SDC

STORAGE DEVELOPER CONFERENCE

SNIA SANTA CLARA, 2014

# Tango: distributed data structures over a shared log

Mahesh Balakrishnan

Microsoft Research

Collaborators: Dahlia Malkhi, Ted Wobber, Ming Wu, Vijayan Prabhakaran, Michael Wei, John D. Davis, Sriram Rao, Tao Zou, Aviad Zuck.

# what this talk is really about

building distributed systems with strong properties*
does not require complex distributed protocols…

all you need is the right *storage* abstraction

*fault-tolerance, persistence, high availability, strong consistency, elastic scalability, failure atomicity, transactional isolation, disaster tolerance…


2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# big (meta)data

- design pattern: distribute data, *centralize metadata*
- schedulers, allocators, coordinators, namespaces, indices (e.g. HDFS namenode, SDN controller…)
- usual plan: harden centralized service later

"***Coordinator failures will be handled safely*** using the ZooKeeper service [14]." Fast Crash Recovery in RAMCloud, Ongaro et al., SOSP 2011.

"***Efforts are also underway to address high availability*** of a YARN cluster by having passive/active failover of RM to a standby node." Apache Hadoop YARN: Yet Another Resource Negotiator, Vavilapalli et al., SOCC 2013.

"However, ***adequate resilience can be achieved*** by applying standard replication techniques to the decision element." NOX: Towards an Operating System for Networks, Gude et al., Sigcomm CCR 2008.

- … but hardening is difficult!


2014 Storage  Developer Conference. © Insert Your Company Name.  All Rights Reserved.

# the abstraction gap for metadata

centralized metadata services are built using in-memory data structures (e.g. Java / C# Collections)

- state resides in maps, trees, queues, counters, graphs…
- transactional access to data structures
  - example: a scheduler atomically moves a node from a free list to an allocation map

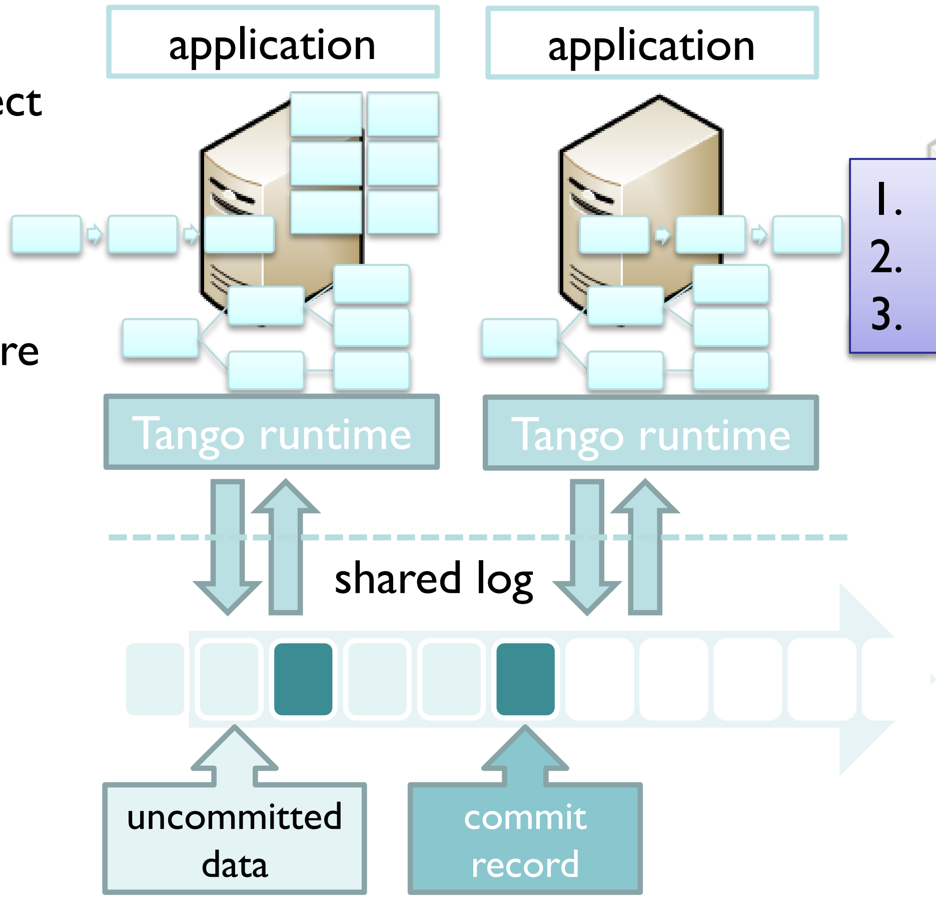
adding high availability requires different abstractions

- move state to external service like ZooKeeper
- restructure code to use state machine replication
- implement custom replication protocols


2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# the Tango abstraction

a Tango object
**=**
**view**
in-memory
data structure
**+**
**history**
ordered
updates in
shared log




1. Tango objects are **easy to use**
2. Tango objects are **easy to build**
3. Tango objects are **fast, scalable**

the shared log is the source of
- persistence
- availability
- elasticity
- atomicity and isolation

… across multiple objects

no messages… only appends/reads on the shared log!

2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

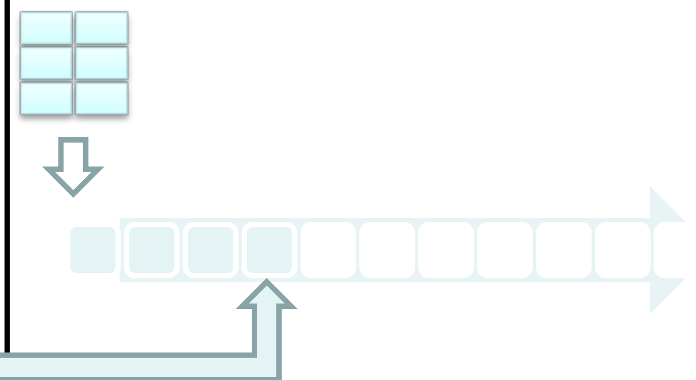
# Tango objects are easy to use

- implement standard APIs (Java/C# Collections)
- linearizability for single operations

**example:**

```
curowner = ownermap.get("ledger");
if(curowner.equals(myname))
        ledger.add(item);
```

**under the hood:**



SDC 14

2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# Tango objects are easy to use

- implement standard APIs (Java/C# Collections)
- linearizability for single operations
- serializable transactions

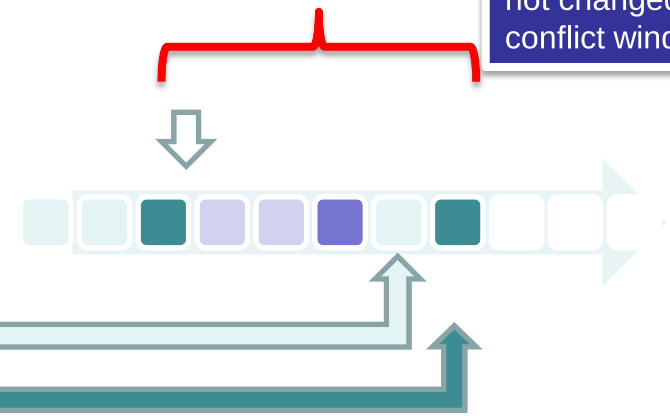
**example:**

```
TR.BeginTX();
curowner = ownermap.get("ledger");
if(curowner.equals(myname))
        ledger.add(item);
status = TR.EndTX();
```

speculative commit records: each client decides if the TX commits or aborts **independently but deterministically**
[similar to Hyder (Bernstein et al., CIDR 2011)]

**under the hood:**

TX commits if read-set (*ownermap*) has not changed in conflict window



TX commit record:
read-set: (ownermap, ver:2)
write-set: (ledger, ver:6)

SDC 14

2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# Tango objects are easy to build

15 LOC == persistent, highly available, transactional register

```
class TangoRegister {
        int oid;
        TangoRuntime *T;
        int state;
        void apply(void *X) {
                state = *(int *)X;
        }
        void writeRegister (int newstate) {
                T->update_helper(&newstate , sizeof (int) , oid);
        }
        int readRegister () {
                T->query_helper(oid);
                return state;
        }
}
```

object-specific state

invoked by Tango runtime on EndTX to change state

mutator: updates TX write-set, appends to shared log

accessor: updates

ate

Other examples:
Java ConcurrentMap: 350 LOC
Apache ZooKeeper: 1000 LOC
Apache BookKeeper: 300 LOC

simple API exposed by runtime to object: 1 upcall + two helper methods
arbitrary API exposed by object to application: mutators and accessors

2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# are Tango objects fast and scalable?

problem: shared logs don’t scale!

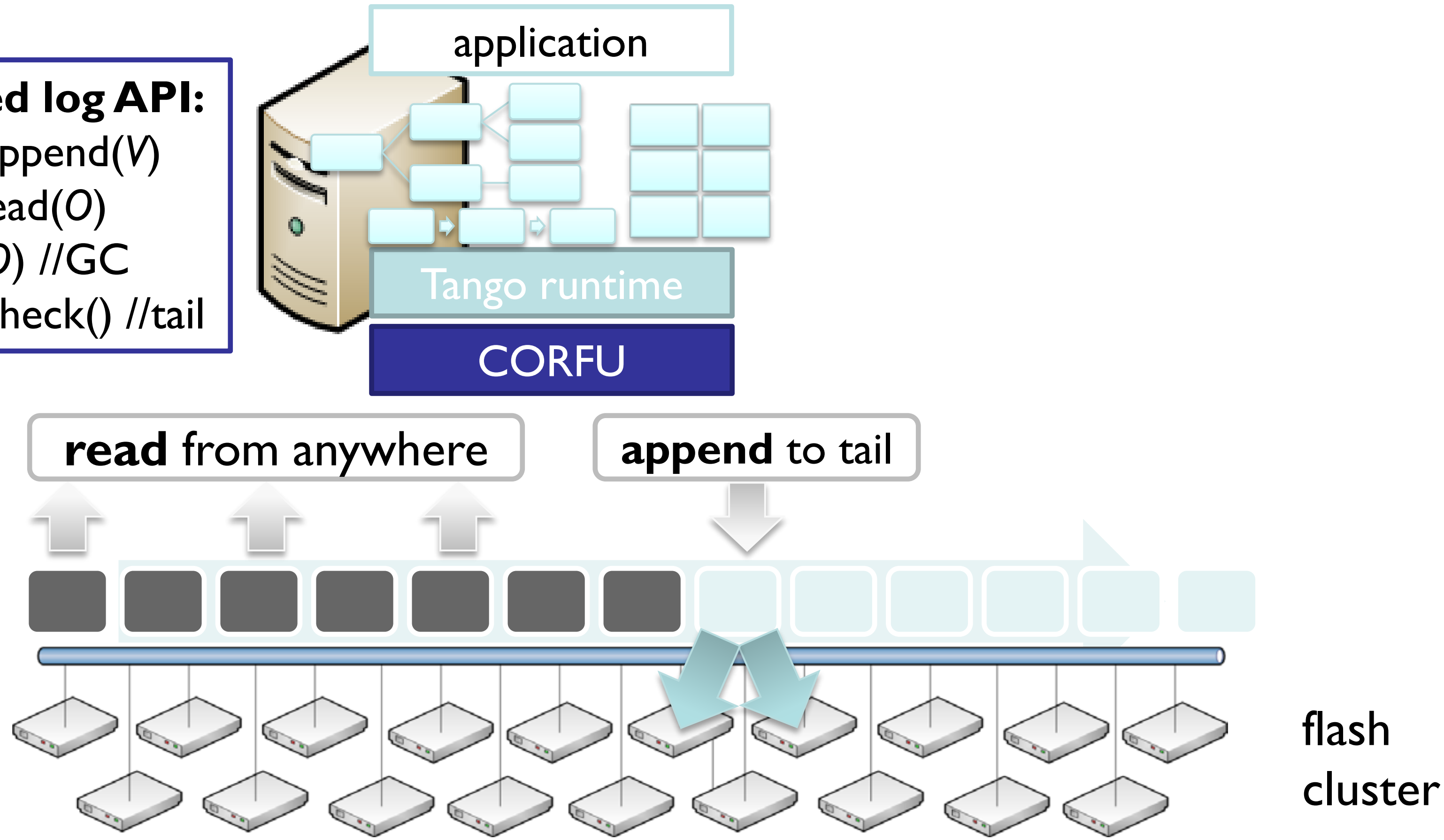
- fault-tolerant implementation requires a Paxos-like consensus protocol…
- … and Paxos doesn’t scale.

secret sauce: the CORFU distributed shared log


2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# the CORFU distributed shared log

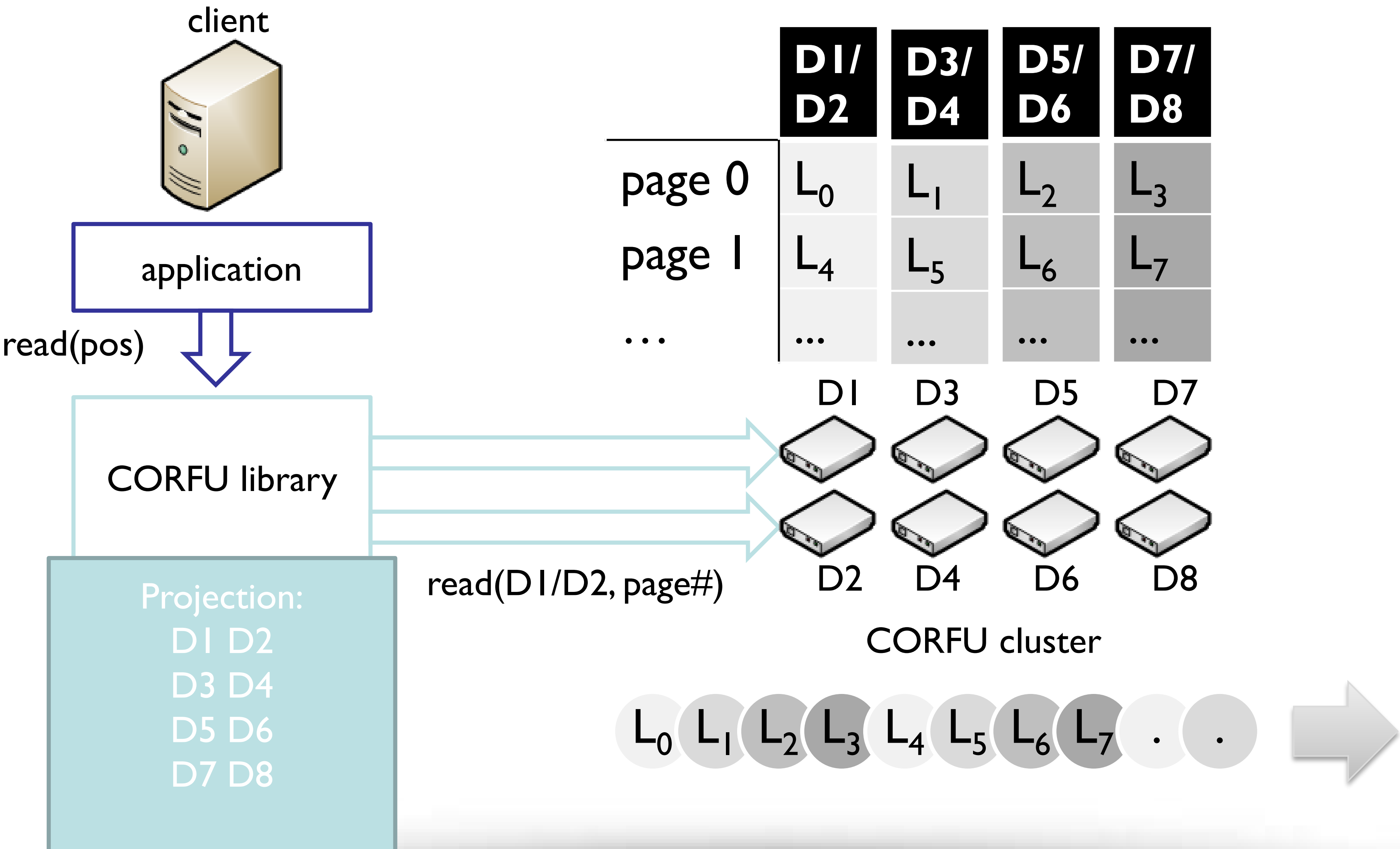
**shared log API:**
O = append(*V*)
V = read(*O*)
trim(*O*) //GC
O = check() //tail




each logical entry is mapped to a replica set of flash pages


2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

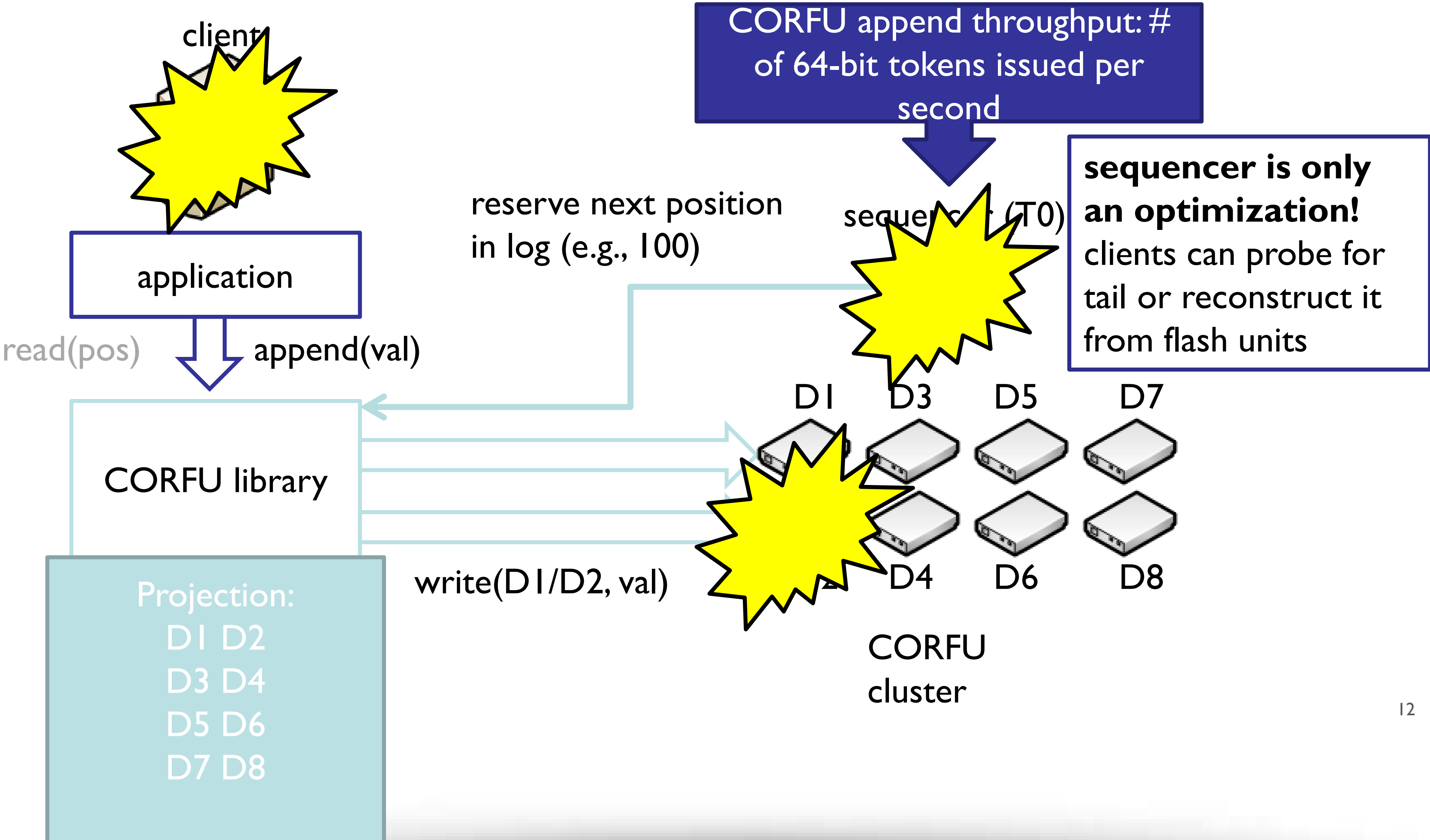
# the CORFU protocol: reads






2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

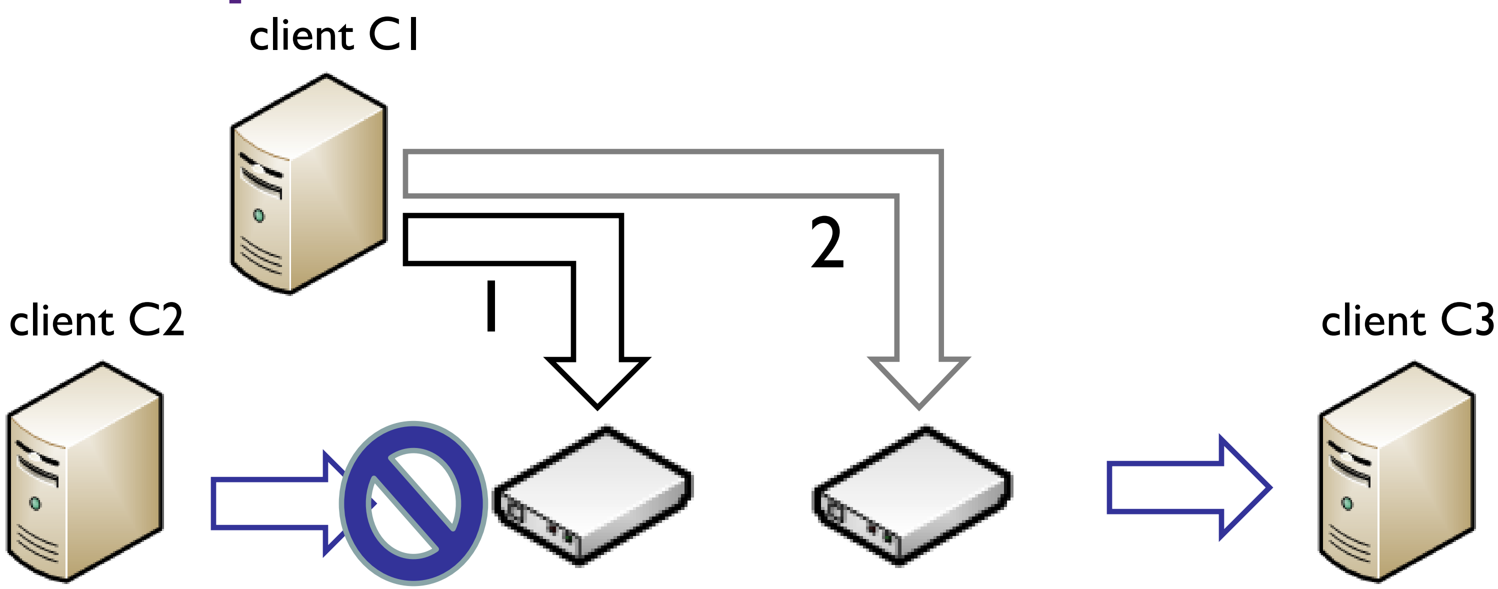
# the CORFU protocol: appends






SDC 14
2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# chain replication in CORFU




**safety under contention:**
if multiple clients try to write to same log position concurrently, only one wins
writes to already written pages => error

**durability:**
data is only visible to reads if entire chain has seen it
reads on unwritten pages => error

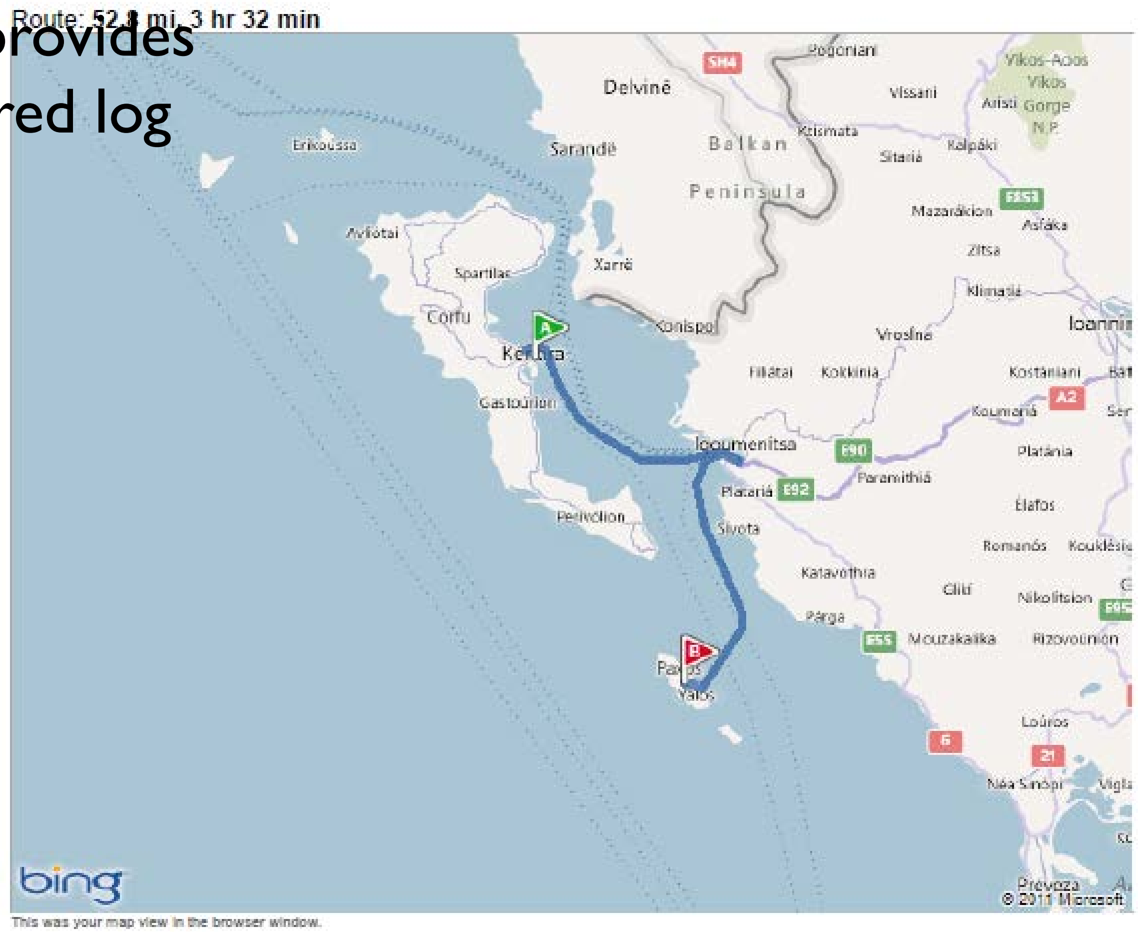
requires `write-once' semantics from flash unit


2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# how far is CORFU from Paxos?

Multi-Paxos provides subset of shared log functionality

Multi-Paxos p
IO-bound at
so is a single



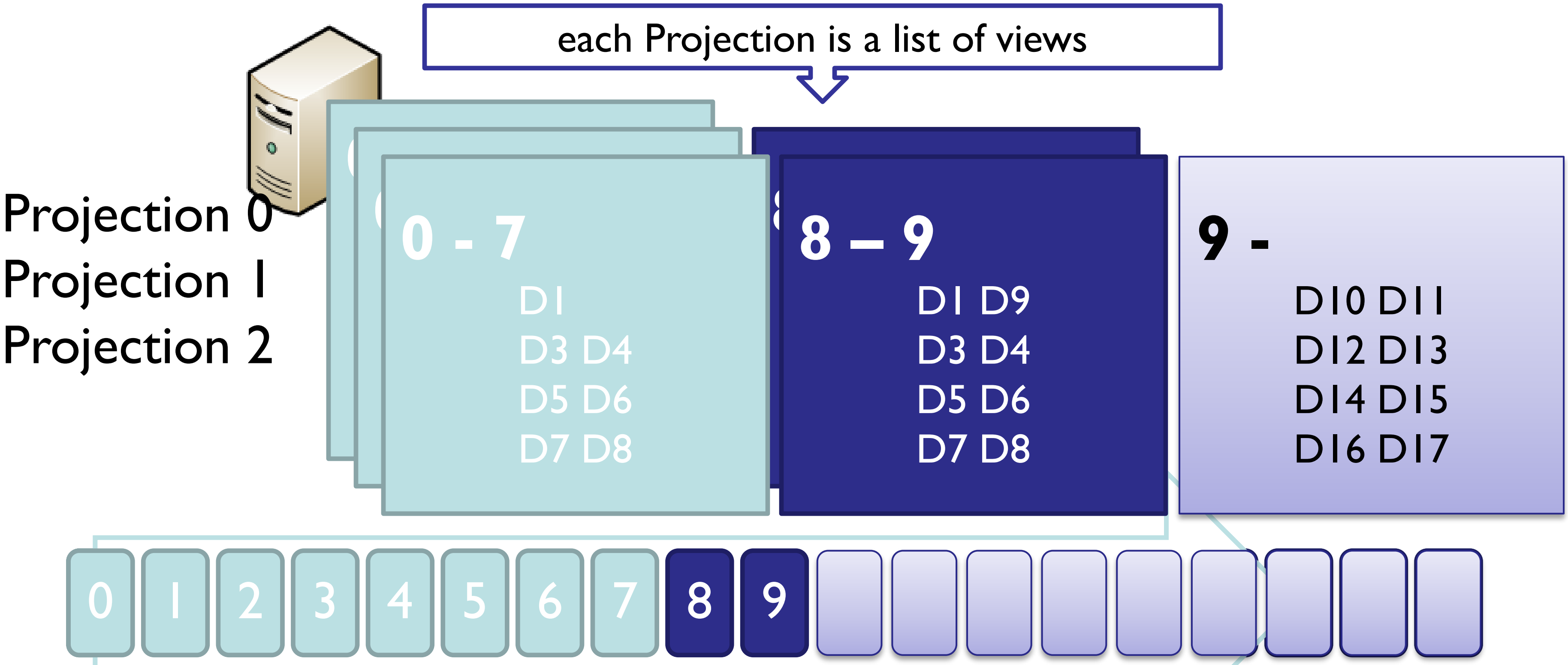

7

8

CORFU shar
across multiple chains.
no I/O bottleneck!


2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# CORFU failures: flash units




reconfiguration steps:
1. 'seal' current projection at flash units
2. write new projection at auxiliary

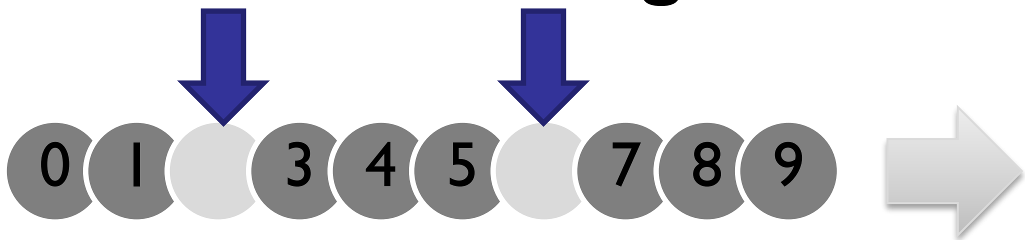
latency for 32-drive cluster:
**tens of milliseconds**

SDC 14
2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# CORFU failures: clients

client obtains token from sequencer and crashes:

**holes in the log**




solution: other clients can fill the hole

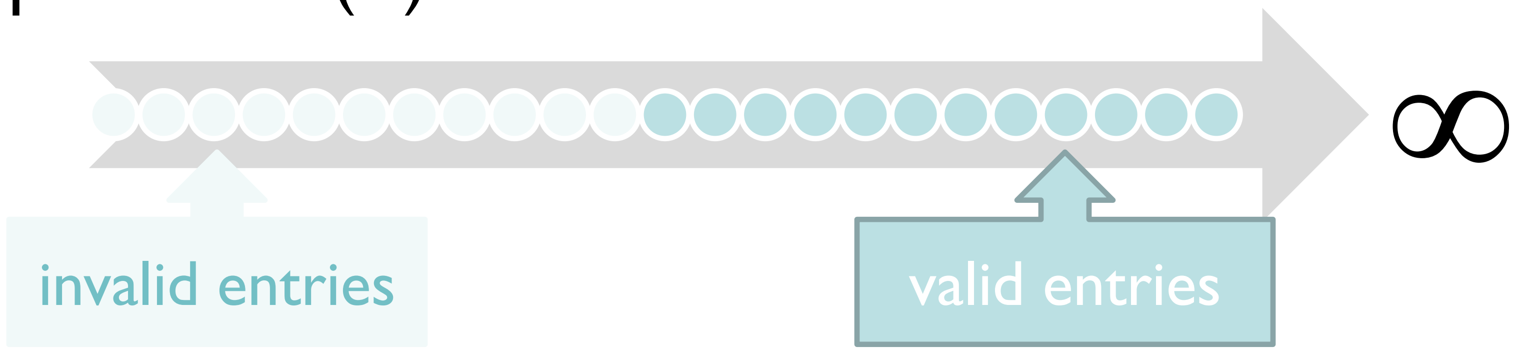
fast CORFU *fill* operation (<1ms) 'walks the chain':
-completes half-written entries
-writes junk on unwritten entries (metadata operation, conserves flash cycles, bandwidth)

SDC 14

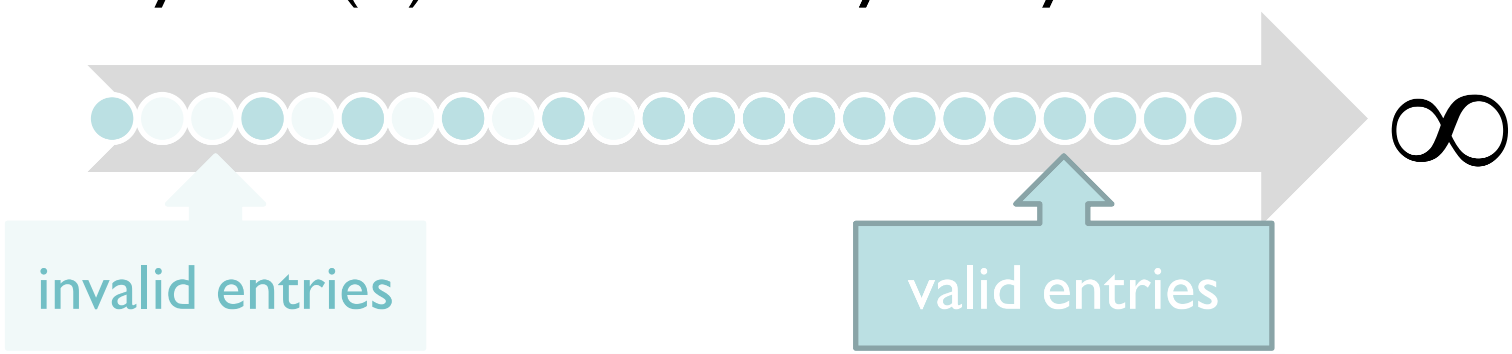
2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# CORFU garbage collection: two models

– prefix trim(*O*): invalidate all entries before offset *O*



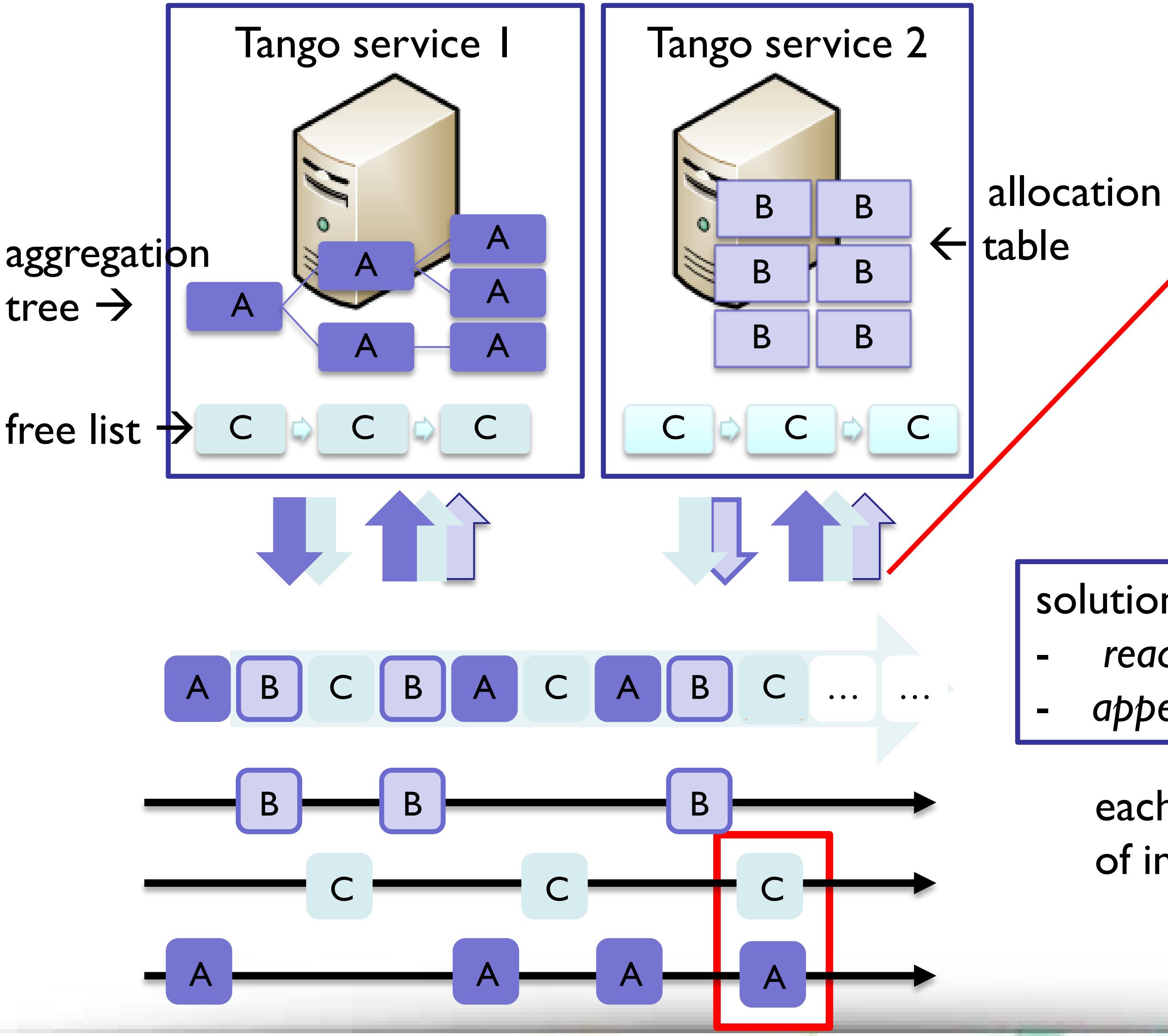

– entry trim(*O*): invalidate only entry at offset *O*





2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# a fast shared log isn't enough…



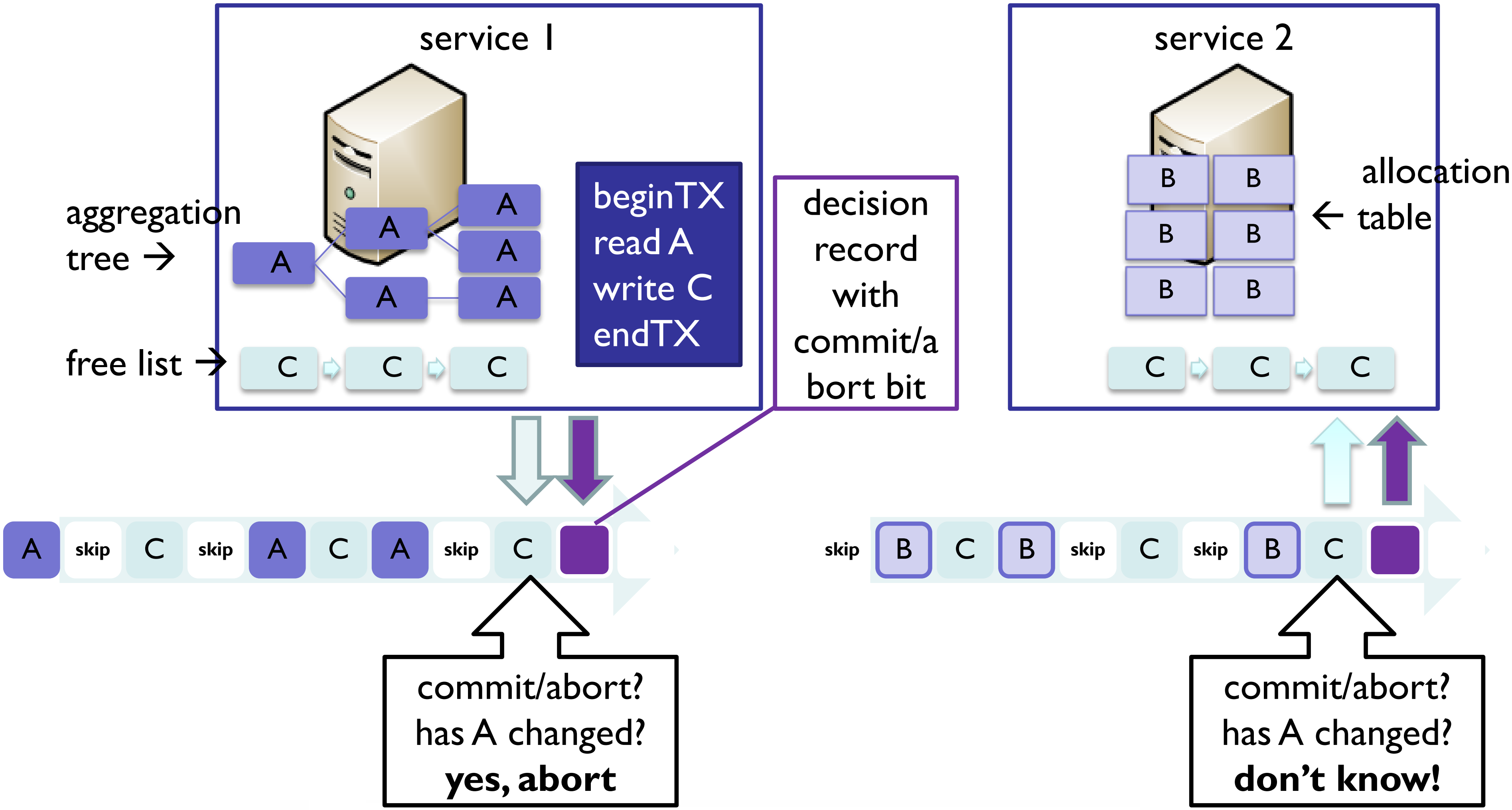

**the playback bottleneck**:
clients must read all entries → inbound NIC is a bottleneck

solution: **stream** abstraction
- *readnext(streamid)*
- *append(value, streamid1, … )*

each client only plays entries of interest to it


2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

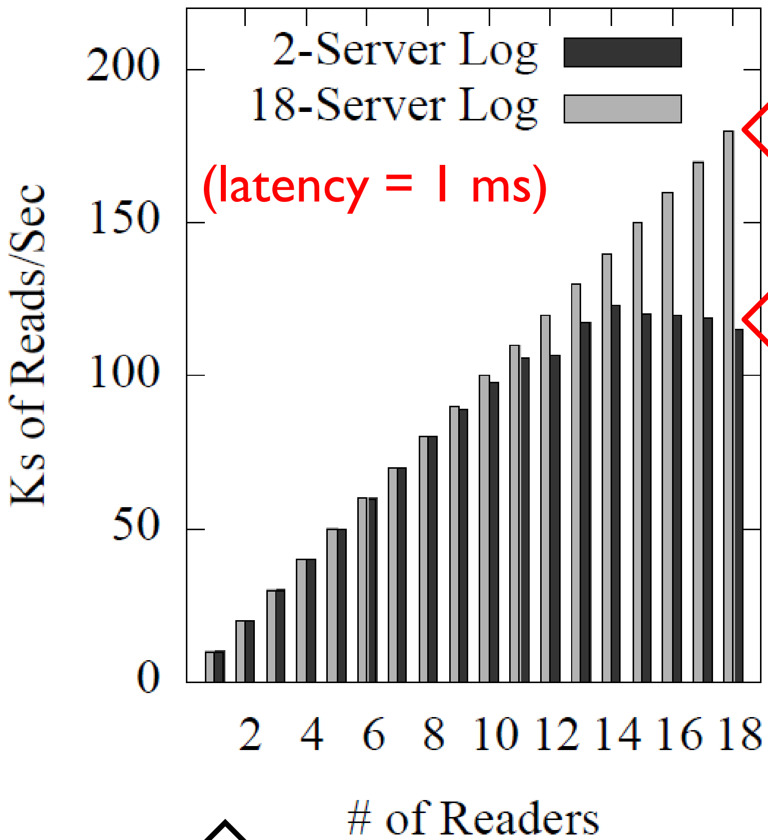
# transactions over streams





2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# evaluation: linearizable operations




beefier shared log → scaling continues…
ultimate bottleneck: sequencer

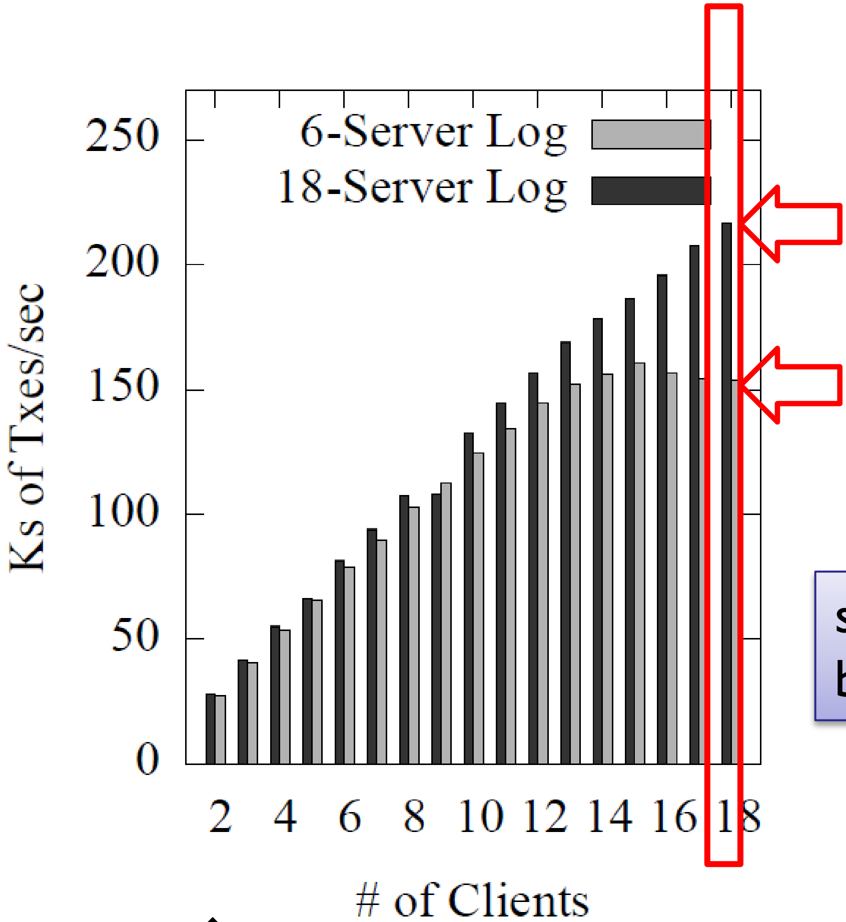
adding more clients → more reads/sec
… until shared log is saturated

a Tango object provides elasticity for strongly consistent reads

constant write load (10K writes/sec), each client adds 10K reads/sec

2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# evaluation: single object txes




beefier shared log → scaling continues…

ultimate bottleneck: sequencer

adding more clients → more transactions

… until shared log is saturated

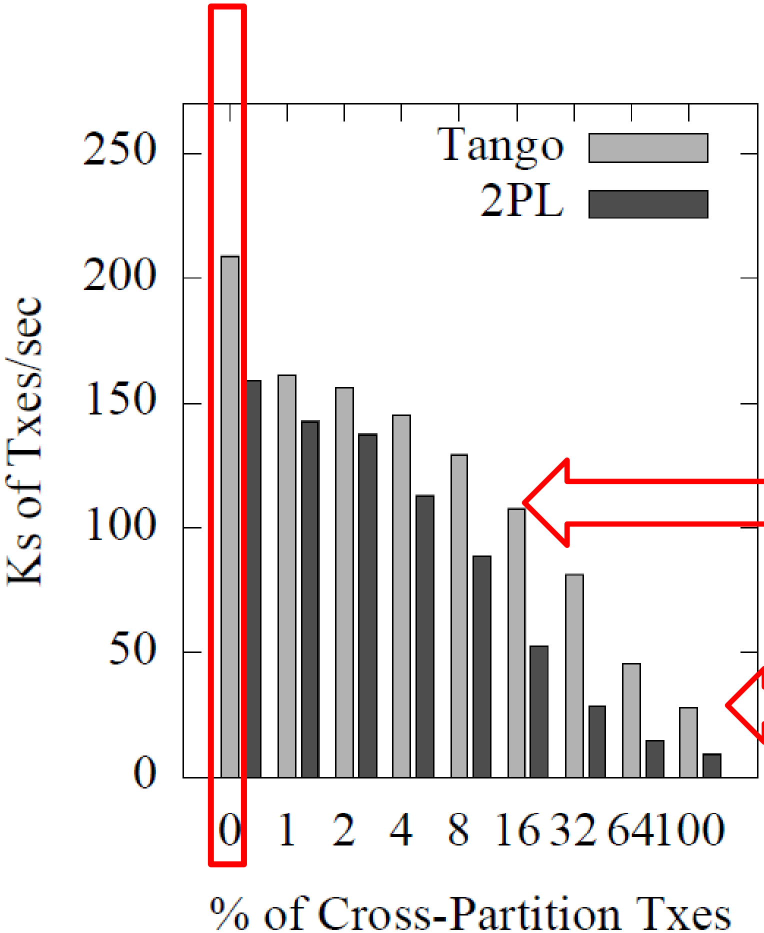
scales like conventional partitioning…
but there's a cap on aggregate throughput

each client does transactions over its own TangoMap

SDC 14

2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# evaluation: multi-object txes




Tango enables fast, distributed transactions across multiple objects

over 100K txes/sec when 16% of txes are cross-partition

similar scaling to 2PL…
without a complex distributed protocol

18 clients, each client hosts its own TangoMap
cross-partition tx: client moves element from its TangoMap to some other TangoMap



SDC 14

2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# conclusion

Tango objects: data structures backed by a shared log

key idea: the shared log does all the heavy lifting (persistence, consistency, atomicity, isolation, history, elasticity…)

Tango objects are easy to use, easy to build, and fast.

Distributed systems do not require complex distributed protocols… all you need is the right *storage* abstraction!



2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

# thank you!

2014 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.