Liberating distributed consensus

Heidi Howard @ University of Cambridge

heidi.howard@cl.cam.ac.uk
@heidiann360
www.heidihoward.co.uk

Distributed Dream

- **Performance** - scalability, low latency, high throughput, low cost, energy efficiency, versatility, adaptability
- **Reliability** - fault-tolerance, dependability, high availability, AP of CAP, self-healing, geo-replicated
- **Correctness** - consistency, bug-free, easy to understand

A Hundred Impossibility Proofs for Distributed Computing

Nancy A. Lynch *
Lab for Computer Science
MIT, Cambridge, MA 02139
lynch@tds.lcs.mit.edu

1 Introduction

This talk is about impossibility results in the area of distributed computing. In this category, I include not just results that say that a particular task cannot be accomplished, but also lower bound results, which say that a task cannot be accomplished within a certain bound on cost.

I started out with a simple plan for preparing this talk: I would spend a couple of weeks reading all the impossibility proofs in our field, and would categorize them according to the ideas used. Then I would make wise and general observations, and try to predict where the future of this area is headed. That turned out to be a bit too ambitious; there are many more such results than I thought. Although it is often hard to say what constitutes a "different result", I managed to count over 100 such impossibility proofs! And my search wasn't even very systematic or exhaustive.

It's not quite as hopeless to understand this area as it might seem from the number of papers. Although there are 100 different results, there aren't 100 different ideas. I thought I could contribute something by identifying some of the commonality among the different results.

So what I will do in this talk will be an incomplete version of what I originally intended. I will give you a tour of the impossibility results that I was able to collect. I apologize for not being comprehensive, and in particular for placing perhaps undue emphasis on results I have been involved in (but those are the ones I know best!). I will describe the techniques used, as well as giving some historical perspective. I'll intersperse this with my opinions and observations, and I'll try to collect what I consider to be the most important of these at the end. Then I'll make some suggestions for future work.

*This work was supported in part by the National Science Foundation (NSF) under Grant CCR-86-11442, by the Office of Naval Research (ONR) under Contract N00014-85-K-0168 and by the Defense Advanced Research Projects Agency (DARPA) under Contract N00014-83-K-0125.

Keywords: impossibility, distributed computing

2 The Results

I classified the impossibility results I found into the following categories: shared memory resource allocation, distributed consensus, shared registers, computing in rings and other networks, communication protocols, and miscellaneous.

2.1 Shared Memory Resource Allocation

This was the area that introduced me not only to the possibility of doing impossibility proofs for distributed computing, but to the entire distributed computing research area.

In 1976, when I was at the University of Southern California, Armin Cremers and Tom Hibbard were playing with the problem of *mutual exclusion* (or allocation of one resource) in a shared-memory environment. In the environment they were considering, a group of asynchronous processes communicate via shared memory, using operations such as read and write or test-and-set.

The previous work in this area had consisted of a series of papers by Dijkstra [38] and others, each presenting a new algorithm guaranteeing mutual exclusion, along with some other properties such as progress and fairness. The properties were specified somewhat loosely; there was no formal model used for

[PODC'89]

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.

Categories and Subject Descriptors: C.2.2 **[Computer-Communication Networks]**: Network Protocols–*protocol architecture*; C.2.4 **[Computer-Communication Networks]**: Distributed Systems–*distributed applications; distributed databases; network operating systems*; C.4 **[Performance of Systems]**: Reliability, Availability, and Serviceability; F.1.2 **[Computation by Abstract Devices]**: Modes of Computation–*parallelism*, H.2.4 **[Database Management]**: Systems–*distributed systems*; *transaction processing*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, fault tolerance, impossibility proof, reliability

1. *Introduction*

The problem of reaching agreement among remote processes is one of the most fundamental problems in distributed computing and is at the core of many

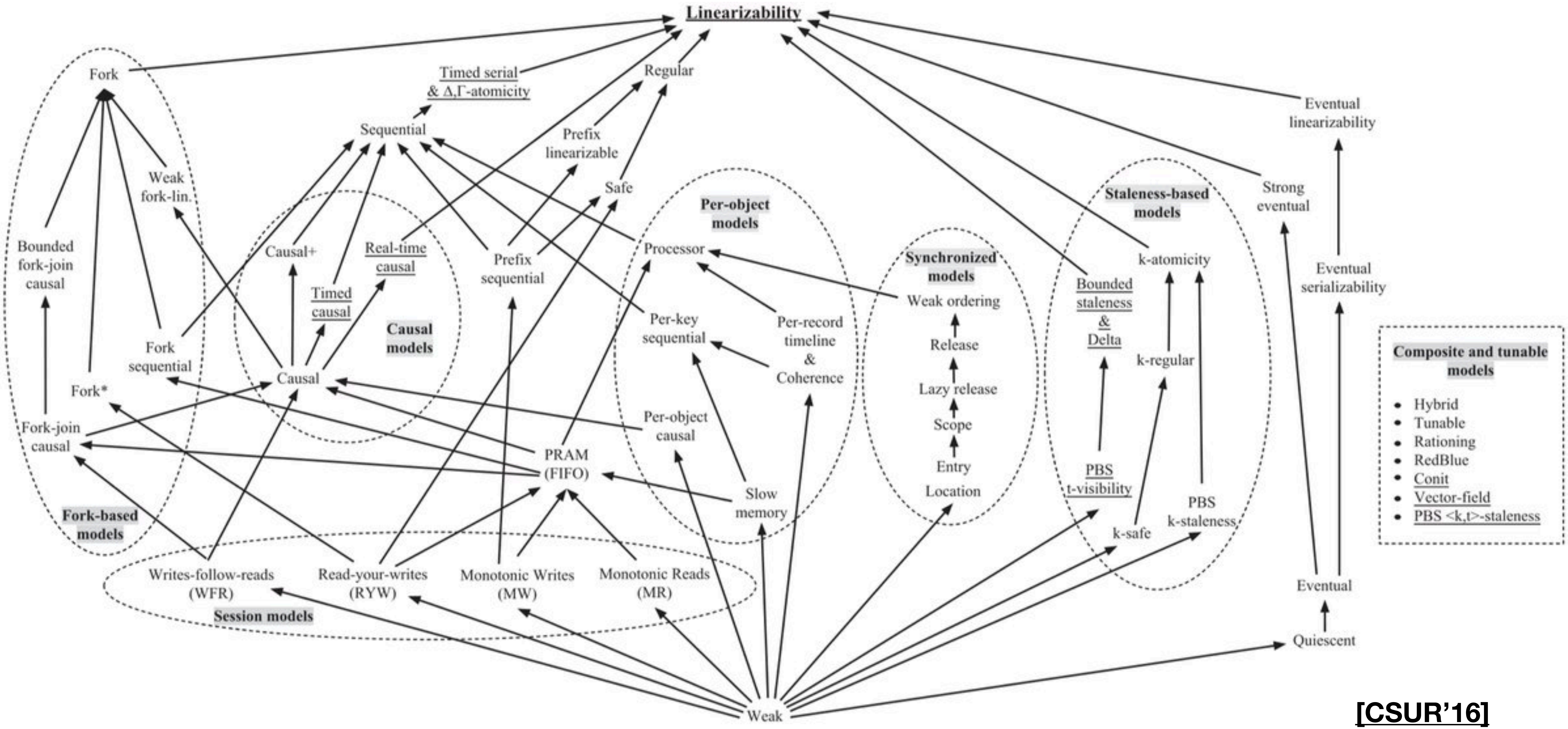
Editing of this paper was performed by guest editor S. L. Graham. The Editor-in-Chief of JACM did not participate in the processing of the paper.

This work was supported in part by the Office of Naval Research under Contract N00014-82-K-0154, by the Office of Army Research under Contract DAAG29-79-C-0155, and by the National Science Foundation under Grants MCS-7924370 and MCS-8116678.

This work was originally presented at the 2nd ACM Symposium on Principles of Database Systems, March 1983.

Authors' present addresses: M. J. Fischer, Department of Computer Science, Yale University, P.O. Box 2158, Yale Station, New Haven, CT 06520; N. A. Lynch, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139; M. S. Paterson, Department of Computer Science, University of Warwick, Coventry CV4 7AL, England

[JACM'85]


Linearizability
Fork
Timed serial & Δ,Γ-atomicity
Regular
Sequential
Prefix linearizable
Eventual linearizability
Weak fork-lin.
Safe
Per-object models
Staleness-based models
Strong eventual
Bounded fork-join causal
Causal+
Real-time causal
Prefix sequential
Processor
Synchronized models
k-atomicity
Eventual serializability
Timed causal
Causal models
Per-key sequential
Per-record timeline & Coherence
Weak ordering
Bounded staleness & Delta
Release
k-regular
Fork sequential
Causal
Lazy release
Composite and tunable models
Hybrid
Tunable
Rationing
RedBlue
Conit
Vector-field
PBS <k,t>-staleness
Fork*
Per-object causal
Scope
Fork-join causal
PRAM (FIFO)
Entry
PBS t-visibility
Location
Slow memory
PBS k-staleness
k-safe
Fork-based models
Writes-follow-reads (WFR)
Read-your-writes (RYW)
Monotonic Writes (MW)
Monotonic Reads (MR)
Eventual
Session models
Quiescent
Weak

[CSUR'16]

Deciding a single value

In this talk, we will reach agreement over a single value

The system is comprised of:

- **servers** which store the value
- **clients** which propose values and learn the decided value

We assume a non-Byzantine system.

Requirements of consensus

- **Safety -** All client must learn the same decided value
- **Progress -** Eventually, all clients must learn the decided value

Safety must hold even in unreliable and asynchronous systems

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state machine approach to the design of distributed systems.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.5 [**Operating Systems**]: Reliability—*fault-tolerance*; J.1 [**Computer Applications**]: Administrative Data Processing—*government*

General Terms: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

1. THE PROBLEM

1.1 The Island of Paxos

Early in this millennium, the Aegean island of Paxos was a thriving mercantile center.[1] Wealth led to political sophistication, and the Paxons replaced their ancient theocracy with a parliamentary form of government. But trade came before civic duty, and no one in Paxos was willing to devote his life to Parliament. The Paxon Parliament had to function even though legislators continually wandered in and out of the parliamentary Chamber.

The problem of governing with a part-time parliament bears a remarkable correspondence to the problem faced by today's fault-tolerant distributed systems, where legislators correspond to processes, and leaving the Chamber corresponds to failing. The Paxons' solution may therefore be of some interest to computer scientists. I present here a short history of the Paxos Parliament's protocol, followed by an even shorter discussion of its relevance for distributed systems.

Author's address: Systems Research, Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, CA 94301.

[1] It should not be confused with the Ionian island of Paxoi, whose name is sometimes corrupted to *Paxos*.

ACM Transactions on Computer Systems, Vol. 16, No. 2, May 1998, Pages 133–169.

[TOCS'98]

“The Paxos algorithm, when presented in plain English, is very simple.”

“The Paxos algorithm … is among the simplest and most obvious of distributed algorithms”

“… this consensus algorithm follows almost unavoidably from the properties we want it to satisfy.”

Leslie Lamport, Paxos Made Simple

Theory community perspective

“There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system. ”

“Despite the existing literature on [Paxos], building a production system turned out to be a non-trivial task”

Chandra et al, Paxos Made Live

Engineering community perspective

“Paxos is exceptionally difficult to understand. The full explanation is notoriously opaque; few people succeed in understanding it, and only with great effort. …”

“… we found few people who were comfortable with Paxos, even among seasoned researchers.”

“We concluded that Paxos does not provide a good foundation either for system building or for education.”

Diego Ongaro and John Ousterhout, In Search of an Understandable Consensus Algorithm

Research community perspective

Limitations of Paxos

- **Subtlety** - Paxos is famously difficult to understand.
- **Performance** - Paxos is slow. Each decision requires at least two round trips to a majority of servers.

Today’s Talk

Instead of mitigating these issues, we rethink the underlying principles.

Part 1

We reframe the problem of consensus using immutable state.

→

Part 2

We generalise the Paxos algorithm.

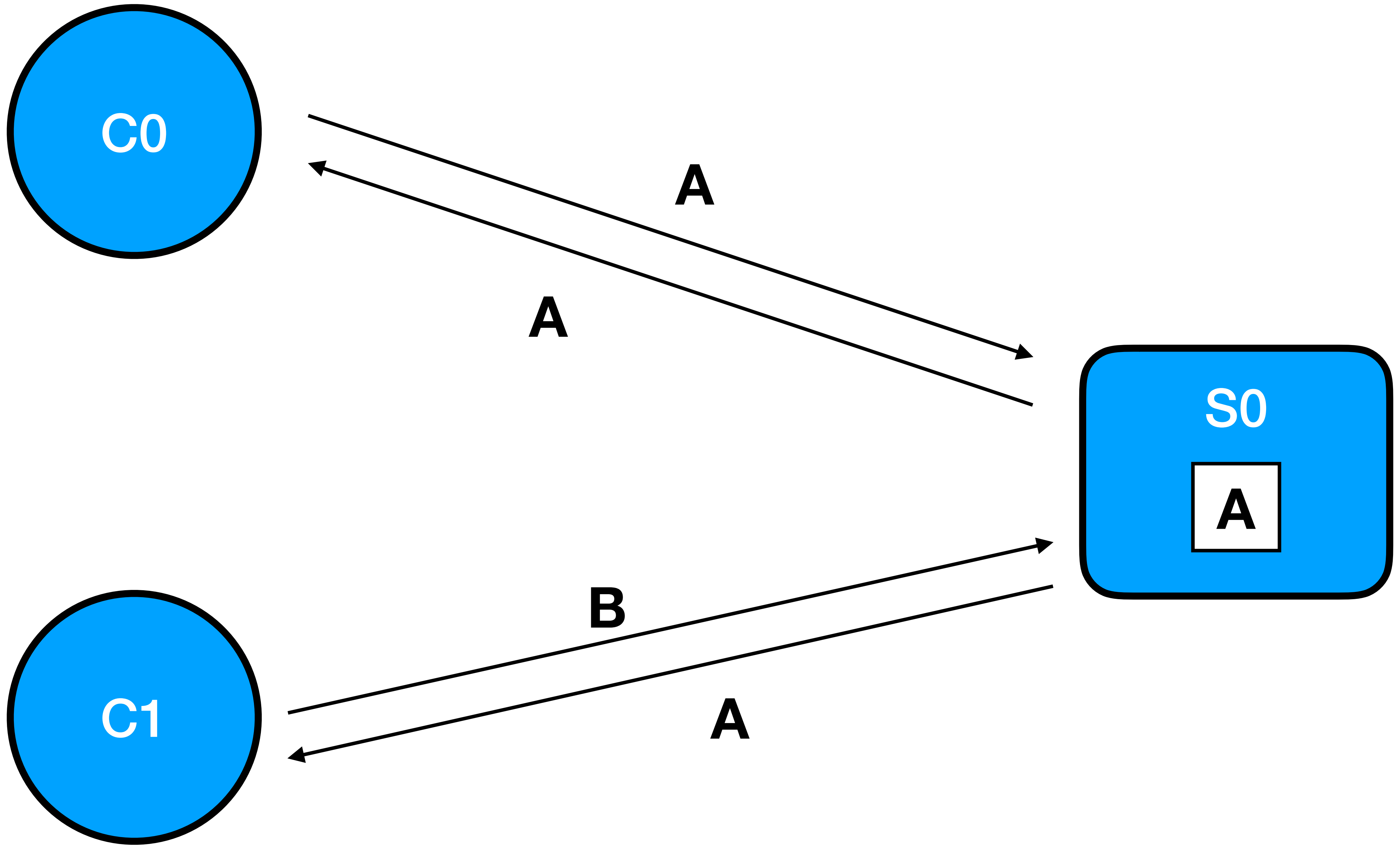
→

Part 3

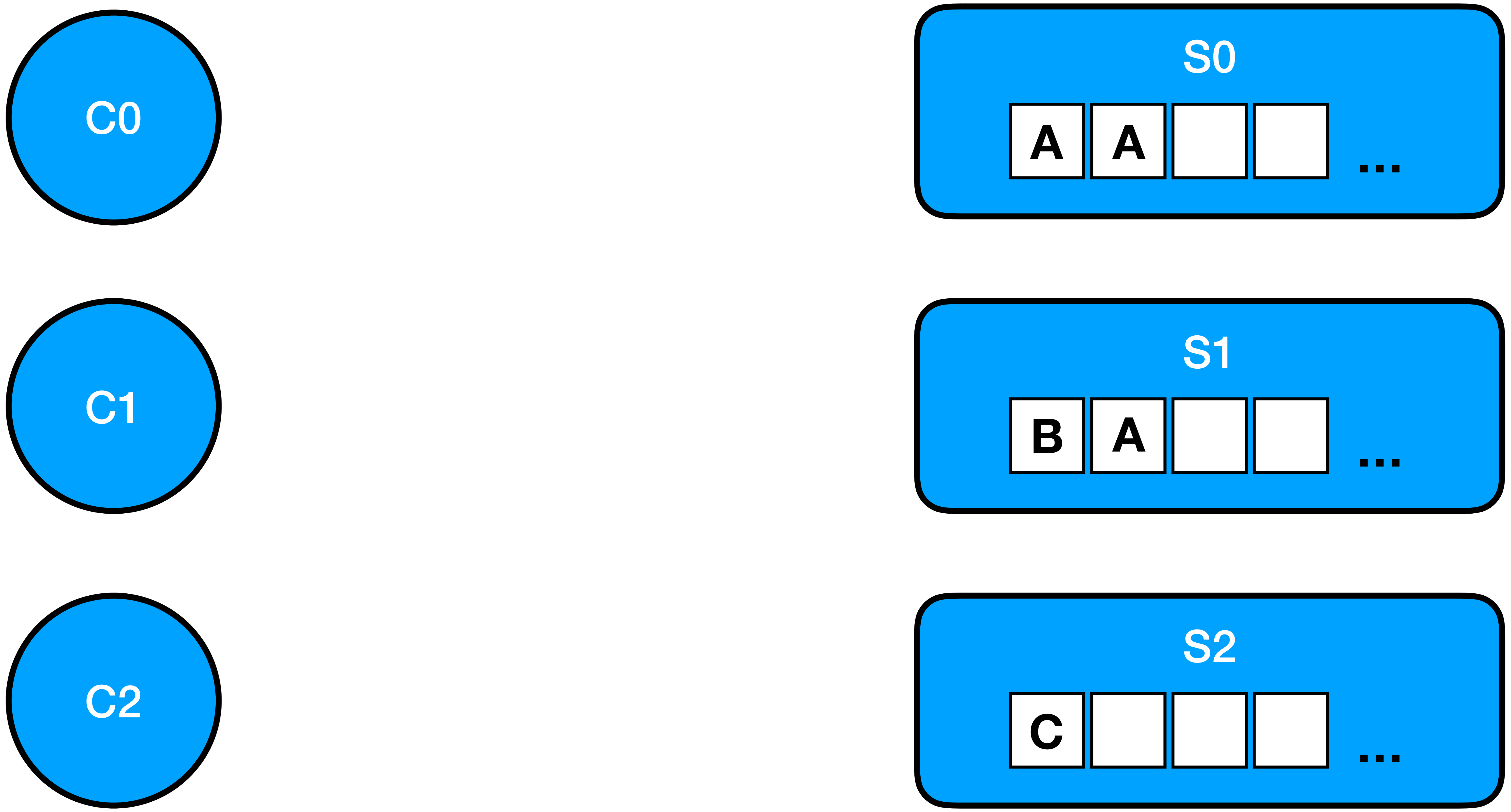
We introduce the All aboard consensus algorithm.

Part 1
Distributed consensus
using write-once registers

Single server



Multiple servers, multiple registers



State table

Servers

	S0	S1	S2
R0	A	B	C
R1	A	A	C
R2	A	A	A
R3			-

Register sets

Nil value

Decision point

A value is ***decided*** when it has been written to the same register on a subsets of servers, known as a **quorum**.

Once a client reads the same value from a quorum of registers, it learns that the value has been decided.

Quorum table

Registers	Quorums
R0+	{{S0,S1},{S1,S2},{S0,S2}}

	S0	S1	S2
R0	-	A	A
R1	-	-	A
R2	A	A	A
R3	A		-

A is decided

A is decided

Quorum table

Registers	Quorums
R0	{{S0,S1,S2,S3}}
R1+	{{S0,S1},{S2,S3}}

	S0	S1	S2	S3
R0	B	B		A
R1	-	-	A	A
R2	A	A	A	
R3	A			

A is decided

A is decided

However we can decide multiple values

	S0	S1	S2	S3
R0	-	A	A	
R1	C	C	A	A
R2	A		A	

Registers	Quorums
R0	{{S0,S1,S2,S3}}
R1+	{{S0,S1},{S2,S3}}

	S0	S1	S2
R0	C	A	A
R1	B	B	A
R2	A	C	C
R3	A		-

Registers	Quorums
R0+	{{S0,S1},{S1,S2},{S0,S2}}

Safety

Only one value should ever be decided

Before a client writes a value to register *i* it must ensure that no other values are decided in register sets *0* to *i.*

Part 2
Generalising Paxos

Classic Paxos

Paxos is a two phase, majority based algorithm which solves distributed consensus.

Registers	Quorums
R0+	{{S0,S1},{S1,S2},{S0,S2}}

Safety

Only one value should ever be decided

Before a client writes a value to register *i* it must ensure that:

1. No other values are decided in register set *i*

2. No other values are decided in register sets *0* to *i-1*

Register allocation rule

We allocate registers to clients round robin and require clients to write at most one value to each of their allocated registers.

This ensures that at most one value will be written to each register set.

Round robin allocation of registers to servers

Registers	**Client**
R0, R3, …	C0
R1, R4, …	C1
R2, R6, …	C3

Safety

Only one value should ever be decided

Before a client writes a value to register *i* it must ensure that:

1. No other values are decided in register set *i*
2. No other values are decided in register sets *0* to *i-1*

Register allocation rule

Client write rule

A client can achieve this by reading one register from each quorum over register sets 0 to i-1 and ensuring that:

- None of the registers are unwritten
- If any registers contain values, the client must write the value from the greatest register.

Safety

Only one value should ever be decided

Before a client writes a value to register *i* it must ensure that:

1. No other values are decided in register set *i*

Register allocation rule

2. No other values are decided in register sets *0* to *i-1*

Client write rule

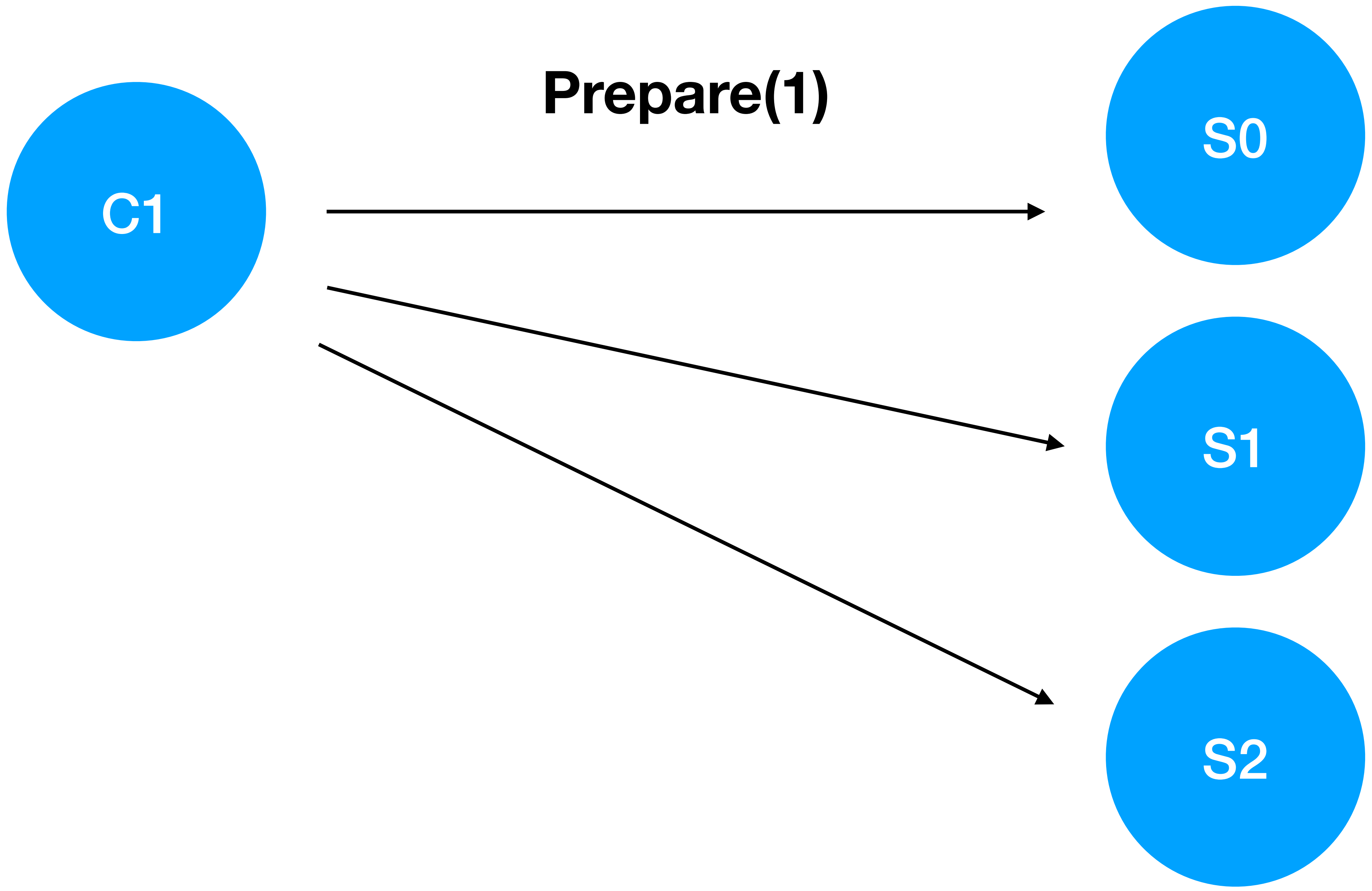
Classic Paxos - Phase one

- The client chooses an allocated register *i* and sends *prepare(i)* to all servers.
- Provided register *i* is unwritten, each server writes *nil* in any unwritten registers from *0* to *i-1* and replies with the register number *j* and value *w* of the greatest non-nil register using *promise(i,j,w)*

Classic Paxos - Phase two

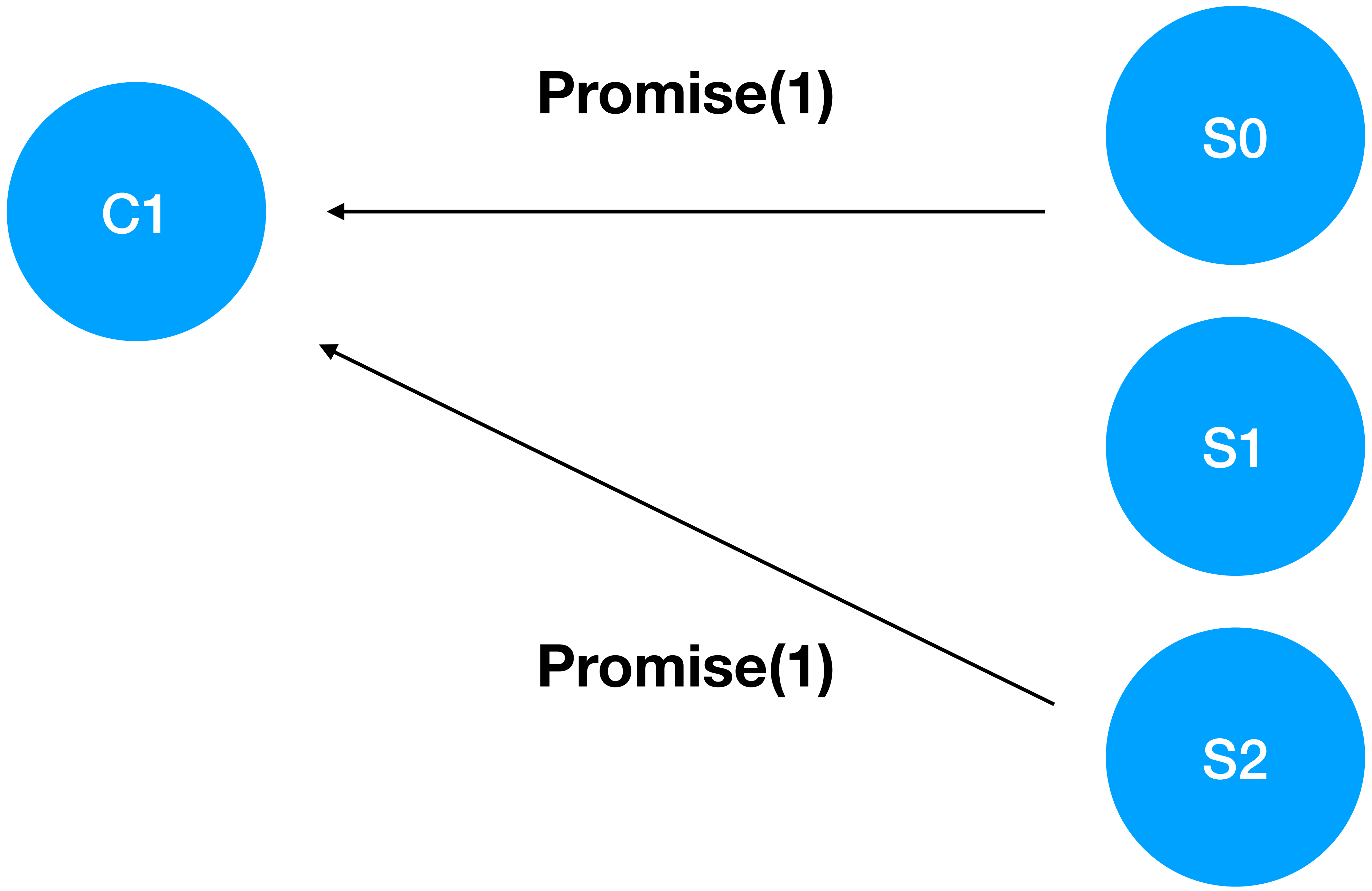
- After a majority of servers reply, the client chooses the value *v* from the greatest register or its own value if none. Client sends *propose(i,v)* to all servers.
- Provided i is unwritten, each server writes nil to any unwritten registers from *0* to *i-1 and* value *v* to the register *i*. The server replies to the client using *accept(i)*
- The client terminates when *accept(i) is received from the majority of servers.*

Example - Phase one



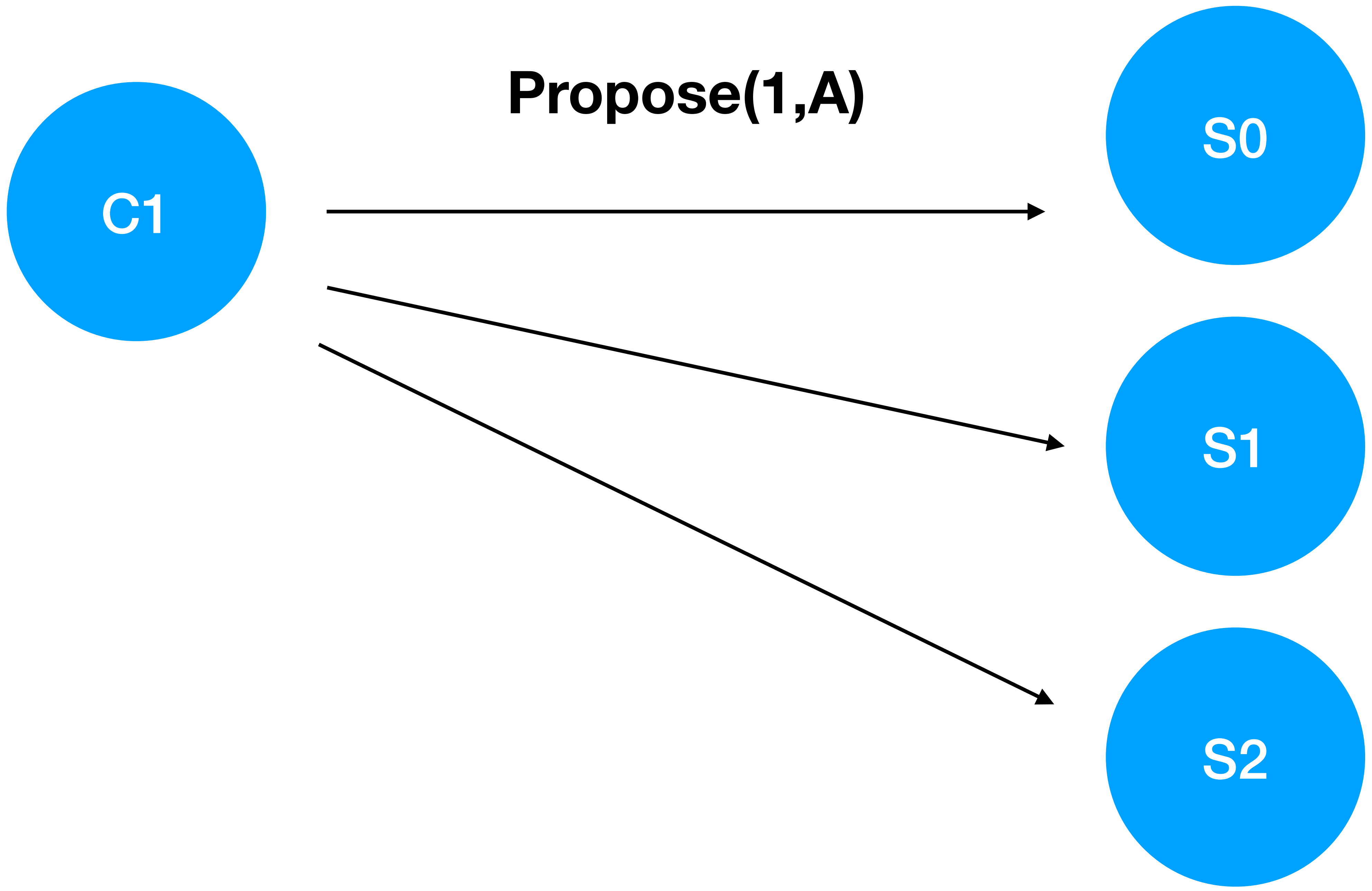
	S0	S1	S2
R0			
R1			
R2			
R3			

Example - Phase one



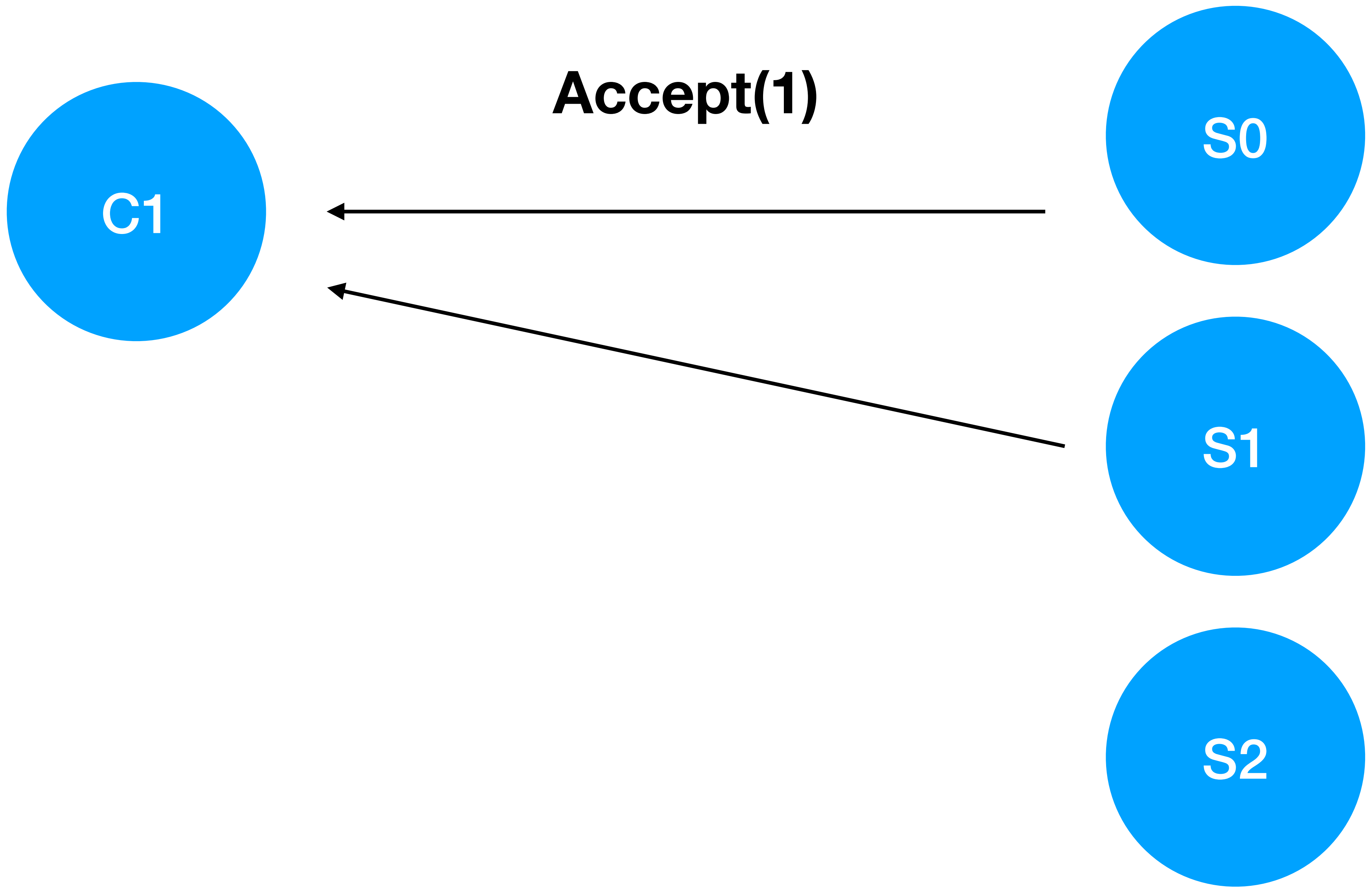
	S0	S1	S2
R0	-	-	-
R1			
R2			
R3			

Example - Phase two



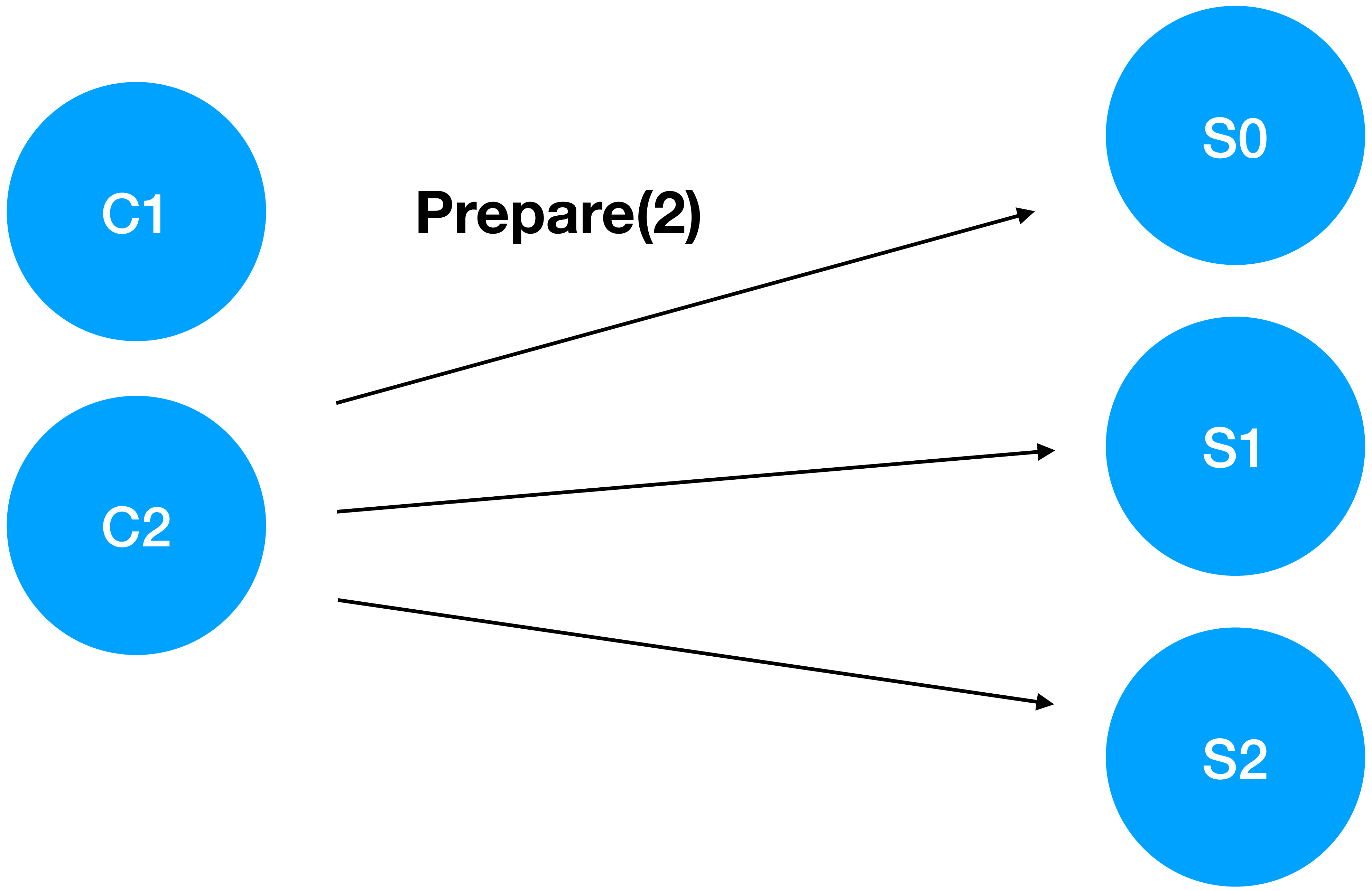
	S0	S1	S2
R0	-	-	-
R1			
R2			
R3			

Example - Phase two



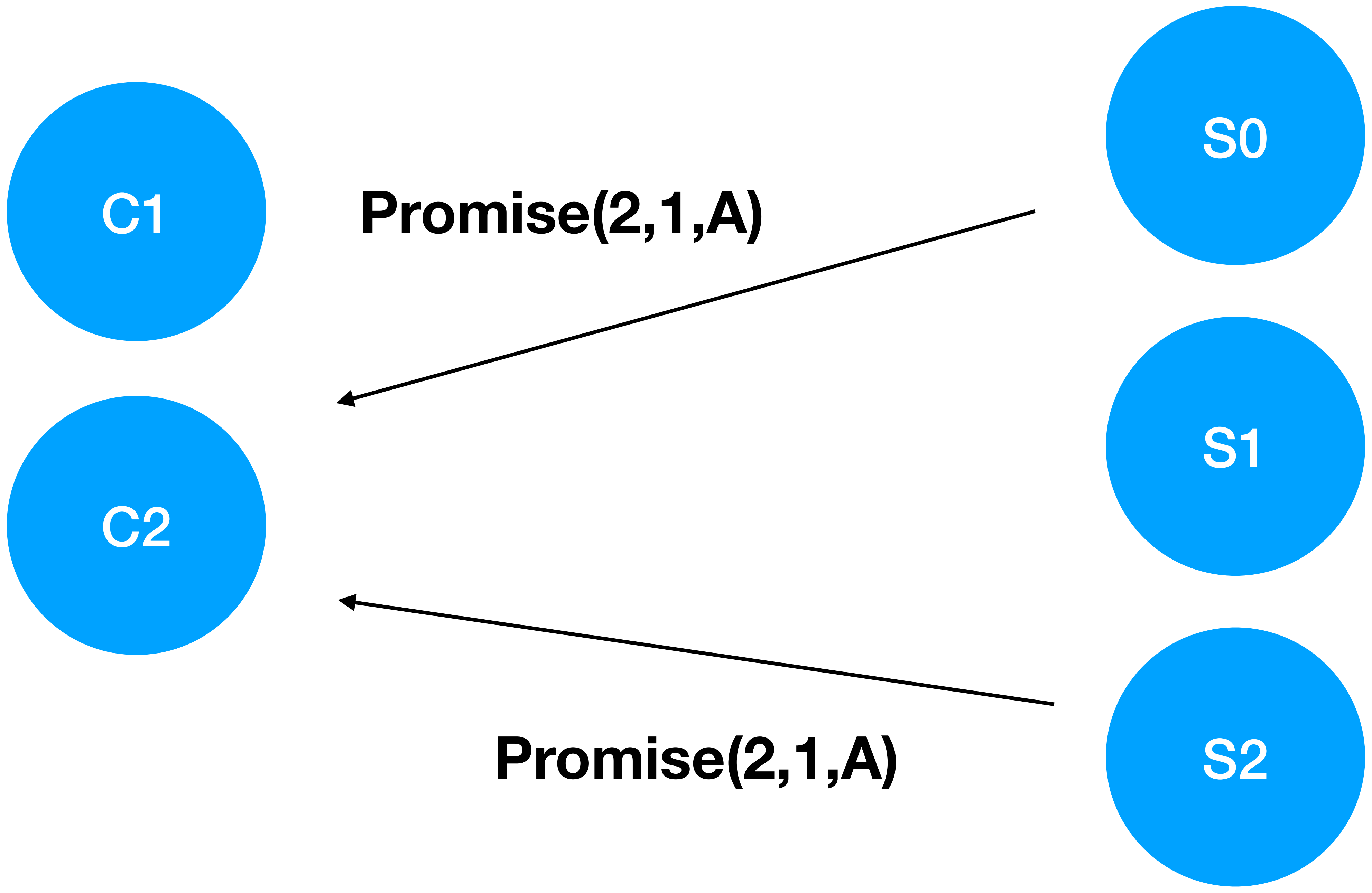
	S0	S1	S2
R0	-	-	-
R1	A	A	A
R2			
R3			

Example - Phase one



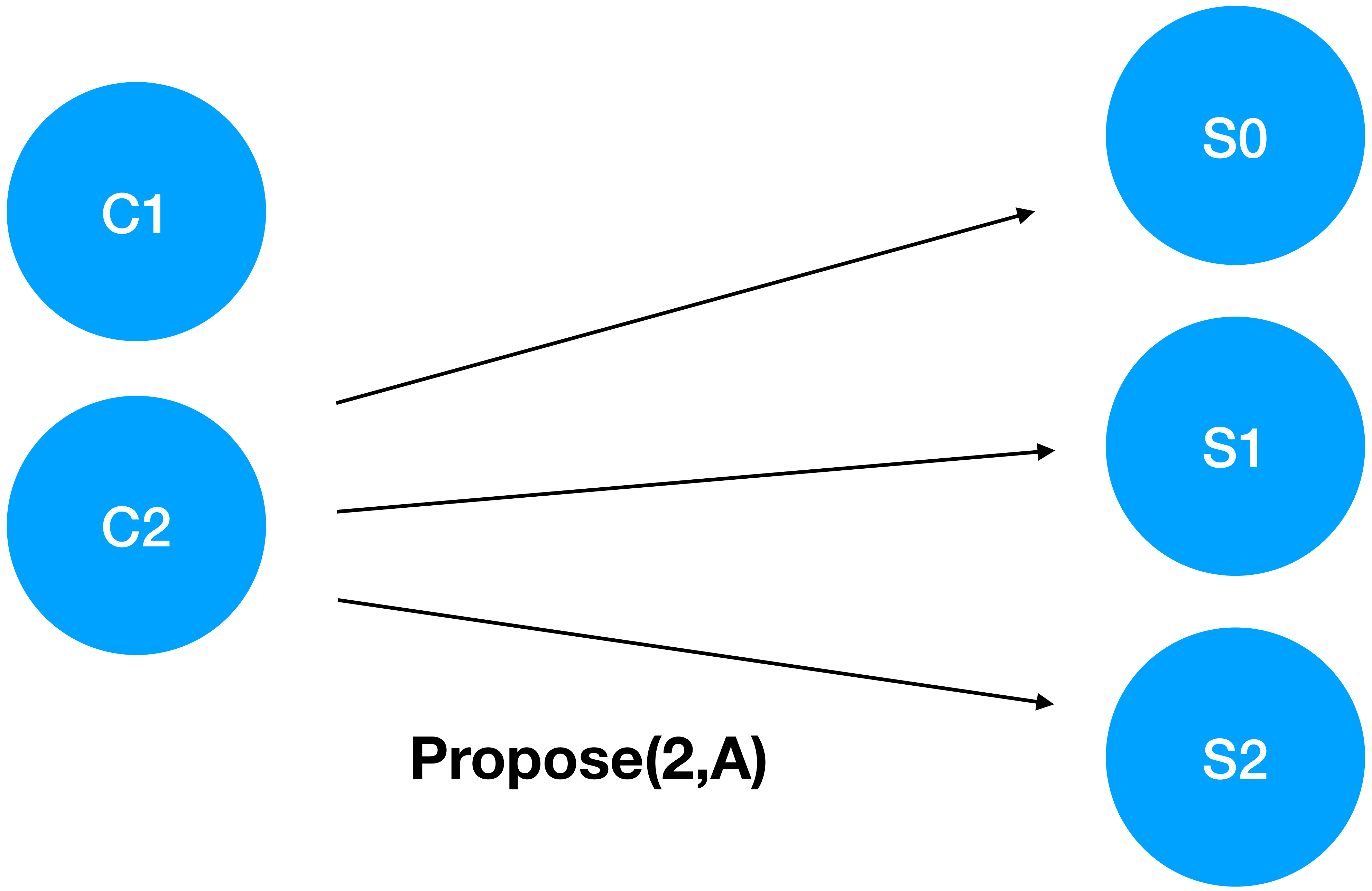
	S0	S1	S2
R0	-	-	-
R1	A	A	A
R2			
R3			

Example - Phase one



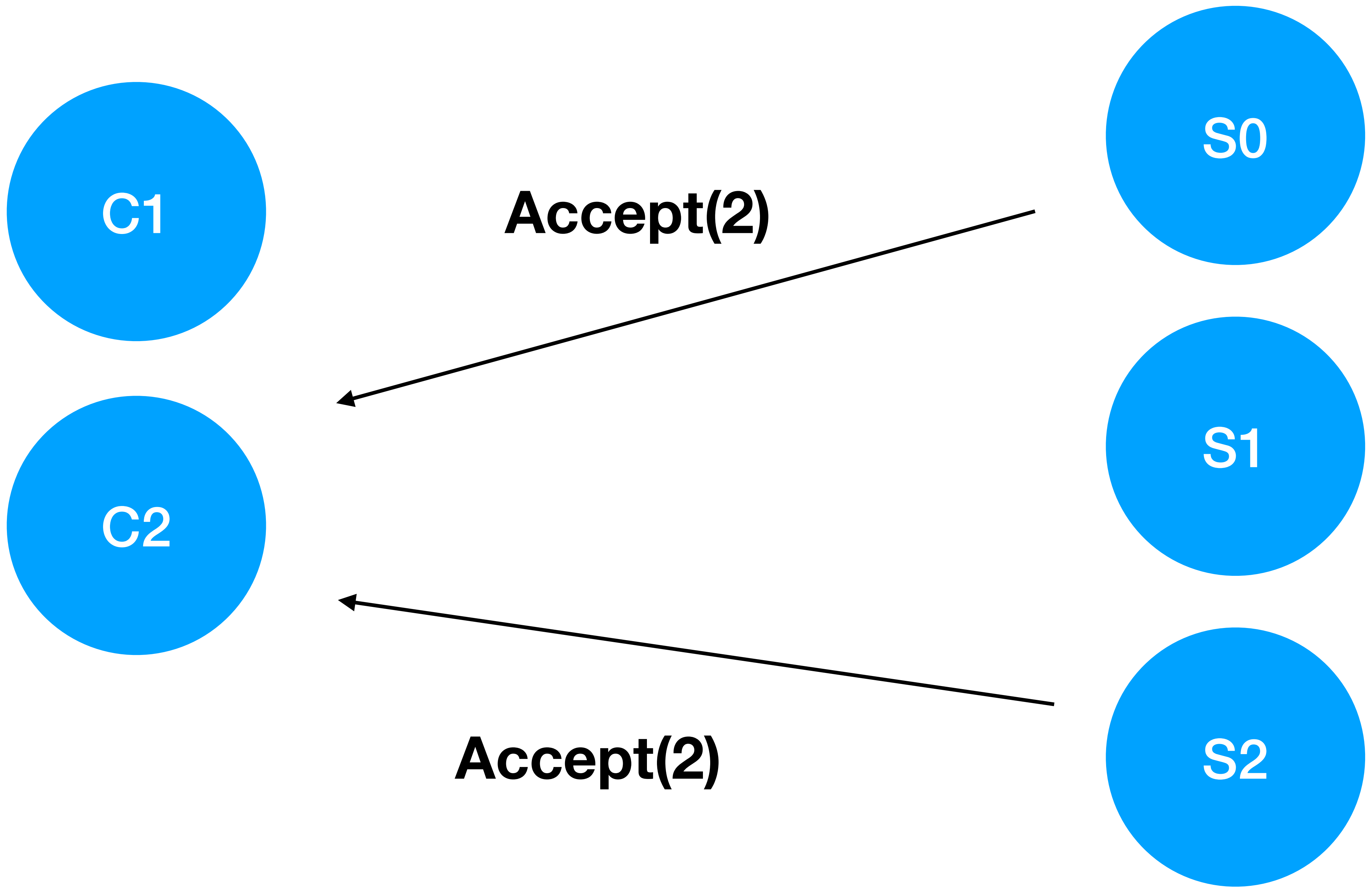
	S0	S1	S2
R0	-	-	-
R1	A	A	A
R2			
R3			

Example - Phase two



	S0	S1	S2
R0	-	-	-
R1	A	A	A
R2			
R3			

Example - Phase two



	S0	S1	S2
R0	-	-	-
R1	A	A	A
R2	A	A	A
R3			

Quorum intersection

Original requirement - Paxos requires that each of its two phases use a quorum of servers and that any two quorums must intersect.

Revised requirement - A client using register *i* must get at least one server from each quorum of registers *0* to *i-1* to participate in phase one.

Part 3
All aboard consensus

Current Reality

	Classic Paxos	Multi Paxos
Minimum round trips?	2	1
Which client can decide the value?	Any	Leader only

Can we design an algorithm in which ***any client*** can achieve consensus in just ***1 round trip***?

Design

In many distributed systems:

- Each server and client is co-located on the same host
- Failures are rare

All aboard - Quorum table

Registers	Quorums
R0, R1, R2	{{S0,S1,S2}}
R3+	{{S0,S1},{S1,S2},{S0,S2}}

Registers partitioned at 3

All aboard - Algorithm

Fast path [R0, R1, R2]

Execute phase one locally, followed by phase two with all participants. If unsuccessful, try slow path.

Slow path [R3+]

Classic two phase paxos with majorities.

All aboard consensus

Pros

- If all servers are up then all clients can terminate in 1 RTT
- If two clients collide, one will succeed and the other will retry.

Cons

- Requires co-location
- 2 RTTs are needed if a server is slow/unavailable

This is just the beginning

- Flexible Paxos: Quorum intersection revisited [OPODIS’16]
- A generalised solution to distributed consensus [arXiv’19]
- Distributed consensus revised [PhDthesis’19]

Closing Remarks

Paxos is a single point on a broad and diverse spectrum of consensus algorithms.

Any questions?

Heidi Howard
heidi.howard@cl.cam.ac.uk
@heidiann360
heidihoward.co.uk