Formal Modeling and Analysis of Cassandra in Maude

Si Liu, Muntasir Raihan Rahman, Stephen Skeirik, Indranil Gupta, and José Meseguer

University of Illinois at Urbana-Champaign

Presented by Jinghan Sun

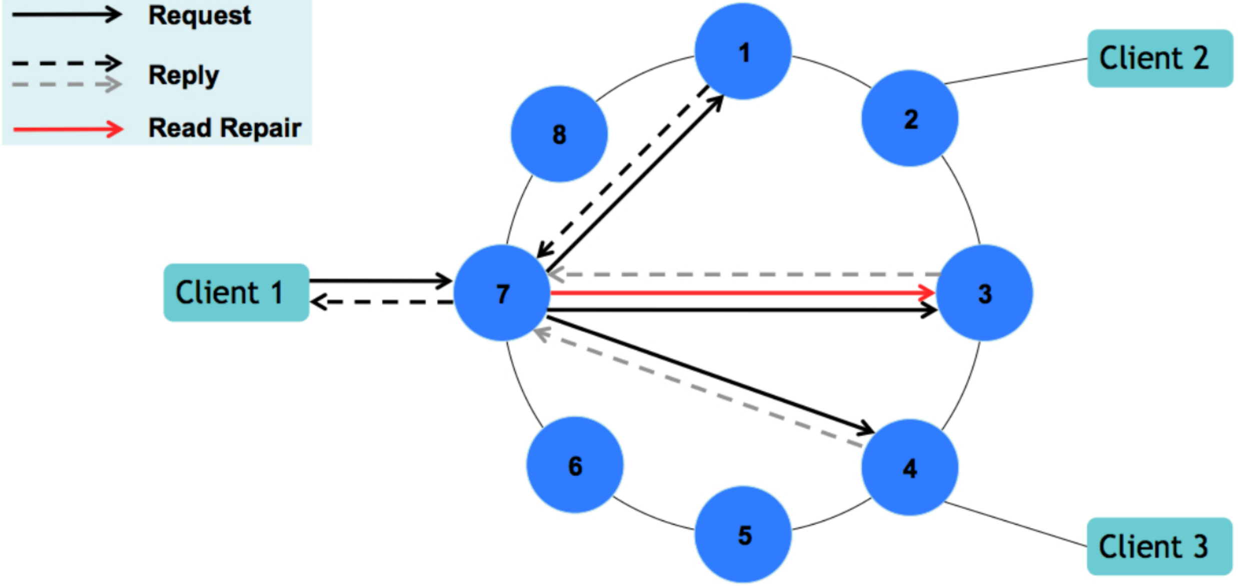
The paper in a nutshell

- Presented a formal model for the Cassandra key-value store using Maude
- Formally specified and model checked Cassandra’s consistency properties
- Cassandra
  - a scalable, fault-tolerant, and distributed NoSQL database
  - widely used in the industry, e.g. IBM, HP, Netflix, Facebook
- Formal analysis results
  - strong consistency can be violated:
    - WRITE(key, “**orange**”) = 1; WRITE(key, “**apple**”) = 1; READ(key) = “**orange**”.

Outline

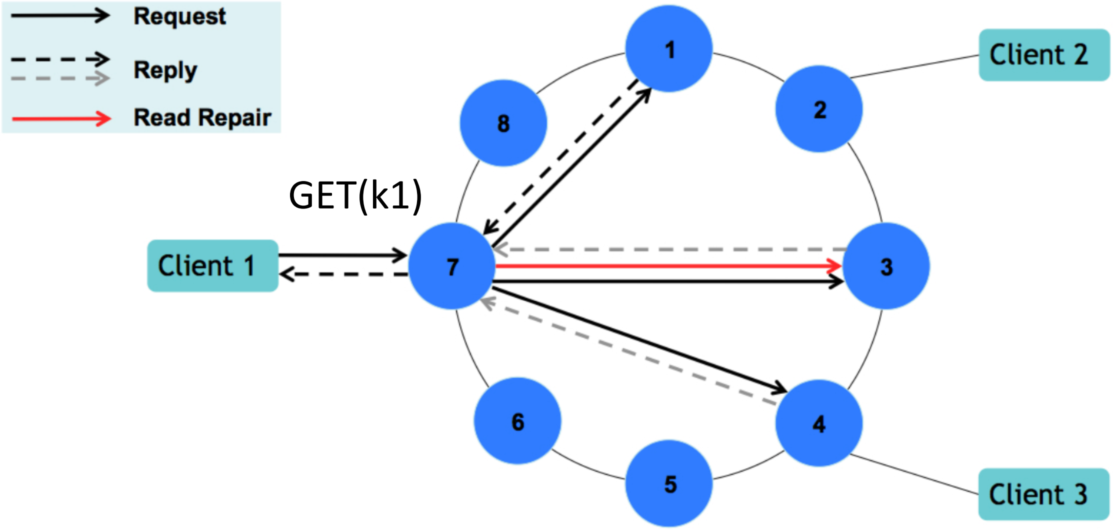
- **Background**
- Cassandra model in Maude
- Consistency model check
- Formal analysis results

Cassandra Overview



- Servers store key-value pairs (k,v)
- Each k-v pair repliacated at multiple servers
- Clients can read/write k-v pairs
- Tunable Consistency Levels
  - Client can specify how many replicas need to answer
  - One, Quorum, All
- An example system with 8 servers, 3 clients and a replication factor of 3

Cassandra Overview



Server	Key	Value	timestamp
1	k1	"red"	9.0
3	k1	"black"	10.0
4	k1	"red"	8.0

1. Client 1 sends a read request to its coordinator (server 7).
2. The coordinator forwards read request to replicas S1, S3, and S4.
3. Each replica responds with a non-deterministic delay (e.g. d(R1) < d(R4) < d(R3)).
4. The coordinator forwards the value back to client after N replicas respond (ONE: 1, Quorum: 2, All: 3). The copy with the latest timestamp is taken as the true one.
5. The coordinator issues a read repair to the replica with out-of-date value.

Outline

- Background
- **Cassandra model in Maude**
- Consistency model check
- Formal analysis results

Concurrent state in Maude (lec. 12a)

An object in a given state is represented as a term

$$\langle O : C \mid a_1 : v_1, \ldots, a_n : v_n \rangle$$

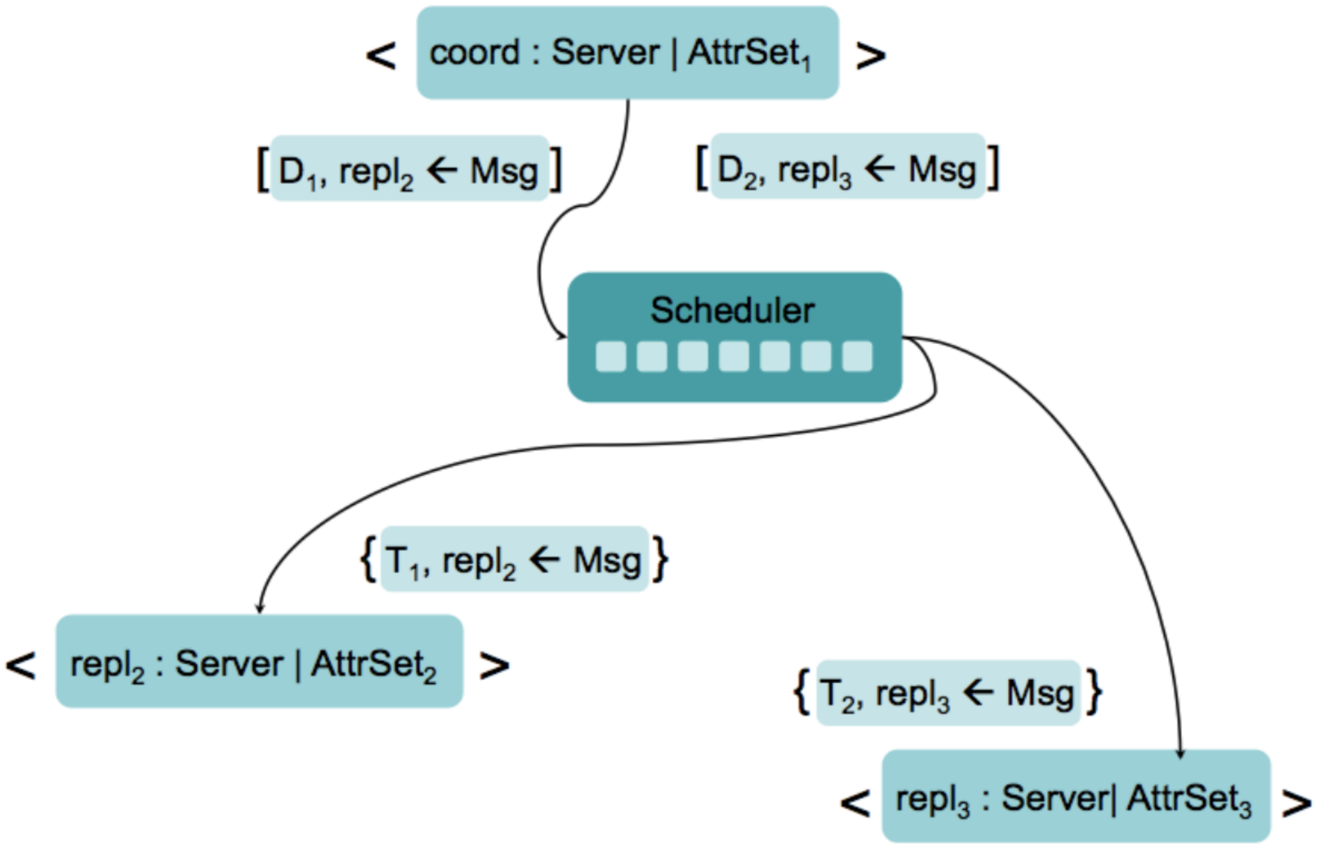
where O is the object's name or identifier, C is its class, the a_i's are the names of the object's attribute identifiers, and the v_i's are the corresponding values.

The syntax of messages is user-definable; it can be declared in Full Maude by message operator declarations. In our example by:

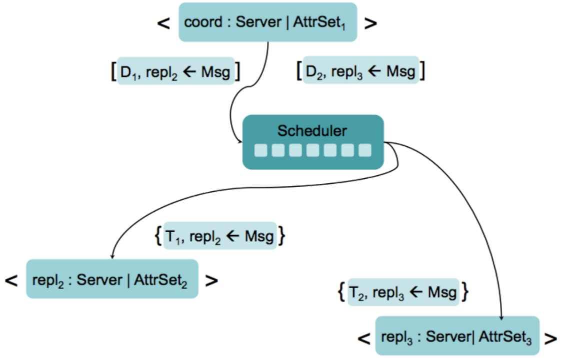
```
msg (to _ : _ from (_,_)) : Oid Int Oid Int -> Msg .
```

Cassandra Model in Maude

- Components: clients, servers, scheduler and messages



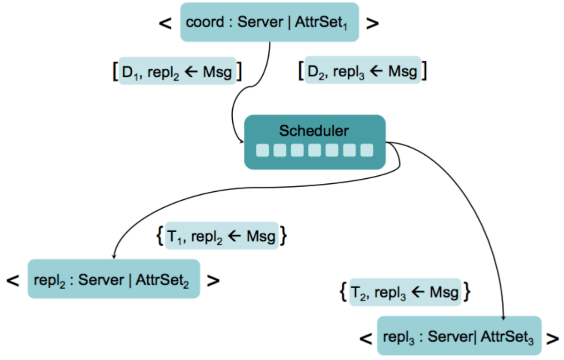
Cassandra Model in Maude



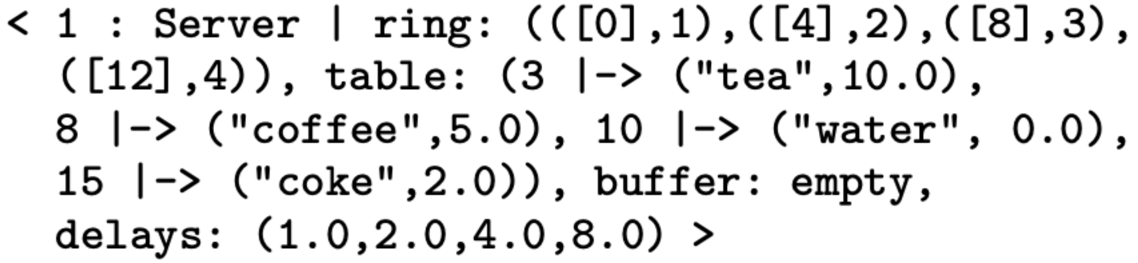
- Client:
    - *op* **coord :_** : Address -> Attribute . --- *coordinator*
    - *op* **store :_** : List{Value} -> Attribute . --- value of incoming messages
    - *op* **requestQueue :_** : List{Elt} -> Attribute . --- requests ready to send out
    - *op* **lockedKey :_** : Set{Key} -> Attribute . --- set of locked keys
    - *op* **pendingQueue :_** : List{Elt} -> Attribute . --- pending requests

```
< 100 : Client | coord: 1, store: nil,
  requestQueue: (r1 r2), lockedKey: empty,
  pendingQueue: nil >
```

Cassandra Model in Maude



- Server:
  - *op* **ring :_** : Set{RingPair} -> Attribute .   --- *set of tokens*
  - *op* **table :_** : Table -> Attribute .   --- a table of k-v pairs
  - *op* **buffer :_** : LocalRequestQueue -> Attribute .   --- cached requests to replica
  - *op* **delays :_** : Set{Delay} -> Attribute .   --- a set of delays for outgoing msgs



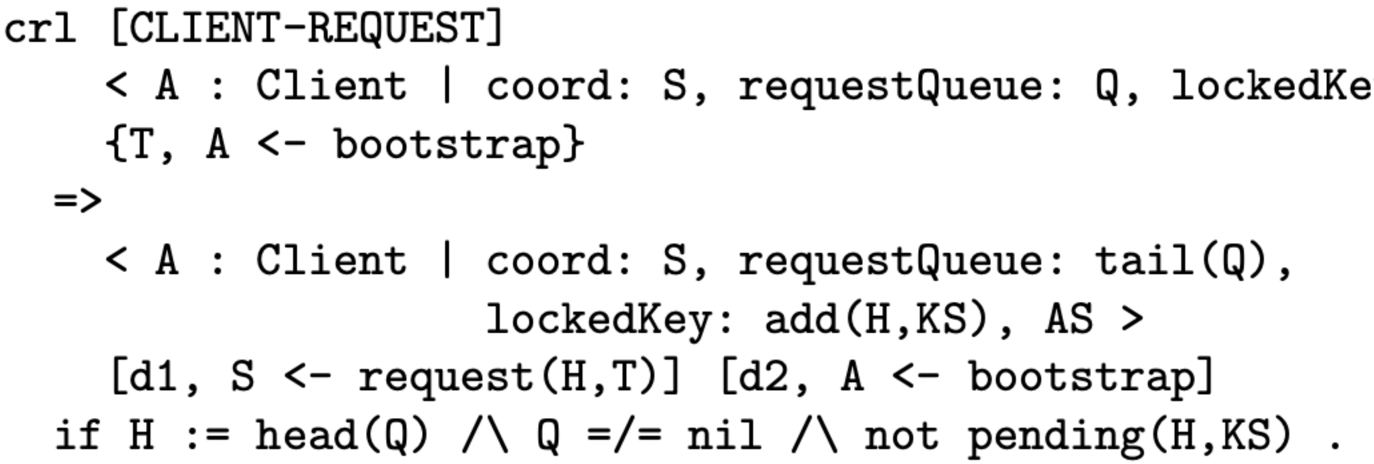
```
< 1 : Server | ring: (([0],1),([4],2),([8],3),
  ([12],4)), table: (3 |-> ("tea",10.0),
  8 |-> ("coffee",5.0), 10 |-> ("water", 0.0),
  15 |-> ("coke",2.0)), buffer: empty,
  delays: (1.0,2.0,4.0,8.0) >
```

Formalizing Reads and Writes

- Four Stages:
  1. Client-to-Coordinator
  2. Coordinator-to-Replica
  3. Replica-to-Coordinator
  4. Coordinator-to-Client and Read Repair

Formalizing Reads and Writes

- Stage 1: Client-to-coordinator
  - client trigger by the *bootstrap* msg
  - adds key to the *KS* and checks if we block the current request *H*
  - generates a message to the coordinator *coord* and a self-triggered *bootstrap* msg



```
crl [CLIENT-REQUEST]
    < A : Client | coord: S, requestQueue: Q, lockedKey: KS, AS >
    {T, A <- bootstrap}
  =>
    < A : Client | coord: S, requestQueue: tail(Q),
                   lockedKey: add(H,KS), AS >
    [d1, S <- request(H,T)] [d2, A <- bootstrap]
  if H := head(Q) /\ Q =/= nil /\ not pending(H,KS) .
```

T: global time
d1, d2: message delays
AS: a set of attributes
pending: op to check if key locked

Formalizing Reads and Writes

- Stage 2: Coordinator-to-replica
  - the coordinator S receives the request *ReadRequestCS*
  - *S* updates the request *buffer*
  - generates messages to all the replicas holding the value of key *K*
  - the auxiliary function *replicas* returns a set of replica addresses.

```
crl [COORD-FORWARD-READ-REQUEST] :
    < S : Server | ring: R, buffer: B, delays: DS, AS >
    {T, S <- ReadRequestCS(ID,K,CL,A)}
  =>
    < S : Server | ring: R, buffer: insert(ID,fac,CL,K,B),
                   delays: DS, AS > C
  if generate(ID,K,DS,replicas(K,R,fac),S,A) => C .
```

ID: client request id
K: key
CL: consistency level
A: client
fac: replication factor

Formalizing Reads and Writes

- Adding delays to messages
  - The coordinator non-deterministically selects a message delay D for each out-going request.

```
rl [GENERATE-READ-REQUEST-1] :
   generate(ID,K,(D,DS),(A’,AD’),S,A)
 =>
   generate(ID,K,(D,DS),AD’,S,A)
   [D, A’ <- ReadRequestSS(ID,K,S,A)] .

rl [GENERATE-READ-REQUEST-2] :
   generate(ID,K,DS,empty,S,A) => null .
```

ID: request id
K: key
DS: a set of delays
AD: a set of replica addrs
S: addr of the coordinator

Outline

- Background
- Cassandra model in Maude
- **Consistency model check**
- Formal analysis results

Consistency Models

- Strong consistency model
  - each read returns the value of the last write that occurred before that read
- Read-your-writes
  - all writes performed by a client are visible to its subsequent reads
- Eventual consistency model
  - eventually all reads to a key will return the last updated value if no new updates are made to the key

Model checking using Maude

- LTL (linear temporal logic) model checking
  - The semantics of state propositions is defined by

```
ceq statePattern |= prop = b if cond .
```

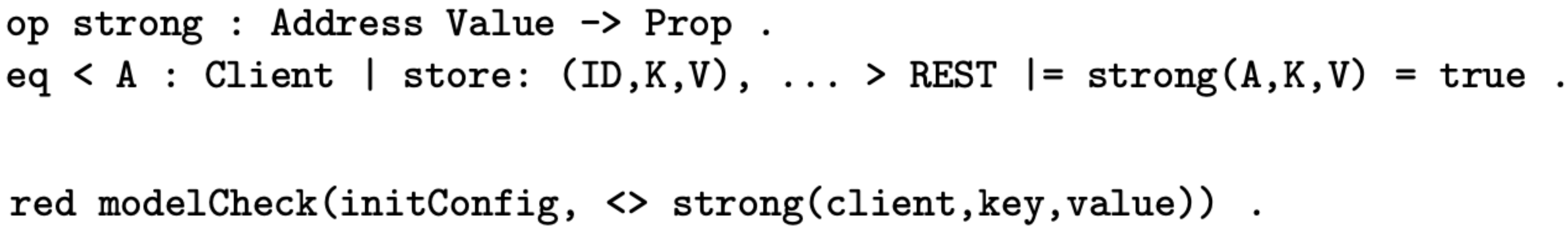
    - *prop* evaluates to *b* in states that are instances of statePattern when the condition *cond* holds
  - Model checking command

```
red modelCheck(t, φ) .
```

    - checks whether the temporal logic formula ϕ (state propositions and temporal logical operators) holds starting from the initial state t
  - Logical operators
    - Boolean connectives: **True**, **False**, **~** (negation), **/**, **\\/**, **->** (implication)
    - Temporal operators : **[]** (“always”), **<>** (“eventually”), and **U** (“until”).

Formal Consistency Models

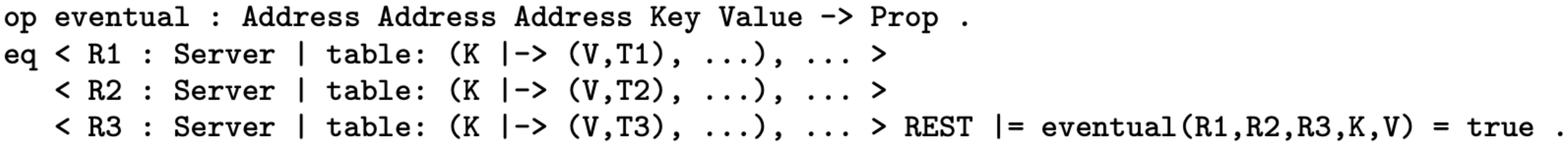
- Strong consistency
  - proposition strong(client, key, value)
  - holds true if we can match the value V returned by the subsequent read on key K in client A’s local store with that in the preceding write



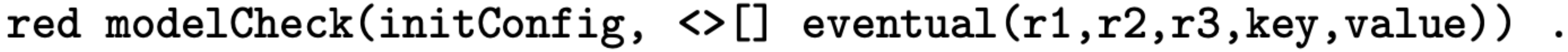
```
op strong : Address Value -> Prop .
eq < A : Client | store: (ID,K,V), ... > REST |= strong(A,K,V) = true .

red modelCheck(initConfig, <> strong(client,key,value)) .
```

Formal Consistency Models

- Eventual consistency
  - proposition eventual(r1,r2,r3,key,value)
  - holds true if we can match the value V on key K in the subsequent (or the last) write with those in the local tables of all replicas R1, R2 and R3.



```
op eventual : Address Address Address Key Value -> Prop .
eq < R1 : Server | table: (K |-> (V,T1), ...), ... >
   < R2 : Server | table: (K |-> (V,T2), ...), ... >
   < R3 : Server | table: (K |-> (V,T3), ...), ... > REST |= eventual(R1,R2,R3,K,V) = true .
```



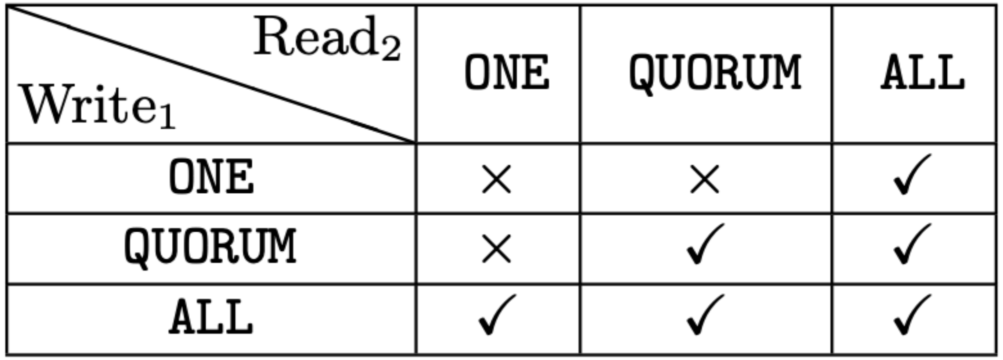
```
red modelCheck(initConfig, <>[] eventual(r1,r2,r3,key,value)) .
```

Outline

- Background
- Cassandra model in Maude
- Consistency model check
- **Formal analysis results**

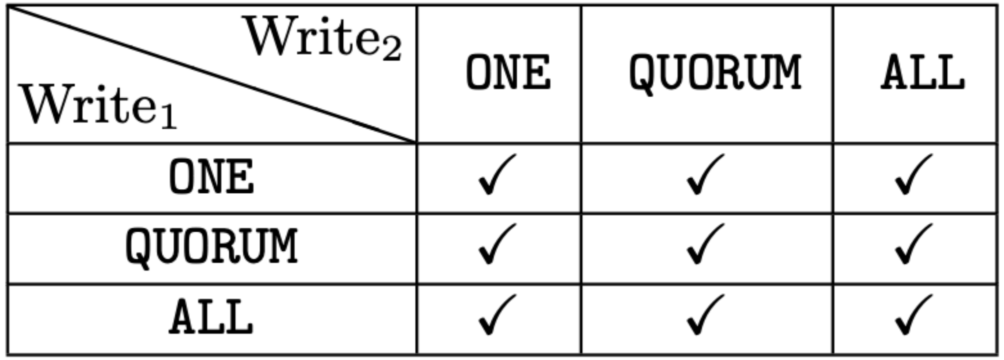
Formal Analysis of Consistency with One Client

- One client, 3 replicas, 3 different consistency levels
- The client issues a write request followed by a read on same key
- The two requests could have different consistency levels



Read$_2$ Write$_1$	`ONE`	`QUORUM`	`ALL`
`ONE`	×	×	✓
`QUORUM`	×	✓	✓
`ALL`	✓	✓	✓

Strong Consistency Property

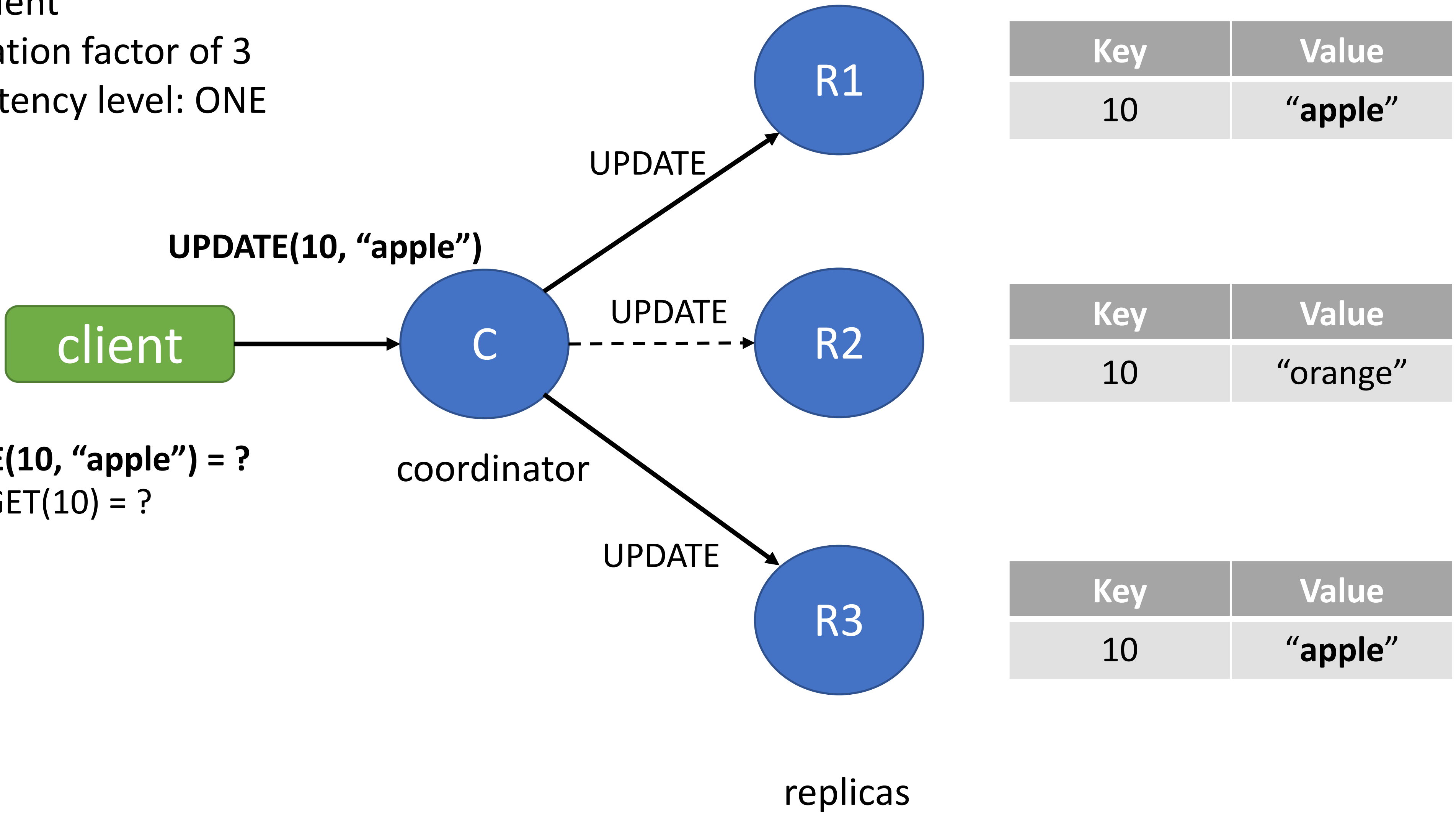


Write$_2$ Write$_1$	`ONE`	`QUORUM`	`ALL`
`ONE`	✓	✓	✓
`QUORUM`	✓	✓	✓
`ALL`	✓	✓	✓

Eventual Consistency Property

A Counterexample

- one client
- replication factor of 3
- consistency level: ONE



Key	Value
10	“**apple**”

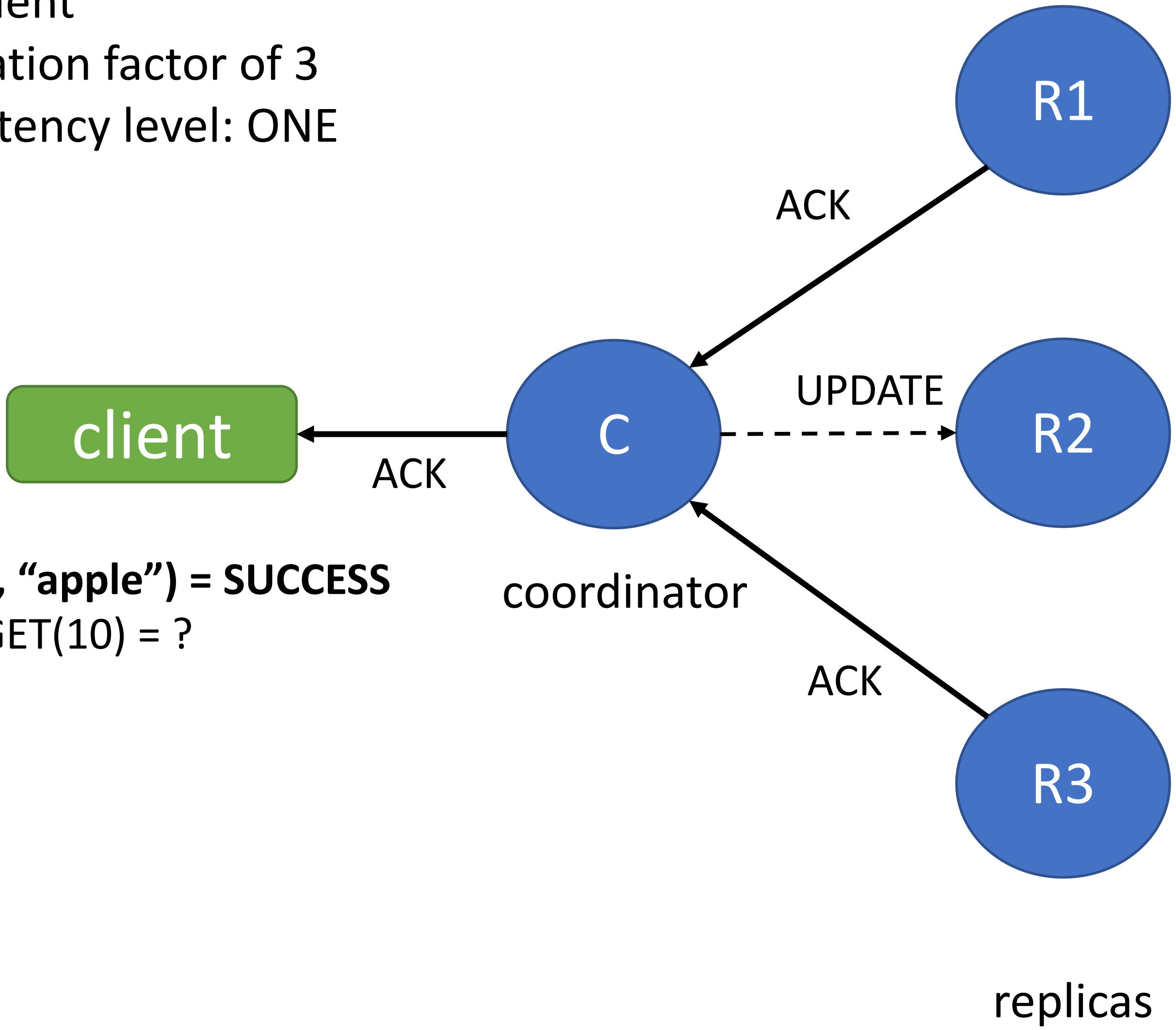
Key	Value
10	“orange”

Key	Value
10	“**apple**”

UPDATE(10, “apple”) = ?
GET(10) = ?

A Counterexample

- one client
- replication factor of 3
- consistency level: ONE



Key	Value
10	**“apple”**

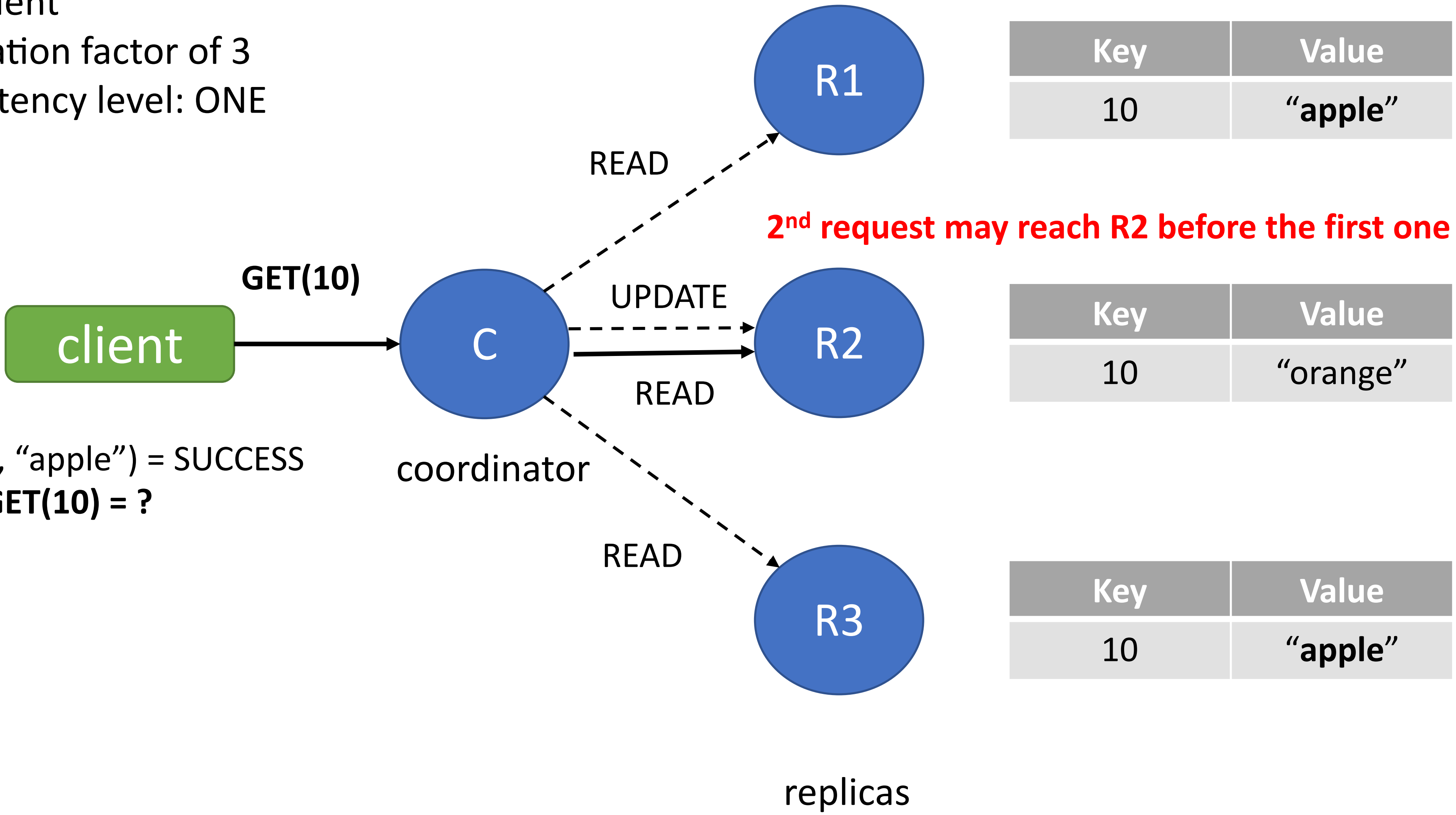
Key	Value
10	“orange”

Key	Value
10	**“apple”**

UPDATE(10, “apple”) = SUCCESS
GET(10) = ?

A Counterexample

- one client
- replication factor of 3
- consistency level: ONE



2nd request may reach R2 before the first one

Key	Value
10	“**apple**”

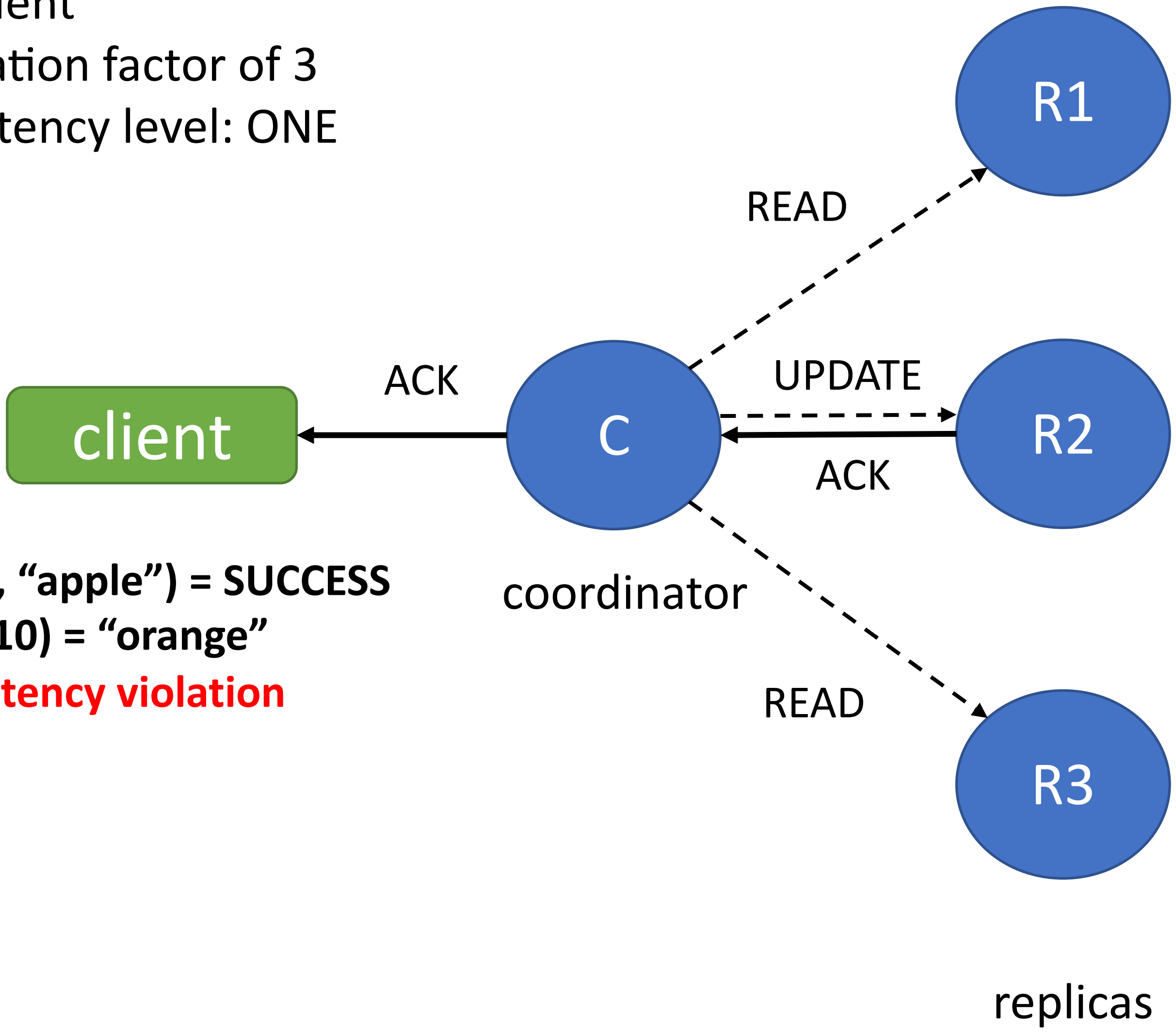
Key	Value
10	“orange”

Key	Value
10	“**apple**”

UPDATE(10, “apple”) = SUCCESS
GET(10) = ?

A Counterexample

- one client
- replication factor of 3
- consistency level: ONE



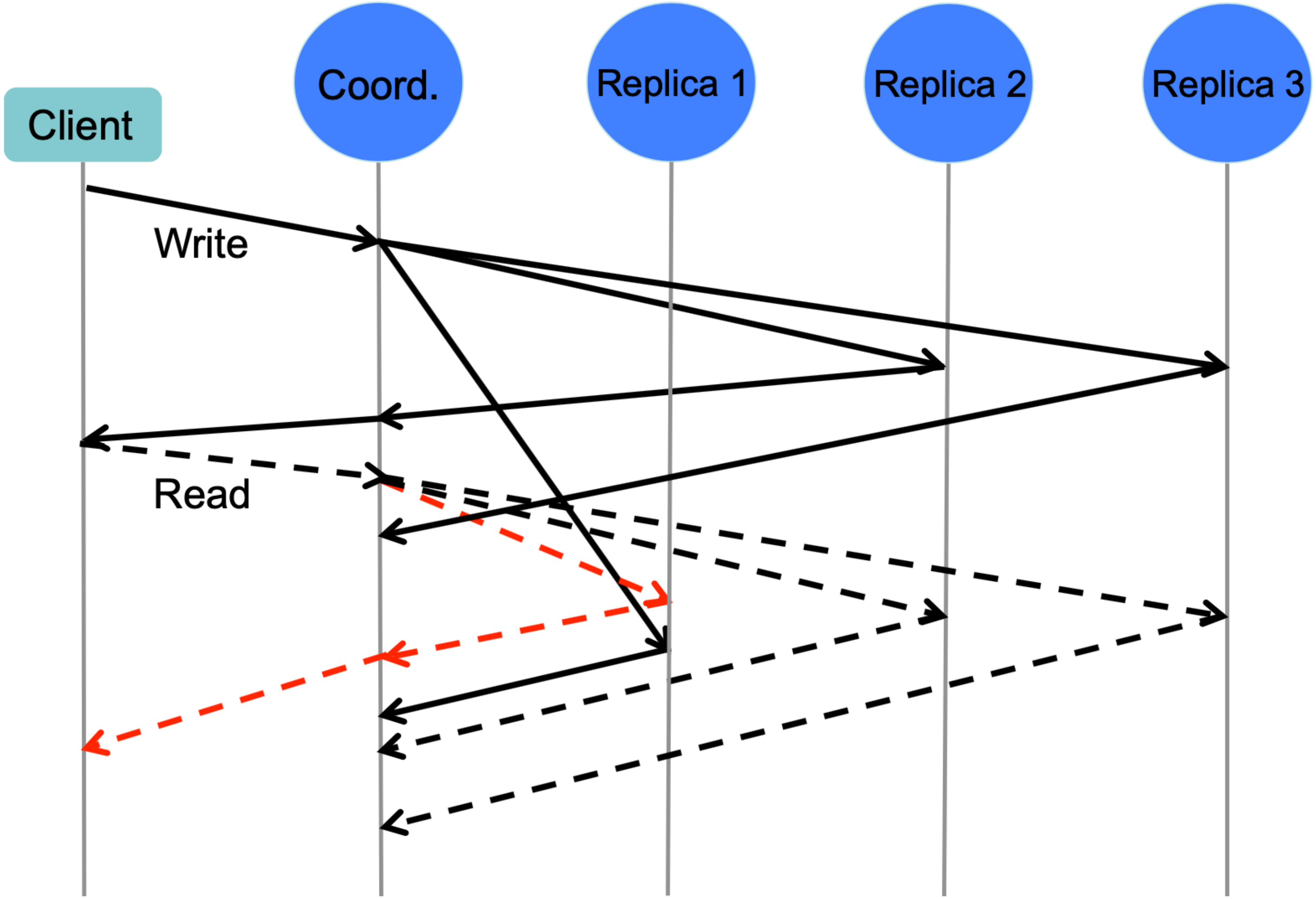
UPDATE(10, “apple”) = SUCCESS
GET(10) = “orange”
consistency violation

Key	Value
10	“**apple**”

Key	Value
10	“orange”

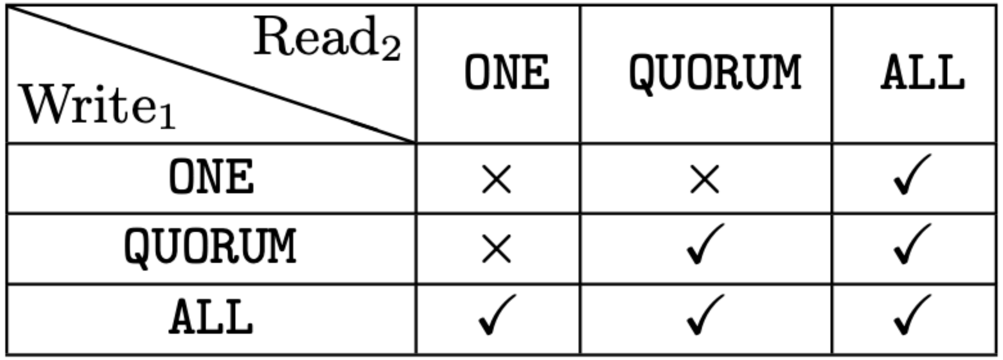
Key	Value
10	“**apple**”

A Strong Consistency Violation for Consistency Level Combination (One,One)

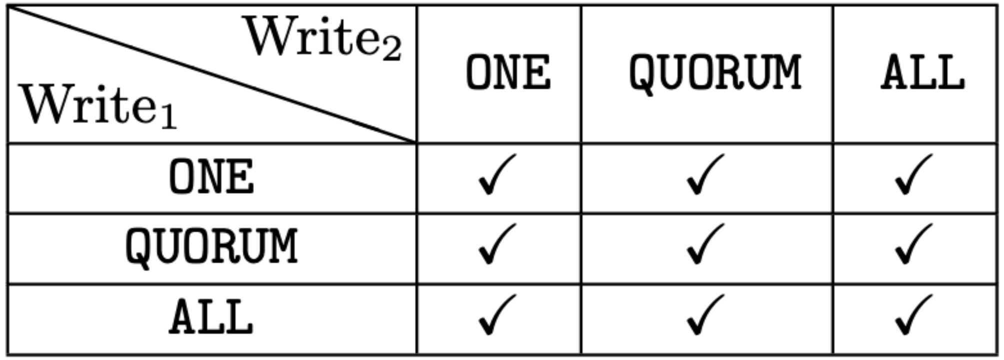


Formal Analysis of Consistency with One Client

- One client, 3 replicas, 3 different consistency levels
- The client issues a write request followed by a read of on same key
- The two requests could have different consistency levels



Read$_2$ Write$_1$	ONE	QUORUM	ALL
ONE	×	×	✓
QUORUM	×	✓	✓
ALL	✓	✓	✓



Write$_2$ Write$_1$	ONE	QUORUM	ALL
ONE	✓	✓	✓
QUORUM	✓	✓	✓
ALL	✓	✓	✓

- Strong consistency with one client depends on the combination of consistency levels
- Eventual consistency with one client holds for all combinations

Summary

- Presented a formal model for the Cassandra key-value store using Maude
    - formal models for clients, servers, schedulers and messages
    - formalized read and write requests
- Formally specified and model checked Cassandra’s consistency properties
    - strong consistency and eventual consistency
- Formal analysis of consistency properties
    - showed that strong consistency can be violated

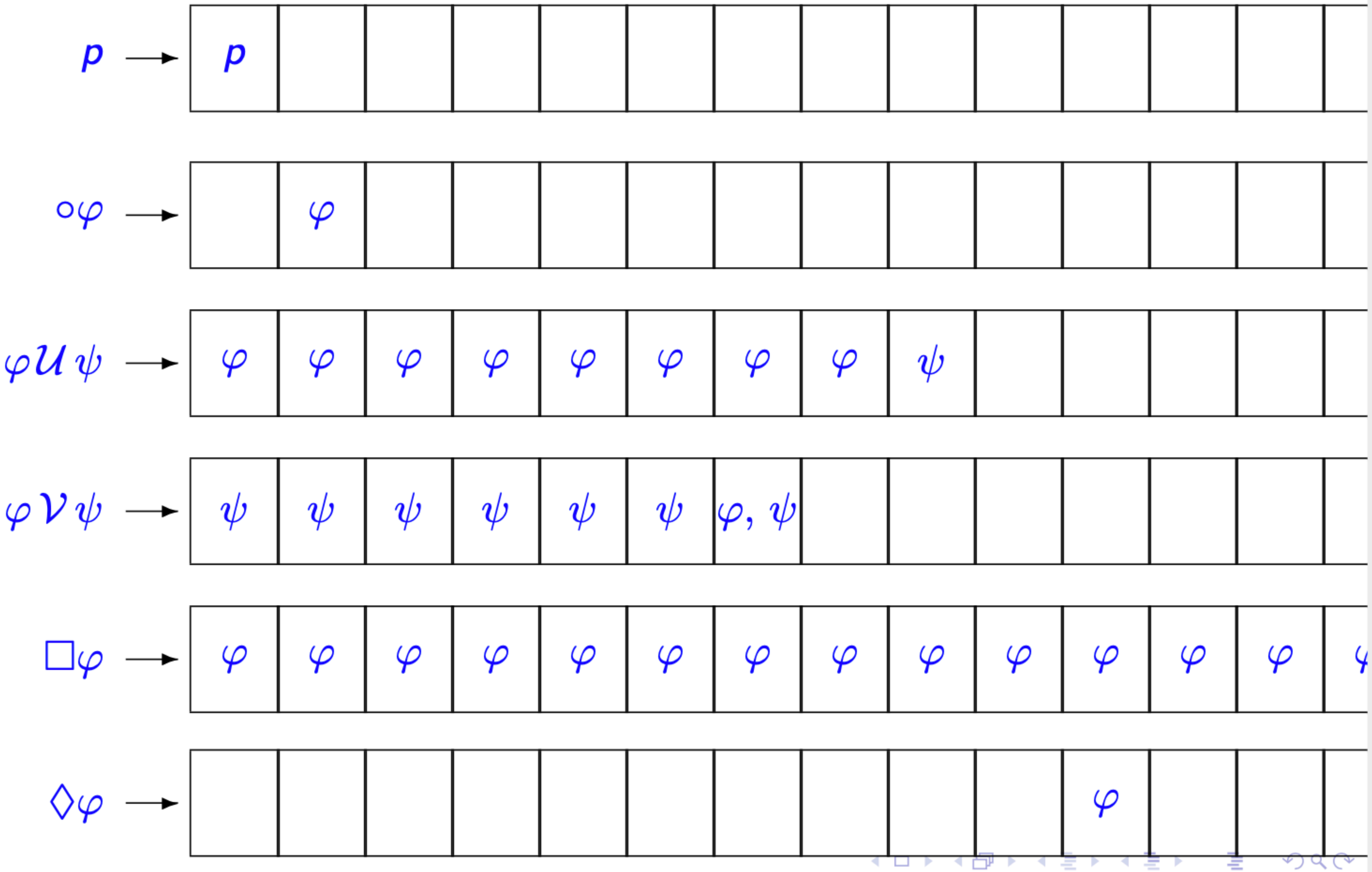
Backup Slides

Linear Temporal Logic - Syntax

$$\begin{aligned} \varphi \;::=\; & \boldsymbol{p} \mid (\varphi) \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \\ \mid\; & \circ\varphi \mid \varphi\,\mathcal{U}\,\varphi' \mid \varphi\,\mathcal{V}\,\varphi' \mid \Box\varphi \mid \Diamond\varphi \end{aligned}$$

- $\boldsymbol{p}$ – a propostion over state variables
- $\circ\varphi$ – “next”
- $\varphi\mathcal{U}\varphi'$ – “until”
- $\varphi\mathcal{V}\varphi'$ – “releases”
- $\Box\varphi$ – “box”, “always”, “forever”
- $\Diamond\varphi$ – “diamond”, “eventually”, “sometime”

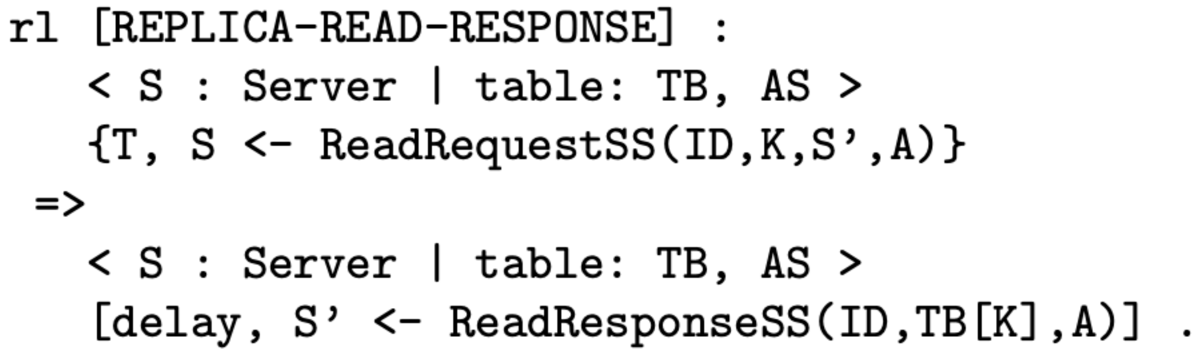
LTL Semantics: The Idea



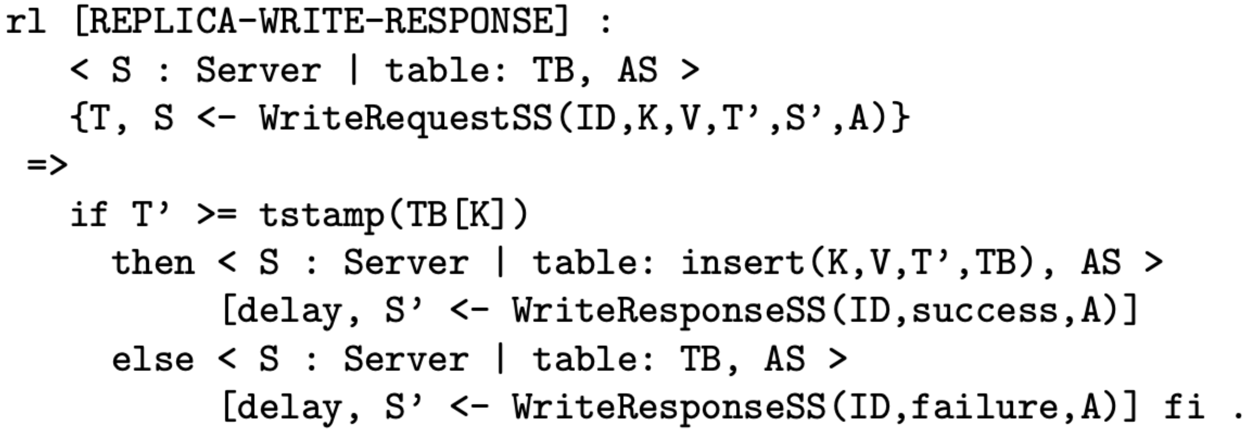
$p \longrightarrow$	p													
$\circ\varphi \longrightarrow$		φ												
$\varphi\,\mathcal{U}\,\psi \longrightarrow$	φ	φ	φ	φ	φ	φ	φ	φ	ψ					
$\varphi\,\mathcal{V}\,\psi \longrightarrow$	ψ	ψ	ψ	ψ	ψ	ψ	φ, ψ							
$\Box\varphi \longrightarrow$	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
$\Diamond\varphi \longrightarrow$											φ			

Formalizing Reads and Writes

- Stage 3: Coordinator-to-replica



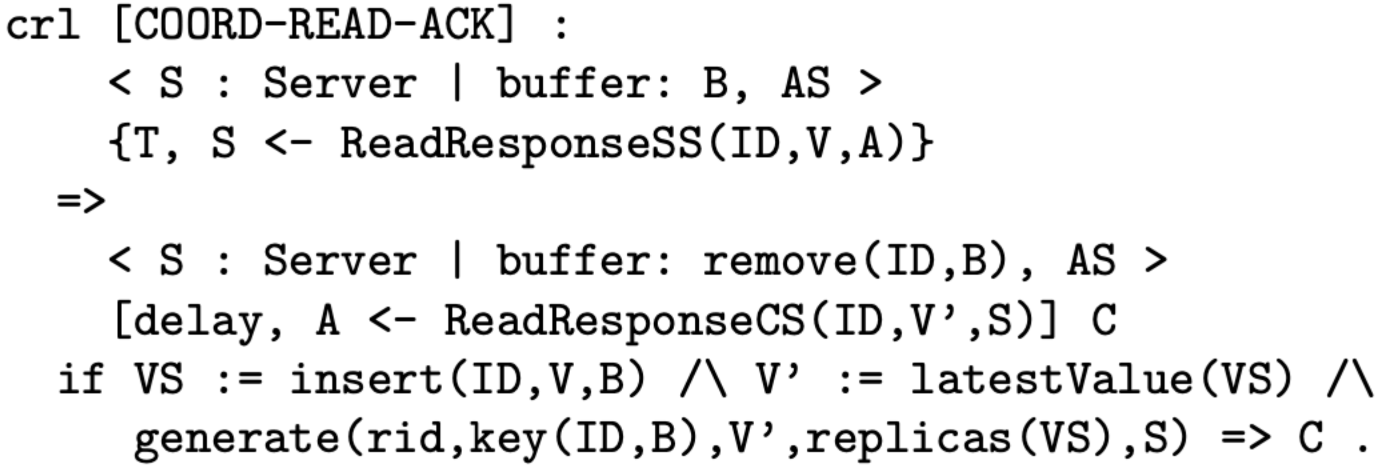
```
rl [REPLICA-READ-RESPONSE] :
   < S : Server | table: TB, AS >
   {T, S <- ReadRequestSS(ID,K,S’,A)}
 =>
   < S : Server | table: TB, AS >
   [delay, S’ <- ReadResponseSS(ID,TB[K],A)]  .
```



```
rl [REPLICA-WRITE-RESPONSE] :
   < S : Server | table: TB, AS >
   {T, S <- WriteRequestSS(ID,K,V,T’,S’,A)}
 =>
   if T’ >= tstamp(TB[K])
     then < S : Server | table: insert(K,V,T’,TB), AS >
          [delay, S’ <- WriteResponseSS(ID,success,A)]
     else < S : Server | table: TB, AS >
          [delay, S’ <- WriteResponseSS(ID,failure,A)] fi  .
```

Formalizing Reads and Writes

- Stage 4: Coordinator-to-client



```
crl [COORD-READ-ACK] :
    < S : Server | buffer: B, AS >
    {T, S <- ReadResponseSS(ID,V,A)}
  =>
    < S : Server | buffer: remove(ID,B), AS >
    [delay, A <- ReadResponseCS(ID,V’,S)] C
  if VS := insert(ID,V,B) /\ V’ := latestValue(VS) /\
     generate(rid,key(ID,B),V’,replicas(VS),S) => C .
```