Case Study of CSG712

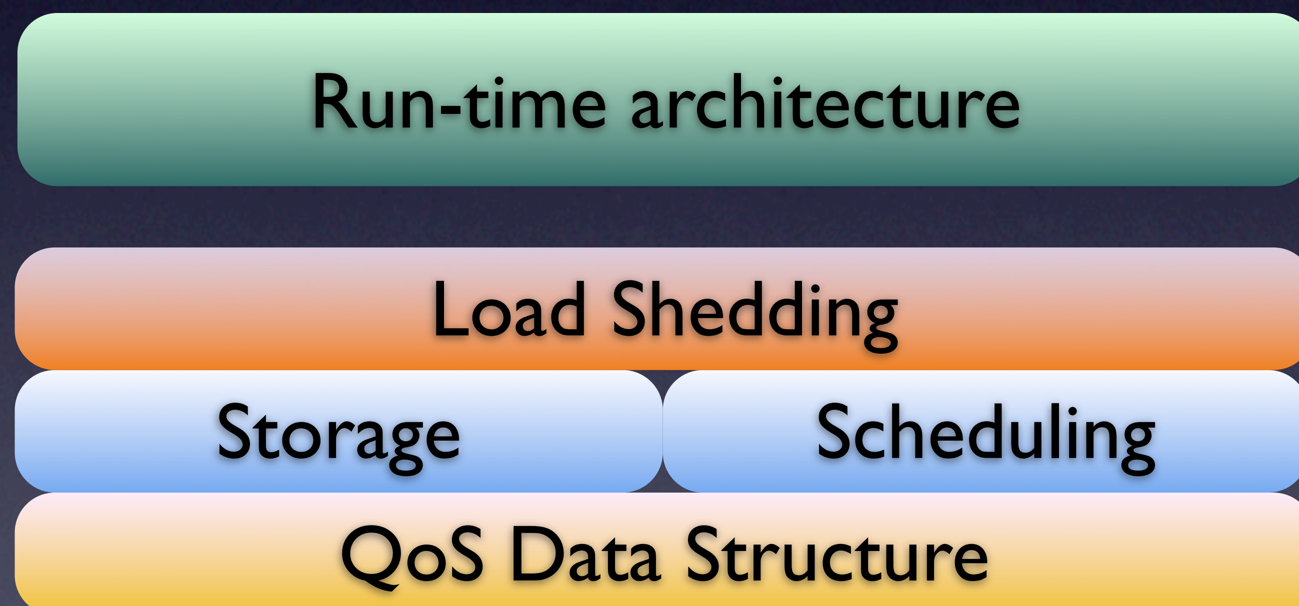
Data Stream Management System

Jian Wen

Spring 2008
Northeastern University

Outline

- Traditional DBMS v.s. Data Stream Management System
- First-generation: Aurora



- Second-generation: Medusa & Borealis

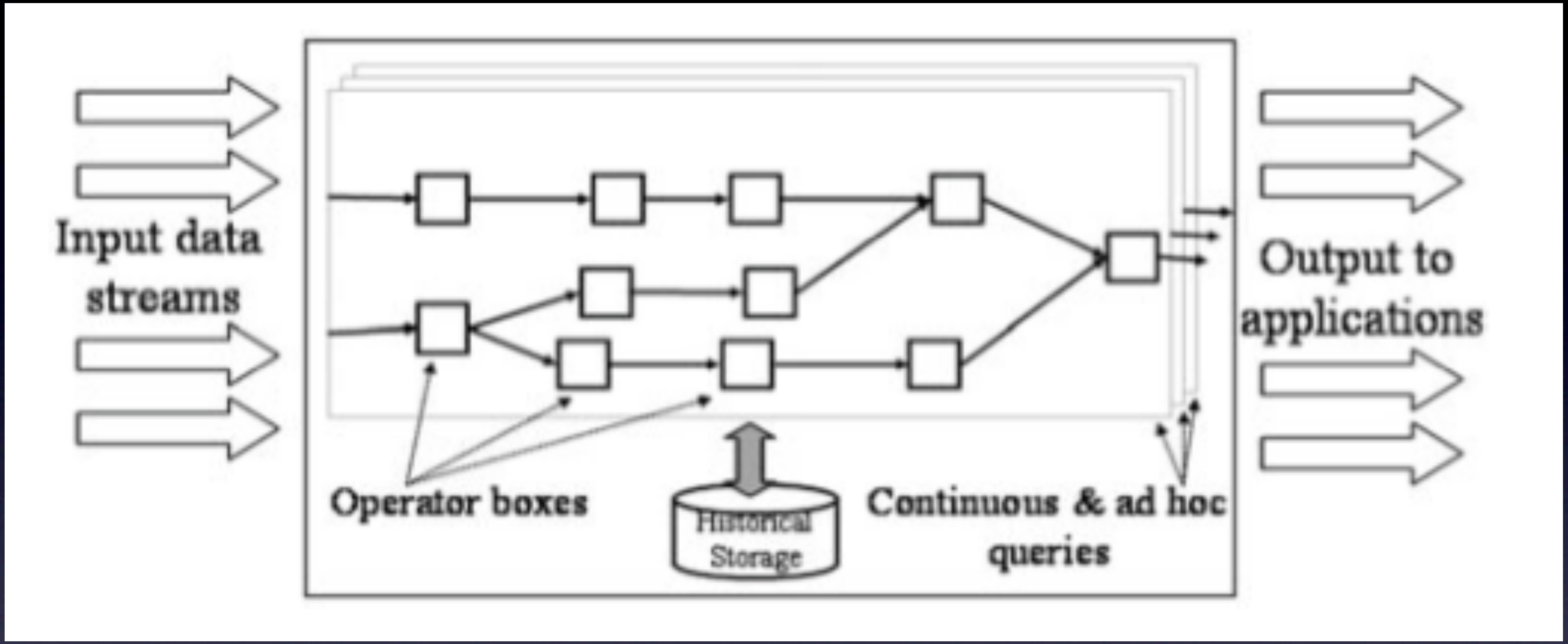
DBMS v.s. DSMS

HADP	DAHP
Current state of data is important.	Management over some history.
Triggers and alerters are uncommon.	Trigger-oriented
Synchronized data and exact-answer queries.	Unstable data and time-based queries.
No Real-time	Real-time

Aurora

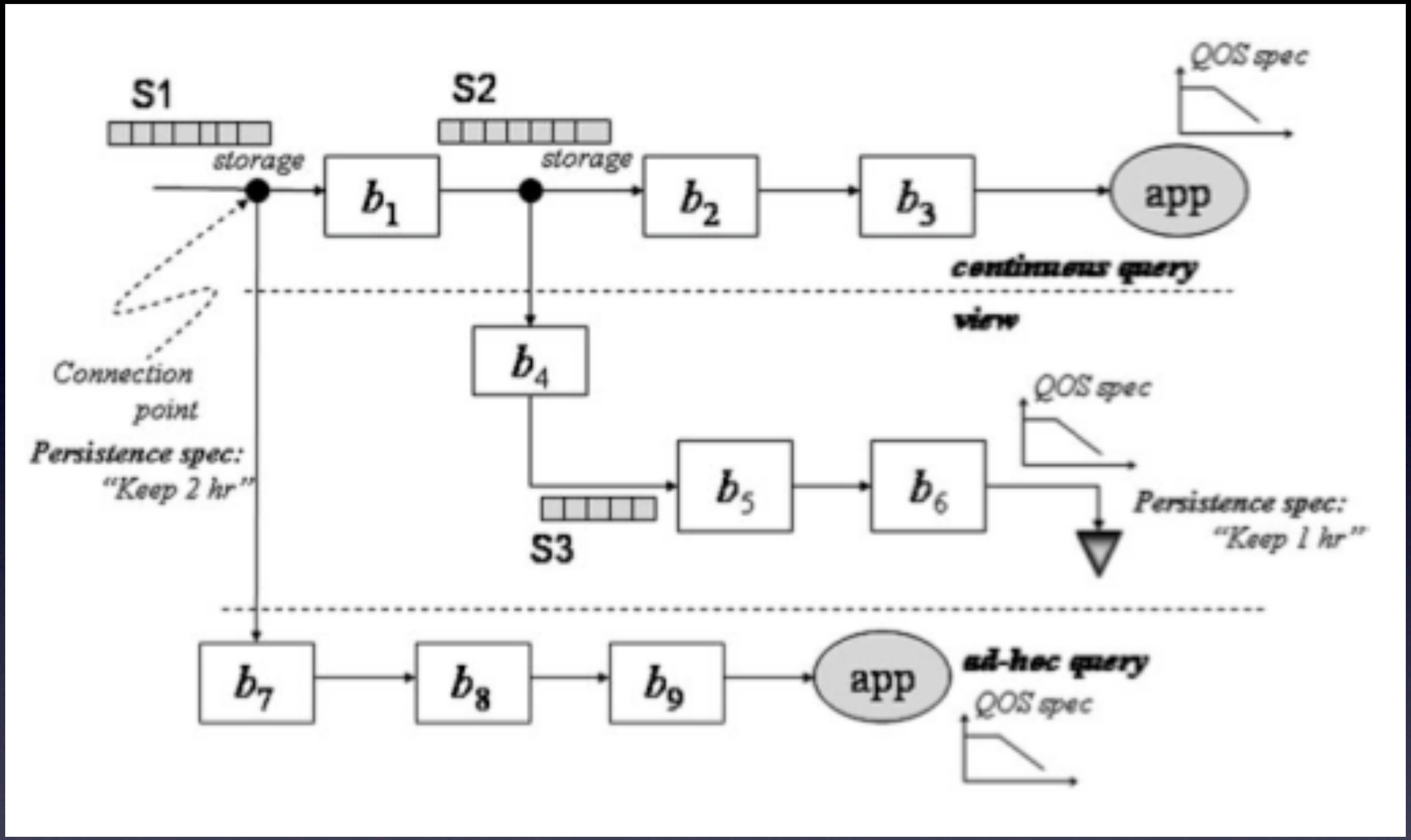
- First-generation data stream management system.
- Aimed to manage data streams for monitoring applications.
  - Sensors with limited capacity
  - Multiple data processing and queries(query network)

Aurora System Model



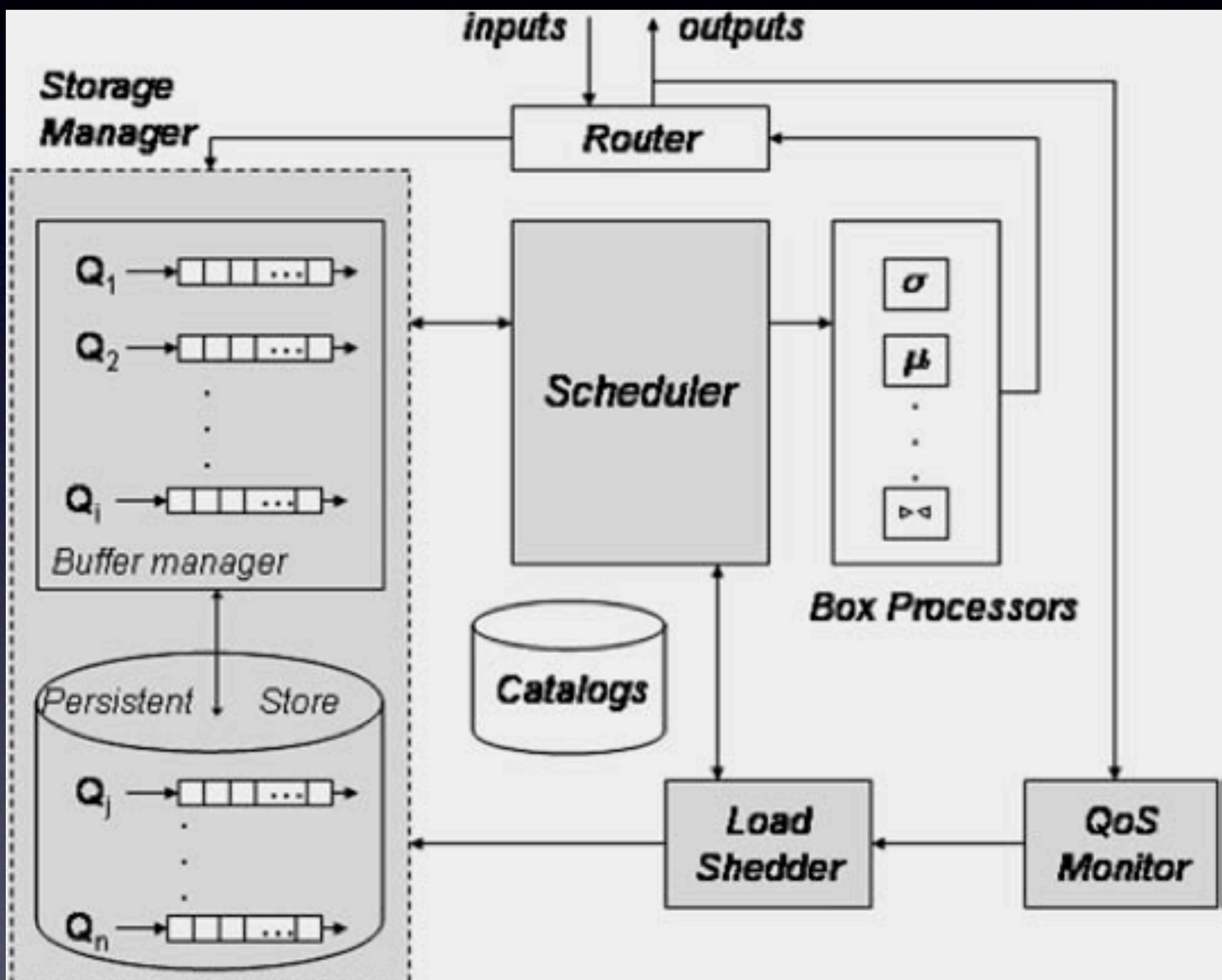
- Incoming streams are processed in the way defined by an ***application administrator***.
- Application administrator decides the processes adaptive to accepted queries requests.

Aurora Query Model



- Three kinds of queries: continuous queries(real-time processing), views and ad hoc queries(attached to connection points).
- Connection points provide persistent storage.
- QoS graphs specify the utility of the output in terms of performance and quality attributes.

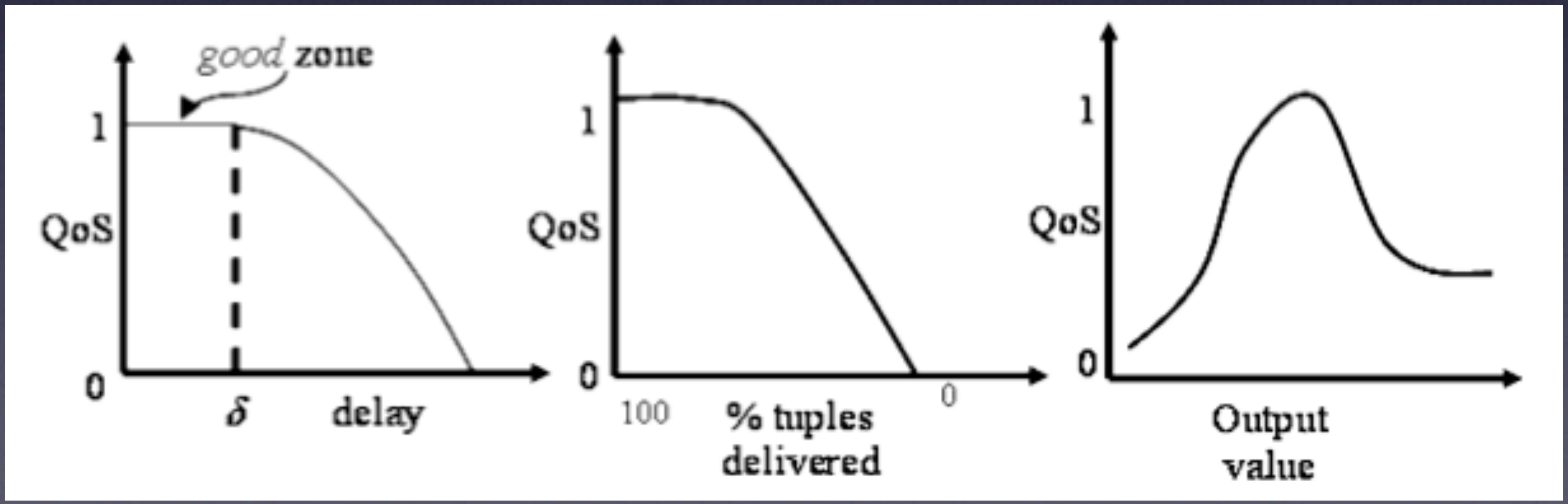
Aurora Run-time Architecture



- QoS Data Structure
- Aurora Storage Management(ASM)
- Real-time Scheduling
- Load Shedding

QoS Data Structure

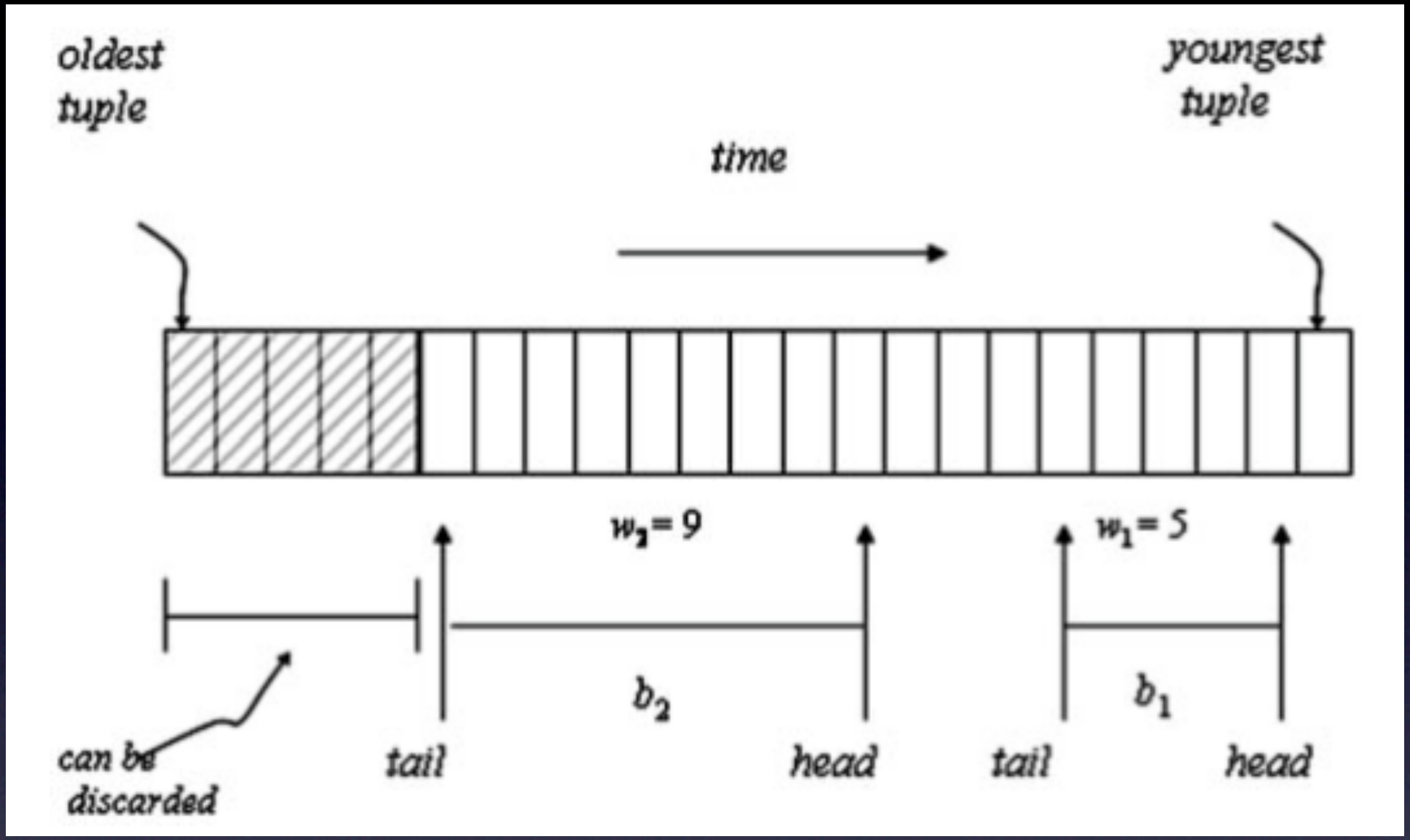
- Statistical information about Quality of Services
- Used to tune up the system to maximize QoS
- Three ways to measure QoS in Aurora



Aurora Storage Management

- Requirements for ASM:
  - Store the tuples being passed through an Aurora network -- main memory
  - Maintain extra storage for connection points -- external memory
- For connection points:
  - Like traditional DBMS: use B-Tree
  - Batch operations: ASM will gather up batches of tuples and then update the B-Tree.
- For tuples passing: queue & buffer

Aurora Storage Management



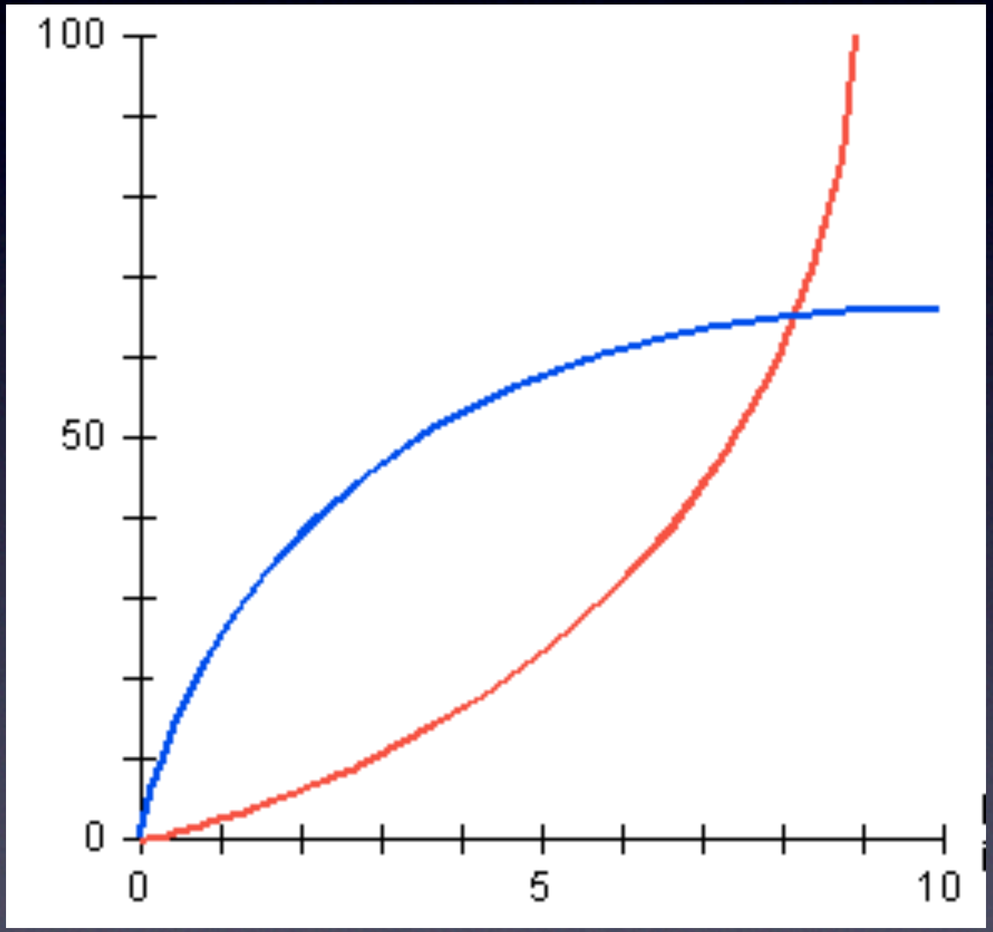
- Each operator box will have a variable-length queue.
  - The successor box will maintain two pointers on the queue. The gap between head and tail shows the size of the window.
  - The length of the queue can be adjusted by ASM dynamically(in the unit of fixed size)

Aurora Storage Management

- ASM maintains a buffer pool at start-up for queue storage.
- Buffer replacement policy:
  - ASM evicts the lowest-priority blocks in main memory(notice that one queue is not necessarily one block).
  - ASM periodically checks the buffer whether some blocks in buffer are not “running”, and replaces them with required, higher-priority blocks.

Aurora Run-time Scheduler

- Goal:
  - Maximize overall QoS.
  - Reduce overall tuple execution costs.
- In order to improve the performance, Aurora exploits two kinds of *nonlinearities*:
  - Interbox nonlinearity: E2E tuple processing costs may **drastically** increase if buffer space is not sufficient and tuples need to be shuttled back and forth between memory and disk several times in their lifetime. (red line if x is number of tuples and y is cost)
  - Intrabox nonlinearity: The cost of tuple processing may decrease as the number of tuples that are available for processing at a given box increases, by cutting down the number of box calls and optimizing in batch mode. (blue line if x is number of tuples and y is cost)



Aurora Run-time Scheduler

- Basic idea: try to avoid the Interbox nonlinearity and propagate the Intrabox nonlinearity.
- Two scheduling policies:
  - Train scheduling: batching multiple tuples as input to a single operation box.
  - Superbox scheduling: pushing a tuple train through multiple boxes.
- In details:
  - have boxes queue as many tuples as possible without processing them, thereby generating long tuple train;
  - process complete train at once;
  - pass whole train to subsequent boxes without going to disk;
  - scheduler tells each box when to execute and how many queued tuples to process.

Aurora Run-time Scheduler

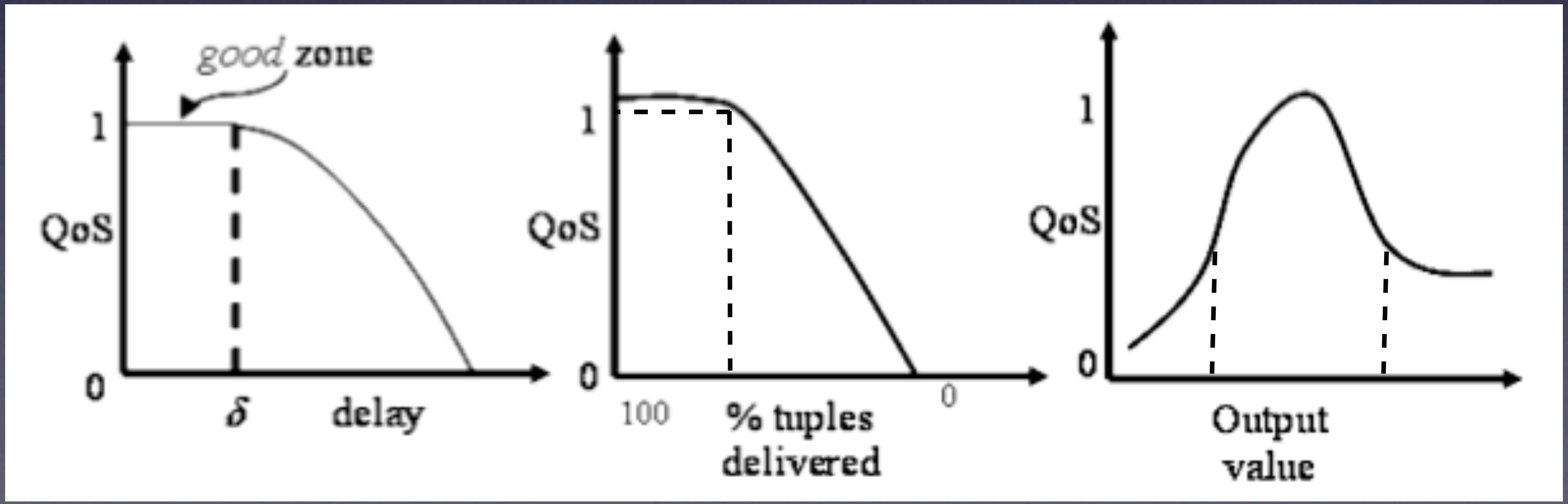
- Priority assignment is based on the utility of outputs:
  - Static-based approach: if we can know ahead the expectation of utility of the output from some box, we will try to assign higher priority to it.
  - Feedback-based approach: continuously observes the performance of the system and dynamically reassign the priorities: increase the priorities of those that are not doing well and decrease priorities of the application that are already in their good zones(evaluated by the QoS).
- Combine scheduling with priority:
  - first assigning priorities to select individual outputs and then exploring opportunities for constructing and processing tuple trains.

Aurora Load Shedding

- Try to avoid overload and keep good performance
- Detect/Monitor - Shedding
  - Two introspection schemes are used to check the overload in system.
    - Static analysis and dynamic analysis
      - Static: if we have known the expectation of the stream and also the capacity of the processing path, we can easily judge whether there are too much flows on the processing path.
      - Dynamic: for each time when we finish the query processing, we check the QoS-Delay graph to see whether most of the outputs are in the good zone. If not, we can say that there is an overload.

Aurora Load Shedding

- Two dropping policy to minimize the degrade of overall system utility and keep the application semantics.
  - Tolerant dropping
    Based on QoS-Drop graph, randomly drop with the percentage with minimum QoS lost.
  - Semantic load shedding by filtering tuples
    Based on QoS-Value graph, filter tuples which are less important.



Distributed DSMS

- Second-generation DSMS ...
- Prototype came out with the first-generation!
  - At the same time when Aurora came up, Aurora* and Medusa had been proposed for distributed data stream management.
  - Borealis is the youngest heir of Aurora and Medusa, which is aimed high-available distributed stream services.

Scalable Distributed Stream Processing

- Aurora*: intra-participant distribution
  - Multiple single-node Aurora servers that belong to the same administrative domain.
  - Partition operation boxes in original one Aurora system into several peer systems.
- Medusa: inter-participant federated operation
  - Distributed infrastructure that provides service delivery among autonomous participants.
  - Medusa is a *agoric system*, using economic principles to regulate participant collaborations and solve problems on load and sharing.

Reference

- Abadi et al. Aurora: a new model and architecture for data stream management. The VLDB Journal The International Journal on Very Large Database (2003)
- Stan Zdonik, Michael Stonebraker, Mitch Cherniack. The Aurora and Medusa Projects. IEEE Data Engineering Bulletin (2003)
- Cherniack et al. Scalable Distributed Stream Processing. CIDR Conference (2003)
- Abadi et al. The Design of the Borealis Stream Processing Engine. Second Biennial Conference on Innovative Data Systems Research (2005)

Questions?

Thanks!