Load Shedding in a Data Stream Manager

Nesime Tatbul, Uğur Çetintemel, Stan Zdonik
Brown University

Mitch Cherniack
Brandeis University

Michael Stonebraker
M.I.T.

The Overload Problem

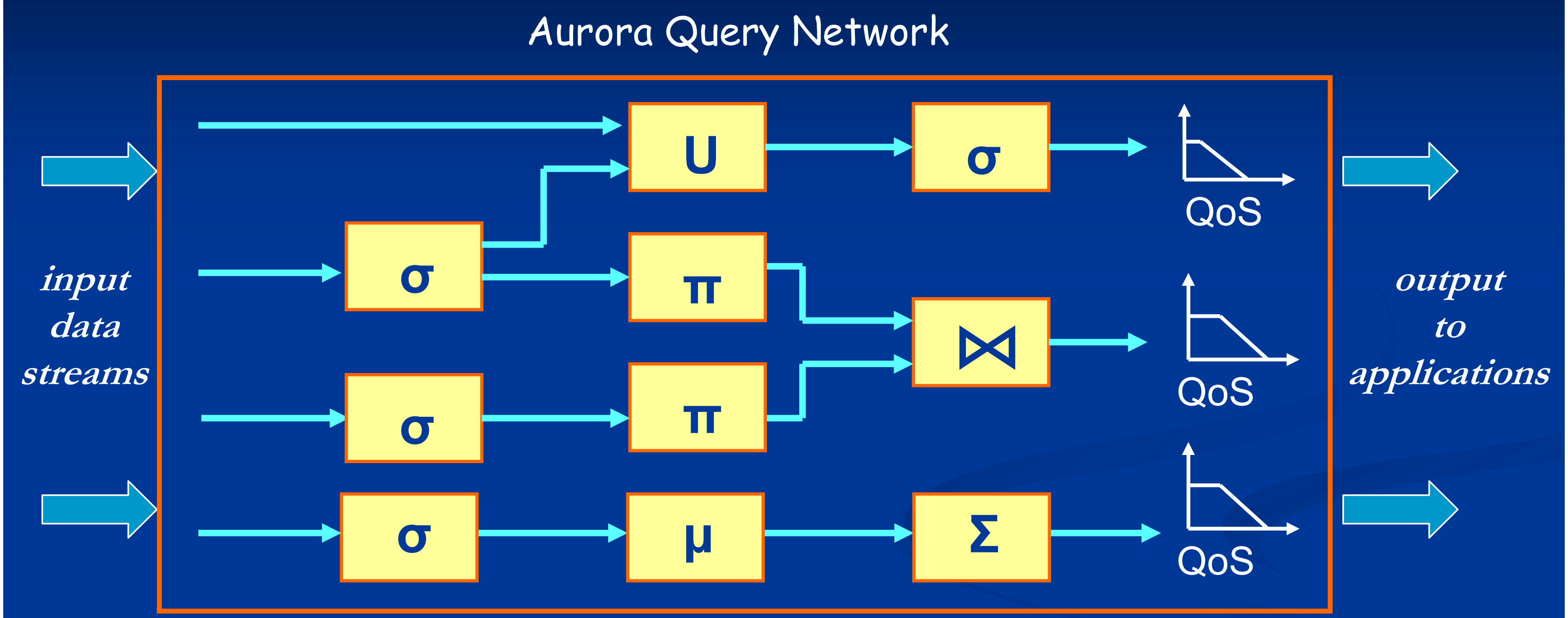
- Push-based data sources
- High and unpredictable data rates
- Problem:

  Load > Capacity

  during spikes

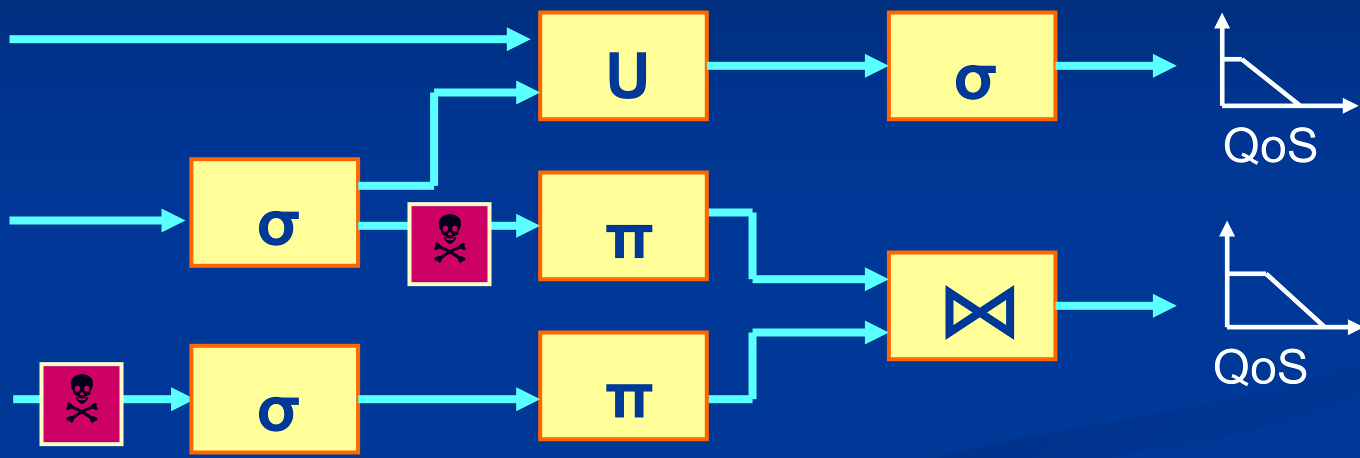
"Load Shedding": eliminating excess load by <u>dropping data</u>

Aurora Data Stream Management System

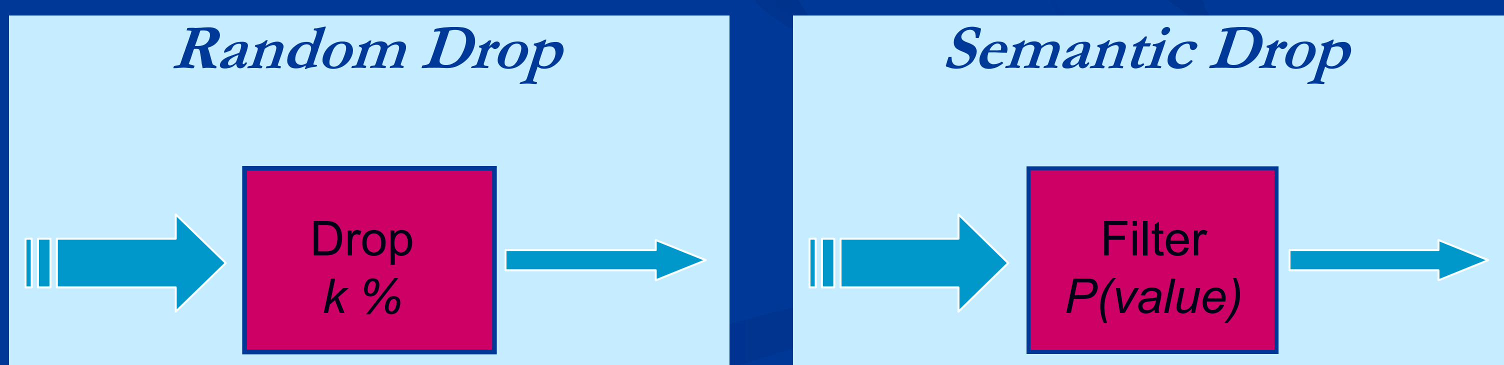


- **Stream Query Algebra (SQuAl)**
- Quality of Service
- Run-time Operation:
  - Operator Scheduling
  - Storage Management
  - Performance Monitoring
  - Load Shedding

Load Shedding by Inserting Drops

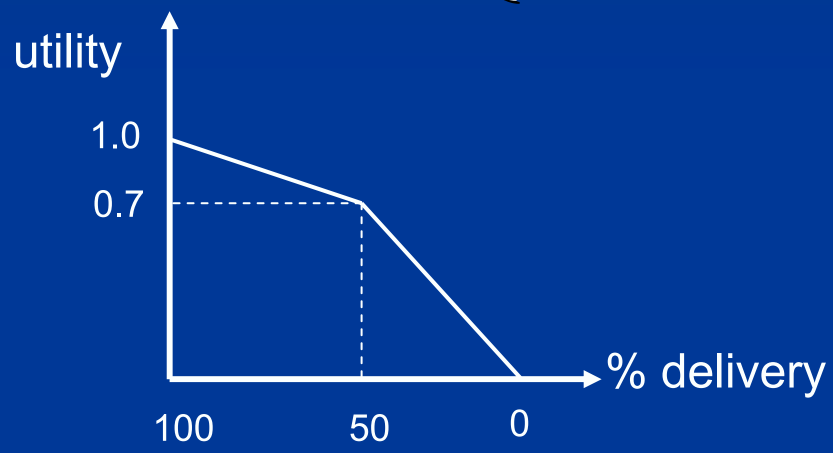


two types of drops:

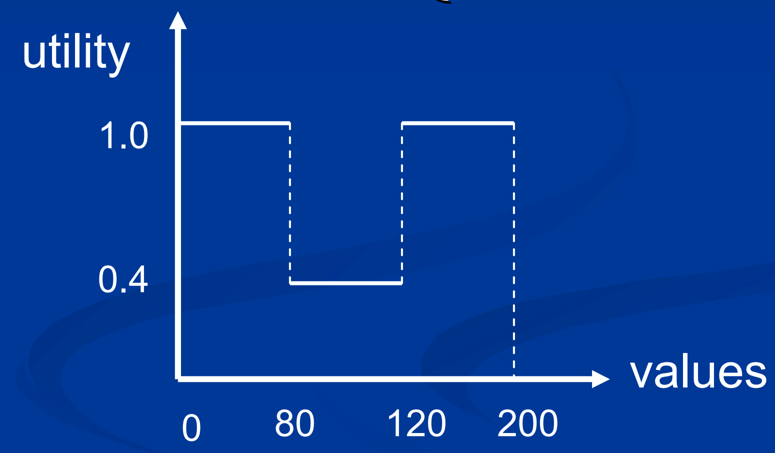


Quality of Service

- Loss-tolerance QoS



- Value-based QoS



Assumptions

- Processor is the main scarce resource
- Operators that produce new attribute values are not considered (e.g., aggregate, map)
- Load Shedder operates independently from Scheduler

Problem Statement

N: query network

I : set of input streams

C : processing capacity

when $Load(N(I)) > C$, transform N to N' such that

- $Load(N'(I)) < C$
- $Utility(N(I))-Utility(N'(I))$ is minimized

key questions:

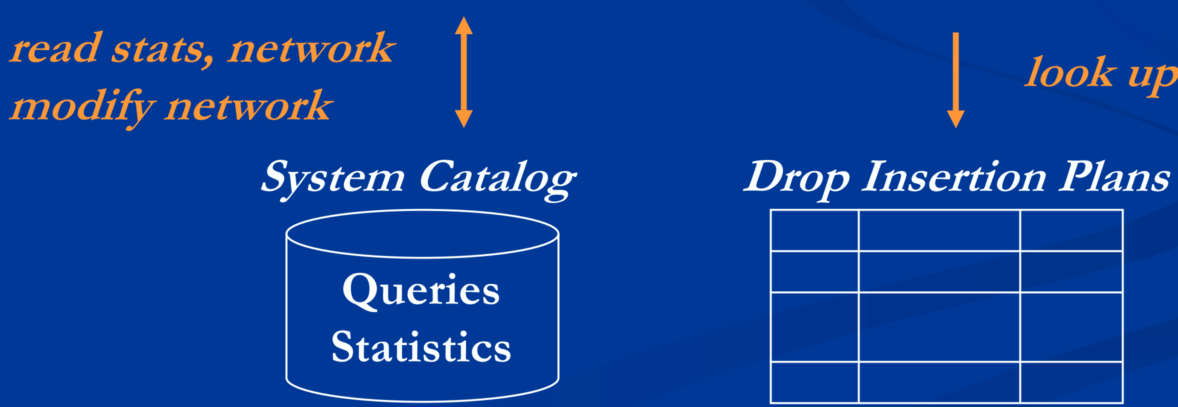
- when to shed load?
- where to shed load?
- how much load to shed?
- which tuples to drop?

Talk Outline

- Introduction
- Technical Overview
- Experimental Results
- Related Work
- Conclusions and Future Work

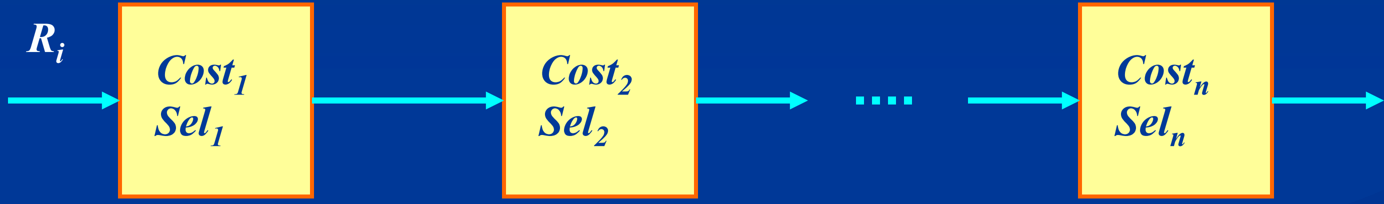
Algorithmic Overview

```
evaluate the Query Network Load

IF Load > Capacity
    insert Drops to the Query Network

ELSE IF Load < Capacity
     AND Drops in the Query Network
    remove Drops from the Query Network
```



Load Evaluation

- "Load Coefficients" for each input stream



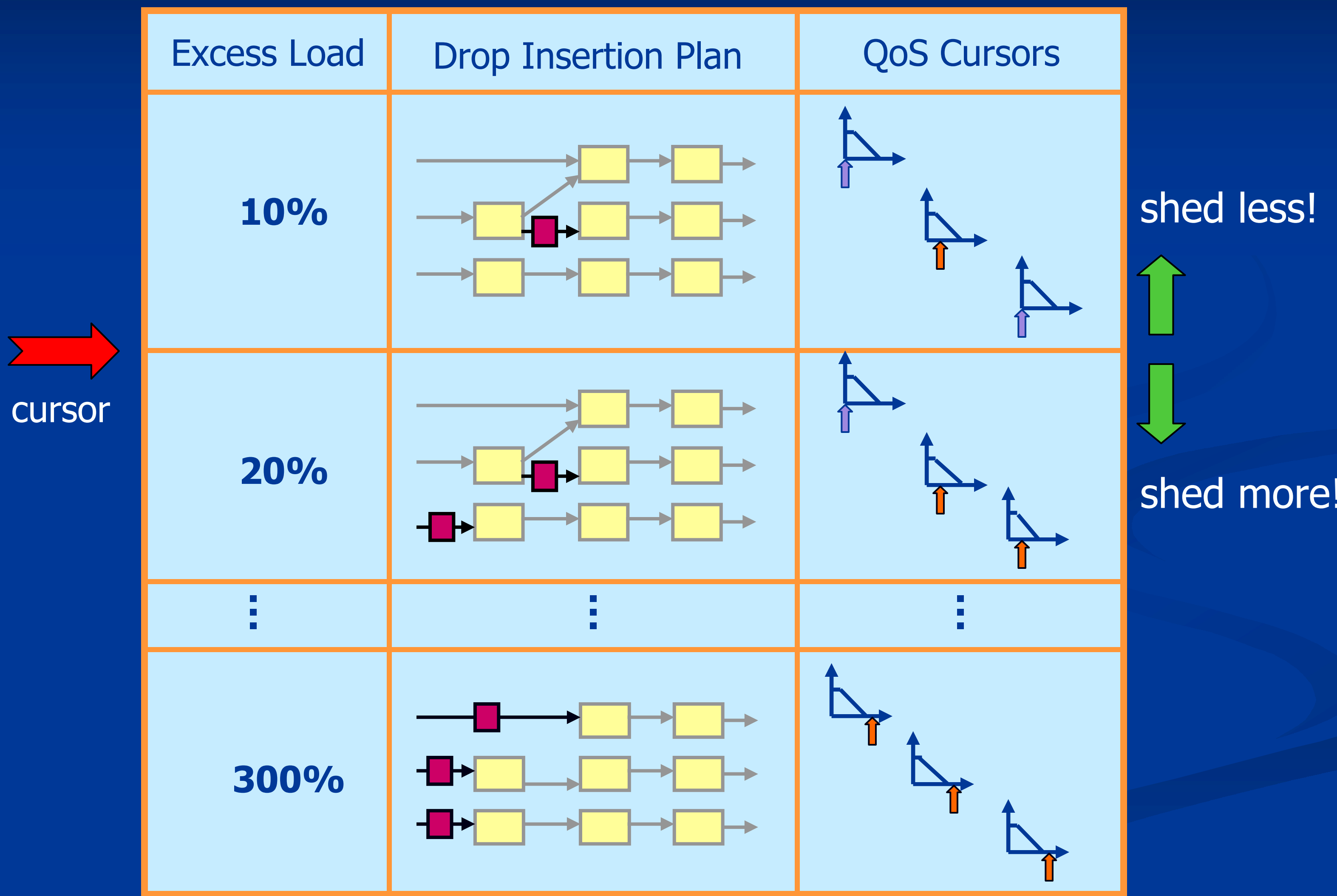
$$L_i = \sum_{j=1}^{n} \left(\prod_{k=1}^{j-1} Sel_k \right) \times Cost_j$$ (CPU cycles per tuple)

can be pre-computed!

- Total Load at run time

$$\sum_{i=1}^{m} L_i \times R_i$$ (CPU cycles per time unit)

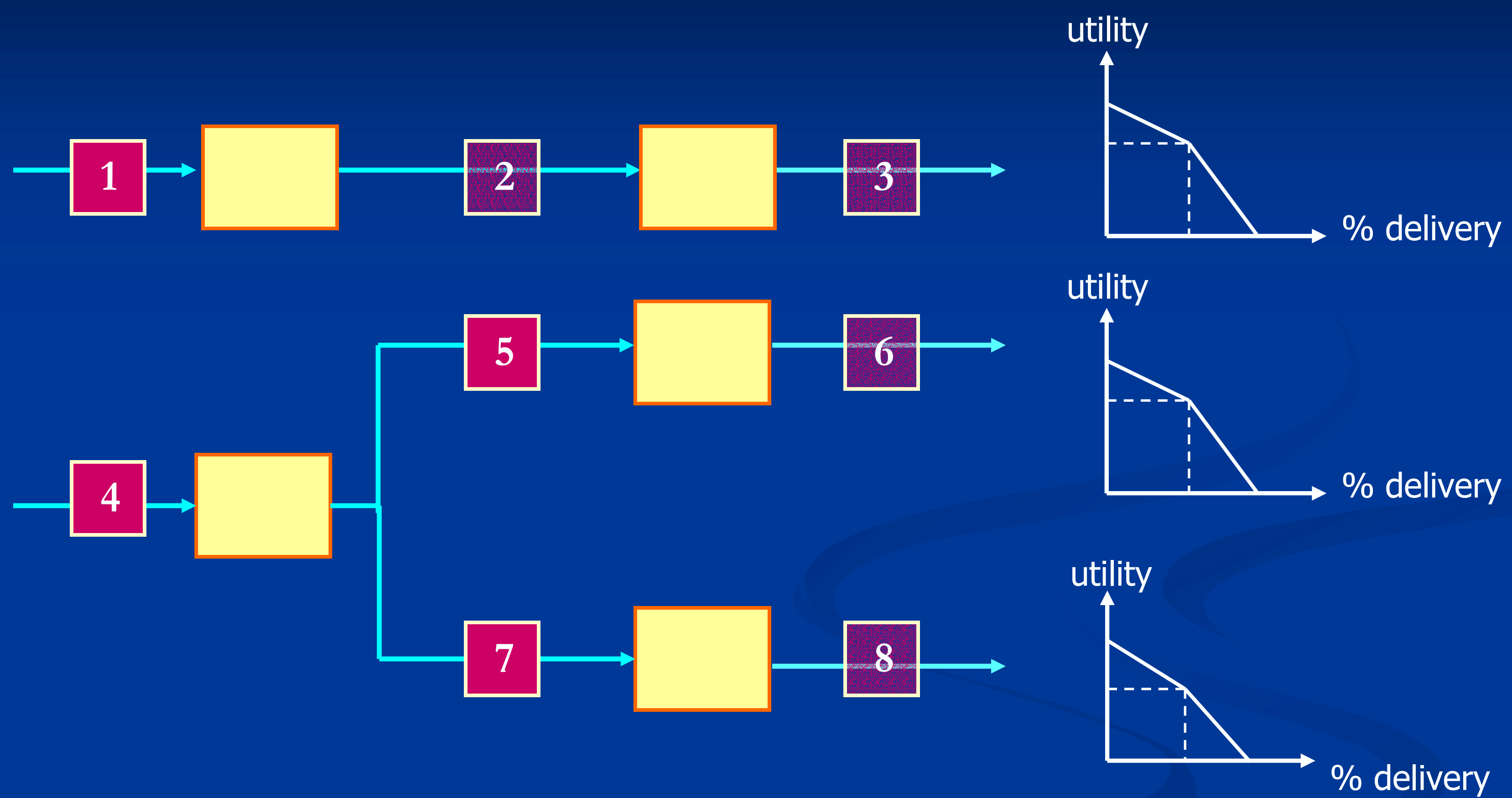
Load Shedding Road Map (LSRM)



Constructing the LSRM

1. identify potential "drop locations"
2. sort the drop locations
3. apply the drop locations in order:
   – *insert* one unit of drop
   – if semantic drop, determine the filter predicate
   – create an LSRM entry

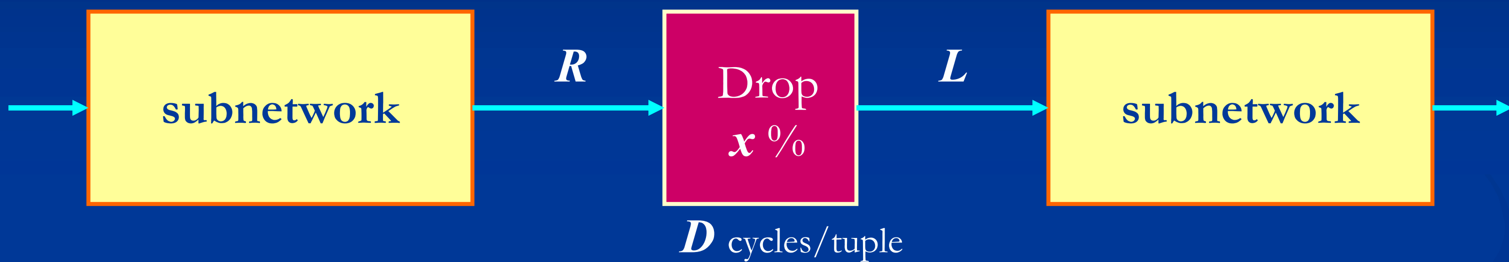
Drop Locations



- early drops save more cycles
- drops before splits cause more utility loss

Best Drop Location

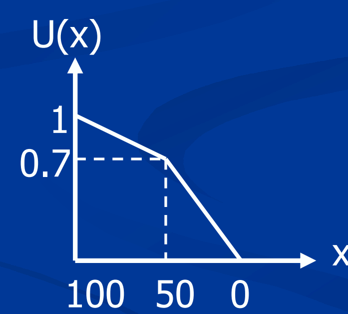
- Goal: maximize cycle gain, minimize utility loss



Cycle gain:

$$G(x) = \begin{cases} R \times (x \times L - D) & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Utility loss:



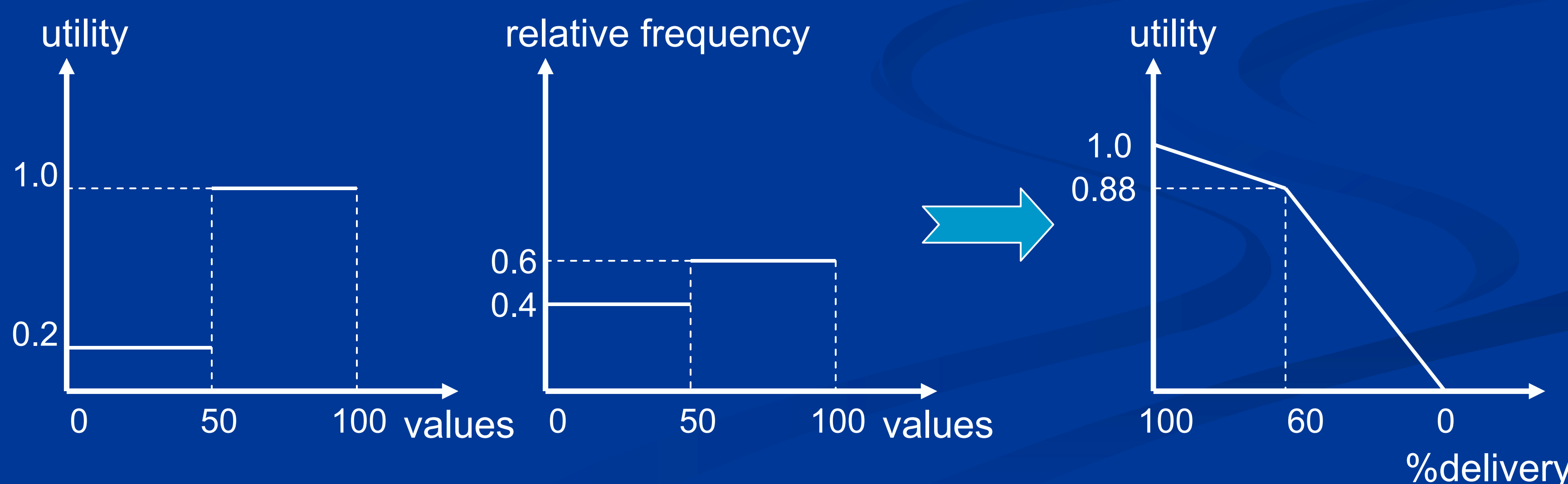
- Loss/Gain Ratio

$$\frac{-dU(x)/dx}{dG(x)/dx} = \frac{\text{negative slope of } U(x)}{R \times L}$$

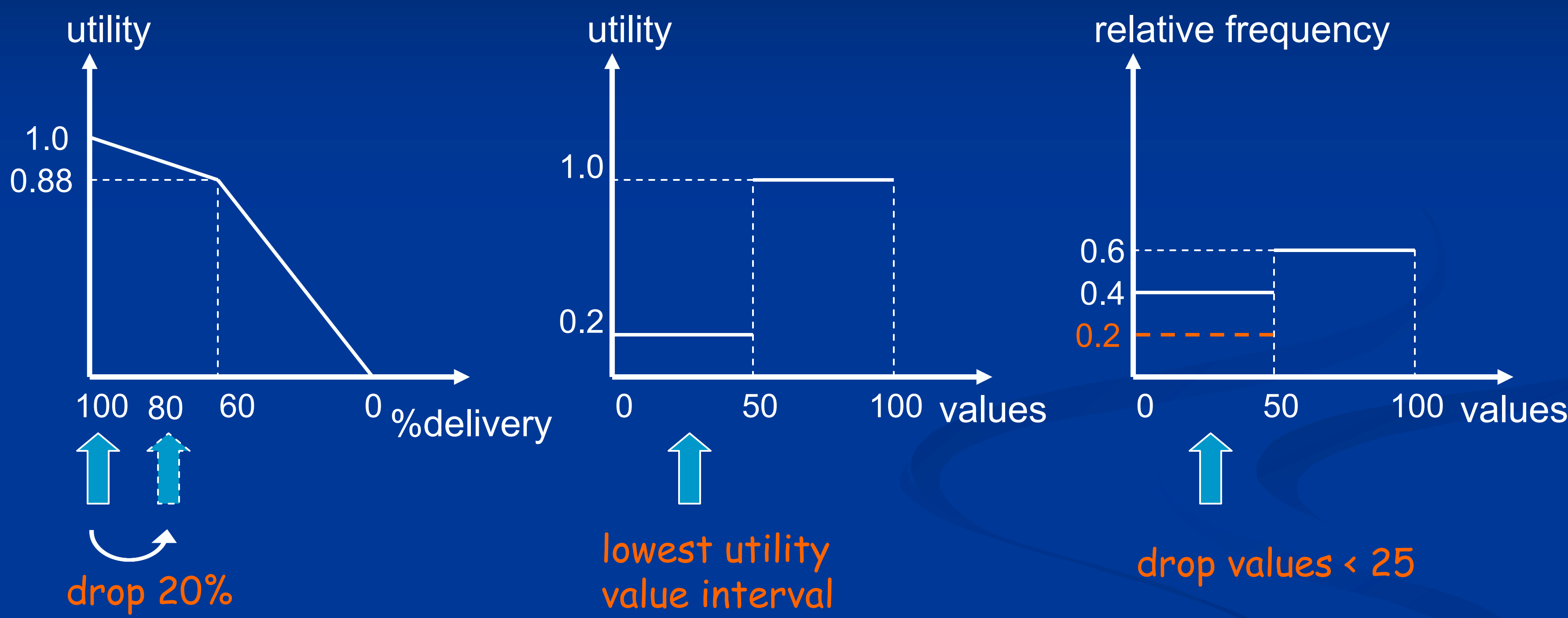
the smaller,
the better!

From Values to % Delivery

- "when, where, how much?" have the same answer for Random and Semantic Load Shedding
- <u>Trick:</u> translation between QoS functions
- <u>Assumption:</u> drop the lowest utility values first



Determining the Filter Predicate



Filter Predicate:

***value* ≥ 25**

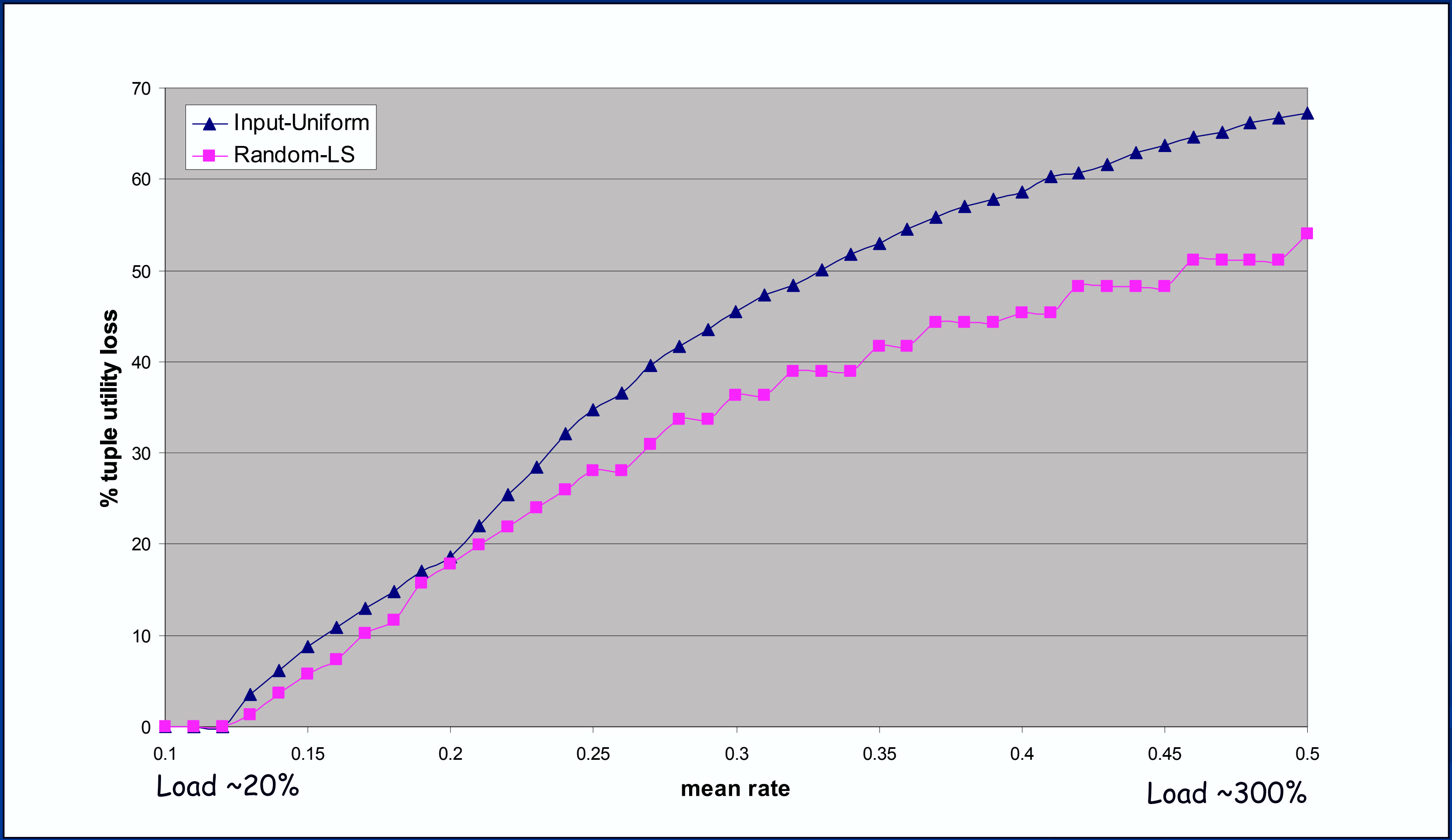
Talk Outline

- Introduction
- Technical Overview
- Experimental Results
- Related Work
- Conclusions and Future Work

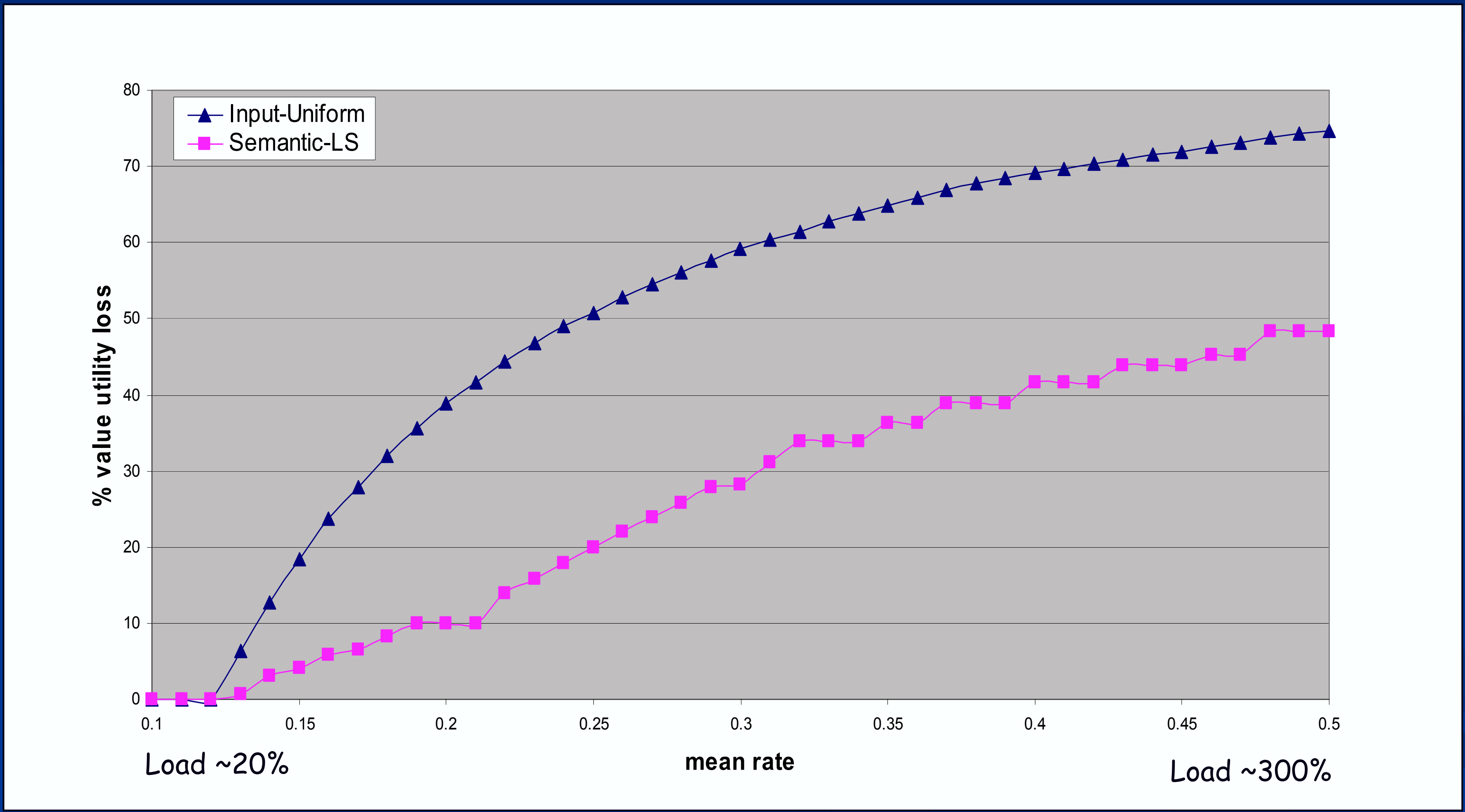
Experimental Study

- Setup:
  - streams: uniformly distributed integer values
  - networks: a mix of filters and unions
  - QoS: value-based QoS, range utilities chosen from a Zipf distribution
- Metrics used:
  - percent loss in total average utility on
    - loss-tolerance QoS (*Tuple Utility Loss*)
    - value-based QoS (*Value Utility Loss*)

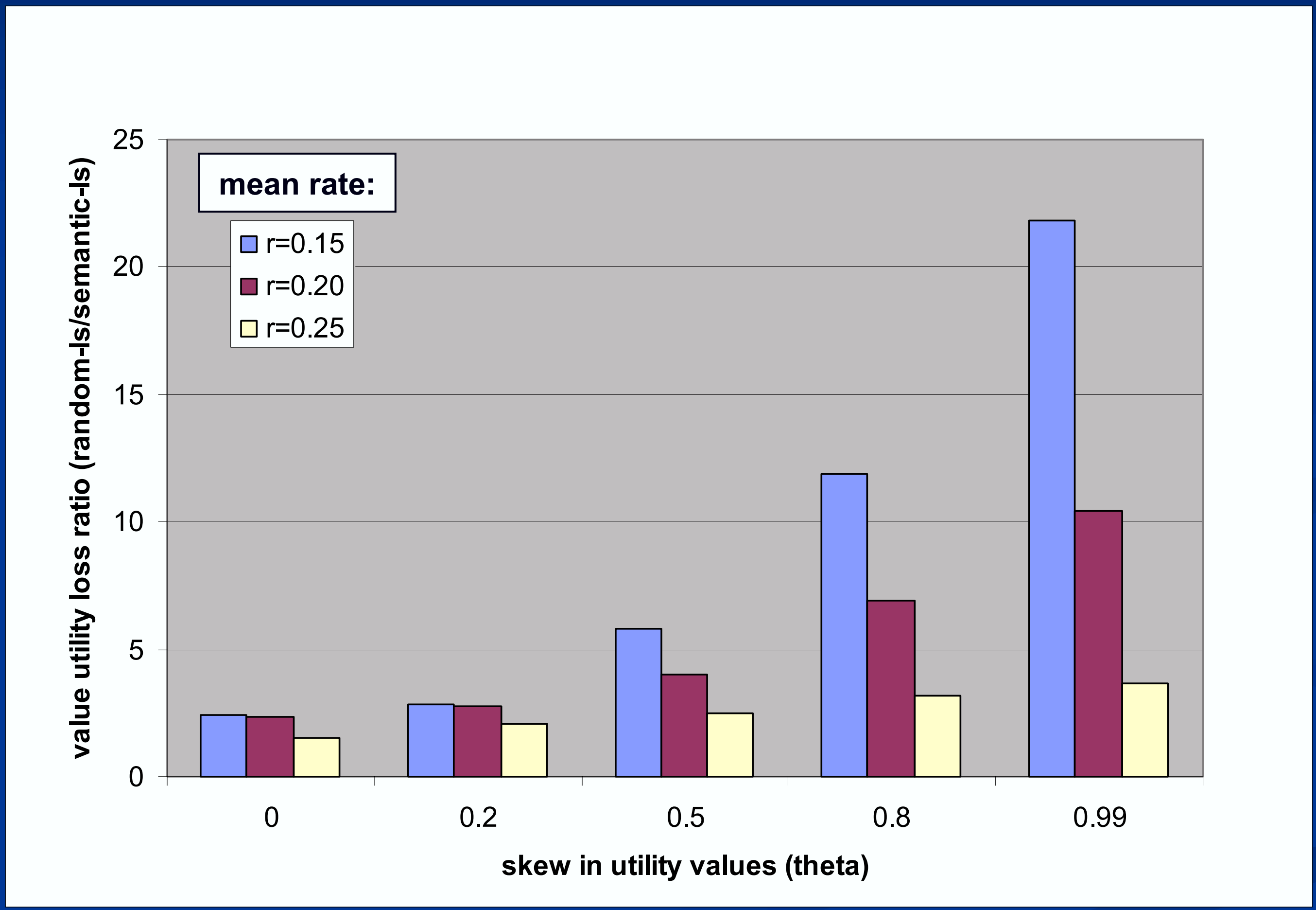
Random-LS beats Admission Control



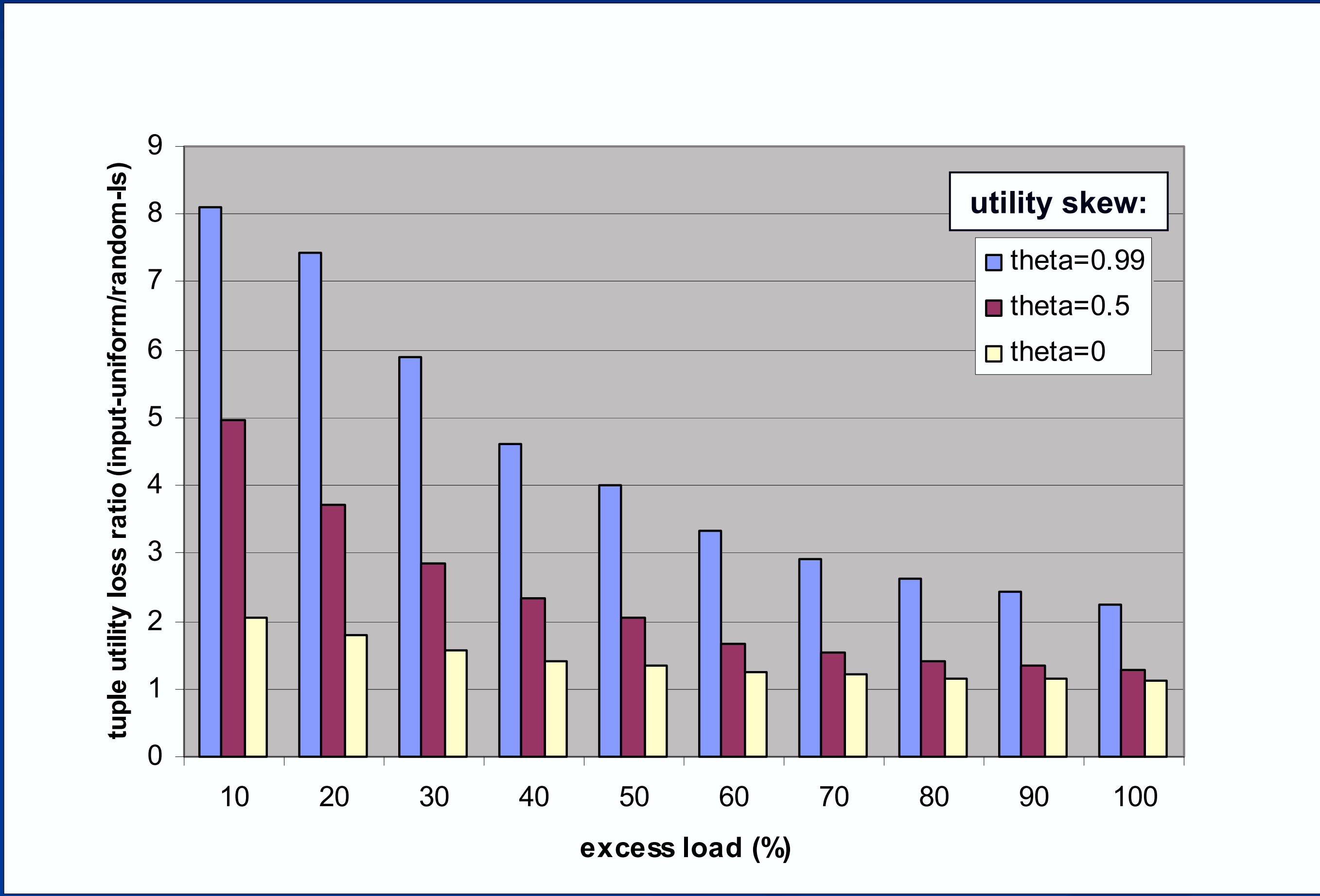
Semantic-LS beats Admission Control



Semantic-LS beats Random-LS



Performance for Networks with Sharing



Talk Outline

- Introduction
- Technical Overview
- Experimental Results
- Related Work
- Conclusions and Future Work

Relevance to Existing Work

- Congestion control in networks
- Multimedia streaming
- Approximate query answering
- Data stream processing
  - sampling, shedding on aggregates STREAM [MW+03, BDM03]
  - approximate join processing [DGR03]
  - adjusting rates to windowed joins [KNV03]

Talk Outline

- Introduction
- Technical Overview
- Experimental Results
- Related Work
- Conclusions and Future Work

Highlights

- An end-to-end solution for detecting and resolving overload in data stream systems
- Utility loss minimization for networks of queries which share resources and processing
  - → semantic utility as well as quantity-based utility
- Static drop insertion plans, dynamic instantiation

Future Directions

- Handling complex operators
  - joins and aggregates
- Other resource limitations
  - memory - windowed operators
  - bandwidth - Aurora*
  - power - at sensor level

More Information

- Aurora Web Page:
  http://www.cs.brown.edu/research/aurora
- Email:
  tatbul@cs.brown.edu
- Demo