Blizzard

A distributed scalable queue service

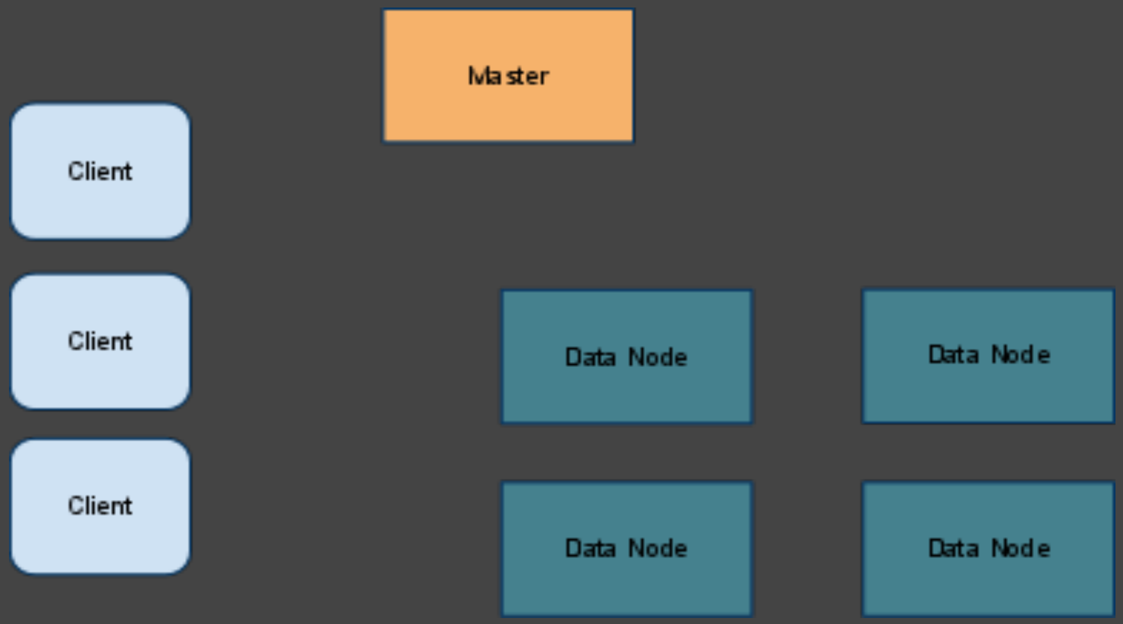
Motivation

- Managing large data sets
- Many concurrent clients
- Request tracking
  - E-commerce
  - User support
- Distributed computation
- Dynamic Scalability

Goals

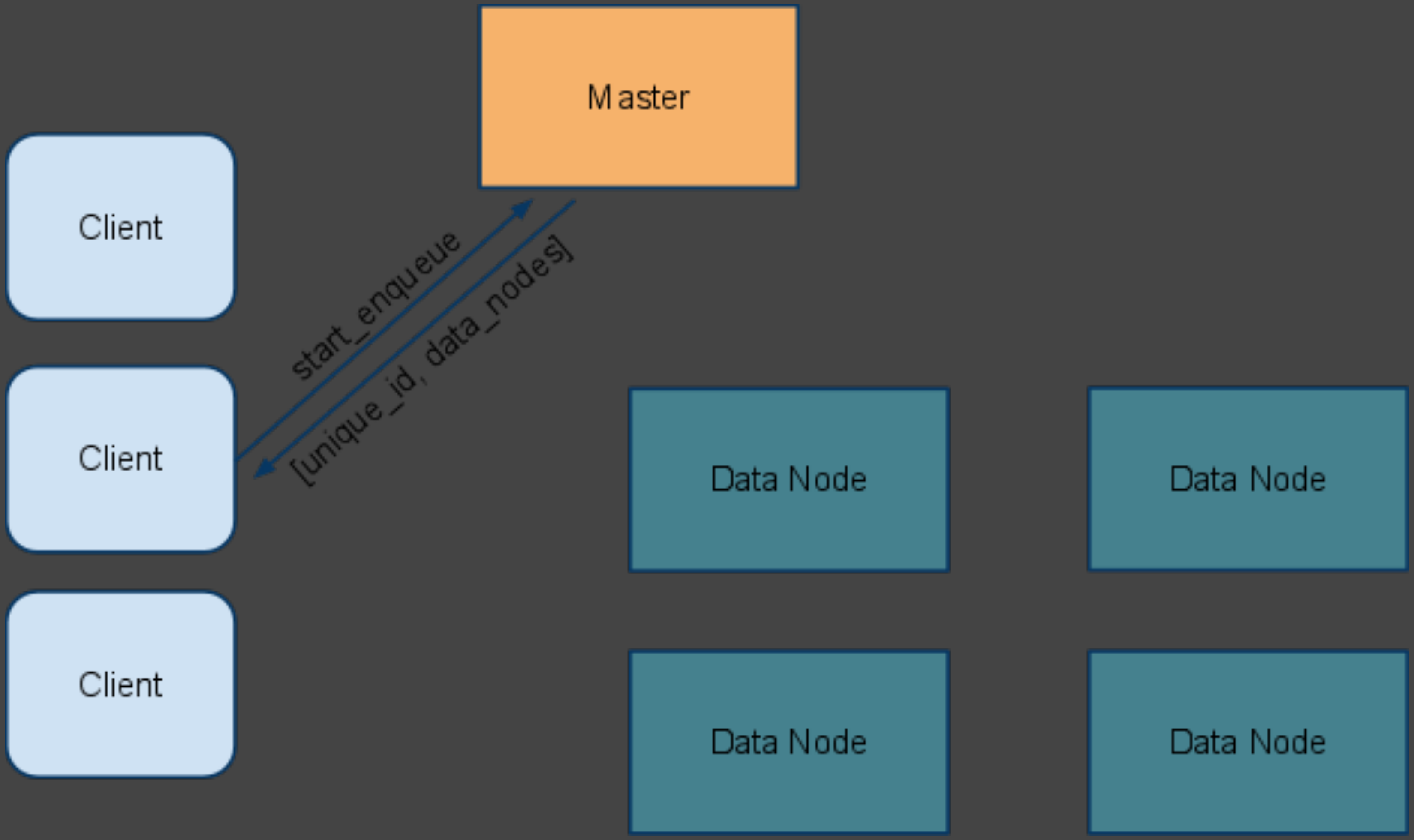
- Fault Tolerance
  - Adjustable failures tolerance
- Persistence
  - of data
  - of queue state
- Concurrency
- Scalable Performance
- Order perseverance
  - No FIFO in current systems

Implementation



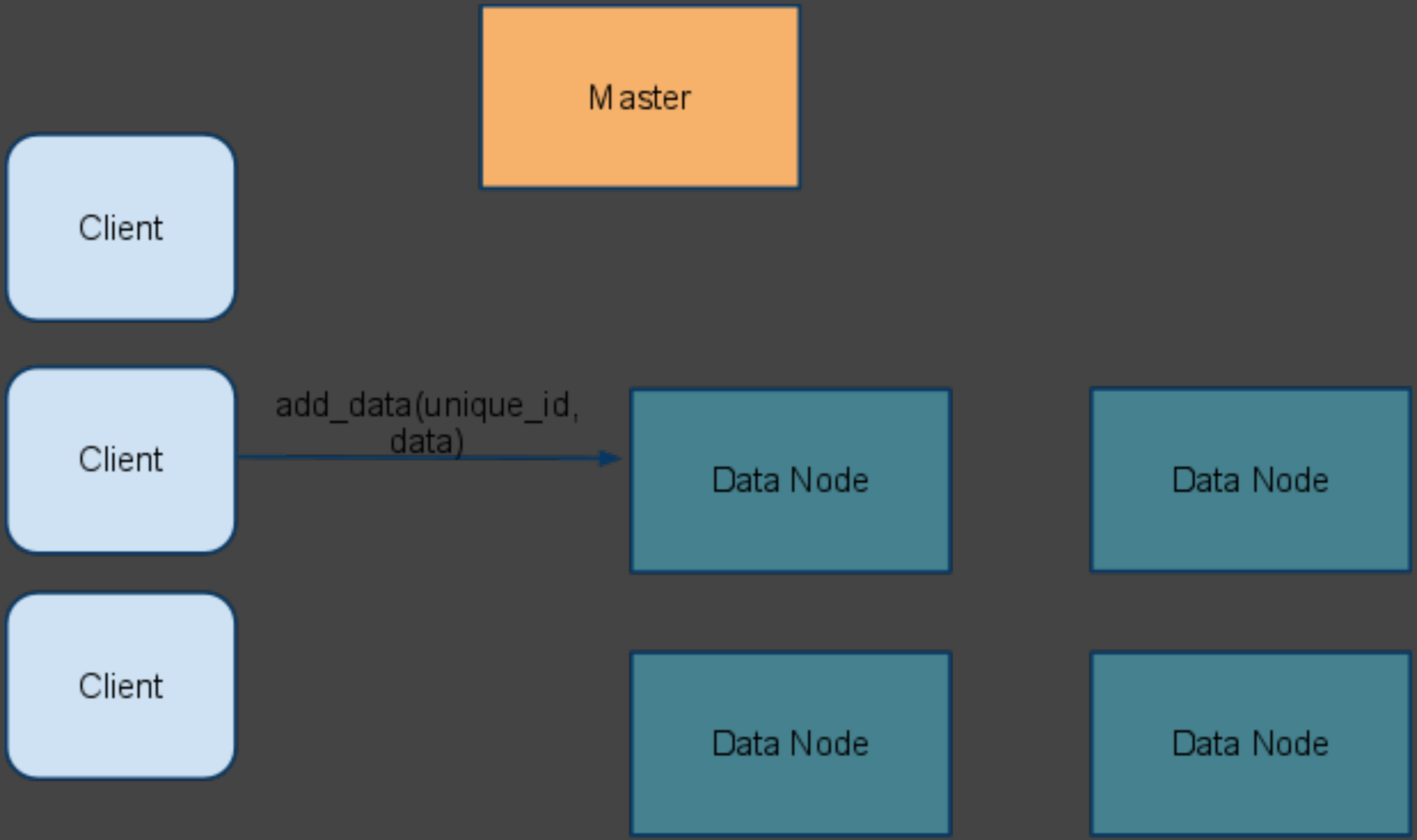
- Single master to simplify design
- Multiple data nodes that store queue data
- Multiple concurrent clients performing enqueue/dequeue operations
- Logging and replication for durability

Implementation - Begin Enqueue



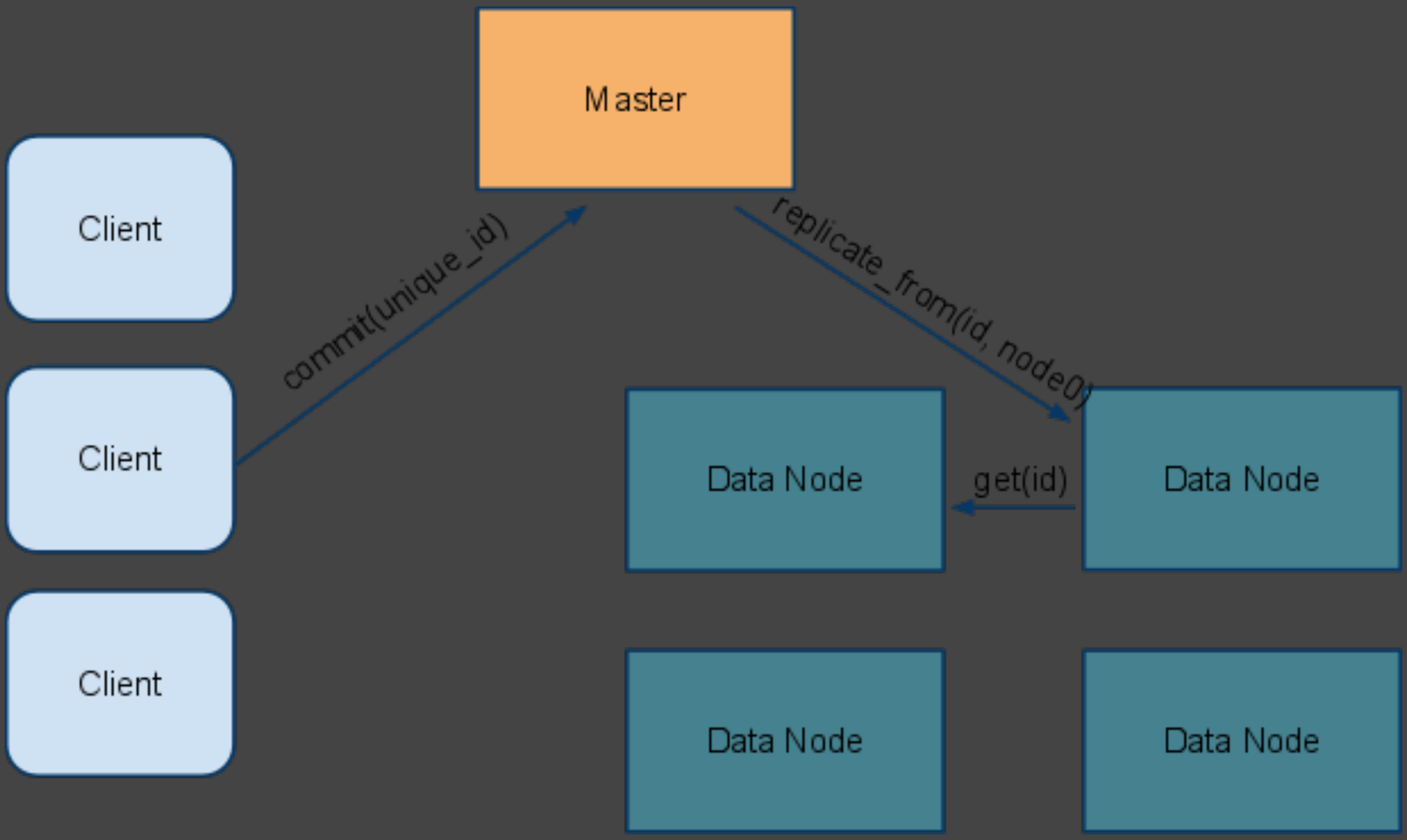
1. Client asks master for a node and ID

Implementation - Store data



2. Client stores/removes the (ID, data) pair on node

Implementation - Commit

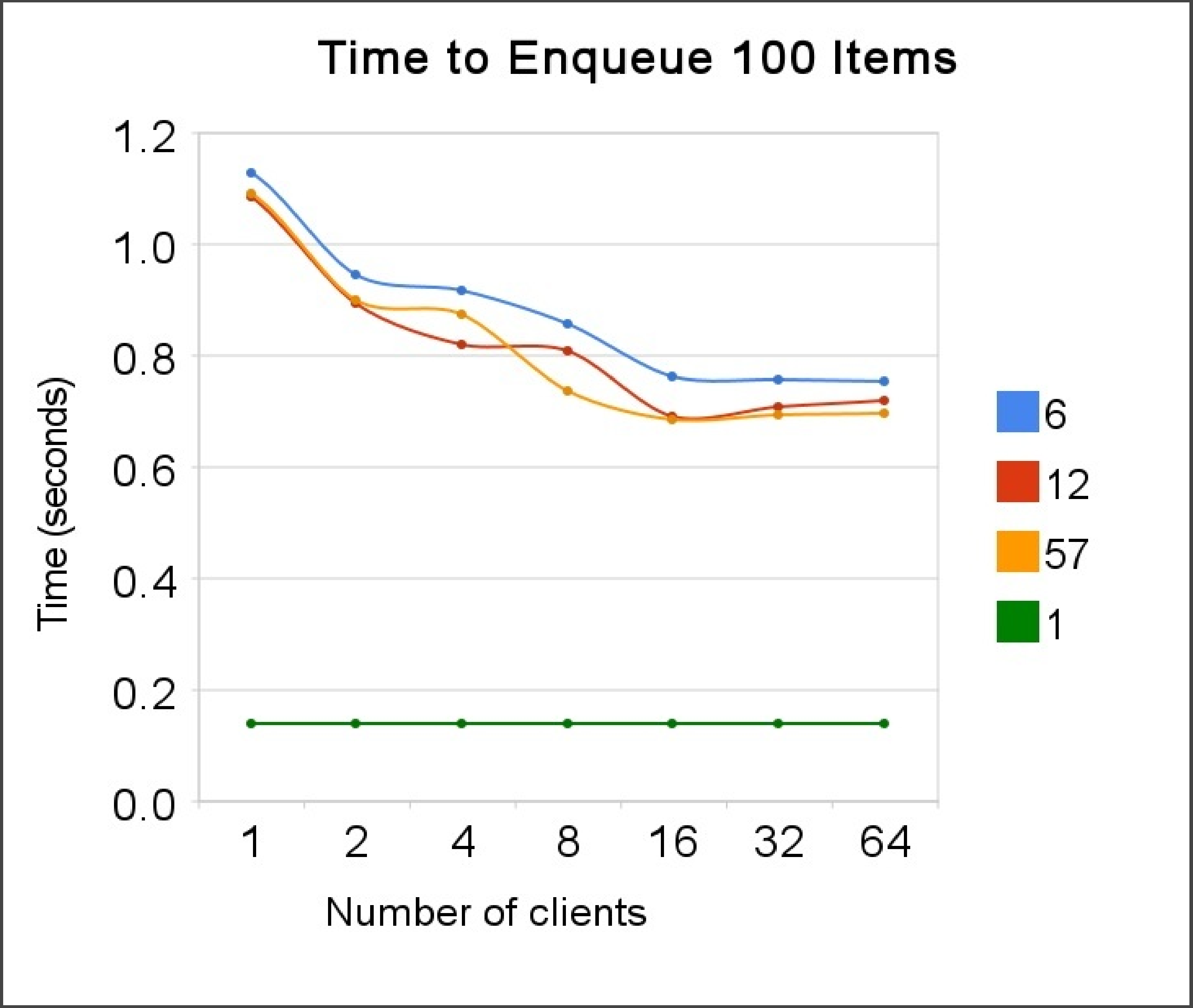


3. Client notifies the master of successful store/remove.
4. Master adds/removes item from logical queue

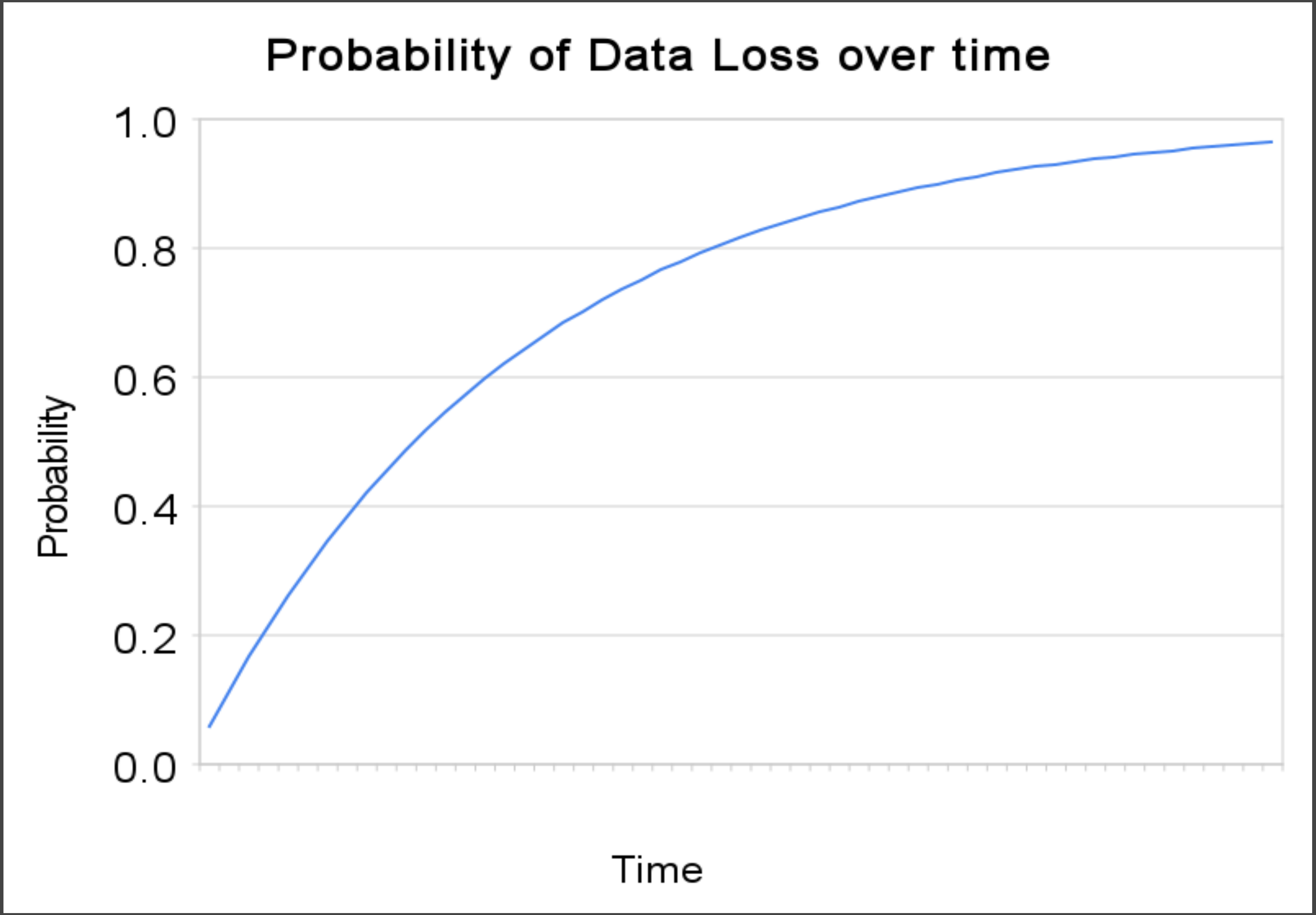
Evaluation

1. Throughput
   - Number of clients
   - Cluster size
2. Persistence
   - Churn
3. Expected error from FIFO
   - Number of clients

Evaluation - Throughput



Evaluation - Durability



Conclusion

Distributed Queue is slower but:

- not by much
- gets better with parallelism
- provides larger scale and dynamic scalability
- durability from failures

and...

- can provide guarantees about ordering
- can account for errors on client