ECE 6115 / CS 8803 - ICN
Interconnection Networks for High Performance Systems
Spring 2020

DEADLOCKS

Tushar Krishna
Assistant Professor
School of Electrical and Computer Engineering
Georgia Institute of Technology

tushar@ece.gatech.edu

NETWORK ARCHITECTURE

- **Topology**
  - How to connect the nodes
  - ~Road Network
- **Routing**
  - Which path should a message take
  - ~Series of road segments from source to destination
- **Flow Control**
  - When does the message have to stop/proceed
  - ~Traffic signals at end of each road segment
- **Router Microarchitecture**
  - How to build the routers
  - ~Design of traffic intersection (number of lanes, algorithm for turning red/green)

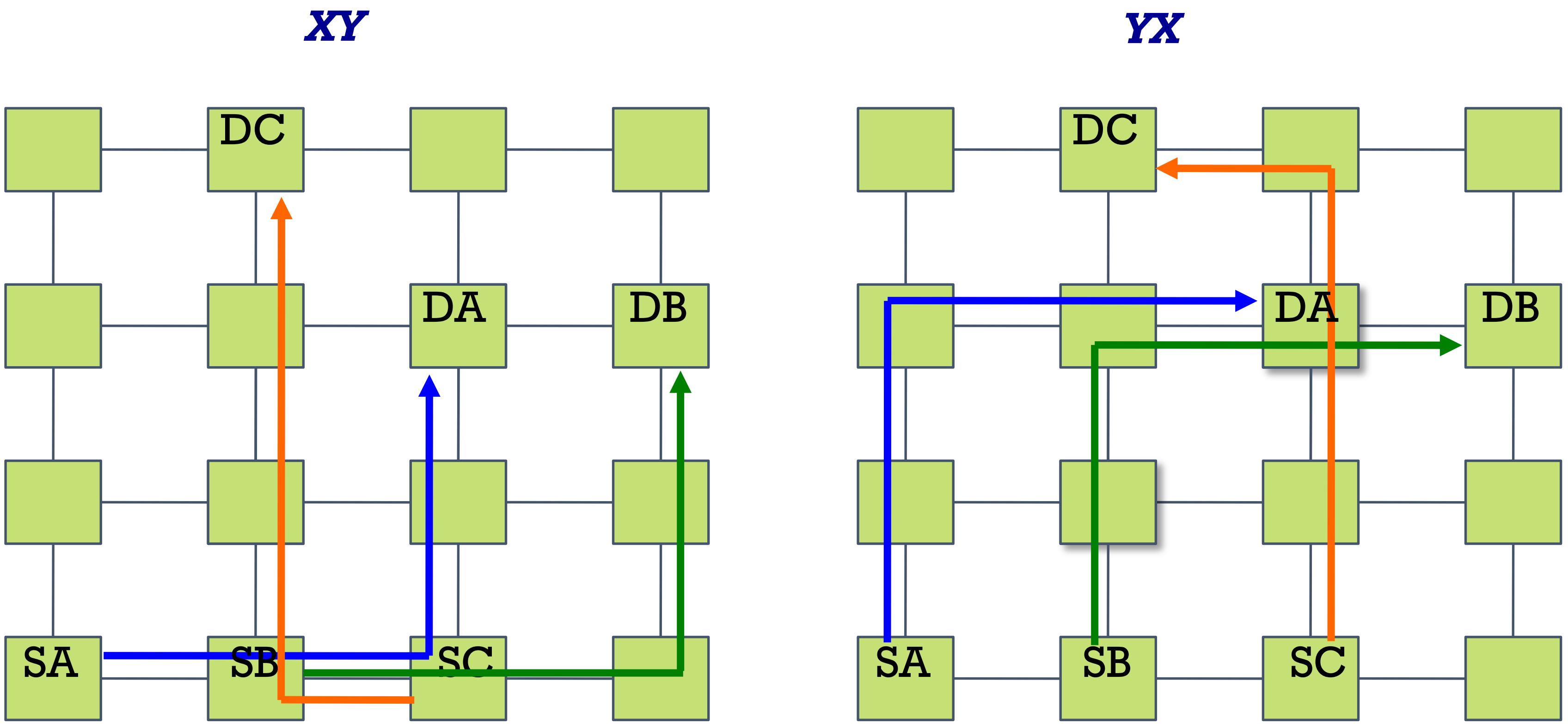
TAXONOMY OF ROUTING ALGORITHMS

- **Classification I: path length**
  - **Minimal**: shortest paths
    - Example: Greedy over Ring, XY over Mesh
  - **Non-minimal**: non-shortest paths
    - Example: Random and Adaptive over Ring/Mesh

TAXONOMY OF ROUTING ALGORITHMS

- **Classification II: path diversity** (how to select between the set of all possible paths R_{xy} from the source x to the dest y)
  - **Deterministic:** always choose the same route between x and y, even if $|R_{xy}| > 1$
    - **Example:** Greedy over Ring, XY over Mesh
    - + Easy to Implement
    - - Inefficient use of bandwidth
  - **Oblivious:** choose any of the routes in R_{xy} without considering any information about current network state (i.e., congestion)
    - **Example:** Random over Ring, O1Turn over Mesh
    - + More path diversity
    - - Can lead to deadlocks (this lecture)
  - **Adaptive:** choose one of the routes in R_{xy} depending on the current network state (i.e., congestion)
    - **Example:** Adaptive over Ring/Mesh
    - + Best use of available bandwidth
    - - Need to track congestion, can lead to deadlocks

RECAP: O1TURN ROUTING ALGORITHM

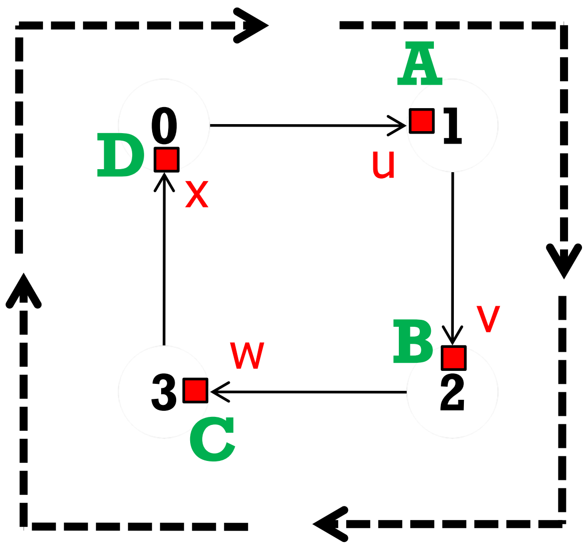


Randomly send over XY or YX

Minimal and Oblivious

DEADLOCK

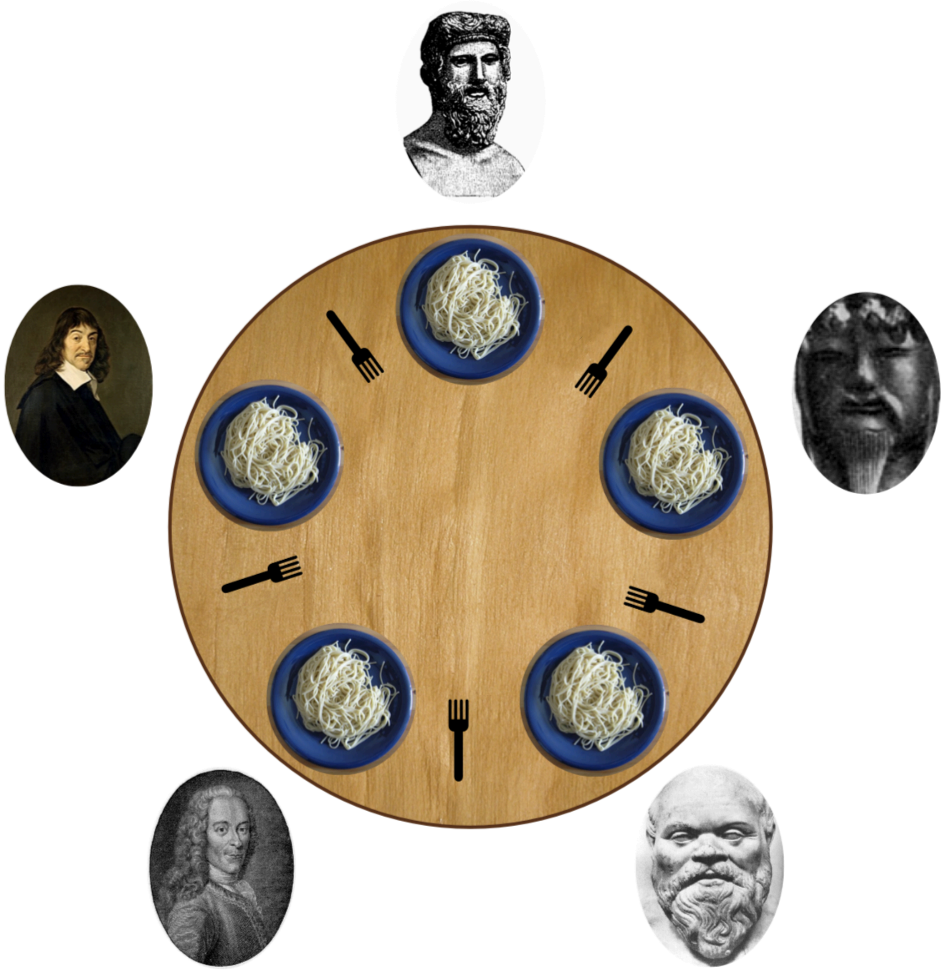
- A condition in which a set of **agents** wait indefinitely trying to acquire a set of **resources**



Note: holding buffer u == holding Channel 01 as no other packet can use channel 01 till buffer u becomes free

- Packet A holds buffer u (in 1) and wants buffer v (in 2)
- Packet B holds buffer v (in 2) and wants buffer w (in 3)
- Packet C holds buffer w (in 3) and wants buffer x (in 0)
- Packet D holds buffer x (in 0) and wants buffer u (in 1)

CLASSIC EXAMPLE: DINING PHILOSOPHER'S PROBLEM

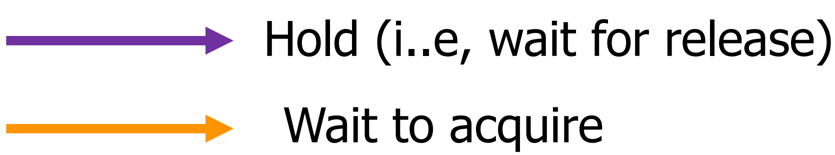


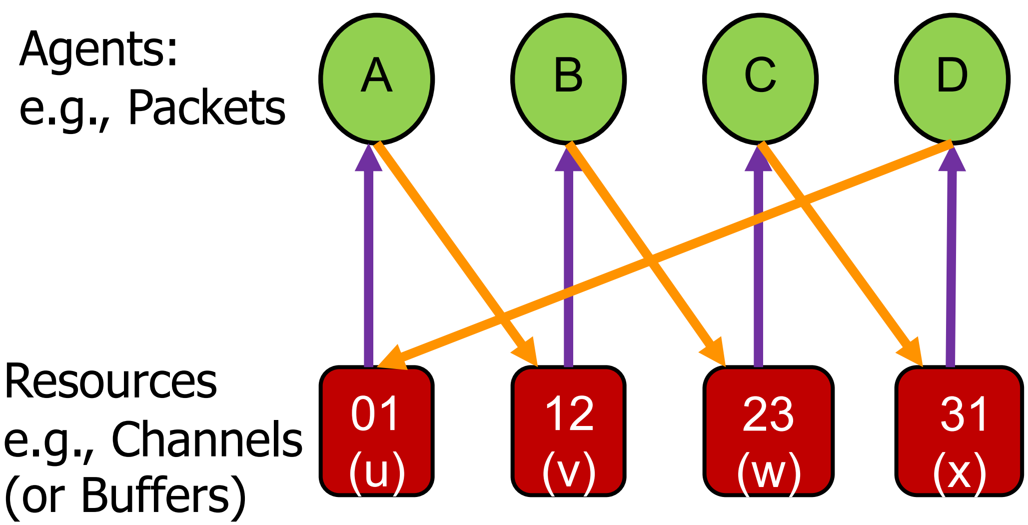
Agents:
Philosophers

Resources:
Forks

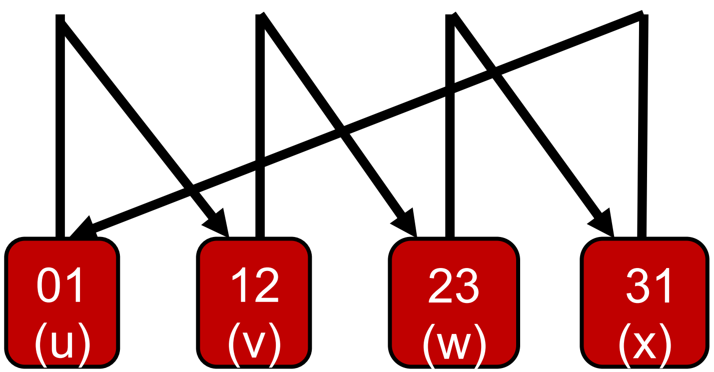
RESOURCE DEPENDENCE

Resource A is ***dependent*** on resource B if it is possible for A to be *held-by* an agent X and it is also possible for X to *wait-for* B





Resource (Channel or Buffer) Dependence Graph



DEADLOCK CONDITION

- Agents hold and do not release a resource while waiting for access to another

- A cycle exists between waiting agents such that there exists a set of agents A_0, .. A_{n-1}, where agent A_i holds resource R_i, while waiting on resource $R_{(i+i \bmod n)}$, for i = 0, …, n-1

- To avoid deadlock – resource dependence graph should not have any cycles

DEALING WITH DEADLOCKS

- **Proactive / Avoidance**
  - Guarantee that the network will never deadlock
  - Almost all modern networks use deadlock avoidance

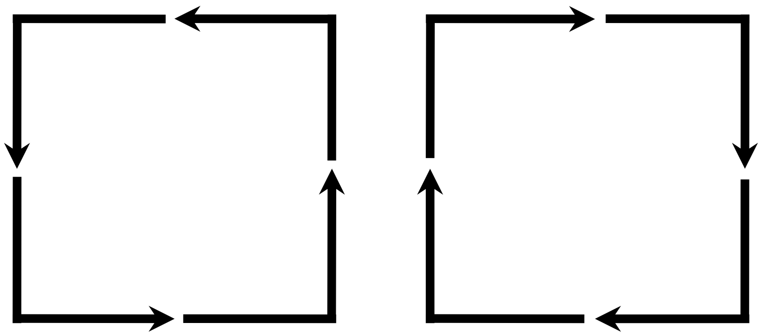
- **Reactive / Recovery**
  - Detect deadlock and correct

- **Subactive**
  - Introduce periodic forced movement among packets

DEADLOCK AVOIDANCE

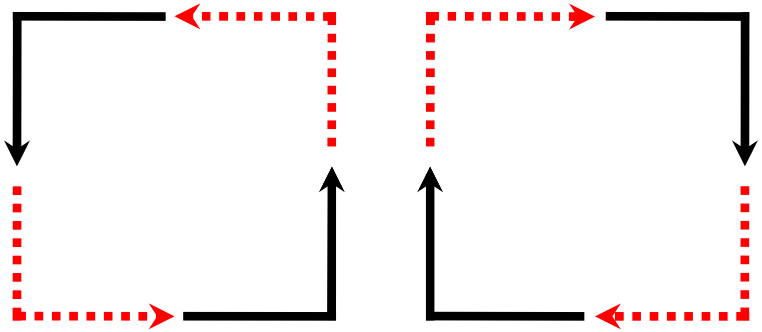
- Eliminate cycles in Resource Dependency Graph
  - Resource Ordering
    - Enforce a partial/total order on the resources, and insist that an agent acquire the resources in ascending order
    - Deadlock avoided since a cycle must contain at least one agent holding a higher numbered resource waiting for a lower-numbered resource which is not allowed by the ordering allocation
  - Implementation
    - Restrict certain routes so that a higher numbered resource cannot wait for a lower numbered resource
    - Partition the buffers at each node such that they belong to different resource classes. A packet on any route can only acquire buffers in ascending order of resource class

TURN MODEL (GLASS AND NI 1994) FOR MESH

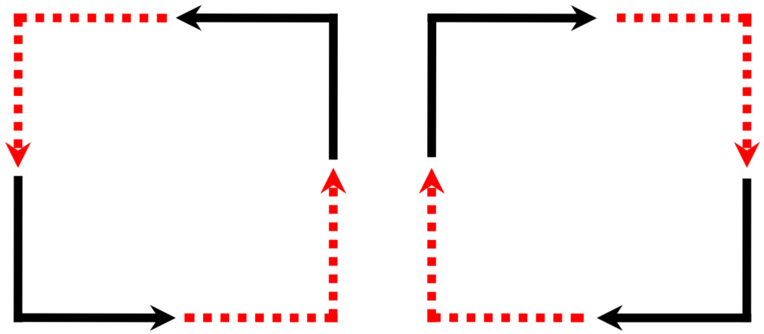


- Deadlocks may occur if the turns taken form a cycle
  - Removing some turns can make the routing algorithm deadlock free

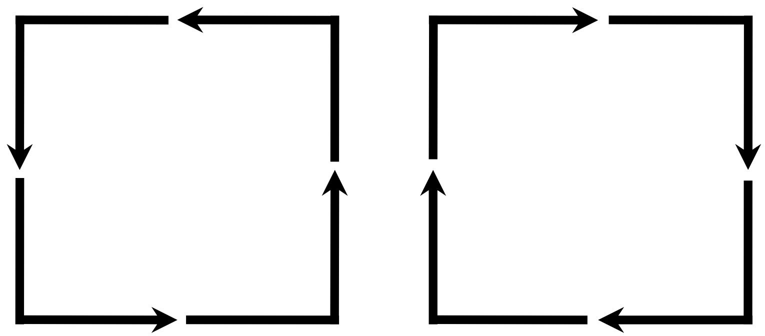
DIMENSION ORDERED ROUTING



XY Model

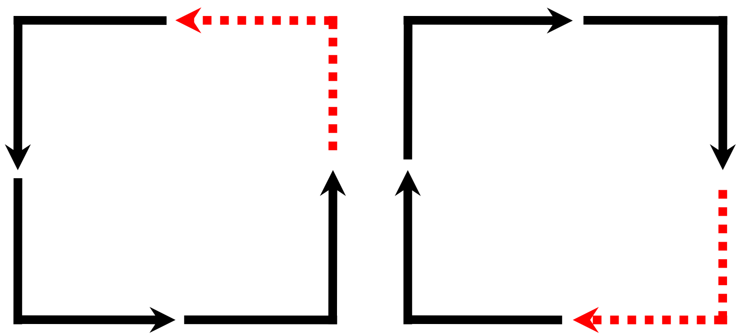


YX Model

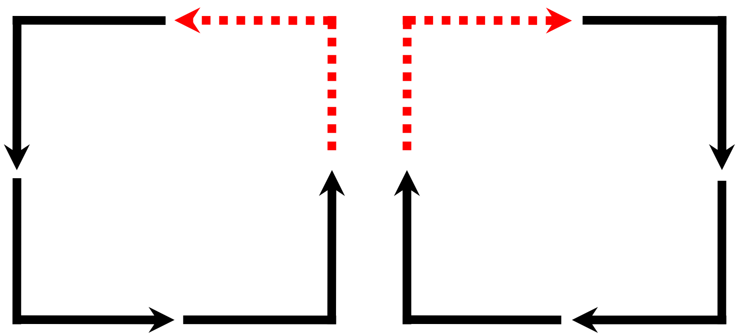


O1Turn Deadlock!

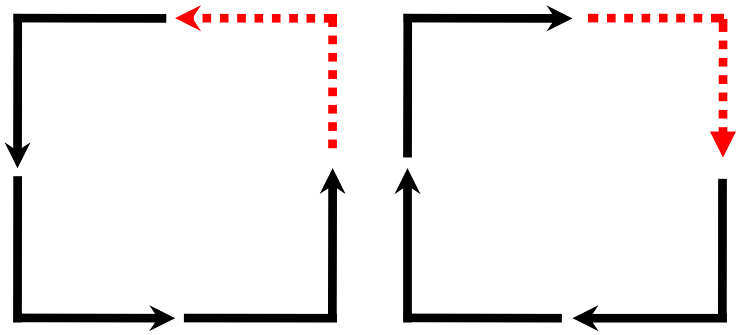
DEADLOCK-FREE OBLIVIOUS/ADAPTIVE ROUTING ALGORITHMS



West-First Turn Model

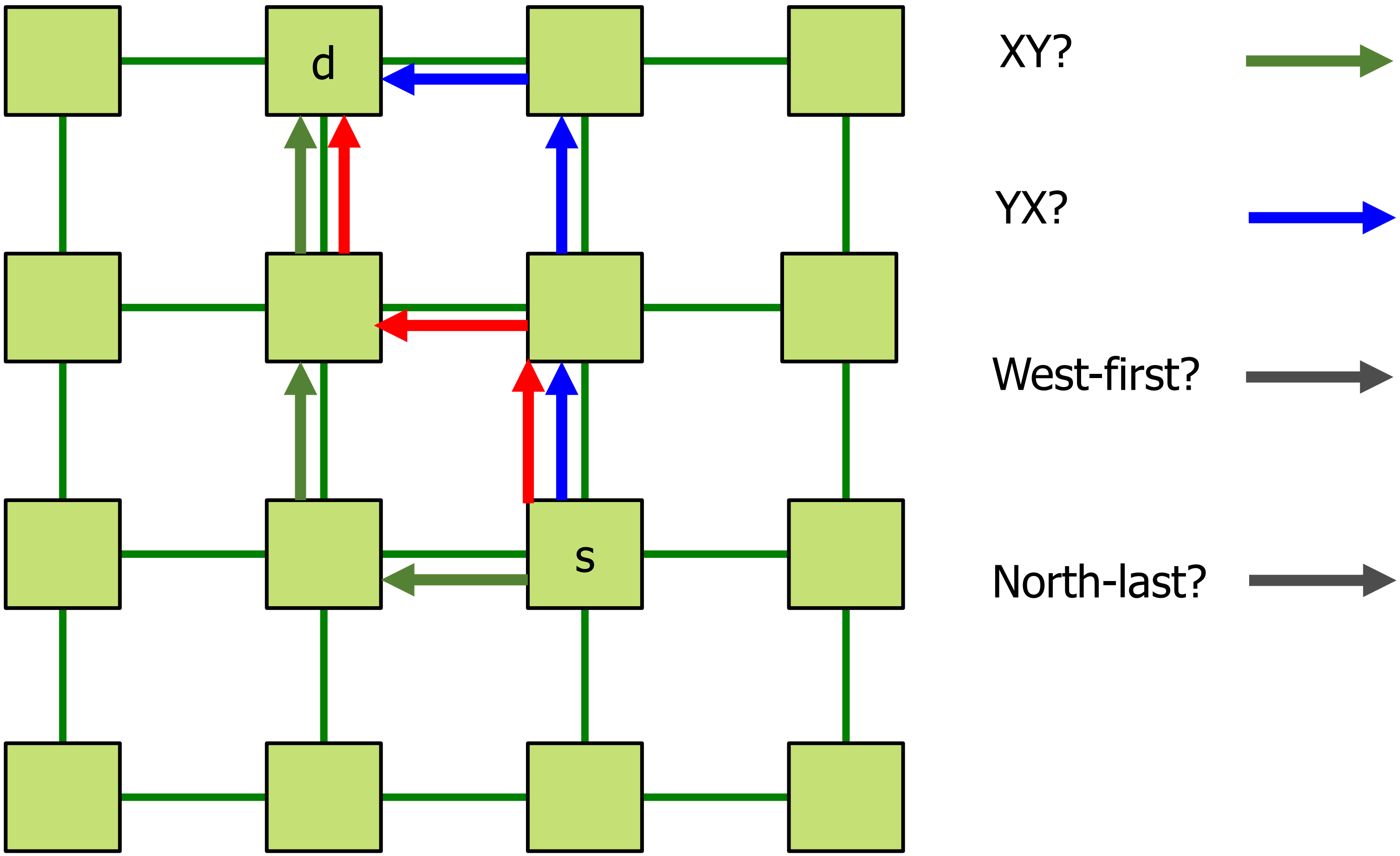


North-Last Turn Model

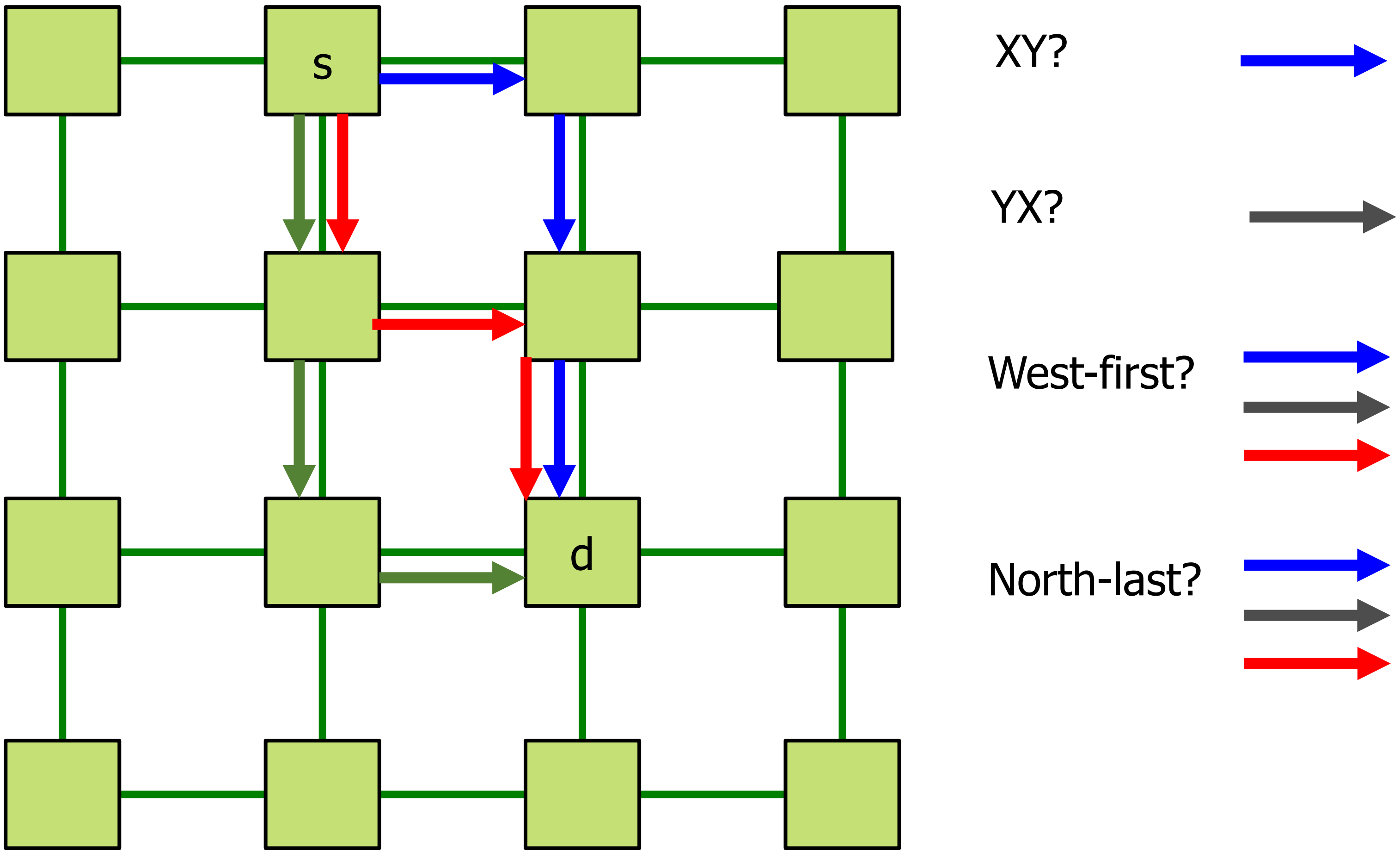


Negative-First Turn Model

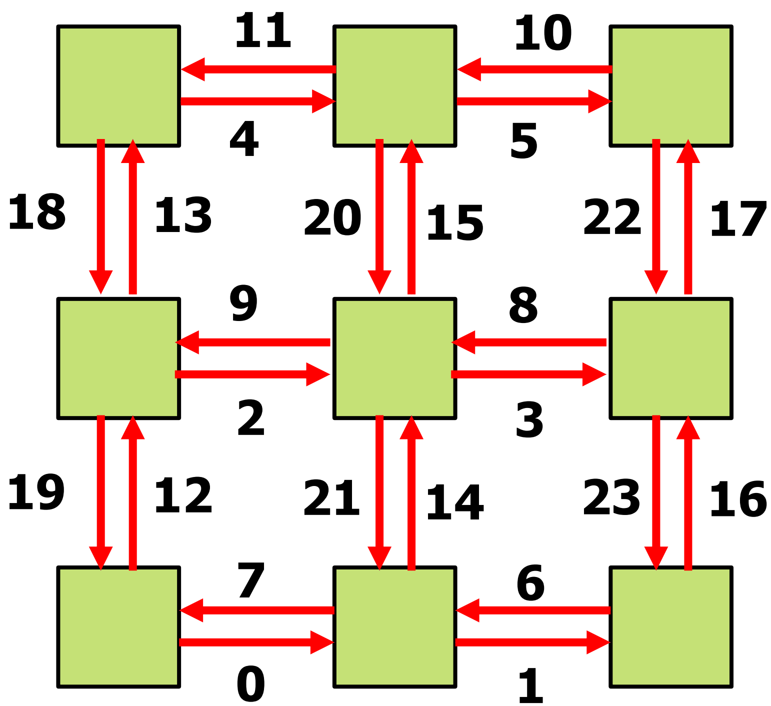
EXAMPLE 1



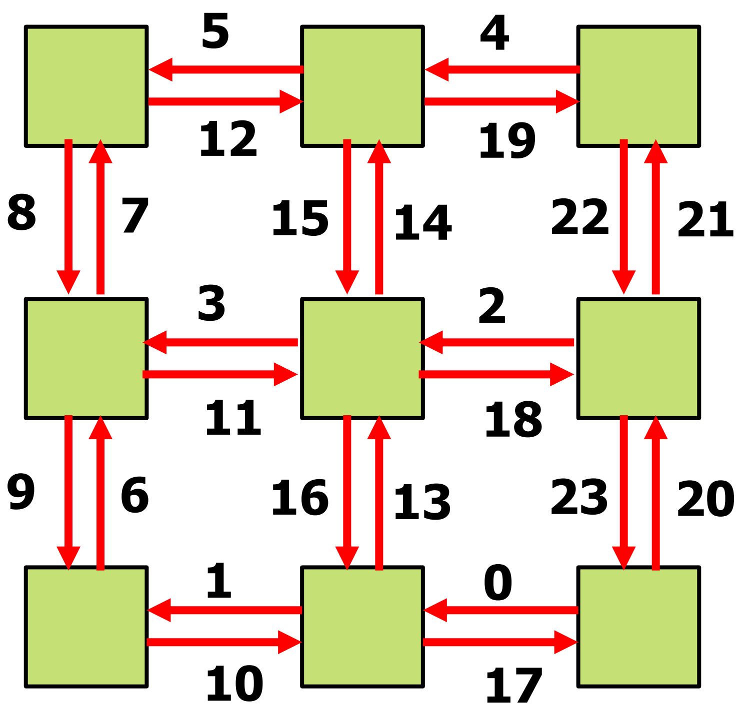
EXAMPLE 2



RESOURCE (CHANNEL) ORDERING

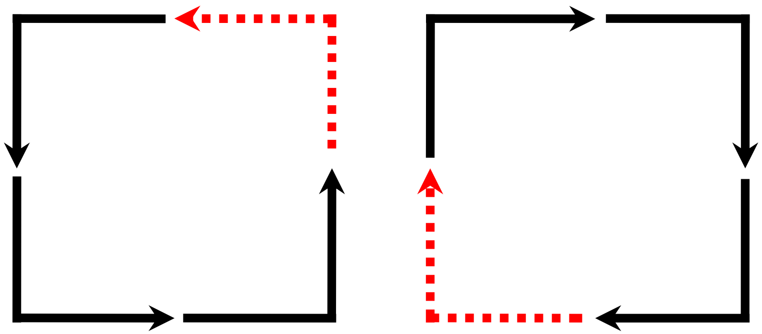


XY Model

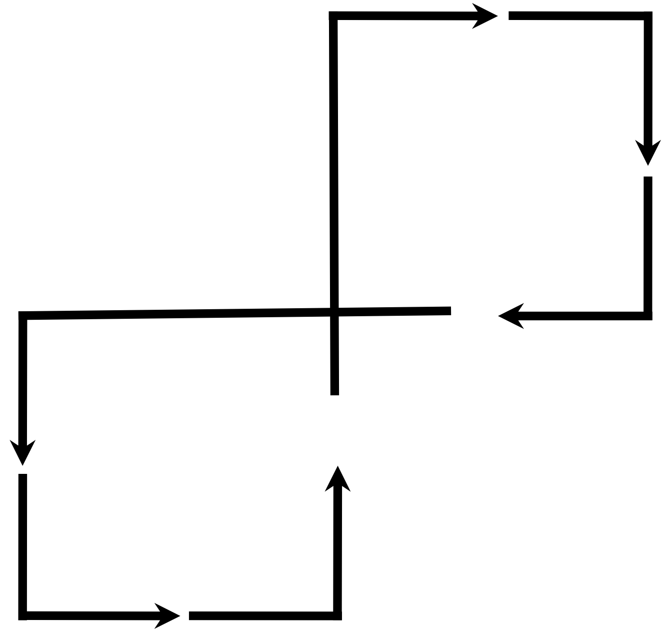


West-First Turn Model

CAN WE ELIMINATE *ANY* 2 TURNS?



Six turn model

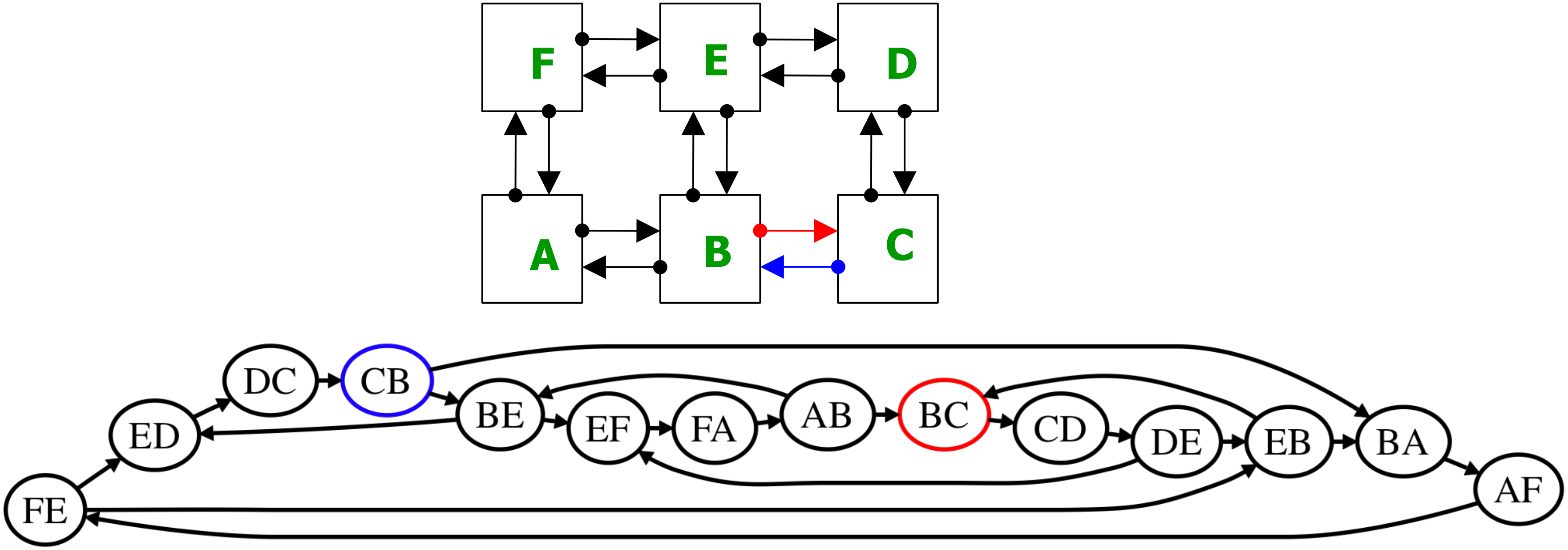


Total Turn Models = 16
Deadlock Free = 12
Unique (non-symmetrical) = 3

Deadlock!

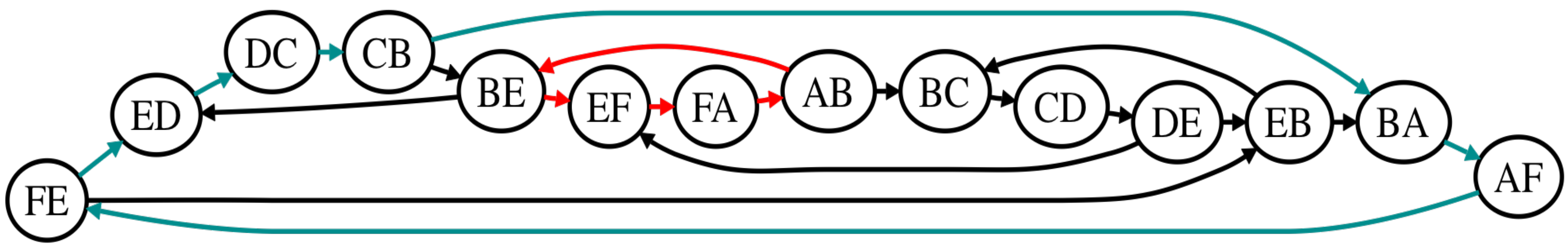
CHANNEL DEPENDENCY GRAPH (CDG)

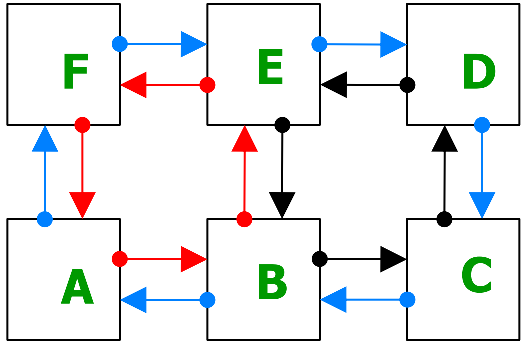
- Vertices represent network links (channels)
- Edges represent turns
  - *180º turns not allowed, e.g., AB → BA*



CYCLES IN THE CDG

The channel dependency graph D derived from the network topology may contain many *cycles*

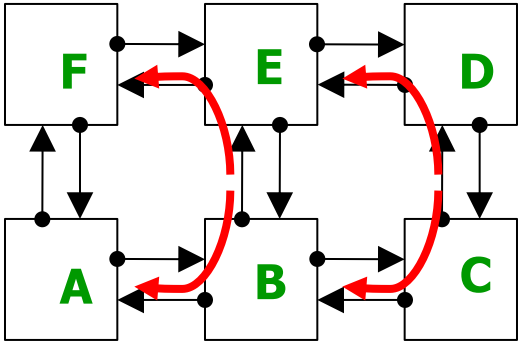




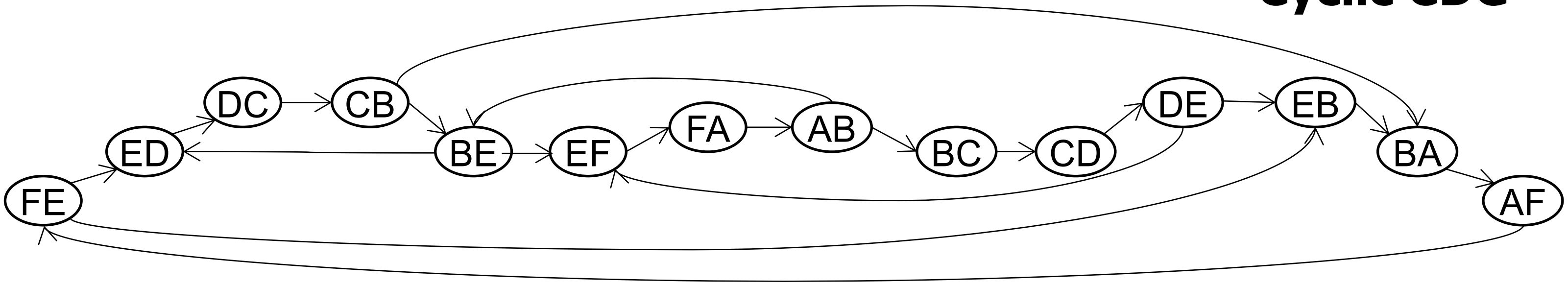
Flow routed through links AB, BE, EF
Flow routed through links EF, FA, AB
Deadlock!

Edges in CDG = Turns in Network
➔ Disallow/Delete certain edges in CDG

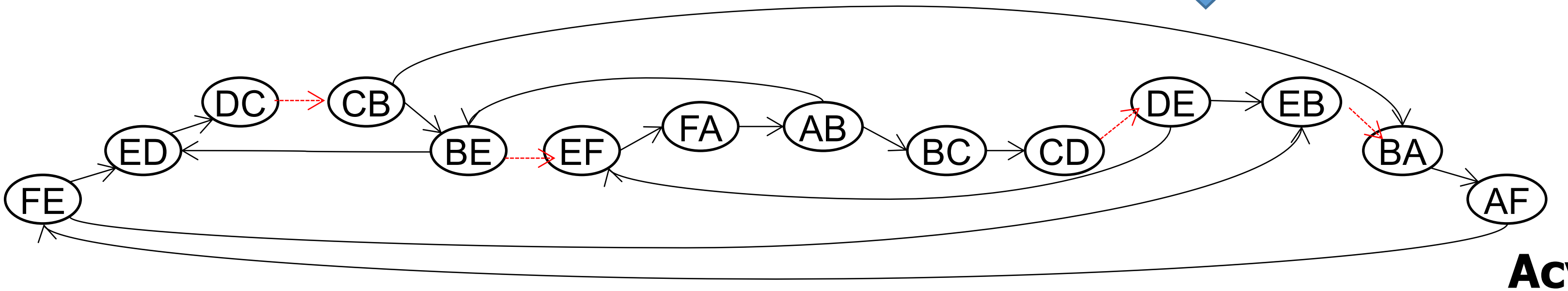
ACYCLIC CDG



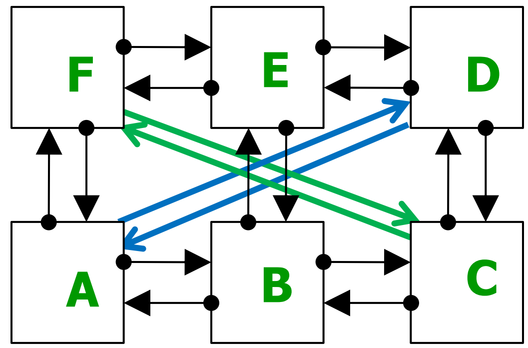
This is the West-first turn model!



Disable certain edges

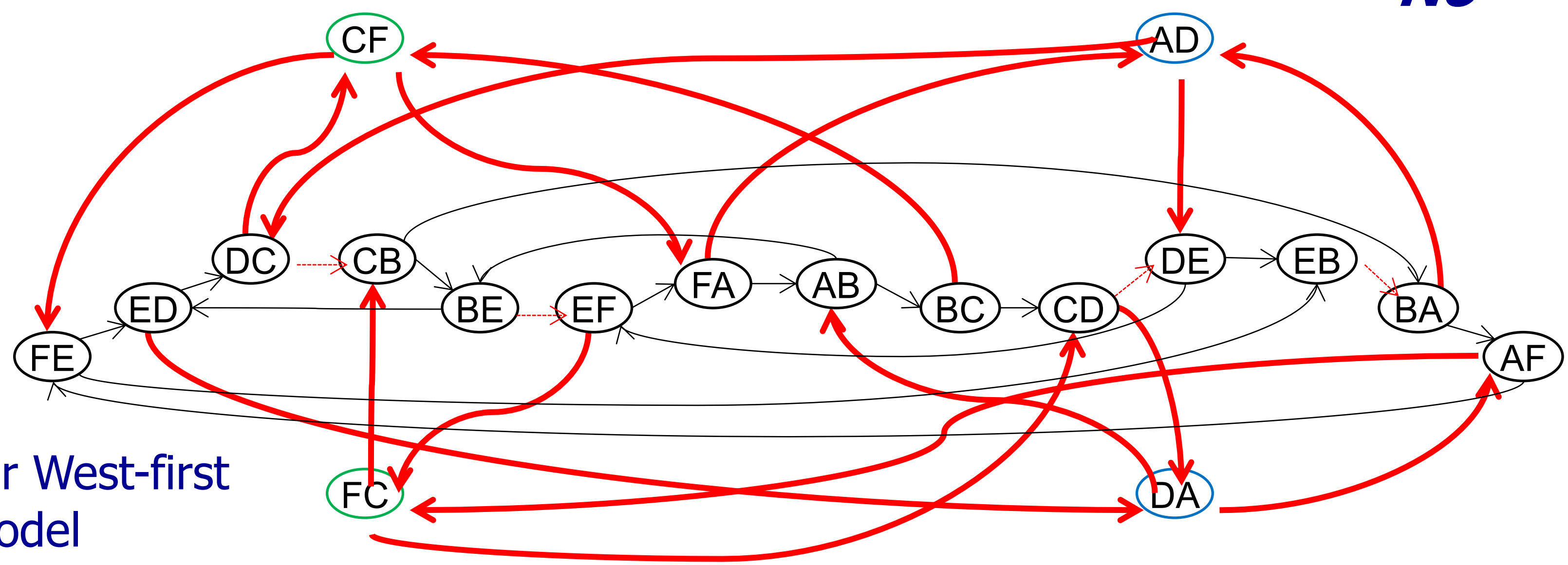


CDG FOR ARBITRARY TOPOLOGY



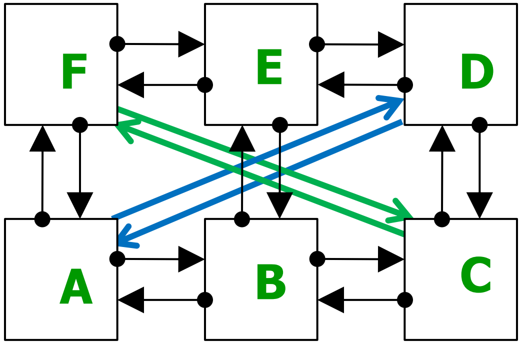
Deadlock free?

No

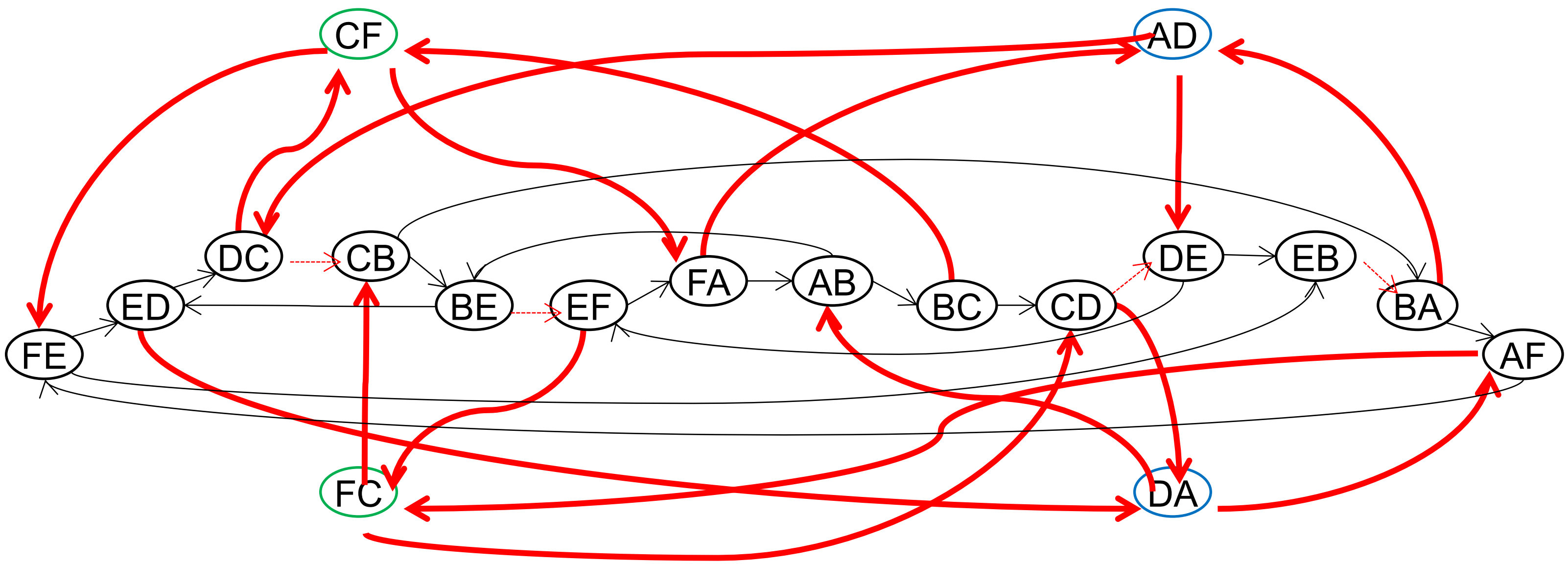


CDG for West-first turn model

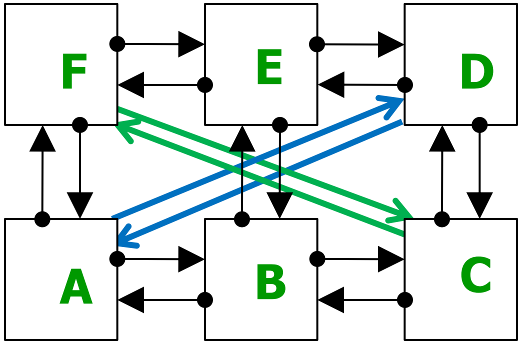
DEADLOCK-FREE ROUTING ALGORITHM



Suppose: Diagonal links should be traversed last (i.e., no edge from blue/green channel to black)



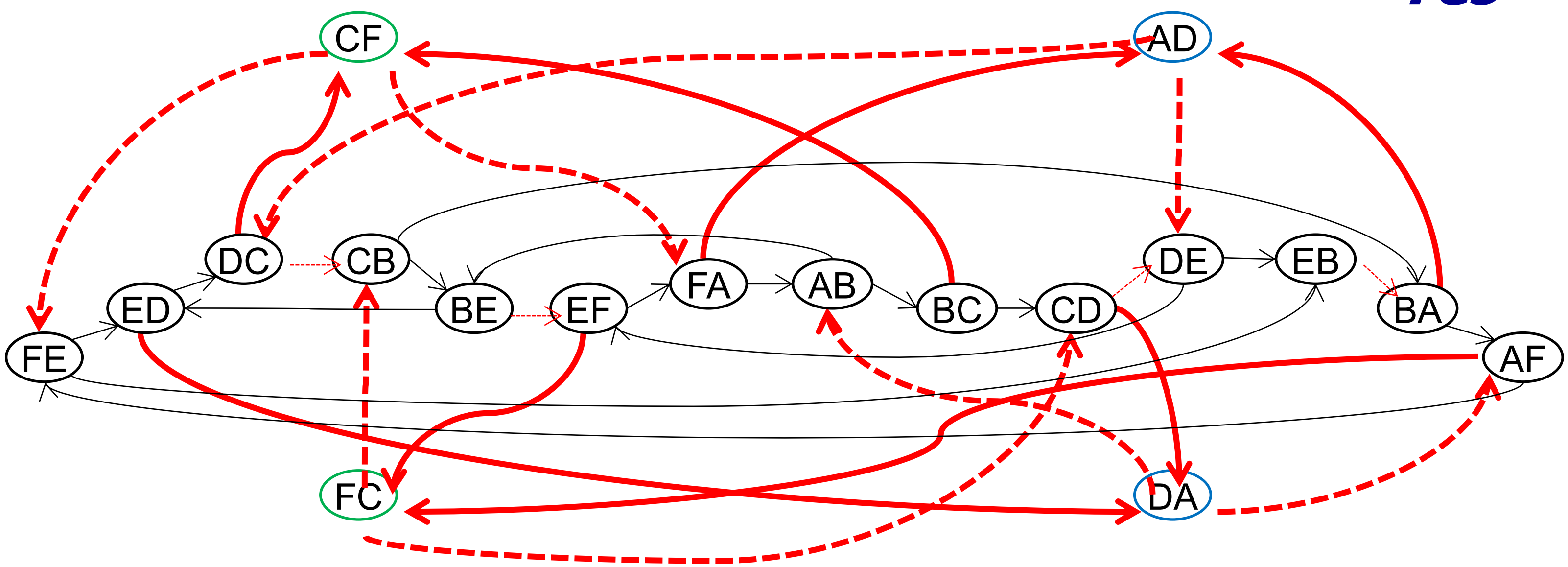
DEADLOCK-FREE ROUTING ALGORITHM



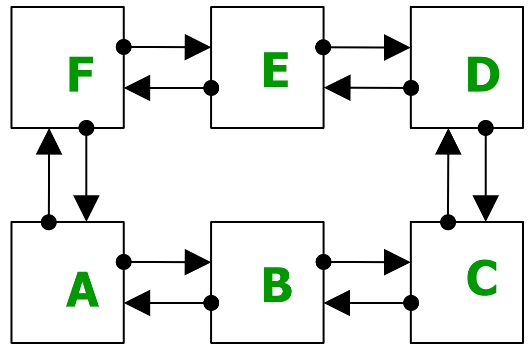
Suppose: Diagonal links should be traversed last (i.e., no edge from blue/green channel to black)

Deadlock free?

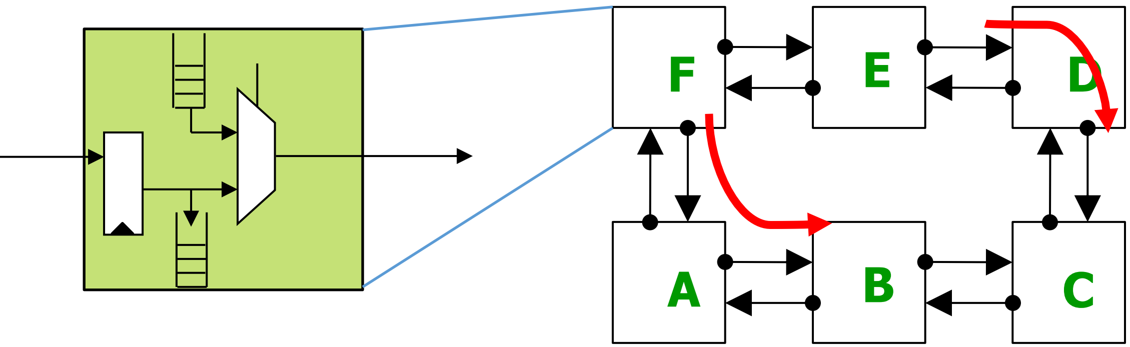
Yes



WHAT ABOUT A RING?



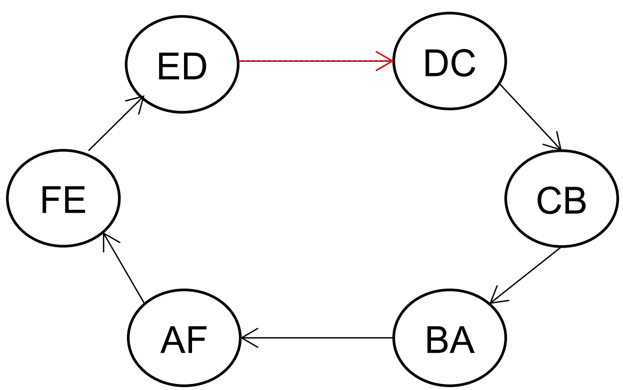
ACYCLIC CDG FOR A RING



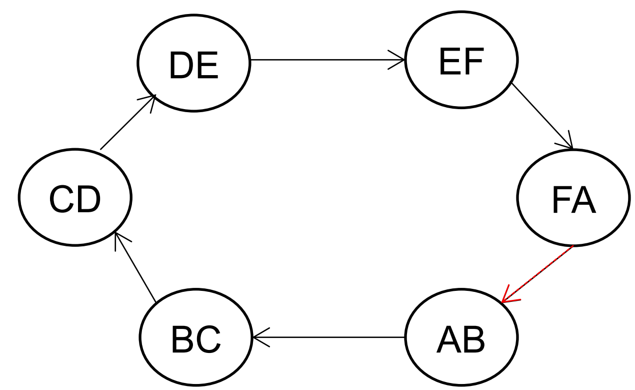
Route from E to C disabled
(E to D) and (D to C) allowed

Route from F to B disabled

Option 1

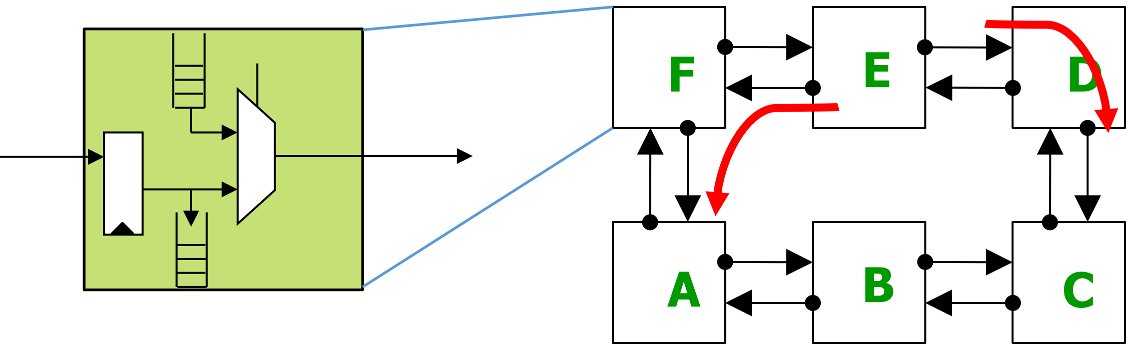


CDG



Problem? No route from E/F to B/C

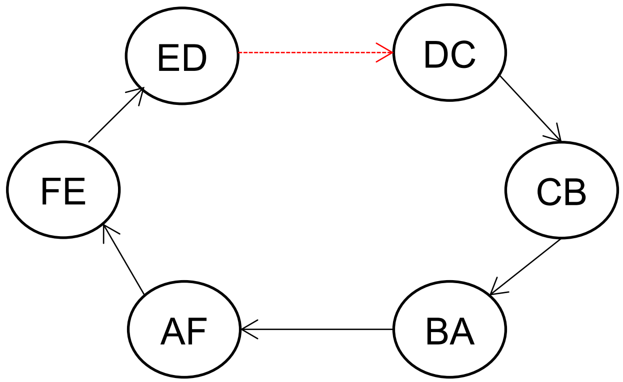
ACYCLIC CDG FOR A RING



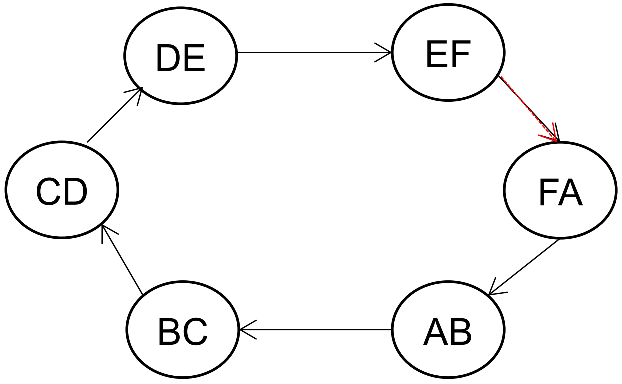
Route from E to C disabled
(E to D) and (D to C) allowed

Route from E to A disabled

Option 2

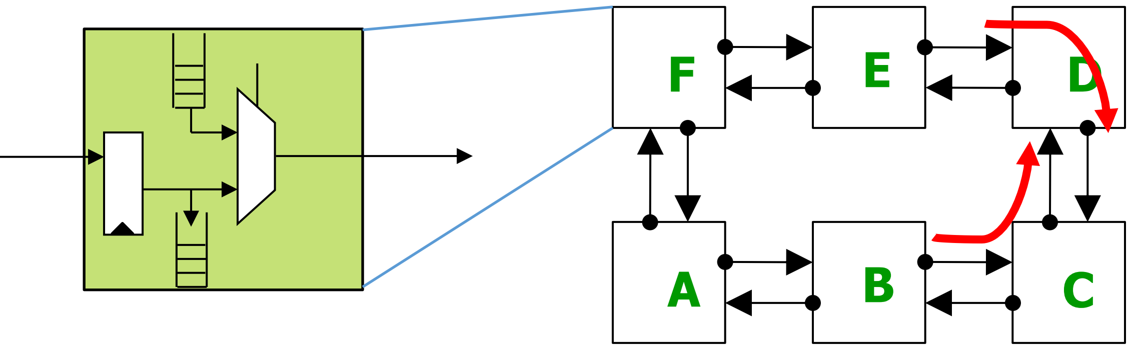


CDG



Problem? No route from E to A/B/C

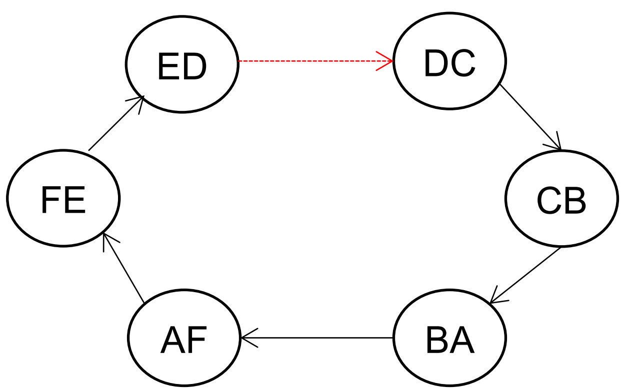
ACYCLIC CDG FOR A RING



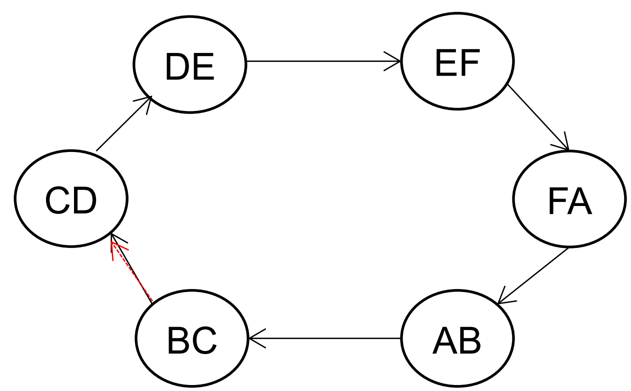
Route from E to C disabled
(E to D) and (D to C) allowed

Route from B to D disabled

Option 3



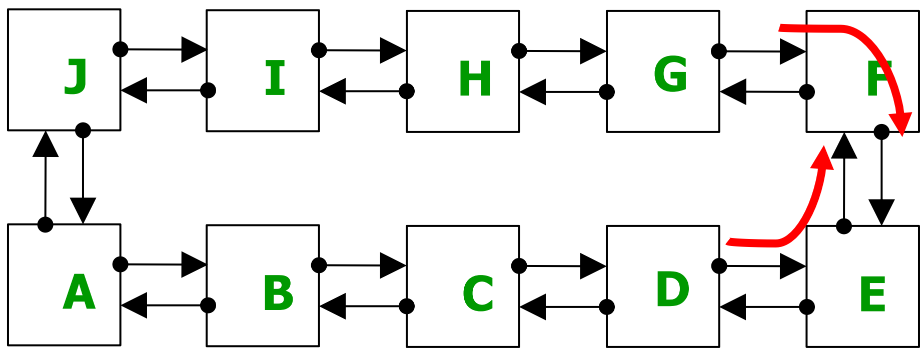
CDG



Acceptable CDG

Problem? E to C no longer minimal

ACYCLIC CDG FOR A LARGE RING

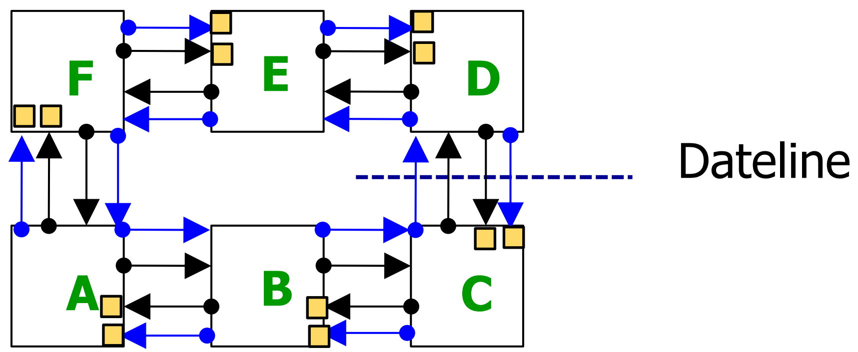


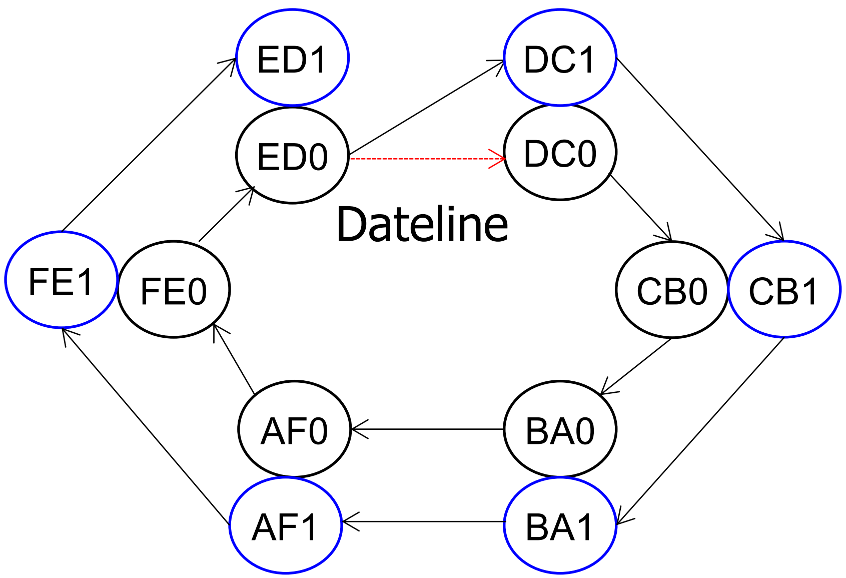
Problem?

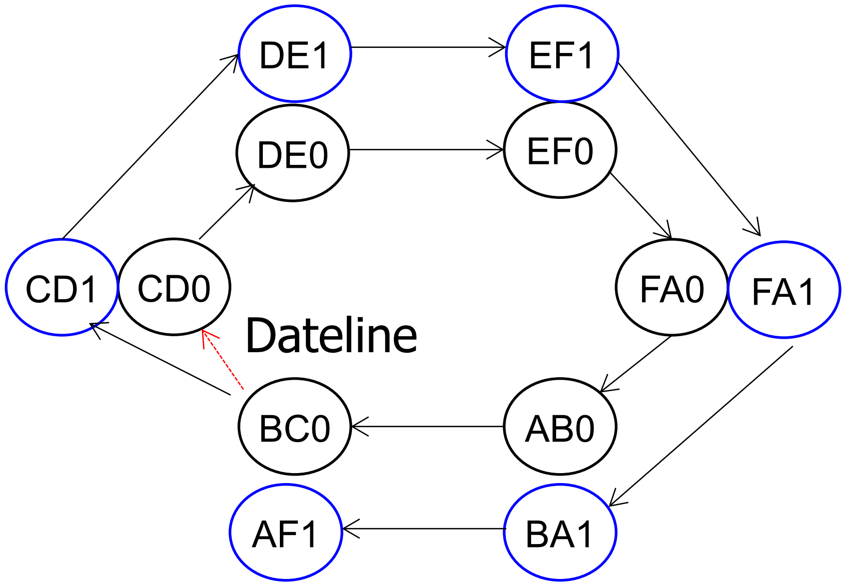
G, H, I have to take non-minimal paths to reach E!

D, C, B have to take non-minimal paths to reach F

SUPPOSE *TWO* CHANNELS

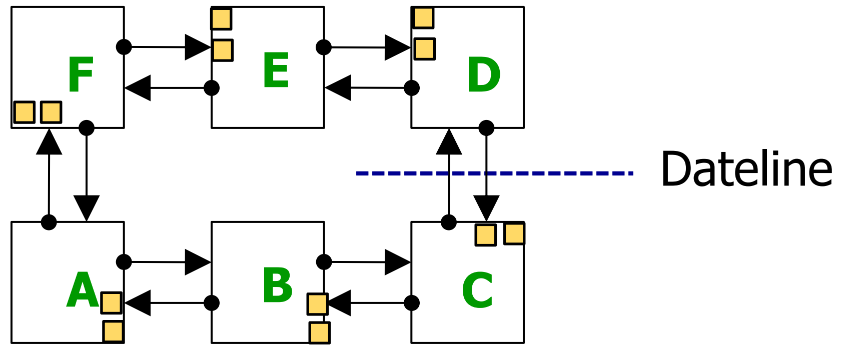




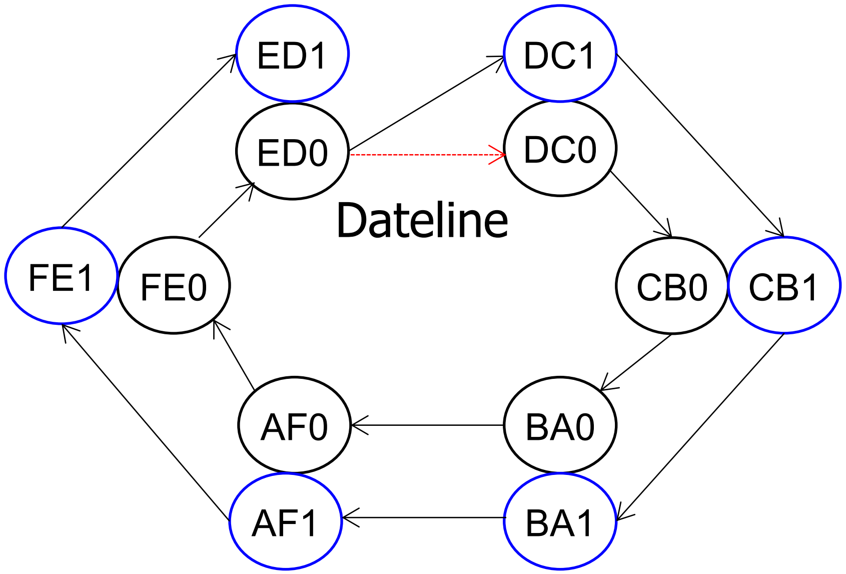


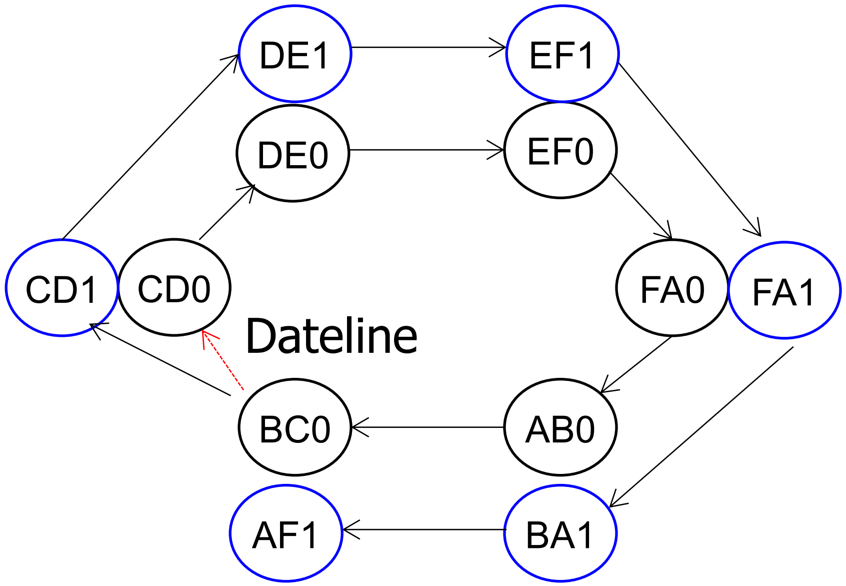
NEED NOT BE PHYSICAL CHANNELS

Need at least 2 classes of buffers - called **"Virtual Channels"**



Start in VC in Class0
After Dateline, jump to VC in Class1





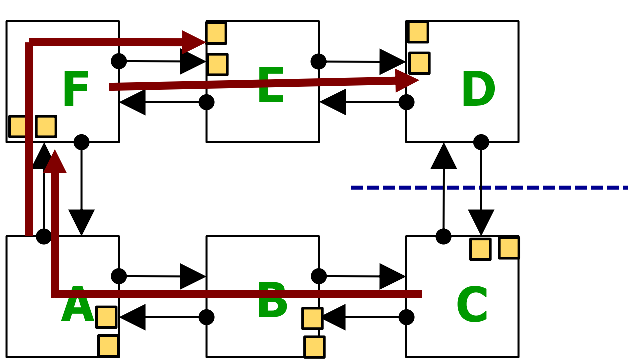
DEADLOCK AVOIDANCE

- Eliminate cycles in Resource Dependency Graph
  - Resource Ordering
    - Enforce a partial/total order on the resources, and insist that an agent acquire the resources in ascending order
    - Deadlock avoided since a cycle must contain at least one agent holding a higher numbered resource waiting for a lower-numbered resource which is not allowed by the ordering allocation
  - Implementation
    - Restrict certain routes so that a higher numbered resource cannot wait for a lower numbered resource
    - Partition the buffers at each node such that they belong to different **resource classes**. A packet only any route can only **acquire buffers in ascending order of resource class**

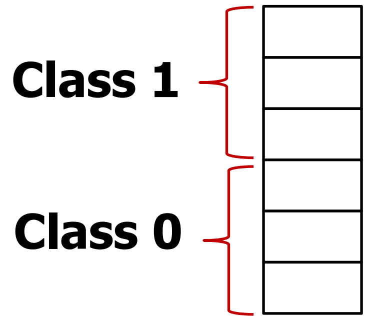
USING VCS FOR DEADLOCK AVOIDANCE

- Ring
  - Use VC from class 0 before dateline
  - Use VC from class 1 after dateline
- Fully-Oblivious (e.g., O1turn)
  - Use VC 0 for XY, VC 1 for YX
- Fully-Adaptive Routing (no turns restricted)
  - Use VC from class 0 before turning
  - Use VC from class 1 after turning
- Valiant's Routing Algorithm
  - DOR over VC in class 0 from source till intermediate
  - DOR over VC in class 1 from intermediate to destination

VC UTILIZATION

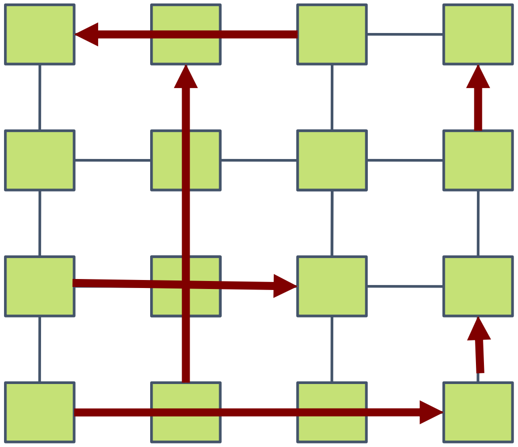


Ring with Dateline



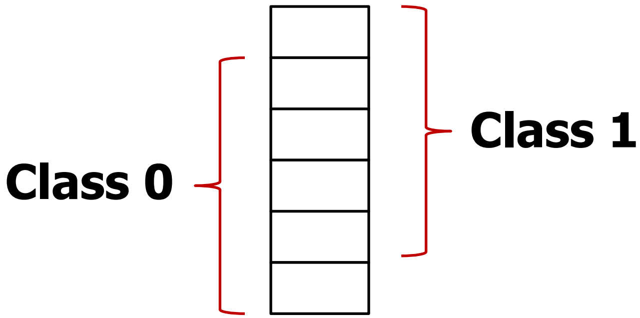
Problem? Packet on Ring never crosses dateline
Packet on Mesh does not make any turns

VC from Class 1 never used!



Mesh with O1Turn

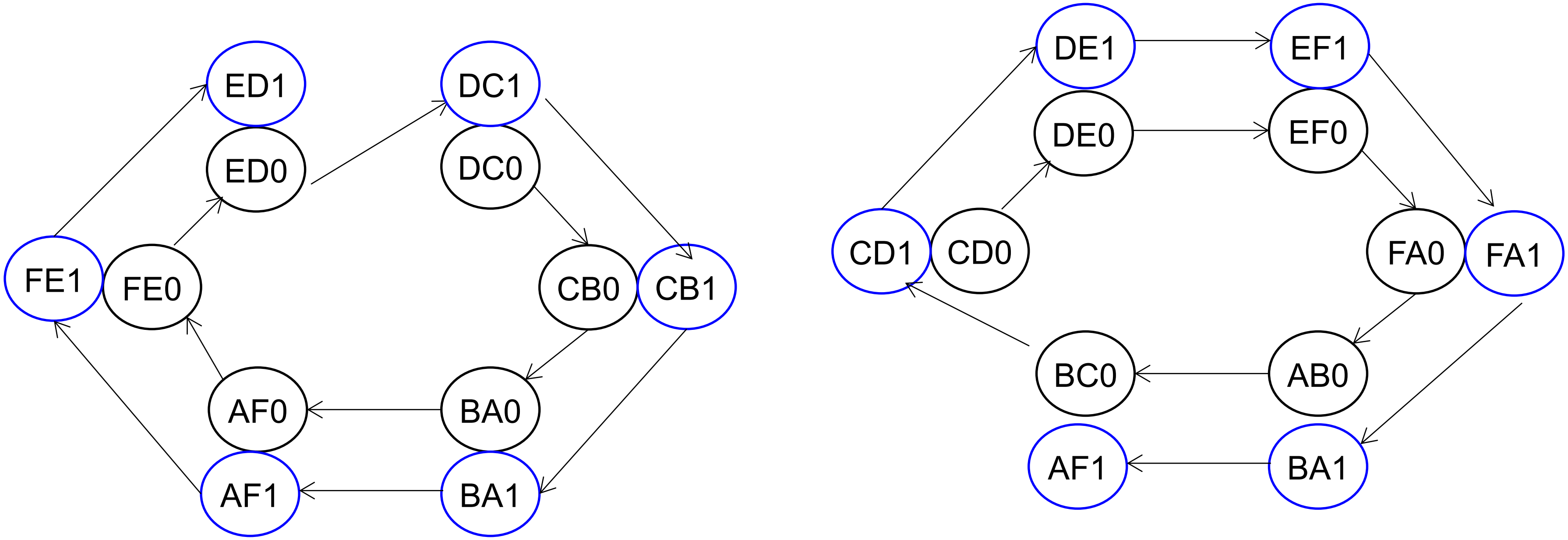
Solution: Overlapping Resource Classes



As long at least one buffer per class, can guarantee deadlock freedom

DEADLOCK AVOIDANCE

- So far, we said deadlock is avoided if cycles eliminated in Channel Dependence Graph
  - Remove cycles via turn restriction
  - Convert cyclic CDG into a spiral using VCs
    - Called *extended* CDG



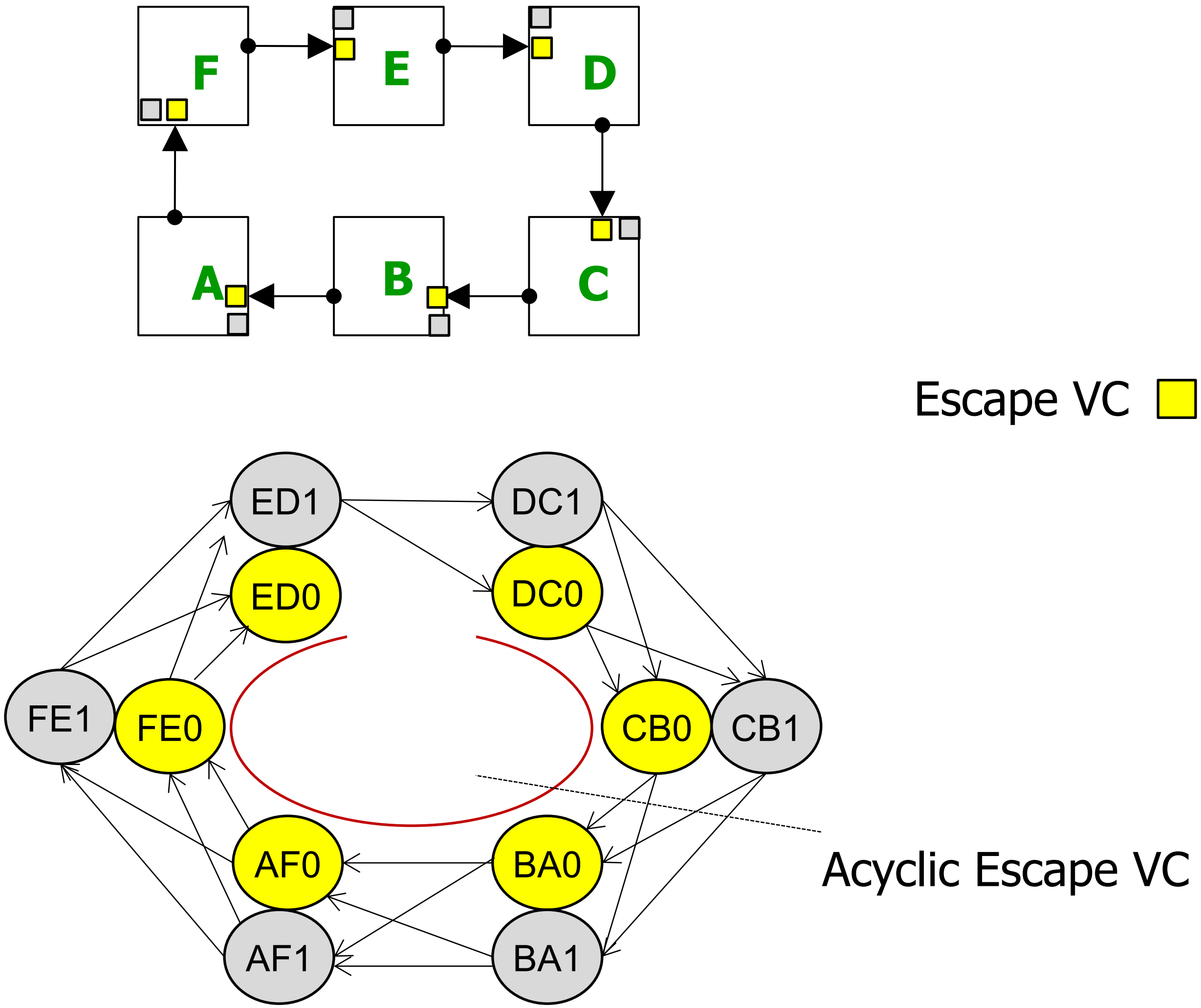
DEADLOCK AVOIDANCE

- So far, we said deadlock is avoided if cycles eliminated in Channel Dependence Graph
  - Remove cycles via turn restriction
  - Convert cyclic CDG into a spiral using VCs
    - Called *extended* CDG

- However, it is possible for a (extended) CDG to have cycles and still be deadlock-free (Duato*, 1993)
  - As long as the cycle connects to some sub-graph within the (extended) CDG that is acyclic
  - Known as the *escape path* or *escape VC*

*José Duato. A new theory of deadlock-free adaptive routing in wormhole networks. IEEE Transactions on Parallel and Distributed Systems, 4(12):1320–1331, December 1993.

CDG FOR ESCAPE VCS



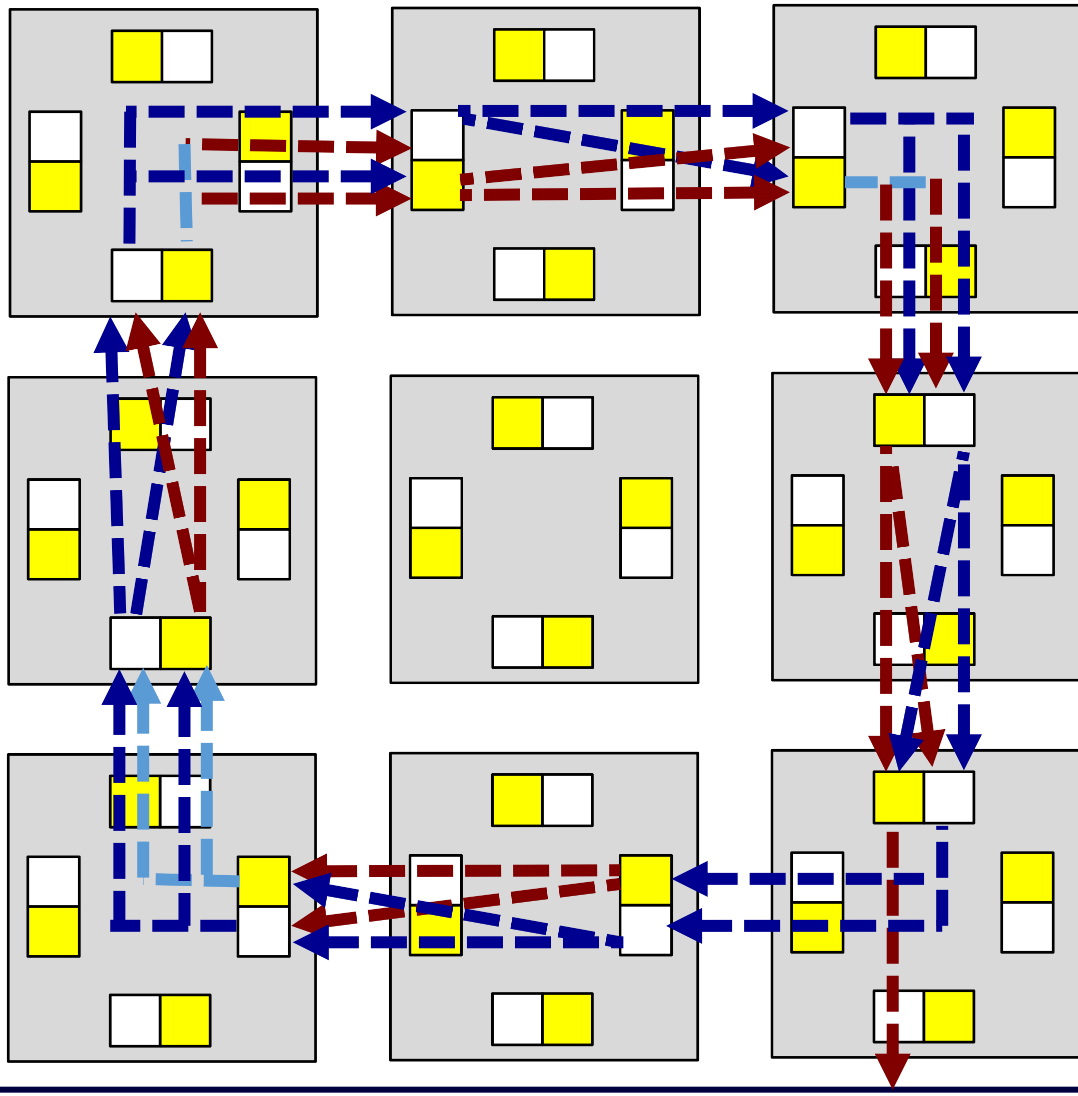
WHY ESCAPE VCS WORK

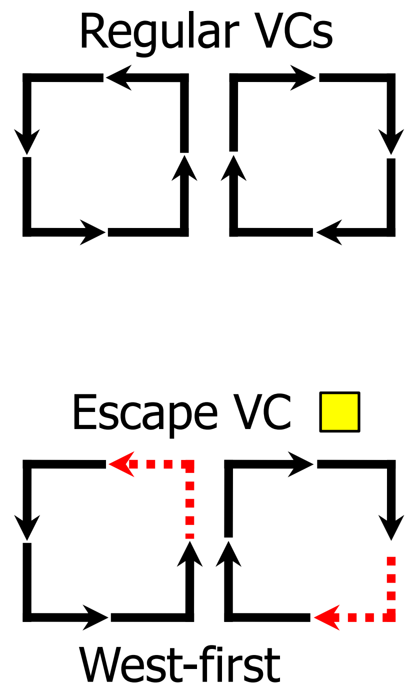
- Intuitively, at least one packet in the cycle has an option to take an acyclic route
  - Packets should not wait on any *specific* channel
  - If allocation is fair, escape VCs guaranteed to show up eventually
- Use of escape channels by a message is not unidirectional
  - If a message enters the escape network it can move back to the adaptive network, and vice versa, if minimal* routes
    - *for non-minimal routes, message has to continue on escape VC once it gets in, without going back to the adaptive VCs

EXAMPLE

- Consider a 2D Mesh with 8 VCs and minimal routing
  - VC 1-7 can use any arbitrary minimal routing
    - Cyclic CDG
  - VC 0 (escape VC) is restricted to DOR (provides escape path)
    - Acyclic CDG
  - As long as a packet can allocate all VCs fairly, there will always be an escape path available in case the network deadlocks

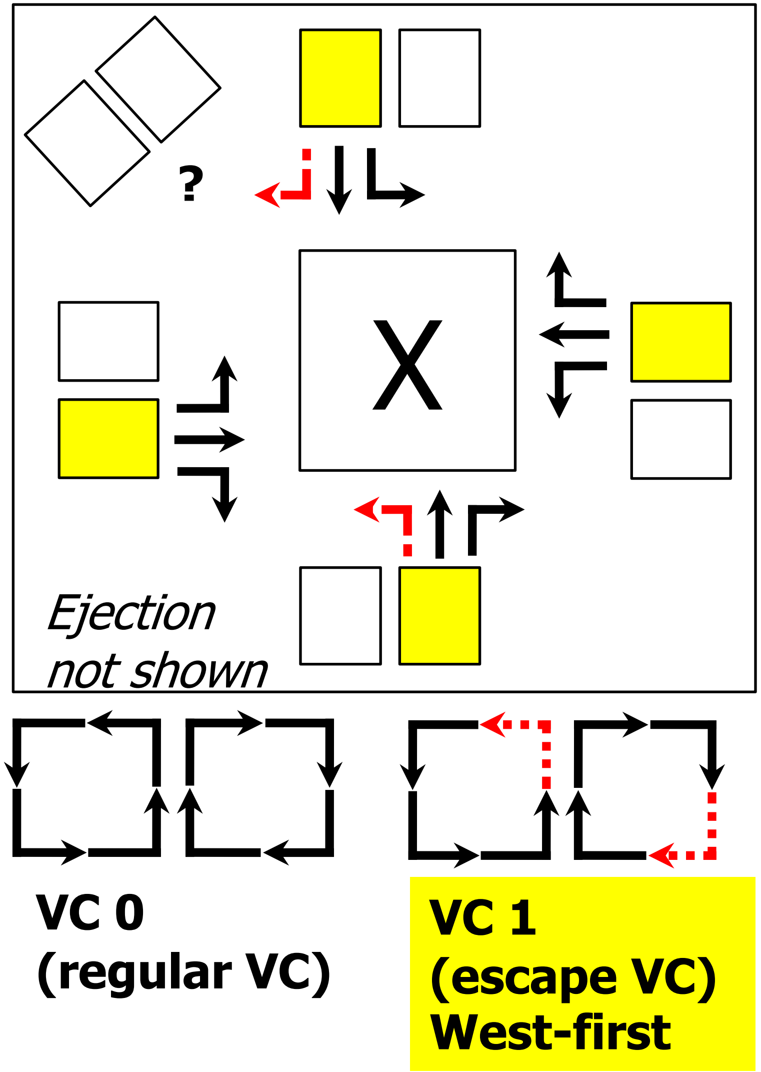
EXAMPLE





Deadlock-free
escape path

RULES FOR GETTING IN/OUT OF ESCAPE VCS



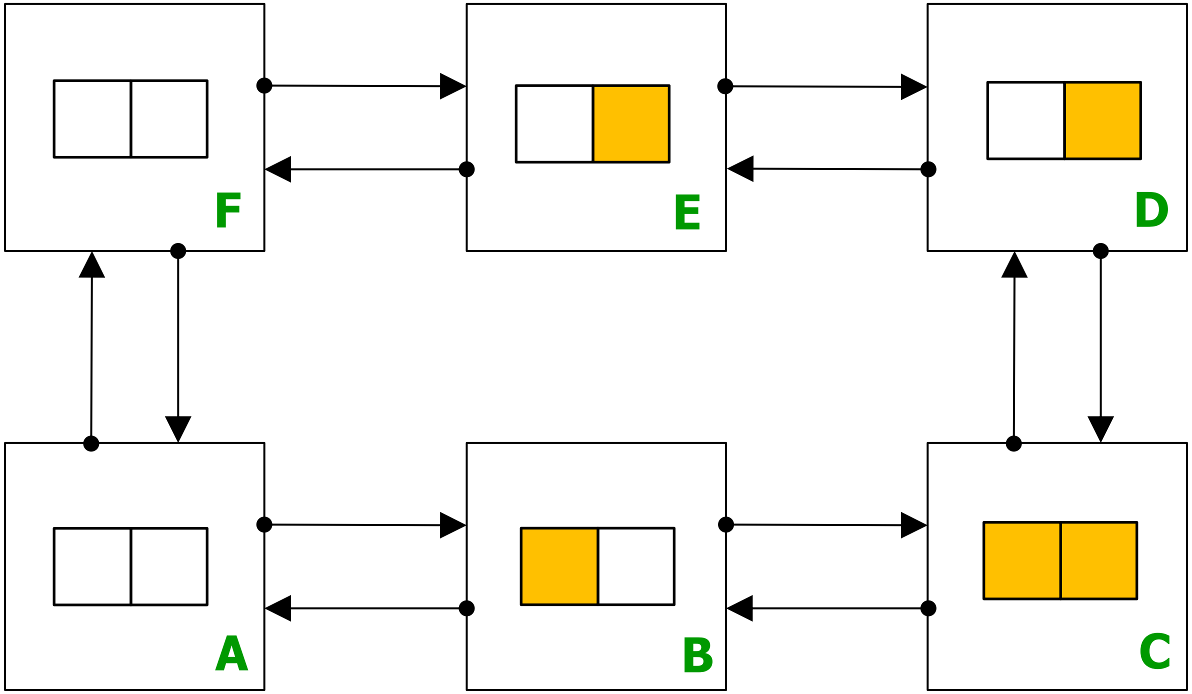
- The escape VC should always makes forward progress!
  - A flit that is going NW or SW should never enter a router from the S or N port in escape VC, else S→W or N→W turn is inevitable
    - How to guarantee this?
      - When selecting VC at previous router
      - Lab 3!

DEADLOCK AVOIDANCE SUMMARY

- Eliminate cycles in Channel Dependency Graph
  - Routing Restrictions (e.g., Turn Model in Mesh)
    - Acyclic CDG
  - Buffer Assignment
    - Acquire new VC every time a "cyclic turn" is made
      - e.g., Dateline in Ring, XY in VC 0, YX in VC 1 in Mesh, …
      - Acyclic Extended CDG
    - Escape VCs
      - Cyclic CDG (regular VC) + Acyclic sub-graph (Escape VC)
- Can we avoid deadlocks even if CDG is cyclic?
  - What if we guarantee that a dependence cycle will never get created at runtime by clever *flow-control*?

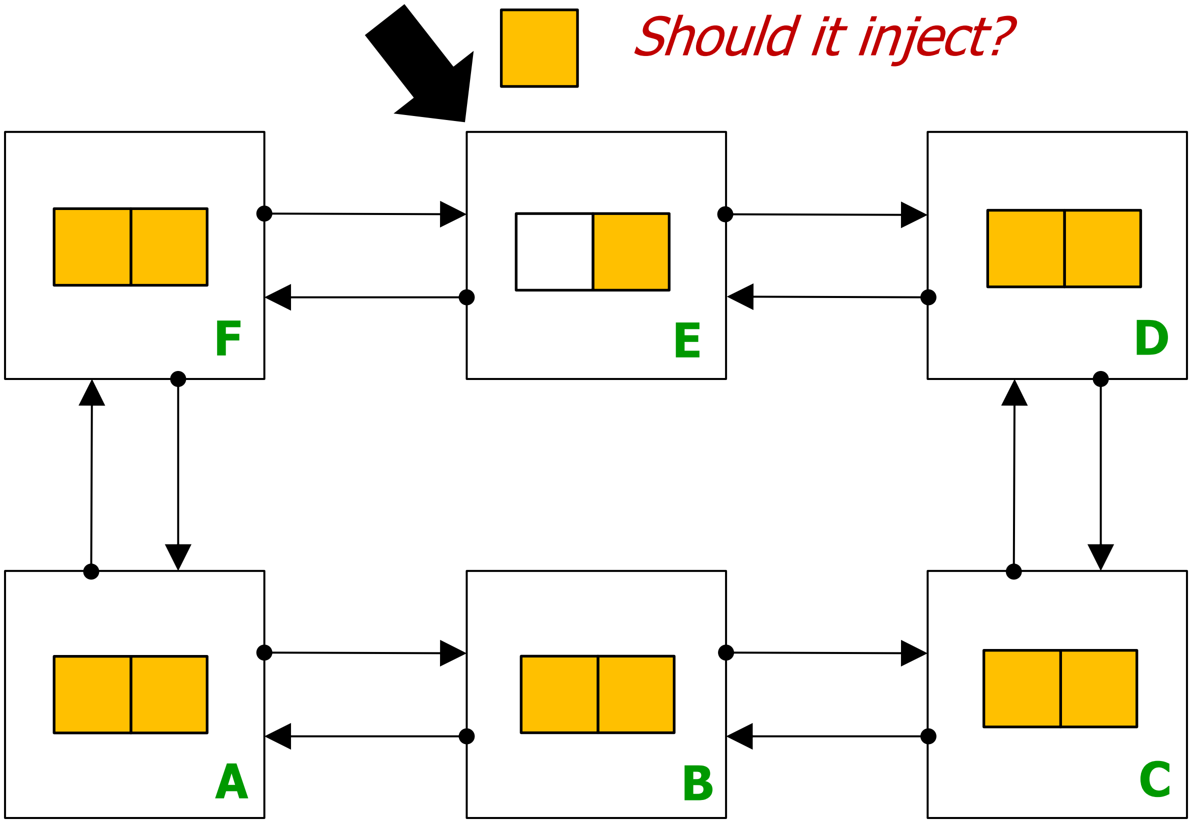
BUBBLE FLOW CONTROL

Ring Traversal Rule:
traverse if one bubble free



V. Puente et al. ***The adaptive bubble router.*** *Journal of Parallel and Distributed Computing, 2001.*

BUBBLE FLOW CONTROL

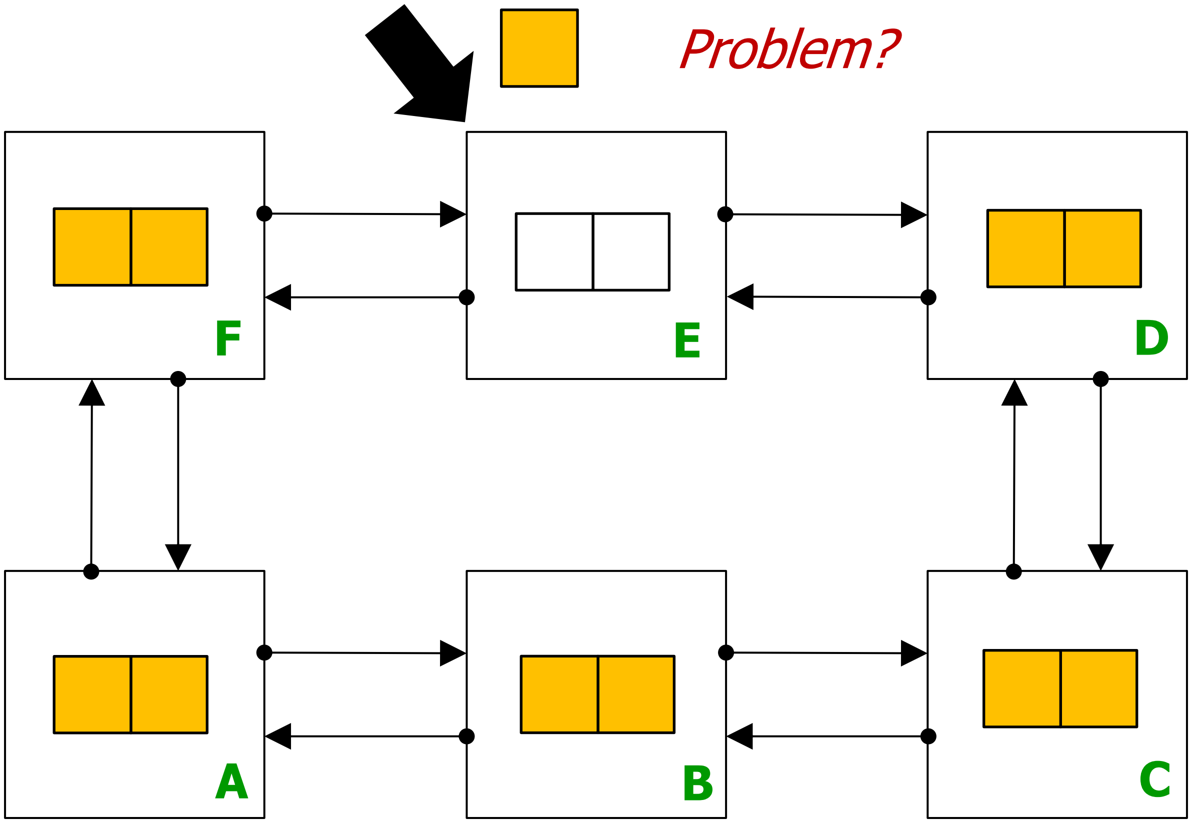


Ring Traversal Rule:
traverse if one bubble free

BFC Injection Rule:
only inject if 2 bubbles free.

V. Puente et al. ***The adaptive bubble router.*** *Journal of Parallel and Distributed Computing, 2001.*

BUBBLE FLOW CONTROL

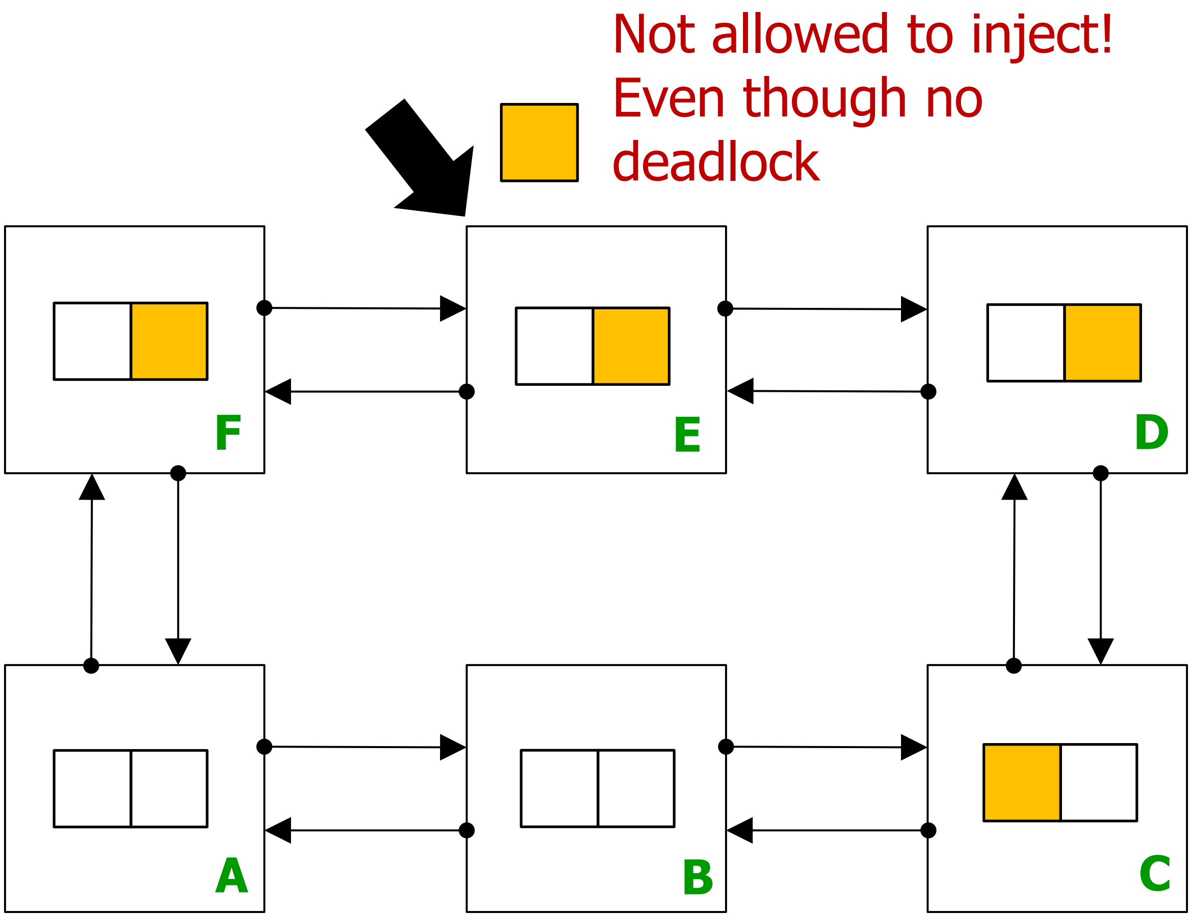


Ring Traversal Rule:
traverse if one bubble free

BFC Injection Rule:
only inject if 2 bubbles free.

V. Puente et al. ***The adaptive bubble router.*** *Journal of Parallel and Distributed Computing, 2001.*

BUBBLE FLOW CONTROL

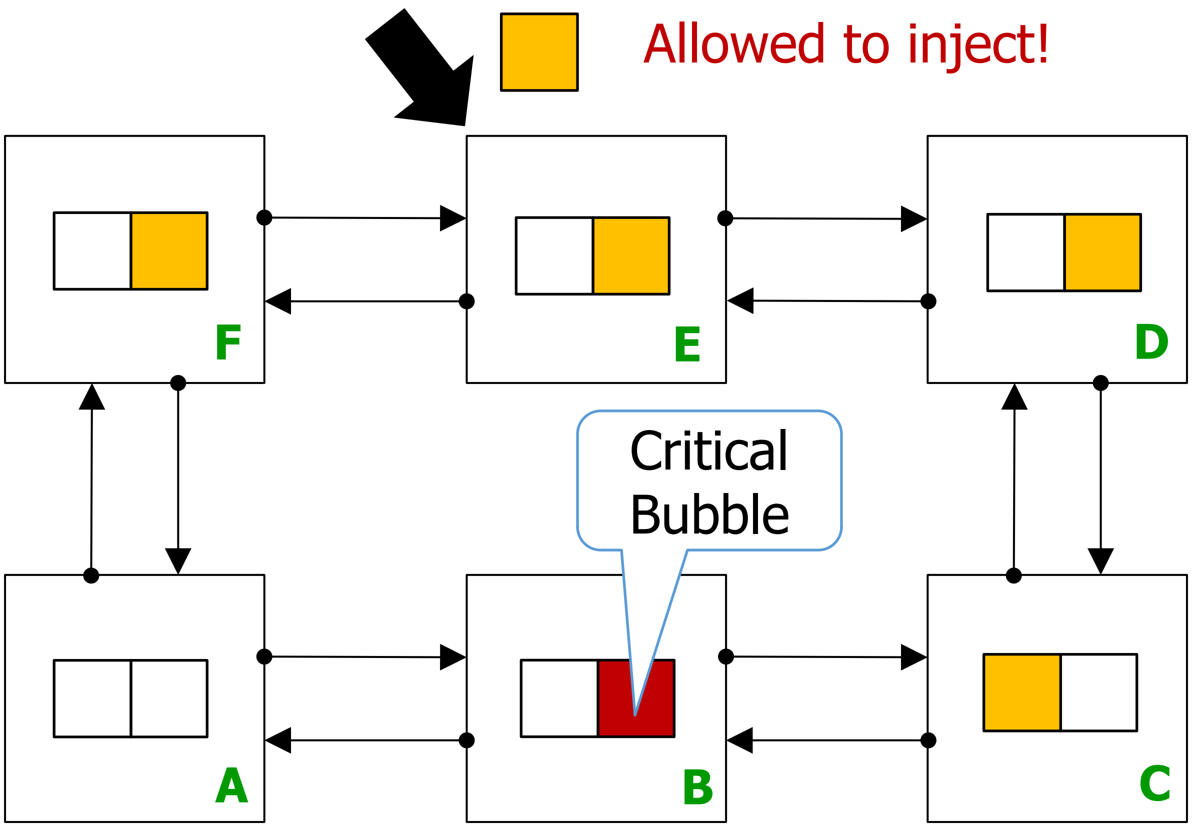


Ring Traversal Rule:
traverse if one bubble free

BFC Injection Rule:
only inject if 2 bubbles free.

V. Puente et al. ***The adaptive bubble router.*** *Journal of Parallel and Distributed Computing, 2001.*

CRITICAL BUBBLE FLOW CONTROL



Ring Traversal Rule:
traverse if one bubble free

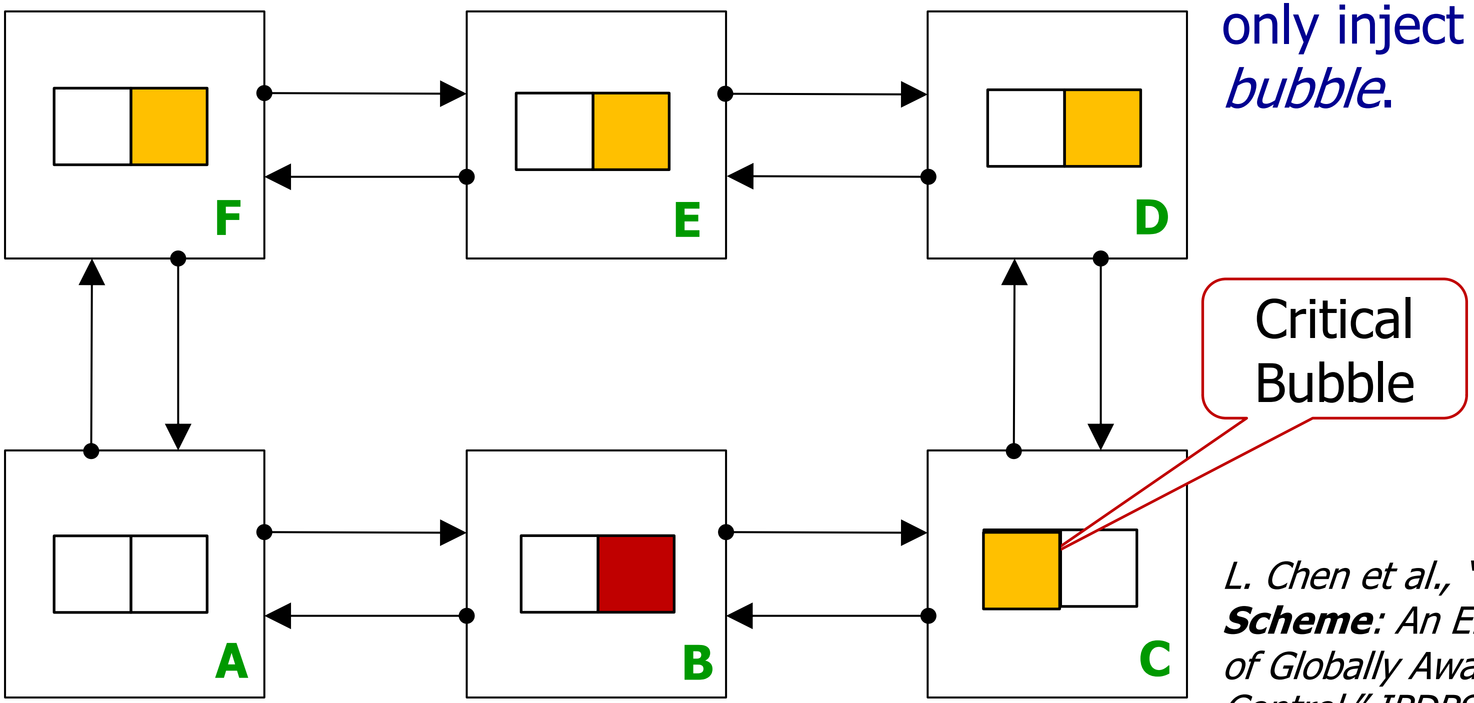
CBFC Injection Rule:
only inject if not *critical bubble*.

*L. Chen et al., "**Critical Bubble Scheme**: An Efficient Implementation of Globally Aware Network Flow Control," IPDPS 2011*

CRITICAL BUBBLE FLOW CONTROL

How does critical bubble move?

If flit moves into critical bubble, its own buffer becomes new critical bubble



Ring Traversal Rule: traverse if one bubble free

CBFC Injection Rule: only inject if not *critical bubble*.

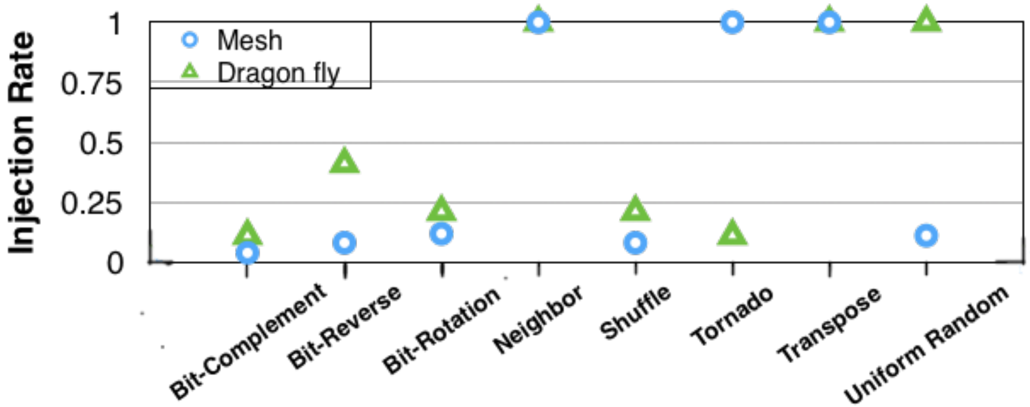
*L. Chen et al., "**Critical Bubble Scheme**: An Efficient Implementation of Globally Aware Network Flow Control," IPDPS 2011*

DEALING WITH DEADLOCKS

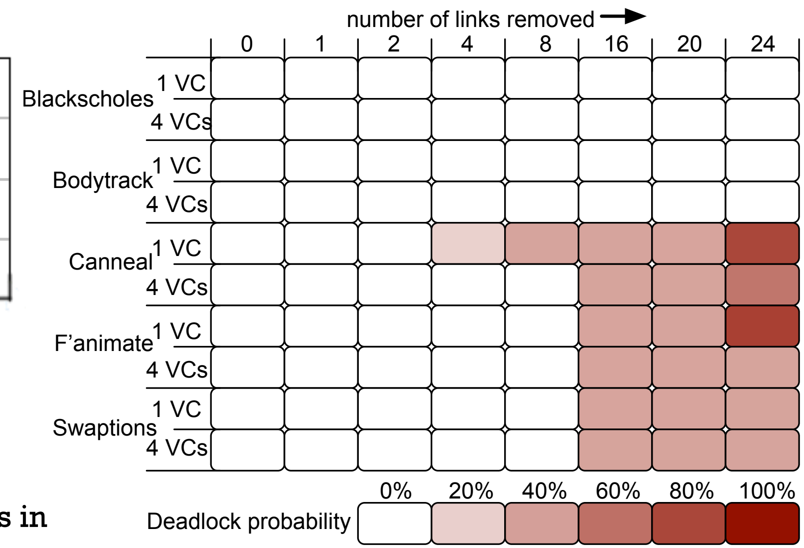
- **Proactive / Avoidance**
  - Guarantee that the network will never deadlock
  - Almost all modern networks use deadlock avoidance
- **Reactive / Recovery**
  - Detect deadlock and correct
- **Subactive**
  - Introduce periodic forced movement among packets

DEADLOCK RECOVERY - MOTIVATION

Deadlocks are rare!



Minimum injection rate (flits/node/cycle) at which 64-core Mesh and 1024-node Dragon-fly deadlock with different traffic patterns in 100K cycles with 3 VCs per port and 1-flit packets



But -- Need a solution for ***functional correctness!***

CHALLENGES WITH DEADLOCK AVOIDANCE

- Performance
  - due to Routing Restrictions in all VCs / subset of VCs
- Area/Power
  - Need additional Virtual Channels (buffers) to compensate

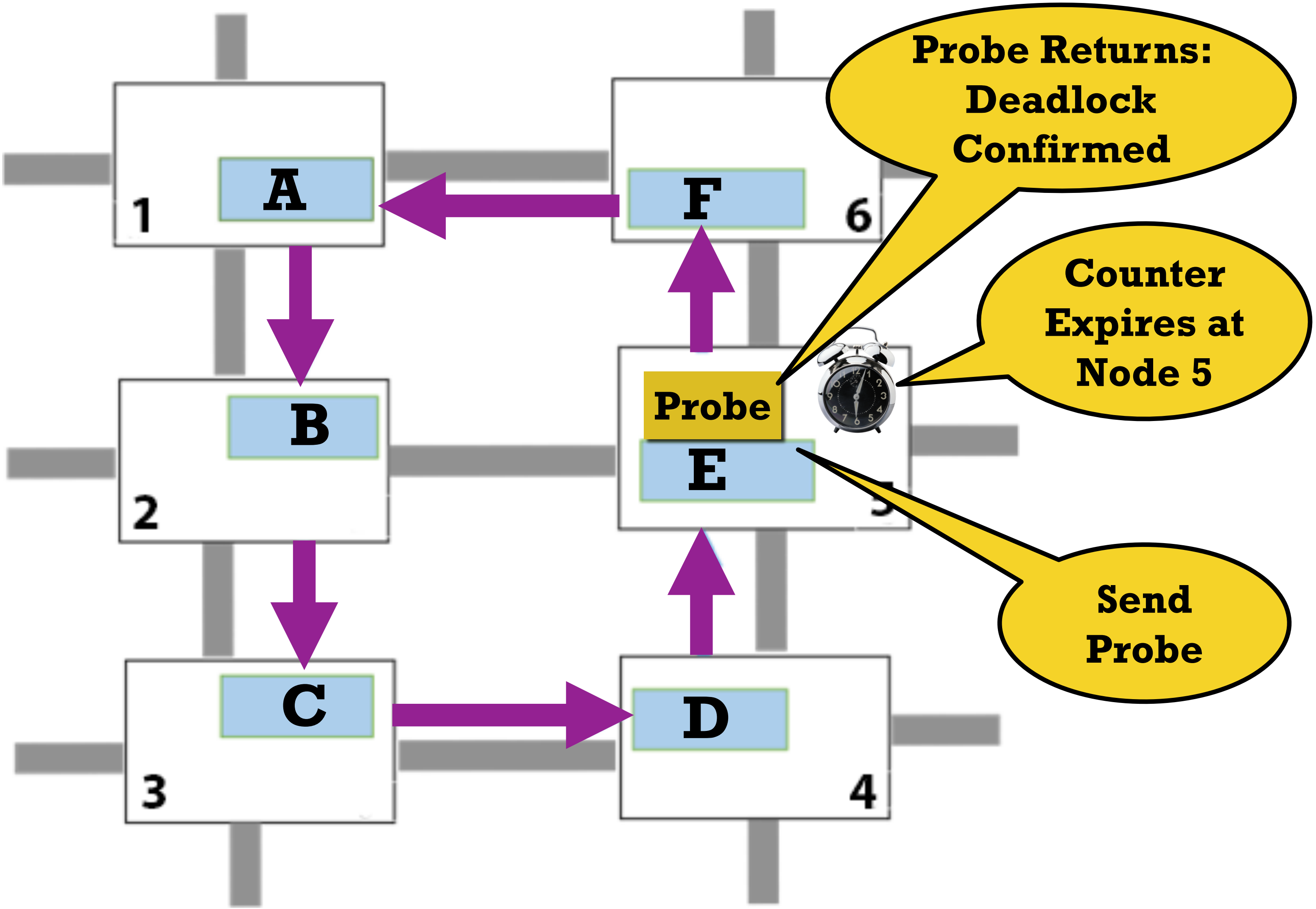
DEADLOCK RECOVERY

- Two phases
  - **Detection:**
    - E.g., timeouts attached with each resource
      - Can lead to false positives
  - **Recovery:**
    - Regressive – remove packets/connections that are deadlocked
      - E.g., drop packets after timeout
    - Progressive – recover without removing packets/connections
      - E.g. shared escape buffer to drain deadlocked packets
        - DISHA [ISCA 95], Static Bubble [HPCA 2017]
      - Coordinated Movement
        - SPIN [ISCA 2018]

DEADLOCK DETECTION

- Use counters.
- Placed at every node at design time.
  - Can be optimized further by exploiting topology symmetry (Static Bubble [HPCA 2017])
- If packet does not leave in threshold time (configurable), it indicates a potential deadlock.
  - Counter expired → Send probe to verify deadlock.

PROBE MESSAGE



PROBE MESSAGE

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

- Probe Traversal Mechanism
  - Drop Probe
    - If input port has at least one free VC
    - If input port has at least one VC pointing to ejection port
      - Ejection port guaranteed to eventually eject packet
      - Known as "Consumption Assumption"
  - Fork Probe
    - If none of the drop conditions are met
    - Fork probe out of all output ports that VCs at the input port are waiting on

- If Probe returns to sender:
  - Cyclic buffer dependence, hence deadlock.
    - There may be false positives

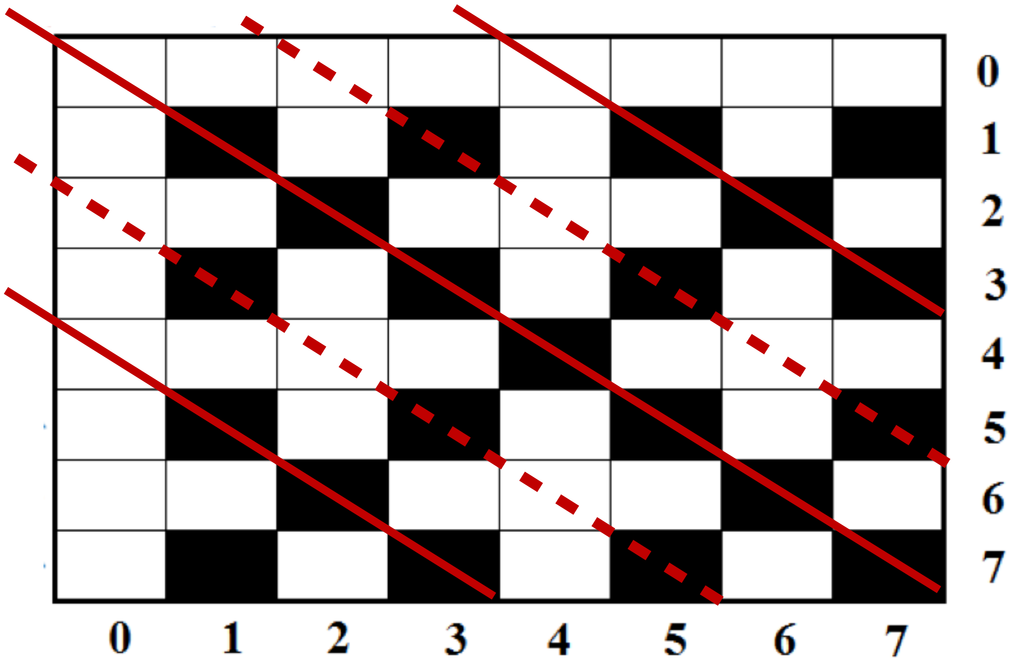
- Next, send other special messages to handle recovery

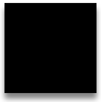
STATIC BUBBLE

Static Bubble: A Framework for Deadlock-free Irregular On-chip Topologies
Aniruddh Ramrakhyani and Tushar Krishna
In Proc of the 23rd IEEE International Symposium on High-Performance Computer Architecture ***(HPCA)****, Feb 2017*

STATIC BUBBLES

- Place static bubbles at design time to guarantee deadlock-freedom for any irregular runtime topology.

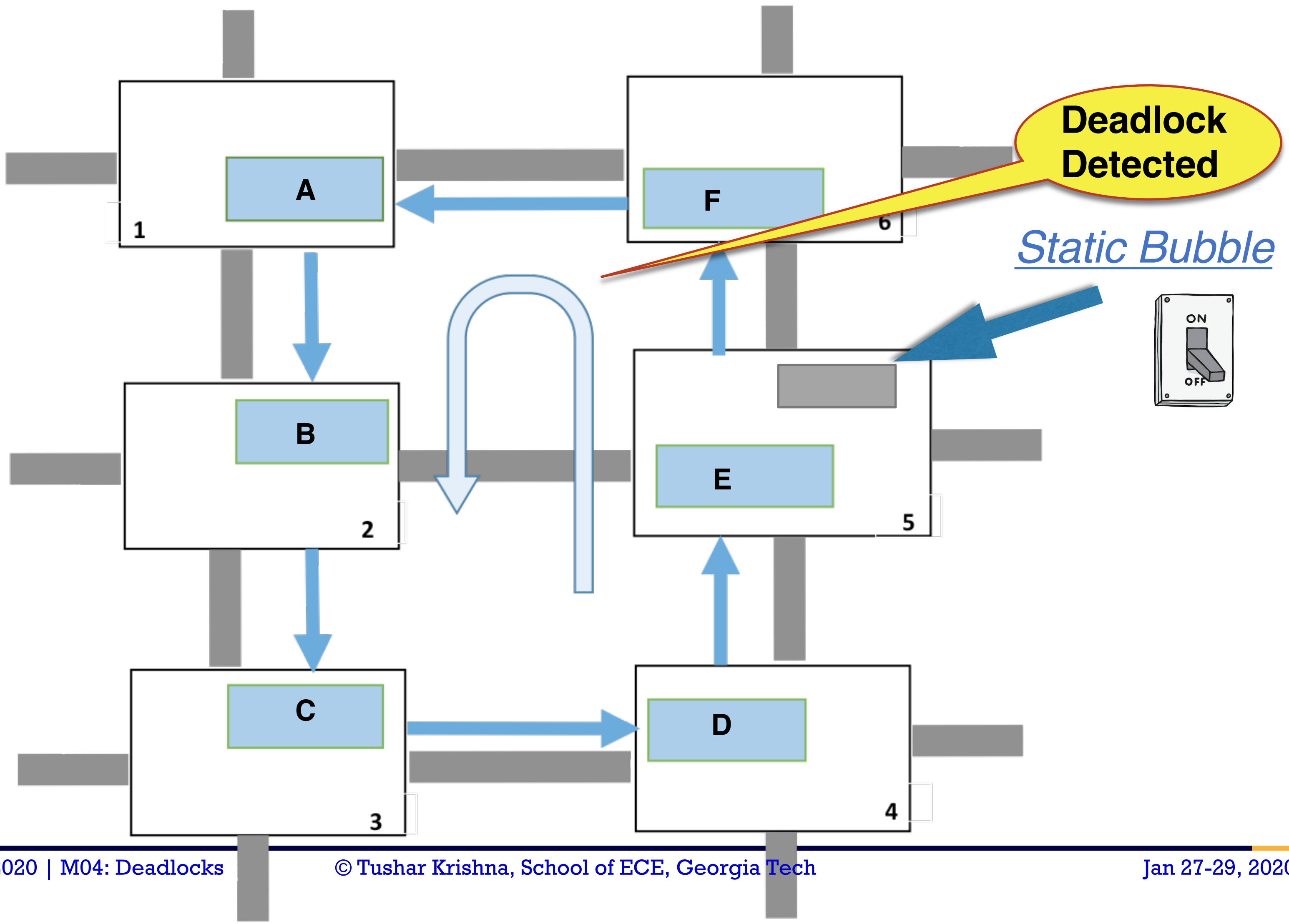




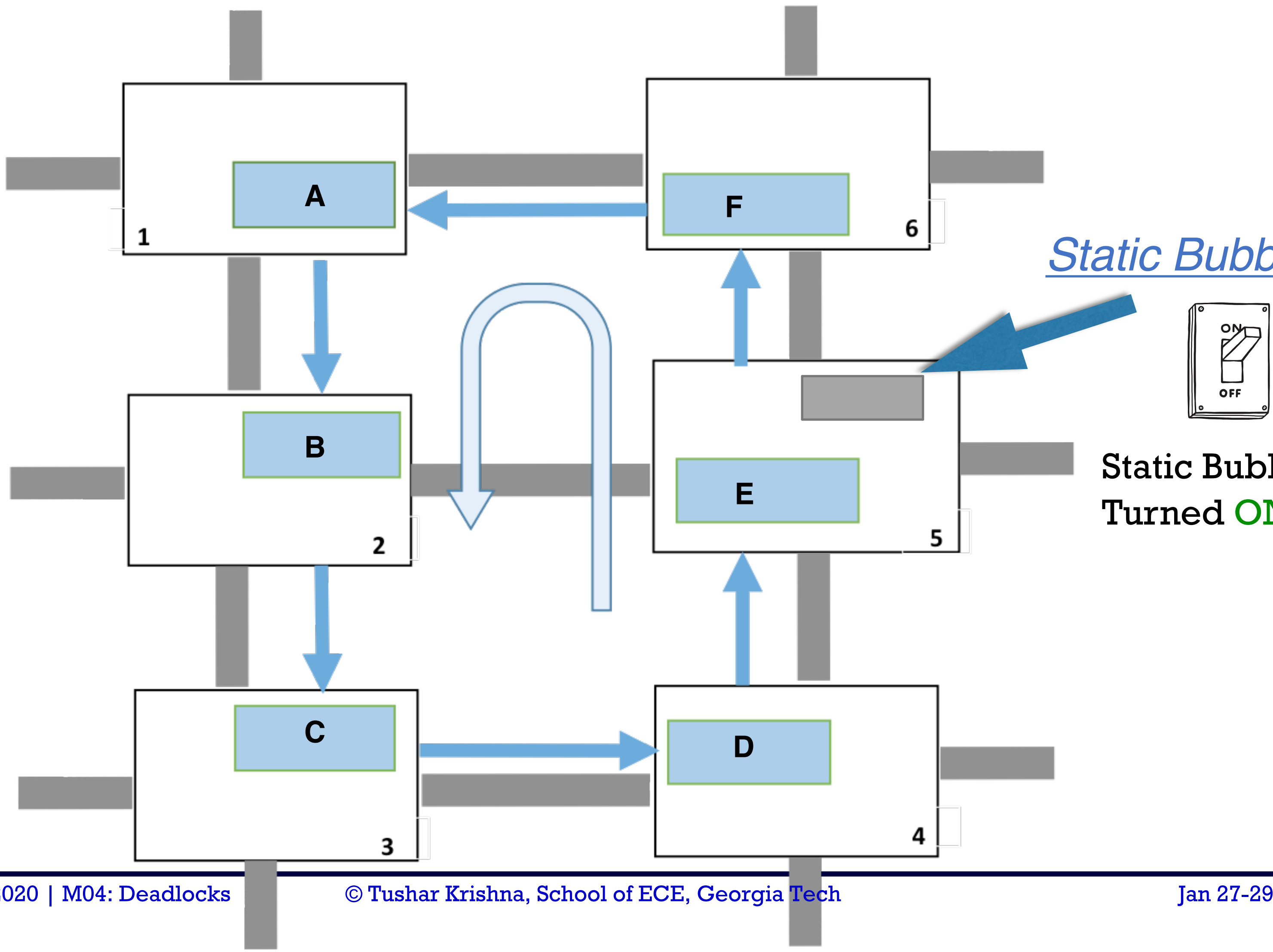
Static Bubble Node : contains *one* Static Bubble.

Algorithm Guarantee: Every possible cycle in mesh will have at least **one** *Static Bubble*.

STATIC BUBBLE : *KEY IDEA*



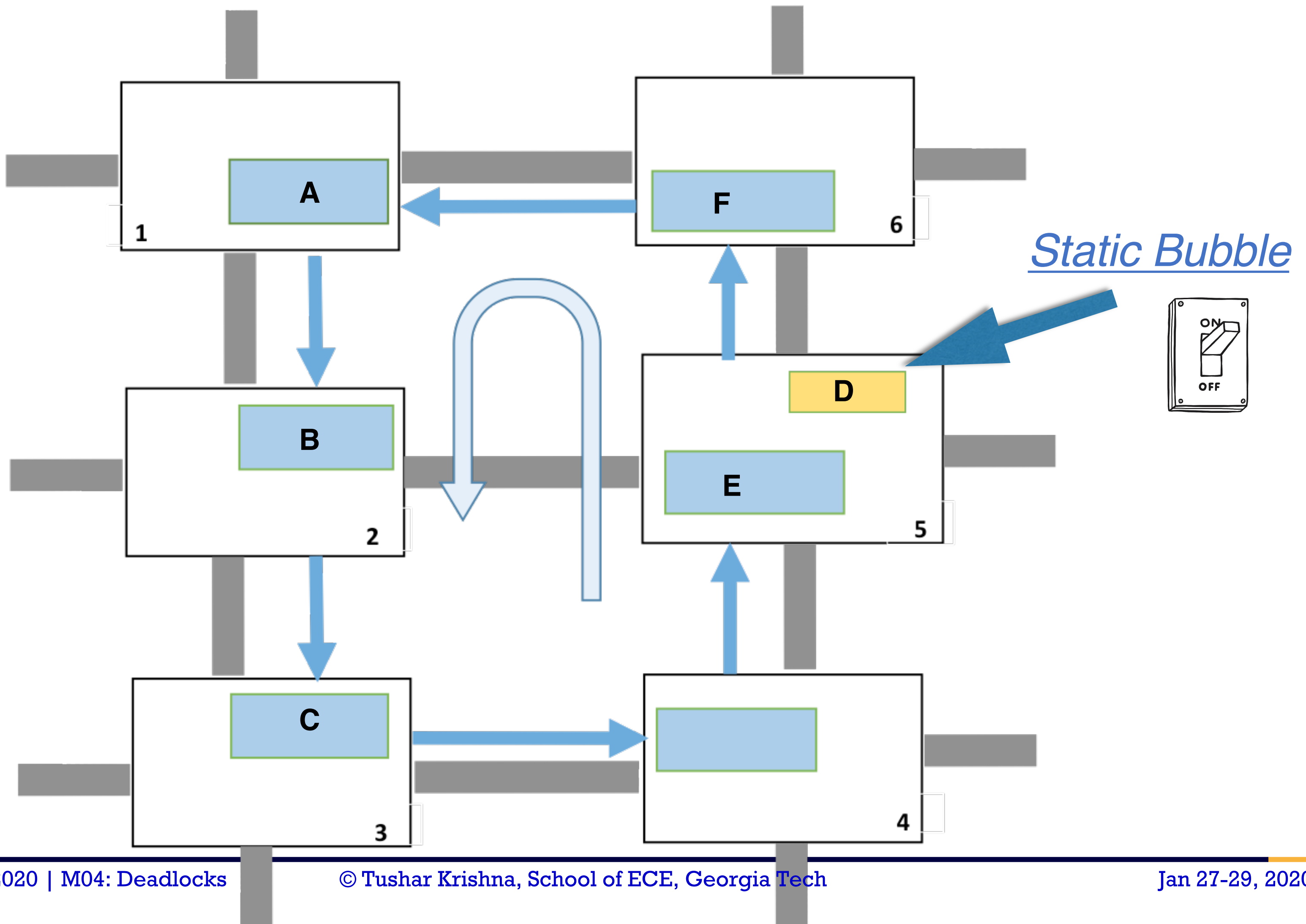
STATIC BUBBLE : *KEY IDEA*



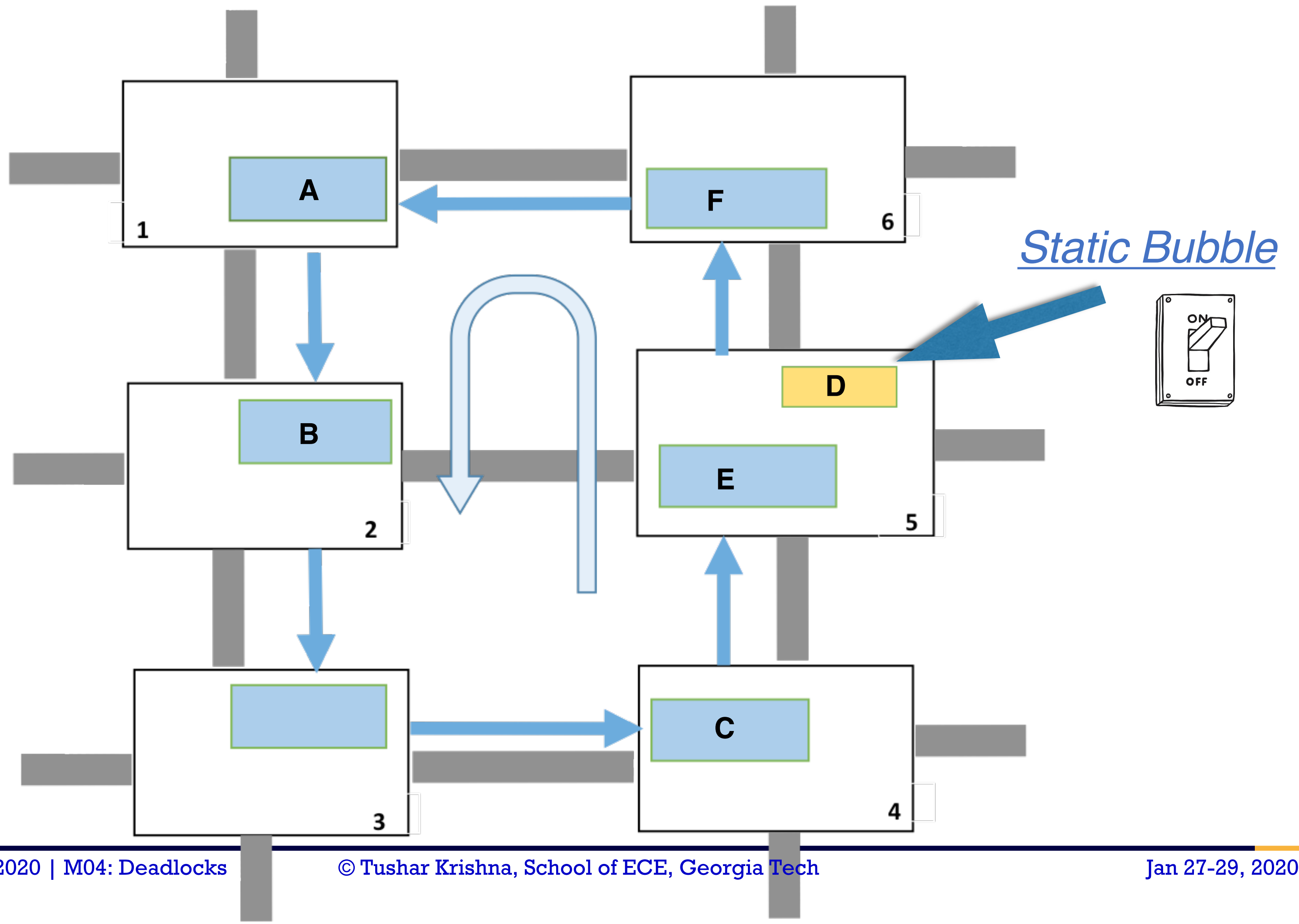
Static Bubble

Static Bubble Turned ON

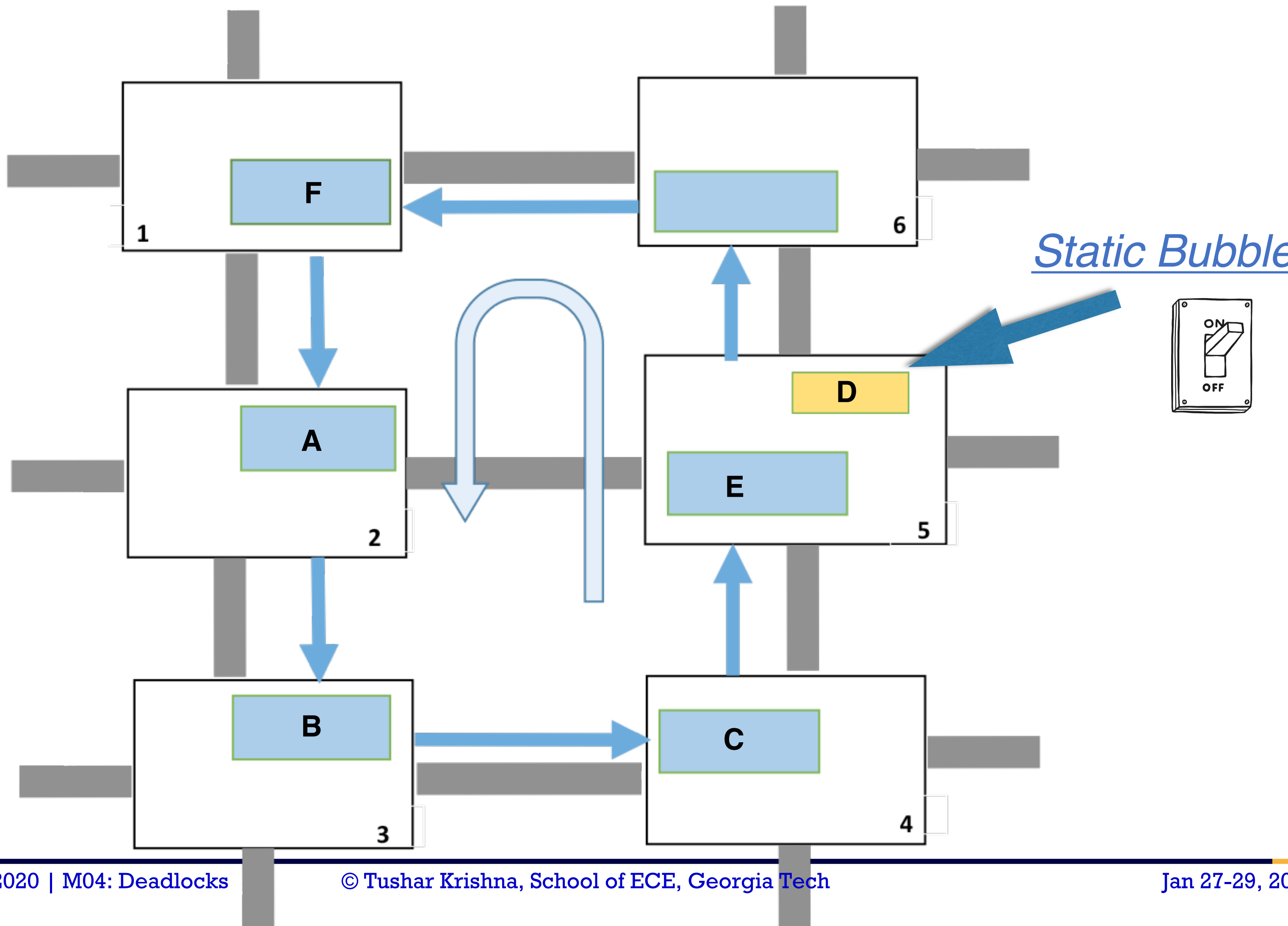
STATIC BUBBLE : *KEY IDEA*



STATIC BUBBLE : *KEY IDEA*

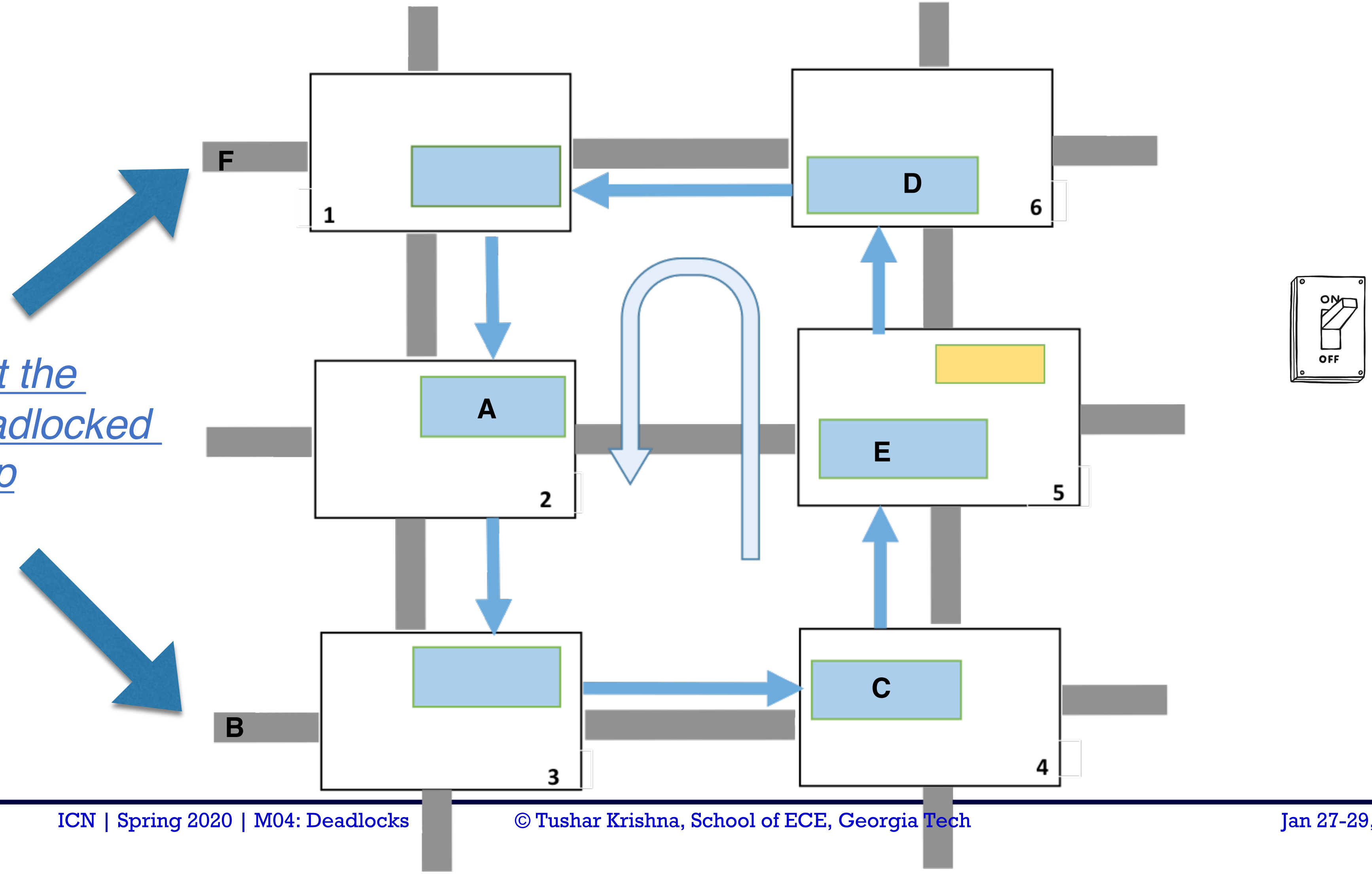


STATIC BUBBLE : *KEY IDEA*

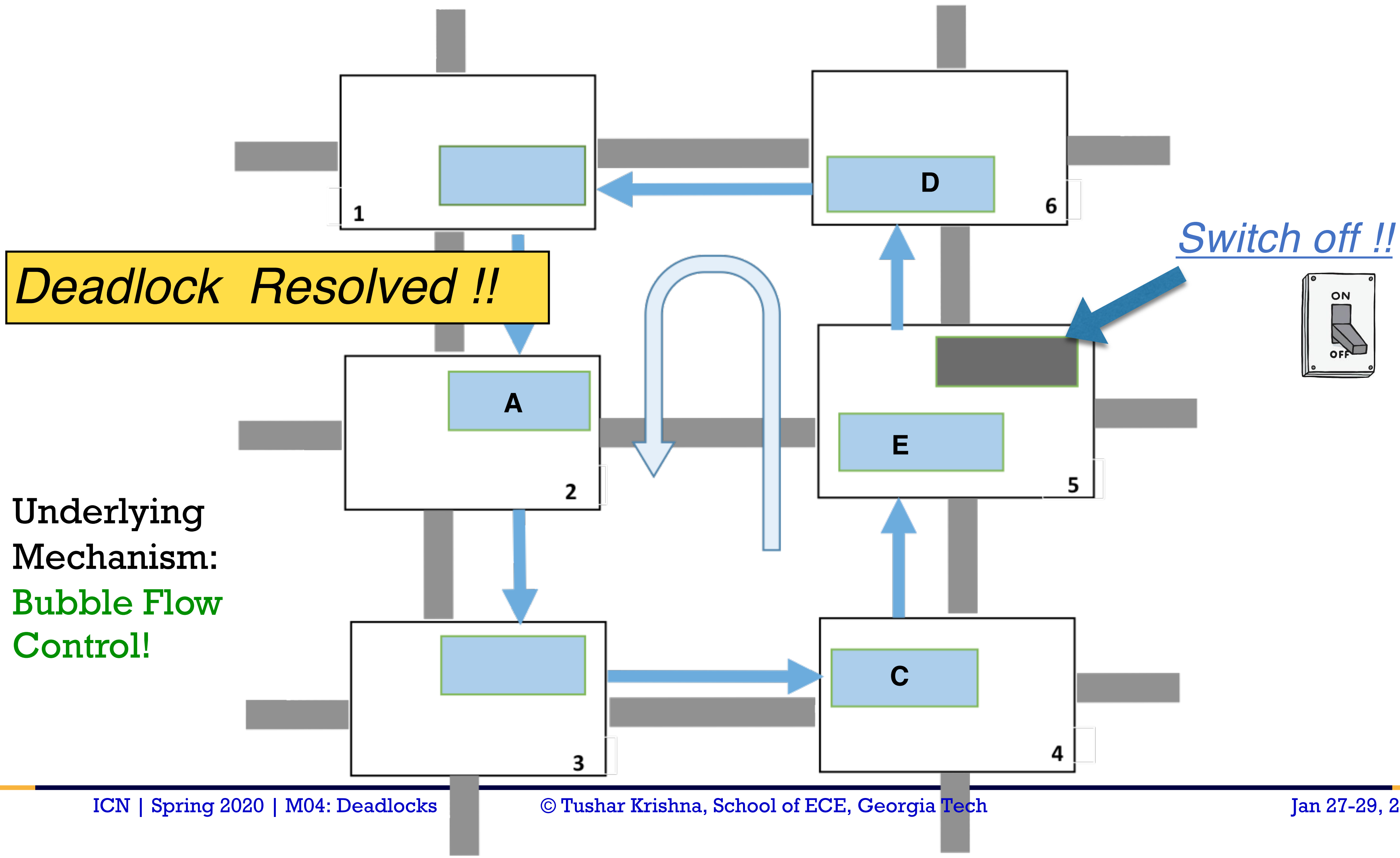


STATIC BUBBLE : *KEY IDEA*

Exit the deadlocked loop



STATIC BUBBLE : *KEY IDEA*



Underlying Mechanism:
Bubble Flow Control!

SPIN

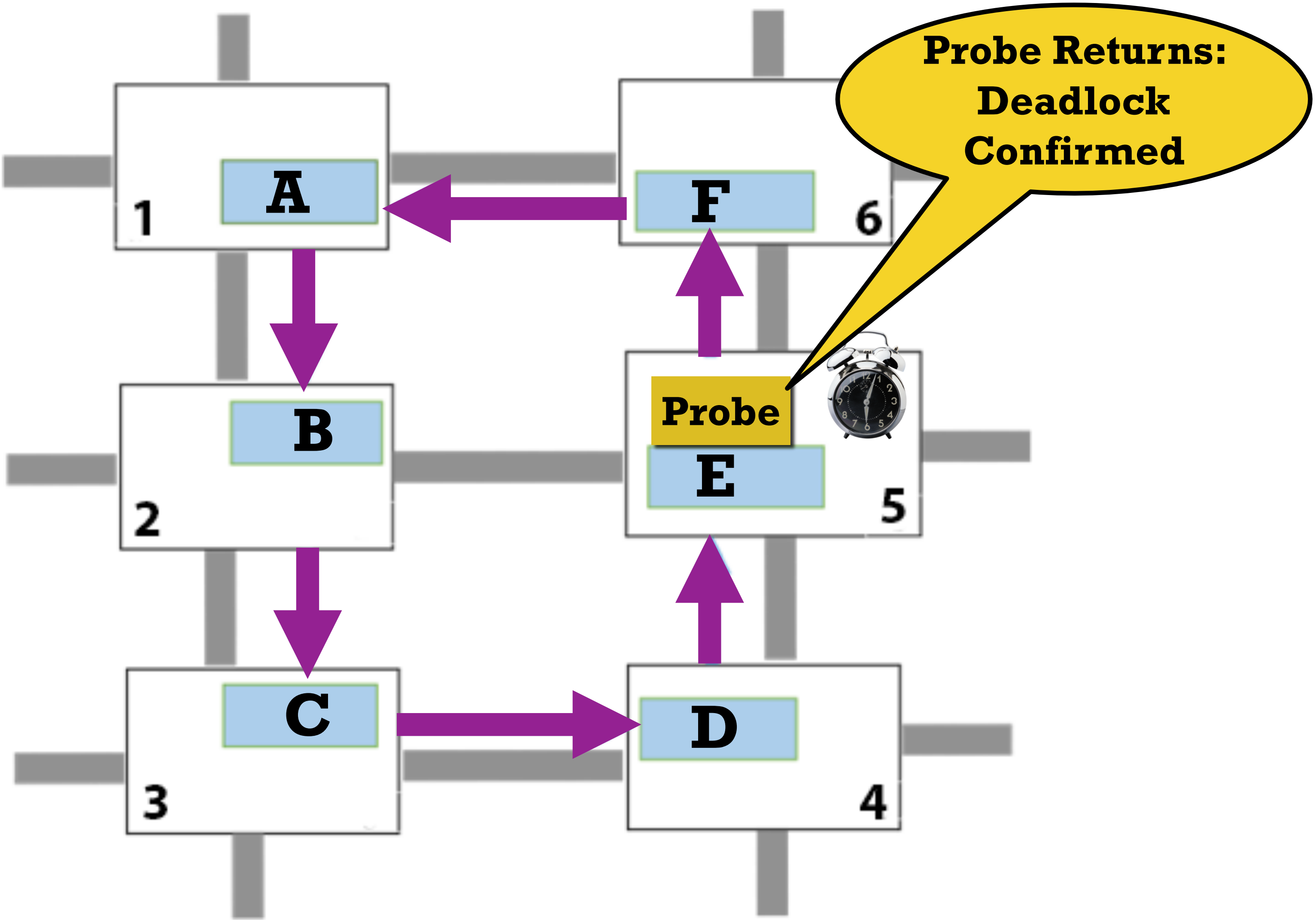
Synchronized Progress in Interconnection Networks (SPIN) : A New Theory for Deadlock Freedom

Aniruddh Ramrakhyani, Paul Gratz, and Tushar Krishna

In Proc of 45th International Symposium on Computer Architecture ***(ISCA)****, Jun 2018*

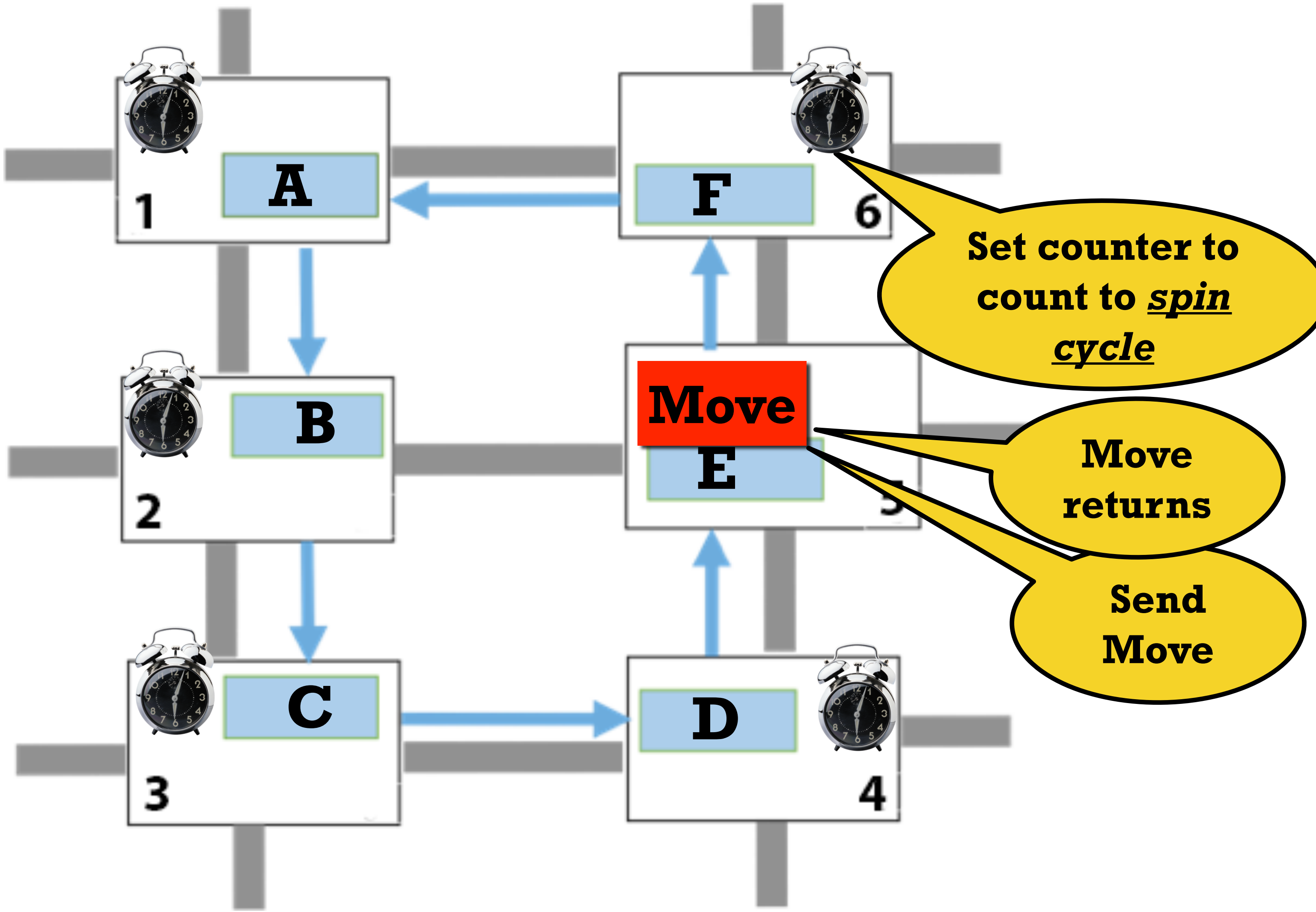
IMPLEMENTATION EXAMPLE : *PROBE MSG*

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



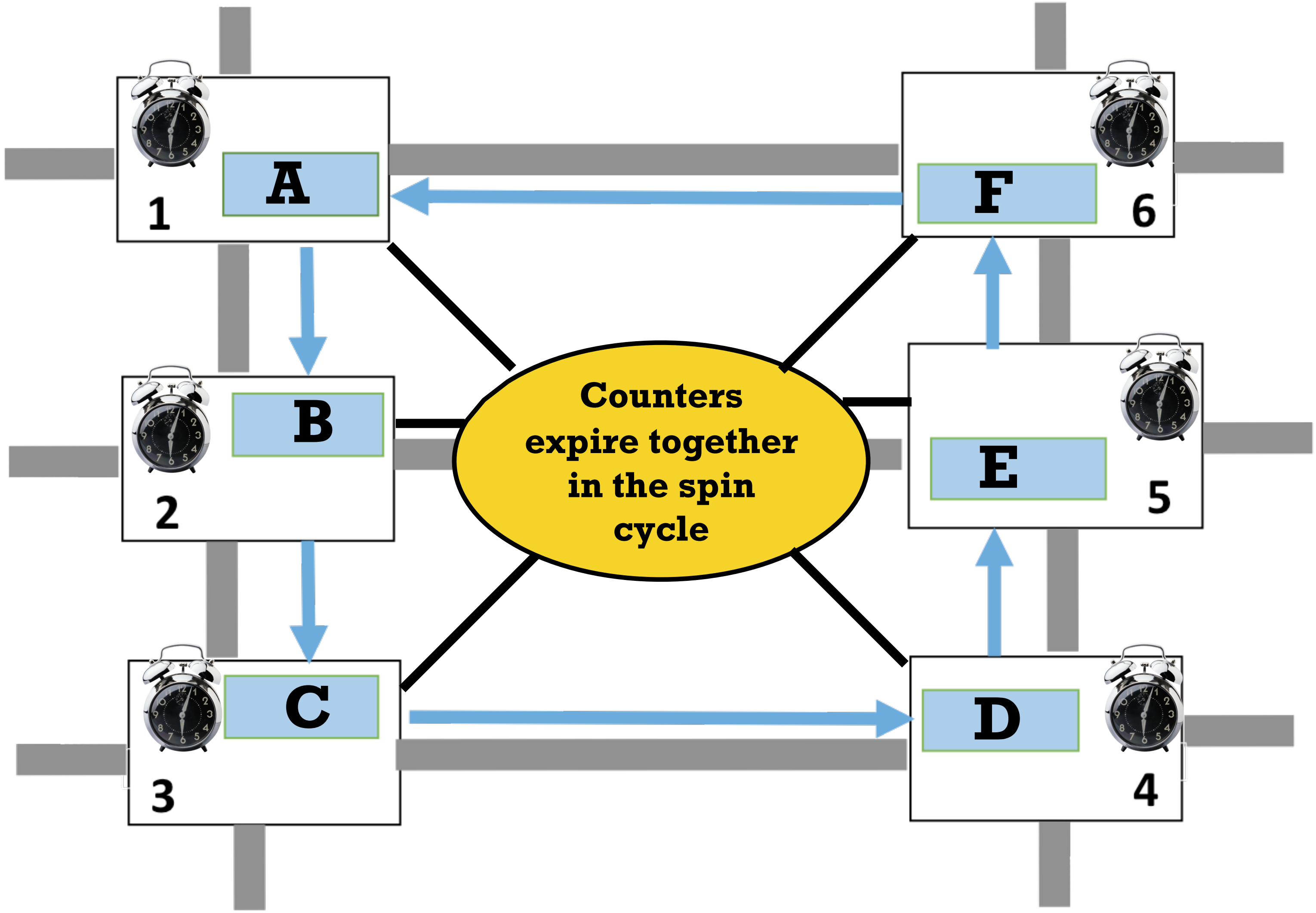
IMPLEMENTATION EXAMPLE : *MOVE MSG.*

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



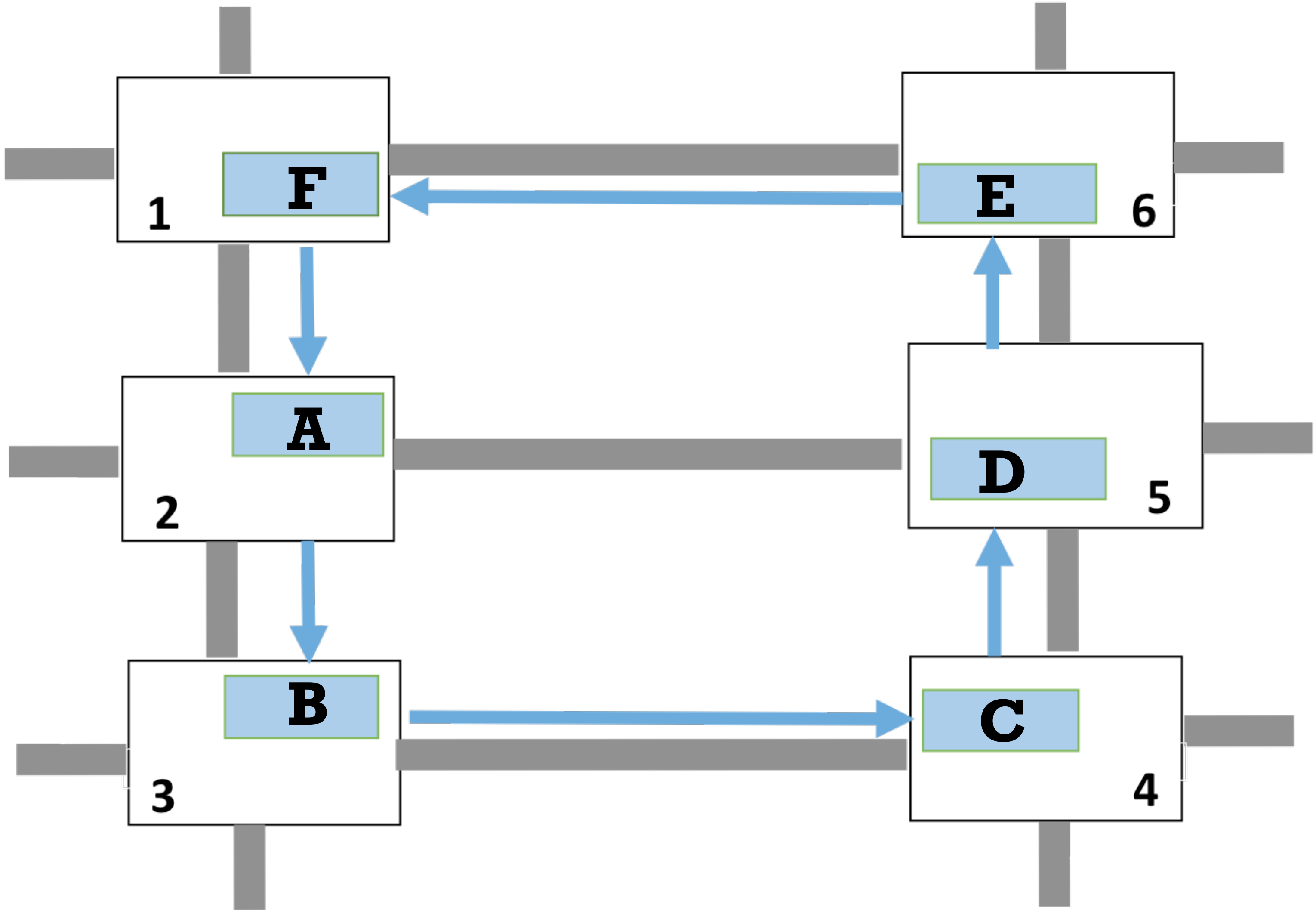
IMPLEMENTATION EXAMPLE : *SPIN*

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.**



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).

DEALING WITH DEADLOCKS

- **Proactive / Avoidance**
  - Guarantee that the network will never deadlock
  - Almost all modern networks use deadlock avoidance

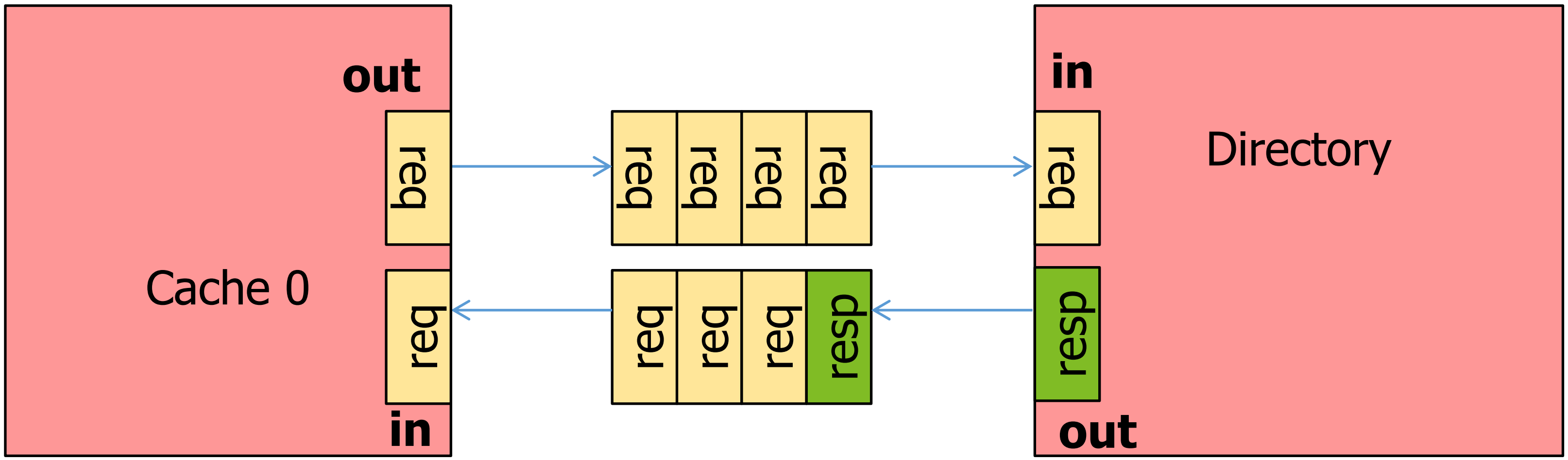
- **Reactive / Recovery**
  - Detect deadlock and correct

- **Subactive**
  - Introduce periodic forced movement among packets

Brownian Bubble Router (NOCS 2018), BINDU (NOCS 2019), SWAP (MICRO 2019), DRAIN (HPCA 2020) → Next Lecture

ANOTHER KIND OF DEADLOCK: PROTOCOL DEADLOCK

Cache / Directory can process a request only if there is space in its output queue to send a response



Deadlock, even though network is deadlock-free

Need separate Virtual Channels* for requests and responses (called Virtual Networks)

Responses should always be drained ("consumption assumption")