

Castle: Reinventing Storage for Big Data

Tom Wilkie

Founder & VP Engineering



Acunu

Before the Flood

1990

Small databases

BTree indexes

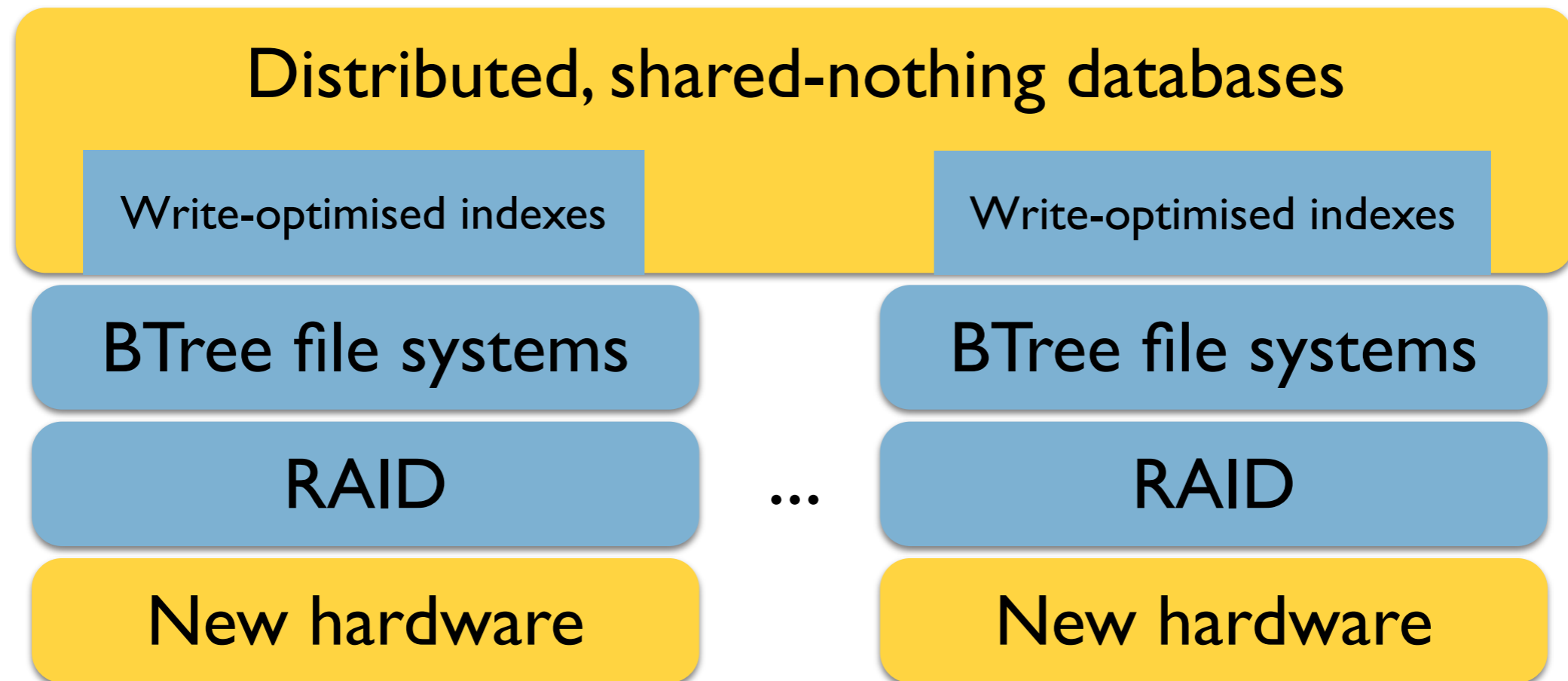
BTree File systems

RAID

Old hardware

Two Revolutions

2010



Bridging the Gap

2011

Distributed, shared-nothing databases

Castle

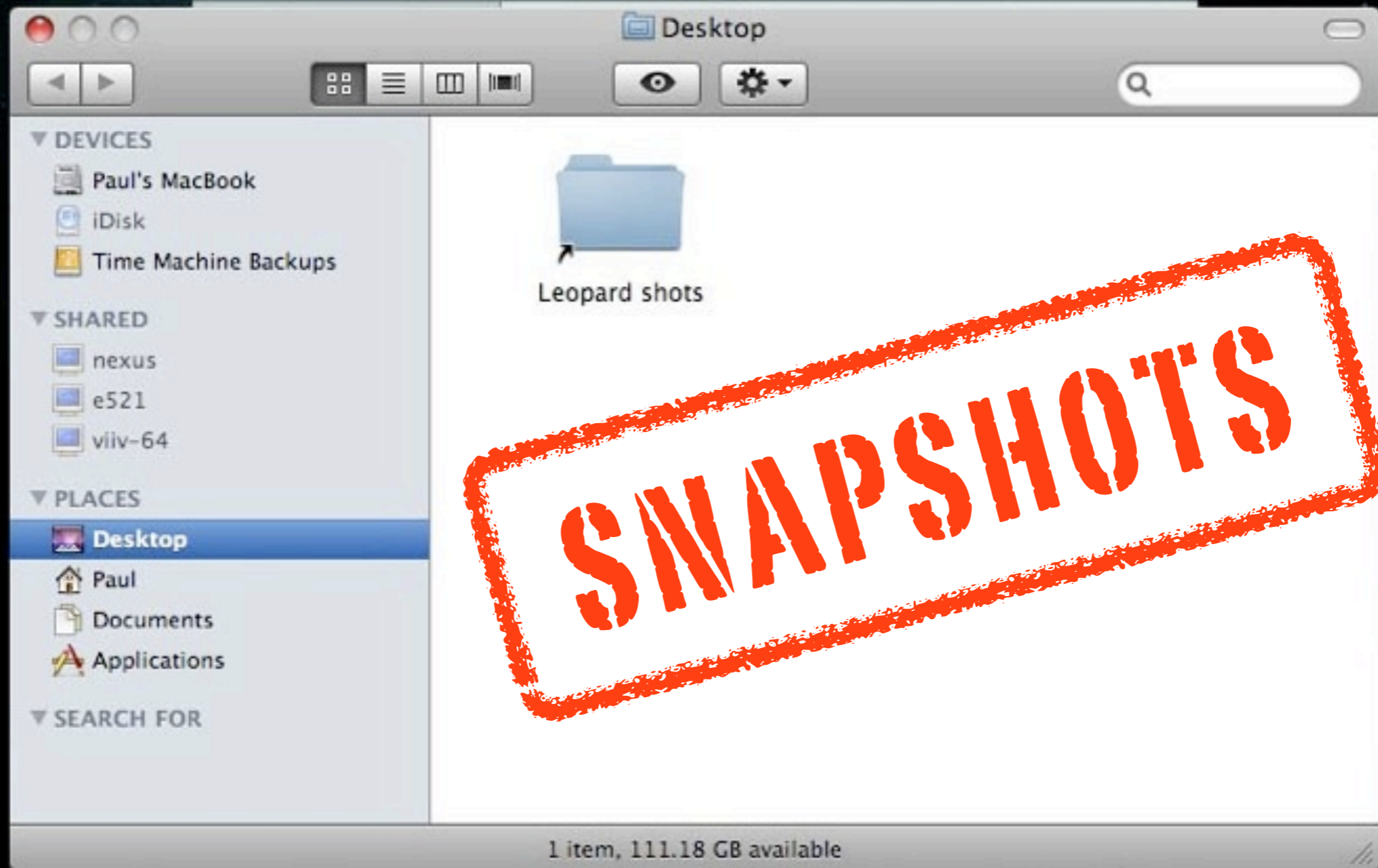
...

Castle

New hardware

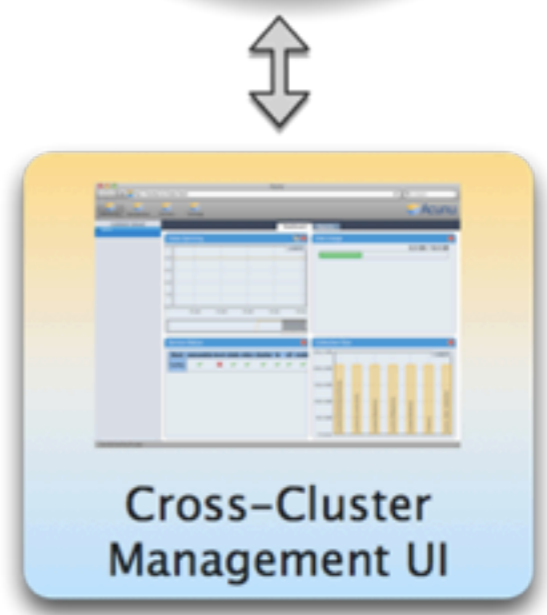
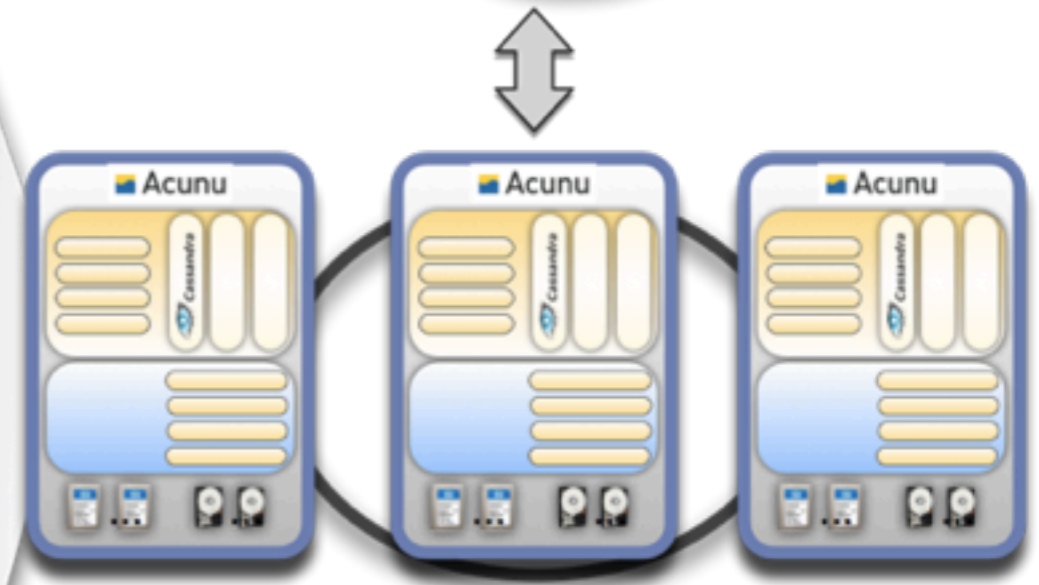
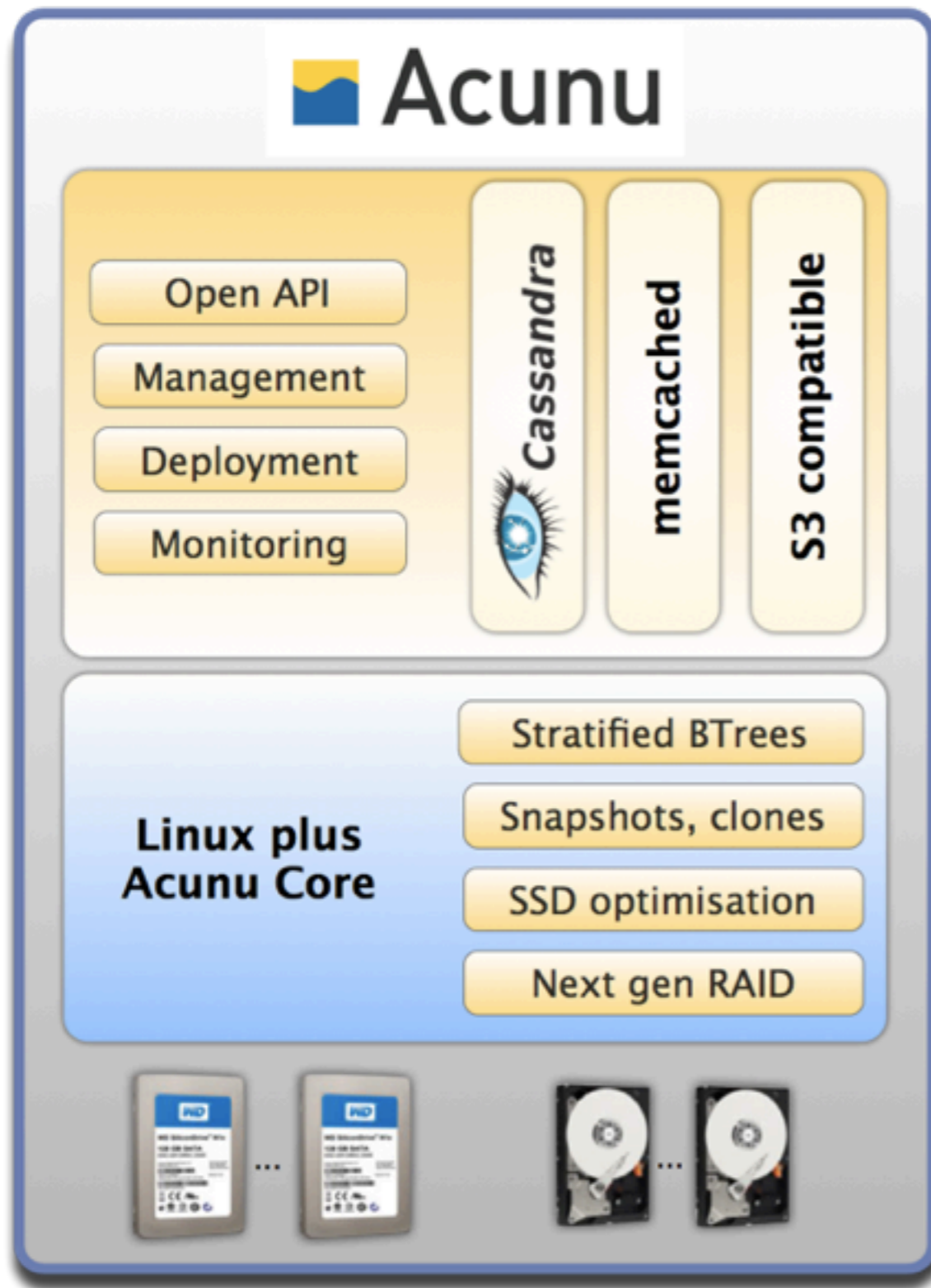
New hardware

With Big Data, how do I...

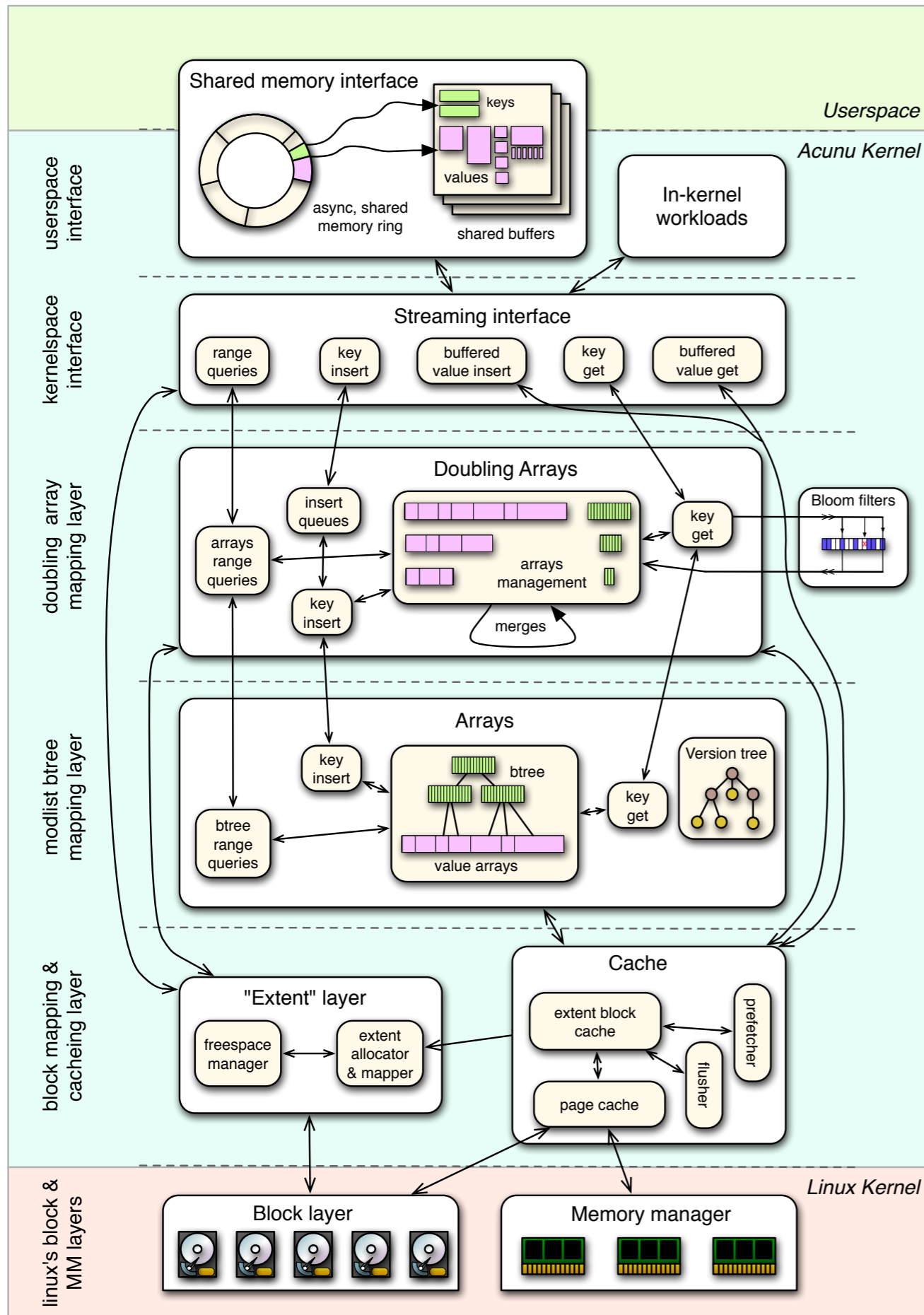




What's in the Castle?

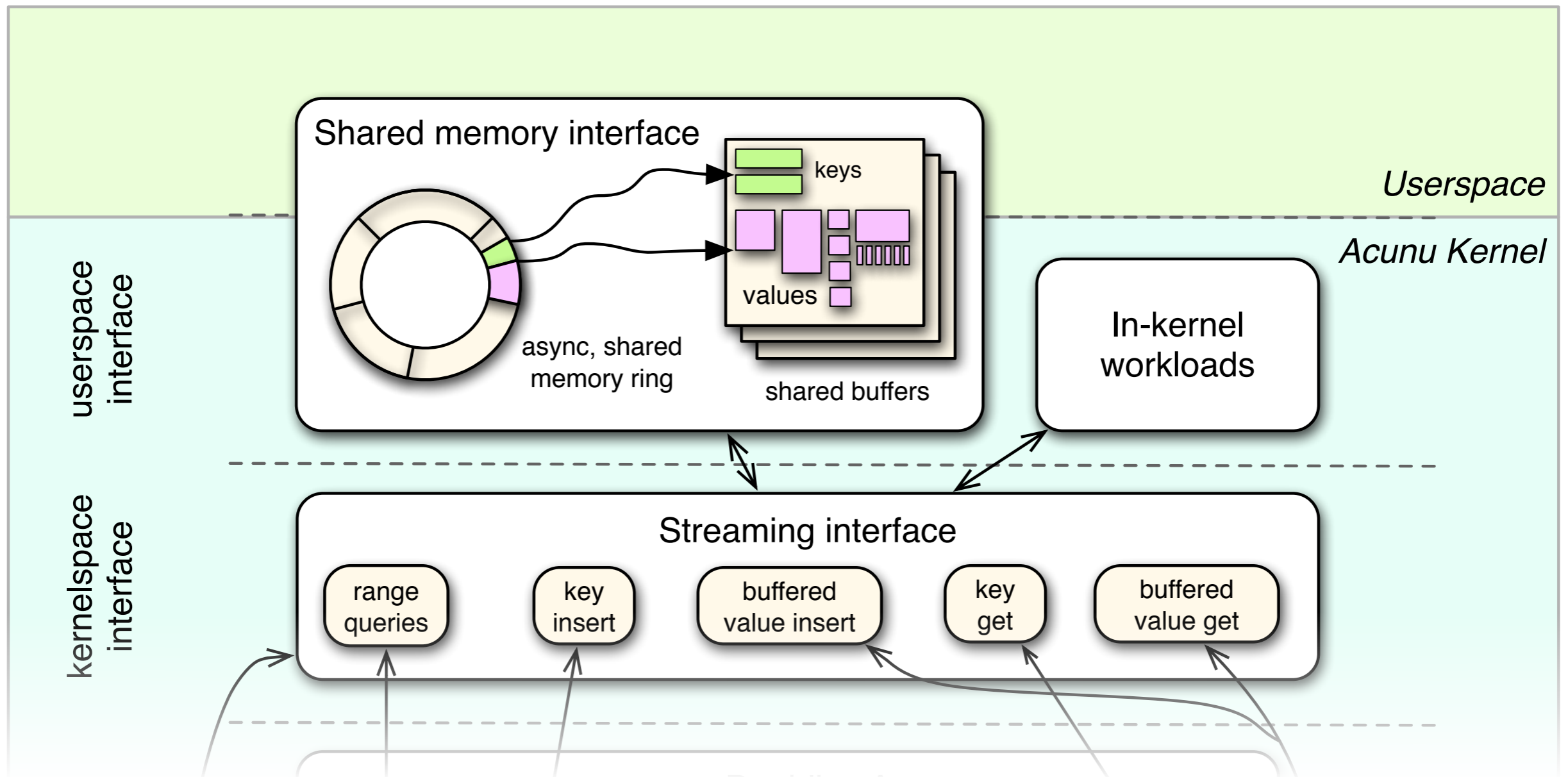


Castle



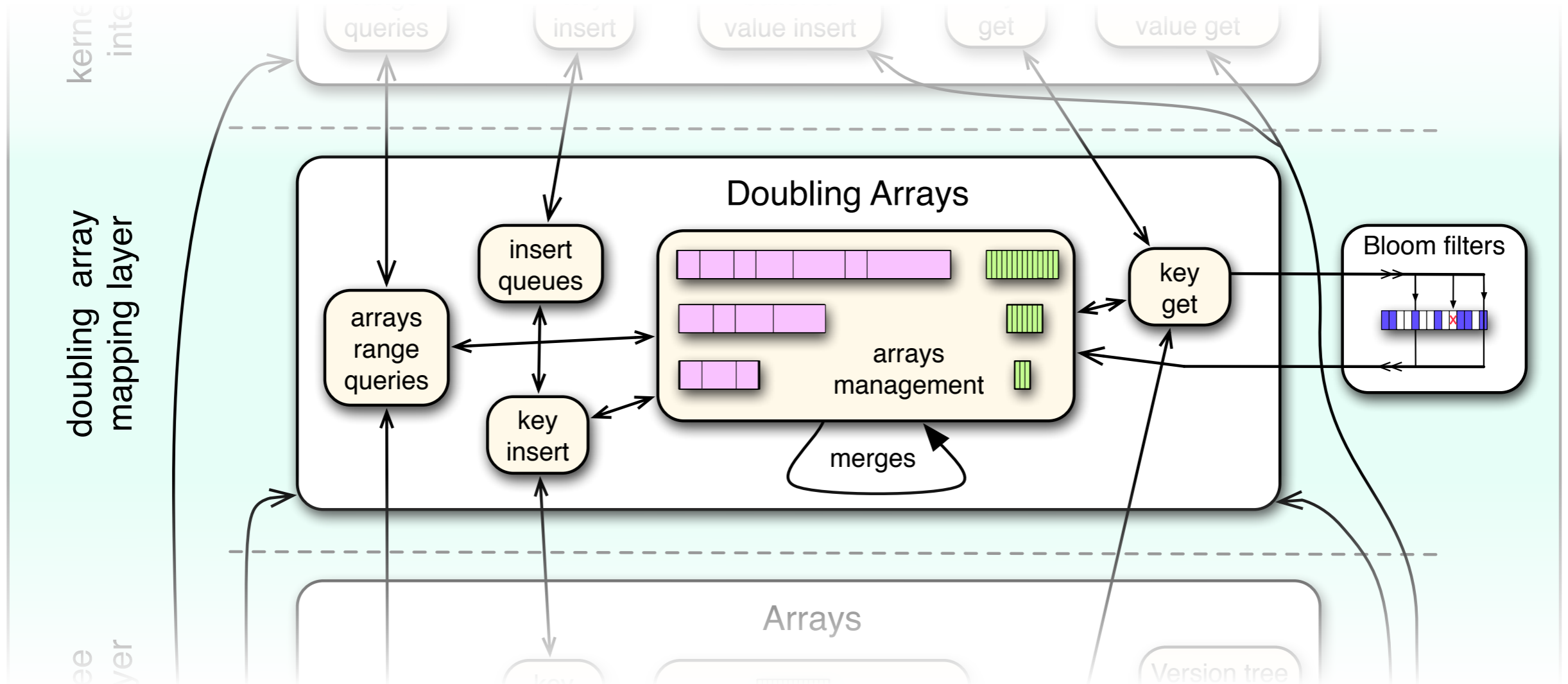
- Opensource (GPLv2, MIT for user libraries)
- <http://bitbucket.org/acunu>
- Loadable Kernel Module, targeting CentOS's 2.6.18
- <http://www.acunu.com/blogs/andy-twigg/why-acunu-kernel/>

The Interface



`castle_{back,objects}.c`

Doubling Array



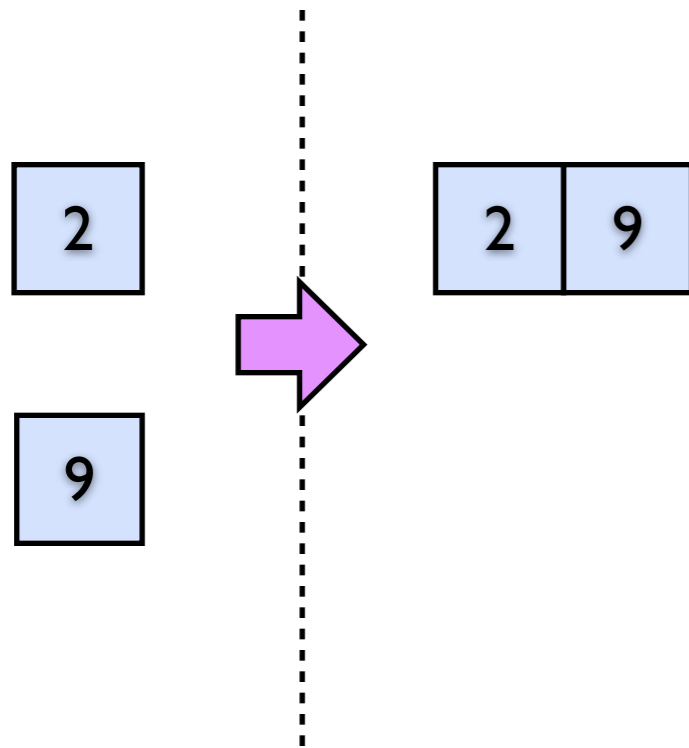
`castle_{da,bloom}.c`

	Update	Range Query (Size Z)
B-Tree	$O(\log_B N)$ random IOs	$O(Z/B)$ random IOs

B = “block size”, say 8KB at 100 bytes/entry \approx 100 entries

Doubling Array

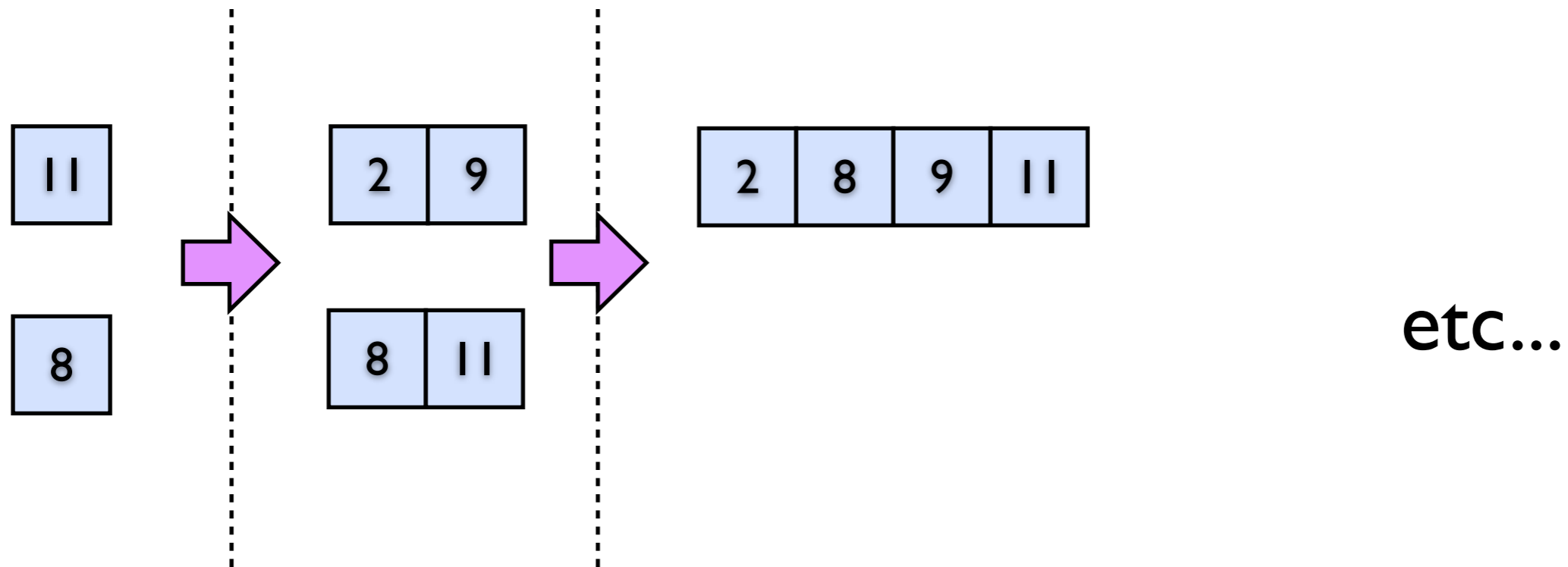
Inserts



Buffer arrays in memory
until we have $> B$ of them

Doubling Array

Inserts



Similar to log-structured merge trees (LSM), cache-oblivious lookahead array (COLA), ...

Demo

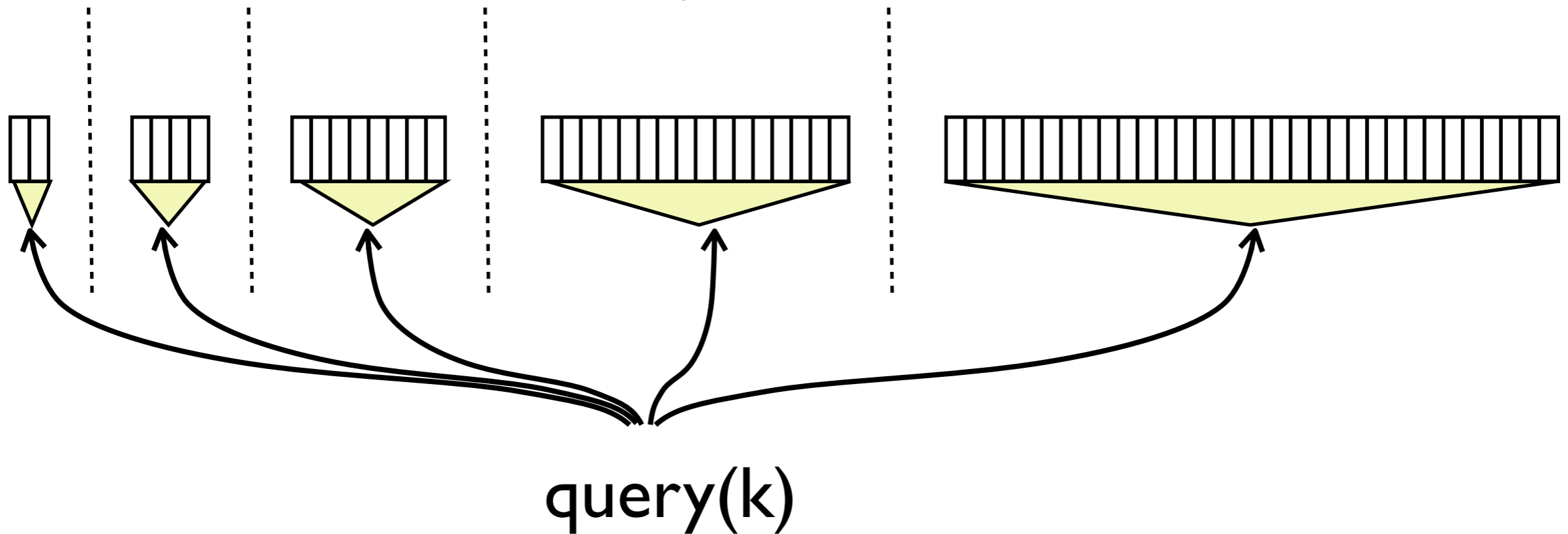
<https://acunu-videos.s3.amazonaws.com/dajs.html>

	Update	Range Query (Size Z)
B-Tree	$O(\log_B N)$ random IOs	$O(Z/B)$ random IOs
Doubling Array	$O((\log N)/B)$ sequential IOs	

B = “block size”, say 8KB at 100 bytes/entry \approx 100 entries

Doubling Array

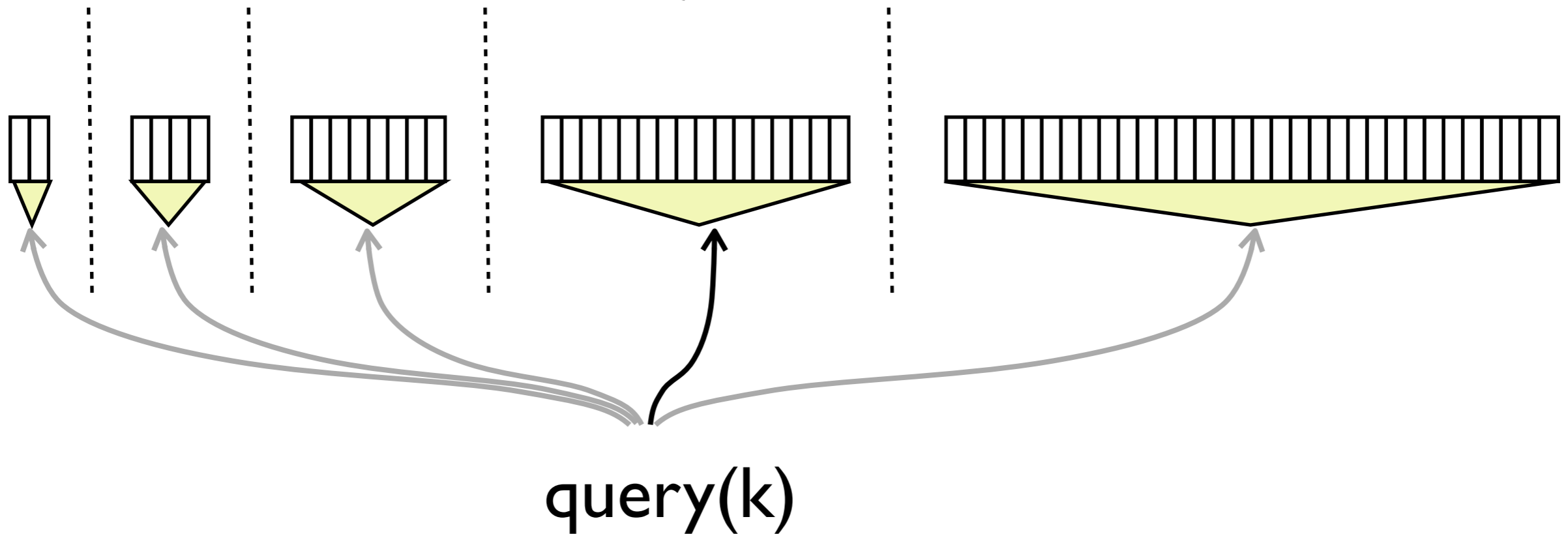
Queries



- Add an index to each array to do lookups
- $\text{query}(k)$ searches each array independently

Doubling Array

Queries



- Bloom Filters can help exclude arrays from search
- ... but don't help with range queries

8KB @ 100MB/s, w/ 8ms seek
= **100 IOs/s**

100 / 5
= **20 updates/s**

$\sim \log(2^{30}) / \log 100$
= **5 IOs/update**

	Update	Range Query (Size Z)
B-Tree	$O(\log_B N)$ random IOs	$O(Z/B)$ random IOs
Doubling Array	$O((\log N)/B)$ sequential IOs	$O(Z/B)$ sequential IOs

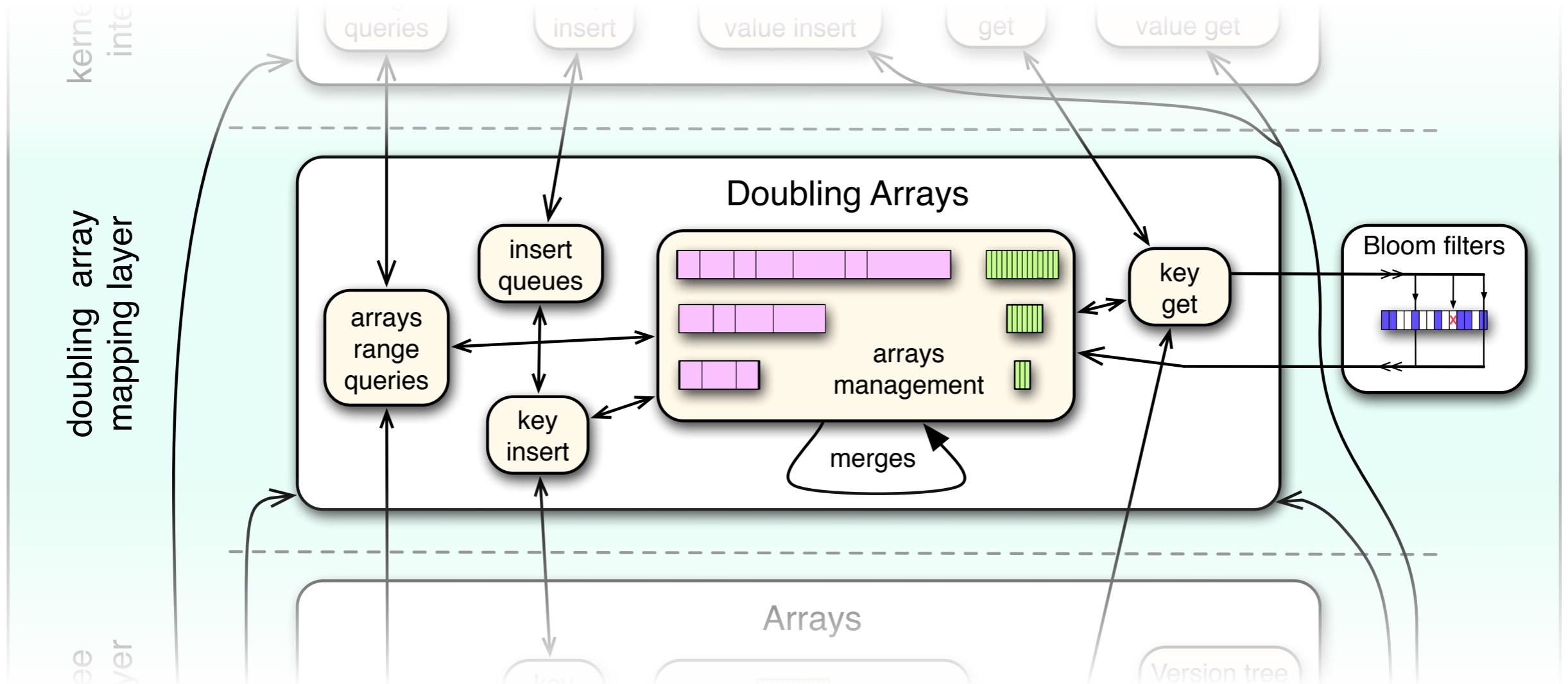
$\sim \log(2^{30}) / 100$
= **0.2 IOs/update**

8KB @ 100MB/s
= **13k IOs/s**

13k / 0.2
= **65k updates/s**

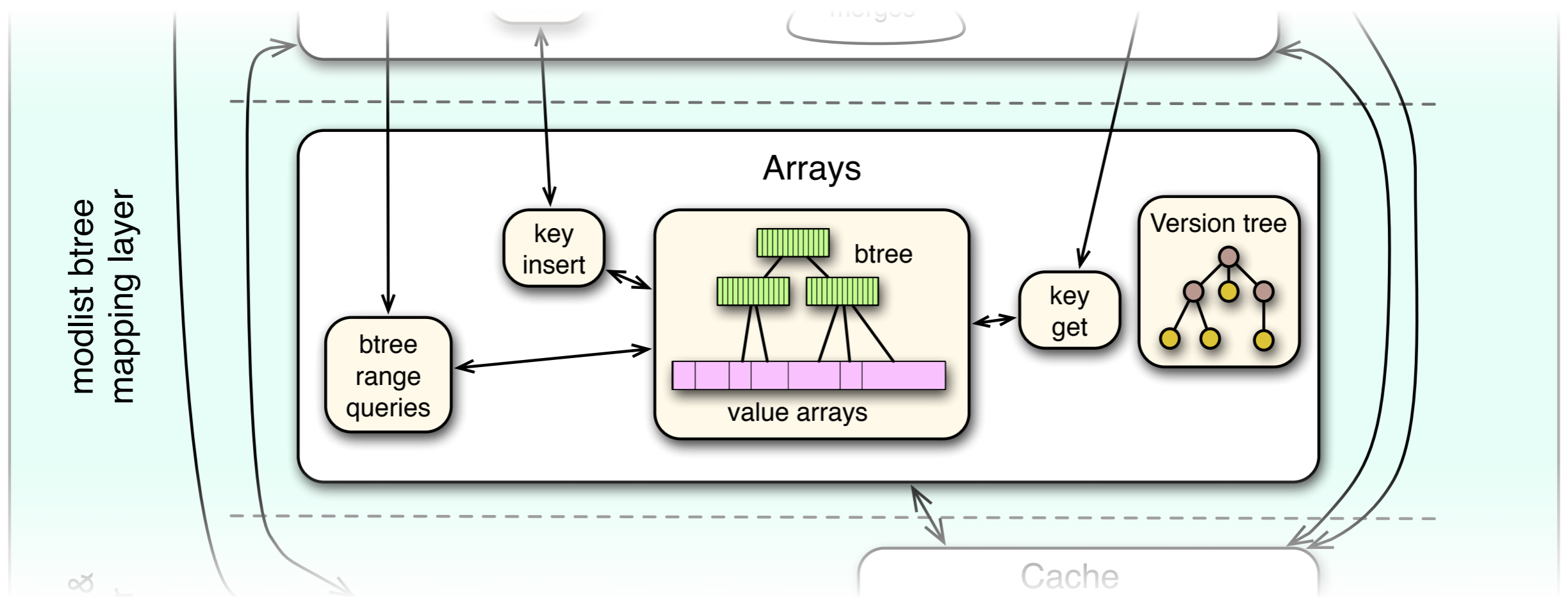
B = "block size", say 8KB at 100 bytes/entry \sim 100 entries

Doubling Array



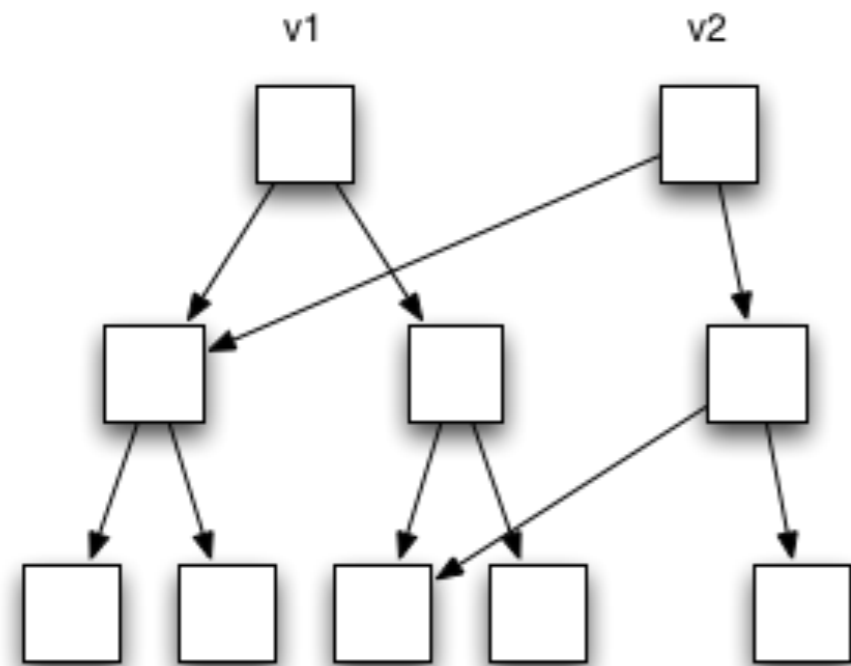
`castle_{da,bloom}.c`

“Mod-list” B-Tree



`castle_{btree,versions}.c`

Copy-on-Write BTree



Idea:

- Apply *path-copying* [DSST] to the B-tree

Problems:

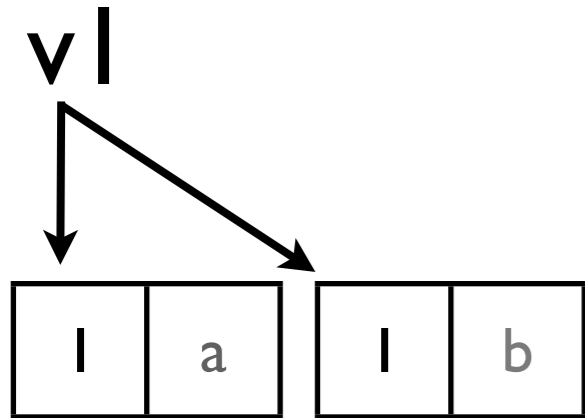
- **Space blowup**: Each update may rewrite an entire path
- **Slow updates**: as above

A log file system makes *updates* sequential, but relies on random access and garbage collection (achilles heel!)

	Update	Range Query	Space
CoW B-Tree	$O(\log_B N_v)$ random IOs	$O(Z/B)$ random IOs	$O(N B \log_B N_v)$

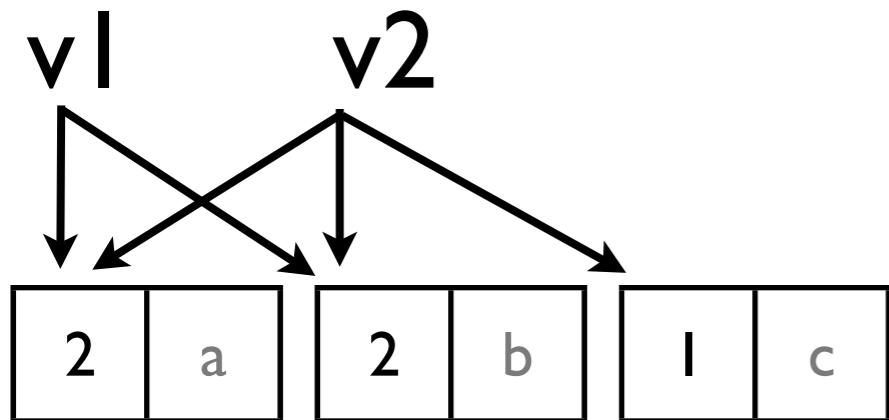
$N_v = \#keys \text{ live (accessible) at version } v$

“BigTable” snapshots



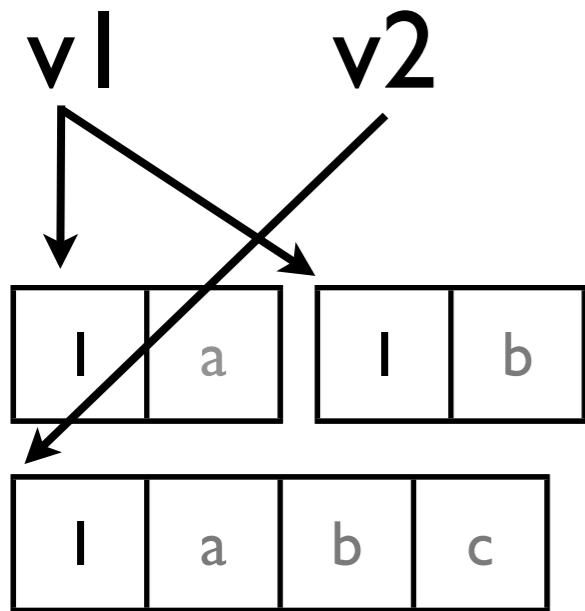
- Inserts produce arrays

“BigTable” snapshots



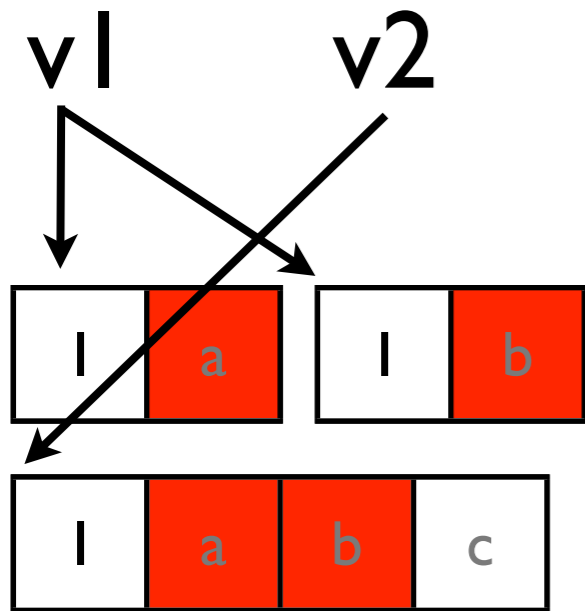
- Inserts produce arrays
- Snapshots increment ref counts on arrays
- Merges produce more arrays, decrement ref count on old arrays

“BigTable” snapshots



- Inserts produce arrays
- Snapshots increment ref counts on arrays
- Merges produce more arrays, decrement ref count on old arrays

“BigTable” snapshots

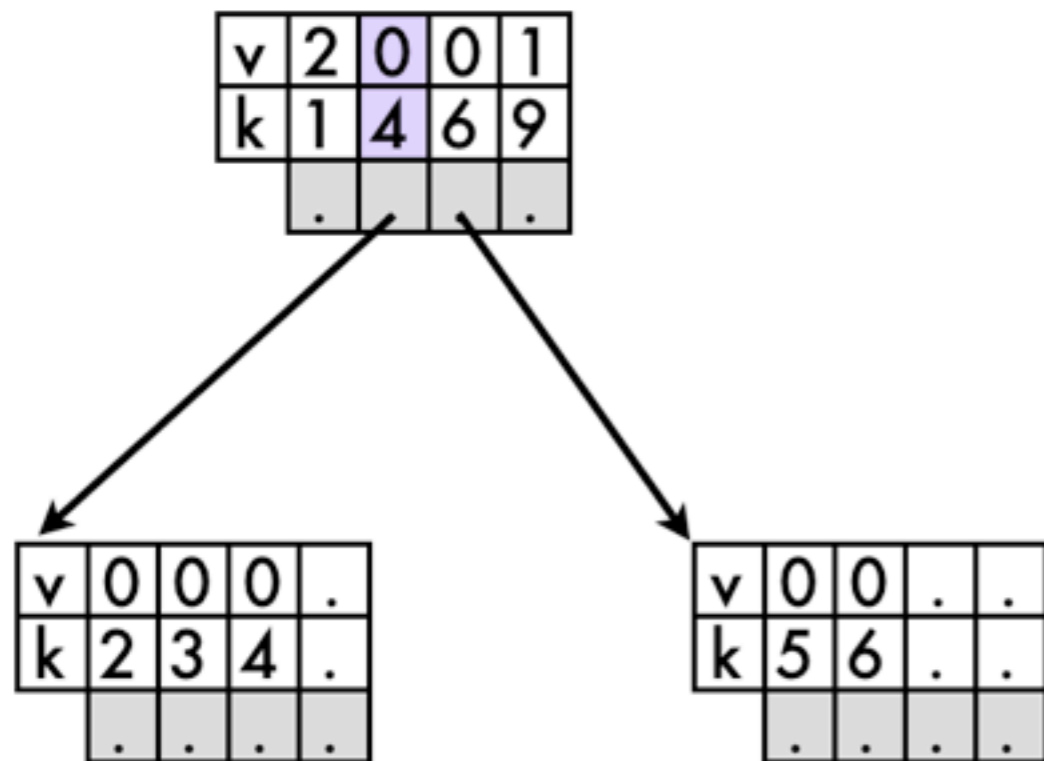


- Inserts produce arrays
- Snapshots increment ref counts on arrays
- Merges produce more arrays, decrement ref count on old arrays
- Space blowup

	Update	Range Query	Space
CoW B-Tree	$O(\log_B N_v)$ random IOs	$O(Z/B)$ random IOs	$O(N B \log_B N_v)$
“BigTable” style DA	$O((\log N)/B)$ sequential IOs	$O(Z/B)$ sequential IOs	$O(VN)$

$N_v = \#keys \text{ live (accessible) at version } v$

“Mod-list” BTree





Idea:

- Apply *fat-nodes* [DSST] to the B-tree
- ie insert (key, version, value) tuples, with special operations

Problems:

- Similar performance to a BTree

If you limit the #versions, can be constructed sequentially, and embedded into a DA

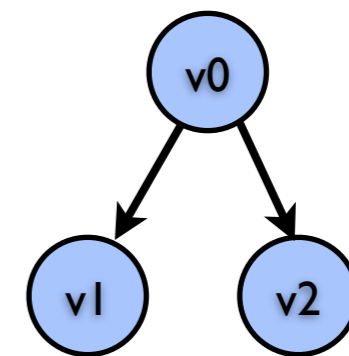
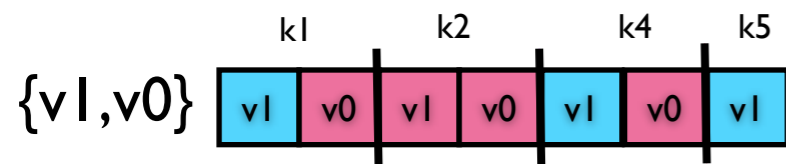
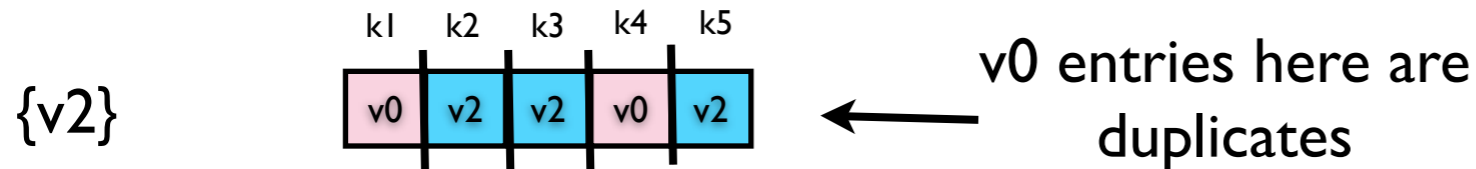
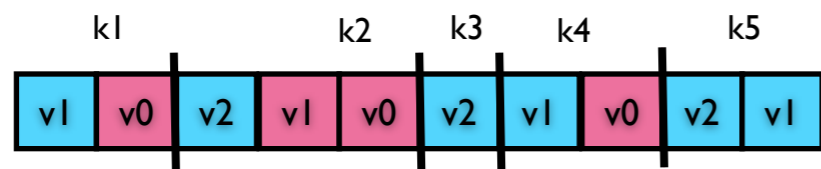
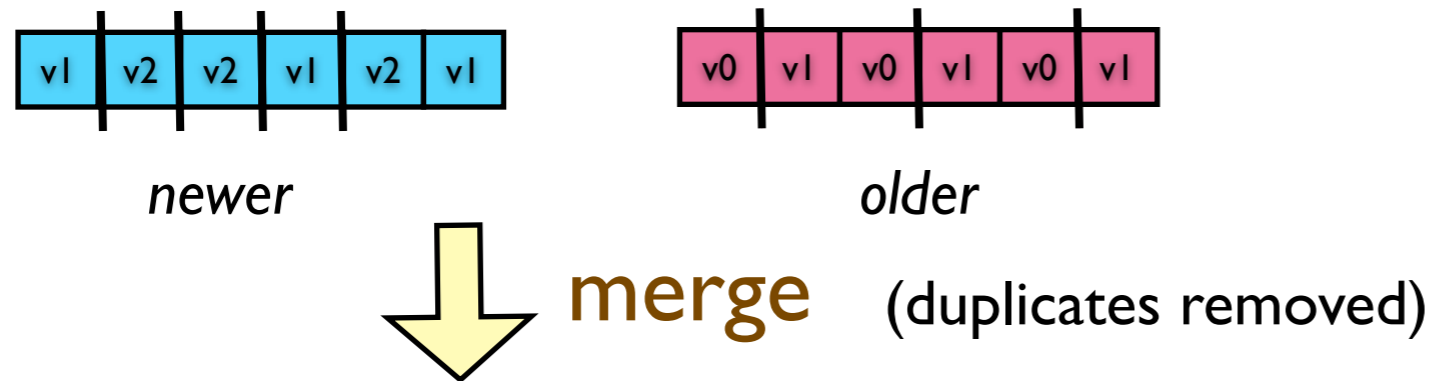
	Update	Range Query	Space
	$O(\log_B N_v)$ random IOs	$O(Z/B)$ random IOs	$O(N B \log_B N_v)$
	$O((\log N)/B)$ sequential IOs	$O(Z/B)$ sequential IOs	$O(VN)$
CASTLE	$O((\log N)/B)$ sequential IOs	$O(Z/B)$ sequential IOs	$O(N)$

$N_v = \#keys \text{ live (accessible) at version } v$

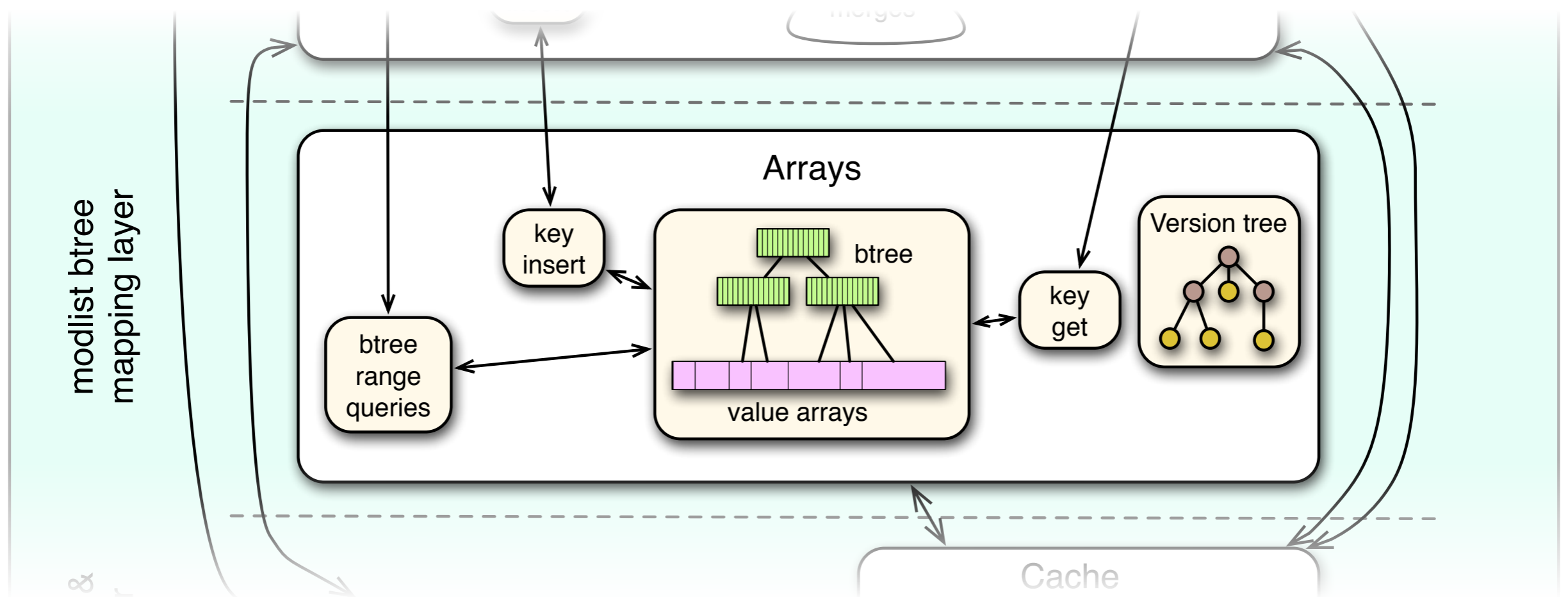
Stratified BTree

Problem:
Embedded “Mod-
list” #versions limit

Solution:
Version-split arrays
during merges

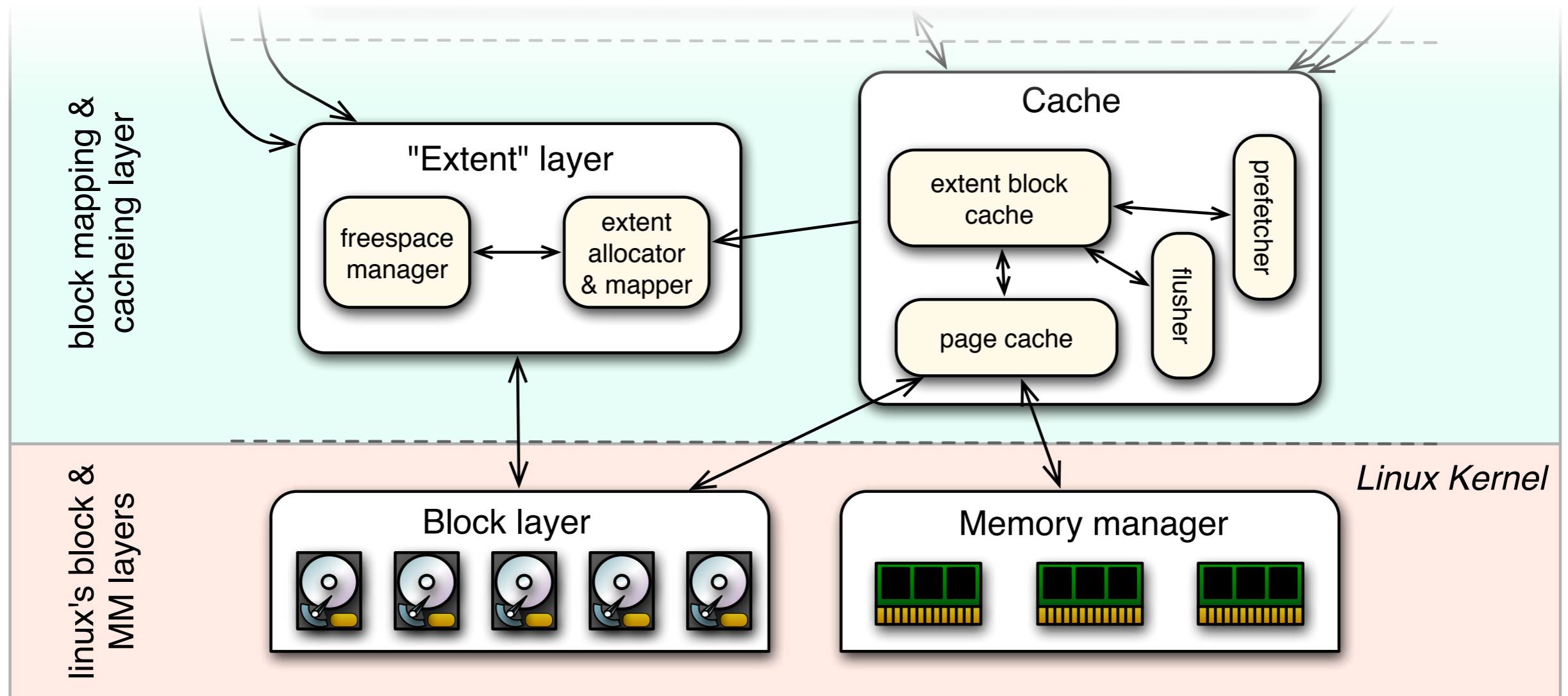


“Mod-list” B-Tree



`castle_{btree,versions}.c`

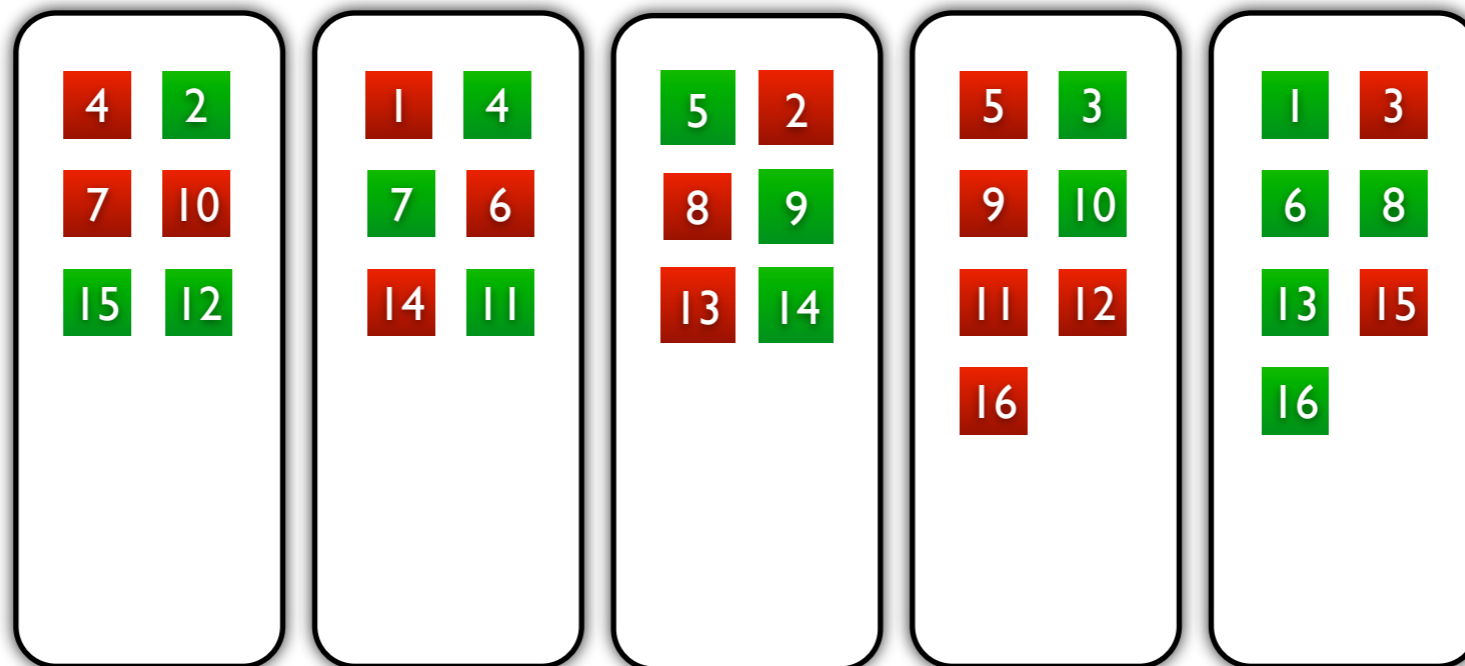
Disk Layout: RDA




`castle_{cache,extent,freespace,rebuild}.c`

Disk Layout: RDA

random duplicate allocation

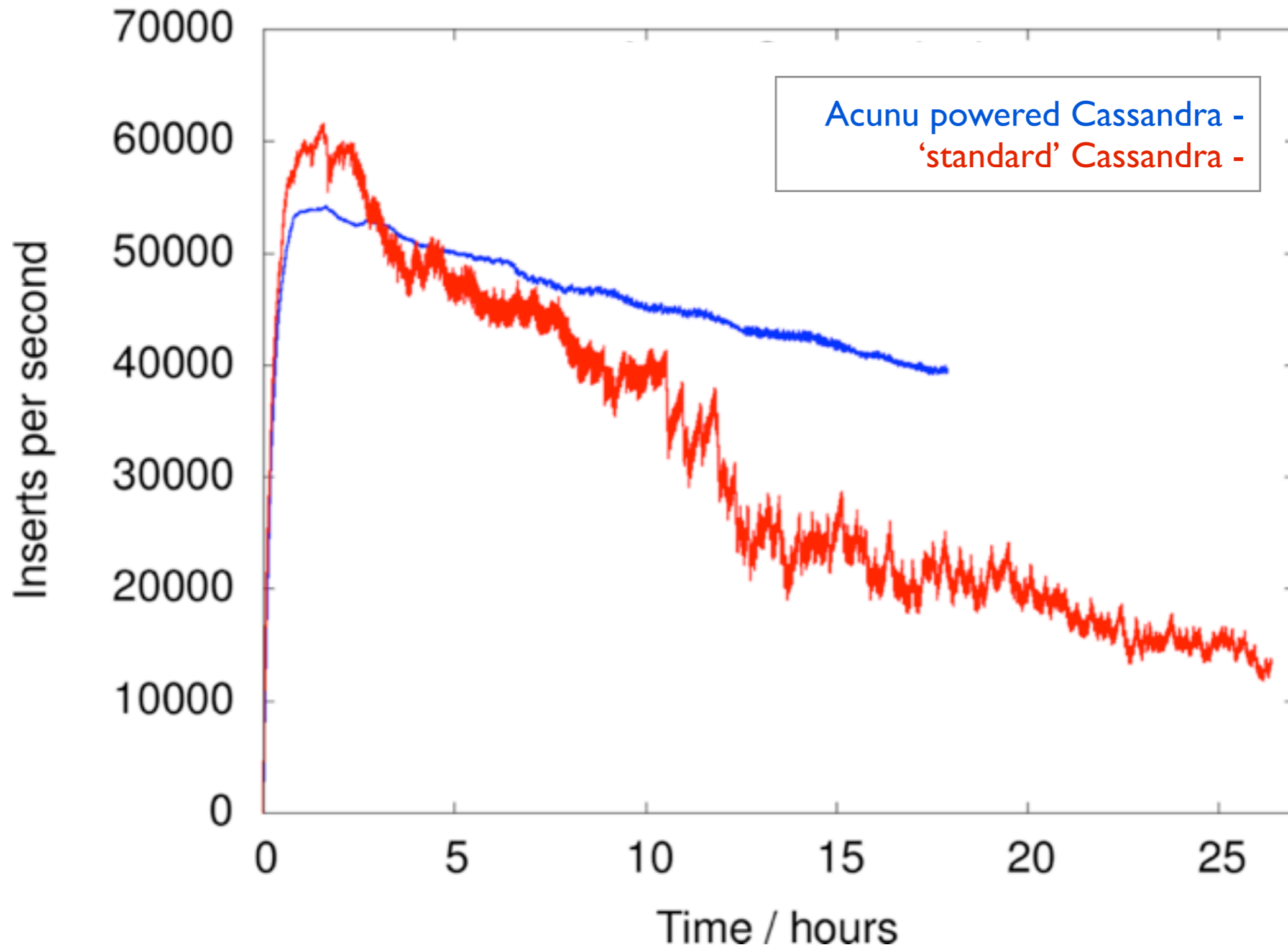




Performance Comparison

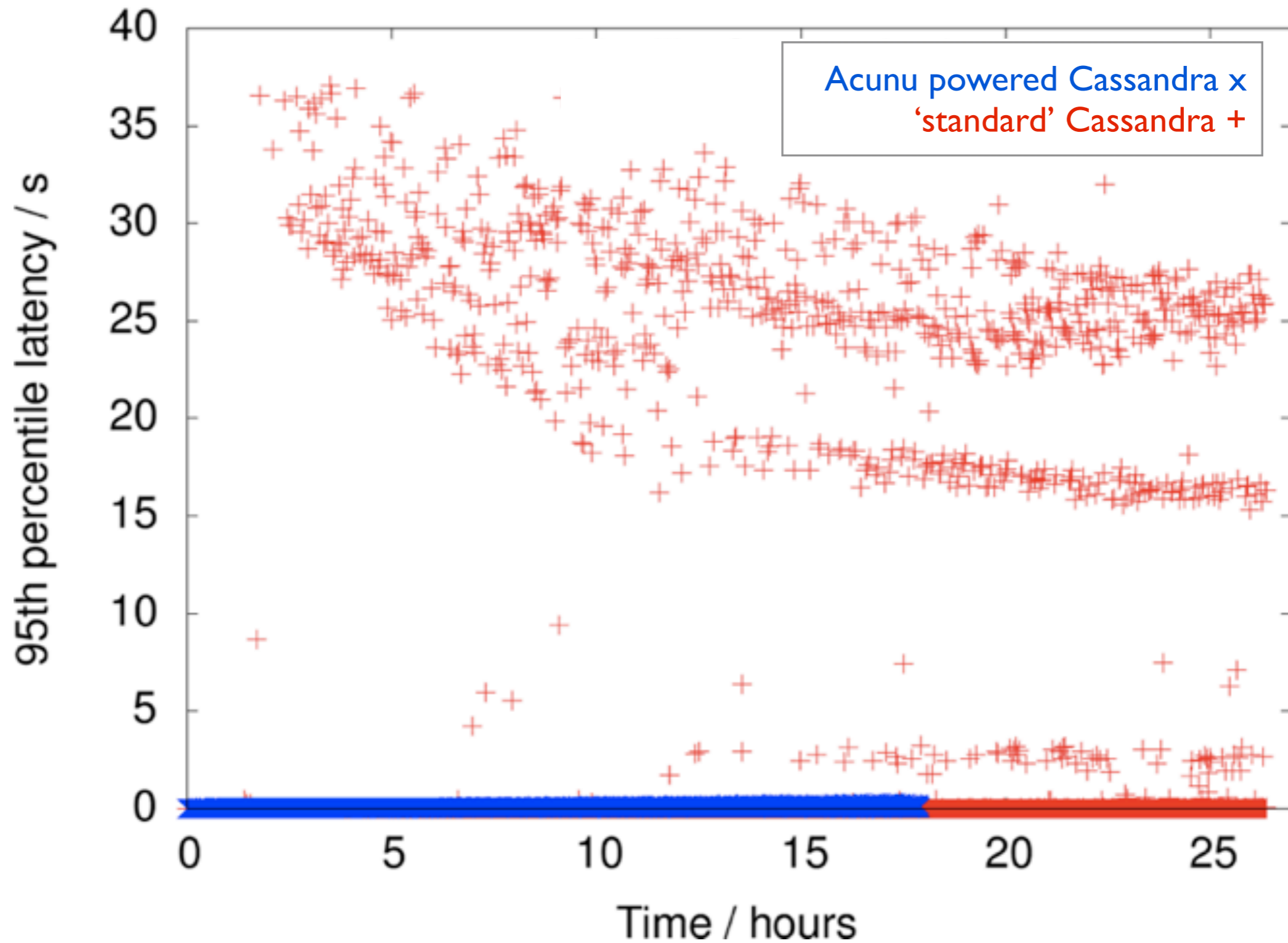
Small random inserts

Inserting 3 billion rows



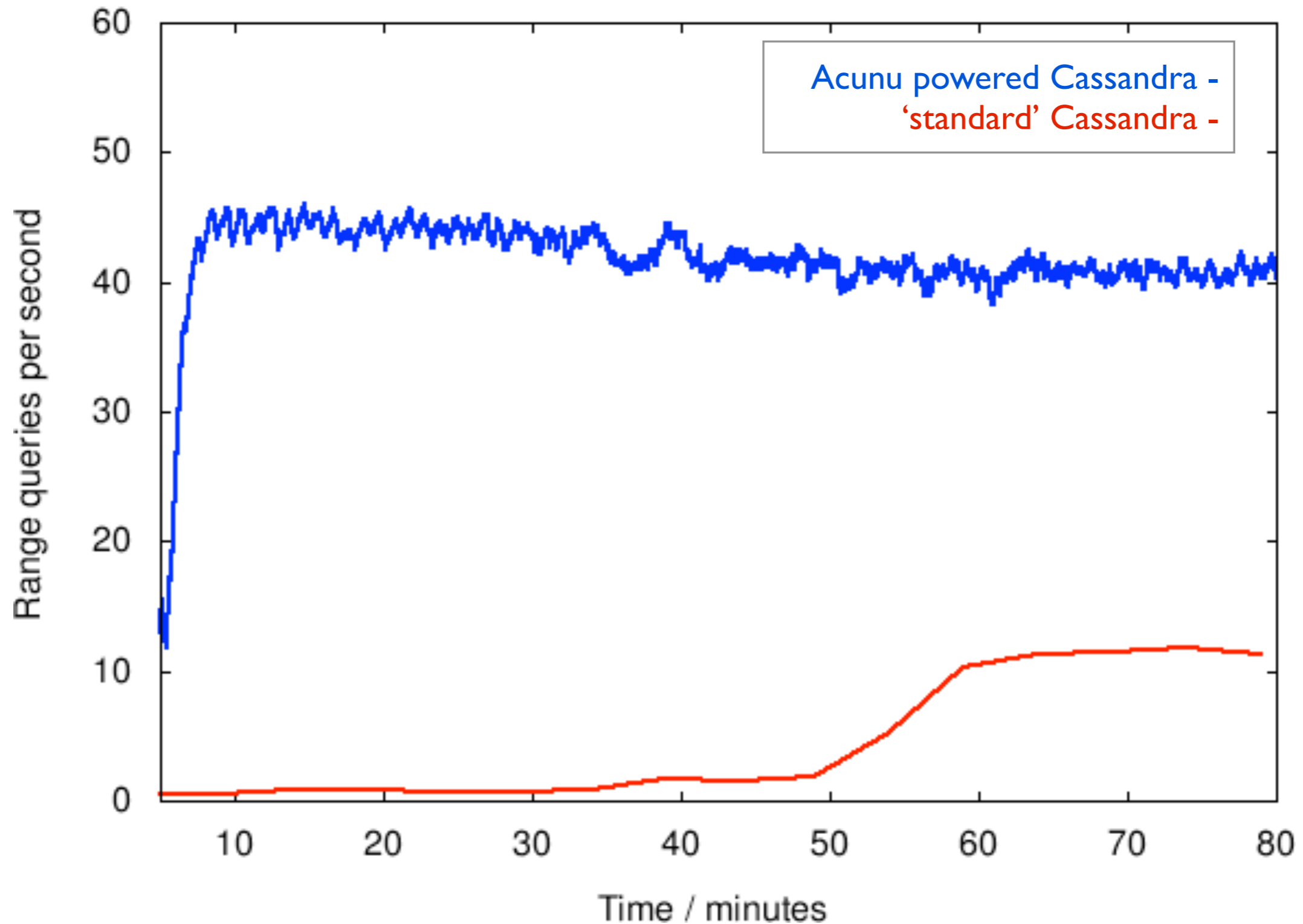
Insert latency

While inserting 3 billion rows



Small random range queries

Performed immediately after inserts



Performance summary

	Standard	Acunu	Benefits
inserts rate 95% latency	~32k/s ~32s	~45k/s ~0.3s	> 1.4x > 100x
gets rate 95% latency	~100/s ~2s	~350/s ~0.5s	> 3.5x > 4x
range queries 95% latency	~0.4/s ~15s	~40/s ~2s	> 100x > 7.5x

- Castle: like BDB, but for Big Data
- DA: transforms random IO into sequential IO
- Snapshots & Clones: addressing real problems with new workloads
- 2 orders of magnitude better performance and predictability

Questions?

Tom Wilkie

@tom_wilkie

tom@acunu.com

<http://bitbucket.org/acunu>

<http://www.acunu.com/download>

<http://www.acunu.com/insights>



References

[LSM] The Log-Structured Merge-Tree (LSM-Tree)

Patrick O'Neil, Edward Cheng, Dieter Gawlick,
Elizabeth O'Neil

[http://staff.ustc.edu.cn/~jpq/paper/flash/1996-The
%20Log-Structured%20Merge-Tree%20%28LSM-
Tree%29.pdf](http://staff.ustc.edu.cn/~jpq/paper/flash/1996-The%20Log-Structured%20Merge-Tree%20%28LSM-Tree%29.pdf)

[COLA] Cache-Oblivious Streaming B-trees,

Michael A. Bender et al

[http://www.cs.sunysb.edu/~bender/newpub/
BenderFaFi07.pdf](http://www.cs.sunysb.edu/~bender/newpub/BenderFaFi07.pdf)

[DSST] Making Data Structures Persistent - J. R.

Driscoll, N. Sarnak, D. D. Sleator, R. E. Tarjan, Making
Data Structures Persistent, Journal of Computer
and System Sciences, Vol. 38, No. 1, 1989

[http://www.cs.cmu.edu/~sleator/papers/making-
data-structures-persistent.pdf](http://www.cs.cmu.edu/~sleator/papers/making-data-structures-persistent.pdf)

Stratified B-trees and versioned dictionaries, - Andy
Twigg, Andrew Bye, Grzegorz Miłoś, Tim Moreton,
John Wilkes, Tom Wilkie, HotStorage'11

[http://www.usenix.org/event/hotstorage11/tech/
final_files/Twigg.pdf](http://www.usenix.org/event/hotstorage11/tech/final_files/Twigg.pdf)

[RDA] Random duplicate storage strategies for
load balancing in multimedia servers, 2000, Joep
Aerts and Jan Korst and Sebastian Egner

<http://www.win.tue.nl/~joep/IPL.ps>

Apache, Apache Cassandra, Cassandra, Hadoop, and
the eye and elephant logos are trademarks of the
Apache Software Foundation.