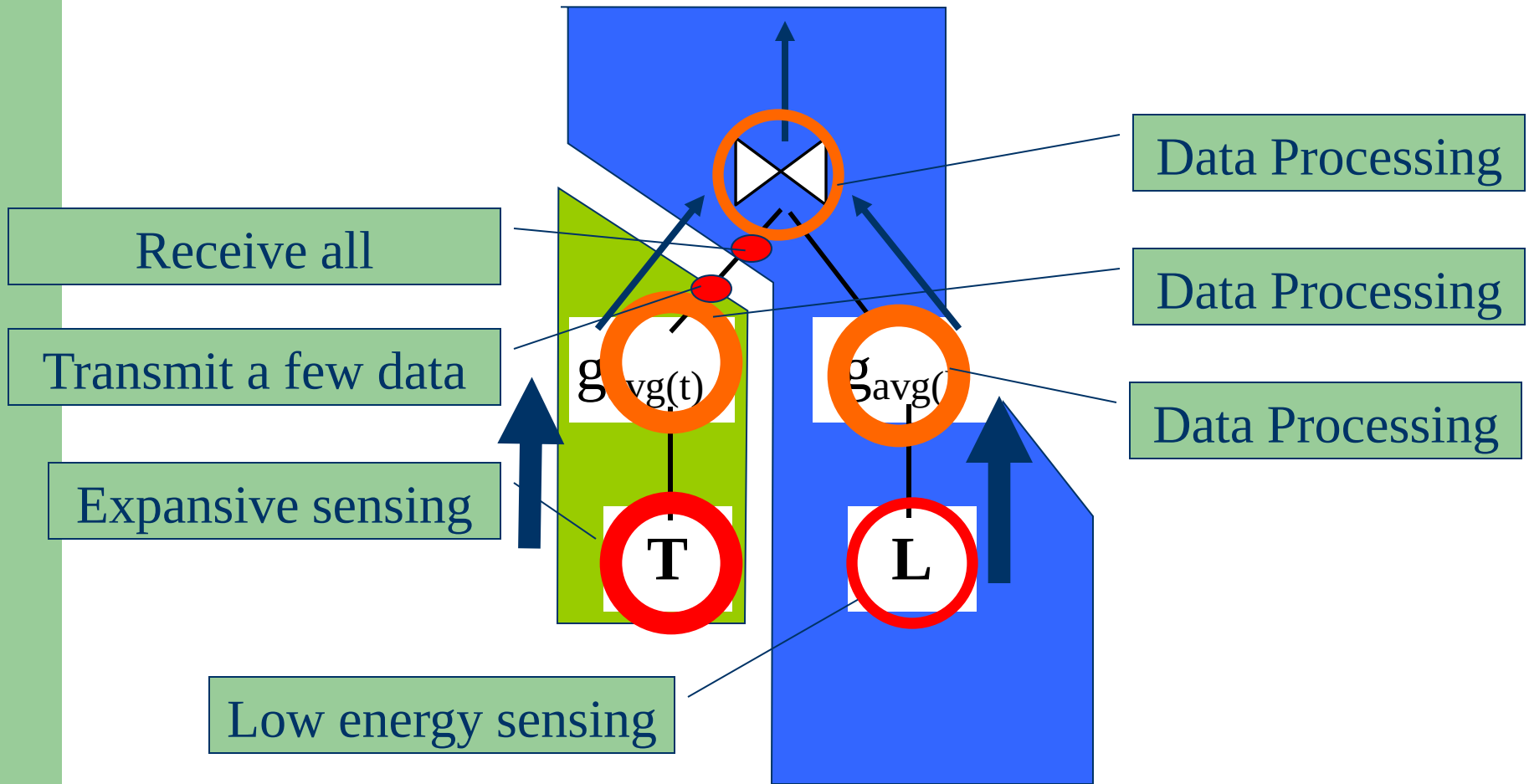




Execution on two nodes of a WSN (1)



Execution on two nodes of a WSN (2)



Outline

- Data management in WSN
- Query processing in WSN
- *State of the art*
  - Cougar
  - Fjord
  - TAG
  - TinyDB
- Future research directions

Cougar Approach

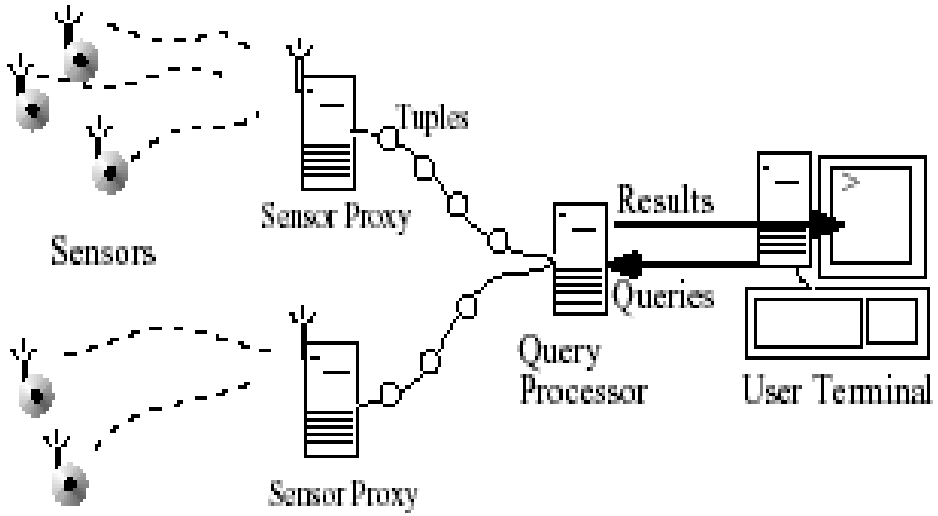
- Data models:
  - stored data (node ID, position, …) → relations
  - sensor data (acquired from physical environment) → sequences
- Sensors model:
  - a sensor Abstract Data Types (ADT) is defined for all sensors of a same type;
  - a physical sensor is an instance of an ADT;
  - public interface consists of signal processing functions.
  - For instance an ADT may contain a function
    - *getTemp()* which when invoked returns current temperature
    - *detectTempAlarm(threshold)* which when invoked returns temperature when above the threshold

Cougar Approach (2)

- Three layers
  - Sensor layer
    - ADT
      - When a function returns a result it sends it to the above layer and then it is re-invoked
  - Leader layer
    - Special nodes that coordinate activity of group of nodes
      - For instance the aggregate operations
  - Front-end
    - Data acquired by nodes is processed on a PC
    - It is possible to relate data acquired by different nodes
    - However, no temporal aggregates are possible

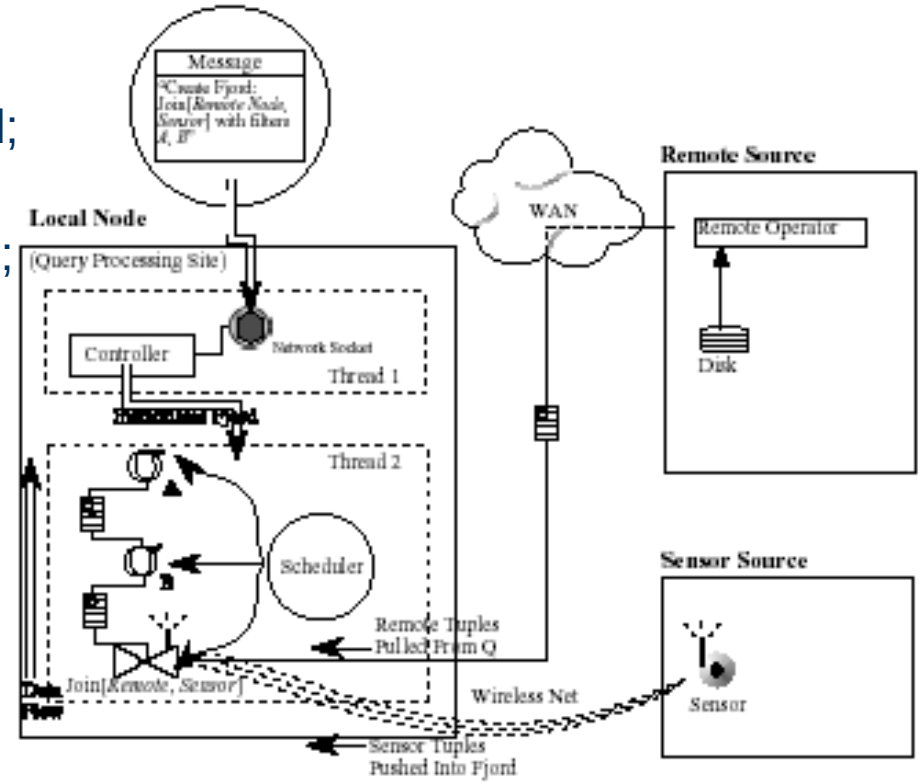
Fjord Approach (1)

- Centralized architecture
- Two advantages:
  - supporting the combination of data stream and disk-saved data;
  - defining power-sensitive operators (sensor proxies) as mediator between sensors and query processor.
- Architecture:



Fjord Approach (2)

- Other sensor proxy functions:
  - adjusting the sampling and delivering rate of sensors depending on user demand;
  - asking sensors to transmit only data required by users;
  - asking sensor for aggregation.



Fjord Approach (3)

- Important result:
  - Using only one Fjord for all similar queries over a sensor consumes less energy than allocating a separated Fjord for each new query.
- Reasons:
  - no overhead due to context switch between threads;
  - sensor tuples are put in the buffer pool of the only one Fjord.

TAG Approach (1)

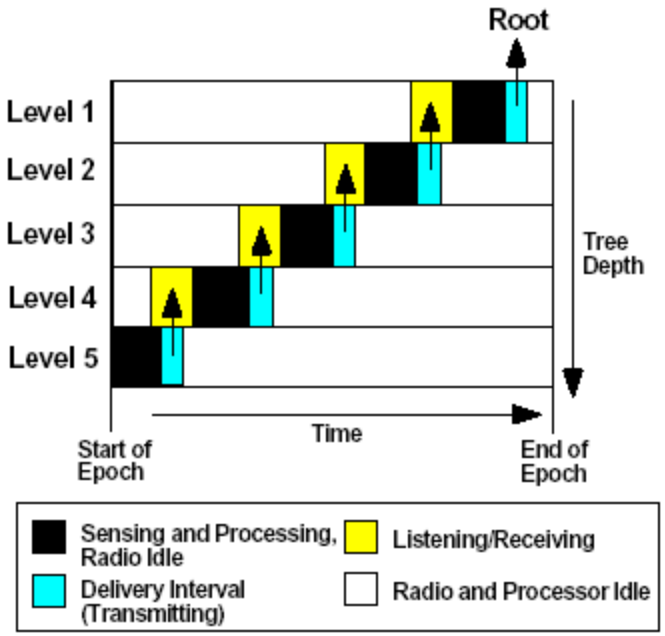
- A distributed (spatial) aggregation service for ad hoc networks of TinyDB Motes
  - Sensors acquire data once per epoch
  - TAG aggregate data produced by different sensors in the same epoch
  - Example: average temperature in the first floor
- Steps:
  - users pose aggregation queries from a powered base station;
  - each query is routed to all nodes of network;
    - During the query diffusion a routing tree is built
  - each node delivers results back to the user through a routing tree rooted at the base station;
  - as data flows up the tree, each node combines received data and locally produced ones.

TAG Approach (2)

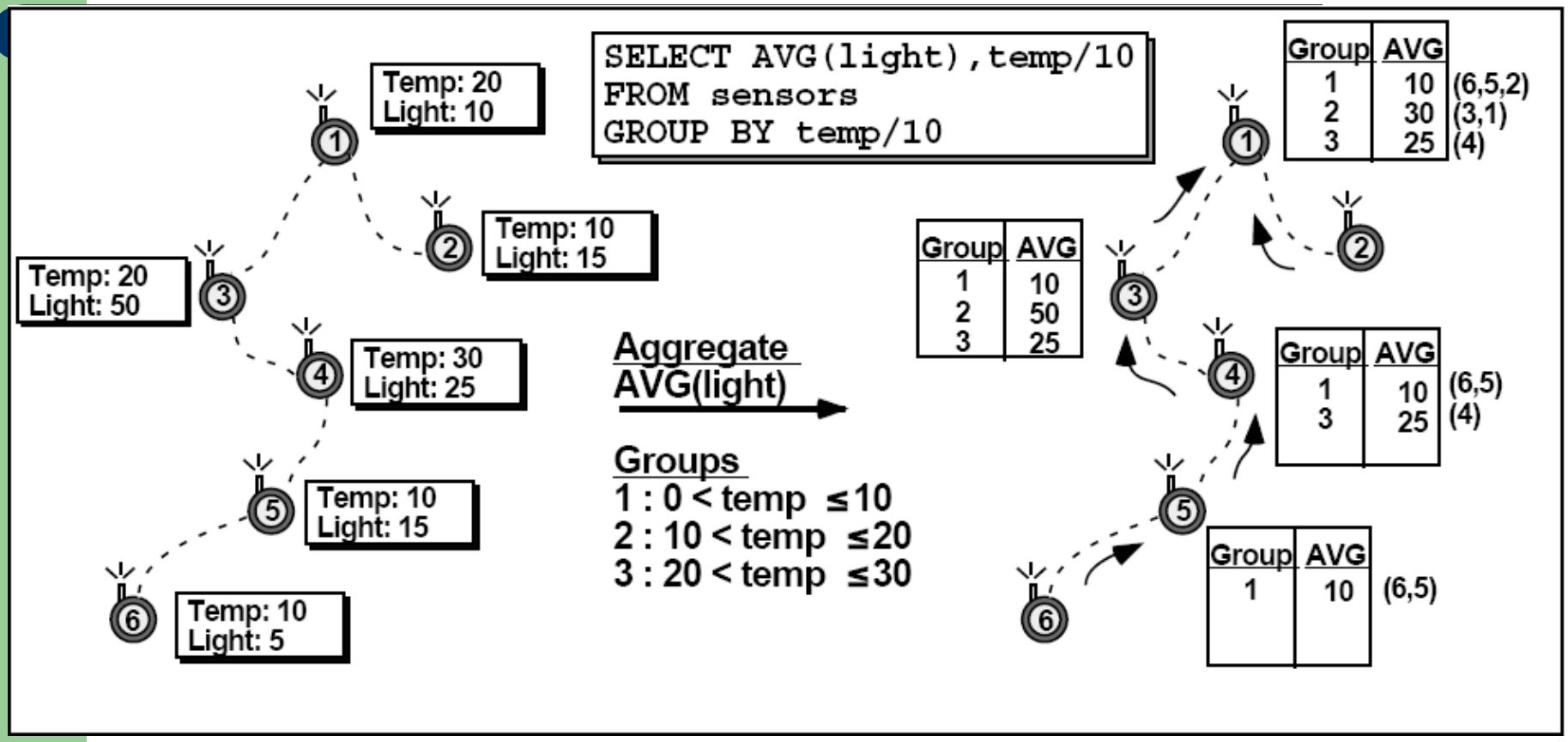
- Building of the routing tree:
  - the base station broadcasts a message into the network;
  - when any node receives this message, it chooses the sender as its parent and rebroadcasts the message;
  - the tree building ends when all nodes have set their level.
- When a node has a data to send to the root, it delivers it to its parent, and so on until the message reaches the root
- Routing messages are transmitted periodically in order to adapt the routing tree to topology changes

TAG Approach (3)

- Query execution (two-phase process):
  - query is sent to all nodes down into the tree;
  - aggregate values are routed up from children to parents.
- Advantages:
  - reducing communication
  - tolerating disconnections;
  - idle intervals for processor and radio are easily identified.



Tag Approach (4)



TinyDB Approach (1)

- Architecture for distributed execution of queries in networks of Mica motes
- Extends the TAG approach
  - Not limited to data aggregation
  - Optimize the data acquisition process from transducers
- A Query is received by the base station that
  - parses it,
  - optimizes it
  - send to the network
- A query is global: it is (possibly) processed by all nodes
  - Restrictions on static attributes may limit the nodes that actually process the query

TinyDB Approach (2)

- Example:

```
select light, temperature
from sensors
where light>20
```

  - All nodes return light and temperature when light >20

```
select light, temperature
from sensors
where x>20, y>100
```

  - Nodes with specified coordinates return light and temperature

TinyDB Approach (3)

- Queries process a single table “sensors”
- Table “Sensors” is *logically* populated adding new tuples every epoch
  - In every epoch each node add a tuple in the table: each row corresponds to a node reading in an epoch
  - The number of row generated in an epoch is equal to the number of nodes
- “Sensors” has a column for each type of physical sensor

node ID	time	node posit.	temp. sensor	light sensor	press. sensor	...

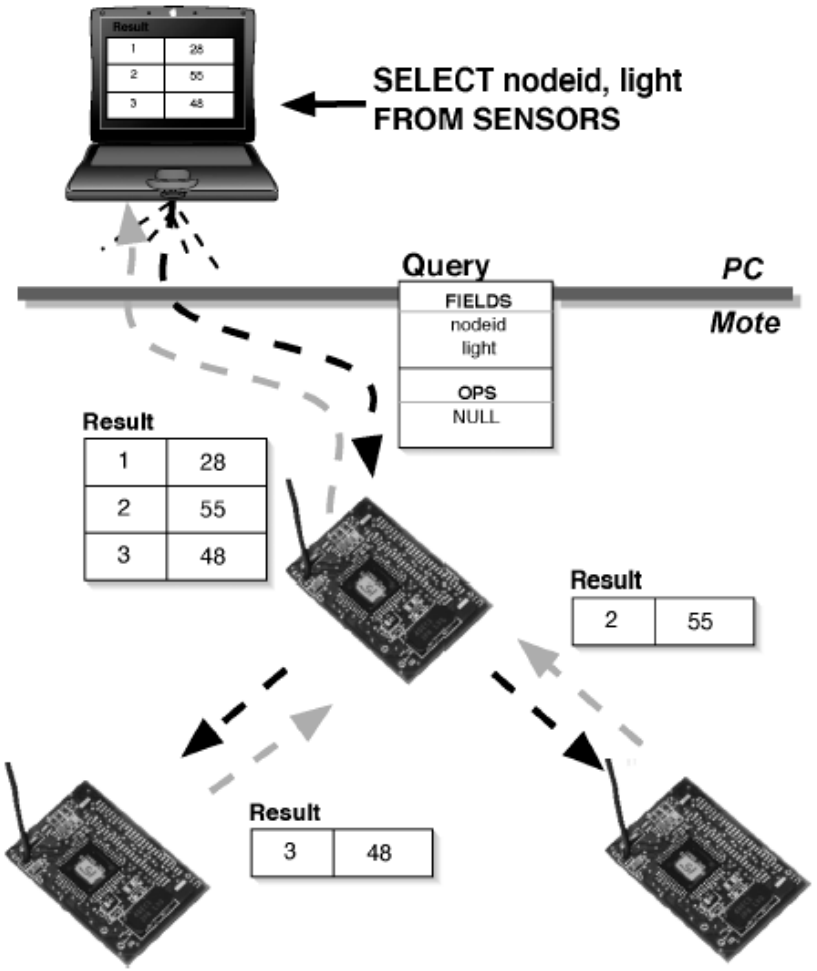
- Table sensor *non-materialized*: *it* is *logically* distributed across nodes of the entire network
  - Every node owns records corresponding to data that it produces
  - A query is executed in parallel in all nodes. The query execution in a node just process the records produced by that node
- A query is processed repetitively every epoch

TinyDB Approach (4)

- Query execution steps:
  - users submit queries to a base station where they are parsed and optimized;
  - data is acquired only when it is required by a predicate;
  - sampling operations are executed in increasing energy order;
  - queries are sent only to those nodes with relevant data;
  - a semantic routing tree (SRT) is built (only for static attributes);
  - nodes are synchronized and sleep for most of each epoch;
  - acquired data is filtered and routed to operators;
  - the result is put into a queue with data from children, waiting for delivery to the parent.

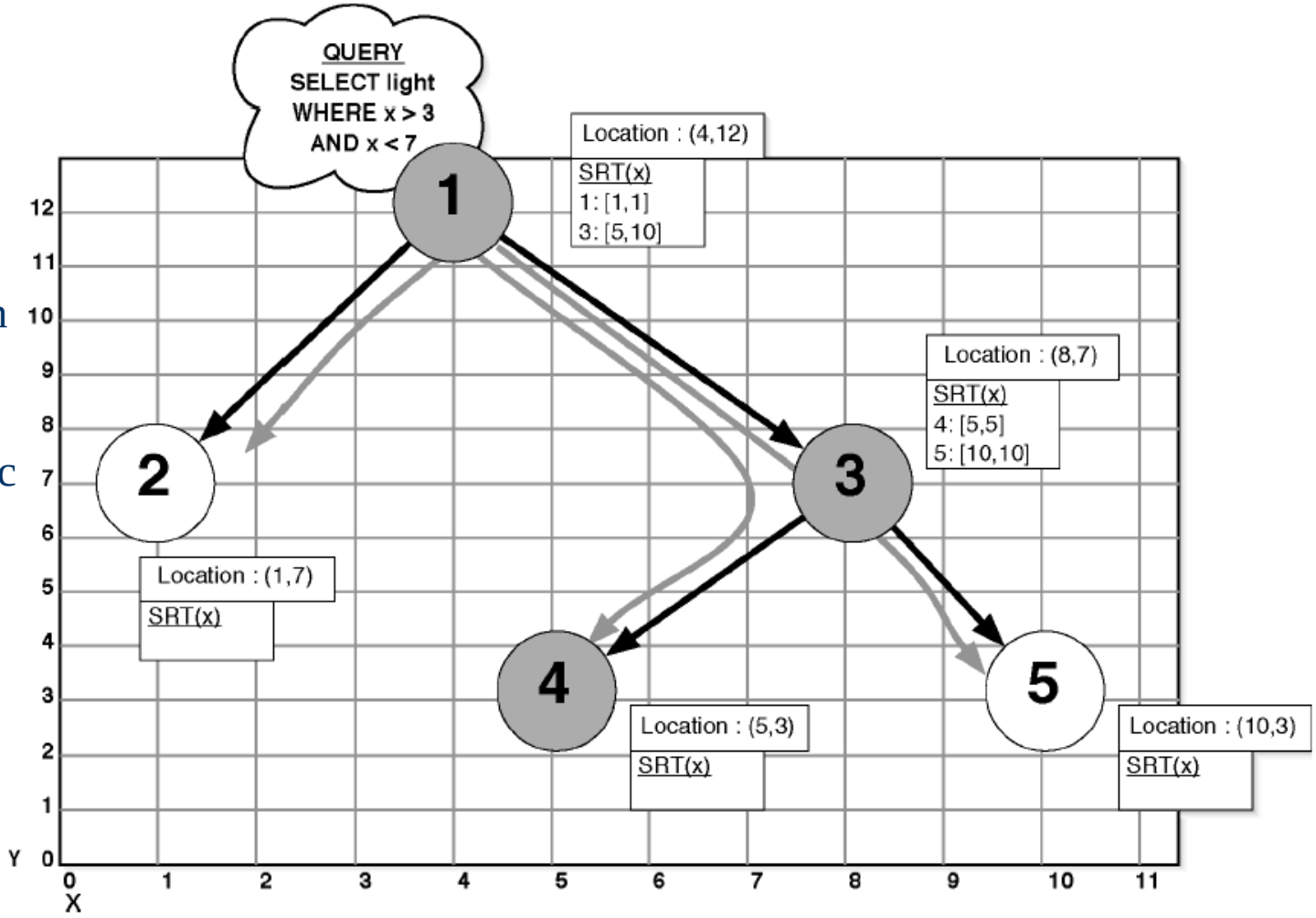
TinyDB Approach (5)

Query processing model:



TinyDB Approach (6)

Semantic Routing Tree (SRT): SRT is also used as an "index" to decide where a query should be sent, by using static attributes (x, in this example)



TinyDB Approach (7)

- Acquisitional query processing:
  - Granularity for query optimization is the field rather than the record
    - In traditional databases query optimization consider number of records (or block of data)
    - In sensors “generating” a value for a record has a cost
  - A transducer is activated just if needed
    - Some field in the “sensor” table might be empty just because they are not needed

TinyDB Approach (8)

- Example:

```
select light, mag
from sensors
where light > 20
        and mag > 70
```

- No need to acquire *temp*, *accel*, etc.
- Heuristic: acquisitions are ordered by increasing cost
  - First *light* is acquired. If condition (*light > 20*) is true then also *mag* is acquired
  - In several cases *mag* acquisition is not performed
  - This is an heuristic: it does not work in all cases

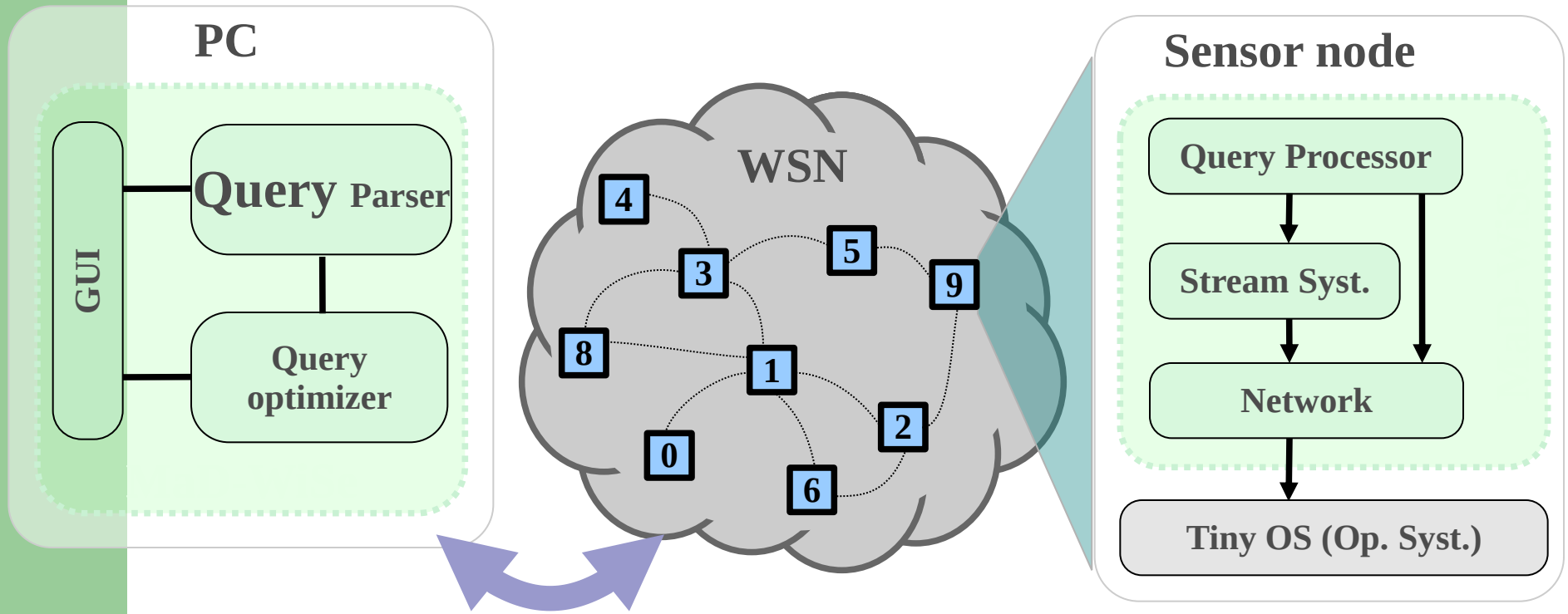
TinyDB Approach (9)

- Limitations:
  - Optimization made on global considerations
    - It is not possible to generate a query specially optimized for a specific node
  - It is not possible to relate data generated by different nodes (sensor table is distributed)
    - Example: is the temperature of room 1 greater than that of room2?
  - It is not possible to compute temporal aggregates (a query is processed once per epoch)
    - Example: give me the average temperature measured every minute during last 10 minutes

MaDWiSe approach

- Existing approaches do not distinguish among data acquisition, data transfer, data processing phases
- Our approach -> layered architecture:
  - Network layer
  - Stream system
  - Stream query processing
- All nodes of the WSN have these layers

MaD-WiSe architecture



Query Examples

```
SELECT *
FROM 5.Temperature
WHERE 5.Temperature>35
EVERY 20 SECONDS
```

```
SELECT Accel
FROM avg(5.Accel, 4.Accel)
EVERY 50 MSECONDS
```

```
SELECT avg(3.Audio)
FROM 3.Audio
EPOCH 100 SAMPLES
EVERY 10 MSECONDS
```

MaDWiSe-Network layer (1)

- Localization and Routing:
  - Virtual Coordinate assignment protocol (VCap):
    - Large sensor networks, not equipped with localization devices such as GPS
    - Distributed protocol which defines a coordinate systems unrelated to the sensor location
    - VCap selects three anchors in the network boundary and assigns to each node a triplet of coordinates which represents the hop distance of the node from the three anchor nodes
    - The coordinate system can efficiently support greedy geographic routing

MaDWiSe -Network layer (2)

- Energy-efficient, application-driven communication:
  - Many applications use channels at fixed data rate
    - E.g.: directed-diffusion paradigm or data-base oriented applications
  - Connection-oriented communication protocol
  - Estimation of the next packet arrival time and turn on/off the radio accordingly
    - Minimization of the packet losses due to radio off
    - Minimization of energy consumption
  - Sensors not involved in the communication channels turn off the radio

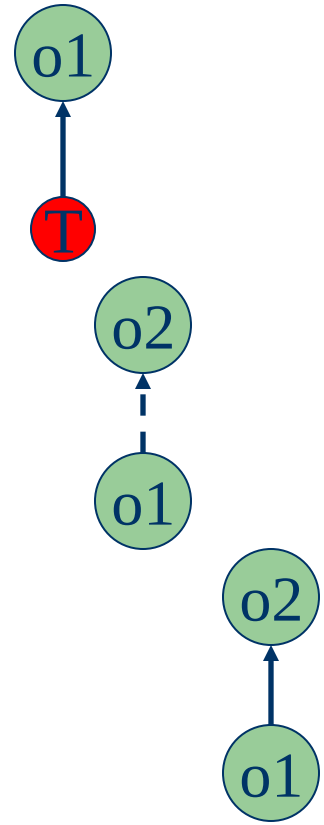
MaDWiSe - Stream system (1)

- Wireless sensor network mainly produce and process streams of data
- Tree types of data sources
  - Transducers -> *Sensor streams*
  - Local applications -> *Local streams*
  - Network -> *Remote streams*

- Stream system: the equivalent of the “file system” for WSN applications
  - open, close, read, write like operations on various type of streams
  - Streams are n -> 1 (n can write, 1 can read)
    - This limit is easily manageable

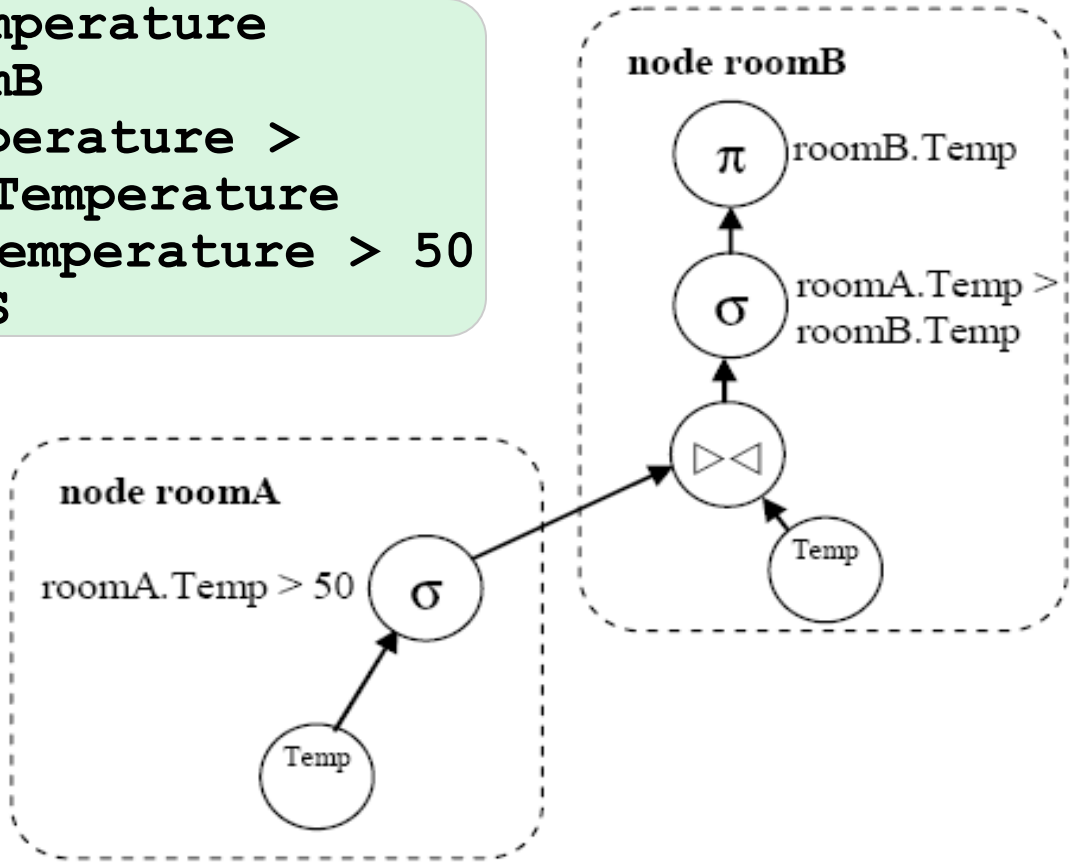
MaD-WiSe: Stream system (2)

- Algebra operators read and produce data streams
- Three types of data streams in MadWise
  - **Sensor streams**
    - Connect transducers with operators
    - Sampling:
      - Periodic (“every x milliseconds”)
      - On Demand (“when needed”)
    - Cost depends on the type of transducer
  - **Remote streams**
    - Connect query operators on different nodes
    - Radio communication is needed
    - Cost depends on length of paths between nodes
  - **Local streams**
    - Connect query operators on the same node
    - in RAM
    - Negligible cost



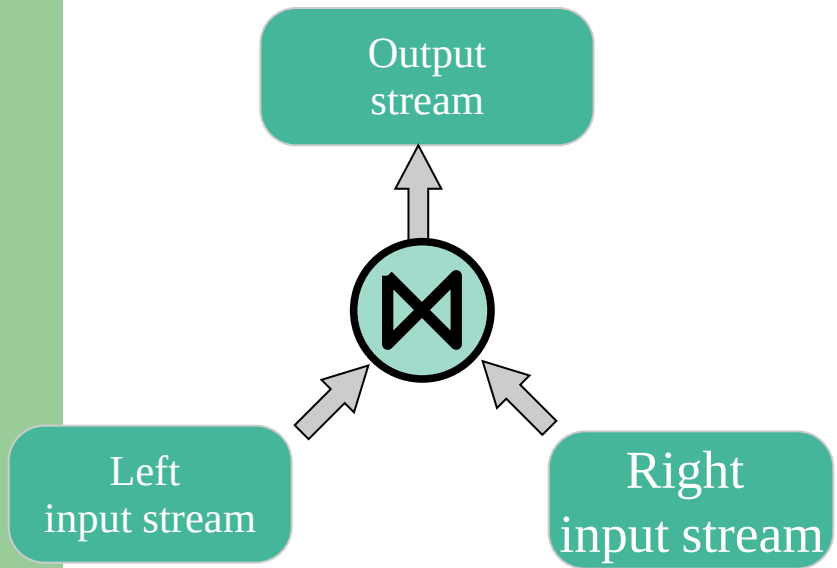
Mad-Wise: Distributed query processing

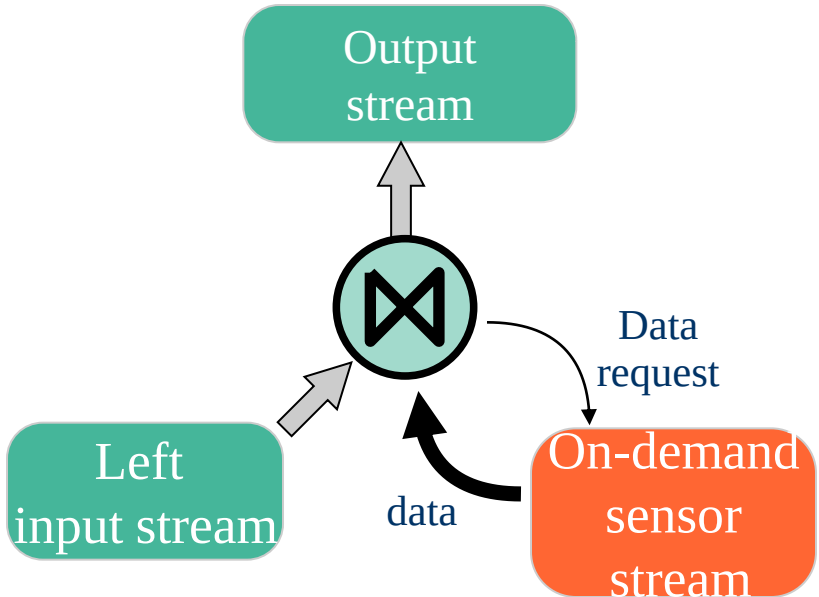
```
SELECT roomA.Temperature
FROM roomA, roomB
WHERE roomA.Temperature >
          roomB.Temperature
     and roomA.Temperature > 50
EVERY 20 SECONDS
```



MaD-WiSe: algebra operators

- "**Temp. Join**": joins tuples with same timestamp
  - Two implementations
    - Continuous join
    - Sync join





Optimisation heuristics

- In addition to various typical optimisation heuristics it is very relevant to:
  1. Use sync-join and on-demand streams when possible
     - To reduce acquisition cost
  2. Put unary operators on the node where data is acquired
     - To reduce communication cost
  3. Use left-deep-join-trees
     - to increase chances of applicability of 1

Query optimisation example

SELECT *
FROM 1.Magnetism, 2.Acceleration,
3.Temperature
WHERE *p*1(1.Magnetism)
and *p*2(2.Acceleration)
and *p*3(3.Temperature)
EVERY 1000

where we suppose that

Pr(*p1*)=0.01, *Pr(p2*)=0.05, Pr(*p3*)=0.1

and

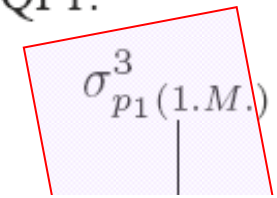
C(Magn.) = 0.27, C(Accel.)= 0.03, C(Temp) = 0.00009 mJ

Left deep join tree

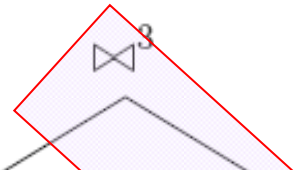
Push down and placement

Sync-join

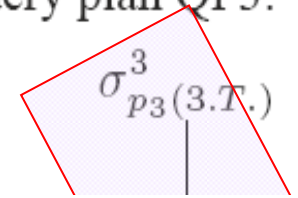
Query plan QP1:



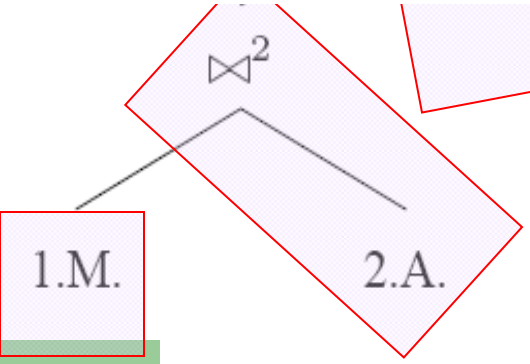
Query plan QP2:

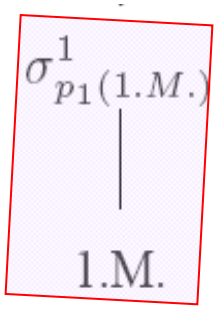


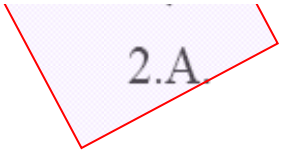
Query plan QP3:



		QP1:		QP2:		QP3:	
Action	**Energy(mJ)**	**Freq.**	**Power**	**Freq.**	**Power**	**Freq.**	**Power**
Acquire M.	0.2685	1	0.2685	1	0.2685	1	0.2685
Send M.	0.31087	1	0.31087	0.01	0.00310	0.01	0.0031
Acquire A.	0.03222	1	0.03222	1	0.03222	0.01	0.00032
Send M.A.	0.31087	1	0.31087	0.0005	0.00016	0.0005	0.00016
Acquire T.	0.00009	1	0.00009	1	0.00009	0.0005	4.46E-08
Send M.A.T.	0.31087	5.0E-5	1.55E-05	5.0E-5	1.55E-05	5.0E-5	1.55E-05
Total Cost:			0.92256		0.30408		0.2721







Ordering of operators

- Given a query execution plan structure, new query execution plans can be obtained reordering operators
- Possible strategies:
  - Selectivity based ordering
  - Cost of acquisition based ordering
  - Cost of transmission (topology) based ordering

Selectivity

QP3:			
Action	**Energy(mJ)**	**Freq.**	**Power**
Acquire M.	0.2685	1	0.2685
Send M.	0.31087	0.01	0.00311
Acquire A.	0.03222	0.01	0.00032
Send M., A.	0.62174	0.0005	0.00031
Acquire T.	0.00009	0.0005	4.46E-08
Send M., A., T.	1.24347	0.00005	6.21E-05
Total Cost:			0.2723

QP4:			
Action	**Energy(mJ)**	**Freq.**	**Power**

However: different results might be obtained with
- different selectivity statistics,
- different acquisition costs,
- and different transmission costs

QP5:			
Action	**Energy(mJ)**	**Freq.**	**Power**
Acquire T.	0.00009	1	0.00009
Send T.	0.31087	0.1	0.03109
Acquire M.	0.2685	0.1	0.02685
Send T., M.	0.31087	0.001	0.00031
Acquire A.	0.03222	0.001	0.00003
Send T., M., A.	0.31087	0.00005	1.55E-05
Total Cost:			0.05838

Outline

- Data management in WSN
- Query processing in WSN
- State of the art
- *Future research directions*

Future research directions

- Increasing query language expressivity
- Identifying significant abstraction levels in the architecture design
- Using distributed storage when needed
- Sharing portions of query plans
- Similarity matching functionalities