Write-Optimized Data Structures

Michael A. Bender
Stony Brook & Tokutek

STONY BROOK
STATE UNIVERSITY OF NEW YORK

Tokutek®

Tokutek

A few years ago I started collaborating with Martin Farach-Colton and Bradley Kuszmaul on I/O-efficient and cache-oblivious data structures.

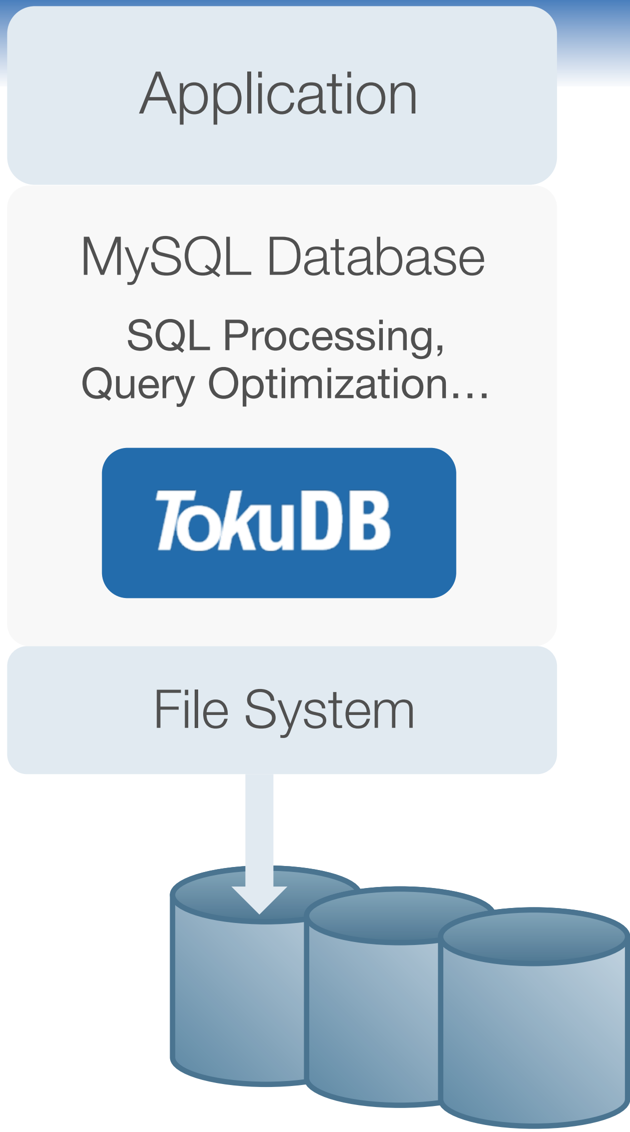
Michael

Martin

Bradley

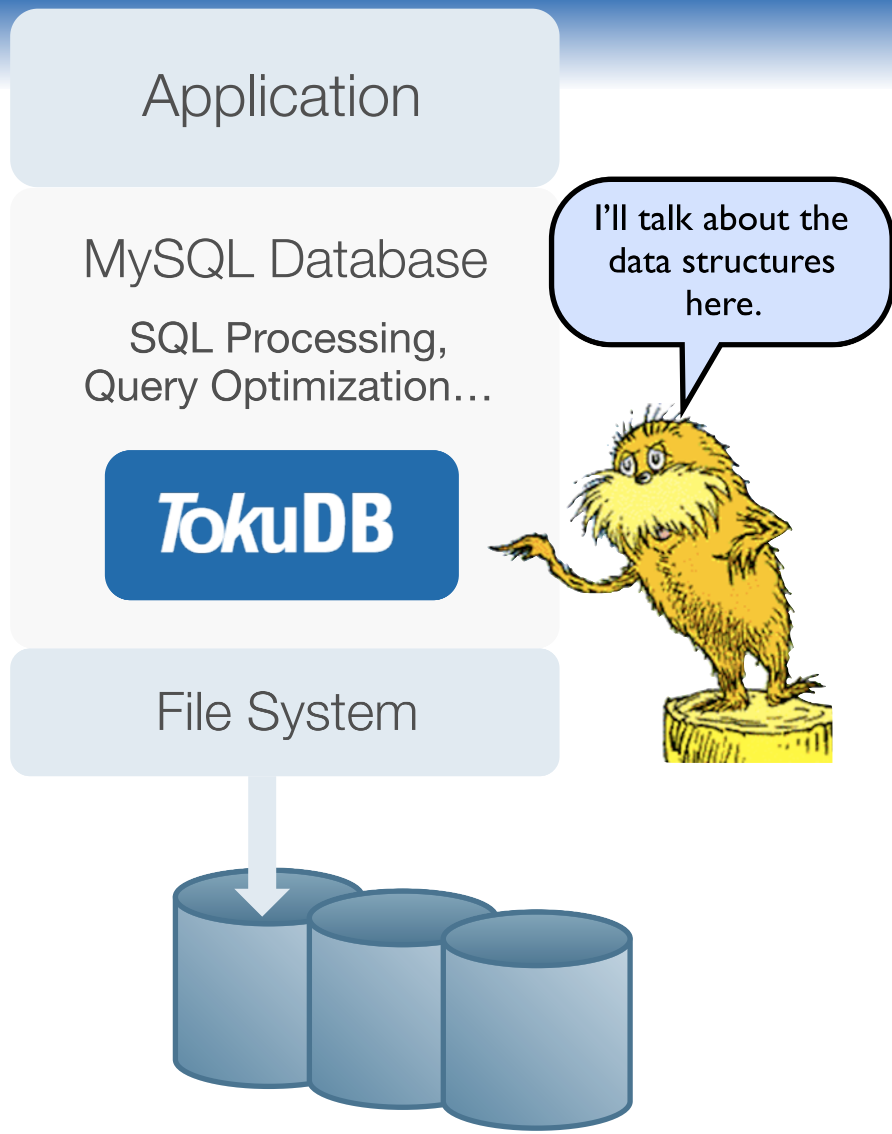
We started Tokutek to commercialize our research.

Storage engines in MySQL

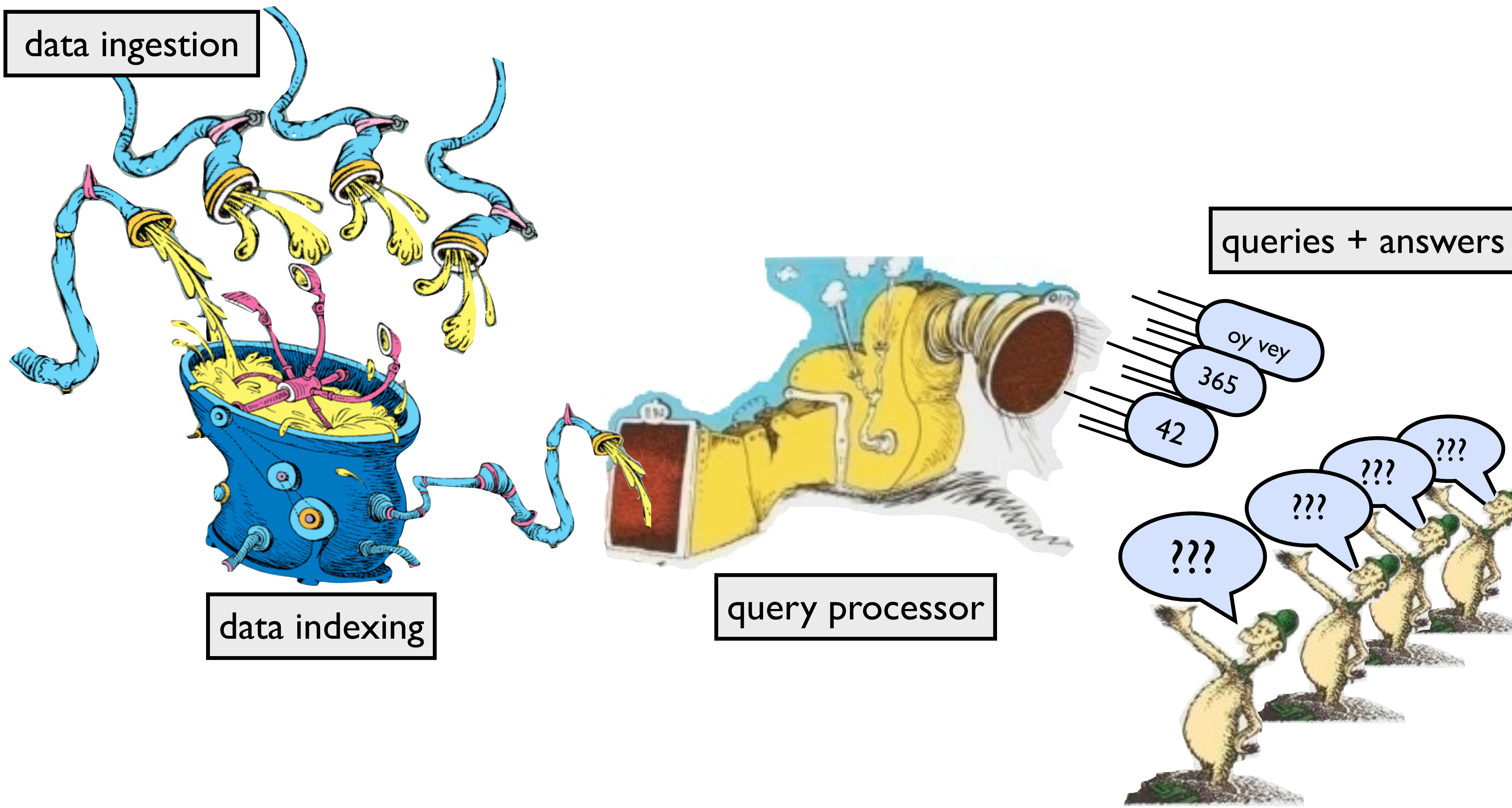


Tokutek sells TokuDB, an ACID compliant, closed-source storage engine for MySQL.

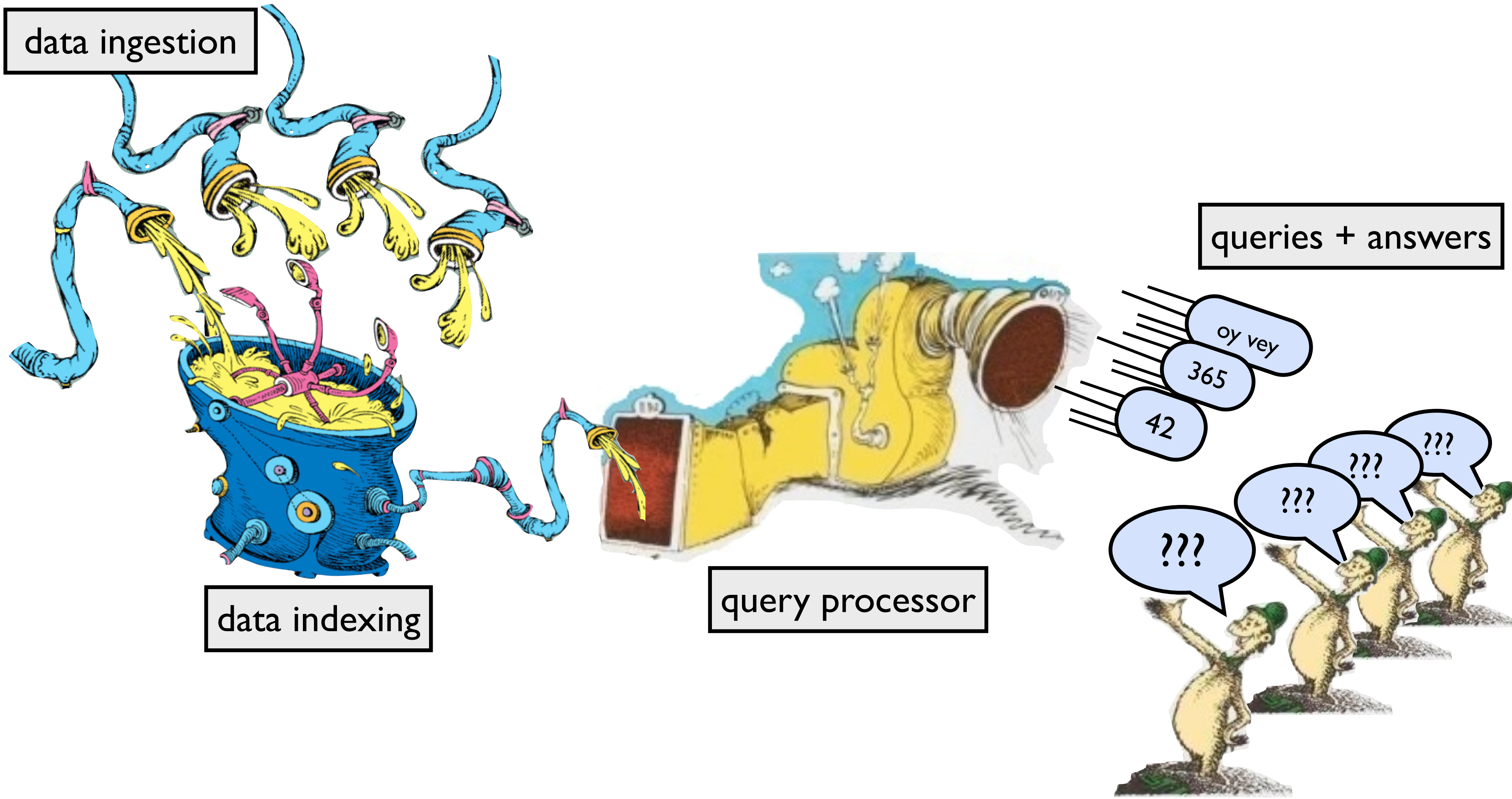
Storage engines in MySQL



Tokutek sells TokuDB, an ACID compliant, closed-source storage engine for MySQL.

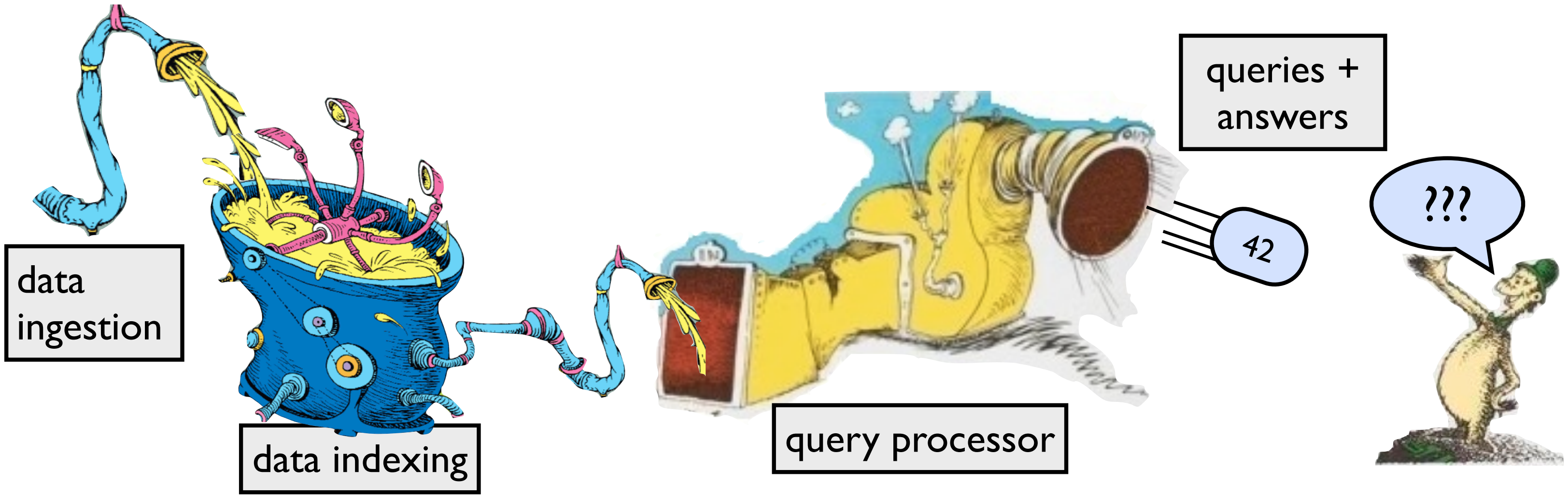

data ingestion
data indexing
query processor
queries + answers
oy vey
365
42
???
???
???
???

For on-disk data, one sees funny tradeoffs in the speeds of data ingestion, query speed, and liveness of data.



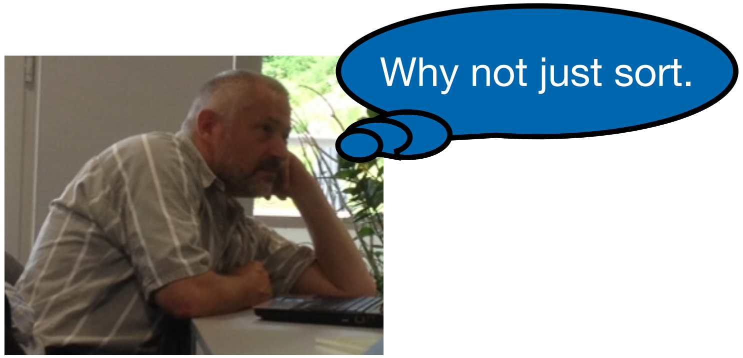
Funny tradeoff in ingestion, querying, liveness

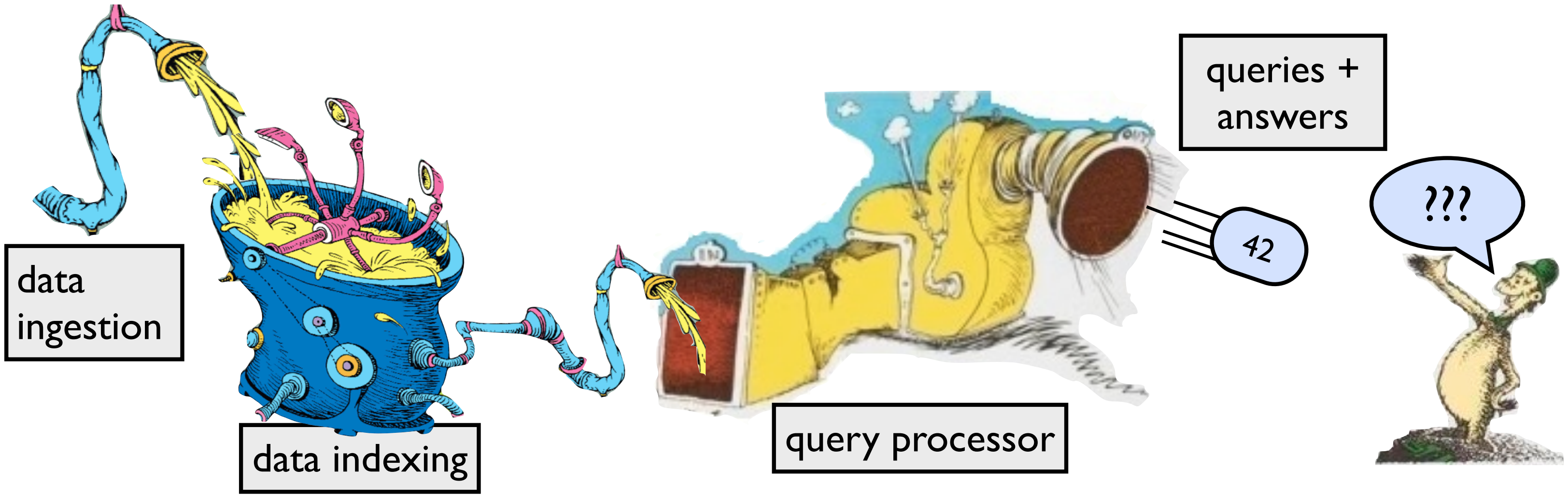
- “I'm trying to create indexes on a table with 308 million rows. It took ~20 minutes to load the table but 10 days to build indexes on it.”
  - MySQL bug #9544



Funny tradeoff in ingestion, querying, liveness

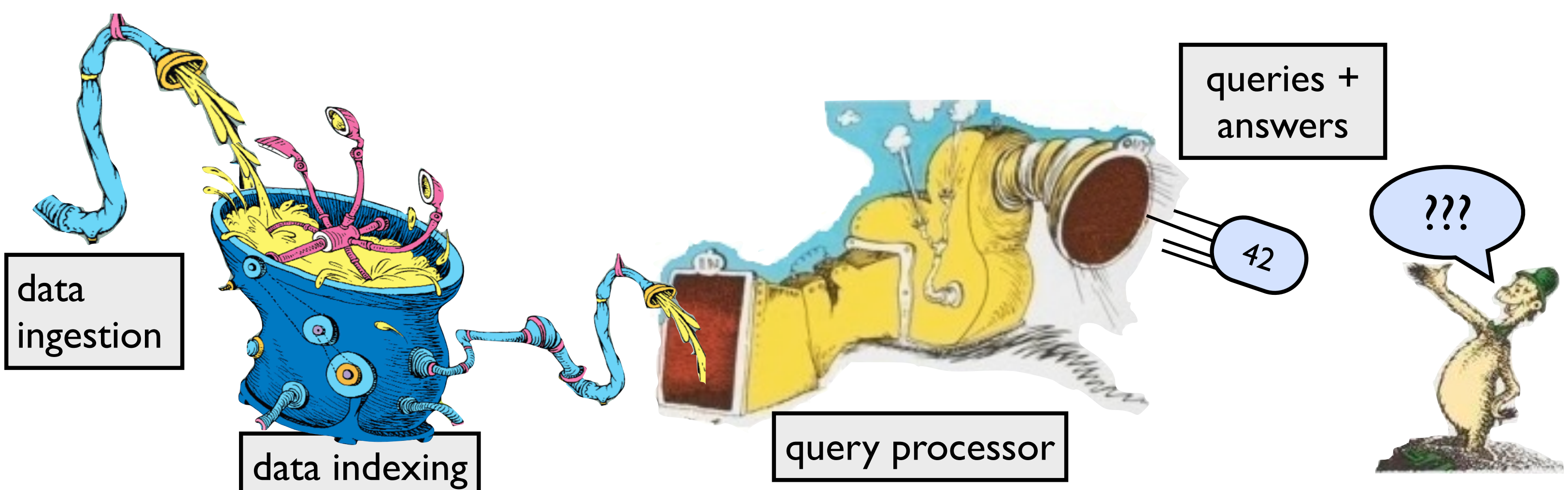
- “I'm trying to create indexes on a table with 308 million rows. It took ~20 minutes to load the table but 10 days to build indexes on it.”
  - MySQL bug #9544





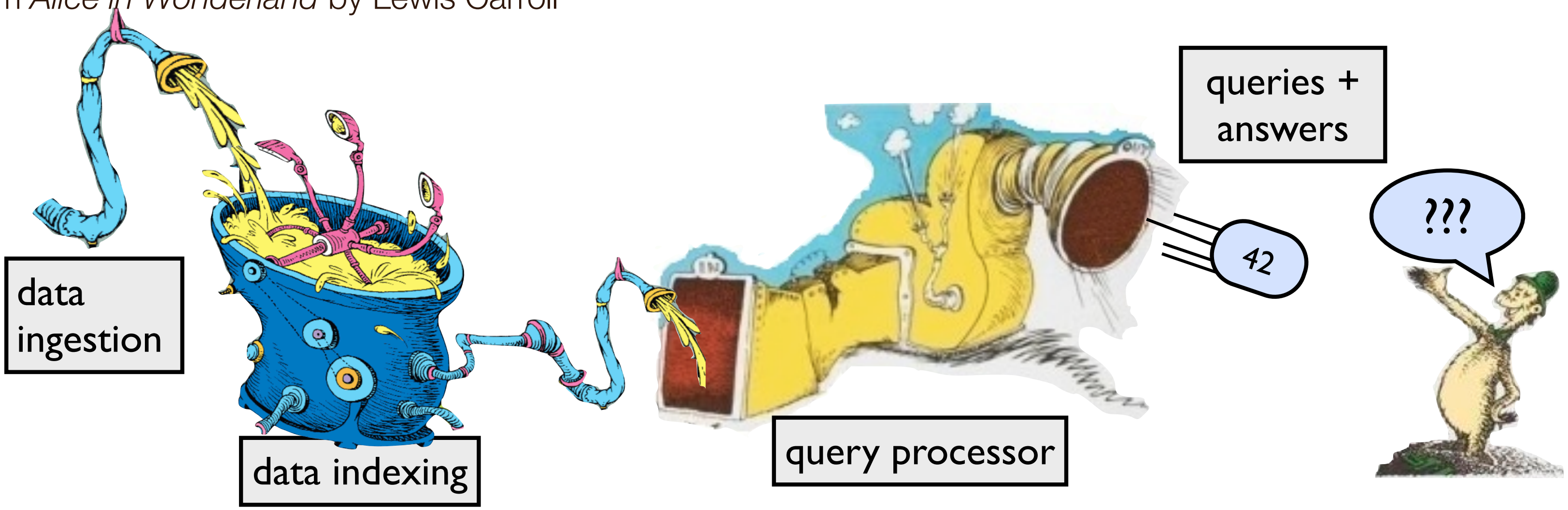
Funny tradeoff in ingestion, querying, liveness

- “I'm trying to create indexes on a table with 308 million rows. It took ~20 minutes to load the table but 10 days to build indexes on it.”
  - MySQL bug #9544
- “Select queries were slow until I added an index onto the timestamp field... Adding the index really helped our reporting, BUT now the inserts are taking forever.”
  - Comment on mysqlperformanceblog.com

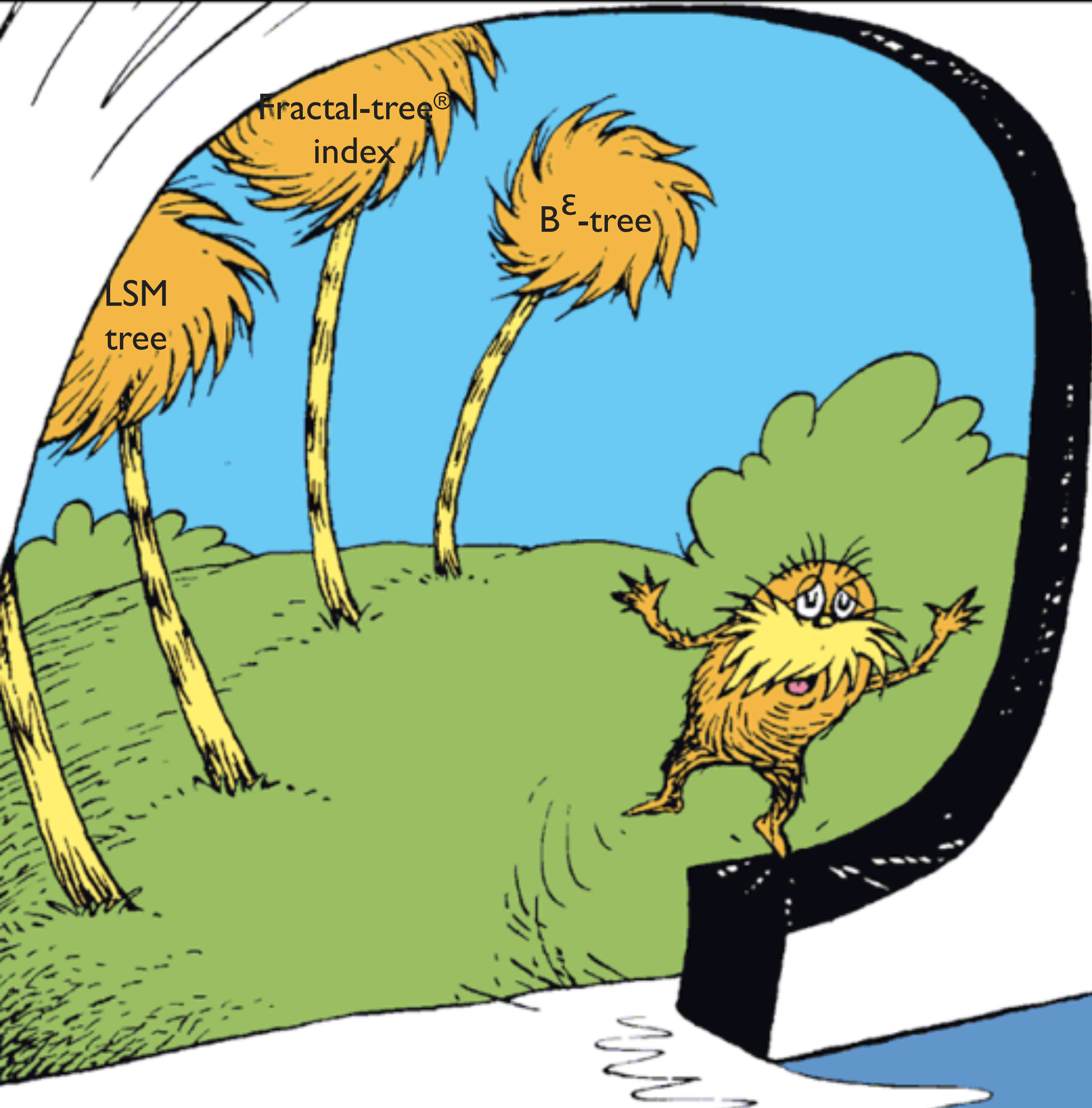


Funny tradeoff in ingestion, querying, liveness

- “I'm trying to create indexes on a table with 308 million rows. It took ~20 minutes to load the table but 10 days to build indexes on it.”
  - MySQL bug #9544
- “Select queries were slow until I added an index onto the timestamp field... Adding the index really helped our reporting, BUT now the inserts are taking forever.”
  - Comment on mysqlperformanceblog.com
- “They indexed their tables, and indexed them well,
And lo, did the queries run quick!
But that wasn’t the last of their troubles, to tell–
Their insertions, like treacle, ran thick.”
  - Not from *Alice in Wonderland* by Lewis Carroll



This talk



- Write-optimized structures significantly mitigate the insert/query/liveness tradeoff.
- One can insert 10x-100x faster than B-trees while achieving similar point query performance.

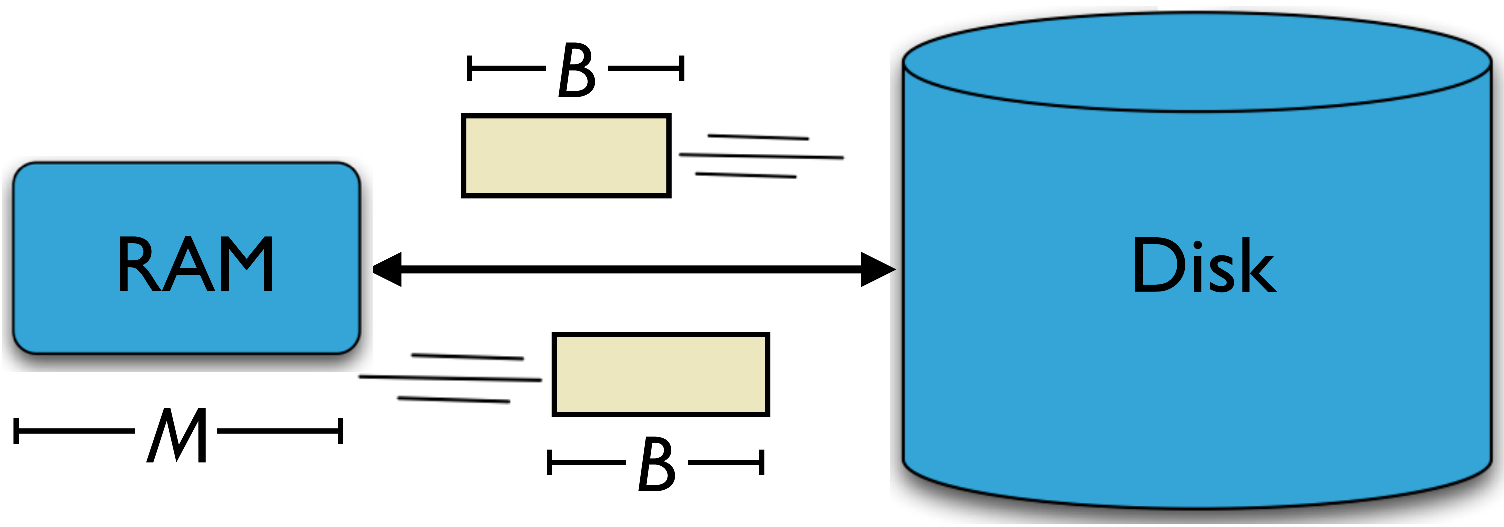
An algorithmic performance model

How computation works:

- Data is transferred in blocks between RAM and disk.
- The number of block transfers dominates the running time.

Goal: Minimize # of block transfers

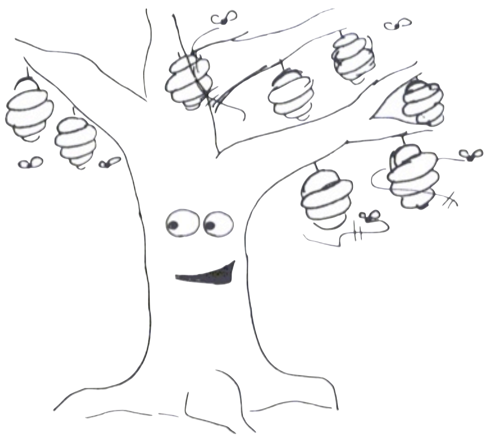
- Performance bounds are parameterized by block size ***B***, memory size ***M***, data size ***N***.

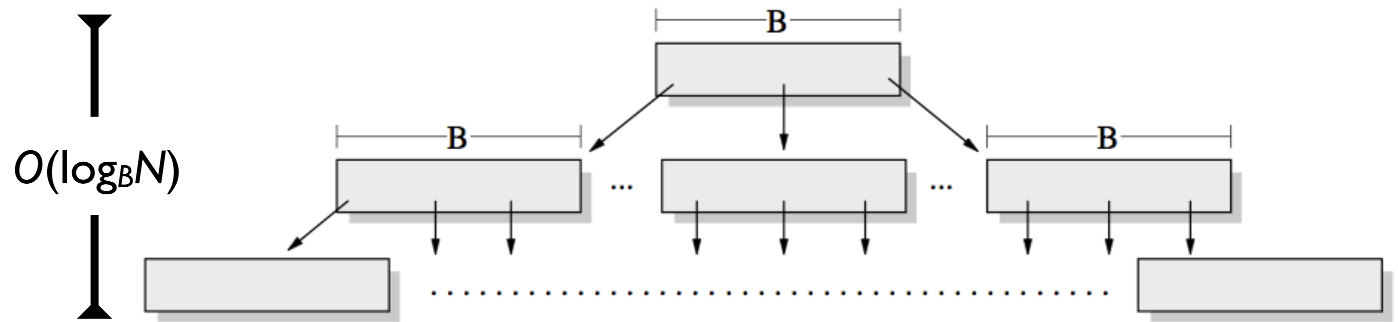


[Aggarwal+Vitter '88]

An algorithmic performance model

B-tree point queries: O($\log_B N$) I/Os.





Write-optimized data structures performance

Data structures: [O'Neil,Cheng, Gawlick, O'Neil 96], [Buchsbaum, Goldwasser, Venkatasubramanian, Westbrook 00], [Argel 03], [Graefe 03], [Brodal, Fagerberg 03], [Bender, Farach,Fineman,Fogel, Kuszmaul, Nelson'07], [Brodal, Demaine, Fineman, Iacono, Langerman, Munro 10], [Spillane, Shetty, Zadok, Archak, Dixit 11].
Systems: BigTable, Cassandra, H-Base, LevelDB, TokuDB.

	B-tree	Some write-optimized structures
Insert/delete	$O(\log_B N) = O\left(\frac{\log N}{\log B}\right)$	$O\left(\frac{\log N}{B}\right)$

- If B=1024, then insert speedup is $B/\log B \approx 100$.
- Hardware trends mean bigger B, bigger speedup.
- Less than 1 I/O per insert.

Optimal Search-Insert Tradeoff [Brodal, Fagerberg 03]

	insert	point query
Optimal tradeoff (function of ε=0...1)	$O\left(\frac{\log_{1+B^{\varepsilon}} N}{B^{1-\varepsilon}}\right)$	$O\left(\log_{1+B^{\varepsilon}} N\right)$
B-tree (ε=1)	$O\left(\log_B N\right)$	$O\left(\log_B N\right)$
ε=1/2	$O\left(\frac{\log_B N}{\sqrt{B}}\right)$	$O\left(\log_B N\right)$
ε=0	$O\left(\frac{\log N}{B}\right)$	$O\left(\log N\right)$

10x-100x faster inserts

Illustration of Optimal Tradeoff [Brodal, Fagerberg 03]

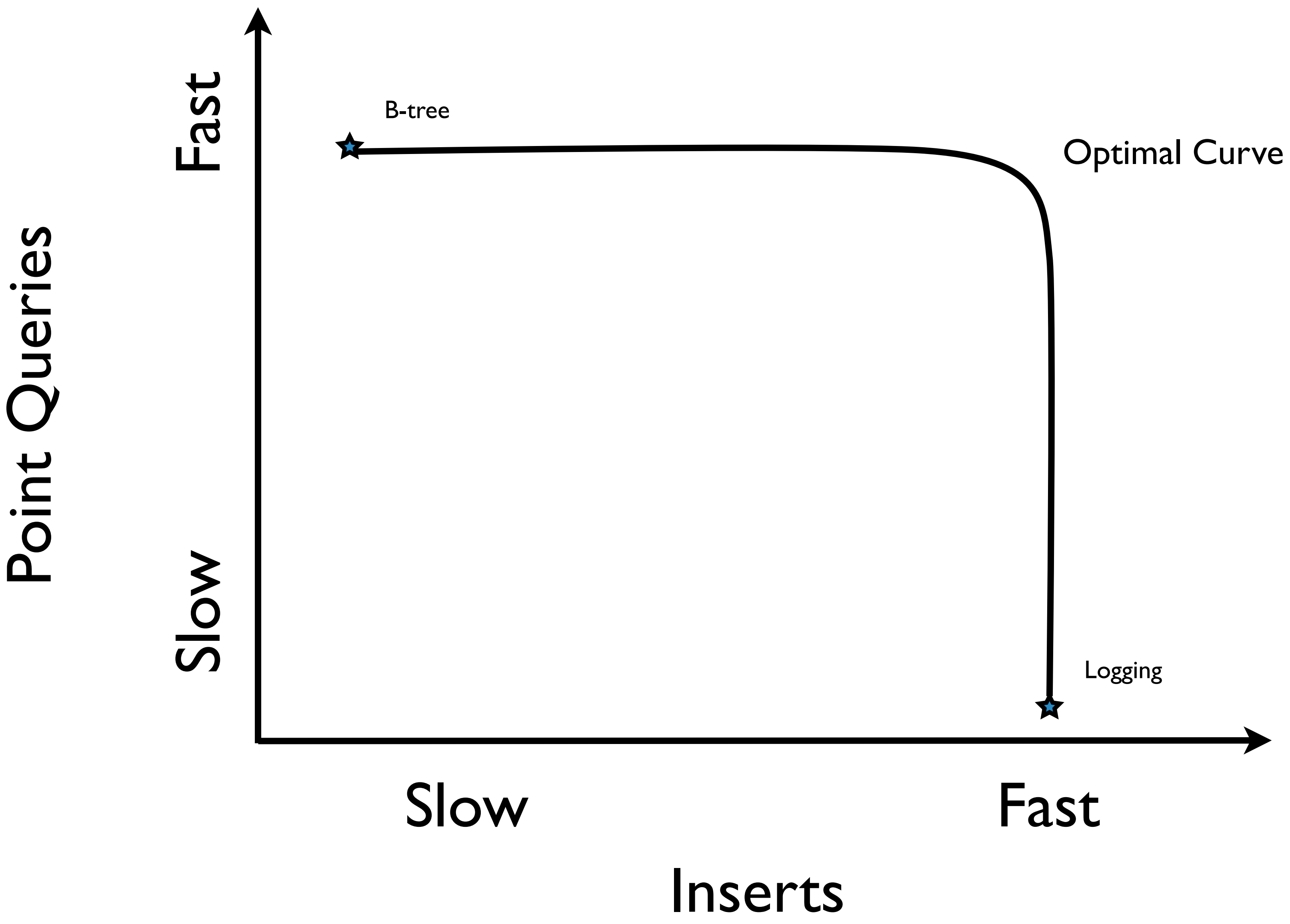


Illustration of Optimal Tradeoff [Brodal, Fagerberg 03]

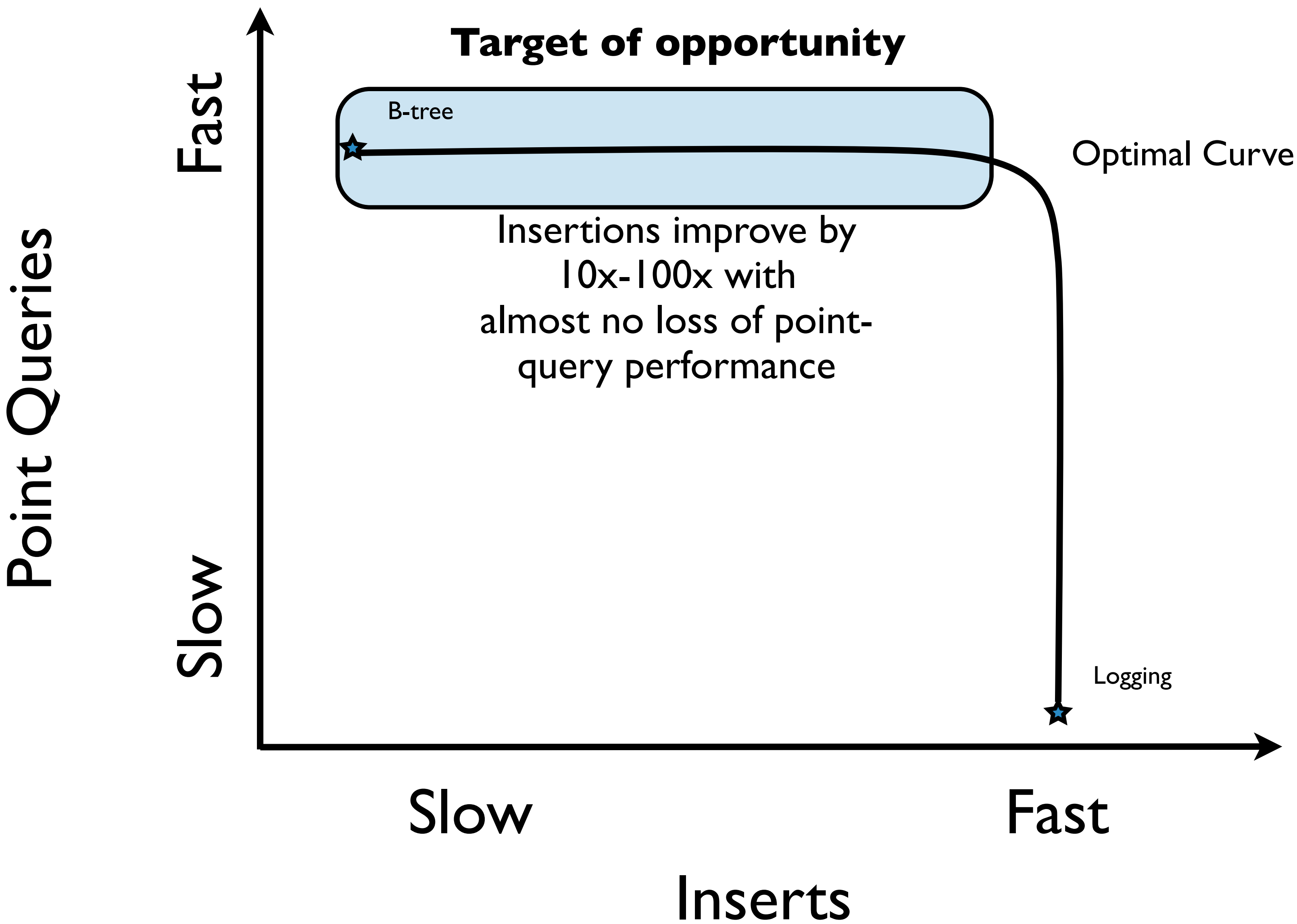
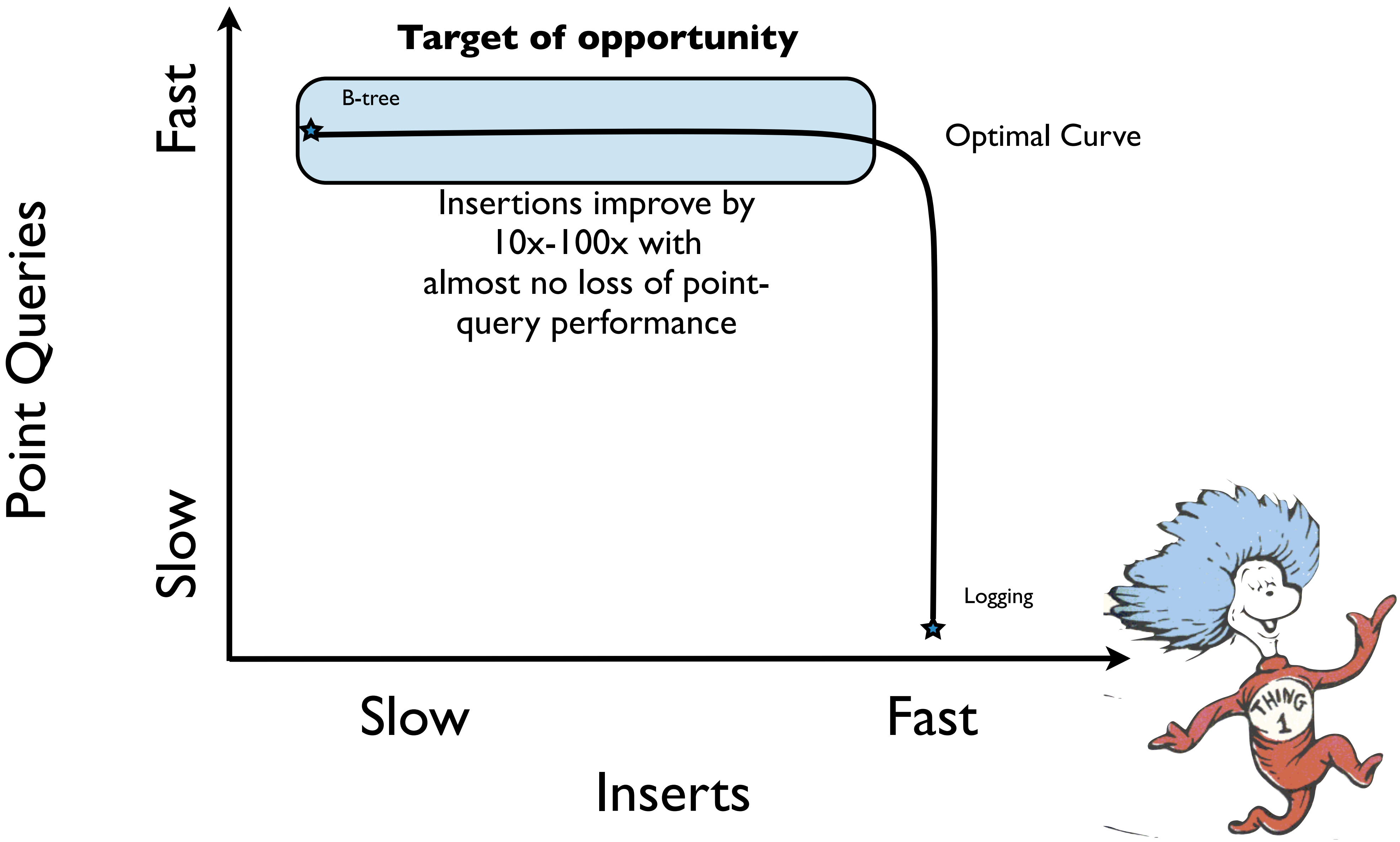


Illustration of Optimal Tradeoff [Brodal, Fagerberg 03]

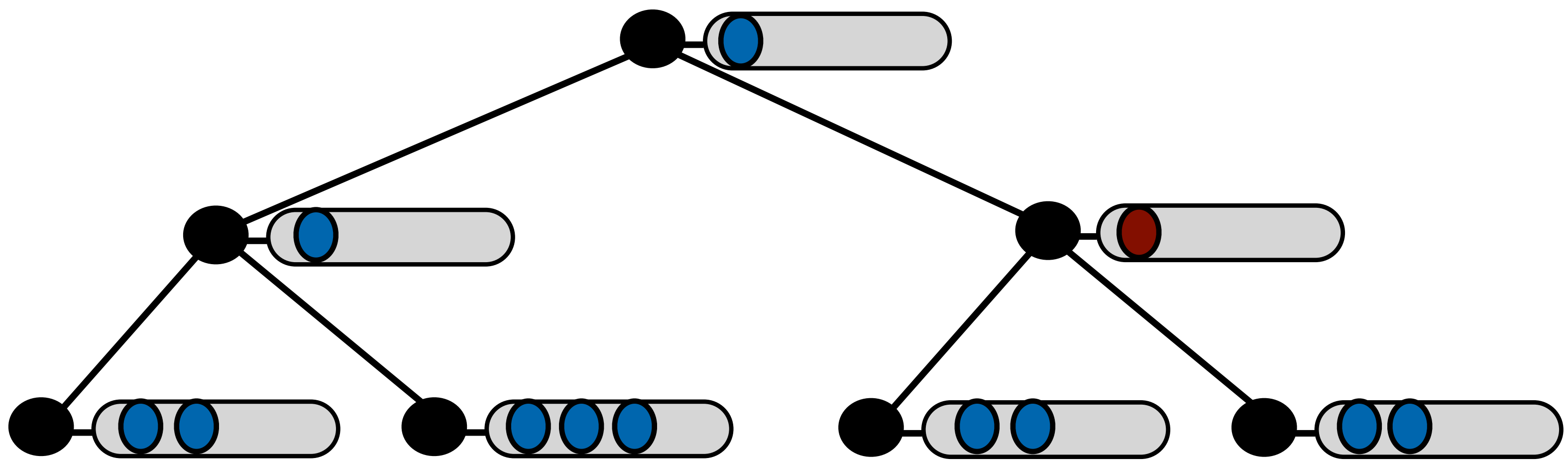


How to Build Write-Optimized Structures

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



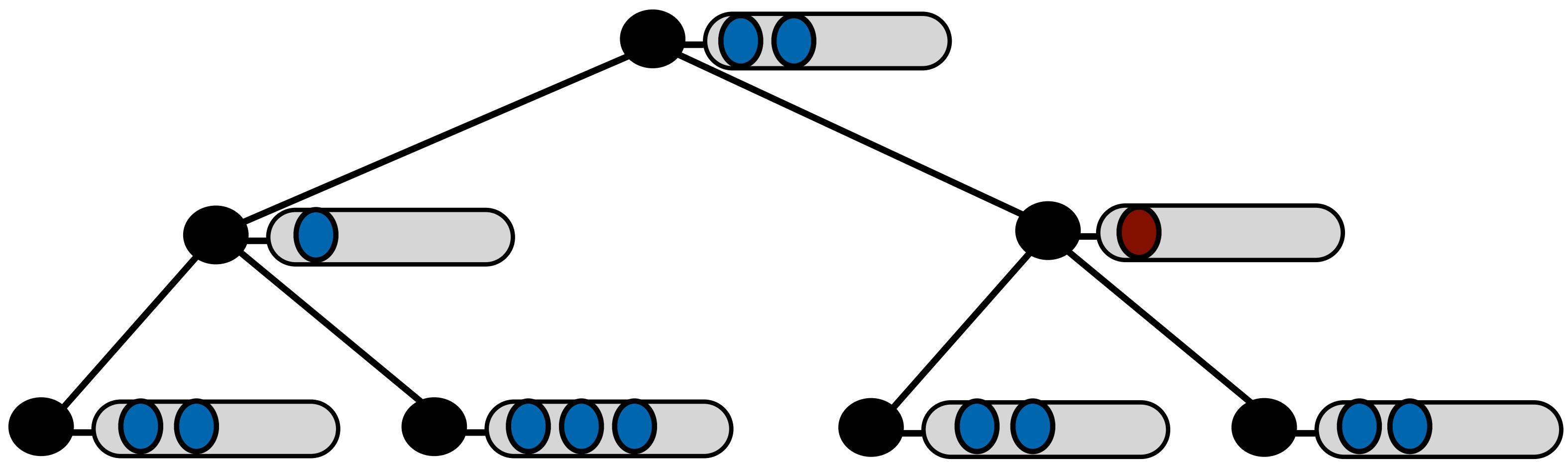
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



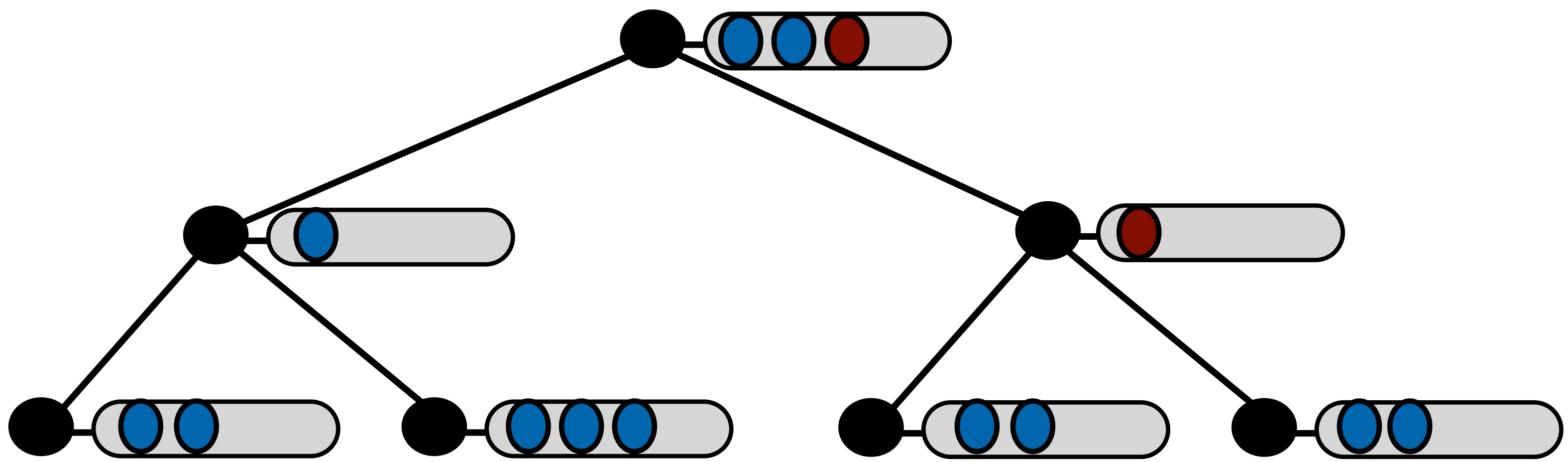
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



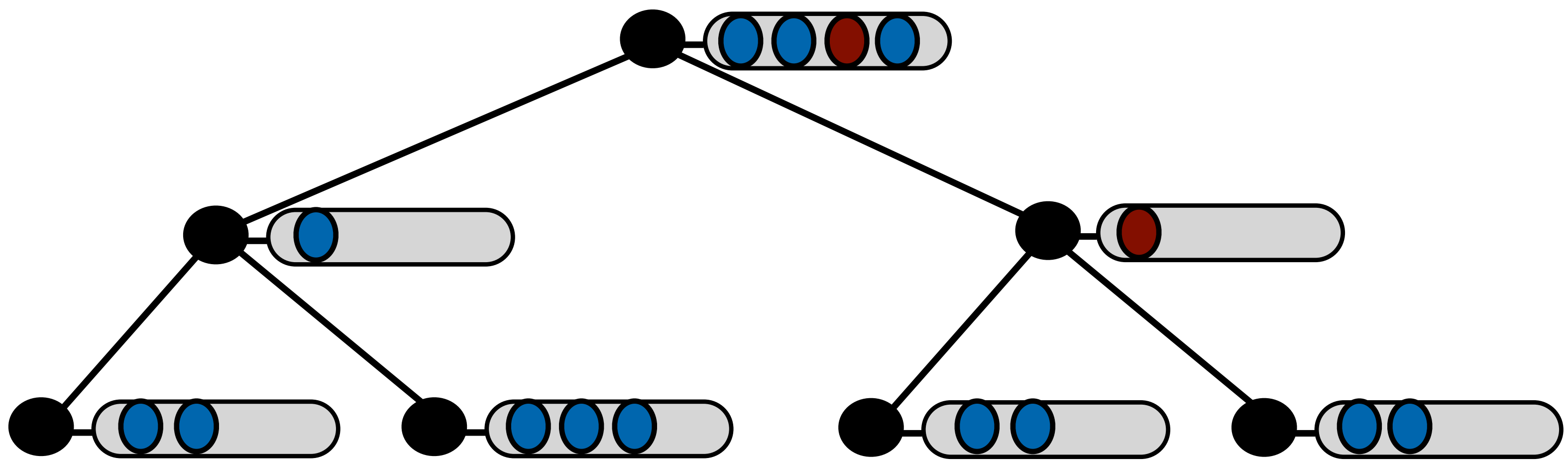
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



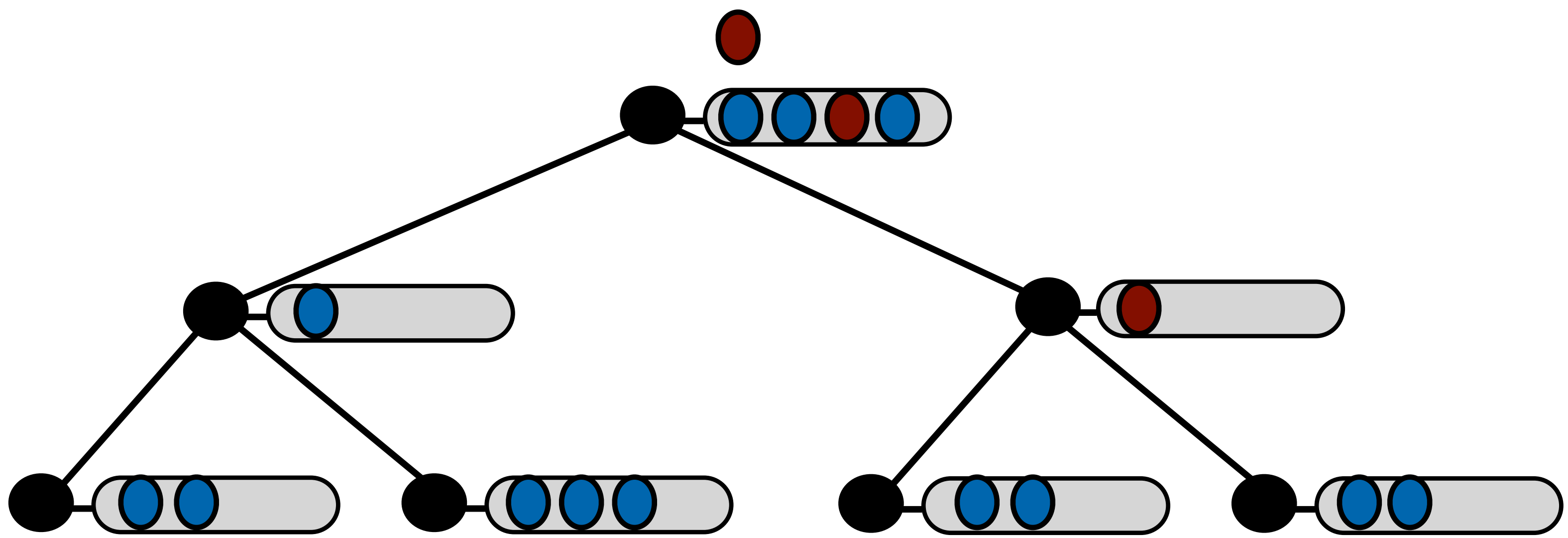
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



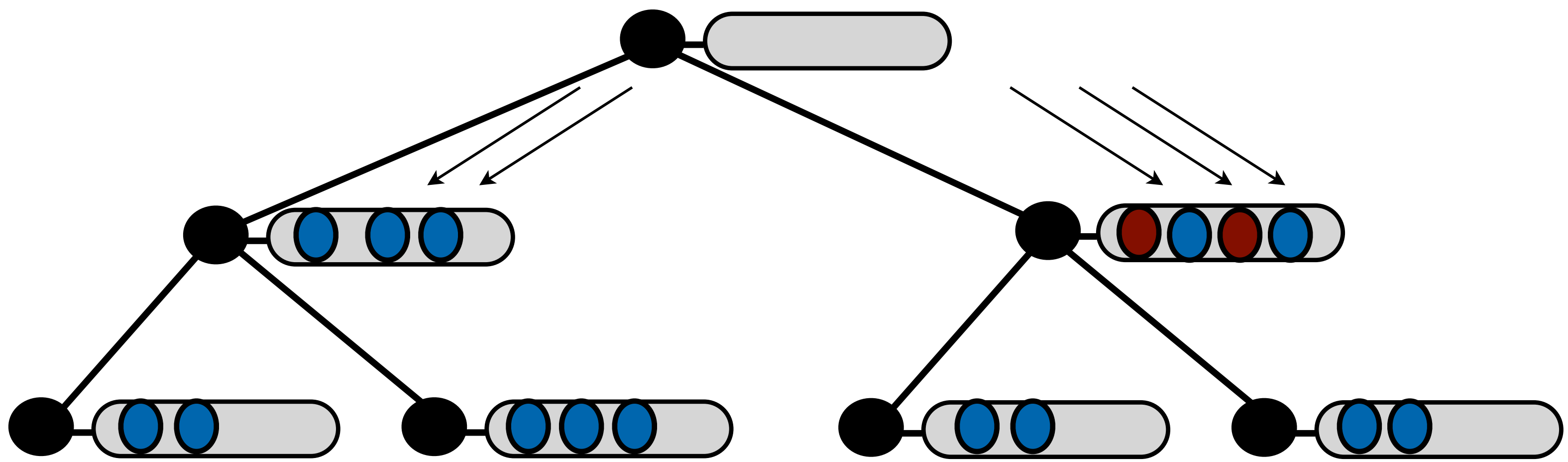
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



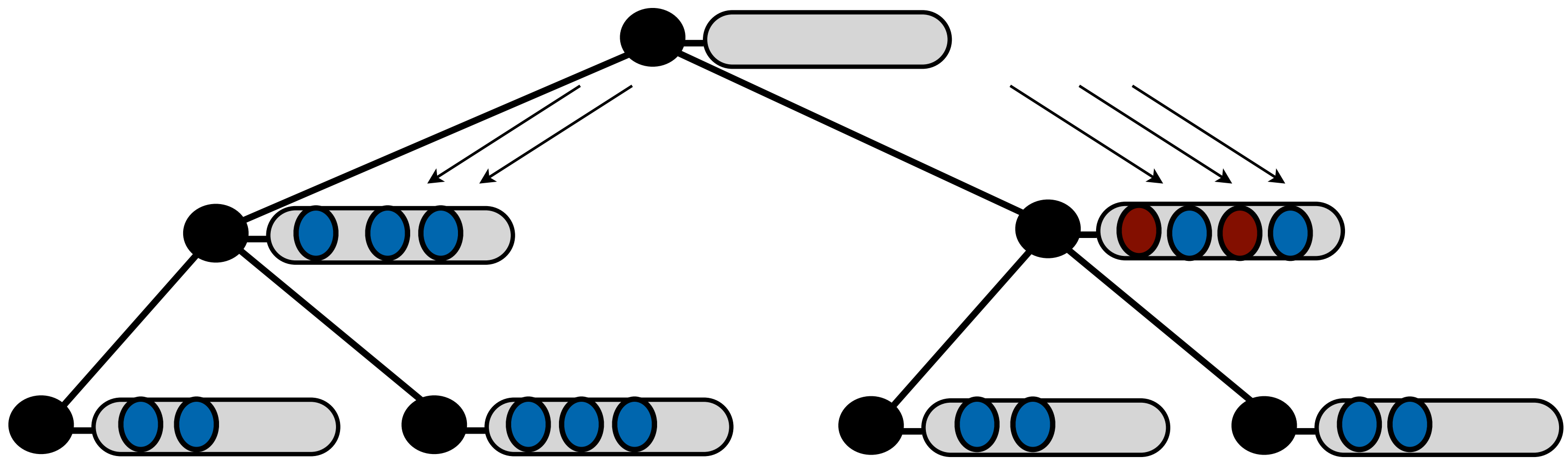
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

Analysis of writes

An insert/delete costs amortized O((log *N*)/B) per insert or delete

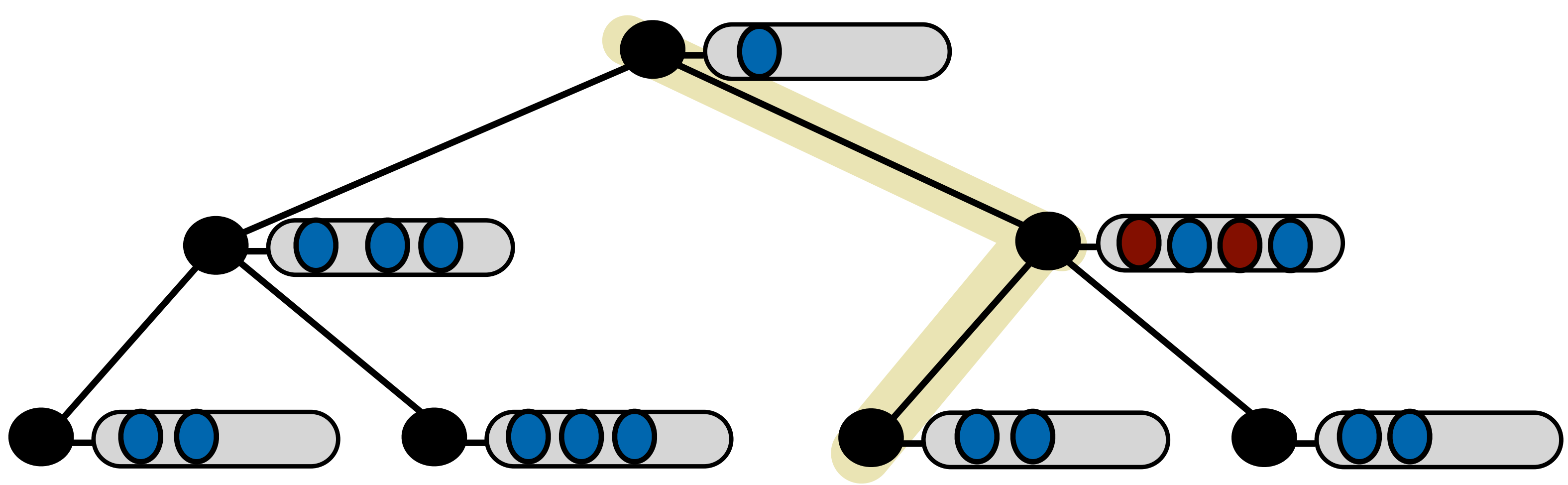
- A buffer flush costs O(1) & sends *B* elements down one level
- It costs O(1/*B*) to send element down one level of the tree.
- There are O(log *N*) levels in a tree.



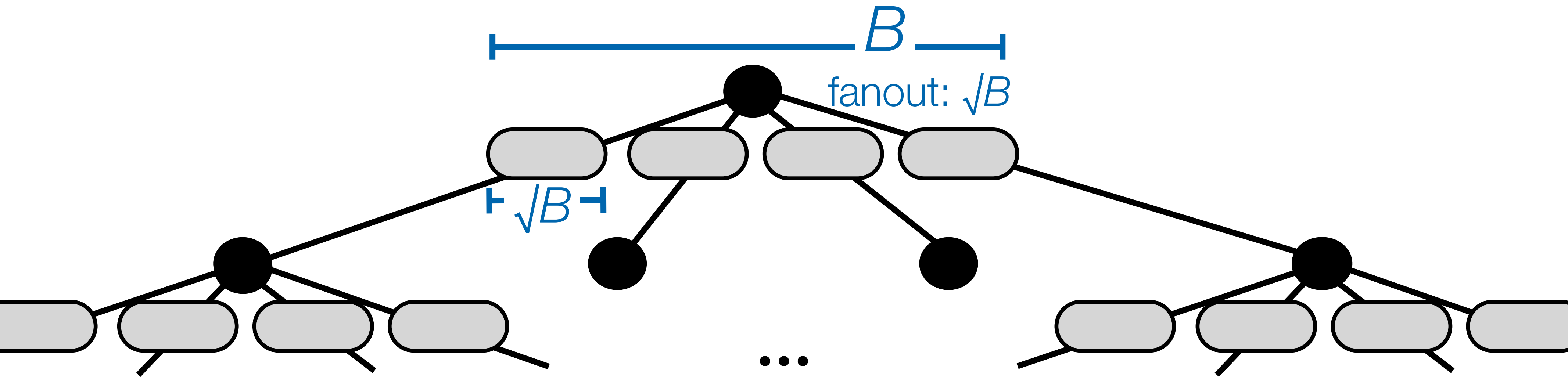
Analysis of point queries

To search:

- examine each buffer along a single root-to-leaf path.
- This costs O(log *N*).



Obtaining optimal point queries + very fast inserts



Point queries cost $O(\log_{\sqrt{B}} N) = O(\log_B N)$

- This is the tree height.

Inserts cost $O((\log_B N)/\sqrt{B})$

- Each flush cost O(1) I/Os and flushes $\sqrt{B}$ elements.

Cache-oblivious write-optimized structures

You can even make these data structures cache-oblivious.

[Bender, Farach-Colton, Fineman, Fogel, Kuszmaul, Nelson, SPAA 07]

[Brodal, Demaine, Fineman, Iacono, Langerman, Munro, SODA 10]

This means that the data structure can be made *platform independent (no knobs)*, i.e., works simultaneously for all values of *B* and *M*.

Cache-oblivious write-optimized structures

You can even make these data structures cache-oblivious.

[Bender, Farach-Colton, Fineman, Fogel, Kuszmaul, Nelson, SPAA 07]

[Brodal, Demaine, Fineman, Iacono, Langerman, Munro, SODA 10]

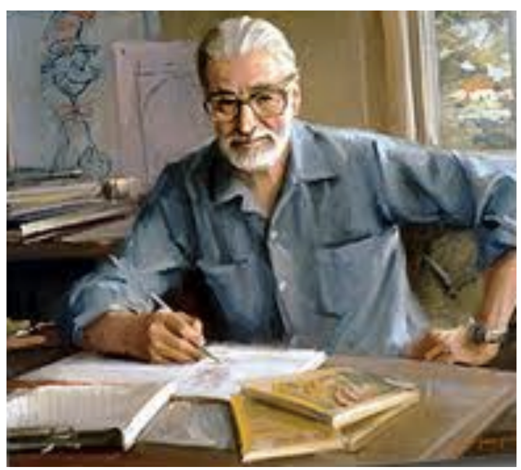
This means that the data structure can be made *platform independent (no knobs)*, i.e., works simultaneously for all values of *B* and *M*.

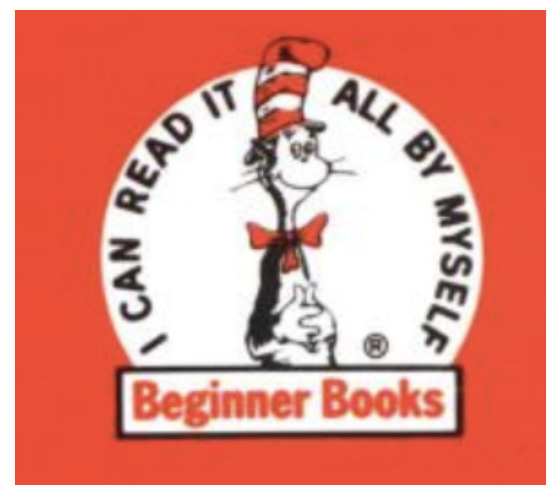
What the world looks like

Insert/point query asymmetry

- Inserts can be fast: >50K high-entropy writes/sec/disk.
- Point queries are necessarily slow: <200 high-entropy reads/sec/disk.

We are used to reads and writes having about the same cost, but writing is easier than reading.

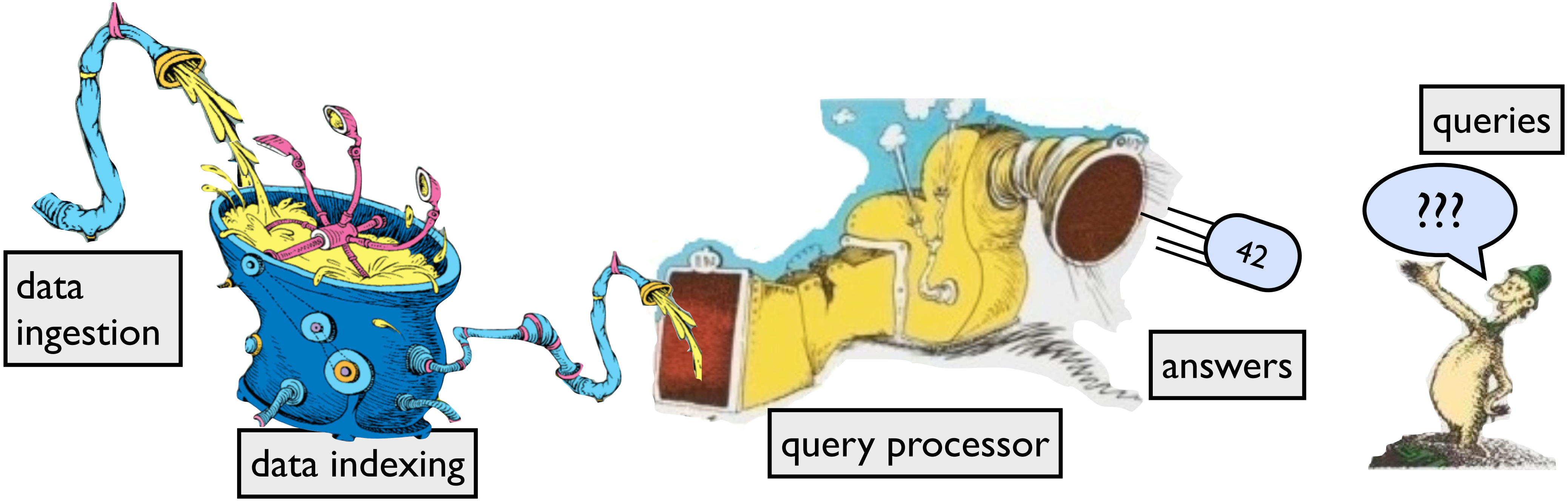




The right read-optimization is write-optimization

The right index makes queries run fast.

- Write-optimized structures maintain indexes efficiently.

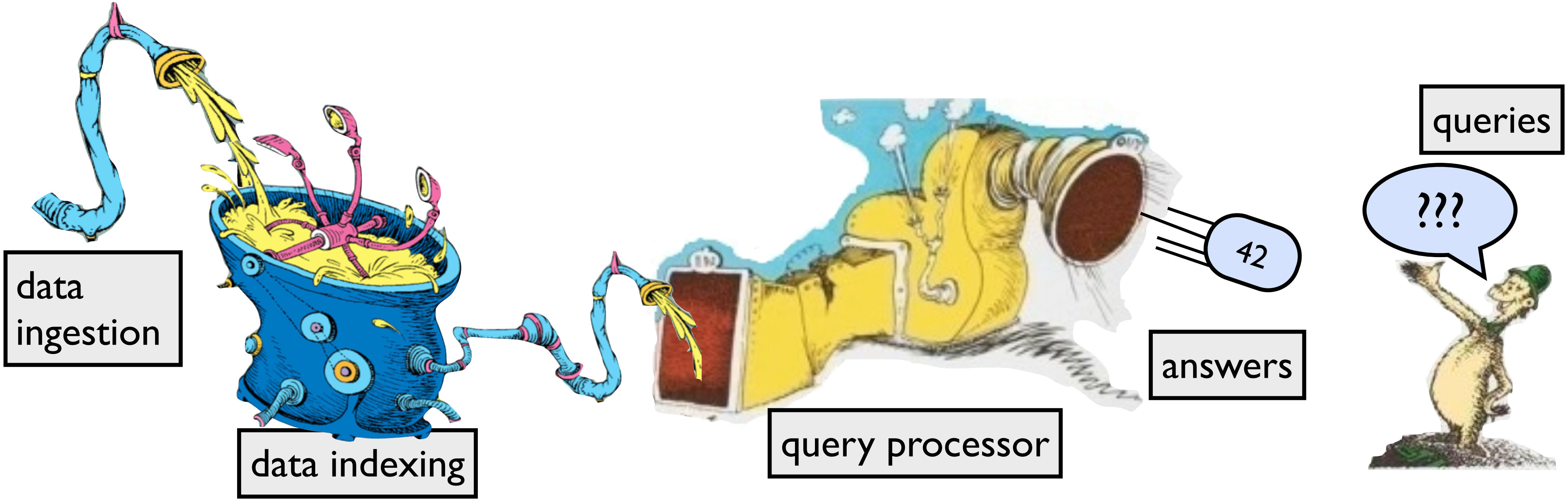


The right read-optimization is write-optimization

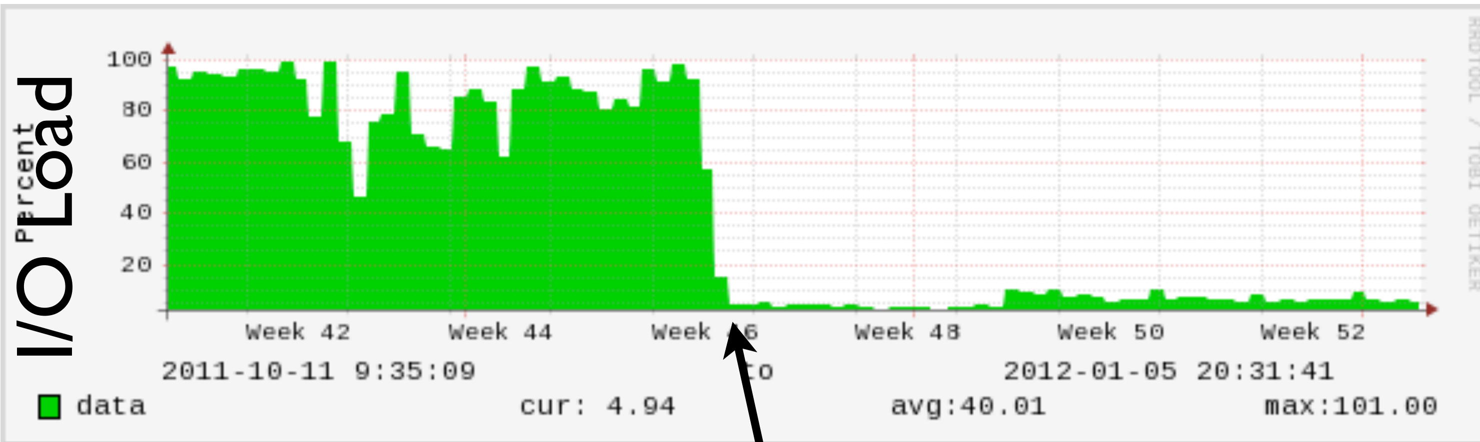
The right index makes queries run fast.

- Write-optimized structures maintain indexes efficiently.

Fast writing is a currency we use to accelerate queries. Better indexing means faster queries.



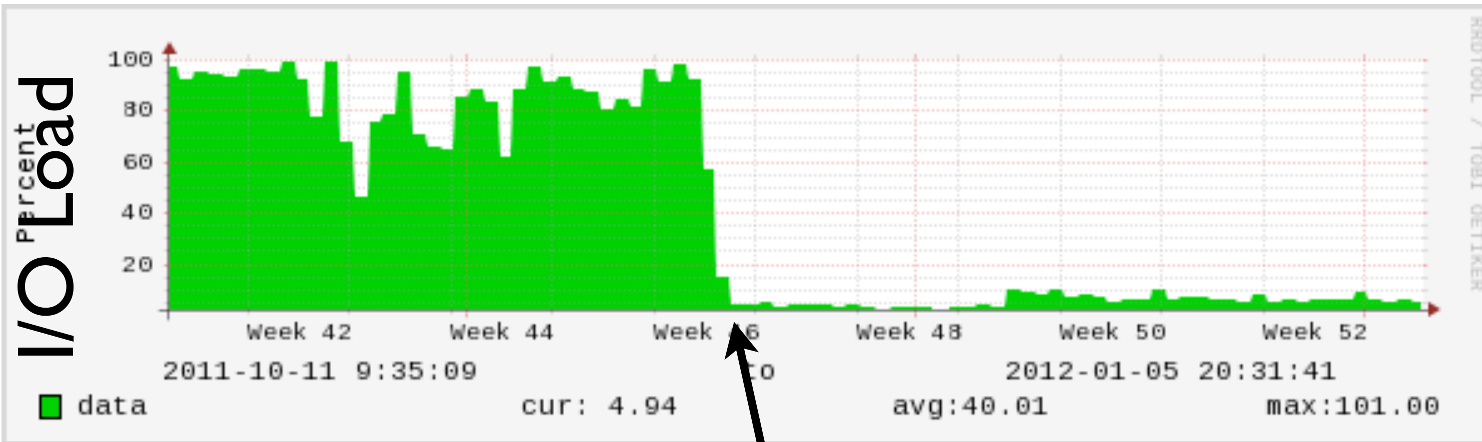
The right read-optimization is write-optimization



Add selective indexes.

(We can now afford to maintain them.)

The right read-optimization is write-optimization



Add selective indexes.

(We can now afford to maintain them.)

Write-optimized structures can significantly mitigate the insert/query/liveness tradeoff.

Write-optimization also helps file systems

HEC FSIO Grand Challenges

Store 1 trillion files

Create tens of thousands of files per second

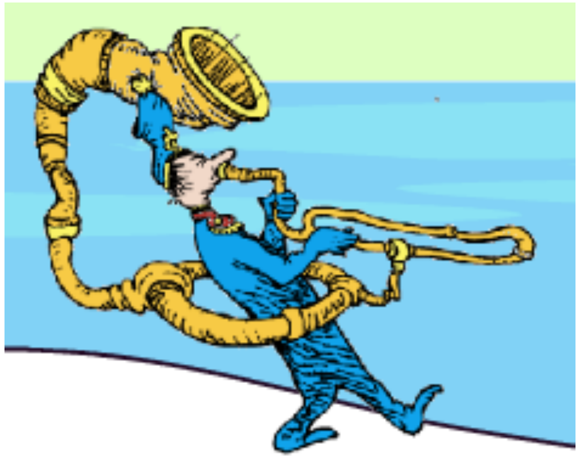
Traverse directory hierarchies fast (`ls -R`)

B-trees would require at least hundreds of disk drives.

TokuFS

TokuFS [Esmet, Bender, Farach-Colton, Kuszmaul HotStorage12]

- A file-system prototype
- >20K file creates/sec
- very fast `ls -R`
- HEC grand challenges on a cheap disk



Write-optimization going forward

Example: Time to fill a disk in 1973, 2010, 2022.

- log high-entropy data sequentially versus index data in B-tree.

Year	Size	Bandwidth	Access Time	Time to log data on disk	Time to fill disk using a B-tree (row size 1K)
1973	35MB	835KB/s	25ms	39s	975s
2010	3TB	150MB/s	10ms	5.5h	347d
2022	220TB	1.05GB/s	10ms	2.4d	70y

Better data structures may be a luxury now, but they will be essential by the decade's end.

Summary of Talk

Write-optimization changes the relative difficulty of database operations.

- There is a provable point-query insert tradeoff. We can insert 10x-100x faster without hurting point queries.
- We can avoid much of the funny tradeoff between data ingestion, liveness, and query speed.
- We can avoid knobs.
- File systems also benefit.