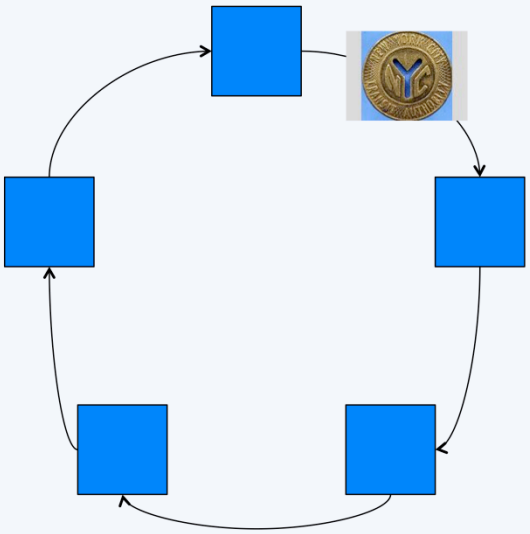
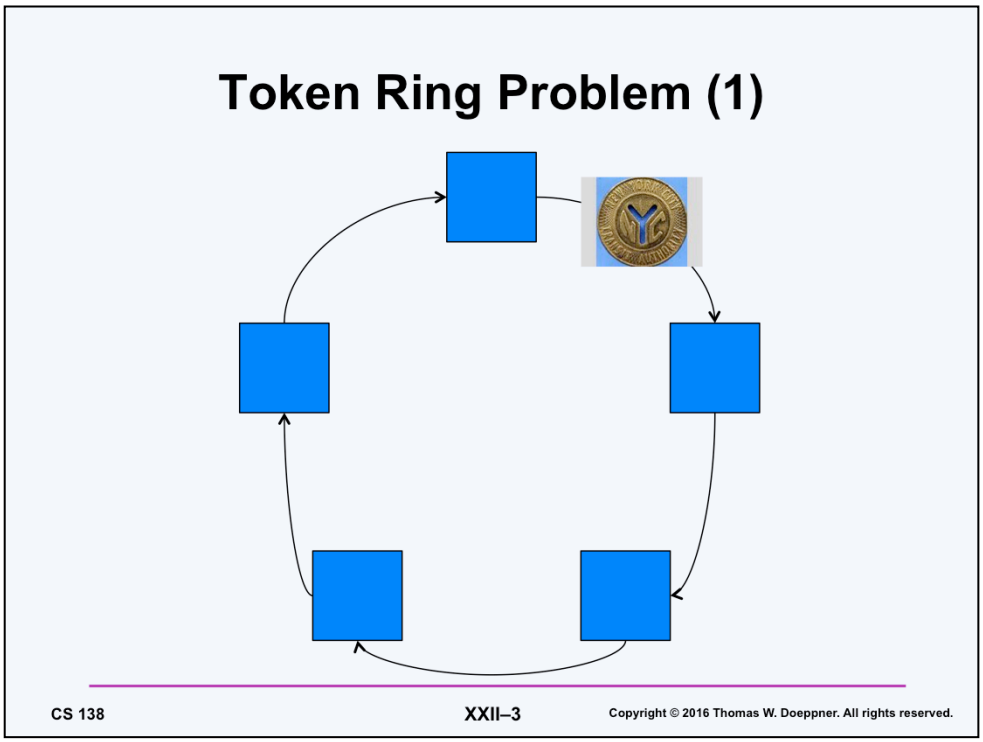
CS 138: Self-Stabilizing Systems

Copyright © 2016 Thomas W. Doeppner. All rights reserved.

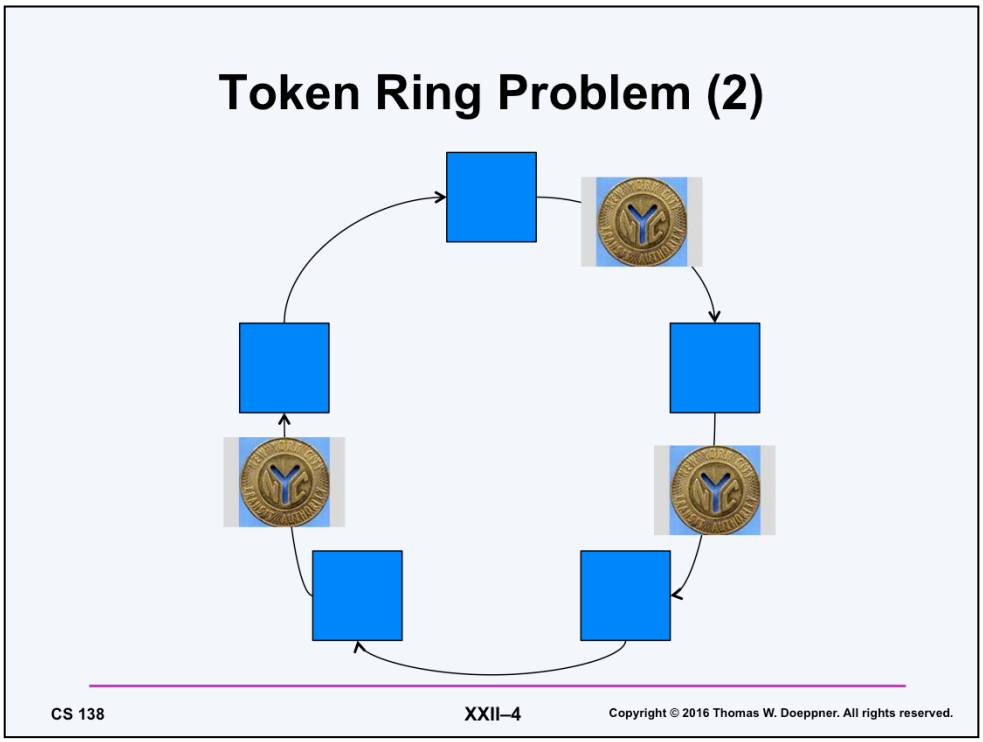
Token Ring



Copyright © 2016 Thomas W. Doeppner. All rights reserved.



What happens if the token disappears?



What if extra tokens spontaneously appear?

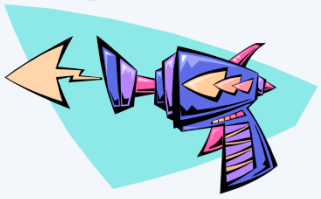
Enter Dijkstra

 Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Among his many contributions, Edsger W. Dijkstra wrote a seminal paper on "Self-stabilizing Systems in Spite of Distributed Control." It was all of two pages long (1.75, to be precise) and appeared in Communications of the ACM, Vol. 17, No. 11 (November 1974). One of the journal's editors supplied a comment on the paper: "the appreciation is left as an exercise to the reader."

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?

Copyright © 2016 Thomas W. Doeppner. All rights reserved.

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)

Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Solution

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

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

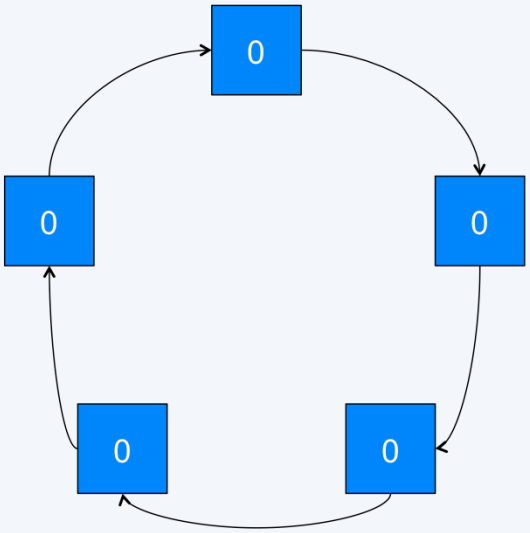
- All other nodes

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

- Legal system states
  - exactly one guard is true

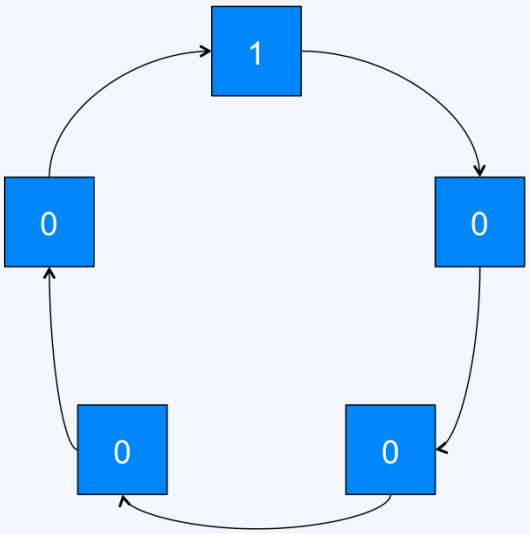
Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Example (1)



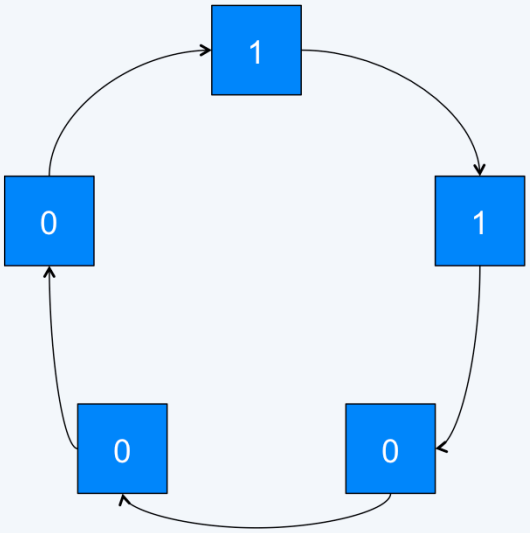
Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Example (2)



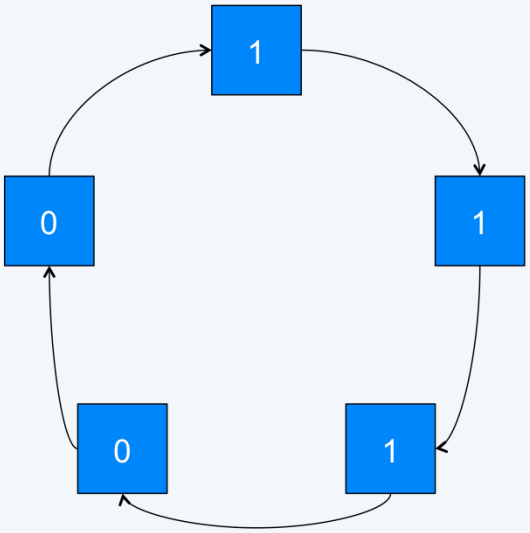
Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Example (3)



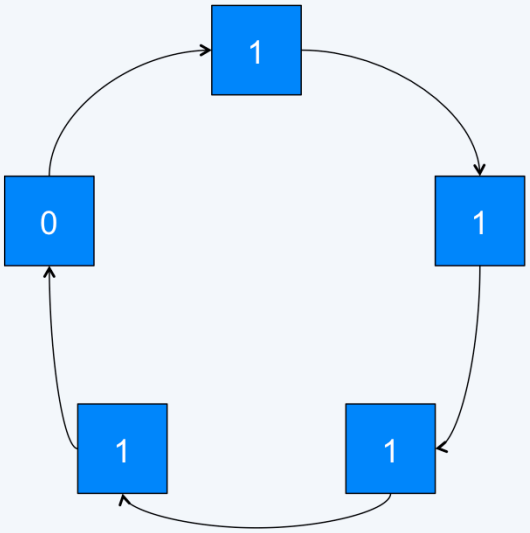
Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Example (4)



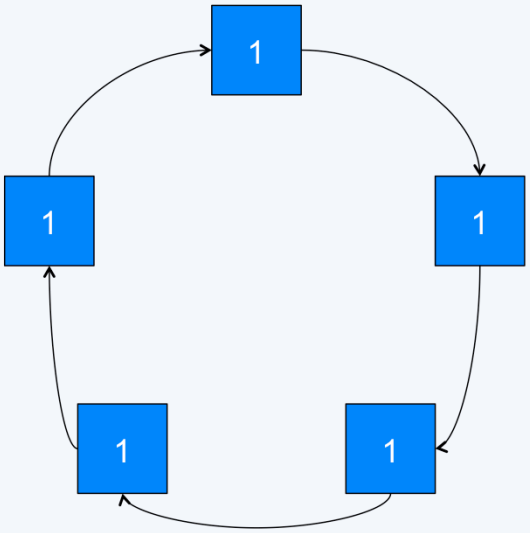
Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Example (5)

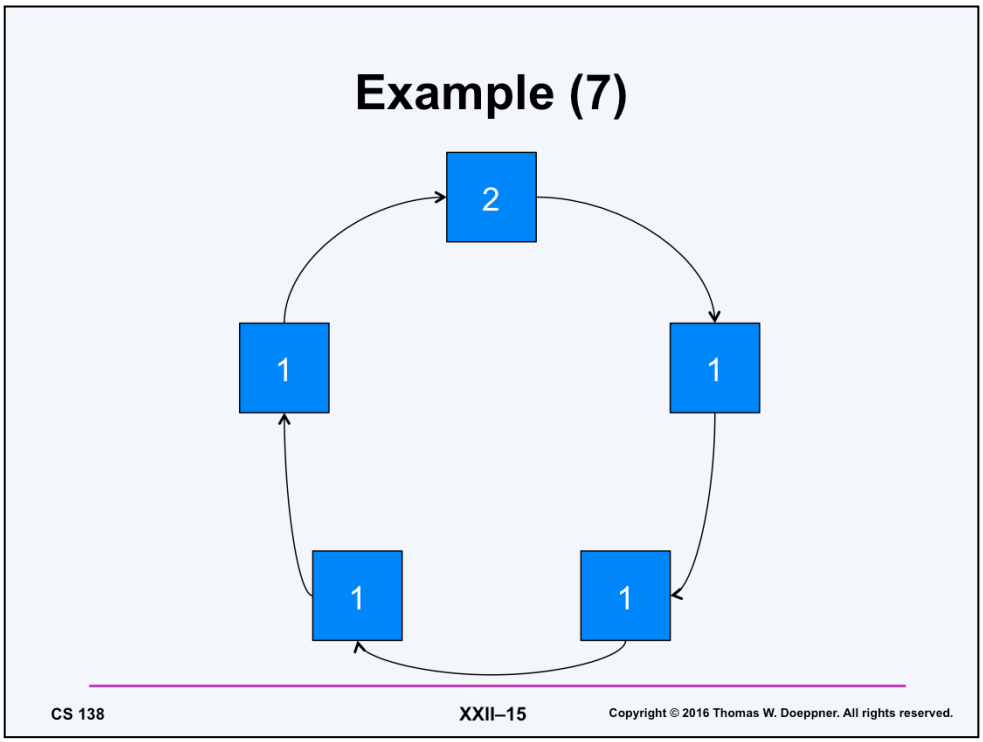


Copyright © 2016 Thomas W. Doeppner. All rights reserved.

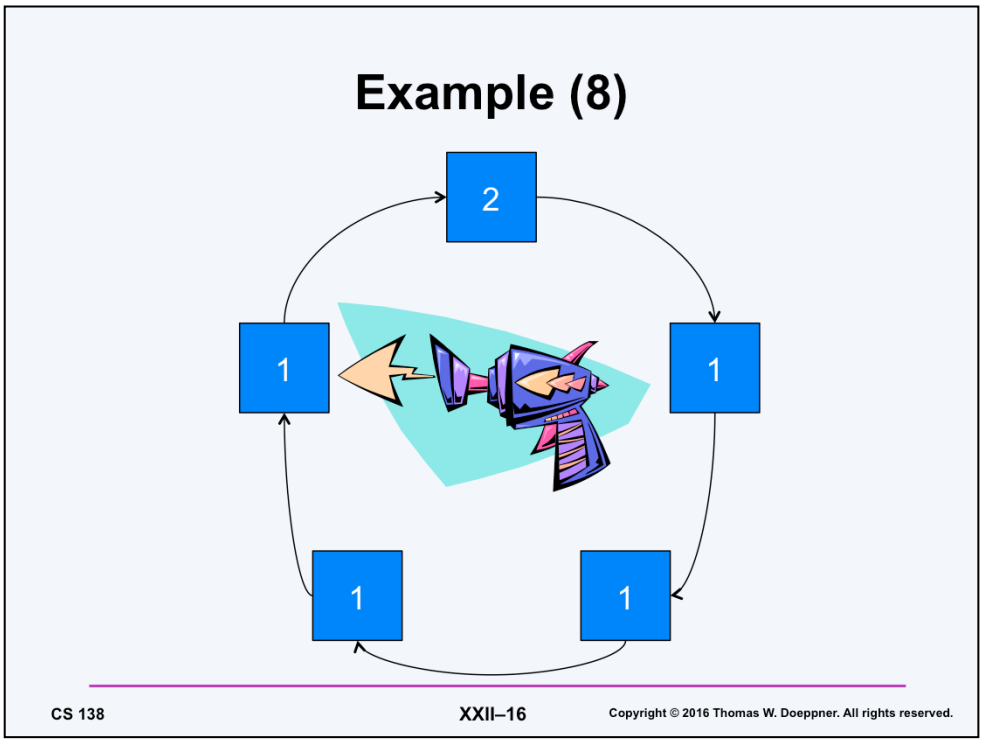
Example (6)



Copyright © 2016 Thomas W. Doeppner. All rights reserved.

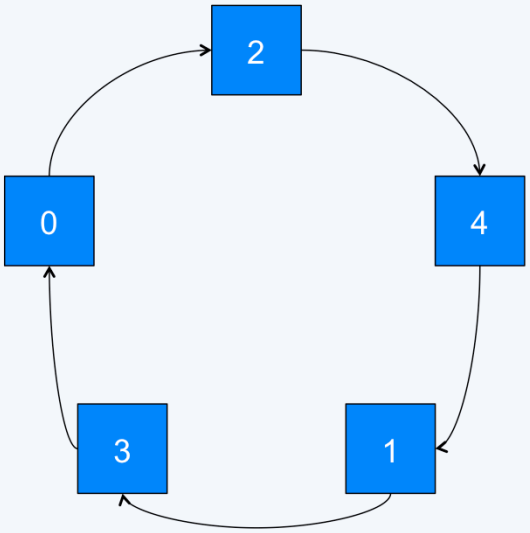


At this point, the system has finally achieved a legal state.



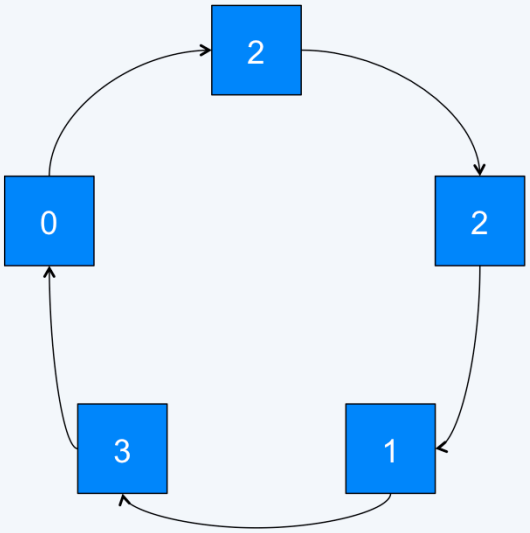
Now a machine gets zapped …

Example (9)



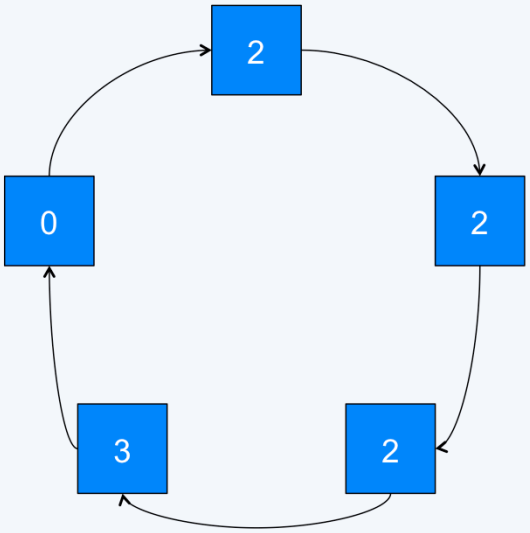
Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Example (10)



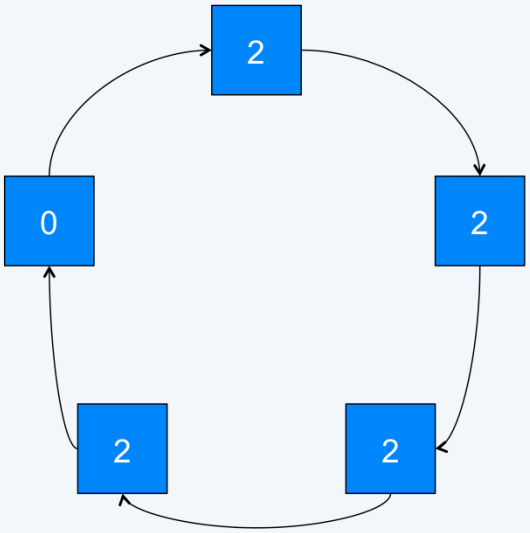
 Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Example (11)



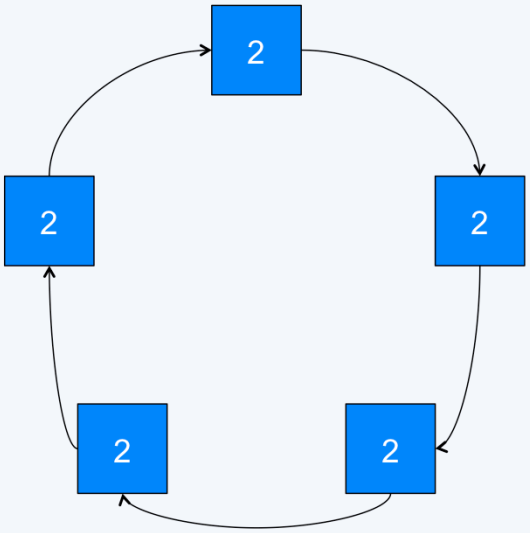
Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Example (12)

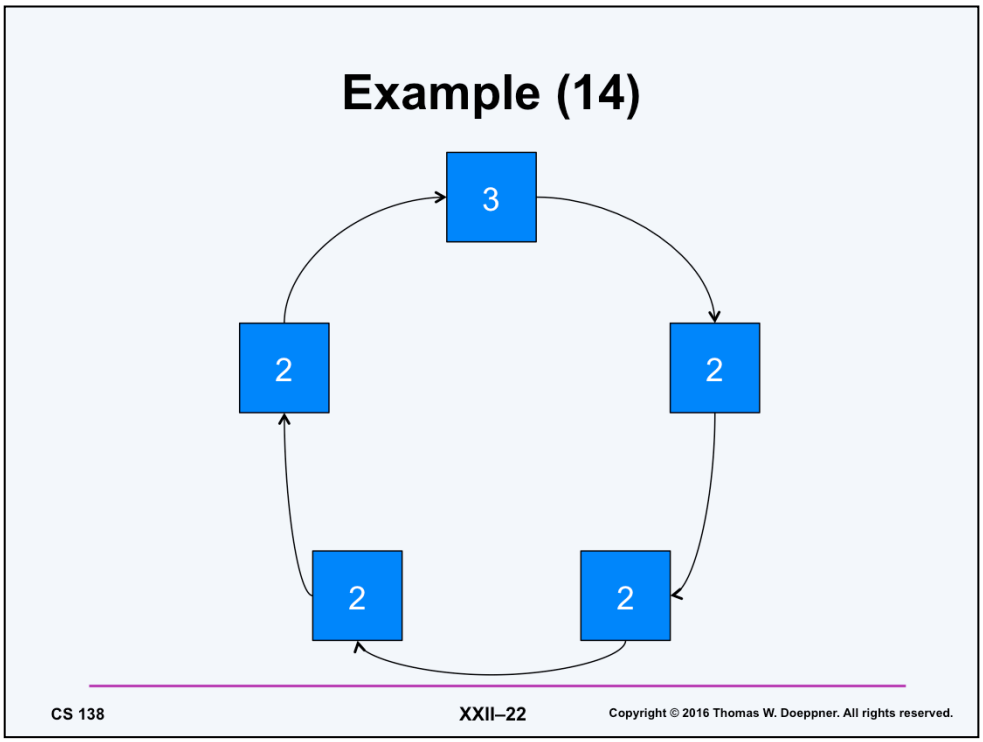


 Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Example (13)



Copyright © 2016 Thomas W. Doeppner. All rights reserved.



Once again, a legal state is reached.

Also …

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

Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Proof

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

Copyright © 2016 Thomas W. Doeppner. All rights reserved.

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
```


Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Either all nodes have the same state or there are at least two nodes with different states. If the former, then node 0's guard is true. If the latter, then there must be at least two nodes whose states are different from their predecessors. At least one of these nodes is not the distinguished node, and thus its guard is true.

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
```


Copyright © 2016 Thomas W. Doeppner. All rights reserved.

In this global state, only node 0's guard is true. Once its command is executed, its state becomes one greater (mod k) and its guard is no longer true, but its successor's guard is now true, and no other guards are true. Once this node's command is executed, its state is replaced with its predecessor's state and its guard is no longer true (and no other guards are true). This continues around the ring until we are back to node 0. At this point, all states again have equal values, and thus only node 0's guard is true.

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
```


Copyright © 2016 Thomas W. Doeppner. All rights reserved.

In such a global state, node 0's guard is false, but its successor's guard is true. Regardless of what else happens in the system, this condition will hold until the successor executes its command. At that point, its state becomes equal to that of node 0 and its guard becomes false. The system is now in state in which the next node's guard must be true, and this condition will hold until that node executes its command. When it does, its state becomes that of its predecessor, which is the same as that of node 0. Thus node 0's state will propagate all the way around the ring, until all nodes have the same state. Not until this global state is reached will node 0's guard become true. However, as shown in part 2, the system is now in a stable 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
```


Copyright © 2016 Thomas W. Doeppner. All rights reserved.

Assume there exists at least one node whose state is greater than or equal to that of node 0, and that it's not the case that all nodes have the same value. We know from part 1 that there's always at least one node whose guard is true. Each time a command (associated with a true guard) is executed, either node 0's state gets larger, or one other node's state is set equal to its predecessor's state (that previously was different from it).

Let node i be the first node beyond node 0 whose state is different from node 0's. After some number of executions, either node 0's state will increase by 1, or node i will set its state to be the same as node 0's (and thus the index of the first node beyond node 0 whose state is different from 0's increases by at least 1). Thus, in a finite number of executions, either node 0's state increases by 1 or all nodes have the same state as node 0.

Furthermore, the maximum of the nodes' state values does not get larger unless node 0 has the state with the maximum value. Thus, eventually, either node 0's state becomes larger than all others or the system reaches a global state in which all nodes have the same state (and thus after the next command execution, node 0's state is larger than all other's).

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
```


Copyright © 2016 Thomas W. Doeppner. All rights reserved.

We can modify the proof of part 4 by changing all occurrences of "increase by 1" to "increase by 1 (mod k)". However, we have a problem with the last sentence because it is no longer clear that node 0's state can get larger than all others, since there is now a bound on its size (k-1). But we can safely say instead that, eventually, either node 0's state becomes 0 or the system reaches a global state in which all nodes have the same state. This clearly isn't what we're after. However, starting from this state we can repeat the argument of part 4: Again, let node i be the first node beyond node 0 whose state is different from node 0's. After some number of executions, either node 0's state will increase by 1 (mod k), or node i will set its state to be the same as node 0's. In a finite number of executions, either node 0's state increases by 1 (mod k) or all nodes have the same state as node 0. However long this takes, we will call it a round. As before, the maximum of the nodes' state values does not get larger unless node 0 has the state with the maximum value. After no more than n rounds (and thus node 0's state has not wrapped around), since we started with node 0's state being 0, either node 0's state is larger than all others or the system has reached a global state in which all nodes have the same 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
```


Copyright © 2016 Thomas W. Doeppner. All rights reserved.