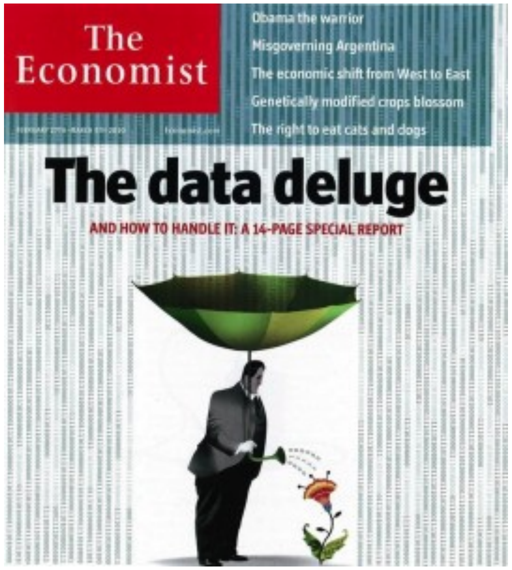
Handling Big Streaming Data with DILoS

Alexandros Labrinidis

Advanced Data Management Technologies Lab
Department of Computer Science
University of Pittsburgh

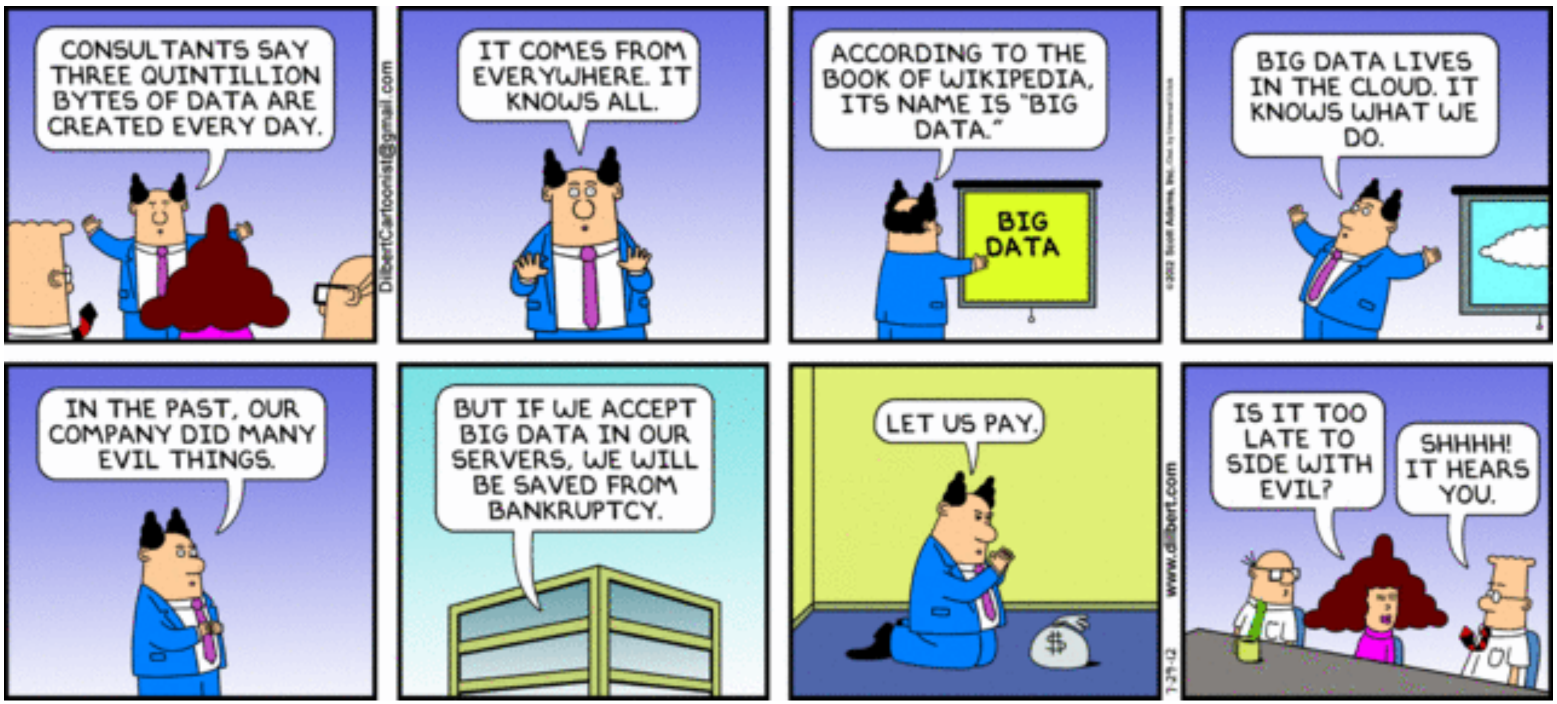
University of Waterloo – May 21, 2014

You know Big Data is an important problem if…



- It is featured on the cover of Nature and the Economist!

You know Big Data is an *even more* important problem if...



- It has a Dilbert cartoon!

What is Big Data?

Definition #1:

- Big data is like teenage sex:
  - everyone talks about it,
  - nobody really knows how to do it,
  - everyone thinks everyone else is doing it,
  - so everyone claims they are doing it...

Definition #2:

- Anything that Won't Fit in Excel!

Definition #3:

- Using the Vs

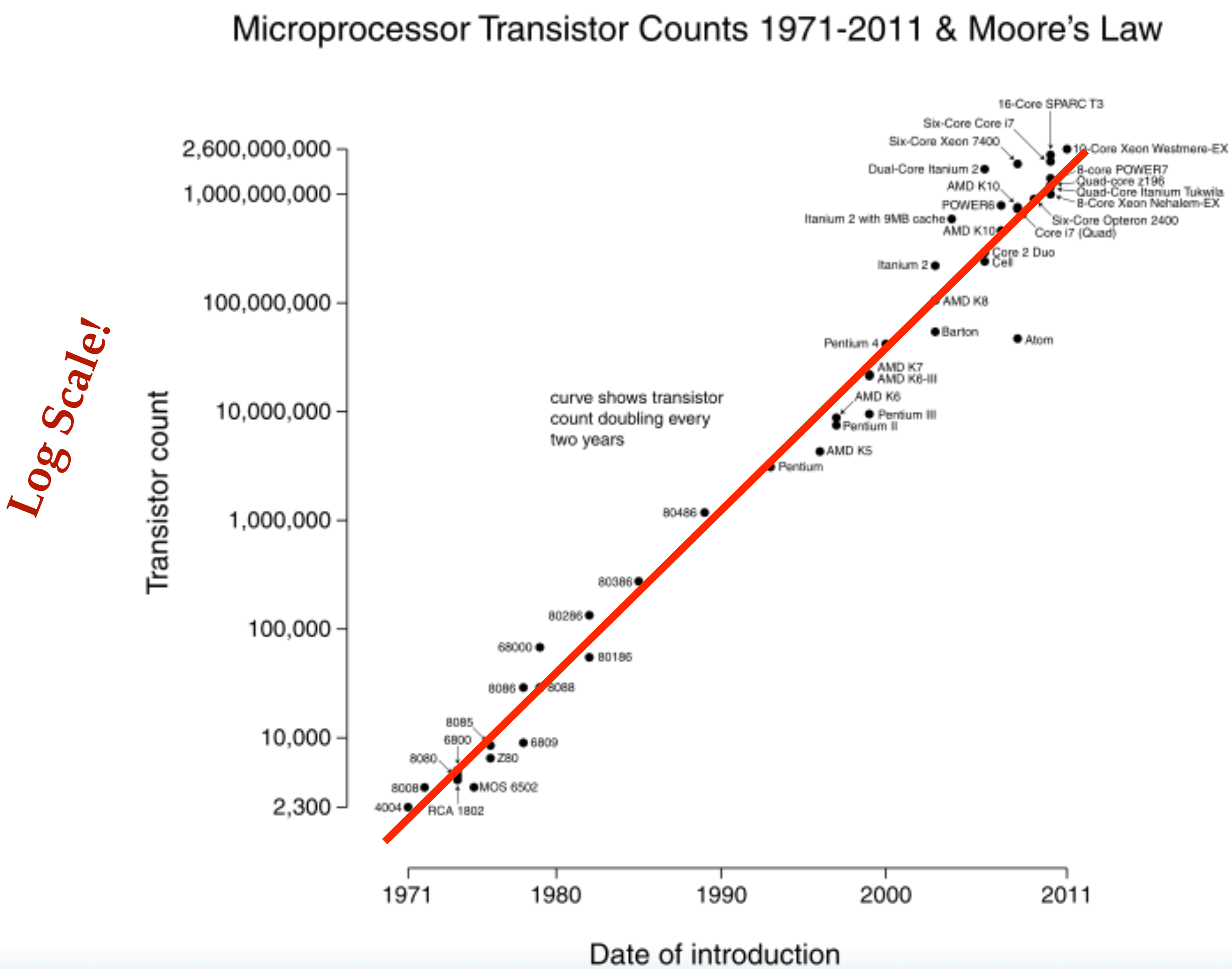
The three Vs

- **Volume** - size does matter!
- **Velocity** - data at speed, i.e., the data “fire-hose”
- **Variety** - heterogeneity is the rule

Five more Vs

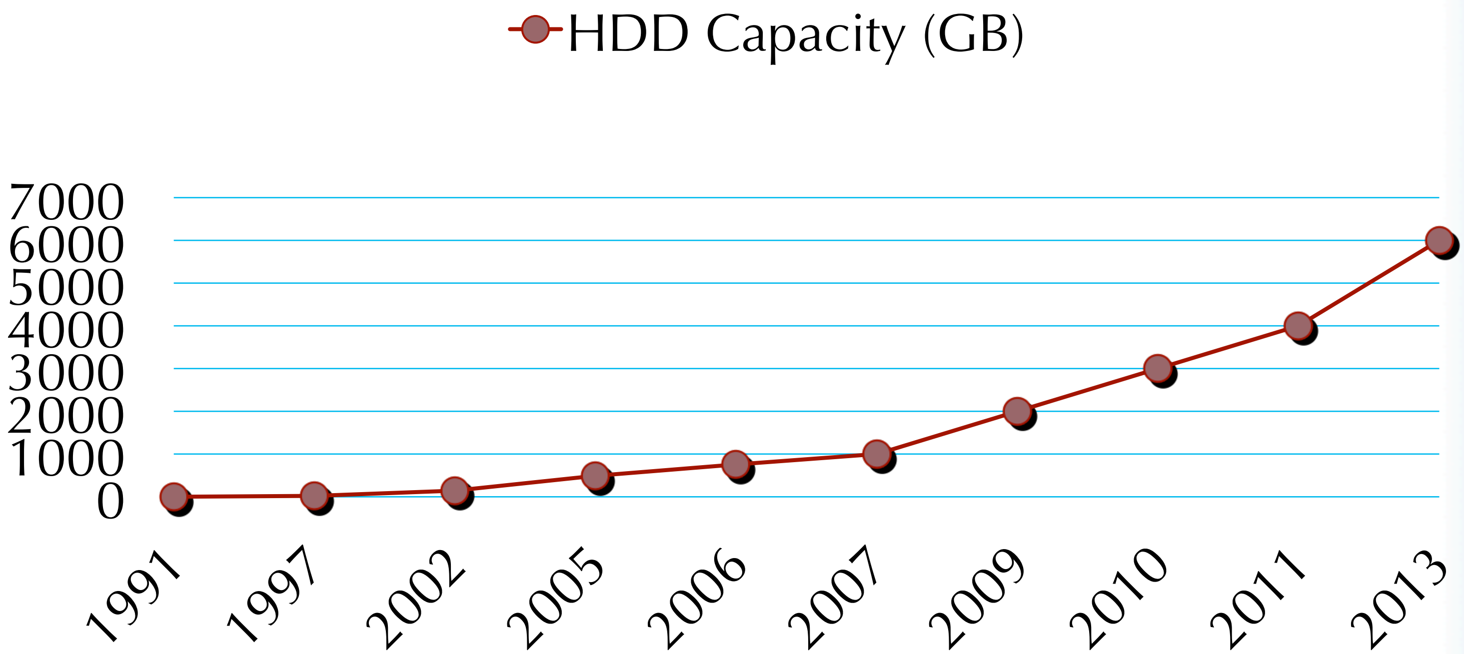
- **Variability** - rapid change of data characteristics over time
- **Veracity** - ability to handle uncertainty, inconsistency, etc
- **Visibility** – protect privacy and provide security
- **Value** – usefulness & ability to find the right-needle in the stack
- **Voracity** - strong appetite for data!

Enter Moore's Law



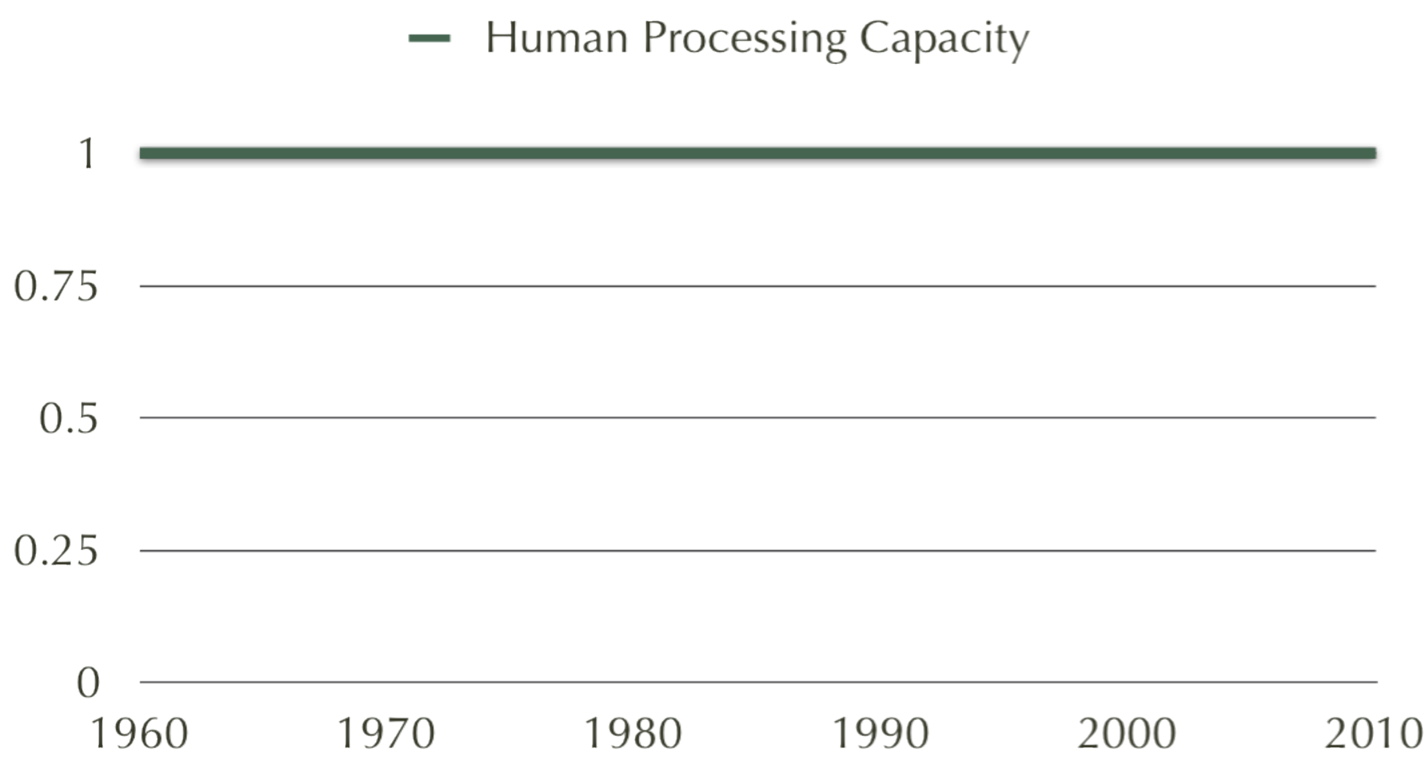
[Wikipedia Image]

Storage capacity increase



[Wikipedia Data]

But

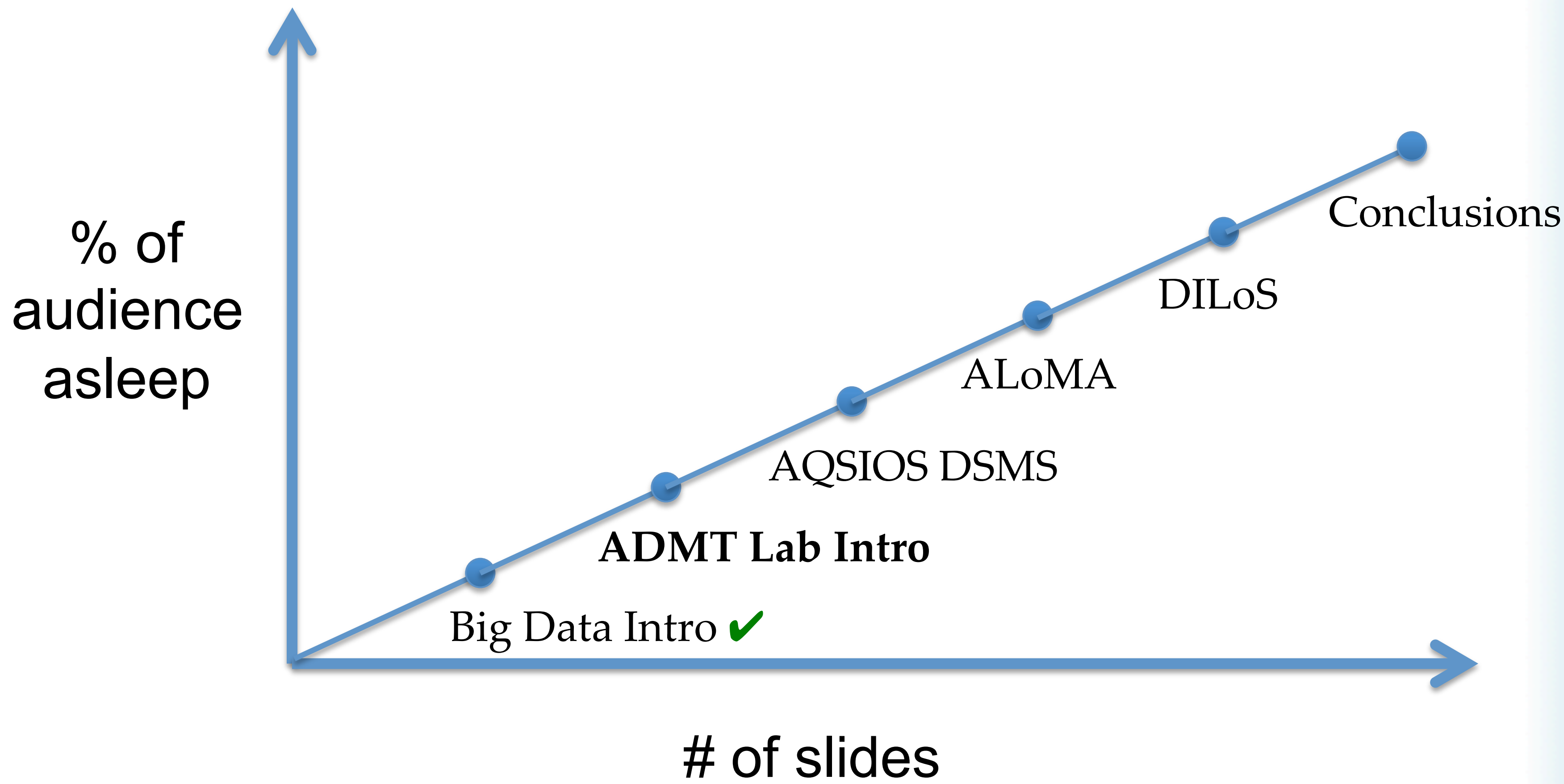


- **Human processing capacity** remains roughly the same!

We refer to this as the:

Big Data – Same Humans Problem

Roadmap



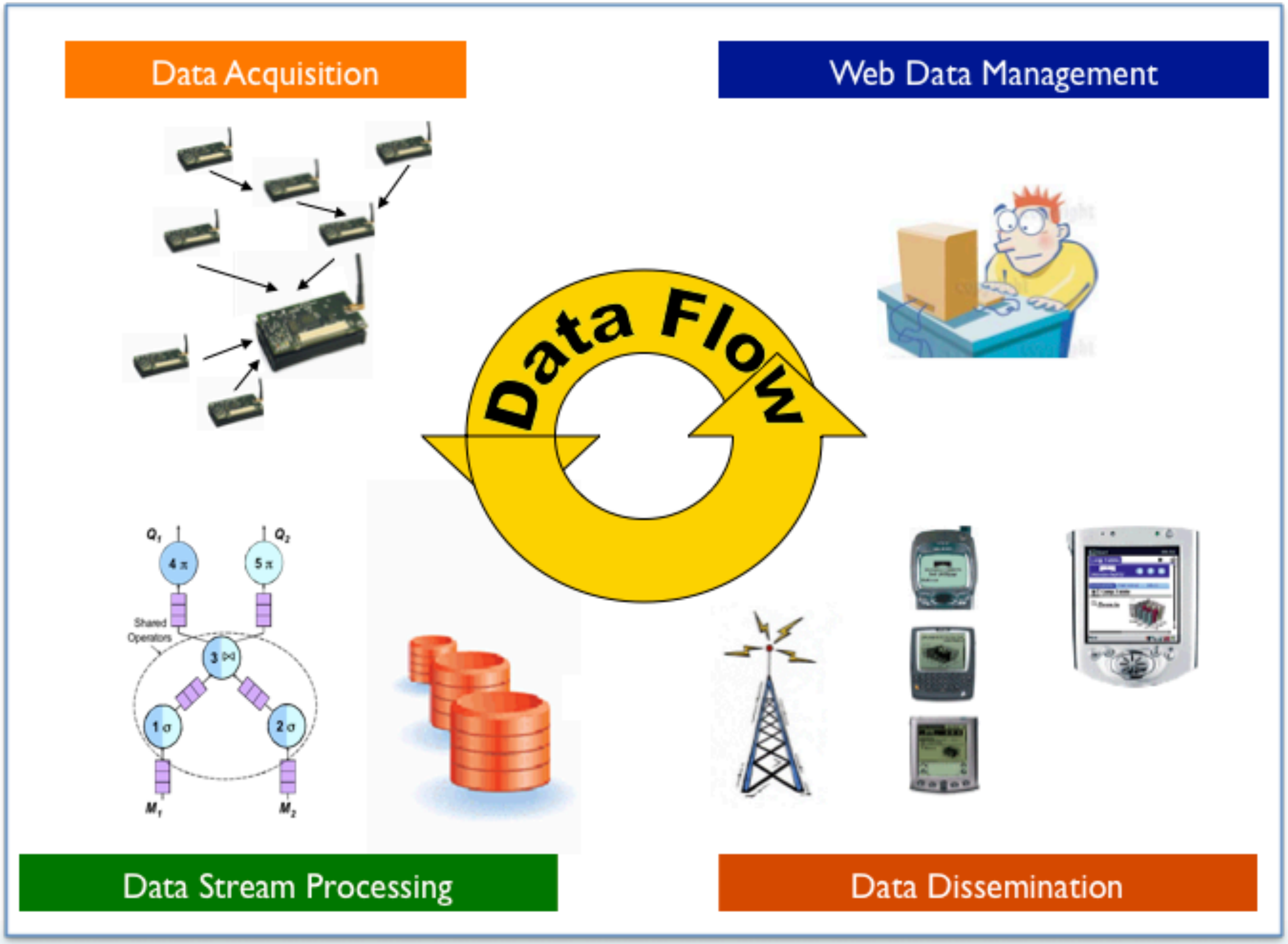
% of audience asleep
Conclusions
DILoS
ALoMA
AQSIOS DSMS
ADMT Lab Intro
Big Data Intro ✔
# of slides

About the ADMT Lab

- Directed by
  - Panos K. Chrysanthis
  - Alexandros Labrinidis

- Established in 1995
- 4+2 PhD students, 2 MS students, 6 REUs
- *User-centric data management for network-centric applications*

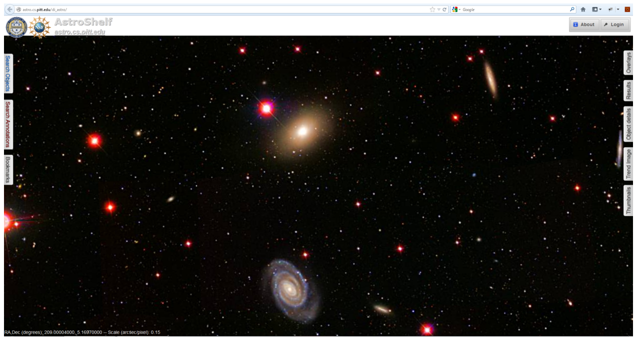
Entire Data Lifecycle

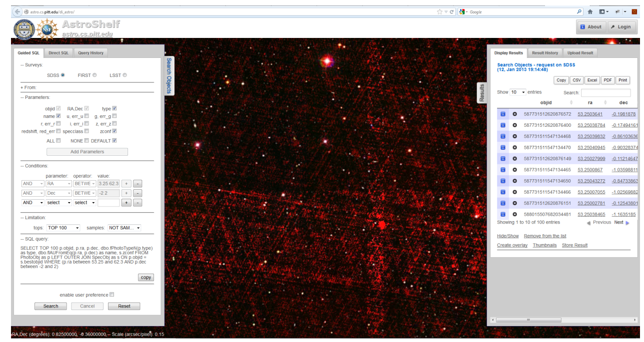


AstroShelf

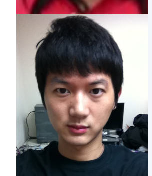
Volume Velocity Variety
Veracity Visibility

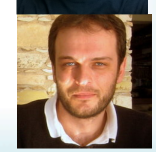
- *Understanding the Universe through scalable navigation of a galaxy of annotations*





- Astronomy data from multiple sources (images & catalogs)
- Support collaboration of:
  - ➢ people (view-based, declarative annotations)
  - ➢ software / data (web services)
  - ➢ resources (utilizing local and remote storage)
- CONFLuEnCE prototype: *continuous workflows* [Sigmod 2011 & 2012]

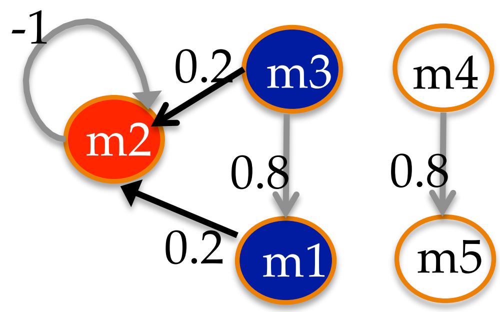




AstroShelf (cont.)

Volume Velocity Variety
Veracity Visibility

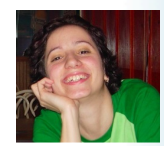
- *User-centric features:*



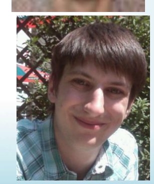
"I like **drama movies** a bit more than **horror movies**, Intensity of preference 0.2"

```
SELECT * FROM Plants, Supplies, Polluted_H2O
WHERE Supplies.type = "solvent"
    AND Supplies.name = Polluted_H2O.pollutant
    AND Polluted_H2O.location = Plants.location
    AND Plant.id = Supplies.plant_id
PREFERRING $l = Querier  HOLDS
   OVER <*,{(pollutant)},$l>
 CASCADE LESSTHAN(runtime, 120)
    AND $l = Querier HOLDS OVER <join,*,$l>;
```

- Unified model for user preferences
  - combine quantitative & qualitative user preferences into a single graph model to guide query result personalization



- Protecting privacy in distributed query processing
  - declarative preferences allow users to balance the tradeoff between privacy and performance



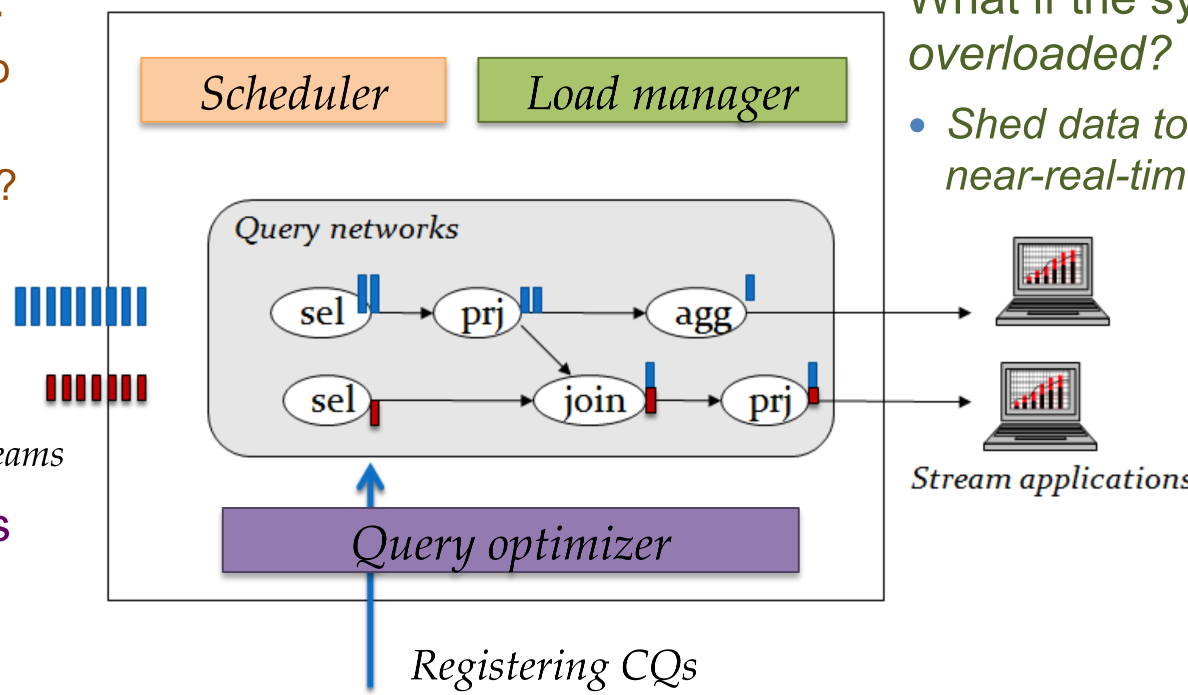
AQSIOS

Volume Velocity
Variability

- Efficiently Utilizing Resource in a Data Stream Management System

CPU time sharing:

- Which operator to execute now?
- And for how long?



Which query plans are the best?

What if the system is *overloaded?*

- *Shed data to meet the near-real-time requirement*

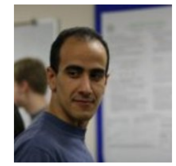
Multiple classes of CQs

- Each class has a different ***priority***

AQSIOS

Volume Velocity
Variability

- Prototype Data Stream Management Systems
  - o **Aggregate Continuous Query optimizer**
    - WeaveShare and TriWeave
      [Shenoda et al., CIKM'11 and ICDE'12]
      - o Optimized processing to eliminate redundant computation



  - o **Continuous Query Schedulers**
    - HR, HNR [Sharaf et al., VLDB'06 and TODS'08]
      - o Average vs Max Response Time
      - o Average vs Max Slowdown

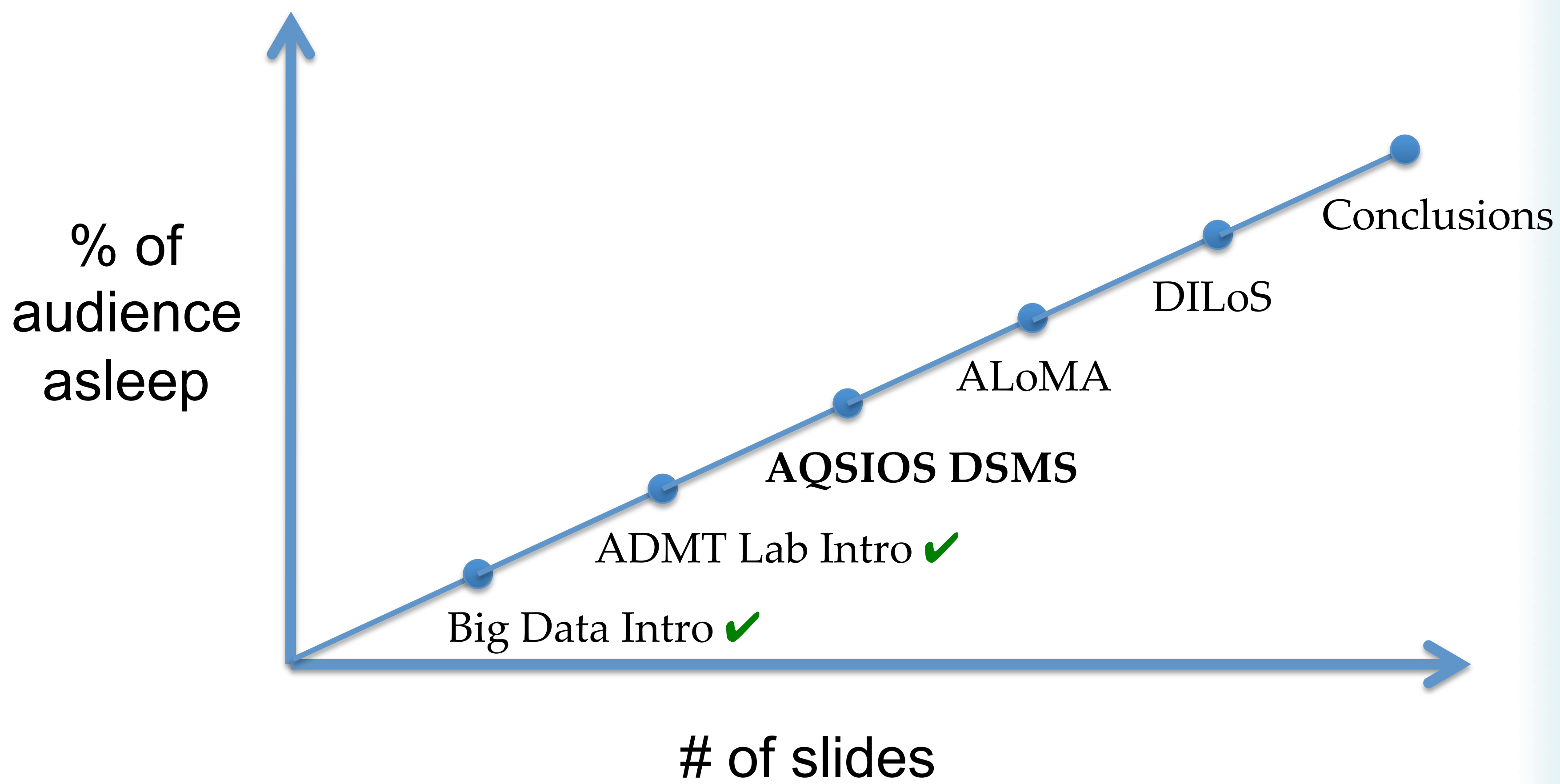
    - CQC and ABD [Al Moakar et al., DMSN'09 and SMDB'12]
      - o Priority Classes
      - o Single-, Dual-, Multi-core, Cloud

AQSIOS (cont.)

Volume Velocity
Variability

- **Load shedder and scheduler-load shedder synergy**
  - SEaMLeSS [Pham et al., SMDB'13]
    - SElf Managing Load Shedding for data Stream management systems
  - DILoS [Pham et al., SMDB'11]
    - Seamless integration of priority-based scheduler and load shedder
    - Consistently honor worst-case delay target with differentiated classes of service
    - Exploit system capacity better

Roadmap


% of audience asleep
Conclusions
DILoS
ALoMA
AQSIOS DSMS
ADMT Lab Intro ✔
Big Data Intro ✔
# of slides

System Model & Metrics

- *Multiple priority classes* of CQs
  - Priorities have been quantified into numbers
    - Higher value means higher priority
- Two requirements under *overload state*:
  1. *Guarantee worst-case Quality of service* **(QoS)**
     - **Worst-case QoS** = **worst-case response time** = **delay target**
     - Each class can require a different worst-case QoS
     - Supported by load manager (load shedder)
  2. *Maximize Quality of Data* **(QoD)** *with priority consideration*
     - QoD = 100% - data loss due to shedding
     - Need to consider priorities of CQ classes
     - **Involve both scheduler and load manager - Why?**

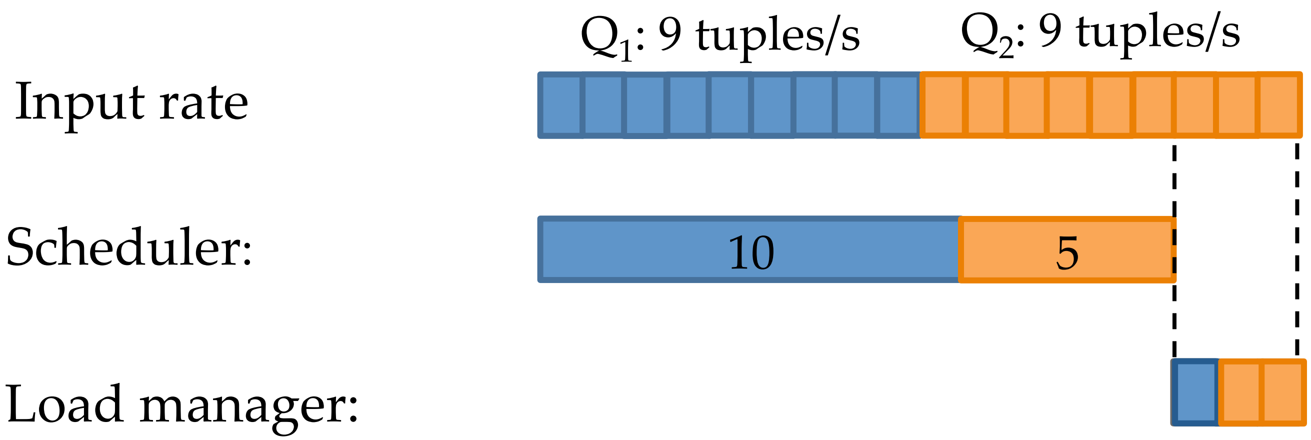
State-of-the-art

- Previous works consider either…
  - Priority-based scheduling
    - CQ's priority (through QoS function, deadline): e.g., [Carney et al., VLDB'03], [Wei et al.,, ISORC' 06]
    - Class' priority: [Al Moakar et al., DMSN'09, SMDB'12]
  - Or priority-based load shedding
    - CQ's loss-tolerance functions [Tatbul et al., VLDB'03]

Now we need both of them to work together …

Motivation

- Two CQs **Q_1** and **Q_2**
  - The same cost
  - Q_1's priority is twice as high as Q_2's



→ Q_2 is still overloaded

→ Q_1 suffers from unnecessary shedding

→ System capacity is not fully used

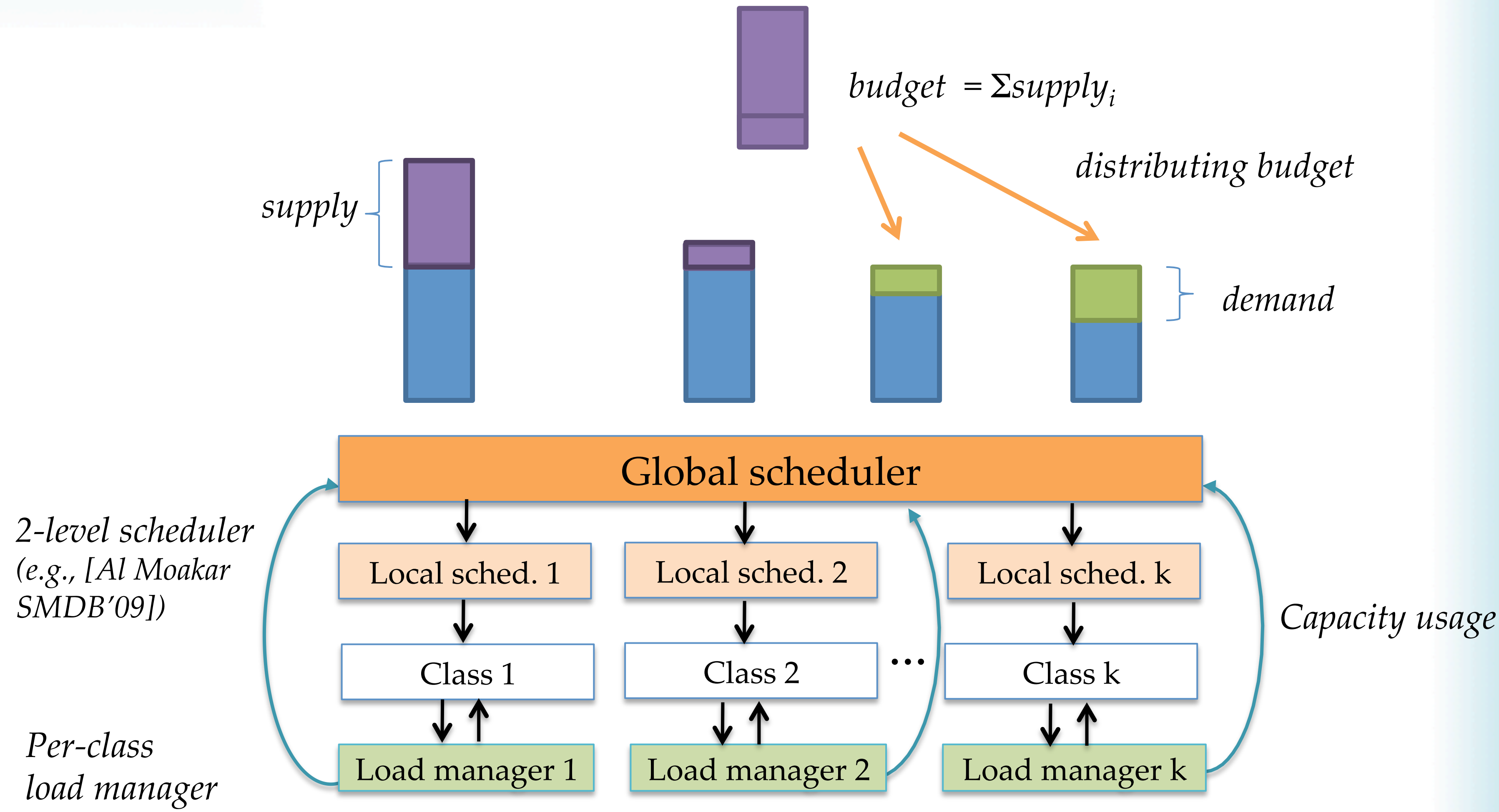
Motivation

- Making the load manager aware of the scheduler's policy?
  - **Load manager:** I should know that the scheduler can process up to 10 tuples of Q_1 and 5 tuples of Q_2 and…
  - **Scheduler**: well, all I can tell you is *in this cycle* I am giving Q_1 x% of time to execute and Q_2 *y% and…*, also *many things out of my control*
    - Context switching time
    - Background jobs that share the CPU resource
    - The actual query load
  - **Load manager**: 

Our Hypothesis

- By exploiting the **synergy** between the scheduler and the load shedder we can
  - Support CQ's priority consistently
  - Improve the utilization of CPU resource

Our solution: DILoS framework

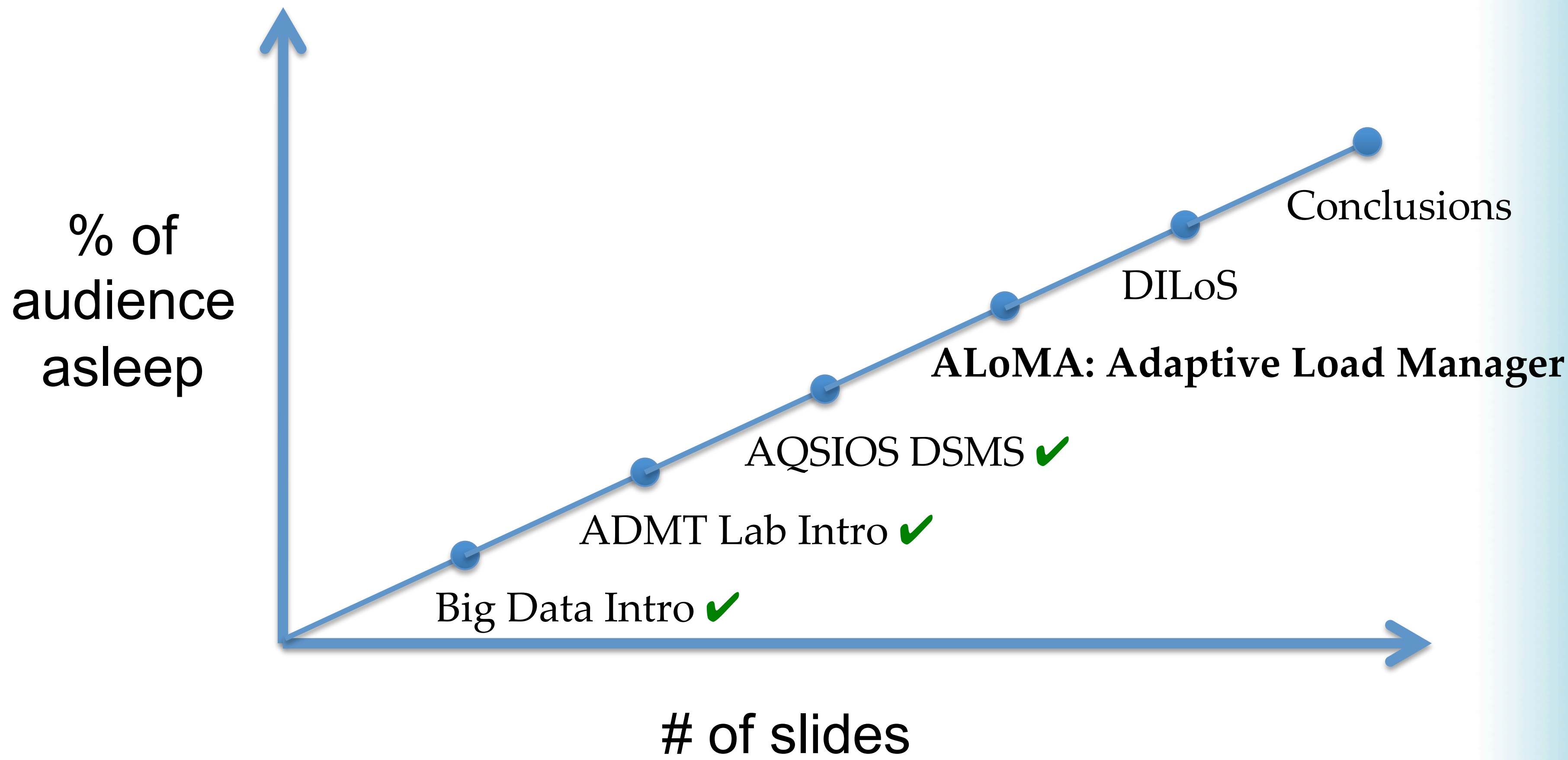


Benefit of our proposed DILoS framework

- The load manager works in concert with the scheduler in honoring CQs' priority
  - The load manager does not needs to have its own priority-based policy
    - Controls the load in each class as if it is a *virtual system*
    - *Follows exactly the priority enforcement of the scheduler*

- Load manager's feedback improves scheduler's decision
  - Better exploits system capacity

Roadmap



% of audience asleep
Conclusions
DILoS
ALoMA: Adaptive Load Manager
AQSIOS DSMS ✔
ADMT Lab Intro ✔
Big Data Intro ✔
# of slides

Load manager for DILoS

- Each class load manager needs to decide "*when and how much load to shed*"
  - Estimate the load of each class
    - [Tatbul et al. , 2003], based on input rates, operator's cost and selectivities
  - *Estimate the system capacity each class actually has*
    - **???**

"When and how much"- related definitions

- *Incoming load* ***L***
  - The amount of time needed to process all the tuples coming in per time unit (say, a second)

- *System capacity* L_C:
  - The fraction of each time unit the system can spend on processing the incoming tuples
  - Approximated by a ***headroom factor H*** in [0-1]

- *Overload*:
  - when $L > L_C$

“when and how much” state-of-the-art

- Aurora [Tatbul et al., 2003]
  - Excess load = $L-L_C$
  - No feedback loop, cannot honor delay target
- CTRL [Tu et al., 2006]
  - Based on number of queued tuples to adjust shedding decisions
  - Honors delay target, outperforms Aurora
- *Both require manually tuned headroom factor **H** to estimate the system capacity!*
  - Offline, manual tuning of H is impractical
  - *Clearly not applicable in this context of per-class load manager!*

Our Proposal: ALoMa – Adaptive Load Manager

- *Starts with some reasonable value of H, and adjusts it accordingly*

- *Has two modules:*
  - ***Statistics–based load monitor***: estimates the system load based on input rate, operators' costs and selectivities
  - ***Response time monitor***: monitors the *level* and *moving trend* of the *actual response time* to infer about the system load status

ALoMa- Headroom Factor Adjustment

- The two modules disagree: adjust H
  - The load monitor says "*overloaded*" but the response time monitor says "*not overloaded*":
    - ***Increase*** *H so that* L_C *is increased towards* L
  - The load monitor says "*not overloaded*" but the response time monitor says "overloaded"
    - ***Decrease*** *H so that* L_C *is reduced towards L*
- The two modules agree: *excess load* = $L - L_C$

ALoMa – Headroom Factor Adjustment

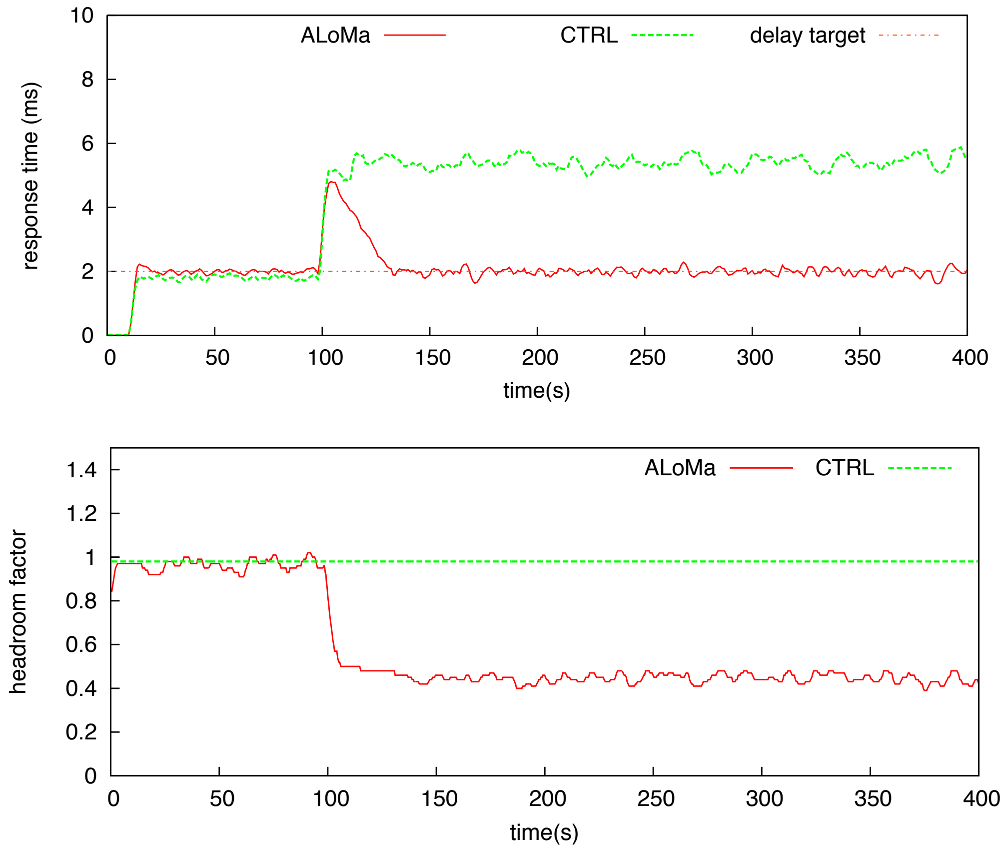
- We use heuristic in the adjustment of H (or L_C)
  - Accommodating system fluctuation and the inherent lag of the statistics

$$L_{C_{new}} = L_C \pm \frac{\log_2(z+1)}{z} |L - L_C|$$

$$\text{where z } = \begin{cases} \frac{|L-L_C|}{L_C}.100 & \text{if } \frac{|L-L_C|}{L_C}.100 \geq 1 \\ 1 & \text{otherwise} \end{cases}$$

  - Principle: bigger the difference, smaller the % of change but bigger in absolute value of change

ALoMa – Performance Evaluation



Effect of environment changes on CTRL [Tu et al.] and adaptation of ALoMa. Total data loss for ALoMa and CTRL is 62.98% and 62.69%, respectively

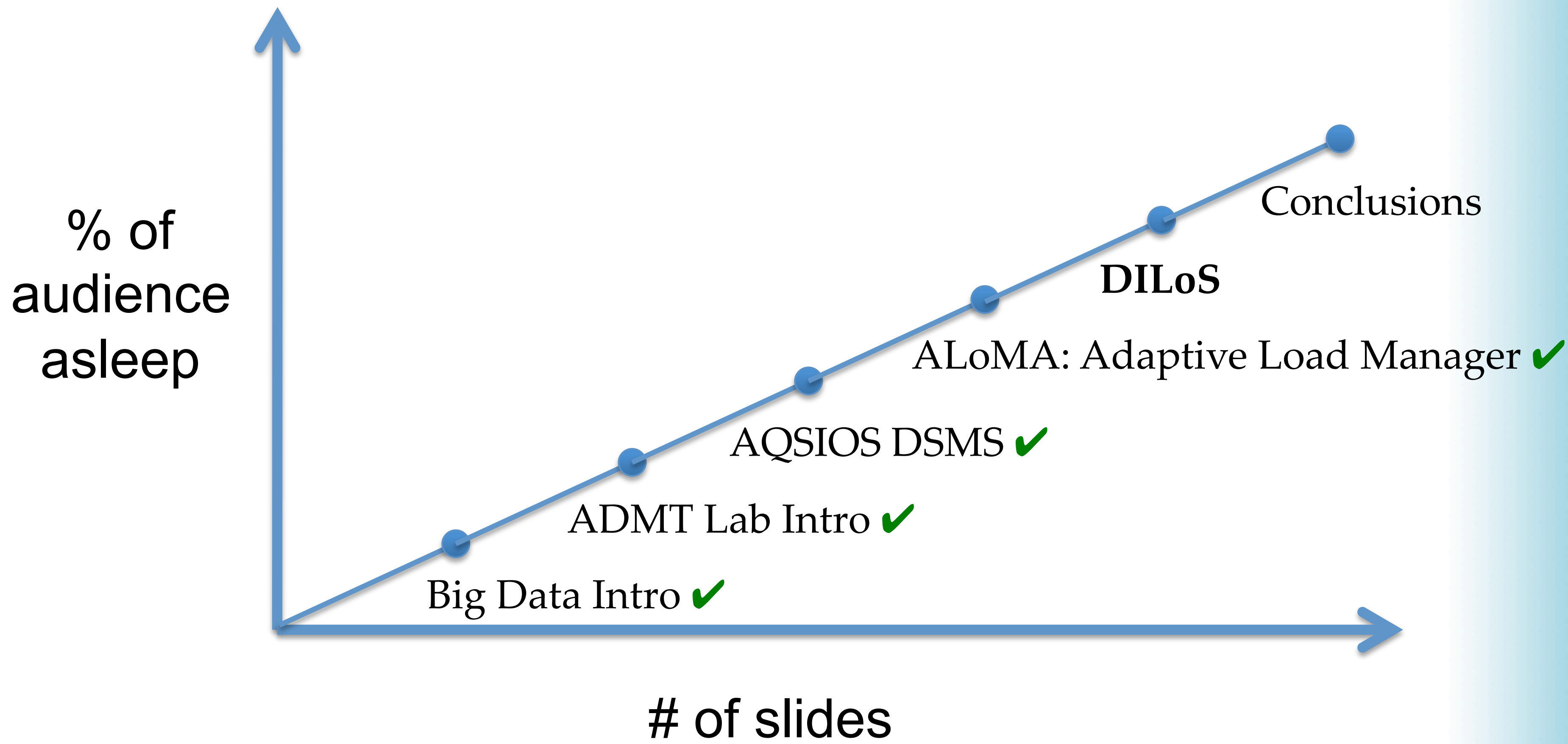
ALoMa

- We showed how ALoMa can *automatically recognize the system capacity* spent on query processing

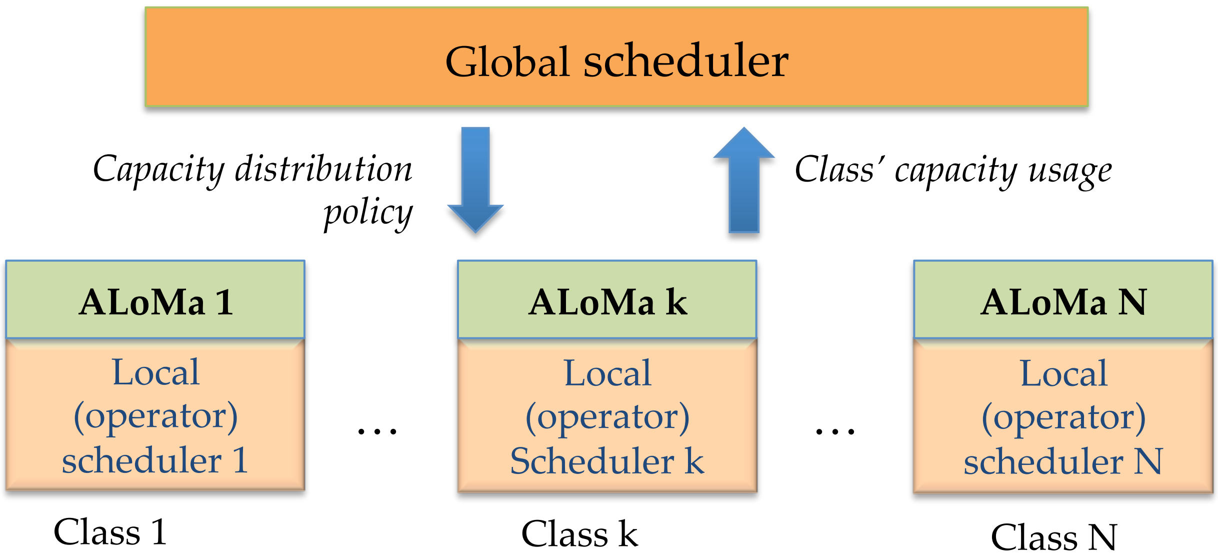
- ALoMa's other important advantages over the state-of-the-art

Ideal properties	ALoMa	CTRL	Aurora
Aware of delay target	✔	✔	
Auto-adjusting of H	✔		
Applicable to all query networks	✔		✔
Independent of scheduler	✔		✔

Roadmap


% of audience asleep
Conclusions
DILoS
ALoMA: Adaptive Load Manager ✔
AQSIOS DSMS ✔
ADMT Lab Intro ✔
Big Data Intro ✔
# of slides

Back to DILoS Framework

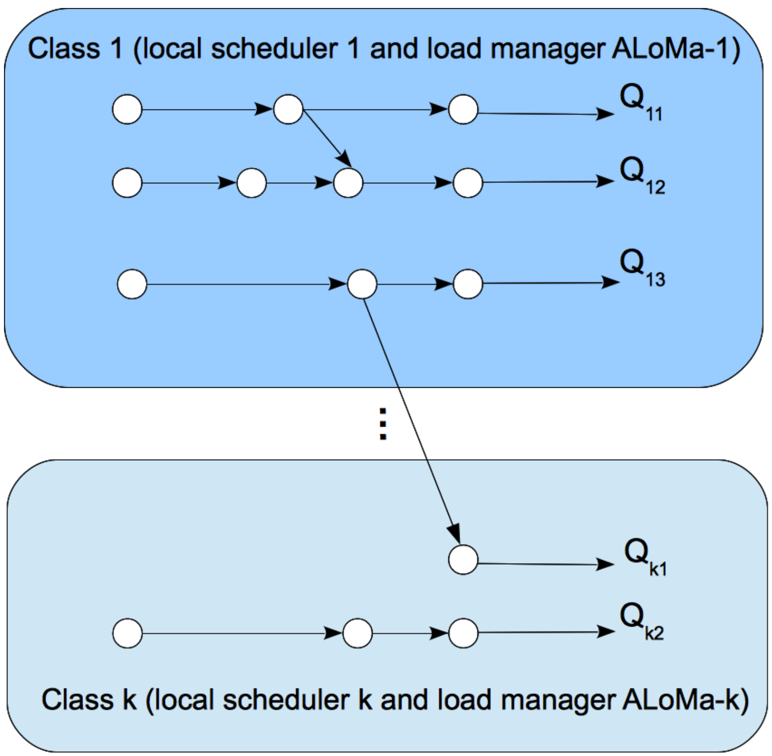


Scheduling Policy

- A concrete policy implemented:
  - A class with priority P_k is guaranteed a share of $P_k \Big/ \sum_{i=1}^{N} P_i$ of total system processing capacity if needed.
    - Adopted from CQC [Al Moakar et al., 2009]
  - Redundant capacity from a class is distributed to other classes in need with "highest priority first"
- Different policies can be plugged in, for example:
  - Absolute priority for higher-priority class:
    - Higher class can use as much of the available capacity as needed
  - Relative priority with workload consideration
    - Higher class receives better QoD regardless of its workload

Inter-class Sharing

- ***Congestion*** can happen when a higher-priority class share a query segment with a lower-priority one under class-based scheduling



- The shared segment receives the higher-priority as it should
- However, the higher-priority class is blocked waiting for the lower priority one to consume the intermediate result

→ DILoS naturally provides a solution, enabling inter-class operator sharing

<u>Claim:</u> *As long as the load of the lower-priority class is controlled to its capacity, congestion will not happen*

Experiments

Experimental Settings

- AQSIOS DSMS prototype

- Three **classes 1, 2, 3** of **priorities 6, 3, 1**; 6 is the highest
- All classes have the same workload of 11 queries
- **Worst-case QoS** of class 1, 2, 3 is **300, 400, 500 ms**

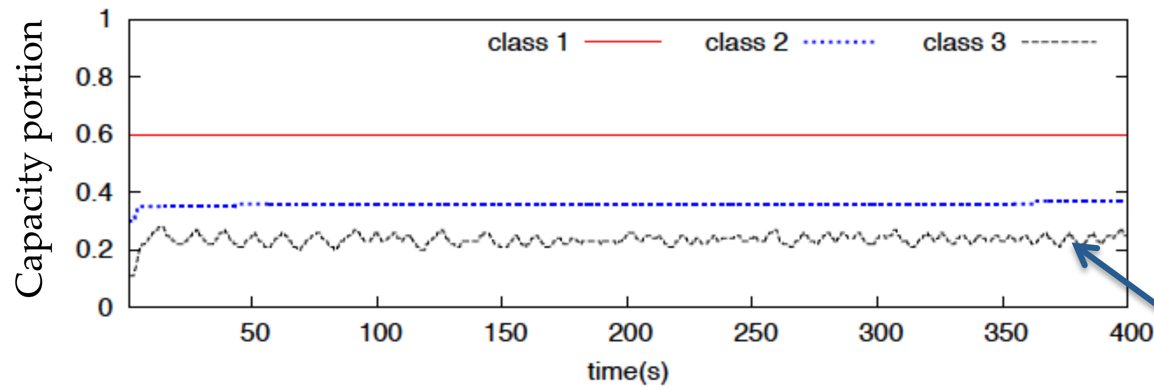
- Input rate:
  - Constant, step changes, and real input trace for class 1
  - Constant input rate for class 2 and 3, at a level that would overload the classes within its assigned capacity.

Result with Constant Input Rate

	Average response time (ms)			Average data loss (%)		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
No load manager	3.40	3.53	56541.69	0	0	0

Understand the Benefit of the Synergy

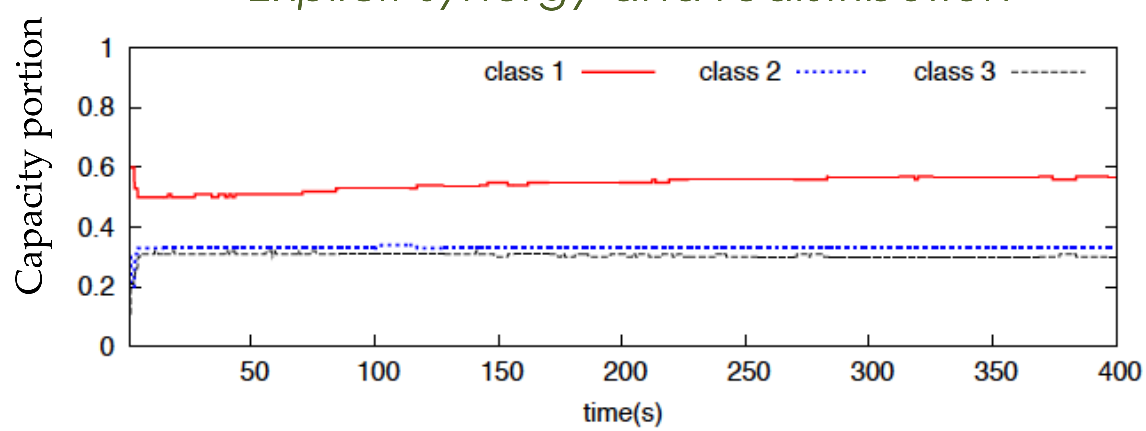
Implicit redistribution observed without explicit synergy



Data loss:
- Class 1: 0 %
- Class 2: 0 %
- Class 3: 35.9

Higher than 0.1!

Explicit synergy and redistribution



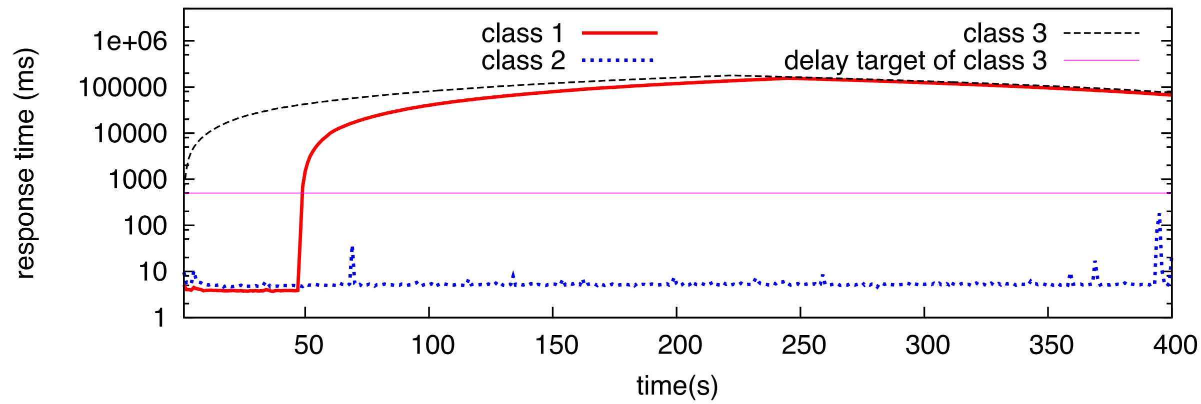
Data loss:
- Class 1: 0 %
- Class 2: 0 %
- Class 3: **0%**

→ *Better capacity usage by exploiting batch processing!*

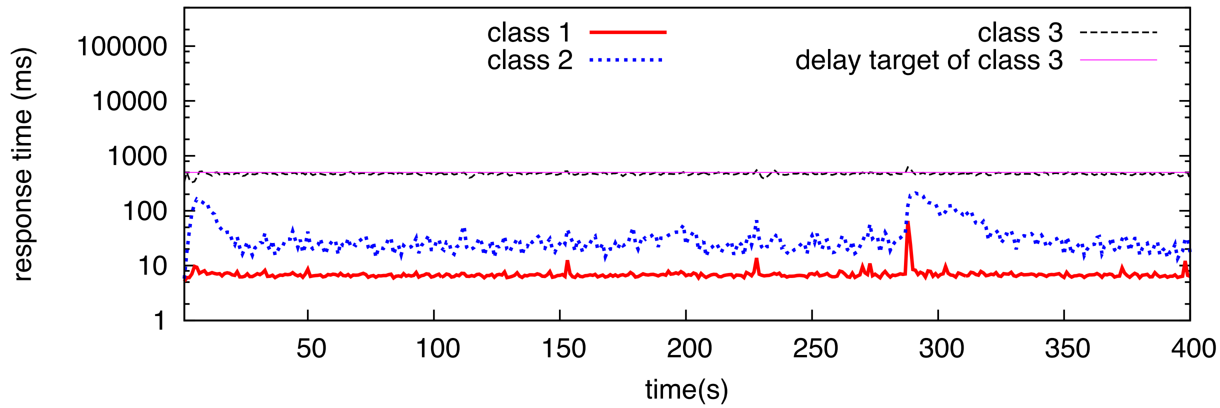
Enabling inter-class sharing

Class 1 shares a query segment with class 3 under a class-based scheduling policy (CQC [Al Moakar et al., 2011]) (constant input rate)

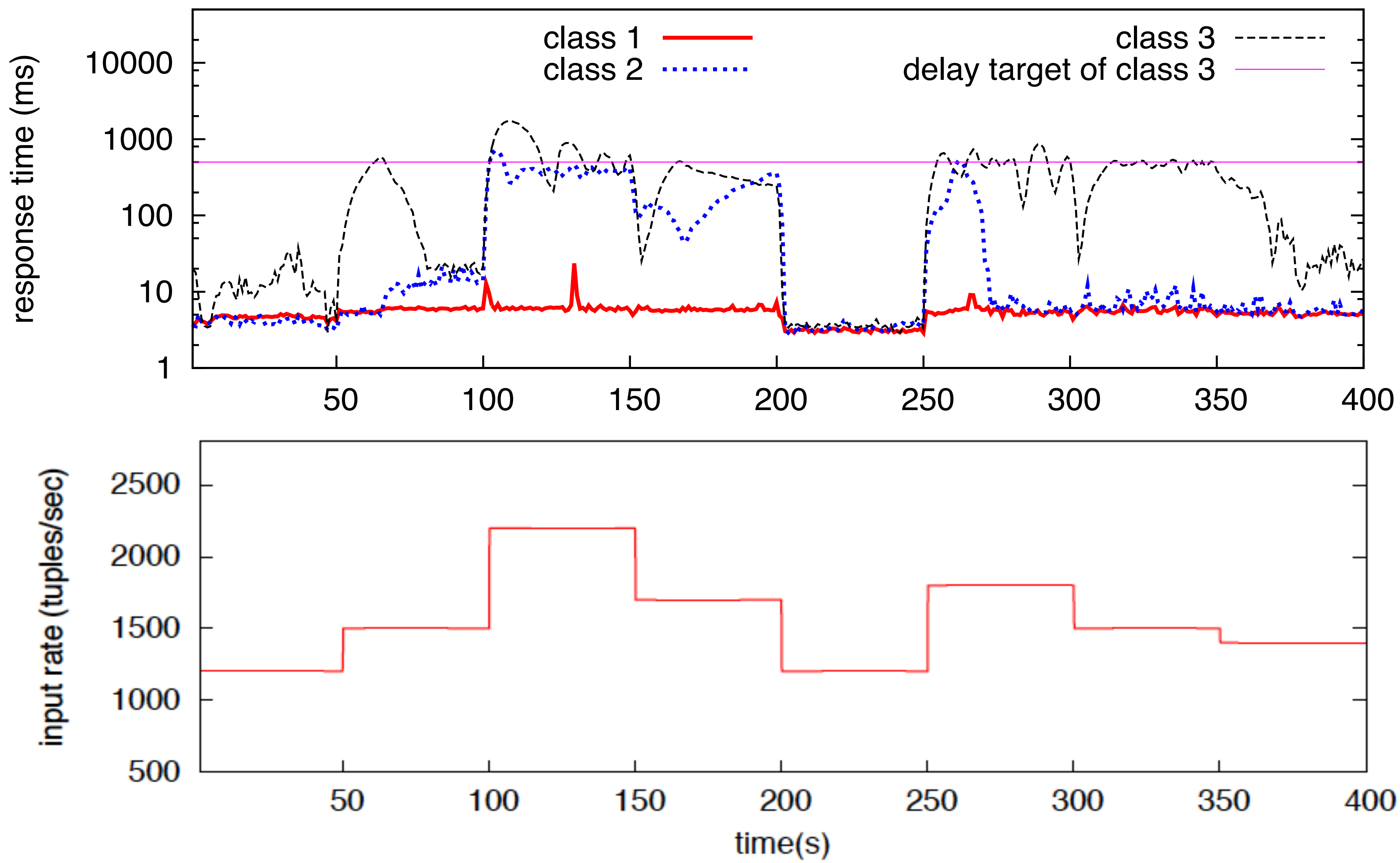
Congestion



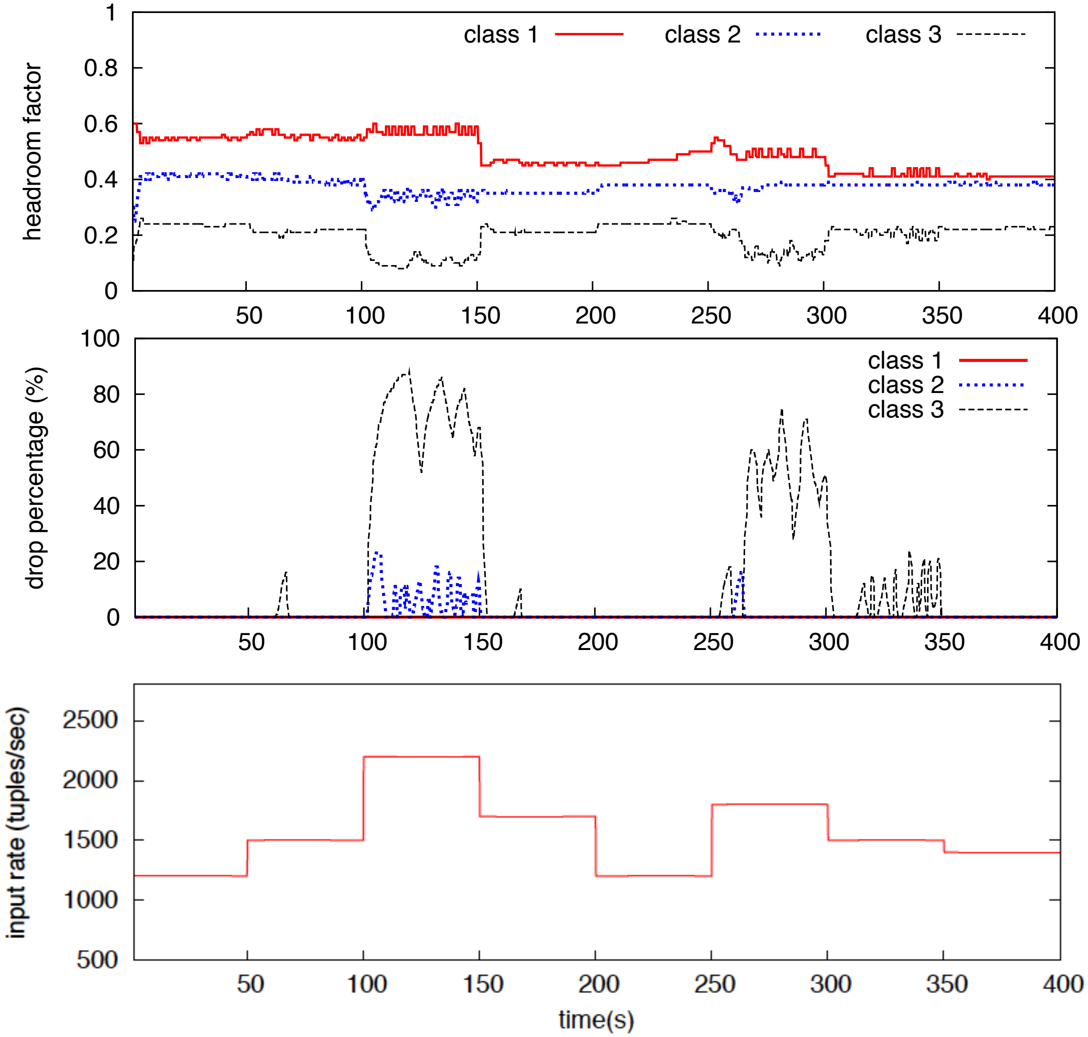
We solved it with DILoS



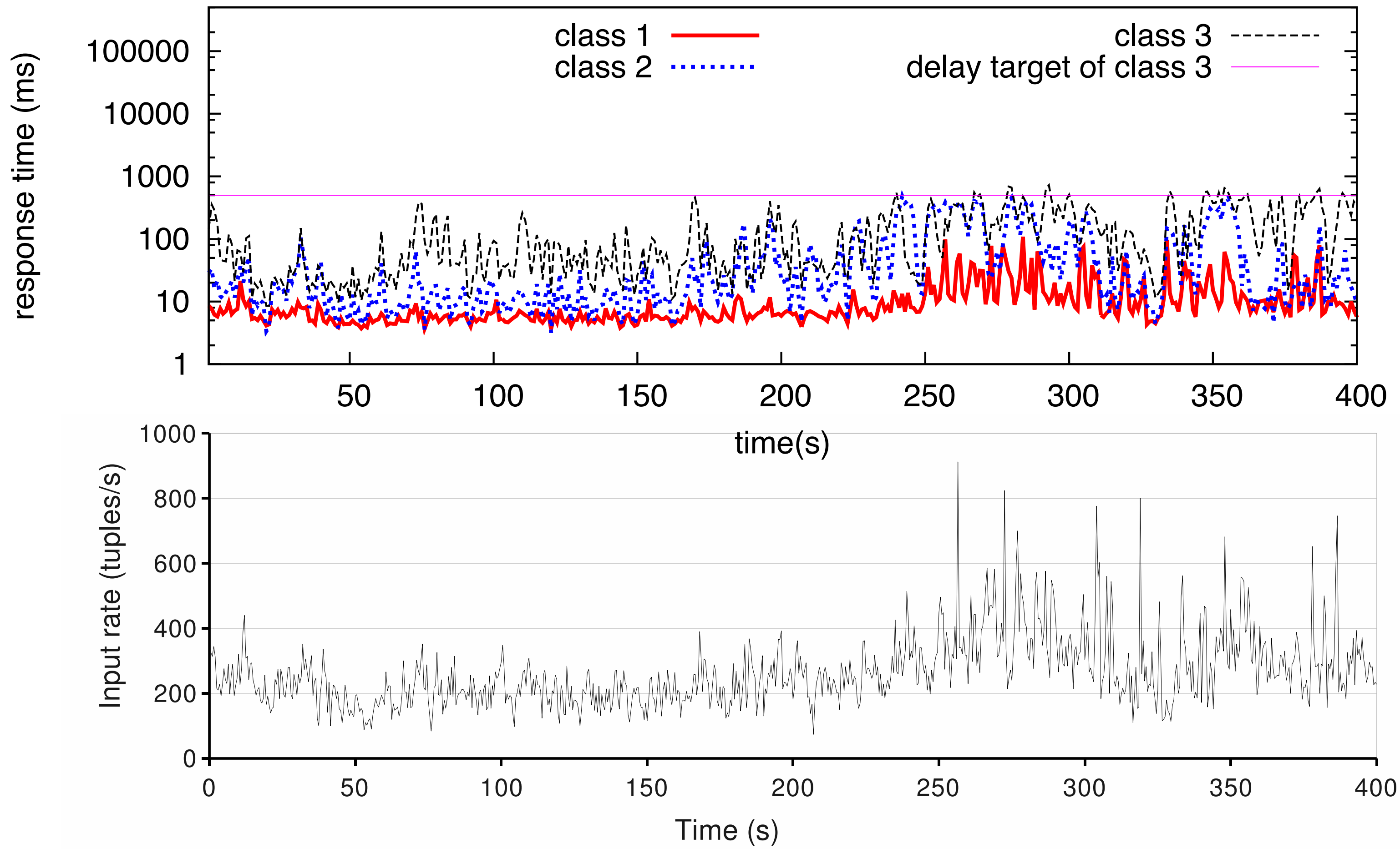
Result with Step Changes in Class 1's Input Rate



Result with Step Changes in Class 1's Input Rate



Result with Real Input Rate for Class 1

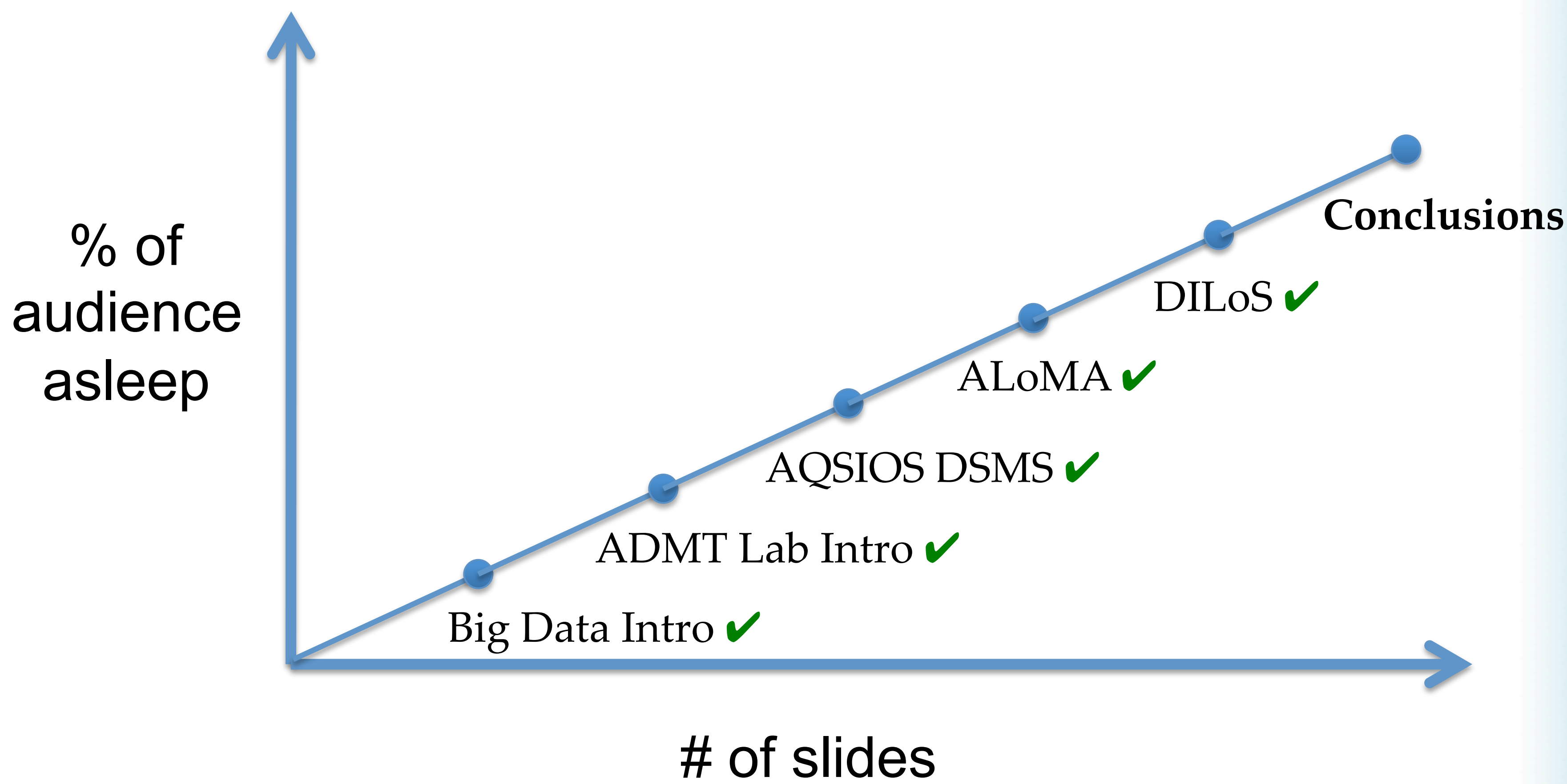


The real input is the trace of TCP packages to and from The Berkeley Lab (http://ita.ee.lbl.gov/html/contrib/LBL- PKT.html)

Result with Real Input Rate for Class 1's

	Average response time (ms)			Average data loss (%)		
	Class 1	Class 2	Class 3	Class 1	Class 2	Class 3
No synergy (& no sharing)	22.31	68.23	300.91	0.01	0.79	21.67
DILoS without sharing	25.69	76.86	122.66	0.46	0.68	8.70
DILoS with sharing	25.03	70.29	127.28	0.44	0.82	6.54

Roadmap



% of audience asleep
Conclusions
DILoS ✔
ALoMA ✔
AQSIOS DSMS ✔
ADMT Lab Intro ✔
Big Data Intro ✔
# of slides

Conclusions

Volume Velocity
Variability

- Advantages of DILoS:
  - Seamless integration:
    - The load manager ***detects*** and ***follows exactly*** the current priority enforcement of the global scheduler
  - Global scheduling decision improved
    - ***Explicitly control the distribution*** of available capacity
    - ***Exploit batch processing*** to increase capacity utilization
    - Enable inter-class sharing to maximize the chance for query optimization
  - Different priority policies can be plugged in
- Future works:
  - Synergy with priority-based memory management
  - Consider advanced architecture (multi-core, cloud)

A (Big) Team Effort

Faculty

- Panos Chrysanthis
- Alexandros Labrinidis
- Adam Lee
- Kirk Pruhs

FUNDING (DILoS)

- NSF IIS-0534531
- NSF CAREER IIS-0746696
- EMC/Greenplum
- Andrew Mellon Predoctoral Fellowship

Students

- Lory Al Moakar
- Di Bao
- Nick Farnan
- Roxana Gheorghiu
- Shenoda Guirguis
- Qinglan Li
- Panickos Neophytou
- Thao Pham
- Mohamed Sharaf
- Matt Schroeder
- Nikhil Venkatesh

http://db.cs.pitt.edu