# SILT
# S3 Integrity Locks & Transactions

Hussam Abu-Libdeh
CS 6464 Project Demo

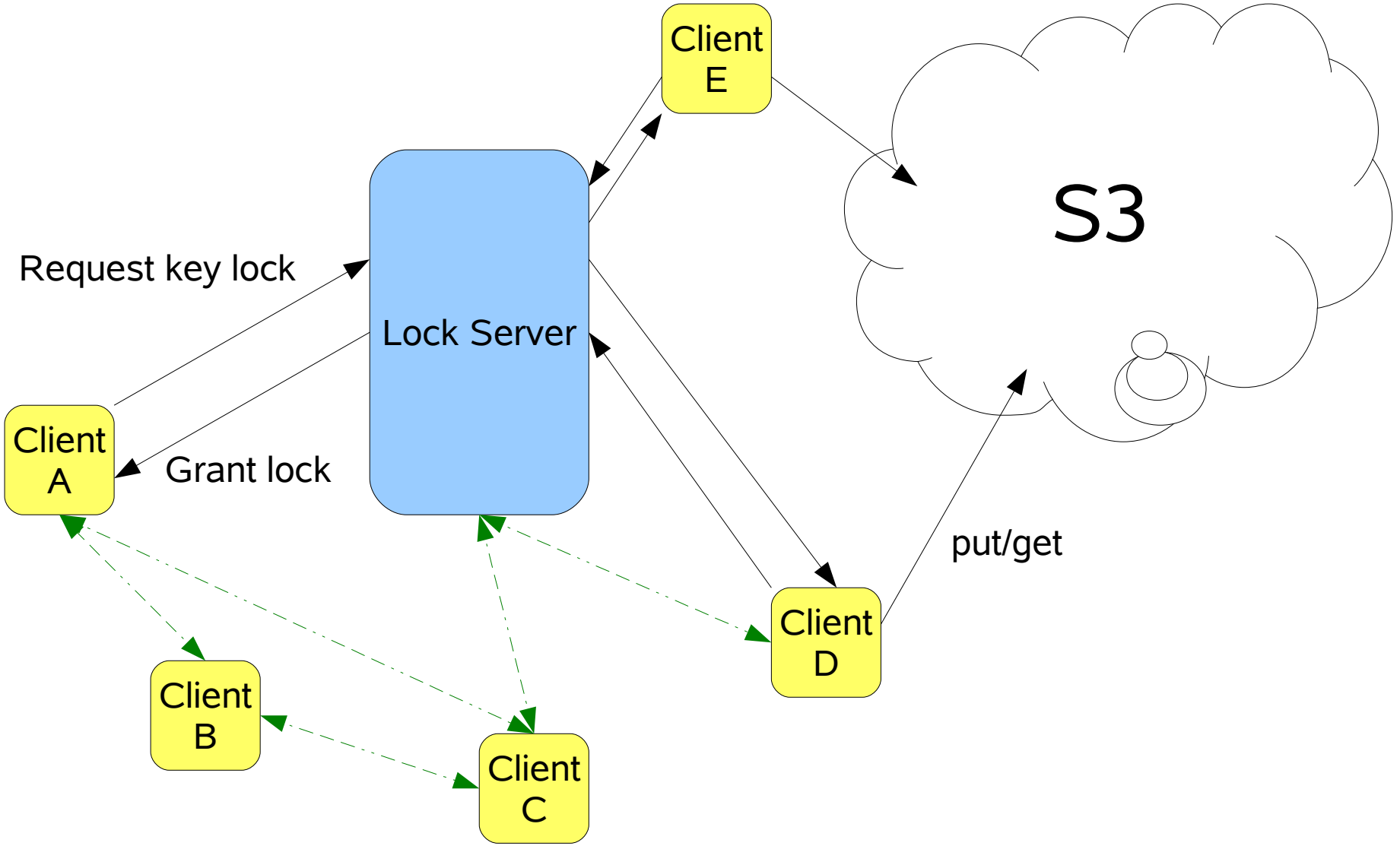
# Motivation

- Amazon S3
  - Highly available, scalable, reliable, inexpensive
- However,
  - No tangible trust guarantees
  - How can we trust that an S3 “get” corresponds to a previous “put” ?
  - Not necessarily because of maliciousness
    - Can we build a collaborative storage system on S3 ?
  - S3 only guarantees eventual consistency

# Main Idea

- Maintain key-value hashes outside S3
- Clients use an external locking server to serialize key access
- The clients and lock server gossip latest key-hash pairs
- Locks can expire
- Inconsistent values elevated to user
- Locking server restores state via gossip

# Pictorially



# Components

- Global node tracker
  - Assumed to never fail
  - Just tells us which nodes joined the system
  - Can be replaced by using a multicast channel
- Locking Server
  - Can fail
  - Grants/Expires locks
  - Receives key hashes when clients release write locks
  - Issues key hash timestamps
- Clients
  - Can fail
  - Request/Release locks
  - Compute/Check hashes

# Integrity

- Users manually compute hash after reading or writing data
- Users responsible for repairing inconsistencies
  - Report, fix, or try again
- Key hashes timestamped by the lock server
- Clients and lock server gossip hash values
- New hashes also piggybacked on lock requests/releases

# Locking

- Read and Read/Write locks
- Lock requests can span multiple keys
- Locks are granted in order received
- Locks have expiration time
  - If exceeded, automatically removed by locking server
  - Written data marked as inconsistent

# Transactions

- Implemented with locks
- Transaction writes are “put” on temporary keys
  - On commit introduce a redirection from original keys to temporary keys
  - On abort remove temporary keys
- Checkpoints implemented similarly