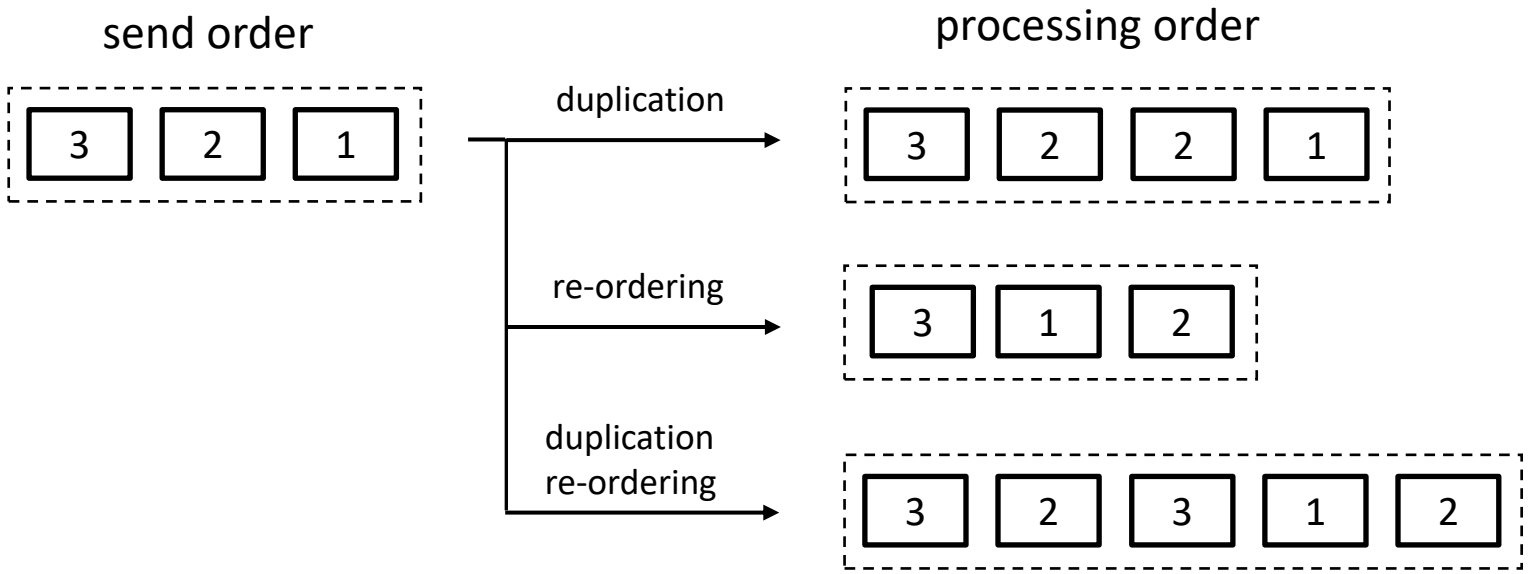
Checking safety in Exactly-Once

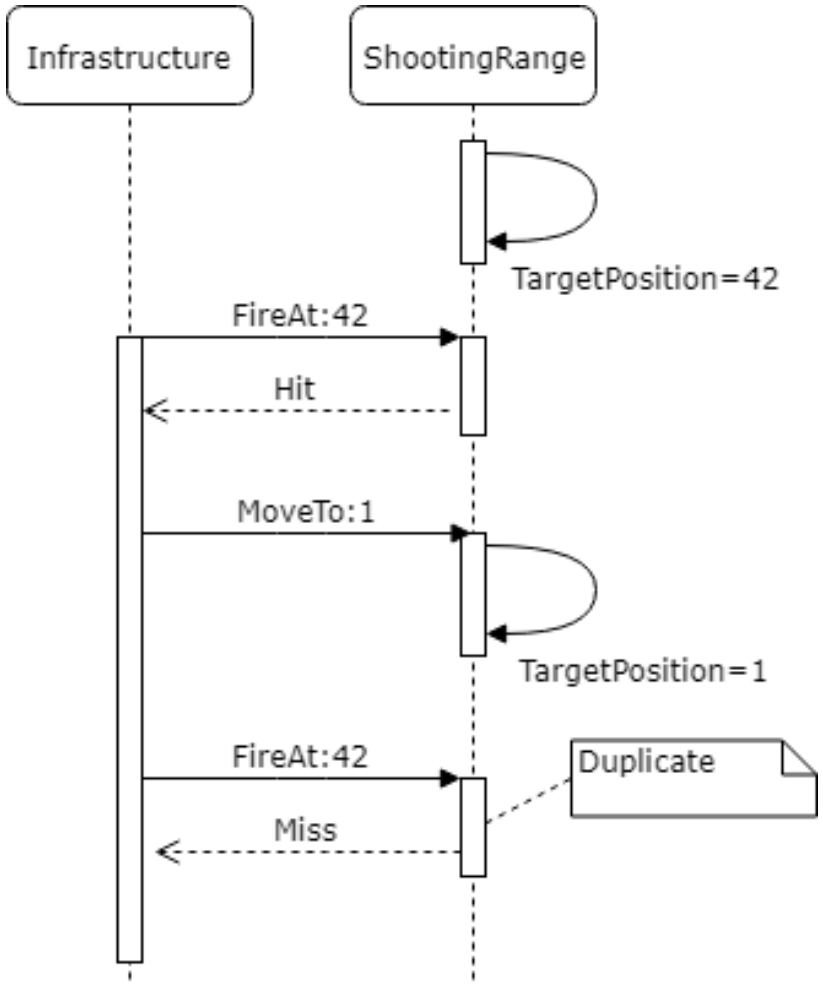
Tomek Masternak, Szymon Pobiega

The Problem

De-facto standard

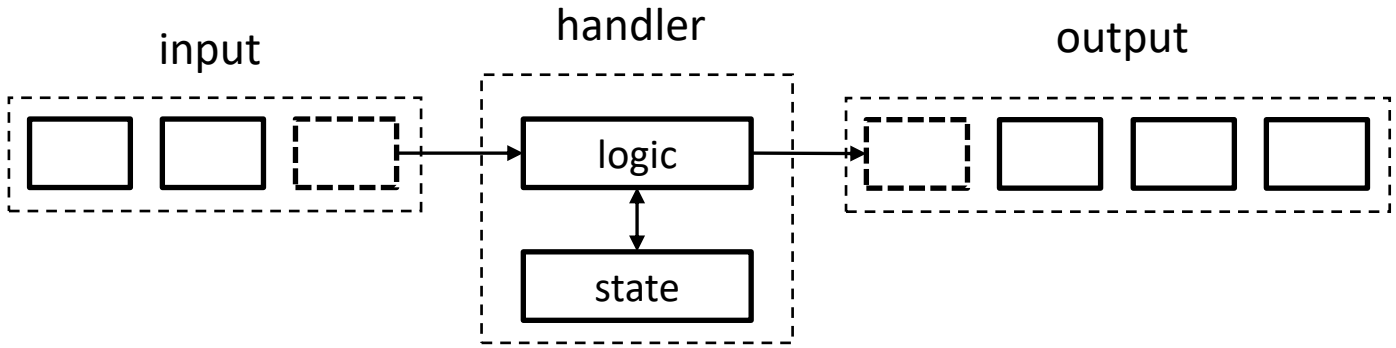


Idempotency might not be tricky



System model

- Persistent messaging infrastructure
- Set of message handlers with distinct piece of state



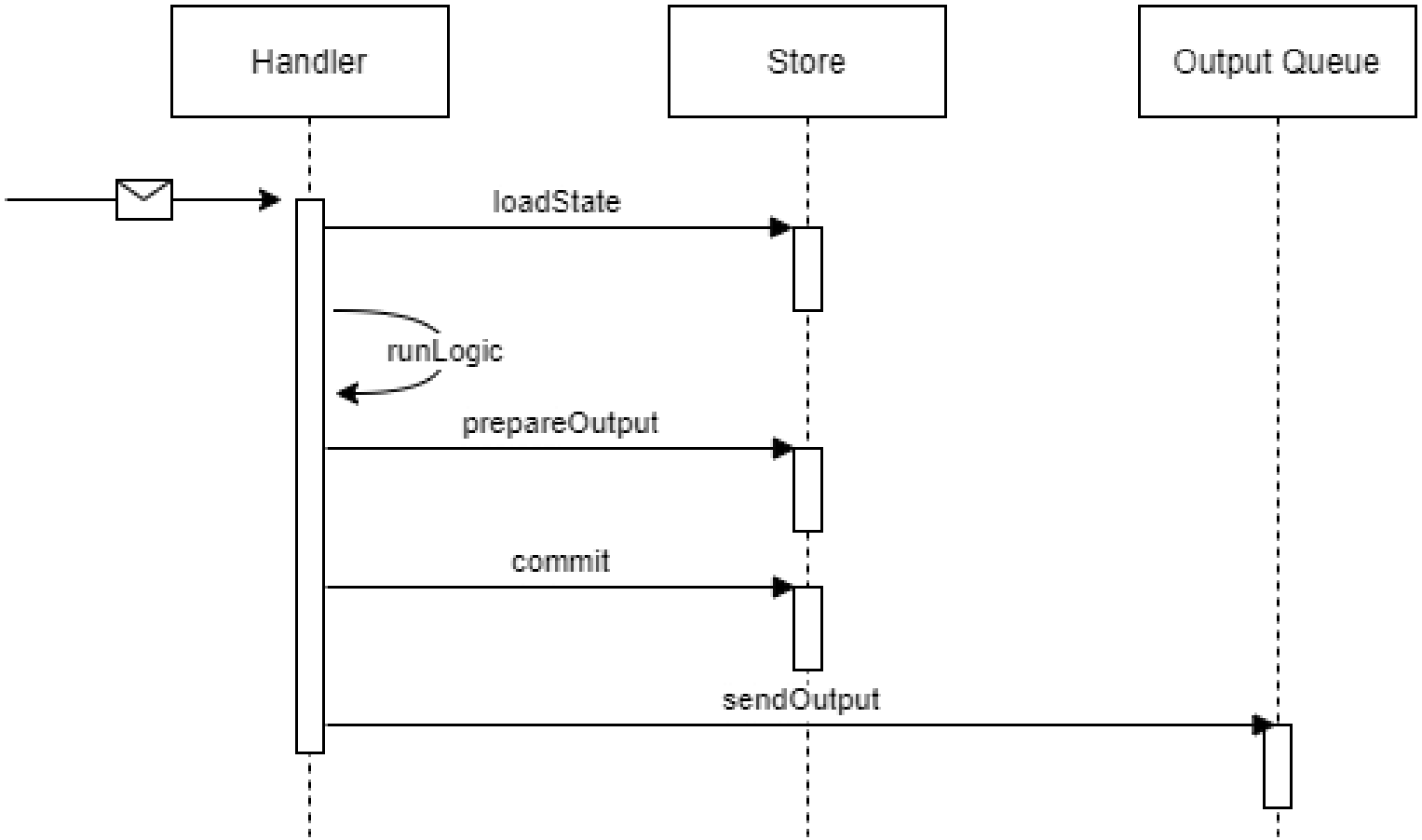
Goal: A bit stronger guarantees

Observable side-effects should correspond to some serial execution of input messages with atomic commit guarantee between business storage and the output queue

Implementation

- A library
- Azure Functions hosting with built-in messaging options
- Azure CosmosDB
  - Optimistic concurrency control
  - Exactly-Once data in separate partition
  - Session consistency model
- High-performance scenarios are out of scope

General idea



TLA+ Model

Model checking goals

- Happy path is easy, what about failure recovery
- Check safety
  - Atomic commit
  - Consistent side-effects
- Memory model for the main and infrastructural state storages

Not in the model

- Infrastructure clean-up logic
- Lock-leases that lower the chance of concurrency collisions
- Exponential back-offs and processing retries

Handler Processes

```
begin
MainLoop:
    while TRUE do
    LockInMsg:
        await ~ IsEmpty(inputQueue);

        with m \in inputQueue do ···
        end with;

        state := store;

        if(state.tx /= NULL) then
        Redo_OutboxCommit: ···
        Redo_StateCommit: ···
        end if;

    Process:
        outboxRecord := outbox[msg.id];

        if outboxRecord = NULL then ···
        StageOutbox: ···
        StateCommit: ···
        OutboxCommit: ···
        StateCleanup: ···
        end if;

    SendAndAck: ···
    end while;
end process;
```

State modelling

```
variables
    inputQueue = [id : MessageIds, dupId : DupIds],
    store = [history |-> <<>>, ver |-> 0, tx |-> NULL],
    outbox = [r \in MessageIds |-> NULL],
    outboxStagging = [t \in TxIdx |-> NULL],
    pendingOutbox = {},
    output = { },
    processed = { }
```

Termination

```
Termination == <>(/\ \A self \in Processes: pc[self] = "LockInMsg"
                 /\ IsEmpty(inputQueue))
```

Safety Invariants

```
AtMostOneStateChange ==
    \A id \in MessageIds : Cardinality(WithId(Range(store.history),id)) <= 1

AtMostOneOutputMsg ==
    \A id \in MessageIds : Cardinality(WithId(output, id)) <= 1

ConsistentStateAndOutput ==
    LET InState(id)  == CHOOSE x \in WithId(Range(store.history), id) : TRUE
        InOutput(id) == CHOOSE x \in WithId(output, id) : TRUE
    IN \A m \in processed: InState(m.id).ver = InOutput(m.id).ver

Safety == AtMostOneStateChange /\ AtMostOneOutputMsg /\ ConsistentStateAndOutput
```

Process failures

```
macro Fail() begin
    goto MainLoop;
end macro;

macro CommitState(state) begin
    if store.ver + 1 = state.ver then
        either
            store := state;
        or
            Fail();
        end either;
    else
        Fail();
    end if;
end macro;

macro CleanupState(state) begin
    if store.ver = state.ver then
        either
            store.tx := NULL || store.ver := store.ver + 1;
        or
            Fail();
        end either;
    else
        Fail();
    end if;
end macro;
```

Results

- Expected
  - Bugs in post-failure commits
  - Bugs in transaction id assignment logic

- Unexpected
  - Distilling the algorithm to it’s essentials
    - Realizing potential extensions
  - CosmosDB memory model challenges
  - Learning about additional possible failure modes

Validation

Coverage

Module	Action	Total	Distinct
exactly_once_none_atomic	Init	1	1
exactly_once_none_atomic	MainLoop	24 643 656	13 743 801
exactly_once_none_atomic	LockInMsg	10 322 633	4 542 192
exactly_once_none_atomic	Redo_OutboxCommit	5 310 156	2 765 302
exactly_once_none_atomic	Redo_StateCommit	7 470 660	4 306 915
exactly_once_none_atomic	Process	12 090 053	5 837 076
exactly_once_none_atomic	StageOutbox	3 653 858	2 139 607
exactly_once_none_atomic	StateCommit	5 334 532	3 248 889
exactly_once_none_atomic	OutboxCommit	3 393 456	1 887 241
exactly_once_none_atomic	StateCleanup	6 311 280	3 809 841
exactly_once_none_atomic	SendAndAck	11 165 916	6 513 929

Q/A

- Twitter
  - @Masternak
- Website
  - https://exactly-once.github.io