

# An Efficient Randomized Algorithm for Input-Queued Switch Architectures

Paolo Giaccone

jointly with:

Balaji Prabhakar & Devavrat Shah

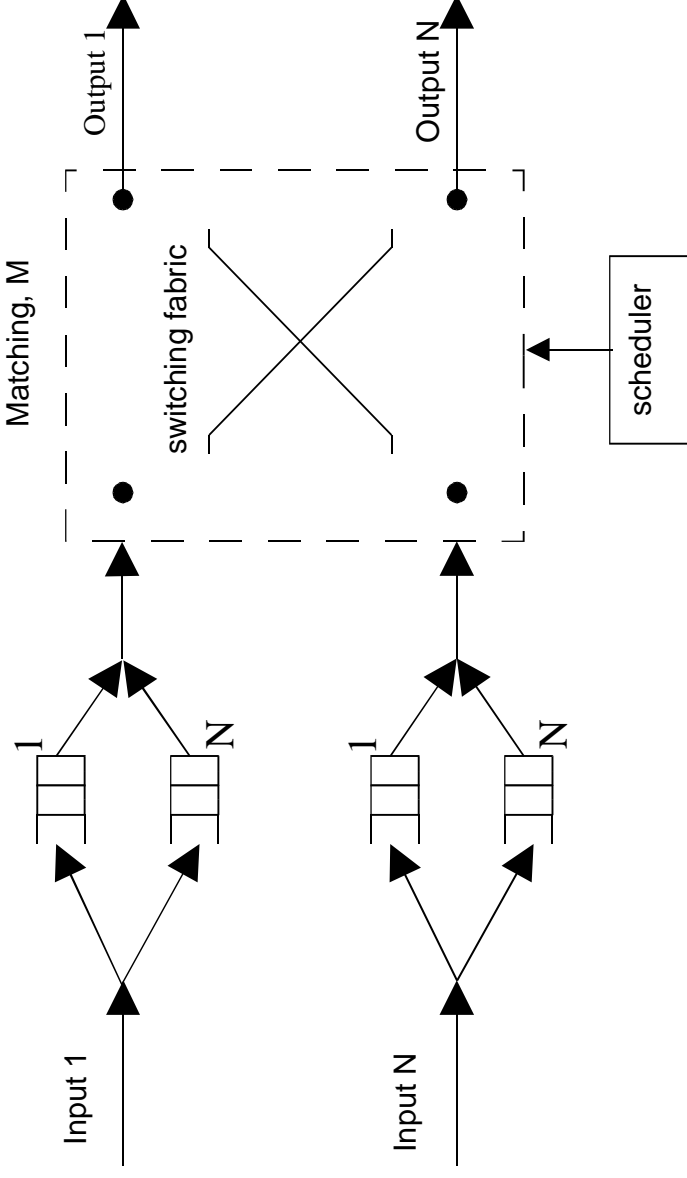
Hot Interconnects 9, Stanford, 22<sup>nd</sup> August 2001

# Outline

- Scheduling in IQ switches
- The randomized approach
- Algorithms: LAURA and SERENA
- Conclusions

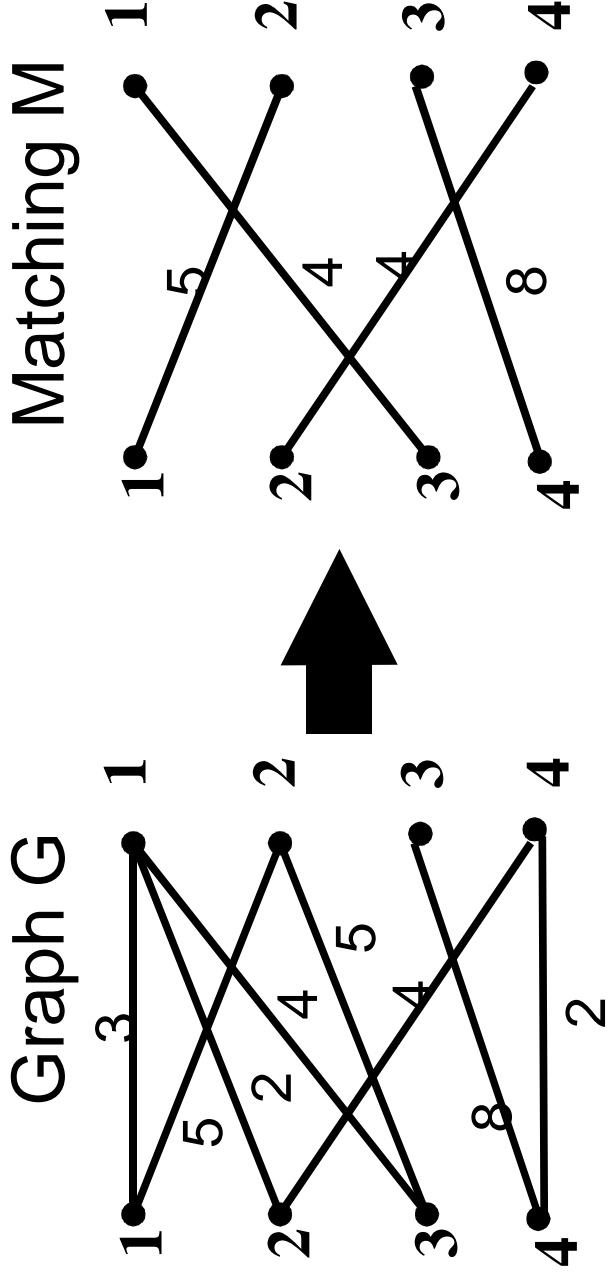
# Input-Queued Switch

- with Virtual Output Queue architecture



# Scheduling model

- scheduling can be modeled as a matching problem in a bipartite graph



- Maximum Weight Matching (MWM) is one "optimal" solution to the problem
  - achieves 100% throughput
  - has good delays

# Scheduling at very high speed

- In practice, IQ switches can be used as forwarding engine in high-speed routers
  - 1024 ports x 1 Gbps
  - 32 ports x 40 Gbps (OC-768)
- Schedule
  - should be computed in very little time
  - but, its complexity grows very fast with N
- MWM is not implementable
- Currently implementable algorithms
  - can be very inefficient

# The randomized approach

- Randomization is a method we will use to simplify the implementation
- The main idea of randomized algorithms is
  - to simplify the decision-making process by basing decisions upon a **small, randomly chosen** sample of the state rather than upon the **complete** state

# An illustrative example

- Problem: find the largest element of a set  $S$  of size 1 billion
- The **deterministic** algorithm: linear search
  - has a complexity of 1 billion
  - finds the absolute largest element  $R$
- The **randomized** algorithm: find the largest of 10 randomly chosen samples
  - has a complexity of 10
  - can find a very good approximation of  $R$  with an high probability
    - $P(R \text{ is at least the } 100 \text{ millionth largest element}) = 1 - (1/10)^{10}$

# The randomized approach

## Basic algorithm

- at time  $t$ , pick  $d$  matchings uniformly at random from all  $N!$  possible matchings
- use the heaviest of these matchings as the schedule
- RND I :  $d=1$
- RND II :  $d=32$

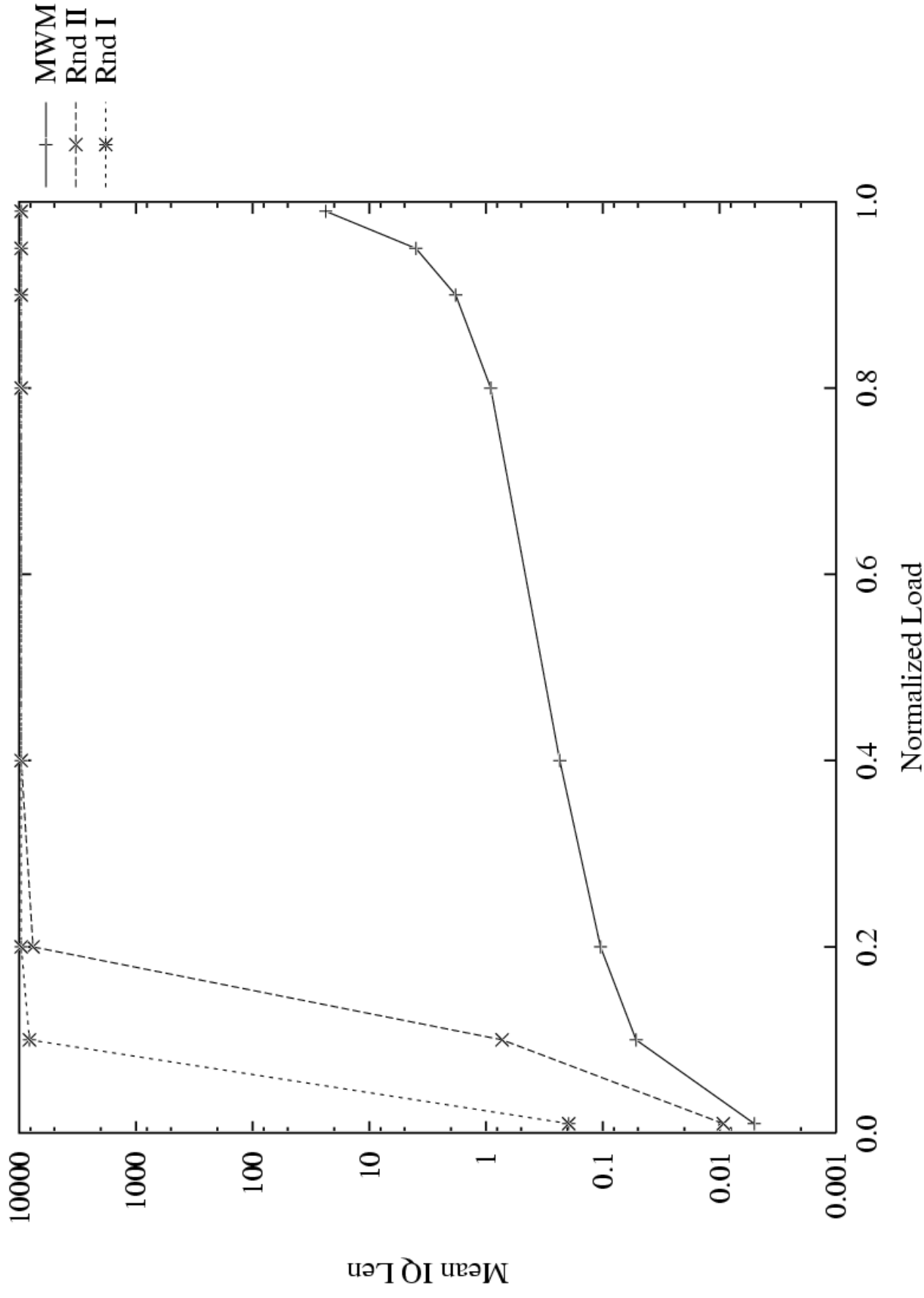
# Simulation scenario

- Switch Size : 32 X 32
- Input Traffic (shown for a 4 X 4 switch)
  - Bernoulli i.i.d. inputs
  - diagonal load matrix:
    - normalized load= $x+y < 1$
    - $x=2y$
- this is a good test-bed since:

$$\begin{bmatrix} x & y & 0 & 0 \\ 0 & x & y & 0 \\ 0 & 0 & x & y \\ y & 0 & 0 & x \end{bmatrix}$$

- most algorithms perform badly
- it is critical for random algorithms

# A simulation comparison



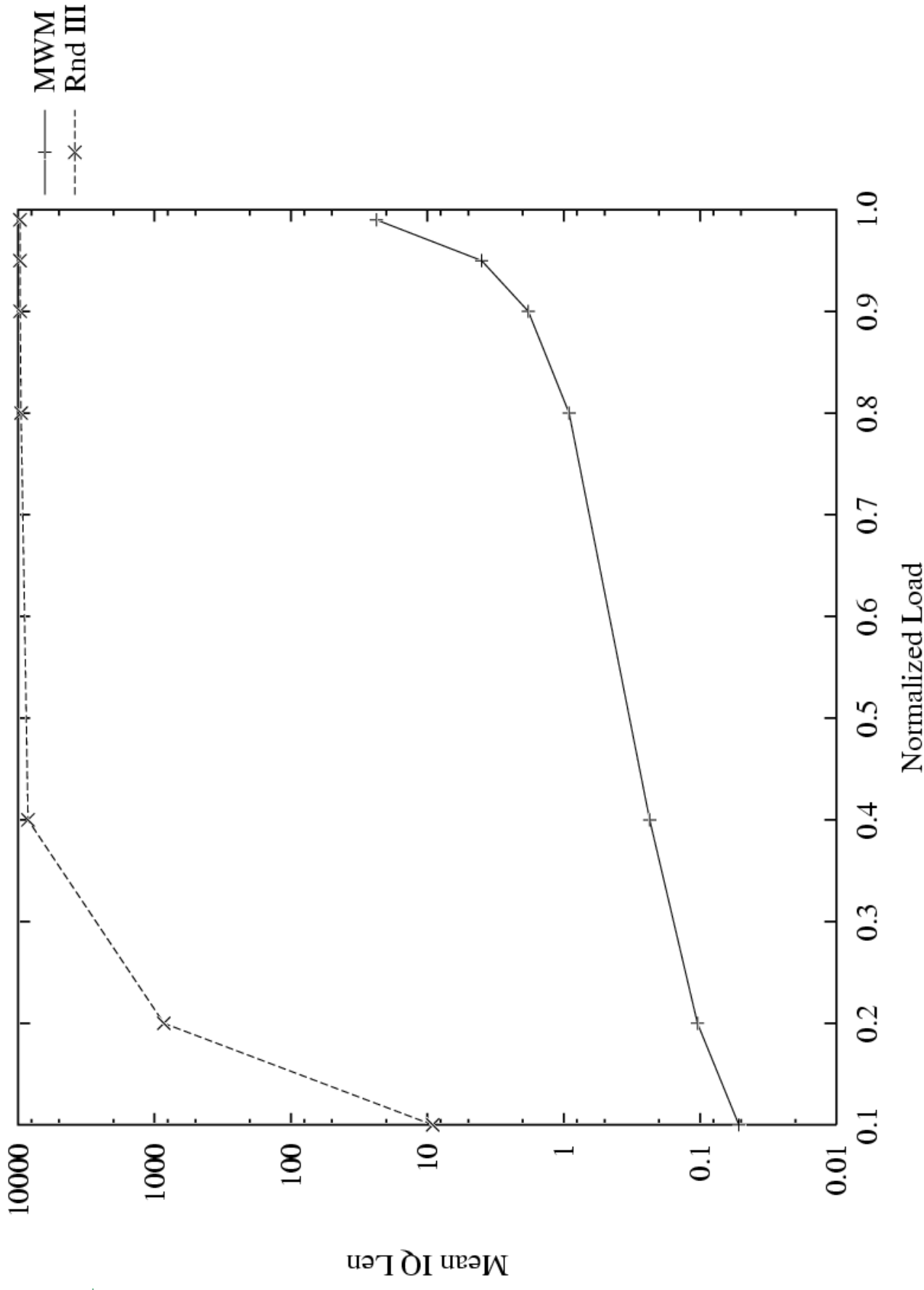
# An illustrative example

- Problem: find the largest element of a set  $S$  of size 1 billion at time  $t$
- The **randomized+memory** algorithm: find the largest among 9 randomly chosen samples and the maximum found at time  $t-1$ 
  - has a complexity of  $10$
  - can find a very good approximation of  $R$  with an high probability

# The randomized approach

- RND III (Tassiulas, 98)
  - let  $S(t-1)$  the matching used at time  $t-1$
  - pick a matching  $M$  uniformly at random from all possible  $N!$  matchings
  - let  $S(t)$  be the heavier matching of the matchings  $S(t-1)$  and  $M$
- RND III gives 100% throughput under any admissible i.i.d. Bernoulli traffic patterns
  - Note: one random sample enough ( $d=1$ )!

# The randomized approach



# Our goals

- Summary of past results
  - randomization alone  $\Rightarrow$  does not give 100% throughput
  - randomization + memory  $\Rightarrow$  100% throughput and bad delays
- Our aim is to find an algorithm that:
  - achieves 100% throughput
  - has small delays
  - and it is easy to implement

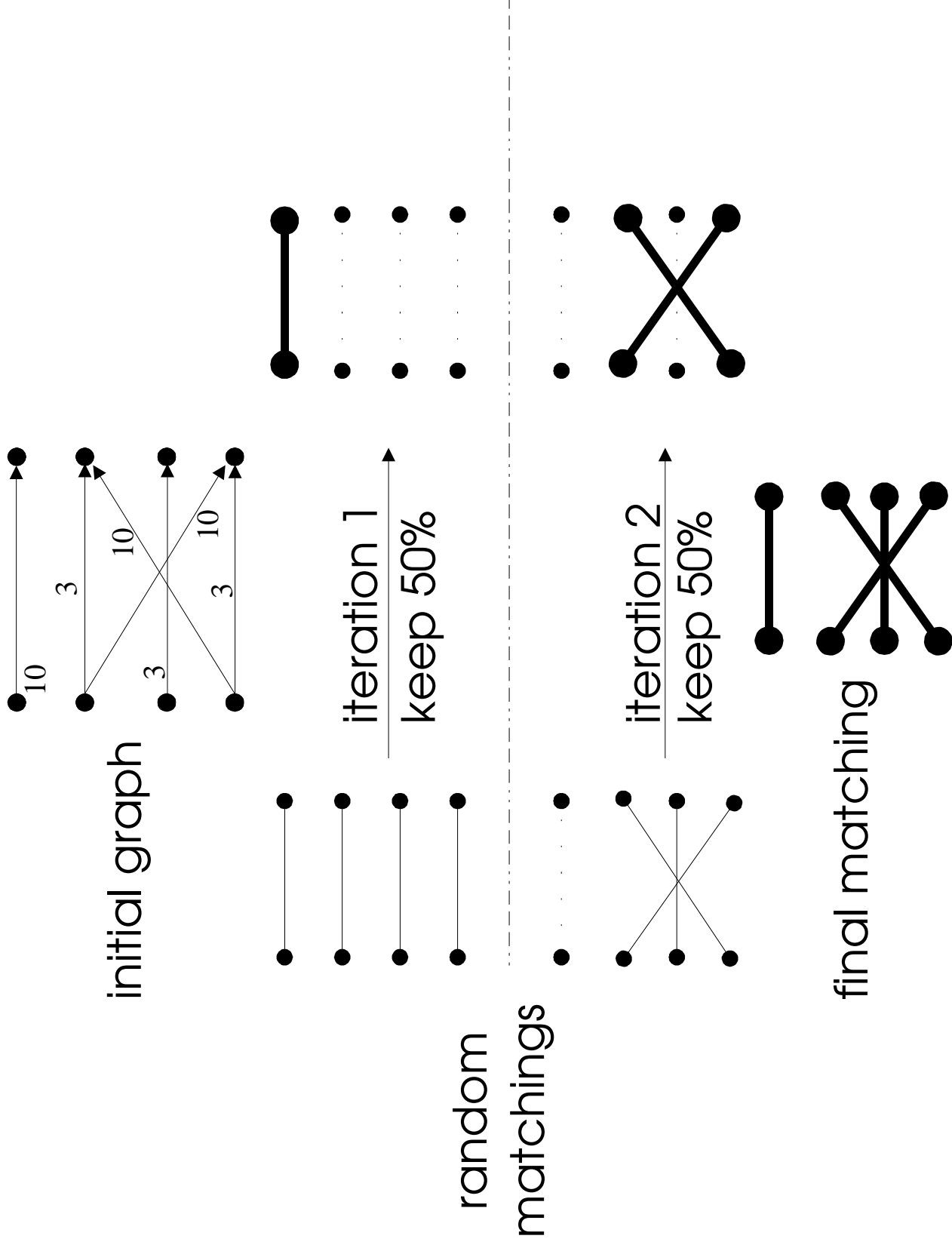
# Observations for improvement

- Most of the weight of a matching is carried in a small number of edges
  - remember edges not matchings
- The "quality" of the past matching can be improved using a "genetic approach"

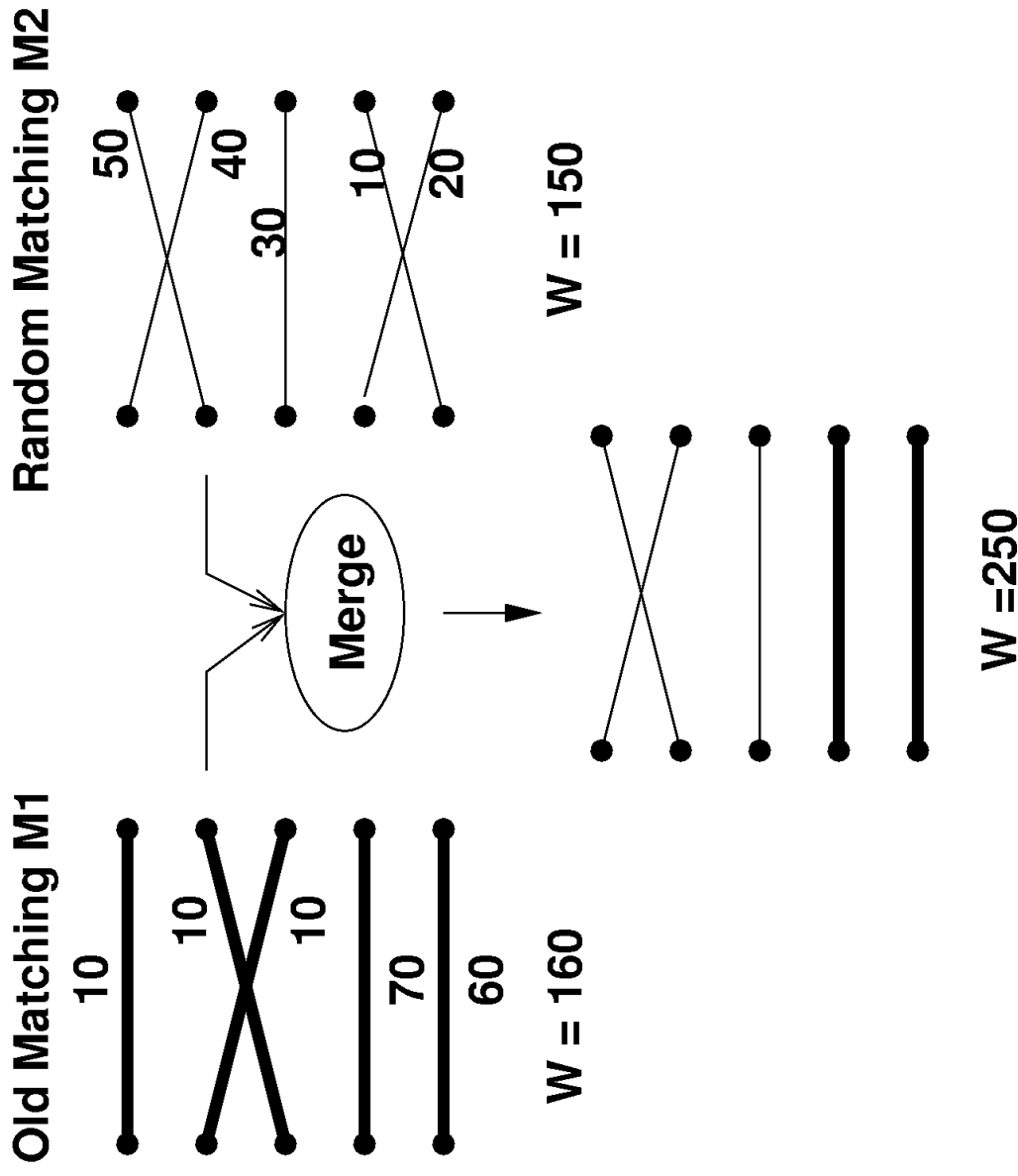
# LAURA

- How the algorithm works
  - store the past matching,  $S(t-1)$
  - obtain "good" random matching  $M$
  - *merge*  $M$  with  $S(t-1)$ 
    - by taking the best edges from each

# LAURA: RANDOM sampling



# LAURA: the MERGE procedure

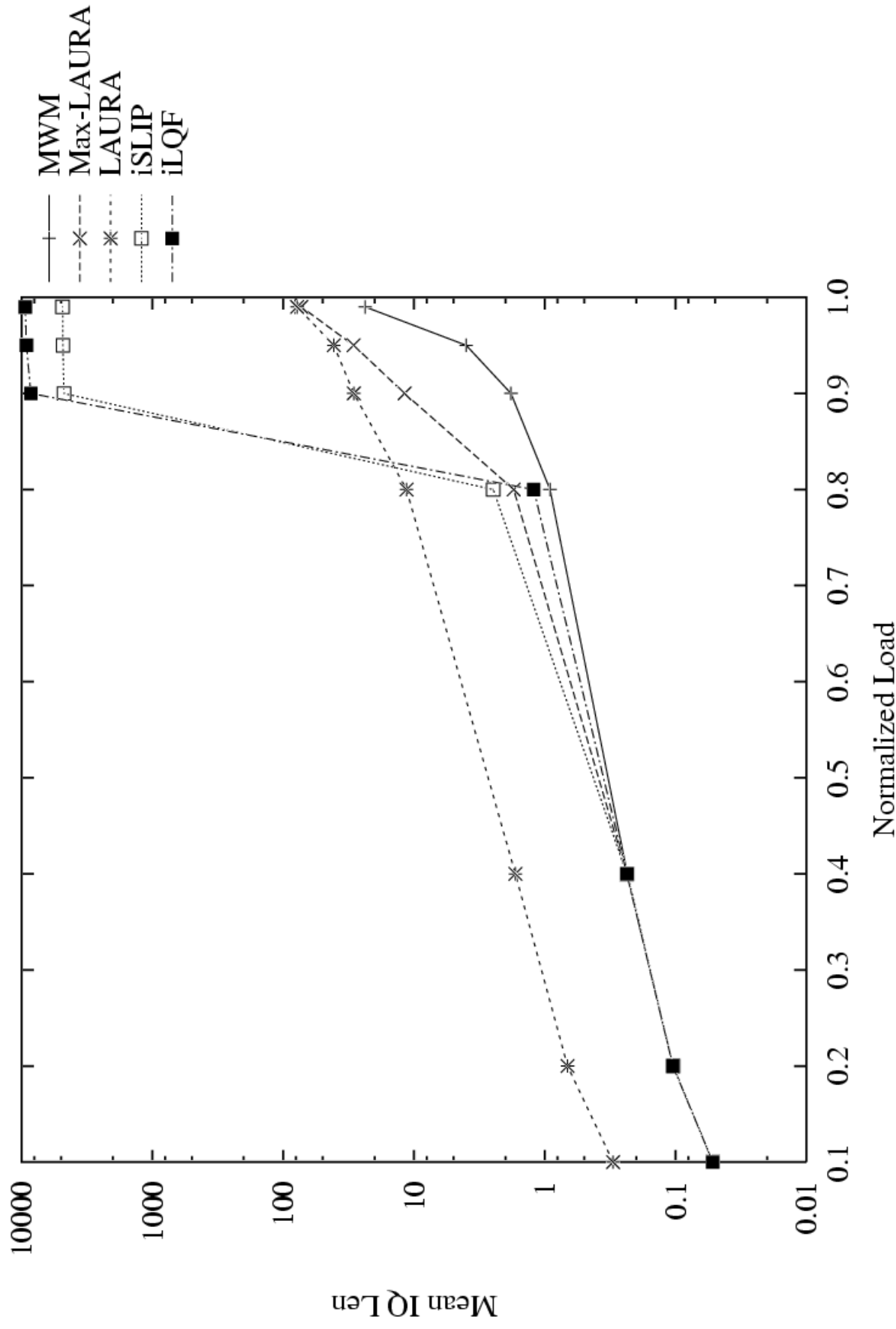


# LAURA stability

Theorem:

- LAURA achieves 100% throughput under any admissible i.i.d. Bernoulli traffic patterns

# LAURA performance



# SERENA

(A Self Randomized Algorithm that Exploits New Arrivals)

Main idea:

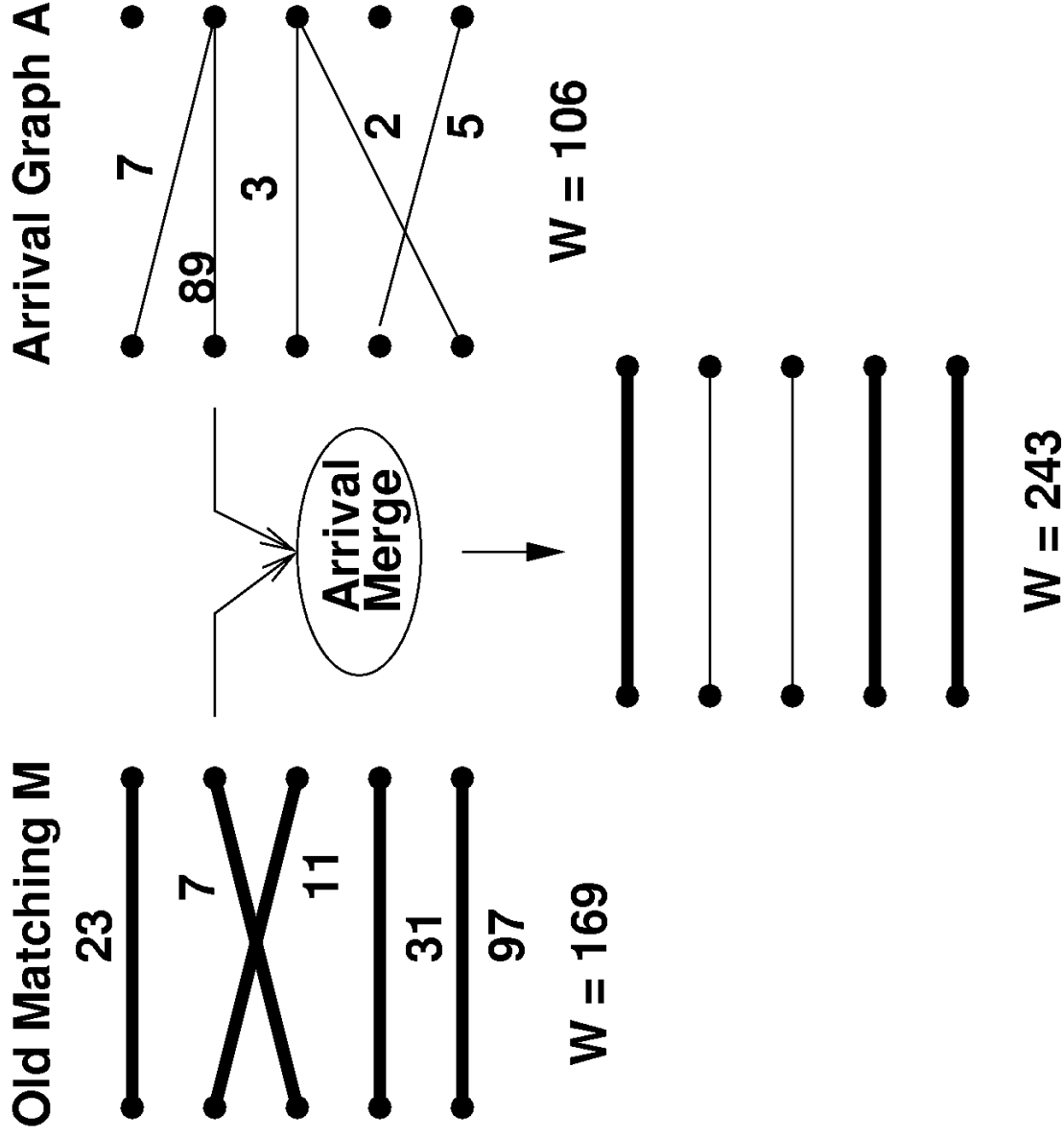
- SERENA exploits the **information in arrivals**
  - arrivals cause queues to build up: therefore, watch them
  - also, the system can "track" rapid changes in traffic by observing arrivals

# SERENA stability

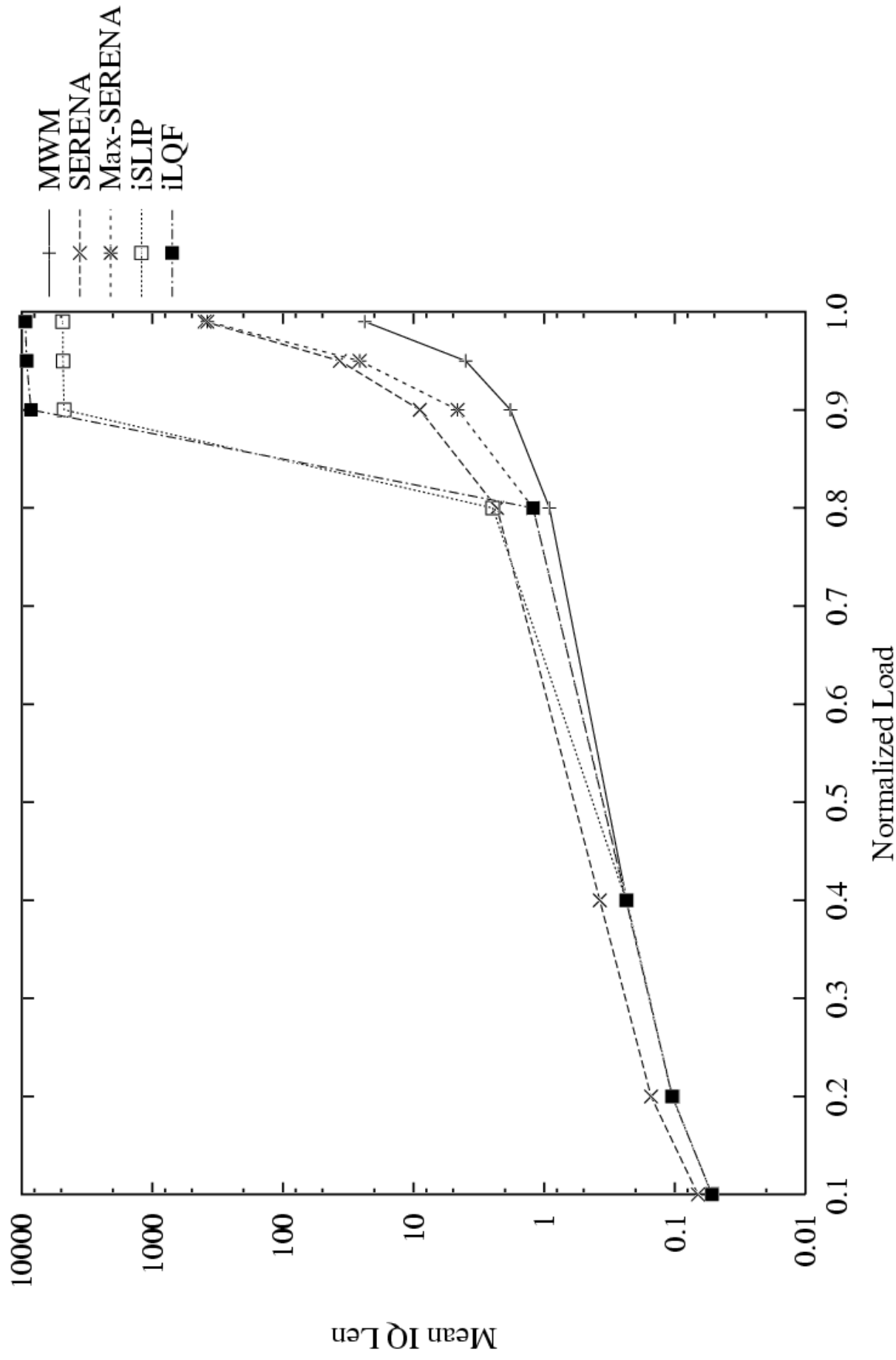
Theorem:

- SERENA achieves 100% throughput under any admissible i.i.d. Bernoulli traffic pattern

# SERENA: the arrival merge procedure



# SERENA performance



# Conclusions

- Design approach for schedulers
  - use of **memory**
  - **genetic merging** procedure
  - **information in arrivals**
- Performance:
  - 100% throughput
  - very competitive delays with respect to MWM
  - robust to changes in input traffic