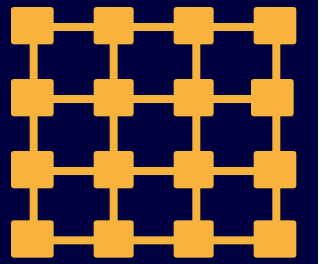
ECE 8823 A / CS 8803 - ICN
Interconnection Networks
Spring 2017



http://tusharkrishna.ece.gatech.edu/teaching/icn_s17/

Lecture 10: Router Microarchitecture

Tushar Krishna

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

tushar@ece.gatech.edu

Acknowledgment: Arbiter slides adapted from Univ of Toronto ECE 1749 H (N Jerger)

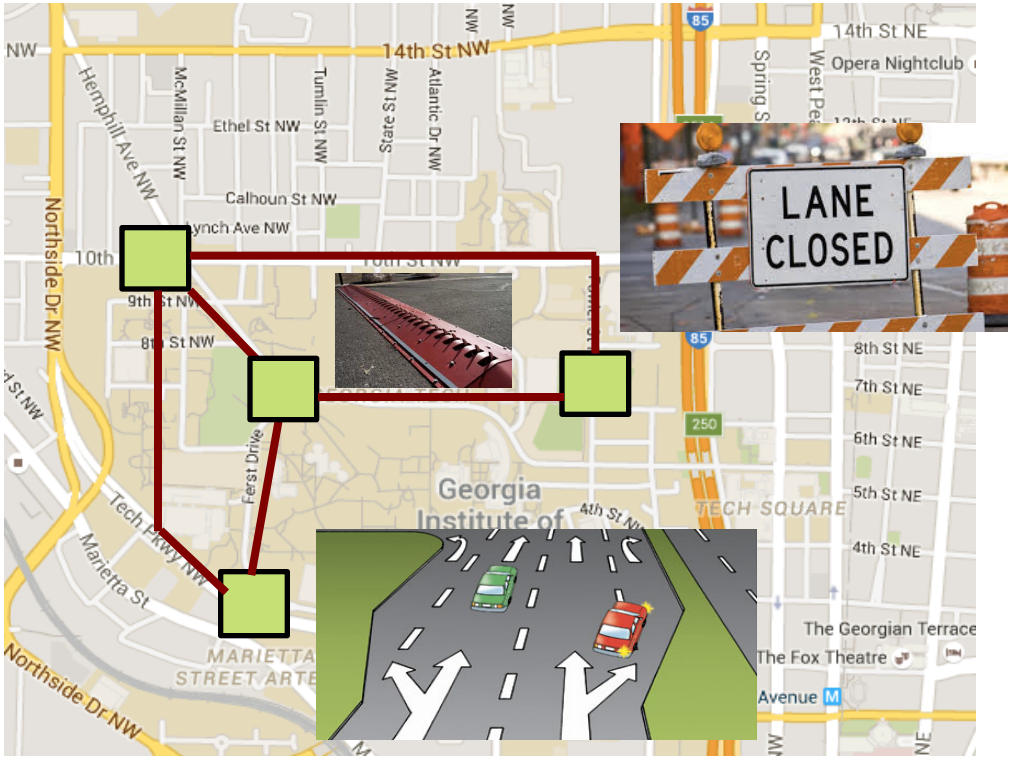
Network Architecture

- **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)

© Tushar Krishna, School of ECE, Georgia Tech

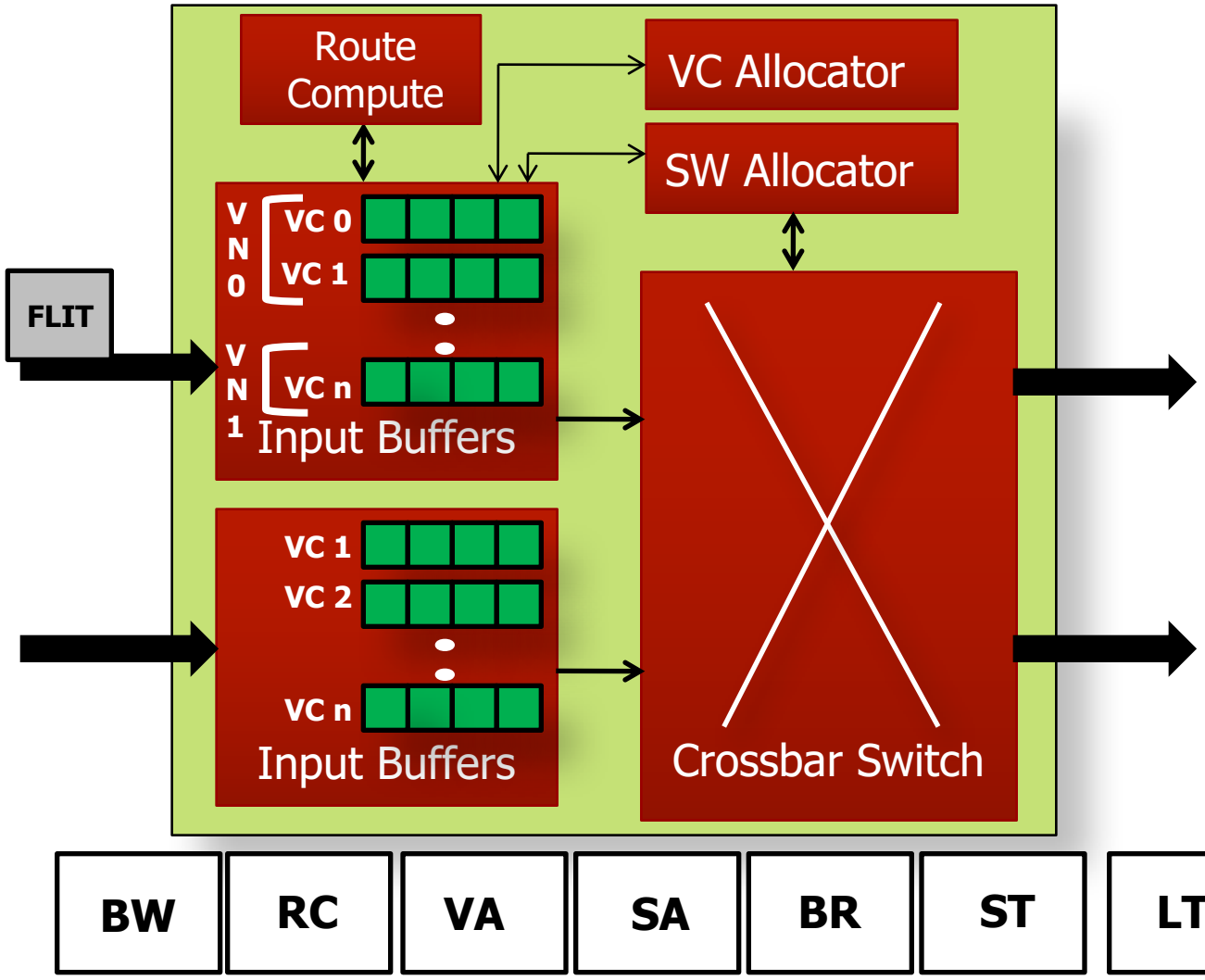
Router Microarchitecture

- Implementation of routing, flow control, and switching
  - Impacts per-hop delay and energy



© Tushar Krishna, School of ECE, Georgia Tech

Virtual Channel Router



BW: Buffer Write

RC: Route Compute

VA: VC Allocation
Input VCs arbitrate for "output" VCs (Input VCs at next router)

SA: Switch Allocation
Input ports arbitrate for output ports

BR: Buffer Read

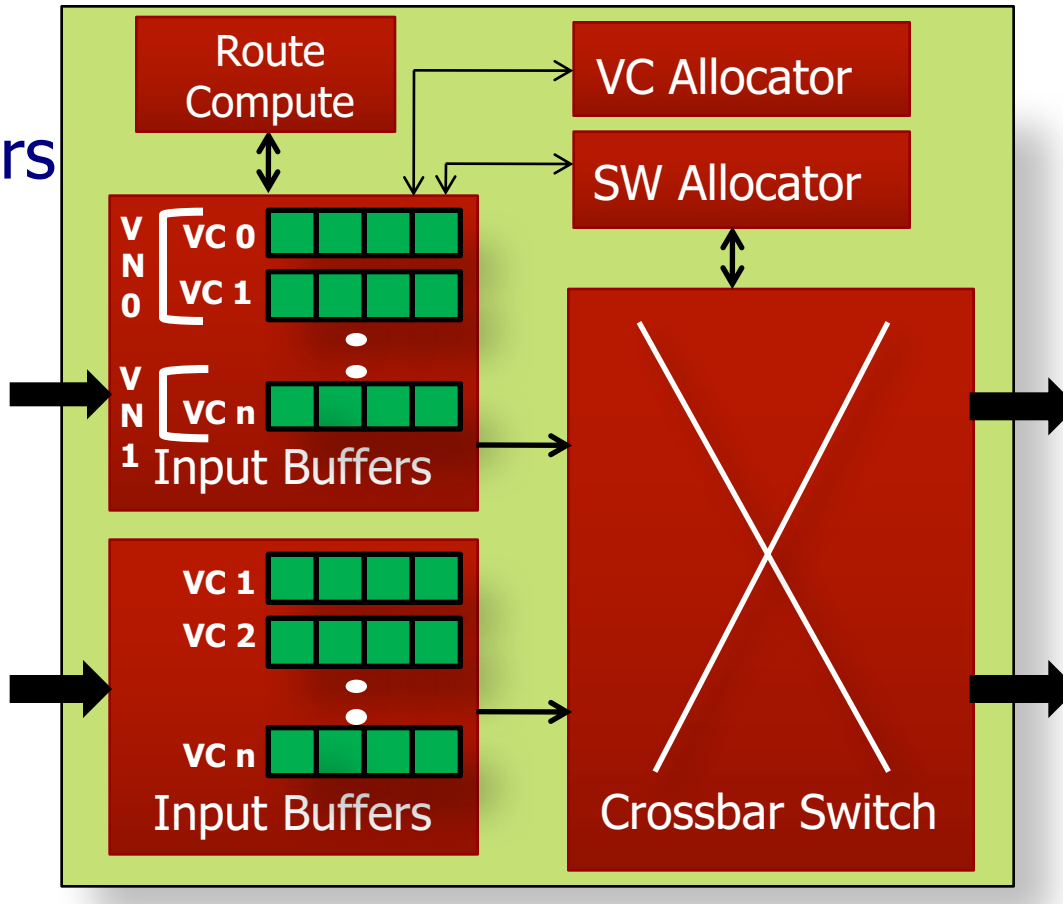
ST: Switch Traversal

LT: Link Traversal

© Tushar Krishna, School of ECE, Georgia Tech

Router Microarchitecture

- Components
  - Virtual Channel Buffers
  - Routing Logic
  - Allocation
    - Switch Allocation
    - VC Allocation
  - Crossbar Switch
  - Link
- Pipeline
  - 5-cycle router → 1-cycle router



© Tushar Krishna, School of ECE, Georgia Tech

Router Microarchitecture

- Components
  - Virtual Channel Buffers
  - Routing Logic
  - Allocation
    - Switch Allocation
    - VC Allocation
  - Crossbar Switch
  - Link
- Pipeline
  - 5-cycle router → 1-cycle router

© Tushar Krishna, School of ECE, Georgia Tech

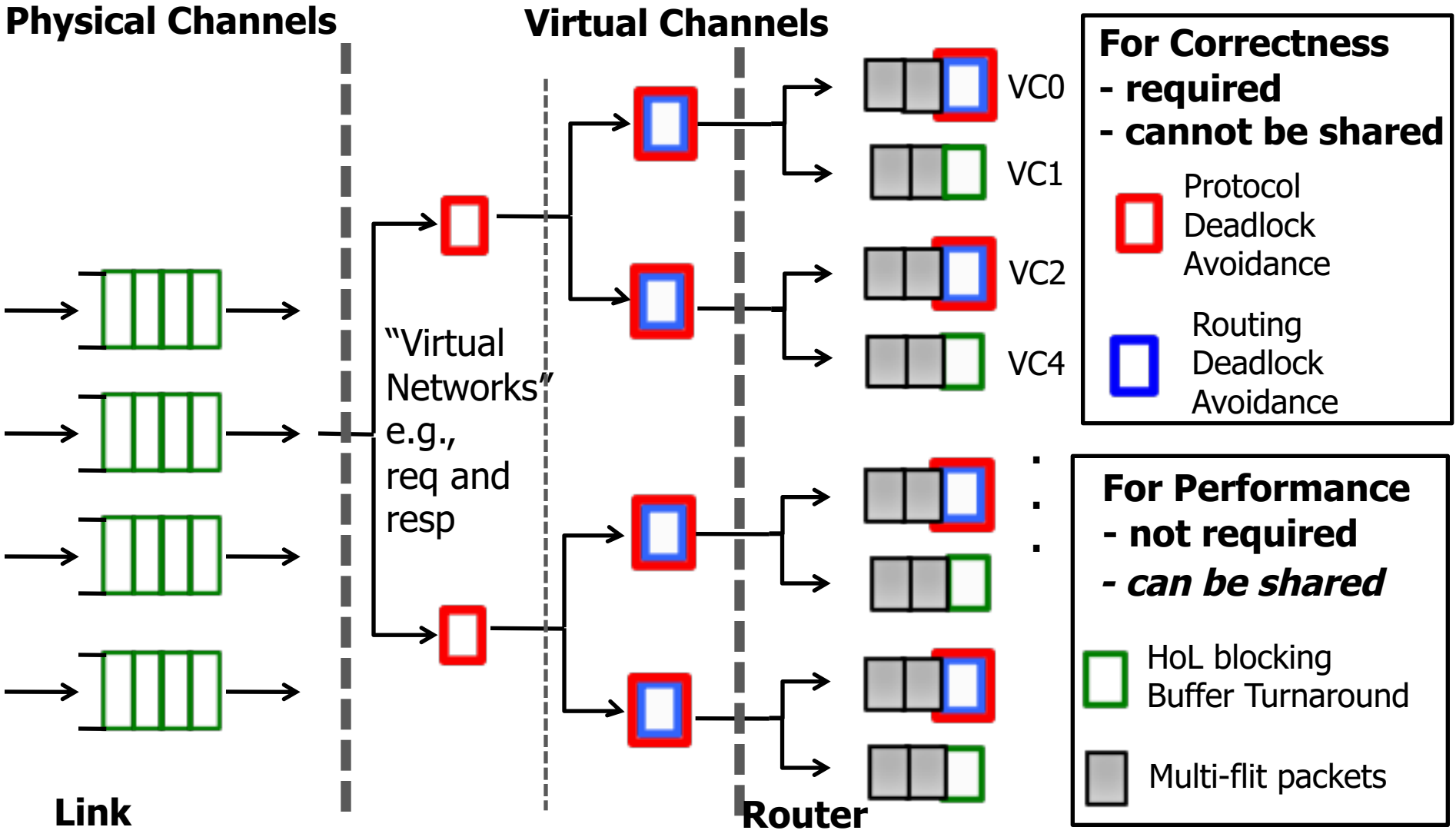
Buffer Organization

- Why does the router have buffers?
  - To manage contention between shared links

- Minimum number of buffers needed?
  - Functionality/Correctness
    - One per Virtual Channel to avoid deadlocks
      - Messages in two different VCs will never indefinitely block one another
        - If one of the VCs is blocked, the second one can go ahead
    - How many VCs required to avoid deadlocks?
      - Two kinds of deadlocks: Protocol and Routing (next slide)
  - Performance (Flow Control)
    - Message going out of congested output port should not block a message behind it going out of different output port
      - i.e., avoid "Head-of-Line Blocking"
    - Cover buffer turnaround time to sustain full throughput

© Tushar Krishna, School of ECE, Georgia Tech

Minimum number of buffers



© Tushar Krishna, School of ECE, Georgia Tech

Virtual Channel Implementation

- **State Information**
  - G (Global): Idle, Routing, waiting for output VC, waiting for credits in output VC, active
  - R (Route): output port for the packet
  - O (Output VC): output VC (VC at next router) for this packet
  - C (Credit Count): number of credits (i.e., downstream flit buffers) in output VC O at output port R
  - P (pointers): pointers to head and tail flits in buffer pool VCs implemented as shared pool (next slide)

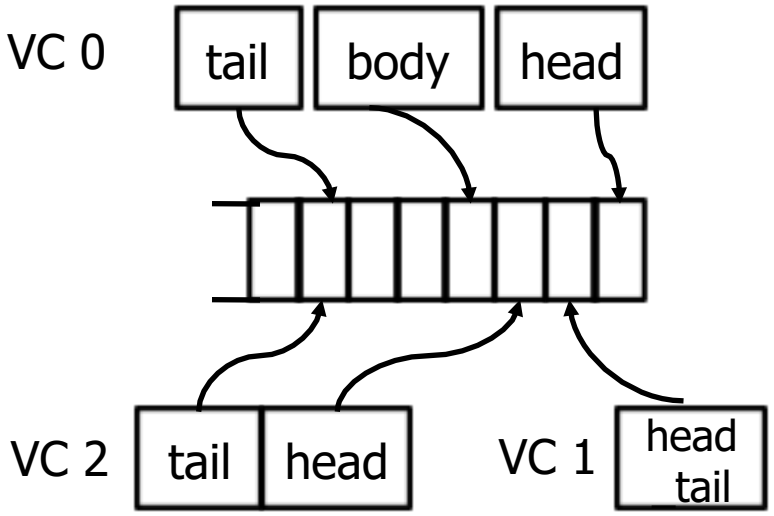
© Tushar Krishna, School of ECE, Georgia Tech

Virtual Channel Implementation

- **Storage**
  - Private Buffers Per VC
    - n-flit deep FIFO per VC
      - n >= 1, but can be smaller than the size of the packet

  Or

  - Shared Buffers
    - All VCs share a pool of buffers
      - One reserved buffer per VC
    - \+ Allows variable sized VCs
    - \- More complex circuitry
      - Pointers for every flit
      - Linked list of free buffer slots



© Tushar Krishna, School of ECE, Georgia Tech

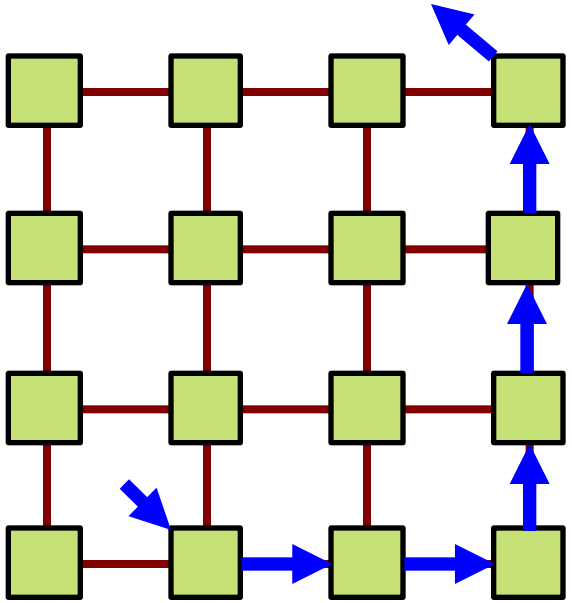
Router Microarchitecture

- Components
  - Virtual Channel Buffers
  - Routing Logic
  - Allocation
    - Switch Allocation
    - VC Allocation
  - Crossbar Switch
  - Link
- Pipeline
  - 5-cycle router → 1-cycle router

© Tushar Krishna, School of ECE, Georgia Tech

Routing Logic

- **Source Routing** – each packet comes with a fixed output port
  - Example: (E, E, N, N, N, N, Eject)
  - Each router reads left most entry, and then strips it away for next hop
  - Pros
    - + Save latency at each hop
    - + Save routing-hardware at each hop
    - + Can reconfigure routes based on faults
    - + Supports irregular topologies
  - Cons
    - Overhead to store **all routes** at NIC
    - Overhead to carry routing bits in every packet (3-bits port x max hops)
    - Cannot adapt based on congestion



© Tushar Krishna, School of ECE, Georgia Tech

Routing Logic

- **Routing Table** at every router
  - Packet can index into it via destination bits, or some static VCid
  - Pros

    + Any routing algorithm can be implemented by reconfiguring the tables
  - Cons

    - Latency, Energy, and Area Overhead – not recommended on-chip

Routing Table for West-first routing in a 3x3 Mesh

	To								
From	00	01	02	10	11	12	20	21	22
00	X -	N -	N -	E -	E N	E N	E -	N E	N E
01	S -	X -	N -	E S	E -	E N	E S	E -	E N
02	S -	S -	X -	E S	E S	E -	E S	E S	E -
10	W -	W -	W -	X -	N -	N -	E -	E N	E N
11	W -	W -	W -	S -	X -	N -	E S	E -	N E
12	W -	W -	W -	S -	S -	X -	E S	E S	E -
20	W -	W -	W -	W -	W -	W -	X -	N -	N -
21	W -	W -	W -	W -	W -	W -	S -	X -	N -
22	W -	W -	W -	W -	W -	W -	S -	S -	X -

Routing Logic

- **Combinational Logic** - Compute output port at each router
  - packet carries only destination coordinates, and each router computes output port based on packet state and router state
    - e.g., **deterministic:** use remaining hops and direction
    - e.g., **oblivious:** use remaining hops and direction and some randomness factor
    - e.g., **adaptive:** use congestion metrics (such as buffer occupancy), history, etc.
  - Pros

    \+ Simple to implement – **most common approach in NoCs**
  - Cons

    - Routing delay is in critical path
    - Routing algorithm has to be fixed at design time

© Tushar Krishna, School of ECE, Georgia Tech

Router Microarchitecture

- Components
  - Virtual Channel Buffers
  - Routing Logic
  - Allocation
    - Switch Allocation
    - VC Allocation
  - Crossbar Switch
  - Link
- Pipeline
  - 5-cycle router → 1-cycle router

© Tushar Krishna, School of ECE, Georgia Tech

Switch and VC Allocation

- "**Allocator**" matches N requests to M resources
- "**Arbiter**" matches N requests to 1 resource
- VC Allocation
  - Requests: Input VCs
  - Resources: Output VCs (i.e., VCs at next router)
- Switch Allocation
  - Requests: Input VCs/Input ports
  - Resources: Output ports of the router
- Allocator that delivers the highest matching translates to higher network throughput
- In most NoCs, the allocation logic determines cycle time
  - Allocators must be fast and/or able to be pipelined

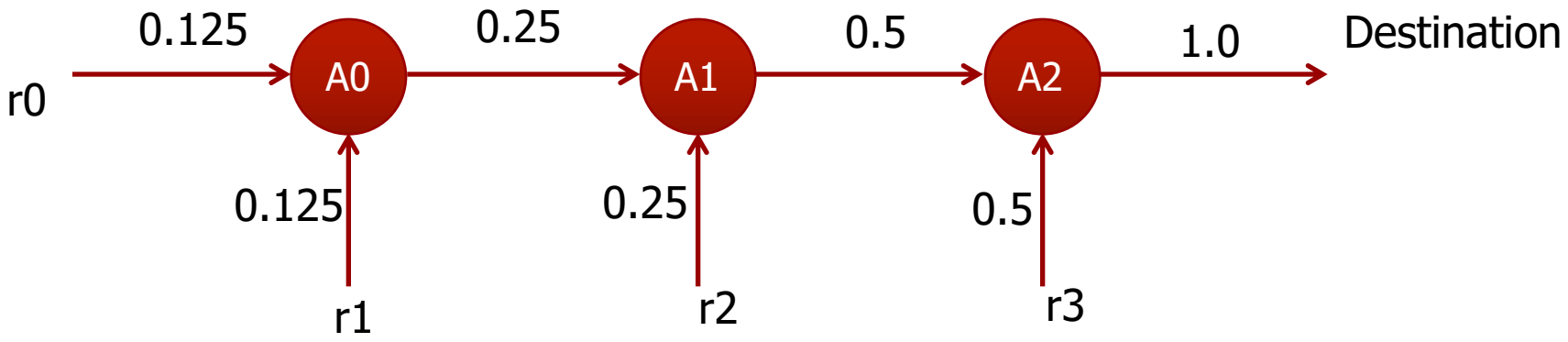
© Tushar Krishna, School of ECE, Georgia Tech

Fairness

- Intuitively, a fair arbiter is one that provides equal service to different requesters
- Weak fairness: Every request is **eventually** served
- Strong fairness: Requesters will be served **equally often**
- FIFO Fairness: Requesters are served **in the order** they make their requests

© Tushar Krishna, School of ECE, Georgia Tech

Locally Fair Example



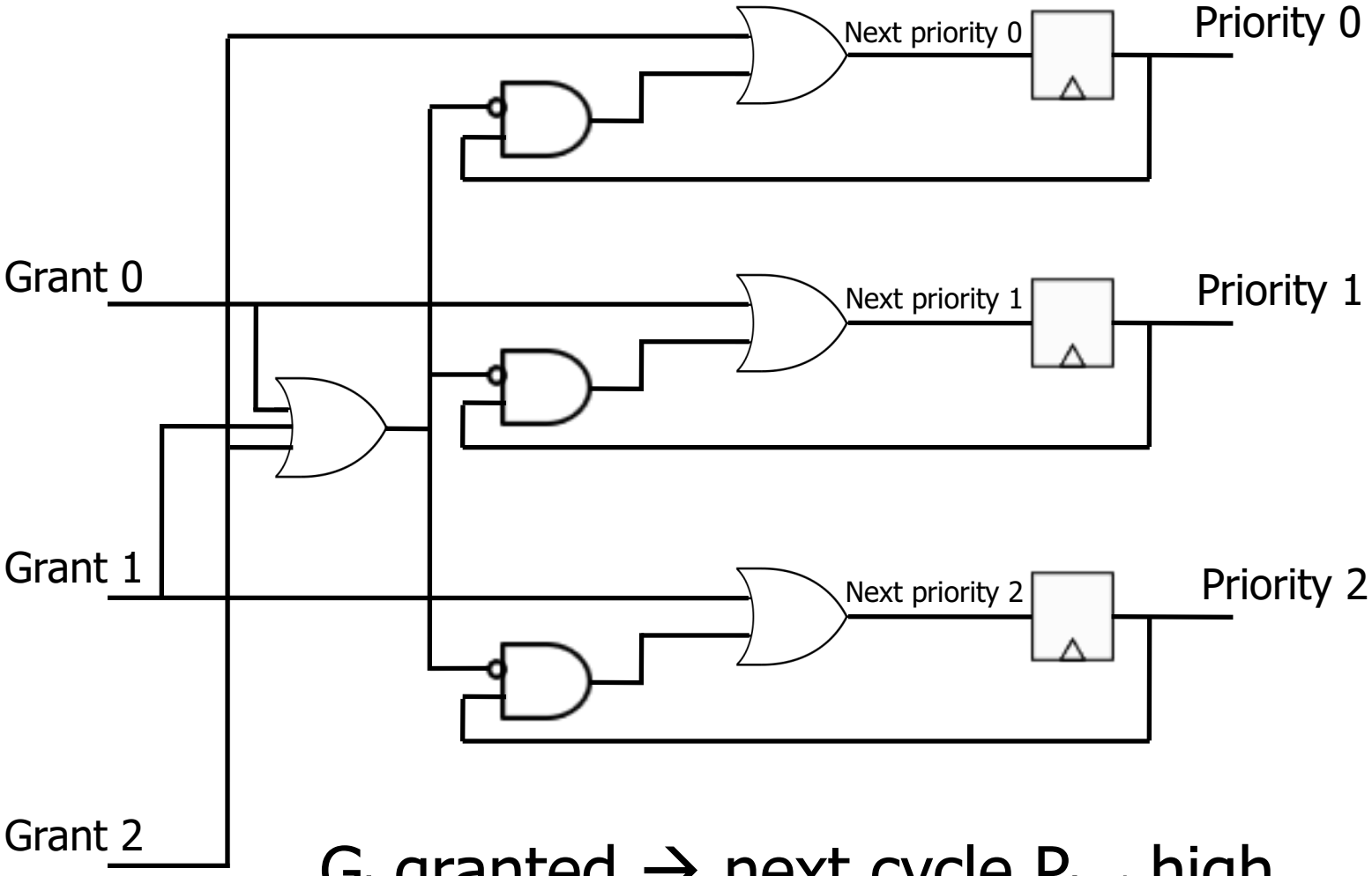
R3 receives 4 times the bandwidth as r0, even though individual arbiters provide strong fairness

Round Robin Arbiter

- Last request serviced given lowest priority
- Generate next priority vector from current grant vector
- If no requests, priority is unchanged
- Exhibits fairness

© Tushar Krishna, School of ECE, Georgia Tech

Round Robin Arbiter Implementation



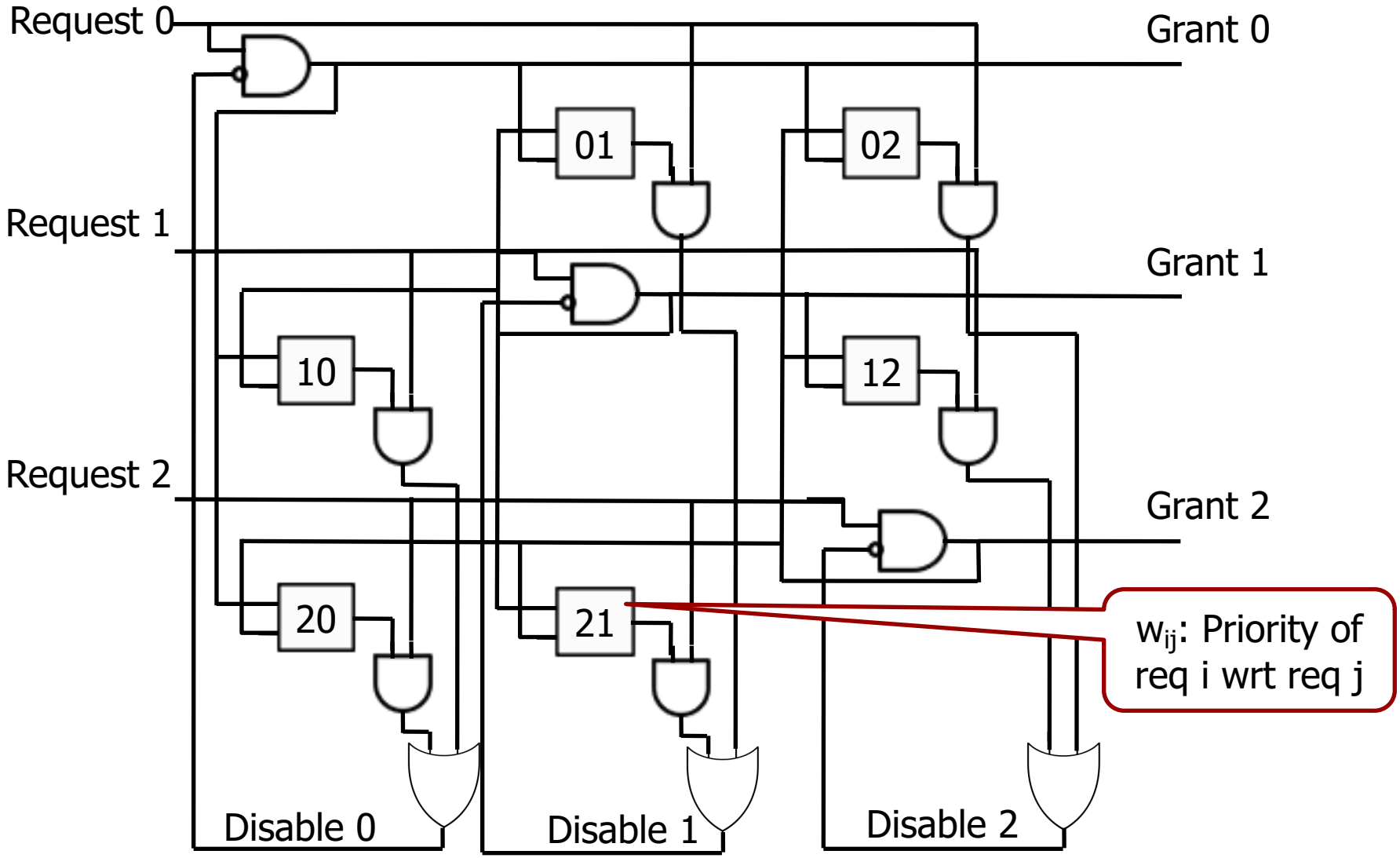
G_i granted → next cycle P_{i+1} high

Matrix Arbiter

- Least recently served priority scheme
- Triangular array of state bits w_{ij} for j < i
  - **Bit w_{ij} indicates request i takes priority over j**
  - **Each time request k granted, clears all bits in row k and sets all bits in column k**
- Good for small number of inputs
- Fast, inexpensive and provides strong fairness

© Tushar Krishna, School of ECE, Georgia Tech

Matrix Arbiter Implementation



© Tushar Krishna, School of ECE, Georgia Tech

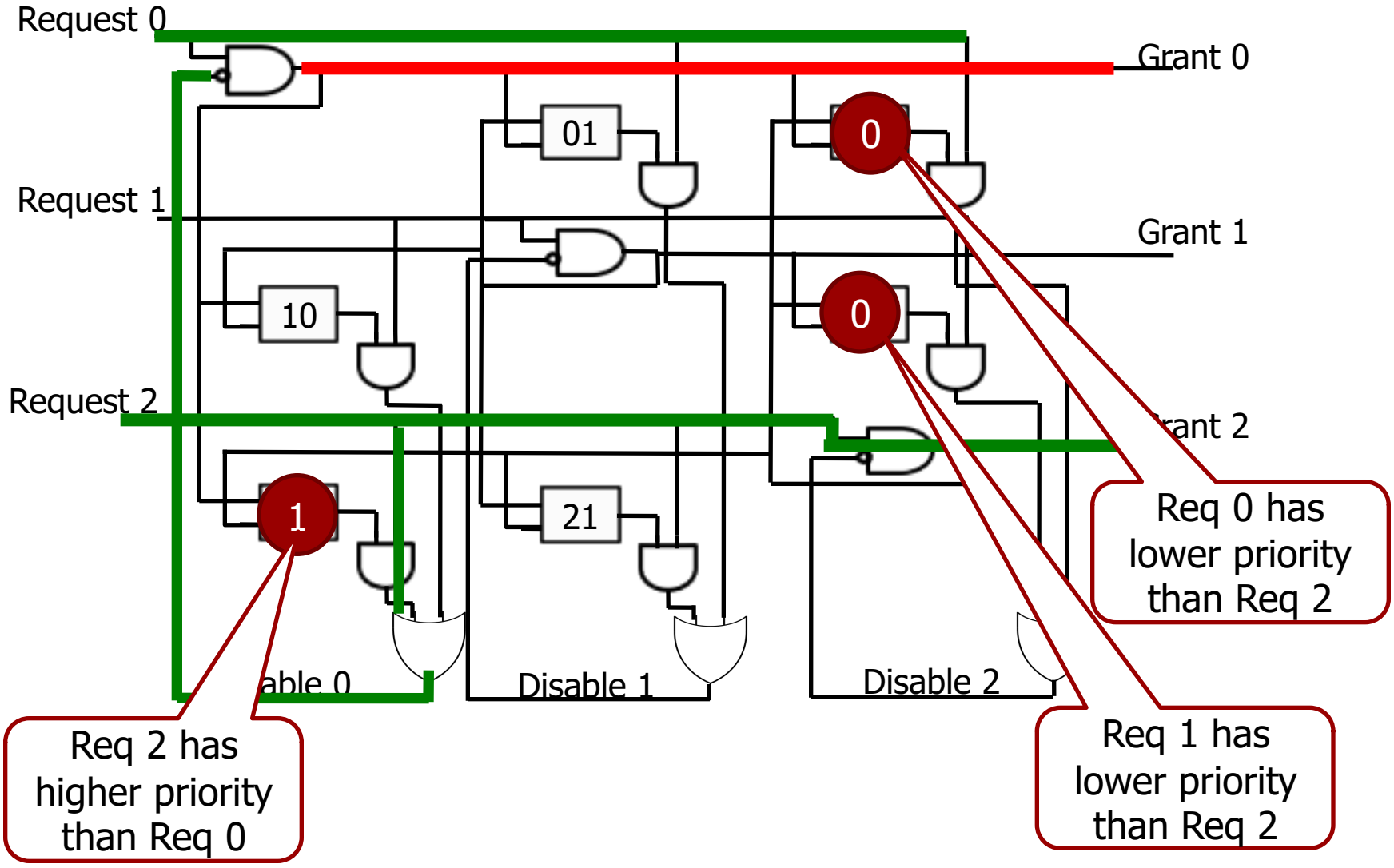
Matrix Arbiter Operation

- Grant Policy
  - When a request is asserted, it is AND-ed with the state bits in its row to disable any lower priority requests
  - Request with highest priority is granted

- Update Policy
  - Each time a request k is granted, the state of the matrix is updated by clearing all bits in row k and setting all bits in column k.

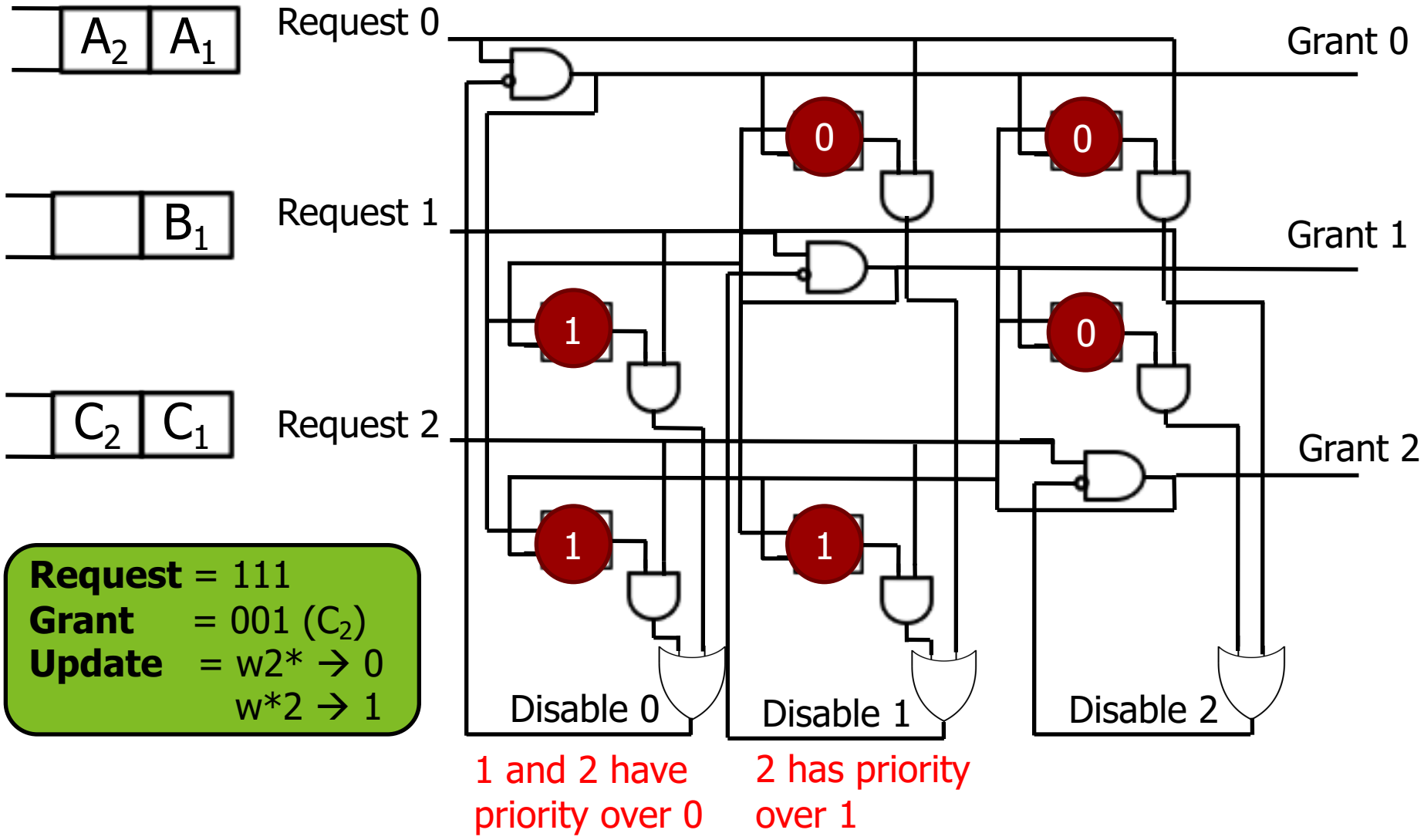
© Tushar Krishna, School of ECE, Georgia Tech

Matrix Arbiter Implementation



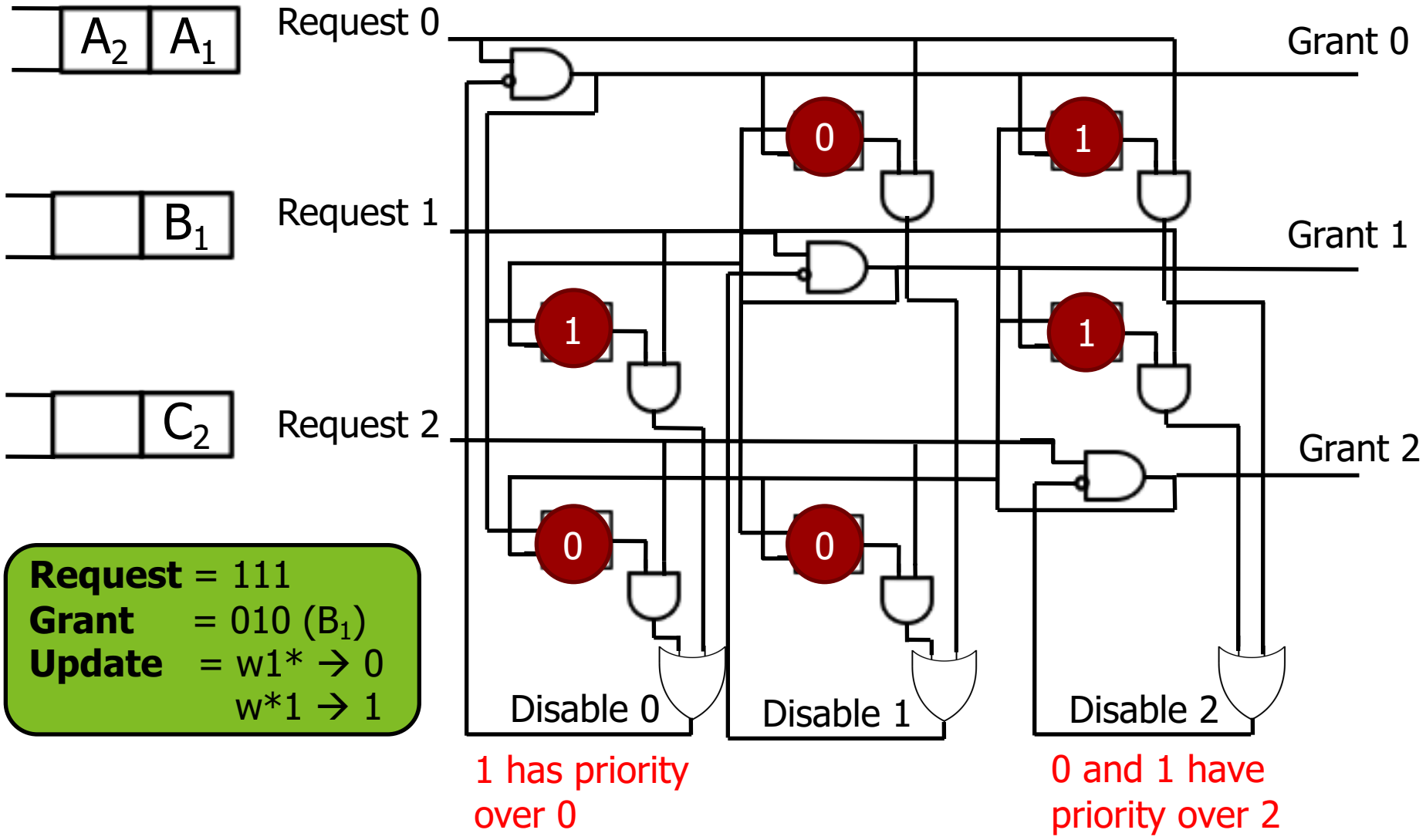
© Tushar Krishna, School of ECE, Georgia Tech

Matrix Arbiter Example



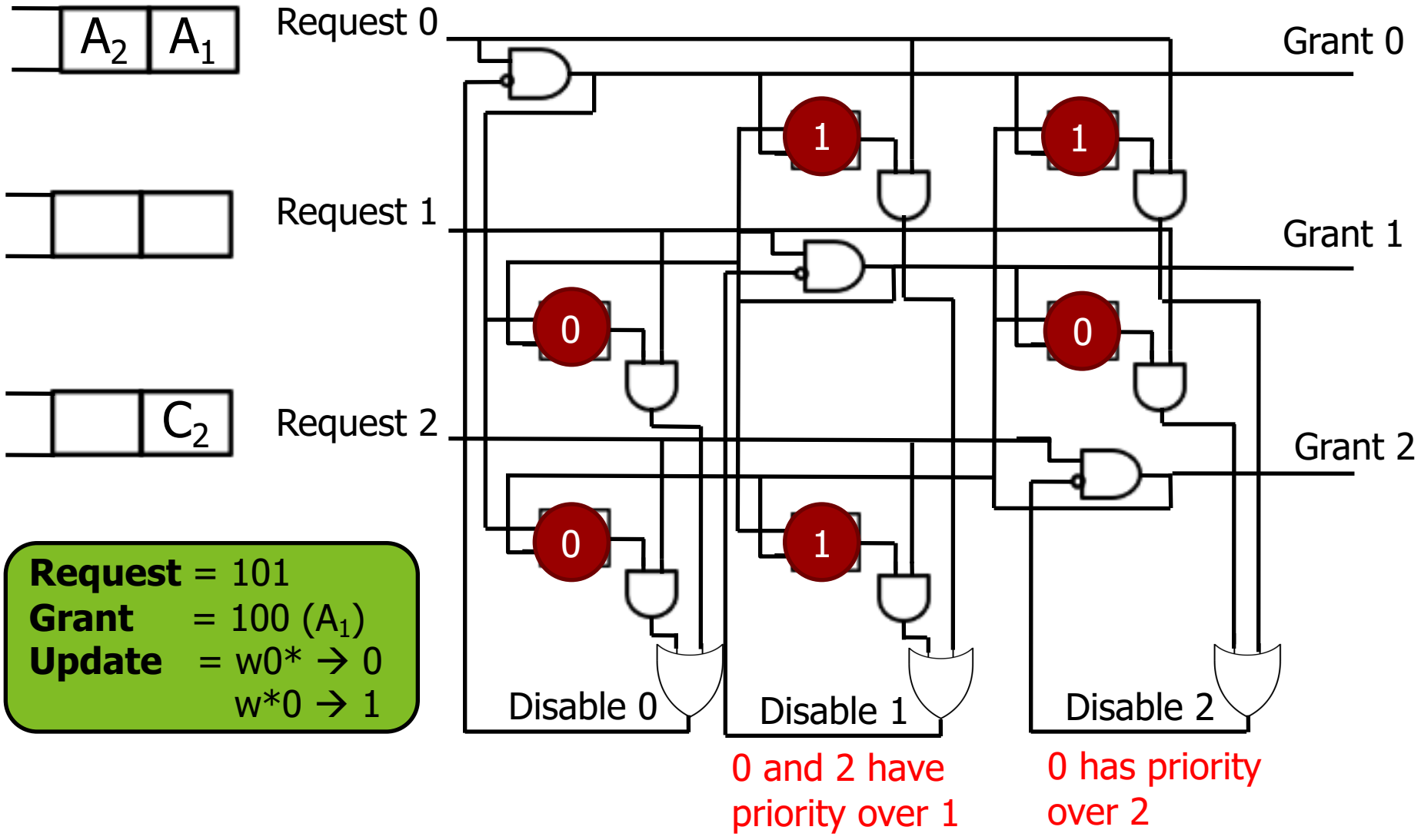
Request = 111
Grant = 001 (C_2)
Update = w2* → 0
w*2 → 1

Matrix Arbiter Example



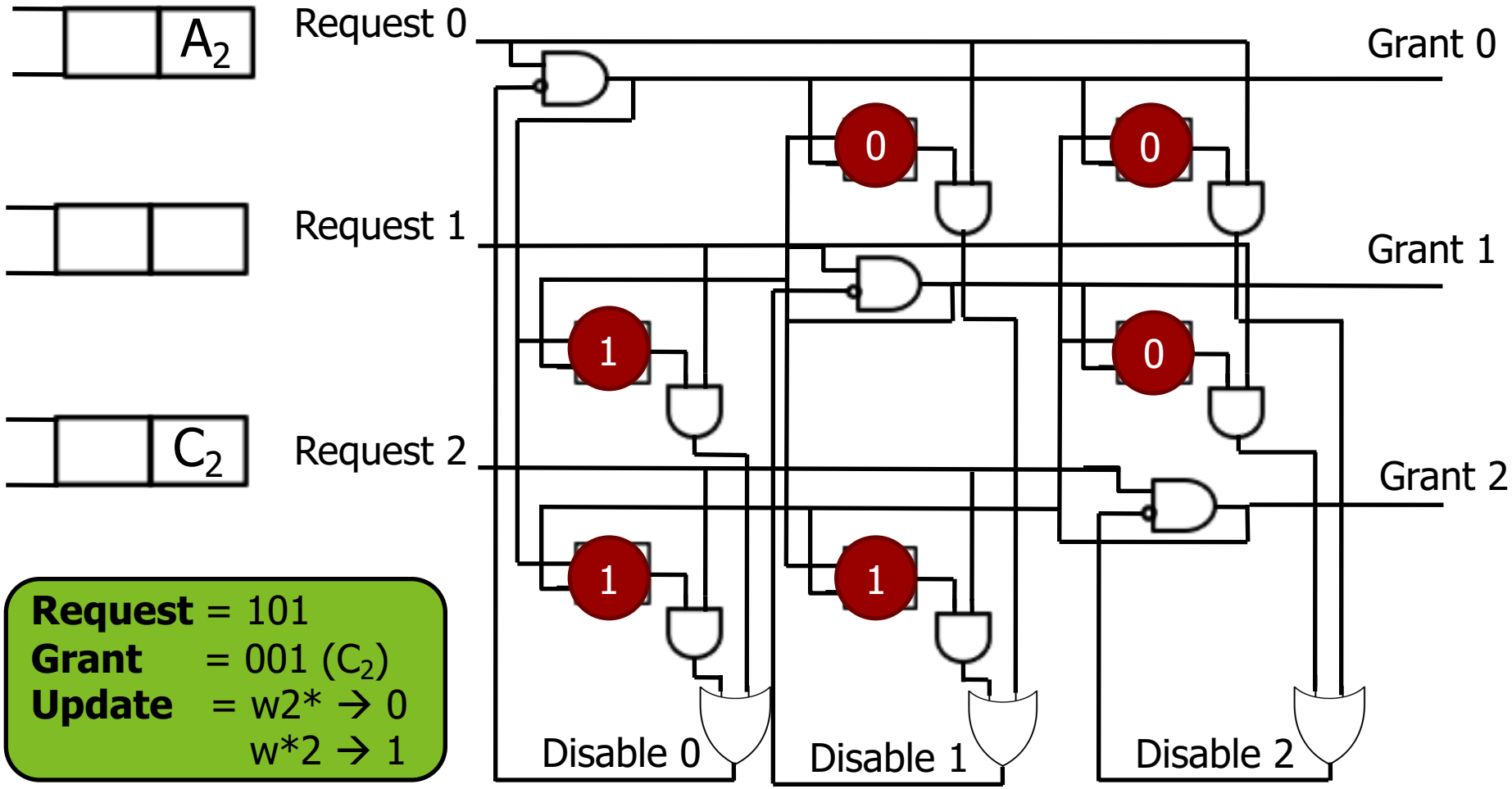
Request = 111
Grant = 010 (B_1)
Update = w1* → 0
w*1 → 1

Matrix Arbiter Example



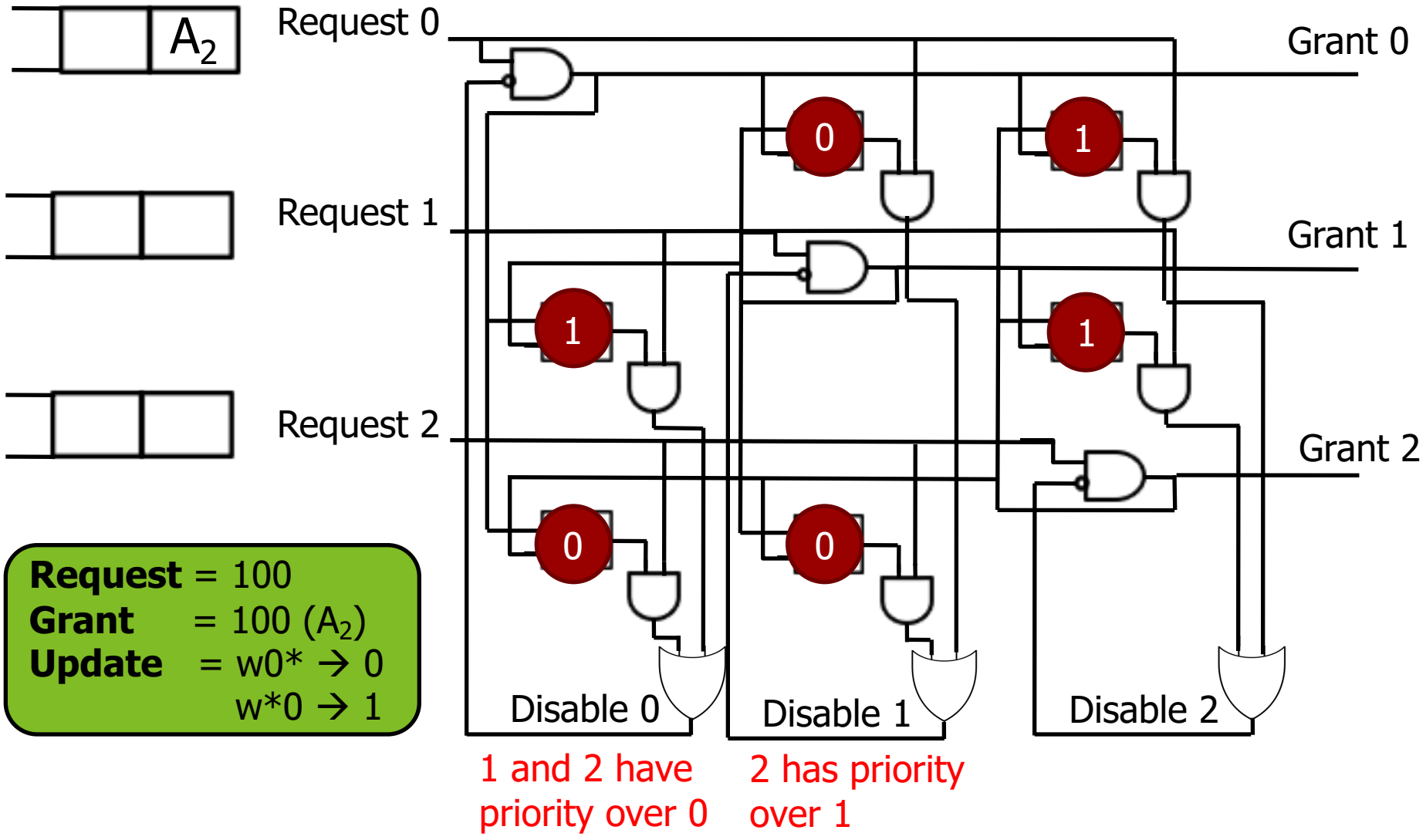
Request = 101
Grant = 100 (A_1)
Update = w0* → 0
w*0 → 1

Matrix Arbiter Example



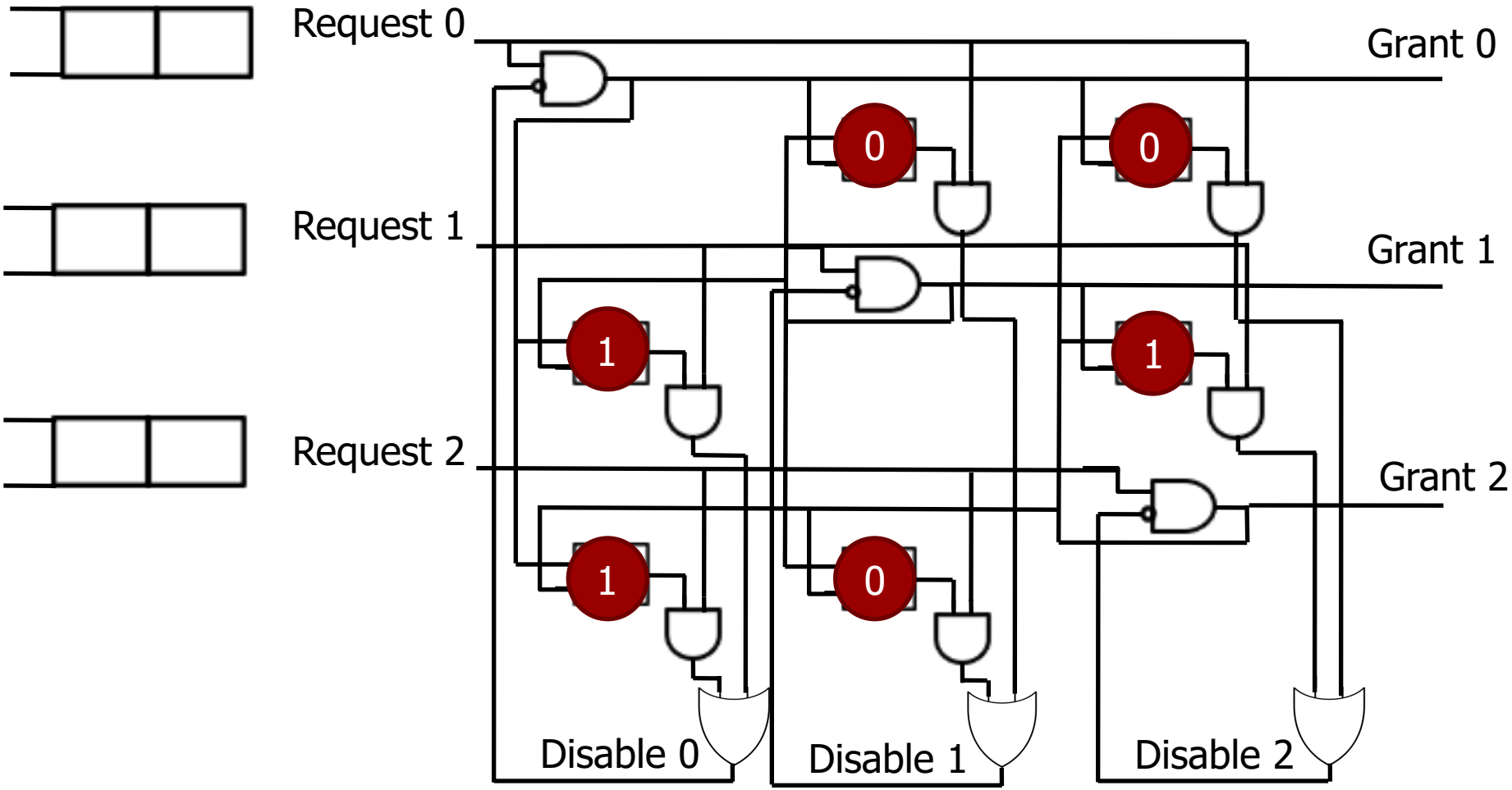
Request = 101
Grant = 001 (C_2)
Update = w2* → 0
w*2 → 1

Matrix Arbiter Example



Request = 100
Grant = 100 (A_2)
Update = w0* → 0
w*0 → 1

Matrix Arbiter Example



© Tushar Krishna, School of ECE, Georgia Tech

Allocators

- Arbiter assigns a single resource to one of a group of requesters (i.e., N : 1)
- Allocator performs a matching between a group of resources and a group of requestors (i.e., M : N)
  - Each of which may request one or more resources
- 3 rules
  - A grant can be asserted only if the corresponding request is asserted
  - At most one grant for each input (requester) may be asserted
  - At most one grant for each output (resource) can be asserted

© Tushar Krishna, School of ECE, Georgia Tech

Allocation Example

$$\text{Request Matrix, } R = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$G1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad G2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- Both G1 an G2 satisfy rules
- Which is more desirable?
  - G2
  - All three resources assigned to inputs
  - **Maximum matching**: solution containing maximum possible number of assignments

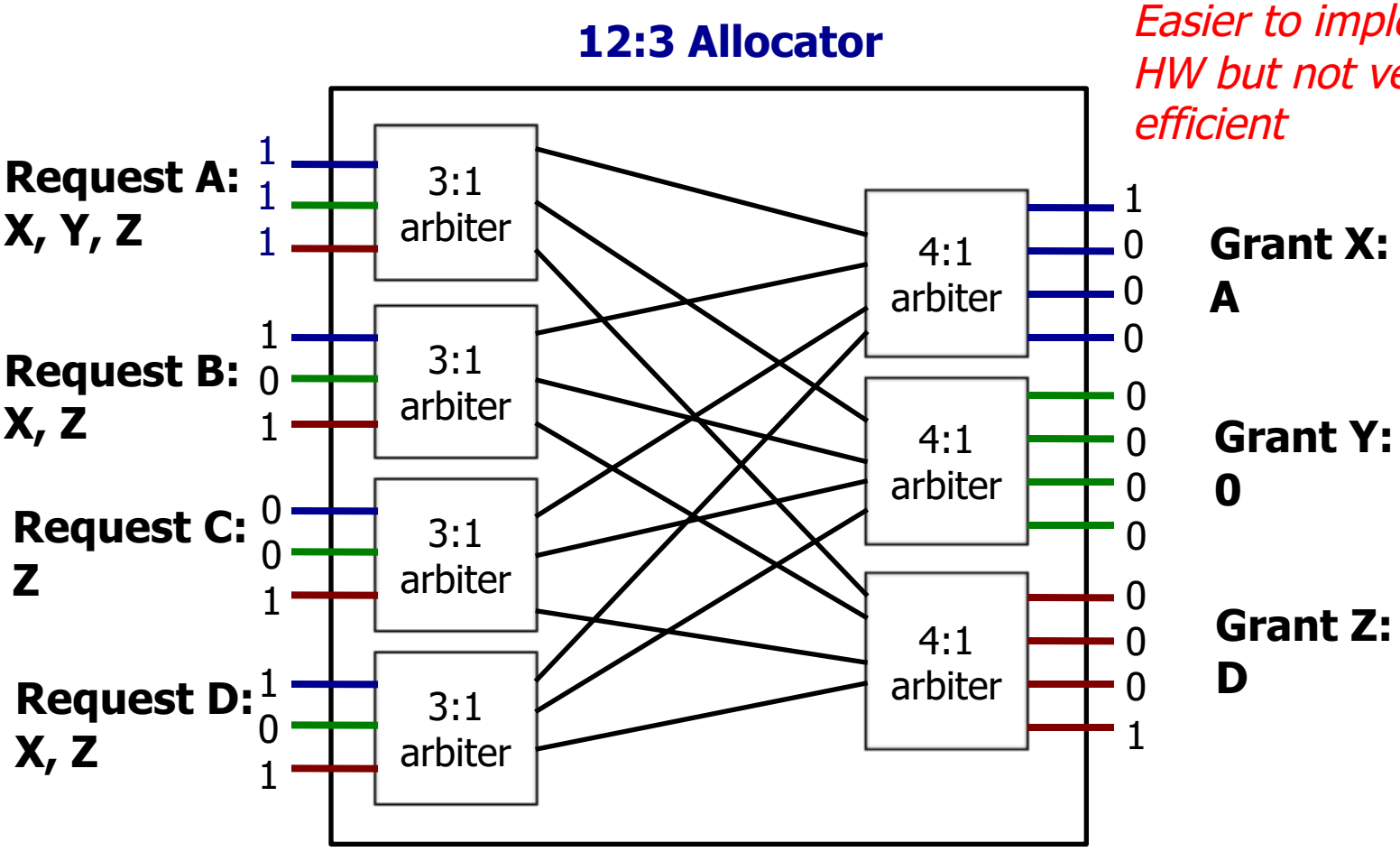
Separable Allocator

- Hard to design an allocator that is both fast and gives high-matching
- Need pipeline-able allocators for NoCs
- Separable allocator composed of arbiters
  - First stage: select single request at each input port
  - Second stage: selects single request for each output port

© Tushar Krishna, School of ECE, Georgia Tech

Separable Allocator Example

Suppose 4 requestors (A, B, C, D) and 3 resources (X, Y, Z)



 © Tushar Krishna, School of ECE, Georgia Tech

Separable Switch Allocation

- **N** input ports, **v** VCs per input port
- **N x v : N** Allocator
  - **Stage 1:** At every input port, choose one VC
    - N v:1 arbiters
  - **Stage 2:** At every output port, choose one input port
    - N N:1 arbiters

© Tushar Krishna, School of ECE, Georgia Tech

VC Allocation

- **N** input ports, **v** VCs per input port
- Separable VC Allocator
  - **Stage 1:** At every input port, choose one VC
    - N v:1 arbiters
  - **Stage 2:** At every output VC, choose an input VC
    - Nxv N:1 arbiters
- VC Selection (Kumar et al, ICCD 2007)
  - No point in allocating VC before flit wins SA
  - Maintain a pool of free VCs at every output port
    - Perform SA only if output port has at least one free VC
    - Winner of SA is granted this VC

© Tushar Krishna, School of ECE, Georgia Tech