

Design Tradeoffs of Data Access Methods

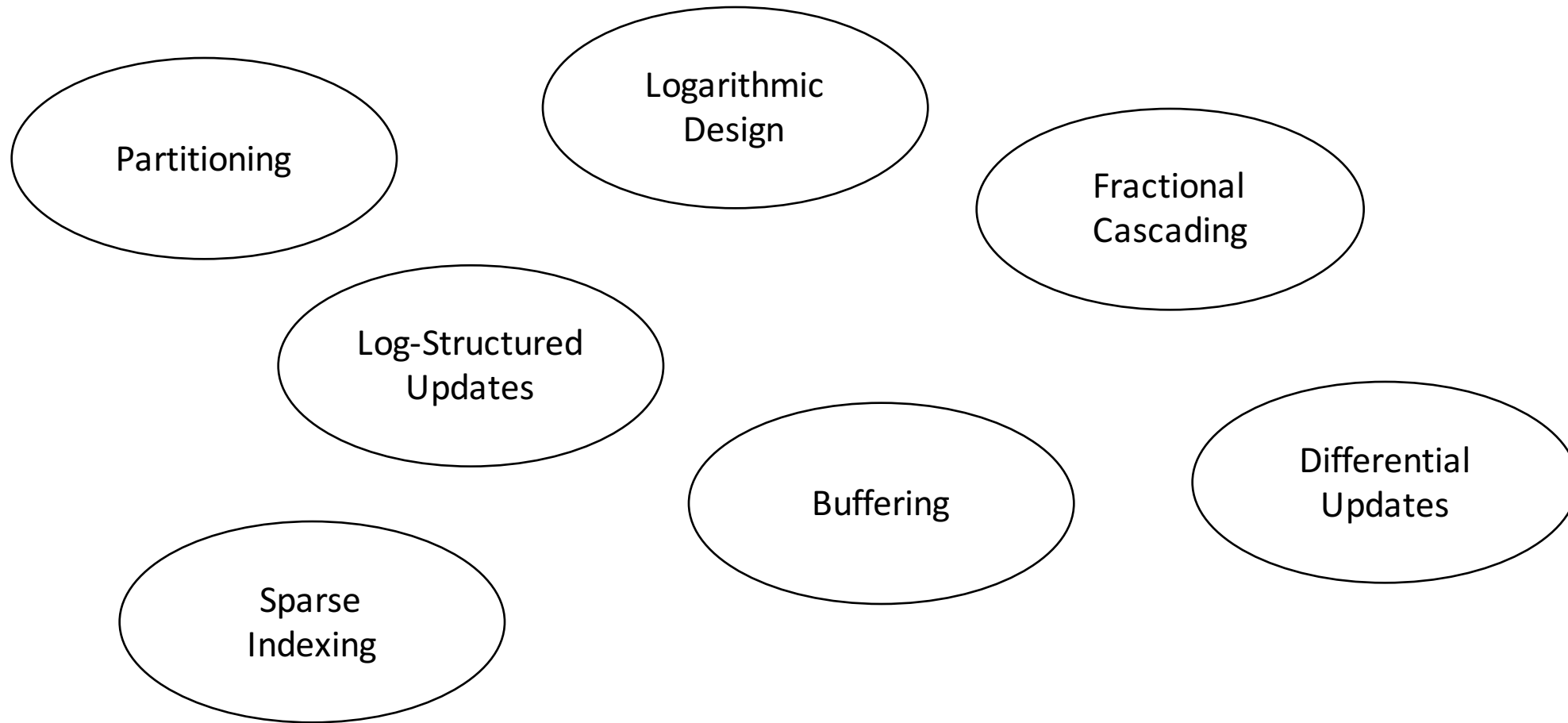
Manos Athanassoulis

Stratos Idreos

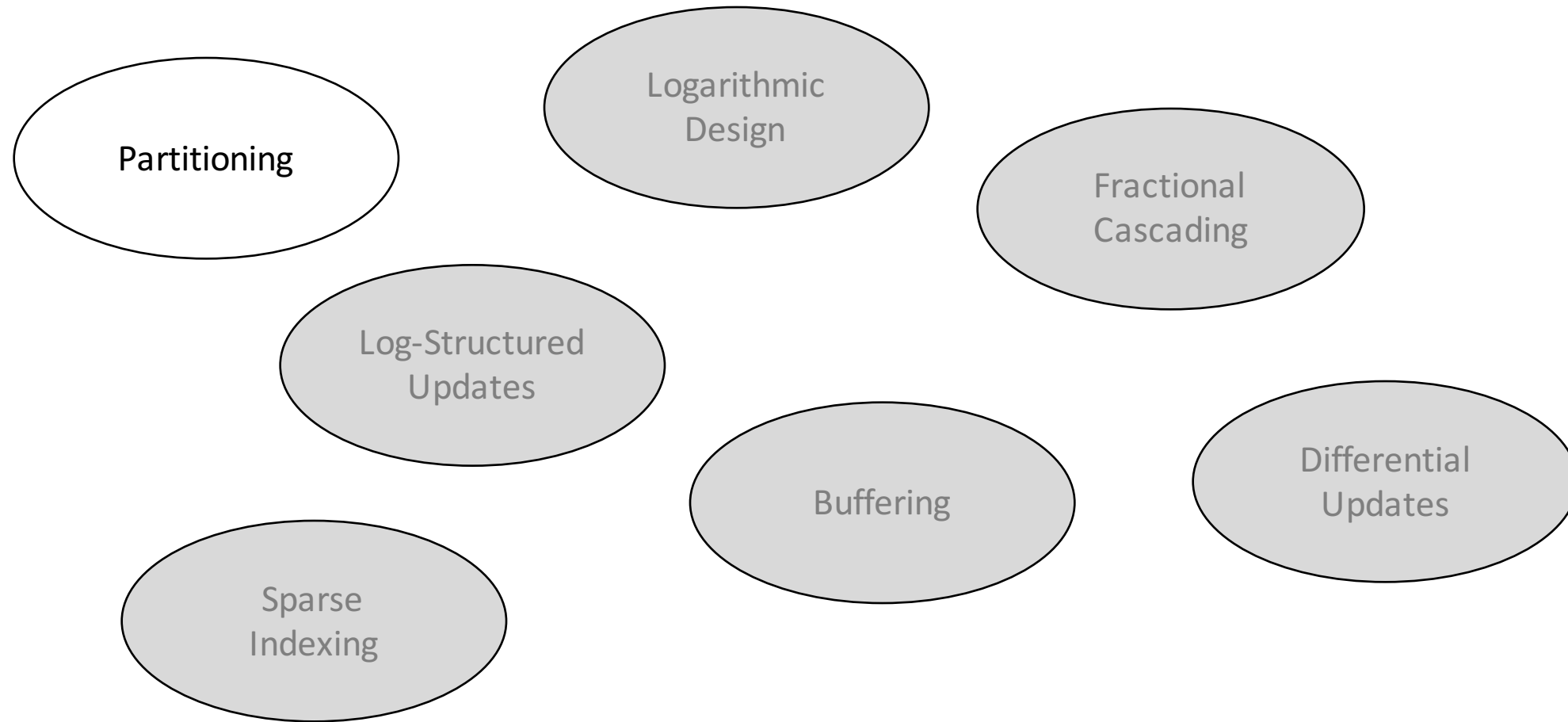
Harvard University



Part B: Design Dimensions

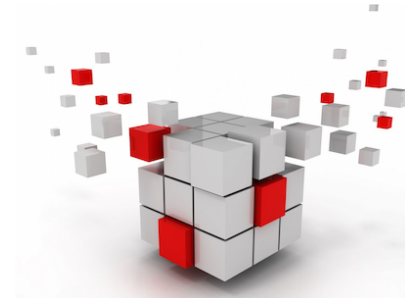
 $\mathcal{R} \quad \mathcal{U} \quad \mathcal{M}$

each design decision affects
read / update / memory overheads



Partitioning

Definition



adds structure to the data

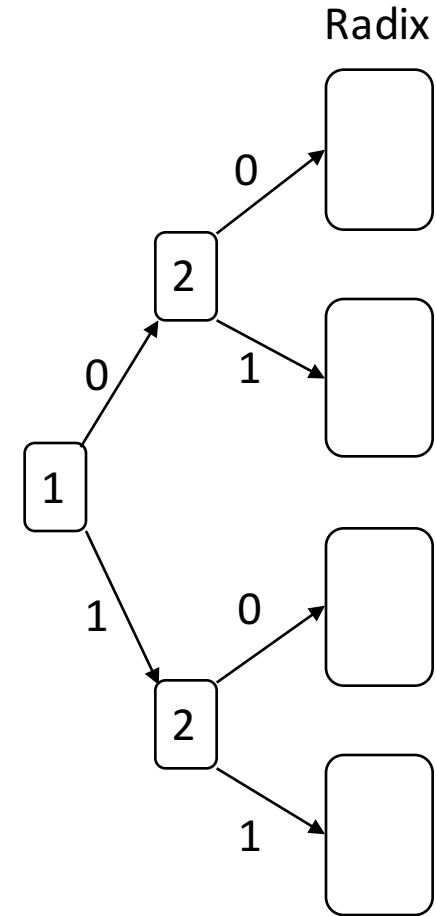
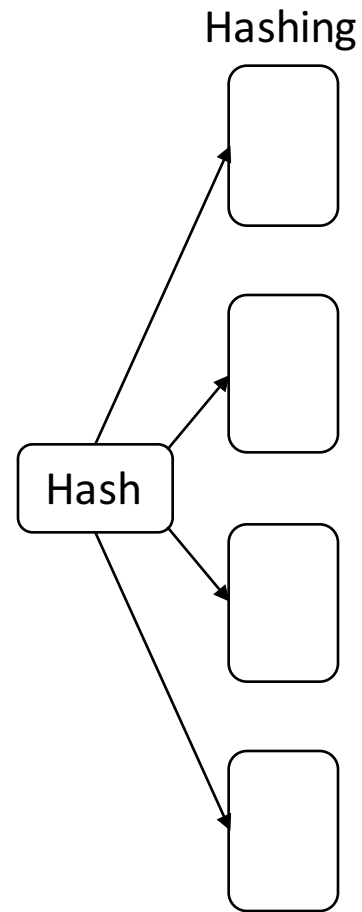
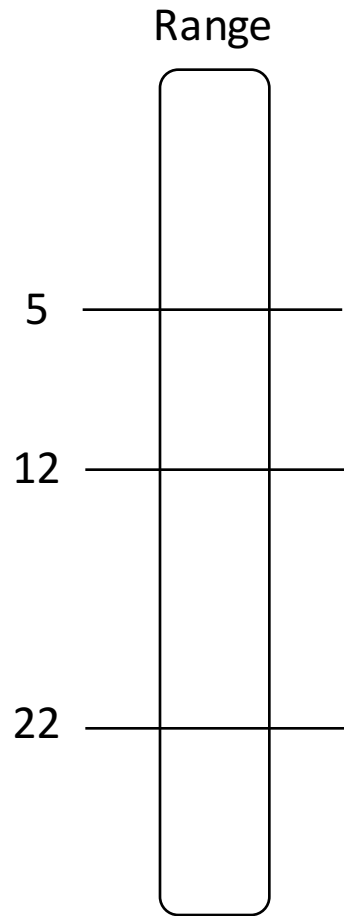
helps reads

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

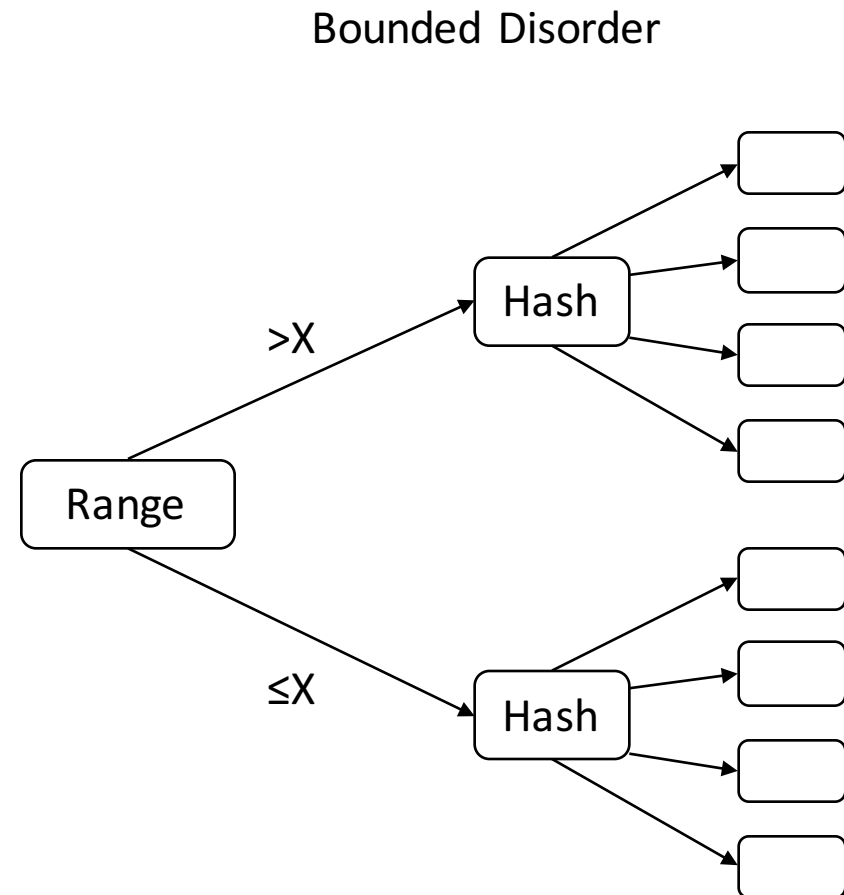
$\mathcal{R} \downarrow \mathcal{U} \uparrow \mathcal{M} \uparrow$

Partitioning

Feature Implementation



Partitioning



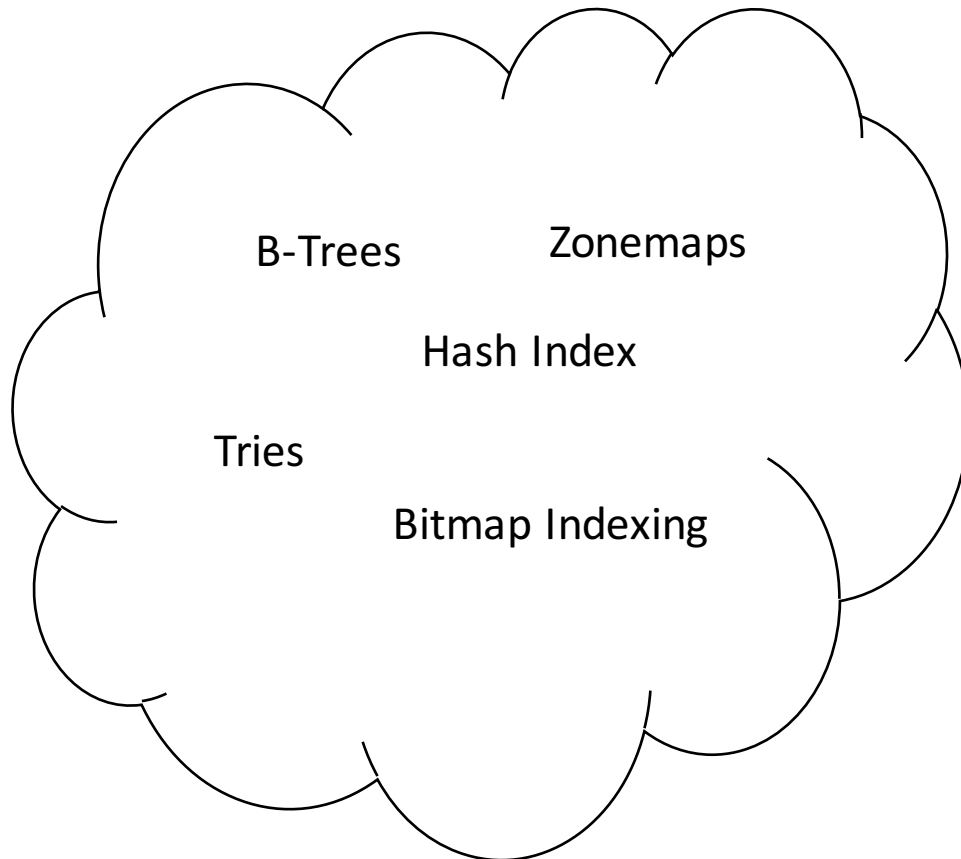
Feature Implementation

	Time
	5
	1
	14
epoch 1	5
	1
	4
epoch 2	3
	19
	22
epoch 3	15
	9
	13
	6

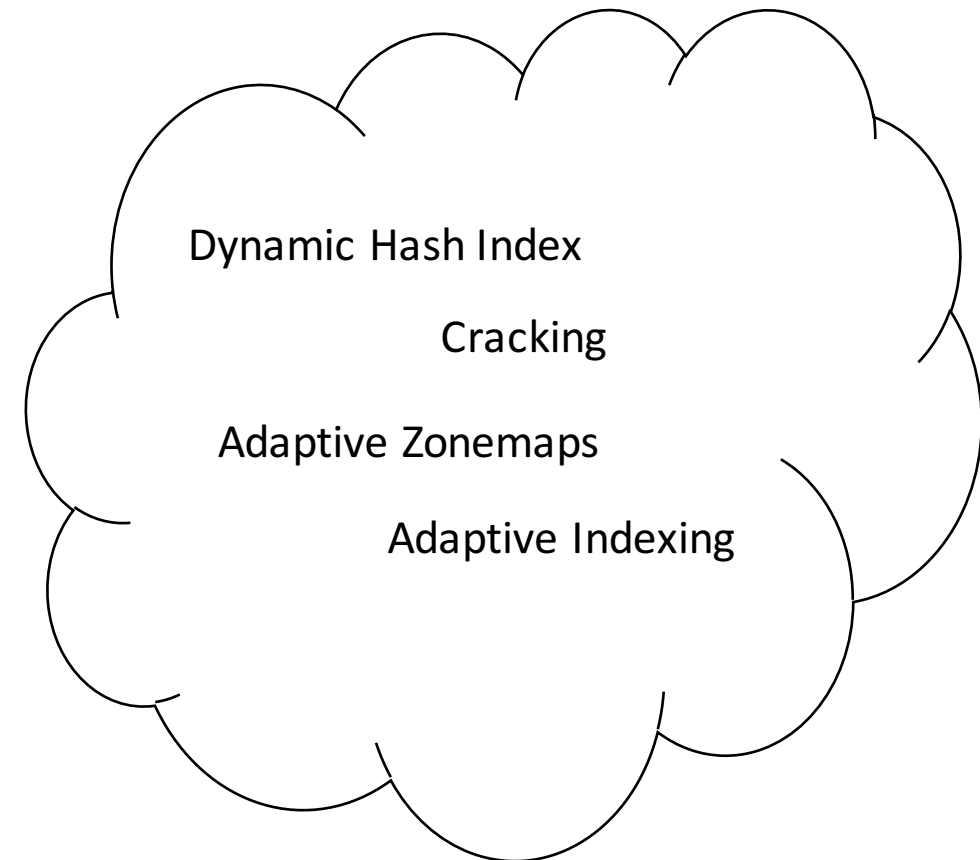
Partitioning

... by example

Static



Dynamic



Partitioning: a detailed example

a tight column:

8	2	1	7	6	9	3	
---	---	---	---	---	---	---	--

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

Partitioning: a detailed example

a tight column:

8	2	1	7	6	9	3	
---	---	---	---	---	---	---	--

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

a tight sorted column:

1	2	3	6	7	8	9	
---	---	---	---	---	---	---	--

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

Partitioning: a detailed example

a tight column:

8	2	1	7	6	9	3	
---	---	---	---	---	---	---	--

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

a tight sorted column:

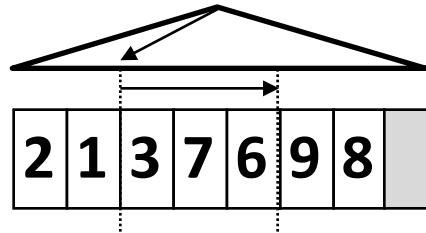
1	2	3	6	7	8	9	
---	---	---	---	---	---	---	--

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

adding clustering:

2	1	3	7	6	9	8	
---	---	---	---	---	---	---	--

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



Partitioning: a detailed example

a tight column:

8	2	1	7	6	9	3	
---	---	---	---	---	---	---	--

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

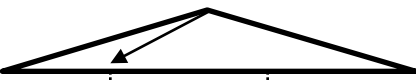
a tight sorted column:

1	2	3	6	7	8	9	
---	---	---	---	---	---	---	--

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

adding clustering:


2	1	3	7	6	9	8	
---	---	---	---	---	---	---	--



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

... and ghost values:

2	1		3	7	6		9	8	
---	---	--	---	---	---	--	---	---	--

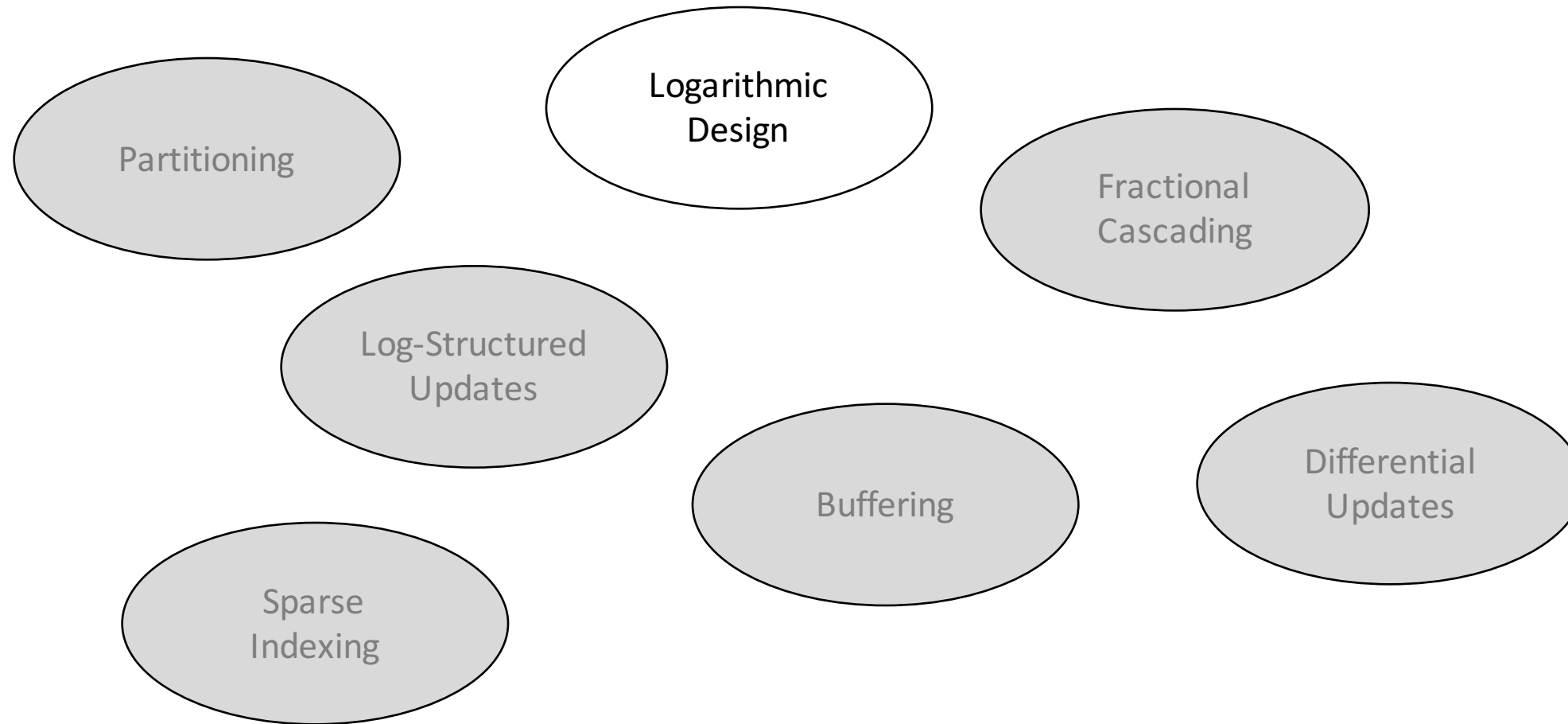


- efficient **reads**
- small **memory** overhead (but increased)
- **updates**: reorganization (but inserts for free)

Design Opportunity

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>
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?



Logarithmic Design

Definition



organize metadata in an exponentially increasing manner

helps reads (logarithmic search)

helps updates (update in place/amortize update cost)

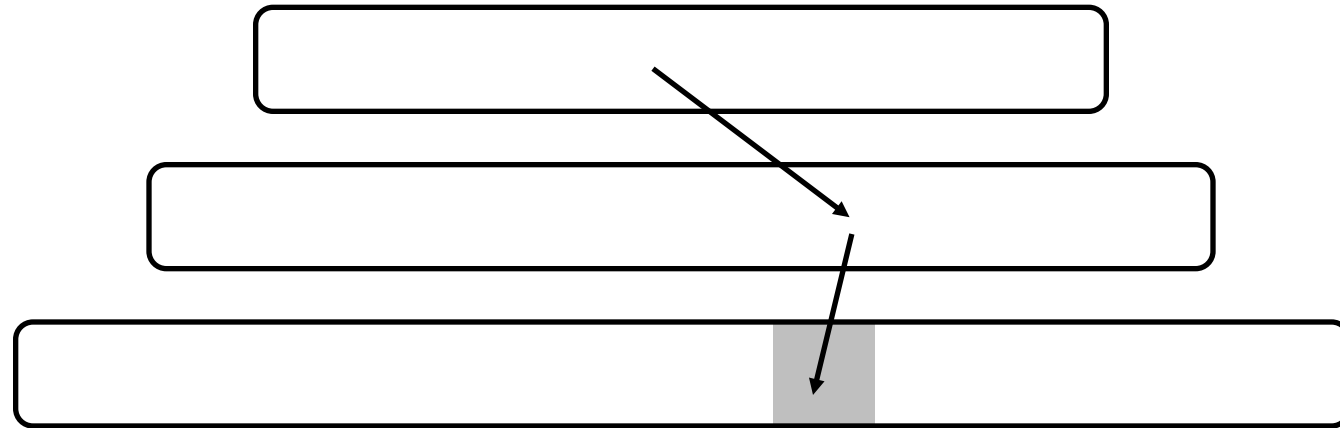
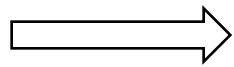
... at the expense of the metadata

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

Logarithmic Design

Feature Implementation

connected levels



Tries & Variants

Traditional Tree Structures

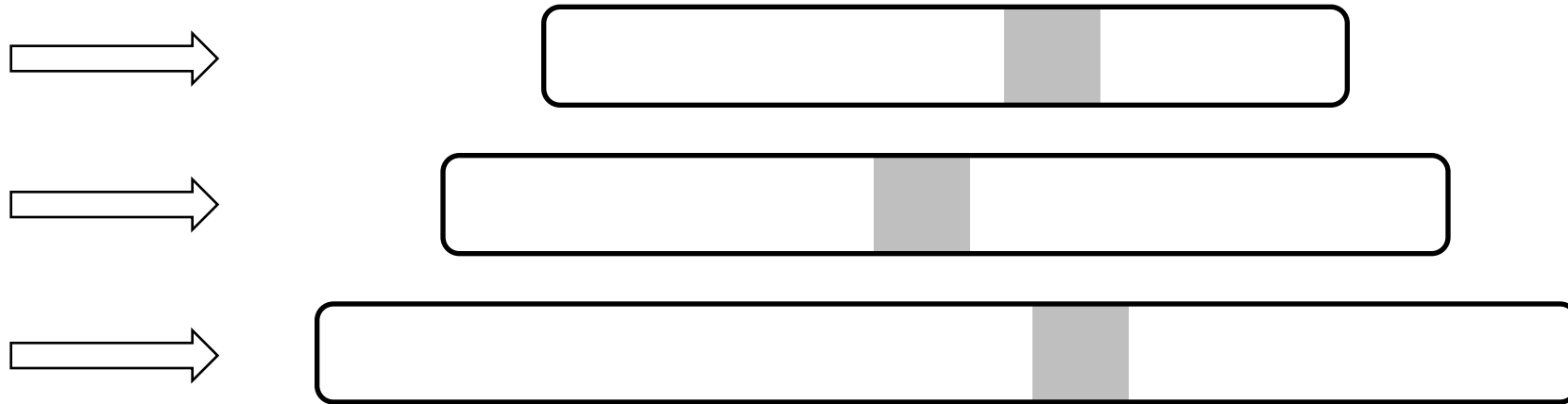
B-Trees & Variants

Tree-Trie hybrids

Logarithmic Design

Feature Implementation

independent levels



LSM Trees & Variants

MaSM

Update-optimized data organization:

FD-Tree

Stepped-Merge

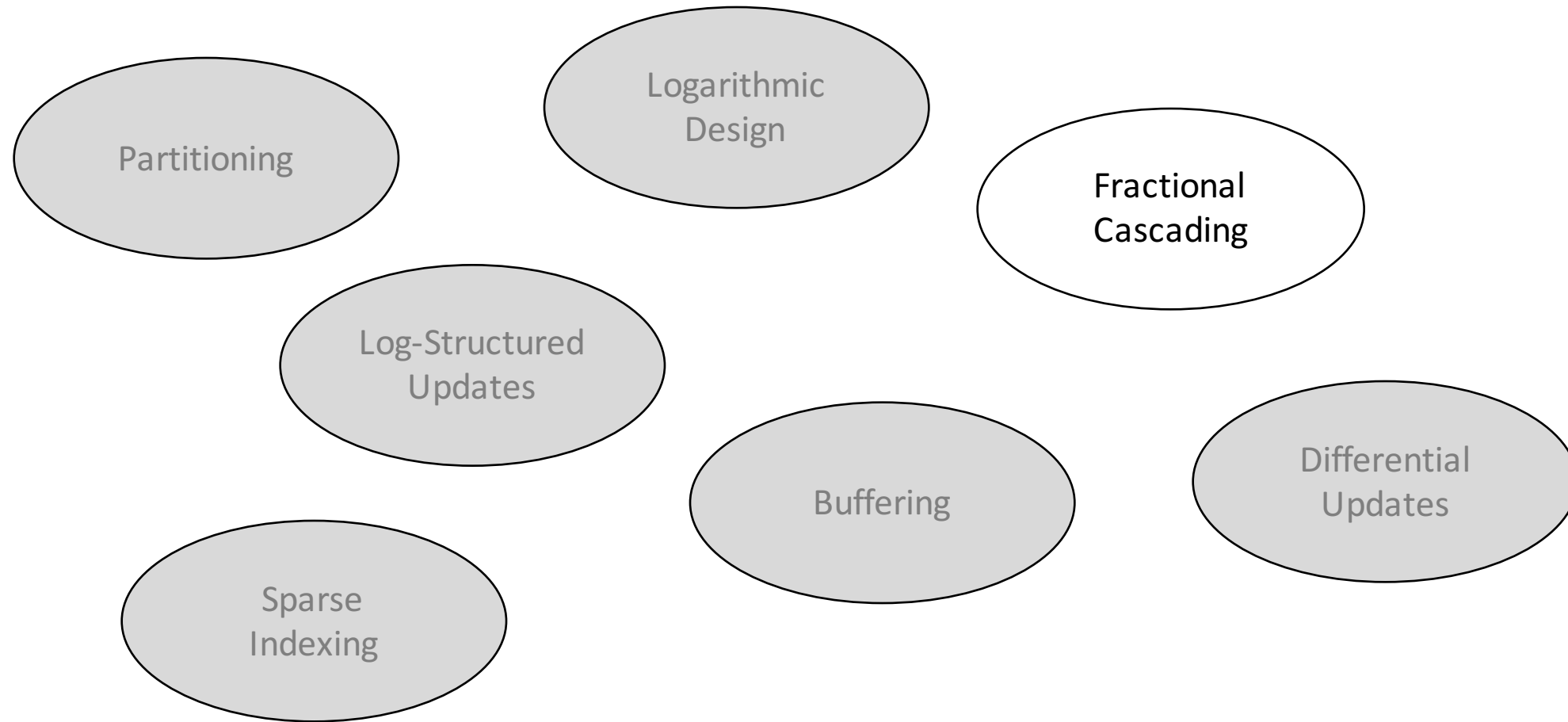
Design Opportunity

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>
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)

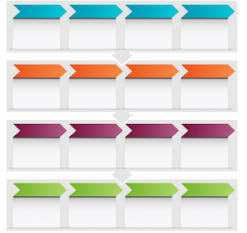
[2] Tries (CACM 1960), PATRICIA (JACM 1968), ART (ICDE 2013)

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



Fractional Cascading

Definition



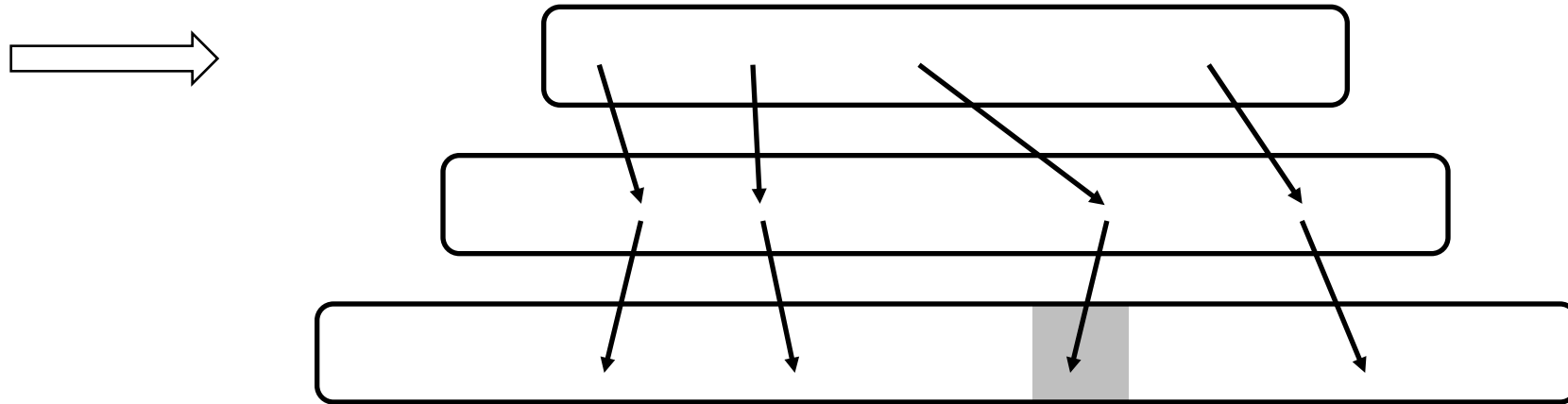
adds metadata for efficient accessing/searching
pointers between different "levels" of access methods
easy navigation to the "corresponding" partitions
need maintenance on updates

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

Fractional Cascading

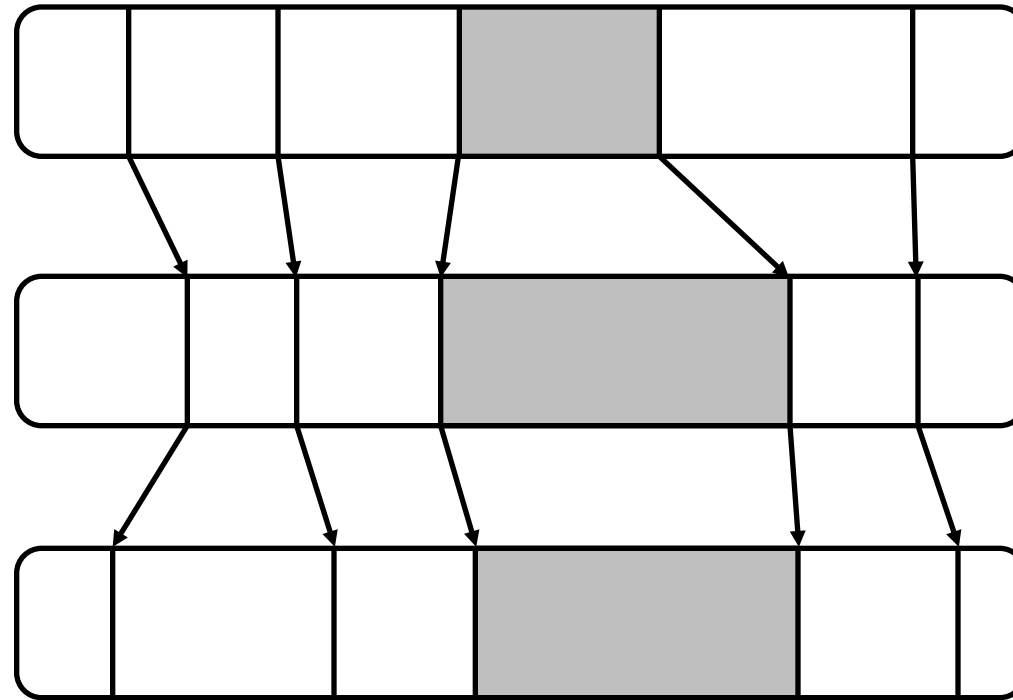
Feature Implementation

Naturally exists in connected levels!



Fractional Cascading *Feature Implementation*

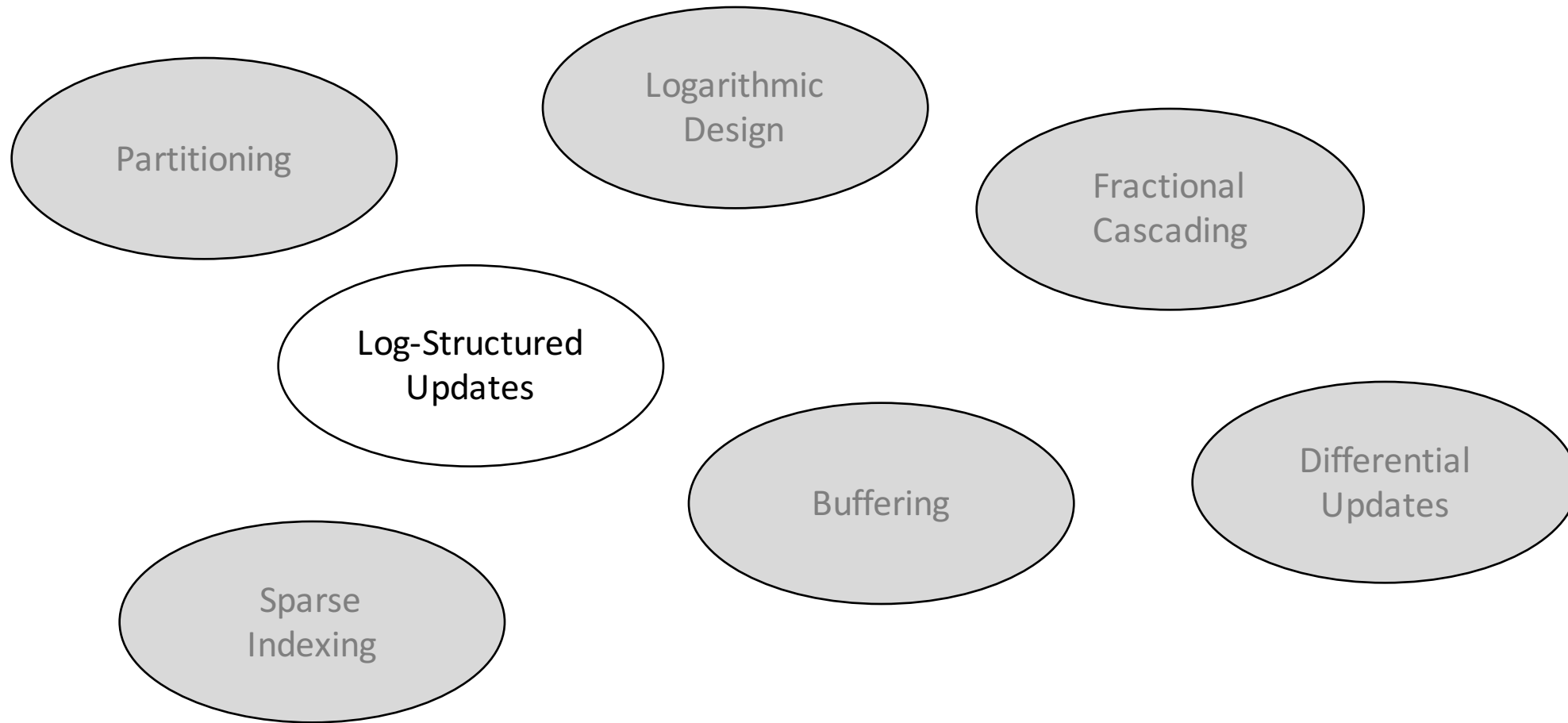
An additional layer of metadata for independent levels



Design Opportunity

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

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



Log-Structured Updates

Definition



apply and organize updates by

appending instead of in-place updates

reads need to merge updates with old data

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

Design Opportunity

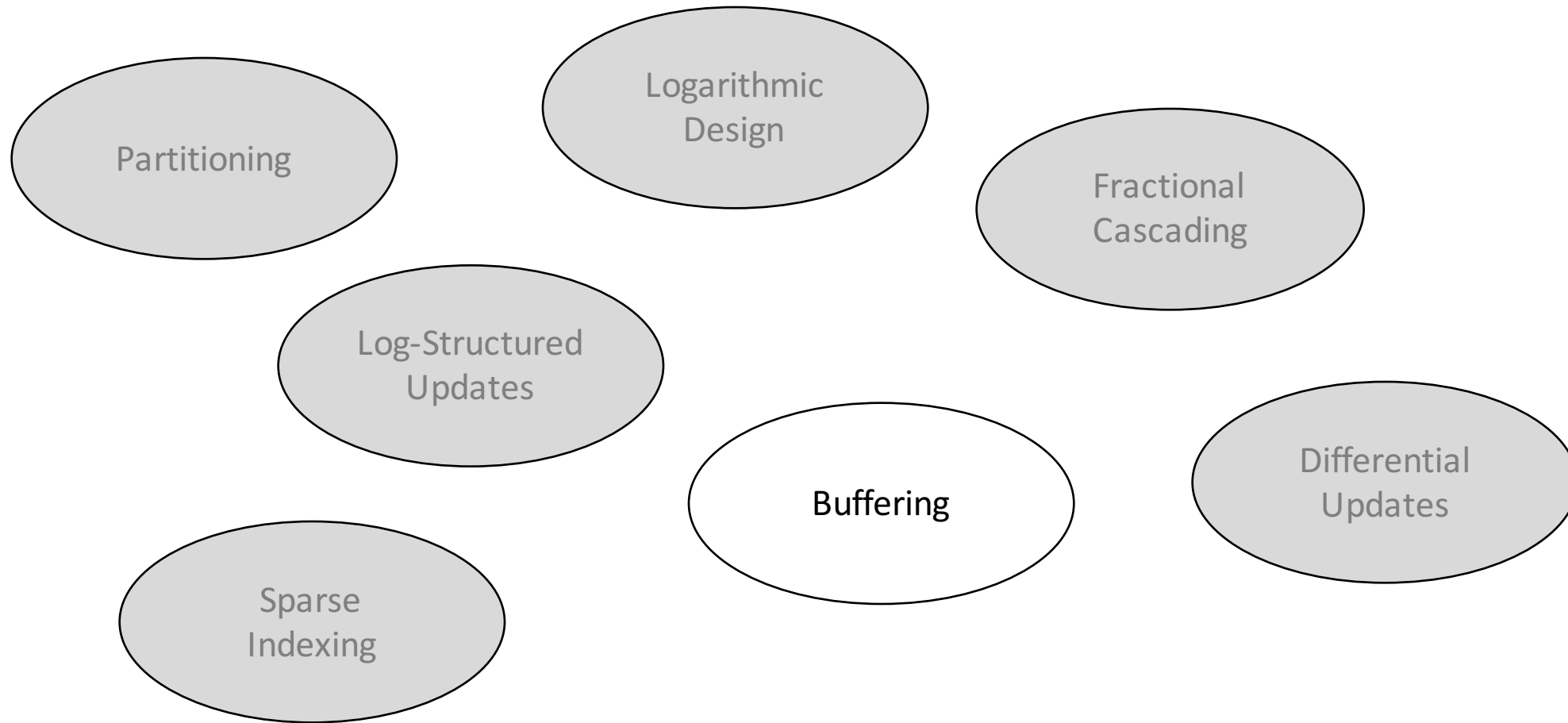
	Partitioning	Logarithmic Design	Fractional Cascading	Log- Structured	Buffering	Differential Updates	Sparse Indexing
B-Trees & Variants	range	✓	≈	[1]			
Tries & Variants	radix	✓	✓				
LSM-Trees & Variants	time	✓	≈	✓ (naturally)			
Differential Files	time, range			[2]			

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

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

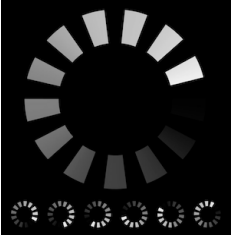
[1] Storage/Memory-Aware Trees: Bw-Tree (ICDE 2013), μ -Tree (EMSOFT 2007), IPLB⁺-Tree (JISE 2011)

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



Buffering

Definition



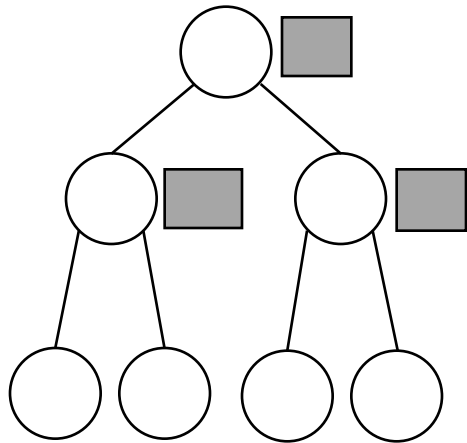
explicitly buffer recently read / updated objects / requests

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

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

Buffering

Buffering Recent Reads/Updates/Requests



Feature Implementation

Buffering Recent Updates



Design Opportunity

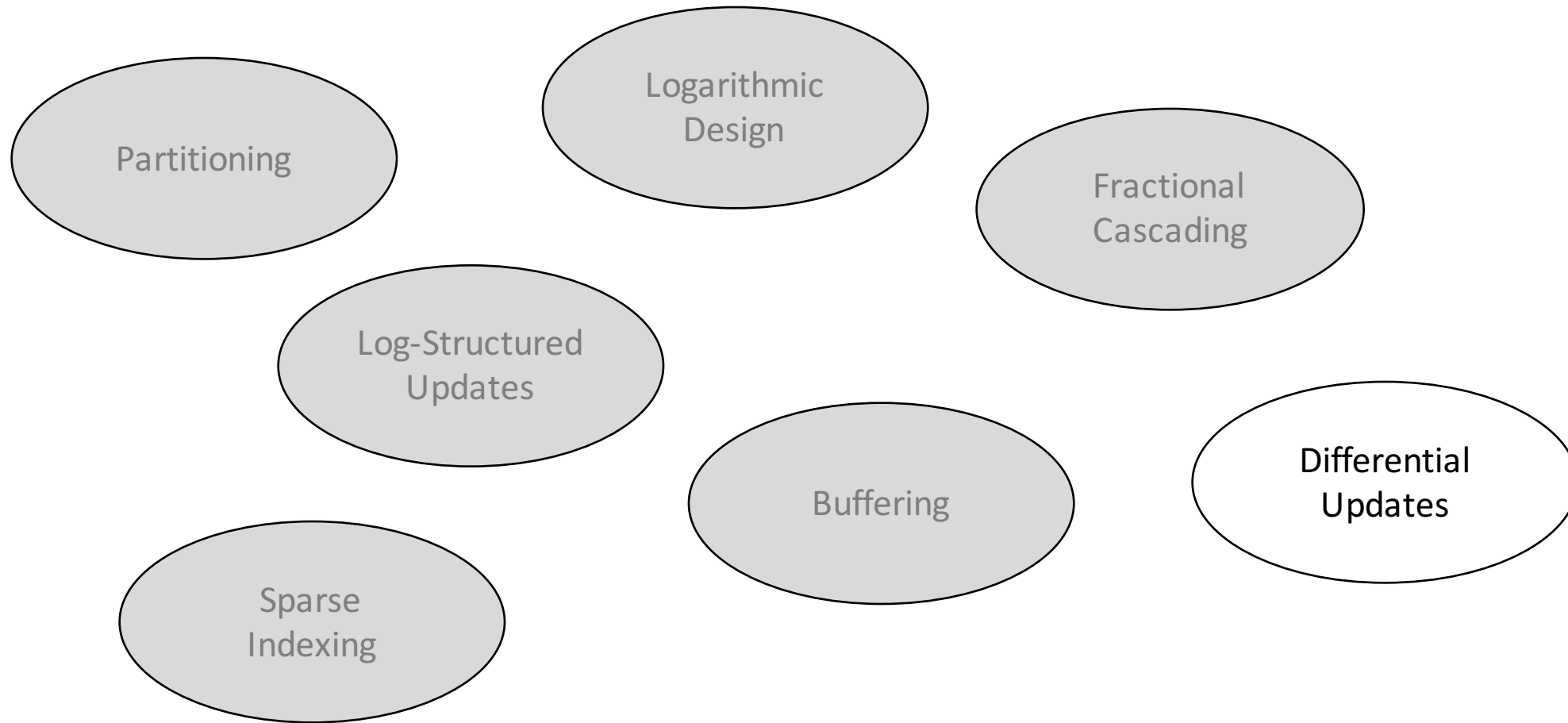
	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>
B-Trees & Variants	range	✓	✓	≈	reads [1] updates [2] requests [3]		
Tries & Variants	radix	✓	✓				
LSM-Trees & Variants	time	✓	≈	✓	✓		
Differential 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)



Differential Updates

Definition



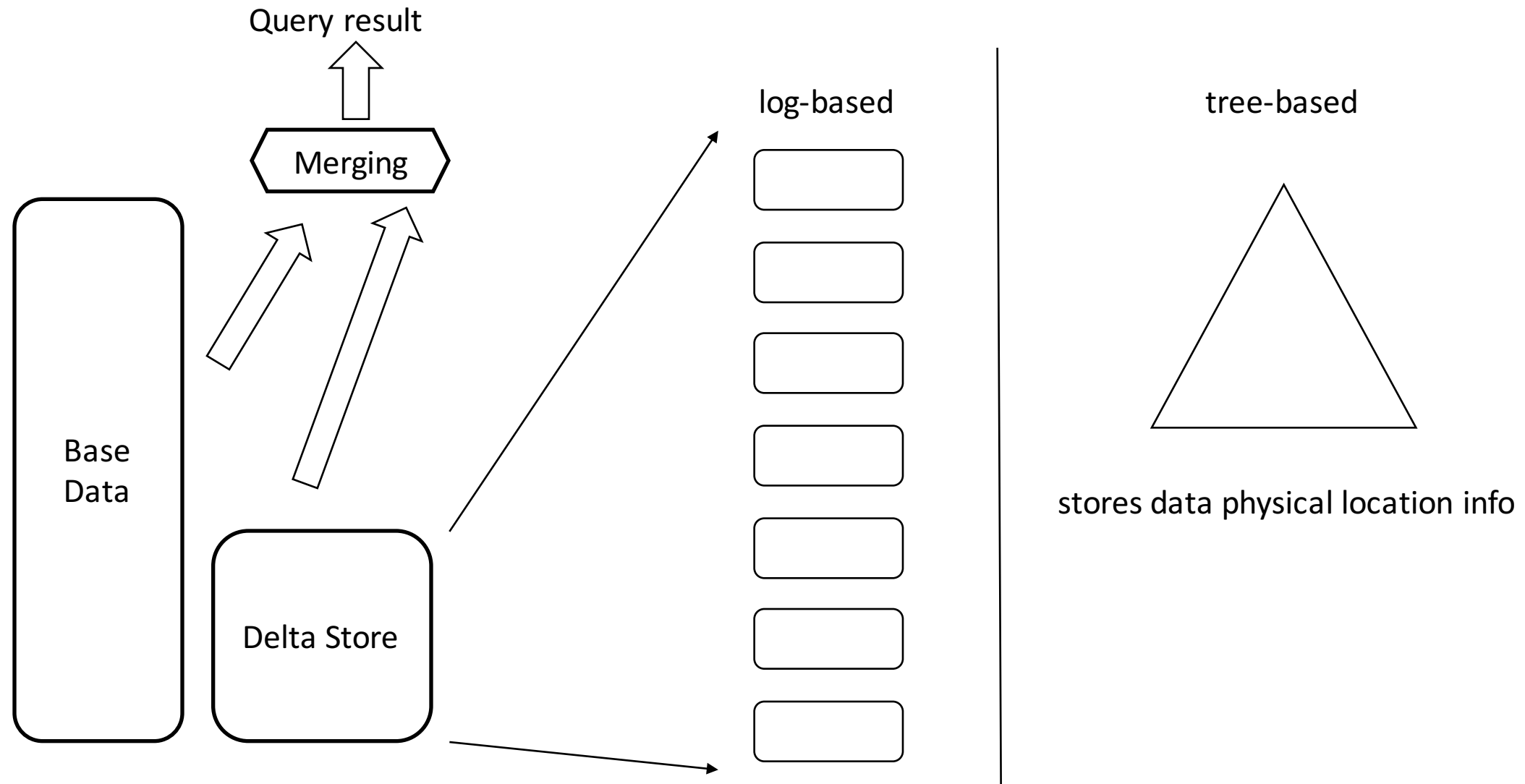
next step for **log-structure**

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

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

Differential Updates

Feature Implementation

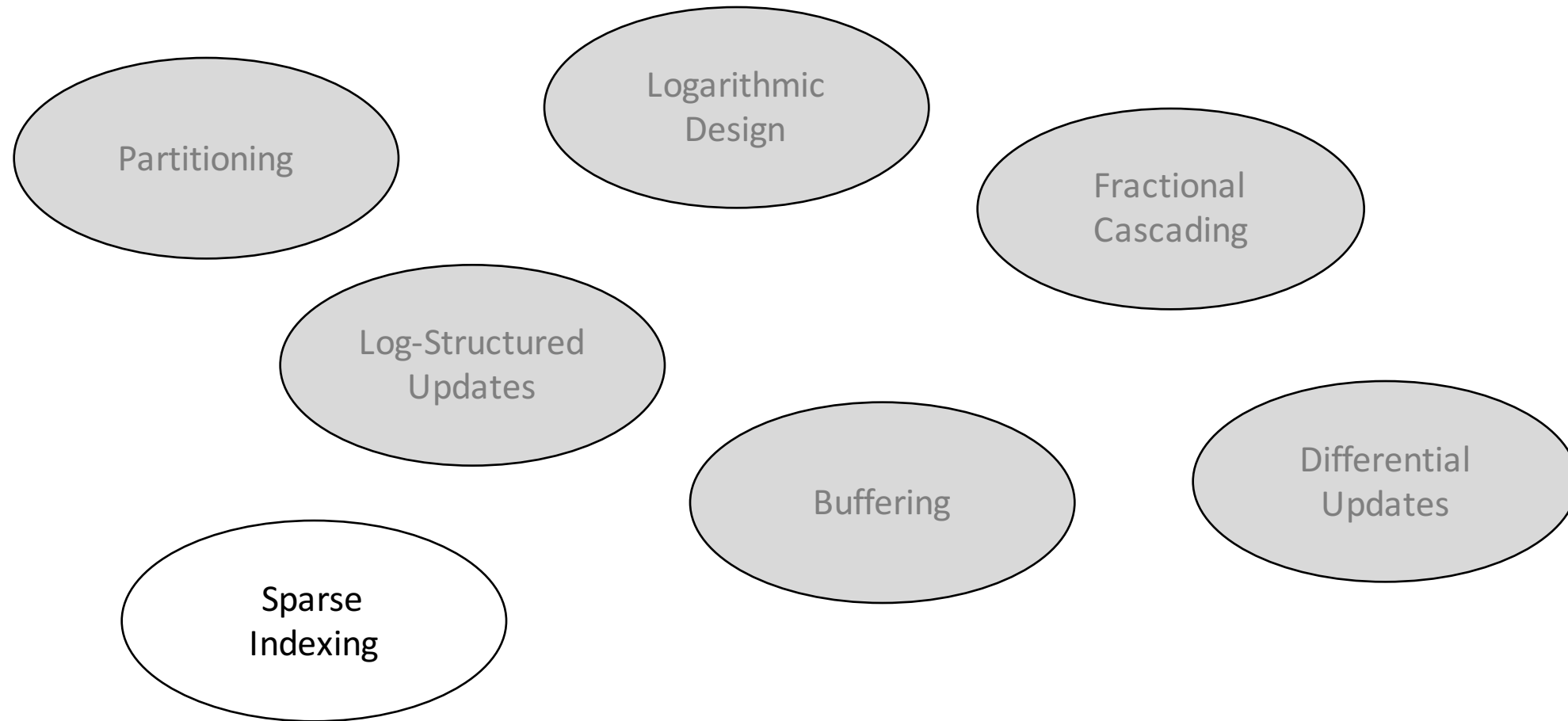


Design Opportunity

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>
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)



Sparse Indexing

Definition

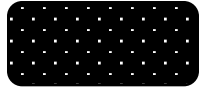


light-weight indexing that allows for skipping unnecessary data

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

Sparse Indexing

membership tests



Data

bitwise representation



Feature Implementation

sparse range partitioning



Data

Design Opportunity

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>
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)

Design Opportunity

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>
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						

Design Opportunity

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>
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!

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!

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!

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>	<i>Adaptivity</i>
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	?	?	?	?	?	?	?

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>	<i>Adaptivity</i>
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)

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>	<i>Adaptivity</i>
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)

	<i>Partitioning</i>	<i>Logarithmic Design</i>	<i>Fractional Cascading</i>	<i>Log- Structured</i>	<i>Buffering</i>	<i>Differential Updates</i>	<i>Sparse Indexing</i>	<i>Adaptivity</i>
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)

DATA SYSTEMS. LABORATORY

@ Harvard School of Engineering and Applied Sciences

Designing data systems for the big data era

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

thank you!