Rewriting the storage stack for Big Data

A. Byde, G. Milos, T. Moreton, A. Twigg, T. Wilkie, J. Wilkes
Acunu

Abstract

Over the last two decades, commodity hardware has changed dramatically. In parallel, Big Data, NOSQL and other non-relational stores are becoming mainstream. Yet these applications access hardware via a legacy storage stack – FS, buffer cache, LVM, RAID – built for the hardware and applications of the 80s. Acunu is rewriting the storage stack for these applications and today's hardware, and at the same time deploying some new algorithms and techniques. For example, our file system replacement contains the first O(N)-space external-memory dictionaries that support fast updates (in o(1) IOs) while allowing fully-persistent versioning.

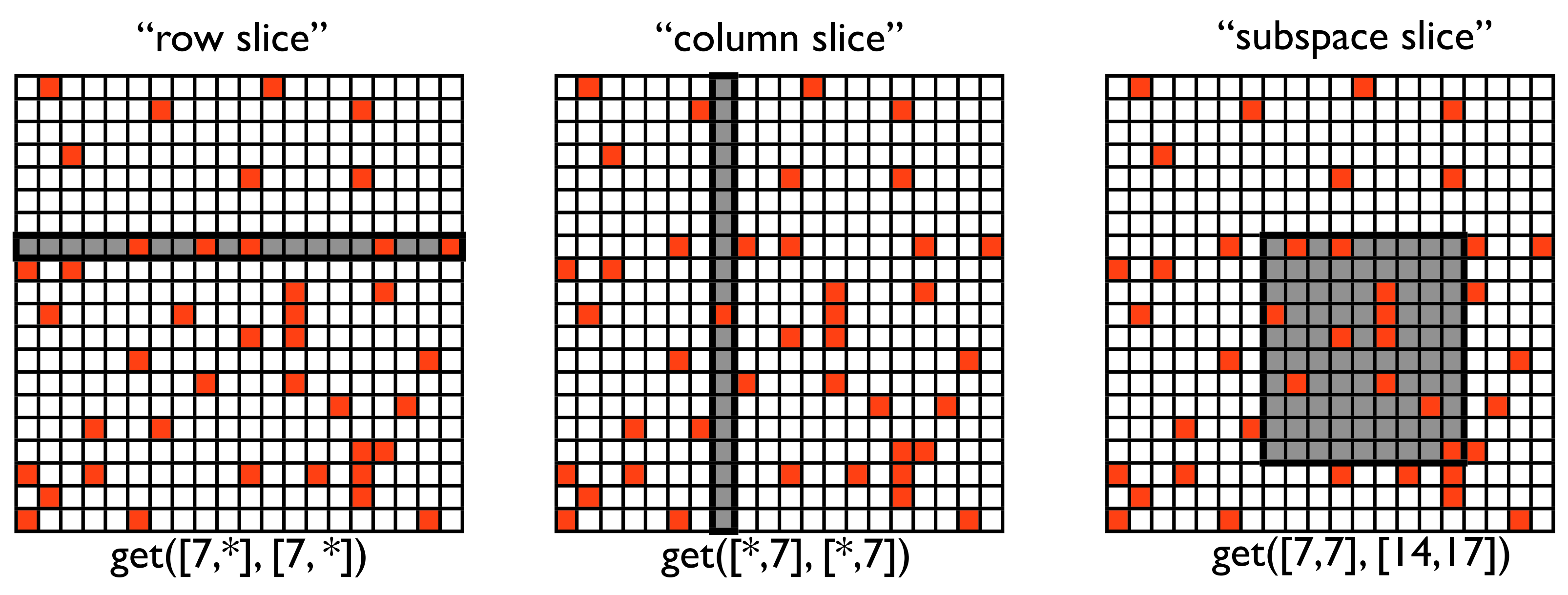
Motivation: think back to 1987..

Year	Capacity	Bandwidth	Random access
2008	2TB	100MB/s	8ms
2012	4.5TB	150MB/s	7ms
2017	67TB	540MB/s	~7ms

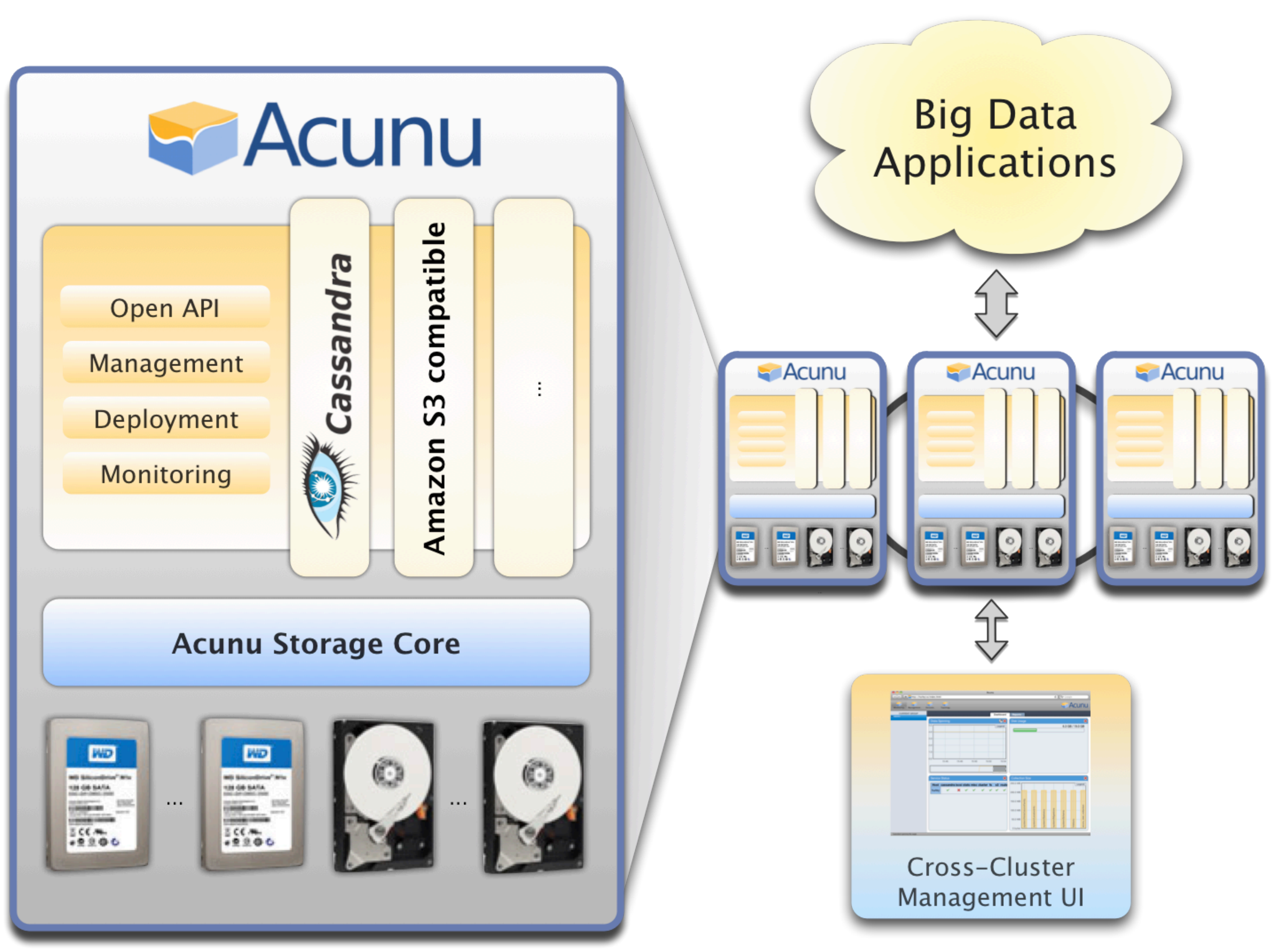
- Dramatic improvements in commodity hardware (large SATA disks, SSDs, many-core, complex memory hierarchies).
- Dramatic changes in scale and workloads
- Revisit assumptions in RAID, disk layouts, caching, and FS interface.
- Take advantage of in-kernel visibility for, eg caching, device control

The stack, revisited

- No more circumventing the interface and writing dictionary data structures inside mmapped files (eg Cassandra)
- Data model: sparse, multidimensional, ordered, versioned dictionaries



- Interface: high-performance asynchronous shared-memory ring buffer (ala Xen devices)

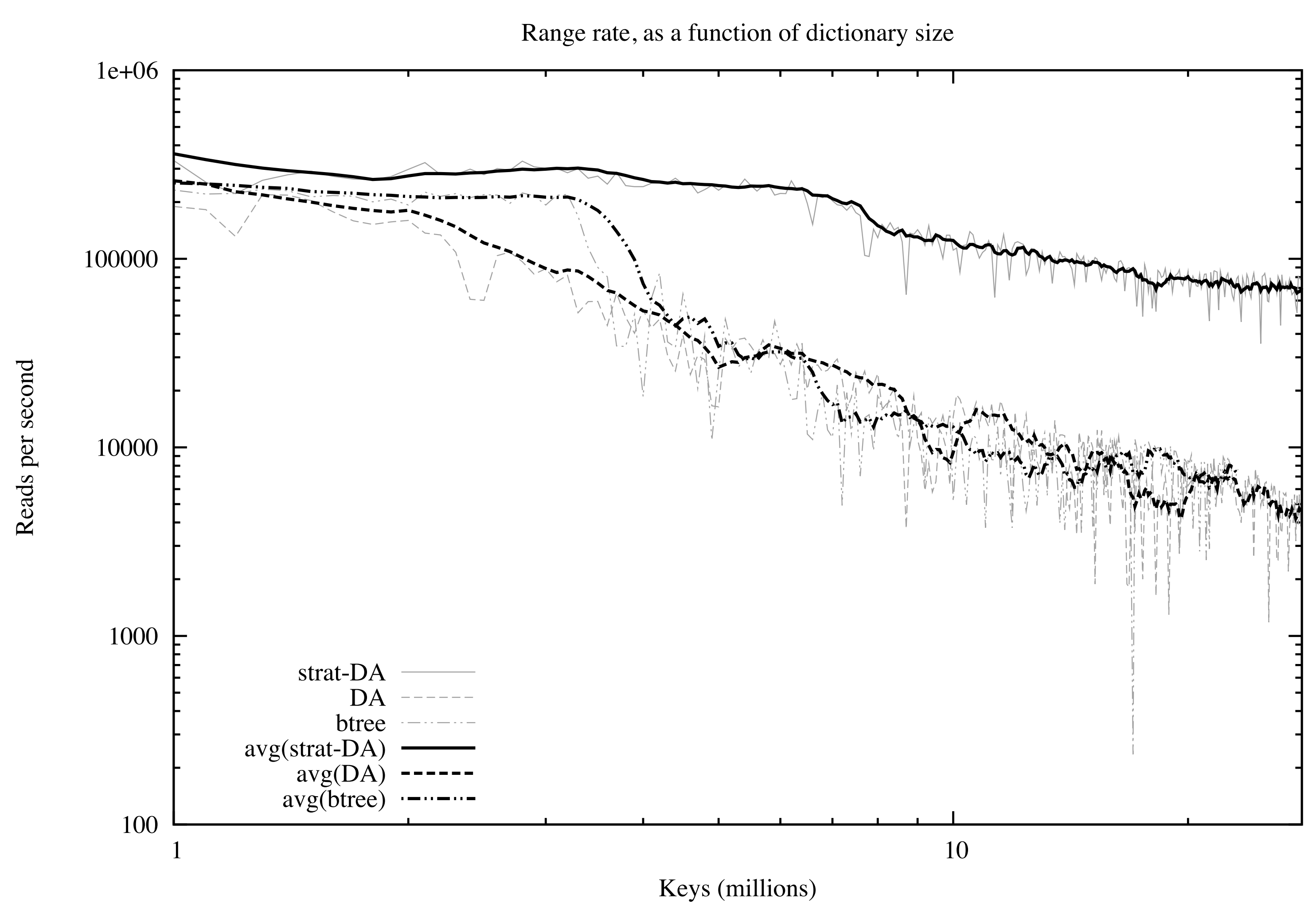


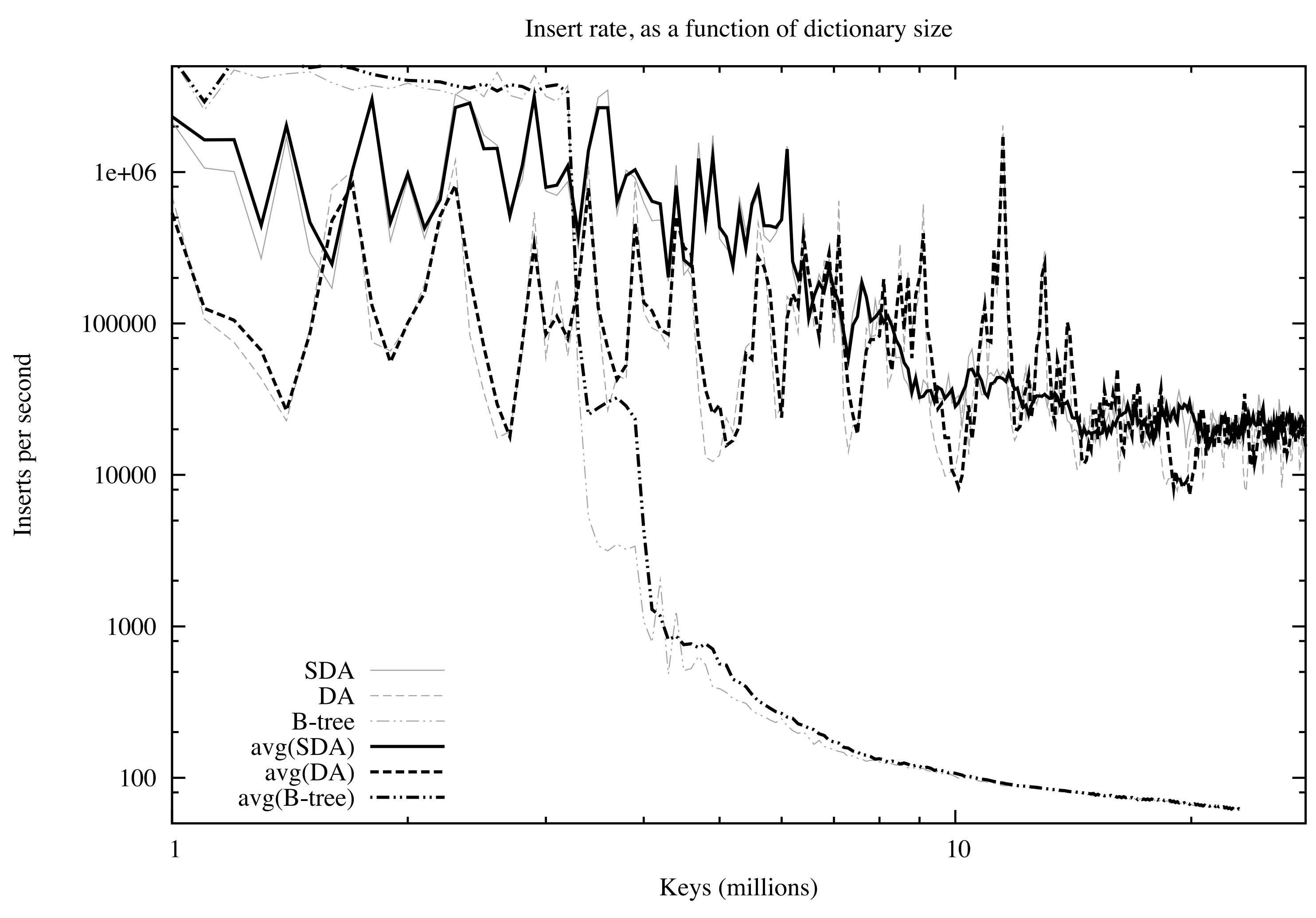
Fast Updates with Versioning

CoW B-trees are an increasingly bad fit for write-heavy workloads and large SATA disks. They need random IO to scale, aren't space optimal, and are optimized for read-heavy workloads. Using a log file system doesn't really help.

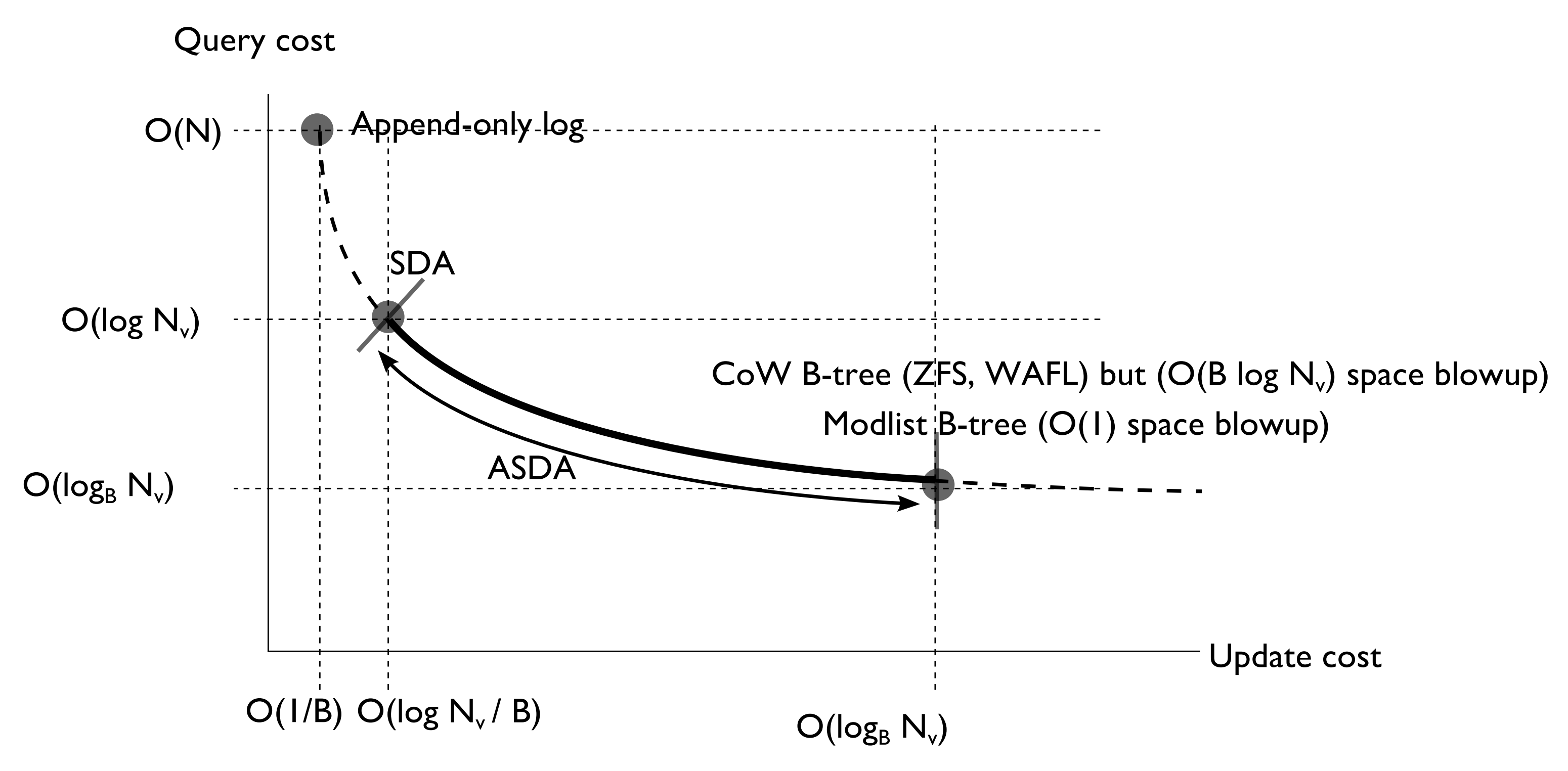
Castle uses new cache-oblivious structures - the first data structures to offer fast updates with fully-persistent versioning.

- Data structures don't age as versions diverge
- Make excellent use of sequential IO
- Naturally deployed onto SSDs, avoiding write coalescing





Query-update tradeoff for versioned dictionaries



RAID, revisited

- Capacity is cheap, but only if you can safely use the cheapest disks
- Randomized disk layouts and scheduling offer guarantees on load balancing and rebuild time
- No hotspots, no hot spares, no need for complex triple-parity schemes.

(for background, see eg RDA [Sanders et al.])