Our second case study: What are we paying for?

A core transactional ledger for a commodities exchange. The system handles intense point-lookups for absolute real-time account balances and date-range auditing. It demands highly predictable, sub-millisecond read responses.

This is the natural home territory of a B-Tree engine, like PostgreSQL. Because it lacks background compaction cascades, it provides highly deterministic read latencies without erratic tail-latency spikes.

Let's think about the physical cost here. Since B-Trees rely on in-place updates and cause random page splitting over time, we inevitably face internal fragmentation where pages end up sitting half-empty.

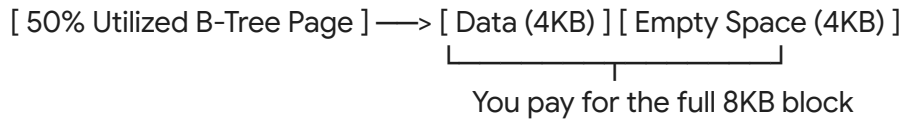
Question:
If our index pages drop to 50% physical utilization across terabytes of data, how does this structural layout inefficiency directly inflate our cloud-managed storage hosting bills? What are we paying for?

When your B-Tree index pages drop to 50% physical utilization—a phenomenon often called **index bloat** or internal fragmentation—you are effectively running into a structural layout inefficiency where every gigabyte of actual data requires two gigabytes of physical disk storage.

In a cloud-managed environment (like AWS EBS, GCP Persistent Disks, or managed database services like AWS Aurora and Google Cloud SQL), this physical layout inefficiency directly inflates your hosting bills in three distinct, costly ways.

1. You Pay for "Dead Space" Storage Allocation

Cloud storage is billed on **provisioned capacity** (or consumed storage blocks in serverless models), not on the volume of meaningful application data you store.
If you have 5 TB of actual commodity trading ledger data, but your B-Tree pages are only 50% full due to aggressive random page splits, your database physically occupies 10 TB of disk space



- **The Bill Impact:** You are directly paying a 2x premium on raw storage volume ($/GB/month). For enterprise-grade SSDs (like AWS gp3 or io2), this structural inefficiency silently burns hundreds or thousands of dollars a month just to host empty disk bytes.

2. The Provisioned IOPS and Bandwidth Tax

As we established with how disk I/O works, data is transferred between disk and memory in fixed block sizes (typically 8KB in PostgreSQL). The storage controller does not care if a page is full or empty; it must read or write the entire block.
When your pages are only 50% full, the **information density** of each I/O operation drops by half.

- To scan a date range containing 1,000 ledger records, a pristine 100% utilized index might require reading 10 pages from disk.
- A 50% fragmented index will require reading **20 pages** from disk to fetch the exact same 1,000 records.
- **The Bill Impact:** In the cloud, high-performance database workloads require you to provision and pay for a specific tier of IOPS and Throughput (e.g., AWS EBS io2 or gp3 provisioned performance). Because your database has to perform **twice as many physical disk operations** to achieve the same business objective, you are forced to over-provision your storage performance tier, dramatically driving up your infrastructure costs.

3. The Downstream Memory (Buffer Pool) Inflation

This is the hidden tax that often costs more than the storage itself. Databases like PostgreSQL do not read data directly from disk for every query; they pull pages into an in-memory buffer pool (RAM) and keep them there for fast access.
Because the unit of memory management is *also* the database page, a 50% utilized page takes up exactly the same amount of precious, expensive RAM as a 100% utilized page.

- **The Bill Impact:** Your buffer pool becomes severely diluted with empty space. To maintain your strict requirement for sub-millisecond, highly deterministic read latencies, your active working set must fit in memory. If your indexes are heavily bloated, you will experience a high cache-miss ratio, forcing you to vertically scale your database instance to a larger tier (e.g., moving from an r6g.2xlarge to an r6g.4xlarge) simply to acquire more RAM. Cloud memory-optimized compute instances are among the most expensive line items on a cloud bill.

The Compounding Ledger Tax

Beyond the core database instance, internal fragmentation creates a cascading cost effect across your entire cloud ecosystem:

- **Backups and Snapshots:** Managed snapshots (like AWS EBS snapshots) capture the block-level state. While changed-block tracking helps, your initial and absolute baseline

backup sizes are twice as large as necessary.

- **Replication Bandwidth:** In cross-region or multi-AZ high-availability setups, those fragmented pages, page splits, and WAL entries must be mirrored across the network, potentially incurring higher cross-AZ or inter-region data transfer fees.

Summary of What You Are Paying For

In a heavily fragmented B-Tree system, your cloud bill doesn't just reflect the size of your business data. You are actively subsidizing **the cost of hosting, reading, writing, and caching empty air.**