SMIS: Secured and Mobile Information Systems

INRIA Paris-Rocquencourt
Joint team with CNRS & University of Versailles (UVSQ)

Junior Seminar
INRIA may 19th 2015

Background and research fields

A Core Database Culture …

- Storage and indexing models, query execution and optimization
- Transaction protocols (atomicity, isolation, durability)
- Database security (access and usage control, encryption)
- Distributed DB architectures

Current composition of the team

Permanent members

Nicolas Anciaux CR INRIA
Luc Bouganim, DR INRIA
Benjamin Nguyen, MC UVSQ, since 2010
Philippe Pucheral, PR UVSQ
Iulian Sandu Popa, MC UVSQ, since 2012

Engineers

Quentin Lefebvre
Aydogan Ersoz

PhD students

Saliha Lallali: Document Indexing for Embedded Personal Databases
Athanasia Katsouraki: Access and Usage Control for Personal Data in Trusted Cells
Cuong To: Secure Global Computations on Personal Data Servers
Paul Tran-Van: Sharing file in Embedded Personal Databases

…

A Scalable Search Engine for Mass Storage Smart Objects

SMIS project (INRIA, Prism, Univ. Versailles)

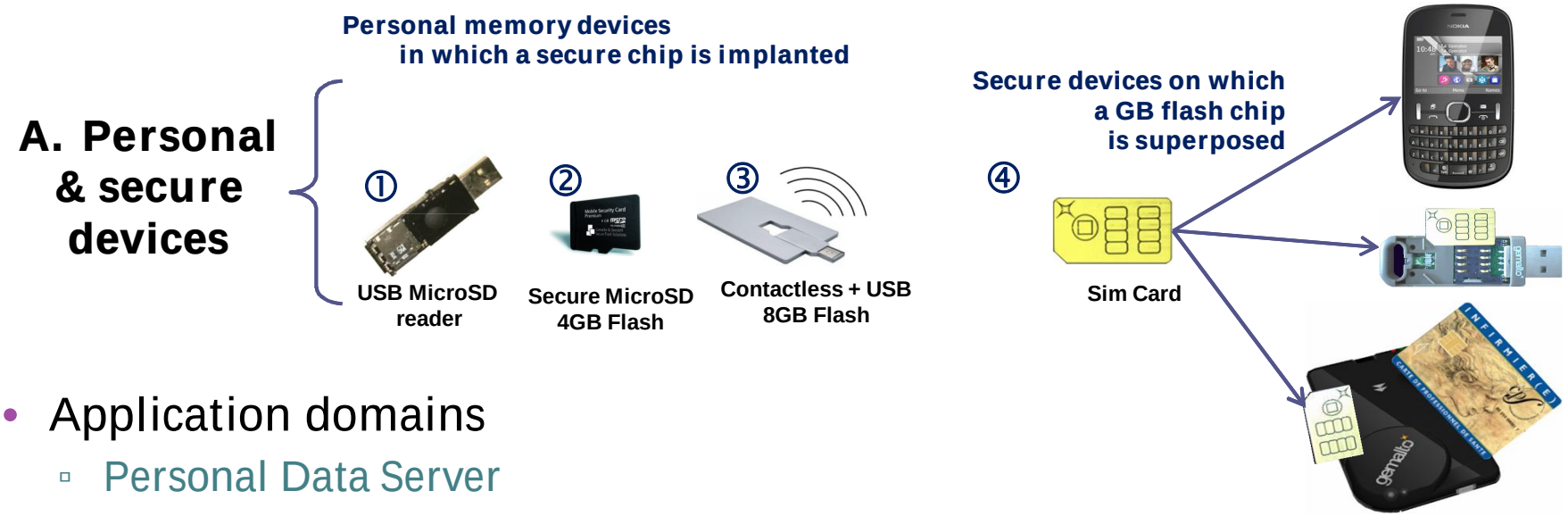
Saliha LALLALI
Nicolas ANCIAUX
Iulian SANDU POPA
Philippe PUCHERAL

Inria INVENTEURS DU MONDE NUMÉRIQUE

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

cnrs

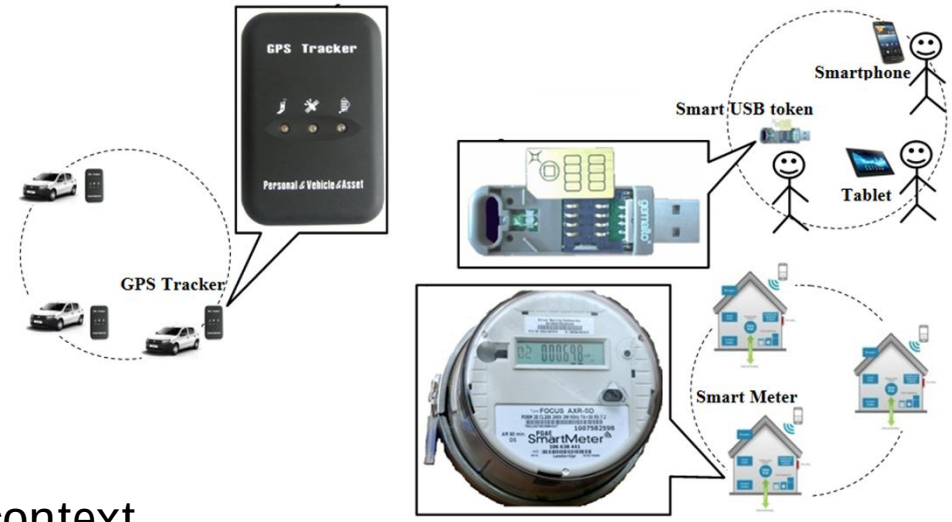
Motivation – Advent of Smart Objects



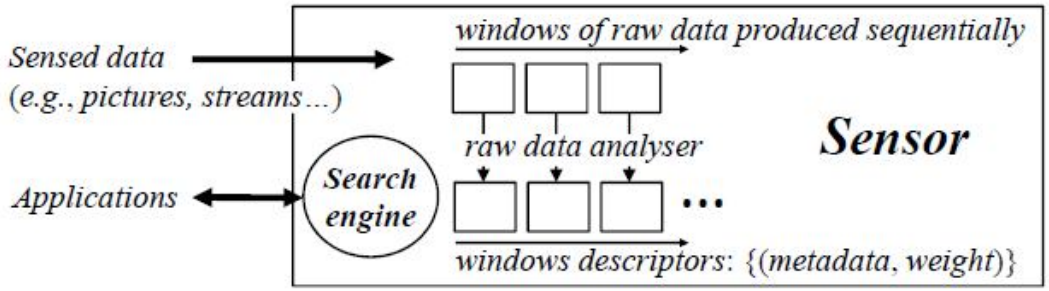
- Application domains
  - Personal Data Server
  - Personal Cloud / Personal Web
- Securely store, query and share personal user's files and their metadata
  - Documents, photos, emails, links, profiles, preferences ...
- Base required functionality: *full text search* (similar to an embedded Google desktop or Spotlight)

Motivation – Smart Metering and Internet of Things

B. Smart meters and IoT



- Smart sensor context
  - Smart meters/objects (Linky, GPS tracker, set-up box, ...)
  - Smart sensors recording events in theirs surroundings (camera sensor, Google glass)

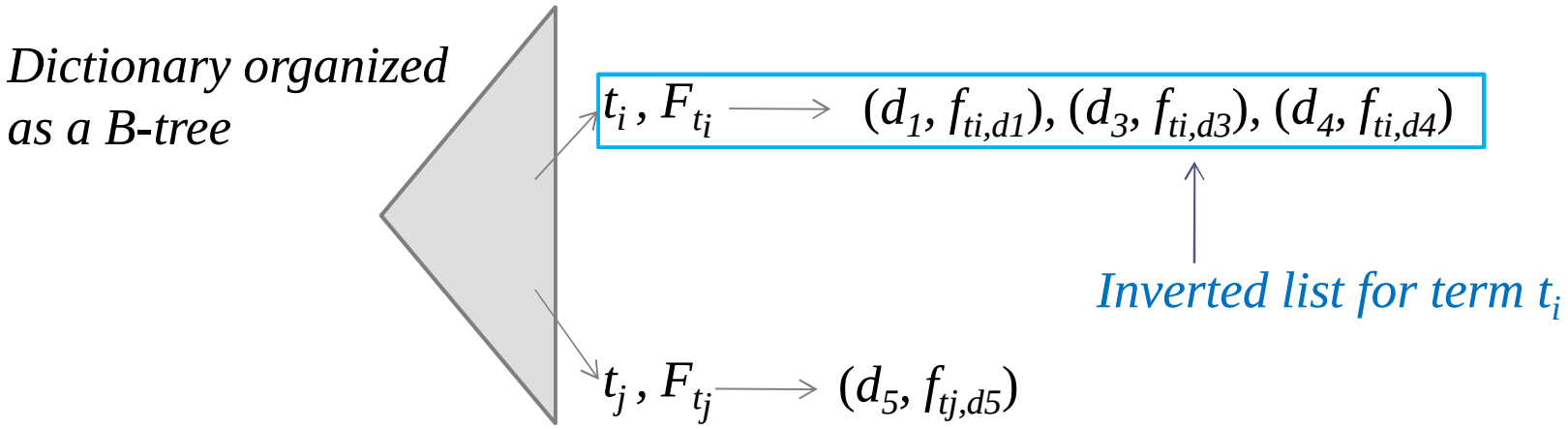


Smart Objects and Data Management

- Why transposing traditional data management functionalities directly into the smart objects?
- Managing large collections of data locally in smart objects exhibits very good properties in terms of:
  - Privacy & security
    - Data distribution
    - Transfer only the results and not the data
  - Energy saving
    - Avoiding to transmit all the data to a central server
    - Transferring few data (the results)
  - Bandwidth savings
- Several works consider the problem of data management in SOs:
    - Basic filtering and SQL query support
    - Facial recognition
    - **Full text search (documents, images: tags/visterms, any tagged data objects)**

Full-Text Search Requirements (1)

- Inverted index
  - A search structure or (a dictionary): stores for each term *t* appearing in the documents the number F_t of documents containing *t* and a pointer to the inverted list of *t*
  - A set of inverted lists: where each list stores for a term *t* the list of (d, $f_{d,t}$) pairs where *d* is a document identifier that contains *t* and $f_{d,t}$ is the weight of the term *t* in the document *d*

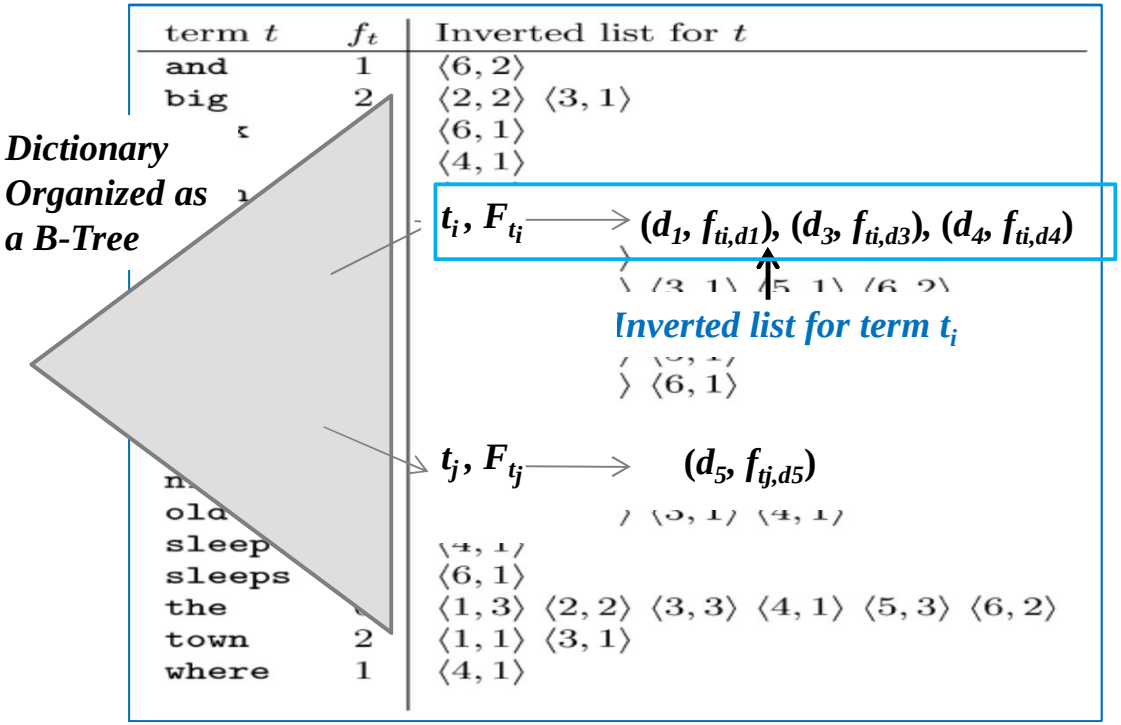


Full-text search requirements (1)

1. The old night keeper keeps the keep in the town.
2. In the big old house in the big old gown.
3. The house in the town had the big old keep.
4. Where the old night keeper never did sleep.
5. The night keeper keeps the keep in the night.
6. And keeps in the dark and sleeps in the light.

« Keeper » Document Set

« Keeper » Inverted Index



Full-Text Search Requirements (2)

- Answer full-text search queries
  - For a set of query keywords, produce the *k* most relevant documents (according to a weight function like TF-IDF)

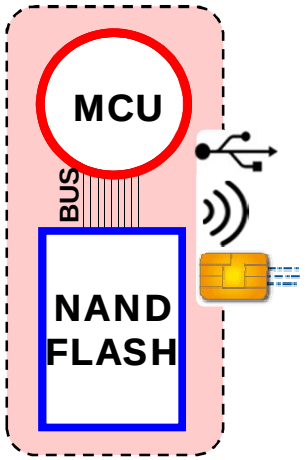
$$\text{TF-IDF(doc)} = \sum_{\{t_i\}\ \text{query keywords}} \left(f_{d,t_i} * \text{Log}(N / F_{t_i}) \right)$$

- To evaluate the query:
  1. Access the inverted index search structure, retrieve for each query term t the inverted lists elements
  2. Allocate in RAM one container for each document identifier in these lists
  3. Compute the score of each of these documents using the *TF-IDF* formula
  4. Rank the documents according to their score and produce the *k* documents with the highest score

too much!

Smart Object HW Architecture

- Smart objects share a common architecture
- (Secure) Microcontroller
  - Low cost
  - But small RAM (5KB ~ 128KB)
- NAND Flash
  - Dense, robust, low cost
  - But high cost of random writes
    - Pages must be erased before being rewritten
    - Erase by block vs. write by page

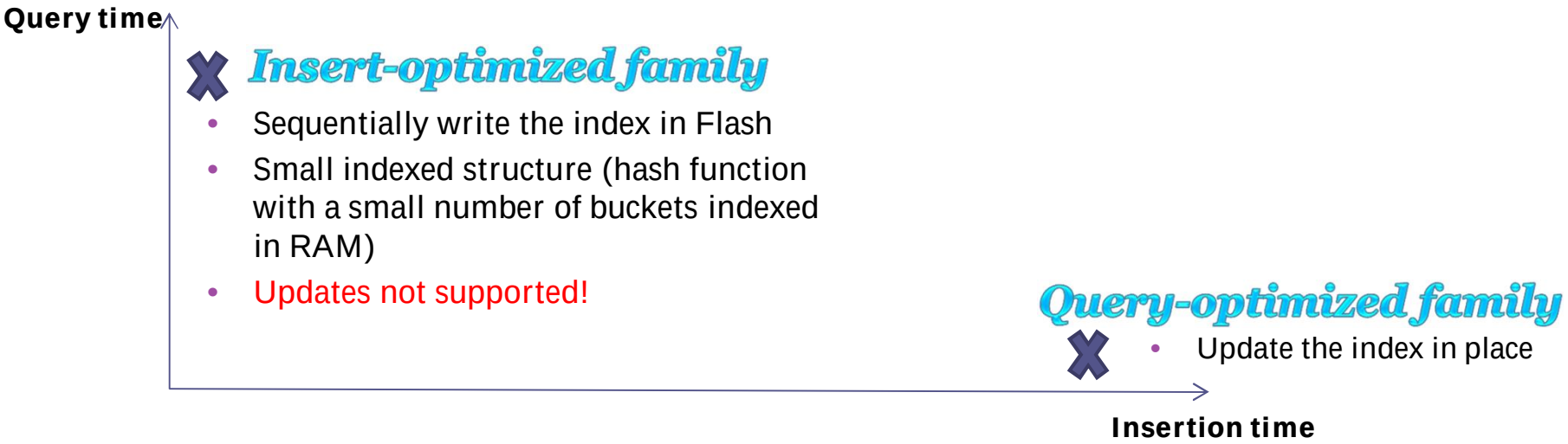


- Tiny RAM and NAND Flash introduce conflicting constraints for data indexing

How do existing techniques deal with these constraints ?

Problem Statement

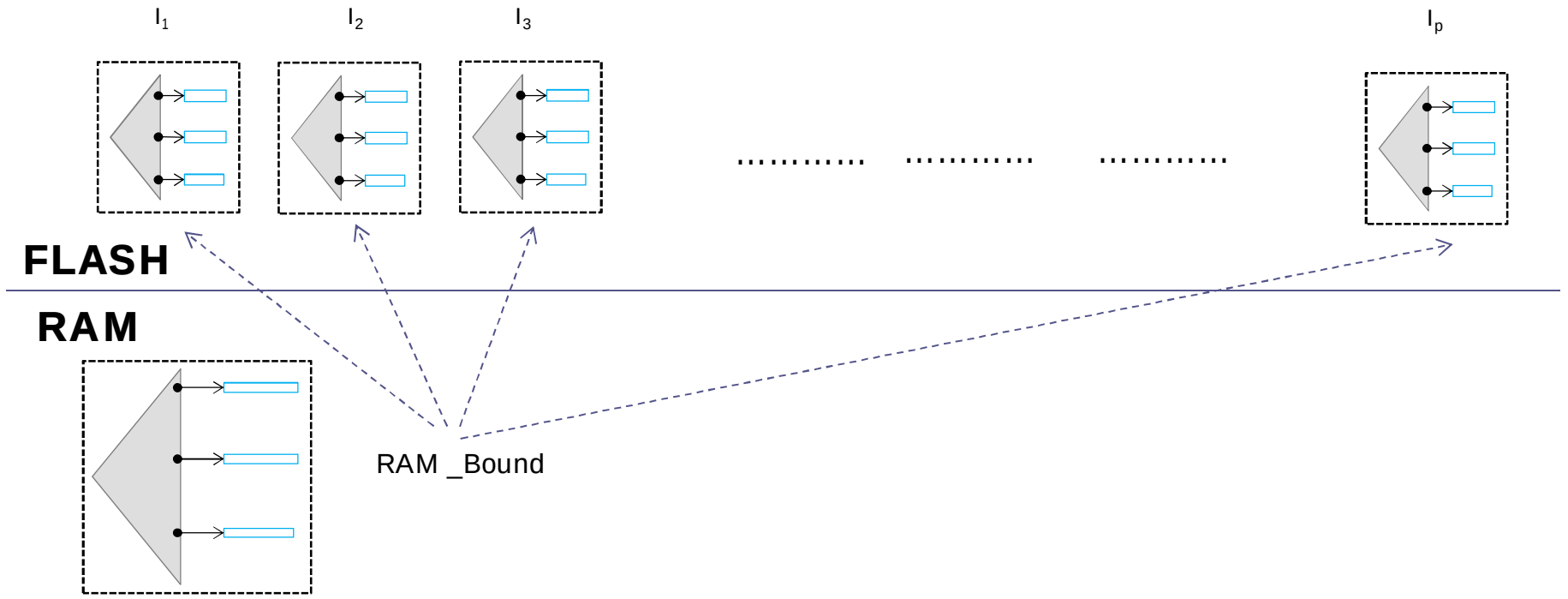
- Challenge: execute queries with a very small RAM on large volumes of data indexed in NAND Flash



- **Objectives** of the proposed solution: *Bounded RAM (a few KB) & Full Scalability (both for updates and queries)*
- **Design principles**
  - Write-once partitioning (update scalability)
  - Linear pipelining (query evaluation under a Bound RAM)
  - Background merging (query/update scalability)

Principle1: Write-Once Partitioning

Split the inverted index structure in successive partitions such that a partition is flushed only once in Flash and is never updated.

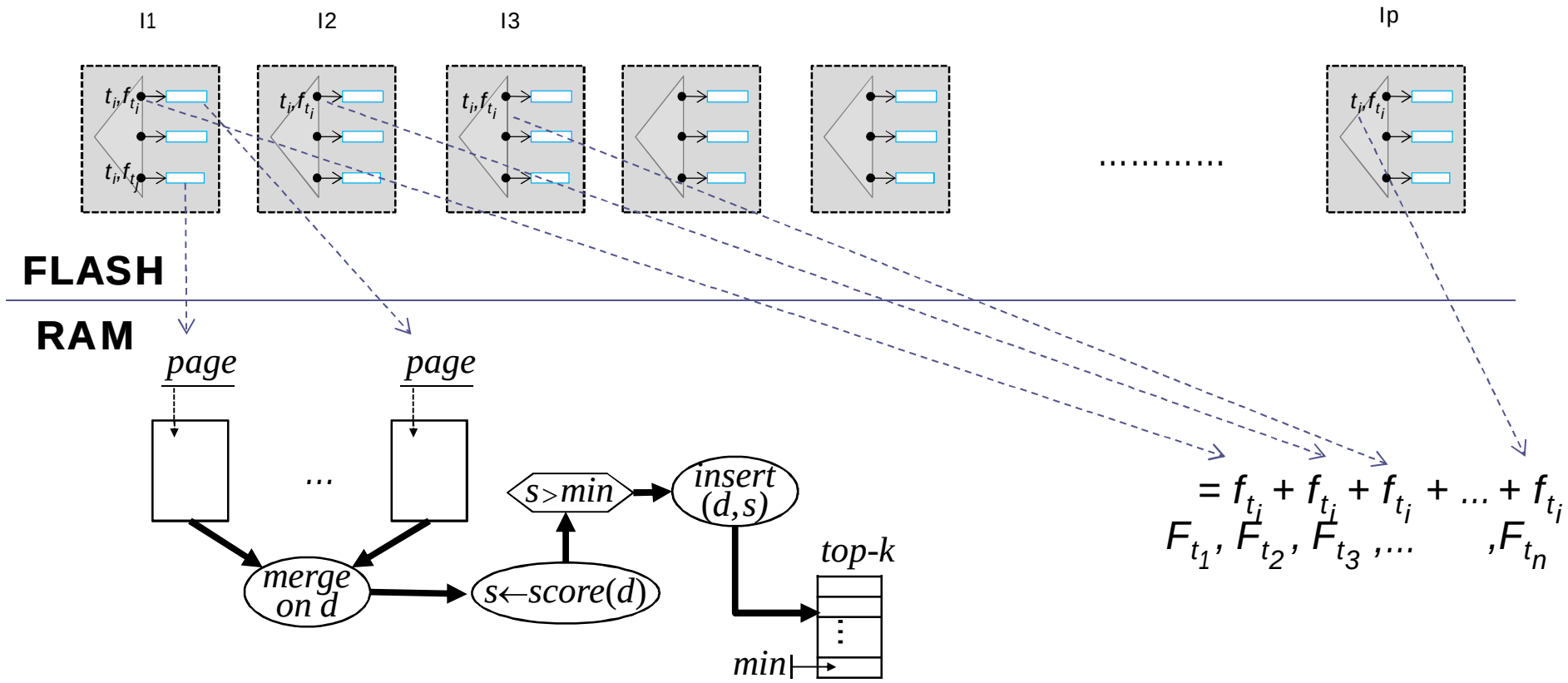


Principle2: Linear Pipelining

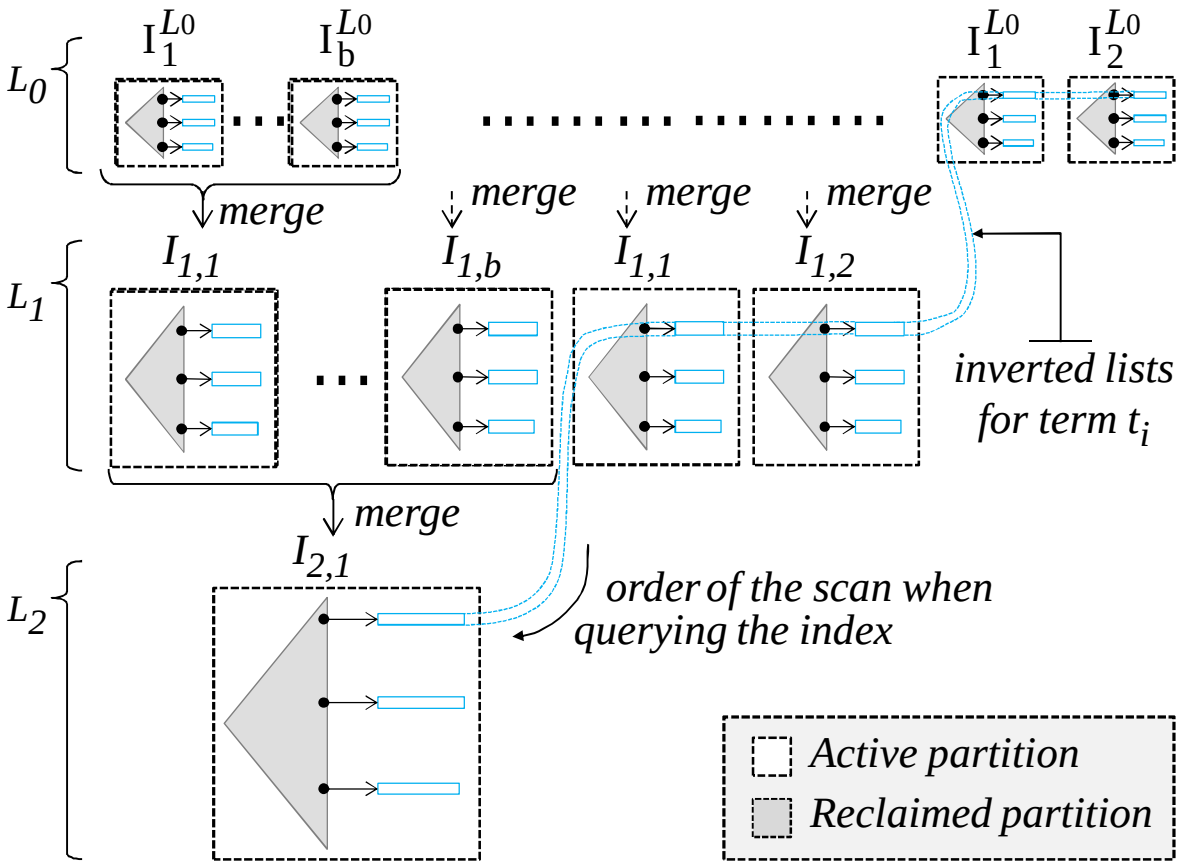
For a $Q = \{t_1, t_2, \ldots, t_n\}$:

$$Top_k \left[\sum_{t_i \in Q} W(f_{d,t} * \log\left(\frac{N}{F_{t_i}}\right)) \right]$$

a global metadata



Principle3: Background Linear Merging



SSF (Scalable and Sequential Flash structure)

Document Deletions (1)

- Implementing the delete operation is challenging :
  - Index updating → Random updates in the index
  - State of the art embedded search indexes do not support/consider document deletions/updates

- The alternative to updating in-place is *compensation*:
  - Store the Deleted Document Identifier (DDIs) as a sorted list in Flash
  - Intersect DDIs lists at query execution time with the inverted lists of the query terms

- Compensation problems:
  - Random documents deletion → maintaining a sorted list of DDIs in Flash
    ⇨ violate the Write-Once Partitioning principle
  - The F_t computation need an additional merge operation to subtract the sorted list of DDIs from the inverted lists for each term in the query
  - the full DDI list has to be scanned for each query regardless of the query selectivity
    ⇨ violate the Linear Pipelining principle

Document Deletions (2)

- Retained deletion method:
  - Compensate the index structure itself:
    - A pair of (*term*, *d*, - $f_{d,t}$) is inserted in I_i for each term in the deleted document *d*
    - f_t of each term *t* in *d* is decremented by *1*

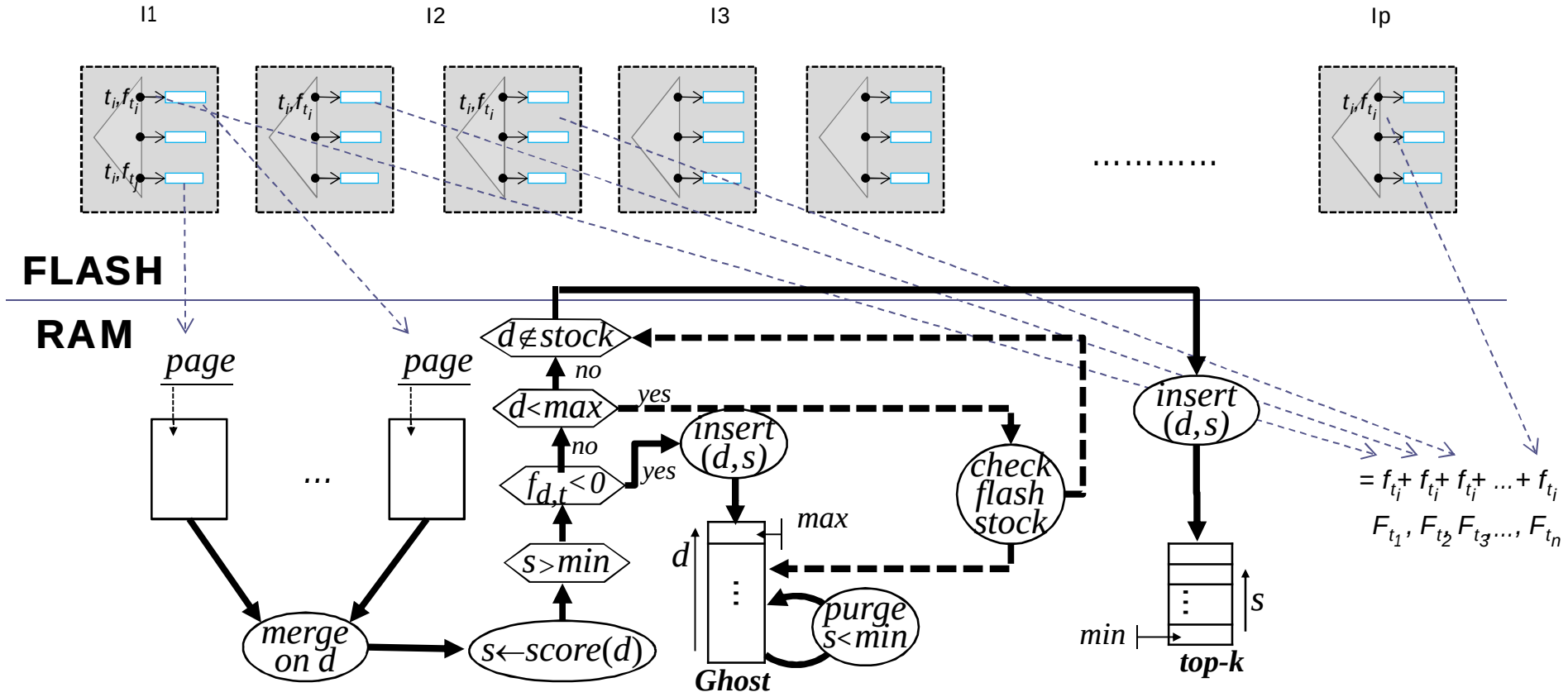
- The objective is threefold:
  - Preclude random writes ⇨ Write-Once Partitioning principle
  - Query selectivity ⇨ Linear Pipelining principle
  - Absorb the deleted documents in Background Merging

Document Deletions (3)

- **Query:**

$$Top_k \left[\sum_{t_i \in Q} W(f_{d,t} x \log\left(\frac{N}{F_{t_i}}\right)) \right]$$

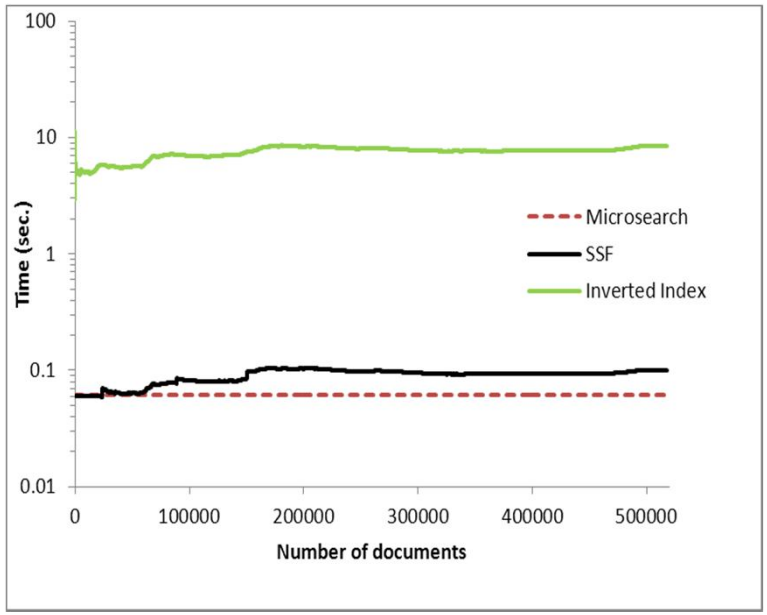
a global metadata



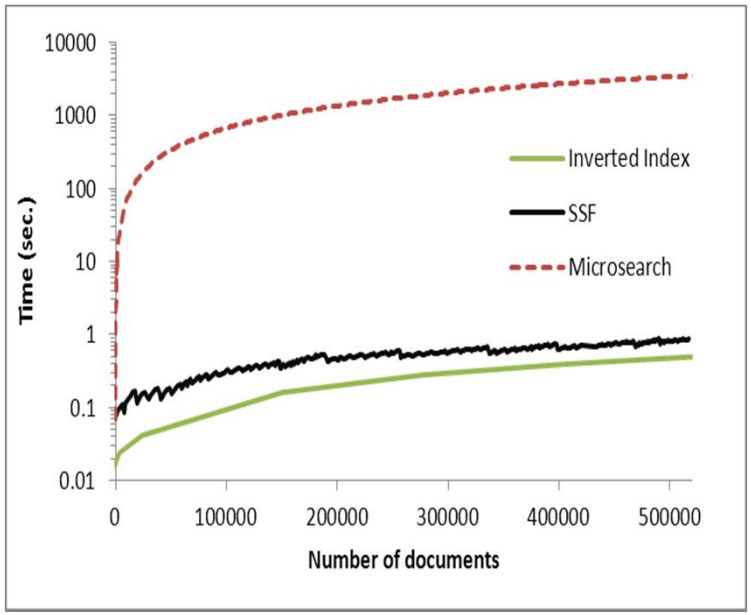
Experimental Evaluation

- HW platform:
  - development board ST3221G-EVAL
    - MCU STM32-F217IG and microSD card
    - Storage on two SD cards (Silicon Power SDHC Class 10 4GB & Kingston microSDHC Class 10 4GB)
  - Index RAM bound = 5KB
    - SSF branching factors: *b*=8 and *b*'=3
- Datasets and query sets
  - PDS/Personal Cloud use-case: "rich" documents
    - very large vocabulary (500k terms) and documents (more than 1000 terms per doc on average)
      - ENRON email dataset: 500k emails (946MB of raw text)
      - Pseudo-desktop dataset (CIKM'09): 27k documents, i.e., email, html, pdf, doc and ppt (252 MB of raw text)
  - Smart sensor use-case: "poor" documents
    - moderate vocabulary (10k terms) and documents (100 terms per doc on average)
      - Synthetic dataset: 100k documents (129MB of raw text)

Comparison with the State-of-the-Art Search Engine Methods

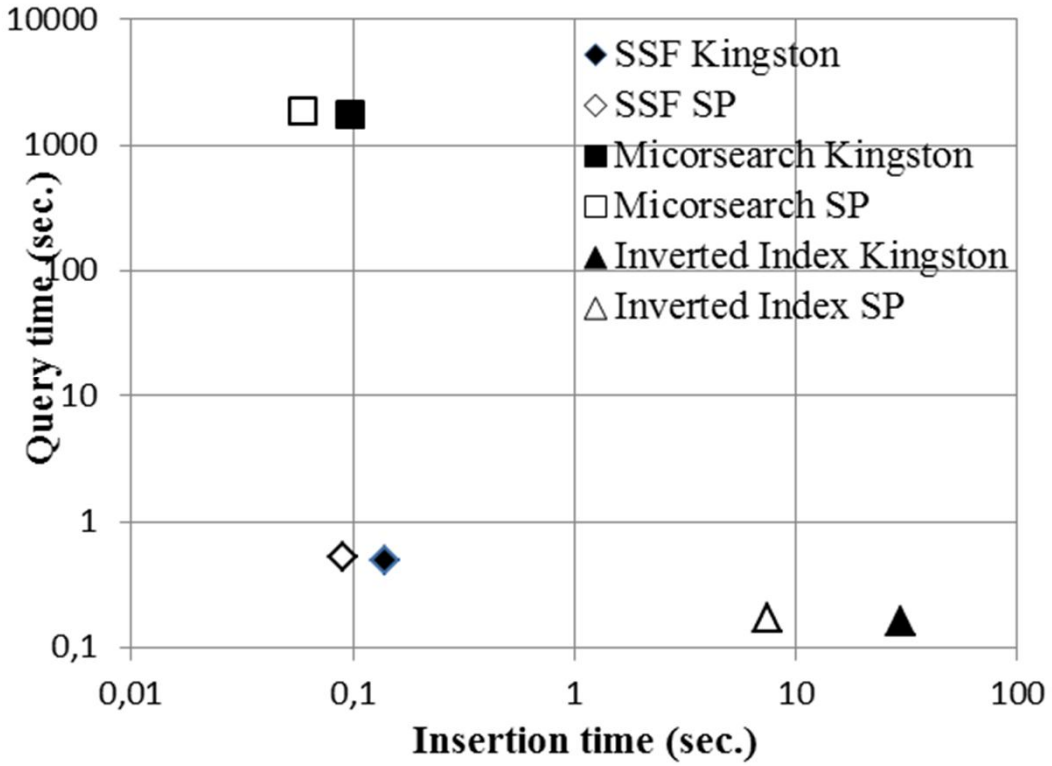


a. Average document insertion times of Microsearch, SSF and the Inverted Index with Silicon Power storage.



b. Query execution times with the Inverted Index, SSF and Microsearch with Silicon Power storage

Comparison with the State-of-the-Art Search Engine Methods



Overall performance comparison

Conclusion & Future Work

- We presented an embedded search engine for smart objects equipped with extremely low RAM and large Flash storage
- Our proposal is founded on three design principles, to produce an embedded search engine reconciling high insert/delete rate and query scalability
- Our inverted index supports document deletions, while the state-of-the-art embedded search indexes do not consider document deletions.

Future work :

- Efficient **tag-based access control** using the embedded search engine

- Apply our 3 designs principle (Write-Once Partitioning, Linear Pipelining, Background Merging) to other indexing structures for smart objects

Merci !

Questions/suggestions ?