A Forest-structured Bloom Filter with Flash Memory

Guanlin Lu, Biplob Debnath, David H.C. Du
Department of Computer Science and Engineering
University of Minnesota Twin Cities

This work was partially supported by NSF grants 0960833 and 0934396

Introduction to Bloom Filter

- What's it?
  - A bit vector that compactly represents a set of items (keys)
    - Support key query/insert operations
    - Tell definitely if a key is NOT present; couldn't tell with guarantee that a key is indeed present (a few false positives may exist)
- Where is Bloom Filter (BF) used for?
  - Database applications
  - Network applications
    - E.g., router
  - Backup applications
    - E.g., chunking based data dedupe (not found→new chunk!)

Extending BF to Secondary Storage Device

- Why?
  - In-RAM BF size is limited by the available RAM size on the machine. However, some Apps like dedupe needs BF size beyond RAM capacity.
- Main concept
  - Utilize a limited amount of RAM space combined with a much larger secondary storage space to form a BF
- Secondary storage device choices
  - flash memory vs. magnetic disk

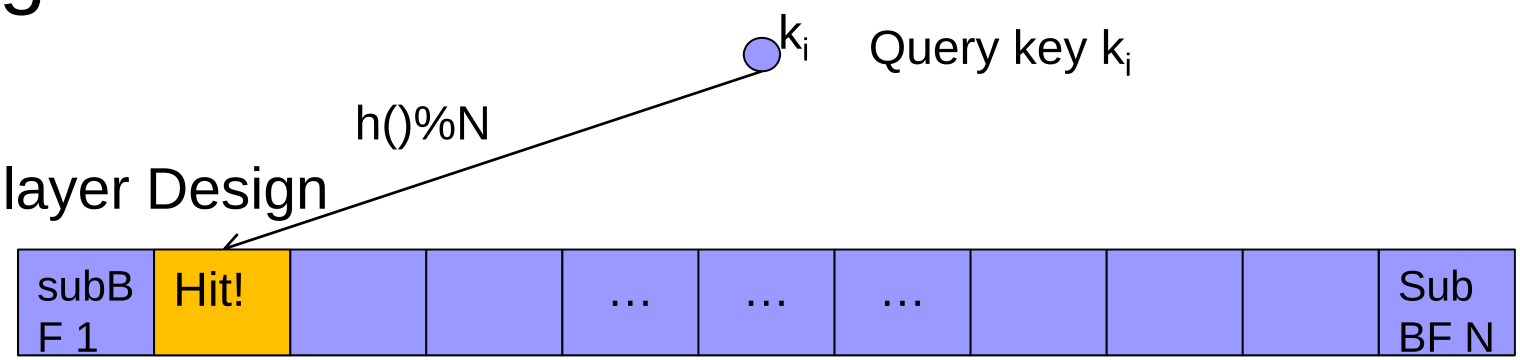
Building a BF with Flash Memory

- Special cha...
  - page-lev... se
  - random ... equential page read
  - page wri... ge update needs a flash erase firs...
  - each flas... unt during life-cycle

How is the BF design optimized for flash characteristics?

Existing Works

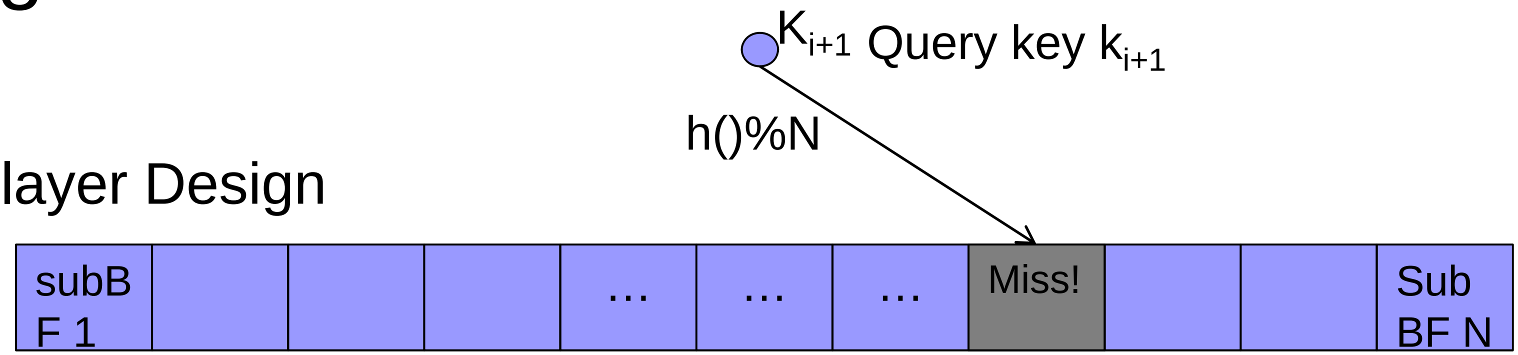
- Single-layer Design



- Pros
  - It requires only 1 flash page R /key query → best for key query
- Cons
  - Buffer space is very limited for each sub-BF→ many flash read-then-write ops are required for each sub-BF during the run.
  - Some sub-BFs tend to receive more keys than others (by single hash function), but buffer space is equally pre-partitioned
  - BF size has to be determined in advance and could not be changed during the run

Existing Works

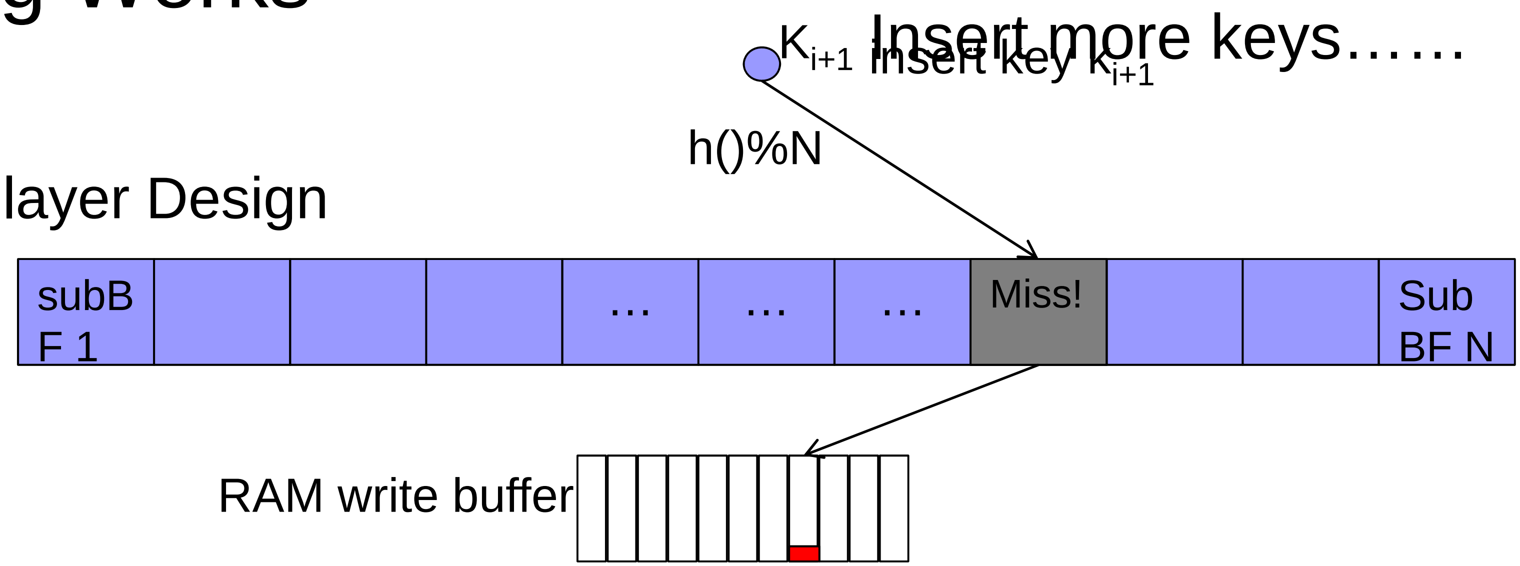
- Single-layer Design



- Pros
  - It requires only 1 flash page R /key query → best for key query
- Cons
  - Buffer space is very limited for each sub-BF→ many flash read-then-write ops are required for each sub-BF during the run.
  - Some sub-BFs tend to receive more keys than others (by single hash function), but buffer space is equally pre-partitioned
  - BF size has to be determined in advance and could not be changed during the run

Existing Works

- Single-layer Design

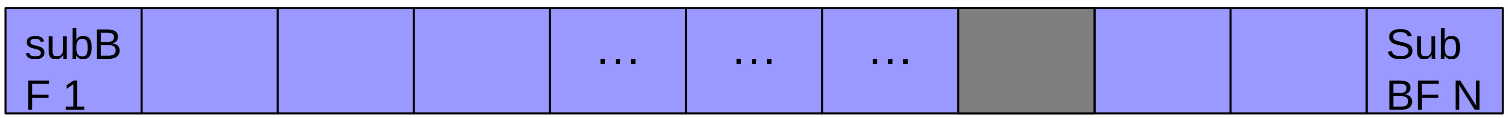


- Pros
  - It requires only 1 flash page R /key query → best for key query
- Cons
  - Buffer space is very limited for each sub-BF→ many flash read-then-write ops are required for each sub-BF during the run.
  - Some sub-BFs tend to receive more keys than others (by single hash function), but buffer space is equally pre-partitioned
  - BF size has to be determined in advance and could not be changed during the run

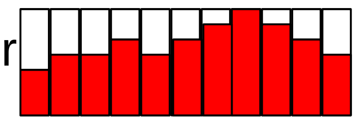
Existing Works

update sub-BF

- Single-layer Design



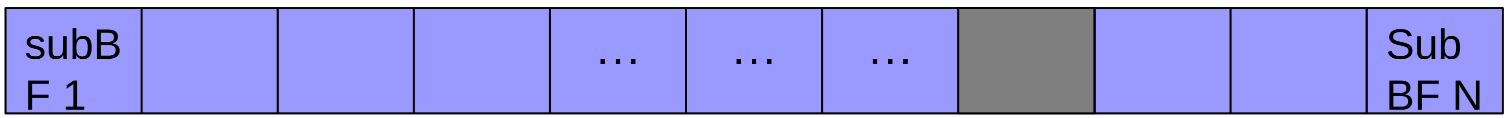
RAM write buffer

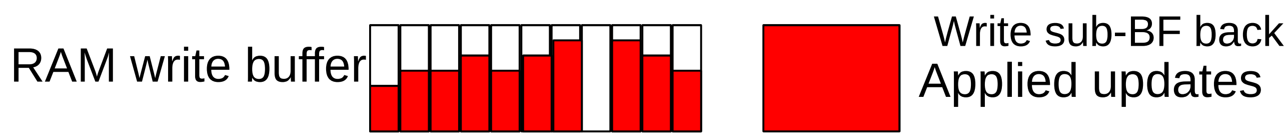


- Pros
  - It requires only 1 flash page R /key query → best for key query
- Cons
  - Buffer space is very limited for each sub-BF→ many flash read-then-write ops are required for each sub-BF during the run.
  - Some sub-BFs tend to receive more keys than others (by single hash function), but buffer space is equally pre-partitioned
  - BF size has to be determined in advance and could not be changed during the run

Existing Works

- Single-layer Design

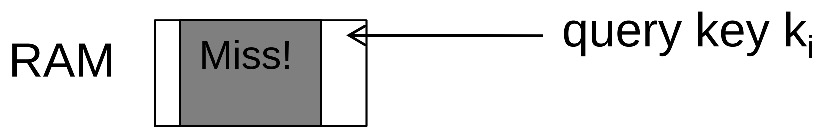




- Pros
  - 1 flash page R /key query → best for key query
- Cons
  - Buffer space is very limited for each sub-BF→ many flash read-then-write ops are required for each sub-BF during the run.
  - Some sub-BFs tend to receive more keys than others (by single hash function), but buffer space is equally pre-partitioned
  - BF size has to be determined in advance and could not be changed during the run

Existing Works

- Linear-chaining Design

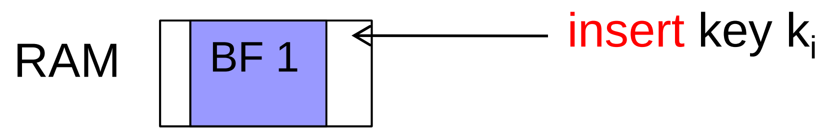


- Pros
  - best for key insertion: each chained BF will be only written once, hence the flash write # is minimized
  - BF size grows dynamically as the # of chained BFs increased
- Cons
  - Querying a key may require traverse all chained BFs
  - False positive errors tend to be much higher than single-layer design

Existing Works

K_i

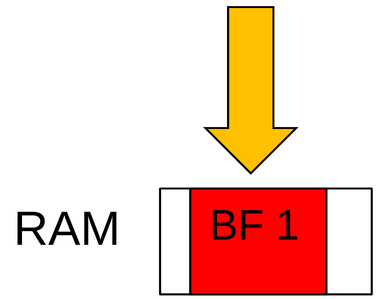
- Linear-chaining Design



- Pros
  - best for key insertion: each chained BF will be only written once, hence the flash write # is minimized
  - BF size grows dynamically as the # of chained BFs increased
- Cons
  - Querying a key may require traverse all chained BFs
  - False positive errors tend to be much higher than single-layer design

Existing Works

- Linear-chaining Design

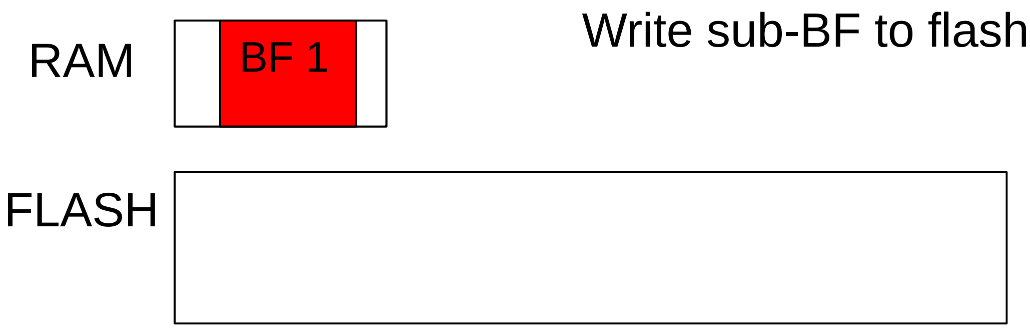


insert more keys …

- Pros
  - best for key insertion: each chained BF will be only written once, hence the flash write # is minimized
  - BF size grows dynamically as the # of chained BFs increased
- Cons
  - Querying a key may require traverse all chained BFs
  - False positive errors tend to be much higher than single-layer design

Existing Works

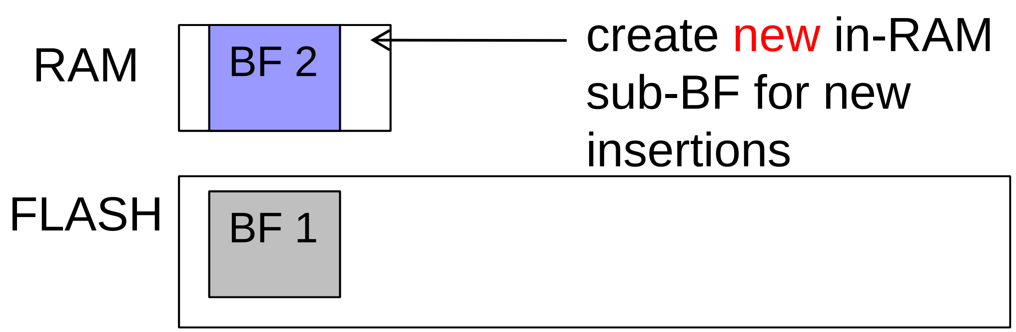
- Linear-chaining Design



- Pros
  - best for key insertion: each chained BF will be only written once, hence the flash write # is minimized
  - BF size grows dynamically as the # of chained BFs increased
- Cons
  - Querying a key may require traverse all chained BFs
  - False positive errors tend to be much higher than single-layer design

Existing Works

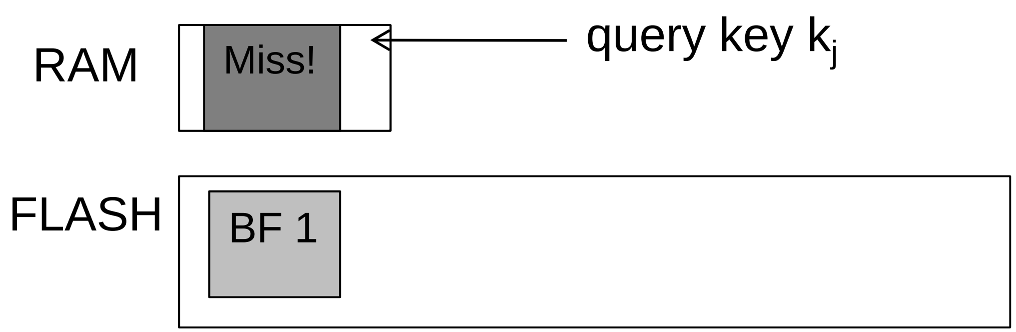
- Linear-chaining Design



- Pros
  - best for key insertion: each chained BF will be only written once, hence the flash write # is minimized
  - BF size grows dynamically as the # of chained BFs increased
- Cons
  - Querying a key may require traverse all chained BFs
  - False positive errors tend to be much higher than single-layer design

Existing Works

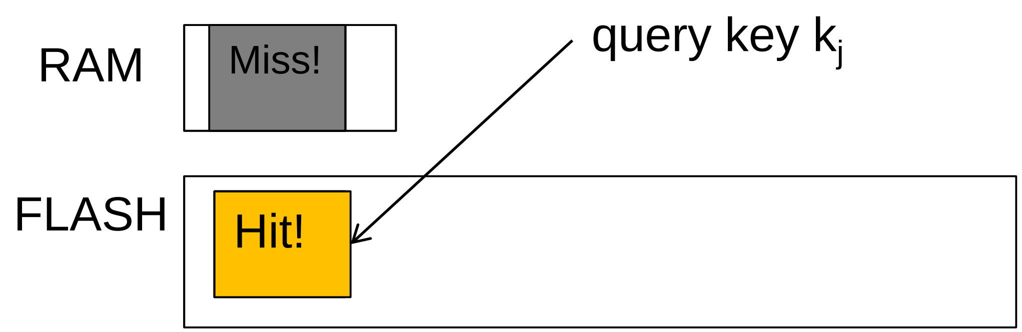
- Linear-chaining Design



- Pros
  - best for key insertion: each chained BF will be only written once, hence the flash write # is minimized
  - BF size grows dynamically as the # of chained BFs increased
- Cons
  - Querying a key may require traverse all chained BFs
  - False positive errors tend to be much higher than single-layer design

Existing Works

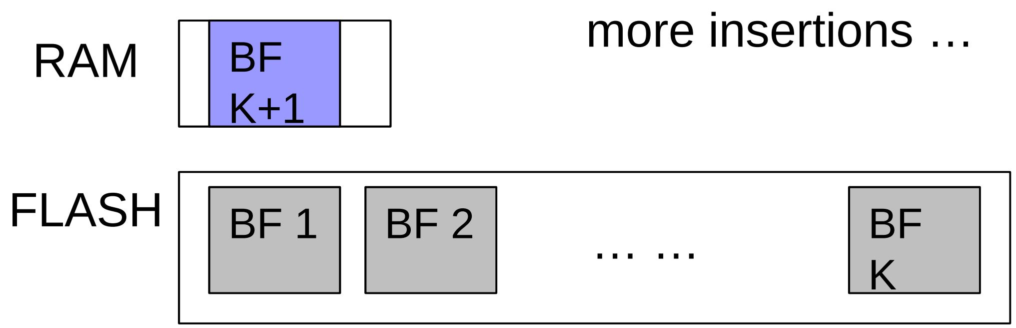
- Linear-chaining Design



- Pros
  - best for key insertion: each chained BF will be only written once, hence the flash write # is minimized
  - BF size grows dynamically as the # of chained BFs increased
- Cons
  - Querying a key may require traverse all chained BFs
  - False positive errors tend to be much higher than single-layer design

Existing Works

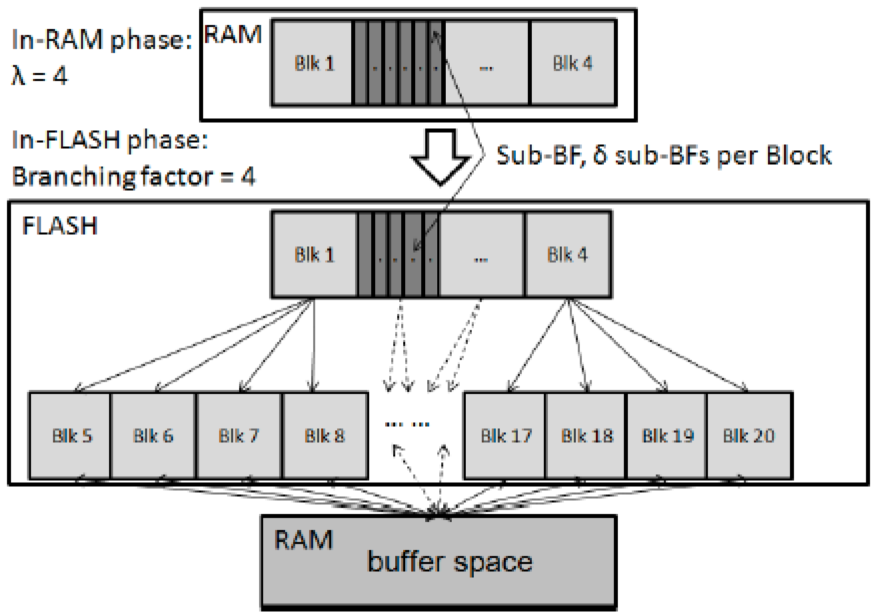
- Linear-chaining Design



- Pros
  - best for key insertion: each chained BF will be only written once, hence the flash write # is minimized
  - BF size grows dynamically as the # of chained BFs increased
- Cons
  - Querying a key may require traverse all chained BFs
  - False positive errors tend to be much higher than single-layer design

Proposed Forest-structured BF(FBF) Design

- Goal: To strike a balance between key query and insert performance
- Partition flash … of flash-page sized and org… .
- Key features
  - Overall BF … ayer of forest
  - Each key c… ls equal to forest heig…
  - Key inserti… uffer, which is designed t… d in next page).

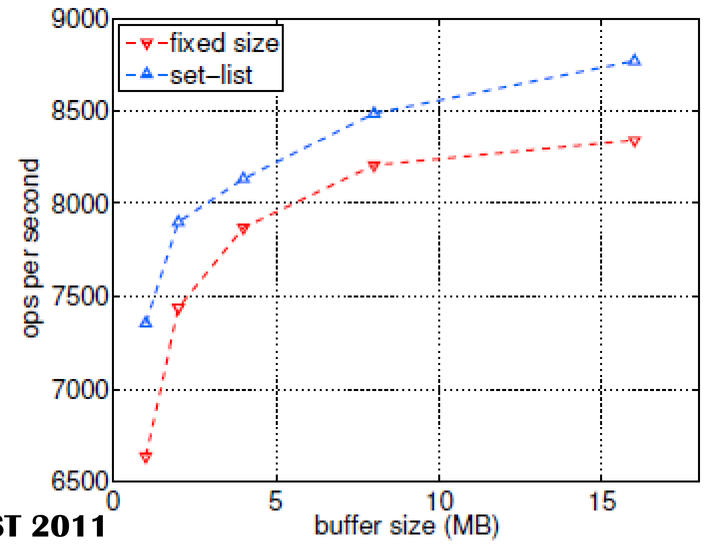


Proposed Buffer Space Management Scheme for FBF Design

- FBF inserts new keys into the lowest-layer of the forest only, which optimizes for
  - allowing larger buffer space per sub-BF
  - Minimize the target address range for flash writes
- FBF manages buffer space by
  - grouping consecutive sub-BFs into blocks
  - buffering key insertions per block in a in-RAM set data structure
  - keeping all sets into a linked-list
  - selecting the block corresponding to the set containing most insertions to update when the entire buffer space is used up.

Experimental Evaluation Results

- Workload description:
  - A sequence (20 millions) of SHA1 hash value of 160-bit length. Each of which represents a chunk-id produced by standard content-defined chunking algorithm; 57% are unique chunk-ids
  - BF access pattern: Key query & insert are interleaved
- TR vs. buffer size for both cache managing schemes:

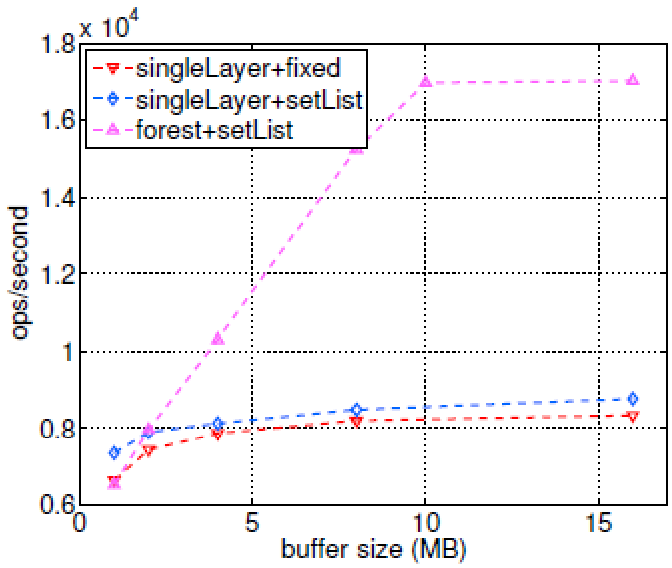


COMPARTMENT VS. SET-LIST SCHEMES ON *vx-20m*

buffer schemes	fixed-size compartment	set-list
number of flash writes	2,024	1,053
ops/sec	8,405	8,657

Experimental Evaluation Results

- Throughput Rate (TR) vs. buffer sizes for forest-structure BF and single-layer BF



UNIVERSITY OF MINNESOTA
Driven to Discover℠

Summary of Contributions

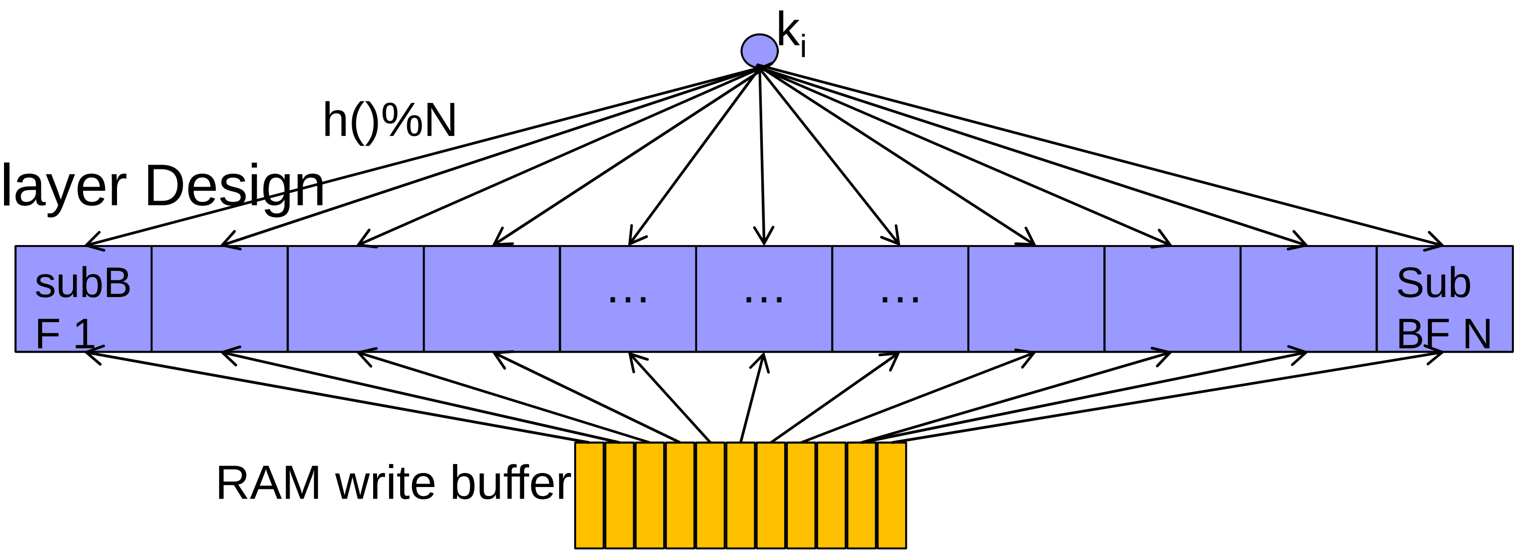
- We present a novel BF design (FBF) with flash memory that
  - strikes a balance between key query and key insert performance
  - achieves a significantly higher TR with the same buffer size compared with existing designs.
- Furthermore, our proposed buffer space managing scheme reduces the number of flash writes remarkably (e.g., 50% less), even with the same existing BF design.

Thank you!

Thank you!

Background Works

- Single-layer Design



- Pros
  - requires only 1 flash page R /key query → best for key query
- Cons
  - Buffer space is very limited for each sub-BF→ many flash read-then-write ops are required for each sub-BF during the run.
  - Some sub-BFs tend to receive more keys than others (by single hash function), but buffer space is equally partitioned ahead
  - BF size should be determined in advance and could not be changed during the run