Wildfire: Fast HTAP over loosely coupled Nodes

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

IBM Research, IBM Analytics

Apps emit tons of events

IOT

Mobile commerce

wearables

Today's DBMS is too slow for this firehose
And too costly (storage)

- Events happen in the real-world
  – not tied to transaction commit

Place Order

Withdraw Cash

- Events always have *asterisks*
  - Concurrent events
  - Event ordering done later

Apps emit tons of events

IOT

Mobile commerce

we

Today's DBMS is too slow for thi
And too costly (storage)

- Events happen in the real-wor
  – not tied to transaction comn

Place Order

Withdraw Cash

- Events always have *asterisks*
  - Concurrent events
  - Event ordering done later

Events need Availability and Consistency

- Multi-master, disconnected operation
- Today: Consistency achieved via application logic
  - Compensation, apologies, coupons, …
  - Weak atomicity and durability
- Growing pressure for DBMS to give both Availability and Consistency
  - Due to globalization
  - e.g., credit cards

Apps emit tons of events

IOT

Mobile commerce

wearables

Today's DBMS is too slow for this fire
And too costly (storage)

- Events happen in the real-world – not tied to transaction commit

Place Order

Withdraw Cash

- Events always have *asterisks*
  - Concurrent events
  - Event ordering done later

Events need Avail. and Consistency

- Multi-
disconn
- Today:
applica
  - Co
cou
  - We
- Growi
both A
  - Du
  - e.g

Events need HTAP

- Mobile commerce
- Retail: inventory analysis, shipping time analysis
- Securities trading: global risk analysis
  - Margin rules
- Complex analysis within transaction
  - Touching lots of rows
  - Handled poorly by both 2PL and OCC
- Analysis involves far more than SQL
  - Graph, machine learning, …

Apps emit tons of events

IOT Mobile wearables

Events need Avail. and Consistency

Events need HTAP

WildFire Goals

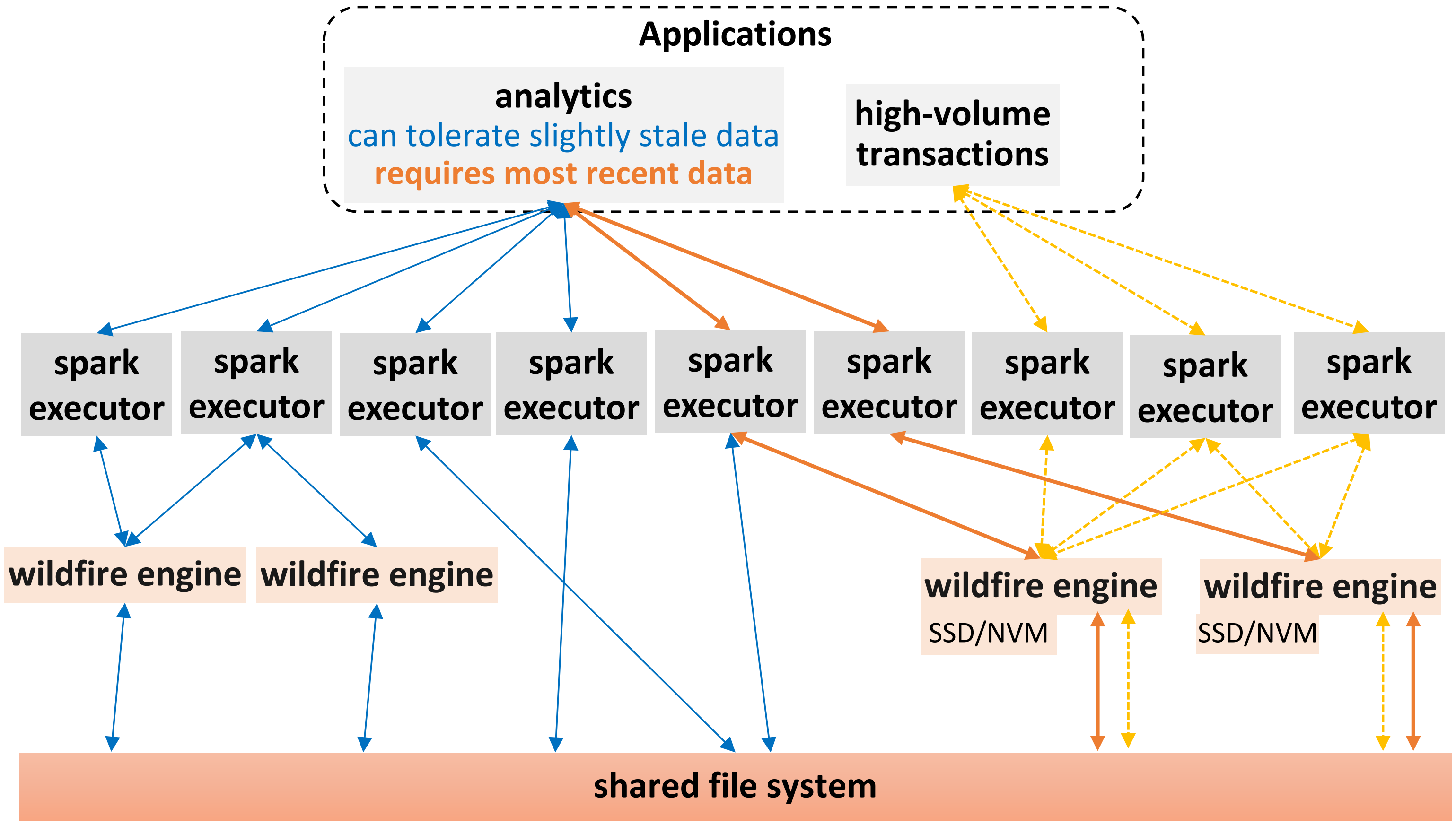
Peak transaction speed
- Inserts/updates: keep up with bandwidth of durability mechanism (today: 1e6/s/node)
- Full indexing: keep up w. random access speed (hash tables + atomics on SSD → NVRAM)
- Fully versioned

Open Format
- All data groomed to Parquet format on shared storage
- Directly accessible by analytics platforms (e.g., Spark)

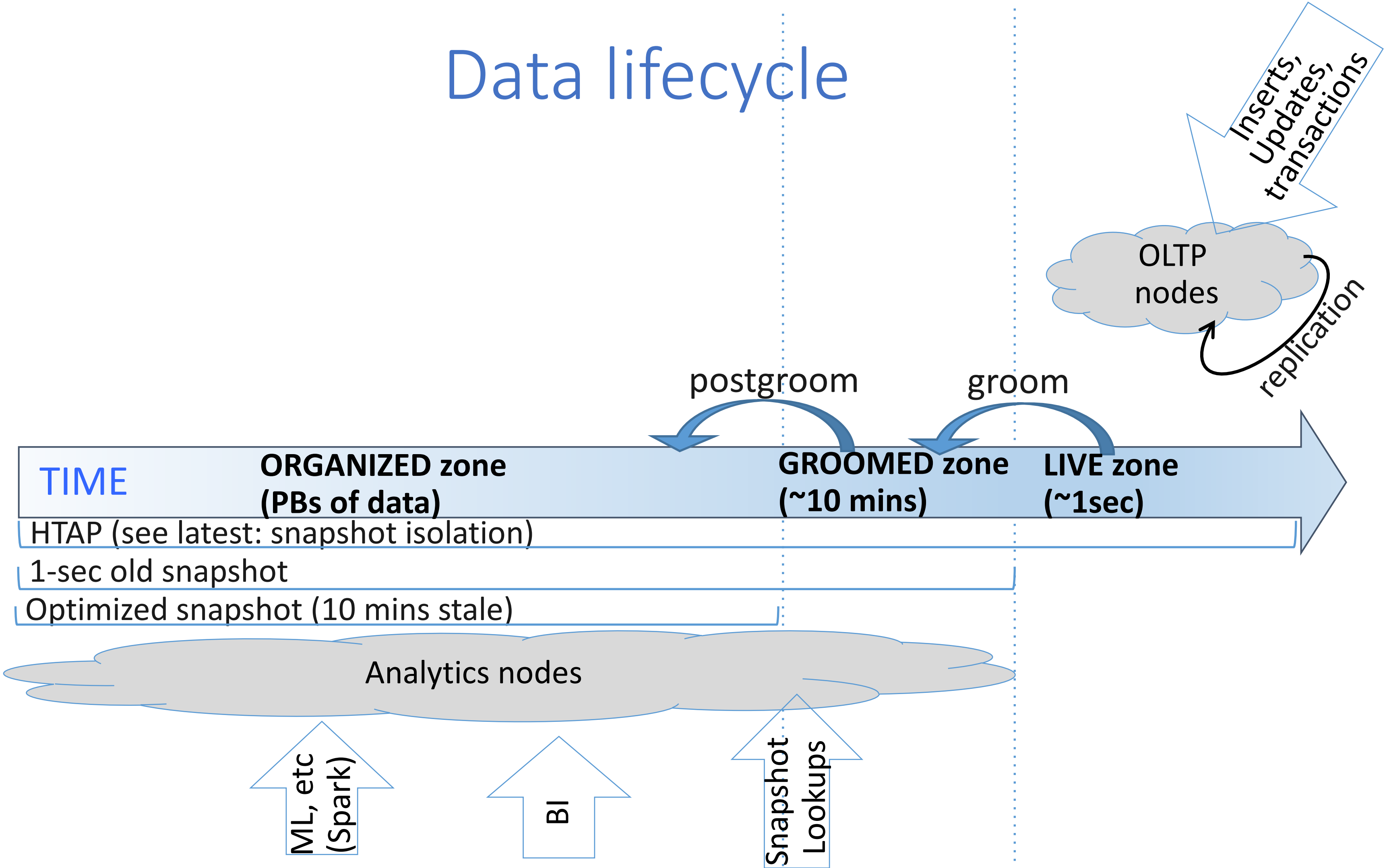
Multi-Master and ACID
- Commit is local (no consensus)
- High-value events can wait for conflict resolution (*after commit*)

HTAP
- In-transaction analytics
- Analytics over snapshot (1s or 10mins)
  - Higher throughput and more economical scaleout

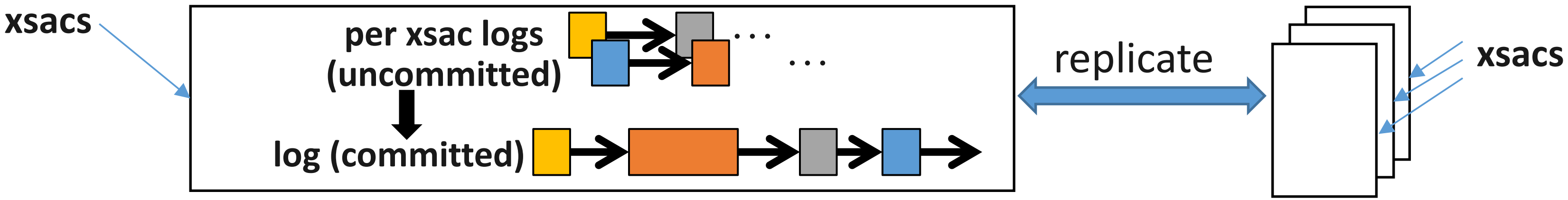
Wildfire architecture



Data lifecycle



Live Zone: Thin Commit



What happens at Commit

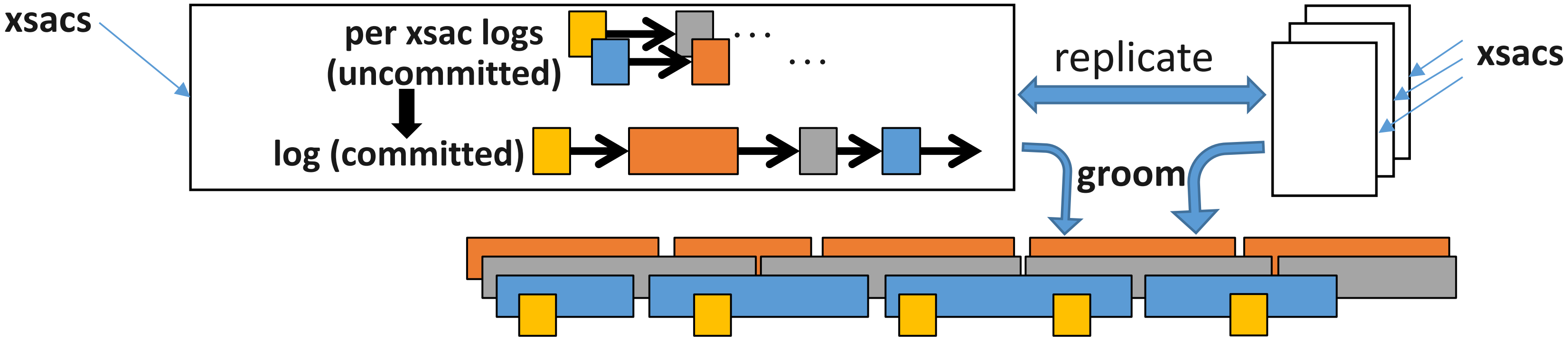
1. append xsac deltas (Ins/Del/Upd) to common log; replicated in background
   -- everything is an upsert: key, (values)*
   -- no synchronous conflict resolution
2. flush to local SSD
3. high-value xsacs wait for grooming (to timestamp the xsac and resolve conflicts)
   -- can time-out

Driven by speed

- No tracking down prior versions
- No indexing
- No waiting for consensus with other nodes

multiple versions
for same key can coexist
-- queries pick right version based
on their xsac snapshot

Grooming (Live → Groomed zone)



- Runs distributed consensus to timestamp the xsacs (pick serialization order)
  - take quorum-visible deltas, form data blocks, and publish to shared file system
  - Add beginTS field to each row: (groomTS | localTime | nodeID)
- Conflicts and constraints resolved lazily (including logical rollback)
- No assumption about
  - Clock synchronization
  - Partitioning / failures (multiple groomers possible)
- Details offline

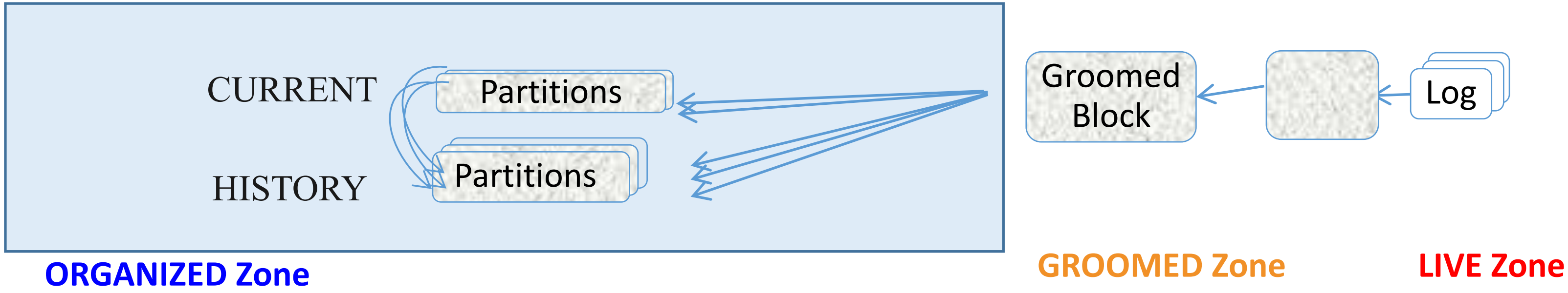
Postgrooming

Organize data so that Queries can run fast
(and deal with immutable storage!)

- Resolve conflicts (and stamp xsacs with resolution/rollback status)
- Compute endTime and prevRID
- Partition data (along multiple dimensions)
- Maintain primary indexes, secondary indexes, and synopses

Continual refinement – done in background

- big challenge is supporting concurrent groom and queries



Postgrooming: index maintenance

- Primary: maps key hash → RID [+ include columns]
- Secondary: maps key hash → pkey [+ TSN hint]

- Works in background to add groomed records to indexes
- Index is variant of LSM tree
  - Merging in background
  - Lives in multiple tiers: memory, SSD, shared storage, and purged

- Multiple versions of each key live in index

Postgrooming: computing endTS

- At groom, each row has a beginTS - but no endTS
  - Without postgrooming, every table scan must group-by on primary key
    - to pick appropriate version
- Postgroom picks groomed rows and assigns endTS
  1. Massive set intersection:
     PrimaryKeyIndex (key→latestRID) with RecentlyGroomed
     prior versions can be arbitrarily far back!
  2. Squeeze in endTS into data blocks (with concurrent readers!)

Postgrooming: resolving transaction status (single-shard)

- **ReadSet tracking is pessimistic, especially with complex queries**
  - Eg: currentInventory ← select sum(..) from ledger where productId=_
    if (currentInventory > 2) insert into ledger values (-1, productId, …); # buy one item
- **Constraint-based resolution**
  - Transaction Type1: { read* ; fullySpecifiedWrite*; If (trigger) { rollback or other action }}
    - ATM withdrawal, Securities trading (higher-granularity checks)
  - Transaction Type2: { read* ; if (trigger) { fullySpecifiedWrite* } }
    - Submit order
  - Trigger Condition is checked as a continuous query (incrementally feed new deltas)
- **ReadSet-based resolution**
  - {Read*; write*; if (trigger) {rollback or other action}}
  - (trigger|| readset changed since query snapshot) is checked as a continuous query
  - More general transactions (eg RWRW) hard to check incrementally
    - fall back to traditional readset tracking

Postgrooming: resolving transaction status (multi-shard)

- Each transaction can produce multiple deltas, spread across groom cycles
- No 2PC
- Each transaction stamped with its *delta count*
- Resolve only considers a delta if all deltas of that transaction are available

Concluding Remarks

- OLTP/OLAP separation is going away

- DBMS needs to be much faster, and
  stop controlling the data format, and
  stop controlling the data storage, and
  stop controlling the kinds of analytics
    - DBMS can be the manager for event data → $V^{V}LDB$

Thank you

BACKUP