THUIRDB :A Large-Scale, Highly-Efficient Index, Fast-Access Key-Value Store

10billion records,1bit index per record,1million/sec throughput in 1 machine

梁斌
清华大学计算机科学与技术系
http://www.thuir.org/thuirdb/
mgigabyte@gmail.com

Structure of Report

- Introduction
  - Requirement and motivation
  - Background review
- Problems
  - Problems for binary-search
  - Problems for B+-tree
  - Problems for hash
  - Problems for building database
- Rational
  - Separating key and value by location
  - Sorting then linear write
  - Data lay-out
  - Building the index bottom-up
  - Highly-efficiency compressing

Requirement and motivation

- A special requirement for key-value store：

（1）Large scale, billions of record

（2）Random query, hundreds of random query for a task

（3）Static dataset, once built never changed, just read no update

（4） Low cost, sometimes should be built in a machine

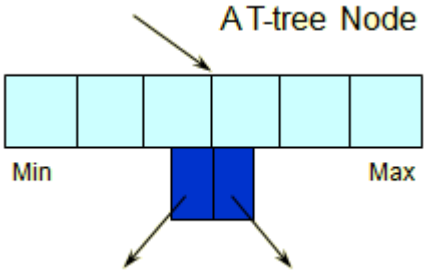
- Pratical application situation：

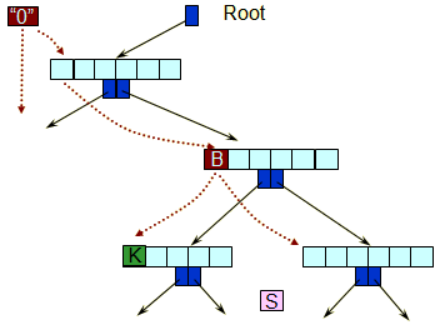
（1）log-based analysis systems

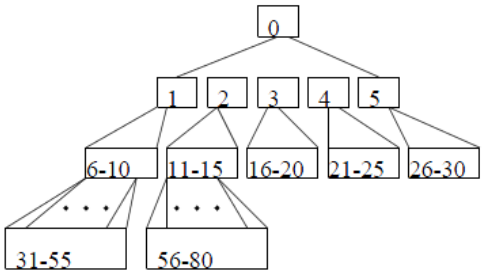
（2）language-model based machine translation

Backgroud Review

- [Tobin 1986] issue an new data structure:**T-tree** . T-tree inherits from both B-tree and AVL-tree and is used by main-memory databases, such as Datablitz, eXtremeDB, MySQL Cluster.
- [Philip 2001]pk-Ttree and pkB-tree add index compression.
- [1998 Rao Jun]Rao created CSS-tree, and developed to CSB+ tree and CSS+tree

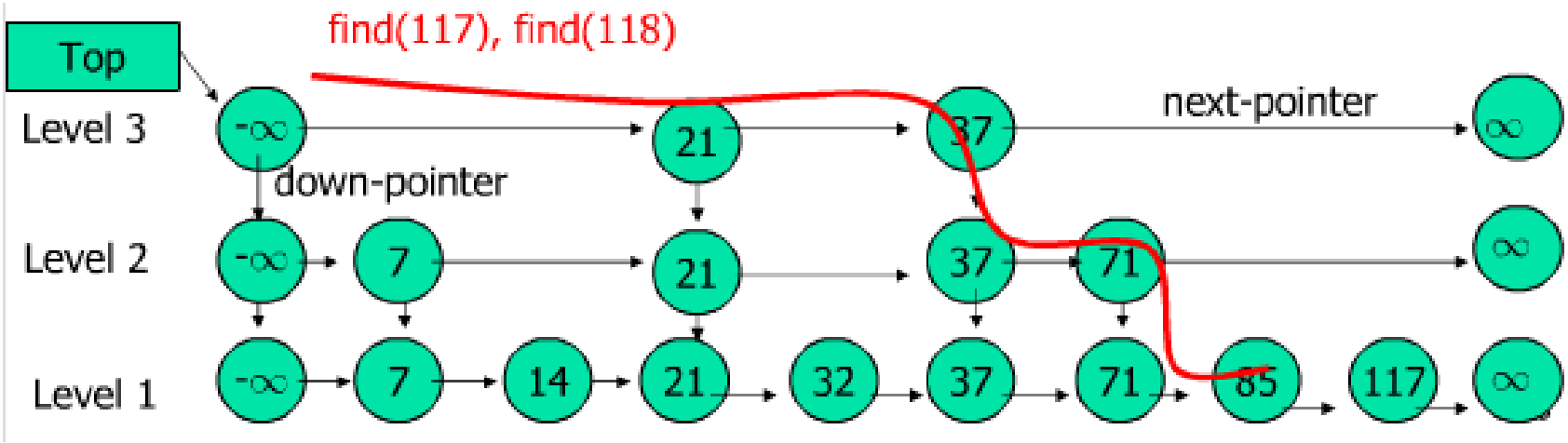






Background Review

Google introduced LMS-tree and multi-level skip list based key-value store called LevelDB in 2011.



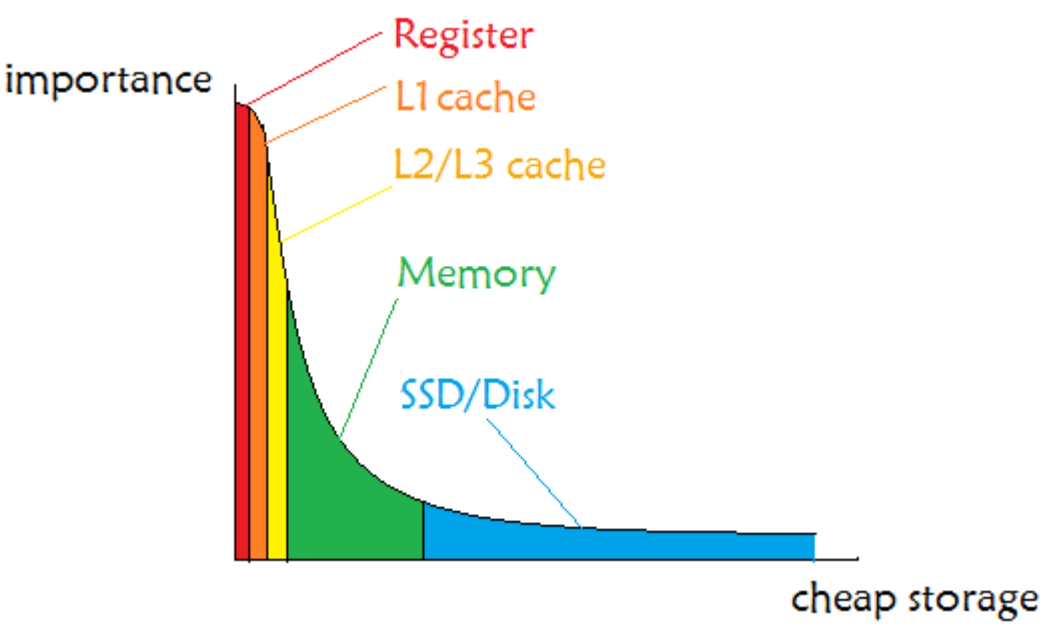
From Alon Efrat's ppt of skip list

http://blog.nosqlfan.com/html/3041.html

Problems for binary-search

- Importance of data is out of control
  - 9 is the key and index key, very important
  - 8 and 10 is only key, not important, but have been put into L1 cache with 9 unfortunately
- Poor Data reference locality

1
21



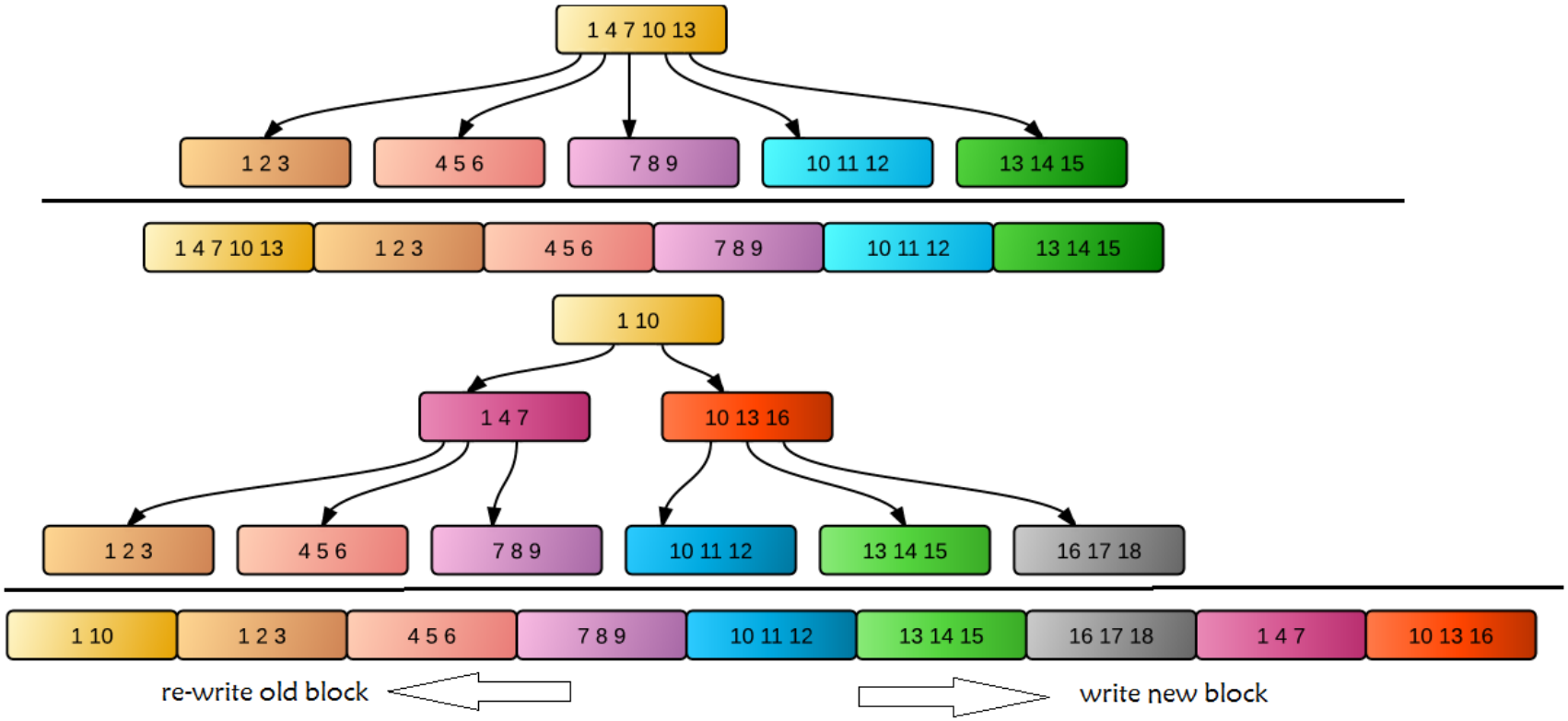
15
109

Problems for B+-tree

(1)Each node stores child pointers and Each node 50% full.

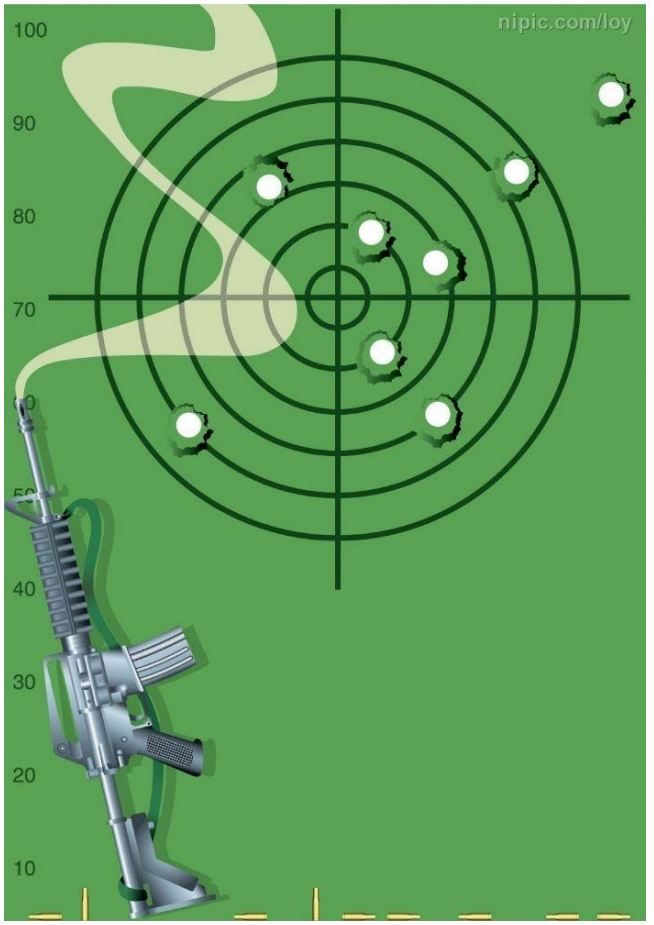
(2)Data write is low-efficiency.

(3)Data write is not sequential



Problems for Hash

- High space overhead
- Hard to compress
- Can’t support ordered access
- Cant’s support range-query



Problems for building database

- Both B+-tree-based and Hash-based databases are in trouble with memory-shortage, why?
- Why them need so much memory in hand? Why not sync these dirty data to disk from buffer?

Rational of THUIR-DB

- (1) separating key and value by location
- (2) sorting the key-value pairs, linear write them
- (3) build the index bottom-up
- (4) pointer eliminating
- (5)highly-efficiency compressing

Separation of index and data

- **Insert:**

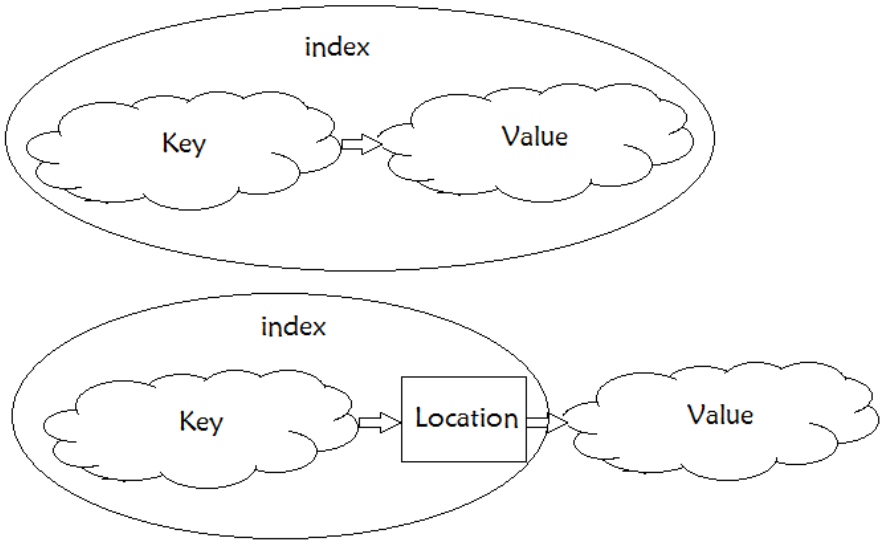
  location = **store_s**.*put*(value)

  search_s.*insert*(key,location)

- **query:**

  location = **search_s**.*find*(key)

  value = **store_s**.*get*(location)



- Separating index and data makes the index smaller, provides chances to apply different compression methods and some other tricks, such as data reordering.

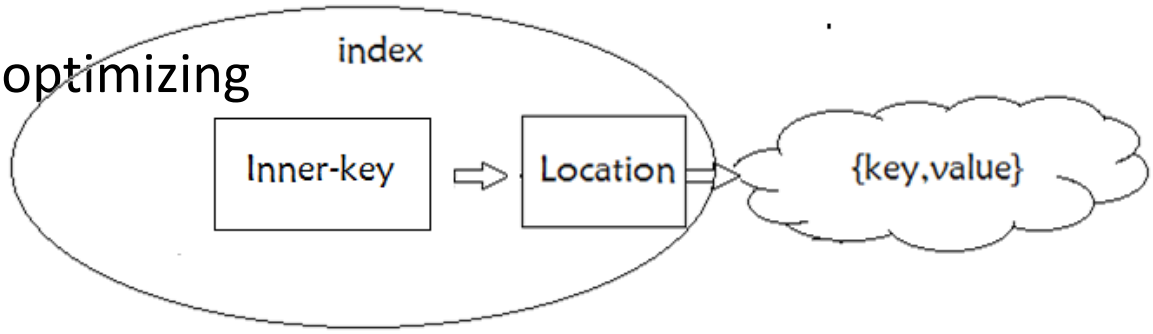
- In a word, Indirection provides flexibility

Variable-length key and value

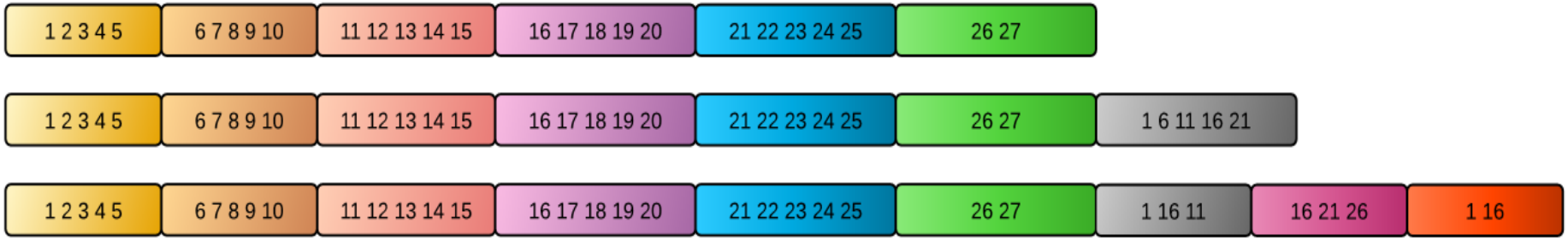
- insert：
  location = **store_s**.*put*(key,value);
  inner_key = md5(key)
  search_s.*insert*(inner_key,location);

- query：
  inner_key = md5(selected_key)
  locations = **search_s**.*find*(inner_key)
  array(<key,value>)=**store_s**.*get*(locations)
  for each item in arry
  if(keyi = selected_key)
  return valuei

- Certainty is the base of optimizing



linear inserting, building index bottom-up



advantage：

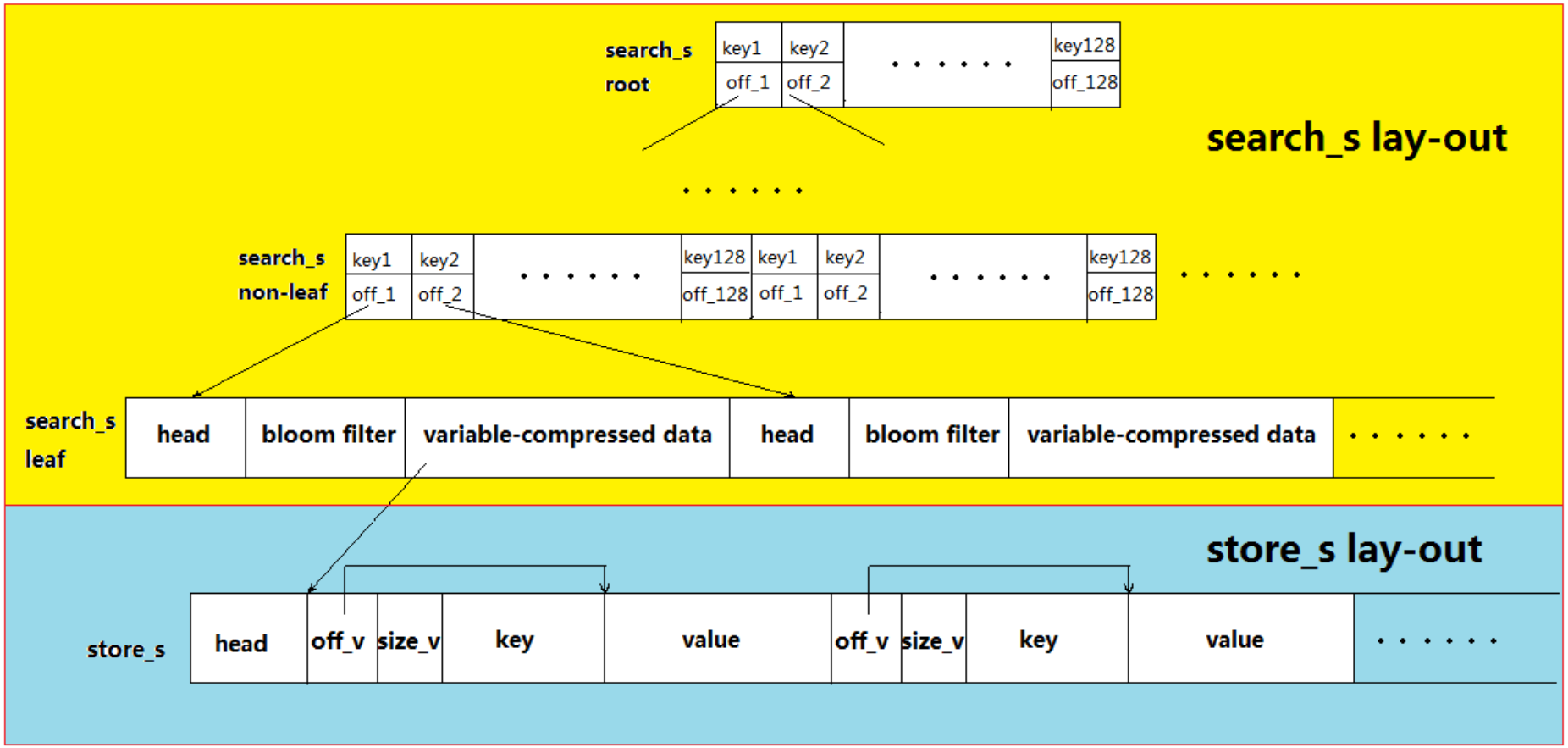
（1）important data constructed together(**cache conscious**)

（2）write data sequentially

（3）full-write each node

（4） set the stage for compression

Lay-out


search_s
root
key1
key2
key128
off_1
off_2
off_128
search_s lay-out
search_s
non-leaf
key1
key2
key128
off_1
off_2
off_128
search_s
leaf
head
bloom filter
variable-compressed data
store_s lay-out
store_s
head
off_v
size_v
key
value

Compression of Integer Sequence

- 1）if we known value-range in advance

  a give integer sequence S， all integer in the range [0， L],then it cost $S*\lceil \log_2 L \rceil$ bits to save them all

- 2）if we know it is ordered .

  use difference to make the smaller, such as

  1,4,6,15,25,40, after differencing, we get 1,3,2,9,10,15， if all integer in the new sequence is in the range [0,L’],then it cost $S*\lceil \log_2 L' \rceil$ bits.

- 3）sometimes there are very large number, such as 1,4,6,15,25,10000,100001, use exception block to save them, Pfordelta and New-Pfordelta solve the problem very well.

Adaptive-compression

- If we have the integer sequence like these.

  1 1 2 3 15 16 16 17 19 20 21 22…

- Try to find a economical way to compress

  1 1 2 3 [1bit gap] -> X bit /record

  1 1 2 3 15 16 17.. [4bit gap]-> Y bit/record

  If X<Y then

  [1 1 2 3] as a block to compress

question

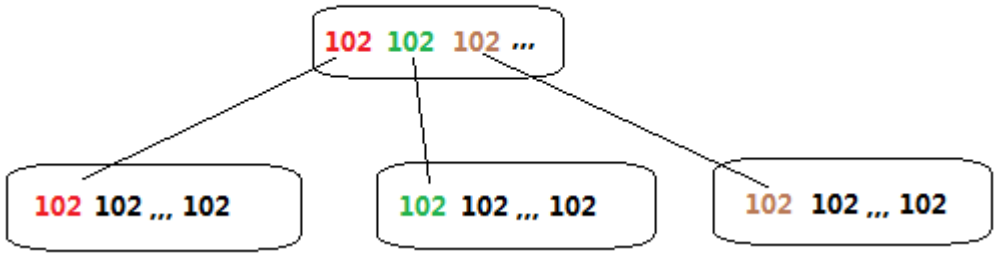
- 1 1 2 3 3 4 5 6 7 8 9 9… 100 | 101 | 102 | 102 | 102

  A B C D

  how the chose the segment point？

  let’s suppose that the max block have 256 sets， and chose A ,B,C or D is ok for the limit

  if we have 2000 keys that all are the 102？ （more than 256）



Apply SIMD to speed-up compression

- #define **dopack1**(x,y,z,o,p) "pxor %%xmm1,%%xmm1\npxor %%xmm2,%%xmm2\npxor %%xmm3,%%xmm3\npxor %%xmm4,%%xmm4\nmovhpd "#x"(%1),%%xmm1\npsllq $32,%%xmm1\nmovhpd "#y"(%1),%%xmm2\npsllq $32,%%xmm2\npsrlq $32,%%xmm2\nORPD %%xmm1,%%xmm2\nmovss "#z"(%1),%%xmm3\npsllq $32,%%xmm3\nmovss "#o"(%1),%%xmm4\nORPD %%xmm3,%%xmm4\npslld $"#p",%%xmm15\nORPD %%xmm2,%%xmm15\nORPD %%xmm4,%%xmm15\n"

  #define **pack1(x)** dopack1(x*4,128+x*4,256+x*4,384+x*4,1)

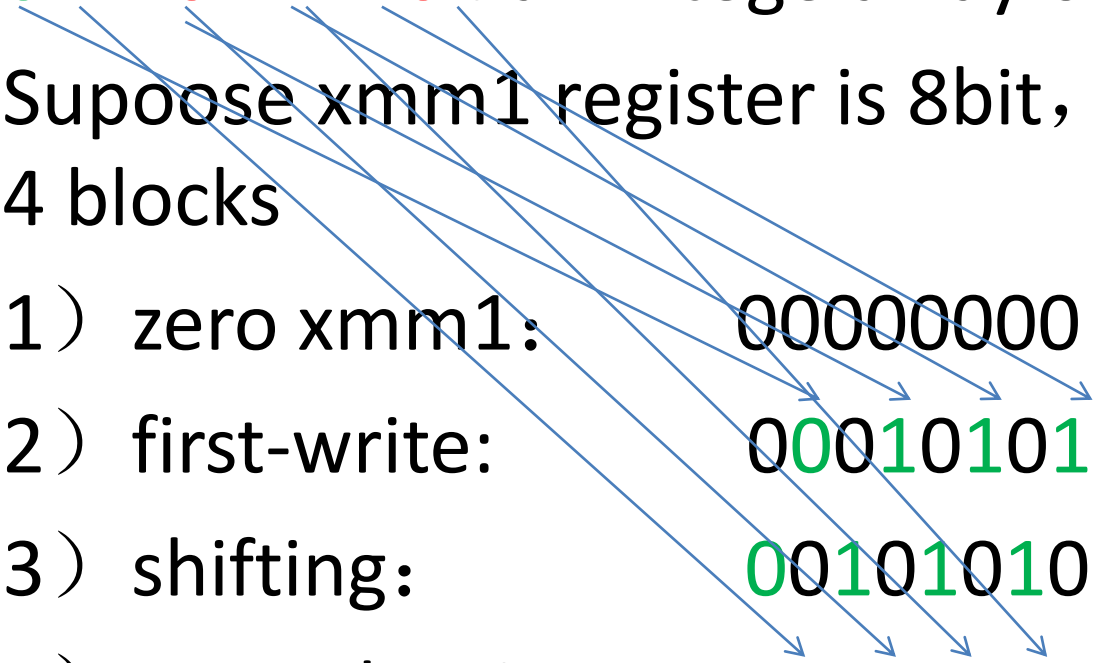
```
void PACK1(uint32_t* code,uint32_t* data,size_t n){
    unsigned char* des = (unsigned char*)code;
    unsigned char* src = (unsigned char*)data;
    __asm__ __volatile__(
        "pxor %%xmm15,%%xmm15\n"
        pack1(0)
        pack1(1)
        ...
        pack1(31)
        "movdqu  %%xmm15 ,(%0)\n"
        ::"r"(des),"r"(src):"memory");
     __asm__ __volatile__ (
        " sfence \n "
        ::
    );
};
```

unfold

- pxor %%xmm1,%%xmm1\n
- pxor %%xmm2,%%xmm2\n
- pxor %%xmm3,%%xmm3\n
- pxor %%xmm4,%%xmm4\n
- movhpd "#x"(%1),%%xmm1\n
- psllq $32,%%xmm1\n
- movhpd "#y"(%1),%%xmm2\n
- psllq $32,%%xmm2\n
- psrlq $32,%%xmm2\n
- ORPD %%xmm1,%%xmm2\n
- movss "#z"(%1),%%xmm3\n
- psllq $32,%%xmm3\n
- movss "#o"(%1),%%xmm4\n
- ORPD %%xmm3,%%xmm4\n
- pslld $"#p",%%xmm15\n
- ORPD   %%xmm2,%%xmm15\n
- ORPD   %%xmm4,%%xmm15\n

explanation

- 01 10 11 10 : an intege array of 8 elements.
- Supoose xmm1 register is 8bit， think of it as 4 blocks
- 1） zero xmm1： 00000000
- 2） first-write: 00010101
- 3） shifting： 00101010
- 4） second-write: 01101110



Analysis

write once for a block

```
pxor %%xmm1,%%xmm1\n
pxor %%xmm2,%%xmm2\n
pxor %%xmm3,%%xmm3\n
pxor %%xmm4,%%xmm4\n
movhpd "#x"(%1),%%xmm1\n
psllq $32,%%xmm1\n
movhpd "#y"(%1),%%xmm2\n
psllq  $32,%%xmm2\n
psrlq  $32,%%xmm2\n
ORPD %%xmm1,%%xmm2\n
movss "#z"(%1),%%xmm3\n
psllq  $32,%%xmm3\n
movss "#o"(%1),%%xmm4\n
ORPD  %%xmm3,%%xmm4\n
pslld $"#p",%%xmm15\n
ORPD   %%xmm2,%%xmm15\n
ORPD   %%xmm4,%%xmm15\n

. . . . . .

"movdqu   %%xmm15 ,(%0)\n"
```

write 128 times for a block

```
code[ i]  = 0;
code[ i]  |= data[ j+0]<<0;
code[ i]  |= data[ j+1]<<1;
code[ i]  |= data[ j+2]<<2;
code[ i]  |= data[ j+3]<<3;

. . . . . .

code[ i]  |= data[ j+26]<<26;
code[ i]  |= data[ j+27]<<27;
code[ i]  |= data[ j+28]<<28;
code[ i]  |= data[ j+29]<<29;
code[ i]  |= data[ j+30]<<30;
code[ i]  |= data[ j+31]<<31;
```

Result of applying SIMD

- Experiments for comperssing 1.28billion integers.

  NO SIMD: 9.02second

  SIMD:1.14second

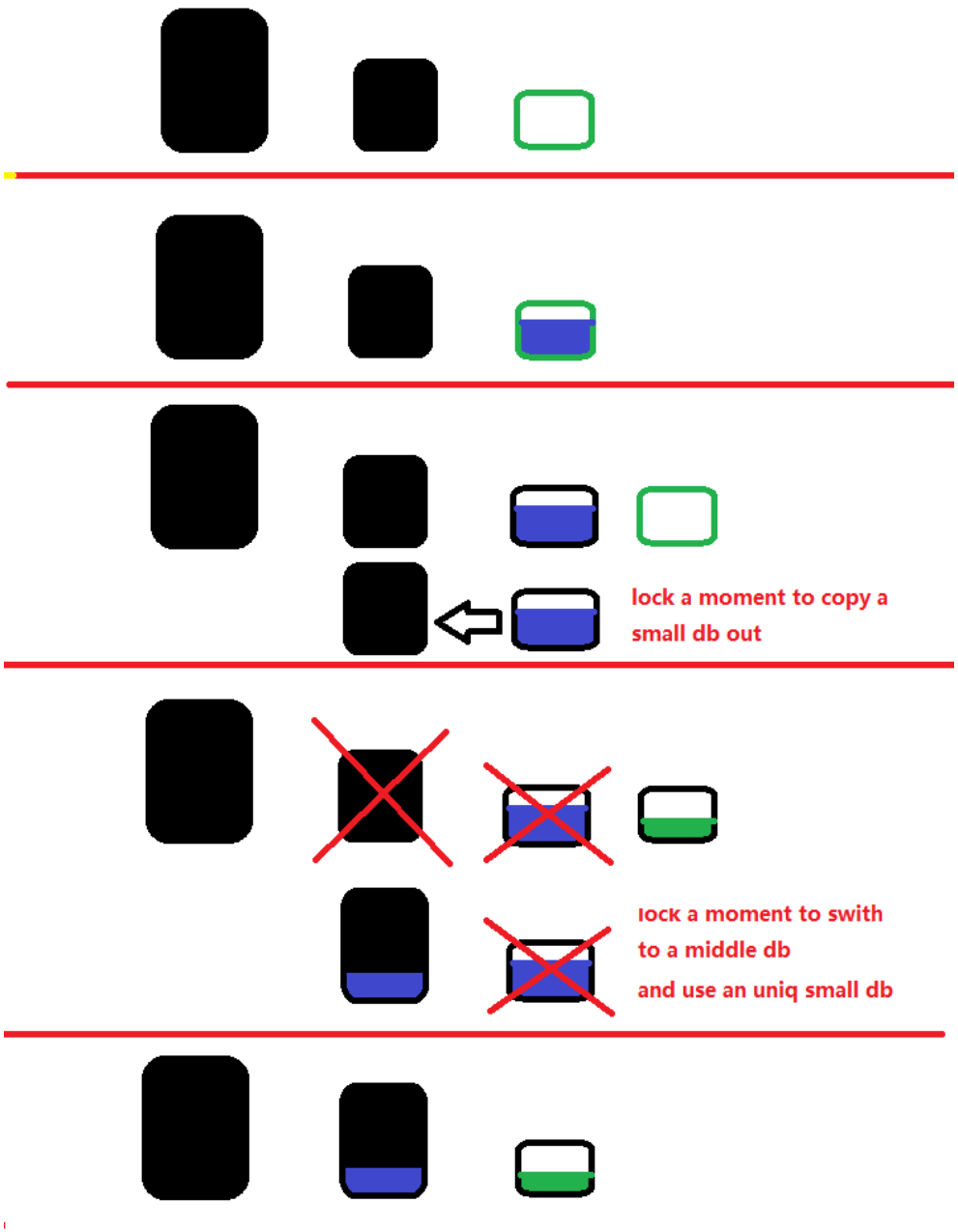
  improvement **6.91** times。

Flynn's taxonomy

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	**SIMD**	MIMD

In application, intel issured SSE instructions set and AMD put forward 3DNow!。AMD declared to support SSE instructions set soon。

Future work
How to insert?



Reference(1)

- [Tobin 1986]A Study of Index Structures for Main Memory Database Management Systems
- [Philip 2001]Main-Memory Index Structures with Fixed-Size partial Keys
- A.Anand.C.Muthukrishnan, S.Kappes, A.Akelaa, and S.Nath. 2010.Cheap and large CAMs for high performance data-intensive networked system.In Proceedings of the 7th USENIX conference on Networked system design and implementation, 29-29.
- Adam Pauls and Dan Klein. 2011.Faster and smaller n-gram language models. In Proceedings of 49th ACL, 258-267,Portland,Oregon.
- Alexy Khrabrov and George Cybenko. 2010. Discovering Influence in Communication Networks using Dynamic Graph Analysis. In Proceedings of SocialComm 2010, 288-294.
- An Oracle White Paper 2011 oracle NoSQL Database.
- B. Debnath, S.Sengupta, and J.Li. 2011. SkimpyStash:RAM space skimpy key-value store on flash-based storage. In Proc. In Proceedings of International Conference on Management of Data, ACM SIGMOD'11,25-36,

Reference(2)

- Biplob Debnath, Sudipta Sengupta, Jin Li . 2010. FlashStore: High Throughput Persistent Key-Value Store . Journal: In Proceedings of The Vldb Endowment - PVLDB , vol. 3, no. 2, 1414-1425
- D.Beaver, S.Kumar, H.C.Li,J.Sobel, and P.Vajgel. 2010.Finding a needle in Haystack:Facebook's photo storage. In Proceedings of 9th USENIX OSDI OCT.1-8
- D. K. Blandford and G. E. Blelloch. 2008.Compact dictionaries for variable-length keys and data with applications. ACM Trans.Alg. 4,2,17:1-17:25.
- Fotakis, D., Pagh, R., Sanders, P., and Spirakis, P. G. 2005. Space efficient hash tables with worst case constant access time. Theory of Computing Systems 38,2, 229-248.
- G.Decandia, D.Hashtorun, M.Jampani, G.Kakulapati, A. Lakshman, A.Pilchin, S. Sivasubramanian, P.Vosshall, and W.Vogels. 2007.Dynamo: Amazon's highly available key-value store. In Proc.21st ACM Symposium on Operating System Principles(SOSP), 205-220.
- Hyeontaek Lim, Bin Fan, David G.Andersen, Michael Kaminsky. 2011.SILT:A Memory-Efficient, High-Performance Key-Value Store, In Proceedings of the 23rd ACM symposium on Operating System Principles,1-13.
- Jun Rao and Kenneth A. Ross. 1999.Cache conscious indexing for decision-support in main memory. In Proceedings of the 25th VLDB Conference.78-89.

Reference(3)

- KNUTH, D. E 1973. The Art of Computer Programming/Sorting and Searching, vol.3.Addison Wesley.
- Patrick O'Neil,Edward Cheng Dieter Gawlick, Elizabeth O'Neil. 1996.The log-structured merge-tree (LSM-tree). Acta Informatica, 33,4,351-385.
- S.Nath and A.kansal. 2007. FlashDB:Dynamic Self-tuning Database for NAND Flash. In Proceedings of 6th IPSN, 410-419.
- Torp, Kristian and Mark, Leo and Jensen, Christian S. 1998. Efficient differential timeslice computation. IEEE Transactions on knowledge and data engineering 10,4. 599-611.
- Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing.858-867.
- Trishul M.Chilimbi James R.Larus Mark D.Hill. Improving pointer-based codes through cache-conscious data placement. Technical report 98, University Wisconsin-Madison, Computer Science Department, University of Wisconsin-Madison Madison, Wisconsin 53706, 1998.

Question??

My son, 11-month old