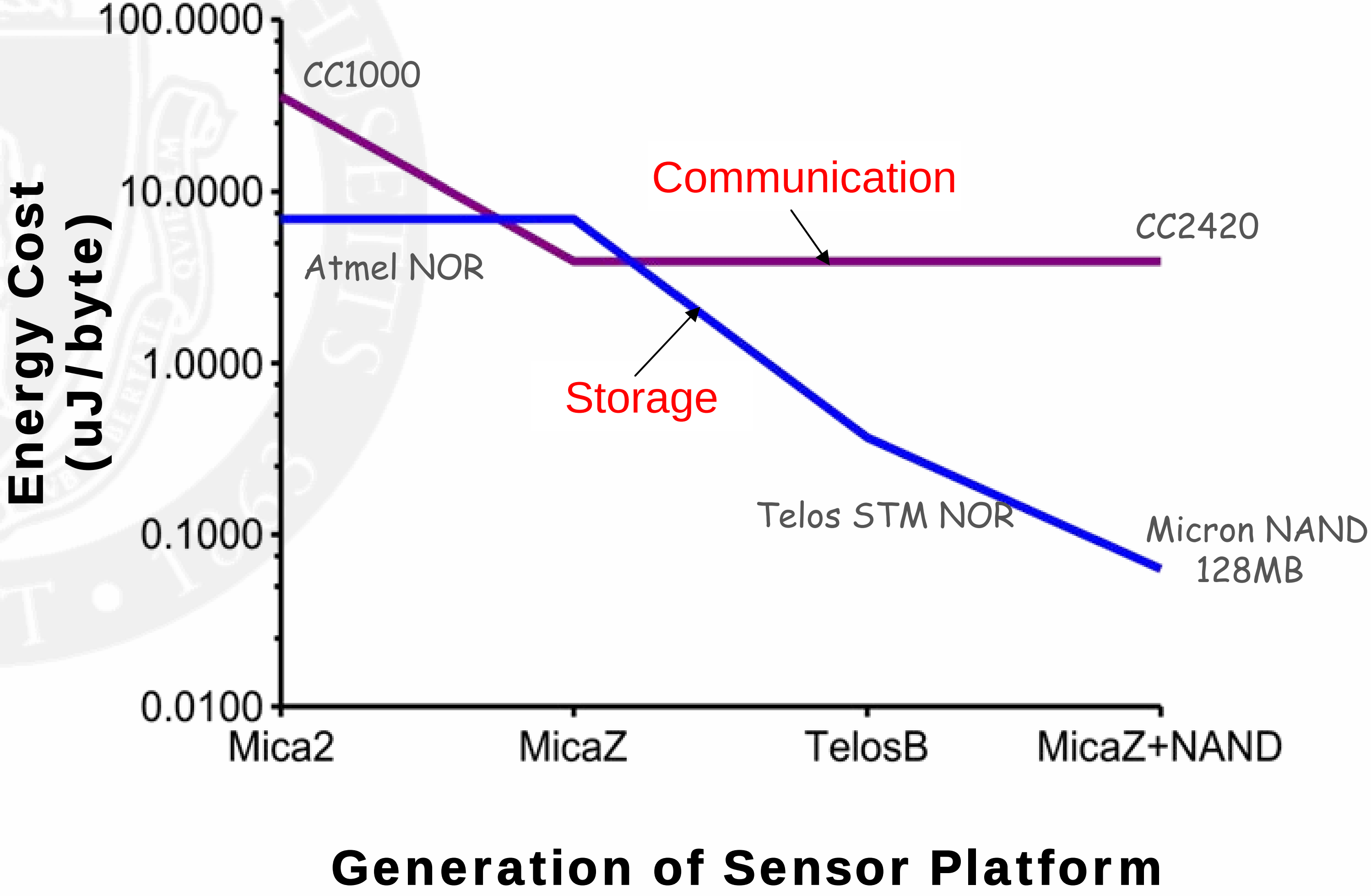
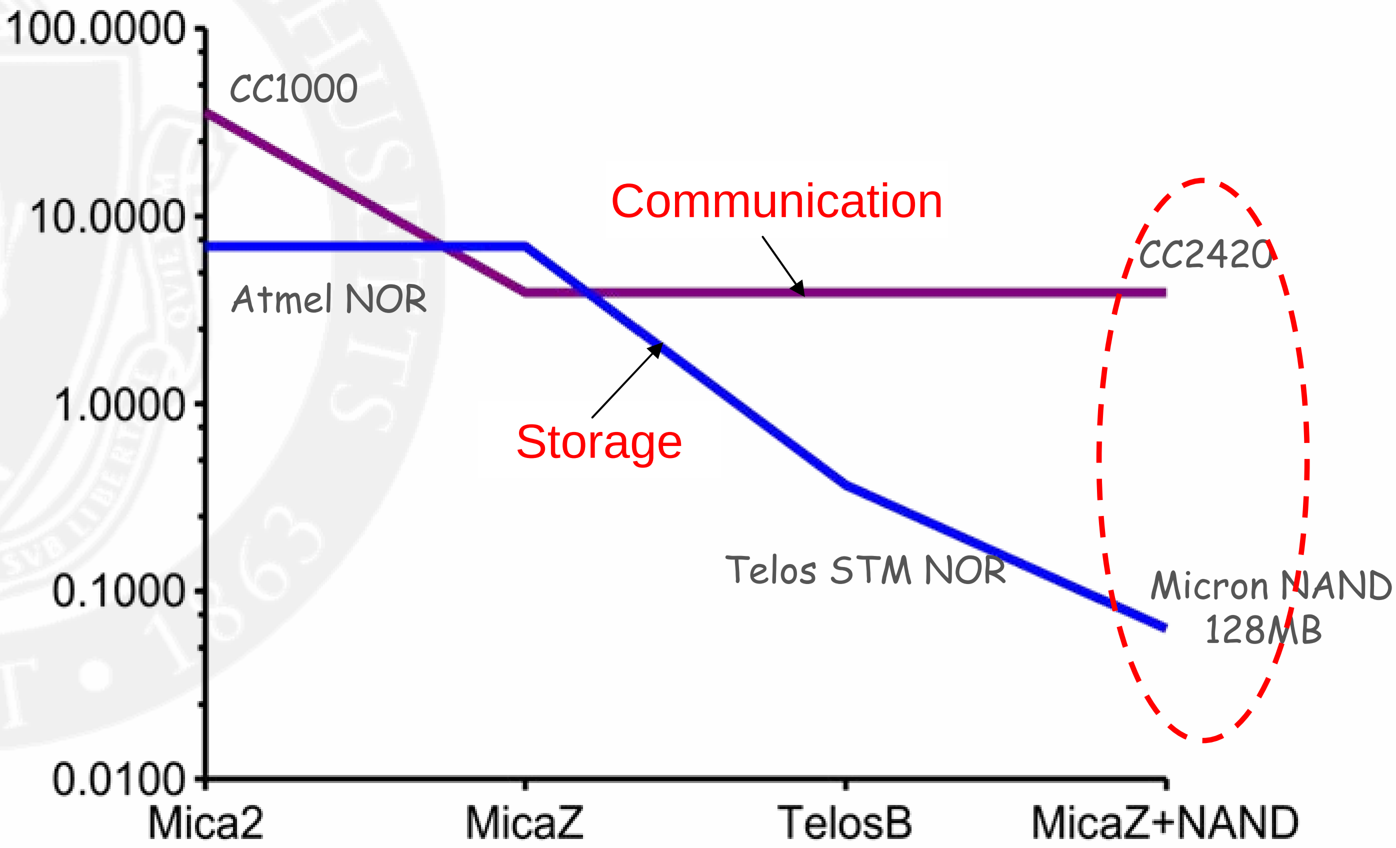
Capsule: Energy-optimized Object-based Storage for Sensors

Gaurav Mathur (presenter), Peter Desnoyers
Deepak Ganesan, Prashant Shenoy
University of Massachusetts, Amherst

Technology Trends in Storage

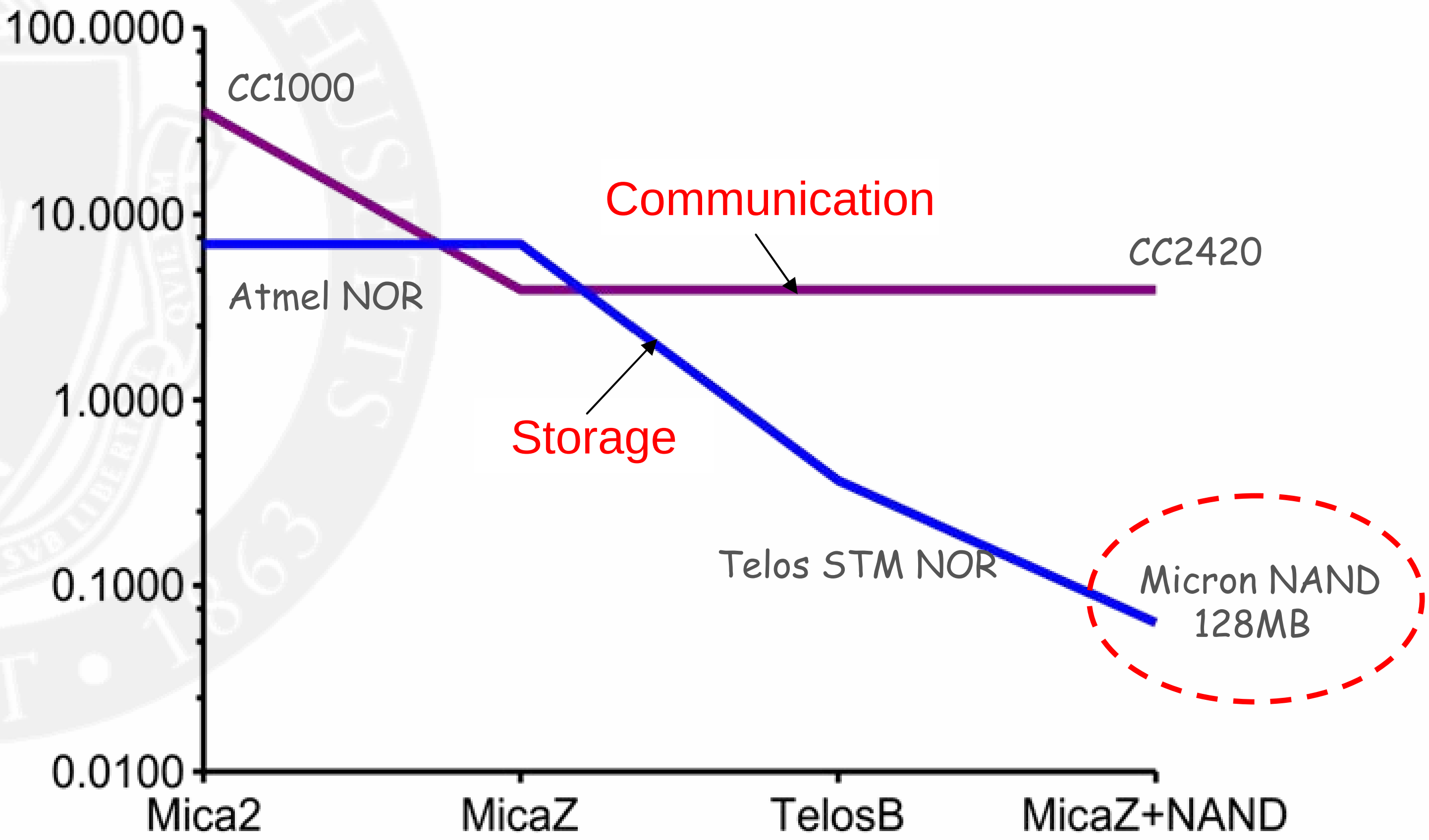


Technology Trends in Storage



Storage has become more than two orders of magnitude more energy-efficient than communication.

Technology Trends in Storage

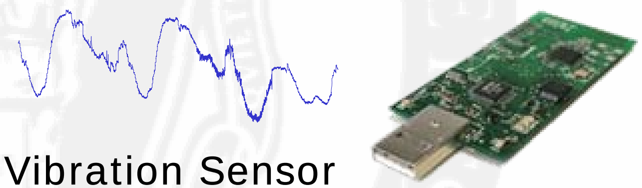


NAND flash enables extremely high capacity sensor storage (1 Gb)

Uses of Cheap, High Capacity Storage

In-network Data Storage

Data Archival
& Indexing



Vibration Sensor

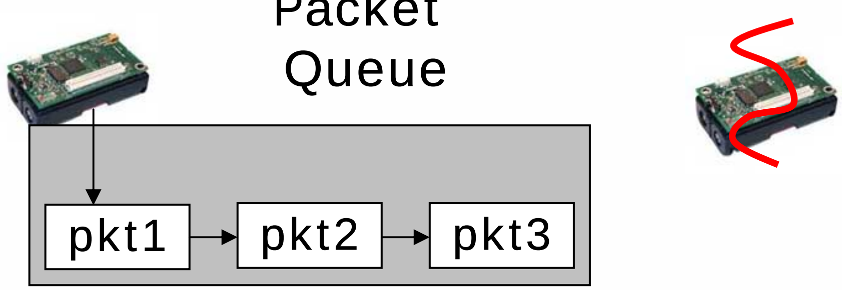
In-network Data Processing

Signal
Processing

Compression

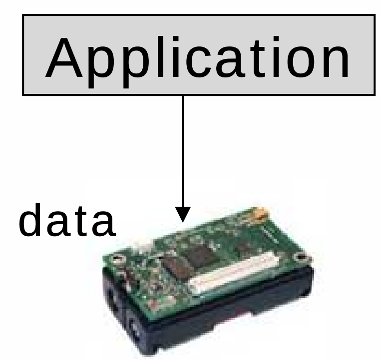
Support Disconnected Operation

Packet
Queue

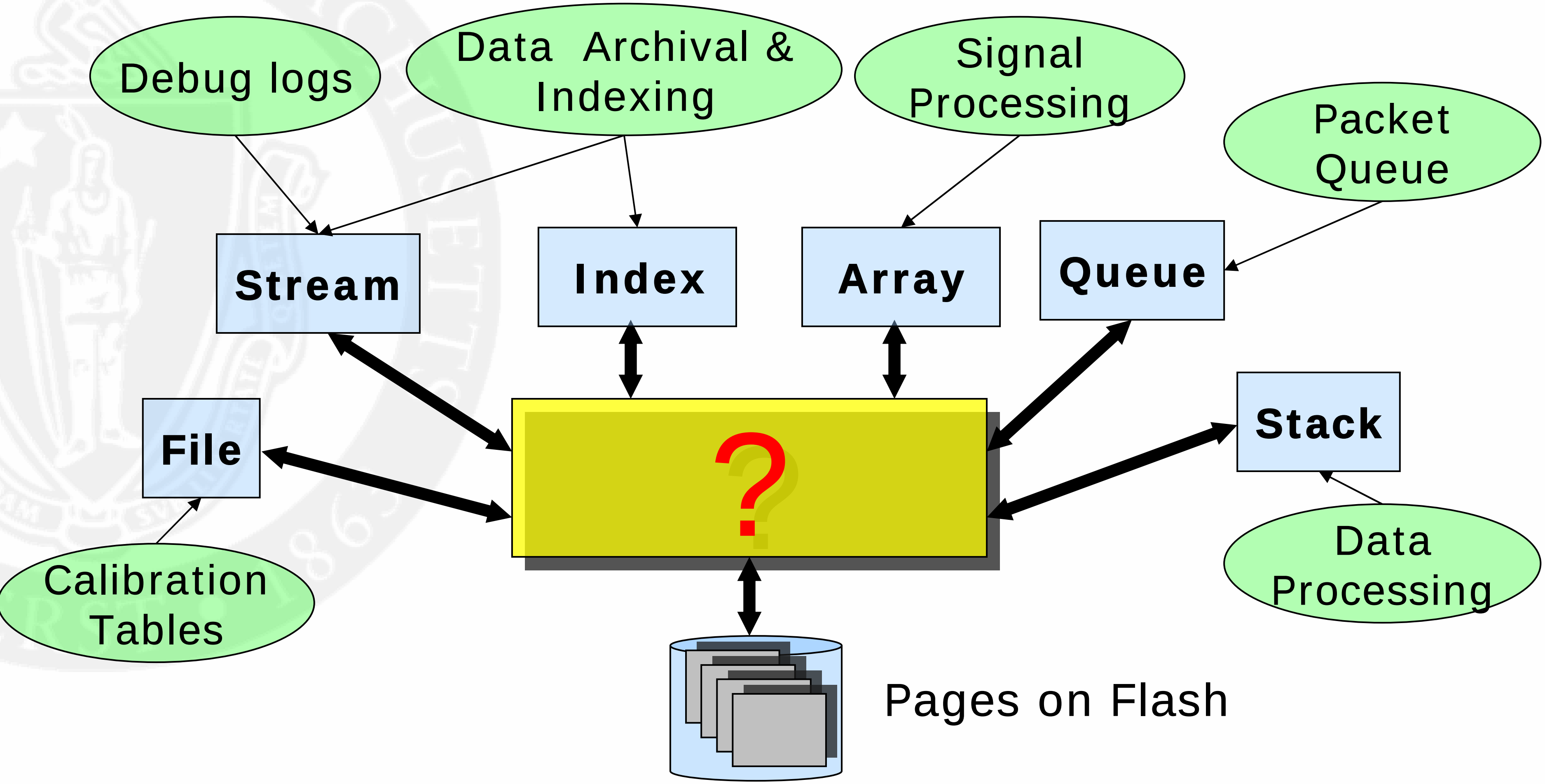


Other uses

Logs,
Calibration data

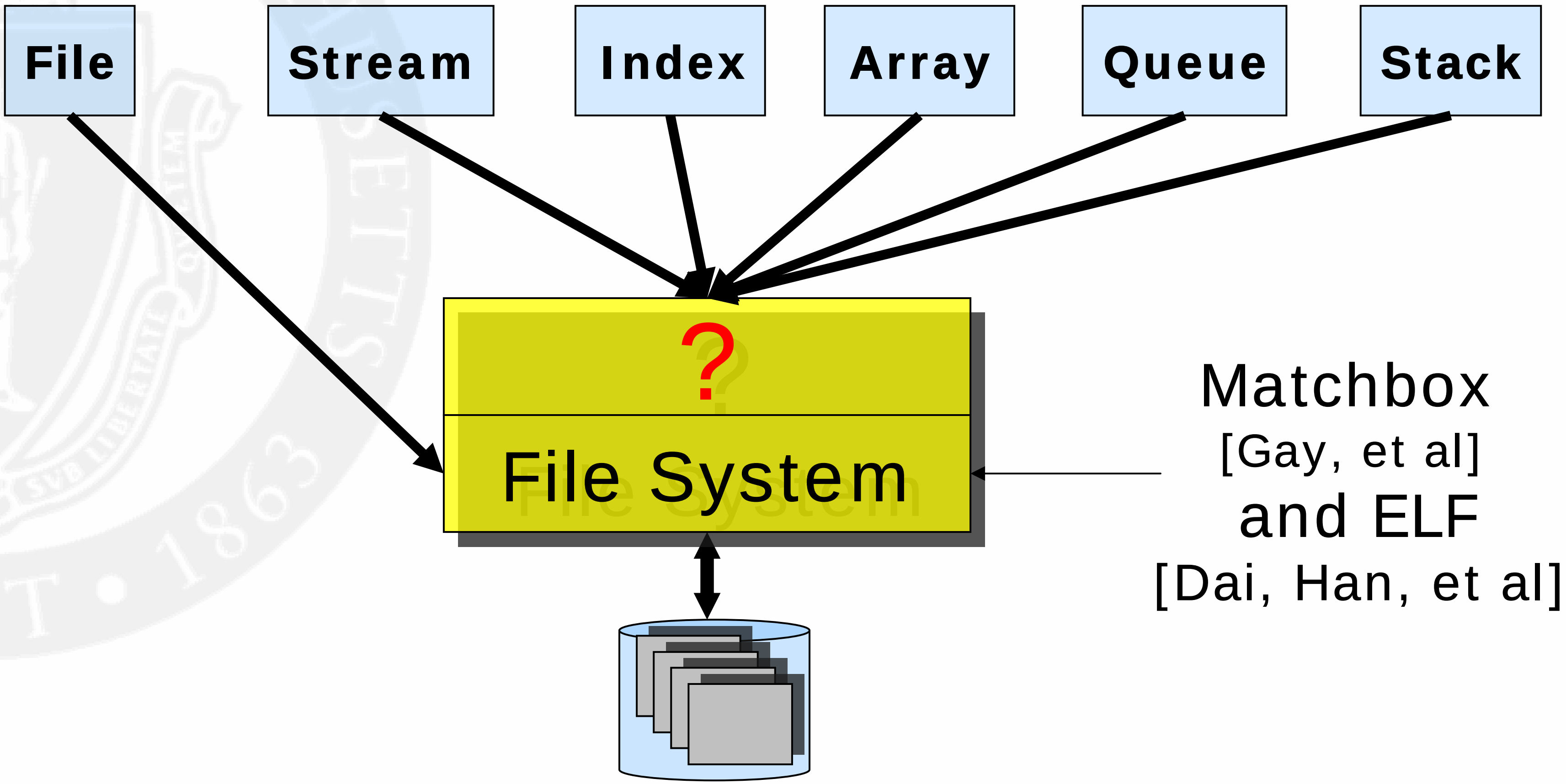


Mapping App Data Needs to Storage



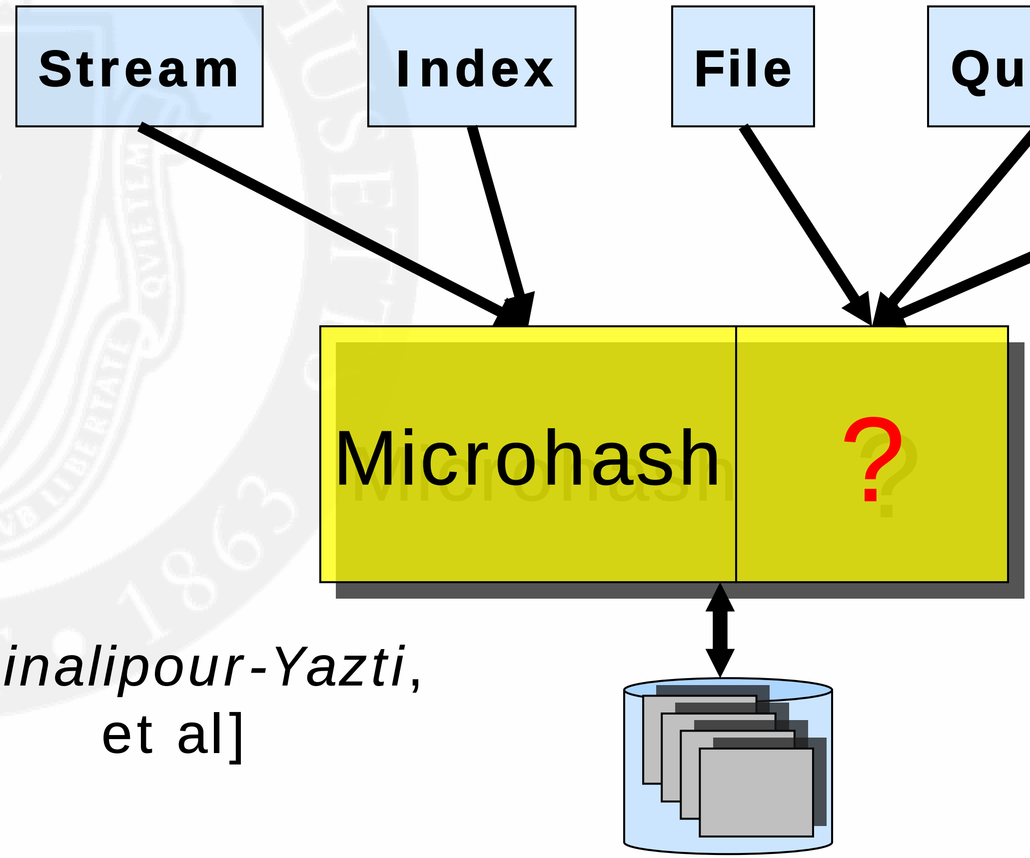
How do we map data structures needed by the application to how they are stored on flash ?

Use a File System ?



Problem reduces to mapping data structures to a *File System*...

Microhash



Stream storage and indexing

[*Zeinalipour-Yazti*, et al]

Only supports storage of a stream and some indexes on it; Design lacks flexibility…

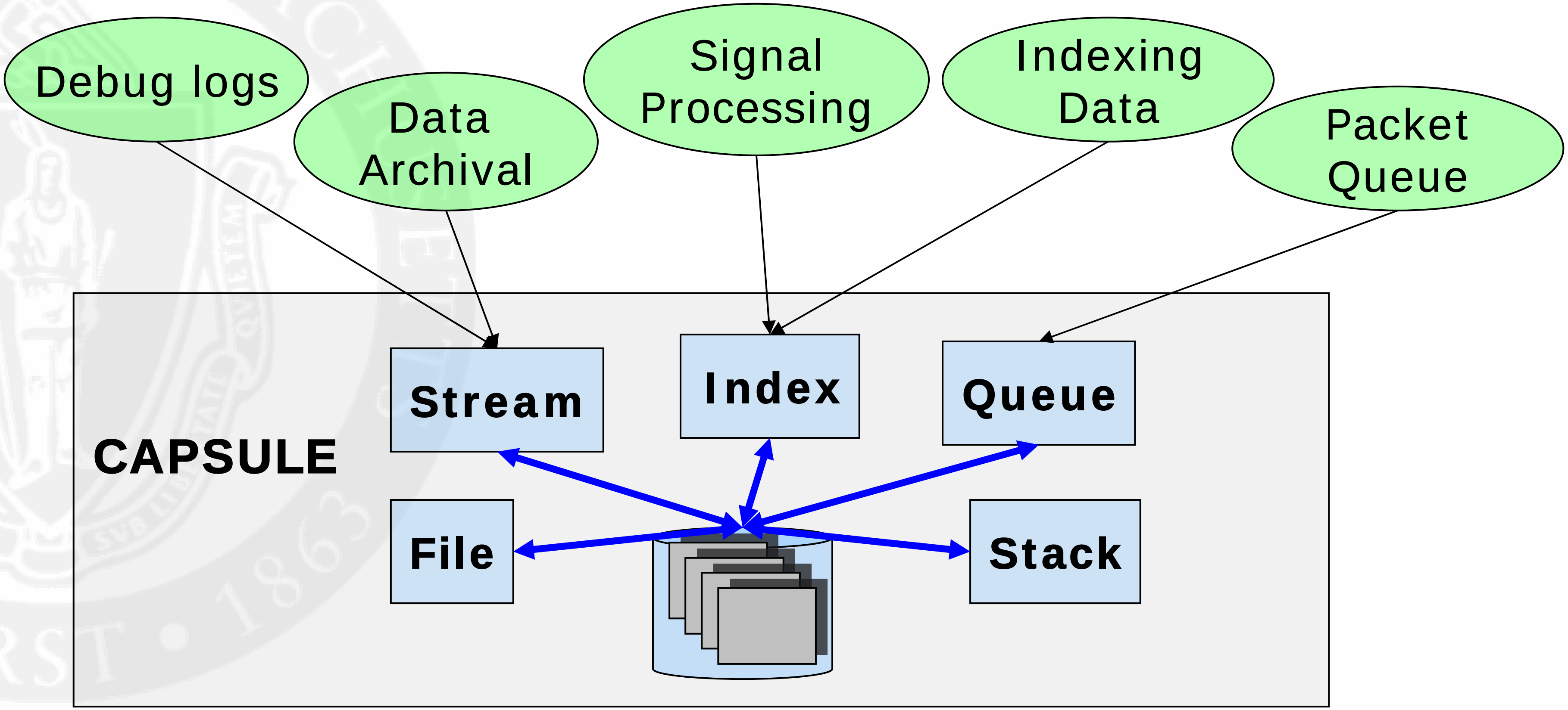
Storage System Requirements

1. Support flexible and varied data storage
   - Stream archival & indexing, Packet queues, debugging logs, calibration tables...
2. Energy and Memory optimized design
3. Recover state after system failures
   - Unreliable hardware; deployment conditions harsh
4. Deal with finite storage capacity
5. Portability across flash technologies
   - Support both NOR and NAND flash memories

Outline

- Motivation
- Requirements
- Capsule Architecture
- Implementation
- Evaluation
- Capsule Deployments
- Conclusion

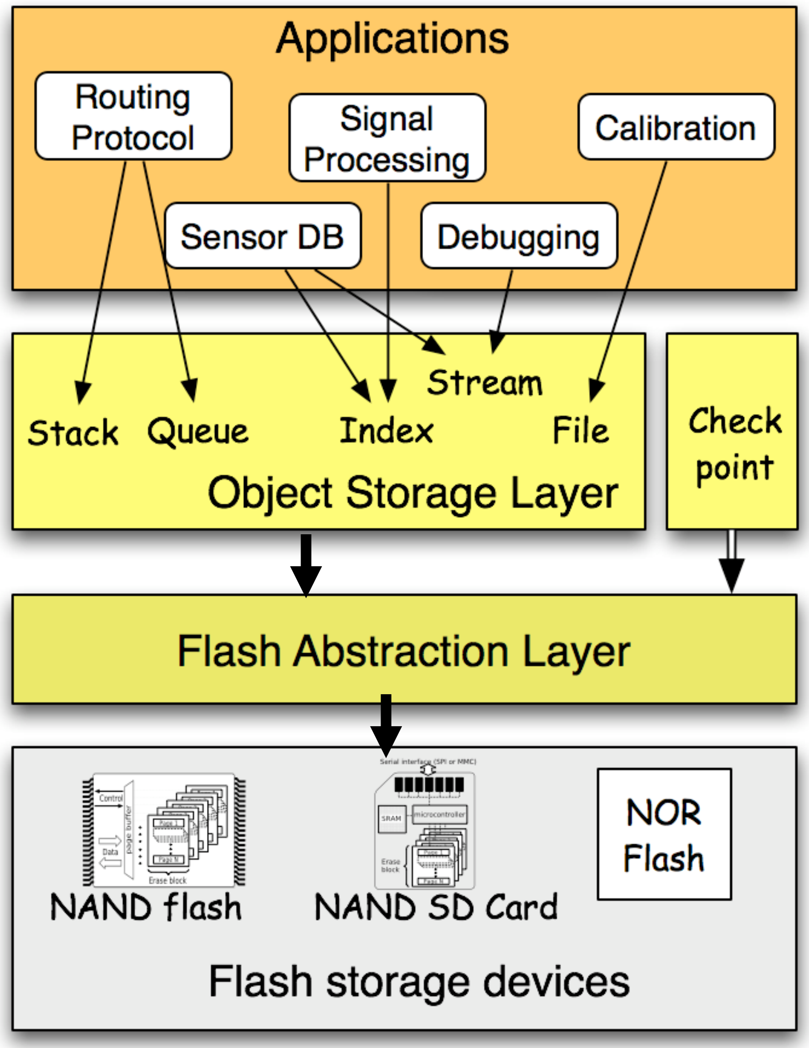
Mapping App Data Needs to Storage



Capsule provides storage objects to the application
a) Natural fit for requirements
b) Optimized for sensor platforms

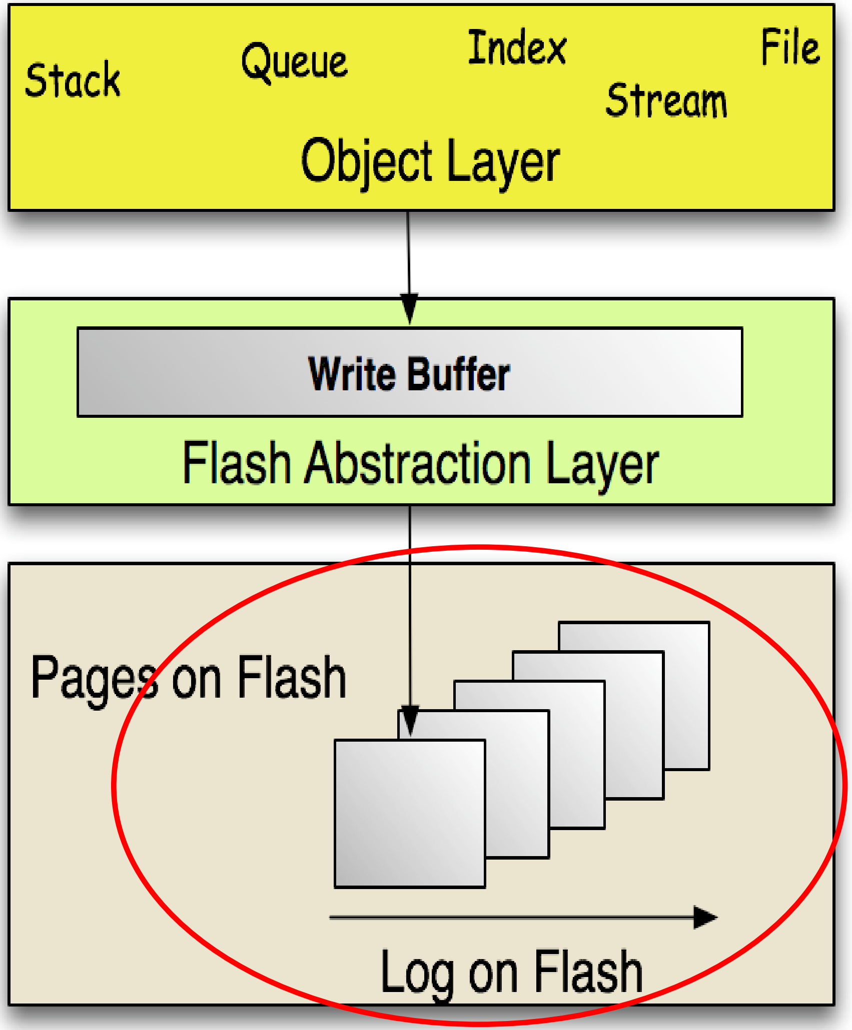
Capsule Architecture

- Object-based storage abstraction
- Energy and memory optimized library of objects
- Checkpointing and rollback for failure recovery
- Storage reclamation to deal with finite storage capacity
- Portable to multiple flash platforms



Flash Abstraction Layer (FAL)

- Flash Memory Constraints
  - Data cannot be over-written, only erased
  - Pages can often only be erased in blocks (16-64KB)
  - Sensors have limited memory (~4-10KB)
  - Unlike magnetic disks, *cannot modify in-place*
- Design : Log-structured storage
  - Treat flash as a write-once, append-only *log*

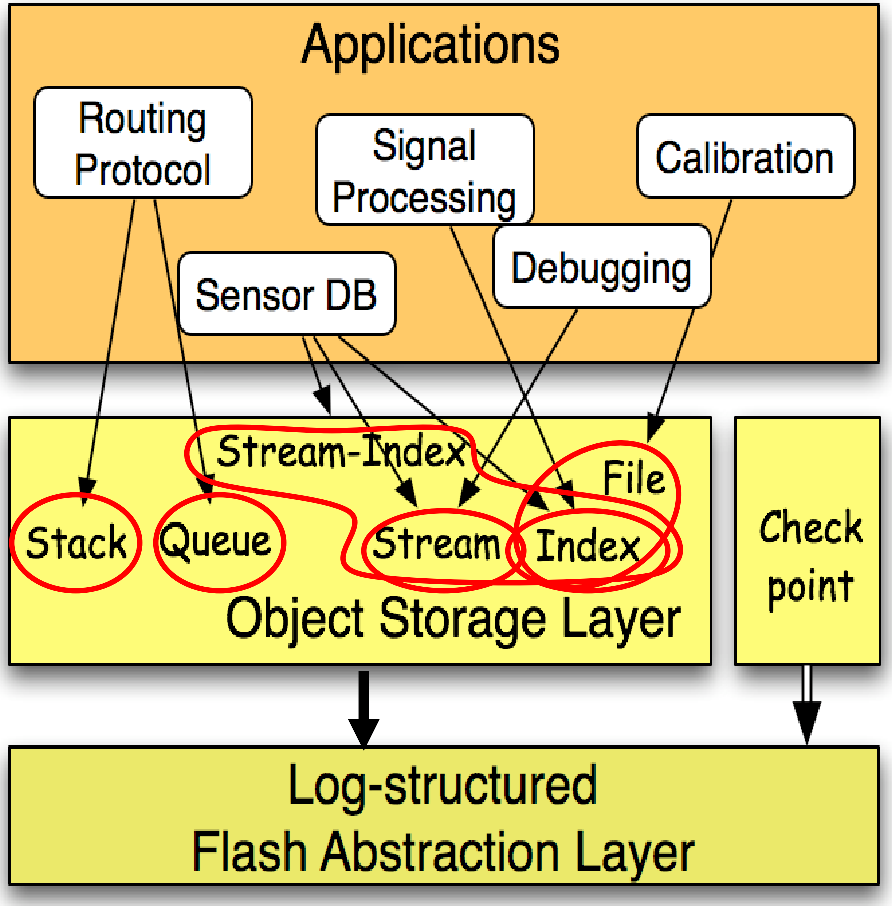


Capsule Objects

- Storage "objects" exposed to application

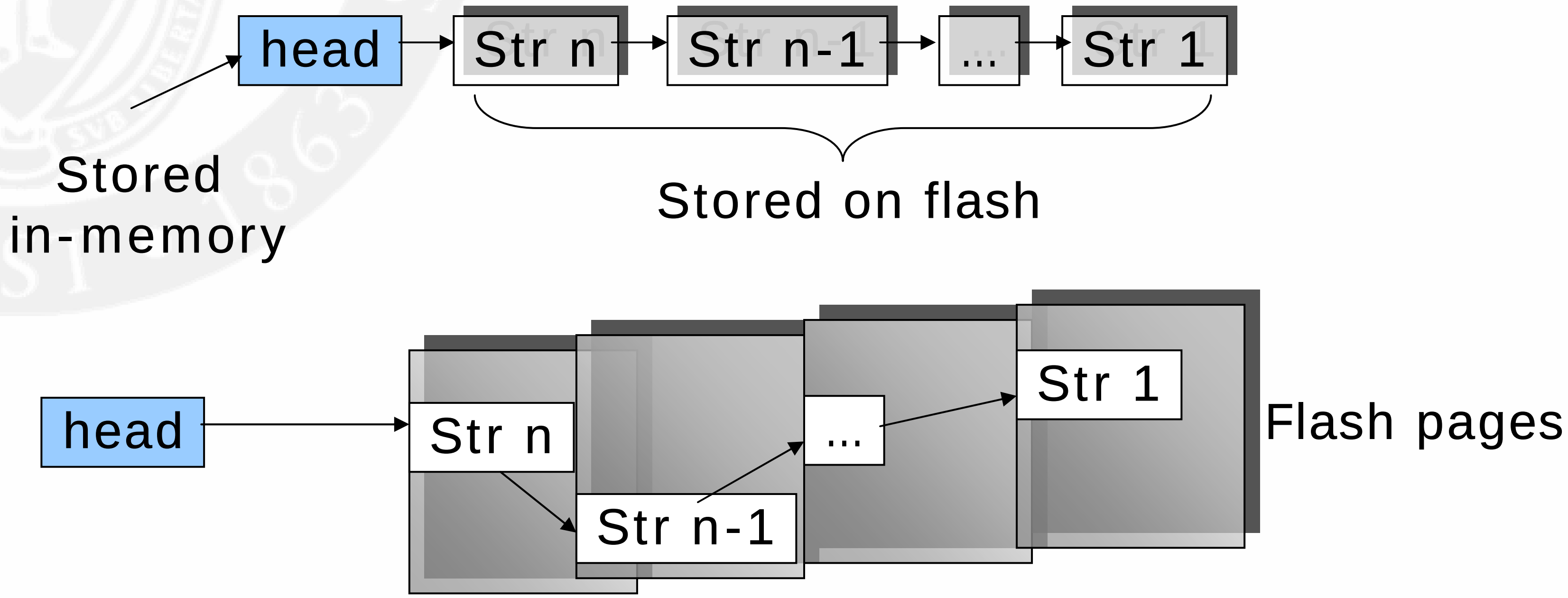
Object classification

- *Core objects* provide optimized implementations
- *Composite objects* can be created from core objects
  - *E.g.* Stream-Index object



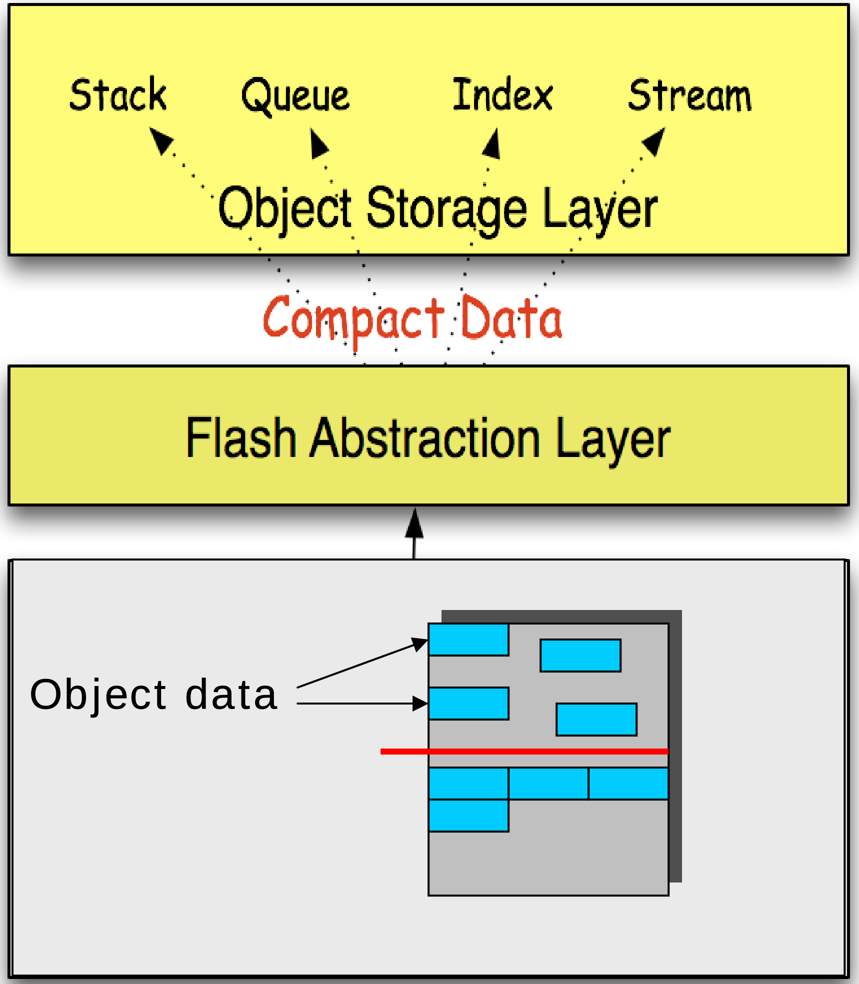
Stream Object

- Stream stored as a reverse linked list
  - Cannot modify pointer of elements already written to flash
  - Maintain in-memory pointer to last element

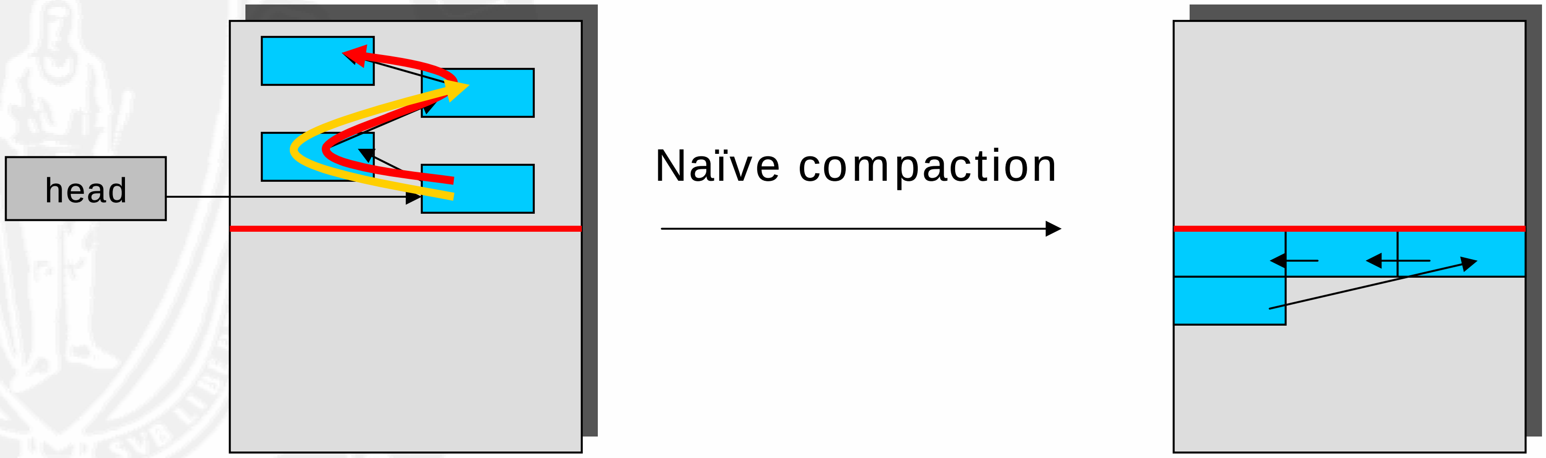


Storage Reclamation

- Storage reclamation needed
  - Flash has finite capacity
- Reclamation needs to be done by Objects
  - Differs from *segment cleaner* used in log systems
  - FAL does not understand object structure
- Use *compaction* technique



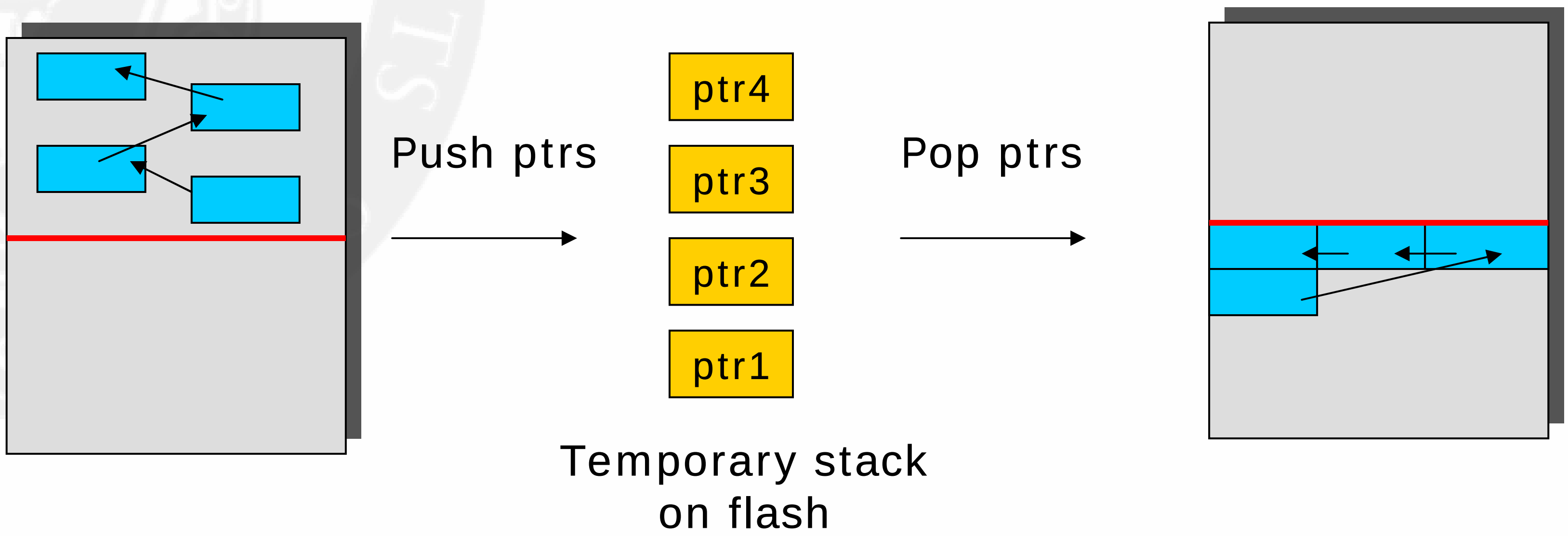
Stream - Reclamation



- Pitfalls with this approach
  - Not energy-efficient: need to traverse the stream multiple times
  - Insufficient memory prohibits buffering entire stream

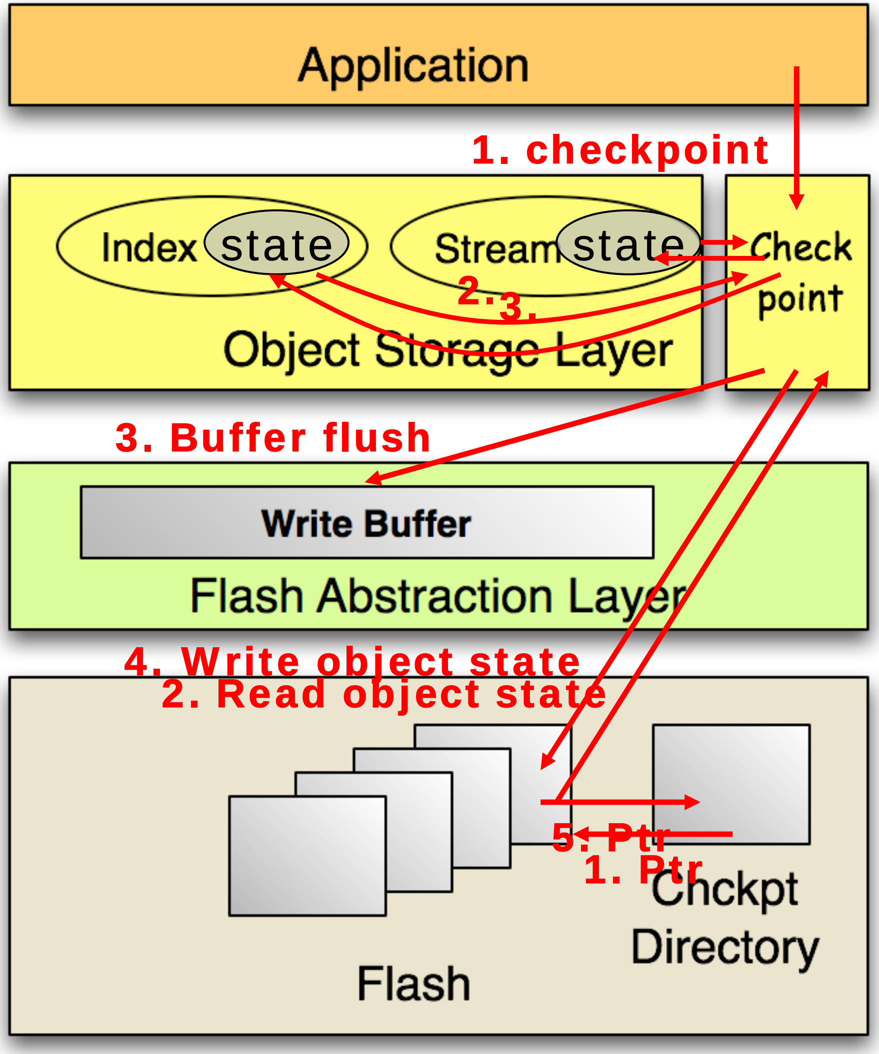
Stream - Reclamation

- Our approach: use a temporary stack on flash
  - Read stream element, push ptr to it onto stack
  - Pop ptr, read entire stream element and write to the compacted stream



Failure Recovery

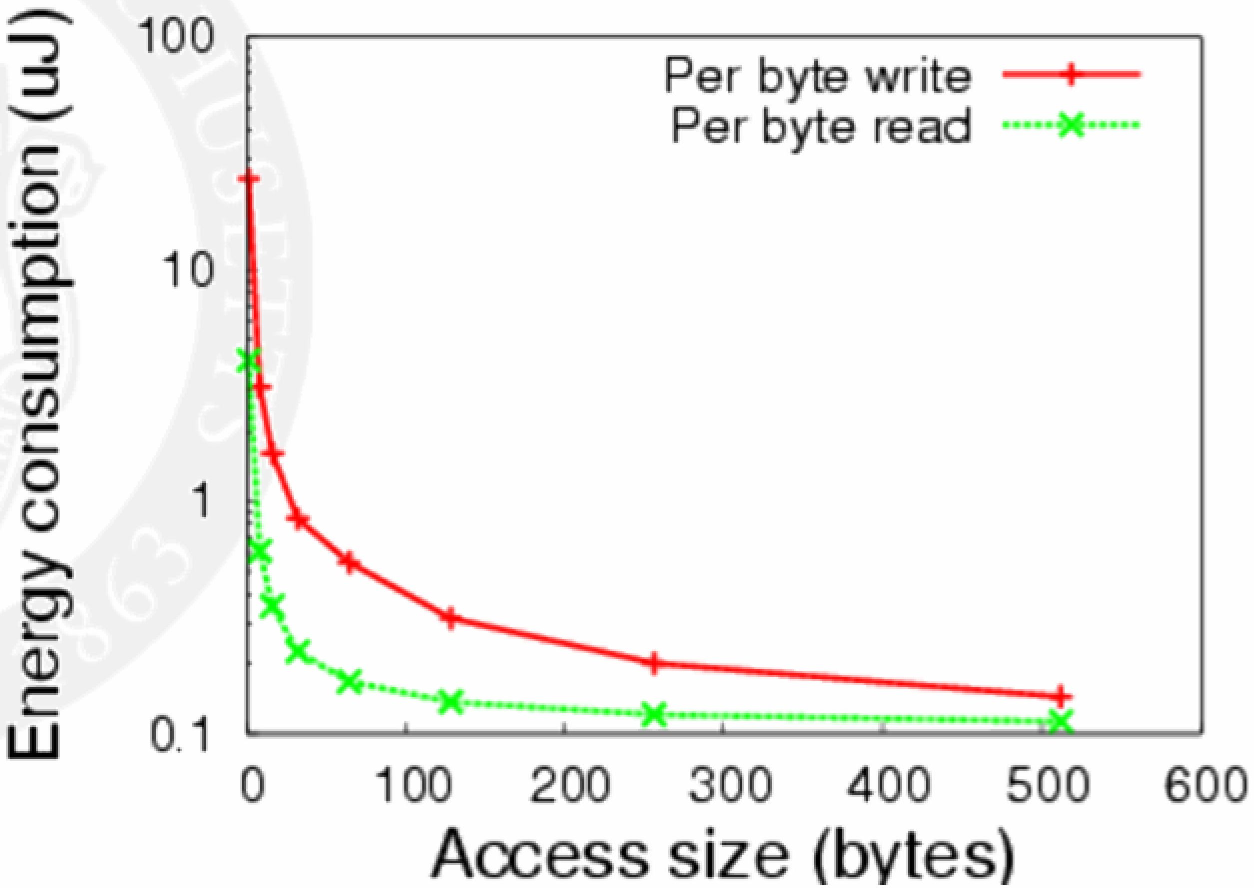
- Use checkpointing and rollback
- Checkpoint: Capture System snapshot
  - Data written to flash (cannot be modified)
  - In-memory object state
- Rollback: Load snapshot from flash and restore object in-memory state



Implementation

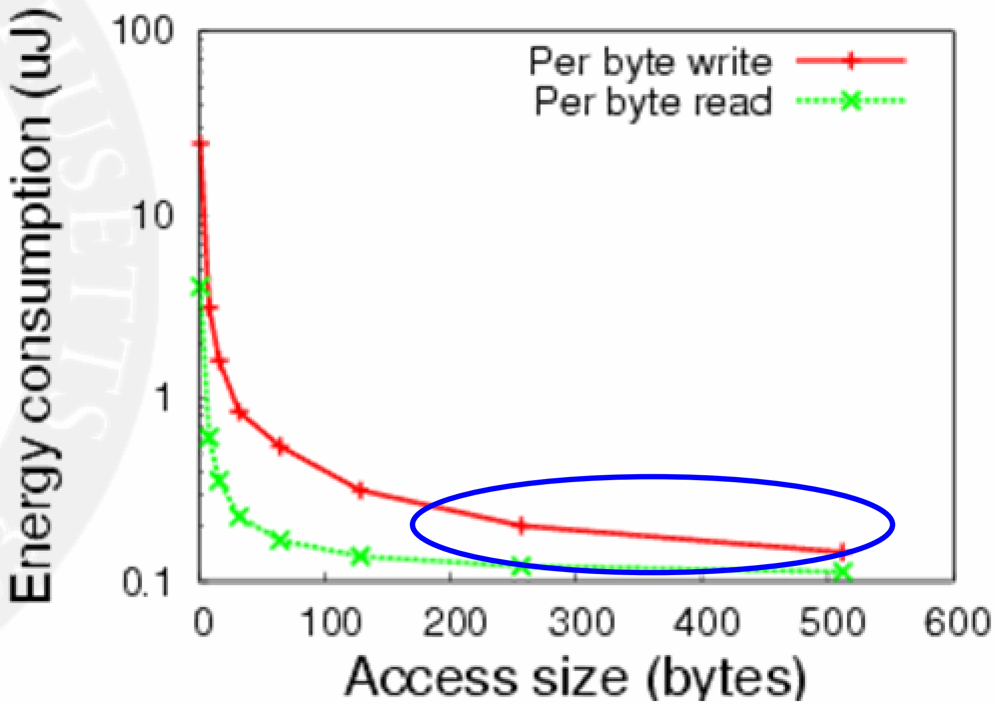
- Implemented in TinyOS 1.x
- Flash devices currently supported
  - NAND flash for Mica2/MicaZ
  - Telos ST NOR flash
  - Mica2 NOR flash
  - Easy to port to other platforms...
- Evaluation
  - System benchmarks
  - Application
  - Comparison with Matchbox

Read / Write Benchmarks



Key Observation: Cost of write operation substantially more than read

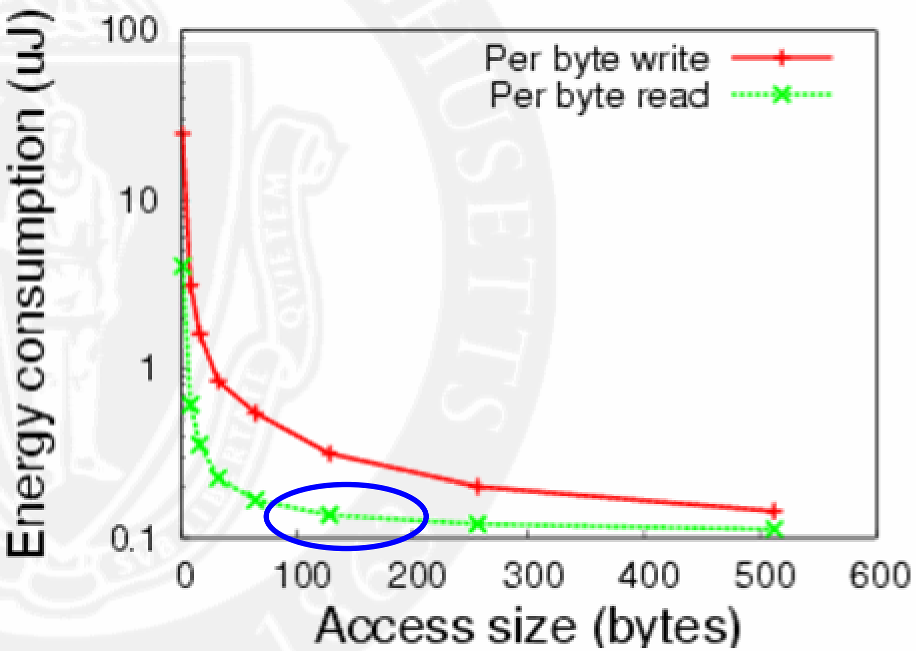
FAL Write Buffer Sizing

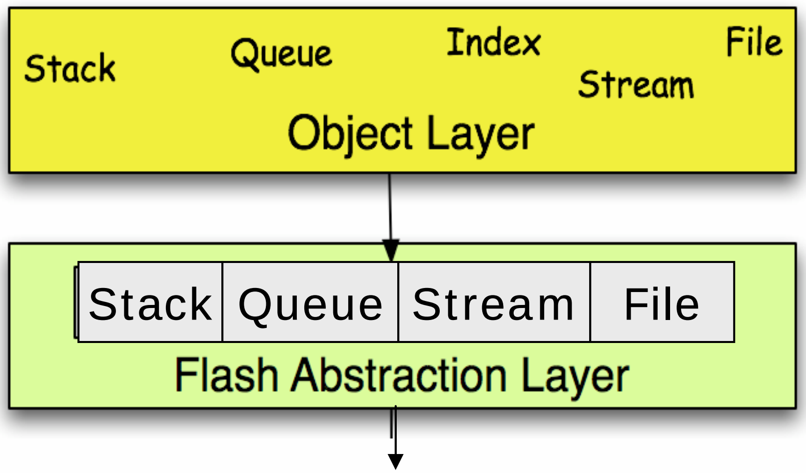


Use maximum size write buffer at FAL

- All objects share this common write buffer
- High cost of write amortized over more bytes

Read Buffer Sizing

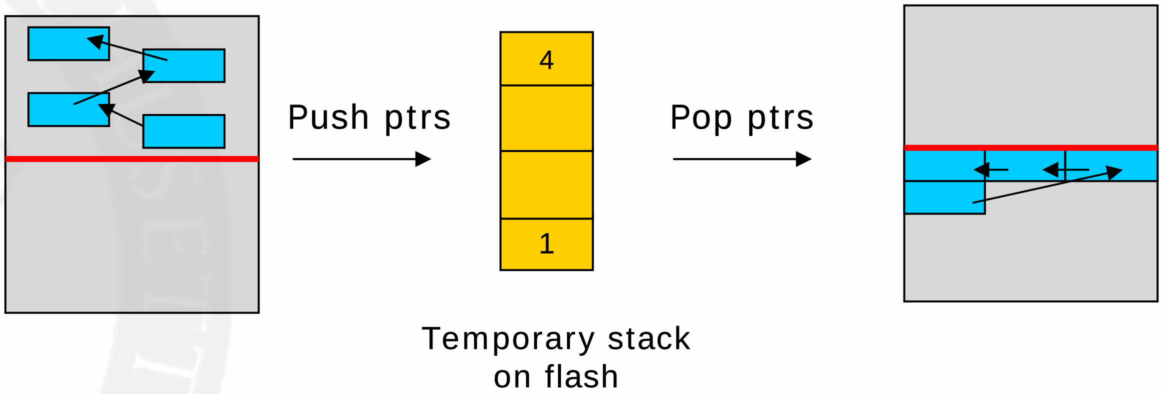




Read buffering

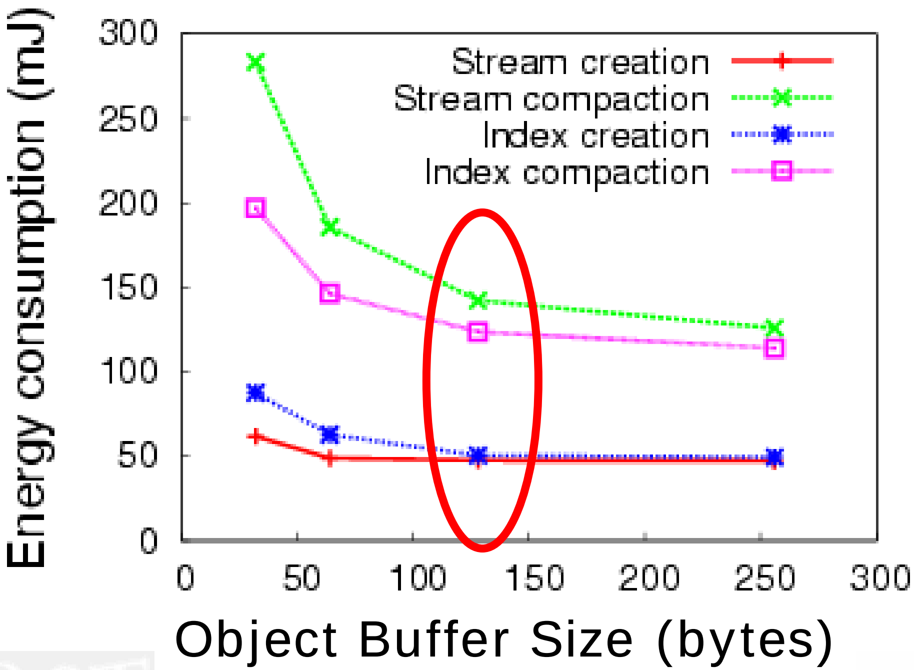
- Not performed at FAL since gains are limited
- Limited object level buffering sufficient to minimize energy cost of read

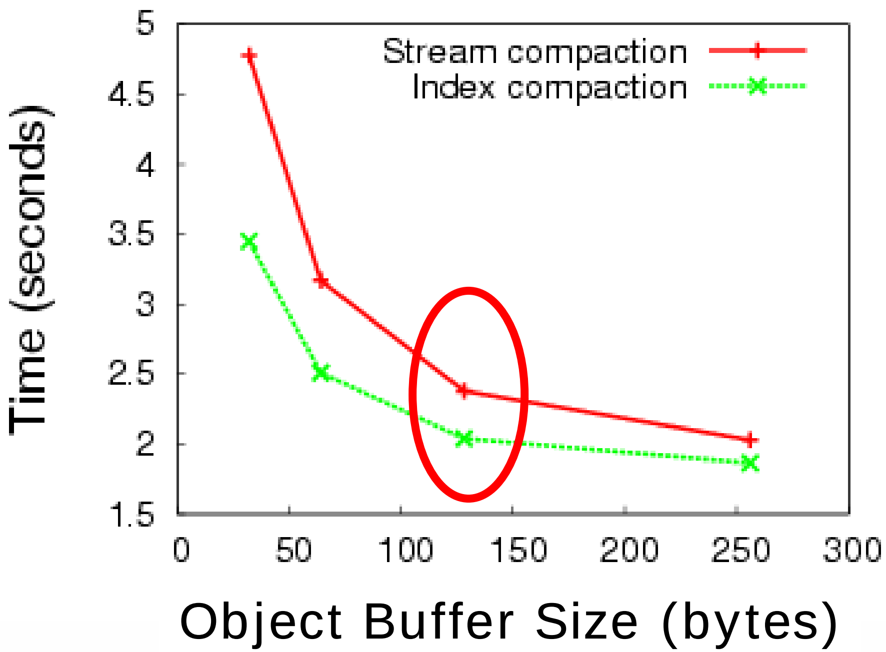
Performance of Compaction



- Goal: Measure performance of compaction
- Experiment
  - Store 128KB of data in Stream & Index objects
- Parameter
  - Clear that the number of elements impacts the performance
  - Vary buffer size at each object between 32 and 256 bytes

Performance of Compaction

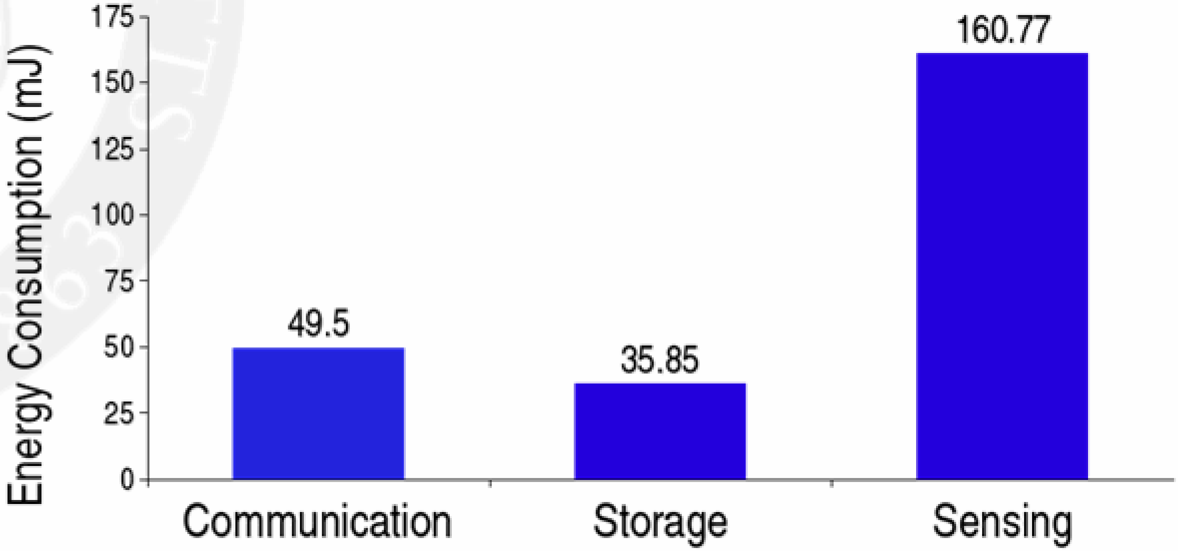




Compaction is fairly cheap even on large datasets
Buffer size of 128 bytes is a sweet spot

Performance of Representative App

- Application Evaluation on Mica2 Mote with 1Gb NAND flash:
  - 1Hz sampling of photo sensor – saved twice to flash
  - 20 byte average summaries computed every 12000 samplings
  - Transmitted using the CC1000 radio (BMAC and 1% duty cycling)



Capsule consumes ***only 14.5%*** of total system energy having ***written 48Kb*** of data and subsequently ***read 24Kb*** !

Comparison with Matchbox

Comparison done on Mica2 platform with Atmel NOR flash

	Capsule		Matchbox	
	Energy (uJ)	Latency (ms)	Energy (uJ)	Latency (ms)
Write (80bx10)	8.83	85.6	10.57	91.60
Read (80bx10)	1.20	18.44	1.12	16.52
Write b/w	18kbps		11.3kbps	
Read b/w	54.2kbps		60.4kbps	

Capsule provides rich additional features at an energy cost equivalent to that of Matchbox

Capsule Deployments

TurtleNet

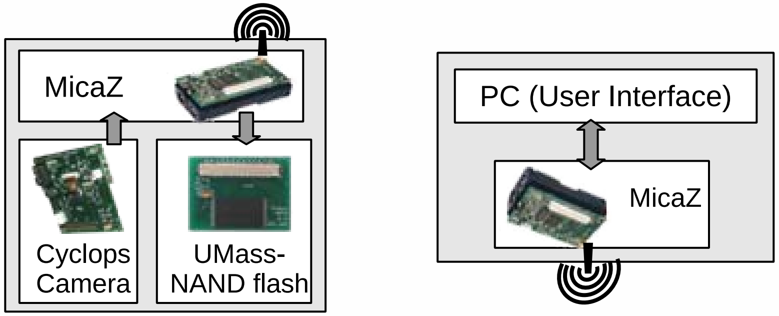
- Goal: Location monitoring of turtles in natural habitat
- Challenges
  - Frequent failures due to harvested energy fluctuations
  - Disconnected network operation

- Capsule used for storage
  - Stream + Index for storing data
  - Uses checkpointing/rollback

[Sorber, Corner, et al, UMass]

Storage-centric Camera Sensor Network (Demo)

- Goal: Use low-power cameras for monitoring
- Challenges
  - Images are large (>4K), typically more than available memory



- Capsule used for storage
  - Images stored and indexed
  - Using packet queue on flash

Future Work

- Supporting compaction for composite objects
- Port to TinyOS 2.0
- Port to use TinyOS Hardware Abstraction Layer (HAL)
- Build an energy-efficient in-network sensor database layer over Capsule

Conclusion

- Advocate rich object-based storage abstraction to support flexible use of storage
- Capsule
  - Optimized implementations of common objects
  - Supports checkpointing & rollback for fault tolerance
  - Portable to multiple platforms
- Experiments and deployments demonstrate energy-efficiency and flexibility of Capsule

Questions ?

Webpage: http://sensors.cs.umass.edu/projects/capsule

UMassAmherst

Backup Slides

Storage-Computation-Commn

Compare storage energy costs against state-of-art low-power radios and low-power microcontrollers.

Computation → TI MSP430; Storage → Toshiba NAND read, Toshiba NAND write; Communication → CC2420 Radio Tx, CC2420 Radio Rx

	TI MSP430	Toshiba NAND read	Toshiba NAND write	CC2420 Radio Tx	CC2420 Radio Rx
Energy/byte(uJ)	0.0008	0.004	0.009	1.8	2.1
Ratio	1	5	11	2250	2600

Computation is 10x cheaper than storage which in turn is 200x cheaper than communication

"Ultra-low power storage for sensor networks", Mathur, Desnoyers, Ganesan, Shenoy, IPSN SPOTS 2006.

Index Object

- Offers direct access to each element
  - Stored as a static array hierarchy
  - Optimize memory usage by sharing one buffer across all arrays at the same level of the index
  - File is a version of the Index object with buffering