Queue and Flow Control Architectures for Interconnection Switches

1. Basic Concepts and Queueing Architectures
2. High-Throughput Multi-Queue Memories
3. Crossbars, Scheduling, & Combination Queueing
4. Flow and Congestion Control in Switching Fabrics

Manolis Katevenis

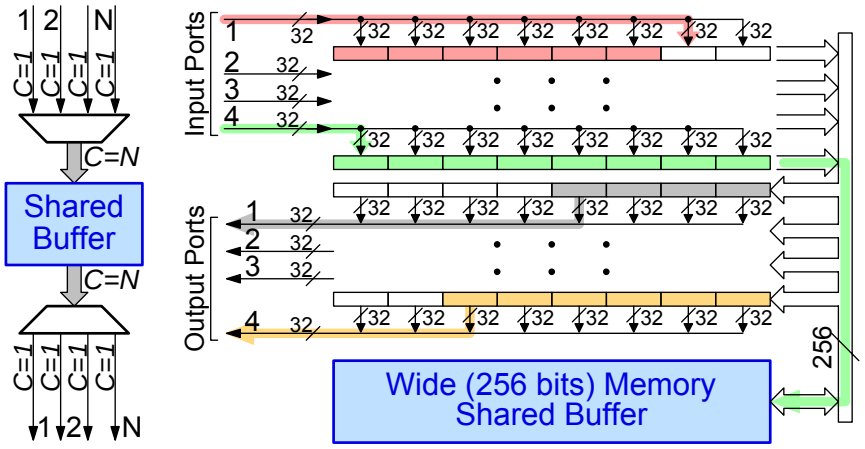
FORTH and Univ. of Crete, Greece

http://archvlsi.ics.forth.gr/~kateveni/534

2. High-Throughput Multi-Queue Memories

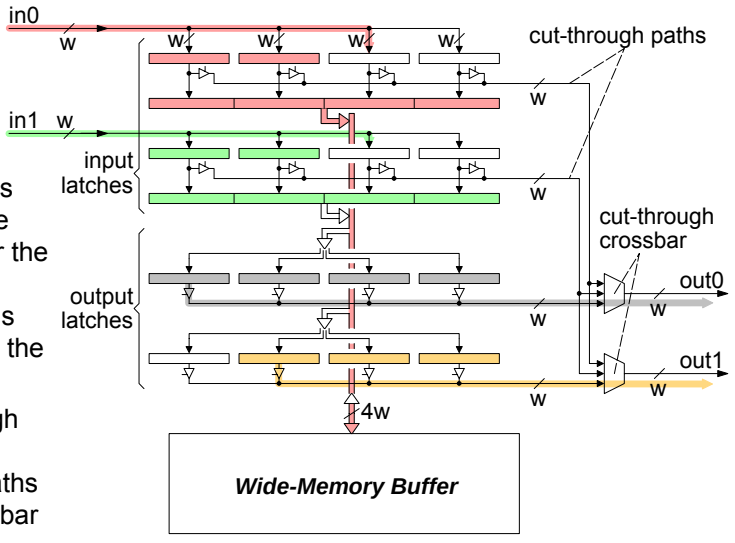
Table of Contents:

2.1 Shared Buffer Switch using a Wide Memory



- Example: 4×4 switch @ 10 Gbps/link = 320 MHz × 32 bits
- Memory serves each I/O register for one clock cycle out of every eight
- Note hidden input and output crossbars; note required cell time alignment

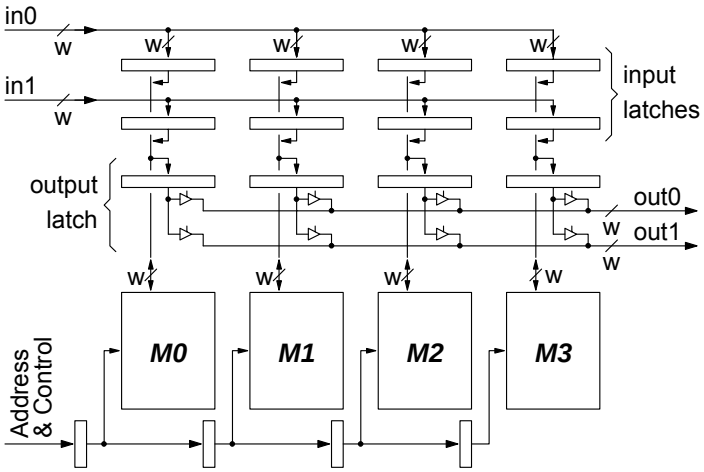
2.1 *Wide Memory:* Double Buffering Needed



- Input latches need double buffering for the case when multiple cells complete at the same time
- If cut-through is desired, separate paths and a crossbar are needed

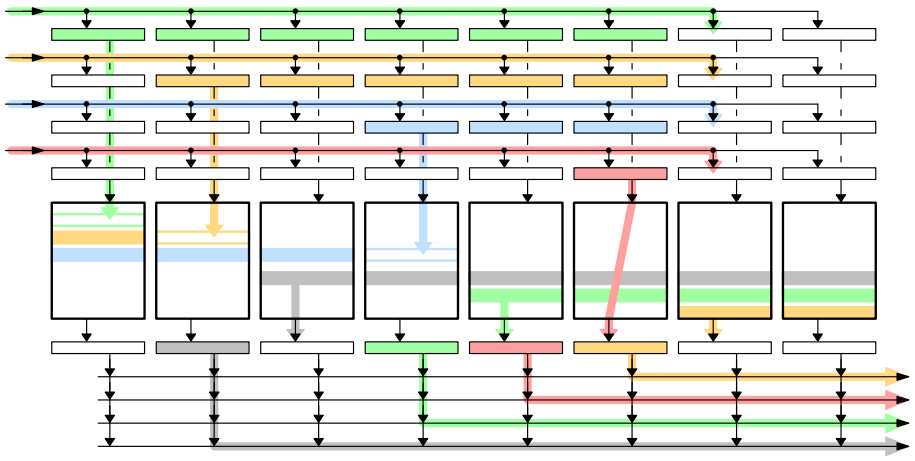
2.1 Optimized Implementation: Pipelined Memory

- Monolithic wide memory replaced by multiple banks
- concurrent rd/wr access to entire width replaced by a "wave" of same-address accesses moving from left to right



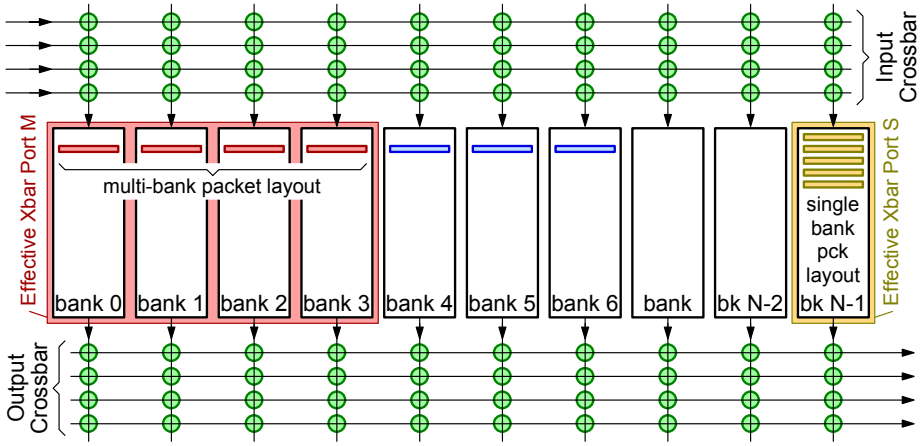
- Benefit 1: no double buffering needed at inputs, single latch at outputs
- Benefit 2: cut-through occurs automatically – no paths, no control needed

Pipelined Memory Operation Illustrated



Ref: Katevenis, Vatsolaki, Efthymiou: "Pipelined Memory Shared Buffer for VLSI Switches", ACM SIGCOMM Conf. 1995, http://archvlsi.ics.forth.gr/sw_arch/pipeMem.html

2.1 Generalization: Interleaved Memory Banks



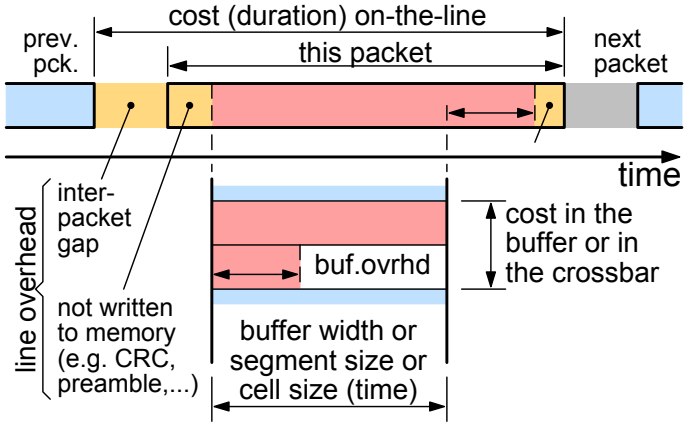
- Can we scale arbitrarily wide-memory throughput by increasing its width?
- Not past individual packet width – then it becomes something else; see ↓

Interleaved Mem. Sh.Buf. – PPS, Birkhoff-vonNeumann

- Wide memory width versus packet size
  - Wide (or pipelined) memory owes its simplicity to the controller generating a single address per cycle for all "banks" (entire width)
  - To have multiple packets per line (e.g. red & blue in figure): need independent address controllers per bank (per effective xbar port)
  - Note: infeasible to pack together into the same line packets that *both* arrive *and* depart consecutively in time
  - Scheduling interleaved memory accesses against bank conflicts is equivalent to crossbar scheduling under input queueing…
- Scalable Shared-Buffer Switches through Interleaved Banks:
  - Complexity: maintain distributed queues, forward cells in-order
  - The Parallel Packet Switch (PPS) – Khotimsky, Krishnan: IEEE Int. Conf. Commun. (ICC) 2001; Iyer, McKeown: IEEE/ACM ToN, Apr. 2003
  - Birkhoff-vonNeumann – C. Chang, W. Chen, H. Huang: Infocom 2000

2.2 Variable Size Packet Segmentation Overhead

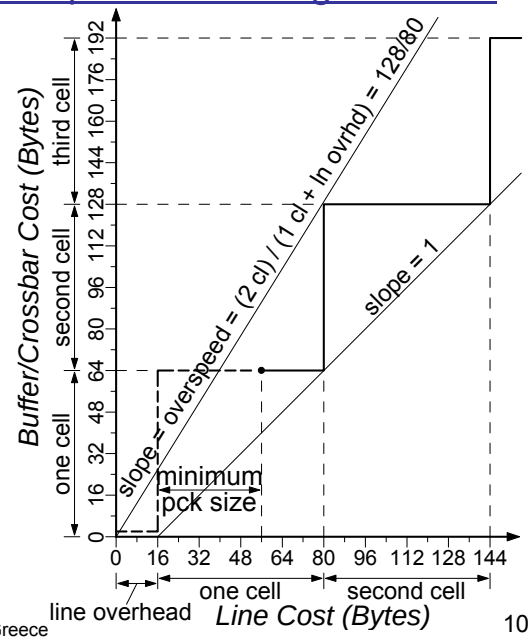
- most buffer memories, queueing structures, & crossbars operate on fixed-size "segments", "blocks", or "cells"
- most appl'ns use variable size packets



- Packets segmented into blocks/cells upon entry from line to switch
- Throughput of buffer memories & crossbars, measured in terms of cells, must be higher than line thruput, to compensate for segment'n overhead

Overspeed req'd to compensate for Segmentation

- Example:
  - min. pck. size = 40 B
  - segment (cell) = 64 B
  - line overhead = 16 B
- Rules of thumb:
  - For line overhead = 0 ⇒ overspeed = often **2.0**
  - … but check min.pck.sz.: if minimum packet is too small (smaller than about half a cell), then higher overspeed is required
  - for line overhead > 0 ⇒ overspeed < 2.0, but check minimum packets
  - for very large line ovrhd ⇒ check larger pck sizes

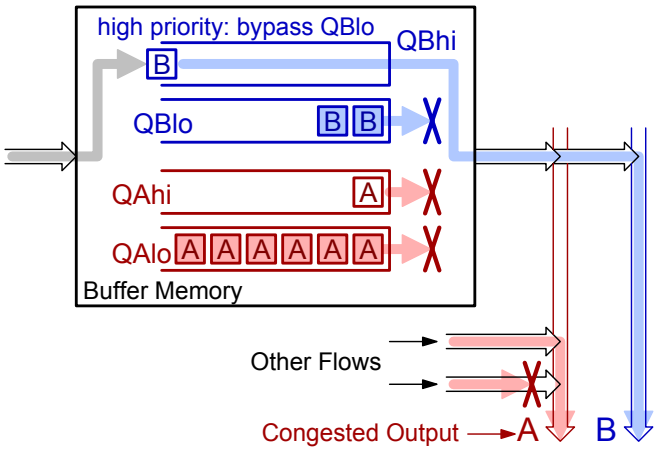


2.3 Multiple Queues within a Buffer Memory

Separate Destinations & Priorities ⇒ Multiple Queues

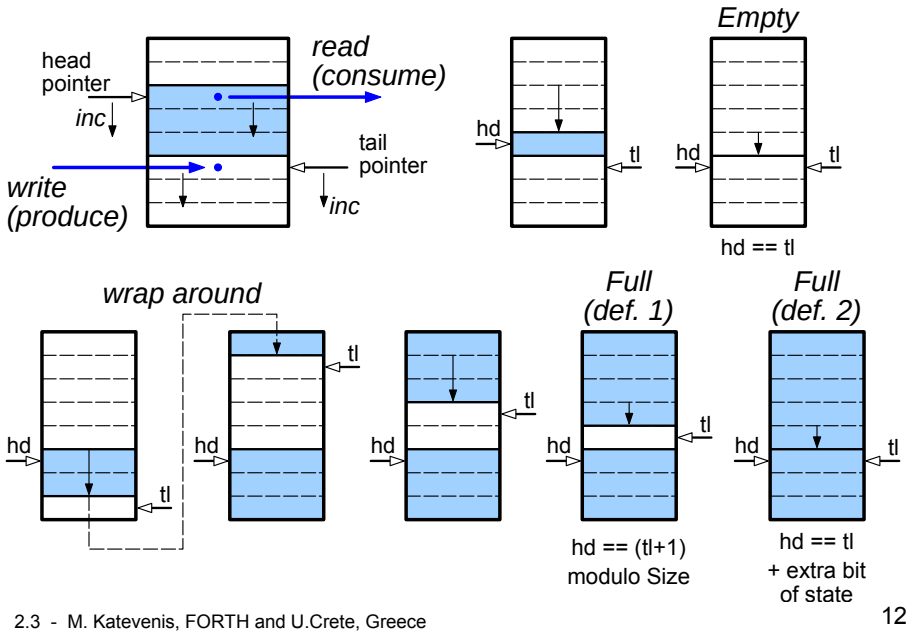
- Switch controller must have access to any packet that is candidate to depart next

⇒ Packets that are allowed to bypass others cannot reside in the same FIFO structure



- Controller needs separate *per-destination* and *per-priority* queue (FIFO) data structures to keep track of packets

Reminder: Circular Array Implem. of FIFO Queue

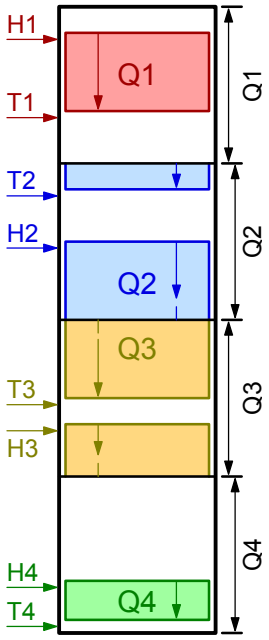


2.3 Multiple Queues – 1 of 2
Statically Partitioned Space

- Multiple queues within a same SRAM block
- Each queue: circular array implementation
- Control overhead: two pointer words per queue (head, tail), incrementor, comparator
- Queue space bounds (partitions) can be hardwired, or off-line configurable (when queues are empty); in the latter case, also need bounds pointers.

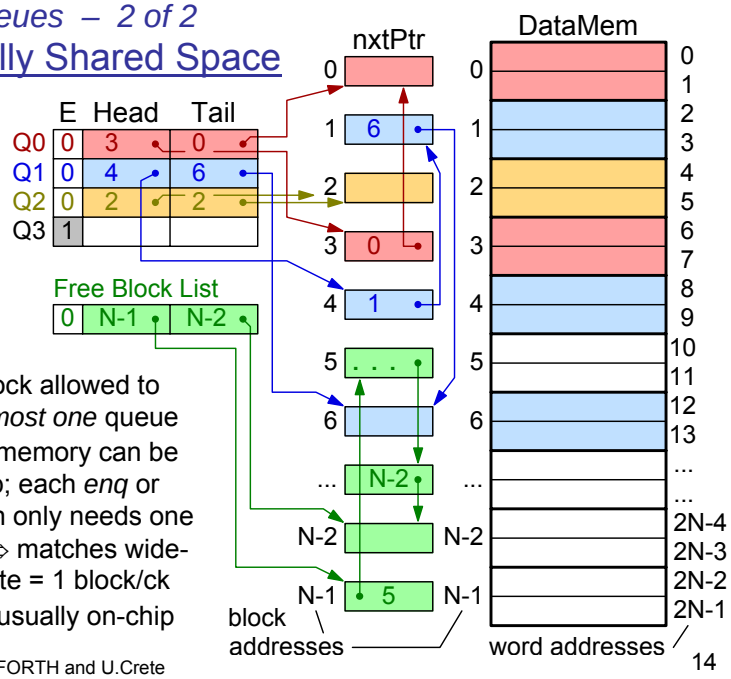
\+ Advantage: *simplicity*.

\- Disadvantage: *partitioned* memory space leads to *underutilization* – one queue may overflow while lots of empty space exists in other memory space partitions.

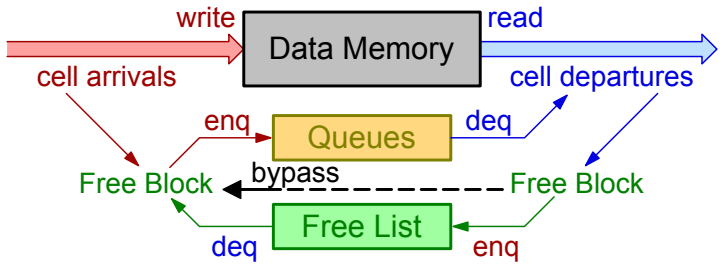


Multiple Queues – 2 of 2
Dynamically Shared Space

- Linked List implem. of queues
- Pointers in separate memories: accessed in parallel
- Each data block allowed to belong in *at most one* queue
- Next-pointer memory can be large, off-chip; each *enq* or *deq* operation only needs one access to it ⇒ matches wide-mem. data rate = 1 block/ck
- *Empty/Hd/Tl* usually on-chip

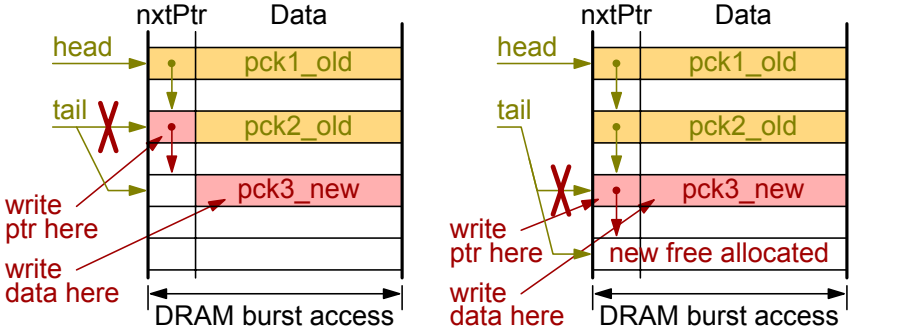


Data vs. Pointer Access Rate – Free List Bypass



- Data memory throughput = 2 cells/cell-time (1 write + 1 read)
  ⇒ data memory access rate = 2 addresses/cell-time
- Both Queue & Free-List operations touch the Next-Pointers, once per op
  ⇒ naïve implementation would require 4 addresses/cell-time to *nxtPtr*
- *Free List Bypass:* put incoming cell into just freed block of departing cell
  ⇒ next -pointer memory access rate = 2 addresses/cell-time
- When no arrival or no departure, other side can use full 2 acc/cl-time rate
- Multicast: departure not always frees the block ⇒ use *Free Block Cache*

nxtPtr in DRAM – Free Block Preallocation



Conventional Enqueue *Enq. w. Free-Block Preallocation*

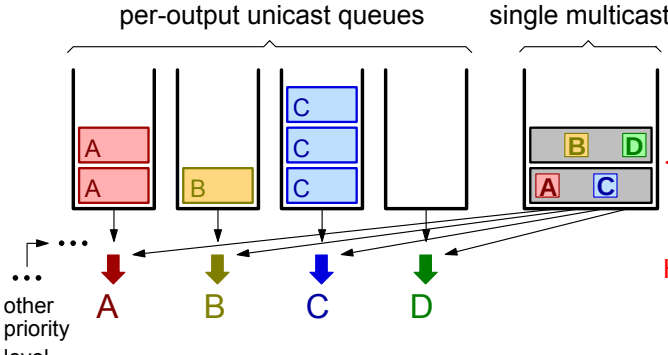
- To economize on *nxtPtr* memory, place these pointers inside data DRAM
  ⇒ conventional *enq* costs twice the number of DRAM row *activate*'s
- Preallocate one free block per queue, at tail, to remedy this
- Reference: Nikologiannis, Katevenis: "Efficient per-flow queueing in DRAM at OC-192 line rate using out-of-order execution…", IEEE Int. Conf. Commun. (ICC) 2001.

2.4 Queueing for Multicast Traffic

- Multicast traffic is expected to become very important in the future
  - but so has it been for many years in the past…
- Supporting multicast traffic usually increases complexity and cost
- Queueing for Multicast Traffic:
  - Each segment (block) allowed in only one queue ⇒ HOL blocking
  - Each segment allowed in multiple queues ⇒ need many nxtPtr's
  - Enqueue throughput and nxtPtr space: static vs. dynamic sharing
- References:
  - F. Chiussi, Y. Xia, V. Kumar: "Performance of Shared-Memory Switches under Multicast Bursty Traffic", IEEE Jour. Sel. Areas in Communications (JSAC), vol. 15, no. 3, April 1997, pp. 473-487.
  - D. Stiliadis: "Efficient Multicast Algorithms for High-Speed Routers", Proc. IEEE Workshop on High Performance Switching and Routing (HPSR 2003), Torino, Italy, June 2003, pp. 117-122.

Same or Different Queues with Unicast Traffic?

Case 1: Each segment is only allowed to belong to a single queue

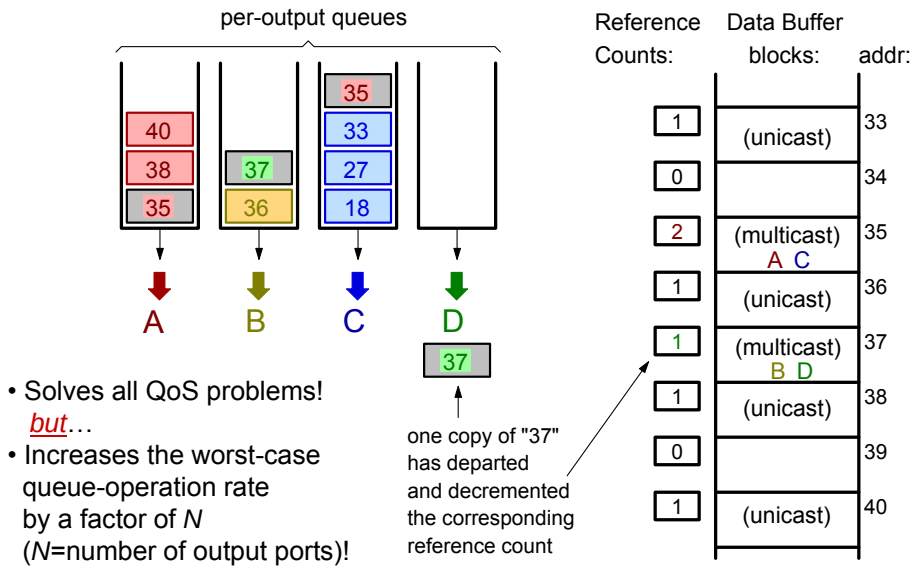


- Impractical to have per-output multicast queues: would need $O(2^n)$ queues

Head-of-line Blocking!

- We may have a different set of these queues (including multicast) per priority level, but it may still happen that traffic destined to outputs A and C currently exists at priority levels higher than "our" cell A-C while all queues destined to B and D at priority levels above "our cell" B-D are empty.

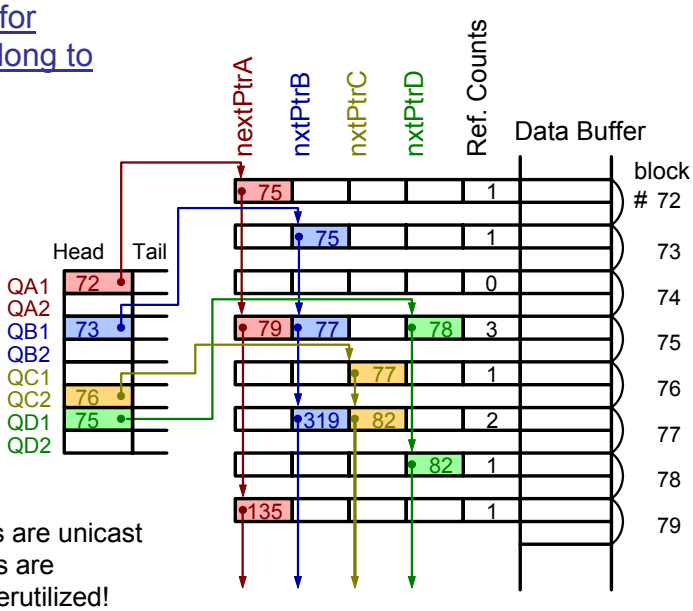
Case 2: Each segment is allowed to belong to multiple queues



- Solves all QoS problems!
  but…
- Increases the worst-case queue-operation rate by a factor of *N* (*N*=number of output ports)!

Data Structures for a segment to belong to up to *N* queues:

Case 2A: N nxtPtr's per memory block

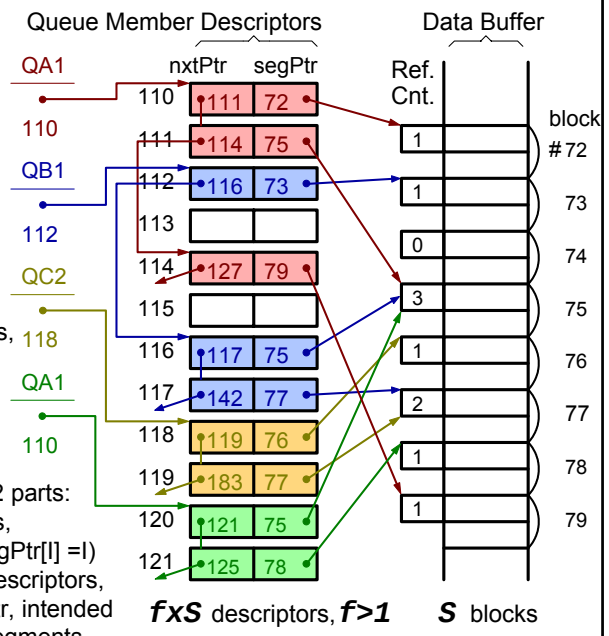


- Most segments are unicast
  → next pointers are grossly underutilized!

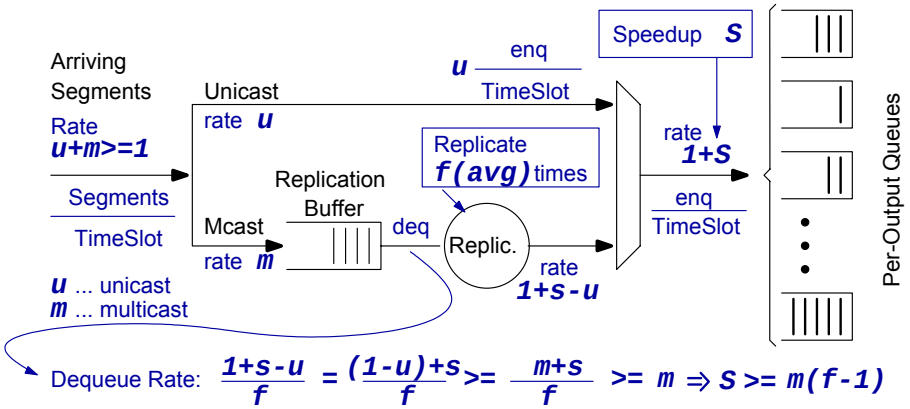
Case 2B: Decouple Linked List Nodes From Data Buffer Addresses

- twice the cost per nxtPtr (need a segPtr as well now)
  but …
- Much fewer than NxS descriptors (based on avg ratio of unicast-to-multicast segments, and avg fan-out of multicast segments, e.g. f =2)

Optimization:
Partition the address space of queue member descriptors into 2 parts:
- 0 to S-1: unicast-only segments, no segPtr needed (segPtr[I] =I)
- S to fS-1: full queue member descriptors, with nxtPtr and segPtr, intended to use by multicast segments



Enqueue operation rate for multicast segments into multiple per output queues



- References:
  - –F. Chiussi, Y. Xia, V. Kumar: IEEE JSAC, April 1997, pp. 473-487.
  - –D. Stiliadis: IEEE HPSR 2003, June 2003, pp. 117-122.