facebook

Building Mission Critical Messaging System On Top Of HBase

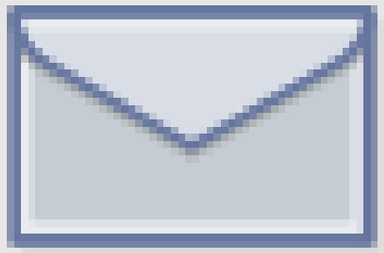
Guoqiang Jerry Chen, *Liyin Tang, Facebook*

Hadoop China 2011, Beijing

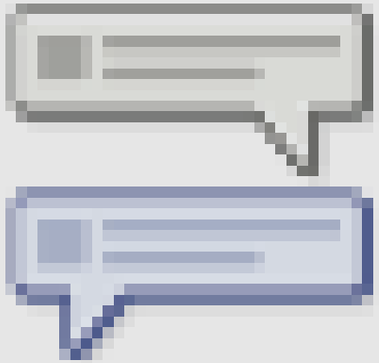

facebook
Search
Home
Profile
Account
Andrew Bosworth
Edit My Profile
News Feed
Messages 1
Other
Events
Friends
Groups Team
Groups Engineering 1
Engineering Spea...
More
Photos
Quikvote
More
Friends on Chat
Messages
+ New Message
Search Messages
James Wang, Dave Troiano
remember that time i made prime rib roast?
33 minutes ago
Will Bailey
k I'll follow up with him
2 hours ago
Ben Chiaramonte
like the song
2 hours ago
Kenny Lau
anything should push you
2 hours ago
Ross Bayer
whatsup?
3 hours ago
Mark Zuckerberg
= and I'll dig into the engineering side more
6 hours ago
Eric Antonow
pretty bad - you win this round
17 hours ago
Pat Kinsel
... and i thought you called yourself a Visio fan for life.
22 hours ago
Katie Zacarian
"people respect him"
Yesterday

Facebook Messages

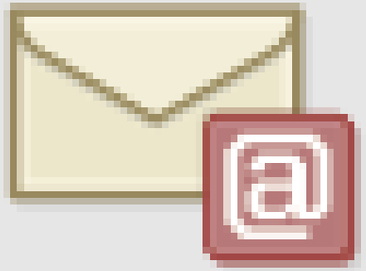
Messages



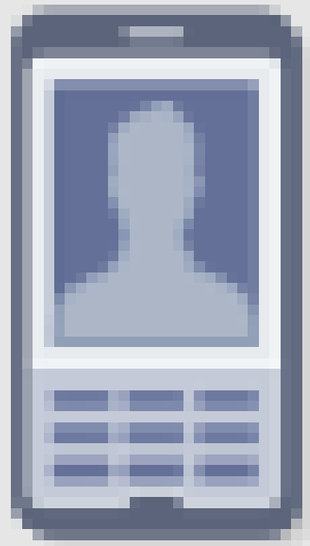
Chats



Emails



SMS



Pasco County deputy uses Facebook to help stop SWAT situation

Posted: 11/17/2011

By: Kimberly Kuizon

ODESSA, Fla. - When Corporal Arthur Morrison responded to a SWAT situation in Odessa he never expected Facebook to become his deal breaker. "I sent him the friend request and he immediately, not even 30 seconds went by and he accepted that first friend request," sai

Deputies said a ma

Morrison said he made an instant connection. The 13 year hostage negoatior said he was setting the stage to end the standoff. The messages tell the story.

> **Morrison: Lying doesn't help. I am not here to lie.**
>
> *Man: I can't be helped this time it will only lead to next time. I'm damaged goods.*
>
> **Morrison: I don't believe that I have many friends that have received help and made it through very tough times. Put me to the test. Tell me how I can help you.**

"We wanted to give him a reason to come out. To give him something that he can grab ahold of, because I don't believe he wanted to hurt himself or kill himself. He was just going through a rough patch right now," said Morrison.

Facebook Messages: brief history

- Project started in Dec 2009
- Oct 2010, selected users are in production
- Jan 2011, meaningful number of users are in production.
- Mid 2011, fully rolled out to all users (800M+), legacy messages for 1B+ accounts are migrated over to HBase as well.

Facebook Messages: Quick Facts

- Unstable vs. stable storage:
  - This is the permanent storage system for all private messages of users of Facebook, and it is the single source of truth → Stable Storage system, backup!
- Online vs. offline storage:
  - It is online! Any problems with HBase are visible to the users → High availability, low latency.

Facebook Messages: Quick Stats

- 8B+ messages/day

- Traffic to HBase
  - 75+ Billion R+W ops/day
  - At peak: 1.5M ops/sec
  - ~ 55% Read vs. 45% Write ops
  - Avg write op inserts ~16 records across multiple column families.

Facebook Messages: Quick Stats (contd.)

- 2PB+ of online data in HBase (6PB+ with replication; excludes backups)
  - message data, metadata, search index
  - All data LZO compressed
  - Growing at 250TB/month

Why we chose HBase

- High write throughput
- Good random read performance
- Horizontal scalability
- Automatic Failover
- Strong consistency
- *Benefits of HDFS*
  - *Fault tolerant, scalable, checksums, MapReduce*
  - *internal dev & ops expertise*

What do we store in HBase

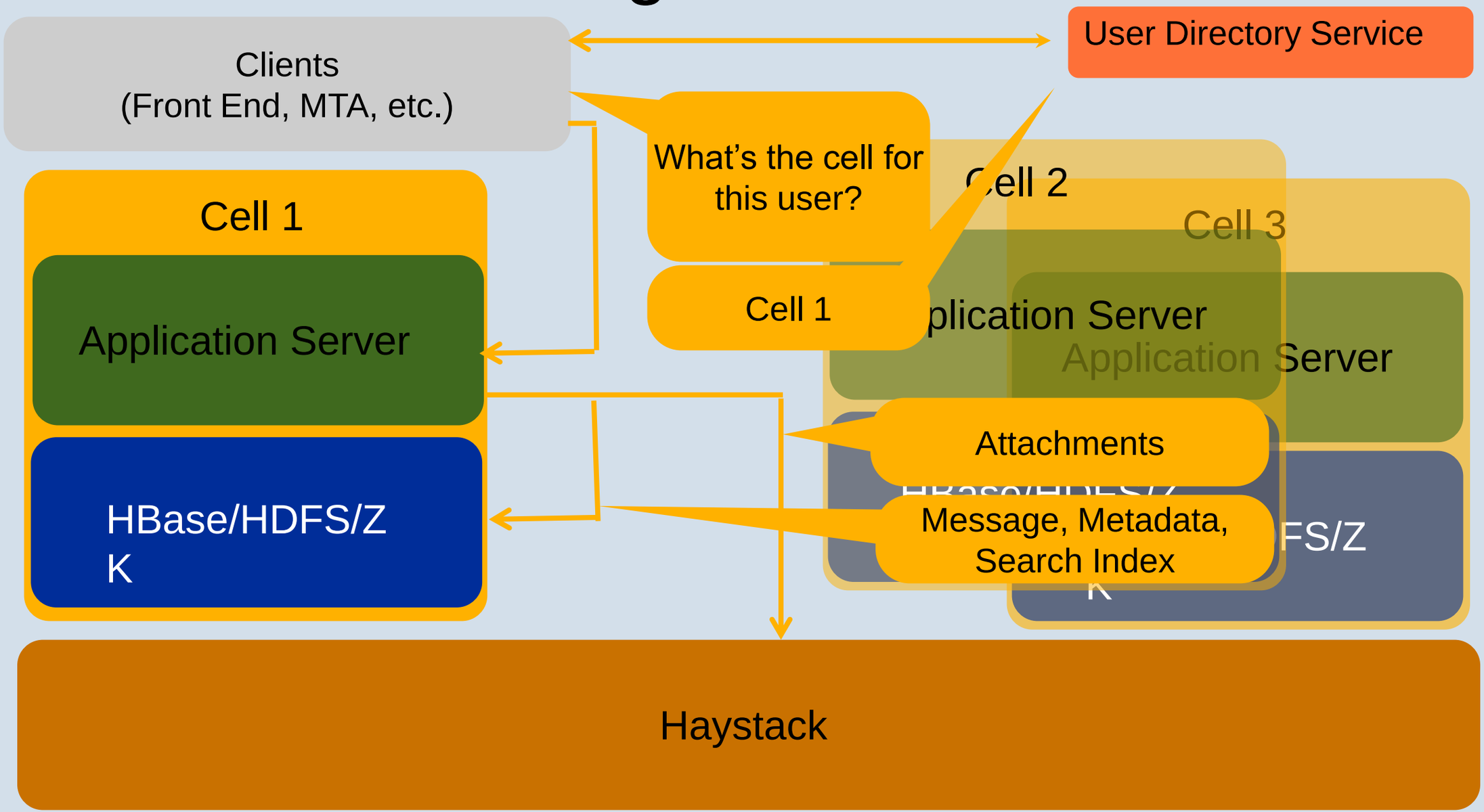
- HBase
  - Small messages
  - Message metadata (thread/message indices)
  - Search index

- Haystack (our photo store)
  - Attachments
  - Large messages

How do we make it work?

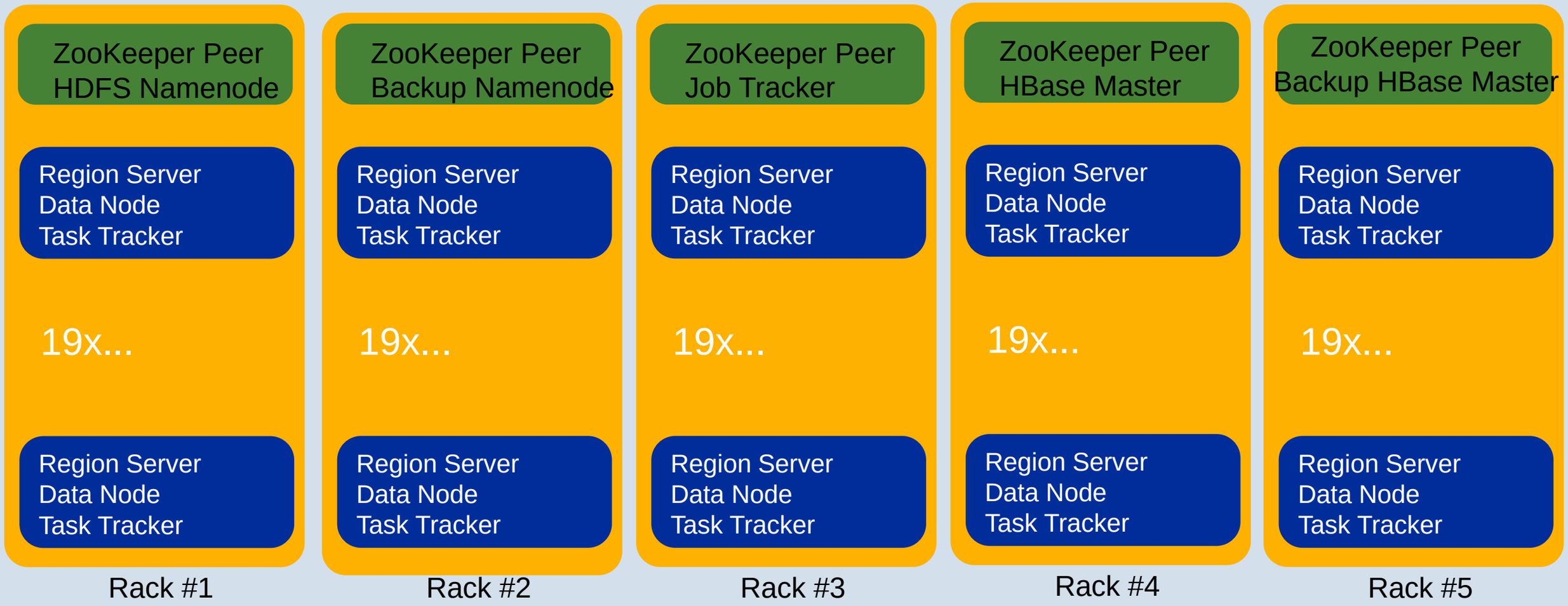
- Cell based architecture
- Schema Design/Iteration
- Stabilize HBase
- Monitoring and operations

Facebook Messages Architecture



Cluster Layout For a HBase Cell

- Multiple clusters/cells for messaging
  - 20 servers/rack; 5 or more racks per cluster
- Controllers (master/Zookeeper) spread across racks



Why multiple cells?

- If HBase goes down, it affects only a portion of the user base
  - Upgrades that need a full cluster restart
  - Bugs resulting in cluster outages

- Easy to move to new datacenters
- Enable us to do shadow testing – a cell with same load as production cells.

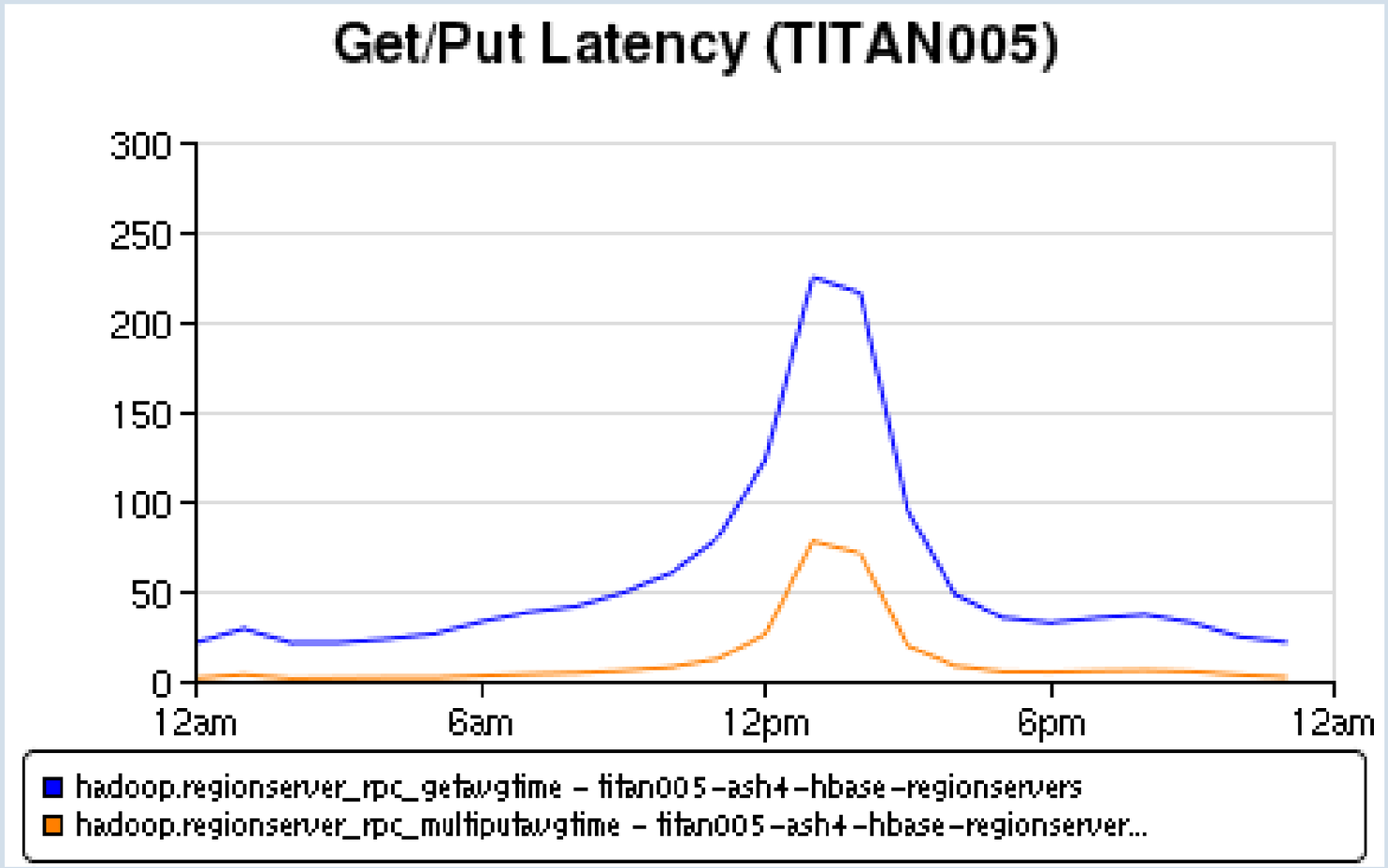
Messages Schema - Row Keys

- Each user is one row:
  - The entire messaging history of a user is in one HBase row. For some users, this can be as big as a few G.
  - Use the row atomicity of HBase well!
  - The message search index is also in the same row. This enables us to atomically update search index.
- The row key is: md5(userid) + userid
  - Randomly distribute users and avoid hot region problem.
  - This also enables us to presplit the table.

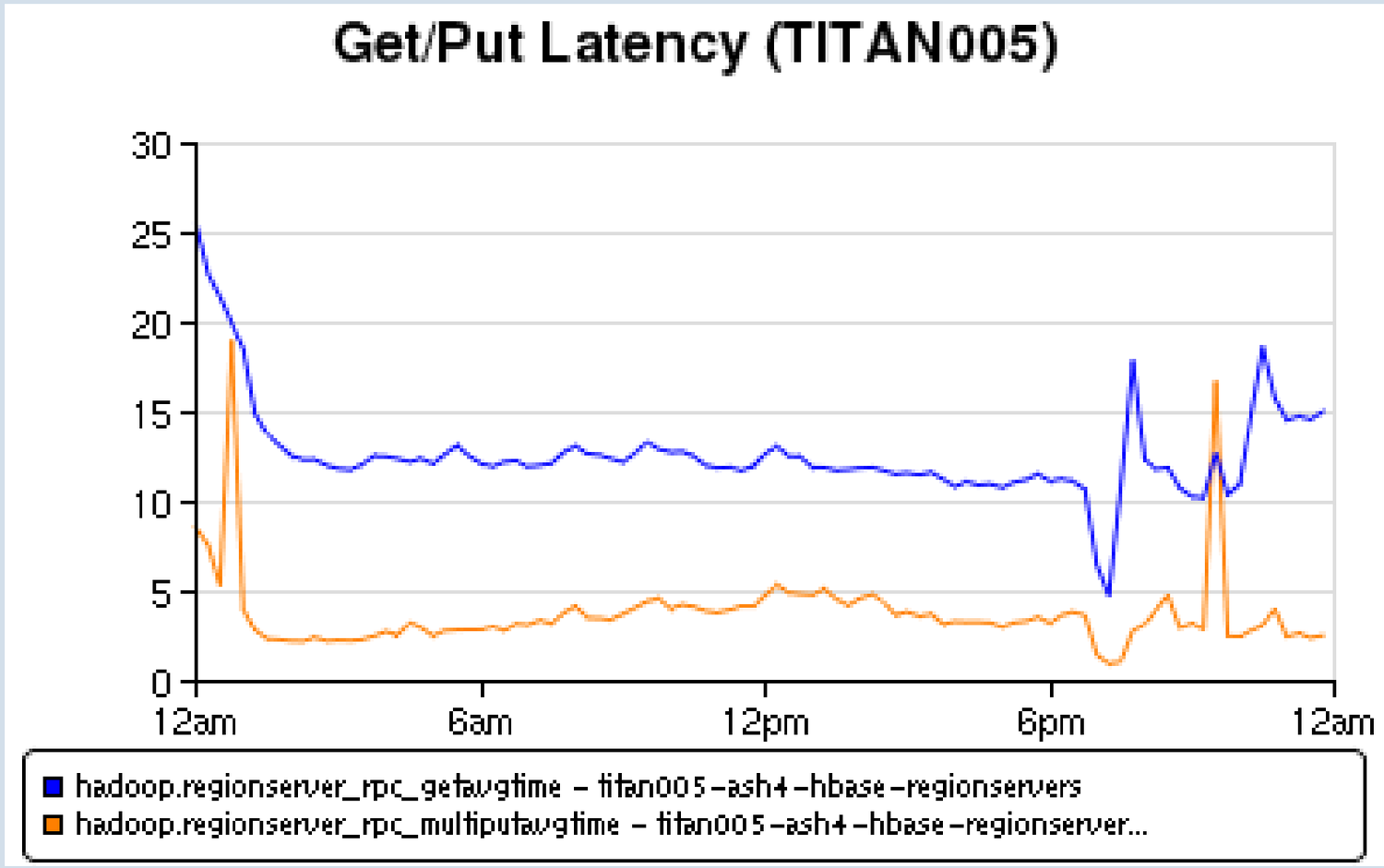
Messages Schema & Evolution

- “Actions” (data) Column Family the source of truth
  - Log of all user actions (addMessage, markAsRead, etc.)
- Metadata (thread index, message index, search index) etc. in other column families
- Metadata portion of schema underwent 3 changes:
  - Coarse grained snapshots (early development; rollout up to 1M users)
  - Hybrid (up to full rollout – 1B+ accounts; 800M+ active)
  - Fine-grained metadata (after rollout)
- MapReduce jobs against production clusters!
  - Ran in throttled way
  - Heavy use of HBase bulk import features

Get/Put Latency (Before)



Get/Put Latency (After)



Stabilizing HBase

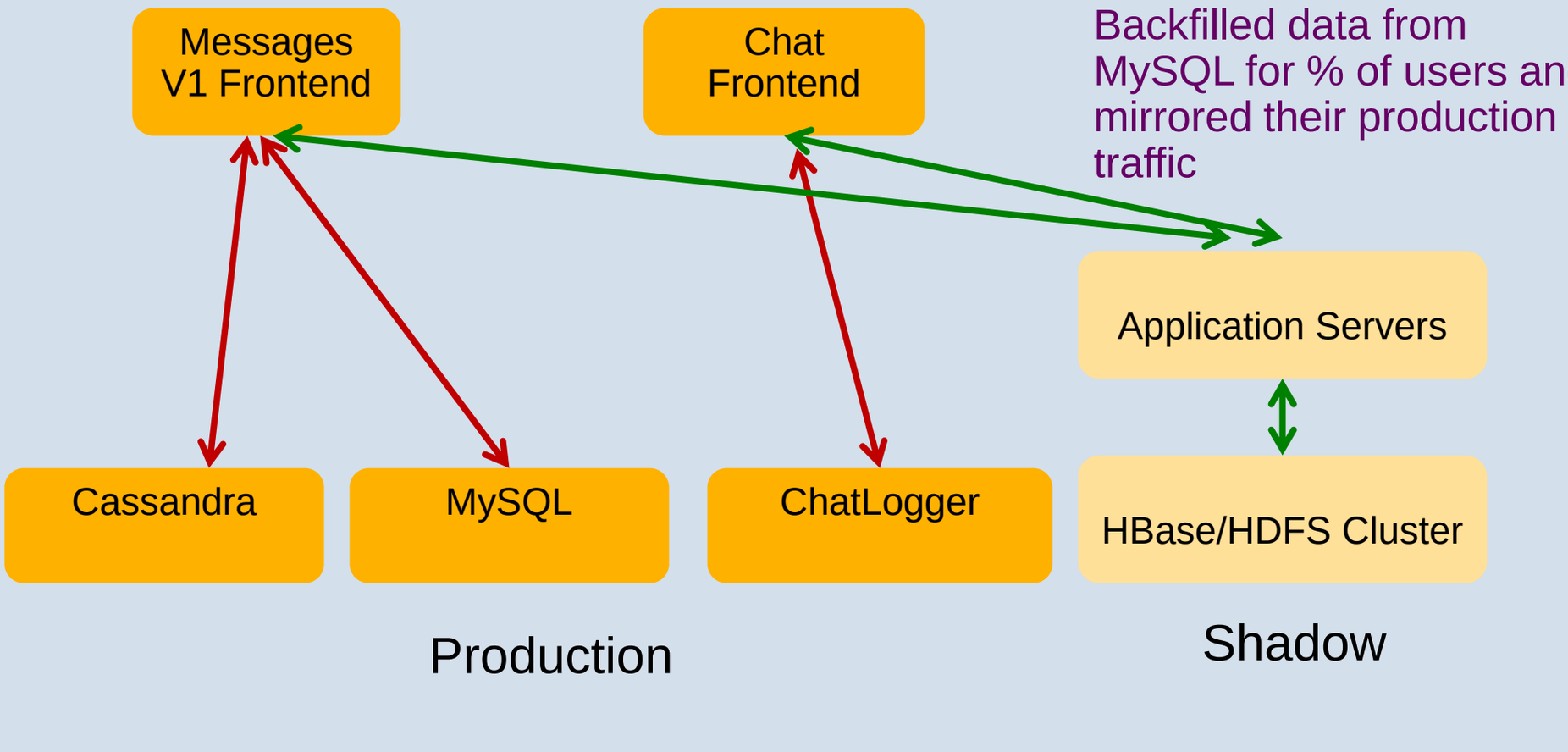
- Right before launch (Oct 2010), we created a production branch from Apache trunk, close to 0.89.20100924. Now the code base is on Apache under 0.89-fb.
- The refactored master code was coming in around the same time and we decided not to take it due to imminent product release.
- Continuously iterate on this branch and contribute back patches to Apache trunk.
- Sometime, knowing what to avoid can also help
  - *disable automatic split, instead presplit your table and set a huge split limit.*

Stabilizing HBase

HBase has never been put into test at this scale.
The only way is to roll it out and iterate rapidly.

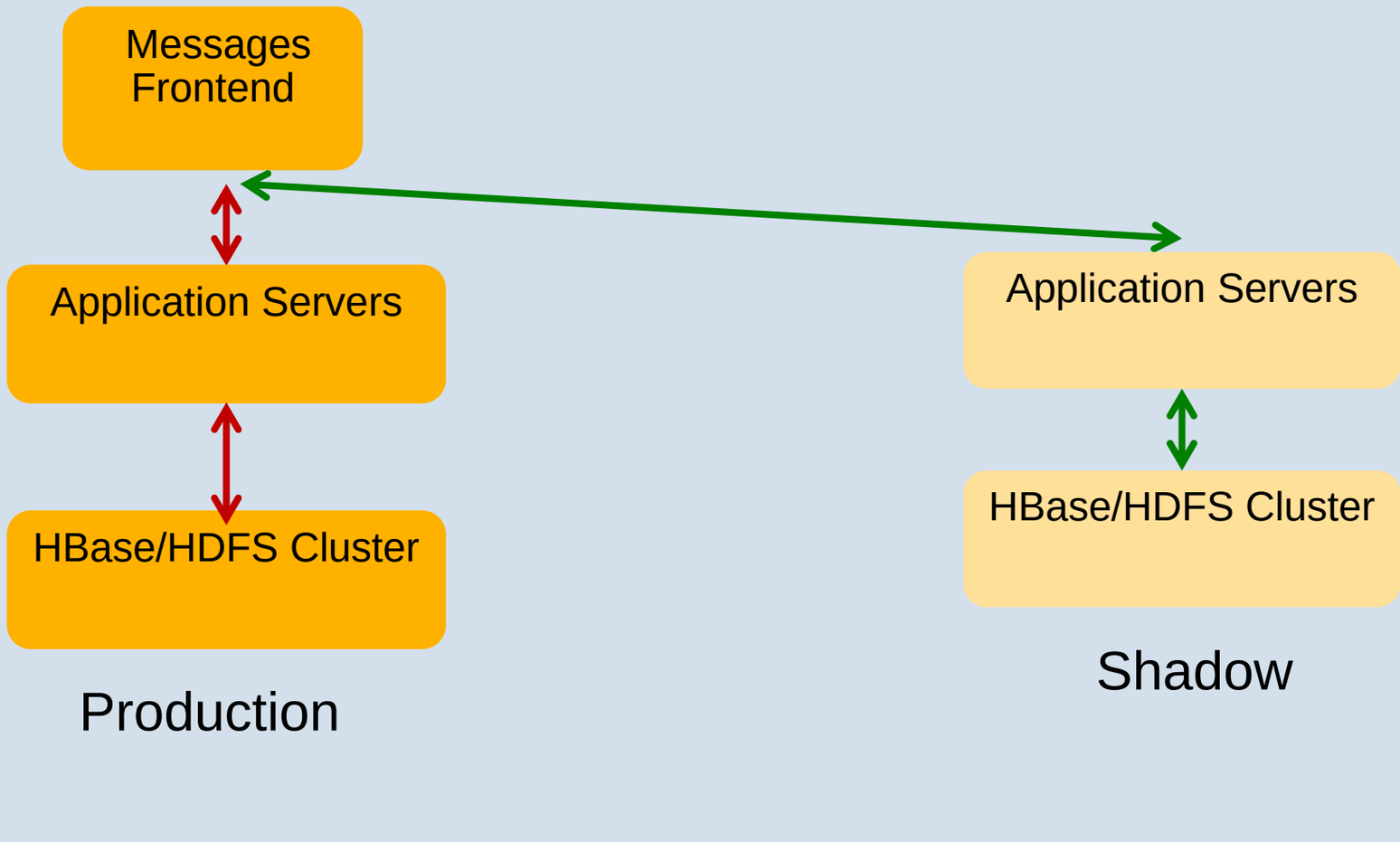
Shadow Testing: Before Rollout

- Both product and infrastructure were changing.
- Shadows the old messages product + chat while the new one was under development



Shadow Testing: After Rollout

- *Shadows the live traffic for a production cell to a shadow cell.*
- *All backend changes go through shadow cluster before prod push*



Shadow Testing

- A high % of issues caught in shadow/stress testing
- Sometimes, tried things which hadn't been tested in shadow cluster!
  - Added a rack of servers to help with performance issue
    - Pegged top of the rack network bandwidth!
      - Had to add the servers at much slower pace. Very manual ☹.
      - Intelligent load balancing needed to make this more automated.

Monitoring and Operation

- Monitoring/ops is a huge part of HBase reliability.
  - Alerts (HBCK, memory alerts, perf alerts, health alerts)
  - auto detecting/decommissioning misbehaving machines
  - Dashboards
- HBase is not very DBA/application developer friendly yet. We've spent quite bit of effort there (slow query log, job monitor, etc). Have to go through master/region server logs in order to identify issues. Even worse, these logs do not make much sense unless you know the inner details of HBase.
- We are learning along the way.

Scares & Scars!

- Not without our share of scares and incidents:
  - 1/3 of region servers in many cells went down around the same time due to a bug in config management.
  - Multiple region servers died around the same time due to DFS time out. It turns out, the NFS filer where we save the namenode log to is slow.
  - HDFS Namenode – SPOF. It happens more often than you thought it would.
  - s/w bugs. (e.g., deadlocks, incompatible LZO used for bulk imported data, etc.)
    - found a edge case bug in log recovery a few weeks ago, which can cause data loss!
  - performance spikes every 6 hours (even off-peak)!
    - cleanup of HDFS's Recycle bin was sub-optimal! Needed code and config fix.
  - transient rack switch failures

facebook

Recent HBase Development At Facebook

HBase Development at Facebook

- History with Apache HBase
  - Based on Apache HBase 0.89
  - Need time to stabilize one branch for production.
- Release Early, Release Soon
  - Benefit from Open Source
  - Contribute back to the Apache HBase 0.90, 0.92, 0.94 …
- HBase Engineer and Ops Team
  - 3 committers and 9 active contributors at Facebook

HBase Applications at Facebook

- Facebook Messages
- Realtime Data Streams and Analytics
  - Ads Insight
  - Open Graph

- More and more applications in the future…

Major Contributions from Facebook in 0.92

- HFile V2
- Compactions Improvement
  - Off peak compactions
  - Multi-thread compactions
  - Improved compaction selection algorithms
- Distributed Log Splitting
- CacheOnWrite and EvictOnClose
- Online Schema Changes

Major Contributions from Facebook (Contd.)

- Rolling Region Server Restarts
- String-based Filter Language
- Per Table/CF Metrics
- Delete Optimizations
  - Lazy Seek Optimizations (31%)
  - Delete Family Bloom Filters (41%)
  - Create Table with Presplit
  - Backups
    - RBU, CBU, DBU
    - Multi CF Bulk Loader

HFile V2 Motiviation

Opening a HFile

- Loading when region server startup
  - Block indexes and Bloom filters may take a while to load
  - Dark launch #1: 250 MB × 20 regions = 5 GB of block index data to load before a region server is up
  - No need to waste memory space for unused or infrequently CFs
- Cache miss/age out for bloom filter, huge outliers in terms of latency
  - Need a cheaper way to load a small chunk of bloom filter

HFile V2 Motivation (Contd.)

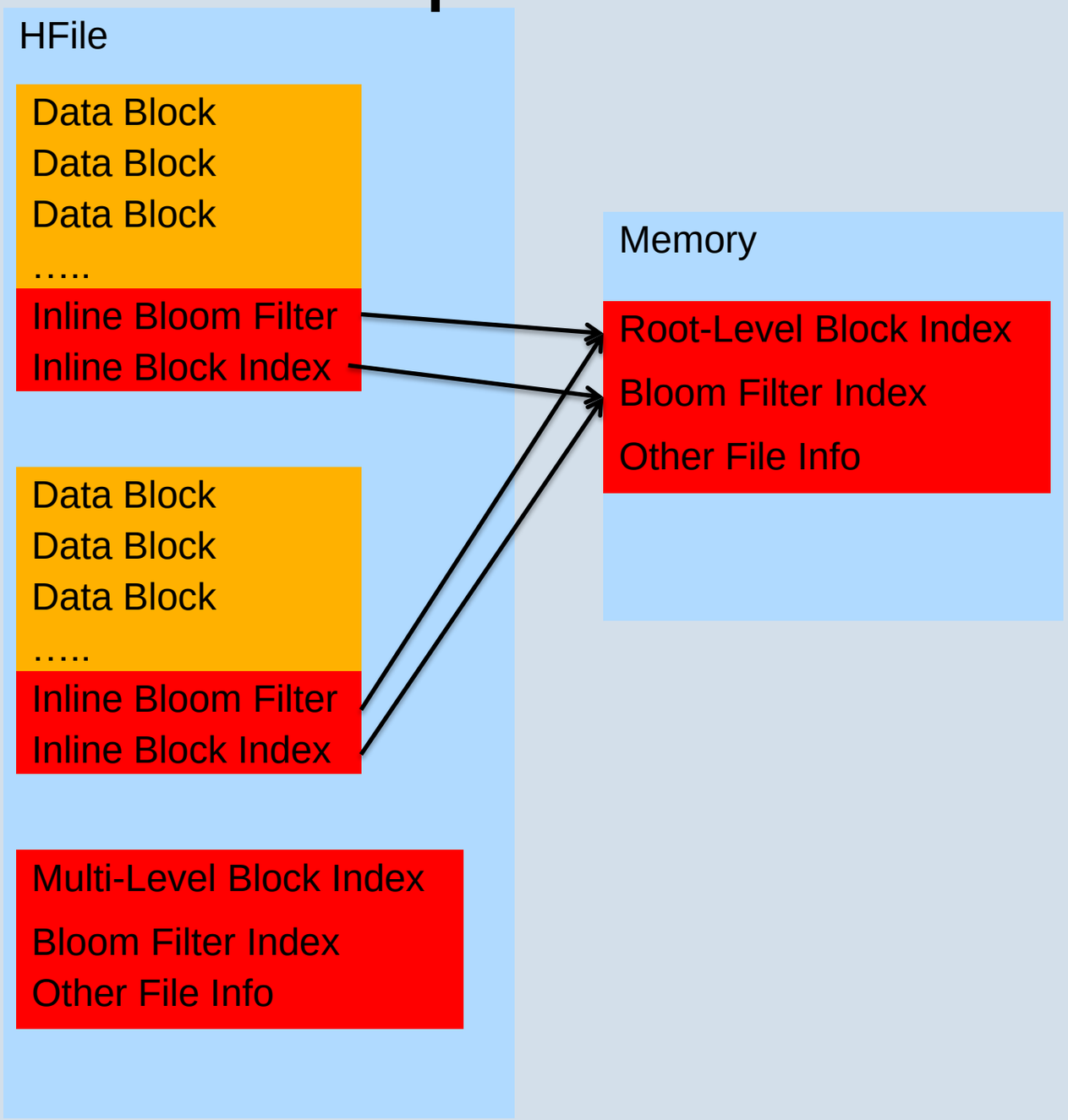
Writing HFile

- Memory footprint when writing an HFile
  - No need to hold all the bloom filter and block index when writing
  - Write block index data and Bloom filter data along with keys/values
  - More important with multithreaded compactions

Ease of bloom configuration

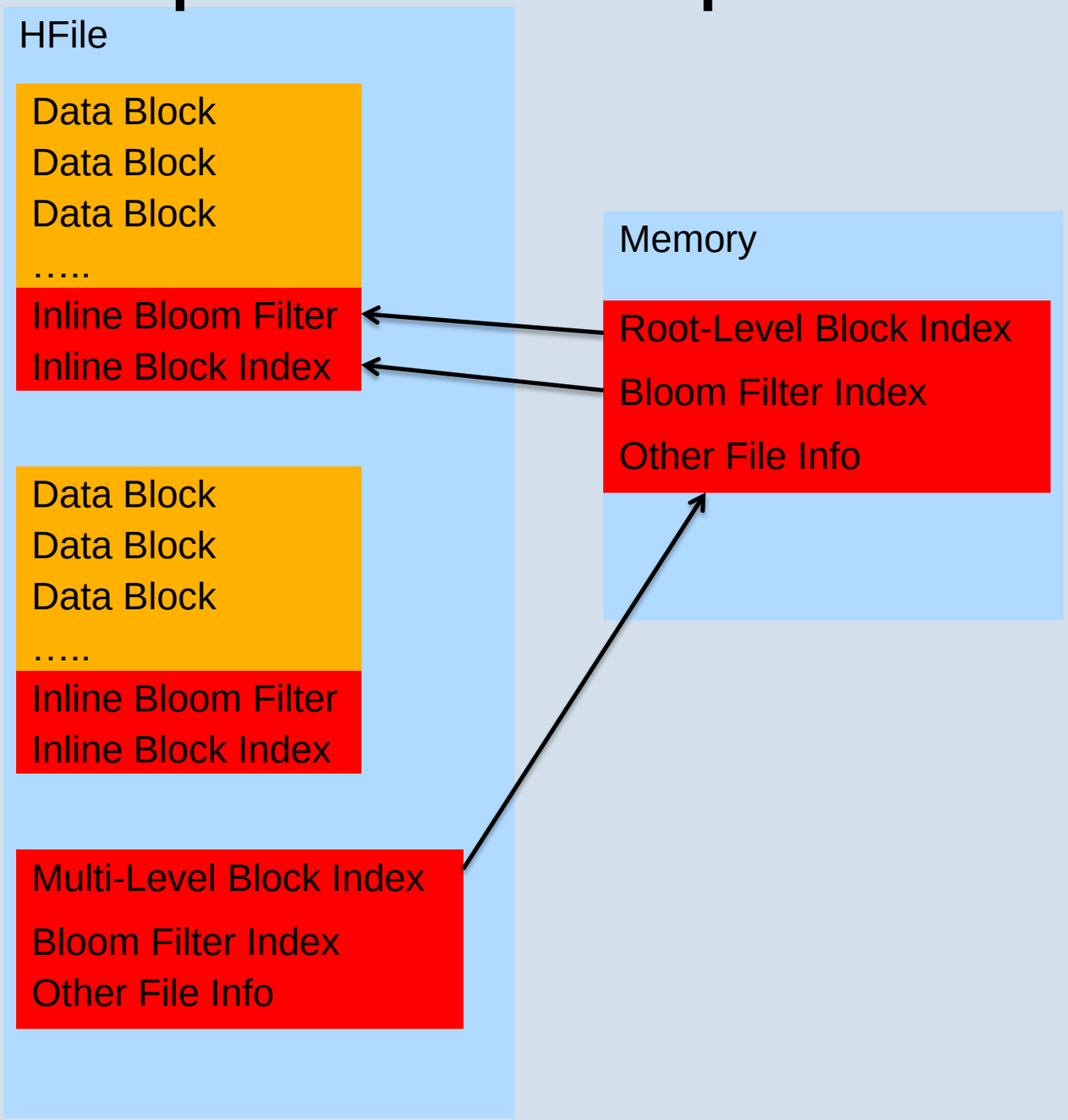
- No more guessing how many keys will be added to an HFile
  - To allocate the blooms space and other config
- New Bloom filter settings:
  - Target false positive rate
  - Allocate bloom block as a fixed size (e.g. 128 K)

Write Operation



1) Data Block
2) Write bloom/ index inline
3) Only hold root level index in memory

Open/Read Operation



Open Operation

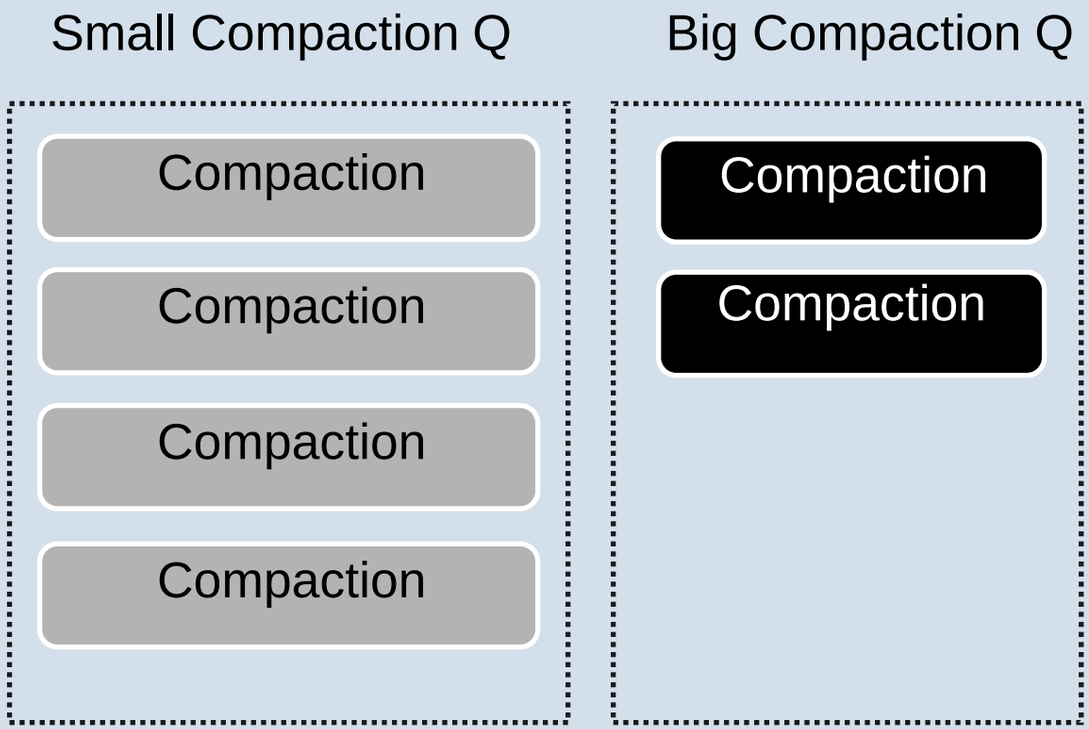
1) Version / backward compatible
2) Loading data

Read Operation

1) Lookup Root level index/Bloom Index
2) Go to leaf level index on demand (Block Cache

Compactions Improvement

- More problems!
  - Read performance dips during peak
  - Major compaction storms
  - Large compactions bottleneck
- Enhancements/fixes:
  - Staggered major compactions
  - Multi-thread compactions; separate queues for small & big compactions
  - Aggressive off-peak compactions



Compactions Improvement

- Critical for read performance
- Old Algorithm:

#1. Start from newest file (file 0); include next file if:

- size[i] < size[i-1] * C (good!)

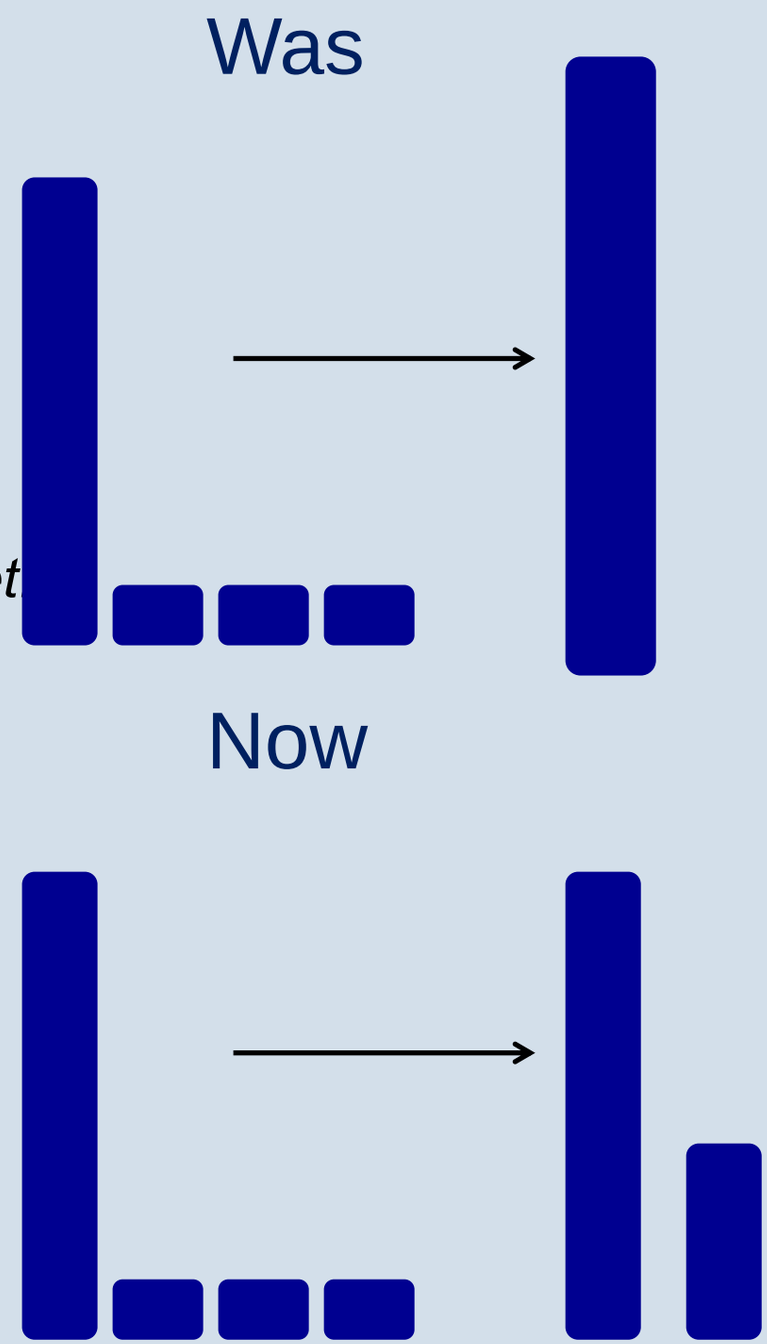
#2. Always compact at least 4 files, even if rule #1 isn't met

Solution:

#1. Compact at least 4 files, *but only if eligible files found*.

#2. Also, new file selection based on summation of sizes.

size[i] < (size[0] + size[1] + …size[i-1]) * C



Metrics, metrics, metrics…

- Initially, only had coarse level overall metrics (get/put latency/ops; block cache counters).
- Slow query logging
- Added Per Table/Column Family stats for:
  - ops counts, latency
  - block cache usage & hit ratio
  - memstore usage
  - on-disk file sizes
  - file counts
  - bytes returned, bytes flushed, compaction statistics
  - stats by block type (data block vs. index blocks vs. bloom blocks, etc.)
  - bloom filter stats

Metrics (contd.)

- HBase Master Statistics:
  - Number of region servers alive
  - Number of regions
  - Load balancing statistics
  - ..
- All stats stored in Facebook's Operational Data Store (ODS).
- Lots of ODS dashboards for debugging issues
  - Side note: ODS planning to use HBase for storage pretty soon!

Backups

- Stage 1 – RBU or in-Rack Backup Unit
  - Backup stored on the live DFS used by Hbase
- Stage 2 – CBU or in-Cluster Backup Unit
  - Backup on another DFS in the same DC
- Stage 3 – DBU or cross-DataCenter Backup Unit
  - Backed up on another DFS in a different DC
  - Protects against DC level failures

Backups

Protection against	**Stage 1 (RBU)**	**Stage 2 (CBU)**	**Stage 3 (DBU/ISI)**
Operator errors	X	X	X
Fast Restores	X	X	
Stable Storage		X	X
Heterogeneous Racks		X	X
Datacenter Separation			X

Backups Tools

- Backup one, multiple or all CF’s in a table
- Backing up to Stage 1 (rbu), Stage 2 (cbu) and Stage 3 (dbu)
- Retain only last n versions of the backups
- Restore backups to a point in time
  - Restore to a running or offline cluster
  - Restore to the same table name or a new one
- Restore tables on the Stage 2 (cbu) as needed
- Verify a percentage of data in the backups (this may not be do-able without app knowledge)
- Alerts and dashboards for monitoring

Need to keep up as data grows on you!

- Rapidly iterated on several new features while in production:
  - Block indexes up to 6GB per server! Cluster starts taking longer and longer. Block cache hit ratio on the decline.
    - Solution: HFile V2
      - Multi-level block index, Sharded Bloom Filters
  - Network pegged after restarts
    - Solution: Locality on full & rolling restart
  - High disk utilization during peak
    - Solution: Several “seek” optimizations to reduce disk IOPS
      - Lazy Seeks (use time hints to avoid seeking into older HFiles)
      - Special bloom filter for deletes to avoid additional seek
      - Utilize off-peak IOPS to do more aggressive compactions during

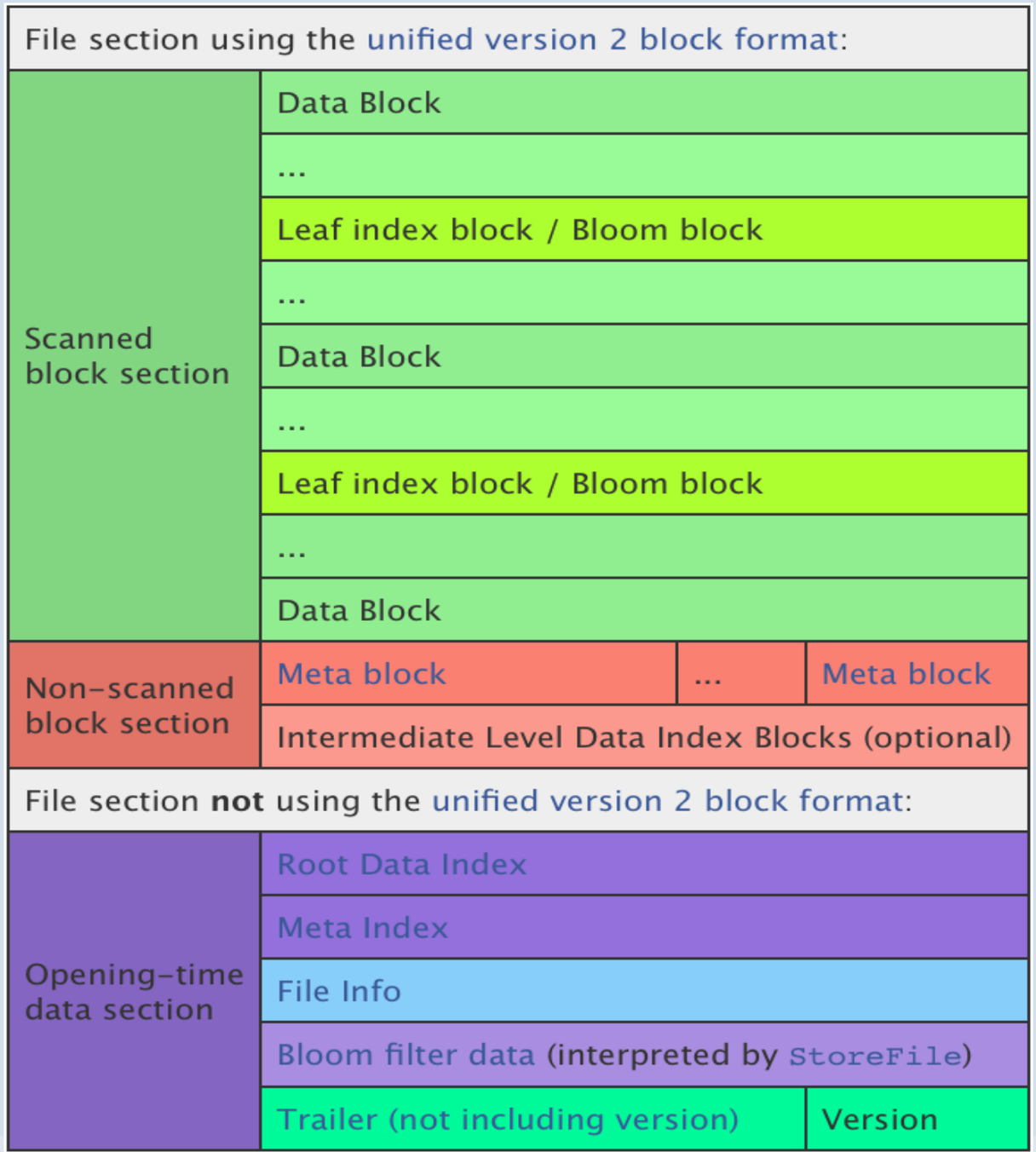
Future Work

- Reliability, Availability, Scalability!
- Lot of new use cases on top of HBase in the works.
  - HDFS Namenode HA
  - Recovering gracefully from transient issues
  - Fast hot-backups
  - Delta-encoding in block cache
  - Replication (Multi-Datacenter)
  - Performance (HBase and HDFS)
  - HBase as a service Multi-tenancy
  - Region Assignment based on Data Locality

Thanks! Questions?

facebook.com/engineering

Open Operation



Open Operation
1) Version / backward compatible
2) Loading data

Write Operation
1) Data Block
2) Write bloom/ index inline
3) Only hold root level index in memory

Read Operation
1) Lookup Root level index/Bloom Index
2) Go to leaf level index on demand (Block Cache)

Scan Operation
1) SeekTo
2) Next
3) ReSeek

HBase-HDFS System Overview