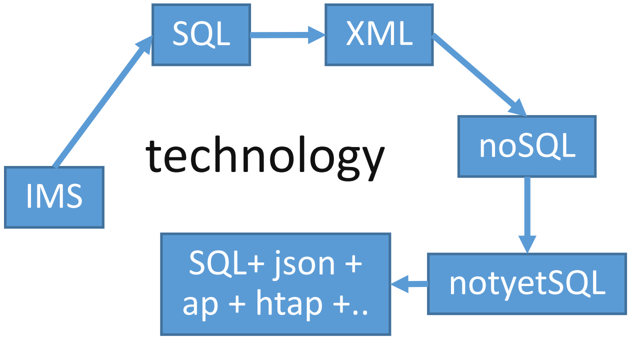
Wildfire: Evolving Databases for New-Gen Big Data Applications

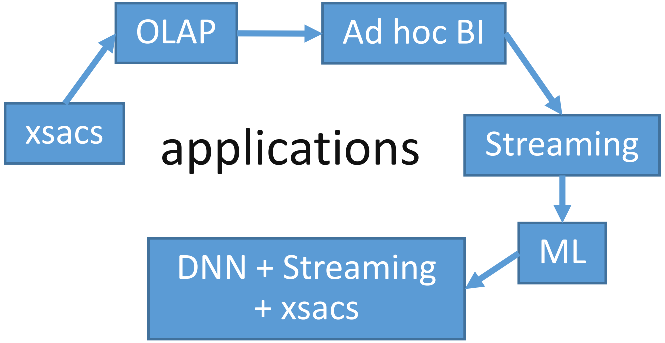
R. Barber, C. Garcia-Arellano, R. Grosman, R. Mueller, **V. Raman**,
R. Sidle, M. Spilchen, A. Storm, Y. Tian, P. Tozun, D. Zilio,
M. Huras, G. Lohman, C. Mohan, F. Ozcan, H. Pirahesh

IBM

What are these New-Gen Big Data Applications?

- World has changed a lot since the 70s
  - Automating business processes → AI everywhere
- But databases are still hot

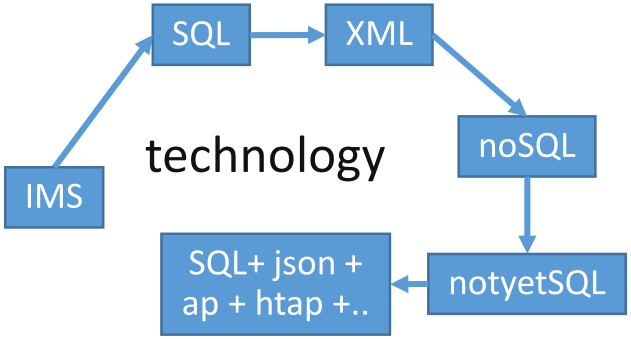




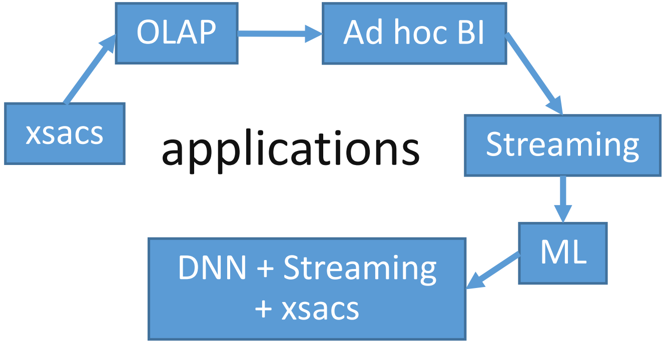
What are these New-Gen Big Data Applications?

- World has changed a lot since the 70s
  - Automating business processes → AI everywhere
- But databases are still hot

And the apps want even more from the database!
-- Higher ingest and update rates
-- versioning, time-travel
-- Ingest and Update anywhere, anytime (“AP” system)
-- More real-time analytics (HTAP)
-- tons of analytics
==> database cannot hold data in proprietary store



IMS
SQL
XML
noSQL
notyetSQL
SQL+ json + ap + htap +..
technology



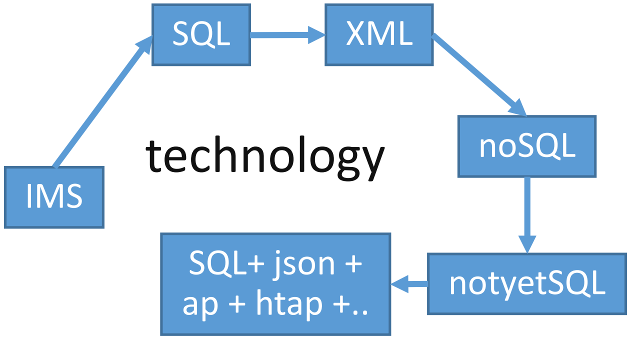
xsacs
OLAP
Ad hoc BI
Streaming
ML
DNN + Streaming + xsacs
applications

What are these New-Gen Big Data Applications?

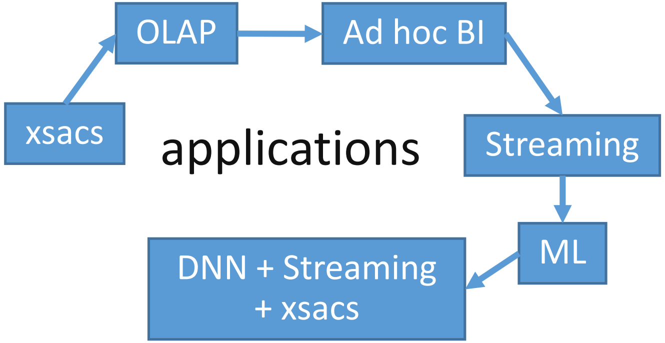
- World has changed a lot since the 70s
  - Automating business processes → AI everywhere
- But databases are still hot

And the apps want even more from the database!
-- Higher ingest and update rates
-- versioning, time-travel
-- Ingest and Update anywhere, anytime ("AP" system)
-- More real-time analytics (HTAP)
-- tons of analytics
==> database cannot hold data in proprietary store

But still want the traditional database goodies:
Updates
Transactions (not eventual consistency)
Point Queries / Indexes
complex queries (joins, optimizer, ..)



technology
IMS → SQL → XML → noSQL → notyetSQL → SQL+ json + ap + htap +..



applications
xsacs → OLAP → Ad hoc BI → Streaming → ML → DNN + Streaming + xsacs

Example: Health Care

Convergence of Prevention/Monitoring (sensors on *healthy* people)
and Cure (healthcare setting)

Example: Health Care

Convergence of Prevention/Monitoring (sensors on *healthy* people)
and Cure (healthcare setting)

High ingest rates

Complex queries, joins, ..

AP: cannot wait for mothership to be reachable

Lots of point queries

Want analytics on latest readings

Looking for outliers => cannot drop data, need durability

Eventual consistency is a pain

```
V1 ← lookup(k1);
V2 ← lookup(k1);
```

// if V1 finds match and V2 doesn't, how to test this app?

Wildfire Goals

HTAP: transactions & queries on same data

- Analytics over latest transactional data
- Analytics over 1-sec old snapshot
- Analytics over 10-min old snapshot

Leapfrog transaction speed, with ACID

- Millions of inserts, updates / sec / node
  - Multi-statement transactions
  - With async quorum replication (sync option)
- Full primary and secondary indexing
  - Millions of gets / sec / node

Open Format

- All data and indexes in Parquet format on shared storage
  - No LOAD
  - Directly accessible by platforms like Spark

Multi-Master and AP

- disconnected operation
- Snapshot isolation, with versioning and time travel
  - Conflict resolution based on timestamp

Wildfire Goals

HTAP: transactions & queries on same data

- Analytics over latest transactional data
- Analytics over 1-sec old snapshot
- Analytics over 10-min old snapshot

Open Format

- All data and indexes
  in Parquet format on shared storage
  - No LOAD
  - Directly accessible by platforms like Spark

Leapfrog transaction speed, with ACID

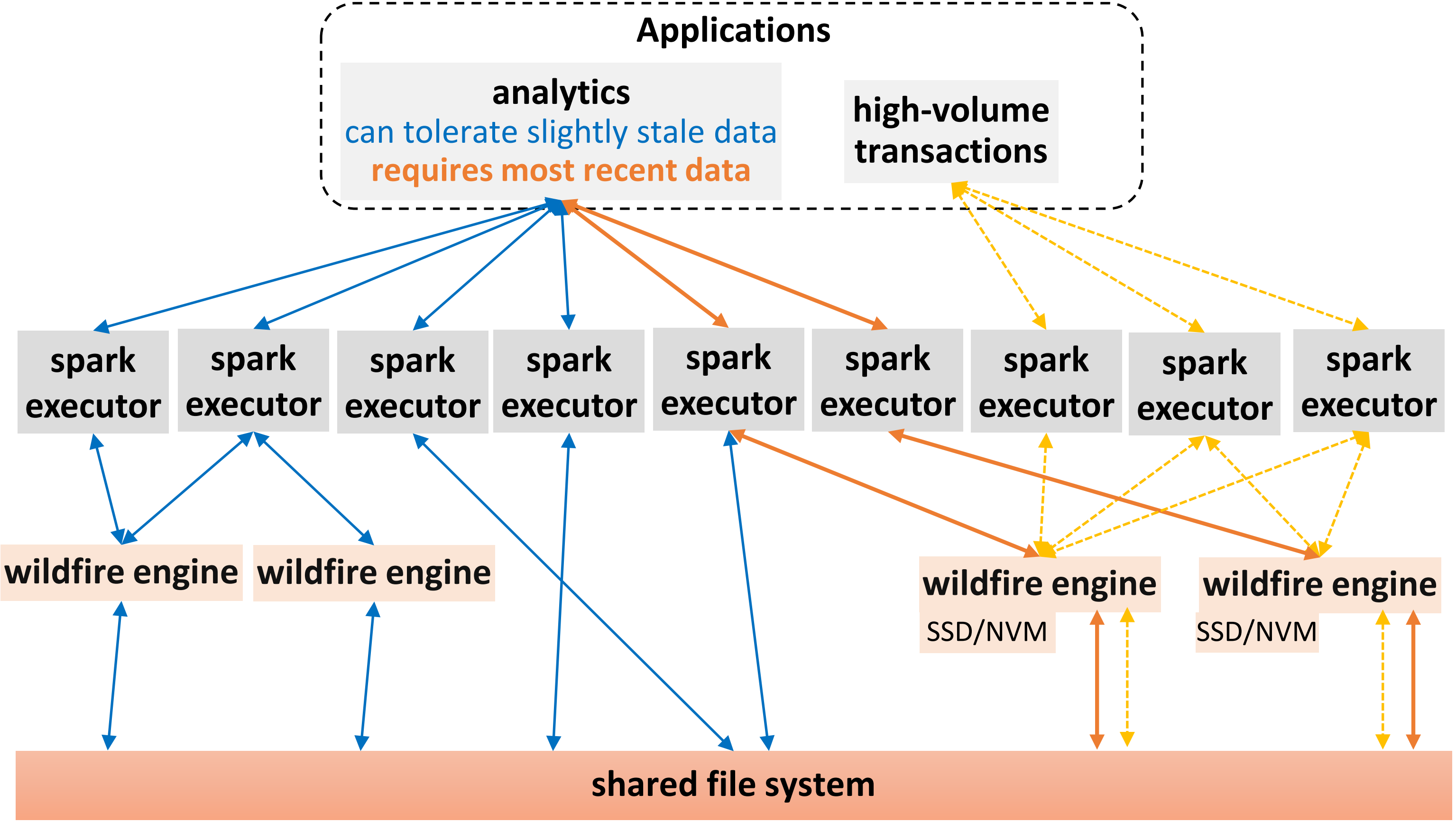
- Millions of inserts, updates / sec / node
  - Multi-statement transactions
  - With async quorum replication
    (sync option)
- Full primary and secondary indexing
  - Millions of gets / sec / node

Multi-Master and AP

- disconnected operation
- Snapshot isolation,
  with versioning and time travel
  - Conflict resolution based on timestamp

Challenge: getting all of these simultaneously

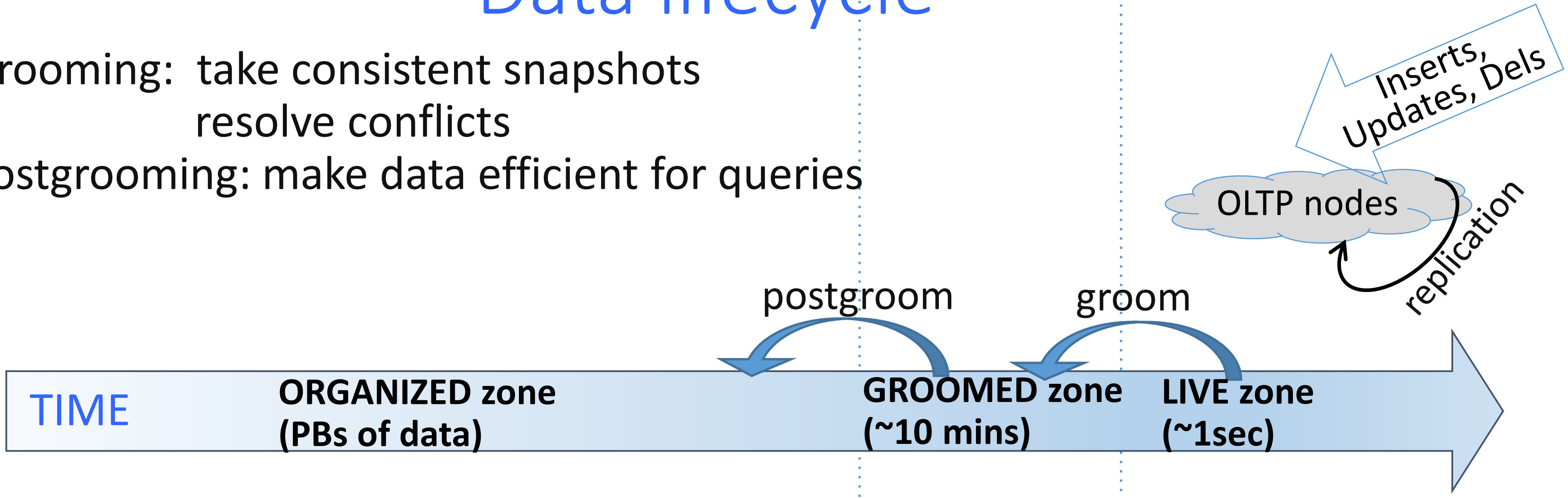
Wildfire architecture



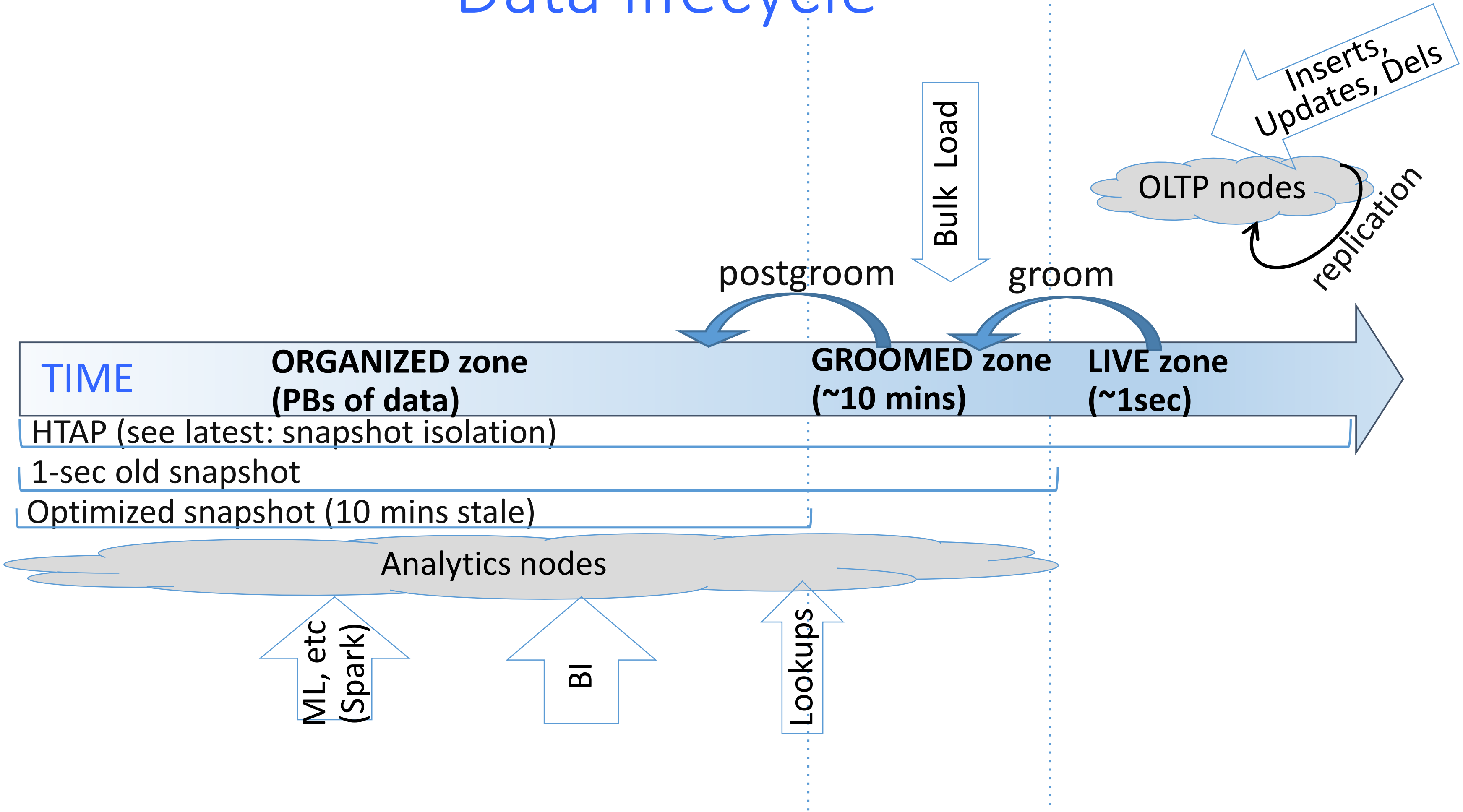
Data lifecycle

Grooming: take consistent snapshots
resolve conflicts

Postgrooming: make data efficient for queries



Data lifecycle



Inserts, Updates, Dels

OLTP nodes

replication

Bulk Load

postgroom

groom

TIME

ORGANIZED zone (PBs of data)

GROOMED zone (~10 mins)

LIVE zone (~1sec)

HTAP (see latest: snapshot isolation)

1-sec old snapshot

Optimized snapshot (10 mins stale)

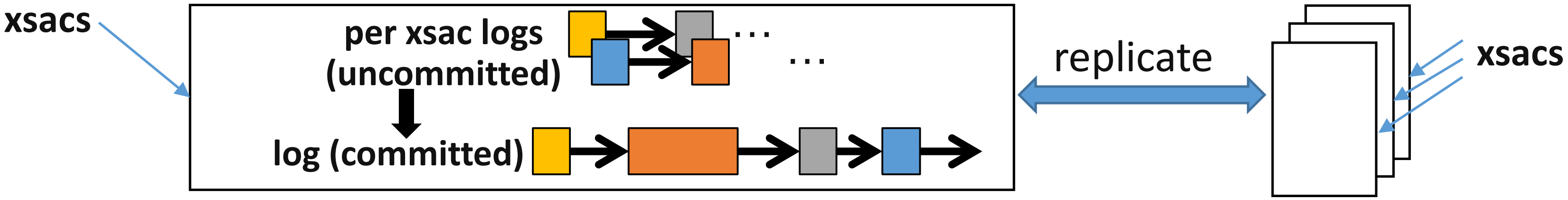
Analytics nodes

ML, etc (Spark)

BI

Lookups

Live Zone



What happens at Commit

1. append xsac deltas (Ins/Del/Upd) to common log; replicated in background
2. flush to local SSD
3. status-check if changes are quorum-visible (via heartbeats)
   -- can time-out

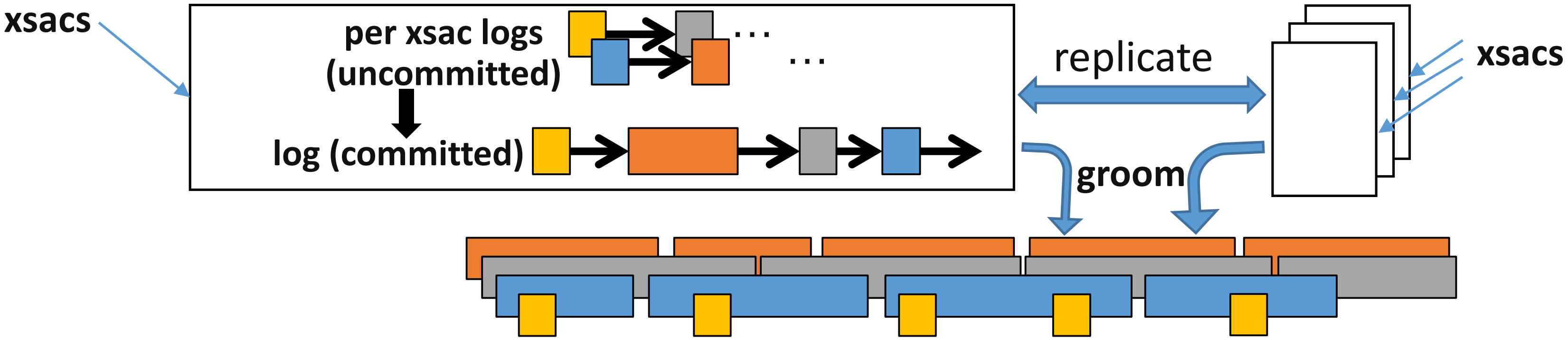
AP: Commit does not wait for other nodes; **conflicts are resolved *after* commit**

(have syncwrite option for higher durability)

Read monotonicity: Queries always read quorum-visible state

- Hence, **later queries see a superset of what prior queries saw**

Grooming data (Live → Groomed zone)



- **Grooming is when conflicts are resolved**

  -- take quorum-visible deltas, form data blocks, and publish to shared file system

  -- groomed zone is always a consistent snapshot

- All deltas (insert/delete/update) are **upserts**: key, (values)*, beginTime
  - beginTime initialized at commit as (localTime | nodeID)
- **No assumption about clock synchronization or speed of replication**

  -- yet, we get read monotonicity
  - Idea: groom sets beginTime ← groomTime|localTime|nodeID
- **Conflict resolution: versioning, based on beginTime**

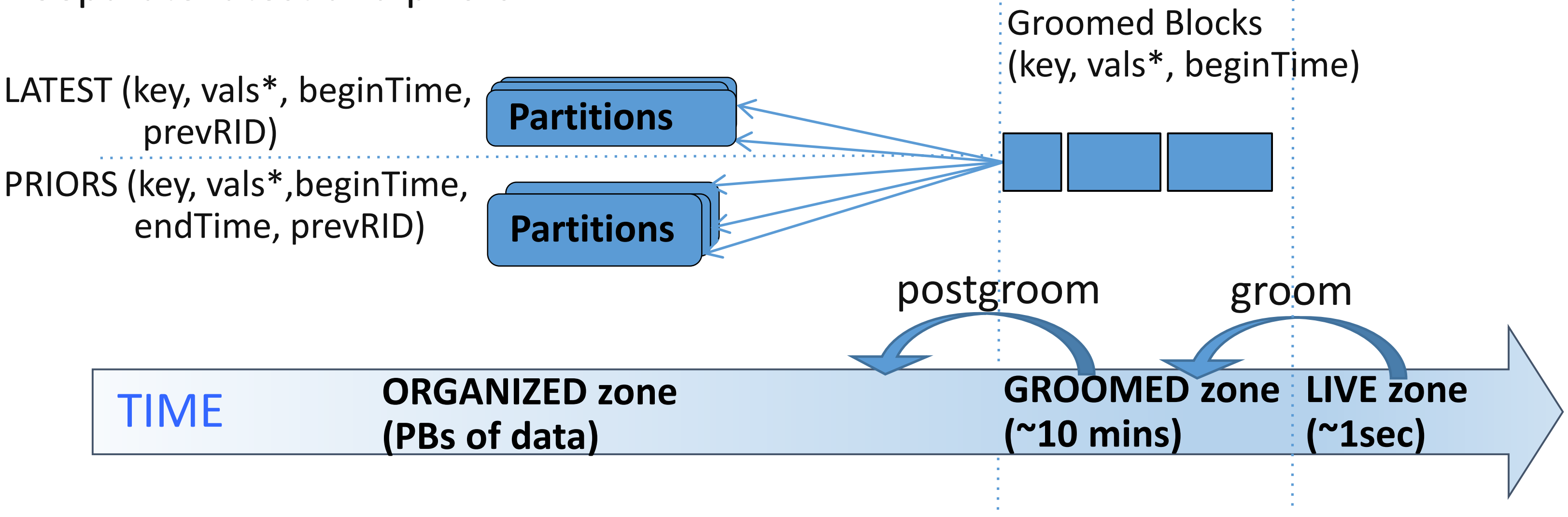
Postgrooming

Queries should run fast (BI and point)

- Compute endTime and prevRID
  - And deal with immutable storage system!
- Partition (along multiple dimensions)
- Build primary and secondary indexes

Want ready access to latest version (for the simple readers)

- Separate latest and priors



Groomed Blocks (key, vals*, beginTime)

LATEST (key, vals*, beginTime, prevRID)

PRIORS (key, vals*,beginTime, endTime, prevRID)

TIME

ORGANIZED zone (PBs of data)

GROOMED zone (~10 mins)

LIVE zone (~1sec)

OLAP queries via SparkSQL

- Extensions to both Catalyst Optimizer and Data Source API
- A new Spark context for SQL
- Catalyst Optimizer
    - Query HCatalog for table schemas
    - Identify plan to send to Wildfire
    - Compose a compensation plan (if needed)
- Data Source API
    - SparkSQL Logical plan → Wildfire plan
    - Plan submission to Wildfire & result passing
- Compensation plan (if needed) executed in SparkSQL
- Paper has details about pushdown analysis

POST-TRUTH

- Big data needs updates, indexes, complex queries, transactions
- AP is the reality
- PB databases will not live in proprietary storage
- It is possible to do ACID with AP
- DBMS can adopt open data formats and immutable stores – while still being fast

POST-ER-TRUTH

- Multi-shard transactions
- Serializability with AP