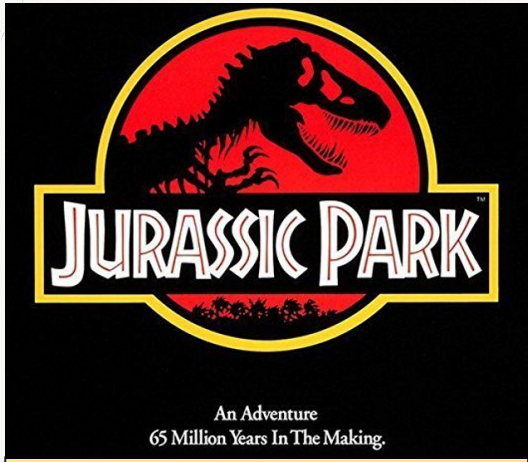
These slides are best seen with Powerpoint/Google Slides in presentation mode since they have animation

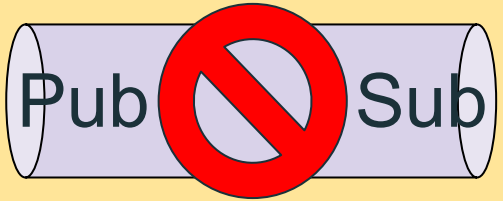
Understanding the limitations of pubsub systems

Atul Adya, Phil Bogle, Colin Meek

(and help from Jonathan Ellithorpe)

databricks





This paper: only 12 years in the making

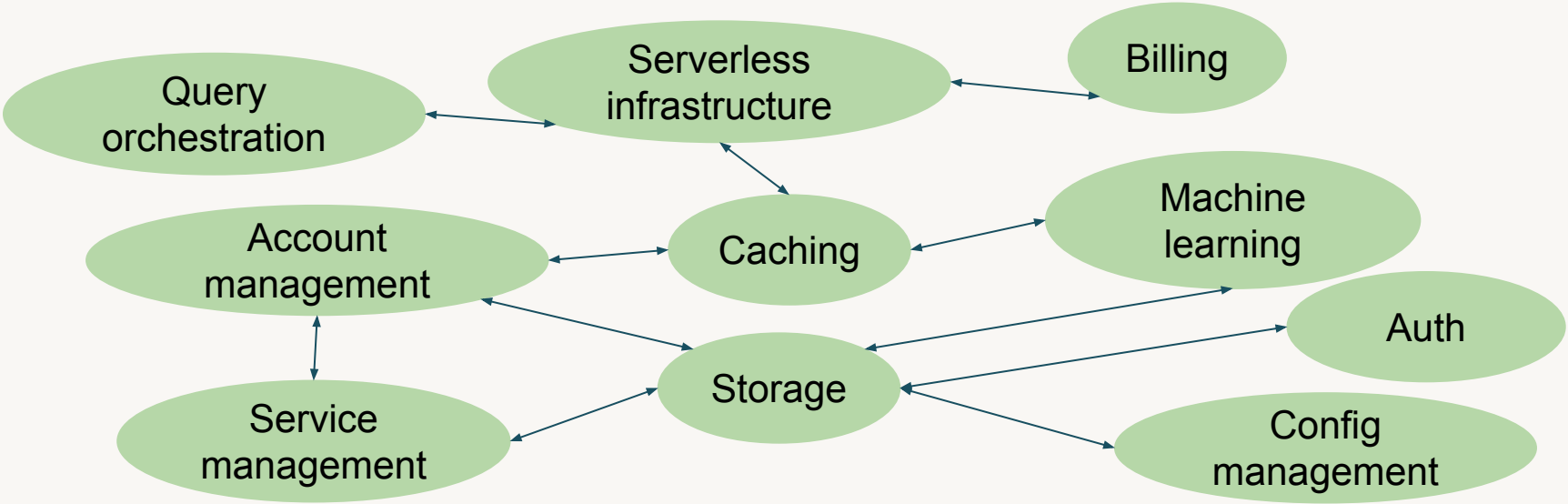
Pubsub abstraction violates the end-to-end argument

Explicit storage with Watch is the way to go

Talk Outline

- What is pubsub
- Problems with pubsub
- Our approach: Watch with explicit store
- Research directions

Connecting datacenter services



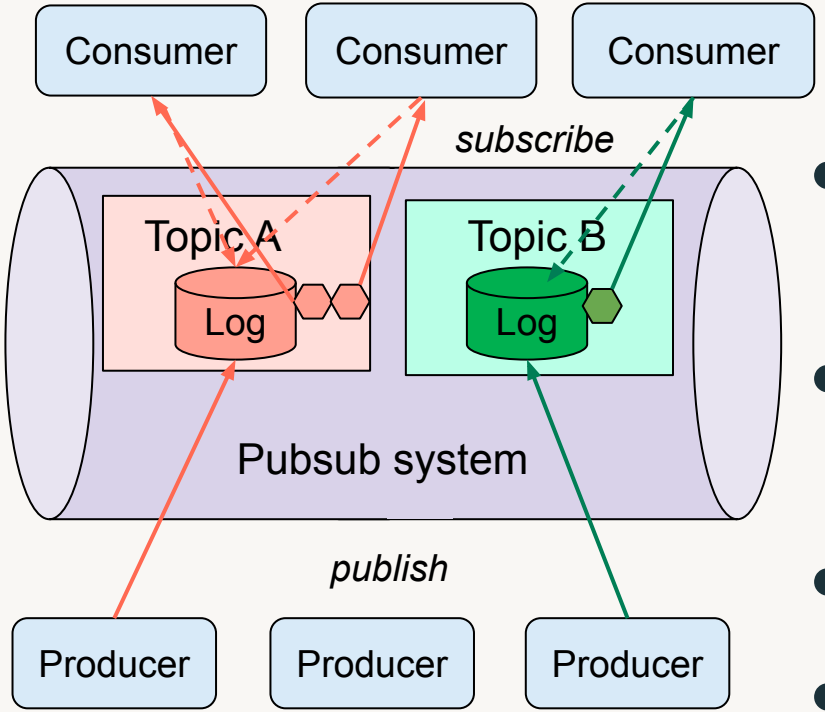
- Datacenter services need to interact
- RPCs are the most common way to communicate

Communicating beyond RPCs

- Applications need more capabilities than RPCs:
  - Decoupling of services, e.g., senders/receivers could work at different speeds/availability
  - Reliable: No loss of messages
- Can use database but have to poll frequently

Instead, use an abstraction designed for this …

Pubsub: a reliable messaging abstraction



- Topics: Logical channel for sending/receiving messages
- Internal durable log to buffer sender messages
- Topics & log enable “loose coupling” and “reliable delivery”
- E.g., Kafka, GCP Pubsub
- Many uses, e.g. database replication and cache freshness

Talk Outline

- What is pubsub
- Problems with pubsub
- Our approach: Watch with explicit store
- Research directions

What is the main problem with Pubsub?

The Pubsub abstraction violates the end-to-end argument in system design

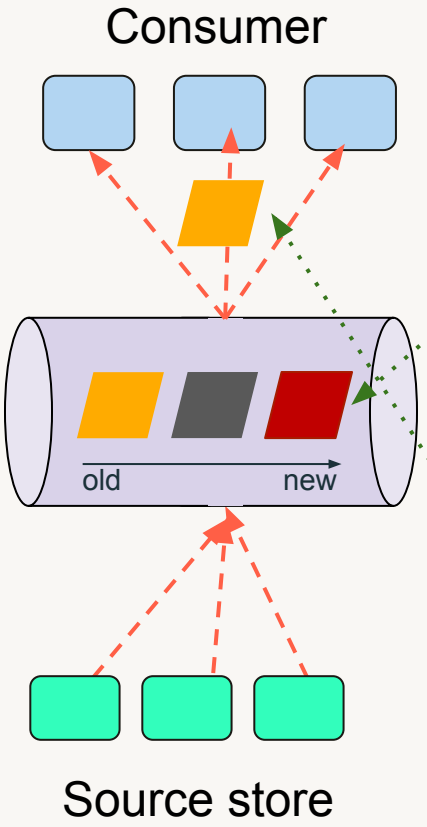
Why pubsub violates end-to-end principle

Violates end-to-end principle by not giving control to “ends”

- Pubsub log contract hurts correctness and/or scalability
  - Log not really durable
  - Hidden log is not controlled by consumer end
  - Ordering guarantee is not relative to source end

- Pubsub breaks end-to-end guarantees in the presence of dynamic partitioning

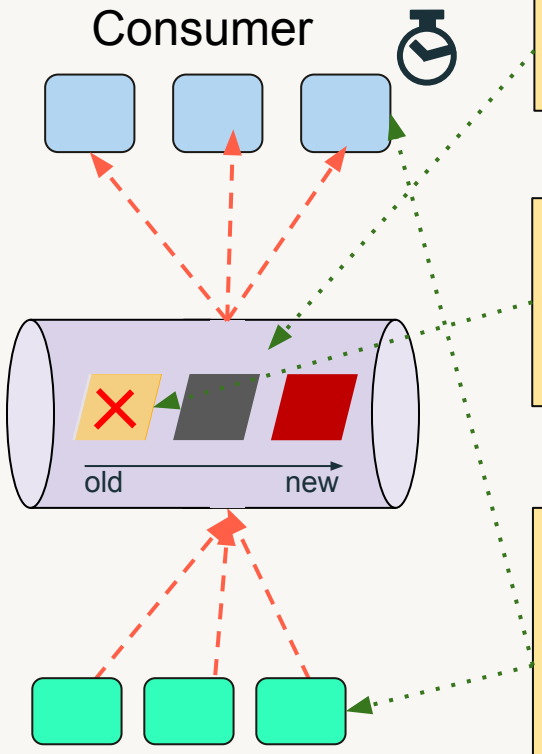
High latency due to backlogs



- Large backlogs due to slow consumers, network, misconfiguration, …
  - Cause high latency/head-of-line blocking

Consumer contract prevents consumer from skipping, e.g., to latest version

Lack of correctness & decoupling



Hidden log cannot be retained forever

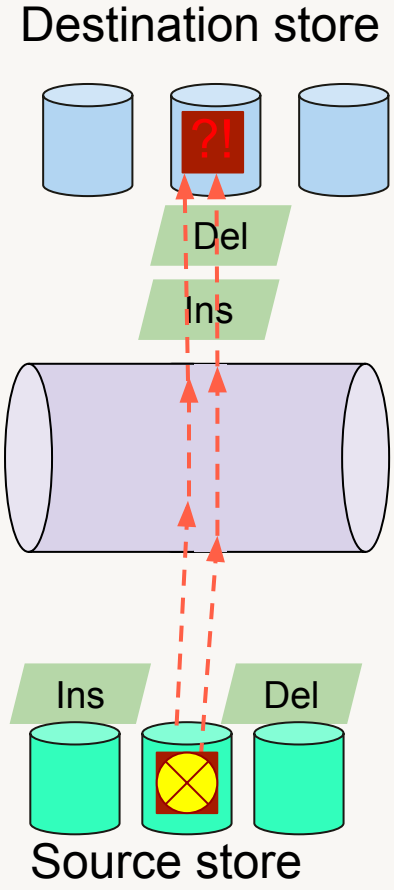
Pubsub retention policy of few days

Expectation: consumers not “that slow”

Reality: consumers *are* that slow

- Can lead to data loss or corruption
- Not decoupled: consumers must recover from source store

Lack of correctness, e.g. for replication

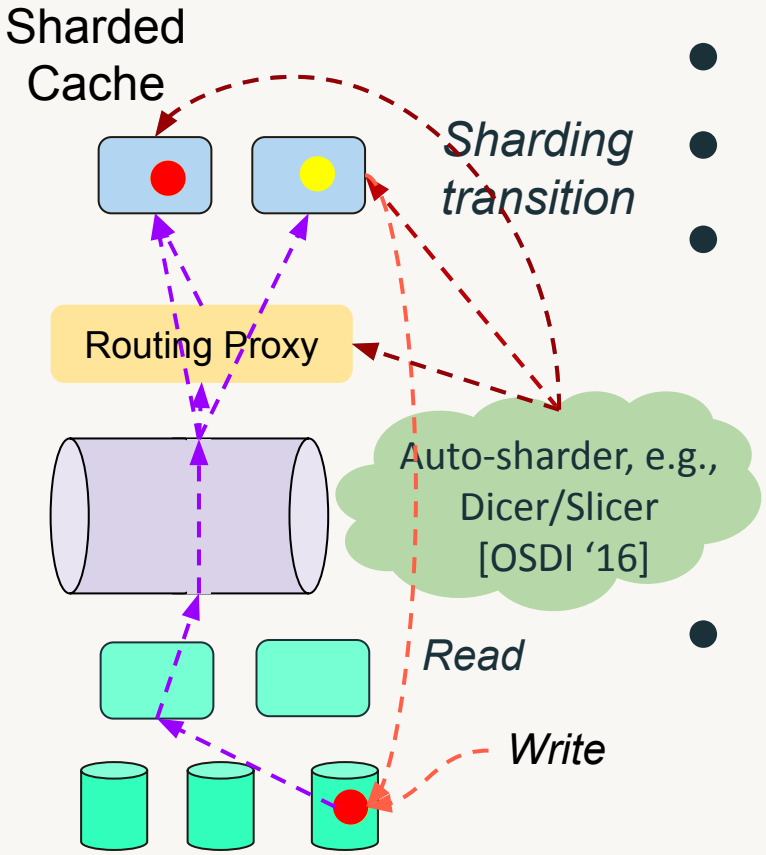


- **Goal:** Snapshot consistency on destination
- Serial publish and apply ⇒ correct, no scale
- Parallel ⇒ reordering, no eventual consistency!

Pubsub ordering guarantee not helpful

- *Get*: Pubsub delivers messages in publish order
- *Need*: Consumer end needs transaction order
- In practice, settle for eventual consistency
  - Reset with daily/weekly snapshots
  - For “old” snapshot & “approximate” queries

Lack of correctness, e.g. for cache freshness



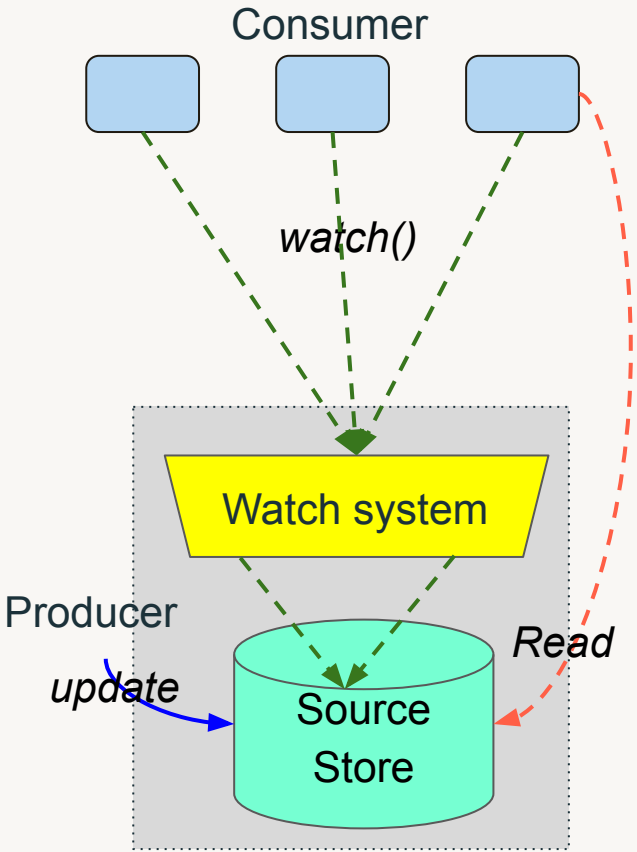
- Messages to all pods: doesn't scale
- Pubsub system's sharding not helpful
- Auto-sharding creates a moving target for updates: Races lead to lost updates

- Cache freshness in practice
  - Backup TTLs/polling to handle unreliability

Talk Outline

- What is pubsub
- Problems with pubsub
- Our approach: Watch with explicit store
- Research directions

Solution: Explicit storage with Watch



- Producers write to source store
- Consumers can access source store
- Watch system: External or built-in
- Consumers watch ranges of interest
- Watch model present in Spanner, k8s

Watch API

```
trait Watchable {
  void watch(lowKey, highKey,
             version, callback);
}

trait WatchCallback {
  void onChange(key, version, change);

  void onResync();

  void onProgress(lowKey, highKey,
                  version);
}
```

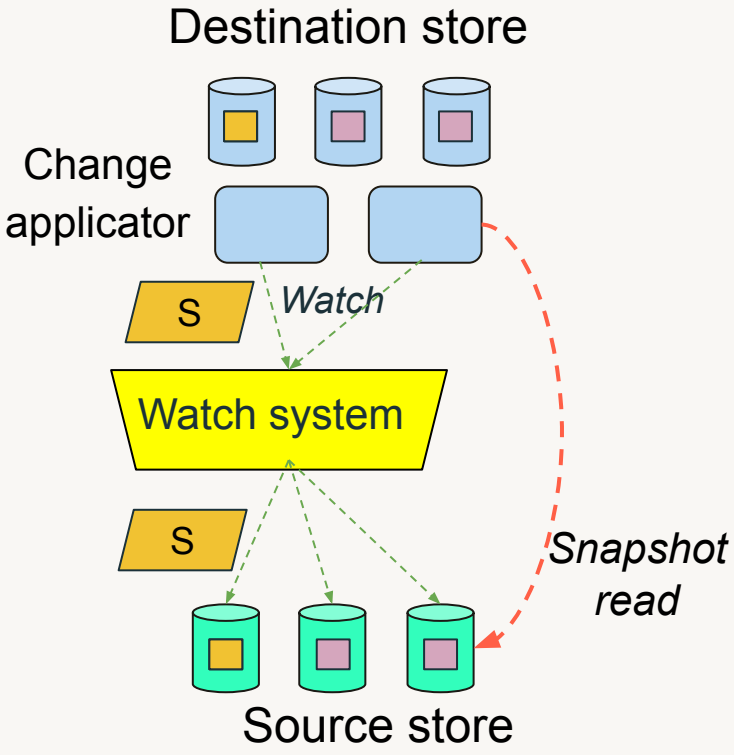
Watcher expresses interest for dynamically assigned key range starting at last known version

Versions are in source store domain

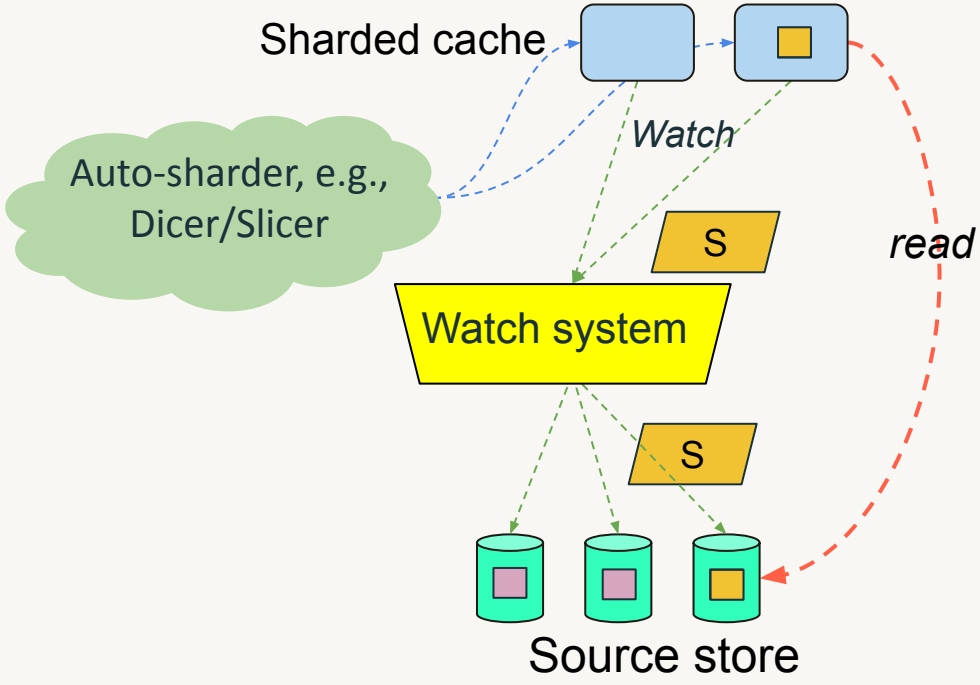
Given excessive backlog, recover from source store

Range-scoped progress signals enable strong e2e semantics (e.g. snapshot consistency)

Architectures based on Watch

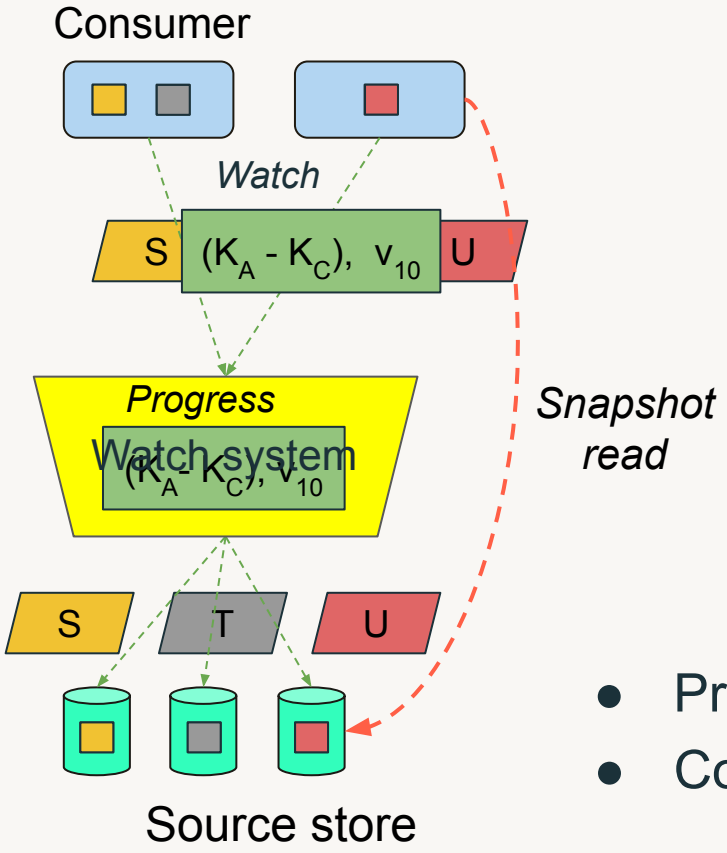


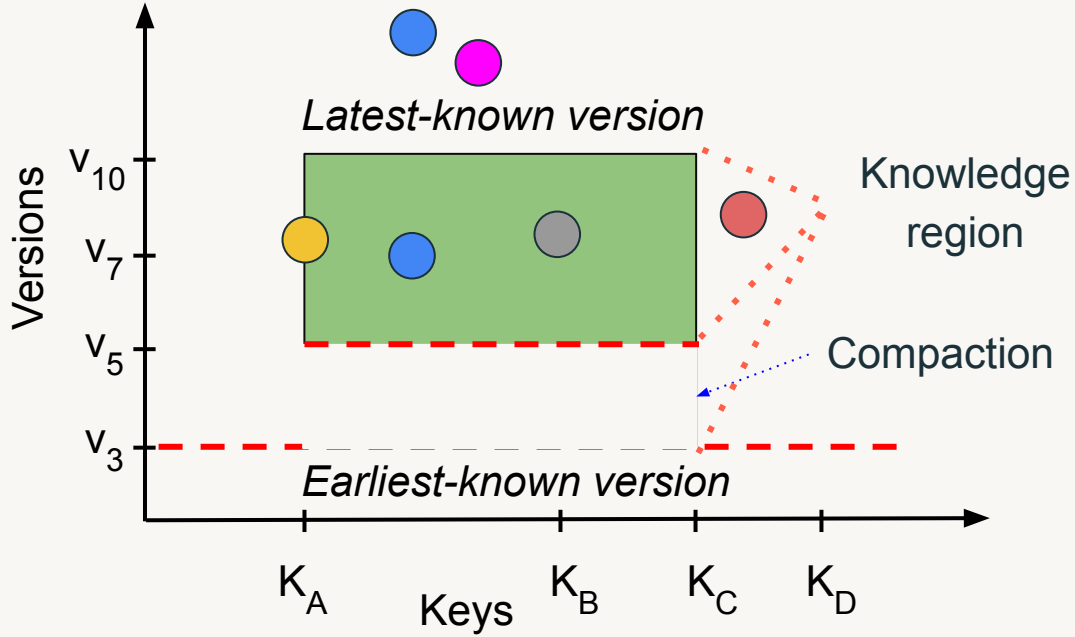
Replication across databases



Cache freshness

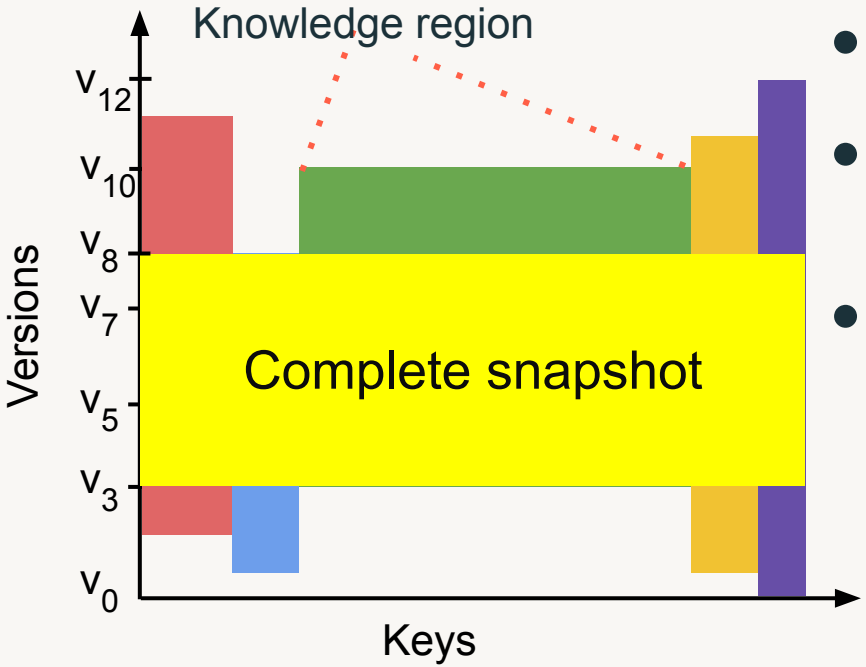
Range-based knowledge using Watch





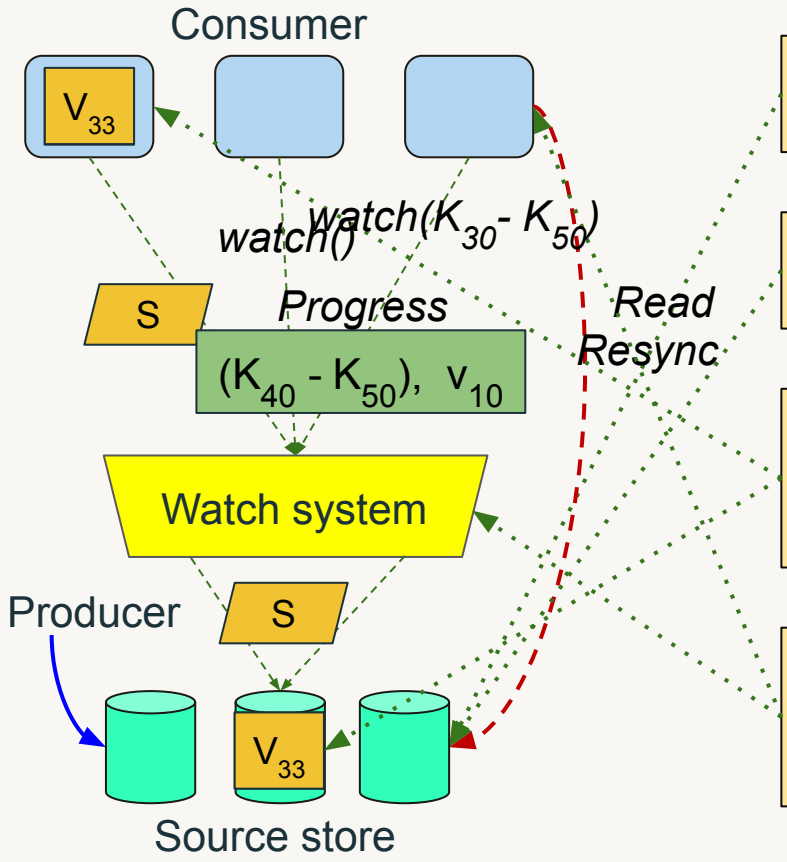
- Progress signals enable range-based guarantees
- Consumer can safely GC ranges to save space

Snapshots using knowledge regions



- Get “jagged” windows from watch system
- Knowledge region upper bound increases on more progress signals
- Serve snapshot across contiguous regions for range of keys

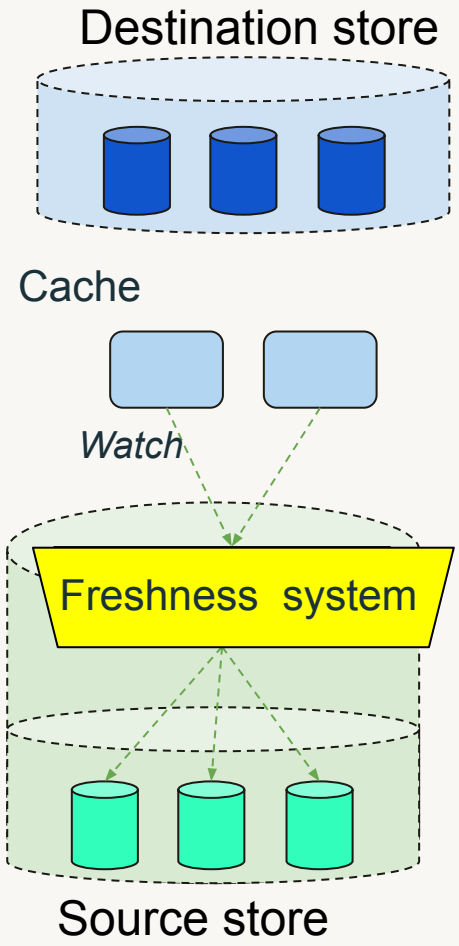
Contrasting with pubsub's weaknesses



- Explicit not hidden store
- Durability: leverage source store
- End-to-end ordering instead of pubsub ordering
- Dynamic partitioning: convey arbitrary ranges

Talk Outline

- What is pubsub
- Problems with pubsub
- Our approach: Watch with explicit store
- Research directions



Research directions

- Scalable Watchable storage, e.g., Spanner requires Truetime, etcd does not scale
- External scalable watch system
- Scalable & snapshot-consistent replication for heterogeneous storage
- Scalable & correct cache freshness system using Watch

Summary

- Pubsub abstraction violates end-to-end principle: Fails to address application needs
- Explicit storage w/ Watch enables correct & scalable solutions
- New research directions in storage, caching, replication