Graph databases, algorithms and indexing

Poseidon: A OneGraph Engine

19th Scheduling for large-scale systems workshop
Villa Clythia, Fréjus (France), March 17, 2026

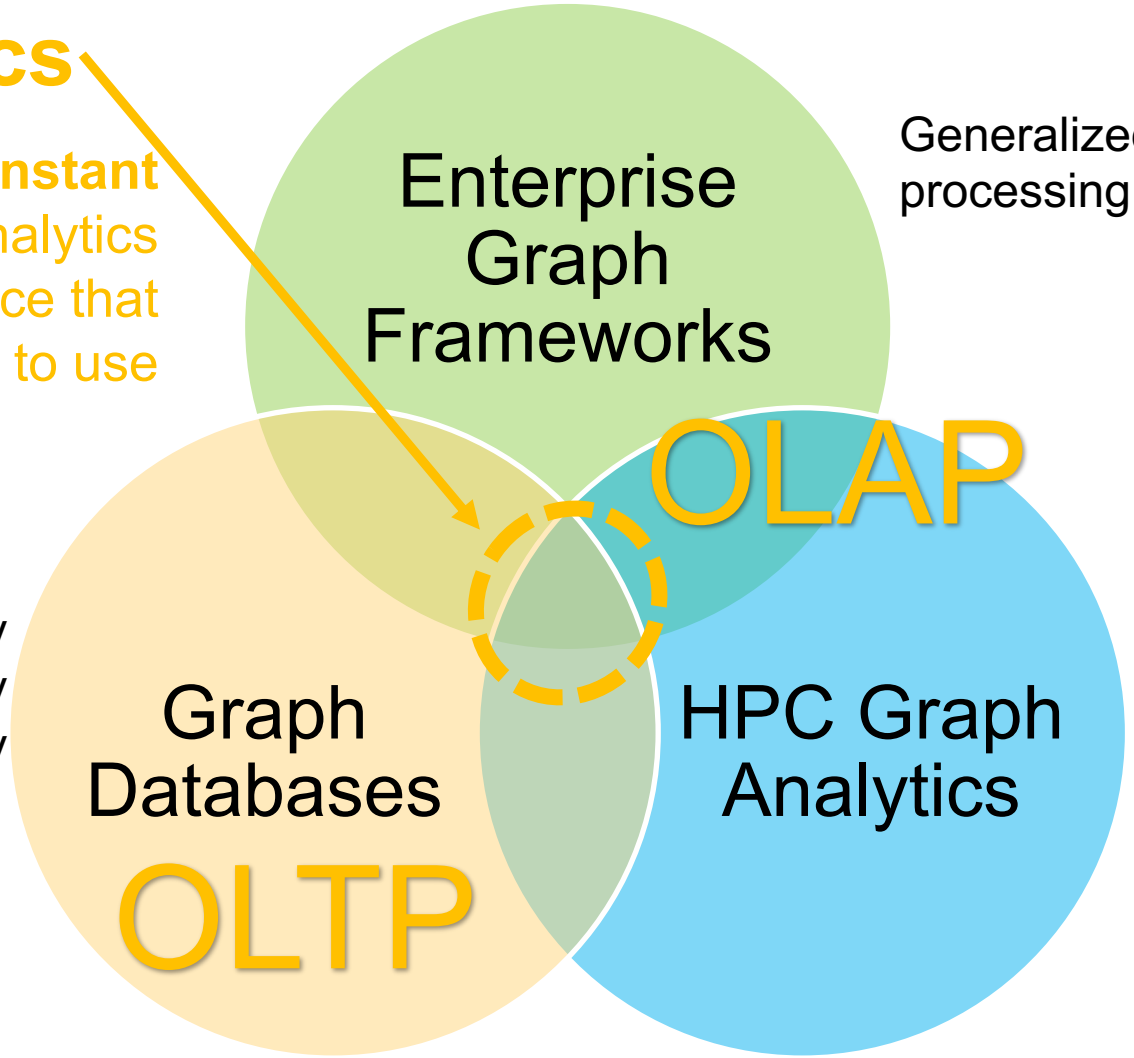
Brad Bebee, Ümit V. Çatalyürek, Olaf Hartig, Ankesh Khandelwal, Simone Rondelli, Michael Schmidt, Lefteris Sidirourgos, and Bryan Thompson
Neptune Team, Amazon Web Services

to appear in SIGMOD'26 available in arXiv

Landscape of current “Graph World”

Neptune Analytics

Performance is within a **constant factor** of HPC Graph Analytics delivered as a managed service that is easy to use



Generalized APIs for graph processing

Generalized graph query languages with consistency and durability

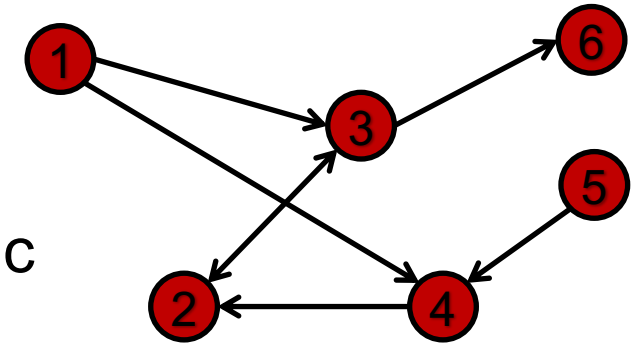
High performance for specific graph problems

Outline

- Graph? What is it?
- Interoperability – Logical Data Model
- DB challenges: In-memory DB – compact storage
- DB challenges: Access Patterns
- DB challenges: (Partitioned) Indices
- Graph Analytics challenges: Kernel Access Patterns

Graph is a sparse matrix

- Adjacency matrix
- For n vertices and m edges
  - Directed -> binary, not symmetric



1	3, 4
2	3
3	2, 6
4	2
5	4
6	

Adjacency List

Space: $O(n+m)$

	1	2	3	4	5	6
1			1	1		
2			1			
3		1				1
4		1				
5				1		
6						

Adjacency ***Sparse*** Matrix

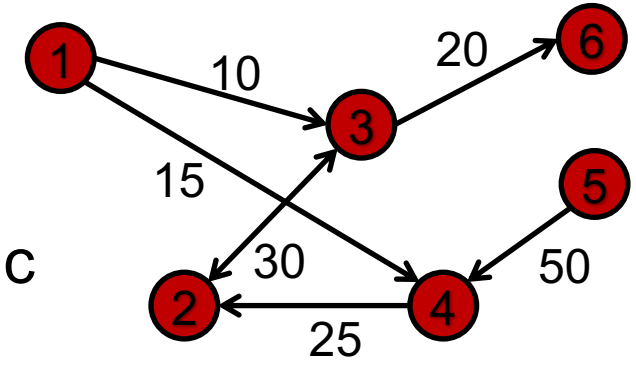
	1	2	3	4	5	6
1	0	0	1	1	0	0
2	0	0	1	0	0	0
3	0	1	0	0	0	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	0

Adjacency Matrix

Space: $O(n^2)$

Graph is a sparse matrix

- Adjacency matrix
- For n vertices and m edges
  - Directed -> binary, not symmetric
  - Weighted



1	3	4
2	3	
3	2	6
4	2	
5	4	
6		

Adjacency List

Space: $O(n+m)$

w	10	15	30	30	20	25	50
adj	3	4	3	2	6	2	4
xadj	1	3	4	6	7	8	8
	1	2	3	4	5	6	7

Compressed Adjacency List / Compressed Sparse Row (CSR)

Space: $O(n+m)$

Space = $2m+n+1$

s	t	w
1	3	10
1	4	15
2	3	30
3	2	30
3	6	20
4	2	25
5	4	50

Edge List / Coordinate (COO)

Space: $O(m)$

Space = $3m$

Fixed schema

Graph Data Models

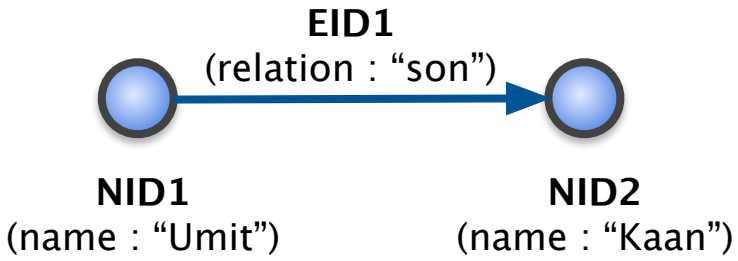
LPG: Labeled Property Graphs

Vertices

Nodes: Label/ID + *Properties* (set of key-value pairs)

Edges

Relationships: Label/ID + Type + Properties



RDF: Resources Description Framework

Vertices

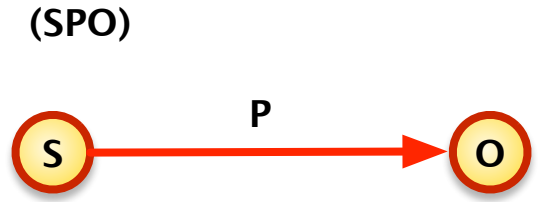
Resources: URIs

Attribute Values: Literals

Edges

Relationships: URIs

RDF Triple: **S**ubject-**P**redicate-**O**bject



There is no internal structure for nodes and edges

Real World Graph Samples

CSV files: Gremlin, OpenCypher (w/ minor differences)

nodes.csv

```
~id, name:String, val:Int, ~label
v1,"Bryan",1,person;human
v2,"Michael",2,person;human
v3,"Umit",3,person;human
v4,"Olaf",4,person;human
v5,"Brad",5,person;human
```

edges.csv

```
~id, ~from, ~to, ~label, since:Int
e1,v1,v2,knows,2010
e2,v1,v3,knows,2020
e3,v2,v4,knows,2014
e4,v3,v4,knows,2022
e5,v3,v5,knows,2020
e6,v4,v5,knows,2015
```

Real World Graph Samples

in an N-Triples compliant 1G syntax

```
<http://www.mapgraph.com/1> <http://xmlns.com/foaf/0.1/knows> <http://www.mapgraph.com/2> _:sid1 .
<http://www.mapgraph.com/1> <http://xmlns.com/foaf/0.1/knows> <http://www.mapgraph.com/3> _:sid2 .
<http://www.mapgraph.com/2> <http://xmlns.com/foaf/0.1/knows> <http://www.mapgraph.com/4> _:sid3 .
<http://www.mapgraph.com/3> <http://xmlns.com/foaf/0.1/knows> <http://www.mapgraph.com/4> _:sid4 .
<http://www.mapgraph.com/3> <http://xmlns.com/foaf/0.1/knows> <http://www.mapgraph.com/5> _:sid5 .
<http://www.mapgraph.com/4> <http://xmlns.com/foaf/0.1/knows> <http://www.mapgraph.com/5> _:sid6 .
<http://www.mapgraph.com/1> <http://www.w3.org/1999/02/22-rdf-syntax-ns:label> "Bryan" _:sid7 .
<http://www.mapgraph.com/2> <http://www.w3.org/1999/02/22-rdf-syntax-ns:label> "Michael" _:sid8 .
<http://www.mapgraph.com/3> <http://www.w3.org/1999/02/22-rdf-syntax-ns:label> "Umit" _:sid9 .
<http://www.mapgraph.com/4> <http://www.w3.org/1999/02/22-rdf-syntax-ns:label> "Olaf" _:sid10 .
<http://www.mapgraph.com/5> <http://www.w3.org/1999/02/22-rdf-syntax-ns:label> "Brad" _:sid11 .
_:sid1 <http://www.mapgraph.com/since> "2010"^^<http://www.w3.org/2001/XMLSchema#int> _:sid17 .
_:sid2 <http://www.mapgraph.com/since> "2020"^^<http://www.w3.org/2001/XMLSchema#int> _:sid18 .
_:sid3 <http://www.mapgraph.com/since> "2014"^^<http://www.w3.org/2001/XMLSchema#int> _:sid19 .
_:sid4 <http://www.mapgraph.com/since> "2022"^^<http://www.w3.org/2001/XMLSchema#int> _:sid20 .
_:sid5 <http://www.mapgraph.com/since> "2020"^^<http://www.w3.org/2001/XMLSchema#int> _:sid21 .
_:sid6 <http://www.mapgraph.com/since> "2015"^^<http://www.w3.org/2001/XMLSchema#int> _:sid22 .
```

Outline

- Graph? What is it?
- Interoperability – Logical Data Model
- DB challenges: In-memory DB – compact storage
- DB challenges: Access Patterns
- DB challenges: (Partitioned) Indices
- Graph Analytics challenges: Kernel Access Patterns

Graph interoperability

- What if we did not have to choose between RDF and LPG?
- What if we could use OpenCyper or Gremlin over RDF, or SPARQL over LPG?
- Interoperability: single graph (meta)model, free use of any query language

- 1G model (“one graph to rule them all”)
  - "Graph? Yes! Which one? Help!", O. Lassila, M. Schmidt, B. Bebee, D. Bechberger, W. Broekema, A. Khandelwal, K. Lawrence, R. Sharda, B. Thompson, arXiv:2110.13348v1, 2021.

Storage Challenges: Interoperable Physical Data Model

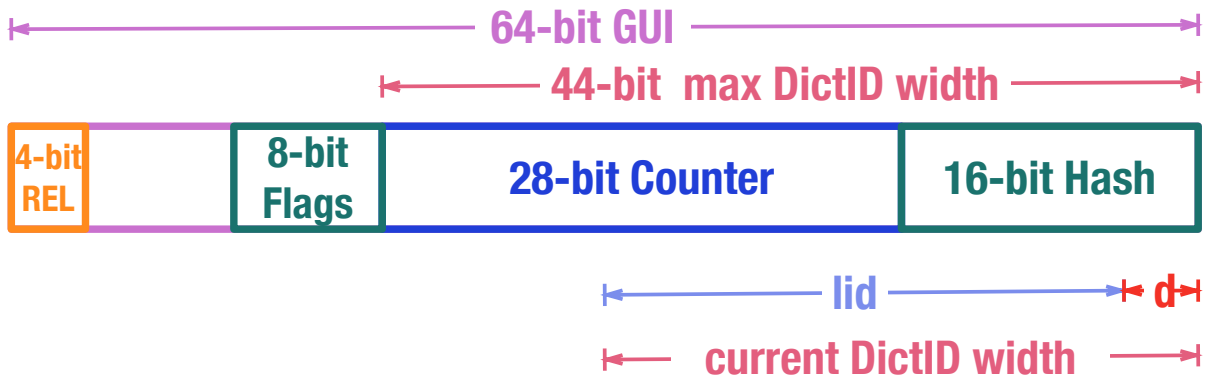
- 1G Graph Storage
  - Three kinds of relations
    - Dictionaries: URIs/Literals → IDs
    - Graph Structure: Topologies – relations between (S)ubject and (O)bject, in other words between "vertices"
    - Graph Data: Values – properties of vertices and edges
  - In 1G, Edges/Properties (of vertices and edges) can become "vertices"
- How to dynamically distribute data?
  - Blocked partition (coarse-grain 2D) as a sweet spot between performance and architecture agnostic algorithm development [1]
- System generated IDs are uniform random
  - Remember graph vertices comes as URIs (strings)
  - Load-balanced partitioning (declustering) is favored against locality for initial load
    - Graph-aware re-partitioning can be done after graph is loaded

[1] PGAbB: A Block-Based Graph Processing Framework for Heterogeneous Platforms
Abdurrahman Yasar, Sivasankaran Rajamanickam, Jonathan W. Berry, Umit V. Catalyurek
https://arxiv.org/abs/2209.04541

Dictionary Encoding

- Convert “Strings” (URIs/Literals) into dictionary encoded 64-bit values

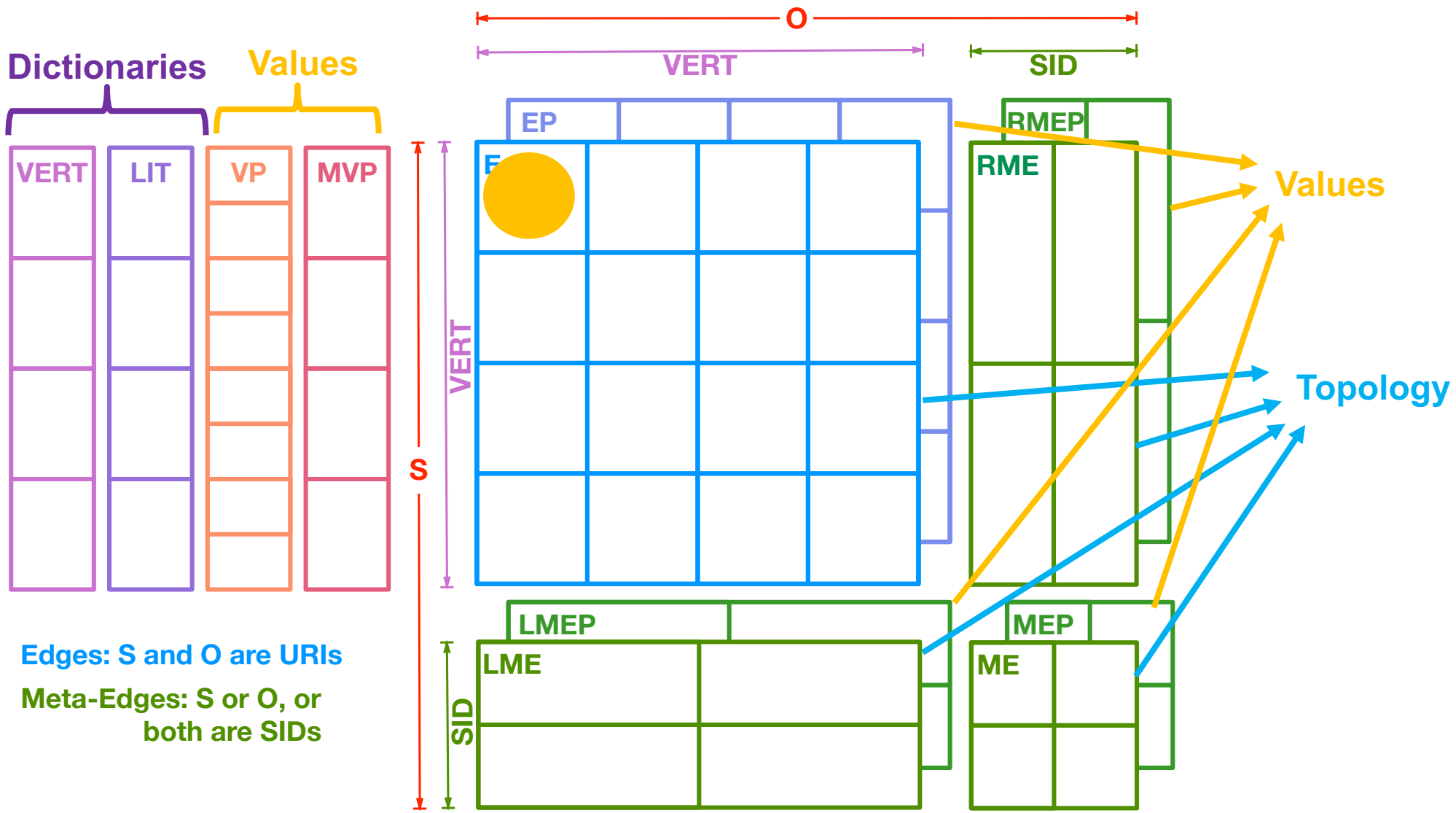
- Partitioned hash
  - Use a hash function to partition (16-bit)
- Use a second hash function to hash/map in each part
- Generate 64-bit Global Unique Identifier (GUI)



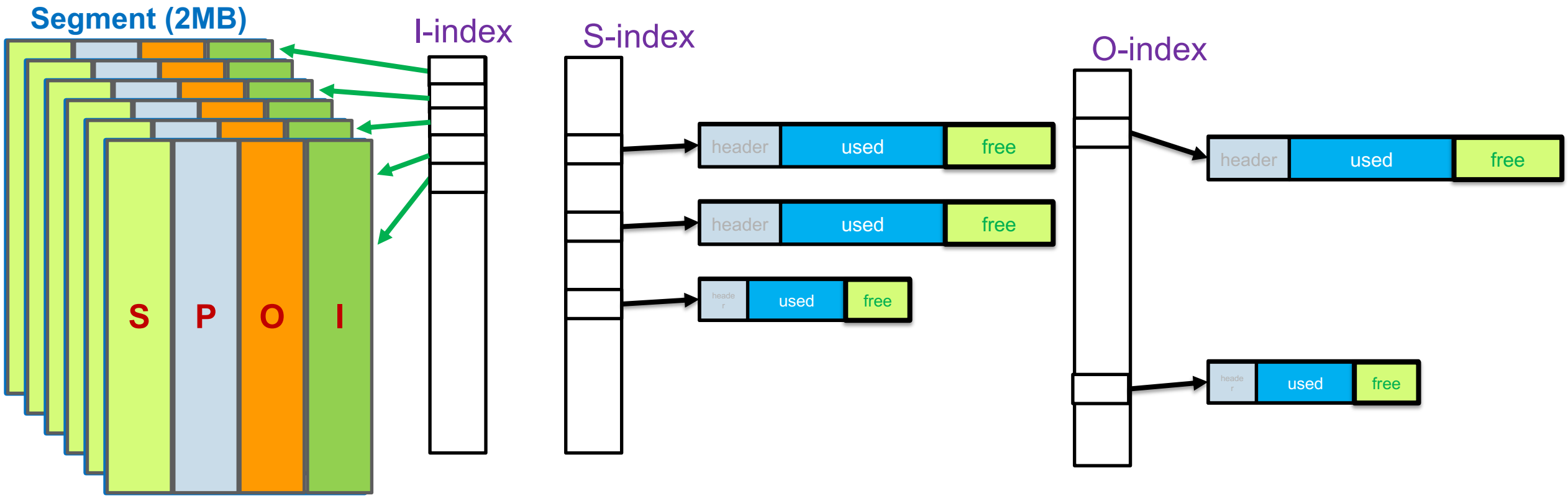
d : partition number, up to 16 bits

lid : 28-bit local id, part of it is coming from Hash, and the rest from the Counter

1D and 2D partitioning of Poseidon Relations



Single Partition of a Relation



TOP: 4x4-byte = 16 bytes

VAL: 3 (S/P/I) x 4-bytes + 8 bytes (O) + 1 byte (Flags) = 21 bytes

WID: 2 (P/I) x 4-bytes + 2 (S/O) x 8-bytes = 24 bytes

Outline

- Graph? What is it?
- Interoperability – Logical Data Model
- DB challenges: In-memory DB – compact storage
- DB challenges: Access Patterns
- DB challenges: (Partitioned) Indices
- Graph Analytics challenges: Kernel Access Patterns

DB Challenges: Access Pattern Language

- Language agnostic to physical data model
- Declarative, abstract (relation separation & internal partitioning)
- Concise and efficient

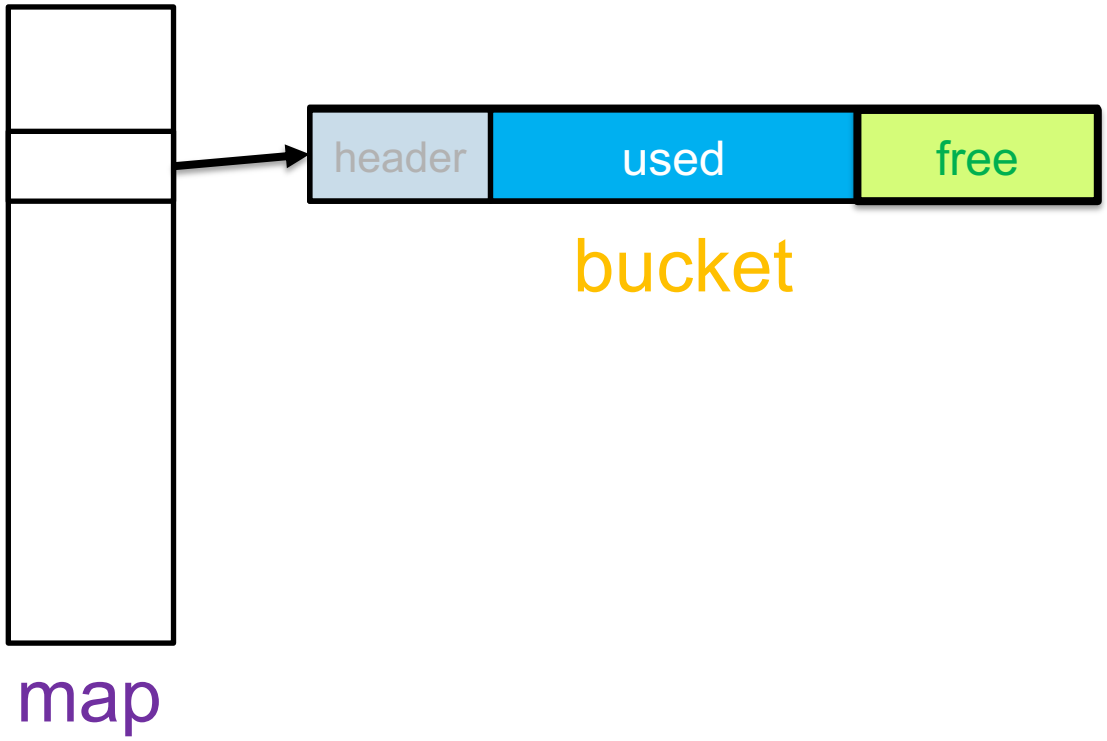
#	Expression	Meaning	Additional Arguments
1	`SP_^_`	retrieve the O value of every SPOI tuple that has a given S value and a given P value	i) value for S, ii) value for P, iii) pointer to data structure to which the O values of the selected tuples are written
2	`_[]P_^_^`	retrieve the O and the I value of all SPOI tuples that have both a specific P value and an S value that is one of the elements in a list of such values	i) value for P, ii) pointer to data structure with frontier for S; iii–iv) pointers to two data structures to which the O and the I values of the selected tuples, respectively are written
3	`s_^_l__`	retrieve all SIDs that are S values of SPOI tuples with a literal in the O position	i) pointer to data structure to which the S values of the selected tuples are written
4	`SP_^x__^_^_`	join every SPOI tuple (s, p, o, i) with every SPOI tuple (s', p', o', i') such that $i = s'$, and retrieve o, p', and o' of every result tuple	i–iii) pointers to three data structures to which the three values of every result tuple are written, respectively

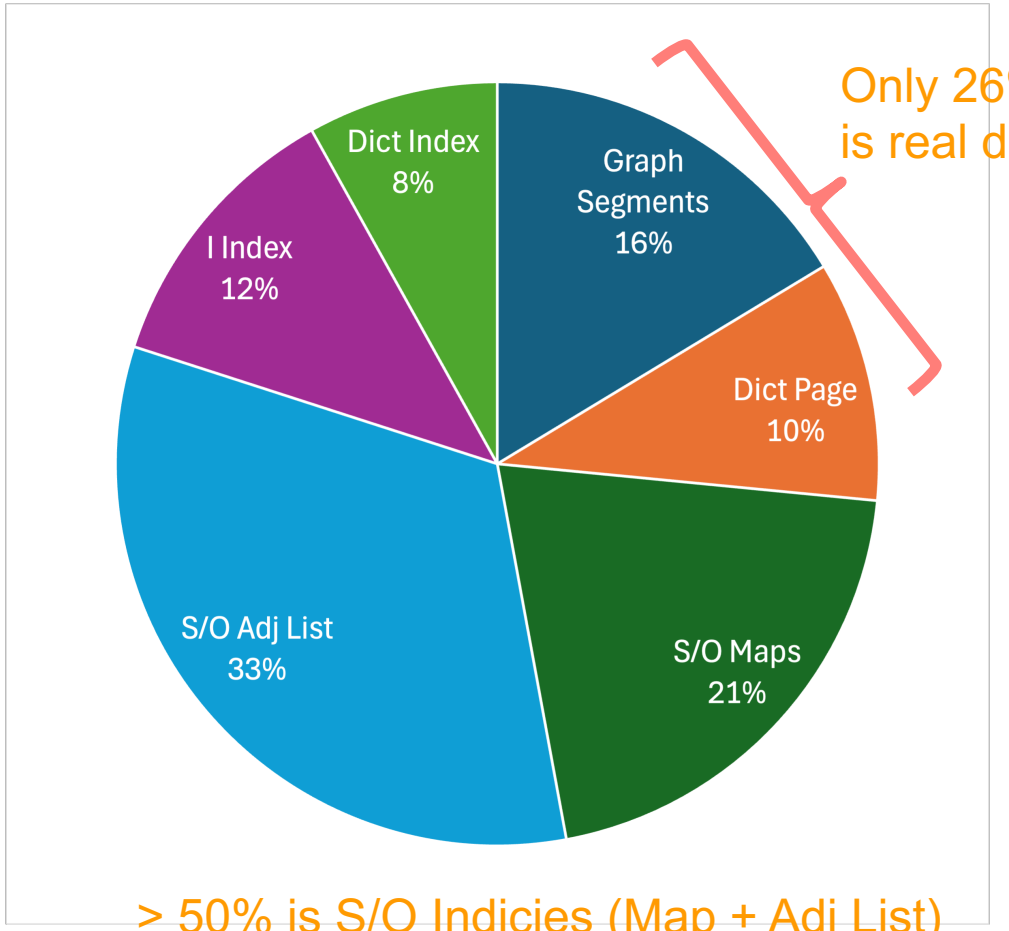
Outline

- Graph? What is it?
- Interoperability – Logical Data Model
- DB challenges: In-memory DB – compact storage
- DB challenges: Access Patterns
- DB challenges: (Partitioned) Indices
- Graph Analytics challenges: Kernel Access Patterns

Poseidon Indices

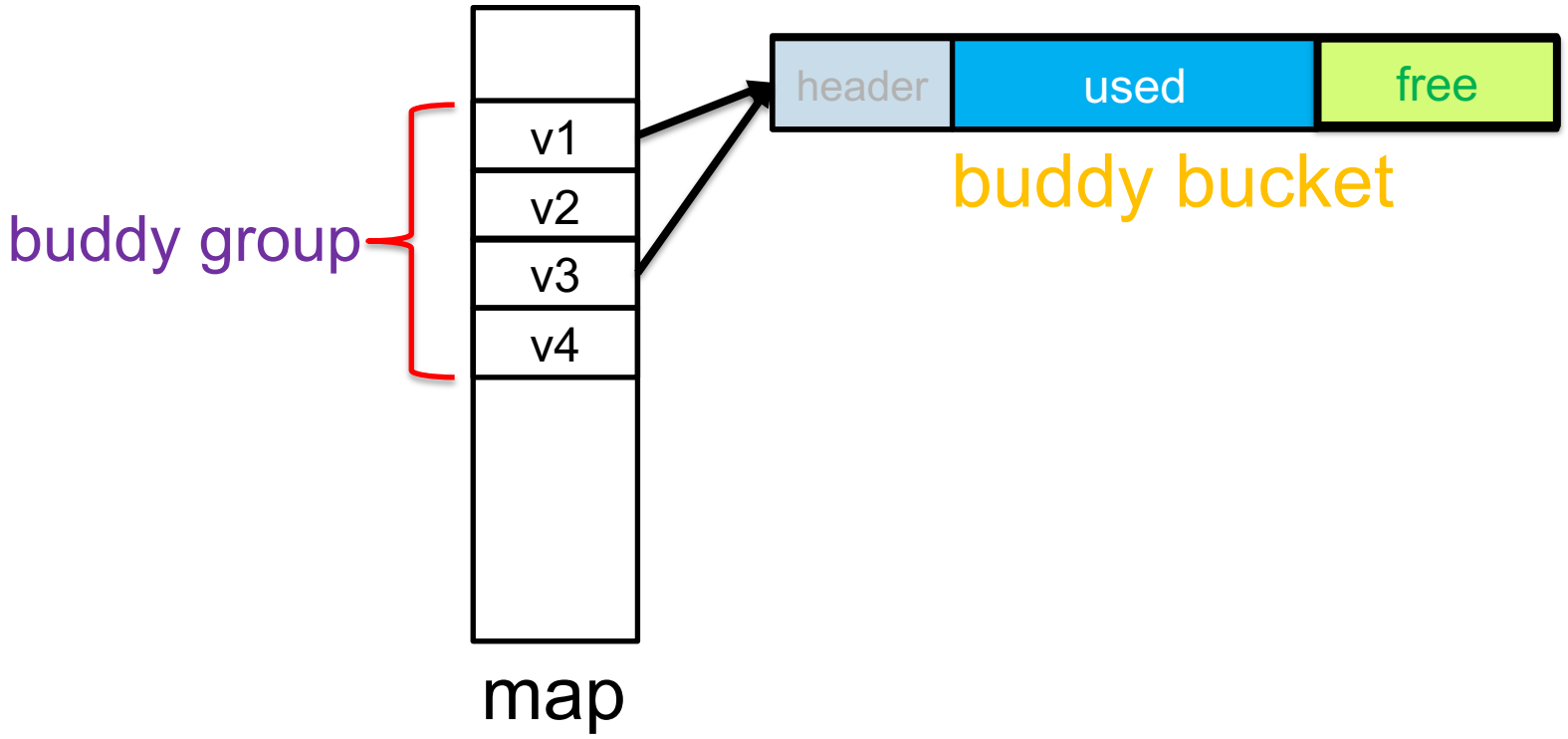
- Each non-dictionary part has three indices : S, O, I
- Each S/O index is a map to a fixed-size bucket that has room for new entries.
- Inserts/deletes are lock-free





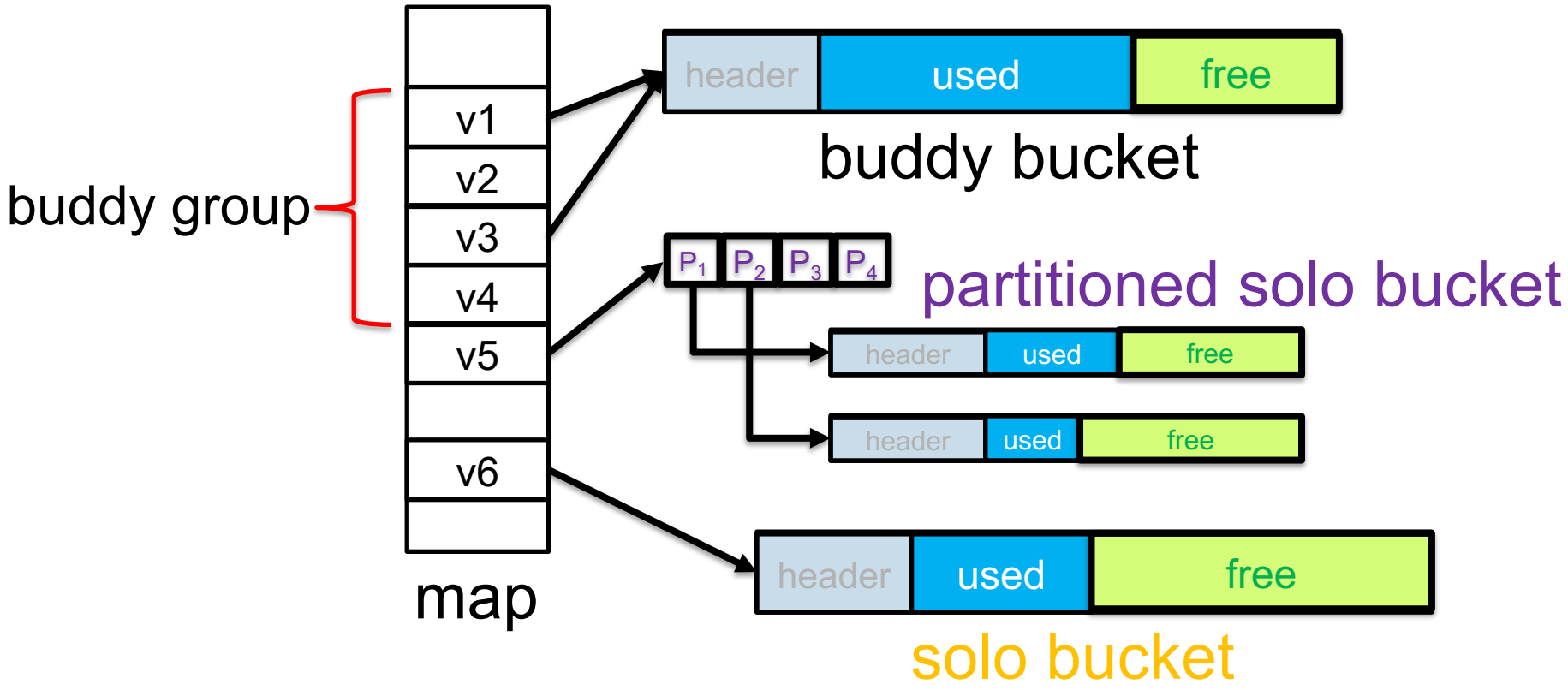
Index improvements: Buddy Buckets

- Share buckets for multiple vertices: buddy group

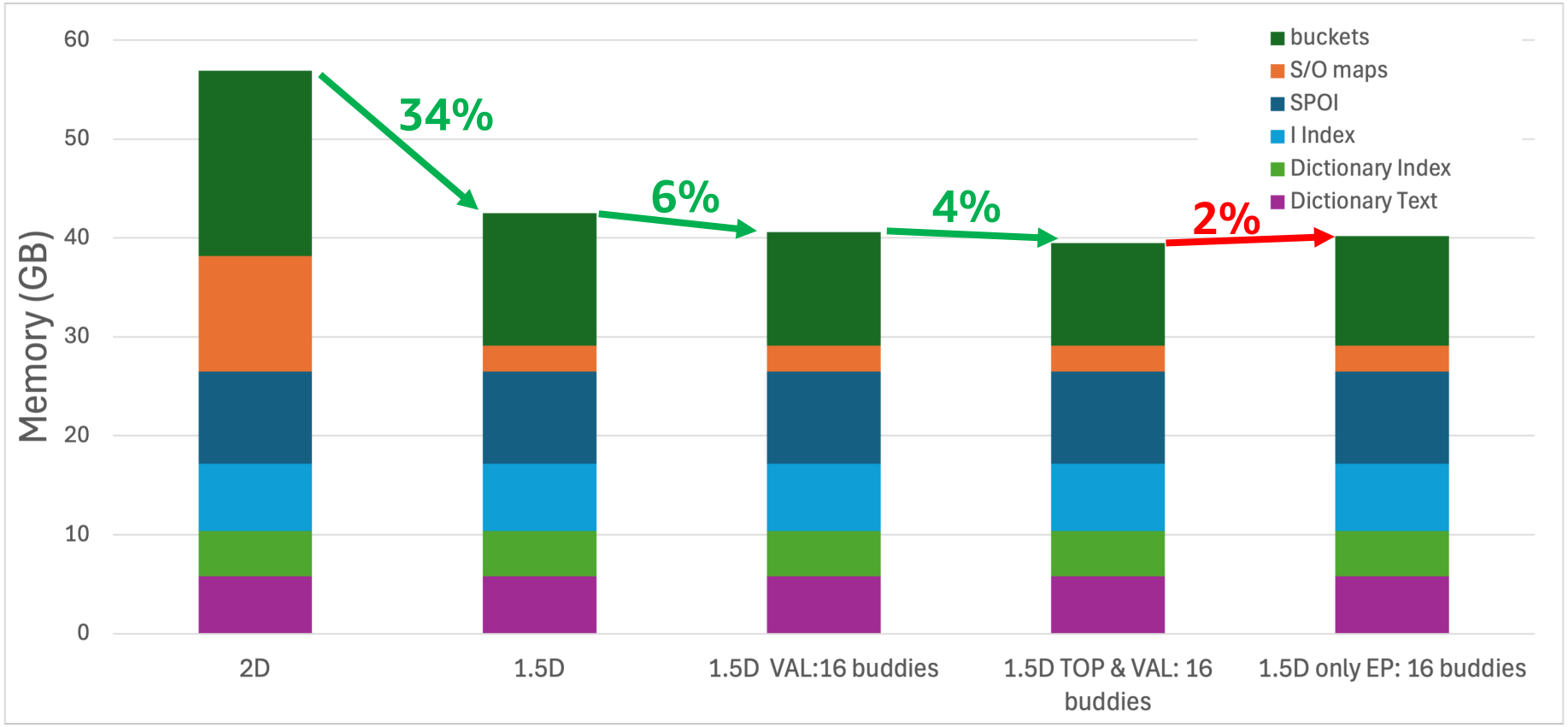


Topology Index improvements: 1.5D

- *Low degree vertices*: Share bucket (called solo bucket) for rows / columns of 2D topology partitioned
- *High degree vertices*: have partitioned solo bucket



Memory: 2D vs 1.5D + buddy buckets

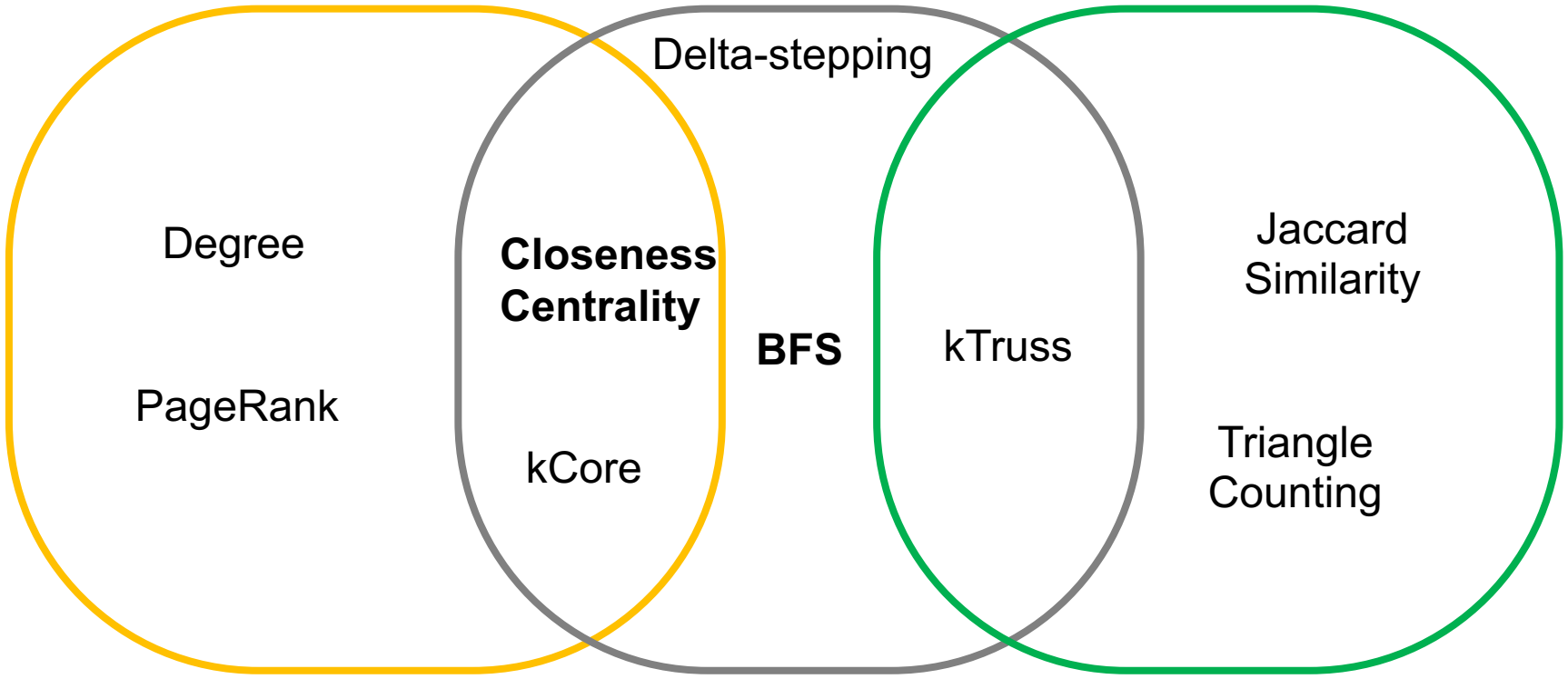


1.5D only EP: 16 buddies:
Sweet spot for runtime performance and 42% space savings

Outline

- Graph? What is it?
- Interoperability – Logical Data Model
- DB challenges: In-memory DB – compact storage
- DB challenges: Access Patterns
- DB challenges: (Partitioned) Indices
- Graph Analytics challenges: Kernel Access Patterns

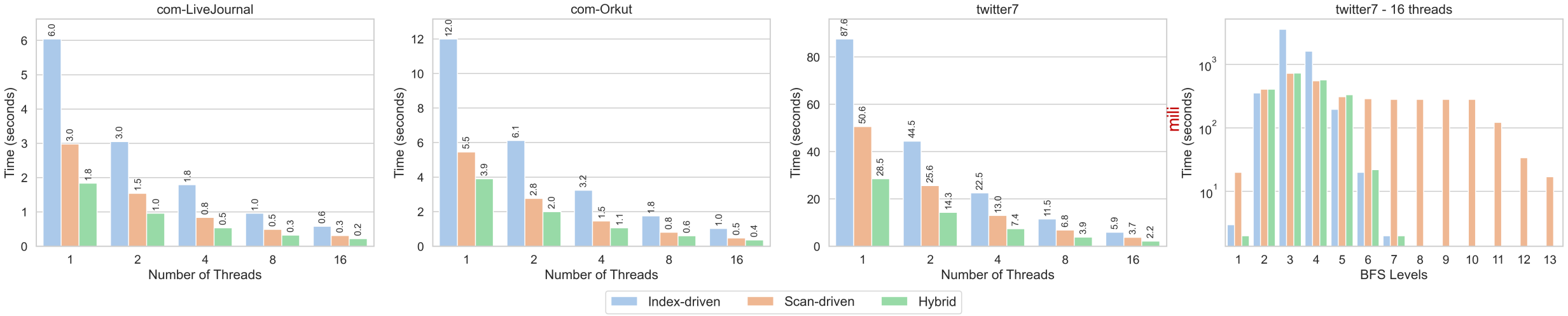
Categorizing graph algorithms



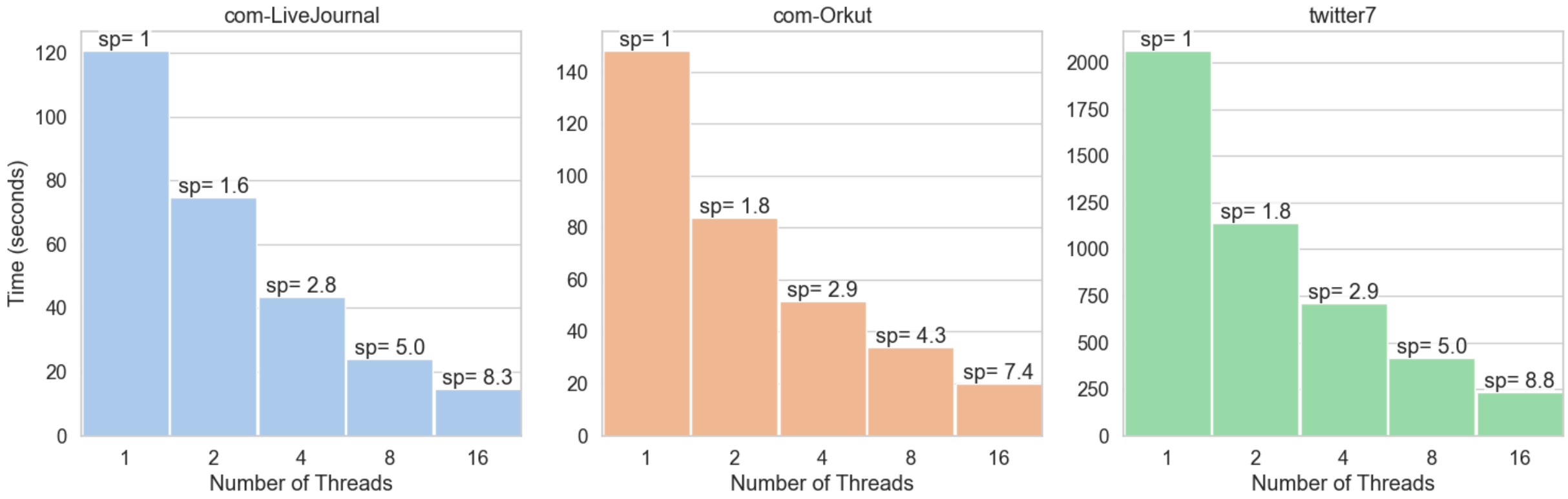
Properties of test graphs

Dataset	Number of vertices	Number of edges
com-LiveJournal	3,997,963	69,362,378
com-Orkut	3,072,441	234,370,166
twitter7	41,652,230	1,468,364,884

BFS: indexed vs scan vs hybrid



Approximate Closeness Centrality



Take away messages

- Identify and optimize common Access Patterns
  - Point lookups requires efficient indices
  - Full pass transaction/algorithms requires efficient scans
- Higher level applications may alternate between different access patterns
- Abstraction at the Access Pattern level
  - Simplifies application development
  - Enables targeted optimizations can be more easily leveraged by multiple applications
- One size never fits all, BUT we can get very close!

What about Scheduling?

- Online scheduling
- We know nothing about Customer workloads and data
  - Characteristic sets capture structural patterns in the data
- Coarse-grain Dataflow engine
- Mixed workload
  - Low-latency queries
  - Analytical queries
- Data-locality matters
  - Even in vertically-scaled (single node, multi NUMA)
  - especially when we move to horizontally-scaled (distributed)

Thank you!

Ümit V. Çatalyürek
uvc@amazon.com