Photon: fault-tolerant and scalable joining of continuous data streams

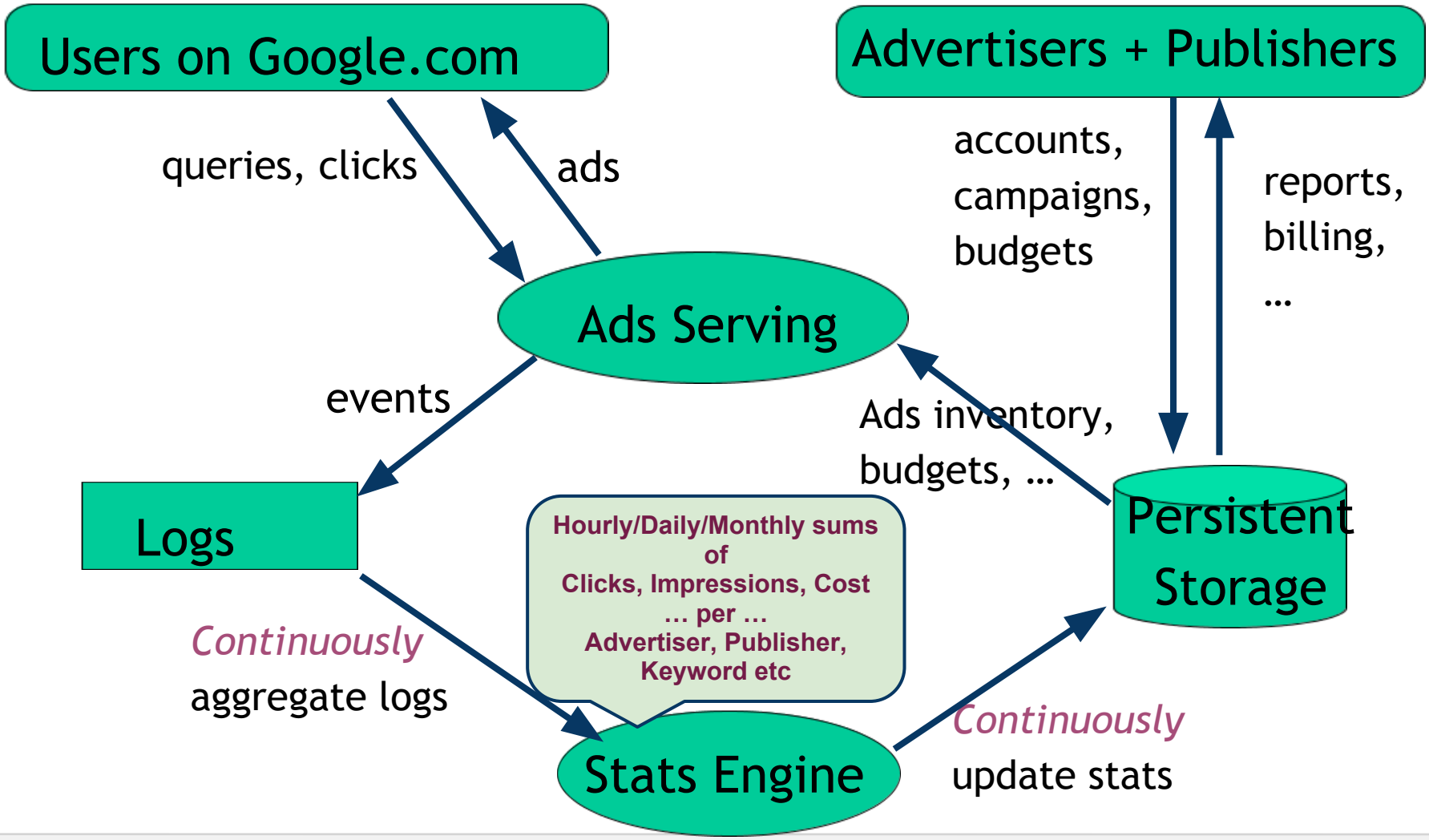
Manpreet (manpreet@google.com)

Google

Agenda

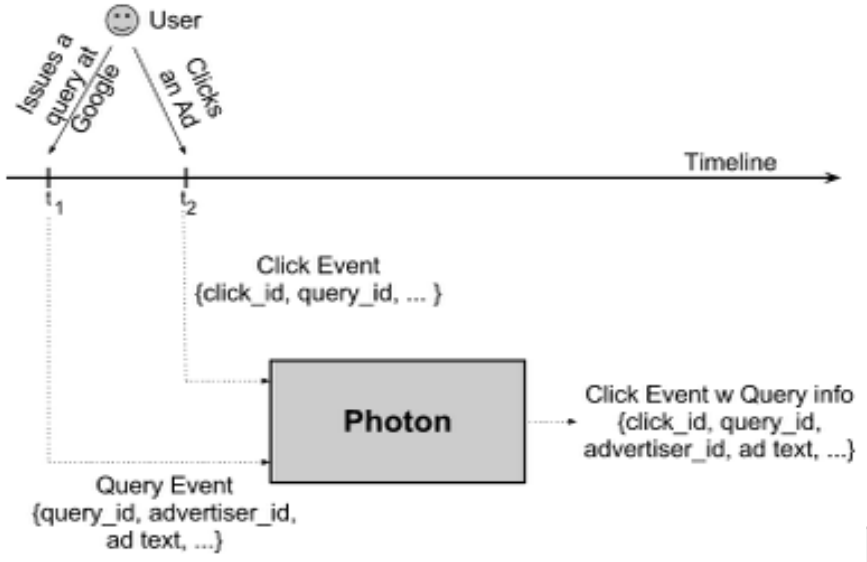
- Problem and motivation
- Systems challenges
- Design
- Production deployment
- Future work

The Stats Loop



Problem

- Join two continuous streams of events
  - Based on a shared id
  - Stored in the Google File System
  - Replicated in multiple datacenters
- Motivation:
  - Join click with query
  - Logged by multiple servers at different times
    - Cannot put query inside the click



Systems challenges

- Exactly-once semantics
  - Output used for billing / internal monitoring
- Reliability
  - Fault-tolerance in the cloud
  - Automatically handle single data-center disaster
- Scalability
  - Millions of joins per minute
- Latency
  - O(seconds)

Singly-homed system

- With patched-on failover tool
  - Ran in production for several years
- Very high maintenance cost
- Datacenters have downtimes:
  - planned
  - unplanned
  - random hiccups
- Cannot provide very high up-time SLA

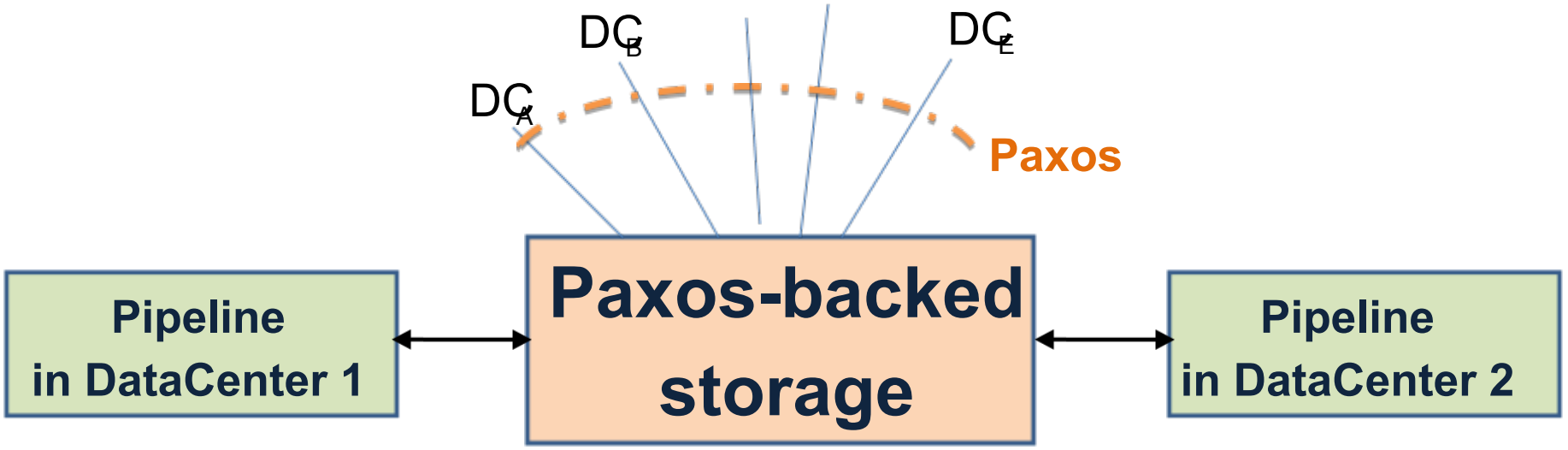
How about running a MapReduce?

- Read both query logs and click logs
- Mapper output key is query_id
- Reducer outputs the joined event
- Issues:
  - batch job
  - high latency (setup cost, stragglers, etc)

Challenges of running in multiple datacenters

- Stateful system
  - Need to ensure exactly-once semantics
- State needs to be replicated synchronously to multiple datacenters
  - ==> Paxos (guarantees majority group members have all the updates)

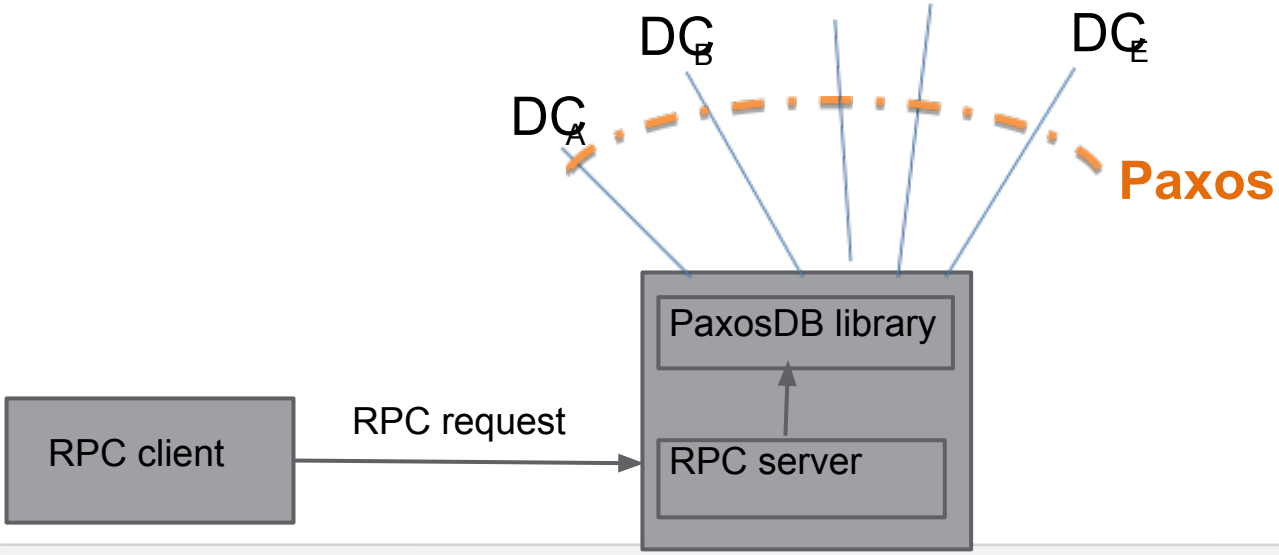
High-level idea



- Run the pipeline in multiple data-centers in parallel:
  - Each pipeline processes every event
- Paxos-backed storage is shared across pipelines
  - Used to dedup events (at-most-once semantics)
  - Maintain persistent state at event-level

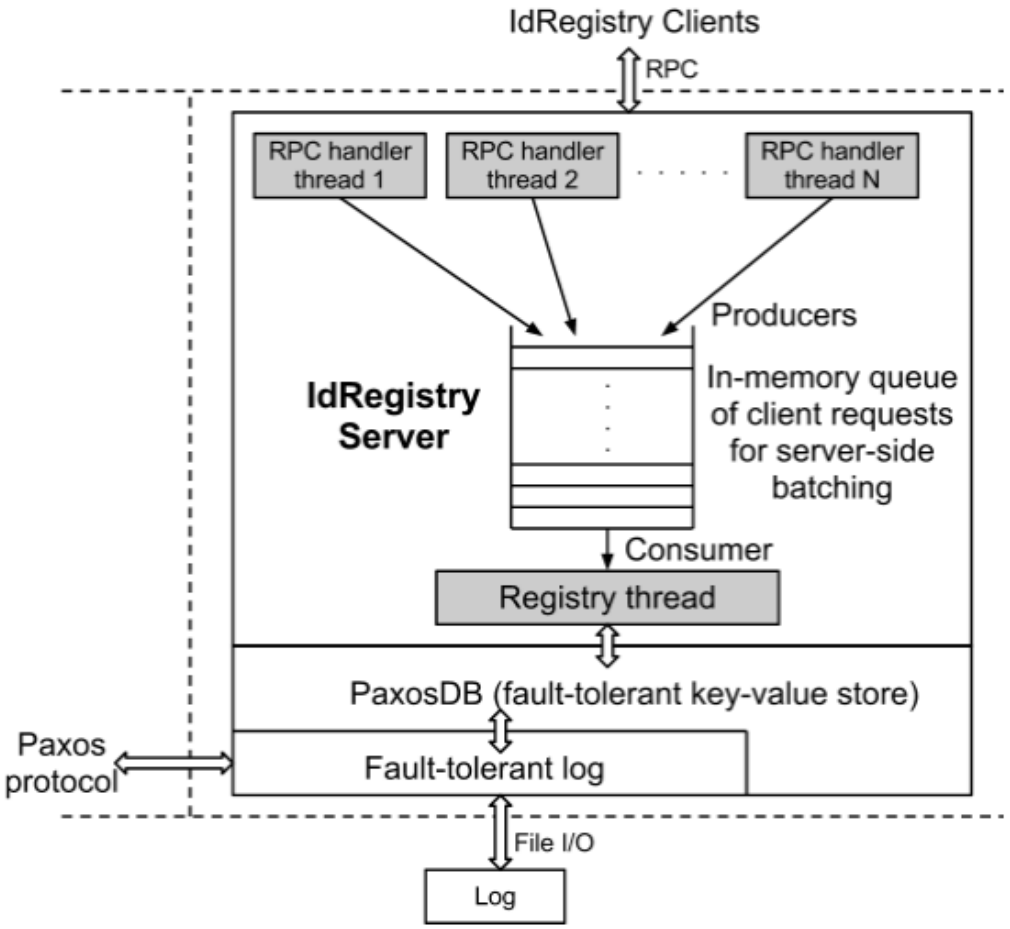
PaxosDB: key building block

- Key-value store built on top of raw Paxos
- Runs paxos algorithm to guarantee consistent replication
  - Multi-row transactions with conditional updates
  - Auto-elects new master
- Key challenge: scalability
  - Need to join Millions of events/minute
  - With cross-country replicas, less than 10 paxos transactions/sec



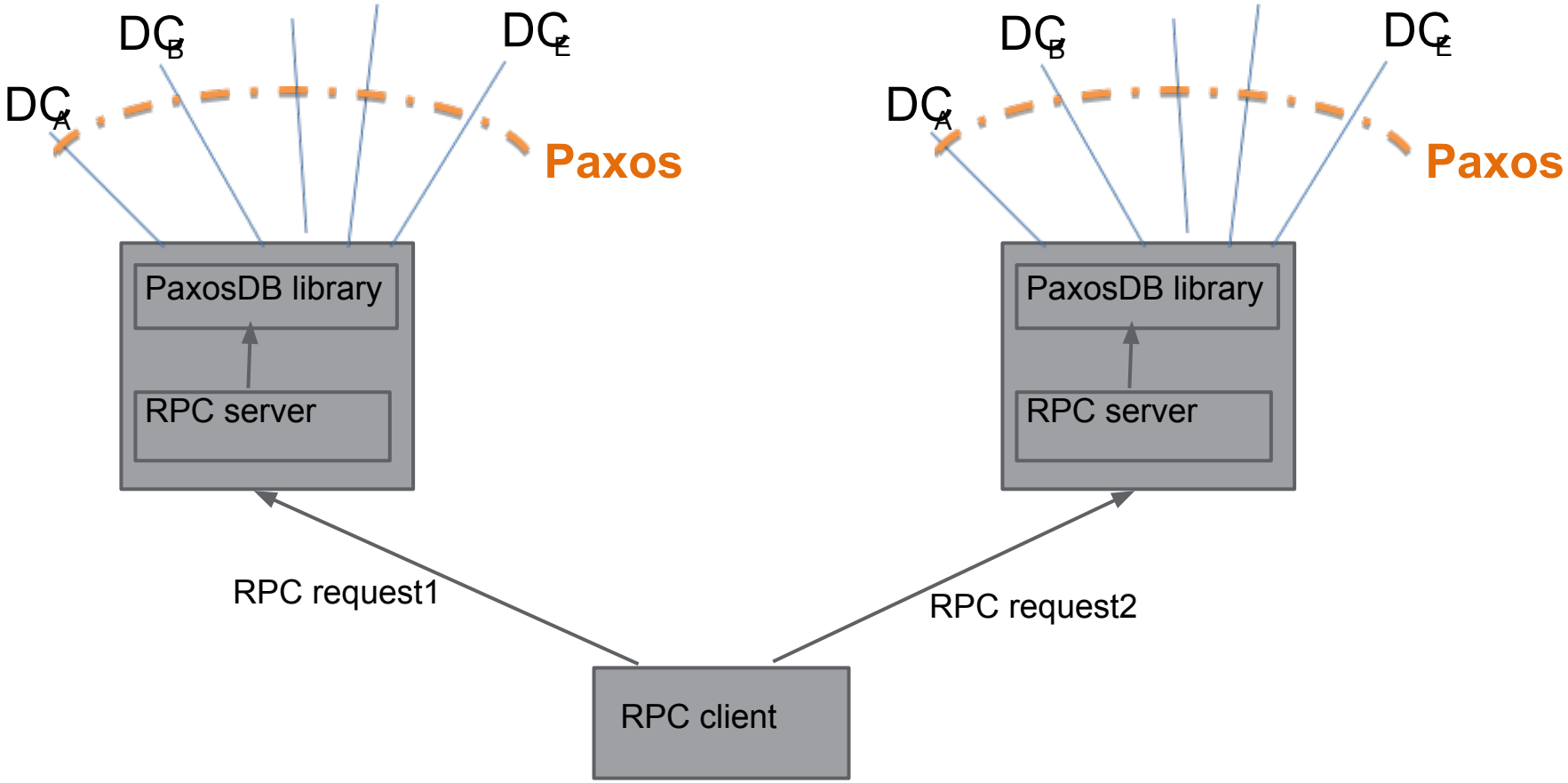
Google

Architecture of a single IdRegistry server



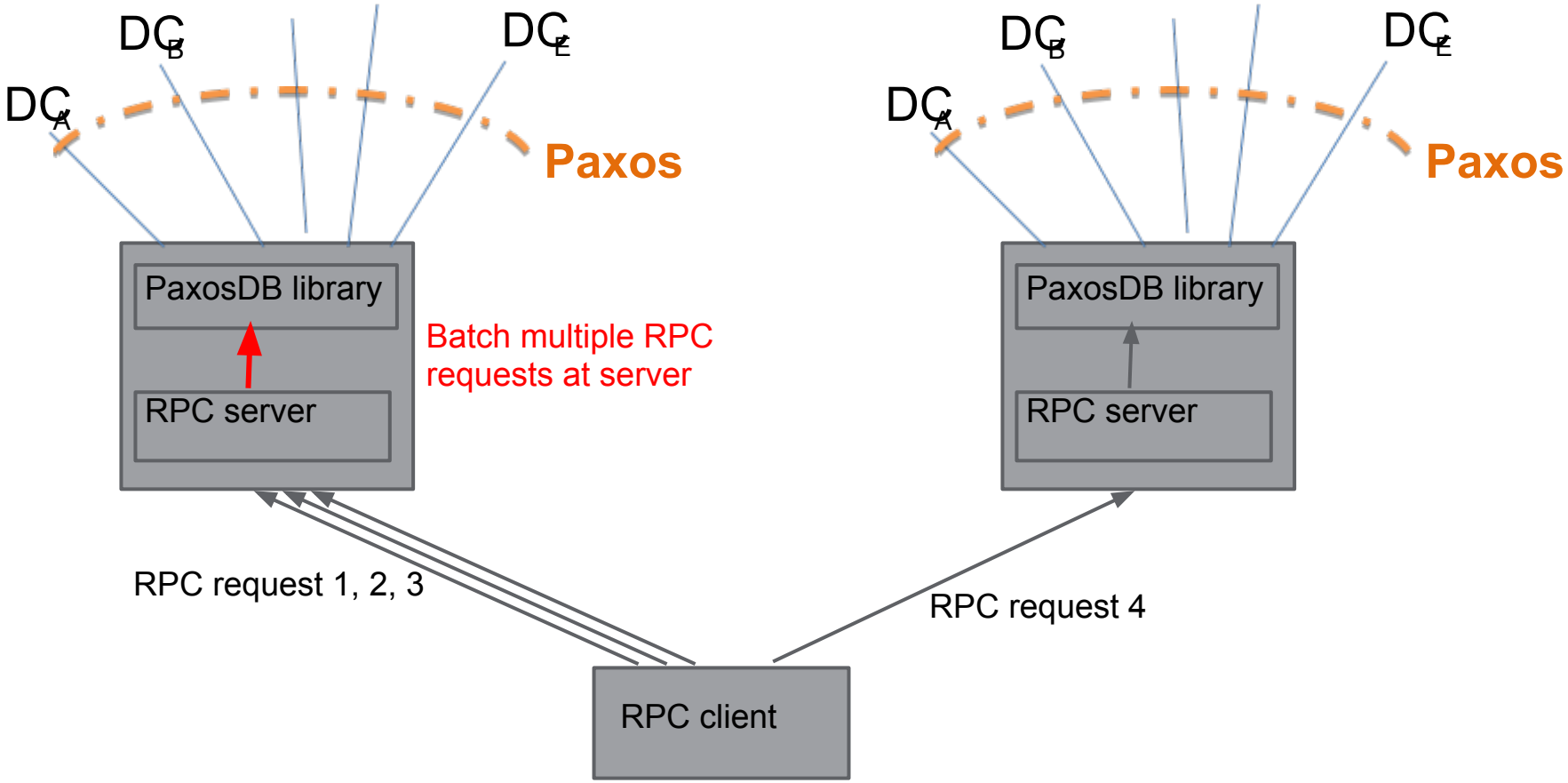
IdRegistry made scalable: **sharding**

- Run paxos transactions in parallel for independent keys



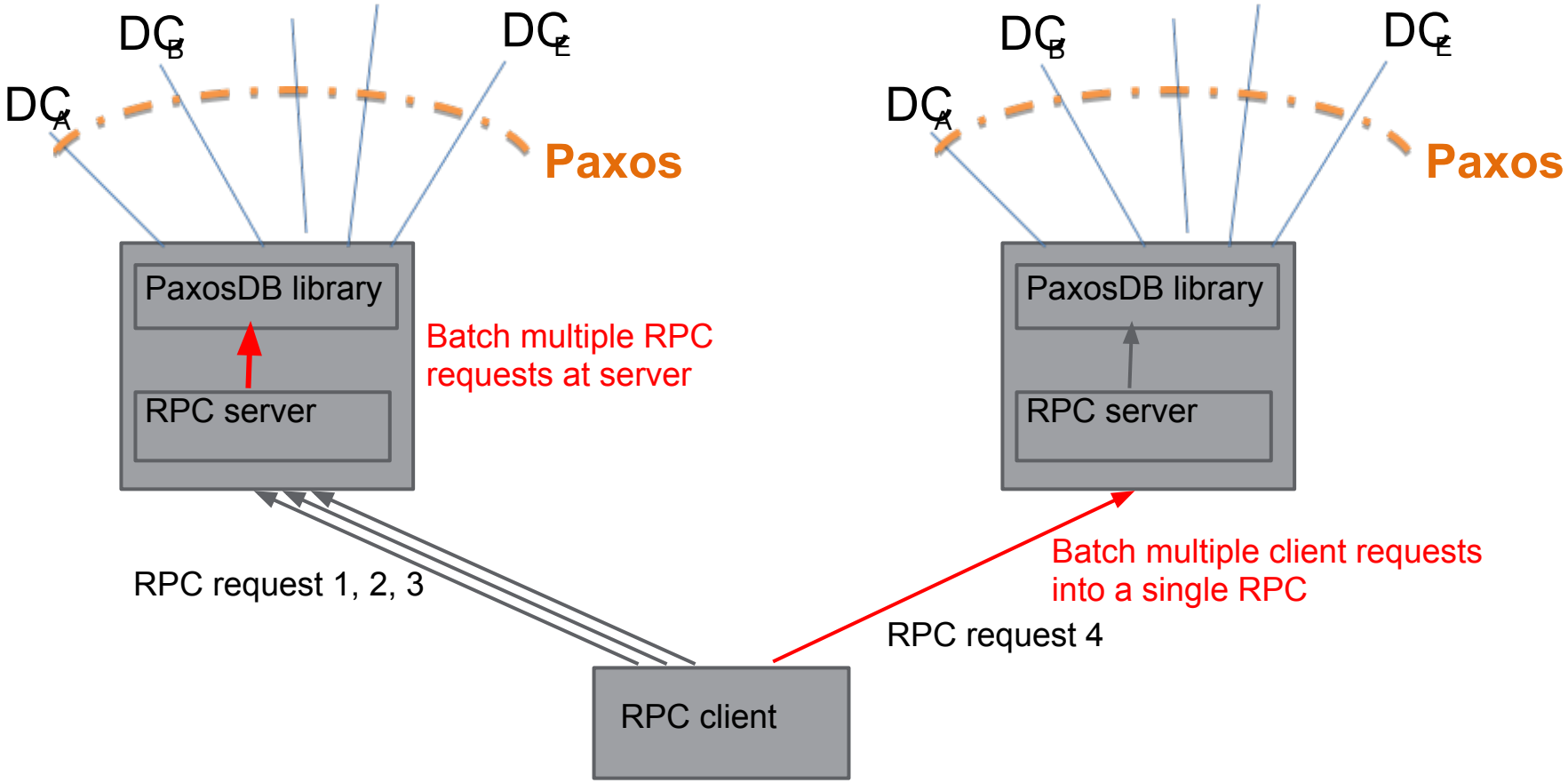
IdRegistry made scalable: **server-side batching**

- Batch multiple RPC requests into a single Paxos transaction



IdRegistry made scalable: **client-side batching**

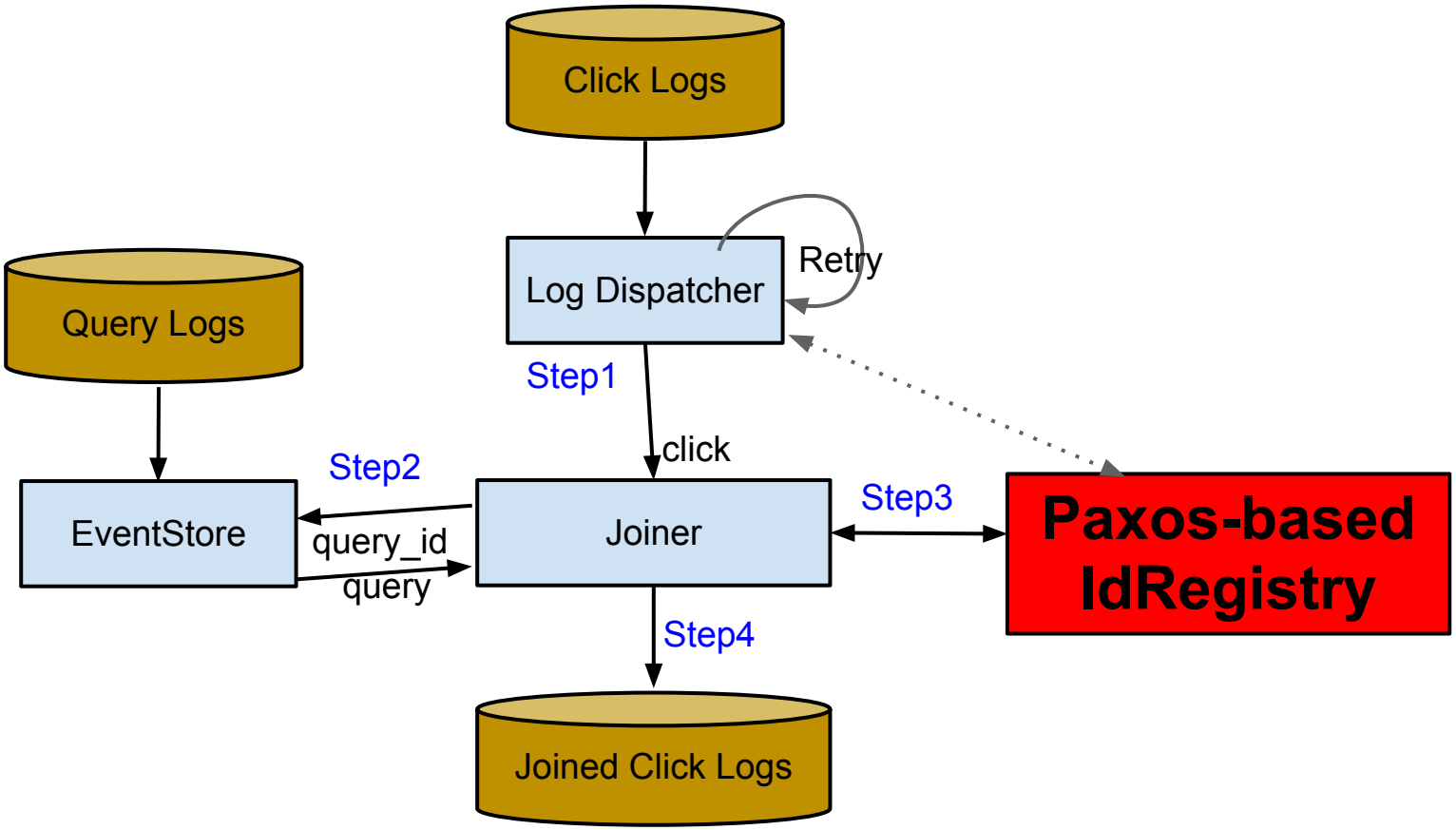
- Batch multiple client requests into a single RPC



IdRegistry made scalable

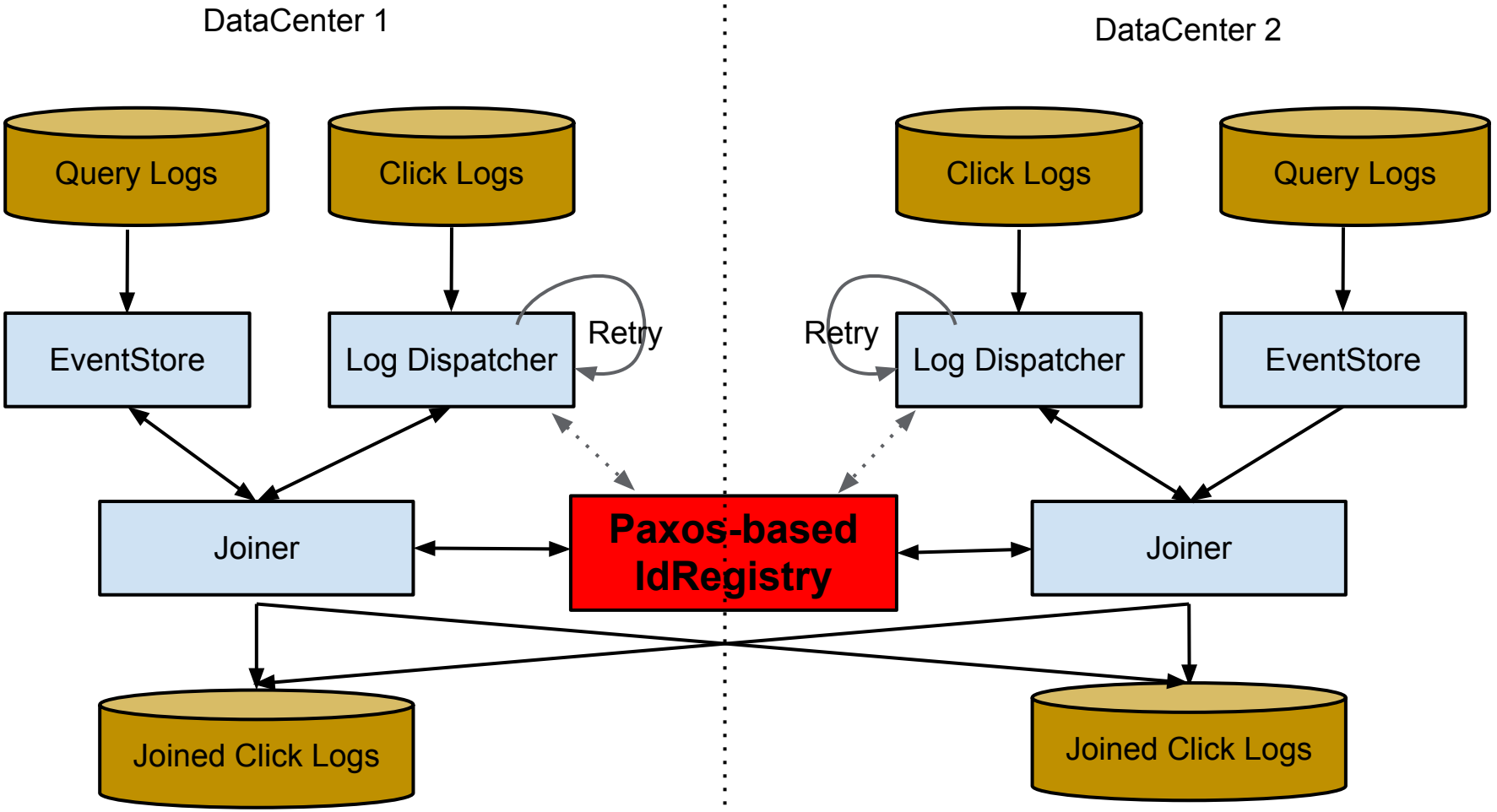
- Sharding
  - Paxos transactions happen in parallel for each shard
  - Load-balance amongst shards:
    - Shard by hash(event_id) mod num_shards
  - Built automated support for resharding
    - Attach timestamp to each key
    - Use timestamp-based sharding
    - GC old keys
- Server-side batching
  - RPC thread adds input request to an in-memory queue
  - Single paxos thread extracts multiple requests, performs a single multi-row paxos transaction, sends rpc response
- Client-side batching
  - Client batches multiple RPC requests into a single RPC request.

Photon architecture in a single data-center



Google

Photon architecture in multiple datacenters



PaxosDB rpc semantics

- InsertKey:
  - Returns false if the key already exists
  - Else inserts the key
- What if the PaxosDB inserts the key but Joiner does not get the response in time?
  - Observed 0.01% loss in production
- Write unique id for joiner along with the event_id:
  - event_id is key
  - Joiner id is value
  - Handles Joiner rpc retries gracefully
- If Joiner crashes after writing to Paxos, run offline recovery

Reading input events continuously

- Events stored in Google File System
- Periodically stat the directory:
  - identify new files
  - check the growth update on existing files
- Keep track of <filename, next_read_offset>

EventStore

- Sequentially read the query logs and store a mapping from query_id to query
  - In a distributed hash table (e.g. bigtable)
- CacheEventStore:
  - Majority of clicks happen within few minutes of query
  - Cache mapping from query_id ---> query for recent queries in RAM
  - Use distributed Cache Servers (similar to MemCache)
- LogsEventStore
  - Performance optimization since our query logs are sorted
  - Fallback in case of cache miss
  - Use binary search in sorted event stream in disks

Purgatory (in Log Dispatcher)

- What if a subset of query logs are delayed?
- Log Dispatcher reads every event from the log, keeps retrying until successful join
  - Maintains state in peristent storage (Google File System)
  - Ensures at-least-once semantics
- Exponential backoff in case of failure

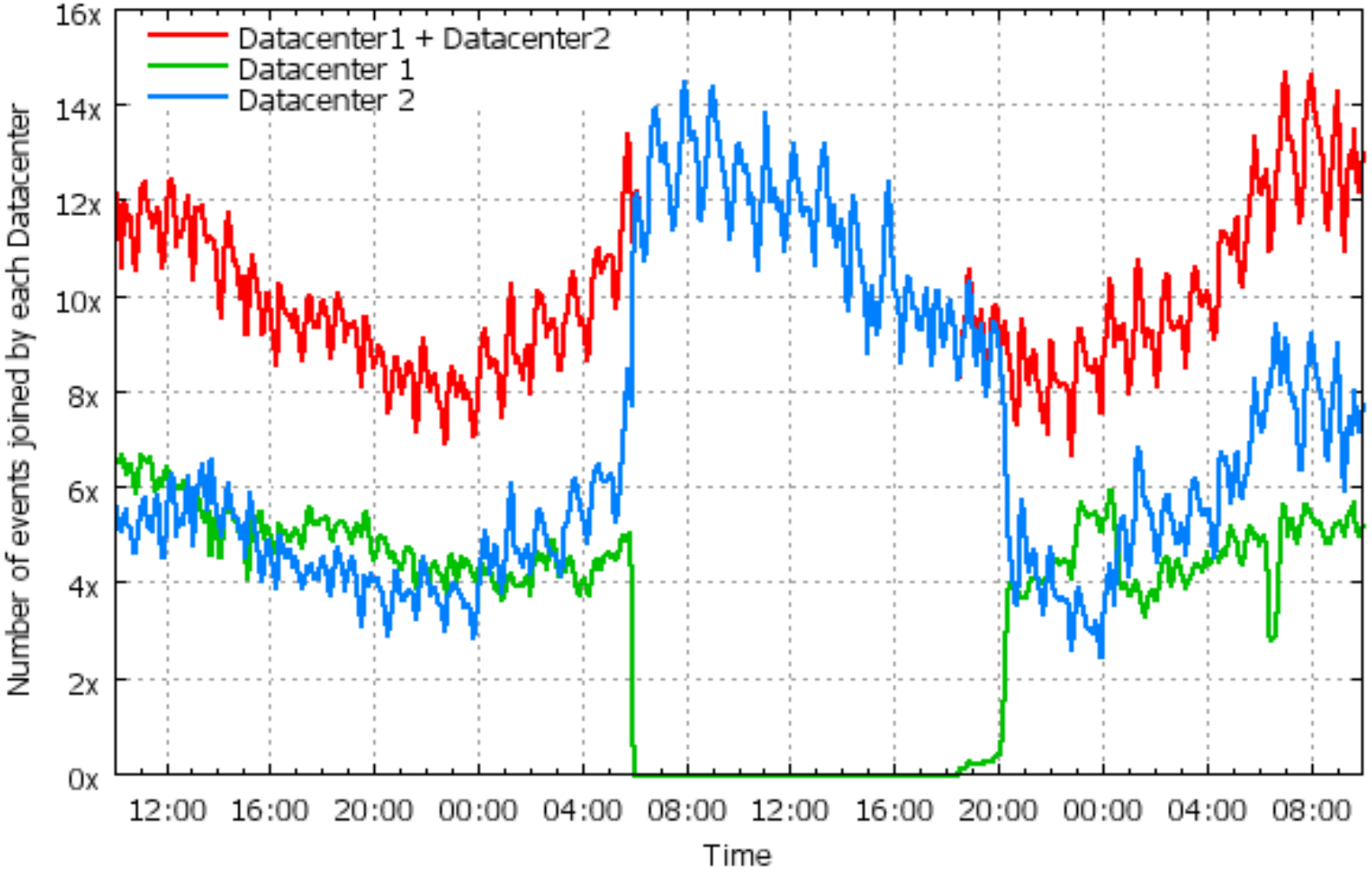
Operational challenges

- Need 24x7 running system
  - Low maintenance
- Volume of input traffic fluctuates by 2x within a day
- Bursts of traffic in case of network issues
- Predicting resource requirements
- Auto-resize the jobs to handle traffic growth and spike
- Reliable monitoring

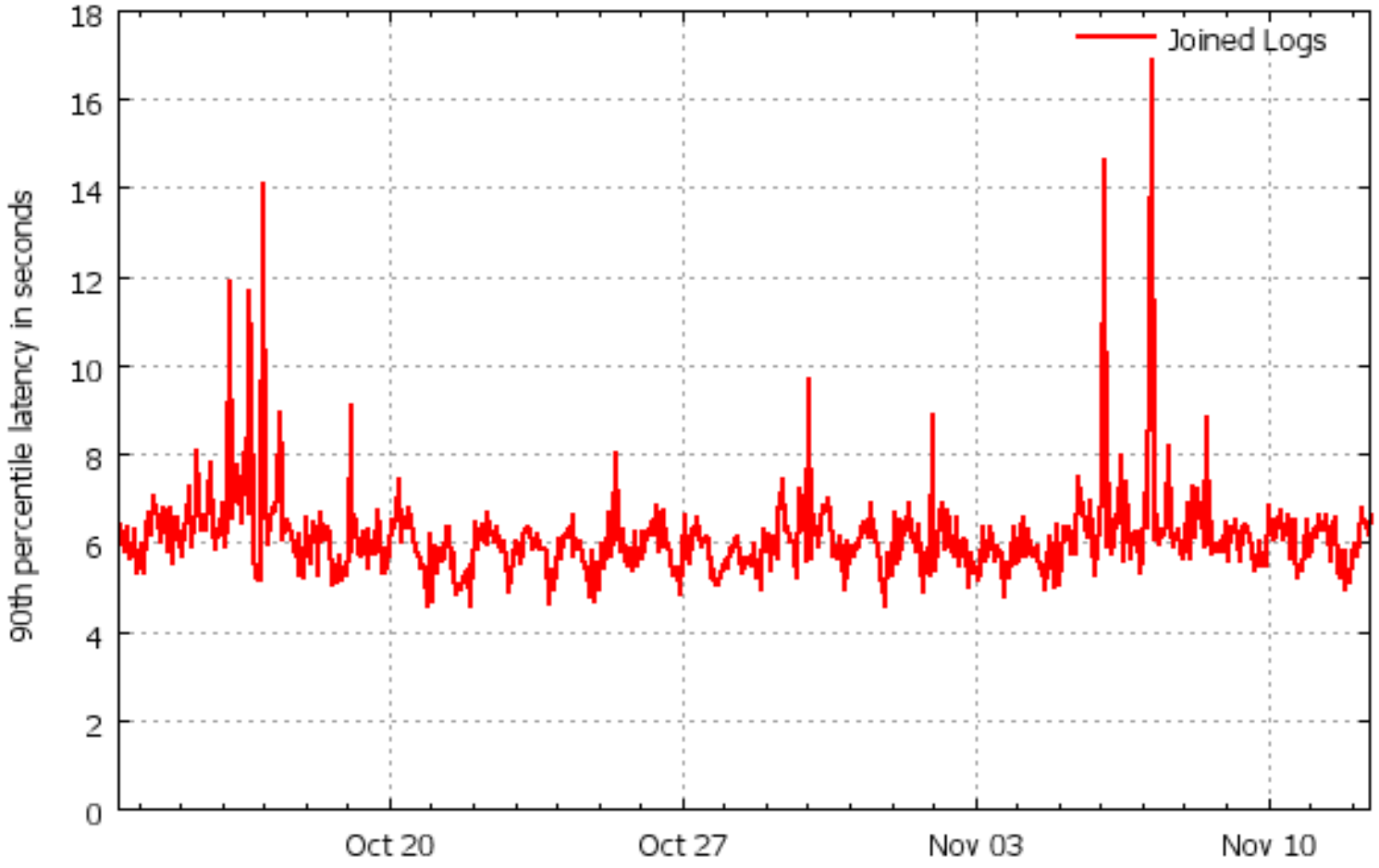
Production deployment

- System running in production for several months
  - O(700)-way sharded PaxosDB
  - 5 PaxosDB replicas spread across East Coast, West Coast, Mid-West
  - 2 Photon pipelines running in East Coast, West Coast
  - Order of magnitude higher uptime SLA than singly-homed system
  - Survived multiple planned / unplanned datacenter downtimes
  - Much less noisy alerts

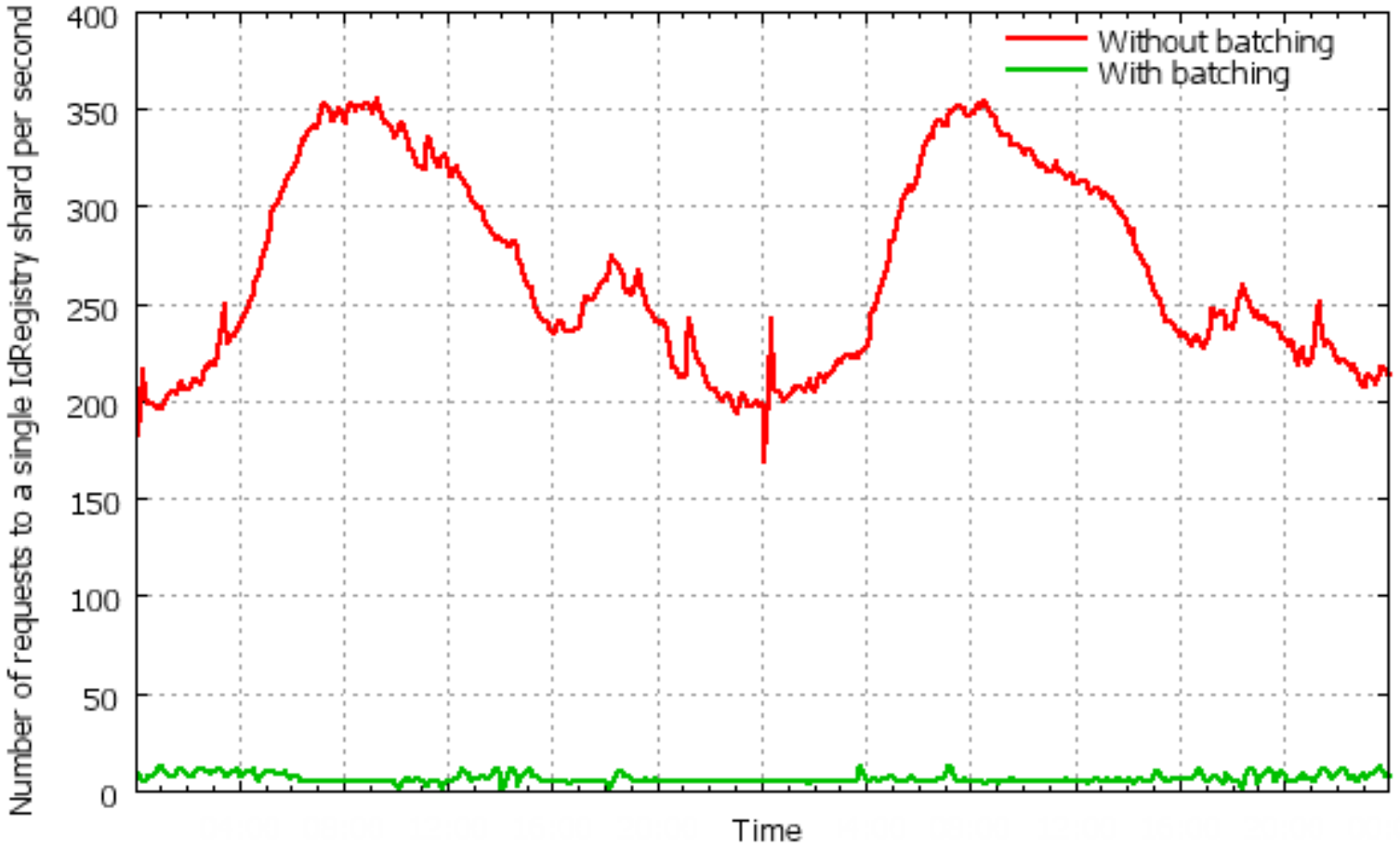
Photon withstanding a real datacenter disaster



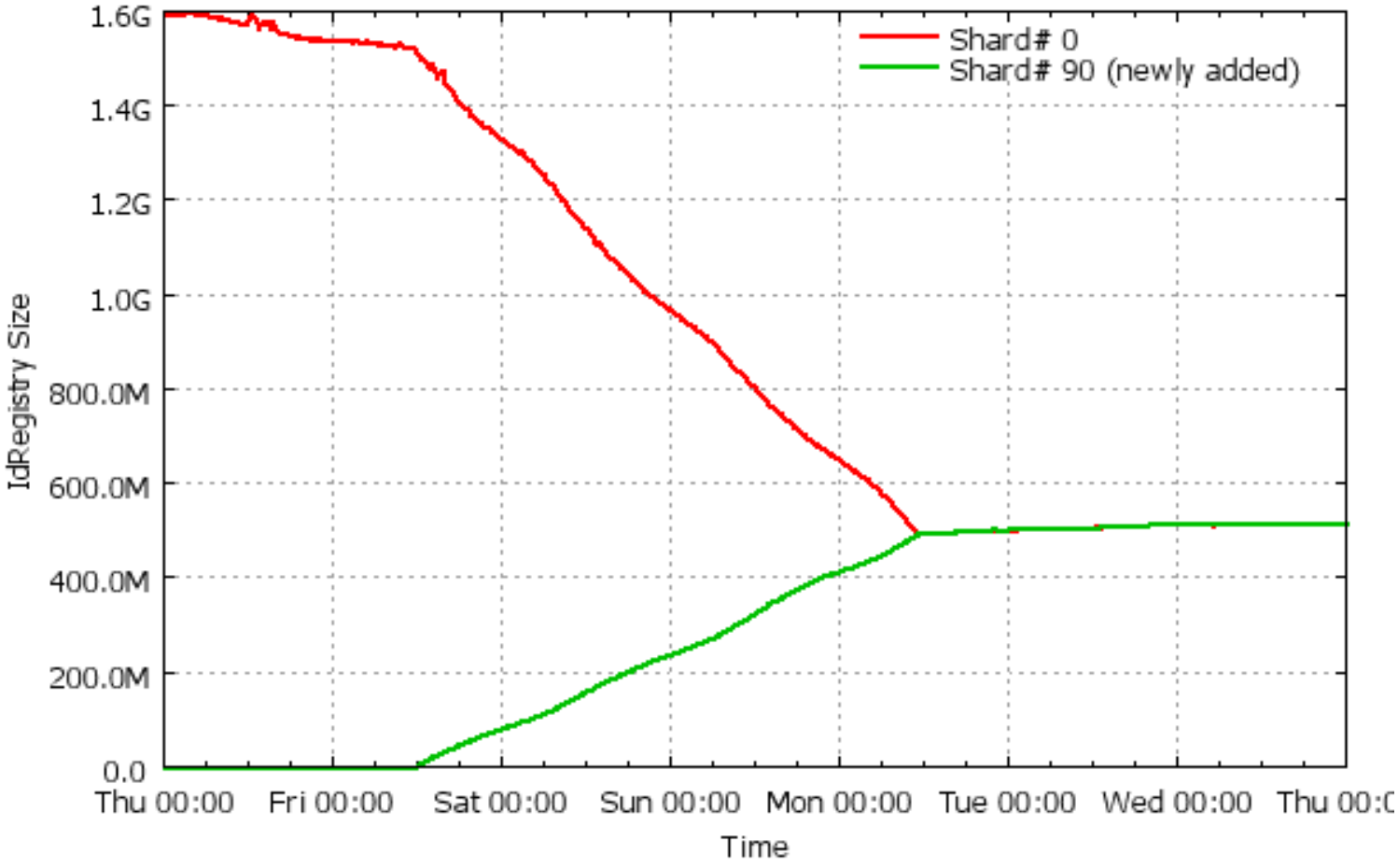
End-to-end latency of Photon



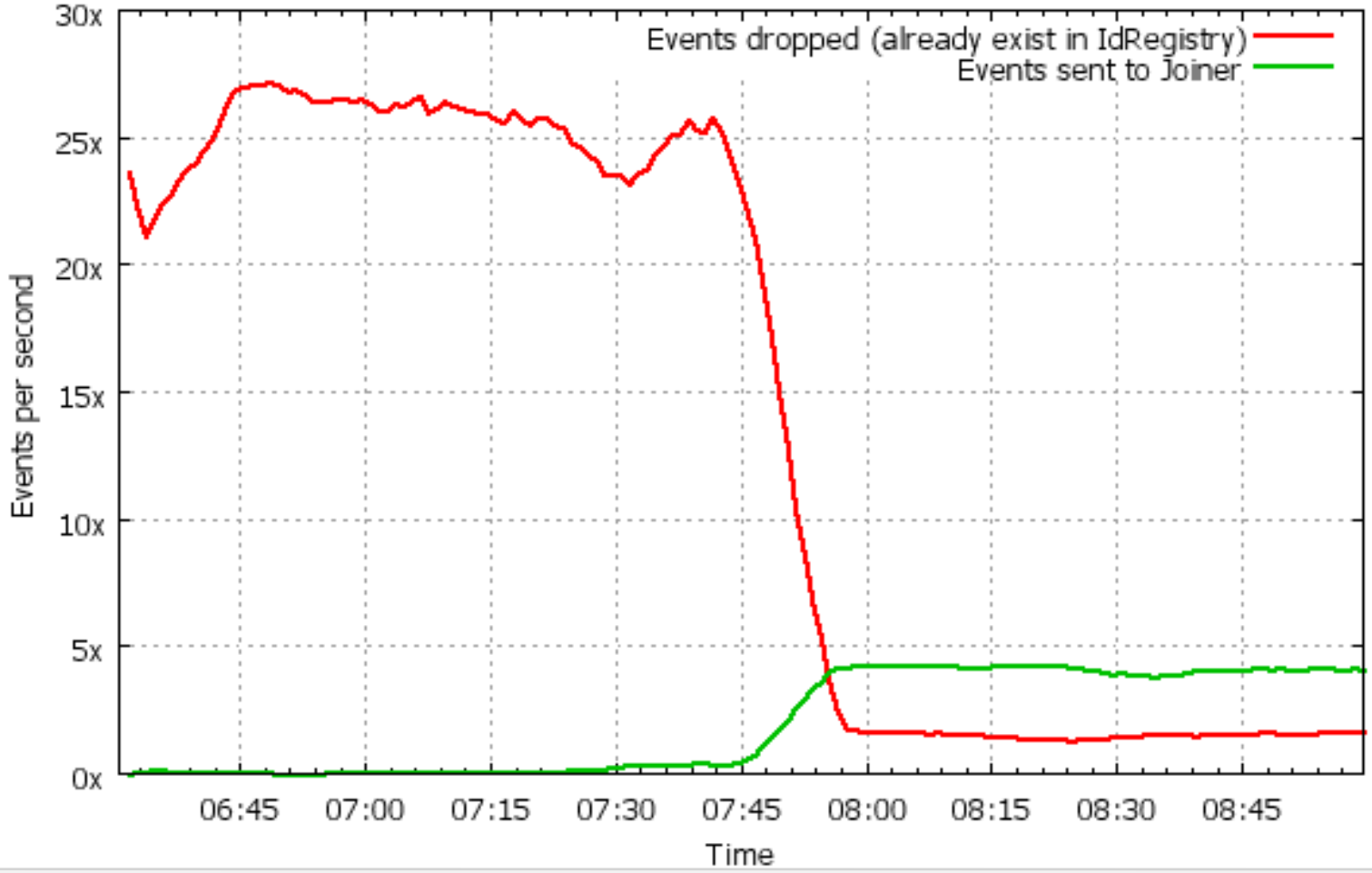
Effectiveness of server-side batching in IdRegistry



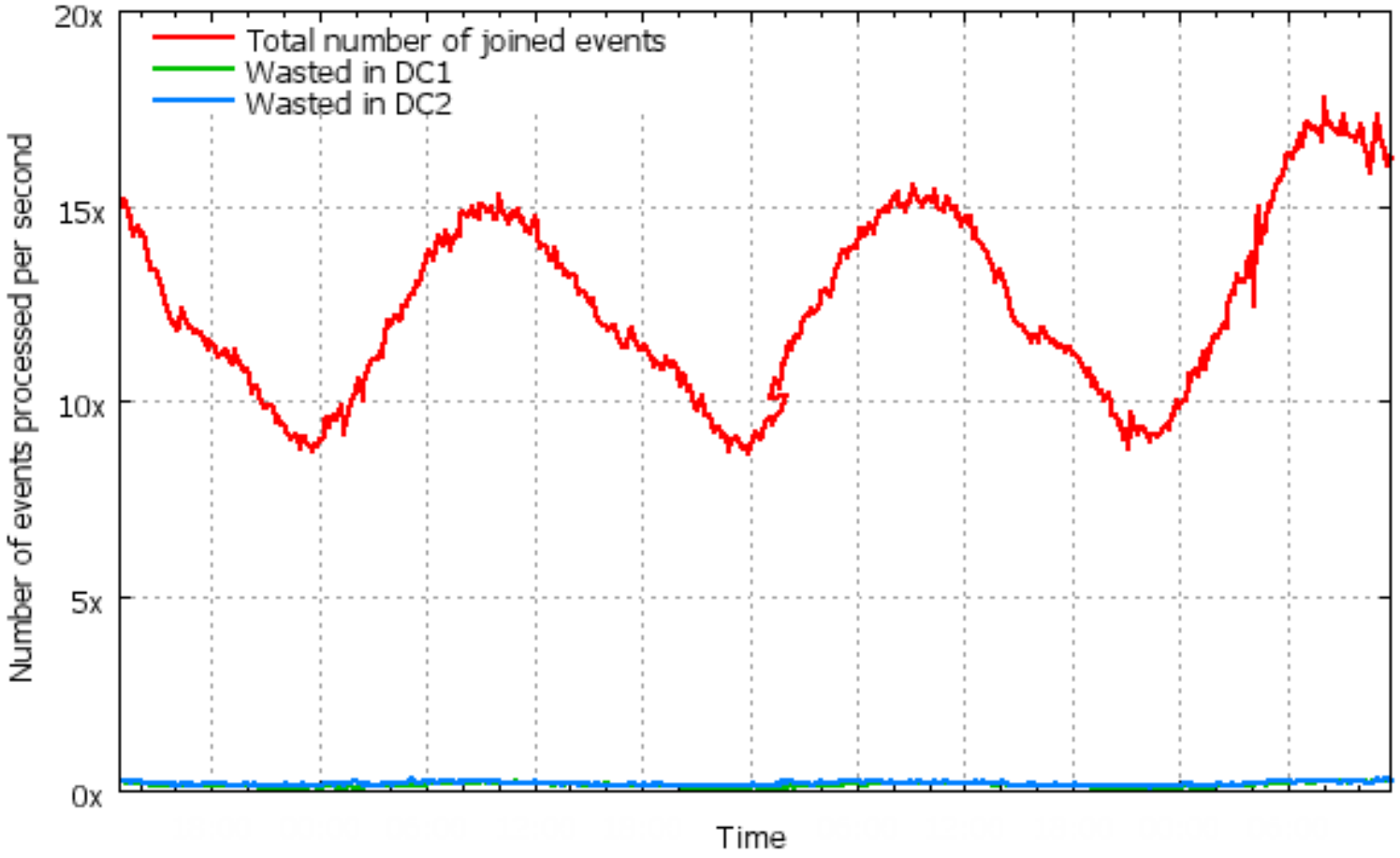
IdRegistry dynamic time-based upsharding



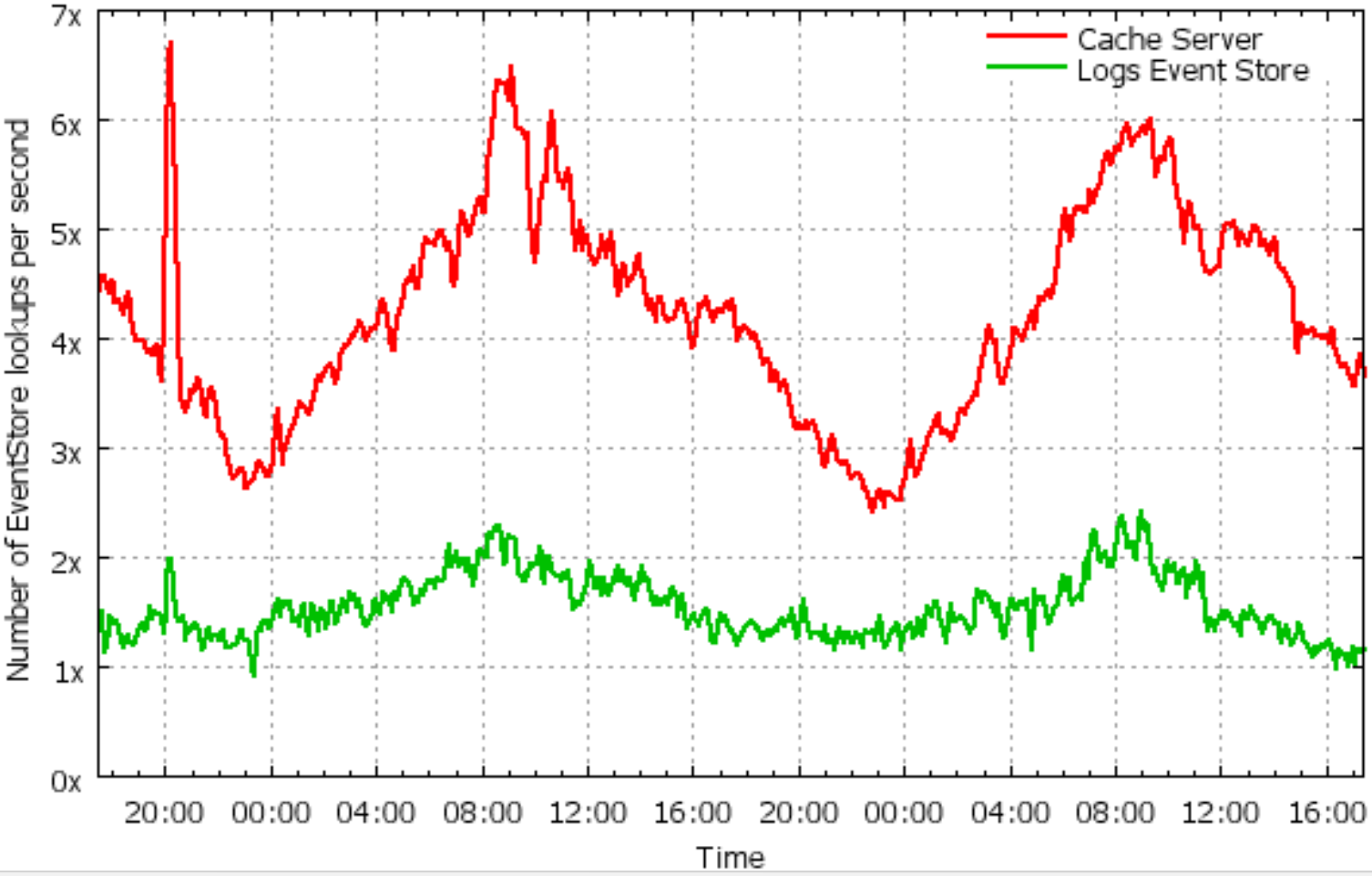
Dispatcher in one datacenter catching up after downtime



Photon wasted joins minimized



Photon EventStore lookups in a single datacenter



How we addressed the systems challenges

- Exactly-once semantics
  - PaxosDB ensures at-most-once
  - Dispatcher retry ensures at-least-once
- Reliability
  - PaxosDB needs only majority members to be up.
  - Storing global state in PaxosDB allows run in multiple datacenters
- Scalability
  - Reshard the number of PaxosDB servers
  - All the other workers are stateless
- Latency
  - Mostly RPC-based communication amongst jobs
  - Most data transfers in RAM

Google

Next BIG challenges

- Better resource utilization
- Join multiple log sources

Google

More cool problems we are solving in AdWords Backend

- Real-time logs processing
  - Read 100T logs per day
  - Compute stats over 200+ dimensions with low latency
- Petabyte scale database storage engine
  - 32M rows updated per sec, 140T total rows
  - 200K read queries/sec with latency of 90ms
- Efficiently backfill stats for the last N years
  - O(50B) rows per day
  - Too big to store in a database
- And many more cool projects...
  - Join us and find out more!
  - Manpreet Singh (manpreet@google.com)

Google Application Process

1 Apply Now!

Full-time: **google.com/students/eng**

Internships: **google.com/students/intern**

2 Resume Review and Qualification

3 First Round Interviews - 2 Technical Phone Screens

4 **Full-time Positions**: 3-5 Onsite Interviews
Internships: 1 Host Matching Interview

5 Hiring Committee - **Offer**

Google

Questions