DATA ANALYTICS USING DEEP LEARNING

GT 8803 // FALL 2018 // NIDHI MENON

LECTURE #15:

THE DATA CALCULATOR: DATA STRUCTURE DESIGN AND COST SYNTHESIS FROM FIRST PRINCIPLES

Georgia Tech

CREATING THE NEXT®

TODAY's PAPER

- **The Data Calculator: Data Structure Design and Cost Synthesis from First Principles**
  - **Authors**:
    - Stratos Idreos, Kostas Zoumpatianos, Brian Hentschel, Michael S. Kester, Demi Guo
    - DASlab (Data Systems Laboratory) @ Harvard School of Engineering and Applied Sciences
    - Presentation based on content from SIGMOD 2018 slide deck used with permission of Prof. Idreos and the Data Calculator project webpage http://daslab.seas.harvard.edu/datacalculator

TODAY'S AGENDA

- Problem Overview
- Key Idea
- Technical Details
- Experiments
- Discussion

INTRODUCTION

MOTIVATION

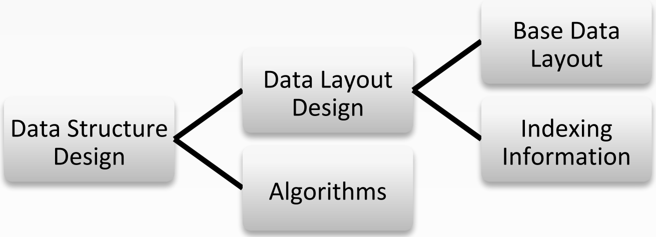
- Data Systems in the critical path of everything we do today
- Data Structures are everywhere, but there is no ‘**perfect data structure**’
- Need to accelerate design of data structures

RESULT

- A design engine that accelerates research and improves developer productivity
- Makes it easy to design, tune and use data systems for evolving hardware and workloads

BACKGROUND

- Every operation goes through a data structure
- Growing need for alternative designs:
  1. New applications
  2. New hardware
- Vast and complex design space



PROBLEM

DESIGN QUESTIONS:

1. Designing data structures for a specific workload
2. How to handle shifts in workload?
3. What will be the impact on adding more system memory, or flash drives with more bandwidth?
4. How can we improve throughput?

PROBLEMS:

- Slow design process
- Severe cost side-effects
- Increased complexity in predicting impact on performance

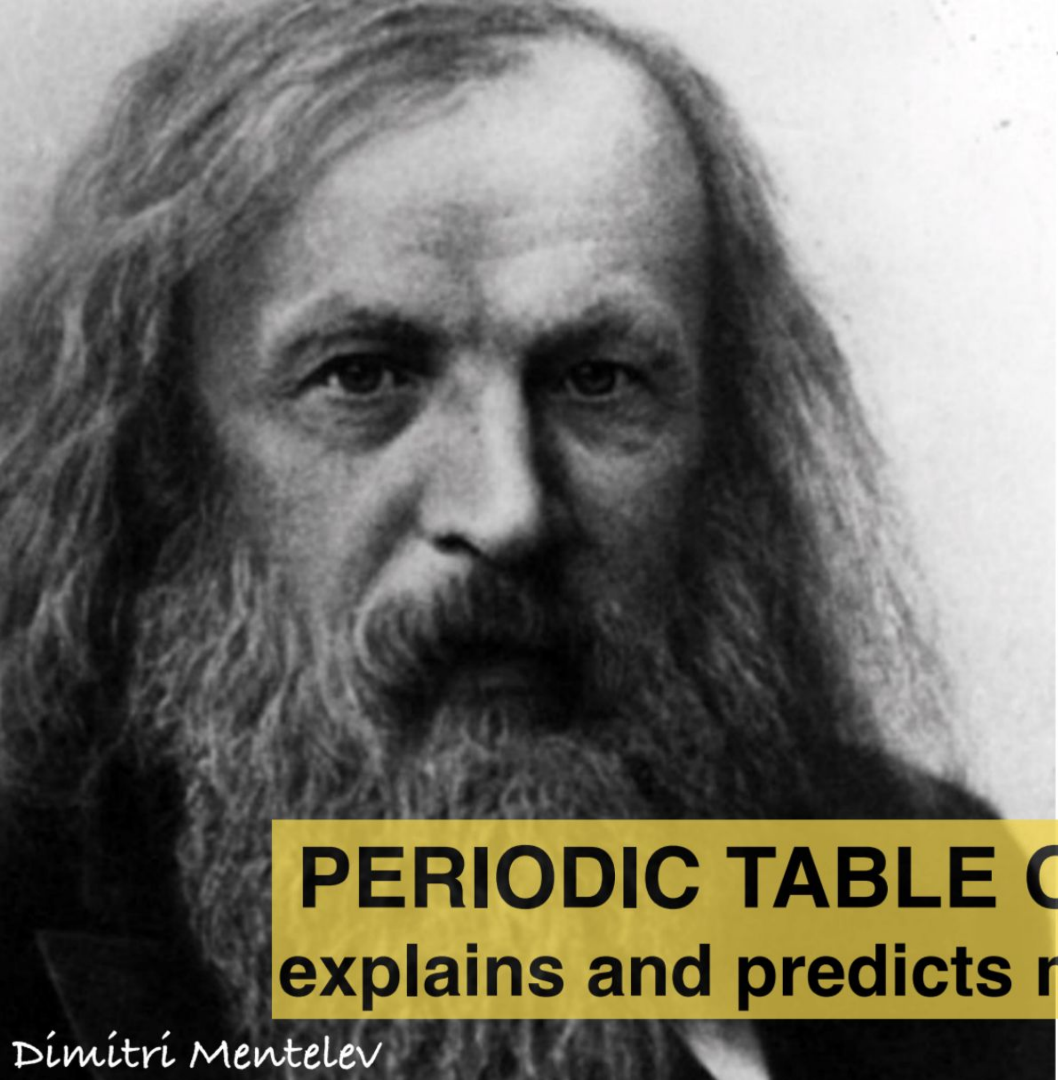
VISION

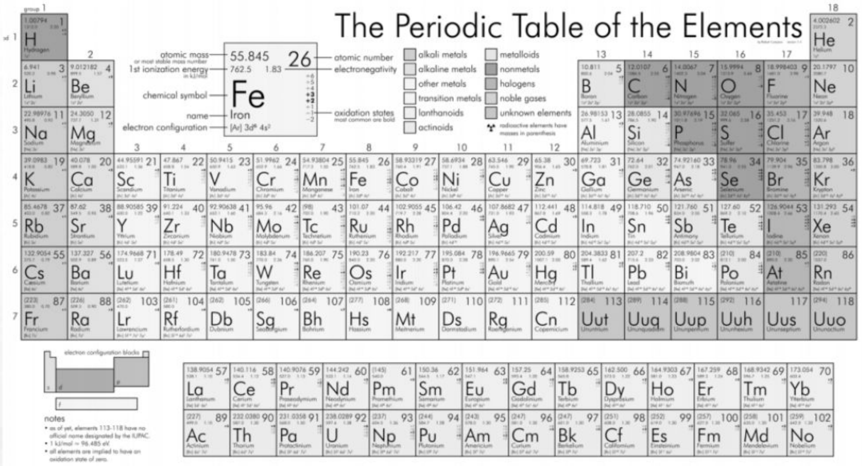
(1) Design Synthesis from First Principles

- What are the first principles?
- Why is it useful?
- How can we improve upon it?

(2) Cost Synthesis from Learned Models

- What is the goal?
- Why will it be helpful?
- How can we achieve the goal?





structures elements based on atomic number, electron configuration, and recurring chemical properties

PERIODIC TABLE OF ELEMENTS

explains and predicts missing elements

Dimitri Mentelev

FOCUS OF THE PAPER

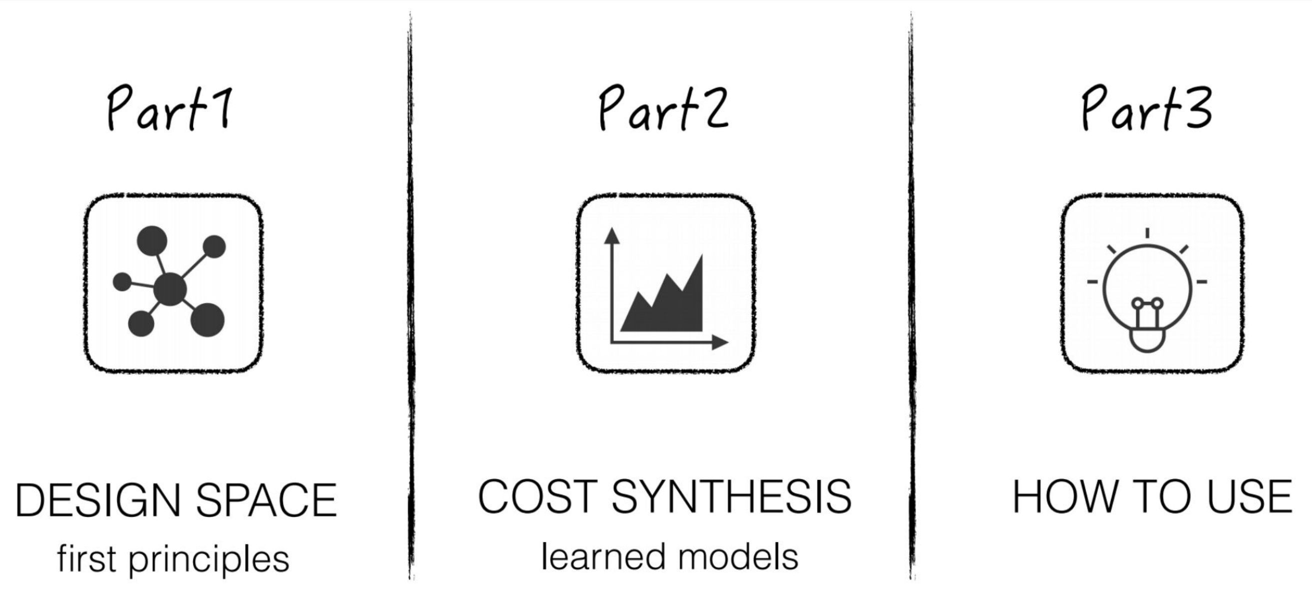


Image used with permission of Prof. Idreos from SIGMOD 2018 slide deck

DATA CALCULATOR

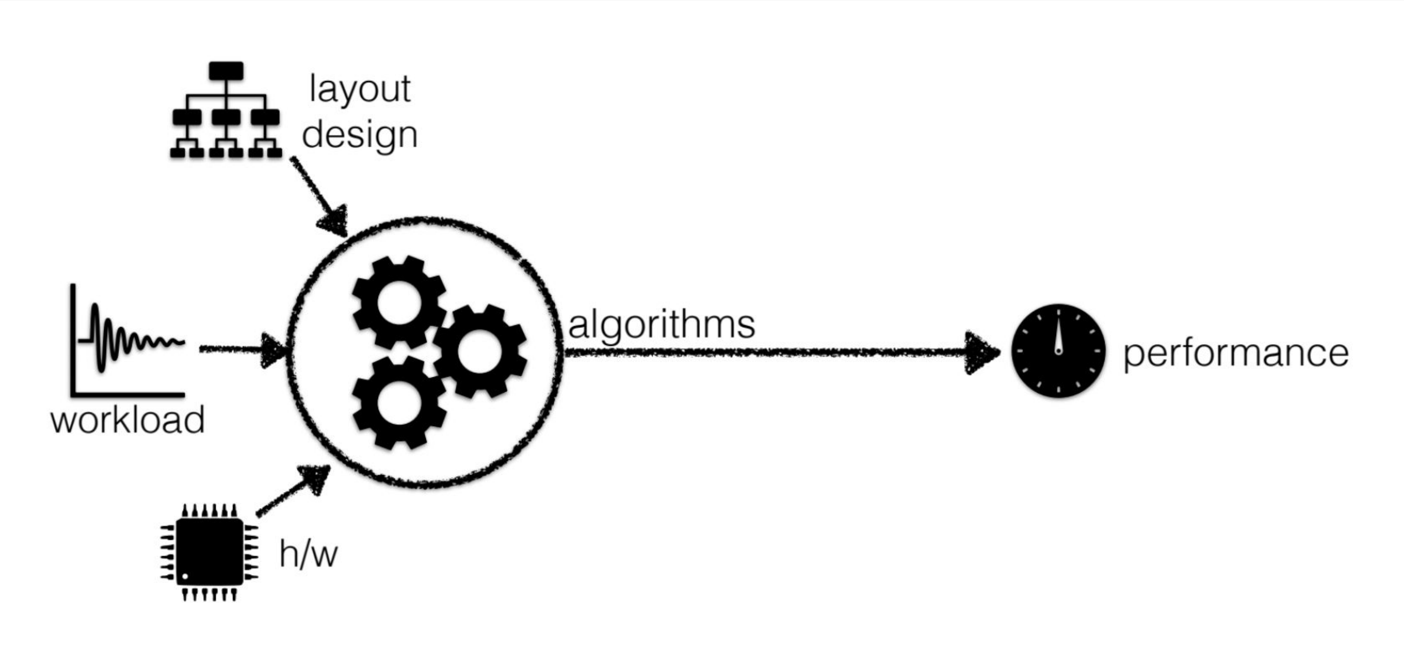
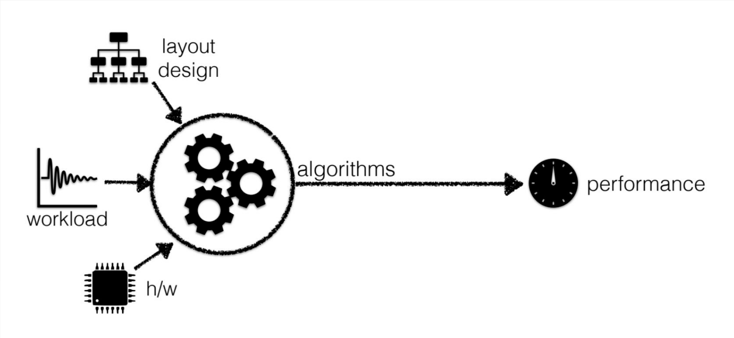


Image used with permission of Prof. Idreos from SIGMOD 2018 slide deck

DATA CALCULATOR



- Interactive and semi-automated design of data structures
- No need to code the data structure, to run the workload, or to access the hardware
- Two innovations

1. Design primitives that capture first principles of data layout design
2. Performance computation using learned cost models

Image used with permission of Prof. Idreos from SIGMOD 2018 slide deck

CONTRIBUTIONS

1. Introduced a set of data layout design primitives that capture the first principles
2. Illustrated that combinations of the design primitives can describe known data structure designs
3. Demonstrated synthesis of latency cost from a small set of access primitives
4. Introduce a design synthesis algorithm that completes partial layout specifications given a workload and hardware input
5. Accurate computation of the performance impact of design choices, and its acceleration

DATA CALCULATOR ARCHITECTURE

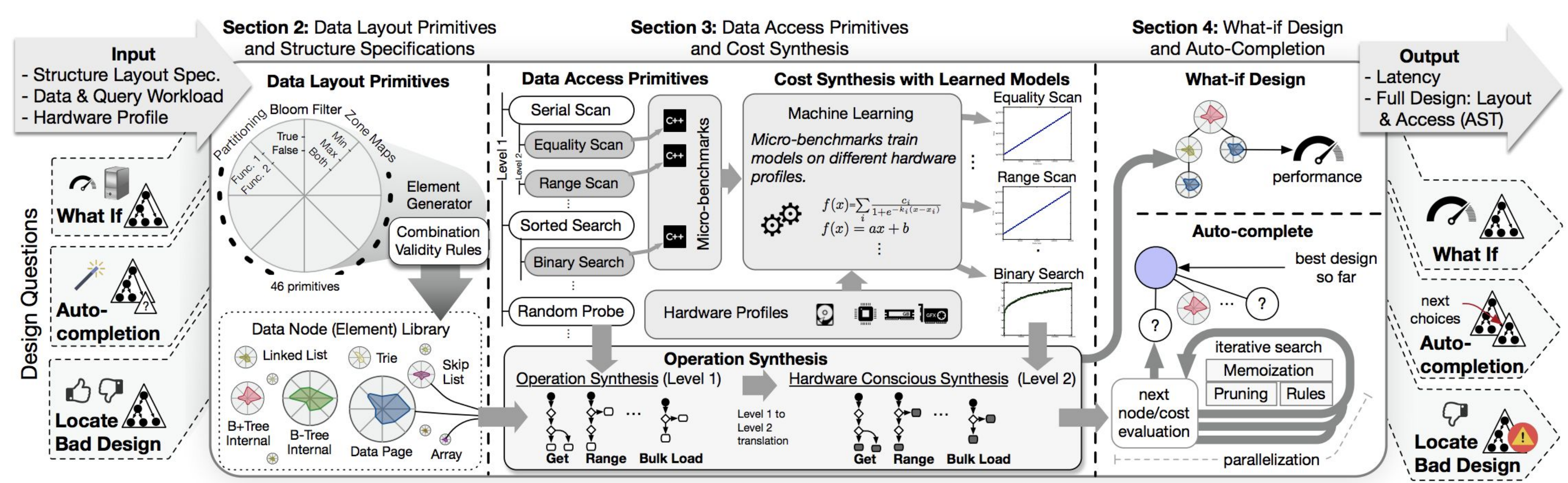


Figure 2: The architecture of the Data Calculator: From high-level layout specifications to performance cost calculation.

Image used from Page 3 of the paper 'Data Calculator'

Step 1: Design Synthesis from First Principles

- Library of fine-grained data layout primitives
- New designs formed by combining fundamental concepts in arbitrary ways
- Helps find the first principles using which all data structures can be designed

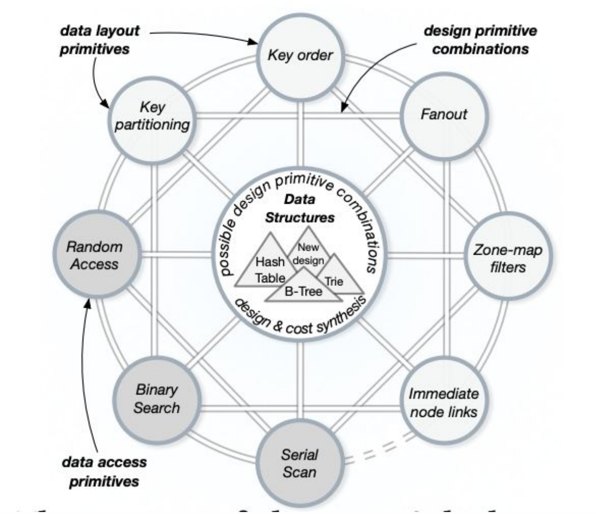


Image used from Page 1 of the paper 'Data Calculator'

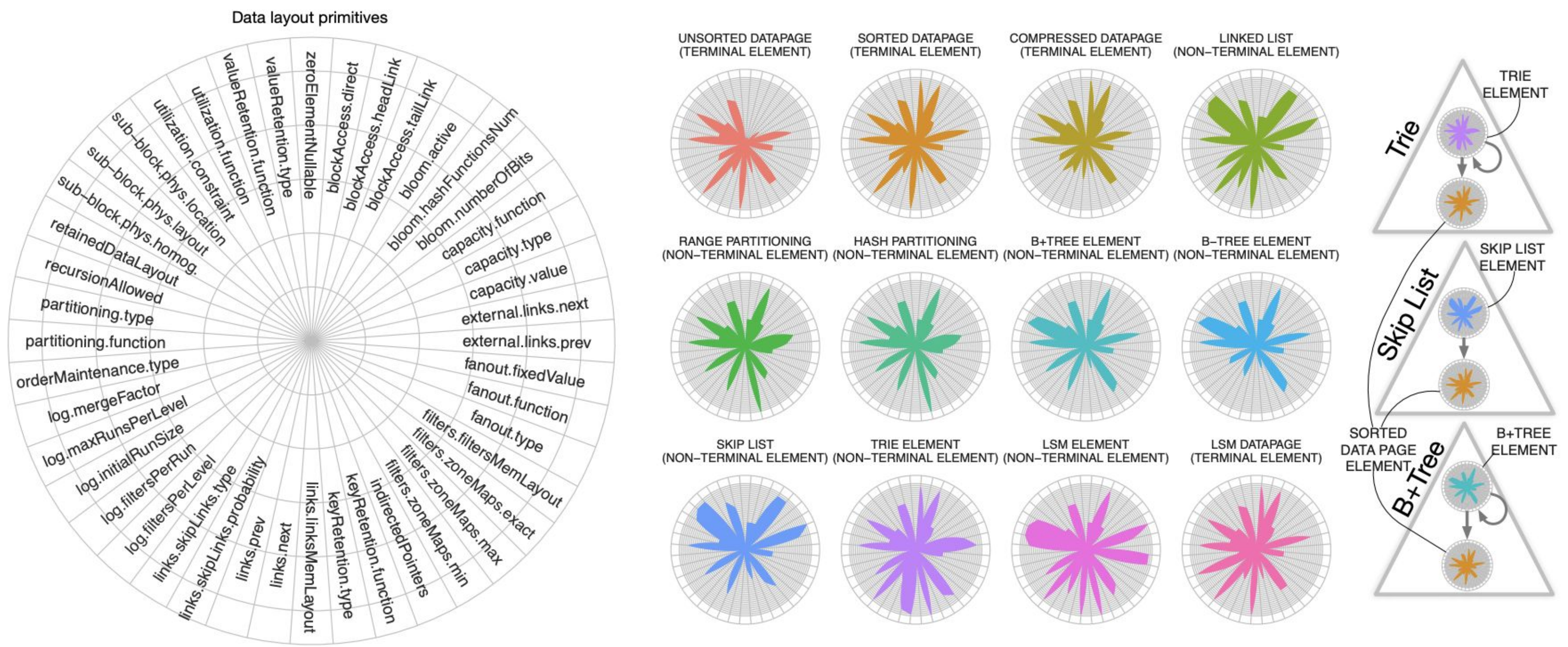


Figure 3: The data layout primitives and examples of synthesizing node layouts of state-of-the-art data structures.

Image used from Page 5 of the paper 'Data Calculator'

STRUCTURE SPECIFICATIONS

- Elements 'without data'
  - E.g. linked-lists, skip-lists
  - Flat data structures without an indexing layer
  - Not an issue since the algorithm is a model that doesn't deal with data
  - It only synthesizes a collective model on how keys should be distributed

- Recursive design through blocks
  - Block: logical portion of data divided into smaller blocks based on data structure specification
  - Elements applied recursively to blocks to construct data structure
  - Used when we test, cost, and search through multiple possible designs concurrently over the same data for a given workload and hardware

STRUCTURE SPECIFICATIONS

- Cache-conscious designs
  - Relative positioning of data structure nodes critical to overall cost for traversal
  - Data Calculator design space allows to dictate how nodes should be positioned explicitly
  - This makes it possible to fit more data in internal nodes

- Size of the Design Space
  - Design space is very large if we consider possible node elements and their combinations
  - For polymorphic structures, possible design space grow more quickly
  - Data structure design is still a wide-open space with numerous opportunities for innovative designs as data keeps growing, application workloads keep changing, and hardware keeps evolving

Step 2: Learned Primitive Access Models

- Library of data access primitives that can be combined to generate operation designs
- Operation synthesis at Level 1, Hardware conscious synthesis at Level 2
- Micro-benchmarks train machine learning models on different hardware profiles
- Synthesizer computes design of operations and latency for given inputs

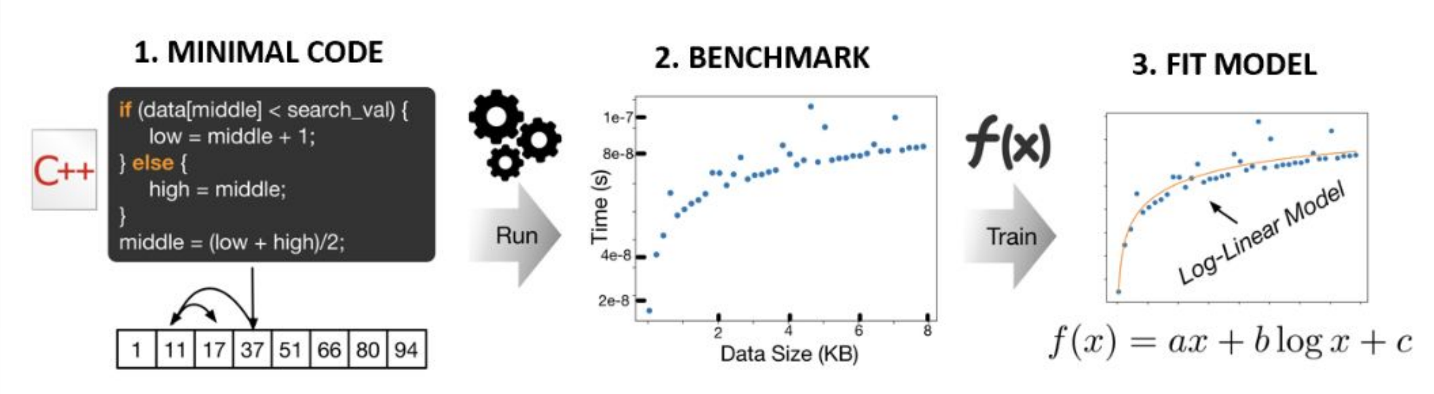


Image Source: http://daslab.seas.harvard.edu/datacalculator

Step 3: Algorithm and Cost Synthesis

- For each algorithm in workload, exact algorithm is synthesized
- Cost for target hardware using an expert system is also synthesized
- Based on layout specification of each data structure node in the path of operation, best access pattern and expected cost is decided based on the learned models

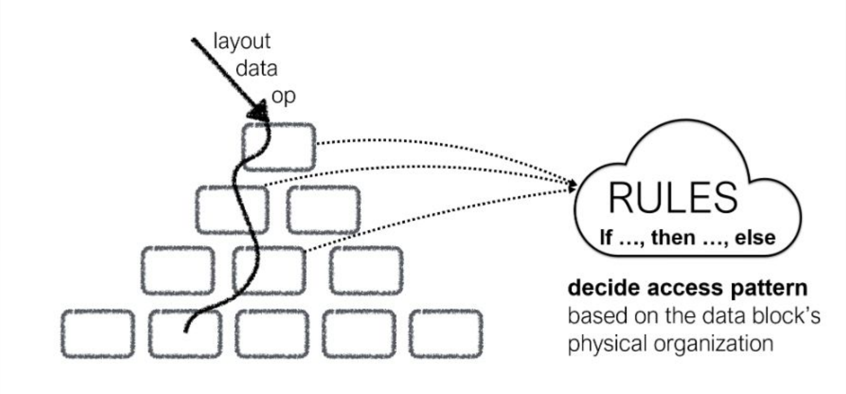


Image Source: http://daslab.seas.harvard.edu/datacalculator

EXAMPLE: BINARY SEARCH MODEL

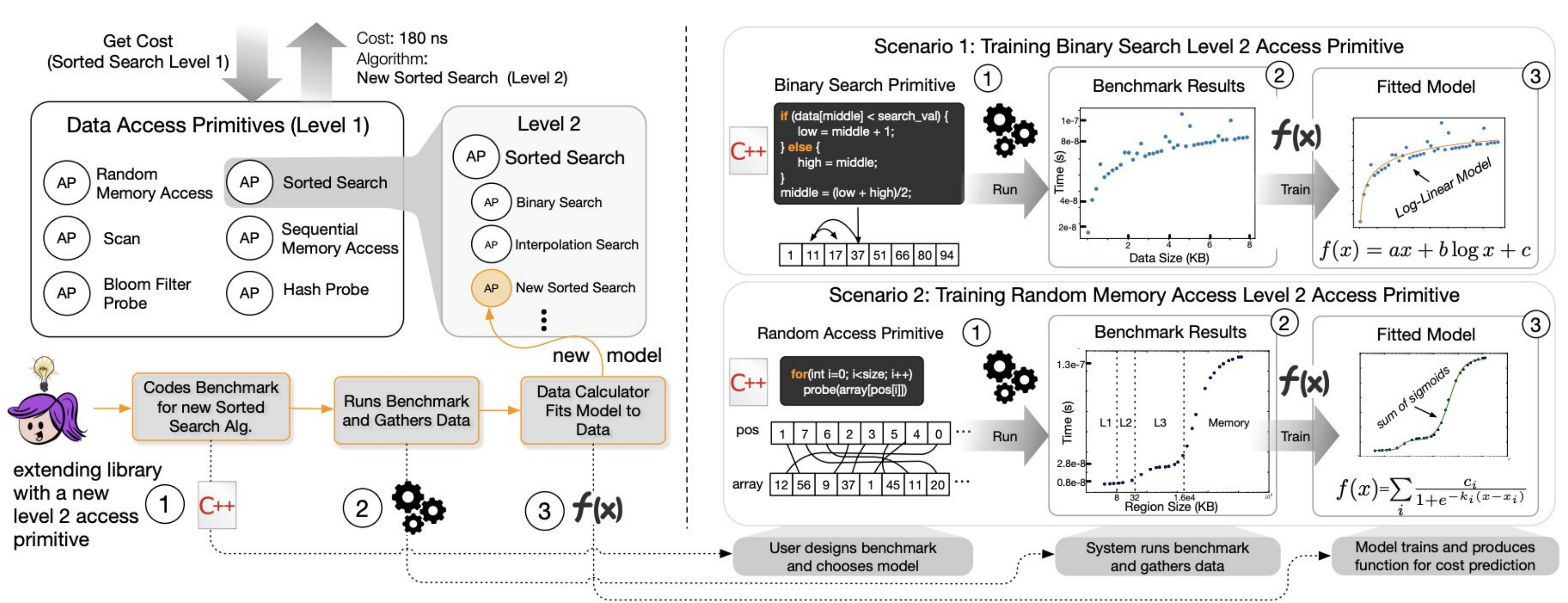


Figure 4: Training and fitting models for Level 2 access primitives and extending the Data Calculator.

EXAMPLE: DICTIONARY OPERATION GET

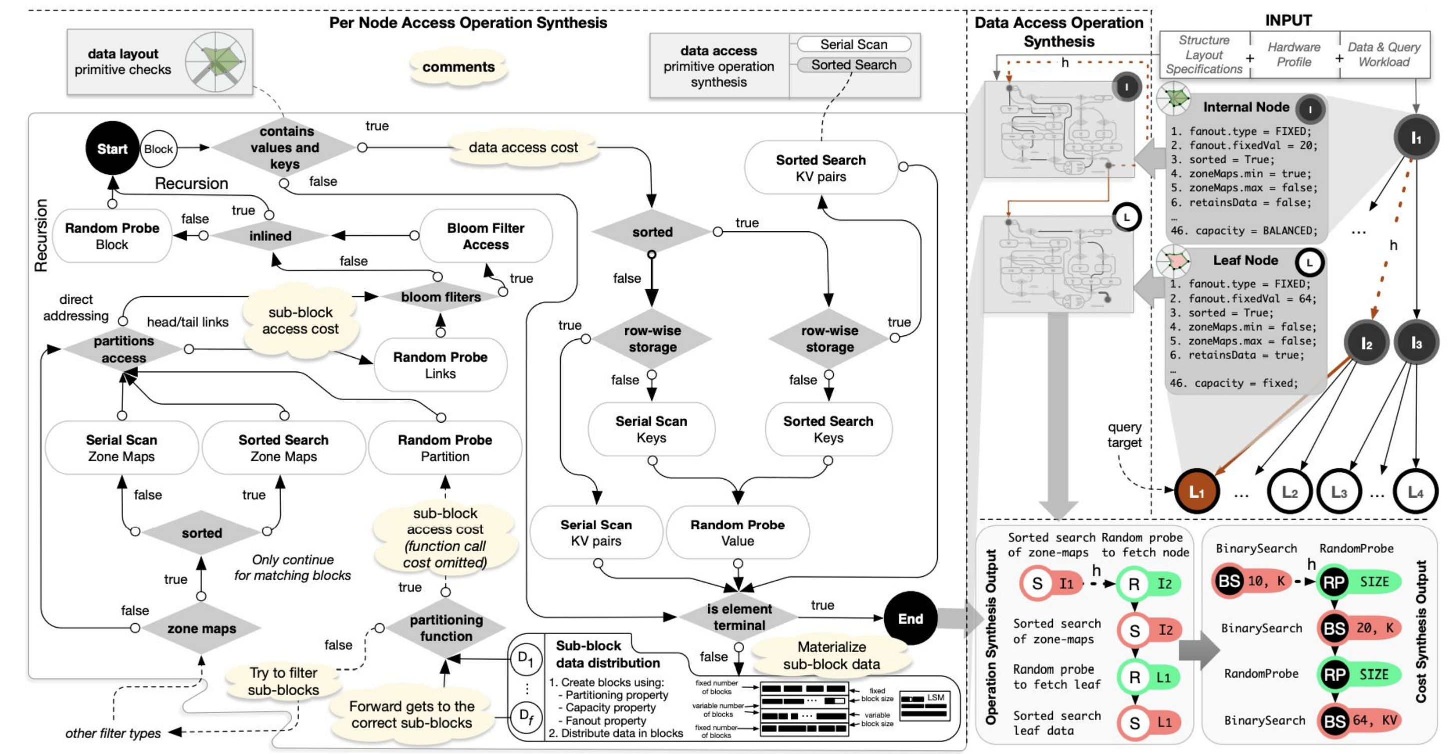


Figure 5: Synthesizing the operation and cost for dictionary operation Get, given a data structure specification.

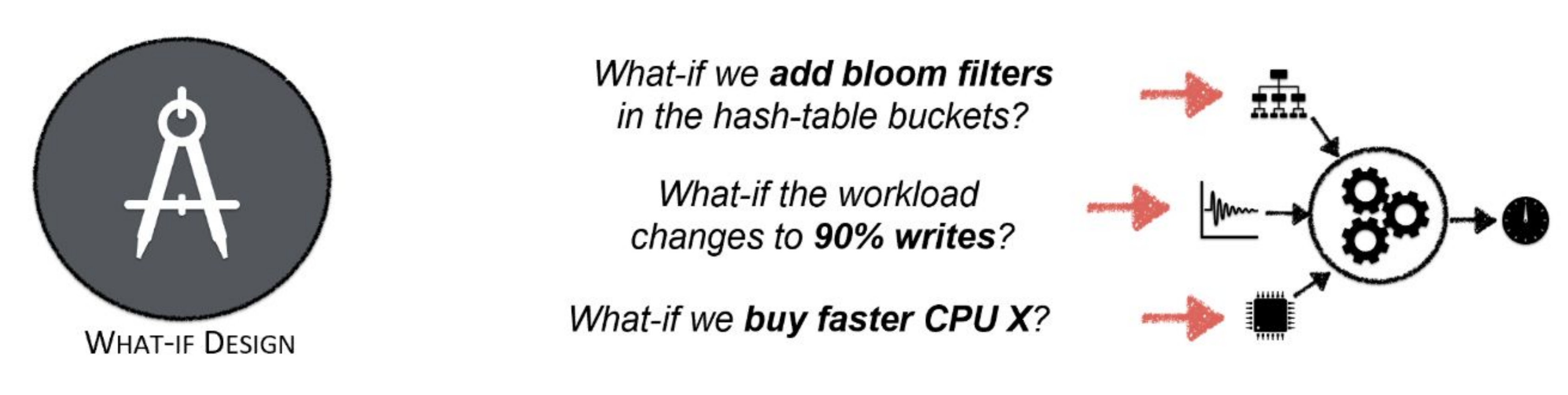
THE PERIODIC TABLE OF DATA STRUCTURES

classes of designs / *classes of primitives*

	B-trees & Variants	Tries & Variants	LSM-Trees & Variants	Differential Files	Membership Tests	Zone maps & Variants	Bitmaps & Variants	Hashing	Base Data & Columns	
Partitioning	DONE	DONE	DONE					DONE	DONE	↓↑↑ R U M
Logarithmic Design	DONE	DONE	DONE							↓↓↑ R U M
Fractional Cascading	DONE		DONE	DONE						↓↑↑ R U M
Log-Structured	DONE		DONE	DONE						↑↓↑ R U M
Buffering	DONE			DONE			DONE			↓↕↑ R U M
Differential Updates	DONE			DONE						↑↓↓ R U M
Sparse Indexing	DONE				DONE	DONE				↓↕↑ R U M
Adaptivity	DONE								DONE	

WHAT-IF DESIGN

Iteratively test different combinations of design/workload/hardware



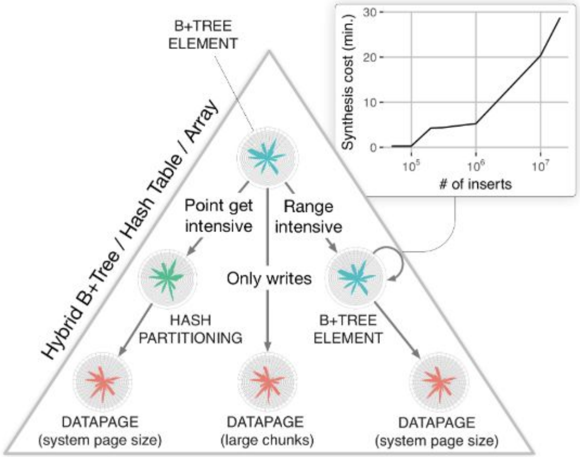
WHAT-IF DESIGN

- Let users form design questions by varying any one input parameter
- Input
  1. High level specifications of existing design
  2. Cost with original design
  3. Cost with bloom filter variation
- Benefits
  1. Quickly test variations of data structure designs simply by altering a high level specification, without having to implement, debug, and test a new design
  2. A given specification can be tested quickly on alternative environments without having to actually deploy code to this new environment

AUTO-COMPLETION

Automatically identify “the best design possible” to match a workload and hardware

AUTO-DESIGN



AUTO-COMPLETION

- Complete partial layout specifications given a workload, and a hardware profile
- Input

1. Partial layout specification
2. Data
3. Queries
4. Hardware
5. List of candidate elements

AUTO-COMPLETION

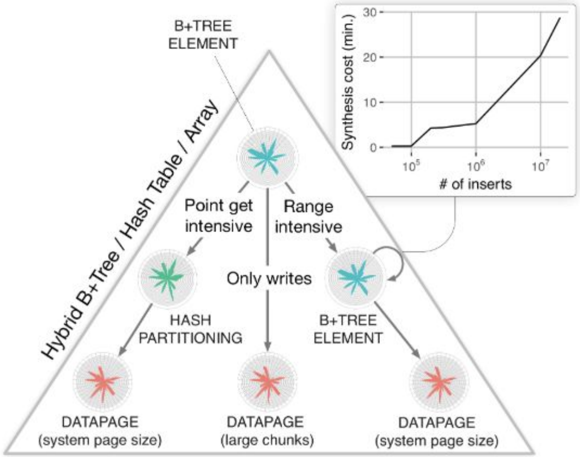
PROCESS

- Start at the last ‘known’ point, compute the rest of the missing subtree of the hierarchy of elements
- At each step consider a new element as candidate for one of the nodes of the missing subtree, compute the cost for the different kinds of dictionary operations present in the workload
- Design kept only if it is better than all previous ones
- Use a cache to remember specifications and their costs to avoid recomputation

SELF-DESIGNING SYSTEM

Utilize design continuums and cross design spaces

AUTO-DESIGN



EXPERIMENTAL ANALYSIS

(1) Implementation

- Core implementation in C++
- Separate module in Python made available for analyzing benchmark results
- Learning process gets done each time we include a new hardware
- Learned coefficients for each model passed to the C++ back-end to be used for cost synthesis during design questions

(2) Accurate Cost Synthesis

- Manually written DS specifications for 8 access methods
- Data Calculator generated design of operations and computed latency for each workload
- Verified results against actual implementation
- Learned coefficients for each model passed to the C++ back-end to be used for cost synthesis during design questions

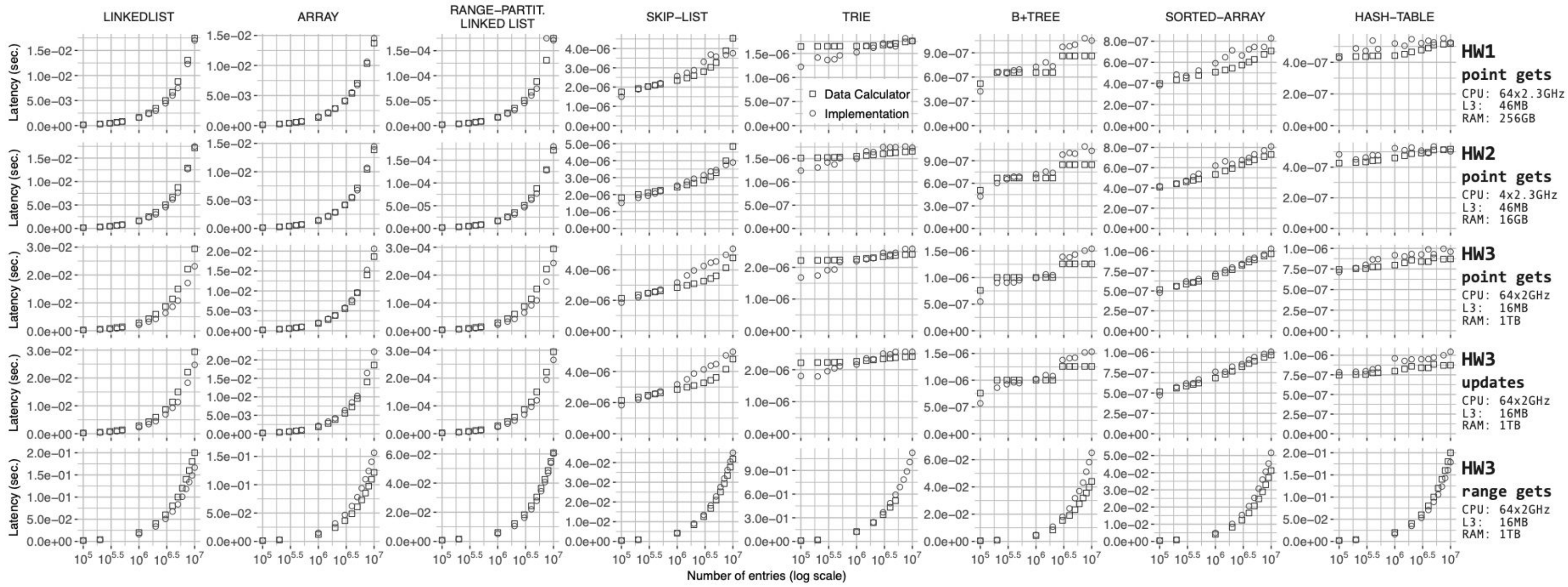


Figure 6: The Data Calculator can accurately compute the latency of arbitrary data structure designs across a diverse set of hardware and for diverse dictionary operations.

EXPERIMENTAL ANALYSIS

(3) Diverse Machines and Operations

- Performance tested with different hardware (in terms of both CPU and memory properties)
- Updates are changes to the value of a key-value pair i.e. a point query with an additional write access

(4) Training Access Primitives

- Inexpensive process that takes just a few minutes

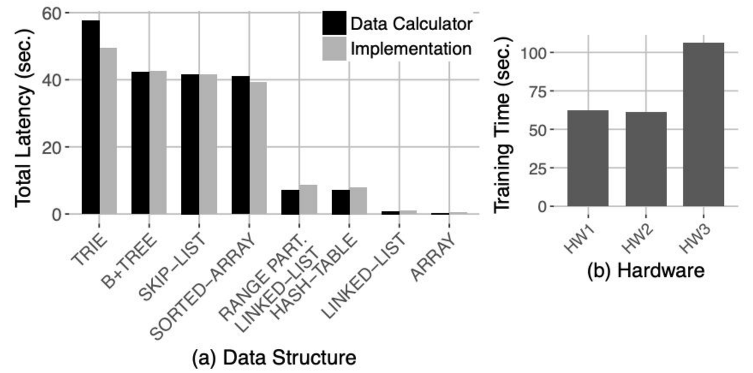


Figure 7: Computing Bulk-loading cost (left) and Training cost across diverse hardware (right).

EXPERIMENTAL ANALYSIS

(5) Cache Conscious Designs and Skew

- Use of a cache-conscious design, Cache Conscious B+ tree (CSB)
- Captures caching effects of growing data sizes and design patterns where the relative position of nodes affects tree traversal costs
- Use of Zipfian distribution and skewed data improves performance

(6) Rich Design Questions

- Capable of handling both point and range queries
- Takes seconds to evaluate new hardware or workload
- Capable of suggesting alternative designs better suitable for the task considering cost and scalability

RELATED WORK

- Interactive design
- Generalized indexes
- Modular/Extensible systems and System synthesizers
- Auto-tuning
- Adaptive systems
- Data representation systems

SUMMARY AND NEXT STEPS

- 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
- The design space presented here includes basic layout primitives and primitives that enable cache conscious designs by dictating the relative positioning of nodes, focusing on read only queries.
- Future steps:

1. Find primitives for additional significant design classes
2. Innovations for cost synthesis
3. Machine learning algorithms capable of searching the whole design space

DISCUSSION

DISCUSSION

- Use of parametric models
  - How does it help?
- L1, L2, L3 cache
  - Was the experiment necessary?
  - Is it incomplete?
- Hardware periodic table
- Comparison to other papers we read
  - reduction of a complex state space or computational pipeline into a series of component primitives
- Useful in teaching data structures and algorithms!

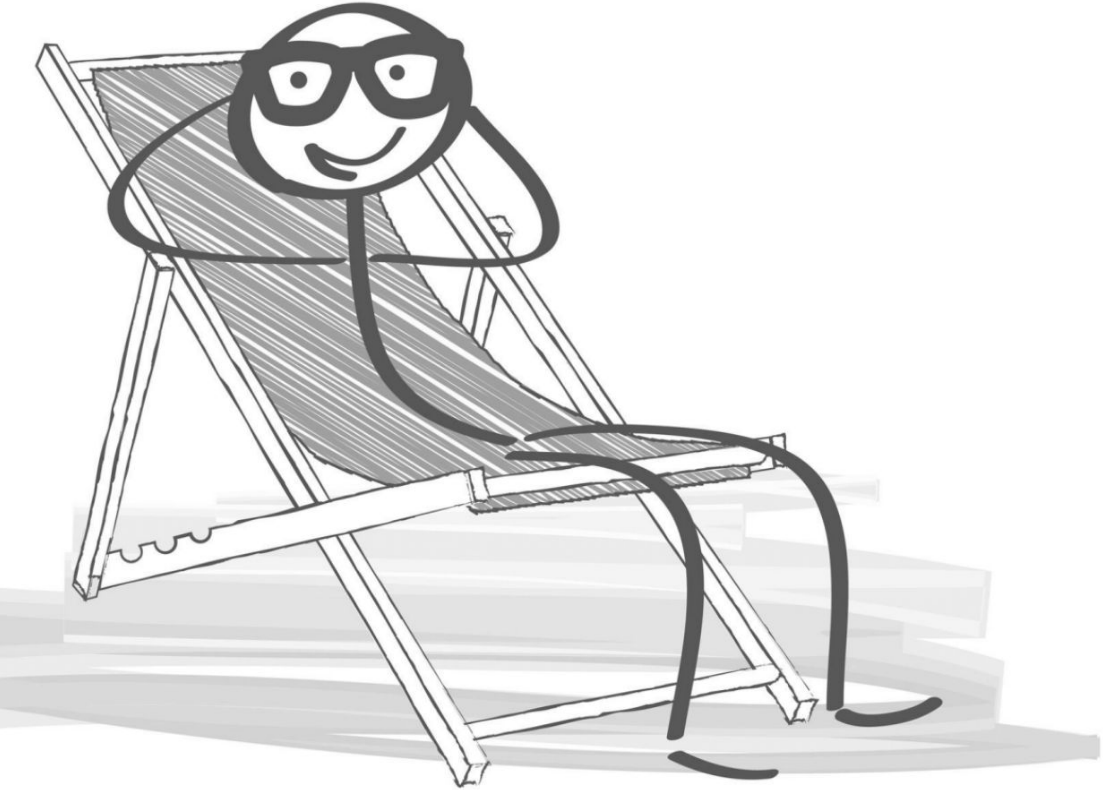
STRENGTHS

- First work that deals with interactive data structure design to compute the impact on performance
- Aids developers in exploring different possible configurations without increasing complexity when designing data structures
- Interesting comparison drawn between elements of periodic table and fundamental principles of data structures
- Use of parametric models for cost prediction
- Support for what-if design queries
- Focus on different classes of possible data structure designs

WEAKNESSES

- Lacks discussion on how to map data structures to their primitives
- Actual performance estimation doesn't work for individual queries since it is hard to precisely estimate certain access primitives without running them
- Current work focuses only on simple updates, while complex ones involving restructuring are left out
- Empirical study focuses only on reducing time complexity. What about space complexity?

IS THIS GOING TO REPLACE RESEARCHERS/ENGINEERS?



Did the arithmetic calculator replace mathematicians?