



The Data Calculator

Jake Bloomfeld

April 25, 2019

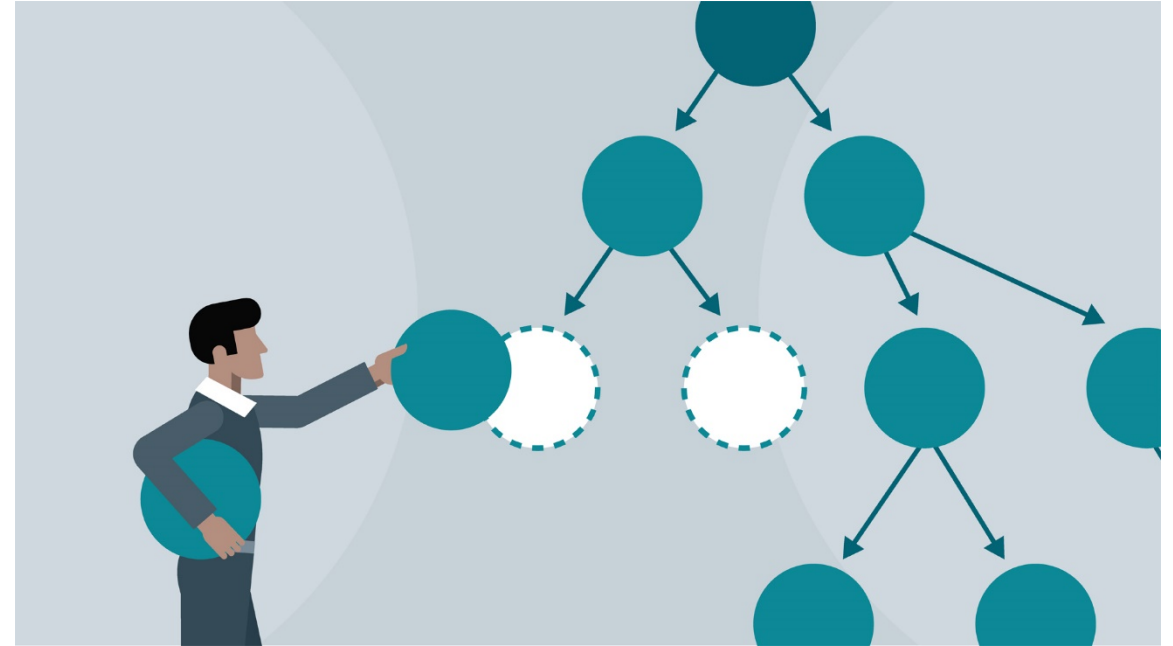
Overview

- I. Background
- II. Problem Statement
- III. Solution Overview
- IV. Inspiration & Contributions
- V. Data Layout Primitives and Structure Specifications
- VI. Data Access Primitives and Cost Synthesis
- VII. What-If Design and Auto-Completion
- VIII. Experimental Analysis
- IX. Summary

I. Background

Data Structure Defined

- A data organization, management, and storage format that enables efficient access and modification
- A collection of data values, the relationships among them, and the functions or operations that can be applied to the data



Types

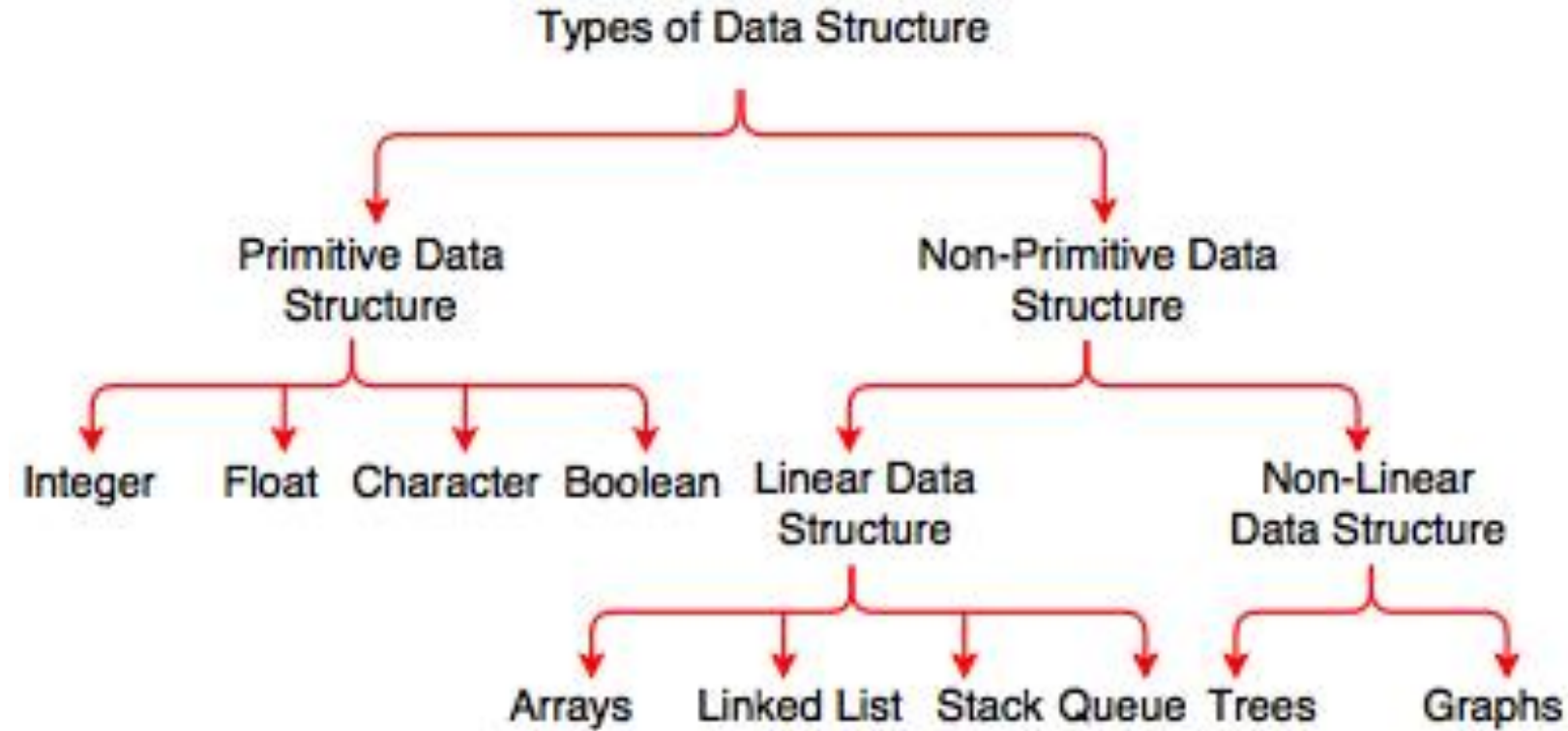
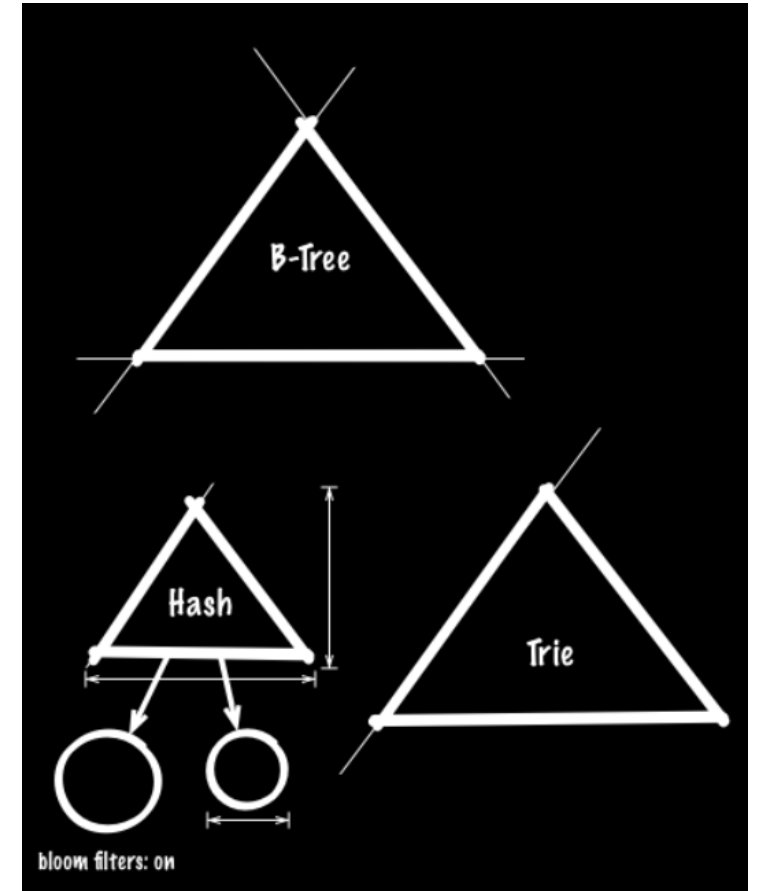
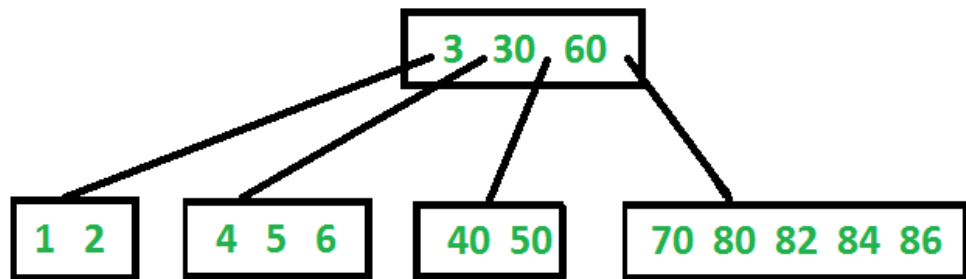
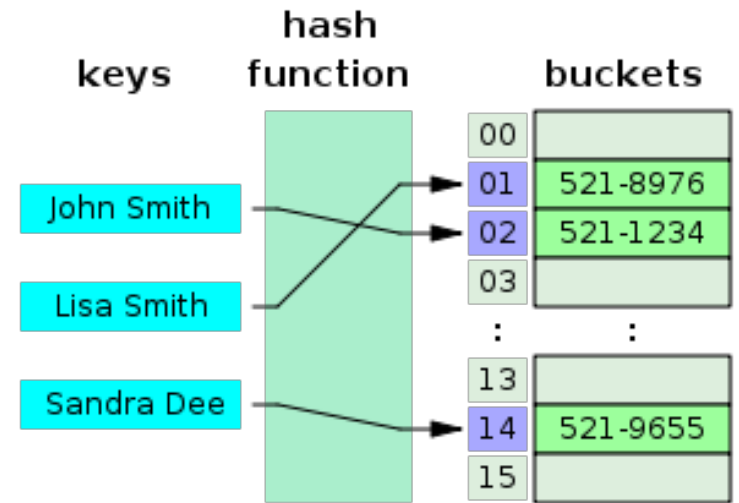


Fig. Types of Data Structure

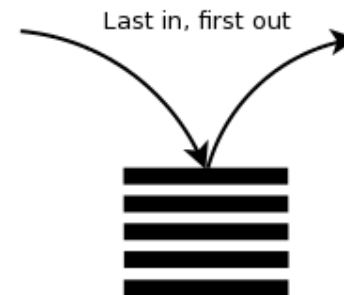


Common Examples

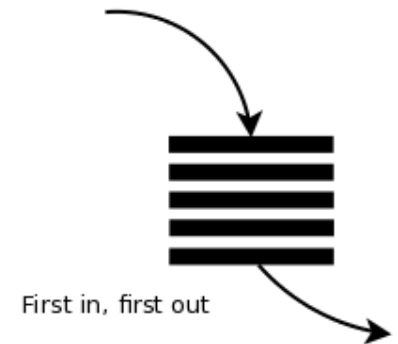
- Relational databases use **B-tree** indexes to retrieve data
- Compilers use **hash tables** to look up identifiers
- CPUs use **queues** to serve FIFO requests or **stacks** to serve LIFO requests



Stack:



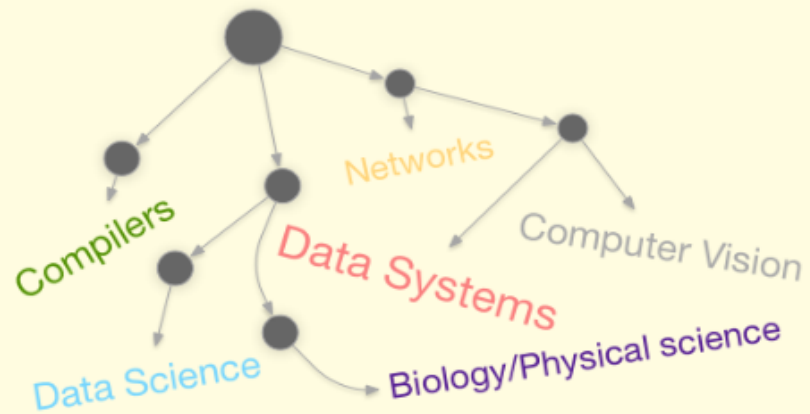
Queue:



Basically...

DATA STRUCTURES ARE EVERYWHERE

this is how any data driven subfield and industry store and access data



II. Problem Statement

The Problem: Human-Driven Design

- Hard to design due to a massive design space and the dependence of performance on variable workload and hardware

LET US CALCULATE

WHAT IF WE COULD REASON ABOUT THE

DESIGN SPACE OF DATA STRUCTURES?

Design Questions to Consider

- Data structure for a specific workload:
 - Should we strip down an existing complex data structure?
 - Should we build off a simpler one?
 - Should we design and build a new one from scratch?
- Data structure for variable workload:
 - How will performance change?
 - Should we redefine our core data structures?

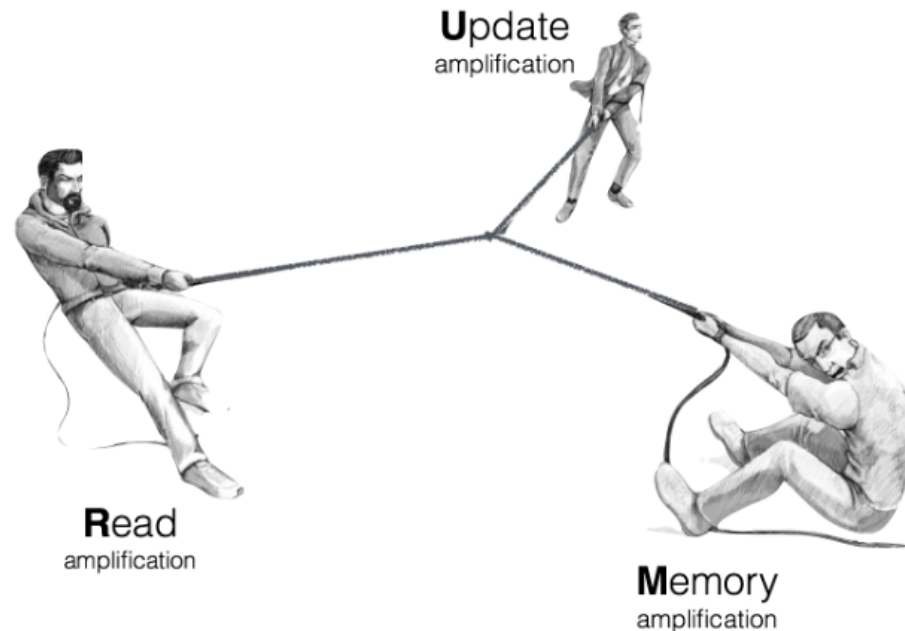
Design Questions to Consider

- We add flash drives with more bandwidth and add more system memory:
 - Should we change the layout of our B-tree nodes?
 - Should we change the size ratio in our LSM-tree?
- We want to improve throughput:
 - How beneficial would it be to buy faster disks? More memory?
 - Should we invest the same budget in redesigning our core data structure?

What's the Perfect Data Structure?

THERE IS NO PERFECT DATA STRUCTURE

~100 new data structures are designed every year due to continuous hardware and workload changes



An Actual Quote

- “I know from experience that getting a new data structure into production takes years. Over several years, assumptions made about the workload and hardware are likely to change, and these changes threaten to reduce the benefit of a data structure. This risk of change makes it hard to commit to multi-year development efforts. We need to reduce the time it takes to get new data structures into production.” – a sad system architect



Another Actual Quote

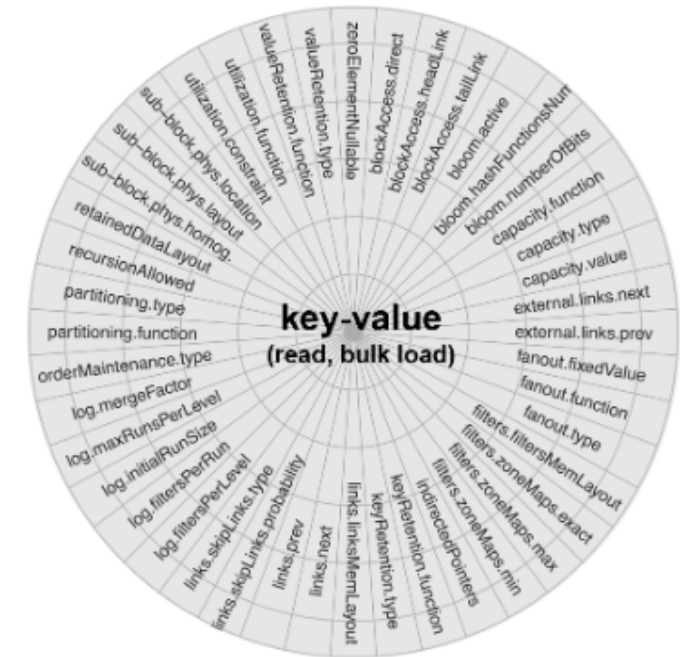
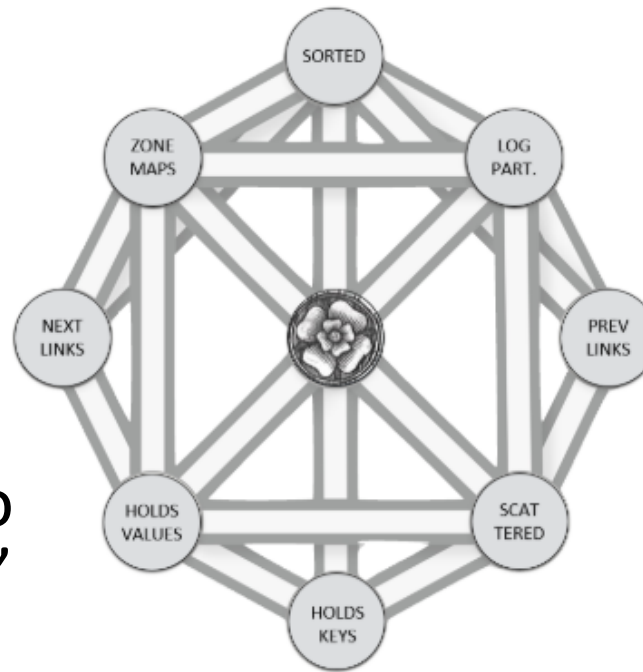
- “Another problem is the limited ability we have to iterate. While some changes only require an online schema change, many require a dump and reload for a data service that might be running 24x7. The budget for such changes is limited. We can overcome the limited budget with tools that help us determine the changes most likely to be useful. Decisions today are frequently based on expert opinions, and these experts are in short supply.” – another sad system architect



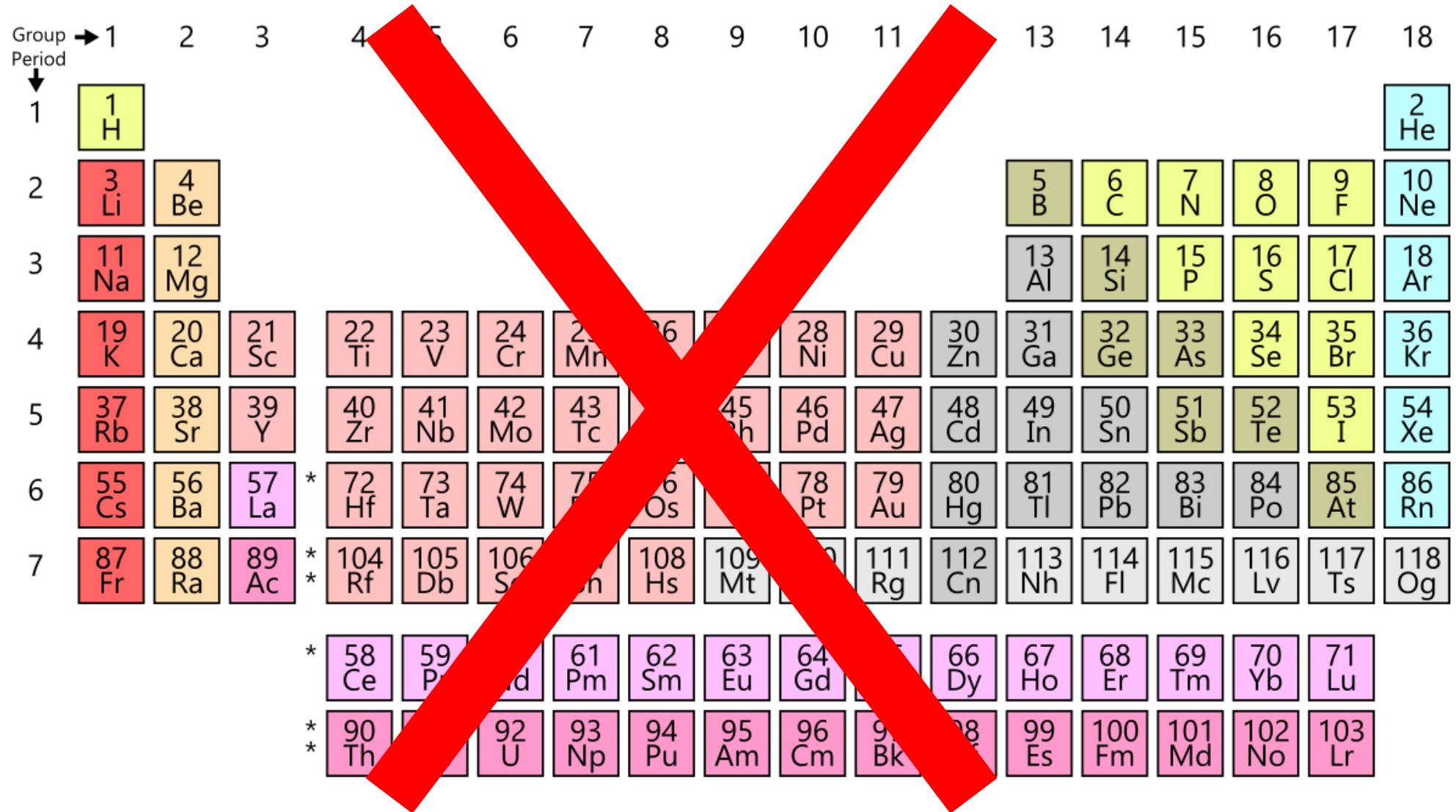
III. Solution Overview

Vision Step 1: Design Synthesis from First Principles

- Nearly all new designs are about combining a small set of fundamental concepts in different ways or tunings
- “If we can describe the set of the first principles of data structure design, then we will have a structured way to express all possible designs.”



Vision: A Periodic Table



The image shows a standard periodic table with a large red 'X' drawn over it. The table is organized by groups (1-18) and periods (1-7). Elements are color-coded by groups: Group 1 (yellow), Group 2 (orange), Groups 13-18 (various shades of green, yellow, and cyan), and Groups 3-12 (various shades of pink and purple). The f-block elements (lanthanides and actinides) are shown at the bottom, separated from the main body of the table by asterisks.

Group	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Period 1	1 H																	2 He
Period 2	3 Li	4 Be											5 B	6 C	7 N	8 O	9 F	10 Ne
Period 3	11 Na	12 Mg											13 Al	14 Si	15 P	16 S	17 Cl	18 Ar
Period 4	19 K	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As	34 Se	35 Br	36 Kr
Period 5	37 Rb	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb	52 Te	53 I	54 Xe
Period 6	55 Cs	56 Ba	57 La	* 72 Hf	73 Ta	74 W	75 Re	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl	82 Pb	83 Bi	84 Po	85 At	86 Rn
Period 7	87 Fr	88 Ra	89 Ac	* 104 Rf	105 Db	106 Sg	107 Bh	108 Hs	109 Mt	110 Ds	111 Rg	112 Cn	113 Nh	114 Fl	115 Mc	116 Lv	117 Ts	118 Og
Lanthanides				* 58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu	
Actinides				* 90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr	

Vision: Periodic Table of Data Structures

- Express massive design space
- Present a set of first principles that can synthesize orders of magnitude more data structure designs than what has been published
- Same concept of Periodic Table of Elements

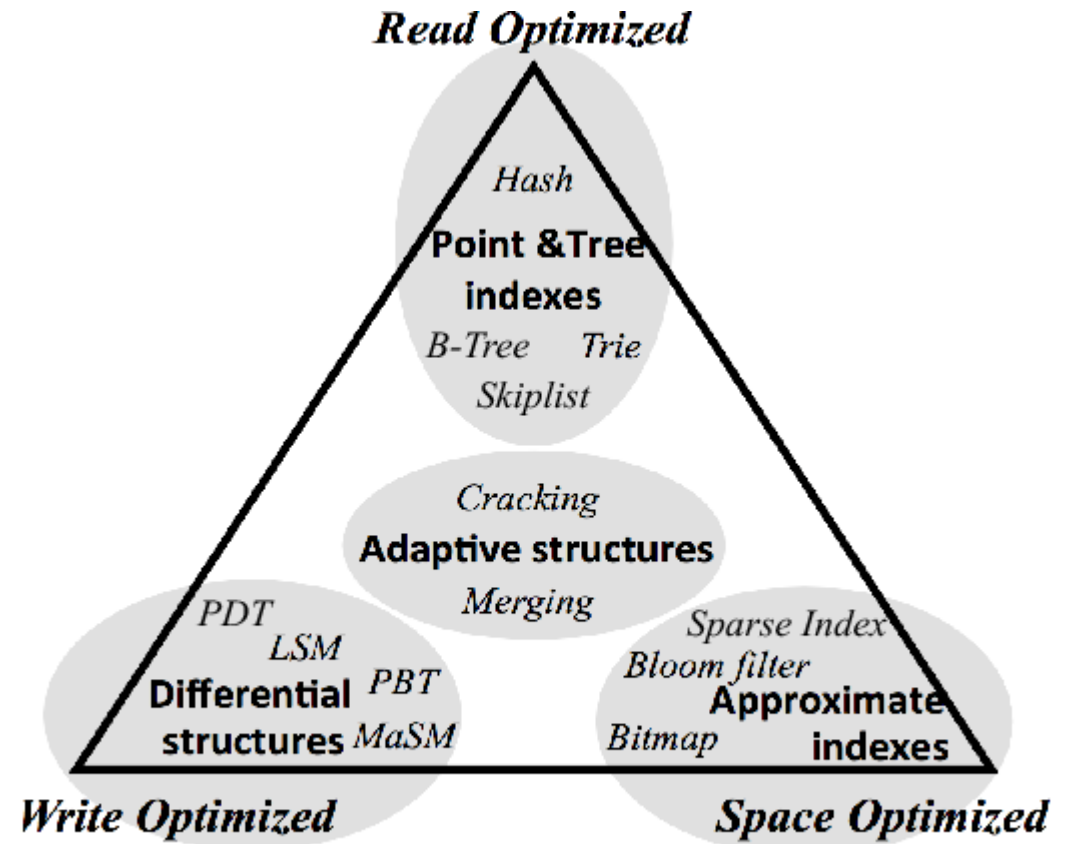


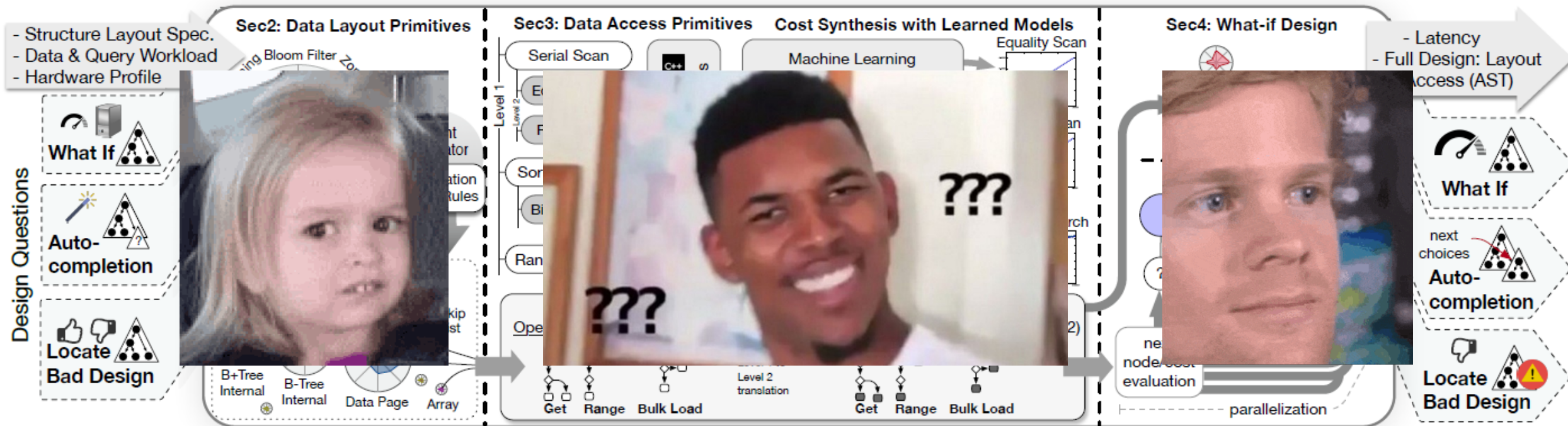
Figure 1: Each design compromises between

Vision Step 2: Cost Synthesis from Learned Models

- Accelerate and automate design and testing process
- Can we accelerate this process to quickly test alternative designs (or combos of hardware, data, queries, etc.) in the order of seconds?
- Enable new kinds of adaptive systems that can decide core parts of their design and the right hardware

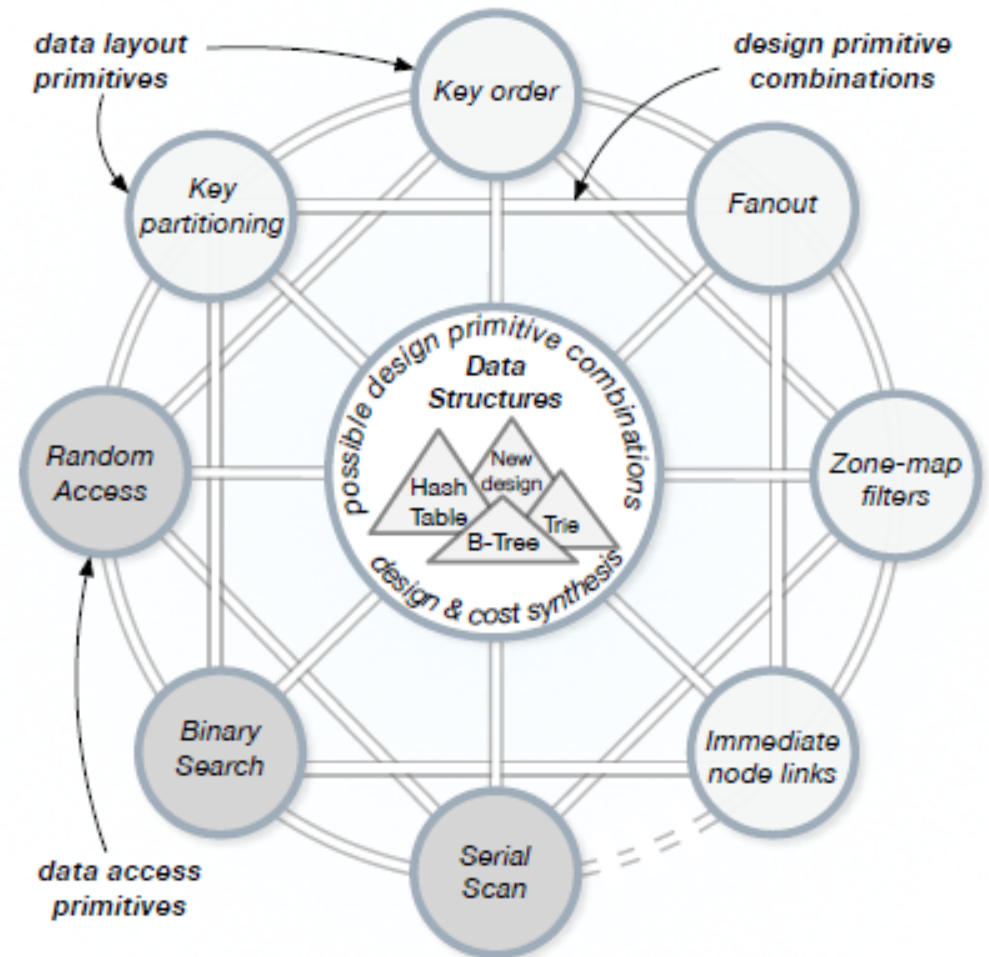


Overall Architecture



A Concept: The Data Calculator

- Computes the performance of arbitrary data structure designs as combinations of fundamental design primitives
- If we can describe the set of the first principles of data structure design, then we will have a structured way to express all possible designs as combinations of these principles

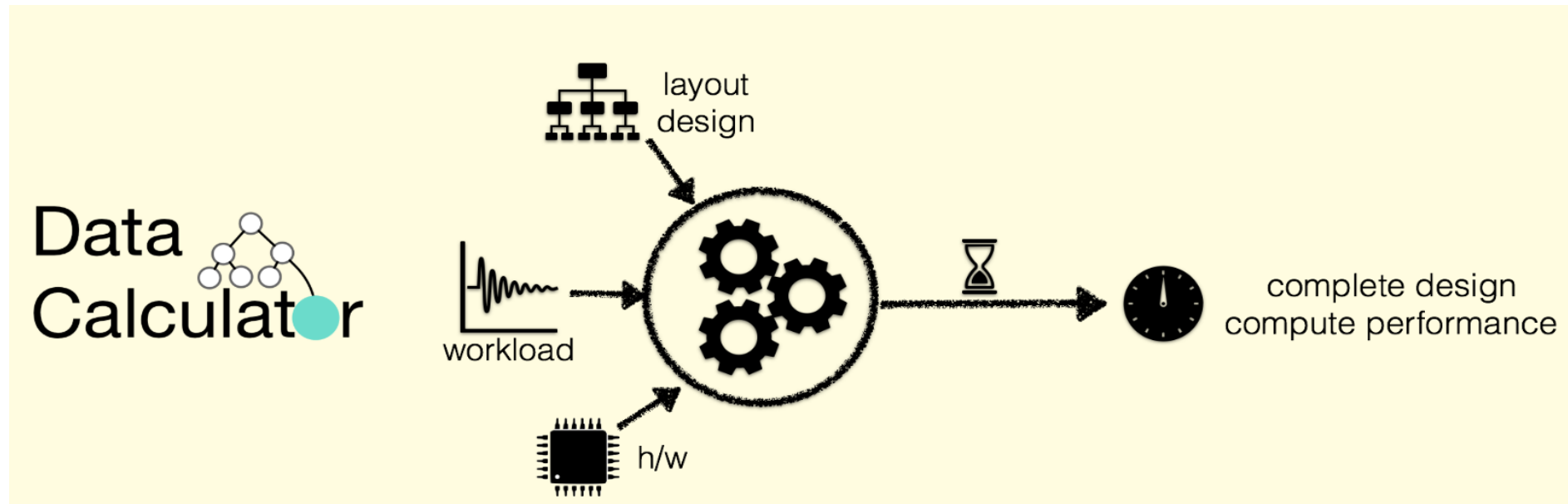


The Data Calculator

- Able to answer what-if data structure design questions to understand how the introduction of new design choices, workloads, and hardware affect the performance of an existing design
- Supports read queries for basic hardware conscious layouts

I/O

- Input: allows users to input a high-level specification of the layout of a data structure (combination of primitives), along with workload and hardware specification
- Output: a calculation of the latency to run the input workload on the input hardware



IV. Inspiration & Contributions

Inspiration

- Can be seen as a step toward the Automatic Programmer challenge set by Jim Gray in his Turing award lecture
- A step toward the “calculus of data structures” challenge set by Turing award winner Robert Tarjan
- *“What makes one data structure better than another for a certain application? The known results cry out for an underlying theory to explain them.”*



Contributions

- Introduce a set of data layouts design primitives that capture the first principles of data layouts, including hardware conscious designs
- Show how combinations of the design primitives can describe typical data structures
- Show that design primitives form a massive space of possible designs that has been minimally explored
- Show how to synthesize the latency cost of basic operations of arbitrary data structure designs from a small set of access primitives

Contributions

- Show how to use cost synthesis to answer what-if design questions
- Introduce a design synthesis algorithm that completes partial layout specifications given a workload and hardware input
- Demonstrate that the Data Calculator can accurately compute the performance impact of design choices for state-of-the-art designs and diverse hardware
- Demonstrate that the Data Calculator can accelerate the design process by answering complex questions in seconds or minutes

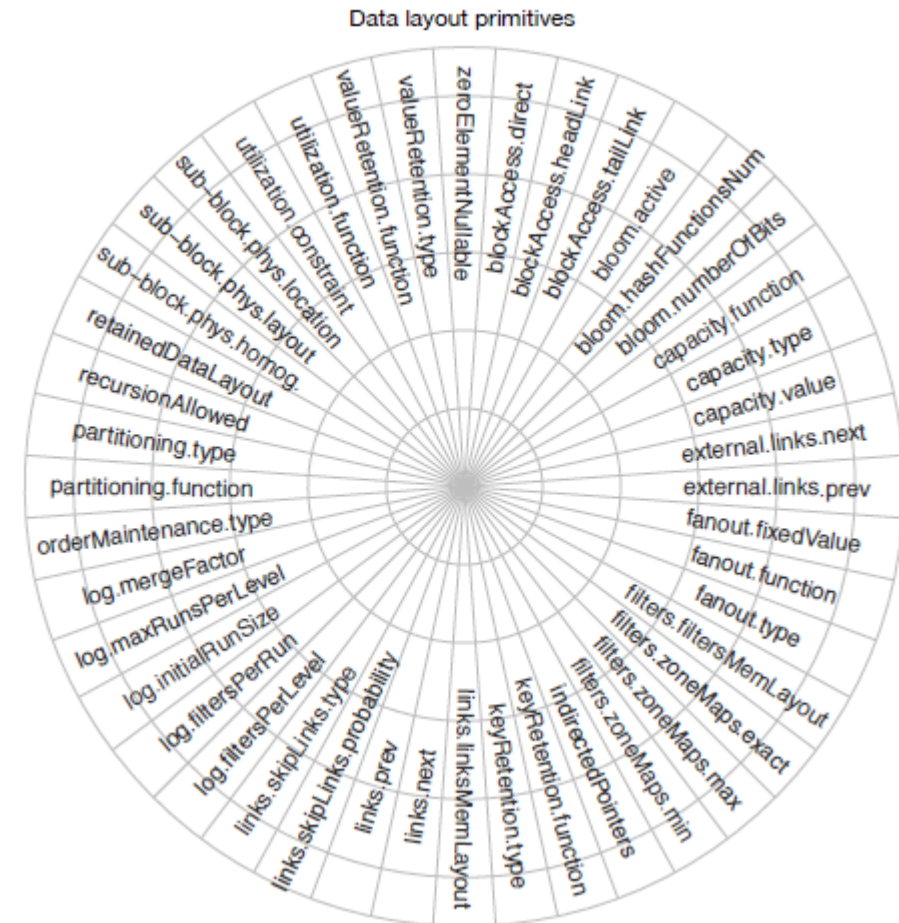
V. Data Layout Primitives and Structure Specifications

Data Layout Primitives

- The Data Calculator contains a small set of design primitives that represent fundamental design choices when constructing a data structure layout
- Each belongs to a higher class:
 - Node data organization
 - Partitioning
 - Node physical placement
 - Node metadata management

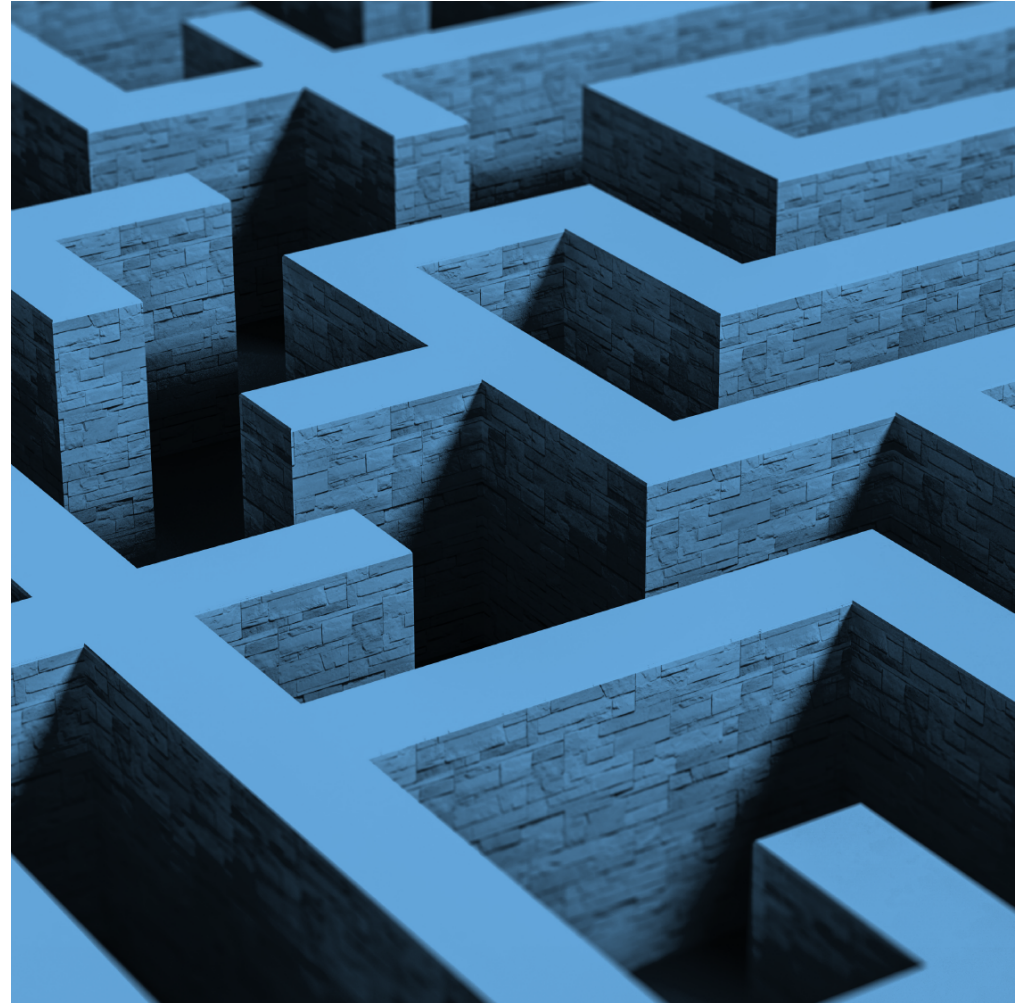
From Layout Primitives to Data Structures

- An element is a full specification of a single data structure node; it defines the data and access methods used to access the node's data
- An element may be “terminal” or “non-terminal”, which means it may be describing a node that further partitions data to more nodes or not



Size of Design Space

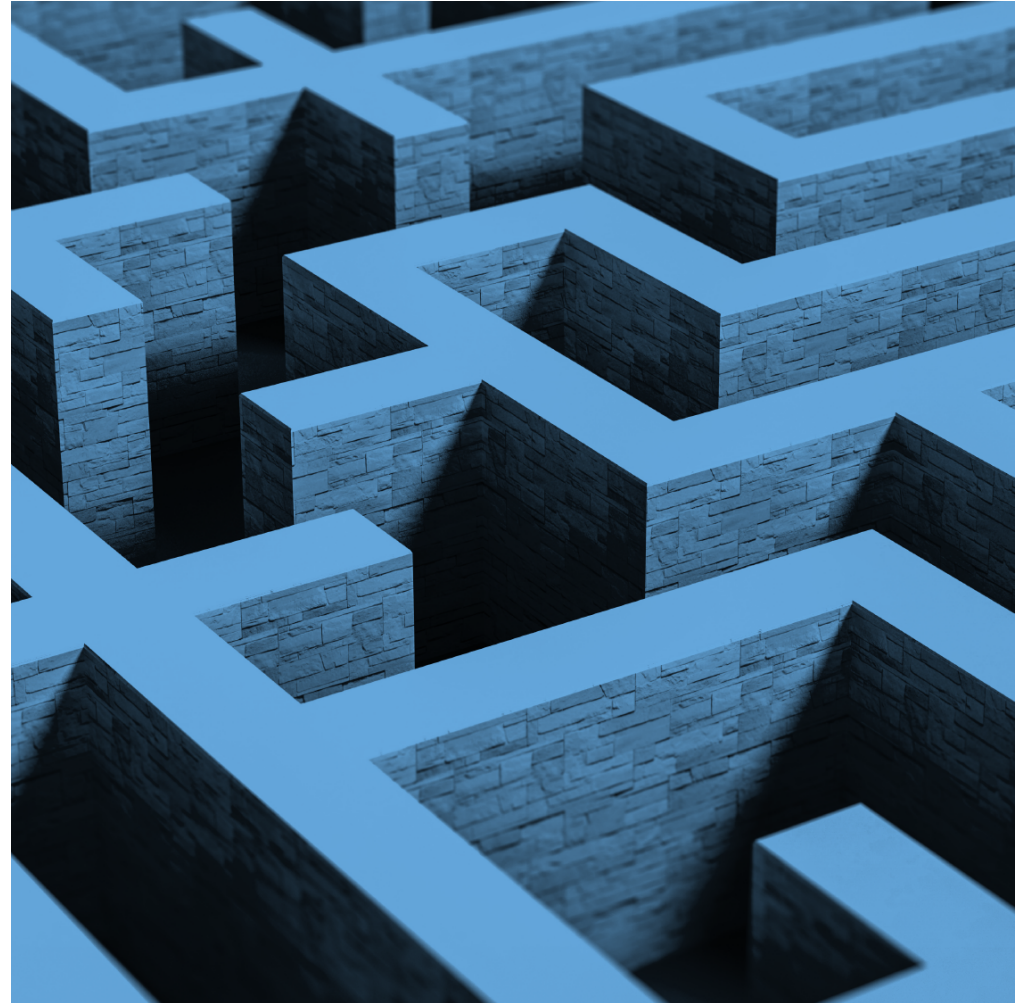
- Data Layout Primitive: A primitive p_i belongs to a domain of values P_i and describes a layout aspect of a data structure node
- Data Structure Element: A data structure element E is defined as a set of data layout primitives: $E = \{p_1, \dots, p_n\} \in P_1 \times \dots \times P_n$ that uniquely identify it



Size of Design Space

- Given a set of $Inv(\mathcal{P})$ invalid combinations, the set of all possible elements that can be designed as distinct combinations of data layout primitives has the following cardinality:

$$|\mathcal{E}| = \mathcal{P}_1 \times \dots \times \mathcal{P}_n - Inv(\mathcal{P}) = \prod_{\forall \mathcal{P}_i \in \mathcal{E}} |\mathcal{P}_i| - Inv(\mathcal{P})$$



VI. Data Access Primitives and Cost Synthesis

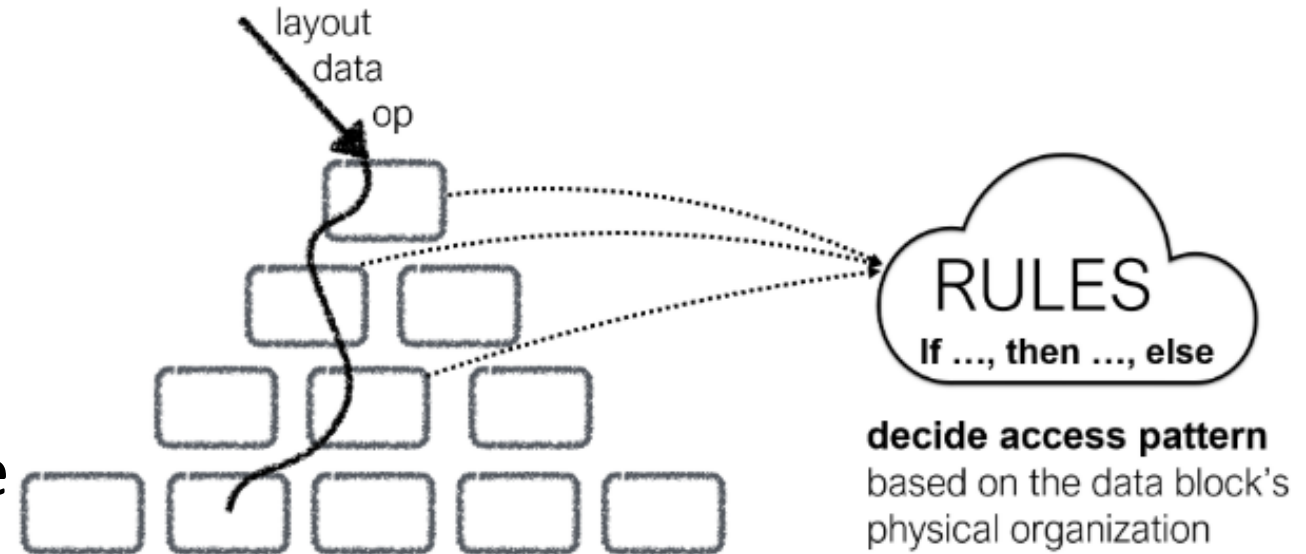
So, what's the damage?

- Traditional cost analysis in systems and data structures happen through experiments and analytical cost models – not scalable
- Intuition: synthesize complex operations from their fundamental components, and develop a hybrid way to assign costs to each individual component



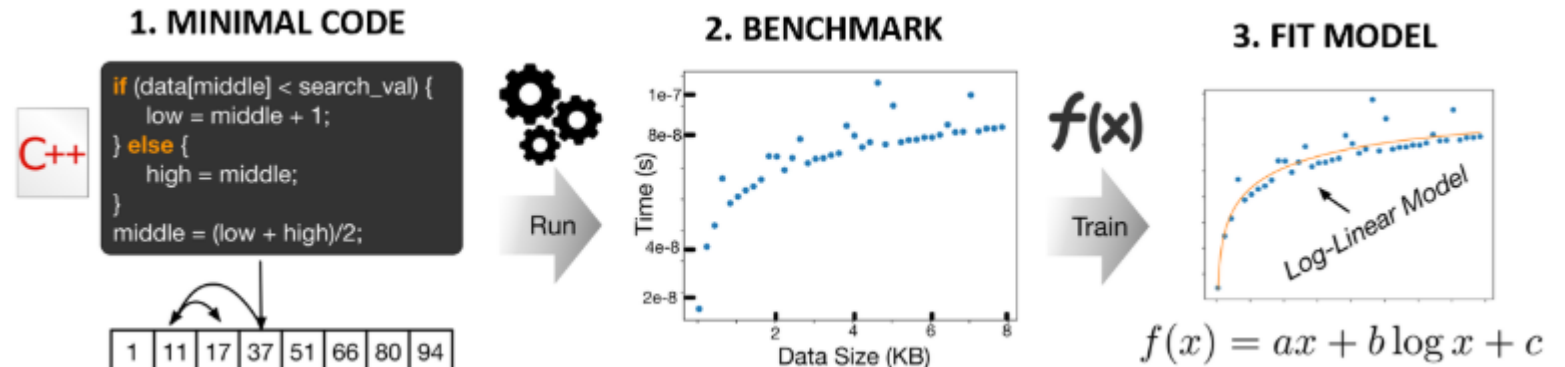
Cost Synthesis

- For each operation in a workload, the data calculator synthesizes the exact algorithm and its cost for the target hardware
- Based on the specification of the layout of each data structure node in the path of an operation, it decides the best access pattern; based on the learned models, it knows the expected cost



Learned Cost Models

- The data calculator contains a library of learned models that describe fundamental access patterns in blocks of data.
- Captures algorithmic, engineering, and hardware properties



Example: Binary Search Model

- We know that the performance is related to the size of the array by a logarithmic component
- Relationship for small array sizes (< 8 elements) might not exactly fit a logarithmic function, so add a linear term to capture some small linear dependency on the data size
- Cost of binary searching an array of n elements can be approx. as:

$f(n) = c_1n + c_2 \log n + y_0$ where $c_1, c_2,$ and y_0
are coefficients learned through linear regression

Example: Binary Search Model

- The values of these coefficients help us translate the abstract model into a predictive model, which has taken into account factors such as CPU speed and the cost of memory accesses across the sorted array for the specific hardware
- The Data Calculator can then use this learned model to query for the performance of binary search within the trained range of data sizes



VII. What-If Design and Auto-Completion

Benefits of What-If

- Improves the productivity of engineers by quickly iterating over designs and scenarios before committing to an implementation
- Accelerates research by allowing researchers to easily and quickly test completely new ideas
- Develops educational tools that allow for rapid testing of concepts
- Develops algorithms for online auto-tuning and online adaptive systems that transition between designs



WHAT-IF DESIGN

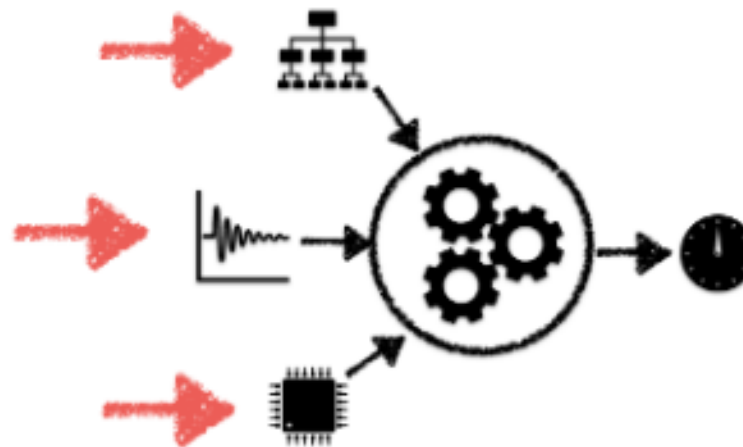
What-If Design

- One can form design questions by varying any one of the input parameters of the Data Calculator
 - Data structure (layout) specification
 - Hardware profile
 - Workload (data and queries)

*What-if we **add bloom filters**
in the hash-table buckets?*

*What-if the workload
changes to **90% writes**?*

*What-if we **buy faster CPU X**?*



Auto-Completion

- Can complete partial layout specifications and a hardware profile, given a workload
- Input is a partial layout specification, data, queries, hardware, and the set of the design space that should be considered as part of the solution
- Data Calculator computes the rest of the missing subtree of the hierarchy of elements
- Algorithm considers a new element as candidate for one of the nodes of the missing subtree and computes the cost



AUTO-DESIGN

VIII. Experimental Analysis

Experiment 1: Accurate Cost Synthesis

- Tests the ability to accurately cost arbitrary data structure specifications across different machines
- Compares the cost generated automatically by the Data Calculator with the cost by testing a full implementation of that data structure



Experiment Setup

- Tested the following structures:
 - Array
 - Sorted array
 - Linked-list
 - Partitioned linked-list
 - Skip-list
 - Trie
 - Hash-table
 - B+ tree
- Tested algorithms on each:
 - Get
 - Range Get
 - Bulk Load
 - Update
- Data workload of 100,000 uniformly distributed integers with a sequence of 100 Get requests
- Incrementally insert more, and repeat query workload

Overview of Experiment Results

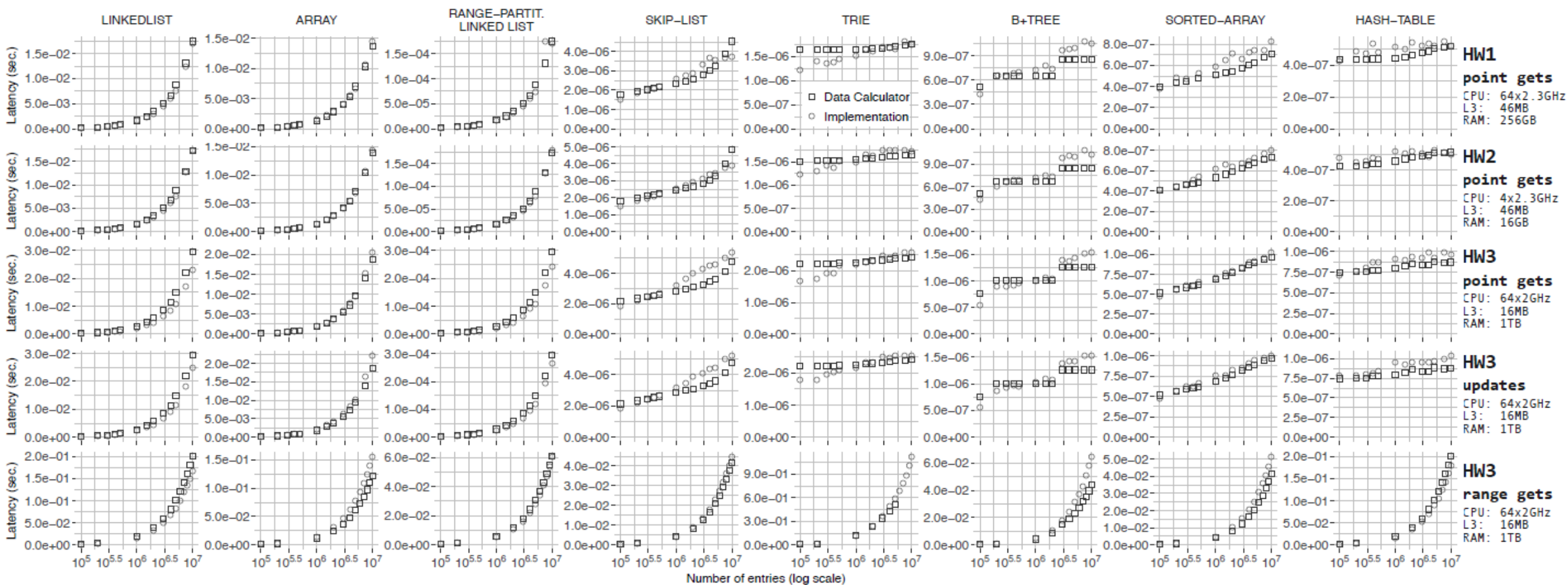
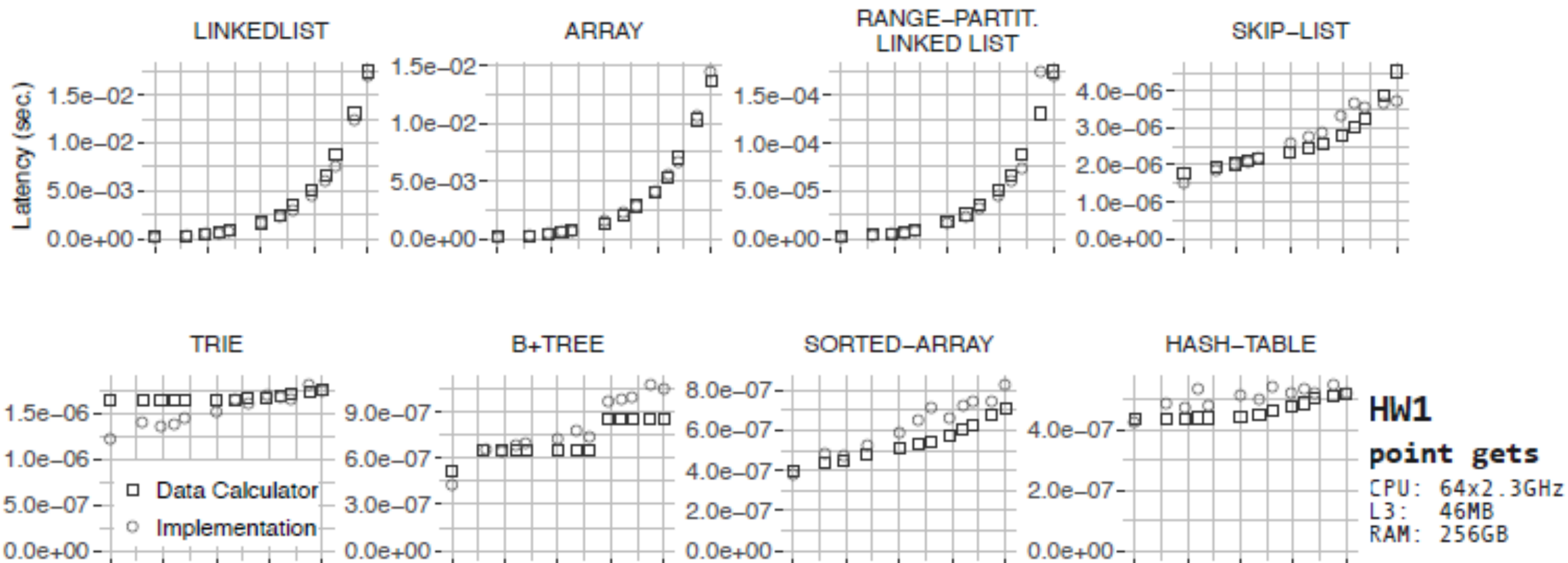
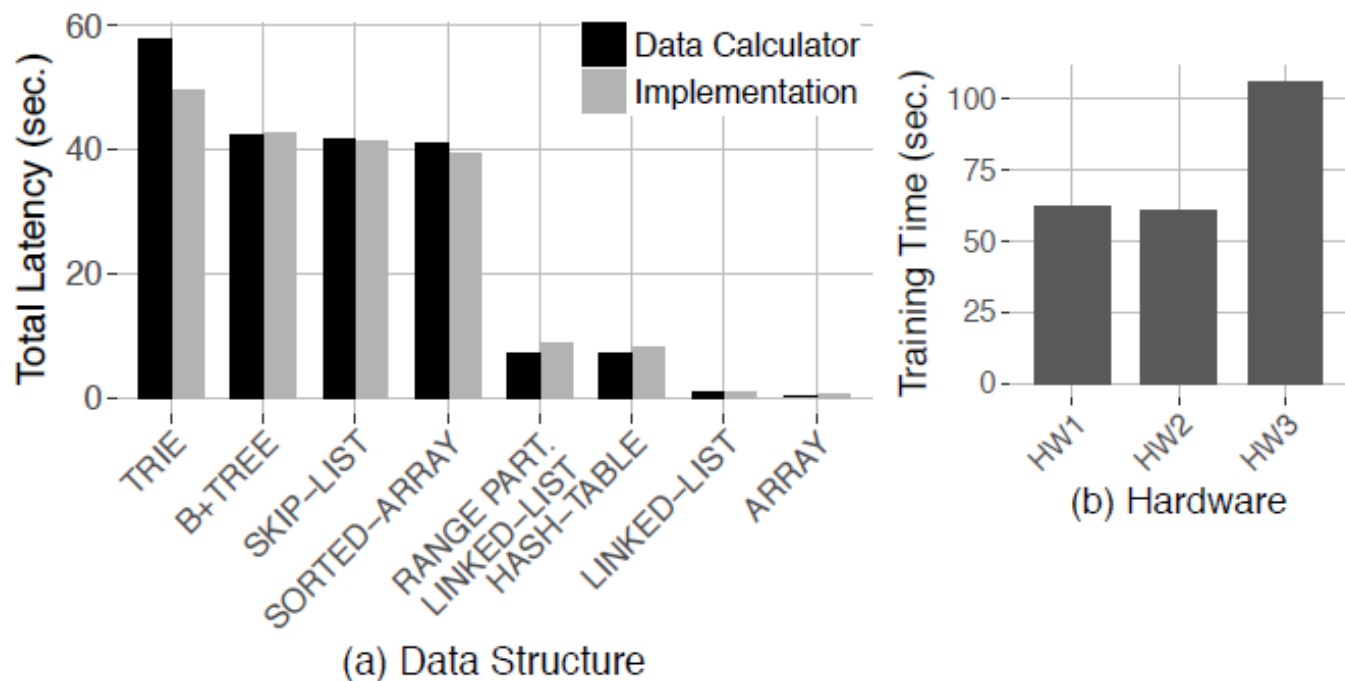


Figure 6: The Data Calculator can accurately compute the latency of arbitrary data structure designs across a diverse set of hardware and for diverse dictionary operations.

Experiment Results for Hardware 1



Experiment 2: Training Access Primitives



- Data Calculator can accurately synthesize the bulk loading costs for all data structures
- The time needed to train all primitives on a diverse set of machines is inexpensive

Figure 7: Computing Bulk-loading cost (left) and Training cost across diverse hardware (right).

Experiment 3: Cache-Friendly Designs

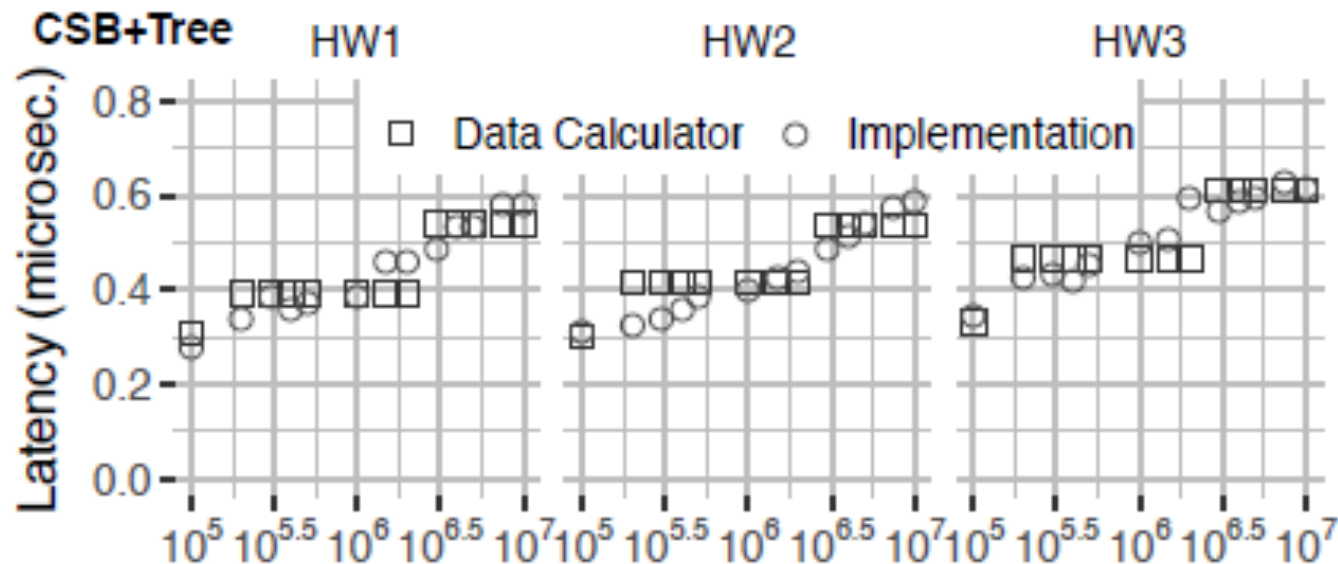


Figure 8: Accurately computing the latency of cache conscious designs in diverse hardware and workloads.

- Data Calculator accurately predicts the performance behavior across a diverse set of machines, capturing caching effects of growing data sizes and design patterns where the relative position of nodes affects tree traversal costs

Experiment 4: Rich Design Questions

- B-Tree design, 1M inserts and 100 point gets, H/W 1
- User asks: “What if we change to H/W 3?”
- Data Calculator takes 20 seconds to compute that performance will decrease
- User asks: “Would it be beneficial to add a bloom filter in all B-tree leaves?”
- Data Calculator takes 20 seconds to compute that the performance will increase

IX. Summary

Takeaways



- Data Calculator allows researchers and engineers to interactively and semi-automatically navigate complex design decisions when designing or re-designing data structures, considering new workloads, and hardware
- Broke down and clearly explained each aspect of overall architecture
- Provided great visualizations and diagrams to get a clear picture
- Great paper to conclude this class; touches on many topics we've studied throughout semester

Improvements & Future Work



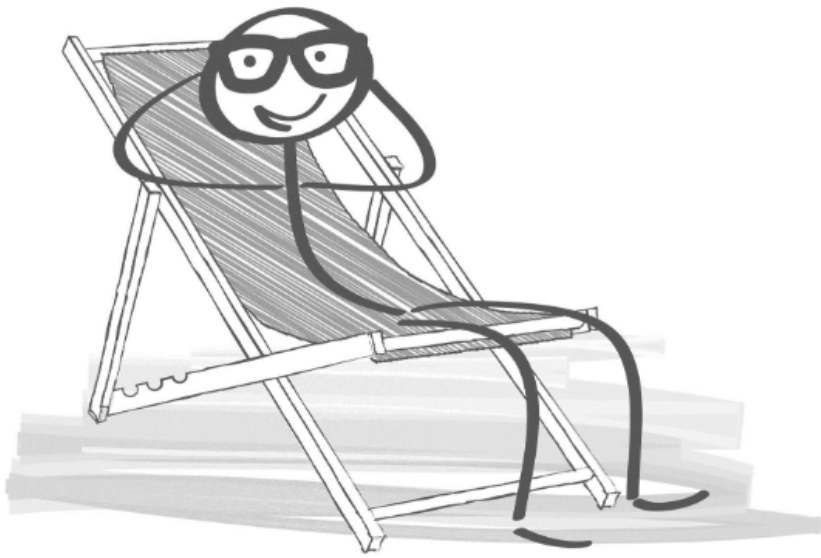
- Vague on how data structures were implemented when comparing to Data Calculator
- Should talk about how this can be commercialized
- Maybe add a case study where a software company uses Data Calculator, and give real-world results and benefits
- Translate performance savings into monetary savings
- The full Periodic Table of Data Structures

Final Remarks

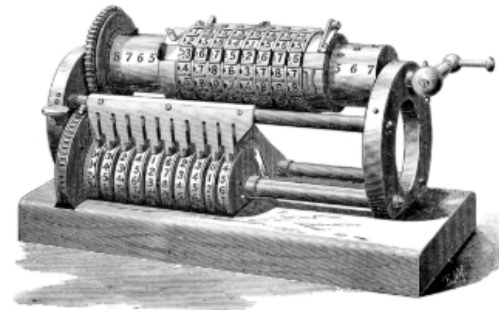
Did you know?

THERE ARE MORE POSSIBLE DATA STRUCTURES THAN **STARS IN THE SKY**

IS THIS GOING TO REPLACE ENGINEERS?



did the arithmetic calculator replace mathematicians?





THANK YOU!

References

- S. Idreos, K. Zoumpatianos, B. Hentschel, M. S. Kester, and D. Guo, [“The Data Calculator: Data Structure Design and Cost Synthesis From First Principles, and Learned Cost Models,”](#) in ACM SIGMOD International Conference on Management of Data.
- <http://daslab.seas.harvard.edu/datacalculator/>