Foundations of Automated Database Tuning

Surajit Chaudhuri — Microsoft Research
Gerhard Weikum — Max Planck Institute for Informatics

Scope and Purpose of This Tutorial

Motivate and enable students and young scientists
to pursue research on the auto-tuning aspect
of autonomic computing

Complementary to
- SIGMOD 02 and VLDB 02 tutorials (Shasha/Bonnet) on tuning techniques for DBAs
- VLDB 04 tutorial (Chaudhuri/Dageville/Lohman) on self-management features of DBMS products

Outline

- Part I: What Is It All About
- Part II: Five Auto-Tuning Paradigms
  1 Auto-Tuning as Tradeoff Elimination
  2 Auto-Tuning as Static Optimization with Deterministic Input
  3 Auto-Tuning as Static Optimization with Stochastic Input
  4 Auto-Tuning as Online Optimization
  5 Auto-Tuning as Feedback Control Loop
- Part III: Wrap-up

Part I: What Is It All About

- The Need for and Nature of Auto-Tuning
- State of the Art
  - Product Features
  - Scientific Principles
- Auto-Tuning Paradigms

Need for Auto-Tuning

- Total cost of ownership (TCO) for DBMS-based IT solution dominated by staff for system admin, management, and tuning
- Increasing complexity of multi-tier application services call for automated management
- DBMS offers hundreds of tuning kobs (system config-time, DB-load-time, startup-time, run-time parameters)

→ DBMS (and multi-tier IT systems) should be **autonomic (self-*)**: self-managing, self-monitoring, self-healing, **self-tuning**

Easy Solutions

- **Throw more hardware (KIWI method)**
  - Use this with caution
  - Where do you throw hardware?
- **Rules of Thumb approach**
  - Finding them is harder than you think
  - May simply not exist – oversimplified wrong solutions are not helpful

Nature of Auto-Tuning

ability to predict

workload $\times$ *config* $\rightarrow$ *performance*

!!! !!! ???

is key to finding the right knob setting

workload $\times$ *config* $\rightarrow$ *performance goal*

!!! ??? !!!

Many difficult ramifications:

- workloads at different levels and time scales
  - app-level vs. internal, long-term steady-state vs. next hour or minute
- variety of performance metrics
  - resource usage, response time, throughput
  - mean values vs. distributions
  - single-class vs. multi-class
- unknown, fluctuating, and evolving parameters

State of the Art: Product Features

Oracle 10g Self-Managing Database:

automatic database diagnostic monitor, automatic memory pool management, automatic workload repository, automatic routine administration, drill-down root-cause analysis, etc.

IBM DB2 Autonomic Technology:

index advisor, configuration advisor, health monitoring, learning query optimizer, etc.

Microsoft SQL Server Self-Tuning Features:

physical design wizard, continuous monitoring, statistics management, memory pressure analysis & heuristic resolution, etc.

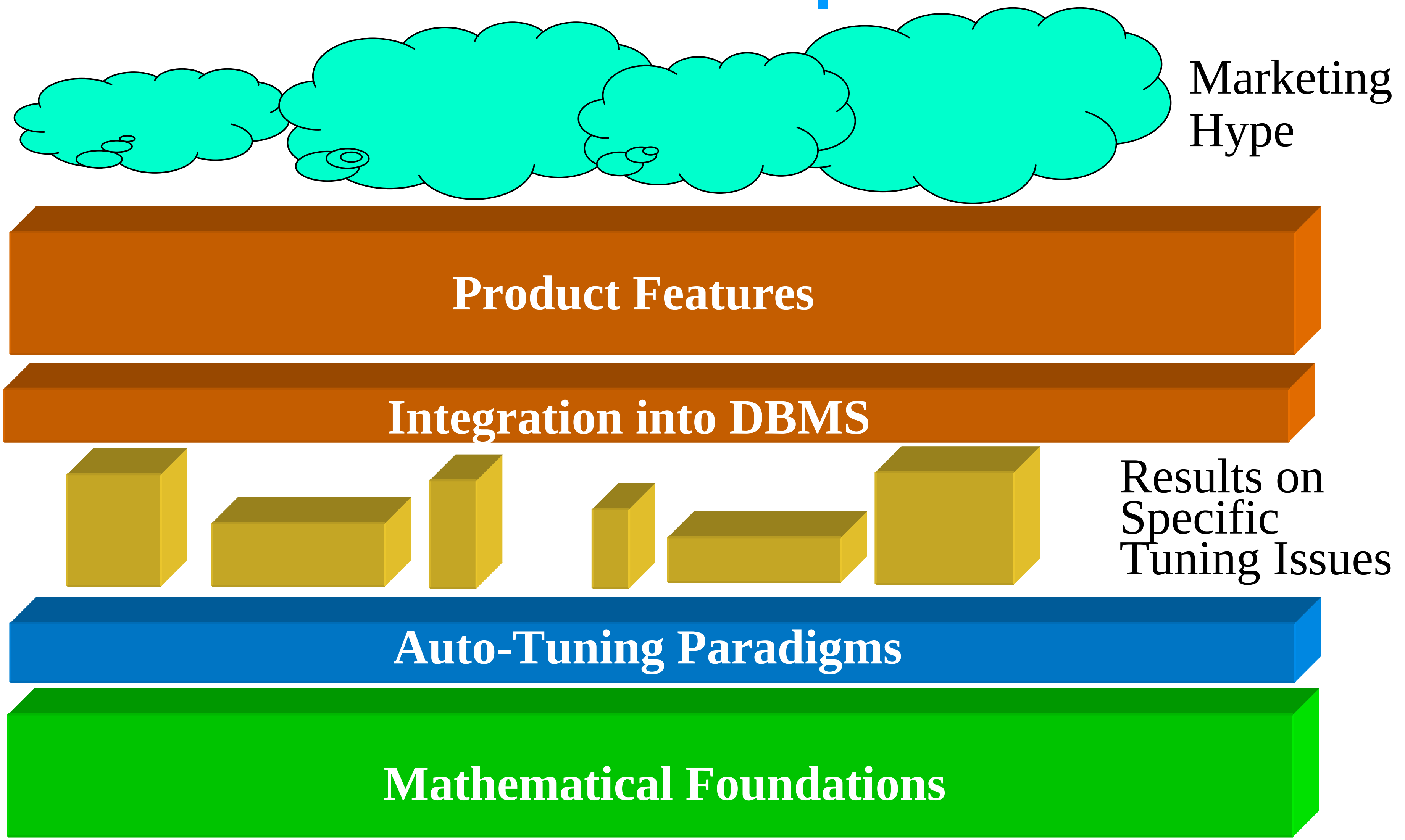
Storage systems: AutoRAID etc.

+ great online profiling & analysis infrastructure
+ viable solutions for specific tuning issues
– progress exaggerated by marketing
? fundamental principles

State of the Art: Scientific Principles

this page is left blank necessarily

Call for Scientific Principles



Foundations, Paradigms, Tuning Issues

physical design, QP statistics management,
memory management, MPL tuning, storage configuration,
application tricks, middleware caching, ...

tradeoff elimination, online optimization,
feedback loop, diagnostics, what-if analysis, ...

combinatorial optimization, queueing theory
control theory, statistical learning, ...

Auto-Tuning Paradigms

Aim: generalize from good approaches to specific tuning problems

Auto-tuning as:

- **tradeoff elimination** (ex. cache replacement)
- **static optimization** (ex. index selection)
- **stochastic prediction** (ex. capacity planning)
- **online optimization** (ex. memory governing)
- **feedback control loop** (ex. MPL tuning)
- what-if analysis (ex. bottleneck identification)
- statistical learning (ex. root-cause analysis)

General Literature

- D. Shasha, P. Bonnet: Database Tuning – Principles, Experiments, and Troubleshooting Techniques, Morgan Kaufmann, 2003 (see also tutorials at SIGMOD 2002 and VLDB 2002)
- S. Chaudhuri, B. Dageville, G. Lohman: Self-Managing Technology in Database, Management Systems, Tutorial Slides, VLDB 2004
- IBM Systems Journal 42(1), 2003, Special Issue on Autonomic Computing
- G. Weikum, A. Mönkeberg, C. Hasse, P. Zabback: Self-Tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering, VLDB 2002
- G. Weikum, C. Hasse, A. Mönkeberg, P. Zabback: The COMFORT Automatic Tuning Project, Information Systems 19(5), 1994
- S. Chaudhuri (Editor): IEEE CS Data Engineering Bulletin 22(2), 1999, Special Issue on Self-Tuning Databases and Application Tuning
- G. Candea, A.B. Brown, A. Fox, D. Patterson: Recovery-Oriented Computing: Building Multitier Dependability. IEEE Computer 37(11), 2004
- David S. Reiner, T.B. Pinkerton: A Method for Adaptive Performance Improvement of Operating Systems, SIGMETRICS 1981
- R. Jain: The Art of Computer Systems Performance Analysis, Wiley 1991
- A. Ailamaki (Editor), IEEE Data Engineering Bulleting Vol.29 No.3, Special Issue on Self-Managing Database Systems, September 2006

Call for Papers

International Workshop on Self-Managing Database Systems (SMDB 2007)

on April 16, 2007, in Istanbul, Turkey
in conjunction with ICDE 2007

Workshop chair: Guy Lohman
Submission deadline: November 20, 2006

for more details see
http://db.uwaterloo.ca/tcde-smdb/SMDB2007_CFP.html

Outline

- Part I: What Is It All About
- **Part II: Five Auto-Tuning Paradigms**
    1 Auto-Tuning as Tradeoff Elimination
    2 Auto-Tuning as Static Optimization with Deterministic Input
    3 Auto-Tuning as Static Optimization with Stochastic Input
    4 Auto-Tuning as Online Optimization
    5 Auto-Tuning as Feedback Control Loop
- Part III: Wrap-up

Part 2: Five Auto-Tuning Paradigms

1 Auto-Tuning as Tradeoff Elimination
2 Auto-Tuning as Static Optimization with Deterministic Input
3 Auto-Tuning as Static Optimization with Stochastic Input
4 Auto-Tuning as Online Optimization
5 Auto-Tuning as Feedback Control Loop

1 Auto-Tuning as Tradeoff Elimination

Tuning parameters handle tradeoffs

If you can find a parameter setting that yields universally close-to-optimal performance
(across a wide spectrum of workloads and for several technology generations)
then the tuning knob can be eliminated !

Examples:
- B^+-tree (vs. hash index): scan vs. random-lookup performance
- Page size: disk IO efficiency vs. memory efficiency
- Striping unit: IO parallelism vs. disk throughput
- LRU-k-style caching: recency (LRU) vs. frequency (LFU)

Example: Caching Strategies

LRU: drop page that has been least recently used
LFU: drop page that has been least frequently used

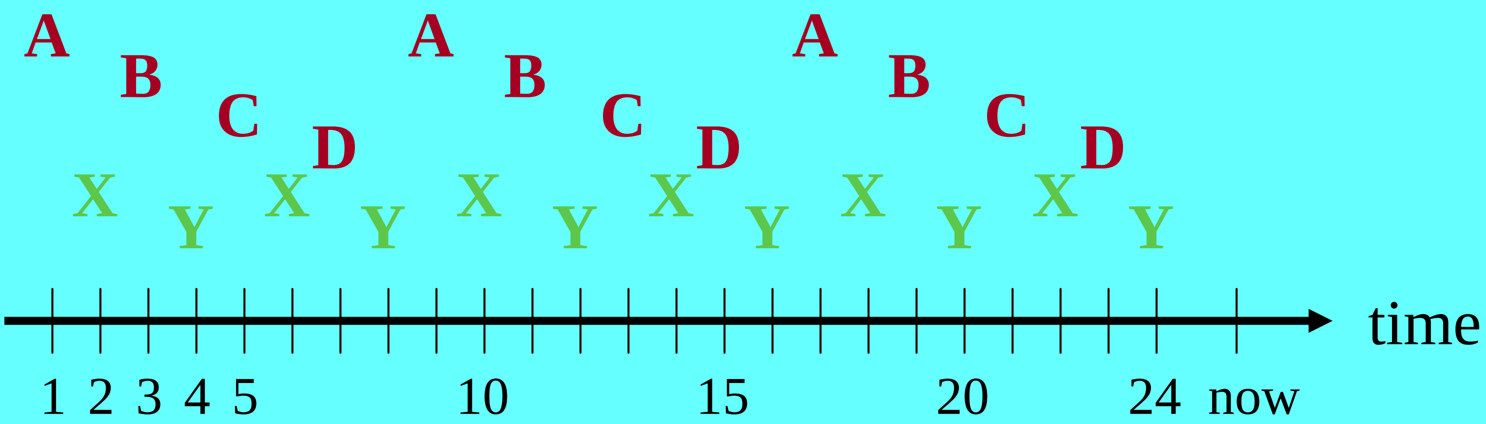
<u>Tradeoff recency vs. frequency:</u>

LFU: optimal for static access probabilities, but has no aging

LRU: optimal if last access is indicative for next future access

LRU degrades for sequential only-once access
and is suboptimal for multiple page pools (e.g., index pages)

Example:



Hybrid **LRU/LFU strategies** have weights that are critical to tune
Using **multiple page-pool caches** (each with LRU) is a tuning nightmare

Example: LRU-k Caching Strategy

LRU-k: drop page with the oldest k-th last reference

estimates $heat(p) = \dfrac{k}{now - t_k(p)}$ optimal for IRM

extensions and variations for variable-size objects, non-uniform storage, etc.

But cache bookkeeping has time and space overhead:

- O(log M) time for priority queue maintenance
- M* > M entries in cache directory to remember k last accesses to M* pages

+ overhead acceptable for improved cache hit rate
+ add'l bookkeeping memory is small and uncritical to tune

→ improved implementations: 2Q, ARC

Lesson: substitute critical tuning param by robust 2nd-order params and accept small overhead

Lessons and Problems

Lessons:

find „sweet spot“ for tuning param by mathematical analyis and/or substitute „difficult“ param by „well-tempered“ param, and accept some overhead for making better run-time decisions

Problems:

- caching for multi-class workload with per-class goals
- extend 2Q / ARC methods to hierarchical & distributed caching
- combine caching & prefetching with response time guarantees
- systematic study & characterization of tuning-parameter sensitivities

Literature on Tradeoff Elimination:

- E.J. O'Neil, P. O'Neil, G. Weikum: The LRU-k Page Replacement Algorithm for Database Disk Buffering, SIGMOD 1993
- T. Johnson, D. Shasha: 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, VLDB 1994
- J. Gray, G. Graefe: The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb, SIGMOD Record 26(4), 1997
- D. Lomet: B-Tree Page Size When Caching is Considered, SIGMOD Record 27(3), 1998
- N. Megiddo, D.S. Modha: Outperforming LRU with an Adaptive Replacement Cache Algorithm, IEEE Computer 37(4), 2004
- HP / Oracle White Paper: Auto-SAME, http://www.oracle.com/technology/tech/hp/storage.pdf
- P.A. Boncz, S. Manegold, M.L. Kersten: Database Architecture Optimized for the New Bottleneck: Memory Access, VLDB 1999
- J. Schindler, A. Ailamaki, G.R. Granger: Lachesis: Robust Database Storage Management Based on Device-specific Performance Characteristics, VLDB 2003
- A. Ailamaki: Database Architecture for New Hardware, Tutorial Slides, VLDB 2004

Outline

- Part I: What Is It All About
- **Part II: Five Auto-Tuning Paradigms**
    1 Auto-Tuning as Tradeoff Elimination
    2 **Auto-Tuning as Static Optimization with Deterministic Input**
    3 Auto-Tuning as Static Optimization with Stochastic Input
    4 Auto-Tuning as Online Optimization
    5 Auto-Tuning as Feedback Control Loop
- Part III: Wrap-up

Auto-Tuning as Static Optimization with Deterministic Input

Physical Database Design

Physical Database Design

- Performance of a query depends on execution plan
- Execution plan picked by optimizer depends on
  - Statistics created by the optimizer
  - Physical design: Objects that exist
- Choice of statistics and physical design objects amortized
- Physical Design Configuration
  - Clustered Indexes + Non-clustered indexes + Materialized Views

Roadmap

- ***Why the problem is hard?***
- Abstract problem Formulation
- Measuring Goodness of a design
- Search: Need for Merging
- Search: Bottom-up vs Top-down
- Search: Leveraging the server

Is this a hard problem?

```
SELECT A,B,C
FROM R
WHERE 10 < A < 20
  AND 20 < B < 100

SELECT B,C,D
FROM R
WHERE 50 < B < 100
  AND 60 < 2*D < 80
```

Storage for (A,B,C) + (D,B,C) is too large!

```
UPDATE R
SET B=B+1
WHERE 10 < C < 20
```

We started fine, but progressively:

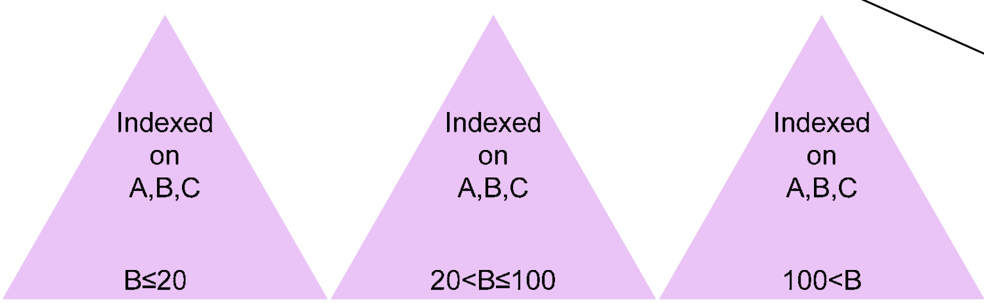
- Used statistical information
- Guessed how the optimizer would use statistics
- Guessed how the optimizer would use proposed indexes
- Gave up

And that was just indexes!

```
SELECT A,B,C
FROM V
WHERE 20 < B < 100
       ||
SELECT A,B,C
FROM R
WHERE 10 < A < 20
  AND 20 < B < 100
```

Views and Indexes on Views

CREATE VIEW V AS
SELECT A,B,C FROM R
WHERE 10 < A < 20
+
INDEX on I_V(B,A,C)



Access only this partition

Partitions on Indexes (on views)

CREATE INDEX ON R(A,B,C)
PARTITIONED ON B [20, 100]

Real Life Queries are Complex!

```
SELECT CNTRYCODE, count(*) as NUMCUST, sum(C_ACCTBAL) as TOTACCTBAL
FROM (
          SELECT substring(C_PHONE,1,2) as CNTRYCODE, C_ACCTBAL
          FROM CUSTOMER
          WHERE substring(C_PHONE,1,2) in ('31', '17', '30', '24', '26', '34', '10', '')
                AND C_ACCTBAL > (
                              SELECT avg(C_ACCTBAL)
                              FROM CUSTOMER
                              WHERE C_ACCTBAL > 0.00
                                    AND substring(C_PHONE,1,2) in
                                         ('31', '17', '30', '24', '26', '34', '10', '')
                                )
               AND NOT EXISTS (
                              SELECT *
                              FROM ORDERS
                              WHERE O_CUSTKEY = C_CUSTKEY
                              )
          ) as CUSTSALE
GROUP BY  CNTRYCODE
ORDER BY  CNTRYCODE
```

TPC-H SAMPLE QUERY

Real Life Queries are Complex!

```
---
--- Galaxy target selection with spectroscopic redshifts
---
SELECT top 15  str(gal.ra,9,4) AS ra, str(gal.dec,8,4) AS dec,
              cast(spec.objTypeName AS CHAR(9)) AS type,
              str(spec.z,7,4) AS Z,
              fSpecZStatusN(spec.zStatus) AS status,
              fGetUrlSpecImg(spec.specObjID) AS Spectra
FROM
         @database..PhotoPrimary AS gal,
         @database..specObj AS spec
WHERE
         gal.objID = spec.bestObjID AND
         -- Our star-galaxy separation AND target selection
         psfMag_r - modelMag_r >= @delta_psf_model AND
         petroMag_r - extinction_r <= @maglim AND
         petroMag_r - 2.5*log10(2*@pi*petroR50_r*petroR50_r) < @SBlim AND
         -- Check flags
         (flags & @bad_flags) = 0 AND
         (((flags & @BLENDED) = 0) OR ((flags & @NODEBLEND) != 0)) AND
         -- Check spectro flags
                         NOT spec.zStatus IN (@FAILED, @NOT_MEASURED)
```

SKYSERVER SAMPLE QUERY

Roadmap

- Why the problem is hard?
- ***Abstract problem Formulation***
- Measuring Goodness of a design
- Search: Need for Merging
- Search: Bottom-up vs Top-down
- Search: Leveraging the server

Physical Database Design as Static Optimization

- Workload
  - queries and updates
- Configuration
  - A set of indexes, materialized views and partitions from a search space
- Constraints
  - Upper bound on storage space for indexes
- Search: Pick a configuration with lowest *cost* for the given database and workload.

Roadmap

- Why the problem is hard?
- Abstract problem Formulation
- ***Measuring Goodness of a design***
  - ***What-if Physical Design***
- Search: Need for Merging
- Search: Bottom-up vs Top-down
- Search: Leveraging the server

What is “cost”?

- Execution cost of the query
  - Requires physical design changes – too disruptive
- Optimizer Estimated Cost
  - Used to compare alternative plans for the *query*
- We choose optimizer estimated cost
  - Better than designing a new cost model
  - Estimate quantitatively the impact of physical design on workload (queries and updates)
    - e.g., if we add an index on T.c, which queries benefit and by how much?
  - Never meant to compare across physical designs/Queries

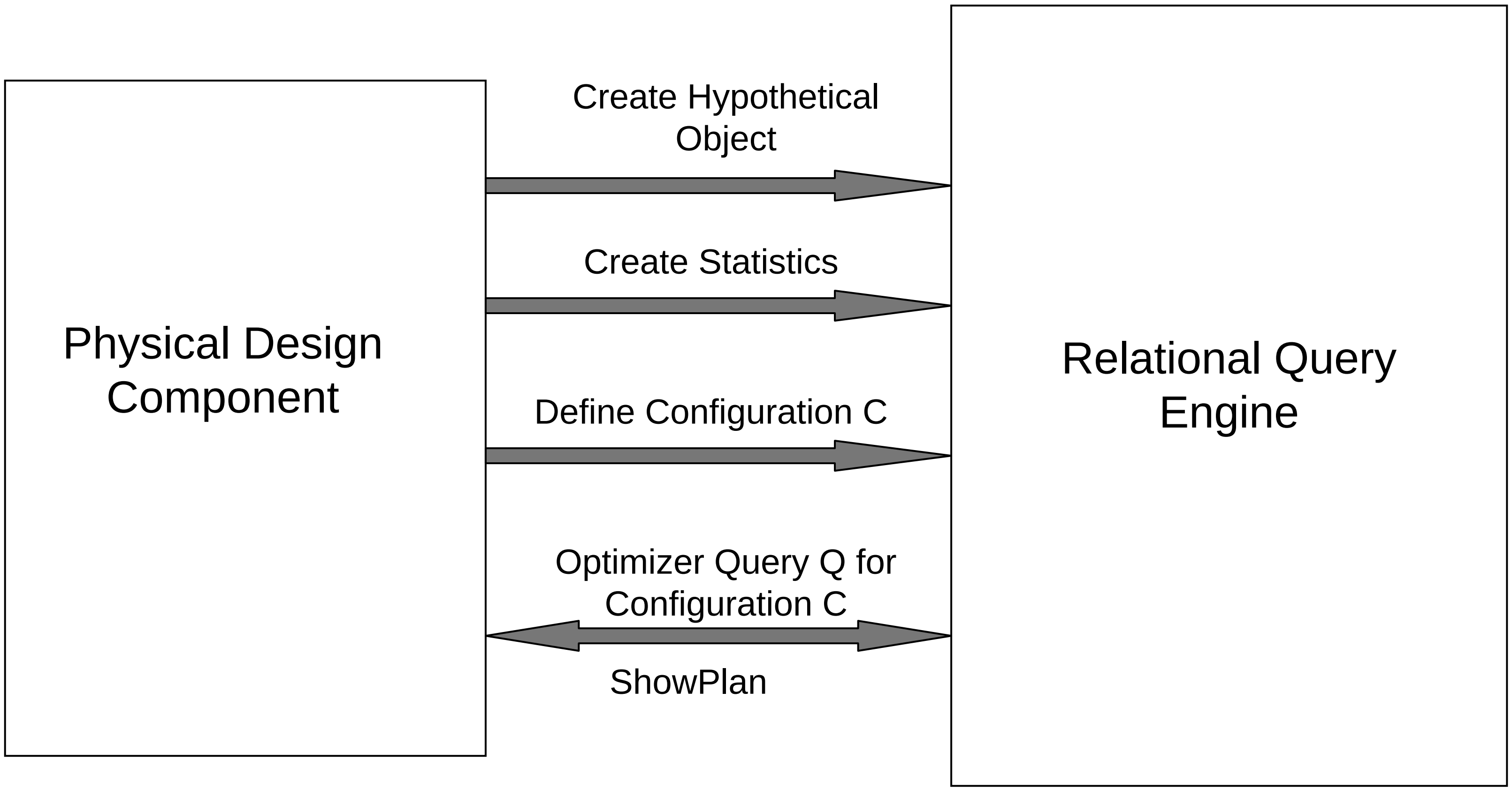
Estimating Cost of a configuration for Search

- Without making actual changes to physical design
- What-If Indexes!

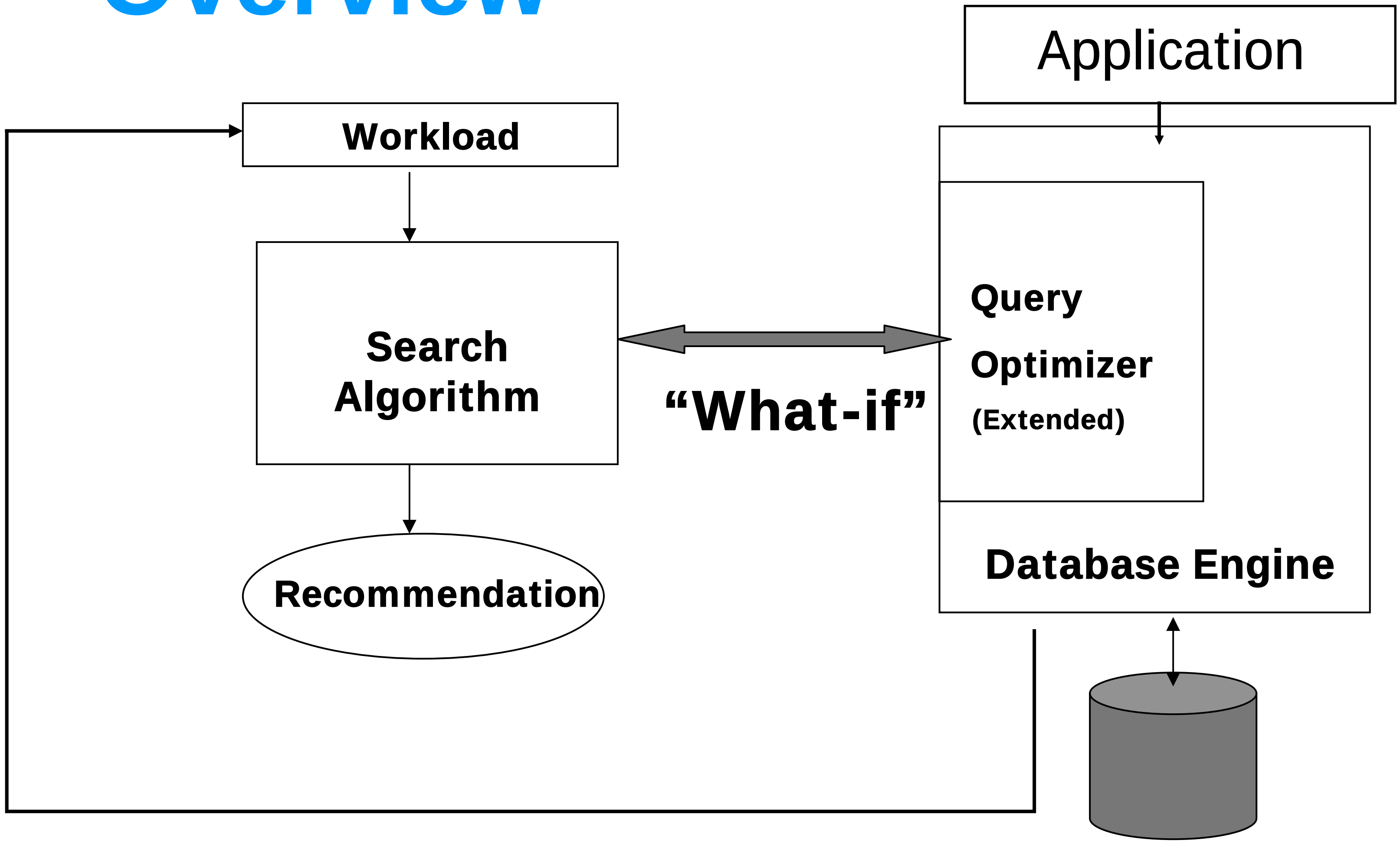
"What-If" Indexes

- Query Optimizer decides which plan to choose given a physical design
- Query optimizer does not require physical design to be materialized
  - Relies on statistics to choose right plan
    - Sampling based techniques for building statistic
- Sufficient to fake existence of physical design
  - Build approximate statistics
  - Change "meta-data" entry

Using What-If Analysis



"What-If" Architecture Overview



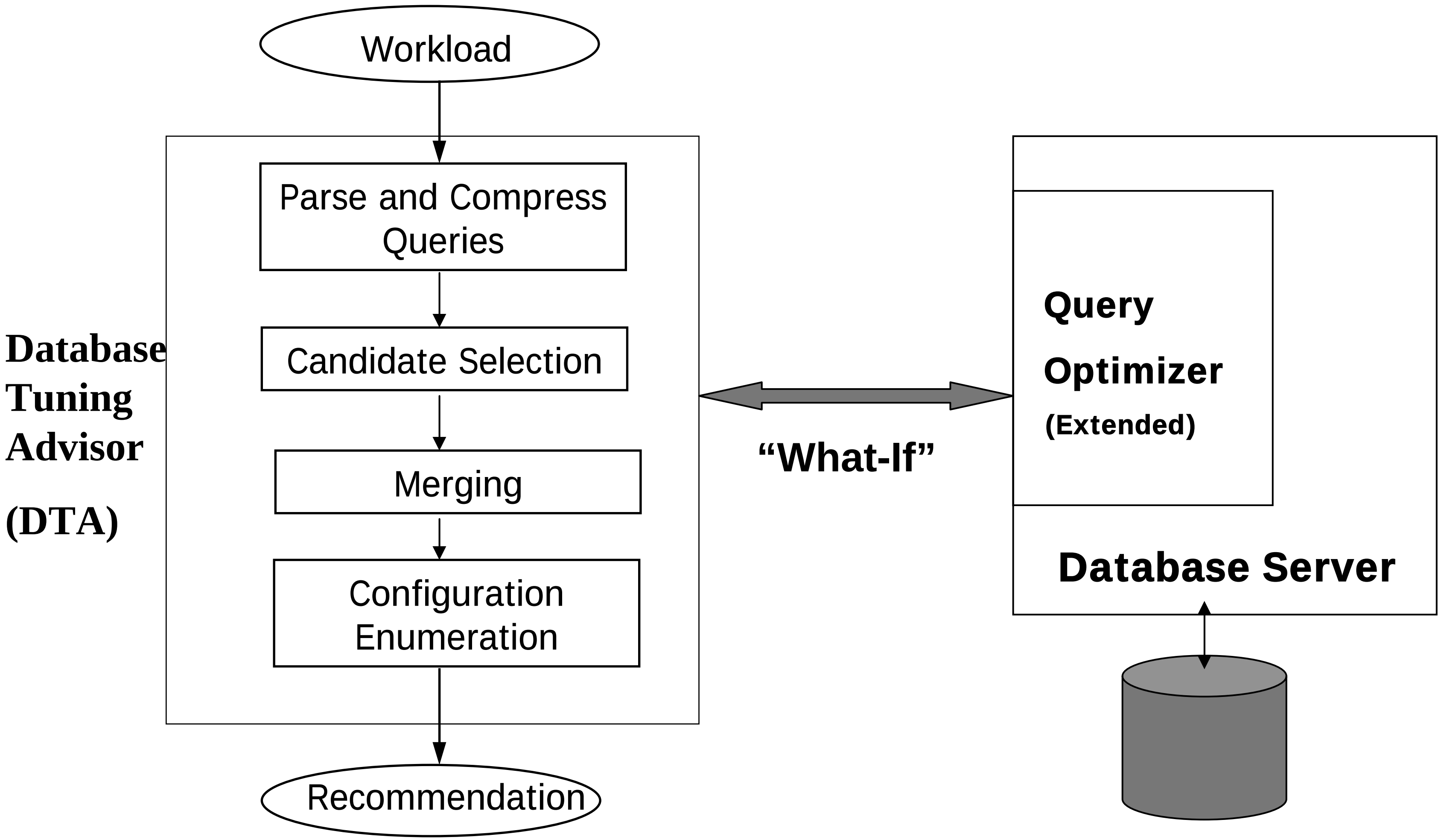
Roadmap

- Why the problem is hard?
- Abstract problem Formulation
- Measuring Goodness of a design
- ***Search: Need for Merging***
- Search: Bottom-up vs Top-down
- Search: Leveraging the server

Balancing Requirements of Multiple Queries

- Simple divide and conquer not enough
- Because, union of “best” configurations for each query may not be feasible
  - Violate storage constraints
  - Maintenance costs for update queries may rule out “ideal” indexes/MV
- Use locally suboptimal alternatives - need for “merging”

Example: Database Tuning Advisor



Characteristics of Merged Candidates

- A derived configuration from one or more seed configurations
- M_{12} is a “merged” candidate from parents P_1, P_2
  - If Q was using P_1, it can have a plan using M_{12}
  - New plans using M_{12} is not “much” more expensive
- Merging can
  - Introduce new logical objects (materialized views)
  - Introduce new physical structures (indexes)

Sample Algorithm: MV Merging Candidates

- V_1 and V_2 be on same set of tables and same join conditions
- Merged MV V_{12} contains
  - Union of projection columns of V_1 , V_2
  - Union of Group-By columns of V_1 and V_2
  - Selection conditions *common* to V_1 and V_2
  - Columns in *different* selection conditions pushed into Group-By
- Reject the merge if size of V_{12} is too large

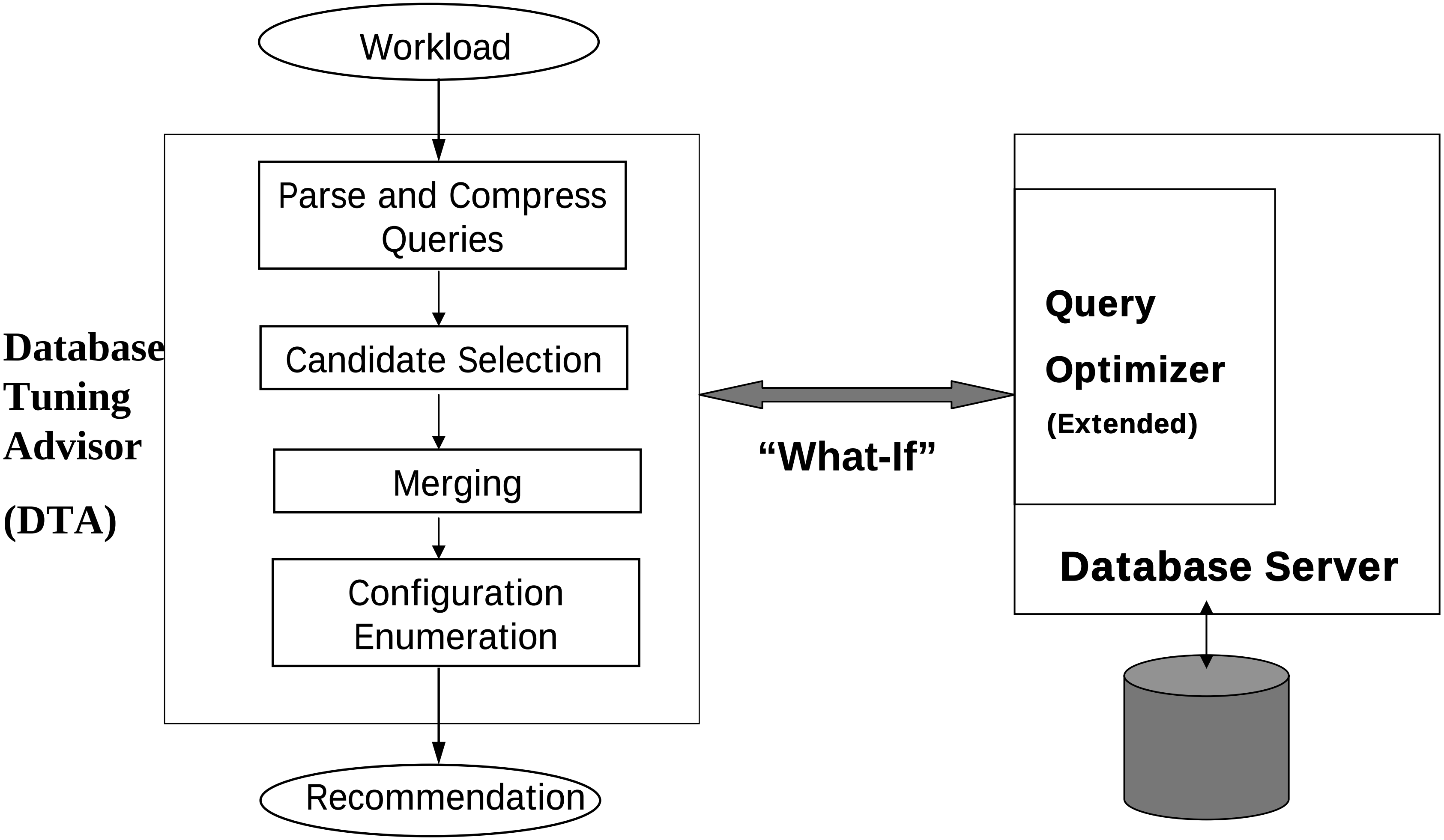
Sample Algorithm: Index Merging Candidates

- Union of columns in I_1 and I_2
  - Index scan benefits preserved
  - Preserve seek benefits to at least one
- A common prefix of two indexes
  - Partial seek benefits
- Multiple thinner indexes
  - Replace covering indexes with Intersection/Union plans (A,B|C,F) [S] (B,E|F) = (B|F) + (A|C) + (E)

Roadmap

- Why the problem is hard?
- Abstract problem Formulation
- Measuring Goodness of a design
- Search: Need for Merging
- ***Search: Bottom-up vs Top-down***
- Search: Leveraging the server

Example: Database Tuning Advisor



Search Algorithm

Search Space = “Locally Best” U “Merged”

- Indexes and Indexed Views need to be considered together
  - Cannot “break” into two sequential selection steps
- Search driven by reduction in optimizer estimated costs
  - *Top-Down*: Get an optimal structure and then modify it
  - *Bottom-up*: Grow by picking the next k-structures

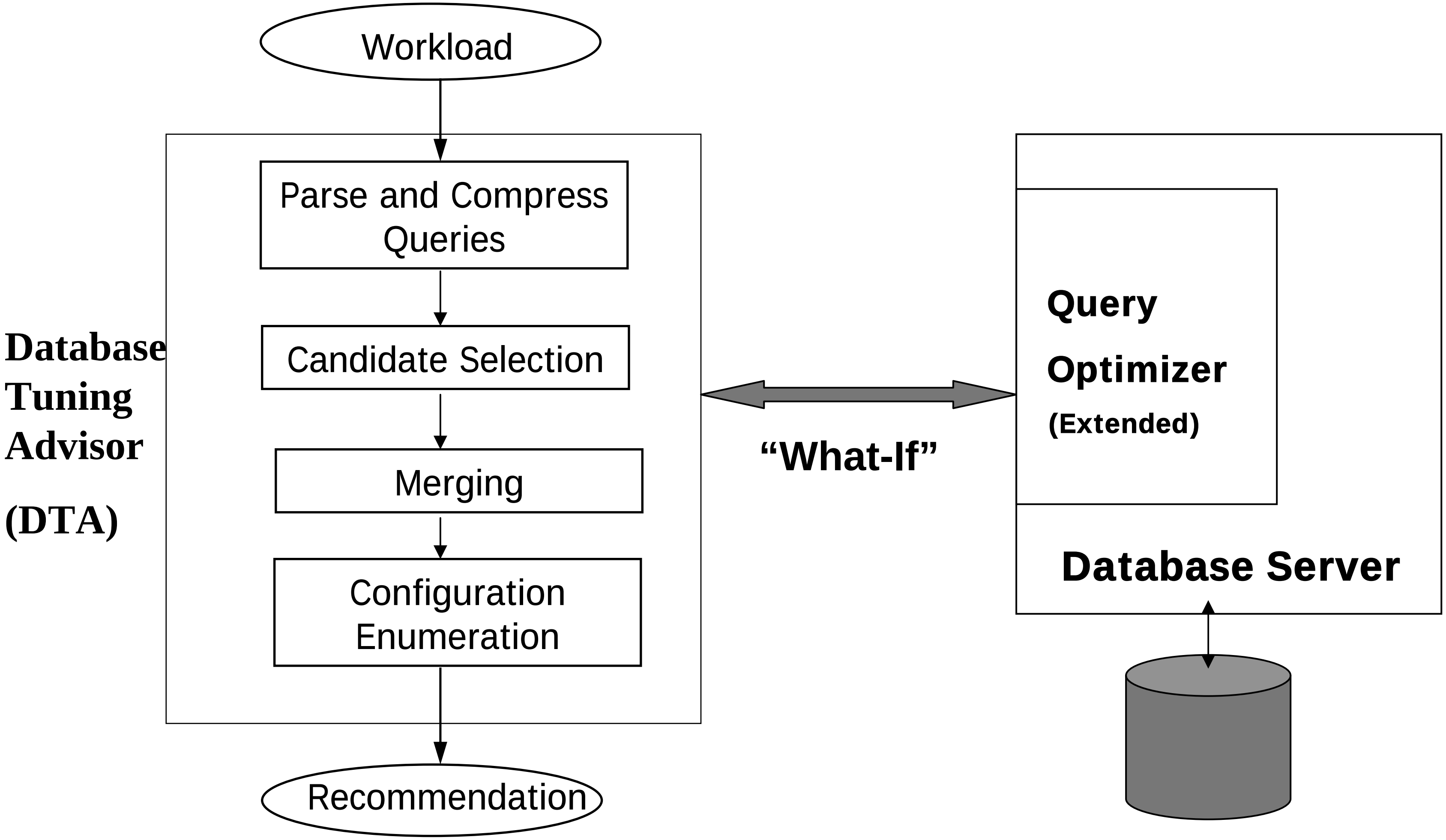
Quality: Incremental Cost/Benefit of a structure

- Benefit of an index/MV is relative to a given configuration
- Example
  - Two clustering indexes together can reduce cost of a join significantly
- Example Metric
  - Incremental penalty for removing a structure: (increase in cost)/(reduction of space)

Efficiency: Reducing Optimizer Invocations

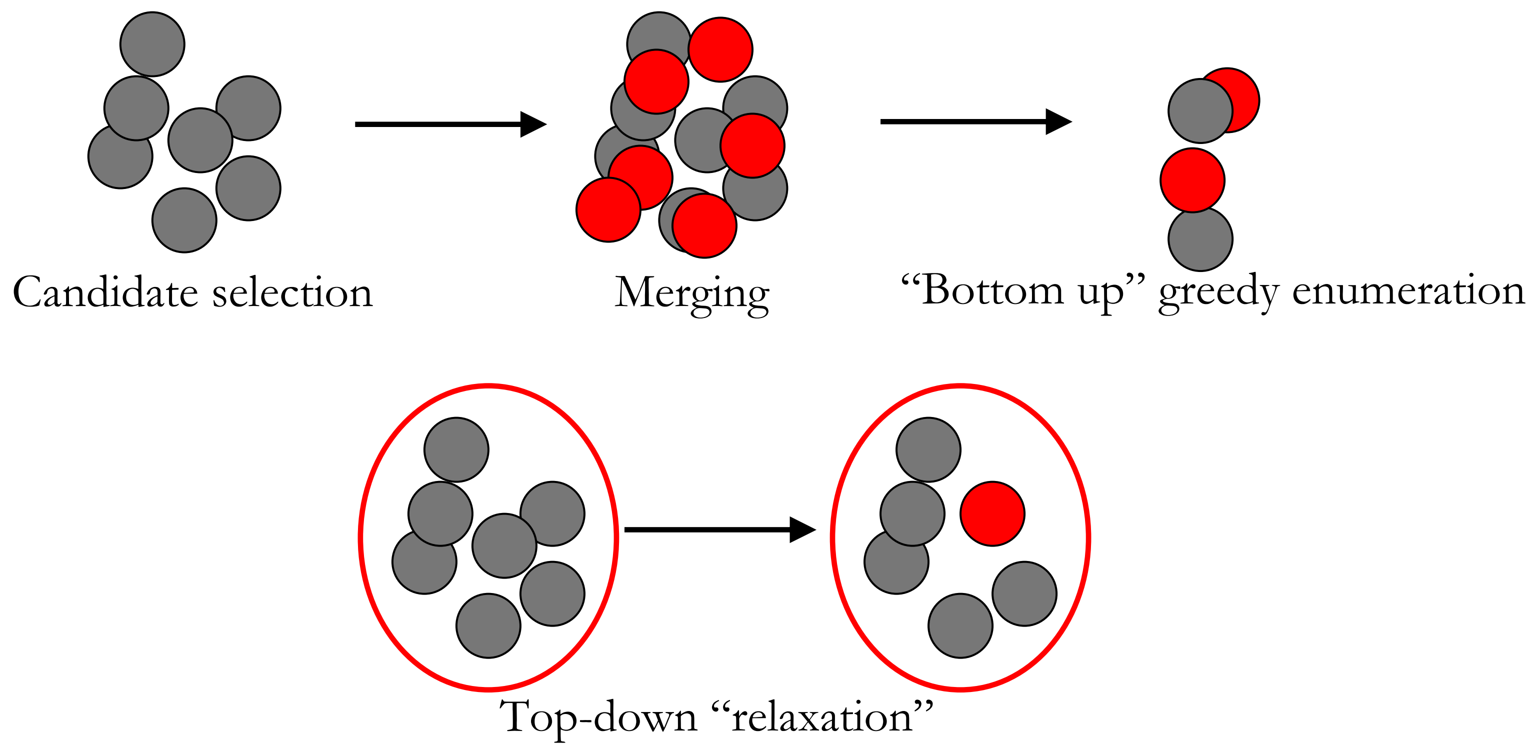
- Each physical design can potentially resul0 88 -31otenti

Example: Database Tuning Advisor



Top-down Search

- **Shrink supersets rather than expanding subsets**
- **Mixes *merging* and *enumeration* phases**



Other Approaches

- [Agrawal et. al 2000] Bottom-up search
  - Incrementally add “most promising” structures
  - But, consider tight interactions
  - Initially exhaustive, degenerate into greedy
- [Valentin et.al. 2000] Knapsack + Genetic
  - Create a feasible solution through knapsack (ignore interactions)
  - Genetic mutations and generate new candidates

Roadmap

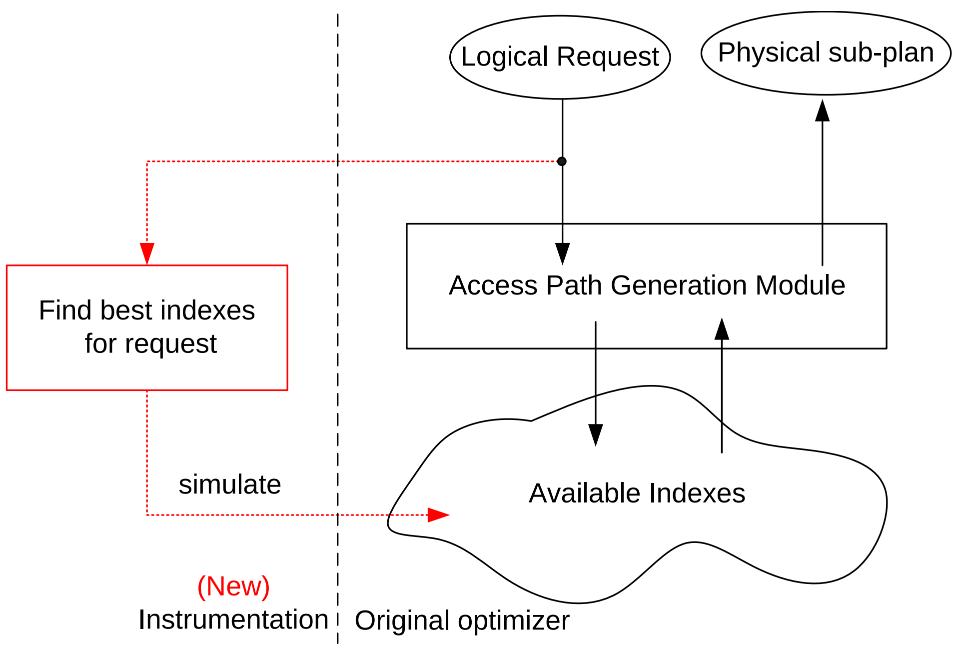
- Why the problem is hard?
- Abstract problem Formulation
- Measuring Goodness of a design
- Search: Need for Merging
- Search: Bottom-up vs Top-down
- ***Search: Leveraging the server***

Architecture: Knowledge of the Optimizer

- Reduce co-dependence on optimizer by
  - Making only broadest assumptions (e.g., importance of covering indexes)
- Use knowledge of key optimizer characteristic selectively (deeper interaction)

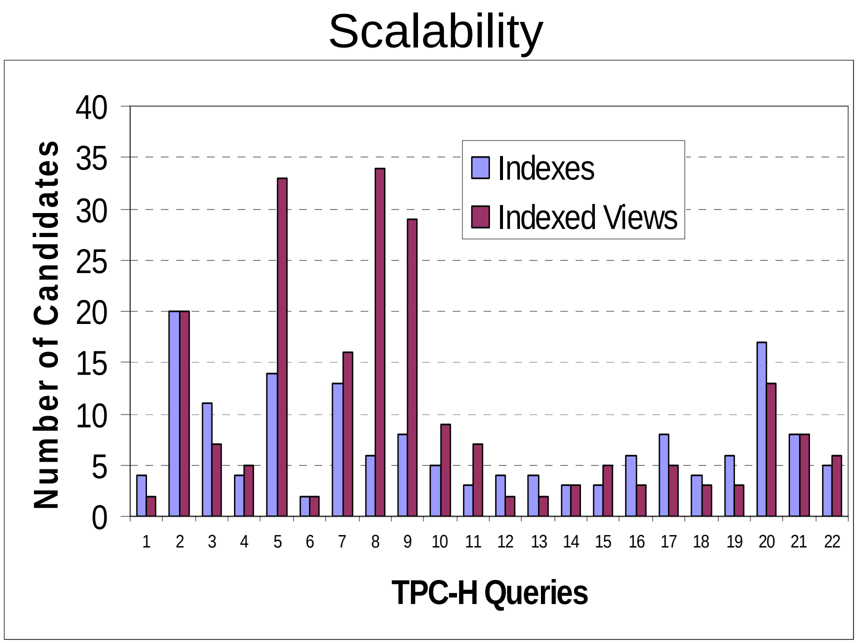
Instrumenting the Query Optimizer

- Intercept index and view "*requests*"
  - Concise, no false nen005m3s/posi05m3s
- Obtain optimal indexes and views from requests

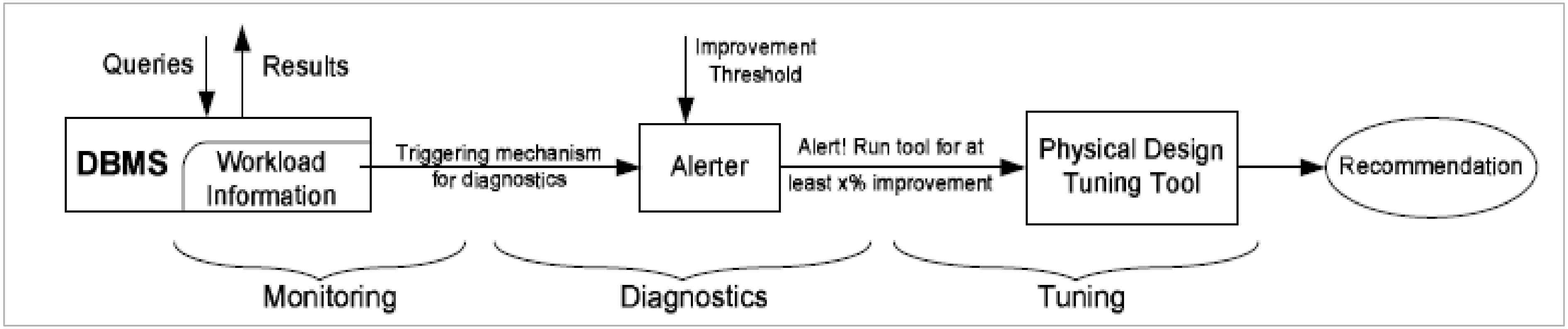


Instrumenting the Query Optimizer

- Intercept “index and view requests”
  - Concise, no false negatives/positives
- Obtain optimal indexes and views from requests
- Inject such structures during optimization



When to Tune?



- Low-overhead diagnostics
- Reliable lower-bound improvement
  - No false positives
  - “Proof” with valid configuration
  - Upper-bound Estimate
  - [Bruno, Chaudhuri 06] (this conference)
- COLT [Schnaitter+ 06] does periodic “epoch-at-a-time” polling distinguishing structure classes

Lessons and Problems

- **Lessons:**
  - Precise static optimization problem
    - Challenges in cost definition
    - Complex search space – depends on server sophistication
- **Problems:**
  - How deeply to exploit optimizer
  - Uncertainty in cost estimation
  - Workload model [Agrawal+06]
  - Search Algorithms (combinatorial optimization)

References (1)

- Surajit Chaudhuri, Benoît Dageville, and Guy M. Lohman. Self-Managing Technology in Database Management Systems. Tutorial presented at VLDB 2004.
- Sheldon J. Finkelstein, Mario Schkolnick, Paolo Tiberio. Physical Database Design for Relational Databases. ACM TODS 13(1): 91-128 (1988).
- Steve Rozen, Dennis Shasha: A Framework for Automating Physical Database Design. VLDB 1991: 401-411
- Surajit Chaudhuri and Vivek R. Narasayya. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. VLDB 1997.
- Surajit Chaudhuri, and Vivek R. Narasayya. AutoAdmin 'What-if' Index Analysis Utility. SIGMOD 1998.
- Surajit Chaudhuri and Vivek R. Narasayya. Index Merging. ICDE 1999.

References (2)

- Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. ICDE 2000.
- Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated Selection of Materialized Views and Indexes in SQL Databases. VLDB 2000.
- Jun Rao, Chun Zhang, Nimrod Megiddo, and Guy M. Lohman. Automating Physical Database Design in a Parallel Database. SIGMOD 2002.
- Sanjay Agrawal, Vivek R. Narasayya, and Beverly Yang. Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. SIGMOD 2004.
- Nicolas Bruno and Surajit Chaudhuri. Automatic Physical Database Tuning: A Relaxation-based Approach. SIGMOD 2005.
- Nicolas Bruno and Surajit Chaudhuri. Physical Design Refinement: The ``Merge-Reduce'' Approach, EDBT 2006.

References (3)

- Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. ICDE 2000.
- Arnd Christian König, Shubha U. Nabar: Scalable Exploration of Physical Database Design. ICDE 2006
- Nicolas Bruno, Surajit Chaudhuri: Physical Design Refinement: The "Merge-Reduce" Approach. EDBT 2006
- Karl Schnaitter, Serge Abiteboul, Tova Milo, Neoklis Polyzotis: COLT: Continuous On-Line Database Tuning, SIGMOD Demo 2006
- Nicolas Bruno, Surajit Chaudhuri: To Tune or not to Tune? A Lightweight Physical Design Alerter, VLDB 2006

Part 2: Five Auto-Tuning Paradigms

1 Auto-Tuning as Tradeoff Elimination

2 Auto-Tuning as Static Optimization with Deterministic Input

3 Auto-Tuning as Static Optimization with Stochastic Input

- **Capacity Planning**
- Example: Cache Sizing
- Queueing Theory
- Further Aspects and Lessons

4 Auto-Tuning as Online Optimization

5 Auto-Tuning as Feedback Control Loop

Auto-Tuning as Static Optimization with Stochastic Input

Capacity Planning and System Configuration

Workload varies statistically
Load may be unbounded
$\Rightarrow$ input is stochastic
$\Rightarrow$ can provide only stochastic guarantees

System Capacity Planning

Key issue for long-term tuning:
how big should you configure your system resources?

- CPU speed, #processors in SMP, #servers in server farm
- amount of memory, cache sizes
- #disks, disk types, storage controller types
- software parameters for (static) resource limitation

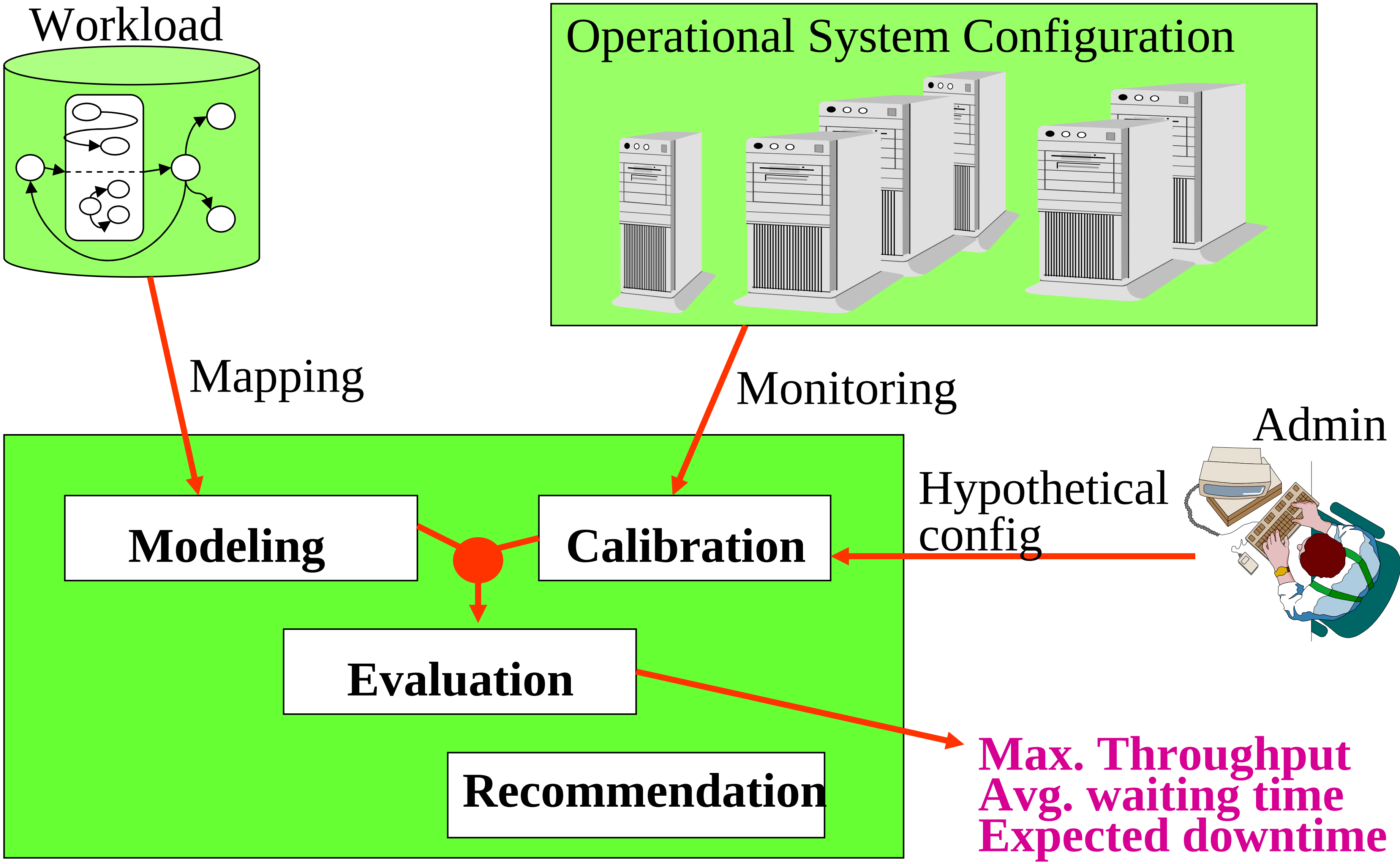
→ configure system so as to meet goals for

- performance: throughput, response time (mean or quantile)
- reliability and availability

reasonably understood for OLTP server, HTTP server, etc.
not so well understood for DBMS, multi-tier Web Services

→ workload and complex system behavior
approximated/abstracted by stochastic models

System Configuration Tool (1)



Part 2: Five Auto-Tuning Paradigms

1 Auto-Tuning as Tradeoff Elimination
2 Auto-Tuning as Static Optimization with Deterministic Input
3 Auto-Tuning as Static Optimization with Stochastic Input
- Capacity Planning
- **Example: Cache Sizing**
- Queueing Theory
- Further Aspects and Lessons

4 Auto-Tuning as Online Optimization
5 Auto-Tuning as Feedback Control Loop

Example: DBMS Cache Sizing

Cost / throughput consideration:

Keep page in cache if $C_{cache} < C_{disk}$

$$\Leftrightarrow 100\,KB \,\frac{1000\,\$}{1GB} < \frac{1000\,\$}{100\,s^{-1}}\,\lambda \qquad \Leftrightarrow \lambda > 0.01\,s^{-1}$$

Response-time guarantee:

Minimum cache size M such that

$$RT_{percentile} = f(hit\ ratio, \ldots) = f(g(M), \ldots) \leq RT_{goal}$$

LRU-k Cache Hit Rate Prediction

$$P(W) := E[\,distinct\ \ pages\ referenced$$

$$= \sum_{i=1}^{n} \sum_{j=k}^{W} \binom{W}{j} \beta_i^{j} (1-\beta_i)^{W-j}$$

$$W: \quad P^{-1}(M)$$

LRU-k Response Time Prediction

with cache size M, page access probabilites $\beta_1, \beta_2, \ldots$,
disk characteristics, global load, ...

- *RT = f (hit rate, disk access time)*
- *disk access time = service time + queueing delay*

$\rightarrow$ *need disk model*
$\rightarrow$ *need queueing analysis*

rich repertoire of math, many models around,
but care needed in adopting models
$\rightarrow$ need understanding of modeling & math

Part 2: Five Auto-Tuning Paradigms

1 Auto-Tuning as Tradeoff Elimination
2 Auto-Tuning as Static Optimization with Deterministic Input
3 Auto-Tuning as Static Optimization with Stochastic Input
- Capacity Planning
- Example: Cache Sizing
- **Queueing Theory**
- Further Aspects and Lessons

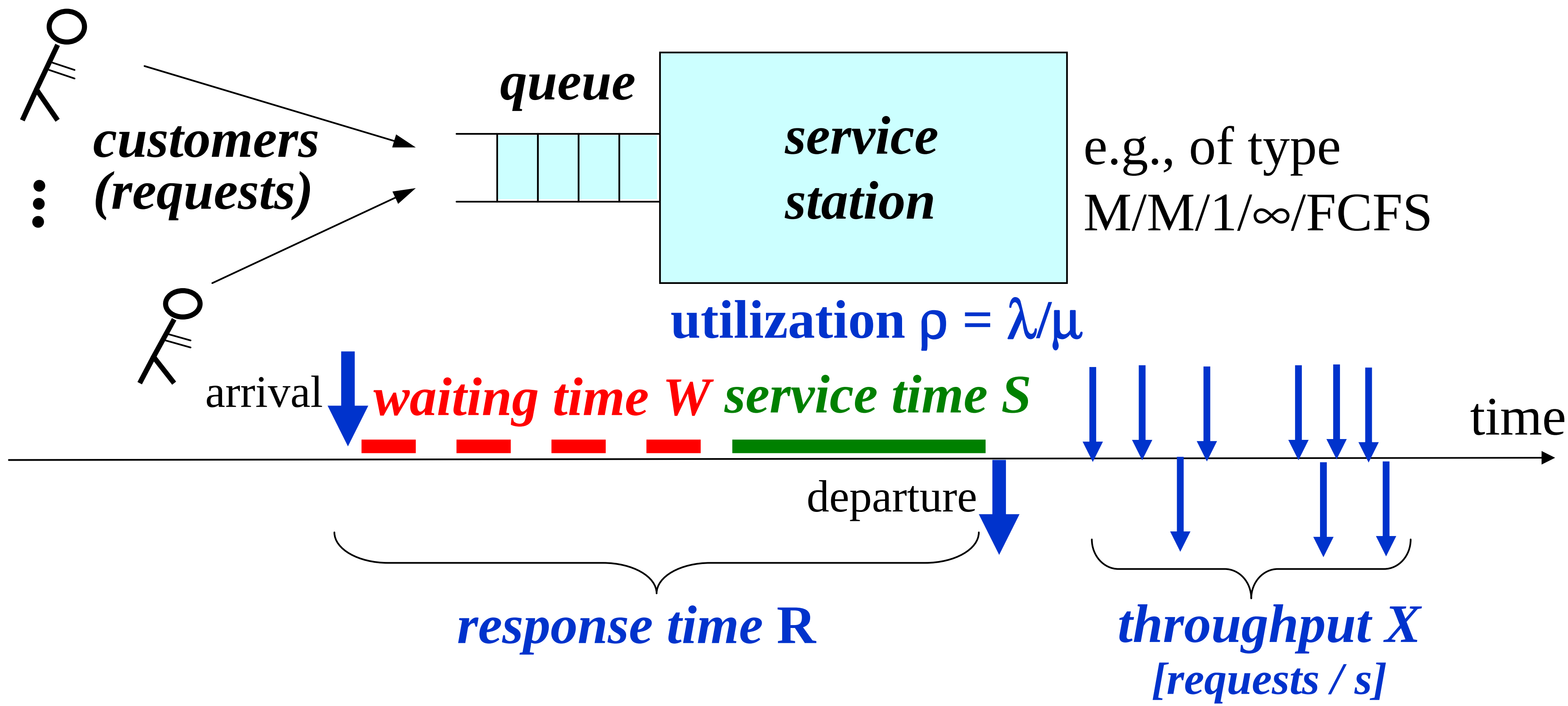
4 Auto-Tuning as Online Optimization
5 Auto-Tuning as Feedback Control Loop

Basics of Queueing Systems

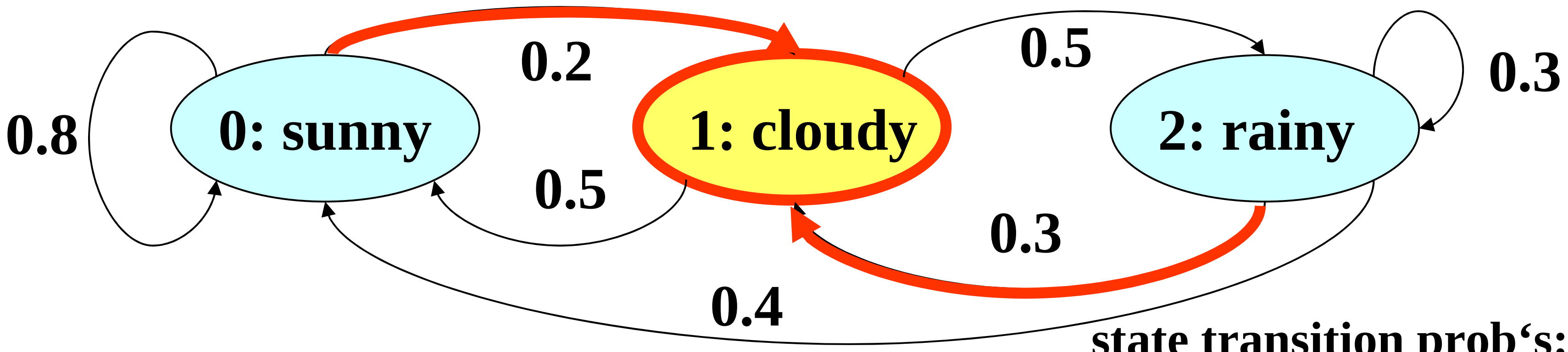
prob. distr. of interarrival time (e.g.: M = exp. distr.)
arrival rate λ

scheduling policy (e.g.: FCFS)

prob. distr. of service time S (e.g.: M = exp. distr.)
service rate μ



Markov Chains



state transition prob‘s: p_{ij}

$p0 = 0.8\ p0 + 0.5\ p1 + 0.4\ p2$

$\boxed{p1 = 0.2\ p0 + 0.3\ p2}$

$p2 = 0.5\ p1 + 0.3\ p2$

$p0 + p1 + p2 = 1$

$\Rightarrow\ p0 \approx 0.657,\ p1 = 0.2,\ p2 \approx 0.143$

state prob‘s in step t: $p_i^{(t)} = P[S(t)=i]$

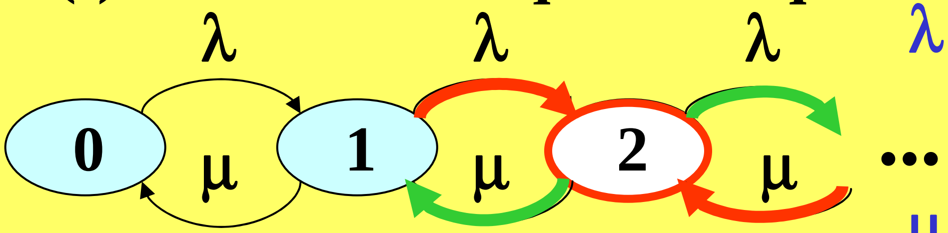
Markov property: $P[S(t)=i \mid S(0), ..., S(t-1)] = P[S(t)=i \mid S(t-1)]$

interested in stationary state probabilities:

$$p_j := \lim_{t\to\infty} p_j^{(t)} = \lim_{t\to\infty} \sum_k p_k^{(t-1)} p_{kj} \qquad p_j = \sum_k p_k\, p_{kj} \qquad \sum_j p_j = 1$$

M/M/1 Queueing Systems

N(t): number of requests in queue (or in service)



λ: arrival rate

μ: service rate

flow rate:

$$\lim_{\Delta t \to 0} \frac{P[\text{transition in } \Delta t]}{\Delta t}$$

flow balance equations:

$$p_1 \mu = p_0 \lambda \quad \text{and} \quad \boxed{\underline{p_{n-1}\lambda + p_{n+1}\mu} = \underline{p_n(\lambda+\mu)}} \quad \text{for } n \ge 1$$

$$\Rightarrow \text{for } \rho := \frac{\lambda}{\mu} < 1: \quad p_n = \rho^n (1-\rho) \quad \text{for } n \ge 0$$

$$\Rightarrow \quad E[N] = \sum_{n=0}^{\infty} n\, p_n = \frac{\rho}{1-\rho} \quad \Rightarrow \quad \boxed{E[R] = \frac{E[N]}{\lambda} = \frac{E[S]}{1-\rho}}$$

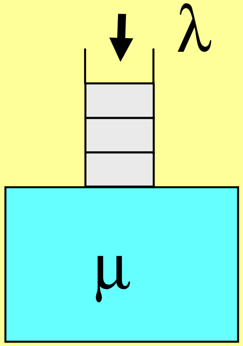
response time distribution: $F_R(t) = P[R \le t] = 1 - e^{-t/E[R]}$

but more complex for non-exponential service time

Insights (Example): Variability Matters

M/G/1:

with 2 workload classes



$S_1 = 0.01$ s $\quad S_2 = 0.1$ s

$\lambda_1 = 40\ s^{-1} \quad \lambda_2 = 4\ s^{-1}$

$E[S] \approx 0.01818$ s

$E[S^2] \approx 0.00091\ s^2$

$\rho = 0.8$

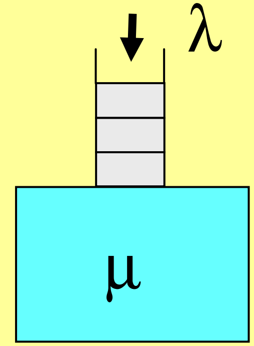
$$\Rightarrow E[R] = E[S] + \frac{E[S^2]\rho}{2(1-\rho)E[S]}$$

$$\approx 0.01818 + \frac{0.00091 \cdot 0.8}{0.4 \cdot 0.01818}\ s$$

$$\approx 0.118\ s$$

M/D/1:

with 1 „average“ class



$S \approx 0.01818$ s

$\lambda = 44\ s^{-1}$

$E[S] \approx 0.01818$ s

$E[S^2] = 0.00033\ s^2$

$\rho = 0.8$

$$\Rightarrow E[R] = E[S] + \frac{E[S^2]\rho}{2(1-\rho)E[S]}$$

$$\approx 0.01818 + \frac{0.00033 \cdot 0.8}{0.4 \cdot 0.01818}\ s$$

$$\approx 0.054\ s$$

Other Queueing Systems

many variations and generalizations:

- M/G/1 models with general service time distributions
- multiple request (customer) classes, with priorities
- service scheduling other than FIFO
- GI/G/1 models
- discrete-time models
- queueing networks

etc. etc.

Mathematical Tools (1)

X, Y, ...: continuous random variables with non-negative real values

A, B, ...: discrete random variables with non-negative integer values

$F_X(x) = P[X \le x]$: prob. distribution of X

$f_X(x) = F'_X(x)$: prob. density of X

$f_A(k) = P[A = k]$: prob. density of A

$$f^*_X(s) = \int_0^\infty e^{-sx} f_X(x)\,dx = E[e^{-sX}]:$$

Laplace-Stieltjes transform (LST) of X

$$G_A(z) = \sum_{i=0}^\infty z^i f_A(i) = E[z^A]:$$

generating function of A

Examples:

exponential:

$$f_X(x) = \alpha e^{-\alpha x}$$

$$f^*_X(s) = \frac{\alpha}{\alpha + s}$$

Erlang-k:

$$f_X(x) = \frac{\alpha k(\alpha k x)^{k-1}}{(k-1)!} e^{-\alpha k x}$$

$$f^*_X(s) = \left(\frac{k\alpha}{k\alpha + s}\right)^k$$

Poisson:

$$f_A(k) = e^{-\alpha} \frac{\alpha^k}{k!}$$

$$G_A(z) = e^{\alpha(z-1)}$$

Mathematical Tools (2)

Convolution of independent random variables:

$$F_{X+Y}(z) = \int_0^z f_X(x) F_Y(z-x)\,dx \qquad F_{A+B}(k) = \sum_{i=o}^{k} f_A(i) F_Y(k-i)$$

$$f^*_{X+Y}(s) = f^*_X(s)\, f^*_Y(s) \qquad G_{A+B}(z) = G_A(z) G_B(z)$$

Chernoff tail bound: $P[X \ge t] \le \inf \left\{ e^{-\theta t} f^*_X(-\theta) \mid \theta \ge 0 \right\}$

M/G/1 Queueing Systems

N(t) at request departure times forms embedded Markov chain

$$E[W] = \frac{\rho E[S]}{1-\rho} \frac{1+C_S^2}{2} \quad \text{with } C_S^2 = \frac{Var[S]}{E[S]^2} = \frac{E[S^2]-E[S]^2}{E[S]^2}$$

$$E[R] = E[W] + E[S]$$

$$E[W^2] = 2E[W]^2 + \frac{\lambda E[S^3]}{3(1-\rho)} \qquad E[R^2] = E[W^2] + \frac{E[S^2]}{1-\rho}$$

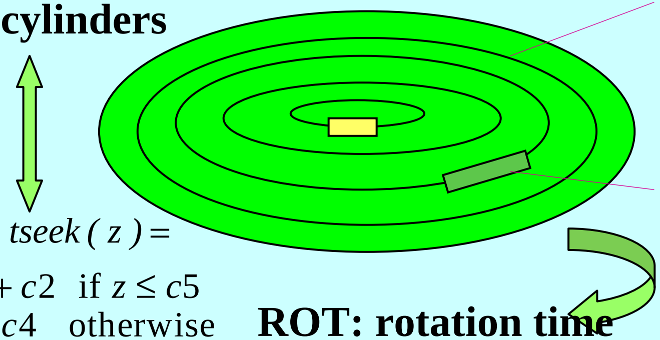
$$W^*[\theta] = \frac{(1-\rho)\theta}{\theta-\lambda+\lambda S^*(\theta)} \qquad R^*[\theta] = W^*(\theta) \cdot S^*(\theta)$$

Modeling Disk Service Times for multi-zone disk

$$C_\nu = C_{min} + \frac{(C_{max} - C_{min}) \cdot (\nu - 1)}{Z - 1}$$

$$P[dist = k \mid on\ cyl\ i] = \begin{cases} C_\nu / C_{disk} & for\ k = 0 \\ (C_{\nu-k} + C_{\nu-k}) / C_{disk} & for\ 0 < k \le \nu \le Z-1-k \\ C_{\nu-k} / C_{disk} & for\ k > 0\ and\ i > Z-1-k \\ C_{\nu+k} / C_{disk} & for\ k > 0\ and\ \nu < k \end{cases}$$

Z: #cylinders



C_i: track capacity

$B_i = C_i / ROT$: disk transfer rate

R: request size

$$T_{seek} = tseek(z) = \begin{cases} c1\sqrt{z} + c2 & \text{if } z \le c5 \\ c3\, z + c4 & \text{otherwise} \end{cases}$$

arm seek time

ROT: rotation time

T_{rot} rotational delay

$T_{trans} = R / B_i$

transfer time

$$f_{dist}(k) = P[dist = k] = \sum_i P[dist = k \mid on\ cyl\ i]$$

$$F_{seek}(t) = \begin{cases} F_{dist}(((t - c2)/c1)^2) & for\ t \le c1\sqrt{c5} + c2 \\ F_{dist}((t - c4)/c3) & otherwise \end{cases}$$

$$f_{rot}(s) = \frac{1}{ROT} \qquad f^*_{rot}(s) = \frac{1 - e^{-s\,ROT}}{s\,ROT}$$

$$C_\nu = C_{min} + \frac{(C_{max} - C_{min}) \cdot (\nu - 1)}{Z - 1} \qquad B_\nu = C_\nu / ROT \qquad P[B \le B_i] = \sum_{\nu=1}^{i} C_\nu \Big/ \sum_{\nu=1}^{Z} C_\nu$$

$$F_{rate}(r) = \frac{(C_{min}/ROT + r)(r - Zr + ZC_{min}/ROT - C_{max}/ROT)}{(C_{min} + C_{max})Z(C_{min} - C_{max})/ROT^2}$$

$$F_{trans}(t) = \int_{r=C_{min}/ROT}^{C_{max}/ROT} f_{rate}(r) F_{size}(tr)\, dr$$

manageable with computer algebra tools like Maple or Matlab

Stochastic Response Time Prediction

for multi-zone disk with seek-time function $t_{seek}(x)$, Z tracks of capacity $C_{min} \le C_i \le C_{max}$, rotation time ROT, disk load λ_{disk}

$$f_R(t) = \sum_{i=1}^{n} \beta_i \, p_i \, f_{Rcache}(t) + \beta_i (1 - p_i) \, f_{Rdisk}(t)$$

$$f^*_R(s) = \sum_{i=1}^{n} \beta_i (1 - p_i) \, f^*_{Rdisk}(s)$$

$$f^*_{Rdisk} = f^*_{serv}(s) \frac{s(1-\rho)}{s - \lambda_{disk} + \lambda_{disk} f^*_{serv}(s)}$$

$$f^*_{serv}(s) = f^*_{seek}(s) \, f^*_{rot}(s) \, f^*_{trans}(s)$$

with M/G/1 queue:

$$\lambda_{disk} = \lambda \sum_{i=1}^{n} \beta_i (1 - p_i)$$

$$\rho = \lambda_{disk} E[t_{serv}]$$

Cache Sizing: Putting It All Together

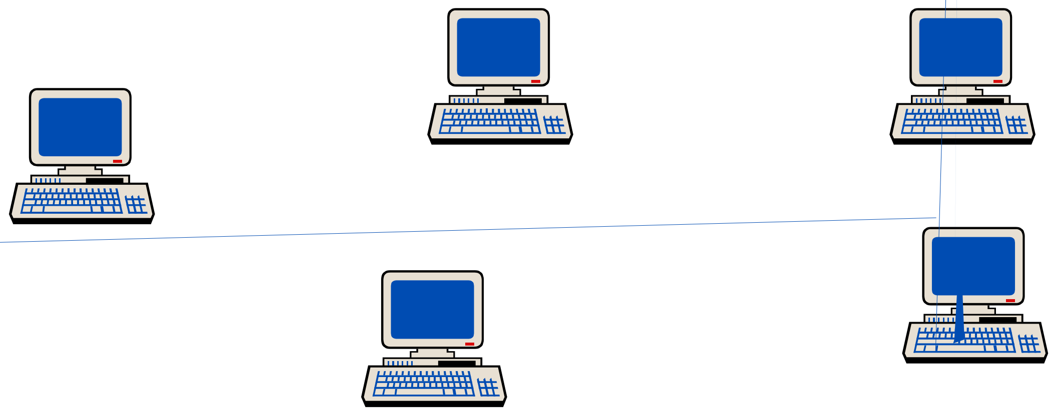
We can now:

- predict the **cache hit ratio** and
  the **page-access response time** (mean and quantiles)
  for given cache size M
- predict **transaction response times** by accumulating page accesses
- solve for **smallest M** that satisfies **response time goal**

Stochastic Model for P2P Message Flooding

Gnutella-style „blind search“:

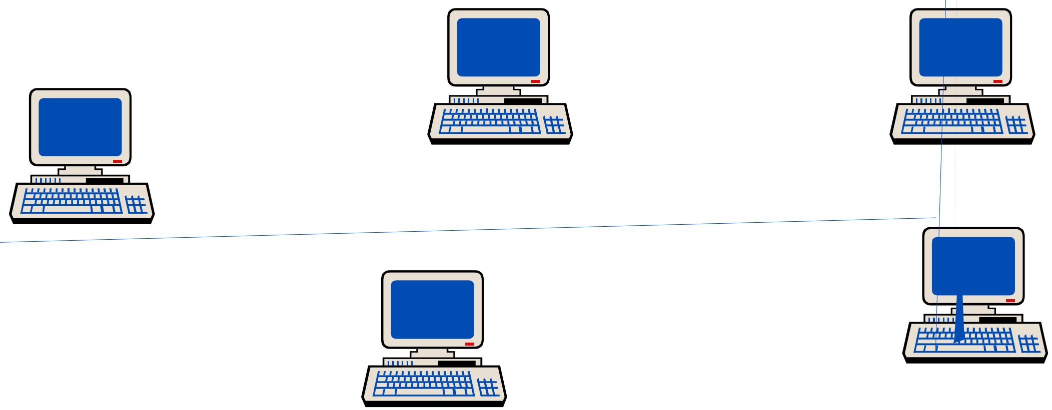
forward query to (random subset of) neighbors,
with TTL reduced at each hop



Stochastic Model for P2P File Swarming

BitTorrent-style file chunk (coupon) collecting:

pick peer & replicate one of its (rare) chunks;
leave (a while) after completing your chunk set



Part 2: Five Auto-Tuning Paradigms

1 Auto-Tuning as Tradeoff Elimination
2 Auto-Tuning as Static Optimization with Deterministic Input
3 Auto-Tuning as Static Optimization with Stochastic Input
- Capacity Planning
- Example: Cache Sizing
- Queueing Theory
- **Further Aspects and Lessons**

4 Auto-Tuning as Online Optimization
5 Auto-Tuning as Feedback Control Loop

Dependability Measures

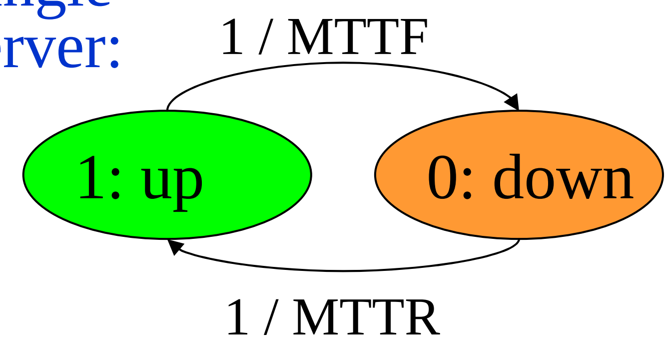
- **Failure tolerance:** ability to recover from failures
- **Failure masking:** ability to hide failures from application program
- **Reliability:** time until failure (a random variable);
  usually given by the expectation value
- **Availability:** probability of service (at random time point);
  often given by #nines (e.g., 99.99 % ≈ 1 hour downtime per year)
- **Performability:** performance with consideration of
  service degradation due to (transient) component failures

Availability Example

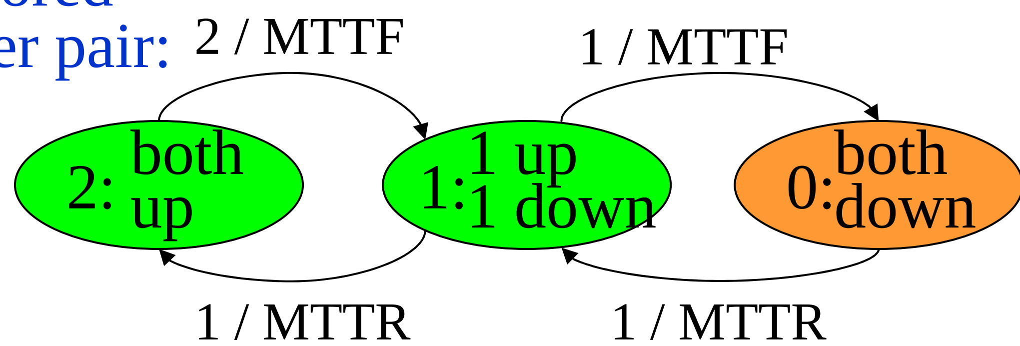
only transient, repairable failures

availability = P[system is operational at random time point]

Single server:



Mirrored server pair:



p0 / MTTR = p1 / MTTF
p1 /MTTF = p0 / MTTR
p0 + p1 = 1

$$\Rightarrow \quad p1 = \frac{MTTF}{MTTF + MTTR}$$

availability of server

p1 / MTTR = 2 p2 / MTTF
2 p2 / MTTF + p0 / MTTR =
p1 / MTTR + p1 / MTTF
p1 / MTTF = p0 / MTTR
p0 + p1 + p2 = 1

$$\Rightarrow \quad p2 \approx \frac{MTTF^2}{2}$$

availability of server pair

Lessons and Problems

Lessons:

- stochastic models are key to predicting performance for workloads with statistical fluctuation, and thus key for capacity planning and system

Literature (1) on II.3: Static Optimization with Stochastic Input

- A. Allen: Probability, Statistics, and Queueing Theory with Computer Science Applications, Academic Press, 1990
- R. Nelson: Probability, Stochastic Processes, and Queueing Theory, Springer 1995
- R.A. Sahner, K.S. Trivedi, A. Puliafito: Performance and Reliability Analysis of Computer Systems, Kluwer, 1996
- B.R. Haverkort: Performance of Computer Communication Systems, Wiley 1998
- D.A. Menasce, V.A.F. Almeida: Capacity Planning for Web Performance – Metrics, Models, and Methods, Prentice Hall, 1998
- C. Millsap: Optimizing Oracle Performance, O'Reilly, 2003
- C.K. Wong: Algorithmic Studies in Mass Storage Systems, Computer Science Press, 1983
- E.G. Coffman Jr., M. Hofri: Queueing Models of Secondary Storage Devices, Queueing Systems 1(2), 1986
- C. Ruemmler, J. Wilkes: An Introduction to Disk Drive Modeling, IEEE Computer 27(3), 1994
- J. Wilkes, R.A. Golding, C. Staelin, T. Sullivan: The HP AutoRAID Hierarchical Storage System, ACM TOCS 14(1), 1996

Literature (2) on II.3: Static Optimization with Stochastic Input

- E.A.M. Shriver, A. Merchant, J. Wilkes: An Analytic Behavior Model for Disk Drives with Readahead Caches and Request Reordering, SIGMETRICS 1998
- G.A. Alvarez et al.: Minerva: An Automated Resource Provisioning Tool for Large-Scale Storage Systems, ACM TOCS 19(4), 2001
- A. Dan, P.S. Yu, J.-Y. Chung: Database Access Characterization for Buffer Hit Prediction, ICDE 1993
- G. Nerjes, P. Muth, G. Weikum: Stochastic Service Guarantees for Continuous Data on Multi-Zone Disks, PODS 1997
- M. Gillmann, G. Weikum, W. Wonner: Workflow Management with Service Quality Guarantees, SIGMOD 2002
- A.E. Dashti, S.H. Kim, C. Shahabi, R. Zimmermann: Streaming Media Server Design, Prentice Hall, 2003
- L. Massoulie, M. Vojnovic: Coupon Replication Systems, SIGMETRICS 2005
- Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker: Search and Replication in Unstructured Peer-to-Peer Networks, ICS 2002
- J. Kleinberg: Complex Networks and Decentralized Search Algorithms, ICM 2006

Outline

- Part I: What Is It All About
- **Part II: Five Auto-Tuning Paradigms**
  - 1 Auto-Tuning as Tradeoff Elimination
  - 2 Auto-Tuning as Static Optimization with Deterministic Input
  - 3 Auto-Tuning as Static Optimization with Stochastic Input
  - **4 Auto-Tuning as Online Optimization**
  - 5 Auto-Tuning as Feedback Control Loop
- Part III: Wrap-up

Auto-Tuning as Online Optimization

Memory Governance
Histogram Maintenance

Online Algorithms

- **Characteristics:**
  - Deal with a sequence of events
  - Future events are unknown to the algorithm
  - The algorithm has to deal with one event at each time**.**
- **Goodness with respect to *uncertainty* measured via *competitive ratio***
  - Compare to offline algorithm with full knowledge of the input
- **Competitive ratio alone is not a sufficient criteria**

Memory Governance

- **Memory = Other Processes + DB**
  - Query OS on the amount of free physical memory
  - Respond to Memory availability
- **DB = Shared Cache + Working Memory**
  - No good answer on how to split across the two
- **Working Memory = Sum (WorkingO-Memory)**
  - Hope is to leverage characteristics of SQL operators
  - No formal problem definition
  - We will look at the state of the art

Shared Cache

- **Buffer Pool**
  - Events are page references
  - Minimize page fault
  - LRU is k-competitive (LB), LFU is unbounded
  - Competitiveness alone is not sufficient
- **Shared Cache more than Buffer Pool**
  - Procedure cache (compiled query plans)
  - Split across different classes
    - Multi-class workload, variant of cache replacement problem

Working Memory Assignment

- **Query Operators must be adaptive with memory assignment**
  - May be assumed with some limitations
  - We will look at Hash Join
  - No formal study of implementations in an online memory adaptive framework ([Barve, Vitter 1994])

Roadmap

- ***Adaptive operators***
- Allocation problem (ROC)
- Example of Memory Governance in Products
- Troubleshooting Memory Pressure

Making Hash Join Memory Adaptive

- **In Memory:Grace Hash: Recursive Hash**
- **Role Reversal**
- **Memory fluctuation across “steps”**
  - Adjust cluster size for partitioning buffers
  - Maximize size of write requests (e.g., flush largest partition to give up memory)
- **Fluctuation during steps**
  - +: Enlarge buffers for build as well as probe
  - -: Reduce partition buffer, not input buffers
  - -: Bit Vector Filtering

Roadmap

- Brief discussion of cache management
- Adaptive operators
- ***Allocation problem (ROC)***
- Example of Memory Governance in Products
- Troubleshooting Memory Pressure

Allocation Problem

- **Challenges: Characterizing each operator**
  - Take into account memory vs. response time profiles of each stage of adaptive operators
    - To estimate value of incremental memory
- **Challenges: Mid-flight changes**
  - Cardinality: Optimizer estimates not reliable
  - Progress of an operator/stage
- **Challenges: Handling multiple operators**
  - Criteria for distribution across operators
  - Preemption, admission control as mechanisms

ROC Framework for Allocation

- **ROC (Return on Consumption) = benefit/cost of incremental memory**
  - Identify dependence on incremental memory for the “current” phase of an operator
  - Capture space-time product
  - $ROC(M) = (T(M_0) – T(M)) / (M*T(M) – M_0*T(M_0))$
- **Optimization problem based on ROC**
  - Still need to resolve multi-operator assignment

Challenges in ROC Model

- **Derive Δperf/ ΔMi for each operator**
  - Decision to take away memory interacts with implied IO costs
  - Limited work on modeling adaptive join operators (Davidson 1995 thesis)
- **Balancing across query groups in the workload may be important**
  - Criticality (OLTP, OLAP, DSS)
  - Small, Medium or Large operands
  - Resource Brokering framework based on ROC (Davidson, Graefe)

Roadmap

- Brief discussion of cache management
- Adaptive operators
- Allocation problem (ROC)
- ***Example of Memory Governance in Products (Oracle and Microsoft)***
  - ***See DB2 paper in VLDB06***
- Troubleshooting Memory Pressure

Example: Approach in Microsoft SQL Server

- **Shared cache**
  - Procedure cache (high cost of replacement) and data page buffers
- **Compile Time**
  - For each operator phase, a min and max memory value is assigned
    - Based on expected cardinalities
  - For multiple concurrently executing phases, division is proportional to expected work (a fraction is assigned)

SQL Server Memory Management (2)

- **Run time**
  - At least min, but give Max if available
  - Below a threshold of total memory
- **Use admission control**
  - Queue new requests instead of preempting active operators
- **Waiting operators and waiting memory**
  - Waiting operators release memory to active operators on-demand
  - Longest waiting operator first to free memory

Oracle Workspace Memory Management

- Adaptive operators modeled with
  - Max, Min setting for memory
- A memory target **M** is provided
- Active Work Area Profiles for each active operator
  - At least Min
  - Below 5% of overall limit of working memory
  - *Fairness:* At most (max_requirement, **g**)
- Memory **M** is distributed among all of them as an optimization problem to maximize **g**

Oracle: Setting Memory Target

- Do you have to adjust Memory Target?
  - DBA induced change
  - Wrong allocation due to slow response of operators or fragmentation
  - Statistical advice from simulator (Memory Target vs. Percentage of In-Memory executions)
- Global bound recomputed frequently in the background
  - Active re-computation needed for severe cases
    - Bootstrapping from idle state

Roadmap

- Brief discussion of cache management
- Adaptive operators
- Allocation problem (ROC)
- Example of Memory Governance in Products
- ***Troubleshooting Memory Pressure***

Troubleshooting Memory Pressure

- **Manifestation of memory pressure**
  - Cache hit ratio/Page Life Expectancy/ IO subsystem under stress
  - Too many recompilations
  - Length of Memory grant queue
- **Possible Solution:**
  - Fix Physical Designs
  - Fix SQL statement and compilation
  - Set transaction isolation level carefully

Lessons and Problems

- Lessons
  - Cache (Buffer Pool) replacement reasonably solved
  - Static optimization not a feasible approach
  - Memory pressure due to many different reasons
  - Use of built-in simulators
- Problems
  - Allocation problem & incremental value of memory analysis open

References (Memory Management)

- Weikum G., Konig C., Kraiss A., Sinnwell, M. Towards Self-Tuning Memory Management for Data Servers, IEEE Data Engineering Bulletin 22(2): 3-11, 1999.
- Yu P., Cornell D. Buffer Management Based on Return on Consumption in a Multi-Query Environmentt, VLDB Journal 2(1): 1-37, 1993.
- Brown K., Carey M., Livny M., Goal-Oriented Buffer Management Revisited, SIGMOD Conference,1996.
- Surajit Chaudhuri, Eric Christensen, Goetz Graefe, Vivek R. Narasayya, Michael J. Zwilling: Self-Tuning Technology in Microsoft SQL Server. IEEE Data Eng. Bull. 22(2): 20-26 (1999)
- Per-Åke Larson, Goetz Graefe: Memory Management During Run Generation in External Sorting. SIGMOD Conference 1998: 472-483

References (Memory Management)

- Goetz Graefe, Ross Bunker, Shaun Cooper: Hash Joins and Hash Teams in Microsoft SQL Server. VLDB 1998: 86-97
- Diane L. Davison, Goetz Graefe: Dynamic Resource Brokering for Multi-User Query Execution. SIGMOD Conference 1995: 281-292
- Diane L. Davison, Goetz Graefe: Memory-Contention Responsive Hash Joins. VLDB 1994: 379-390
- Benoît Dageville, Mohamed Zaït: SQL Memory Management in Oracle9i. VLDB 2002: 962-973
- Qi S., Dang M.: The DB2 UDB Memory Model, IBM DeveloperWorks.
- Adam J. Storm, Christian Garcia-Arellano, Sam S. Lightstone, Yixin Diao, Maheswaran Surendra: Adaptive Self-tuning Memory in DB2, VLDB 2006

Auto-Tuning as Online Optimization

Histogram Maintenance

Histograms as Succinct Data Set Summaries

- Used for selectivity estimation
- Data set partitioned into buckets
  - Each bucket consists of a bounding box and aggregate statistics (count of tuples)
  - Uniformity is assumed inside buckets.
    - Histograms should partition data set in buckets with uniform tuple density.
- Multi-dimensional data makes partitioning even more challenging

Histogram Maintenance

- Scenario 1: Insert/Deletes/Updates to relation take place
  - How can we avoid rebuilding histogram from scratch?
  - "Online incremental maintenance"
- Scenario 2: No updates to relation. But, trying to construct histograms by only looking at query executions
  - How can we modify histogram as we get "additional evidence"?
  - "Online incremental correction"
  - a.k.a Self Tuning Histograms

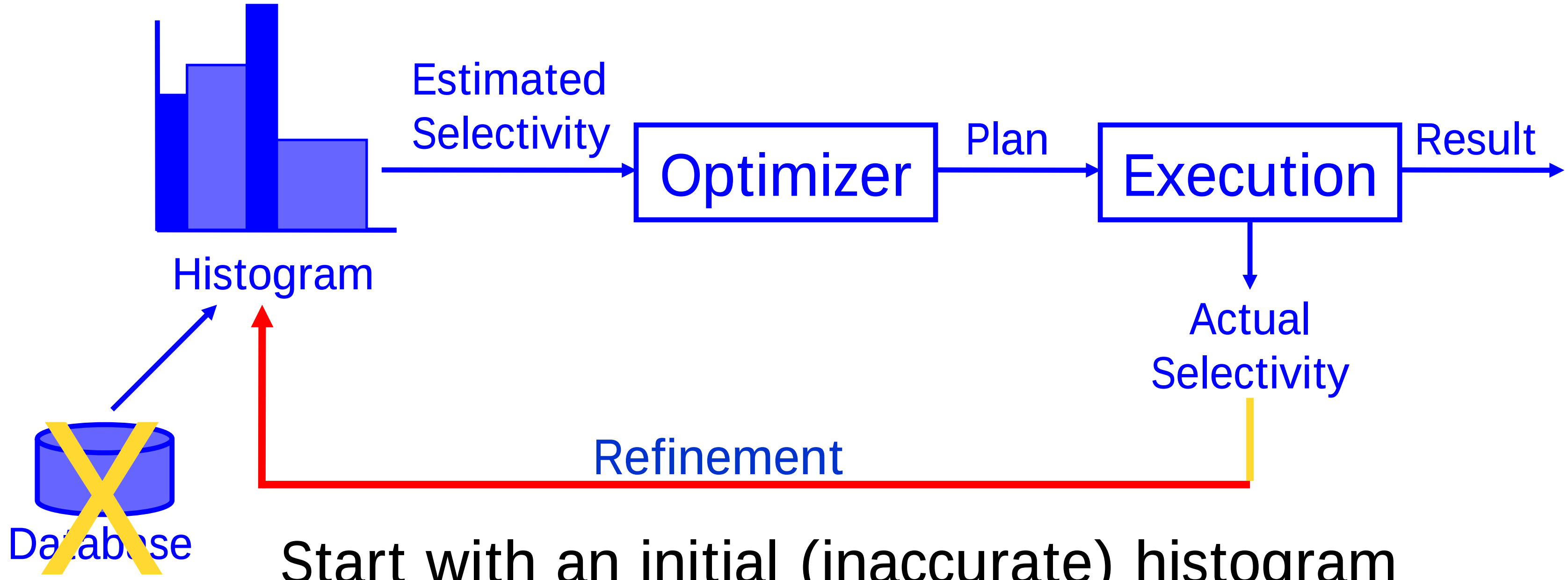
Online Incremental Maintenance

- Maintain a sample of the relation incrementally (Gibbons,Matias,Poosala V. *VLDB 1997)*
  - Insertion: Traditional Reservoir sampling
  - Modification: In-place
  - Deletion: Delete, may trigger a re-sampling (also see paper in VLDB06)
- Incrementally update histogram by changing frequency counts of buckets
  - Detect unbalanced buckets (std deviation)
- If the histogram is not “balanced”, use the sample to rebuild histogram

Histogram Maintenance

- Scenario 1: Insert/Deletes/Updates to relation take place
  - How can we avoid rebuilding histogram from scratch?
  - "Online incremental maintenance"
- Scenario 2: No updates to relation. But, trying to construct histograms by only looking at query executions
    - How can we modify histogram as we get "additional evidence"?
    - "Online incremental correction"
    - a.k.a Self Tuning Histograms

Self-tuning Histograms



Start with an initial (inaccurate) histogram and refine it based on feedback

Online Incremental Correction

- Does not examine actual data set
- Assume uniformity and independence until feedback shows otherwise
- Uses Split and Merge techniques
  - Each query defines a potential new bucket if cardinality error is above threshold
  - Merge victims are chosen based on adjacency and similarity of density
- Goal: Error minimized if the workload is replayed.
- Contrast with online incremental maintenance technique..

Evaluation Metric

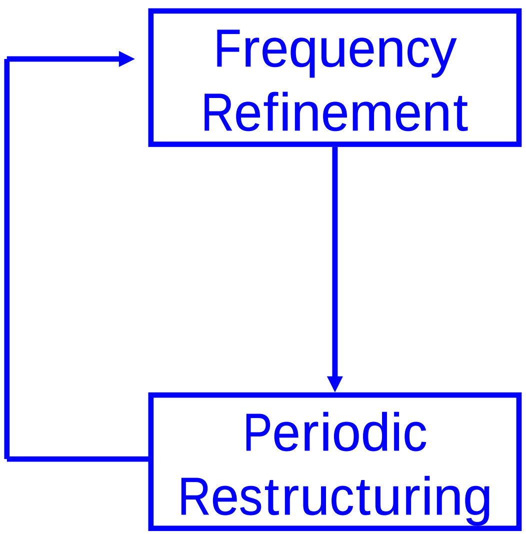
- **Absolute Error:**

$$E(D,H,W) = \frac{1}{|W|} \sum_{q \in W} \left| est(H,q) - act(D,q) \right|$$

- **Normalized Absolute Error:**

$$NAE(D,H,W) = \frac{\sum_{q \in W} |est(H,q) - act(D,q)|}{\sum_{q \in W} |est_{unif}(D,q) - act(D,q)|}$$

Refining STGrid Histograms



Observe error and accumulate information about data distribution in histogram buckets

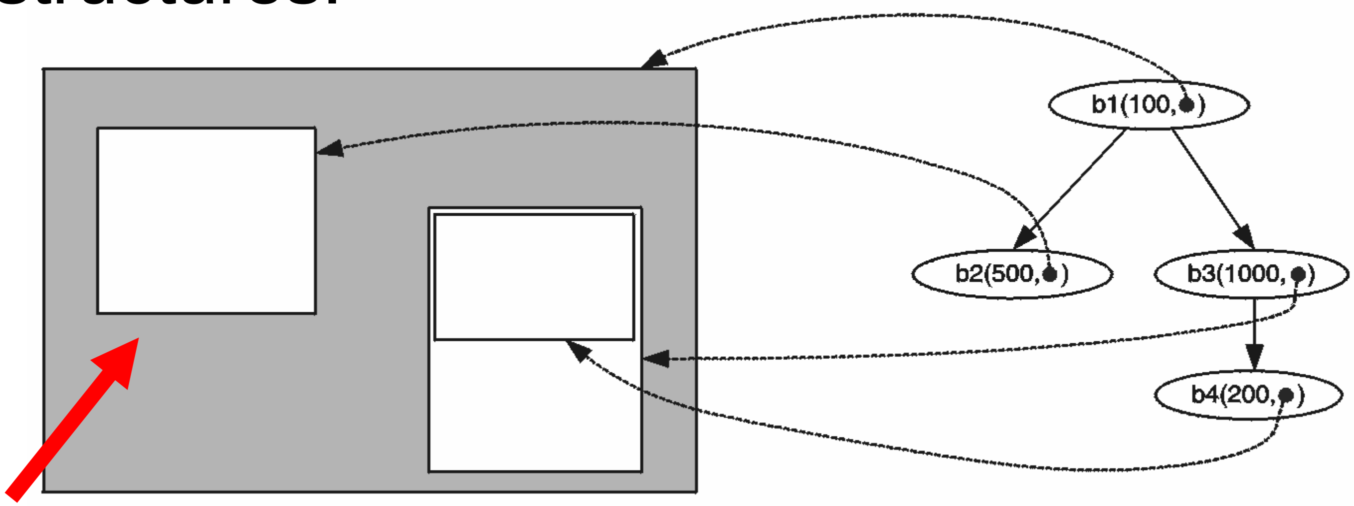
Better bucket boundaries

Split high frequency buckets

Merge buckets with similar frequencies

STHoles Histograms

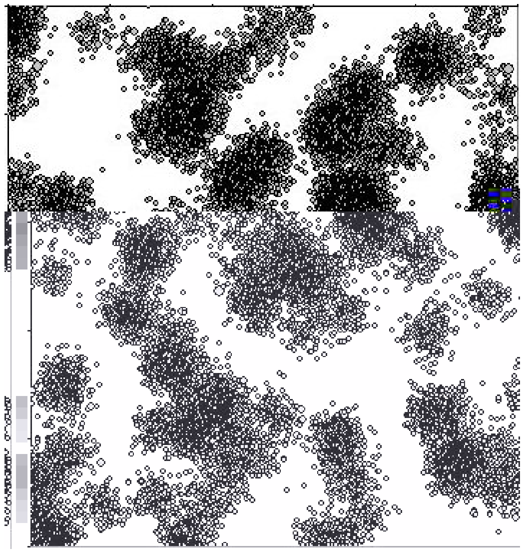
- Tree structure among buckets.
- Buckets with holes: relaxes rectangular regions while using rectangular bucket structures.



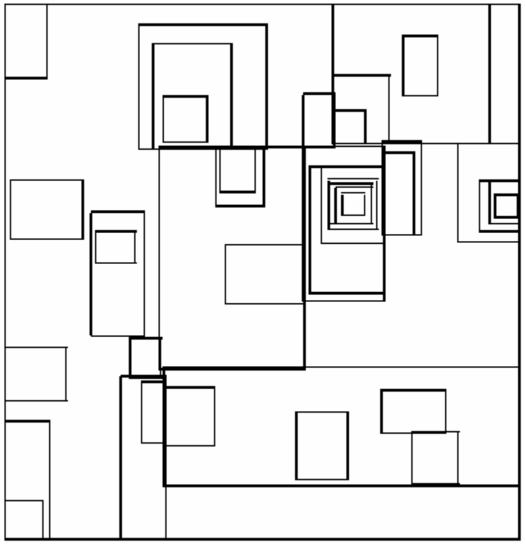
Non rectangular region

$$est(H,q) = \sum_{b \in H} f(b) \frac{v(q \cap b)}{v(b)}$$

Example STHoles Histogram



Gaussian Data Set



STHoles Histogram

Refining STHoles Histograms

- Initialize histogram H assuming uniformity.
- For each query q in workload:

  1- Gather simple statistics from query results.

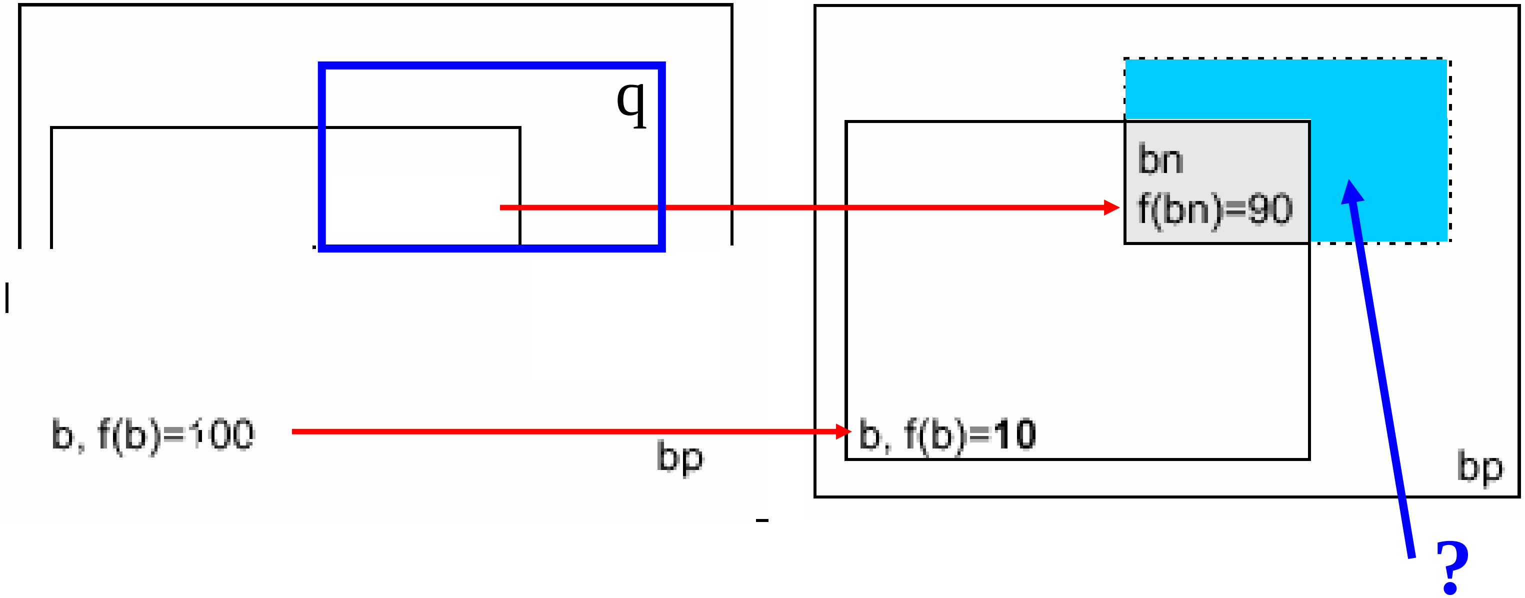
  2- Identify candidate holes and *drill* (add) them as new buckets in H.

  3- Merge superfluous buckets in H.

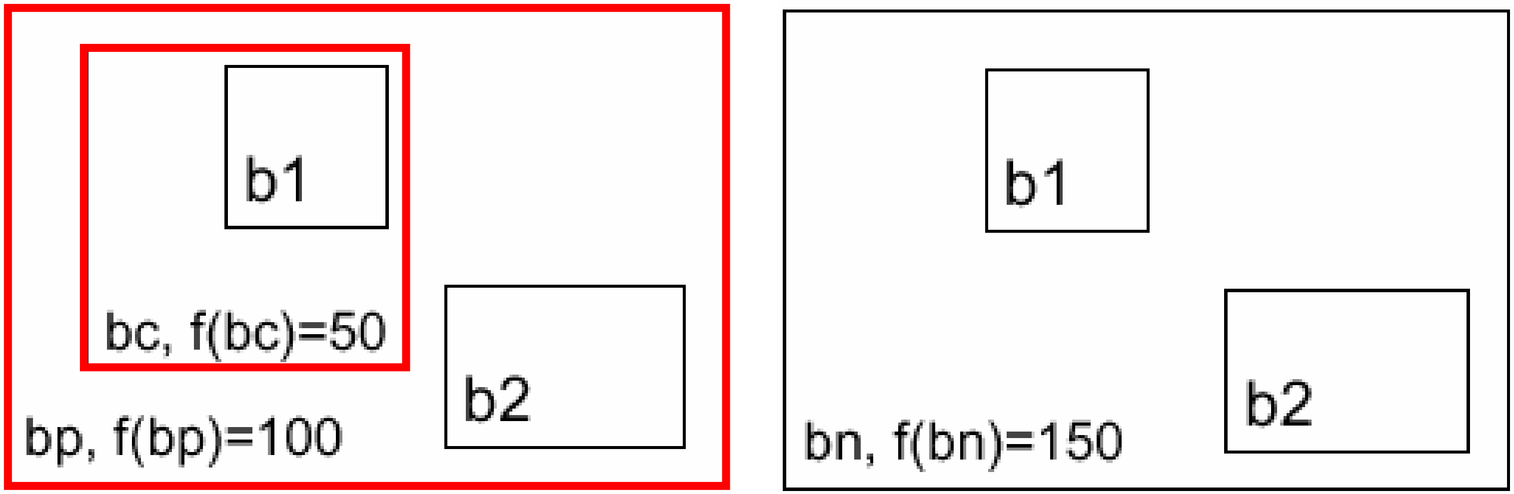
Drilling New Candidate Buckets

For each query *q* in workload and bucket b in histogram:

- Count how many tuples in result stream lie inside *q*∩*b.*
- Drill *q*∩*b* as a new bucket (child of *b*).



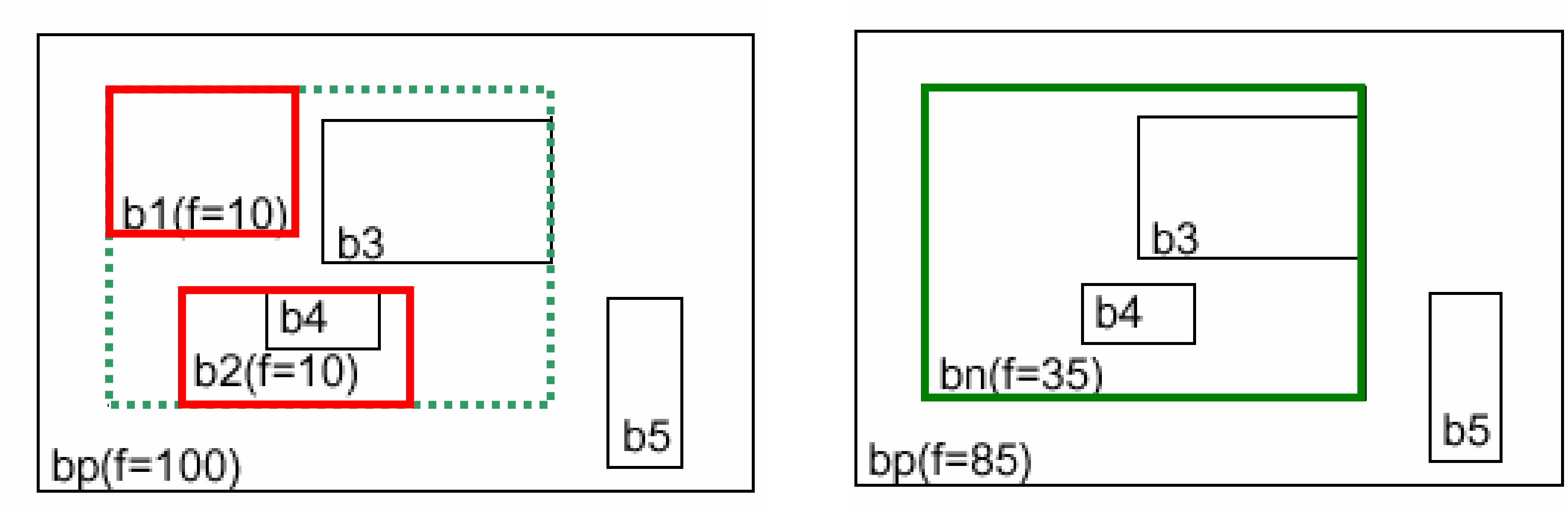
Parent-Child Merges



Eliminate buckets too similar to their parents.

Example: The interesting region in *bc* is covered by its child *b1*.

Sibling-Sibling Merges



- Consolidate buckets with similar densities that cover close regions.
- Extrapolate frequency distributions to yet unseen regions.

Accuracy vs. Overhead

- STGRID
  - Too coarse grained usage of feedback
- STHOLES
  - Accurate, but per-bucket tracking can be expensive
- ISOMER [Srivastava+06]
  - Use maximum entropy principle to divide the inaccuracy across buckets

Lessons and Problems

- Lessons
  - Maintenance: Precise, online threshold driven
    - Needs auxiliary structures for correctness
  - Correction: An attractive approach because it avoids offline a priori decisions
- Problems
  - Correction:
    - Target optimization function alternatives
    - Analysis of convergence

References (Histogram Maintenance)

- Gibbons, P., Matias Y., Poosala V. Fast Incremental Maintenance of Approximate Histograms. *VLDB 1997.*
- Chung-Min Chen, Nick Roussopoulos: Adaptive Selectivity Estimation Using Query Feedback. SIGMOD Conference 1994: 161-172
- Aboulnaga, A. and Chaudhuri, S., Self-Tuning Histograms: Building Histograms Without Looking at Data. *SIGMOD* 1999.
- Yossi Matias, Jeffrey Scott Vitter, Min Wang, Dynamic Maintenance of Wavelet-Based Histograms, *VLDB 2000*
- Bruno N., Chaudhuri S. and Gravano L. STHoles: A Multidimensional Workload-Aware Histogram. *SIGMOD 2001*
- Markl V., Megiddo N., Kutsch M., Tran T.M., Haas P., Srivastava U., Consistently Estimating the Selectivity of Conjuncts of Predicates. VLDB 2005
- Utkarsh Srivastava, Peter J. Haas, Volker Markl, Marcel Kutsch, Tam Minh Tran: ISOMER: Consistent Histogram Construction Using Query Feedback. ICDE 2006

Part 2: Five Auto-Tuning Paradigms

1 Auto-Tuning as Tradeoff Elimination
2 Auto-Tuning as Static Optimization with Deterministic Input
3 Auto-Tuning as Static Optimization with Stochastic Input
4 Auto-Tuning as Online Optimization
5 Auto-Tuning as Feedback Control Loop

- **Example: MPL Tuning Problem & Early Approaches**
- Feedback Control Theory
- Old Problem Reconsidered

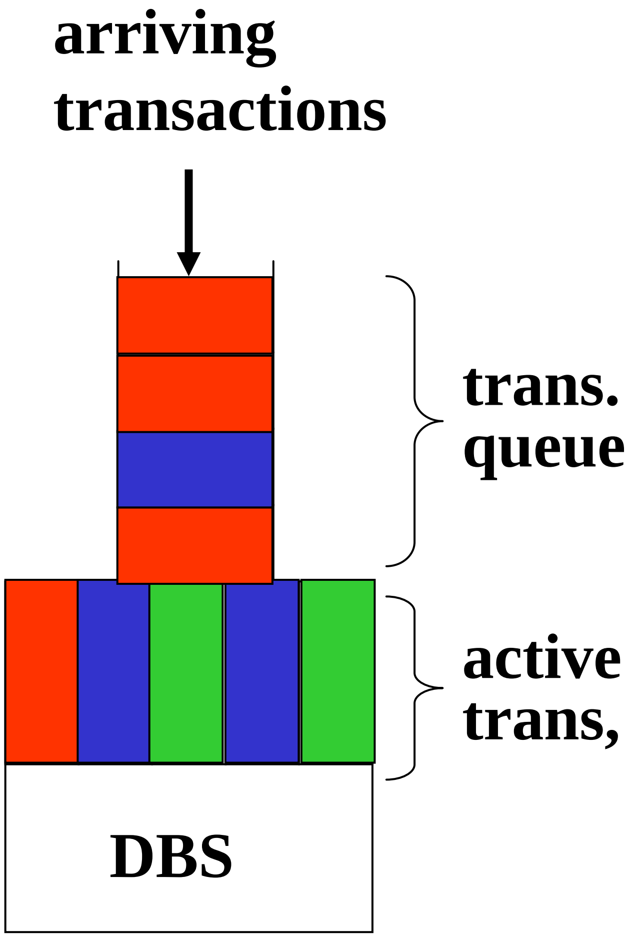
Auto-Tuning as Feedback Control Loop

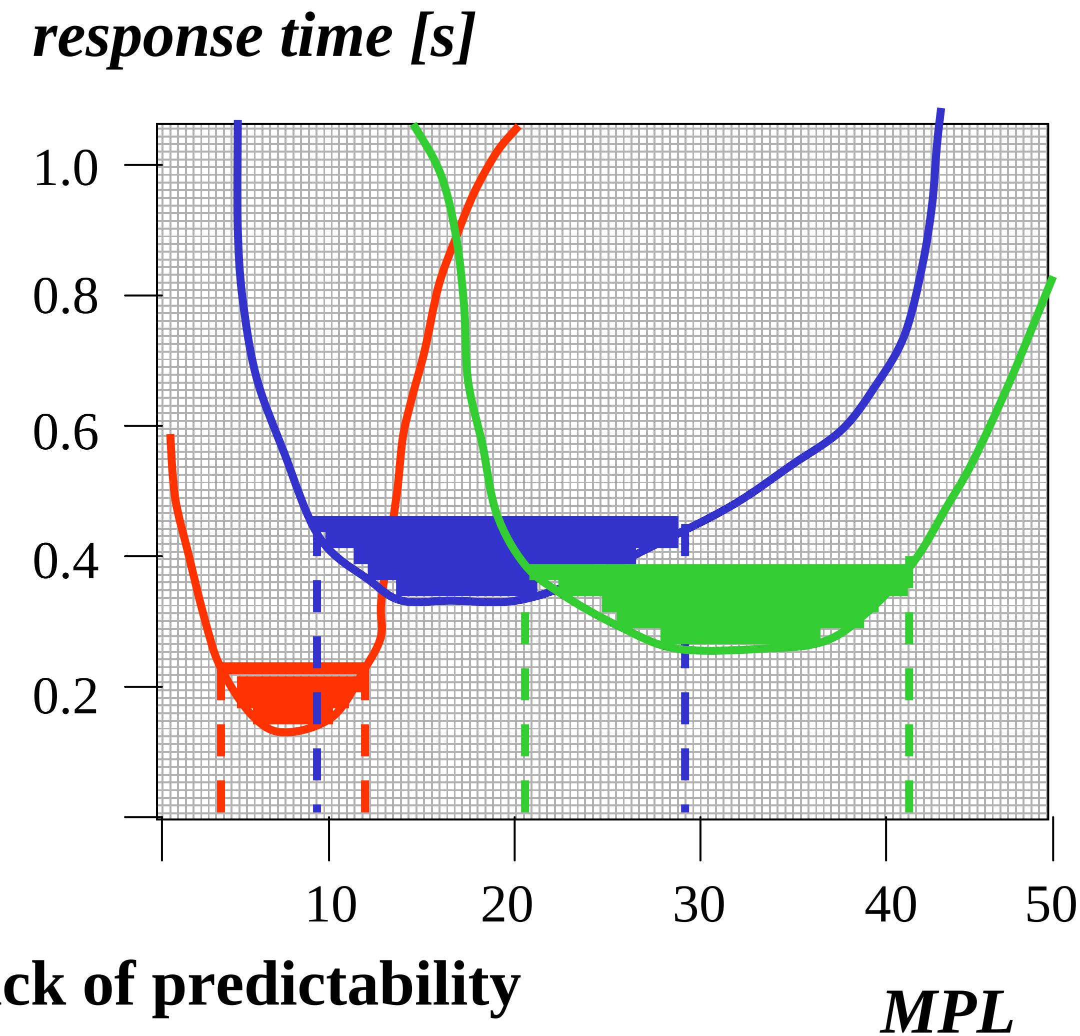
MPL Tuning (Admission Control)

- No full-fledged predictive model of system behavior
- Errors in estimation of parameters and modeling
- Rapid workload evolution: bursts and shifts

  → feedback control
  - is adaptive
  - can work with black-box system,
  - and has theoretical underpinnings

MPL Tuning with Multiple Load Classes





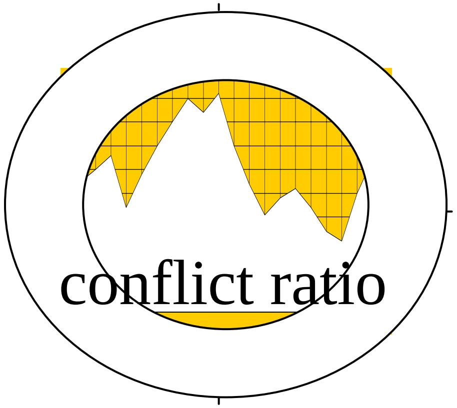
Key problem: dynamics, lack of predictability

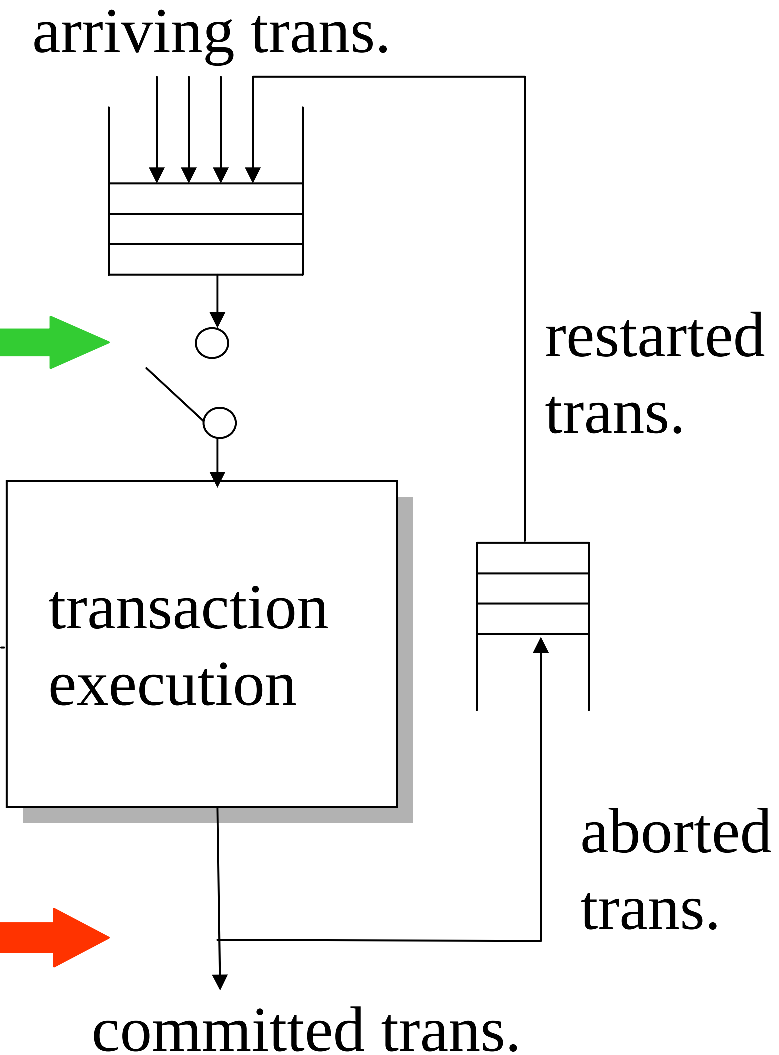
Adaptive Load Control for Avoidance of Lock-Contention Thrashing

$$\textit{conflict ratio} = \frac{\#\,\textit{locks held by all trans.}}{\#\,\textit{locks held by running trans.}}$$

transaction admission

critical conflict ratio ≈ 1.3





transaction cancellation

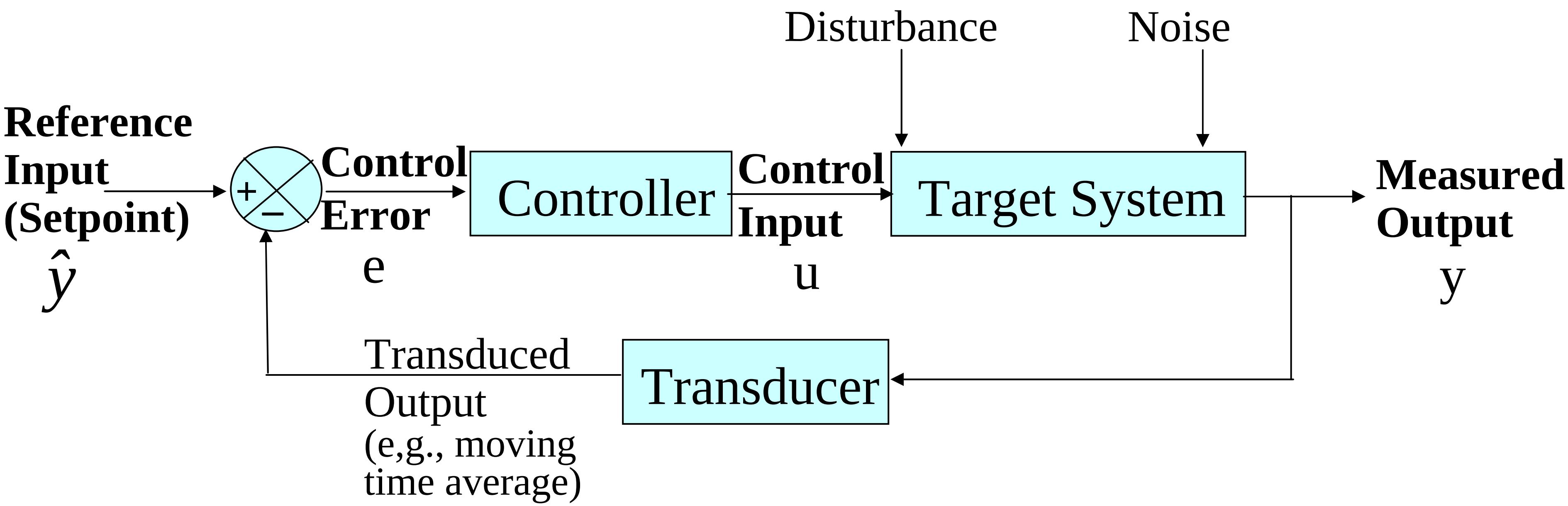
Part 2: Five Auto-Tuning Paradigms

1 Auto-Tuning as Tradeoff Elimination
2 Auto-Tuning as Static Optimization with Deterministic Input
3 Auto-Tuning as Static Optimization with Stochastic Input
4 Auto-Tuning as Online Optimization
5 Auto-Tuning as Feedback Control Loop

- Example: MPL Tuning Problem & Early Approaches
- **Feedback Control Theory**
- Old Problem Reconsidered

Basics of Feedback Control Theory

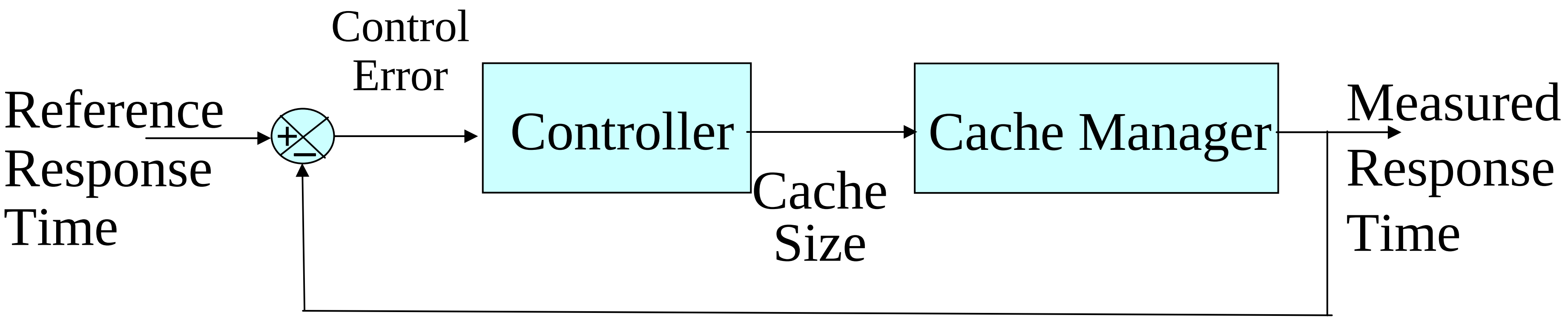
(following J.L. Hellerstein et al.: Feedback Control of Computing Systems, Wiley, 2004)



closed loop with feedback possible even for black-box system;
open loop (feedforward control) possible only with predictive model

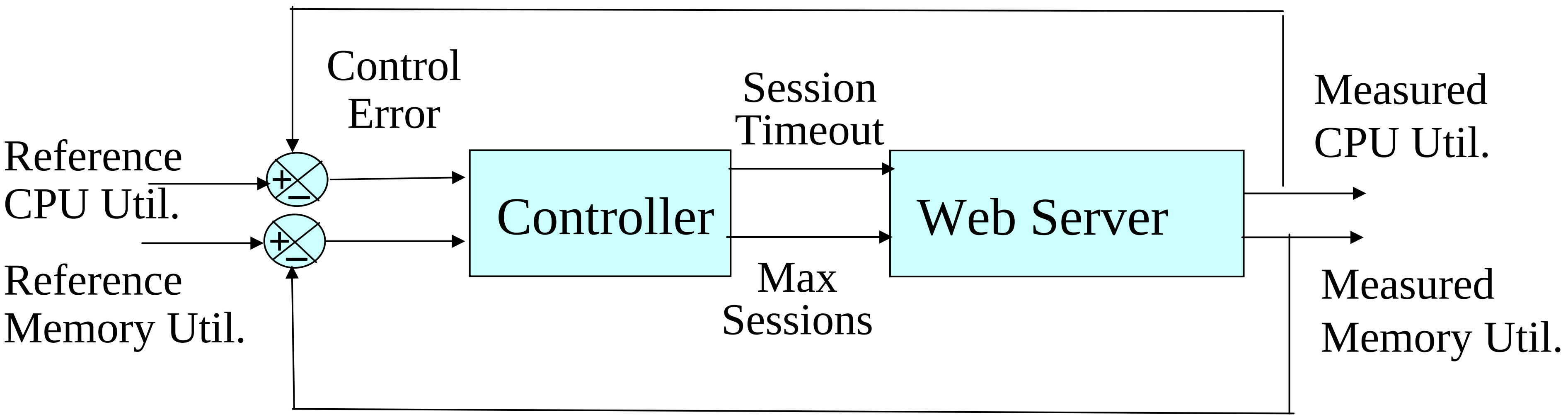
Application examples: thermostat, control valves, cruise control, ABS, building control (heating, energy, etc.)

Example: Dynamic Cache Sizing



SISO controller (single input, single output)

Example: Web Server



MIMO controller (multiple inputs, multiple outputs)

SASO Properties (1)

<u>Desired guarantees:</u>

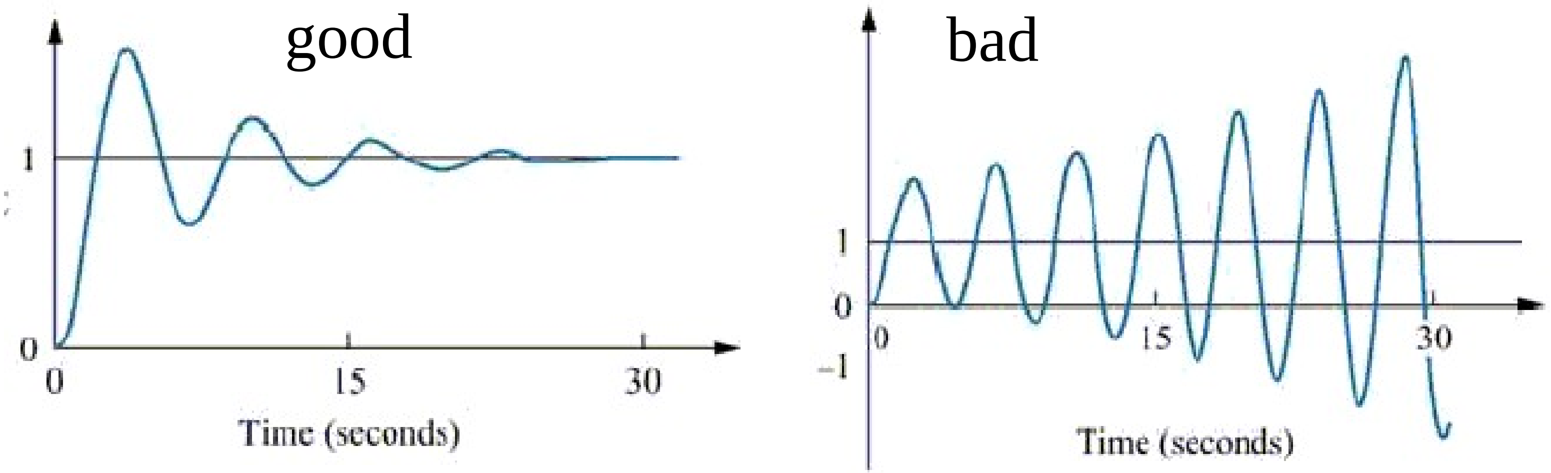
stability – bounded input results in bounded output (BIBO)

accuracy – low error between reference and measured output

short settling time – fast convergence to steady state after excitement

low overshoot – low deviation from steady-state behavior

good bad



SASO Properties (2)

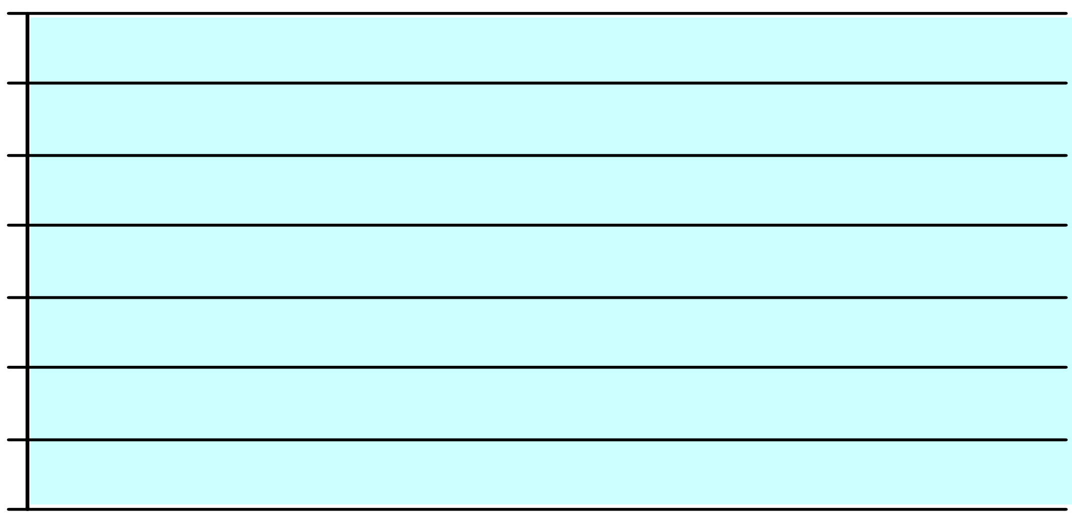
<u>Desired guarantees:</u>

stability – bounded input results in bounded output (BIBO)

accuracy – low error between reference and measured output

short settling time – fast convergence to steady state after excitement

no overshoot – low deviation from steady-state behavior



First-order Linear Models

described by difference equation with discrete time:

$$y(k+1) = ay(k) + bu(k)$$

with coefficients a, b

higher-order controller considers y(k-1), y(k-2), ...

non-linear behavior may be linearly approximated

parameters a, b derived from system model or estimated by regression

Examples:

- linearize M/M/1/K model, to control queue limit K based on resp. time
- MIMO controller for CPU and memory utilization:

$$CPU(k+1) = a_{11}CPU(k) + a_{12}Mem(k) + b_{11}Timeout(k) + b_{12}Sessions(k)$$

$$Mem(k+1) = a_{21}CPU(k) + a_{22}Mem(k) + b_{21}Timeout(k) + b_{22}Sessions(k)$$

Mathematical Tools

Z transform of discrete-time signal u:

$$U(z) = \sum_{k=0}^{\infty} u(k) z^{-k} = G_u(1/z)$$

with generating function G_u

Properties:

$$y(k) = au(k) \Rightarrow Y(z) = aU(z)$$
$$y(k) = u(k) + v(k) \Rightarrow Y(z) = U(z) + V(z)$$
$$y(k) = u(k-1) \Rightarrow Y(z) = z^{-1}U(z)$$

...

invert Z transform by table lookup, partial fraction expansion, etc.

Examples:

Impulse	$u(0)=1, u(k)=0 \text{ for } k>0$	$\Rightarrow U(z)=1$
Step	$u(k)=1 \text{ for } k \ge 0$	$\Rightarrow U(z)=z/(z-1)$
Ramp	$u(k)=k$	$\Rightarrow U(z)=z/(z-1)^2$
Exponential	$u(k)=a^k$	$\Rightarrow U(z)=z/(z-a)$
Sine	$u(k)=\sin k\theta$	$\Rightarrow U(z)=\dfrac{z\sin\theta}{z^2-(2\cos\theta)z+1}$

Transfer Function for Guaranteed Behavior

$$\boxed{F(z) = \frac{Y(z)}{U(z)}}$$

Z transform of output

Z transform of input

$$U(z) = \sum_{k=0}^{\infty} u(k) z^{-k} = G_u(1/z)$$

with generating function G_u

Transfer function of linear first-order model with y(0)=0 :

$$y(k+1) = ay(k) + bu(k)$$

$$\Rightarrow zY(z) - zy(0) = aY(z) + bU(z) \quad \Rightarrow Y(z) = \frac{bU(z)}{z-a}$$

$$\Rightarrow \boxed{F(z) = b/(z-a)}$$

<u>Theorem:</u> system is stable iff all poles of F(z) have abs ≤ 1
(poles: roots of denominator polynomial)

more theorems about convergence, steady-state error, transient responses, settling times, overshoot, oscillation, etc.

Controller Design

Proportional Control (P Control):

$$u(k) = K_p e(k)$$

with control error

$$e(k) = y(k) - \hat{y}$$

Integral Control (I Control):

$$u(k) = u(k-1) + K_I e(k)$$

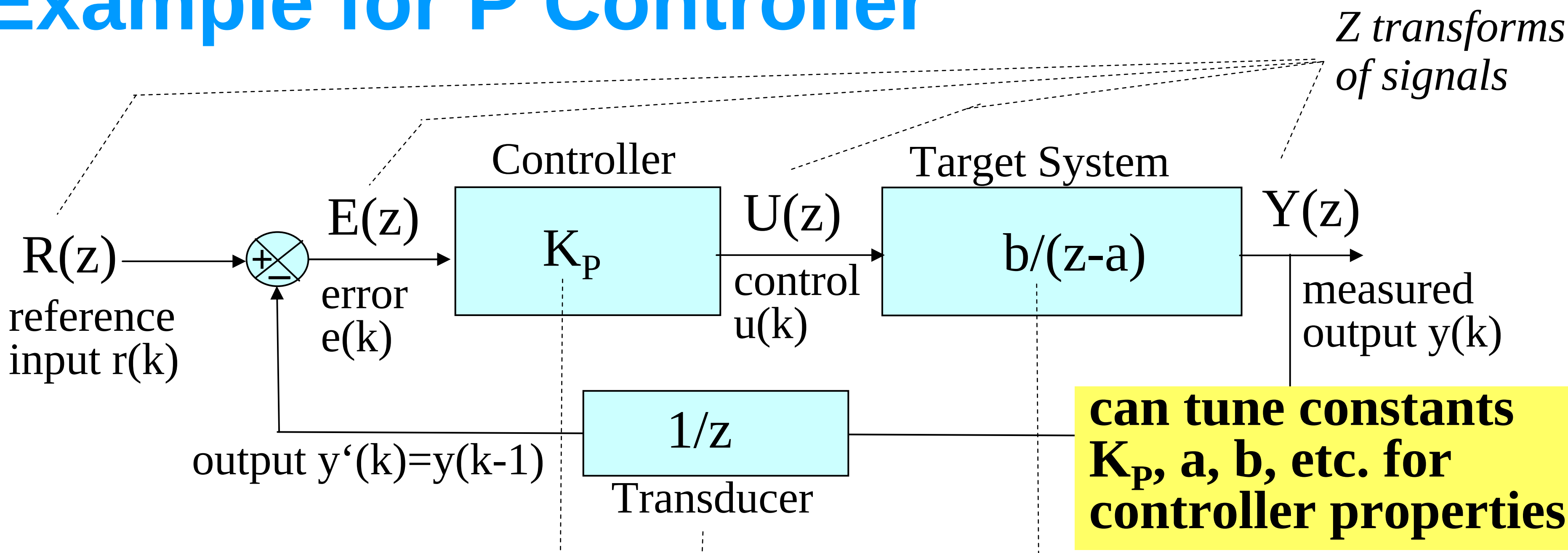
PI Control:

$$u(k) = u(k-1) + (K_P + K_I)e(k) - K_P e(k-1)$$

rich results on SASO properties

plus many more controller types

Example for P Controller



can tune constants K_P, a, b, etc. for controller properties

$$F_{EU}(z) = \frac{U(z)}{E(z)} = \frac{K_P E(z)}{E(z)} \qquad F_{YY'}(z) = \frac{Y'(z)}{Y(z)} \qquad F_{UY}(z) = \frac{Y(z)}{U(z)}$$

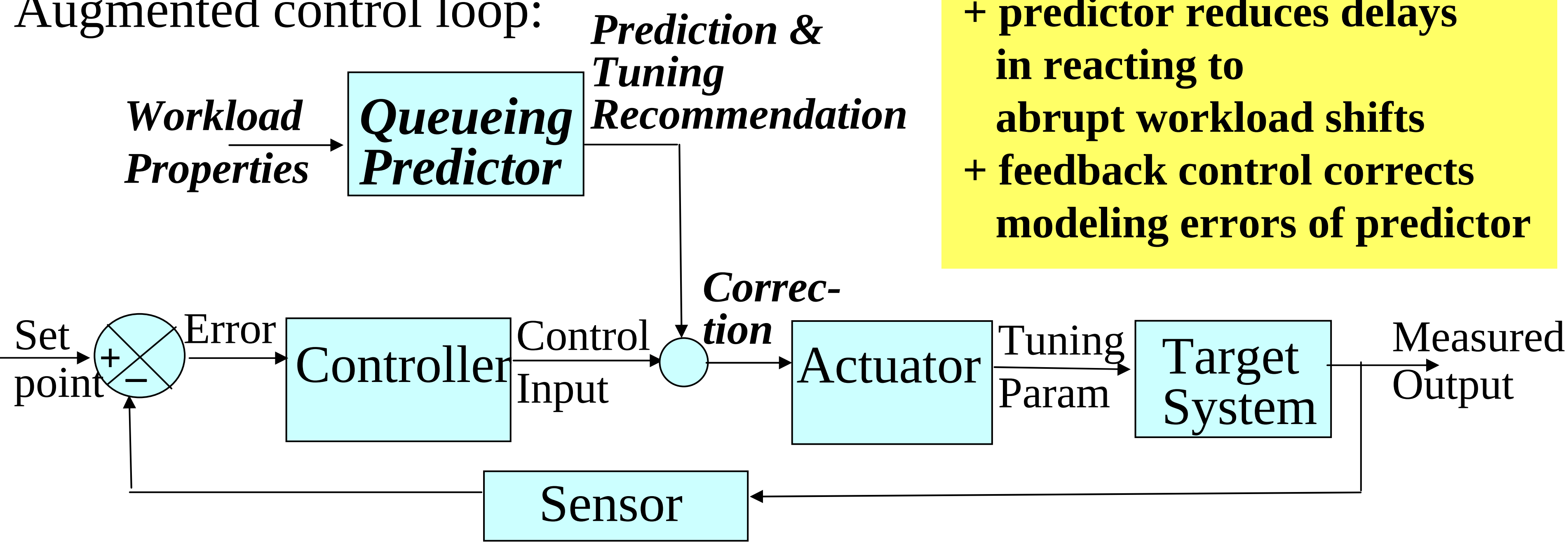
$$G(z) := F_{RY}(z) = \frac{Y(z)}{R(z)} = \frac{F_{EU}(z) F_{UY}(z)}{1 + F_{EU}(z) F_{UY}(z) F_{YY'}(z)}$$

<u>Stability Theorem:</u> system is stable iff all poles of G(z) have abs ≤ 1

more theorems about convergence, steady-state error, transient responses, settling times, overshoot, oscillation, etc.

Combining Feedback Control with Model-based Stochastic Prediction

Augmented control loop:



+ predictor reduces delays in reacting to abrupt workload shifts

+ feedback control corrects modeling errors of predictor

control resource allocations b_i ($b_i > b_{i+1}$) for multi-class workload so as to maintain relative performance guarantees g_i/g_{i+1} ($g_i < g_{i+1}$)

$$u_i(k) = u_i(k-1) + \gamma e_i(k) \quad \longrightarrow \quad \frac{b_i(k)}{b_{i+1}(k)} = \frac{b_i(k-1)}{b_{i+1}(k-1)} + \gamma \frac{g_{i+1}(k)}{g_i(k)} - \frac{W_{i+1}}{W_i}$$

Part 2: Five Auto-Tuning Paradigms

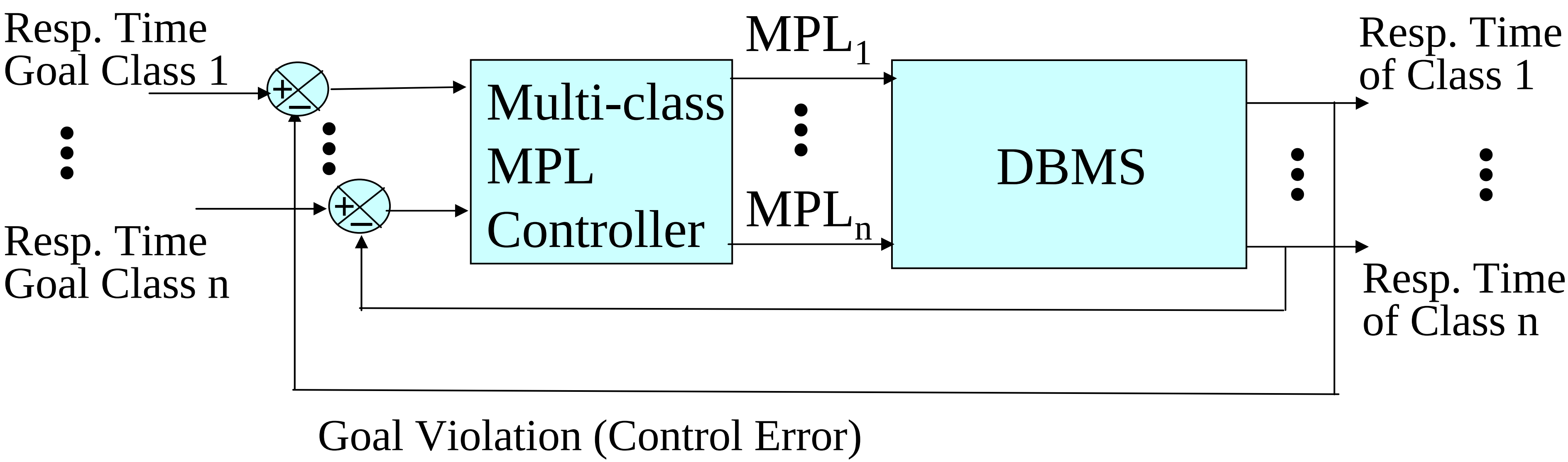
1 Auto-Tuning as Tradeoff Elimination
2 Auto-Tuning as Static Optimization with Deterministic Input
3 Auto-Tuning as Static Optimization with Stochastic Input
4 Auto-Tuning as Online Optimization
5 Auto-Tuning as Feedback Control Loop

- Example: MPL Tuning Problem & Early Approaches
- Feedback Control Theory
- **Old Problem Reconsidered**

MIMO Controller for Multi-class DBMS

for lock-contention (and memory-contention) avoidance

Intriguing (and obvious?) approach:

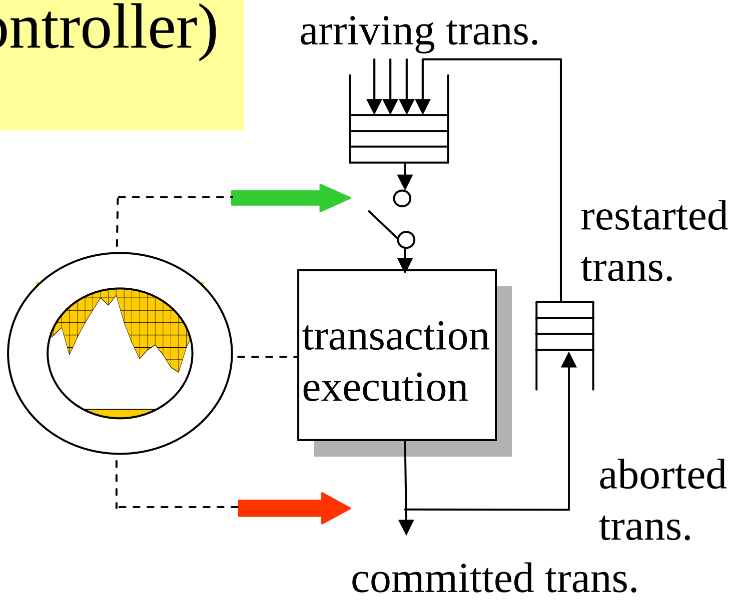


but a viable solution is not that simple!

Lock-Contention Thrashing Reconsidered

Reference input metric is crucial:

response time or wait time (to drive MPL controller) do not work robustly



need deeper insight and math to identify viable metrics and setpoints:

- conflict ratio: $\frac{\#\ \text{locks held by all trans.}}{\#\ \text{locks held by running trans.}}$
  - should be < 1.3 (backed up by math analysis)
- wait depth:
  - wait depth of running trans.: 0
  - wait depth of trans. blocked by trans. at depth i: i+1
  - limit wait depth to 1 by cancelling trans. that are blocked and block other trans.

Details of control steps are crucial:

cancellation victim selection and restart waiting

Lessons and Problems

Lessons:

- feedback control adequate for tuning issues with limited predictive/causal understanding
- no panacea: controller design can be an art
- controller fine-tuning (e.g., sampling rates) can be critical
- can (and must) be combined with other paradigms (queueing models, regression, etc.)

Problems:

- extend successful work on Web & mail servers to DBMS
- full-fledged MIMO controller for multi-class MPL tuning problem (and memory allocation) in DBMS
- from stochastic or convergence guarantees to hard predictability („bounded surprise“)
- integrate control theory into curriculum

Literature (1) on II.5: Feedback Control Loop

- J.L. Hellerstein, Y. Diao, S. Parekh, D.M. Tilbury: Feedback Control of Computing Systems, Wiley 2004 (see also tutorial at SIGMETRICS 2005)
- G.F. Franklin, J.D. Powell, M.L. Workman: Digital Control of Dynamic Systems, Addison-Wesley, 1998
- K. Ogata: Modern Control Engineering, Prentice Hall, 2001
- K.J. Astrom, R. M. Murray: Analysis and Design of Feedback Systems Preprint, 2003 http://www.cds.caltech.edu/~murray/courses/cds101/fa03/caltech/am03.html
- T.F. Abdelzaher, J.A. Stankovic, C. Lu, R. Zhang, Y. Lu: Feedback Performance Control in Software Services, IEEE Control Systems Magazine 23(3), 2003
- Y. Diao, J.L. Hellerstein, G. Kaiser, S. Parekh, D. Phung: Self-Managing Systems: A Control Theory Foundation, 2nd IEEE Conf. on Engineering of Autonomic Systems, 2005
- J.L. Hellerstein, Y. Diao, S. Parekh: A First-Principles Approach to Constructing Transfer Functions for Admission Control in Computing Systems, Conference on Decision and Control, 2002
- M. Karlsson, C. Karamanolis, X. Zhu: Triage: Performance Isolation and Differentiation for Storage Systems, Int. Workshop on Quality of Service, 2004
- Y. Lu, T. Abdelzaher, C. Lu, L. Sha, X. Liu: Feedback Control with Queueing-Theoretic Prediction for Relative Delay Guarantees in Web Servers, IEEE Real-Time and Embedded Technology and Applications Symposium, 2003

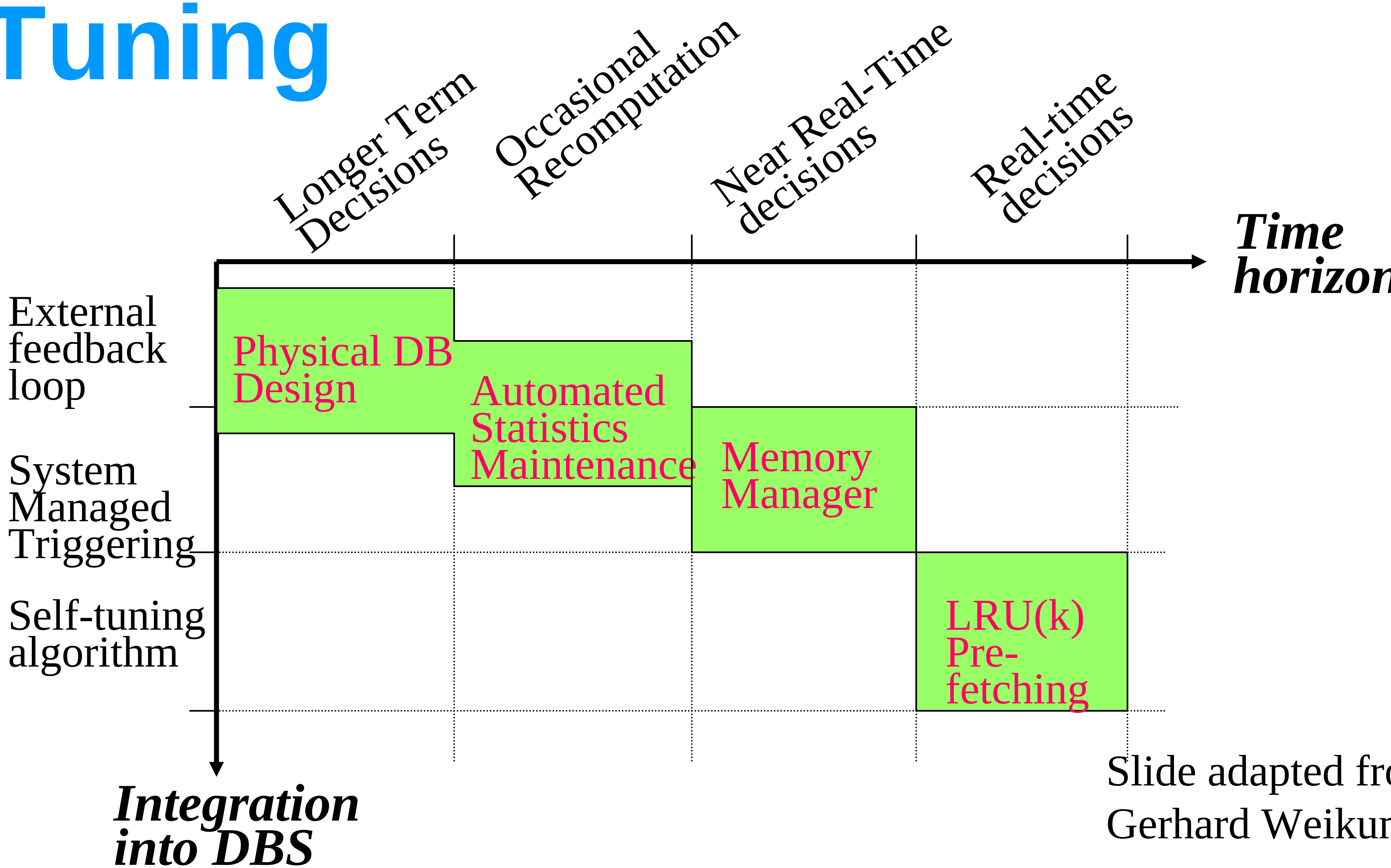
Literature (2) on II.5: Feedback Control Loop

- D. Reiner, T.B. Pinkerton: A Method for Adaptive Performance Improvement of Operating Systems, SIGMETRICS 1981
- G. Weikum, C. Hasse, A. Moenkeberg, P. Zabback: The COMFORT Automatic Tuning Project, Information Systems 19(5), 1994
- A. Thomasian: Two-Phase Locking and its Thrashing Behavior, TODS 18(4), 1993
- K.P. Brown, M. Mehta, M.J. Carey, M. Livny: Towards Automated Performance Tuning for Complex Workloads, VLDB 1994
- P.J.Denning, K.C. Kahn, J. Leroudier, D. Potier, R. Suri: Optimal Multiprogramming, Acta Informatica 7, 1976
- H.-U. Heiss: Overload Effects and Their Prevention, Performance Eval. 12(4), 1991
- S. Parekh, K. Rose, Y. Diao, V. Chang, J. Hellerstein, S. Lightstone, M. Huras: Throttling Utilities in the IBM DB2 Universal Database Server, American Control Conference, 2004
- B. Schroeder, M. Harchol-Balter, A. Iyengar, E. Nahum, A. Wierman: How to Determine a Good Multi-programming Level for External Scheduling, ICDE 06
- Y.-C. Tu, S. Liu, S. Prabhakar, B. Yao: Load Shedding in Stream Databases: A Control-based Approach, VLDB 06
- C. Pautasso, T. Heinis, G. Alonso: Autonomic Execution of Web Service Compositions, ICWS 05

Outline

- Part I: What Is It All About
- Part II: Five Auto-Tuning Paradigms
  1 Auto-Tuning as Tradeoff Elimination
  2 Auto-Tuning as Static Optimization with Deterministic Input
  3 Auto-Tuning as Static Optimization with Stochastic Input
  4 Auto-Tuning as Online Optimization
  5 Auto-Tuning as Feedback Control Loop
- **Part III: Wrap-up**

The Spectrum for Self-Tuning



Slide adapted from Gerhard Weikum

Other Notable Areas for Automated Tuning

- Statistics management
- Choice of isolation levels
- Application tuning
- Tuning of middleware caching

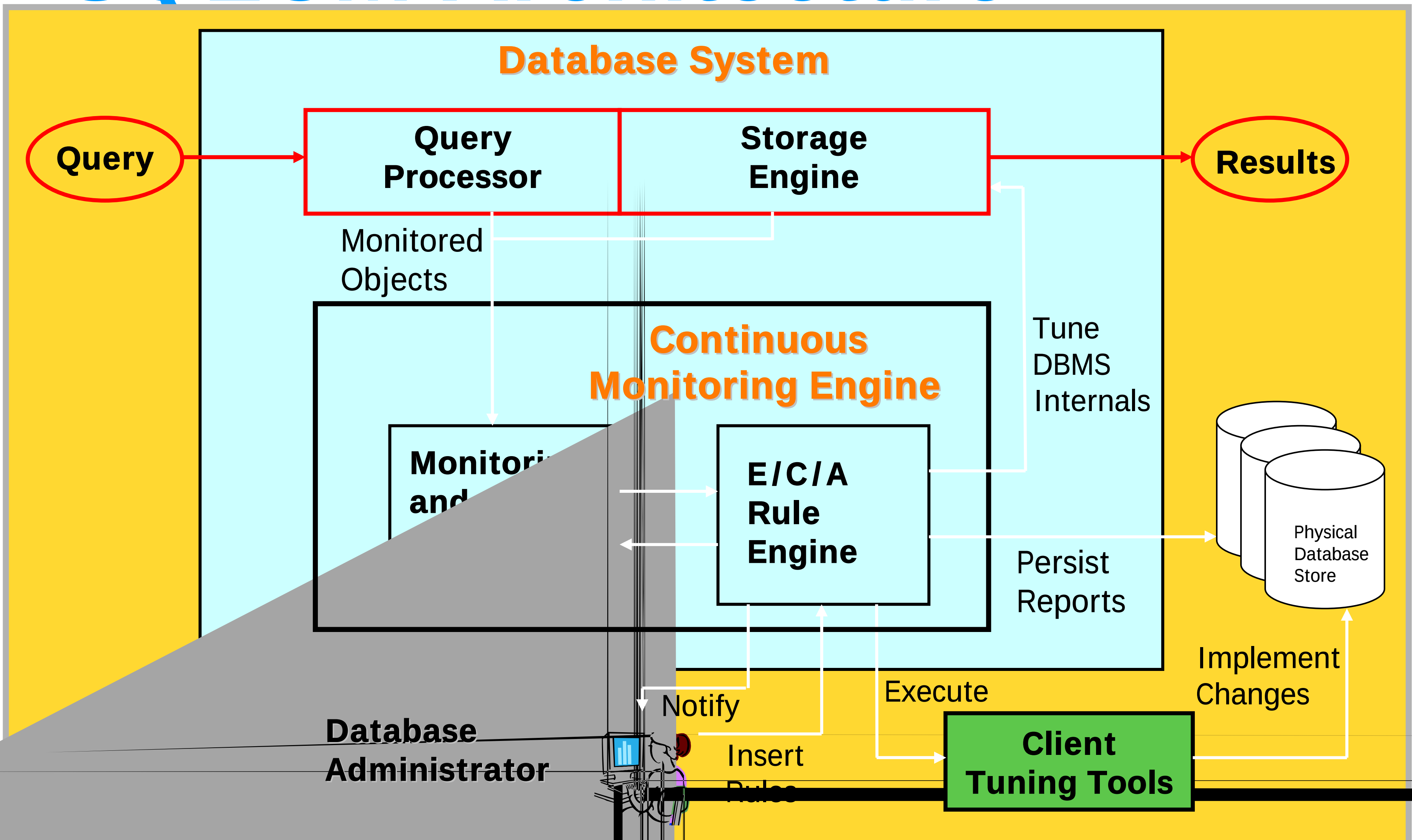
How to evaluate a tuning solution

- Clarity for target of tuning
- Input parameters for tuning
  - Take into account their degree of precision (e.g., uncertainty in estimation)
  - Right model of workload
- Choice of a paradigm influenced by
  - Immediacy of tuning
  - Criticality of a decision (robustness) vs. optimality

Even before Tuning we need..

- **Monitoring**
  - Only a very tiny part of the state of the server is accessible
  - Increasing awareness (Oracle ADDM Warehouse of system events, SQL Server DMV)
  - A flexible infrastructure for looking at system snapshot and its *aggregation* is useful
- **Diagnostics**
  - Ability to do root cause analysis from the knowledge of the system

SQLCM Architecture



Monitoring Progress of SQL Query Execution

- Today's DBMS provides little feedback to DBA *during query execution*
- Goal: Provide *reliable progress estimator* during query execution for long running queries
  - Accuracy, Fine Granularity, Low Overhead, Monotonicity, *Leverage feedback from execution*
- See papers in SIGMOD 2004, 2005, ICDE 2006

Diagnostics

- Requires a careful model of the system
  - Distinguish normal from unusual
  - Analyze events as well as phases of execution over a time interval (Dias et.al. CIDR 2005)
- Decision trees are used as a representation
  - I/O bottleneck split into disk load imbalance, too many seeks, poor cache hit rate, insufficient bandwidth

Principles for Self Tuning

- Complex problems have simple, easy to understand <u>wrong</u> answers
- "Observe-Predict-React" cycle can only be implemented locally
  - Develop self-tuning, adaptive algorithms for individual tuning tasks
  - Need robust models – when and how
- Monitoring/Global knowledge necessary for identification of bottlenecks
- Watch out for too many Tuning parameters

"Learning" != "Magic"

- Conceptually enticing to say that the system will "learn from observation"
- In reality, learning requires
  - Identifying a learning model
  - Several thresholds
  - Essentially, "fits" the parameters given observation
- Learning could be a tool but not a shortcut for thinking

Rethinking Systems: Wishful Thinking?

- VLDB 2000 Vision paper (Chaudhuri and Weikum 2000)
- Enforce Layered approach and Strong limits on interaction (narrow APIs)
  - Package as components of modest complexity
  - Encapsulation must be equipped with self-tuning
- Featurism can be a curse
  - Don’t abuse extensibility - Eliminate 2nd order optimization

Final Words

- **Self-Tuning servers crucial for bounding cost**
  - Policy based adaptive control "observe-predict-react"
  - Monitoring infrastructure – leverage workload and events
  - What-if analysis
  - Mathematical tools
  - Deep understanding of local systems needed
    - Some limited successes so far
    - Plenty of opportunities/challenges

Literature:

- Gang Luo, Jeffrey F. Naughton, Philip S. Yu: Multi-query SQL Progress Indicators. EDBT 2006
- Gang Luo, Jeffrey F. Naughton, Curt Ellmann, Michael Watzke: Increasing the Accuracy and Coverage of SQL Progress Indicators, ICDE 2006
- Surajit Chaudhuri, Raghav Kaushik, Ravishankar Ramamurthy: When Can We Trust Progress Estimators for SQL Queries? SIGMOD 2005
- Gang Luo, Jeffrey F. Naughton, Curt Ellmann, Michael Watzke: Toward a Progress Indicator for Database Queries. SIGMOD 2004
- Surajit Chaudhuri, Vivek R. Narasayya, Ravishankar Ramamurthy: Estimating Progress of Long Running SQL Queries. SIGMOD 2004
- Dushyanth Narayanan, Eno Thereska, Anastassia Ailamaki. Continuous resource monitoring for self-predicting DBMS, MASCOTS 2005
- Surajit Chaudhuri, Christian König, Vivek Narasayya: SQLCM: A Continuous Monitoring Framework for Relational Database Engines. ICDE 2004
- Ning Jiang, Roy Villafane, Kien A. Hua, Abhijit Sawant, Kiran Prabhakara: ADMiRe: An Algebraic Data Mining Approach to System Performance Analysis. IEEE Trans. Knowl. Data Eng. 17(7), 2005

- IEEE CS Data Engineering Workgroup on Self-Managing Database Systems, http://db.uwaterloo.ca/tcde-smdb/

Call for Papers

International Workshop on Self-Managing Database Systems (SMDB 2007)

on April 16, 2007, in Istanbul, Turkey
in conjunction with ICDE 2007

Workshop chair: Guy Lohman
Submission deadline: November 20, 2006

for more details see
http://db.uwaterloo.ca/tcde-smdb/SMDB2007_CFP.html