Scaling Out Without Partitioning

A Novel Transactional Record Manager for Shared Raw Flash

Phil Bernstein & Colin Reid
Microsoft Corporation

HPTS 2009
October 26, 2009

What is Hyder?

<u>It's an incubation, i.e. research project.</u>

A software stack for transactional record management

- Stores [key, value] pairs, which are accessed within transactions
- It's a standard interface that underlies all database systems

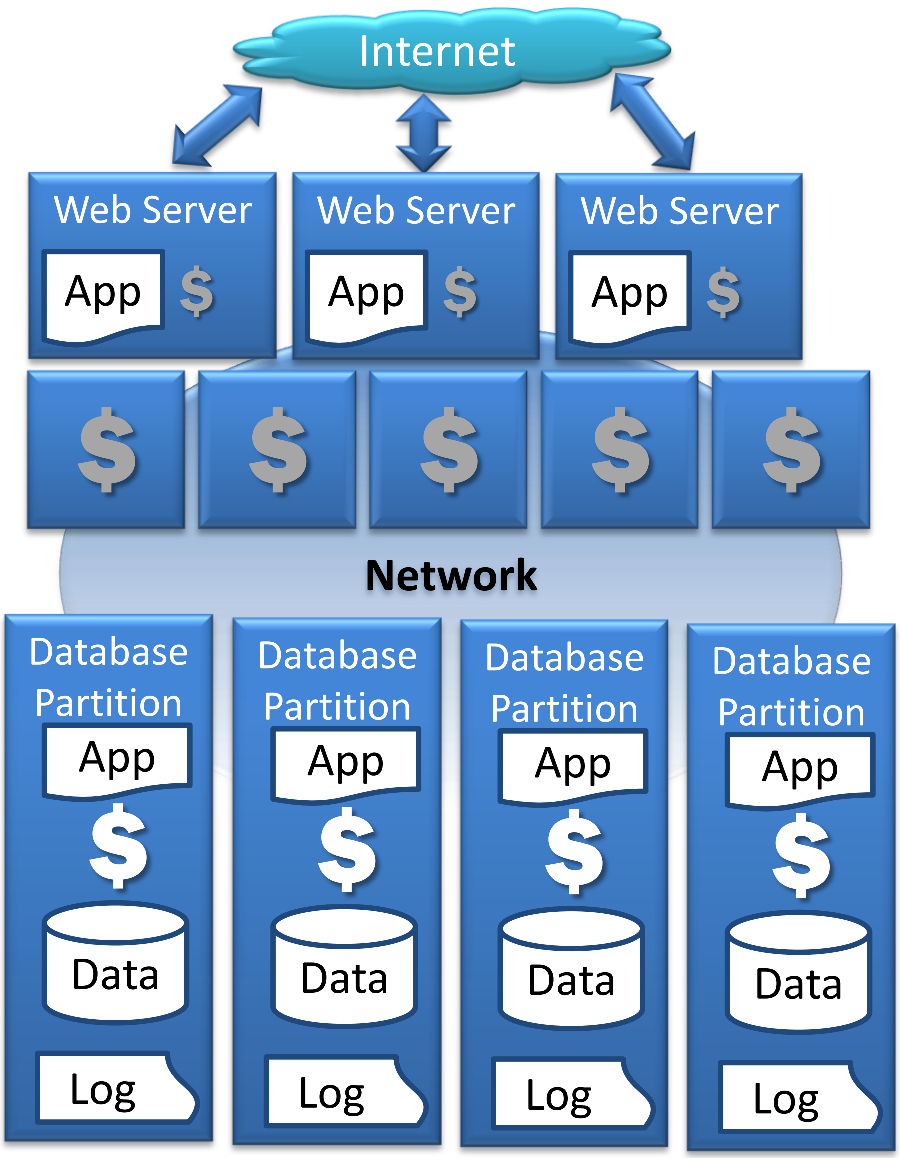
Functionality

- Records: Stored [key, value] pairs
- Record operations: Insert, Delete, Update, Get record where field = X; Get next
- Transactions: Start, Commit, Abort

Why build another one?

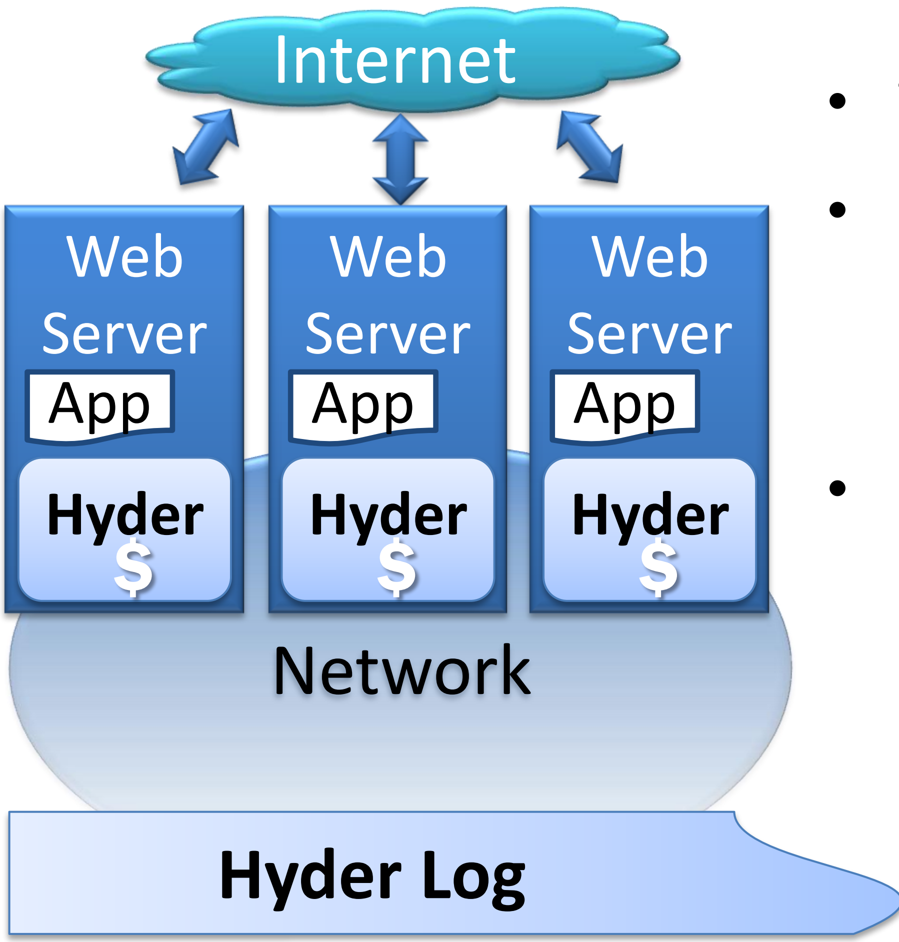
- Make it easier to scale out for large-scale web services
- Exploit technology trends: flash memory, high-speed networks

Scaling Out with Partitioning



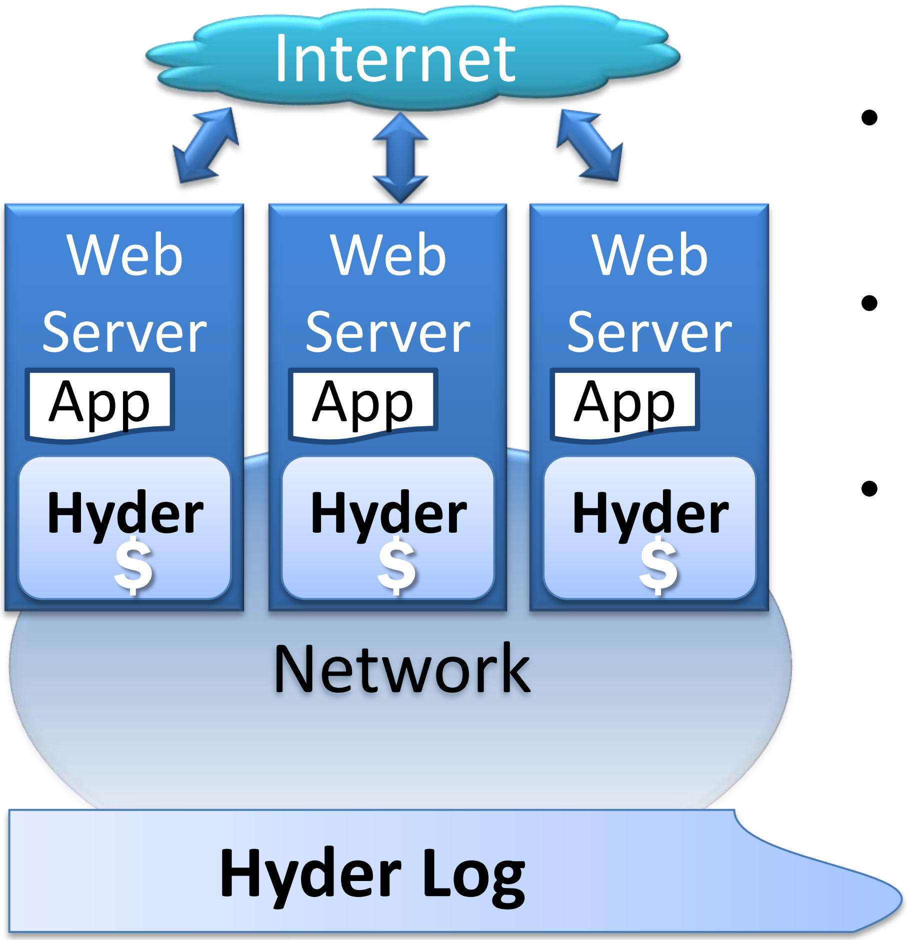
- Database is partitioned across multiple servers
- For scalability, avoid distributed transactions
- Several layers of caching
- App is responsible for
  - cache coherence
  - consistency of cross-partition queries
- Must carefully configure to balance the load

Hyder Scales Out Without Partitioning



- The log is the database
- No partitioning is required
  - Servers share a reliable, distributed log
- Database is multi-versioned, so server caches are trivially coherent
  - Servers can fetch pages from the log or other servers' caches

Hyder Runs in the Application Process

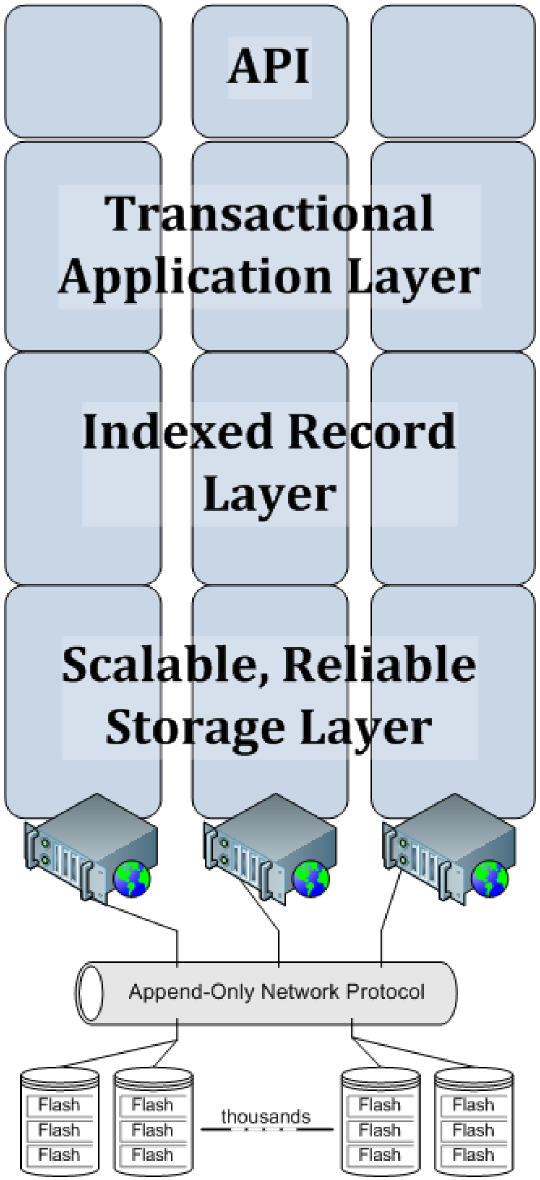


- Simple high performance programming model
- No need for client and server caches, plus a cache server
- Avoids the expense of RPC’s to a database server

Enabling Hardware Assumptions

- I/O operations are now cheap and abundant
  - Raw flash offers at least 10^4 more IOPS/GB than HDD
  - $\Rightarrow$ Can spread the database across a log, with less physical contiguity
- Cheap high-performance data center networks
  - 1Gbps broadcast, with 10Gbps coming soon
  - Round-trip latencies already under 25 µs on 10 GigE
  - $\Rightarrow$ Can have many servers sharing storage, with high performance
- Large, cheap, 64-bit addressable memories
  - Commodity web servers can maintain huge in-memory caches
  - $\Rightarrow$ Reduces the rate that Hyder needs to access the log
- Many-core web servers
  - Computation can be squandered
  - $\Rightarrow$ Hyder uses it to maintain consistent views of the database....

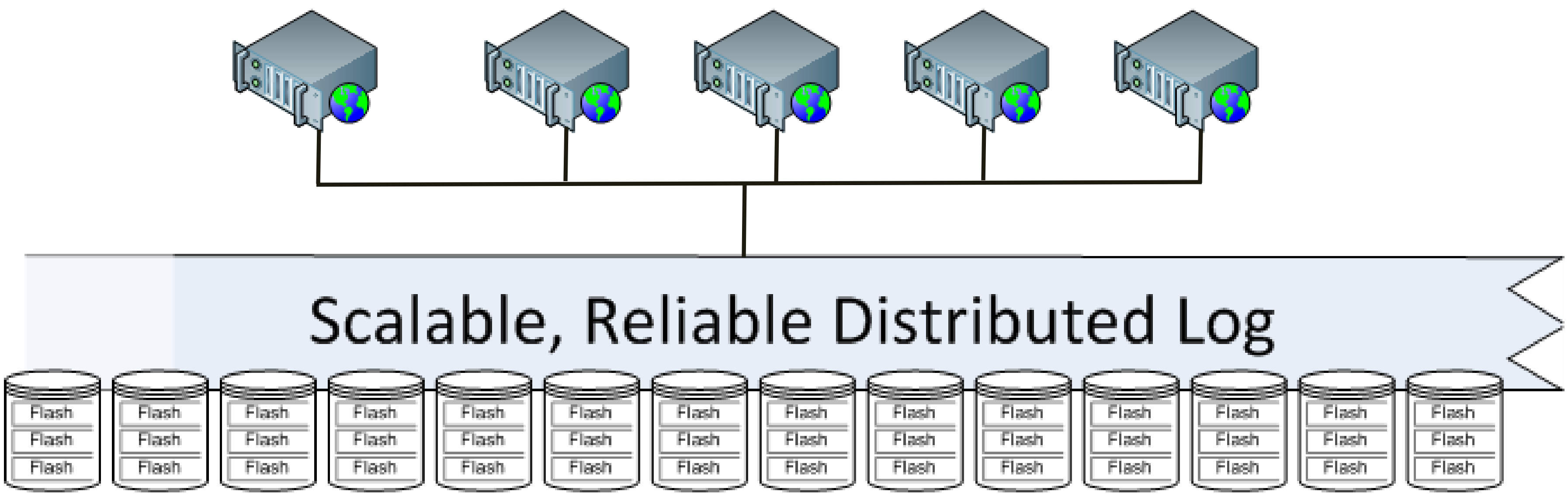
The Hyder Stack



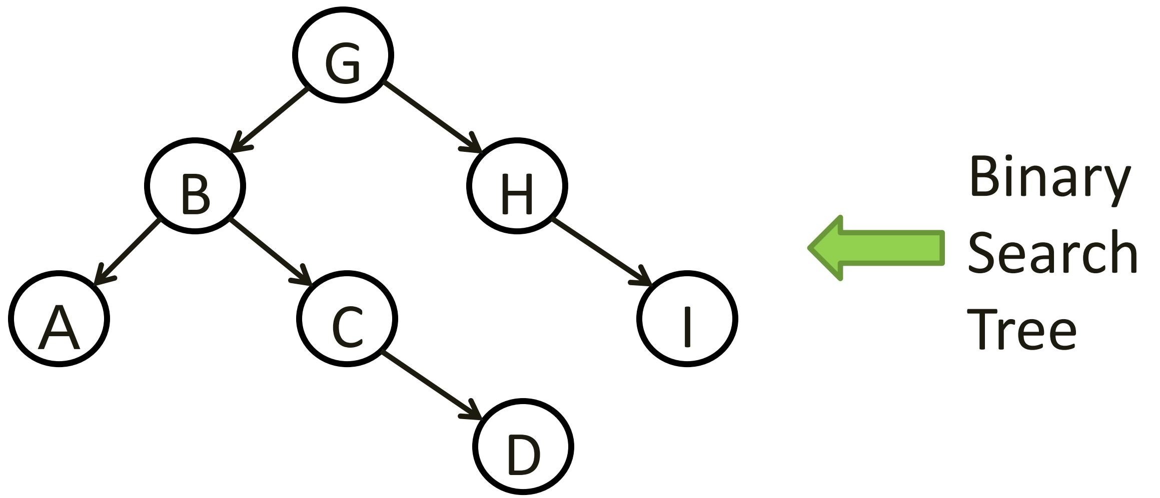
- **Persistent programming language**
  LINQ or SQL layered on Hyder
- **Optimistic transaction protocol**
  Supports standard isolation levels
- **Multi-versioned binary search tree**
  Mapped to log-structured storage
- **Segments, stripes and streams**
  Highly available, load balanced and self-managing log structured storage
- **Custom controller interface**
  Flash units are append-only

Hyder Stores its Database in a Log

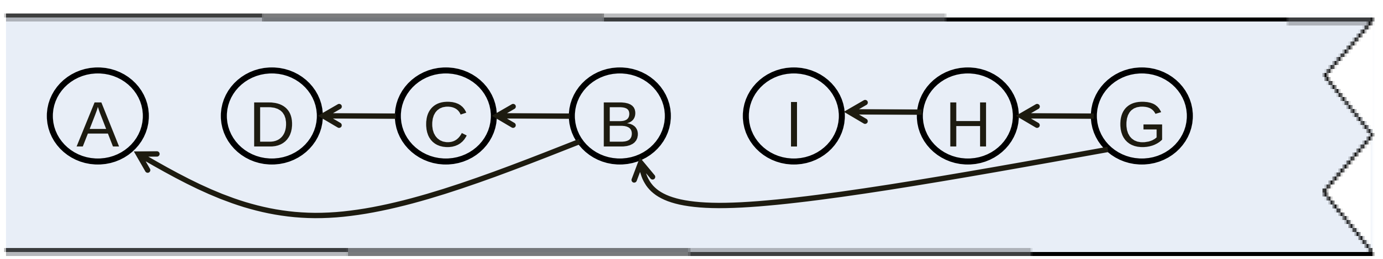
- Log uses RAID erasure coding for reliability



Database is a Binary Search Tree

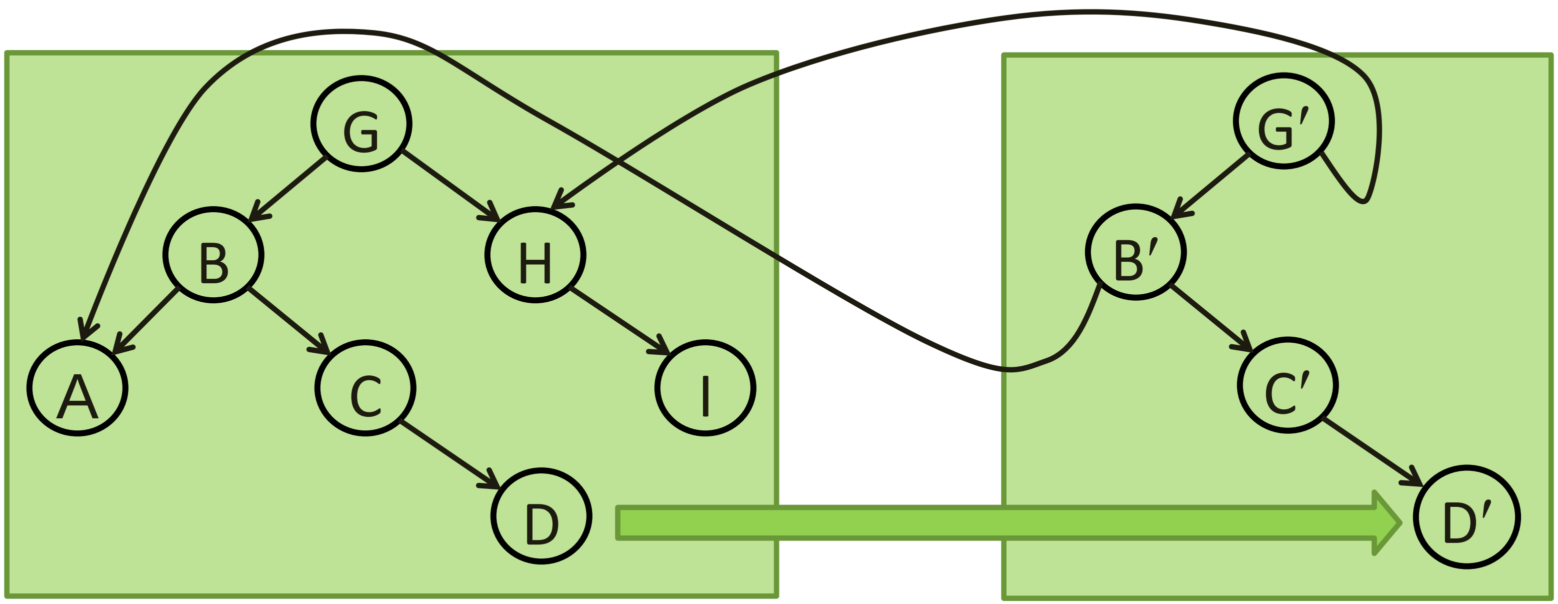


Tree is marshaled into the log



Binary Tree is Multi-versioned

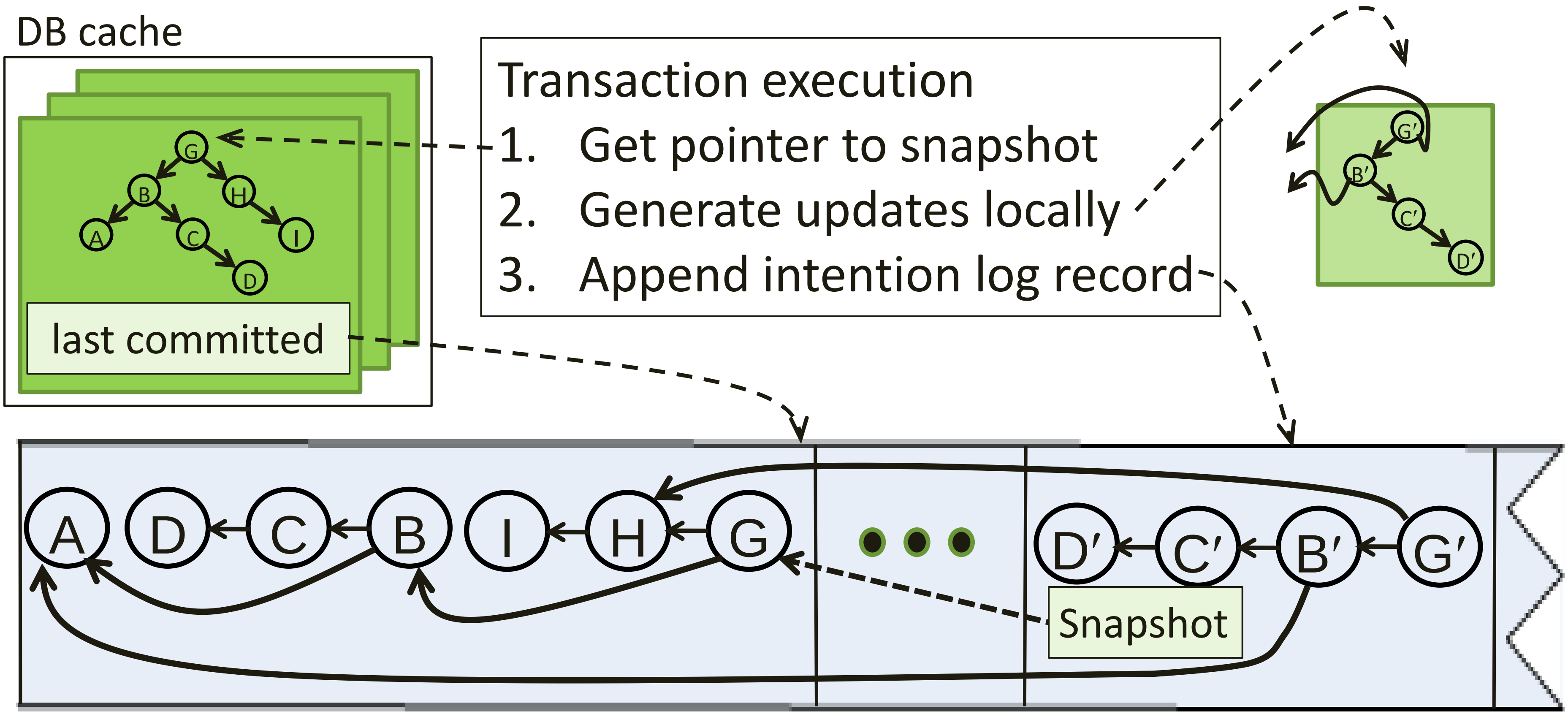
- Copy on write
- To update a node, replace nodes up to the root



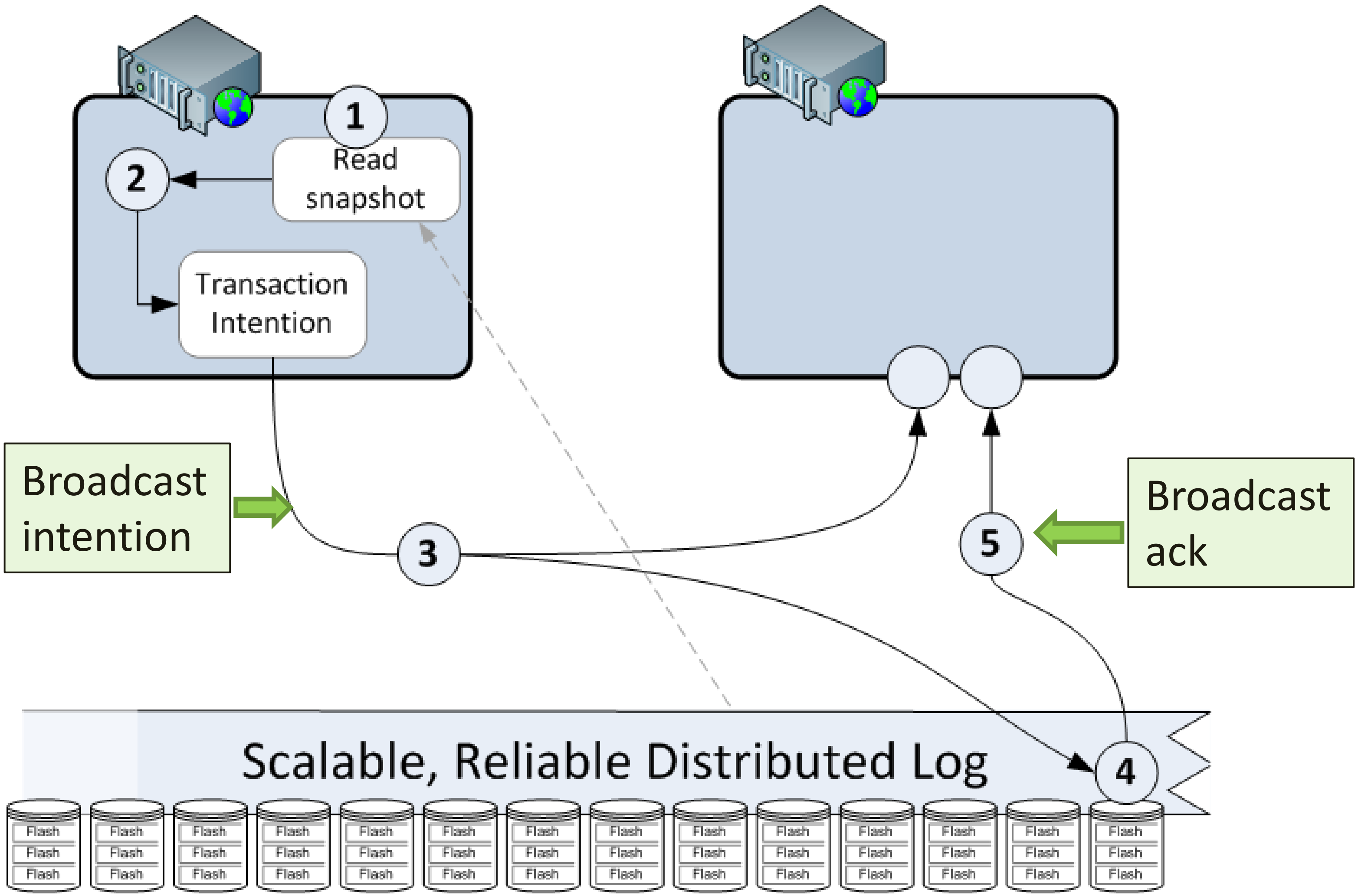
Update
D’s value

Transaction Execution

- Each server has a cache of the last committed database state
- A transaction reads a snapshot and generates an **intention log record**

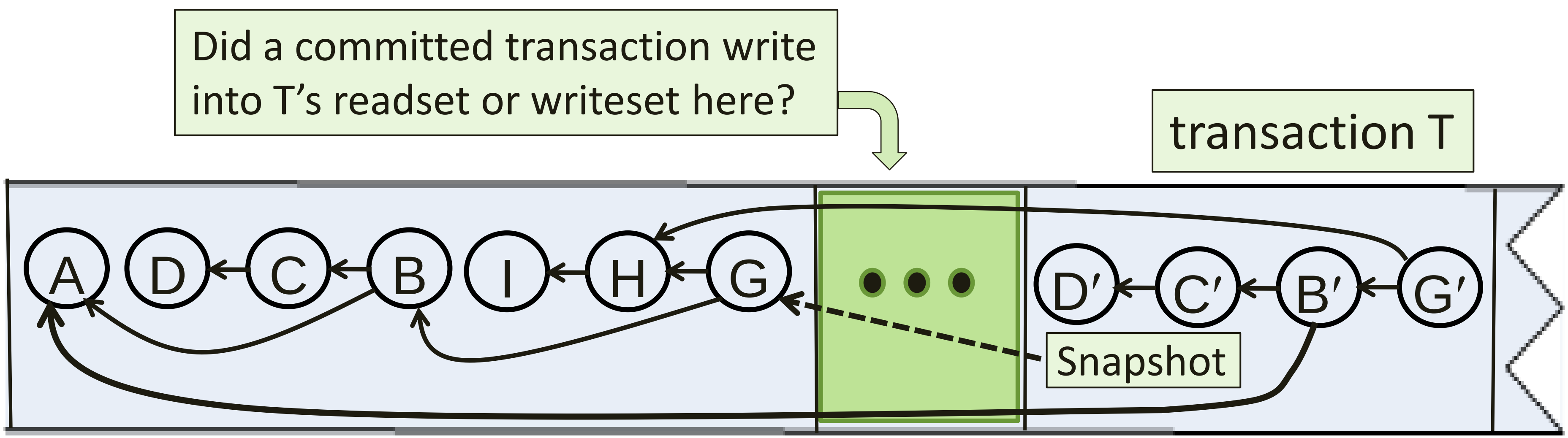


Log Updates are Broadcast



Transaction Commit

- Every server executes a roll-forward of the log
- When it processes an intention log record,
  - it checks whether the transaction experienced a conflict
  - if not, the transaction committed and the server merges the intention into its last committed state
- All servers make the same commit/abort decisions



Hyder Transaction Flow

Transaction starts with a recent consistent snapshot

Optimistic concurrency violation causes transaction to abort and optionally retry

Each server sequentially merges each intention into the committed state cache

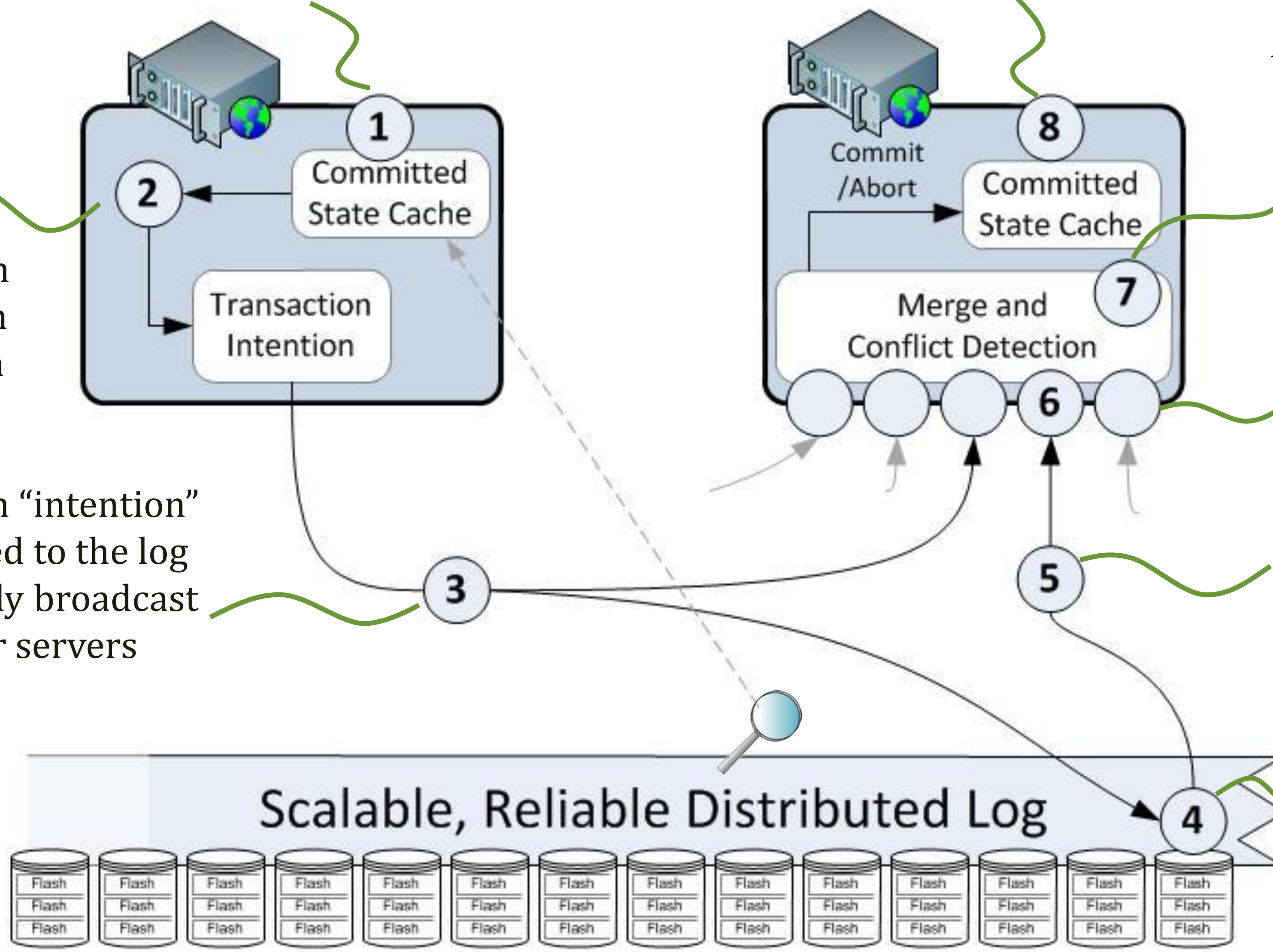
Transaction executes on application server

Messages are received over UDP and parsed in parallel

Transaction "intention" is appended to the log and partially broadcast to other servers

Intention log sequence is broadcast to all servers

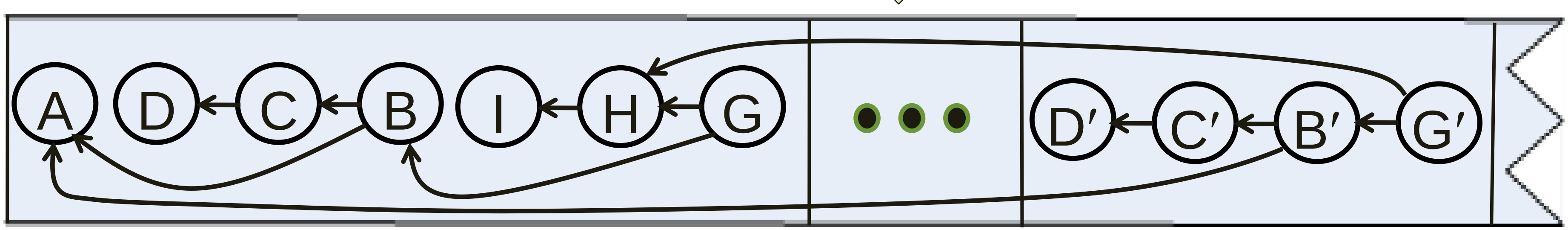
Intention is durably stored in the log



Performance

- The system scales out without partitioning
- System-wide throughput of update transactions is bounded by the slowed step in the update pipeline
  - 15K update transactions per second possible over 1 Gigabit Ethernet
  - 150K update transactions per second expected on 10 Gigabit Ethernet
  - Conflict detection & merge can do about 300K update transactions per second
- Abort rate on write-hot data is bounded by txn's conflict zone
  - Which is determined by end-to-end transaction latency.
  - About 200 μs in our prototype $\Rightarrow$ ~ 1500 update TPS if all txns conflict

Minimize the length of the conflict zone



Major Technologies

- Flash is append-only. Custom controller has mechanisms for synchronization & fault tolerance
- Storage is striped, with a self-adaptive algorithm for storage allocation and load balancing
- Fault-tolerant protocol for a totally ordered log
- Fast algorithm for conflict detection and merging of intention records into last-committed state

Status

- Most parts have been prototyped.
  – But there’s a long way to go.

- We’re working on papers
  – HTPS abstracts are the first.