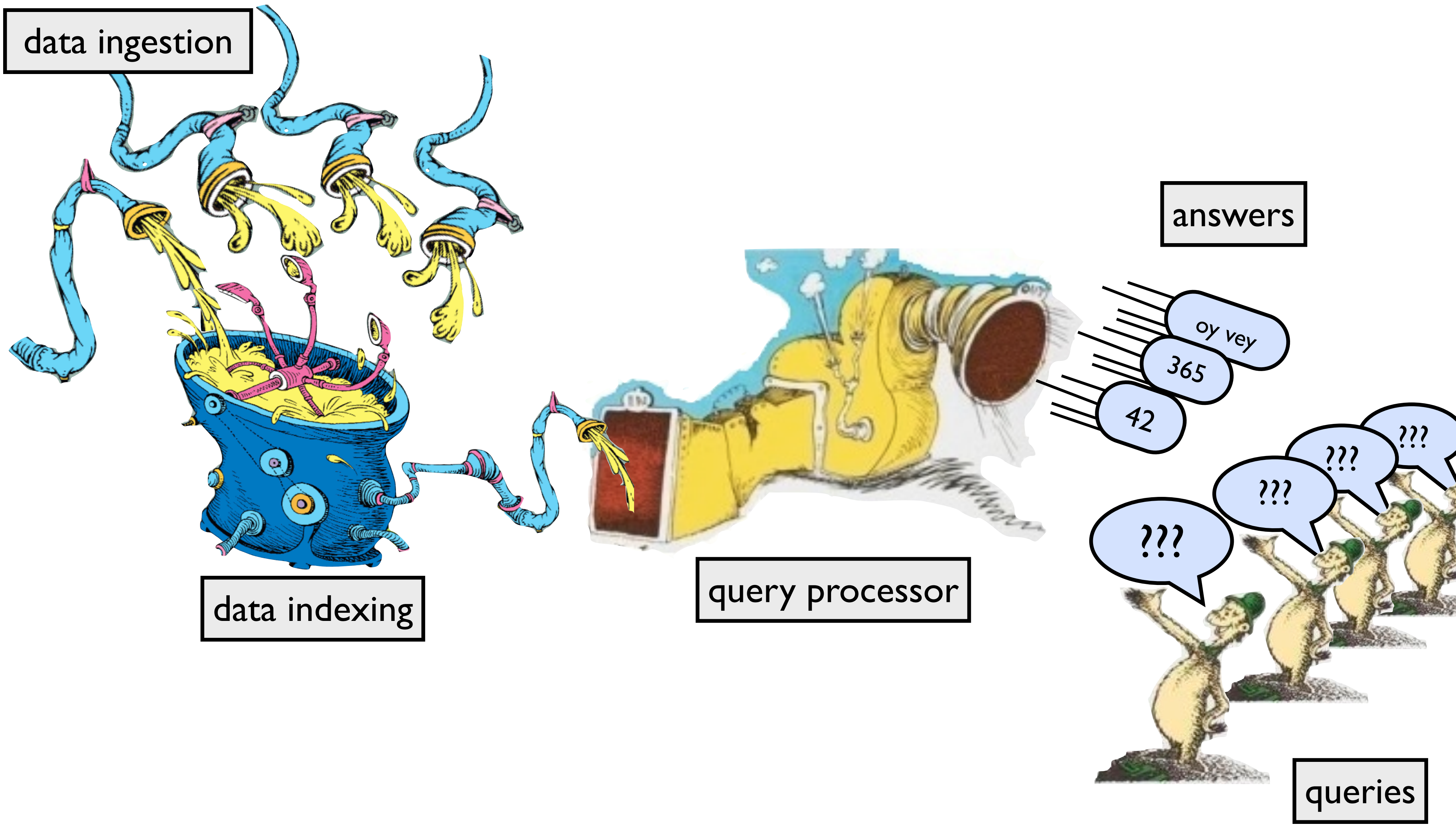
Indexing Big Data

Michael A. Bender



The problem with big data is microdata

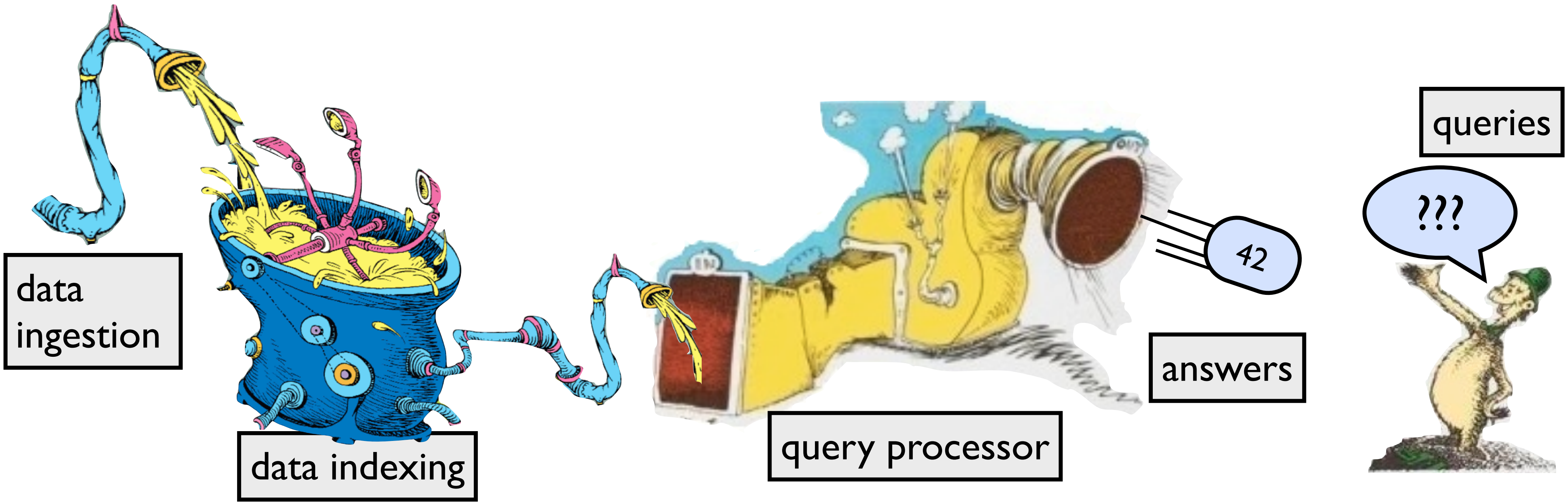
Microdata is small data.

The most intractable data consists of lots of small pieces.

Microdata is everywhere

Metadata≈microdata.

- Even when data has big chunks, metadata is small.



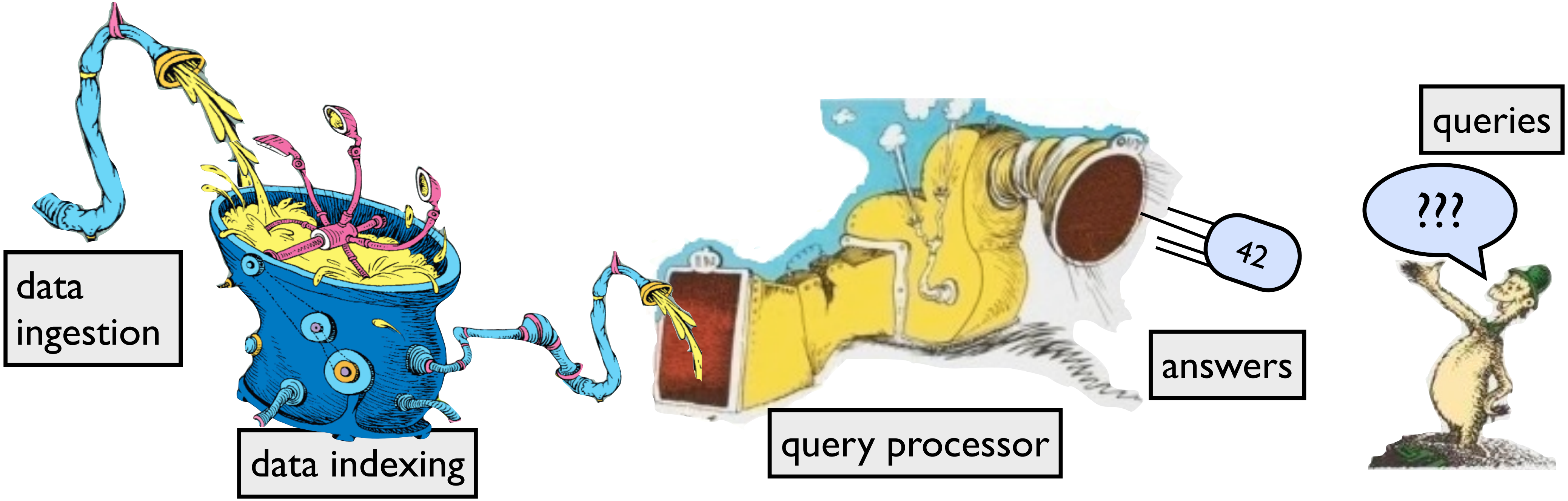
Microdata is everywhere

Metadata≈microdata.

- Even when data has big chunks, metadata is small.

Small files in parallel/cloud file systems.

- HPC workloads use the names of many small files, e.g., to describe experiment metadata.



Microdata is everywhere

Metadata≈microdata.

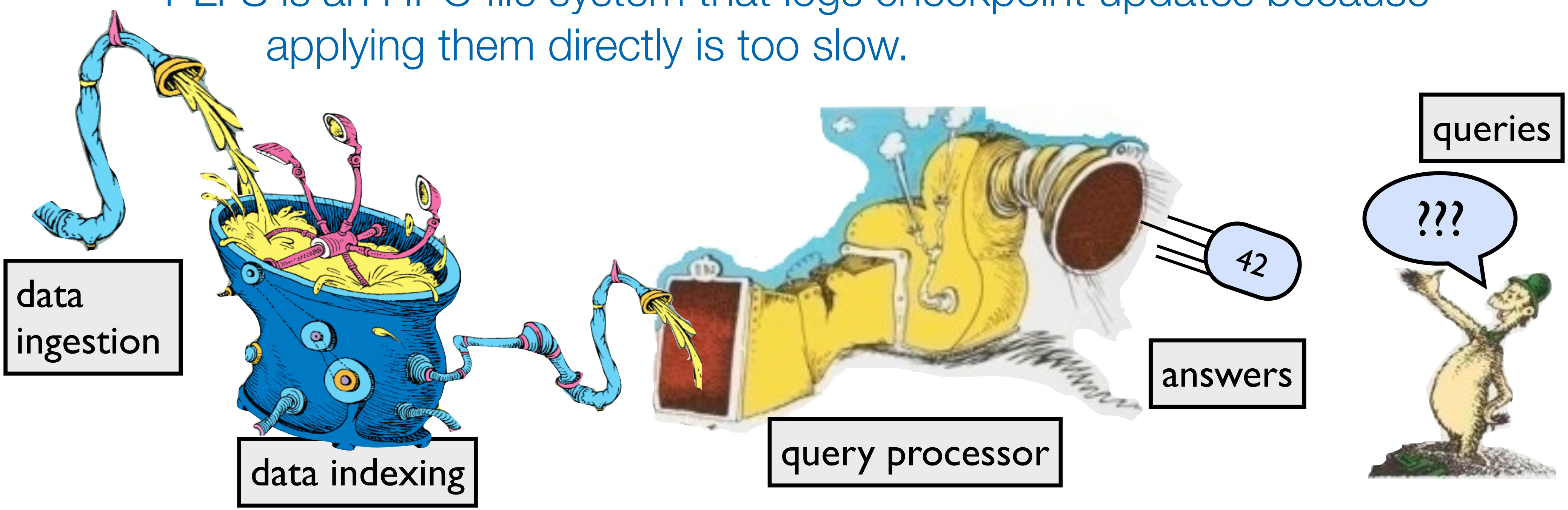
- Even when data has big chunks, metadata is small.

Small files in parallel/cloud file systems.

- HPC workloads use the names of many small files, e.g., to describe experiment metadata.

Small updates inside large files.

- PLFS is an HPC file system that logs checkpoint updates because applying them directly is too slow.



The microdata problem is getting worse

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

- log 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.

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.

HEC FSIO Grand Challenges

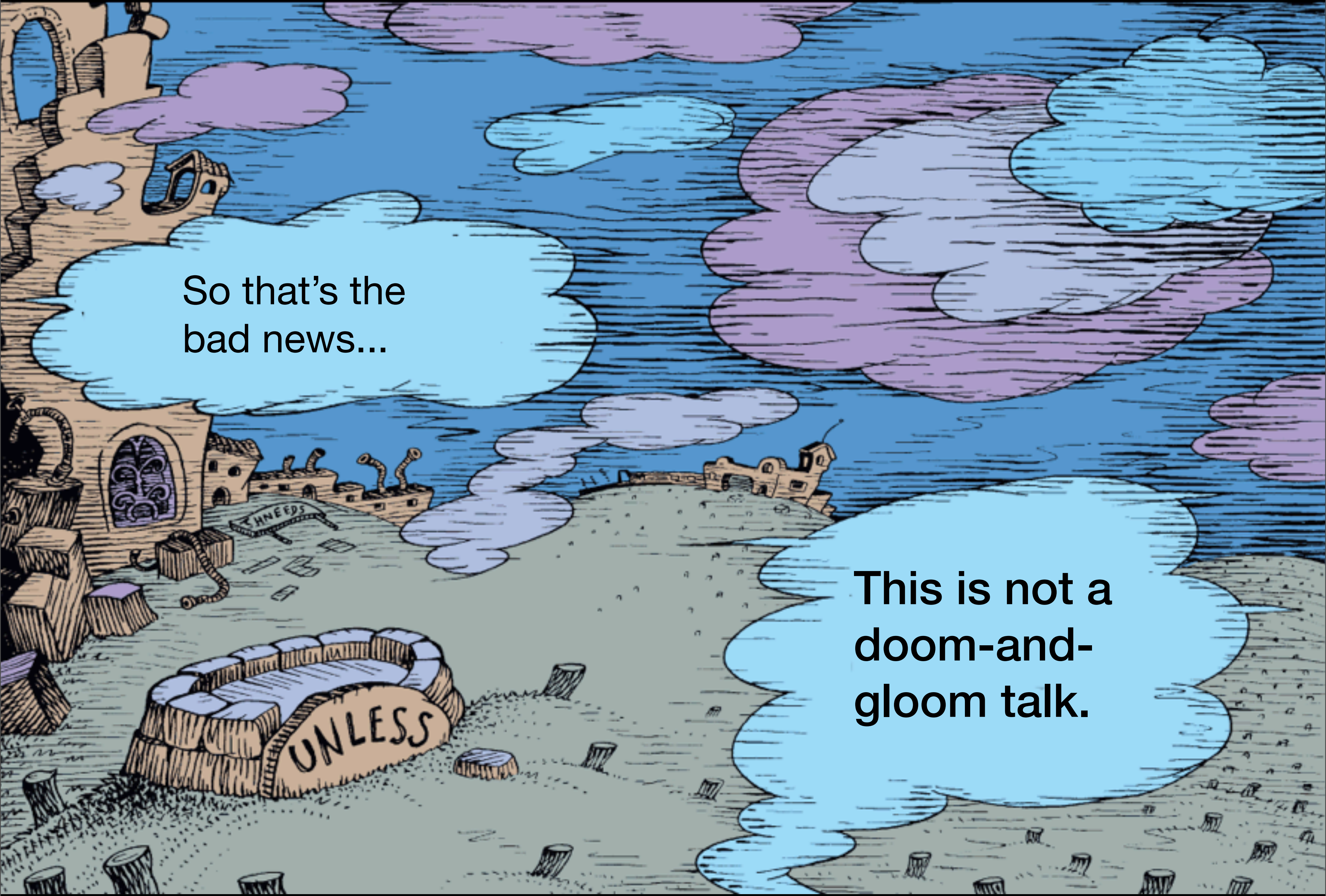
Store 1 trillion files

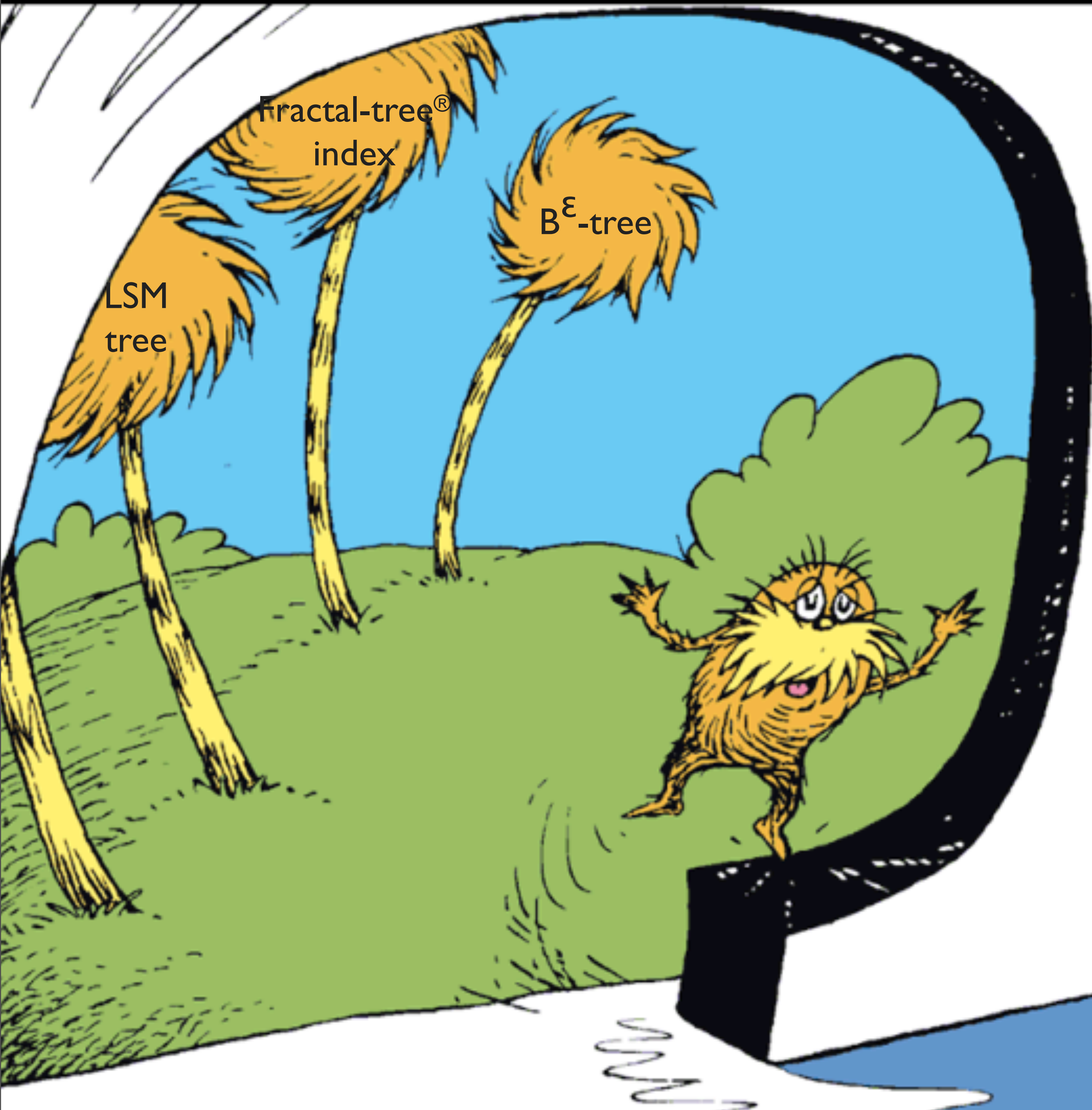
Create tens of thousands of files per second

Traverse directory hierarchies fast (`ls -R`)

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

Parallel computing is about high performance. To get high performance, we need high-performance I/O.


So that’s the bad news...
This is not a doom-and-gloom talk.
THNEEDS
UNLESS



We ***can*** solve the microdata problem with the right kind of data structures + fundamental research.

Write-optimized data structures can help

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 because microdata is transferred in parallel in data blocks.

Optimal Search-Insert Tradeoff [Brodal, Fagerberg 03]

	insert	point query
Optimal tradeoff (function of ε)	$O\left(\frac{\log_{1+B^\varepsilon} N}{B^{1-\varepsilon}}\right)$	$O\left(\log_{1+B^\varepsilon} N\right)$
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]

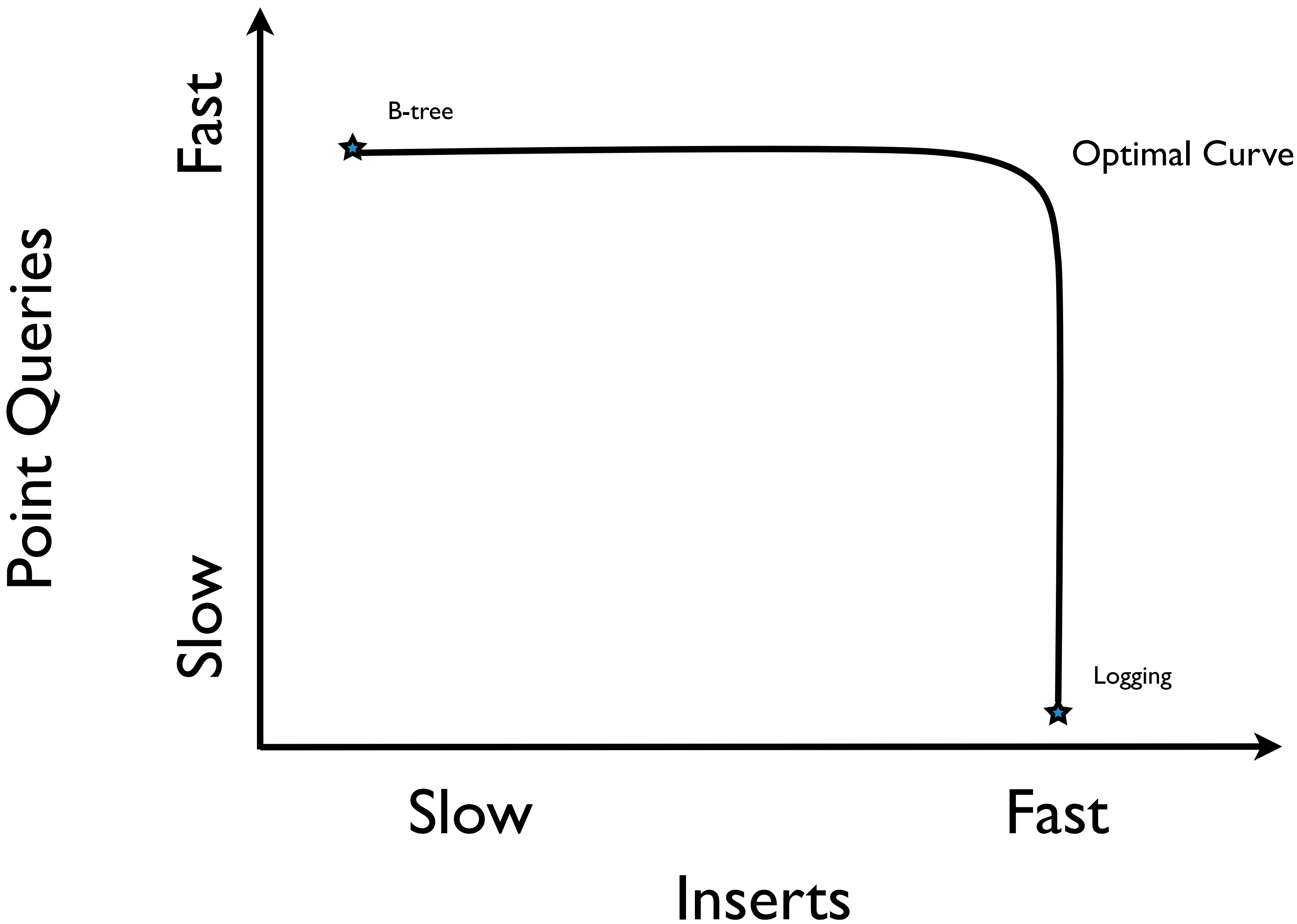
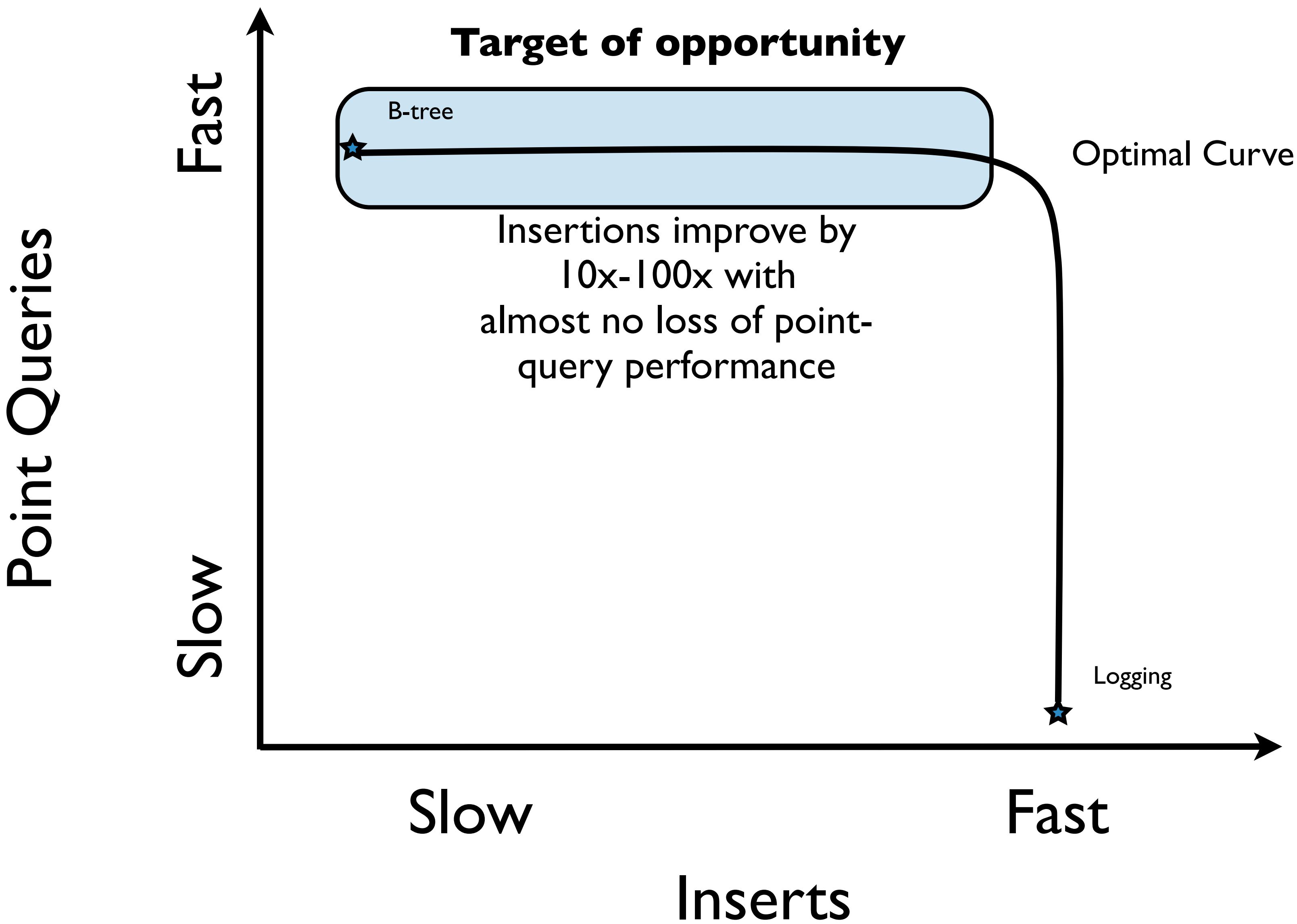


Illustration of Optimal Tradeoff [Brodal, Fagerberg 03]



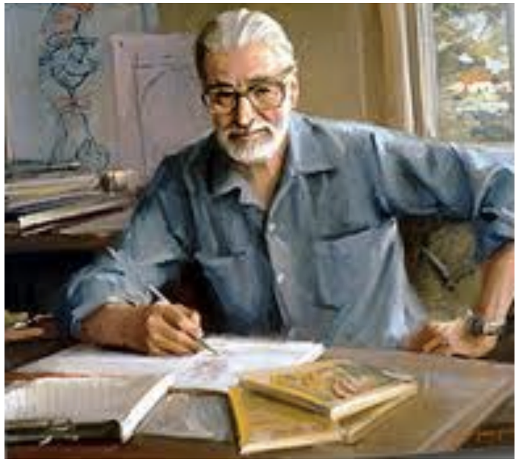
What the world looks like and research directions

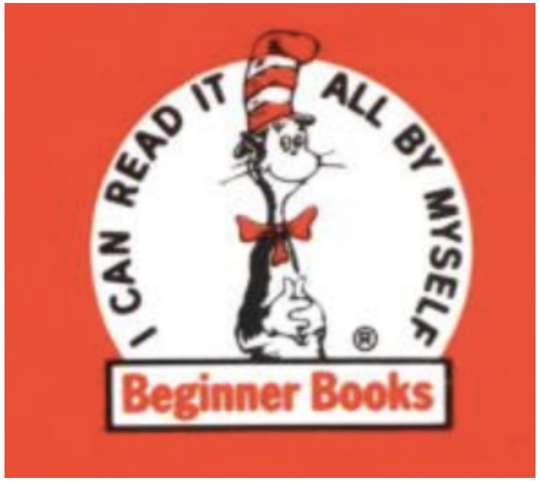
What the world looks like

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.



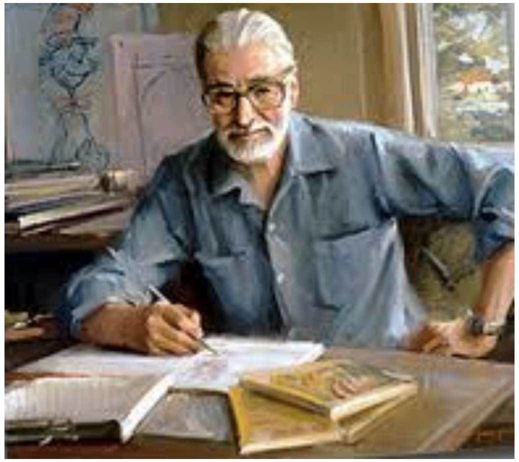


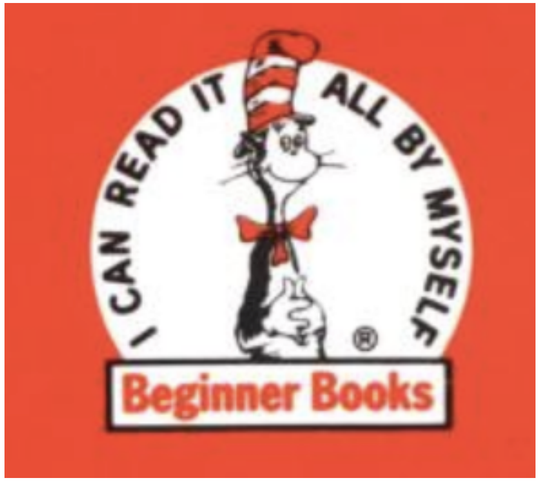
What the world looks like

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.



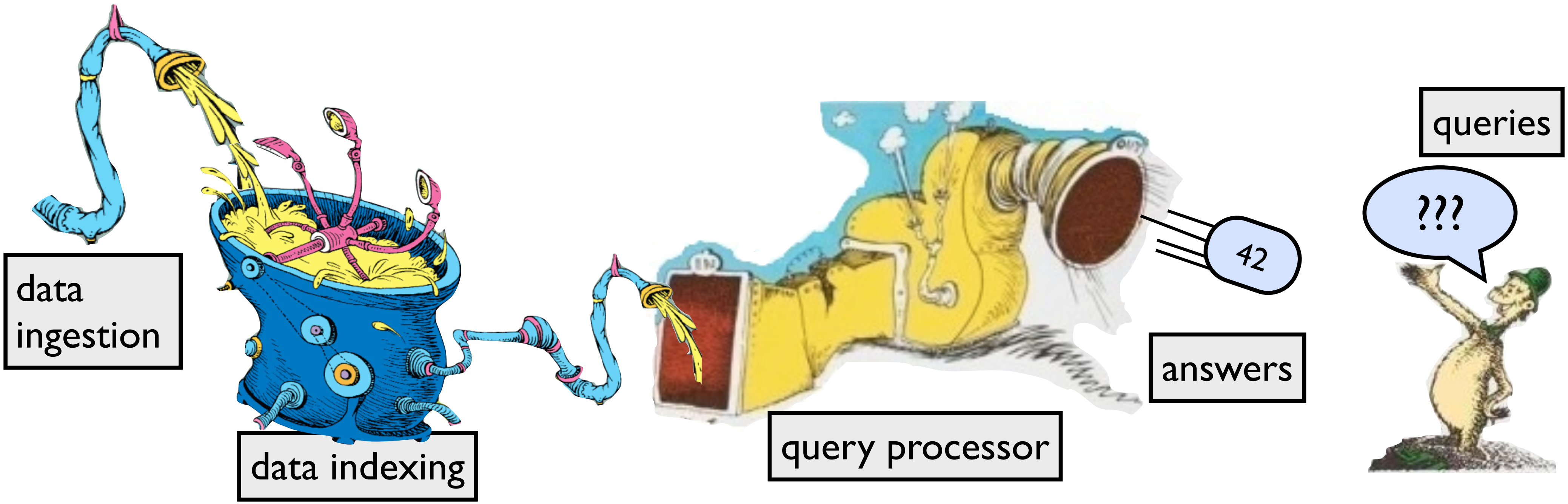


How can we revisit system design in light of this asymmetry?

The best read-optimization is write-optimization

The right index (e.g., database index=data ordering) makes queries run fast.

- E.g., *find* or *spotlight* on a file system can be fast if the data is indexed correctly.
- Write-optimized structures maintain indexes efficiently.

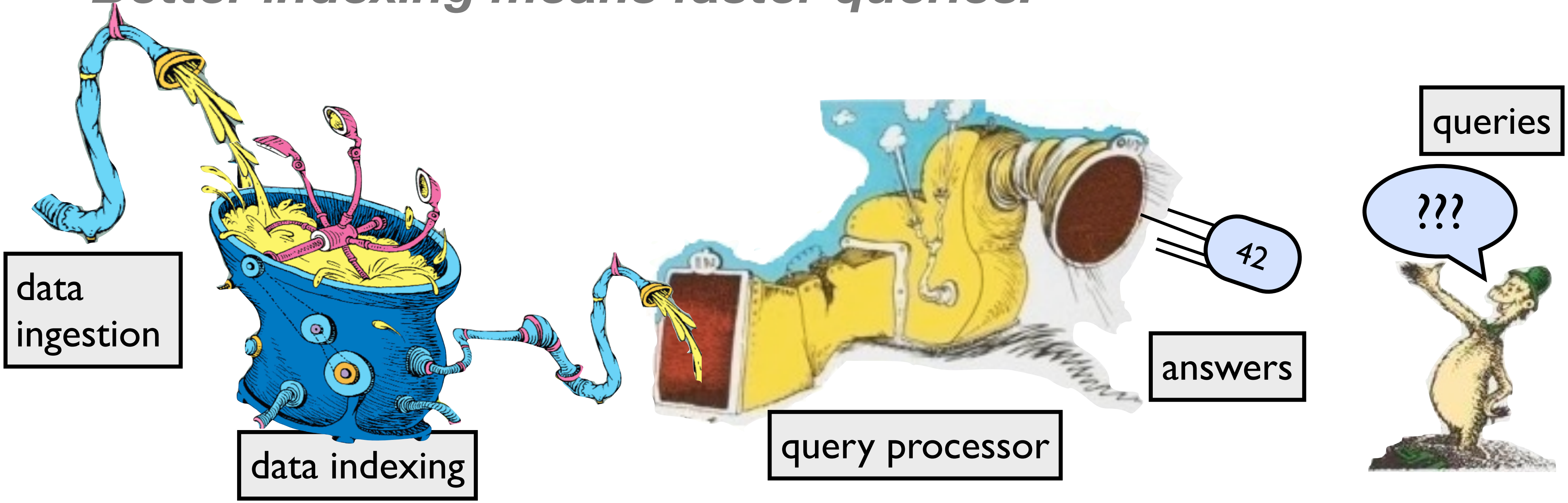


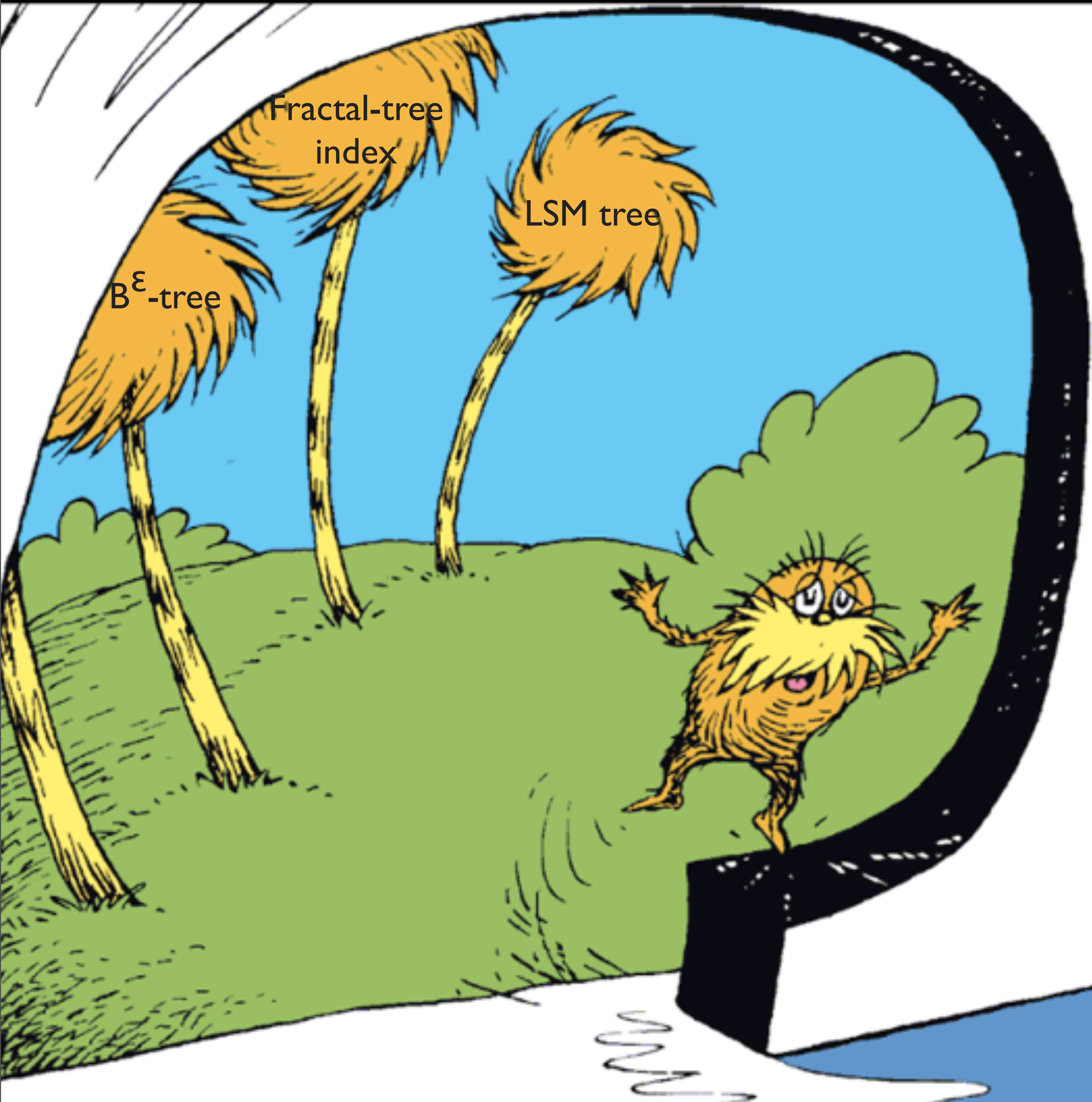
The best read-optimization is write-optimization

The right index (e.g., database index=data ordering) makes queries run fast.

- E.g., *find* or *spotlight* on a file system can be fast if the data is indexed correctly.
- Write-optimized structures maintain indexes efficiently.

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





Challenge

Replace traditional structures (e.g., B-trees), with write-optimized structures in parallel/cloud file systems and end-end databases.

Our progress building write-optimized systems

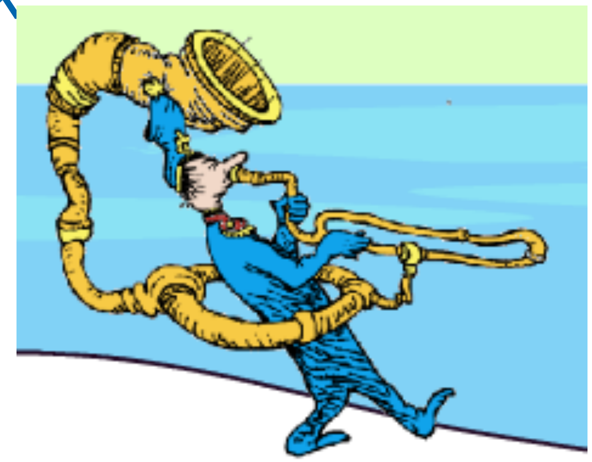
TokuDB

- A write-optimized storage engine for the MySQL
- Commercial product from Tokutek

TokuFS

[Esmet, Bender, Farach-Colton, Kuszmaul 12]

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



Systems often assume search cost = insert cost

Some inserts/deletes have hidden searches.

Example:

- return error when a duplicate key is inserted.
- return # elements removed on a delete.

These “cryptosearches” throttle insertions down to the performance of B-trees.

- Write-optimized data structures run quickly because inserts don’t put elements directly in their final positions, so that you can insert without incurring a disk I/O, even in the worst case and for large data sets.

Redesign systems in light of search/point query asymmetry

Redesign systems to avoid cryptosearches.

- *Design new algorithms for concurrency control, transactional mechanisms, crash safety, etc, in light of the new performance characteristics.*
- *Redesign existing APIs.*

Some cryptosearches, cannot be avoided.

- *How can we mitigate their damage (e.g., Bloom filters, SSDs)?*

Highly parallel/concurrent write-optimized structures

Write-optimized structures are CPU-bound even for workloads on which traditional storage systems are I/O bound.

Need highly concurrent, multithreaded write-optimized data structures.

Summary of Talk

Microdata in parallel and high-end computing:

- We need high-performance I/O.

How foundational parallelism research helps:

- Better write-optimized data structures.
- Lower bounds.
- Algorithms for concurrency control, transactions.
- Multithreading + high concurrency.