Information Management

IBM

In-Memory Databases

Trends and Technologies

Vilho Raatikka

solidDB Development
IBM Helsinki Lab

March 2012

Author

Vilho Raatikka, Software Engineer, IBM solidDB core development,
Vilho.Raatikka@fi.ibm.com, +358 50 372 0394

- **Background in research at University of Helsinki**
  - RODAIN : in-memory database indexing
  - SeCo : database migration, RDF(S) databases
- **Joined Solid Information Technology in 2004, IBM acquired Solid in 2008.**
- **Started as a performance tester, and technical writer on Solid.**
- **Since 2005 concentrated mostly design & implementing in-memory indexing, high-availability (HA) technologies, and improving the use of multi-core processors.**

Overview

- Motivation behind in-memory databases (imdb).
- What makes imdb faster?
- Where is it used best?
- Is imdb suitable for Big Data?

Disk-based databases

- For decades, databases have provided robust, and popular set of features for applications.
- Disk-based databases utilize cheap, and large storage.
- They provide large volume, but long disk latency is hard to overcome.
- Even with large page buffers, disk access causes unpredictable long response times.

Therefore:

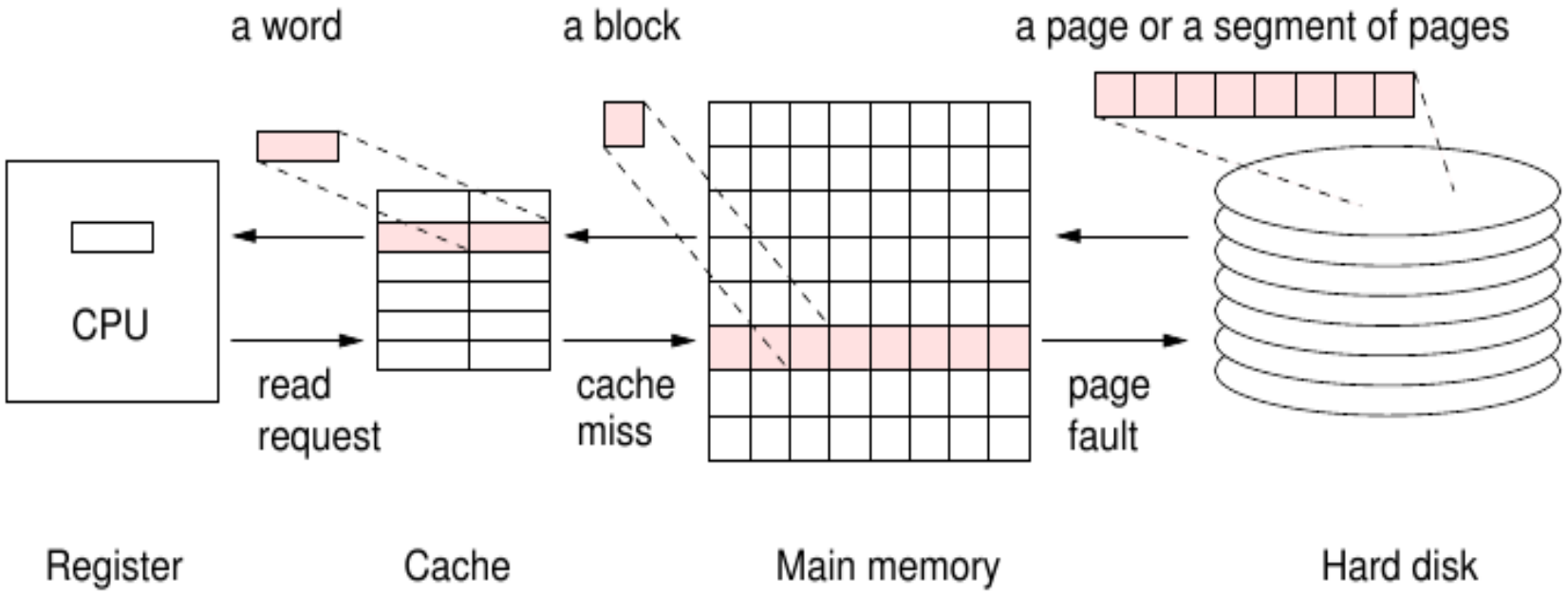
- Shortening response time is difficult
- Even page cache access is relatively slow

In services user expectation matters

- **Amazon reported that** *every 100ms delay costs 1% of sales* *(Greg Linden)*
- **Google increased the number of search results from 10 to 30. Additional 500ms** *caused a 20% drop in traffic. Half a second delay killed user satisfaction.* *(Marissa Mayer in Web 2.0)*
- *Stock Traders Find Speed Pays, in Milliseconds* *(Charles Duhiagg, The New York Times)*
- **Sometimes I/O stalls, or blocking operations are not tolerated**
  - HLR operations in TelCo
  - Stock trading SW
  - Medical devices

Making more throughput is easy, but once you have bad latency you're stuck with it. Stuart Cheshire, May 1996

Data access through memory levels



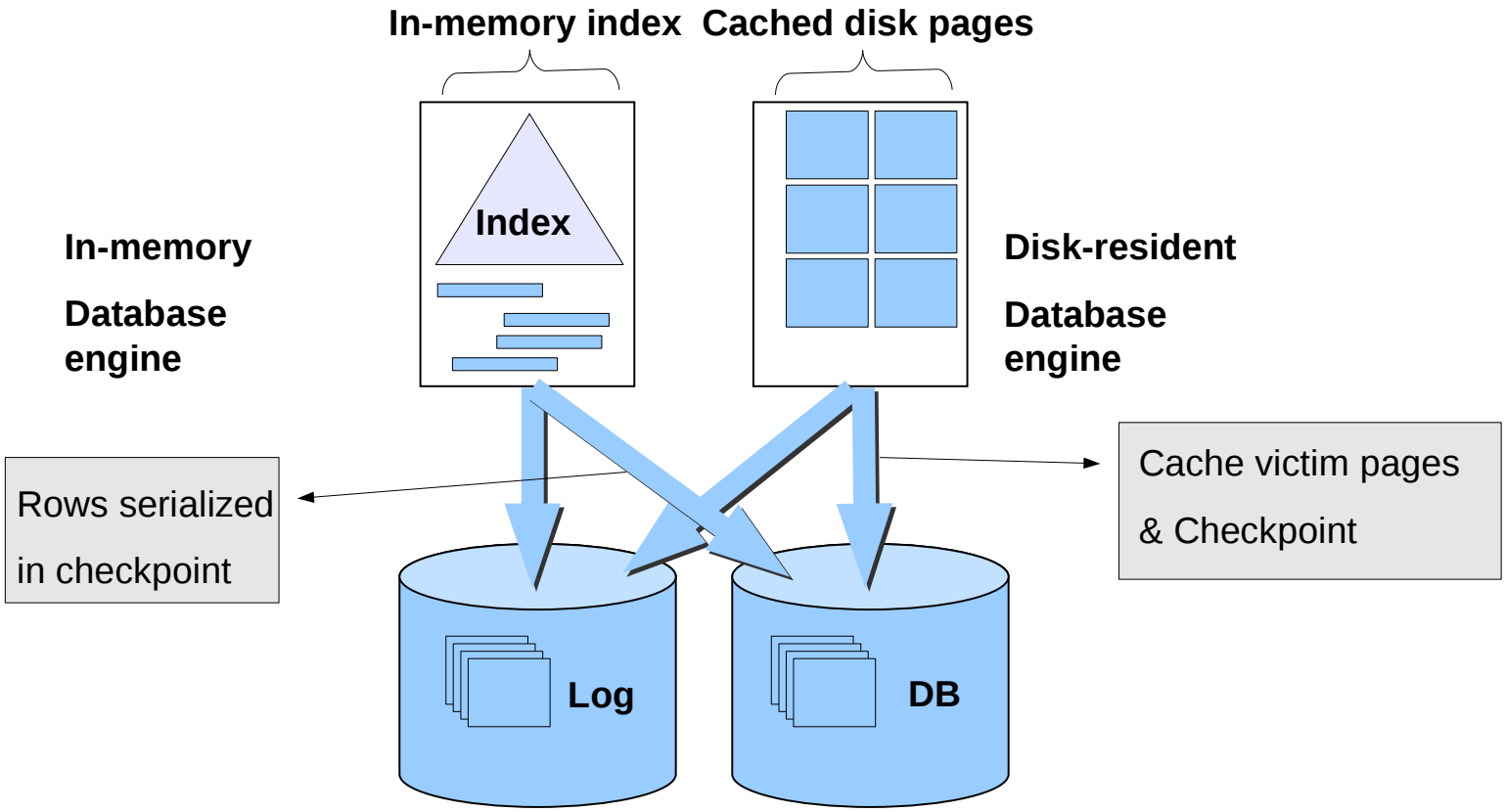
Cost of data access

x1

x15

x40000

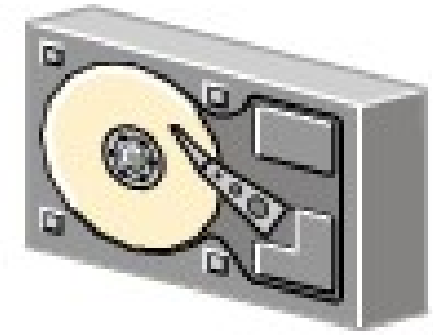
Data layout in a database server



Why in-memory database is faster?

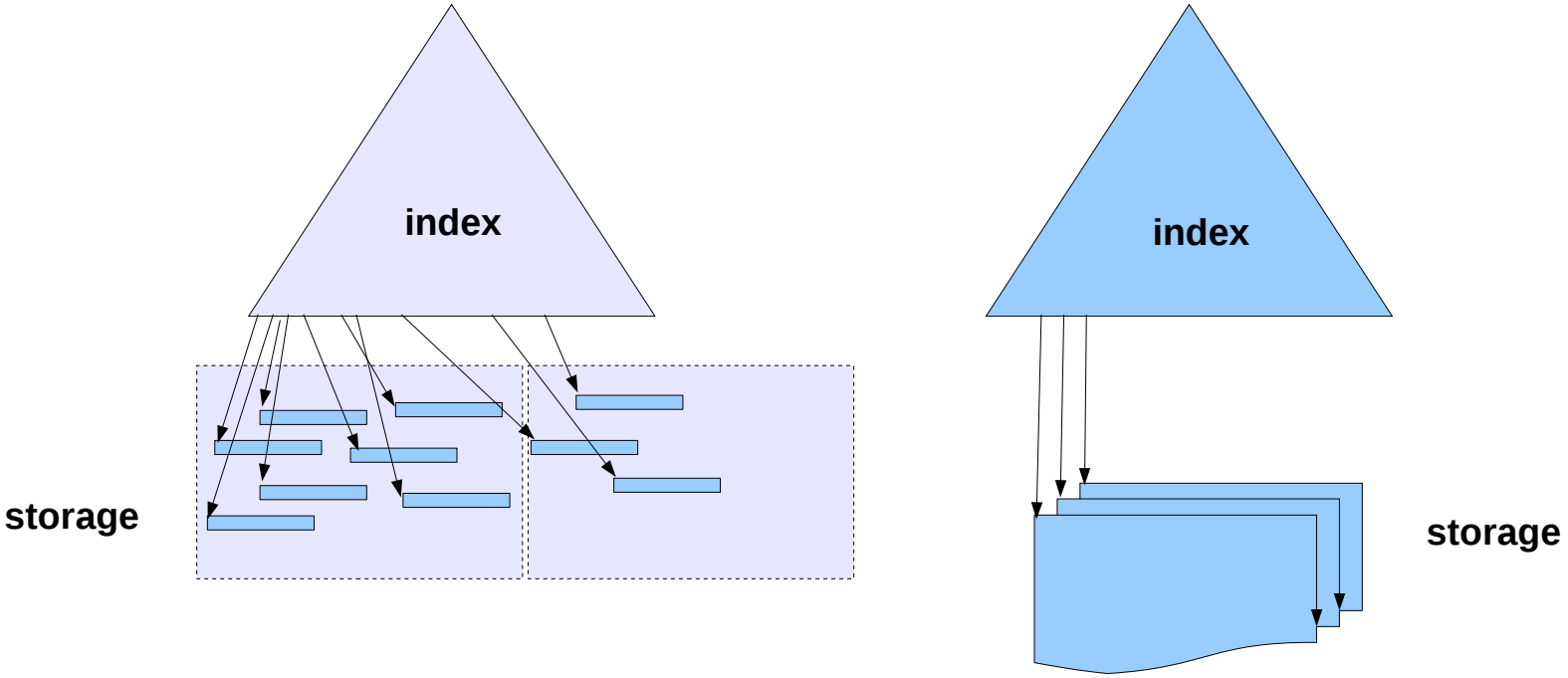
- **Seek time dominates the cost**
  - Reading sequential disk blocks is 1000 times faster than reading random blocks.
- **Database's access pattern is mostly random.**
- **Reading from RAM is roughly 100 times faster than sequential disk read.**

- **Disk I/O involves lot overhead to read, and write operations**
- **Unit of access is block.**
  - Amount of data transferred can be 10-100 times more than what was requested.



Why in-memory database is faster also in memory?

- **In-memory database uses dense index in contrast to disk-resident db.**
- **Pages in disk-resident db may or may not be found from cache.**
  - Also index nodes may be swapped to disk.

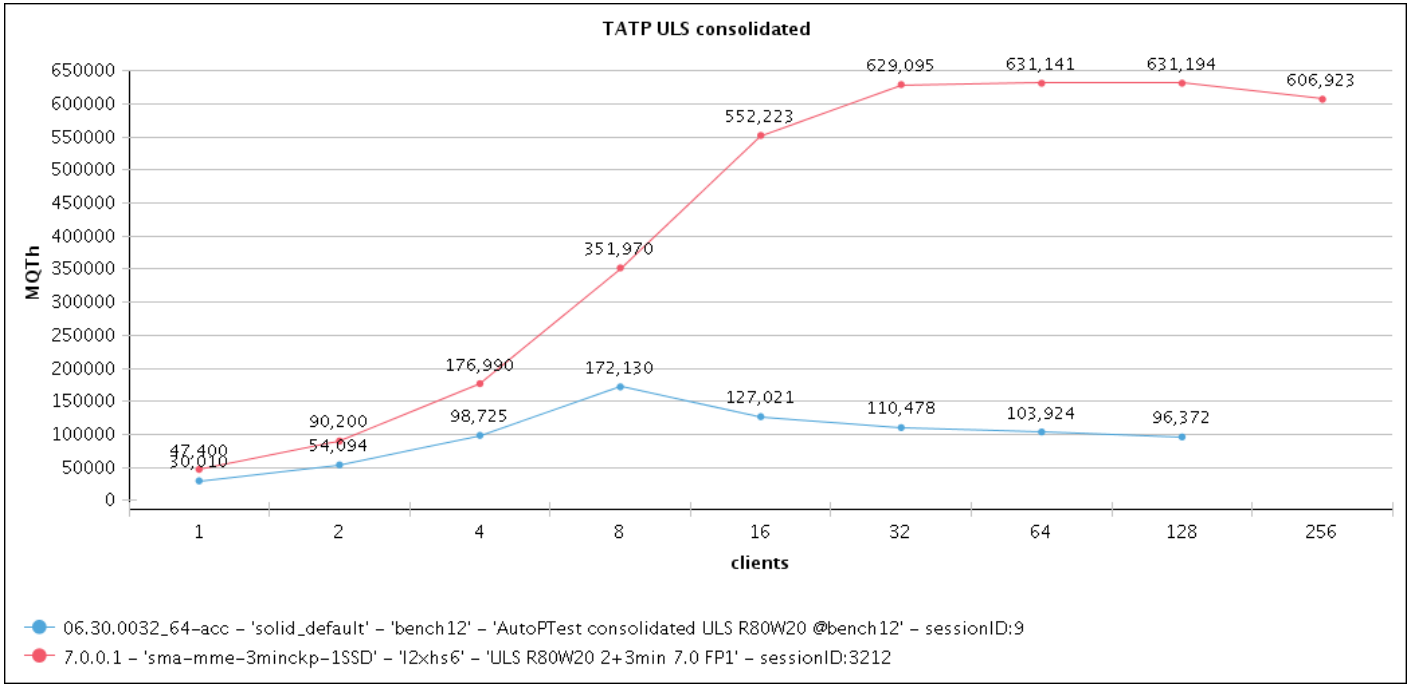


Main challenges in in-memory databases

- Maximize parallel processing in one machine
- Maintain ACID but avoid storage/network bottleneck
- Provide low latencies consistently
- Scale out (generic requirement)

Vertical scaling

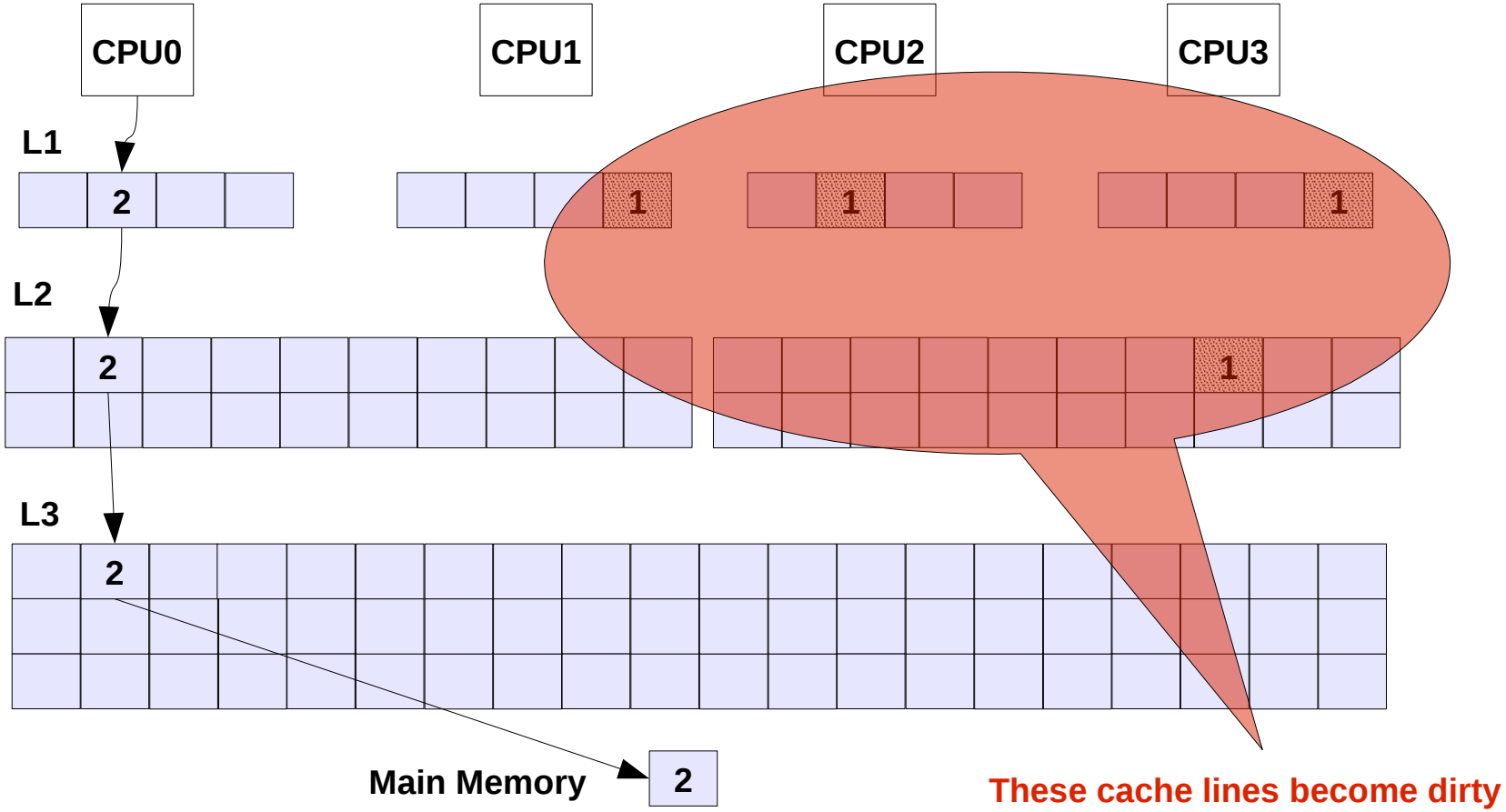
- **Increase throughput in line with the # of users**
  - do not trade latency for throughput
- **Use fine-grained concurrency control where needed**
- **Remember cache, especially false sharing**



TATP Benchmark : *http://tatpbenchmark.sourceforge.net/*

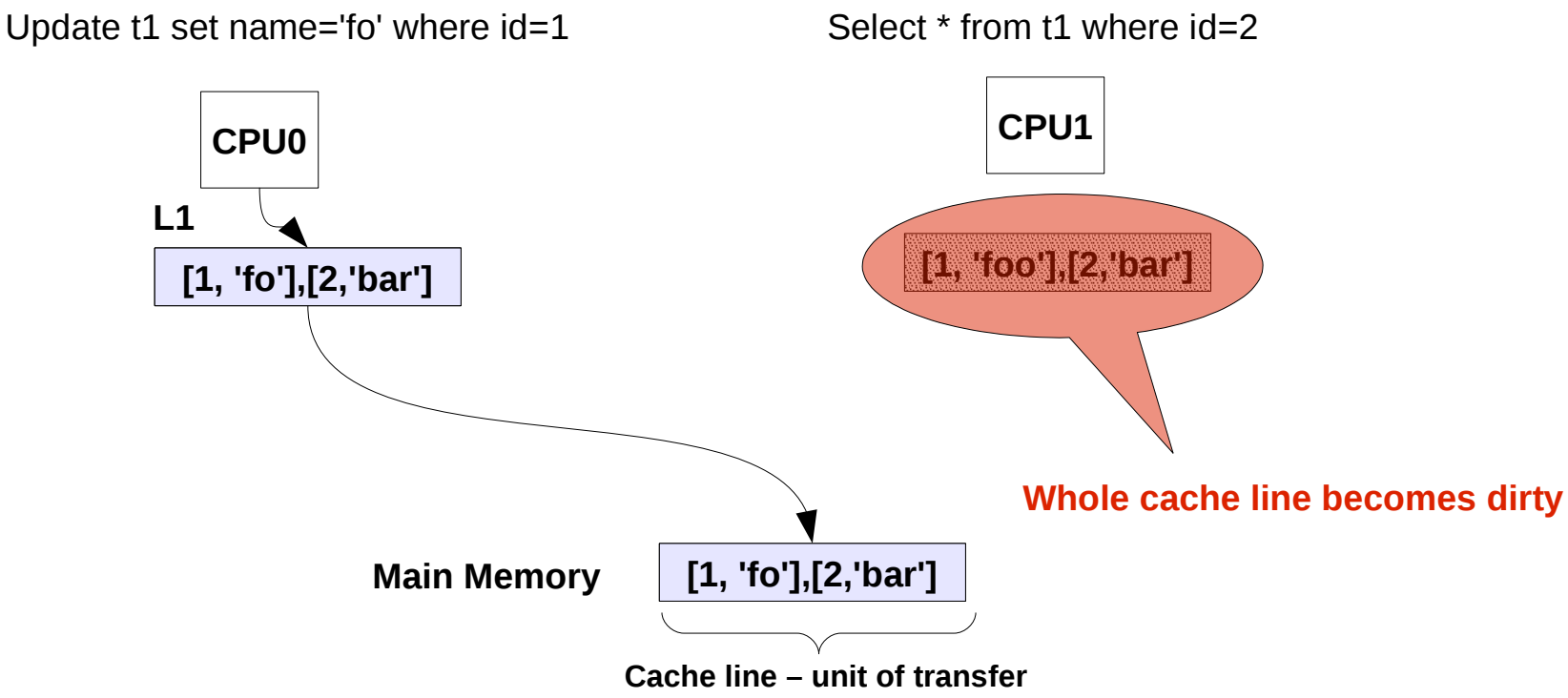
Example : coherency cache miss

- A shared transaction counter for all clients to use



Example : false sharing

- **Independent data share a cache line**
- **Remember spatial locality, avoid false sharing**



Tuning log writing

- **Synchronized log writing (WAL) doesn't leave much to do**
  - Buy faster hardware
- **With asynchronous logging, there is time window to be used**
  - Typically at least 1-5 seconds
  - Clients are provided with several pre-allocated (recycled) log entry buckets
  - Every now and then, a leaving client is requested to allocate/recycle buckets – share the load
  - Client adds filled buckets to a circular buffer
  - Log writer thread flushes every full buffer maintaining transaction sequence number ordering
- **Sequential file write is fast enough, problem is to prevent the swarm of clients from colliding with each other**

Unobtrusive checkpoint writing

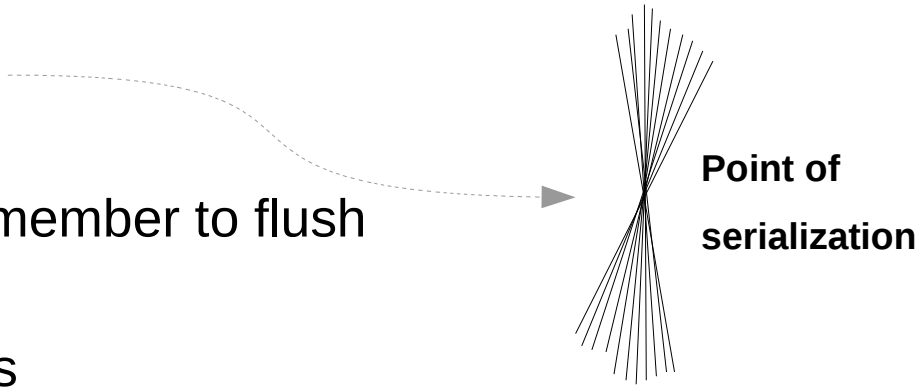
- **Consistent, or fuzzy**
  - Fuzzy spreads write load evenly over time but it's not consistent
  - Consistent doesn't need log for restoring
- **Every checkpoint starts with synchronization point – freeze**
  - Chekpoint counter is increased, and dirty data is marked
- **The rest is sequence of non-blocking operations**
- **Techniques:**
  - Shadow copying
  - versioning
- **Write only dirty pages, write fast, but don't block disk IO**
- **Write everything, or data only**
  - Latter is trading faster checkpoint to slower restore

Low latencies make services possible

- **Database often sits on the bottom of sw stack of the service**
- **Service's response time includes application and transaction processing altogether**
- **Assume that transaction execution shortens to 1/10th – how would you spend the free extra time?**
  - Execute more transactions?
  - Add business logic to application layer?
  - Meet the deadlines?
- **Low latencies enable deplying more intelligent applications and services without violating Service-Level Agreements (SLA).**

Threats of latency

- **Unnecessarily large critical sections in code**
  - N threads accessing M resources & one mutex protecting them all
  - Divide threads, and resources to, say, 10 groups which are protected by 10 mutexes
- **Avoid mutex trashing**
  - Shared counter, for example
- **Disk IO**
  - Avoid synchronous writes, remember to flush
- **Network IO**
  - Avoid synchronous operations
- **Large memory operations**
  - OS tries to buffer everything, and swap arbitrarily
  - Flush large file writes periodically
  - Prevent swappiness if necessary
  - Split operations to resonably-sized ones

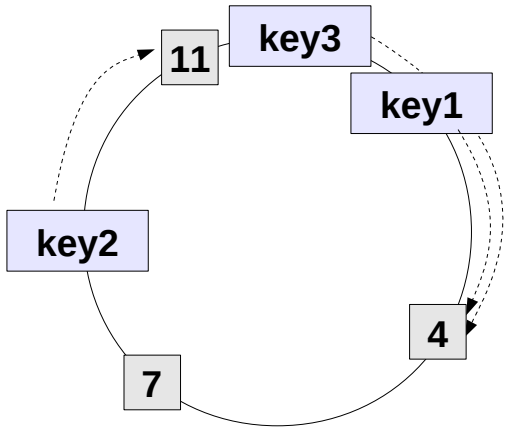


Massive data, and parallelism

- **Why to scale out ?**
  - In some cases, multiple cheap boxes provide more than a few bigger and more expensive servers
  - Adding soap boxes one by one provides flexible, low-risk solution
  - If the solution meets the needs, it is better to ask, why not?
- **How to make tens or hundreds of servers constitute a service?**
  - Distribute requests transparently to nodes where data is stored
  - Make server and network failures invisible to users
  - Make administrative tasks transparent to users
  - Relax consistency (from C) when acceptable
  - Make conflict resolution when data is read
  - Emphasis on local operations
  - Avoid or prevent inter-node operations
    - Avoid range queries in row storage
    - Avoid joins
    - Avoid select * 's in column storages

Distributing load

- **By key : assume three server nodes, each assigned with a value between 0-11**
- **Every key value is gets a position on the ring from the result of key mod 12**
- **Key is stored to nearest server clockwise on the ring**
- **Key1 gets position 2, key2 position 9, and key3 position 12**
- **Load balancing soon becomes an issue**
  - Move servers towards crowded areas
  - Create multiple virtual nodes
- **Partition methods**
  - Vertical, horizontal partitioning of tables
  - Federated model
- **Partition criteria**
  - Range, hash, list, complex rules



Miscellaneous readings

IBM solidDB RedBook http://www.redbooks.ibm.com/abstracts/sg247887.html

Google : ***olfit concurrency-control***, ***cache-conscious database***, ***cache-conscious trie, Dynamo, Cassandra, project Voldemort***

Some interesting new products:

- **SQLFire**
  http://pubs.vmware.com/vfabric5/index.jsp?topic=/com.vmware.vfabric.sqlfire.1.0/getting_started/book_intro.html
- **NuoDB**
  http://www.odbms.org/blog/2011/12/re-thinking-relational-database-technology-interview-with-barry-morris-founder-ceo-nuodb/

Blogs:

- **Daniel Abadi's blog** http://dbmsmusings.blogspot.com/
- **and more about CAP** http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html
- **DBMS2 blogs** http://www.dbms2.com
- **ODBMS** http://www.odbms.org/blog/