



TEXAS A&M
UNIVERSITY®

ISCA 2018

Session 8B: Interconnection Networks

Synchronized Progress in Interconnection Networks (SPIN) : A new theory for deadlock freedom

Aniruddh Ramrakhyani

Georgia Tech
(aniruddh@gatech.edu)

Tushar Krishna

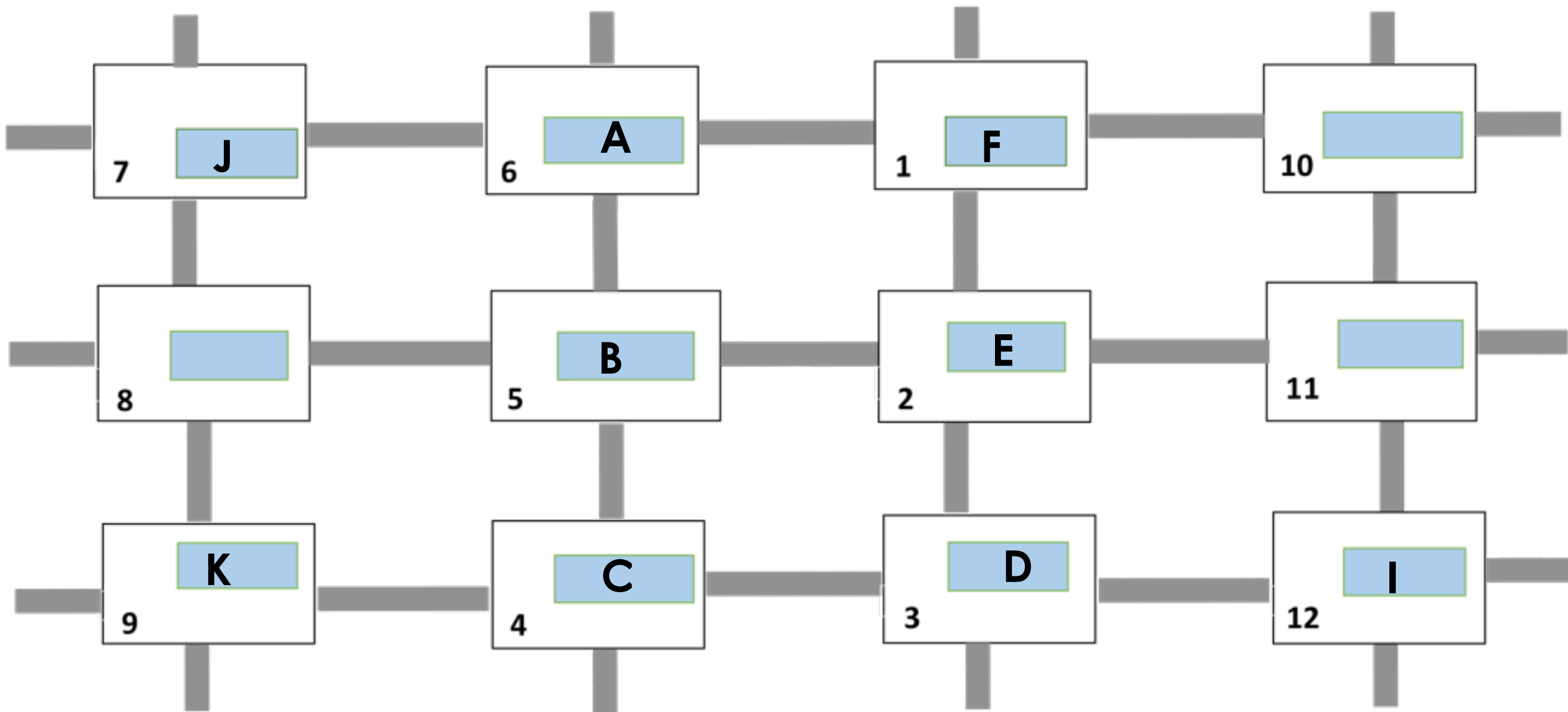
Georgia Tech
(tushar@ece.gatech.edu)

Paul V. Gratz

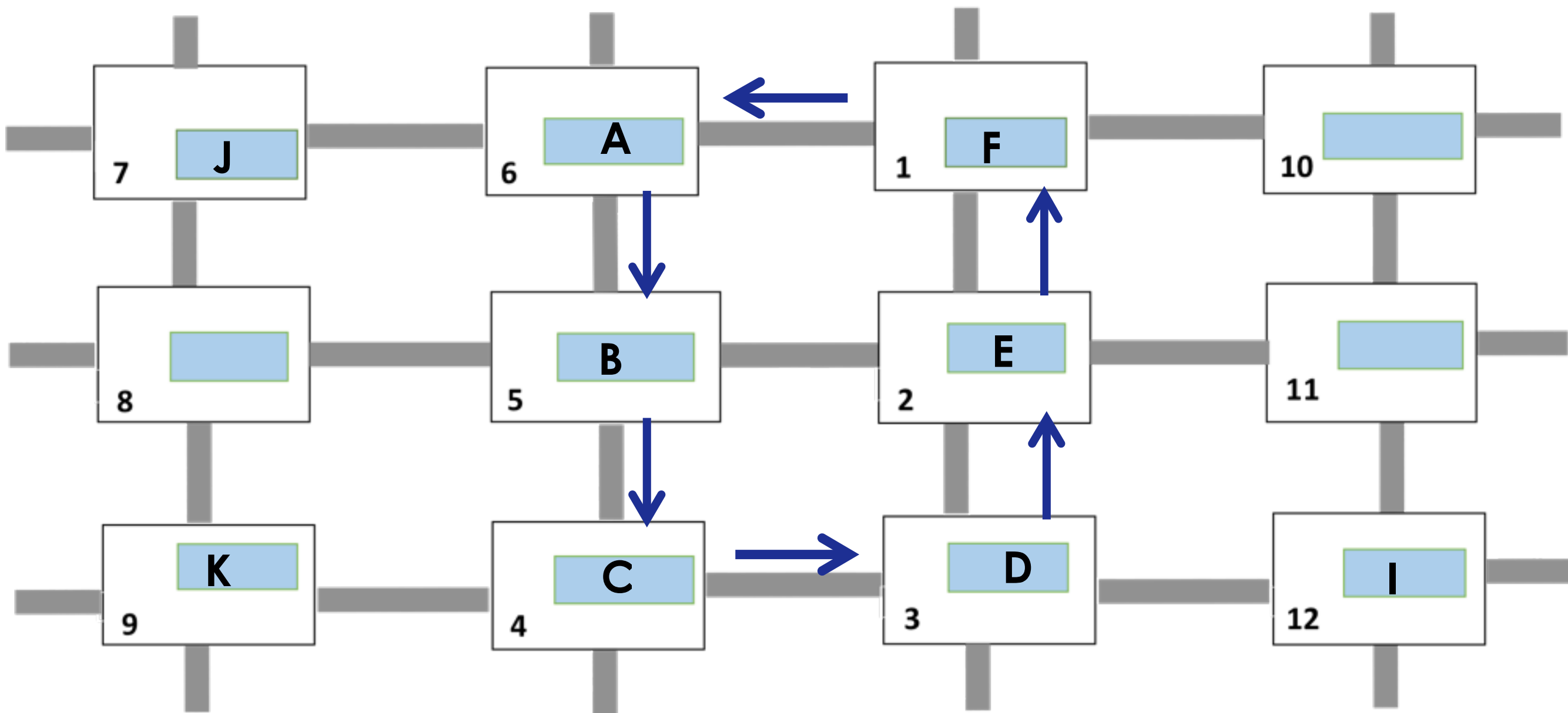
Texas A&M University
(pgratz@tamu.edu)

Network Routing

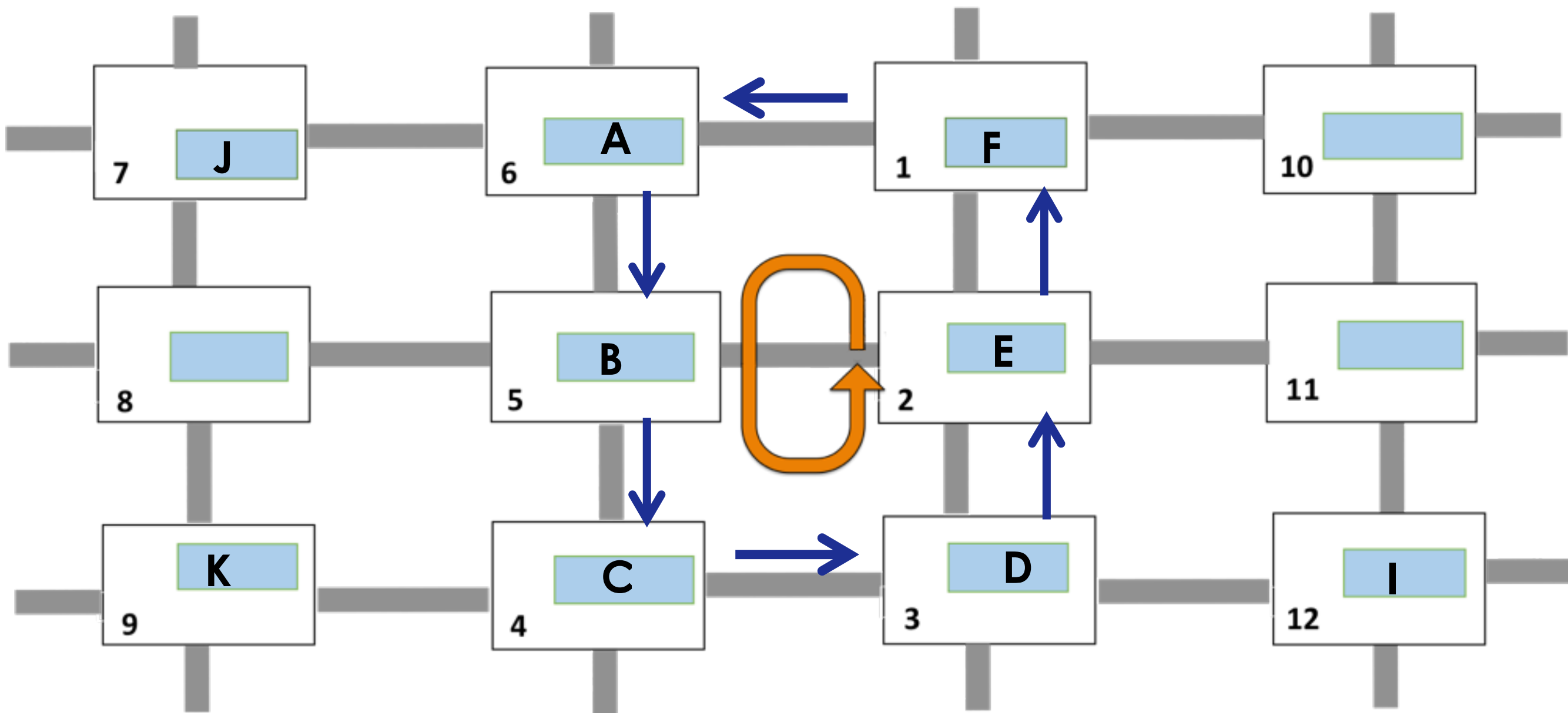
Network Routing



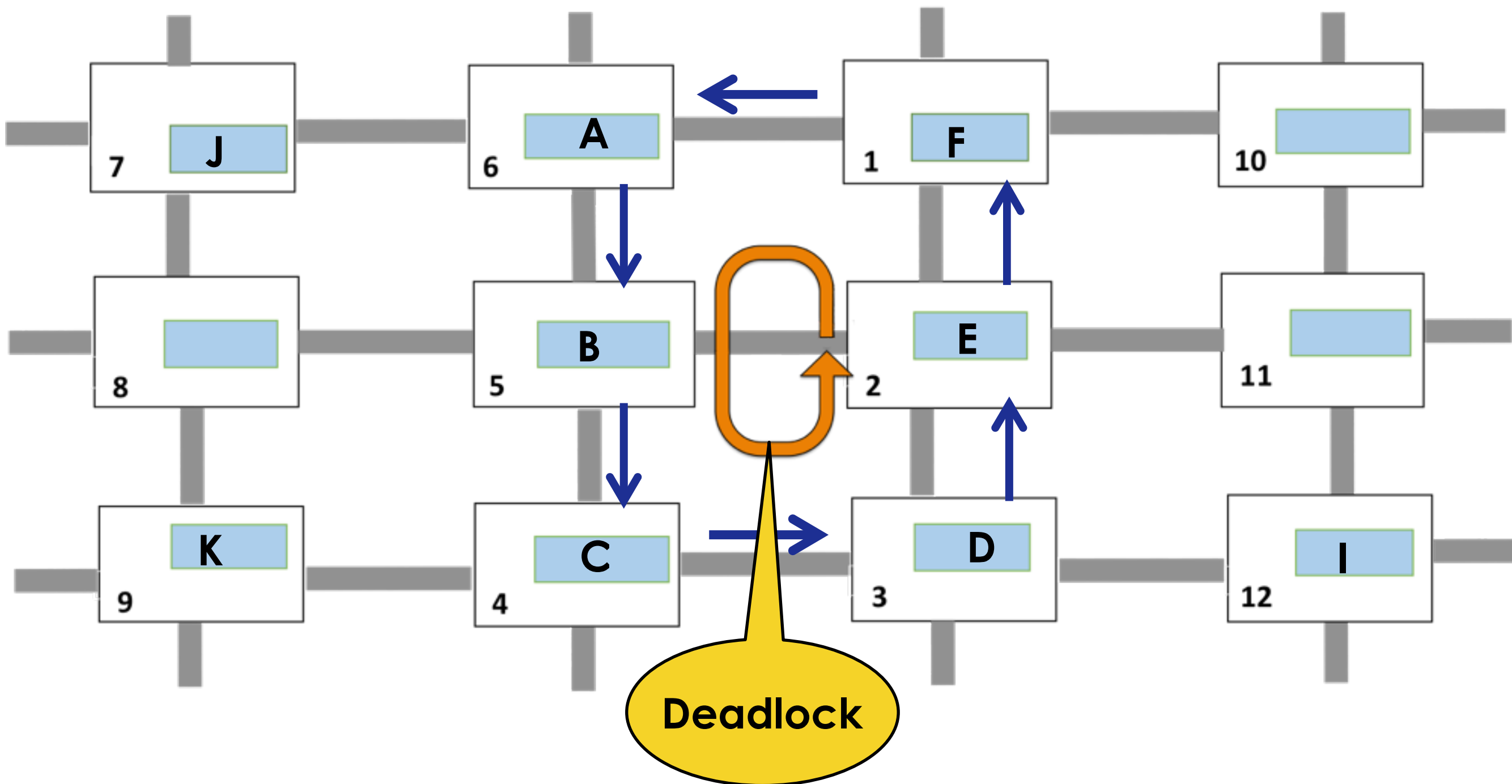
Network Routing



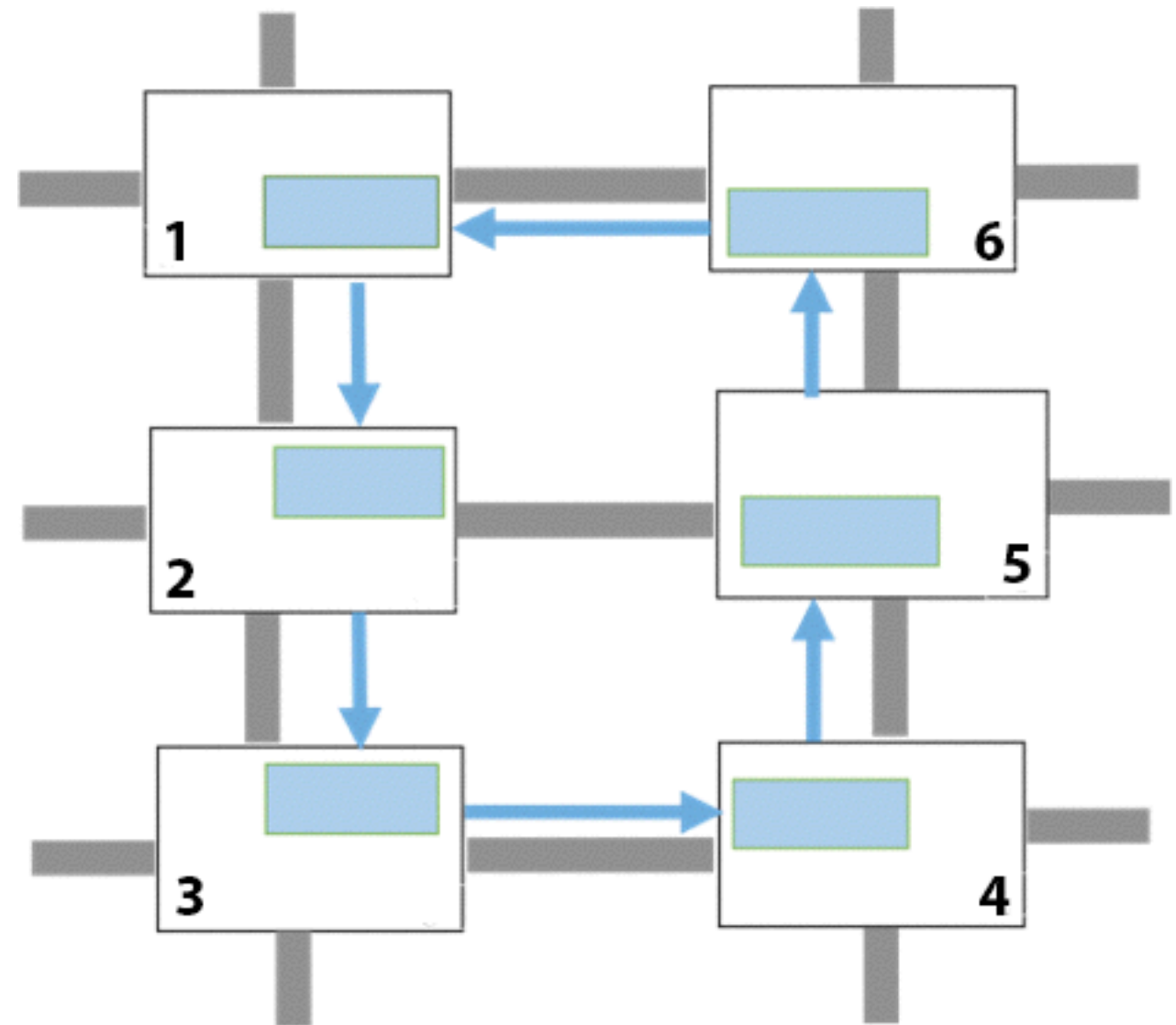
Network Routing



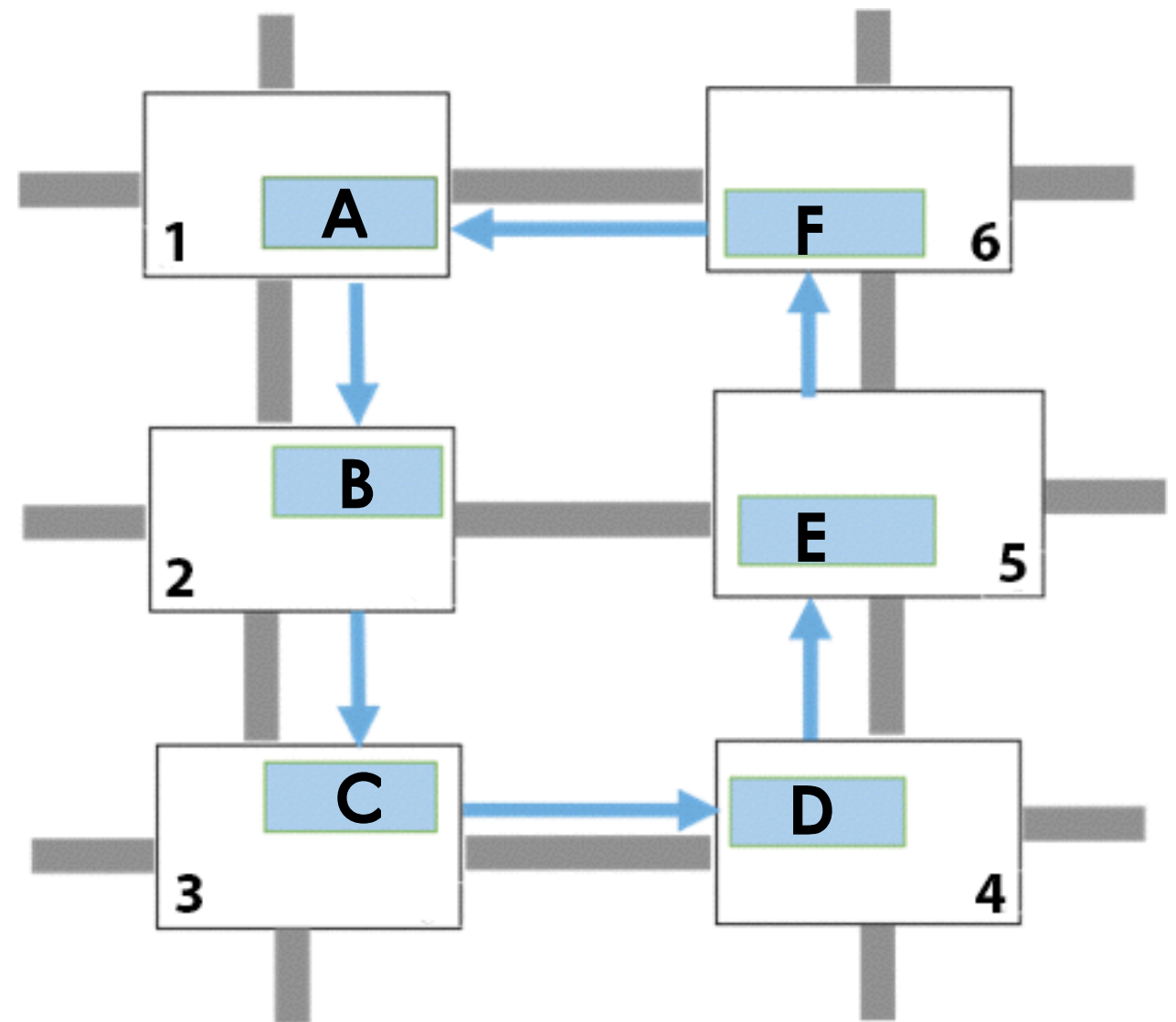
Network Routing



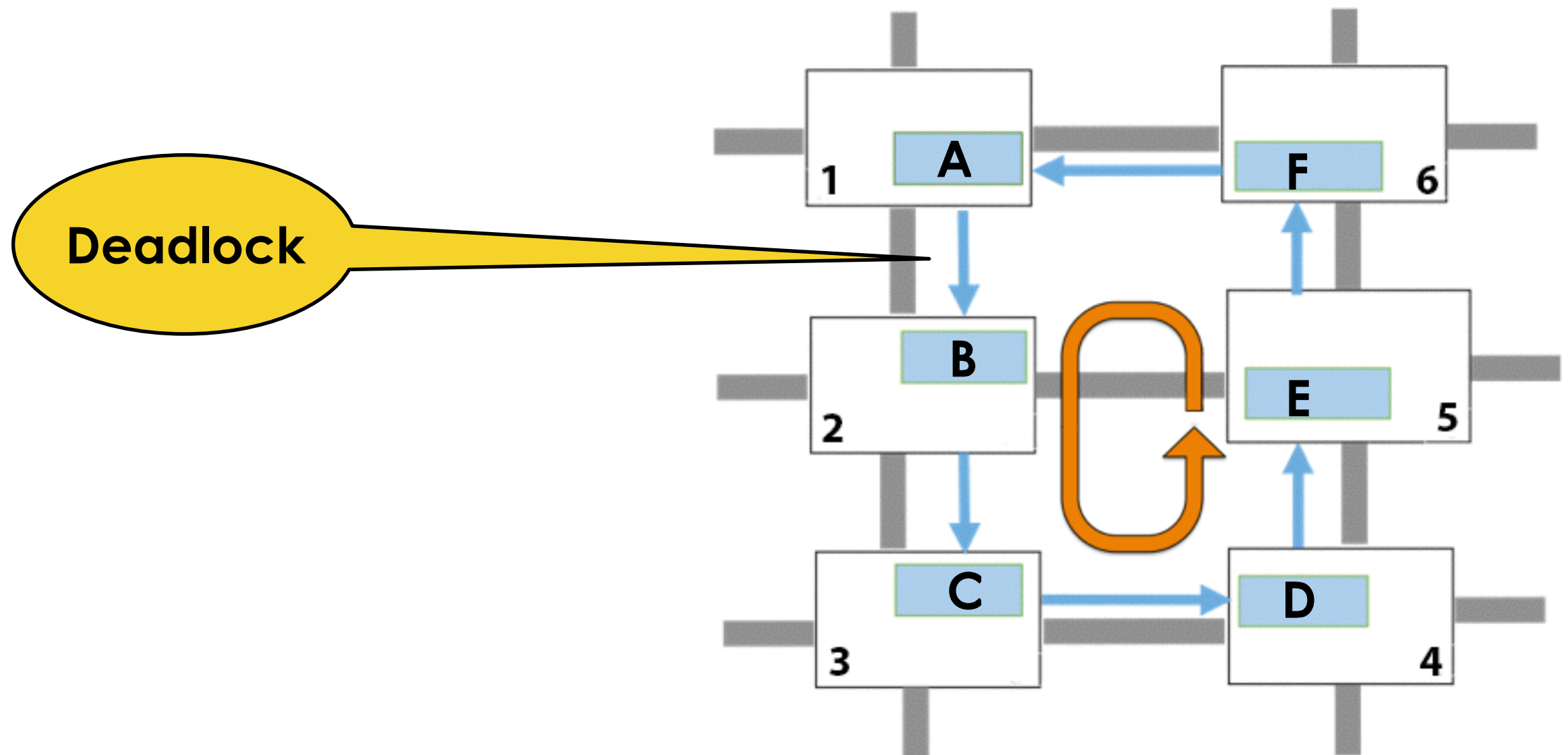
Routing Deadlocks



Routing Deadlocks

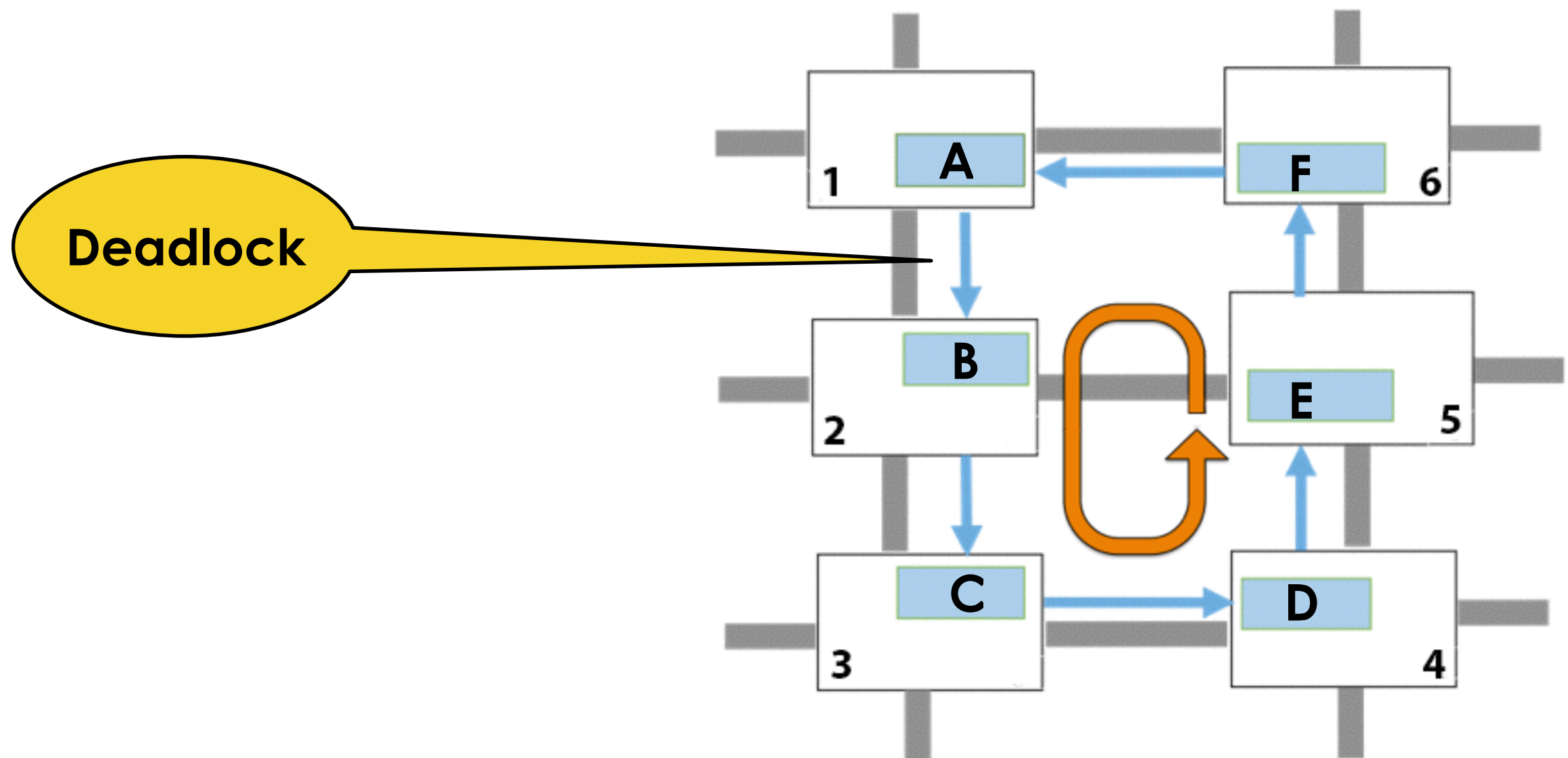


Routing Deadlocks



Routing Deadlocks

- A Routing Deadlock is a *cyclic buffer dependency chain* that renders forward progress impossible.



Routing Deadlocks

- A Routing Deadlock is a *cyclic buffer dependency chain* that renders forward progress impossible.

Routing Deadlocks

- A Routing Deadlock is a *cyclic buffer dependency chain* that renders forward progress impossible.
- Deadlocks are a *fundamental problem* in both off-chip and on-chip interconnection networks.

Routing Deadlocks

- A Routing Deadlock is a **cyclic buffer dependency chain** that renders forward progress impossible.
- Deadlocks are a **fundamental problem** in both off-chip and on-chip interconnection networks.
 - Cause **system breakdown** and kill chips.

Routing Deadlocks

- A Routing Deadlock is a **cyclic buffer dependency chain** that renders forward progress impossible.
- Deadlocks are a **fundamental problem** in both off-chip and on-chip interconnection networks.
 - Cause **system breakdown** and kill chips.
- Deadlocks are **hard to detect**.

Routing Deadlocks

- A Routing Deadlock is a **cyclic buffer dependency chain** that renders forward progress impossible.
- Deadlocks are a **fundamental problem** in both off-chip and on-chip interconnection networks.
 - Cause **system breakdown** and kill chips.
- Deadlocks are **hard to detect**.
 - Manifest after a long use time.

Routing Deadlocks

- A Routing Deadlock is a **cyclic buffer dependency chain** that renders forward progress impossible.
- Deadlocks are a **fundamental problem** in both off-chip and on-chip interconnection networks.
 - Cause **system breakdown** and kill chips.
- Deadlocks are **hard to detect**.
 - Manifest after a long use time.
 - Show up due to **system wear out faults** and **power-gating** of network elements which are hard to simulate.

Routing Deadlocks

- A Routing Deadlock is a **cyclic buffer dependency chain** that renders forward progress impossible.
- Deadlocks are a **fundamental problem** in both off-chip and on-chip interconnection networks.
 - Cause **system breakdown** and kill chips.
- Deadlocks are **hard to detect**.
 - Manifest after a long use time.
 - Show up due to **system wear out faults** and **power-gating** of network elements which are hard to simulate.
- Need a solution for **functional correctness** !!

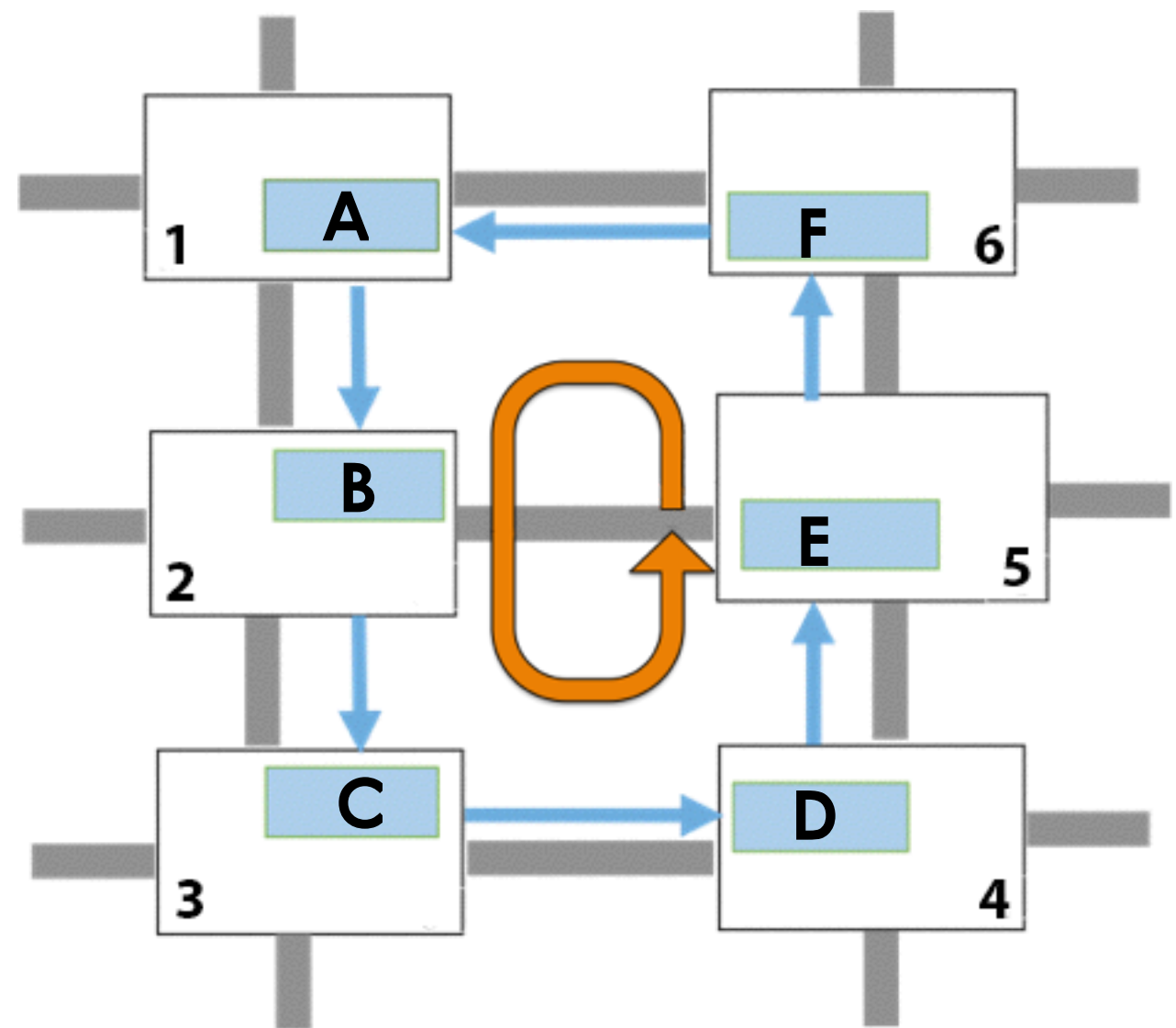
Solution I: Dally's Theory

Solution I: Dally's Theory

- Defines a *strict order* in acquisition of links and/or buffers which ensures a cyclic dependency is never created.

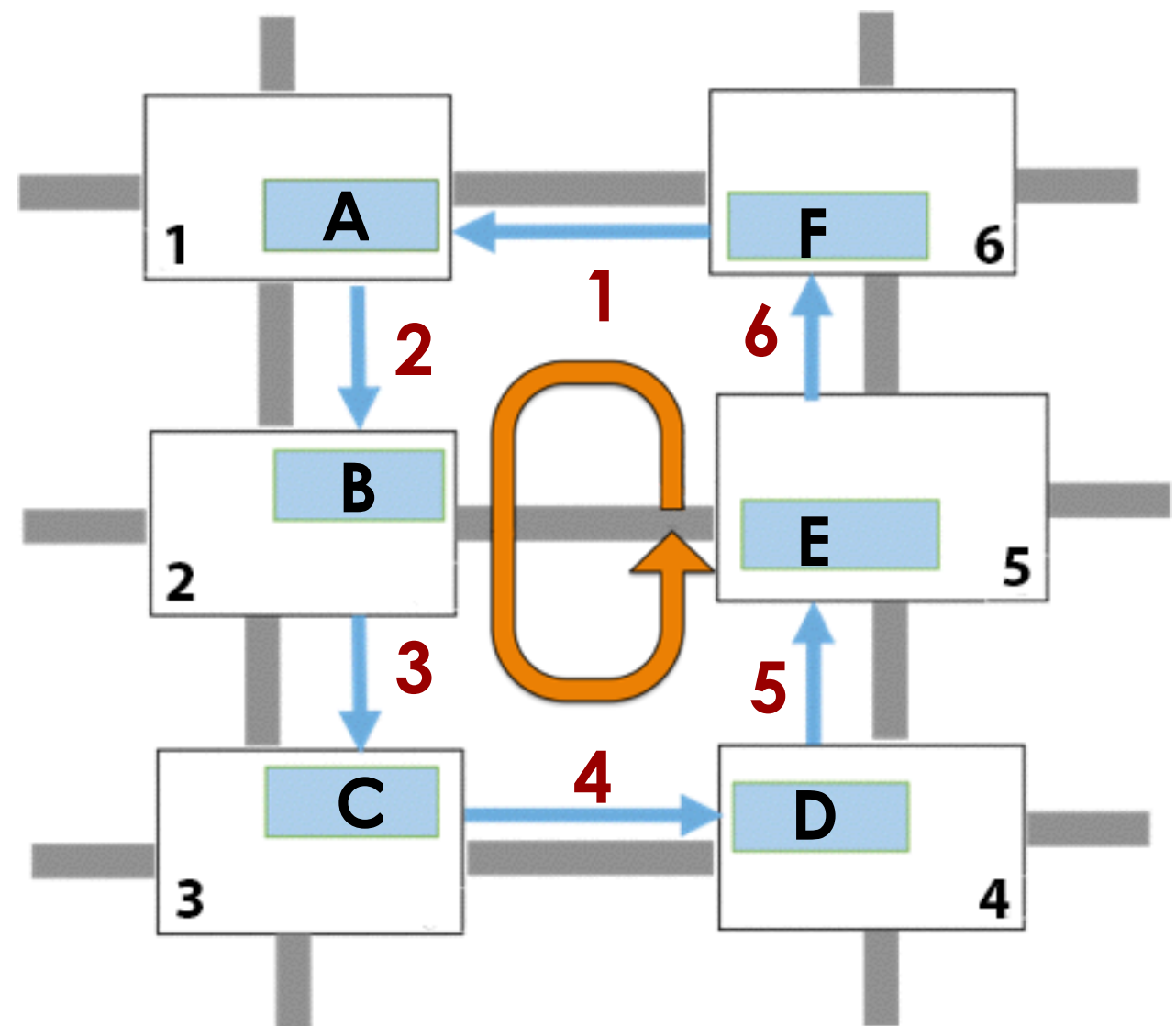
Solution I: Dally's Theory

- Defines a *strict order* in acquisition of links and/or buffers which ensures a cyclic dependency is never created.



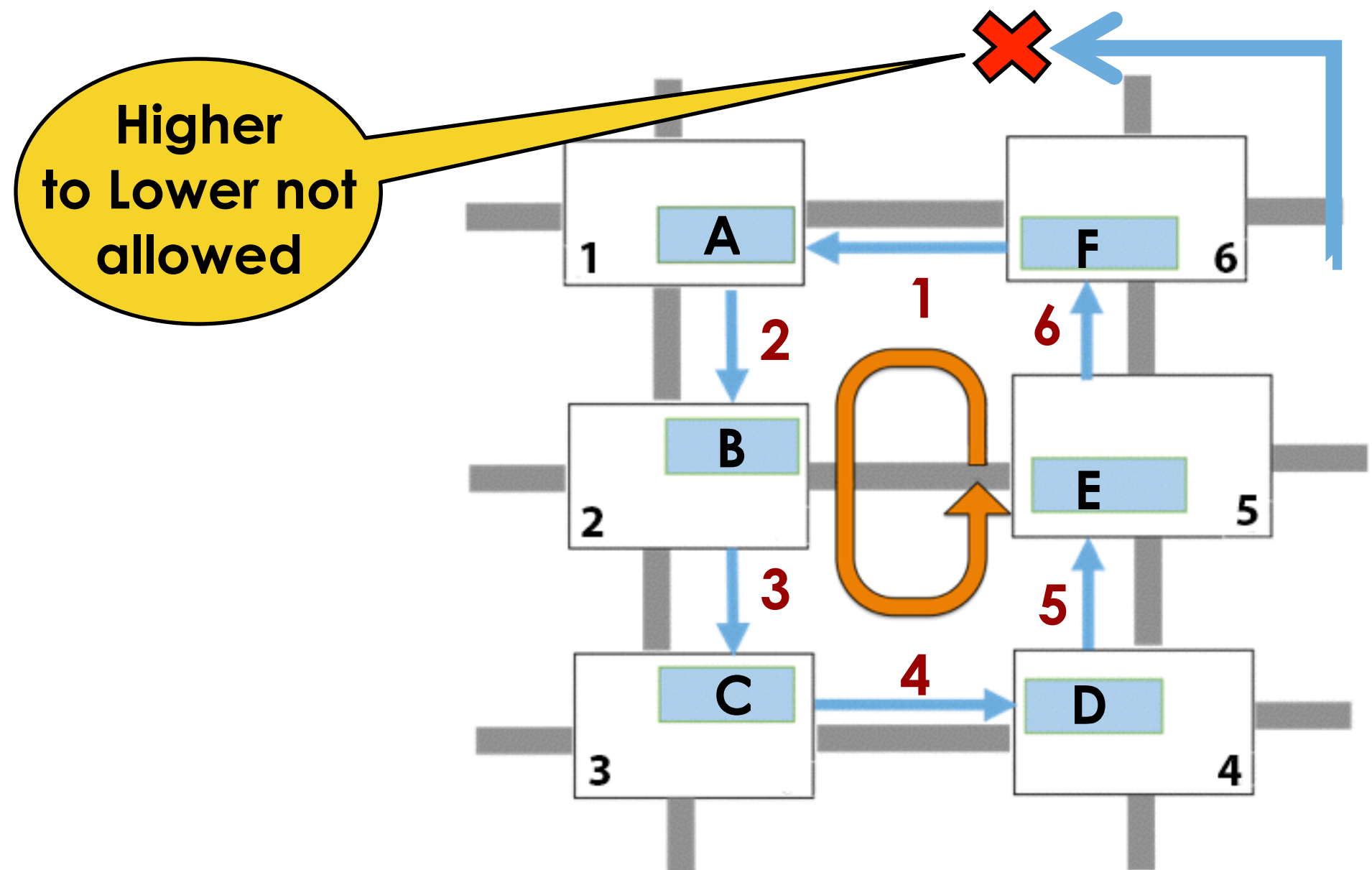
Solution I: Dally's Theory

- Defines a *strict order* in acquisition of links and/or buffers which ensures a cyclic dependency is never created.



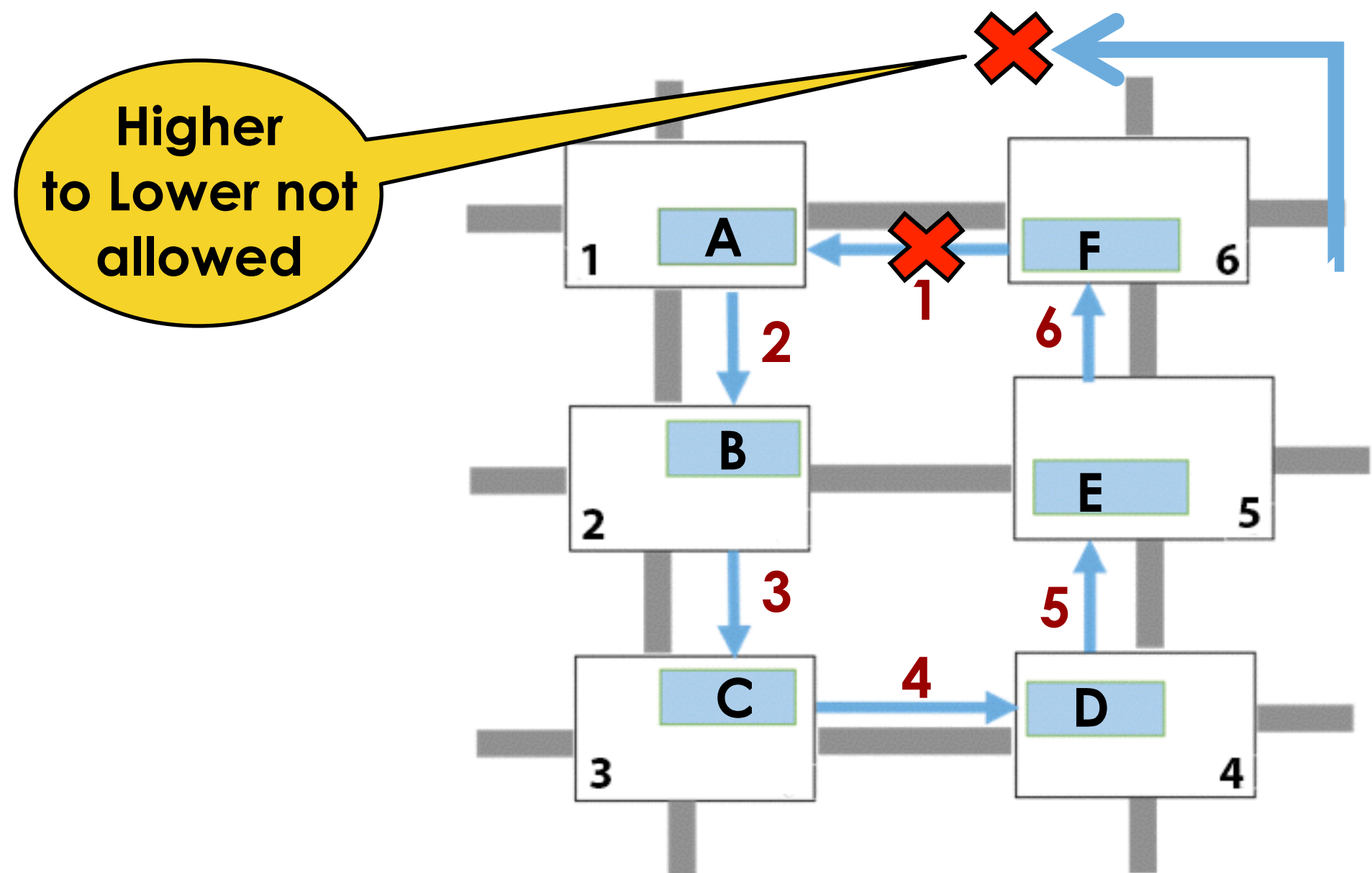
Solution I: Dally's Theory

- Defines a **strict order** in acquisition of links and/or buffers which ensures a cyclic dependency is never created.



Solution I: Dally's Theory

- Defines a **strict order** in acquisition of links and/or buffers which ensures a cyclic dependency is never created.



Solution I: Dally's Theory

- Defines a *strict order* in acquisition of links and/or buffers which ensures a cyclic dependency is never created.

Solution I: Dally's Theory

- Defines a ***strict order*** in acquisition of links and/or buffers which ensures a cyclic dependency is never created.
- ***Implementations:*** Turn model [5], XY routing, Up-Down routing [20].

Solution I: Dally's Theory

- Defines a **strict order** in acquisition of links and/or buffers which ensures a cyclic dependency is never created.
- **Implementations:** Turn model [5], XY routing, Up-Down routing [20].
- Limitations:
 - **Routing Restrictions:** Increased Latency, Throughput loss, Energy overhead
 - Require large no. of VCs for fully adaptive routing.

Solution II: Duato's Theory

Solution II: Duato's Theory

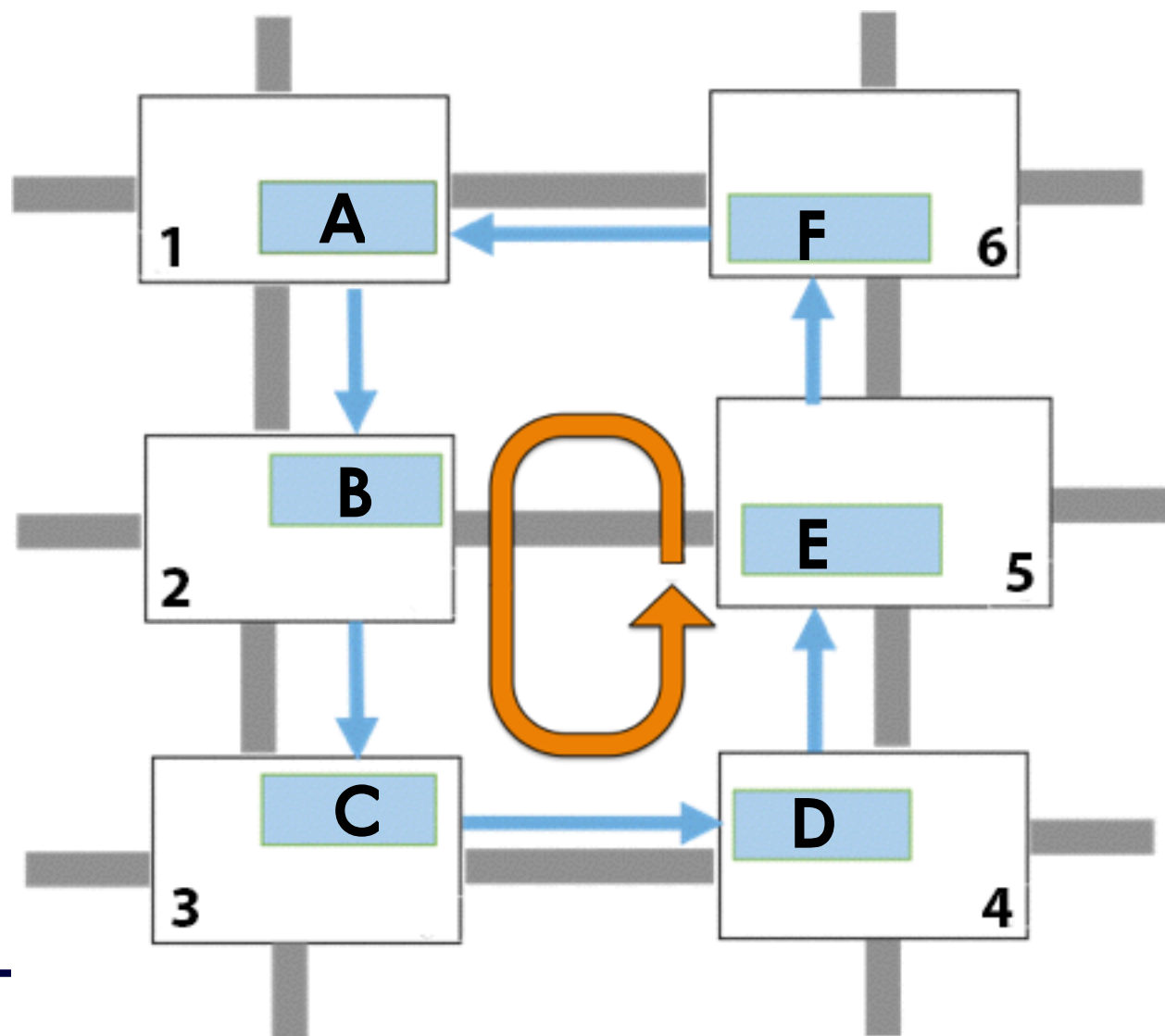
- *Adds buffers* to create a *deadlock free escape path* that can be used to avoid/recover from deadlocks.

Solution II: Duato's Theory

- *Adds buffers* to create a *deadlock free escape path* that can be used to avoid/recover from deadlocks.
- *Implementation:* turn restrictions in escape-VC.

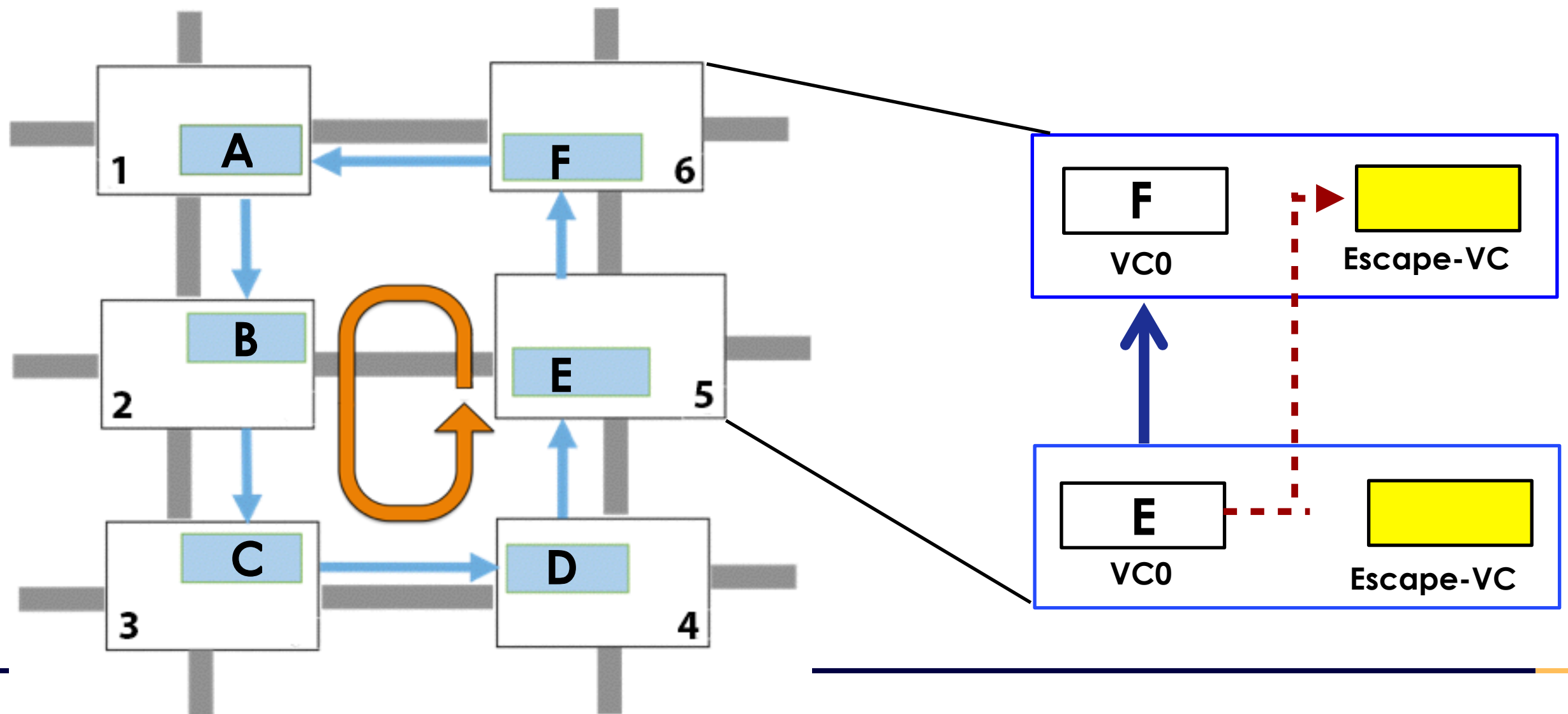
Solution II: Duato's Theory

- *Adds buffers* to create a *deadlock free escape path* that can be used to avoid/recover from deadlocks.
- *Implementation:* turn restrictions in escape-VC.



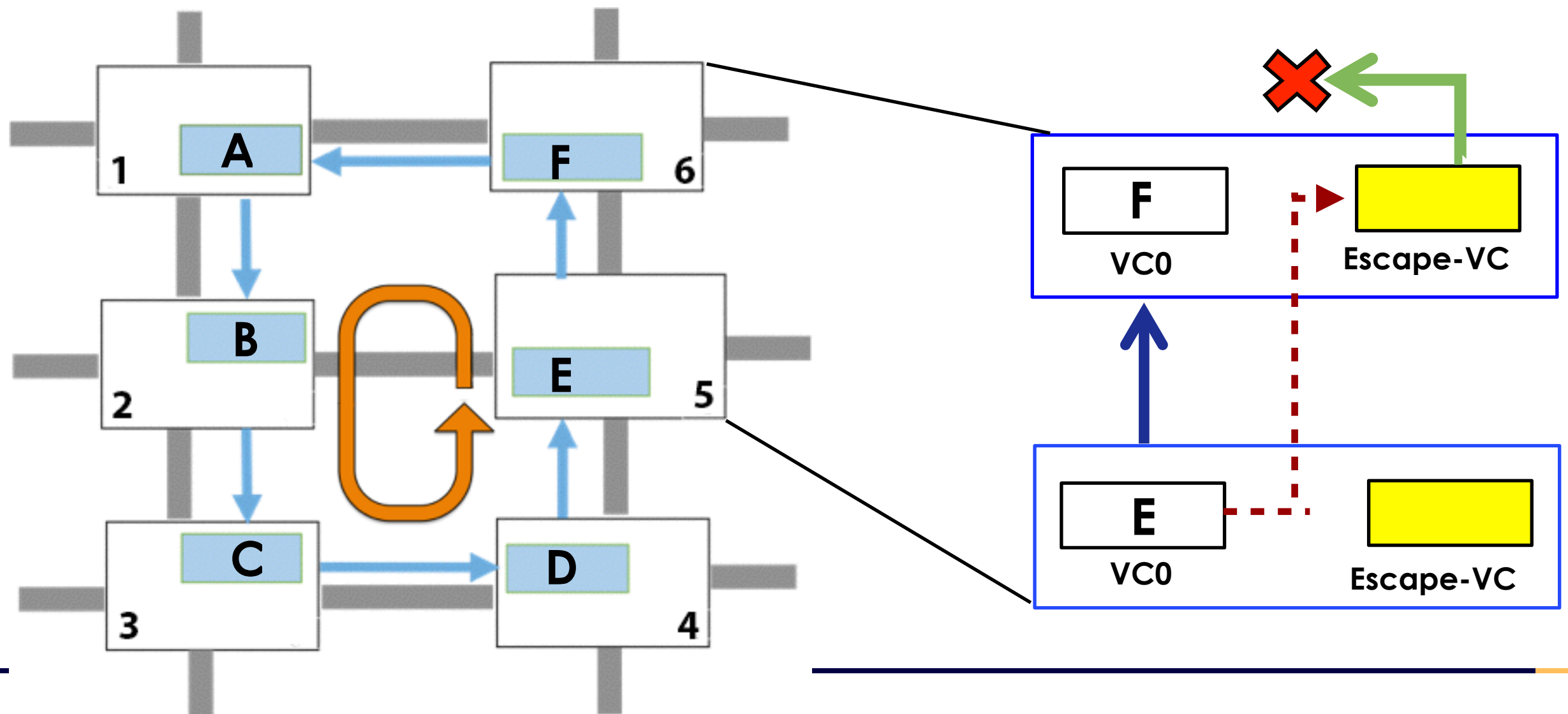
Solution II: Duato's Theory

- *Adds buffers* to create a *deadlock free escape path* that can be used to avoid/recover from deadlocks.
- *Implementation:* turn restrictions in escape-VC.



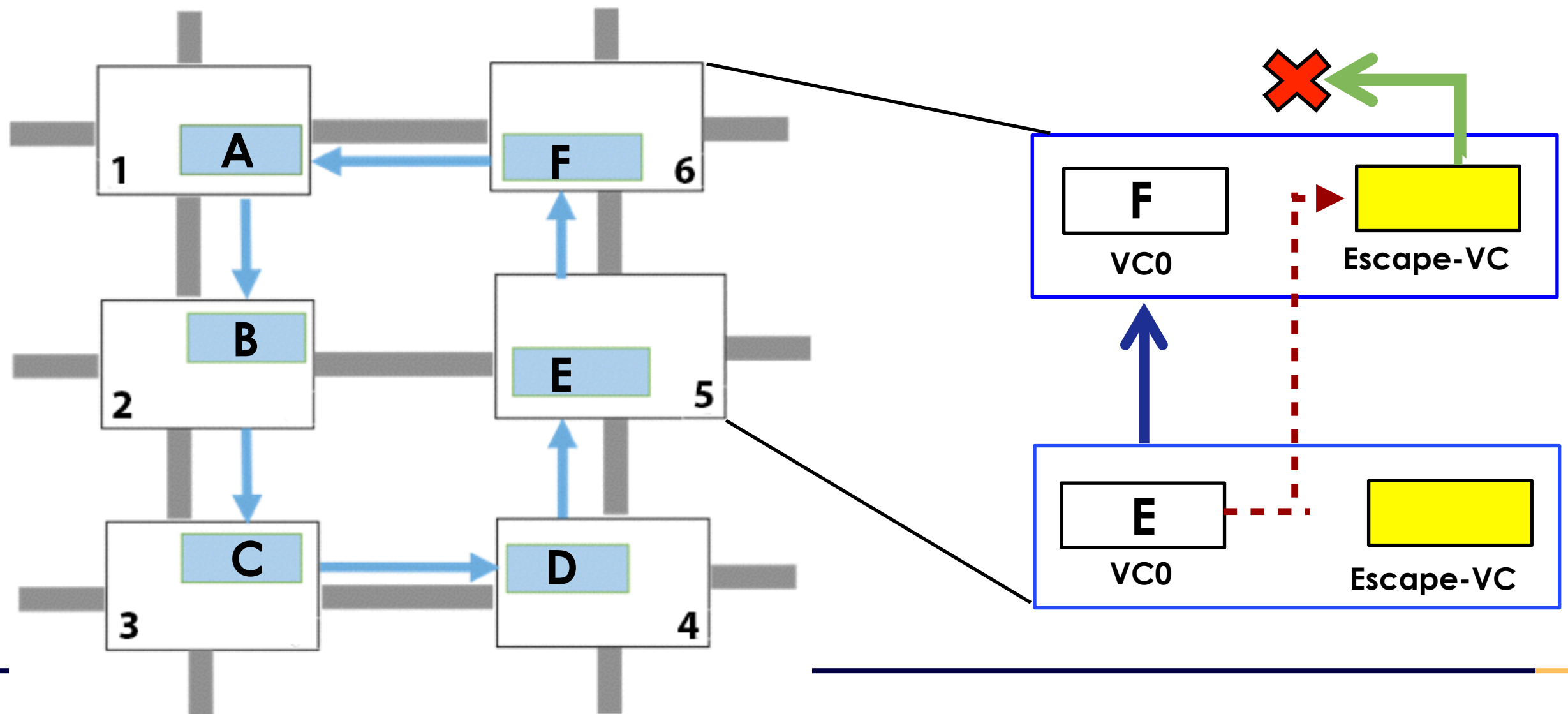
Solution II: Duato's Theory

- *Adds buffers* to create a *deadlock free escape path* that can be used to avoid/recover from deadlocks.
- *Implementation:* turn restrictions in escape-VC.



Solution II: Duato's Theory

- **Adds buffers** to create a **deadlock free escape path** that can be used to avoid/recover from deadlocks.
- **Implementation:** turn restrictions in escape-VC.



Solution II: Duato's Theory

- *Adds buffers* to create a *deadlock free escape path* that can be used to avoid/recover from deadlocks.
- *Implementation:* turn restrictions in escape-VC.

Solution II: Duato's Theory

- *Adds buffers* to create a *deadlock free escape path* that can be used to avoid/recover from deadlocks.
- *Implementation:* turn restrictions in escape-VC.
- Limitations:
 - *Energy* and *Area overhead* of escape VCs.
 - Additional routing tables/logic for routing within escape-VC.

Other Solutions

Other Solutions

- *Solution III: Flow Control*

Other Solutions

- *Solution III: Flow Control*
 - *Restrict injection* when no. of empty buffers fall below a threshold

Other Solutions

- *Solution III: Flow Control*
 - *Restrict injection* when no. of empty buffers fall below a threshold
 - *Implementation:* Bubble Flow Control [9]

Other Solutions

- *Solution III: Flow Control*
 - ***Restrict injection*** when no. of empty buffers fall below a threshold
 - ***Implementation:*** Bubble Flow Control [9]
 - Limitation: Implementation Complexity.

Other Solutions

- *Solution III: Flow Control*
 - *Restrict injection* when no. of empty buffers fall below a threshold
 - *Implementation:* Bubble Flow Control [9]
 - Limitation: Implementation Complexity.
- *Solution IV: Deflection Routing*

Other Solutions

■ Solution III: Flow Control

- **Restrict injection** when no. of empty buffers fall below a threshold
- **Implementation:** Bubble Flow Control [9]
- Limitation: Implementation Complexity.

■ Solution IV: Deflection Routing

- Assign every flit to some output port even if they get **misrouted**.

Other Solutions

■ Solution III: Flow Control

- **Restrict injection** when no. of empty buffers fall below a threshold
- **Implementation:** Bubble Flow Control [9]
- Limitation: Implementation Complexity.

■ Solution IV: Deflection Routing

- Assign every flit to some output port even if they get **misrouted**.
- **Implementation:** BLESS [10], CHIPPER [35]

Other Solutions

■ Solution III: Flow Control

- **Restrict injection** when no. of empty buffers fall below a threshold
- **Implementation:** Bubble Flow Control [9]
- Limitation: Implementation Complexity.

■ Solution IV: Deflection Routing

- Assign every flit to some output port even if they get **misrouted**.
- **Implementation:** BLESS [10], CHIPPER [35]
- Limitation: **Livelocks**

Comparison of Deadlock Freedom Theories

**Metric
Theory**

**Acyclic
CDG not
Required**

**No Packet
Injection
Restrictions**

**Livelock
Free**

**VC cost for Mesh
Routing
Minimal Adaptive**

**Topology
Indepen-
dent**

Comparison of Deadlock Freedom Theories

Metric Theory	Acyclic CDG not Required	No Packet Injection Restrictions	Livelock Free	VC cost for Mesh Routing		Topology Indepen- dent
				Minimal	Adaptive	
Dally	✗	✓	✓	1	6	✗

Comparison of Deadlock Freedom Theories

Metric Theory	Acyclic CDG not Required	No Packet Injection Restrictions	Livelock Free	VC cost for Mesh Routing		Topology Indepen- dent
				Minimal	Adaptive	
Dally	✗	✓	✓	1	6	✗
Duato	✓	✓	✓	1	2	✗

Comparison of Deadlock Freedom Theories

Metric Theory	Acyclic CDG not Required	No Packet Injection Restrictions	Livelock Free	VC cost for Mesh Routing		Topology Indepen- dent
				Minimal	Adaptive	
Dally	✗	✓	✓	1	6	✗
Duato	✓	✓	✓	1	2	✗
Flow Control	✓	✗	✓	2	2	✓

Comparison of Deadlock Freedom Theories

Metric Theory	Acyclic CDG not Required	No Packet Injection Restrictions	Livelock Free	VC cost for Mesh Routing		Topology Indepen- dent
				Minimal	Adaptive	
Dally	✗	✓	✓	1	6	✗
Duato	✓	✓	✓	1	2	✗
Flow Control	✓	✗	✓	2	2	✓
Deflection Routing	✓	✗	✗	✗	1	✓

Comparison of Deadlock Freedom Theories

Metric Theory	Acyclic CDG not Required	No Packet Injection Restrictions	Livelock Free	VC cost for Mesh Routing		Topology Indepen- dent
				Minimal	Adaptive	
Dally	✗	✓	✓	1	6	✗
Duato	✓	✓	✓	1	2	✗
Flow Control	Can we do <u>better</u> ??					✓
Deflection Routing	✓	✗	✗	✗	1	✓

Comparison of Deadlock Freedom Theories

Metric Theory	Acyclic CDG not Required	No Packet Injection Restrictions	Livelock Free	VC cost for Mesh Routing		Topology Indepen- dent
				Minimal	Adaptive	
Dally	✗	✓	✓	1	6	✗
Duato	✓	✓	✓	1	2	✗
Flow Control	✓	✗	✓	2	2	✓
Deflection Routing	✓	✗	✗	✗	1	✓

Comparison of Deadlock Freedom Theories

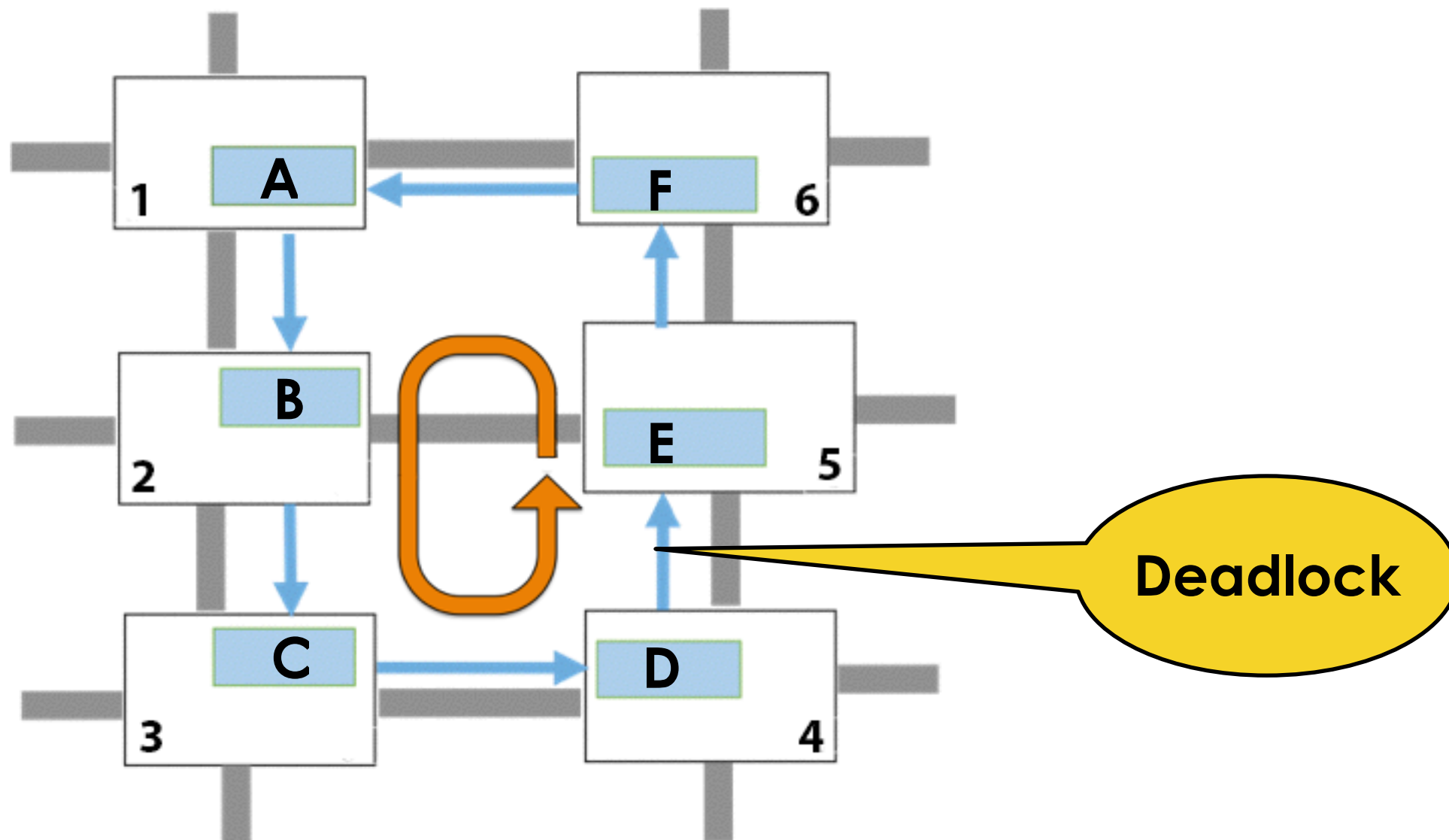
Metric Theory	Acyclic CDG not Required	No Packet Injection Restrictions	Livelock Free	VC cost for Mesh Routing		Topology Indepen- dent
				Minimal	Adaptive	
Dally	✗	✓	✓	1	6	✗
Duato	✓	✓	✓	1	2	✗
Flow Control	✓	✗	✓	2	2	✓
Deflection Routing	✓	✗	✗	✗	1	✓
SPIN	✓	✓	✓	1	1	✓

Outline

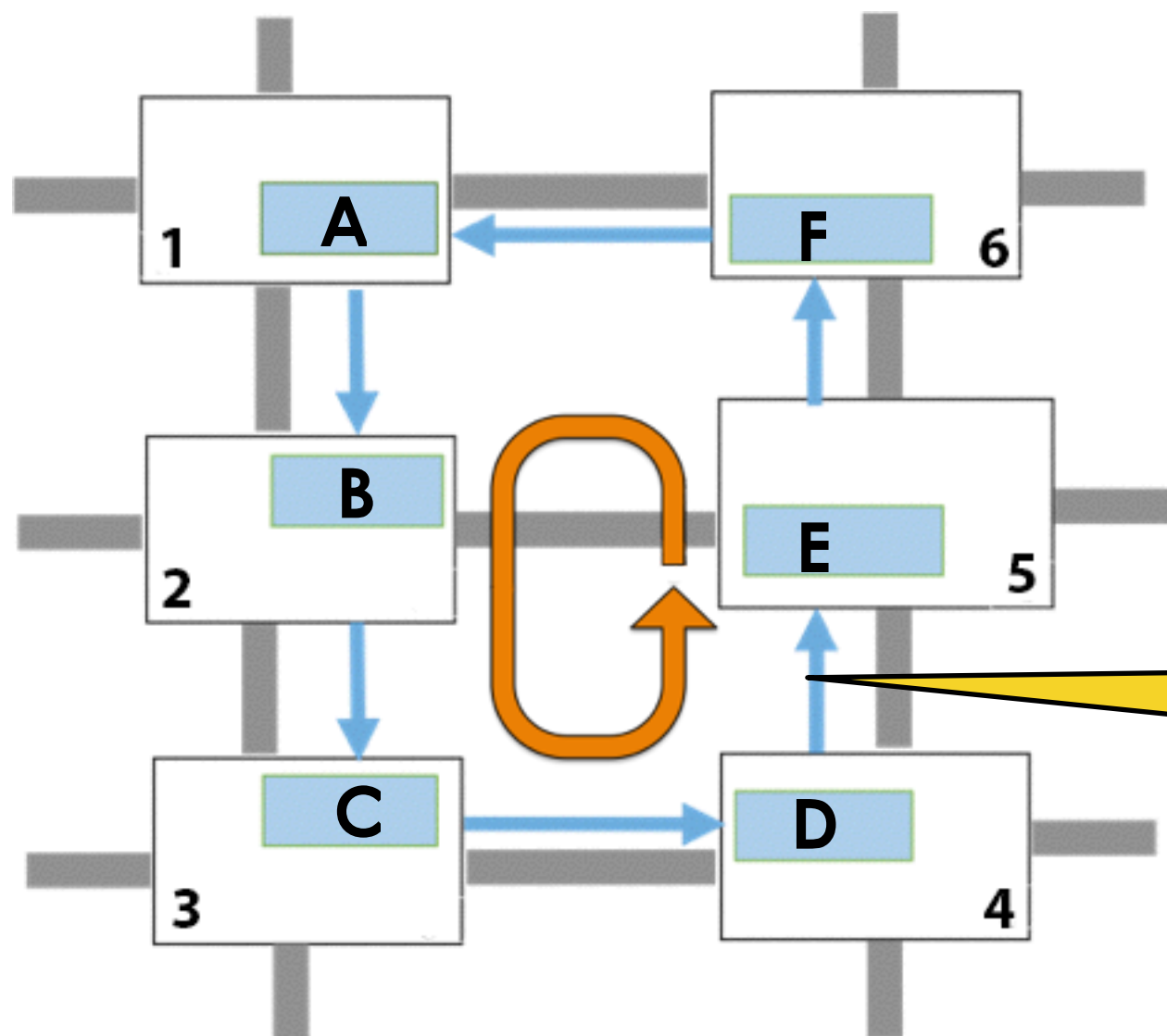
- Routing Deadlocks
 - State of the Art
 - Dally's Theory
 - Duato's Theory
 - Flow Control Routing
 - Deflection Routing
 - **SPIN** : **S**ynchronized **P**rogress in **I**nterconnection **N**etworks
 - Evaluations
 - Conclusion
-

SPIN : Key Idea

SPIN : Key Idea



SPIN : Key Idea

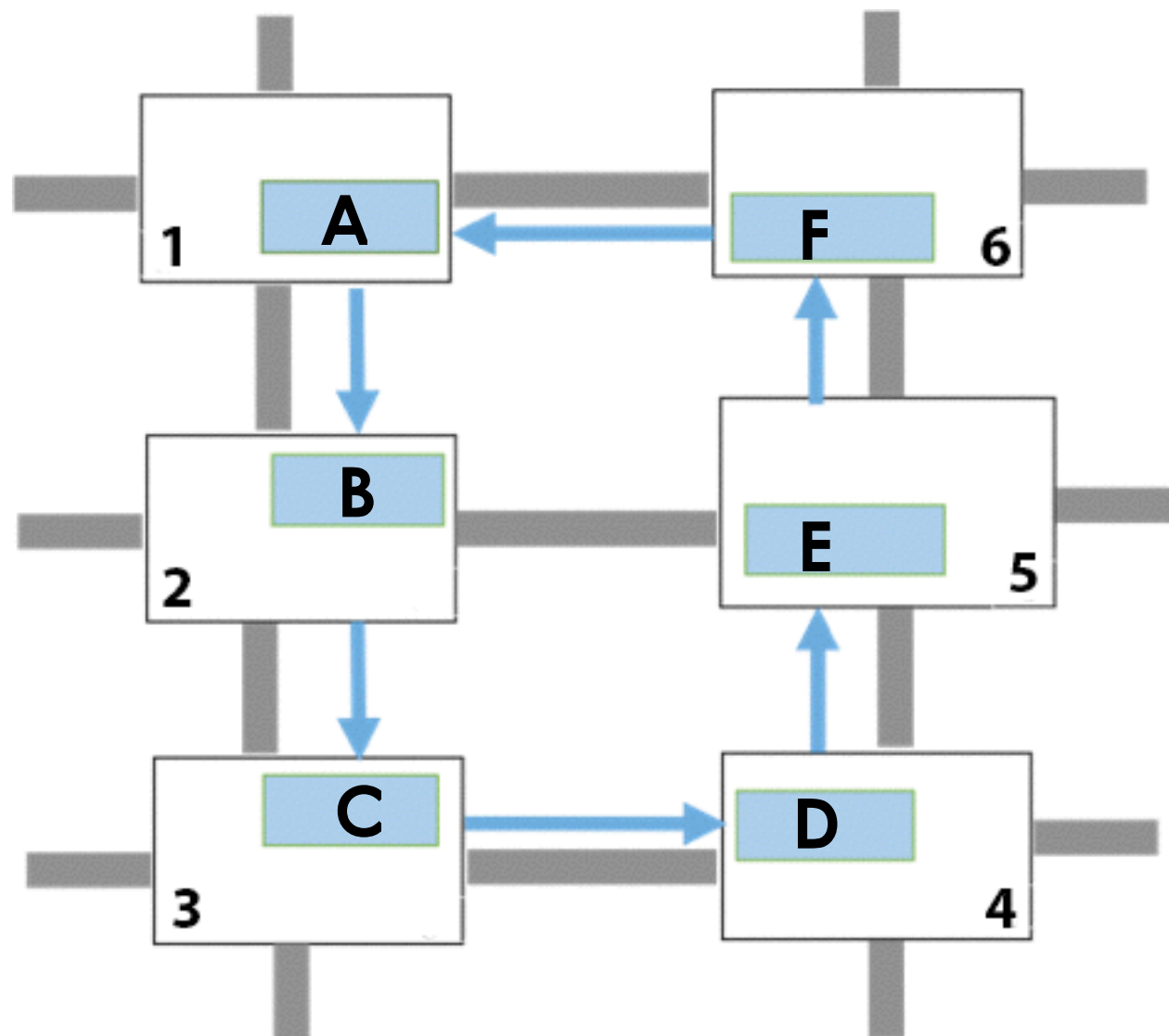


What if:

We **coordinate** the **movement** of **every packet** to the next hop at a given time ??

Deadlock

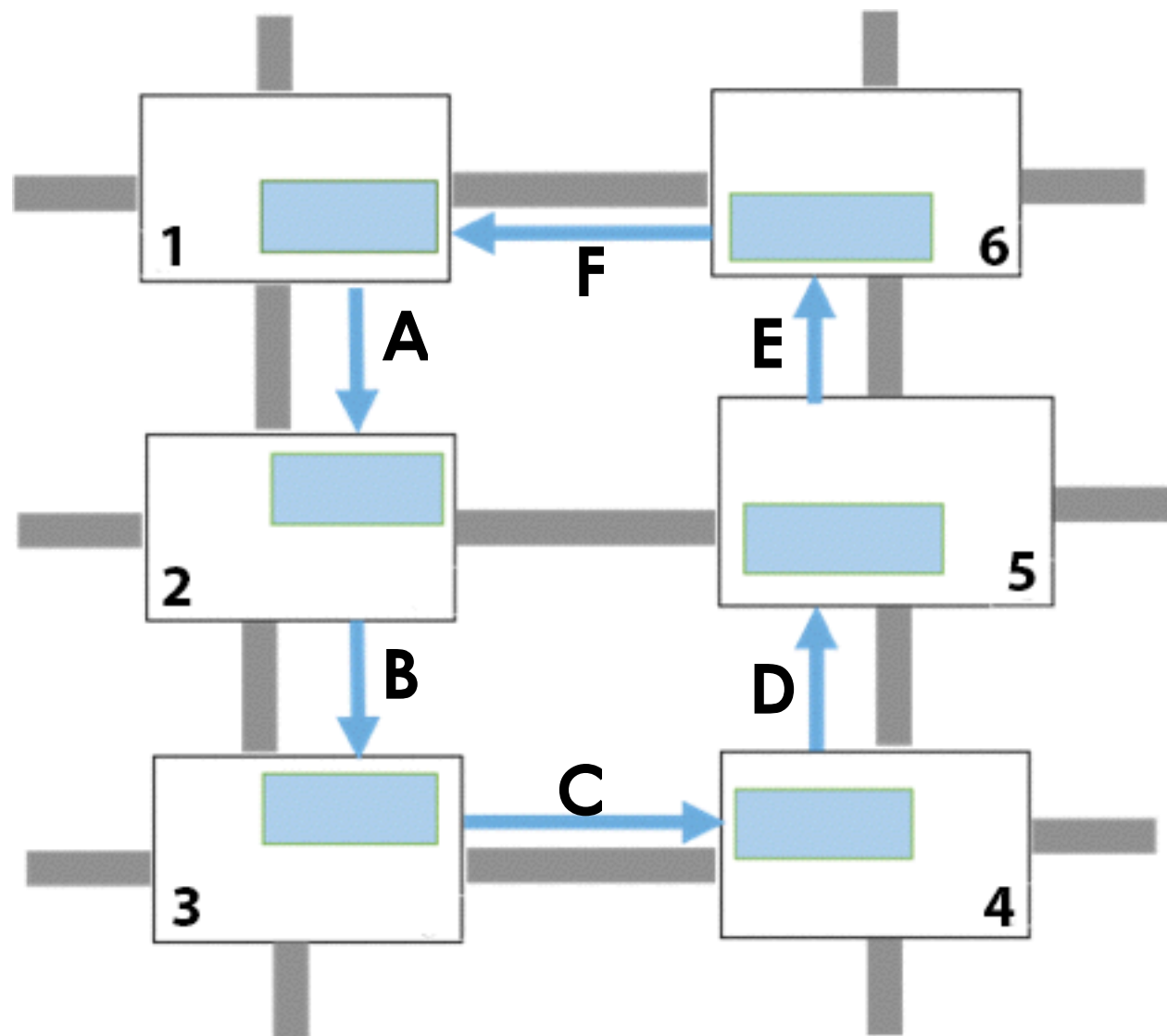
SPIN : Key Idea



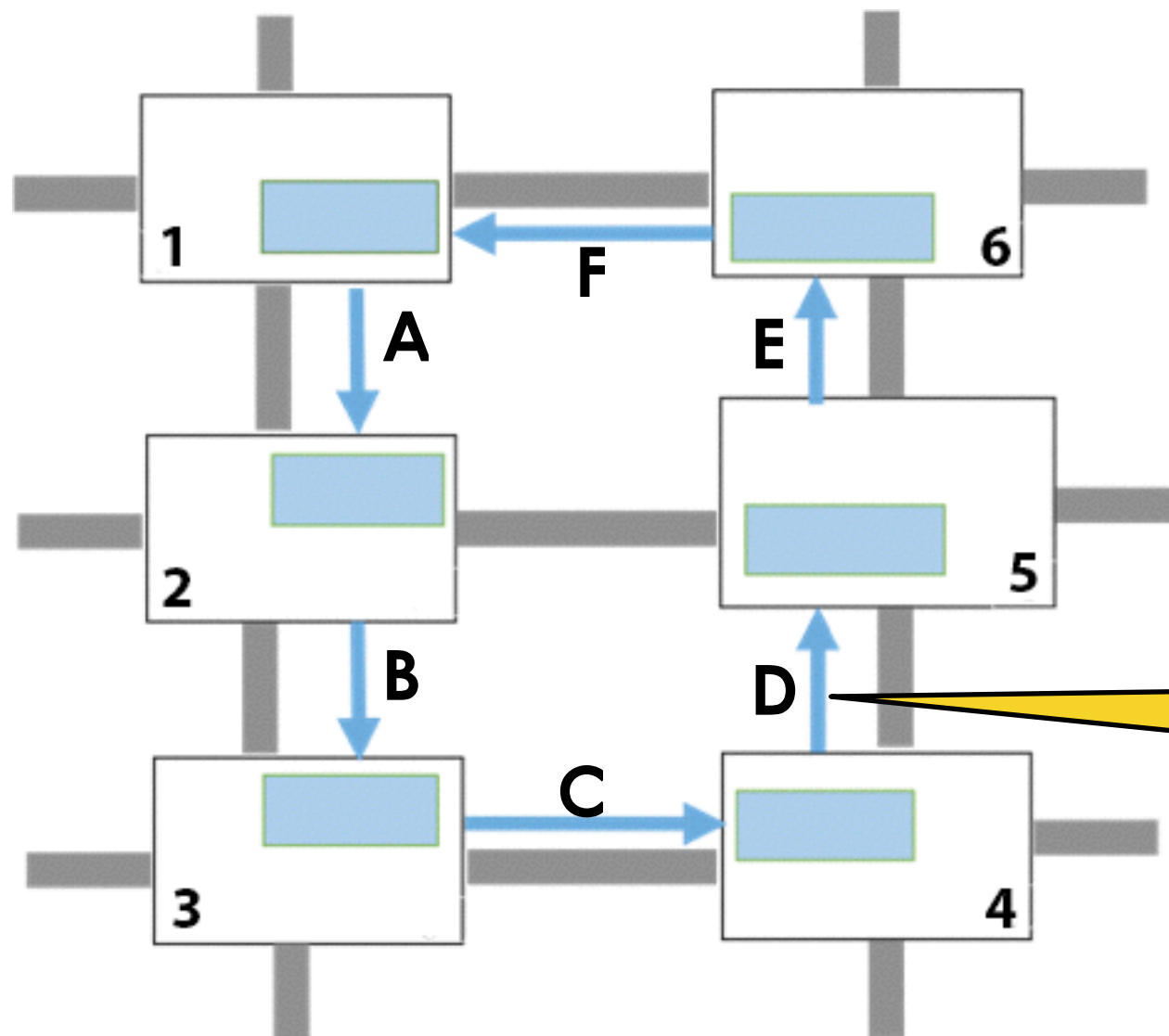
What if:

We **coordinate** the **movement** of **every packet** to the next hop at a given time ??

SPIN : Key Idea

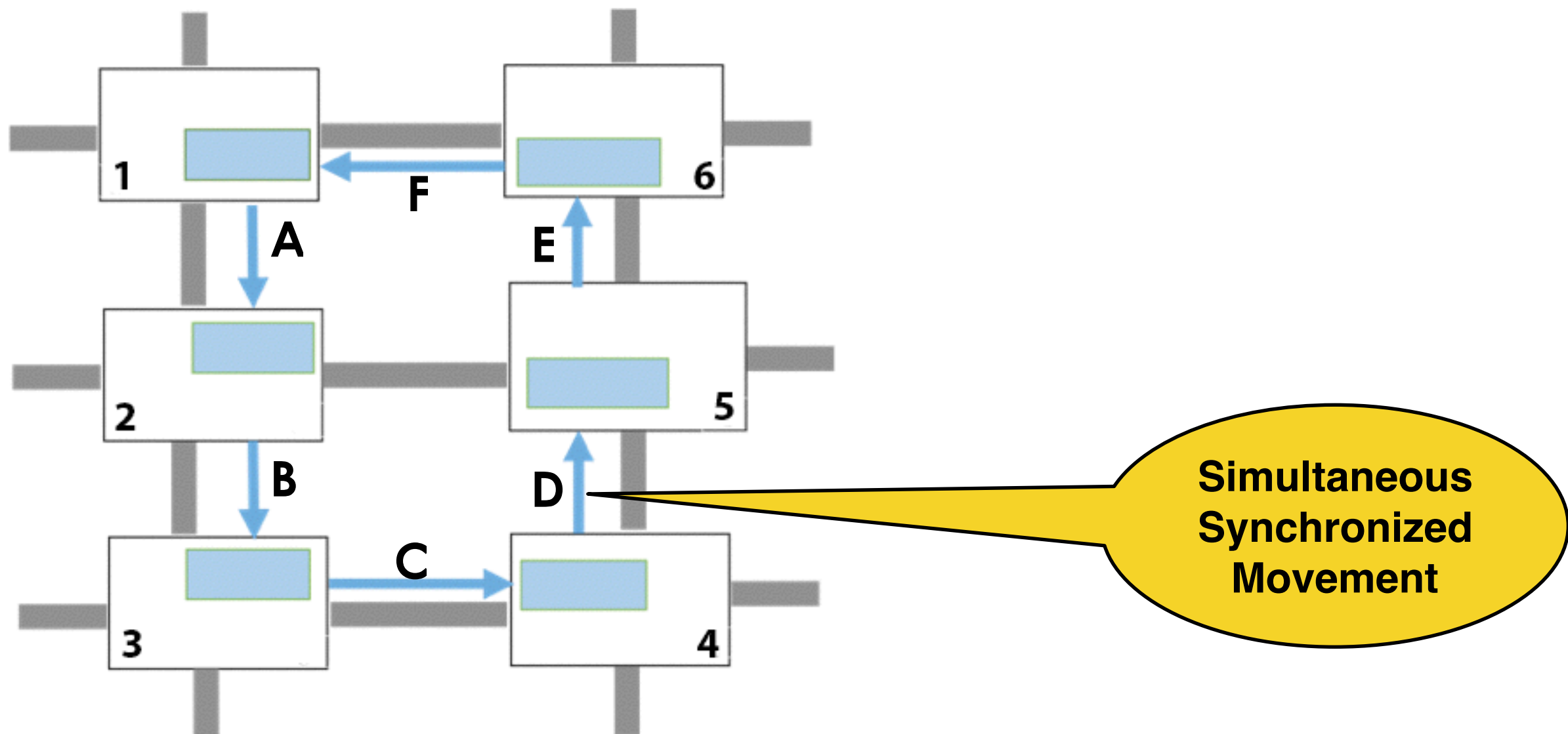


SPIN : Key Idea



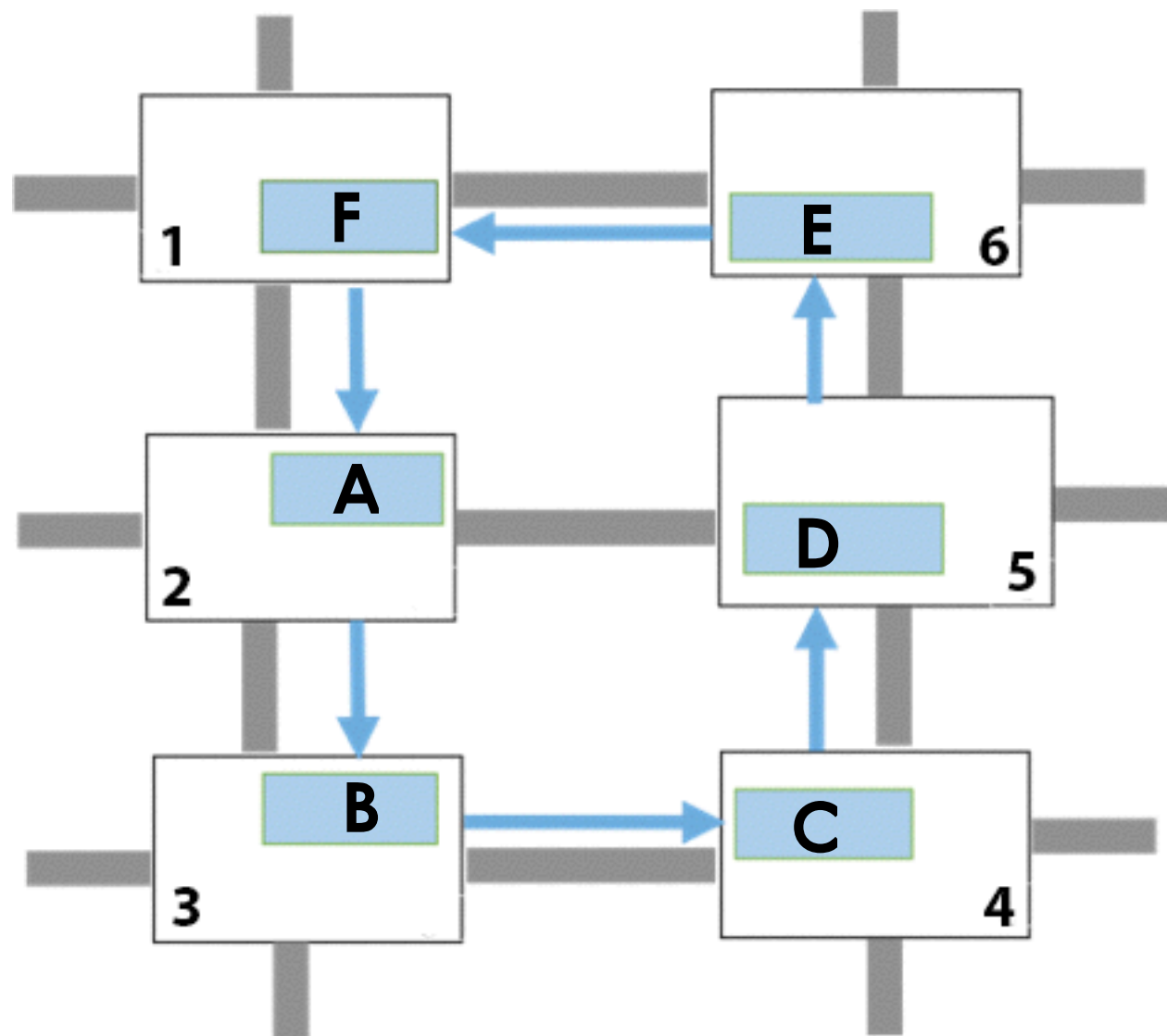
SPIN : Key Idea

- *Simultaneous Synchronized Movement* of all deadlocked packets in the loop is called a *spin*.



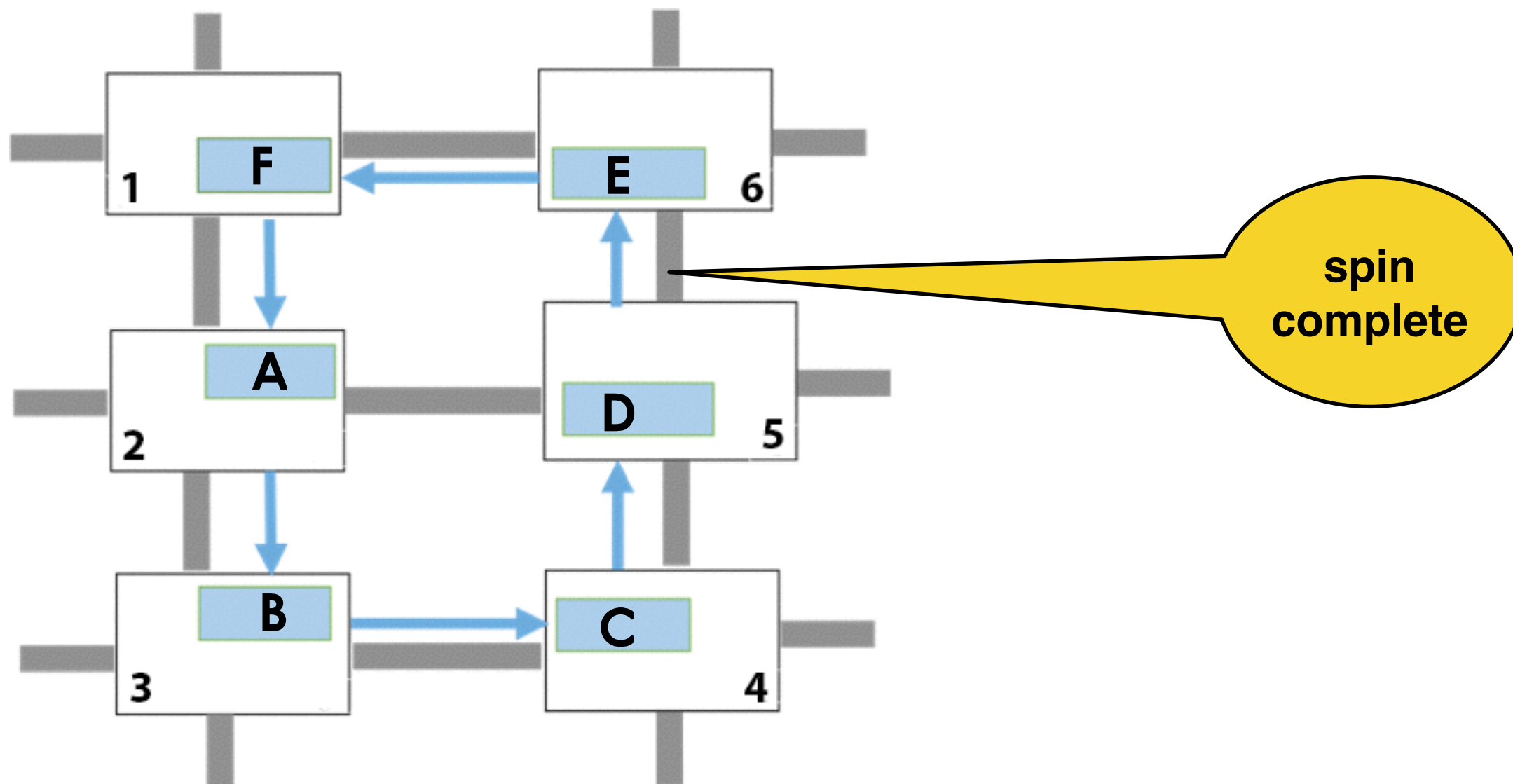
SPIN : Key Idea

- *Simultaneous Synchronized Movement* of all deadlocked packets in the loop is called a *spin*.



SPIN : Key Idea

- *Simultaneous Synchronized Movement* of all deadlocked packets in the loop is called a *spin*.



SPIN : Key Idea

- *Simultaneous Synchronized Movement* of all deadlocked packets in the loop is called a *spin*.

SPIN : Key Idea

- *Simultaneous Synchronized Movement* of all deadlocked packets in the loop is called a *spin*.
- Each spin leads to *one hop forward movement* of all deadlock packets.

SPIN : Key Idea

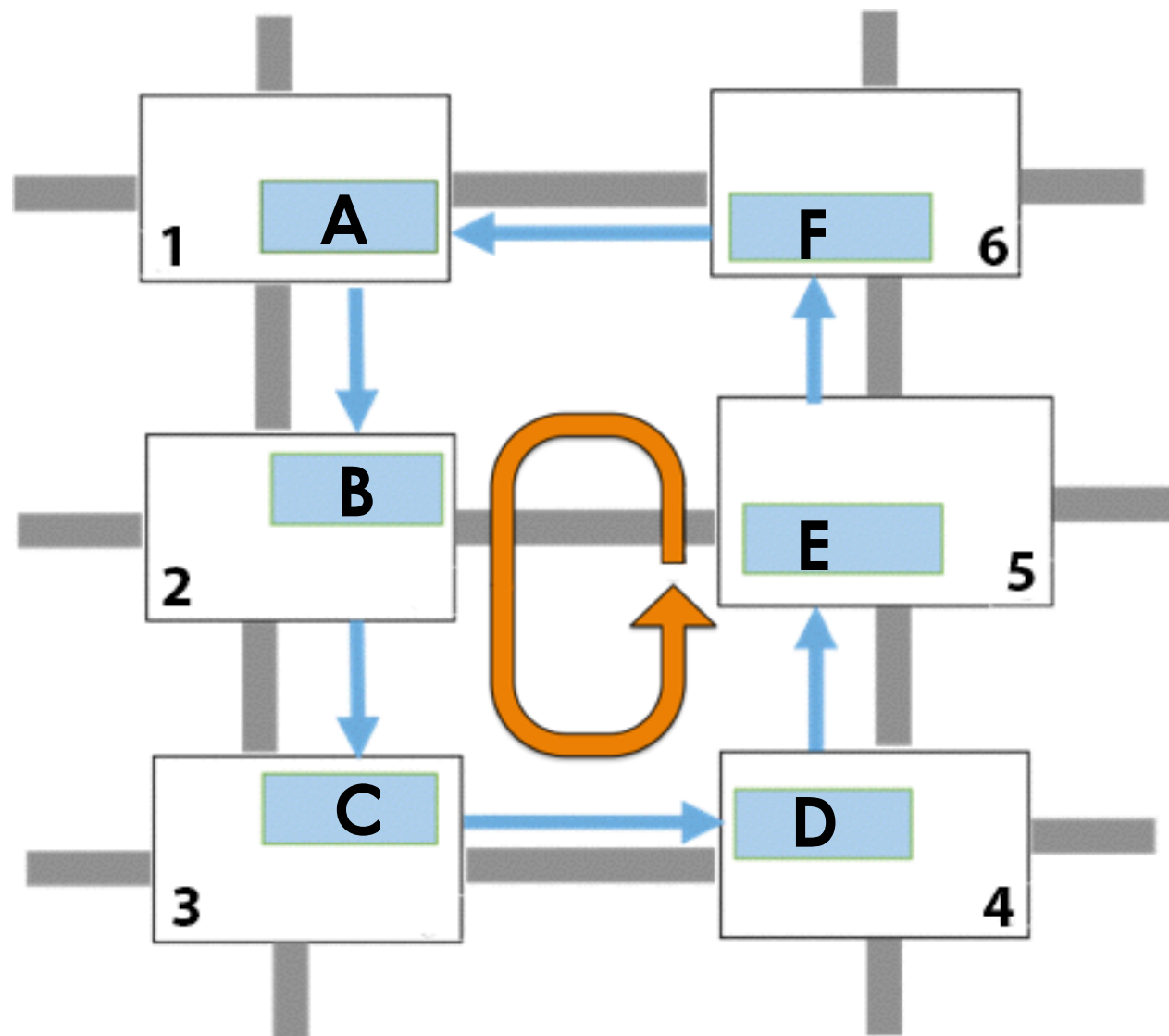
- *Simultaneous Synchronized Movement* of all deadlocked packets in the loop is called a *spin*.
- Each spin leads to *one hop forward movement* of all deadlock packets.
- One spin may not resolve the deadlock. If so, spin can be repeated

SPIN : Key Idea

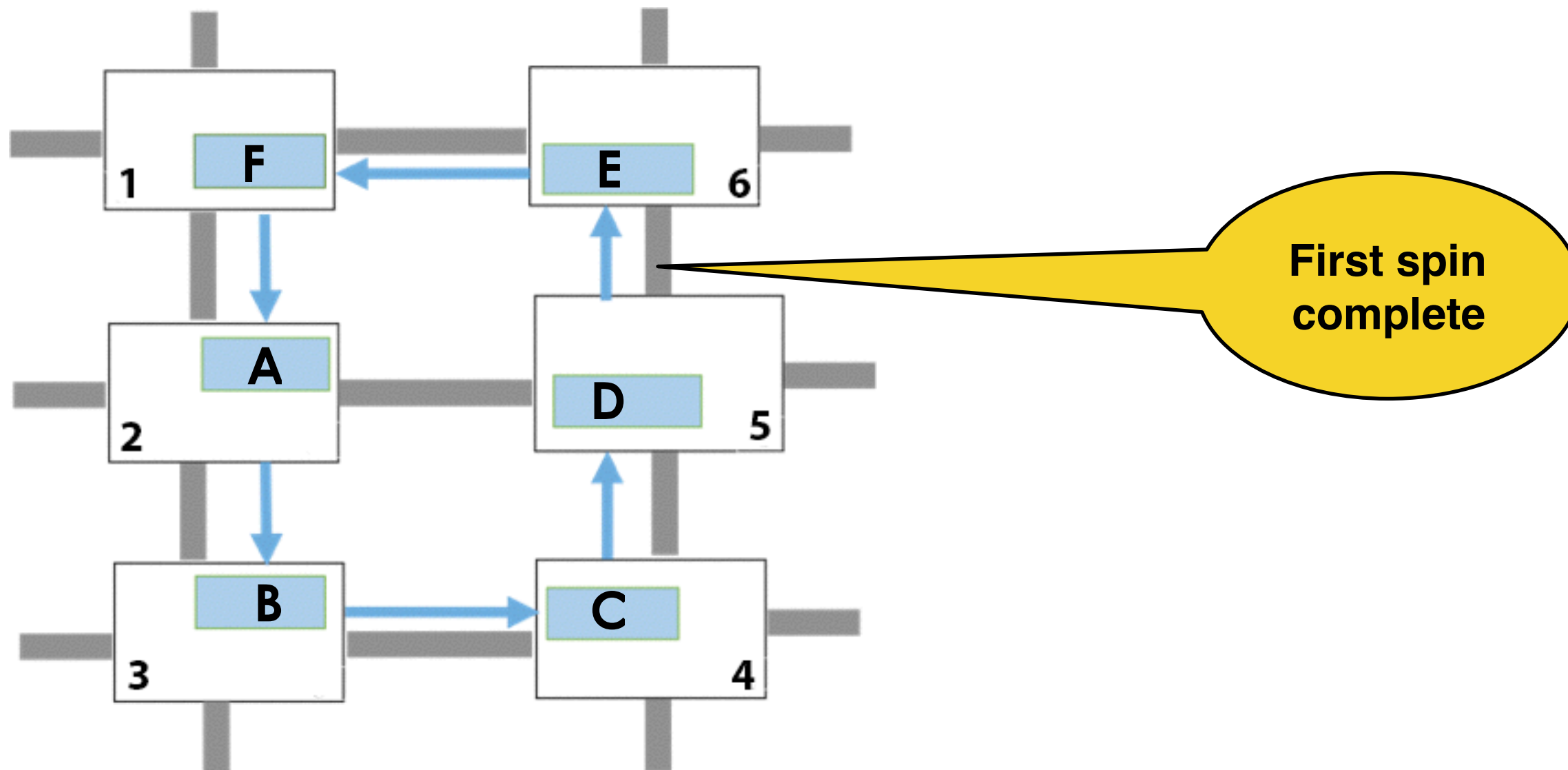
- *Simultaneous Synchronized Movement* of all deadlocked packets in the loop is called a *spin*.
- Each spin leads to *one hop forward movement* of all deadlock packets.
- One spin may not resolve the deadlock. If so, spin can be repeated
- Deadlock is *guaranteed to be resolved* in a *finite* number of spins [proof in paper, Sec. III]

SPIN : Key Idea

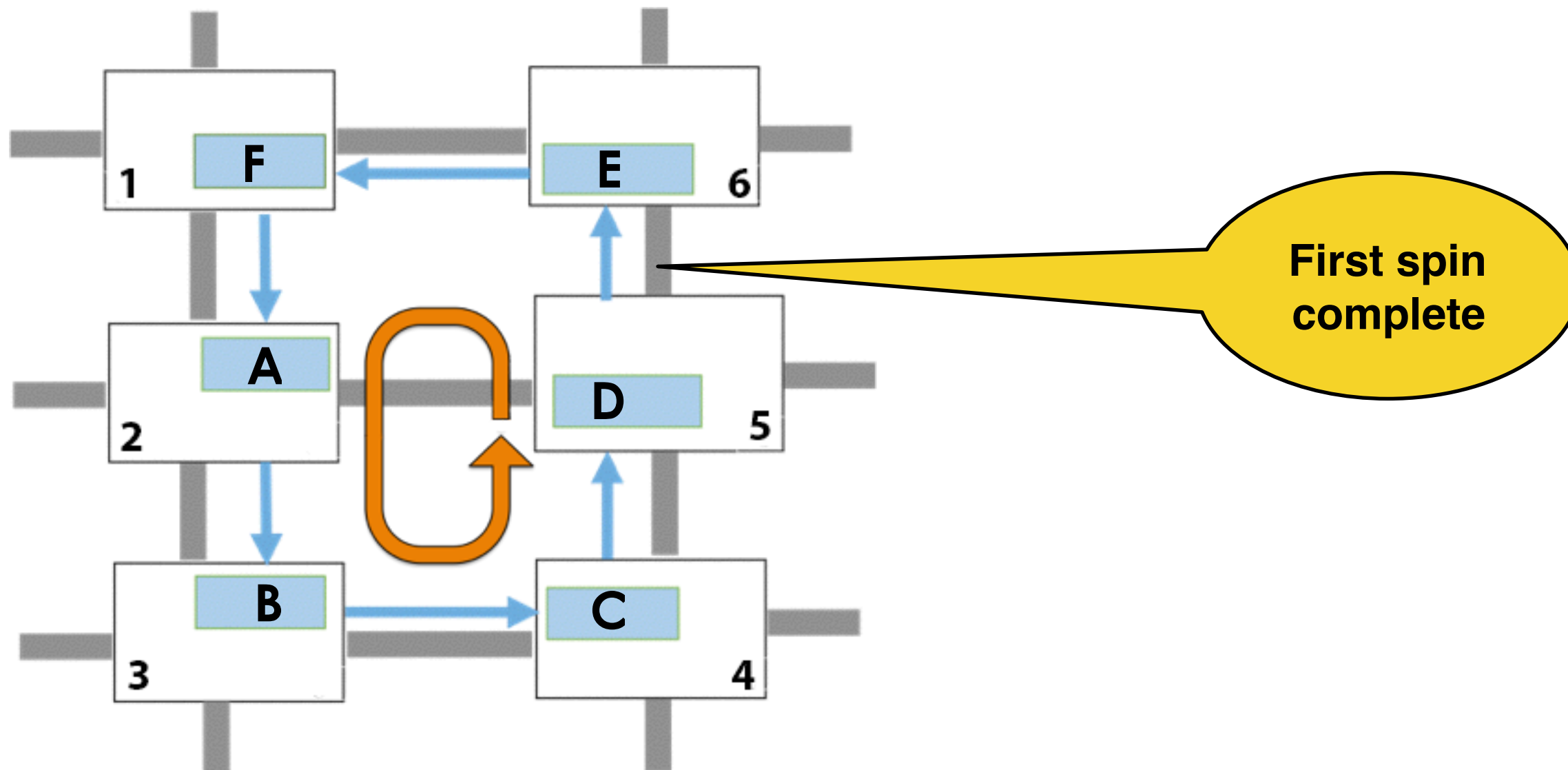
SPIN : Key Idea



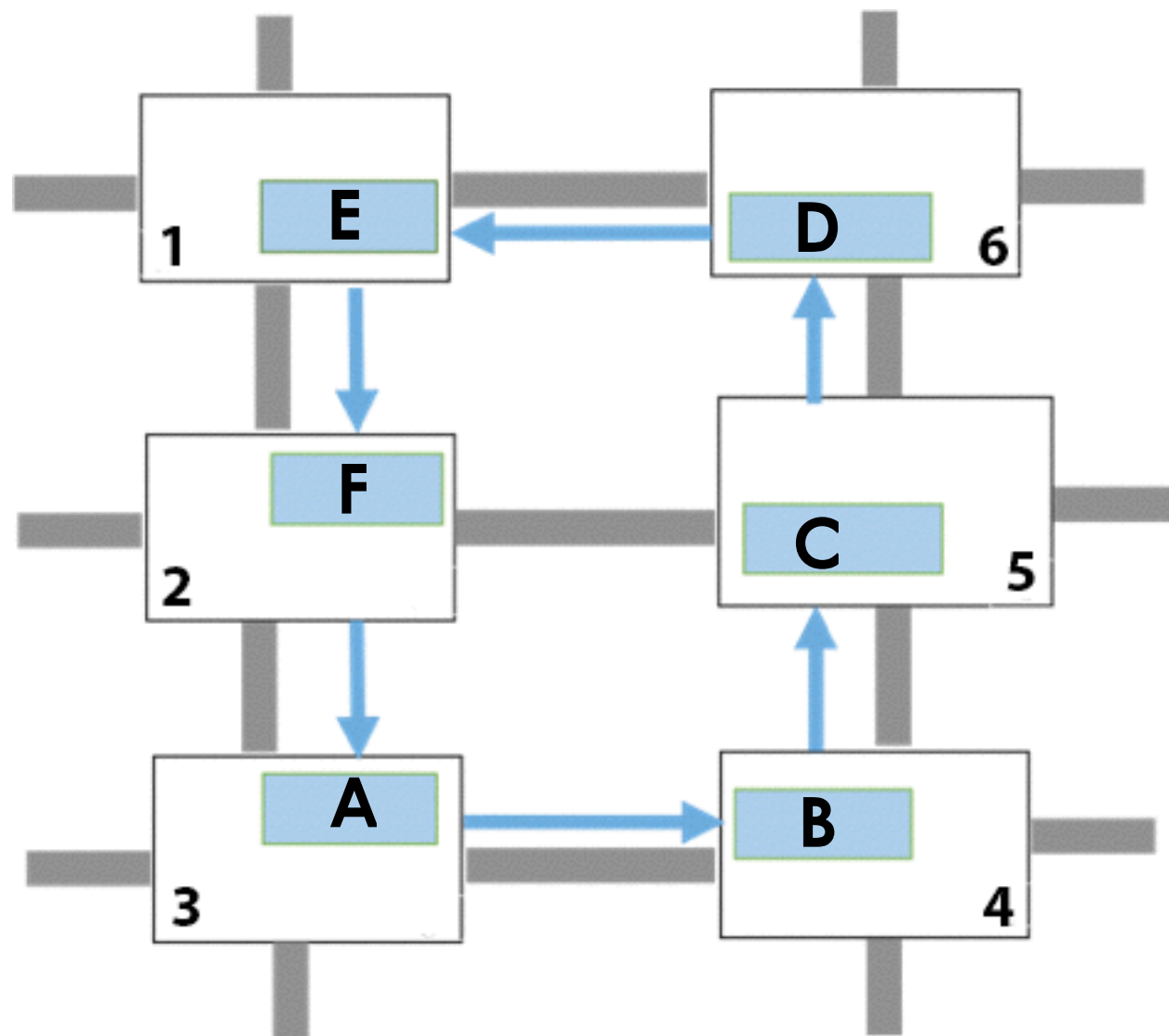
SPIN : Key Idea



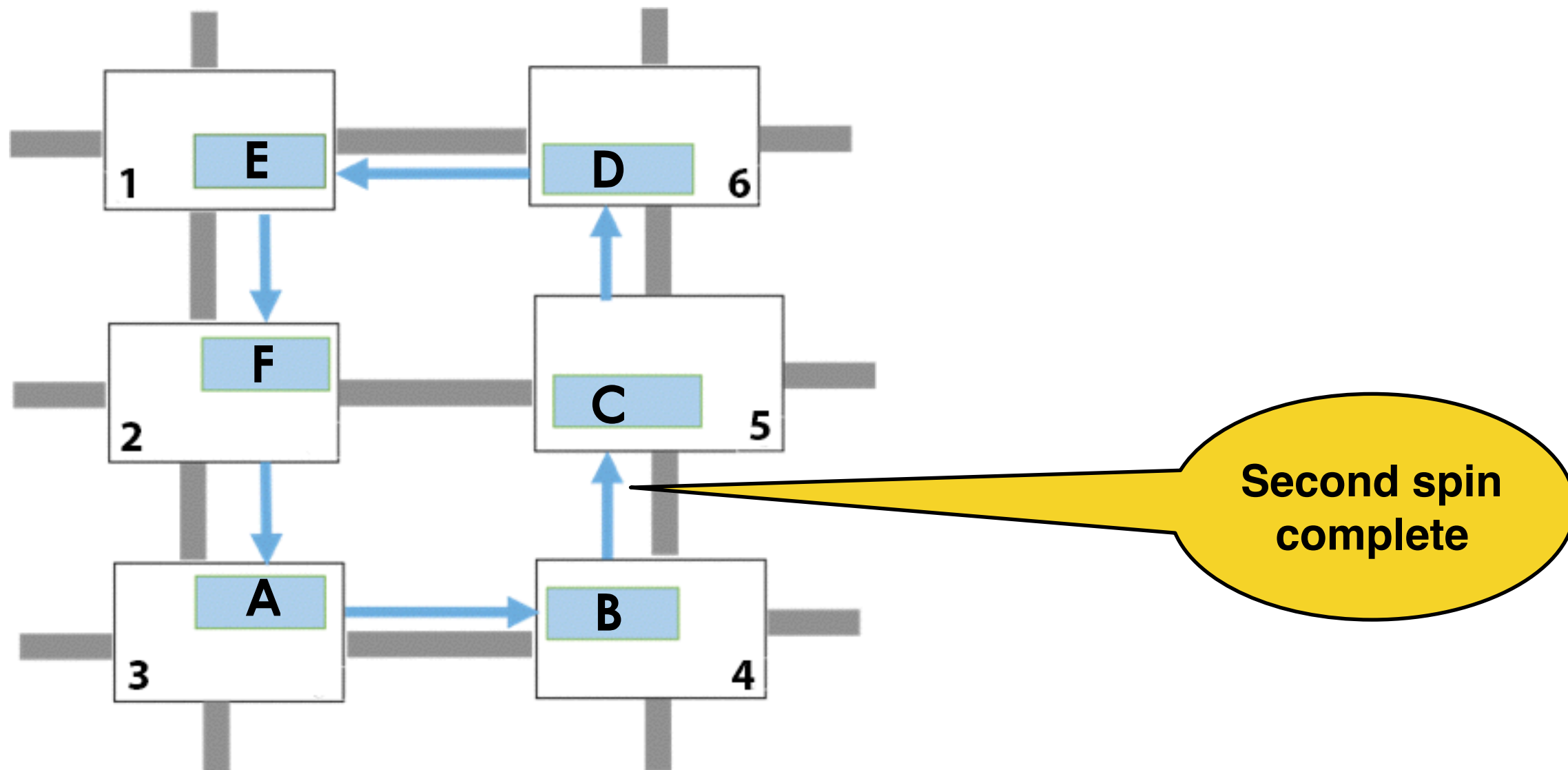
SPIN : Key Idea



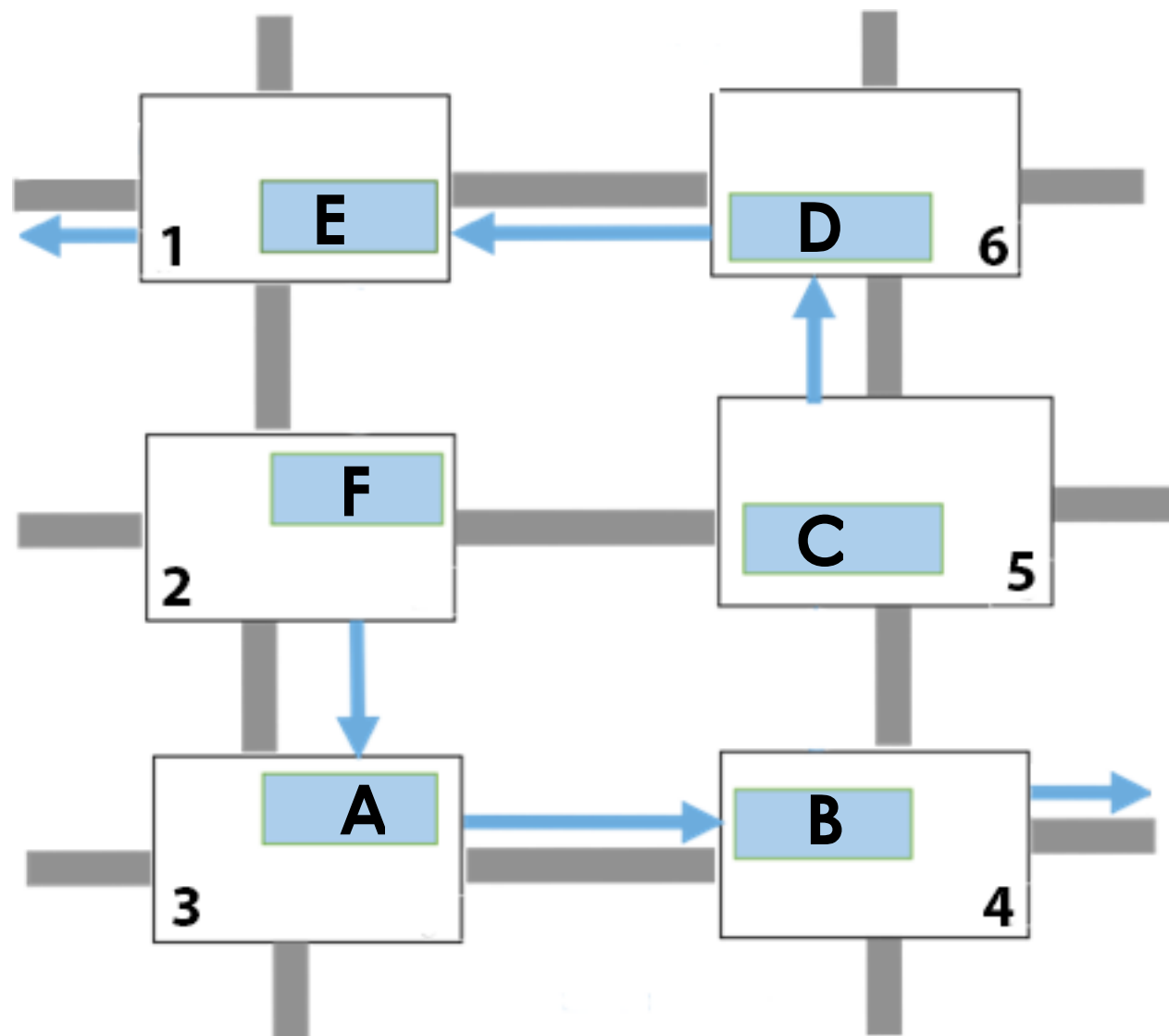
SPIN : Key Idea



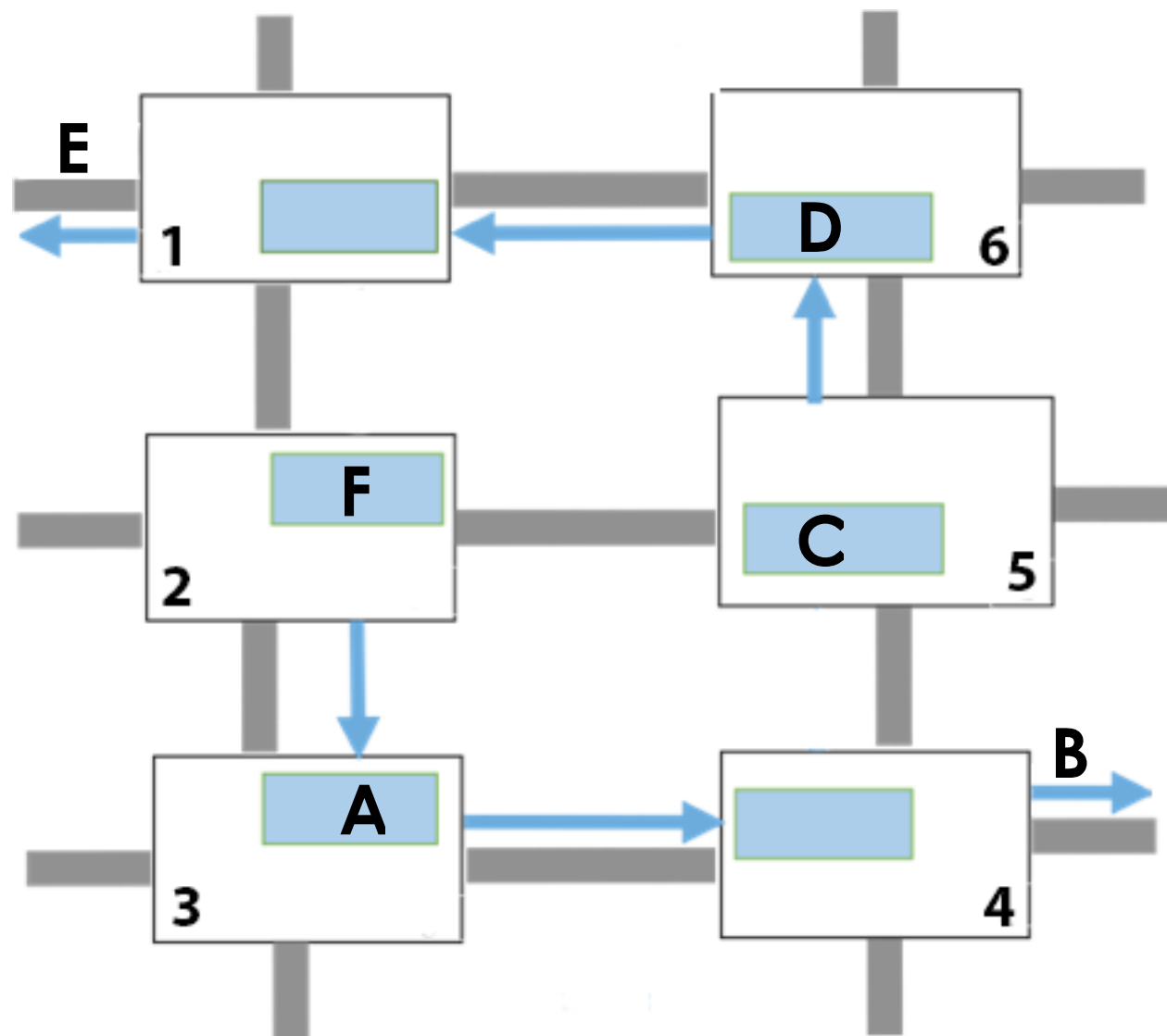
SPIN : Key Idea



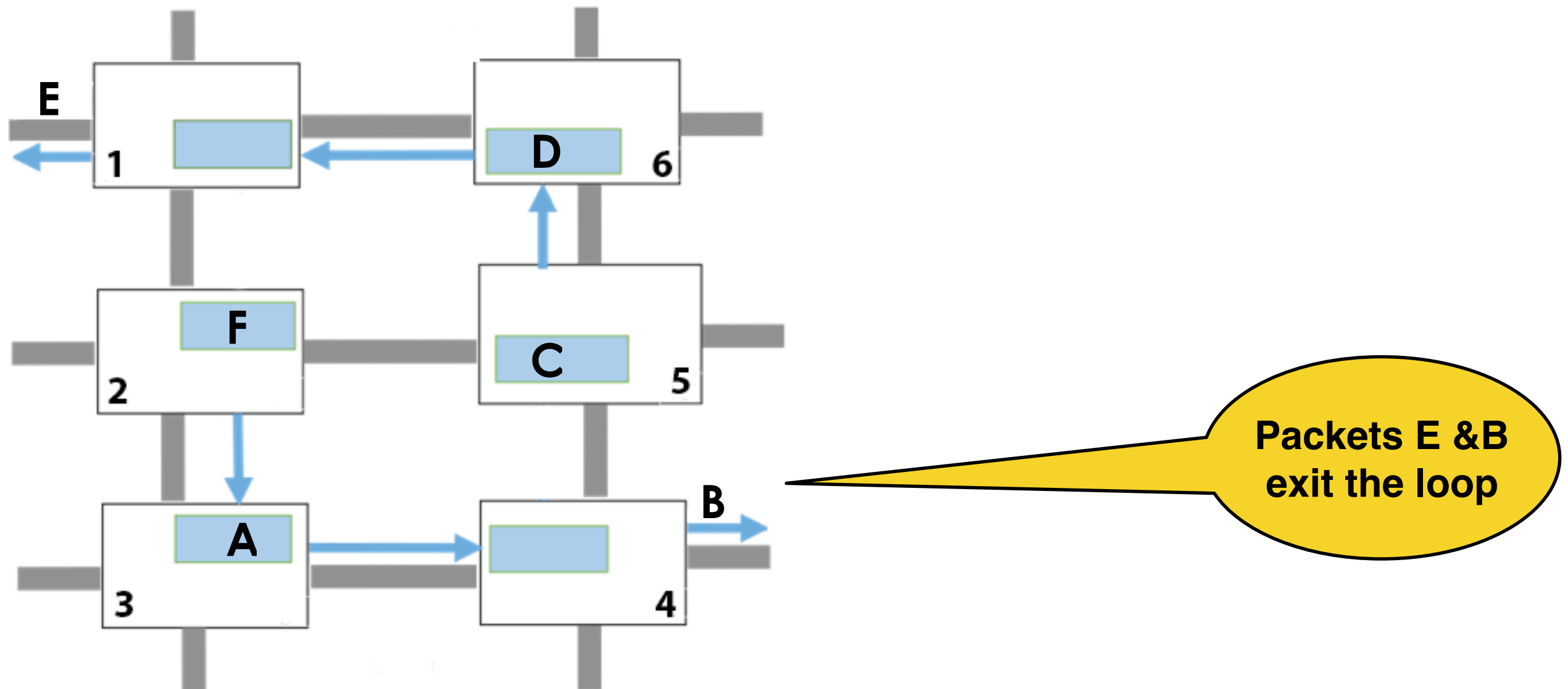
SPIN : Key Idea



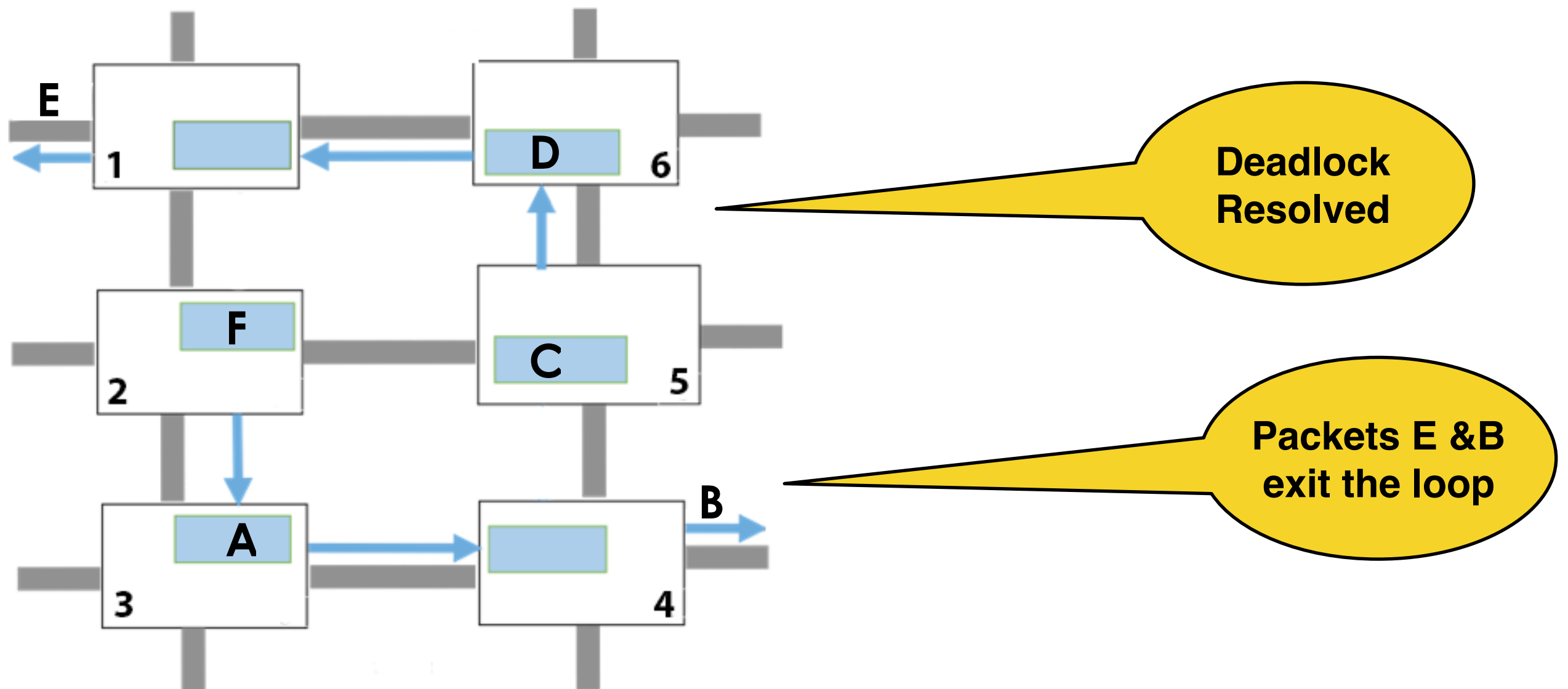
SPIN : Key Idea



SPIN : Key Idea



SPIN : Key Idea



Outline

- Routing Deadlocks
 - State of the Art
 - **SPIN** : **S**ynchronized **P**rogress in **I**nterconnection **N**etworks
 - Key Idea
 - Implementation Example
 - Micro-architecture
 - FAvORS
 - Evaluations
 - Conclusion
-

SPIN: Implementation Example

SPIN: Implementation Example

- SPIN is a generic deadlock freedom theory that can have *multiple implementations*.

SPIN: Implementation Example

- SPIN is a generic deadlock freedom theory that can have *multiple implementations*.
- *SPIN Implementation*:
 - Detect the Deadlock.
 - Coordinate a time for spin.
 - Execute the spin.

SPIN: Implementation Example

- SPIN is a generic deadlock freedom theory that can have *multiple implementations*.
- *SPIN Implementation*:
 - Detect the Deadlock.
 - Coordinate a time for spin.
 - Execute the spin.
- We choose a *recovery approach* as *deadlocks* are *rare* scenarios (See Sec. II-F).

Implementation Example : Detect Deadlocks

Implementation Example : Detect Deadlocks

- Use *counters*.

Implementation Example : Detect Deadlocks

- Use *counters*.
- Placed at *every node* at design time.


Implementation Example : Detect Deadlocks

- Use *counters*.
- Placed at *every node* at design time.
 - Optimize by exploiting topology symmetry (See Static Bubble [6]).

Implementation Example : Detect Deadlocks

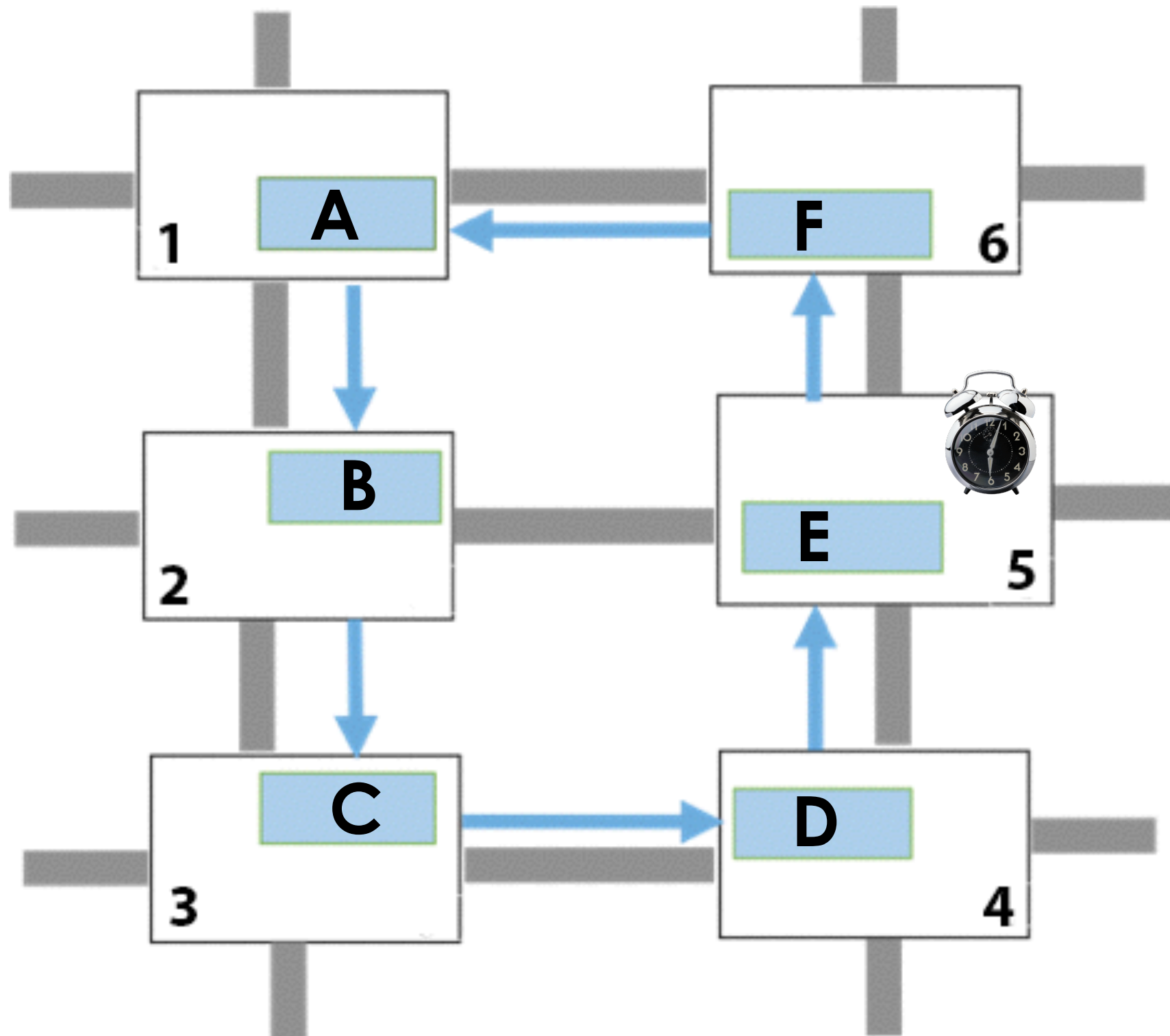
- Use *counters*.
- Placed at *every node* at design time.
 - Optimize by exploiting topology symmetry (See Static Bubble [6]).
- If packet does not leave in *threshold time* (configurable), it indicates a *potential deadlock*.

Implementation Example : Detect Deadlocks

- Use **counters**.
- Placed at **every node** at design time.
 - Optimize by exploiting topology symmetry (See Static Bubble [6]).
- If packet does not leave in **threshold time** (configurable), it indicates a **potential deadlock**.
- Counter expired ?  Send **probe** to verify deadlock.

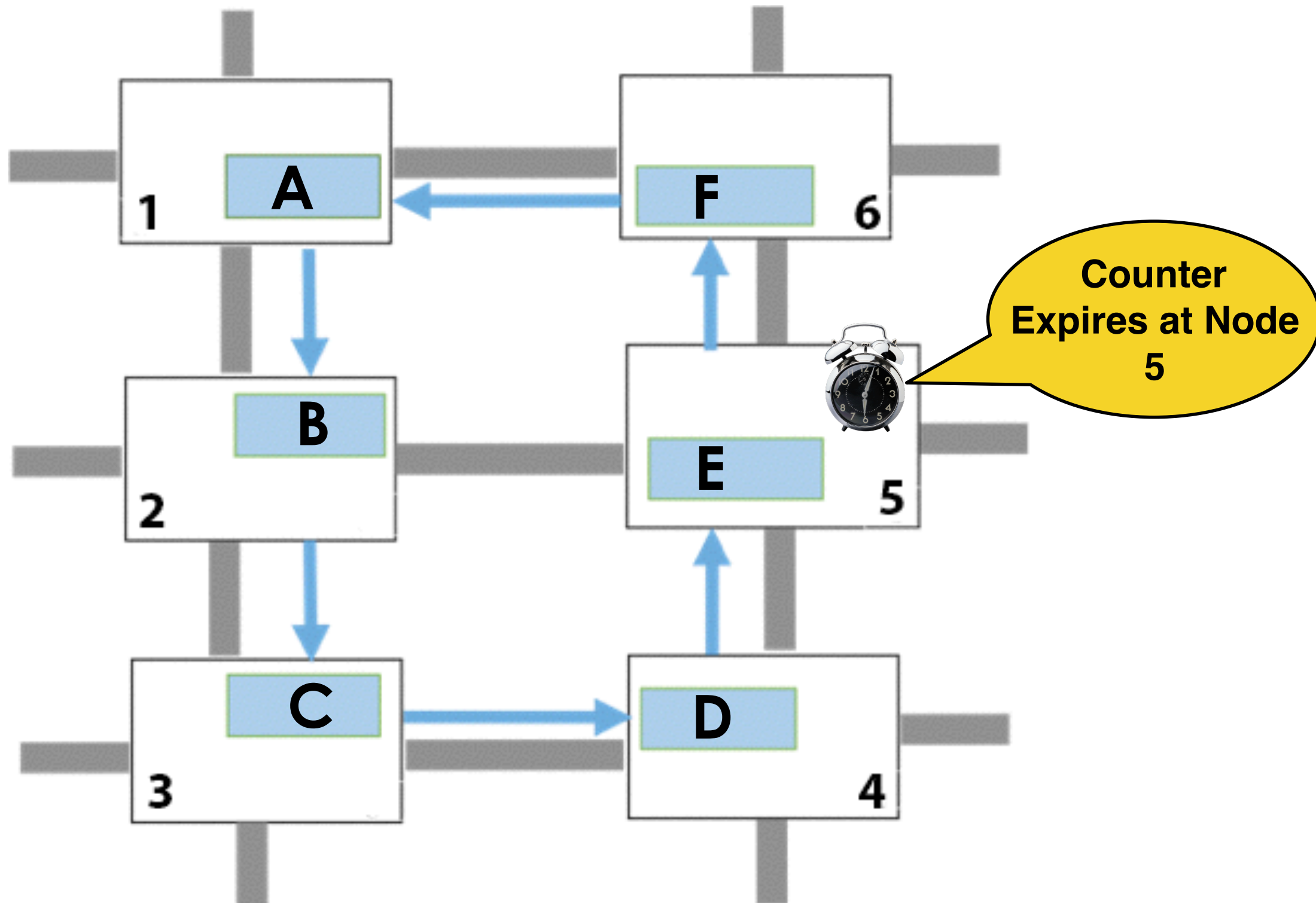
Implementation Example : *Probe Msg.*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.



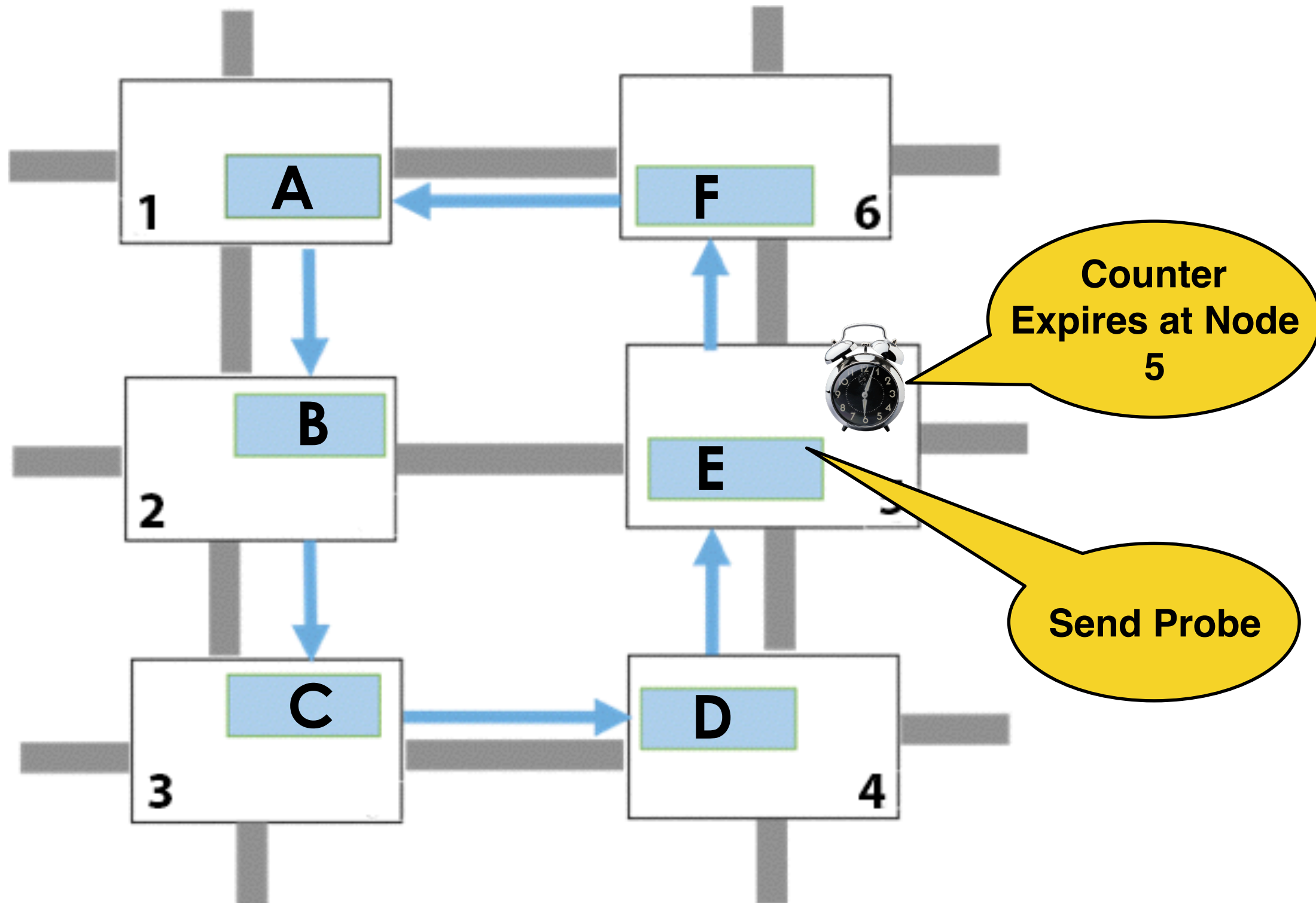
Implementation Example : *Probe Msg.*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.



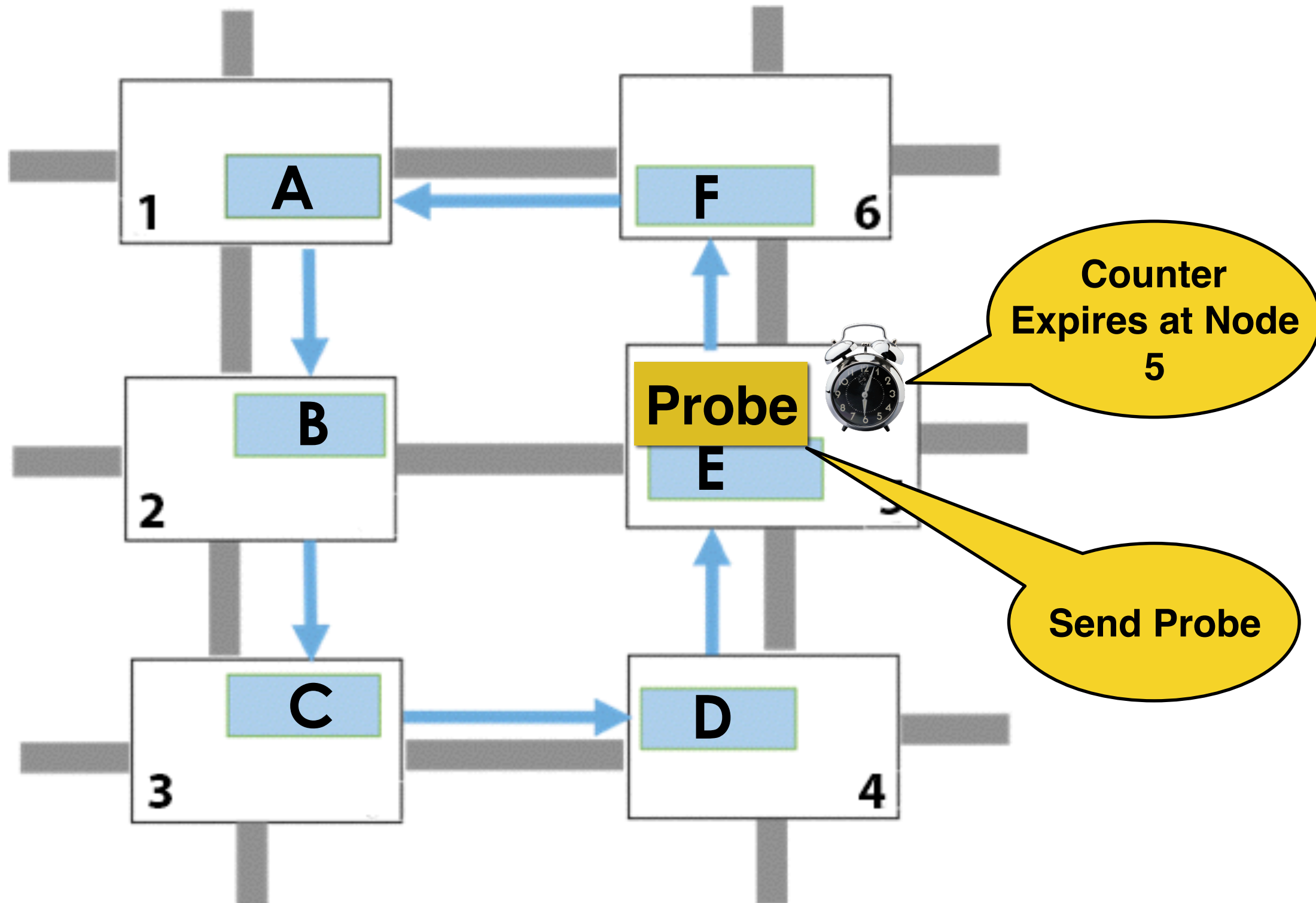
Implementation Example : *Probe Msg.*

1. **Deadlock Detection**
2. Coordinating the spin.
3. Executing the spin.



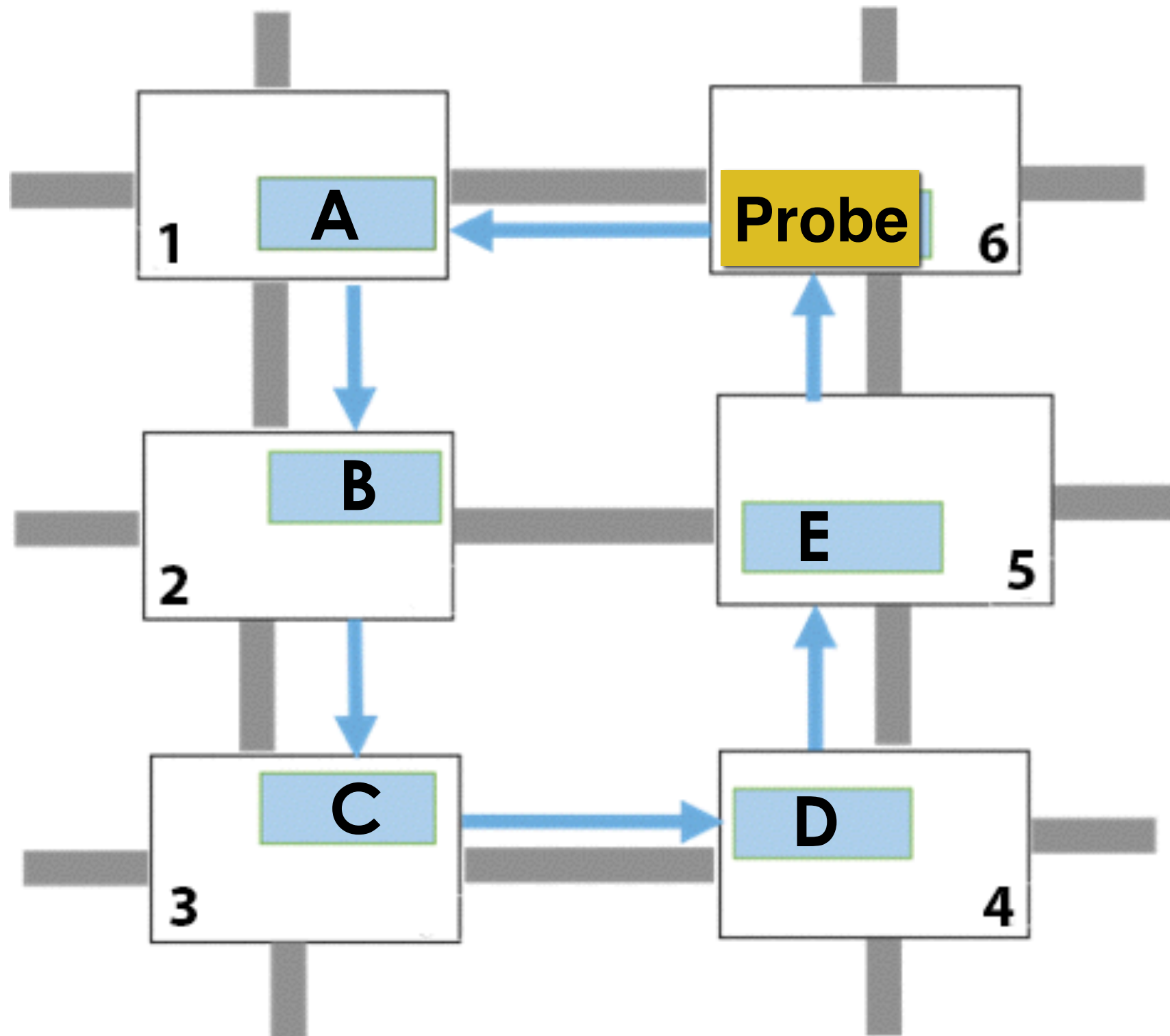
Implementation Example : *Probe Msg.*

1. **Deadlock Detection**
2. Coordinating the spin.
3. Executing the spin.



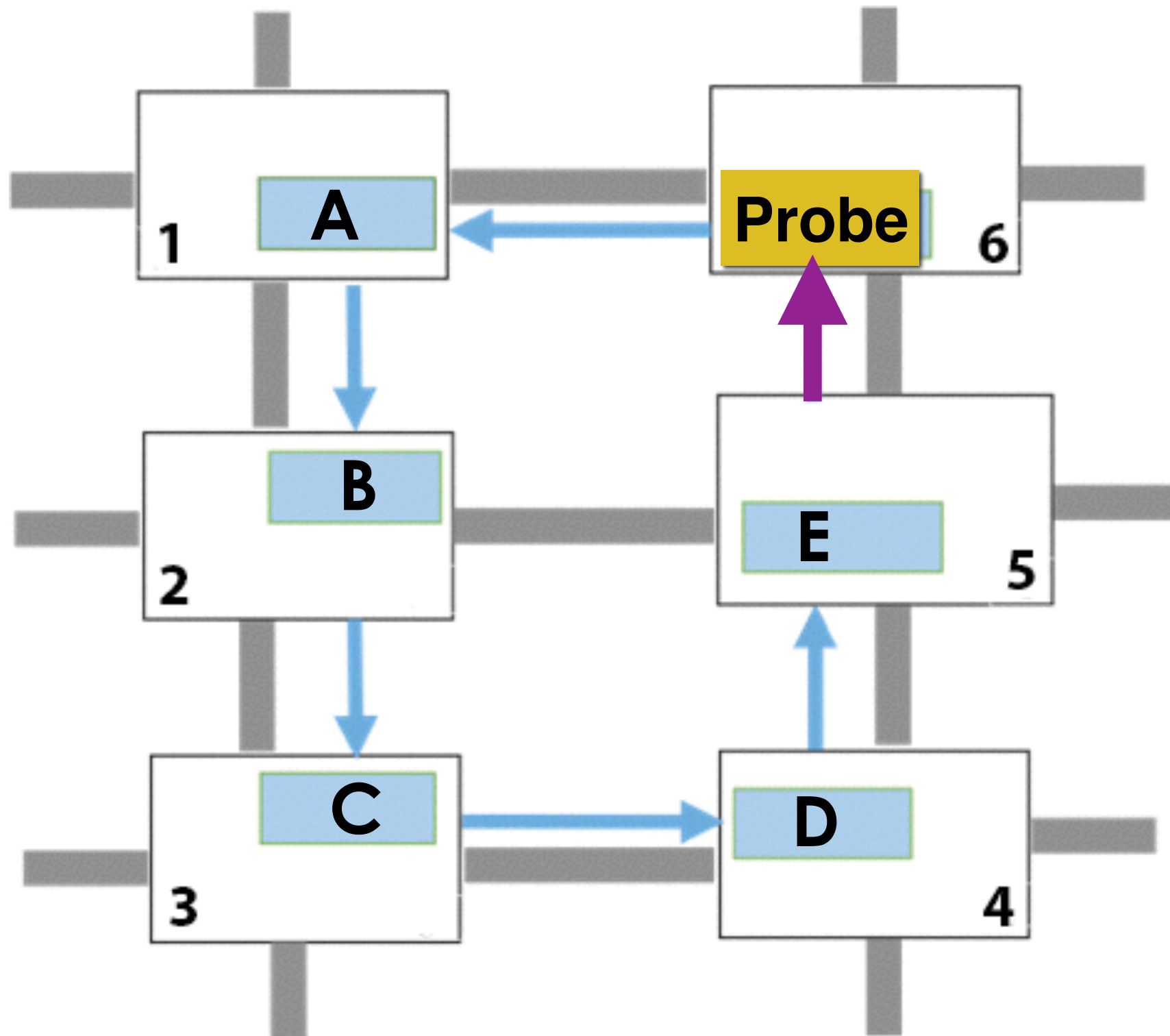
Implementation Example : *Probe Msg.*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.



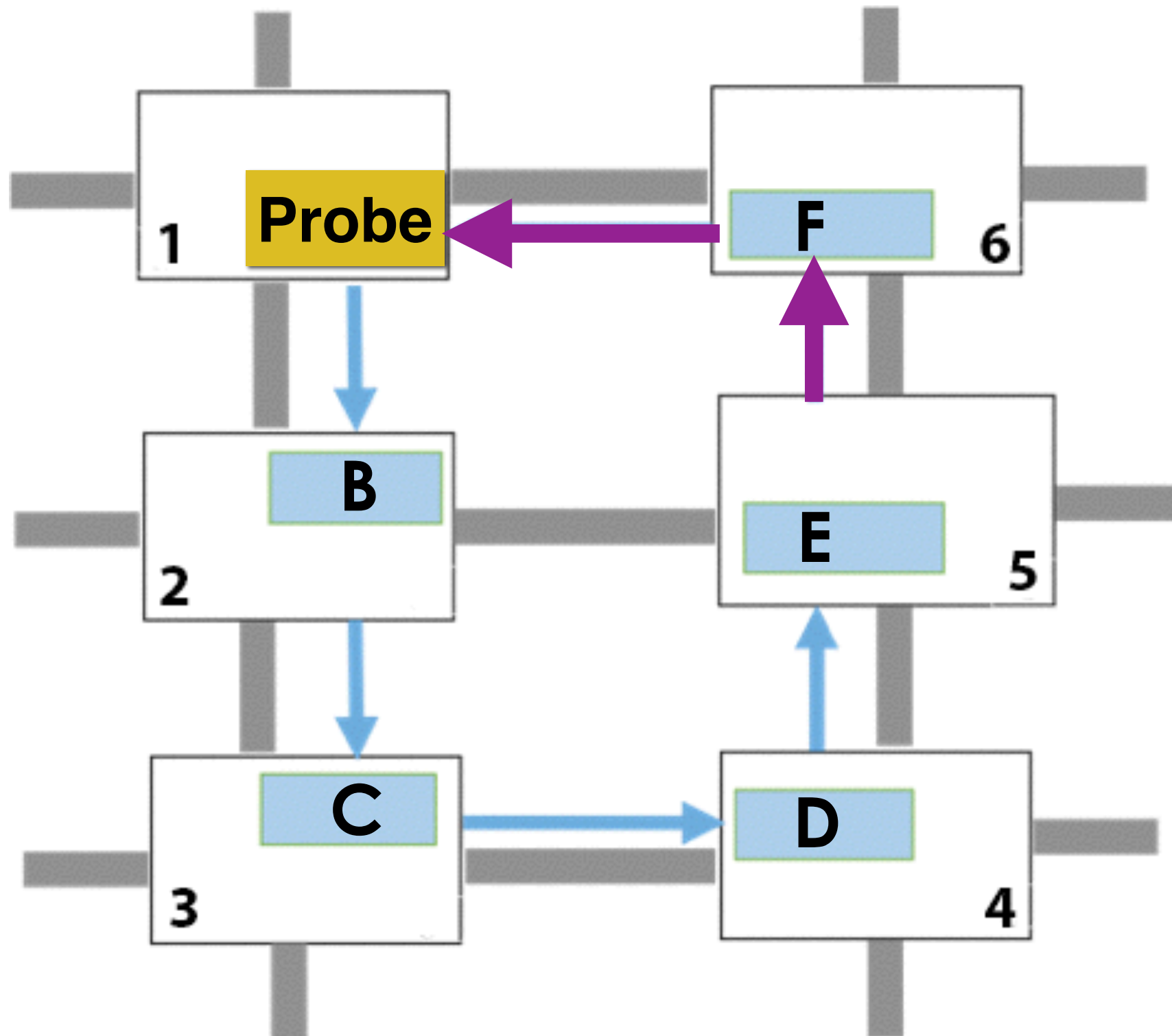
Implementation Example : *Probe Msg.*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.



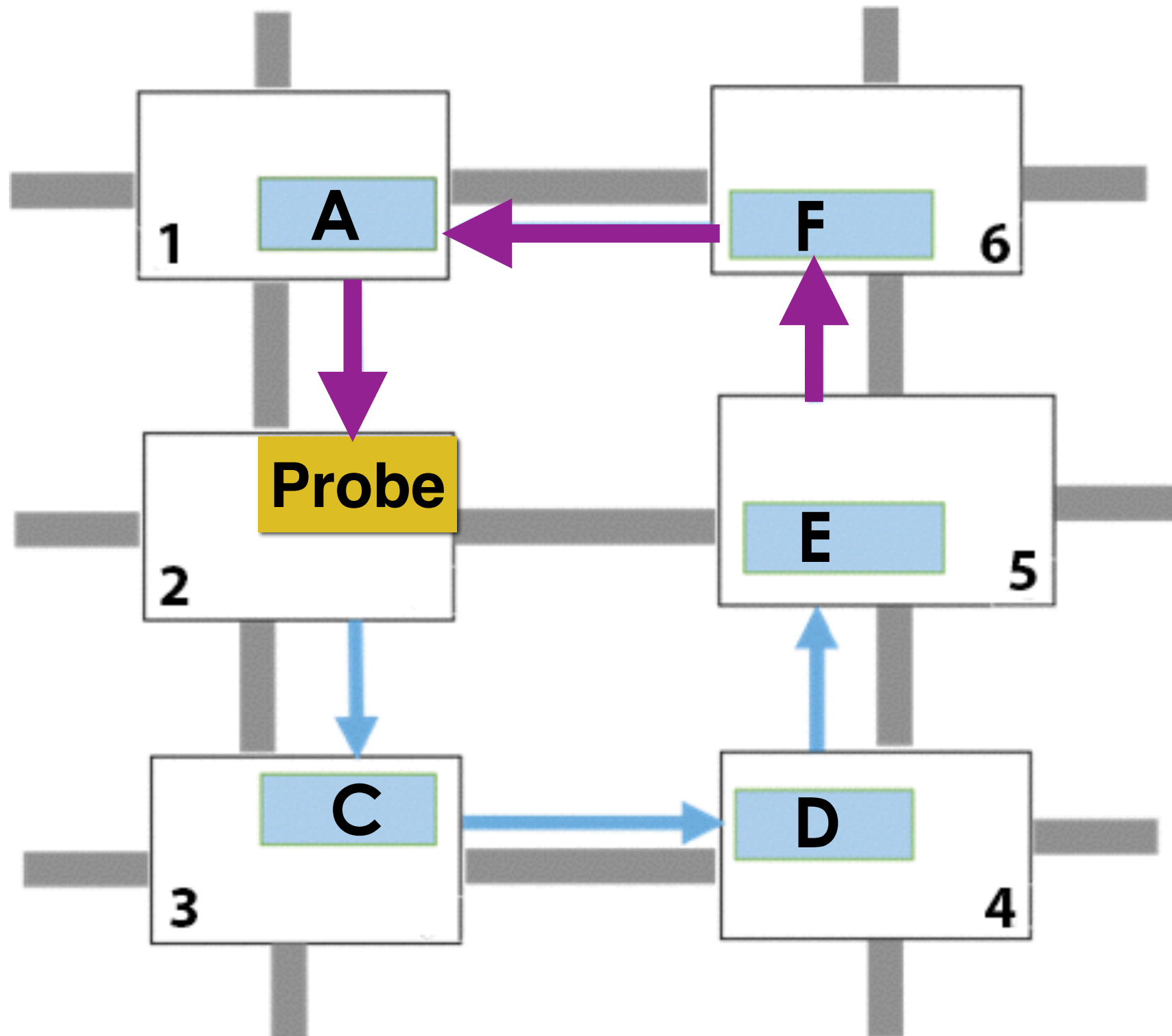
Implementation Example : *Probe Msg.*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.



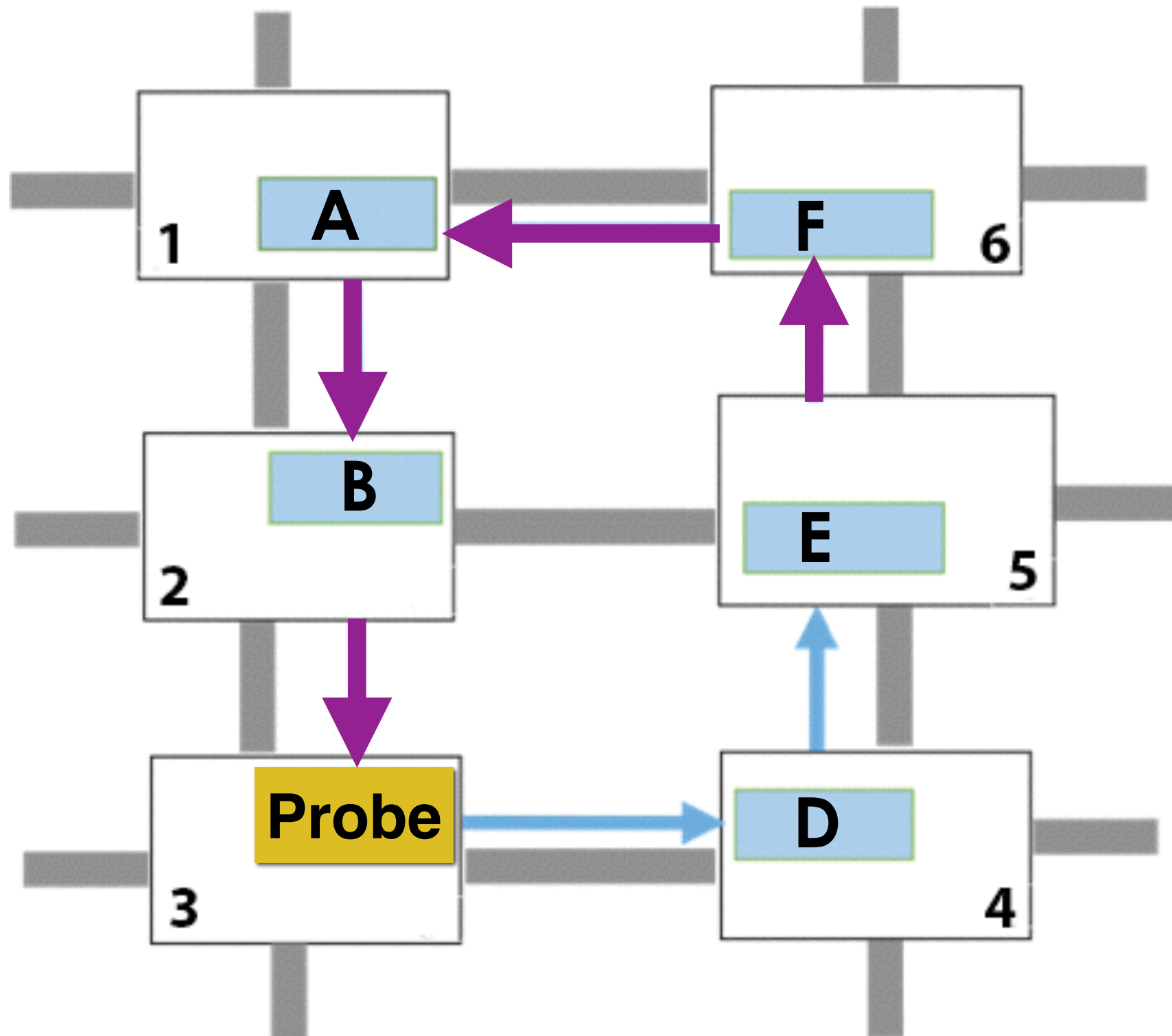
Implementation Example : *Probe Msg.*

1. **Deadlock Detection**
2. Coordinating the spin.
3. Executing the spin.



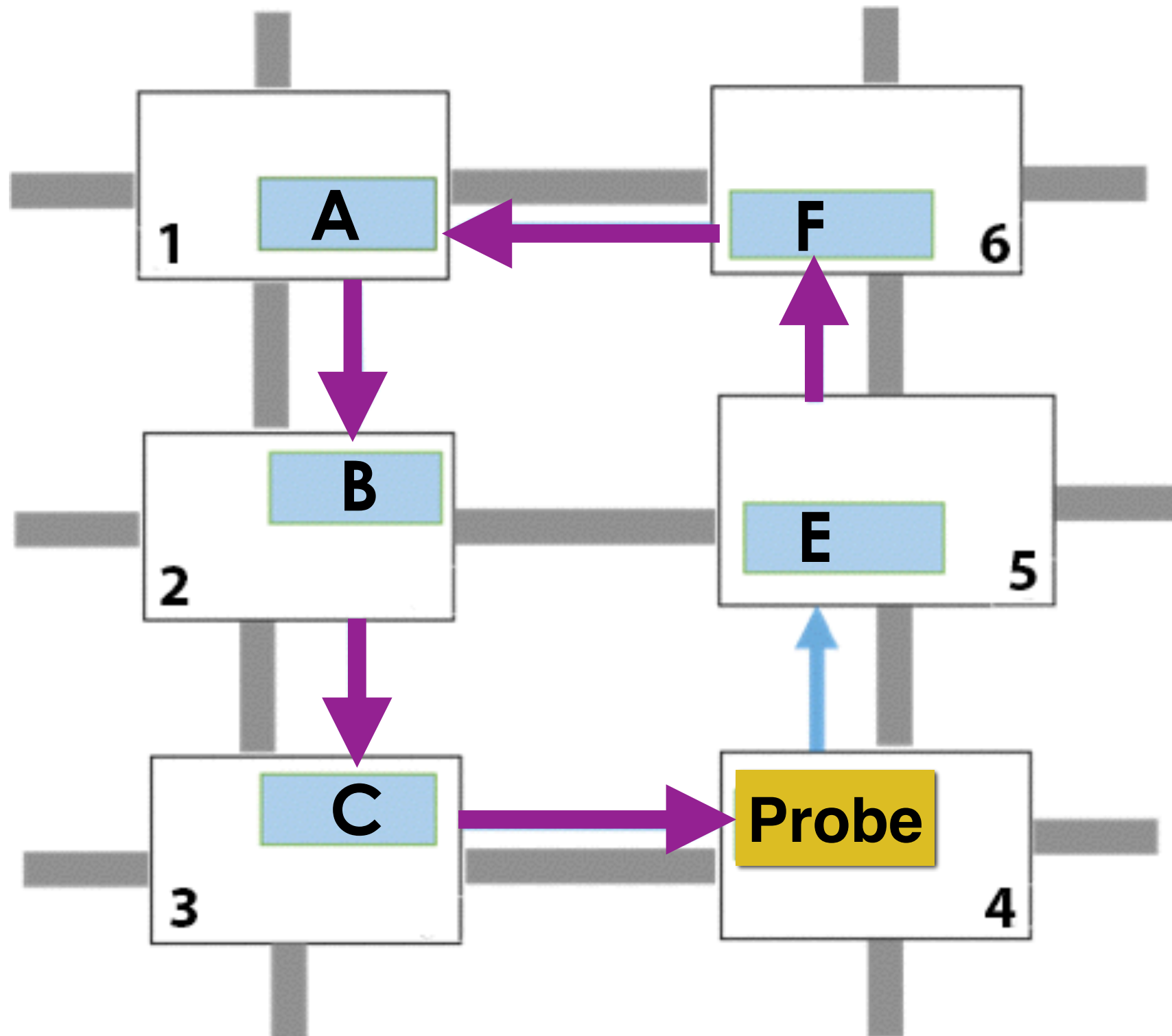
Implementation Example : *Probe Msg.*

1. **Deadlock Detection**
2. Coordinating the spin.
3. Executing the spin.



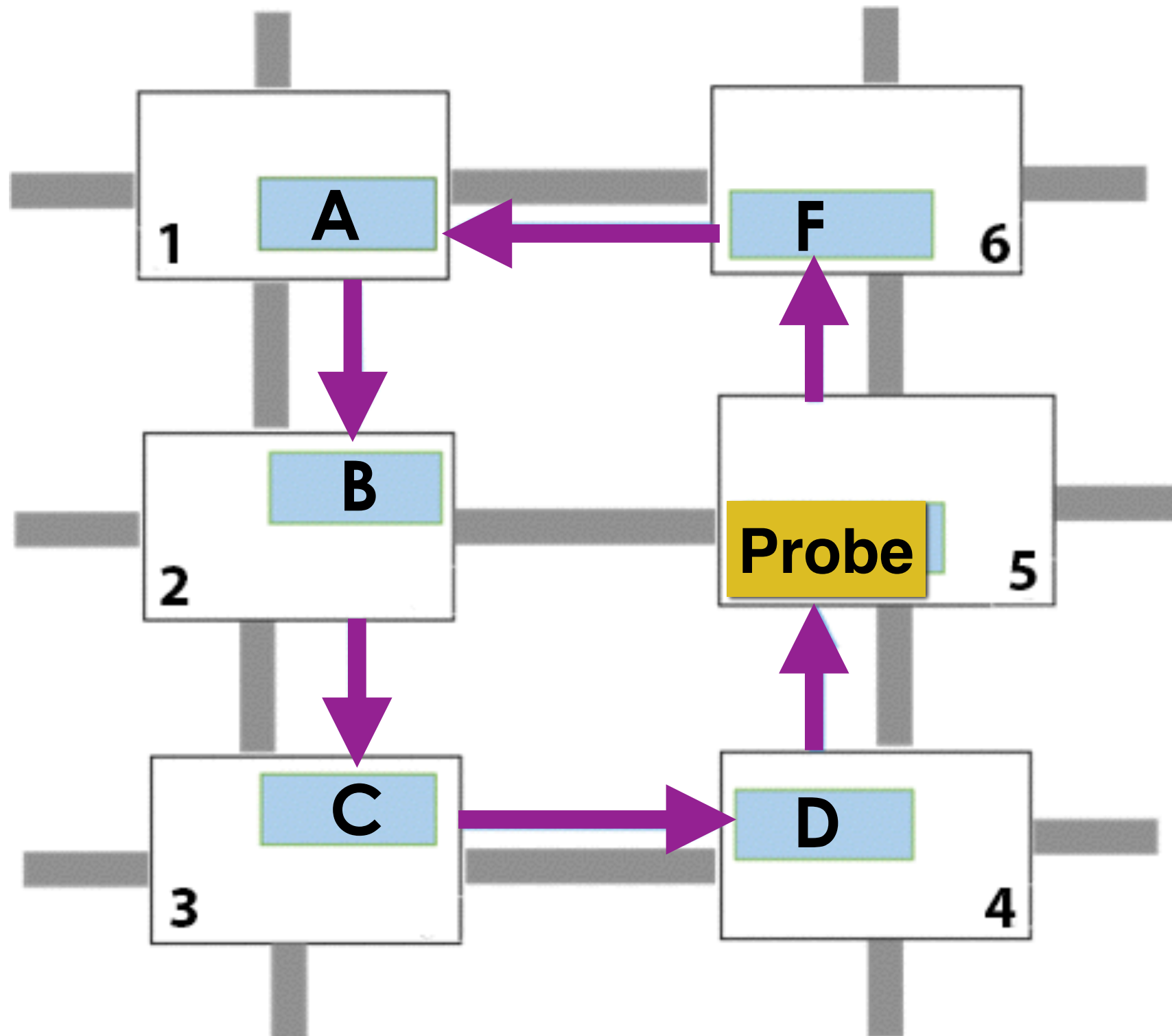
Implementation Example : *Probe Msg.*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.



Implementation Example : *Probe Msg.*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.

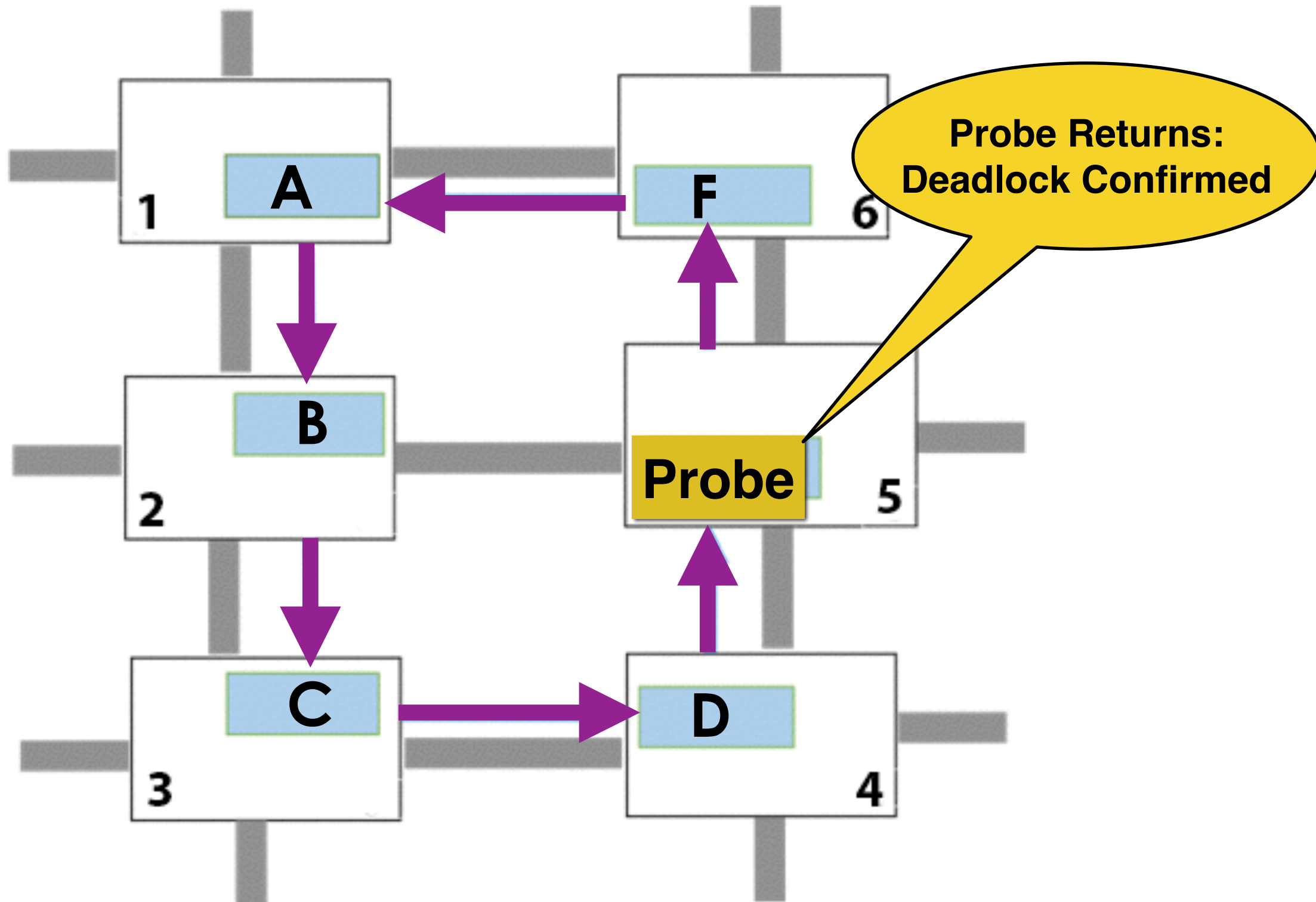


Implementation Example : *Probe Msg.*

1. **Deadlock Detection**

2. Coordinating the spin.

3. Executing the spin.



Implementation Example : *Probe Msg.*

Implementation Example : *Probe Msg.*

- ***Probe*** is a special message that ***tracks*** the ***buffer dependency***.

Implementation Example : *Probe Msg.*

- ***Probe*** is a special message that ***tracks*** the ***buffer dependency***.
- ***Probe returns*** to sender:

Implementation Example : *Probe Msg.*

- **Probe** is a special message that **tracks** the **buffer dependency**.
- **Probe returns** to sender:
 - Cyclic buffer dependence, hence **deadlock**.

Implementation Example : *Probe Msg.*

- **Probe** is a special message that **tracks** the **buffer dependency**.
- **Probe returns** to sender:
 - Cyclic buffer dependence, hence **deadlock**.
- Next, send a **move** msg. to convey the **spin time**

Implementation Example : *Probe Msg.*

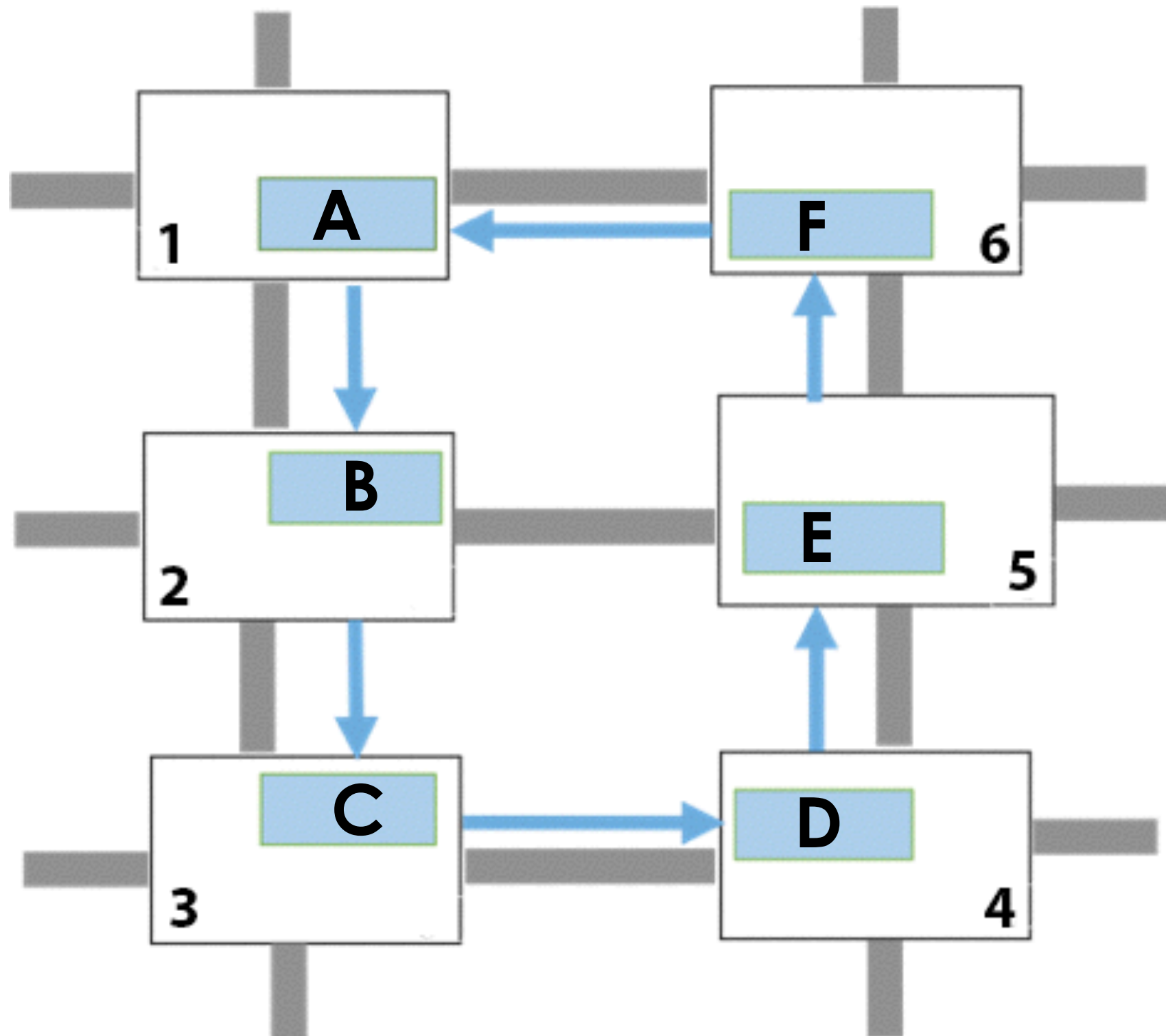
- **Probe** is a special message that **tracks** the **buffer dependency**.
- **Probe returns** to sender:
 - Cyclic buffer dependence, hence **deadlock**.
- Next, send a **move** msg. to convey the **spin time**
 - Upon receiving move msg., router sets its **counter** to count to **spin cyle**.

Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

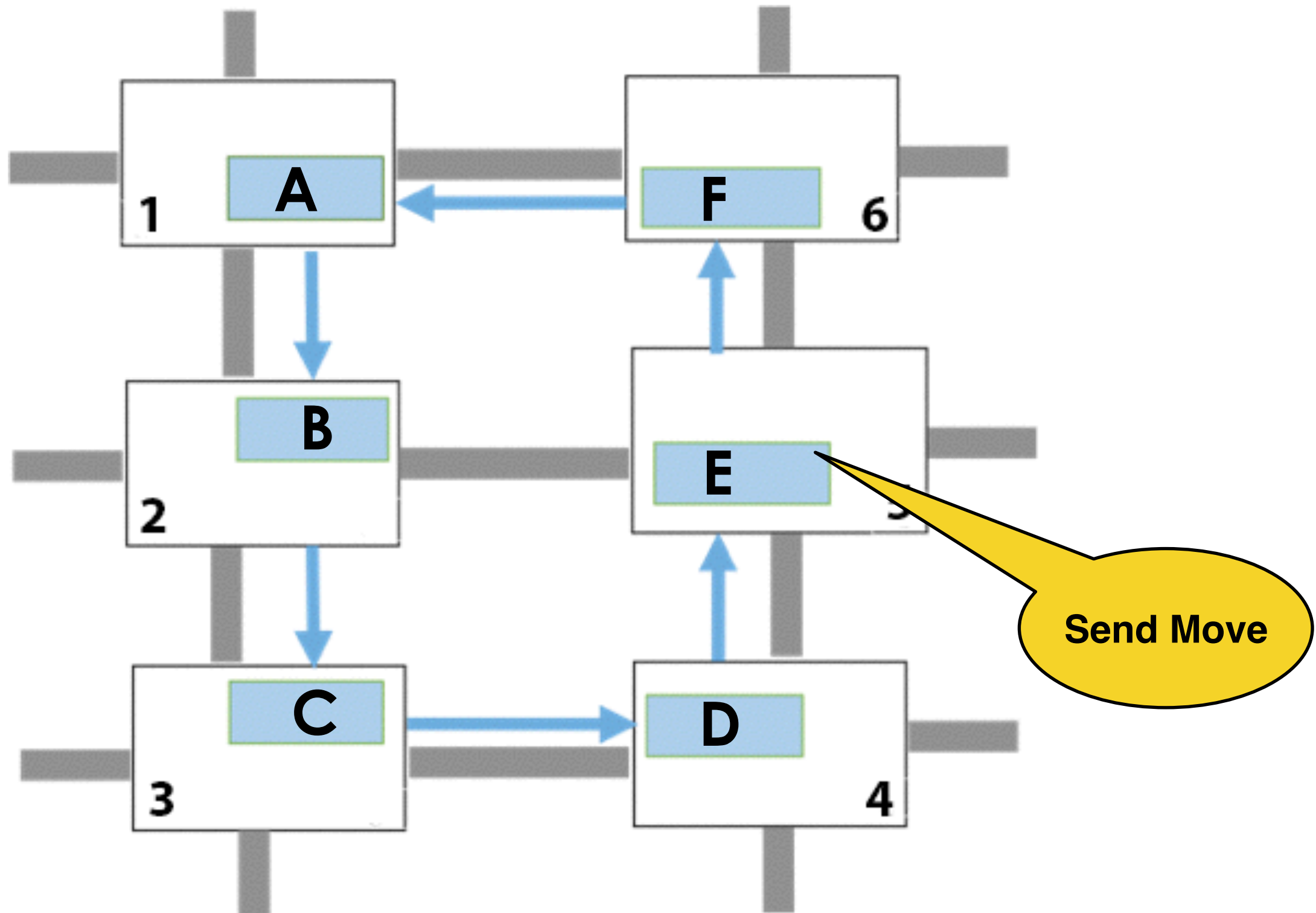


Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

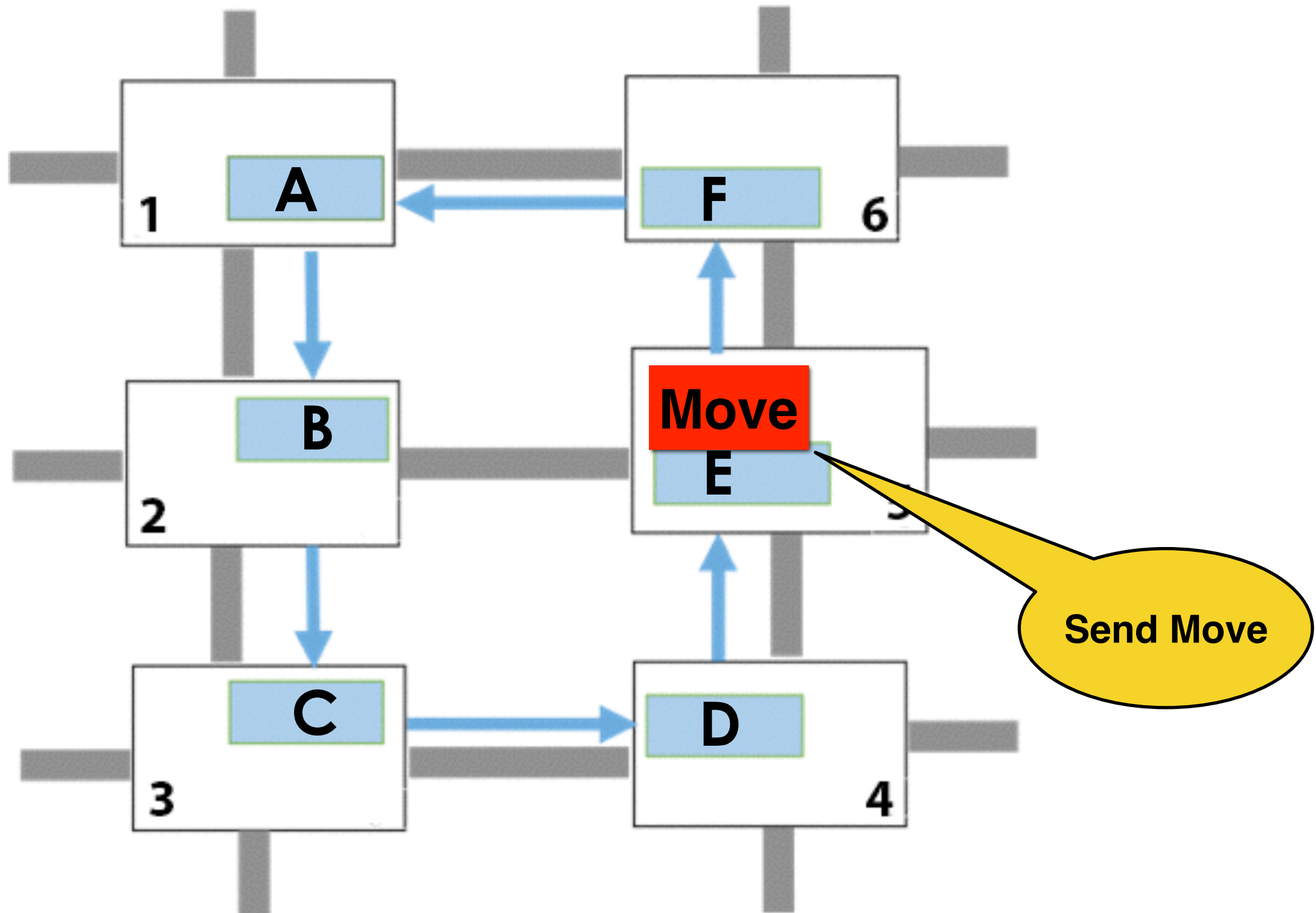


Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

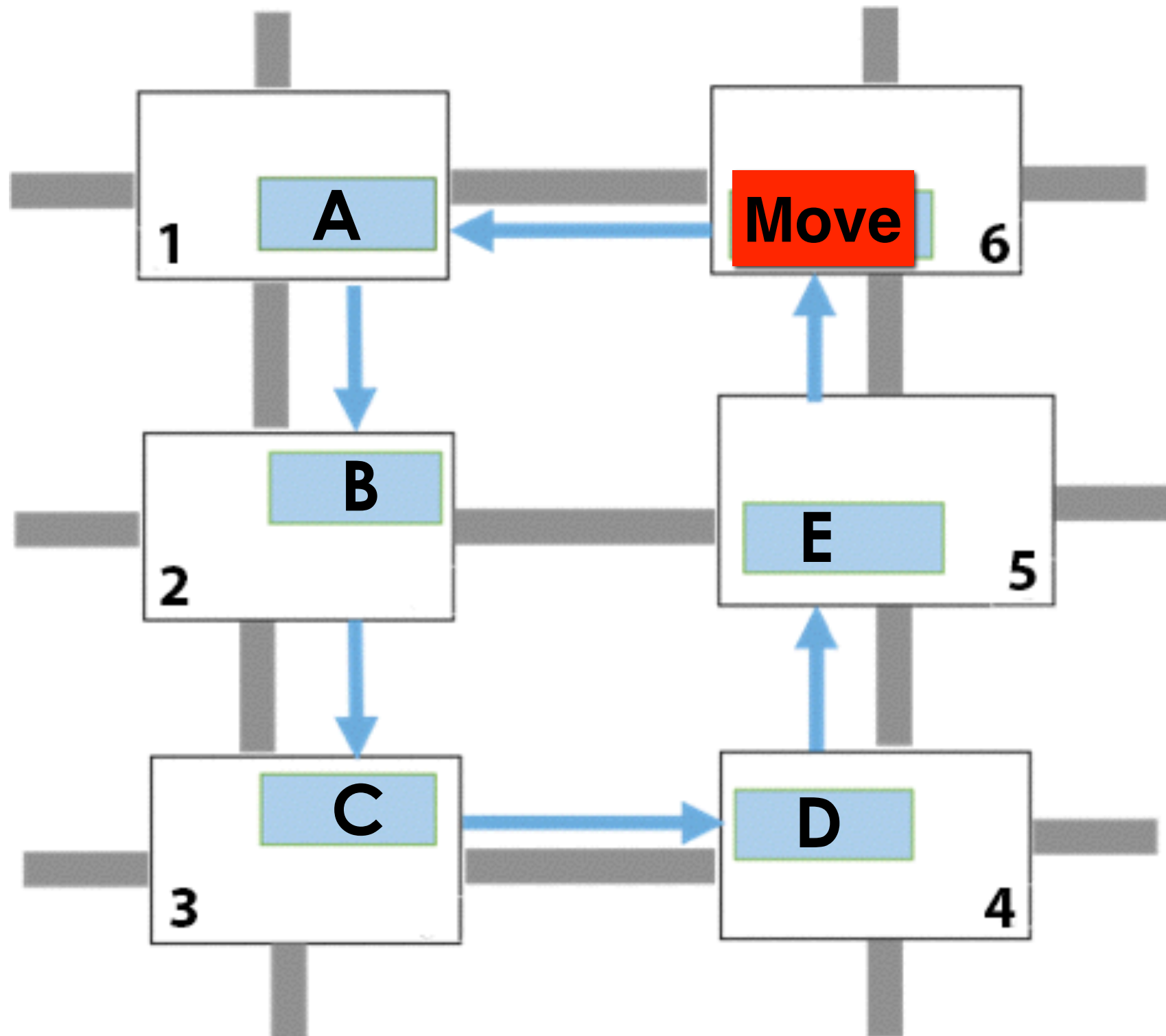


Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

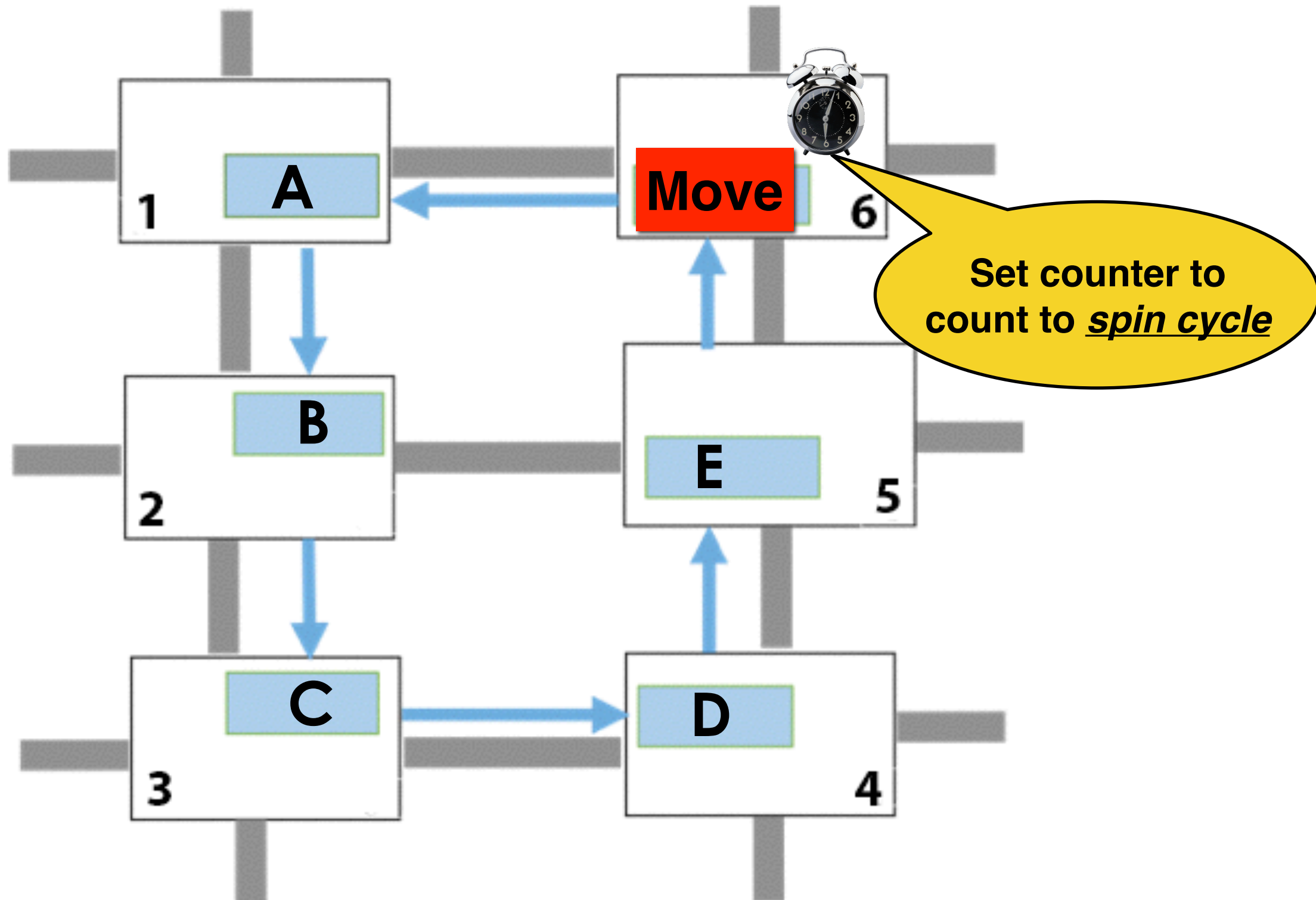


Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

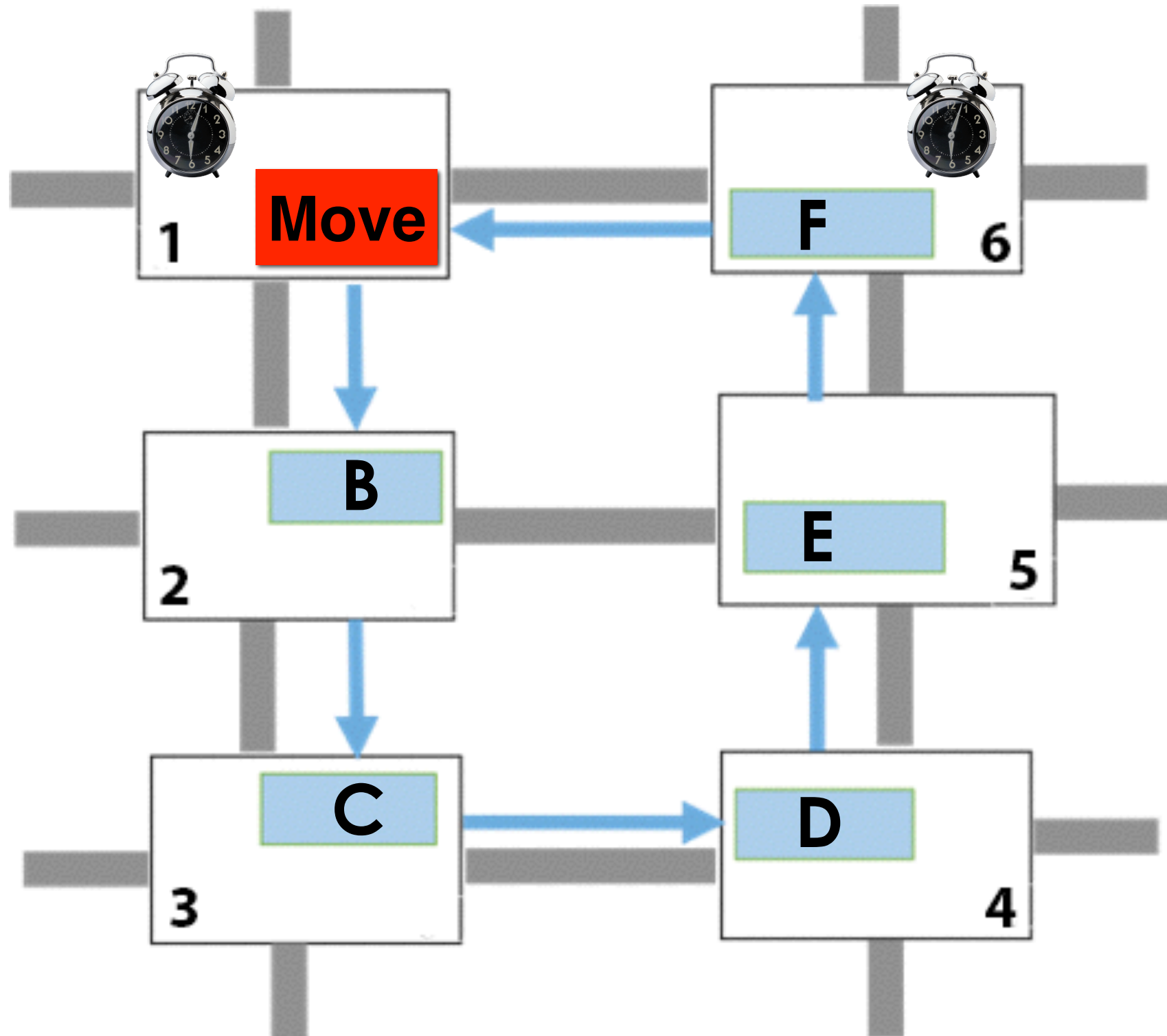


Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

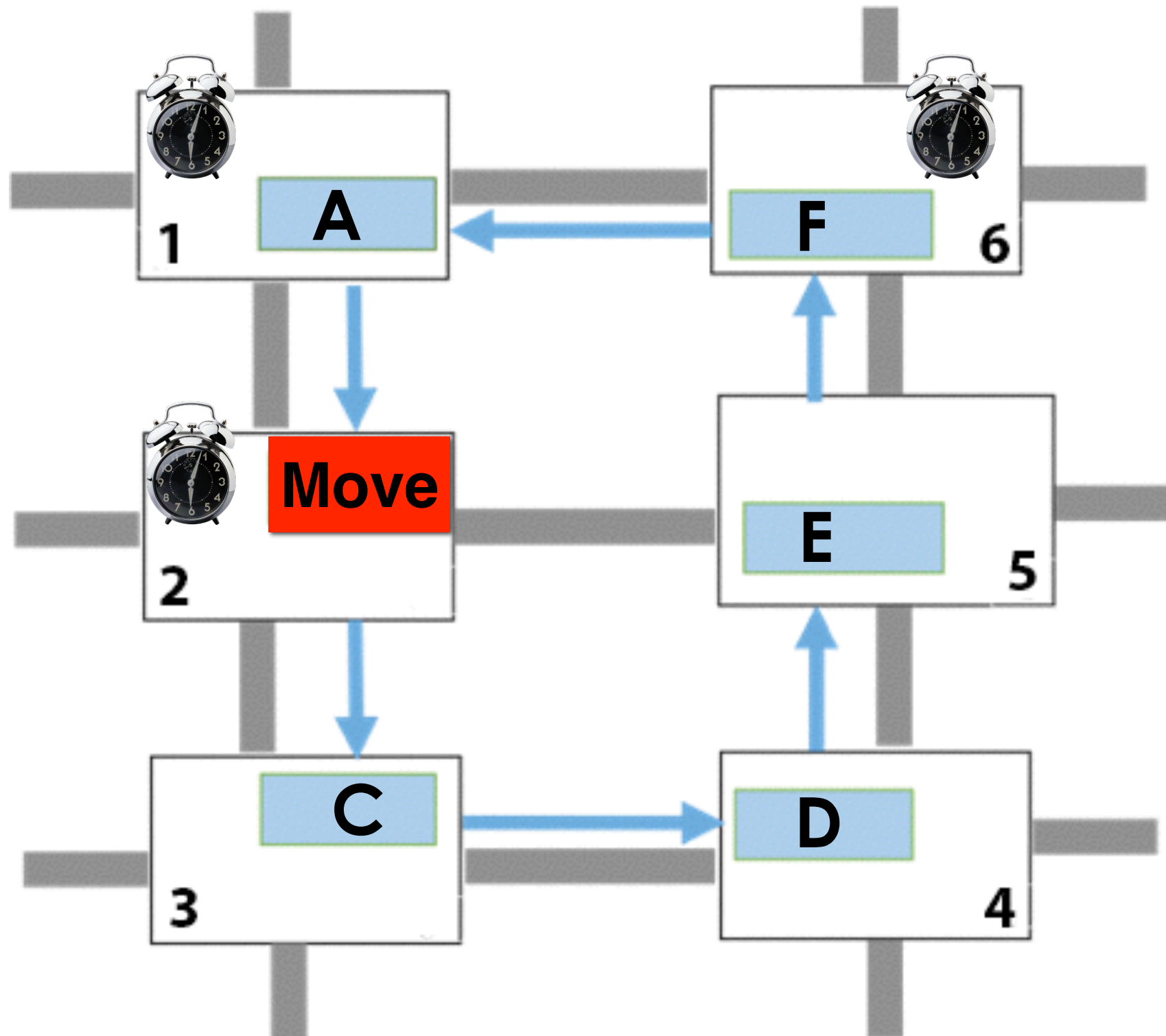


Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

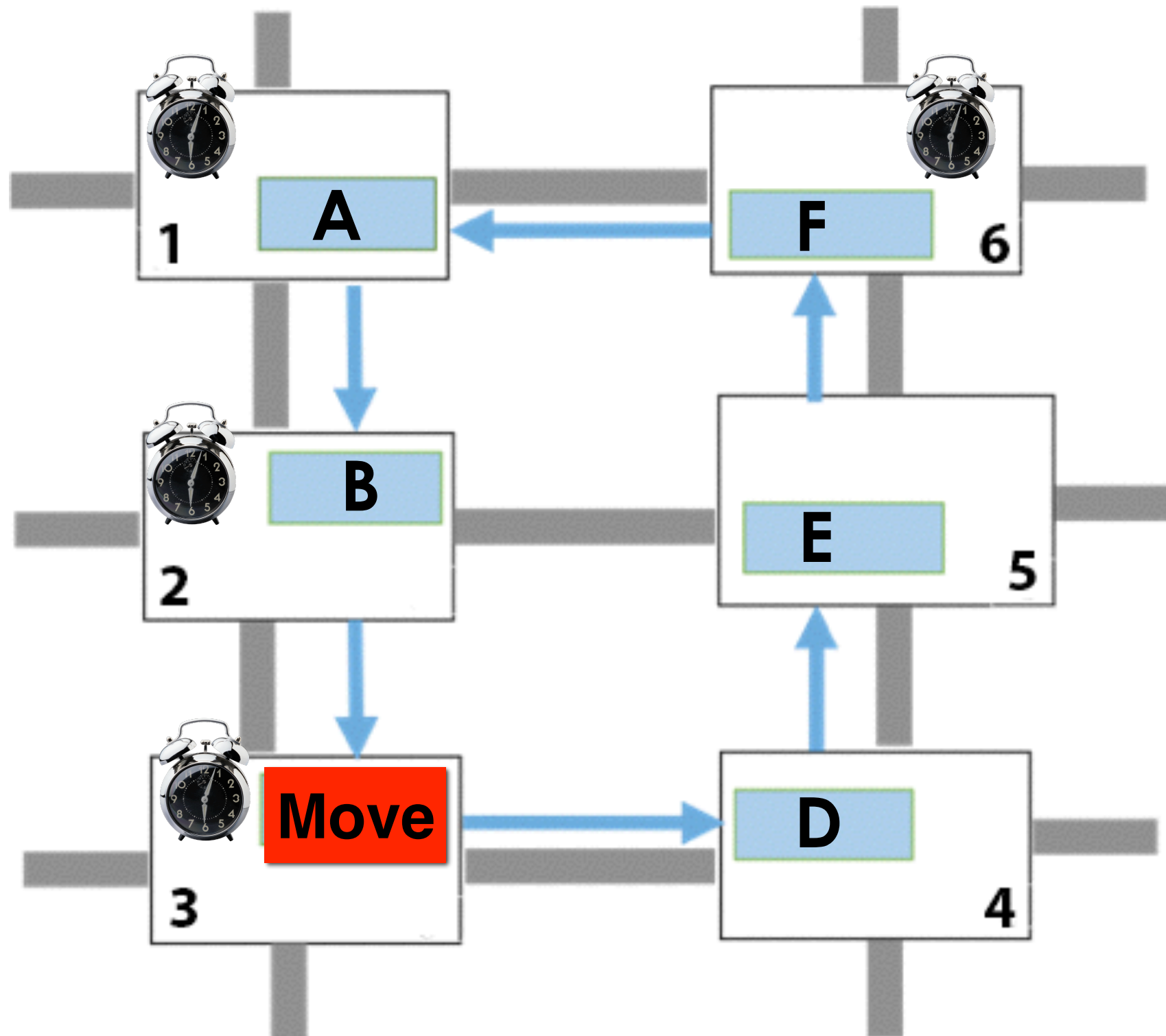


Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

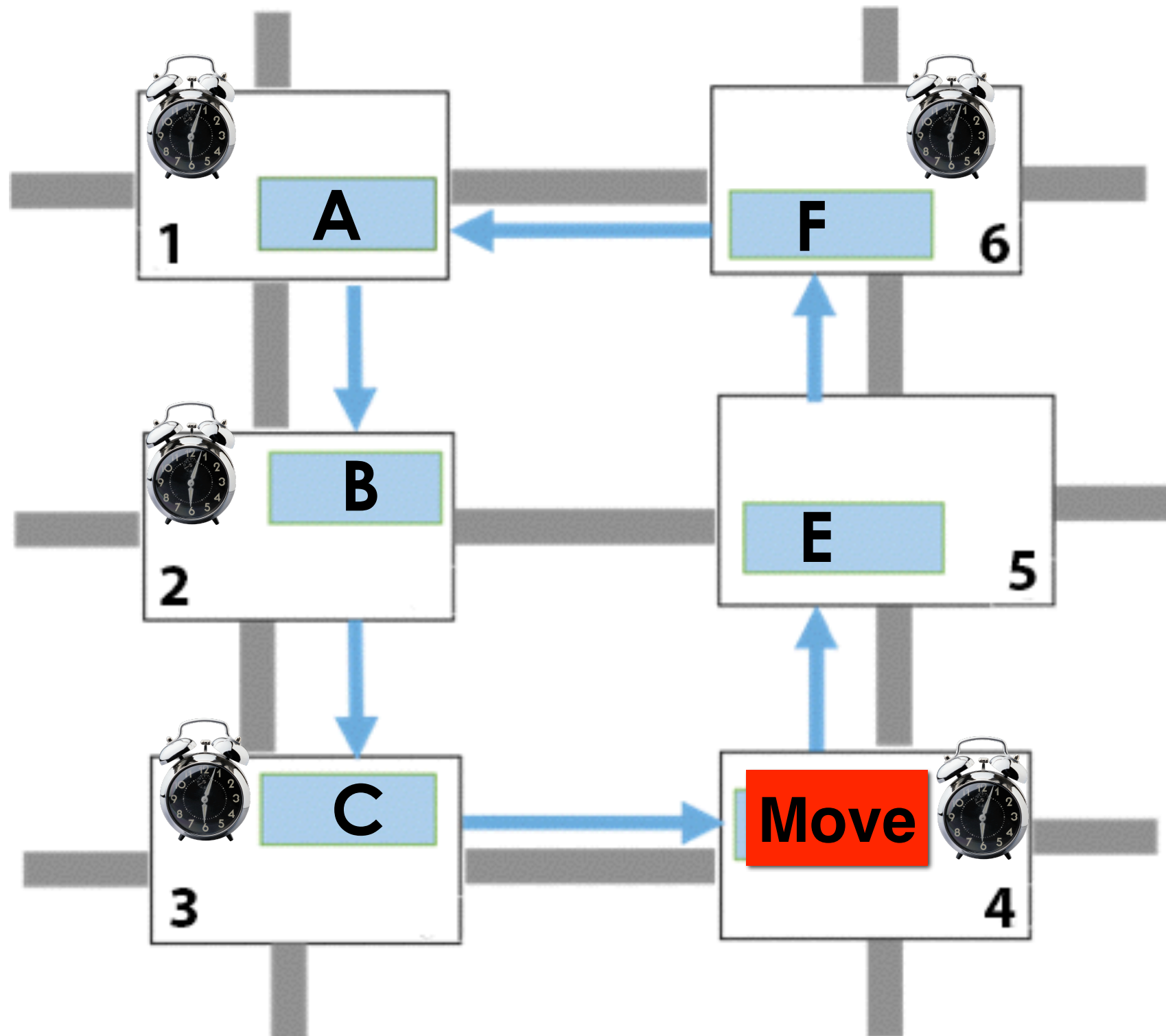


Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

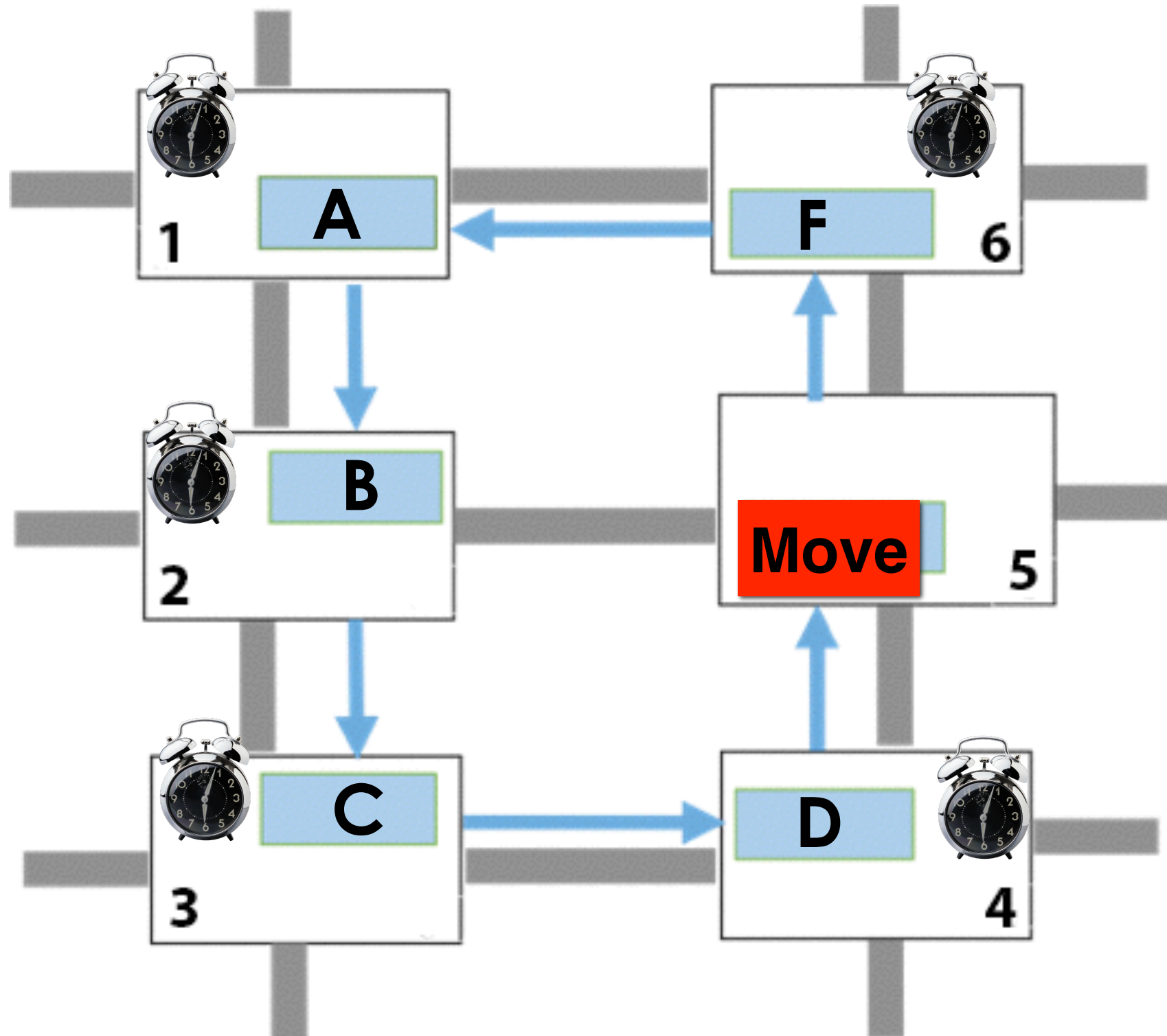


Implementation Example : *Move Msg.*

1. Deadlock
Detection

2. Coordinating
the spin.

3. Executing
the spin.

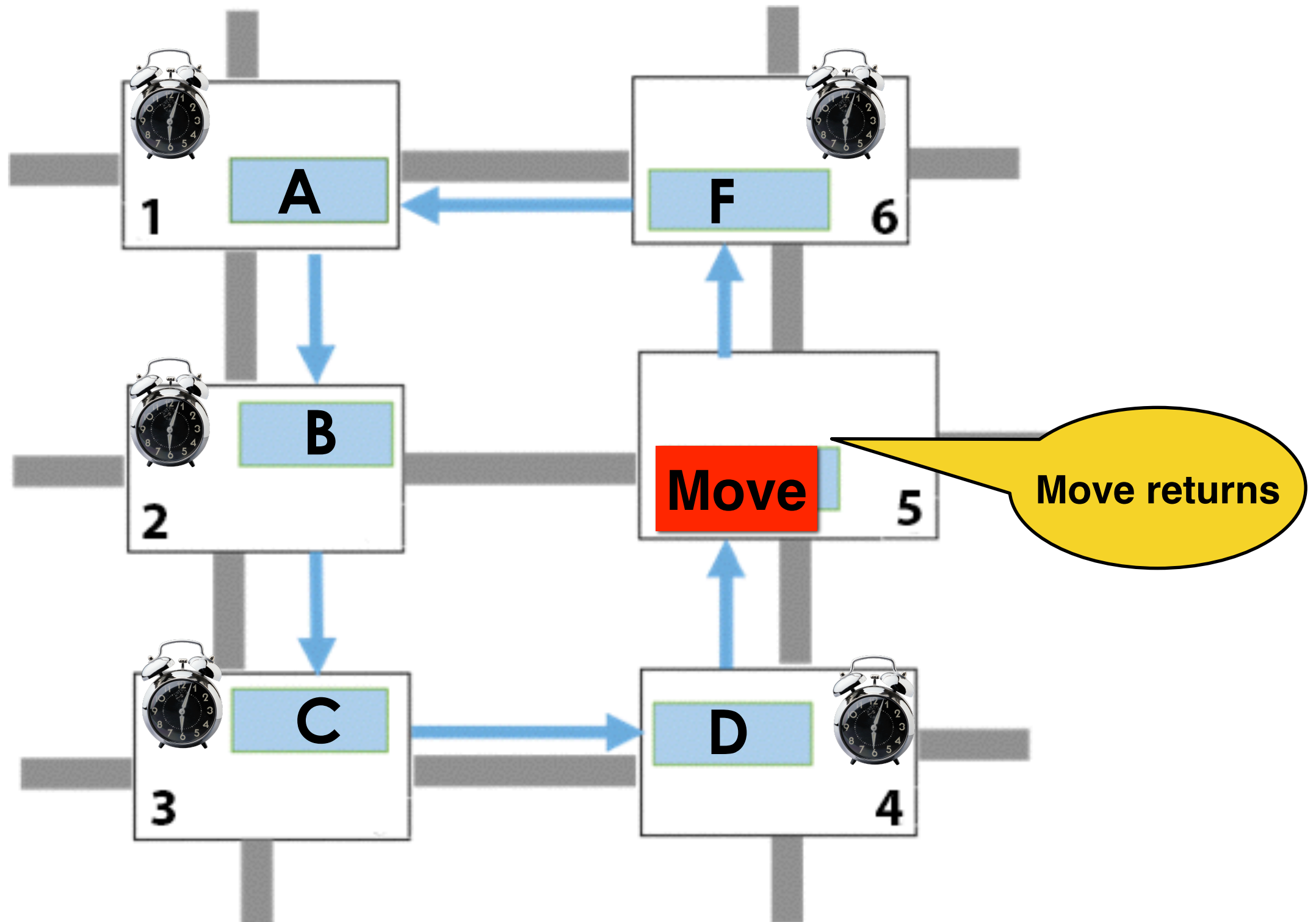


Implementation Example : *Move Msg.*

1. Deadlock
Detection

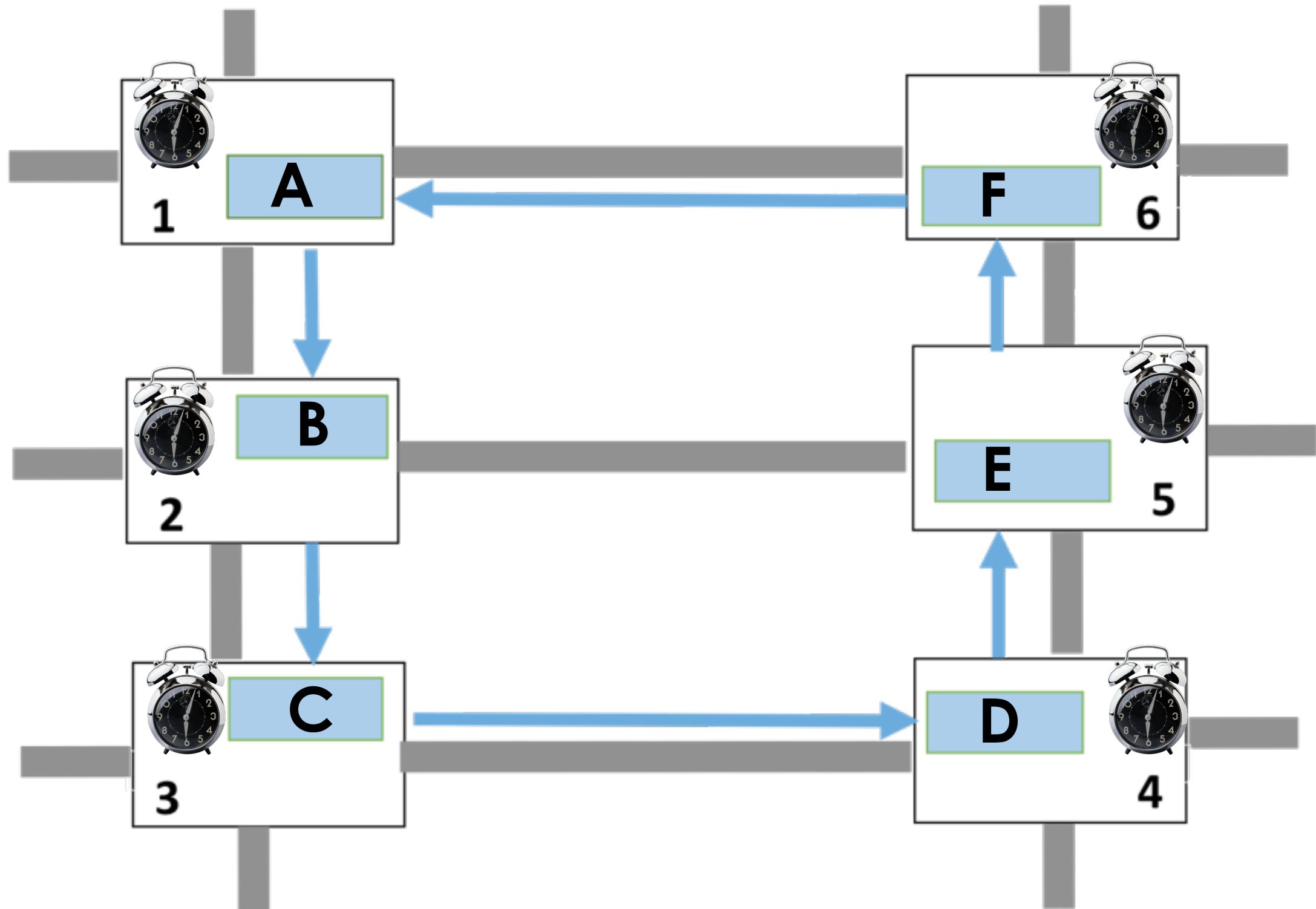
2. Coordinating
the spin.

3. Executing
the spin.



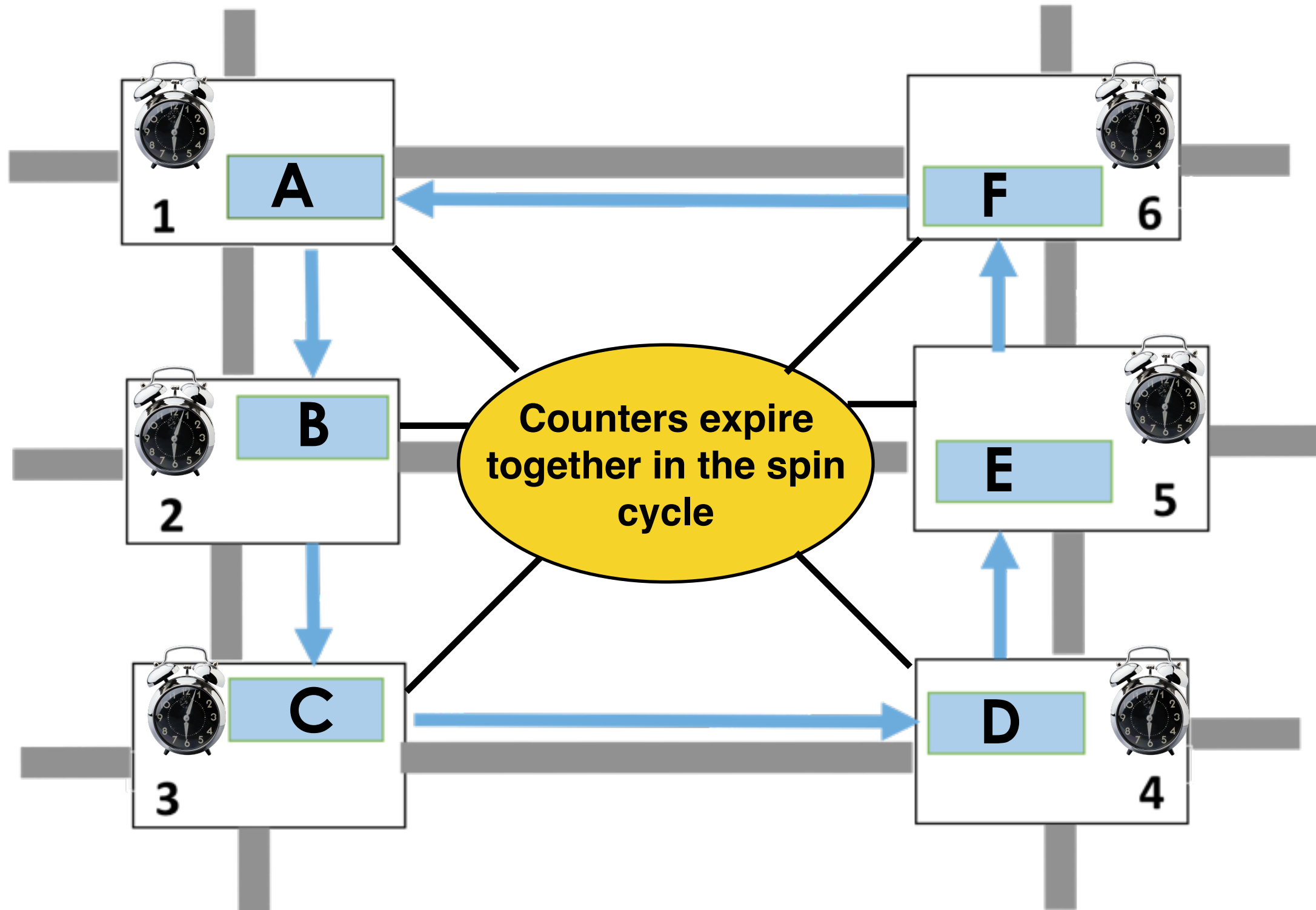
Implementation Example : *spin*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.



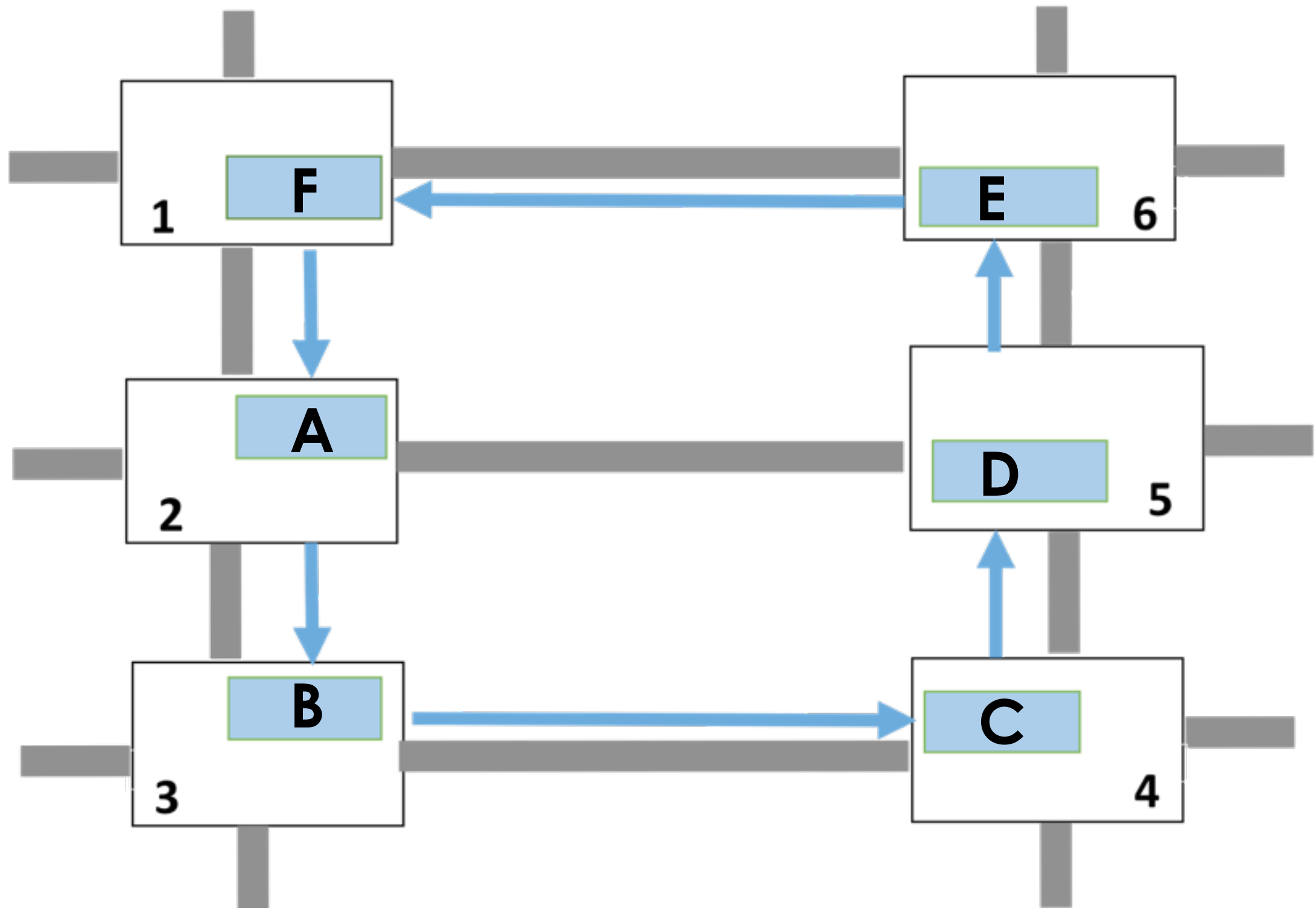
Implementation Example : *spin*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.



Implementation Example : *spin*

1. Deadlock Detection
2. Coordinating the spin.
3. Executing the spin.



Multiple SPIN Optimization

Multiple SPIN Optimization

- Resolving a deadlock may require *multiple spins*

Multiple SPIN Optimization

- Resolving a deadlock may require *multiple spins*
 - After spin, router can resume normal operation.

Multiple SPIN Optimization

- Resolving a deadlock may require *multiple spins*
 - After spin, router can resume normal operation.
 - Counter expires again, process repeated.

Multiple SPIN Optimization

- Resolving a deadlock may require *multiple spins*
 - After spin, router can resume normal operation.
 - Counter expires again, process repeated.
- Optimization: send *probe_move* after spin is complete.

Multiple SPIN Optimization

- Resolving a deadlock may require *multiple spins*
 - After spin, router can resume normal operation.
 - Counter expires again, process repeated.
- Optimization: send *probe_move* after spin is complete.
 - probe_move *checks* if *deadlock still exists* and if so, sets the time for the next spin.

Multiple SPIN Optimization

- Resolving a deadlock may require *multiple spins*
 - After spin, router can resume normal operation.
 - Counter expires again, process repeated.
- Optimization: send *probe_move* after spin is complete.
 - probe_move *checks* if *deadlock still exists* and if so, sets the time for the next spin.
- Details in paper (Sec. IV-B).

Outline

- Routing Deadlocks
 - State of the Art
 - **SPIN** : **S**ynchronized **P**rogress in **I**nterconnection **N**etworks
 - Key Idea
 - Implementation Example
 - Micro-architecture
 - FAvORS
 - Evaluations
 - Conclusion
-

Implementation Micro-architecture

Implementation Micro-architecture

- ***No additional links:*** Spl. Msgs. use the same links as regular flits.

Implementation Micro-architecture

- ***No additional links:*** Spl. Msgs. use the same links as regular flits.
- Spl. Msgs. have higher priority in link usage over regular flits.

Implementation Micro-architecture

- ***No additional links:*** Spl. Msgs. use the same links as regular flits.
- Spl. Msgs. have higher priority in link usage over regular flits.
- Links are anyways idle during deadlocks.

Implementation Micro-architecture

- **No additional links:** Spl. Msgs. use the same links as regular flits.
- Spl. Msgs. have higher priority in link usage over regular flits.
- Links are anyways idle during deadlocks.
- **Bufferless Forwarding:** Spl. Msgs. are not buffered anywhere (either forwarded or dropped).

Implementation Micro-architecture

- **No additional links:** Spl. Msgs. use the same links as regular flits.
- Spl. Msgs. have higher priority in link usage over regular flits.
- Links are anyways idle during deadlocks.
- **Bufferless Forwarding:** Spl. Msgs. are not buffered anywhere (either forwarded or dropped).
- **Distributed Design:** any router can initiate the recovery.

Implementation Micro-architecture

- **No additional links:** Spl. Msgs. use the same links as regular flits.
- Spl. Msgs. have higher priority in link usage over regular flits.
- Links are anyways idle during deadlocks.
- **Bufferless Forwarding:** Spl. Msgs. are not buffered anywhere (either forwarded or dropped).
- **Distributed Design:** any router can initiate the recovery.
- **4% area overhead** compared to traditional mesh router in 15nm Nangate [42].

Outline

- Routing Deadlocks
 - State of the Art
 - **SPIN** : **S**ynchronized **P**rogress in **I**nterconnection **N**etworks
 - Key Idea
 - Walkthrough Example
 - Micro-architecture
 - FAvORS
 - Evaluations
 - Conclusion
-

FAvORS Routing Algorithm

FAvORS Routing Algorithm

- *SPIN* is the *first scheme* that enables *true one-VC* fully adaptive deadlock-free routing for *any topology*.

FAvORS Routing Algorithm

- *SPIN* is the *first scheme* that enables *true one-VC* fully adaptive deadlock-free routing for *any topology*.
- *FAvORS* : *F*ully *A*ddaptive *O*ne-vc *R*outing with *S*PIN.

FAvORS Routing Algorithm

- *SPIN* is the *first scheme* that enables *true one-VC* fully adaptive deadlock-free routing for *any topology*.
- *FAvORS* : *F*ully *A*daptive *O*ne-vc *R*outing with *S*PIN.
- Algorithm has two flavors:

FAvORS Routing Algorithm

- **SPIN** is the *first scheme* that enables *true one-VC* fully adaptive deadlock-free routing for *any topology*.
- **FAvORS** : *F*ully *A*daptive *O*ne-vc *R*outing with *S*PIN.
- Algorithm has two flavors:
 - Minimal Adaptive

FAvORS Routing Algorithm

- *SPIN* is the *first scheme* that enables *true one-VC* fully adaptive deadlock-free routing for *any topology*.
- *FAvORS* : *F*ully *A*daptive *O*ne-vc *R*outing with *S*PIN.
- Algorithm has two flavors:
 - Minimal Adaptive
 - Non-minimal Adaptive.

FAvORS Routing Algorithm

- *SPIN* is the *first scheme* that enables *true one-VC* fully adaptive deadlock-free routing for *any topology*.
- **FAvORS** : **F**ully **A**daptive **O**ne-vc **R**outing with **S**PIN.
- Algorithm has two flavors:
 - Minimal Adaptive
 - Non-minimal Adaptive.
- Route Selection Metrics:

FAvORS Routing Algorithm

- **SPIN** is the **first scheme** that enables **true one-VC** fully adaptive deadlock-free routing for **any topology**.
- **FAvORS** : **F**ully **A**daptive **O**ne-vc **R**outing with **S**PIN.
- Algorithm has two flavors:
 - Minimal Adaptive
 - Non-minimal Adaptive.
- Route Selection Metrics:
 - Credit turn-around time

FAvORS Routing Algorithm

- **SPIN** is the **first scheme** that enables **true one-VC** fully adaptive deadlock-free routing for **any topology**.
- **FAvORS** : **F**ully **A**daptive **O**ne-vc **R**outing with **S**PIN.
- Algorithm has two flavors:
 - Minimal Adaptive
 - Non-minimal Adaptive.
- Route Selection Metrics:
 - Credit turn-around time
 - Hop Count

FAvORS Routing Algorithm

- **SPIN** is the **first scheme** that enables **true one-VC** fully adaptive deadlock-free routing for **any topology**.
- **FAvORS** : **F**ully **A**daptive **O**ne-vc **R**outing with **S**PIN.
- Algorithm has two flavors:
 - Minimal Adaptive
 - Non-minimal Adaptive.
- Route Selection Metrics:
 - Credit turn-around time
 - Hop Count
- More details in paper (Sec. V).

Outline

- Routing Deadlocks
 - State of the Art
 - SPIN : Synchronized Progress in Interconnection Networks
 - Evaluations
 - Conclusion
-

Evaluations

- Network Configuration:



Evaluations

■ Network Configuration:

Simulator	gem5 + Garnet 2.0 Network simulator	
Topologies	8x8 Mesh	
Link Latency	1-cycle	
Traffic	Synthetic + Multi-threaded (PARSEC)	

Evaluations

■ Network Configuration:

Simulator	gem5 + Garnet 2.0 Network simulator	
Topologies	8x8 Mesh	1024 node Off-chip Dragon-fly
Link Latency	1-cycle	Inter-group: 1-cycle Intra-group: 3-cycle
Traffic	Synthetic + Multi-threaded (PARSEC)	Synthetic

Evaluations : Baselines

Evaluations : Baselines

■ 8x8 Mesh:

Design	Routing Adaptivity	Minimal	Theory	Deadlock Freedom Type
West-first Routing	Partial	Yes	Dally	Avoidance
Escape-VC	Full	Yes	Duato	Avoidance
Static-Bubble [6]	Full	Yes	Flow-Control	Recovery

Evaluations : Baselines

■ 8x8 Mesh:

Design	Routing Adaptivity	Minimal	Theory	Deadlock Freedom Type
West-first Routing	Partial	Yes	Dally	Avoidance
Escape-VC	Full	Yes	Duato	Avoidance
Static-Bubble [6]	Full	Yes	Flow-Control	Recovery

■ 1024 Node Off-chip Dragon-fly:

Design	Routing Adaptivity	Minimal	Theory	Deadlock Freedom Type
UGAL [37]	Full	No	Dally	Avoidance

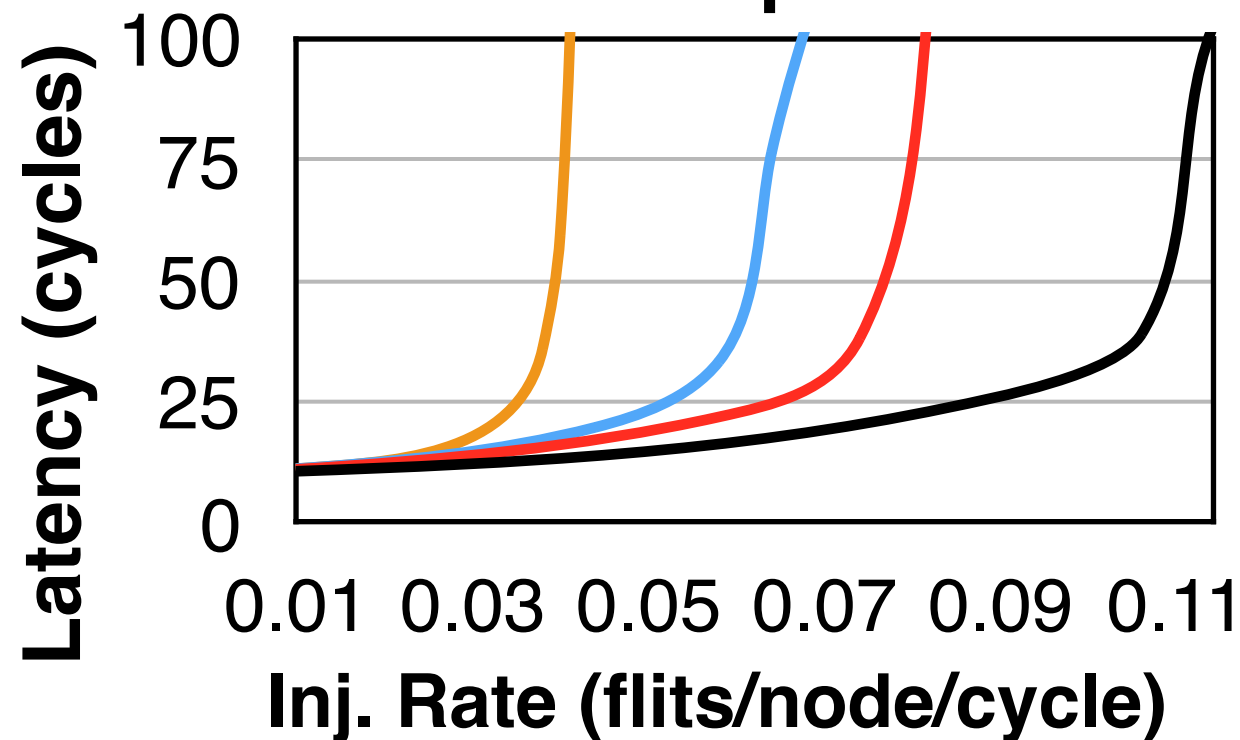
Saturation Throughput

- 1024-node Off-chip Dragon-fly:

Saturation Throughput

- 1024-node Off-chip Dragon-fly:

Bit-complement

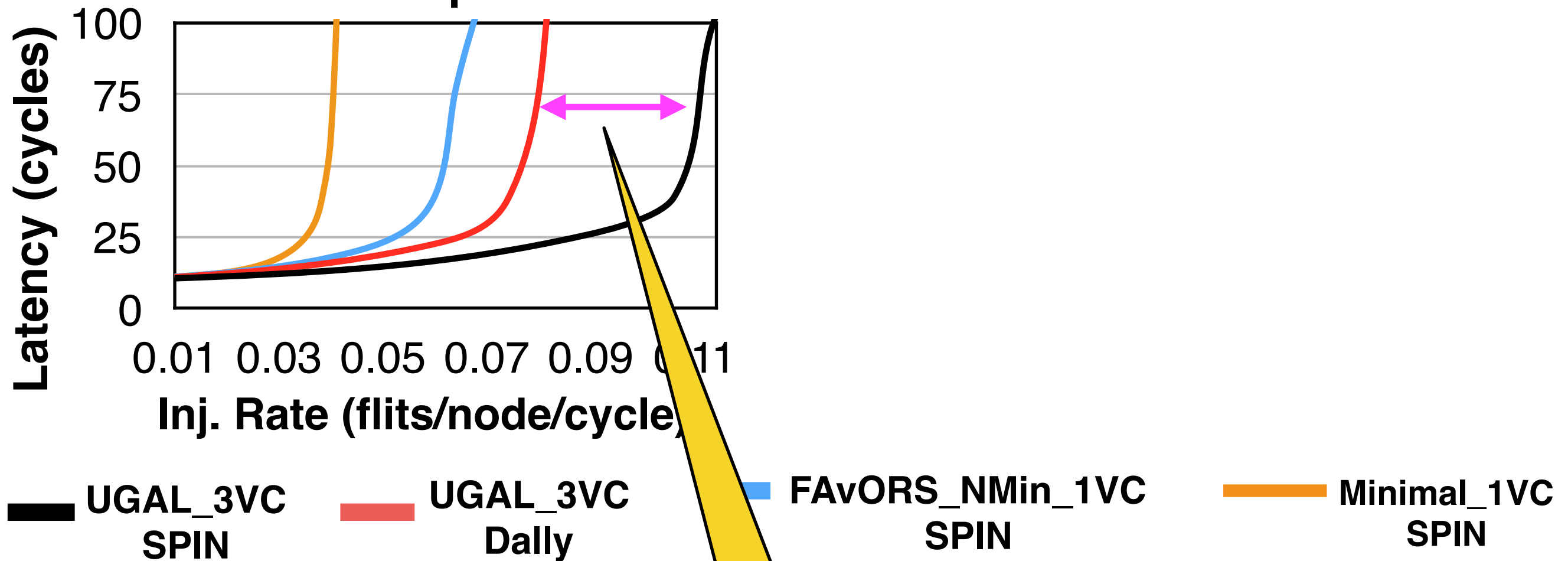


■ UGAL_3VC SPIN ■ UGAL_3VC Dally ■ FAvORS_NMin_1VC SPIN ■ Minimal_1VC SPIN

Saturation Throughput

- 1024-node Off-chip Dragon-fly:

Bit-complement

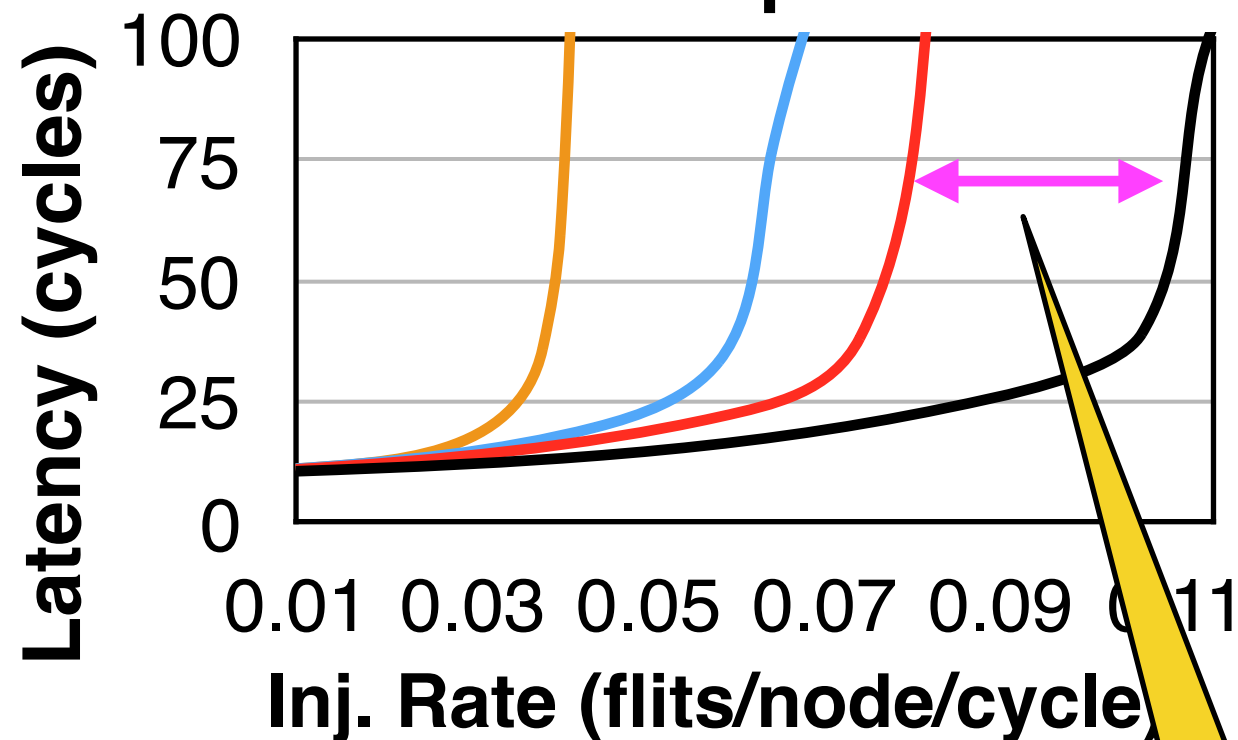


50% higher throughput
compared to UGAL_Dally

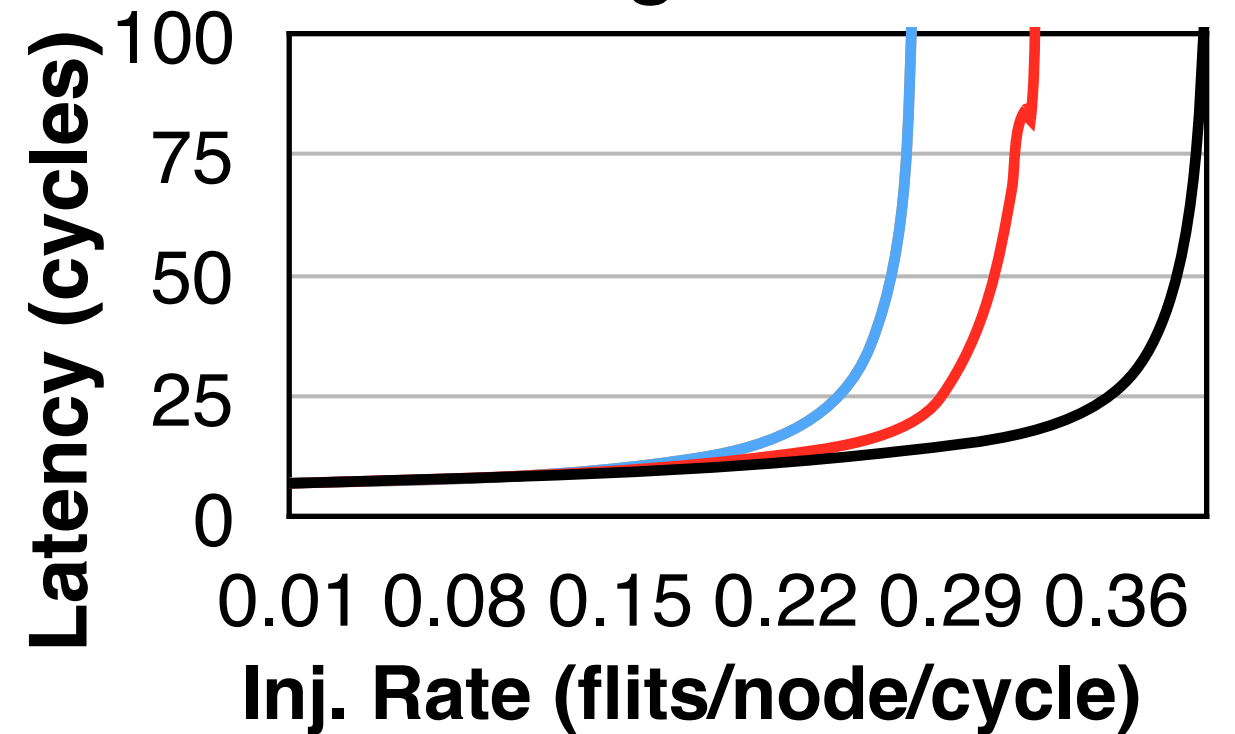
Saturation Throughput

- 1024-node Off-chip Dragon-fly:

Bit-complement



Neighbor



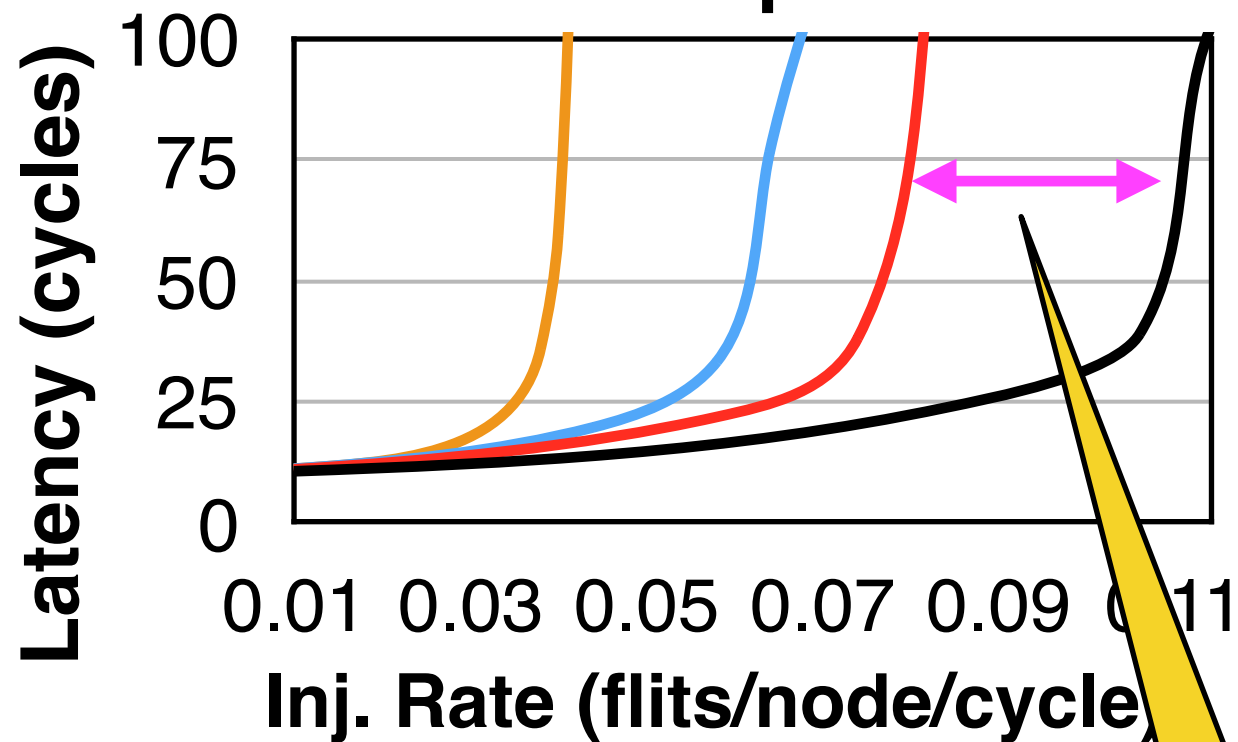
■ UGAL_3VC SPIN ■ UGAL_3VC Dally ■ FAvORS_NMin_1VC SPIN ■ Minimal_1VC SPIN

50% higher throughput compared to UGAL_Dally

Saturation Throughput

1024-node Off-chip Dragon-fly:

Bit-complement



UGAL_3VC
SPIN

UGAL_3VC
Dally

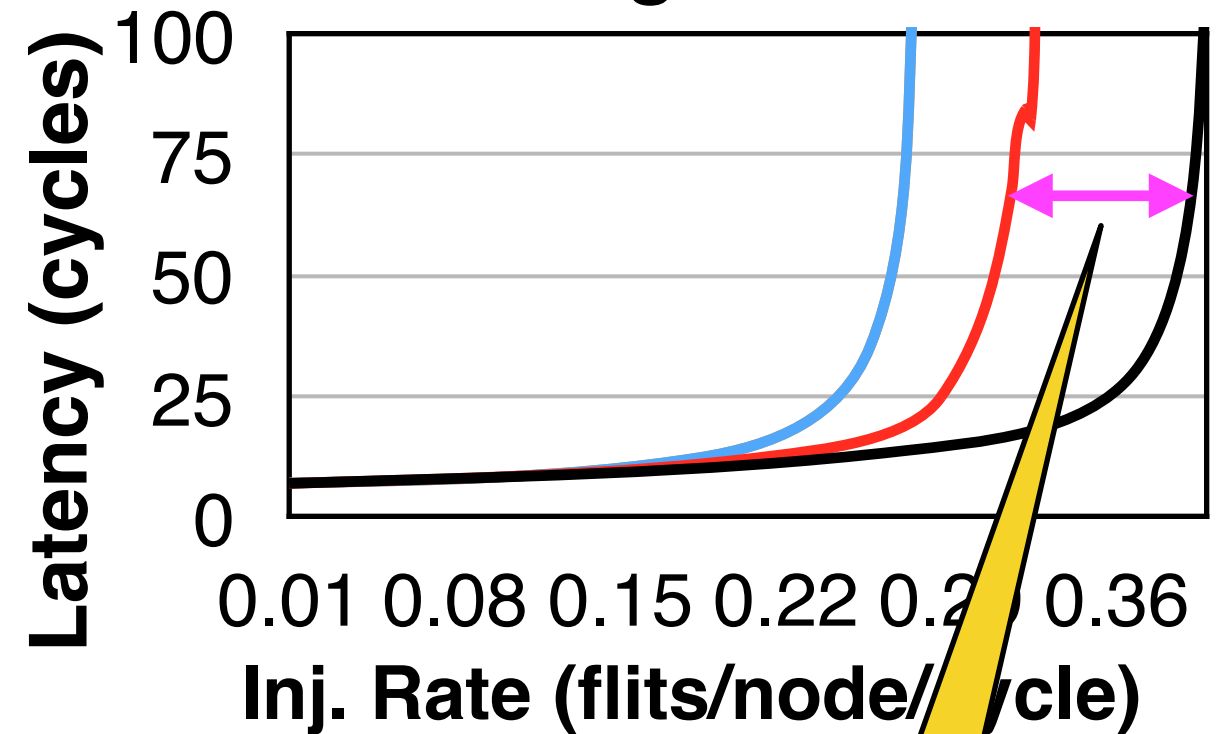
FAvORS_NMin_1VC
SPIN

Minimal_1VC
SPIN

50% higher throughput
compared to UGAL_Dally

25% higher throughput
compared to UGAL_Dally

Neighbor

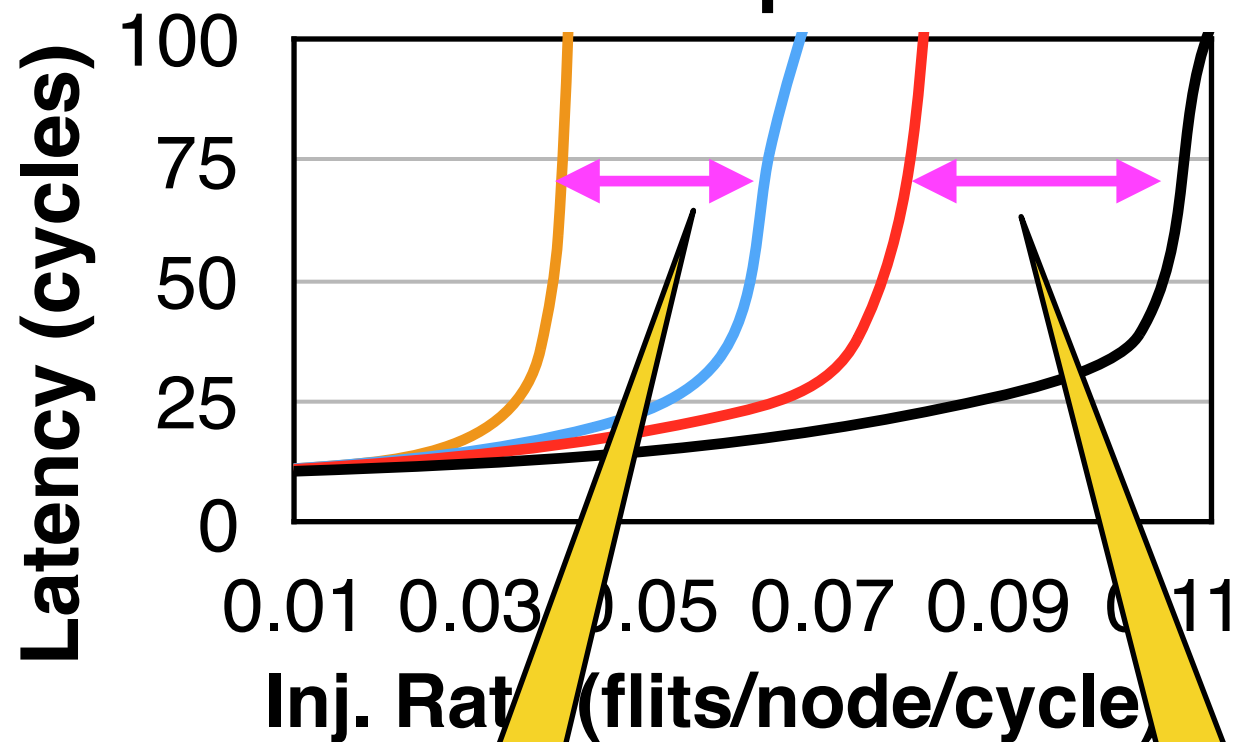


25% higher throughput
compared to UGAL_Dally

Saturation Throughput

1024-node Off-chip Dragon-fly:

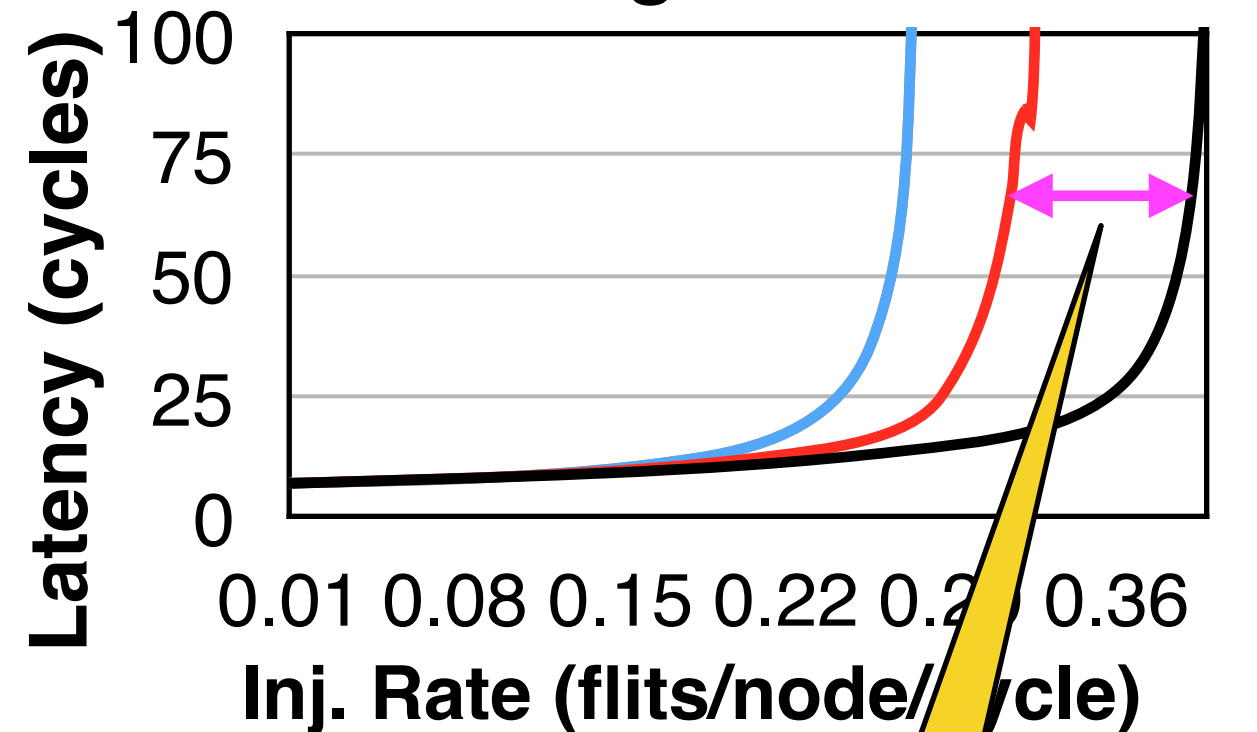
Bit-complement



62% higher
throughput compared to
Minimal Routing 1-VC

50% higher throughput
compared to UGAL_Dally

Neighbor



25% higher throughput
compared to UGAL_Dally

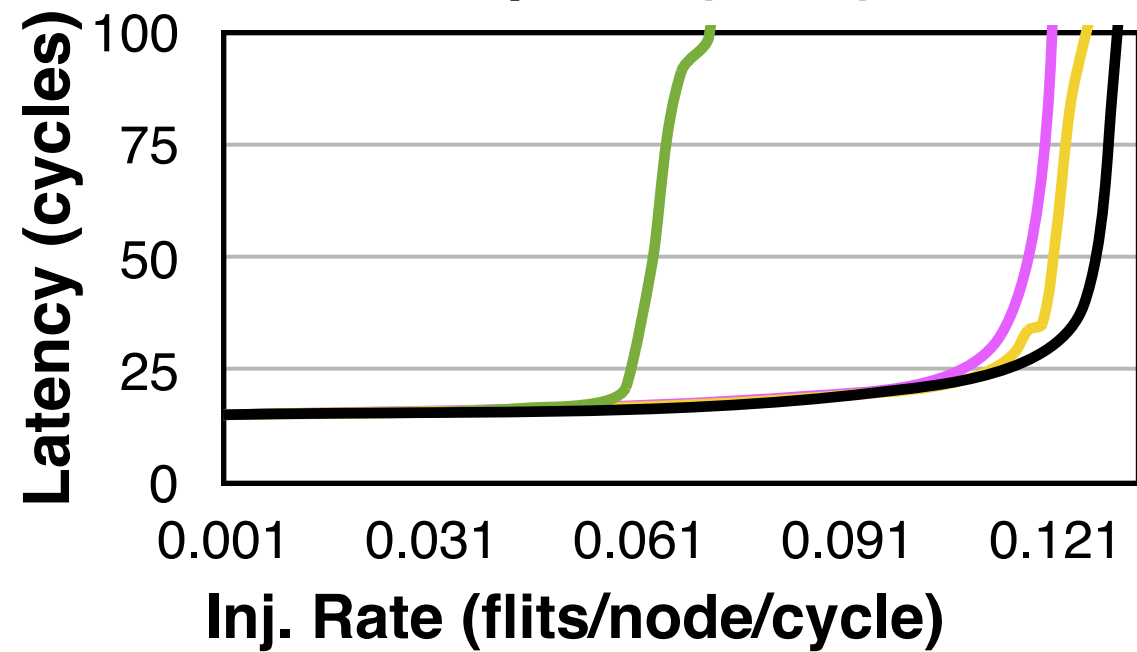
Saturation Throughput

- 8x8 On-chip Mesh:

Saturation Throughput

8x8 On-chip Mesh:

Transpose (3-VC)



West-First_3VC
Dally

Static_Bubble_3VC
Flow-Control

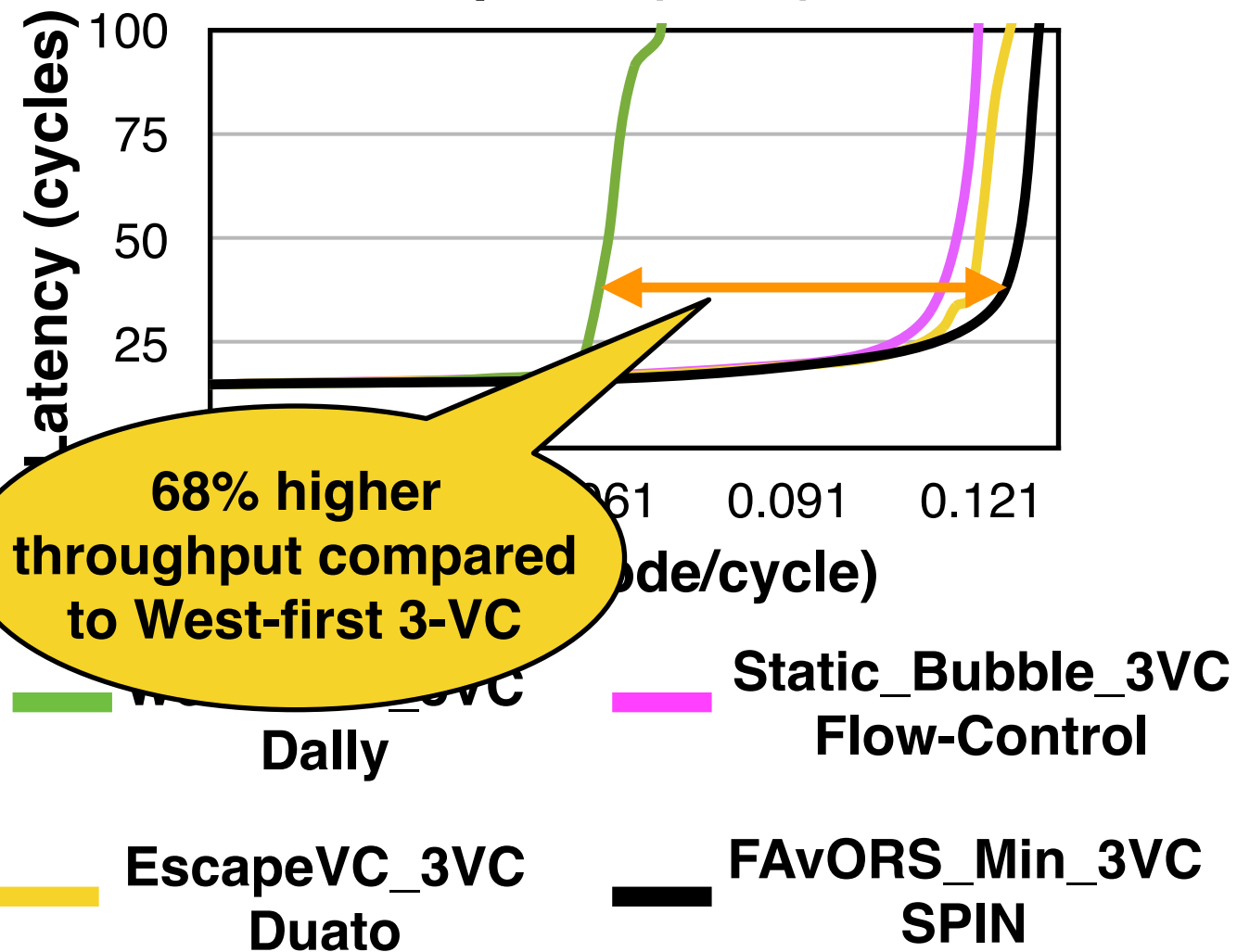
EscapeVC_3VC
Duato

FAvORS_Min_3VC
SPIN

Saturation Throughput

8x8 On-chip Mesh:

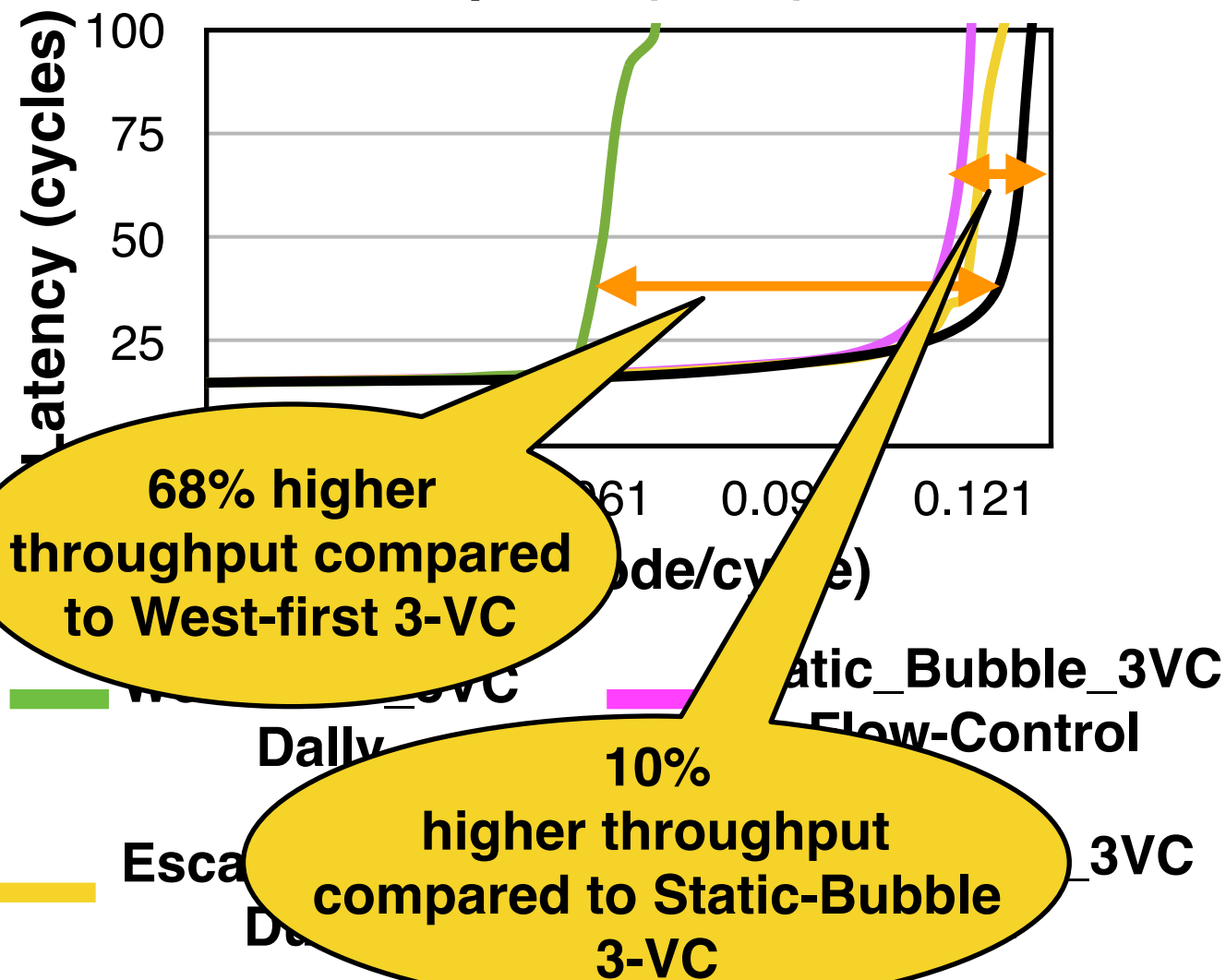
Transpose (3-VC)



Saturation Throughput

8x8 On-chip Mesh:

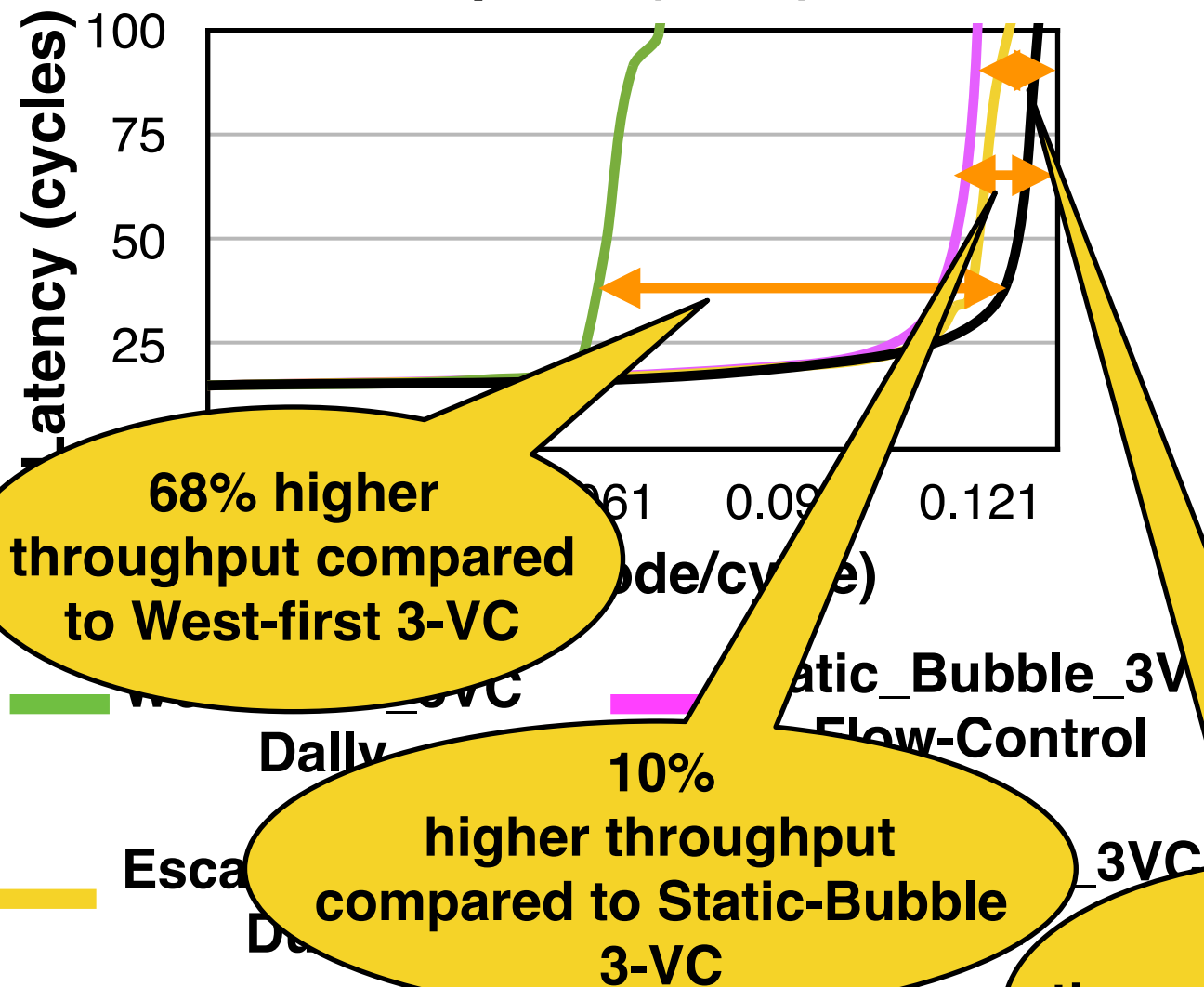
Transpose (3-VC)



Saturation Throughput

■ 8x8 On-chip Mesh:

Transpose (3-VC)



68% higher throughput compared to West-first 3-VC

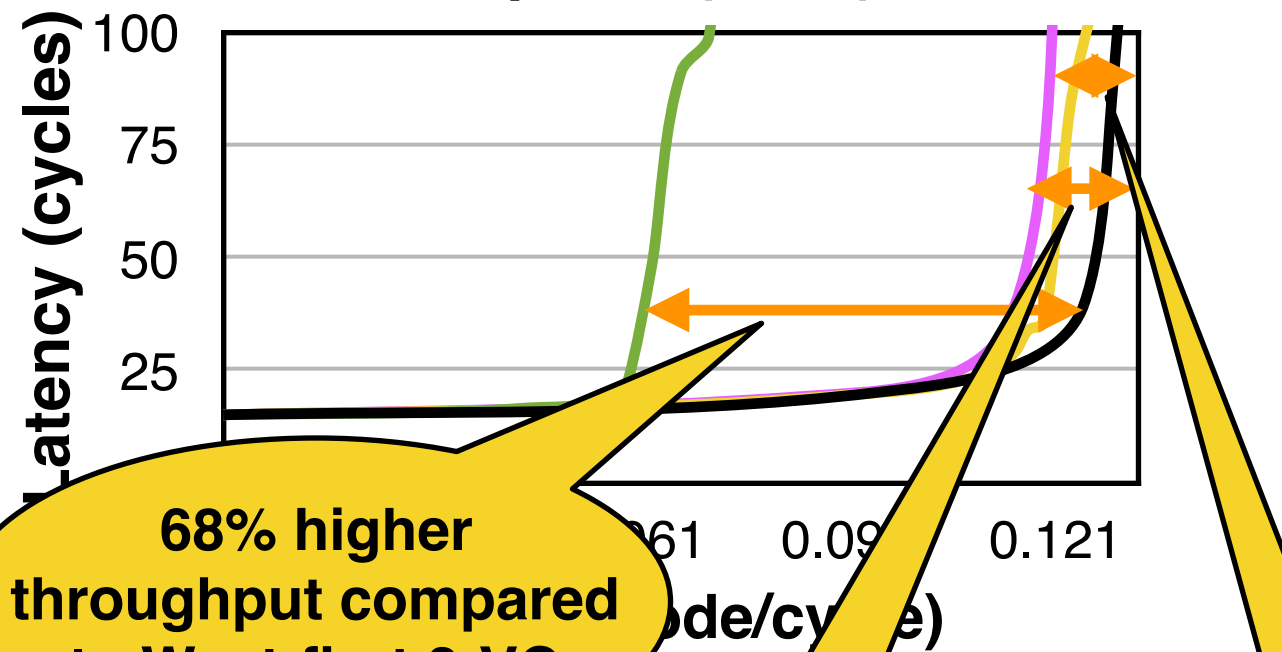
10% higher throughput compared to Static-Bubble 3-VC

8% higher throughput compared to Escape-VC 3-VC

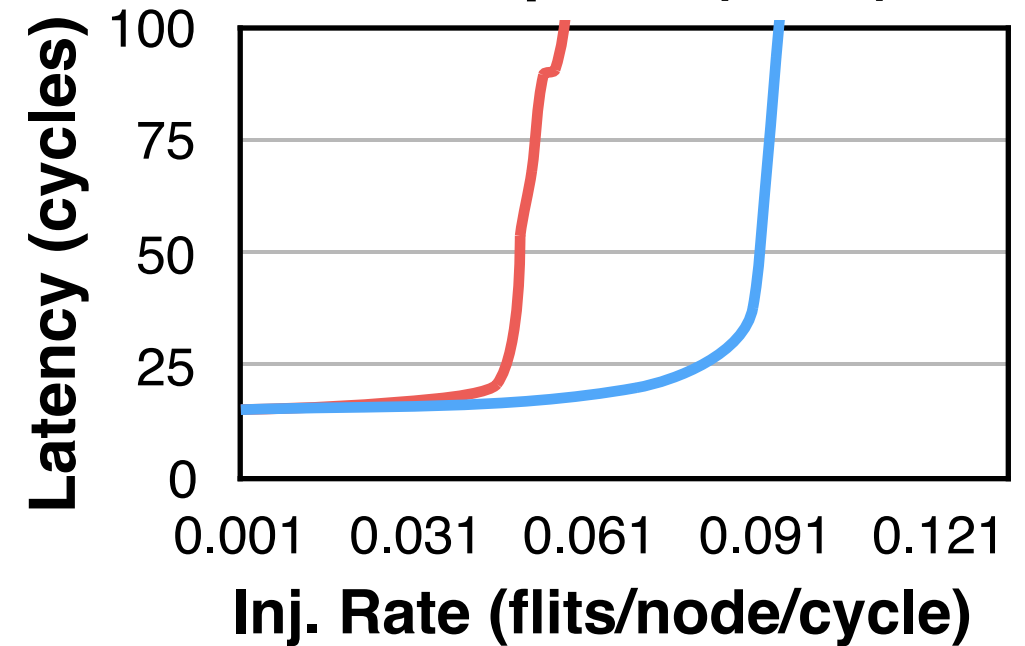
Saturation Throughput

8x8 On-chip Mesh:

Transpose (3-VC)



Transpose (1-VC)



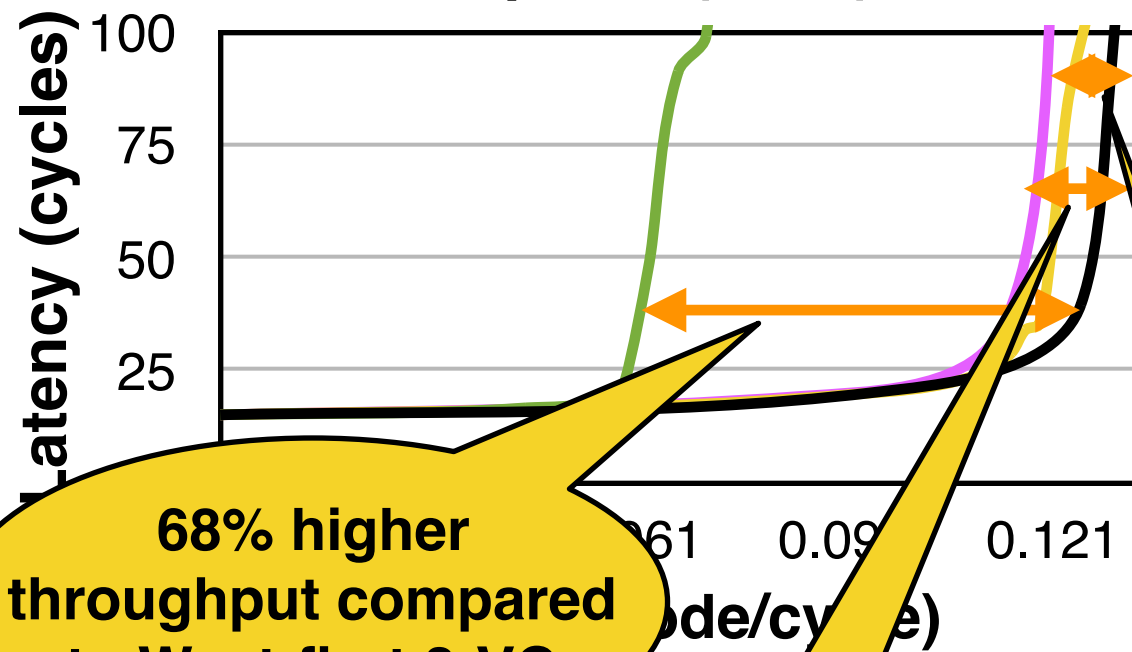
West-First_1VC
Dally

FAvORS_Min_1VC
SPIN

Saturation Throughput

8x8 On-chip Mesh:

Transpose (3-VC)

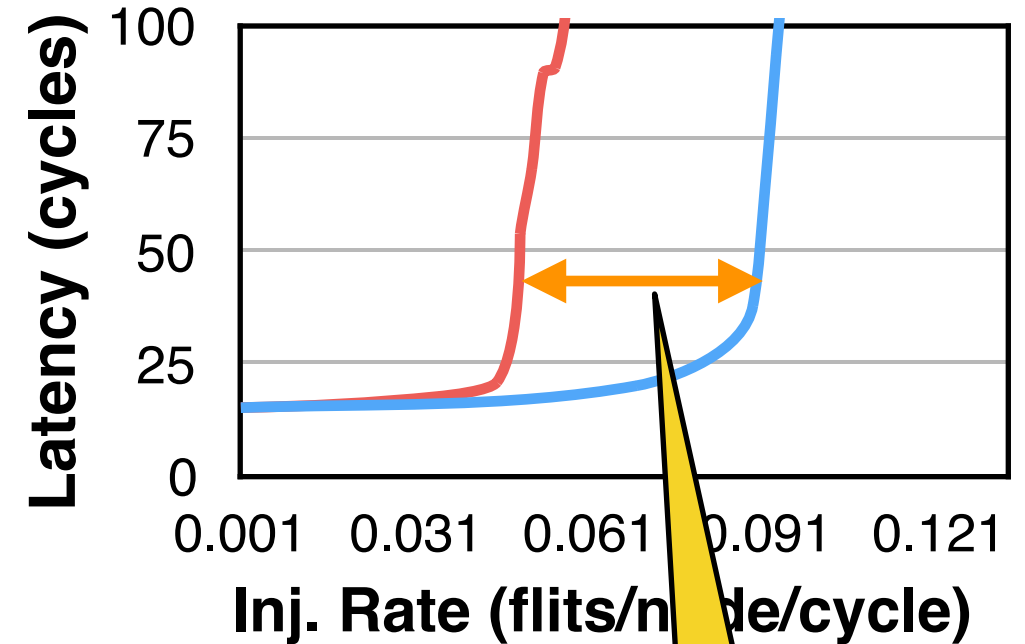


68% higher throughput compared to West-first 3-VC

10% higher throughput compared to Static-Bubble 3-VC

8% higher throughput compared to Escape-VC 3-VC

Transpose (1-VC)



80% higher throughput compared to West-First 1-VC

Conclusion

Conclusion

- *Deadlocks* are a *fundamental problem* in Interconnection Networks.

Conclusion

- *Deadlocks* are a *fundamental problem* in Interconnection Networks.
 - *SPIN* is a generic deadlock freedom theory
 - *Scalable*: Distributed Deadlock Resolution
 - *Plug-n-Play*: topology agnostic
 - Enables *true one-VC fully adaptive routing* for any topology
-

Conclusion

- **Deadlocks** are a **fundamental problem** in Interconnection Networks.
- **SPIN** is a generic deadlock freedom theory
 - **Scalable:** Distributed Deadlock Resolution
 - **Plug-n-Play:** topology agnostic
 - Enables **true one-VC fully adaptive routing** for any topology
 - Performance (Saturation Throughput):
 - On-chip mesh: upto **68% higher**
 - Off-chip Dragon-fly: upto **62% higher**

Conclusion

- Practical Applications:

Conclusion

■ Practical Applications:

On-Chip	Mesh (Intel SCC, Tiler Tile64)
Super-computers	Dragon-fly (Cray XC Networks)
Datacenters	JellyFish (HP), Fat Tree (Google)
Irregular Topologies	Faults (Static Bubble [6]) Power-gating (Router Parking[29])
NoC Generators	FlexNoc (ARTERIS), Sonics GN
Domain specific Accelerators	Corelink Interconnect (ARM)

Conclusion

■ Practical Applications:

On-Chip	Mesh (Intel SCC, Tiler Tile64)
Super-computers	Dragon-fly (Cray XC Networks)
Datacenters	JellyFish (HP), Fat Tree (Google)
Irregular Topologies	Faults (Static Bubble [6]) Power-gating (Router Parking[29])
NoC Generators	FlexNoc (ARTERIS), Sonics GN
Domain specific Accelerators	Corelink Interconnect (ARM)

■ Thank you !!

Back-up

SPIN : Applications

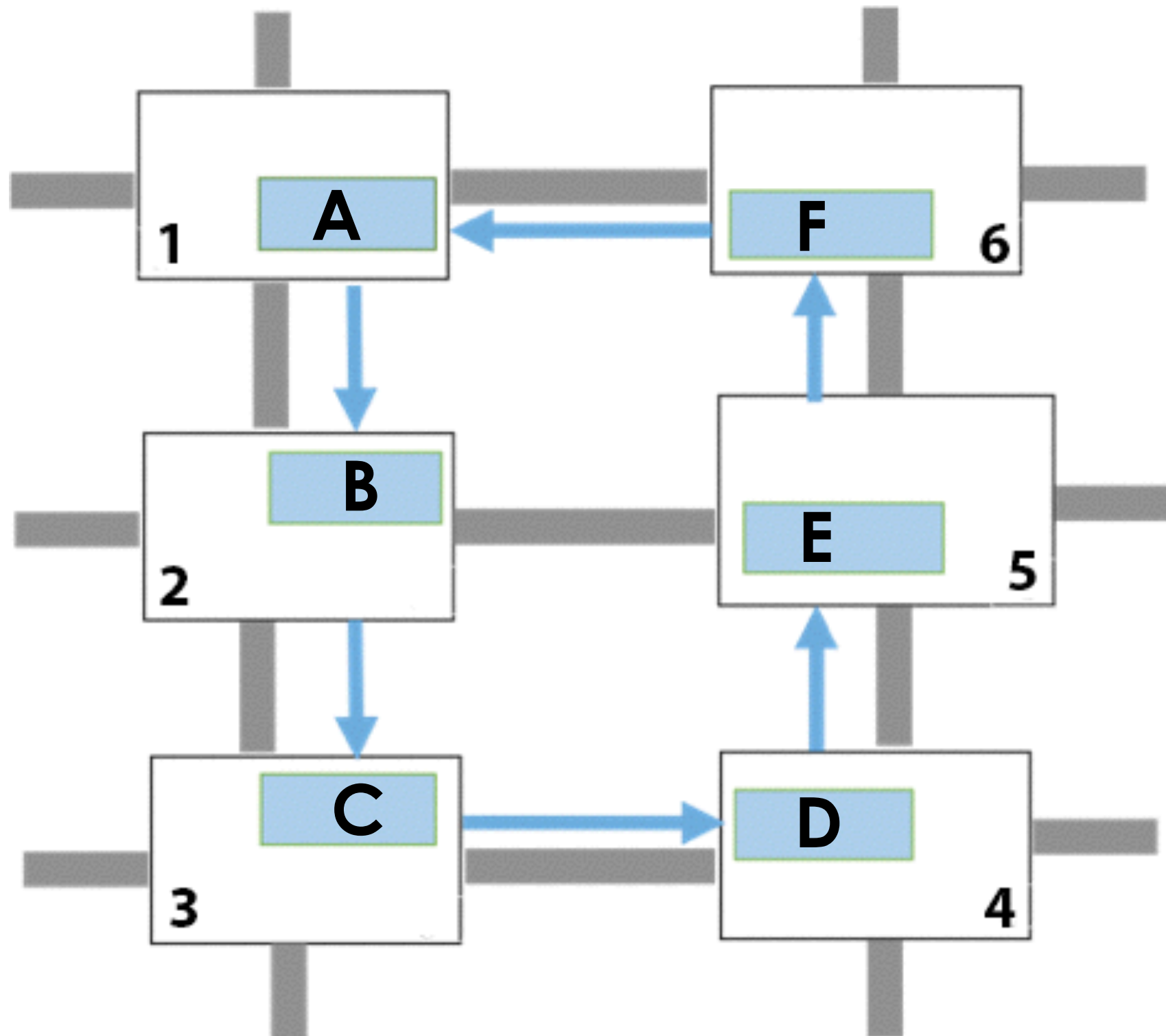
SPIN : Applications

- SPIN is a **generic** deadlock freedom theory
 - **Scalable**: distributed deadlock resolution
 - **Plug-n-Play**: doesn't require knowledge of topology

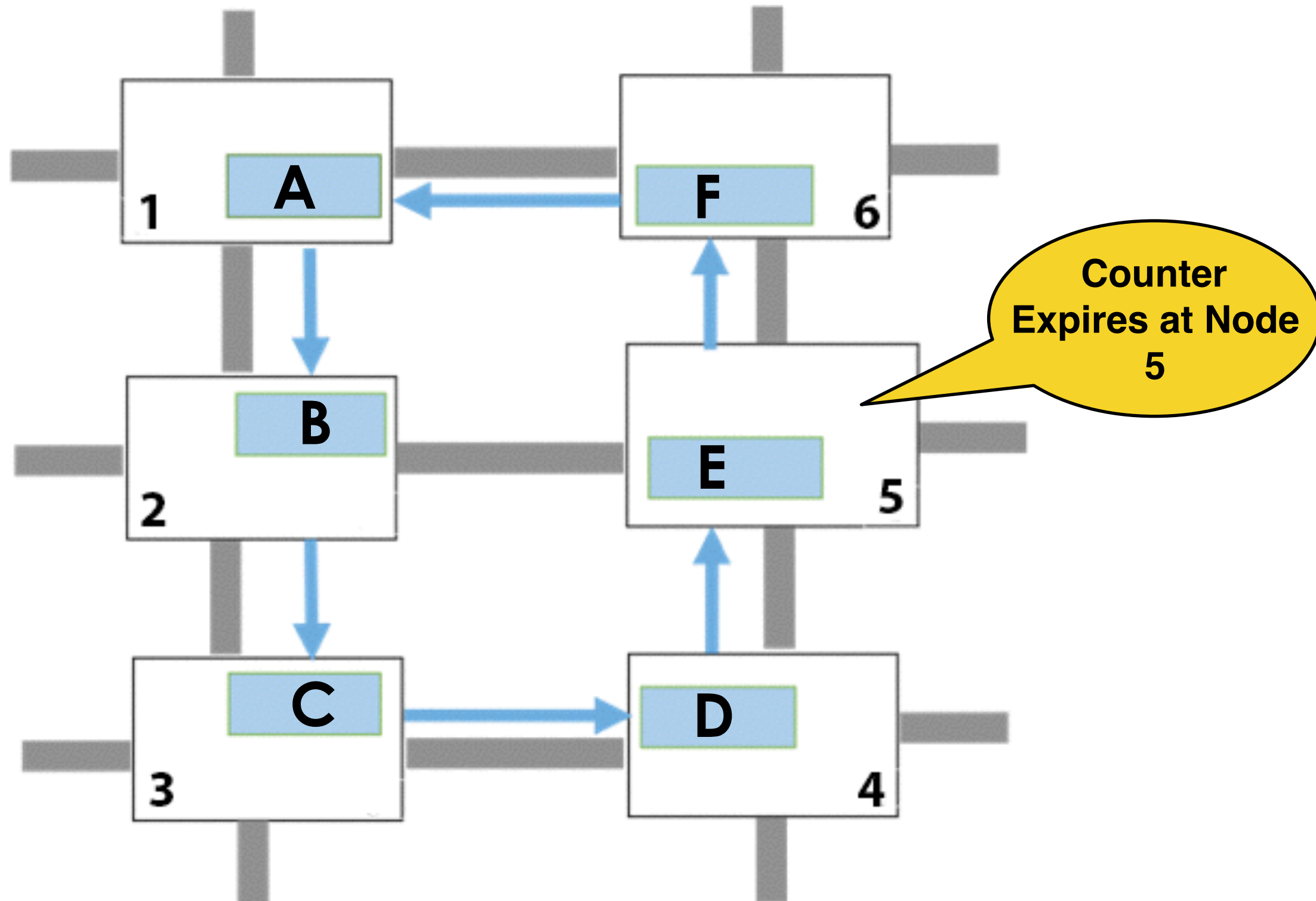
SPIN : Applications

- SPIN is a **generic** deadlock freedom theory
 - **Scalable**: distributed deadlock resolution
 - **Plug-n-Play**: doesn't require knowledge of topology
- SPIN can thus be used in :
 - **On-chip networks**: Mesh (Intel SCC, Tiler Tile64)
 - **Supercomputers**: Dragon-fly (Cray XC Networks)
 - **Datacenters**: Jellyfish (HP), Fat Tree (Google)
 - Static & Dynamically Changing **Irregular topologies** due to faults (Static Bubble [6]) & power-gating (Router Parking [29])
 - **NoC Generators** (Opensmart [13]) & Domain specific **accelerator** (Eyeriss[15])

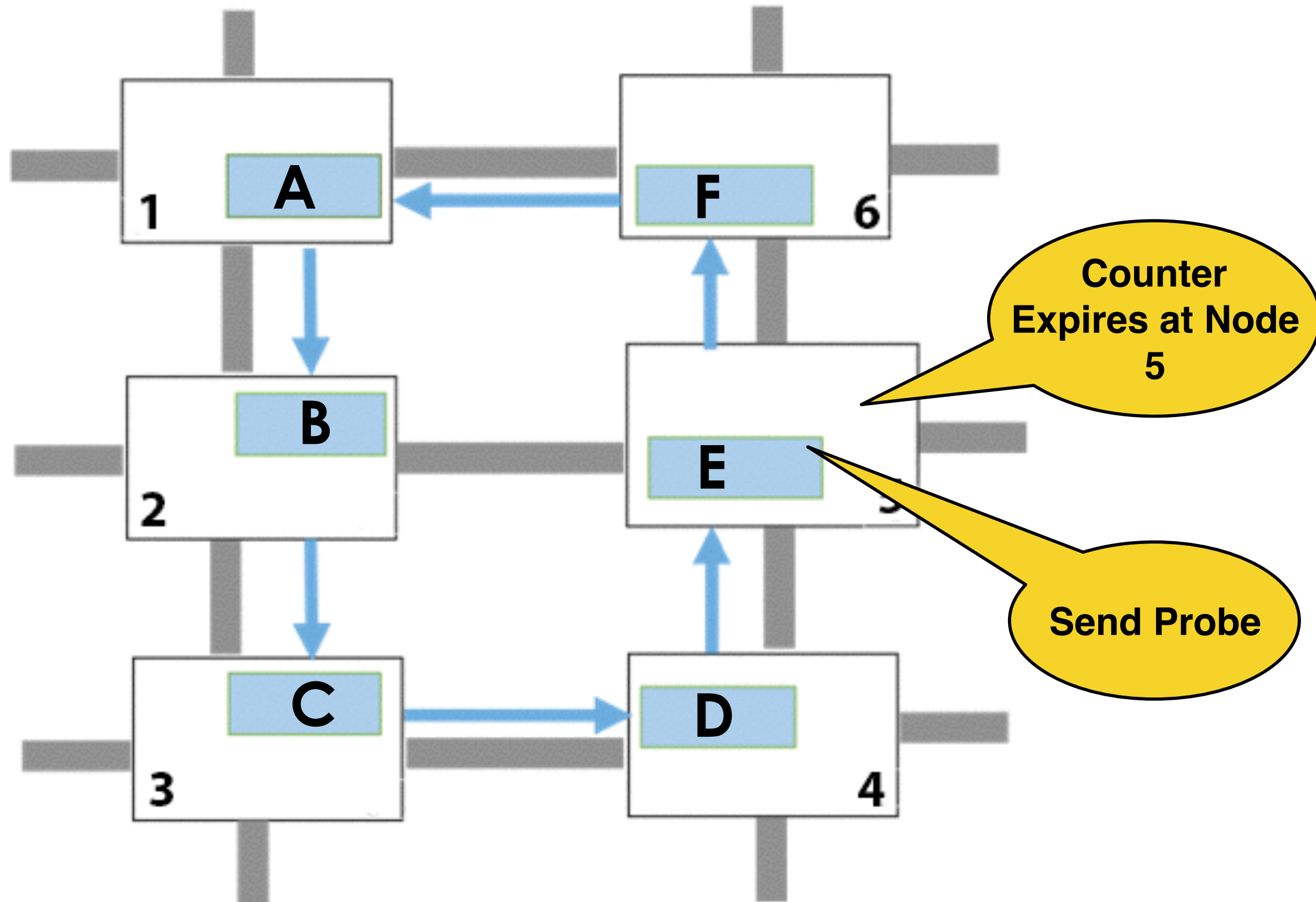
Implementation Example : *Probe Msg.*



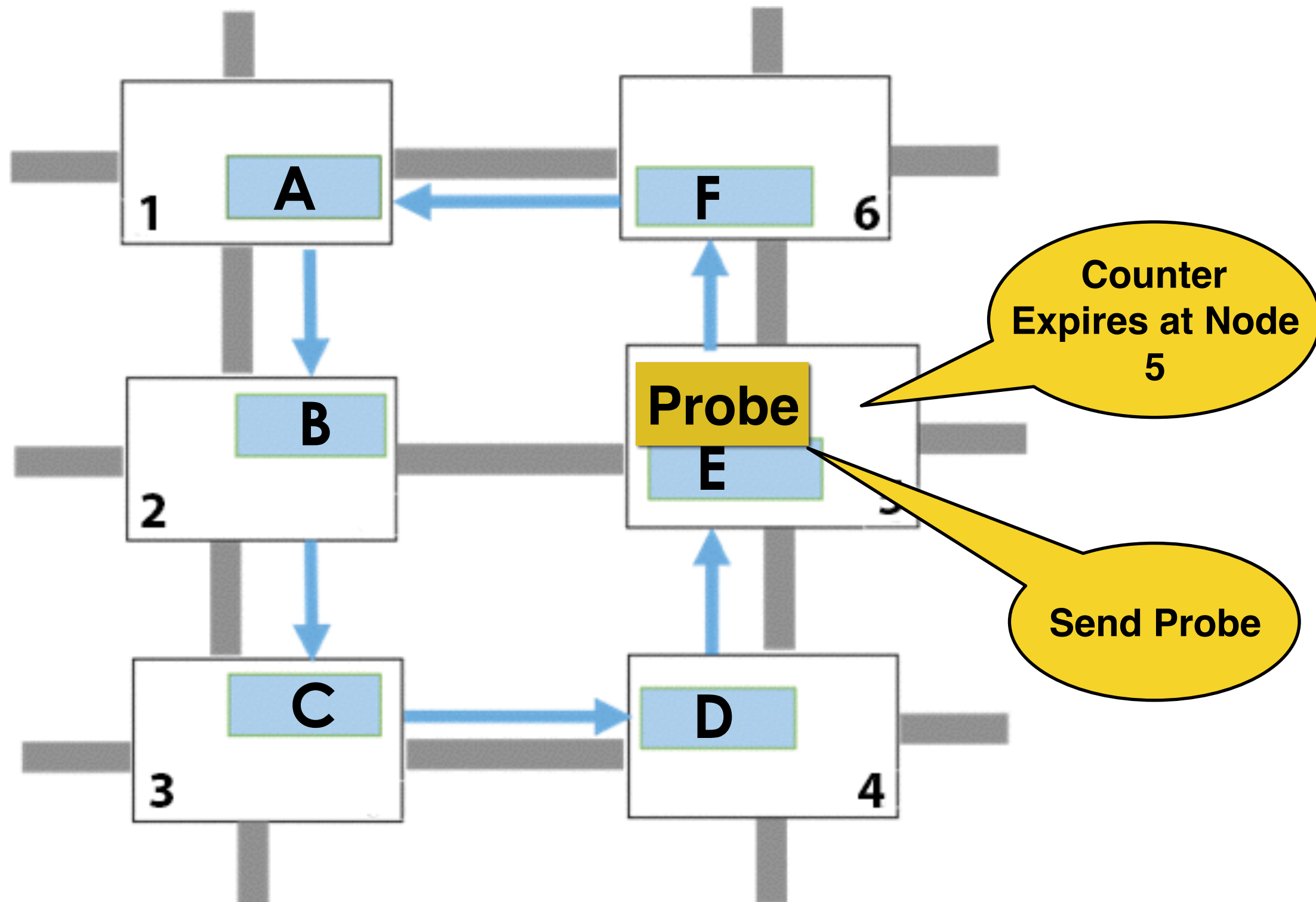
Implementation Example : *Probe Msg.*



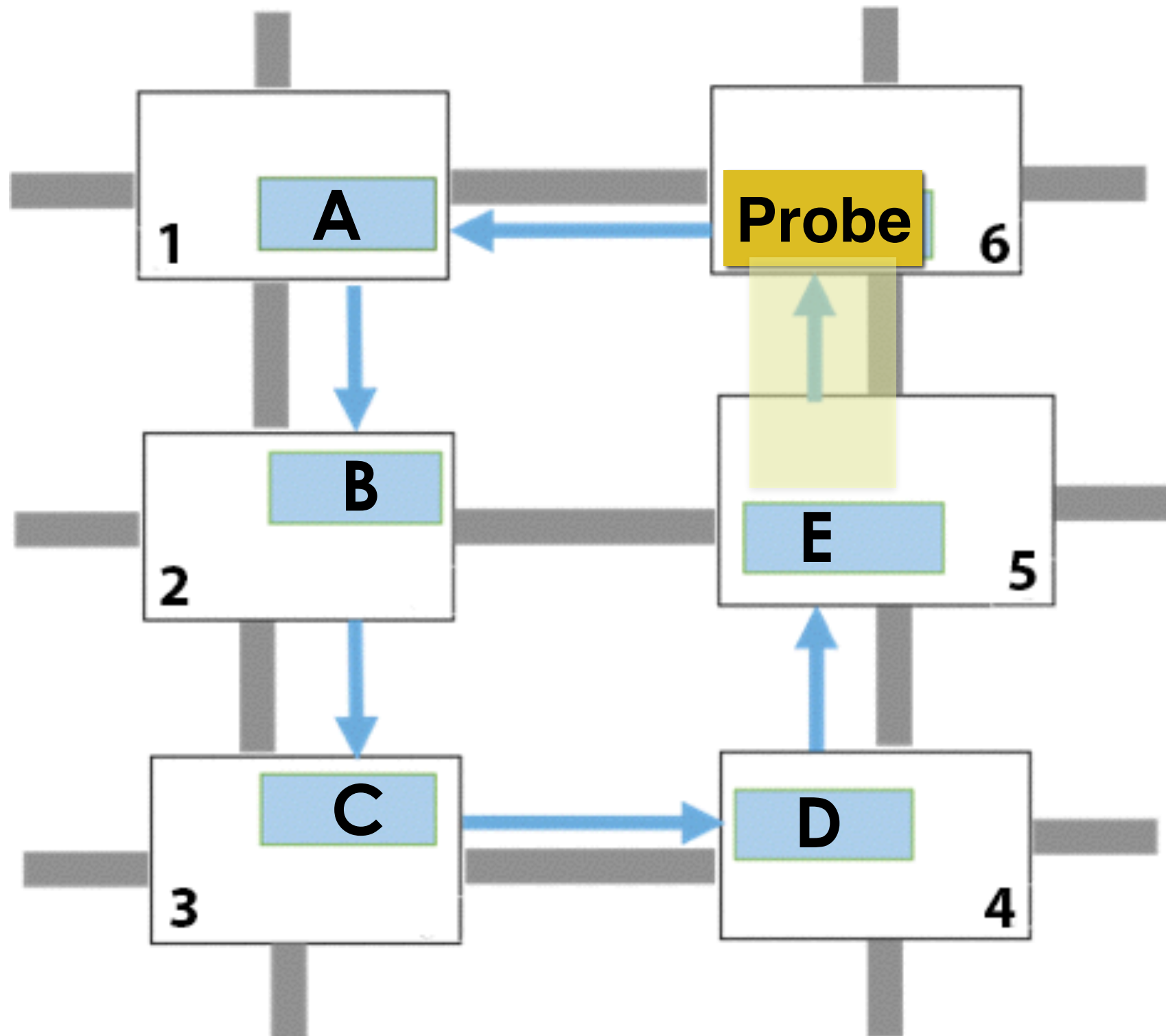
Implementation Example : *Probe Msg.*



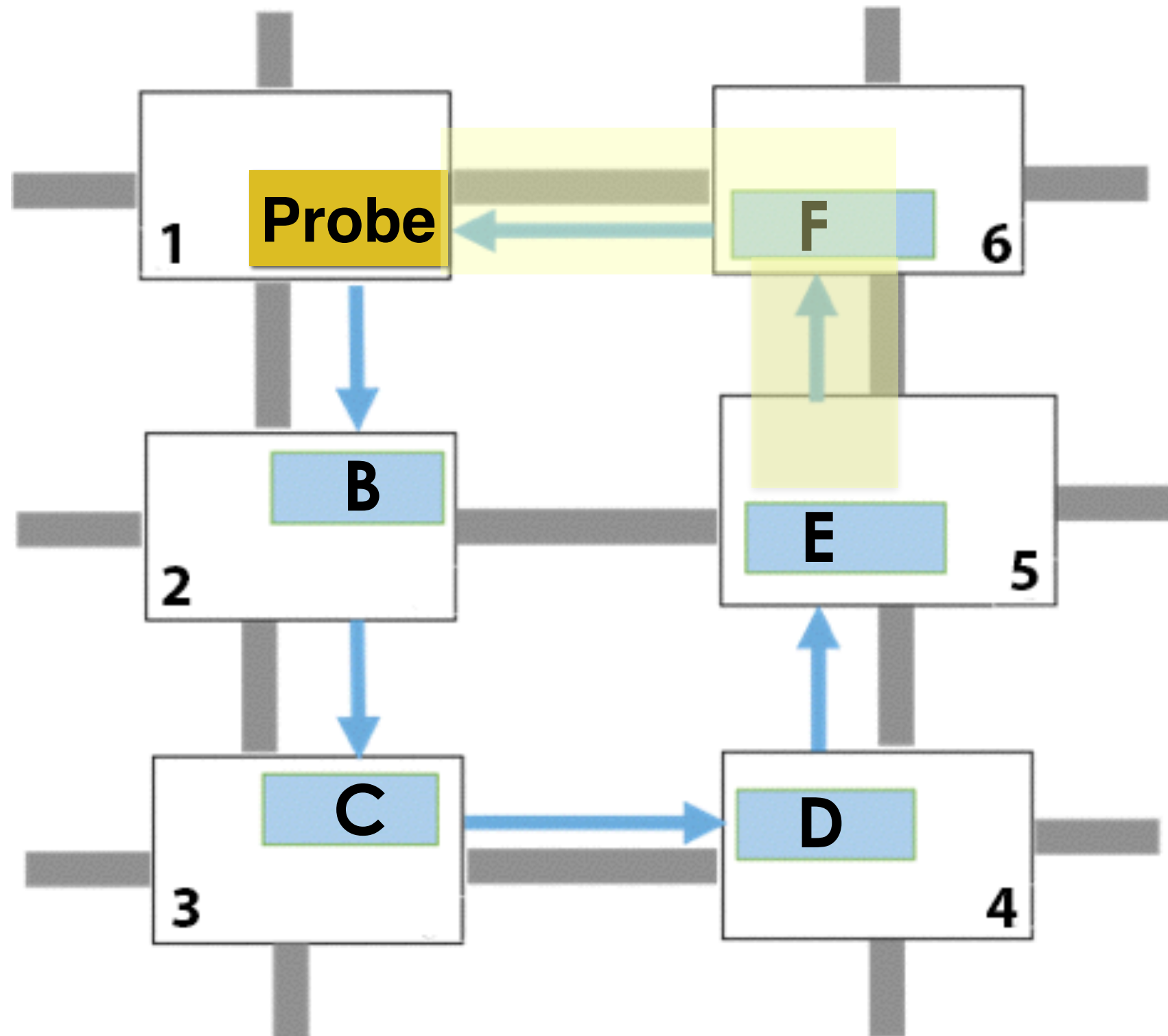
Implementation Example : *Probe Msg.*



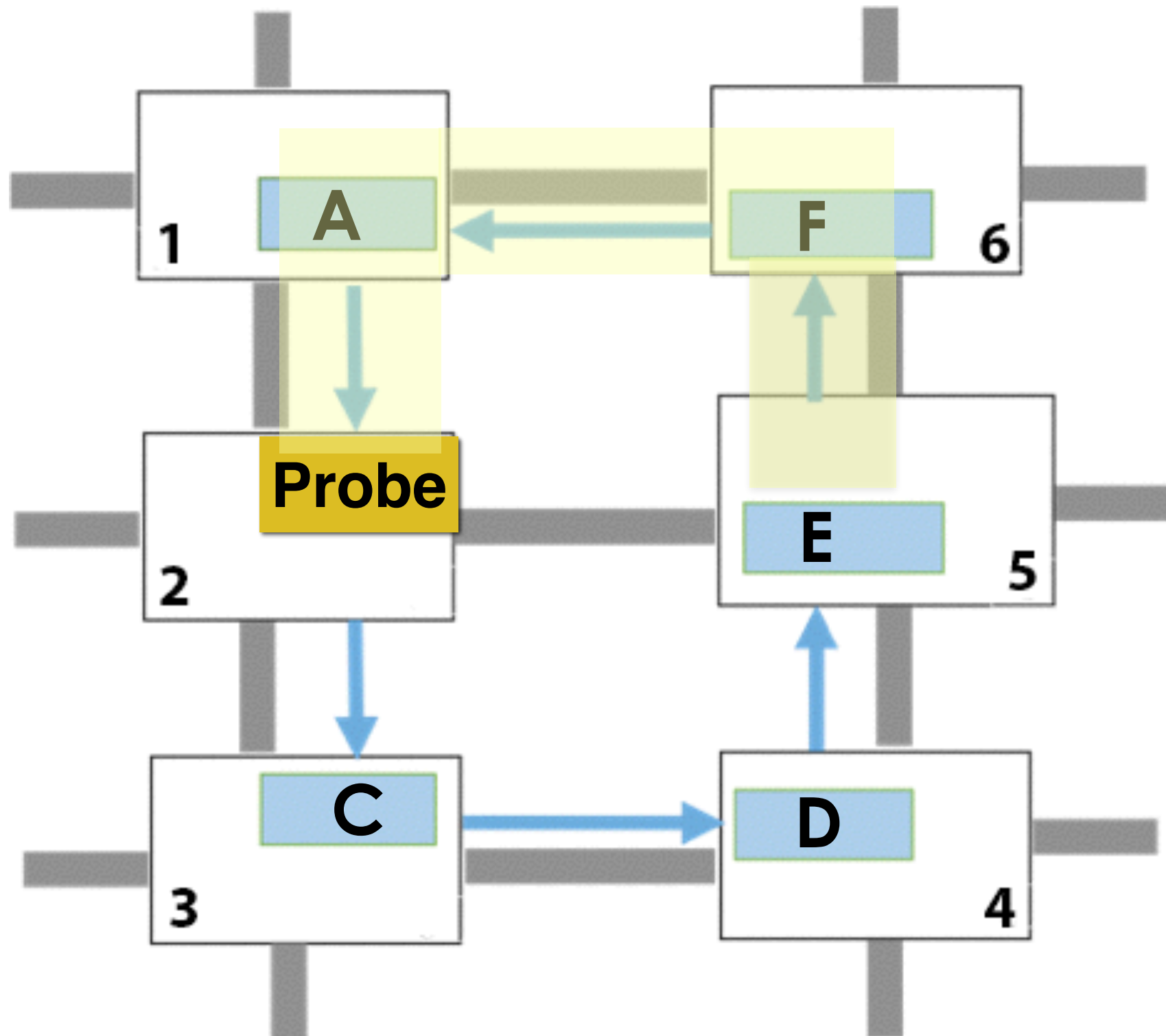
Implementation Example : *Probe Msg.*



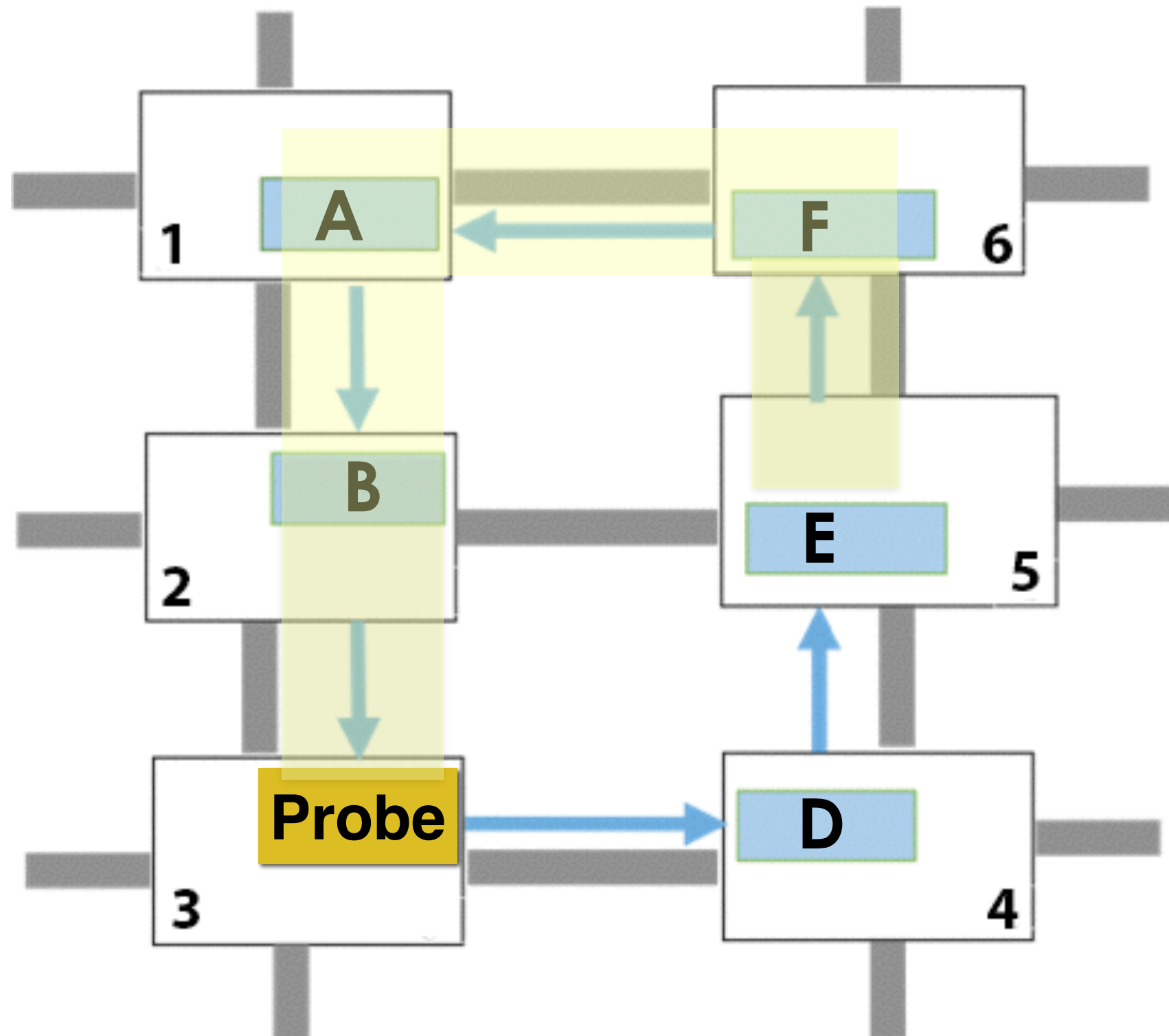
Implementation Example : *Probe Msg.*



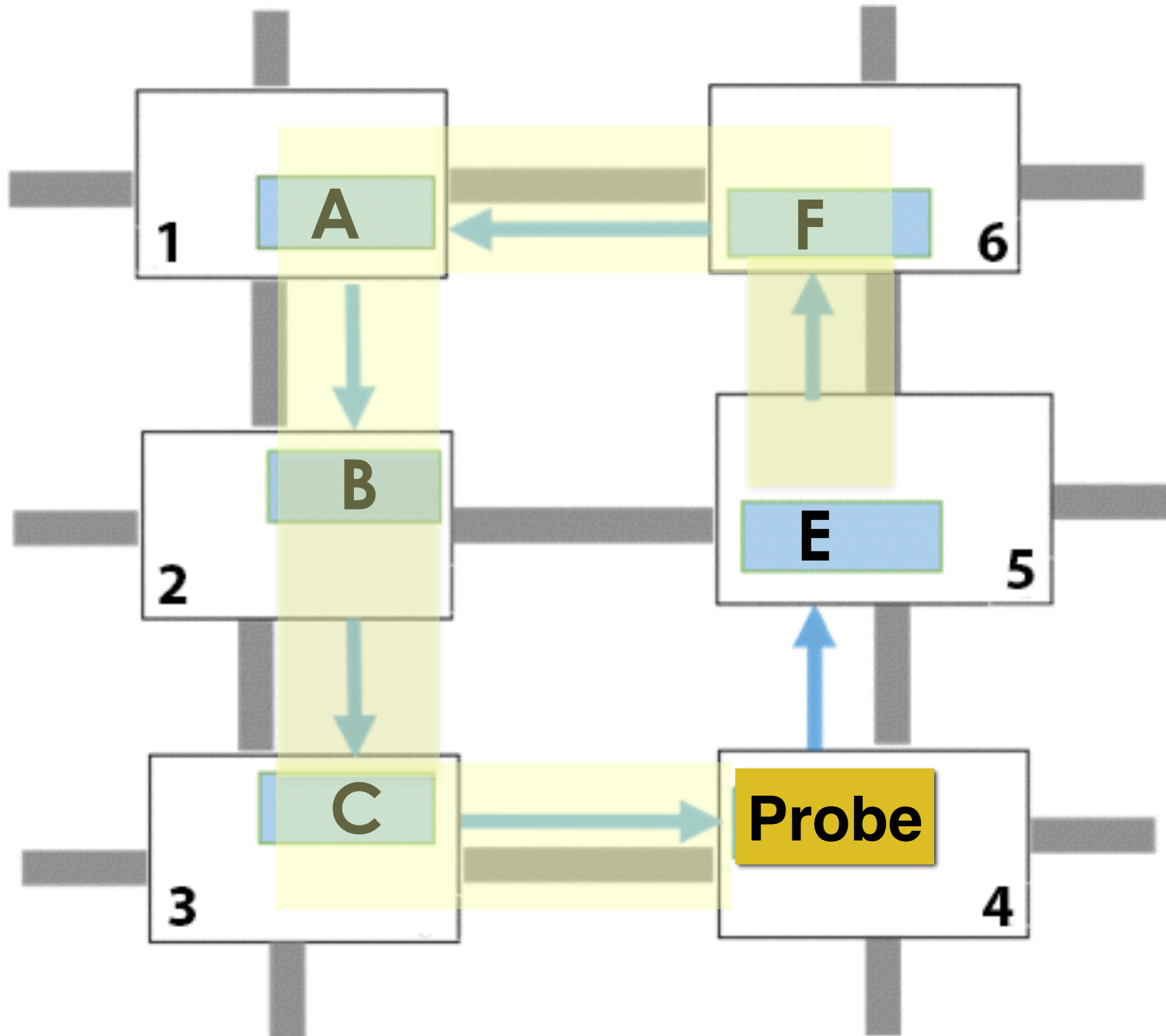
Implementation Example : *Probe Msg.*



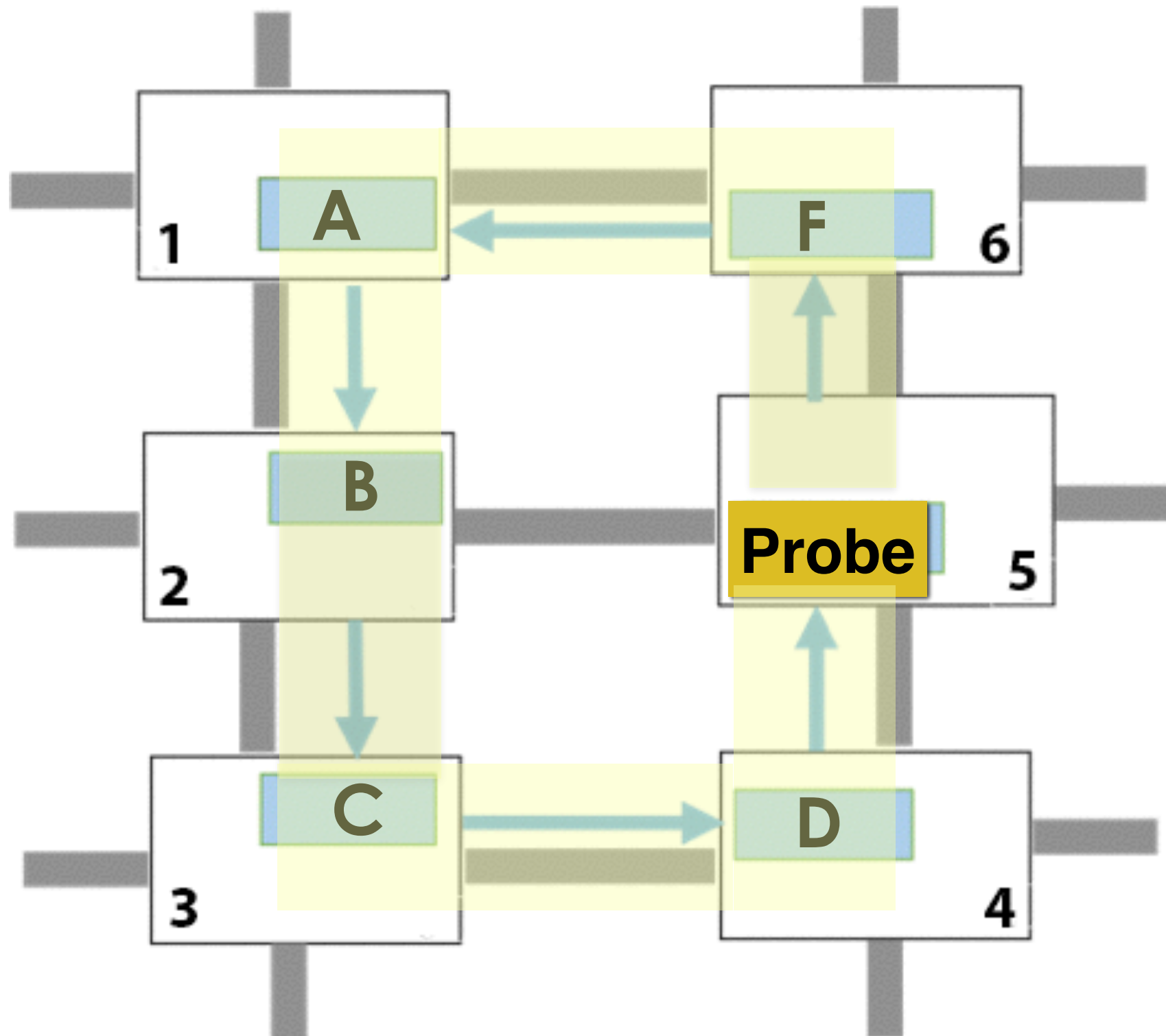
Implementation Example : *Probe Msg.*



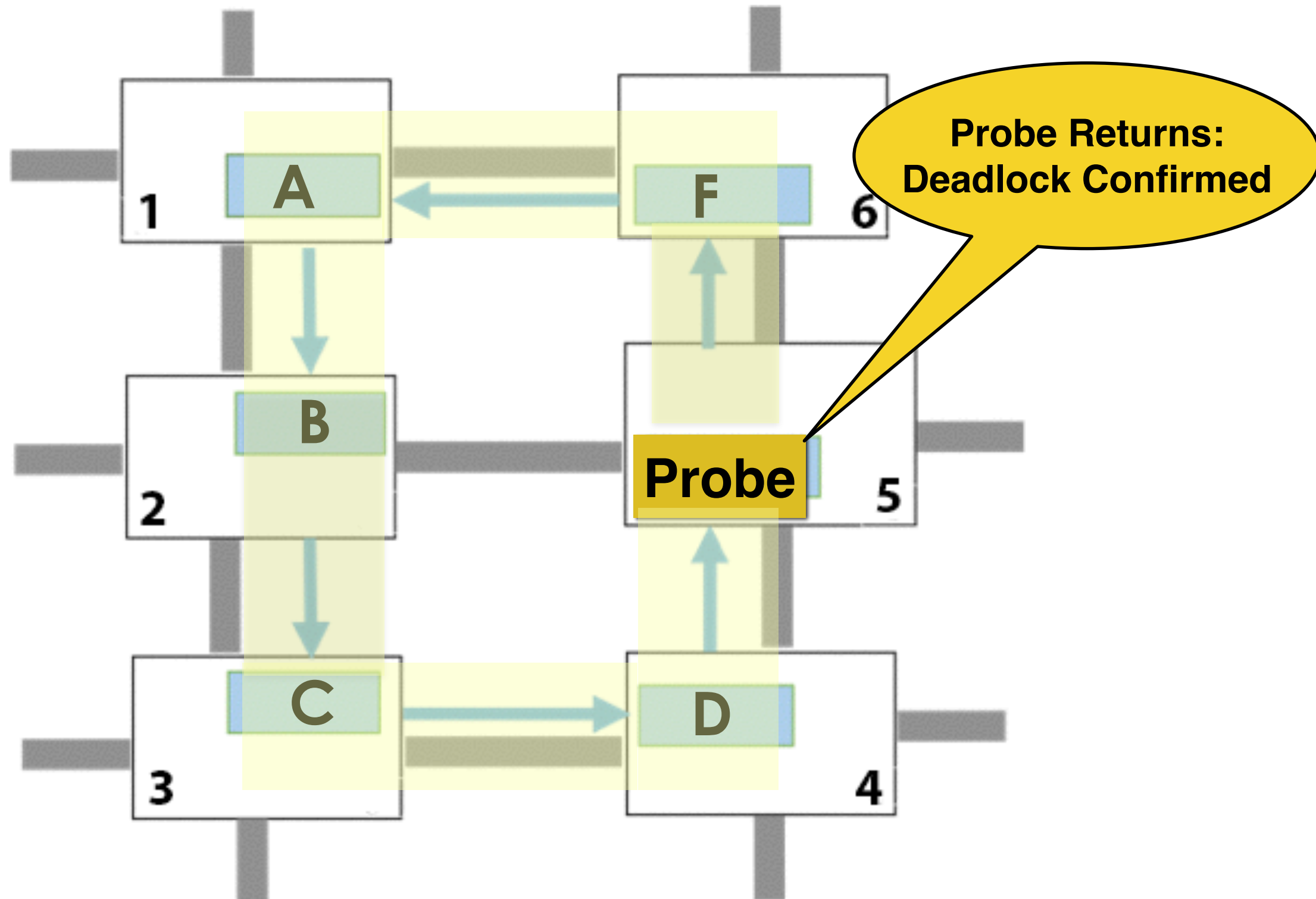
Implementation Example : *Probe Msg.*



Implementation Example : *Probe Msg.*

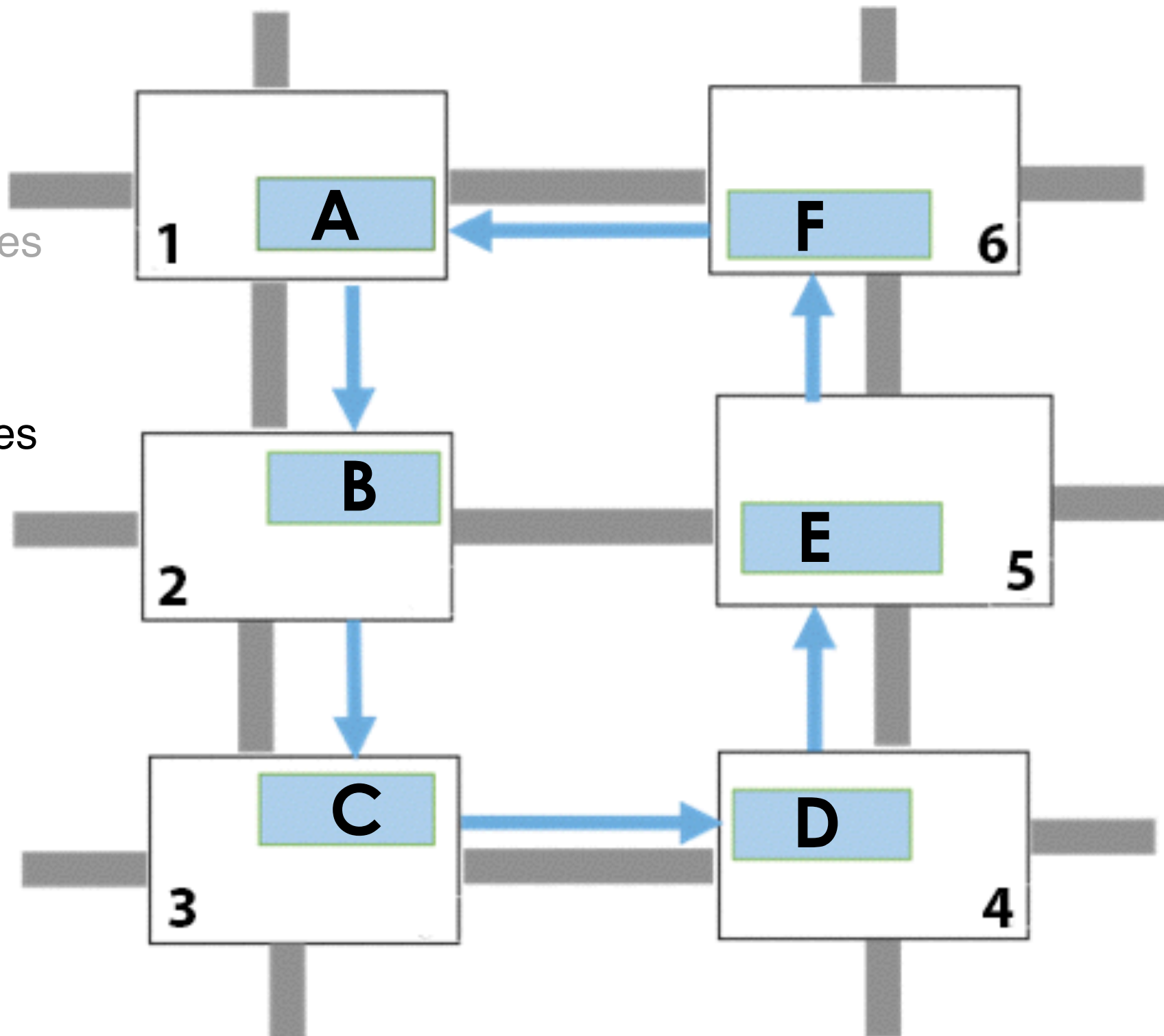


Implementation Example : *Probe Msg.*



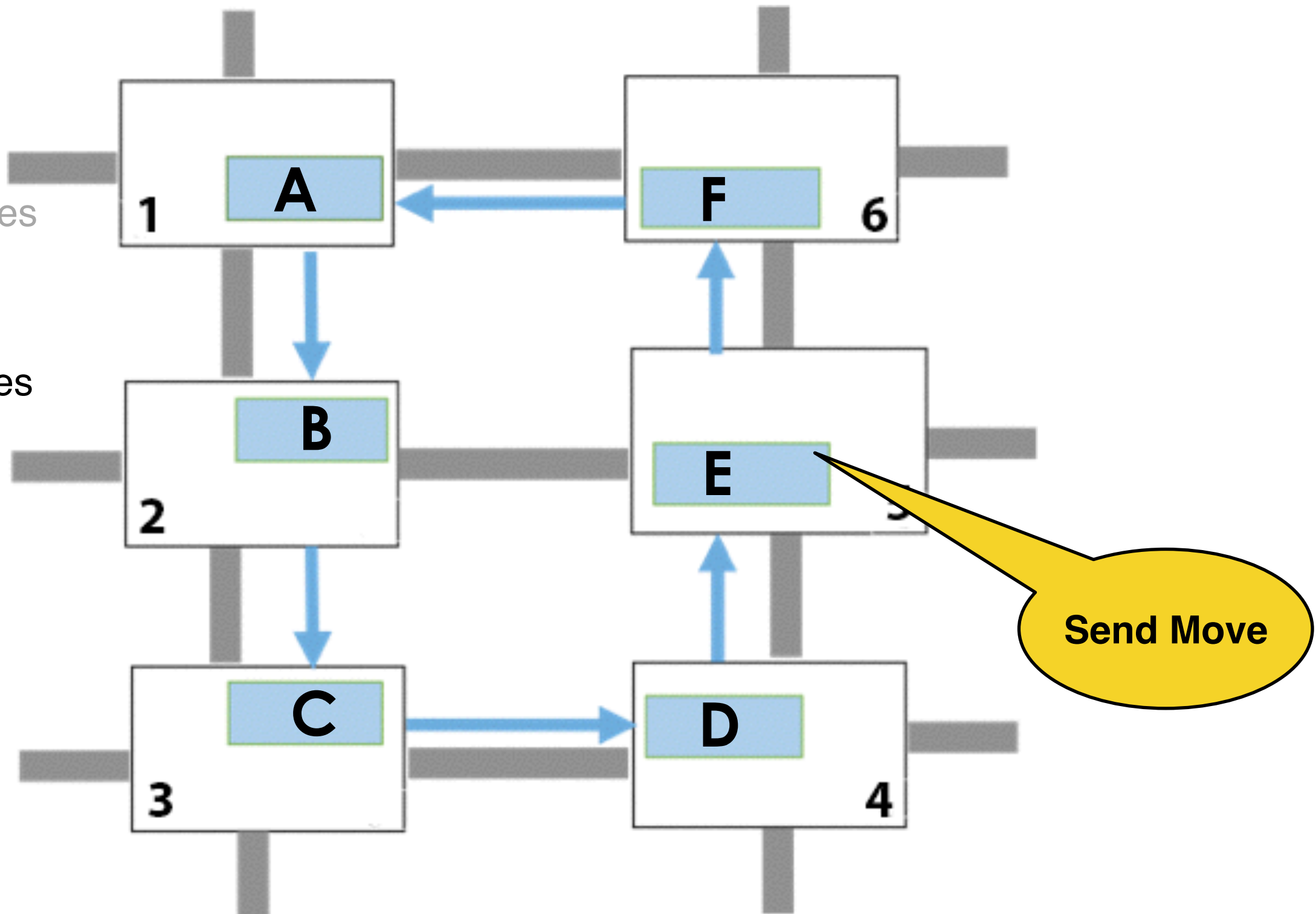
Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



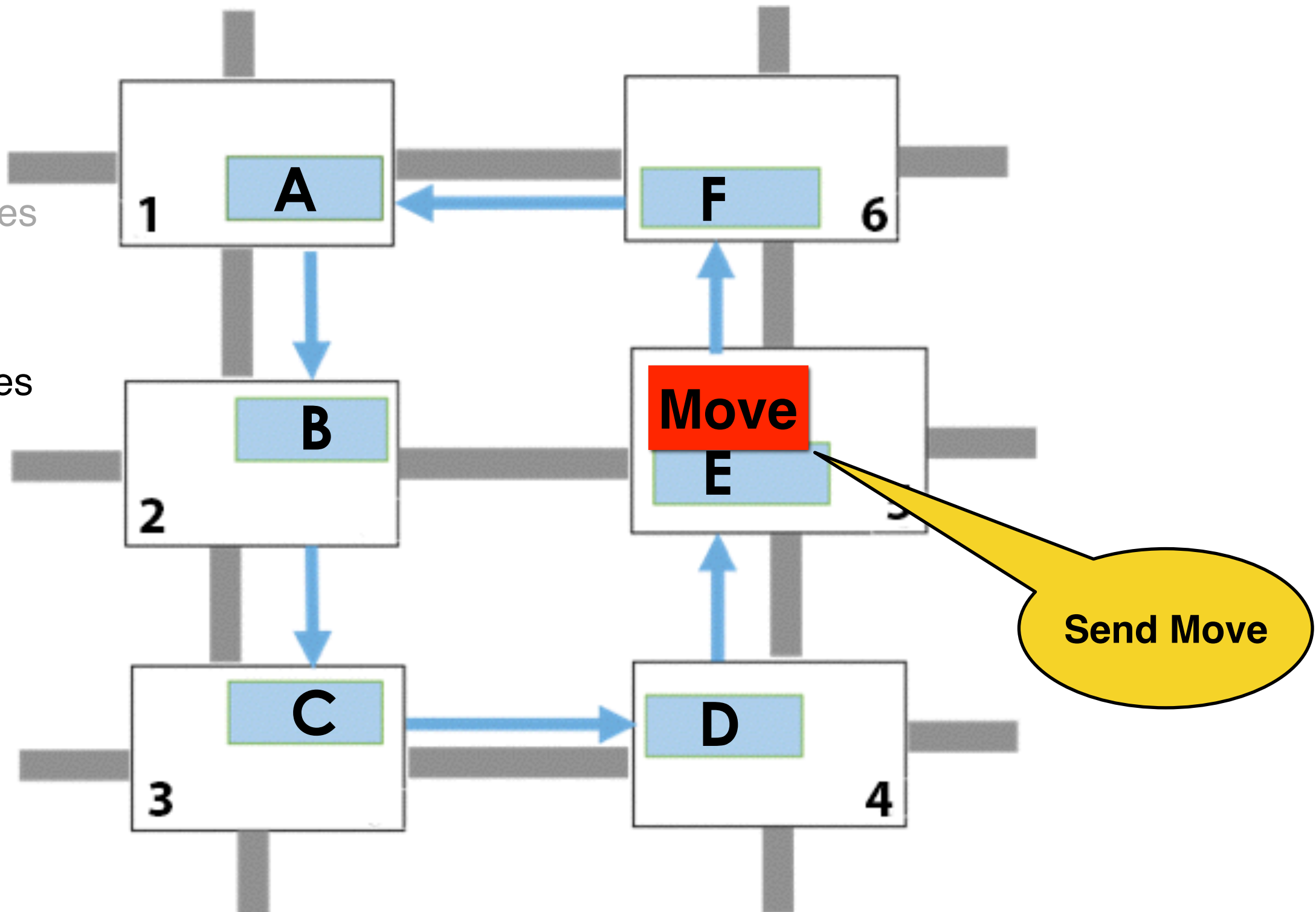
Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



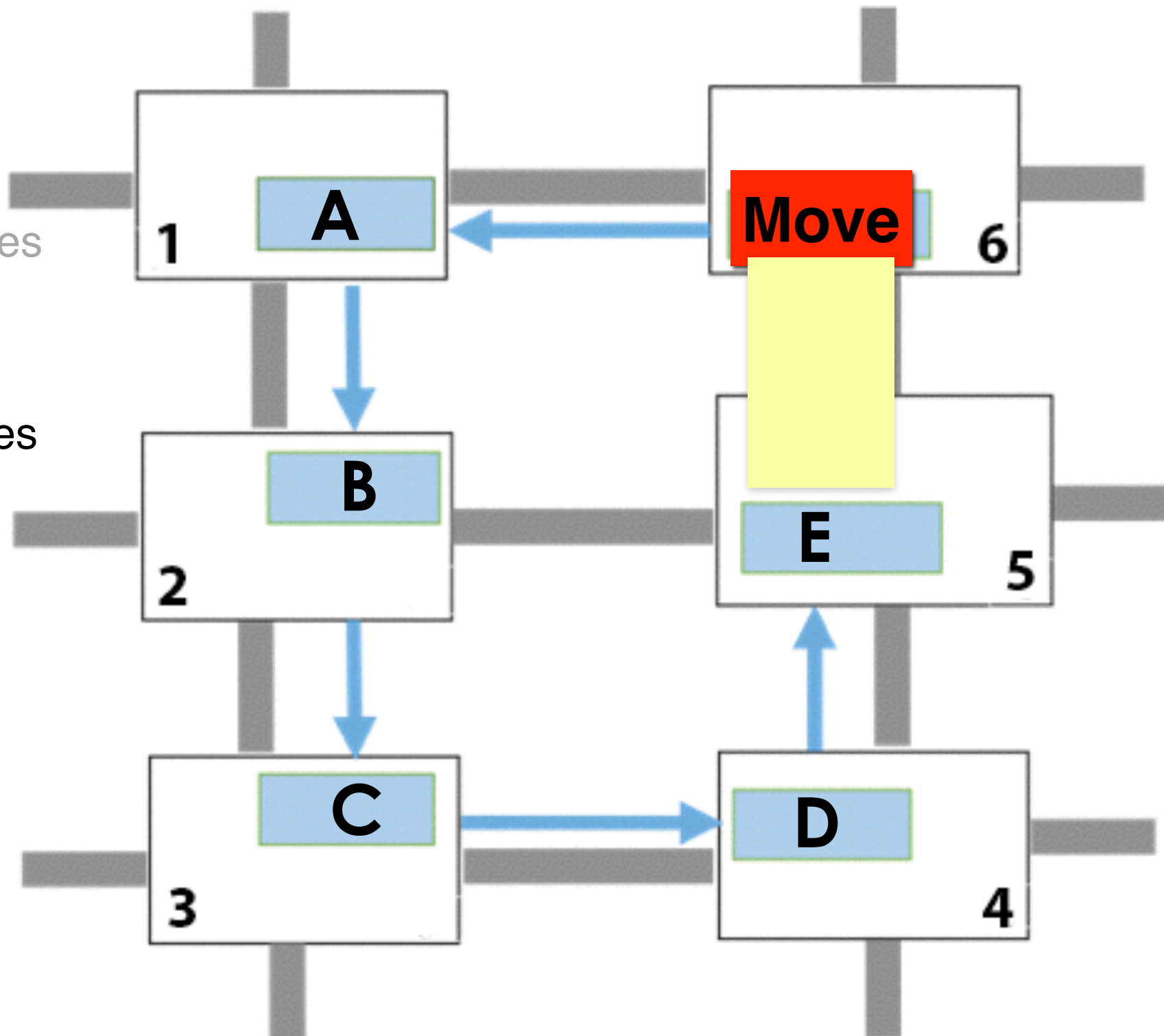
Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



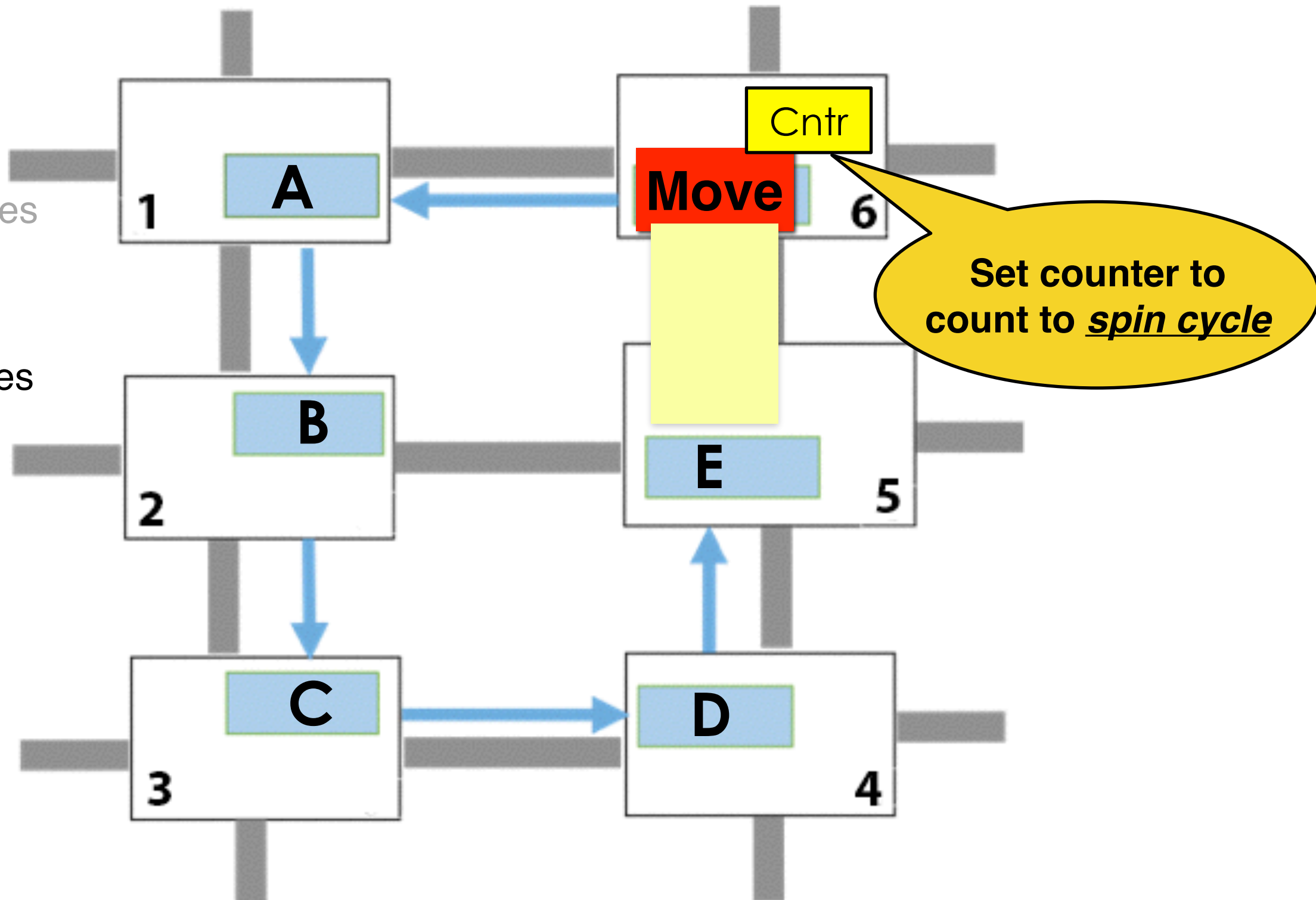
Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



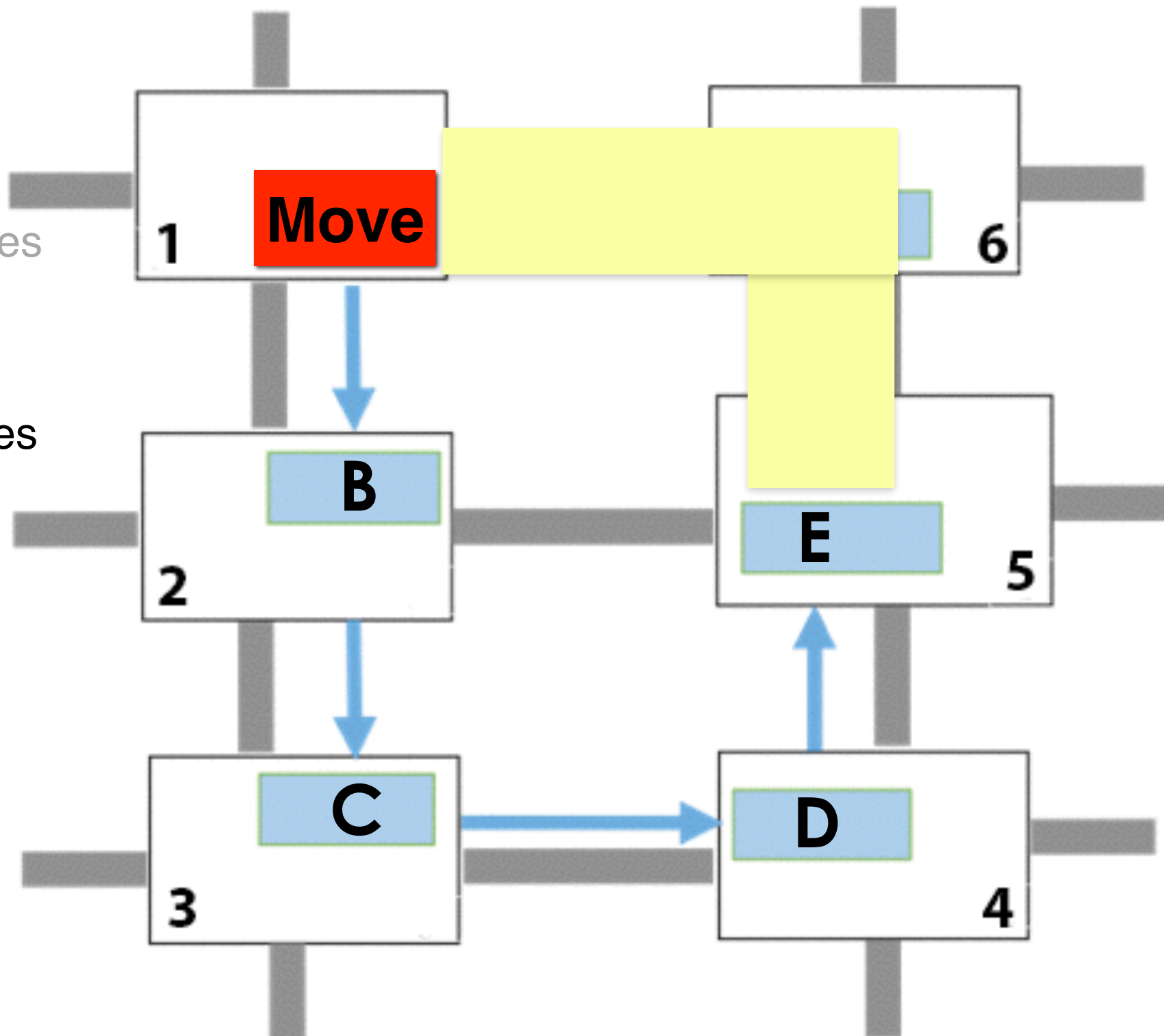
Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



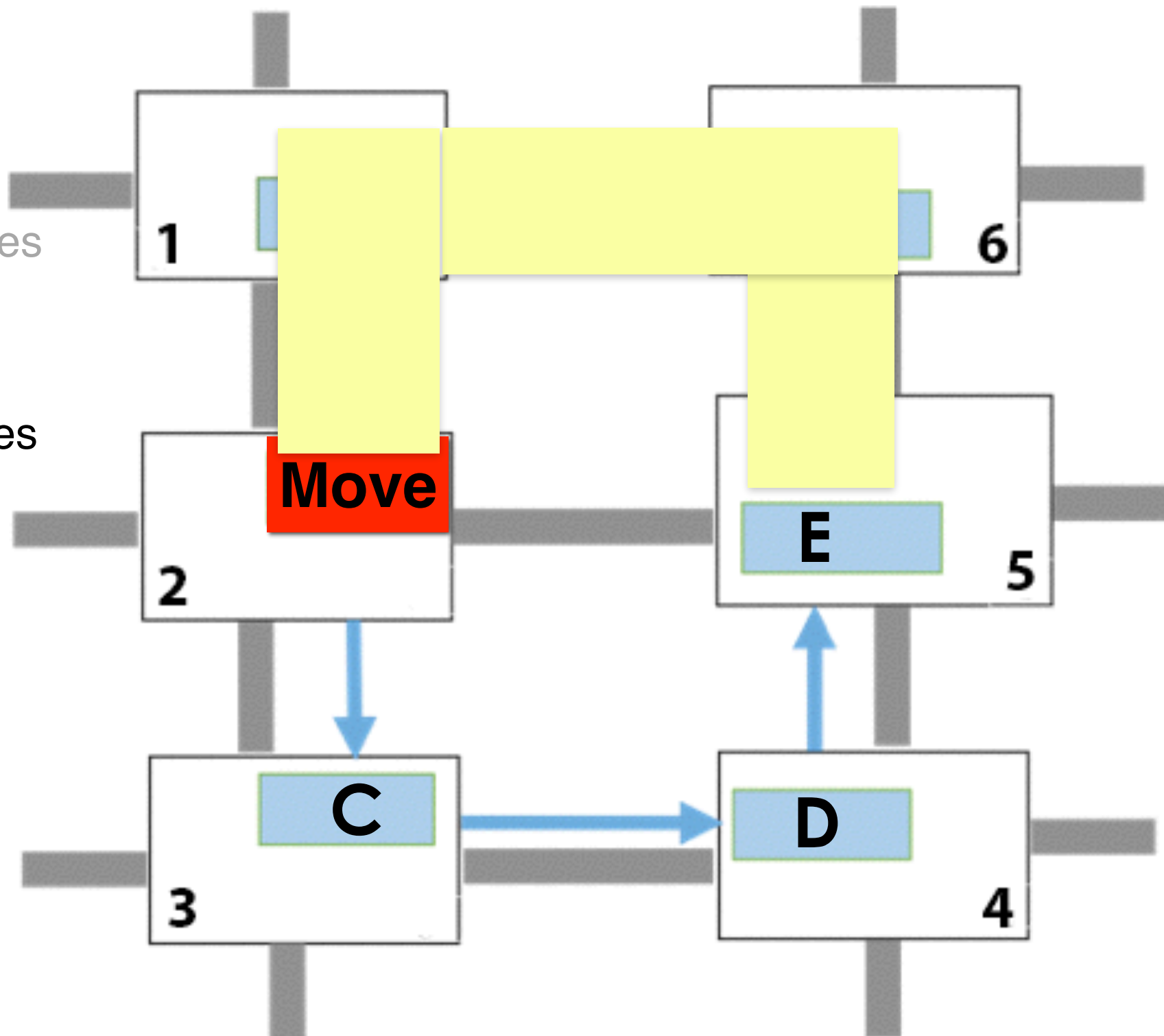
Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



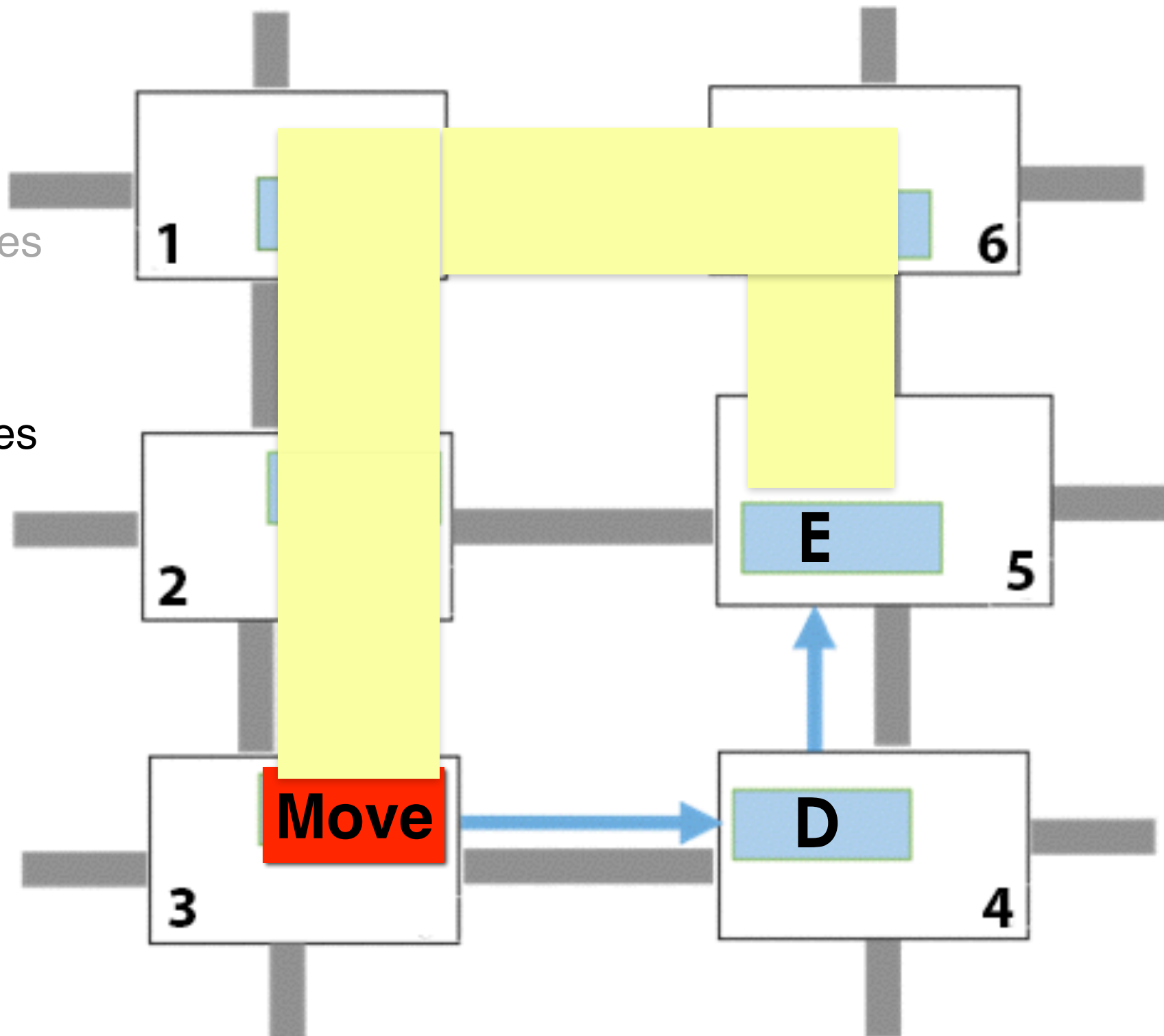
Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



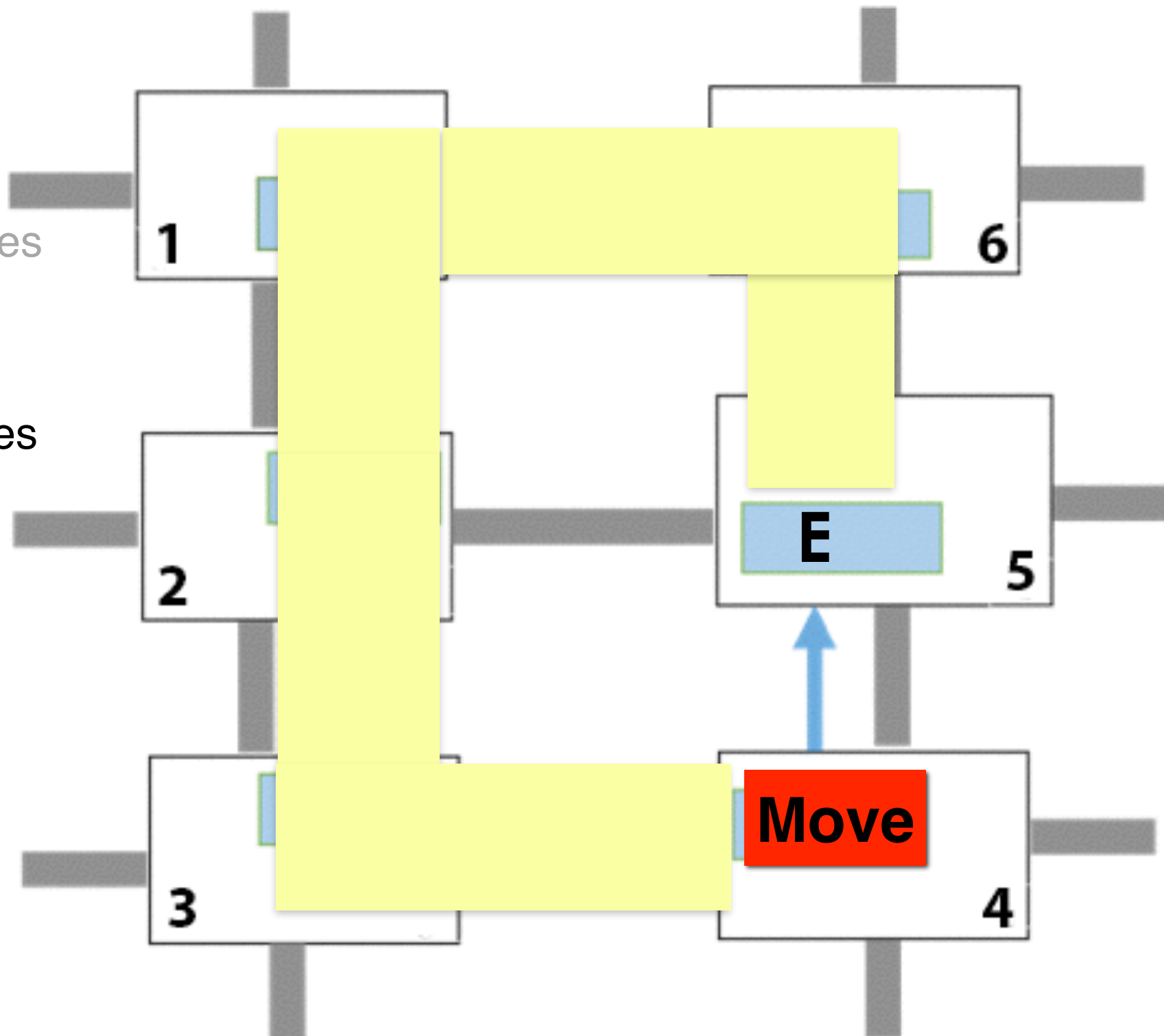
Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



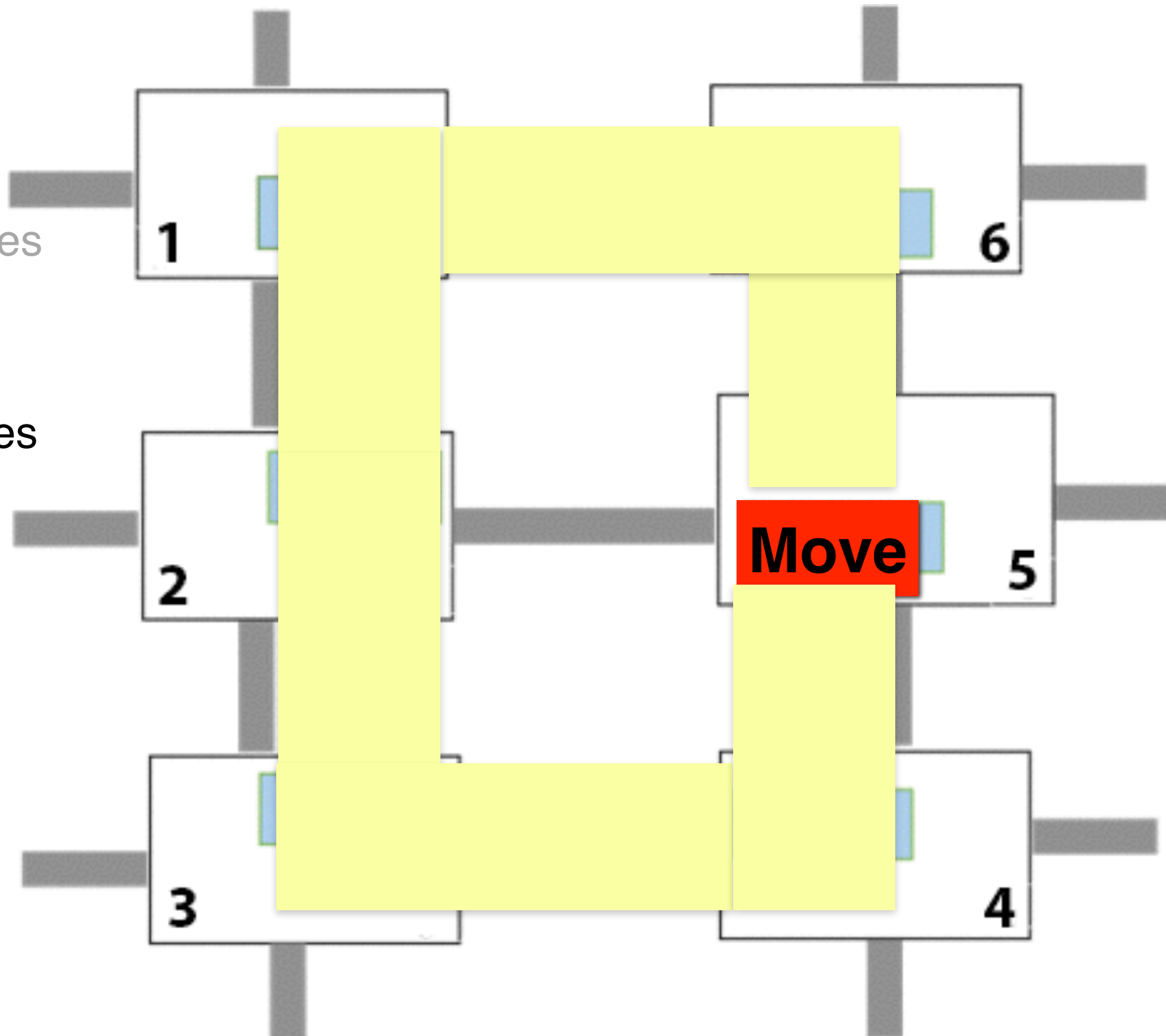
Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin



Implementation Example : *Move Msg.*

1. Counter Expires
2. Send Probe
- 3. Send Move**
4. Counter expires in spin cycle
5. Spin

