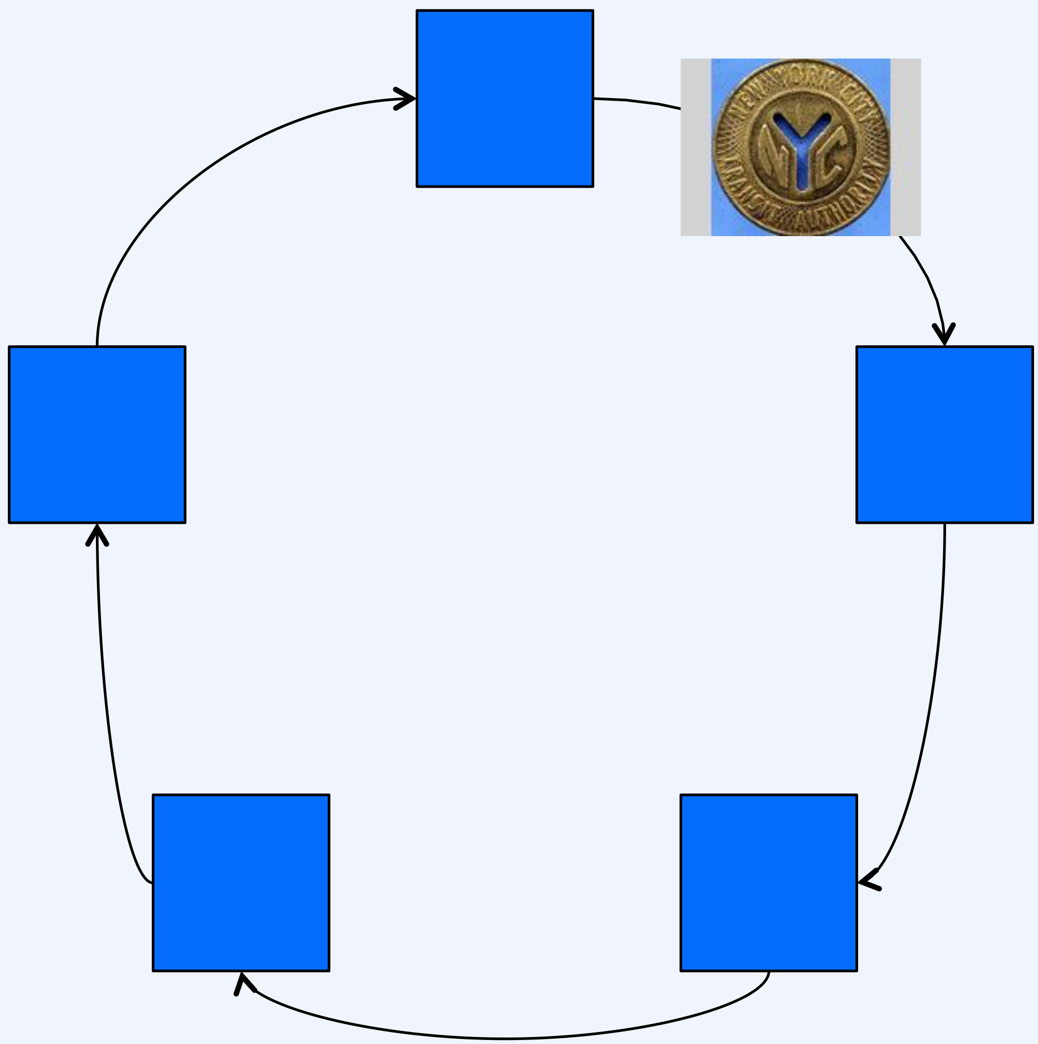
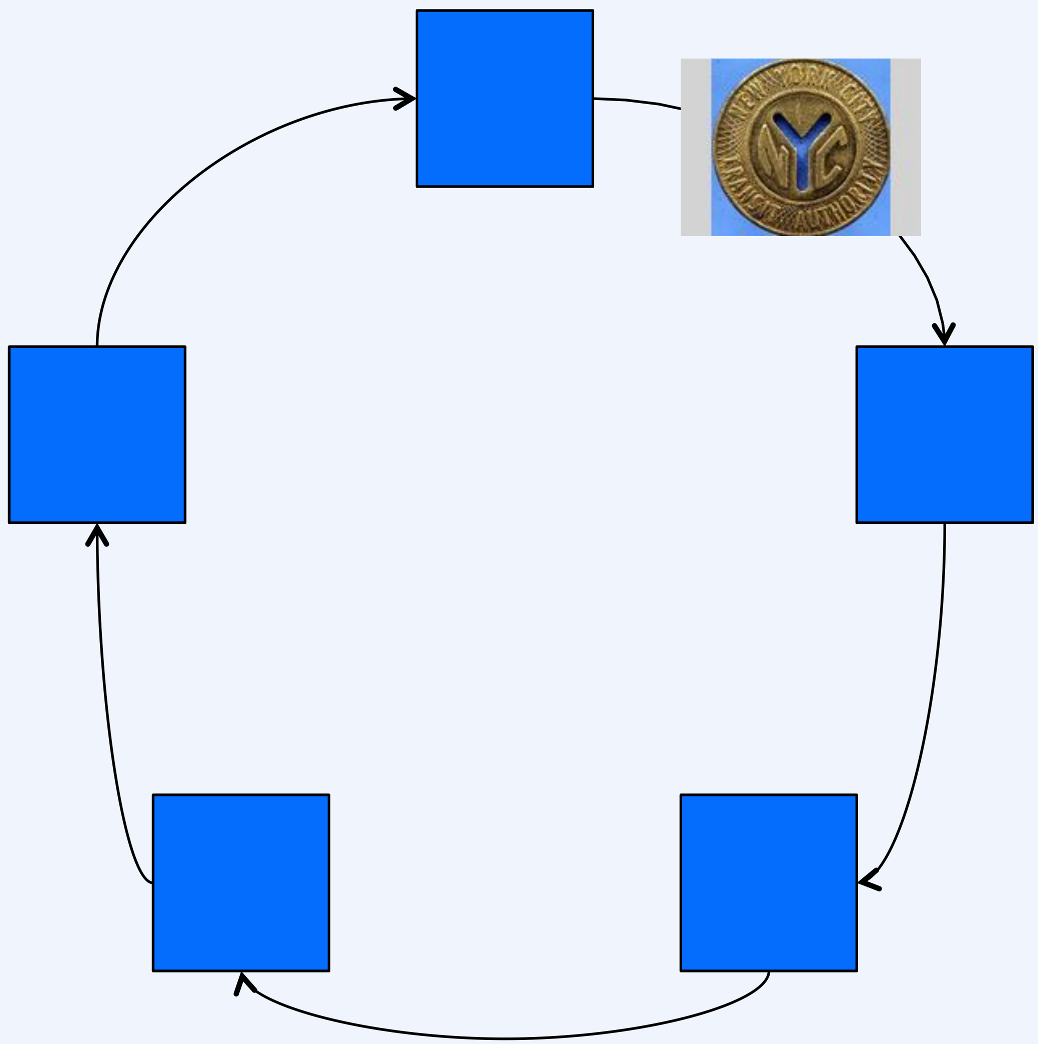
CS 138: Self-Stabilizing Systems

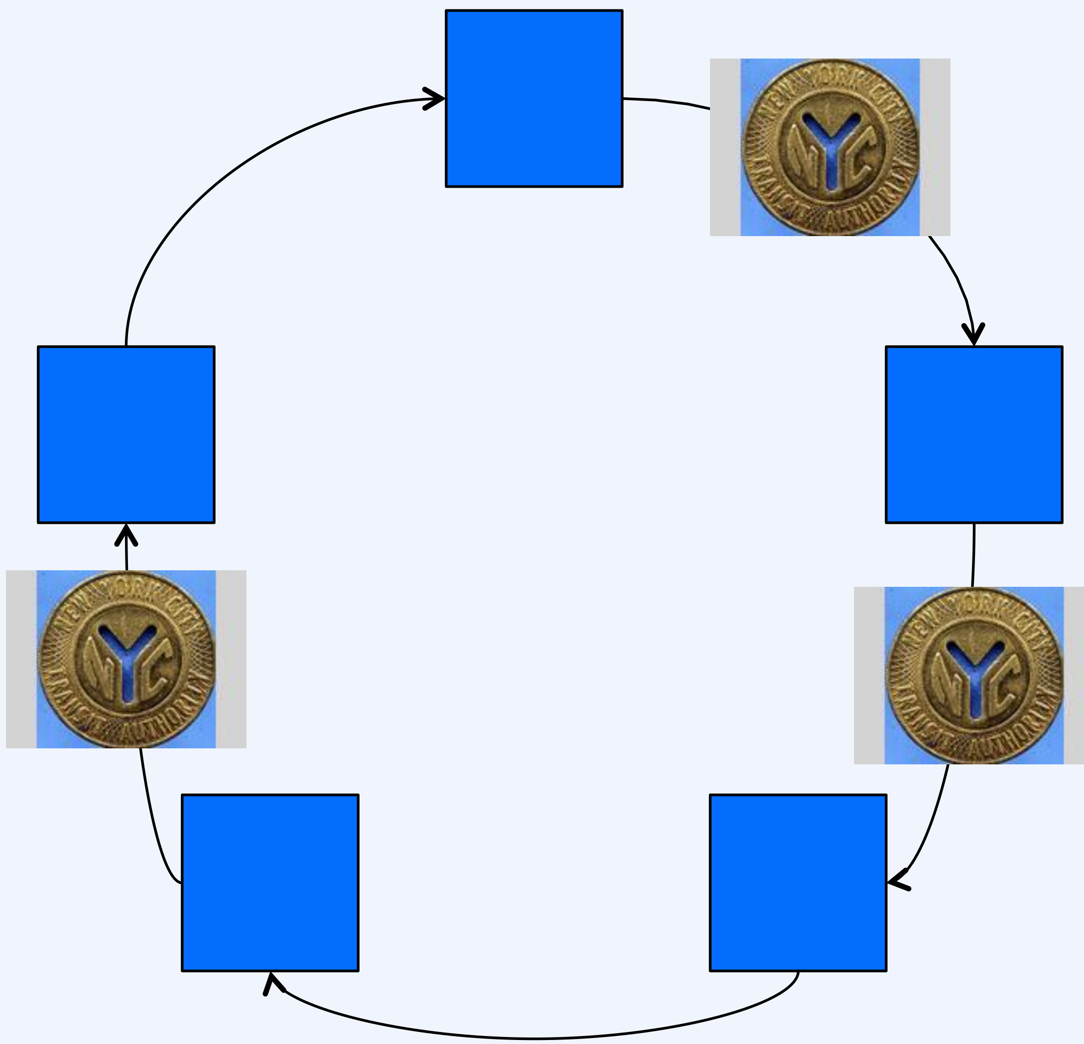
Token Ring



Token Ring Problem (1)



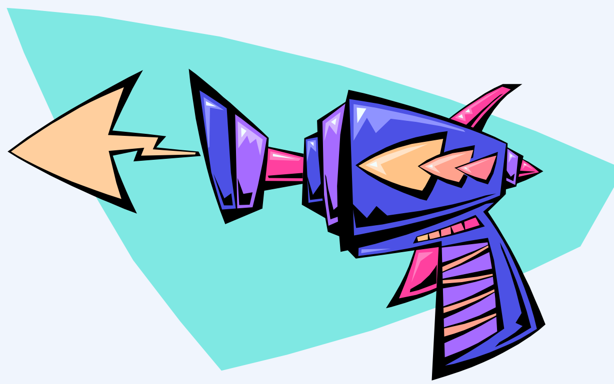
Token Ring Problem (2)



Enter Dijkstra

Self-Stabilizing Systems

- A distributed system has a set of legal states
- Suppose it's zapped by some outside force and enters an illegal state



- Can it be constructed so that it is guaranteed to return to a legal state in a bounded amount of time?

Notation, etc.

- **Guarded commands**

  `guard → command`

  - execute *command* when *guard* is true
- **Token ring**
  - node.state
    - integer state of node
  - node.next
    - next node (clockwise)
  - node. prev
    - previous node (counter clockwise)

Solution

- N nodes, each with k states, k > N
- Special distinguished node

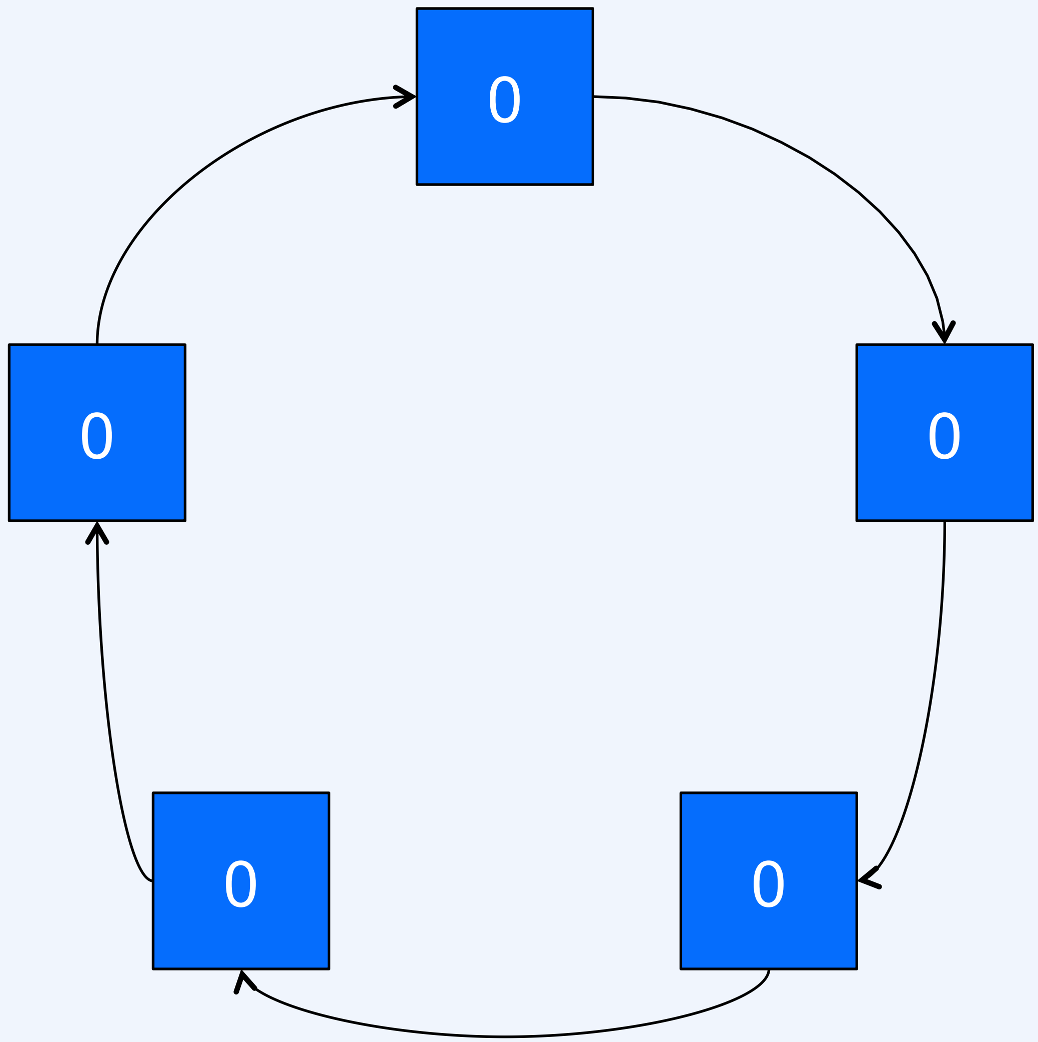
```
(node.prev.state == node.state) →
        node.state++(mod k)
```

- All other nodes

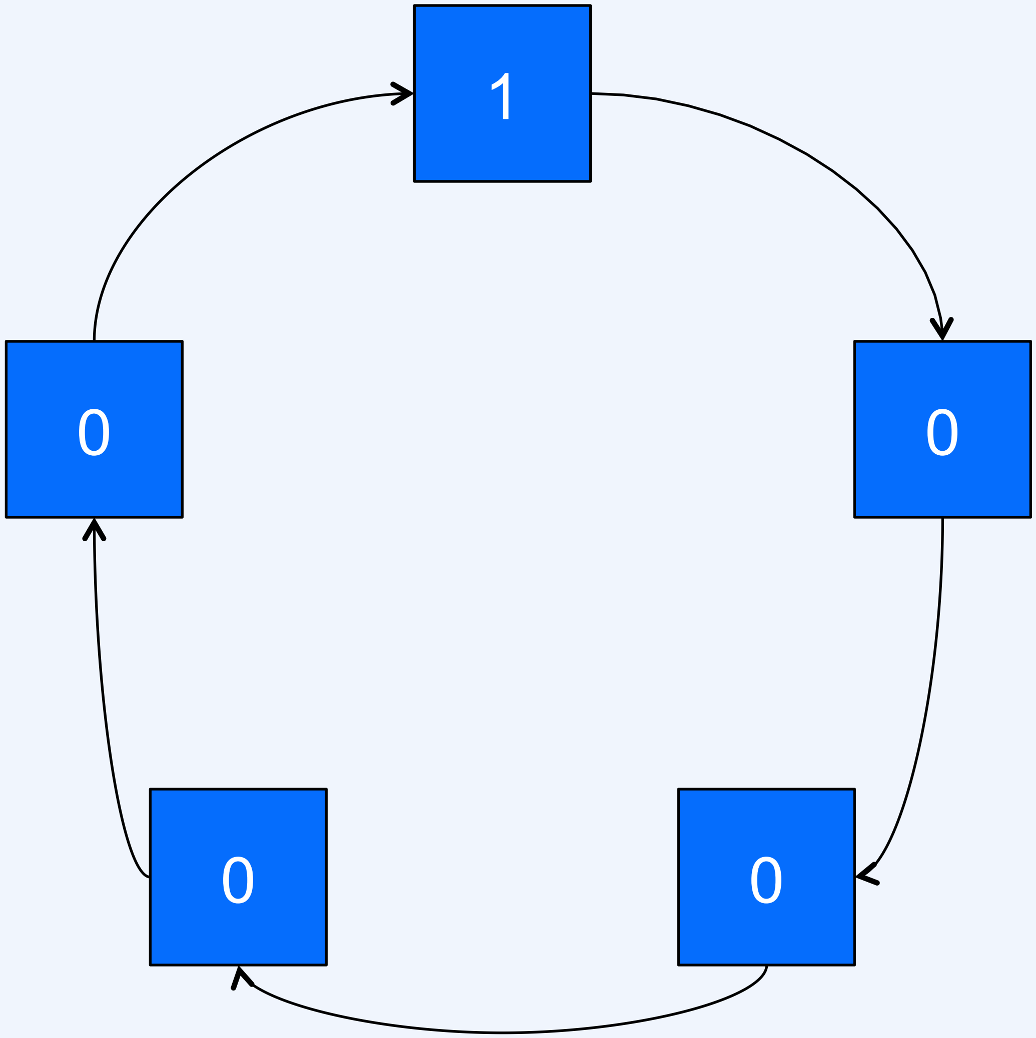
```
(node.prev.state != node.state) →
        node.state = node.prev.state
```

- Legal system states
  – exactly one guard is true

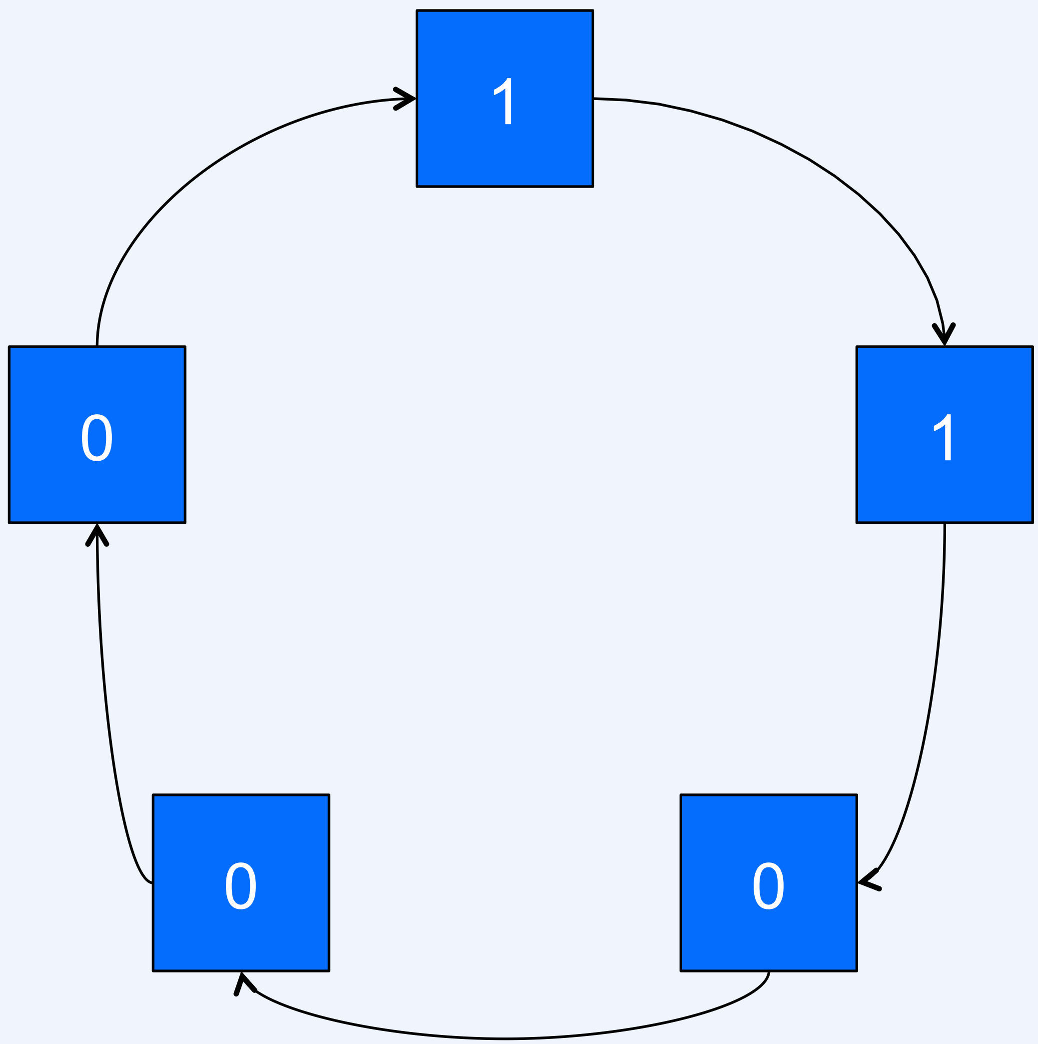
Example (1)



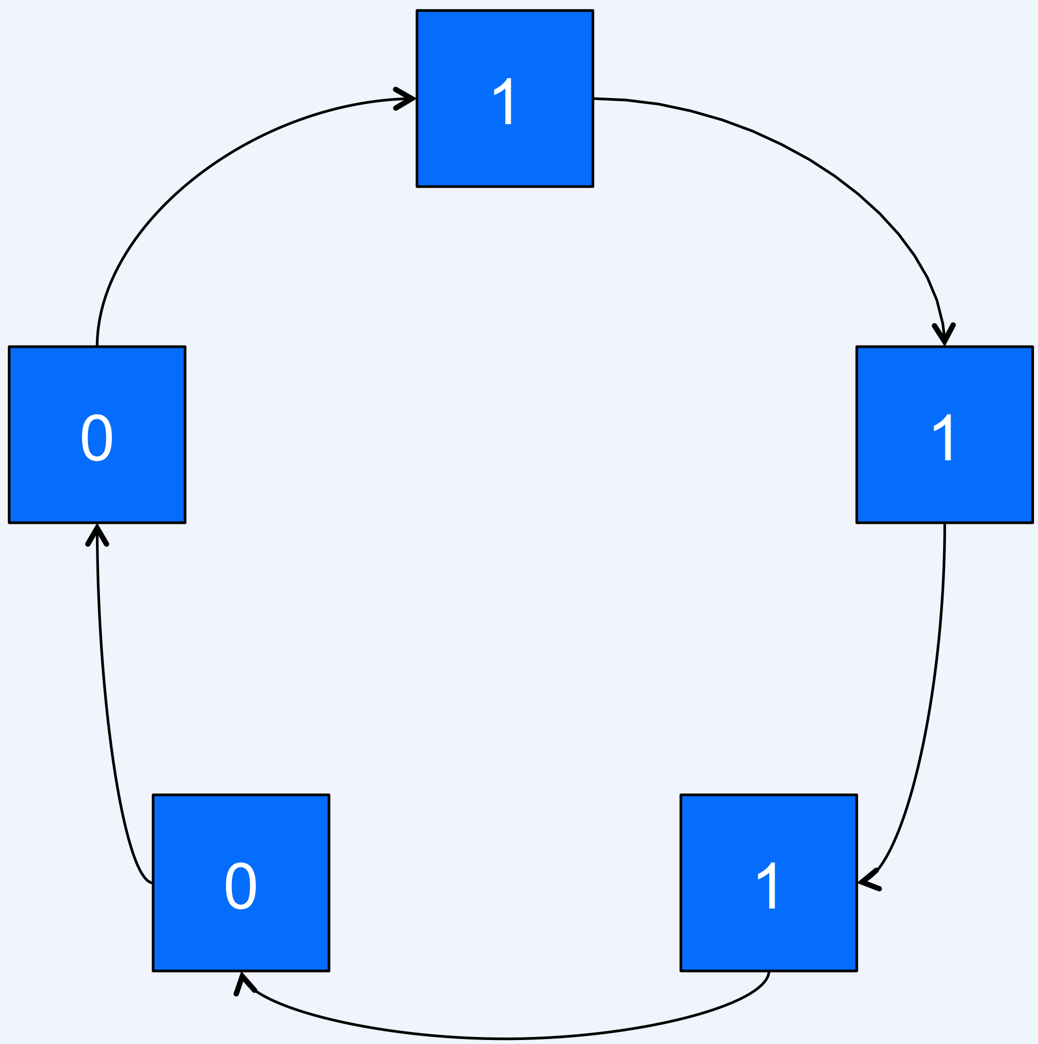
Example (2)



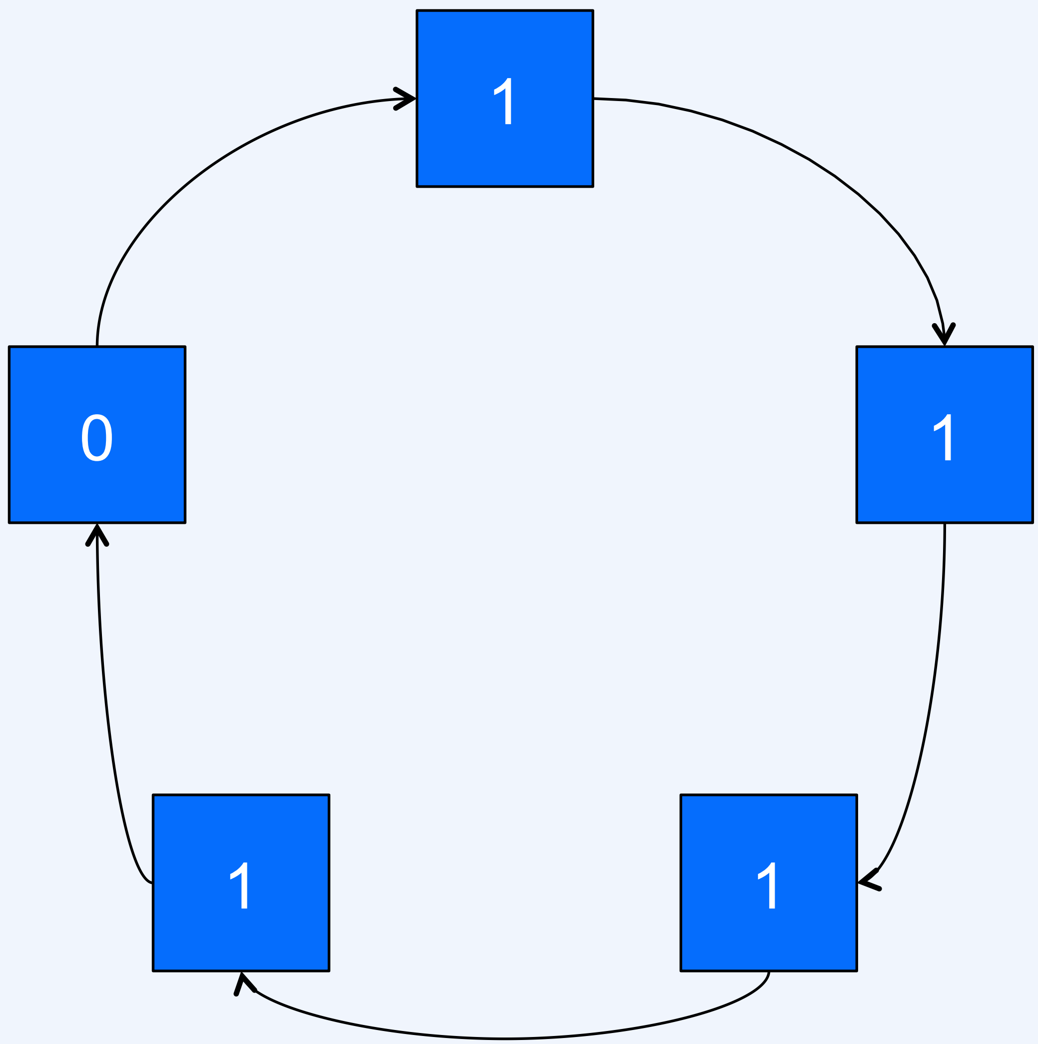
Example (3)



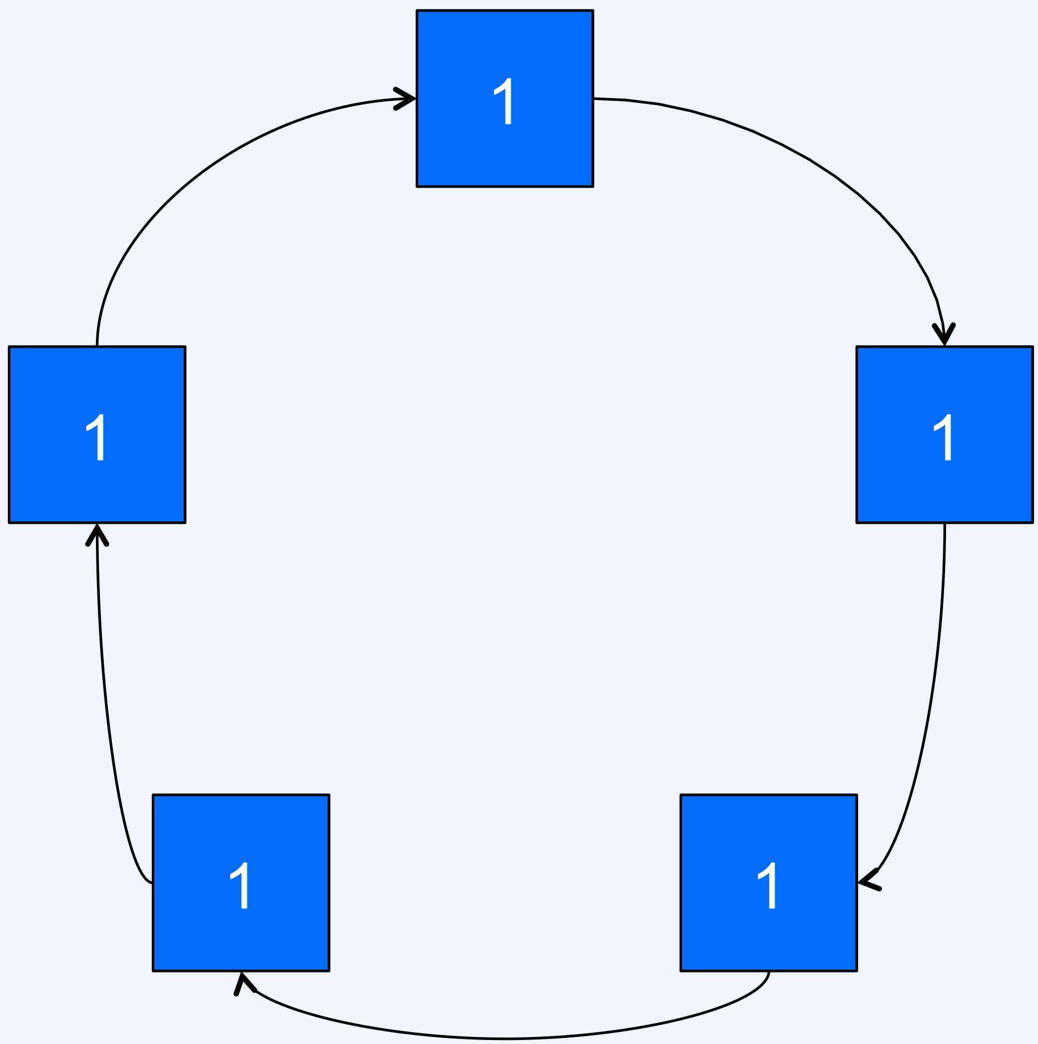
Example (4)



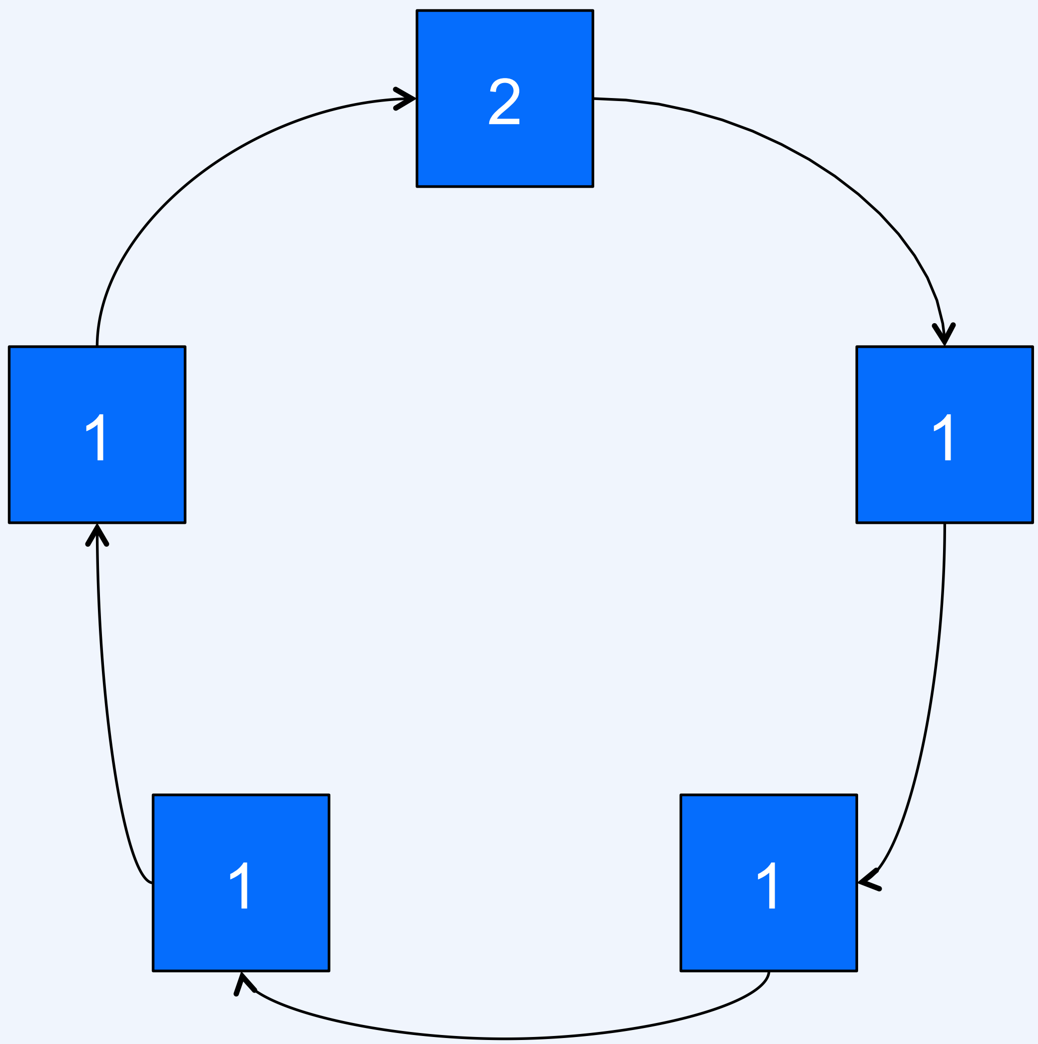
Example (5)



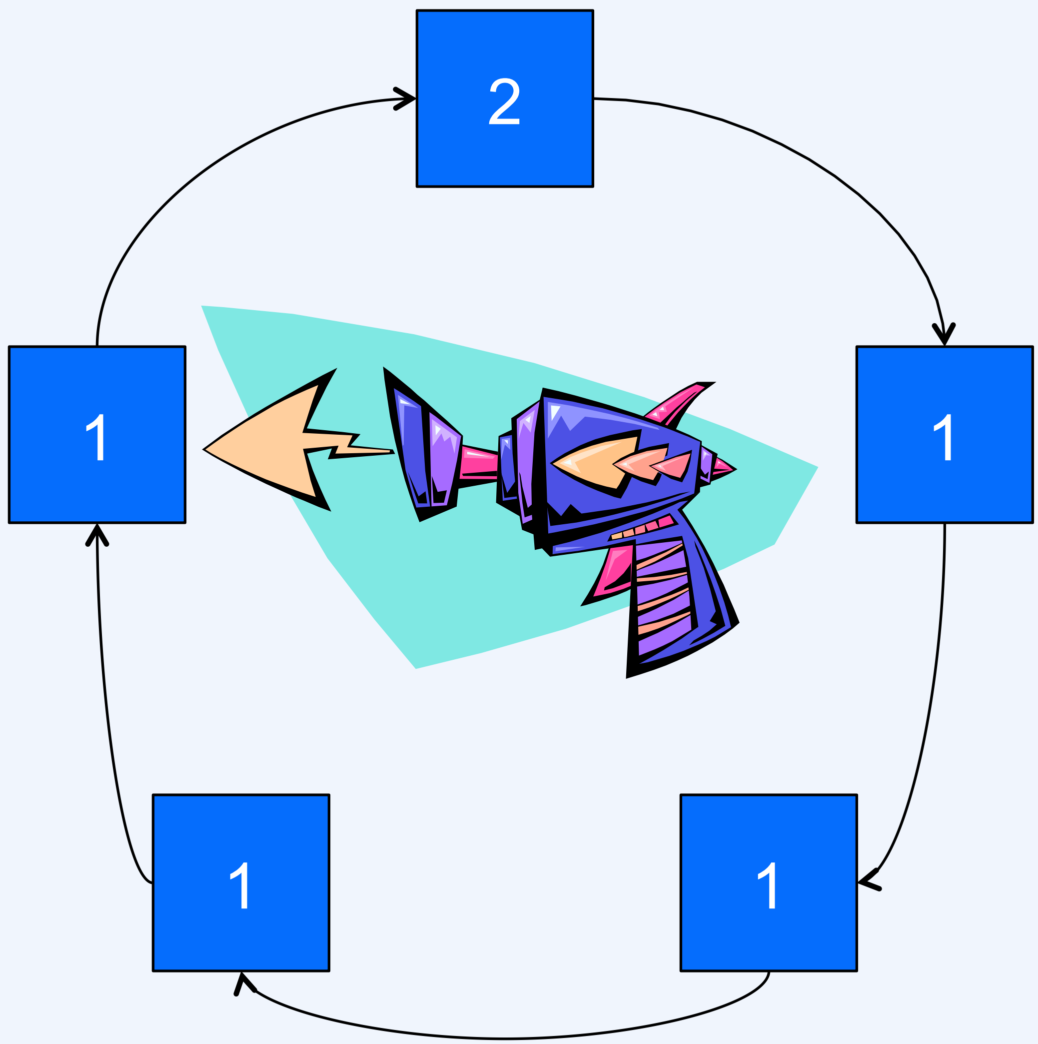
Example (6)



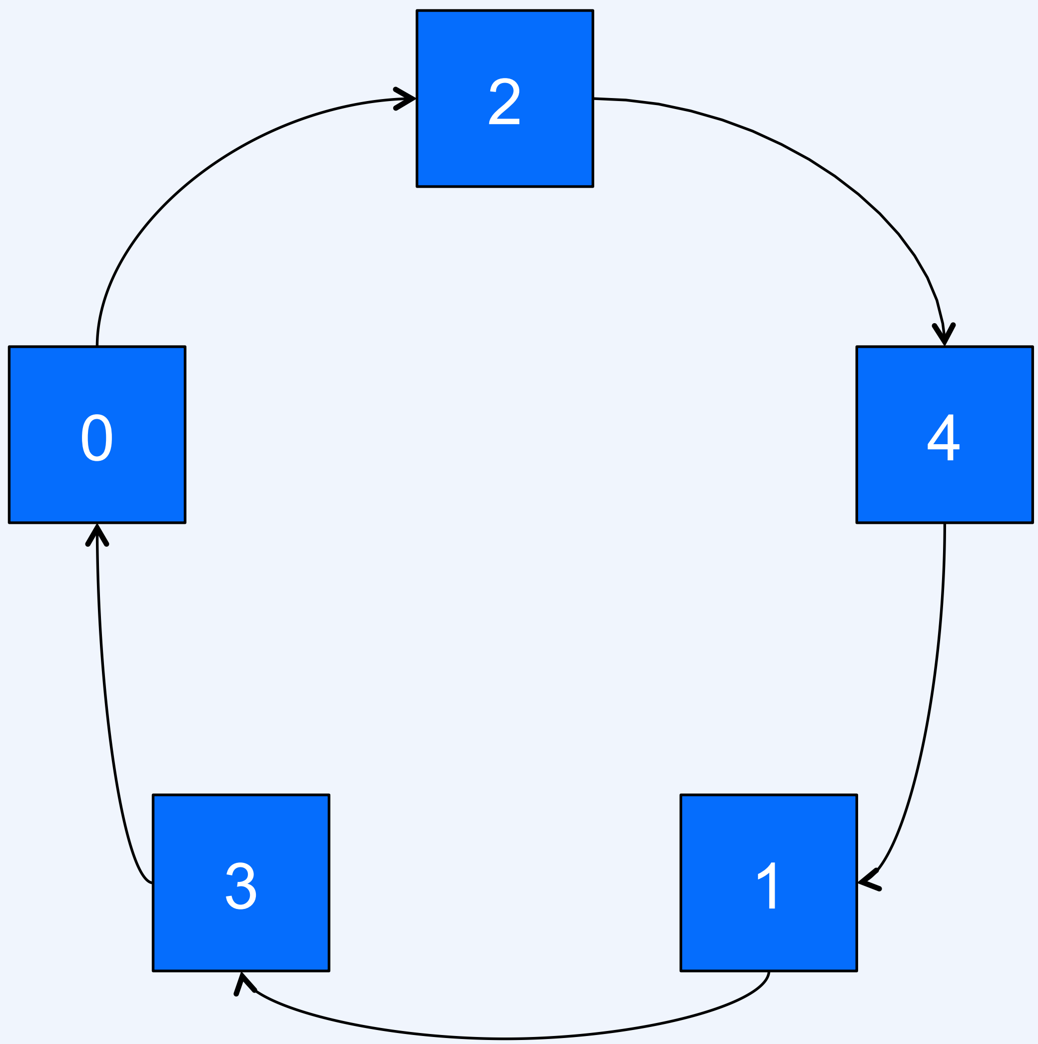
Example (7)



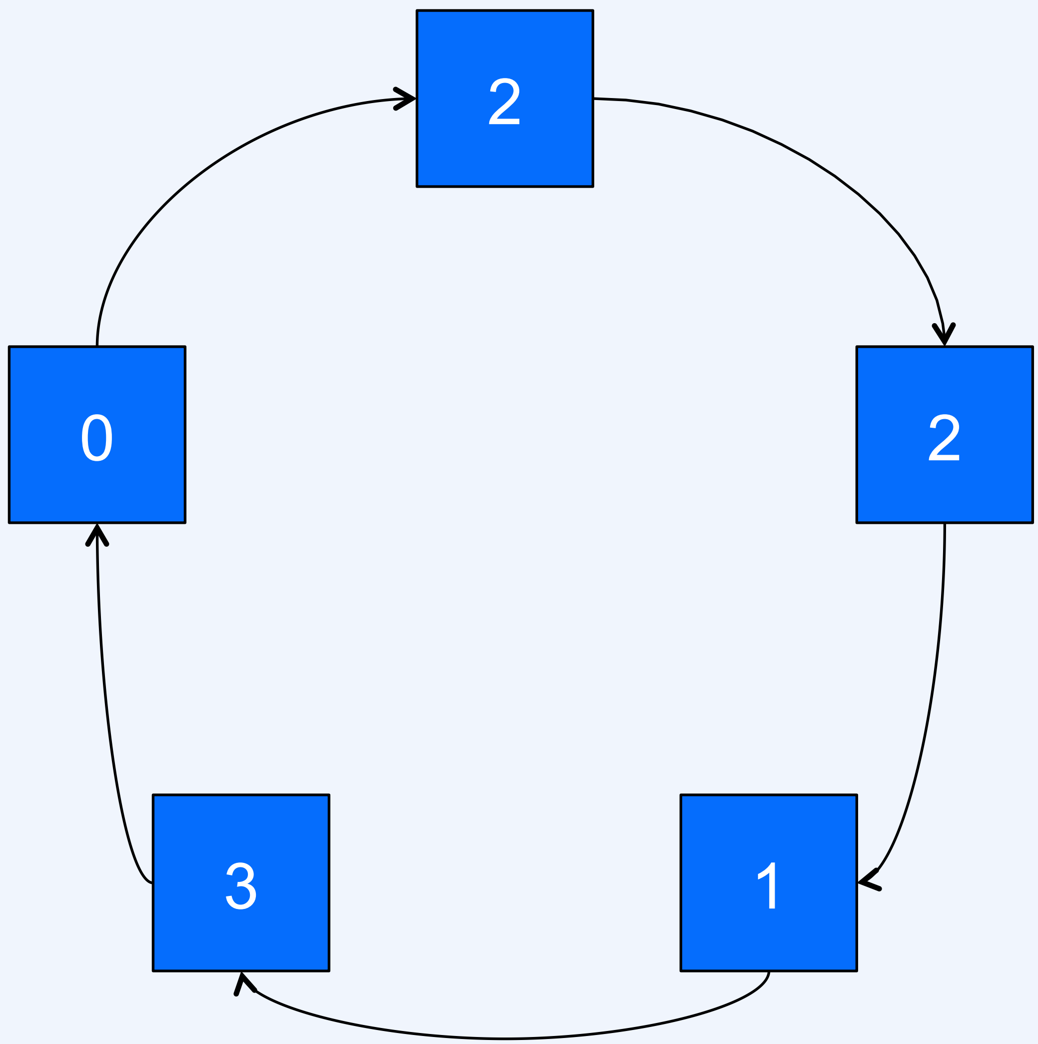
Example (8)



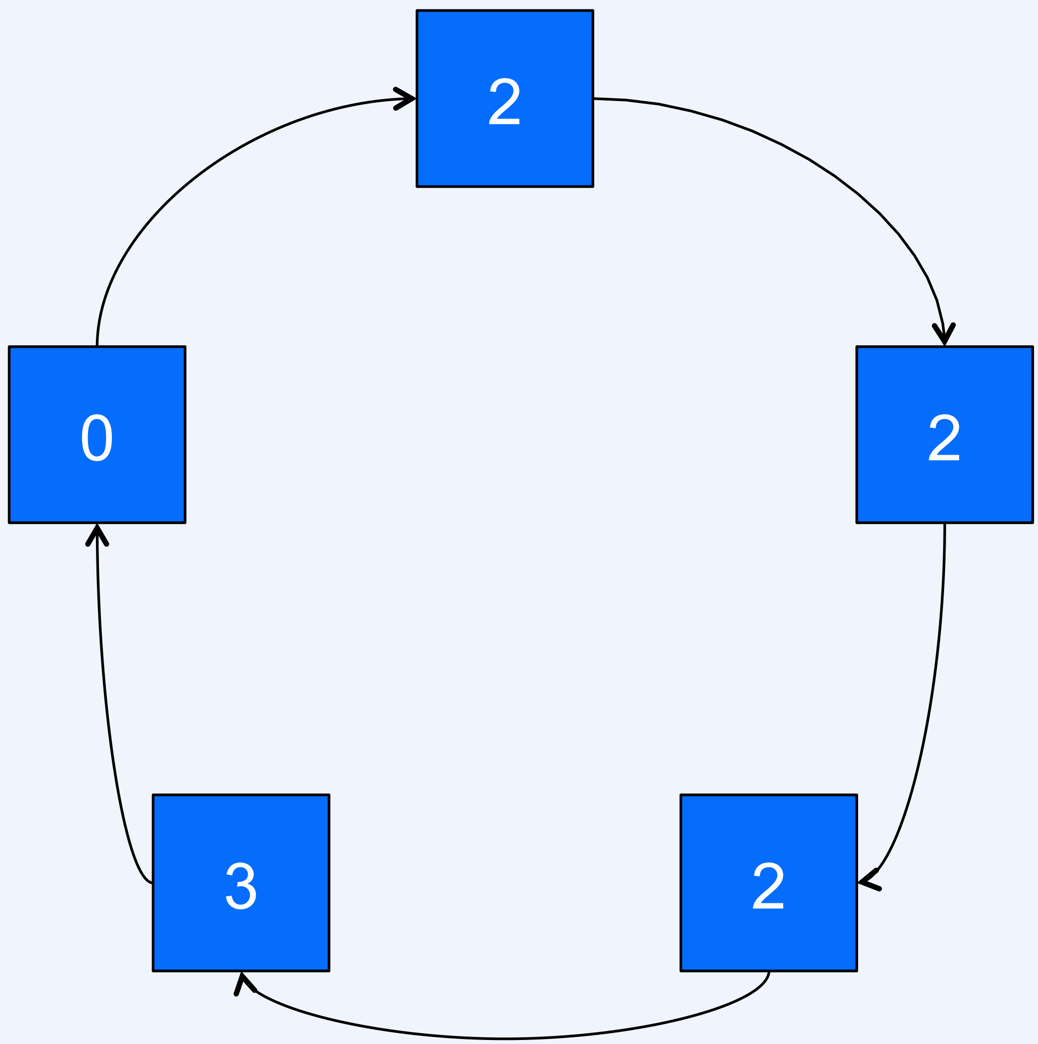
Example (9)



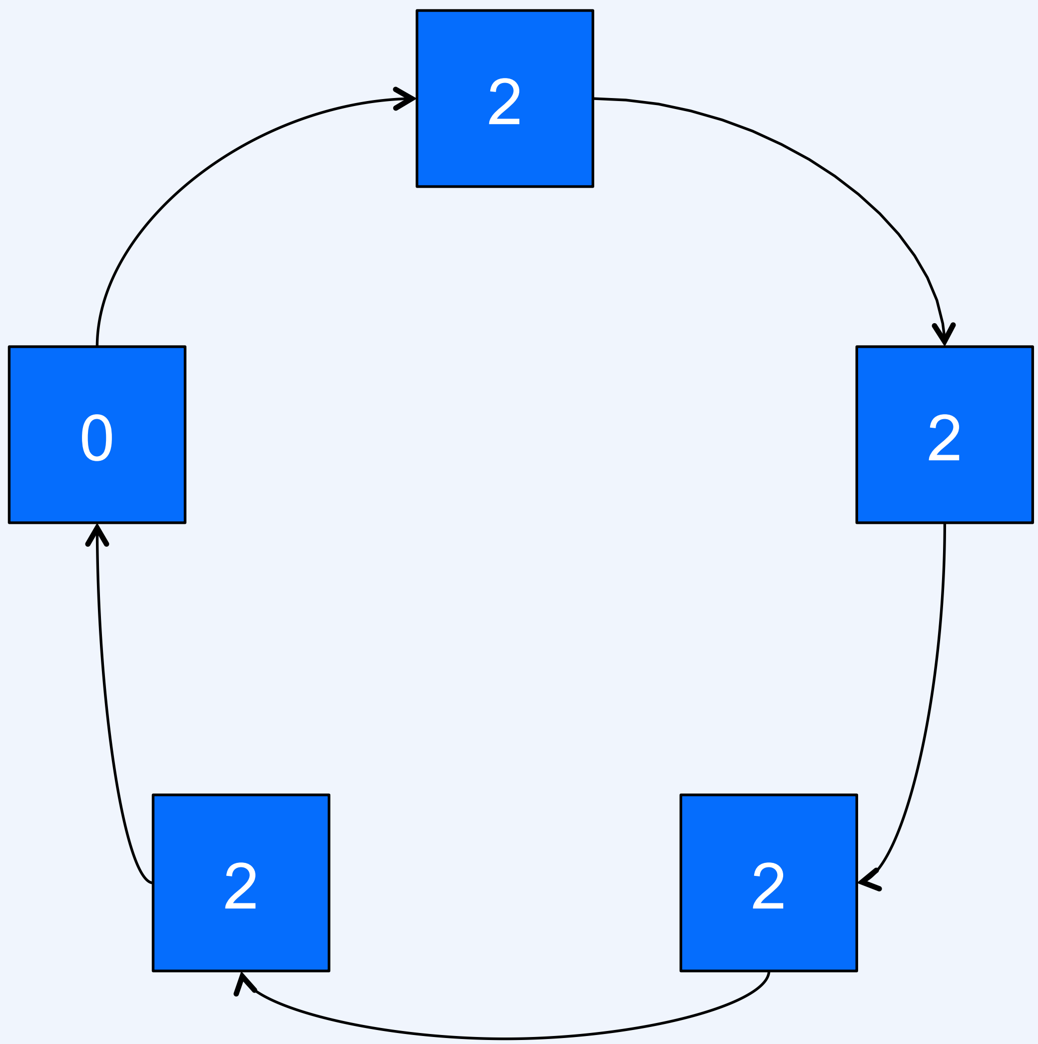
Example (10)



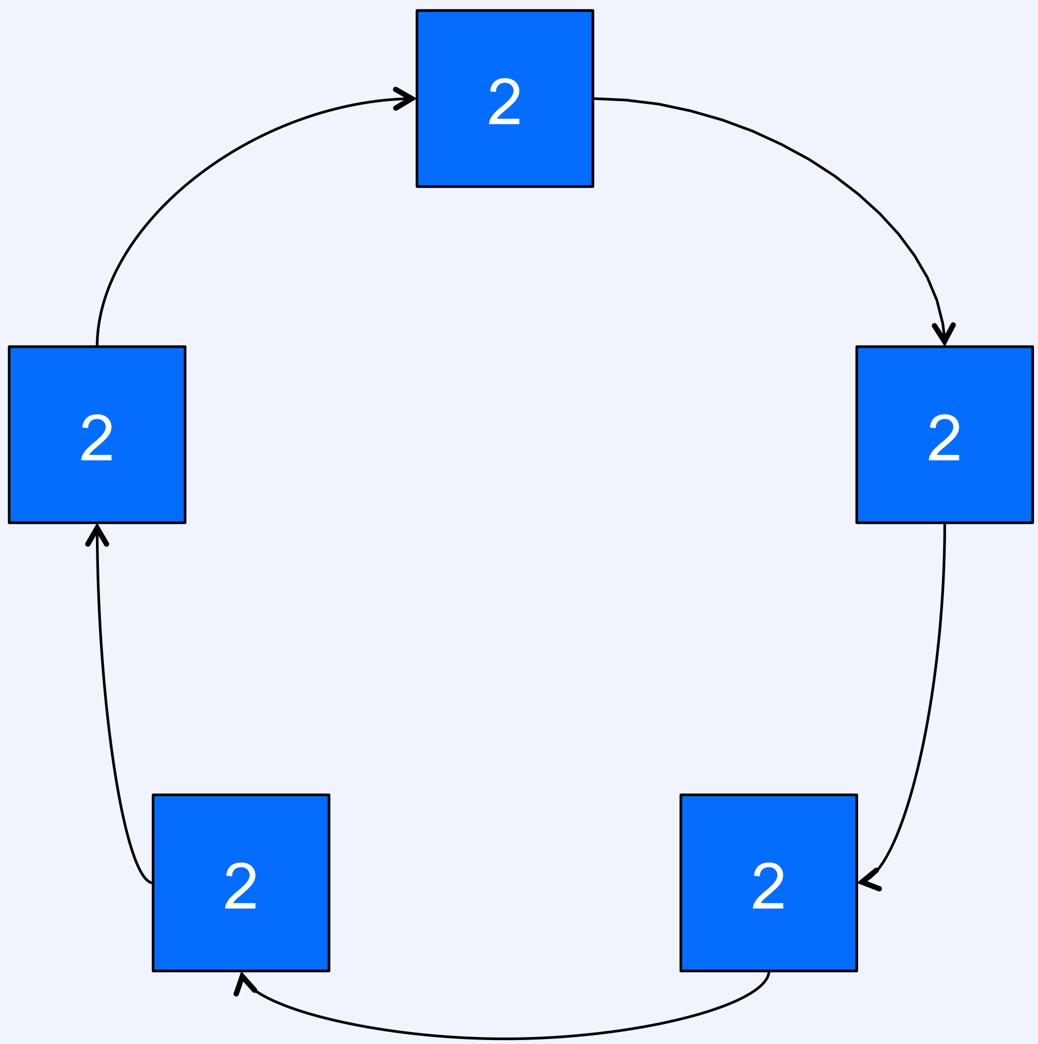
Example (11)



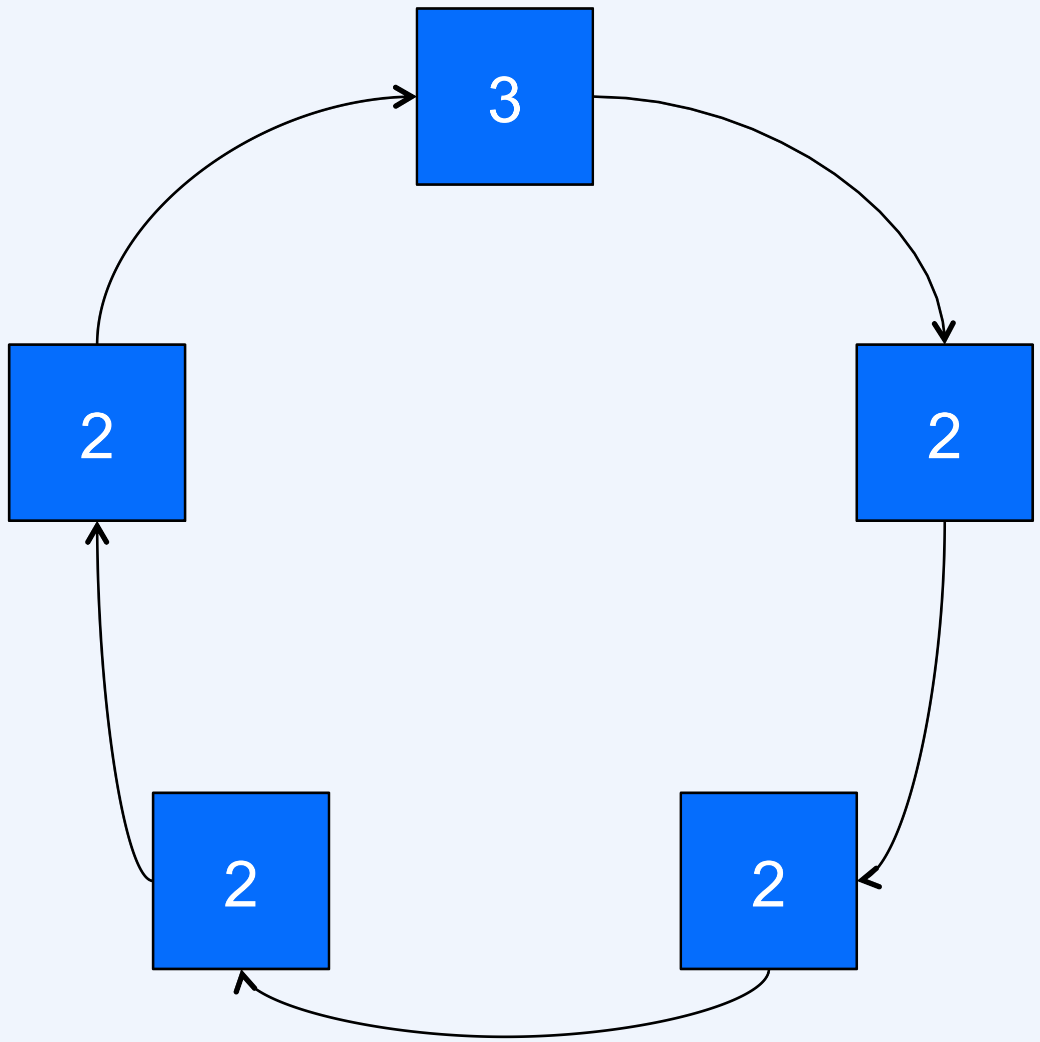
Example (12)



Example (13)



Example (14)



Also …

- Gave solutions with 4-state machines and 3-state machines
- Someone later proved that it cannot be done with 2-state machines

Proof

- Dijsktra didn't bother …
- It's up to us

Proof (1)

- **Explain why it is that at any particular moment, at least one guard must be true, even if the system has been zapped**

- **Special distinguished node**

```
(node.prev.state == node.state) →
        node.state++(mod k)
```

- **All other nodes**

```
(node.prev.state != node.state) →
        node.state = node.prev.state
```

Proof (2)

- Show that if all nodes have the same value for their states, the system is stable
  - stable: the system is in a state in which only one node's guard is true; whenever the system changes global state legally, it goes to a global state in which the next node's guard is the only one that's true

- Special distinguished node

```
(node.prev.state == node.state) →
        node.state++(mod k)
```

- All other nodes

```
(node.prev.state != node.state) →
        node.state = node.prev.state
```


Proof (3)

- Show that if node 0's state is greater than those of all other nodes, the system will necessarily reach a stable global state.

- Special distinguished node

```
(node.prev.state == node.state) →
        node.state++(mod k)
```

- All other nodes

```
(node.prev.state != node.state) →
        node.state = node.prev.state
```


Proof (4)

- Assume now that each node's state value is an unbounded non-negative integer (i.e., k is infinite). Show that, regardless of its current state, the system will necessarily reach a global state in which node 0's state is greater than those of all others.

- Special distinguished node

```
(node.prev.state == node.state) →
        node.state++(mod k)
```

- All other nodes

```
(node.prev.state != node.state) →
        node.state = node.prev.state
```


Proof (5)

- Redo part 4, this time assuming k>=n: the system will necessarily reach a global state in which node 0's state is greater than those of all others

- Special distinguished node

```
(node.prev.state == node.state) →
        node.state++(mod k)
```

- All other nodes

```
(node.prev.state != node.state) →
        node.state = node.prev.state
```

Proof (6)

- Show that the system won't necessarily ever enter a stable state after being zapped if k<n

- Special distinguished node

```
(node.prev.state == node.state) →
        node.state++(mod k)
```

- All other nodes

```
(node.prev.state != node.state) →
        node.state = node.prev.state
```