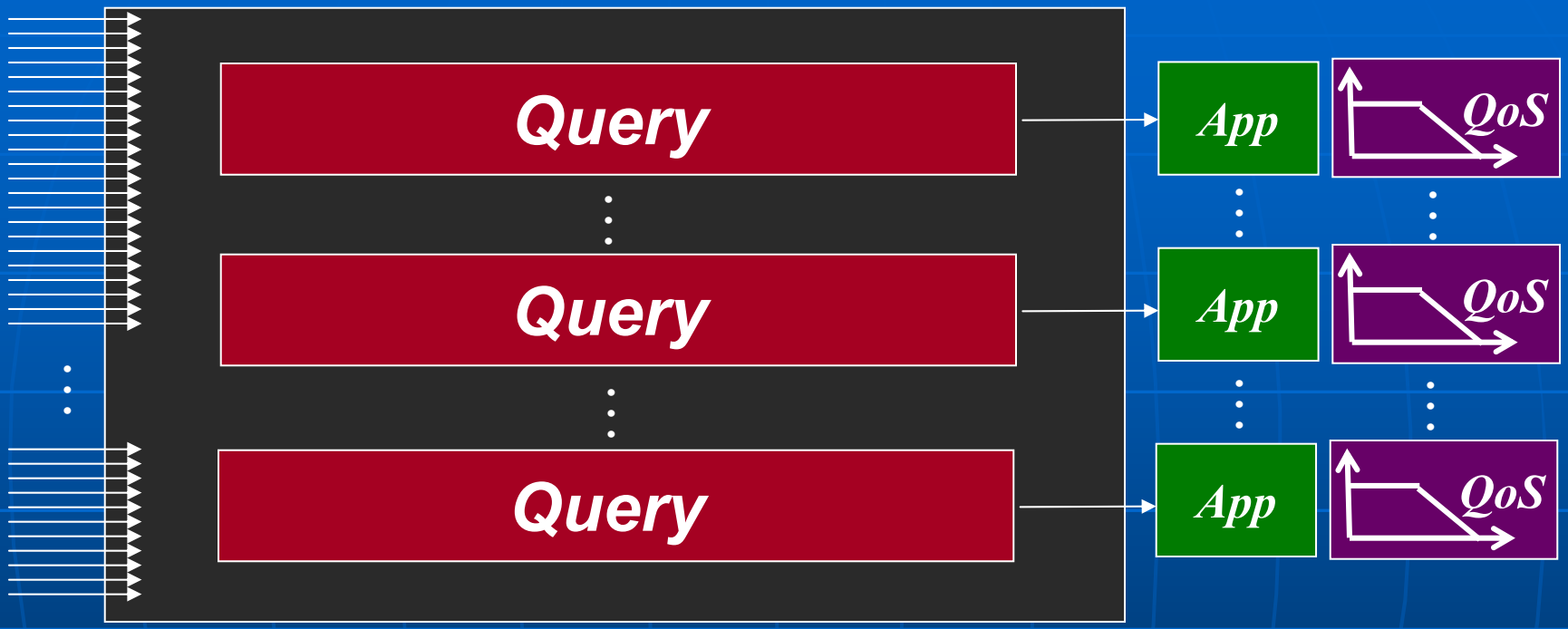
Operator Scheduling in a Data Stream Manager

Don Carney	*Brown University*
Uğur Çetintemel	*Brown University*
Alex Rasin	*Brown University*
Stan Zdonik	*Brown University*
Mitch Cherniack	*Brandeis University*
Michael Stonebraker	*MIT*

Stream-based Applications

- Examples
  - Traffic analysis
    - Streams of automobile locations
  - Market analysis
    - Streams of stock ticker data
  - Sensor monitoring
    - Streams of soldier locations
- Characteristics
  - Lots of data sources
  - Unpredictable and high rates of input
  - Latency expectations / deadlines
  - Timely & Sophisticated processing

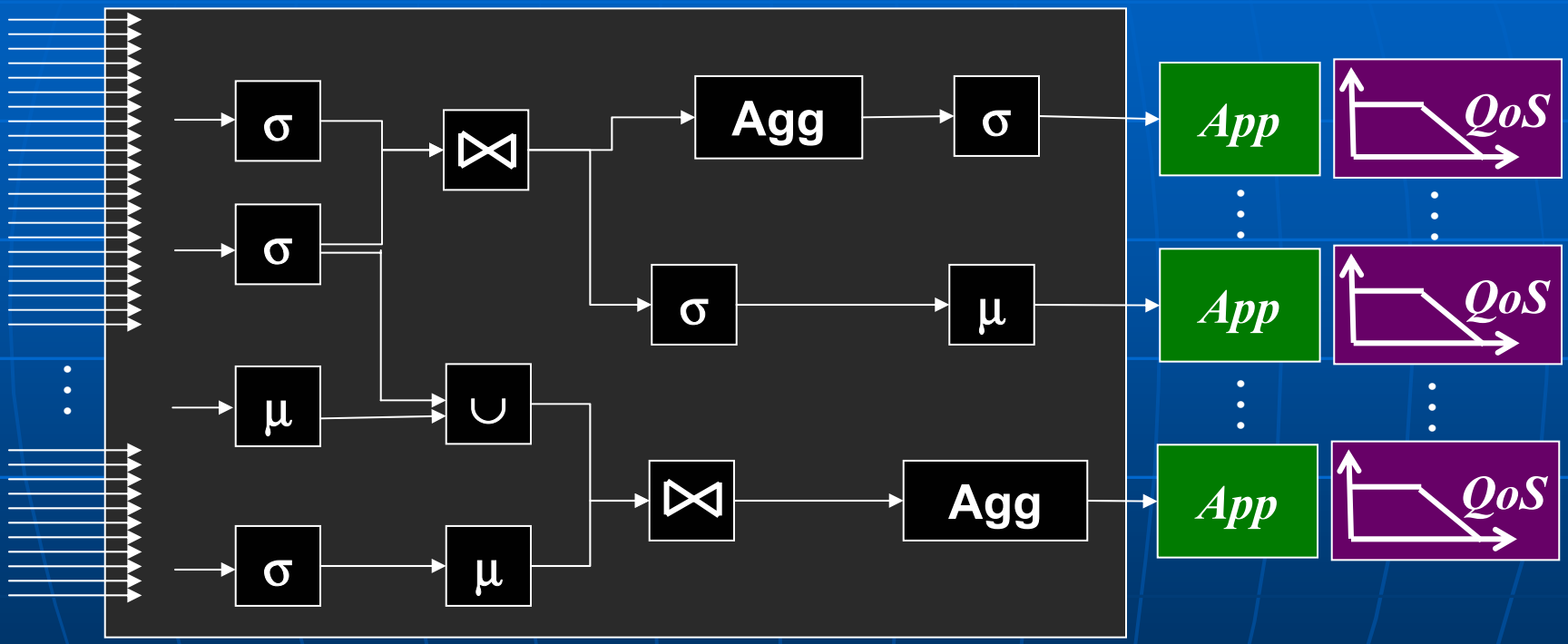
Aurora from the Sky



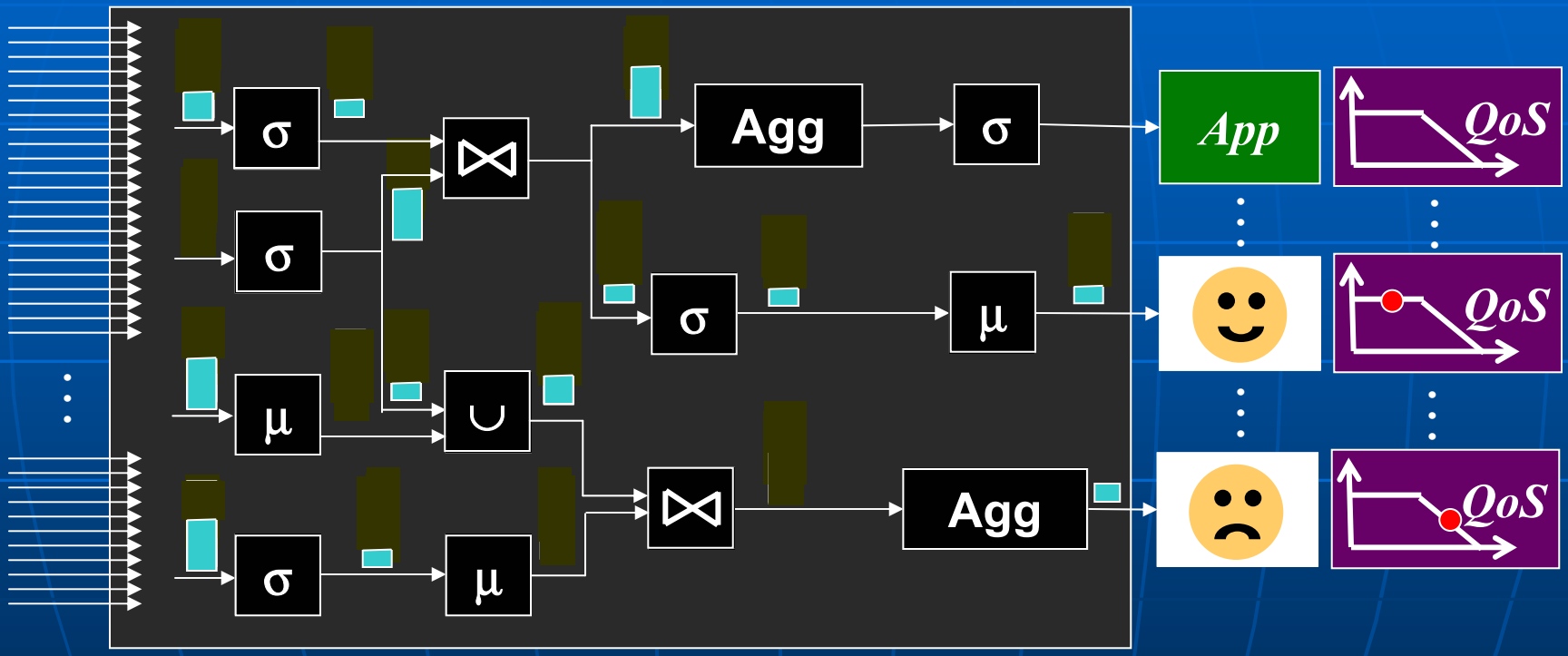
Each Application Provides:

- A *Query* over input data streams
- A Quality-Of-Service Specification (QoS)
  (specifies utility of results)

A Look Inside



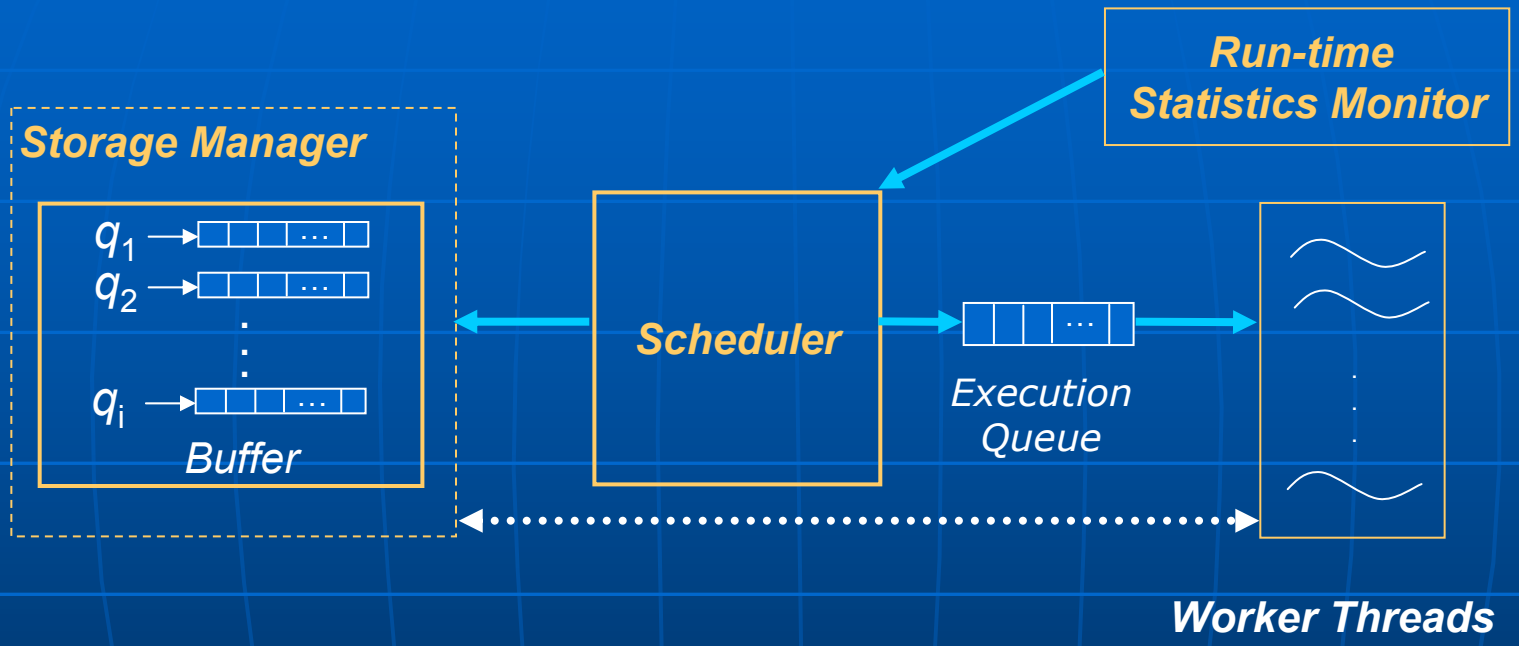
Scheduling in Action



Traditional Thread-driven Execution

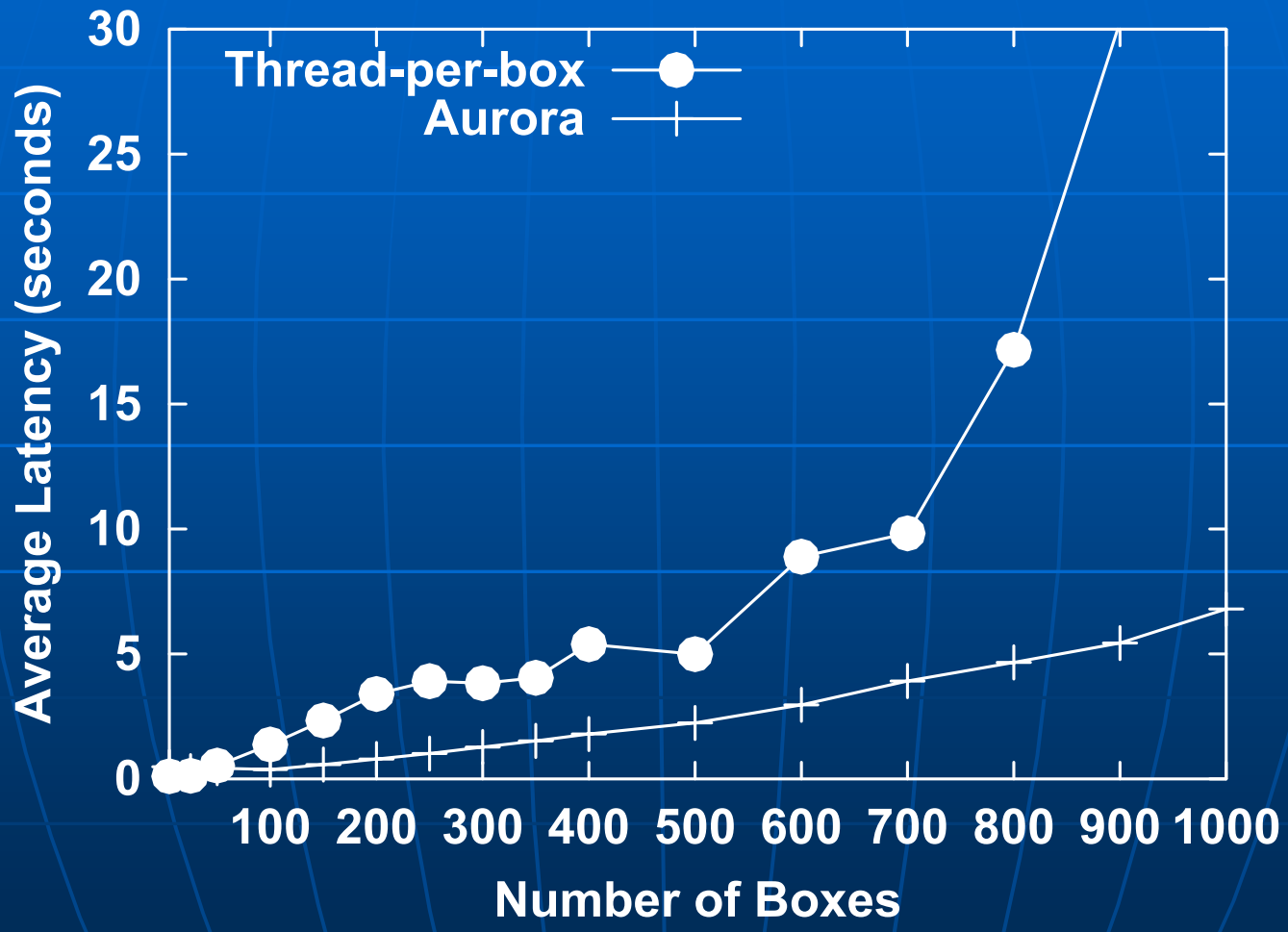
- Thread per query or operator
- Resource management done by OS
  - Easy to program
  - Problems
    - No Application specific QoS
    - Scalability

Basic Architecture



"How to make this light-weight enough to meet QoS constraints under heavy load"

Aurora vs. Thread-Based

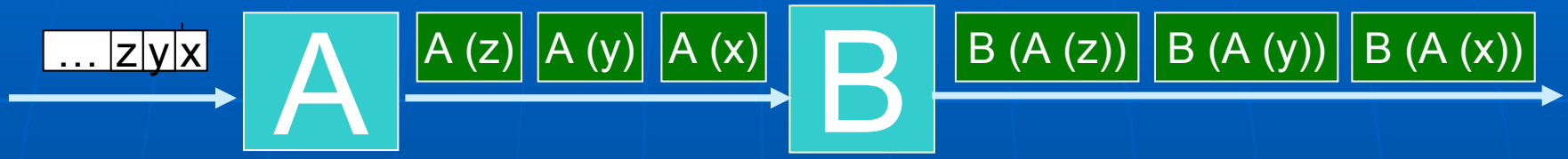


Scheduler Specifics

- Overhead reduction
- Box execution order
- Scalability

Minimizing Per Tuple Overhead

Tuple at a time:



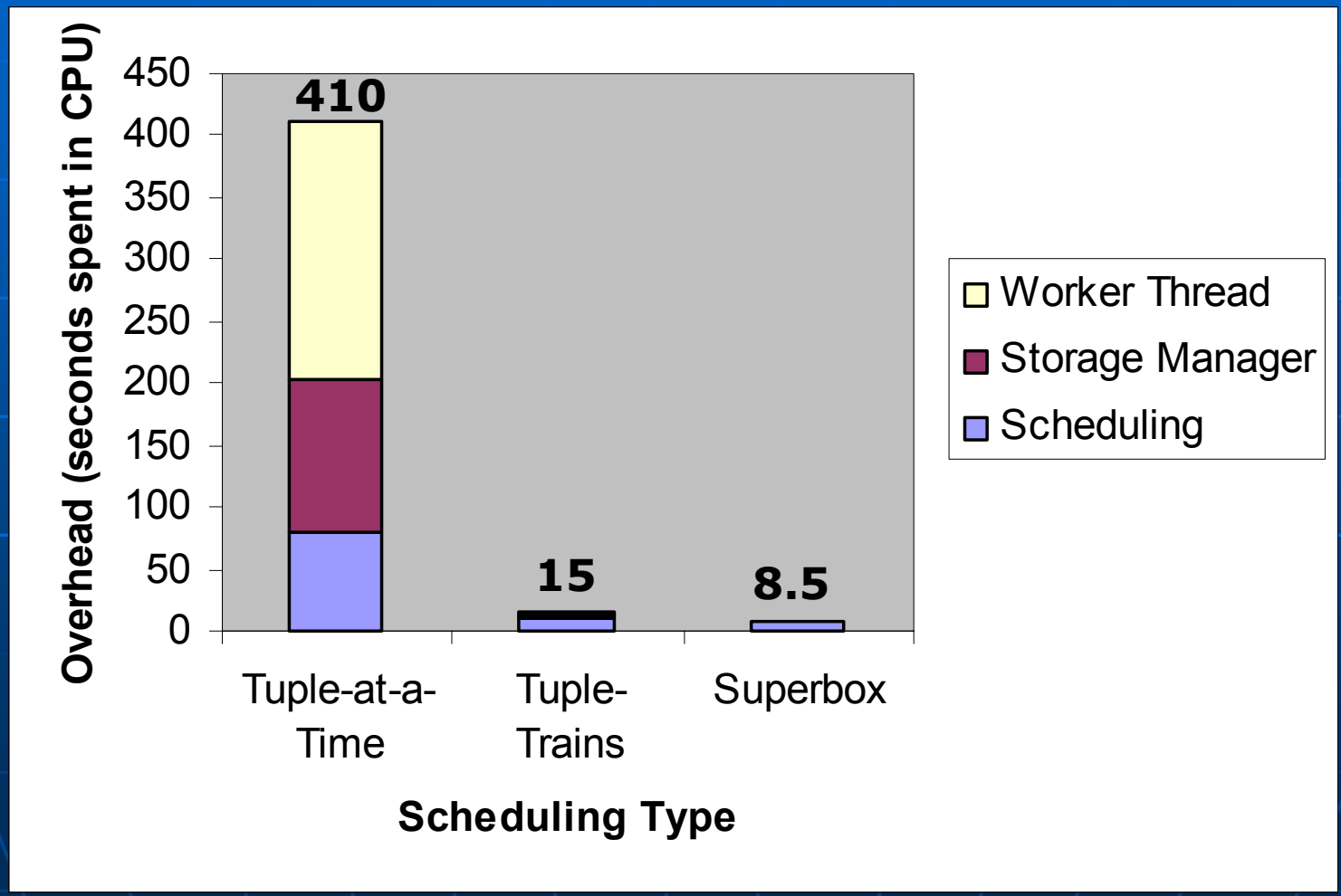
= *Scheduler Action*

Train Scheduling:

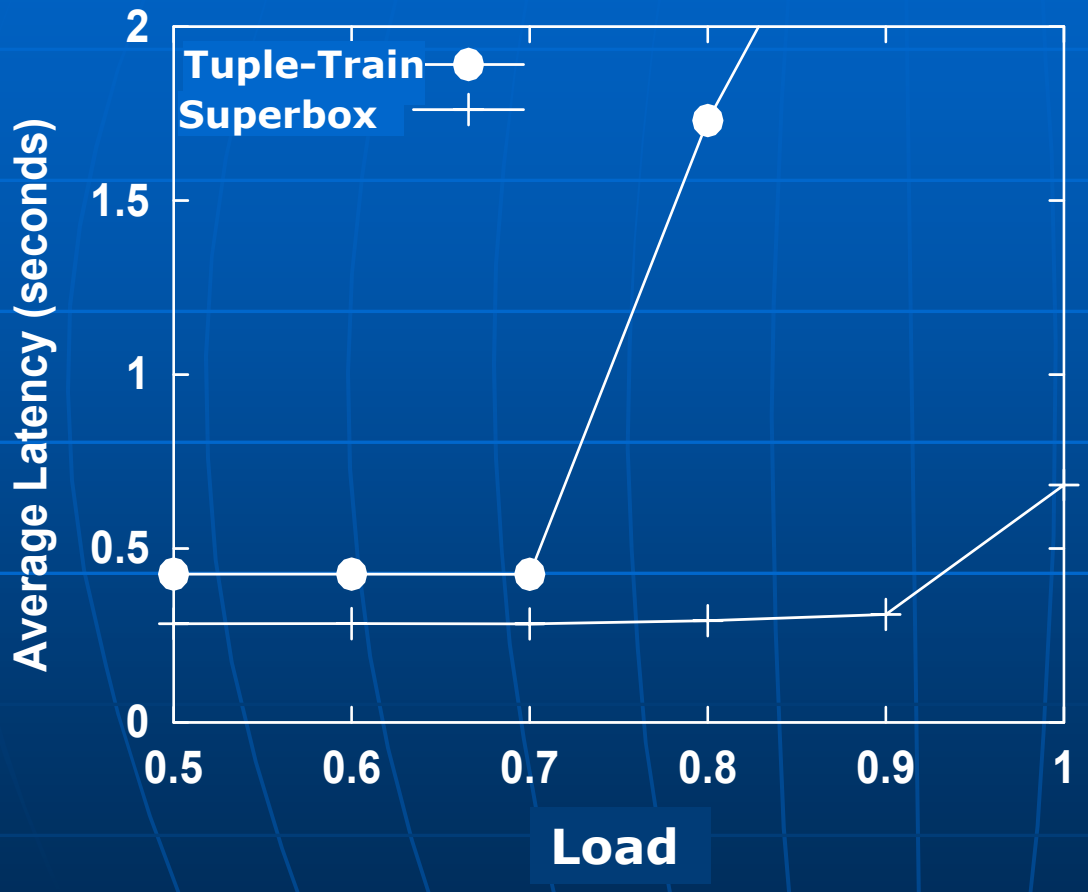
Tuple Trains:

Box & Tuple Trains (superbox):

Scheduling Superboxes incurs lowest overhead



Superboxes provide best performance

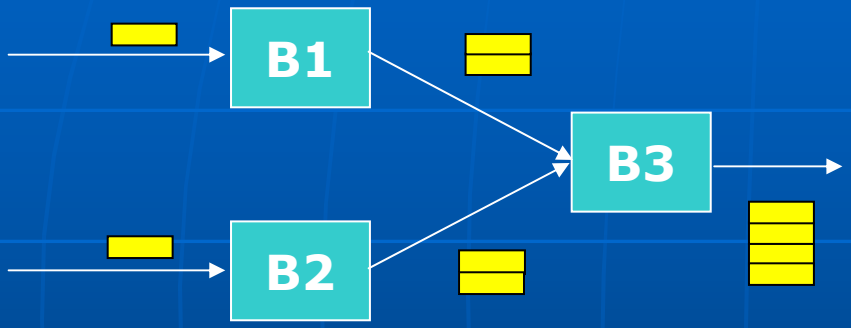


- 5 Apps
- 500 boxes

Traversal Matters

Min-Cost Traversal

B1 → B2 → B3



Processing Cost

- **Execution of Box**

Call Overhead

- **Context Switch**

Average Latency

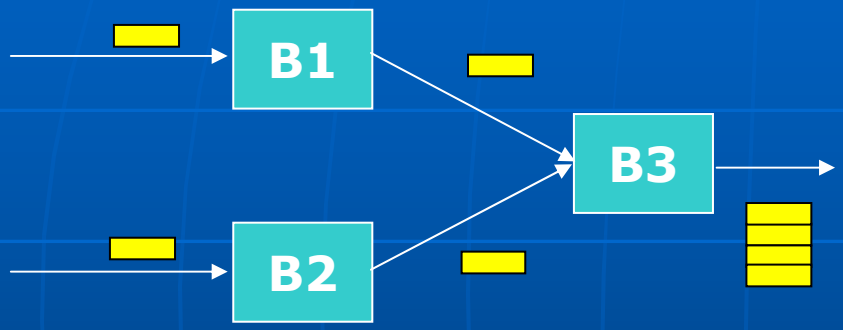
- **Measured as average of above 2**

	Processing (p)	Call Overhead (o)	Avg Latency
Min-Cost	4p	3o	3.5p + 3o

Traversal Matters

Min-Latency Traversal

B3 → B1 → B3 → B2 → B3



Processing Cost

- **Execution of Box**

Call Overhead

- **Context Switch**

Average Latency

- **Measured as average of above 2**

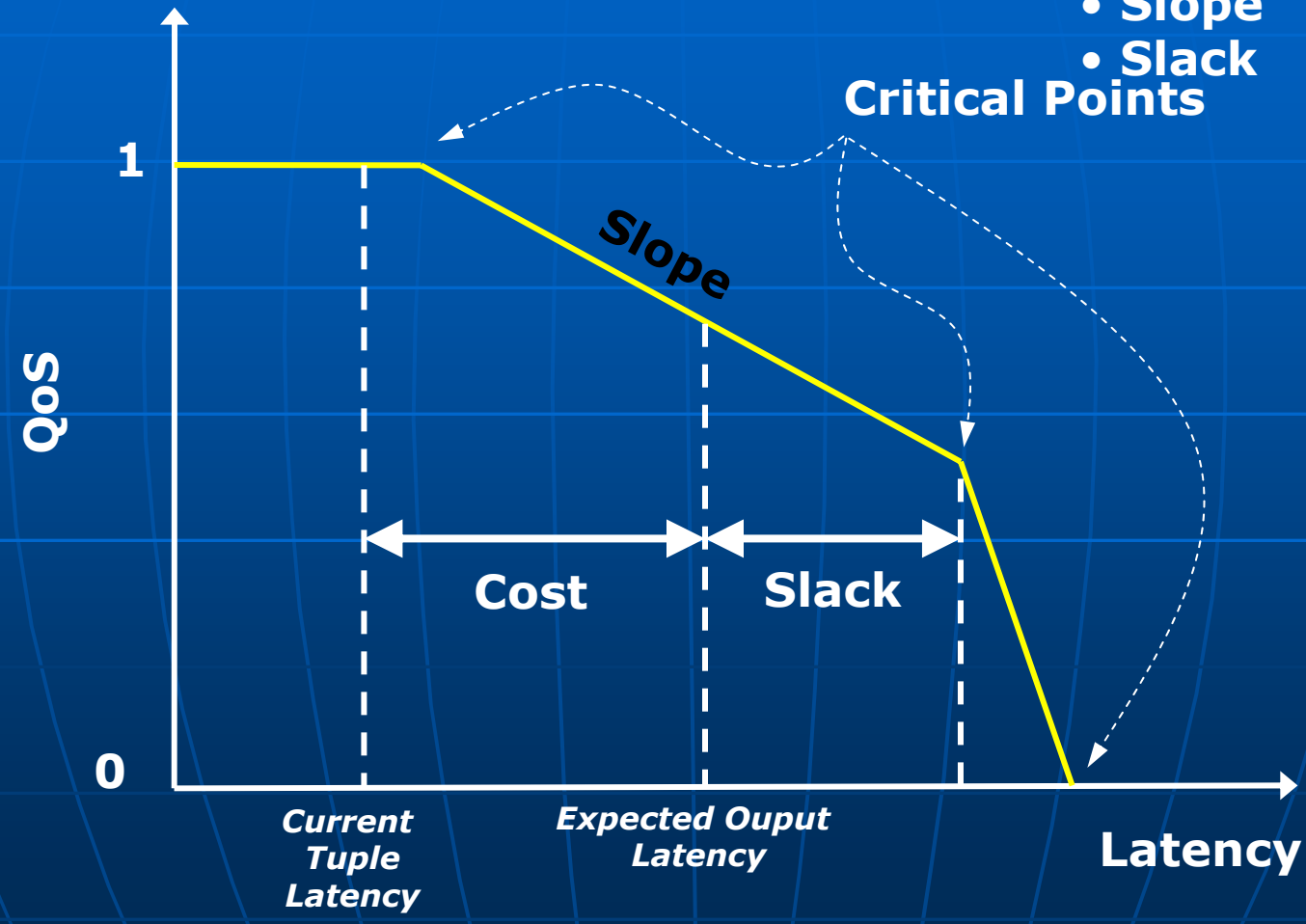
	Processing (p)	Call Overhead (o)	Avg Latency
Min-Cost	6p	3o	4.5p+3o
Min-Latency	6p	5o	3.25p+2.5o

Superbox Traversal

- Box execution order to improve
  - Throughput (Min-Cost)
    - Minimizes number of box calls
  - Latency (Min-Latency)
    - Produces tuples fastest
  - Memory Usage (Min-Memory)
    - Maximizes consumption of data per unit time
- Traversal selection based on
  - Targeted overhead
  - Achieving best QoS

Priority Assignment

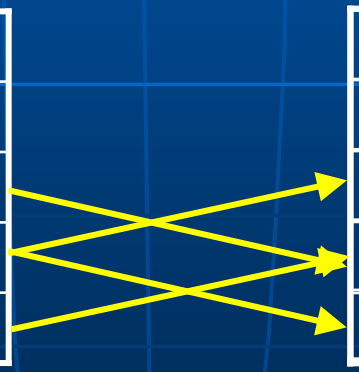
p-tuple ordering:
- **Slope**
- **Slack**



P-tuple Ordering

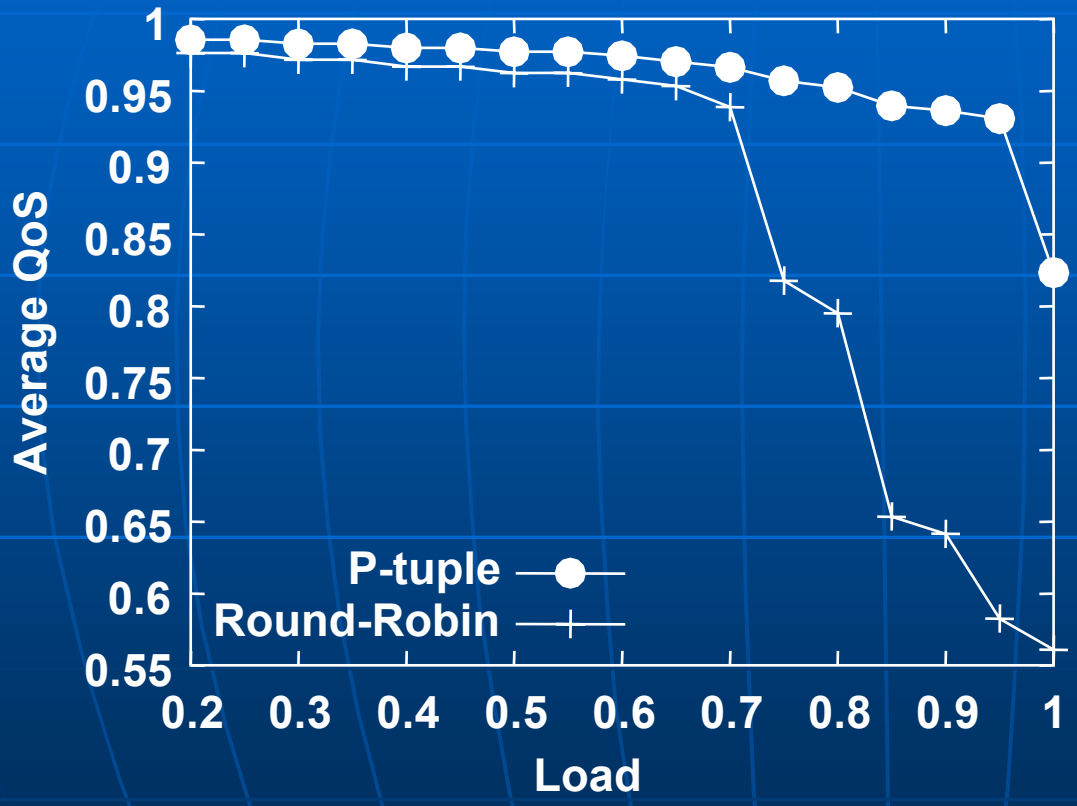
- At each scheduling event
  1. Compute **p-tuple** for each box
  2. Sort
- Example:

	p-tuple	
	Slope	Slack
b1	2	3
b2	4	1
b3	2	2



	p-tuple	
	Slope	Slack
b2	4	1
b3	2	2
b1	2	3

Priority Assignment Matters



20 applications

100 boxes

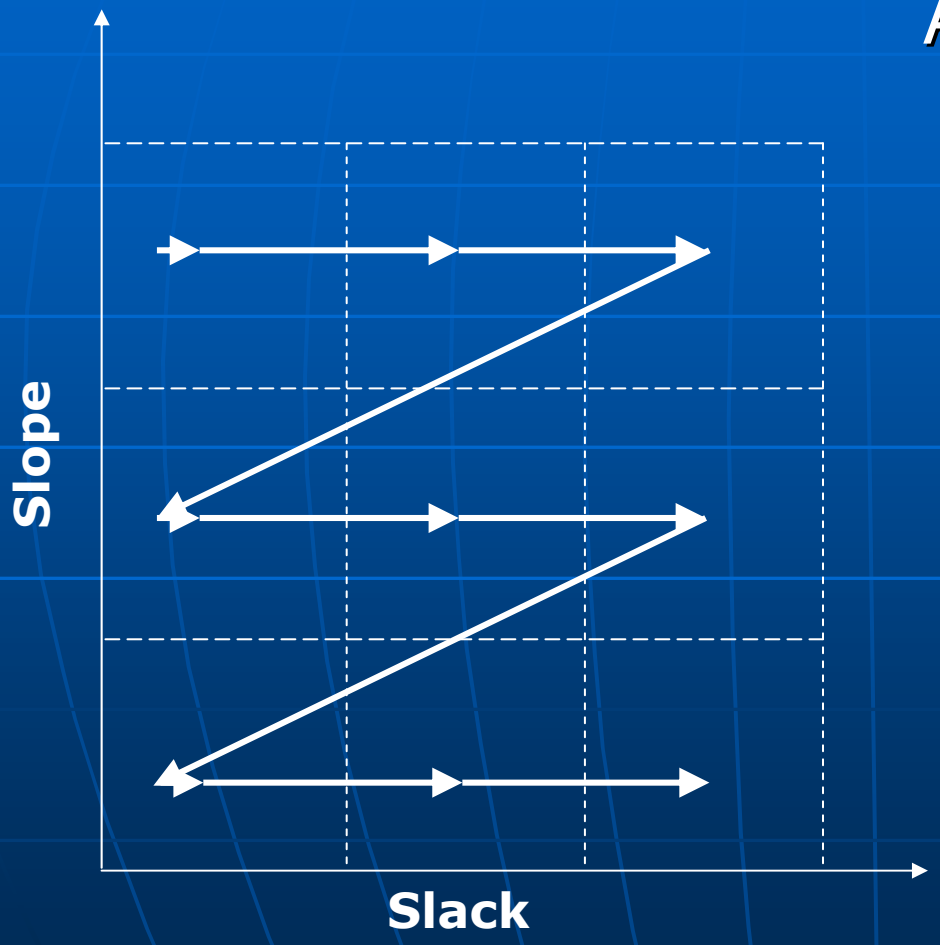
2 QoS graphs

1. Loose
2. Tight

Approximation for Scalability

- P-tuple method is slow
  - Compute for each box
  - Sort costly for large numbers of boxes
- Approximation to trade off quality for overhead
  - Bucketing
  - Pre-computation

Bucketing

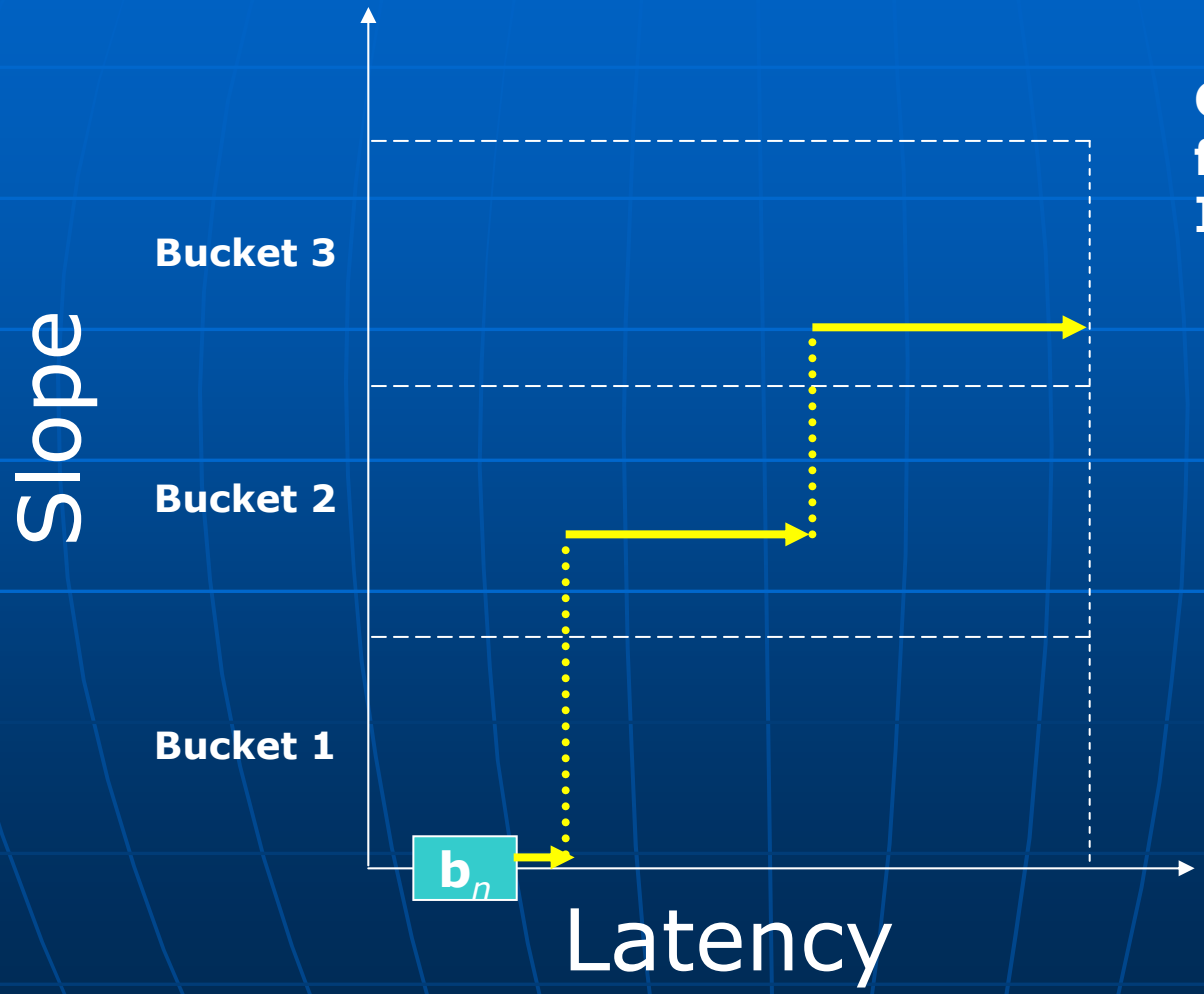


Approach:

- Partition slope/slack space into buckets
- At Scheduling event
  - Assign boxes to buckets
  - traverse buckets in p-tuple order
- # buckets controls approximation

But we still have to compute slope and slack

Pre-Computing Bucket Assignments

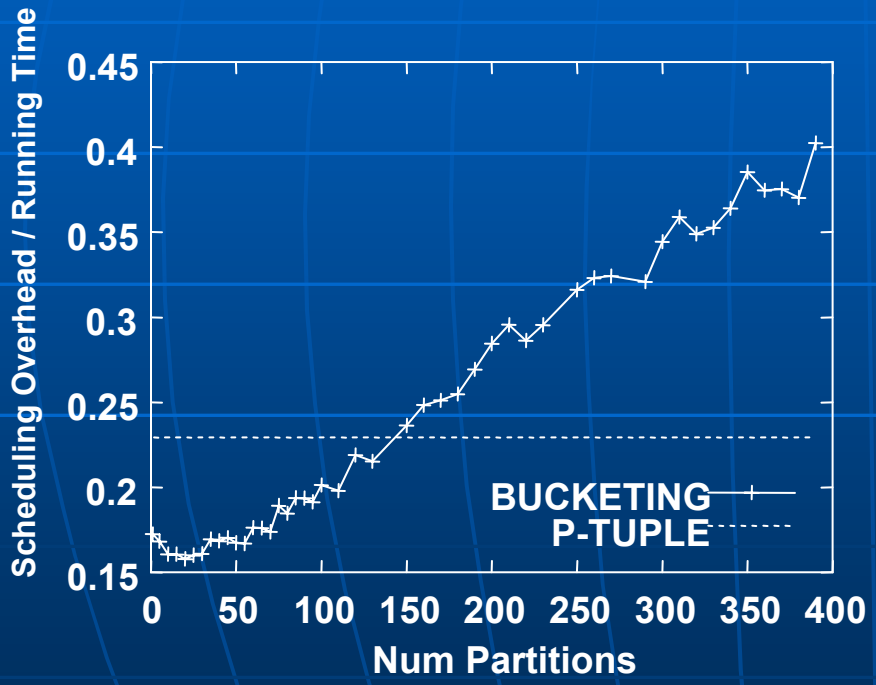


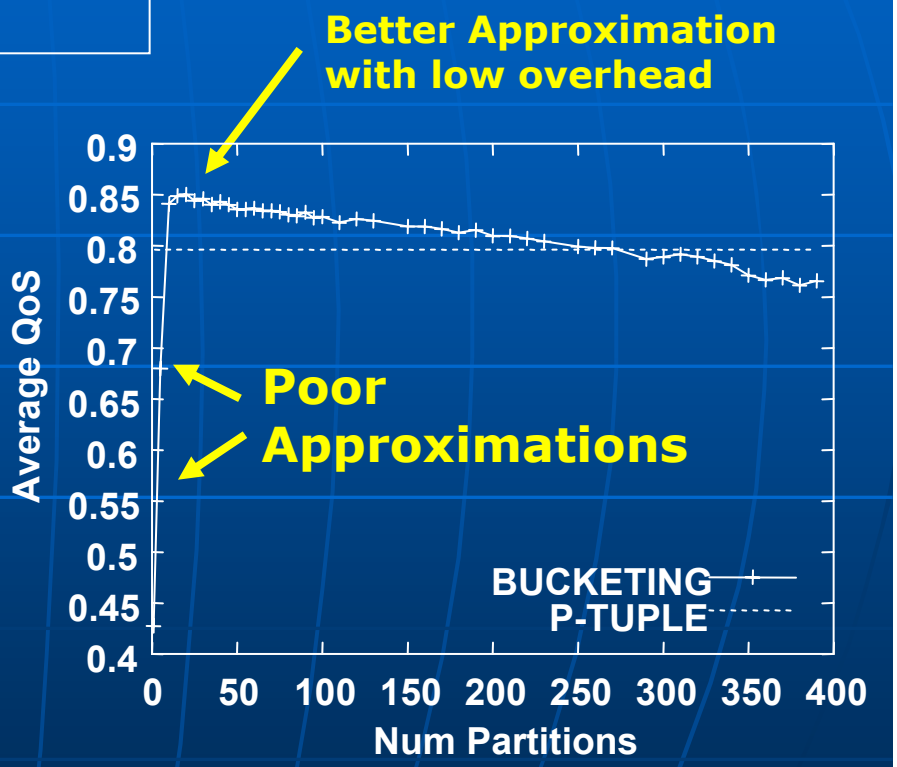
Given Latency, finding bucket In *O*(1)

Bucketing Works

200 Apps
1000 Boxes

2 QoS graph types
- Loose
- Tight





Related Work

- Operating Systems
  - [HLC91],[JRR97],[L88],[RS94]
- Real-time Databases
  - [AG93],[HCL-VLDB93],[KG94],[OS95],[R93]
- DSMS
  - Chain [BBDM-SIGMOD03]
    - Focuses on minimizing run-time memory usage
  - Eddies [AH-SIGMOD00]
    - Adaptability

Conclusions

- Overhead matters
  - Algorithms to reduce overhead
- Addressed QoS issues
  - Approximation technique trades scheduling quality for overhead
- Experimental investigation of scheduling algorithms
  - Run on Aurora prototype