JUXT

HÅKAN RÅBERG'S

HÅKAN RÅBERG'S

LIGHT AND ADAPTIVE INDEXING

FOR IMMUTABLE DATABASES

JUXT

A JUXT PRODUCTION

1980

1980

The Fifth Workshop
on Computer Architecture for
Non-Numeric Processing in Pacific Grove.

WHAT IF MASS STORAGE WERE FREE?

George Copeland
Tektronix, Inc.
Beaverton, Oregon 97077

Abstract

This paper investigates how database systems would be designed and used under the limiting-case assumption that mass storage is free. It is argued that free mass storage would free database systems from the limitations and problems caused by conventional deletion techniques. A non-deletion strategy would significantly simplify database systems and their operation, as well as increase their functionality and availability. Consideration of this limiting case helps shed light on a more realistic argument: if the cost of mass storage were low enough, then deletion would become undesirable.

It is also argued that the often [illegible] in archival and retrieval of older data can be minimized if a single technology were available

have been required to place a greater emphasis on human costs than on hardware costs.

This paper considers the limiting-case assumption that mass storage is free. Its purpose is to examine some of the implications that this assumption would have concerning the design of future database systems.

Section 2 argues that this free-storage assumption leads to the elimination of deletion in database systems. A non-deletion strategy is suggested using timestamps that would allow significant simplifications in database systems and their operation, as well as a significant increase in application functionality and data integrity. Also, the non-deletion strategy eliminates the need to make periodic checkpoint rollouts for recovery from errors and its resulting impact on system availability. Consideration of this limiting case

What if Mass Storage Were Free?

George P. Copeland. 1980.

Section 5 summarizes the arguments and states their conclusions.

2 Deletion considered harmful

This section argues that significant improvements in functionality, integrity, availability, and simplicity can be achieved in database systems if the deletion mechanism is eliminated.

2.2.1 The importance of access to past states

In human memory, no deletion mechanism exists (Underwood 1969, Nielsen 1958). Although human memory exhibits a decay characteristic, people do not delete. The deletion concept was invented to reuse expensive computer storage.

In the real world of everyday life, people commonly use knowledge of past information to make decisions that control their individual lives, their governments, their businesses, and other organizations. For example, it is quite common to make comparisons of current data with previous periods for trend analysis. Auditing is commonly used to insure the internal control of almost every organization. Also, people are often interested in

What if Mass Storage Were Free?

George P. Copeland. 1980.

JUXT

1986

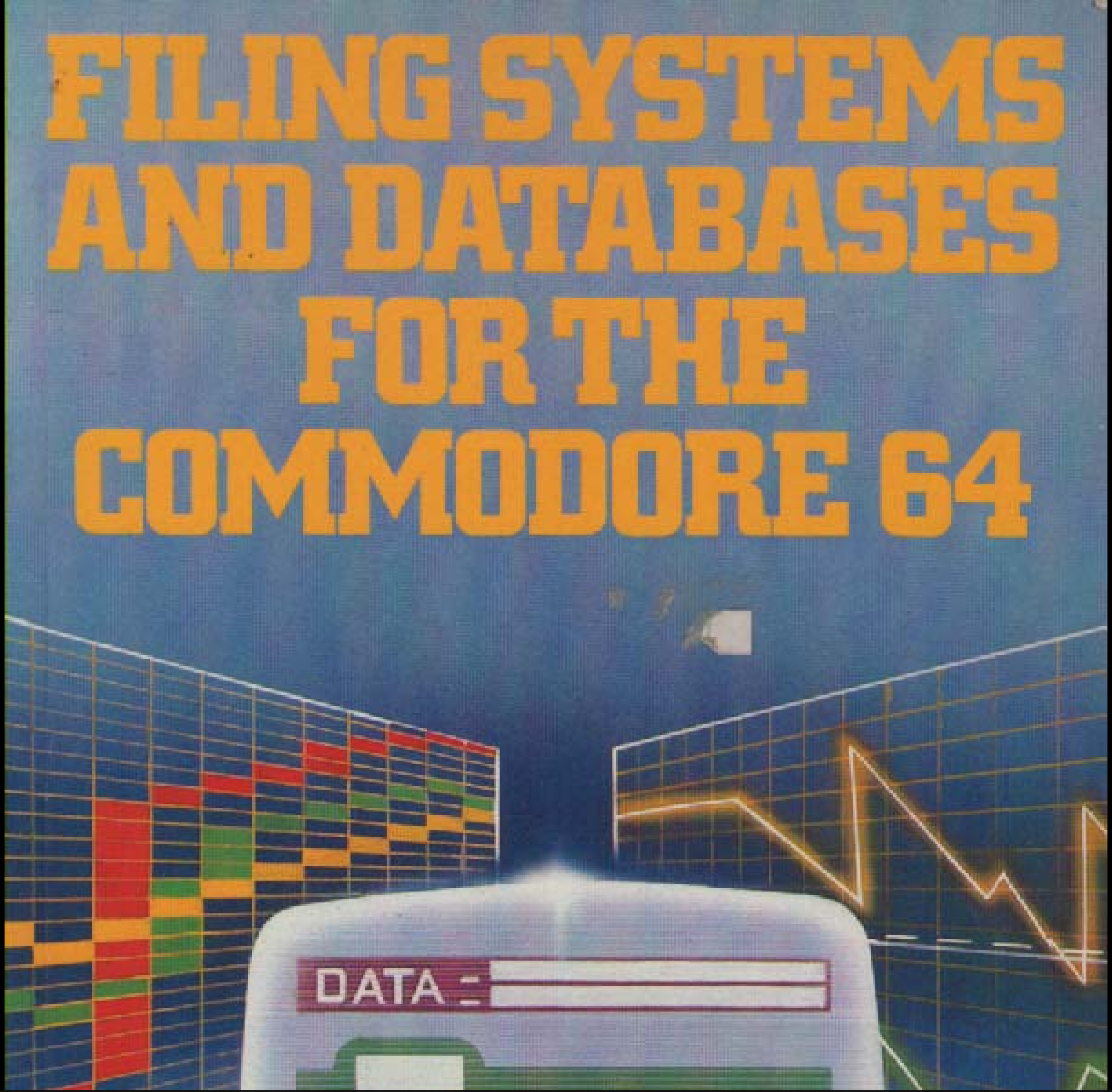
1986

The Vision 86

Radio & TV Fair in Stockholm.

The Vision 86 Radio & TV Fair.

Stockholm metro system advert. 1986.



JUXT

Filing Systems and Databases for the Commodore 64.

A P Stephenson and D J Stephenson. 1985.

With Maxell's diskettes you won't lose memory.

Datormagazin advert. 1989.

Cheaper diskettes you will hardly find anywhere else!

Datormagazin advert. 1989.

2022

2022

NOW

TOWARDS IMMUTABILITY

We are building one called XTDB.

This talk is about indexing.

The ideas are not tied to our architecture.

References and links to papers at end.

Analytics

JUXT

Big Data and OLAP.

Data Lake and Table Formats.

Separation of Storage and Compute.

HTAP. Hybrid Transactional/Analytical Processing.

Separation of Storage and Compute

JUXT

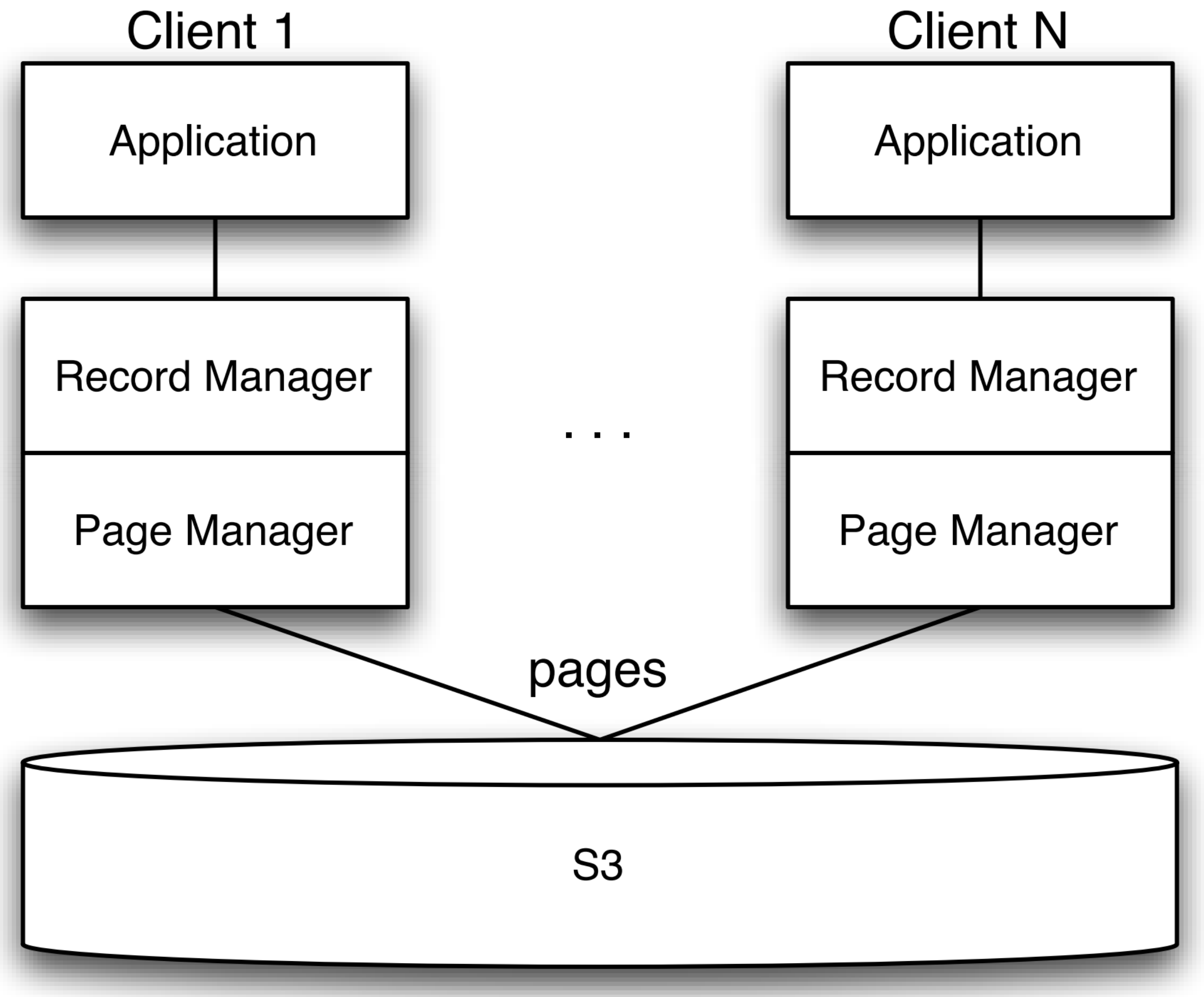
CPU Cache.

RAM.

SSD.

Distributed Cache.

Object Store.



JUXT

Building a Database on S3.

Matthias Brantner, Daniela Florescu, David A. Graf, Donald Kossmann, and Tim Kraska. 2008.

A Decomposition Storage Model

JUXT

Copeland and Khoshafian in 1985.

Today known as column stores.

Embraces sequential array access.

Our system is an HTAP column store.

database system The purpose of this report is not to claim that decomposition is better Instead, we claim that the consensus opinion is not well founded and that neither is clearly better until a closer analysis is made along the many dimensions of a database system The purpose of this report is to move further in both scope and depth toward such an analysis We examine such dimensions as simplicity, generality, storage requirements, update performance and retrieval performance

1 INTRODUCTION

Most database systems use an n-ary storage model (NSM) for a set of records This approach stores data as seen in the conceptual schema Also, various inverted file or cluster indexes might be added for improved access speeds The key concept in the NSM is that all attributes of a conceptual schema record are stored together For example, the conceptual schema relation

R	sur	a1	a2	a3
	s1	v11	v21	v31
	s2	v12	v22	v32
	s3	v13	v23	v33

contains a surrogate for record identity and three attributes per record The NSM would store s1,

a transposed storage model with surrogates included The DSM pairs each attribute value with the surrogate of its conceptual schema record in a binary relation For example, the above relation would be stored as

a1	sur	val
	s1	v11
	s2	v12
	s3	v13

a2	sur	val
	s1	v21
	s2	v22
	s3	v23

a3	sur	val
	s1	v31
	s2	v32
	s3	v33

In addition, the DSM stores two copies of each attribute relation One copy is clustered on the value while the other is clustered on the surrogate These statements apply only to base (i e , extensional) data To support the relational model, intermediate and final results need an n-ary representation If a richer data model than normalized relations is supported, then intermediate and final results need a correspondingly richer representation

This report compares these two storage models based on several criteria Section 2 compares the relative complexity and generality of the two storage models Section 3 compares their storage requirements Section 4 compares their update performance Section 5 compares their retrieval performance Finally, Section 6 provides a summary and suggests some refinements for the DSM

2 SIMPLICITY AND GENERALITY

A Decomposition Storage Model.

George P. Copeland and Setrag Khoshafian. 1985.

database system The purpose of this report is not to claim that decomposition is better Instead, we claim that the consensus opinion is not well founded and that neither is clearly better until a closer analysis is made along the many dimensions of a database system The purpose of this report is to move further in both scope and depth toward such an analysis We examine such dimensions as simplicity, generality, storage requirements, update performance and retrieval performance

1 INTRODUCTION

Most database systems use an n-ary storage model (NSM) for a set of records This approach stores data as seen in the conceptual schema Also, various inverted file or cluster indexes might be added for improved access speeds The key concept in the NSM is that all attributes of a conceptual schema record are stored together For example, the conceptual schema relation

R	sur	a1	a2	a3
	s1	v11	v21	v31
	s2	v12	v22	v32
	s3	v13	v23	v33

Rows

contains a surrogate for record identity and three attributes per record The NSM would store s1,

v1...

a transposed storage model with surrogates included The DSM pairs each attribute value with the surrogate of its conceptual schema record in a binary relation For example, the above relation would be stored as

a1	sur	val
	s1	v11
	s2	v12
	s3	v13

a2	sur	val
	s1	v21
	s2	v22
	s3	v23

a3	sur	val
	s1	v31
	s2	v32
	s3	v33

In addition, the DSM stores two copies of each attribute relation One copy is clustered on the value while the other is clustered on the surrogate These statements apply only to base (i e , extensional) data To support the relational model, intermediate and final results need an n-ary representation If a richer data model than normalized relations is supported, then intermediate and final results need a correspondingly richer representation

This report compares these two storage models based on several criteria Section 2 compares the relative complexity and generality of the two storage models Section 3 compares their storage requirements Section 4 compares their update performance Section 5 compares their retrieval performance Finally, Section 6 provides a summary and suggests some refinements for the DSM

2 SIMPLICITY AND GENERALITY

A Decomposition Storage Model.

George P. Copeland and Setrag Khoshafian. 1985.

database system The purpose of this report is not to claim that decomposition is better Instead, we claim that the consensus opinion is not well founded and that neither is clearly better until a closer analysis is made along the many dimensions of a database system The purpose of this report is to move further in both scope and depth toward such an analysis We examine such dimensions as simplicity, generality, storage requirements, update performance and retrieval performance

1 INTRODUCTION

Most database systems use an n-ary storage model (NSM) for a set of records This approach stores data as seen in the conceptual schema Also, various inverted file or cluster indexes might be added for improved access speeds The key concept in the NSM is that all attributes of a conceptual schema record are stored together For example, the conceptual schema relation

R	sur	a1	a2	a3
	s1	v11	v21	v31
	s2	v12	v22	v32
	s3	v13	v23	v33

contains a surrogate for record identity and three [redacted]

a transposed storage model with surrogates included The DSM pairs each attribute value with the surrogate of its conceptual schema record in a binary relation For example, the above relation would be stored as

Columns

a1	sur	val
	s1	v11
	s2	v12
	s3	v13

a2	sur	val
	s1	v21
	s2	v22
	s3	v23

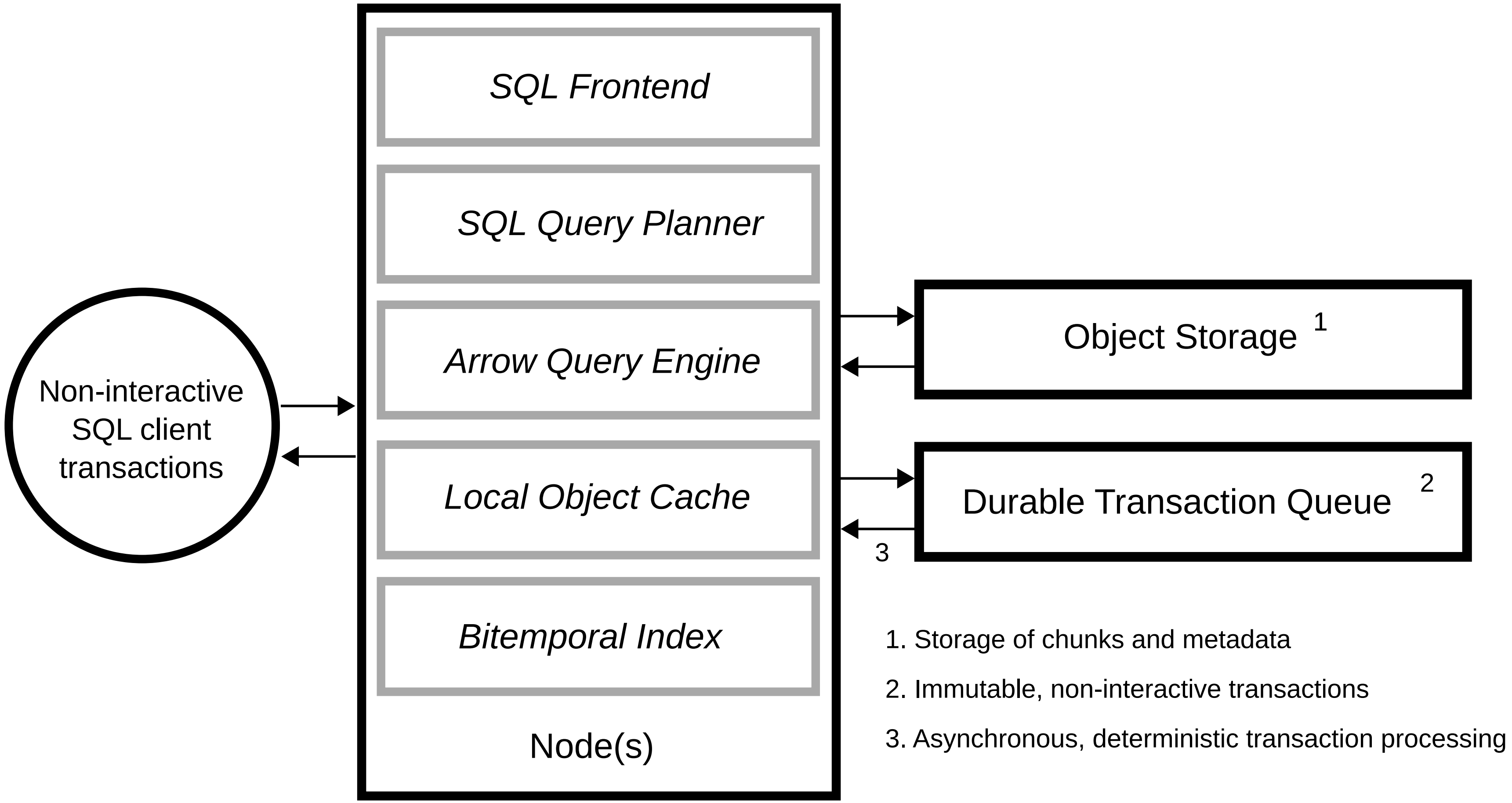
a3	sur	val
	s1	v31
	s2	v32
	s3	v33

In addition, the DSM stores two copies of each attribute relation One copy is clustered on the value while the other is clustered on the surrogate These statements apply only to base (i e , extensional) data To support the relational model, intermediate and final results need an n-ary representation If a richer data model than normalized relations is supported, then intermediate and final results need a correspondingly richer representation

This report compares these two storage models based on several criteria Section 2 compares the relative complexity and generality of the two storage models Section 3 compares their storage requirements Section 4 compares their update performance Section 5 compares their retrieval performance Finally, Section 6 provides a summary [redacted]

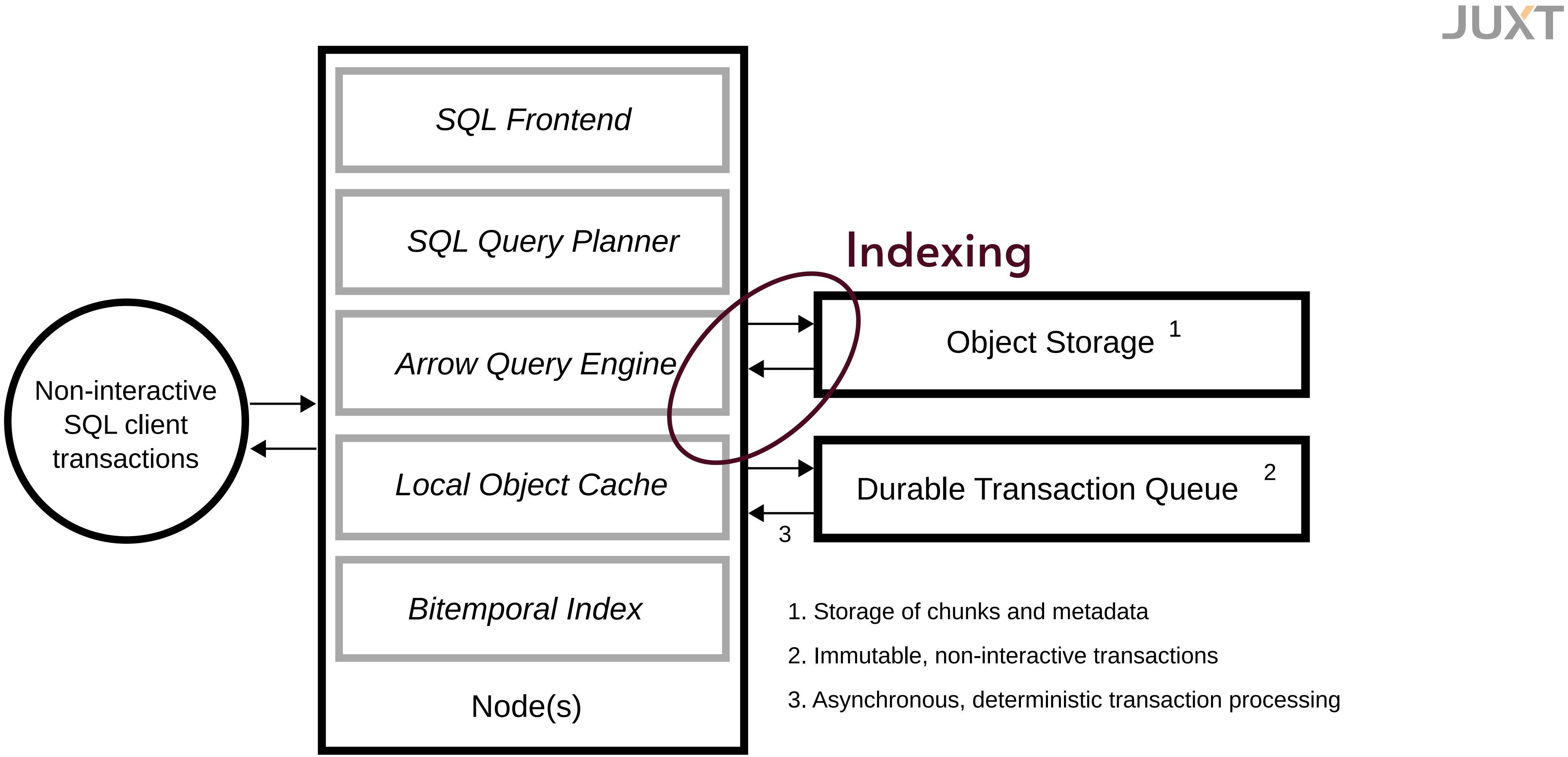
A Decomposition Storage Model.

George P. Copeland and Setrag Khoshafian. 1985.



XTDB Core2 Architecture.

JUXT. 2020-2022.



1. Storage of chunks and metadata
2. Immutable, non-interactive transactions
3. Asynchronous, deterministic transaction processing

XTDB Core2 Architecture.

JUXT. 2020-2022.

THE PROBLEM

The Latency Problem

JUXT

Shared disk strikes back.

Enables Copeland's vision.

Data needs to be found and navigated.

Introduces latency and request costs.

The Latency Solution

Indexing and caching.

Light immutable indexes in storage.

Adaptive indexing for compute workload.

Academia for inspiration.

ACADEMIA

Finding and Reading Papers

Skim abstract, then look for interesting pictures.

Resist temptation to understand each paper.

Find a pragmatic approach via references.

Invalidate your ideas, avoid costly experiments.

arxiv.org, semanticscholar.org, paperswelove.org.

Self-Driving Databases, CMU.

Adaptive Indexing, Harvard.

Instance-Optimized Data Systems, MIT.

Takes DBAs out of the loop.

JUXT

INDEX SELECTION IN A SELF-ADAPTIVE DATA BASE MANAGEMENT SYSTEM

Michael Hammer
Arvola Chan

Laboratory for Computer Science, MIT,
Cambridge, Massachusetts, 02139.

We address the problem of automatically adjusting the physical organization of a data base to optimize its performance as its access requirements change. We describe the principles of the automatic index selection facility of a prototype self-adaptive data base management system that is currently under development. The importance of accurate usage model acquisition and data characteristics estimation is stressed. The statistics gathering mechanisms that are being incorporated into our prototype system are discussed. Exponential smoothing techniques are used for averaging statistics observed over different periods of time in order to predict future characteristics. An heuristic algorithm for selecting indices to match projected access requirements is presented. The cost model on which the decision procedure is based is flexible enough to incorporate the overhead costs of index creation, index storage and application program recompilation.

Index Selection in a Self-Adaptive Data Base Management System.

Michael Hammer and Arvola Chan. 1976.

LEARNED INDEXING

RadixSpline

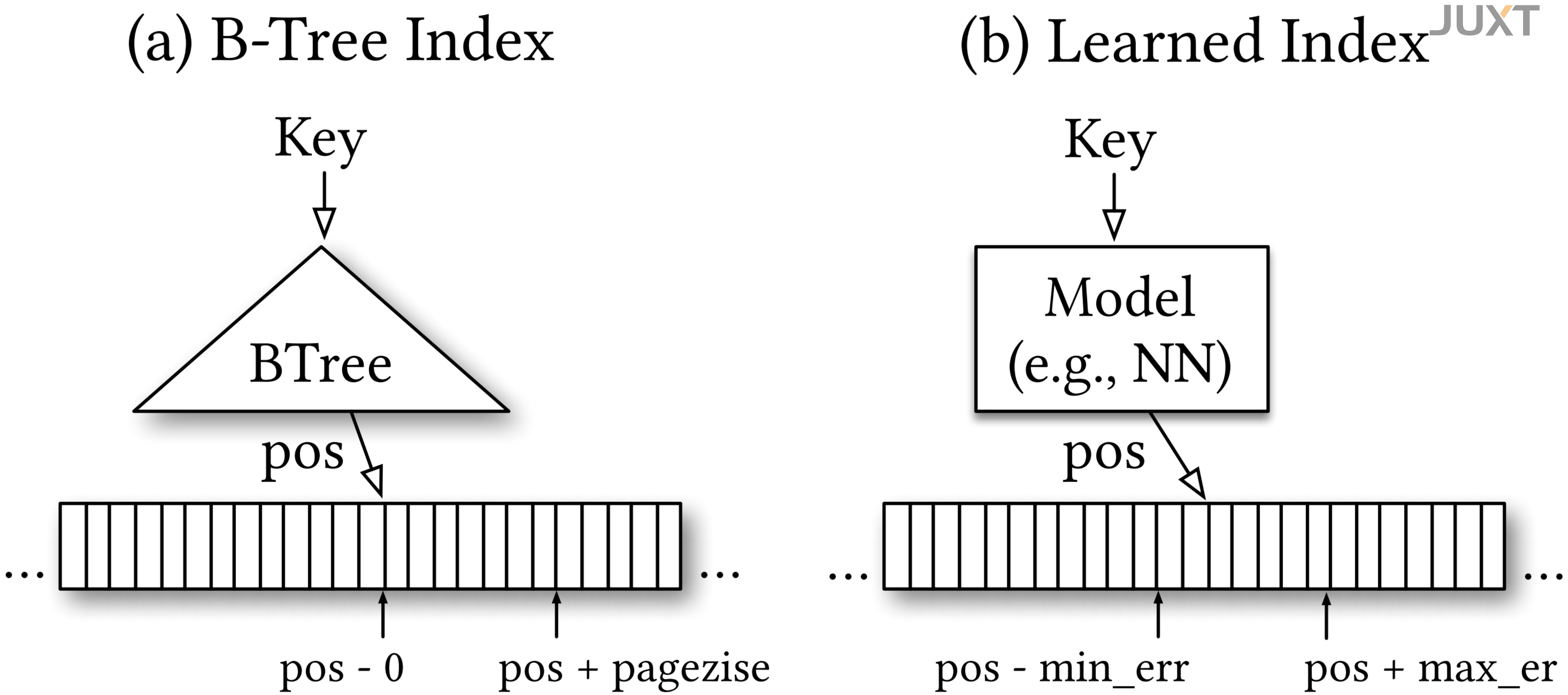
JUXT

A Single-Pass Learned Index. Kipf et al. 2020.

Estimates CDF. Cumulative Distribution Function.

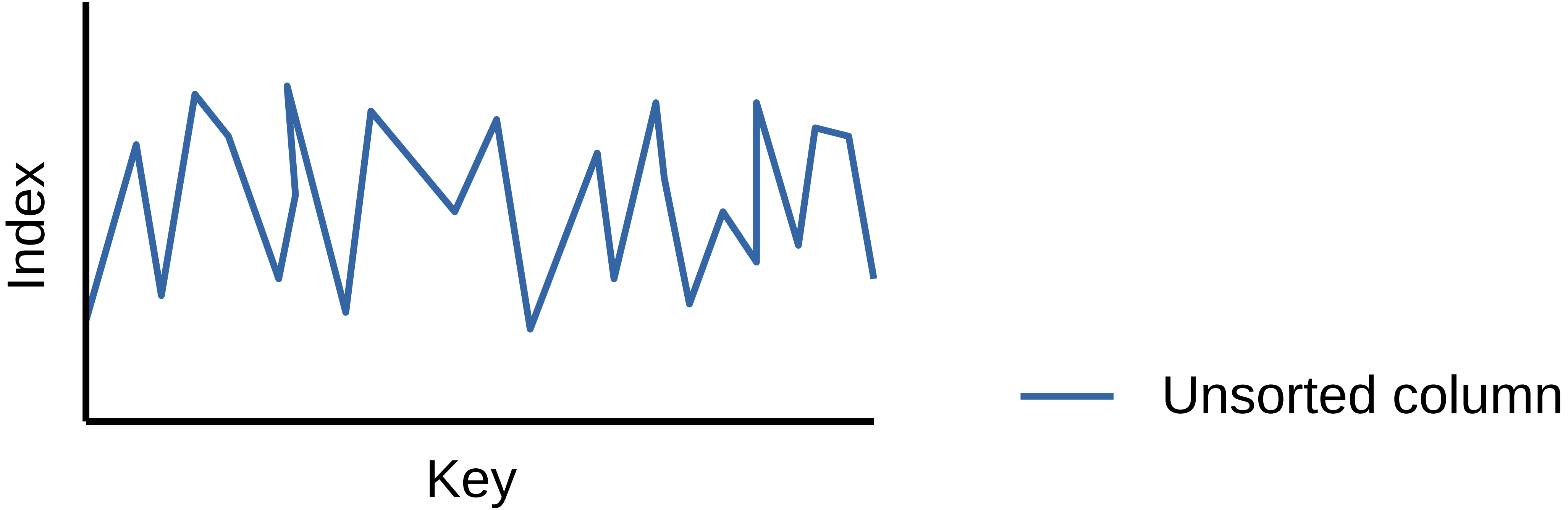
Adds spline points to radix layer.

Model maps from key to position.



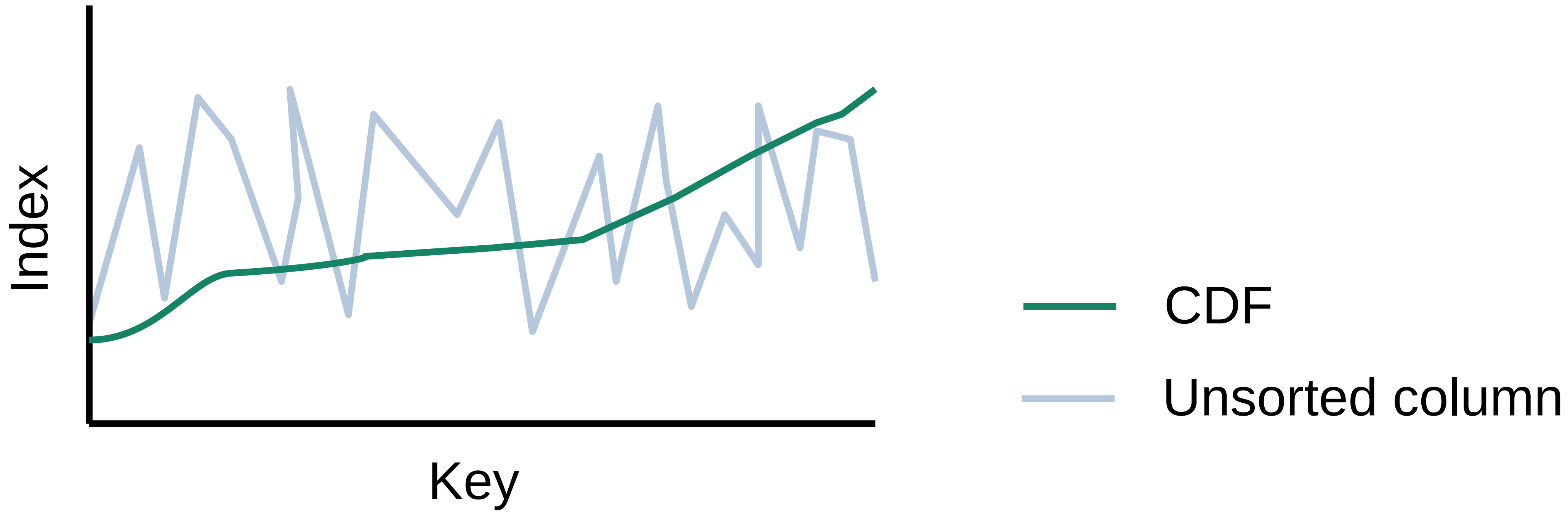
The Case for Learned Index Structures.

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018.



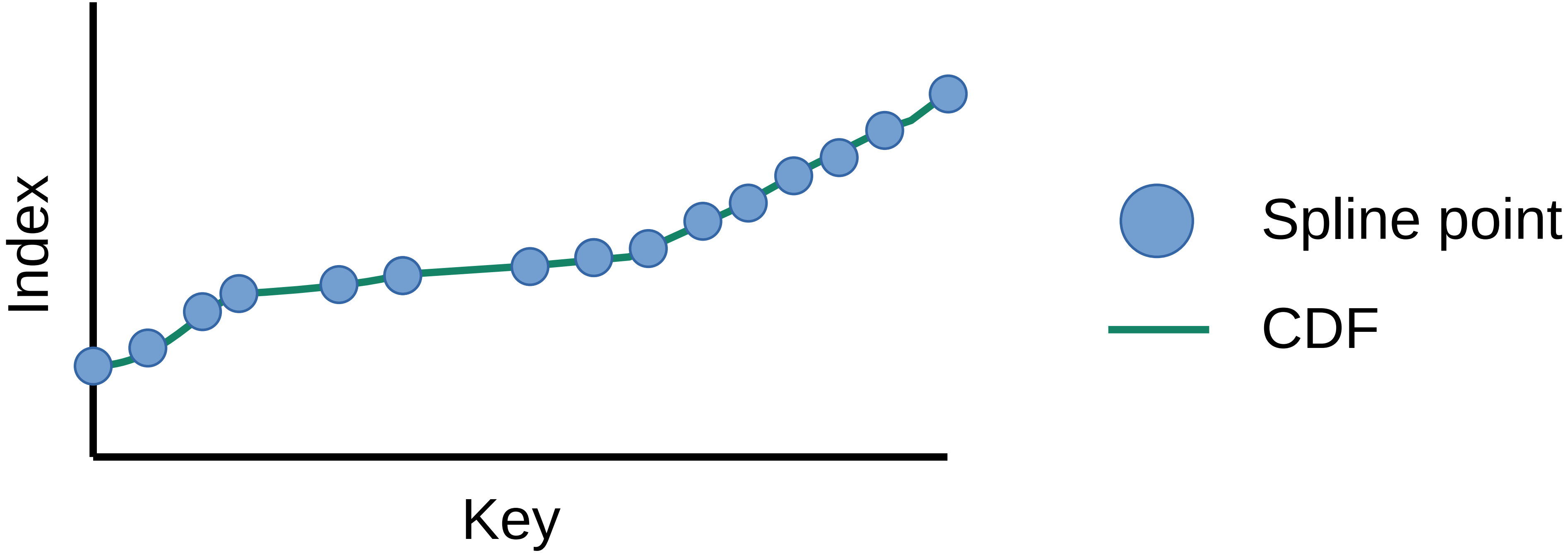
RadixSpline: A Single-Pass Learned Index.

Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2020.



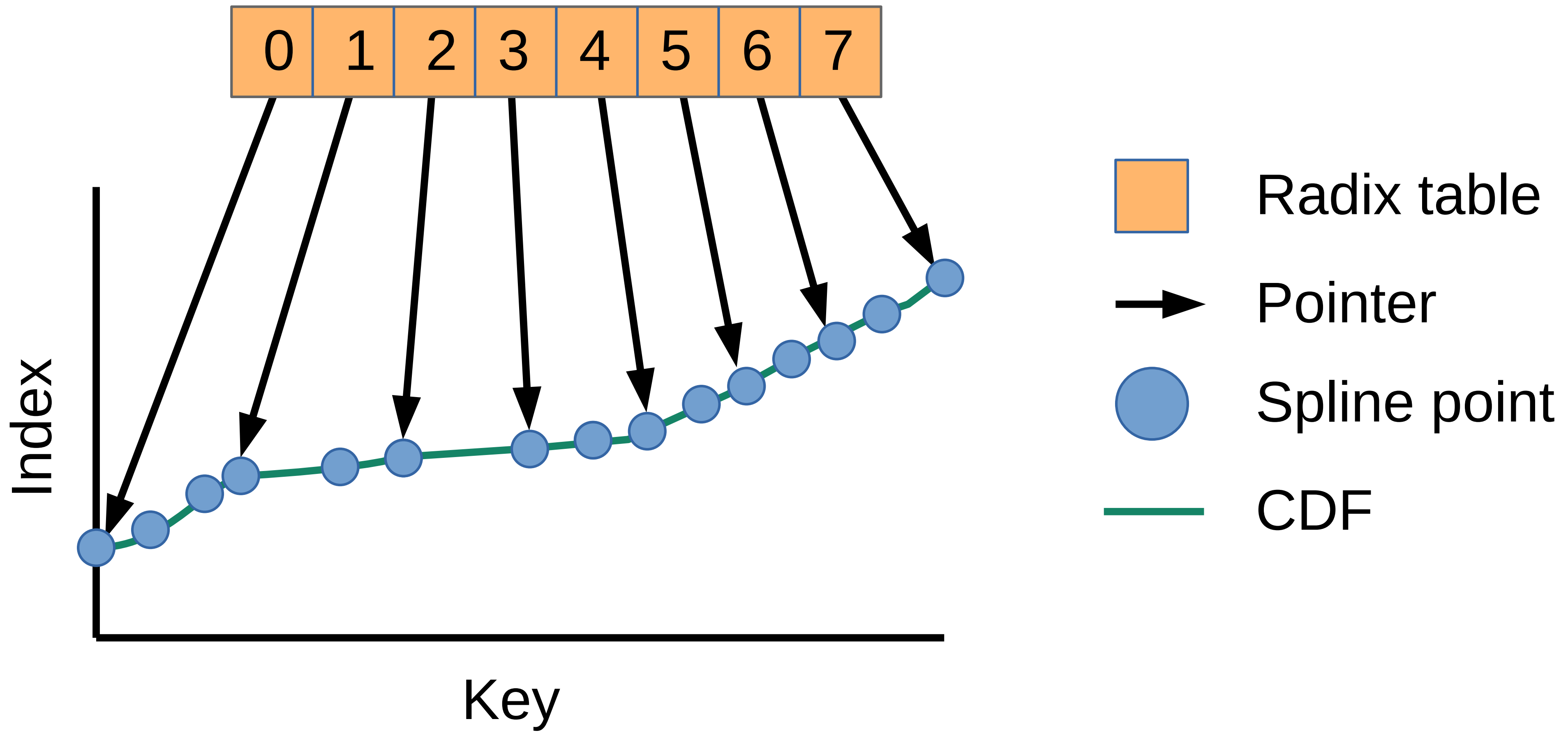
RadixSpline: A Single-Pass Learned Index.

Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2020.



RadixSpline: A Single-Pass Learned Index.

Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2020.

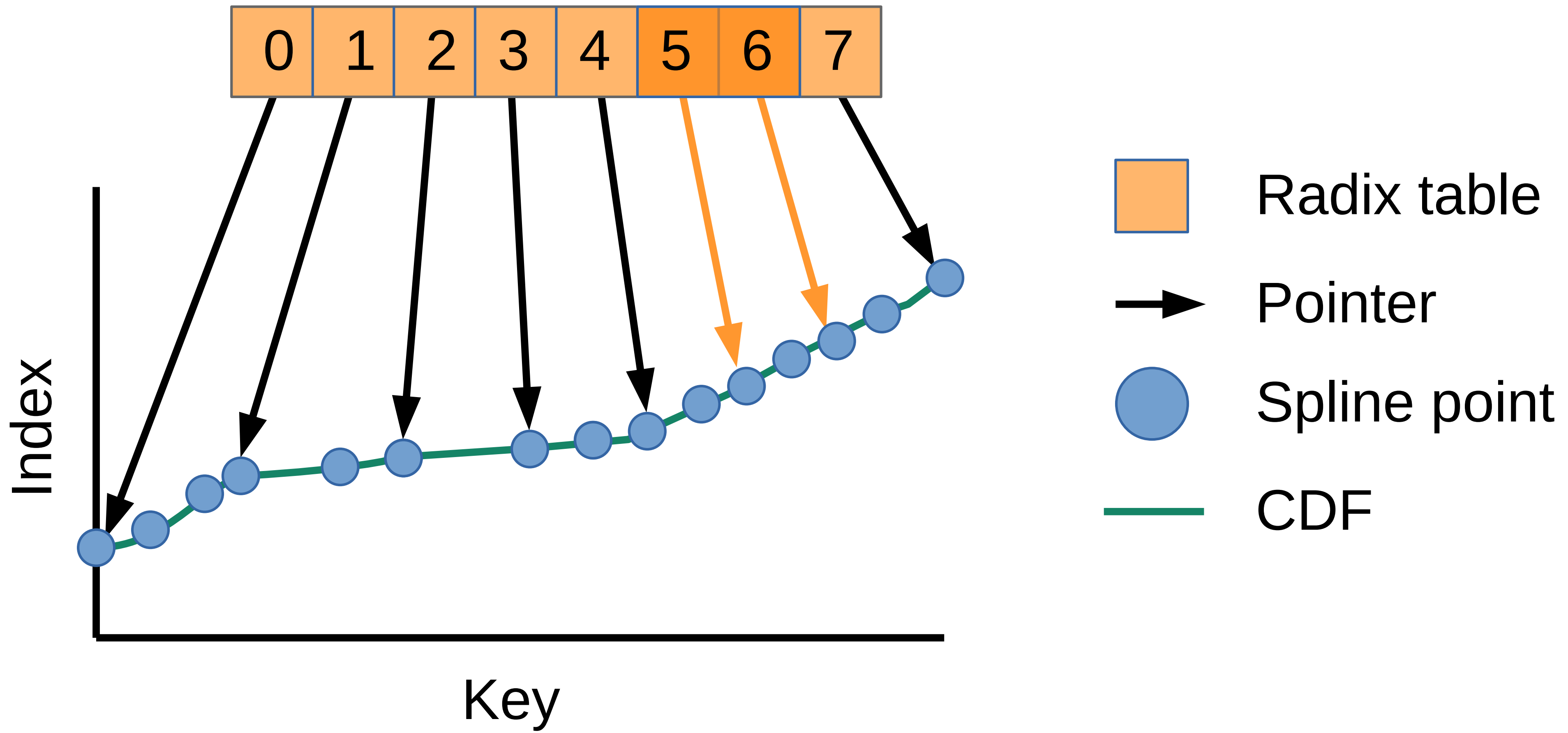


RadixSpline: A Single-Pass Learned Index.

Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2020.

Lookup Key: 47183_{10} $1011\ 1000\ 0100\ 1111_2$

JUXT



RadixSpline: A Single-Pass Learned Index.

Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2020.

GreedySplineCorridor

Input: a spline S, $|S| = n$ and an error corridor size ϵ

Output: a spline connecting $S[1], S[n]$ through the corridor

$B = S[1], R =< B >$ // $S[1]$ is the first base point

$U = S[2] + \epsilon, L = S[2] - \epsilon$ // error corridor bounds

for $i = 3$ **to** n

  $C = S[i]$

  if $\overline{BC}$ is left of $\overline{BU}$ or right of $\overline{BL}$

    $B = S[i-1], R = R \circ < B >$

    $U = C + \epsilon, L = C - \epsilon$

  else

    $U' = C + \epsilon, L' = C - \epsilon$

    if $\overline{BU}$ is left of $\overline{BU'}$

      $U = U'$

    if $\overline{BL}$ is right of $\overline{BL'}$

      $L = L'$

$R = R \circ < S[n] >$

return R

Fig. 1. Greedy Spline Approximation with a Given Error Corridor

Smooth Interpolating Histograms with Error Guarantees.

Thomas Neumann and Sebastian Michel. 2008.

GreedySplineCorridor

Input: a spline S, $|S| = n$ and an error corridor size ϵ

Output: a spline connecting $S[1], S[n]$ through the corridor

$B = S[1], R =< B >$ // $S[1]$ is the first base point

$U = S[2] + \epsilon, L = S[2] - \epsilon$ // error corridor bounds

for $i = 3$ **to** n

  $C = S[i]$

  if $\overline{BC}$ is left of $\overline{BU}$ or right of $\overline{BL}$

    $B = S[i-1], R = R \circ < B >$

    $U = C + \epsilon, L = C - \epsilon$

  else

    $U' = C + \epsilon, L' = C - \epsilon$

    if $\overline{BU}$ is left of $\overline{BU'}$

      $U = U'$

    if $\overline{BL}$ is right of $\overline{BL'}$

      $L = L'$

$R = R \circ < S[n] >$

return R

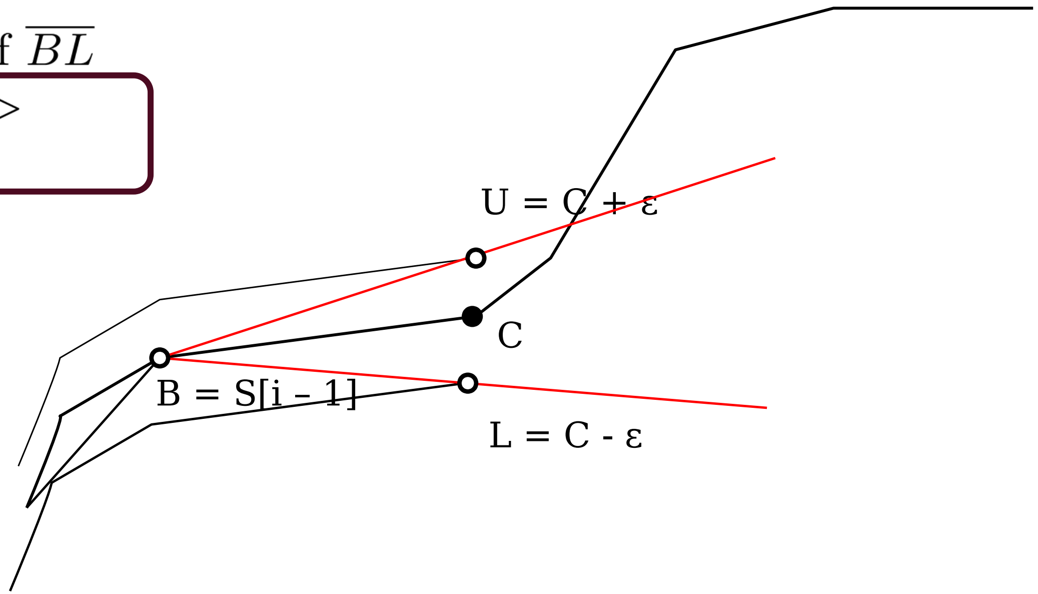


Fig. 1. Greedy Spline Approximation with a Given Error Corridor

Smooth Interpolating Histograms with Error Guarantees.

Thomas Neumann and Sebastian Michel. 2008.

GreedySplineCorridor

Input: a spline S, $|S| = n$ and an error corridor size ϵ

Output: a spline connecting $S[1], S[n]$ through the corridor

$B = S[1], R =< B >$ // $S[1]$ is the first base point

$U = S[2] + \epsilon, L = S[2] - \epsilon$ // error corridor bounds

for $i = 3$ **to** n

    $C = S[i]$

    if $\overline{BC}$ is left of $\overline{BU}$ or right of $\overline{BL}$

        $B = S[i-1], R = R \circ < B >$

        $U = C + \epsilon, L = C - \epsilon$

    else

        $U' = C + \epsilon, L' = C - \epsilon$

        if $\overline{BU}$ is left of $\overline{BU'}$

            $U = U'$

        if $\overline{BL}$ is right of $\overline{BL'}$

            $L = L'$

$R = R \circ < S[n] >$

return R

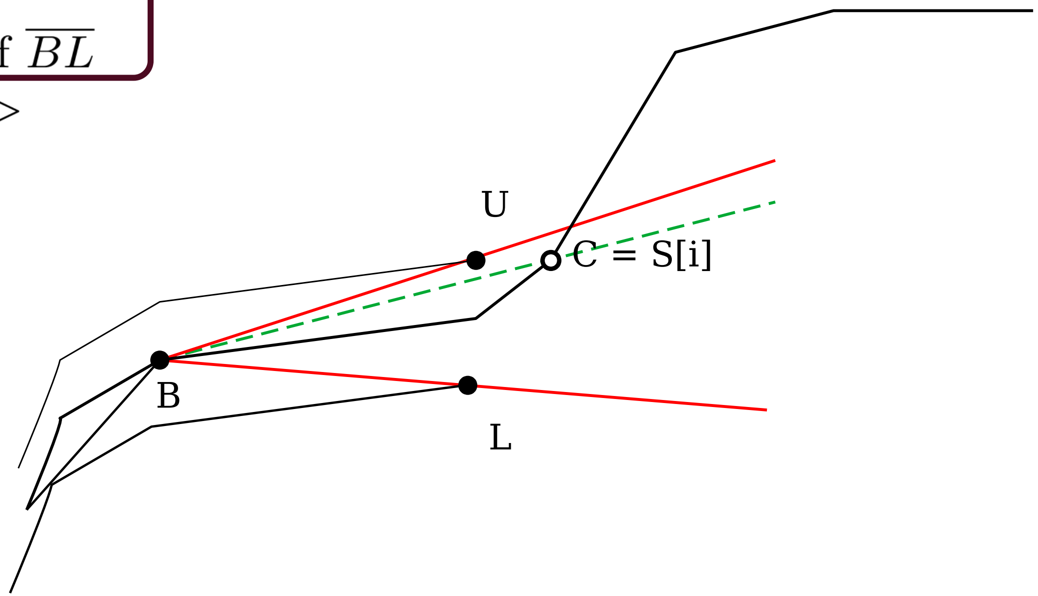


Fig. 1. Greedy Spline Approximation with a Given Error Corridor

Smooth Interpolating Histograms with Error Guarantees.

Thomas Neumann and Sebastian Michel. 2008.

GreedySplineCorridor

Input: a spline S, $|S| = n$ and an error corridor size ϵ

Output: a spline connecting $S[1], S[n]$ through the corridor

$B = S[1], R =< B >$ // $S[1]$ is the first base point

$U = S[2] + \epsilon, L = S[2] - \epsilon$ // error corridor bounds

for $i = 3$ **to** n

  $C = S[i]$

  if $\overline{BC}$ is left of $\overline{BU}$ or right of $\overline{BL}$

    $B = S[i-1], R = R \circ < B >$

    $U = C + \epsilon, L = C - \epsilon$

  else

    $U' = C + \epsilon, L' = C - \epsilon$

    if $\overline{BU}$ is left of $\overline{BU'}$

      $U = U'$

    if $\overline{BL}$ is right of $\overline{BL'}$

      $L = L'$

$R = R \circ < S[n] >$

return R

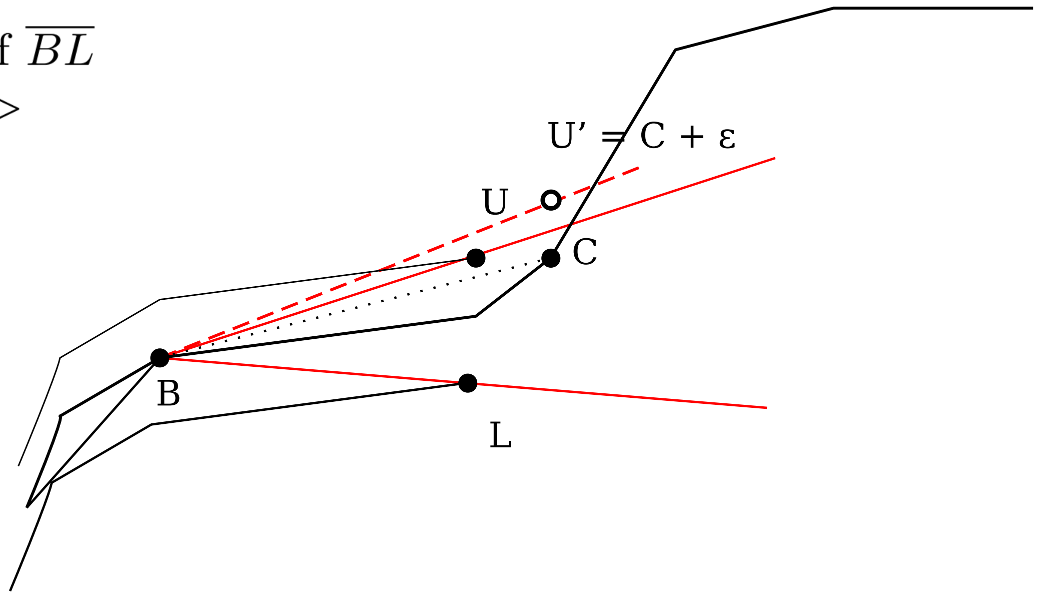


Fig. 1. Greedy Spline Approximation with a Given Error Corridor

Smooth Interpolating Histograms with Error Guarantees.

Thomas Neumann and Sebastian Michel. 2008.

GreedySplineCorridor

Input: a spline S, $|S| = n$ and an error corridor size ϵ

Output: a spline connecting $S[1], S[n]$ through the corridor

$B = S[1], R =< B >$ // $S[1]$ is the first base point

$U = S[2] + \epsilon, L = S[2] - \epsilon$ // error corridor bounds

for $i = 3$ **to** n

  $C = S[i]$

  if $\overline{BC}$ is left of $\overline{BU}$ or right of $\overline{BL}$

    $B = S[i-1], R = R \circ < B >$

    $U = C + \epsilon, L = C - \epsilon$

  else

    $U' = C + \epsilon, L' = C - \epsilon$

    if $\overline{BU}$ is left of $\overline{BU'}$

      $U = U'$

    if $\overline{BL}$ is right of $\overline{BL'}$

      $L = L'$

$R = R \circ < S[n] >$

return R

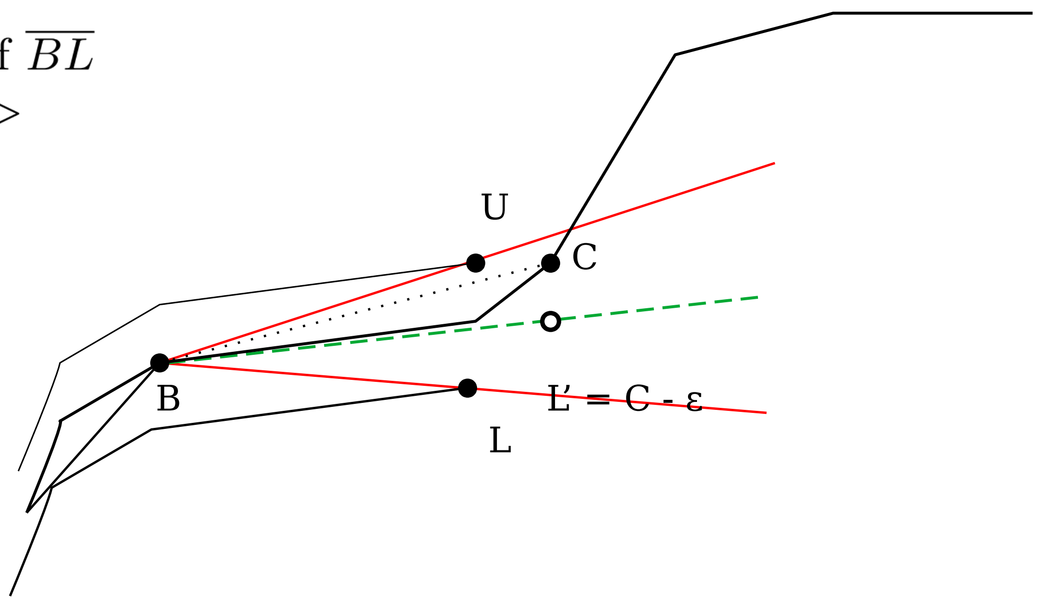


Fig. 1. Greedy Spline Approximation with a Given Error Corridor

Smooth Interpolating Histograms with Error Guarantees.

Thomas Neumann and Sebastian Michel. 2008.

GreedySplineCorridor

Input: a spline S, $|S| = n$ and an error corridor size ϵ

Output: a spline connecting $S[1], S[n]$ through the corridor

$B = S[1], R =< B >$ // $S[1]$ is the first base point

$U = S[2] + \epsilon, L = S[2] - \epsilon$ // error corridor bounds

for $i = 3$ **to** n

  $C = S[i]$

  if $\overline{BC}$ is left of $\overline{BU}$ or right of $\overline{BL}$

    $B = S[i-1], R = R\circ < B >$

    $U = C + \epsilon, L = C - \epsilon$

  else

    $U' = C + \epsilon, L' = C - \epsilon$

    if $\overline{BU}$ is left of $\overline{BU'}$

      $U = U'$

    if $\overline{BL}$ is right of $\overline{BL'}$

      $\boxed{L = L'}$

$R = R\circ < S[n] >$

return R

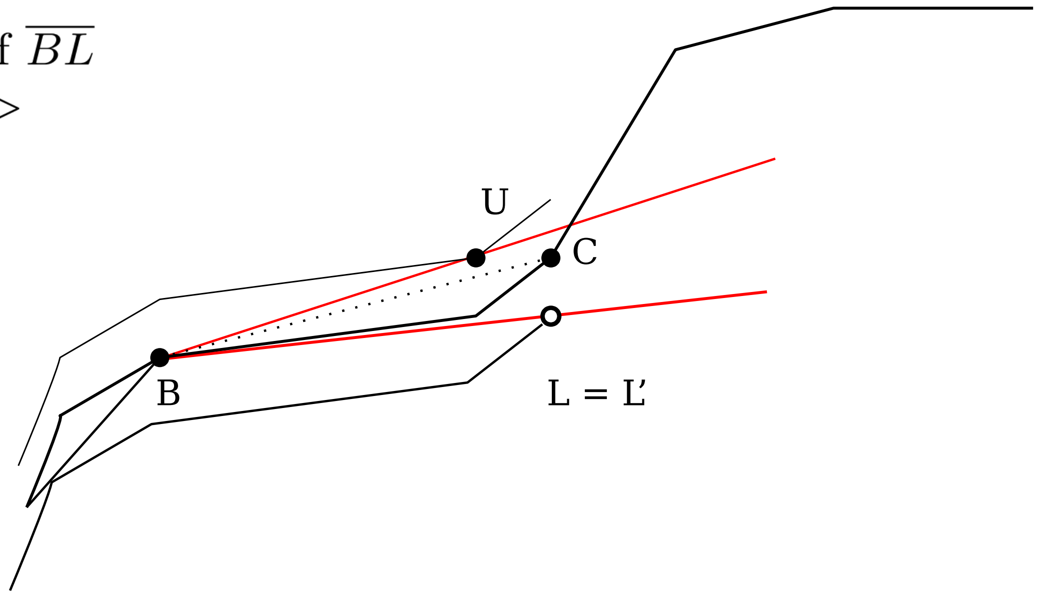


Fig. 1. Greedy Spline Approximation with a Given Error Corridor

Smooth Interpolating Histograms with Error Guarantees.

Thomas Neumann and Sebastian Michel. 2008.

```
GreedySplineCorridor
Input: a spline S, |S| = n and an error corridor size ϵ
Output: a spline connecting S[1], S[n] through the corridor
B = S[1], R =< B > // S[1] is the first base point
U = S[2] + ϵ, L = S[2] − ϵ // error corridor bounds
for i = 3 to n
   C = S[i]
   if BC is left of BU or right of BL
      B = S[i − 1], R = R◦ < B >
      U = C + ϵ, L = C − ϵ
   else
      U′ = C + ϵ, L′ = C − ϵ
      if BU is left of BU′
         U = U′
      if BL is right of BL′
         L = L′
R = R◦ < S[n] >
return R
```

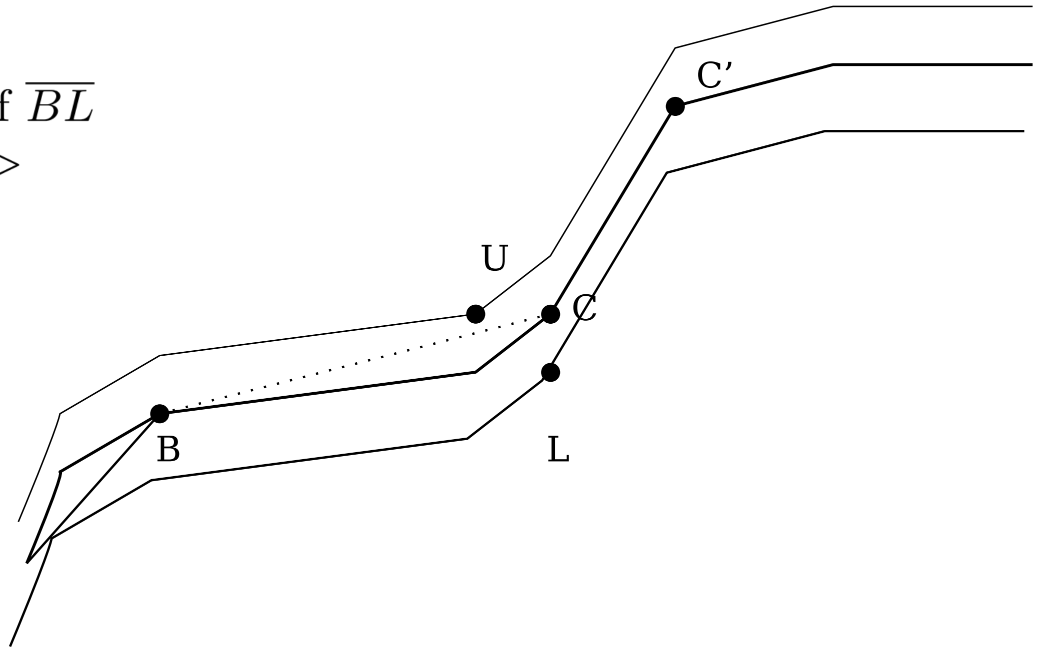
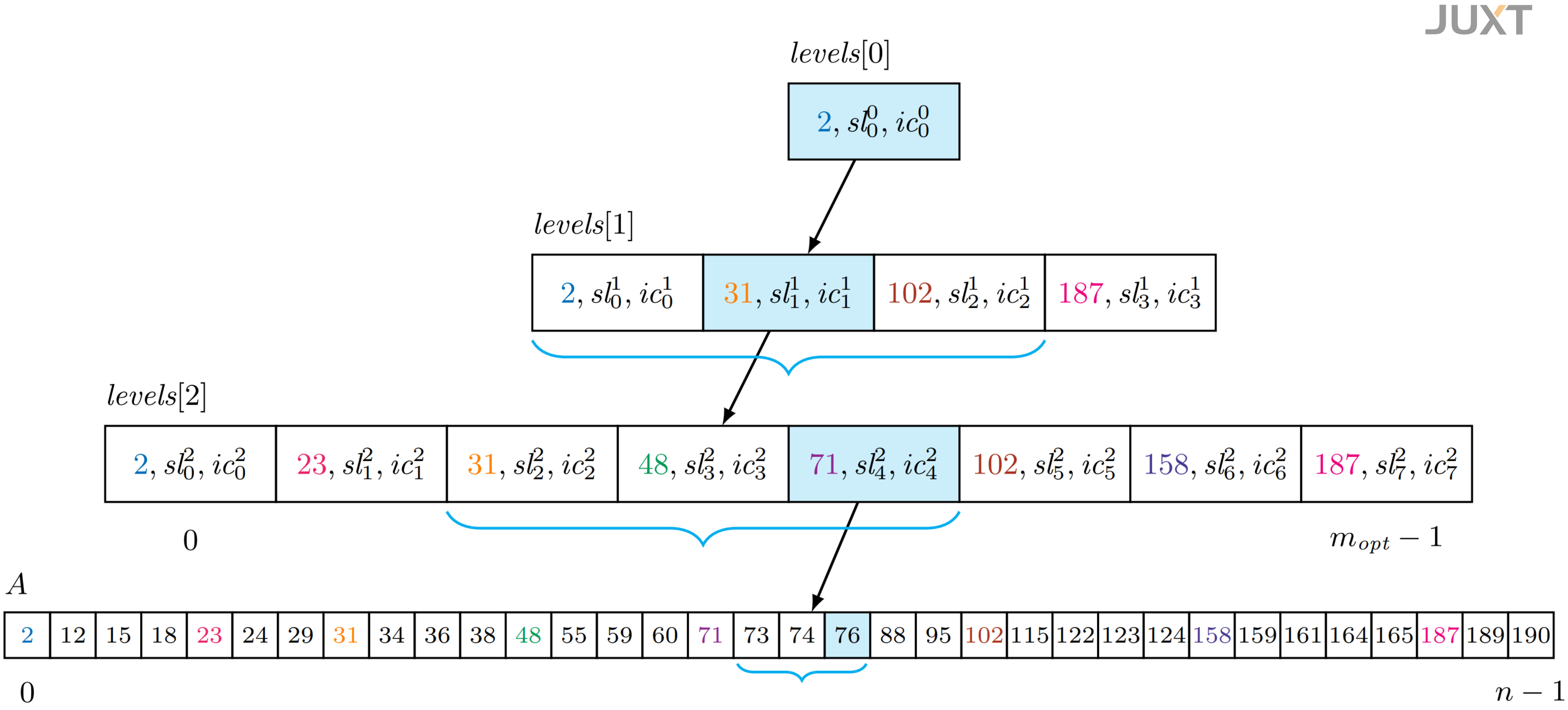


Fig. 1. Greedy Spline Approximation with a Given Error Corridor

Smooth Interpolating Histograms with Error Guarantees.

Thomas Neumann and Sebastian Michel. 2008.



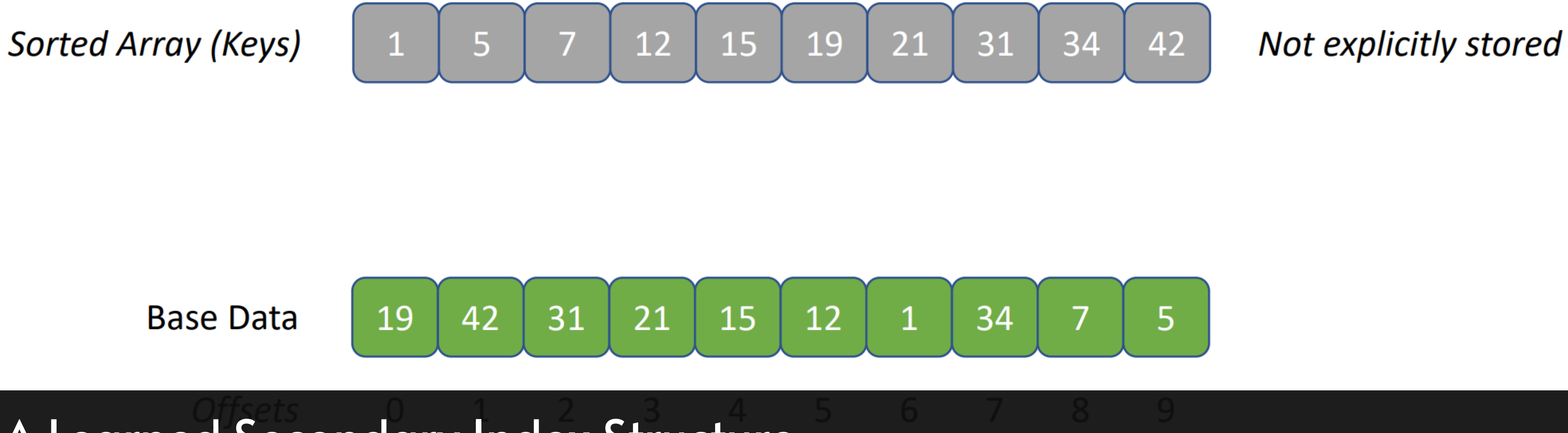
The PGM-index.

Paolo Ferragina and Giorgio Vinciguerra. 2020.

Base Data	19	42	31	21	15	12	1	34	7	5
Offsets	0	1	2	3	4	5	6	7	8	9

LSI: A Learned Secondary Index Structure.

Andreas Kipf, Dominik Horn, Pascal Pfeil, Ryan Marcus, and Tim Kraska. 2022.



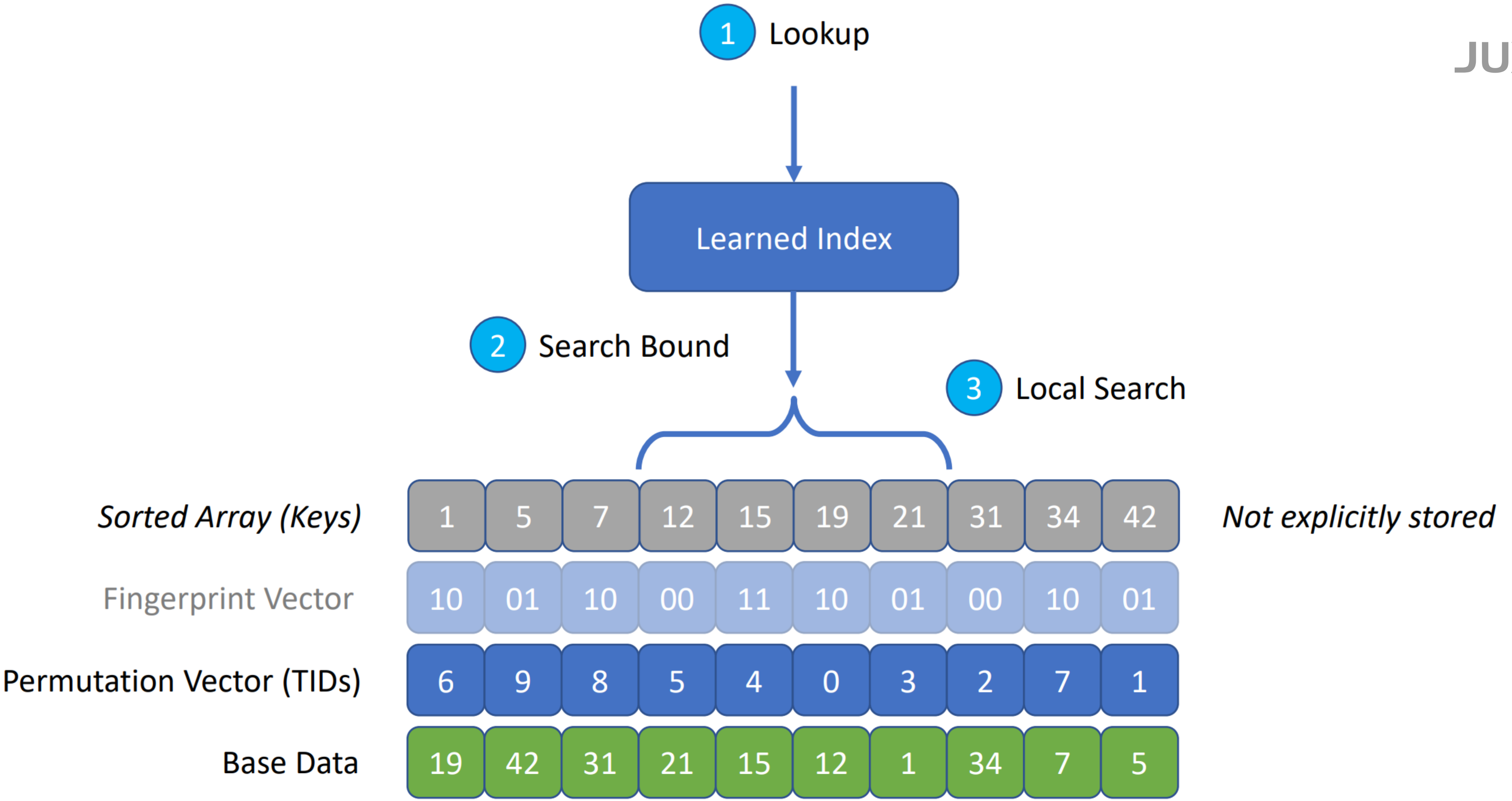
LSI: A Learned Secondary Index Structure.

Andreas Kipf, Dominik Horn, Pascal Pfeil, Ryan Marcus, and Tim Kraska. 2022.

Sorted Array (Keys)	1	5	7	12	15	19	21	31	34	42	*Not explicitly stored*
Permutation Vector (TIDs)	6	9	8	5	4	0	3	2	7	1	
Base Data	19	42	31	21	15	12	1	34	7	5	
Offsets	0	1	2	3	4	5	6	7	8	9	

LSI: A Learned Secondary Index Structure.

Andreas Kipf, Dominik Horn, Pascal Pfeil, Ryan Marcus, and Tim Kraska. 2022.



LSI: A Learned Secondary Index Structure.

Andreas Kipf, Dominik Horn, Pascal Pfeil, Ryan Marcus, and Tim Kraska. 2022.

ADAPTIVE INDEXING

Highly influential paper. Idreos et al. 2007.

Incrementally sorts copy of column.

Requires tracking of piece boundaries.

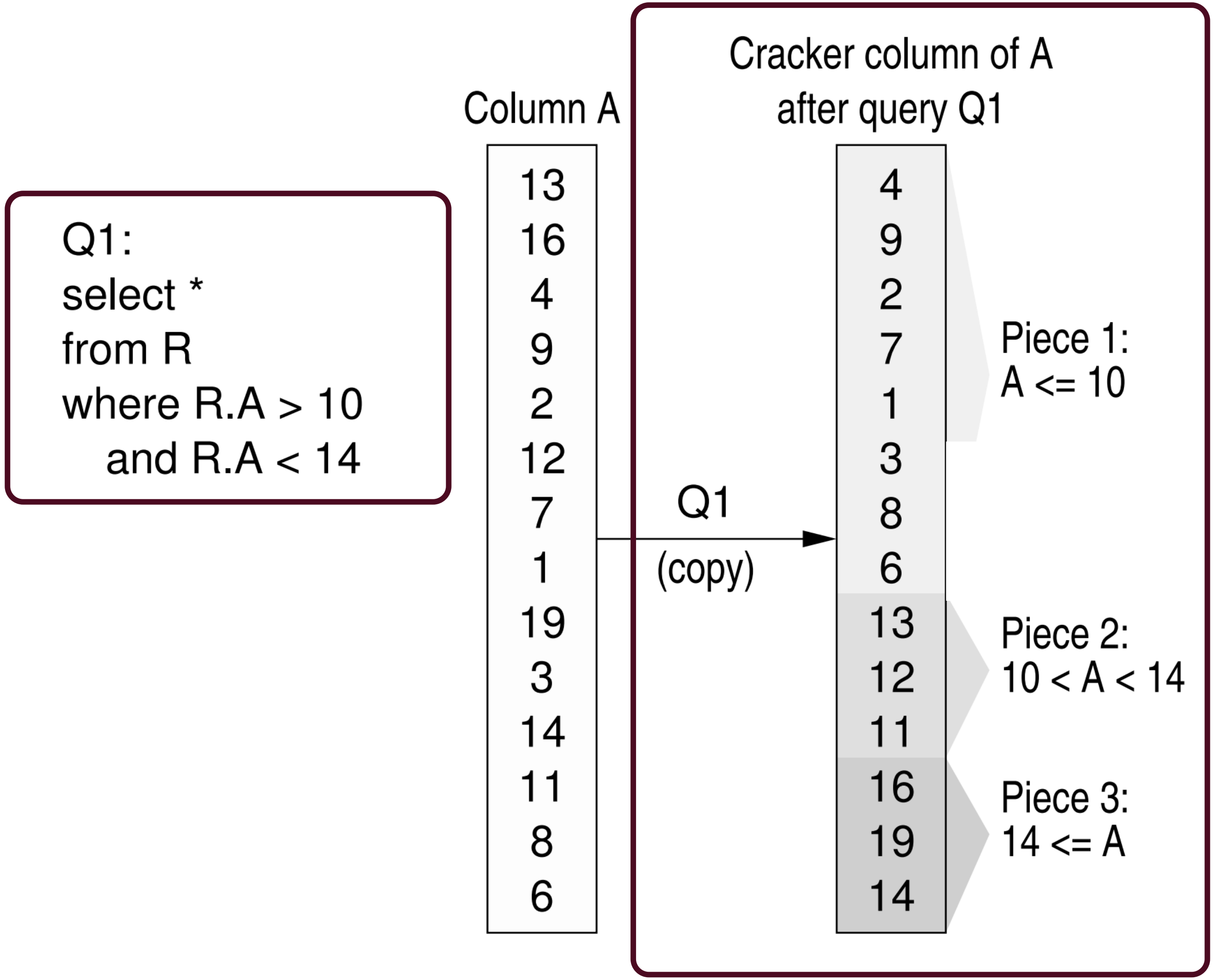
Adapting by query has drawbacks.

Column A

Column A
13
16
4
9
2
12
7
1
19
3
14
11
8
6

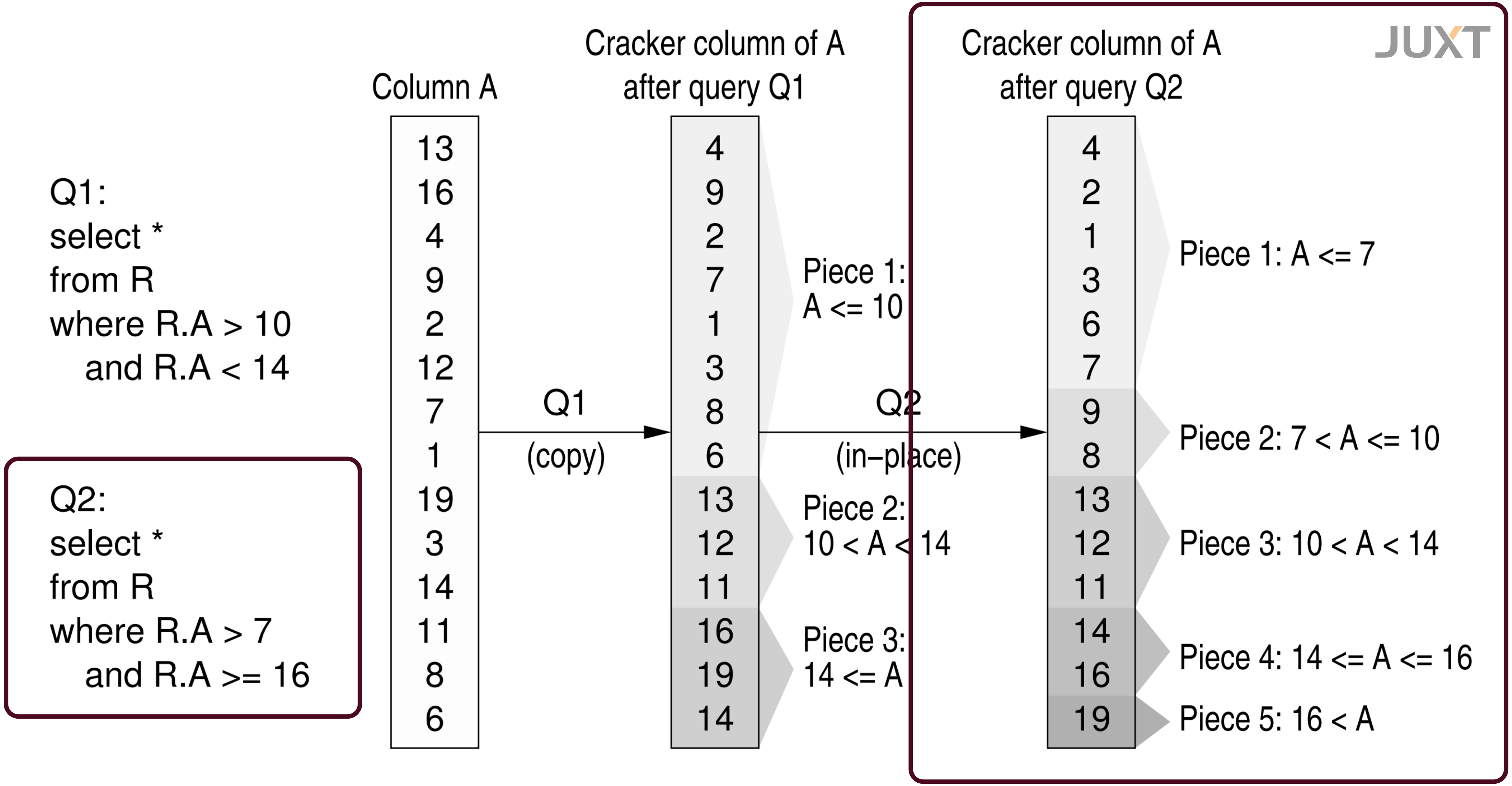
Database Cracking.

Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2007.



Database Cracking.

Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2007.



Database Cracking.

Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2007.

INSTANCE-OPTIMIZED SYSTEMS

Self-Organizing Data Containers

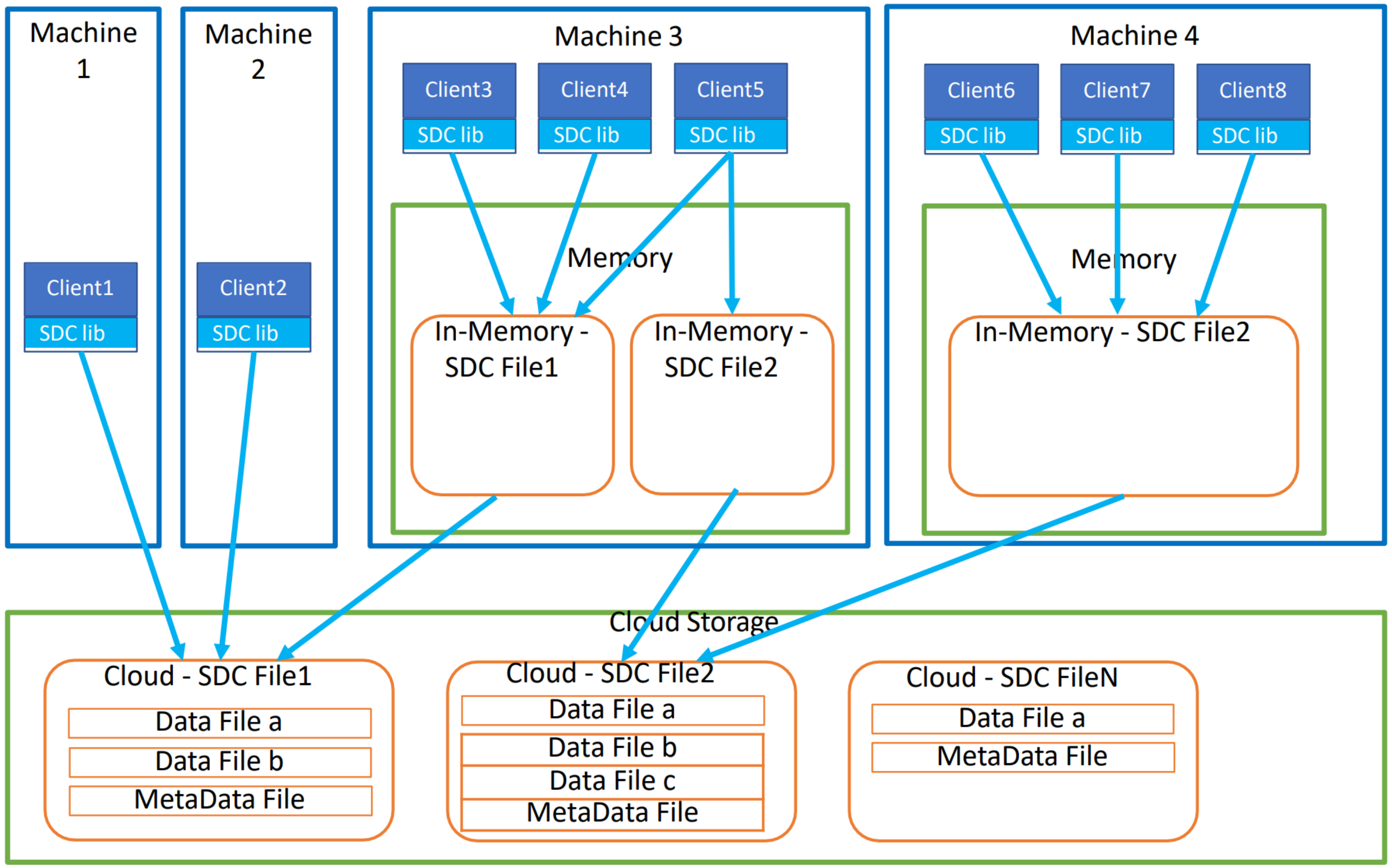
JUXT

Vision paper. Madden et al. 2022.

Separation of Storage and Compute.

Automatically create and modify indexes.

Goal is to minimize object access.



JUXT

Self-Organizing Data Containers.

Samuel Madden, Jialin Ding, Tim Kraska, Sivaprasad Sudhir, David Cohen, Timothy G. Mattson, and Nesime Tatbul. 2022.

ENGINEERING

Think `make`.

Queries ← Derived Indexes ← Immutable Data.

Navigate storage by layers of indexes.

Goal is to minimize latency.

Light Indexes for Storage

JUXT

Created on demand in background.

Idempotent, immutable and shared.

Merged to cluster data.

Data skipping is key.

Adaptive Indexes for Compute

JUXT

Dynamic in RAM.

Adapted by workload.

Local and transient.

Concurrent modification.

CODA

Outlook

Multi-Dimensional Indexes.

Succinct Data Structures.

Workload Prediction.

Summary

Index on demand, think `make`.

Copeland: "people do not delete."

Separation of Storage and Compute.

Academia is inspiration, not gospel.

References [1/3]

JUXT

Michael Armbrust, Tathagata Das, Sameer Paranjpye, Reynold Xin, Shuqin Zhu, Ali Ghodsi, Burak Yavuz, Mukul Murthy, Joseph Torres, Liwen Sun, Peter A. Boncz, Mostafa Mokhtar, Herman Van Hovell, Adrian Ionescu, Alicja Luszczak, Michal Switakowski, Takuya Ueshin, Xiao Li, Michal Szafranski, Pieter Senster, and Matei Zaharia. 2020. Delta Lake. *Proceedings of the VLDB Endowment* 13, (2020), 3411–3424.

Yael Ben-Haim and Elad Tom-Tov. 2010. A Streaming Parallel Decision Tree Algorithm. *J. Mach. Learn. Res.* 11, (2010), 849–872.

Jon Louis Bentley and James B. Saxe. 1980. Decomposable Searching Problems I: Static-to-Dynamic Transformation. *J. Algorithms 1*, (1980), 301–358.

Peter A. Boncz, Stefan Manegold, and Martin L. Kersten. 1999. Database Architecture Optimized for the New Bottleneck: Memory Access. In *VLDB*.

Matthias Brantner, Daniela Florescu, David A. Graf, Donald Kossmann, and Tim Kraska. 2008. Building a Database on S3. *In SIGMOD Conference*.

Lu Chen, Yunjun Gao, Xuan Song, Zheng Li, Xiaoye Miao, and Christian S. Jensen. 2022. Indexing Metric Spaces for Exact Similarity Search. *ACM Computing Surveys (CSUR)* (2022).

☆ George P. Copeland. 1980. What if Mass Storage Were Free? *IEEE Computer* 15, (1980), 27–35.

☆ George P. Copeland and Setrag Khoshafian. 1985. A Decomposition Storage Model. *In SIGMOD '85*.

Graham Cormode. 2017. Data Sketching. *Communications of the ACM (CACM)* 60, 9 (2017), 48–55.

Andrew Crotty, Viktor Leis, and Andrew Pavlo. 2022. Are You Sure You Want to Use MMAP in Your Database Management System? In *CIDR*.

Paolo Ferragina and Giorgio Vinciguerra. 2020. The PGM-index. Proceedings of the *VLDB Endowment 13*, (2020), 1162–1175.

Franz Färber, Norman May, Wolfgang Lehner, Philippe Grosse, Ingo Müller, Hannes Rauhe, and Jonathan Dees. 2012. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.* 35, (2012), 28–33.

Anurag Gupta, Deepak K. Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. 2015. Amazon Redshift and the Case for Simpler Data Warehouses. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015).

Felix Halim, Stratos Idreos, Panagiotis Karras, and Roland H. C. Yap. 2012. Stochastic Database Cracking: Towards Robust Adaptive Indexing in Main-Memory Column-Stores. *ArXiv* abs/1203.0055, (2012).

Michael Hammer and Arvola Chan. 1976. Index Selection in a Self-Adaptive Data Base Management System. In *SIGMOD '76*.

Pat Helland. 2015. Immutability Changes Everything. *Communications of the ACM* 59, (2015), 64–70.

Pedro Holanda, Stefan Manegold, Hannes Mühleisen, and Mark Raasveldt. 2019. Progressive Indexes: Indexing for Interactive Data Analysis. *Proc. VLDB Endow.* 12, (2019), 2366–2378.

Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Ling Shen, Liu Tang, Yuxing Zhou, Menglong Huang, Wan Wei, Cong Liu, Jian Zhang, Jianjun Li, Xuelian Wu, Lingyu Song, Ruoxi Sun, Shuaipeng Yu, Lei Zhao, Nicholas Cameron, Liquan Pei, and Xin Tang. 2020. TiDB: A Raft-based HTAP Database. *Proc. VLDB Endow.* 13, (2020), 3072–3084.

☆ Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2007. Database Cracking. In *CIDR*.

Stratos Idreos, Lukas M. Maas, and Michael S. Kester. 2017. Evolutionary Data Systems. *ArXiv* abs/1706.05714, (2017).

References [2/3]

JUXT

Stratos Idreos, Stefan Manegold, Harumi A. Kuno, and Goetz Graefe. 2011. Merging What's Cracked, Cracking What's Merged: Adaptive Indexing in Main-Memory Column-Stores. *Proc. VLDB Endow.* 4, (2011), 585–597.

Stratos Idreos, Konstantinos Zoumpatianos, Manos Athanassoulis, Niv Dayan, Brian Hentschel, Michael S. Kester, Demi Guo, Lukas M. Maas, Wilson Qin, Abdul Wasay, and Yiyou Sun. 2018. The Periodic Table of Data Structures. *IEEE Data Eng. Bull.* 41, (2018), 64–75.

Stratos Idreos, Konstantinos Zoumpatianos, Brian Hentschel, Michael S. Kester, and Demi Guo. 2018. The Data Calculator: Data Structure Design and Cost Synthesis from First Principles and Learned Cost Models. *Proceedings of the 2018 International Conference on Management of Data* (2018).

Alfons Kemper and Thomas Neumann. 2011. HyPer: A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots. *2011 IEEE 27th International Conference on Data Engineering* (2011), 195–206.

S. Keshav. 2007. How to Read a Paper. *Comput. Commun. Rev.* 37, (2007), 83–84.

Andreas Kipf, Dominik Horn, Pascal Pfeil, Ryan Marcus, and Tim Kraska. 2022. LSI: A Learned Secondary Index Structure. *Proceedings of the Fifth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management* (2022).

☆ Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2020. RadixSpline: A Single-Pass Learned Index. *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management* (2020).

Tim Kraska. 2021. Towards Instance-Optimized Data Systems. *Proc. VLDB Endow.* 14, (2021), 3222–3232.

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. *Proceedings of the 2018 International Conference on Management of Data* (2018).

Guido Moerkotte. 1998. Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing. In *VLDB*.

☆ Samuel Madden, Jialin Ding, Tim Kraska, Sivaprasad Sudhir, David Cohen, Timothy G. Mattson, and Nesime Tatbul. 2022. Self-Organizing Data Containers. In *CIDR*.

☆ Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning Multi-Dimensional Indexes. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (2020).

☆ Thomas Neumann and Sebastian Michel. 2008. Smooth Interpolating Histograms with Error Guarantees. In *BNCOD*.

Jürg Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. 1984. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Trans. Database Syst.* 9, (1984), 38–71.

Matthaios Olma, Manos Karpathiotakis, Ioannis Alagiannis, Manos Athanassoulis, and Anastasia Ailamaki. 2017. Slalom: Coasting Through Raw Data via Adaptive Partitioning and Indexing. *Proc. VLDB Endow.* 10, (2017), 1106–1117.

Patrick E. O'Neil, Edward Y. C. Cheng, Dieter Gawlick, and Elizabeth J. O'Neil. 2009. The Log-Structured Merge-Tree (LSM-tree). *Acta Informatica* 33, (2009), 351–385.

Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *CIDR*.

References [3/3]

JUXT

Eleni Petraki, Stratos Idreos, and Stefan Manegold. 2015. Holistic Indexing in Main-memory Column-stores. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015).

Adam Prout, Szu-Po Wang, Joseph Victor, Zhou Sun, Yongzhu Li, Jack Chen, Evan Bergeron, Eric N. Hanson, Robert Walzer, Rodrigo Gomes, and Nikita Shamgunov. 2022. Cloud-Native Transactions and Analytics in SingleStore. *Proceedings of the 2022 International Conference on Management of Data* (2022).

Konstantin V. Shvachko, Hairong Kuang, Sanjay R. Radia, and Robert J. Chansler. 2010. The Hadoop Distributed File System. *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (2010), 1–10.

Benjamin Spector, Andreas Kipf, Kapil Vaidya, Chi Wang, Umar Farooq Minhas, and Tim Kraska. 2021. Bounding the Last Mile: Efficient Learned String Indexing. *CoRR* abs/2111.14905, (2021).

Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil I. Hachem, and Pat Helland. 2007. The End of an Architectural Era (It's Time for a Complete Rewrite). In *VLDB.*

Ben Vandiver, S Durga Prasad, Pratibha Rana, Eden Zik, Amin Saeidi, Pratyush Parimal, Styliani Pantela, and Jaimin Dave. 2018. Eon Mode: Bringing the Vertica Columnar Database to the Cloud. *Proceedings of the 2018 International Conference on Management of Data* (2018).

Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal K. Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. *Proceedings of the 2017 ACM International Conference on Management of Data* (2017).

Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building An Elastic Query Engine on Disaggregated Storage. *In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 449–462.

Fangjin Yang, Eric Tschetter, Xavier Léauté, Nelson Ray, Gian Merlino, and Deep Ganguli. 2014. Druid: A Real-Time Analytical Data Store. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (2014).

Matei A. Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI.*

☆ Huanchen Zhang, Hyeontaek Lim, Viktor Leis, David G. Andersen, Michael Kaminsky, Kimberly Keeton, and Andrew Pavlo. 2018. SuRF: Practical Range Query Filtering with Fast Succinct Tries. *Proceedings of the 2018 International Conference on Management of Data* (2018).

Arrow Columnar Format. Retrieved from https://arrow.apache.org/docs/format/Columnar.html

CMU 15-445/645 :: Intro to Database Systems (Fall 2021) Retrieved from https://15445.courses.cs.cmu.edu/fall2021/schedule.html

CMU 15-721 :: Advanced Database Systems (Spring 2020) Retrieved from https://15721.courses.cs.cmu.edu/spring2020/schedule.html

Iceberg Table Spec. Retrieved from https://iceberg.apache.org/spec/

Rockset Concepts, Design and Architecture. Retrieved from https://rockset.com/Rockset_Concepts_Design_Architecture.pdf

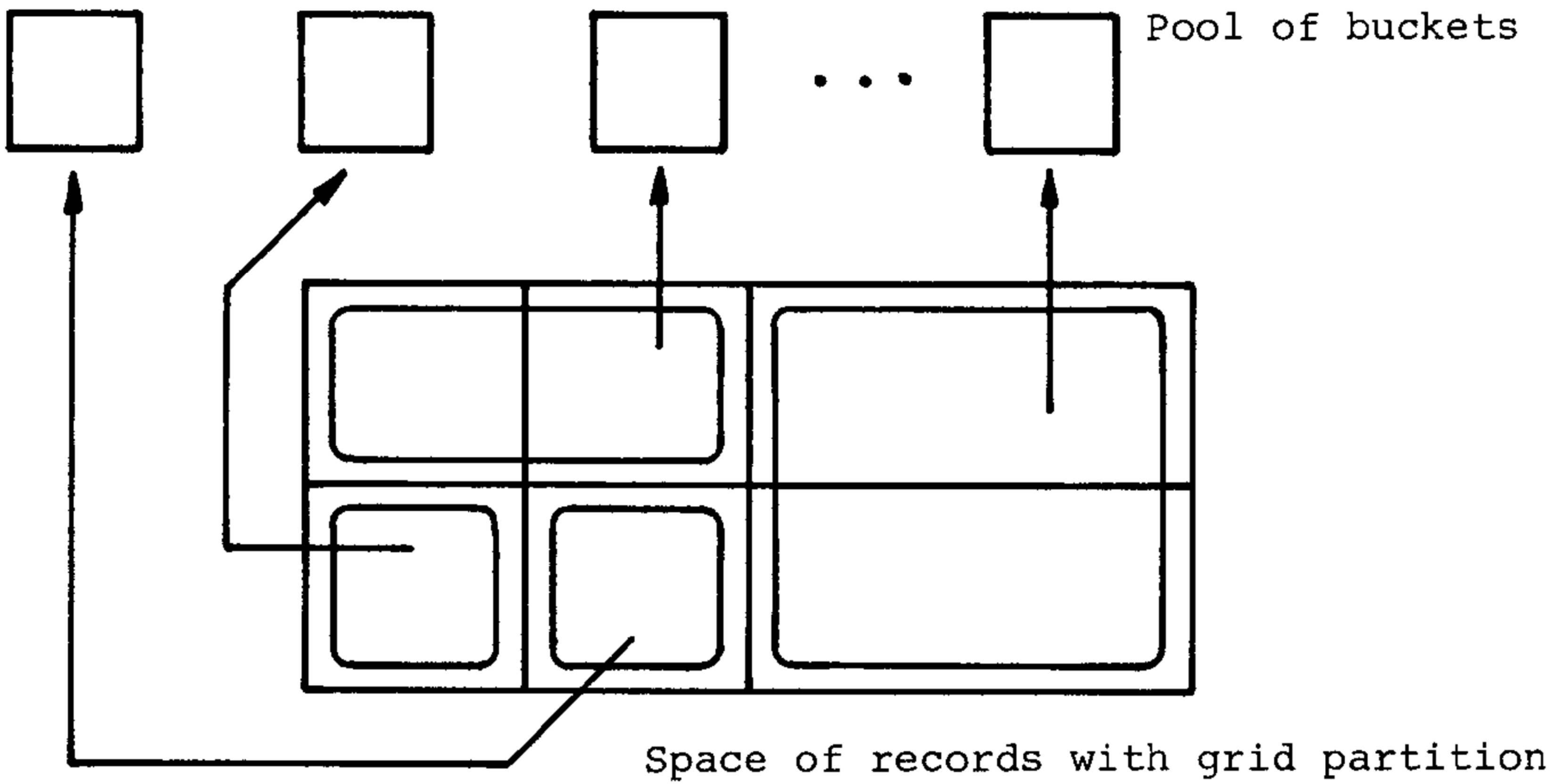
☆ Adam Storm. 2019. Separating Compute and Storage. What it Means, and Why it's Important for Databases. Retrieved from https://ajstorm.medium.com/separating-compute-and-storage-59def4f27d64

JUXT

XTDB

xtdb.com/core2

MULTI-DIMENSIONAL INDEXES
BONUS MATERIAL



The Grid File: An Adaptable, Symmetric Multikey File Structure.

Jürg Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. 1984.

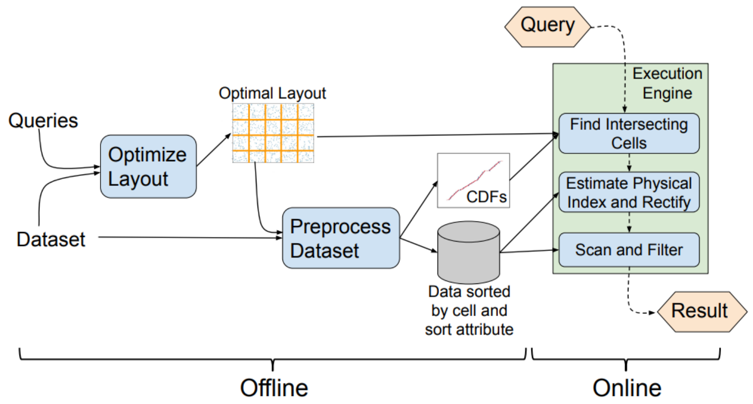


Figure 1: Flood's system architecture.

SageDB [22] proposed the idea of a learned multi-dimensional index but did not describe any details.

3 INDEX OVERVIEW

Flood is a multi-dimensional clustered index that speeds up the processing of relational queries that select a range over one or more attributes. For example:

```
SELECT SUM(R.X)
FROM MyTable
WHERE (a ≤ R.Y ≤ b) AND (c ≤ R.Z ≤ d)
```

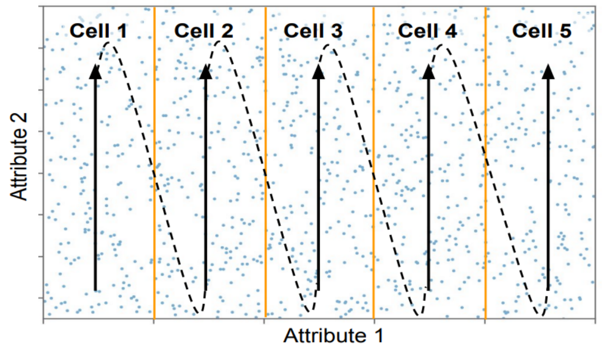


Figure 2: A basic layout in 2D, with dimension order (x, y) and $c_0 = 5$. Points are bucketed into columns along x and then sorted by their y-values, creating the seriliaziation order indicated by the arrows.

along the ith dimension, then define the dimension's range as $r_i = M_i - m_i + 1$. Then the cell for point $p = (p_1, \ldots, p_d)$ is:

$$\text{cell}(p) = \left(\left\lfloor \frac{p_1 - m_1}{r_1} \cdot c_1 \right\rfloor, \ldots, \left\lfloor \frac{p_{d-1} - m_{d-1}}{r_{d-1}} \cdot c_{d-1} \right\rfloor \right)$$

Note that the cell is determined only by the first $d-1$ dimensions; the dth dimension, the *sort dimension*, will be used to order points within a cell.

Learning Multi-Dimensional Indexes.

Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020.

JUXT

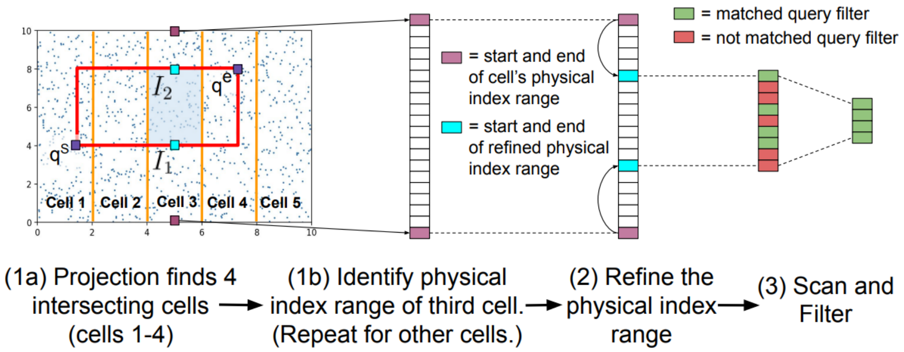


Figure 3: Basic flow of Flood's operation

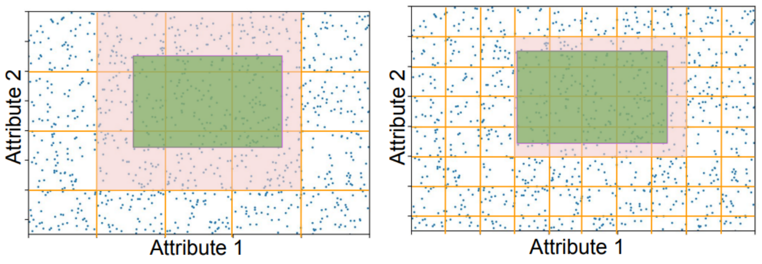


Figure 4: Doubling the number of columns can increase the number of visited cells but decreases the number of scanned points that don't match the filter (light red).

$(d-1)$-dimensional grid, computing intersections is straightforward. Suppose that each filter in the query is a range of the form $[q_i^s, q_i^e]$ for each indexed dimension i. If an indexed dimension is not present in the query, we simply take the start and end points of the range to be $-\infty$ and $+\infty$, respectively. Conversely, if the query includes a dimension not in the index, that filter is ignored at this stage of query processing.

(Fig. 4). However, adding more columns also increases the number of sub-ranges, which incurs extra cost for projection and refinement. Striking the right balance requires choosing a layout with an optimal number of columns in each dimension.

Flood can also select the sort dimension. The sort dimension is special because it will incur no scan overhead; given a query, Flood finds the precise sub-ranges to scan in the refinement step, so that the values in the sort dimension for scanned points are guaranteed to lie in the desired range. On the other hand, the grid dimensions do incur scan overhead because a certain column might only lie partially within the

Learning Multi-Dimensional Indexes.

Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020.

(3) For each of these d possible orderings, run a gradient descent search algorithm to find the optimal number of columns $\{c_i\}_{0 \le i < d-1}$ for the $d-1$ grid dimensions. The objective function is Eq. 1. For each call to the cost model, Flood computes the statistics $N = \{N_c, N_s\}$ and the input features of the weight models using the data sample instead of the full dataset D.

(4) Select the layout with the lowest objective function cost amongst the d layouts.

Optimizing the layout is efficient (§7.7) because each iteration of gradient descent does not require building the layout, sorting the dataset, or running the query. Instead, statistics are either estimated using a sample of D or computed exactly from the query rectangle and layout parameters.

5 LEARNING FROM THE DATA

The simple index presented in §3 does not consider or adapt to the underlying distribution of the data. Here, we present two ways that Flood learns its layout from the data. First, *Flood uses a model of each attribute to better determine column*

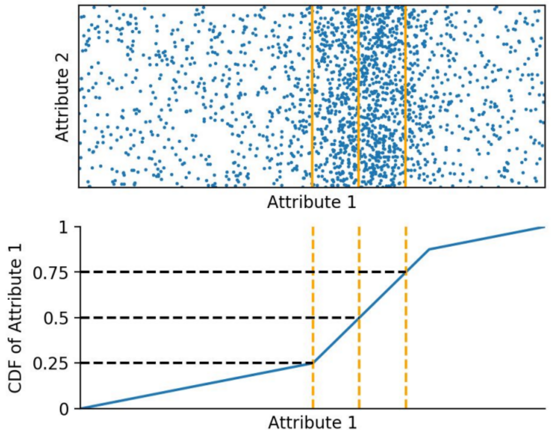


Figure 6: By flattening, each of the four columns in a dimension will contain a fourth of the points.

the datasets used in our evaluation (§7), flattening provides a performance boost of 20−30× over a non-flattened layout.

Note that while flattening may assign an equal number of points to each column of a single attribute, it does not guarantee that each cell in the final grid has a similar number of points. In particular, if two attributes are correlated, flattening each attribute independently will not yield uniformly sized cells. This may lead to some cells incurring a high scan overhead. In practice, we found that modeling single attributes,

Learning Multi-Dimensional Indexes.

Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020.

Algorithm 1 Update Procedure

input A histogram $h = \{(p_1, m_1), \ldots, (p_B, m_B)\}$, a point p.

output A histogram with B bins that represents the set $S \cup \{p\}$, where S is the set represented by h.

```
if p = p_i for some i then
    m_i = m_i + 1
else
    Add the bin (p, 1) to the histogram, resulting in a histogram of B+1 bins h ∪ {(p, 1)}. Denote
    p_{B+1} = p and m_{B+1} = 1.
    Sort the sequence p_1, ..., p_{B+1}. Denote by q_1, ..., q_{B+1} the sorted sequence, and let π be a
    permutation on 1, ..., B+1 such that q_i = p_{π(i)} for all i = 1, ..., B+1. Denote k_i = m_{π(i)},
    namely, the histogram h ∪ (p, 1) is equivalent to (q_1, k_1), ..., (q_{B+1}, k_{B+1}), q_1 < ... < q_{B+1}.
    Find a point q_i that minimizes q_{i+1} − q_i.
    Replace the bins (q_i, k_i), (q_{i+1}, k_{i+1}) by the bin
end if
```

$$\left(\frac{q_i k_i + q_{i+1} k_{i+1}}{k_i + k_{i+1}}, k_i + k_{i+1}\right).$$

A histogram is a set of B pairs (called bins) of real numbers $\{(p_1, m_1), \ldots, (p_B, m_B)\}$, where B is a

A Streaming Parallel Decision Tree Algorithm.

Yael Ben-Haim and Elad Tom-Tov. 2010.

THE END