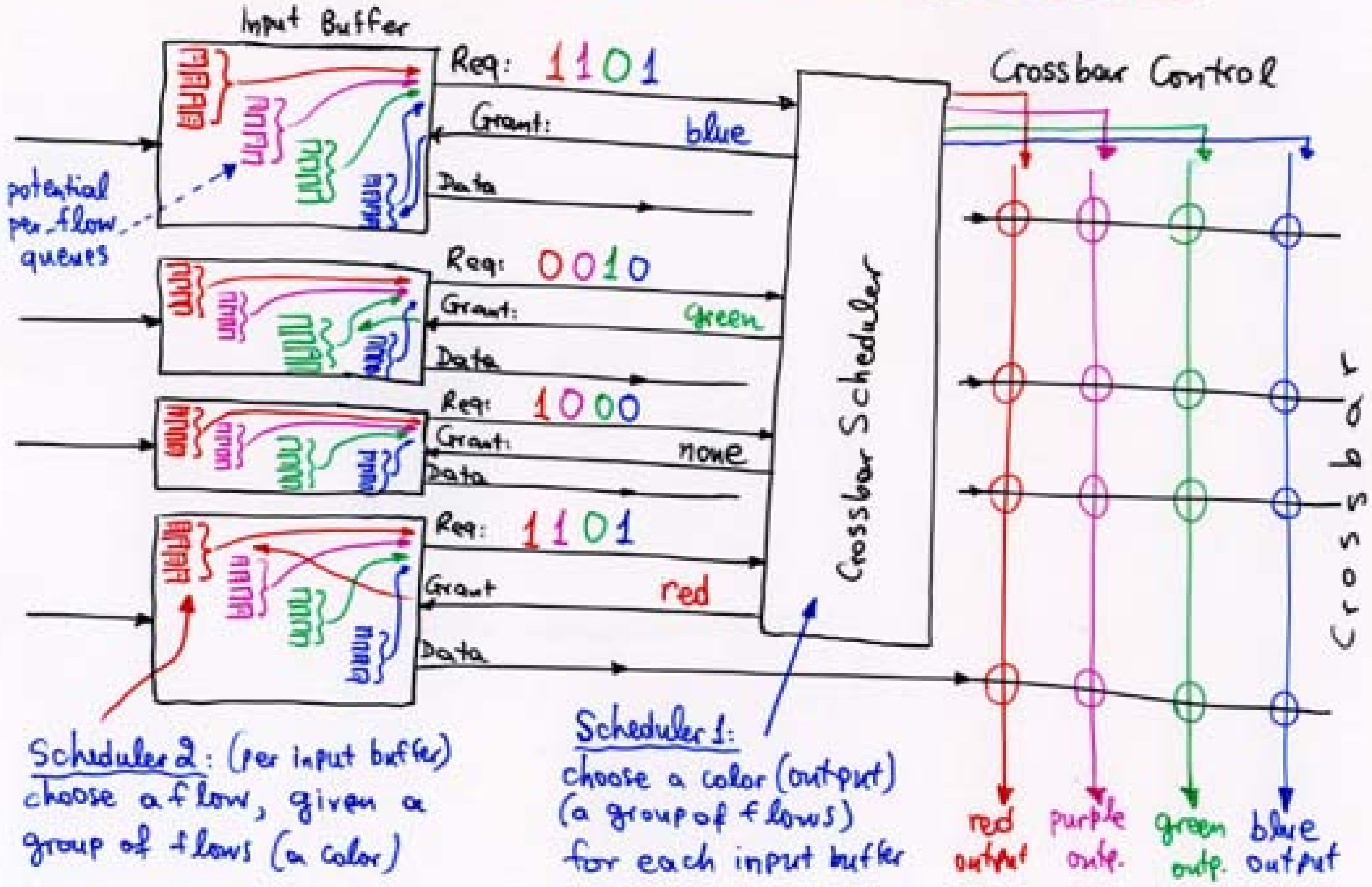
4.3 Virtual Output Queue (VOQ) Input Q'ing and the Crossbar Scheduling Problem

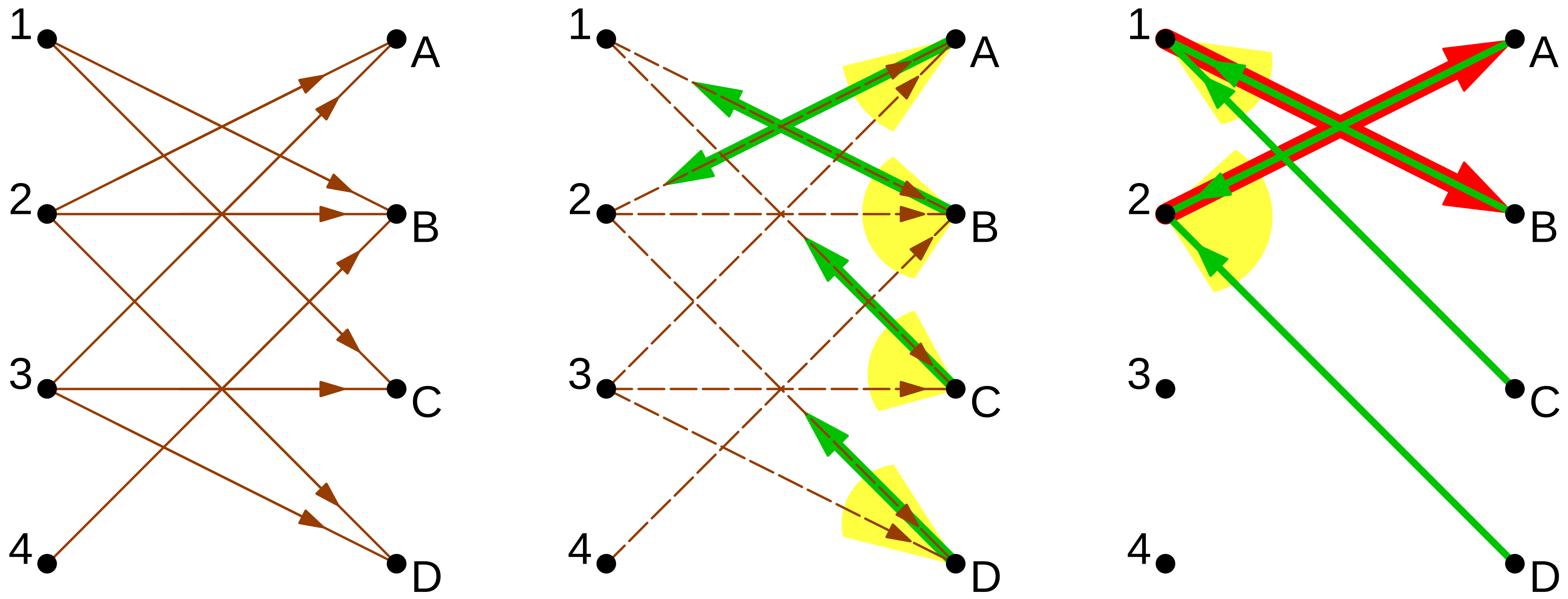
- Crossbar Switch with one Buffer Memory per Input Line
- Throughput per Buffer Memory = 1 (incoming) + 1 (outgoing)
- Multiple (one per output) Queues per Buffer Memory:

  "Virtual Output Queues – VOQ"
  - N queues per input, N^2 queues total, for $N \times N$ switch
- Crossbar Scheduling, per cell-time:
  - pairings ("marriages") between inputs and outputs – each input specifies a subset of the outputs that it accepts to be married to
  - interdependent decisions – difficult problem!

Advanced Input Queueing (Buffering) – "Virtual Output" Queues



Crossbar Scheduling: Parallel Itarative Matching (PIM)



Request Phase:

All inputs send their requests in parallel

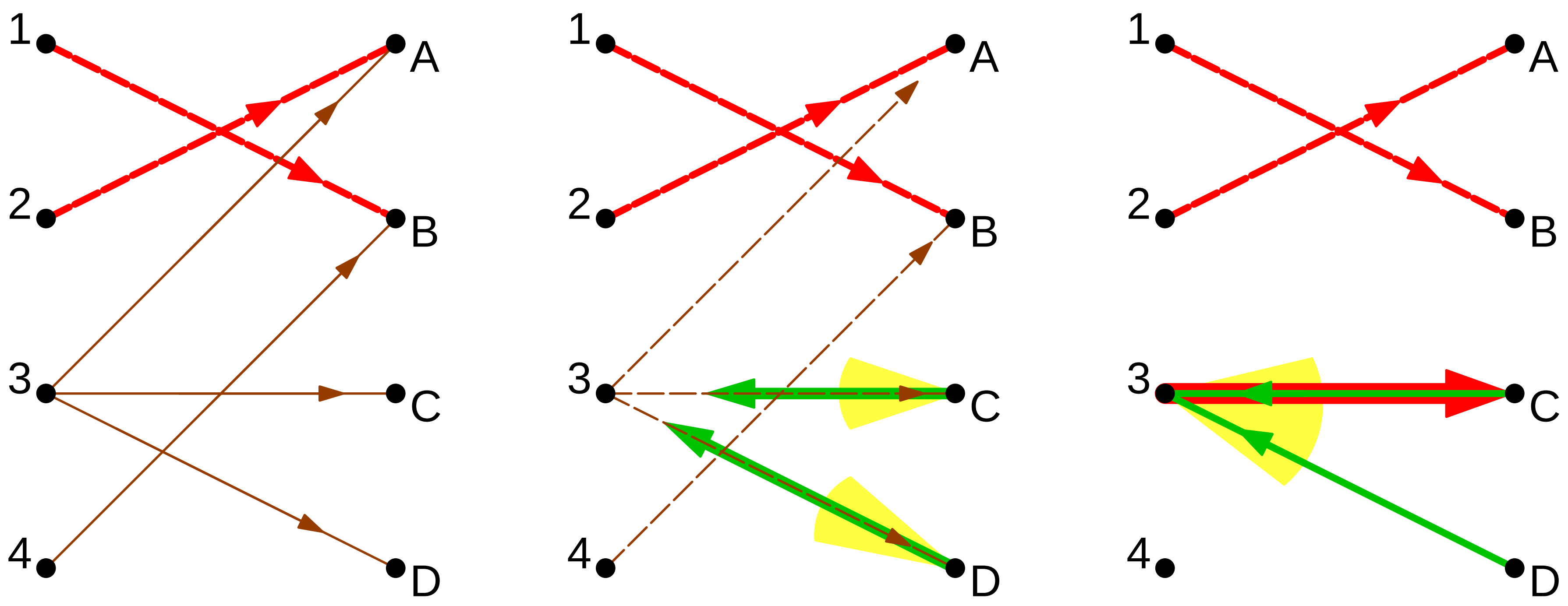
Grant Phase:

Each output, *independently*, grants to one of the requests that it received

Accept Phase:

Each input accepts one of the grants that it received

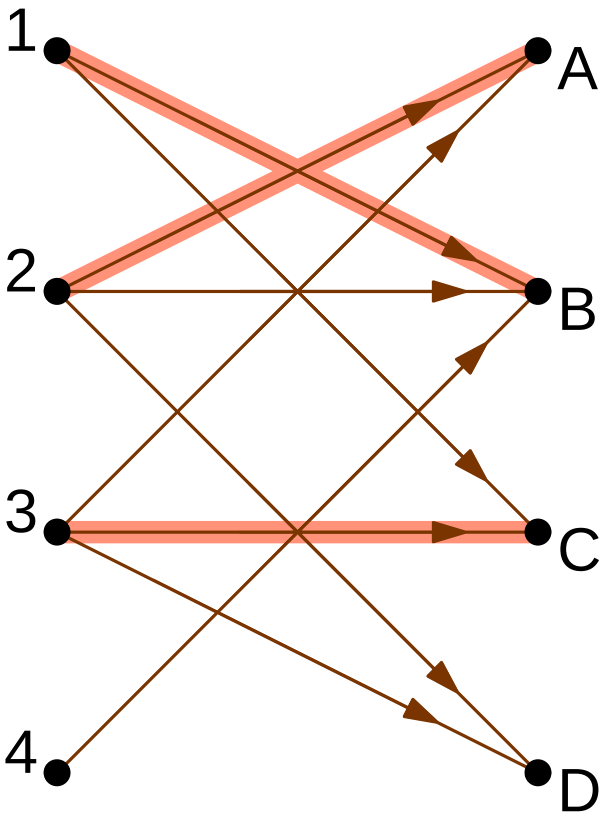
First Iteration



Request Phase:	Grant Phase:	Accept Phase:
Unmatched inputs (received no grant) resend their requests	Unmatched outputs (rcv'd no accept) grant to one of the received requests	Unmatched inputs accept one of the received grants

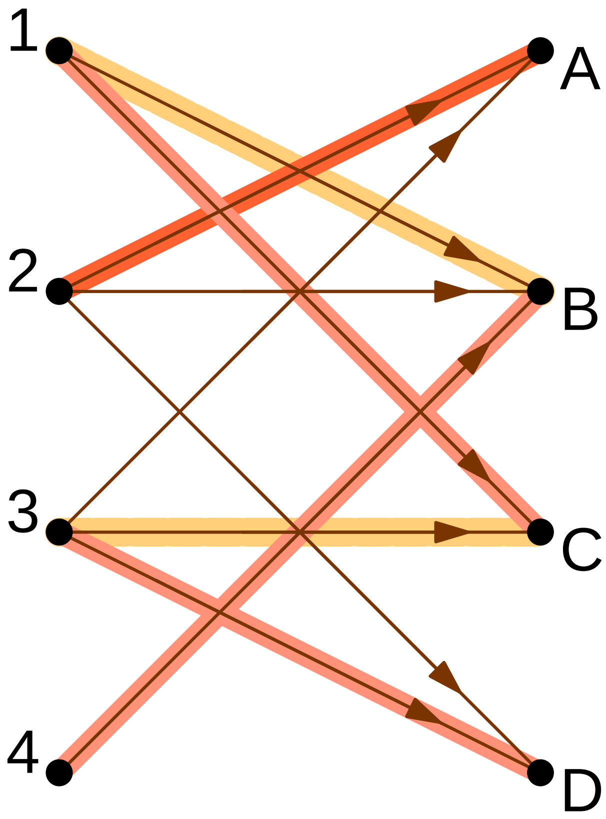
Second Iteration

- Original PIM proposal: outputs grant randomly among requesting inputs, inputs accept randomly among granting outputs



Maximal Matching

Cannot add any new connection
without breaking some
already made connection(s)



Maximum Matching

Maximum possible number
of connections
for the given request pattern

Maximum Matching is complex ... and may be Unfair:

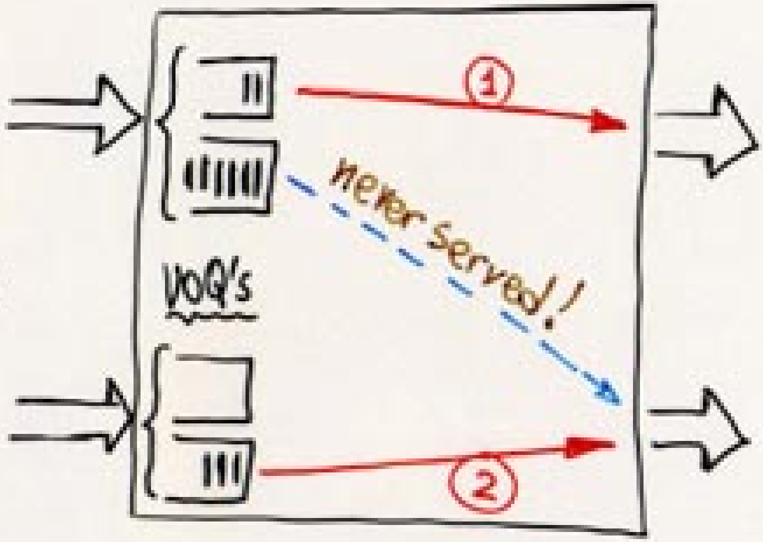
- N×N switch
- M requests

- O(N·(N+M)) deterministic algorithm

  Tarjan: Data Structures & Network Algorithms
  SIAM, 1983

- O(N+M) randomized algorithm

  Karp e.a.: ACM STOC, 1990



Performance of PIM:

- each iteration resolves, on average, $\geq \frac{3}{4}$ of the remaining unresolved requests

$$\Downarrow$$

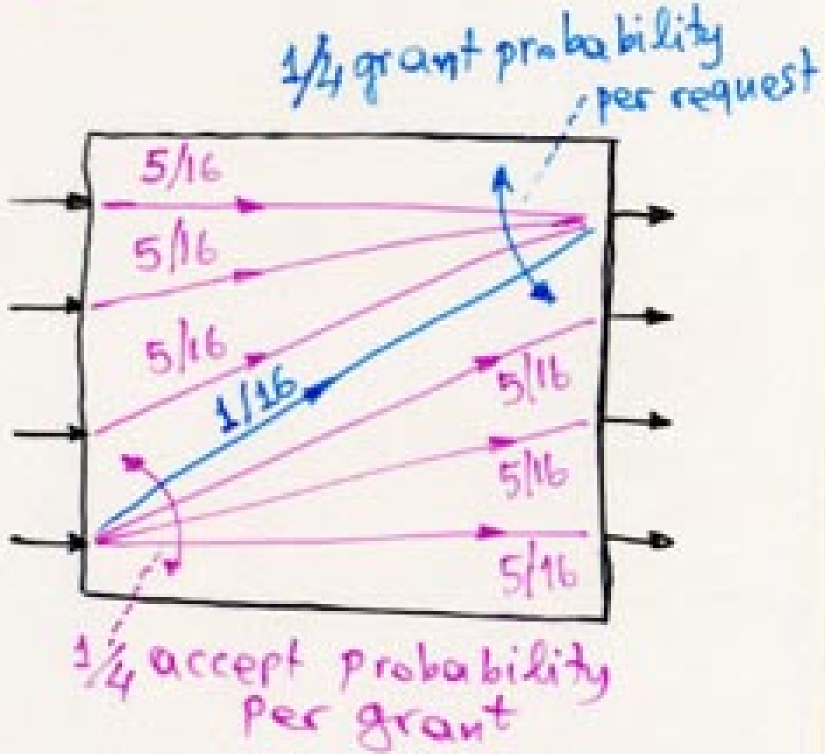
$O(\log N)$ iterations

- 16×16 switch:

# iterations	1	2	3	4
% matched	64-87%	88-100%	97-100%	99.9-100%

- delay (VOQ w. PIM) ≈ 2× delay (Output Queueing)
- how to implement random selection ???

Fairness Issue:



(weighted random selection ???)

iSlip: Most Popular, Practical Crossbar Scheduler

- Practical variation of PIM
- Widely used in commercial switch products
- Nick McKeown: “The iSLIP Scheduling Algorithm for Input-Queued Switches”, IEEE/ACM Tr. on Networking, April 1999
- Performance properties:
  - performs well under uniform and heavy load, when most VOQ’s are non-empty and matching almost “rotate” among inputs & outputs
  - adds delay under medium loads, until most VOQ’s become non-empty
  - does not perform very well under *“unbalanced”* traffic (each input preferentially sends to one or a few “favored” output(s) of itself)

iSLIP → variation of PIM → fewer iterations; → better fairness

Nick McKeown: IEEE/ACM ToN April 1999

• was used in CISCO GSR-12000, Tiny Tera, Abrizio/PMC-Sierra, e.a.

(a) REQUEST: like PIM

(b) GRANT: • PIM: each output randomly grants to a requesting input

• iSLIP: grant in a Round-Robin fashion to a requesting input; top priority = next of: ~~previous grant~~ NO!!! – see below

previous grant that was accepted!

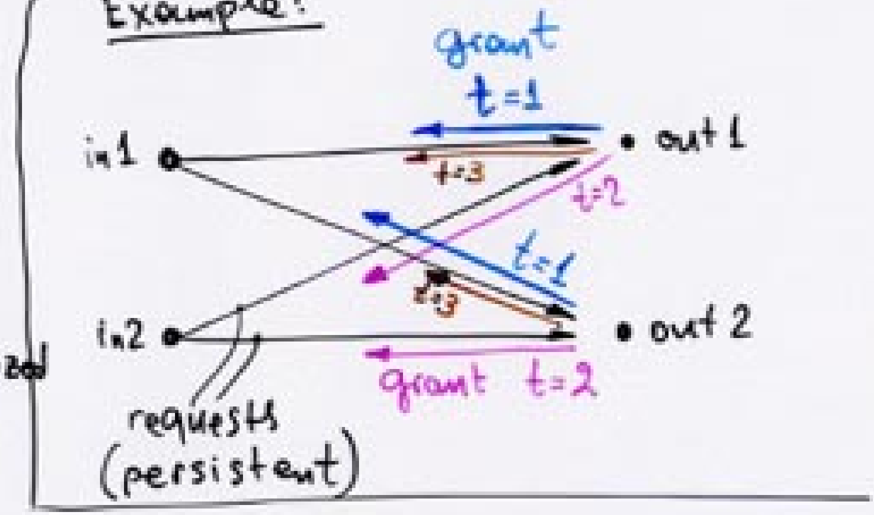
(c) ACCEPT: • PIM: each input randomly accepts one of the granting outputs

• iSLIP: accept in Round-Robin priority a granting output, with top priority = next of previous accept

- if we were granting round-robin after the previous grant, without regard as to whether the grant was accepted or not, then the grant pointers may get synchronized in a "bad way" and stay that way for a long time, resulting in very poor perf.

- perform well even with a single iteration ! (versus 2 to 4 iterations of PIM)
  reason: under heavy load, with ~all inputs requesting ~all outputs, pointers get and stay de-synchronized and scheduler degenerates into time-division multiplexing
  ⇒ can reach even 100% throughput under uniform load

Example:



- more iterations improve the delay
- has good fairness properties
- is relatively easy to implement

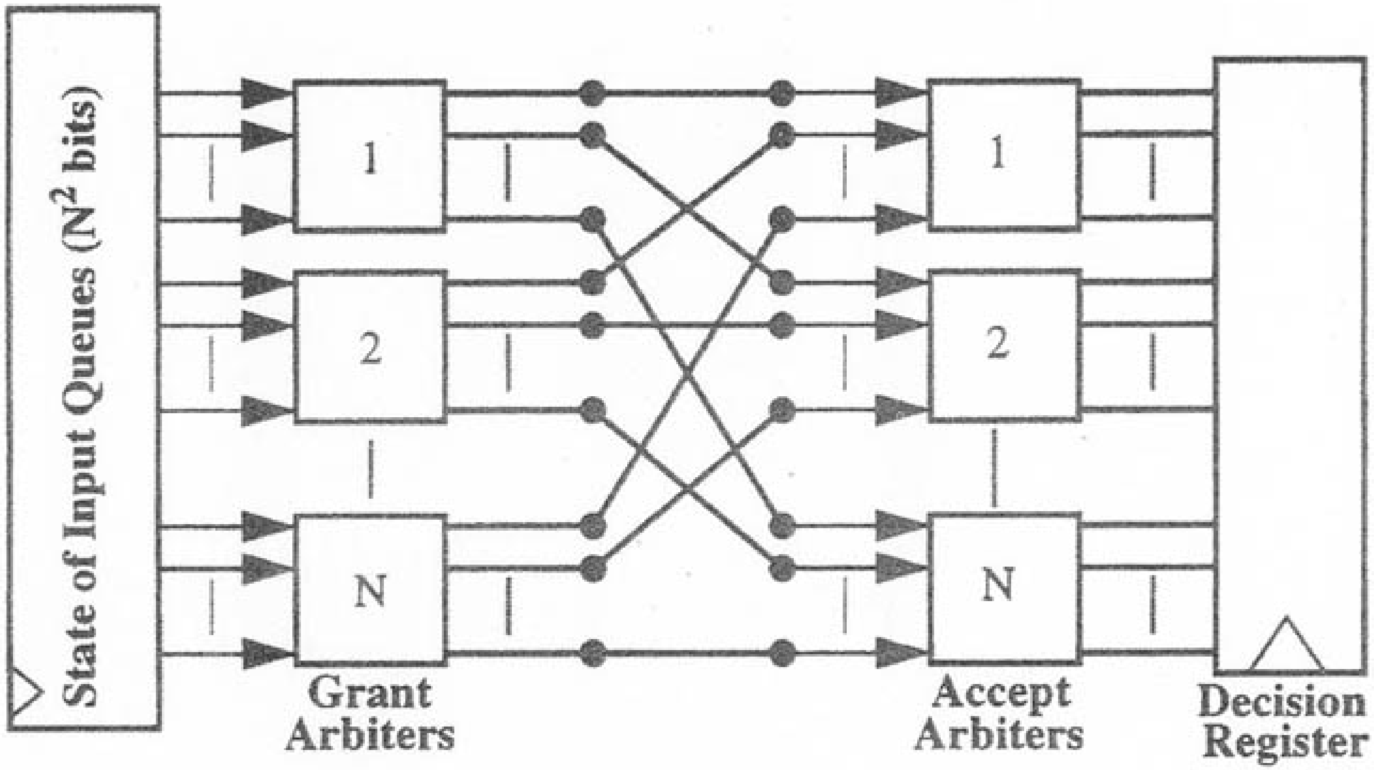
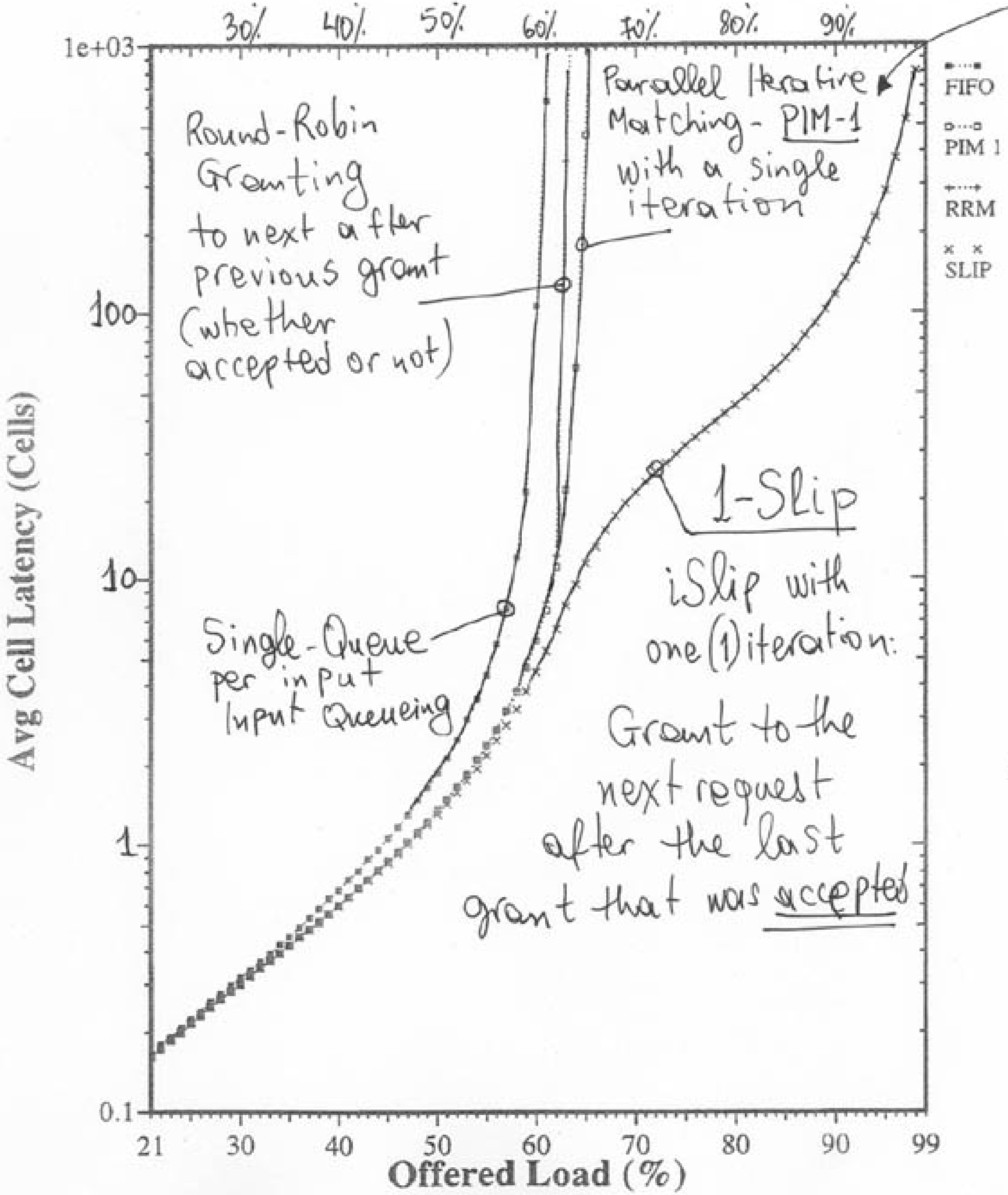


Fig. 21. Interconnection of $2N$ arbiters to implement iSLIP for an $N \times N$ switch.



Probability an input remains ungranted

$$= \left(\frac{N-1}{N}\right)^N \text{ for } N \times N \text{ switch}$$

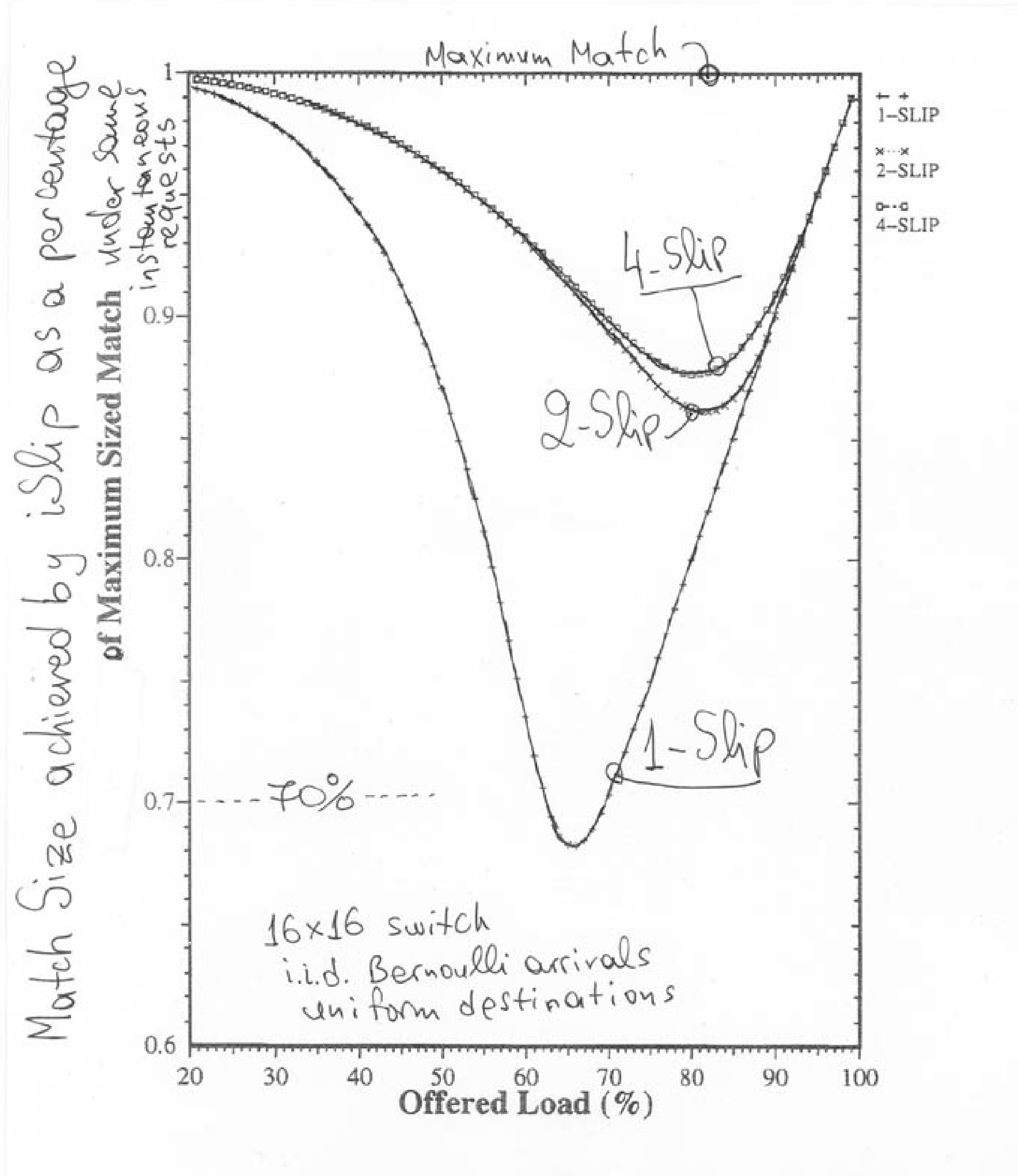
$$\xrightarrow{N\to\infty} 1 - \frac{1}{e} \approx 63\%$$

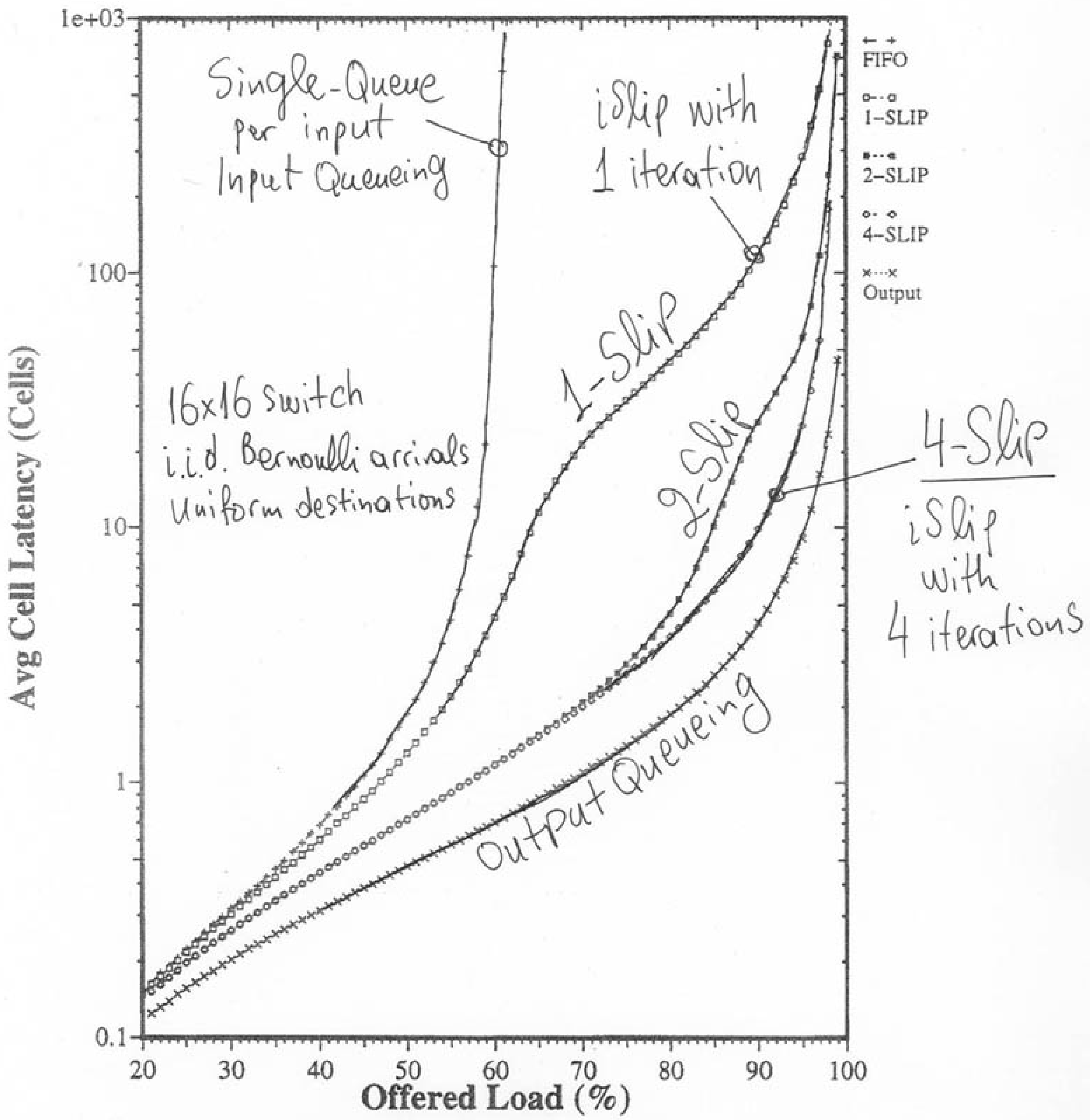
16x16 switch simulations

i.i.d. Bernoulli arrivals

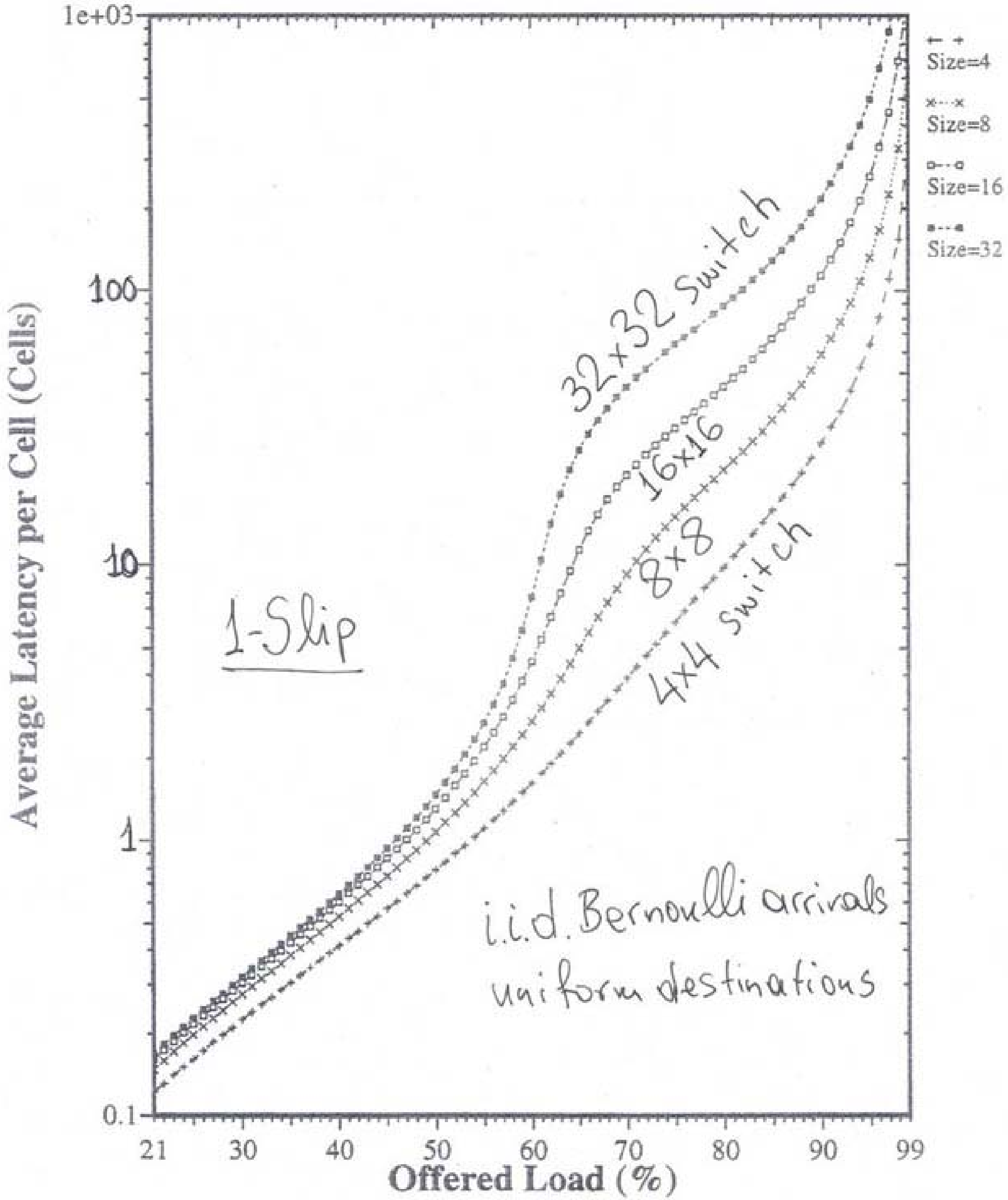
(non-bursty)

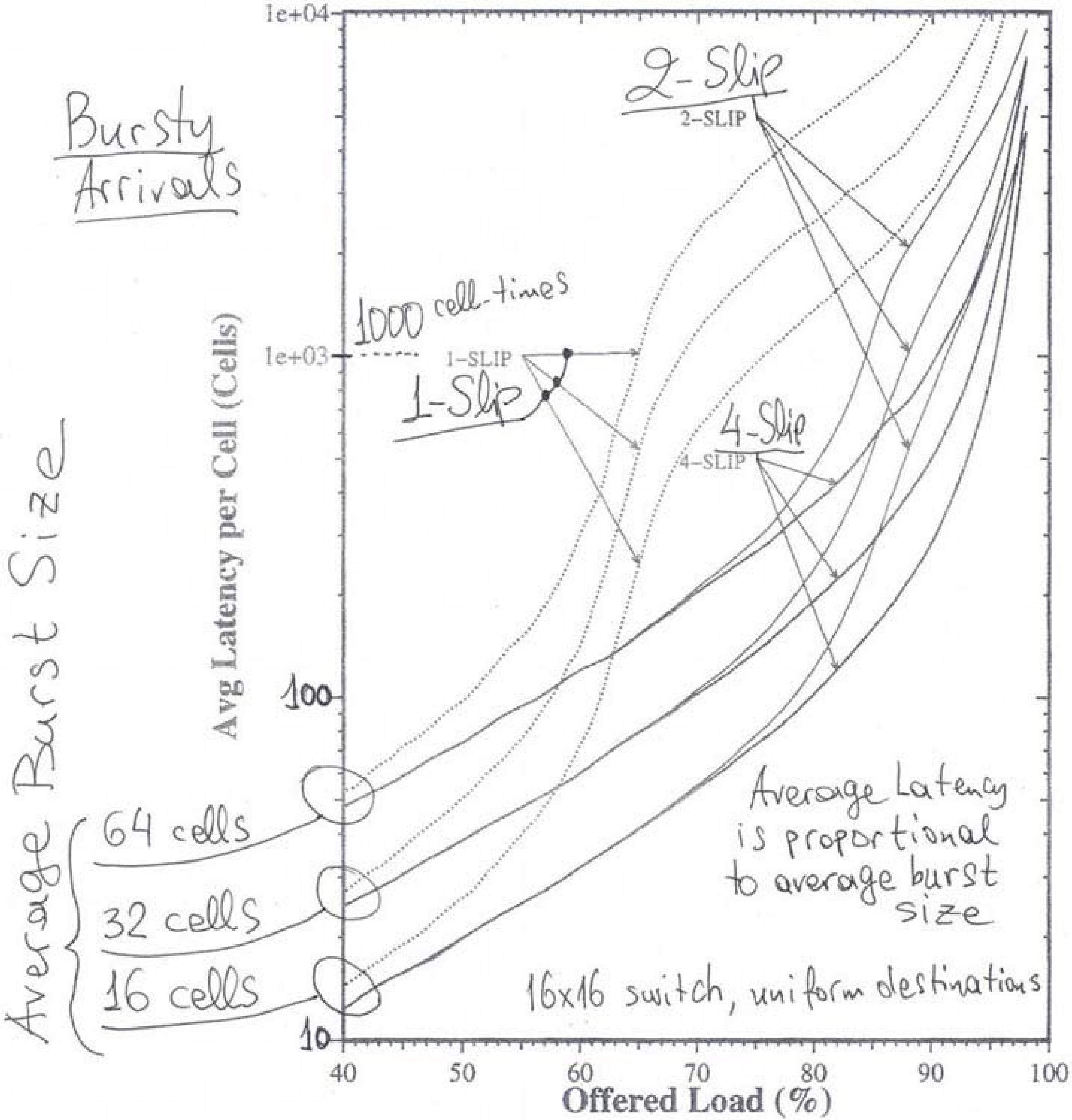
uniformly destined


Maximum Match
1-SLIP
2-SLIP
4-SLIP
4-Slip
2-Slip
1-Slip
70%
16x16 switch
i.i.d. Bernoulli arrivals
uniform destinations
Match Size achieved by iSlip as a percentage of Maximum Sized Match under same instantaneous requests
Offered Load (%)
1
0.9
0.8
0.7
0.6
20
30
40
50
60
70
80
90
100

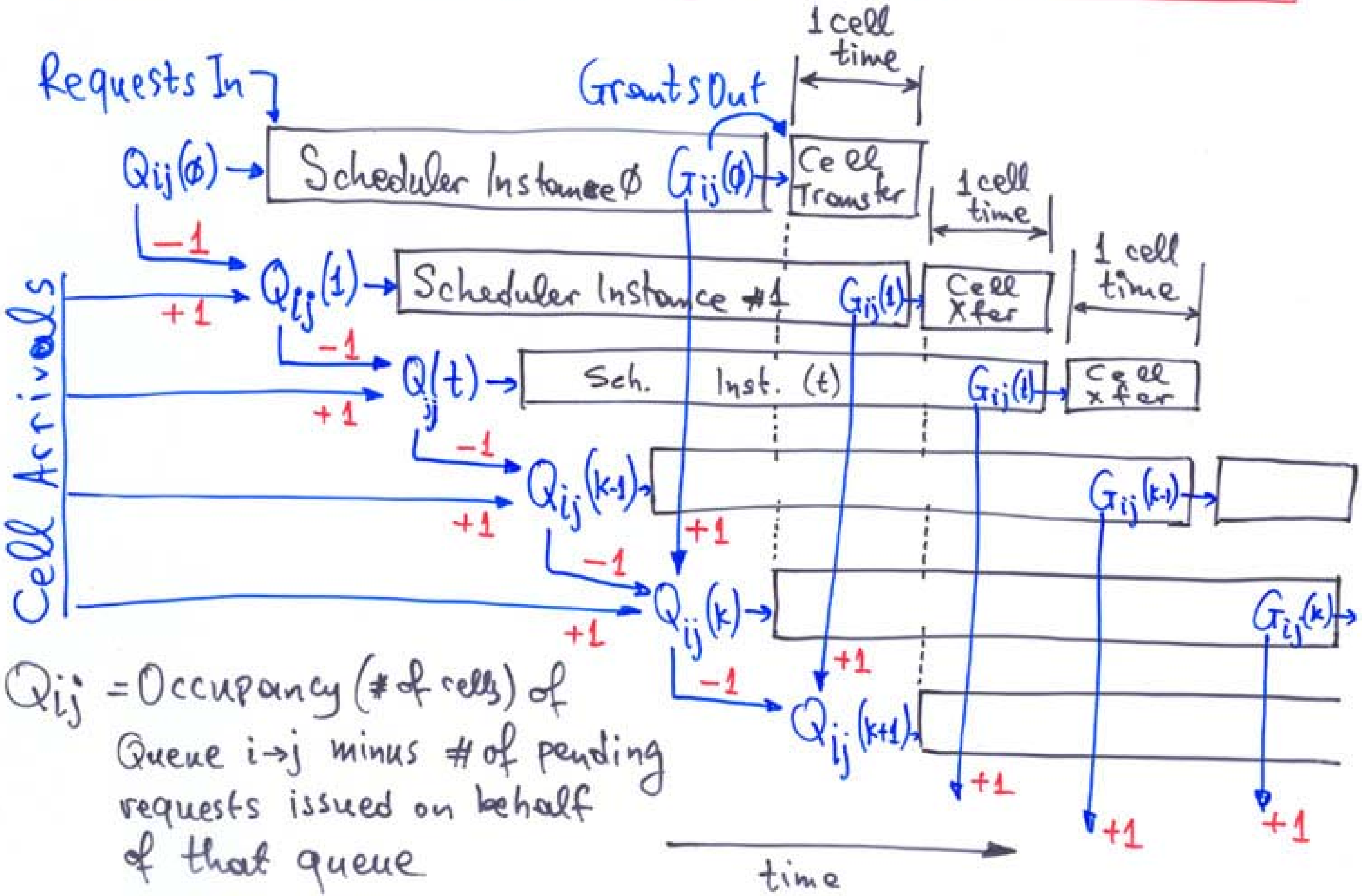


Avg Cell Latency (Cells)
Offered Load (%)
FIFO
1-SLIP
2-SLIP
4-SLIP
Output
Single-Queue per input Input Queueing
iSlip with 1 iteration
1-Slip
2-Slip
4-Slip
iSlip with 4 iterations
16x16 switch
i.i.d. Bernoulli arrivals
Uniform destinations
Output Queueing


1e+03
100
10
1
0.1
Average Latency per Cell (Cells)
21
30
40
50
60
70
80
90
99
Offered Load (%)
Size=4
Size=8
Size=16
Size=32
32x32 switch
16x16
8x8
4x4 switch
1-Slip
i.i.d. Bernoulli arrivals
uniform destinations


Bursty Arrivals
Average Burst Size
64 cells
32 cells
16 cells
1000 cell-times
1-SLIP
1-Slip
2-Slip
2-SLIP
4-Slip
4-SLIP
Average Latency is proportional to average burst size
16x16 switch, uniform destinations
Avg Latency per Cell (Cells)
1e+04
1e+03
100
10
40
50
60
70
80
90
100
Offered Load (%)

Pipelined Scheduling for Cell-Based Crossbars



Qij = Occupancy (# of cells) of Queue i→j minus # of pending requests issued on behalf of that queue

- Cell Arrivals increment Q_{ij}
- When $Q_{ij} \geq 1 \Rightarrow$ a Request R_{ij} is issued
- Requests issued decrement Q_{ij} (pending decision on cell transfer)
- When Pending Decision is resolved:
  - Successful Grants leave Q_{ij} as is (already decremented)
  - Failed Grants re-increment Q_{ij} (restore unaccepted request)

Assumption for this scheme to work: fixed-size cells

$\Rightarrow$ requests for cells and actual cell transfers are interchangeable with each other: if the "first" request, on behalf of the "first" cell in the queue is not granted, then the "second" request (issued on behalf of the "second" cell) will result in transfering the "first" cell, if granted.

Ref: Oki, Rojas-Cessa, Chao: "A Pipeline-Based Approach for Maximal-Sized Matching Scheduling in Input-Buffered Switches" IEEE Communications Letters, June 2001 – http://acts.poly.edu/~chao/publications.html