Ordering Transactions with Prediction in Distributed Object Stores

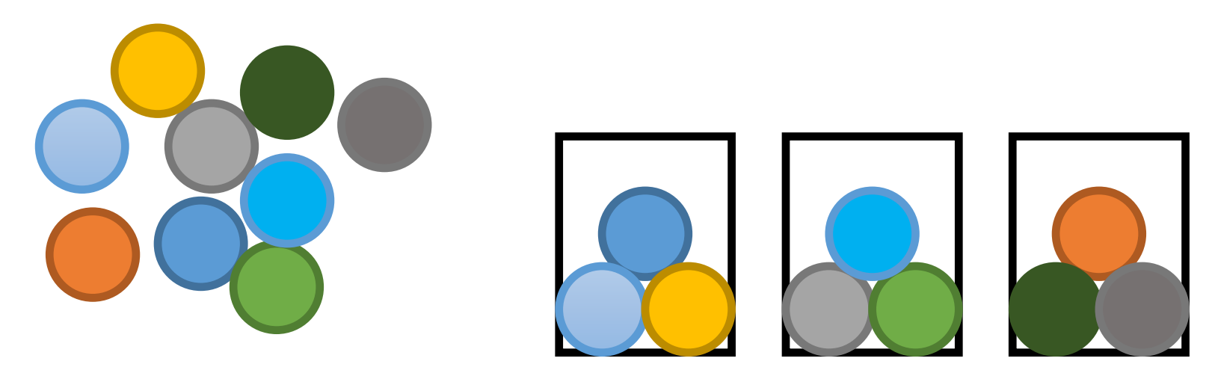
Ittay Eyal[1] Ken Birman[1] Idit Keidar[2] Robbert van-Renesse[1]
[1] Cornell [2] Technion

In a world of big data

we want transactions

```
begin_txn
  Reads        (return value)
  writes       (return ack)
  ...
end_txn        (returns commit/abort)
```

of sharded data



with ACID guarantees

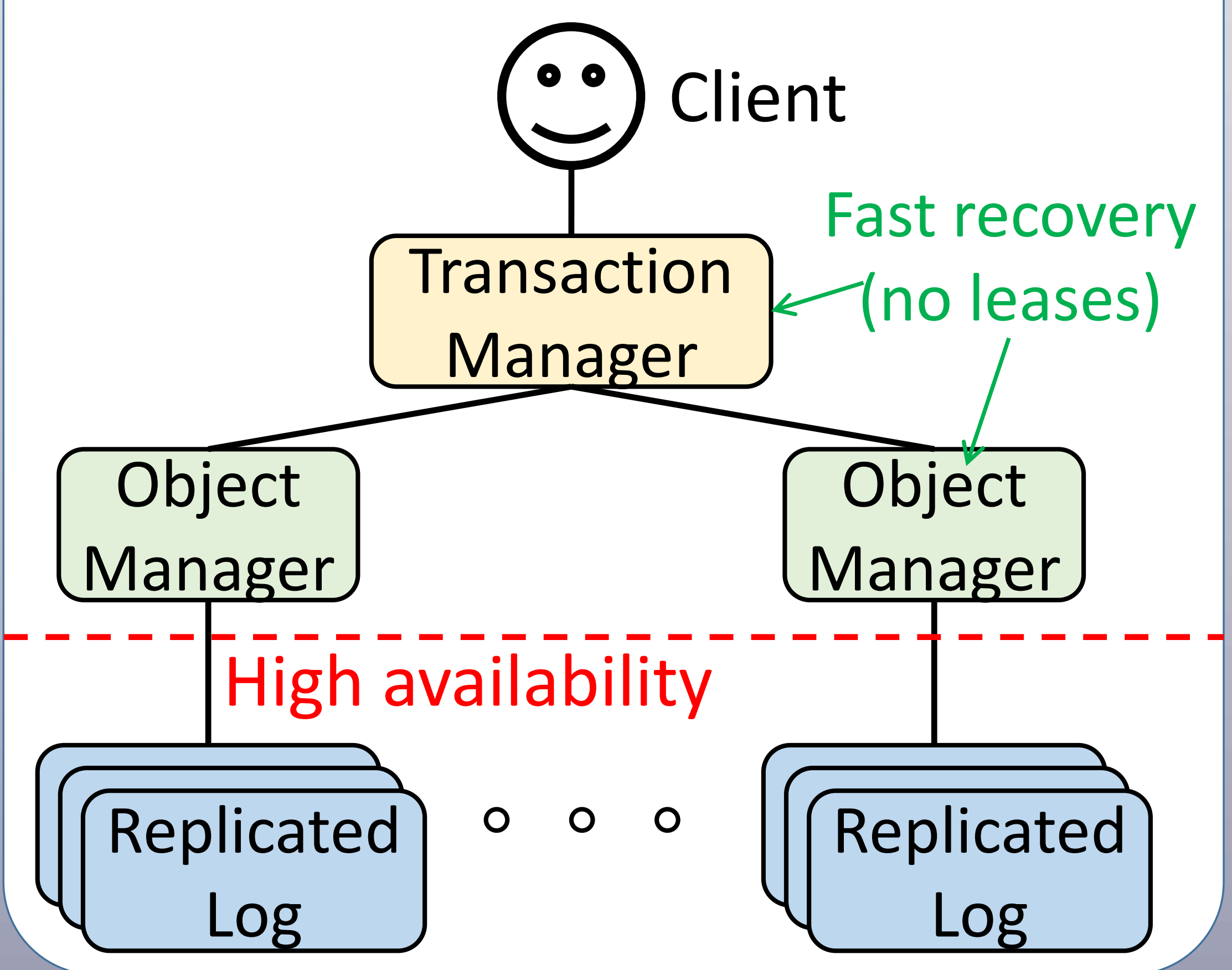
- Atomic
- Consistent
- Isolated
- Durable

Atomic, Consistent, Isolated: Atomic transactions
Durable: High availability

But 2PC doesn't scale.

ACID-RAIN: Ordering with Prediction, Committing with Independent Logs

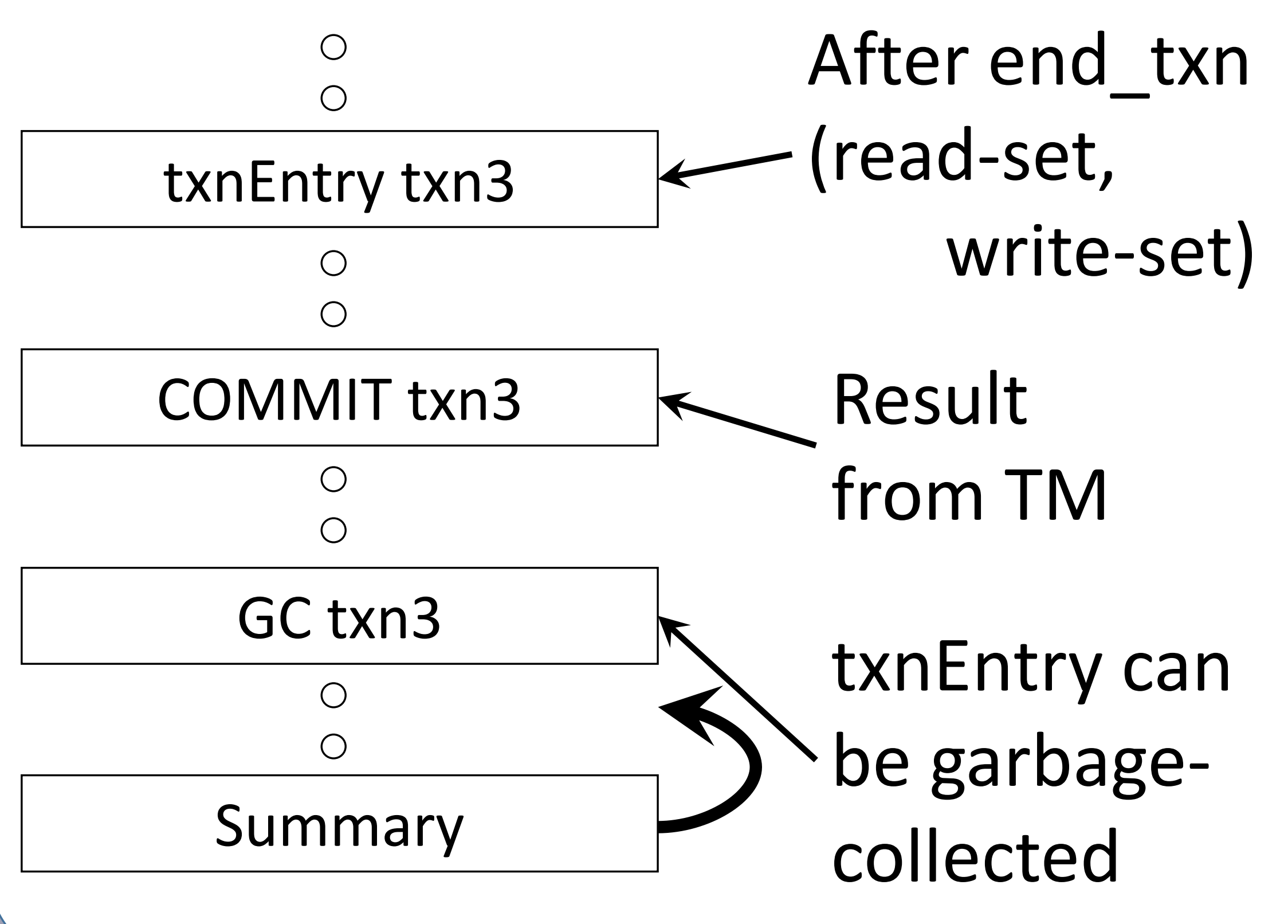
Architecture



Concurrency Control

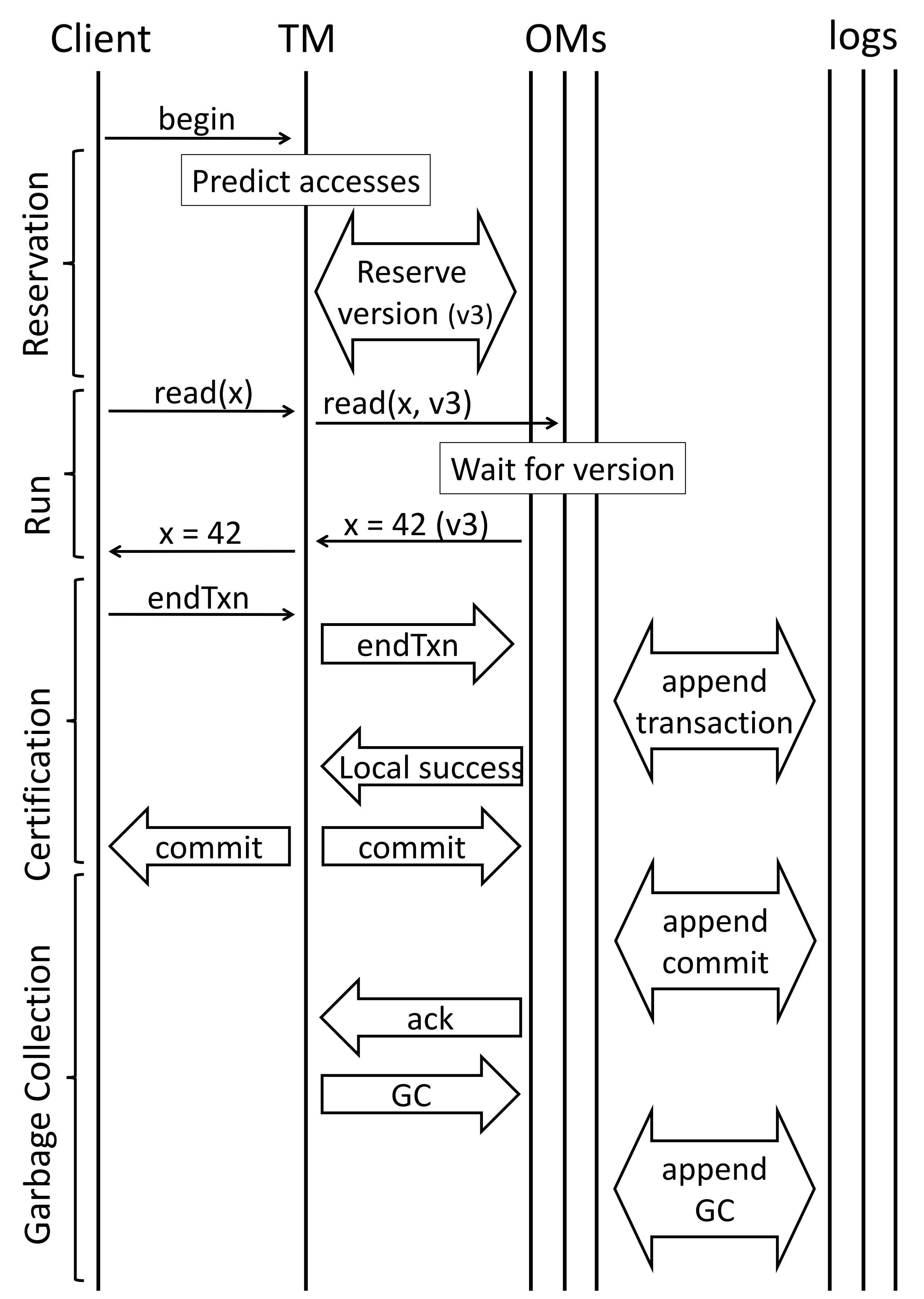
1. Optimistic, transactions run speculatively and then certify.
2. Conflict detection w/ timestamps.
3. Version reservation (lock on future version) by prediction.
4. Final certification at transaction end → lock-free: can replace slow/failed nodes immediately; reservations are only hints.

Log Structure


txnEntry txn3
COMMIT txn3
GC txn3
Summary
After end_txn (read-set, write-set)
Result from TM
txnEntry can be garbage-collected

Execution Example with Prediction

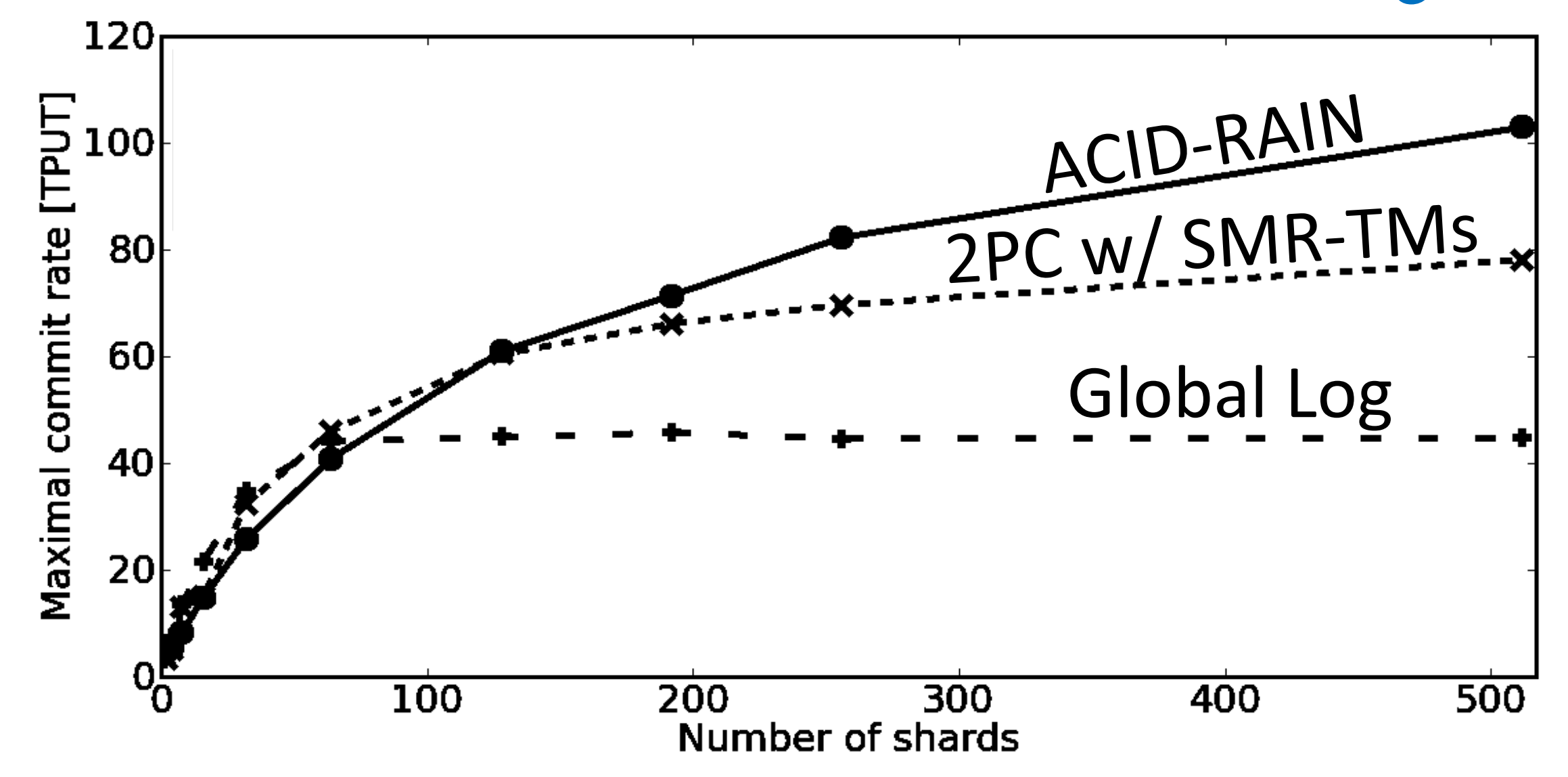
1. Prediction and reservation.
2. Transaction run.
3. Certification.
4. Garbage collection (asynchronous).



Simulation Results

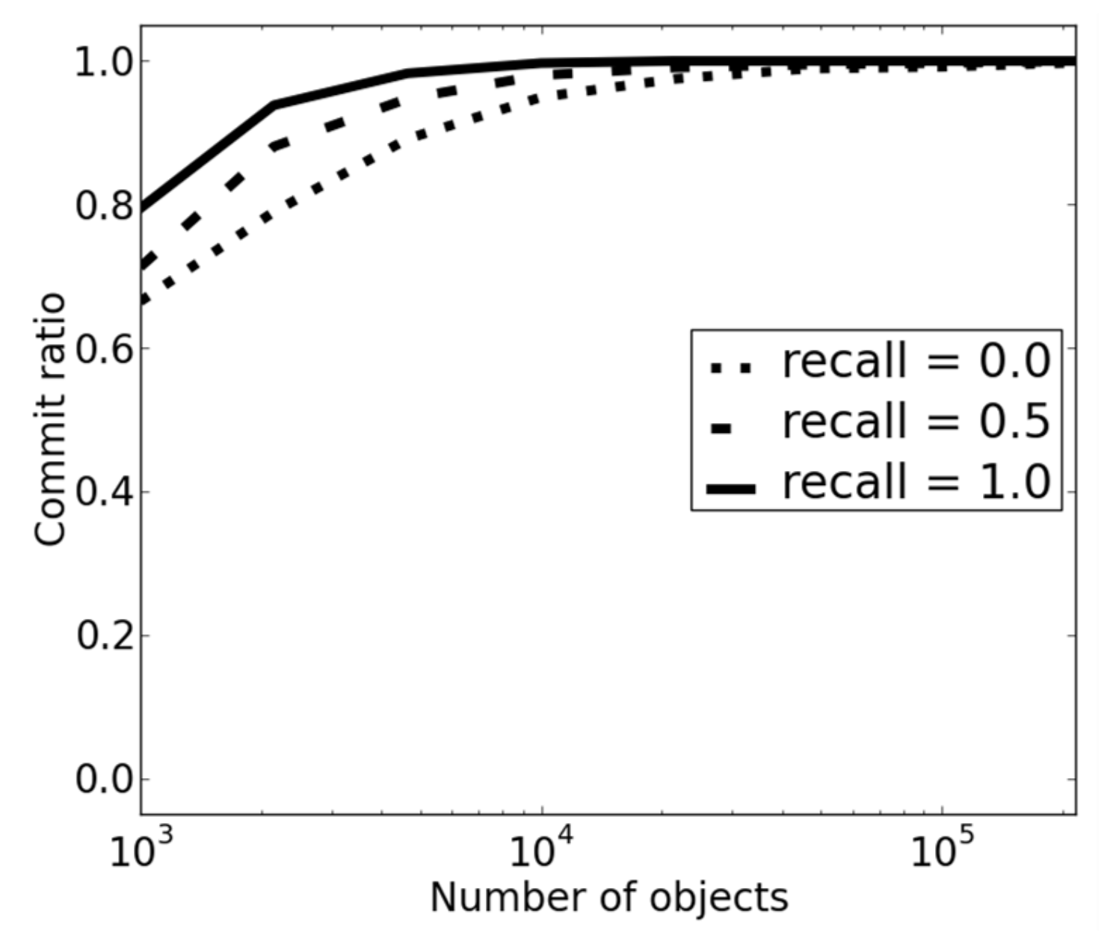
- Custom-made simulator.
- Transactional YCSB workloads.
- Uniform random object access.

Certification Scalability

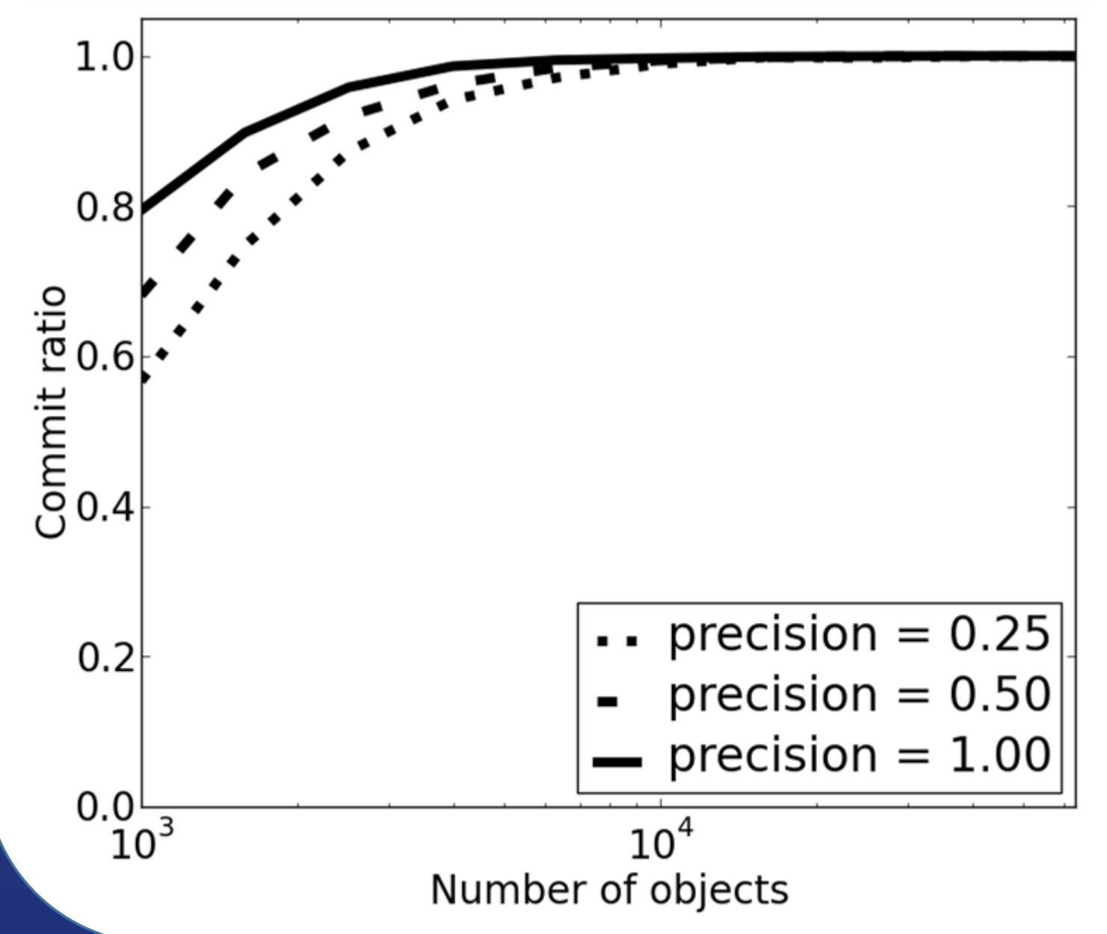


- **Global log**: Forms a bottleneck.
- **2PC with SMR TMs**: longer certification time so higher contention.

Benefits of Prediction



Different recall ratios with perfect precision (no wrong guesses).
recall = 0: no prediction and no reservation (classical approach)
recall = 1.0: predicting all accesses.
Better recall → higher commit ratio



Different precision ratios (wrong guesses) with perfect recall.
Bad precision → more conflicts in small data sets