



# 时序数据库关键技术简介

南京大學 刘世喆

注：此幻灯片模板是黄奕诚同学提供

# 大纲



- 论文数量统计
- 深入理解时序数据库
  - 数据库发展历史：了解过去，认识现在，把握未来
  - 影响数据库发展的六大因素：理清方向
  - 时序数据库简介：定义
  - 时序数据库六问：搞清楚为什么，继续前行
  - 数据库产品与排名：了解时序数据库产品
- 时序数据三大核心技术
  - LSM-Tree：解决大规模数据写入问题
  - 数据压缩和分布式存储：解决大规模数据存储问题
  - 索引：解决亿万级别数据在秒级时间内查询问题
- 个人思考与总结
- 参考文献



壹

# 论文数量统计



南京大學  
NANJING UNIVERSITY



# 论文数量统计说明

Time series Database



CONNECTED PAPERS

Share Follow About Feedback Donate

CONNECTED PAPERS

Applications of Frequent Pattern Mining

Share Follow About Feedback Donate

Applications of Frequent Pattern Mining

Prior works

Derivative works

Search... Expand

Origin paper  
Applications of Frequent Pattern Mining  
C. Aggarwal 2014

Frequent Pattern Mining  
C. Aggarwal, Jiawei Han 2014

An Introduction to Frequent Pattern Mining  
C. Aggarwal 2014

Frequent pattern mining: current status and future directions  
Jiawei Han, Hong Cheng, Dong Xin, Xifeng Yan 2007

Frequent Pattern Mining Algorithms: A Survey  
C. Aggarwal, Mansurul Bhuiyan, M. Hasan 2014

Mining and Using Sets of Patterns through Compression  
M. V. Leeuwen, Jilles Vreeken 2014

Data Mining : Concepts and Techniques  
M. Ramaswamy 2006

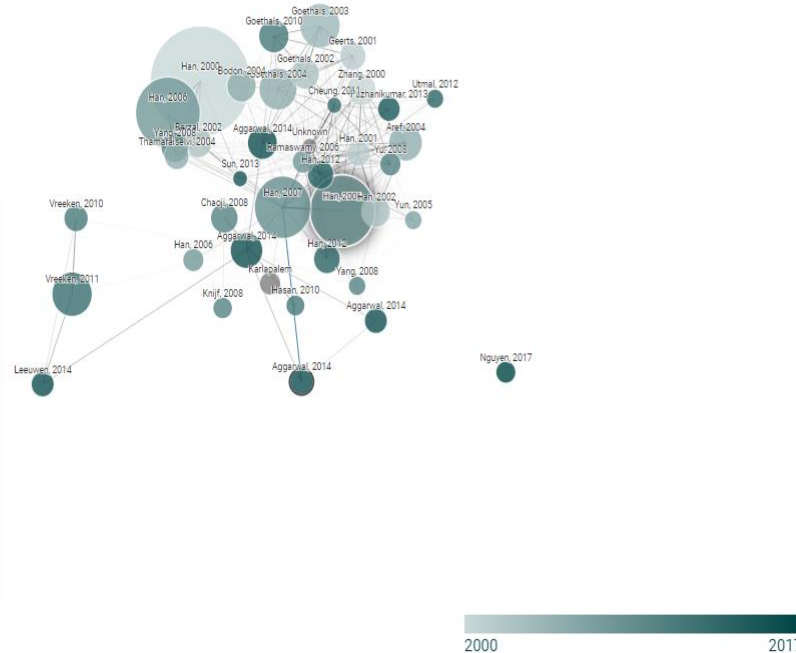
## How to read the graph

Each node is an academic paper related to the origin paper.

- Papers are arranged according to their similarity (this is not a citation tree)
- Node **size** is the number of citations
- Node **color** is the publishing year
- **Similar** papers have strong connecting lines and cluster together

[Learn more](#)

Created on Nov 26 2021



## Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach

Jiawei Han + 2 authors Runying Mao

2006, Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)

2527 Citations, 48 References

Open in:

Mining frequent patterns in transaction databases, time-series databases, and many other kinds of databases has been studied popularly in data mining research. Most of the previous studies adopt an Apriori-like candidate set generation-and-test approach. However, candidate set generation is still costly, especially when there exist a large number of patterns and/or long patterns. In this study, we propose a novel frequent-pattern tree (FP-tree) structure, which is an extended prefix-tree structure for storing compressed, crucial information about frequent patterns, and develop an efficient FP-tree-based mining method, FP-growth, for mining the complete set of frequent patterns by pattern fragment growth. Efficiency of mining is achieved with three techniques: (1) a large database is compressed into a condensed, smaller data structure, FP-tree which avoids costly, repeated database scans, (2) our FP-tree-based mining adopts a pattern-fragment growth method to avoid the costly generation of a large number of candidate sets,

非正式和其他出版物 Informal and Other Publications

# 近六年论文数量统计表



主要研究方向	CCF级别	2016	2017	2018	2019	2020	2021
A System	A	0	0	0	0	1	1
	B	0	0	1	0	0	1
Data Storage	A	1	0	0	1	2	2
LSM-Tree	A	0	2	4	9	10	14
Query Index	A	3	0	2	2	3	0
	B	1	1	2	1	4	0

目前偏重于统计A和B类文章

数据库最新的研究：查询可解释性、隐私数据查询、LSM+NVM、AI4DB and DB4AI



# 贰

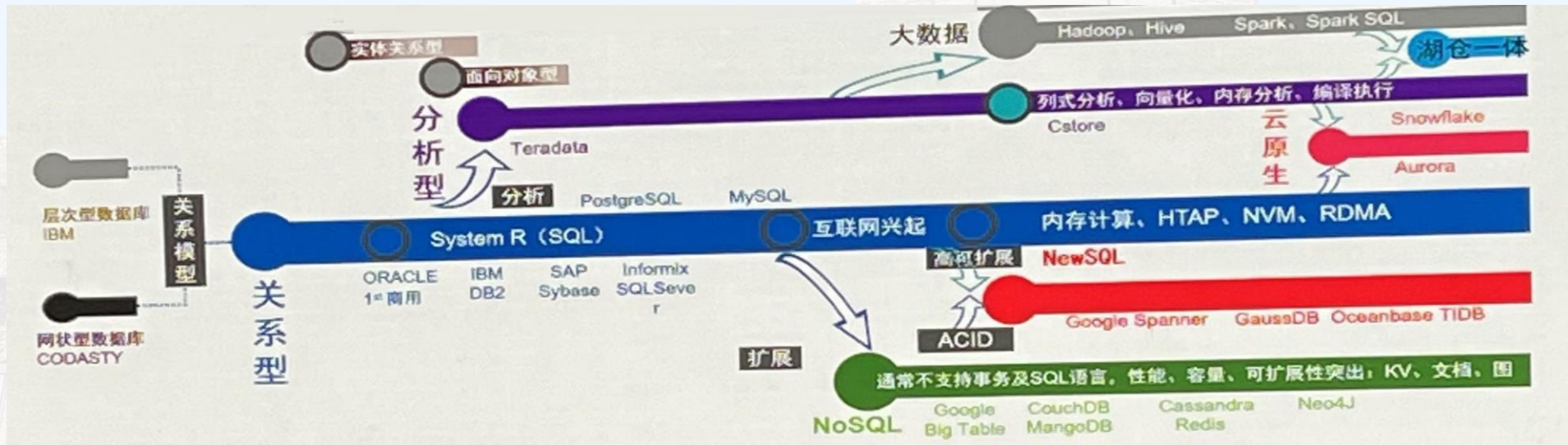
## 深入理解时序数据库





# 数据库发展历史

60年来，数据库技术顶天立地，蓬勃发展，数据库产业波澜壮阔，枝繁叶茂！——清华大学李国良教授



云原生  
智能化  
分布式  
边缘云

软硬协同  
安全隐私  
多模处理  
智能管理

数据库概念  
图灵奖：Charles Bachman

关系代数  
图灵奖：Edgar Codd

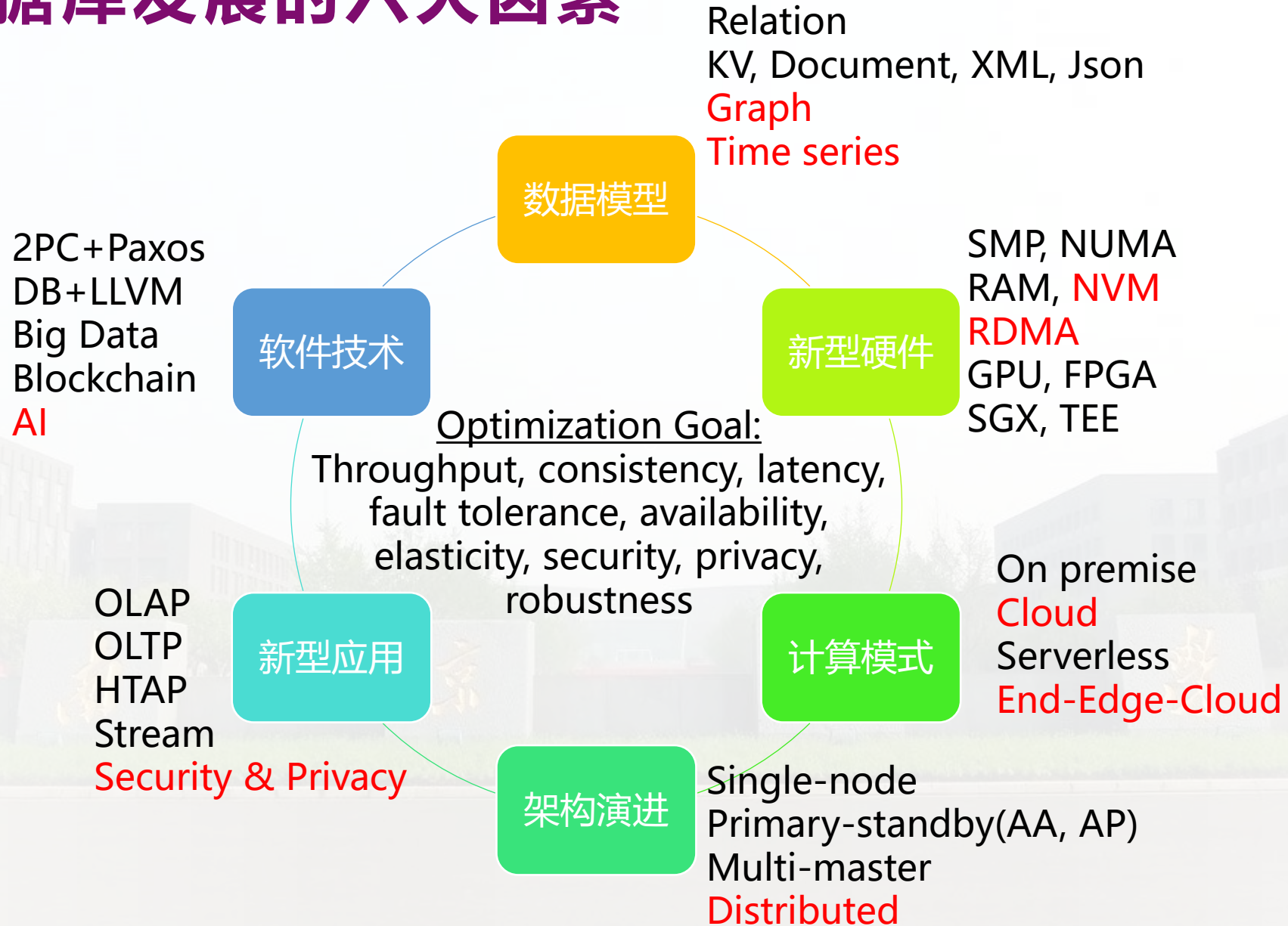
事务处理  
图灵奖：Jim Gray

数据库系统  
图灵奖：Mick.Stonebraker

分布式一致仲裁算法  
图灵奖：Leslie Lamport



# 影响数据库发展的六大因素







# 时序数据库简介

定义:

Wikipedia: A time series database (TSDB) is a software system that is optimized for storing and serving time series through associated pairs of time(s) and value(s).

时序数据库是为时间序列存储而优化的一个软件系统，时间序列是由多个时间和值对组成。

Timestamp	Name	Temperature
1580950800	Jack	36.5
1580950800	Peter	36.9
1580950800	Harry	36.7
1580950800	Rose	36.3
1580950946	Peter	37.0
1580950957	Harry	36.6
1580951566	Peter	37.1



# 时序数据详解

时序数据特点:

- **时序特性**: 每条数据都会有一个时间戳
- **数据特性**: 数据顺序可追加的, 多维可关联, 最近产生的时序数据往往是大家所关注的, 访问频率会比较高
- **CRUD特性**: **C**reate & **R**etrieve & **U**ppdate & **D**elete

写操作频率远远大于读, 极少更新数据, 支持批量删除操作。时序数据库不具备事物能力。



1.时序(Time Series): 时序的一个维度由顺序数据点和一个独特的 series key组成, 用公式表示为:  $\langle SK, \langle DP1, DP2, \dots, DP3 \rangle \rangle$ 。

2.单维度查询(Single-dimension query)

3.多维度查询(Multi-dimension query)



# 时序数据举例



时序数据库存储的时序数据格式举例

a dimension

series key		data point	
series_name	tags	timestamp	value
mem_usage	local=nanjing,os=linux,id=1	1580540293000	0.3
mem_usage	local=shanghai,os=win,id=2	1580540293010	0.4
mem_usage	local=nanjing,os=win,id=3	1580540293014	0.2
mem_usage	local=shanghai,os=linux,id=4	1580540293017	0.1
cpu_usage	local=nanjing,os=linux,id=1	1580540293026	0.5
cpu_usage	local=nanjing,os=win,id=3	1580540293032	0.6
cpu_usage	local=nanjing,os=linux,id=1	1580540293049	0.7
cpu_usage	local=shanghai,os=win,id=2	1580540293084	0.6



# 时序数据库六问

1. 为什么工程师要不断开发新的数据库?

→ 新业务需求

→ 国家战略需要

→ 资金因素



2. 为什么没有在各种场景下都适用的数据库?

**本质上看，数据库是在特定的问题场景需求下对性能折中。**





# 有关时序数据库六问



## 3.为什么要有时序数据库?



→存储数据成本：将海量数据全部存储起来开销很大。

→需要快速分析海量数据

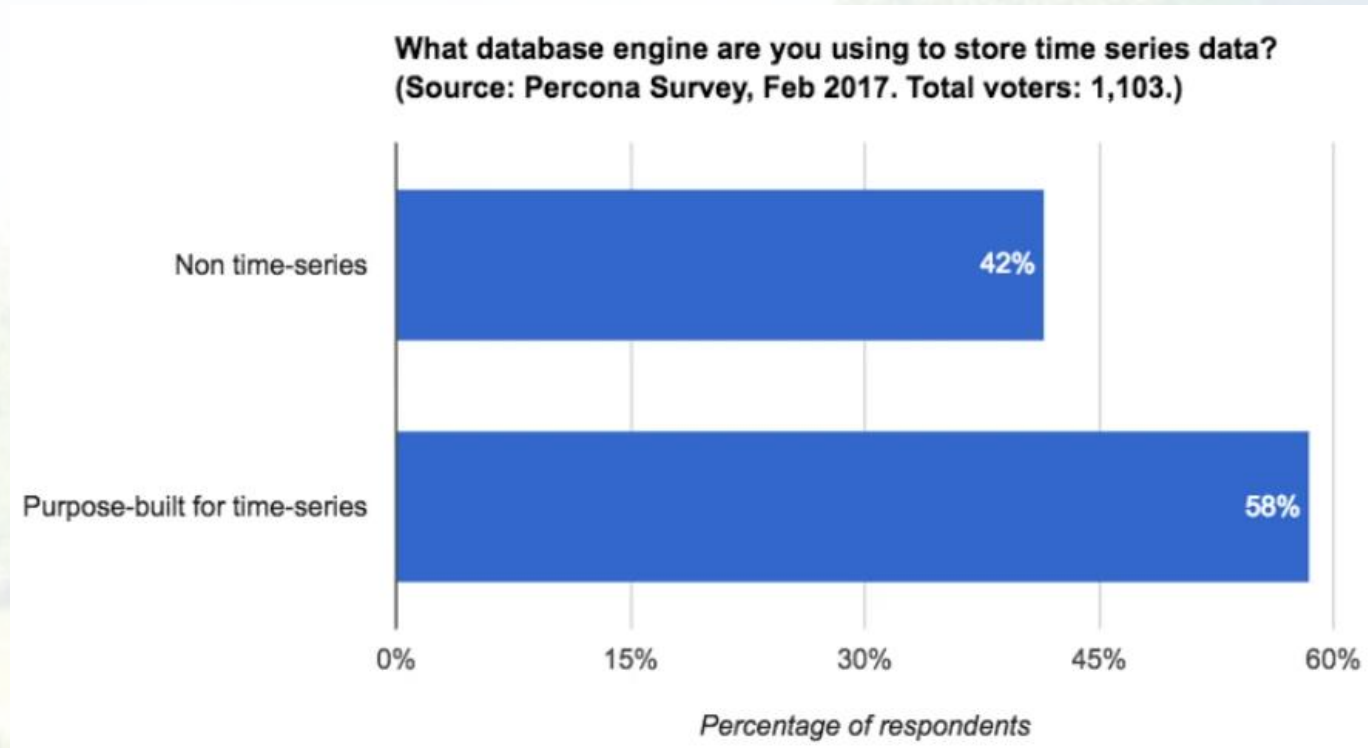
业界需要一种新的数据库来支持**快速写入**，**快速查询和分析**等功能。



# 时序数据库六问

4. 可以用非时序数据库来存储时序数据吗，例如：关系数据库？

当然可以。数据库就是用来存储数据的，时序数据也是数据，自然用非时序数据库来存储。但是，非时序数据库面对海量数据存储的时候，略显示不足。（图：存储时序数据的各类数据库所占比例）



数据来源：<https://www.percona.com/blog/2017/02/10/percona-blog-poll-database-engine-using-store-time-series-data/>





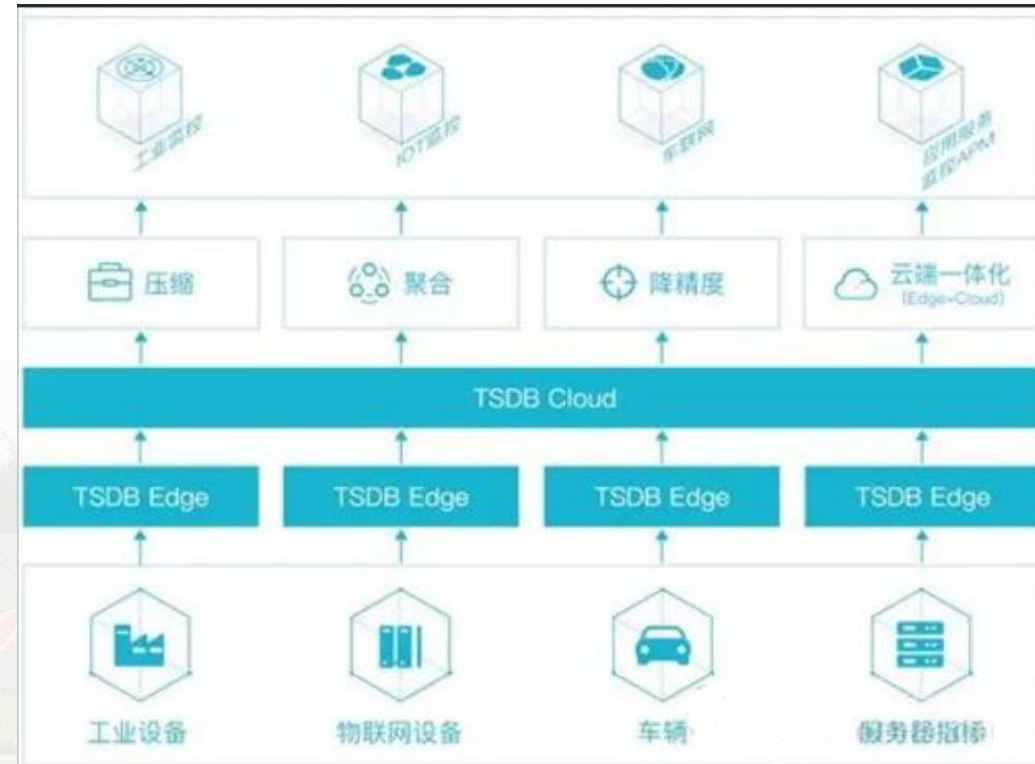
# 时序数据库六问

## 5.时序数据库需要解决的问题？

→时序数据的写入：如何支持每秒钟上亿数据点的写入。

→时序数据的读取：如何支持在秒级别对上亿数据的查询

→时序数据的存储：海量数据存储带来的是成本问题。如何低成本存储这些数据，是时序数据库需要解决的问题





# 时序数据库六问

## 6.时序数据库可以应用的场景?

- 监控软件系统：虚拟机、容器、服务、应用
- 监控物理系统：车联网、人
- 金融交易系统：传统证券、新兴的加密数字货币
- 事件应用程序：跟踪用户、客户的交互数据
- 商业智能工具：跟踪关键指标和业务的总体健康情况
- 更多场景.....

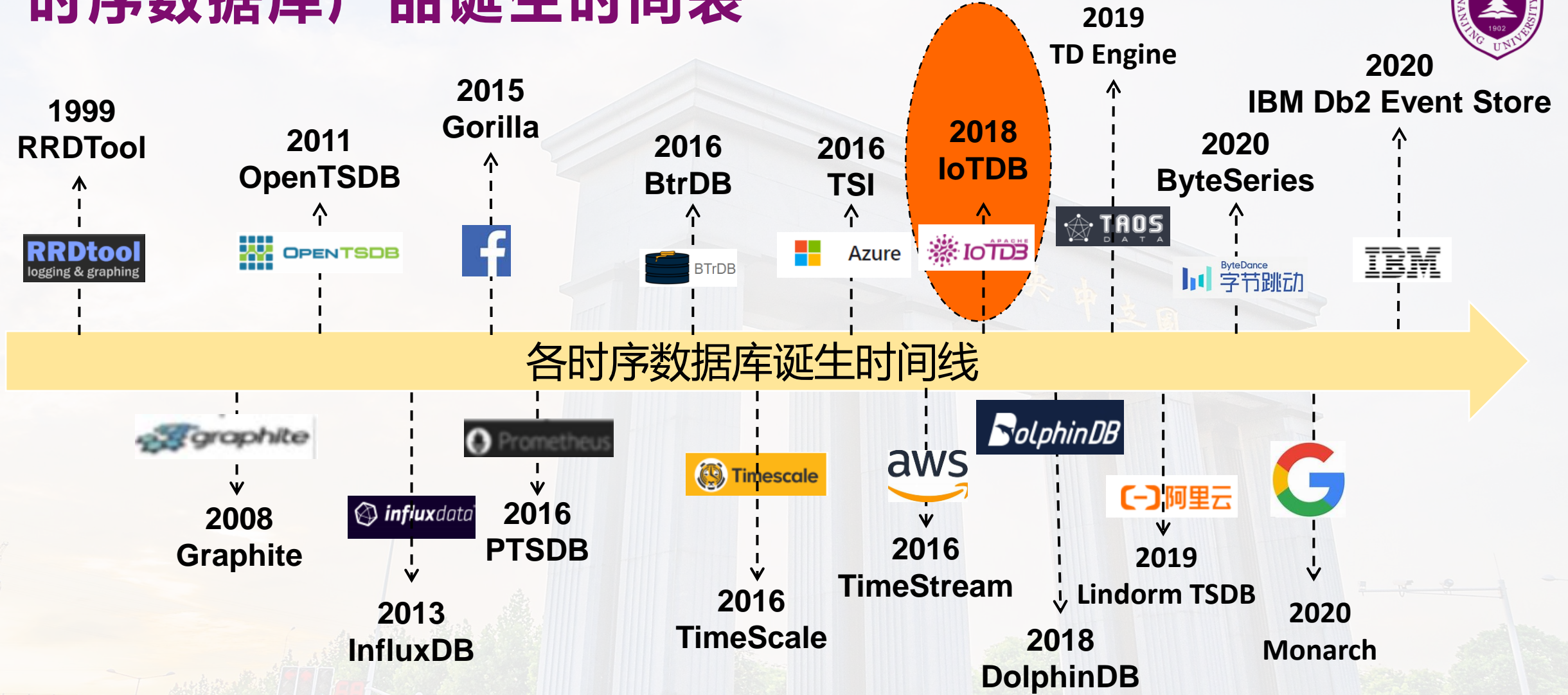
即使时序数据库能应用到这么多场景下，业界仍需选择最适合他们数据写入和读取模式的时间序列数据库。







# 时序数据库产品诞生时间表

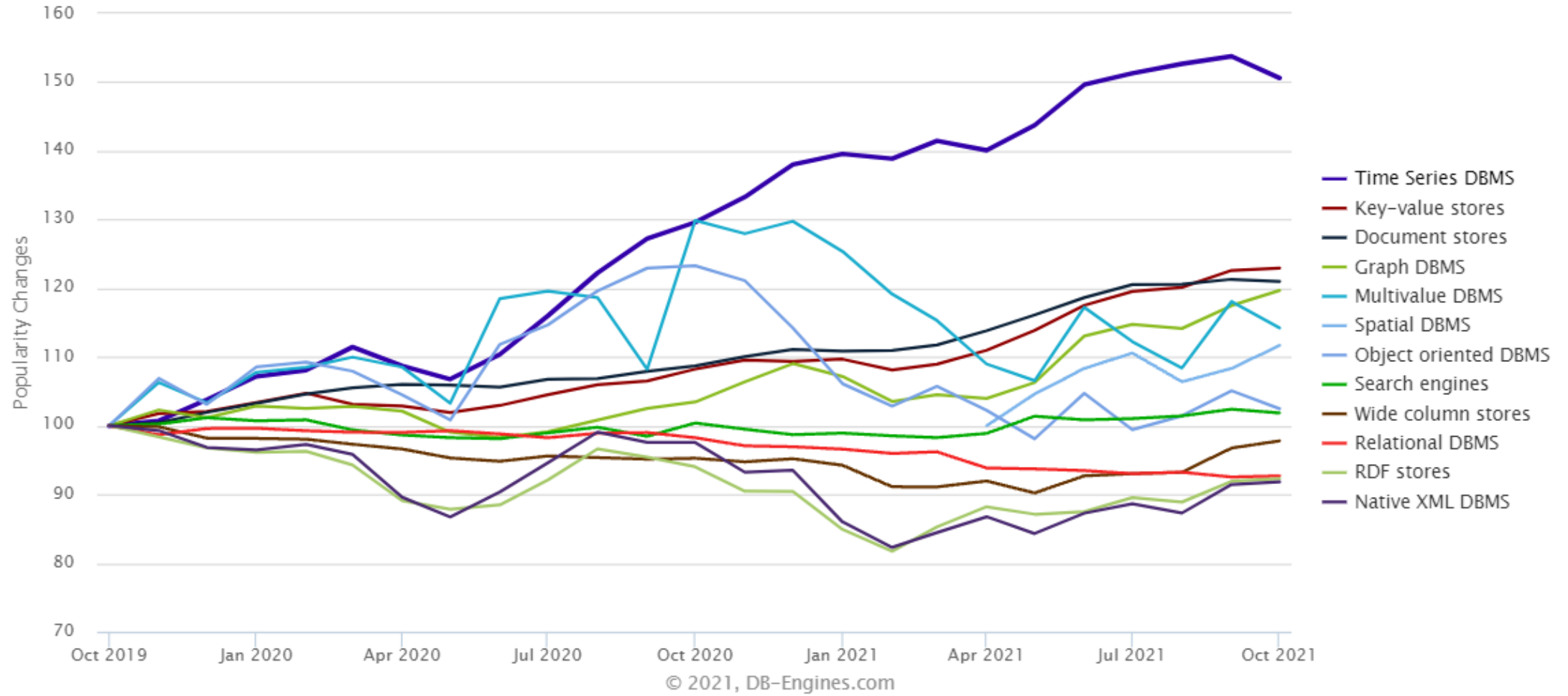


注：只列出了部分有名的时序数据库产品



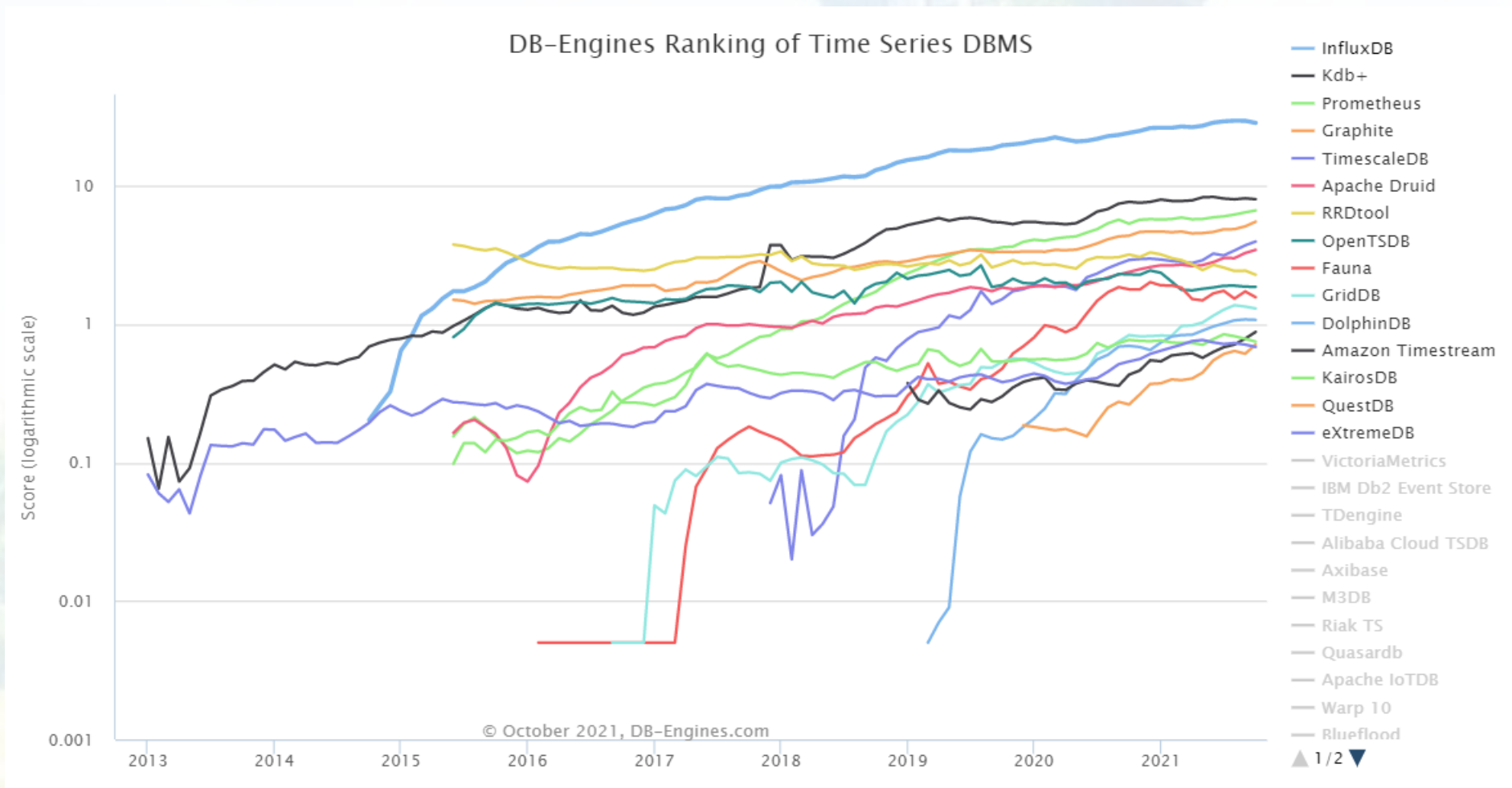


# 最近两年各类型数据库流行度



数据来源: DB-Engines

# 时序数据库分数排名图



数据来源: DB-Engines



# 时序数据库分数排名表



浙江智奥科技公司研发

涛思公司研发

阿里公司研发

清华大学团队研发

include secondary database models

38 systems in ranking, October 2021

Rank			DBMS	Database Model	Score		
Oct 2021	Sep 2021	Oct 2020			Oct 2021	Sep 2021	Oct 2020
1.	1.	1.	InfluxDB +	Time Series, Multi-model	28.52	-0.98	+4.37
2.	2.	2.	Kdb+ +	Time Series, Multi-model	8.00	-0.13	+0.34
3.	3.	3.	Prometheus	Time Series	6.64	+0.21	+1.31
4.	4.	4.	Graphite	Time Series	5.50	+0.41	+1.14
5.	5.	↑ 6.	TimescaleDB +	Time Series, Multi-model	3.96	+0.24	+1.05
6.	6.	↑ 7.	Apache Druid	Multi-model	3.44	+0.17	+1.06
7.	7.	↓ 5.	RRDtool	Time Series	2.28	-0.16	-0.91
8.	8.	8.	OpenTSDB	Time Series	1.86	0.00	-0.43
9.	9.	9.	Fauna	Multi-model	1.57	-0.16	-0.22
10.	10.	10.	GridDB	Time Series, Multi-model	1.30	-0.04	+0.46
11.	11.	↑ 12.	DolphinDB	Time Series	1.08	-0.01	+0.38
12.	12.	↑ 16.	Amazon Timestream	Time Series	0.88	+0.09	+0.45
13.	13.	↓ 11.	KairosDB	Time Series	0.75	-0.04	-0.02
14.	↑ 15.	↑ 21.	QuestDB +	Time Series, Multi-model	0.71	+0.10	+0.45
15.	↓ 14.	↓ 13.	eXtremeDB +	Multi-model	0.69	-0.03	+0.15
16.	16.	↑ 24.	VictoriaMetrics +	Time Series	0.54	-0.05	+0.30
17.	17.	17.	IBM Db2 Event Store	Multi-model	0.48	+0.04	+0.08

18.	↑ 20.		TDengine +	Time Series, Multi-model	0.36	+0.03	
19.	19.	↓ 15.	Alibaba Cloud TSDB	Time Series	0.33	-0.01	-0.12
20.	↓ 18.	↓ 19.	Axibase	Time Series	0.30	-0.06	-0.04
21.	↑ 22.	↑ 23.	M3DB	Time Series	0.24	0.00	-0.02
22.	↓ 21.	↓ 18.	Riak TS	Time Series	0.23	-0.03	-0.15
23.	23.	↓ 22.	Quasardb +	Time Series	0.19	-0.04	-0.07
24.	24.		Apache IoTDB	Time Series	0.18	+0.01	
25.	↑ 26.	↑ 26.	Warp 10	Time Series	0.16	+0.02	-0.02
26.	↓ 25.	↓ 25.	Blueflood	Time Series	0.11	-0.05	-0.08
27.	27.	↑ 31.	Bangdb +	Multi-model	0.09	-0.03	+0.07
28.			ArcadeDB	Multi-model	0.07		
29.	↓ 28.	↓ 14.	Heroic	Time Series	0.07	-0.02	-0.39
30.	↓ 29.	↓ 20.	Machbase +	Time Series	0.05	-0.04	-0.25
31.	↓ 30.	↓ 27.	Hawkular Metrics	Time Series	0.04	-0.03	-0.06
32.	32.	↓ 30.	SiteWhere	Time Series	0.03	0.00	+0.02
33.	33.	↓ 32.	NSDb	Time Series	0.00	0.00	+0.00
34.	↑ 35.	↓ 32.	Hyprcubd	Time Series	0.00	±0.00	±0.00
34.	34.	↓ 28.	IRONdb	Time Series	0.00	0.00	-0.10
34.	↑ 35.	↓ 32.	Newts	Time Series	0.00	±0.00	±0.00
34.	↓ 31.	↓ 29.	SiriDB	Time Series	0.00	-0.04	-0.09
34.	↑ 35.	↓ 32.	Yanza	Time Series	0.00	±0.00	±0.00

数据来源: DB-Engines



# 叁

## 时序数据库三大核心技术



南京大學  
NANJING UNIVERSITY

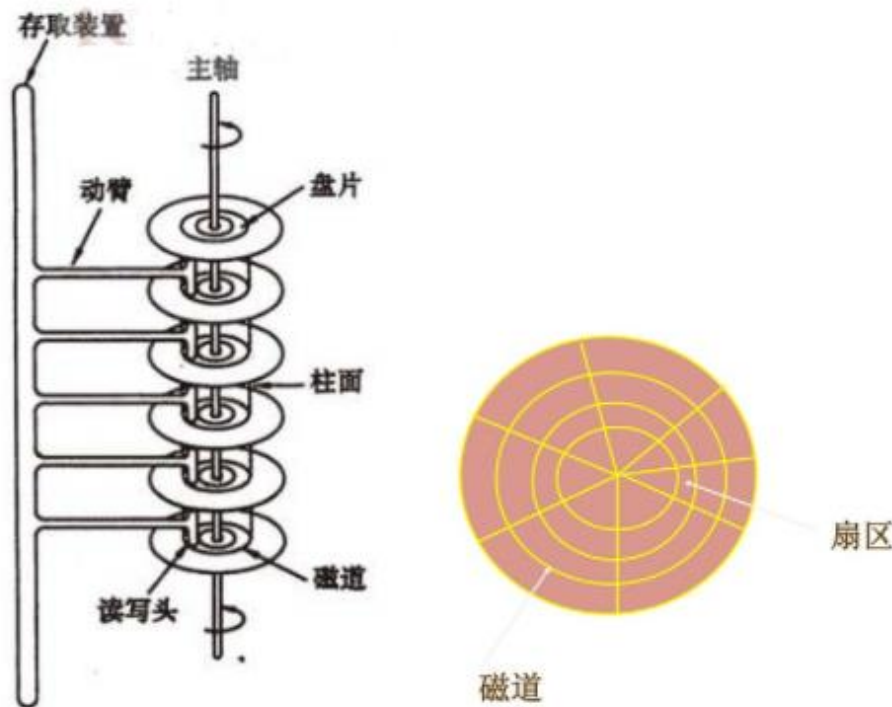


# LSM-Tree



日志合并树(Log Structured Merge Tree, LSM-Tree): 解决大规模数据写问题

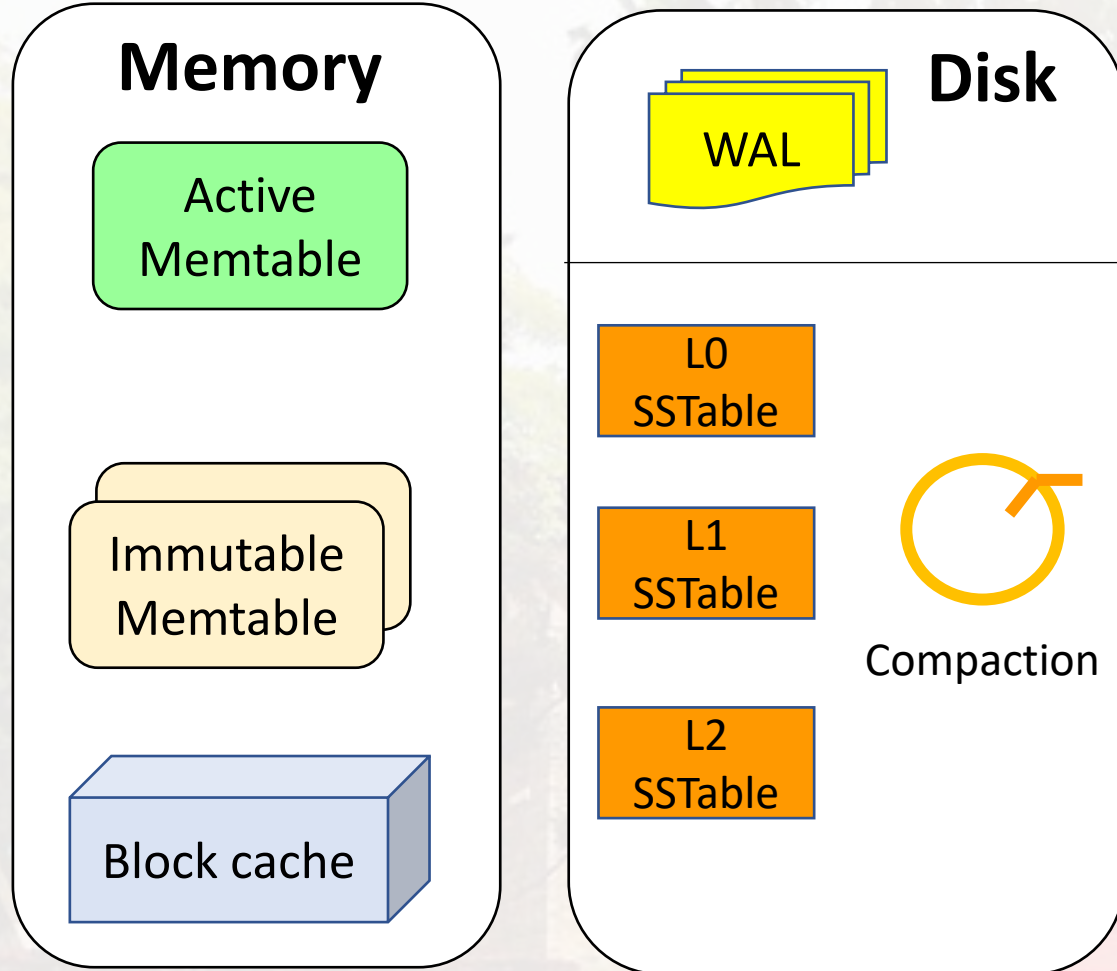
传统数据库存储采用的都是 B Tree/B+ Tree, 这是由于其在查询和顺序插入时有利于减少寻道次数的组织形式。



LSM核心思想就是**利用顺序写来提高写性能**, 但因为LSM分层设计(此处分层是指的分为内存和文件两部分), 降低了LSM读性能 (LSM是通过牺牲小部分读性能换来高性能写的)。



# LSM结构介绍

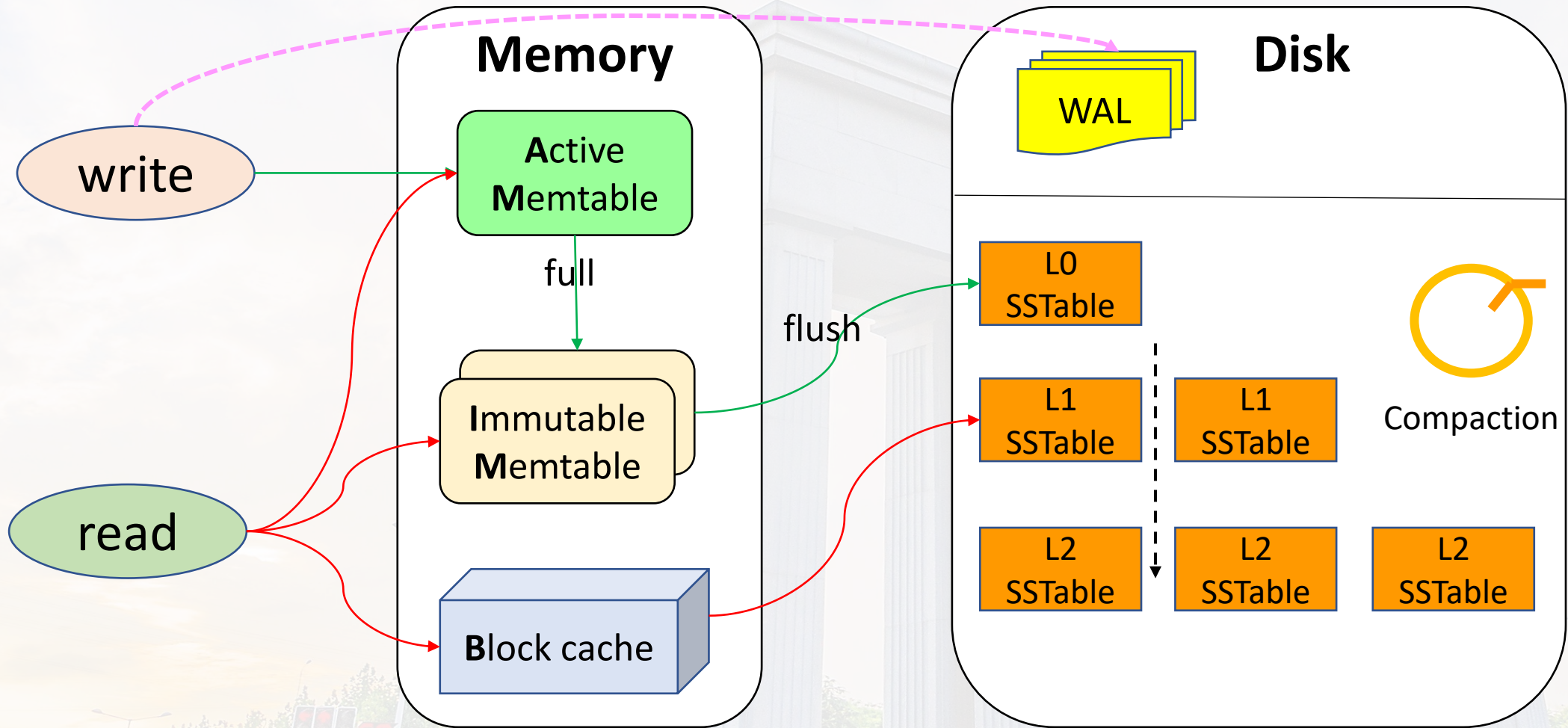


LSM树由Active MemTable(AM)、Immutable MemTable(IM)、SSTable三个重要部分组成。

- Active MemTable(AM) : 是在内存中的数据结构, 用于保存最近更新的数据, 会按照Key有序地组织这些数据
- Immutable MemTable(IM) : 当 MemTable达到一定大小后, 会转化成Immutable MemTable。写操作由新的MemTable处理, 在转存过程中不阻塞数据更新操作。
- SSTable(Sorted String Table) : 有序键值对集合, 是LSM树组在磁盘中的数据结构



# LSM读写流程





# LSM查询加速

LSM牺牲了读速率来换取写速率，那么如何尽可能保证读速率够高呢？

Key <sub>1</sub>	Value <sub>1</sub>	Key <sub>2</sub>	Value <sub>2</sub>	Key <sub>3</sub>	Value <sub>3</sub>	...	...
------------------	--------------------	------------------	--------------------	------------------	--------------------	-----	-----

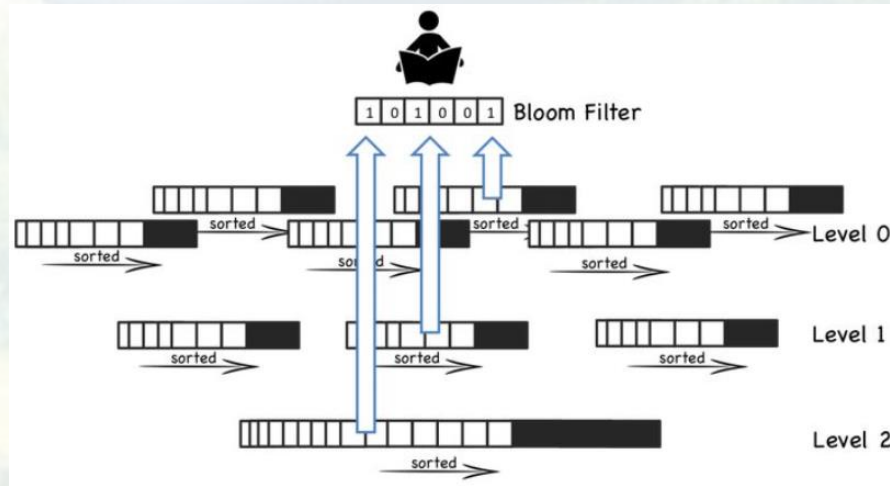
SSTable

由于内存读取数据比磁盘快很多，那么加速查询主要要优化磁盘的数据查询。为了加快SSTable的数据读取，可以使用key的索引表以及布隆过滤器来加快key的查找。

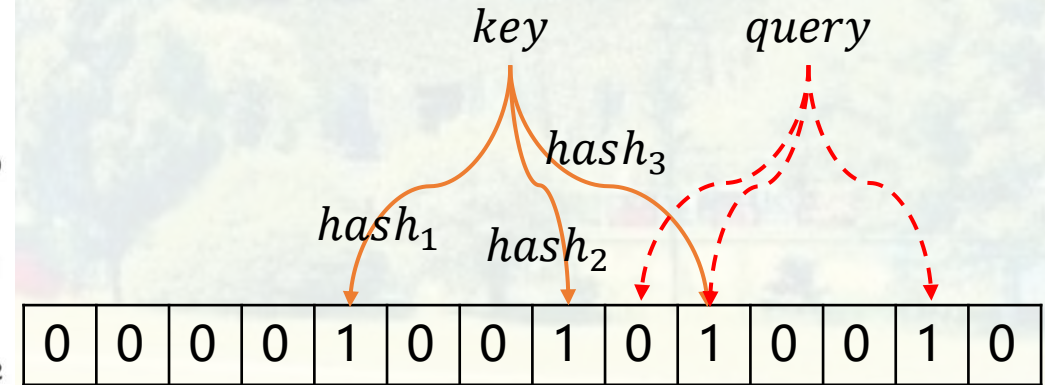
Index

Key <sub>1</sub>	Offest <sub>1</sub>
Key <sub>2</sub>	Offest <sub>2</sub>
Key <sub>3</sub>	Offest <sub>3</sub>
...	...

Key的索引表



使用布隆过滤器避免不必要的读



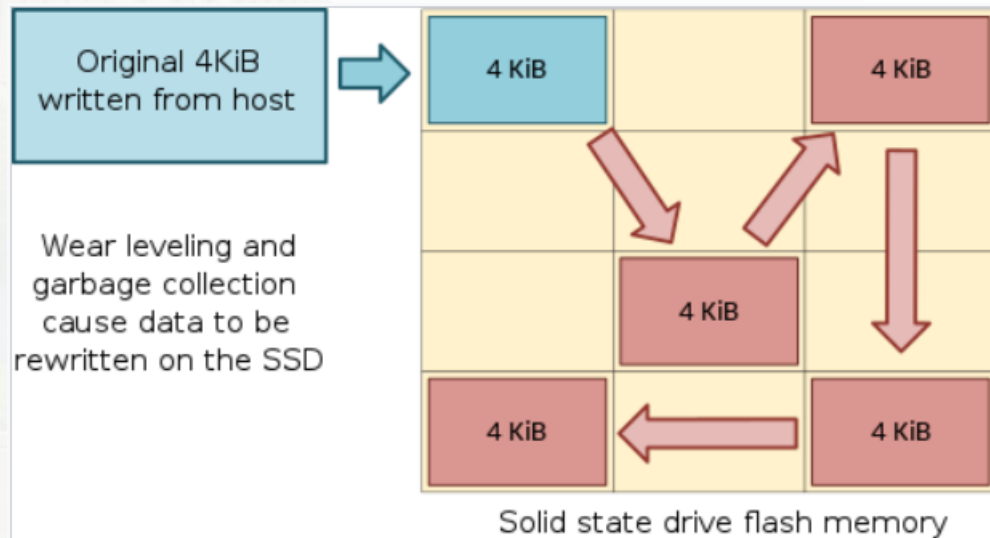
布隆过滤器





# LSM不足

- 1) **读放大**：读取数据时实际读取的数据量大于真正的数据量。例如在LSM树中需要先在MemTable查看当前key是否存在，不存在继续从SSTable中寻找。
- 2) **写放大**：写入数据时实际写入的数据量大于真正的数据量。例如在LSM树中写入时可能触发Compact操作，导致实际写入的数据量远大于该key的数据量。
- 3) **空间放大**：数据实际占用的磁盘空间比数据的真正大小更多。因为可能key即存在内存中，又存在磁盘各层SSTable中，这个key是冗余存储的。



写放大例子



部分顶会文章

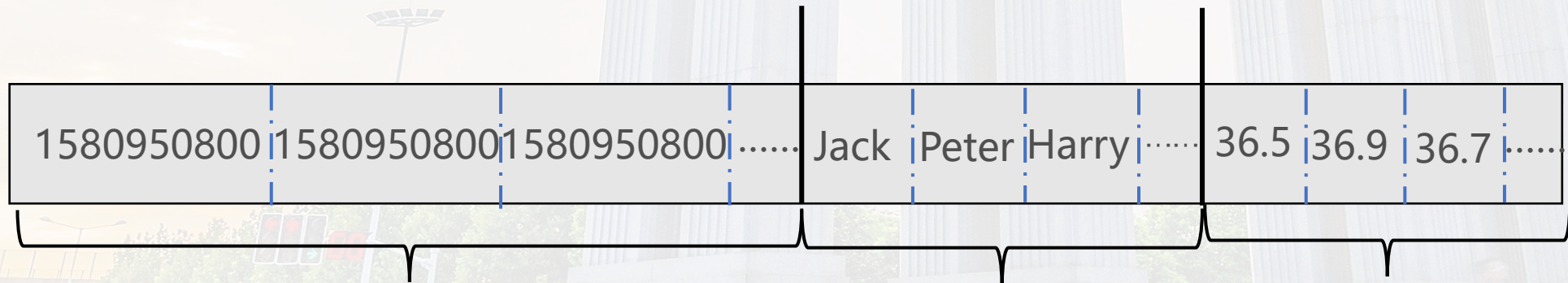
# 数据压缩

数据压缩解决的是数据存储问题，目前存储数据采取两种存储方式：行式存储与列式存储。

Timestamp	Name	Temperature
1580950800	Jack	36.5
1580950800	Peter	36.9
1580950800	Harry	36.7
.....	.....	.....



行式存储



Encode Chunk

列式存储

Encode Chunk

Encode Chunk





# 行式 vs. 列式

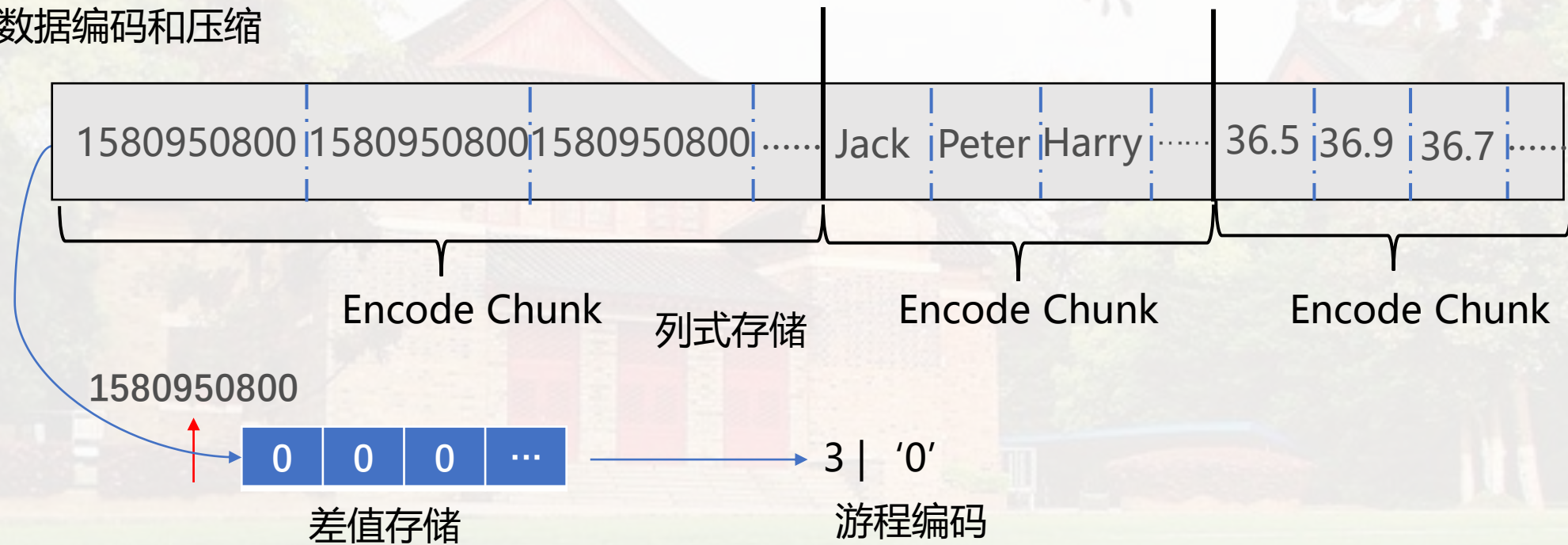
1. 行式存储：把逻辑相关的数据在硬盘上放到一起

2. 列式存储：将物理相关的数据放到一起

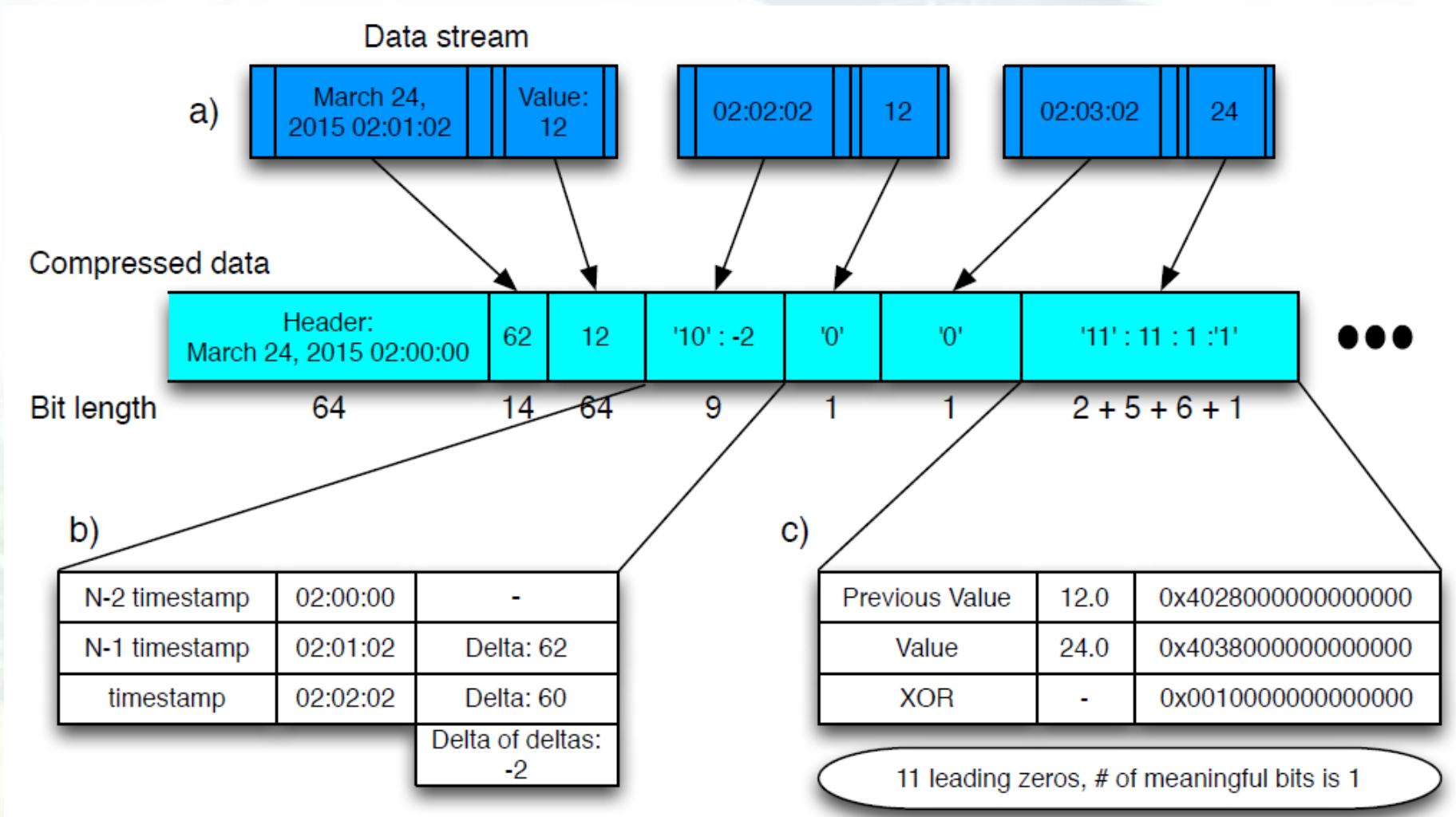
→ 只读投影列

e.g., **select** Temperature **from** table **where** Temperature > 36.5

→ 数据编码和压缩



# 压缩算法举例

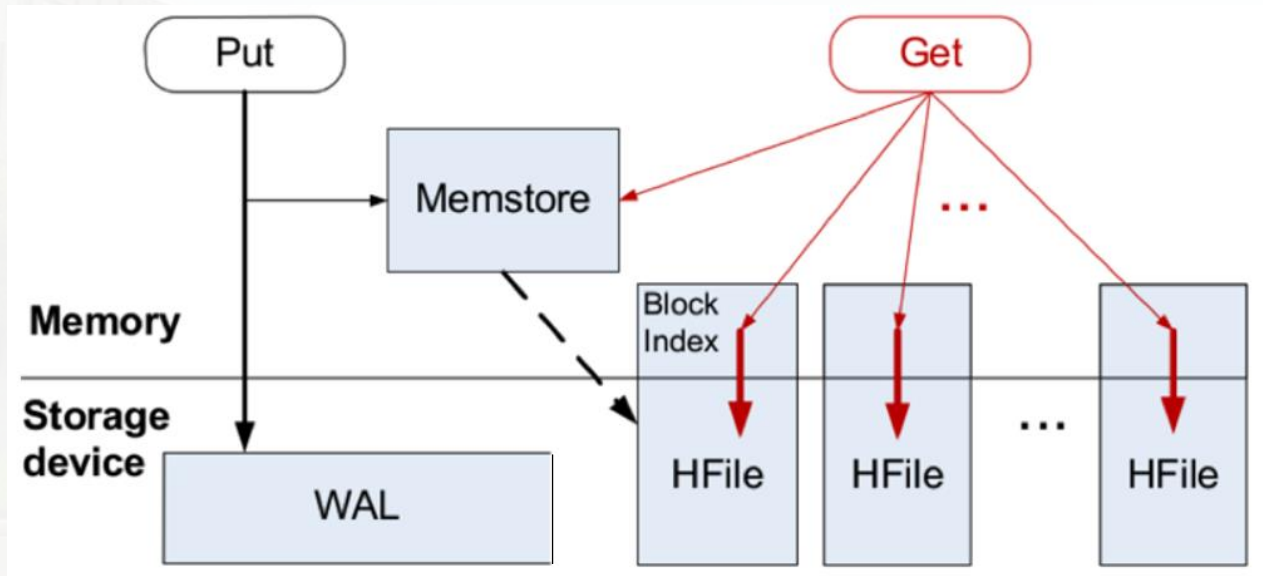




# 分布式存储

分布式存储是时序数据库另一个关键技术

存储结构：单机、集中、分布式、云



单机存储结构

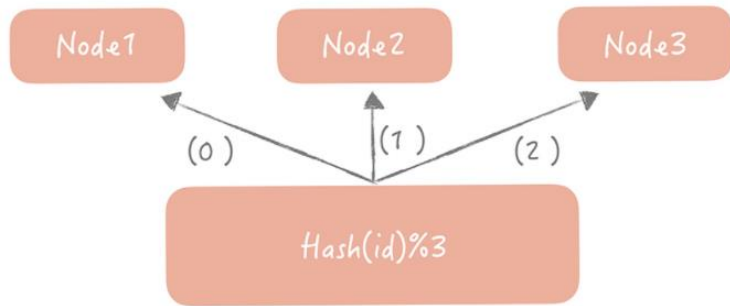


→单机存储缺点?

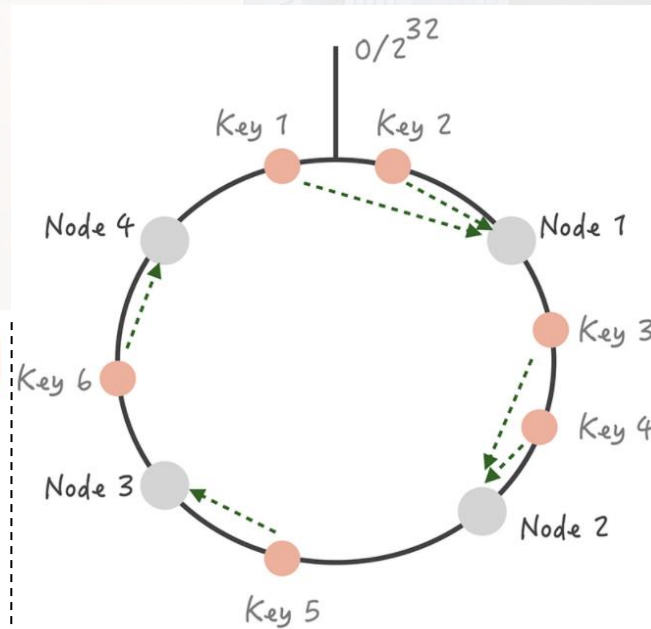
# 分布式存储之分片

分布式存储需要考虑如何将数据分布到多台机器上面，即分片(sharding)的问题。分片问题包括分片方法的选择和分片的设计问题。分片方法有：

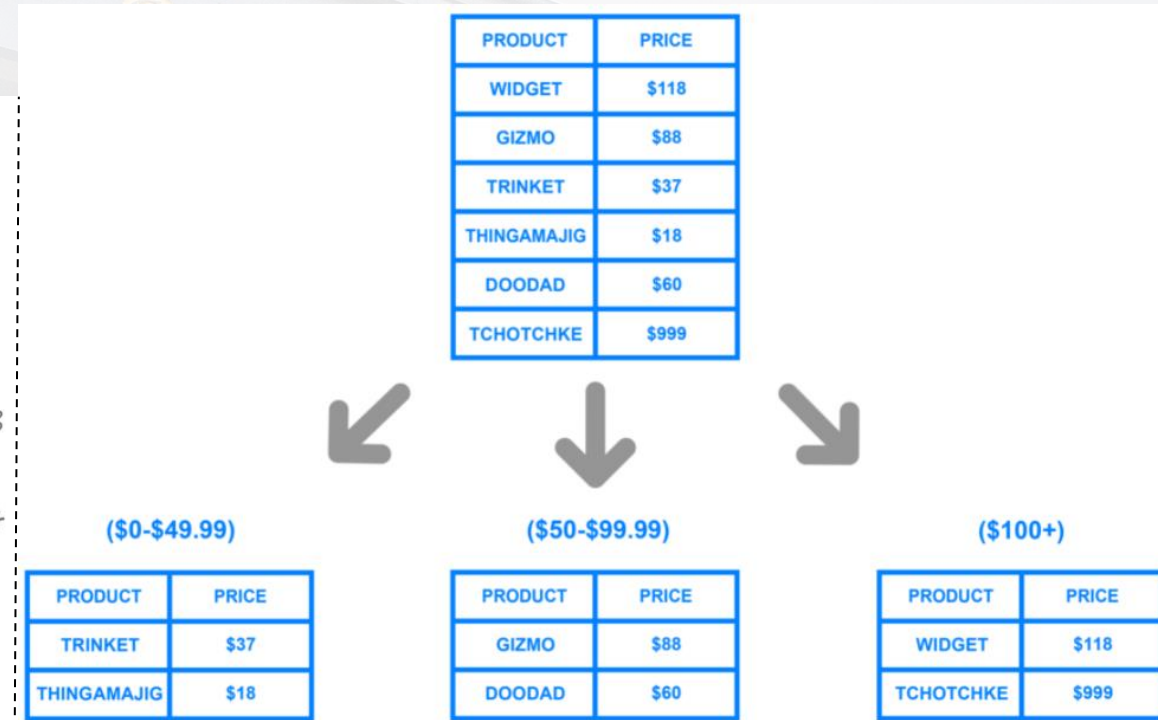
- 哈希分片
- 一致哈希
- 范围划分



哈希分片



一致哈希



范围划分





# 分片方法总结

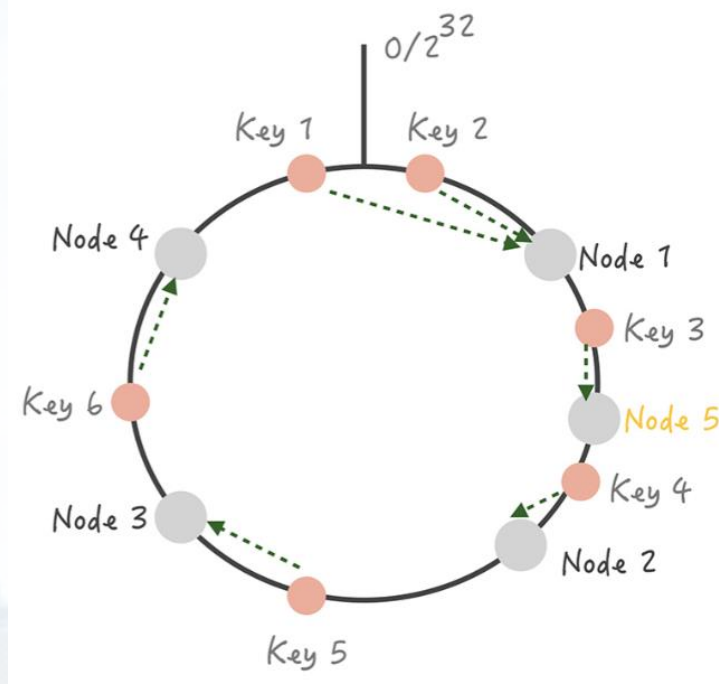
□ 哈希分片：均衡性较好，但集群不易扩展

□ 一致哈希：均衡性好，集群扩展易（增加删除结点），但实现复杂

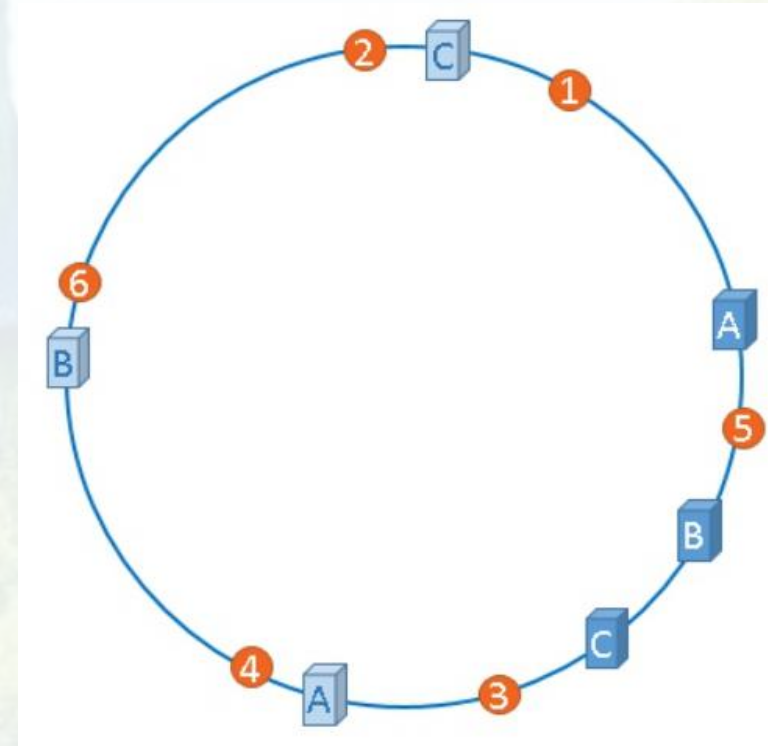
➢ 脏数据问题

➢ 雪崩效应与虚拟结点

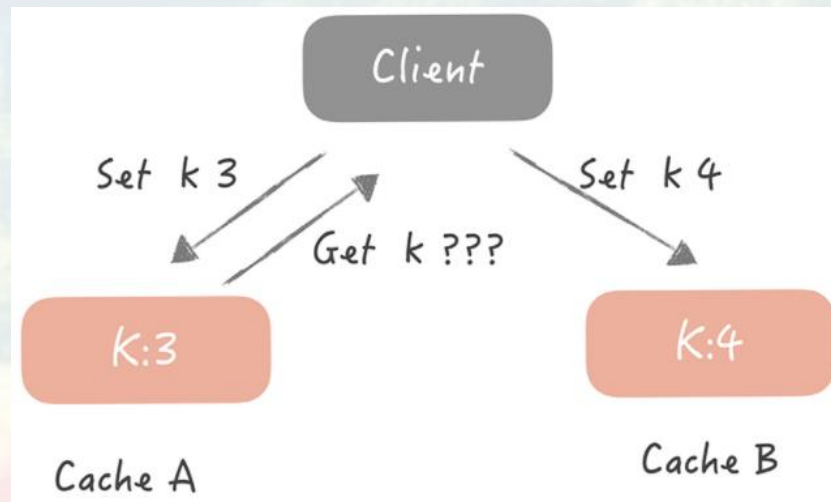
□ 范围划分：易水平扩展，复杂度在于合并和分裂



一致哈希中增加结点



一致哈希中虚拟结点



一致哈希中脏数据问题

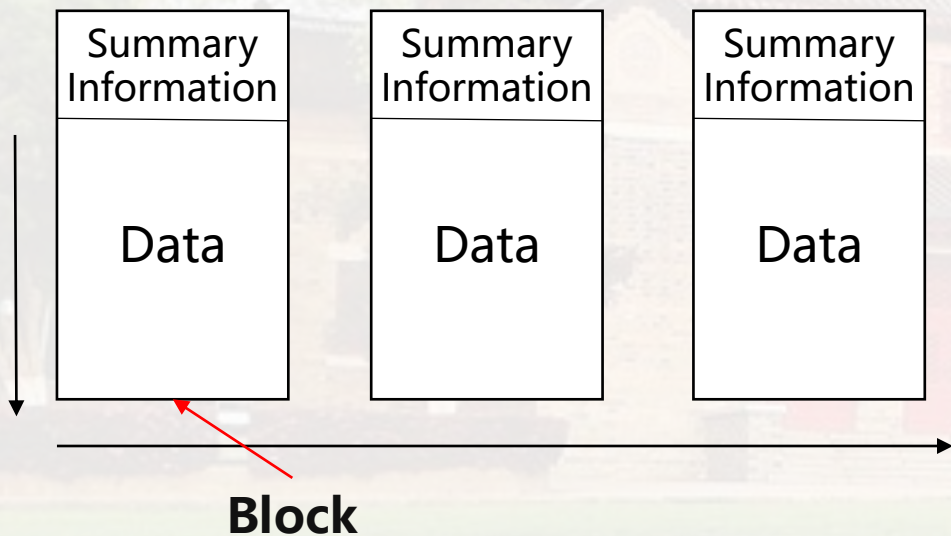
[CIKM'16] PISA: An Index for Aggregating Big Time Series Data

解决的问题: 对上亿数据快速进行聚合查询(SUM, MAX, MIN, AVG)

传统聚合查询使用到的方法:

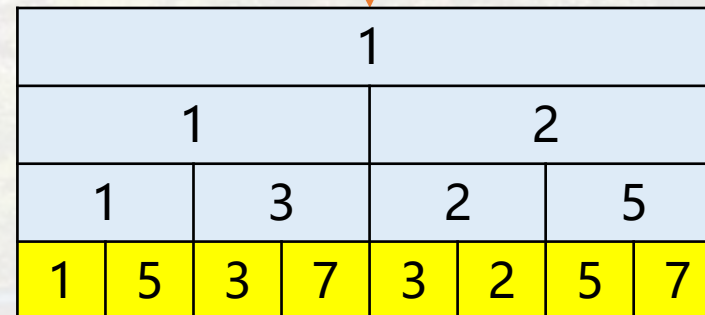
1. 摘要表

2. 线段树(二叉搜索树, 用来存区间信息)



[1, 5, 3, 7, 3, 2, 5, 7]

构造其最小线段树



The diagram shows a segment tree structure. The root node is a light blue rectangle labeled '1'. It branches into two child nodes, also labeled '1' and '2'. The node labeled '1' branches into two leaf nodes labeled '1' and '3'. The node labeled '2' branches into two leaf nodes labeled '2' and '5'. The leaf nodes are arranged in a row, and their corresponding data values are shown in a yellow row below them: 1, 5, 3, 7, 3, 2, 5, 7. An orange arrow points from the array [1, 5, 3, 7, 3, 2, 5, 7] to the root node of the tree.

1							
1				2			
1		3		2		5	
1	5	3	7	3	2	5	7





# 线段树不足

1. 插入新的数据对数据写入速度有影响
2. 占用的内存还是比较大
3. 读取有时开销较大

1							
1				2			
1	3	2	5				
1	5	3	7	3	2	5	7

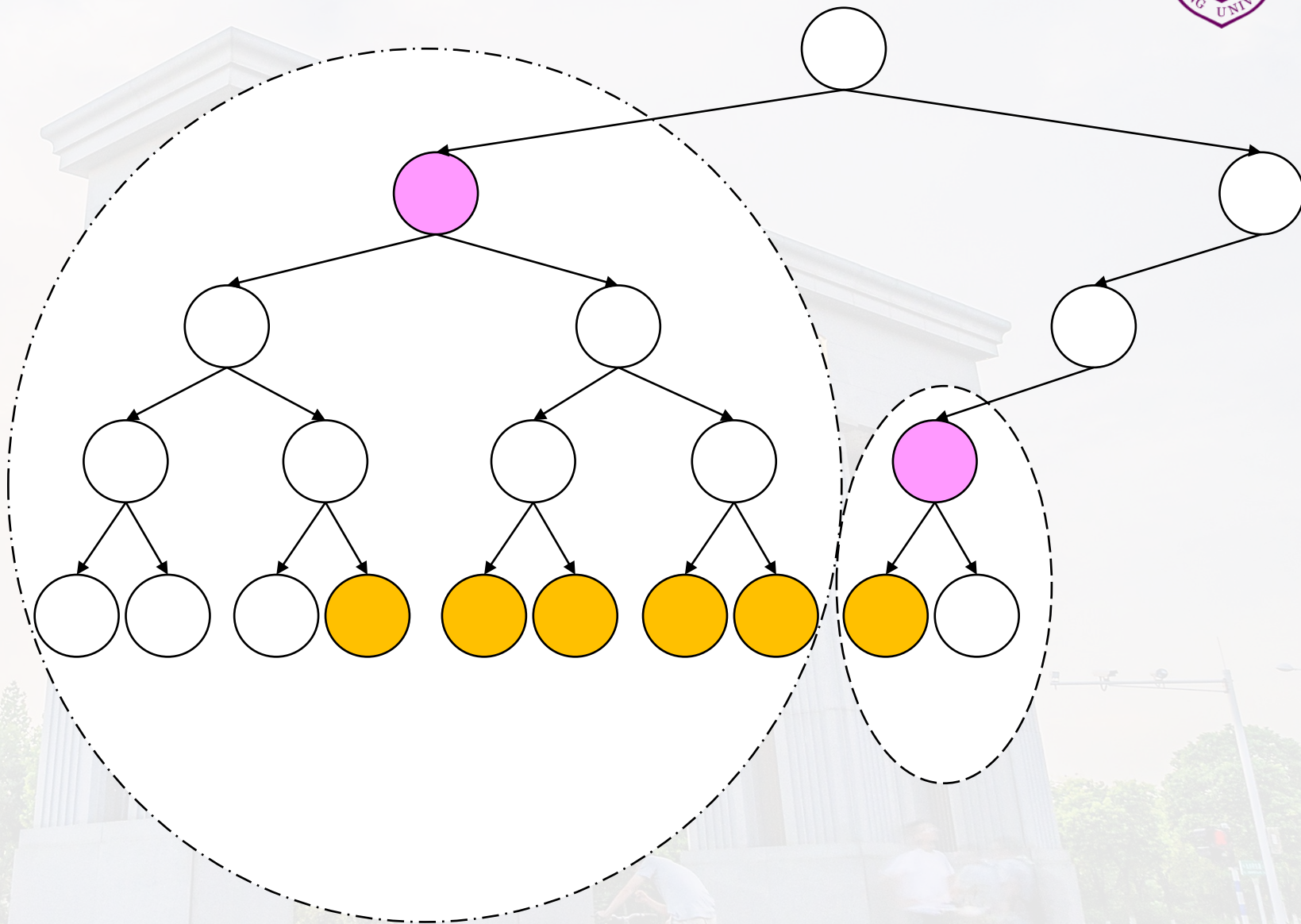
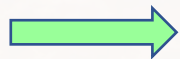
增加两个数据点

1									
1								2	
1				2				2	
1	3	2	5					2	
1	5	3	7	3	2	5	7	9	2

# PISA解决方案



1									
1				2					
1		2		2					
1	3	2	5	2					
1	5	3	7	3	2	5	7	9	2



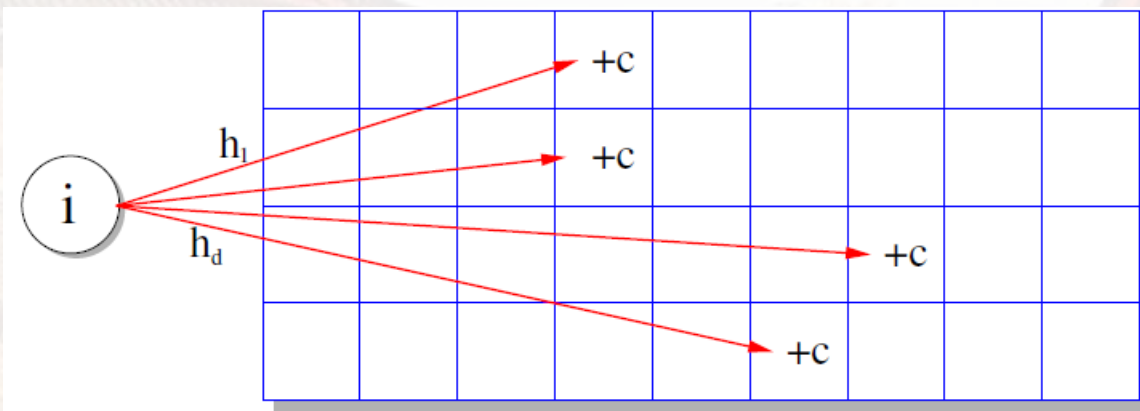


# Sketch



问题描述：如何查询在某个范围内的元素出现的总次数？例如数据库常常需要SELECT count(v) WHERE v  $\geq$  l AND v < r。要求查询速度要快，结果可以不完全精确。

初步解决方案：Count-Min Sketch可以计算每个元素的频率，那么我们把指定范围内所有元素的频率加起来，不就是这个范围内元素出现的总数了吗？



问题：由于每个sketch都是近似值，多个近似值相加，误差会被放大，所以这个方法不可行。



# Multi-Layer CMS

解决的办法就是使用多个“分辨率”不同的Count-Min Sketch。

对于一个数据流 $S = \{x_1, x_2, \dots, x_n\}$ ,  $S$ 中不同元素个数是 $R$ , 则对 $R$ 进行编号 $\{0, 1, 2, \dots, R-1\}$ , 假设 $R=256$ , 需要统计编号为 $[127, 172)$ 的元素出现次数。

$$127_{(10)} = 01111111_{(2)}$$

$$172_{(10)} = 10101100_{(2)}$$

使用8个sketch, 每个sketch统计的是一个前缀出现的次数 ( $256=2^8$ ), 8个bit可以表示这256个元素。

前缀长度为1的前缀: 0、1;

前缀长度为2的前缀: 00、01、10、11;

前缀长度为3的前缀: 000、001、010、011、100、101、110、111;

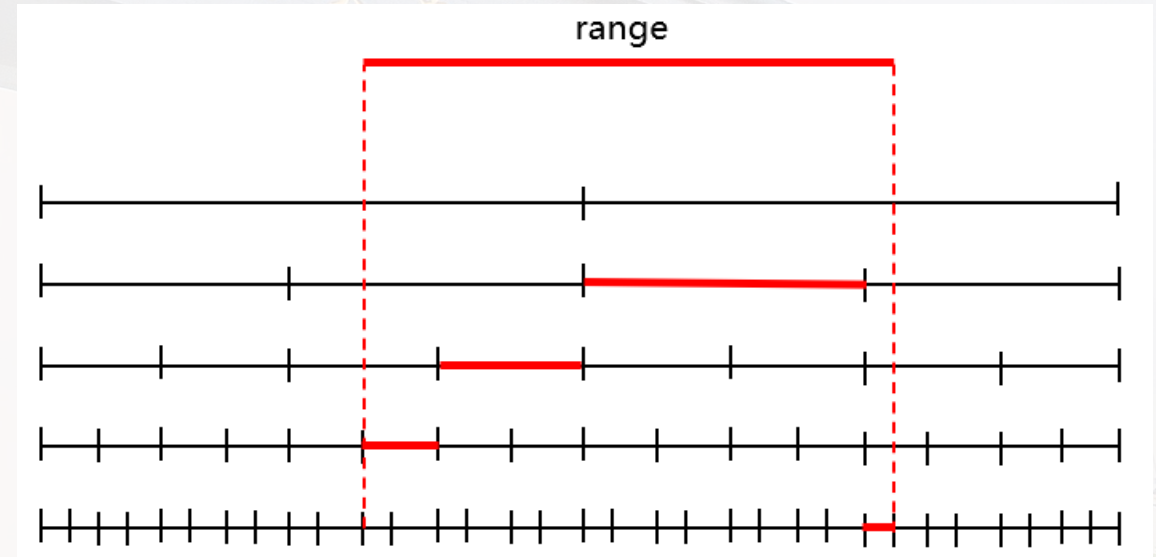
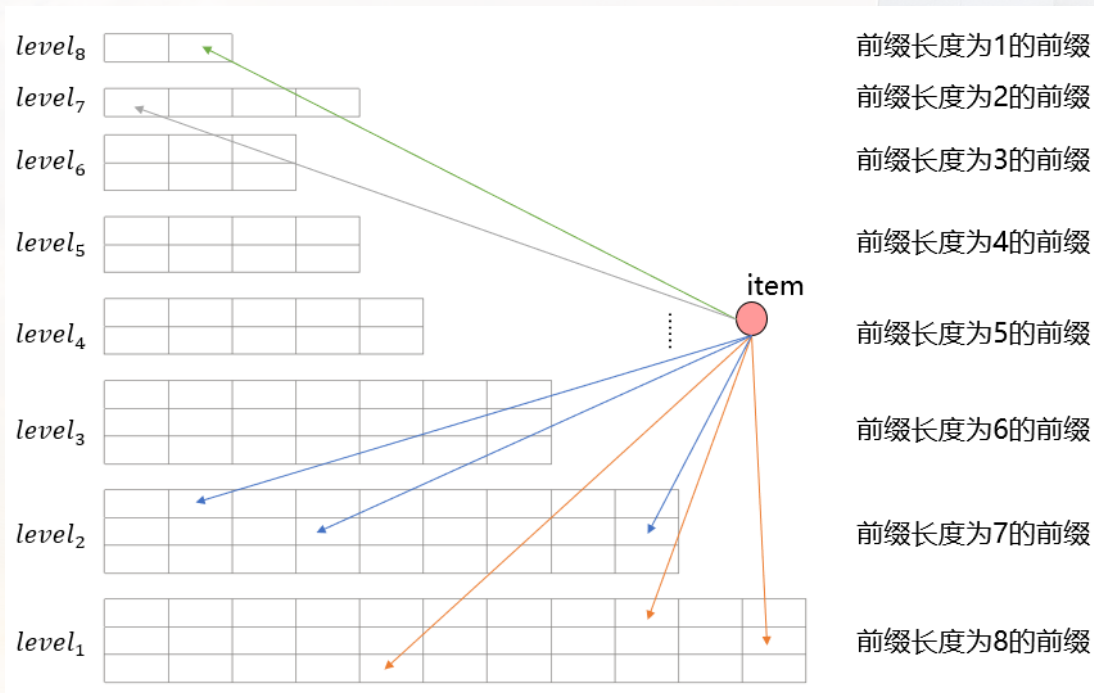
前缀长度为4的前缀: 0000、0001、0010、0011、0100、0101、0110、0111、1000、1001、1010、1011、1100、1101、1110、1111;

.....

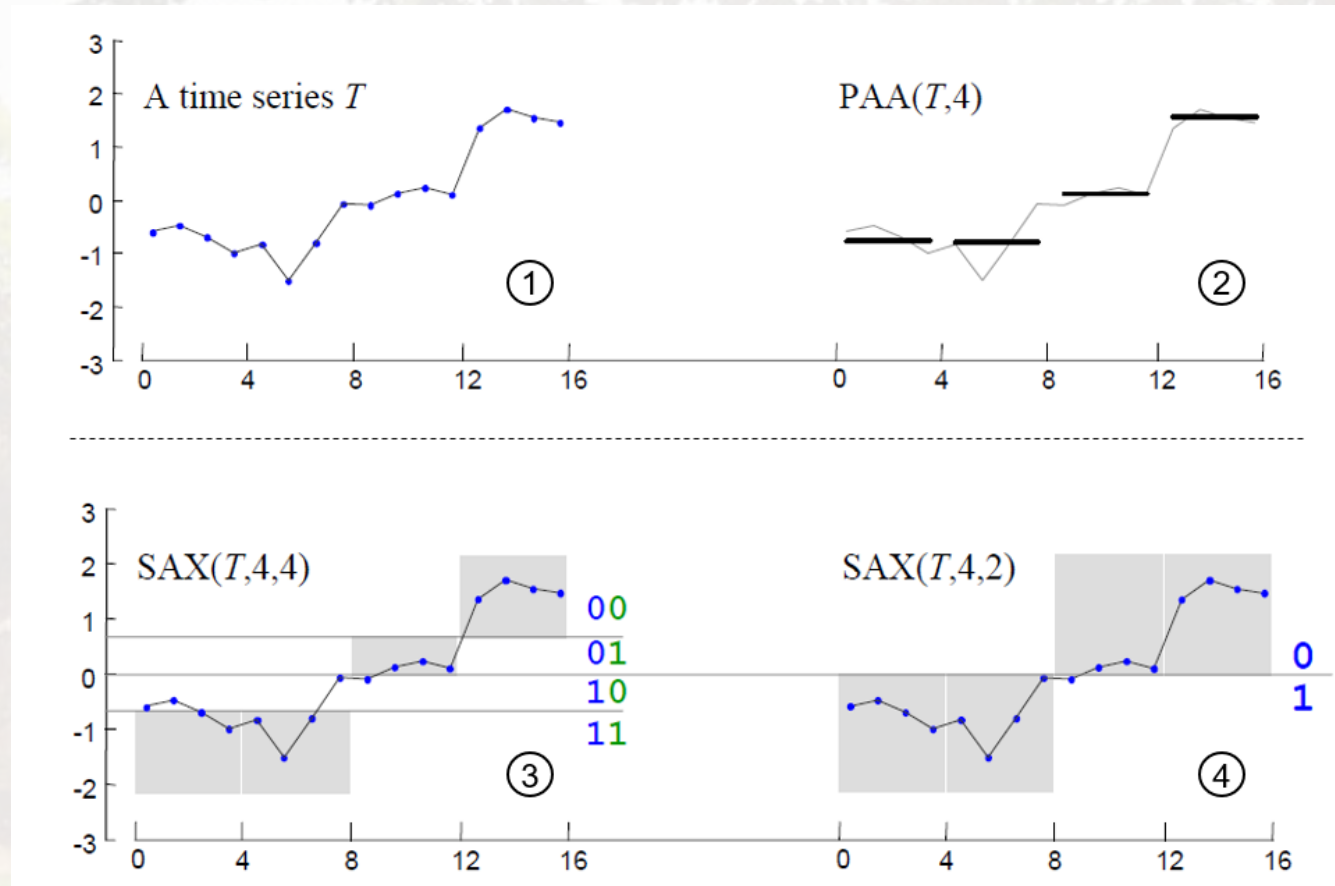


# Multi-Layer CMS

Multi-Layer CMS构建、插入和查询:



符号聚合近似是在分段聚合近似表示方法(**P**iecewise **A**ggregate **A**pproximation, PAA)的基础上, 根据正态分布将时间序列转化为符号化字符。







肆

# 个人思考与总结



南京大学  
NANJING UNIVERSITY

# 研究方向与总结



## 可行的研究方向

- ◆ 开发时序数据库系统 (×) : 开发系统难度大、时间有限; 可以考虑针对现有系统做优化;
- ◆ 研究分布式存储与数据压缩算法 (×) : 分布式存储方案趋向成熟, 压缩算法冷门且不讨好、要天赋;
- ◆ LSM-Tree优化 (√) : 结合一些新场景做特定优化, 如: NVM+LSM, Machine Learning + LSM;
- ◆ 索引 (√) : 优化不同场景下的查询速度;
- ◆ 时序数据挖掘 (√) : 模式匹配、分类或聚类、异常检测、频繁项识别;

## 总结

- ◆ 实际上, 时序数据库三大核心技术 (LSM-Tree、 Data Storage and Compression 、 Index for Big Time Series Data) 研究历史开始时间很早, 但是之前需要处理的数据量并没有现在这么大, 随着第四次工业革命 (AI+IoT) 发展, 之前这些方法可能存在性能瓶颈, 这些方面还有很大的研究价值;
- ◆ 有关时序数据库新的研究领域还有很大挖掘空间。





# 伍

## 参考文献



南京大學  
NANJING UNIVERSITY

## 参考文献



- [1] Schlossnagle T, Sheehy J, McCubbin C. Always-on time-series database: keeping up where there's no way to catch up[J]. **Communications of the ACM**, 2021, 64(7): 50-56.
- [2] Karger D, Sherman A, Berkheimer A, et al. Web caching with consistent hashing[J]. *Computer Networks*, 1999, 31(11-16): 1203-1213.
- [3] Huang X, Wang J, Wong R, et al. Pisa: An index for aggregating big time series data[C]//Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. 2016: 979-988.
- [4] Karger D, Lehman E, Leighton T, et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web[C]//Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. 1997: 654-663.
- [5] Wang C, Huang X, Qiao J, et al. Apache IoTDB: time-series database for internet of things[J]. *Proceedings of the VLDB Endowment*, 2020, 13(12): 2901-2904.



## 参考文献



- [6] Pelkonen T, Franklin S, Teller J, et al. Gorilla: A fast, scalable, in-memory time series database[J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1816-1827.
- [7] Yu X, Peng Y, Li F, et al. Two-level data compression using machine learning in time series database[C]//2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, 2020: 1333-1344.
- [8] McBride B, Reynolds D. Survey of time series database technology[J]. 2020.
- [9] Deri L, Mainardi S, Fusco F. tsdb: A compressed database for time series[C]//International Workshop on Traffic Monitoring and Analysis. Springer, Berlin, Heidelberg, 2012: 143-156.
- [10] O'Neil P, Cheng E, Gawlick D, et al. The log-structured merge-tree (LSM-tree)[J]. Acta Informatica, 1996, 33(4): 351-385.
- [11] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 1-26.

# 参考文献



[12] 刘涛华. 时序数据库 Apache-IoTDB源码解析之文件格式简介 (三) [EB/OL]. <https://blog.csdn.net/a13965645/article/details/104239769>, 2020-02-09.

[13] Ajay Kulkarni, Ryan Booz. What the heck is time-series data (and why do I need a time-series database)? [EB/OL]. <https://blog.timescale.com/blog/what-the-heck-is-time-series-data-and-why-do-i-need-a-time-series-database-dcf3b1b18563>, 2020-12-1.

[14] 史中. 阿里巴巴“数据库侠客”：此行路远，不问归期[EB/OL]. <https://zhuanlan.zhihu.com/p/65295184#comments>, 2019-05-10.

[15] LSM-Tree 与 LevelDB 的原理和实现[EB/LO]. <https://wingsxdu.com/post/database/leveldb/#gsc.tab=0>. 2020-11-05

[16]从InfluxDB到MatrixDB -- 重新定义时序数据库[EB/LO].<https://zhuanlan.zhihu.com/p/406545402>. 2020-09-03.

THE End