The Data Calculator

Authors: Stratos Idreos, Kostas Zoumpatianos, Brian Hentschel, Michael S. Kester, Demi Guo

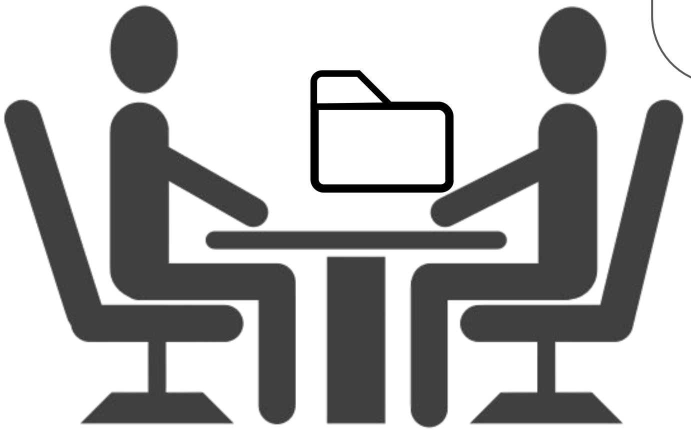
Presented by: Minjie Tang, Alec Gallardo, Ge Gao

Once Upon A Time…

What would be the <u>best data structure</u> to store this data?

We need more <u>information</u>!

I have this workload with **<u>80% read</u>** and **<u>20% write</u>**.
How can we design a database for it?



We need more **information**!

How **large** is the dataset?

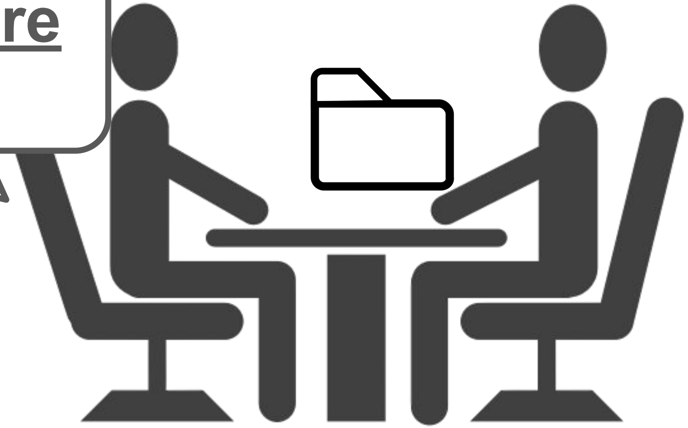
Is data **sorted** or **unsorted**?

What type of **queries** will be perform

Will there be **delete**?

Will workload **grow** in the future?

What are the **hardware** constraints?



I have this workload with **80% read** and **20% write**…

We need more information!

How la

Data layout

Is data **sorted** or **unsorted**?

What

forr

Workload information

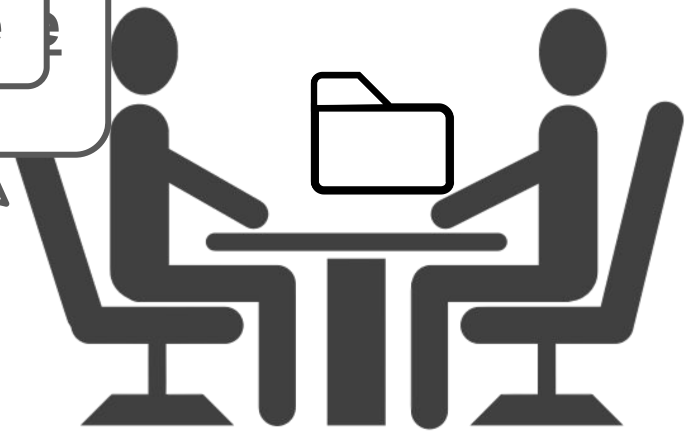
Will there be **delete**?

Will workload **grow** in the future?

W

Hardware Profile

constraints?



I have this workload with **80% read** and **20% write**…

Design Questions

Can we build upon a simple data structure?

Can we tune it based on our needs

Or build a new one from scratch?

How can we take advantage of the hardware?

What if we add a Bloom filter? **What if** the workload shifts? **What if…**

Design Questions

Can we build upon a simple data structure?

Can we tune it based on our needs

Or build a new one from scratch?

How can we take advantage of the hardware?

What if we add a Bloom filter?**What if** the workload shifts?**What if…**

We wouldn’t be able to implement them all!

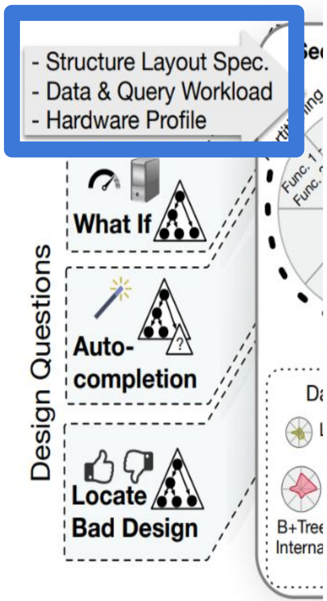


Figure 2: The arch

We know these

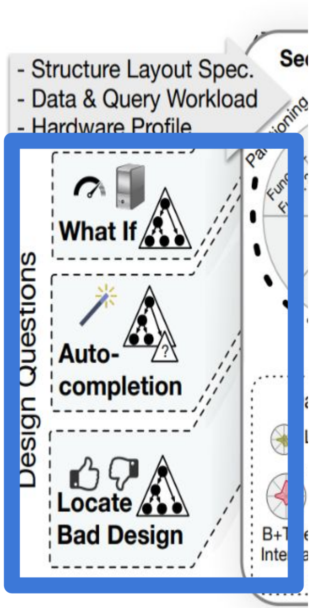
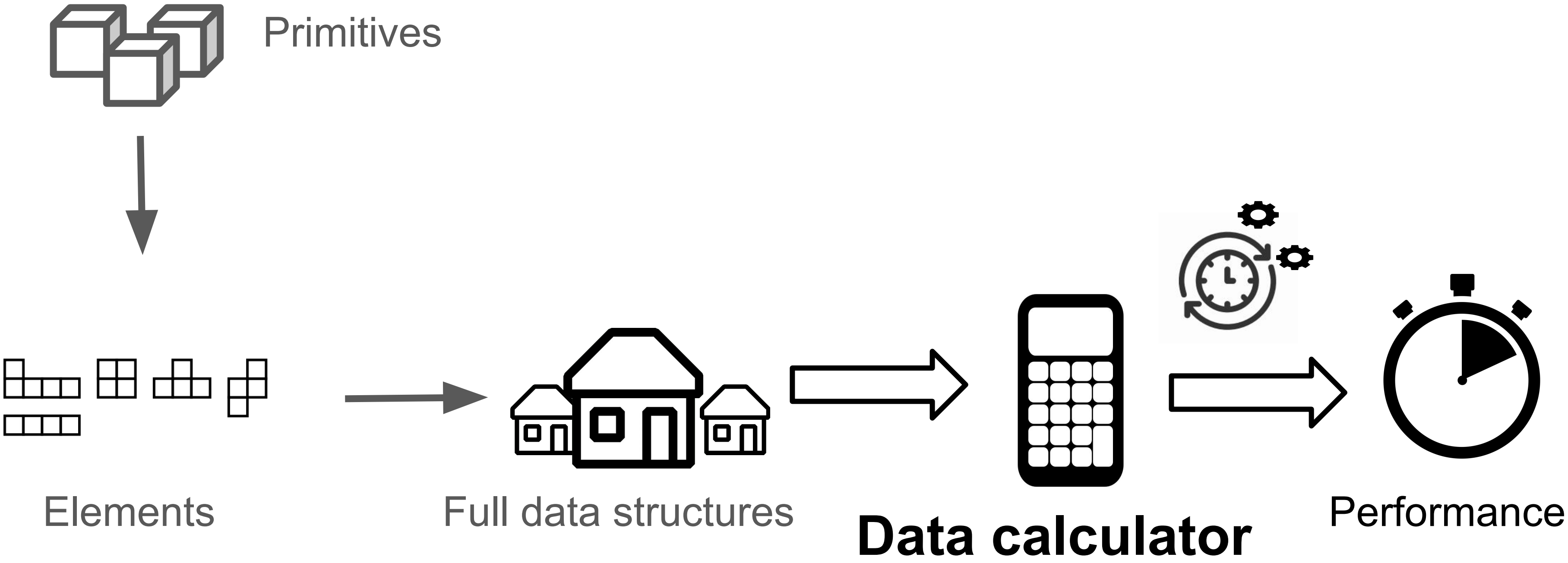


Figure 2: The arch

Now we want to answer these

The Data Calculator

The Data Calculator



Using Data Calculator

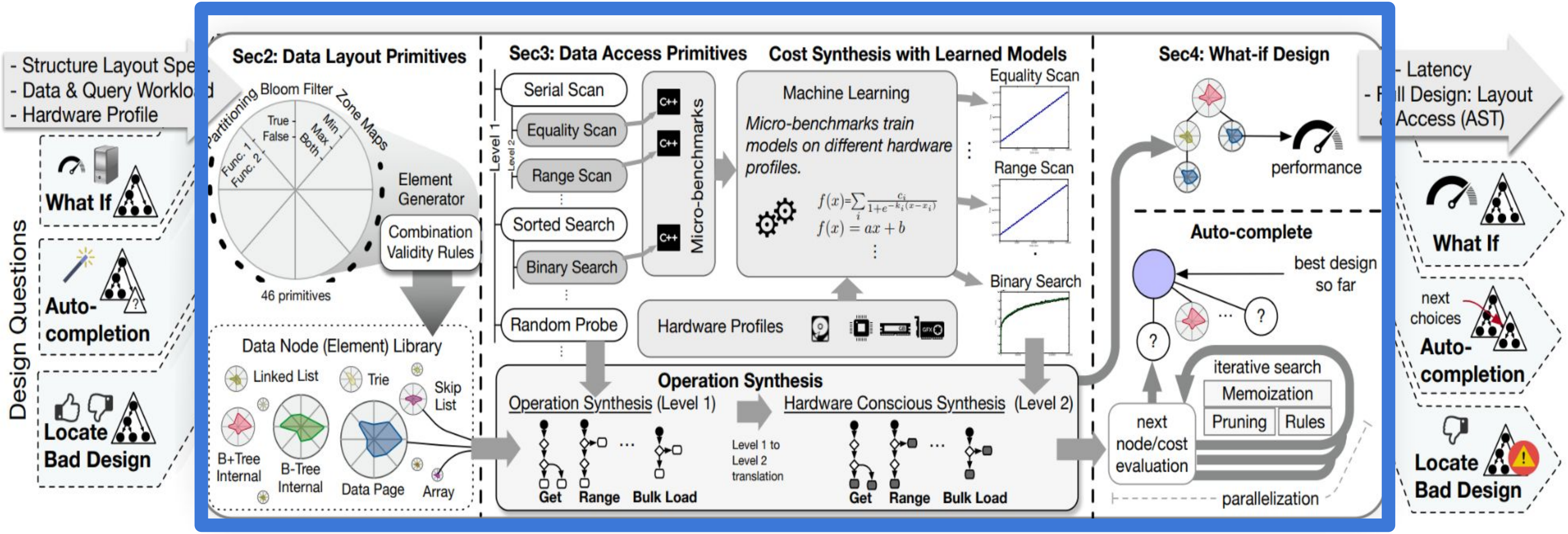
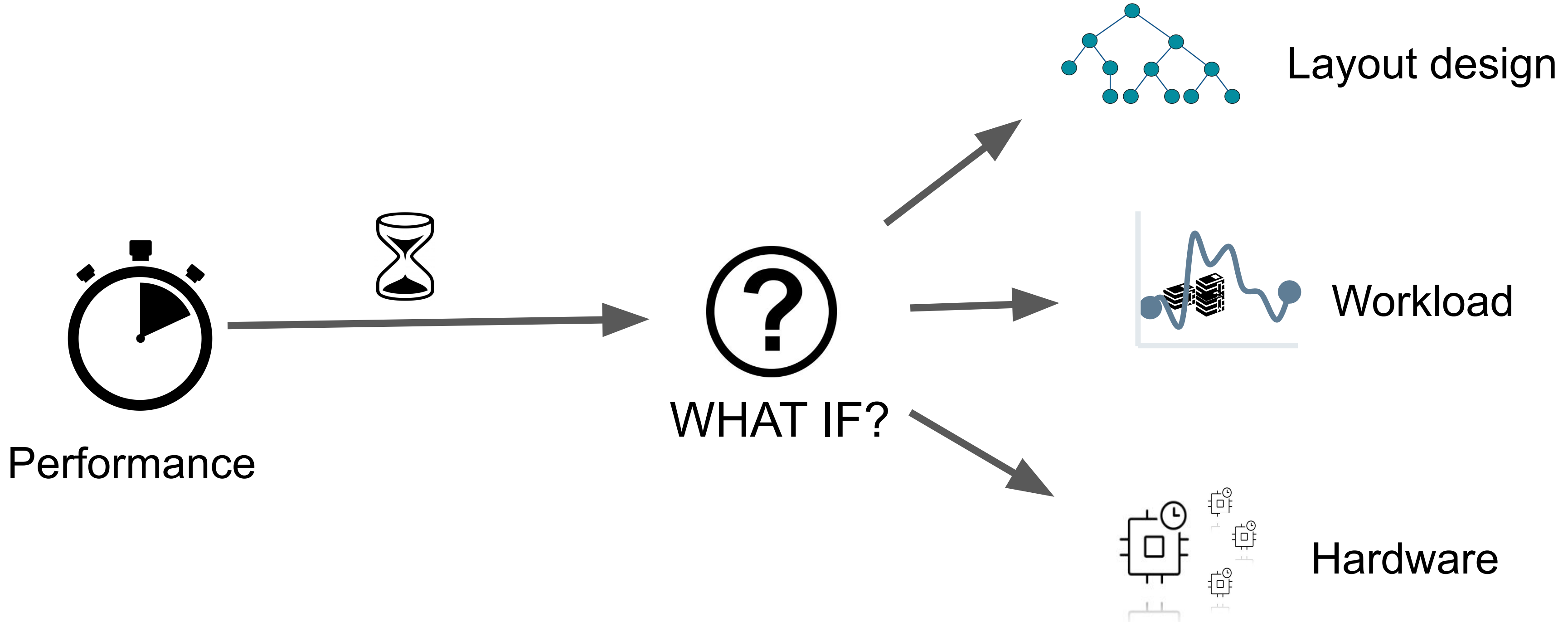


Figure 2: The architecture of the Data Calculator: From high-level layout specifications to performance cost calculation.

The Data Calculator



Based on predicted performance, we can explore endless possibilities **—without implementing them!**

Using Data Calculator

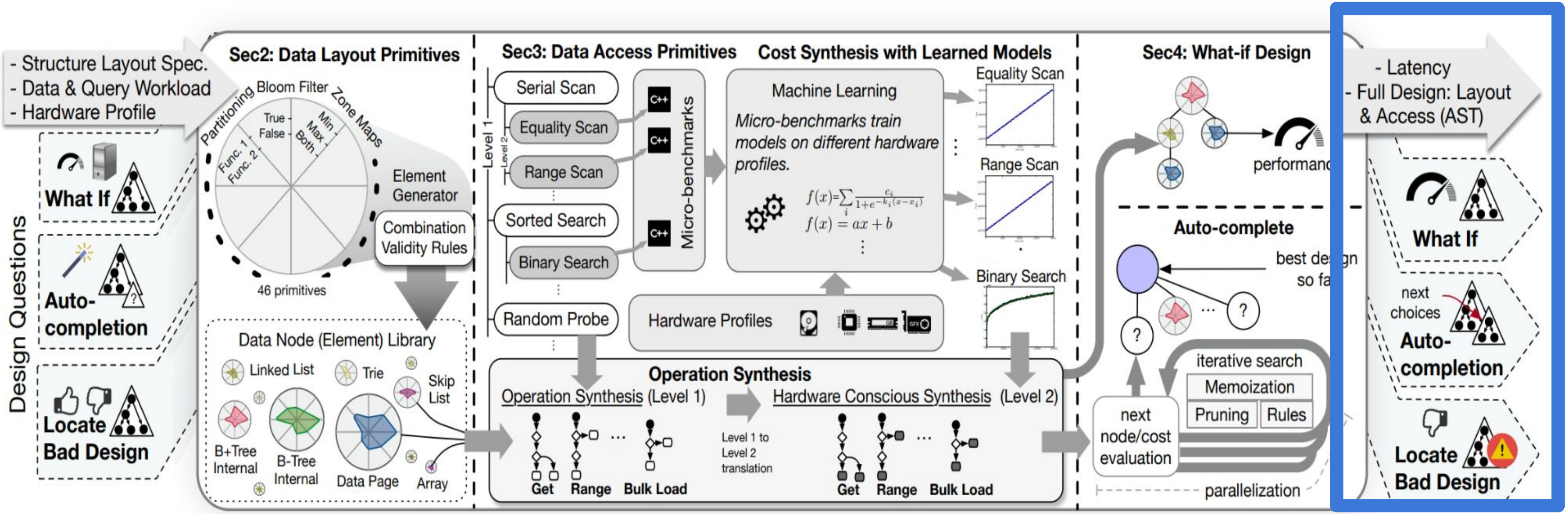
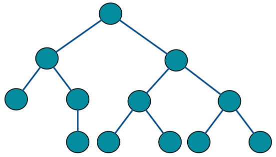
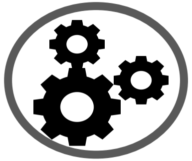


Figure 2: The architecture of the Data Calculator: From high-level layout specifications to performance cost calculation.

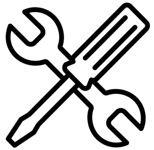
The Data Calculator



Layout Primitives

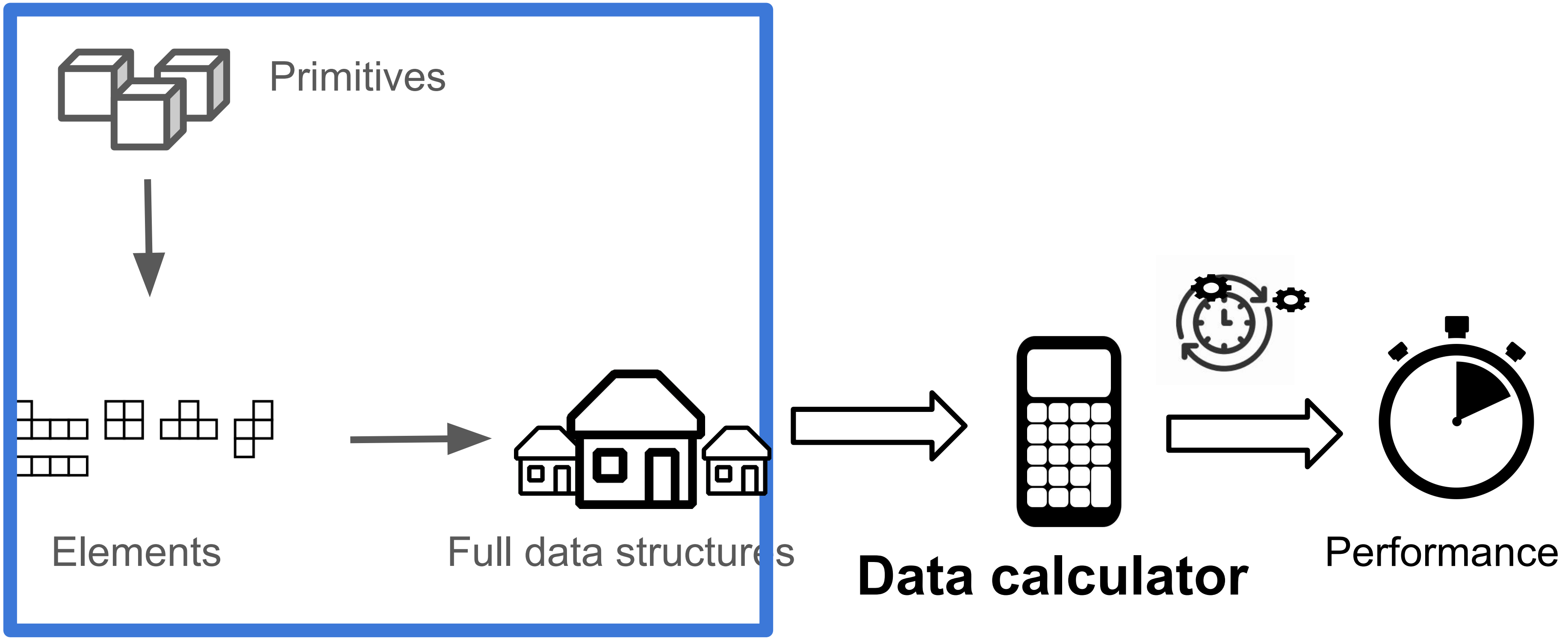


Access Primitives
&
Cost Synthesis

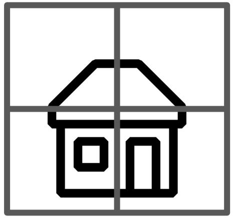


Usage
&
Experiment

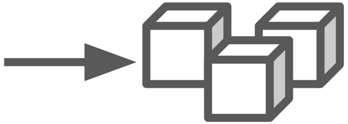
The Data Calculator



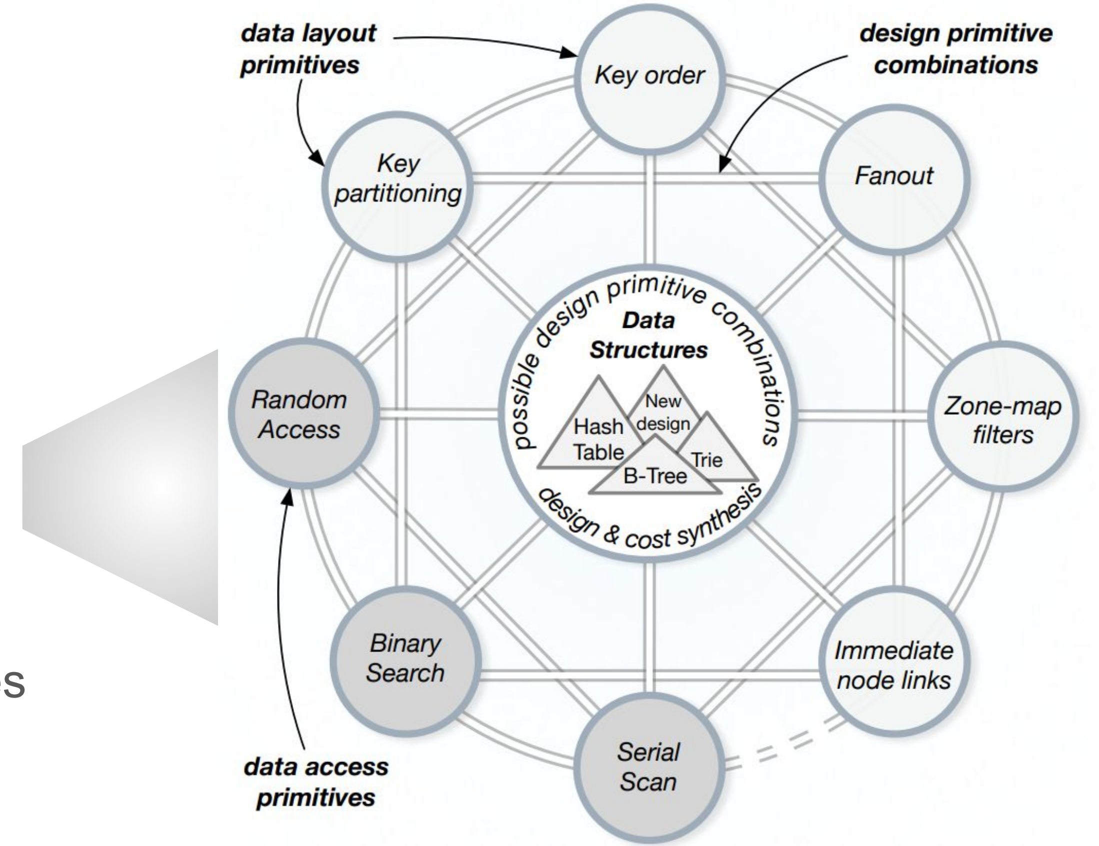
Primitives



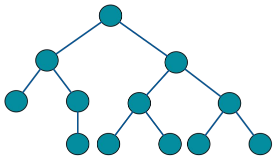
Existing
Data Structures



Primitives



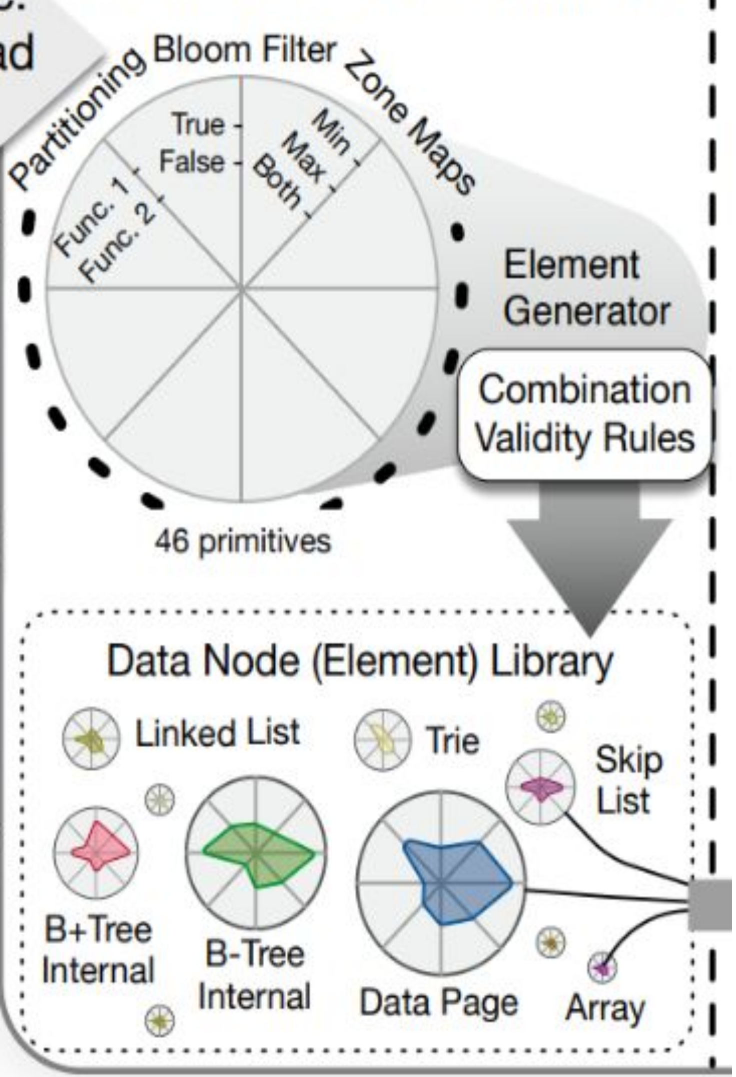
The Data Calculator



Layout Primitives

Access Primitives
&
Cost Synthesis

Usage
&
Experiment



Data Layout Primitives

- defines aspects of a data structure's layout

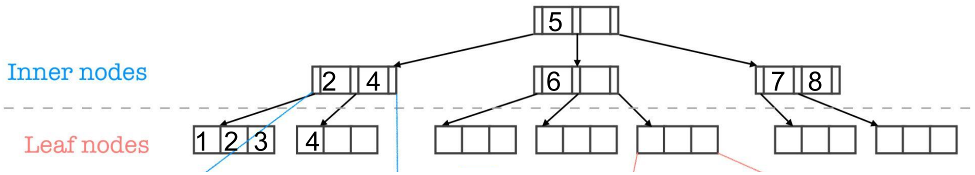
node organization

partitioning

physical placement

metadata

Data Layout Primitives - B+ Tree



Key order: Sorted

Sub-block physical layout: BFS

Key retention: None (Inner nodes),

Yes(Leaf nodes)

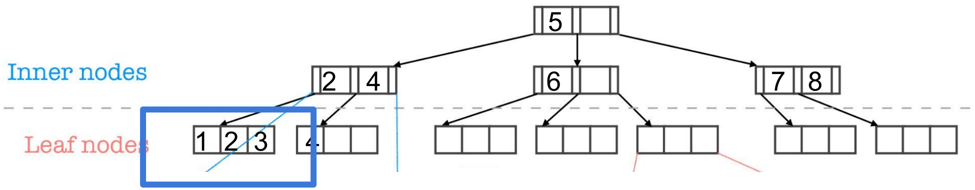
Data Layout Primitives - B+ Tree

Key order: Sorted

Sub-block physical layout: BFS

Key retention: None (Inner nodes),

Yes(Leaf nodes)



Memory layout:

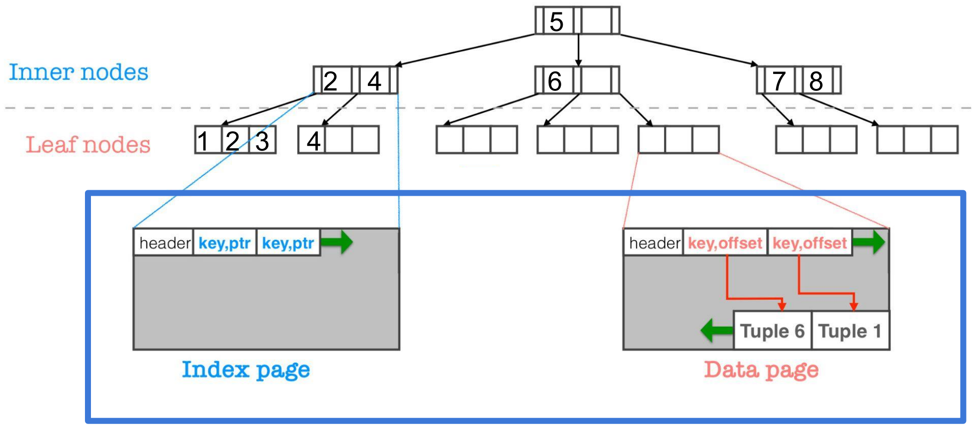
1	2	3

Data Layout Primitives - B+ Tree

Key order: Sorted

Sub-block physical layout: BFS

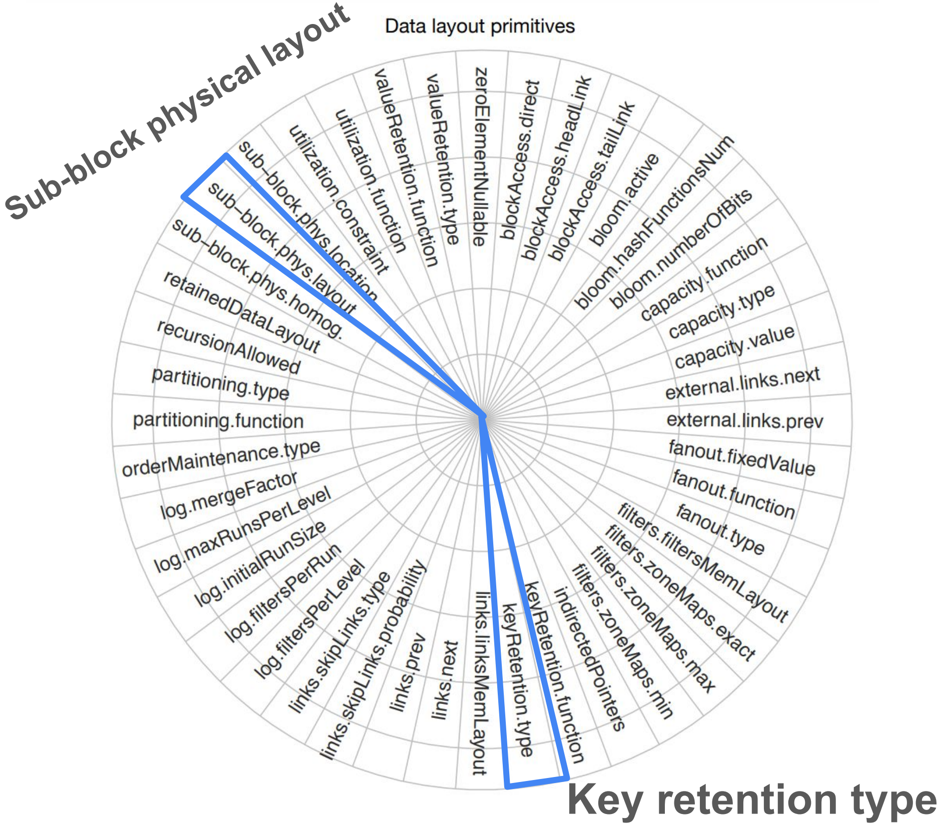
Key retention: None (Inner nodes), Yes(Leaf nodes)



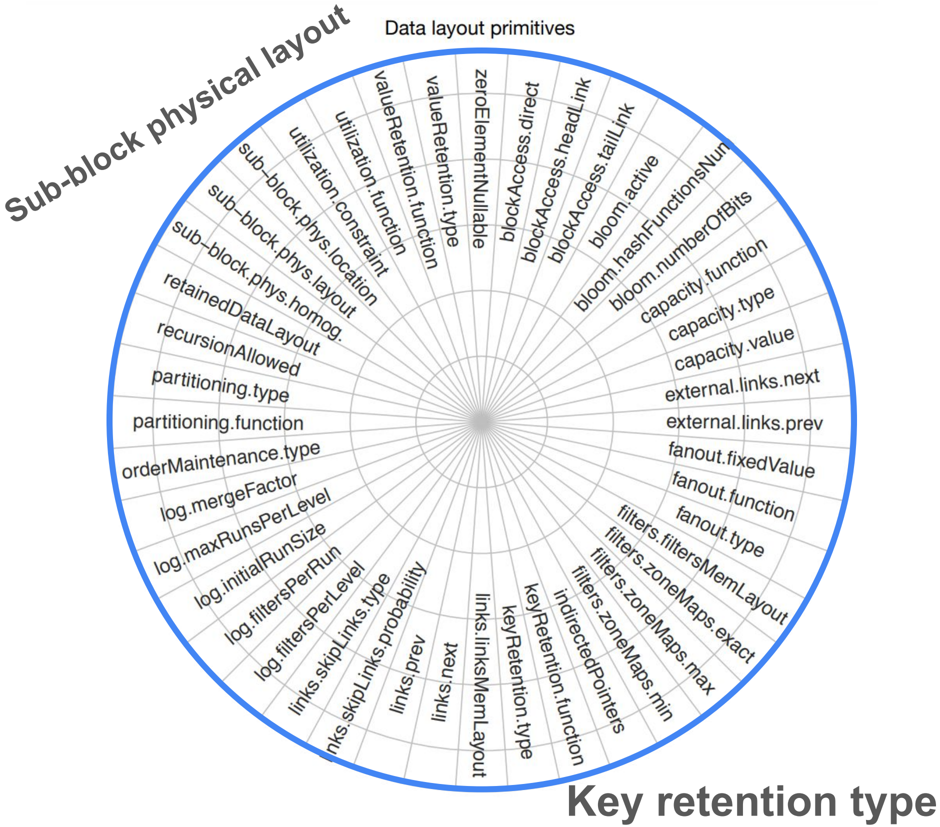
Memory layout:

1	2	3

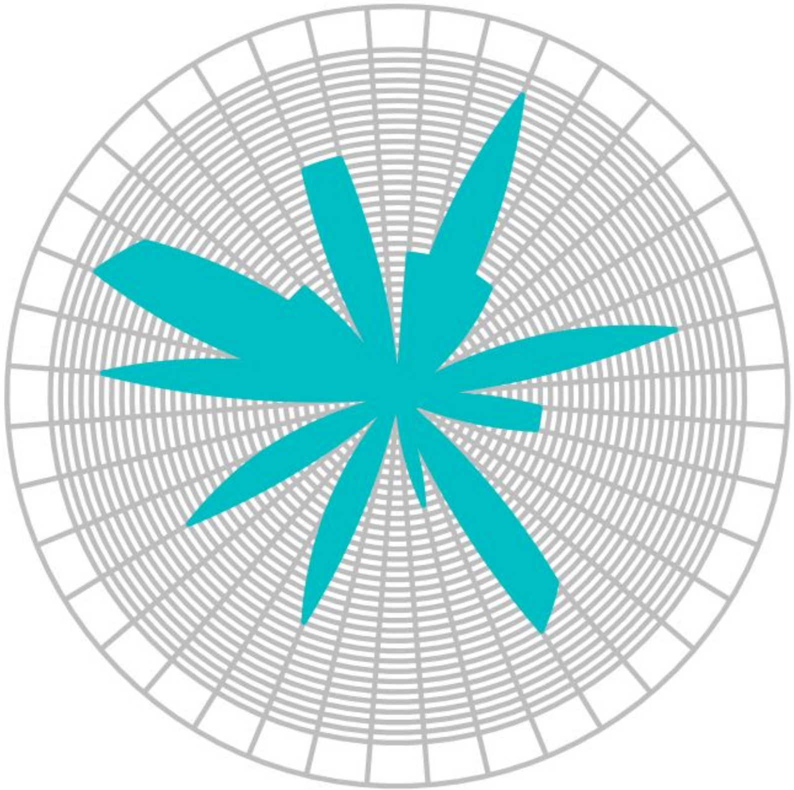
B+ Tree Elements



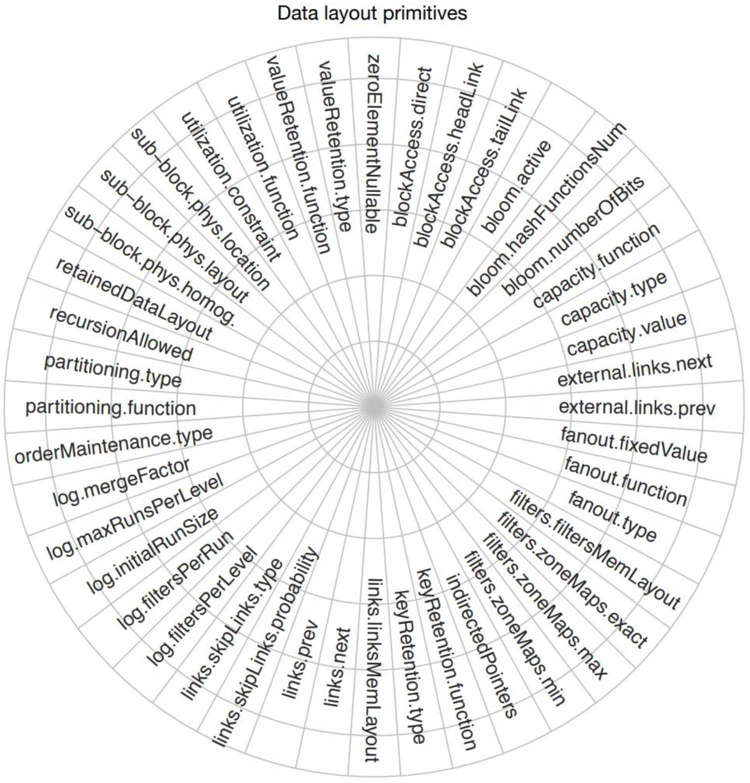
B+ Tree Elements

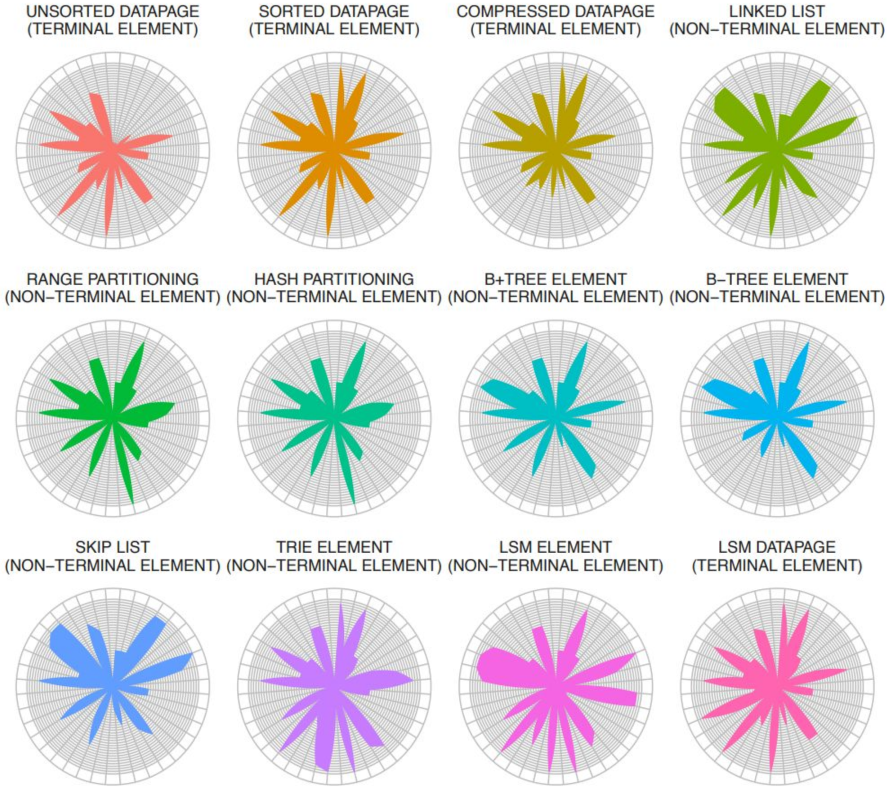


B+TREE ELEMENT (NON–TERMINAL ELEMENT)



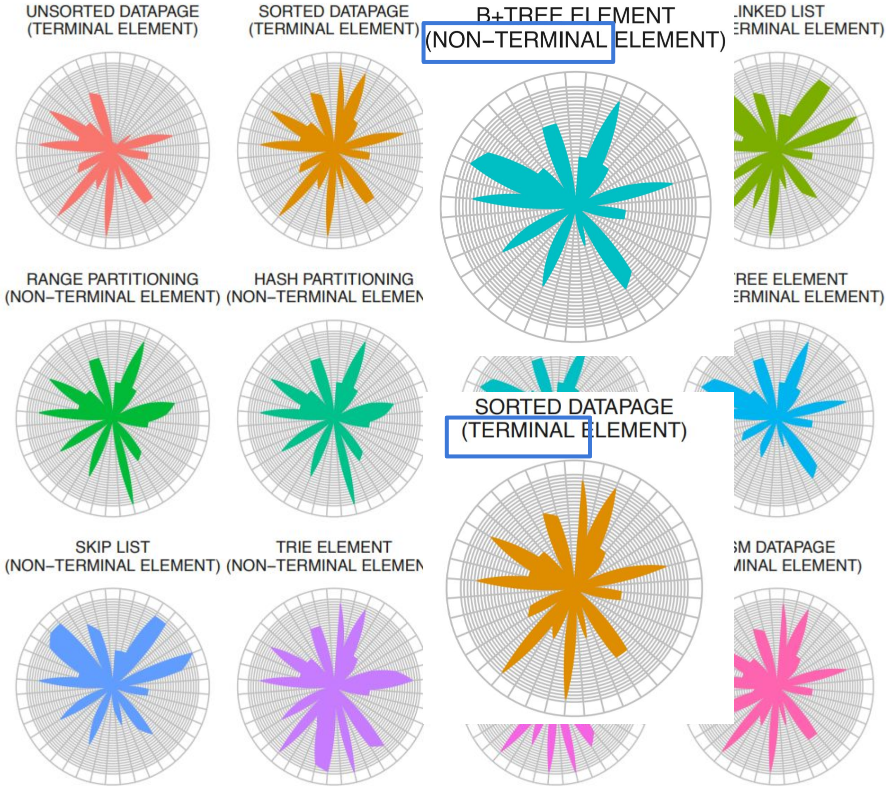
Elements


Data layout primitives
zeroElementNullable
blockAccess.direct
blockAccess.headLink
blockAccess.tailLink
bloom.active
bloom.hashFunctionsNum
bloom.numberOfBits
capacity.function
capacity.type
capacity.value
external.links.next
external.links.prev
fanout.fixedValue
fanout.function
fanout.type
filters.filtersMemLayout
filters.zoneMaps.exact
filters.zoneMaps.max
filters.zoneMaps.min
indirectedPointers
keyRetention.function
keyRetention.type
links.linksMemLayout
links.next
links.prev
links.skipLinks.probability
links.skipLinks.type
log.filtersPerLevel
log.filtersPerRun
log.initialRunSize
log.maxRunsPerLevel
log.mergeFactor
orderMaintenance.type
partitioning.function
partitioning.type
recursionAllowed
retainedDataLayout
sub–block.phys.homog.
sub–block.phys.layout
sub–block.phys.location
utilization.constraint
utilization.function
valueRetention.function
valueRetention.type

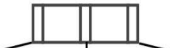

UNSORTED DATAPAGE (TERMINAL ELEMENT)
SORTED DATAPAGE (TERMINAL ELEMENT)
COMPRESSED DATAPAGE (TERMINAL ELEMENT)
LINKED LIST (NON–TERMINAL ELEMENT)
RANGE PARTITIONING (NON–TERMINAL ELEMENT)
HASH PARTITIONING (NON–TERMINAL ELEMENT)
B+TREE ELEMENT (NON–TERMINAL ELEMENT)
B–TREE ELEMENT (NON–TERMINAL ELEMENT)
SKIP LIST (NON–TERMINAL ELEMENT)
TRIE ELEMENT (NON–TERMINAL ELEMENT)
LSM ELEMENT (NON–TERMINAL ELEMENT)
LSM DATAPAGE (TERMINAL ELEMENT)

Elements

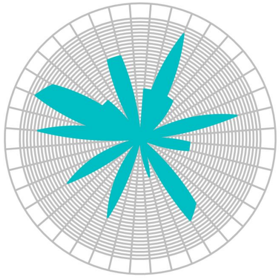
- **Element:** defining how data is stored and accessed in that node.



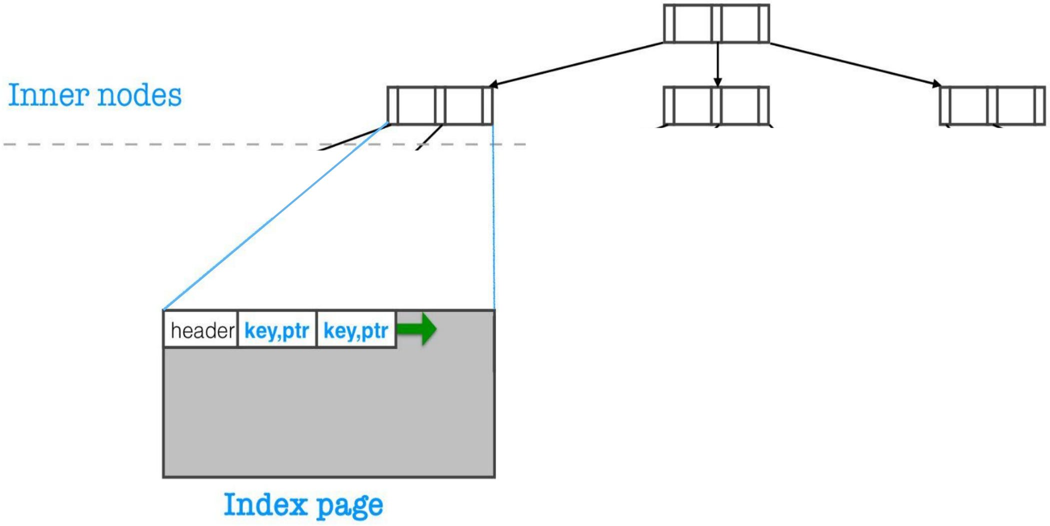
Terminal & Non-Elements B+Tree

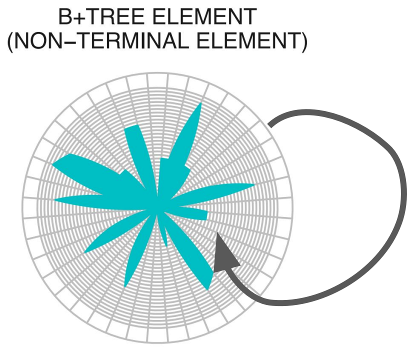


B+TREE ELEMENT
(NON–TERMINAL ELEMENT)

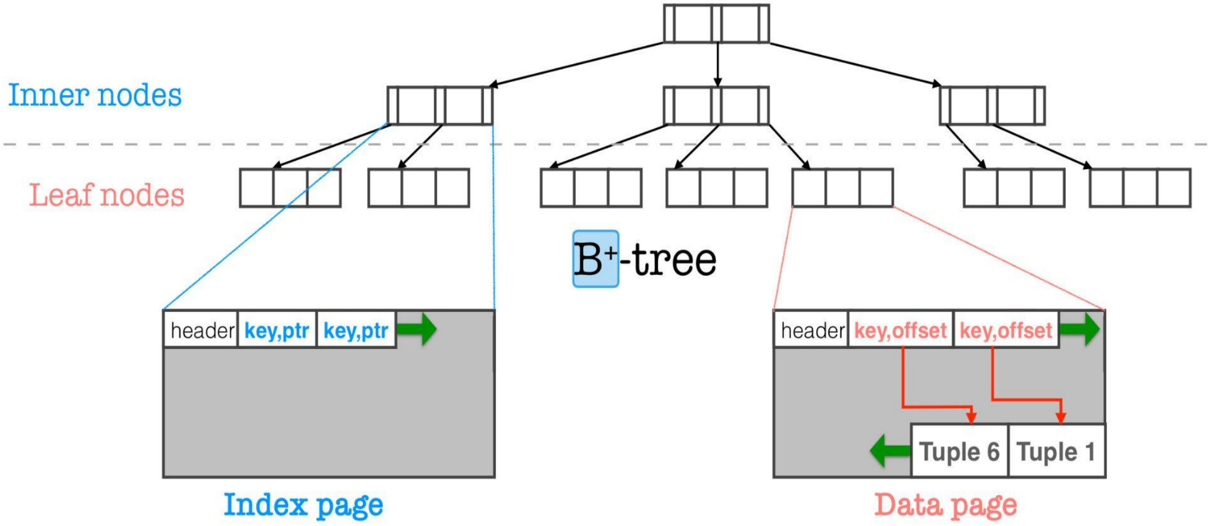


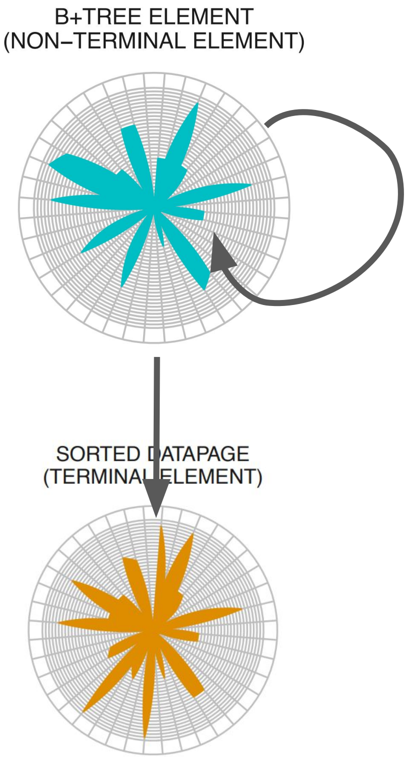
Terminal & Non-Elements B+Tree



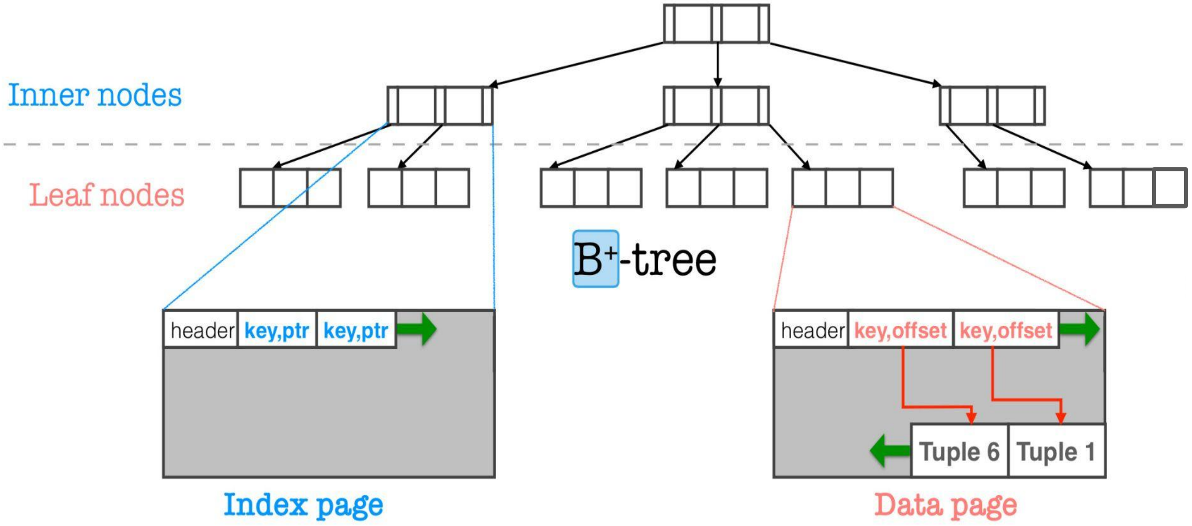


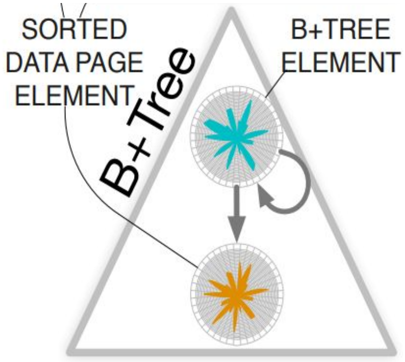
Terminal & Non-Elements B+Tree





Terminal & Non-Elements B+Tree

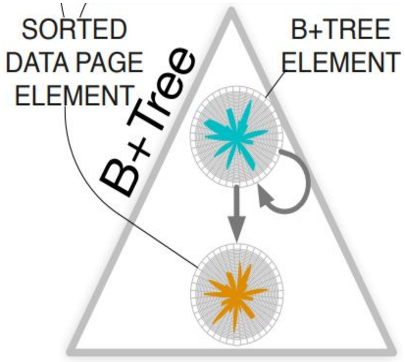




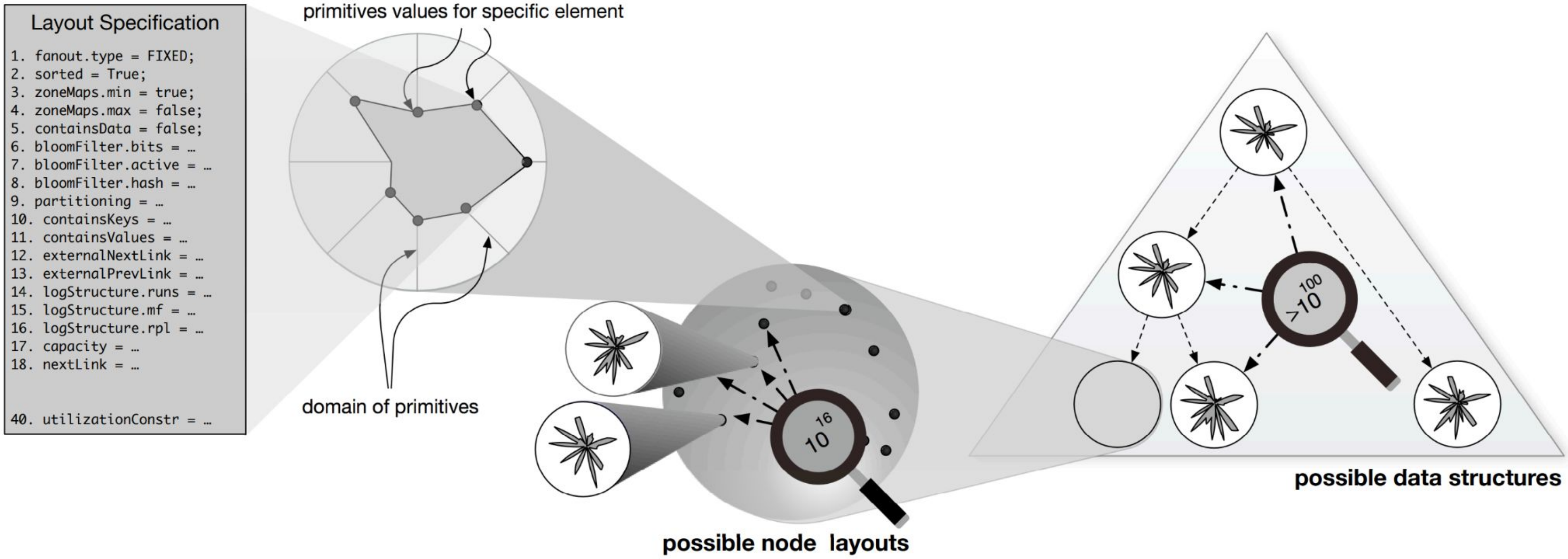
Terminal & Non-Elements

Non-terminal element: act as internal nodes that point to other elements (e.g., internal nodes in a tree).

Terminal element: contain the actual data (e.g., leaves in a B+Tree).

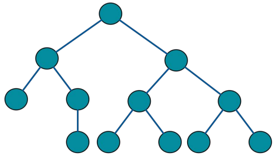


Size of Design space

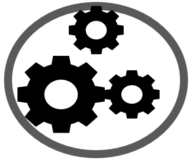


https://stratos.seas.harvard.edu/files/stratos/files/periodictabledatastructures.pdf

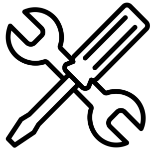
The Data Calculator



Layout Primitives



Access Primitives
&
Cost Synthesis



Usage
&
Experiment

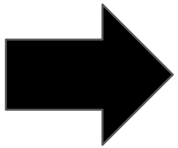
Data Access Primitives

Access Primitives
&
Cost Synthesis

What are access primitives?

Definition: Each access primitive characterizes one aspect of how data is accessed.

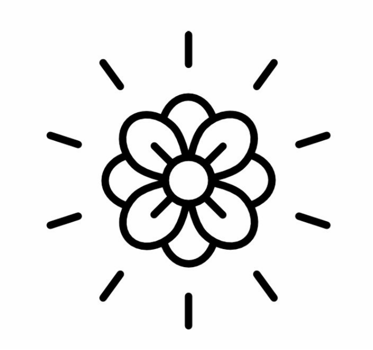
Examples of Data Access Primitives



Scan

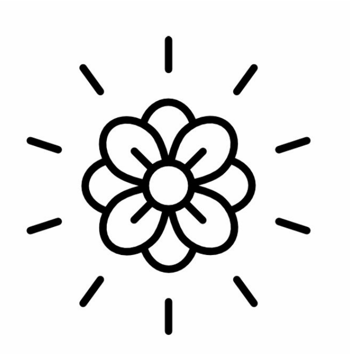
Sorted Search

Hash Probe



Bloom Filter Probe

Examples of Data Access Primitives

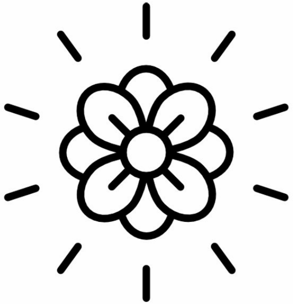


Bloom Filter Probe

Examples of Data Access Primitives

Advantages:

- Reduced I/O’s
- Fast
- Space Efficient

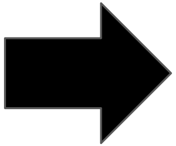


Bloom Filter Probe

Disadvantages:

- False Positives

Advantages of Different Data Access Primitives



Scan

- Good for non-selective queries
- No Index Required

Hash Probe

- Good for point queries
- Efficient Memory Usage

Examples of Data Access Primitives

Different Access Primitives Have Different Advantages

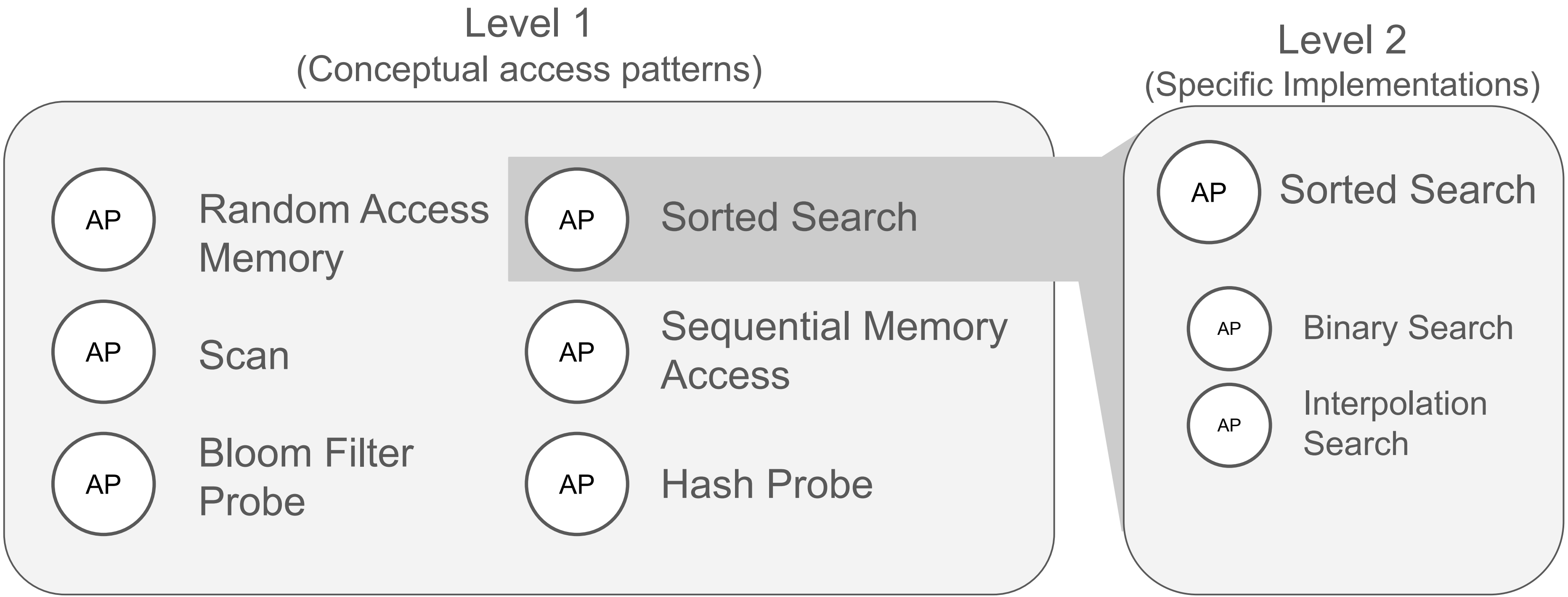
Data Access Primitives and Fitted Models				
	Data Access Primitives Level 1 (required parameters ; optional parameters)	**Model Parameters**	**Data Access Primitives Layer 2**	**Fitted Models**
1	**Scan** (Element Size, Comparison, Data Layout; None)	Data Size	Scalar Scan (RowStore, Equal)	Linear Model (1)
2			Scalar Scan (RowStore, Range)	Linear Model (1)
3			Scalar Scan (ColumnStore, Equal)	Linear Model (1)
4			Scalar Scan (ColumnStore, Range)	Linear Model (1)
5			SIMD-AVX Scan (ColumnStore, Equal)	Linear Model (1)
6			SIMD-AVX Scan (ColumnStore, Range)	Linear Model (1)
7	**Sorted Search** (Element Size, Data Layout;)	Data Size	Binary Search (RowStore)	Log-Linear Model (2)
8			Binary Search (ColumnStore)	Log-Linear Model (2)
9			Interpolation Search (RowStore)	Log + LogLog Model (3)
10			Interpolation Search (ColumnStore)	Log + LogLog Model (3)
11	**Hash Probe** (; Hash Family)	Structure Size	Linear Probing (Multiply-shift [29])	Sum of Sigmoids (5), Weighted Nearest Neighbors (7)
12			Linear Probing (k-wise independent, k=2,3,4,5)	Sum of Sigmoids (5), Weighted Nearest Neighbors (7)
13	**Bloom Filter Probe** (; Hash Family)	Structure Size, Number of Hash Functions	Bloom Filter Probe (Multiply-shift [29])	Sum of Sum of Sigmoids (6), Weighted Nearest Neighbors (7)
14			Bloom Filter Probe (k-wise independent, k=2,3,4,5)	Sum of Sum of Sigmoids (6), Weighted Nearest Neighbors (7)
15	**Sort** (Element Size; Algorithm)	Data Size	QuickSort	NLogN Model (4)
16			MergeSort	NLogN Model (4)
17			ExternalMergeSort	NLogN Model (4)
18	**Random Memory Access**	Region Size	Random Memory Access	Sum of Sigmoids (5), Weighted Nearest Neighbors (7)
19	**Batched Random Memory Access**	Region Size	Batched Random Memory Access	Sum of Sigmoids (5), Weighted Nearest Neighbors (7)
20	**Unordered Batch Write** (Layout;)	Write Data Size	Contiguous Write (RowStore)	Linear Model (1)
21			Contiguous Write (ColumnStore)	Linear Model (1)
22	**Ordered Batch Write** (Layout;)	Write Data Size, Data Size	Batch Ordered Write (RowStore)	Linear Model (1)
23			Batch Ordered Write (ColumnStore)	Linear Model (1)
24	**Scattered Batch Write**	Number of Elements, Region Size	ScatteredBatchWrite	Sum of Sum of Sigmoids (6), Weighted Nearest Neighbors (7)

Levels of Data Access Primitives

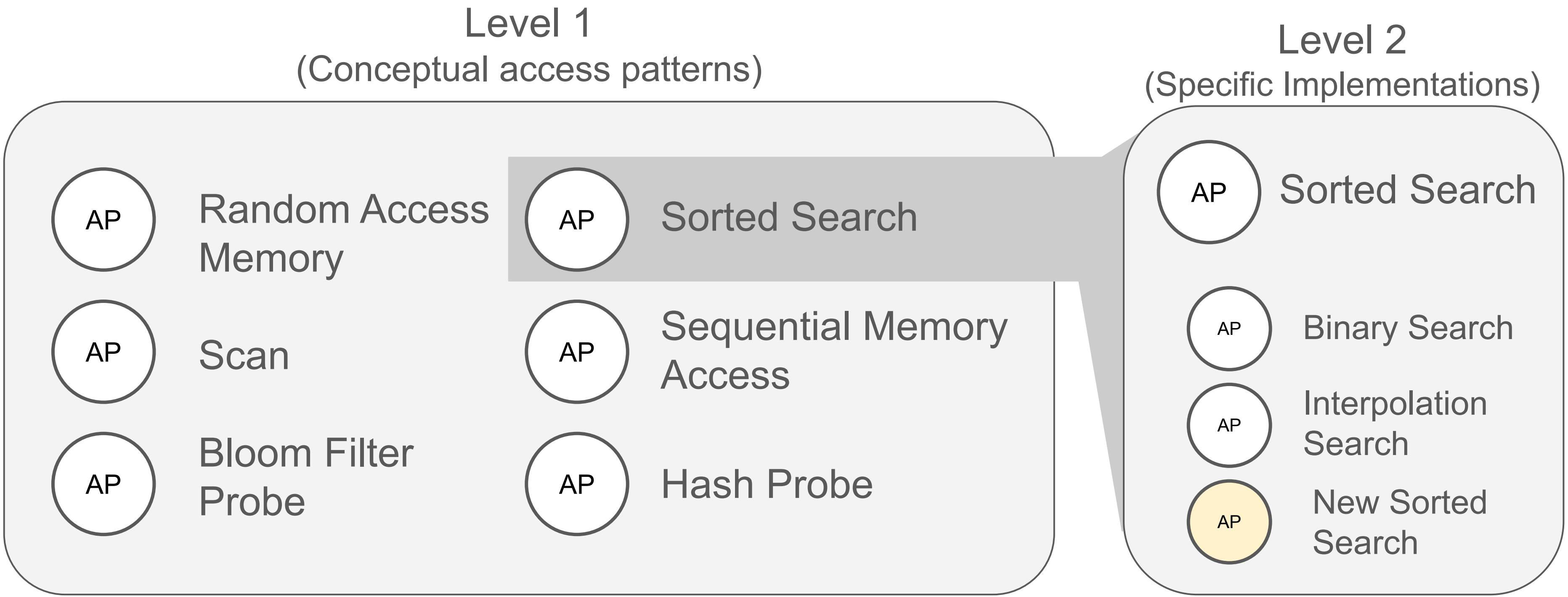
Level 1
(Conceptual access patterns)

- AP Random Access Memory
- AP Sorted Search
- AP Scan
- AP Sequential Memory Access
- AP Bloom Filter Probe
- AP Hash Probe

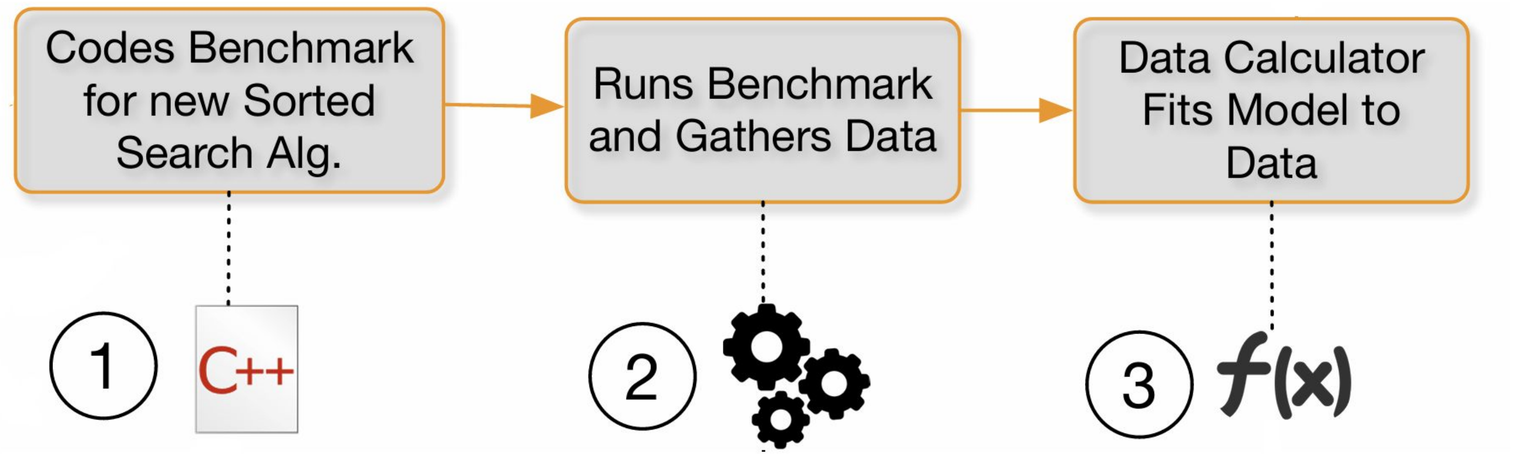
Levels of Data Access Primitives



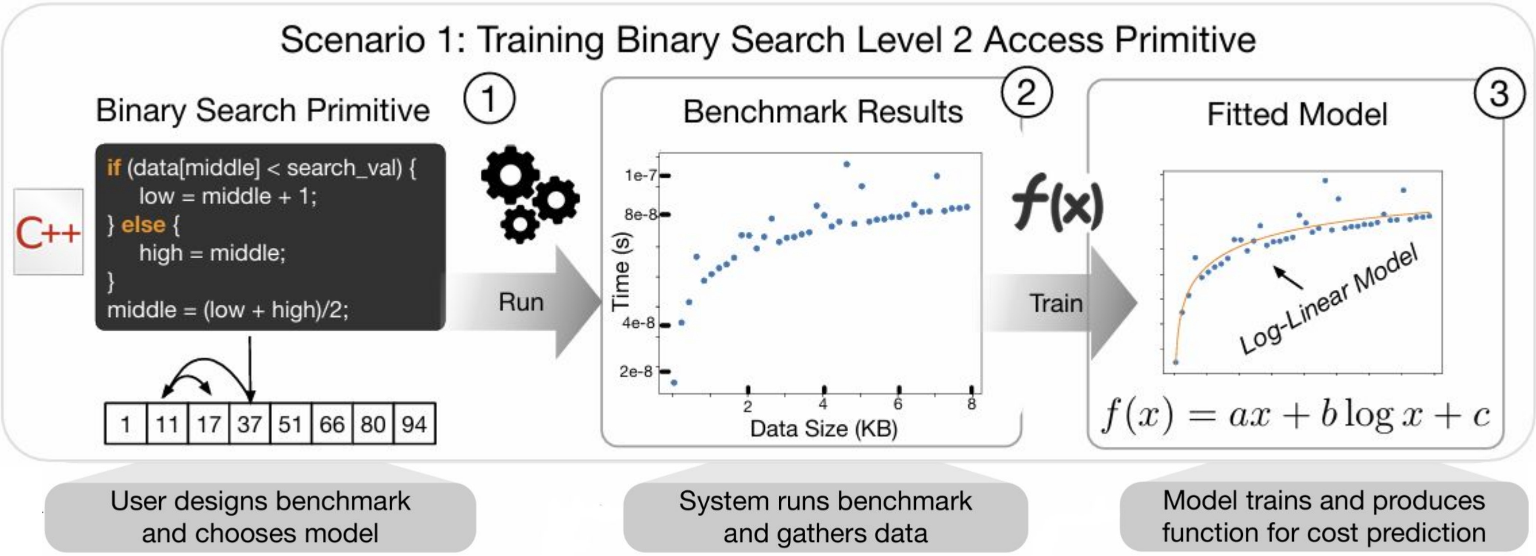
Levels of Data Access Primitives



Learned Cost Models



Learned Cost Model Examples



Learned Cost Models (examples)

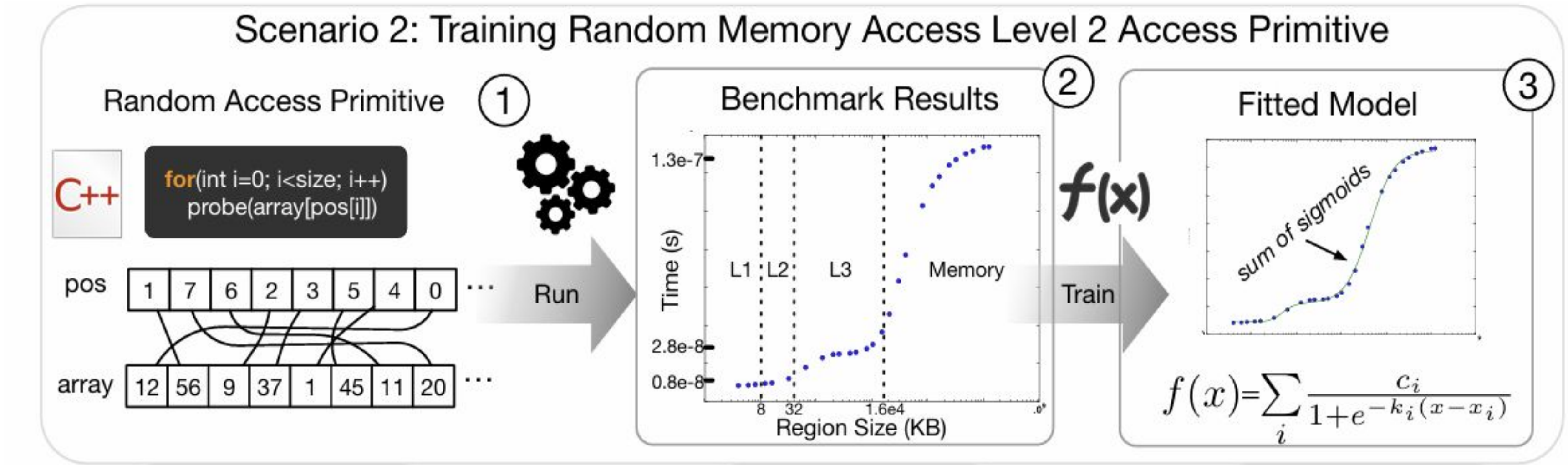
Models used for fitting data access primitives		
	Model	**Description**
1	Linear	Fits a simple line through data
2	Log-Linear	Fits a linear model with a basis composed of the identity and logarithmic functions plus a bias
3	Log+LogLog	Fits a model with log, log-log, and linear components
4	NLogN	Fits a model with primarily an NlogN and linear component
5	Sum of Sigmoids	Fits a model which has two cost components, both of which have k approximate steps occurring at the same locations.
6	Sum of Sum of Sigmoids	Fits a model which has two cost components, both of which have k approximate steps occurring at the same locations.
7	Weighted Nearest Neighbors	Takes the k nearest neighbors under the I_2 norm and computes a weighted average of their outputs. The input x is allowed to be a vector of any size.

Learned Cost Models (examples)

Scan

	Models used for fitting data access primitives	
	Model	**Description**
1	Linear	Fits a simple line through data
2	Log-Linear	Fits a linear model with a basis composed of the identity and logarithmic functions plus a bias
3	Log+LogLog	Fits a model with log, log-log, and linear components
4	NLogN	Fits a model with primarily an NlogN and linear component
5	Sum of Sigmoids	Fits a model which has two cost components, both of which have *k* approximate steps occurring at the same locations.
6	Sum of Sum of Sigmoids	Fits a model which has two cost components, both of which have *k* approximate steps occurring at the same locations.
7	Weighted Nearest Neighbors	Takes the *k* nearest neighbors under the I_2 norm and computes a weighted average of their outputs. The input *x* is allowed to be a vector of any size.

Learned Cost Model Examples



Cost Synthesis

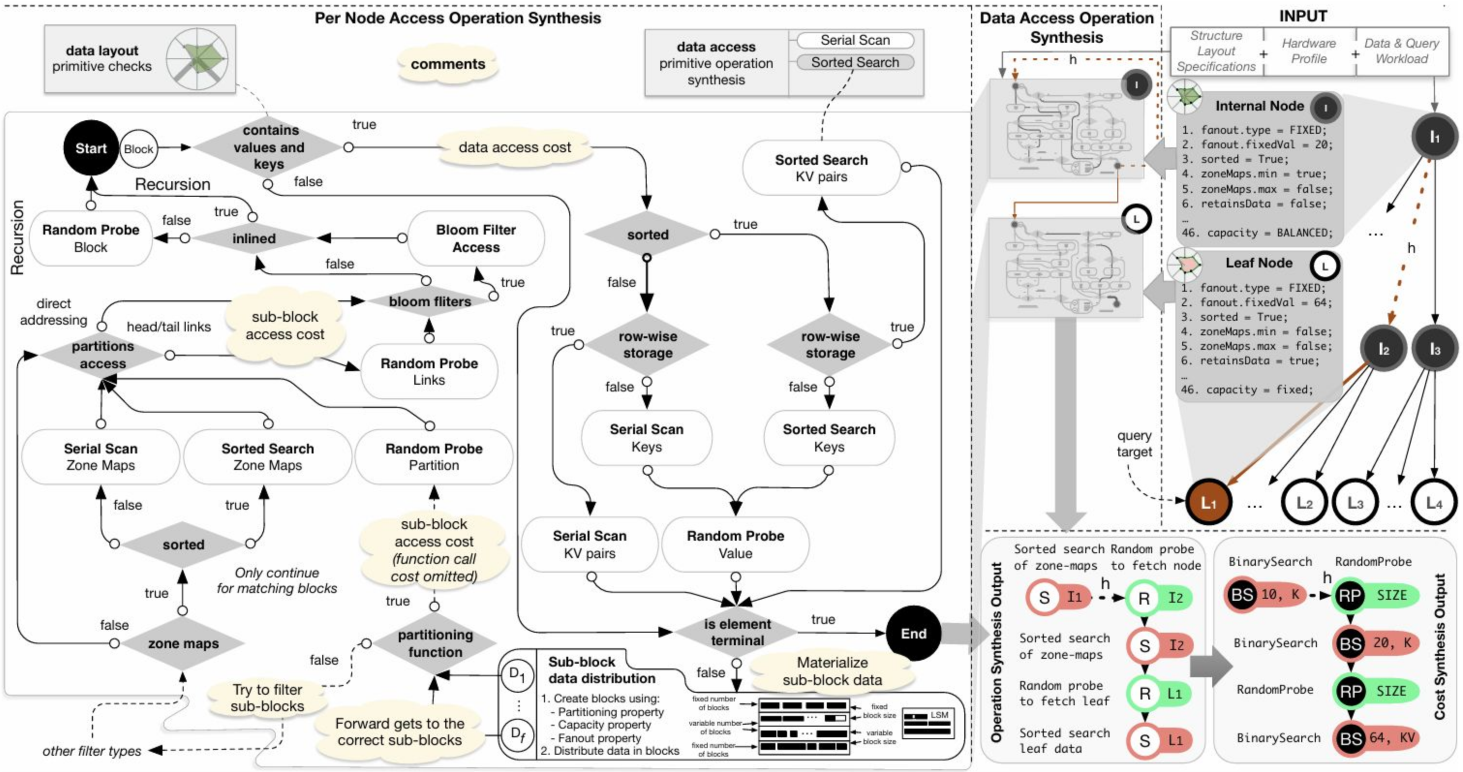


Figure 5: Synthesizing the operation and cost for dictionary operation Get, given a data structure specification.

Cost Synthesis

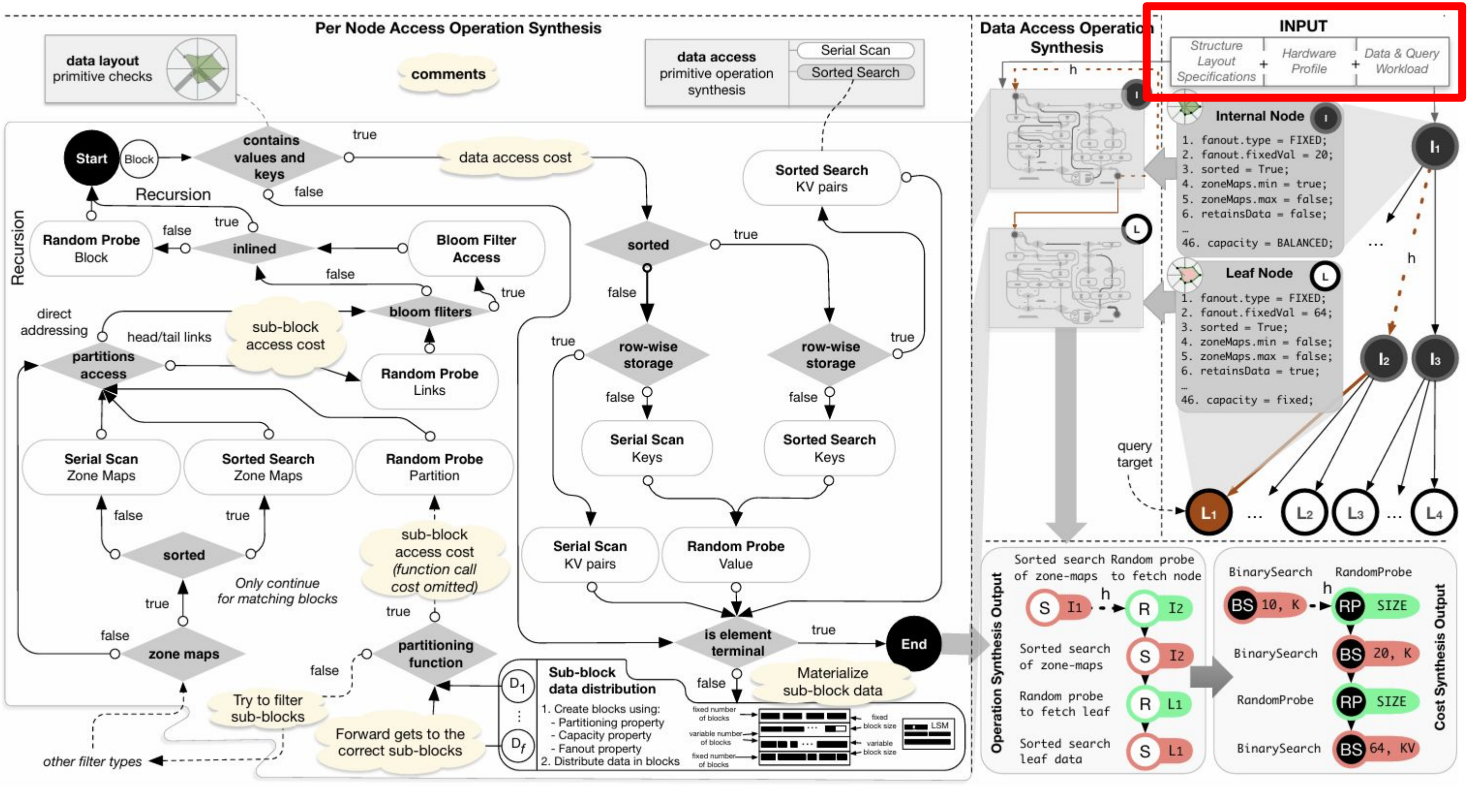


Figure 5: Synthesizing the operation and cost for dictionary operation Get, given a data structure specification.

INPUT

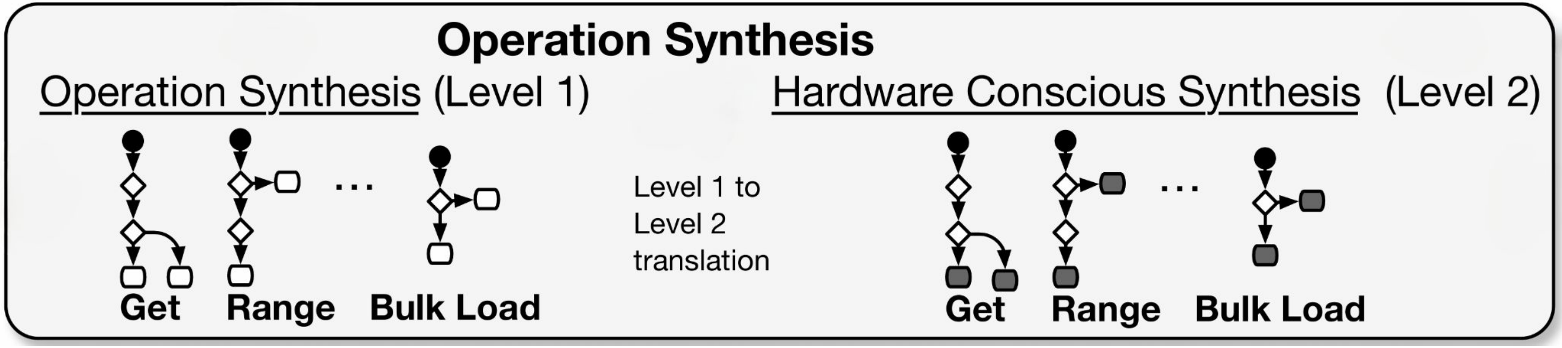
Structure Layout Specifications	+	Hardware Profile	+	Data & Query Workload

INPUT

Structure Layout Specifications + Hardware Profile + Data & Query Workload

INPUT

Structure Layout Specifications	+ *Hardware Profile*	+ *Data & Query Workload*



Cost Synthesis

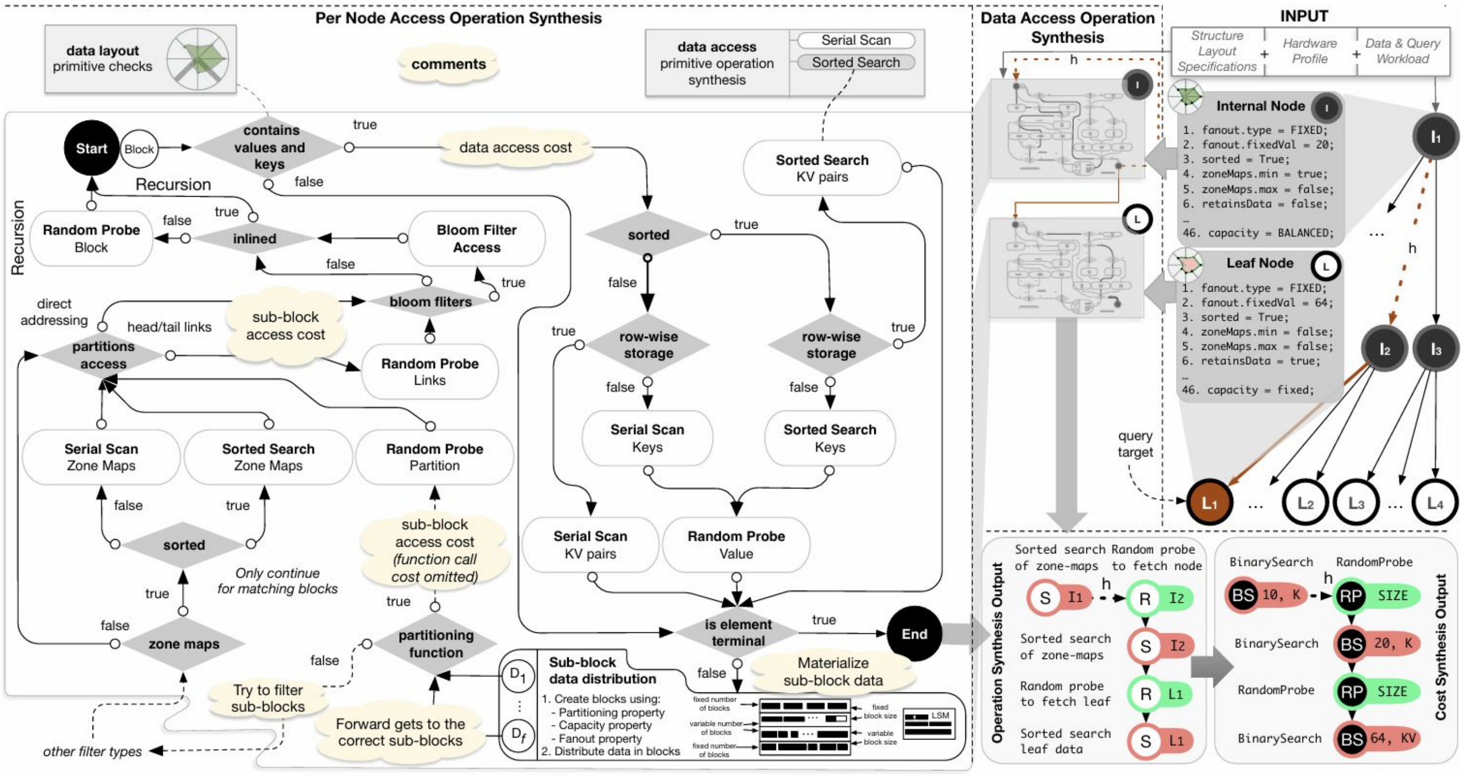


Figure 5: Synthesizing the operation and cost for dictionary operation Get, given a data structure specification.

Cost Synthesis

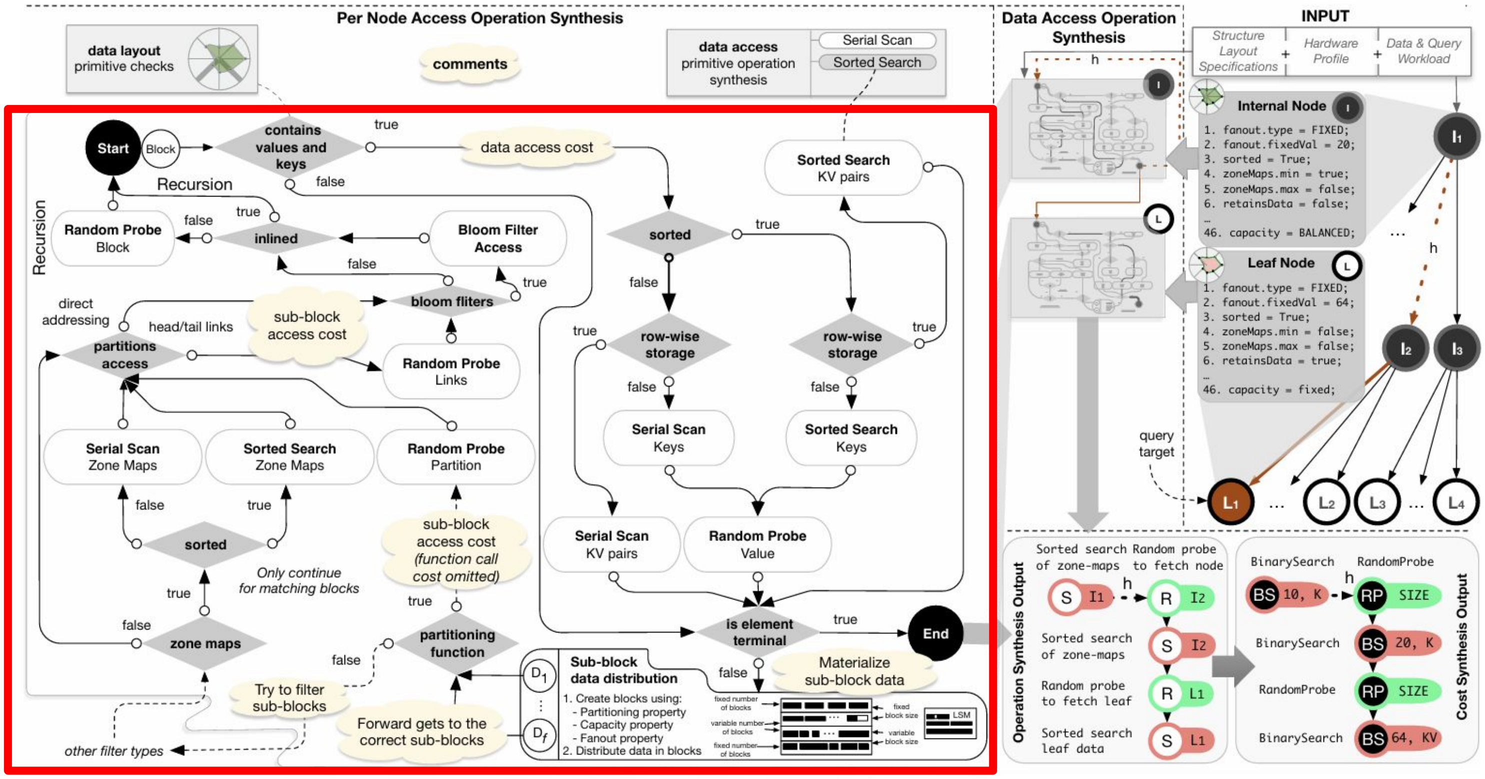
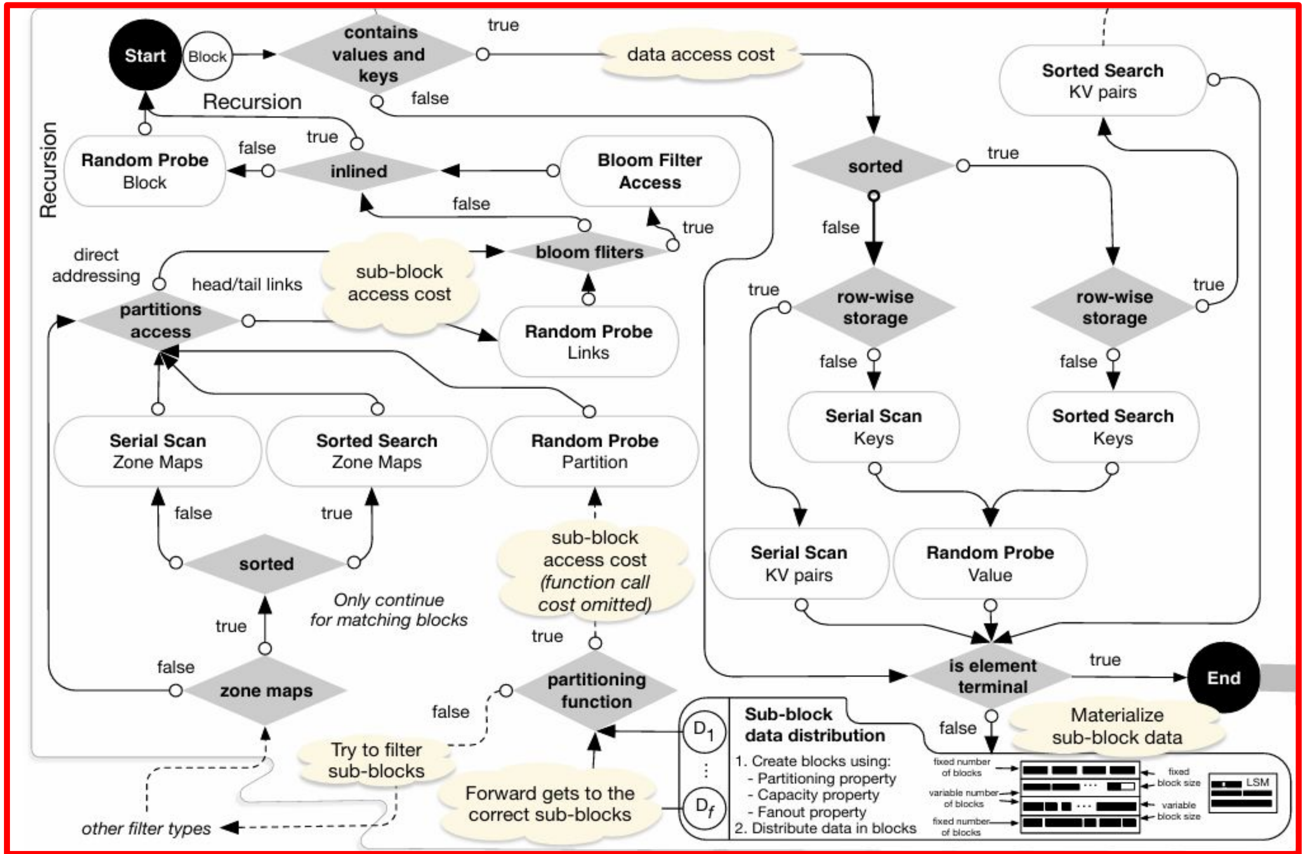
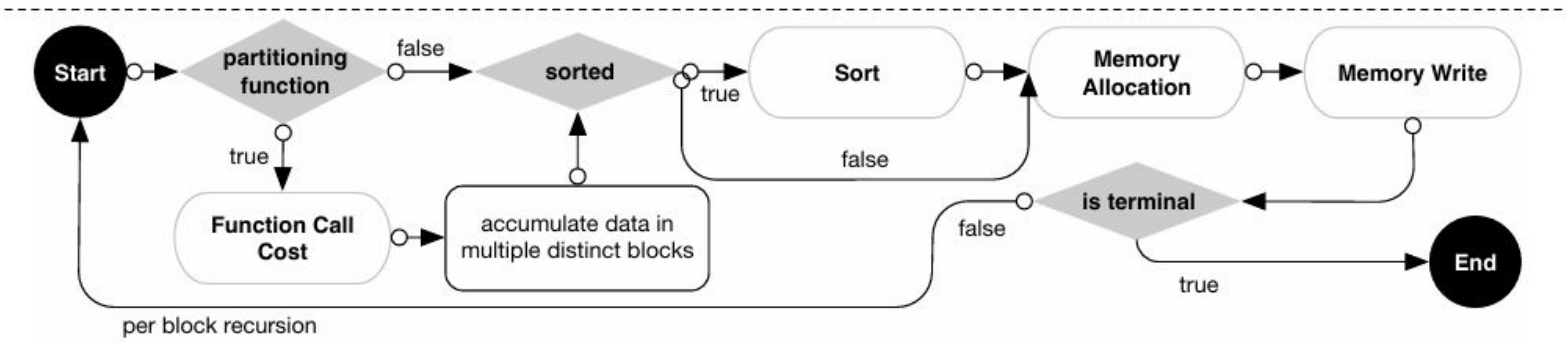


Figure 5: Synthesizing the operation and cost for dictionary operation Get, given a data structure specification.

Cost Synthesis



Cost Synthesis (Bulk Loading Example)



Cost Synthesis

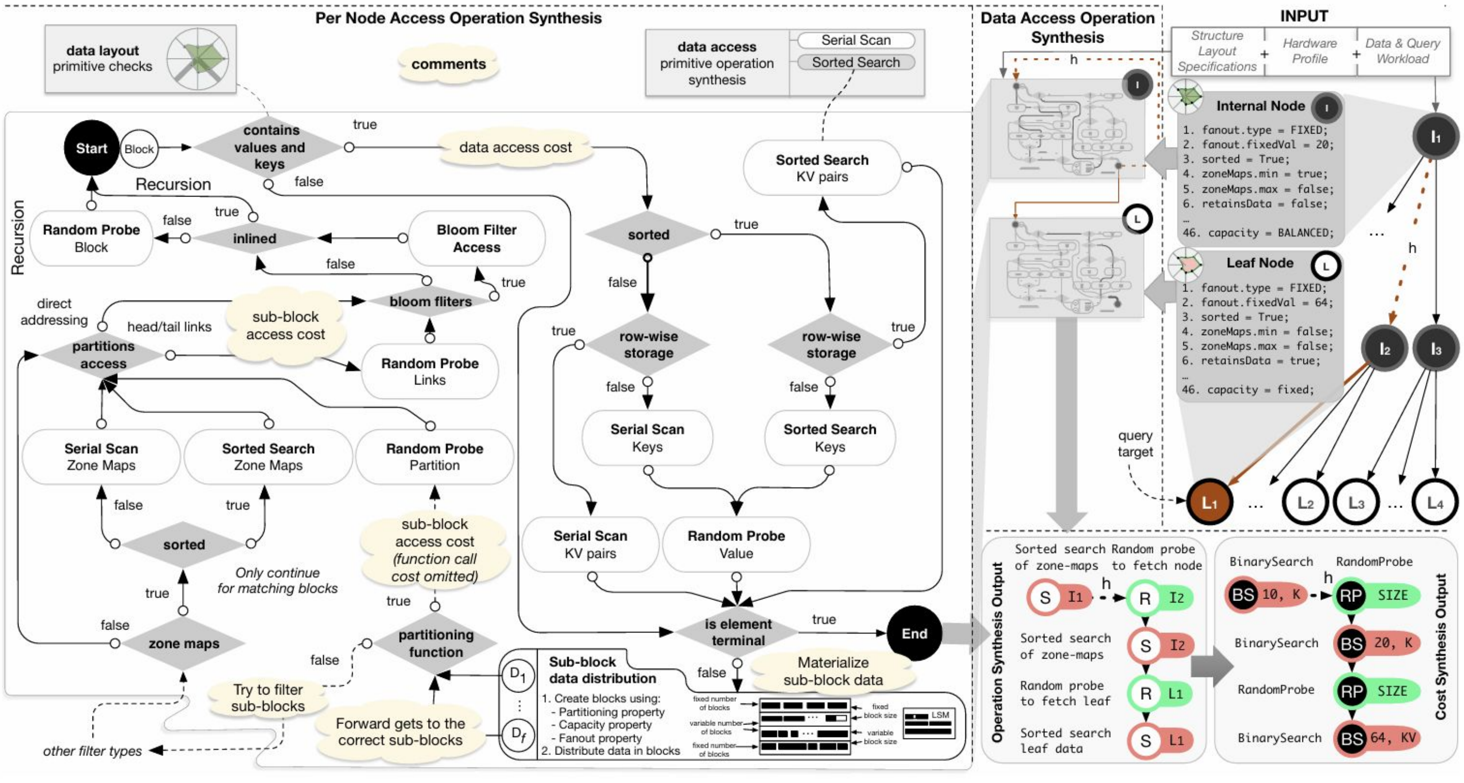


Figure 5: Synthesizing the operation and cost for dictionary operation Get, given a data structure specification.

Cost Synthesis

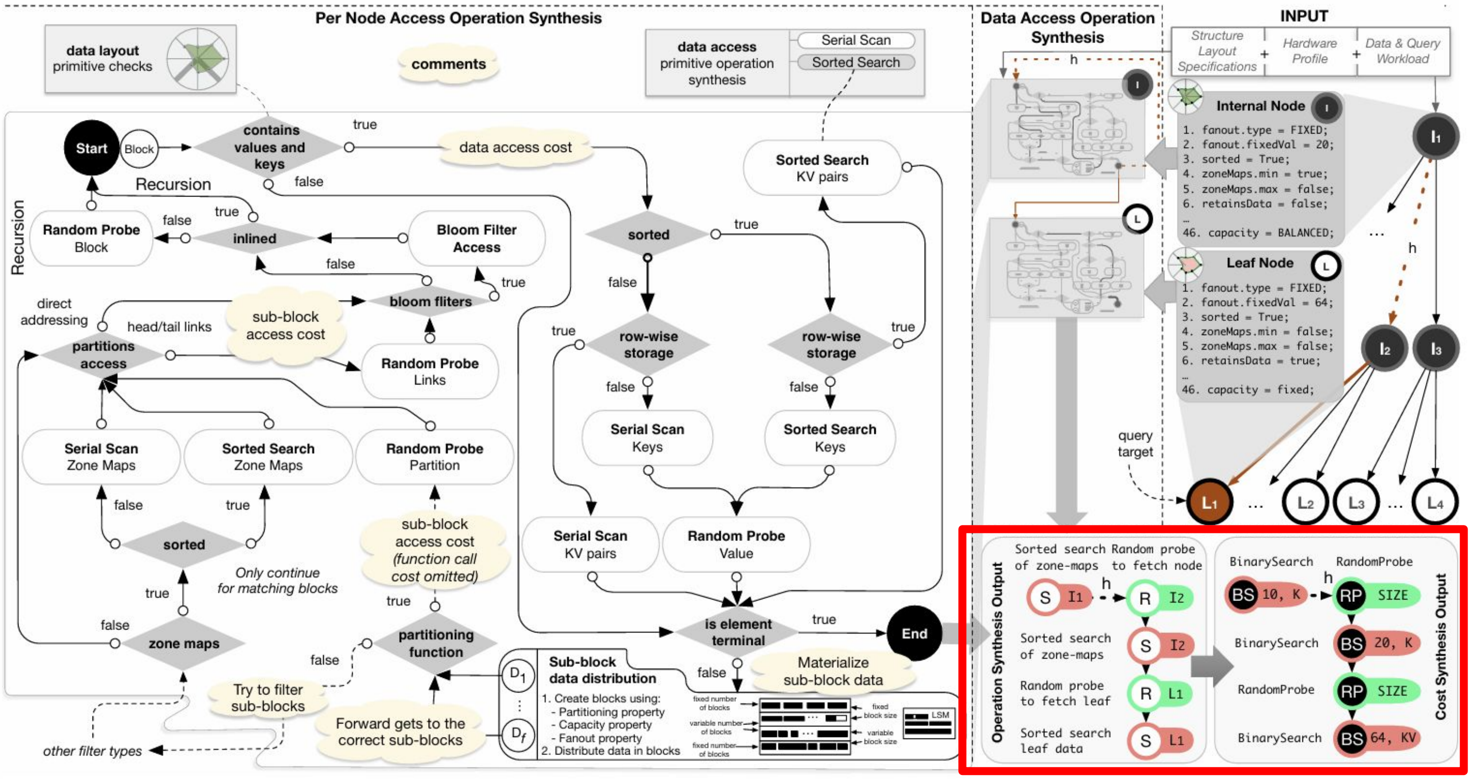
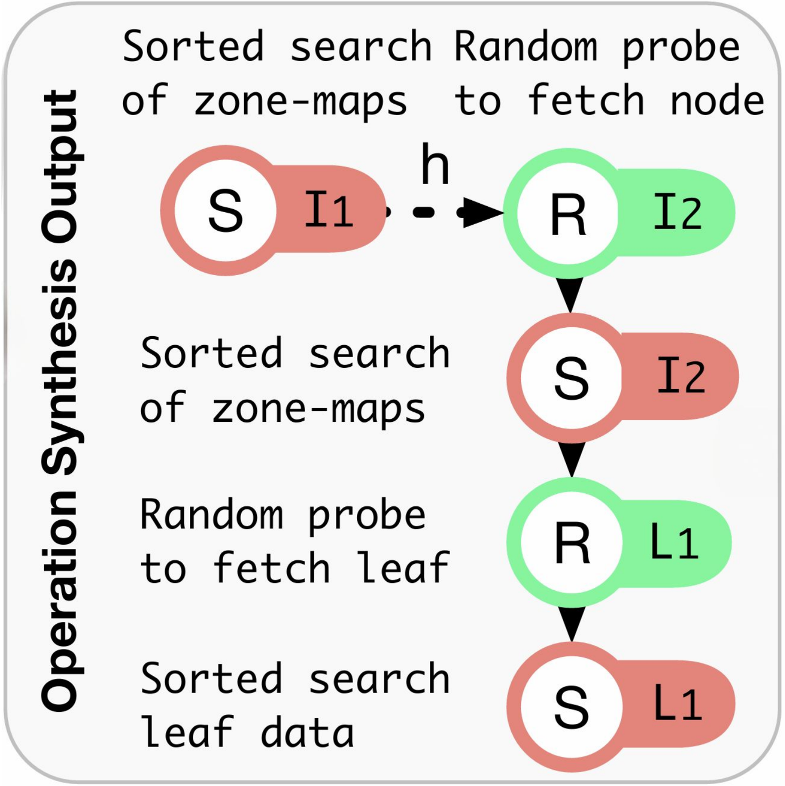
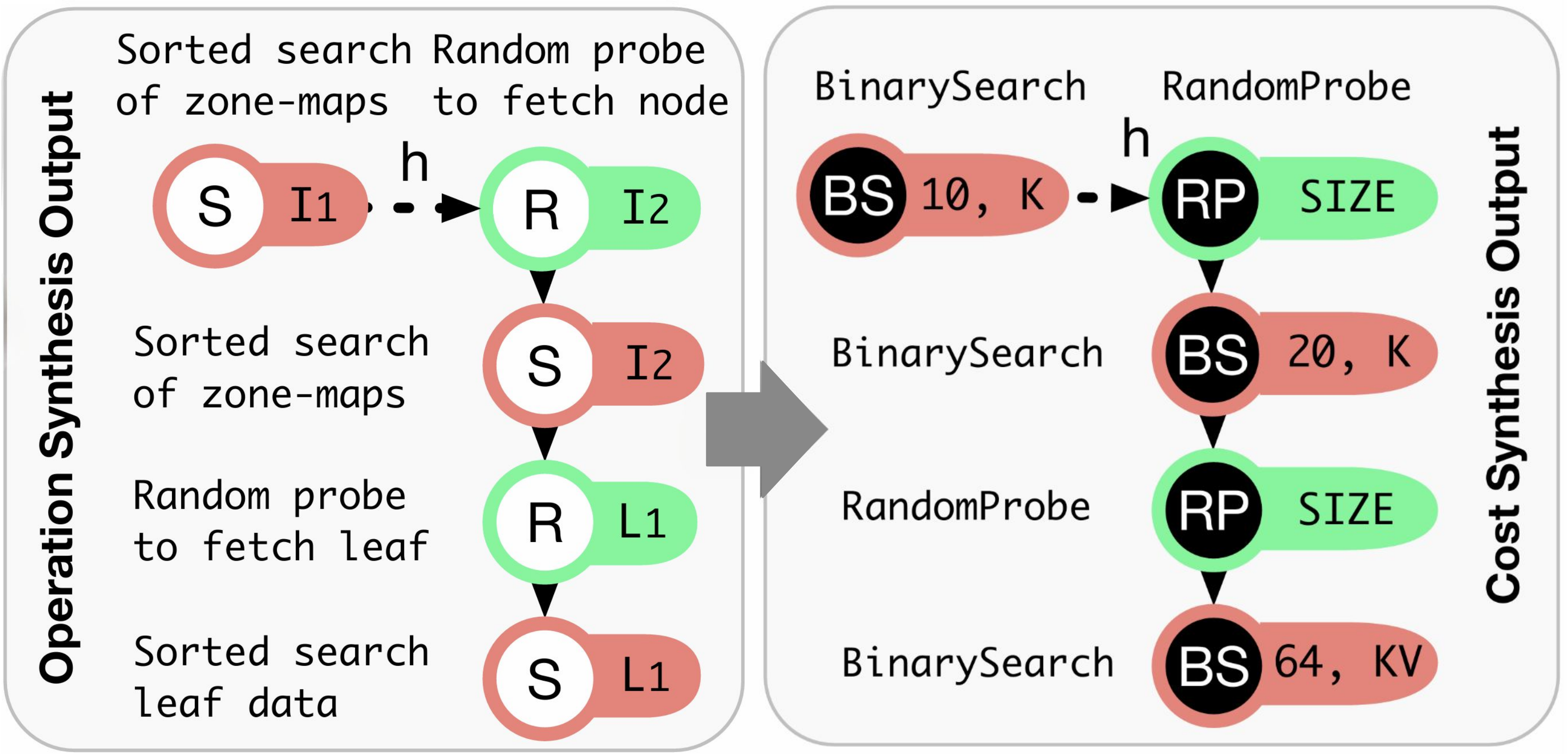


Figure 5: Synthesizing the operation and cost for dictionary operation Get, given a data structure specification.

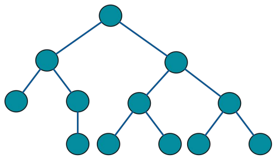
Operation Synthesis to Cost Synthesis



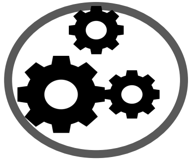
Operation Synthesis to Cost Synthesis



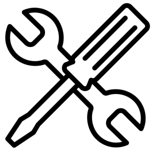
The Data Calculator



Layout Primitives



Access Primitives
&
Cost Synthesis



Usage
&
Experiment

What Can We Do With Data Calculator?

There are too many WHAT-IF questions for designing database!

WHAT IF I want to bring bloom filter to my B-Tree?

WHAT IF I am given a different workload?

WHAT IF I have to use a different hardware?

WHAT IF I need to adjust for a different cache?

Without Data Calculator

DAYS OR EVEN WEEKS OF ACTUAL IMPLEMENTATION

With Data Calculator

HOURS OR EVEN MINUTES OF COST SYNTHESIS

Workflow

Complete Design

↓

Benchmarking

↓

Adjusting

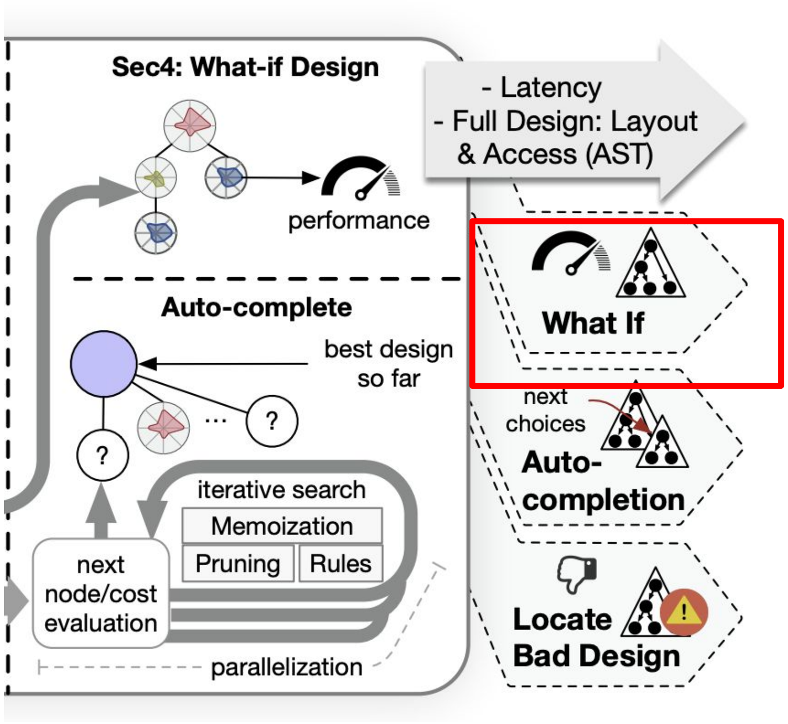
↓

Benchmarking Again

↓

Compare & Decide

EVERY STEP WITHIN A MINUTE!

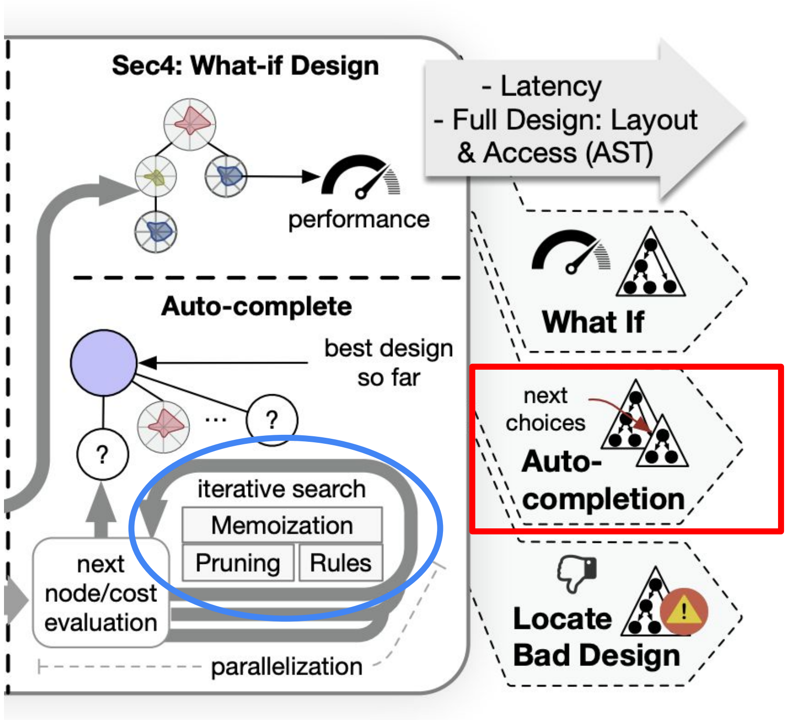


But What If I Don't Even Have a Complete Design?

Data Calculator can auto-complete for you!

? How?

Dynamic Programming!



```
Function CompleteDesign (Q, ℰ, l, currentPath = [], H)
    if blockReachedMinimumSize(H.page_size) then
        return END_SEARCH;
    if !meaningfulPath(currentPath, Q, l) then
        return END_SEARCH;
    if (cacheHit = cachedSolution(Q, l, H)) != null then
        return cacheHit;
    bestSolution = initializeSolution(cost=∞);
```

Q → workload
ε → design space
l → current hierarchy
currentPath → specification to be done
H → hardware profile

```
Function CompleteDesign (Q, ℰ, l, currentPath = [], H)
    if blockReachedMinimumSize(H.page_size) then
        return END_SEARCH;
    if !meaningfulPath(currentPath, Q, l) then
        return END_SEARCH;
    if (cacheHit = cachedSolution(Q, l, H)) != null then
        return cacheHit;
    bestSolution = initializeSolution(cost=∞);
```

Terminate when

1. Size exceeds capacity

2. Further Design not meaningful

3. Design already in cache

```
Function CompleteDesign (Q, ℰ, l, currentPath = [], H)
    if blockReachedMinimumSize(H.page_size) then
        return END_SEARCH;
    if !meaningfulPath(currentPath, Q, l) then
        return END_SEARCH;
    if (cacheHit = cachedSolution(Q, l, H)) != null then
        return cacheHit;
    bestSolution = initializeSolution(cost=∞);
```

Initialize the solution assuming no limit on cost

```
for E ∈ ℰ do
    tmpSolution = initializeSolution();
    tmpSolution.cost = synthesizeGroupCost(E, Q);
    updateCost(E, Q, tmpSolution.cost);
    if createsSubBlocks(E) then
        Q′ = createQueryBlocks(Q);
        currentPath.append(E);
        subSolution = CompleteDesign(Q′, ℰ, l + 1, currentPath);
        if subSolution.cost != END_SEARCH then
            tmpSolution.append(subSolution);

    if tmpSolution.cost ≤ bestSolution.cost then
        bestSolution = tmpSolution ;
```

For each candidate element:

1. Initialize solution and calculate its cost with element E

2. Update current solution with the cost

```
for E ∈ ℰ do
    tmpSolution = initializeSolution();
    tmpSolution.cost = synthesizeGroupCost(E, Q);
    updateCost(E, Q, tmpSolution.cost);
    if createsSubBlocks(E) then
        Q′ = createQueryBlocks(Q);
        currentPath.append(E);
        subSolution = CompleteDesign(Q′, ℰ, l + 1, currentPath);
        if subSolution.cost != END_SEARCH then
            tmpSolution.append(subSolution);
    if tmpSolution.cost ≤ bestSolution.cost then
        bestSolution = tmpSolution ;
```

If there exists element under E:
Recursively update cost with subelement as a new level of hierarchy

```
for E ∈ ℰ do
    tmpSolution = initializeSolution();
    tmpSolution.cost = synthesizeGroupCost(E, Q);
    updateCost(E, Q, tmpSolution.cost);
    if createsSubBlocks(E) then
        Q′ = createQueryBlocks(Q);
        currentPath.append(E);
        subSolution = CompleteDesign(Q′, ℰ, l + 1, currentPath);
        if subSolution.cost != END_SEARCH then
            tmpSolution.append(subSolution);

    if tmpSolution.cost ≤ bestSolution.cost then
        bestSolution = tmpSolution ;
```

Update current best solution if the new solution costs less

```
    cacheSolution(Q, l, bestSolution);
    return bestSolution;
```

Store the current best solution in the cache and return it

Experiment: Accuracy of Cost Synthesis

System implementation: C++

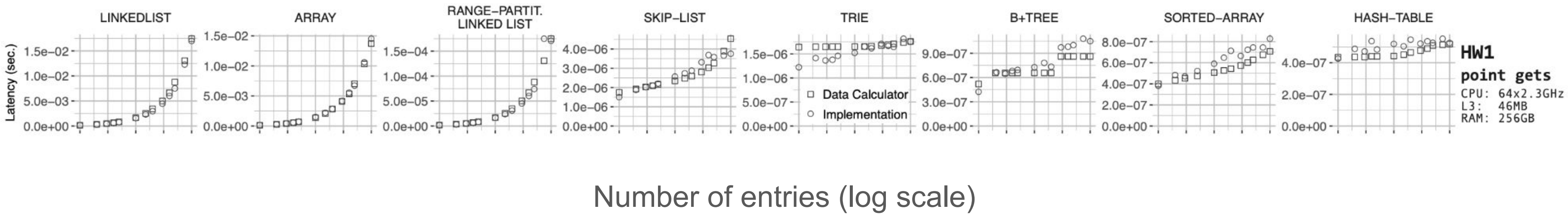
Benchmark analysis & Learning: Python

Idea: Compare cost with actual implementation and cost synthesis

Operation type: Point Get, Range Get, Update, Bulk Load

HW1	HW2	HW3
CPU: 64 x 2.3 GHz	CPU: 4 x 2.3 GHz	CPU: 64 x 2 GHz
L3: 46MB	L3: 46MB	L3: 16MB
RAM: 256GB	RAM: 16GB	RAM: 1TB

Experiment: Accuracy of Cost Synthesis



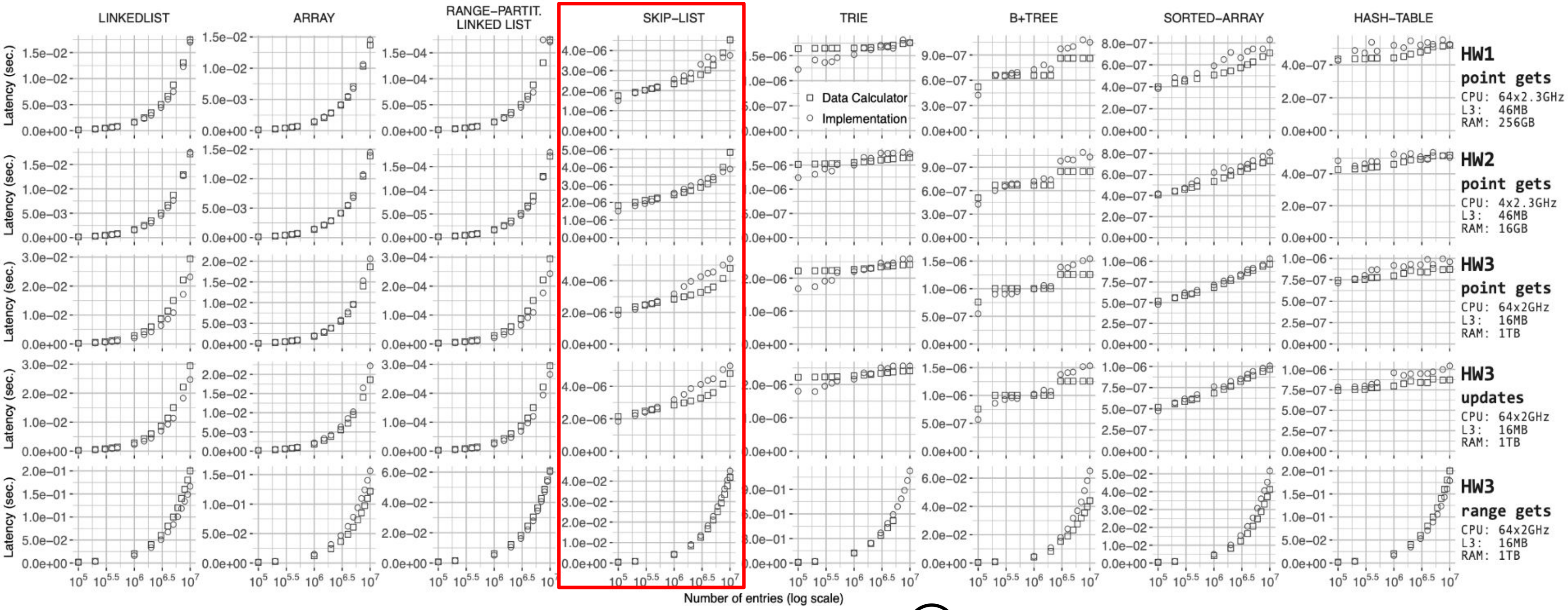
Data Calculator

Implementation

Cost Synthesis is sufficiently accurate!

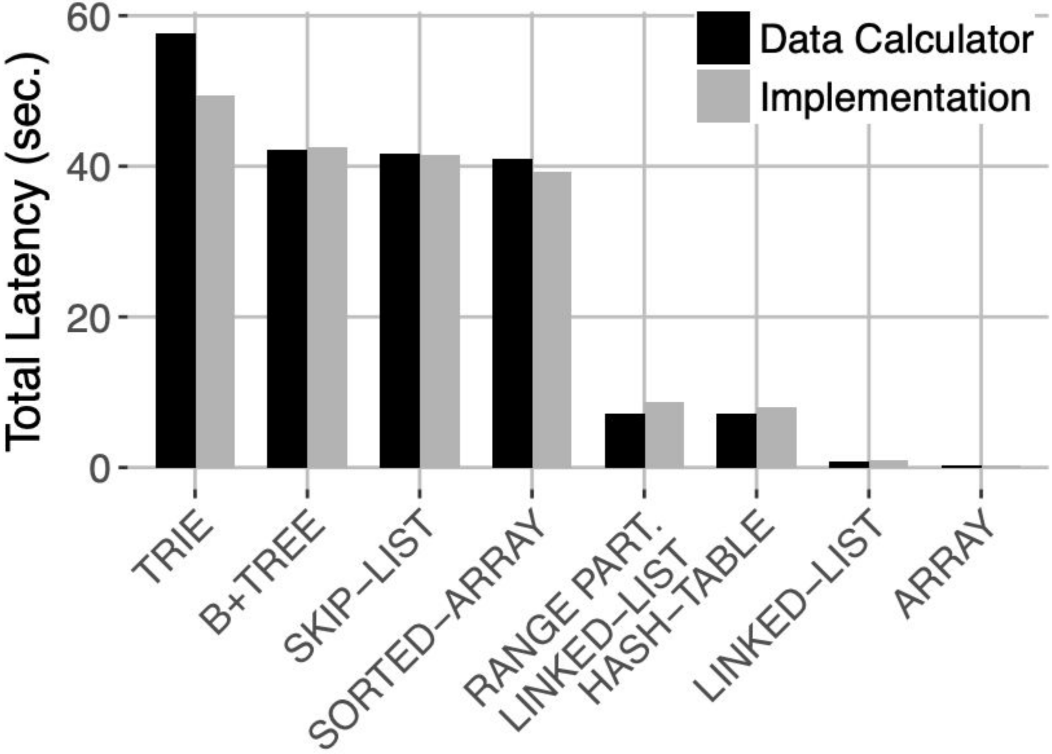
Experiment: Accuracy of Cost Synthesis

Cost synthesis is accurate for all experimented data structures!



? any interesting pattern?

Experiment: Accuracy of Cost Synthesis



For bulk loading, cost synthesis is accurate

? Why the result for trie is the least accurate?

Cache-sensitive Design

How “far” each node is placed to each other

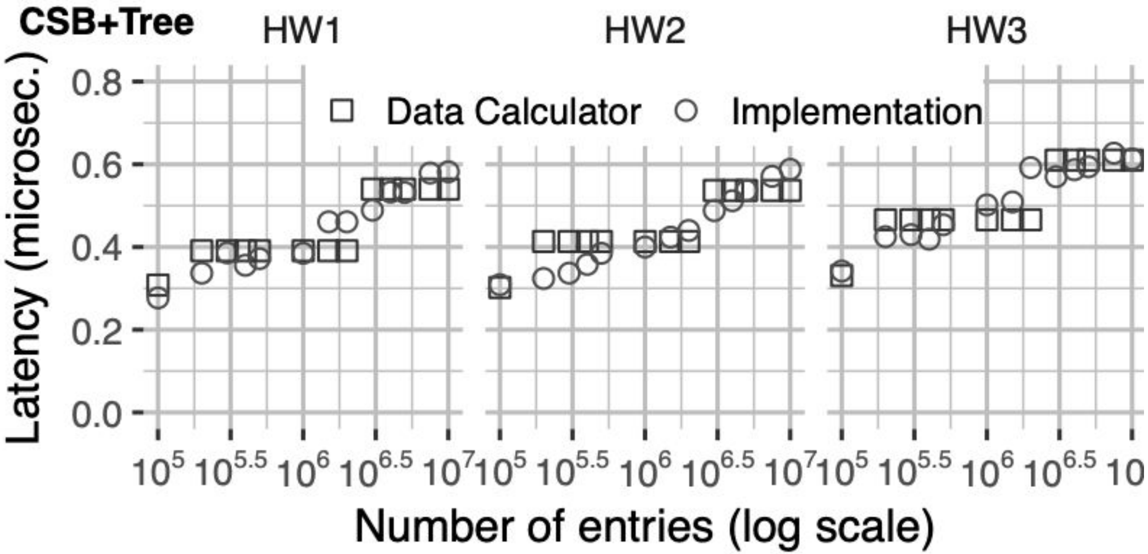
Crucial for calculating the cost of traversing data structure

Represented as layout primitive

Cache-Sensitive B+ Tree vs. B+ Tree

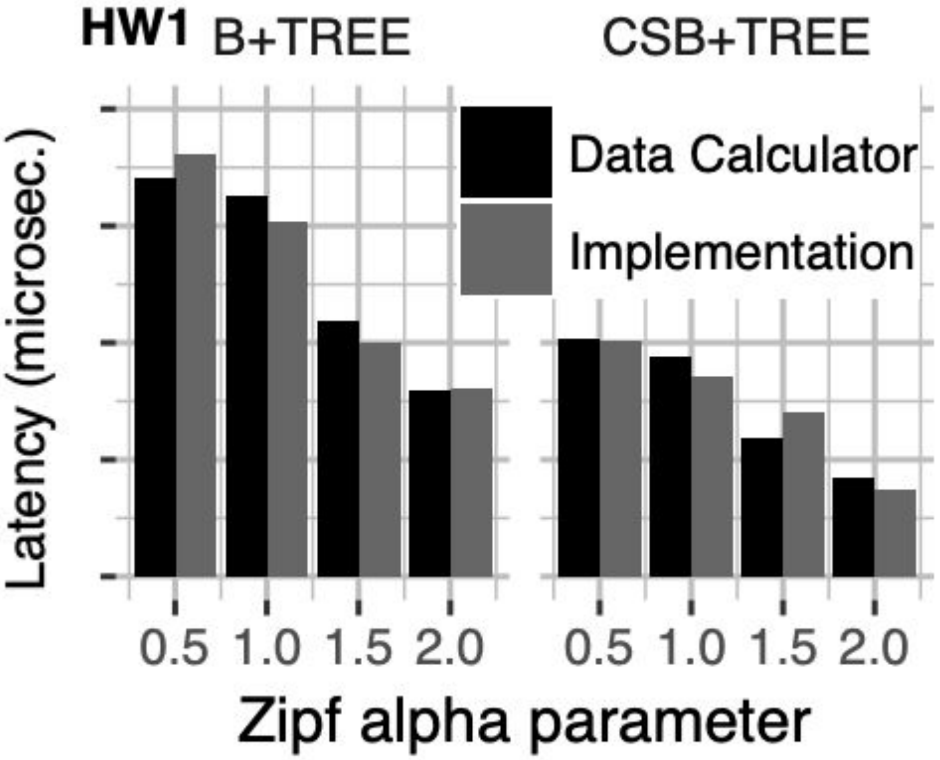
				Hash Table			B+Tree/CSB+Tree/FAST			
					LPL					
	Primitive	Domain	size	H	LL	UDP	B+	CSB+	FAST	ODP
17	**Sub-block physical layout.** This represents the physical layout of sub-blocks. Scatter: random placement in memory. BFS: laid out in a breadth-first layout. BFS layer list: hierarchical level nesting of BFS layouts. Rules: requires fanout/radix != terminal.	BFS | BFS layer(level-grouping: int) | scatter *(up to 3 different values for layer-grouping are considered)*	5	scatter	scatter		scatter	BFS	BFS-LL	

Experiment: Accuracy of Cost Synthesis



It is accurate for CSB+ tree as well!

Experiment: Accuracy of Cost Synthesis

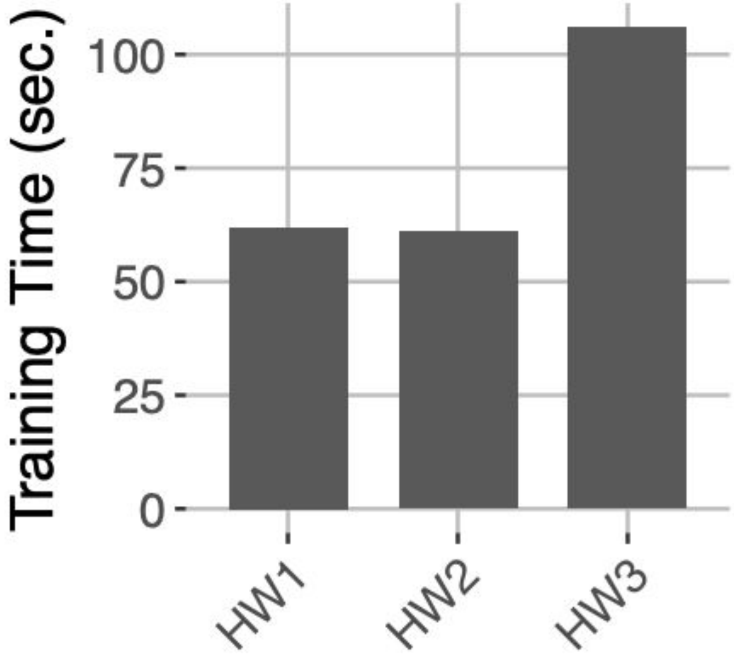


Accuracy of cost synthesis increases as skewness of workload increases.

? Why is that?

? Why does it improve more for B+ Tree?

Experiment: Speed of Cost Synthesis



Training by seconds
vs.
Implementing by days

Experiment: Speed of Cost Synthesis

"What if we change our **hardware** from HW1 to HW3?"

20s

"Is there a better design for **this new hardware and workload** if we restrict search on a specific set of possible elements?"

47s

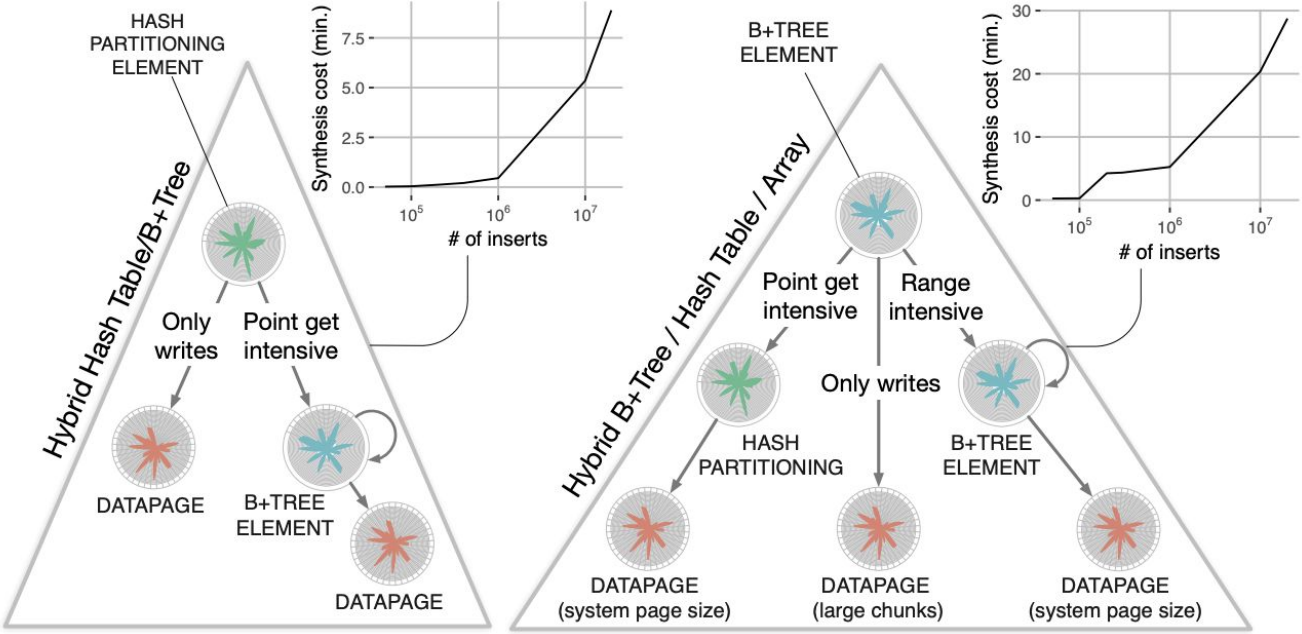
"Would it be beneficial to **add a Bloom filter in all B-tree leaves**?"

20s

"What if the **query workload** changes to have **skew** targeting just 0.01% of the key space?"

24s

Give Me a Better Design → in 30 Minutes!



(a) Scenario 1

(b) Scenario 2

Scenario 1: mixed read&write, all read are point queries in 20% of the domain

Scenario 2: mixed read&write, half of read are point queries in 10% of the domain, the other half are range queries in another 10% of the domain

Potential for improvement

Introduction of more design elements

Support for cost synthesis of advanced operations (such as point/range delete)

Optimization for cost synthesis

? Others?

Recap:

Design Primitives:

Fundamental building blocks for describing data structures.

Enables the exploration of a massive design space.

Learned Cost Models:

Predict operation latencies without implementation or workload execution.

Adaptable to hardware and workload profiles.

What-If Analysis:

Answer complex questions about design, hardware, and workload trade-offs interactively.

Auto-Completion:

Semi-automated synthesis of new data structures and auto-completion of partial designs.

Reflections

Ge:

The paper provides a transformative approach to data structure design by introducing a framework that combines fundamental design primitives and learned cost models to explore, synthesize, and evaluate a vast design space. The tool's ability to predict costs without implementation or workload execution is particularly impressive.

Alec:

I enjoyed working on this paper, it provided a good framework for evaluating cost of more complex structures by breaking things down into the costs of their constituent parts, and accounts for differences in hardware (without the ambiguity of big O notation for actual numbers). It also provides algorithms for optimization rather than just being a tool to view results of a hypothetical data structure which is pretty neat.

Minjie:

Even though nowadays when we design database we need to consider more factors like cloud storage, distributed environments and so on. This paper set up a nice starting point for how we can extend these ideas, combining them with new paradigms.