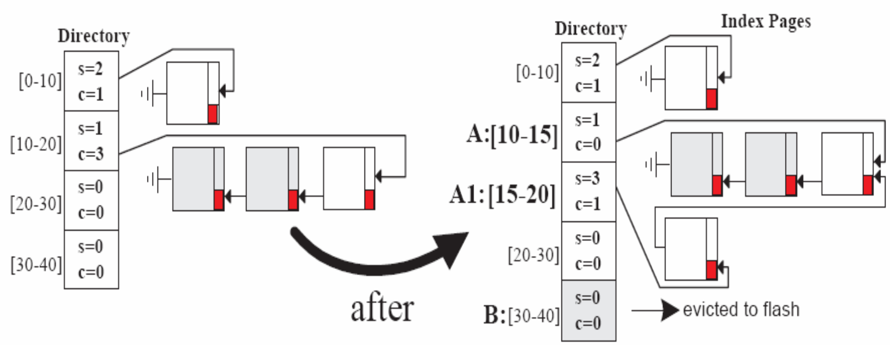
MicroHash: An efficient Index Structure for Wireless Sensor Devices

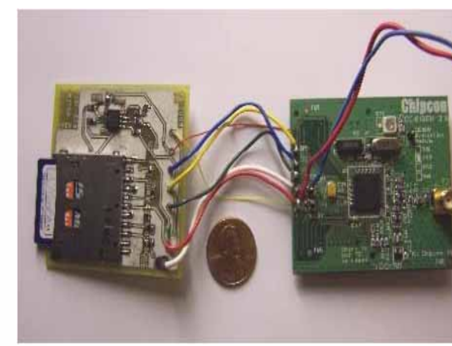
Demetris Zeinalipour

[dzeina@cs.ucy.ac.cy]

Department of Computer Science

University of Cyprus





EPL671 - Computer Science: Research and Technology Course, Dept. of Computer Science, University of Cyprus, Friday, 31st March 2006, Nicosia, Cyprus

http://www2.cs.ucy.ac.cy/~dzeina/

Presentation Goals

- To provide an **overview** of the most important developments in Sensor Network Technology

- To highlight some important **storage and retrieval (database) challenges** that arise in this context

Acknowledgements

- **This is a joint work with my collaborators at the University of California – Riverside.**

- **Our results were presented in the following paper:**

  "MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices",

  D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos and W. Najjar, The 4th USENIX Conference on File and Storage Technologies (FAST'05), San Fransisco, USA, December, 2005.

Talk Outline

1. **Overview of Sensor Networks**

2. Data Storage Models in Sensor Networks

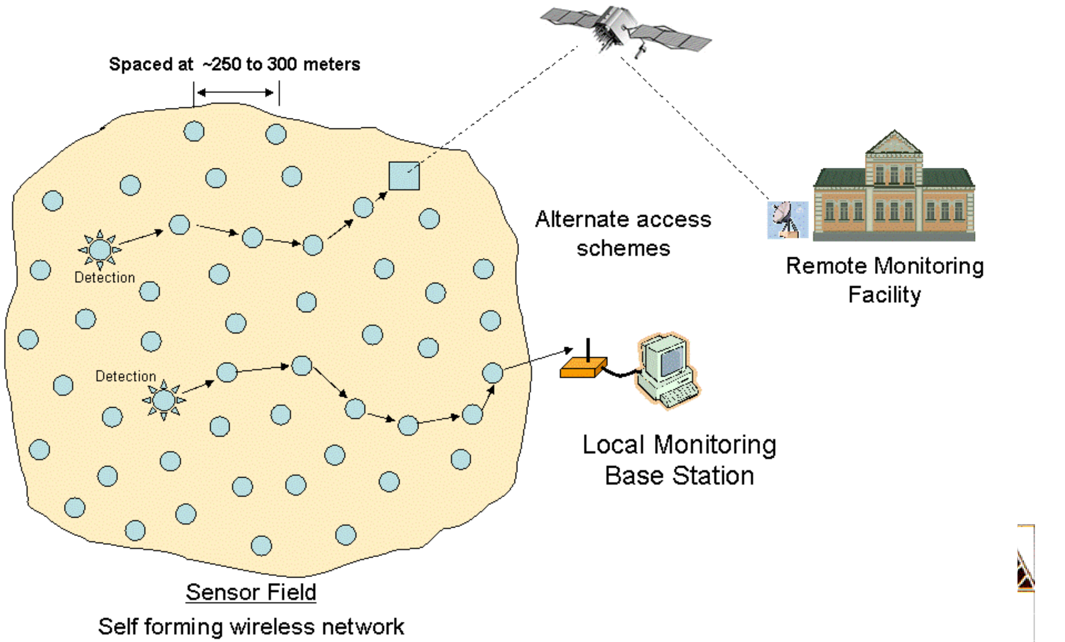
3. The MicroHash Index Structure.

4. MicroHash Experimental Evaluation

5. Conclusions and Future Work

Wireless Sensor Networks (WSNs)

- A collection of resource constrained devices utilized for **monitoring** and **understanding** the physical world.
- 



Sensor Networks Applications

- WSNs offer a **Non-Intrusive** and **Non-Disruptive** technology that enables the human to **study physical phenomena** at **extremely high resolutions.**
- Applications have already emerged in:
  - Environmental and habitant monitoring
  - Seismic and Structural monitoring
  - Understanding Animal Migrations & Species interactions

Monitoring hazards

xbow.com (Automation, Tracking)

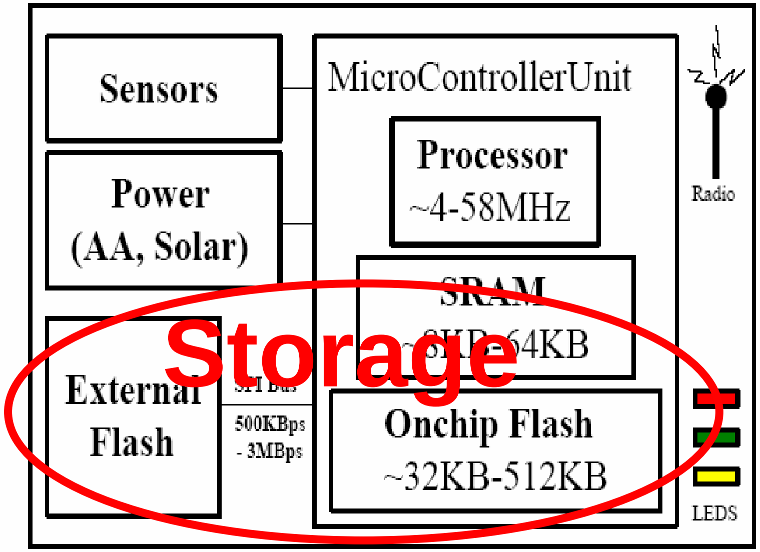
Great Duck Island – Maine (Temperature, Humidity etc).

Golden Gate – SF, Vibration and Displacement of the bridge Structure

Zebranet (Kenya) GPS trajectory

The Anatomy of a Sensor Device

- **Processor,** in various (sleep, idle, active) modes
- **Power source** AA or Coin batteries, Solar Panels
- **SRAM** used for the program code and for in-memory buffering.
- **LEDs** used for debugging
- **Radio,** used for transmitting the acquired data to some storage site (SINK) (9.6Kbps-250Kbps)



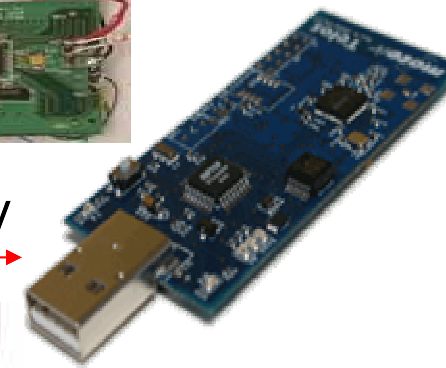
- **Sensors**: Numeric readings in a **limited range** (e.g. temperature -40F..+250F with one decimal point precision) at a **high frequency** (2-2000Hz)

Sensor Devices & Capabilities

Sensing Capabilities

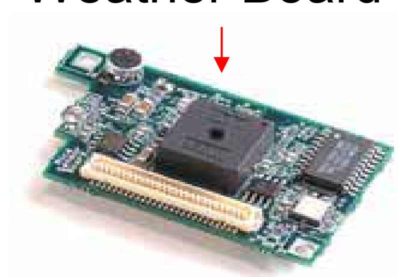
- Light
- Temperature
- Humidity
- Pressure,
- Tone Detection,
- Wind Speed,
- Soil Moisture,
- Location (GPS),
- etc...

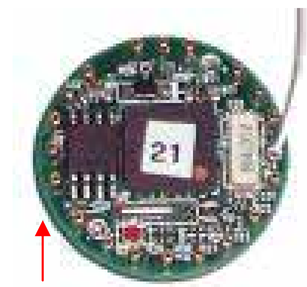
UC-Riverside RISE

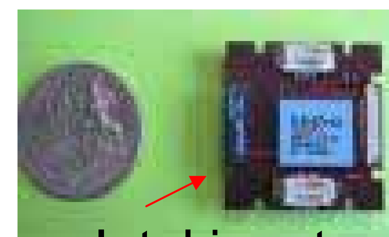


UC-Berkeley Telos

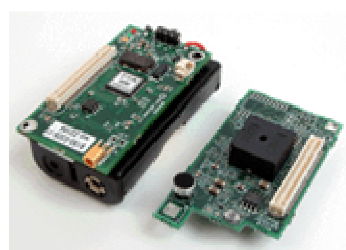
UC-Berkeley Weather Board



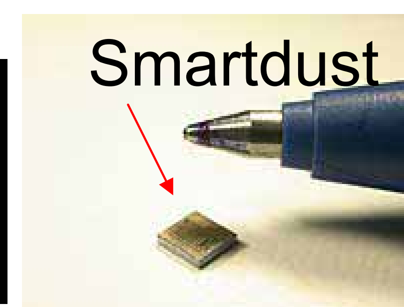




Intel i-mote



UC-Berkeley mica2dot



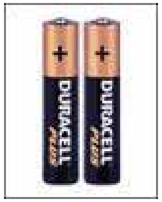
TinyMote 584

Range 2Km

Crossbow Mica Box

Characteristics

1. **Energy Consumption is the critical part.**
   Energy source: AA batteries, Solar Panels



2. **Local Processing** is **cheaper** than **transmitting over the radio**.
   1 Byte over the Radio consumes as much energy as ~1200 CPU instructions.
3. **Local Storage** is **cheaper** than **transmitting over the radio**.
   Transmitting 512B over a single-hop 9.6Kbps (915MHz) radio requires 82,000μJ, while writing to local flash only 760μJ.

Talk Outline

1. Overview of Sensor Networks
2. **Data Storage Models in Sensor Networks**
3. The MicroHash Index Structure.
4. MicroHash Experimental Evaluation
5. Conclusions and Future Work

The Centralized Storage Model

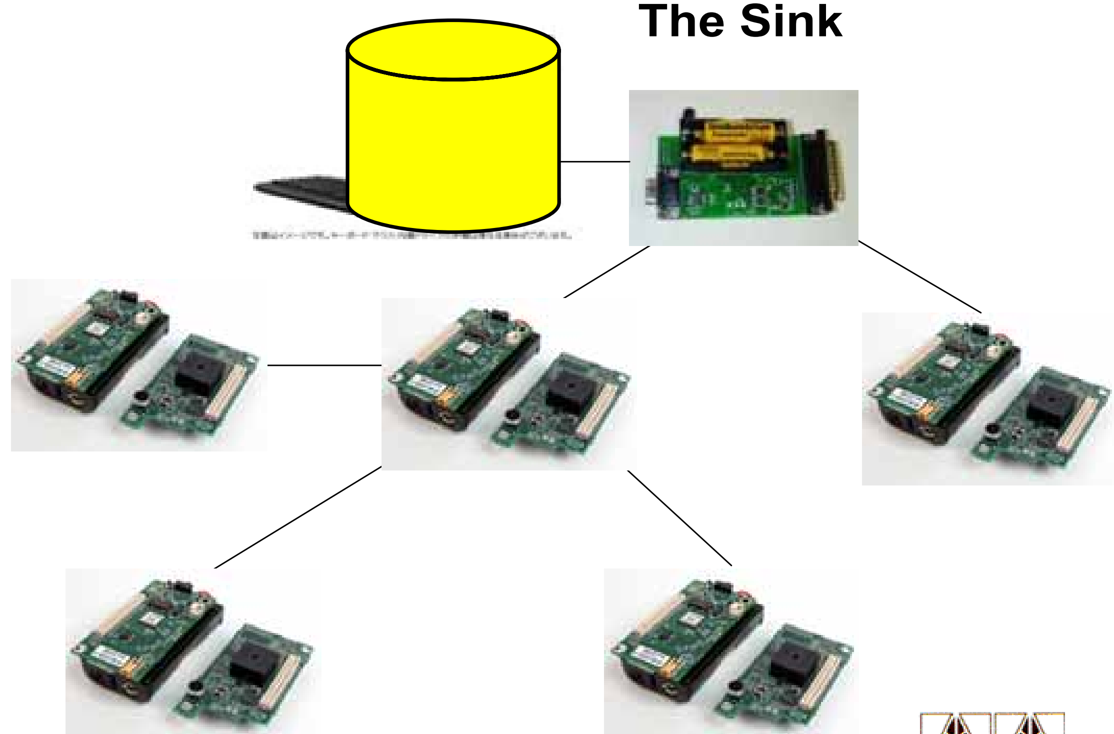
Sense and Send Paradigm

Sensors acquire environmental parameters and transmit these to the sink at pre-specified intervals

A Database that collects readings from many Sensors

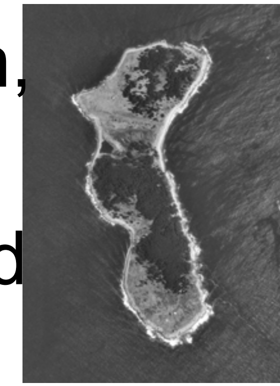
Centralized:

- **Storage, Indexing**
- **Query Processing**
- **Triggers, etc..**



The Centralized Storage Model

The Great Duck Island Study (Maine, USA)

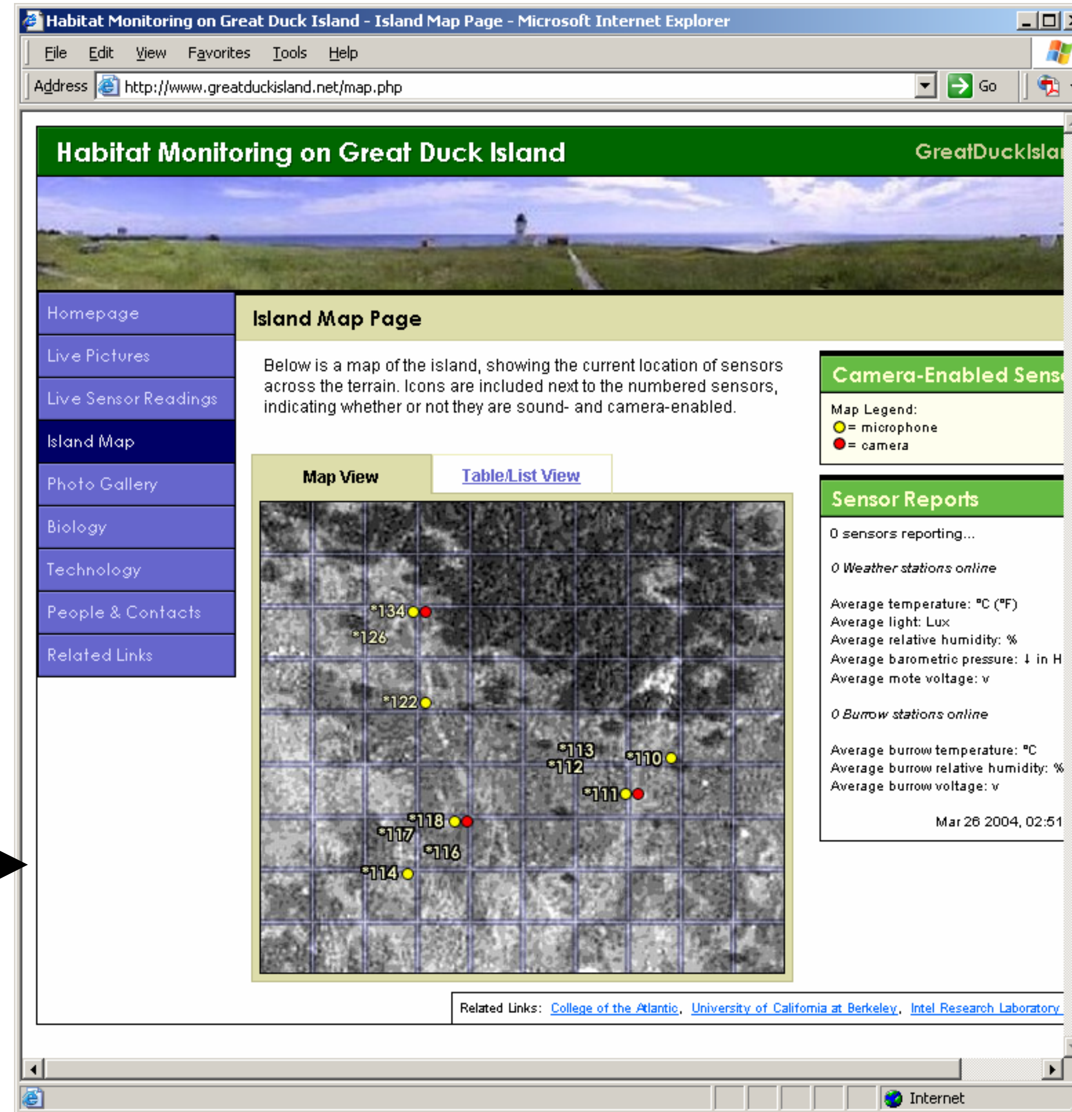


- Large-Scale deployment by Intel Research, Berkeley in 2002-2003 (Maine USA).
- Focuses on monitoring microclimate **in** and **around** the nests of endangered species which are **sensitive to disturbance.**
- They deployed more than 166 motes installed in remote locations (such as 1000 feets in the forest)

The Centralized Storage Model

Real Time Monitoring

WebServer



The Centralized Storage Model

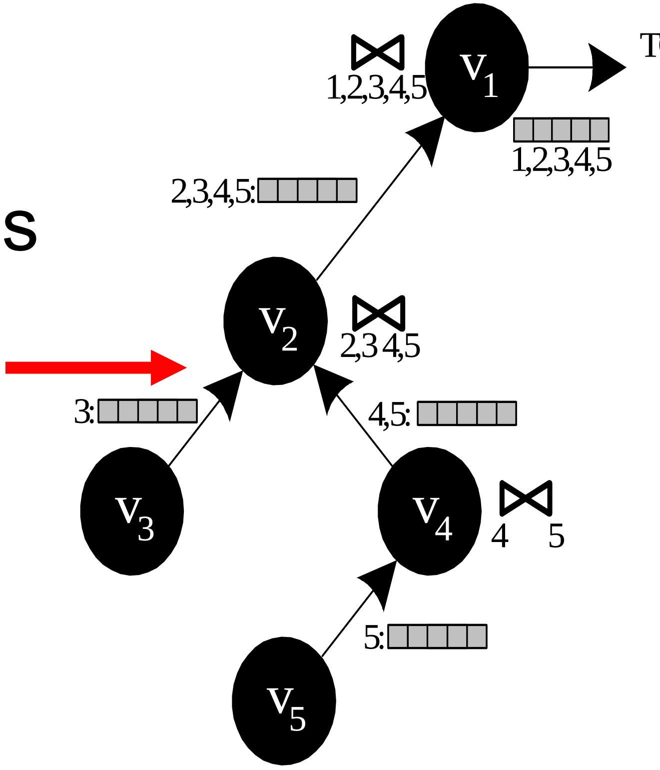
The James Reserve Project, CA, USA

Sensor	24hr trend	Current value (if available)
Barometer	[...]	1012.40 mBar
Dew Point	[...]	-2 C
Humidity	[...]	98 %
Leaf Wetness	[...]	0.000000 0-10 dry-wet
Lightning	[...]	0.000000 Cnt
Rain	[...]	0.00 mm
Solar Radiation	[...]	0.000000 W/sqM
Temperature	[...]	-2.2 C
Wind Direction	[...]	67 Degrees
Wind Speed	[...]	0.0 km/hr

Centralized Storage & Query Processing

- All the pre-mentioned projects utilize the **Centralized** (Sense and Send) **Model**.
- **Although Query Aggregation** is **pushed in the network** (e.g. with TinyDB/TAG or Directed Diffusion), still **each and every event is percolated to a centralized database**.
- Transmitting over the radio is **extremely expensive**.

Tree-Based Routing



e.g. Sum, Max, Min, Count

Our Model: In-Situ Data Storage

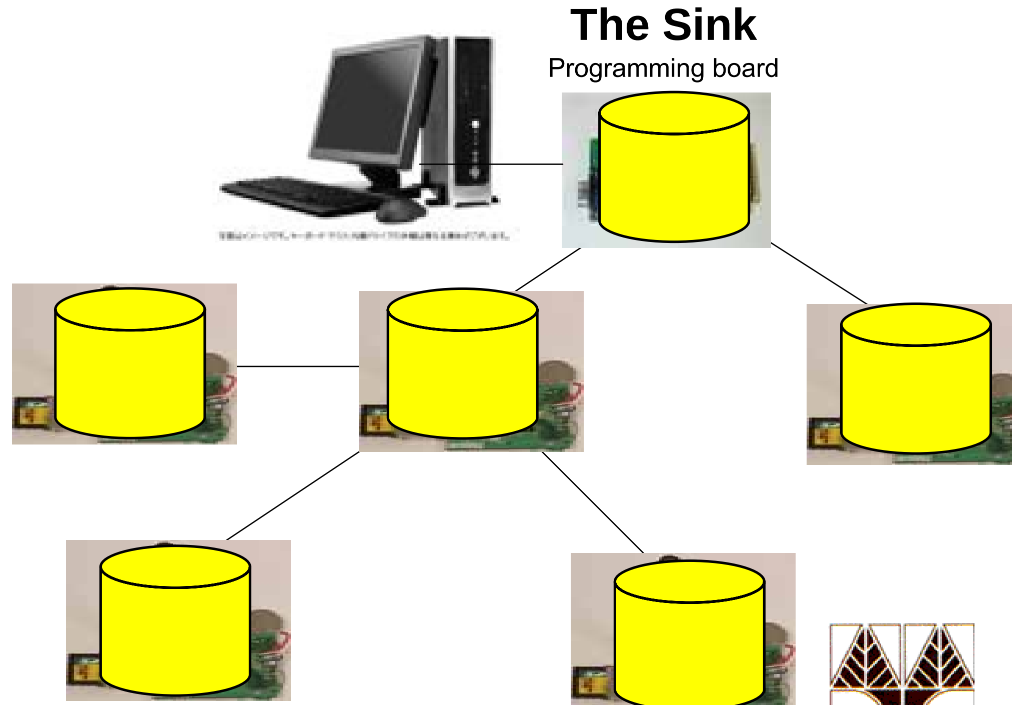
1. **Sensors acquire readings from their surrounding environment.**
2. **The data remains In-situ (at the generating site) in a sliding window fashion.**
3. ***When Users want to search/retrieve some information they perform optimized on-demand queries.***

A network of

Sensor Databases

- **Distributed Storage**
- **Distributed Query Processing**

Objective: To minimize the utilization of the radio

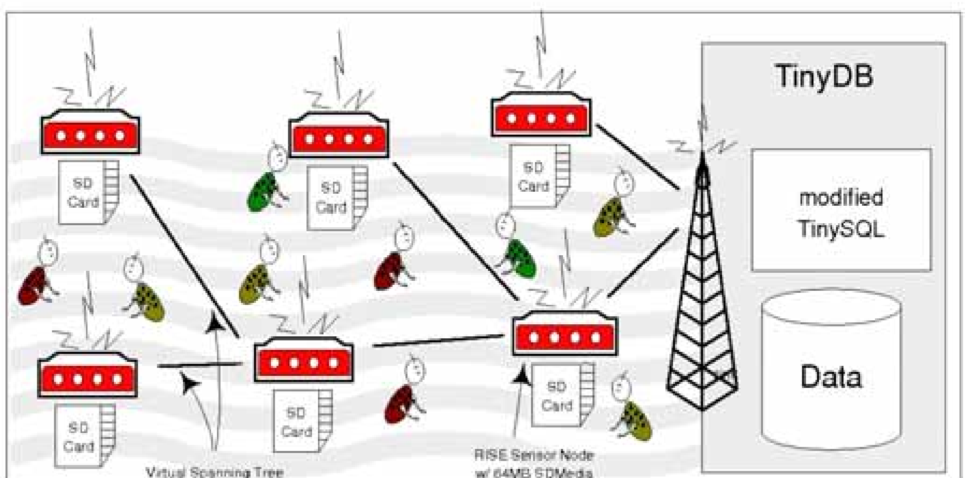


In-Situ Data Storage: Motivation

Soil-Organism Monitoring

(Center for Conservation Biology, UCR)

- A set of sensors monitor the CO_2 levels in the soil over a large window of time.
- Not a real-time application.
- Many values may not be very interesting.



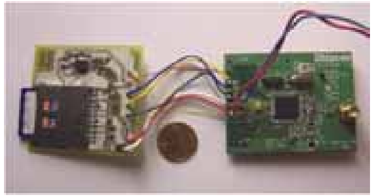
D. Zeinalipour-Yazti, S. Neema, D. Gunopulos, V. Kalogeraki and W. Najjar, **"Data Acquision in Sensor Networks with Large Memories",** IEEE Intl. Workshop on Networking Meets Databases NetDB (ICDE'2005), Tokyo, Japan, 2005.

Challenges of the In-Situ Model

- **How to efficiently store information locally**

  Solution: Our group built the RISE Sensor that features an external flash memory)

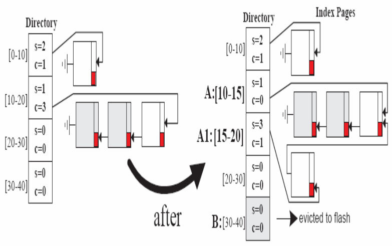
  [IEEE/ACM IPSN’05, IEEE SECON’05, ACM Senmetrics’05]



- **How to efficiently access a Giga-Scale storage medium of a Sensor Device?**

  Solution: We build the MicroHash Index Structure

  [IEEE NetDB (ICDE’05), USENIX FAST’05]



- **How to find the most important events without pulling together all distributed relations?**

  Solution: We build the Threshold Join Algorithm

  [IEEE DMSN’05 **(**VLDB’05)]

Talk Outline

1. Overview of Sensor Networks
2. Data Storage Models in Sensor Networks
3. **The MicroHash Index Structure**
4. MicroHash Experimental Evaluation
5. Conclusions and Future Work

MicroHash

Objective

- Provide **efficient access to any record** stored on flash by timestamp or value
- Execute a **wide spectrum of queries** based on our index, similarly to generic DB indexes.

Requirements:

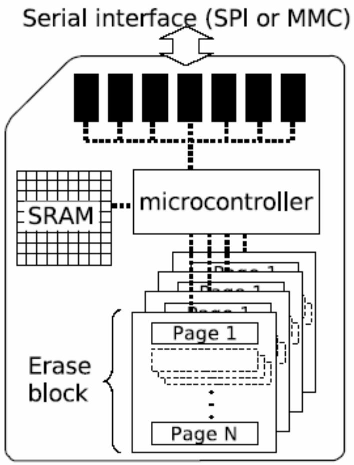
- Minimize the size of SRAM-structures. (only 2-64KB is available).
- Address the distinct characteristics of Flash Memory in order to minimize energy consumption and increase lifetime

A) Flash Memory at a Glance

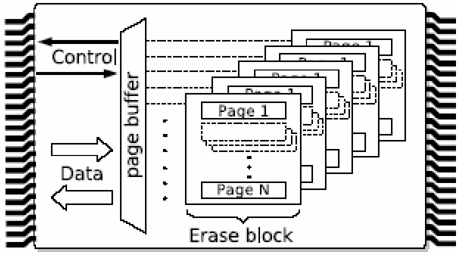
- The most prevalent storage medium used for Sensor Devices is **Flash Memory** (NAND Flash)
- The fastest growing memory market $8.7B (Micron.com)

Flash (NAND) Advantages

- Simple Cell Architecture (high capacity in a small surface)
- Economical Reproduction
- Shock Resistant
- Fast Random Access (50-80 μs)
- Power Efficiency



Removable Devices



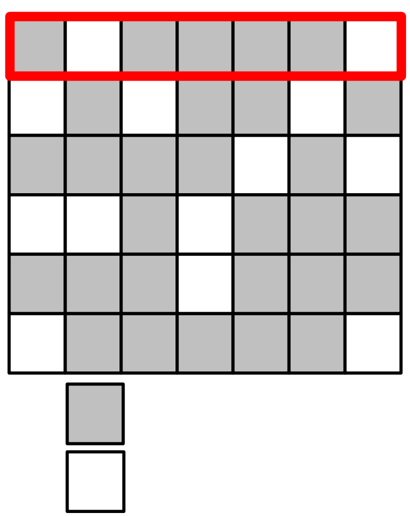
Surface mount NAND flash

A) Flash Memory at a Glance

1. **Delete-Constraint:** Deleting can only be performed at a block granularity (i.e. 8KB~64KB)

2. **Write-Constraint:** Writing data can only be performed at a page granularity (256B~512B), after the respective page (and its respective 8KB~64KB block!) has been deleted

3. **Wear-Constraint:** Each page can only be written a limited number of times (typically 10,000-100,000)



Measurements using RISE

NAND Flash installed on a Sensor Node			
Page = 512B	Page **Read** 1.17mA	Page **Write** 37mA	Block **Erase** 57mA
Time	6.25ms	6.25ms	2.26ms
Data Rate	82KBps	82KBps	7MBps
Energy	$24\mu J$	$763\mu J$	$425\mu J$

<u>Asymmetric Read/Write Energy Cost</u> : Writing is **3 orders of magnitudes** more expensive than **Reading**

Summary of Our Objectives

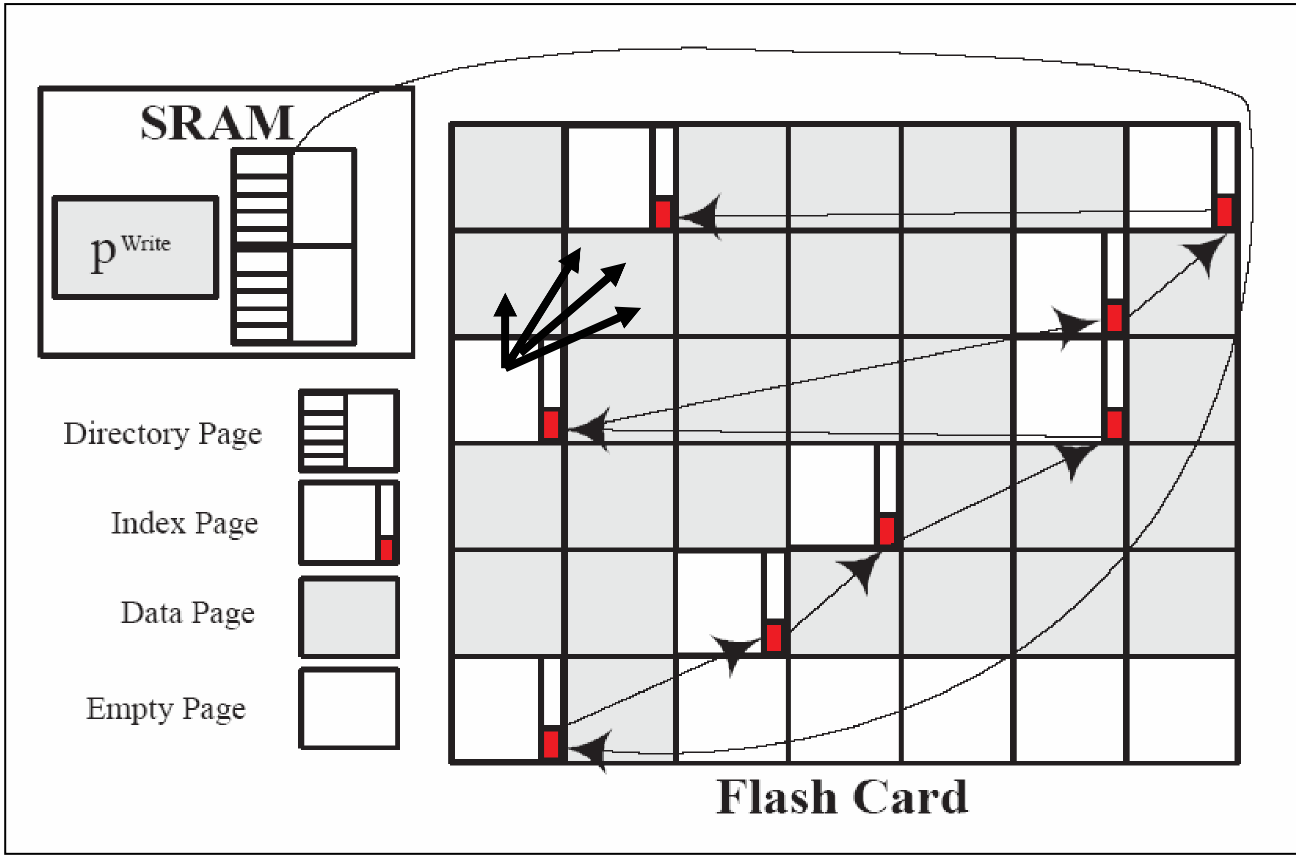
- **Maximize Wear-Leveling:** Spread page writes out uniformly across the storage media in order to avoid wearing out specific pages.

- **Minimize Block-Erase Operations:** by minimizing *random access deletions*.

- **Minimize SRAM structures:** because we have limited memory and require fast initialization.

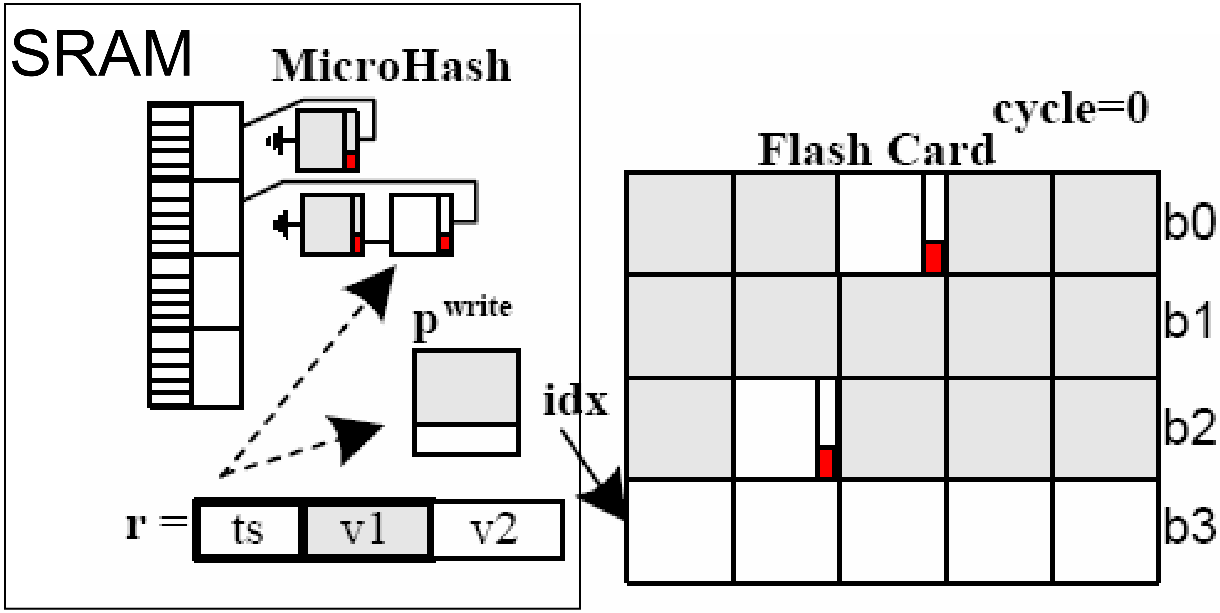
MicroHash Overview

- 4 types of pages
  - Root Page
  - Directory Page
  - **Index Page**
  - **Data Page**
- 4 operation phases
  - a) Initialization
  - b) Growing
  - c) Repartition
  - d) Deletion



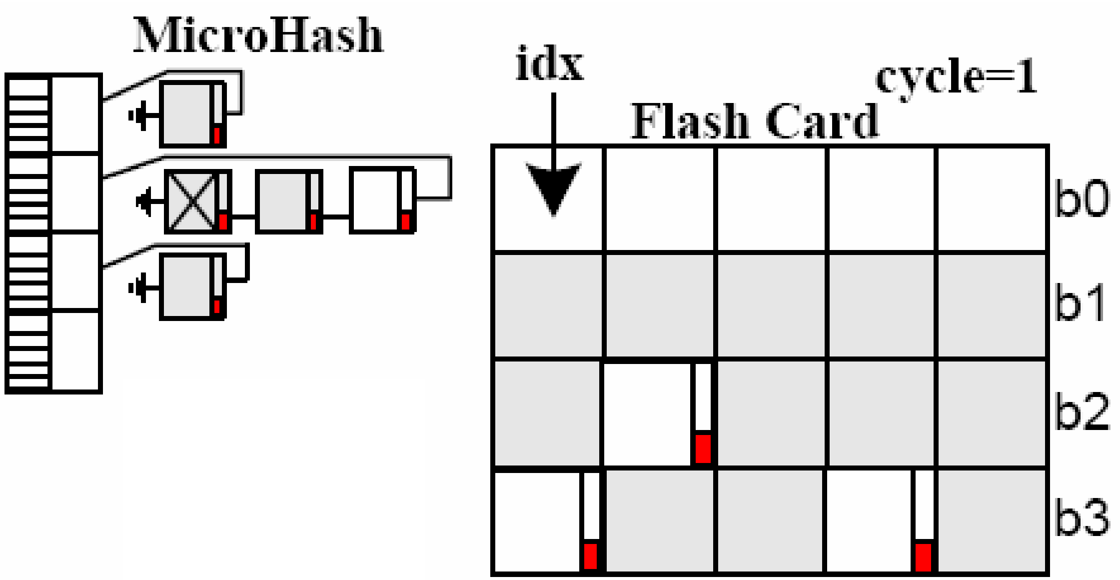
Operations in MicroHash: Insertion

- **A) Growing Phase**
  - Collect data and fill up data buffer page P^{write} in *SRAM*.
  - Then force P^{write} out to flash media.
  - Create index records for each data record in P^{write}.
  - If *SRAM* is too small to hold the new generated index records, Index pages are forced out by *LRU*.



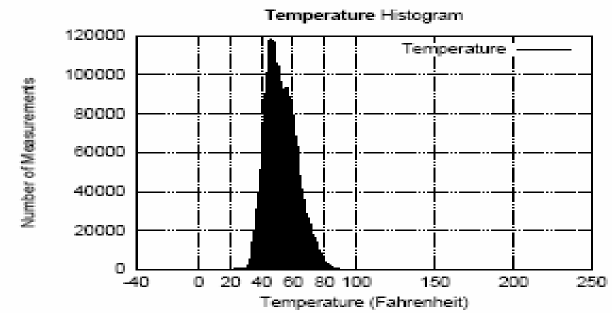
Operations in MicroHash: Deletion

- **B) Deletion Phase**
  - Take the flash media as a circular array and keep a pointer at the next writing position (idx).
  - If we want to write and the flash media is full, delete the next block pointed by the idx pointer



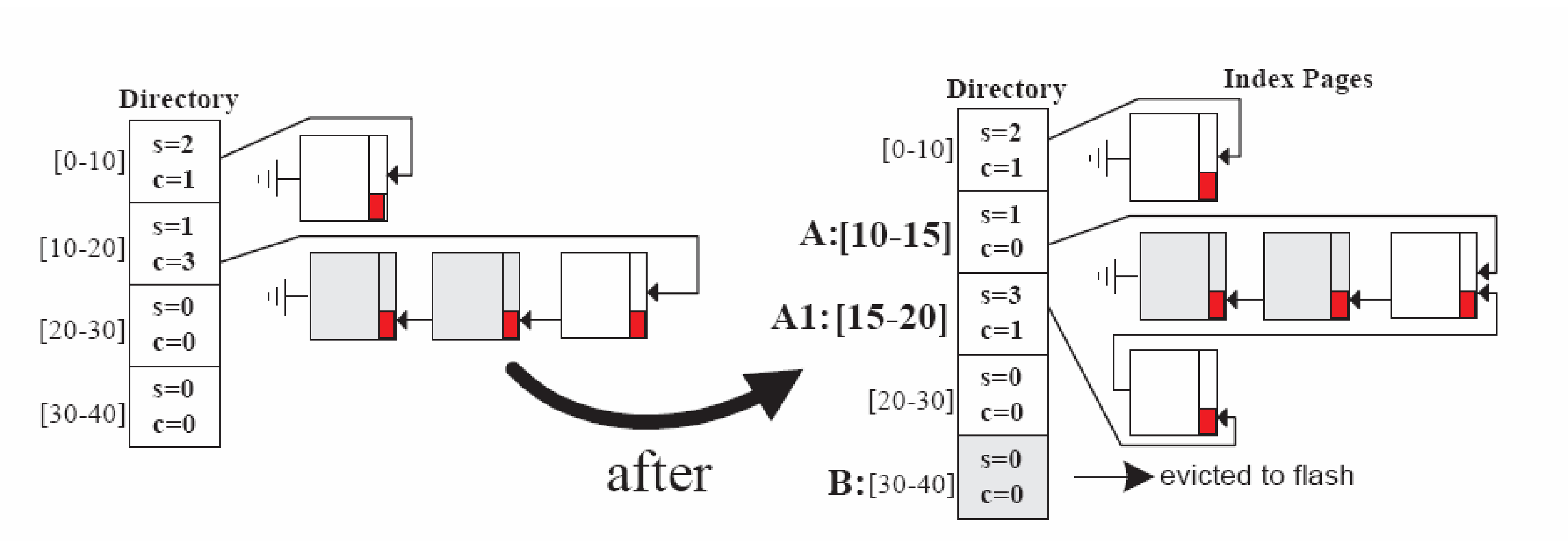
Operations in MicroHash: Repartition

- MicroHash starts out with a Equi-width bucket table
- Equi-width bucket splitting deteriorates under biased data.
- We want to obtain finer intervals for the buckets utilized most.



Splitting policy:

- If bucket A links to more than τ index records, evict the **least used** bucket B and segment bucket A into A and A’
- No bucket reassignments of old records => Expensive



Searching in MicroHash

- Searching by value
  "Find the ***timestamp (s)*** *on which the temperature was 100F"*
  - Simple operation in MicroHash
  - We simply find the right Directory Bucket, from there the respective index page and then data record (page-by-page)

- Searching by timestamp
  "Find the ***temperature*** *of some sensor at some time instance tj (or in the range [tj..tk])"*
  - **Problem:** Index pages are mixed together with data pages.
  - *How can we search by timestamp if pages are mixed?*
    1. Binary Search (O(log(n)) ~20 pages for 512MB flash media)
    2. *LBSearch* (less than 10 pages)
    3. *ScaleSearch* (better than *LBSearch, ~4.5 pages*)

LBSearch and ScaleSearch

Solutions to the Search By Timestamp Problem:

A) LBSearch: *We recursively create a lower bound on the position of tq until tq is located.*

Idea: *Fetch page at* ***tq (the lower bound)****, denoted as* ***P****. If* ***P*** *contains* ***tq*** *terminate, else extract the last known timestamp in that page and recursively refine the lower bound until tq is located.*

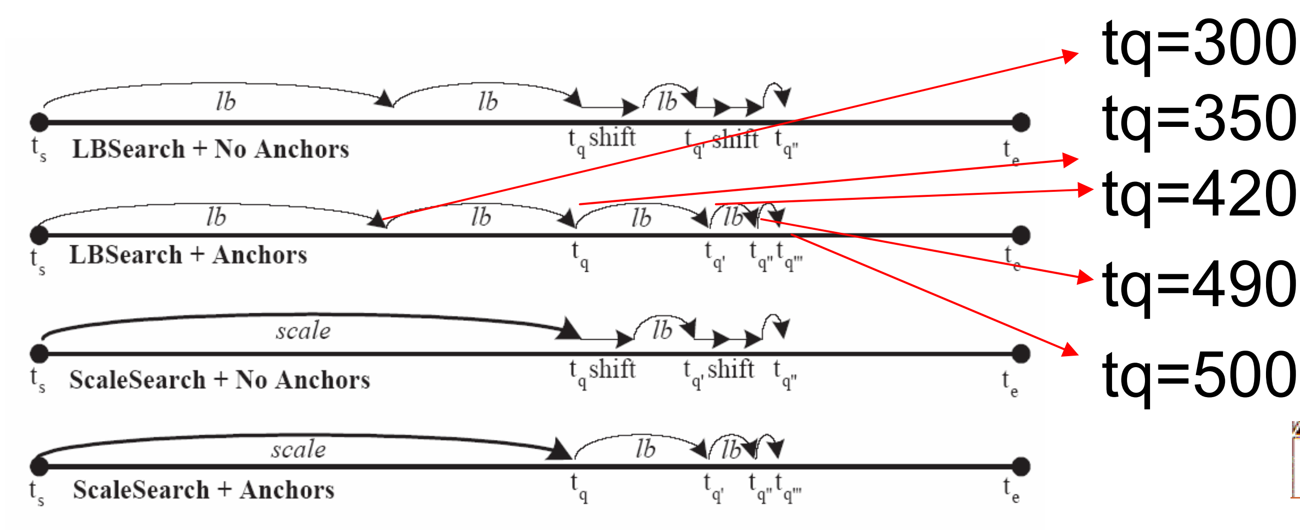
B) ScaleSearch:

Idea: *Quite similar to LBSearch, however in the first step we position the read more intelligently (by exploiting data distribution)*

Query

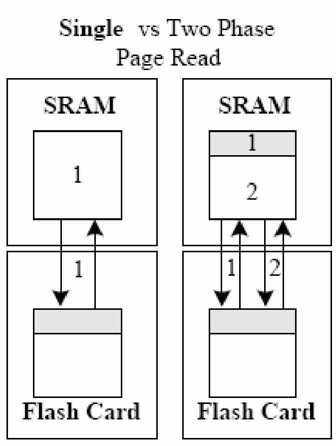
tq=500

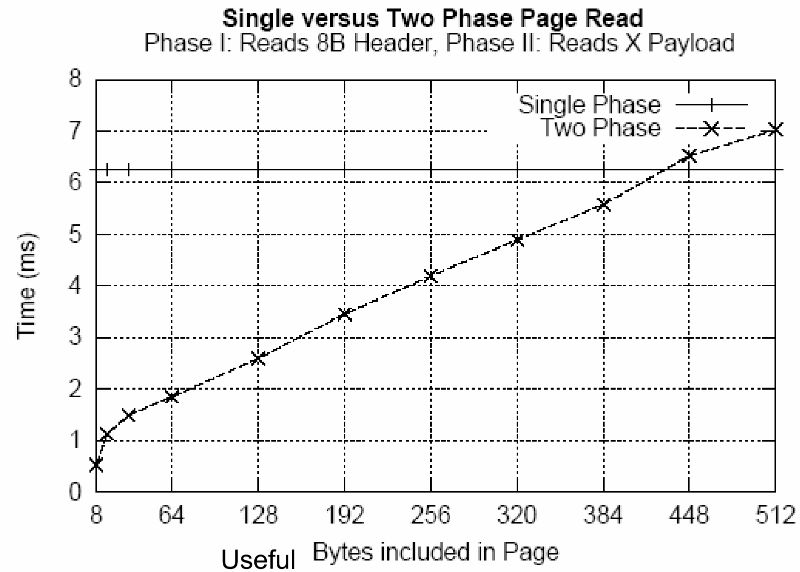
in practice 4.75 page reads



Two-Phase Page Reads

- **Problem**
  - ***Index Pages*** *written on flash might not be fully occupied*
  - When we access these pages we transfer a lot of **empty bytes (padding)** between the flash media and SRAM.
- **Our Solution 1: Two-Phase Page Reads**
  - Reads the **8B header** from flash in the first phase, and then reads the exact amount of bytes in the next phase.



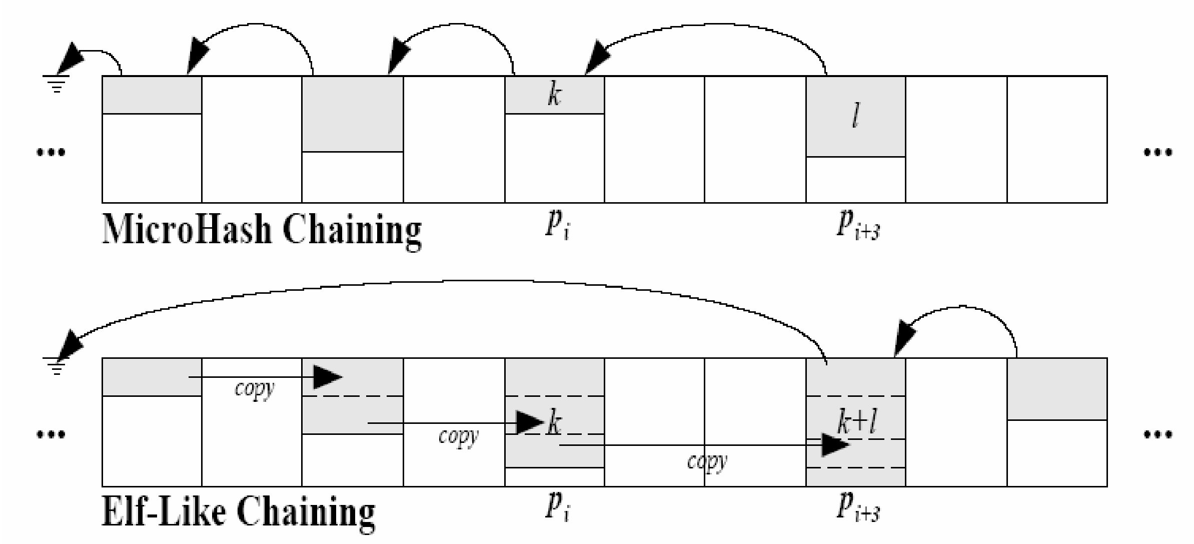


MicroHash vs ELF

- **Solution 2:** Avoid non-full index pages using ELF*.

ELF:

  - a linked list in which each page, other than the last page, is completely full.
  - keeps copying the last non-full page into a newer page, when new records are requested to be added.



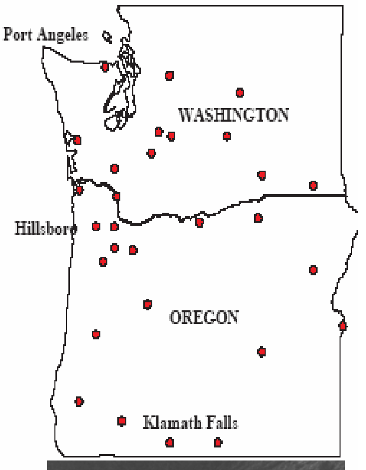
Dai et. al., Efficient Log Structured Flash File System, SenSys 2004

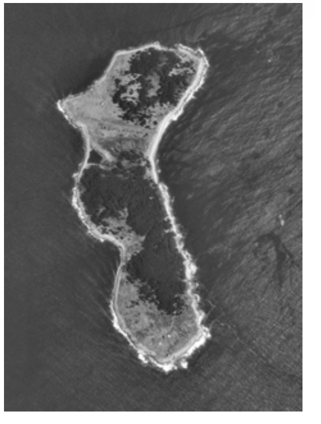
Talk Outline

1. Overview of Sensor Networks
2. Data Storage Models in Sensor Networks
3. The MicroHash Index Structure
4. **MicroHash Experimental Evaluation**
5. Conclusions and Future Work

Experimental Evaluation

- Implemented MicroHash in nesC.
- We tested it using TinyOS along with a trace-driven experimental methodology.
- **Datasets**:
  - **Washington State Climate**
    - 268MB dataset contains readings in 2000-2005.
  - **Great Duck Island**
    - 97,000 readings between October and November 2002.



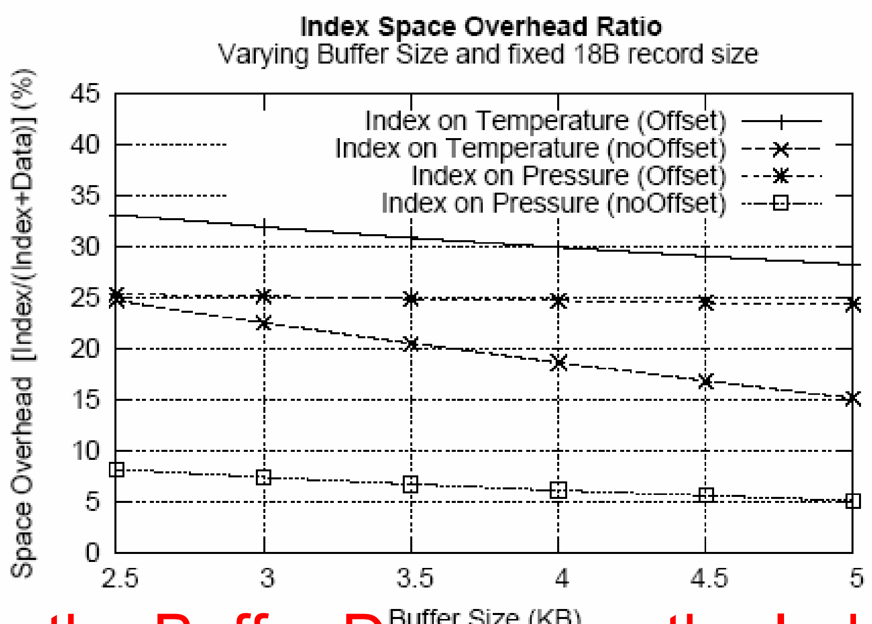


- **Evaluation Parameters**: i) Space Overhead, ii) Energy Overhead, iii) Search Performance

1) Space Overhead of Index

- Index page overhead Φ = IndexPages/(DataPages+IndexPages)
- Two Index page layouts
  - ***Offset*****,** an index record has the following form {datapageid,offset}
  - ***NoOffset*****,** in which an index record has the form {datapageid}
- 128 MB flash media (256,000 pages)
  - varying SRAM (buffer) size (2.5 - 5KB)

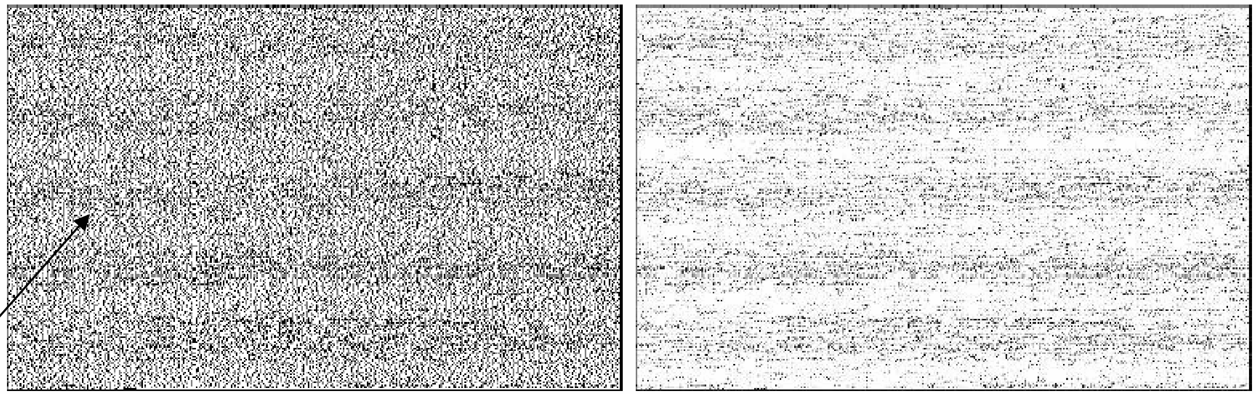
    [same applies to record size(10– 22 Bytes)]



Increasing the Buffer Decreases the Index Overhead

1) Space Overhead of Index

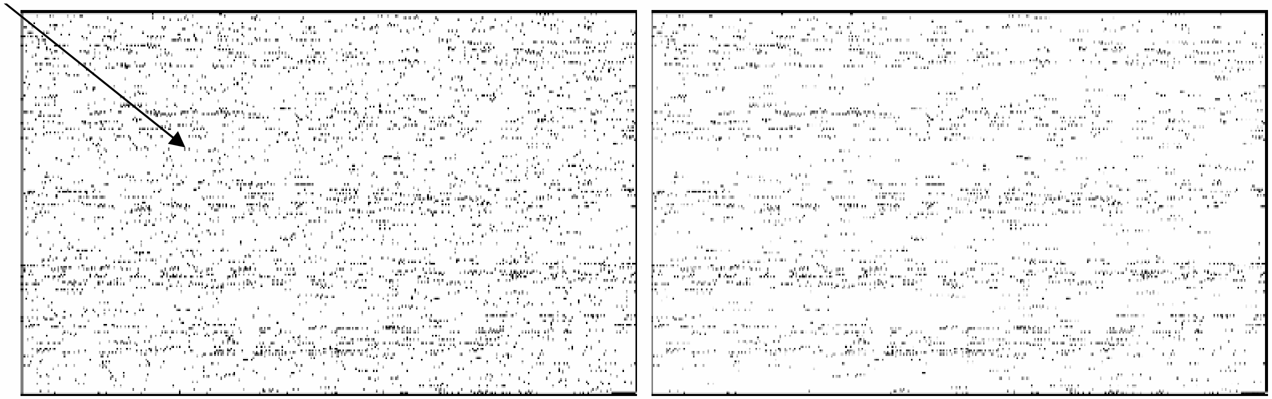
Increasing the Buffer Decreases the Index Overhead



Index/Data Pages (left) — Grayscale Occupancy (right)

2.5K Buffer

Black denotes the index pages

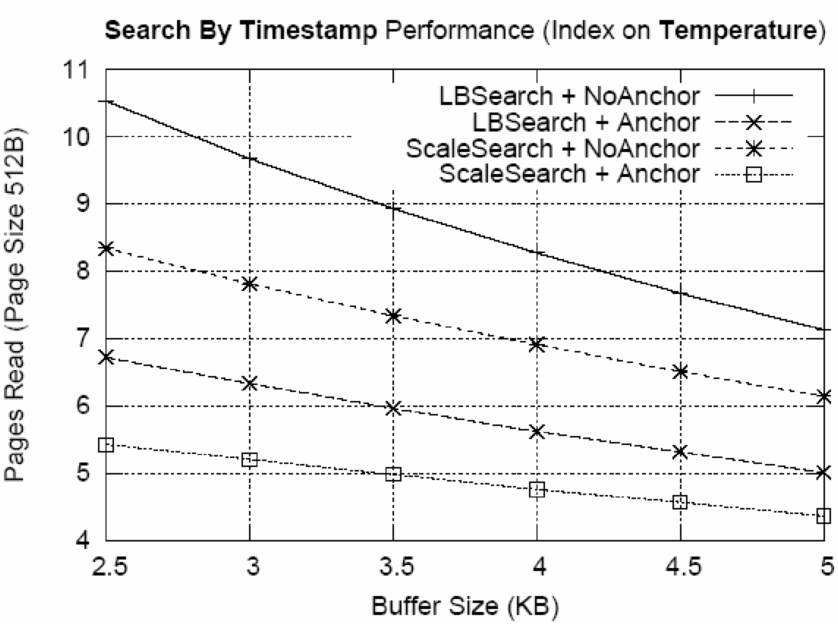


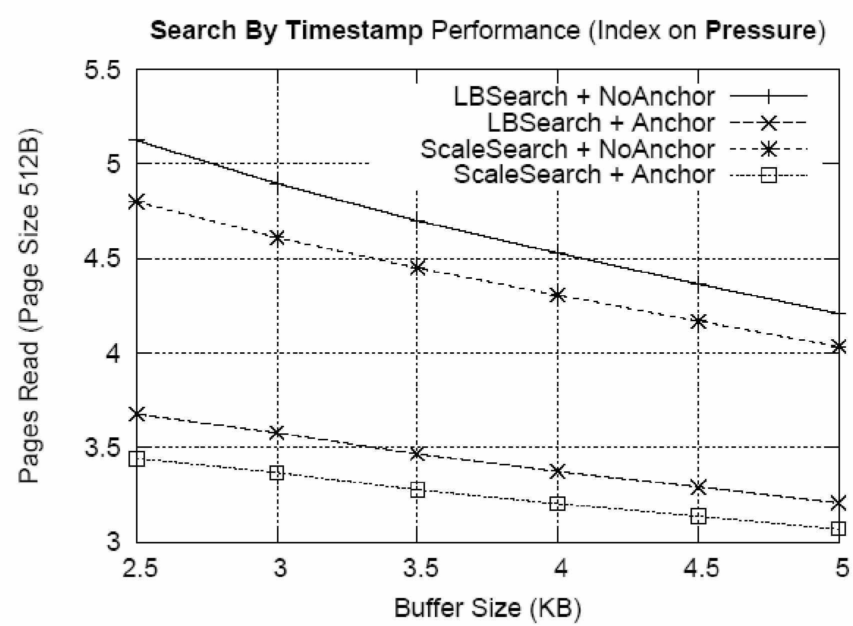
Index/Data Pages (left) — Grayscale Occupancy (right)

10K Buffer

2) Search Performance

- 128 MB flash media (256,000 pages), varied SRAM (buffer) size
- 2 Index page layouts
  - *Anchor*, every index page stores the last known data record timestamp
  - *No Anchor*, the index page does not contain any timestamp information

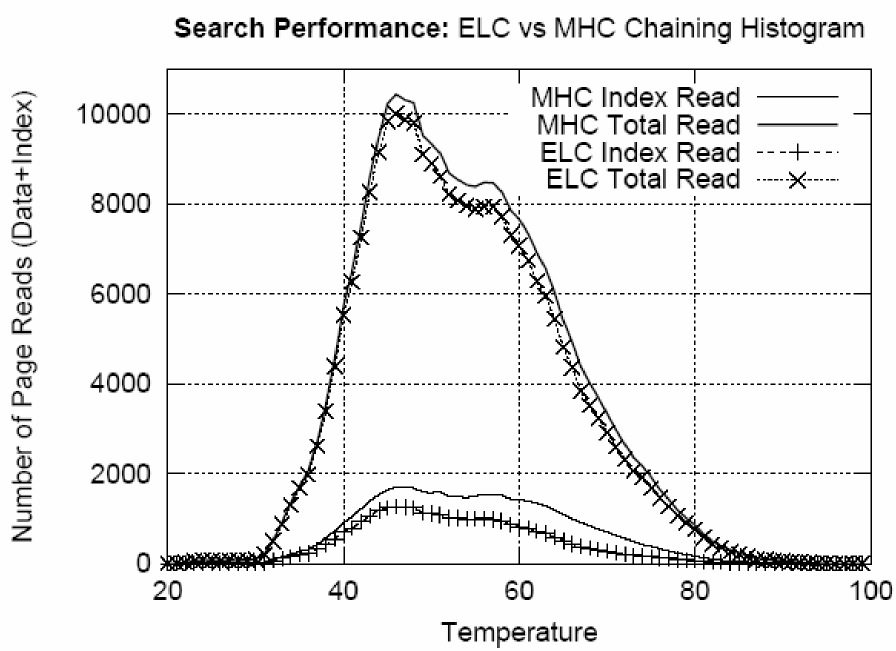




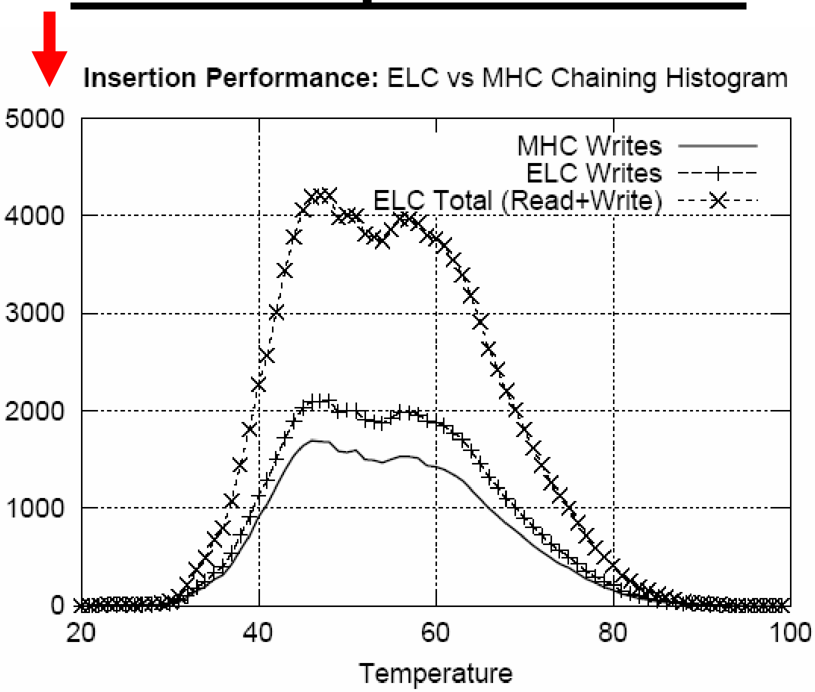
+ Searching by Timestamp can be performed efficiently + Increasing the Buffer (during indexing) Increases Search Performance

2) Search Performance

- We compared MicroHash vs. ELF Index Page Chaining.
- Keeping full index pages increases **<u>search performance</u>** but **decreases <u>insertion performance</u>.**



Increasing search performance using ELF (10% less reads)



Decreasing indexing performance using ELF (15% more writes)

Indexing on Great Duck Island Trace

- Used 3KB index buffer and a 4MB flash card to store all the 97,000 20-byte data readings.
  - The index pages never require more that **30%** additional space
  - Indexing the records has only a small increase in energy demand: the energy **cost of storing the records on flash without an index is 3042mJ**
  - We are able to find any record by its timestamp with **4.75 page reads on average**

Index On Attribute	Overhead Ratio Φ %	Energy Index (mJ)	ScaleSearch Page Reads
Light	26.47	4,134	4.45
Temperature	27.14	4,172	5.45
Thermopile	24.08	4,005	6.29
Thermistor	14.43	3,554	5.10
Humidity	7.604	3,292	2.97
Voltage	20.27	3,771	4.21

Talk Outline

1. Overview of Sensor Networks
2. Data Storage Models in Sensor Networks
3. The MicroHash Index Structure
4. MicroHash Experimental Evaluation
5. **Conclusions and Future Work**

Conclusions

- We Proposed the ***MicroHash* index**, which is an efficient external memory hash index that addresses the distinct characteristics of flash memory

- Our experimental evaluation shows that the structure we propose is both efficient and practical

- This is a new area with many new challenges and opportunities!

http://www2.cs.ucy.ac.cy/~dzeina/

Future Work

- Indexing **multidimensional datasets**
- Exploiting **Temporal Locality** along with Compression Algorithms to minimize even further the storage cost.
- Realize the ***In-Situ Data Storage and Retrieval system*** which binds together all the aforementioned ideas.

MicroHash: An efficient Index Structure for Wireless Sensor Devices

Demetris Zeinalipour

Thank you!

http://www2.cs.ucy.ac.cy/~dzeina/