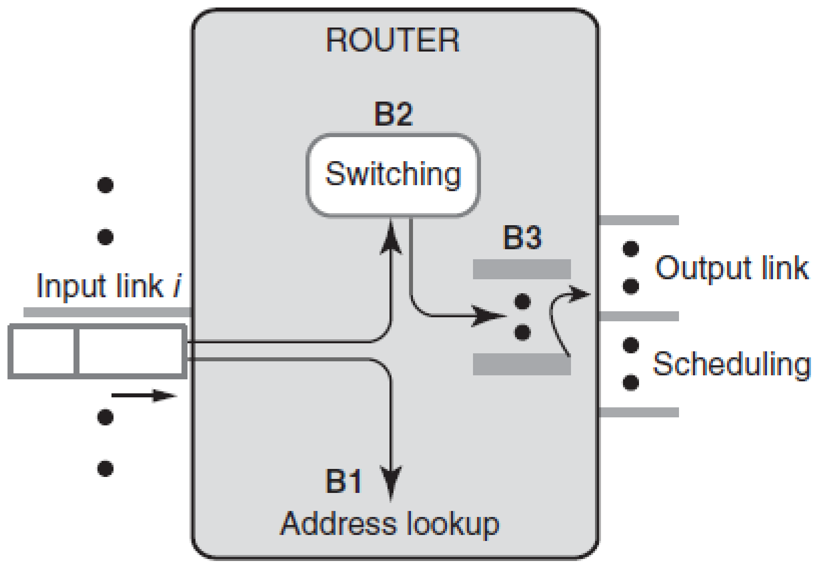
Switches

Switching

Switching

The core part of a router (or in general a switch) moves the packets from the input link to the output link



Switching

The switch must decide which input and output links should be matched

- packet-by-packet (nanoseconds!)
- as many couples of links as possible at the same time
- possibly multicast

Mathematically this is a bipartite matching problem

- match as many input links to as many output links as possible
- in 8 ns @40Gbit/s!

Several tricks:

- trade accuracy for time
- hardware parallelism
- randomization
- priorities

Shared-memory Switches

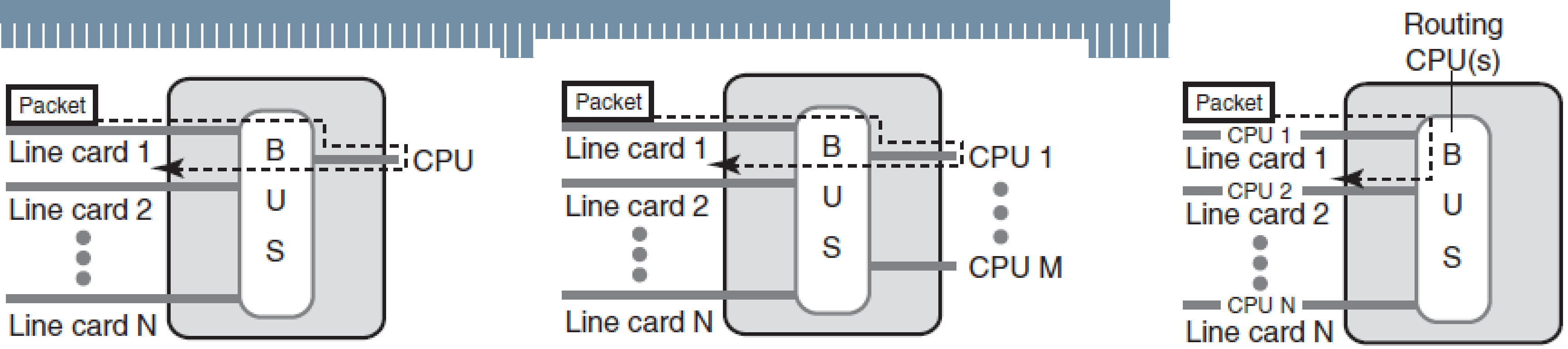
Packets are read into memory as they arrive and read from memory as output links become available

Problem: memory bandwidth

- with 32 input / output 1Gbit/s-ports, memory bandwidth must be at least 64 Gbit/s

Feasible with small number of ports (and/or slow links)

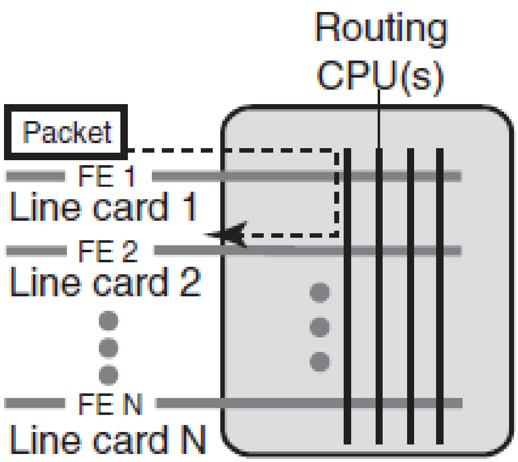
Bus Architecture

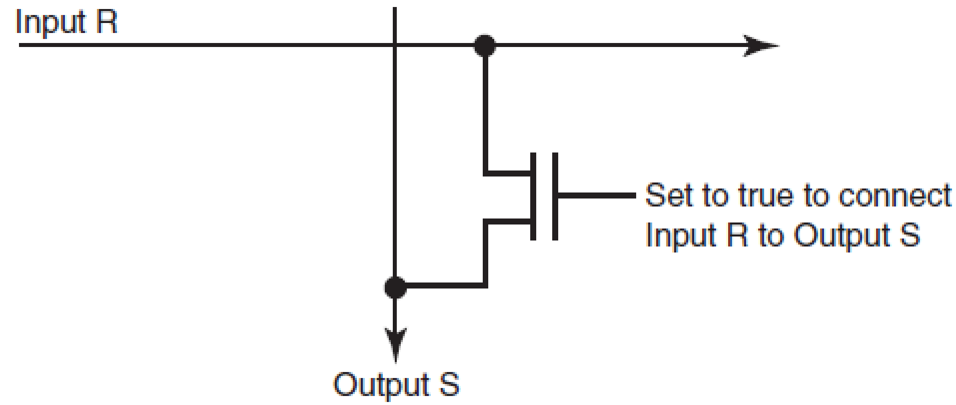


One packet at a time on the bus. Forwarding decision by

- a single CPU
  - slow, a CPU has a lot of other things to do
  - each packet must traverse the bus twice (once to the CPU and once from the CPU)
- multiple CPUs, each CPU serves *M/N* line cards
  - still each packet must traverse the bus twice
  - coordination overhead
- one CPU per line card + routing CPUs on the bus
  - the bus is still a bottleneck

The Crossbar Switch





A crossbar is a set of 2*N* buses: one for each input, one for each output

Possibly *N* times faster than a single bus if you can find *N* disjoint source/destination pairs

For every pair of buses there is a crosspoint (N^2 total)

- conceptually a transistor, in practice more sophisticated

Variable-length packets are generally divided in fixed-size cells and time is divided in fixed-duration time slots

For each time slot the scheduler must decide the status of the N^2 crosspoints.

Crossbar scheduling

Problems to solve:

- every output connected to at most one input
- maximixe number of input/output relations

Factors limiting parallelism:

- no data at some input port
- contention at the output port

An example: take-a-ticket algorithm

- works with variable sized packets (no need to split in cells)
- similar to a real-life queue management system

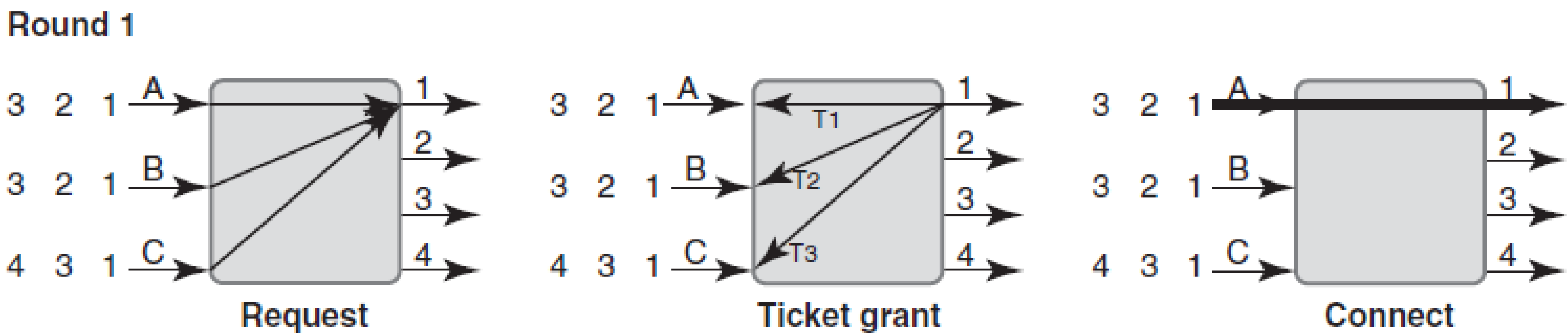
Take-a-Ticket

Basic idea

- each output line card *S* maintains a distributed queue of packets waiting to be sent to *S*
- the various pieces of the queue are stored in a single queue at the input *R* (input buffering)
- if line card *R* wants to send a packet to *S* asks on a control bus for a ticket number

- when S is free, it announces on the control bus the next number to be served
- S sets the R/S crosspoint
- R sends the packet

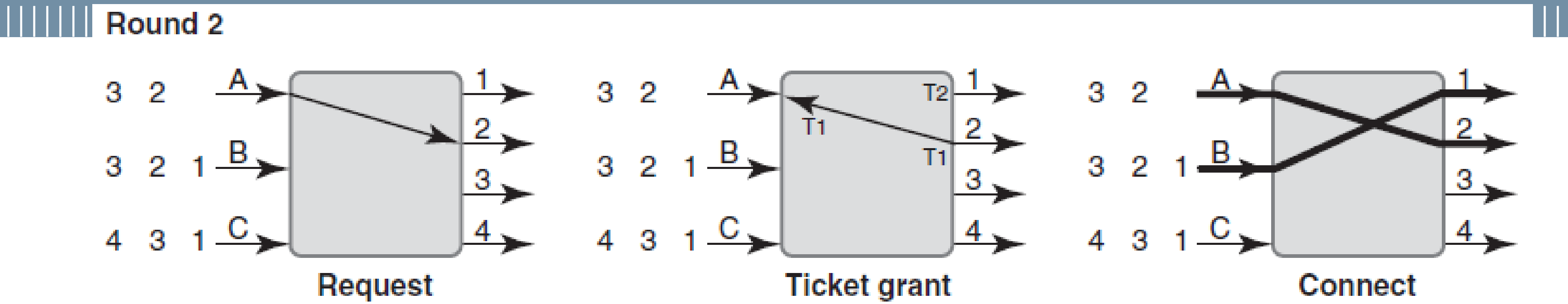
TaT: round 1



At the input queues each packet is labeled with its destination port

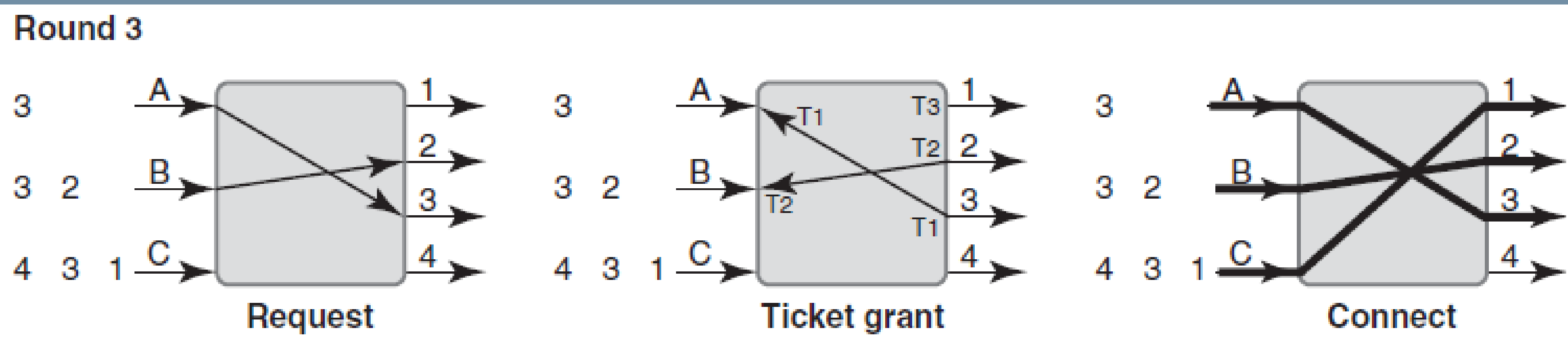
- A wants to send a packet to 1, 2, and 3
- A, B, and C ask for a ticket for exit 1
- Port 1 gives ticket #1 to A, #2 to B, and #3 to C
- A can transmit
- the other ports are blocked (Head-of-Line blocking problem!)

TaT: round 2



- A asks for a ticket for port 2
- port 2 gives ticket #1
- port 1 announces #2
- port 1 announces #1
- A, B transmit
- C is blocked

TaT: round 3



- A asks for a ticket for port 3, which gives #1
- B asks for a ticket for port 2, which gives #2
- ports 1, 2, 3 announce #3, #2, #1
- A, B, C transmit
- no port is blocked

Head-of-Line blocking

With single input queues all the packets are blocked if the head-of-line is blocked

Rough estimate of throughput saturation:

- assume each head-of-line wants to go to one of the N output ports chosen uniformly and randomly
- Probability that a given output port is used is:

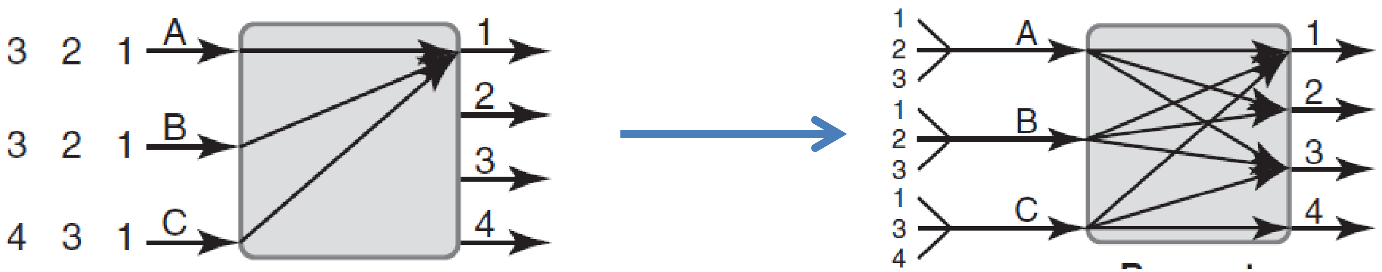
$$1 - \left(1 - \frac{1}{N}\right)^N$$

- For N large converges to (1-1/e) ~ 63%
- More precise calculations give efficiency ~ 58%
- Non uniform traffic distributions can give worse results

How to deal with Head-of-Line Blocking

- Speedup the switch and queue ot the output
  - to deal with the worst case the needed speedup is *N*
  - more likely, the necessary speedup is *k*, where *k* is the expected number of cells contending for the same output
- Virtual Output Queues (VOQ)

Parallel Iterative Matching (PIM)



Decompose the input queues in separate input queues: one for each output.

Note that PIM requires fixed size packets and that time is divided in slots.

Parallel Iterative Matching (PIM)

1) Request
   - Each input port communicates its status using a bitmap.
   - Each bit represents an output queue.
     - 1 = virtual output queue has at least one packet
     - 2 = virtual output queue is empty

2) Grant
   - Each output port choses an input port.
   - When an output port receives request from multiple input ports, it chooses one at random.
   - Note that multiple

3) Accept
   - If an input port was chosen by multiple output ports, it chooses one randomly.

Parallel Iterative Matching (PIM)