system project description: Friday
system project description: Friday
system project description: Friday

core storage engine embedded kv-store
~ third milestone of cs165 project
> complex than 1 data structure
system project description: Friday

core storage engine  embedded kv-store
~ third milestone of cs165 project
> complex than 1 data structure

input/output:
kv model (tests included)
design requirement:

basic LSM + monkey/dostoevsky

three design optimizations
design requirement:
- basic LSM + monkey/dostoevsky
- three design optimizations

final report: experimental analysis
- presentation and demo/code review
design requirement:
- basic LSM + monkey/dostoevsky
- three design optimizations

final report: experimental analysis
presentation and demo/code review

implementation in C/C++

performance goals:
- minimize cache misses
- maximize IPC
Teaching Fellows:

Off class discussions are key! question on readings, ideas, help with code/analysis
Teaching Fellows:

subarna  brian  siqiang  wilson  neil  varun
Teaching Fellows:

- Subarna
- Brian
- Siqiang
- Wilson
- Neil
- Varun

6 slots of TF OH in week, +3 slots by Stratos, +2 remote slots in the weekend
Teaching Fellows:

subarna  brian  sijiang  wilson  neil  varun

6 slots of TF OH in week, +3 slots by Stratos, +2 remote slots in the weekend

OH discussions are key! (readings, ideas, help with code/analysis)
class participation?

class + OH (w Stratos)
20% of grade
as of 2nd half of semester
How many and which structures are possible?

Can we compute performance w/o coding?
1,2,3… 12,15,17 20,…

page size: 64K - holds 16K 4 byte ints
N elements, P pages
sorted pages
info to navigate
lower level
value-pointer

page size: 64K - holds 16K 4 byte ints
N elements, P pages
sorted pages
1, 2, 3…

12, 20

12, 15, 17

20,…

<12  >=12

info to navigate lower level value-pointer

page size: 64K - holds 16K 4 byte ints
N elements, P pages
sorted pages

4+4 bytes for each page (value+pointer)
64K/8= index 8K pages
page size: 64K - holds 16K 4 byte ints
N elements, P pages
sorted pages

info to navigate lower level
value-pointer

<12  >=12

12,20  35,...  50,...

1,2,3...
12,15,17  20,...

>=12  <12

4+4 bytes for each page (value+pointer)
64K/8 = index 8K pages

can index 8K pages of the next level
The diagram shows a tree structure with the following labels:

- **Root**: 35, 50
- **Internal Nodes**: 12, 20, 35, ..., 50, ...
- **Leaves**: 1, 2, 3..., 12, 15, 17, 20, ...

The tree has a fanout pattern, with the root node splitting into three branches, and the leaves representing the smallest values.
The diagram illustrates a tree structure with the root node labeled as 35, 50. Below the root are internal nodes labeled as 12, 20, 35, 50, and so on. The leaves of the tree are labeled as 1, 2, 3, ..., 12, 15, 17, 20, ... The height of the tree is given by $\log_{\text{fanout}} P$. The internal nodes are referred to as internal nodes, and the leaves are referred to as leaves.
The diagram illustrates a tree structure with the following key points:

- **Root Node:** 35, 50
- **Internal Nodes:** 12, 20, 35, ..., 50, ...
- **Leaves:** 1, 2, 3, ..., 12, 15, 17, 20, ...
- **Height:** $\log_{\text{fanout}} P$
- **Random Accesses:**

The diagram shows the relationship between the height of the tree and the fanout, which affects random accesses. The fanout is the number of children nodes a node can have, and the height is the number of levels in the tree.
leaves

1,2,3... 12,15,17 20,...

35,50

<12 >=12

15

get 15
get 15-55
how do we search the leaves?

a) sorted and b) unsorted leaves

get 15
get 15-55
LSM-trees

- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- internal k-v layout
LSM-trees

B-trees

Logs

Arrays

Bitmaps
key retention
value retention
partitioning
sub-block links
fanout
<table>
<thead>
<tr>
<th>Primitive</th>
<th>Domain</th>
<th>size</th>
<th>Hash Table</th>
<th>B+Tree/CSB+Tree/FAST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key retention. <strong>No</strong>: node contains no real key data, e.g., intermediate nodes of b-trees and linked lists. <strong>Yes</strong>: contains complete key data, e.g., nodes of b-trees, and arrays. <strong>Function</strong>: contains only a subset of the key, i.e., as in tries.</td>
<td>yes</td>
<td>no</td>
<td>function(func)</td>
<td>3</td>
</tr>
<tr>
<td>Value retention. <strong>No</strong>: node contains no real value data, e.g., intermediate nodes of b-trees, and linked lists. <strong>Yes</strong>: contains complete value data, e.g., nodes of b-trees, and arrays. <strong>Function</strong>: contains only a subset of the values.</td>
<td>yes</td>
<td>no</td>
<td>function(func)</td>
<td>3</td>
</tr>
<tr>
<td>Key order. Determines the order of keys in a node or the order of fences if real keys are not retained.</td>
<td>none</td>
<td>sorted</td>
<td>k-ary (k: int)</td>
<td>12</td>
</tr>
<tr>
<td>Key-value layout. Determines the physical layout of key-value pairs. <strong>Rules</strong>: requires key retention != no or value retention != no.</td>
<td>row-wise</td>
<td>columnar</td>
<td>col-row-groups(size: int)</td>
<td>12</td>
</tr>
<tr>
<td>Intra-node access. Determines how sub-blocks (one or more keys of this node) can be addressed and retrieved within a node, e.g., with direct links, a link only to the first or last block, etc.</td>
<td>direct</td>
<td>head_link</td>
<td>tail_link</td>
<td>function(func)</td>
</tr>
<tr>
<td>Utilization. Utilization constraints in regards to capacity. For example, &gt;= 50% denotes that utilization has to be greater than or equal to half the capacity.</td>
<td>(X%)</td>
<td>function(func)</td>
<td>none (we currently only consider X=50)</td>
<td>3</td>
</tr>
<tr>
<td>Bloom filters. A node's sub-block can be filtered using bloom filters. Bloom filters get as parameters the number of hash functions and number of bits.</td>
<td>off</td>
<td>on(num_hashes: int, num_bits: int) (up to 10 num_hashes considered)</td>
<td>1001</td>
<td>H</td>
</tr>
<tr>
<td>Zone map filters. A node's sub-block can be filtered using zone maps, e.g., they can filter based on mix/max keys in each sub-block.</td>
<td>min</td>
<td>max</td>
<td>both</td>
<td>exact</td>
</tr>
<tr>
<td>Filters memory layout. Filters are stored contiguously in a single area of the node or scattered across the sub-blocks. <strong>Rules</strong>: requires bloom filter != off or zone map filters != off.</td>
<td>consolidate</td>
<td>scatter</td>
<td>2</td>
<td>H</td>
</tr>
<tr>
<td>Fanout/Radix. Fanout of current node in terms of sub-blocks. This can either be unlimited (i.e., no restriction on the number of sub-blocks), fixed to a number, decided by a function or the node is terminal and thus has a fixed capacity.</td>
<td>fixed(value: int)</td>
<td>function(func)</td>
<td>hited</td>
<td>terminal(cap: int) (up to 10 different capacities and up to 10 fixed fanout values are considered)</td>
</tr>
<tr>
<td>Category</td>
<td>Definition</td>
<td>Options</td>
<td>Constraints</td>
<td>Children layout</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>Whether and how sub-blocks are connected.</td>
<td>next</td>
<td>previous</td>
<td>both</td>
</tr>
<tr>
<td>13</td>
<td><strong>Immediate node links.</strong> Whether and how sub-blocks are connected.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Each sub-block can be connected to another sub-block (not only the next or previous) with skip-links. They can be perfect, randomized or custom.</td>
<td>perfect</td>
<td>randomized(prob: double)</td>
<td>function(func)</td>
</tr>
<tr>
<td>15</td>
<td><strong>Area-links.</strong> Each sub-tree can be connected with another sub-tree at the leaf level through area links. Examples include the linked leaves of a B+Tree.</td>
<td>forward</td>
<td>backward</td>
<td>both</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td><strong>Sub-block physical location.</strong> This represents the physical location of the sub-blocks. Pointed: in heap, Inline: block physically contained in parent. Double-pointed: in heap but with pointers back to the parent.</td>
<td>inline</td>
<td>pointed</td>
<td>double-pointed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td><strong>Sub-block physical layout.</strong> This represents the physical layout of sub-blocks. Scatter: random placement in memory. BFS: laid out in a breadth-first layout. BFS layer list: hierarchical level nesting of BFS layouts.</td>
<td>BFS</td>
<td>BFS layer(level-grouping: nt)</td>
<td>scatter (up to 3 different values for layer-grouping are considered)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td><strong>Sub-block homogeneous.</strong> Set to true if all sub-blocks are of the same type.</td>
<td>boolean</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td><strong>Sub-block instantiation.</strong> If it is set to eager, all sub-blocks are initialized, otherwise they are initialized only when data are available (lazy).</td>
<td>lazy</td>
<td>eager</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td><strong>Sub-block links layout.</strong> If there exist links, are they all stored in a single array (consolidate) or spread at a per partition level (scatter).</td>
<td>consolidate</td>
<td>scatter</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td><strong>Recursion allowed.</strong> If set to yes, sub-blocks will be subsequently inserted into a node of the same type until a maximum depth (expressed as a function) is reached. Then the terminal node type of this data structure will be used.</td>
<td>yes(func)</td>
<td>no</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Rules:</strong> requires fanout/radix != terminal.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Rules:</strong> requires fanout/radix != terminal.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Rules:</strong> requires fanout/radix != terminal.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Are keys retained? (yes, no, function)
Are values retained?
Utilization? (e.g., >50%)
Are keys retained? (yes, no, function)
Are values retained?
Utilization? (e.g., >50%)

**Fanout** (fixed/functional | unlimited | terminal |)
**Key partitioning** (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))
Are keys retained? (yes, no, function)
Are values retained?
Utilization? (e.g., >50%)

**Fanout** (fixed/functional | unlimited | terminal |)
**Key partitioning** (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))

**Intra node access** (direct | head_link | tail_link | link_function(func))
Are keys retained? (yes, no, function)
Are values retained? (e.g., >50%)

Fanout (fixedfunctional | unlimited | terminal |)
Key partitioning (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))

Intra node access (direct | head_link | tail_link | link_function(func))
Sub block links (next | previous | both | none)
Sub block skip links (perfect | randomized(prob: double) | function(func) | none)
Are keys retained? (yes, no, function)
Are values retained?
Utilization? (e.g., >50%)

Fanout (fixed/functional | unlimited | terminal |)
Key partitioning (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))

Intra node access (direct | head_link | tail_link | link_function(func))
Sub block links (next | previous | both | none)
Sub block skip links (perfect | randomized(prob: double) | function(func) | none)

Zone Maps (min | max | both | exact | off)
Bloom filters (off | on(num_hashes: int, num_bits: int))
Filters layout (consolidate | scatter)
Links layout (consolidate | scatter)
Are keys retained? (yes, no, function)
Are values retained? (e.g., >50%)

Utilization? (e.g., >50%)

Fanout (fixed/functional | unlimited | terminal |)
Key partitioning (none | fw-append | bw-append | sorted | range() | radix() | function (func) | temporal(…))

Intra node access (direct | head_link | tail_link | link_function(func))

Sub block links (next | previous | both | none)
Sub block skip links (perfect | randomized(prob: double) | function(func) | none)

Zone Maps (min | max | both | exact | off)
Bloom filters (off | on(num_hashes: int, num_bits: int))

Filters layout (consolidate | scatter)
Links layout (consolidate | scatter)

Physical location (inline | pointed | double-pointed)
Physical layout (BFS | scatter)
UNORDERED ARRAY

no order (fw-append)

K V [256]

UNORDERED LIST OF ARRAYS

No KV retention
No filters
fw-append
L1 L2 L3

Scattered next links
Variable fanout
Head Addressing
Sub-block location: inline

Sub-blocks homogeneous: true
Sub-block physical layout: scatter
Sub-block consolidation: false
Sub-block instantiation: lazy
UNORDERED ARRAY

no order (fw-append)

B+Tree

Consolidated Filters (Fences only)
Balanced
Fixed fanout
Direct Addressing
Sub-block location: pointed

No KV retention

Recursion: log(n)
Sub-blocks homogeneous: true
Sub-block physical layout: scatter
Sub-block consolidation: false
Sub-block instantiation: lazy
POSSIBLE STRUCTURES

- Trie
- Array
- Skip-List
- Hash-Table
- Linked-List
- Sorted Array

POSSIBLE NODE DESIGNS

- Trie
- Array
- Skip-List
- Hash-Table
- Linked-List
- Sorted Array
POSSIBLE STRUCTURES

Trie, Array, Skip-List, Hash-Table, Linked-List, Sorted Array

POSSIBLE NODE DESIGNS

Hash-Table, Array, Skip-List, Trie, Linked-List, Sorted Array, Hash-Table, Array, Skip-List, Trie, Linked-List, Sorted Array

POSSIBLE NODE DESIGNS

Array, Sorted Array, Trie, Hash-Table, Skip-List, Linked-List

POSSIBLE STRUCTURES

Array, Sorted Array, Trie, Hash-Table, Skip-List, Linked-List
POSSIBLE NODE DESIGNS

Trie
Array
Skip-List
Hash-Table
Linked-List
Sorted Array

POSSIBLE STRUCTURES

Unknown 1
Unknown 2
Unknown N

Array
Linked-List
Skip-List
Trie
Hash-Table
Sorted Array

POSSIBLE NODE DESIGNS

10^6

DA@Harvard SEAS
PERIODIC TABLE OF ELEMENTS explains and predicts missing elements

Dimitri Mendelev structures elements based on atomic number, electron configuration, and recurring chemical properties.
Kosuke Morita

nihonium
# Periodic Table of Data Structures

<table>
<thead>
<tr>
<th>classes of designs</th>
<th>B-trees &amp; Variants</th>
<th>Tries &amp; Variants</th>
<th>LSM-Trees &amp; Variants</th>
<th>Differential Files</th>
<th>Membership Tests</th>
<th>Zone maps &amp; Variants</th>
<th>Bitmaps &amp; Variants</th>
<th>Hashing</th>
<th>Base Data &amp; Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioning</td>
<td>DONE</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>Logarithmic Design</td>
<td>DONE</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>Fractional Cascading</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>Log-Structured</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>Buffering</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>Differential Updates</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>Sparse Indexing</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>Adaptivity</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>classes of designs</td>
<td>B-trees &amp; Variants</td>
<td>Tries &amp; Variants</td>
<td>LSM-Trees &amp; Variants</td>
<td>Differential Files</td>
<td>Membership Tests</td>
<td>Zone maps &amp; Variants</td>
<td>Bitmaps &amp; Variants</td>
<td>Hashing</td>
<td>Base Data &amp; Columns</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>---------------------</td>
<td>-------------------</td>
<td>--------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Partitioning</td>
<td>DONE</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
<td>DONE</td>
</tr>
<tr>
<td>Logarithmic Design</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fractional Cascading</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log-Structured</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffering</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Differential Updates</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse Indexing</td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adapitivity</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

periodic table of data structures
# Periodic Table of Data Structures

<table>
<thead>
<tr>
<th>Classes of Designs</th>
<th>B-trees &amp; Variants</th>
<th>Tries &amp; Variants</th>
<th>LSM-Trees &amp; Variants</th>
<th>Differential Files</th>
<th>Membership Tests</th>
<th>Zone maps &amp; Variants</th>
<th>Bitmaps &amp; Variants</th>
<th>Hashing</th>
<th>Base Data &amp; Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioning</td>
<td>DONE</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
<td>DONE</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>DONE</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
<td>DONE</td>
</tr>
<tr>
<td>Differential</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Updates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse Indexing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adaptivity</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
</tr>
</tbody>
</table>

**PAPER MACHINE**
# Periodic Table of Data Structures

## Classes of Designs

<table>
<thead>
<tr>
<th>Method</th>
<th>B-trees &amp; Variants</th>
<th>Tries &amp; Variants</th>
<th>LSM-Trees &amp; Variants</th>
<th>Differential Files</th>
<th>Membership Tests</th>
<th>Zone maps &amp; Variants</th>
<th>Bitmaps &amp; Variants</th>
<th>Hashing</th>
<th>Base Data &amp; Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioning</td>
<td>DONE</td>
<td>DONE</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td>DONE</td>
<td></td>
</tr>
<tr>
<td>Logarithmic</td>
<td>DONE</td>
<td>DONE</td>
<td>DONE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Differential Updates

- **DONE**

## Sparse Indexing

- **DONE**

## Adaptivity

- **DONE**

## Paper Machine

- **DONE**

---

**Updatable Bitmap Indexes**

[@SIGMOD16](#SIGMOD16)

---

**Classes of Primitives**

---

[@IEEE.EngBul18](#IEEE.EngBul18)
The taxonomy is used to shed light both on the nature of the design space and on the performance tradeoffs implied by many of the choices that exist in the design space.
"The taxonomy is used to shed light both on the nature of the design space and on the performance tradeoffs implied by many of the choices that exist in the design space."
STARS IN THE SKY

\[10^{24}\]

POSSIBLE DATA STRUCTURES

\[10^{32}, \text{2-node}\]
\[10^{48}, \text{3-node}\]

@Harvard SEAS
DESIGN SPACE

fundamental building blocks

properties when combined
fundamental building blocks when combined
HOW TO JUDGE A DESIGN?

1. COMPLEXITY ANALYSIS

2. IMPLEMENTATION & TESTING
HOW TO JUDGE A DESIGN?

1. COMPLEXITY ANALYSIS
2. IMPLEMENTATION & TESTING
3. GENERALIZED MODELS