State-Machine Reconfiguration: Past, Present, and the Cloudy Future

Leslie Lamport
Microsoft Research

The Forecast: Clouds

The Forecast: Clouds

In a cloud, computers come and go.

The Forecast: Clouds

In a cloud, computers come and go.

A service provided by a cloud must be continually reconfigured to change the set of servers.

The Forecast: Clouds

In a cloud, computers come and go.

A service provided by a cloud must be continually reconfigured to change the set of servers.

A service is implemented with a state machine.

The Forecast: Clouds

In a cloud, computers come and go.

A service provided by a cloud must be continually reconfigured to change the set of servers.

A service is implemented with a state machine.

I'll discuss state-machine reconfiguration

The Forecast: Clouds

In a cloud, computers come and go.

A service provided by a cloud must be continually reconfigured to change the set of servers.

A service is implemented with a state machine.

I'll discuss state-machine reconfiguration, and make my own forecast.

Part I: (More or Less Ancient) History

Part I: (More or Less Ancient) History

Those who cannot remember the past

Part I: (More or Less Ancient) History

Those who cannot remember the past are young enough to enjoy rediscovering it.

In the Beginning

The Maintenance of Duplicate Databases
by Paul Johnson and Bob Thomas. (1976?)

In the Beginning

The Maintenance of Duplicate Databases
by Paul Johnson and Bob Thomas. (1976?)

How to keep consistent copies of a database in a network.

In the Beginning

The Maintenance of Duplicate Databases
by Paul Johnson and Bob Thomas. (1976?)

How to keep consistent copies of a database in a network.

They missed two things:

In the Beginning

The Maintenance of Duplicate Databases
by Paul Johnson and Bob Thomas. (1976?)

How to keep consistent copies of a database in a network.

They missed two things:

- Commands not executed in causal order.

In the Beginning

The Maintenance of Duplicate Databases
by Paul Johnson and Bob Thomas. (1976?)

How to keep consistent copies of a database in a network.

They missed two things:

- Commands not executed in relativistic causal order.

In the Beginning

The Maintenance of Duplicate Databases
by Paul Johnson and Bob Thomas. (1976?)

How to keep consistent copies of a database in a network.

They missed two things:

relativistic

- Commands not executed in causal order.
- It could be used to implement any system.

Time, Clocks and the Ordering of Events in a Distributed System (1978)

Time, Clocks and the Ordering of Events in a Distributed System (1978)

Two contributions:

Time, Clocks and the Ordering of Events in a Distributed System (1978)

Two contributions:

- The *causality* partial order on events in a distributed system.

Time, Clocks and the Ordering of Events in a Distributed System (1978)

Two contributions:

- The *causality* partial order on events in a distributed system.
- Implementing any distributed system by

Time, Clocks and the Ordering of Events in a Distributed System (1978)

Two contributions:

- The *causality* partial order on events in a distributed system.
- Implementing any distributed system by
  - Describing it as a state machine, and

Time, Clocks and the Ordering of Events in a Distributed System (1978)

Two contributions:

- The *causality* partial order on events in a distributed system.
- Implementing any distributed system by
  - Describing it as a state machine, and
  - Using a general algorithm for implementing any state machine.

Time, Clocks and the Ordering of Events in a Distributed System (1978)

Two contributions:

- The *causality* partial order on events in a distributed system.
- Implementing any distributed system by
  - Describing it as a state machine, and
  - Using a general algorithm for implementing any state machine.

Reaching Agreement in the Presence of Faults

by Marshall Pease, Robert Shostak, and L. L. (1980)

Reaching Agreement in the Presence of Faults
by Marshall Pease, Robert Shostak, and L. L. (1980)

Added (Byzantine) fault tolerance.

Reaching Agreement in the Presence of Faults
by Marshall Pease, Robert Shostak, and L. L. (1980)

Added (Byzantine) fault tolerance.

Inspired by SIFT (Software Implemented Fault Tolerance):
an architecture for a flight-control computer.

Reaching Agreement in the Presence of Faults
by Marshall Pease, Robert Shostak, and L. L. (1980)

Added (Byzantine) fault tolerance.

Inspired by SIFT (Software Implemented Fault Tolerance):
an architecture for a flight-control computer.

Abstracted the kernel problem: consensus.

Reaching Agreement in the Presence of Faults
by Marshall Pease, Robert Shostak, and L. L. (1980)

Added (Byzantine) fault tolerance.

Inspired by SIFT (Software Implemented Fault Tolerance):
an architecture for a flight-control computer.

Abstracted the kernel problem: consensus.

State machine implicit in the software structure:
tasks executed iteratively.

Synchronous implementation.

Synchronous* implementation.

*Assumes known bound on communcication delay.

Synchronous* implementation.

Reconfiguration an important part of the system.

*Assumes known bound on communcication delay.

Synchronous* implementation.

Reconfiguration an important part of the system.

High reliability (mean time to failure of 1M years) depended on software rapidly identifying faulty processors and removing them from the system.

*Assumes known bound on communcication delay.

Synchronous* implementation.

Reconfiguration an important part of the system.

High reliability (mean time to failure of 1M years) depended on software rapidly identifying faulty processors and removing them from the system.

One iterative task decided what processors should execute the next iteration of each task.

*Assumes known bound on communcication delay.

The Asynchronous Case

Synchronous systems designed for process control.

The Asynchronous Case

Synchronous systems designed for process control.

The systems on our desk are synchronous only most of the time.

The Asynchronous Case

Synchronous systems designed for process control.

The systems on our desk are synchronous only most of the time.

Impossibility of Distributed Consensus with One Faulty Process by Michael Fischer, Nancy Lynch, and Michael Paterson (1985)

The Asynchronous Case

Synchronous systems designed for process control.

The systems on our desk are synchronous only most of the time.

Impossibility of Distributed Consensus with One Faulty Process
by Michael Fischer, Nancy Lynch, and Michael Paterson (1985)

Showed asynchronous consensus impossible

For asynchronous systems:

For asynchronous systems:

- Always maintain consistency.

For asynchronous systems:

- Always maintain consistency.
- Achieve progress when synchronous.

For asynchronous systems:

- Always maintain consistency.
- Achieve progress when synchronous.

Asynchronous consensus:

For asynchronous systems:

- Always maintain consistency.
- Achieve progress when synchronous.

Asynchronous consensus:

Consensus in the Presence of Partial Synchrony
by Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer (1988)

For asynchronous systems:

- Always maintain consistency.
- Achieve progress when synchronous.

Asynchronous consensus:

Consensus in the Presence of Partial Synchrony
by Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer (1988)

A state-machine implementation (Paxos)

For asynchronous systems:

- Always maintain consistency.
- Achieve progress when synchronous.

Asynchronous consensus:

Consensus in the Presence of Partial Synchrony by Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer (1988)

A state-machine implementation (Paxos)

Viewstamped Replication for Highly Available Distributed Systems by Brian Oki (1988)

For asynchronous systems:

- Always maintain consistency.
- Achieve progress when synchronous.

Asynchronous consensus:

Consensus in the Presence of Partial Synchrony
by Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer (1988)

A state-machine implementation (Paxos)

Viewstamped Replication for Highly Available Distributed Systems by Brian Oki (1988)

The Part-Time Parliament (1989)

Making Paxos tolerate Byzantine faults:

Making Paxos tolerate Byzantine faults:

Practical Byzantine Fault Tolerance
by Miguel Castro and Barbara Liskov (1999)

State Machines

State Machines

A state machine is a mapping:

$$\langle Command, OldState \rangle \rightarrow \langle Response, NewState \rangle$$

State Machines

A state machine is a mapping:

$$\langle \mathit{Command}, \mathit{OldState} \rangle \rightarrow \langle \mathit{Response}, \mathit{NewState} \rangle$$

You can represent any system as a state machine.

State Machines

A state machine is a mapping:

$$\langle\, Command, OldState \,\rangle \;\rightarrow\; \langle\, Response, NewState \,\rangle$$

You can represent $_\wedge$ any system as a state machine.

almost

An example: A Toy Banking System

An example: A Toy Banking System

$State$: The balance of each depositor's account

An example: A Toy Banking System

$State$: The balance of each depositor's account

$Command$: Deposit \$20 to Alice's account.

An example: A Toy Banking System

$State$: The balance of each depositor's account

$Command$: Deposit \$20 to Alice's account.

$Response$: "OK"

An example: A Toy Banking System

$State$: The balance of each depositor's account

$Command$: Deposit $20 to Alice's account.

$Response$: "OK"

$NewState$ = $Oldstate$ with $20 added to Alice's account

$State$: The balance of each depositor's account

$Command$: Withdraw \$100 from Bob's account.

$State$: The balance of each depositor's account

$Command$: Withdraw \$100 from Bob's account.

$Response$: "Insufficient Funds"

$State$: The balance of each depositor's account

$Command$: Withdraw \$100 from Bob's account.

$Response$: "Insufficient Funds"

$NewState = Oldstate$

$State$: The balance of each depositor's account

$Command$: Transfer \$100 from Alice's account to Bob's account.

$State$: The balance of each depositor's account

$Command$: Transfer $100 from Alice's account
to Bob's account.

$Response$: To Alice: "Done"
To Bob: "$100 received from Alice"

$State$: The balance of each depositor's account

$Command$: Transfer \$100 from Alice's account to Bob's account.

$Response$: To Alice: "Done"
To Bob: "\$100 received from Alice"

$NewState$ = $Oldstate$ with \$100 removed from Alice's account and added to Bob's

A Possible State-Machine Command

```
if (k <= 3) {
   int start = GF_256_ANTILOG (0);
   int factor = GF_256_ANTILOG (1);
   for (i = 0; i != k; i++) {
      int j;
      int colentry = GF_256_ANTILOG (0);
      int colfactor = start;
      for (j = 0; j != m; j++) {
         rs->matrix[i*m + j] = GF_256_LOG (colentry);
         colentry = GF_256_MUL (colentry, colfactor); }
   start = GF_256_MUL (start, factor);  }
     } else {int start = GF_256_ANTILOG (0);
            int factor = GF_256_ANTILOG (1);
...
```

A Possible State-Machine Command

```
if (k <= 3) {
   int start = GF_256_ANTILOG (0);
   int factor = GF_256_ANTILOG (1);
   for (i = 0; i != k; i++) {
      int j;
      int colentry = GF_256_ANTILOG (0);
      int colfactor = start;
      for (j = 0; j != m; j++) {
         rs->matrix[i*m + j] = GF_256_LOG (colentry);
         colentry = GF_256_MUL (colentry, colfactor); }
   start = GF_256_MUL (start, factor);  }
        } else {int start = GF_256_ANTILOG (0);
               int factor = GF_256_ANTILOG (1);
...
```

It just has to be deterministic.

- Implementing any distributed system by
  - Describing it as a state machine

- Implementing any distributed system by
  - Describing it as a state machine

No concurrency/distribution.

- Implementing any distributed system by
  - Describing it as a state machine
  - Using a general algorithm for implementing any state machine.

- Implementing any distributed system by
  - Describing it as a state machine
  - Using a general algorithm for implementing any state machine.

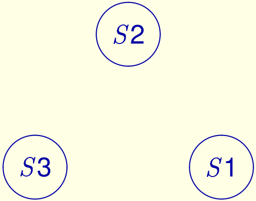
    Just implement once.

- Implementing any distributed system by
  - Describing it as a state machine
  - Using a general algorithm for implementing any state machine.

    Just implement once. (Middleware)

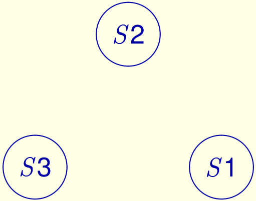
A Typical Non-Byzantine State-Machine Implementation

A Typical Non-Byzantine State-Machine Implementation



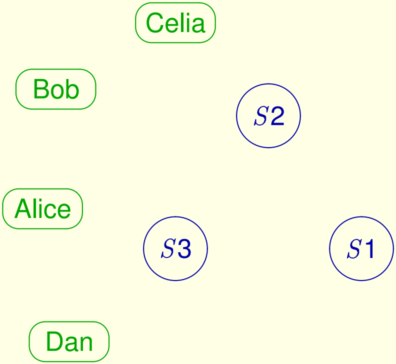
The servers.

A Typical Non-Byzantine State-Machine Implementation



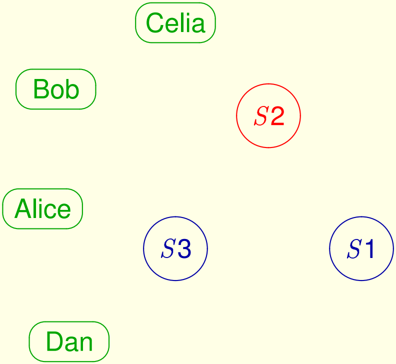
The servers. To tolerate 1 failure, need 3 servers.

A Typical Non-Byzantine State-Machine Implementation

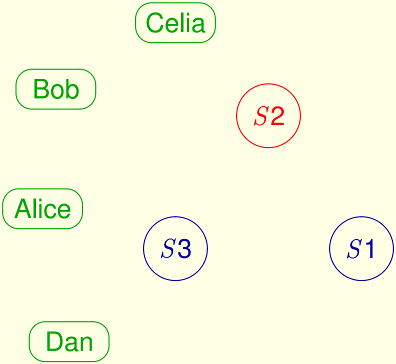


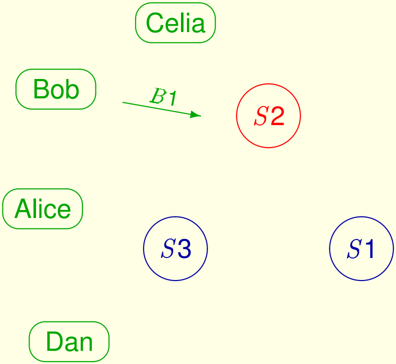
The clients.

A Typical Non-Byzantine State-Machine Implementation

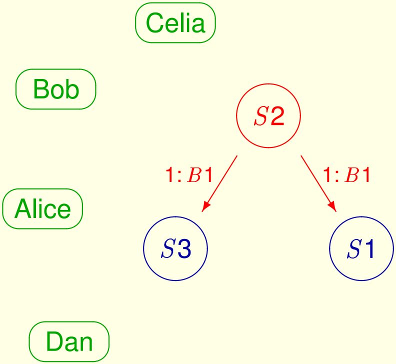


One server chosen to be leader.

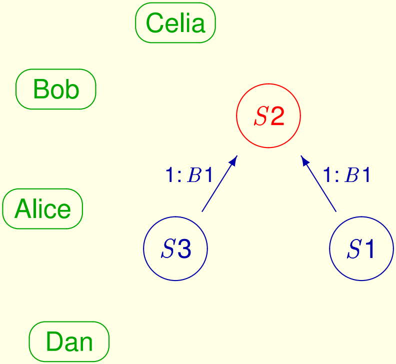

Celia
Bob
S2
Alice
S3
S1
Dan



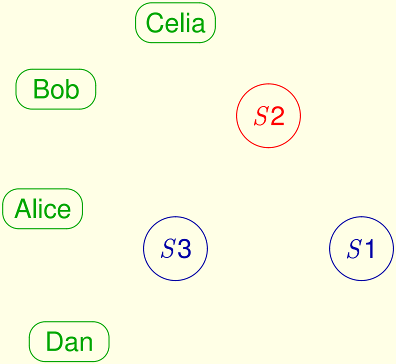
Bob sends command $B1$ to leader.



Leader assigns number 1 to command and sends to servers.



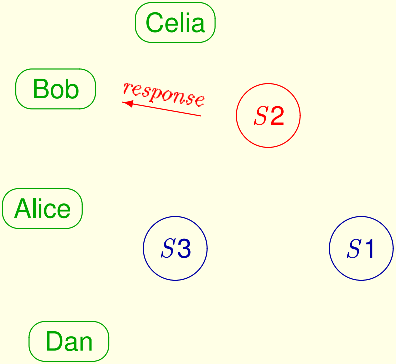
Servers remember $1\colon B1$ and ack.



Chosen Commands

1: $B1$

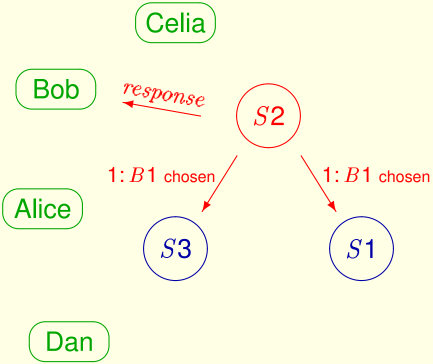
Server knows $B1$ chosen as command 1.



Chosen Commands

1: $B1$

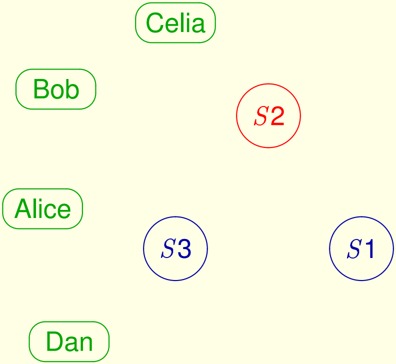
Sends response obtained by executing $B1$



Chosen Commands

1: $B1$

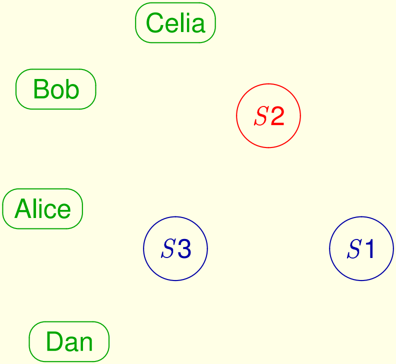
Sends response obtained by executing $B1$ and notifies servers that 1: $B1$ chosen.



Chosen Commands

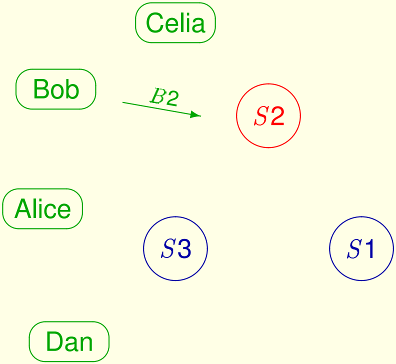
1: $B1$

From now on, I will ignore responses to clients
and notification of servers



Chosen Commands

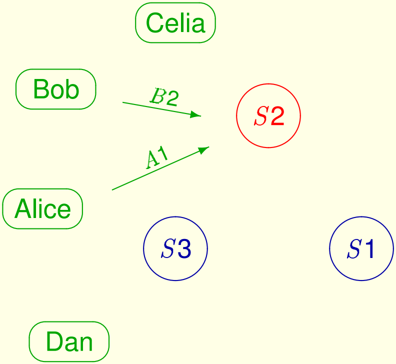
1: $B1$



Chosen Commands

1: $B1$

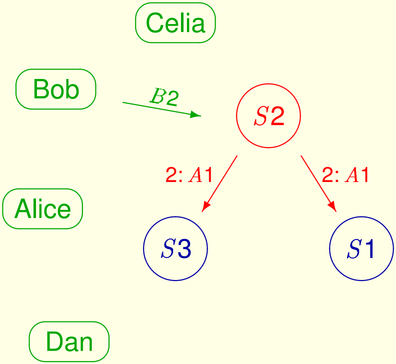
Bob issues command $B2$.



Chosen Commands

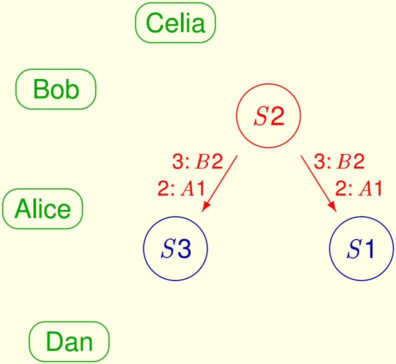
1: $B1$

Alice issues command $A1$.



Chosen Commands
1: $B1$

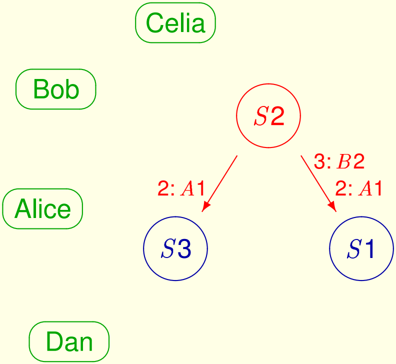
Leader assigns command number 2 to $A1$ and sends to servers.



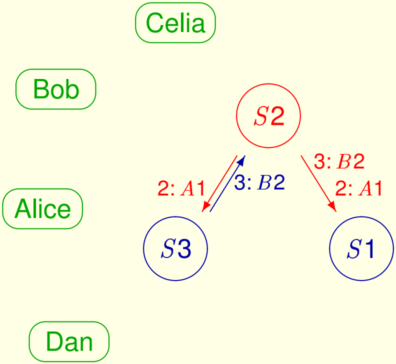
Chosen Commands

1: $B1$

Leader assigns command number 3 to $B2$ and sends to servers.



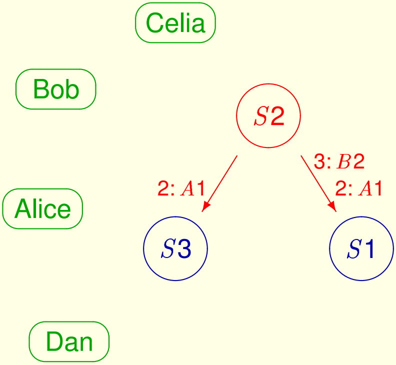
$S3$ receives 3: $B2$.



<u>Chosen Commands</u>

1: $B1$

$S3$ acks 3: $B2$.

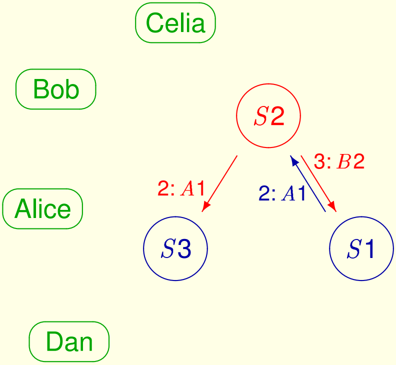


Chosen Commands

1: $B1$

3: $B2$

Leader receives 3: $B2$ and knows it is chosen.

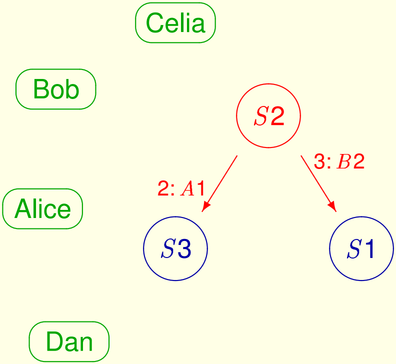


Chosen Commands

1: $B1$

3: $B2$

$S1$ receives 2: $A1$ and acks it.

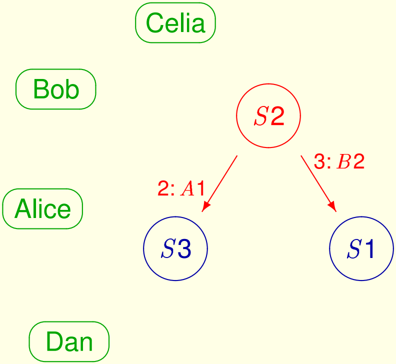


Chosen Commands

1: $B1$

3: $B2$

Leader receives ack and knows $A1$ chosen as command 2.

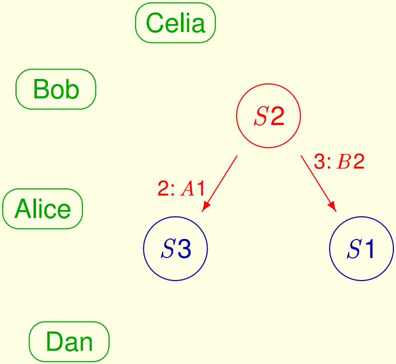


Chosen Commands

1: $B1$

2: $A1$

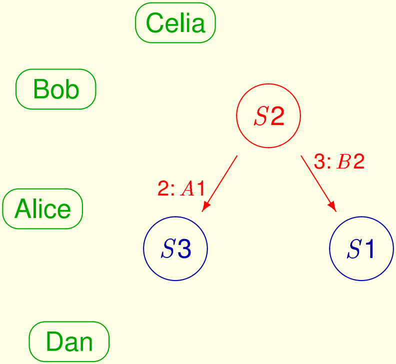
3: $B2$



Chosen Commands

1: $B1$
2: $A1$
3: $B2$

The remaining messages are redundant.

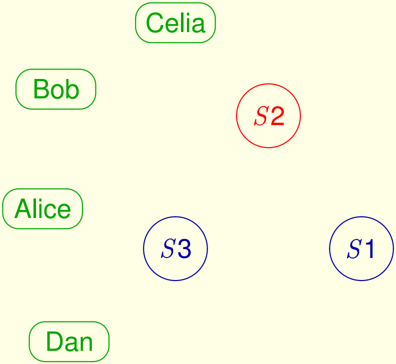


Chosen Commands

1: $B1$
2: $A1$
3: $B2$

The remaining messages are redundant.

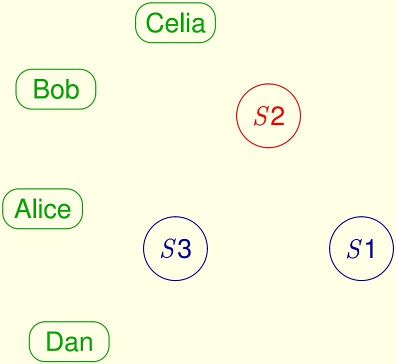
They will be acked if received, but acks are ignored.



<u>Chosen Commands</u>

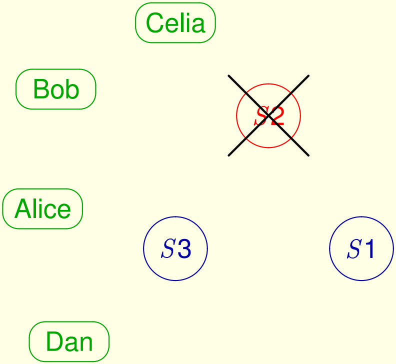
1: $B1$
2: $A1$
3: $B2$

The remaining messages are redundant.
They will be acked if received, but acks are ignored.
1-fault tolerance means leader needs only 1 ack.



Chosen Commands

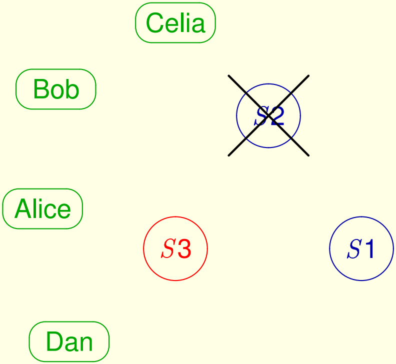
1: $B1$
2: $A1$
3: $B2$



Chosen Commands

1: $B1$
2: $A1$
3: $B2$

Suppose $S2$ fails.



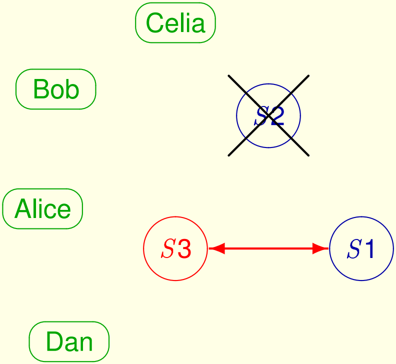
Chosen Commands

1: $B1$

2: $A1$

3: $B2$

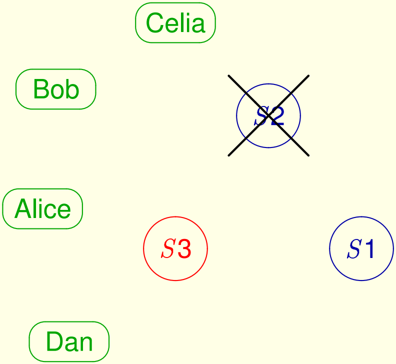
A new leader is chosen.



Chosen Commands

1: $B1$
2: $A1$
3: $B2$

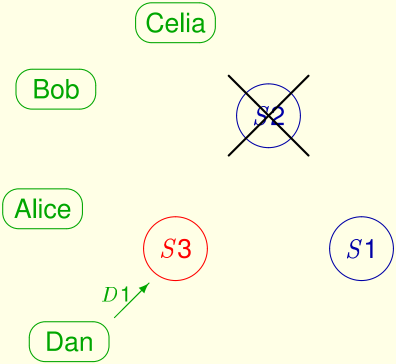
New leader learns what other servers have done.



Chosen Commands

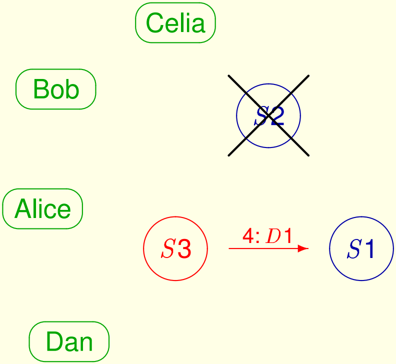
1: $B1$
2: $A1$
3: $B2$

System can new resume normal operation.



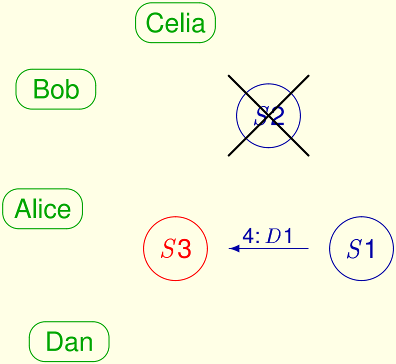
Chosen Commands

1: $B1$
2: $A1$
3: $B2$



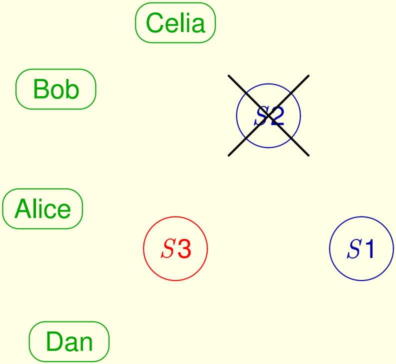
Chosen Commands

1: $B1$
2: $A1$
3: $B2$



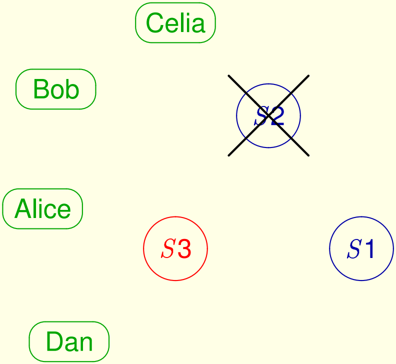
Chosen Commands

1: $B1$
2: $A1$
3: $B2$



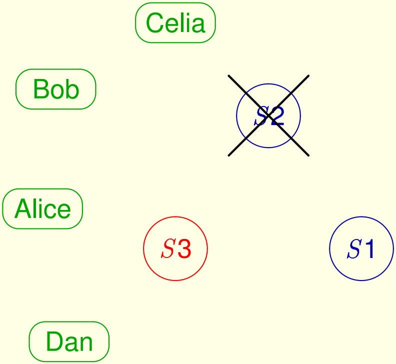
Chosen Commands

1: $B1$
2: $A1$
3: $B2$



Chosen Commands

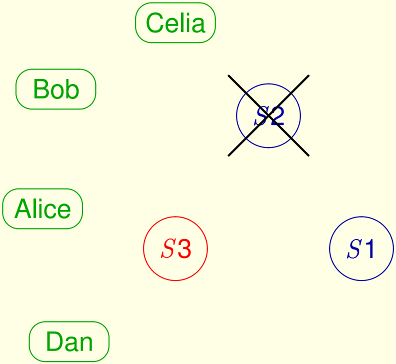
1: $B1$
2: $A1$
3: $B2$
4: $D1$



<u>Chosen Commands</u>

1: $B1$
2: $A1$
3: $B2$
4: $D1$

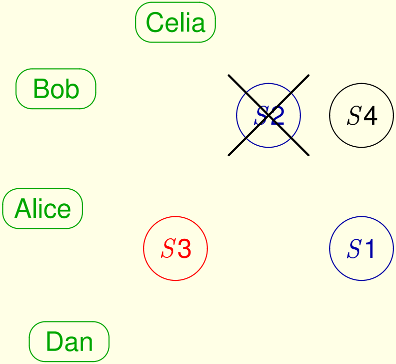
The system can no longer tolerate the failure of a server.



Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$

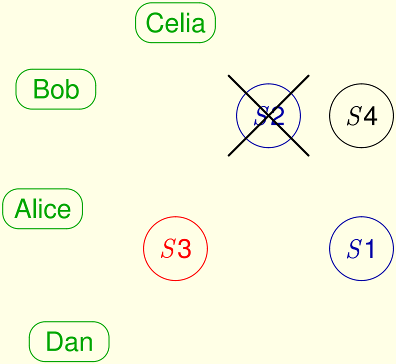
To restore 1-fault tolerance, must reconfigure — to replace $S2$ with a new server.



<u>Chosen Commands</u>

1: $B1$
2: $A1$
3: $B2$
4: $D1$

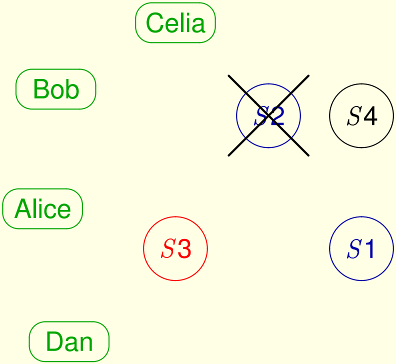
To restore 1-fault tolerance, must reconfigure —
to replace $S2$ with a new server.



Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$

Make the configuration part of the state.

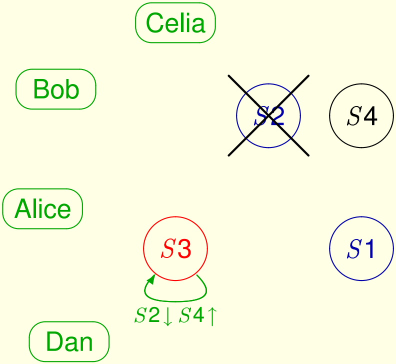


Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$

Make the configuration part of the state.

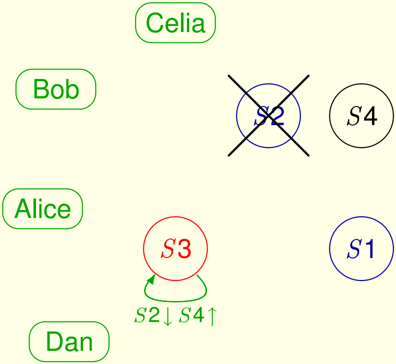
Command n chosen by configuration after command $n-1$.



Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$

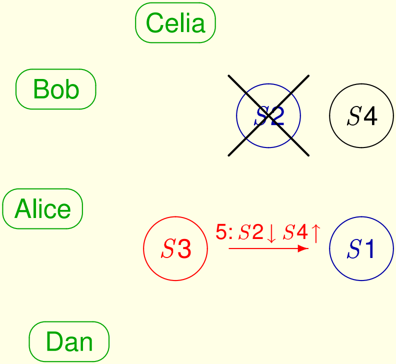
$S3$, acting as a client, issues a reconfiguration command to change the configuration by removing $S2$ and adding $S4$.



Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$

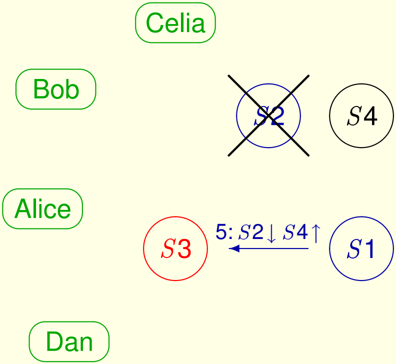
This is treated like an ordinary command.



Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$

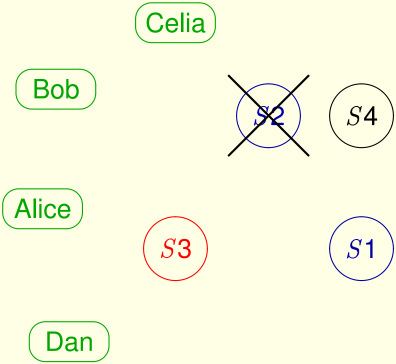
This is treated like an ordinary command.



Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$

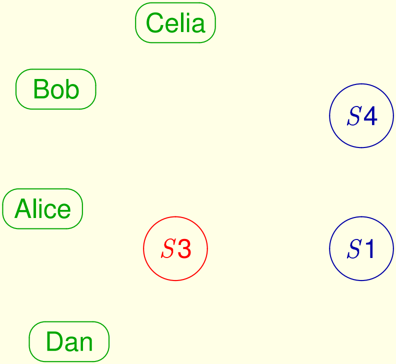
This is treated like an ordinary command.



Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2\downarrow S4\uparrow$

This is treated like an ordinary command.



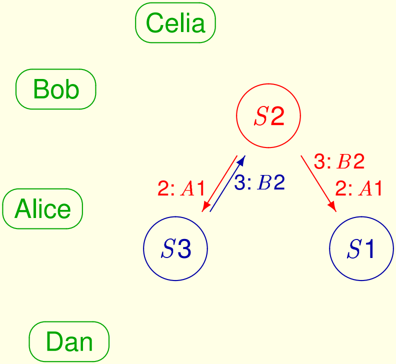
Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2\downarrow S4\uparrow$

Commands starting from number 6 chosen by $S1$, $S3$, and $S4$.

A Problem

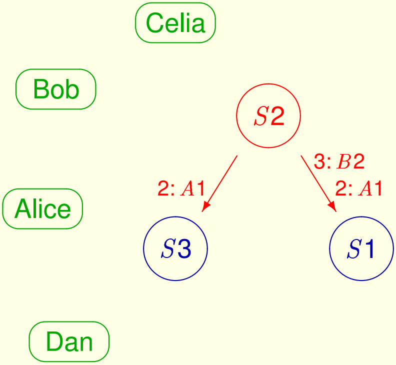
A Problem



Chosen Commands

1: $B1$

A Problem



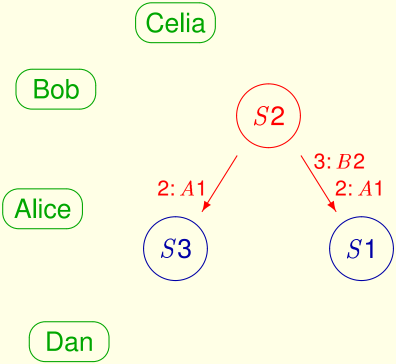
Chosen Commands

1: $B1$

3: $B2$

Leader receives 3: $B2$ from $S3$ and knows it is chosen.

A Problem



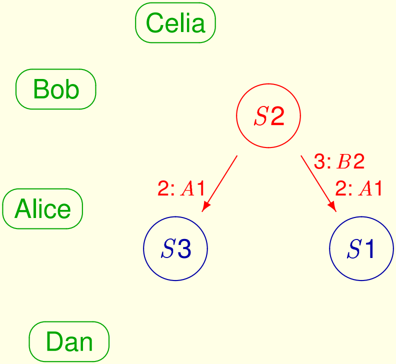
Chosen Commands

1: $B1$

3: $B2$

What if command 2 were a reconfiguration command that removed $S3$?

A Problem



Chosen Commands

1: $B1$

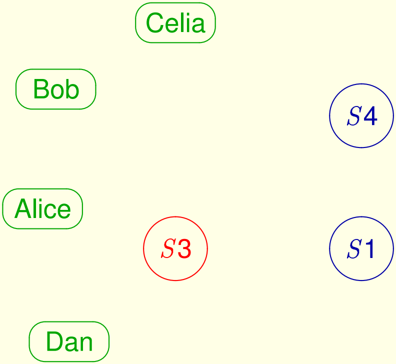
3: $B2$

What if command 2 were a reconfiguration command that removed $S3$?

We have to wait until command 2 is chosen before we can start choosing command 3.

A Solution

A Solution

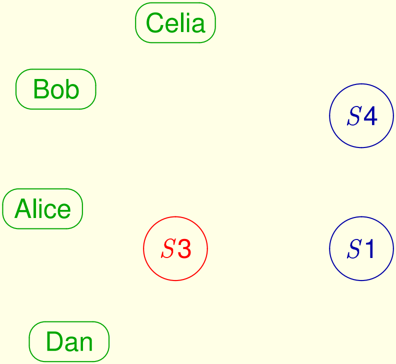


<u>Chosen Commands</u>

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2{\downarrow}\, S4{\uparrow}$

Commands starting from number 6 chosen by $S1$, $S3$, and $S4$.

A Solution

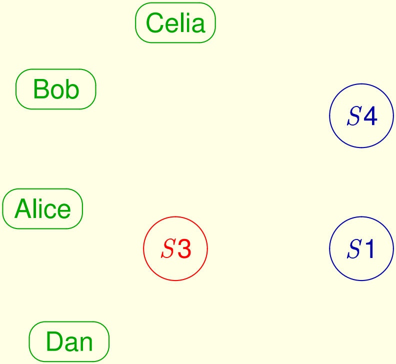


<u>Chosen Commands</u>

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2\downarrow S4\uparrow$

Commands starting from number ~~6~~ $5+\alpha$ chosen by $S1$, $S3$, and $S4$.

A Solution



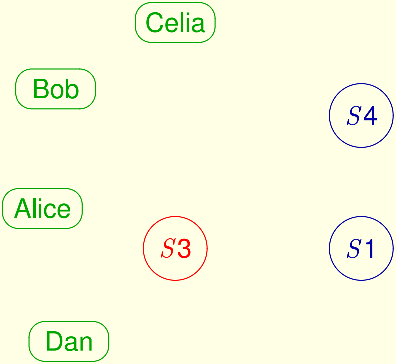
Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2\!\downarrow S4\!\uparrow$

Make the configuration part of the state.

Command n chosen by configuration after command $n-1$.

A Solution



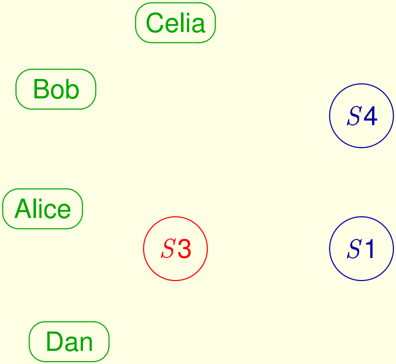
<u>Chosen Commands</u>

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2\downarrow S4\uparrow$

Make the configuration part of the state.

Command n chosen by configuration after command $n - \alpha$.

Problem

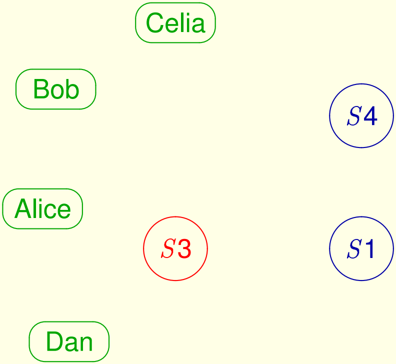


Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2\downarrow S4\uparrow$

Commands starting from number $5+\alpha$ chosen by $S1$, $S3$, and $S4$.

Problem



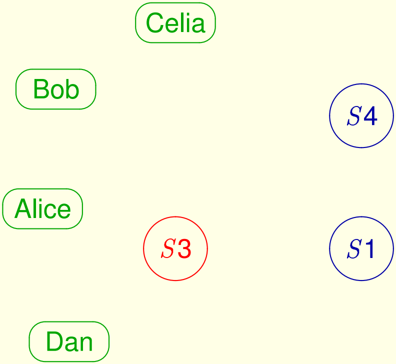
Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2\downarrow S4\uparrow$

Commands starting from number $5+\alpha$ chosen by $S1$, $S3$, and $S4$.

Must choose next $\alpha-1$ commands before reconfiguration takes effect.

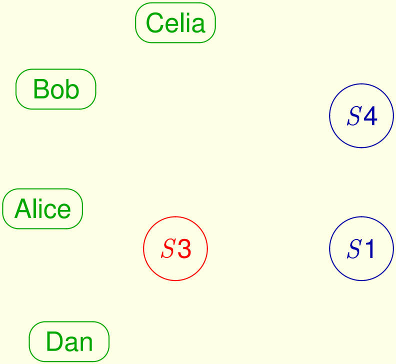
Solution



Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2\downarrow S4\uparrow$

Solution

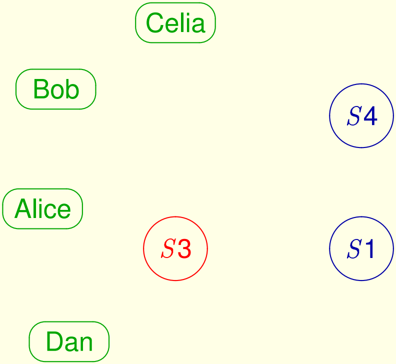


Chosen Commands

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2{\downarrow}\, S4{\uparrow}$

Immediately choose $\alpha - 1$ *no-op* commands

Solution



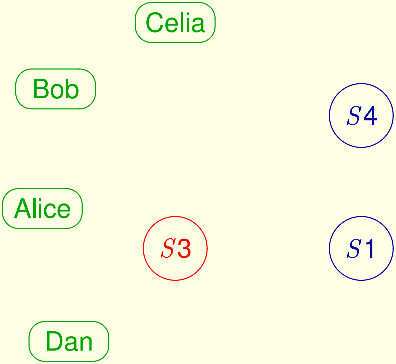
<u>Chosen Commands</u>

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2{\downarrow}\, S4{\uparrow}$

Immediately choose $\alpha - 1$ *no-op* commands

No problem letting $\alpha = 2^{32}$.

Solution



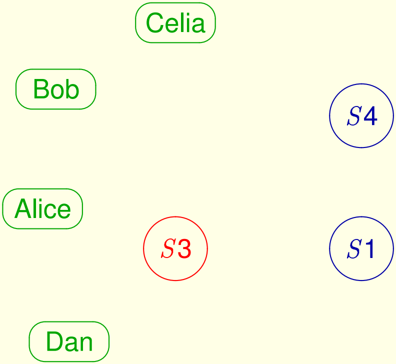
<u>Chosen Commands</u>

1: $B1$
2: $A1$
3: $B2$
4: $D1$
5: $S2{\downarrow}\, S4{\uparrow}$

Immediately choose $\alpha - 1$ *no-op* commands

No problem letting $\alpha = 2^{32}$.
All chosen with same messages as reconfiguration command.

Solution



<u>Chosen Commands</u>

1: B1
2: A1
3: B2
4: D1
5: S2$\downarrow$ S4$\uparrow$

Immediately choose $\alpha - 1$ *no-op* commands

No problem letting $\alpha = \infty$

$$command\ number = \langle epoch\ number, number \rangle$$

What About Byzantine Failures?

What About Byzantine Failures?

The only problem: a malicious node could propose a bad reconfiguration.

What About Byzantine Failures?

The only problem: a malicious node could propose a bad reconfiguration.

The solution: define the state machine so enough nodes must agree to a reconfiguration for it to take effect.

Part IIa: The Present: Vertical Paxos

Part IIa: The Present: Vertical Paxos

Joint work with Dahlia Malkhi and Lidong Zhou.

Part IIa: The Present: Vertical Paxos

Joint work with Dahlia Malkhi and Lidong Zhou.

For non-Byzantine Failures.

A Closer Look at Paxos

1 2 3 4 5 6 7 8 $\ldots$

command number

ballot number

⋮

6

5

4

3

2

1

1	2	3	4	5	6	7	8	...

command number

Each ballot number used by at most one leader.

ballot number									
⋮									
6									
5									
4									
3									
2	B1								
1									
	1	2	3	4	5	6	7	8	...

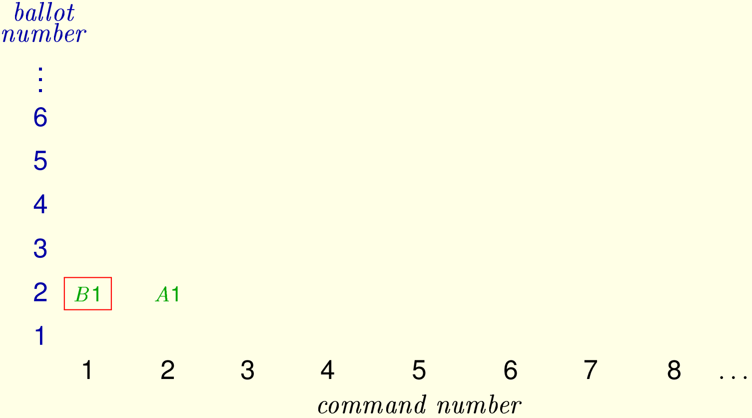
command number

S2 proposes B1 in ballot 2 of command 1.

ballot number									
⋮									
6									
5									
4									
3									
2	$\boxed{B1}$								
1									
	1	2	3	4	5	6	7	8	...

command number

$B1$ chosen in ballot 2 of command 1.

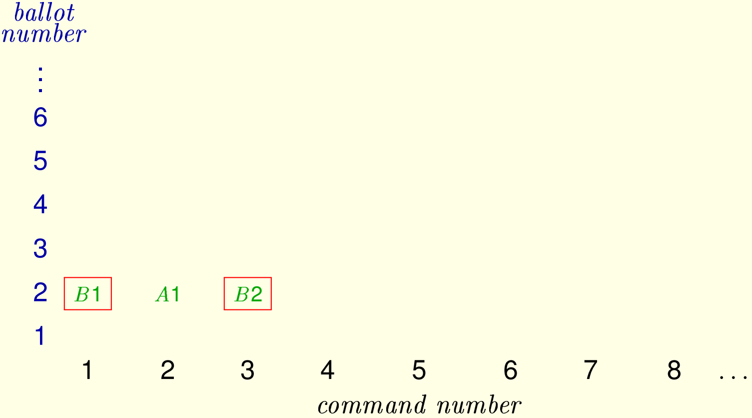


$S2$ proposes $A1$ in ballot 2 of command 2.

ballot number									
⋮									
6									
5									
4									
3									
2	$\boxed{B1}$	$A1$	$B2$						
1									
	1	2	3	4	5	6	7	8	. . .

command number

$S2$ proposes $B2$ in ballot 2 of command 3.



B2 chosen in ballot 2 of command 3.

ballot number									
⋮									
6									
5									
4									
3									
2	$\boxed{B1}$	$\boxed{A1}$	$\boxed{B2}$						
1									
	1	2	3	4	5	6	7	8	...

command number

$A1$ chosen in ballot 2 of command 2.

ballot number									
⋮									
6									
5									
4									
3									
2	$\boxed{B1}$	$\boxed{A1}$	$\boxed{B2}$	$C1$					
1									
	1	2	3	4	5	6	7	8	...

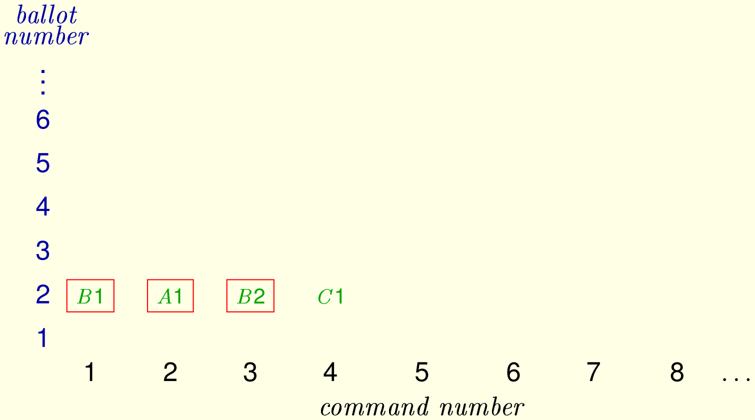
command number

$S2$ proposes $C1$ in ballot 2 of command 4.

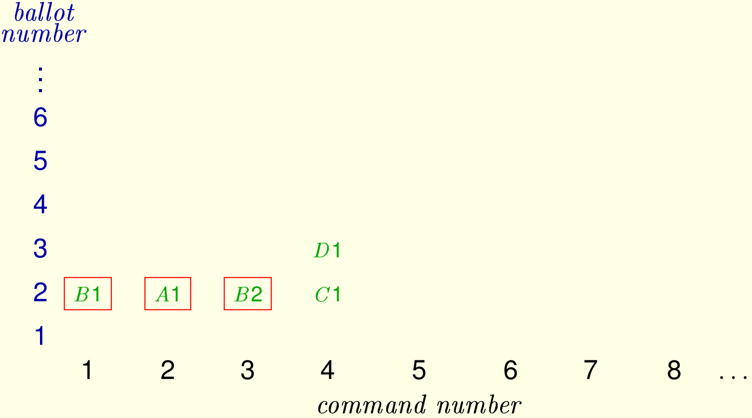
ballot number									
⋮									
6									
5									
4									
3									
2	$\boxed{B1}$	$\boxed{A1}$	$\boxed{B2}$	$C1$					
1									
	1	2	3	4	5	6	7	8	. . .

command number

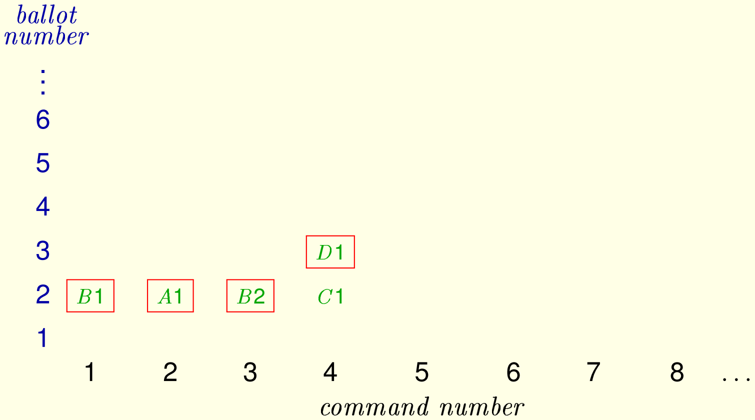
$S2$ fails.



$S3$ elected leader and chooses ballot 3.



$S3$ proposes $D1$ in ballot 3 of command 4.



$D1$ chosen in ballot 3 of command 4.

ballot number									
⋮									
6									
5									
4									
3				$\boxed{D1}$	$S2{\downarrow}\,S4{\uparrow}$				
2	$\boxed{B1}$	$\boxed{A1}$	$\boxed{B2}$	$C1$					
1									
	1	2	3	4	5	6	7	8	...

command number

$S3$ proposes $S2{\downarrow}\,S4{\uparrow}$ in ballot 3 of command 5

ballot number									
⋮									
6									
5									
4									
3				$\boxed{D1}$	$S2{\downarrow}\,S4{\uparrow}$	*no-op*	*no-op*		
2	$\boxed{B1}$	$\boxed{A1}$	$\boxed{B2}$	$C1$					
1									
	1	2	3	4	5	6	7	8	. . .

command number

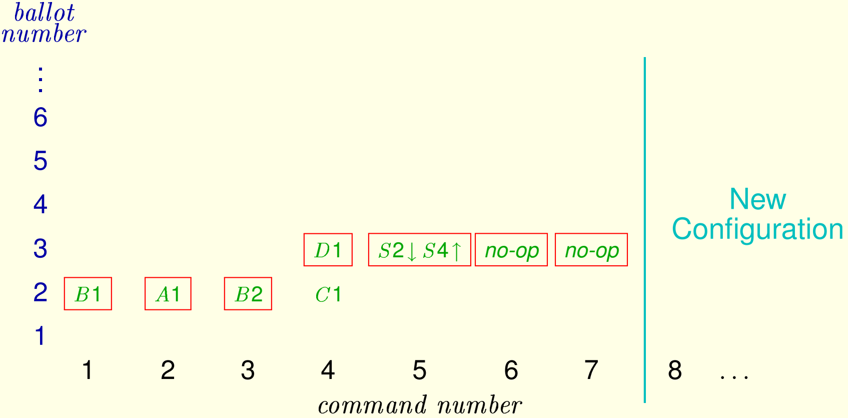
$S3$ proposes $S2{\downarrow}\,S4{\uparrow}$ in ballot 3 of command 5

and *no-op* in ballot 3 of commands 6 and 7 ($\alpha = 3$)

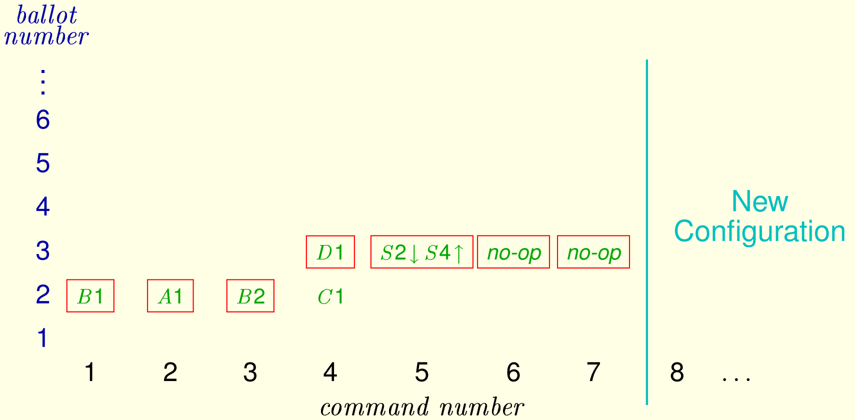
ballot number									
⋮									
6									
5									
4									
3				$\boxed{D1}$	$\boxed{S2{\downarrow}\ S4{\uparrow}}$	$\boxed{\textit{no-op}}$	$\boxed{\textit{no-op}}$		
2	$\boxed{B1}$	$\boxed{A1}$	$\boxed{B2}$	$C1$					
1									
	1	2	3	4	5	6	7	8	. . .

command number

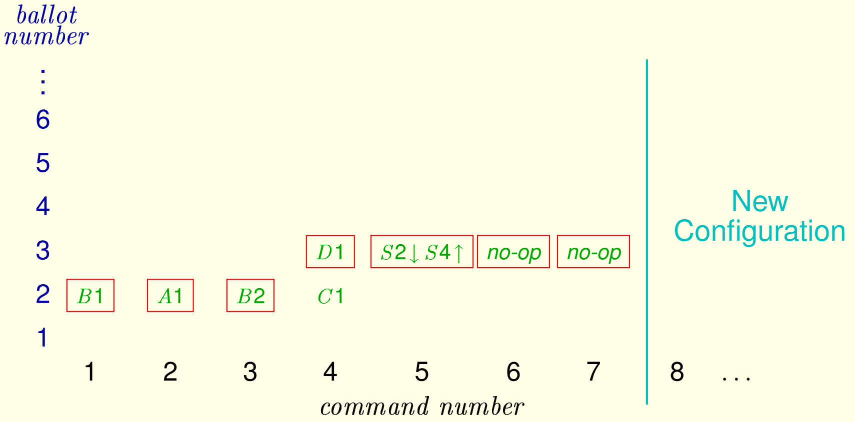
Commands number 5–7 chosen.



New configuration takes effect with command number 8.

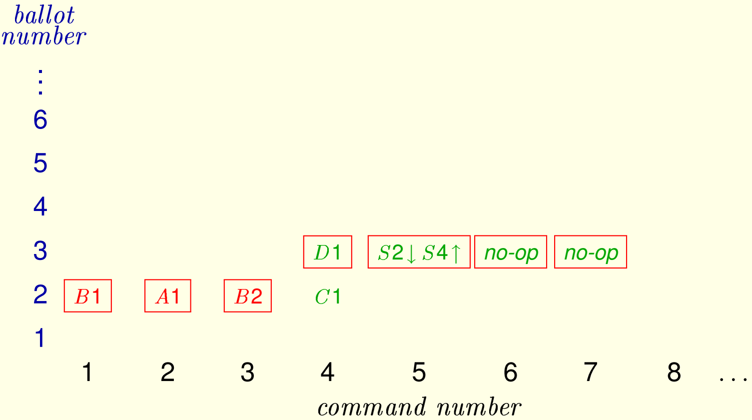


New configuration takes effect with command number 8. $(\alpha = 3)$



New configuration takes effect with command number 8. ($\alpha = 3$)

Horizontal Reconfiguration.



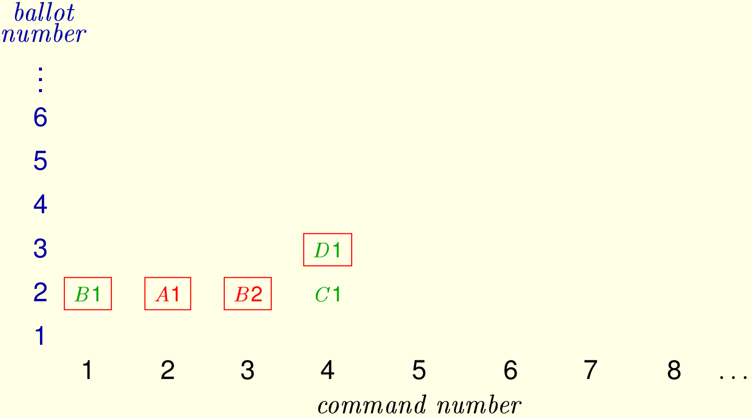
Vertical Reconfiguration

ballot number									
⋮									
6									
5									
4									
3				$\boxed{D1}$	$\boxed{S2\downarrow S4\uparrow}$	$\boxed{\textit{no-op}}$	$\boxed{\textit{no-op}}$		
2	$\boxed{B1}$	$\boxed{A1}$	$\boxed{B2}$	$C1$					
1									
	1	2	3	4	5	6	7	8	. . .

command number

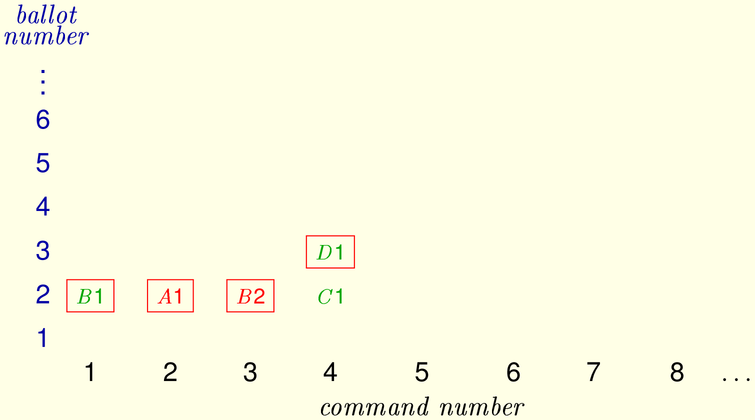
Vertical Reconfiguration

– No state-machine reconfiguration commands.



Vertical Reconfiguration

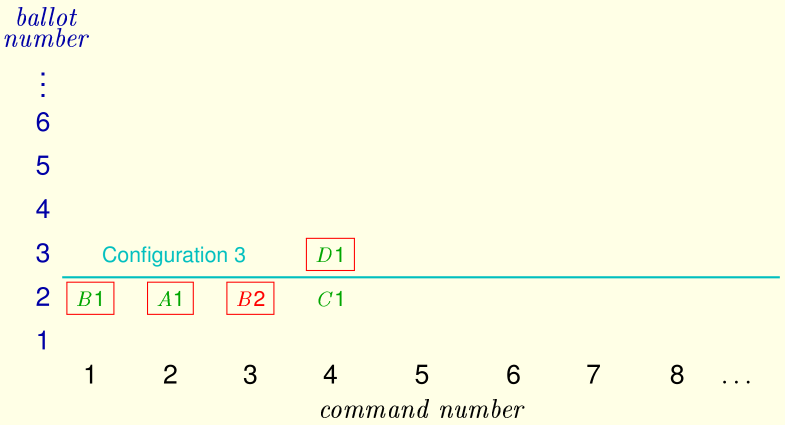
– No state-machine reconfiguration commands.



Vertical Reconfiguration

– No state-machine reconfiguration commands.

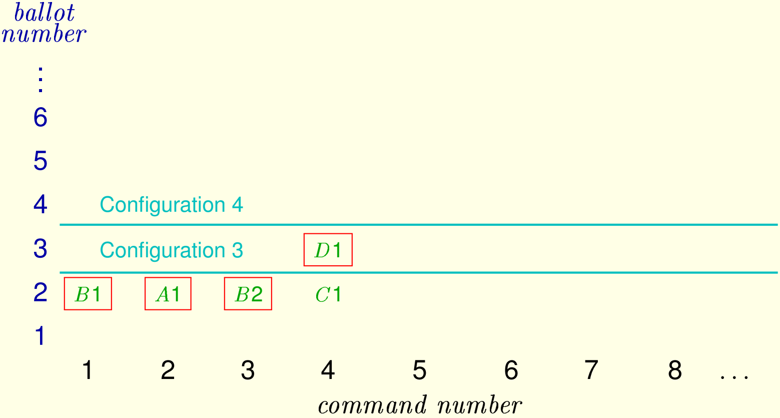
– Each ballot number uses its own configuration.



Vertical Reconfiguration

– No state-machine reconfiguration commands.

– Each ballot number uses its own configuration.

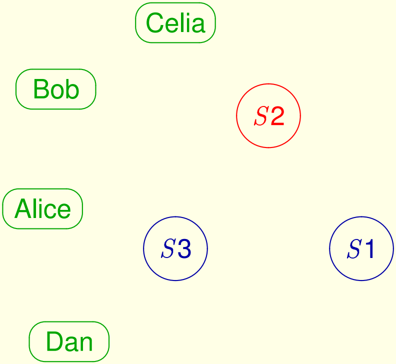


Vertical Reconfiguration

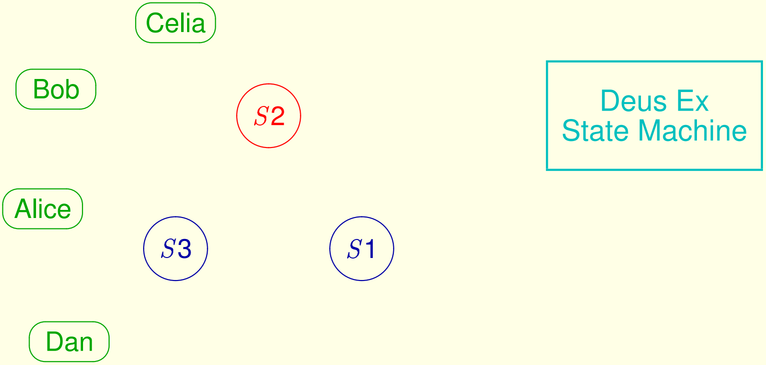
– No state-machine reconfiguration commands.

– Each ballot number uses its own configuration.

How It's Done

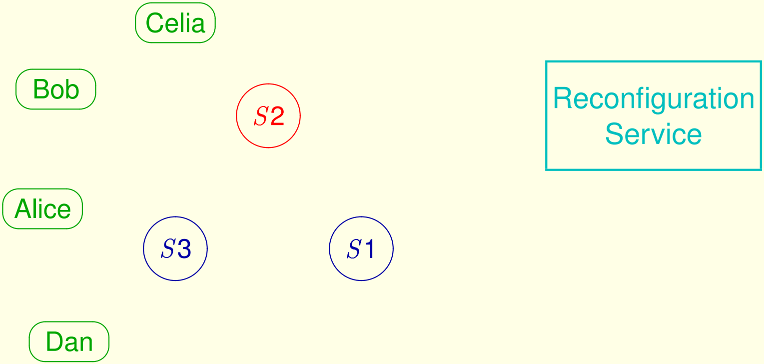


How It's Done



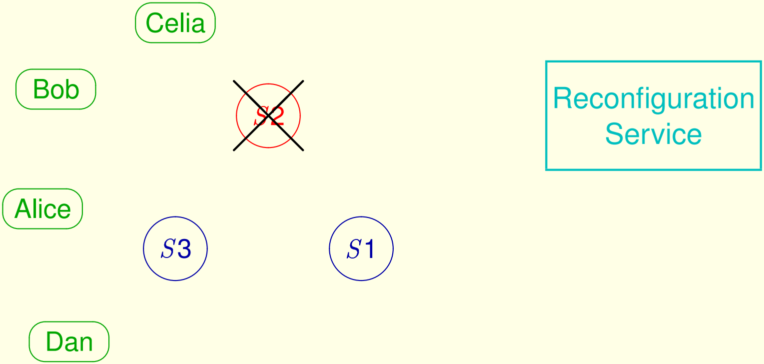
Get external help.

How It's Done



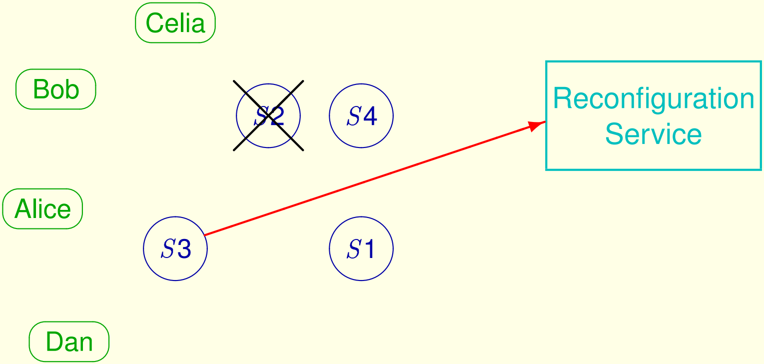
Assume a reliable reconfiguration service.

How It's Done



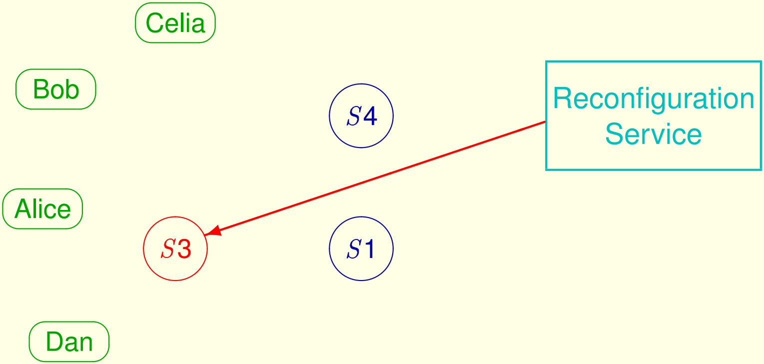
Suppose $S2$ fails.

How It's Done



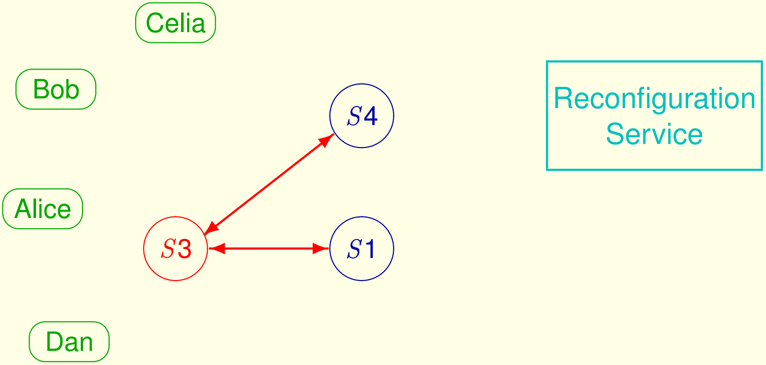
$S3$ asks to be new leader of configuration $S3$, $S1$, $S4$.

How It's Done



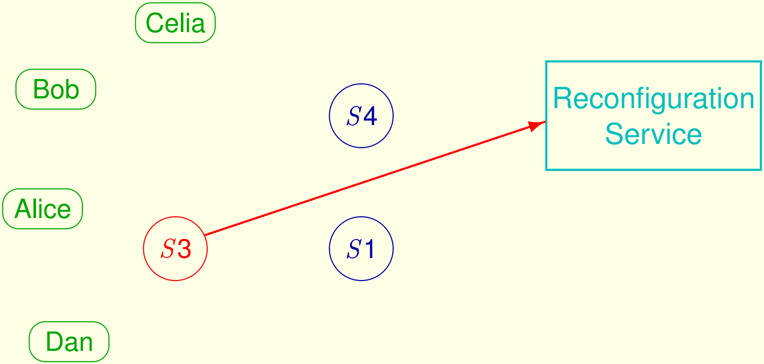
Told to try with ballot number 3.

How It's Done



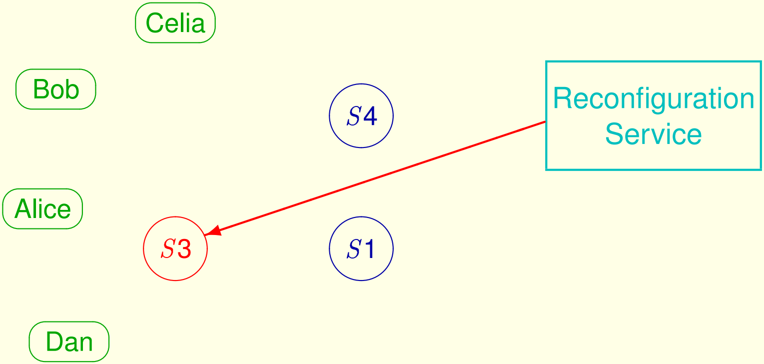
$S3$ ends ballot 2 and starts ballot 3.

How It's Done



$S3$ reports that it has successfully started ballot 3.

How It's Done



Told to resume normal operation of ballot 3.

Where Do We Get a Reconfiguration Service?

Where Do We Get a Reconfiguration Service?

Modern data centers have 10s – 1000s of computers.

Where Do We Get a Reconfiguration Service?

Modern data centers have 10s – 1000s of computers.

Many different state machines being run by many different computers.

Where Do We Get a Reconfiguration Service?

Modern data centers have 10s – 1000s of computers.

Many different state machines being run by many different computers.

The Reconfiguration Service does nothing most of the time.

Where Do We Get a Reconfiguration Service?

Modern data centers have 10s – 1000s of computers.

Many different state machines being run by many different computers.

The Reconfiguration Service does nothing most of the time.
It's needed only when there's a failure.

Where Do We Get a Reconfiguration Service?

Modern data centers have 10s – 1000s of computers.

Many different state machines being run by many different computers.

The Reconfiguration Service does nothing most of the time.
It's needed only when there's a failure.

Run the Reconfiguration Service as a reliable state machine (with horizontal reconfiguration).

Where Do We Get a Reconfiguration Service?

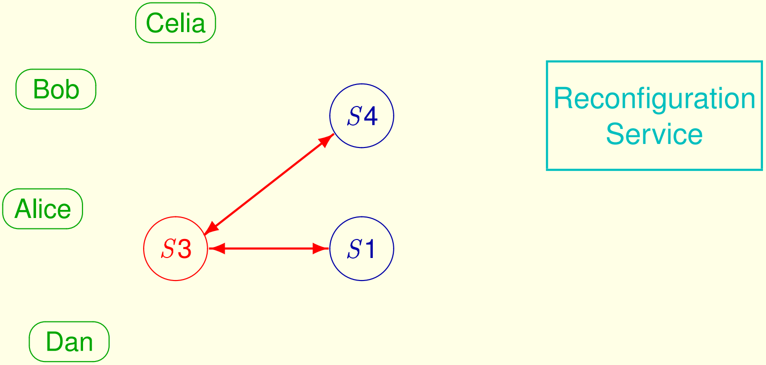
Modern data centers have 10s – 1000s of computers.

Many different state machines being run by many different computers.

The Reconfiguration Service does nothing most of the time. It's needed only when there's a failure.

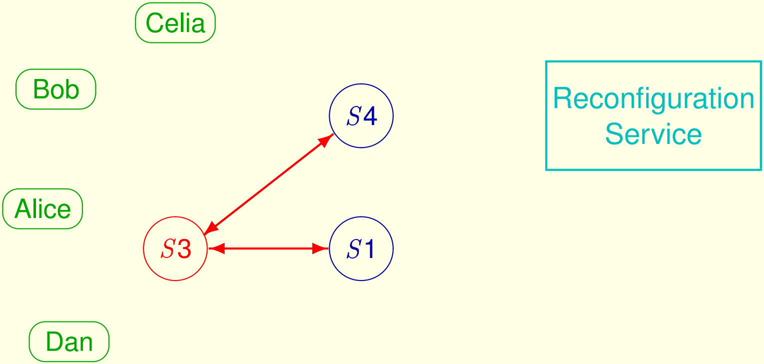
Run the Reconfiguration Service as a reliable state machine (with horizontal reconfiguration).

It can handle lots of different state machines and provide other services as well.



$S3$ ends ballot 2 and starts ballot 3.

What's Going On Here?



$S3$ ends ballot 2 and starts ballot 3.

More About How Paxos Chooses Each Single Command

More About How Paxos Chooses Each Single Command

A command C is *chosen at ballot* b iff a majority of servers vote for C in ballot b.

More About How Paxos Chooses Each Single Command

A command C is *chosen at ballot* b iff a majority of servers vote for C in ballot b.

C is *safe* at ballot b iff no command except (perhaps) C can ever be chosen at a ballot $< b$.

More About How Paxos Chooses Each Single Command

A command C is *chosen at ballot* b iff a majority of servers vote for C in ballot b.

C is *safe* at ballot b iff no command except (perhaps) C can ever be chosen at a ballot $< b$.

Observations:

1. All commands are safe at 0.

More About How Paxos Chooses Each Single Command

A command C is *chosen at ballot* b iff a majority of servers vote for C in ballot b.

C is *safe* at ballot b iff no command except (perhaps) C can ever be chosen at a ballot $< b$.

Observations:

1. All commands are safe at 0.
2. C is safe at b iff C is safe at $b-1$ and a majority of servers will never vote for any command except (perhaps) C in ballot $b-1$.

More About How Paxos Chooses Each Single Command

A command C is *chosen at ballot* b iff a majority of servers vote for C in ballot b.

C is *safe* at ballot b iff no command except (perhaps) C can ever be chosen at a ballot $< b$.

Observations:

1. All commands are safe at 0.
2. C is safe at b iff C is safe at $b-1$ and a majority of servers will never vote for any command except (perhaps) C in ballot $b-1$.
3. Different commands cannot be chosen in different ballots if a server votes in ballot b only for a command safe at b.

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it proposes in ballot b the next command it receives from a client.

Done simultaneously for all command numbers.

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it proposes in ballot b the next command it receives from a client.

Done simultaneously for all command numbers.

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it proposes in ballot b the next command it receives from a client.

Done one command number at a time.

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it proposes in ballot b the next command it receives from a client.

Observation: Can use different servers in different ballots.

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it proposes in ballot b the next command it receives from a client.

Hence, can reconfigure when starting a new ballot.

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it proposes in ballot b the next command it receives from a client.

Problem: removing reliance on ancient configurations.

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it proposes in ballot b the next command it receives from a client.

Problem: removing reliance on ancient configurations.

A new leader starts ballot b by contacting a majority of servers *in all previous ballots* and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it proposes in ballot b the next command it receives from a client.

The Solution

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

tells ballot-b servers that all values are safe; later it

In case (ii), it proposes in ballot b the next command it receives from a client.

The Solution

ballot $b-1$

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

tells ballot-b servers that all values are safe; later it

In case (ii), it proposes in ballot b the next command it receives from a client.

The Solution

ballot $b-1$

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

tells ballot-b servers that all values are safe; later it

In case (ii), it proposes in ballot b the next command it receives from a client.

The Solution

ballot $b-1$

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

tells ballot-b servers that all values are safe; later it

In case (ii), it proposes in ballot b the next command it receives from a client.

The Solution

ballot $b-1$

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

tells ballot-b servers that all values are safe;

In case (ii), it

The Solution

ballot $b-1$

A new leader starts ballot b by contacting a majority of servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it tells ballot-b servers that all values are safe.

The Solution

ballot $b-1$

A new leader starts ballot b by contacting a majority of $\wedge$ servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it tells ballot-b servers that all values are safe.

This is what a new leader does—simultaneously for all command numbers.

The Solution

ballot $b-1$

A new leader starts ballot b by contacting a majority of $\wedge$ servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

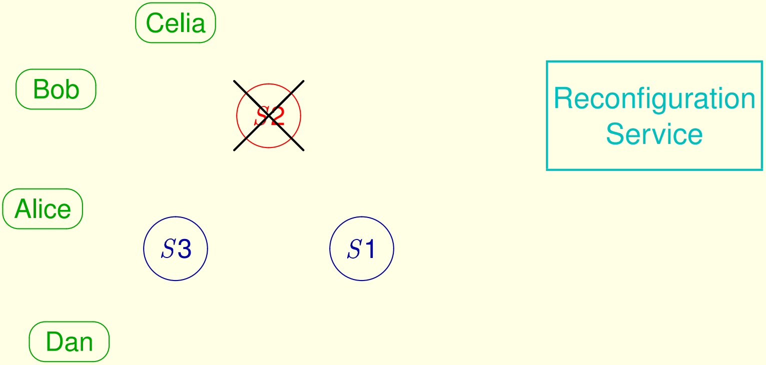
In case (i), it proposes (tells servers to vote for) C in ballot b.

In case (ii), it tells ballot-b servers that all values are safe.

This is what a new leader does—simultaneously for all command numbers.

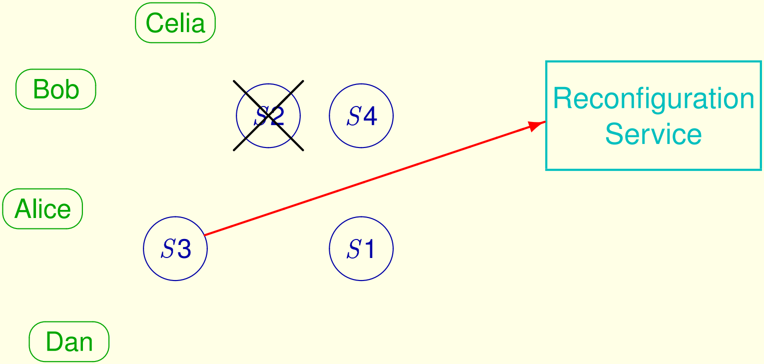
It then tells the reconfiguration service that it has successfully started ballot b.

Remember How It's Done



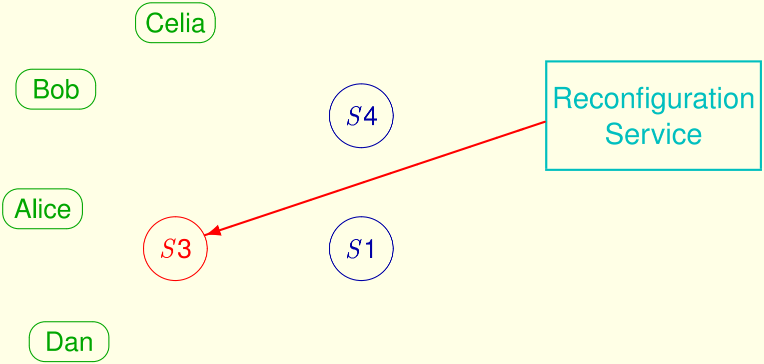
Suppose $S2$ fails.

Remember How It's Done



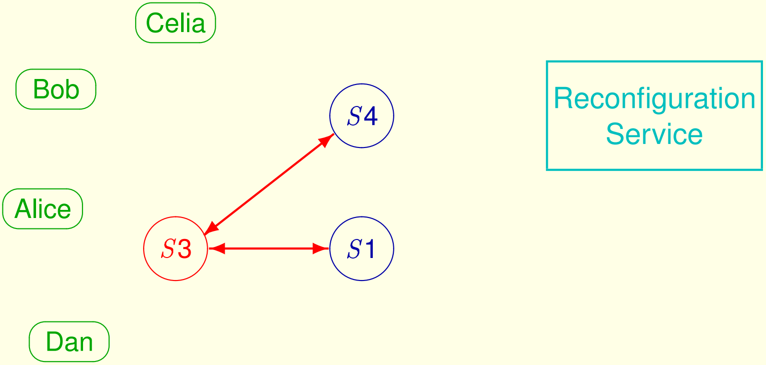
$S3$ asks to be new leader of configuration $S3$, $S1$, $S4$.

Remember How It's Done



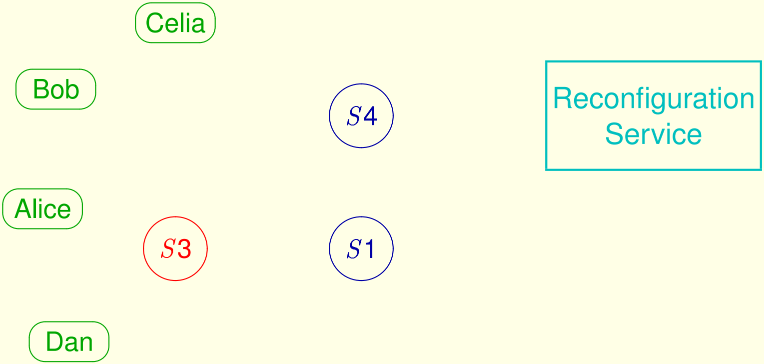
Told to try with ballot number 3.

Remember How It's Done



$S3$ ends ballot 2 and starts ballot 3.

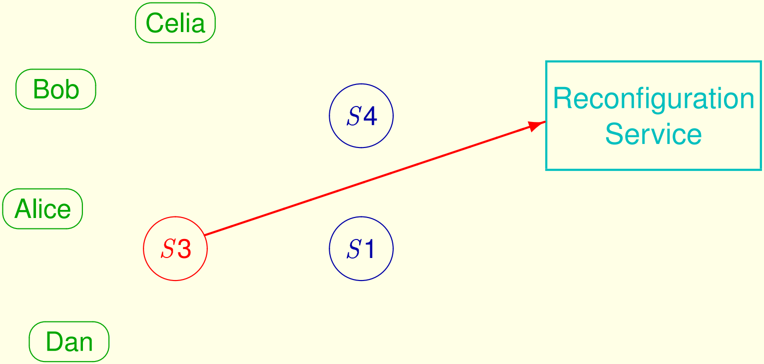
Remember How It's Done



$S3$ ends ballot 2 and starts ballot 3.

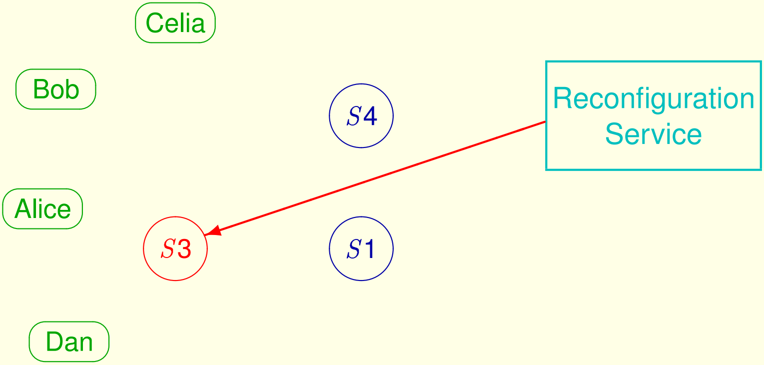
This is the processing at the beginning of ballot 3.

Remember How It's Done



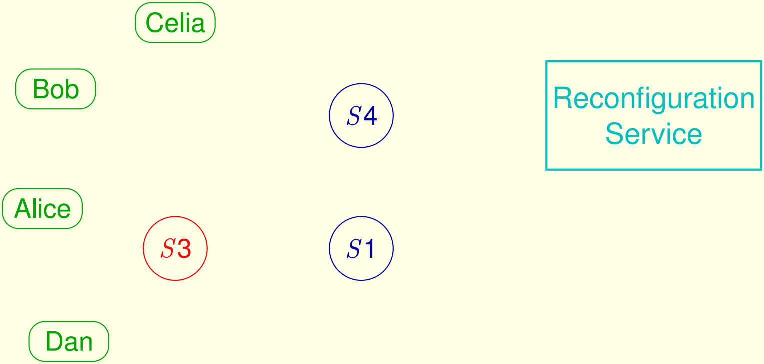
$S3$ reports that it has successfully started ballot 3.

Remember How It's Done



Told to resume normal operation of ballot 3.

Remember How It's Done



Told to resume normal operation of ballot 3.

$S3$ now proposes new client commands.

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

A command C is *chosen at ballot* b iff a majority of servers vote for C in ballot b.

C is *safe* at ballot b iff no command except (perhaps) C can ever be chosen at a ballot $< b$.

Observations:

1. All commands are safe at 0.
2. C is safe at b iff C is safe at $b-1$ and a majority of servers will never vote for any command except (perhaps) C in ballot $b-1$.
3. Different commands cannot be chosen in different ballots if a server votes in ballot b only for a command safe at b.

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

write quorum

A command C is *chosen at ballot* b iff a ~~majority~~ of servers vote for C in ballot b.

C is *safe* at ballot b iff no command except (perhaps) C can ever be chosen at a ballot $< b$.

Observations:

1. All commands are safe at 0.
2. C is safe at b iff C is safe at $b-1$ and a majority of servers will never vote for any command except (perhaps) C in ballot $b-1$.
3. Different commands cannot be chosen in different ballots if a server votes in ballot b only for a command safe at b.

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

A command C is *chosen at ballot* b iff a ~~majority~~ write quorum of servers vote for C in ballot b.

C is *safe* at ballot b iff no command except (perhaps) C can ever be chosen at a ballot $< b$.

Observations:

1. All commands are safe at 0.
2. C is safe at b iff C is safe at $b-1$ and a ~~majority~~ read quorum of servers will never vote for any command except (perhaps) C in ballot $b-1$.
3. Different commands cannot be chosen in different ballots if a server votes in ballot b only for a command safe at b.

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

A command C is *chosen at ballot* b iff a ~~majority~~ write quorum of servers vote for C in ballot b.

Requirement: write quorum $\cap \left\{\begin{array}{c}\text{read}\\ \text{write}\end{array}\right\}$ quorum $\neq \phi$

Observations:

1. All commands are safe at 0.
2. C is safe at b iff C is safe at $b-1$ and a ~~majority~~ read quorum of servers will never vote for any command except (perhaps) C in ballot $b-1$.
3. Different commands cannot be chosen in different ballots if a server votes in ballot b only for a command safe at b.

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

A command C is *chosen at ballot* b iff a ~~majority~~ write quorum of servers vote for C in ballot b.

Does little when the same configuration used for all ballots.

Observations:

1. All commands are safe at 0.
2. C is safe at b iff C is safe at $b-1$ and a ~~majority~~ read quorum of servers will never vote for any command except (perhaps) C in ballot $b-1$.
3. Different commands cannot be chosen in different ballots if a server votes in ballot b only for a command safe at b.

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

write quorum

A command C is *chosen at ballot* b iff a ~~majority~~ of servers vote for C in ballot b.

Does little when the same configuration used for all ballots.

Useful with vertical reconfiguration.

Observations:

1. All commands are safe at 0.

read quorum

2. C is safe at b iff C is safe at $b-1$ and a ~~majority~~ of servers will never vote for any command except (perhaps) C in ballot $b-1$.
3. Different commands cannot be chosen in different ballots if a server votes in ballot b only for a command safe at b.

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

write quorum

A command C is *chosen at ballot* b iff a ~~majority~~ of servers vote for C in ballot b.

Most interesting case:

write quorum = set of all servers

read quorum = any single server

Observations:

1. All commands are safe at 0.

read quorum

2. C is safe at b iff C is safe at $b-1$ and a ~~majority~~ of servers will never vote for any command except (perhaps) C in ballot $b-1$.
3. Different commands cannot be chosen in different ballots if a server votes in ballot b only for a command safe at b.

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

A command C is *chosen at ballot* b iff a ~~majority~~ write quorum of servers vote for C in ballot b.

Most interesting case:

write quorum = set of all servers

read quorum = any single server

Useless for ordinary Paxos because cannot choose any command if one server fails.

A Generalization of Ordinary Paxos (De Prisco & Lynch, 1997)

A command C is *chosen at ballot* b iff a ~~majority~~ write quorum of servers vote for C in ballot b.

Most interesting case:

write quorum = set of all servers

read quorum = any single server

Useless for ordinary Paxos because cannot choose any command if one server fails.

Not a problem with vertical reconfiguration.

The Solution

ballot $b-1$

A new leader starts ballot b by contacting a majority of $_\wedge$servers and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

The Solution

A new leader starts ballot b by contacting a ~~majority~~ read quorum of ballot $b-1$ servers
and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

The Solution

A new leader starts ballot b by contacting any ballot $b-1$ server and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

A new leader starts ballot b by contacting any ballot $b-1$ server
and learning either

(i) some (previously proposed) command C is safe at b, or

(ii) all commands are safe at b.

In most cases, the new leader will be a ballot $b-1$ server,
so it need contact only itself.

Part IIb: The Present: Primary-Backup

Tolerating f Failures

Tolerating f Failures

To make data survive f failures, you need only $f + 1$ copies.

Tolerating f Failures

To make data survive f failures, you need only $f + 1$ copies.

Primary-backup approach:

Tolerating f Failures

To make data survive f failures, you need only $f + 1$ copies.

Primary-backup approach:

– Use a primary and f backups.

Tolerating f Failures

To make data survive f failures, you need only $f + 1$ copies.

Primary-backup approach:

– Use a primary and f backups.

– If the primary fails, make a backup the new primary.

Tolerating f Failures

To make data survive f failures, you need only $f+1$ copies.

Primary-backup approach:

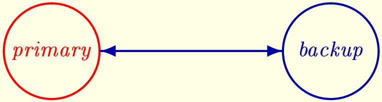
- Use a primary and f backups.
- If the primary fails, make a backup the new primary.

A popular approach (e.g., Tandem).

But consensus requires $2f + 1$ processes to tolerate f faults.

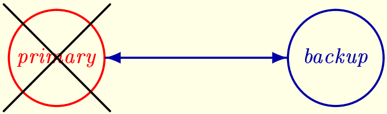
But consensus requires $2f + 1$ processes to tolerate f faults.

With a primary backup system



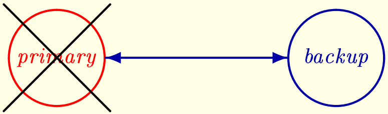
But consensus requires $2f + 1$ processes to tolerate f faults.

With a primary backup system, the backup cannot tell the difference between

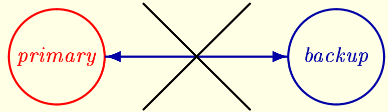


But consensus requires $2f + 1$ processes to tolerate f faults.

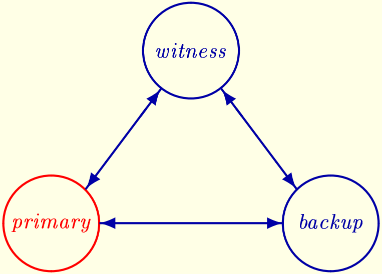
With a primary backup system, the backup cannot tell the difference between



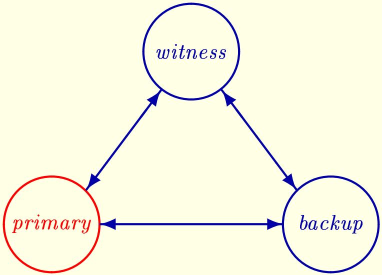
and



Requires an extra “witness” process.

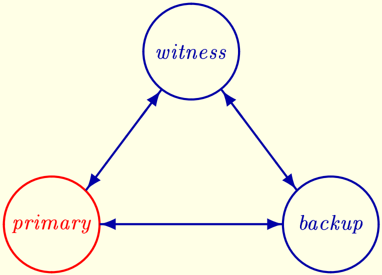


Requires an extra “witness” process.



The “witness” need not maintain the system state.

Requires an extra “witness” process.



The “witness” need not maintain the system state.

It just votes, not even caring what commands are chosen.

How Primary-Backup Systems Work

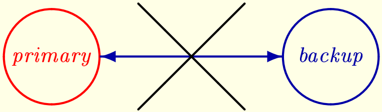
How Primary-Backup Systems Work

- Make assumptions (often implicit) about types of failures.

How Primary-Backup Systems Work

- Make assumptions (often implicit) about types of failures.

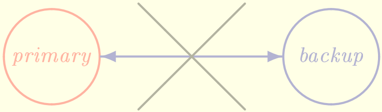
  Usually assume this can't happen:



How Primary-Backup Systems Work

- Make assumptions (often implicit) about types of failures.

  Usually assume this can't happen:

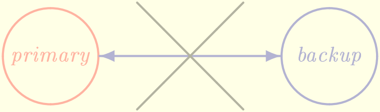


- Rely on an external service (possibly human), using an *ad hoc* algorithm

How Primary-Backup Systems Work

- Make assumptions (often implicit) about types of failures.

  Usually assume this can't happen:

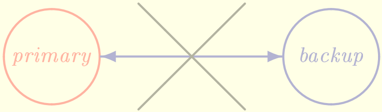


- Rely on an external service (possibly human), using an *ad hoc* algorithm (also known as code).

How Primary-Backup Systems Work

- Make assumptions (often implicit) about types of failures.

  Usually assume this can't happen:



- Rely on an external service (possibly human), using an *ad hoc* algorithm (also known as code).

  Just like the reconfiguration service of Vertical Paxos.

Vertical Paxos provides a rigorous primary-backup algorithm.

Vertical Paxos provides a rigorous primary-backup algorithm.

With proof.

Vertical Paxos provides a rigorous primary-backup algorithm.

With proof.

Vertical Paxos and Primary-Backup Replication
by Leslie Lamport, Dahlia Malkhi, and Lidong Zhou

Vertical Paxos provides a rigorous primary-backup algorithm.

With proof.

Vertical Paxos and Primary-Backup Replication
by Leslie Lamport, Dahlia Malkhi, and Lidong Zhou

Rejected by PODC 2009

Vertical Paxos provides a rigorous primary-backup algorithm.

With proof.

Vertical Paxos and Primary-Backup Replication
by Leslie Lamport, Dahlia Malkhi, and Lidong Zhou

Rejected by PODC 2009

Appeared as a short announcement at PODC 2009

Vertical Paxos provides a rigorous primary-backup algorithm.

With proof.

Vertical Paxos and Primary-Backup Replication
by Leslie Lamport, Dahlia Malkhi, and Lidong Zhou

Rejected by PODC 2009

Appeared as a short announcement at PODC 2009

Available on my publications page, accessible from
`http://lamport.org`

What About Byzantine Failures?

What About Byzantine Failures?

Vertical Paxos should generalize the same way ordinary Paxos does.

What About Byzantine Failures?

Vertical Paxos should generalize the same way ordinary Paxos does.

Should need an extra f processes everywhere, so primary-backup requires $2f + 1$.

What About Byzantine Failures?

Vertical Paxos should generalize the same way ordinary Paxos does.

Should need an extra f processes everywhere, so primary-backup requires $2f + 1$.

A PhD thesis topic?

Part III: The Cloudy Future

Part III: The Cloudy Future

- 1000s of machines, not just 10s.

Part III: The Cloudy Future

- 1000s of machines, not just 10s.

- Have to find the state machine.

Part III: The Cloudy Future

- 1000s of machines, not just 10s.
- Have to find the state machine.
- Initial configuration, not just reconfiguration.

1000s of machines

1000s of machines

Use 10s of them to execute the state machine.

1000s of machines

Use 10s of them to execute the state machine.

The rest are just clients.

1000s of machines

Use 10s of them to execute the state machine.

The rest are just clients.

Will this really work with malicious nodes?

1000s of machines

Use 10s of them to execute the state machine.

The rest are just clients.

Will this really work with malicious nodes?

We won't know until someone builds it and finds out what the engineering problems are.

The problem of a malicious leader

The problem of a malicious leader

The weak point of Byzantine Paxos

The problem of a malicious leader

The weak point of Byzantine Paxos

Will it really work?

Eliminating the Leader

Eliminating the Leader

Fatemeh Borran and André Schiper *DISC 2009*

Eliminating the Leader

Fatemeh Borran and André Schiper *DISC 2009*

L.L. *unpublished 2006*

Eliminating the Leader

Fatemeh Borran and André Schiper *DISC 2009*

L.L. *unpublished 2006*

- Implement a virtual leader as a state machine, using synchronous Byzantine agreement.

Eliminating the Leader

Fatemeh Borran and André Schiper *DISC 2009*

L.L. *unpublished 2006*

– Implement a virtual leader as a state machine, using synchronous Byzantine agreement.

– Loss of synchrony can lead to failure of agreement.

Eliminating the Leader

Fatemeh Borran and André Schiper *DISC 2009*

L.L. *unpublished 2006*

- Implement a virtual leader as a state machine, using synchronous Byzantine agreement.
- Loss of synchrony can lead to failure of agreement.
- But agreement failure just leads to malicious a leader

Eliminating the Leader

Fatemeh Borran and André Schiper *DISC 2009*

L.L. *unpublished 2006*

– Implement a virtual leader as a state machine, using synchronous Byzantine agreement.

– Loss of synchrony can lead to failure of agreement.

– But agreement failure just leads to malicious a leader, which affects only progress in Byzantine Paxos.

Finding the state machine.

Finding the state machine.

In theory, can flood the network with queries.

Finding the state machine.

In theory, can flood the network with queries.

In practice, there are engineering solutions–e.g., DNS.

Finding the state machine.

In theory, can flood the network with queries.

In practice, there are engineering solutions–e.g., DNS.

They use a trusted central service.

Finding the state machine.

In theory, can flood the network with queries.

In practice, there are engineering solutions–e.g., DNS.

They use a trusted central service.

Why not use a trusted central registration & reconfiguration service?

Initial configuration

Initial configuration

It’s easy for any process to try starting up a state machine with its friends.

Initial configuration

It’s easy for any process to try starting up a state machine with its friends.

It or a friend may not know if the start-up succeeded

Initial configuration

It's easy for any process to try starting up a state machine with its friends.

It or a friend may not know if the start-up succeeded, but it can search for the state machine.

Initial configuration

It’s easy for any process to try starting up a state machine with its friends.

It or a friend may not know if the start-up succeeded, but it can search for the state machine.

Why not just use a trusted central registration & (re)configuration service?

My Forecast: Clouds, but not so nebulous

My Forecast: Clouds, but not so nebulous

People are not going to rely on systems that form spontaneously from vapor.

My Forecast: Clouds, but not so nebulous

People are not going to rely on systems that form spontaneously from vapor.

Computing “clouds” will use trusted central services for finding, starting, and possibly reconfiguring services.

My Forecast: Clouds, but not so nebulous

People are not going to rely on systems that form spontaneously from vapor.

Computing "clouds" will use trusted central services for finding, starting, and possibly reconfiguring services.

Those trusted services may be provided by Microsoft, Google, the U.N., . . .

My Forecast: Clouds, but not so nebulous

People are not going to rely on systems that form spontaneously from vapor.

Computing “clouds” will use trusted central services for finding, starting, and possibly reconfiguring services.

Those trusted services may be provided by Microsoft, Google, the U.N., . . .

We have lots of ideas about how to build those trusted services.

THANK YOU