Formally verifying Coco

George Pîrlea

About me

Background:

- formal verification
- distributed systems

- currently undergrad at UCL
- looking to do a PhD

Current focus:

- blockchain consensus protocols

Research motivation:

- trustworthy censorship-resistant digital infrastructure

This work

- Created a formal model of Coco in F⋆
  - node-local behaviour of crash-fault-tolerant replication (CFTR)
  - core Coco service: commands, user-defined application, shared log

- Proved some high-level integrity properties
  - any CFTR protocol which satisfies the spec is *correct*
  - Coco, using such a protocol, is **sequentially consistent**

Outline

- model break-down
  - focus on highlighting assumptions
- sequential consistency
  - prerequisites
  - visual proof
- conclusions & future work

Model

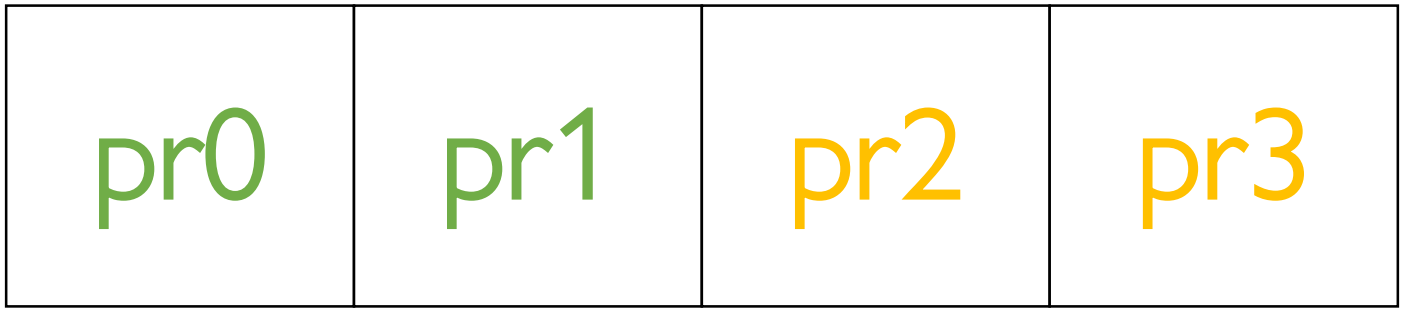
- crash-fault-tolerant consensus
  - described in terms of the local behaviour at each node

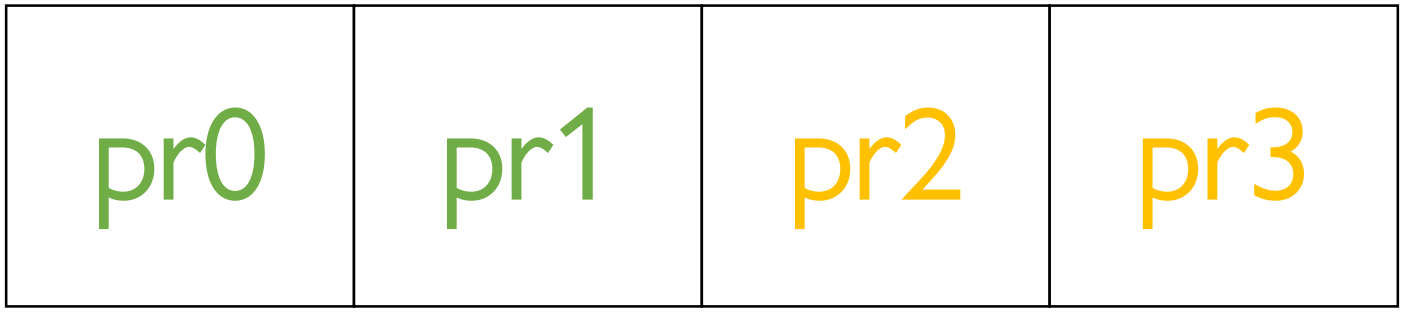
- Coco running “on top” of the CFTR protocol
  - each Coco node:
    - participates in the consensus protocol
    - runs the user-defined service (application)
    - can respond to client commands

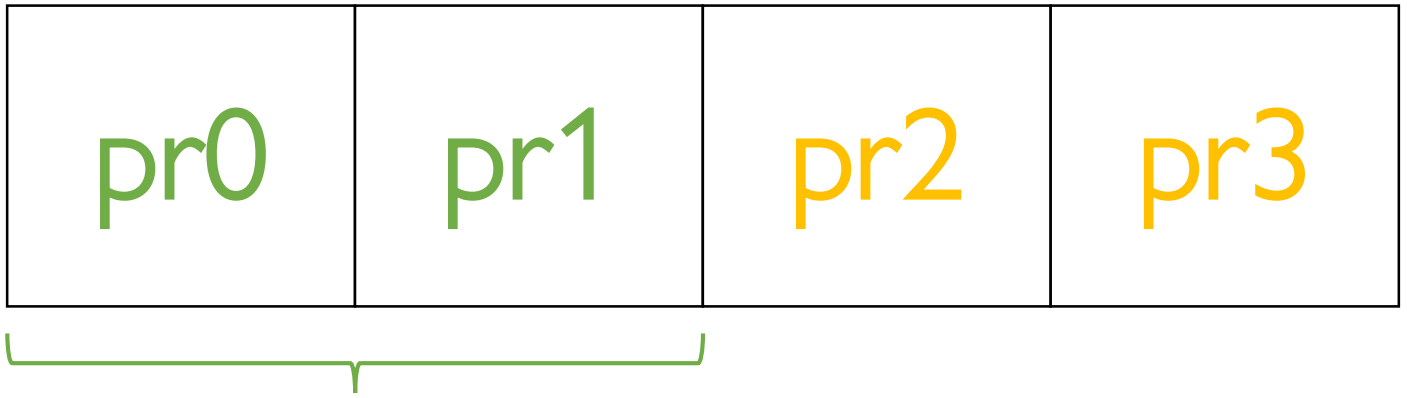
} differs from implementation

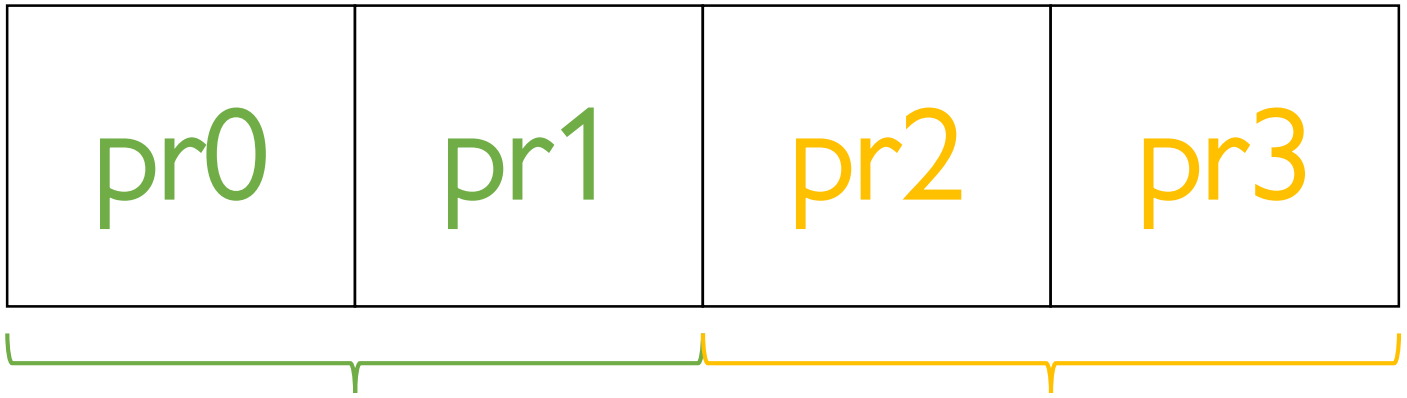
Crash-fault-tolerant replication

Specification of the node-local behaviour of a consensus protocol

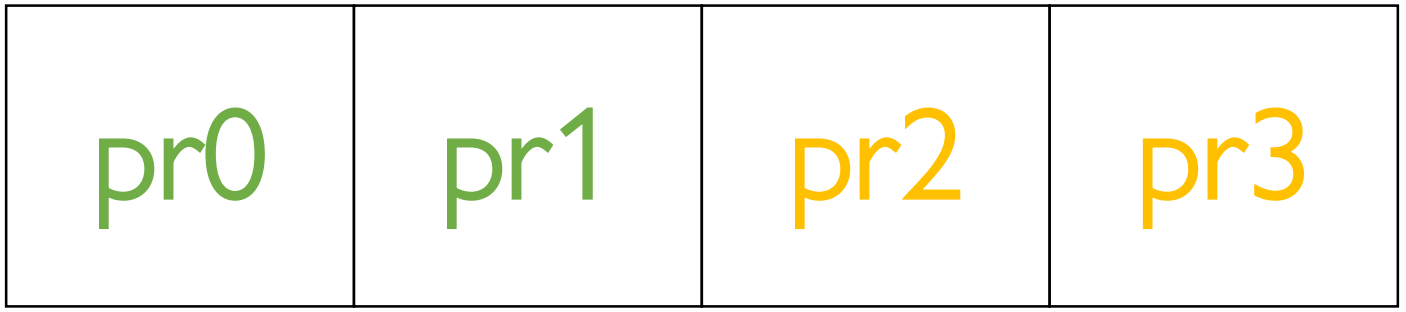

pr0
pr1
pr2
pr3


pr0
pr1
pr2
pr3


pr0
pr1
pr2
pr3
committed part


pr0
pr1
pr2
pr3
committed part
uncommitted part

Extend

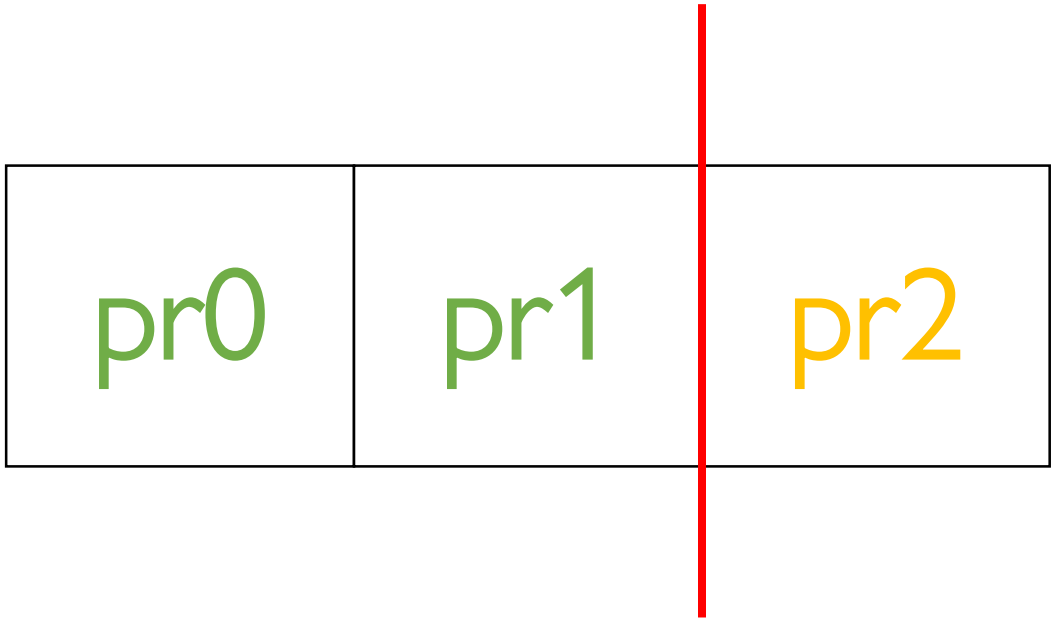


Extend

Truncate

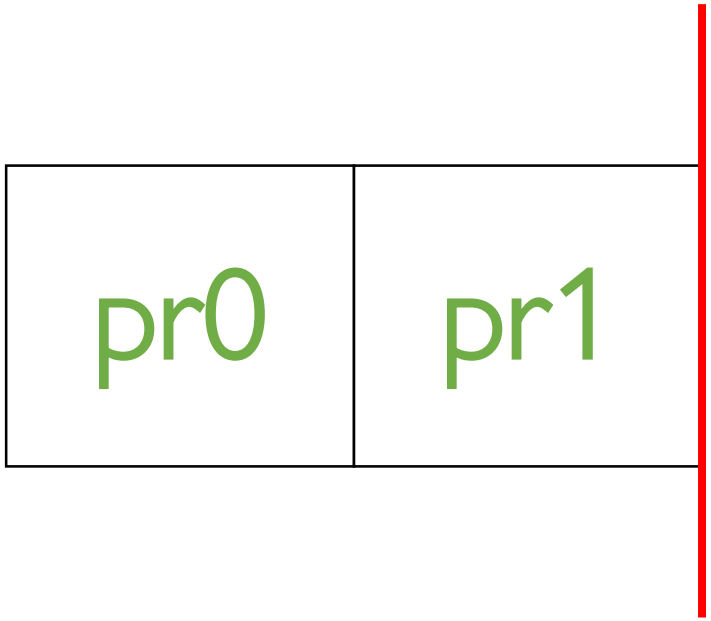
Truncate

Truncate



can’t truncate
committed entries

Truncate



can’t truncate
committed entries

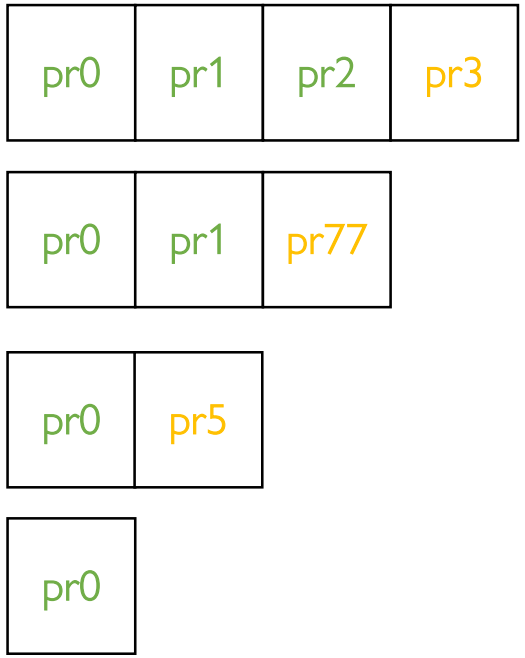
Commit

Commit

local log

Commit

local log

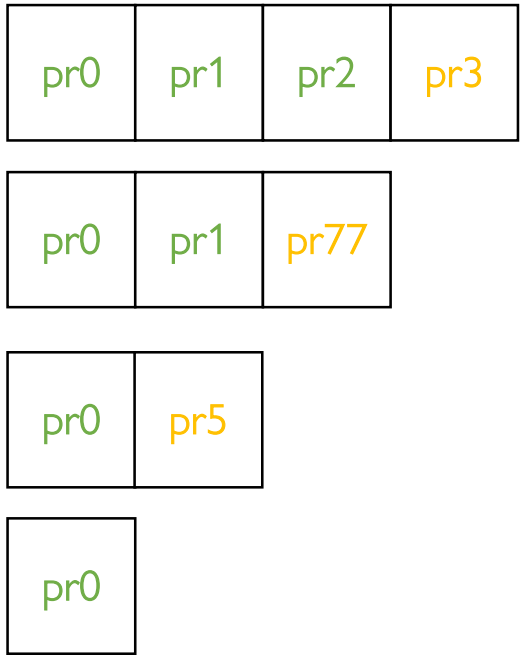


other logs

Commit

committed part consistent
across all nodes

local log

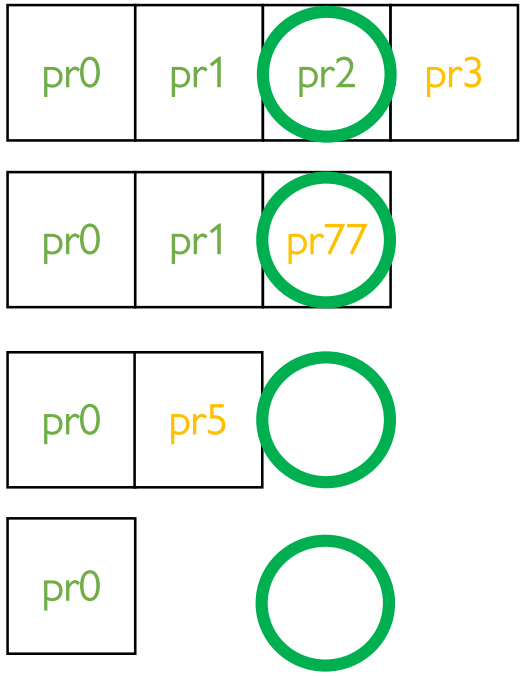


other logs

Commit

committed part consistent
across all nodes

local log

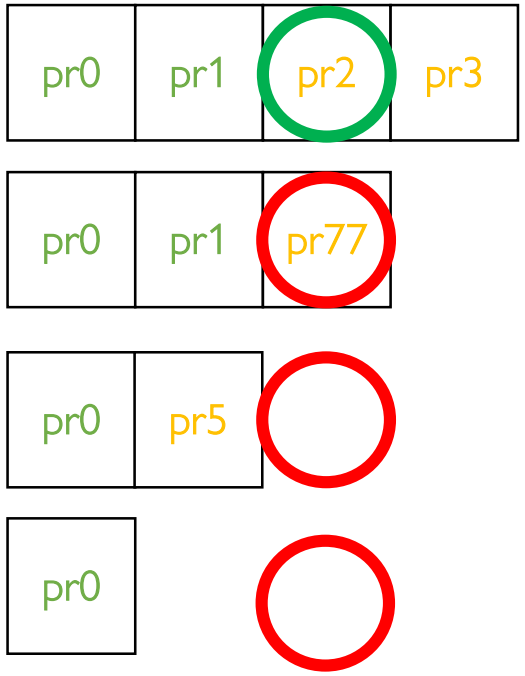


other logs

Raft satisfies a stronger CFTR spec

majority have committed part

local log

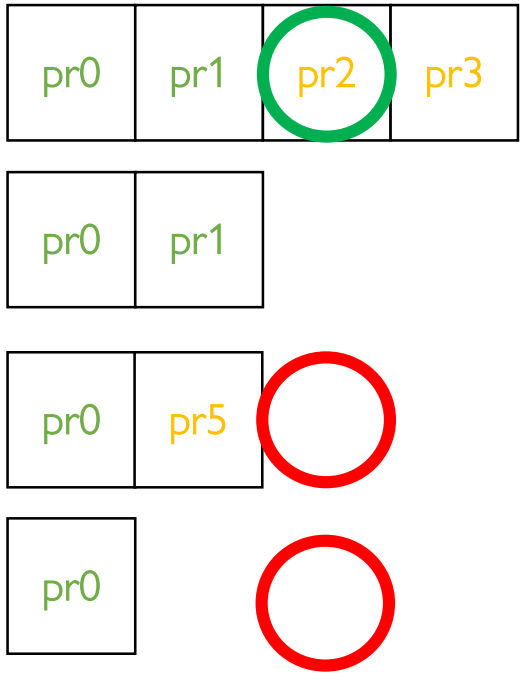


other logs

Raft satisfies a stronger CFTR spec

majority have committed part

local log

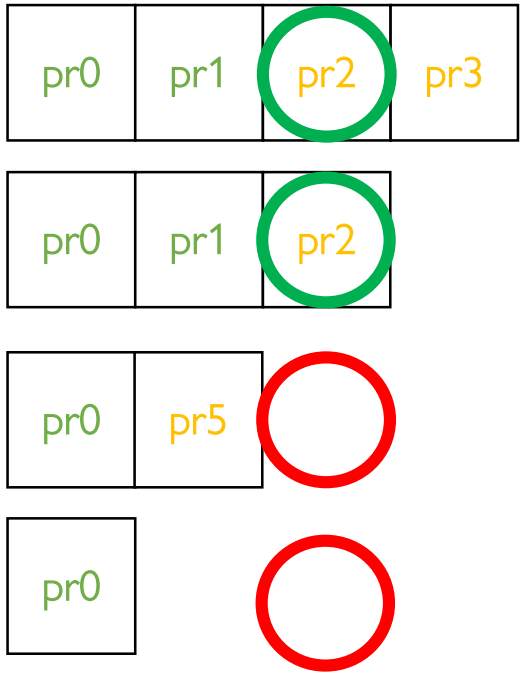


other logs

Raft satisfies a stronger CFTR spec

majority have committed part

local log

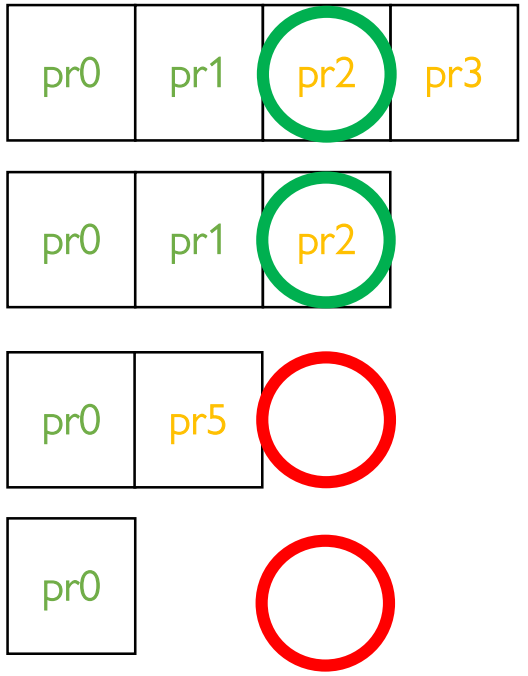


other logs

Raft satisfies a stronger CFTR spec

majority have committed part

local log



other logs

CFTR model

- replication system is a network of nodes
- nodes undergo local transitions (extend, truncate, commit)
  - transitions are relations between network states

- users can inject proposals into the system (propose)
- *ghost* global commit log (GCL)

CFTR model

- replication system is a network of nodes
- nodes undergo local transitions (extend, truncate, commit)
  - transitions are relations between network states

- users can inject proposals into the system (propose)
- *ghost* global commit log (GCL)

state required for the proofs; doesn't exist in implementation

CFTR model

- replication system is a network of nodes
- nodes undergo local transitions (extend, truncate, commit)
  - transitions are relations between network states

- users can inject proposals into the system (propose)
- *ghost* global commit log (GCL)

CFTR correctness

- there are no forks in the committed part of the logs
- a node has the global commit log as its local log
  - every other committed log is a prefix
- all entries in the log correspond to a proposal

Coco_{m}

Our model of Coco, which (hopefully) over-approximates a subset of Coco's real behaviours

What is $Coco_m$?

- User-defined service running “on top” of CFTR

```
eval: log -> kvState
service: command -> kvState -> (tx × result)
```

  - assuming commands have some unique identifier
  - assuming different commands, when run on the same kvState, produce different transactions (extensions to the log)

What can Coco$_m$ do?

Any Coco$_m$ node can:

1. Run a CFTR transition (extend, truncate or commit)

2. Process a client command

What can $Coco_m$ do?

Any $Coco_m$ node can:

1. Run a CFTR transition (extend, truncate or commit)

2. Process a client command

   a) eval log → kvS

What can Coco$_m$ do?

Any Coco$_m$ node can:

1. Run a CFTR transition (extend, truncate or commit)

2. Process a client command
   a) eval log → kvS
   b) service cmd kvS → (tx, result)

What can $Coco_m$ do?

Any $Coco_m$ node can:

1. Run a CFTR transition (extend, truncate or commit)

2. Process a client command
   a) eval log → kvS
   b) service cmd kvS → (tx, result)
   c) serialize_tx tx → propose to CFTR

What can Coco$_m$ do?

Any Coco$_m$ node can:

1. Run a CFTR transition (extend, truncate or commit)

2. Process a client command
   a) eval log → kvS
   b) service cmd kvS → (tx, result)
   c) serialize_tx tx → propose to CFTR
   d) return result to client

What can Coco$_m$ do?

Any Coco$_m$ node can:

1. Run a CFTR transition (extend, truncate or commit)

2. Process a client command
   a) eval log → kvS
   b) service cmd kvS → (tx, result)
   c) serialize_tx tx → propose to CFTR
   d) return result to client

assuming one atomic step

assuming correct transaction serializer & deserializer

assuming proposals uniquely identify the log which they extend

What can Coco$_m$ do?

In implementation, Raft forces us to allow transaction processing only on the leader (at least for writes).

Any Coco$_m$ node can:

1. Run a CFTR transition (extend, truncate or commit)

2. Process a client command
   a) eval log → kvS
   b) service cmd kvS → (tx, result)
   c) serialize_tx tx → propose to CFTR
   d) return result to client

assuming one atomic step

assuming correct transaction serializer & deserializer

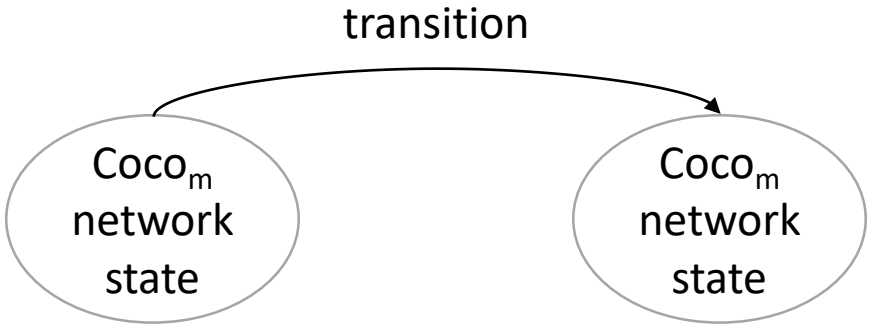
assuming proposals uniquely identify the log which they extend

Proof infrastructure

It’s proofs all the way down

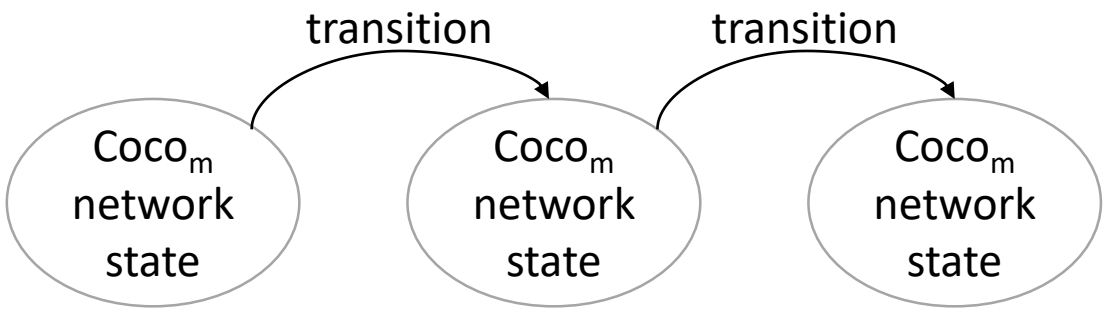
Transitions, invariants

- Transitions are modelled as inductive data types (relations)



- An **invariant** is a property that holds over transitions

Instant vs. history properties



Network history: sequence of network states linked by valid transitions

- **Instant**: *"there are no inconsistencies in the committed logs"*
- **History**: *"if a command is committed at some point, it is committed at all later points"*

$Coco_m$ instants

Model:

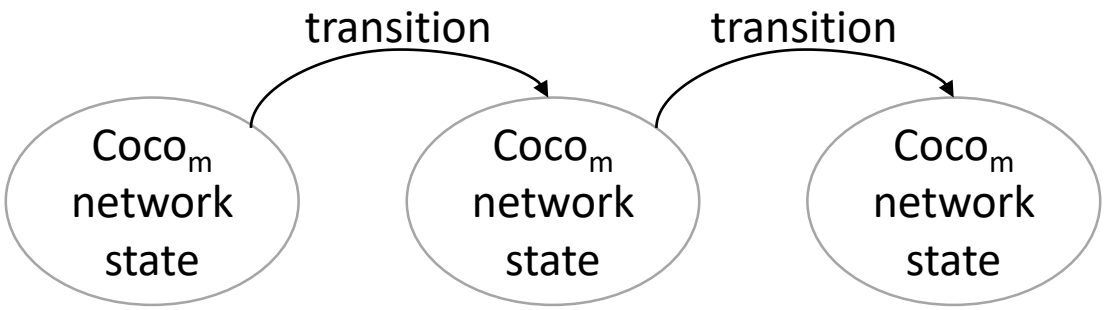
- network of nodes, each running their own replication node
- ghost state:
  - processed commands
  - mapping from commands to proposals and vice-versa

Proofs:

- the ghost state is internally consistent
- if a command is committed in state *m*, and you undergo a transition, it remains committed in state *m'*
- commands are not created committed

$Coco_m$ histories

- history = non-empty list of $Coco_m$ instants



Definitions

Instant properties:

- **committed** *cmd* = processed & included in GCL

Definitions

Instant properties:

- **committed** *cmd* = processed & included in GCL

History properties:

- **sequentially committed** *cmds* = committed at strictly increasing indices in the history

Definitions

Instant properties:

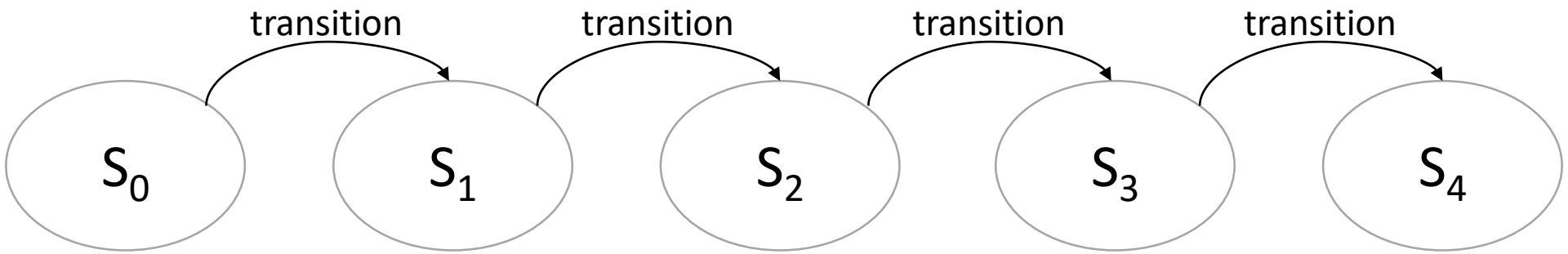
- **committed** *cmd* = processed & included in GCL

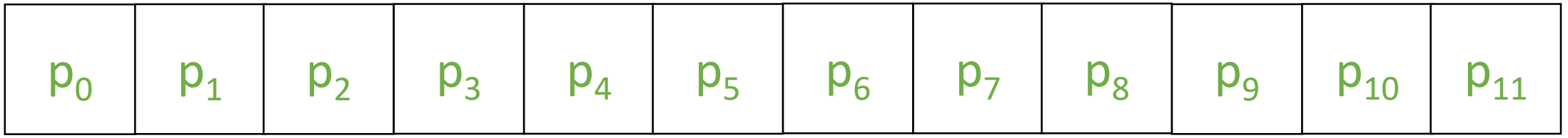
History properties:

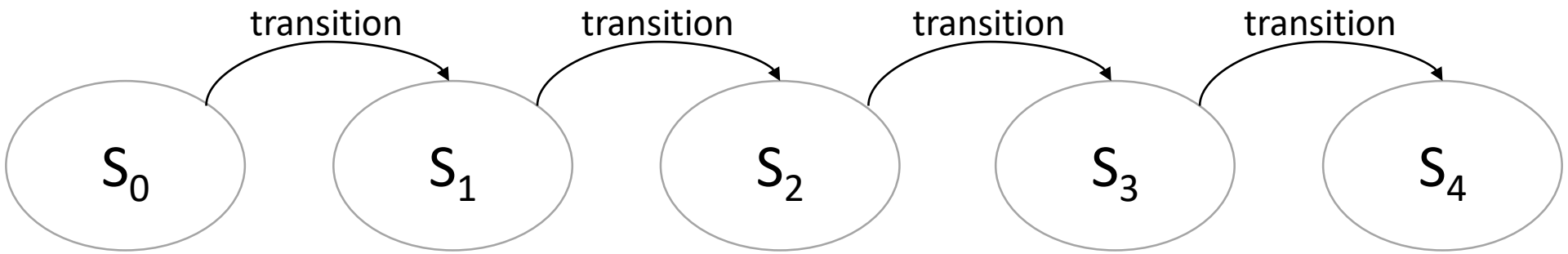
- **sequentially committed** *cmds* = committed at strictly increasing indices in the history
- **sequentially consistent** *cmds hist* = if *cmds* is sequentially committed, then the corresponding proposals form a subsequence in the GCL of the last instant in *hist*

Sequential consistency

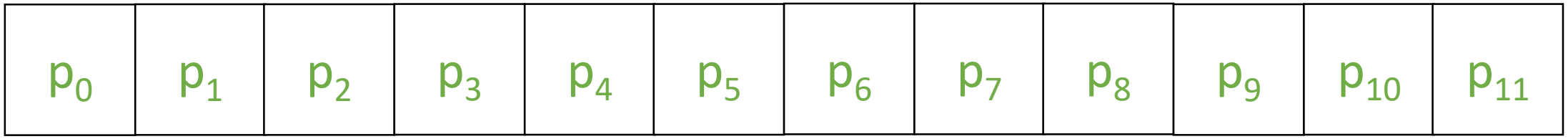
With pictures!

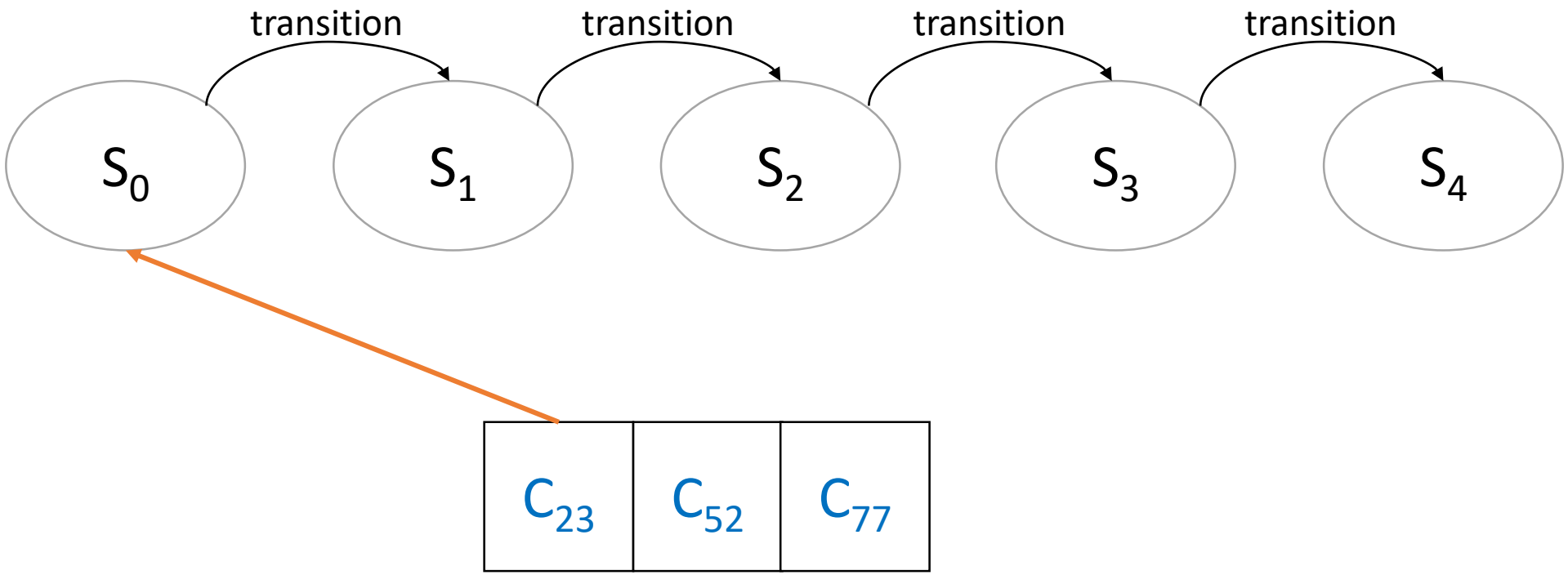

transition
transition
transition
transition
S_0
S_1
S_2
S_3
S_4

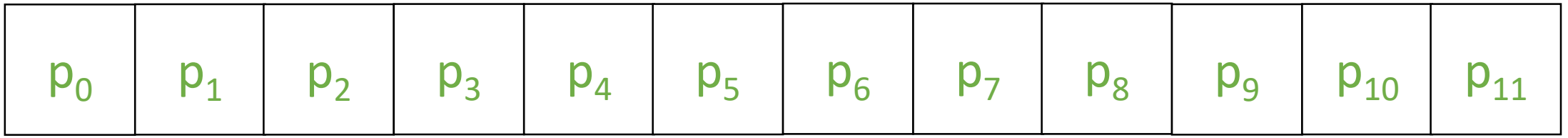

p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
p_{11}

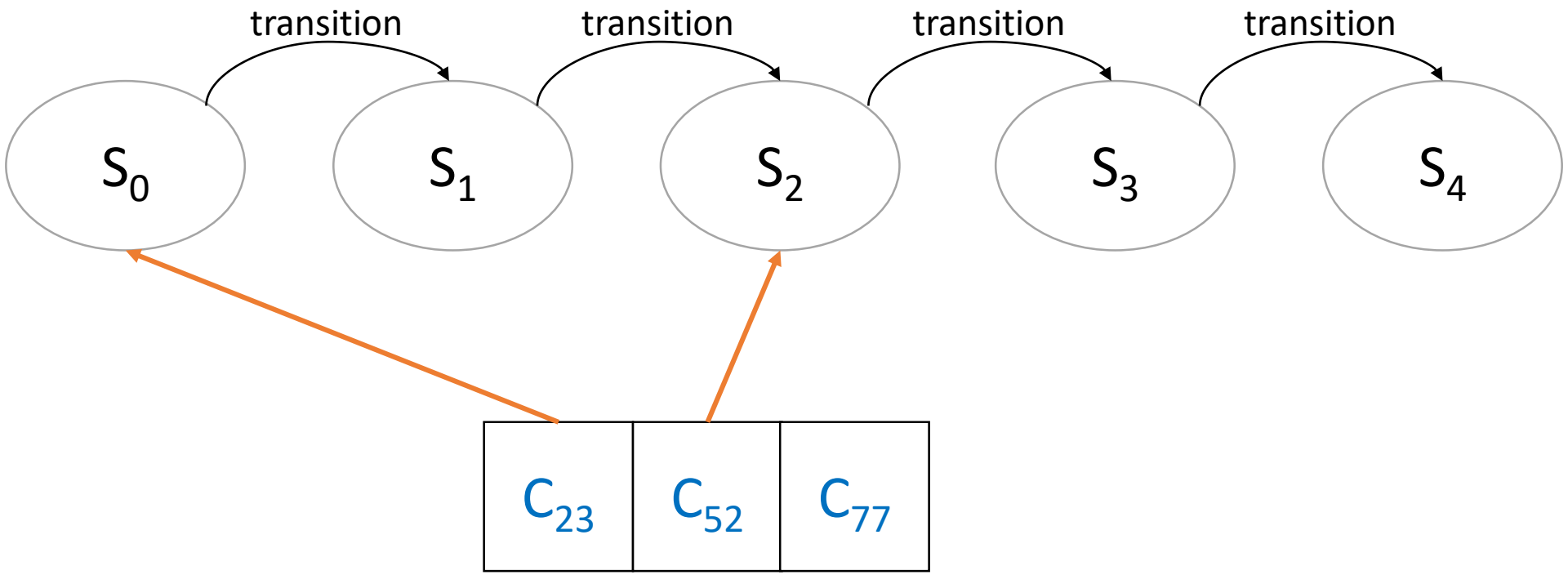

transition
transition
transition
transition
S_0
S_1
S_2
S_3
S_4

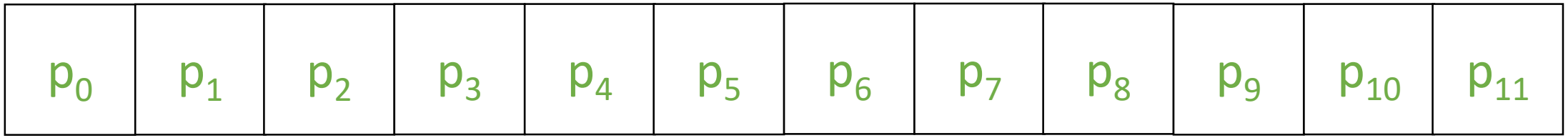

C_{23}
C_{52}
C_{77}

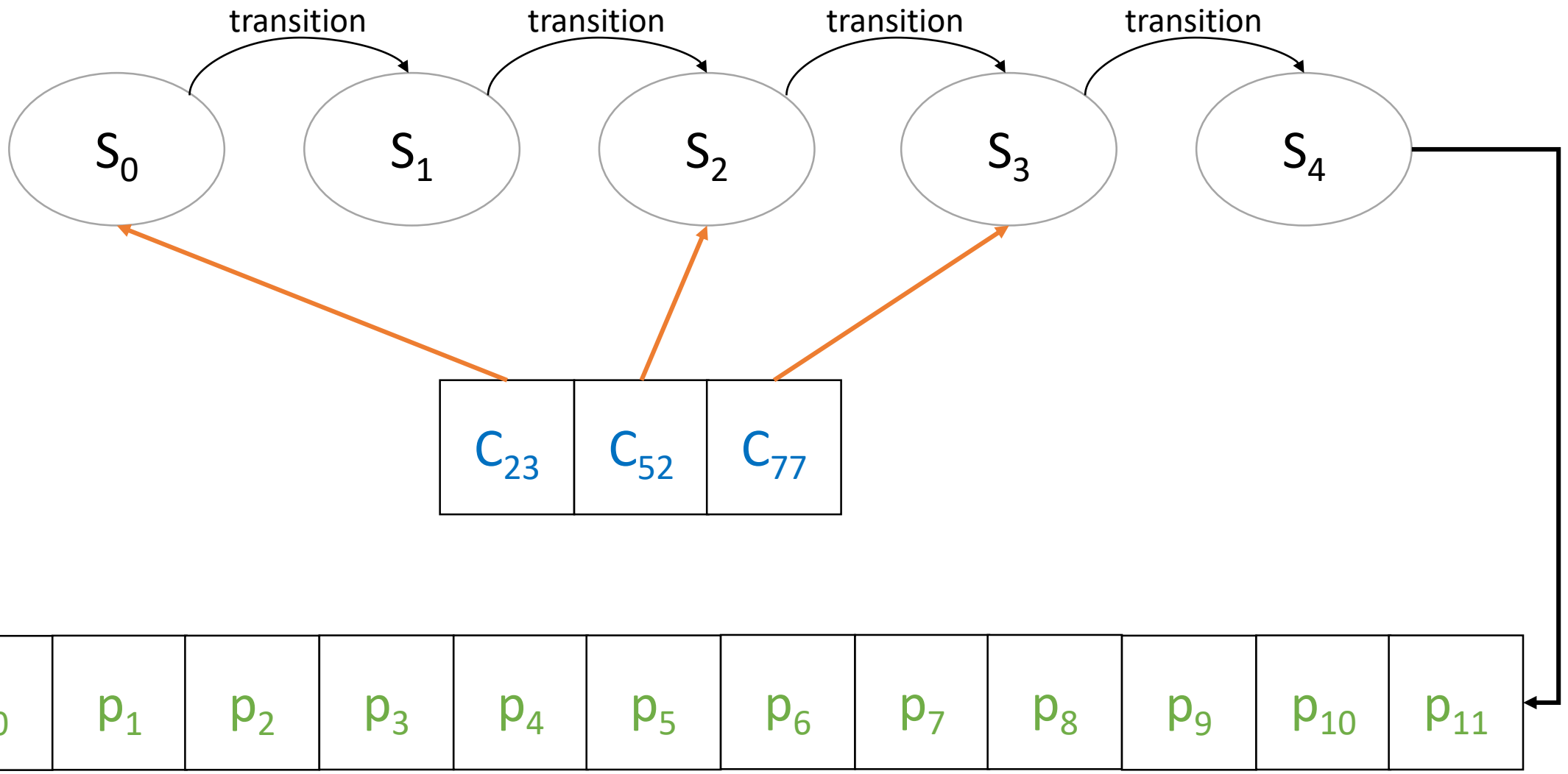

p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
p_{11}

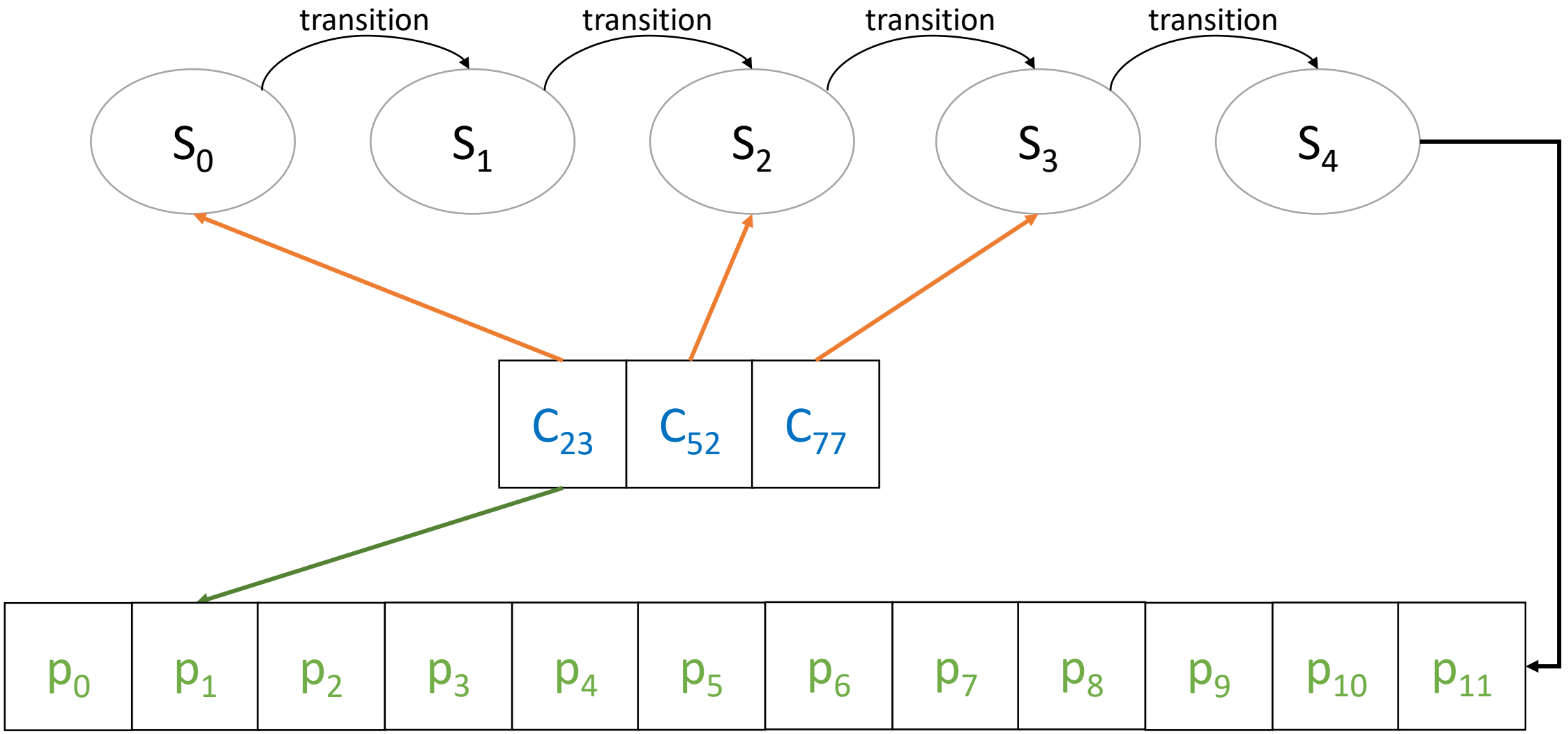

transition
transition
transition
transition
S_0
S_1
S_2
S_3
S_4
C_{23}
C_{52}
C_{77}

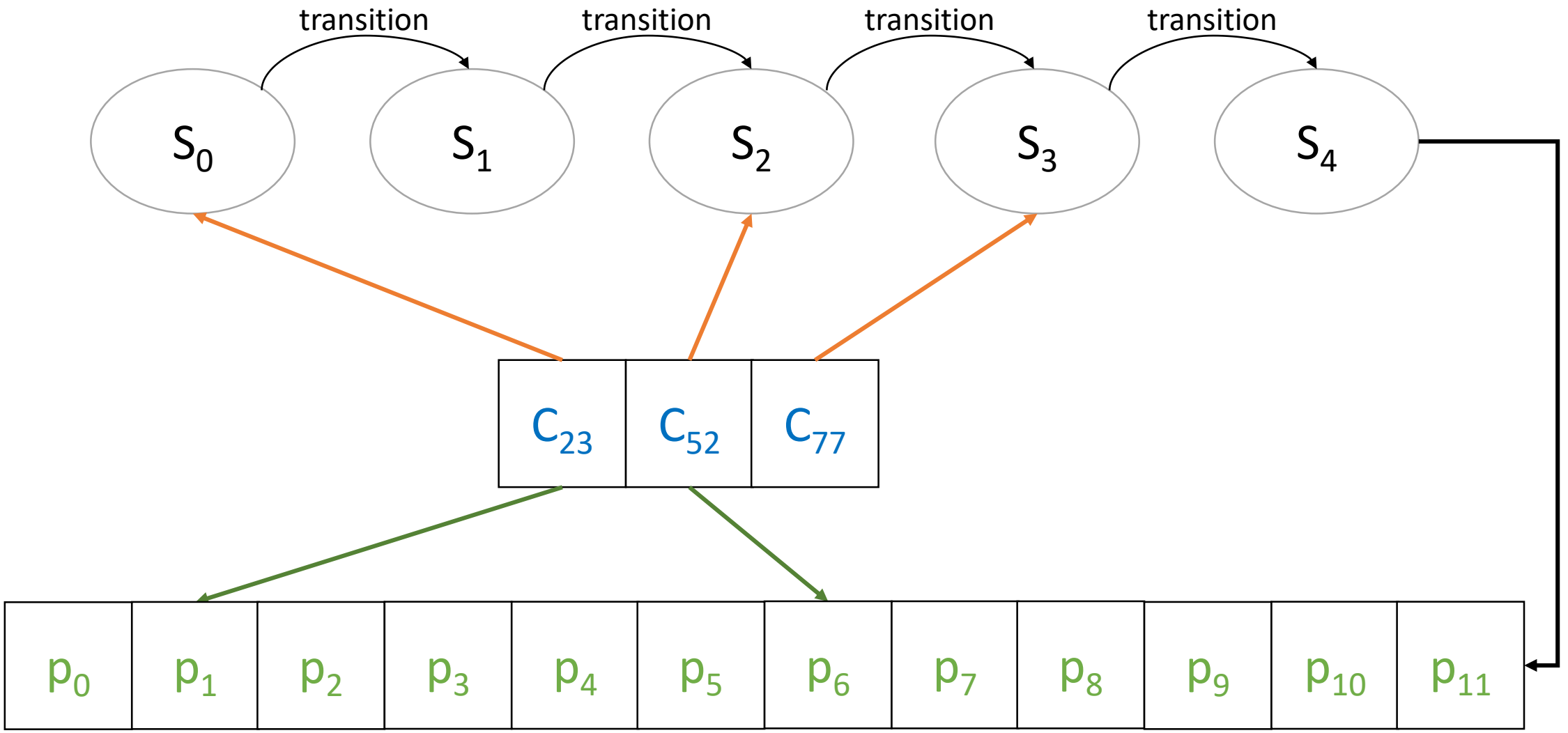

p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
p_{11}

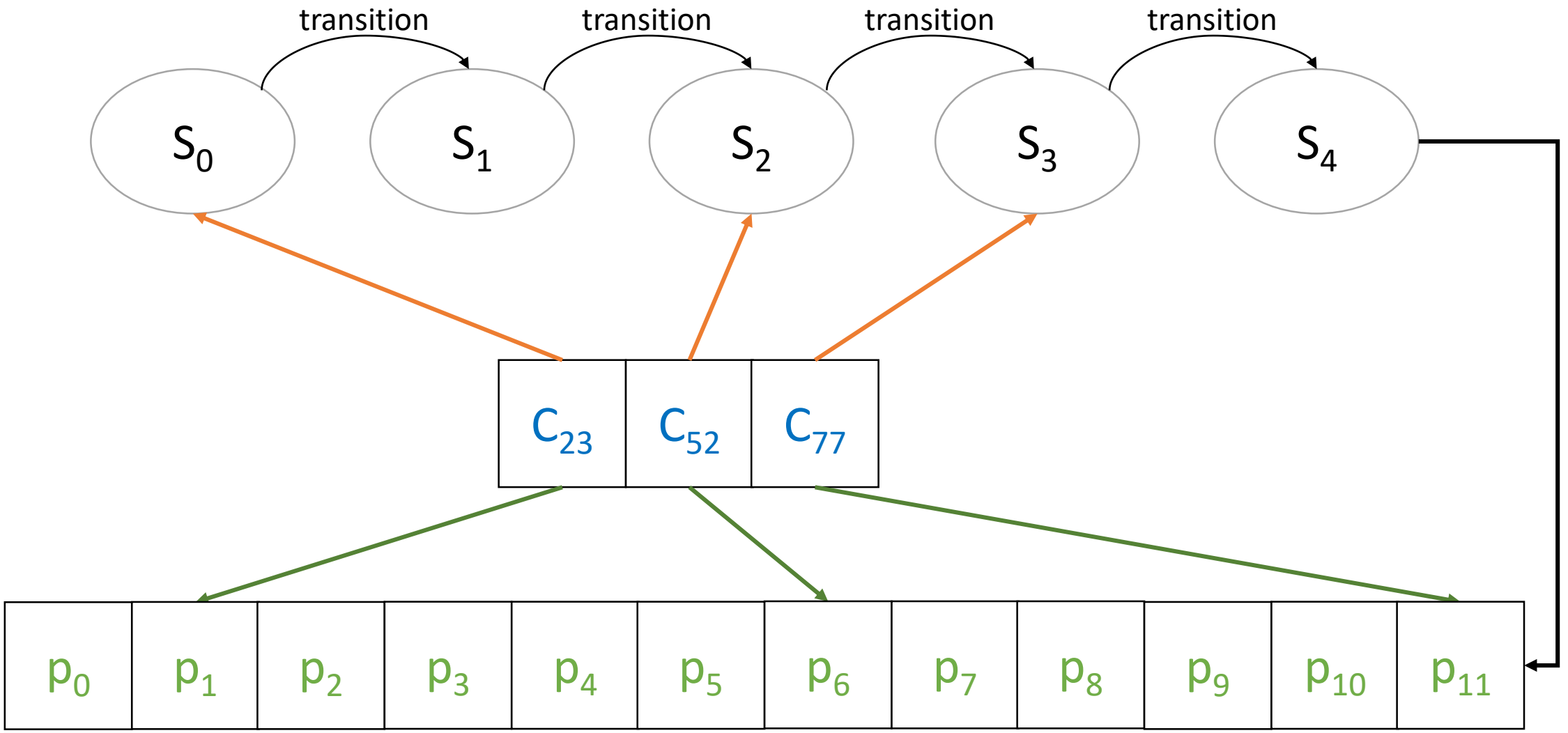

transition
transition
transition
transition
S_0
S_1
S_2
S_3
S_4
C_{23}
C_{52}
C_{77}


p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
p_{11}

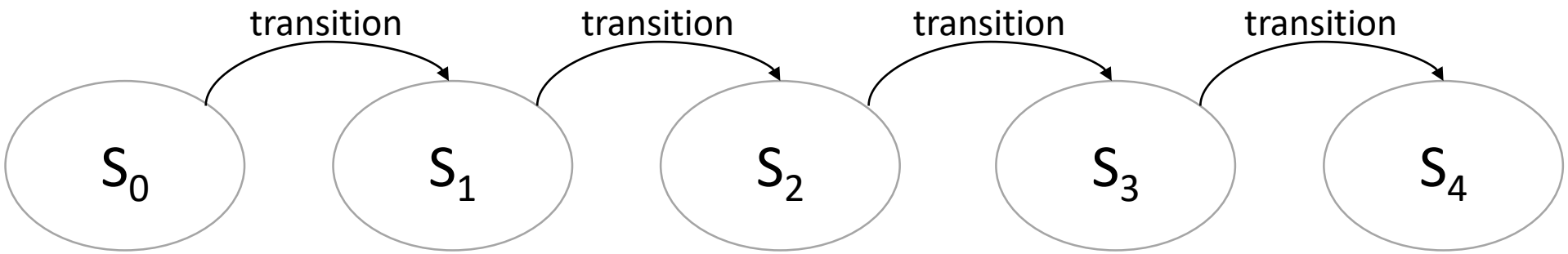

transition
transition
transition
transition
S_0
S_1
S_2
S_3
S_4
C_{23}
C_{52}
C_{77}
p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
p_{11}


transition
transition
transition
transition
S_0
S_1
S_2
S_3
S_4
C_{23}
C_{52}
C_{77}
p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
p_{11}

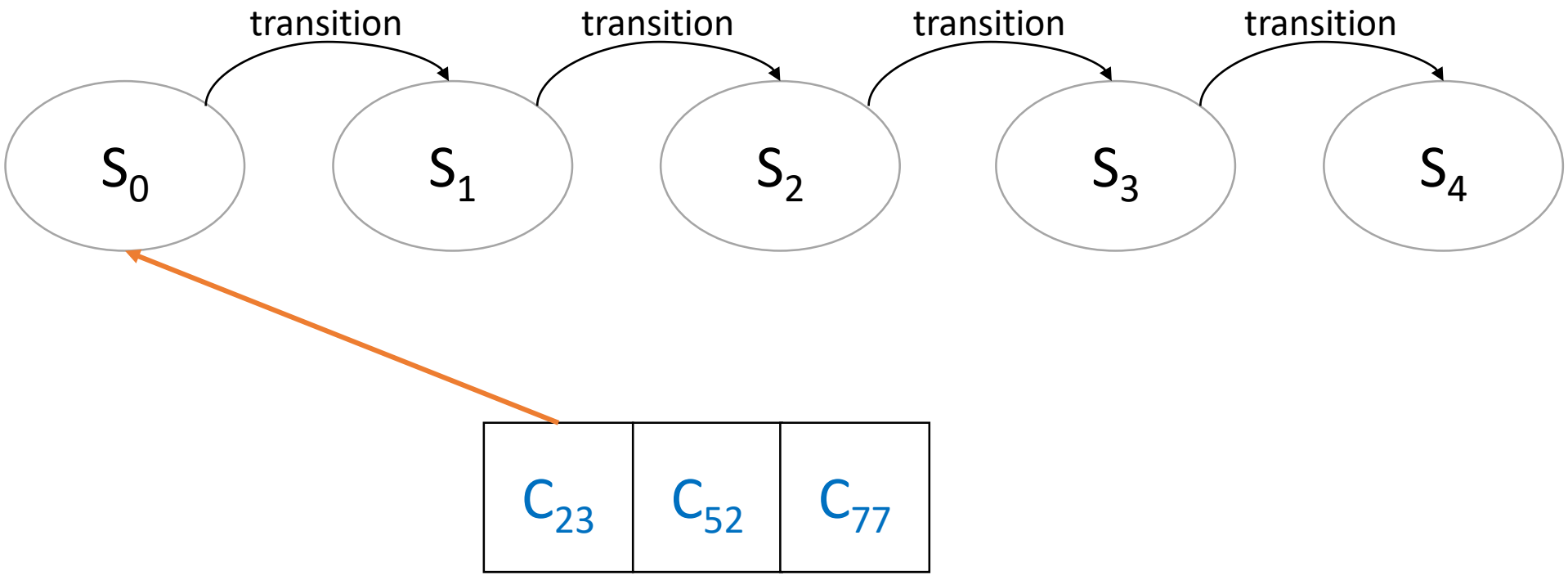

transition
transition
transition
transition
S_0
S_1
S_2
S_3
S_4
C_{23}
C_{52}
C_{77}
p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
p_{11}


transition
transition
transition
transition
S_0
S_1
S_2
S_3
S_4
C_{23}
C_{52}
C_{77}
p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
p_{11}

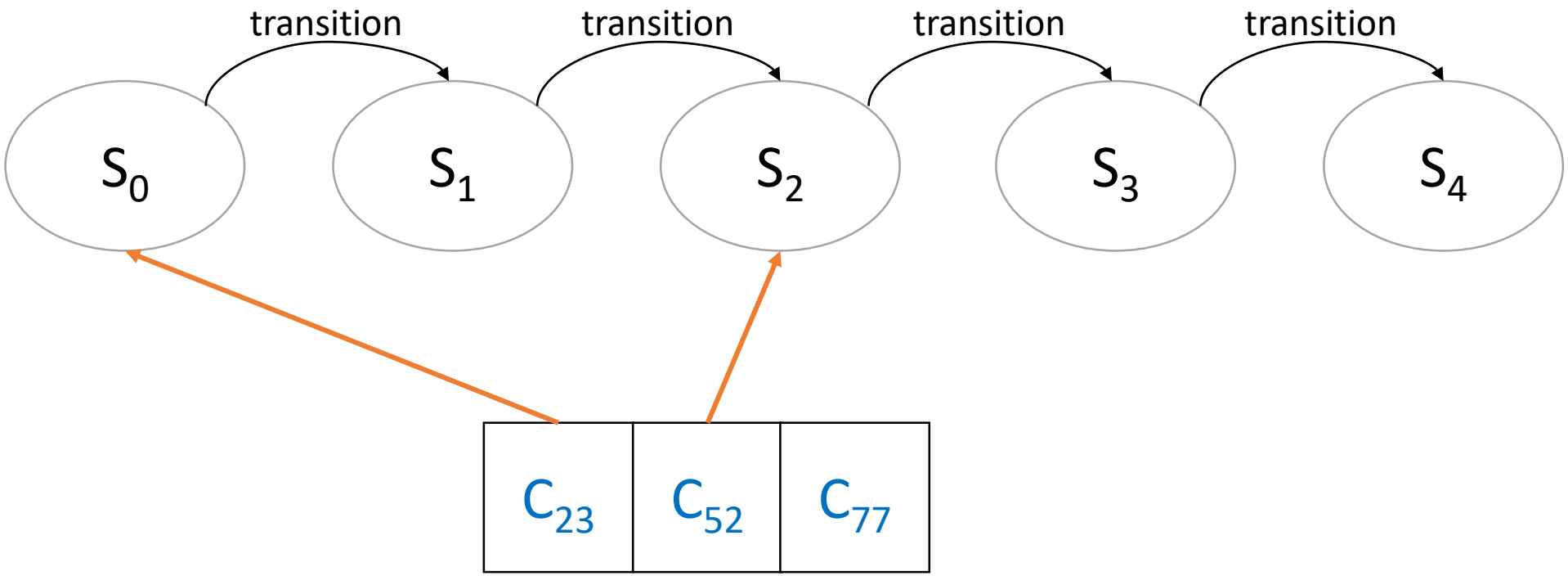
Lemma 1: seq. commit is continuous



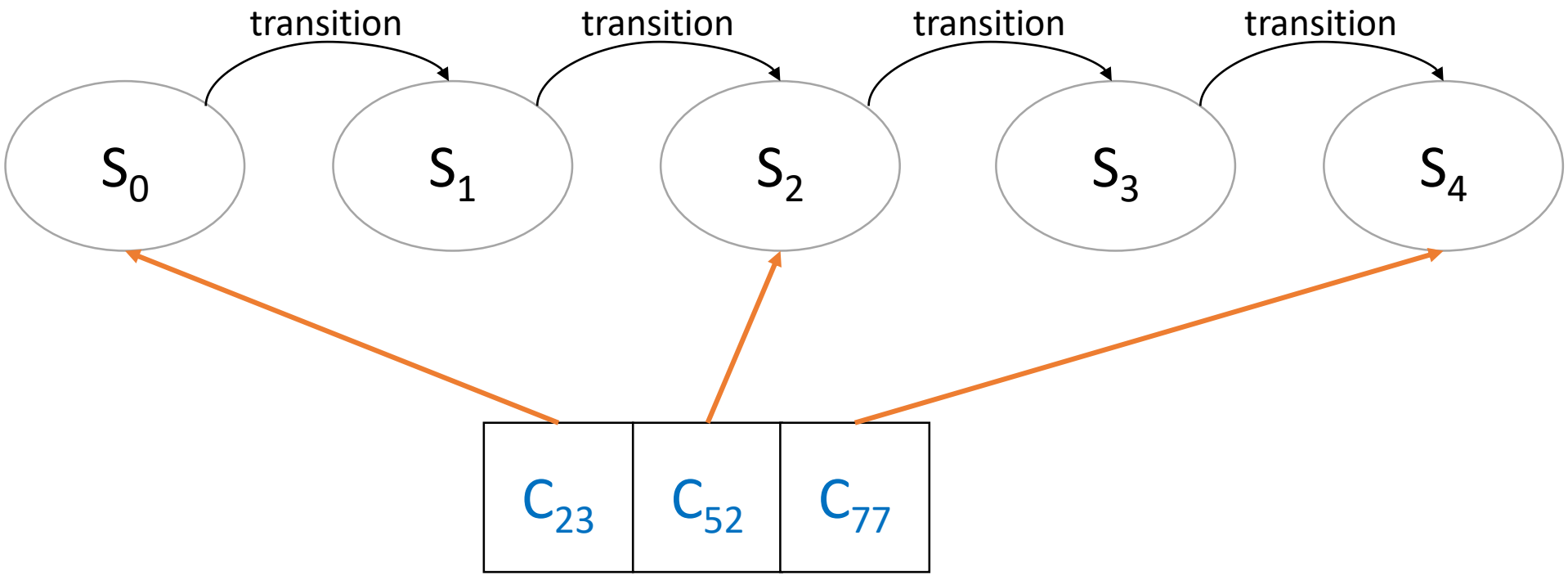
Lemma 1: seq. commit is continuous



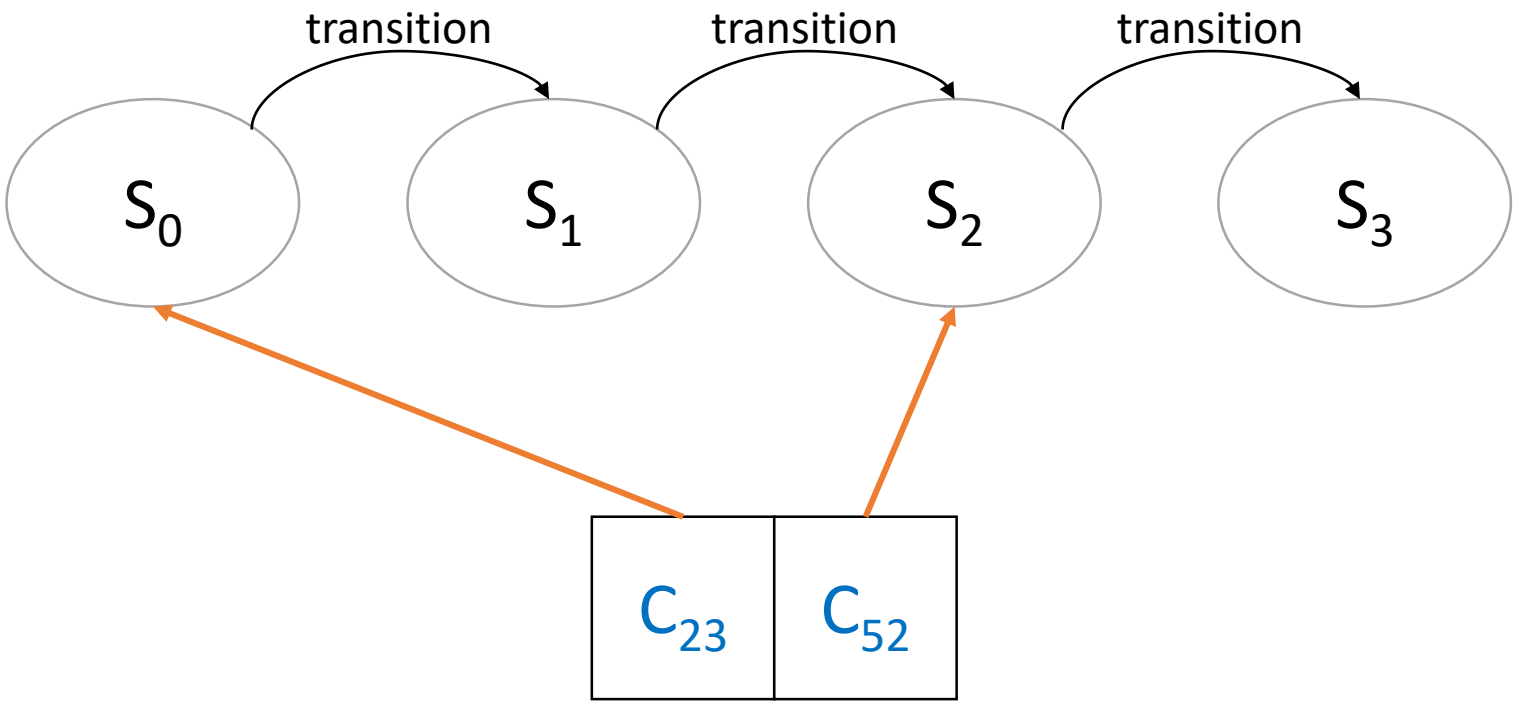
Lemma 1: seq. commit is continuous



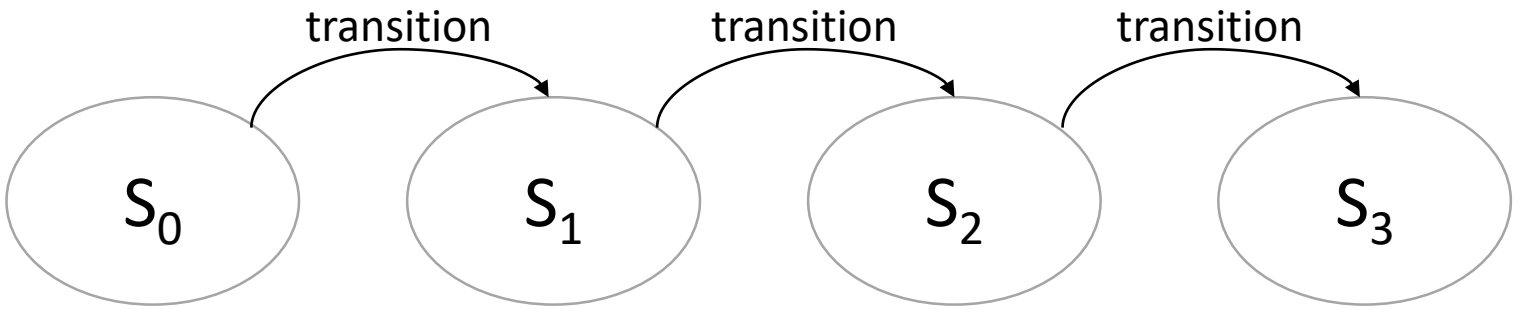
Lemma 1: seq. commit is continuous

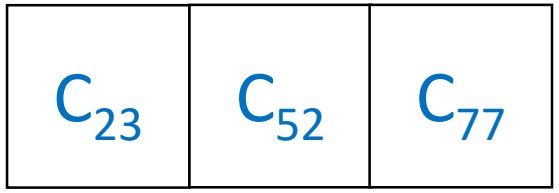


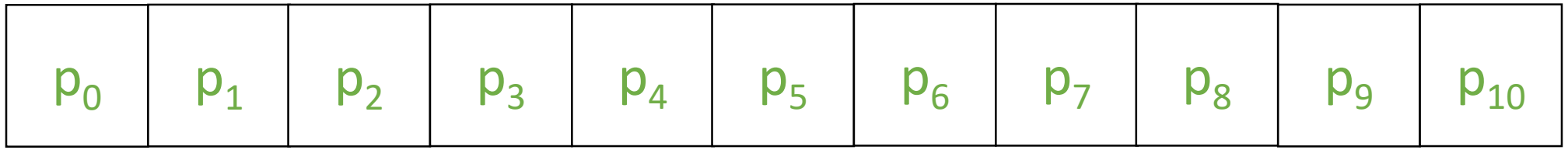
Lemma 1: seq. commit is continuous



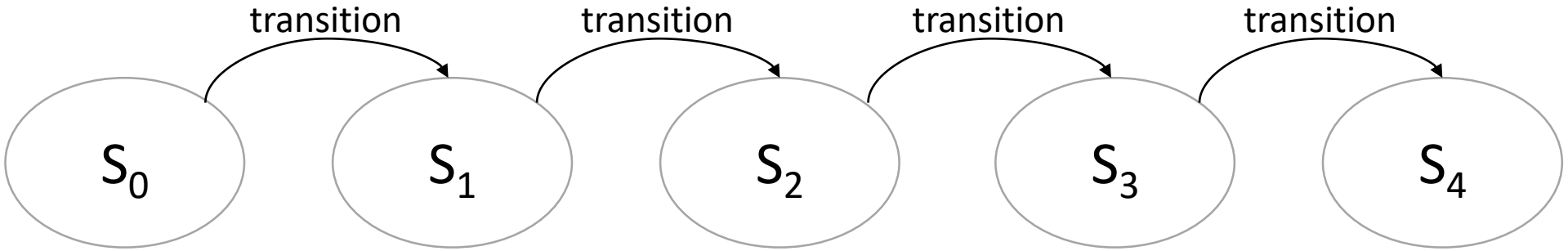
Lemma 2: new commits extend the log

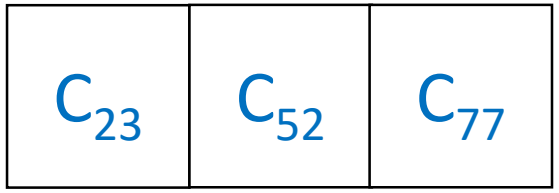


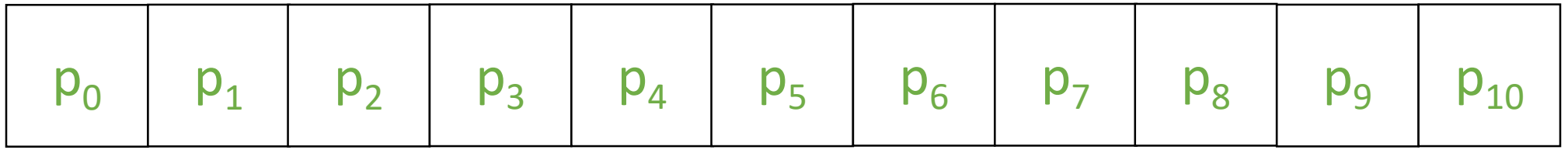




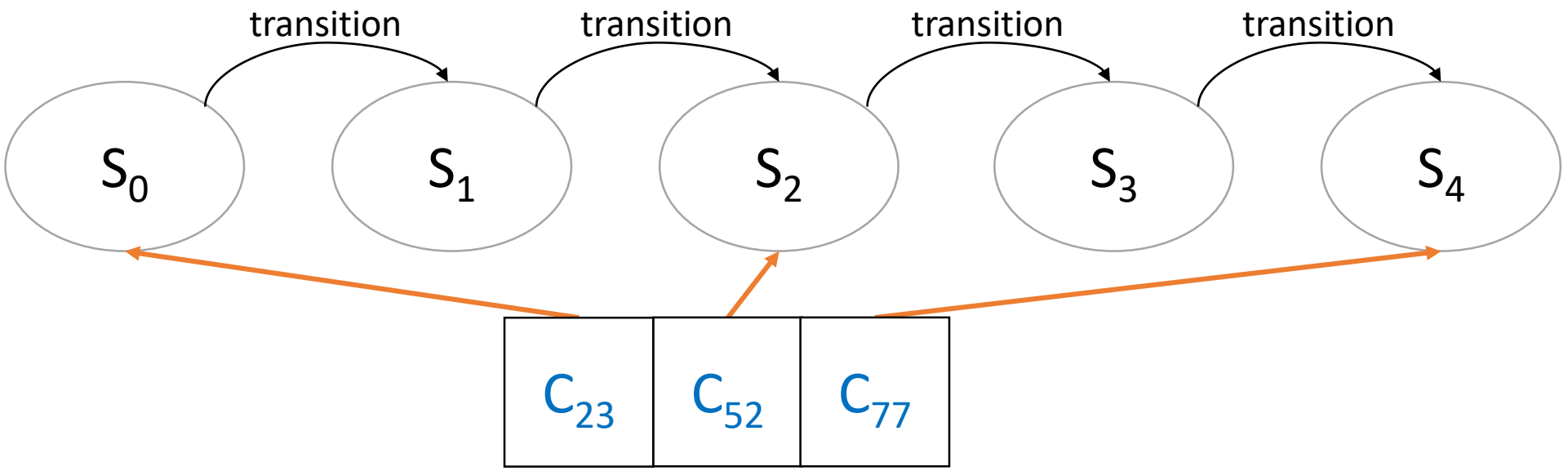
Lemma 2: new commits extend the log

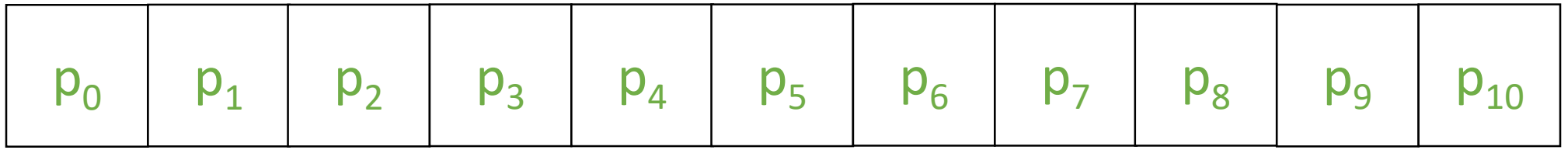




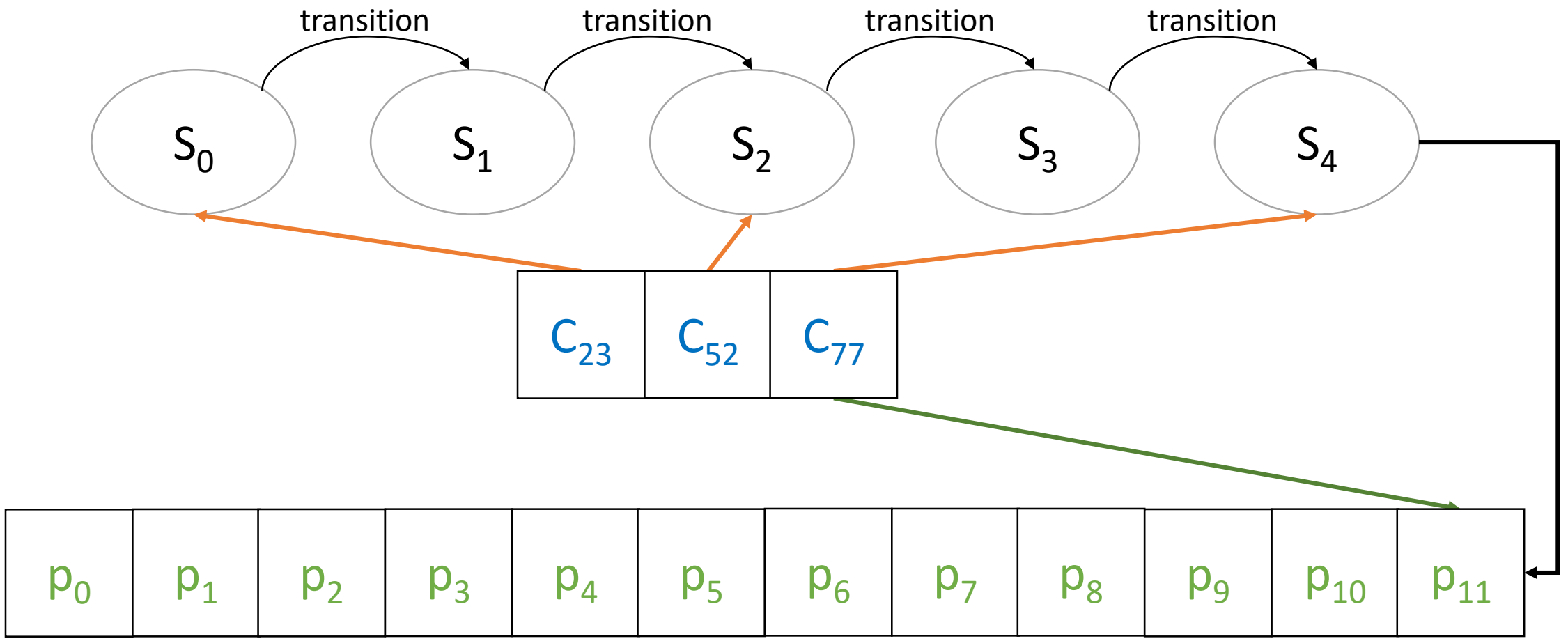


Lemma 2: new commits extend the log

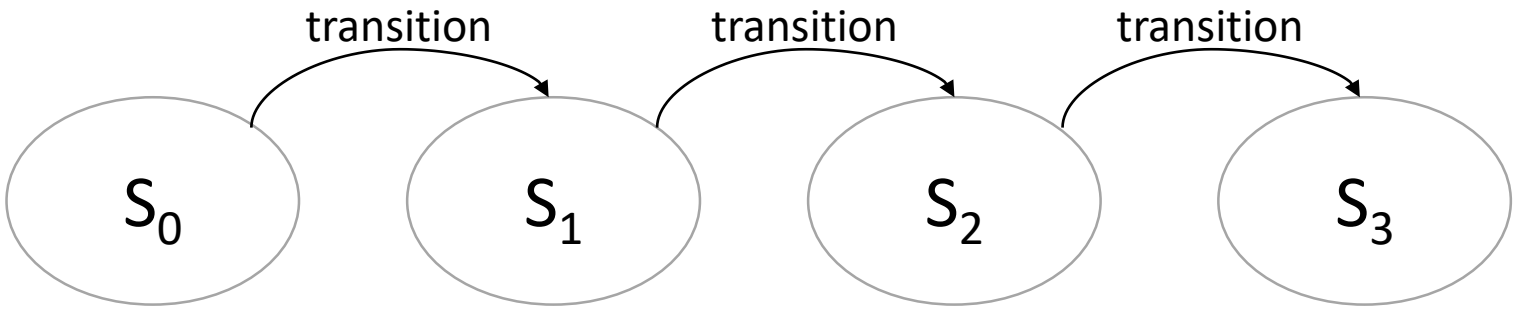


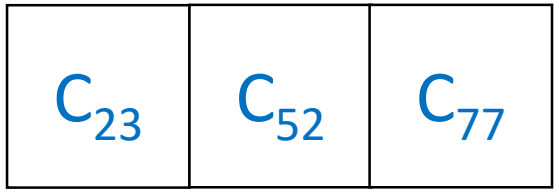


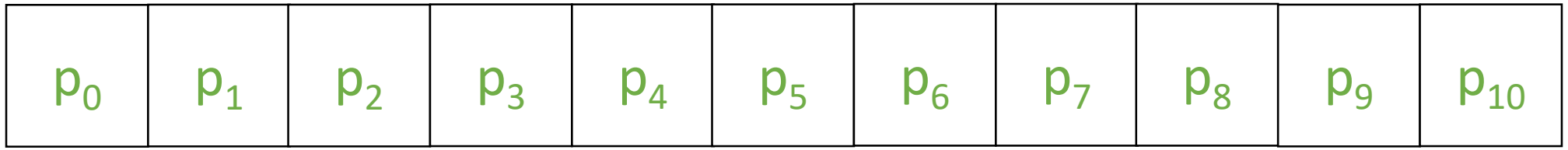
Lemma 2: new commits extend the log



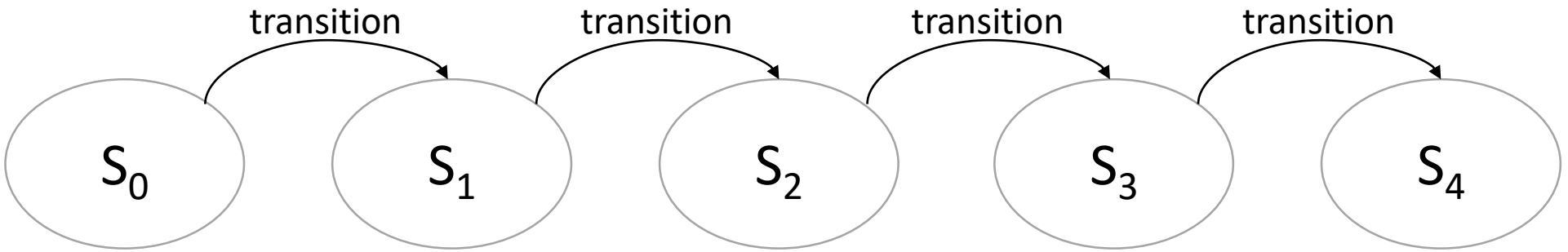
Proof: new commits extend the log

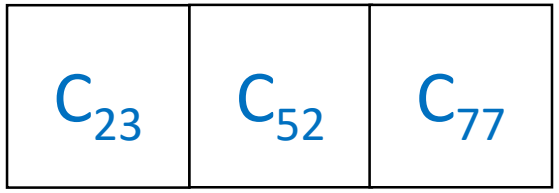


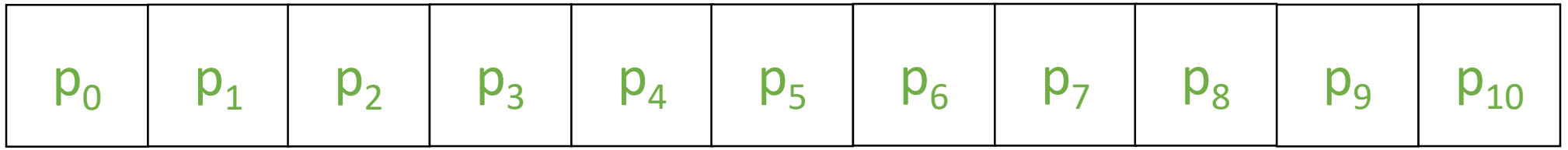




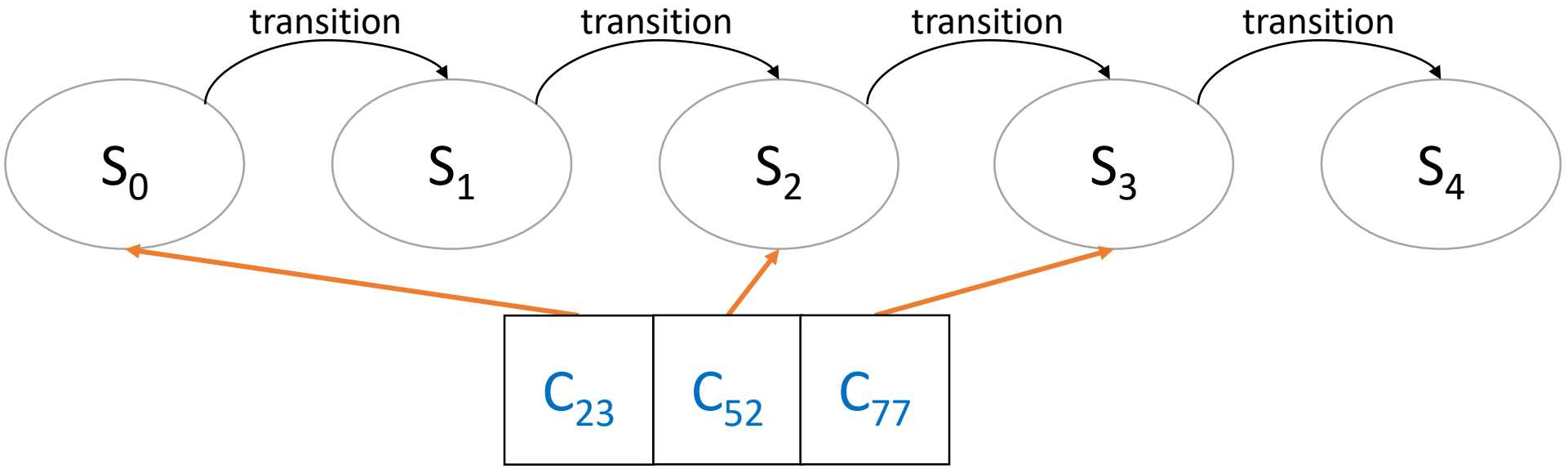
Proof: new commits extend the log

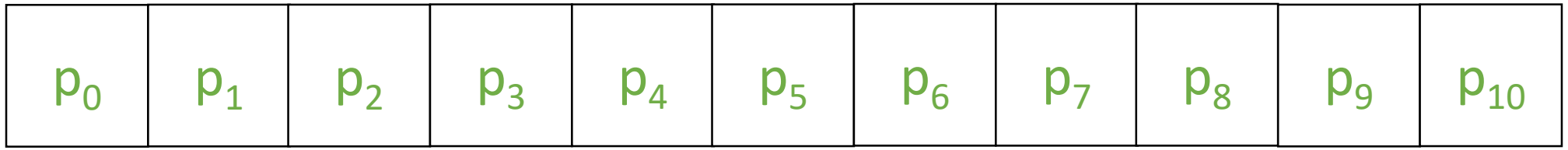




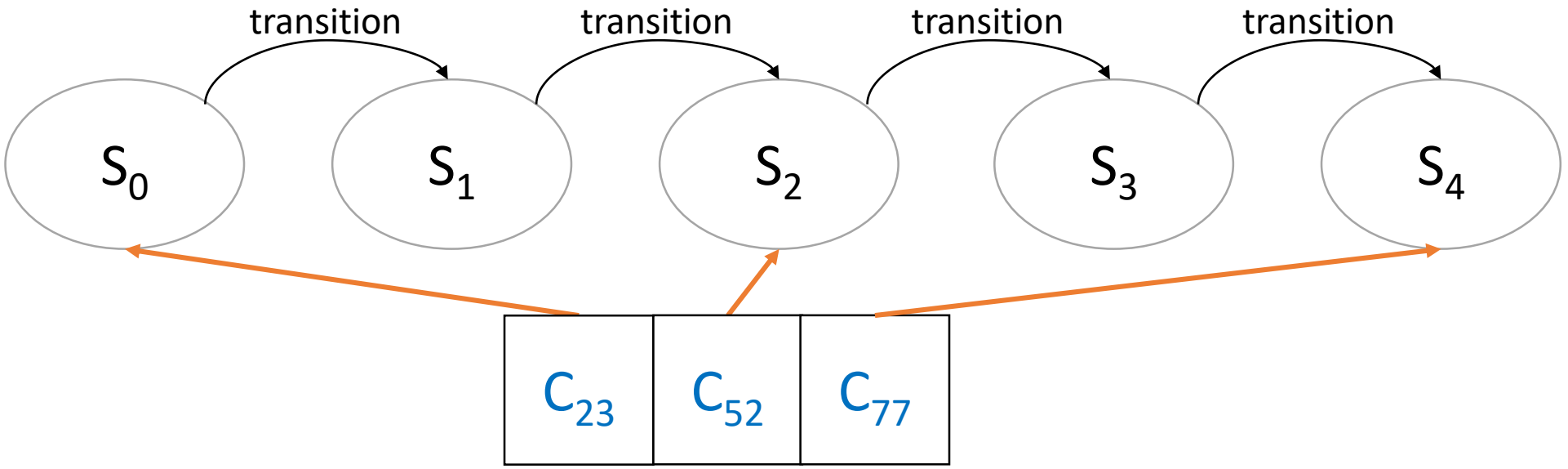


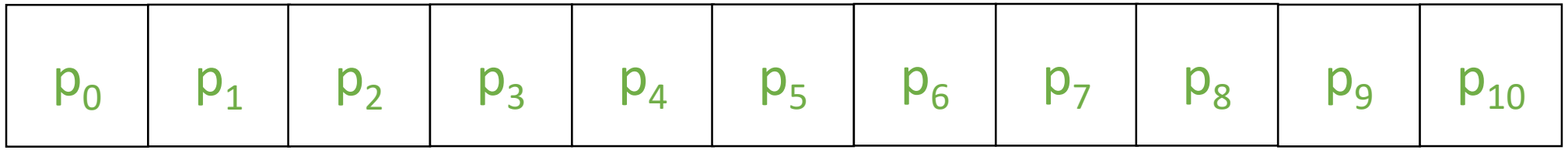
Proof: new commits extend the log



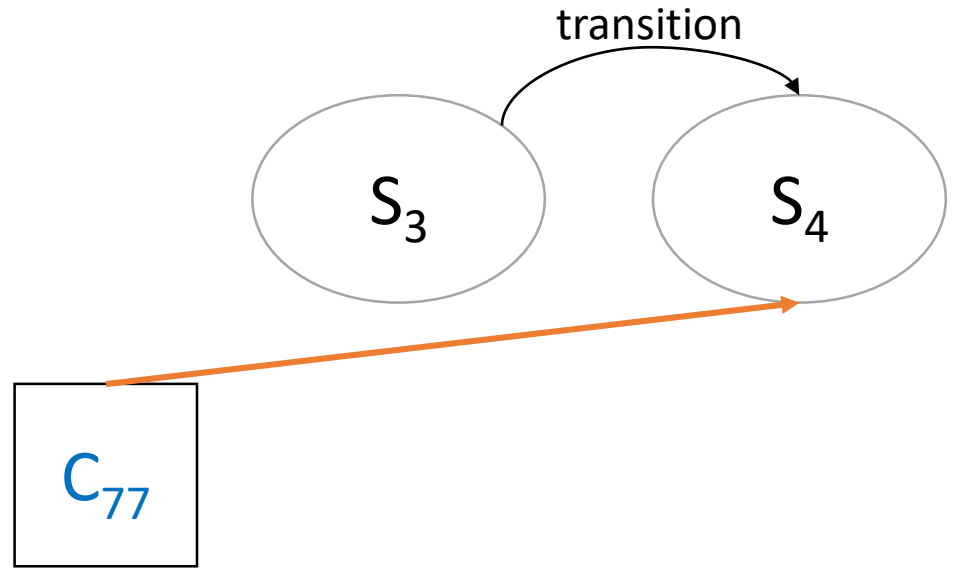


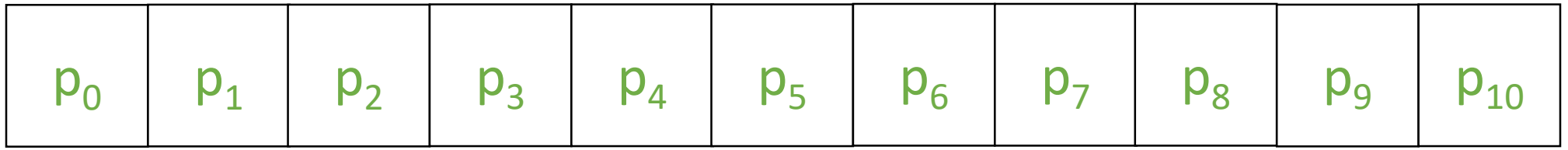
Proof: new commits extend the log



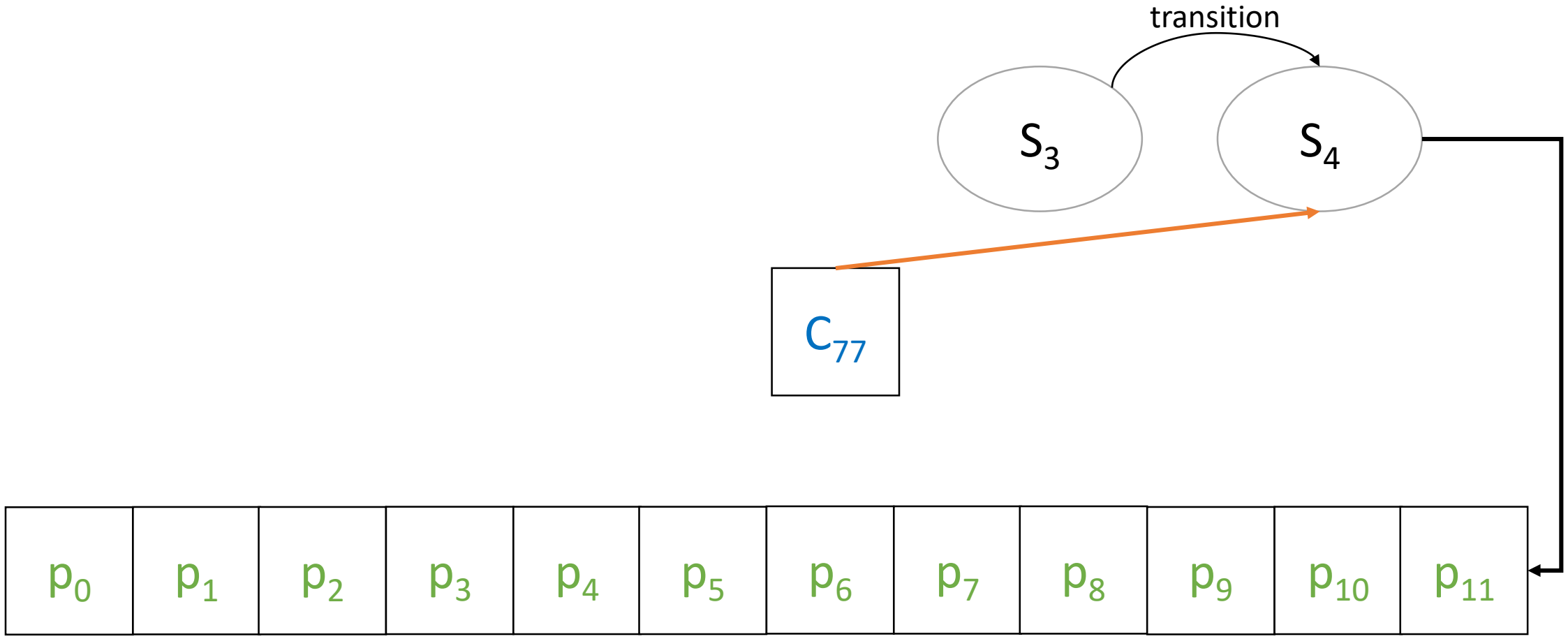


Proof: new commits extend the log

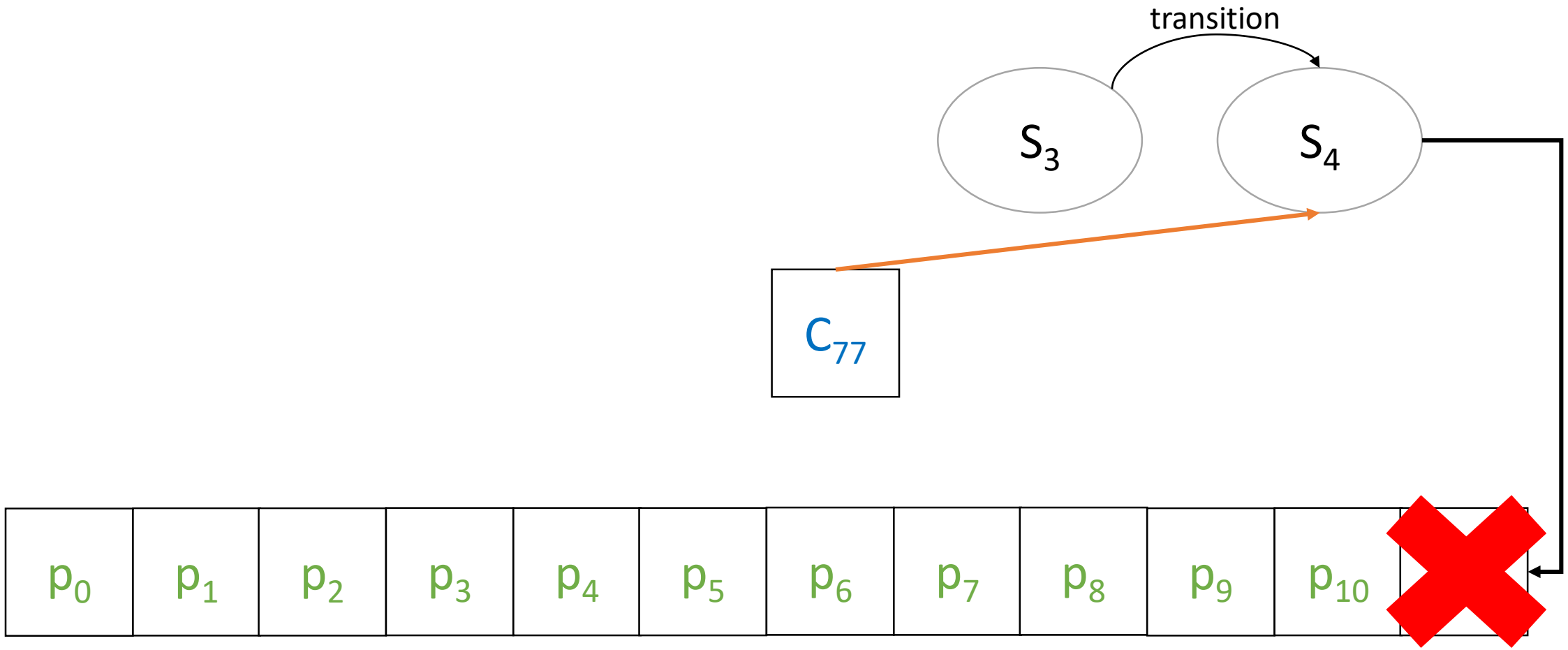




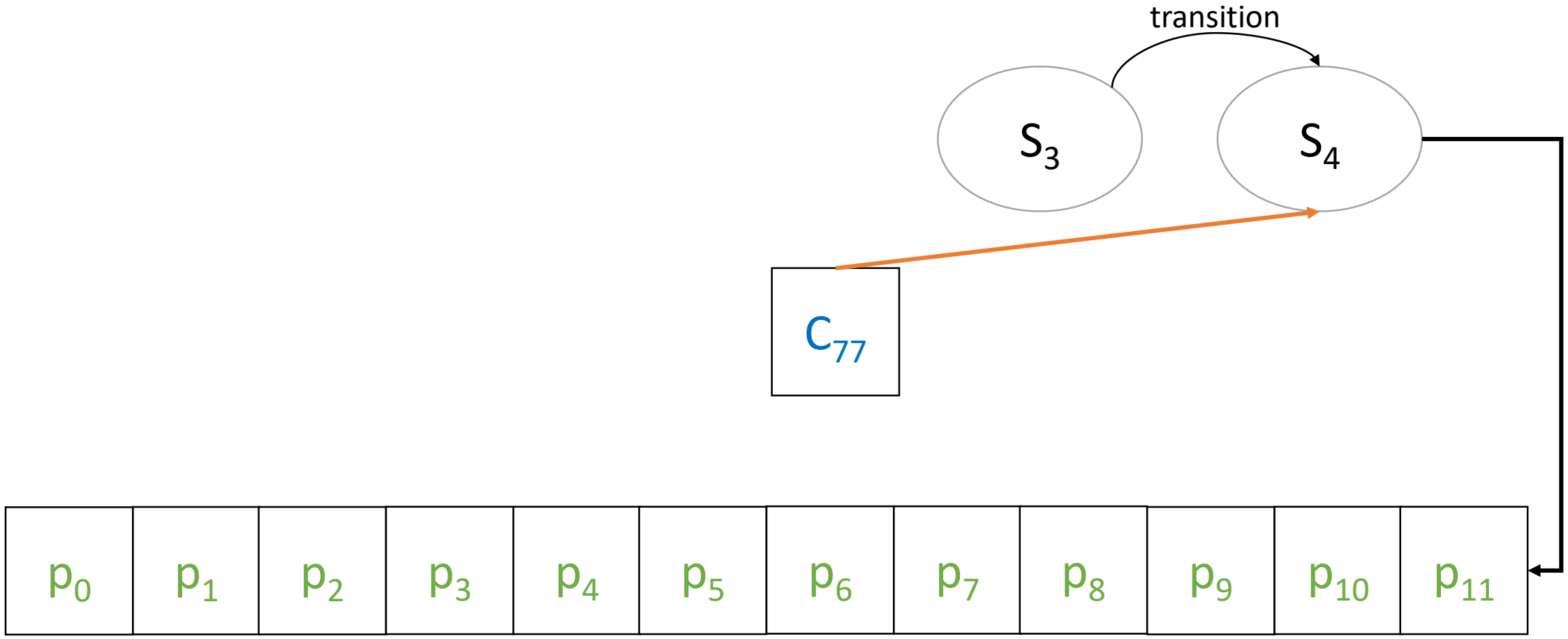
Proof: new commits extend the log



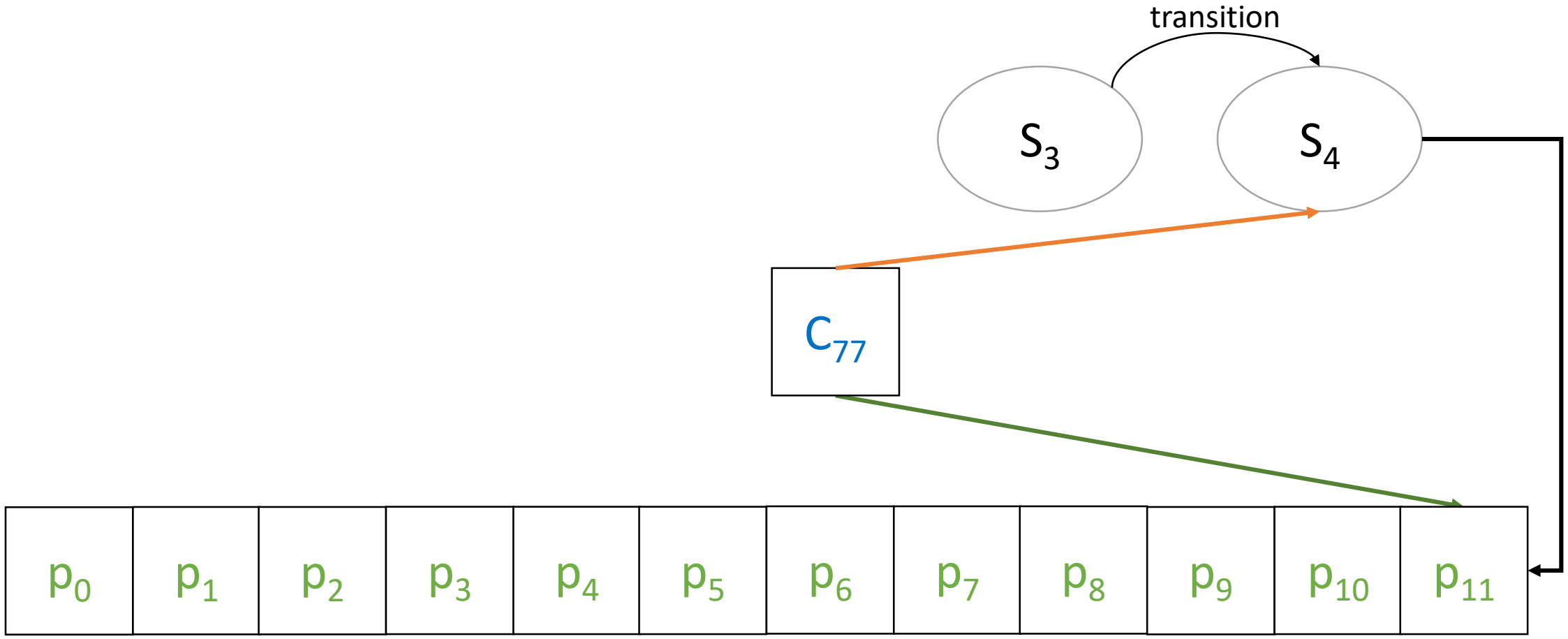
Proof: new commits extend the log



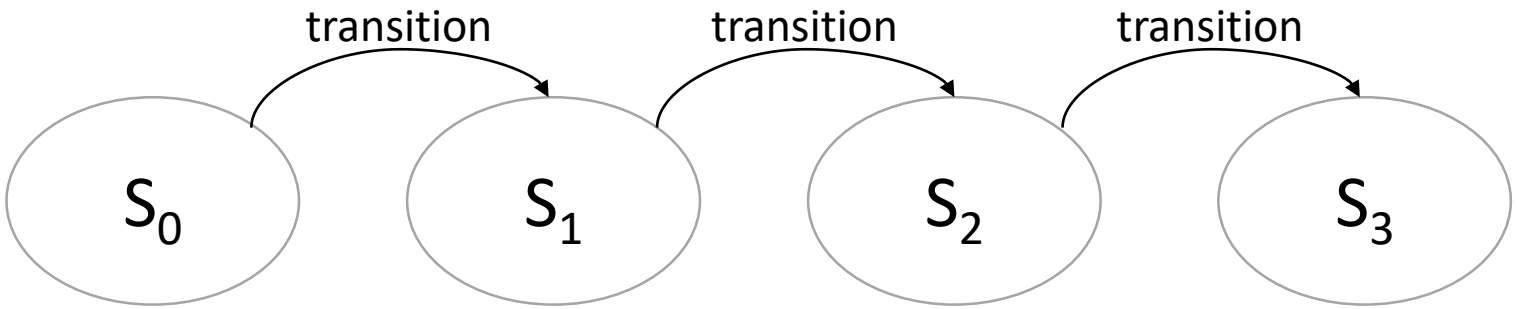
Proof: new commits extend the log

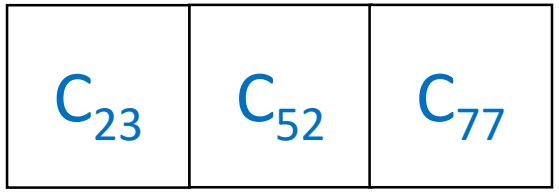


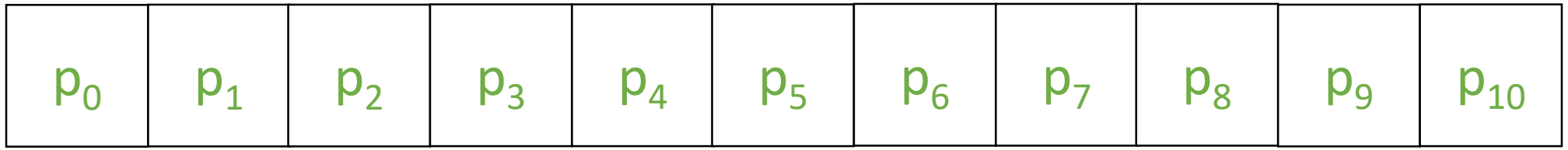
Proof: new commits extend the log



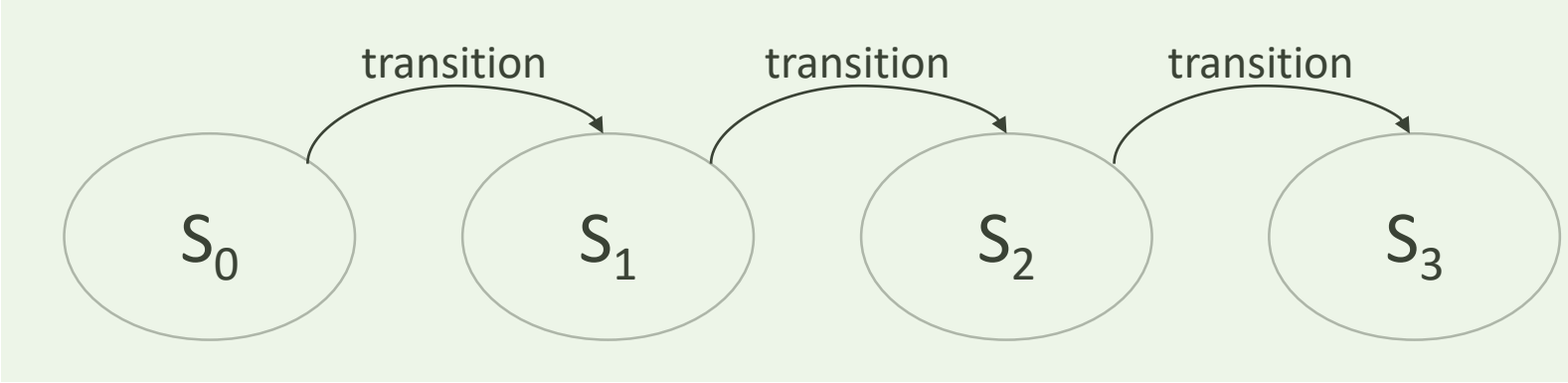
Proof: sequential consistency

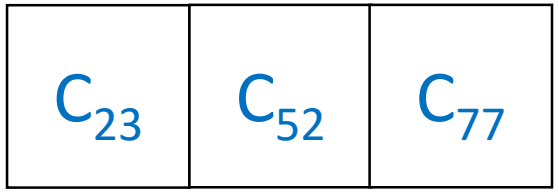


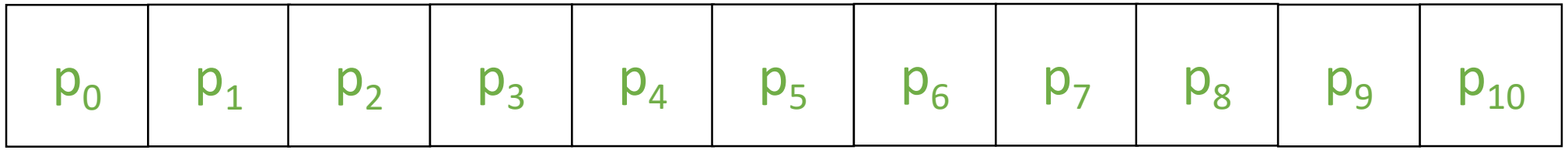




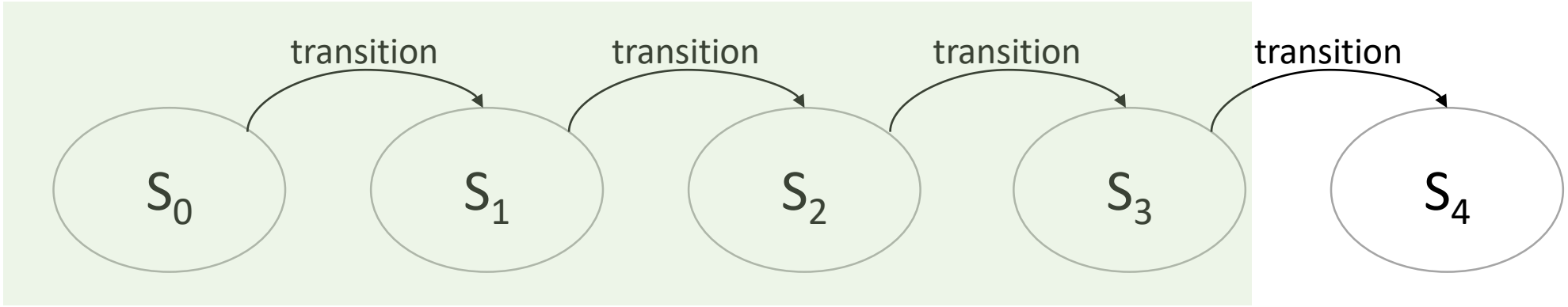
Proof: sequential consistency

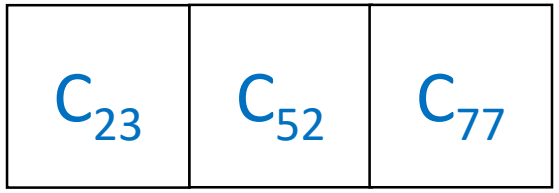


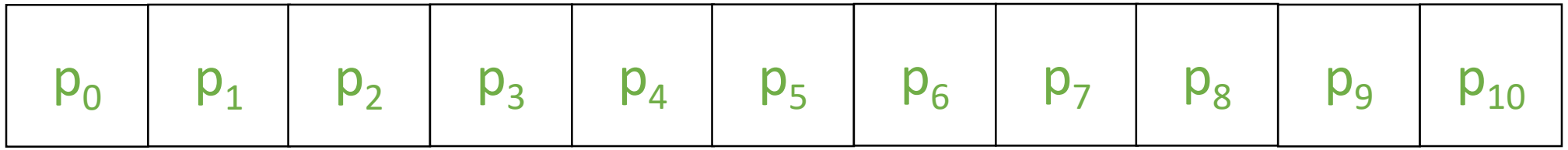




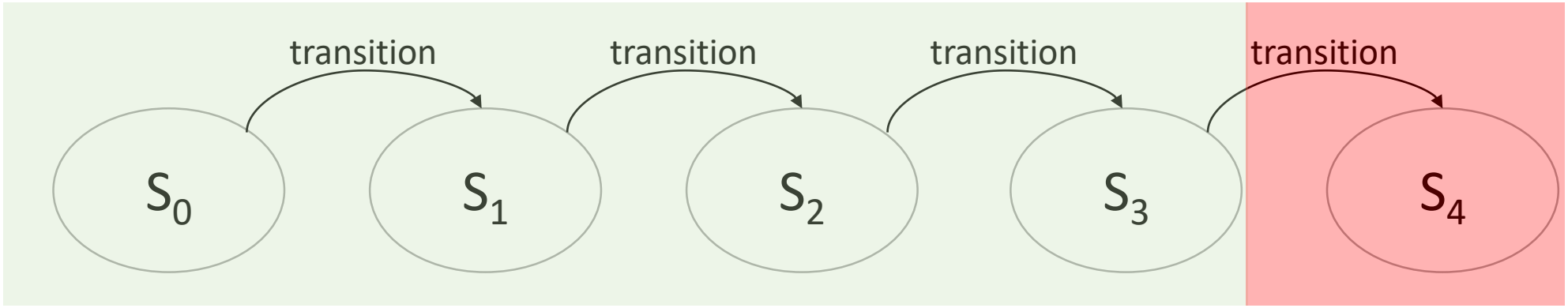
Proof: sequential consistency

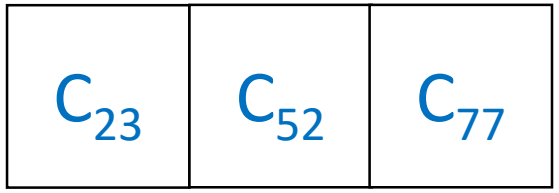


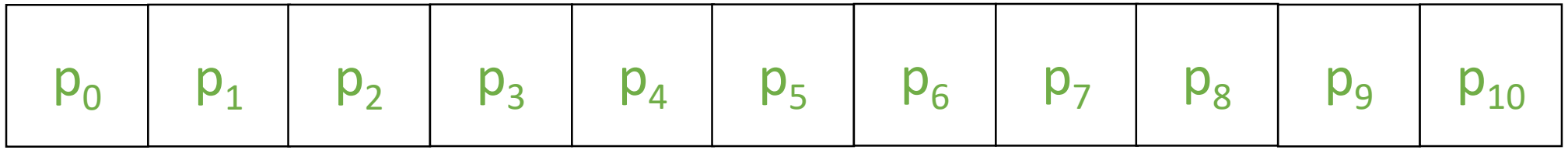




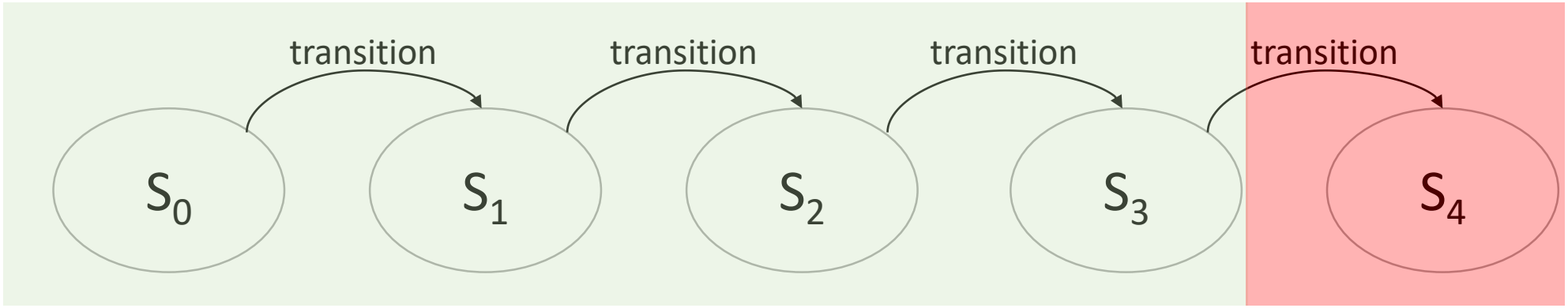
Proof: sequential consistency

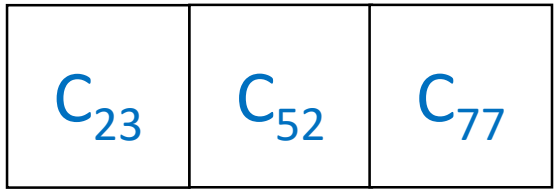


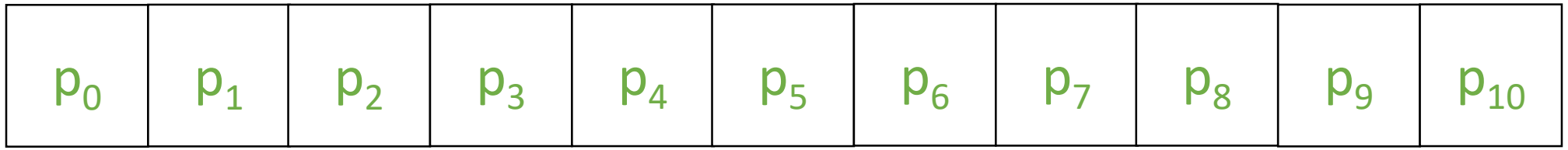




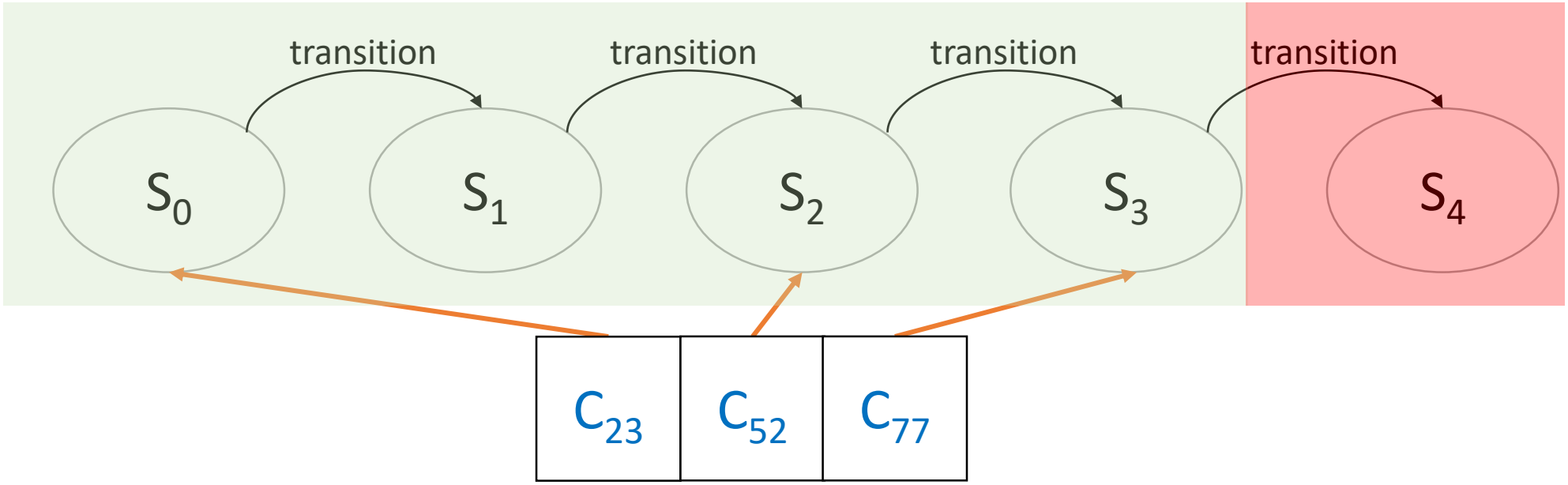
Proof: sequential consistency

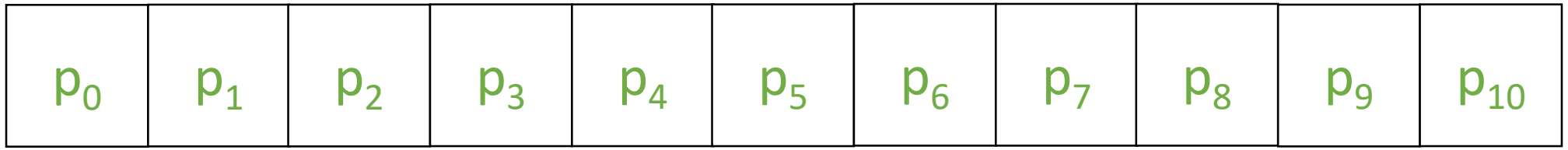




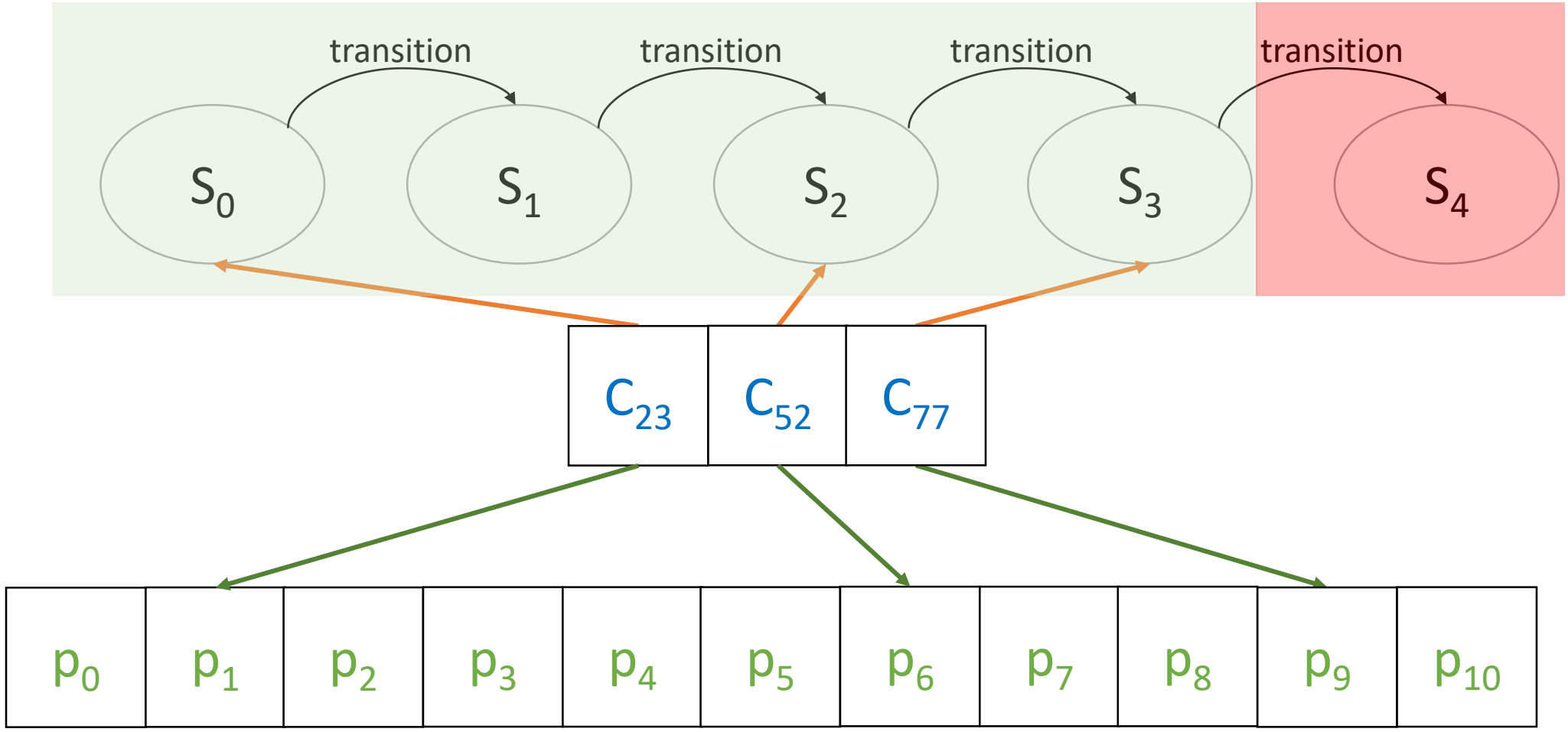


Proof: sequential consistency

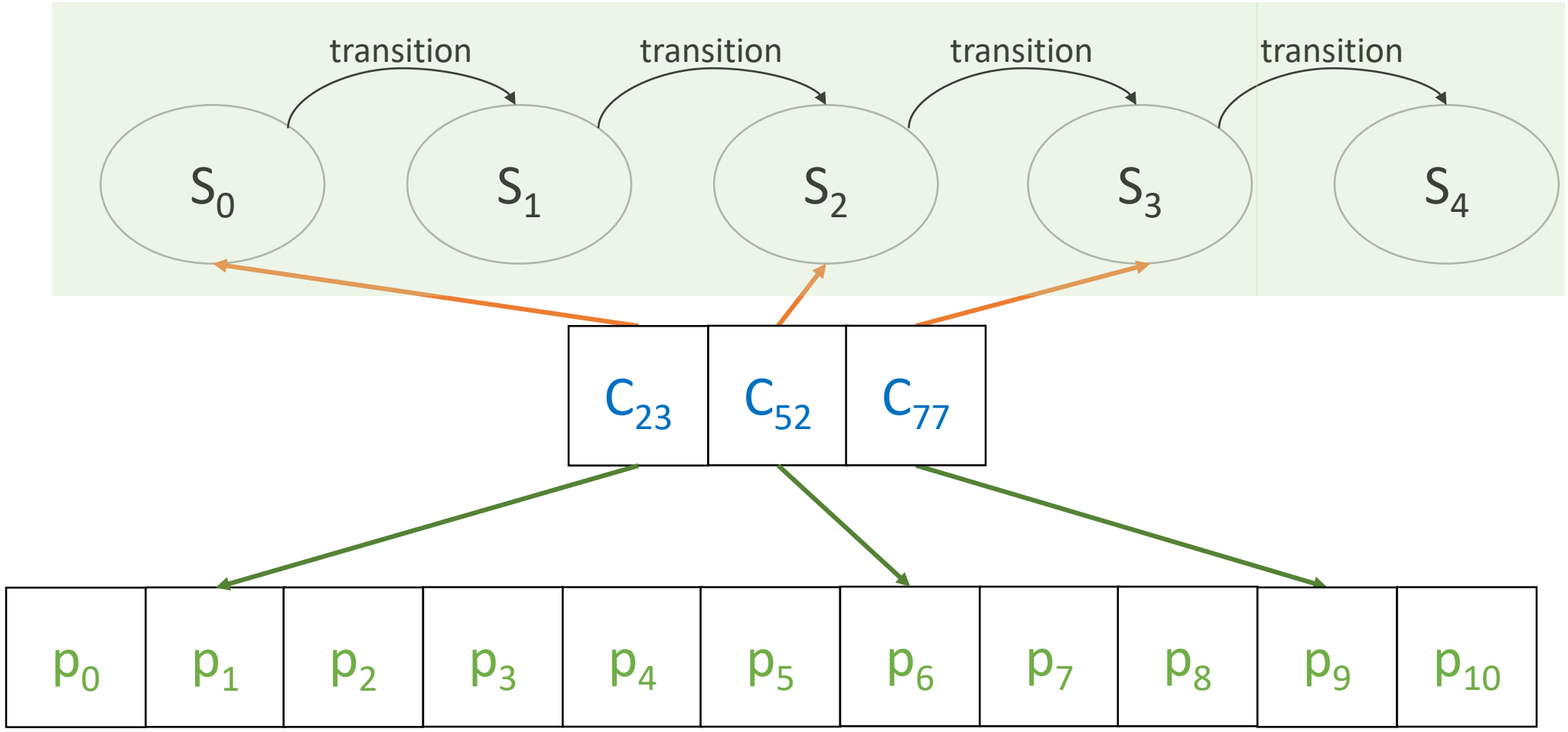




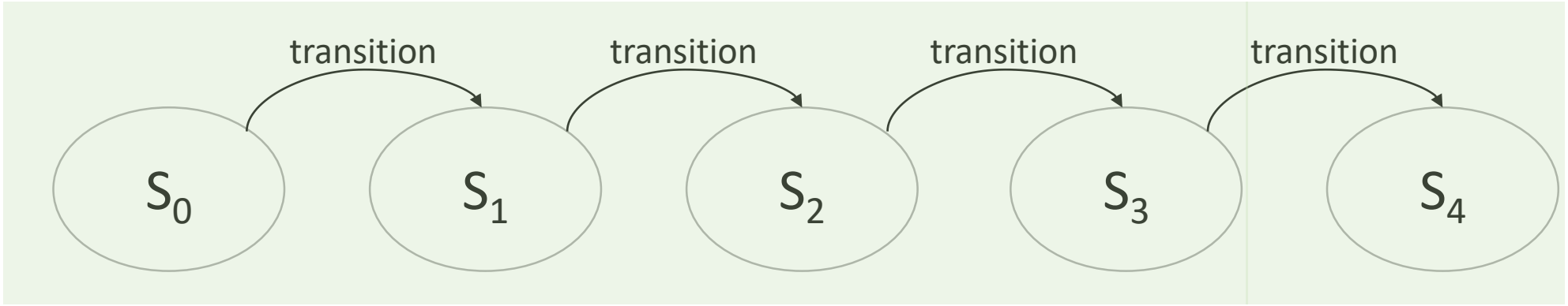
Proof: sequential consistency

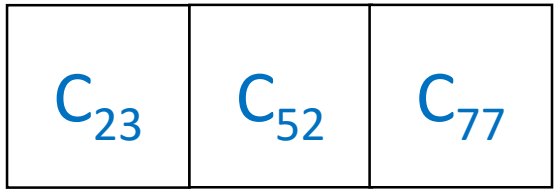


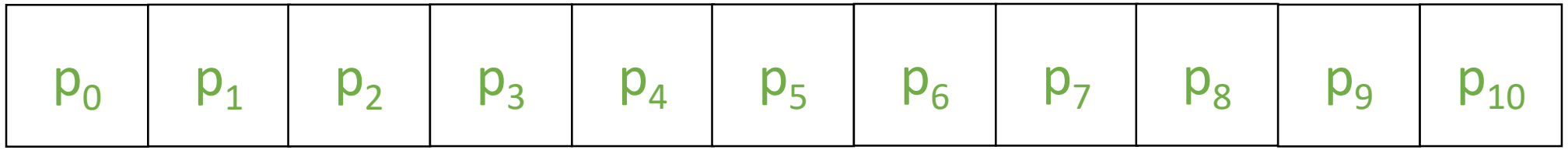
Proof: sequential consistency



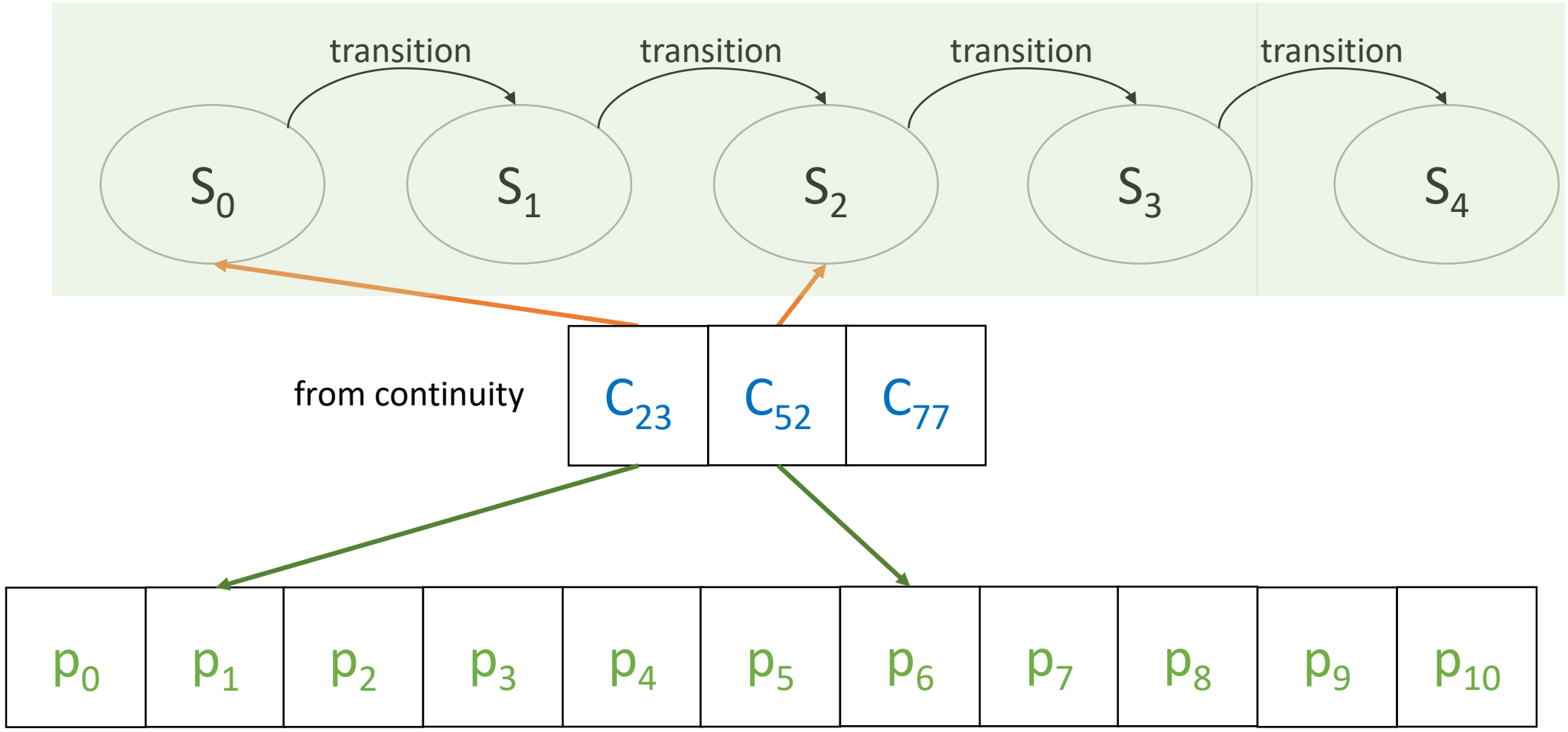
Proof: sequential consistency



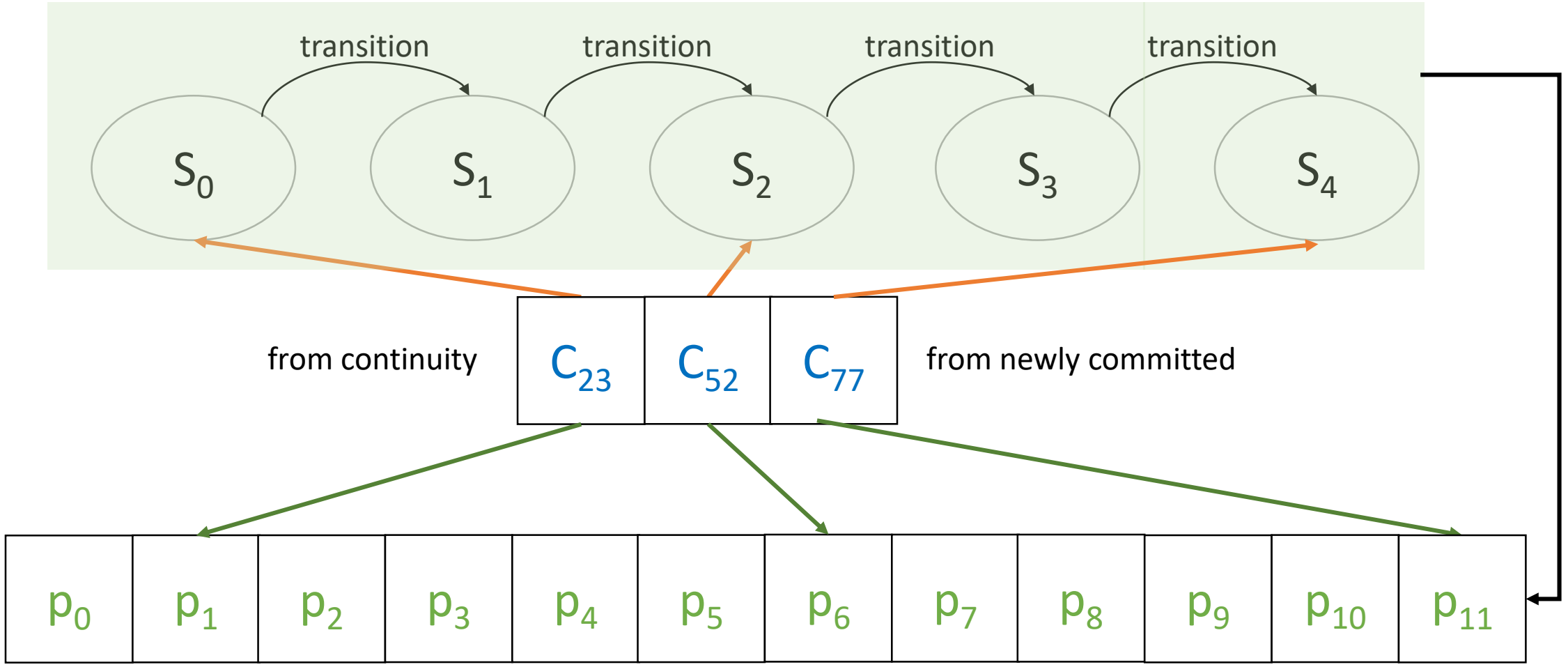




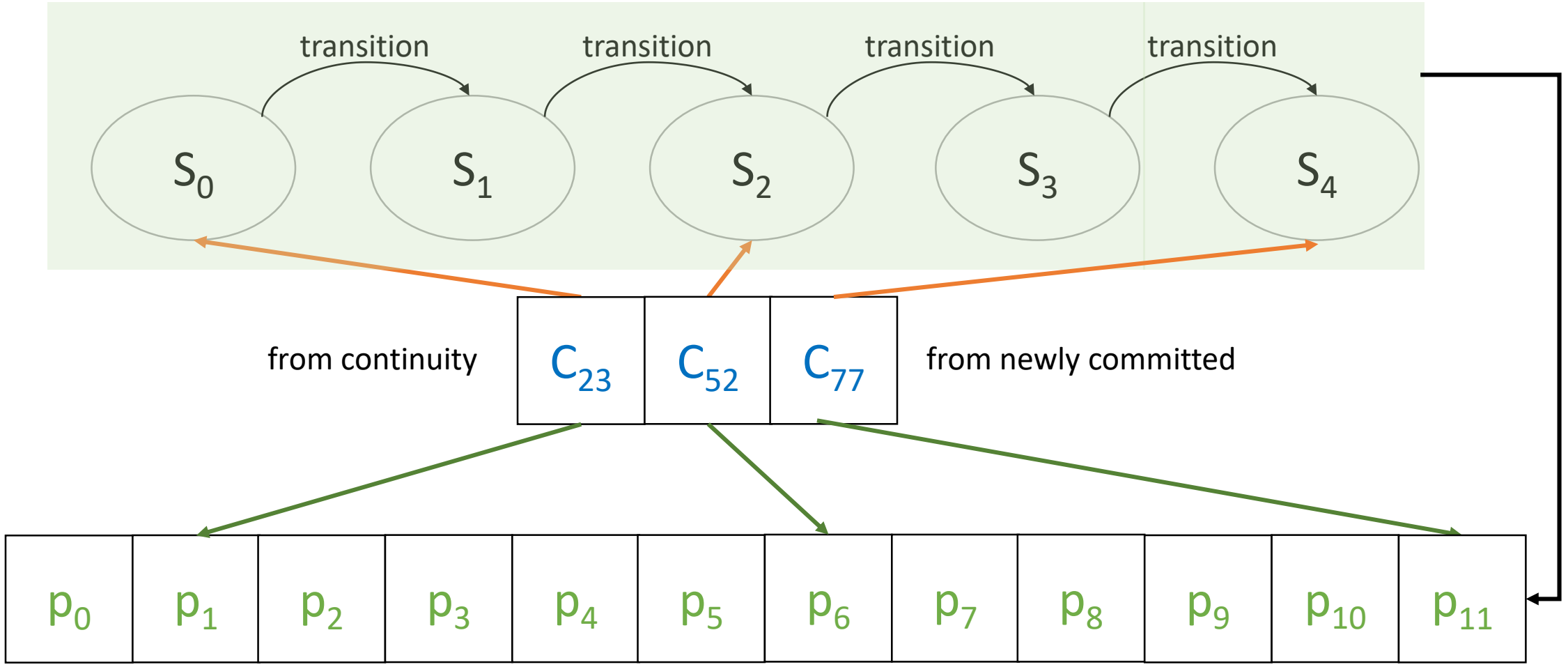
Proof: sequential consistency



Proof: sequential consistency



Proof: sequential consistency



Final thoughts

Implementation effort

I spent a large amount of time proving facts about lists

Component	# of lines
Common	58
Lemmas	724
Replication	140
Coco	115
Replication properties	122
Coco instant properties	120
Coco history properties	515

Specifications were fairly intuitive; took a few hours

Majority of the effort related to lemmas for sequential consistency

Future work

- Extend the CFTR spec to include:
  - snapshots
  - dynamic membership
  - majorities

- Revise $Coco_m$ to allow reasoning about:
  - confidentiality
  - governance
  - disaster recovery