UNIVERSITY OF CALIFORNIA, RIVERSIDE

MicroHash: An efficient Index Structure for Flash-Based Sensor Devices

Demetrios Zeinalipour-Yazti

dzeina@cs.ucy.ac.cy

Dept. of Computer Science, Univ. of Cyprus

Song Lin

Vana Kalogeraki

Dimitrios Gunopulos

Walid A. Najjar

{slin,vana,dg,najjar}@cs.ucr.edu

Computer Science and Engineering Dept.

University of California, Riverside

The Ubiquitous Silicon Era

- Applications:
  - Environmental and habitat monitoring
  - Seismic and Structural monitoring, ….
- Effect:
  - Sense the environment at very high resolutions.
- Is this just the beginning?

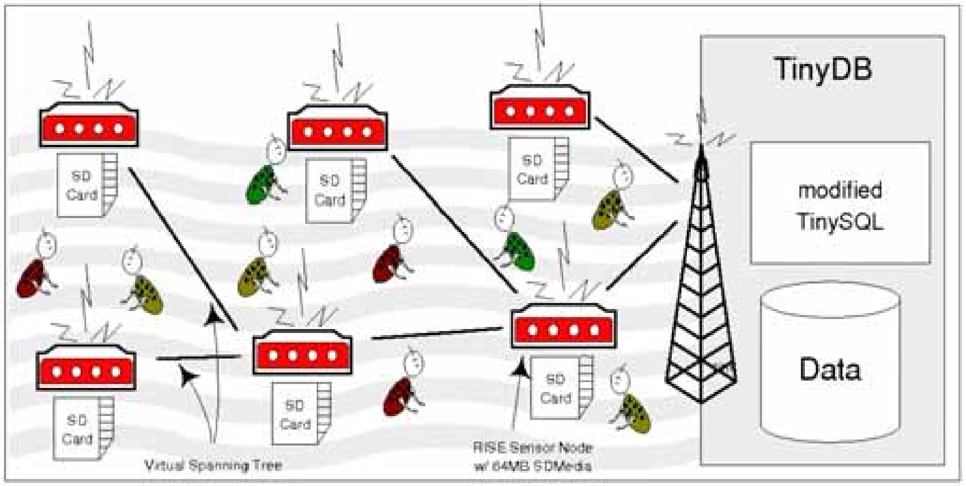
Environmental Monitoring

Structural Monitoring

Wildlife Tracking: GPS Collars

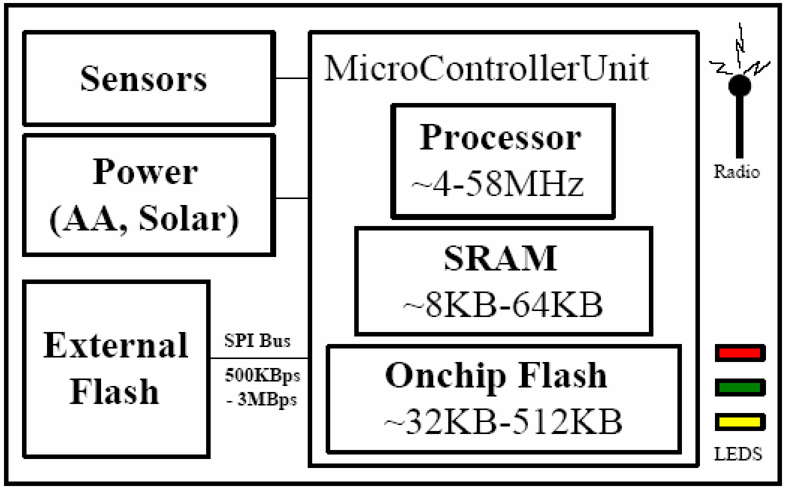
Sensor Network Applications

- Soil-Organism Monitoring
  (Center for Conservation Biology, UCR)
  - A set of sensors monitors the CO_2 levels over a large window of time
  - Not a real-time application
  - Many values may not be very interesting



The RISE Platform

- Motivated by the Center of Conservation Biology, UCR, requirements
- System-on-a-chip (Chipcom CC1010):
  - 24MHz Processor
  - 8KB RAM
  - 32KB Flash
  - 76.8Kbits/sec RF
- SD-Card Interface
- Temperature and CO_2 sensors



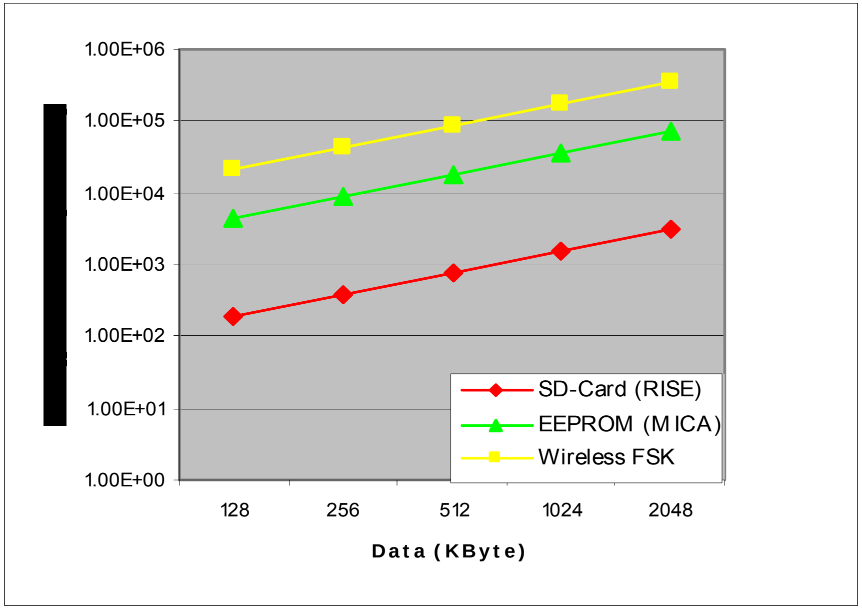
Computing in a Sensor Network

- **Frameworks such as TinyDB, TAG, and Cougar:**
  - Provide a declarative SQL-like approach for accessing data.
  - Are suitable for continuous queries.
  - Push aggregation in the network but keep much of the processing at the sink.
- **New Challenges**:
  - Efficient Query Processing Algorithms that exploit the Hierarchical Structure of Sensor Networks
  - Many applications do not require the query to be evaluated continuously (e.g. Average temperature in the last 6 months?).
  - Local Storage in the future might increase e.g. RISE features an external SDMedia card (up to 4GB!)

Computing in a Sensor Network

An interesting query: “Which 5 time instances had the highest average in the last 15 hours?”

- We need to join the logs from all sensors to answer
- But we can we do better than transmitting and joining everything (using distributed Top-K algorithms)
- Local storage, and local processing is less expensive than shipping data over the network

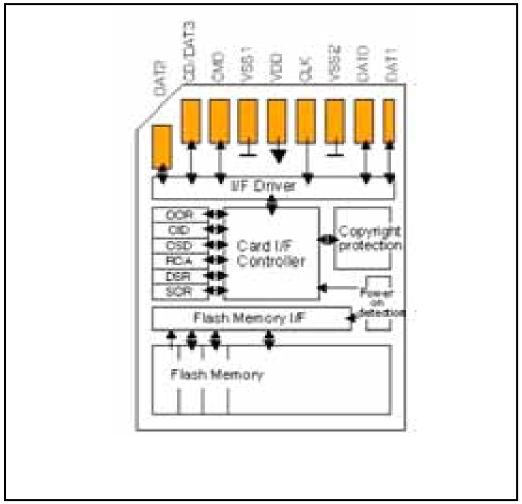


Storage for Sensor Nodes

- Small sensor nodes (with system on a chip architectures) have limited local memory
- But recent designs provide external flash memory modules
- We need to design efficient flash memory index structures to support efficient searches or more sophisticated queries:
  - "Find the reading of sensor S_i at time instance t_j"
  - "When was the reading of sensor S_i equal to T?"
  - "Which are the k time instances that the readings of the sensor S_i were the highest?"
- The design must take the medium characteristics into account

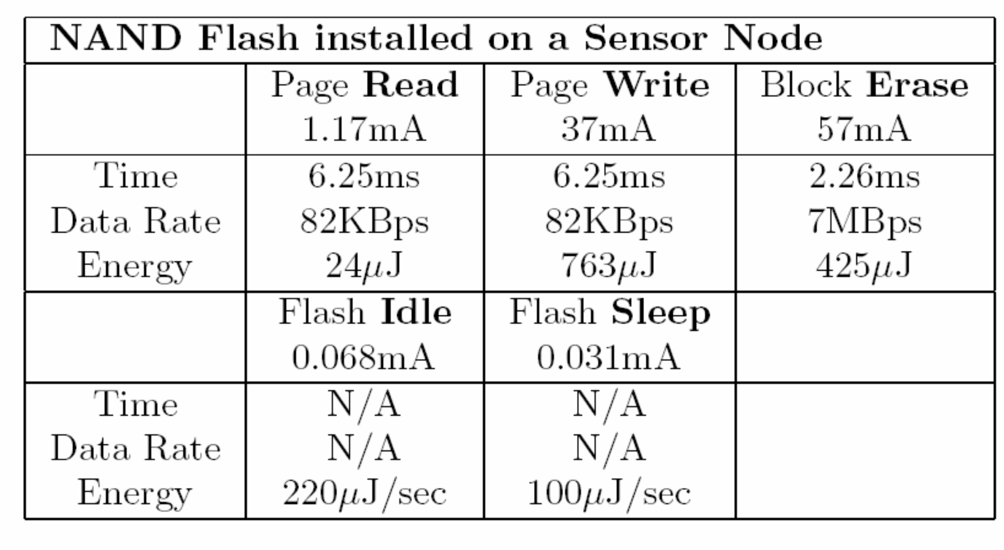
NAND Flash Memory

- The most prevalent storage media:
  - – Non-volatile storage memory
  - – Fast read access and power efficiency
  - – Simple cell architecture, allows for economical production
  - – Commercial off the shelf availability in compact packaging



NAND Flash Memory Constraints

- **Delete-Constraint:** Deleting can only be performed at a block granu-larity (i.e. 8KB~64KB)
- **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
- **Wear-Constraint:** Each page can only be written a limited number of times (typically 100,000)



NAND Flash installed on a Sensor Node			
	Page **Read** 1.17mA	Page **Write** 37mA	Block **Erase** 57mA
Time Data Rate Energy	6.25ms 82KBps 24μJ	6.25ms 82KBps 763μJ	2.26ms 7MBps 425μJ
	Flash **Idle** 0.068mA	Flash **Sleep** 0.031mA	
Time Data Rate Energy	N/A N/A 220μJ/sec	N/A N/A 100μJ/sec	

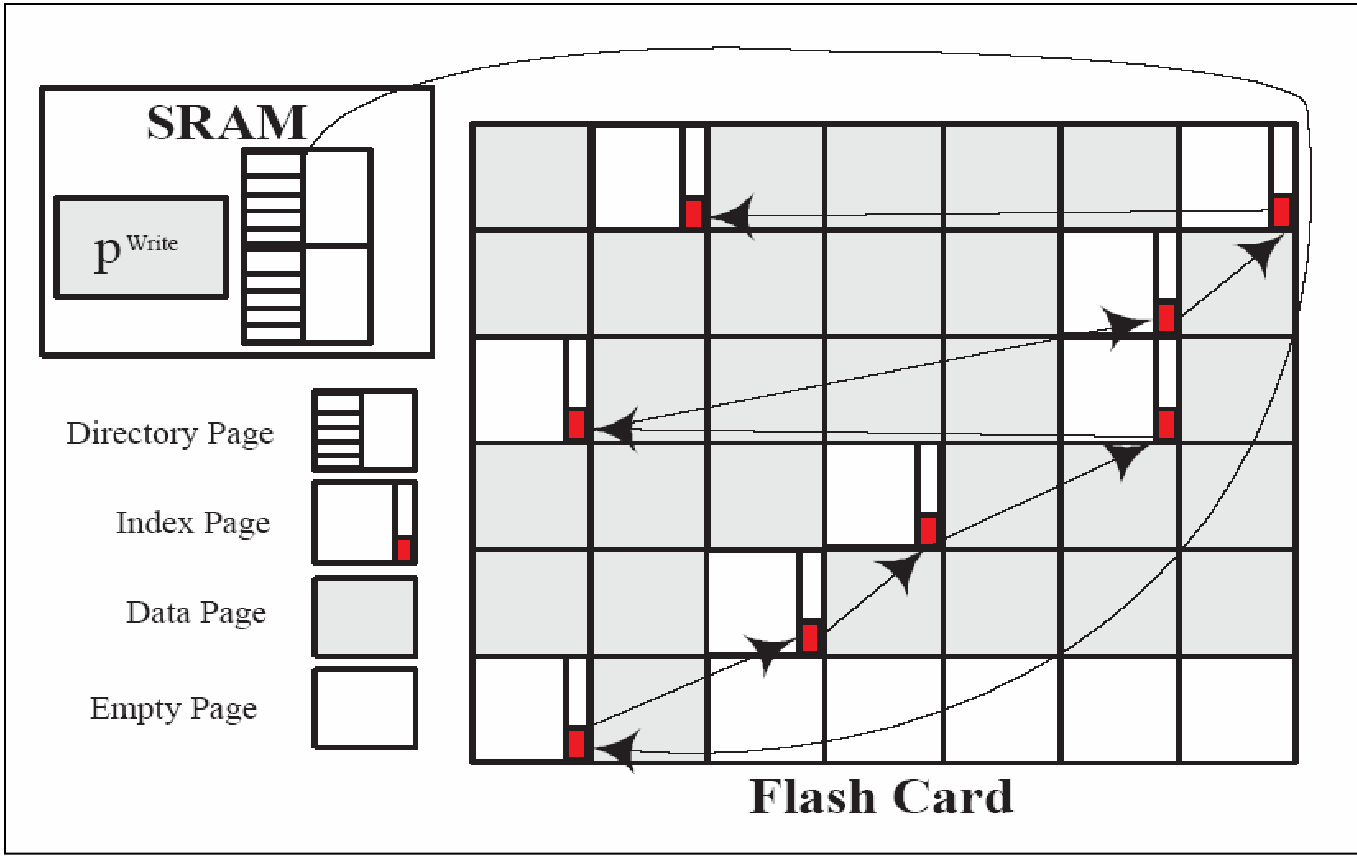
Indexing Techniques for Flash Memory

- Design an efficient index structure on sensor data in flash memories to support
  - ***Value-Based Range or Equality Queries***
  - ***Time-Based Range or Equality Queries***

  that provides
  - **Wear-Leveling:** Spread page writes out uniformly across the storage media in order to avoid wearing out specific pages.
  - **Block-Erase:** Minimize the number of *random access deletions*.
  - **Fast-Initialization:** Minimize the size of in memory (SRAM) structures.

MicroHash

- 4 types of pages
  - Root Page
  - Directory Page
  - Index Page
  - Data Page
- 4 operation phases
  - Initialization
  - Growing
  - Repartition
  - Deletion
- 2-level memory hierarchy
  - On-chip RAM
  - Off-chip Flash Media
  - (also on-chip Flash)

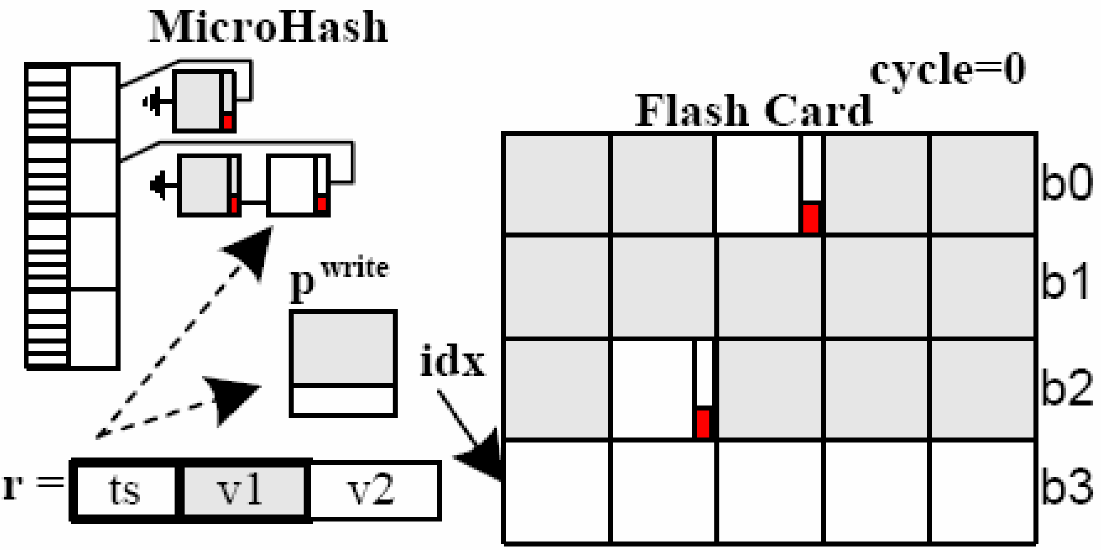


Page Types in MicroHash

- Root Page
  - contains information related to the state of the flash media, e.g. it contains the position of the last write (idx), the current write cycle (cycle) and meta information about the various indexes stored on the flash media
- Directory Page (the hash table)
  - contains a number of directory records (buckets) each of which contains the address of the last known index page mapped to this bucket.
- Index Page
  - contains a fixed number of index records and the 8 byte timestamp of the last known data record. The latter field, denoted as *anchor* is exploited by timestamp searches.
- Data Page
  - contains a fixed number of data records

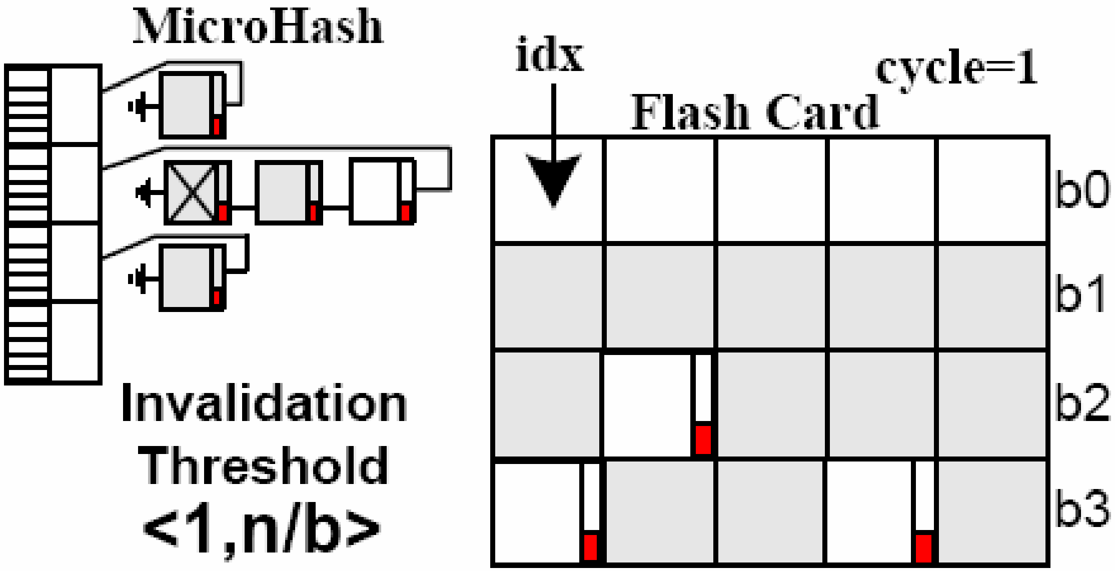
Operations in MicroHash

- Growing
  - Collect data and fill up data buffer page P^{write} in *SRAM*.
  - Force P^{write} out to flash media.
  - Create index records for each data record in P^{write}.
  - Force out index pages by *LRU* if the *SRAM* is too small to hold the new generated index records.



Operations in MicroHash: Deletion

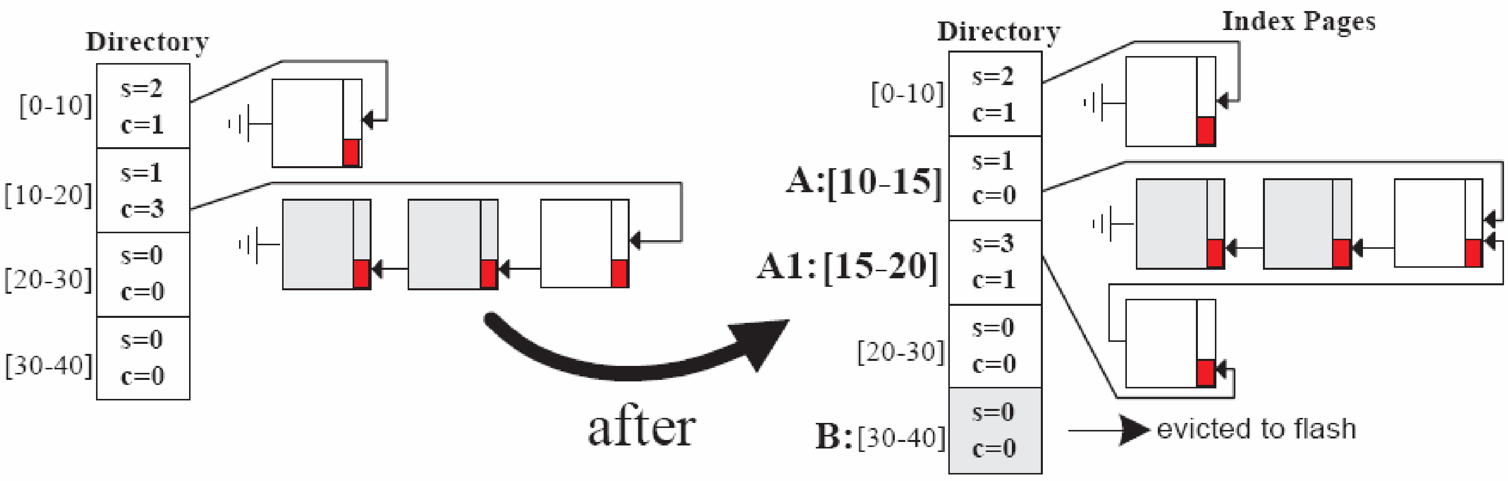
- Deletion
  - Take the flash media as a circular buffer and keep a pointer as the next writing position.
  - If we want to write and the flash media is full, delete the next block pointed by the pointer



Operations in MicroHash: Repartition

- Equi-width bucket splitting deteriorates under biased data distribution
- Splitting policy:

  If bucket A links to more than τ% of total index records, evict the least used bucket B and segment bucket A into A and A’
- No bucket reassignments of old records, which avoids large volume index page access



Searching in MicroHash

- Searching by value
  1. locate the appropriate directory bucket, from which the system can extract the address of the last index page
  2. read the respective index pages on a page-by-page basis
  3. read the data records referred by the index pages on a page-by-page basis

- Searching by timestamp
  - The generated data pages are written out sequentially into flash media.
  - Index pages are mixed together with data pages.
  1. Binary search (O(log(n)) ~20 for 1GB flash media)
  2. *LBSearch* (less than 10)
  3. *ScaleSearch* (better than *LBSearch*)

LBSearch and ScaleSearch

- *LBSearch*

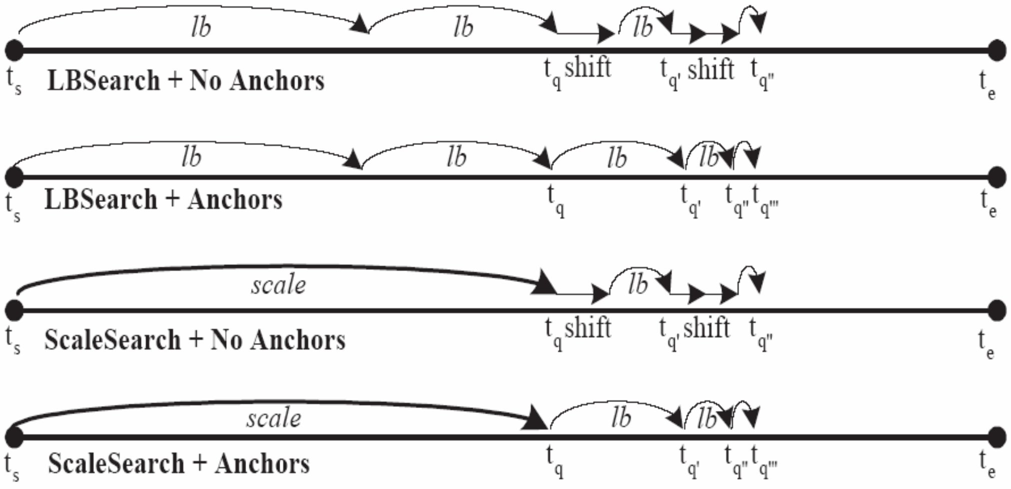
  Let Idx_{start} the last written page in flash media, t_s the earliest timestamp, and R the maximum timestamp range in a data page

  $$Idxlb(t_{q,} t_s) = Idx_{start} + ceil[(t_q - t_s) / R]$$

- *ScaleSearch*

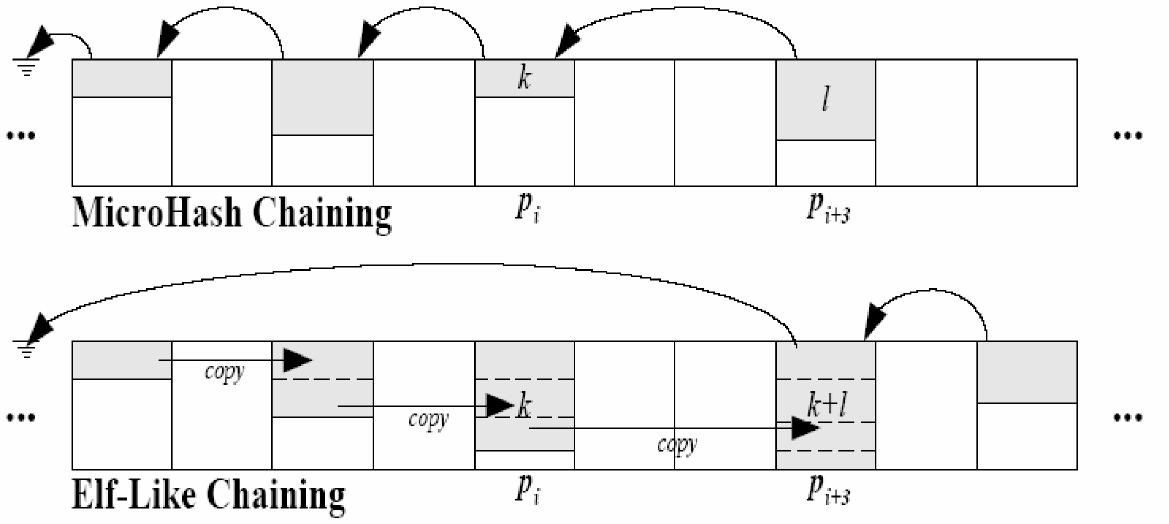
  Let Idx_{start} the last written page in flash media, t_s and t_e the earliest and oldest time stamp, and n the total number of index and data pages

  $$Scalelb(t_{q,} t_s) = Idx_{start} + ceil[n \bullet (t_q - t_s) / (t_e – t_s)]$$



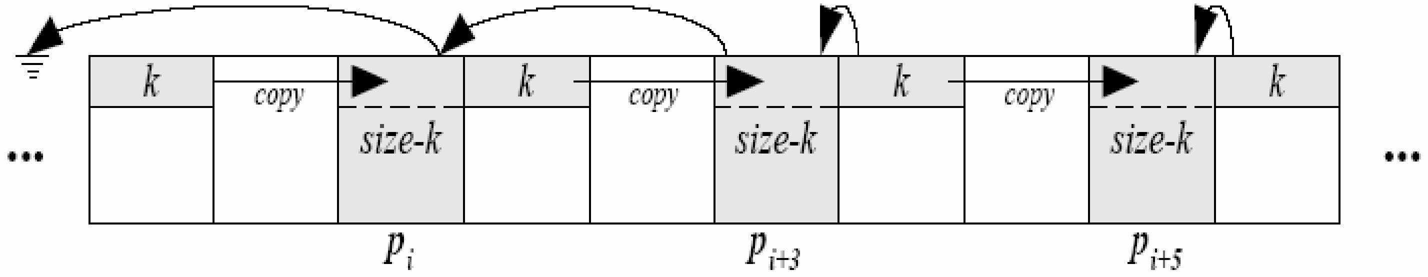
MicroHash vs ELF

- ***MicroHash***
  - pages are chained using a back-pointer, once written out, the page will exist in the chain forever.
- ***ELF (Efficient Log Structured Flash File System, Dai et. al., SenSys 2004)***
  - a linked list in which each node, other than the last node, is completely full.
  - keeps copying the last non-full page into a newer page, when new records are requested to be added.



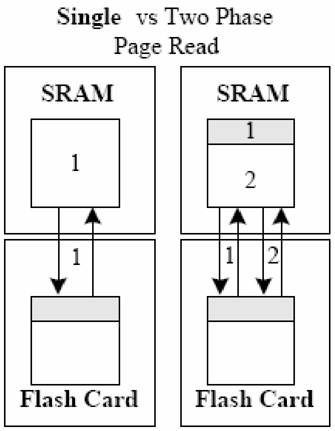
Limitations of the ELF solution

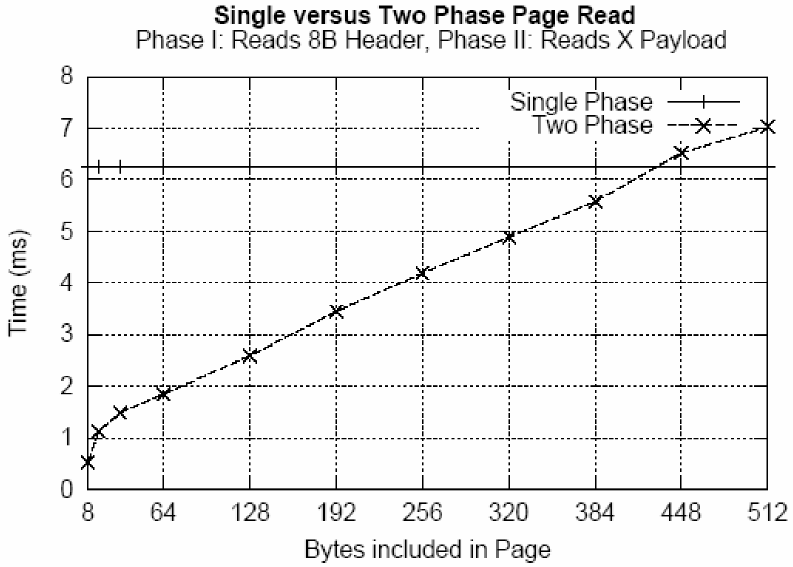
- Example
  - There are *k* records in the last page of ELF chain at the beginning.
  - The buffer manager requests to write out 3 full pages consecutively.
  - ELF will use twice the required space to accommodate all records.



Two-Phase Page Reads

- Page-by-Page Reads
  - When pages are not fully occupied, such as index pages, then a lot of empty bytes (padding) is transferred from the flash media to memory.
- Two-Phase Page Reads
  - Reads a fixed header from flash in the first phase, and then reads the exact amount of bytes in the next phase.





Experimental Evaluation

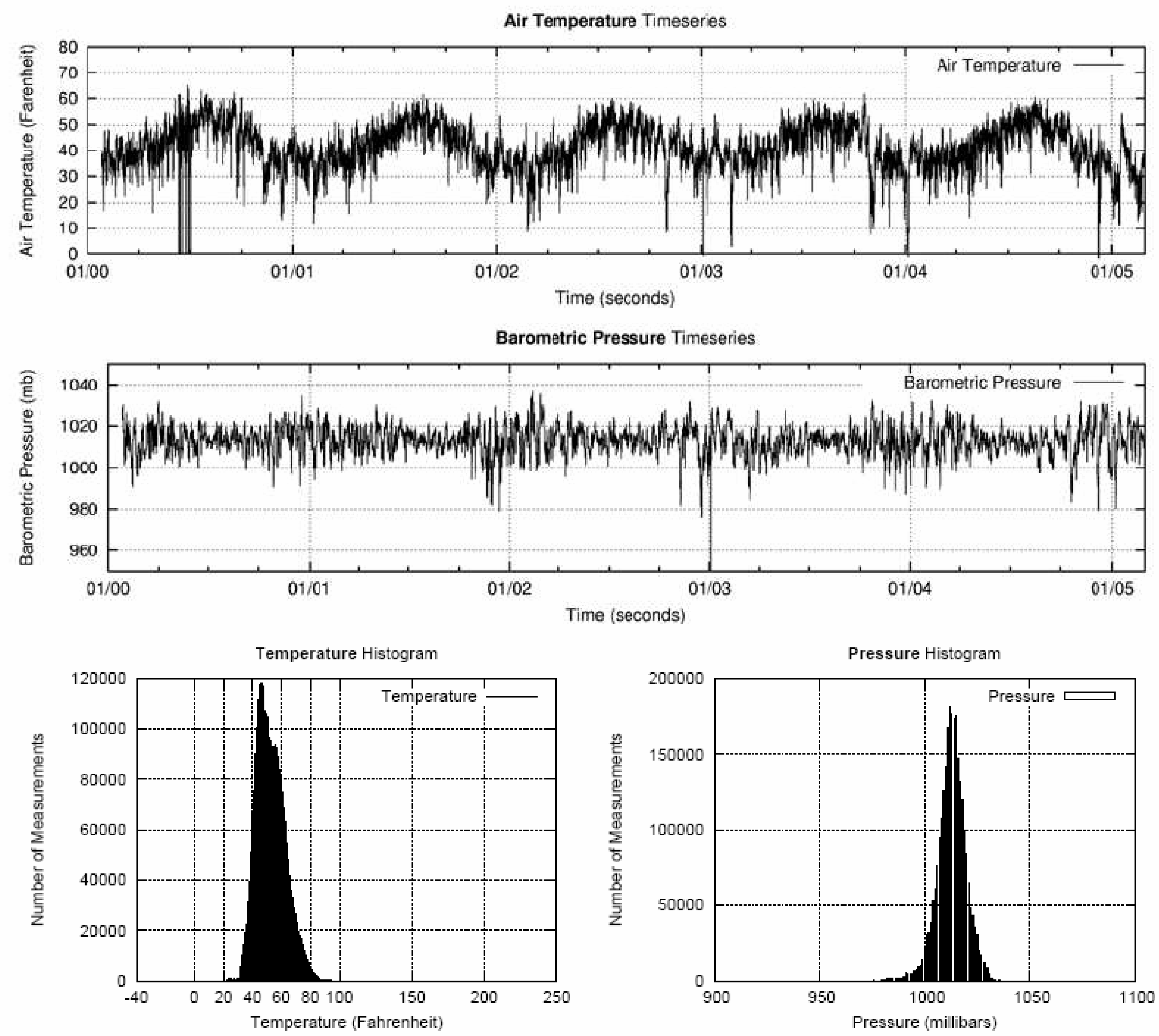
- Implemented with a tiny LRU Buffer Manager
- Run our code in TOSSIM, the simulation environment of TinyOS
- **Datasets**:
  - **Washington State Climate**
    - A real dataset of atmospheric data collected by the Department of Atmospheric Sciences at the University of Washington.

      268MB dataset contains readings for 2 months in ‘05 (barometric pressure, wind speed, relative humidity, and others)
  - **Great Duck Island (GDI 2002):**
    - A real dataset from the habitat monitoring project on the Great Duck Island in Maine.

      We use readings included the following readings: light, temperature, thermopile, humidity and voltage.

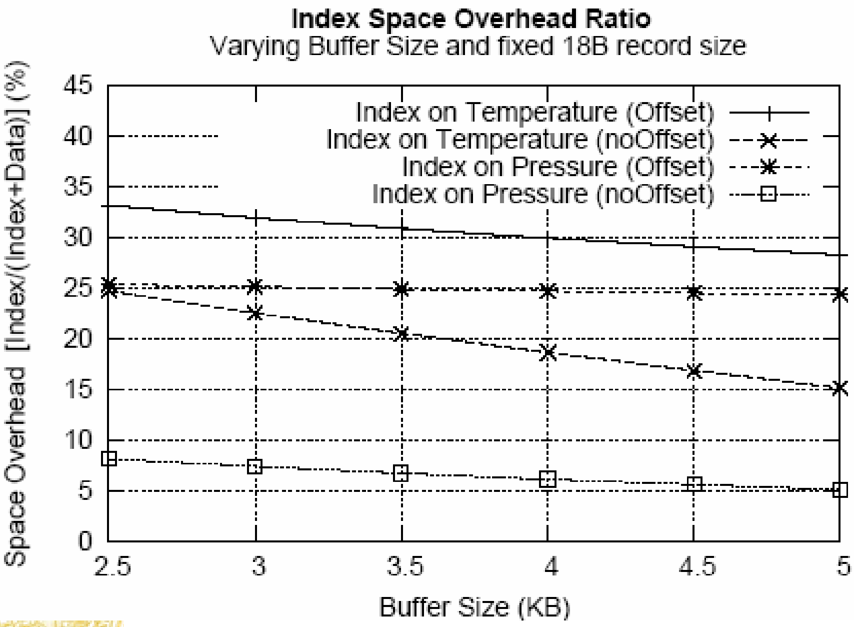
      Our dataset includes approximately 97,000 readings that were recorded between October and November 2002.

Distribution of Data

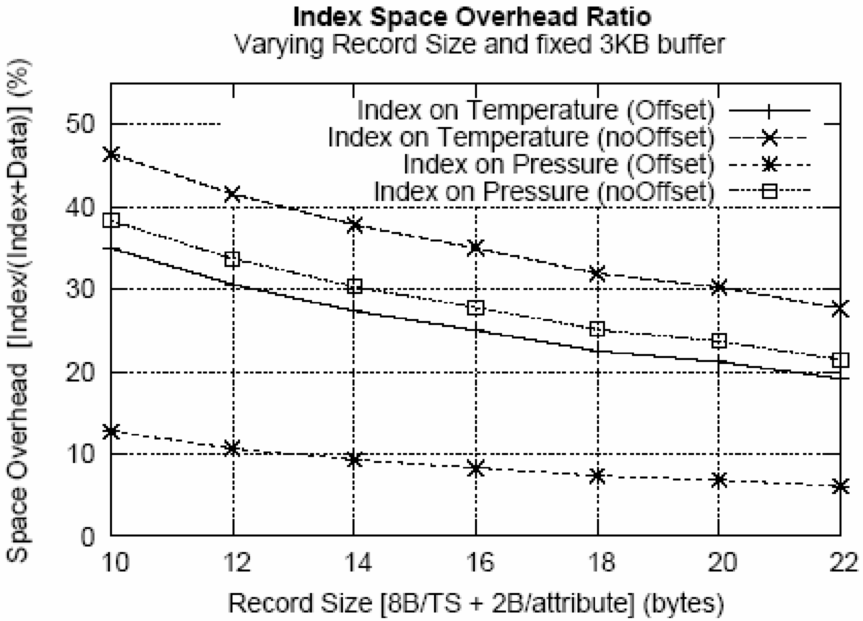


Overhead of Index Pages

- Index page overhead Φ = IndexPages/(DataPages+IndexPages)
- Two Index page layouts
  - *Offset*, an index record has the following form {pageid,offset}
  - *NoOffset*, in which an index record has the form {pageid}
- 128 MB flash media (256,000 pages)
  - varying SRAM (buffer) size (2.5 - 5KB)
  - Varying data record size (10 – 22 Bytes)

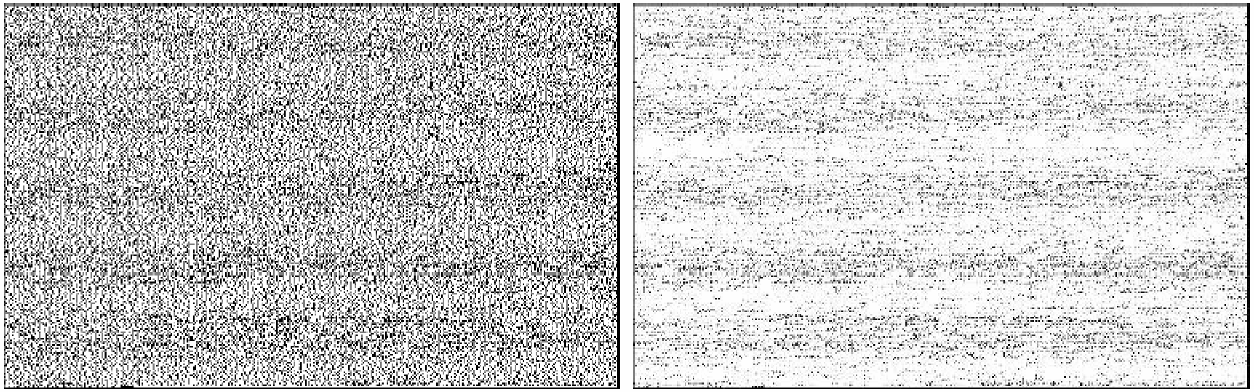


Index Space Overhead Ratio
Varying Buffer Size and fixed 18B record size



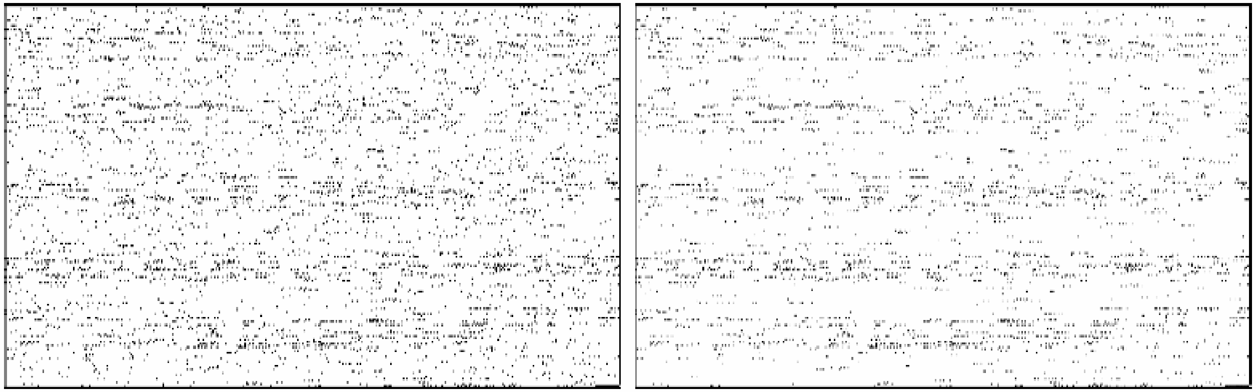
Index Space Overhead Ratio
Varying Record Size and fixed 3KB buffer

Overhead of Index Pages



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

2.5K Buffer

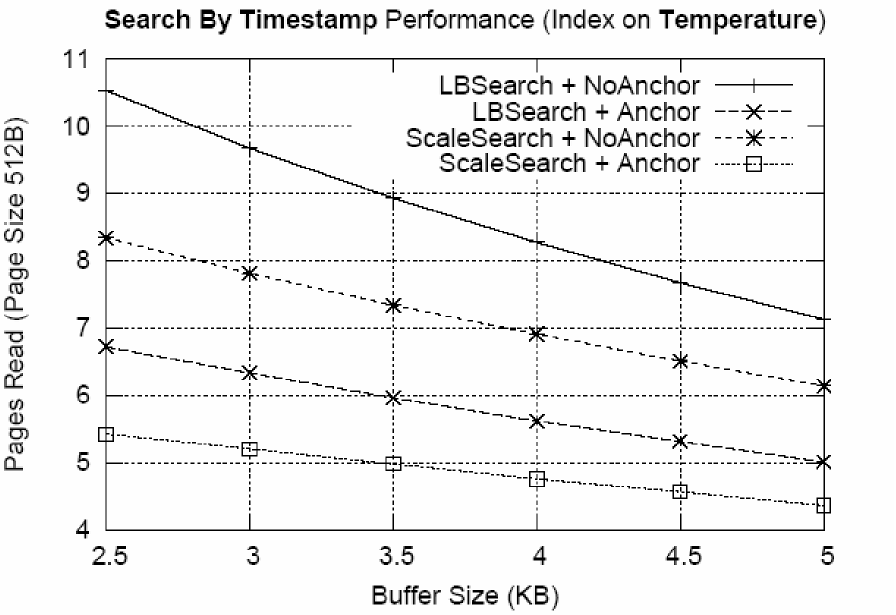


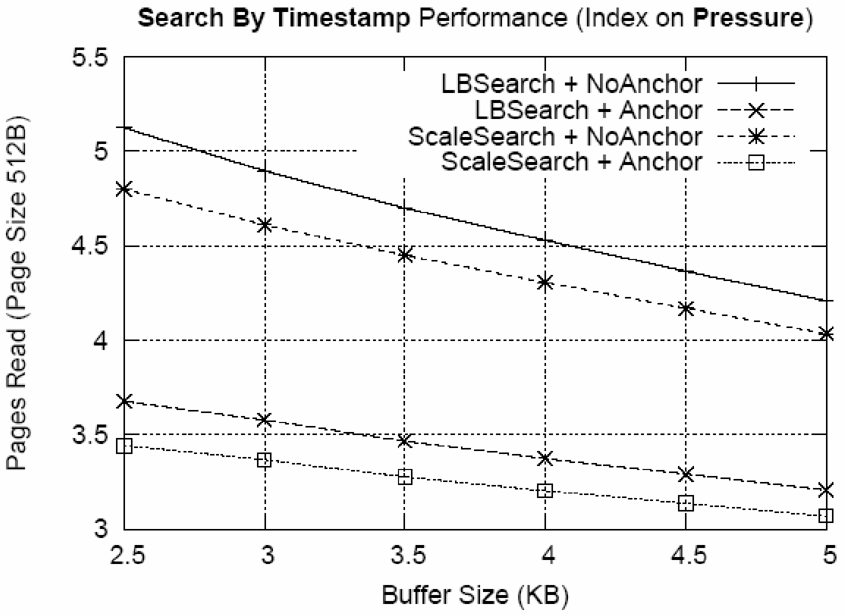
Index/Data Pages (left) — Grayscale Occupancy (right)

10K Buffer

Searching by TimeStamp

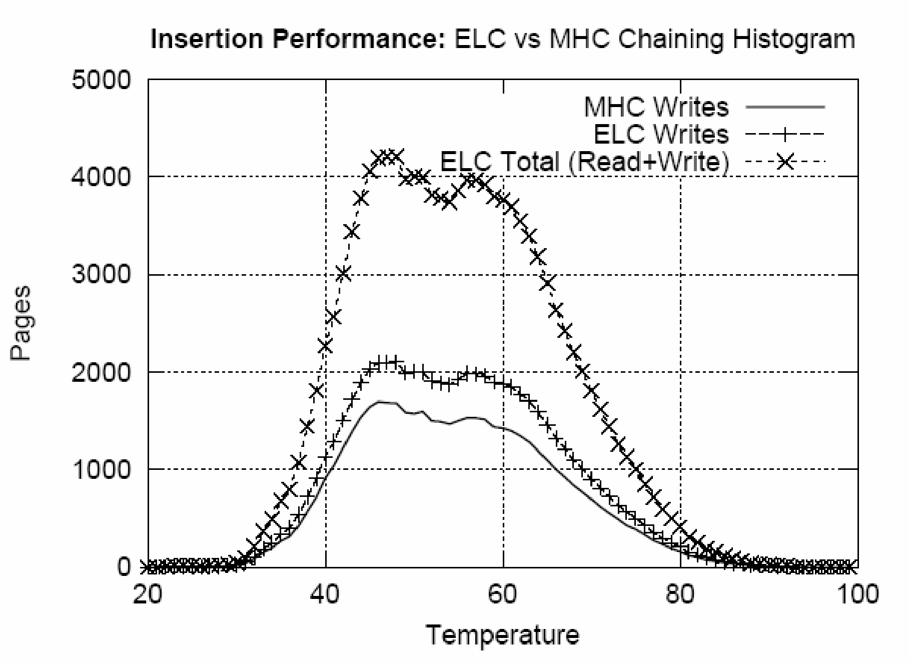
- 128 MB flash media (256,000 pages), varied SRAM (buffer) size
- 2 Index page layouts
  - *Anchor*, every index page stores the last known data record timestamp
  - *No Anchor*, the index page does not contain any timestamp information

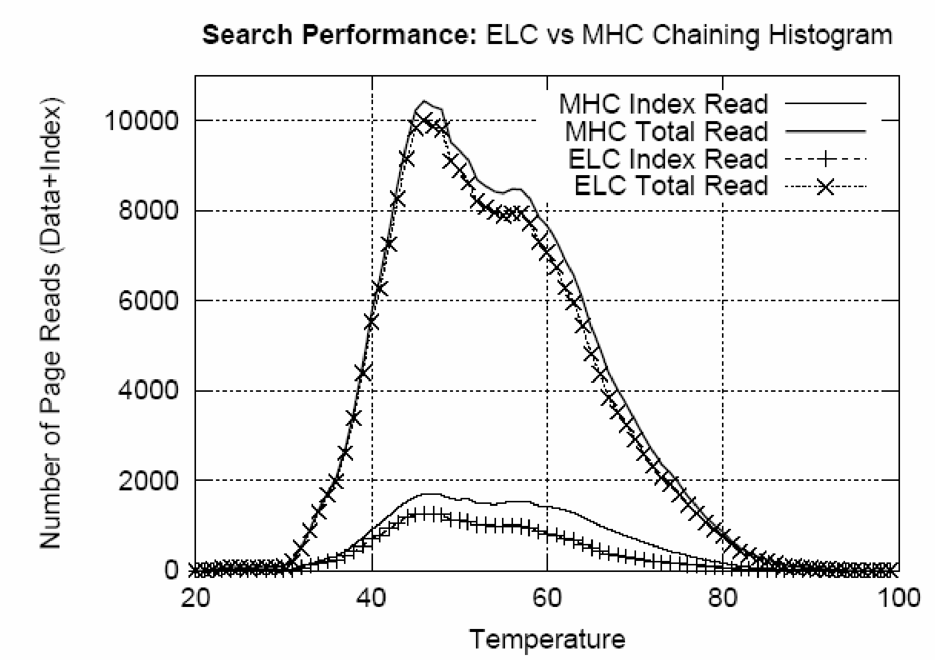




Searching by Value (MicroHash vs ELF)

- 128MB flash media(256,000 pages), fixed 3KB SRAM
- Compare with hash indexing + *ELF*[1]

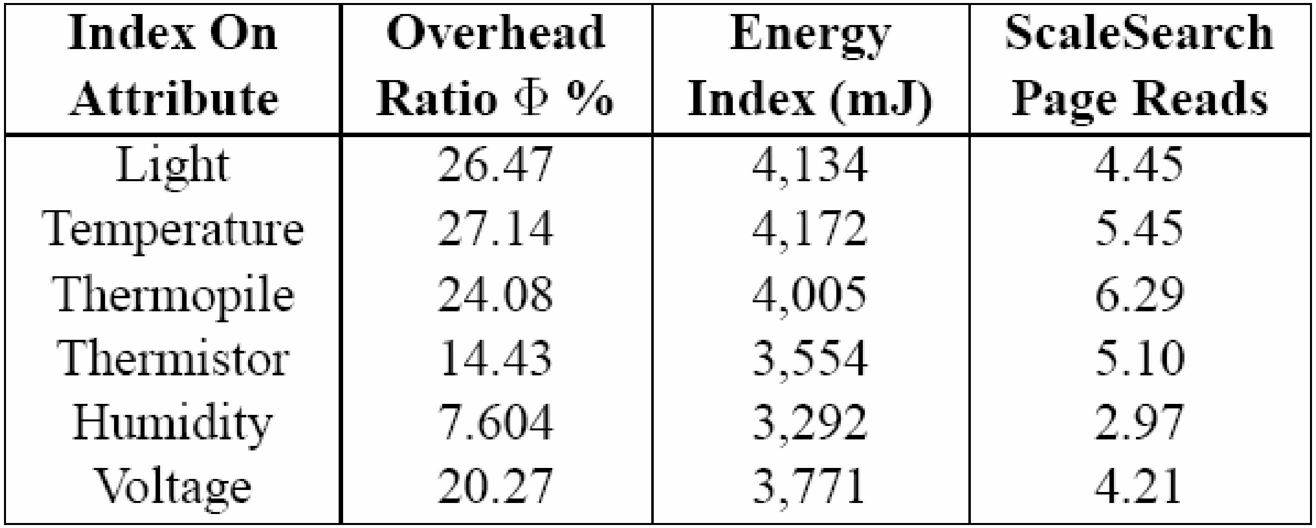




[1] Dai H., Neufeld M., Han R., "ELF: an efficient log-structured flash file system for micro sensor nodes", In SENSYS, Baltimore, pp. 176-187, 2004.

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

Conclusions

- Gave an extensive study of NAND flash memory when this is used as a storage media of a sensor device
- Proposed the *MicroHash* index:
  - an efficient external memory hash index for equality and range queries that addresses the characteristics of flash memory
- Our experimental evaluation with real traces shows that the structure we propose is both efficient and practical
- Future work:
  - Deploy the prototype
  - Buffer optimizations
  - Indexing multidimensional datasets