facebook

facebook

Facebook News Feed

Social data at scale

Serkan Piantino
Site Director, Facebook NYC

Serkan Piantino

Update Info | Activity Log

- Works at Facebook
- Studied Computer Science at Carnegie Mellon University
- Lives in New York, New York
- From Greenwich, Connecticut

About

Friends 1,227

Photos 480

Map 475

4,026

Subscribers

7

Work and Education

Employers

Facebook
Oct 2007 to present · Palo Alto, California

- **News Feed** with Boz and 8 others
  Nov 2007 to present
  I built News Feed again
- **Feed Ranking** with Romain Thibaux and 3 others
- **Chat** with Dan Hsiao
- **Simplicity** with Jared Morgenstern

Show all (7)

History by Year

2007 Started Working at Facebook

2006 Left Job at Bridgewater Associates

2004 Started Working at Bridgewater Associates
Graduated from Carnegie Mellon University

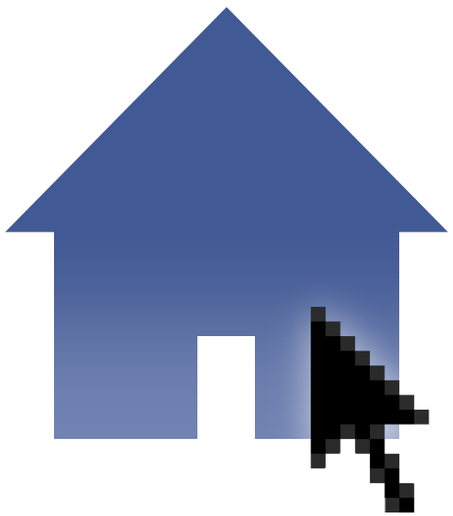
2000 Graduated from John Jay High School
Graduated from Greenwich High School

Feed Basics

What’s the job?

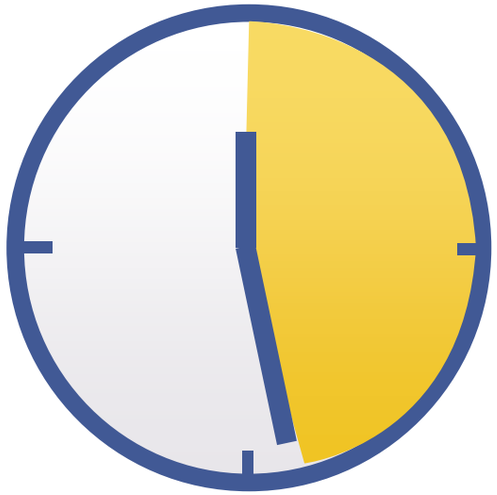
- Fetch recent activity from all your friends
- Gather it in a central place
- Group into stories
- Rank stories by relevance
- Send back results

The Scale



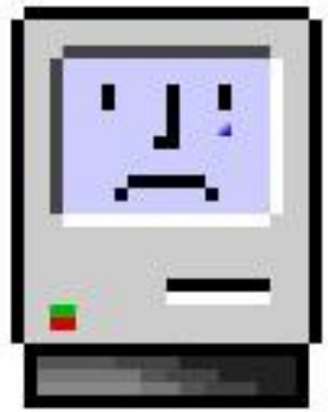
10 billion / day

Homepage views and feed queries



60 ms

Average latency



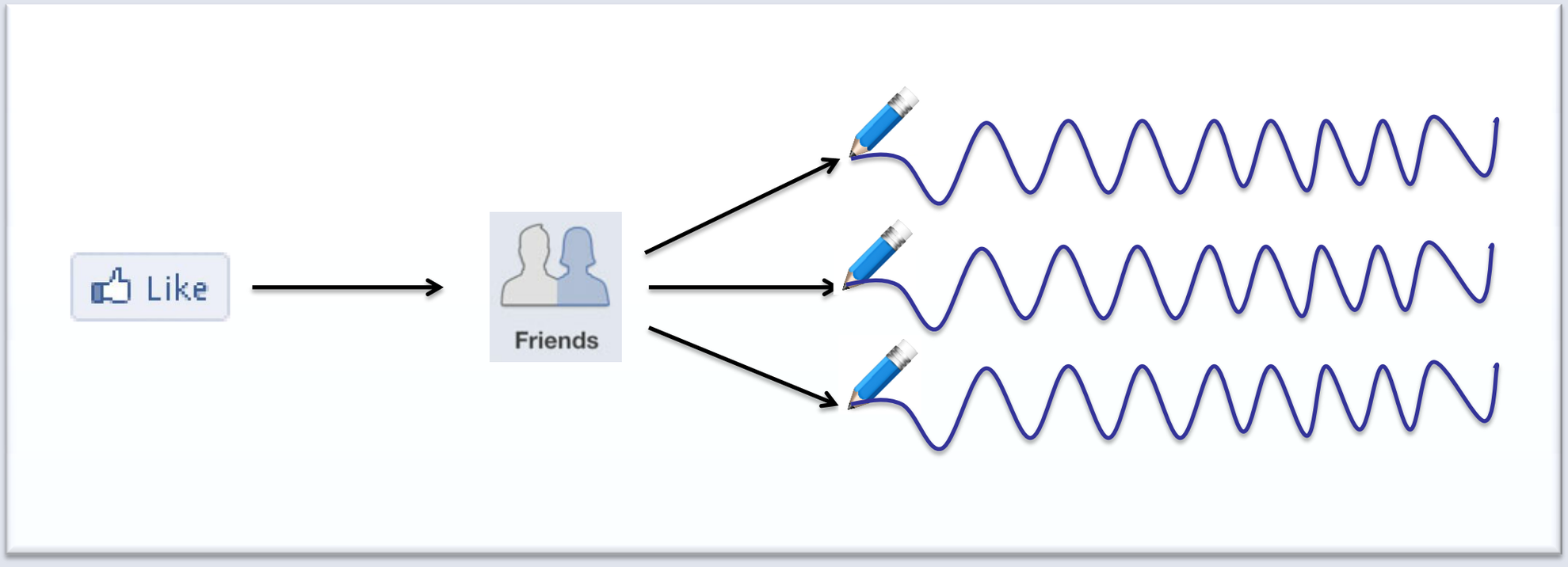
99.999%

Average query success rate

Moving content to your friends

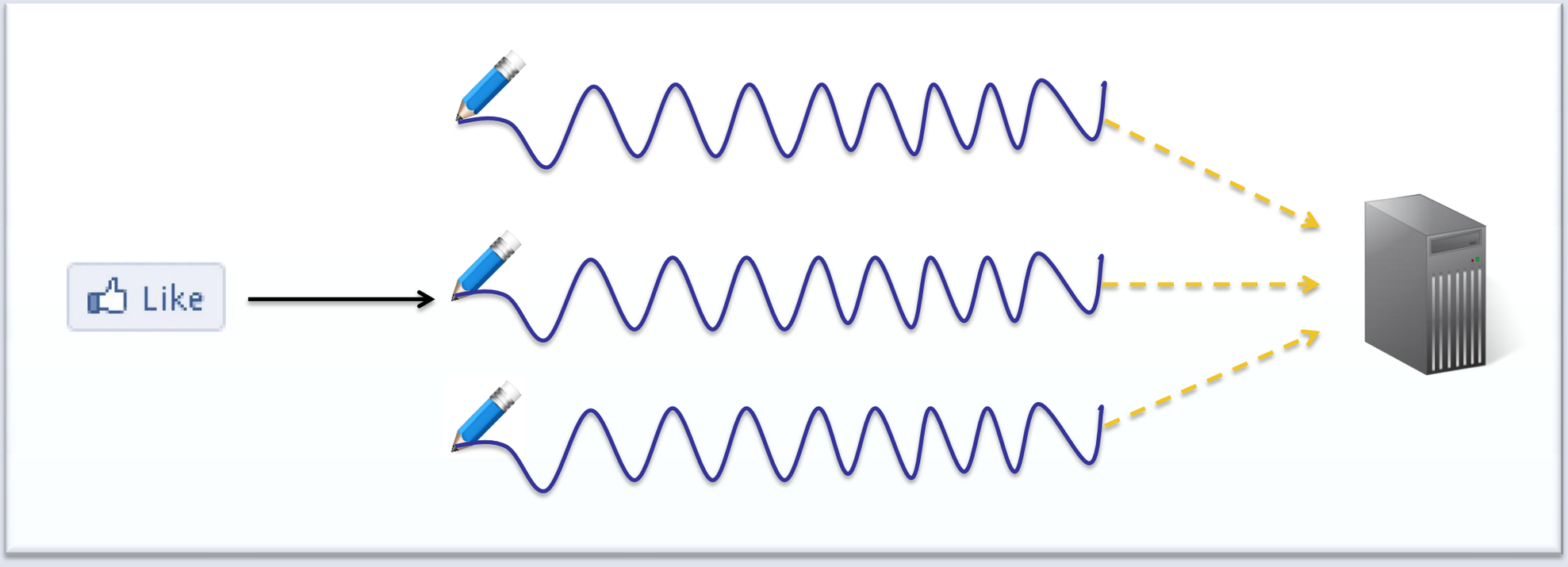
Megafeed

Broadcast writes to your friends



Multifeed

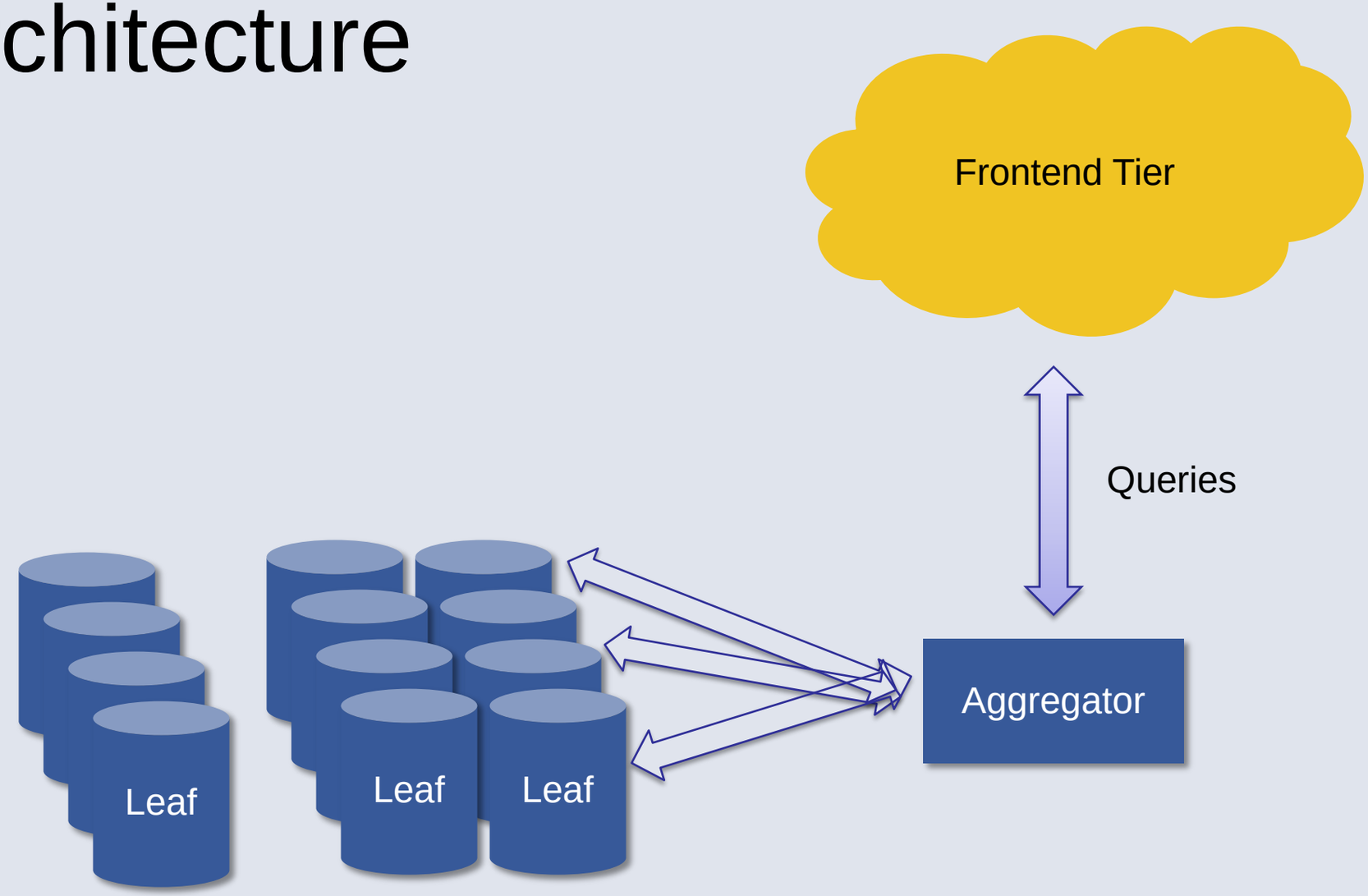
Multi-fetch and aggregate stories at read time



Chose Multifeed

- Write amplification makes the storage needs expensive in Megafeed
- Developing with read-time aggregation is flexible
- Memory and network easier to engineer around
- Fan-out reads can be bounded. Writes, often cannot

The Architecture



Challenges for another day

- Multi-region
- Pushing new code
- Ranking
- Failure/Disaster Recovery

Today: Focus on Leaf Nodes

- In-memory (mostly) databases
- Do ~40 requests per feed query
- About 50% of the total LOC

Storing Feed

Leaf node indexes

- Must store a number of users on each leaf
- Once we find a user, we want to scan his/her activity in time order
- Want to have an easy way of adding new activity without locking

- Most natural data structure is a hashtable to a linked list

First Version

- Use basic STL containers
- stl::unordered_map<int64_t, list<action*> >

- Lots of overhead
  - Storage overhead due to internal structures, alignment
  - Tons of pointer dereference, cache invalidation
- Memory fragmentation, so CPU usage trends upward
- Memory leakage leading to process restarts

A Few Tweaks

- Boost:multi_index_container
- JEMalloc (c/o) Jason Evans
- Slowed memory leakage quite a bit
- Boost library performs basically as well as stl with more syntactic niceness

Memory Pools

- Allocate a huge array once (directly via malloc)
- Round robin insert actions into it
- Fixes memory leaks outside of index structure
- Still use stl for index structures
- Can “scan” for spaces, use more complicated than round robin allocator (e.g. keep at least 2 actions per user)
- Requires fixed size actions

Moore’s Law to the rescue?

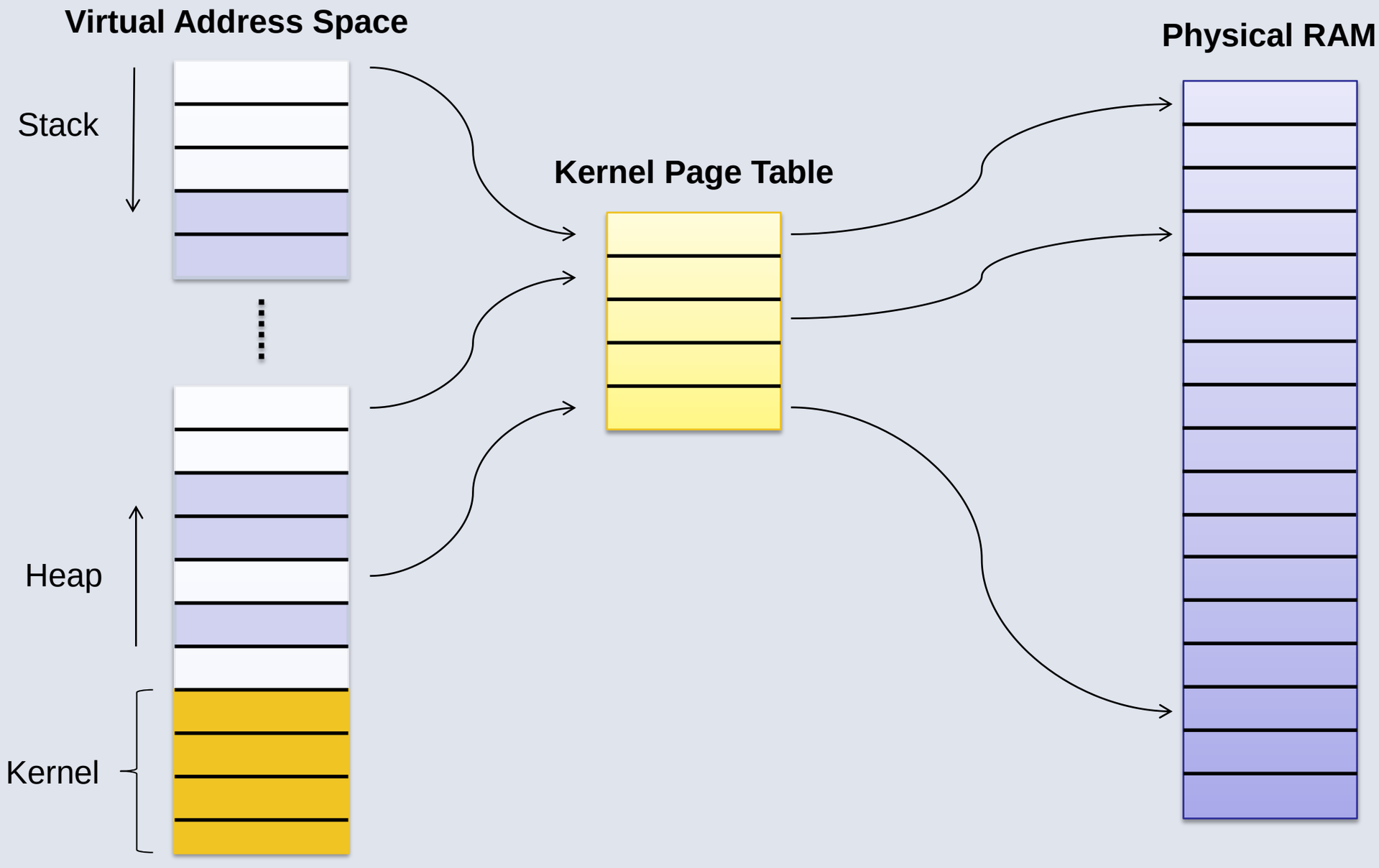
- We’re limited on total data size by how much data can be “local”
  - (i.e. within a single rack)
- Memory footprint of new servers almost doubles every year
- But… total data and query volume triples each year
  - User growth
  - Engagement/user growth
  - New features, Zuck’s Law
- Increasing focus on “needy” users. Few friends, less recent activity

Adding Persistent Storage

- Flash SSD technology has continuously matured
- Read latency and throughput about 10% of main RAM
- Sizes of 1TB or more
- Persistent!
- How do we incorporate this into our design?

Starting From Scratch

Linux Internals



Linux Internals

- Under the hood, all memory is managed through the mapping table
- Not all pages are mapped to physical RAM
  - Can be unmapped to the process (SEGV)
  - Can be unassigned to any physical pages (page fault)
  - Can be mapped to a page that resides on disk (swap file)
  - Can be mapped to another file (via mmap())

Early Thoughts

- Linux provides a mechanism for mapping data on disk to RAM
- Will use it’s own structures for caching pages, syncing writes
- What if we wrote everything on persistent flash and mmapped the whole thing?
  - Sounds ideal – let the kernel do the work of picking pages to keep in RAM, when to flush changes to disk
  - If the process crashes, restart and mmap yourself back to life

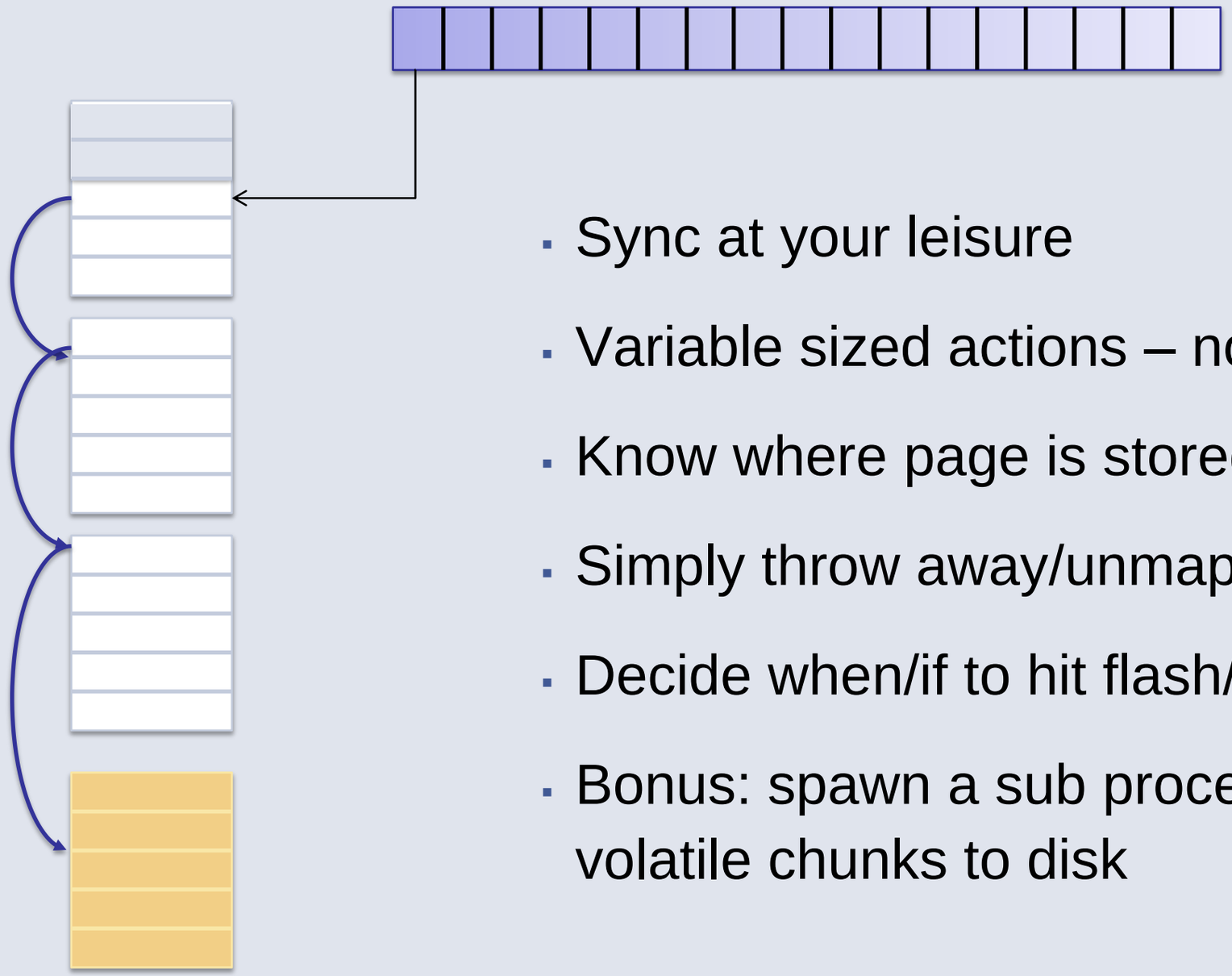
In Reality…

- Syncs written pages aggressively
- Optimized for spinning disks, not flash
  - Avoids concurrency
  - Optimistic read-ahead
  - Prefers sequential writes
- When the kernel does something, it tends to grab locks. End up with unresponsive servers during syncs
- But.. mmap, madvise etc. provide enough flexibility to manage this ourselves

Next Generation

- Mmap large chunks (~1GB) of address space
- Some are volatile and writable, others are persistent and read only
- Do your own syncing of a volatile chunk to persistent chunk
- Keep a separate index into the first action by a user (in a volatile chunk) and linked list down the rest of the way
- Write variable sized actions if you want
- When you run out of space, just unmap/delete old data, and set a global limit so you know not to follow pointers off the end

Tauren Storage



- Sync at your leisure
- Variable sized actions – no alignment
- Know where page is stored by pointer
- Simply throw away/unmap old data
- Decide when/if to hit flash/disk
- Bonus: spawn a sub process and snapshot volatile chunks to disk

Opening Up

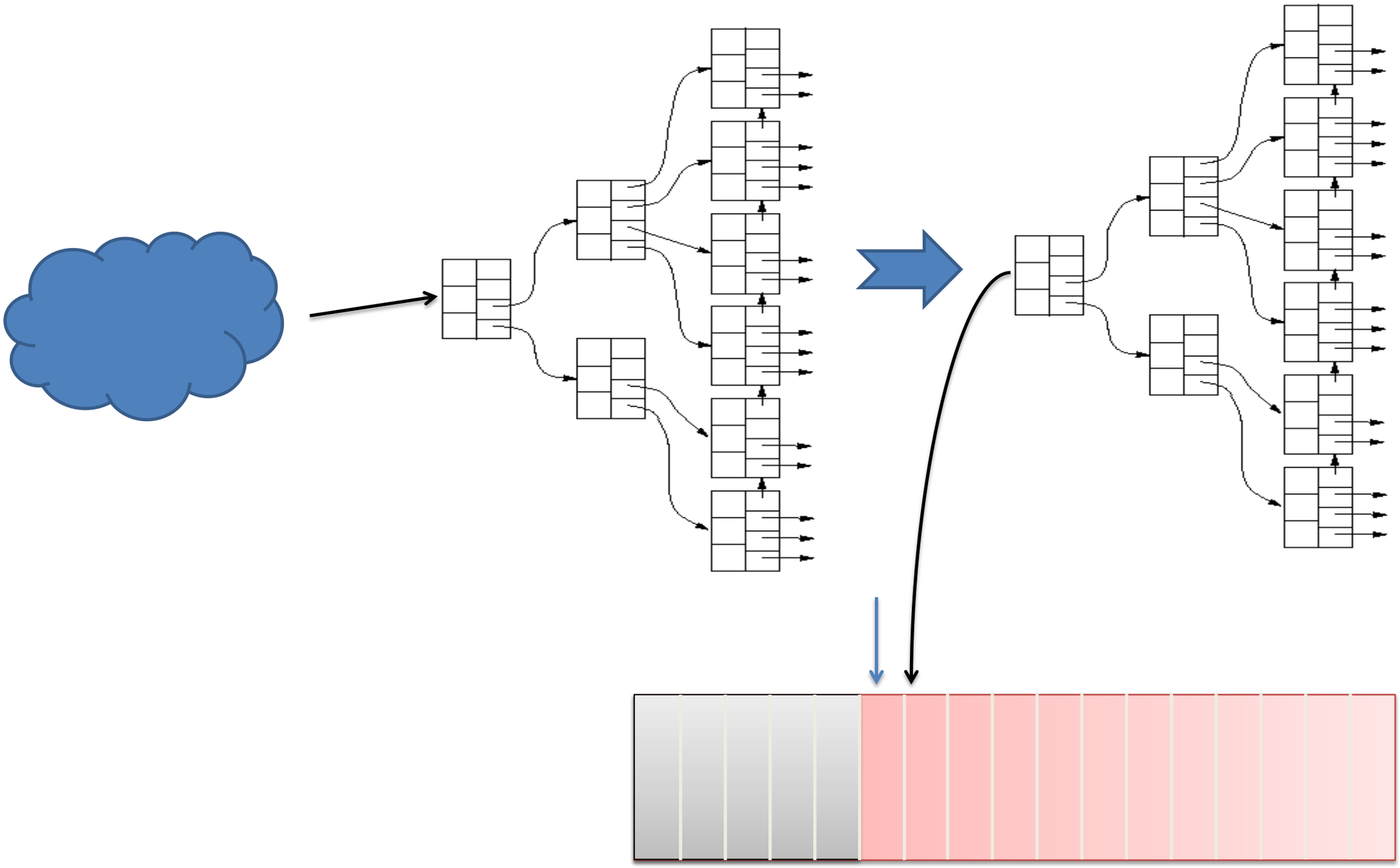
- Lots of products at Facebook look like feed, need fast graph reads
- Abstract Tauren into a c++ template
  - Stored structure T
  - Index key I
  - Order key O
- Assumes things come in roughly sorted by order key
- Get all the snapshots, performance, etc. for free
- Used on a number of projects at Facebook

Let’s do better

- We are unsatisfied
- One giant log file – seems unsophisticated
- If we move to disk we need better locality
- Not everything has inserts already roughly in order
- Let’s support simple keys/values

Centrifuge

- Store stuff in RAM in a big priority queue (b-tree actually)
- Store stuff on disk in a big sorted file
- Periodically merge the ram contents with the file
- Single key space - things can come in any order and still be sorted on disk
- This set forms a single FMap structure.
- Make your own decision about what to keep
- Hoping to Open Source soon



Key Points

Centrifuge

- We find the Multifeed approach to be more flexible, manageable
- Feed is not that much code
  - By using thrift, SMC, other FB infra we have very little glue to write
- As a result, we'd rewrite things even without immediate need
- Directly using the kernel helps a lot. Good code in there.
- We wouldn't necessarily have written from scratch today
  - Redis
  - LevelDB

facebook

Thank You!

facebook