Contiki Ring File System for Real-Time Applications

Sebastian Schildt, Wolf-Bastian Pöttner, Lars Wolf

PWSN 2012

GINSENG Project

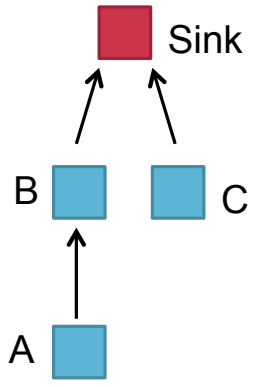
- Developing a WSN for industrial applications
- Petrogal oil refinery as testbed
- Deterministic performance and topology
- Ability to run closed-loop control applications

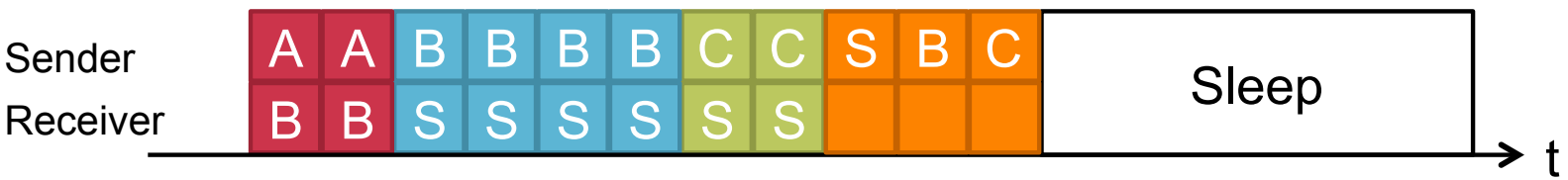
Real Time capable system required

GinMAC: GINSENG Communication

Fixed Tree Topology TDMA MAC

- Static tree toplogy
- GinMAC epoch schedules transmission slots
- Each node has some unused slots for processing





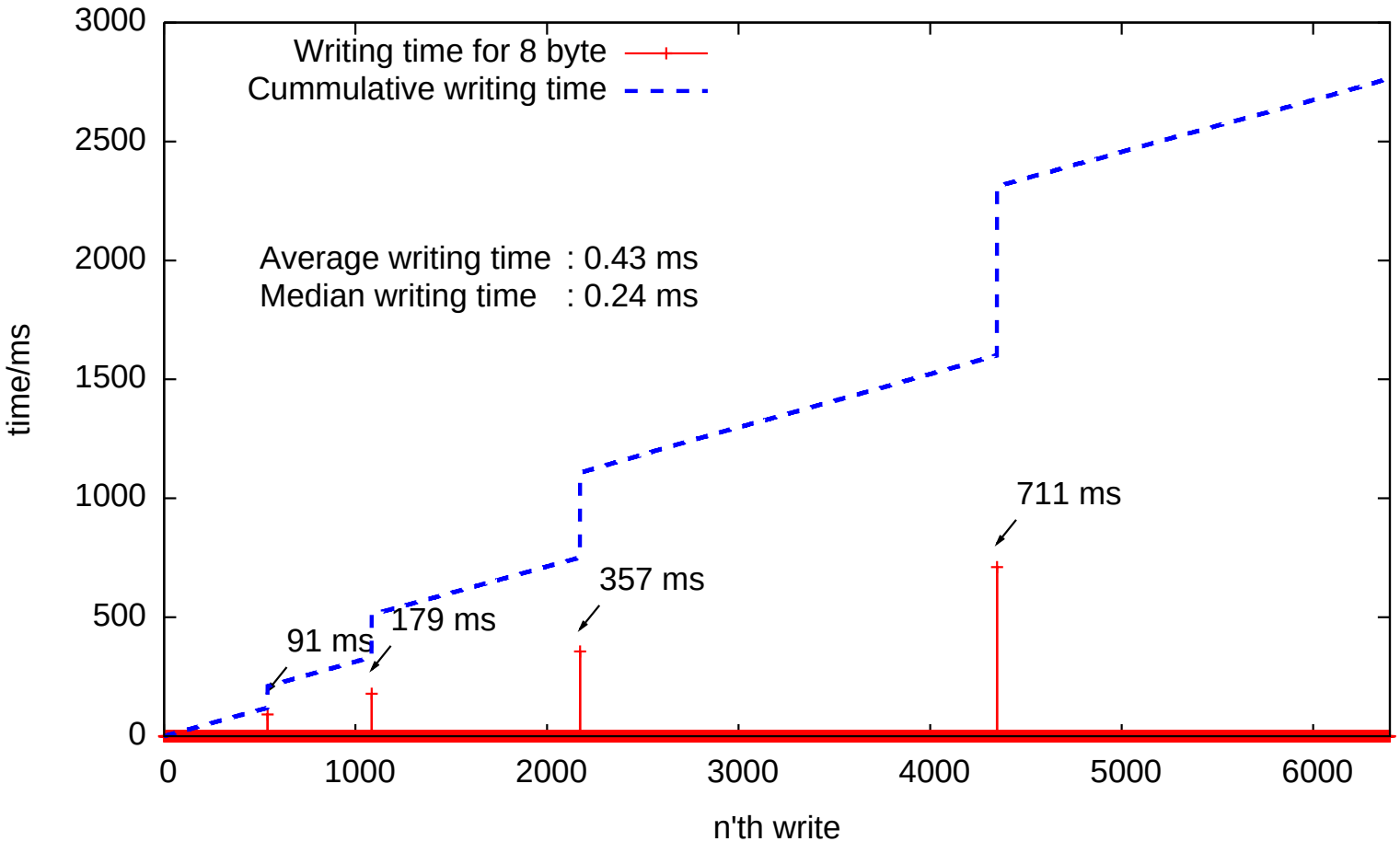
Storing data in WSNs

Many application want to use persistent data storage

- Usually directly on EEPROM, flash ICs
  $\rightarrow$ Page based, can only "program" bits of a clean page, not reset them, potentially big erase blocks
- Long term data logging
- Operational data and statistics not directly related to the applications
- Configuration settings

Contiki provides the COFFEE filesystem

COFFEE Write Times



Write 6000 x 8 bytes with a TMoteSky to a M25P80 flash memory

COFFEE and Real-Time

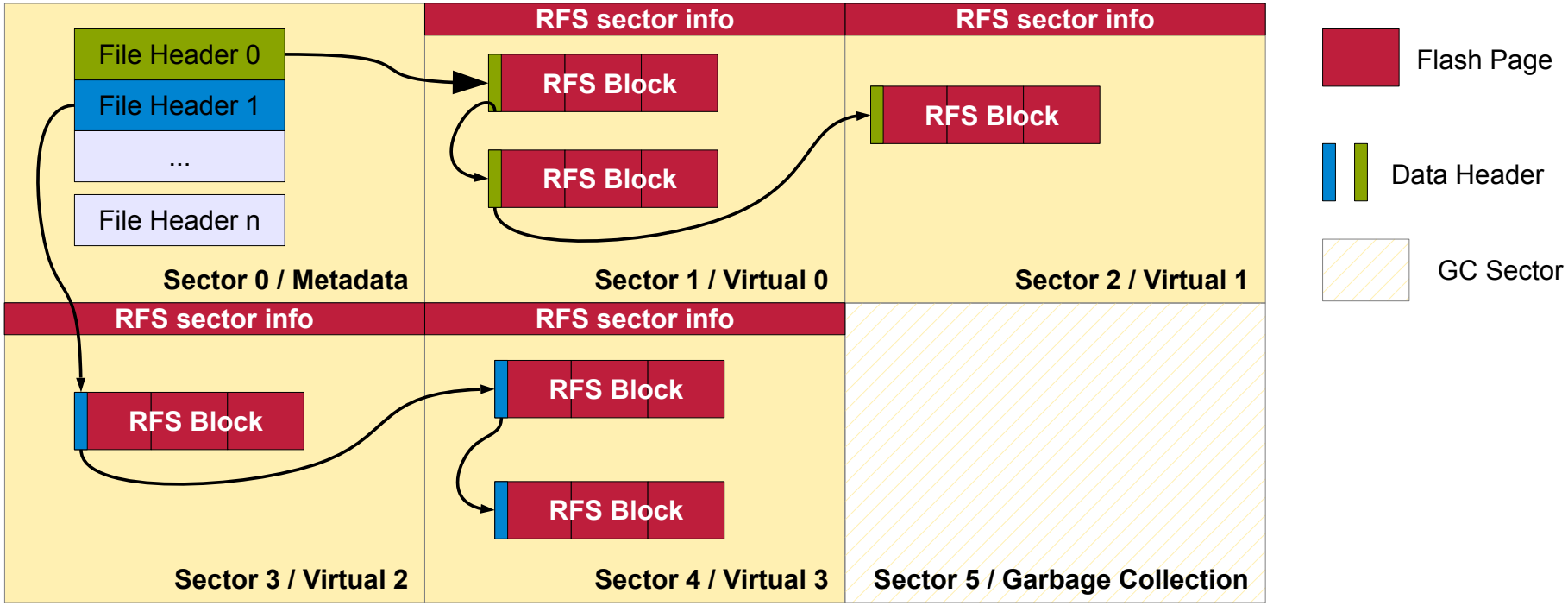
- log structured, using per-file logs instead of a global log
- needs all files to be stored sequentially in flash
- can only transparently extend files by copying their complete contents to a new file

→ If you do not know exactly what and how much you you are going to write, Coffee introduces long and hard to predict write delays

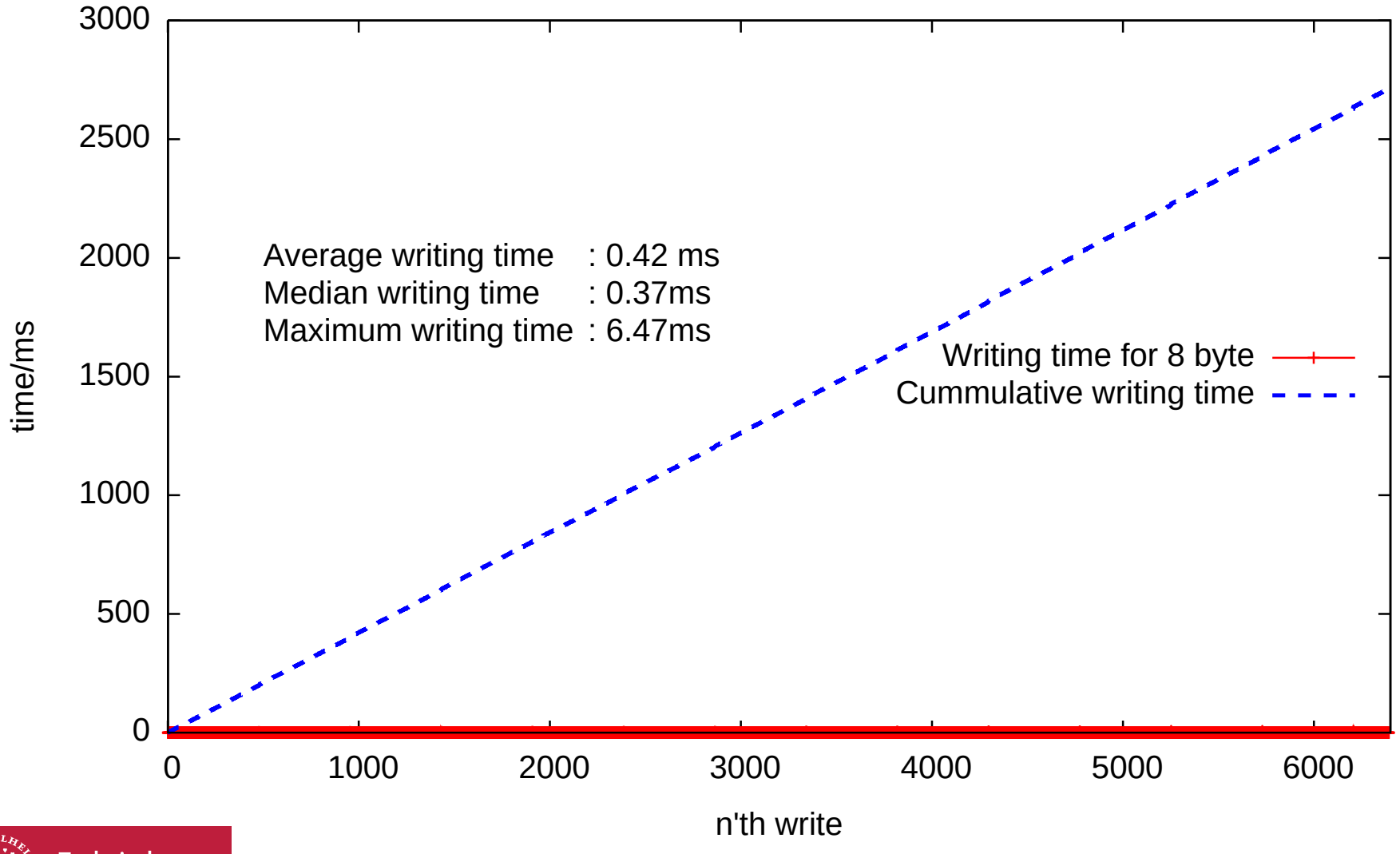
RFS Design Goals

- **Real-Time capabilities**
- Abstraction of a FIFO: Writes append data, reads consume data
- Efficient, even if file sizes and access patterns are not known beforehand
- Minmize copying of user data
- Adherence to CFS API
- Wear leveling

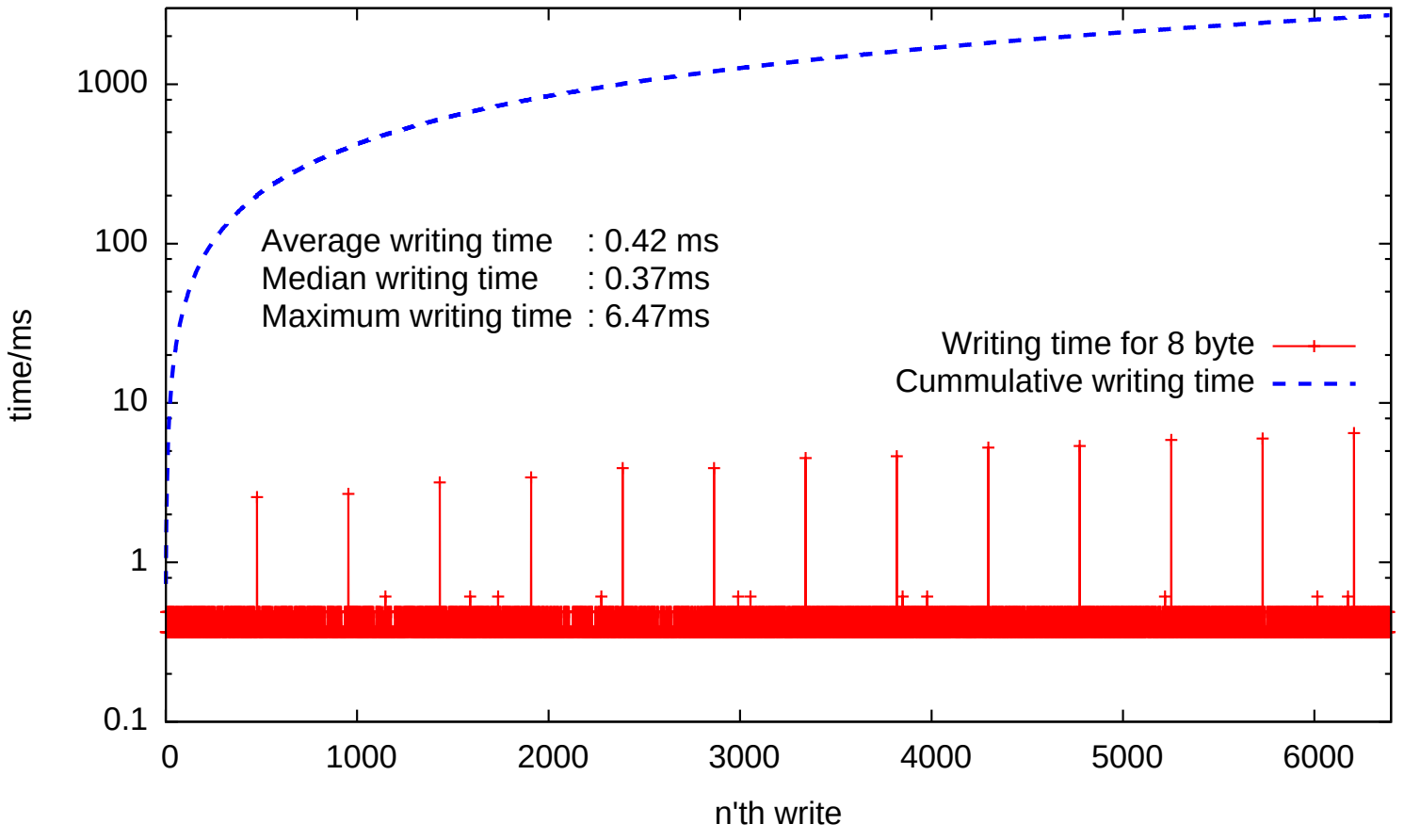
RFS Architecture



RFS Write Times



RFS Write Times: A closer look



Higher delays, when needing to allocate a new RFS block

RFS Write and Real-Time

- Normal write times already much more bounded than in COFFEE
- Extending files by adding another RFS block costs some extra time

→ RFS can be configured to never extend a file. Instead the user can query the remaining bytes in the current RFS block and schedule the extension when convenient

RFS Garbage Collection

- flash can only be erased in blocks, single bytes cannot be overwritten
- garbage collection is needed to free up used flash areas

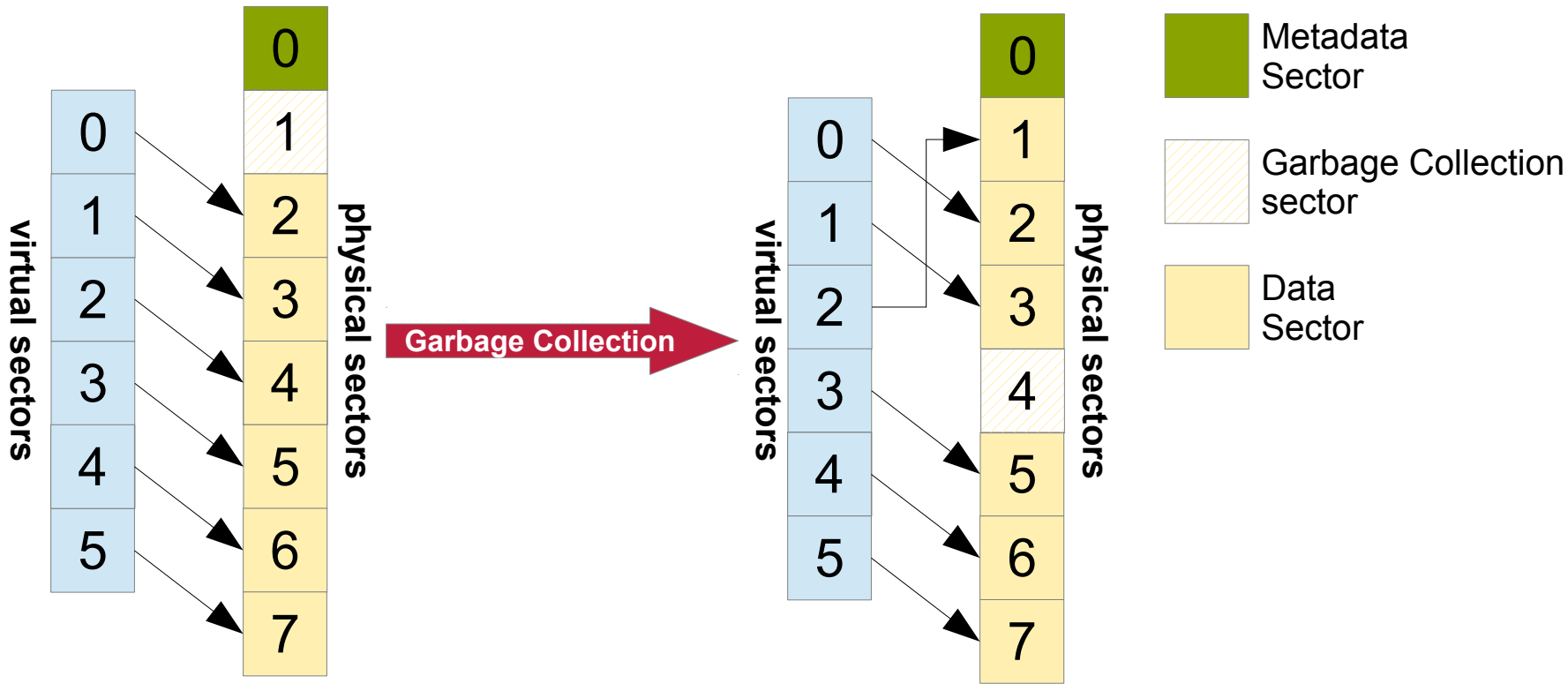
Metadata Sector

Copy life entries from header sector to garbage collection sector. Erase old sector and mark it as new garbage collection sector

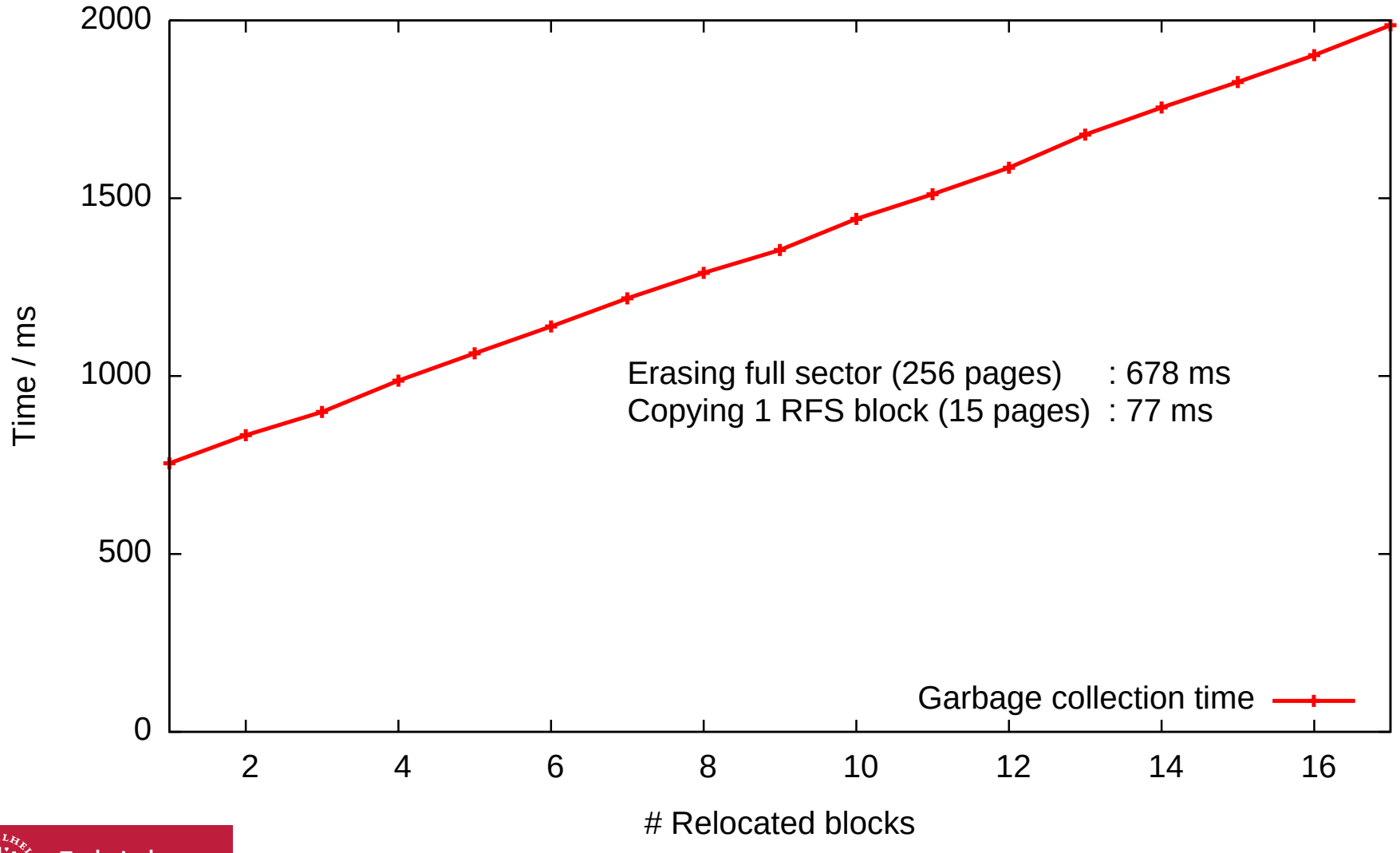
Data Sector

Copy RFS blocks with active data from data sector to garbage collection sector. Erase old sector and mark as the new garbage collection sector

RFS Virtual Sector Mapping



Garbage Collection Times



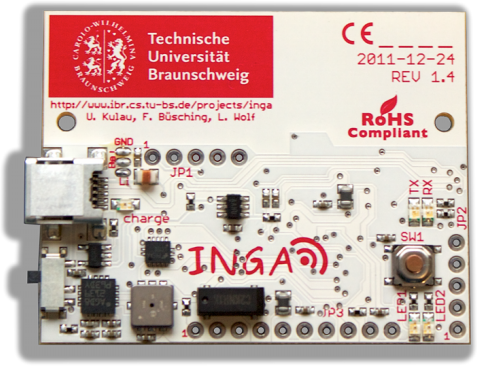
RFS Garbage Collection and Real-Time

- Garbage collection takes a long time compared to other operations
- It will not occur very often.

→ Automatic garbage collection can be disabled. The application can schedule the garbage collection when appropriate.

Advertisement: The INGA Node

The fully open-sourced **I**nexpensive **N**ode for **G**eneral **A**pplications (**INGA**) node has been developed with the motivation to fill in as a more modern, versatile successor to the TMoteSky platform



- ATmega 1284p uC with 128 kiB Flash, 16 kiB SRAM, 4 kiB EEPROM
- AT86RF231 IEEE 802.15.4 Transceiver
- Various sensors: Accelerometer, Gyroscope, Pressure Sensor, Temperature Sensor
- MicroSD Card Slot, LiPo charge controller

Future Work and Conclusions

- RFS provides a Real-Time capable filesystem for buffering tasks
- Allowing the user to schedule potentially long running tasks, allows applications to meet harder real-time constraints
- Some Opportunities for future work:
  - Allow data to stay valid after being read (configuration, OTA updates)
  - More features: Directories, multi-GB scalability?
    → Stay tuned: Real-Time Capable FAT implementation on its way

谢谢! **Thank you!**