Design and Implementation of Signatures in Transactional Memory Systems

Daniel Sanchez

August 2007

University of Wisconsin-Madison

Outline

- Introduction and motivation
- Bloom filters
- Bloom signatures
- Area & performance evaluation
- Influence of system parameters
- Novel signature schemes (brief overview)
- Conclusions

Signature-based conflict detection

- Signatures:
  - Represent an *arbitrarily large set* of elements in *bounded* amount of state (bits)
  - Approximate representation, with *false positives* but *no false negatives*
- Signature-based CD: Use signatures to track read/write sets of a transaction
  - Pros:
    - Transactions can be *unbounded* in size
    - Independence from caches, eases virtualization
  - Cons:
    - False conflicts -> Performance degradation

Motivation of this study

- Signatures play an important role in TM performance. Poor signatures cause lots of unnecessary stalls and aborts.
- Signatures can take significant amount of area
  - Can we find area-efficient implementations?
  - Adoption of TM much easier if the area requirements are small!
- Signature design space exploration incomplete in other TM proposals

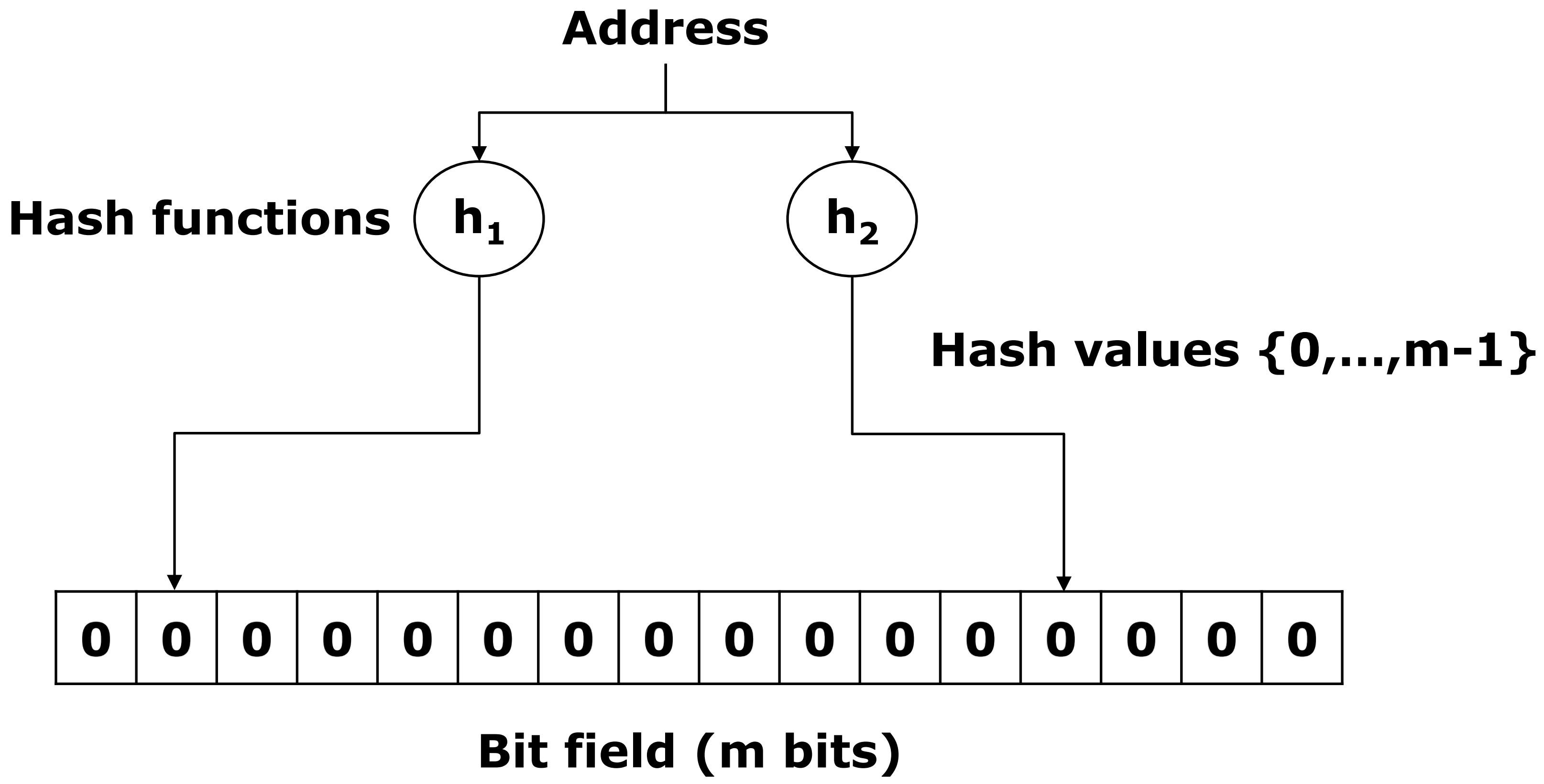
Summary of results

- Previously proposed TM signatures are either true Bloom (1 filter, k hash functions) or parallel Bloom (k filters, 1 hash function each).
  - Performance-wise, True Bloom = Parallel Bloom
  - Parallel Bloom about *8x more area-efficient*
- New Bloom signature designs that *double* the performance and are more robust
- Pressure on signatures greatly increases with the number of cores; directory can help
- Three novel signature designs

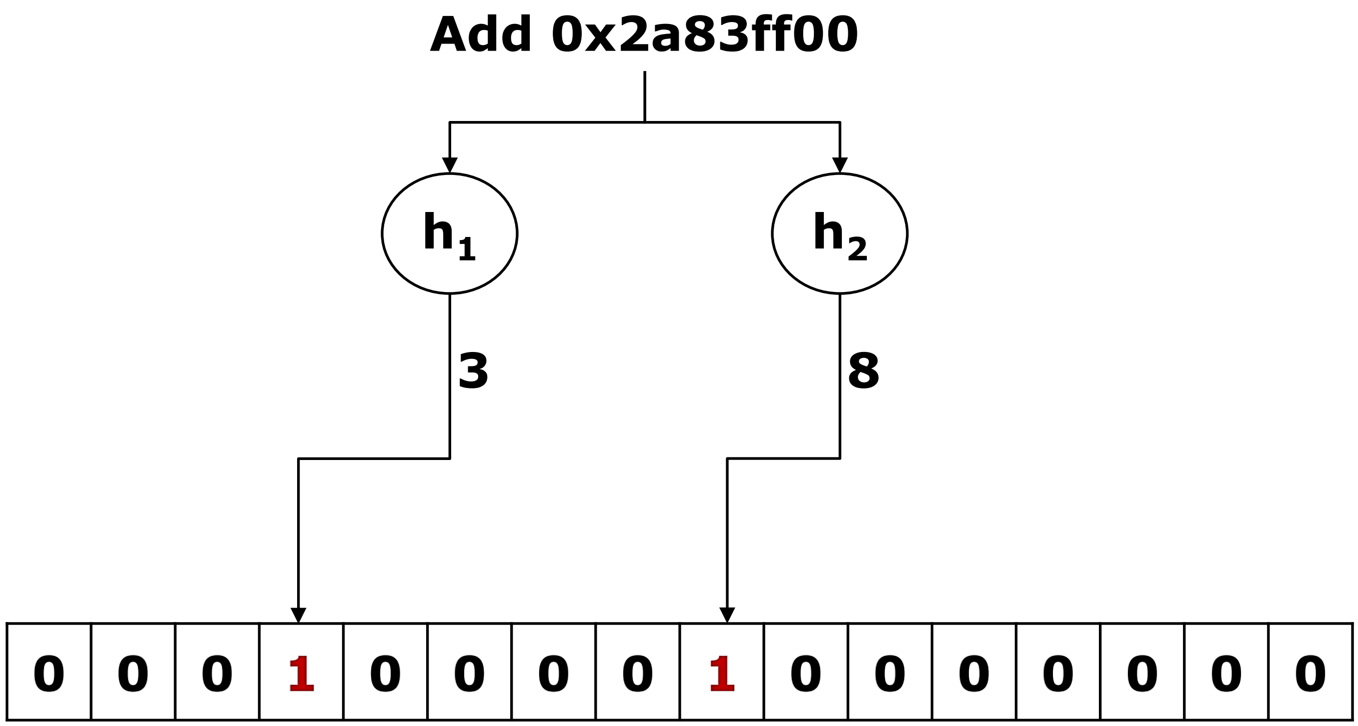
Outline

- Introduction and motivation
- Bloom filters
- Bloom signatures
- Area & performance evaluation
- Influence of system parameters
- Novel signature schemes (brief overview)
- Conclusions

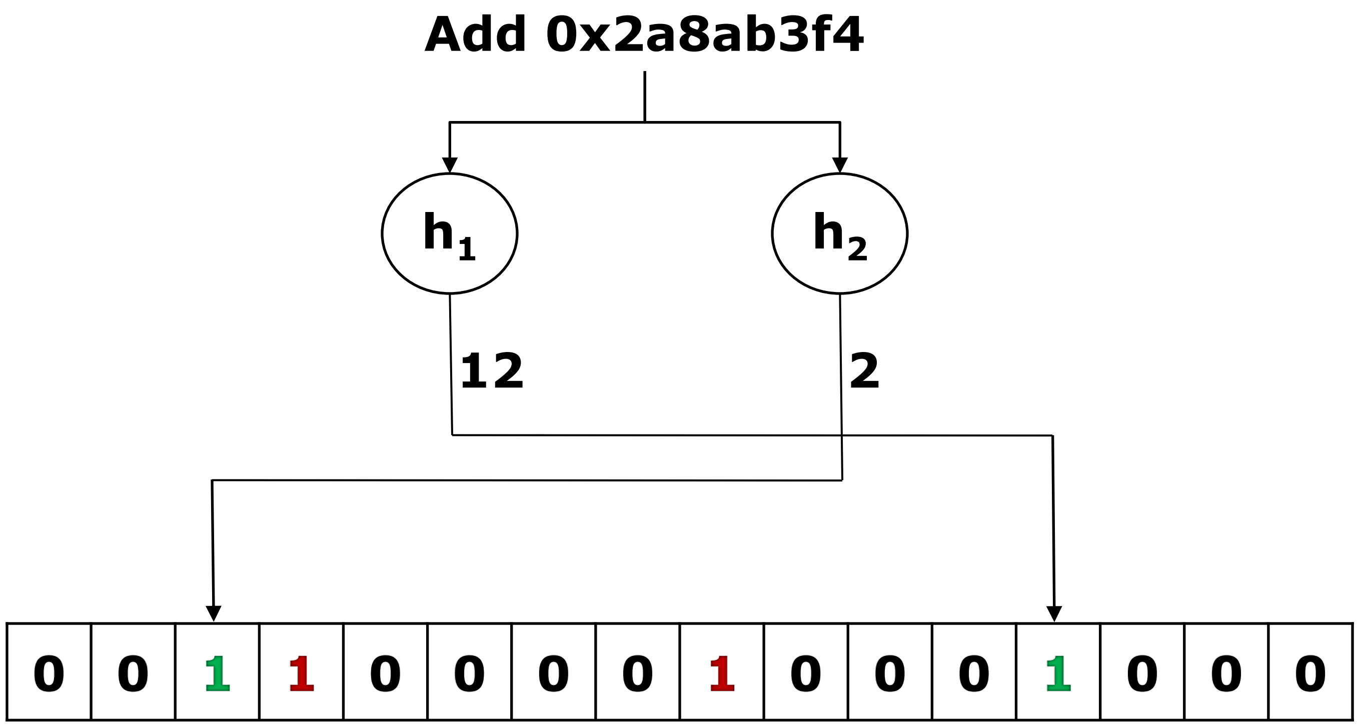
Bloom filters



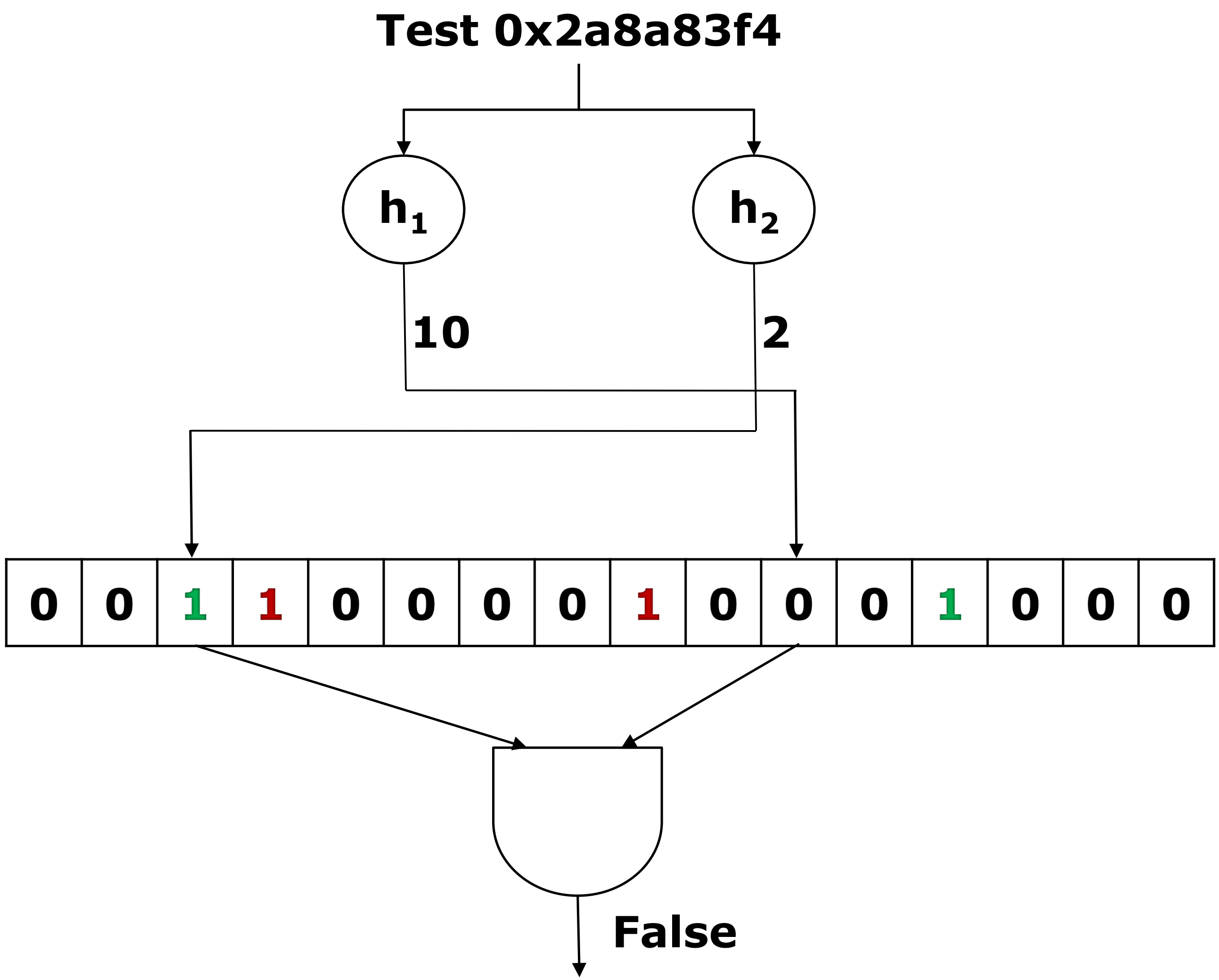
Bloom filters



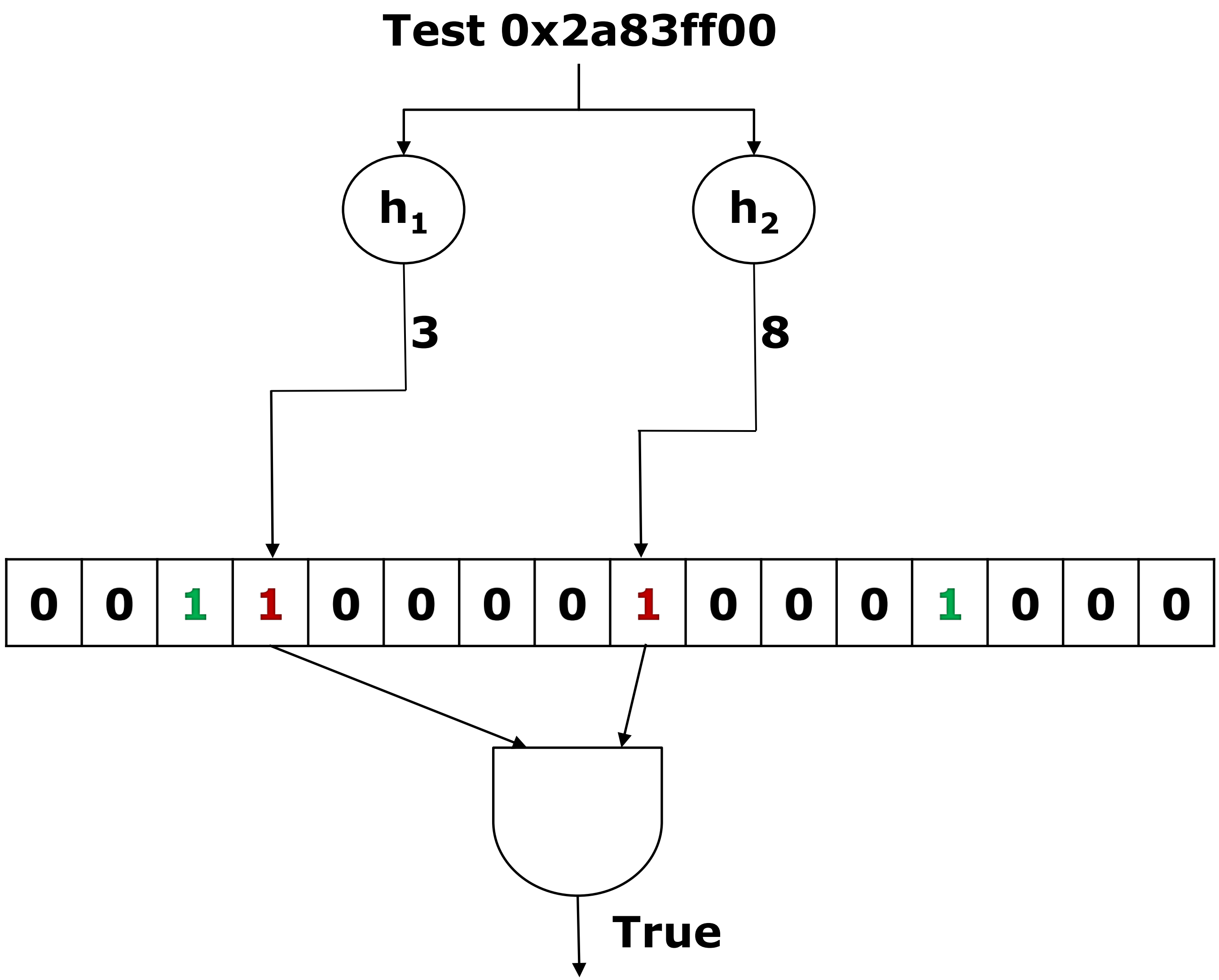
Bloom filters



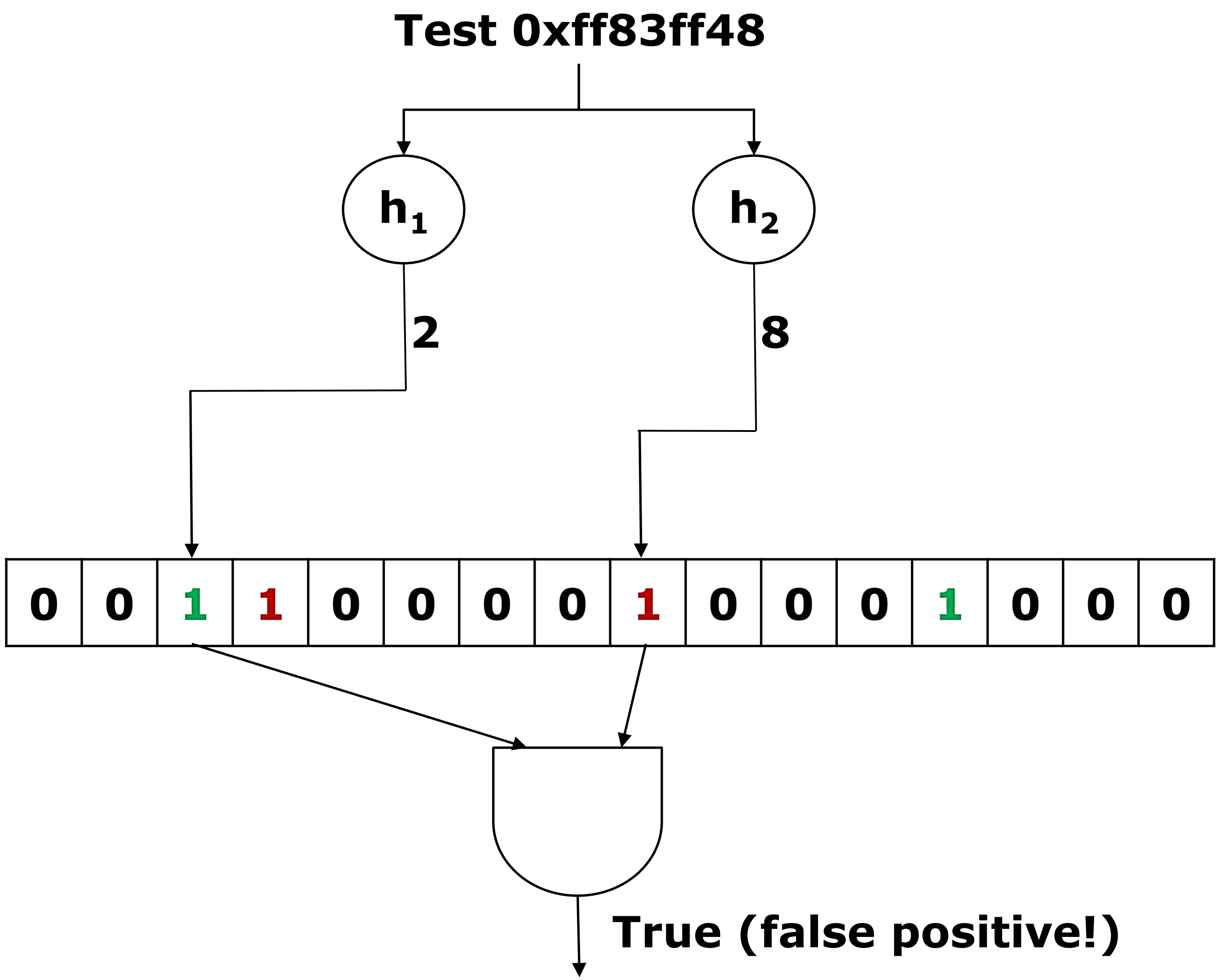
Bloom filters



Bloom filters



Bloom filters



Outline

- Introduction and motivation
- Bloom filters
- Bloom signatures
  - True Bloom signatures
  - Parallel Bloom signatures

  } Design
  Implementation
- Area & performance evaluation
- Influence of system parameters
- Novel signature schemes (brief overview)
- Conclusions

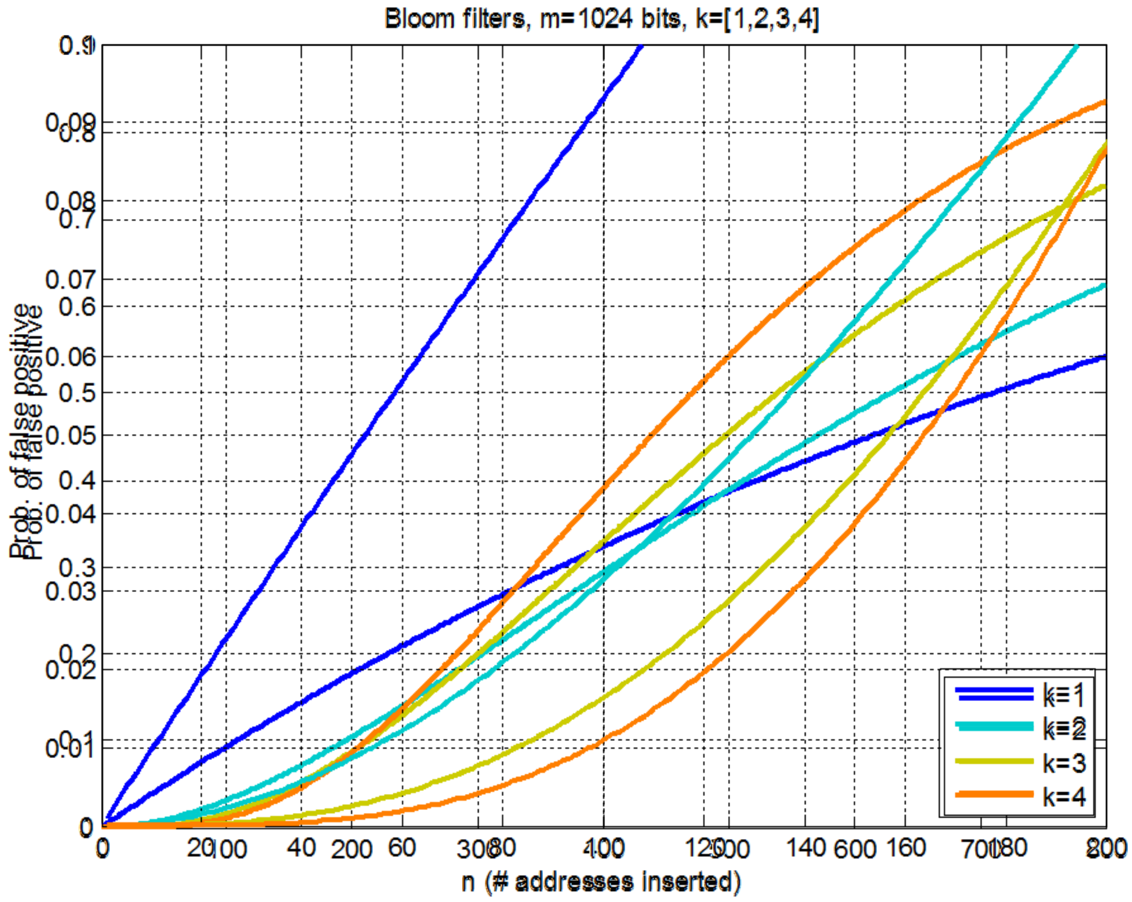
True Bloom signature - Design

- True Bloom signature = Signature implemented with a *single* Bloom filter
- Easy insertions and tests for membership
- Probability of false positives:

$$P_{FP}(n) = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k} \cong \left(1 - e^{\frac{-nk}{m}}\right)^{k} \quad (\text{if } \frac{k}{m} << 1)$$

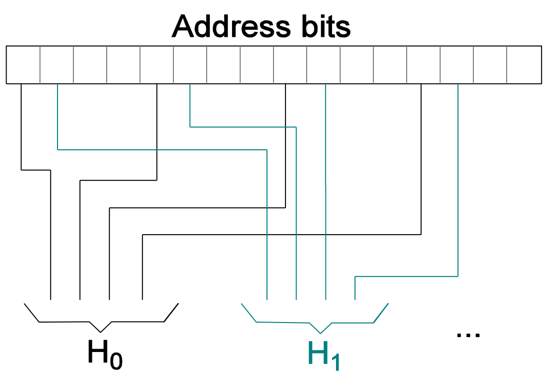
- Design dimensions
  - Size of the bit field (m)
  - Number of hash functions (k)
  - Type of hash functions

Number of hash functions



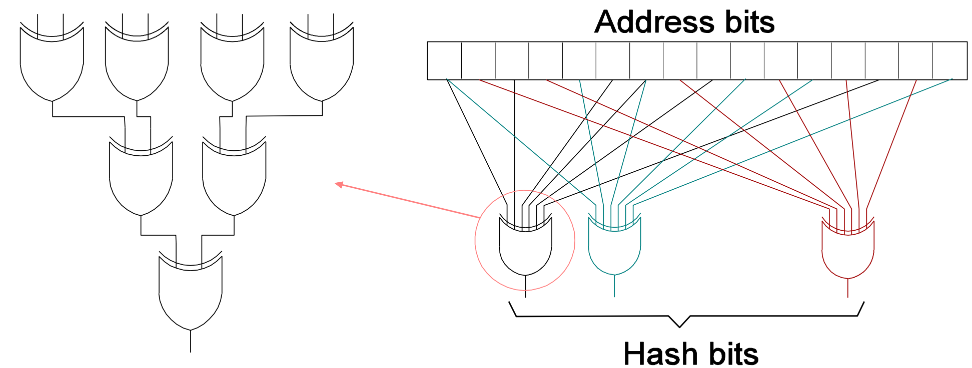
Types of hash functions

- Addresses neither independent nor uniformly distributed (key assumptions to derive $P_{FP}(n)$)
- But can generate hash values that are *almost* uniformly distributed and uncorrelated with good (universal/almost universal) hash functions
- Hash functions considered:



Bit-selection

(inexpensive, low quality)

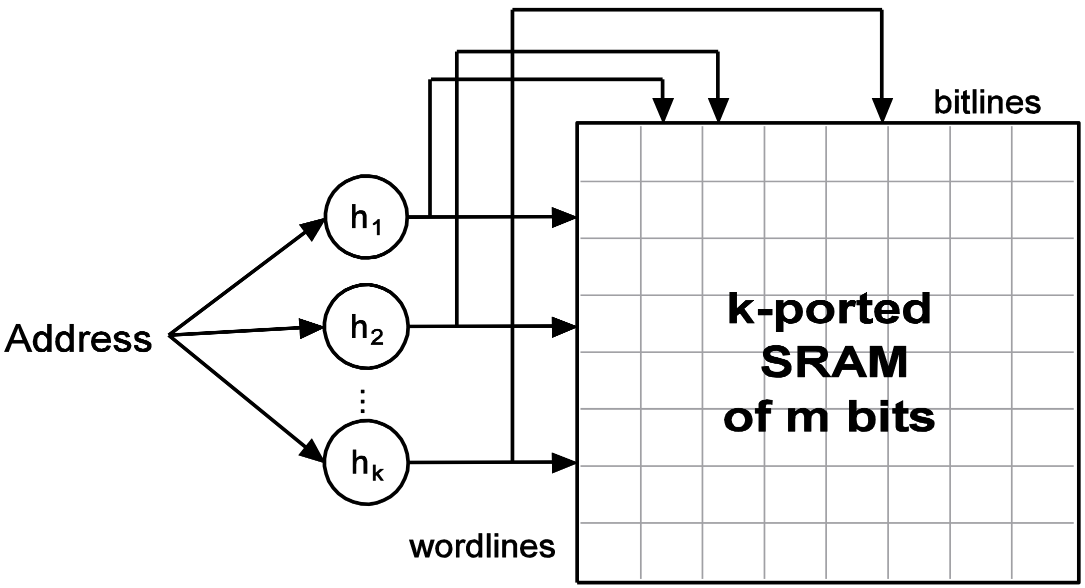


H_3

(moderate, higher quality)

True Bloom signature – Implementation

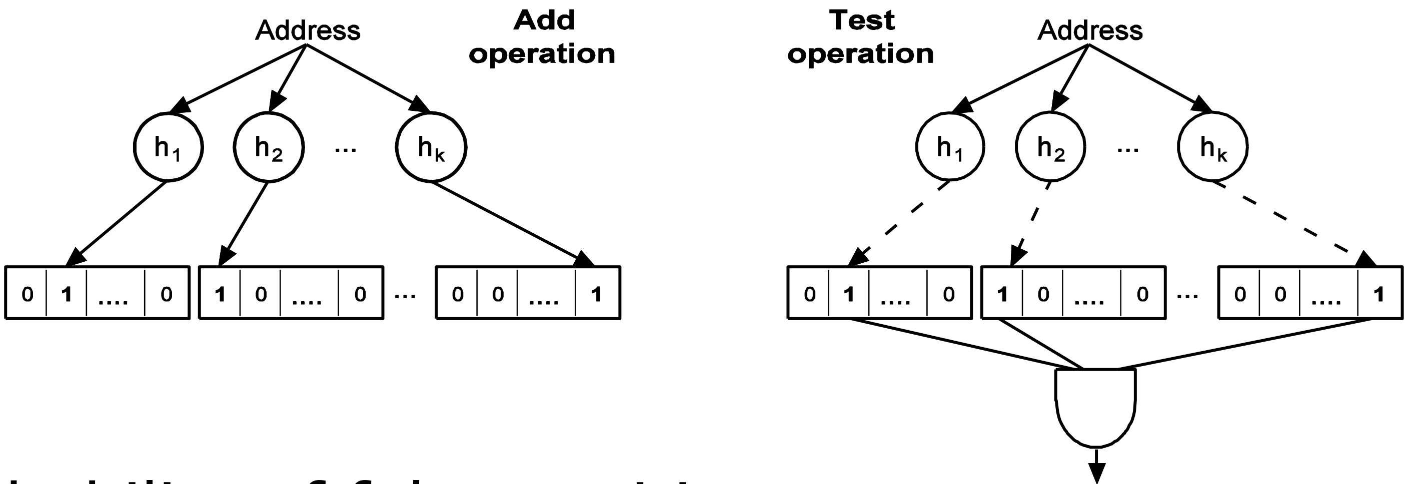
- Divide bit field in words, store in small SRAM
  - Insert: Raise wordline, drive appropriate bitline to 1, leave the rest floating
  - Test: Raise wordline, check the value at bitline
- k hash functions => k read, k write ports



Problem
Size of SRAM cell increases quadratically with # ports!

Parallel Bloom signatures - Design

- Use k Bloom filters of size m/k, with independent hash functions

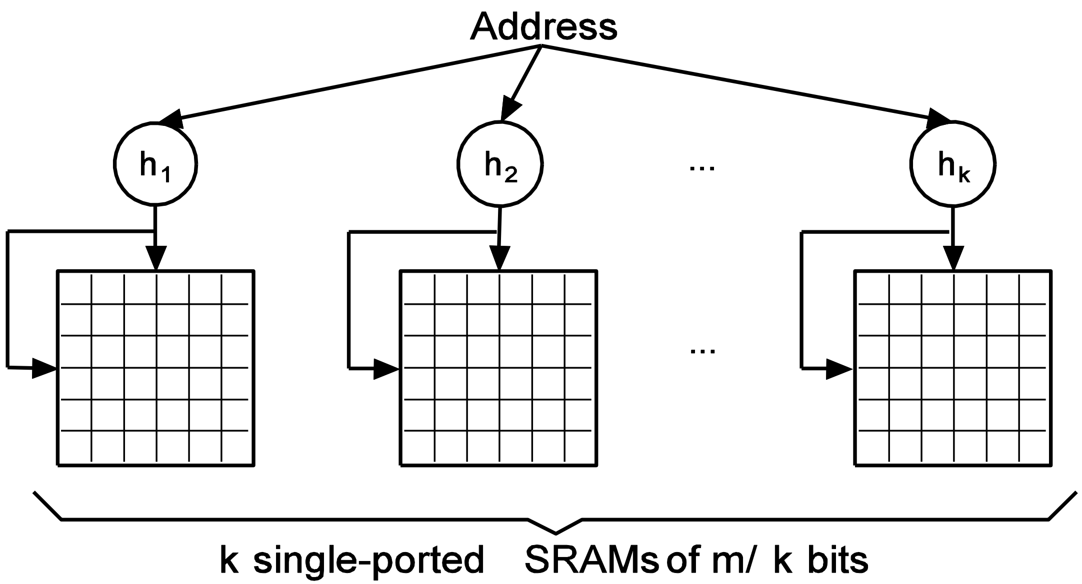


- Probability of false positives:

$$P_{FP}(n) = \left(1 - \left(1 - \frac{1}{m/k}\right)^{n}\right)^{k} \cong \left(1 - e^{\frac{-nk}{m}}\right)^{k}$$

Same as true Bloom!

Parallel Bloom signature - Implementation



- Highly area-efficient SRAMs
- Same performance as true Bloom! (in theory)

Outline

- Introduction and motivation
- Bloom filters
- Bloom signatures
- Area & performance evaluation
  - Area evaluation
  - True vs. Parallel Bloom in practice
  - Type of hash functions
  - Variability in hash functions
- Influence of system parameters
- Novel signature schemes (brief overview)
- Conclusions

Area evaluation

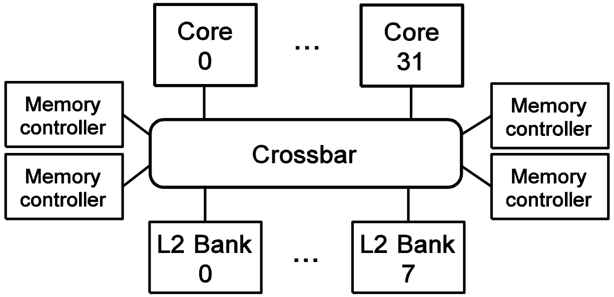
- SRAM: Area estimations using CACTI
  - 4Kbit signature, 65nm

	k=1	k=2	k=4
True Bloom	0.031 mm^2	0.113 mm^2	0.279 mm^2
Parallel Bloom	0.031 mm^2	0.032 mm^2	0.035 mm^2

- 8x area savings for four hash functions!
- Hash functions:
  - Bit selection has no extra cost
  - Four hardwired H_3 require ≈25% of SRAM area

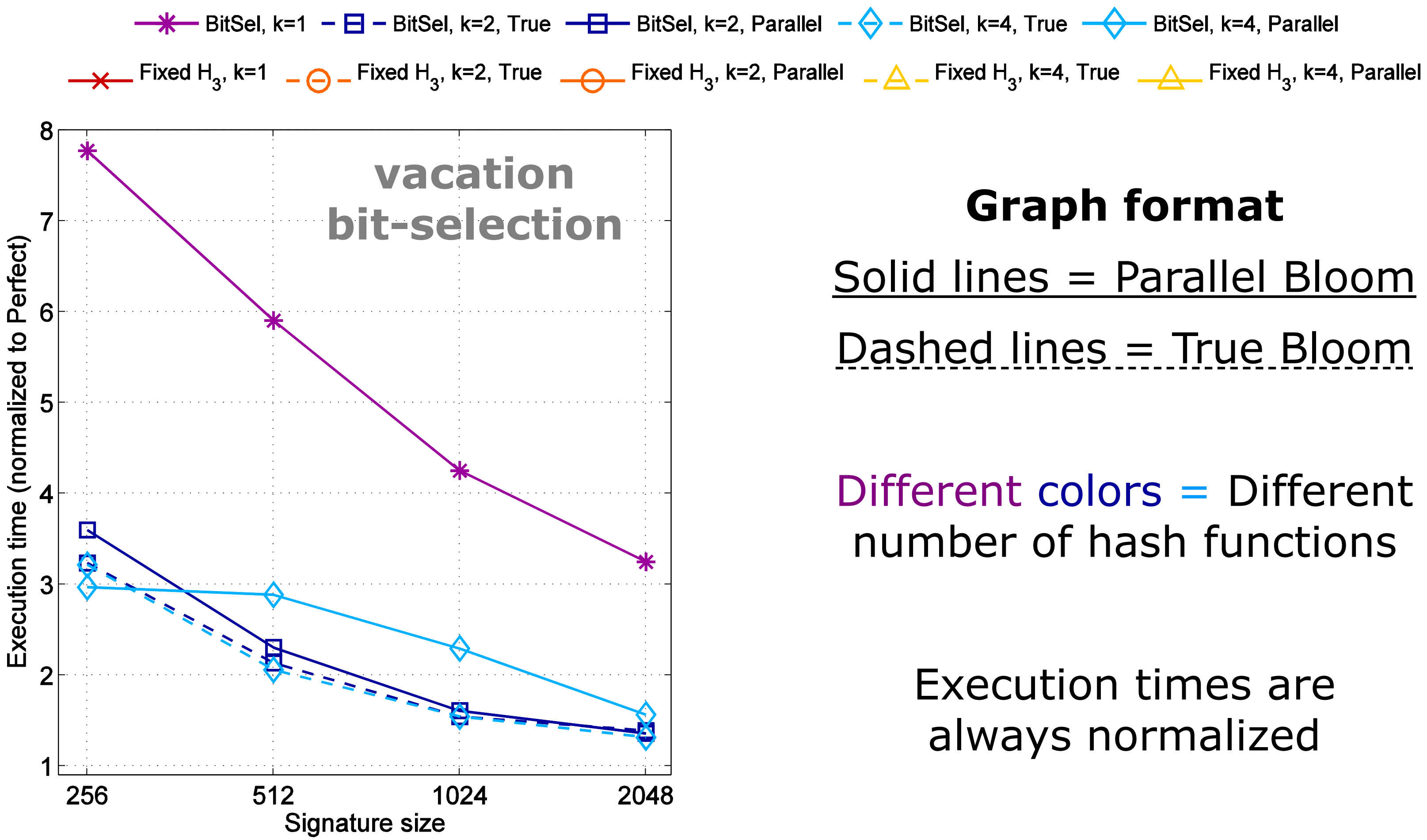
Performance evaluation

- System organization:
  - 32 in-order single-issue cores
  - Private split 32KB, 4-way L1 caches
  - Shared unified 8MB, 8-way L2 cache
  - High-bandwidth crossbar
  - Signature checks are *broadcast* (no directory)
  - Base conflict resolution protocol with *write-set prediction*



- Benchmarks: btree, raytrace, vacation
  - barnes, delaunay, and full set of results in report

True vs. Parallel Bloom signatures



Graph format

Solid lines = Parallel Bloom

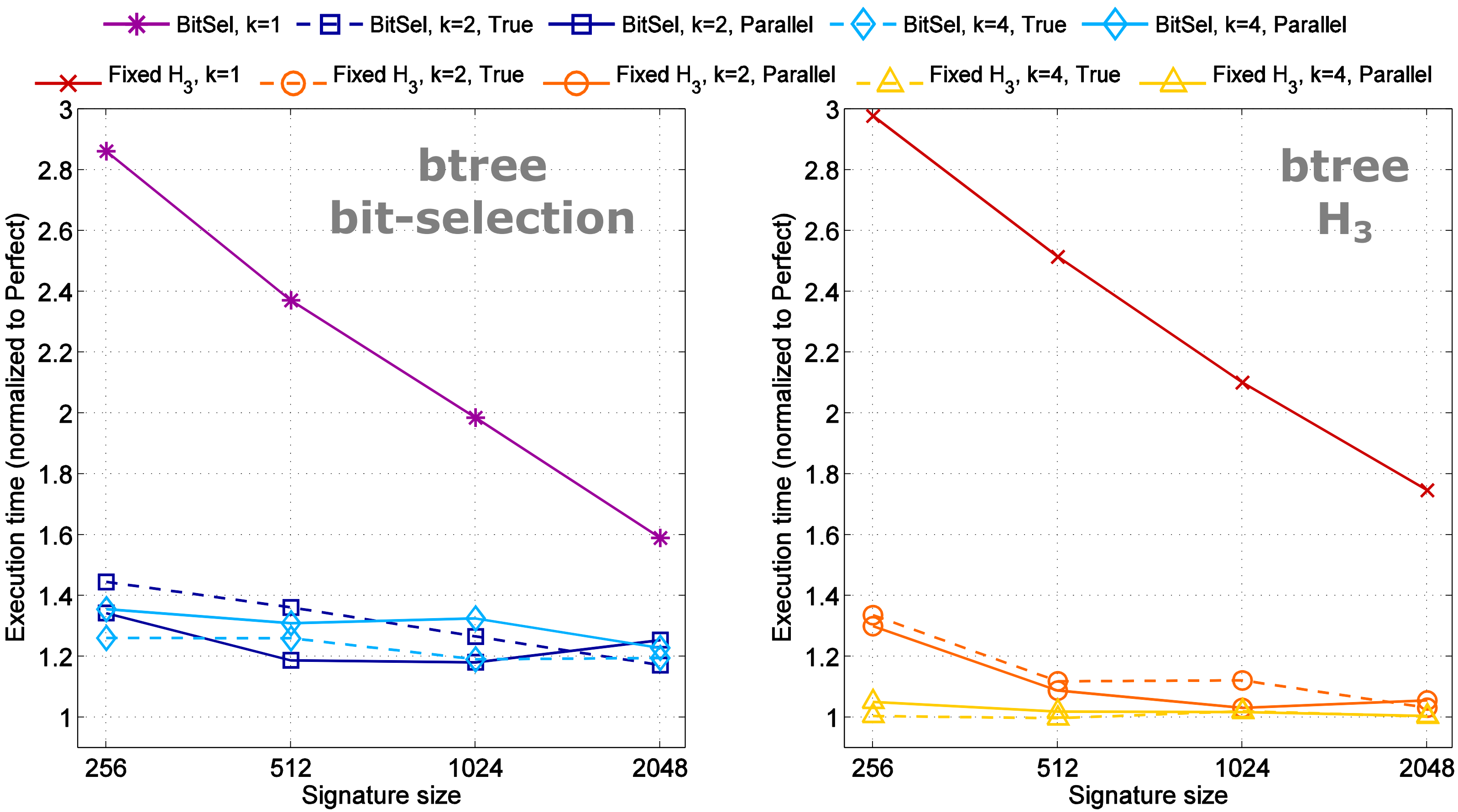
Dashed lines = True Bloom

Different colors = Different number of hash functions

Execution times are always normalized

- Bottom line: True ≈ parallel if we use good enough hash functions

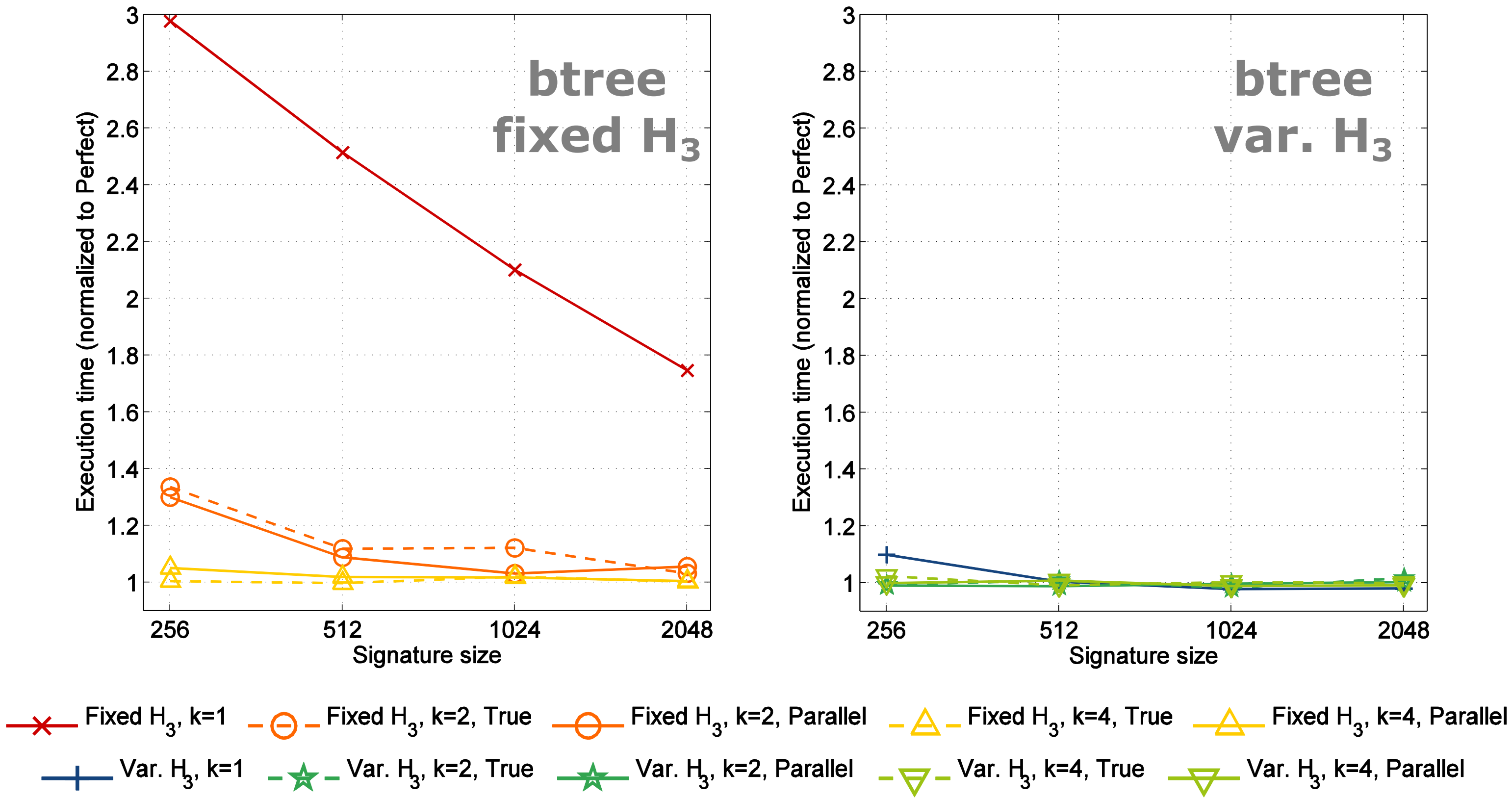
Bit-selection vs. fixed H_3



- H_3 clearly outperforms bit-selection for k≥2
- Only 2Kbit signatures with 4+ H_3 functions cause no degradation over all the benchmarks

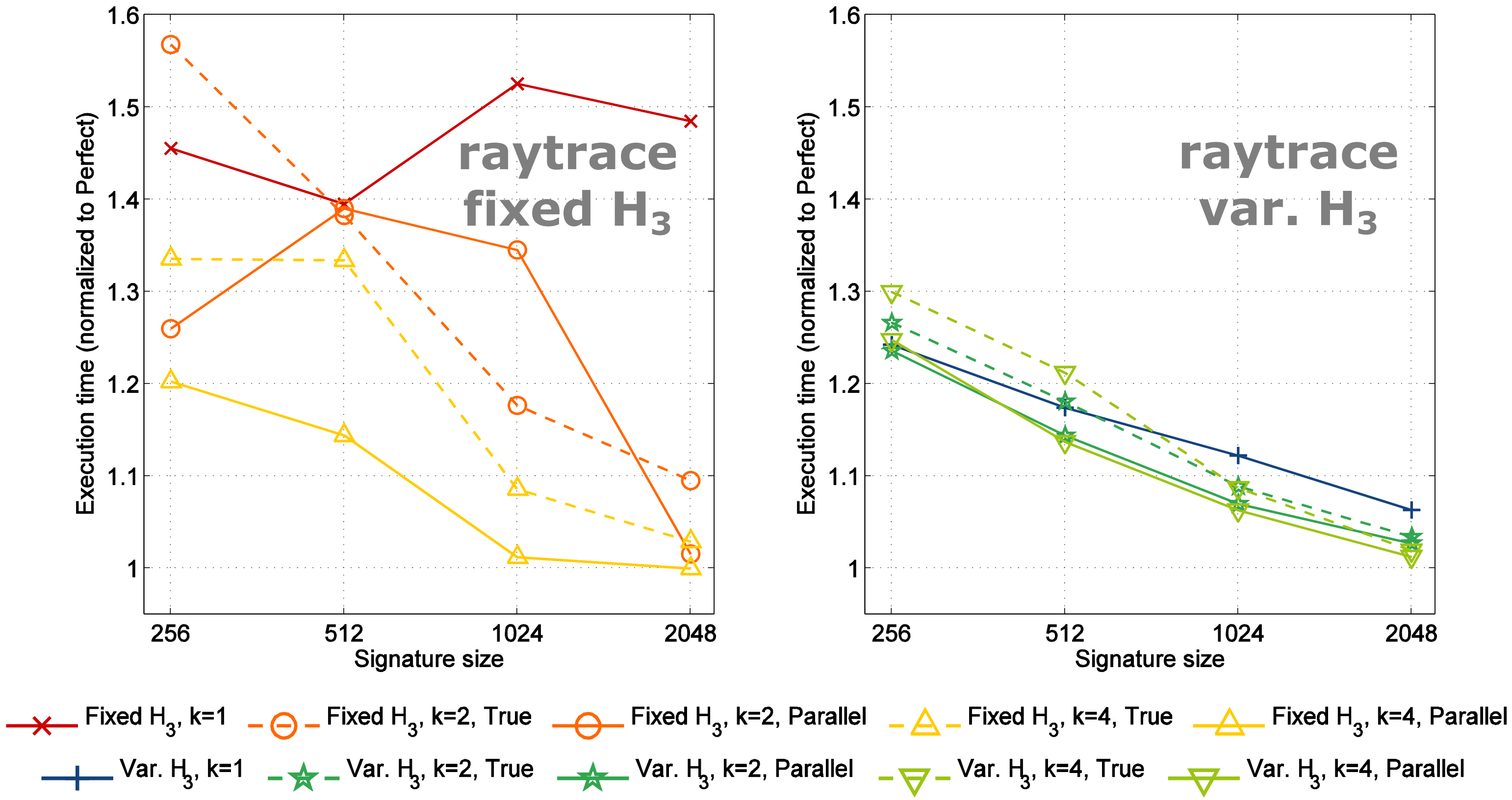
The benefits of variability

- Variable H_3: Reconfigure hash functions after each commit/abort
  - Constant aliases -> Transient aliases
  - Adds robustness



The benefits of variability

- Variable H_3: Reconfigure hash functions after each commit/abort
  - Constant aliases -> Transient aliases
  - Adds robustness



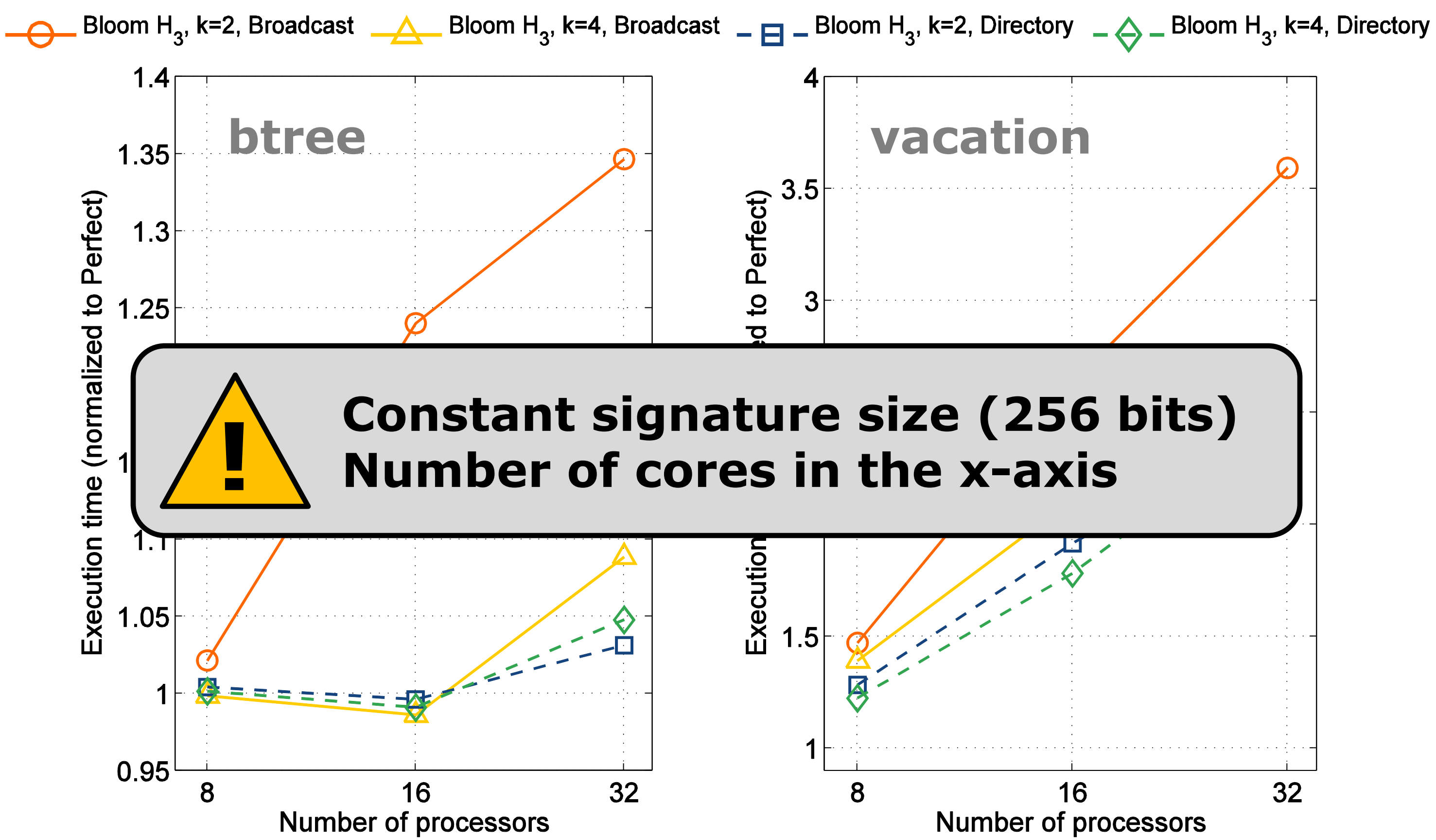
Conclusions on Bloom signature evaluation

- Parallel Bloom enables high number of hash functions "for free"
- Type of hash functions used matters a lot (but was neglected in previous analysis)
- Variability adds robustness
- Should use:
  - About four H_3 or other high quality hash functions
  - Variability if the TM system allows it
  - Size… depends on system configuration

Outline

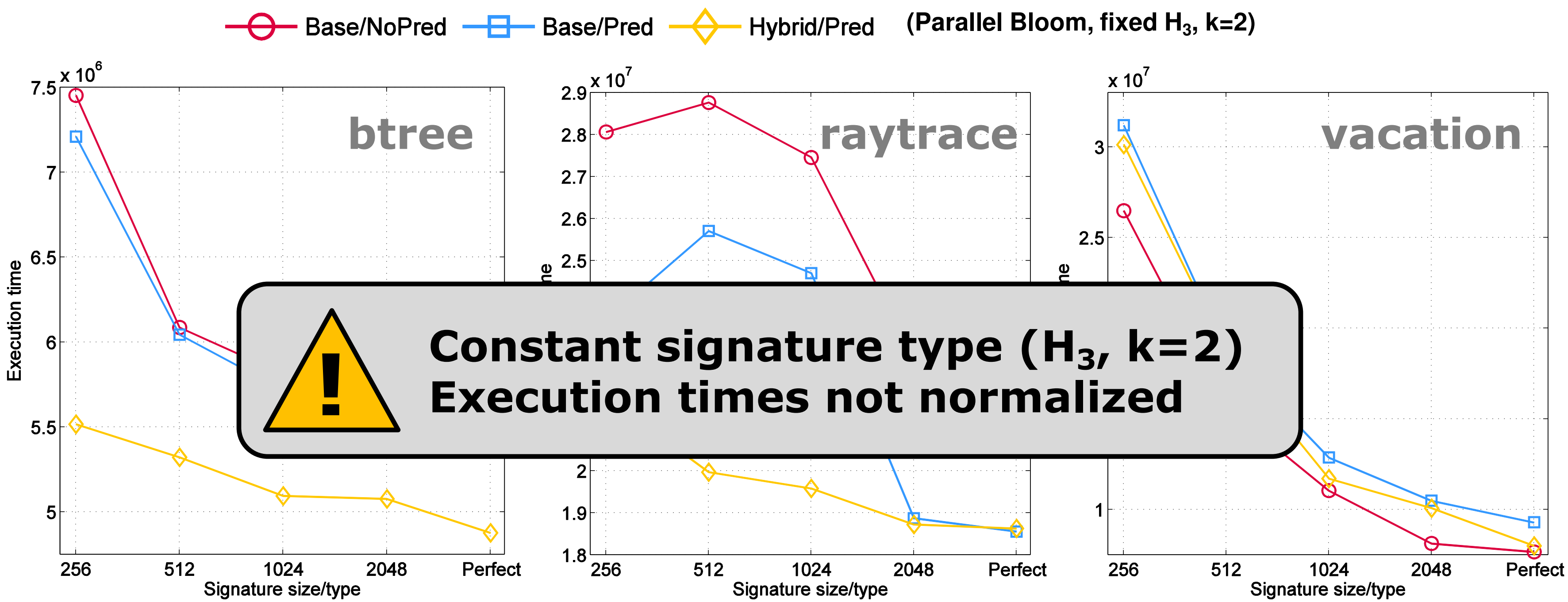
- Introduction and motivation
- Bloom filters
- Bloom signatures
- Area & performance evaluation
- Influence of system parameters
  - Number of cores
  - Conflict resolution protocol
- Novel signature schemes (brief overview)
- Conclusions

Number of cores & using a directory



- Pressure increases with #cores
- Directory helps, but still requires to scale the signatures with the number of cores

Effect of conflict resolution protocol



Constant signature type (H_3, k=2)
Execution times not normalized

- Protocol choice fairly orthogonal to signatures
- False conflicts *boost* existing pathologies in btree/raytrace -> Hybrid policy helps even more than with perfect signatures

Overview of novel signature schemes

- Cuckoo-Bloom signatures
  - Adapts cuckoo hashing for HW implementation
  - Keeps a hash table for small sets, morphs into a Bloom filter dynamically as the size grows
  - Significant complexity, performance advantage not clear
- Hash-Bloom signatures
  - Simpler hash-table based approach
  - Morphs to a Bloom filter more gradually than Cuckoo-Bloom
  - Outperforms Bloom signatures for both small and write sets, in theory and practice
- Adaptive Bloom signatures
  - Bloom signatures + set size predictors + scheme to select the best number of hash functions

Conclusions

- Bloom signatures should always be implemented as parallel Bloom
  - with ≈4 good hash functions, some variability if allowed
  - Overall good performance, simple/inexpensive HW
- Increasing #cores makes signatures more critical
  - Hinders scalability!
  - Using directory helps, but doesn't solve
- Hybrid conflict resolution helps with signatures
- There are alternative schemes that outperform Bloom signatures

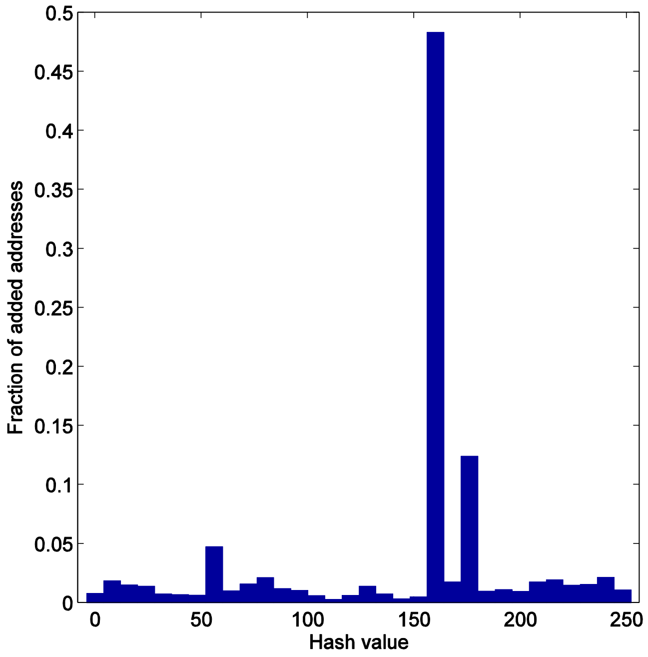
Thanks

for your attention

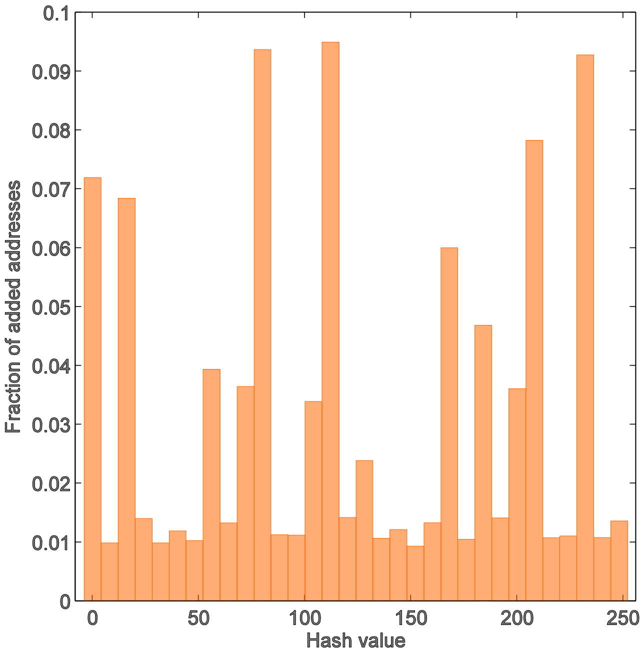
Any questions?

Backup – Hash function analysis

- Hash value distributions for btree, 512-bit parallel Bloom with 2 hash functions



bit-selection

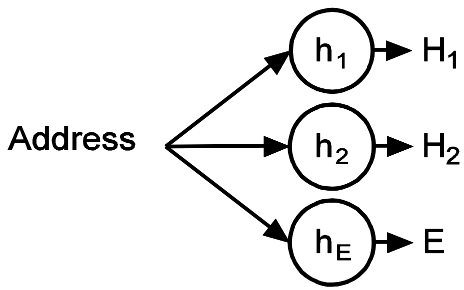


fixed H_3

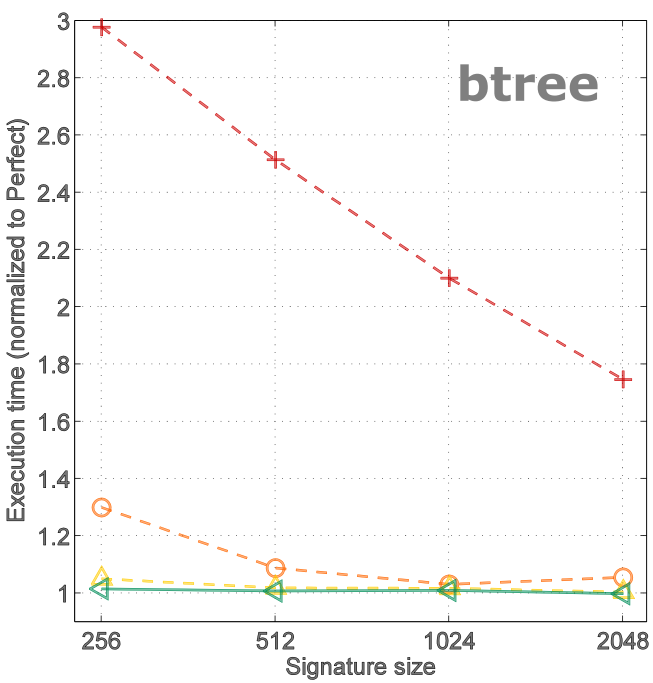
Backup - Conflict resolution in LogTM-SE

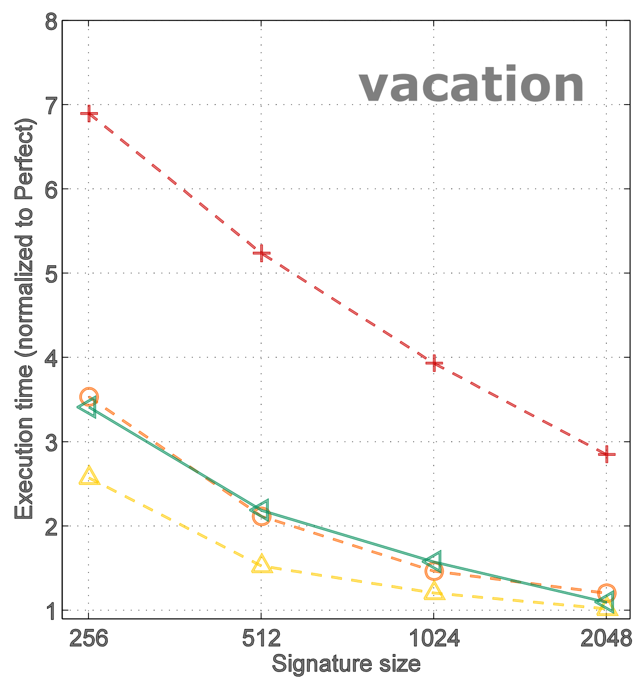
- Base: Stall requester by default, abort if it is stalling an older Tx and stalled bt an older Tx
- Pathologies:
  - DuelingUpgrades: Two Txs try to read-modify-update same block concurrently -> younger aborts
  - StarvingWriter: Difficult for a Tx to write to a widely shared block
  - FutileStall: Tx stalls waiting for other that later aborts
- Solutions:
  - Write-set prediction: Predict read-modify-updates, get exclusive access directly (targets DuelingUpgrades)
  - Hybrid conflict resolution: Older writer aborts younger readers (targets StarvingWriter, FutileStall)

Backup – Cuckoo-Bloom signatures

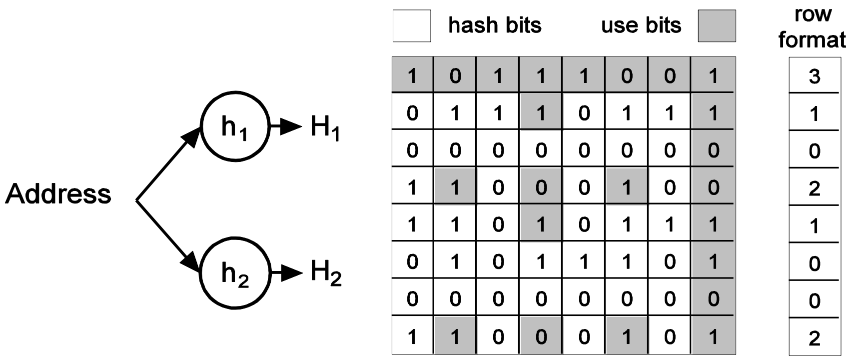


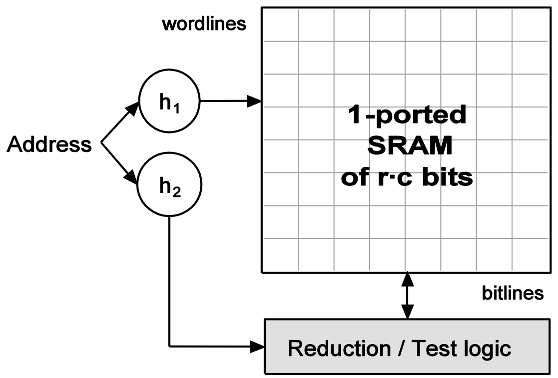
	bucket 0			bucket 1			
WC	H_1	H_2	E	H_1	H_2	E	
0				0	4	562	set 0
0							
0				6	2	453	
0	3	3	156	3	5	942	
0	4	0	244	5	4	671	
0				2	5	027	
1							
0	7	1	391	7	7	234	set 7

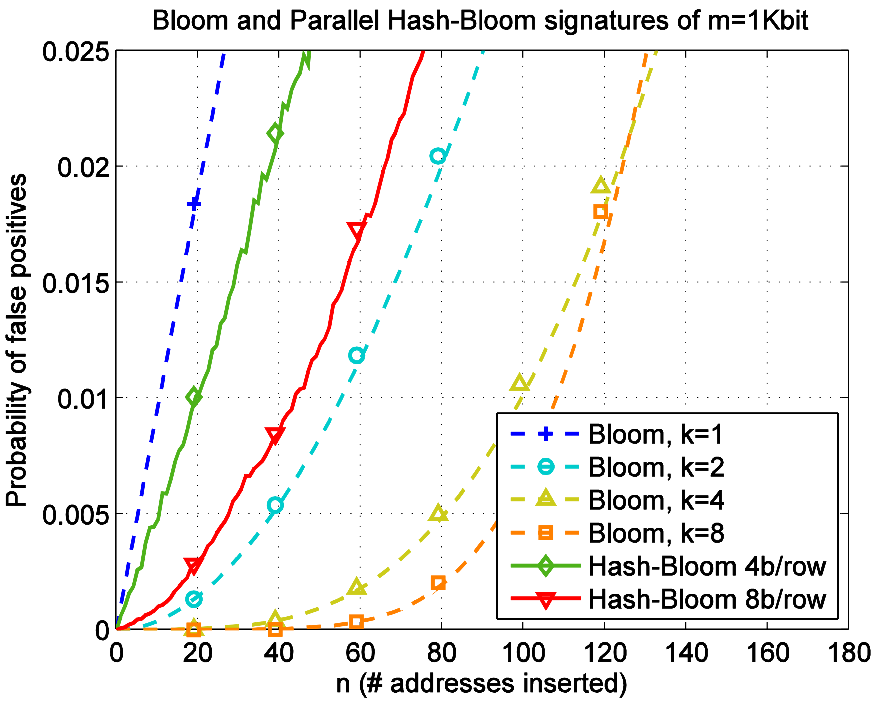


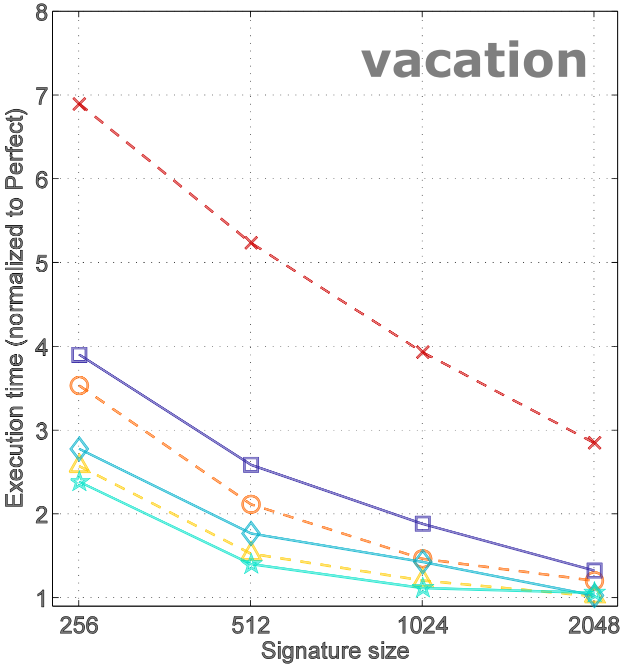


Backup – Hash-Bloom signatures









Backup – Adaptive Bloom signatures

Prediction table

PC	RS size	WS size
0x12f74	25	7
0x13410	138	2
0x104d8	4	1
-	-	-