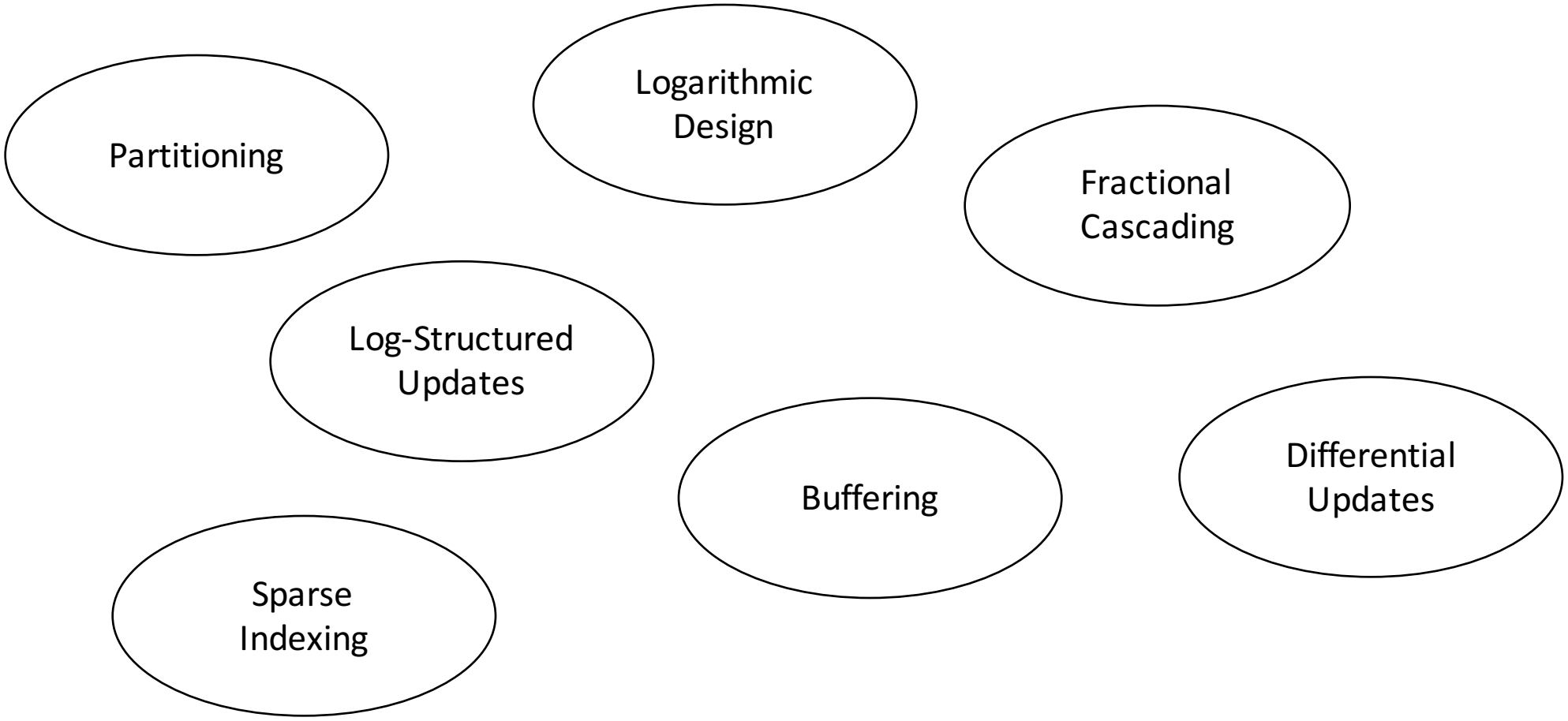
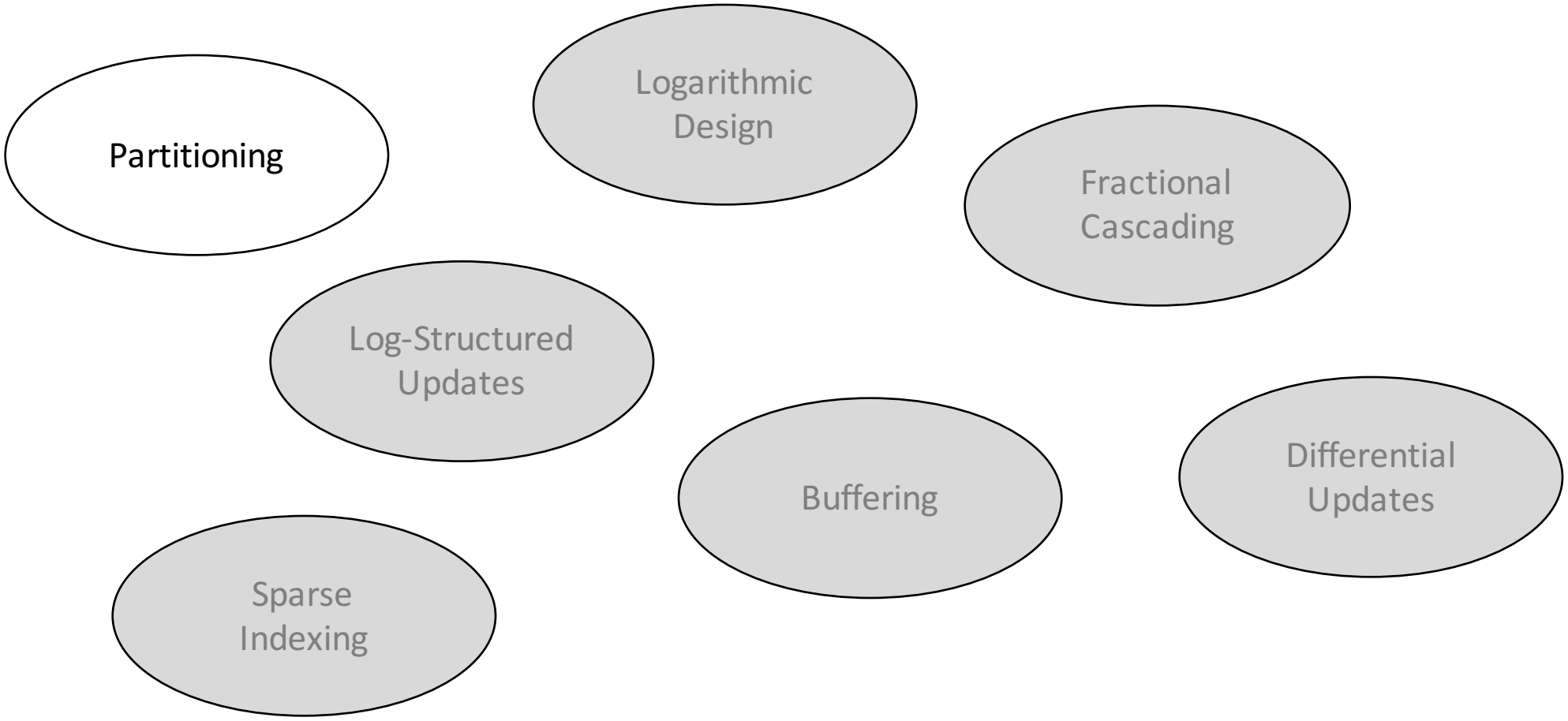
Part B: Design Dimensions



$\mathcal{R}$ $\mathcal{U}$ $\mathcal{M}$

each design decision affects
read / update / memory overheads


Partitioning
Logarithmic Design
Fractional Cascading
Log-Structured Updates
Buffering
Differential Updates
Sparse Indexing

Partitioning

Definition

adds structure to the data

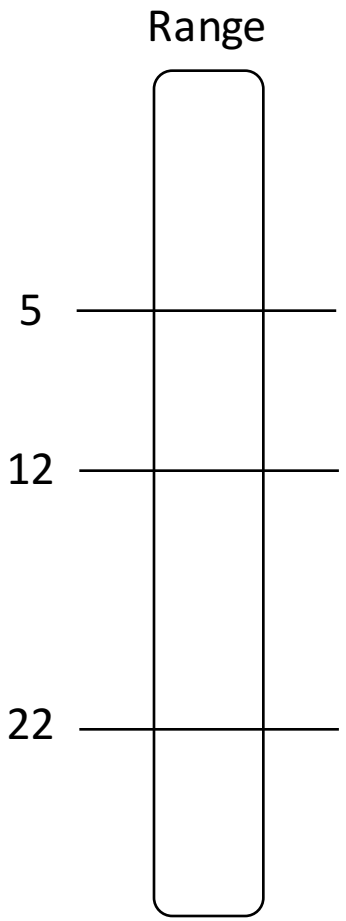
helps reads

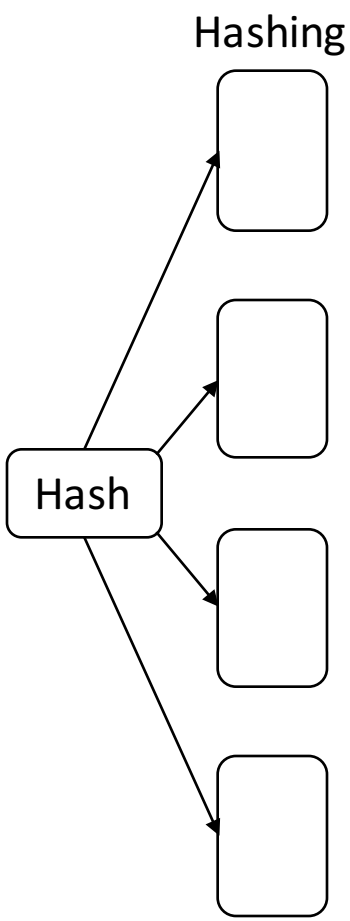
updates are more **expensive** (maintain the structure)

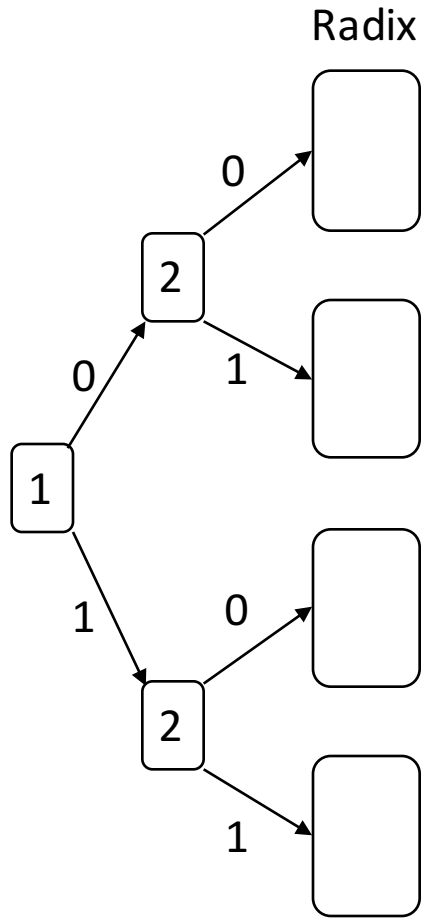
$\mathcal{R}\downarrow\ \mathcal{U}\uparrow\ \mathcal{M}\uparrow$

Partitioning

Feature Implementation

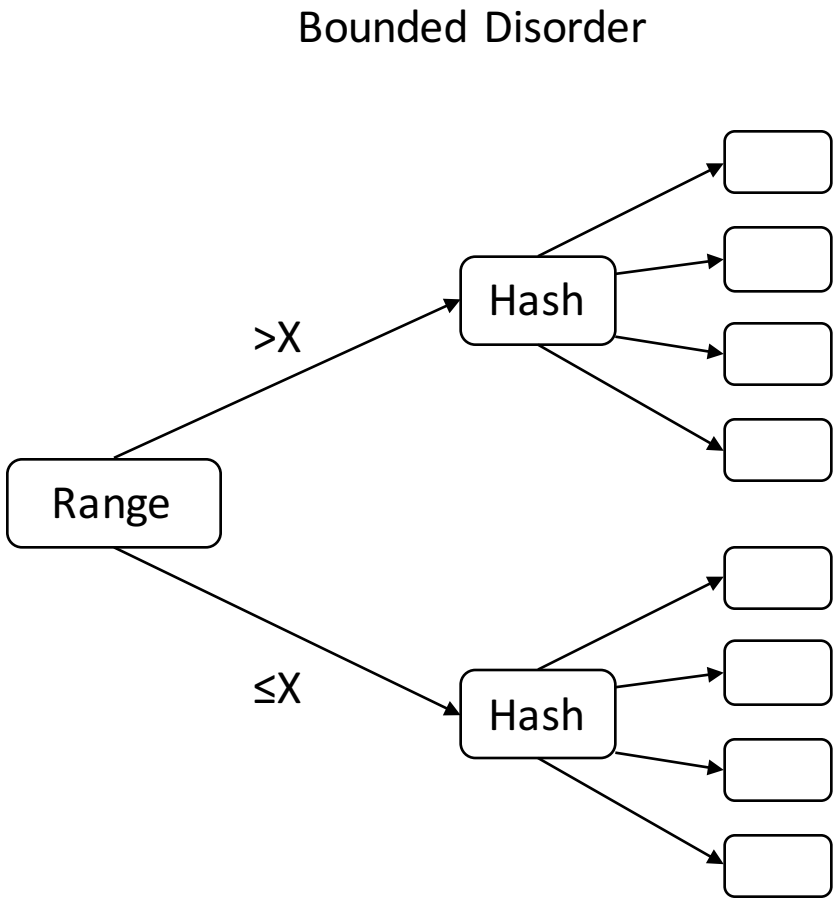


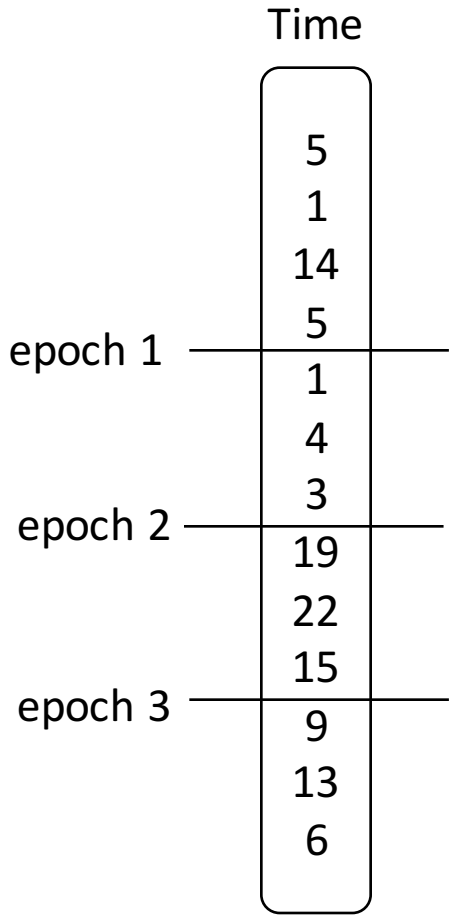




Partitioning

Feature Implementation

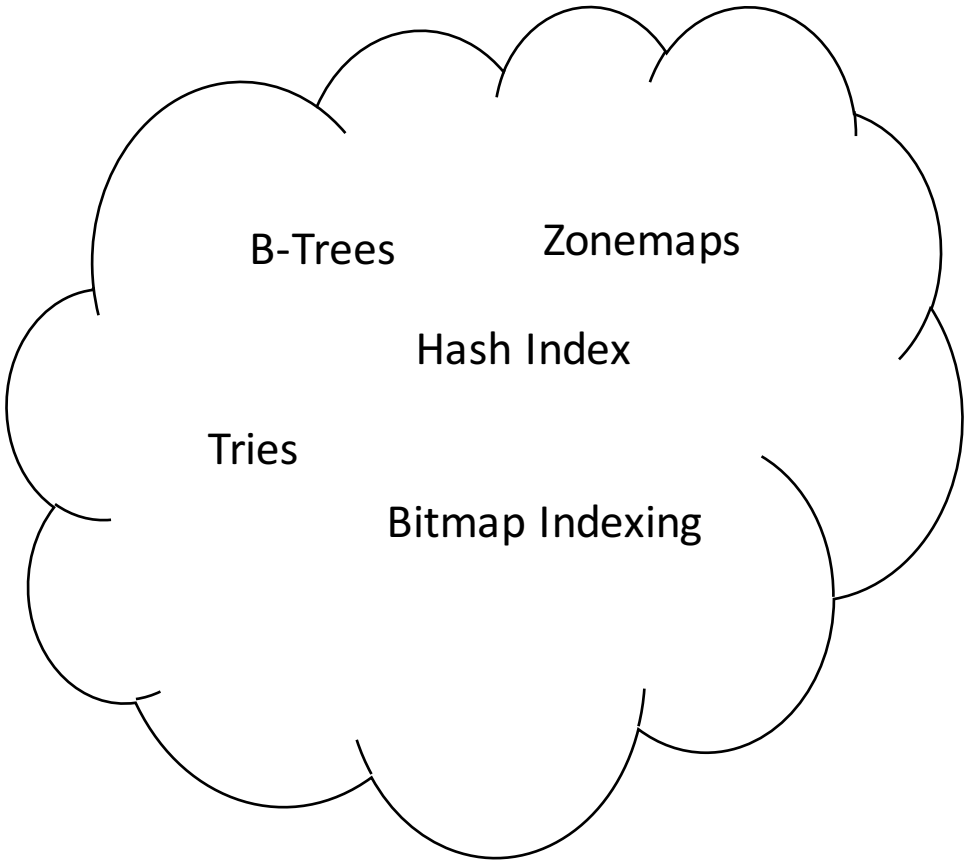




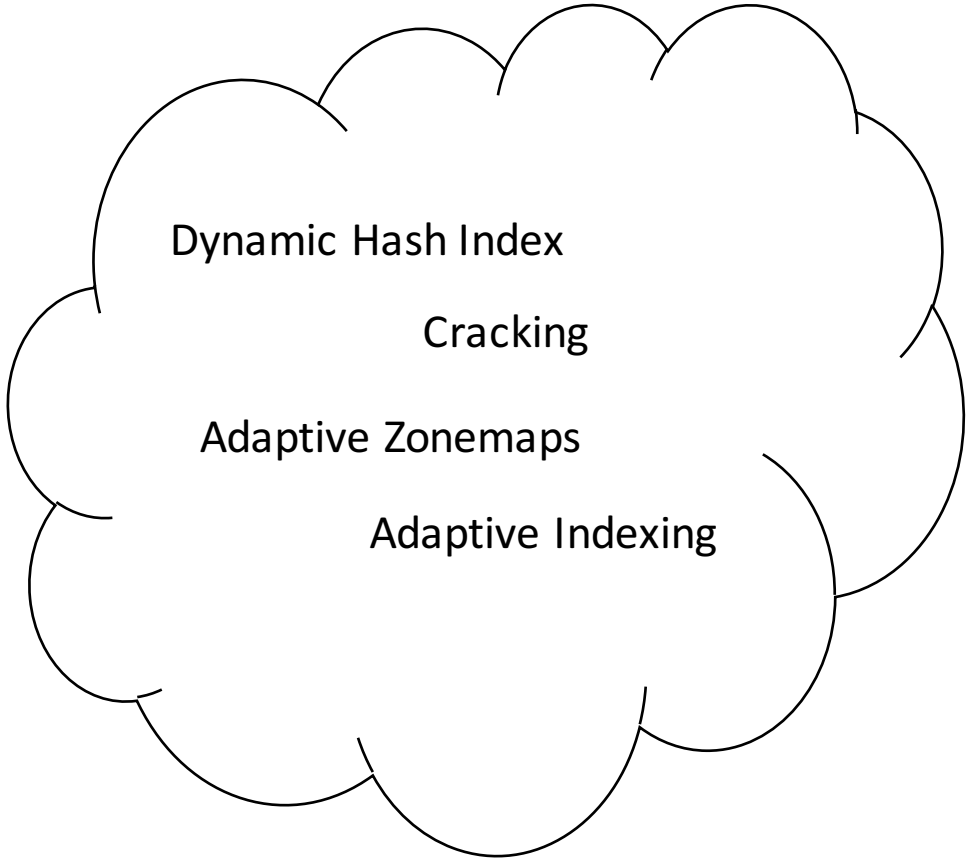
Partitioning

... by example

Static

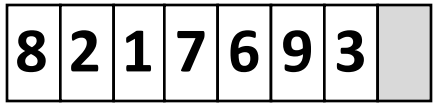


Dynamic



Partitioning: a detailed example

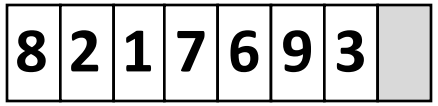
a tight column:



- **reads** have to scan
- no **memory** overhead
- in-place **updates** and efficient inserts

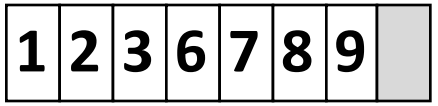
Partitioning: a detailed example

a tight column:



- **reads** have to scan
- no **memory** overhead
- in-place **updates** and efficient inserts

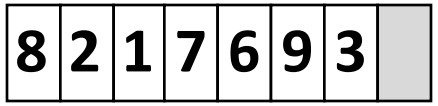
a tight sorted column:



- very efficient **reads** (logarithmic search)
- no **memory** overhead
- **updates** & inserts reorganization

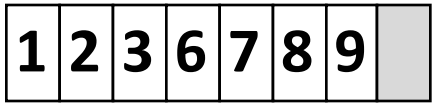
Partitioning: a detailed example

a tight column:



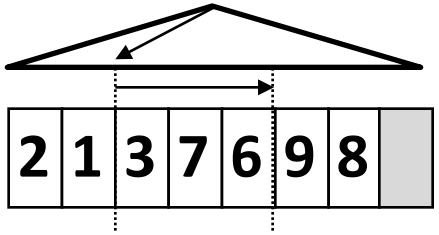
- **reads** have to scan
- no **memory** overhead
- in-place **updates** and efficient inserts

a tight sorted column:



- very efficient **reads** (logarithmic search)
- no **memory** overhead
- **updates** & inserts reorganization

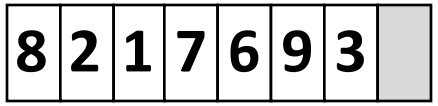
adding clustering:



- efficient **reads**
- small **memory** overhead
- **updates** & inserts: reorganization

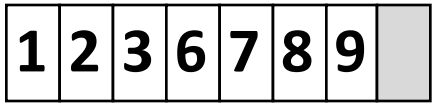
Partitioning: a detailed example

a tight column:



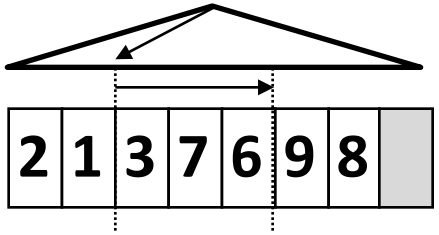
- **reads** have to scan
- no **memory** overhead
- in-place **updates** and efficient inserts

a tight sorted column:



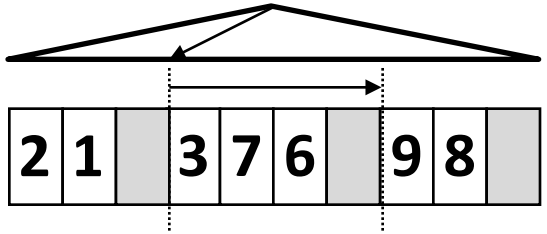
- very efficient **reads** (logarithmic search)
- no **memory** overhead
- **updates** & inserts reorganization

adding clustering:



- efficient **reads**
- small **memory** overhead
- **updates** & inserts: reorganization

… and ghost values:

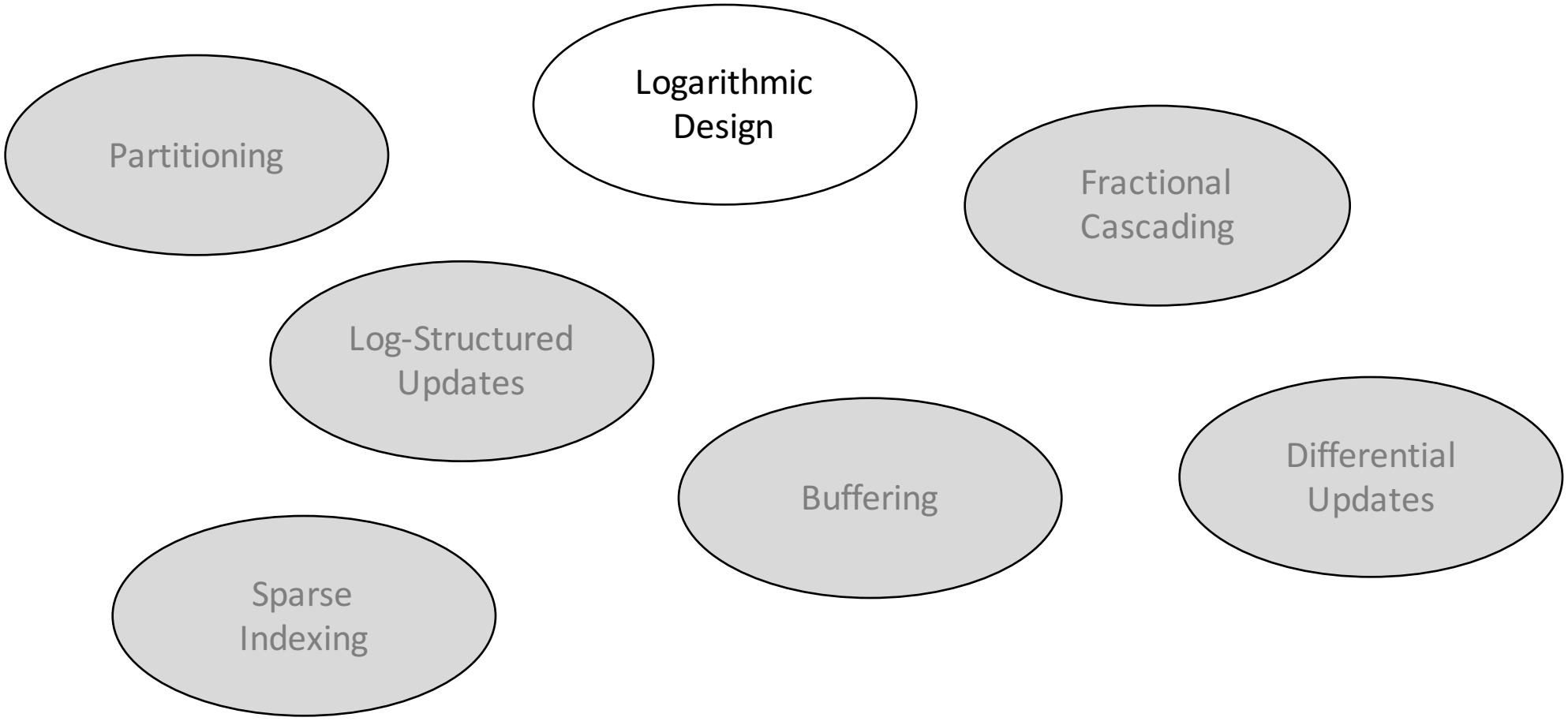


- efficient **reads**
- small **memory** overhead (but increased)
- **updates**: reorganization (but inserts for free)

Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
Base Data & Columns	no, range						
Trees	range, radix, time						
Hashing	hash						
Bitmaps	range, radix						
Differential Files	time, range						

what else is needed to ”come up” with access methods?


Partitioning
Logarithmic Design
Fractional Cascading
Log-Structured Updates
Buffering
Differential Updates
Sparse Indexing

Logarithmic Design

Definition

organize metadata in an exponentially increasing manner

helps reads (logarithmic search)

helps updates (update in place/amortize update cost)

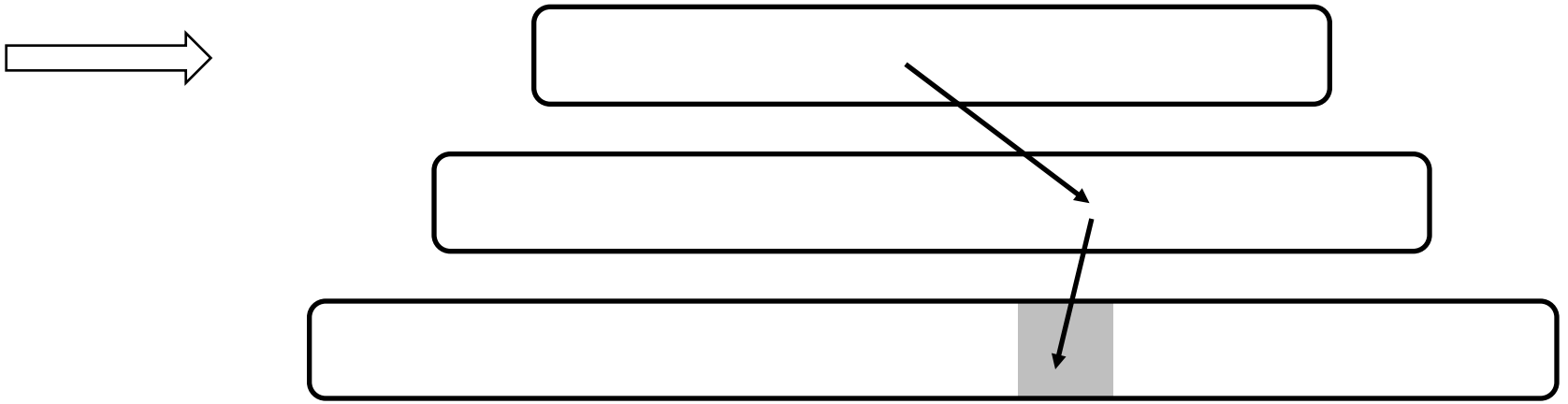
… at the expense of the metadata

$\mathcal{R}\downarrow\ \mathcal{U}\downarrow\ \mathcal{M}\uparrow$

Logarithmic Design

Feature Implementation

connected levels



Traditional Tree Structures

Tries & Variants

B-Trees & Variants

Tree-Trie hybrids

Logarithmic Design

Feature Implementation

independent levels

Update-optimized data organization:

LSM Trees & Variants

MaSM

FD-Tree

Stepped-Merge

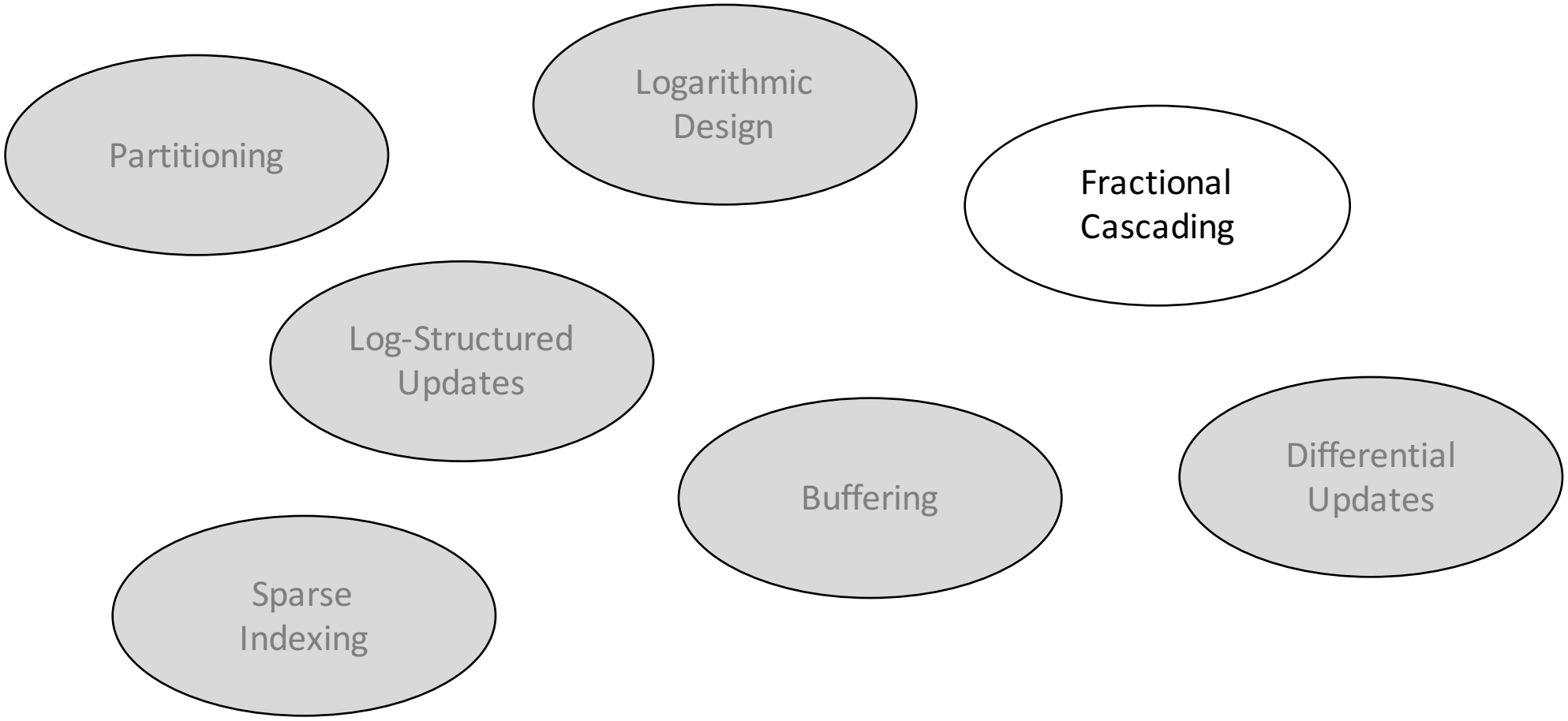
Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
B-Trees & Variants [1]	range	✔ (naturally)					
Tries & Variants [2]	radix	✔ (naturally)					
LSM-Trees & Variants [3]	time	✔ (naturally)					

[1] B-Trees (Acta Inf. 1972), B-Tree techniques (FNT 2011)

[2] Tries (CACM 1960), PATRICIA (JACM 1968), ART (ICDE 2013)

[3] LSM-Tree (Acta Inf. 1996), VT-Trees (FAST 2013), LSM-Trie (ATC 2015)


Partitioning
Logarithmic Design
Fractional Cascading
Log-Structured Updates
Buffering
Differential Updates
Sparse Indexing

Fractional Cascading

Definition

adds metadata for efficient accessing/searching

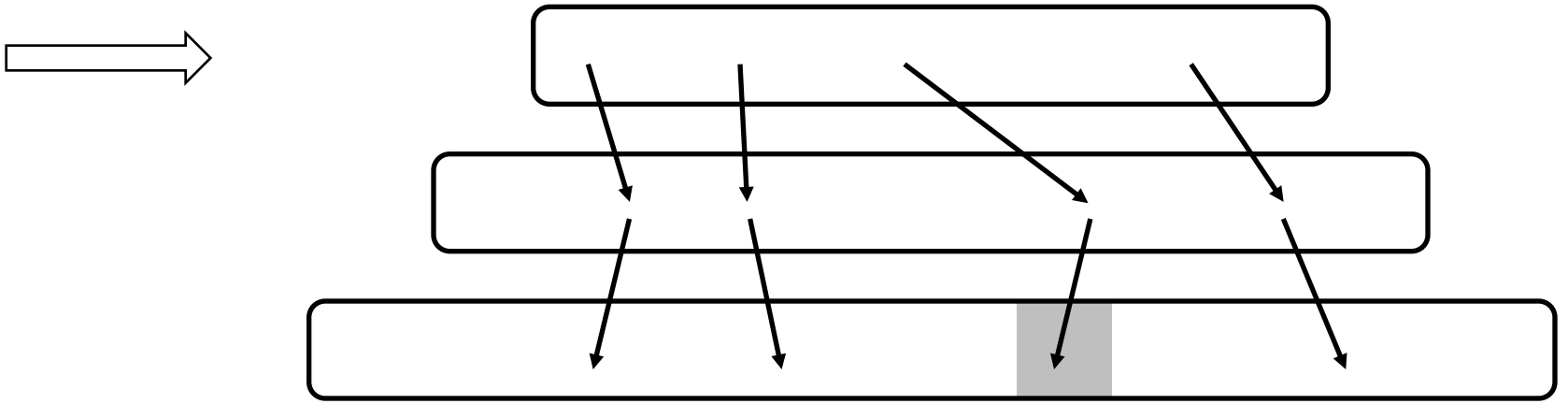
- pointers between different ”levels” of access methods
- easy navigation to the ”corresponding” partitions
- need maintenance on updates

$\mathcal{R}\downarrow\ \mathcal{U}\uparrow\ \mathcal{M}\uparrow$

Fractional Cascading

Feature Implementation

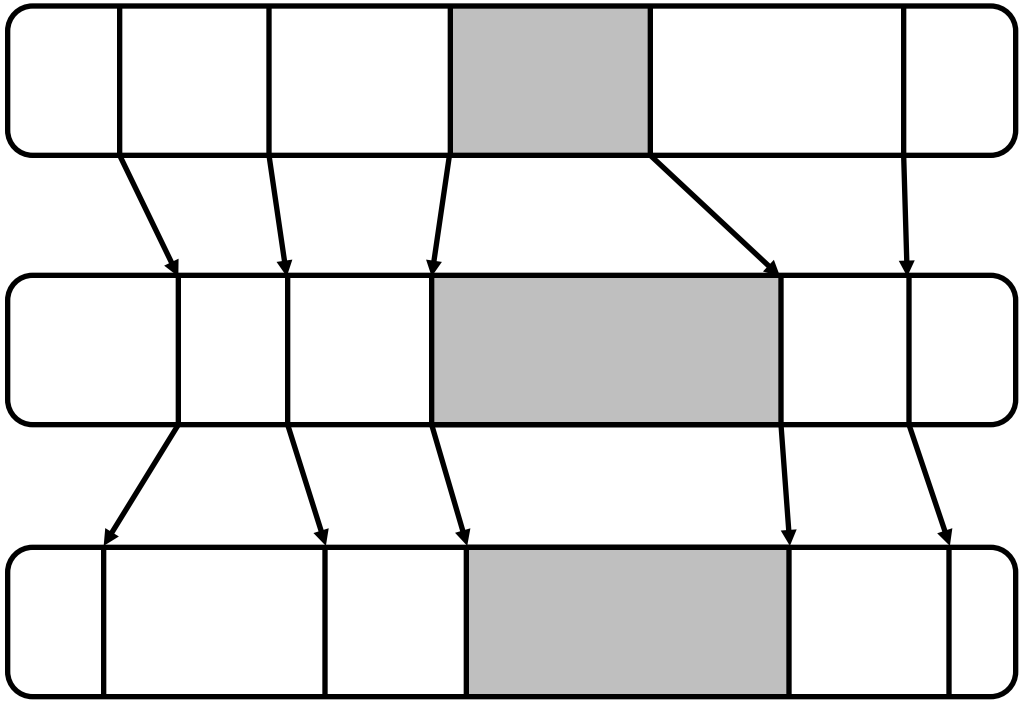
Naturally exists in connected levels!



Fractional Cascading

Feature Implementation

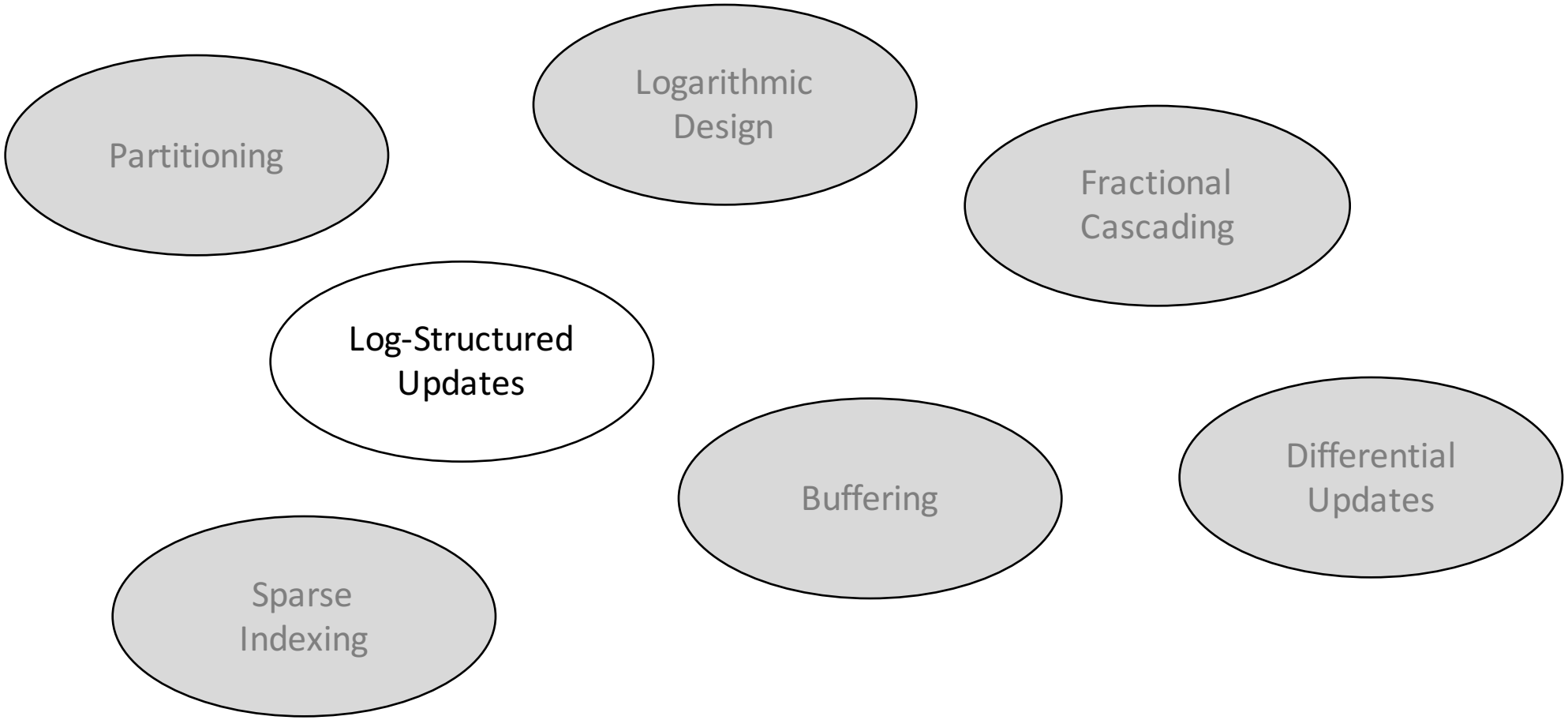
An additional layer of metadata for independent levels



Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
B-Trees & Variants	range	✔	✔				
Tries & Variants	radix	✔	✔ (naturally)				
LSM-Trees & Variants	time	✔	[1]				

[1] FD-Tree (PVLDB 2010), bLSM (SIGMOD 2012)


Partitioning
Logarithmic Design
Fractional Cascading
Log-Structured Updates
Buffering
Differential Updates
Sparse Indexing

Log-Structured Updates

Definition

apply and organize updates by

appending instead of in-place updates

reads need to merge updates with old data

Log-Structured Updates

Feature Implementation

Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
B-Trees & Variants	range	✔	≈	[1]			
Tries & Variants	radix	✔	✔				
LSM-Trees & Variants	time	✔	≈	✔ (naturally)			
Differential Files	time, range			[2]			

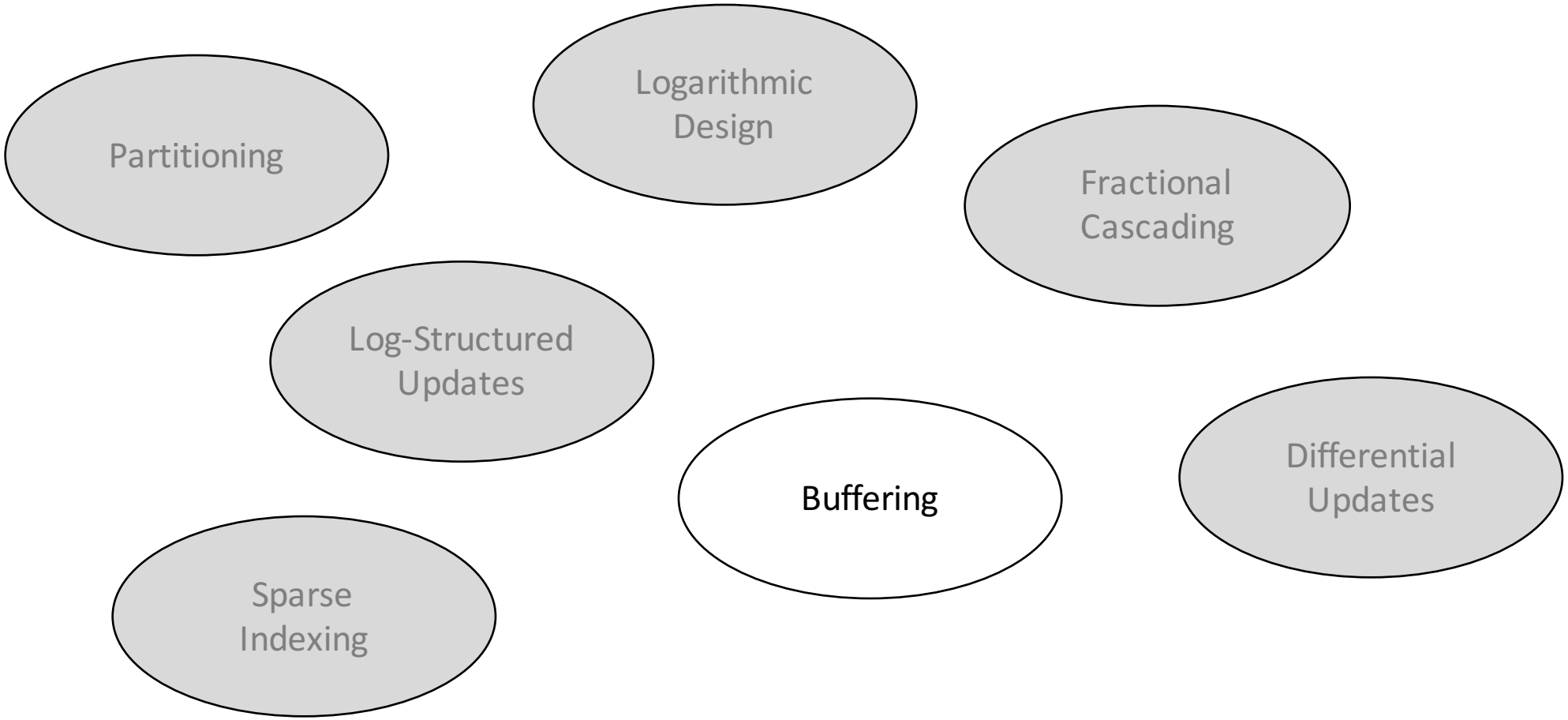
fractional cascading and log-structured updates? *challenging to combine efficiently*

log-structured updates with radix/hash partitioning? *open research question!*

[1] Storage/Memory-Aware Trees: Bw-Tree (ICDE 2013), µ-Tree (EMSOFT 2007), IPLB$^+$-Tree (JISE 2011)

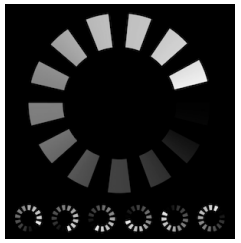
[2] Differential Files (TODS 1976) & Variants: Stepped-Merge (VLDB 1997), MaSM (TODS 2015)

✔ integral part of design ≈ optional design decision


Partitioning
Logarithmic Design
Fractional Cascading
Log-Structured Updates
Buffering
Differential Updates
Sparse Indexing

Buffering

Definition



explicitly buffer recently read / updated objects / requests

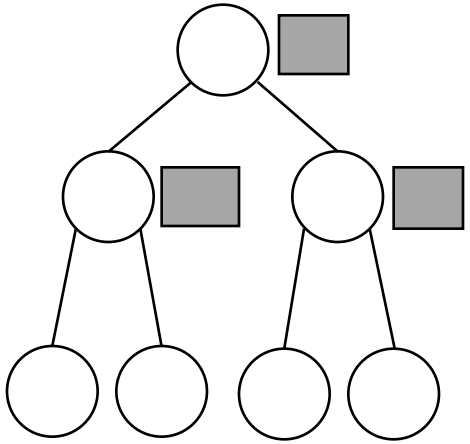
direct tradeoff between **memory** and **read/update** performance

$\mathcal{R}\downarrow\ \mathcal{U}-\ \mathcal{M}\uparrow$ or $\mathcal{R}-\ \mathcal{U}\downarrow\ \mathcal{M}\uparrow$

Buffering

Feature Implementation

Buffering Recent Reads/Updates/Requests



Buffering Recent Updates

Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
B-Trees & Variants	range	✔	✔	≈	reads [1] updates [2] requests [3]		
Tries & Variants	radix	✔	✔				
LSM-Trees & Variants	time	✔	≈	✔	✔		
Differential Files	time, range			✔	updates [4]		

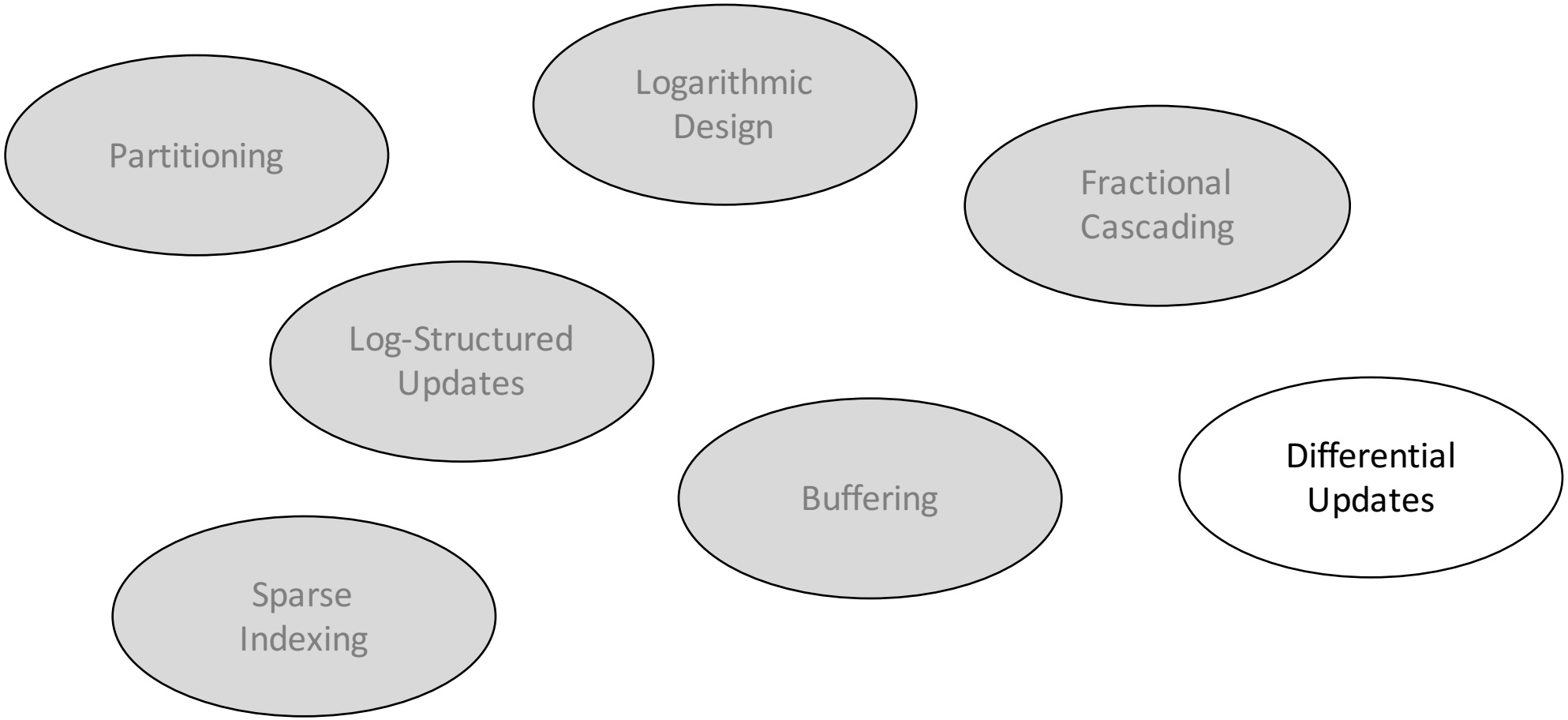
[1] Trees with buffered reads: Fractal Tree, BRT, COLA (SPAA 2007), LA-Tree (VLDB 2009), ADS (SIGMOD 2014)

[2] Trees with buffered updates: IPLB^{+}-Tree (JISE 2011), LA-Tree (VLDB 2009), PDT (SIGMOD 2010)

[3] Trees with buffered requests: PIO B-Tree (VLDB 2011), Virtual Nodes (VLDB 2003)

[4] Differential files with buffered updates: Stepped-Merge (VLDB 1997), MaSM (TODS 2015)

✔ integral part of design ≈ optional design decision


Partitioning
Logarithmic Design
Fractional Cascading
Log-Structured Updates
Buffering
Differential Updates
Sparse Indexing

Differential Updates

Definition

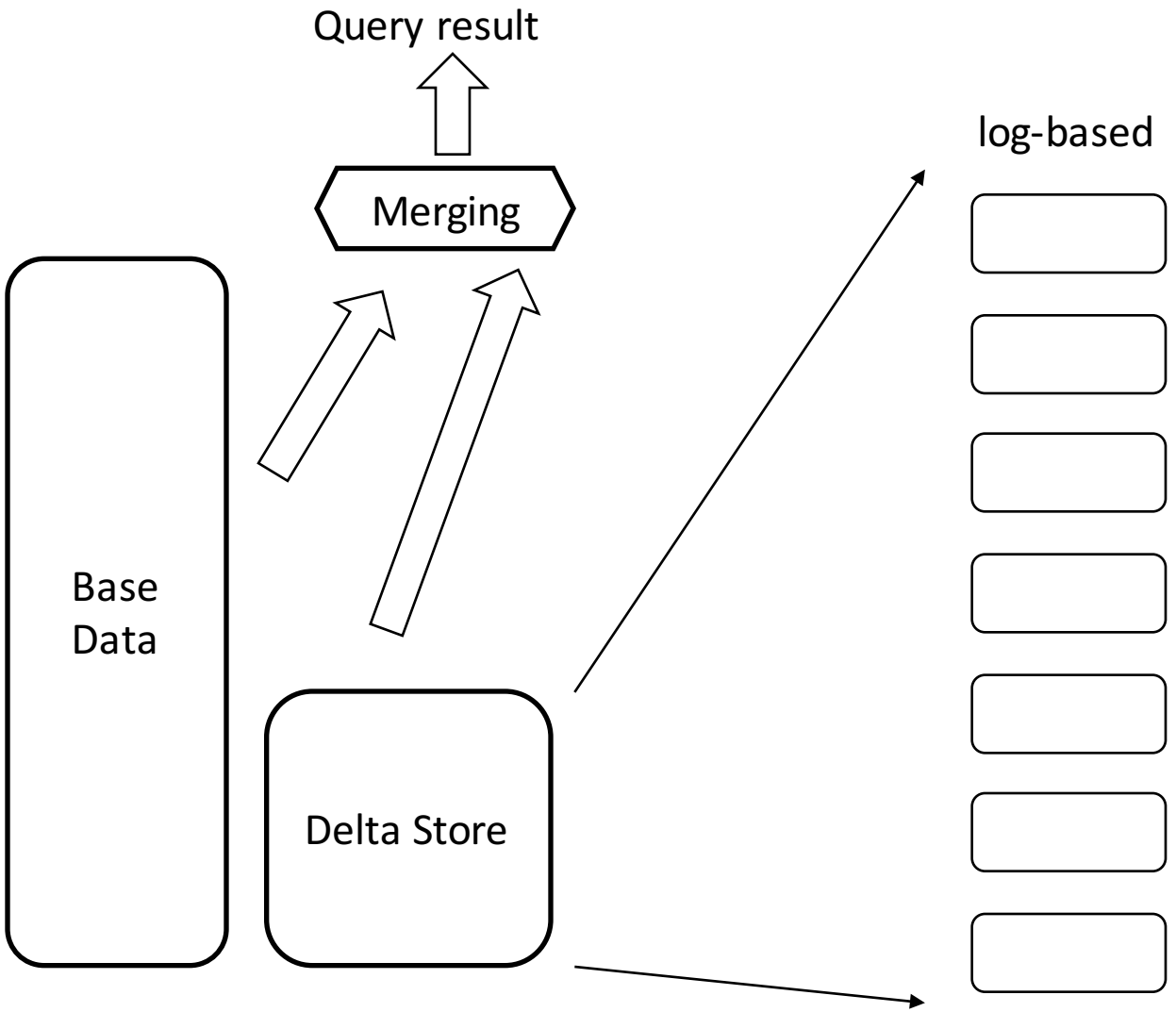
next step for **log-structure**

only deltas are stored in order to **minimize** storage overheads

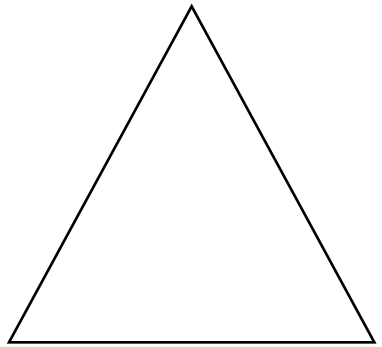
$\mathcal{R}\uparrow\ \mathcal{U}\downarrow\ \mathcal{M}\downarrow$

Differential Updates

Feature Implementation



tree-based



stores data physical location info

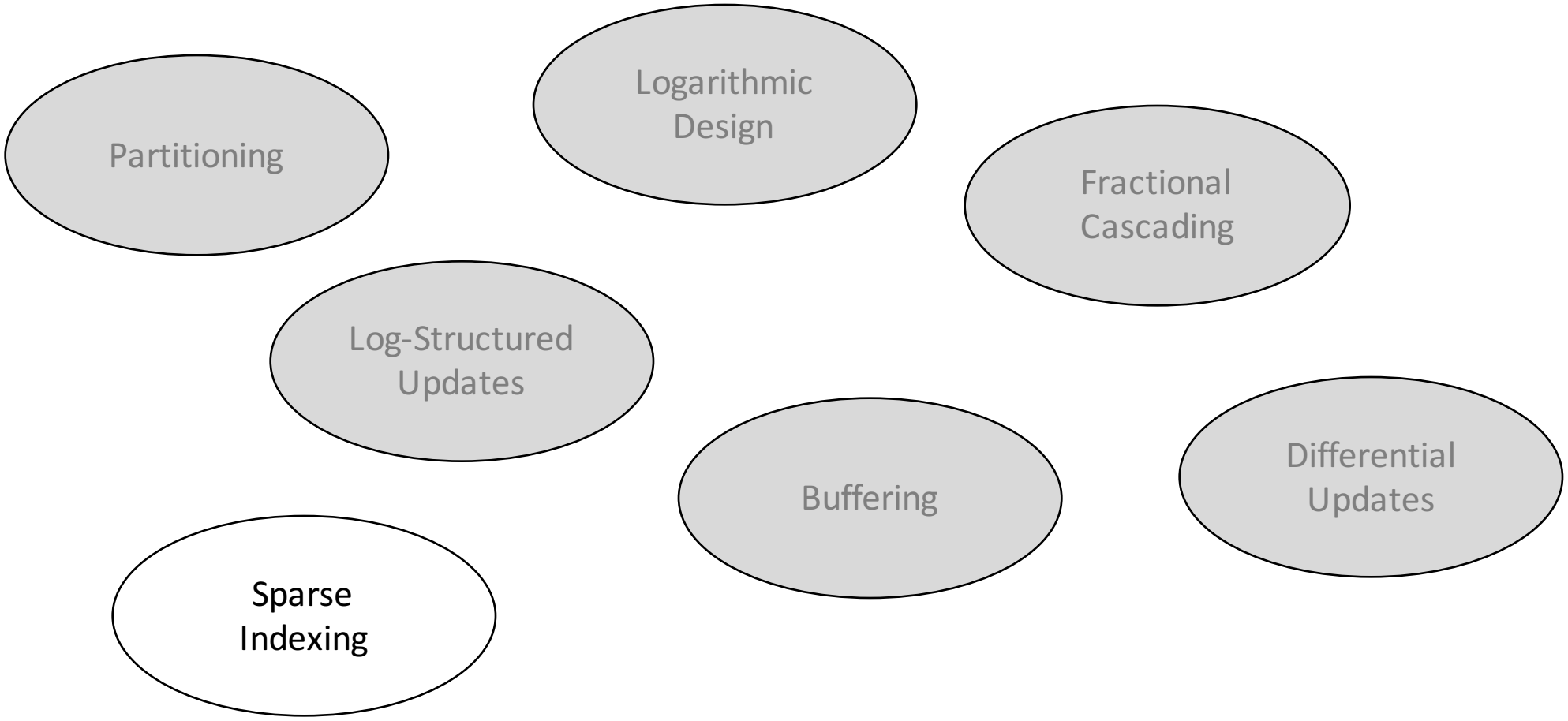
Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
B-Trees & Variants	range	✔	≈	≈	≈	[1]	
Tries & Variants	radix	✔	✔				
LSM-Trees & Variants	time	✔	≈	✔	✔		
Differential Files	time, range			✔	✔	[2]	

[1] PDT (SIGMOD 2010), IPLB$^+$-Tree (JISE 2011), LA-Tree (VLDB 2009), PBT (CIDR 2003)

[2] Differential Files (TODS 1976), MaSM (TODS 2015)

✔ integral part of design ≈ optional design decision


Partitioning
Logarithmic Design
Fractional Cascading
Log-Structured Updates
Buffering
Differential Updates
Sparse Indexing

Sparse Indexing

Definition

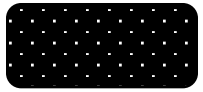
light-weight indexing that allows for skipping unnecessary data

$\mathcal{R}\downarrow\ \mathcal{U}?\ \mathcal{M}\uparrow$

Sparse Indexing

Feature Implementation

membership tests



Data

sparse range partitioning

Data

bitwise representation

Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
B-Trees & Variants	range	✔	✔	≈	≈	≈	[1]
Tries & Variants	radix	✔	✔				
LSM-Trees & Variants	time	✔	≈	✔	✔		
Differential Files	time, range			✔	✔	✔	
Membership Tests	—						[2]
Zonemaps & Variants	range						[3]
Bitmaps & Variants	range						[4]

[1] BF-Tree (VLDB 2014)

[2] Bloom filters (CACM 1970), Quotient Filters (VLDB 2011), Cuckoo Filters (CoNEXT 2014)

[3] Zonemaps (IBM Redbook 2011, VLDB 2013, SIGMOD 2013, SIGMOD 2014), Column Imprints (SIGMOD 2013)

[4] Bit Transposed Files (VLDB 1985), Bitmap Indexing (HPTS 1987, SIGMOD 1997, 1998, 1999)

✔ integral part of design ≈ optional design decision

Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
B-Trees & Variants	range	✔	✔	≈	≈	≈	≈
Tries & Variants	radix	✔	✔				
LSM-Trees & Variants	time	✔	≈	✔	✔		
Differential Files	time, range			✔	✔	✔	
Membership Tests	—						✔
Zonemaps &Variants	range						✔
Bitmaps & Variants	range						✔
Hashing	hash						
Base Data & Columns	no, range						

✔ integral part of design ≈ optional design decision

Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
B-Trees & Variants	range	✔	✔	≈	≈	≈	≈
Tries & Variants	radix	✔	✔	?	?	?	?
LSM-Trees & Variants	time	✔	≈	✔	✔	?	?
Differential Files	time, range	?	?	✔	✔	✔	?
Membership Tests	—	?	?	?	?	?	✔
Zonemaps &Variants	range	?	?	?	?	?	✔
Bitmaps & Variants	range	?	?	?	?	?	✔
Hashing	hash	?	?	?	?	?	?
Base Data & Columns	no, range	?	?	?	?	?	?

Open research questions!

✔ integral part of design ≈ optional design decision

Design Opportunity

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*
B-Trees & Variants	range	✔	✔	≈	≈	≈	≈
Tries & Variants	radix	✔	✔	?	?	?	?
LSM-Trees & Variants	time	✔	≈	✔	✔	?	?
Differential Files	time, range	?	?	✔	✔	✔	?
Membership Tests	—	?	?	?	?	?	✔
Zonemaps &Variants	range	?	?	?	?	?	✔
Bitmaps & Variants	range	?	?	?	?	?	✔
Hashing	hash	?	?	?	?	?	?
Base Data & Columns	no, range	?	?	?	?	?	?

Open research questions!

✔ integral part of design ≈ optional design decision

Hardware-Aware Access Methods

Read vs. Write latency Impact of Read vs. Write Variable latency (due to data placement)

How?

Use design elements to *match* hardware properties!

Examples

Partitioning : ensure local (faster) accesses

Log-Structure/Differential Updates : storage friendly updates

Buffering : exploit additional memory

Workload-Driven Access Methods

workload-driven ***orthogonal*** to design elements

a way to ***incrementally*** reach the goal of a design element

can be a design element!

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*	*Adaptivity*
B-Trees & Variants	range	✔	✔	≈	≈	≈	≈	?
Tries & Variants	radix	✔	✔	?	?	?	?	?
LSM-Trees & Variants	time	✔	≈	✔	✔	?	?	?
Differential Files	time/range	?	?	✔	✔	✔	?	?
Membership Tests	—	?	?	?	?	?	✔	?
Zonemaps &Variants	range	?	?	?	?	?	✔	?
Bitmaps & Variants	range	?	?	?	?	?	✔	?
Hashing	hash	?	?	?	?	?	?	?
Base Data & Columns	no, range	?	?	?	?	?	?	?

✔ integral part of design ≈ optional design decision

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*	*Adaptivity*
B-Trees & Variants	range	✔	✔	≈	≈	≈	≈	**[1]**
Tries & Variants	radix	✔	✔	**?**	**?**	**?**	**?**	**?**
LSM-Trees & Variants	time	✔	≈	✔	✔	**?**	**?**	**?**
Differential Files	time/range	**?**	**?**	✔	✔	✔	**?**	**?**
Membership Tests	—	**?**	**?**	**?**	**?**	**?**	✔	**?**
Zonemaps &Variants	range	**?**	**?**	**?**	**?**	**?**	✔	**?**
Bitmaps & Variants	range	**?**	**?**	**?**	**?**	**?**	✔	**?**
Hashing	hash	**?**	**?**	**?**	**?**	**?**	**?**	**?**
Base Data & Columns	no, range	**?**	**?**	**?**	**?**	**?**	**?**	**[2]**

[1] Adaptive Indexing (VLDB 2011)

[2] Database Cracking (CIDR 2007)

✔ integral part of design ≈ optional design decision

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*	*Adaptivity*
B-Trees & Variants	range	✔	✔	≈	≈	≈	≈	**[1]**
Tries & Variants	radix	✔	✔	?	?	?	?	?
LSM-Trees & Variants	time	✔	≈	✔	✔	?	?	?
Differential Files	time/range	?	?	✔	✔	✔	?	?
Membership Tests	—	?	?	?	?	?	✔	?
Zonemaps &Variants	range	?	?	?	?	?	✔	?
Bitmaps & Variants	range	?	?	?	✔	✔	✔	**[3]**
Hashing	hash	?	?	?	?	?	?	?
Base Data & Columns	no, range	?	?	?	?	?	?	**[2]**

[1] Adaptive Indexing (VLDB 2011)

[2] Database Cracking (CIDR 2007)

[3] UpBit: Updatable Bitmap Indexing (SIGMOD 2016)

✔ integral part of design ≈ optional design decision

	Partitioning	*Logarithmic Design*	*Fractional Cascading*	*Log-Structured*	*Buffering*	*Differential Updates*	*Sparse Indexing*	*Adaptivity*
B-Trees & Variants	range	✔	✔	≈	≈	≈	≈	≈
Tries & Variants	radix	✔	✔	?	?	?	?	?
LSM-Trees & Variants	time	✔	≈	✔	✔	?	?	?
Differential Files	time/range	?	?	✔	✔	✔	?	?
Membership Tests	—	?	?	?	?	?	✔	?
Zonemaps &Variants	range	?	?	?	?	?	✔	?
Bitmaps & Variants	range	?	?	?	≈	≈	✔	≈
Hashing	hash	?	?	?	?	?	?	?
Base Data & Columns	no, range	?	?	?	?	?	?	≈

map existing designs – find commonalities

propose new combinations and predict their behavior

tune existing access methods (altering/adding individual design elements)

✔ integral part of design ≈ optional design decision

http://daslab.seas.harvard.edu/

thank you!