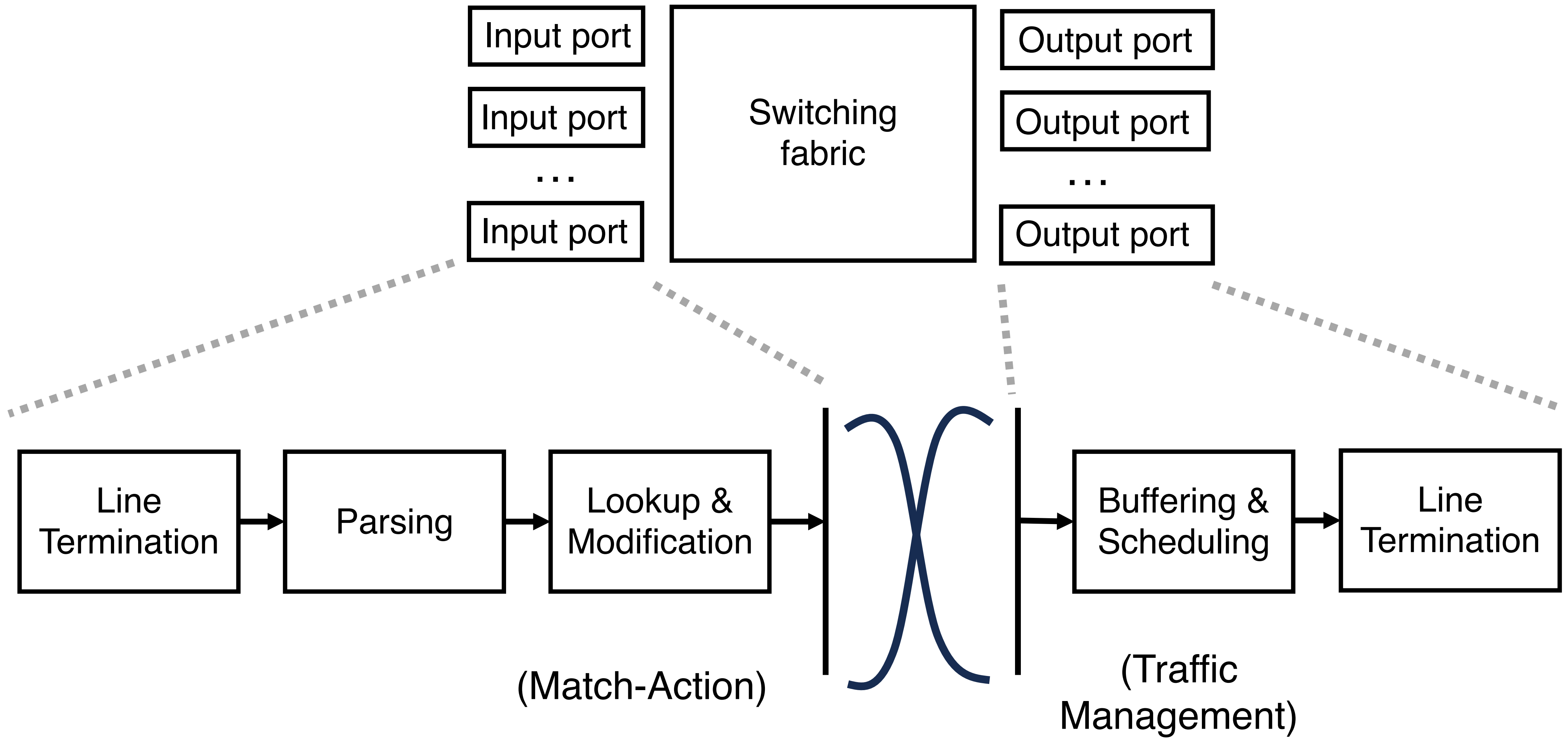
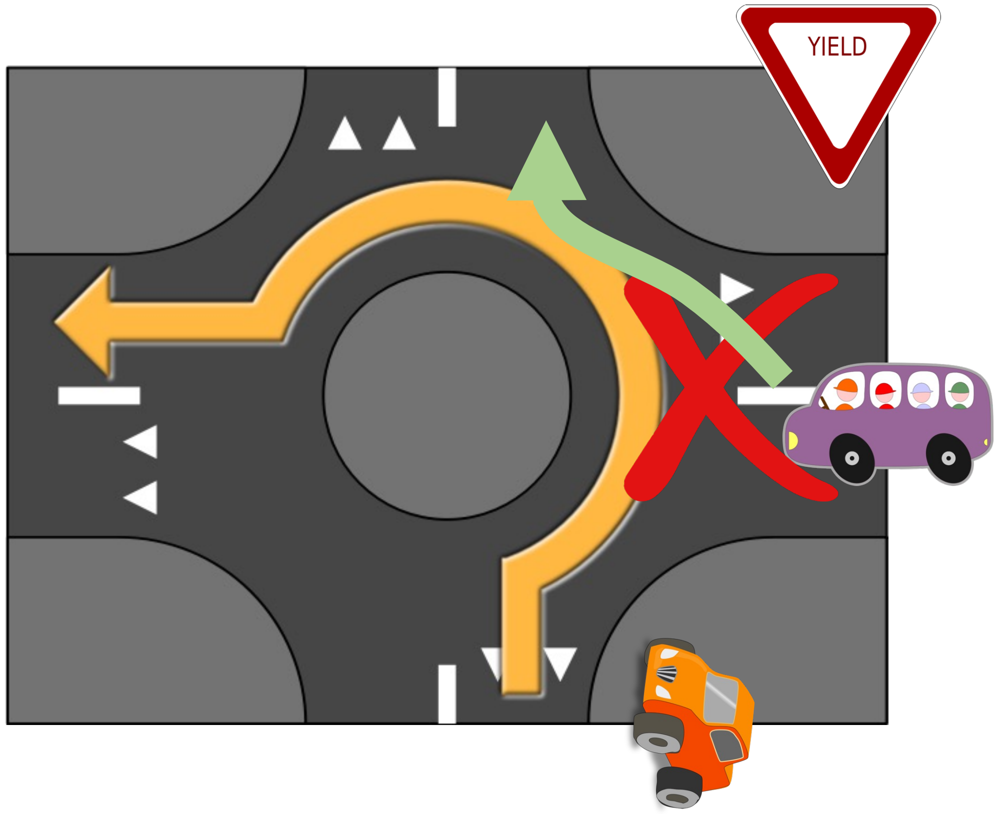
Network

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Hardware Router overview

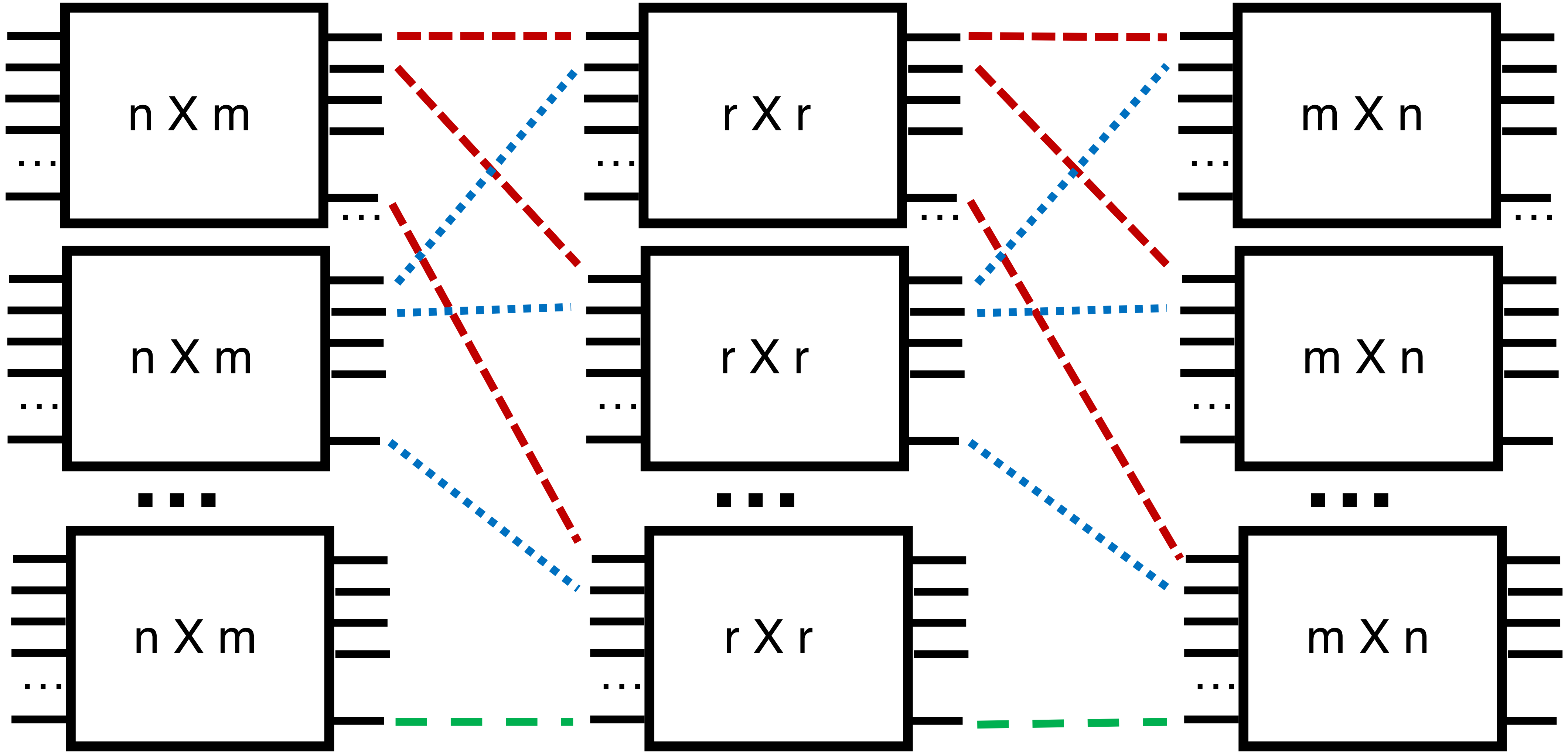


Nonblocking designs are nontrivial



Two aspects: topology and routing

3-stage Clos network (r*n X r*n ports)

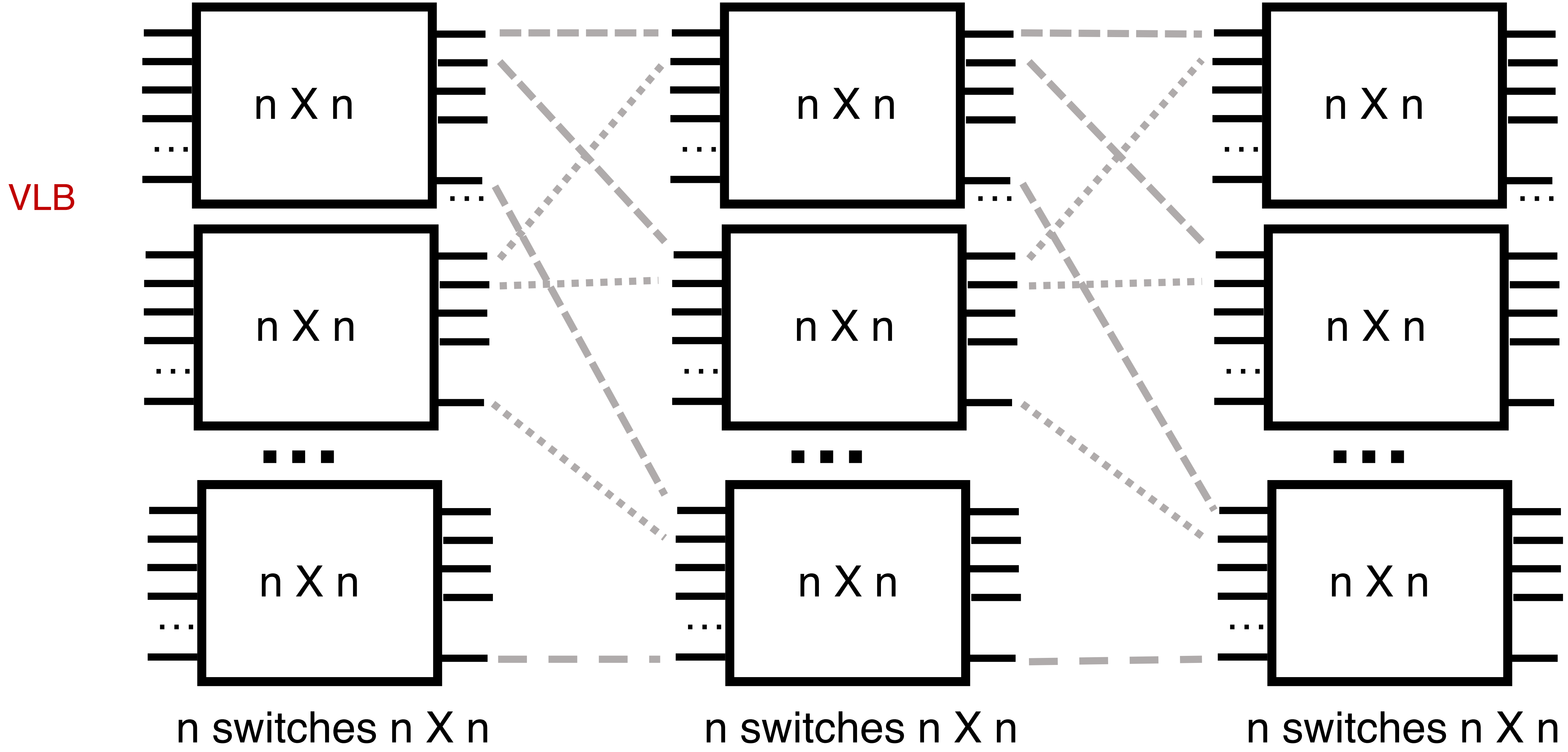


r switches n X m

m switches r X r

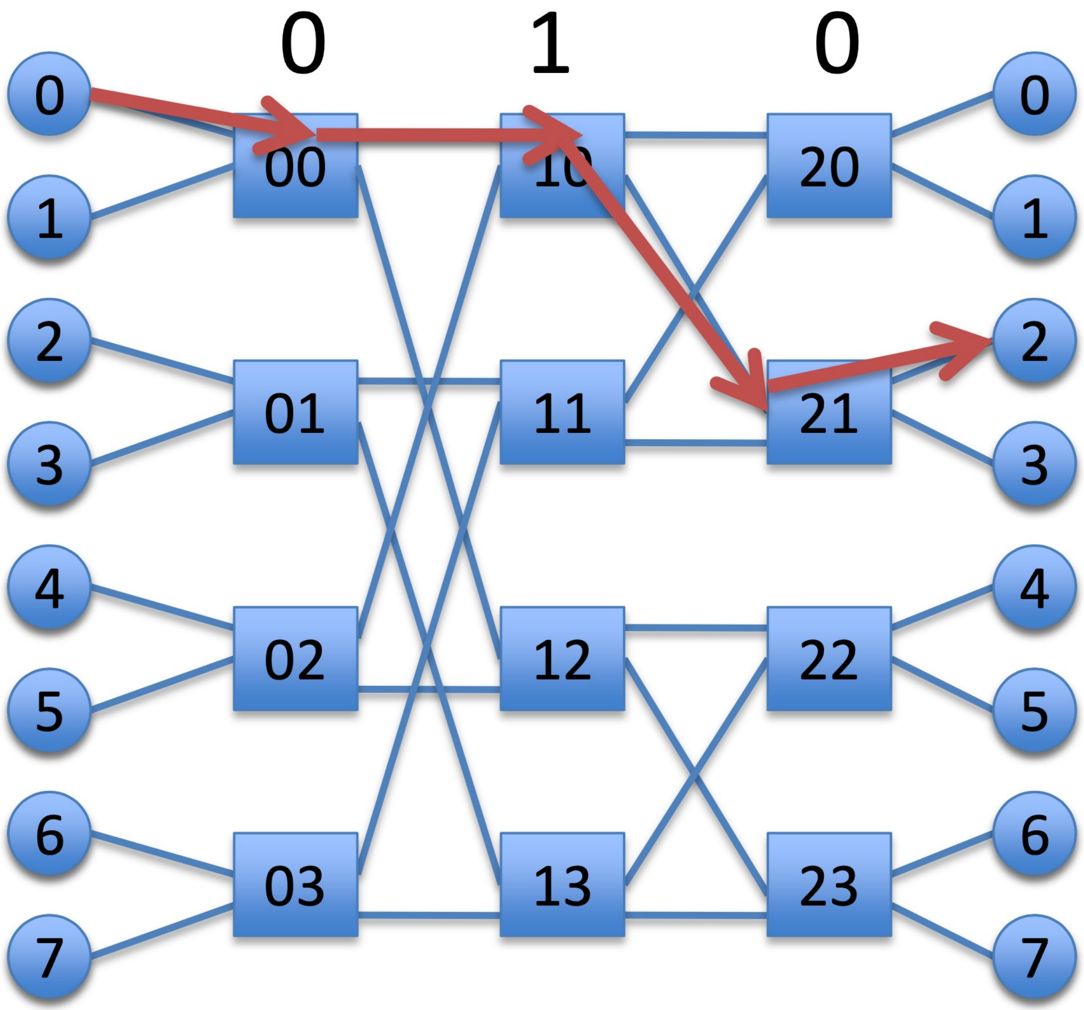
r switches m X n

Rearrangeably nonblocking Clos built with identical switches: $n^2 X n^2$ using 3n nXn



Increasing # ports: Butterfly Networks

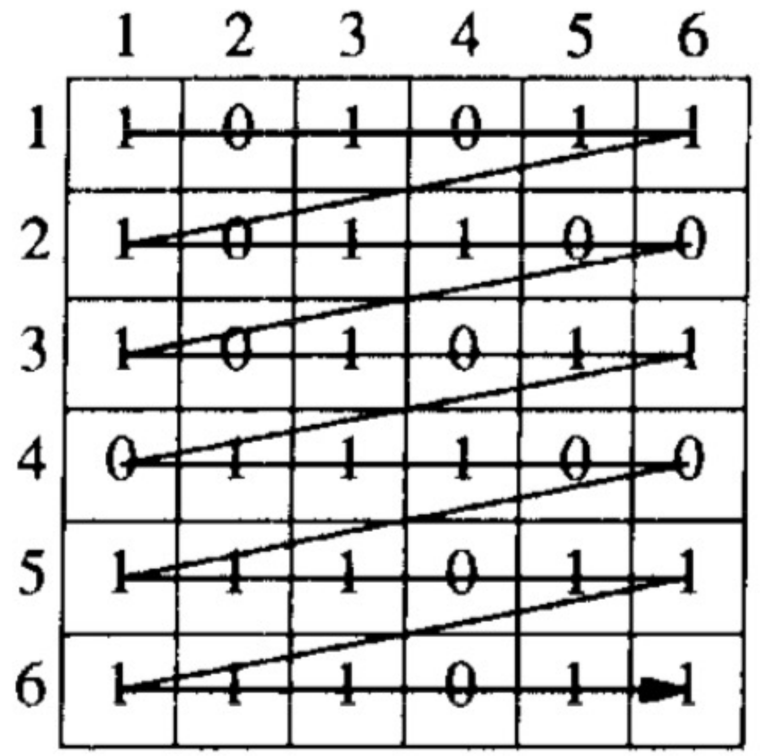
- Can we reduce internal # ports for a given external # ports?
- K-ary L-butterfly:
- Use L*K KxK port switches to build $K^L X K^L$ port switch
- Figure: K = 2, L = 3
- Produce $n^3 X n^3$ switch from 3n nXn switches
  - Clos: $n^2 X n^2$
- Routing is deterministic
- Tradeoff: more blocking

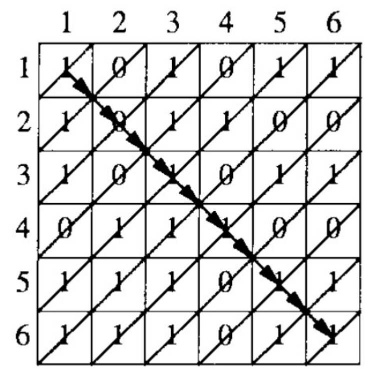


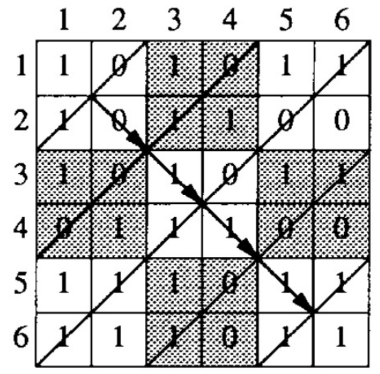
https://ece757.ece.wisc.edu/lect09-interconnects-2-topology.pdf

(4) MGR: Crossbars & Matching

- MGR uses a nonblocking crossbar across 15 ports
- Strategies to match incoming demands & output ports quickly
  - Greedy (simple), wavefront, group
  - Try to address fairness across ports





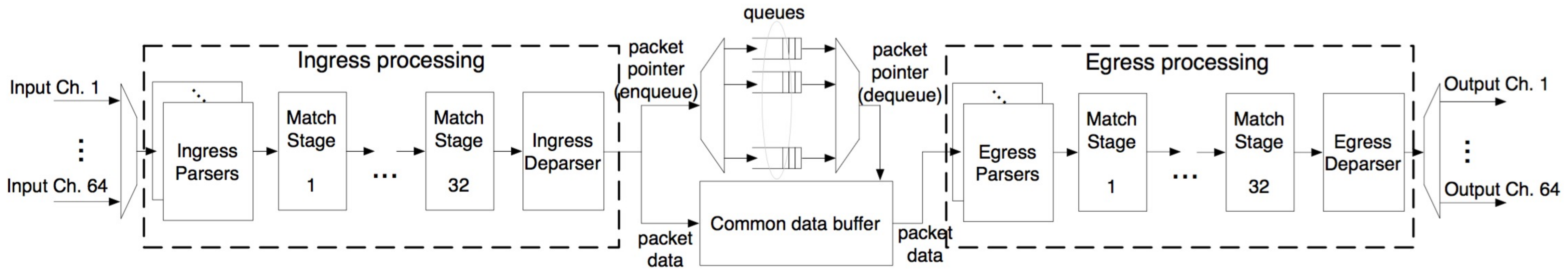


(4) RMT: Memory switching fabric

- RMT uses memory as the fabric to hold packet headers and payloads between any two interfaces
- Key challenge: simultaneous access to memory (N memory ports)
- In the late 90s and early 2000s, there was considerable research on building high-speed packet buffers
- Today: shared memory switches & routers (shared ➔ across ports)
  - Fast memory can be clocked at 1 GHz
- Fundamental tradeoff: faster memories are not very dense
  - Can’t make the memory too large; can’t hold too any packets
- Workaround: exploit memory access patterns: e.g., each queue is FIFO
- Traffic manager implements scheduling & buffer management

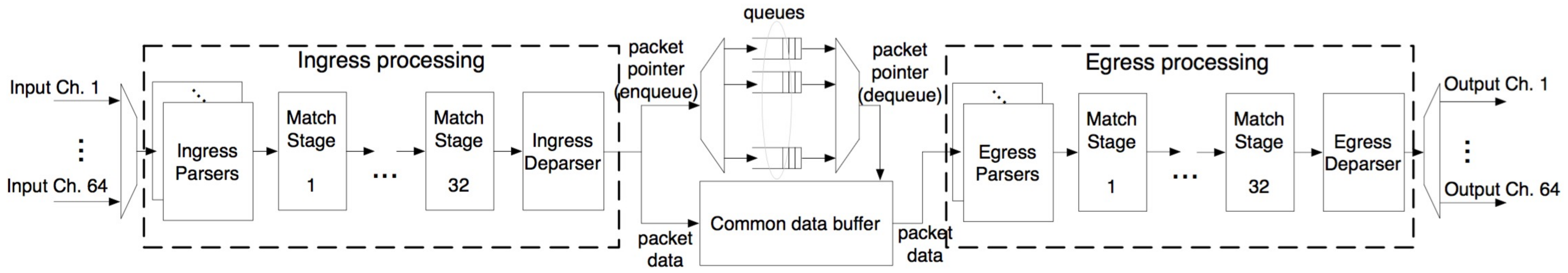
(5) Traffic Manager

- Where should the packets not currently serviced wait?
- Two designs: Input-queued vs. output-queued
- Output queueing avoids HOL blocking exhibited by input queueing.
    - Suppose port 1 wants to send to both 2 and 3 but port 2 busy
    - Packets from p1 towards p3 need not be delayed



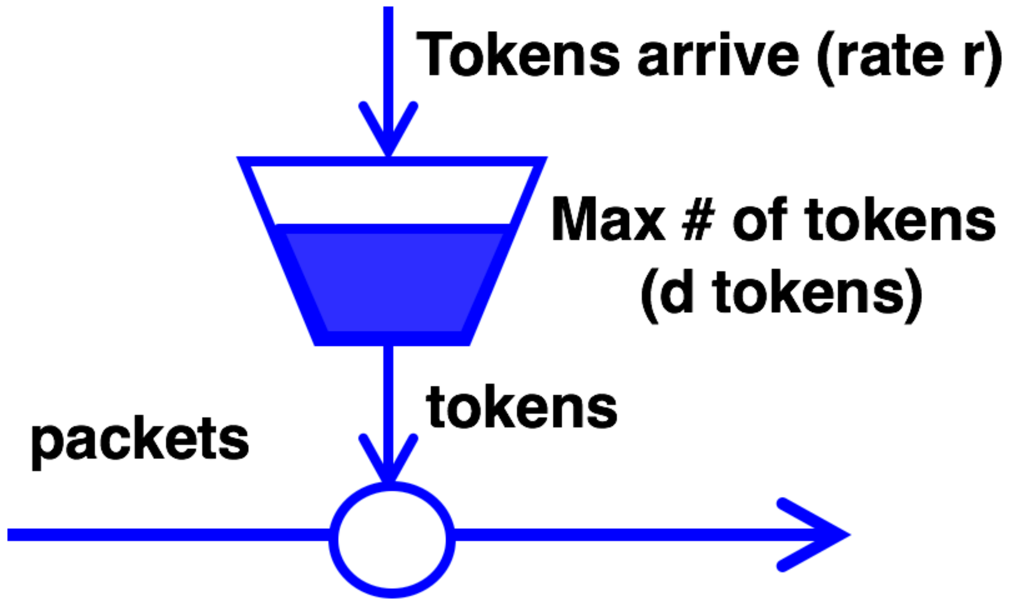
(5) Traffic Manager

- Queueing represents output port contention
- A single output port can be represented by multiple queues
    - e.g., to implement weighted fair queueing
- Each queue is just a linked list in the shared memory
    - Maximum flexibility in queue sizes, but pointer overhead
    - Separate memory to maintain per-queue heads and tails



(5) Traffic Manager: Scheduling policies

- How to dequeue packets in output port buffer? packet scheduling algorithms
- Fair queueing across ports or flows
- Strict prioritization of some ports over others
- Rate limiting per port
- Possible to make it flexible: PIFOs



(5) Traffic Manager: Buffer Management

- Q: how to enqueue packets into buffer?
  - If buffer is full, which packet should be dropped?

- Typical buffer management: Tail-drop
- Want fairness: if queue 1 has too many buffered pkts, don't tail-drop q2
  - Share memory by partitioning (carving memory out) across queues
- Want efficiency: if q1 has no pkts, q2 should be able to use (nearly) all buffer memory
- One possibility: static thresholds for buffer occupancy per port
  - Can be made fair or efficient but not both

(5) Demand-aware buffer management

- DT: “Dynamic Queue Length Thresholds for Shared-Memory Packet Switches”, Choudhury and Hahne
- Compute a critical (dynamic) queue length threshold T

$$T(t) = \alpha \cdot (B - Q(t)) = \alpha \cdot \left(B - \sum_i Q^i(t) \right)$$

- Port blocked from adding packets if

$$Q^i(t) \geq T(t)$$

(6) Egress line termination

- Combine headers with payload for transmission
  - Must incorporate effect of header modifications
  - Also called deparsing or serialization

- Multicast: egress-specific packet processing
  - Ex: different source MAC address for each output port

- Multicast makes almost everything inside the switch (interconnect, lookups, queueing) more complex

Note: three kinds of router hardware data plane programmability

- Packet header formats, i.e., the packet parser
  - Example: Go from IPv4 -> IPv6
  - Custom packet format to carry financial info at high speed on a point-to-point link

- Table formats, actions, sizes, i.e., the match-action tables
  - Change which fields in the packet can be processed by a table
  - Control the table sizes, i.e., # entries, and hence the memory resource footprint according to use case.

Note: three kinds of router hardware data plane programmability

- Packet scheduling, i.e., the traffic manager
  - Flexible classification of packets
  - Flexible assignment of ordering and timing of when packets are transmitted from an outgoing link

… which is distinct from control plane programmability

- The control plane must compute the packet-processing rules put into the memory on the router ASIC
  - Example: packet with IPv4 destination 10.0.0.1 must go out of port 4

- Data plane programmability refers to the flexibility in the allowed set of packet headers, tables, and actions themselves, not the actual rules.
  - Example: There is a table that matches on IPv4 destination addr whose action is to determine the output port

Software Data Plane

Why software?

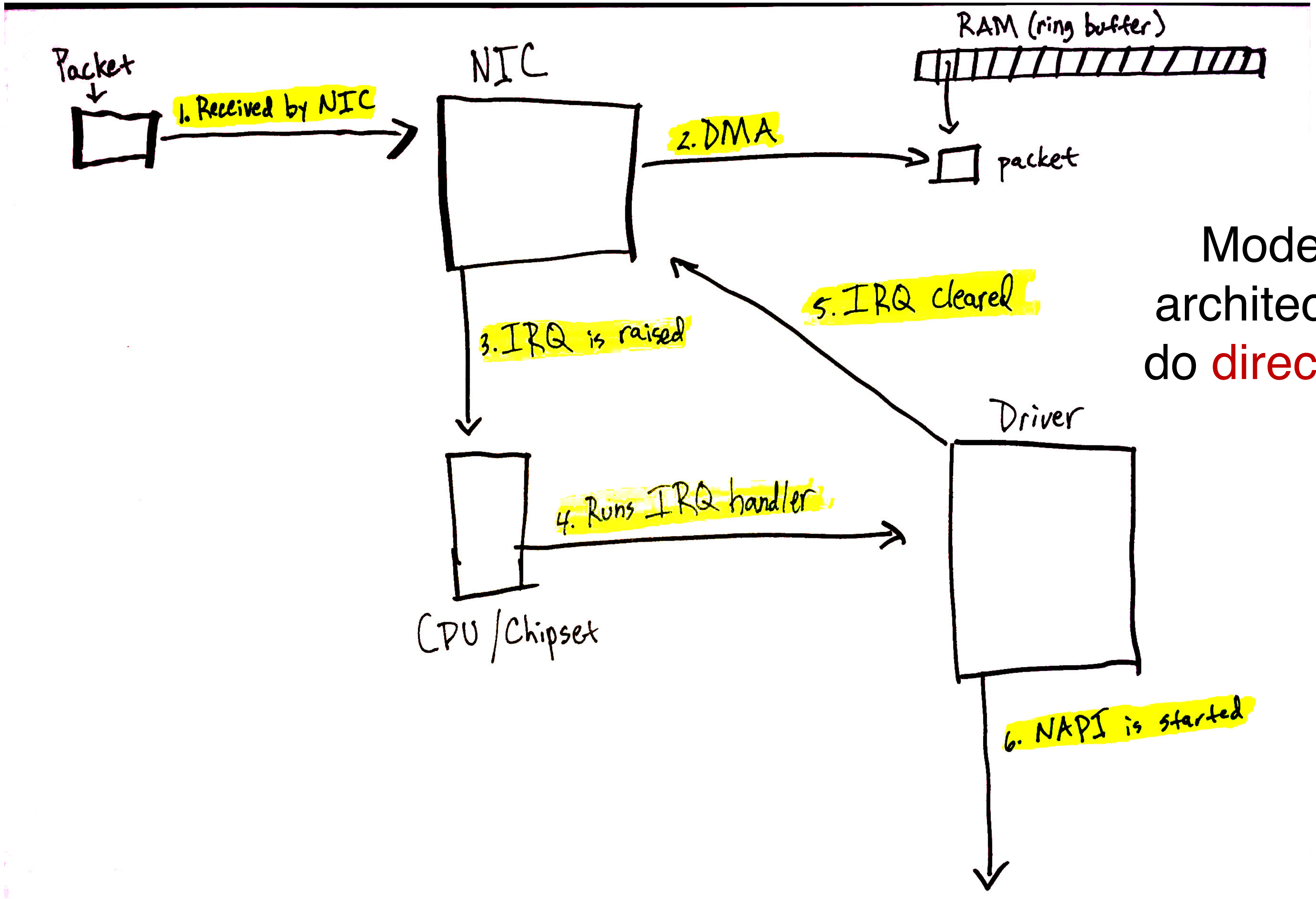
- Applications run in software. Get packets to/from apps quickly
- Software routers:
  - virtualization and cloud (e.g., openvSwitch)
- Middleboxes (network functions)
  - Network Address Translation, mobile processing nodes (packet gateways, radio controllers, …), tunneling gateways (IPsec/SSL VPN), traffic analysis & security (IDS, firewalls, spam), CDNs/caches, video accelerators, …

Packet processing on Linux

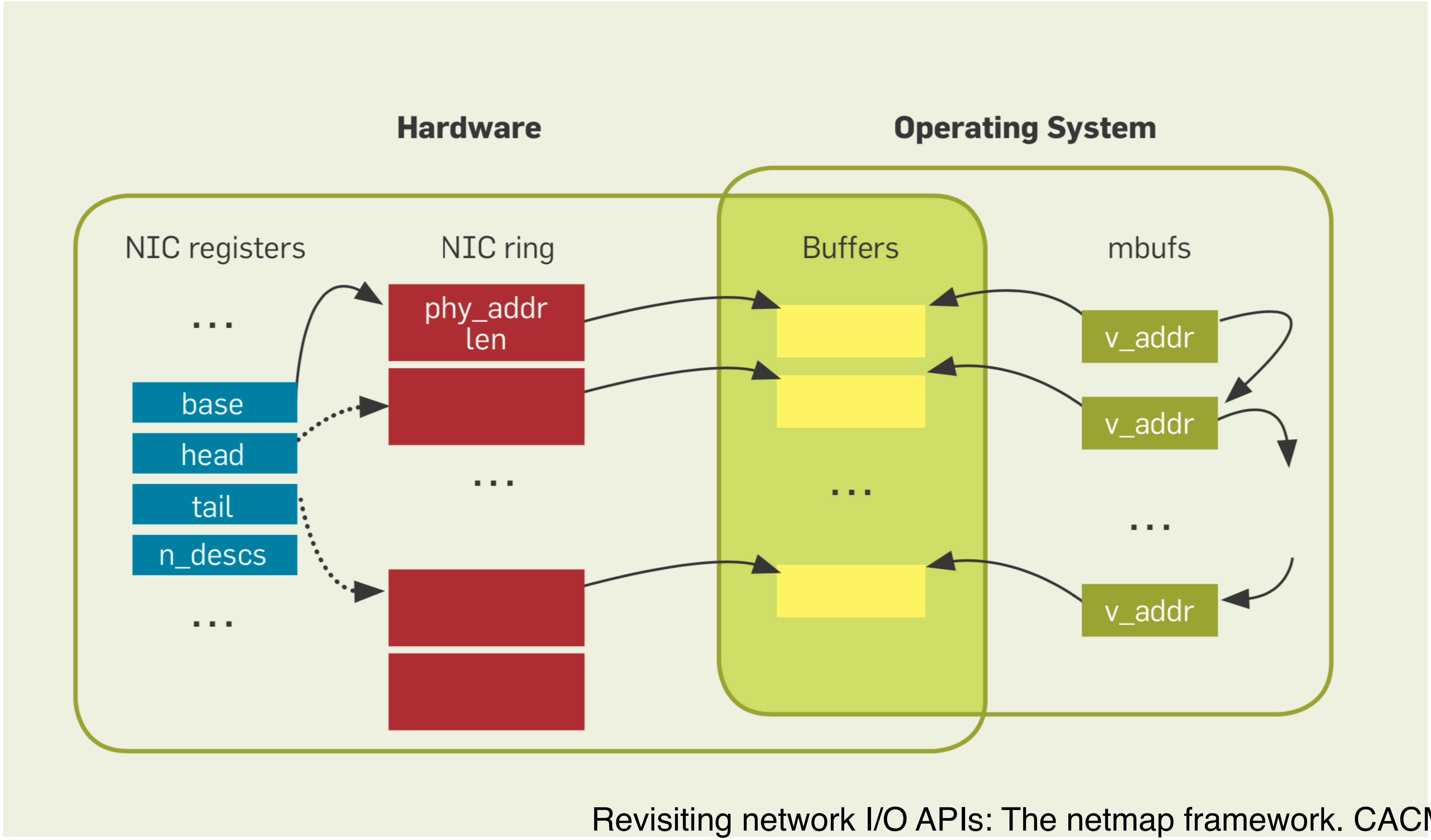
Receive path

How is data received in software?

- Have CPU poll the network interface card (NIC) memory to copy data
- Interrupt from the NIC (“data is available”), then CPU reads memory
- Direct Memory Access (DMA): NIC moves data to memory
    - Reduce or remove CPU from the “data moving” loop
    - Large data or scattered data



Modern NICs and architectures can also do direct cache access (DCA)

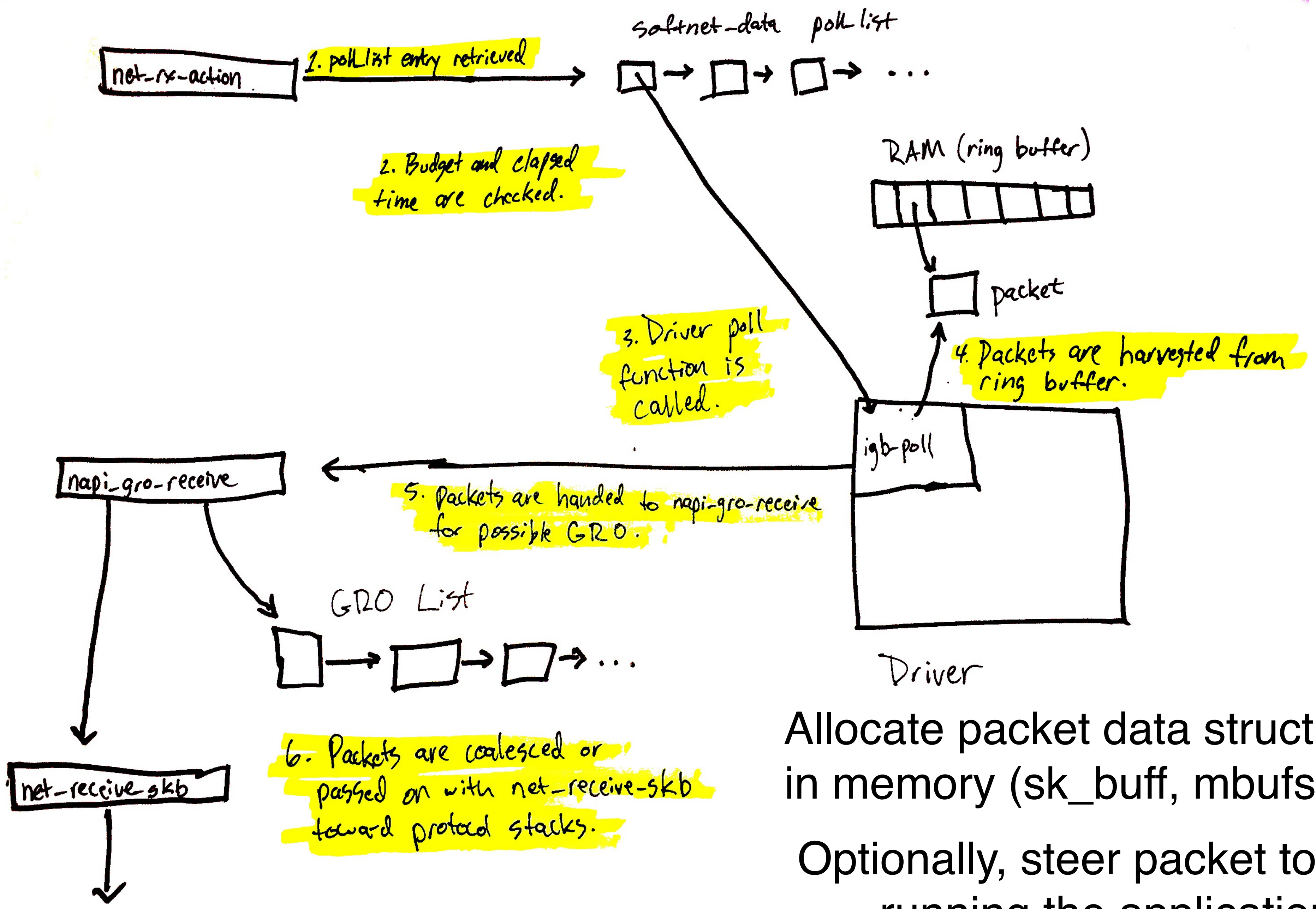


Revisiting network I/O APIs: The netmap framework. CACM'12

Interrupt mitigation

- Interrupt processing at high rate and priority prevents any other part of the system from progressing (receive livelock).
- Mitigations:
- (1) Interrupt coalescing:
  - Wait (at NIC) for more packets or a timeout until interrupting
- (2) Polling to schedule the work, avoiding preemption
- (3) CPU or packet quotas on polling to ensure other parts of the system (e.g. user space app) can progress
  - Re-enable interrupts if there is less work than allotted quota

RSS
aRFS


softnet_data poll list
net_rx_action
1. poll list entry retrieved
2. Budget and elapsed time are checked.
RAM (ring buffer)
packet
3. Driver poll function is called.
4. Packets are harvested from ring buffer.
igb_poll
Driver
napi_gro_receive
5. Packets are handed to napi_gro_receive for possible GRO.
GRO List
6. Packets are coalesced or passed on with net_receive_skb toward protocol stacks.
net_receive_skb

Allocate packet data structures in memory (sk_buff, mbufs, …)

Optionally, steer packet to core running the application