iRODS Logical Locking

Alan King
Software Developer
iRODS Consortium

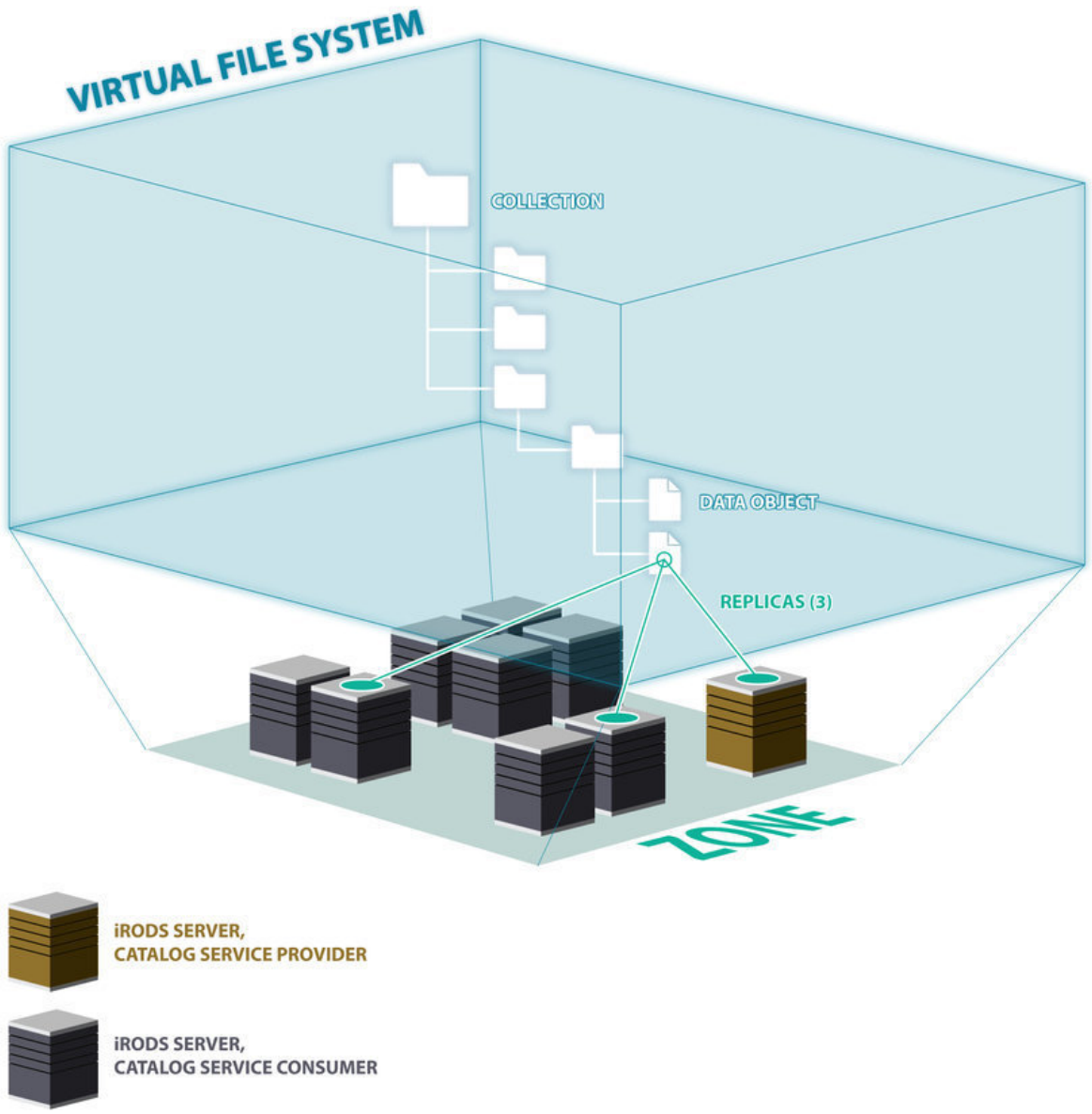
June 8-11, 2021
iRODS User Group Meeting 2021
Virtual Event

...but first, a story

Why did it take so long?

- 2019: The Missing Link
  - https://slides.com/irods/ugm2019-technology-update#/9
- 2020: Logical Locking
  - https://slides.com/irods/ugm2020-technology-update#/12
- 2021: Logical locking is real, but we had to change everything inside

What is a Data Object? What is a Replica?



Data Object: a logical representation of data that maps to one or more physical instances (*Replicas*) of the data at rest in *Storage Resources*

Replica: an identical, physical copy of a *Data Object*

How do we create and modify data in iRODS?

iRODS supports a POSIX-like interface for opening, writing, and closing.

Every data movement operation in iRODS boils down to:

Open replica, *move* data to replica, *close* replica

Most users deal with high-level APIs (put, cp, repl, etc.) which are built using these lower-level APIs.

Policy can be invoked as a result of an operation which can and often is itself a data-moving operation.

How do we define the state of data? What is Truth?

Truth: The latest data known to be "correct"; or, how the data "should" be

Replica status: The state of the data as it relates to the physical storage, the catalog, and the Truth

Good: Data is at rest, matches the catalog, and reflects the Truth

Stale: Data is at rest, but does not meet all criteria for being Good

- It may not match what is in the catalog: data transfer errors, mismatched checksum, corruption, etc.
- It may not reflect the Truth (anymore): more-recently-written data understood as being correct exist (may or may not differ!)
- Note: stale does not necessarily mean the data are incorrect, it is just at least not known to reflect the Truth

Why Locking? Concurrency in a Distributed System

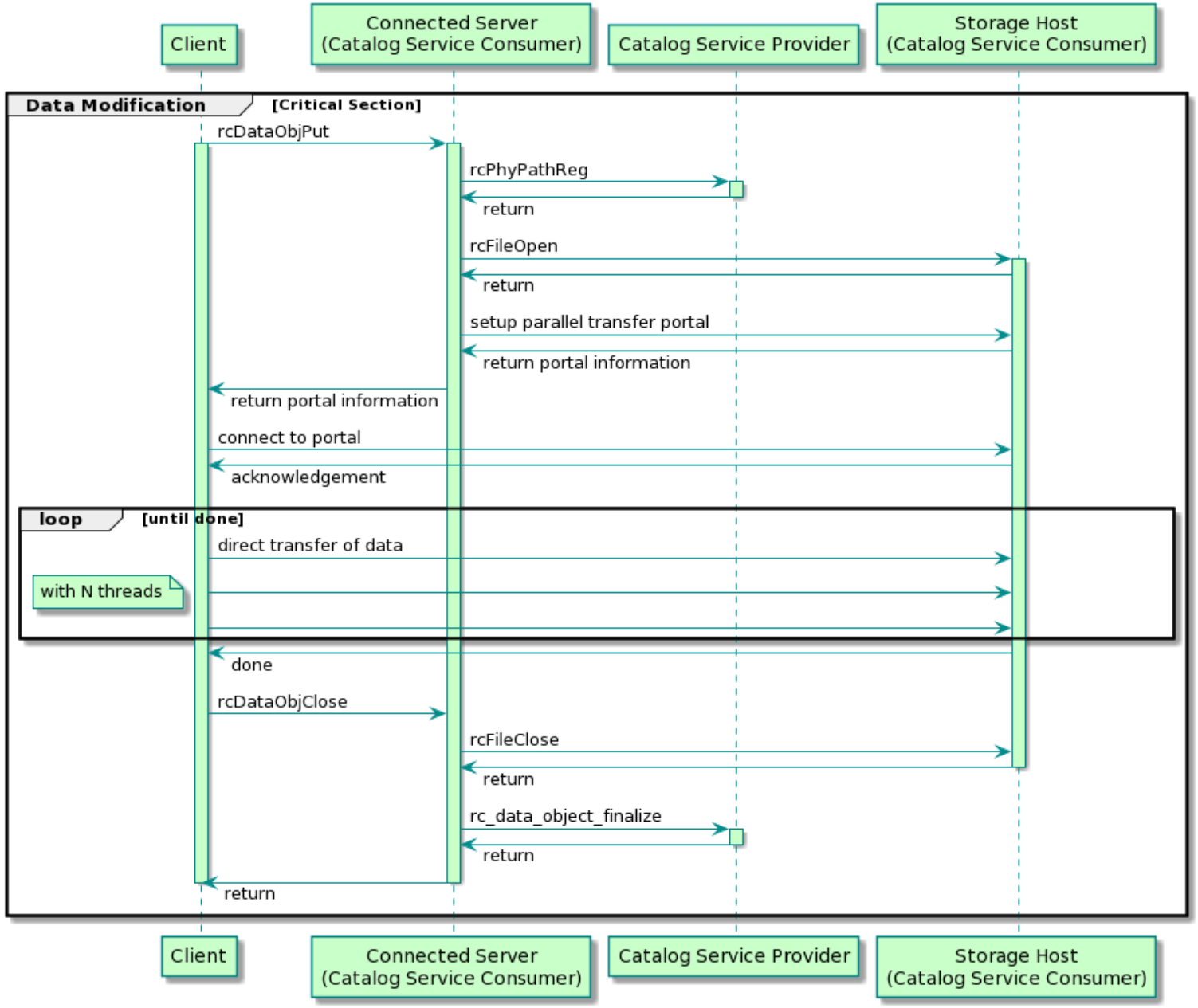
Uncoordinated, concurrent writing to a single replica can lead to *data corruption*.

Uncoordinated, concurrent writing to multiple replicas of the same data object causes *truth corruption*.

Uncoordinated, concurrent operation execution can lead to *policy violations*.

All of these things endanger our understanding of the state of the data, which is how we know that our data is stored and cataloged safely.

Example: iput



Data Corruption: Intermediate Replicas

Uncoordinated, concurrent writing to a single replica can lead to data corruption.

Problem: In-flight replicas can be opened and modified concurrently by multiple agents in an uncoordinated fashion, and the catalog does not reflect the current, true state of the data.

Solution: Mark in-flight replicas as **intermediate** at open time and update the status at close to reflect the state of the replica

- Status of the replica is accurately represented in the catalog
- The system and users can take appropriate action based on whether or not the replica is at rest

Truth Corruption: Logical Locking

Uncoordinated, concurrent writing to multiple replicas of the same data object causes truth corruption.

Problem: It is unclear which replica for a given data object represents the Truth when multiple replicas are in flight at the same time.

Solution: Prevent opening any replica for a given data object when any one of the replicas opened for write.

- The opened replica is marked **intermediate**, as shown previously
- The other replicas are **write locked** which prevents any additional opens for read or write; it is clear which replica represents the Truth

Known Limitations/Trade-offs

Maintaining replica information implies catalog round-trips (time):

- Create: up to 3 (lock object, register replica, close/finalize)
- Open for write: 2 (lock object, close/finalize)

Additional system metadata in the catalog and in memory for in-flight data (space)

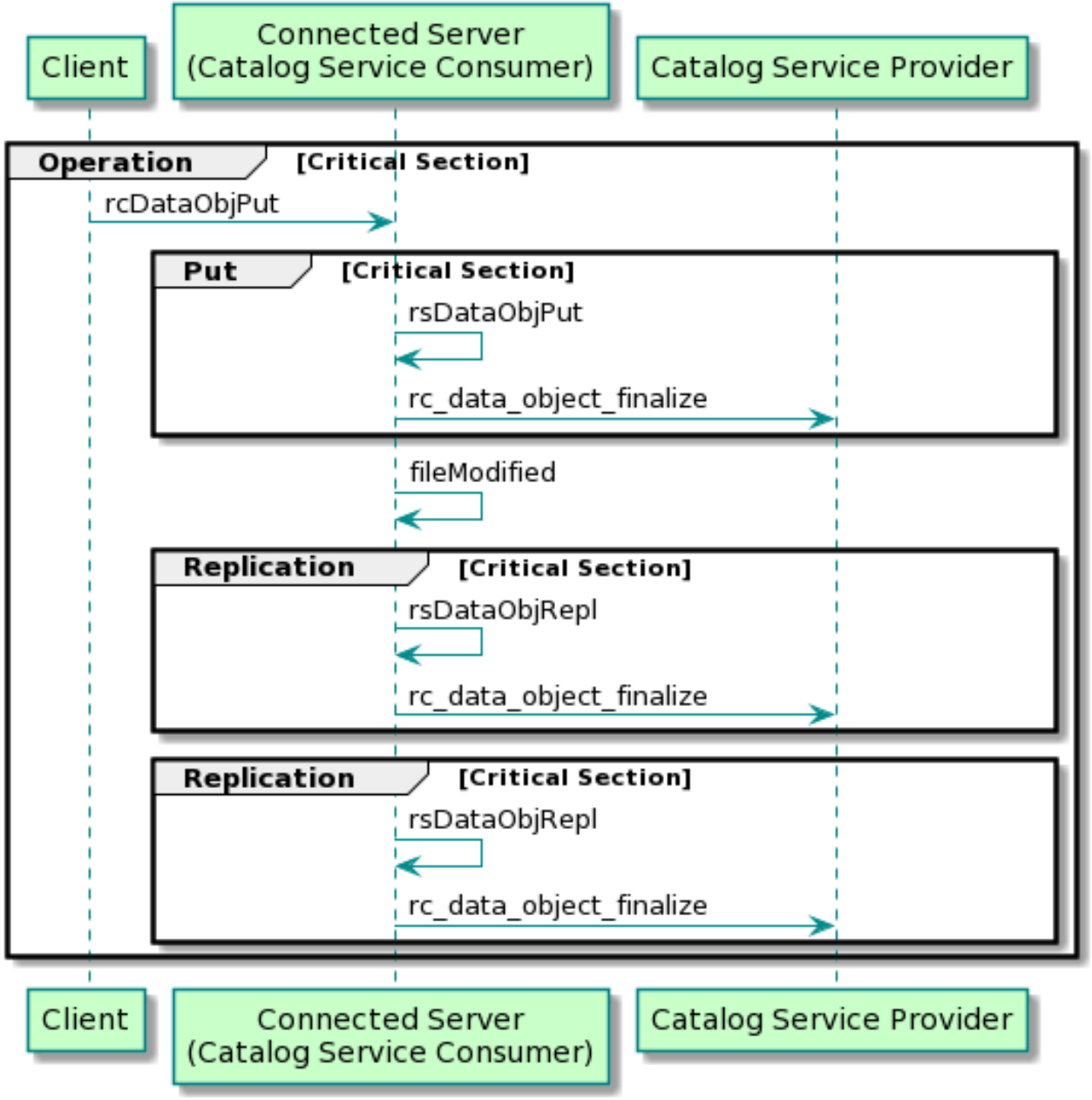
What write locks do NOT solve:

- Database race conditions
- Protection against rogue administrators

Locking is currently scoped by open/close, not by operation

Future Work

Example: iput to a replication resource hierarchy



Policy Violation: Operation Locking (name pending)

Uncoordinated, concurrent operation execution can lead to policy violations.

Problem: If a data-modifying operation is impacted by policy execution which leads to other data-modifying operations, other concurrent, uncoordinated data-modifying operations can lead to violations in said policy.

Solution: Keep data object locked over the lifetime of any given data-modifying operation.

Future Work: Read Locks

Problem: Modifying or unlinking objects which are being read.

Solution: Extend ILL to allow multi-reader, single-writer access

Implementation details:

- Disallows open for write, unlink, or rename while locked
- Maintain list of agent PIDs/hostnames holding locks in the catalog
- Last agent to release lock will be responsible for unlocking the data object by restoring the replica states
- Agents need to be more self-aware with respect to locking to avoid deadlocks (also useful for operation locking); possibly use irods::replica_access_table

Future Work: Lock Checker

Problem: Agent holding open descriptors is responsible for locking/unlocking. Killed agents can leave objects stuck in locked or intermediate state. Leaves admins to identify and modify replica states so they can be healed.

Solution: Asynchronous server task which checks for locked data objects and checks to see if the listed agent(s) are still running.

Schedules asynchronous unlocking of data objects which are no longer owned by a living agent.

Thanks for listening