facebook

HBase Solutions at Facebook

Nicolas Spiegelberg
Software Engineer, Facebook

QCon Hangzhou,
October 28th, 2012


facebook
December 2010

https://our.dev.facebook.com/intern/growth/countdown.php

Fburl | HBC 2012 Welcome t | Registered HBase Cl | Facebook | Titan | Tutorials | TechNews | Personal | Rentals | Conferences | HBase JIRA | Titan/HBase | MLCourse | » | Other Bo

1,000,000,000

Outline

- **HBase Overview**
- **Single Tenant: Messages**
- **Selection Criteria**
- **Multi-tenant Solutions**
  - **Physical**
  - **Self-service**
  - **Hashout**
- **Deployment Recommendations**
- **Recent Work**

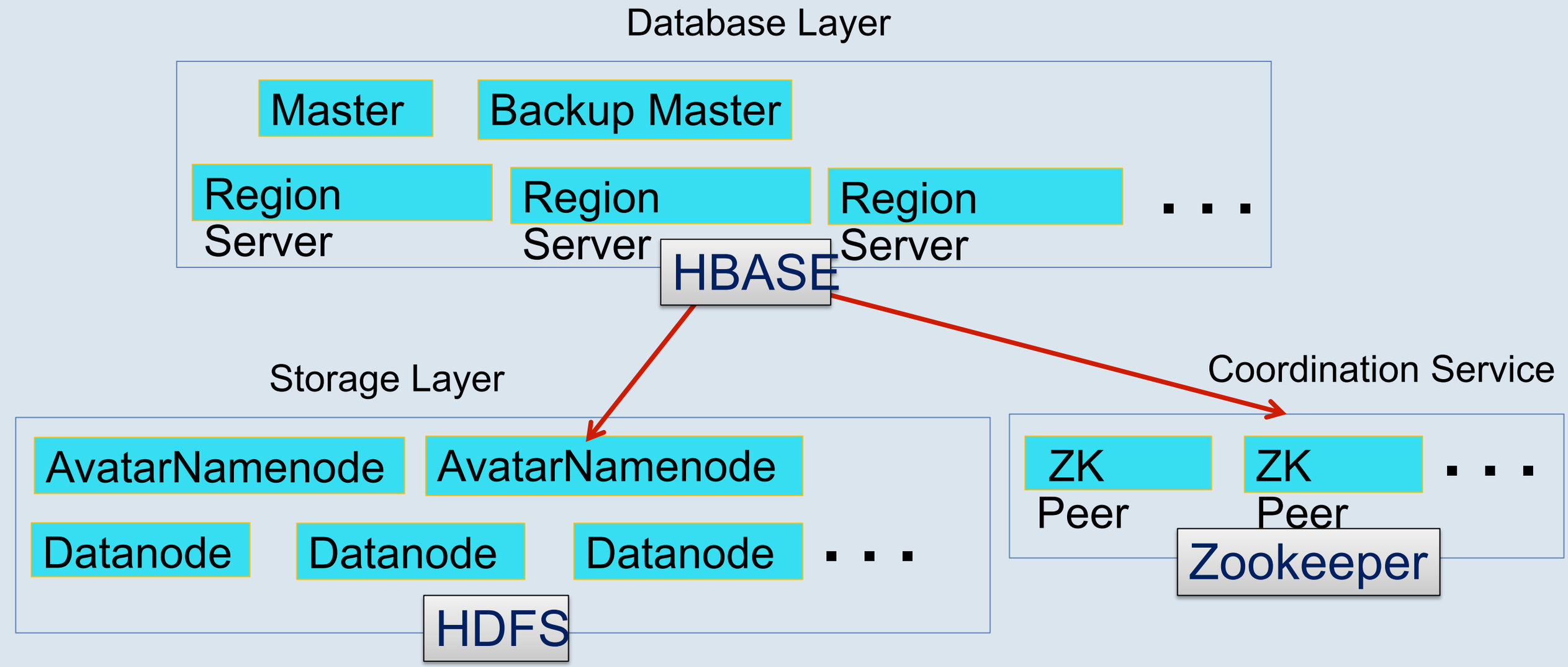
facebook

HBase Overview

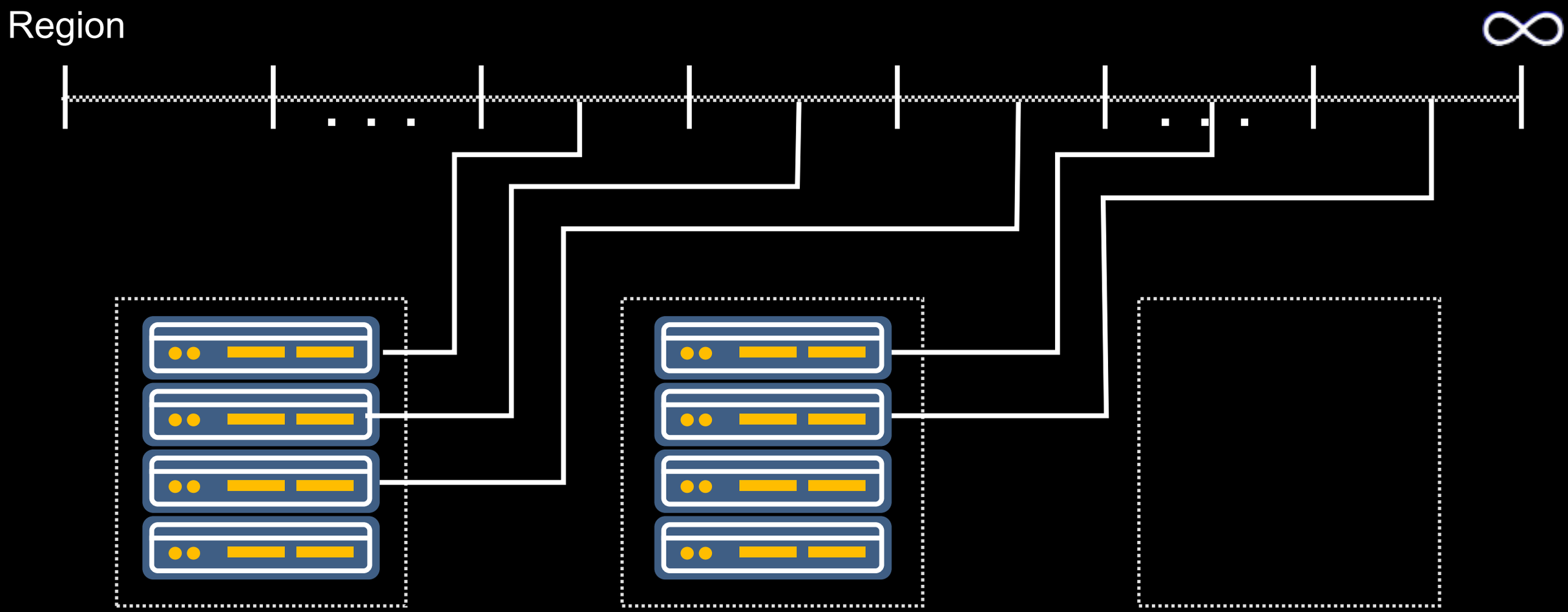
HBase in a nutshell

- Apache open source project modeled after Google's BigTable
- a distributed, large-scale data store
- built on top of Hadoop Distributed File System (HDFS)
- efficient at random writes and reads

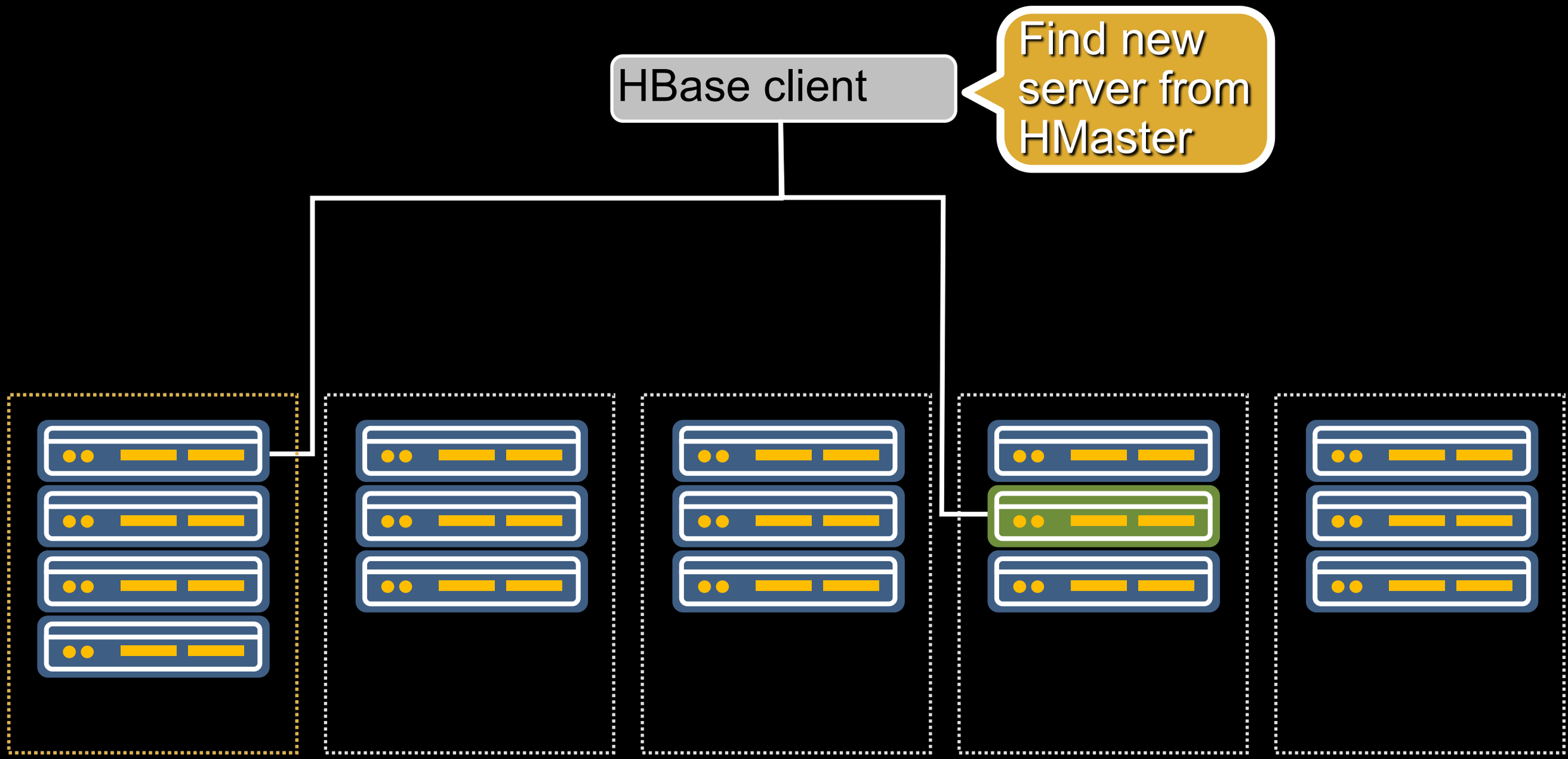
HBase System Overview



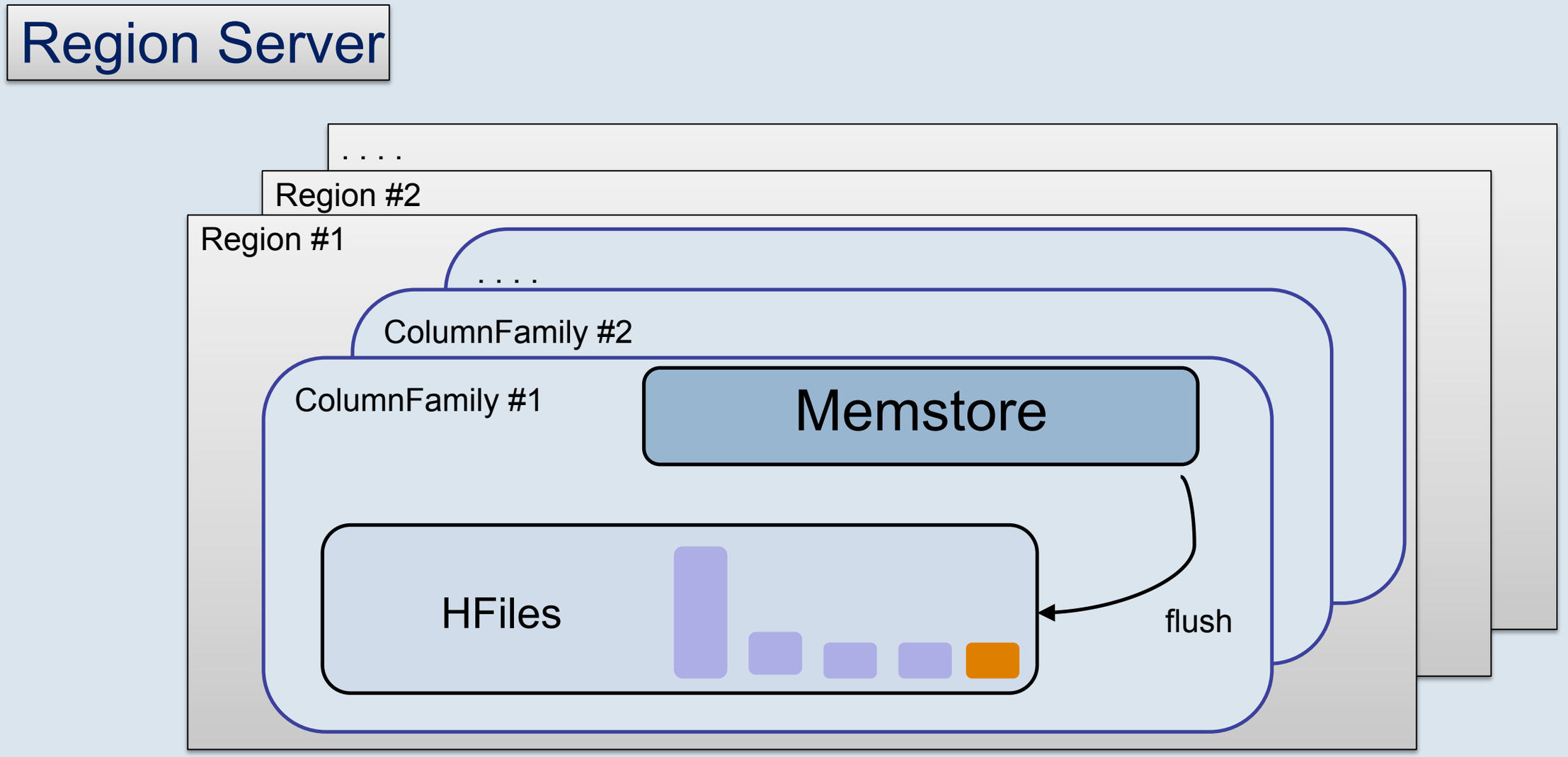
Horizontal Scalability



Automatic Failover

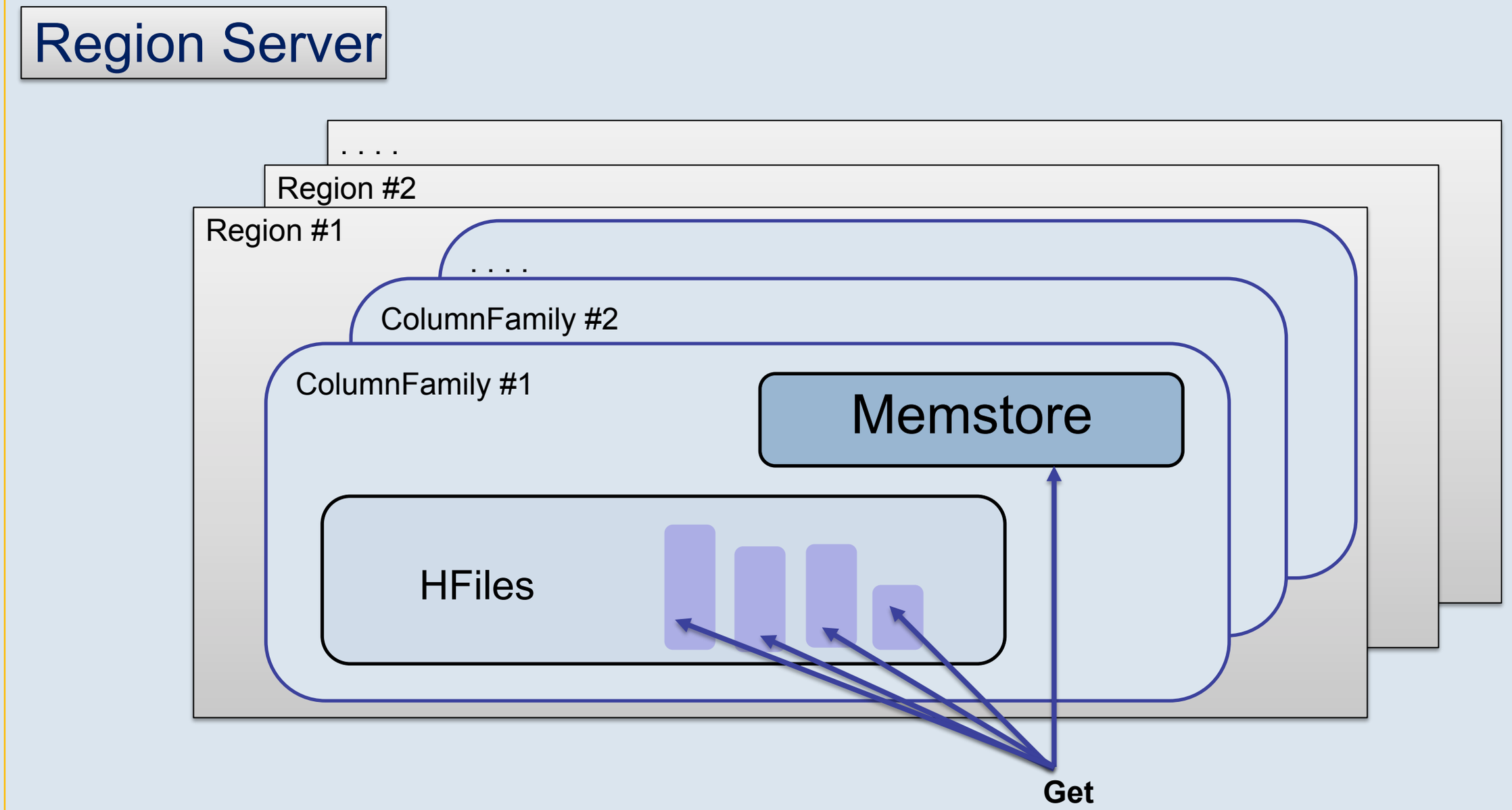


Write Path Overview



Data in HFile is sorted; has block index for efficient retrieval

Read Path Overview



HBase Use Cases @ Facebook

Messages

Facebook Insights
Self-service
Hashout

Operational Data Store
More Analytics/Hashout apps
Site Integrity

2010 2011 2012 2013

Social Graph Search Indexing
Realtime Hive Updates
Cross-system Tracing
… and more

facebook

Flagship App: Facebook Messages

Monthly data volume prior to launch

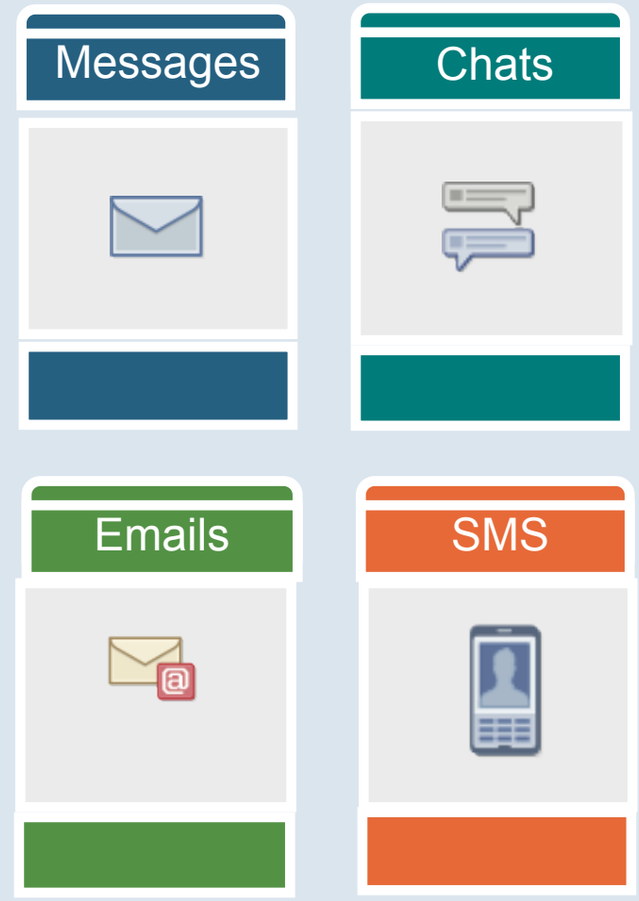
15B x 1,024 bytes = 14TB

120B x 100 bytes = 11TB

Facebook Messages NOW

Quick Stats

- 11B+ messages/day
  - 90B+ data accesses
  - Peak: 1.5M ops/sec
  - ~55%Rd, 45% Wr
- 20PB+ of total data
  - Grows 400TB/month

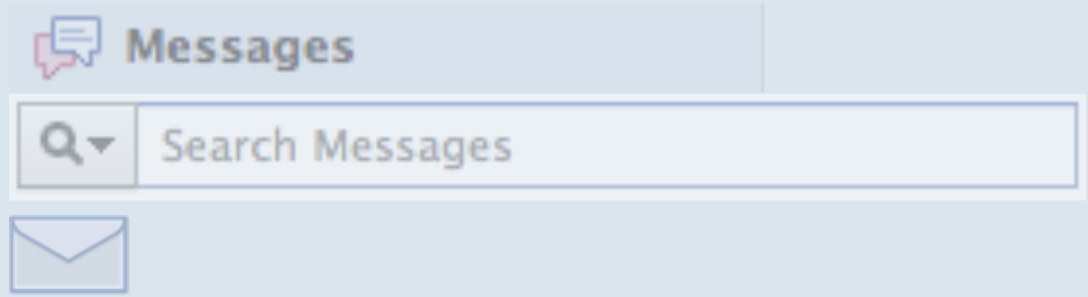


Facebook Messages: Requirements

- Very high write volume
  - Previously, chat was not persisted to disk
- Ever-growing data sets (old data rarely gets accessed)
- Elasticity & automatic failover
- Strong consistency within a single data center
- Large scans/map-reduce support for migrations & schema conversions
- Bulk import data

Messaging Data

- Small/medium sized data in HBase
  - Message metadata & indices
  - Search index
  - Small message bodies



- Attachments and large messages in Haystack (Facebook's photo store)
  - HBase currently not optimized for large objects (multiple megabytes and beyond), but could change in future.

Snapshot Schema Overview

- 3 CFs: Actions, Snapshot, Keywords

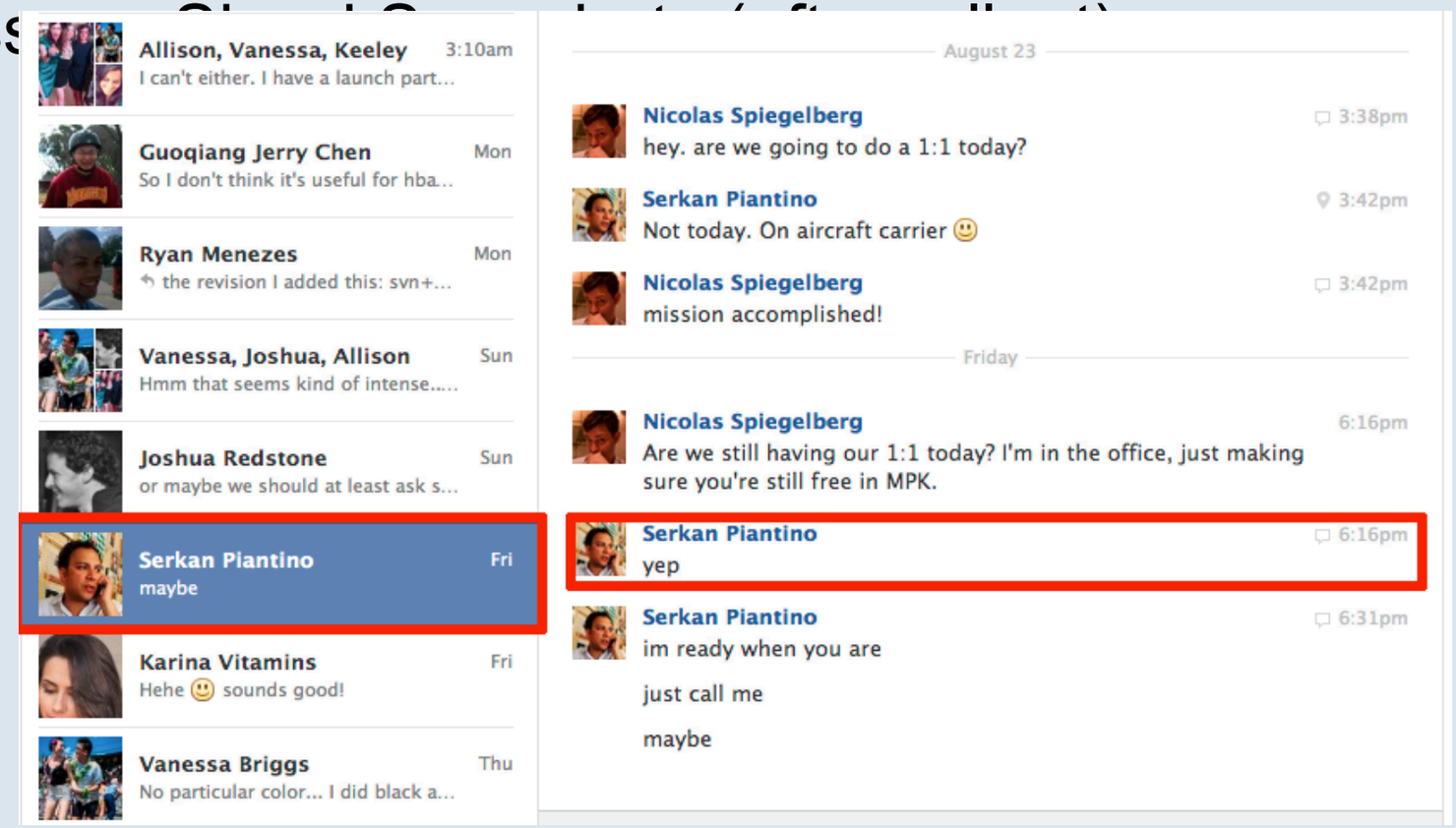
1. Actions:
   - Log of user actions.
2. Snapshot:
   - Blob containing all user metadata (everything but the message itself)
   - Mailbox = Snapshot + Actions after Snapshot
3. Keywords
   - Started out as a Lucene Index.
     - Worried about whether HBase was doing proper prefix seeking.
   - Switched to keyword-based before launch.

Messages Schema & Evolution

- Trick: “Actions” (data) Column Family is the source of truth
  - Regenerate Snapshot + Keywords by replaying Actions in order
  - Custom, Low-Volume Backup Solution
  - Fast Iteration on Schema

Messages Schema & Evolution

- Metadata portion of schema underwent 3 changes:
  - Per-User Snapshots (early development; rollout up to 1M users)
  - Per-Thread Snapshots (up to full rollout – 1B+ accounts; 800M+ active)
  - Per-Mess



How we use MapReduce

- Upgrading schema
  - Old version to new version
  - Deleting old version of schema
- Deletion jobs
  - Messages
  - Search indexes
- Other misc jobs
  - Find all users in a cell
  - For importing, exporting HBase tables

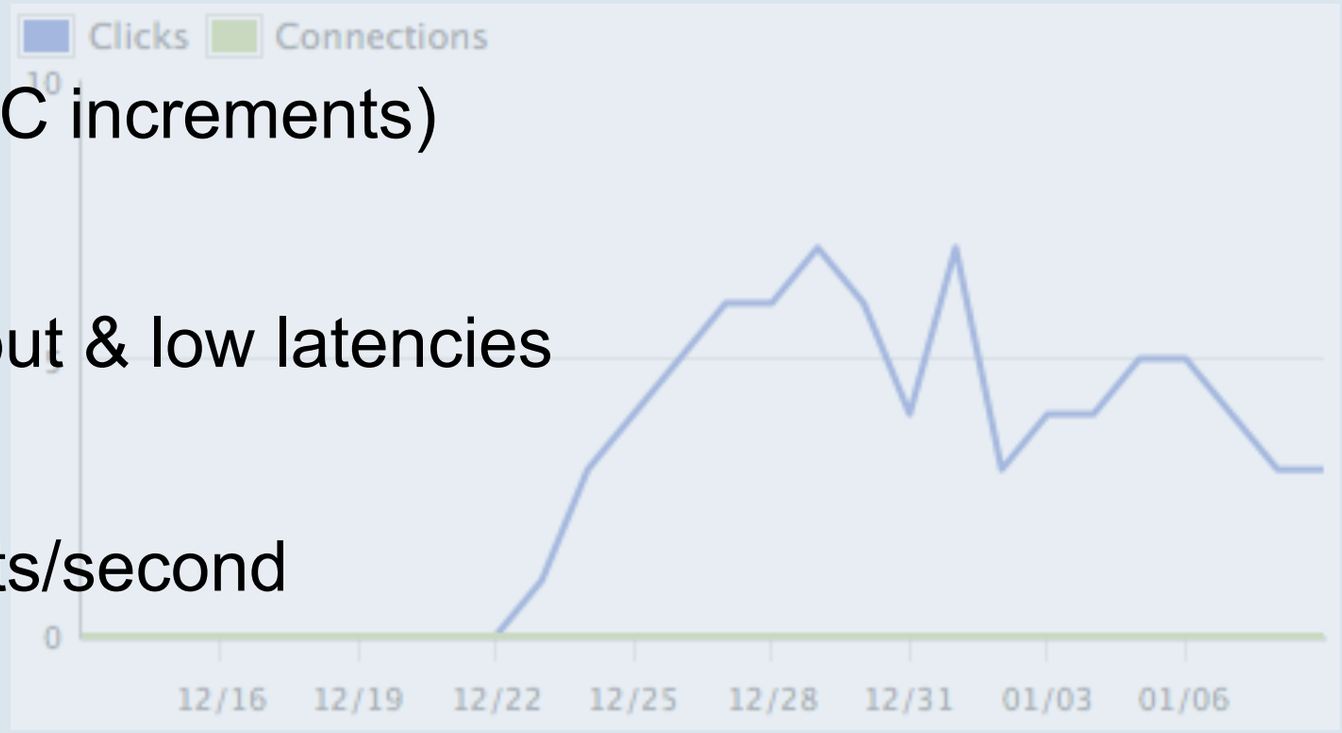
facebook

Physical Multi-tenancy

- Real-time Ads Insights
- Operational Data Store

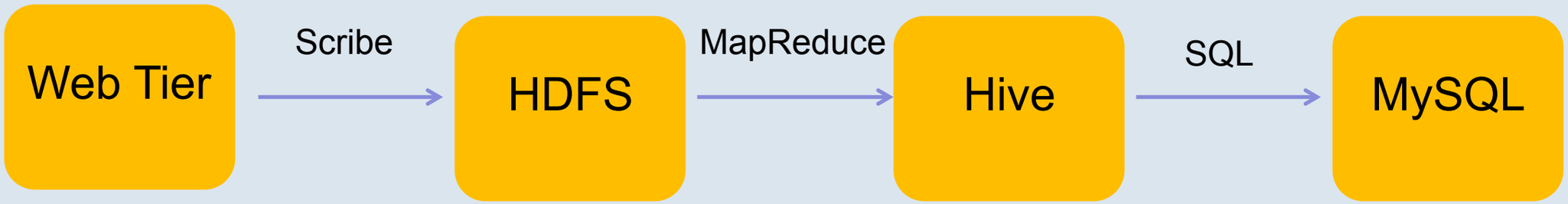
1. Real-time Facebook Insights

- Real-time analytics for social plugins on top of HBase
- Publishers get real-time distribution/engagement metrics:
  - # of impressions, likes
  - analytics by domain/URL/demographics and time periods
- Uses HBase capabilities:
  - Efficient counters (single-RPC increments)
  - TTL for purging old data
- Needs massive write throughput & low latencies
  - Billions of URLs
  - Millions of counter increments/second



Facebook Insights: Before

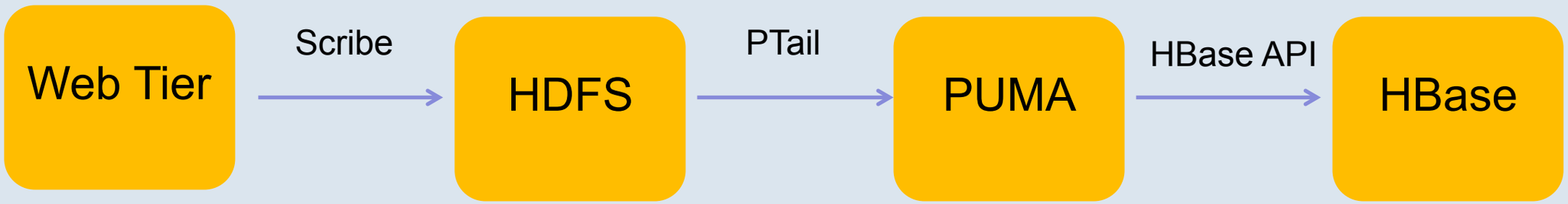
Traditional ETL using Hadoop + Hive



Update time: 15 minute – 24 hours

Facebook Insights: After (on HBase)

Realtime ETL using HBase



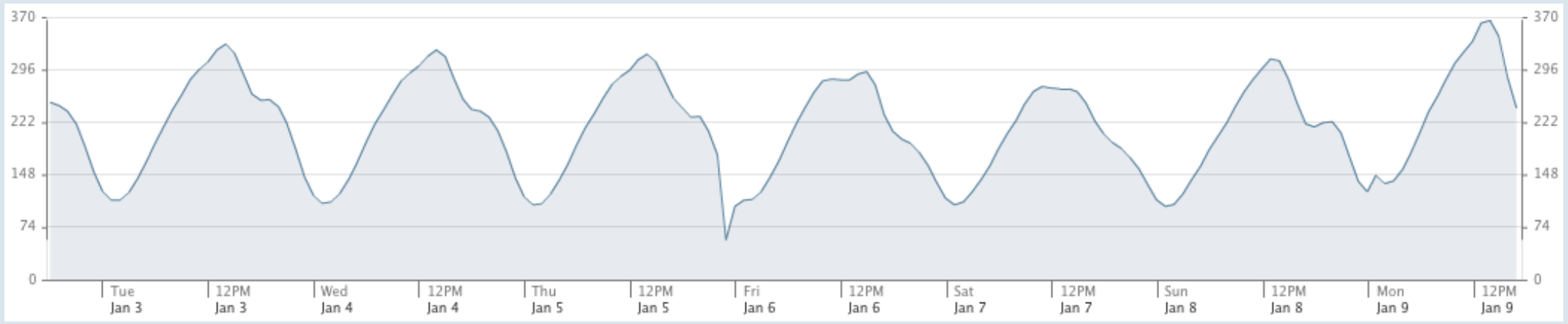
Update time: 10 – 30 seconds!

Real-time Ads Insights : Scaling

- Lessons Learned
  1. Don’t tackle more than you can handle
  2. GC Tuning can hurt
  3. Batch for efficient Writing
     - +5 is faster than 5*(+1)
     - MultiPut is faster than Put
  4. Periodic Aggregation with MR vs Realtime
     - +1000 is *MUCH* faster than 1000*(+1)

2. Operational Data Store

- Collects variety of metrics from production servers
  - System level metrics (CPU, Memory, IO, Network)
  - Application level metrics (Web, DB, Caches)
- Can easily graph historical data
  - e.g., CPU usage for machine over last 6 hours
- Supports complex aggregation, transformations, etc.
- Used by HBase itself!



ODS (contd.)

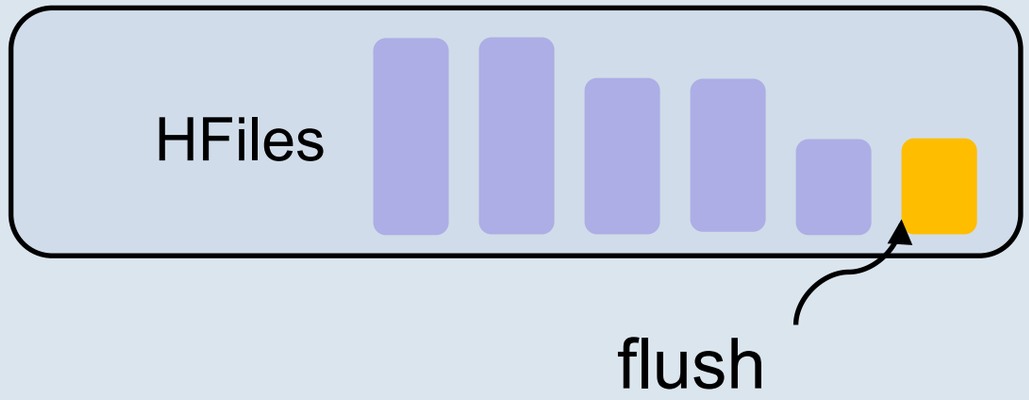
- Difficult to scale with MySQL
  - Currently 4-minute data. We need higher precision.
    - 10s of millions of unique time-series
    - ~100B+ writes per day
  - Hot-shard problem requires manual resharding
    - HBase offers automatic/dynamic splitting of shards
- Uses HBase's TTL feature to purge old data automatically!
- Reads are mostly for recent data
  - HBase's storage model perfectly optimized for reading recent data

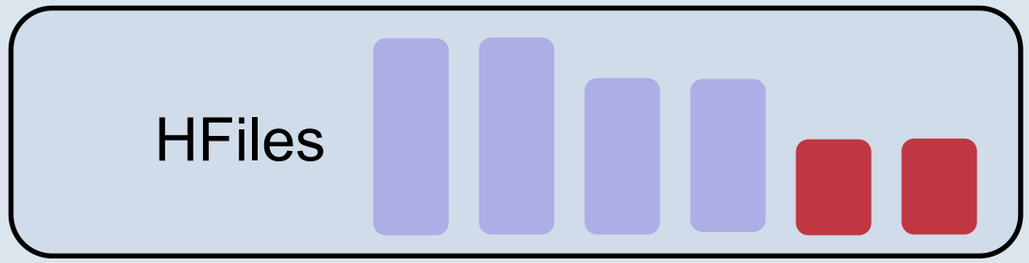
ODS: Schema Design

- Three different Column Families
  - Raw
  - Hour
  - Day
- MR Jobs to handle rollups

ODS: Compaction Tricks

- Log-structured Merge Tree
- Time-ordered Data Storage!
  - Better Spatial Locality
  - Compaction.ratio 1.4 ➔ 0.25
  - Future: Use Coprocessor
    min…
  - Server-side constraints

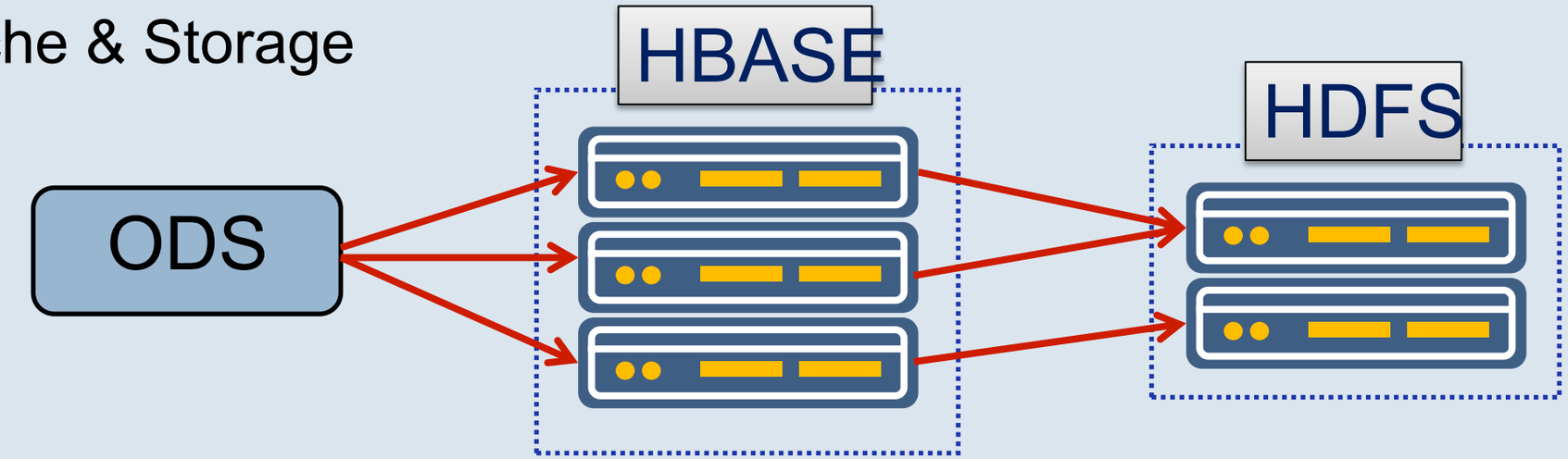




day… hour…

ODS: Lesson Learned

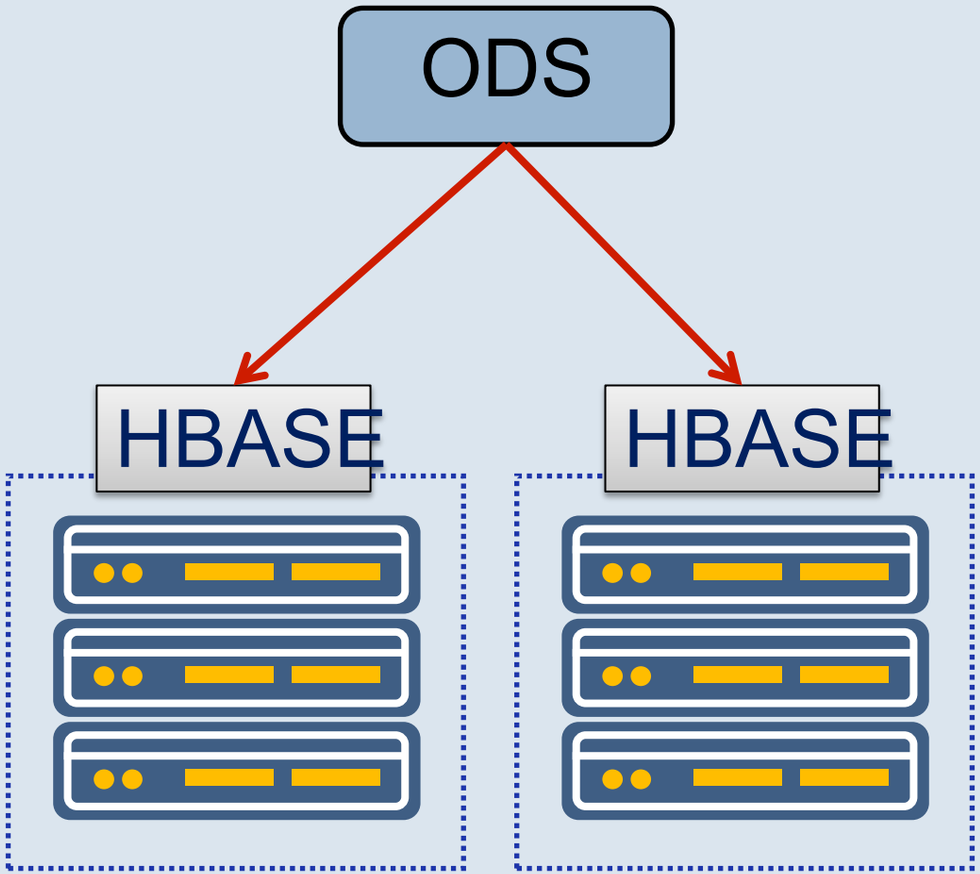
- Split-tier Architecture
  - Idea: Scale Cache & Storage Independently



- Problems:
  1. Network Architecture Insufficient
  2. Memcache + App Logic More Effective
  3. Traffic Between Nodes > User Traffic
     - Flush
     - Compaction (R+W)

ODS: HA

- Problem:
  - Needed High Availability
- Solution:
  - Dual-cluster Architecture
    - Same Location, Different Network
  - Best Effort for Puts
  - MR Jobs for Eventual Consistency
  - Biggest Negative: 6x replication
  - *End-game: Master-Master Replication*



facebook

Self-Service Tier

Self-Service Tier: What is it?

- Optimistic Multi-tenancy
  - **Allow users to create a table**
  - **Inform them about state of multi-tenancy**
  - **Expect users to “do the right thing”**
- Monitoring
  - **JMX Metrics**
  - **RPC Monitor**
  - **Slow-query Logs**
  - **HLog/HFile Pretty Printer**

Self-Service Tier: Problems!

- Except…
  - **Users have a relative opinion about “not a lot of data”**
  - **Users don’t analyze their own data (250MB KVs)**
  - **Users don’t always name their tables well**
  - **Resource Analysis is non-trivial**

Self-service: Lessons Learned

- **Physically Isolate Users**
  - **Thread:Server Mapping**
- **Block Abusive Users**
- **Promote Heavy Users**

facebook

Hashout

Controlled Multi-tenancy

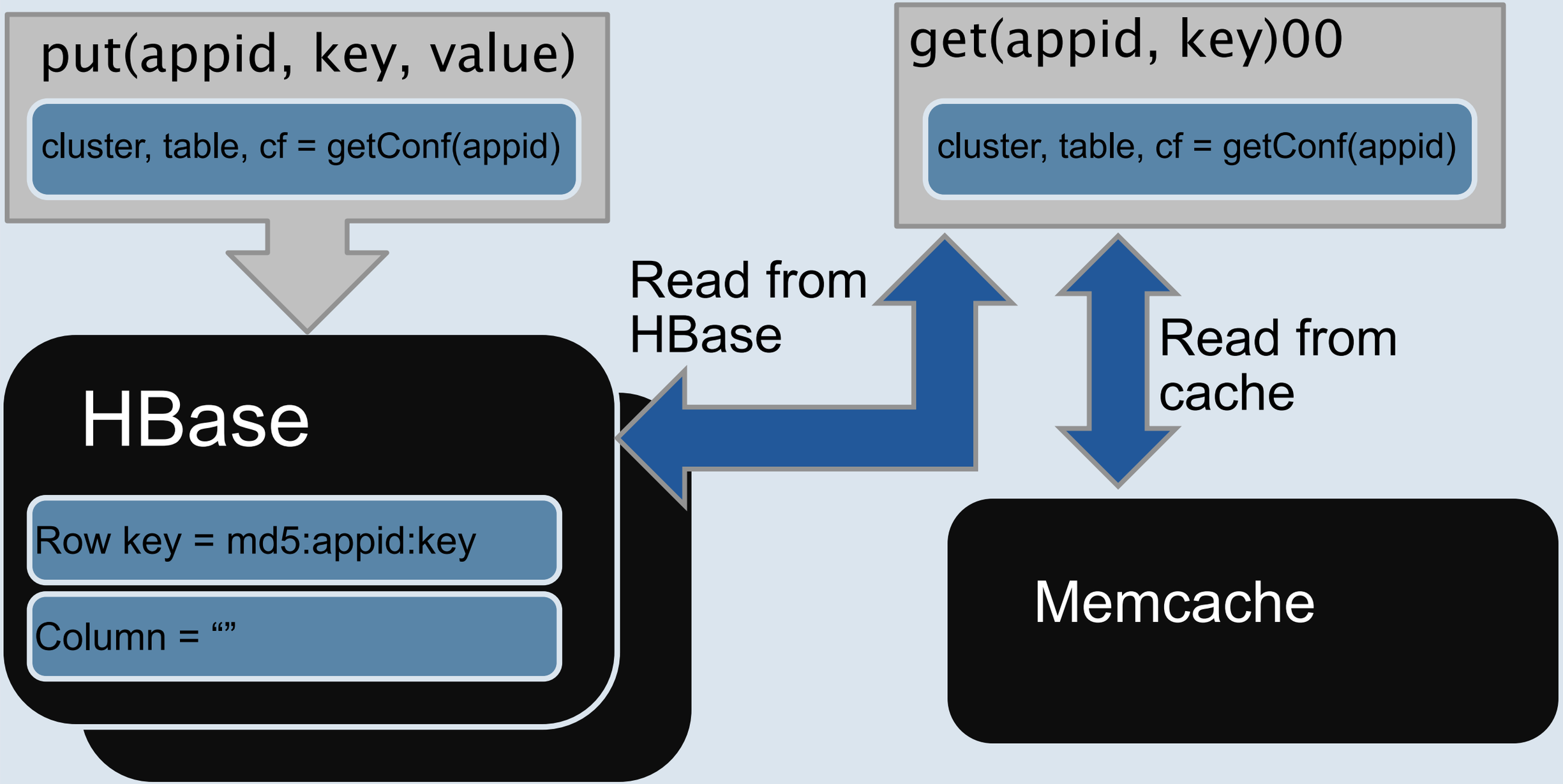
What is it?

- A generic Key-Value store
  - **Multiple apps**
  - **Simple API**

```
put(appid, key, value)
value = get(appid, key)
```

- **Need to declare app in code**

Architecture



How we started

- Not a self service model
  - **Each app is reviewed**
  - **Global and per-app metrics**
    - **Num gets, puts, latencies, errors**
    - **Added Client-side metrics**
- In case things went wrong
  - **Per-app kill switch**
  - **Started with non-critical apps**

What we observed

- Miscellaneous improvements
  - **Memcache for read intensive apps**
  - **Friendly names for apps**
  - **Alerts on exceptions**
- Capacity estimation is hard
  - **Load on HBase (gets/puts/deletes)**
  - **Number of connections to Thrift**
  - **Too many requests from one app ==**

    Silently Point the User to a Separate Tier

facebook

Recent Work

HBase Development @ Facebook

Reliability/Correctness

- Durable commit log in HDFS
- Multi-CF ACID semantics
- Many txn log recovery bug fixes
- Pluggable HDFS block placement policy to reduce probability of data loss
- Thrift gateway fixes

Features

- Index-Only queries
- Hot Backups
- C++ Client
- Intra-row pagination support
- HTableMultiplexer

Availability

- Durable commit log in HDFS
- Rolling upgrades
- Online alter table
- Interruptible compactions
- Faster region opens

Manageability

- HBase Health Checker (hbck)
- Slow query logs
- TaskMonitor (like v$session stats in Oracle)
- Lots of Metrics (per-CF metrics, master metrics)

Performance/Scaling

- Bloom Filters
- Compaction algo improvements
- Multi-threaded compactions
- HFile V2
- Timerange hints/optimization
- Lazy Seeks
- Delete Bloom Filter
- Improved handling of compressed HFiles
- DataBlockEncoding
- Locality on full/rolling cluster restarts
- Per-region data placement
- Compressed RPCs

HBase Future Work

It is still early days…!

- Eliminate HDFS SPOF (Bookkeeper)
- Features (secondary indices, query language)
- Incremental Processing/Indexing Framework
- HBase on Flash
- Lot more performance/availability improvements

Acknowledgements

- Data Infrastructure Team
- Open source community
- And lots of people across Facebook

We are 1% finished

Thanks! Questions?

facebook.com/engineering