ECE 6115 / CS 8803 - ICN

Interconnection Networks for High Performance Systems

Spring 2020

# ROUTER MICROARCHITECTURE

**Tushar Krishna**

Assistant Professor

School of Electrical and Computer Engineering

Georgia Institute of Technology

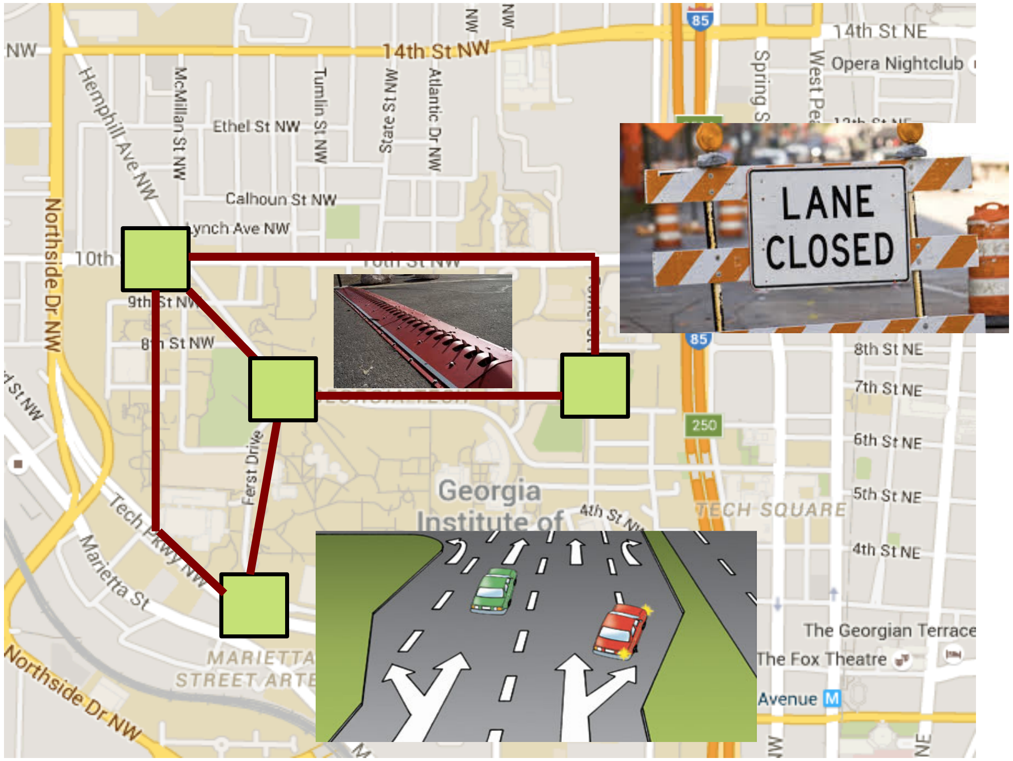
tushar@ece.gatech.edu

# NETWORK ARCHITECTURE

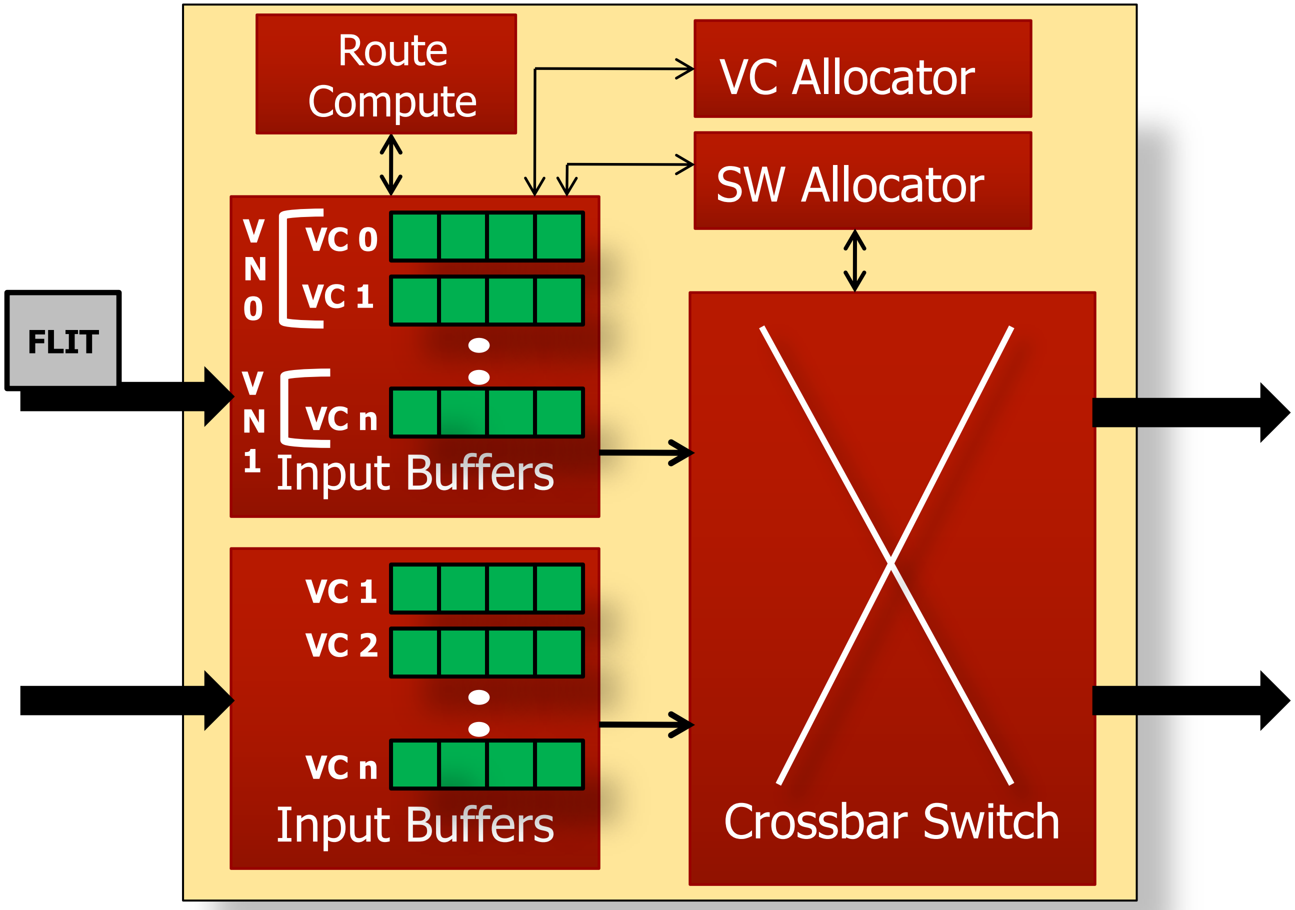
- **Topology**
  - How to connect the nodes
  - ~Road Network
- **Routing**
  - Which path should a message take
  - ~Series of road segments from source to destination
- **Flow Control**
  - When does the message have to stop/proceed
  - ~Traffic signals at end of each road segment
- **Router Microarchitecture**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

# ROUTER MICROARCHITECTURE

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



# 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**

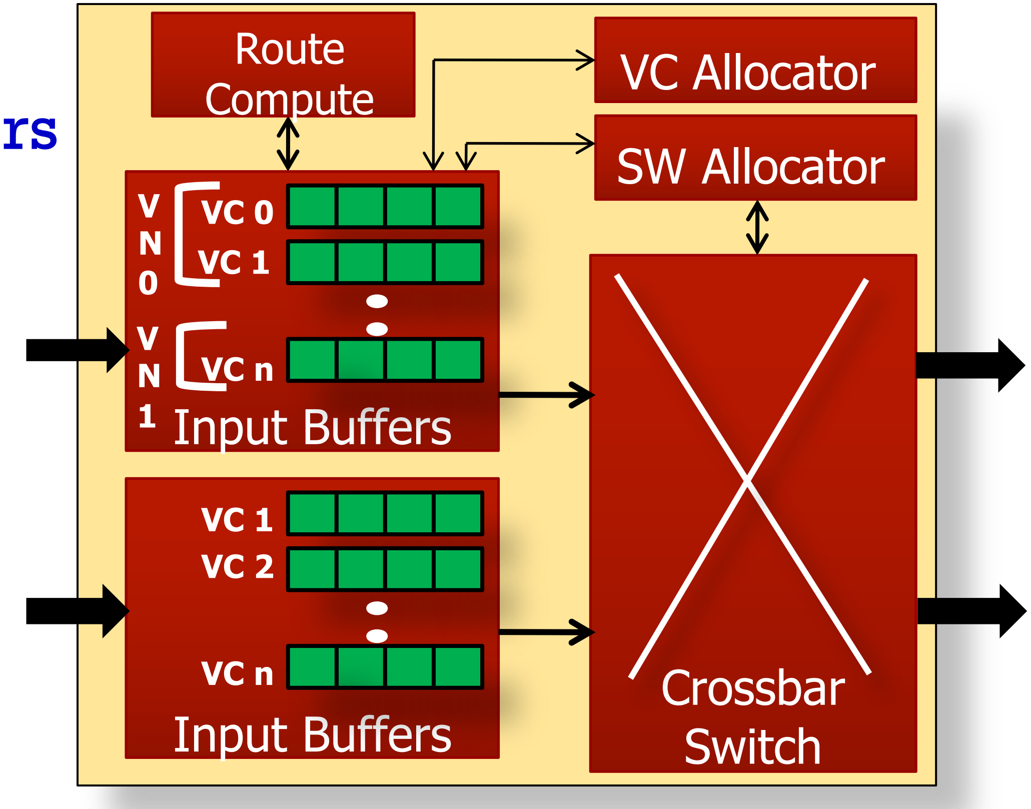
**ST: Switch Traversal**

**LT: Link Traversal**

| | | | | | | |
|---|---|---|---|---|---|---|
| **BW** | **RC** | **VA** | **SA** | **BR** | **ST** | **LT** |

# ROUTER MICROARCHITECTURE

- Components
  - Virtual Channel Buffers
  - Routing Logic
  - Allocation
    - Switch Allocation
    - VC Allocation
  - Crossbar Switch
  - Link



- Pipeline
  - 5-cycle router → 1-cycle router

# ROUTER MICROARCHITECTURE

- Components
  - Virtual Channel Buffers
  - Routing Logic
  - Allocation
    - Switch Allocation
    - VC Allocation
  - Crossbar Switch
  - Link

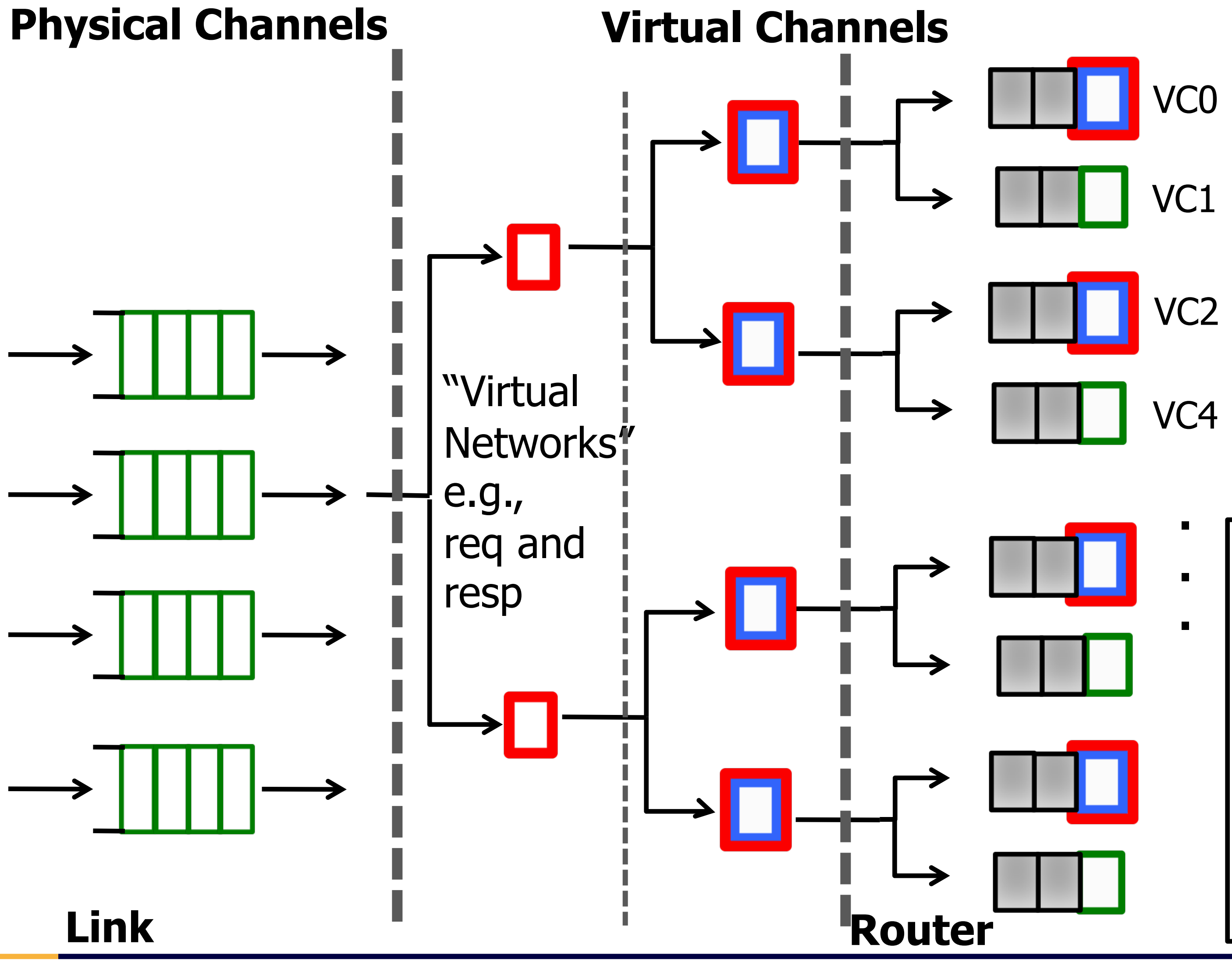
- Pipeline
  - 5-cycle router → 1-cycle router

# 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

# MINIMUM NUMBER OF BUFFERS



**For Correctness**
**- required**
**- cannot be shared**

🟥 Protocol Deadlock Avoidance

🟦 Routing Deadlock Avoidance

**For Performance**
**- not required**
***- can be shared***

🟩 HoL blocking Buffer Turnaround

▨ Multi-flit packets

# 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)

# VIRTUAL CHANNEL IMPLEMENTATION

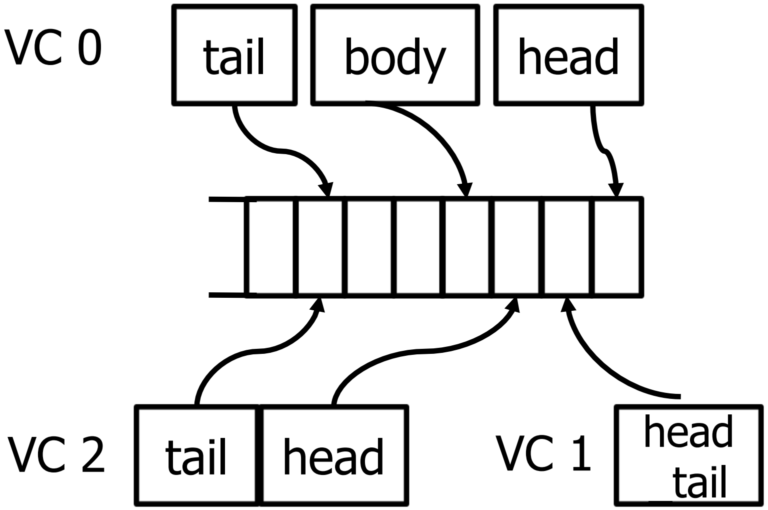
- **Storage**
  - Private Buffers Per VC
    - n-flit deep FIFO per VC
      - n >= 1, but can be smaller than the size of the packet

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



# BACKPRESSURE SIGNALING MECHANISMS

- **On/Off Flow Control**
  - downstream router signals if it can receive or not
- **Credit-based Flow Control**
  - upstream router tracks the number of free buffers available at the downstream router

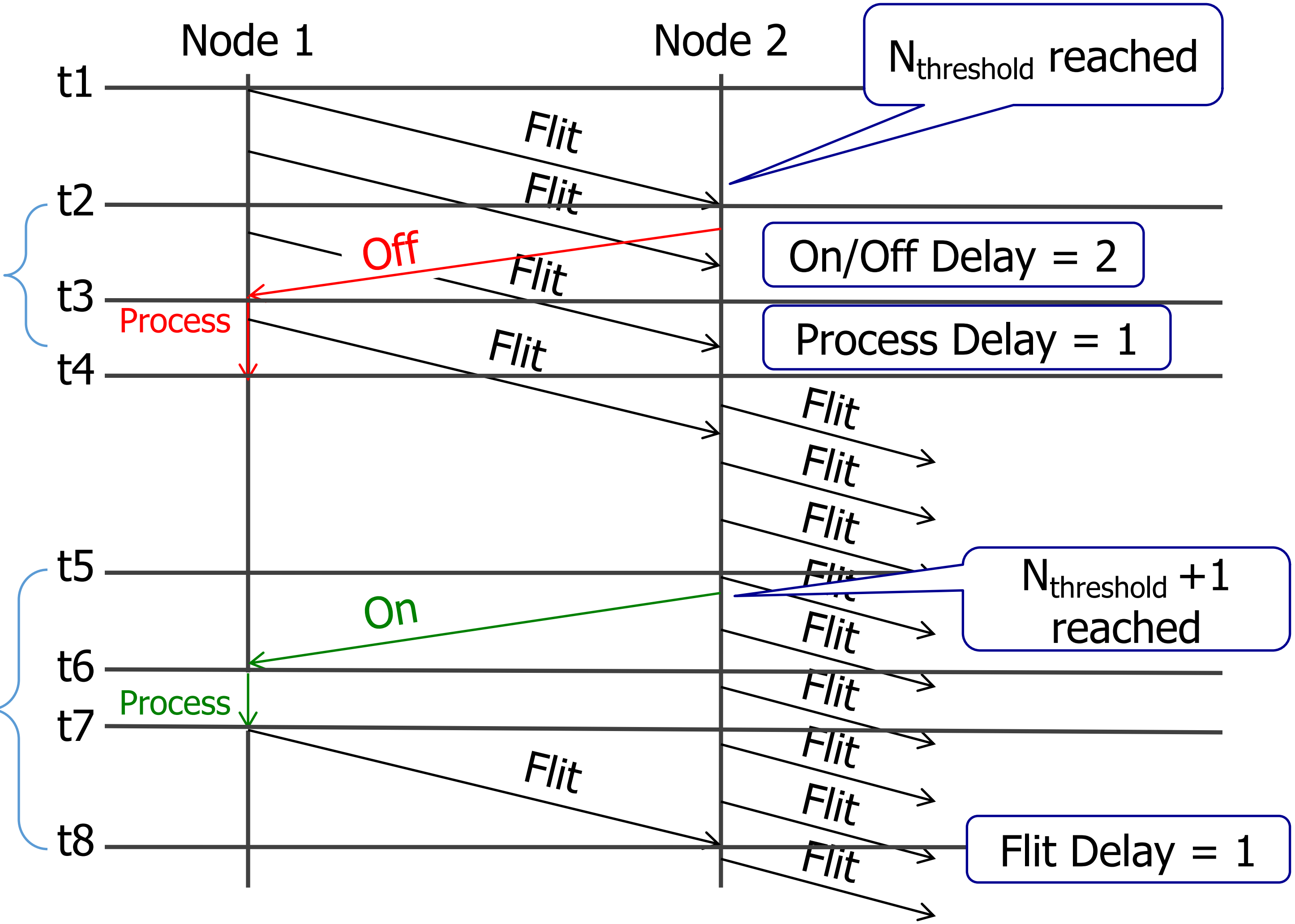
# ON/OFF FLOW CONTROL

- Downstream router sends a 1-bit on/off if it can receive or not
  - Upstream router sends only when it sees on
- Any potential challenge?
  - Delay of on/off signal
  - By the time the on/off signal reaches upstream, there might already be flits in flight
  - Need to send the off signal *once the number of buffers reaches a threshold* such that all potential in-flight flits have a free buffer

# ON/OFF TIMELINE WITH N BUFFERS

$N_{threshold}$ set to 3 to prevent flits departing Node 1 before t4 from overflowing

$N_{total}$ set so that Node 2 does not run out of flits to send between t5 and t8



$N_{threshold}$ reached

On/Off Delay = 2

Process Delay = 1

$N_{threshold}$ +1 reached

Flit Delay = 1

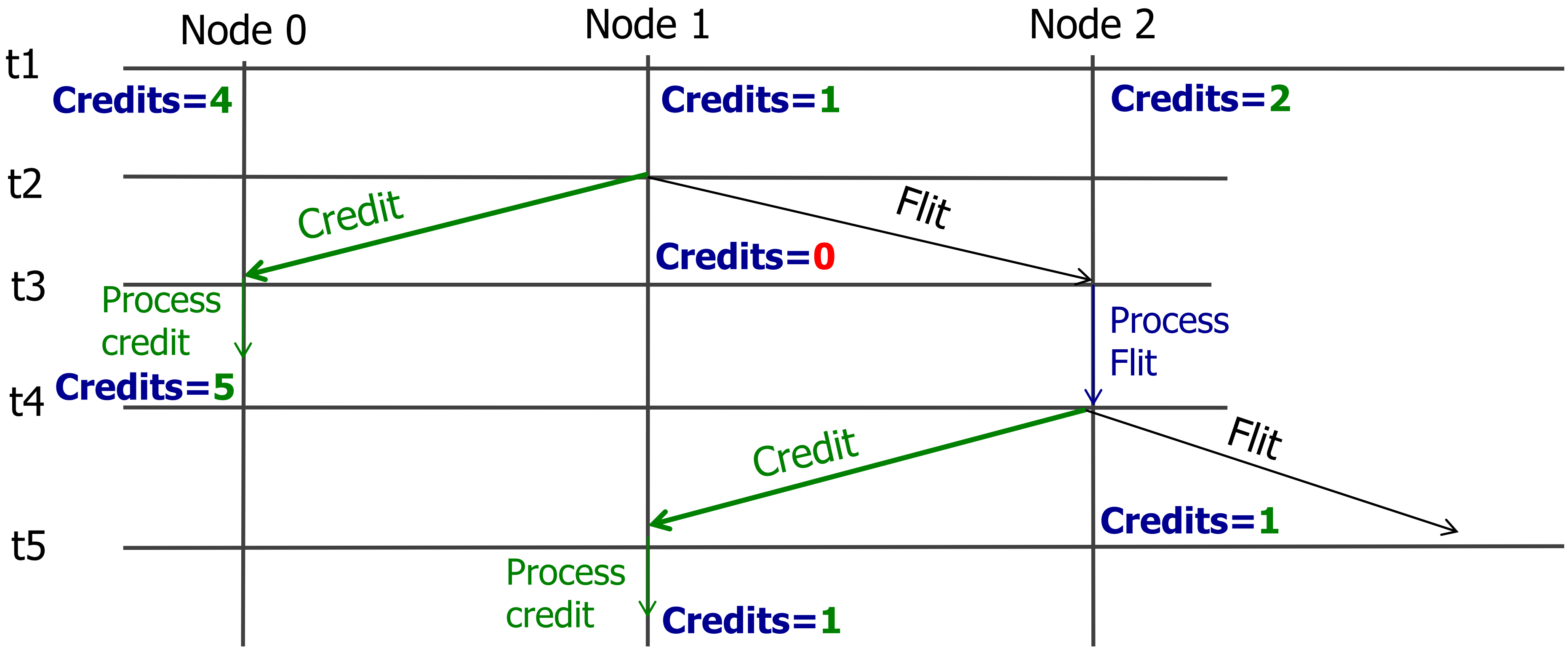
# BACKPRESSURE SIGNALING MECHANISMS

- **On/Off Flow Control**
  - Pros
    - Low overhead: one-bit signal from downstream to upstream node, only switches when threshold crossed
  - Cons
    - Inefficient buffer utilization – have to design assuming worst case of $N_{threshold}$ flights in flight

# CREDIT-BASED FLOW CONTROL

- **Upstream router** tracks the **number of free buffers available at the downstream router**
  - Upstream router sends only if credits > 0
- When should credit be decremented at upstream router?
  - When a flit is sent to the downstream router
- When should credit be incremented at upstream router?
  - When a flit leaves the downstream router

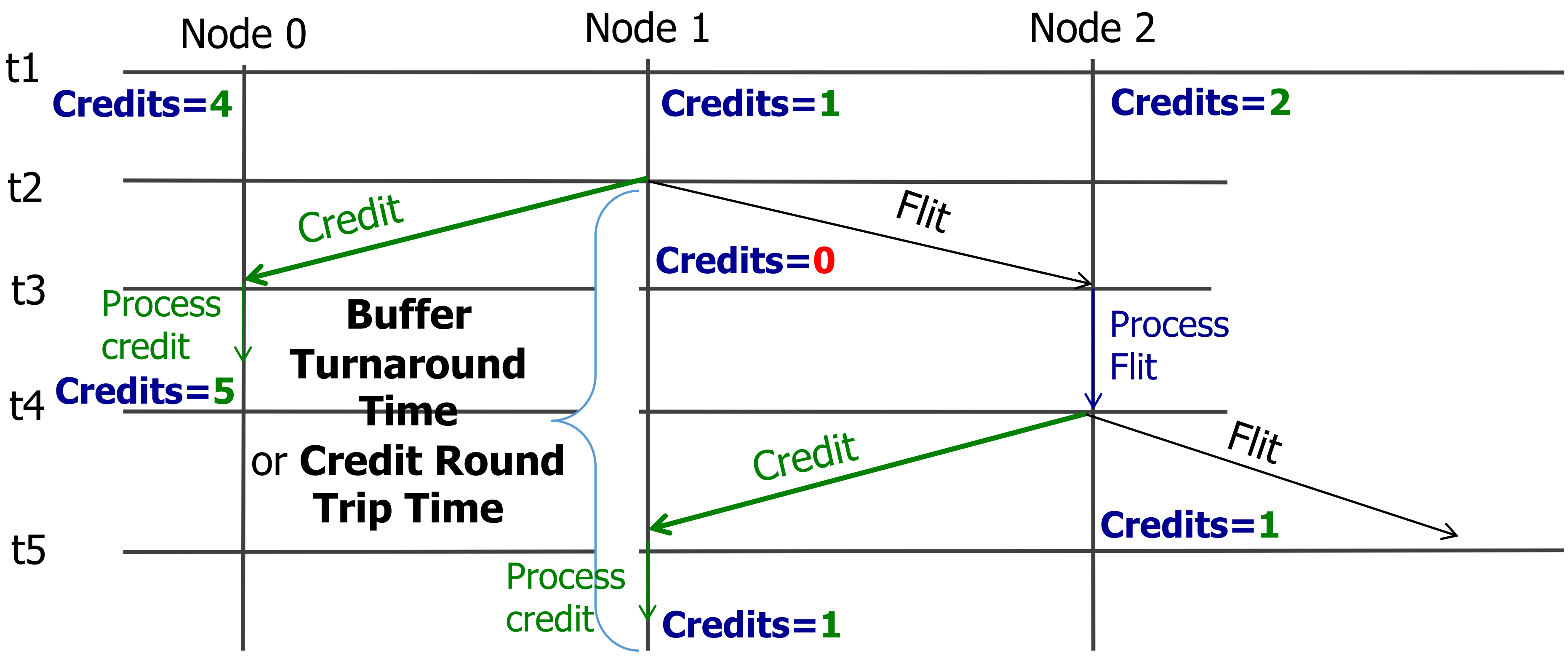
# CREDIT TIMELINE



# BACKPRESSURE SIGNALING MECHANISMS

- **On/Off Flow Control**
  - Pros
    - Low overhead: one-bit signal
  - Cons
    - Inefficient buffer utilization – have to design assuming worst case of $N_{threshold}$ flights in flight
- **Credit Flow Control**
  - Pros
    - Each buffer fully utilized - an keep sending till credits are zero (unlike on/off)
  - Cons
    - More signaling – need to signal upstream for every flit

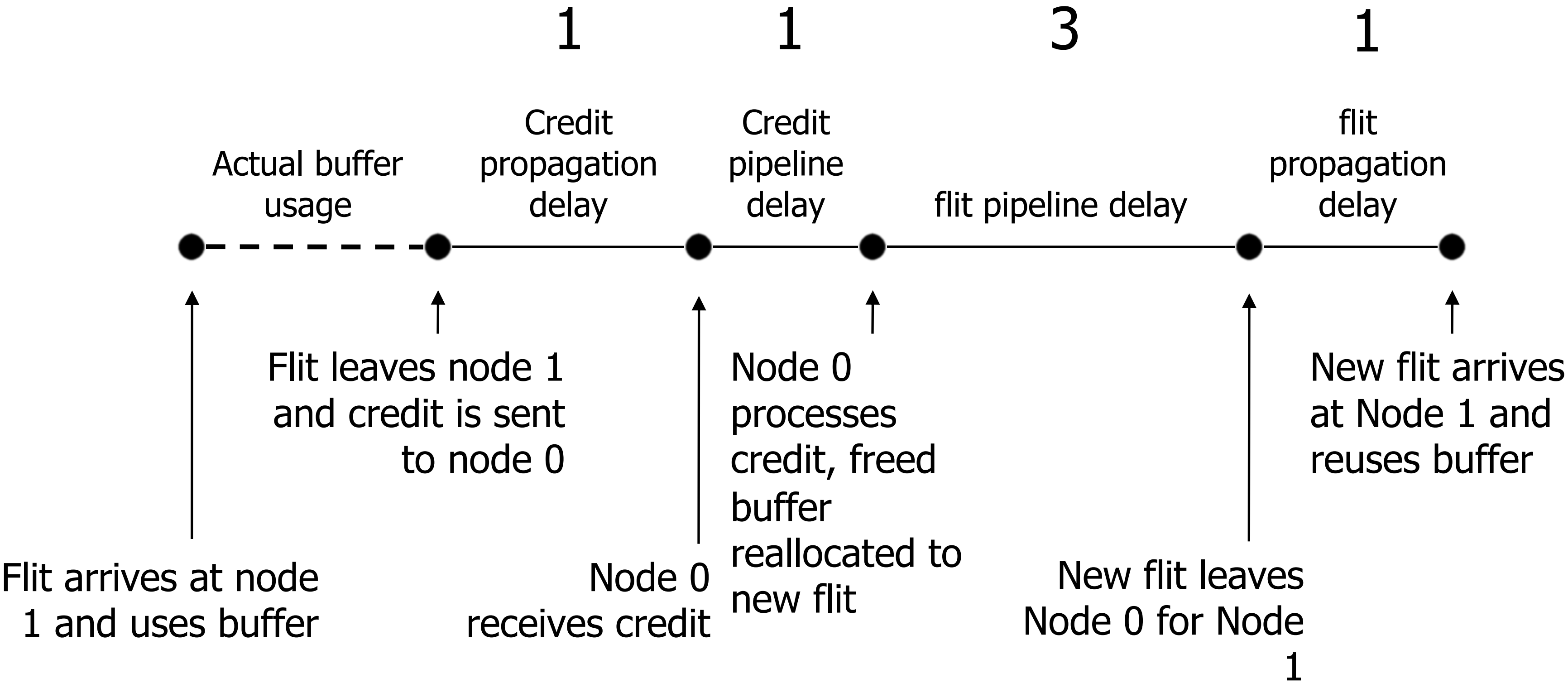
# BACKPRESSURE AND BUFFER SIZING



*No flit can be sent into this buffer during this delay*

To prevent backpressure from limiting throughput, number of buffers >= turnaround time

# "BUFFER TURNAROUND TIME"



1 1 3 1

Actual buffer usage

Credit propagation delay

Credit pipeline delay

flit pipeline delay

flit propagation delay

Flit arrives at node 1 and uses buffer

Flit leaves node 1 and credit is sent to node 0

Node 0 receives credit

Node 0 processes credit, freed buffer reallocated to new flit

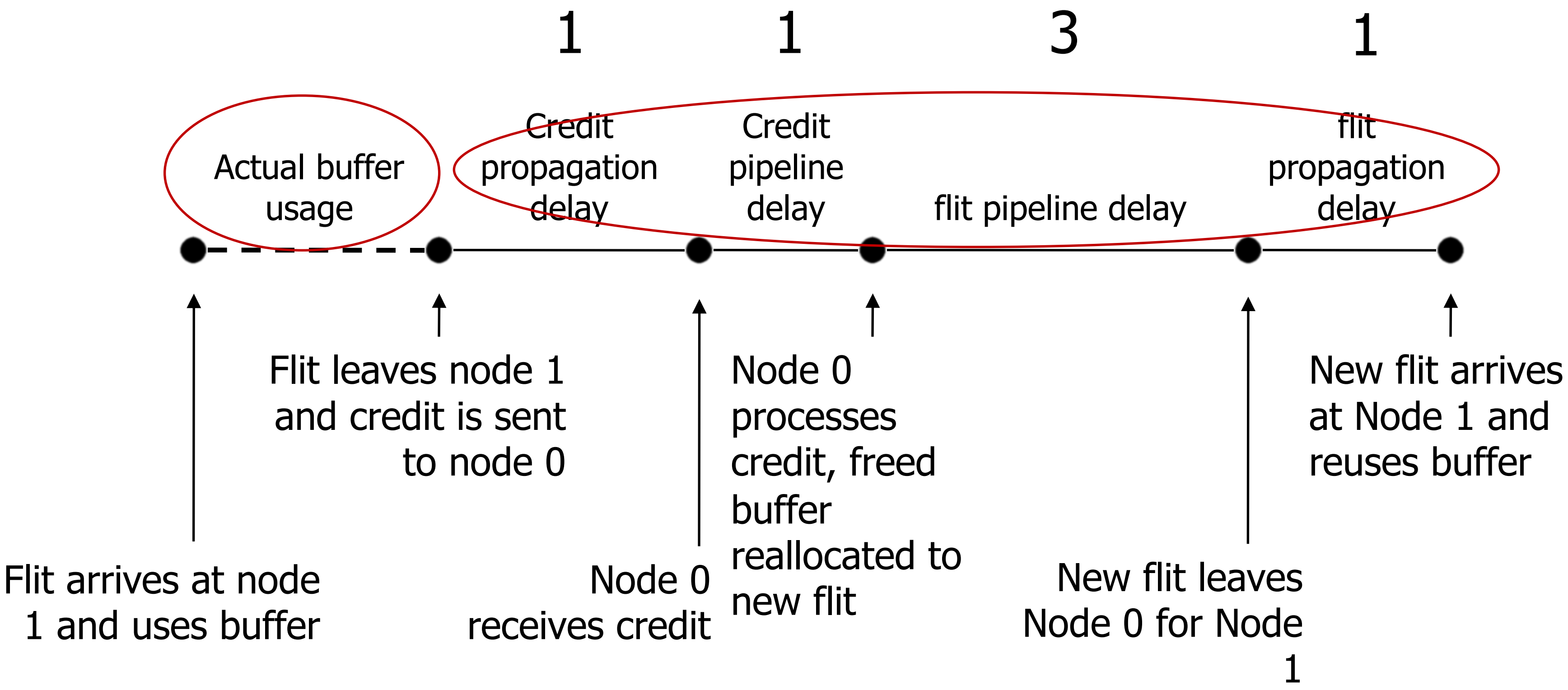
New flit leaves Node 0 for Node 1

New flit arrives at Node 1 and reuses buffer

How many buffers needed? 1+1+3+1 = 6

How many buffers needed in on/off flow-control? 6 + 2 = 8 (off propogation + processing)

# BUT THIS IS INEFFICIENT



1 1 3 1

Actual buffer usage

Credit propagation delay

Credit pipeline delay

flit pipeline delay

flit propagation delay

Flit arrives at node 1 and uses buffer

Flit leaves node 1 and credit is sent to node 0

Node 0 receives credit

Node 0 processes credit, freed buffer reallocated to new flit

New flit leaves Node 0 for Node 1

New flit arrives at Node 1 and reuses buffer

***See: Flit Rsvn Flow Control, HPCA 2000***

# ROUTER MICROARCHITECTURE

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

# 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

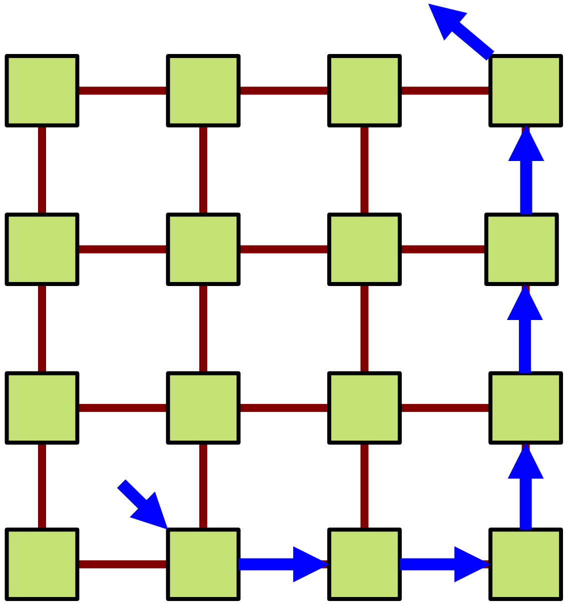
    + Can reconfigure routes based on faults

    + 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



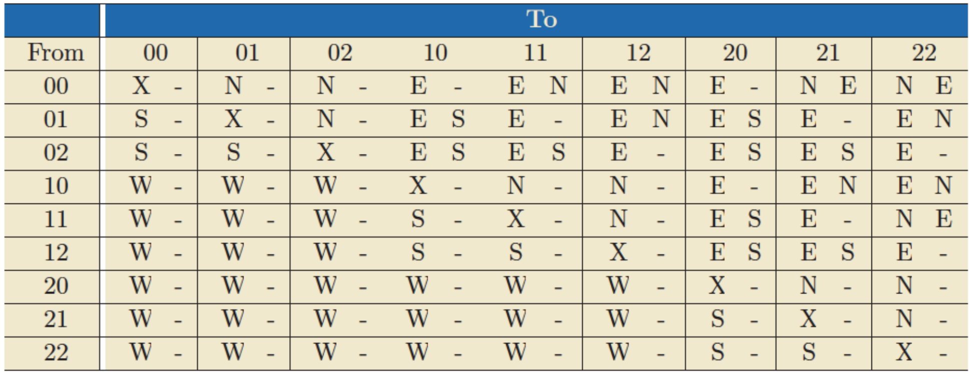
# ROUTING LOGIC

- **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

# ROUTER MICROARCHITECTURE

- Components
  - Virtual Channel Buffers
  - Routing Logic
  - Allocation
    - Switch Allocation
    - VC Allocation
  - Crossbar Switch
  - Link

- Pipeline
  - 5-cycle router → 1-cycle router

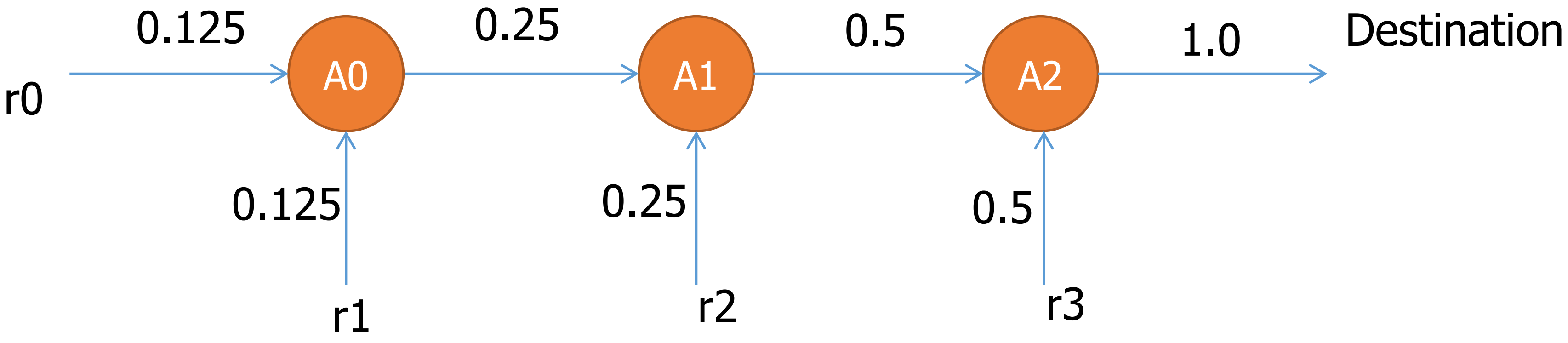
# SWITCH AND VC ALLOCATION

- "**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

# 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

# LOCALLY FAIR EXAMPLE

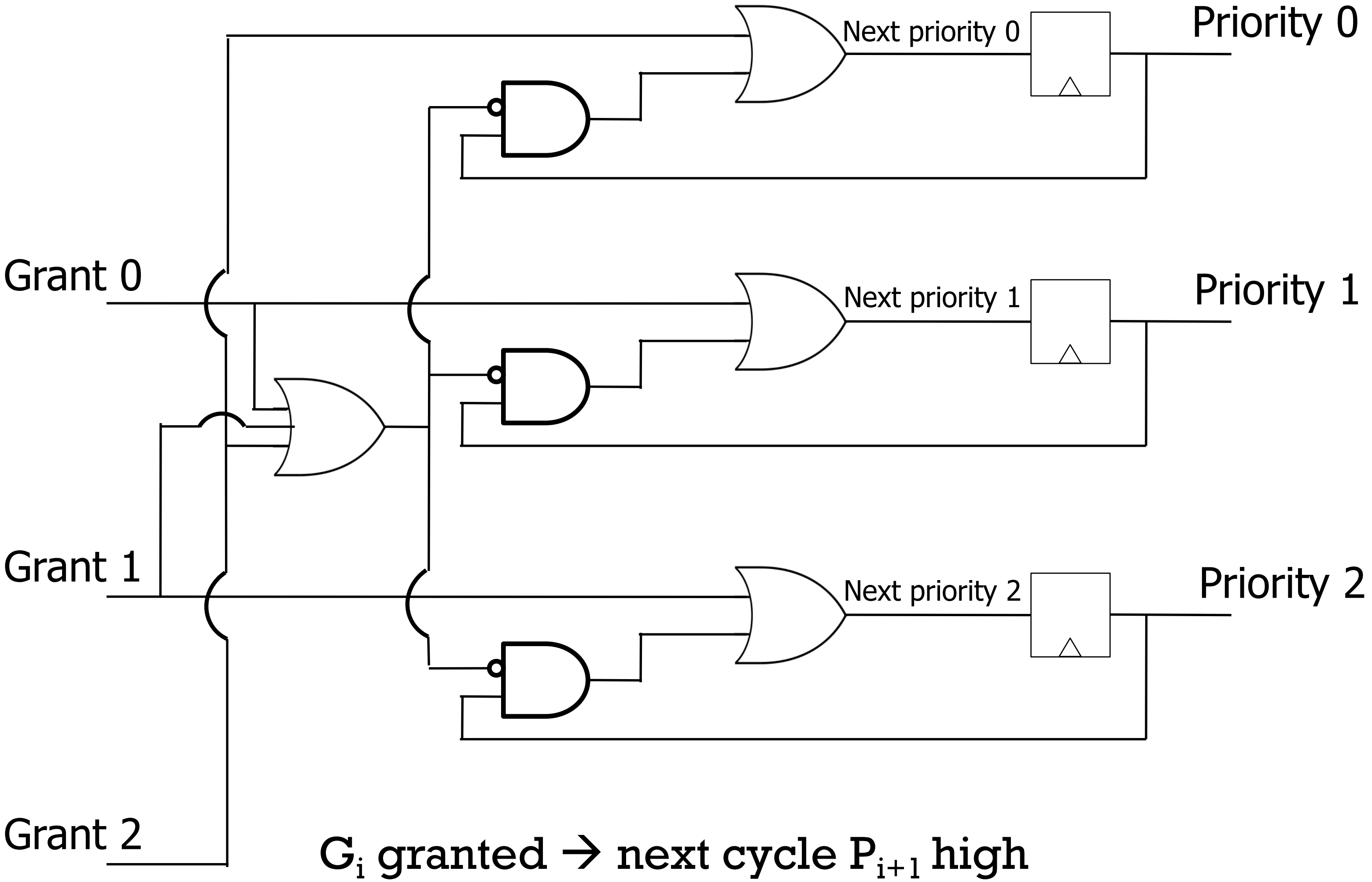


R3 receives 4 times the bandwidth as r0, even though individual arbiters provide strong fairness

# 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

# ROUND ROBIN ARBITER IMPLEMENTATION

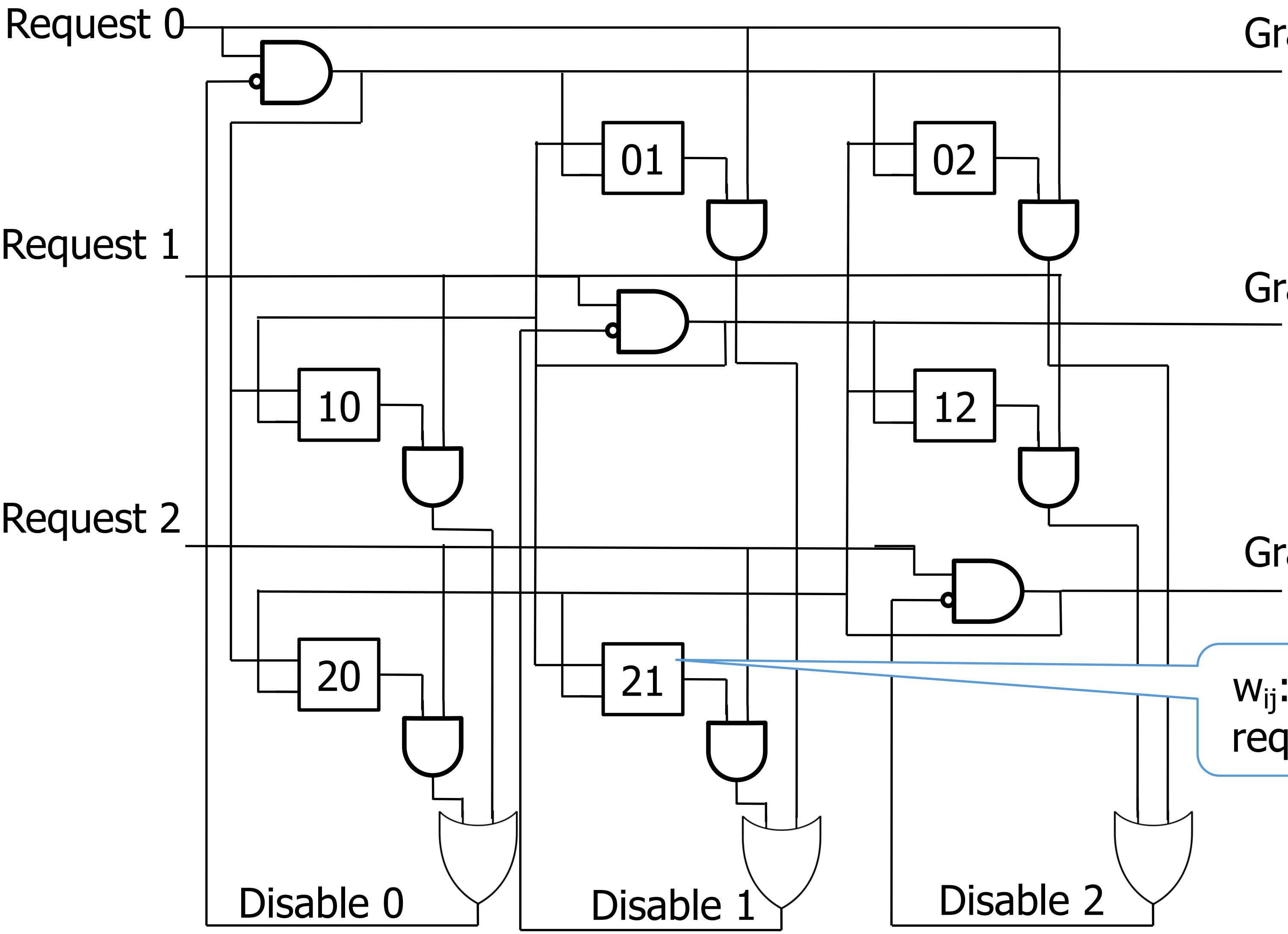


$G_i$ granted → next cycle $P_{i+1}$ high

# 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

# MATRIX ARBITER IMPLEMENTATION



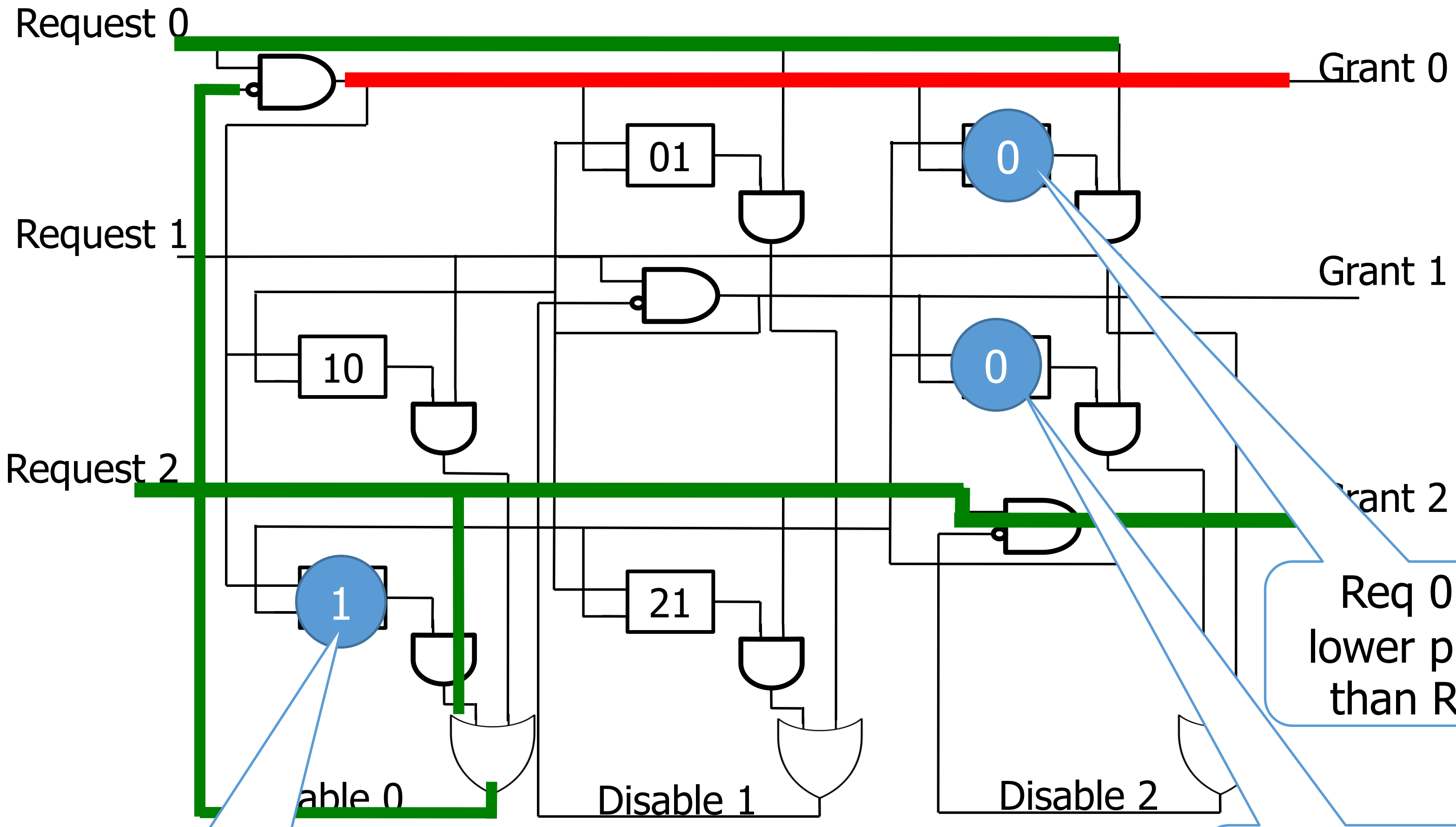
$w_{ij}$: Priority of req i wrt req j

# 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.

# MATRIX ARBITER IMPLEMENTATION

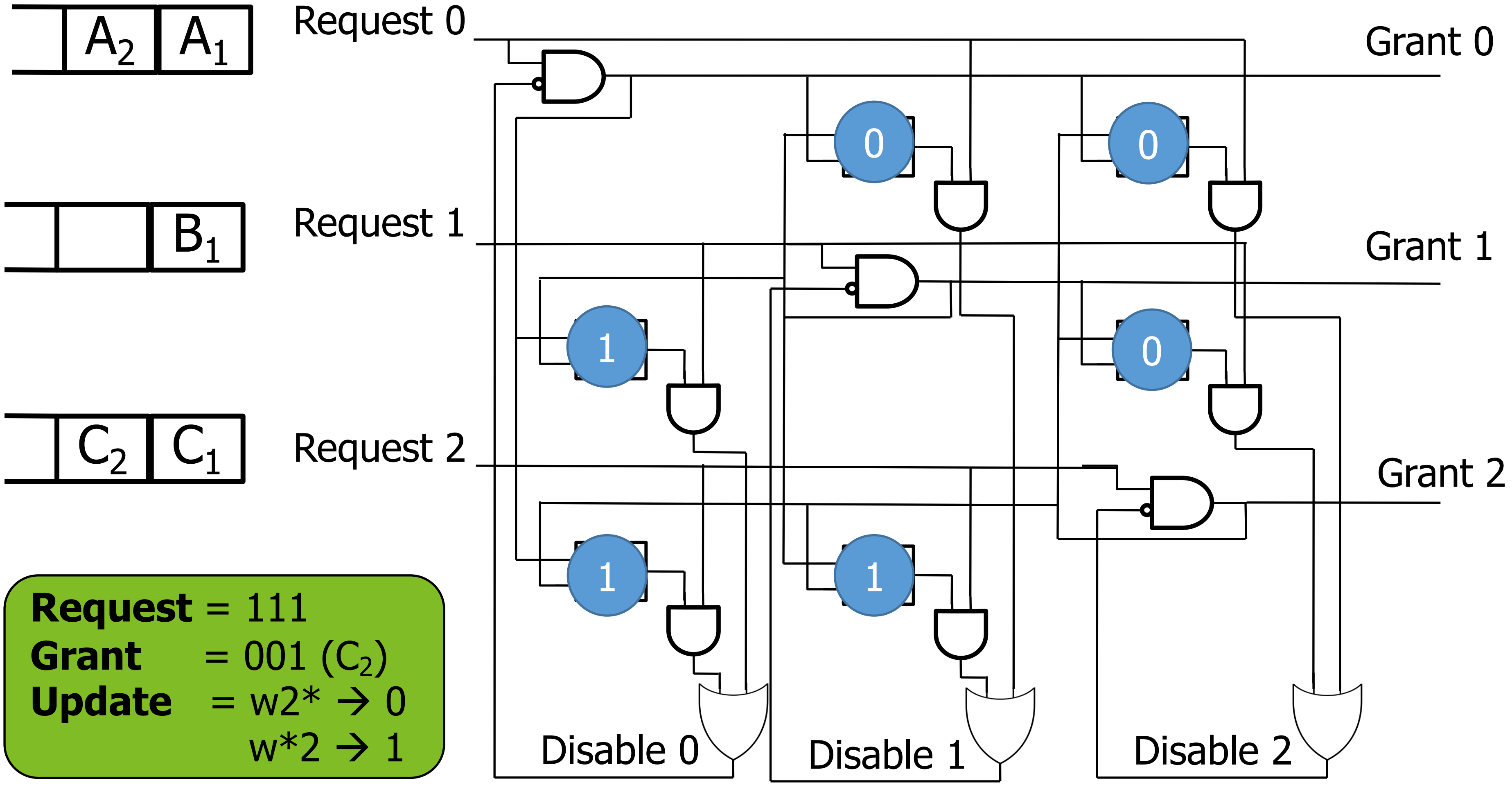


Req 0 has lower priority than Req 2

Req 2 has higher priority than Req 0

Req 1 has lower priority than Req 2

# MATRIX ARBITER EXAMPLE

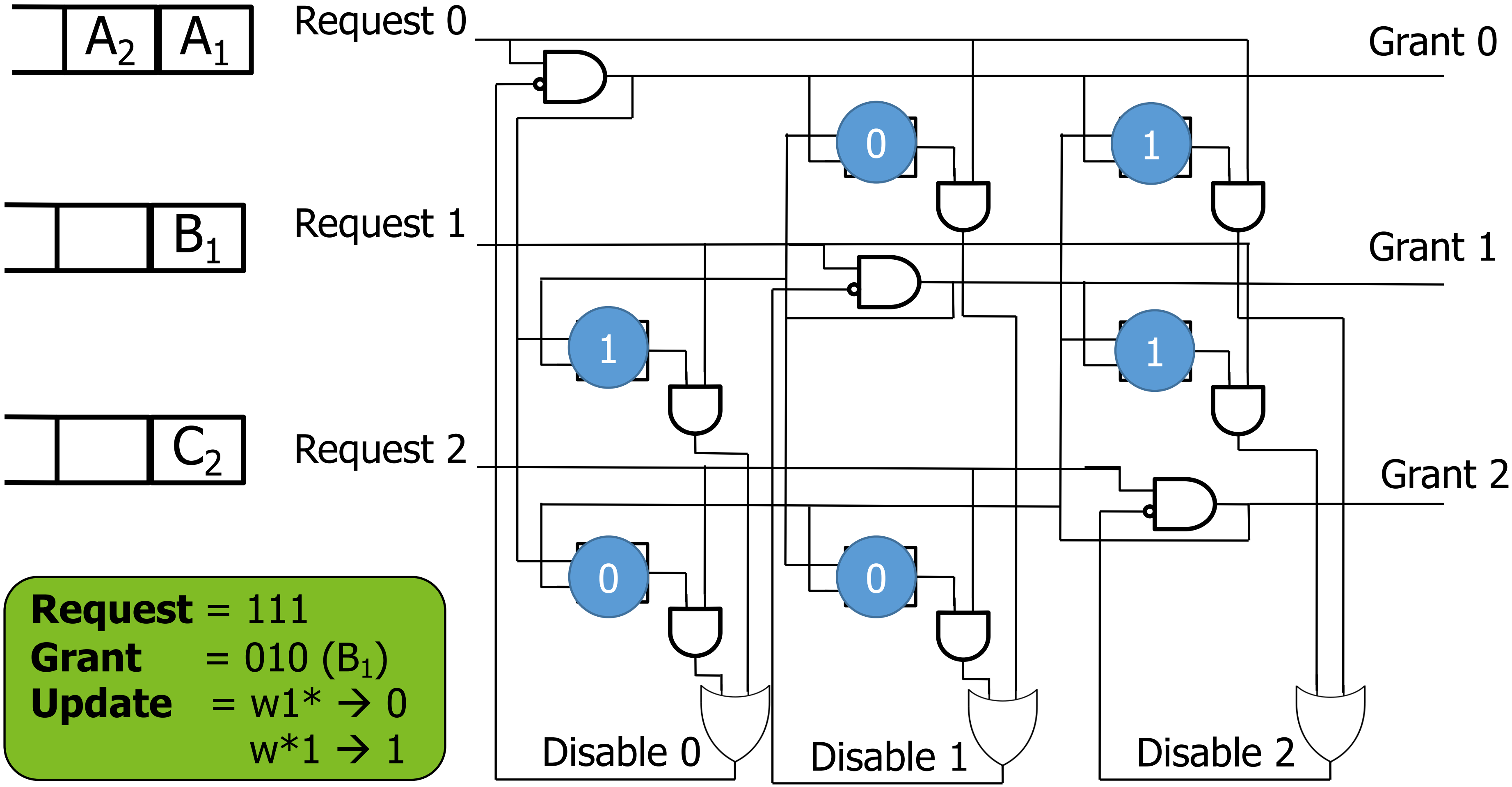


**Request** = 111
**Grant** = 001 ($C_2$)
**Update** = w2* → 0
w*2 → 1

1 and 2 have priority over 0

2 has priority over 1

# MATRIX ARBITER EXAMPLE

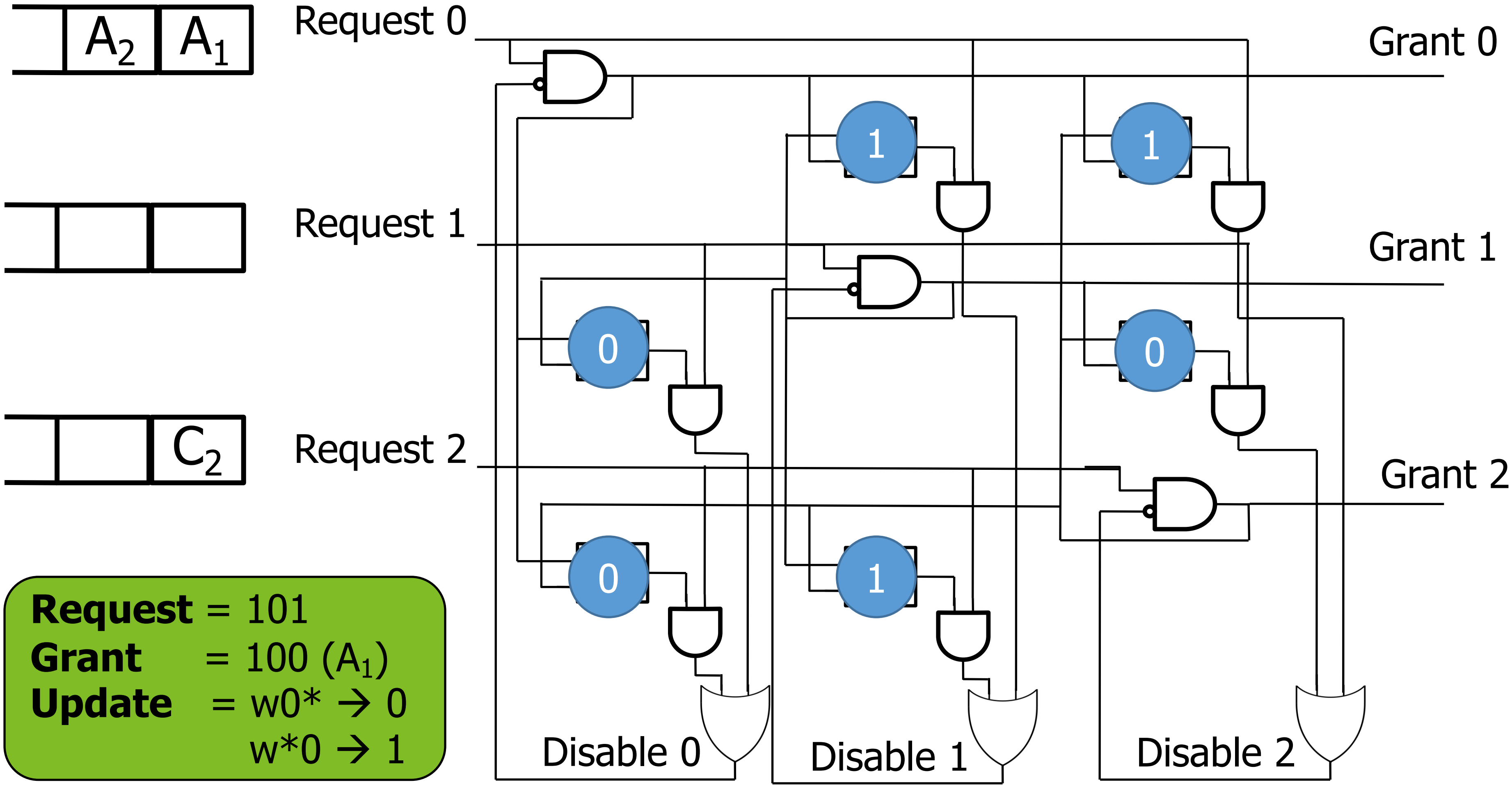


**Request** = 111
**Grant** = 010 ($B_1$)
**Update** = w1* → 0
w*1 → 1

1 has priority over 0

0 and 1 have priority over 2

# MATRIX ARBITER EXAMPLE

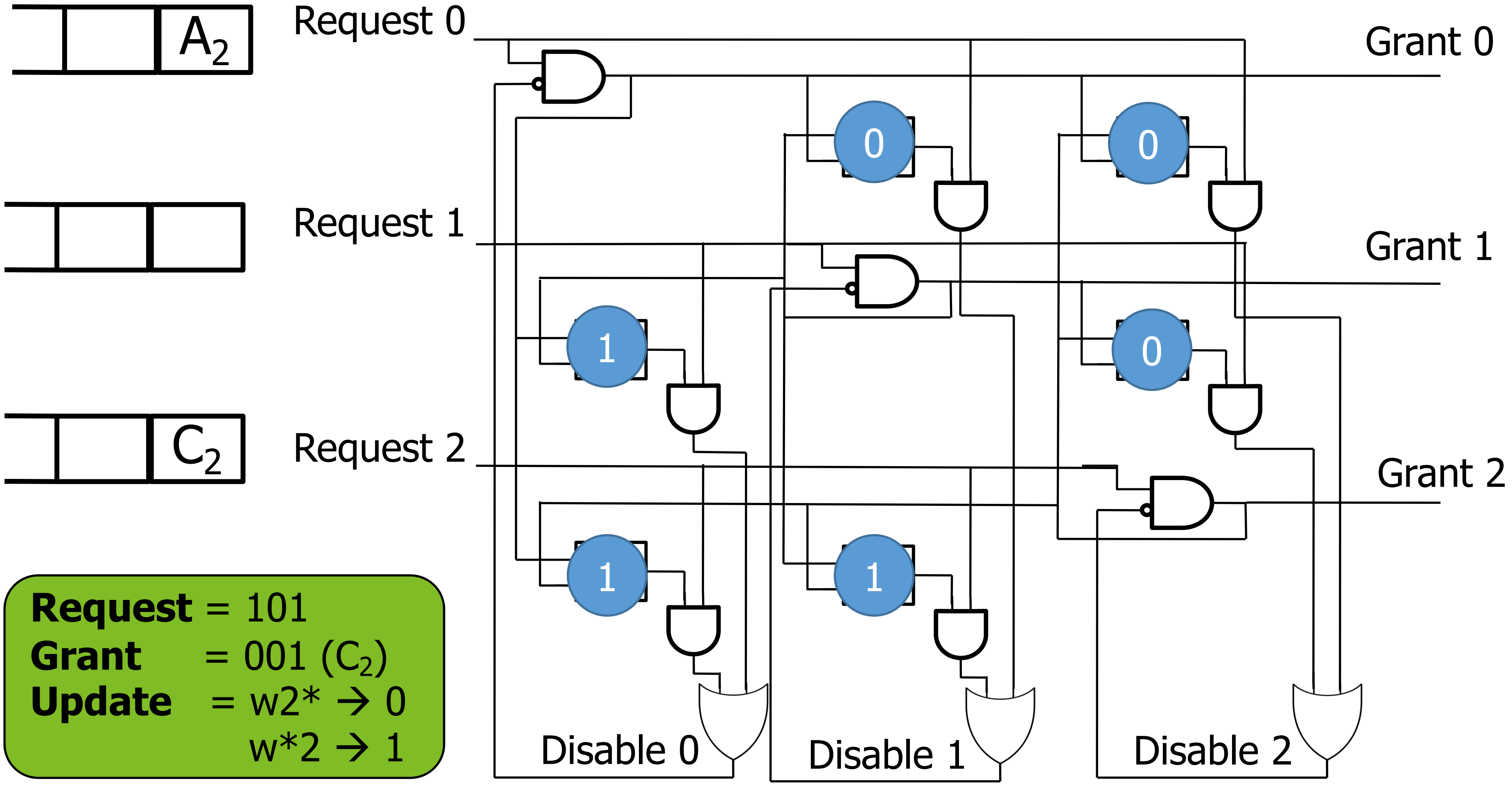


**Request** = 101
**Grant** = 100 ($A_1$)
**Update** = w0* → 0
w*0 → 1

0 and 2 have priority over 1

0 has priority over 2

# MATRIX ARBITER EXAMPLE

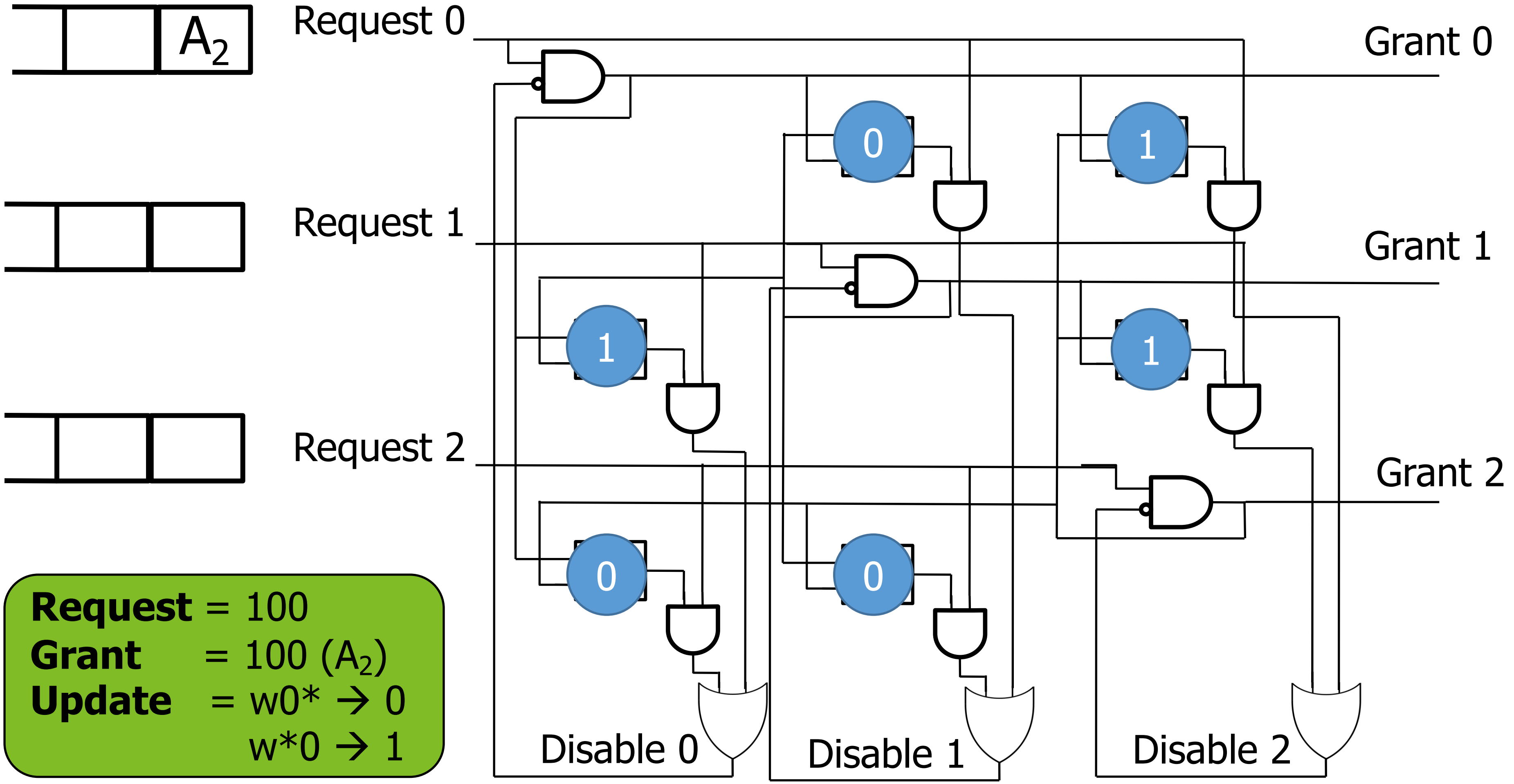


**Request** = 101
**Grant** = 001 ($C_2$)
**Update** = w2* → 0
w*2 → 1

1 and 2 have priority over 0

2 has priority over 1

# MATRIX ARBITER EXAMPLE

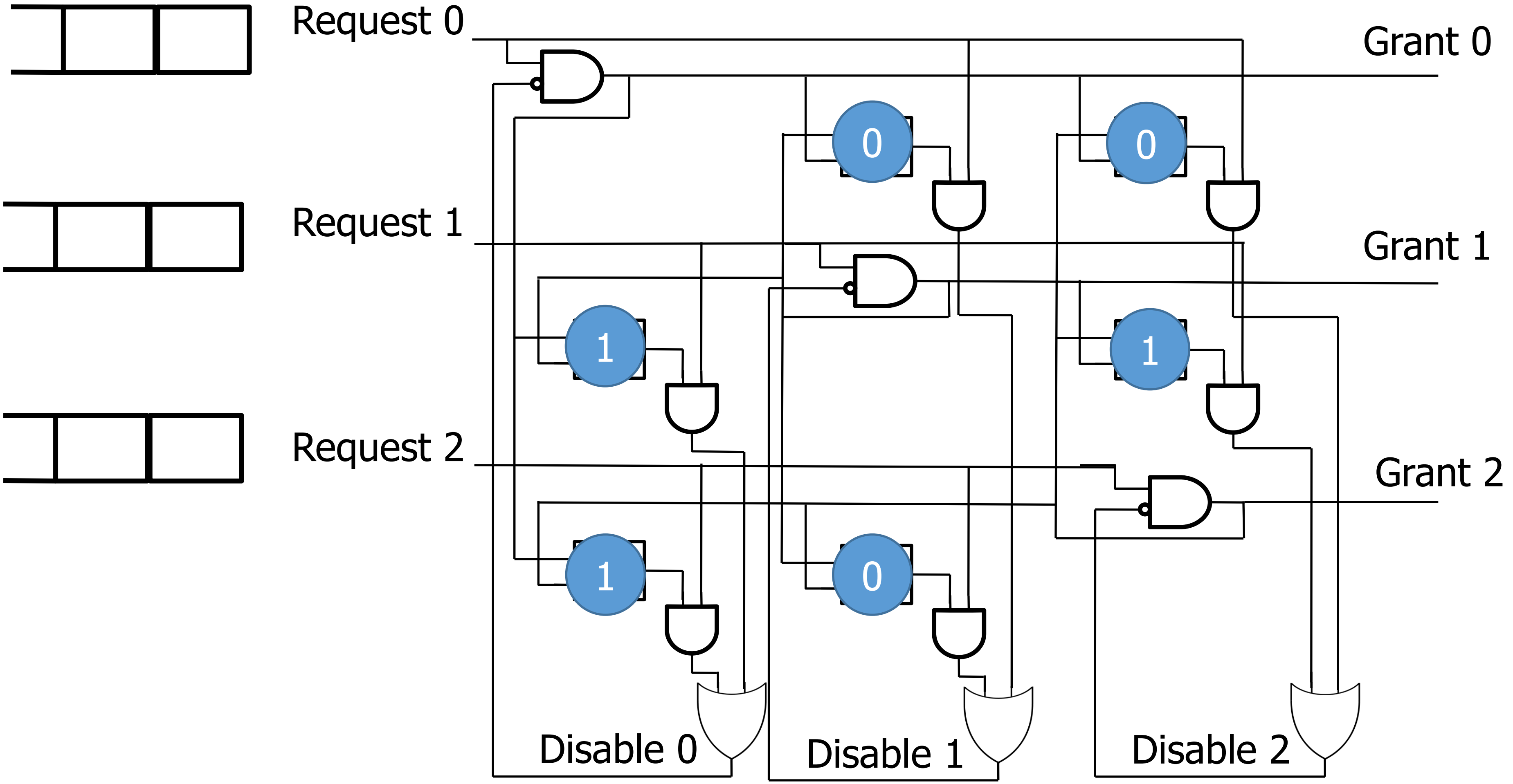


**Request** = 100
**Grant** = 100 ($A_2$)
**Update** = w0* → 0
w*0 → 1

1 and 2 have priority over 0

2 has priority over 1

# MATRIX ARBITER EXAMPLE



1 and 2 have priority over 0

1 has priority over 2

# 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

# ALLOCATION EXAMPLE

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

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

- 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

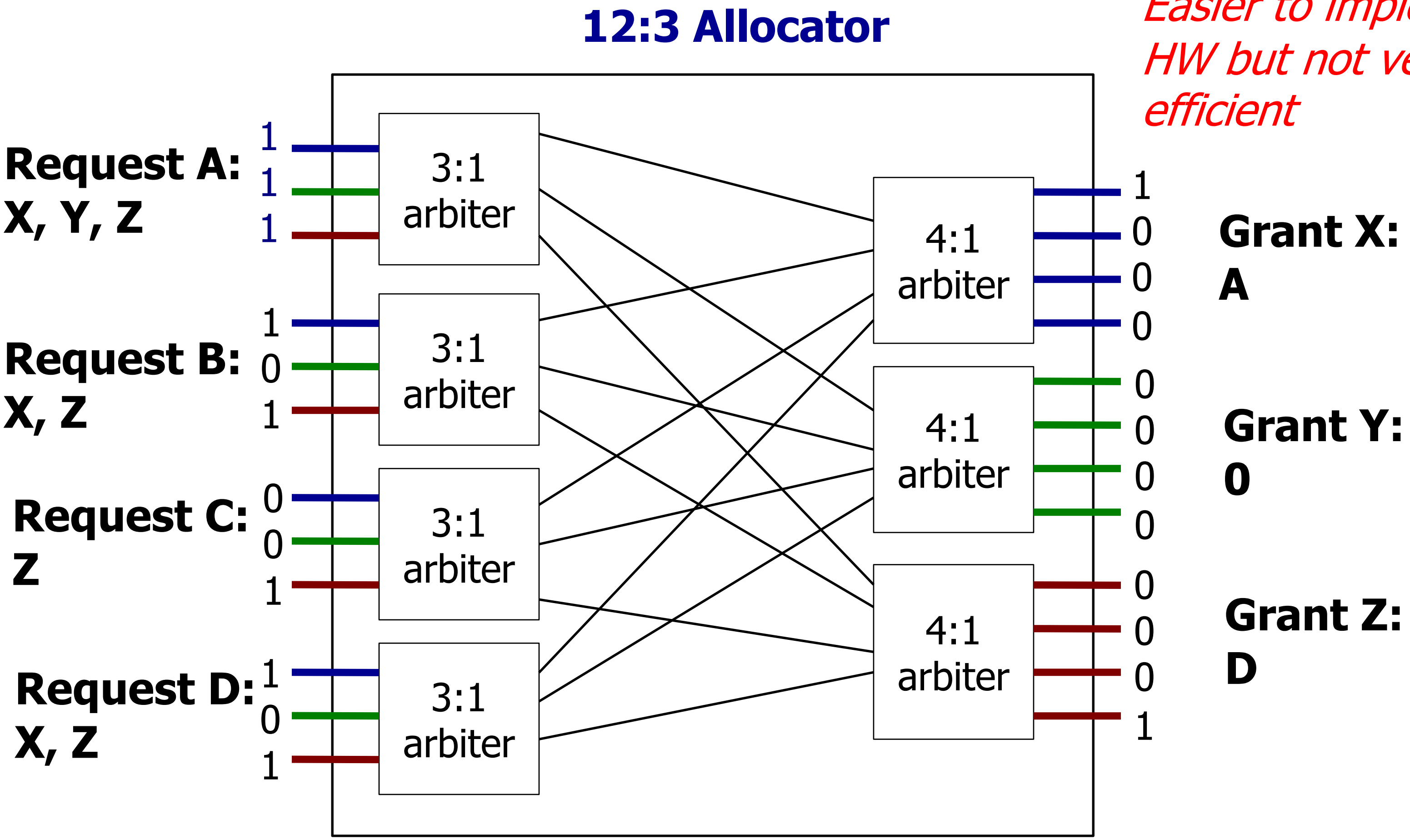
# SEPARABLE ALLOCATOR

- 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

# SEPARABLE ALLOCATOR EXAMPLE

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

*Easier to implement in HW but not very efficient*



# SWITCH ALLOCATION

- **N** input ports, **v** VCs per input port, **N** output ports
  - **N x v** : **N** Allocator
- **Implementation Choices**
  - **Separable Switch Allocator**
    - Allocator composed of Arbiters
    - **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
    - Arbiters: Round-Robin, Queueing, Matrix, …

# TRADE-OFFS

- Pros of Separable?
  - Simple
  - Pipeline-able ➔ Increased frequency
  - Can be synthesized from RTL
- Cons
  - May not be very efficient in the overall matching
    - Bad choice in first phase can limit matching
    - Lower throughput of system
- Which design did Intel SCC use?
  - "Wavefront Allocator"

# WAVEFRONT ALLOCATOR

- Arbitrate among requests for inputs and outputs simultaneously
- A diagonal group of cells is assigned a set of row and column "tokens"
- If a cell is requesting a resource, it needs to consume a row token and a column token to grant its request
  - Intuition: each row represents a request, each column represents a resource. Getting a token for both implies a grant
- Cells that cannot use tokens pass row tokens to the right and column tokens to the left

# EXAMPLE

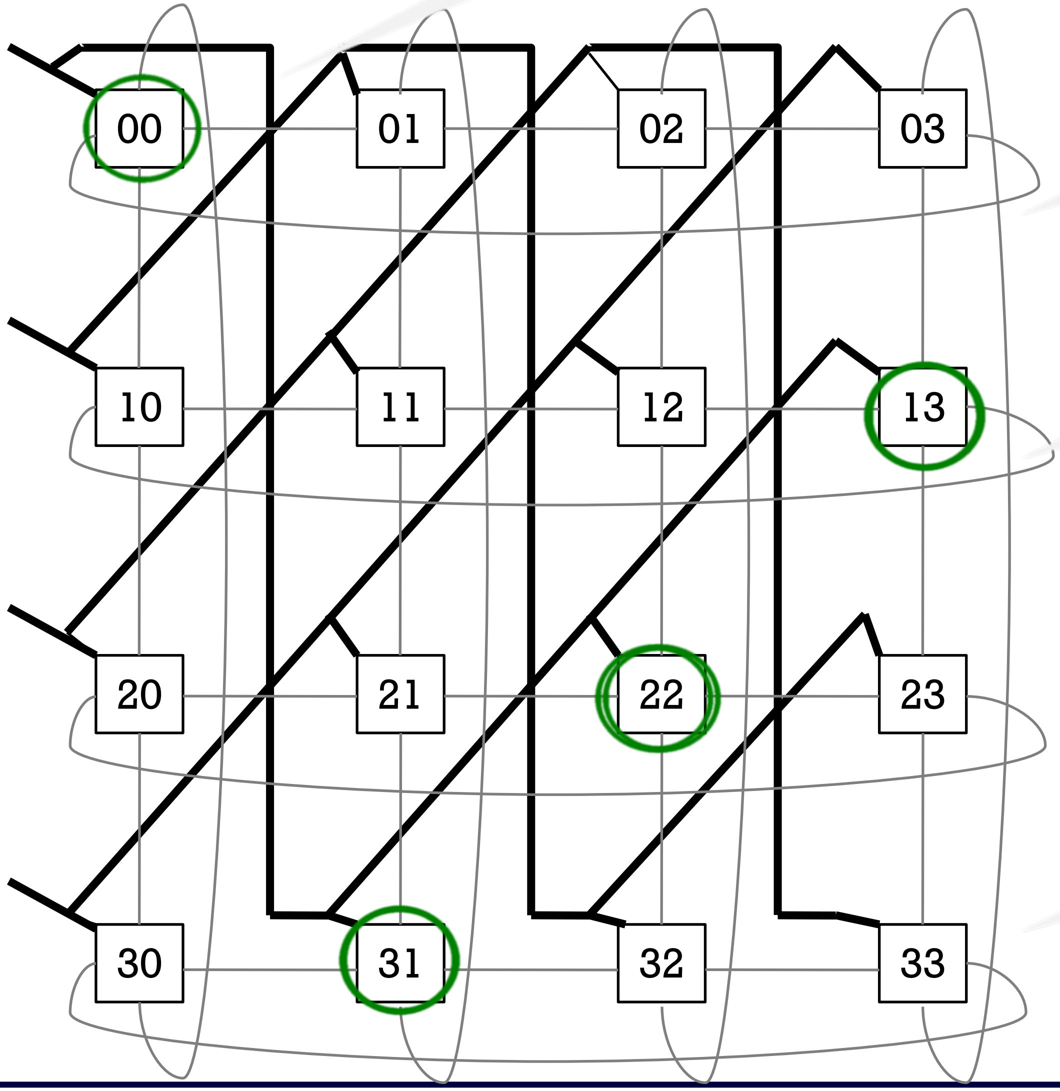
Tokens inserted at P0

A requesting Resources 0, 1 ,2

B requesting Resources 0, 1

C requesting Resource 0

D requesting Resources 0, 2

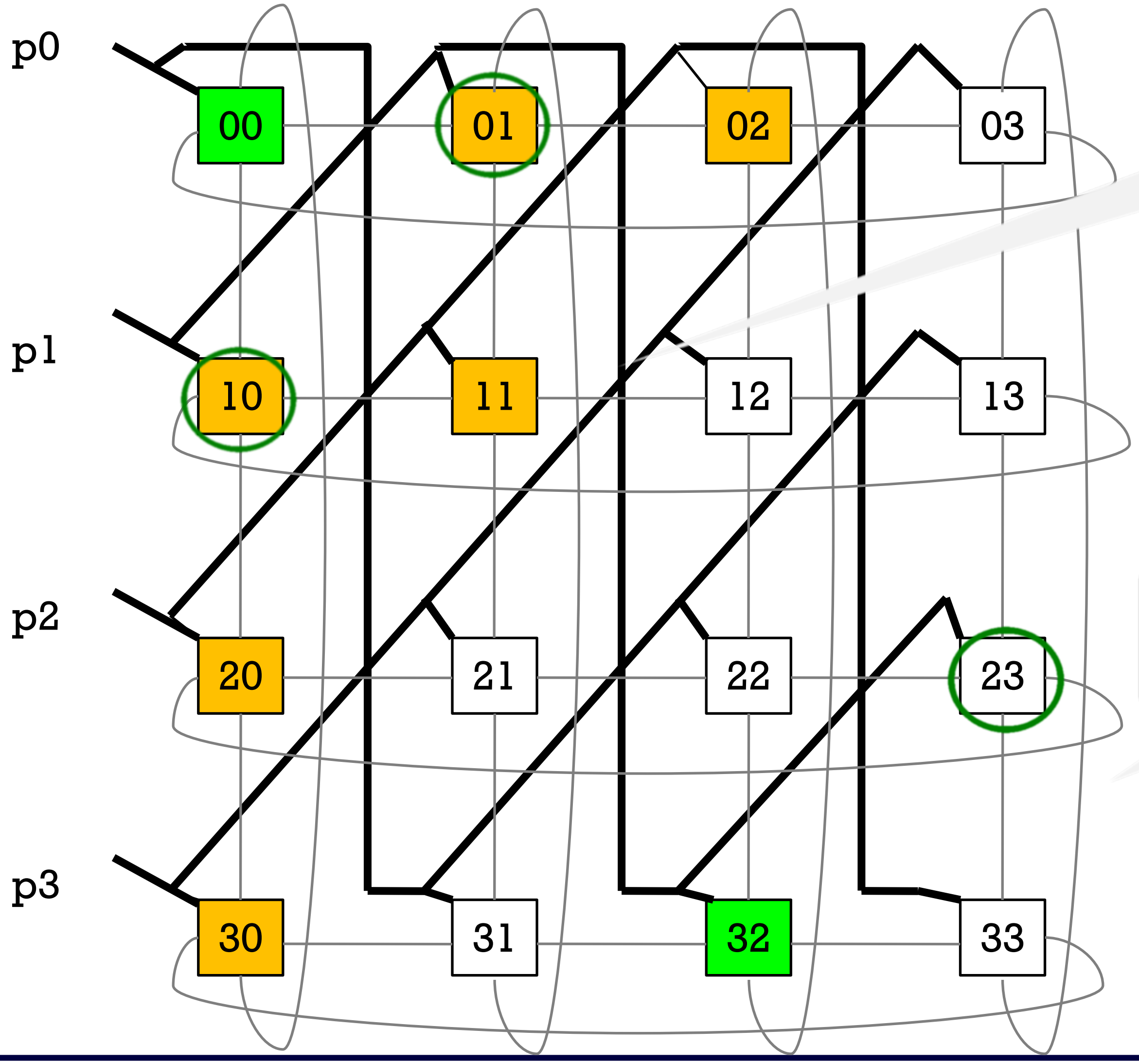


Entry [0,0] receives grant, consumes token

Remaining tokens pass down and righ

[3,2] receives 2 tokens and is granted

# EXAMPLE



[1,1] receives 2 tokens and granted

All wavefronts propagated

# TRADE-OFFS

- Pros of Wavefront?
  - Better matchings
  - Parallel distribution of multiple tokens

- Cons
  - Delay scales linearly
  - Requires custom tiled-design (synthesis from RTL not very efficient)

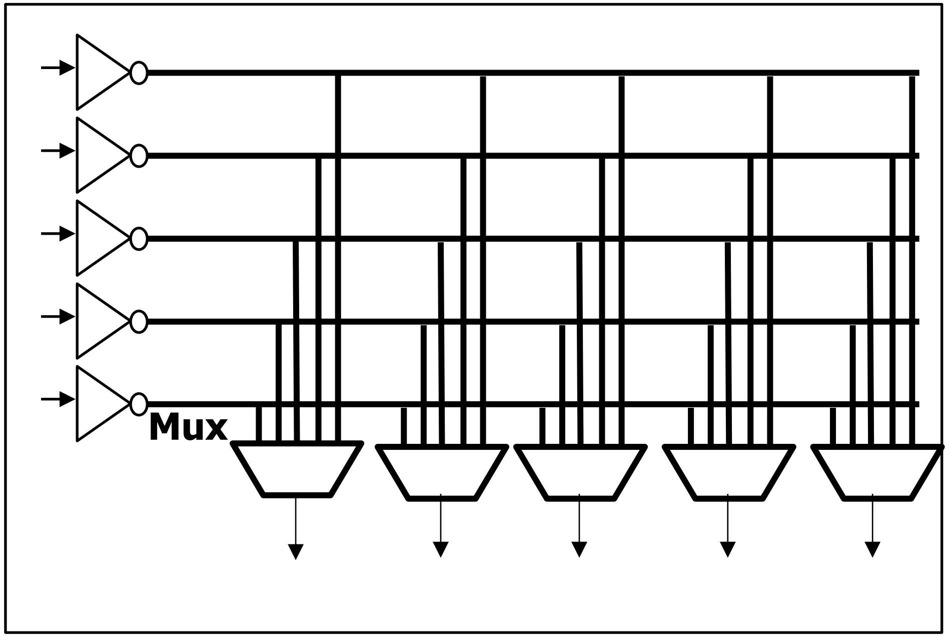
# VC ALLOCATION

- **N** input ports, **v** VCs per input port, **N x v** output VCs
- **Implementations**
  - Separable VC Allocator
  - 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

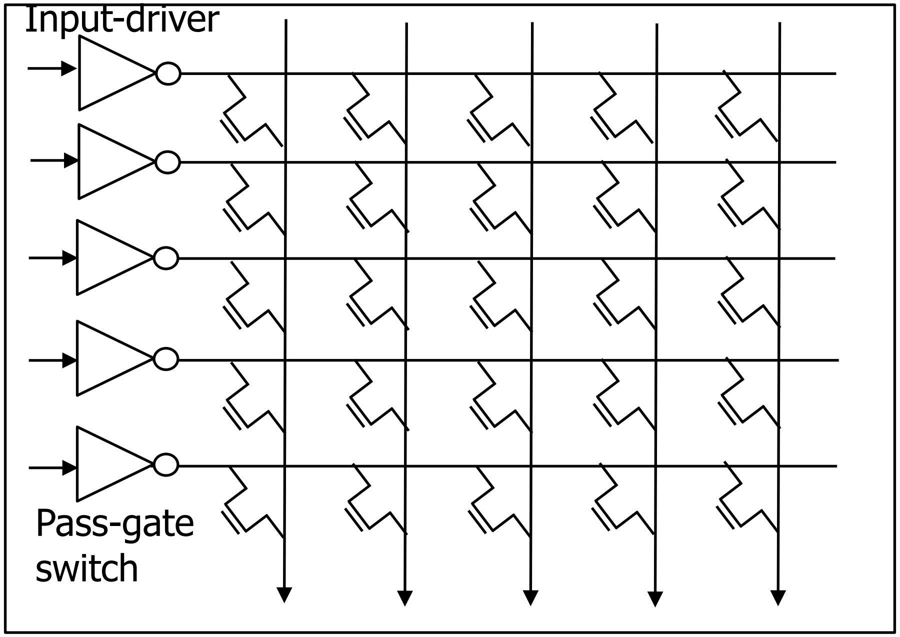
# ROUTER MICROARCHITECTURE

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

# SWITCH (CROSSBAR)



Mux-based Crossbar



Matrix Crossbar

+ Synthesizable from RTL
- Typically More Area and Power

+ Lower area and power
- Requires careful custom design

# CROSSBAR SCALABILITY

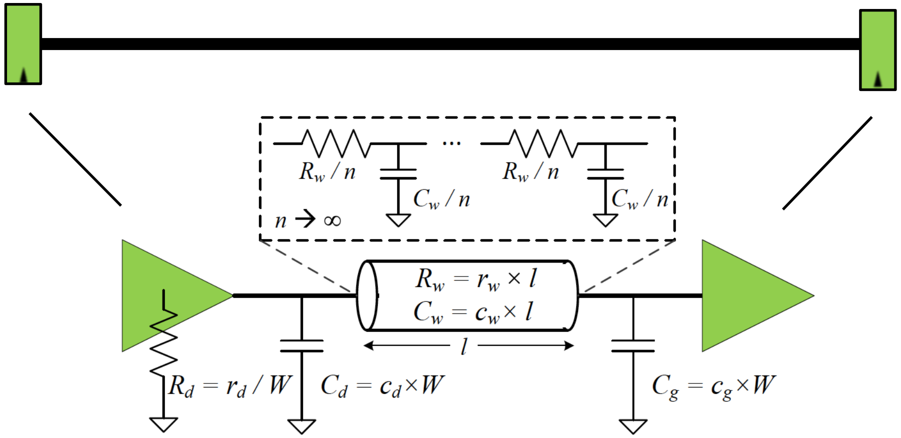
- Key Challenges
  - Area and power scale at $O((pw)^2)$
    - p: number of ports (function of topology)
    - w: port width in bits (determines phit/flit size and impacts packet energy and delay)
  - Arbitration in a large crossbar is challenging

- Crossbar Optimizations
  - Dimension-Slicing
    - 2x2 crossbar for X-dimension
      - Local, X
    - 3x3 crossbar for Y-dimension
      - Local, Y, X (i.e., turning)
  - Bit-Interleaving / Double-pumping (e.g., Intel SCC)
    - Send alternate bits on positive and negative phase of clock

# ROUTER MICROARCHITECTURE

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

# LINK



**Distributed RC**
Delay α $R_w C_w = r_w c_w L^2$
Energy α $C_w V^2$

*Dominated by wire capacitance which does not go down with technology scaling unlike transistor capacitance!*
➔ *Wires slower relative to logic every generation*

- **Reducing Delay**
  - Repeated Wires: Break long wire into N stages
    - Repeater = Inverter or Buffer (2 inverters)
    - Delay α N.$\text{Delay}_{\text{stage}}$ α N.$[r_w c_w (L/N)^2]$ α $r_w c_w L^2/N$
    - If too many stages, then delay of the repeaters can start to dominate
    - CAD tools automatically place repeaters
  - Alternate Technologies
    - RF, Photonics, Wireless

- **Reducing energy**
  - Circuit Techniques
    - Low-swing signaling (i.e., reduce V)
    - Coding techniques to reduce toggles
  - Alternate Technologies
    - Photonics (energy consumption same irrespective of distance)

# ROUTER MICROARCHITECTURE

- Components
  - Virtual Channel Buffers
  - Routing Logic
  - Allocation
    - Switch Allocation
    - VC Allocation
  - Crossbar Switch
  - Link

- Pipeline
  - 5-cycle router → 1-cycle router

# BASELINE ROUTER PIPELINE

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Head | BW | RC | VA | SA | BR | ST | LT | | | |
| Body 1 | | BW | ☒ | ☒ | SA | BR | ST | LT | | |
| Body 2 | | | BW | ☒ | ☒ | SA | BR | ST | LT | |
| Tail | | | | BW | ☒ | ☒ | SA | BR | ST | LT |

- **Per Packet**
  - RC, VA → done by Head flit
- **Per Flit**
  - BW, SA, BR, ST, LT

# WHY DOES ROUTER DELAY MATTER?

$$\boxed{T_N = (t_r + t_w) \times H + T_c + T_S}$$

- $T_N$: Network delay
- $t_r$: router pipeline delay
- $t_w$: wire delay per hop
- H: number of hops
- $T_c$: contention delay
- $T_S$: serialization delay (for multi-flit packets)

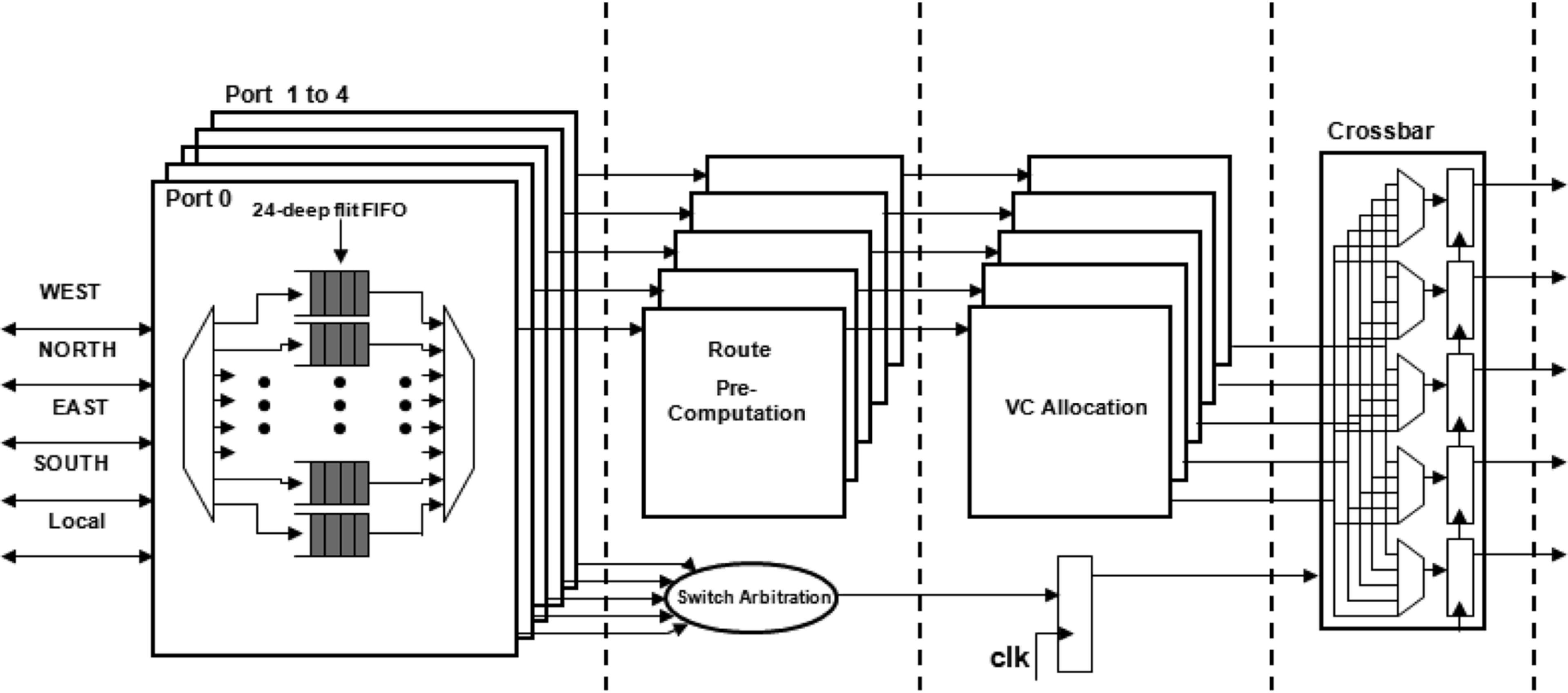
**Which of these is static?**

$t_r$ $t_w$ $T_s$

**Which of these is dynamic (traffic-dependent)?**

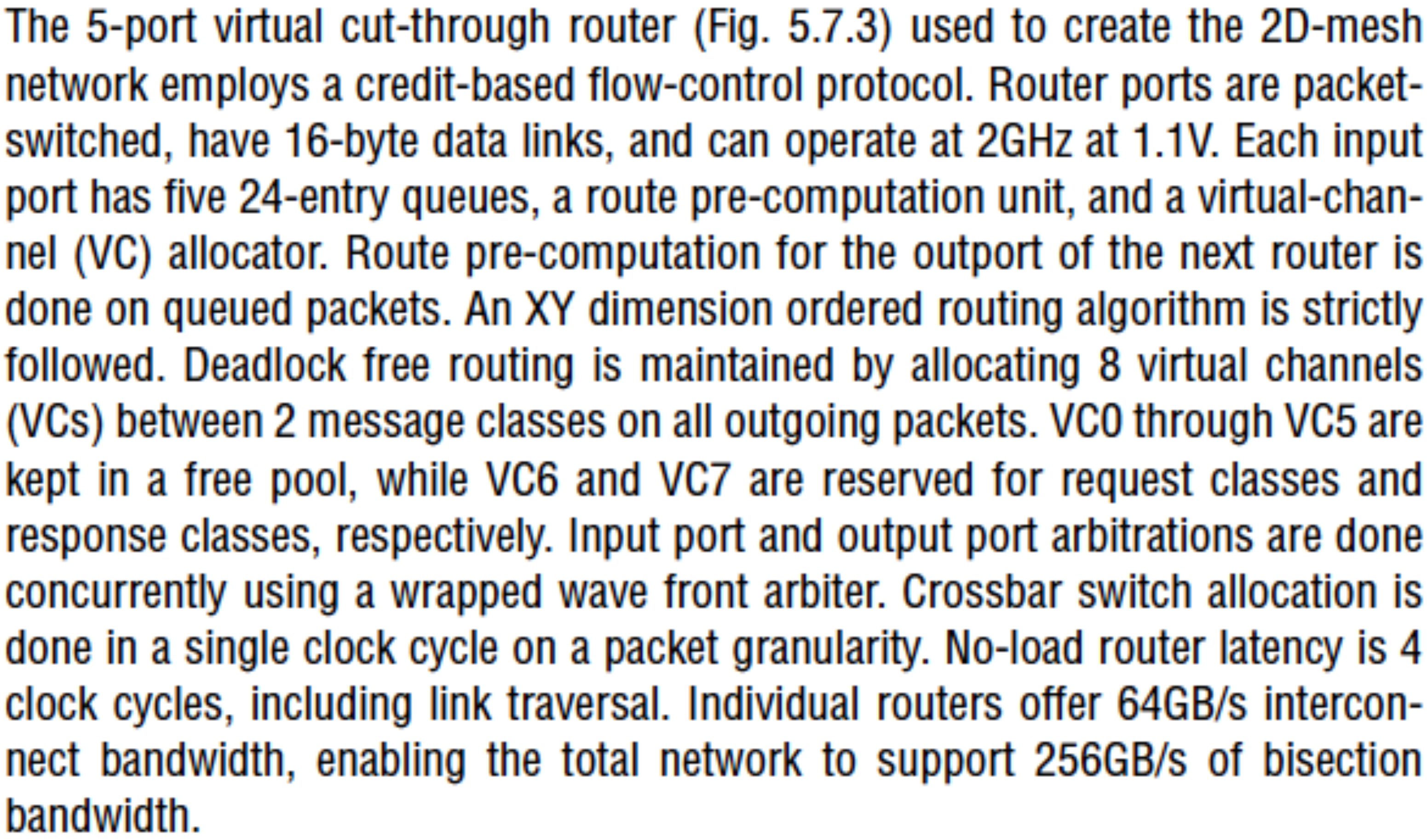
H $T_c$

# CASE STUDY: INTEL SCC (ISSCC 2010)



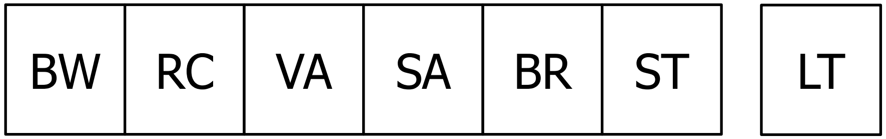
| STAGE 1 | STAGE 2 | STAGE 3 | STAGE 4 |
|---|---|---|---|
| Link Traversal + Buffer Write | Switch Allocation | Buffer Read + VC Allocation | Switch Traversal |

# CASE STUDY: INTEL SCC (ISSCC 2010)



The 5-port virtual cut-through router (Fig. 5.7.3) used to create the 2D-mesh network employs a credit-based flow-control protocol. Router ports are packet-switched, have 16-byte data links, and can operate at 2GHz at 1.1V. Each input port has five 24-entry queues, a route pre-computation unit, and a virtual-channel (VC) allocator. Route pre-computation for the outport of the next router is done on queued packets. An XY dimension ordered routing algorithm is strictly followed. Deadlock free routing is maintained by allocating 8 virtual channels (VCs) between 2 message classes on all outgoing packets. VC0 through VC5 are kept in a free pool, while VC6 and VC7 are reserved for request classes and response classes, respectively. Input port and output port arbitrations are done concurrently using a wrapped wave front arbiter. Crossbar switch allocation is done in a single clock cycle on a packet granularity. No-load router latency is 4 clock cycles, including link traversal. Individual routers offer 64GB/s interconnect bandwidth, enabling the total network to support 256GB/s of bisection bandwidth.

# COMMON PIPELINE OPTIMIZATIONS



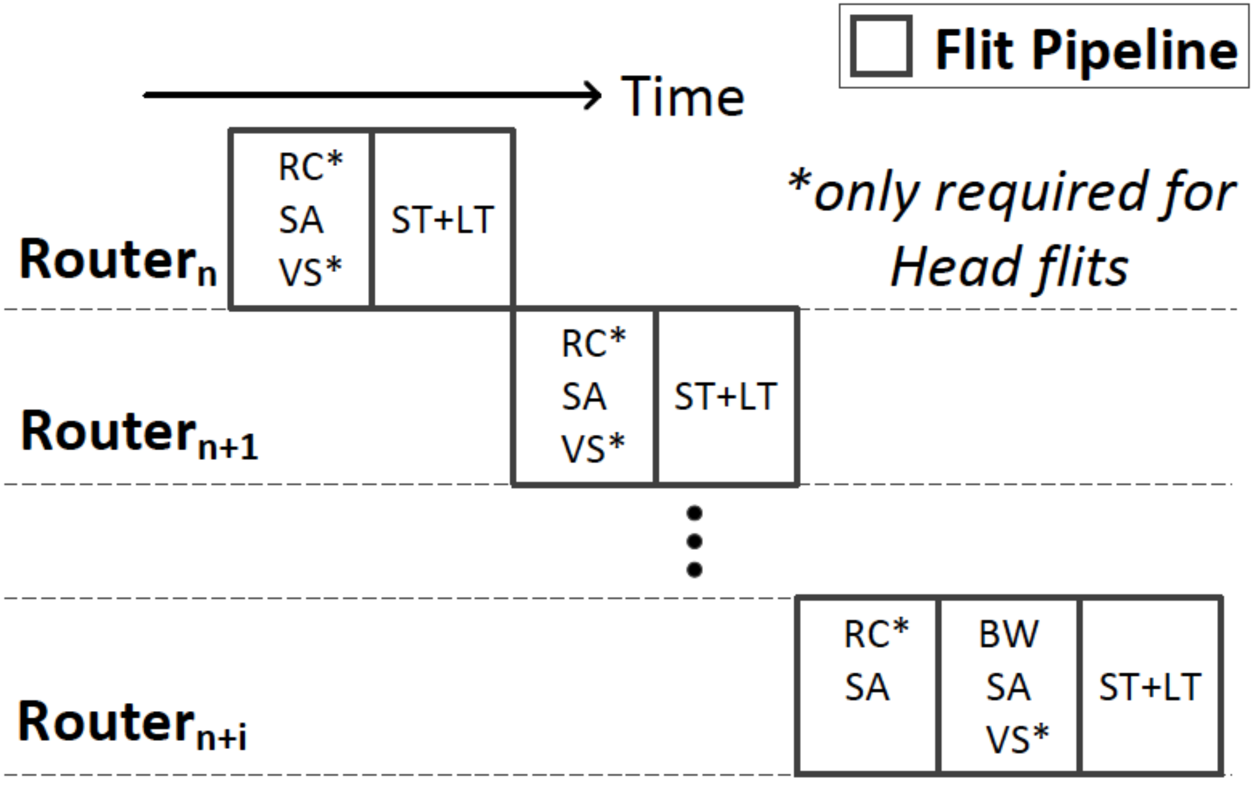
BW RC VA SA BR ST LT

- BW + RC in parallel
  - Lookahead Routing
- SA + VA in parallel
  - VC Select (switch output port winner selects VC from pool of free VCs)
  - Speculative VA (if VA takes long, speculatively allocate a VC while flit performs SA) (Peh and Dally, HPCA 2001)
    - If SA and VA both successful, go for ST
    - If SA or VA fails, retry next cycle
- BR + SA in parallel
  - The winner of Input Arbitration is read out and sent to the input of the crossbar speculatively
- Low-load Bypassing
  - When no flits in input buffer
    - Speculatively enter ST
    - On port conflict, speculation aborted

# Express Virtual Channels (ISCA 2007)

- Analogy – Express Trains and Local Trains
- Flits on Express VCs do not get buffered at intermediate routers
  - Send a "lookahead" to ask local flits to wait (i.e., kill switch allocation)

# MODERN PIPELINES



**1-cycle for arbitration (tr), 1-cycle for traversal (tw)**

*Used by Tilera's iMesh, Intel's Ring, NoC prototypes (Park et al., DAC 2012)*

# IS THAT THE BEST WE CAN DO?

$$\boxed{T_N = (\underset{1}{t_r} + t_w) \times \boxed{H} + T_c + T_S}$$

$t_w = 1$

**fundamental limitation?**

Latency α Hops

*Can we remove the dependence of latency on hops?*

SMART NoC [T. Krishna et al, HPCA 2013].
*Very ripe for project ideas.*

# ALTERNATE ROUTER MICROARCHITECTURES

- Output Queues
  - "Virtual" Output Queues (== Virtual Channels)
- Centralized Buffers
- Rotary Router (in paper presentations)