ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

Consistency Rationing: *Pay only when it matters*

Tim Kraska, Martin Hentschel, Gustavo Alonso, Donald Kossmann

You own a Jewelry Store

- Items are highly valuable
- Any damage is expensive
- Requires protection
  - alarm systems / guards
  - insurance
  - security plans
- Items are handled carefully
  - Precise book-keeping
  - Demand planning etc.

It is expensive!!!

You own a Kiosk

- Items are not as valuable
- Strong protection is not required
- It is easier
- Less expensive
- It scales better

Of course,
more can go wrong!

You own a Kiosk

- Items are not as valuable
- Strong protection i…
- …is easier
- Less expensive
- It scales better

Protection Cost vs. Damage Cost

Of course,
more can go wrong!

Consistency Rationing - Idea

- Strong consistency is expensive
- ACID prevents scaling & availability (CAP theorem)
- But not everything is worth gold!

Transaction Cost vs. Inconsistency Cost

1. Use ABC-analysis to categorize the data
2. Apply different consistency strategies per category

Cloud Computing and Consistency

Traditional architecture

- expensive hardware (mainframes)
- one or few machines (RAC)
- Single data center

Strong consistency requires:
few messages between machines over fast interlinks

Cloud architecture

- COTS hardware
- thousands of machines
- Often multiple data center

Strong consistency requires:
many messages across data centers (and expensive service calls)

Outline

- Consistency Rationing
- Adaptive Guarantees
- Implementation & Experiments
- Conclusion and Future Work

Outline

- **Consistency Rationing**
- Adaptive Guarantees
- Implementation & Experiments
- Conclusion and Future Work

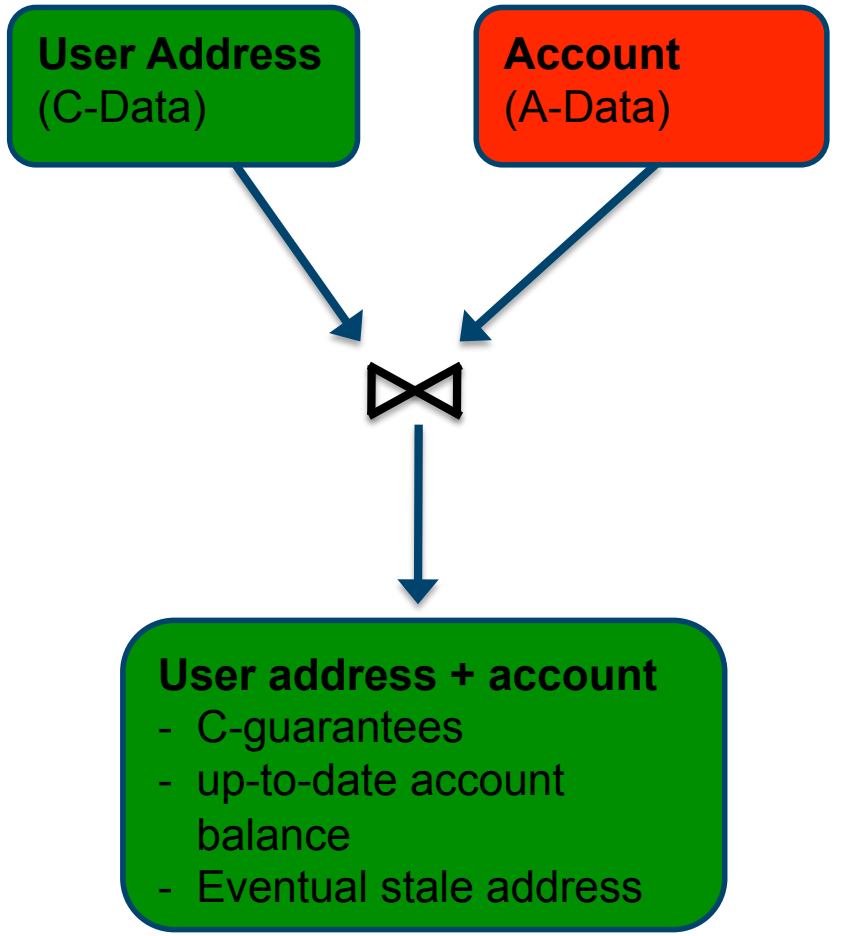
Consistency Rationing (Pre-Classification)

Category	Characteristics	Guarantees	Use Case
A-Data	**Inconsistencies are expensive and/ or cannot be resolved**	**Serializable (2PL)** •Pessimistic CC as conflicts are expected •No staleness: always up-to-date data	• Atomic bomb • Bank data???
B-Data	**Violations might be tolerable**	**Adaptive guarantees** •Switches between A & C guarantees •Depends on some policy	• Product inventory • Tickets
C-Data	**No inconsistency cost and/or inconsistency cannot occur**	**Session consistency** •Practical •Still eventually consistent •Allows for aggressive caching	• Recom-mendations • Customer profiles • Products

In analogy to the ABC-analysis from Inventory Rationing

Consistency Rationing - Transactions

- Consistency guarantees per category instead of transaction level
  - Transactions are still allowed to overwrite consistency requirement
- Different categories can mix in a single operation/transaction
- For joins, unions, etc, the lowest category wins



Outline

- Consistency Rationing
- **Adaptive Guarantees**
- Implementation & Experiments
- Conclusion and Future Work

Adaptive Guarantees for B-Data

- B-data: Inconsistency has a cost, but it might be tolerable
- Often the bottleneck in the system
- Here, we can make big improvements
- Let B-data automatically switch between A and C guarantees
- Use policy to optimize:

Transaction Cost vs. Inconsistency Cost

B-Data Consistency Classes

	Characteristics	Use Cases	Policies
General	Non-uniform conflict rates	Collaborative editing	General policy
Value Constraint	• Updates are commutative • A value constraint/ limit exists	• Web shop • Ticket reservation	• Fixed threshold policy • Demarcation policy • Dynamic policy
Time-Based	Consistency does not matter much until a certain moment in time	Auction systems	Time-based policy
Value-Based	Consistency requirements depend on the data value	Plane-ticket reservation (business vs. economy class)	Value-based policy

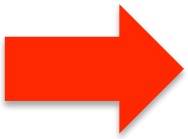
B-Data Consistency Classes

	Characteristics	Use Cases	Policies
General	Non-uniform conflict rates	Collaborative editing	**General policy**
Value Constraint	• Updates are commutative • A value constraint/ limit exists	• Web shop • Ticket reservation	• **Fixed threshold policy** • **Demarcation policy** • **Dynamic policy**
Time-Based	Consistency does not matter much until a certain moment in time	Auction systems	Time-based policy
Value-Based	Consistency requirements depend on the data value	Plane-ticket reservation (business vs. economy class)	Value-based policy

General Policy - Idea

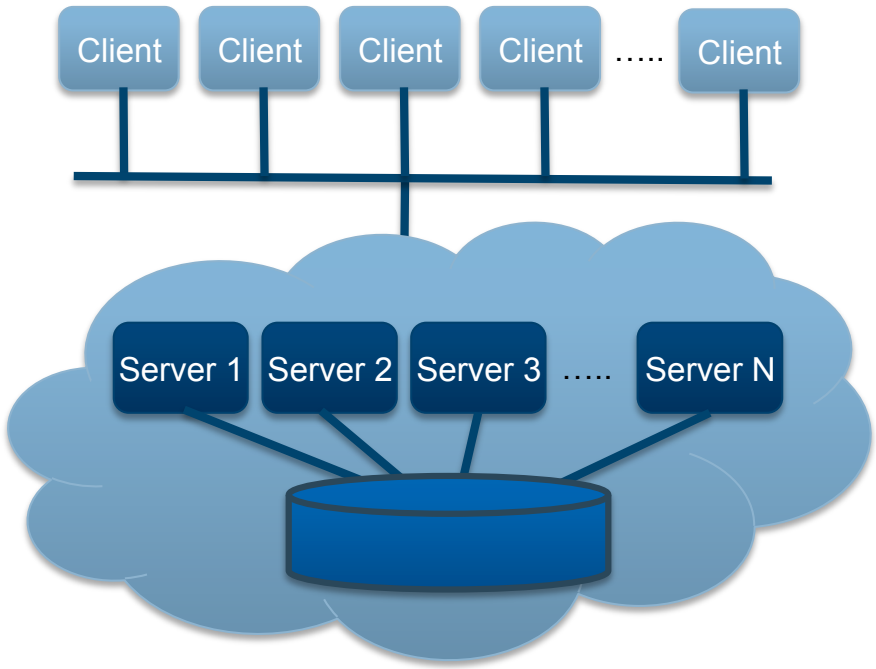
Apply strong consistency protocols only if the likelihood of a conflict is high

1. Gather temporal statistics at runtime
2. Derive the likelihood of a conflict by means of a simple stochastic model
3. Use strong consistency if the likelihood of a conflict is higher than a certain threshold



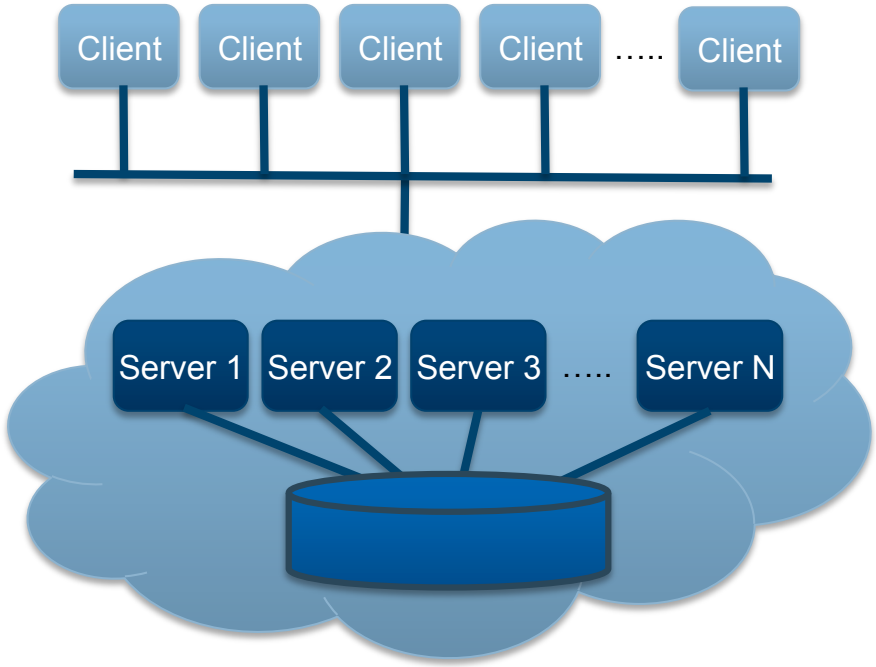
Consistency becomes a probabilistic guarantee

General Policy - Model



- n servers
- Servers cache data with cache interval CI
- Load equally distributed
- Two updates considered as a conflict
- Conflicts for A and B data can be detected and resolved after every CI
- Every server makes local decisions
  (no synchronization)

General Policy - Model



1. Likelihood of a conflict inside one CI

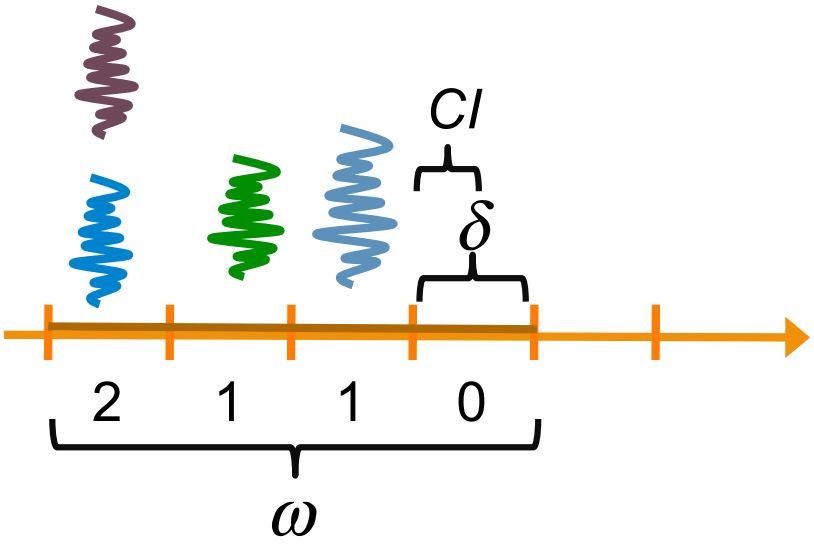
$$P_c(X) = \underbrace{P(X>1)}_{(i)} - \underbrace{\sum_{k=2}^{\infty}\left(P(X=k)\left(\frac{1}{n}\right)^{k-1}\right)}_{(ii)}$$

2. Assuming further a Poisson process (simplification)

$$P_\lambda(X=k) = \frac{\lambda^k}{k!}e^{-\lambda}$$

$$P_c(X) = \underbrace{\left(1-e^{-\lambda}(1+\lambda)\right)}_{(iii)} - \underbrace{\sum_{k=2}^{\infty}\left(\frac{\lambda^k}{k!}e^{-\lambda}\left(\frac{1}{n}\right)^{k-1}\right)}_{(iv)}$$

General Policy – Temporal Statistics



$CI=1$
$\delta=2$
$\bar{x}=1$
$n=4$

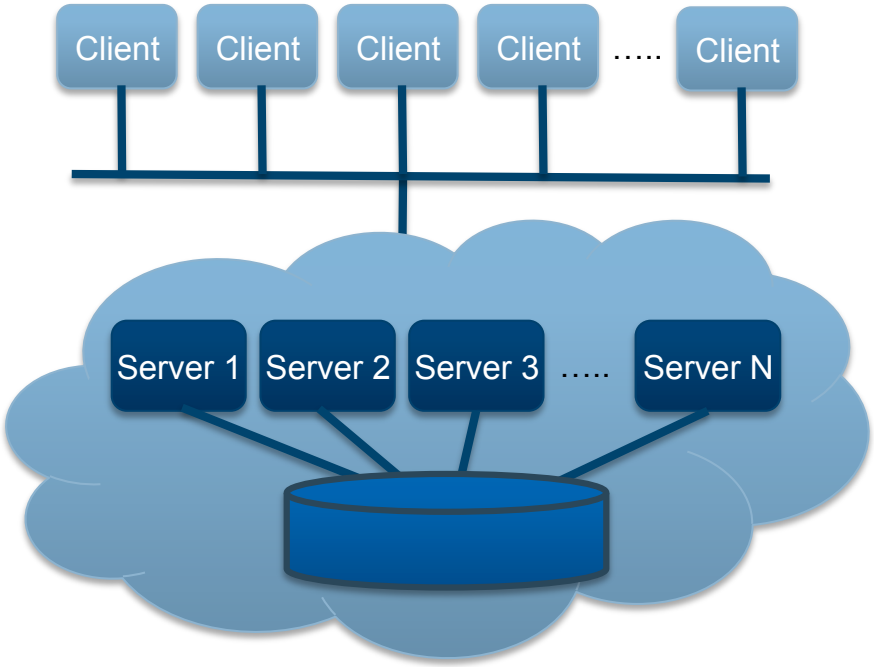
$\lambda=2$

On every server: Collect update rate for a window ω with sliding factor δ per item

- Window size ω is a smoothing factor
- Sliding factor δ is a multiple of CI
- Calculate average update rate $\bar{x}$ over all slices inside a window
- Derive the global state from local information

$$\lambda=\frac{\bar{x}n}{\delta}$$

General Policy – Setting the Threshold



- Use strong consistency protocol if the savings are bigger than the penalty cost

$$C_A - C_C > E_O(X)$$

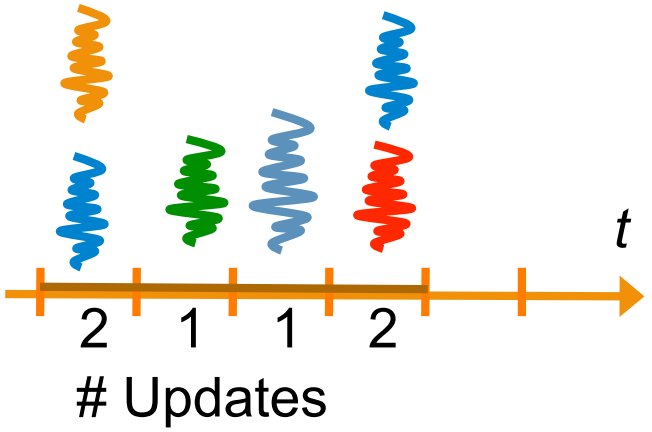
$$C_A - C_C > P_C * C_O$$

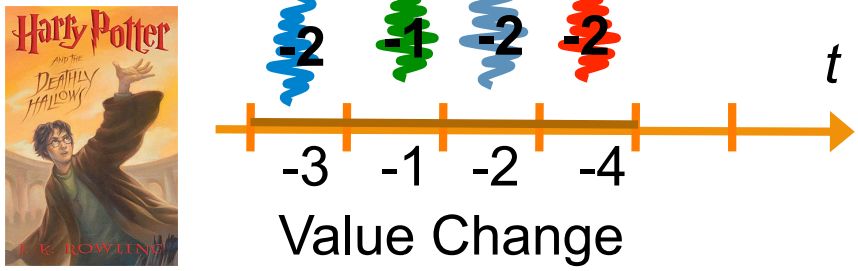
$$\frac{C_A - C_C}{C_O} > P_C$$

- Cost of A(CID) transaction C_A
- Cost of C transaction C_C
- Cost of inconsistency C_O

Value Constraint

- Use Cases:
  - Web shop
  - Ticket reservation
- Commutative updates
- Consistency is defined by a value constraint (e.g., stock >= 0)
- Without loss of generality, the limit is assumed to be 0





Value Constraint - Policies

Fixed threshold policy

Use strong consistency protocol if value drops below a fixed threshold *T*

$$v - \Delta \leq T$$

v = Value of $attr_k$ of record r

Δ = Change by the current TRX

Demarcation policy

- Divide value among servers
- Every server gets a share of the value

$$T = v - \left\lfloor \frac{v}{n} \right\rfloor$$

- Adjust shares after *CI* expires
- Still, doesn't guarantee strong consistency

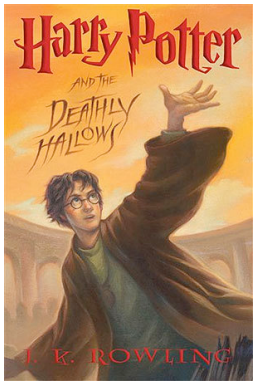
Value Constraint – Dynamic Policy

- Apply strong consistency protocols only if the likelihood of violating a value constraint becomes high
- Likelihood of a conflict is

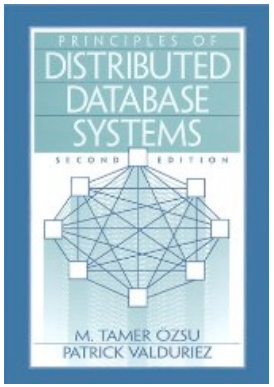
$$P_C = P(T - Y < 0)$$

T = Threshold when to switch

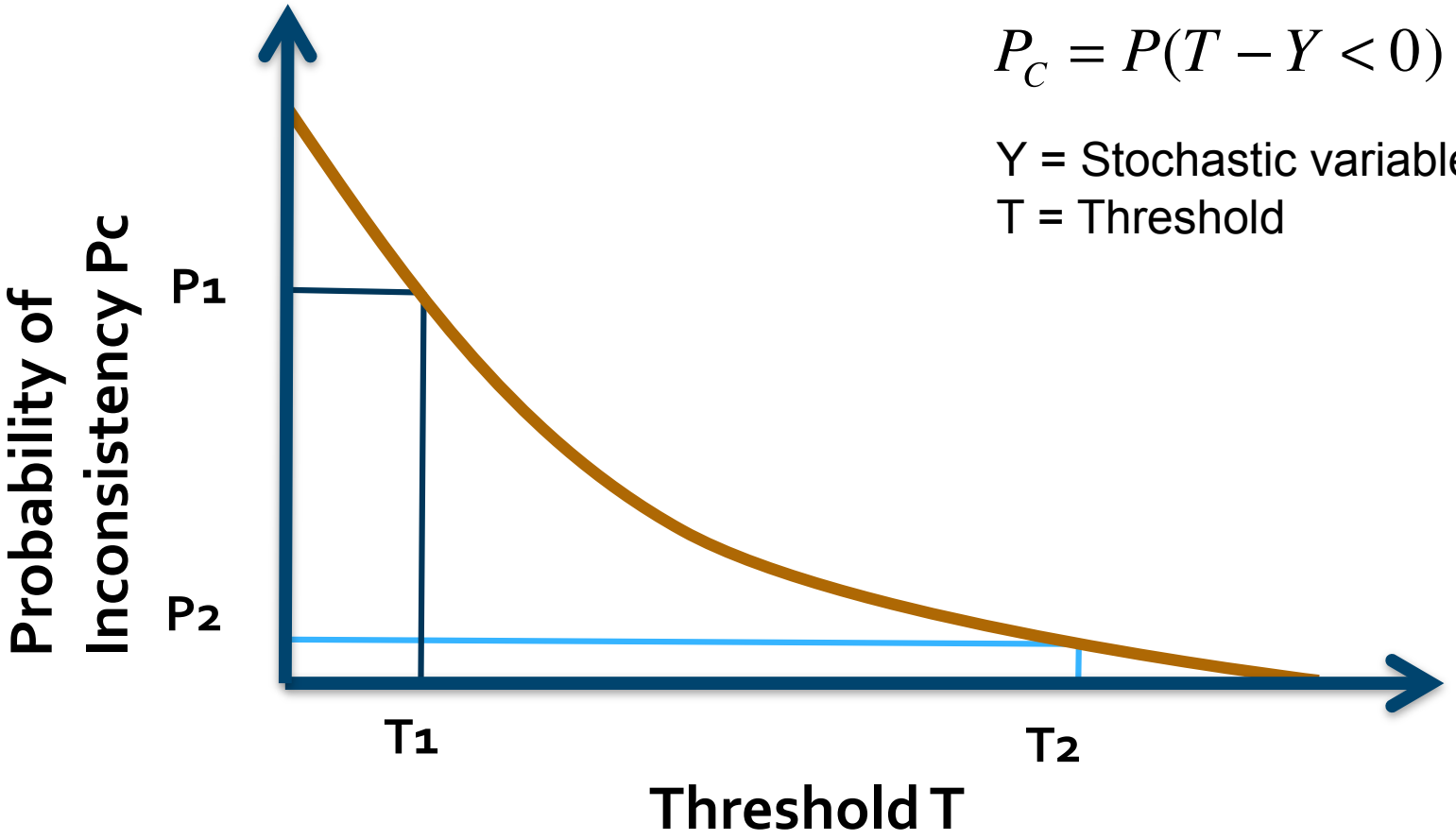
Y = Stochastic variable corresponding to the sum of Δ within *CI*



vs



Value Constraint – Dynamic Policy



$$P_C = P(T - Y < 0)$$

Y = Stochastic variable
T = Threshold

Dynamic policy – Temporal statistics

- Sliding factor δ is factor of *CI* (not multiple)
  - The dynamic policy requires the variance
  - Interest is on hot spots → Less time required to gather statistics
- Convoluting the slices to derive the empirical PDF

$$f_{CI} = \underbrace{f * f * \ldots * f}_{CI/\alpha \text{ times}} \qquad f_{CI*n} = \underbrace{f_{CI} * f_{CI} * \ldots * f_{CI}}_{n \text{ times}}$$

- Determine the threshold by means of the CDF

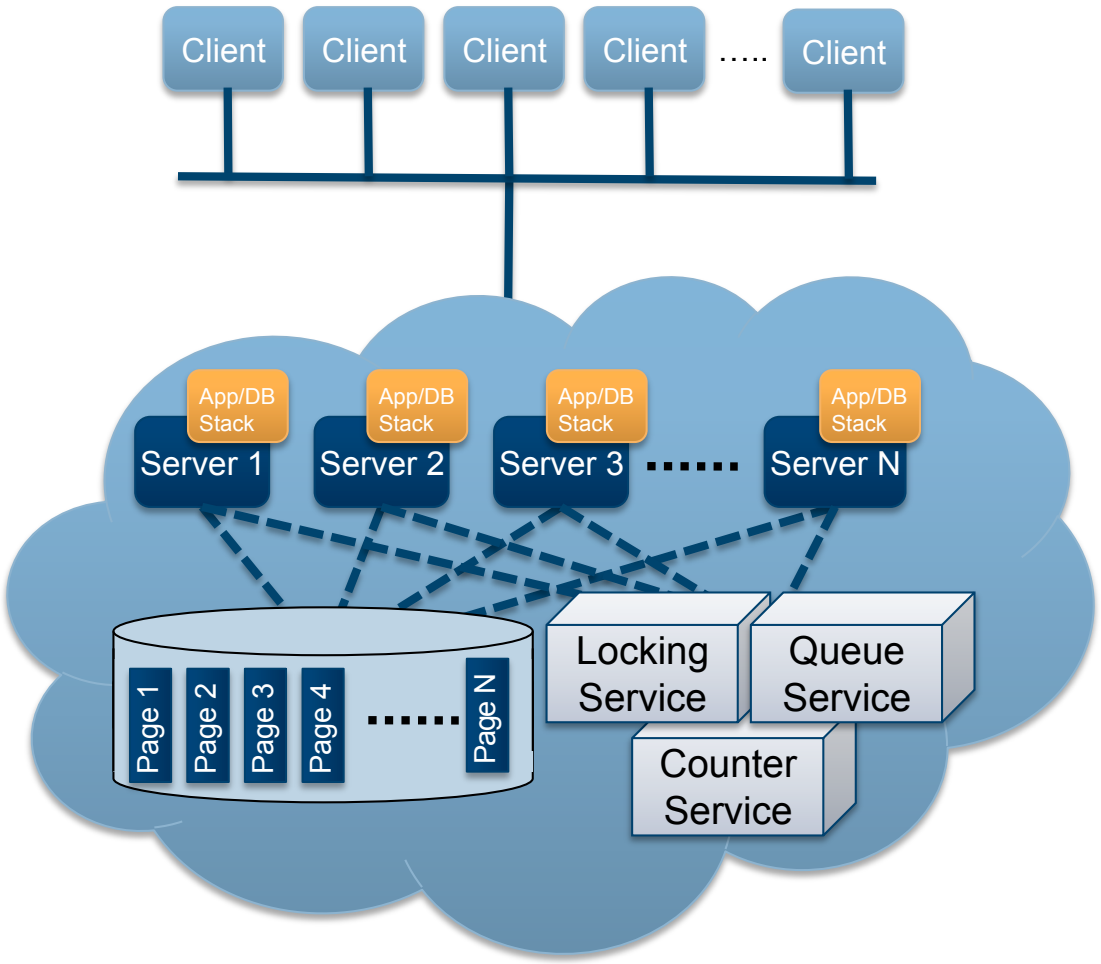
$$\mathrm{F}_{\mathrm{CI*n}}(T) > \mathrm{P}_C(Y) \quad \Rightarrow \quad \mathrm{F}_{\mathrm{CI*n}}(T) > \frac{C_A - C_C}{C_O}$$

- Optimization: Use normal distribution instead of convolution

Outline

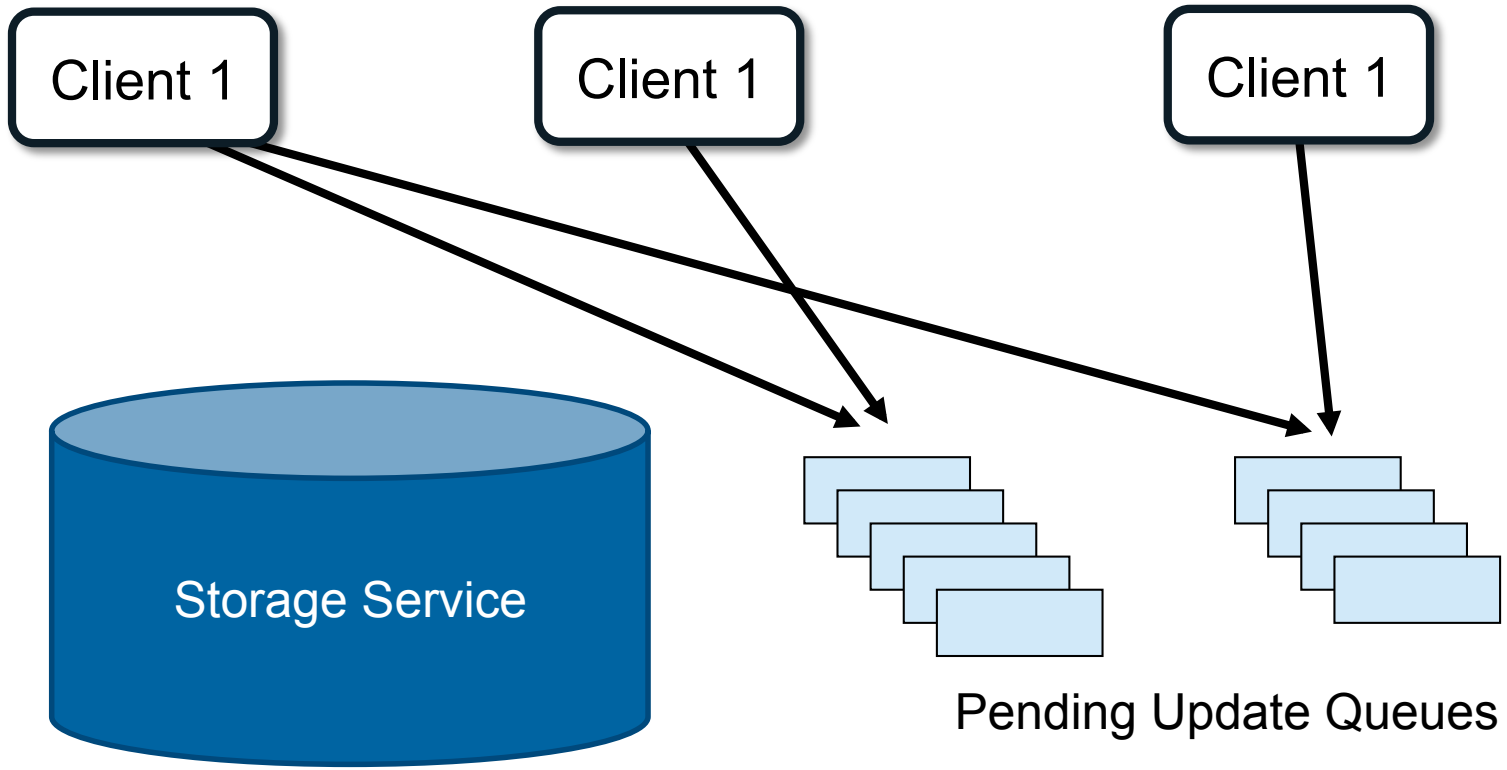
- Consistency Rationing
- Adaptive Guarantees
- **Implementation & Experiments**
- Conclusion and Future Work

Implementation



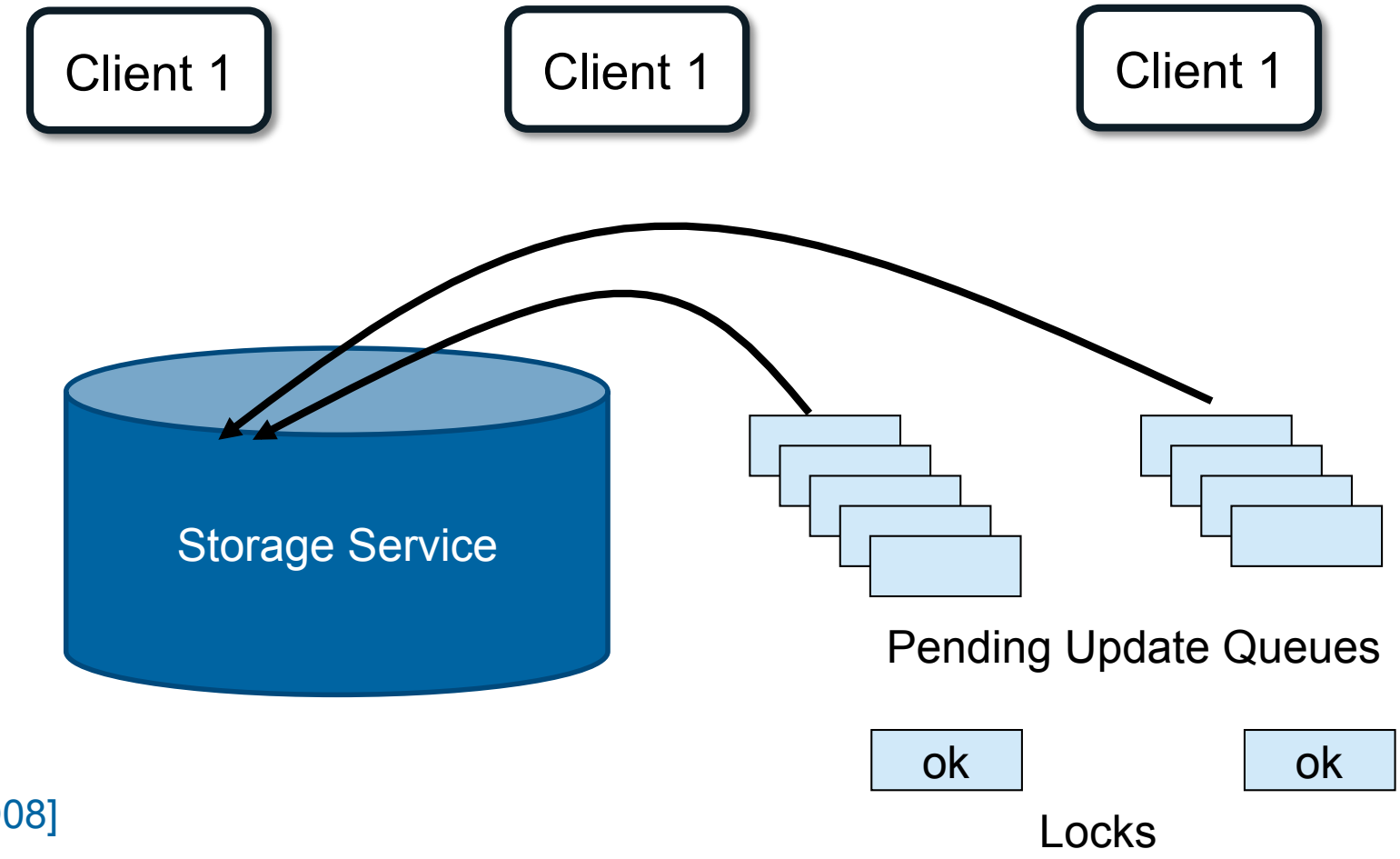
- Architecture of "Building a DB on S3" [Sigmod08]
  - Extended protocols
  - Additional services
    - Locking Service
    - Reliable Queues
    - Counter Service
  - Every call is priced
- Approach is not restricted to this architecture
  - PNUTS
  - Distributed DB
  - Traditional DB

Step 1: Clients commit update records to pending update queues



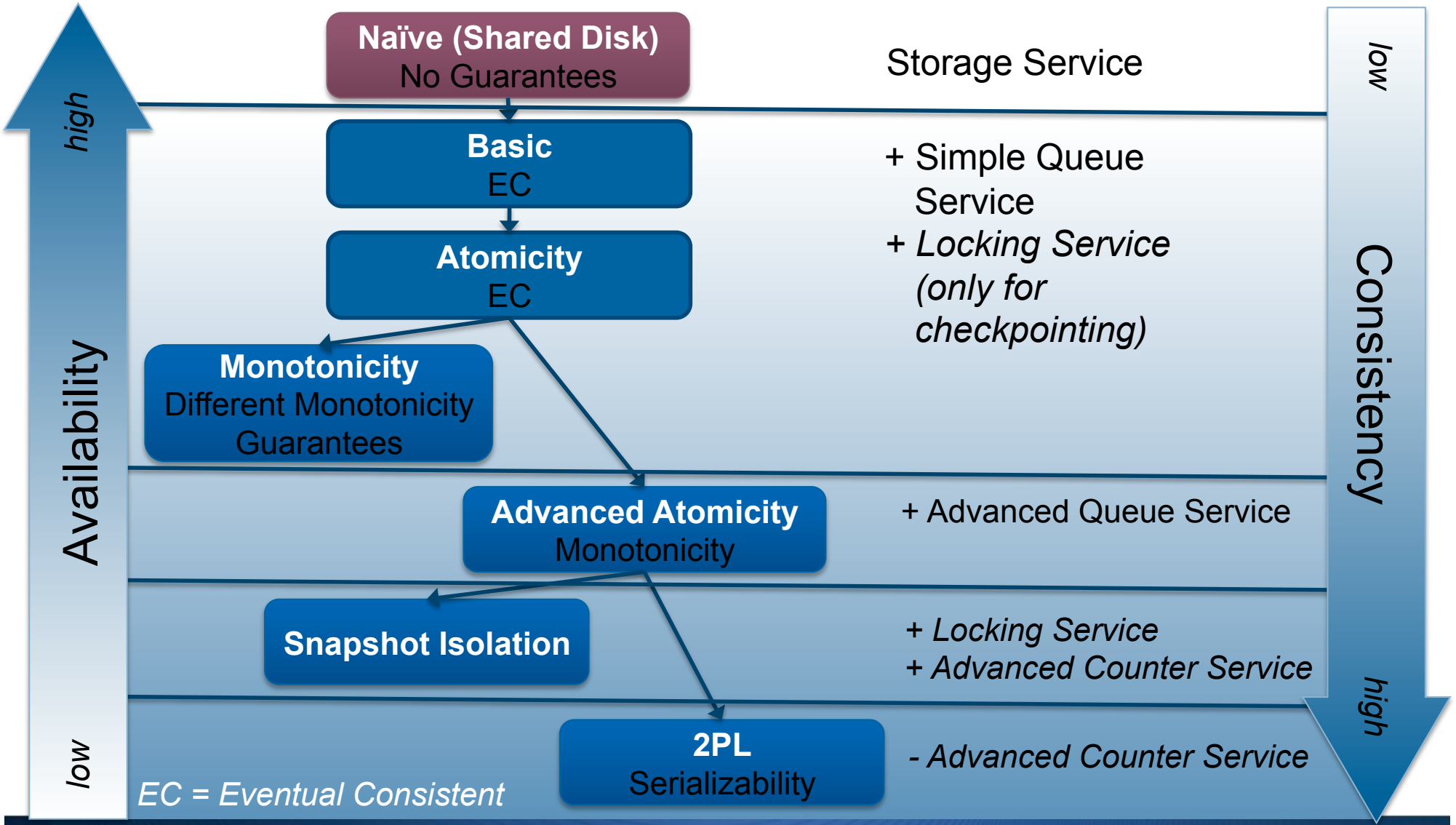
[SIGMOD08]

Step 2: Checkpointing propagates updates from SQS to S3

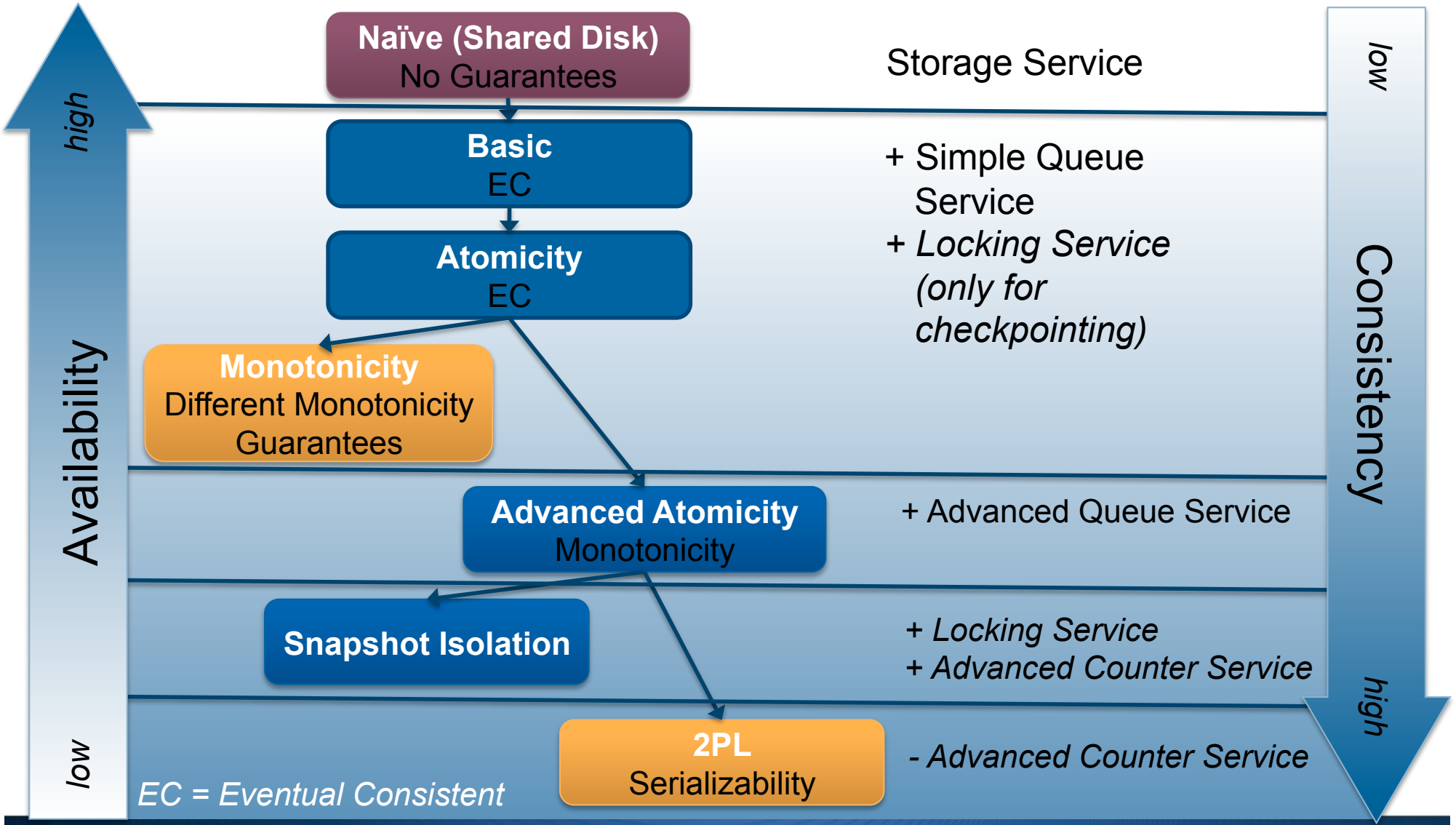


[SIGMOD08]

Levels of Consistency/Availability



Levels of Consistency/Availability



Implementation - Statistics

- Servers make independent decisions
- Logical updates if possible
- General policy
  - Statistics stored at the record
  - Large window size required
  - Assuming a conflict rate of less than 1% → $\lambda \approx 0.22$ updates per CI
  - Assuming a window size of 1 hour and 5 min sliding factor:
    Allows to store the statistics in 48bit per record (4 bit per slide, with value 15 as infinite)
- Dynamic policy
  - Collects statistics only for hot records; all others are handled with a standard threshold or the General Policy
  - 10,000 hot records with 100 slices require 4MB of space.

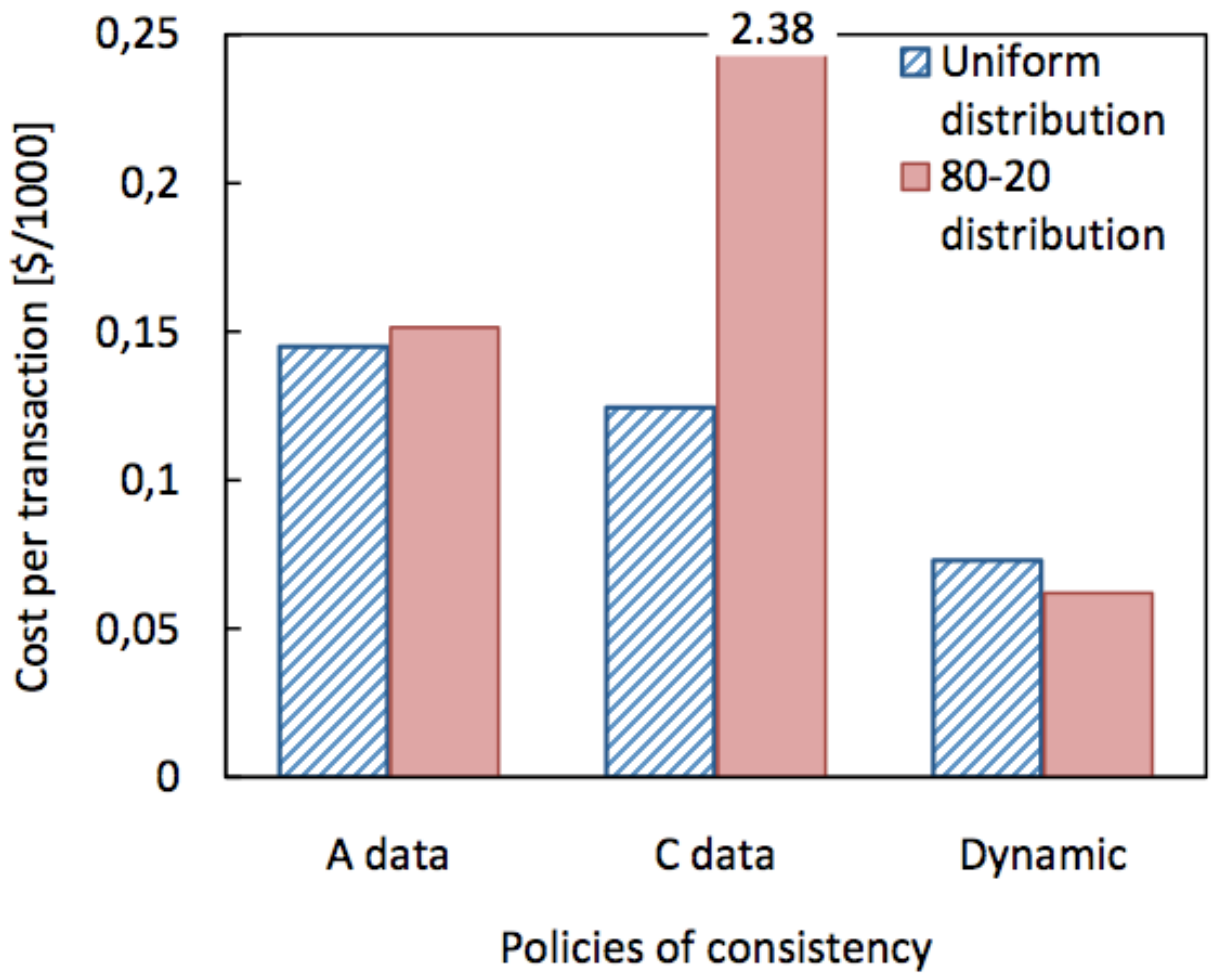
Experiments

Modified TPC-W

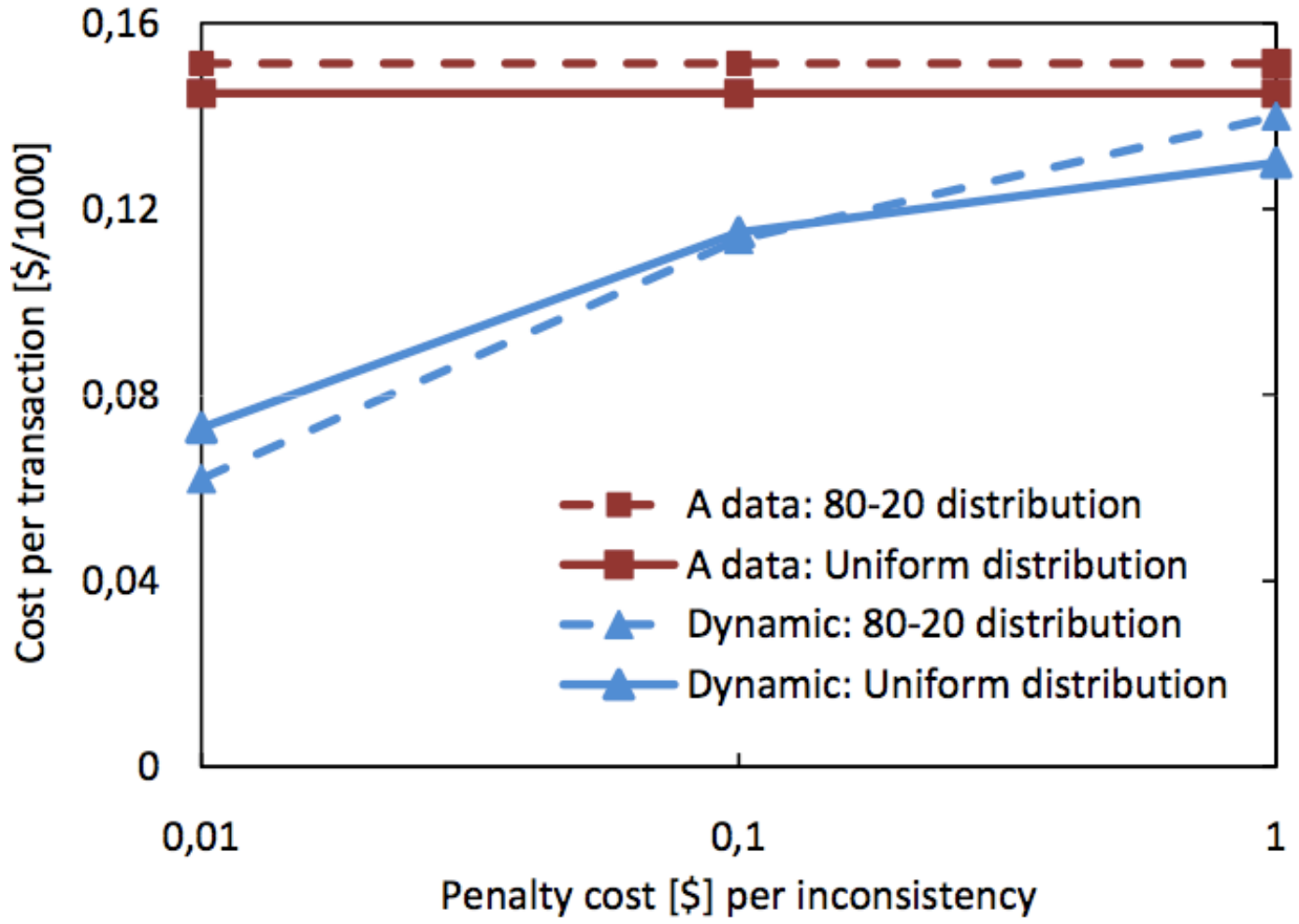
- No refilling of the stock
- Different demand distributions
- Stock is uniformly distributed between 10-100
- Order mix - up to 6 items per basket (80-20 rule)
- 10 app servers, 1000 products
- $C_o = 0.01\$$ but up to 12,000 products are sold in 300sec
- Rationed consistency

Results represent just one possible scenario!!!

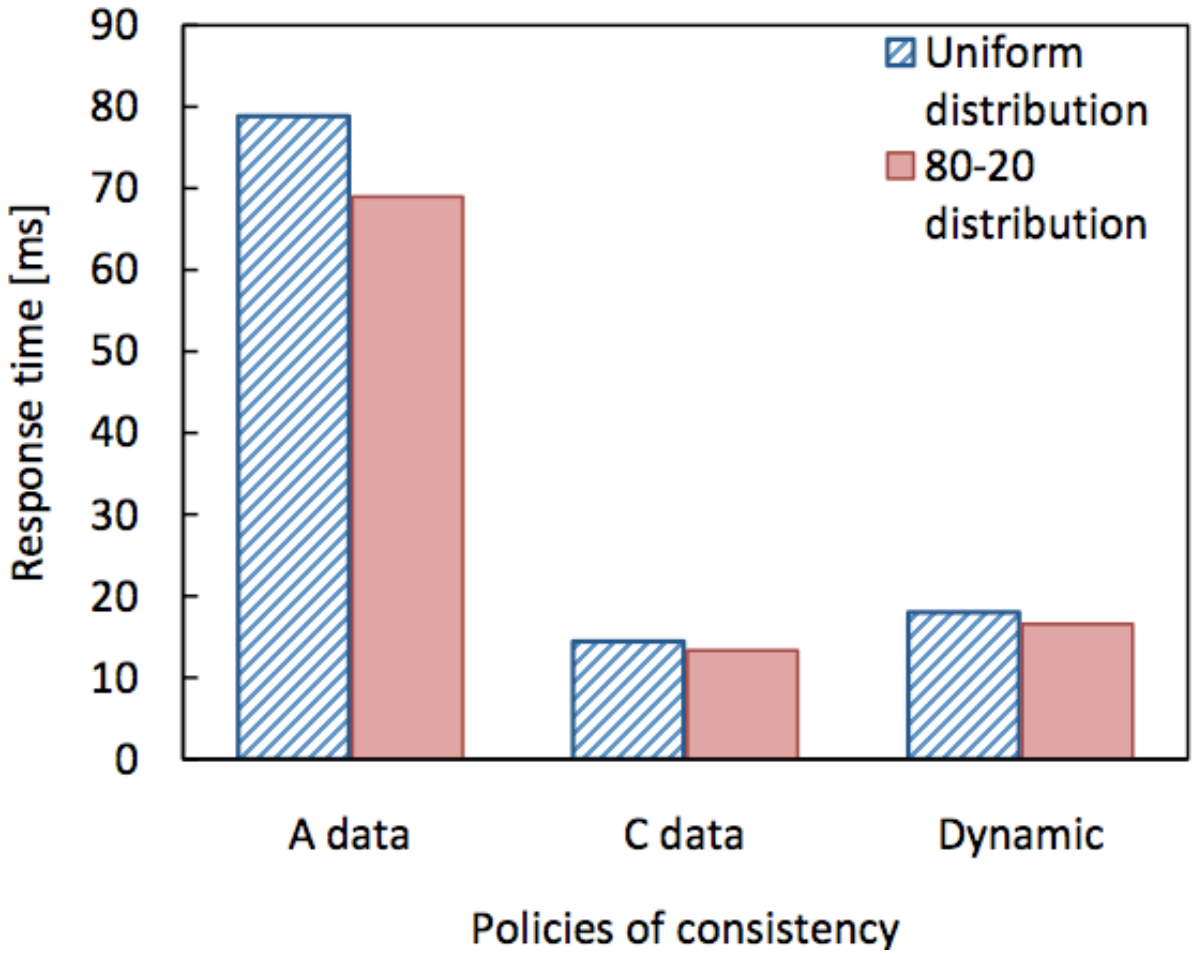
Overall Cost (including the penalty cost) per TRX [$/1000]



Overall Cost per TRX [$/1000]



Response Time [ms]



Conclusion and Future Work

- Rationing the consistency can provide big performance and cost benefits
- With consistency rationing we introduced the notion of probabilistic consistency guarantees
- Self-optimizing system – just the penalty cost is required
- Future work
  - Language support
  - Applying it to other architectures
  - Better and faster statistics
  - Consistency and analytics????