ChunkGraph: Large Graph Processing with Chunk-Based Graph Representation Model

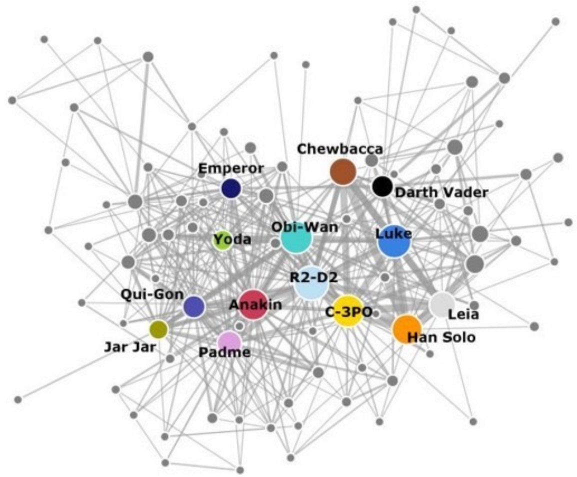
Rui Wang, **Weixu Zong**, Shuibing He, Xinyu Chen, Zhenxin Li, Zheng Dang

Zhejiang University

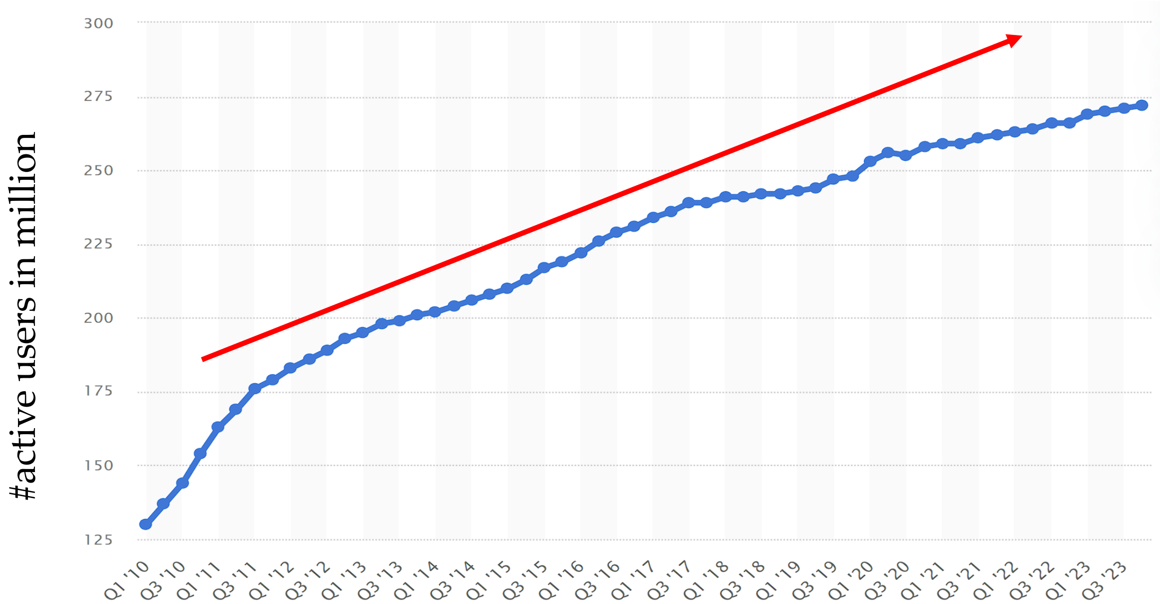
USENIX ATC 2024

Explosive Growth in Graph Data Analytics

- Graph data



- Real-world graph datasets are **continuously growing**

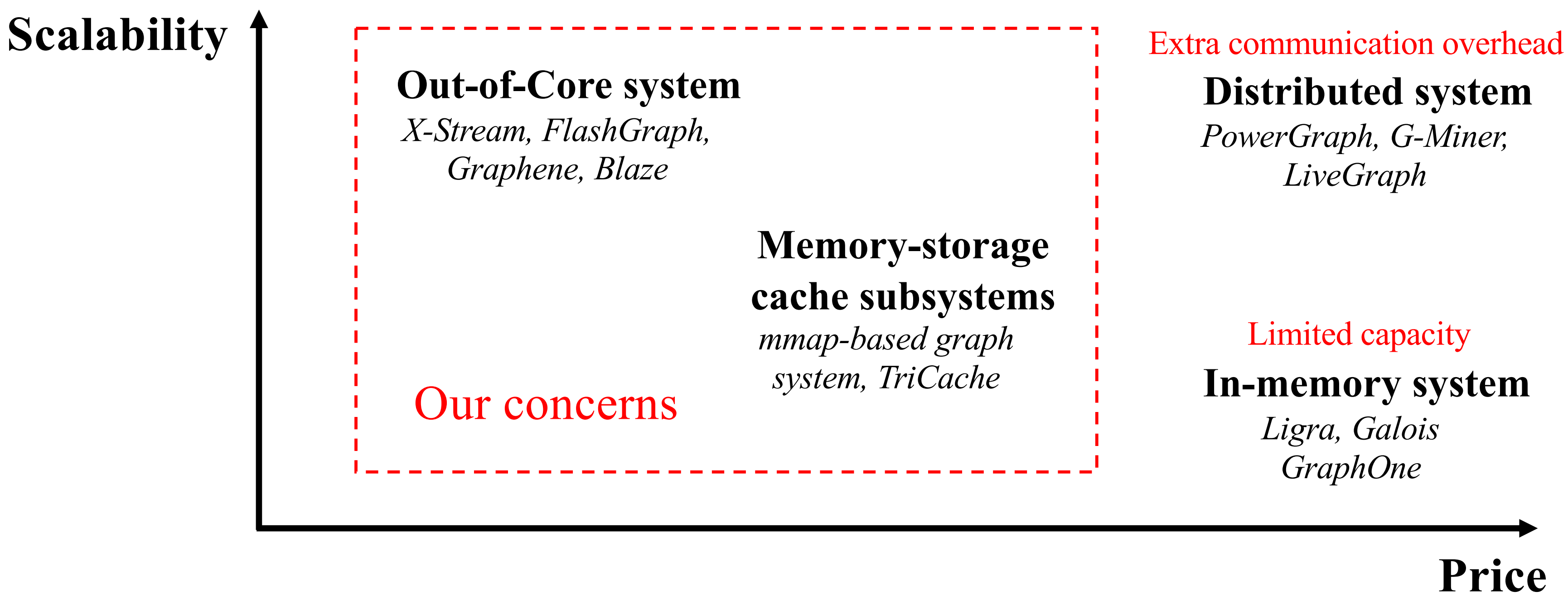


#Facebook monthly active users between 2010 and 2023, Statista

- Graph Application
  - Social networks, webpage links, recommendation systems

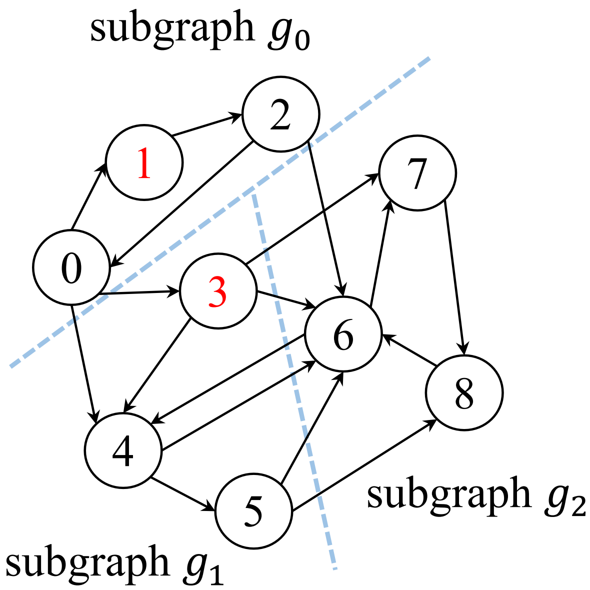
Different Graph Systems Supporting Large Graph Processing

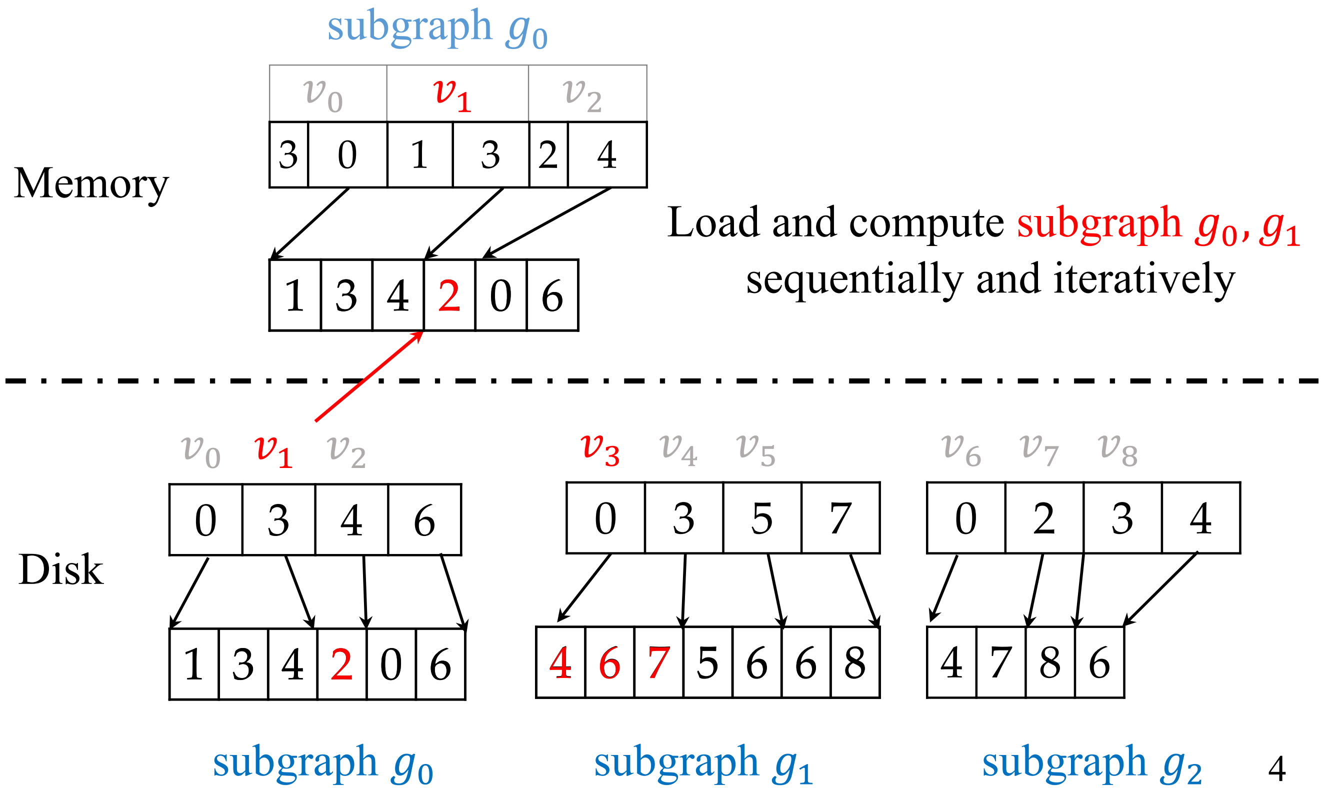
- **Objectives** for large graph processing systems:
  - **Cost-effective, Scalable, Programming-friendly**



SOTA Subgraph-based Out-of-core Graph Systems

- **Subgraph-based iterative model** divides the whole graph into **disjoint** intervals
  - Sequentially load each subgraph from disk during each iteration (e.g. access v_1,v_3)



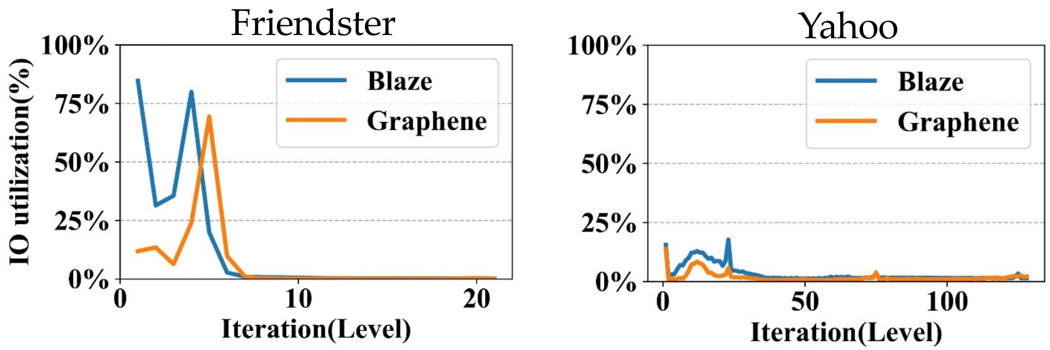


Merge multiple random I/O ops
into fewer sequential I/O ops

Limitations for Subgraph-based Iterative Model Graph Systems

P1 Low I/O efficiency

Access single vertex → load whole subgraph



Average I/O utilization is lower than 13% for BFS.

P2 Extra computing overhead

Subgraph synchronization overhead

Blaze requires up to 154x more CPU instructions compared to in-memory system Ligra's mmap variant.

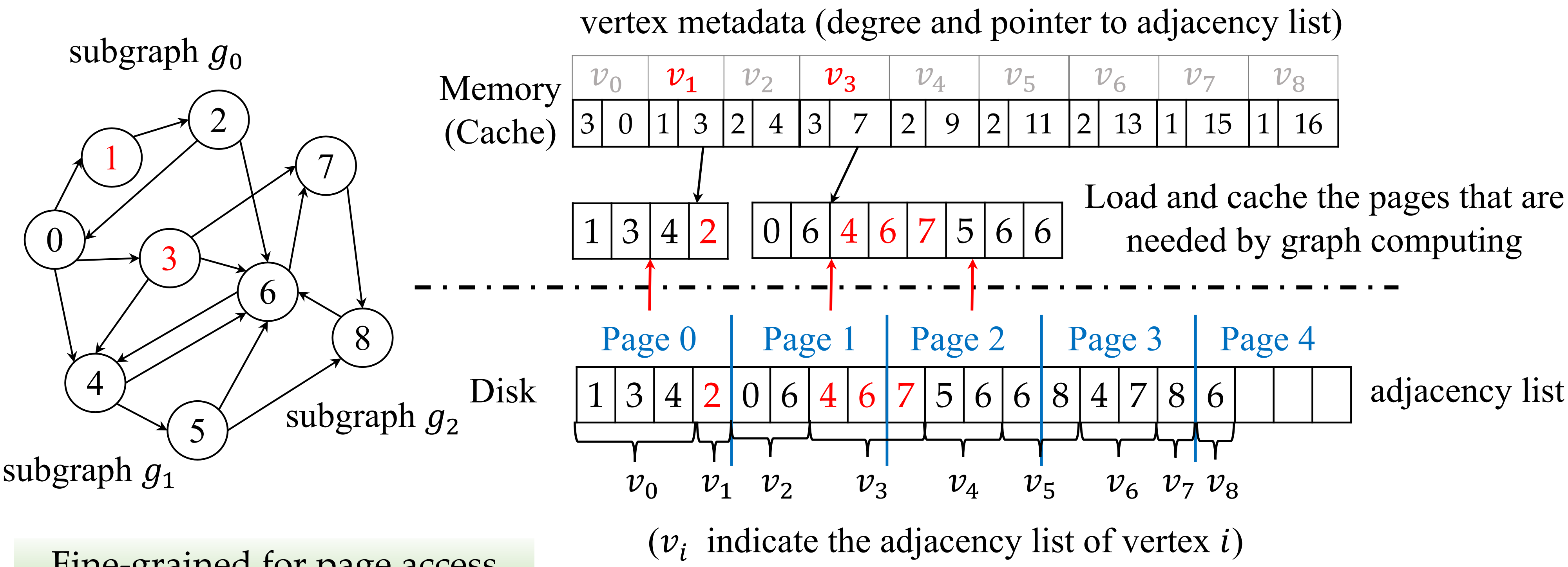
P3 Expensive algorithm development costs

Extra implementation for I/O management

#line of codes	BFS	BC	PageRank	KCore
Ligra	34	123	77	86
Graphene	420	-	763	476
Blaze	75	197	159	133

Alternative: Memory-Storage Cache Subsystems

- Using **page cache based mechanism** to cache data from external storage

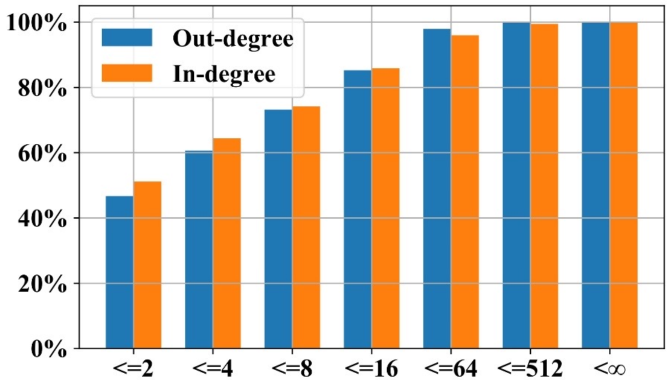


Fine-grained for page access compared to subgraph access

Limitations for Memory-Storage Cache Subsystems

- **Problem 1:** Mismatch between **page granularity** and vertex access
  - Real graph datasets behave power law degree distribution

Degree distribution of Yahoo



Low-degree vertices → **Poor I/O efficiency**

- 51.17% of non-sink vertices have only **one or two** in-neighbors

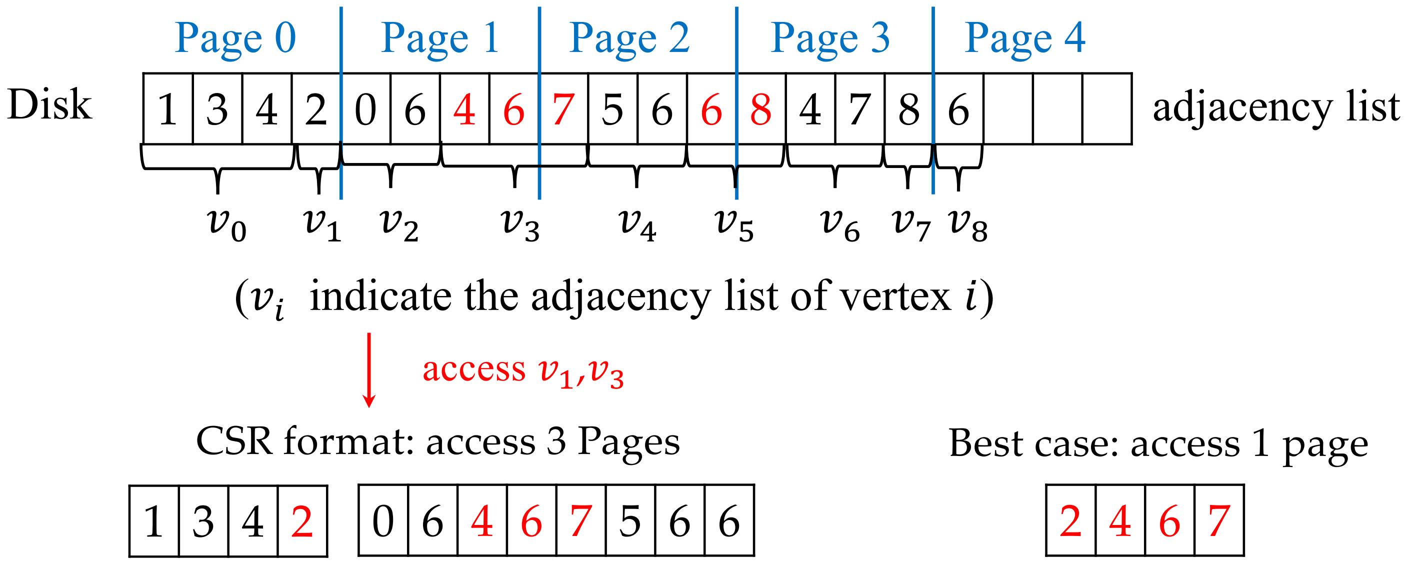
High-degree vertices → **Massive page table entries**

- 0.09% of non-sink vertices account for 58.44% of total edges
- 7459 4KB-pages are needed for largest vertex's neighbors

Solution: Use different storing strategy for vertices with different degrees

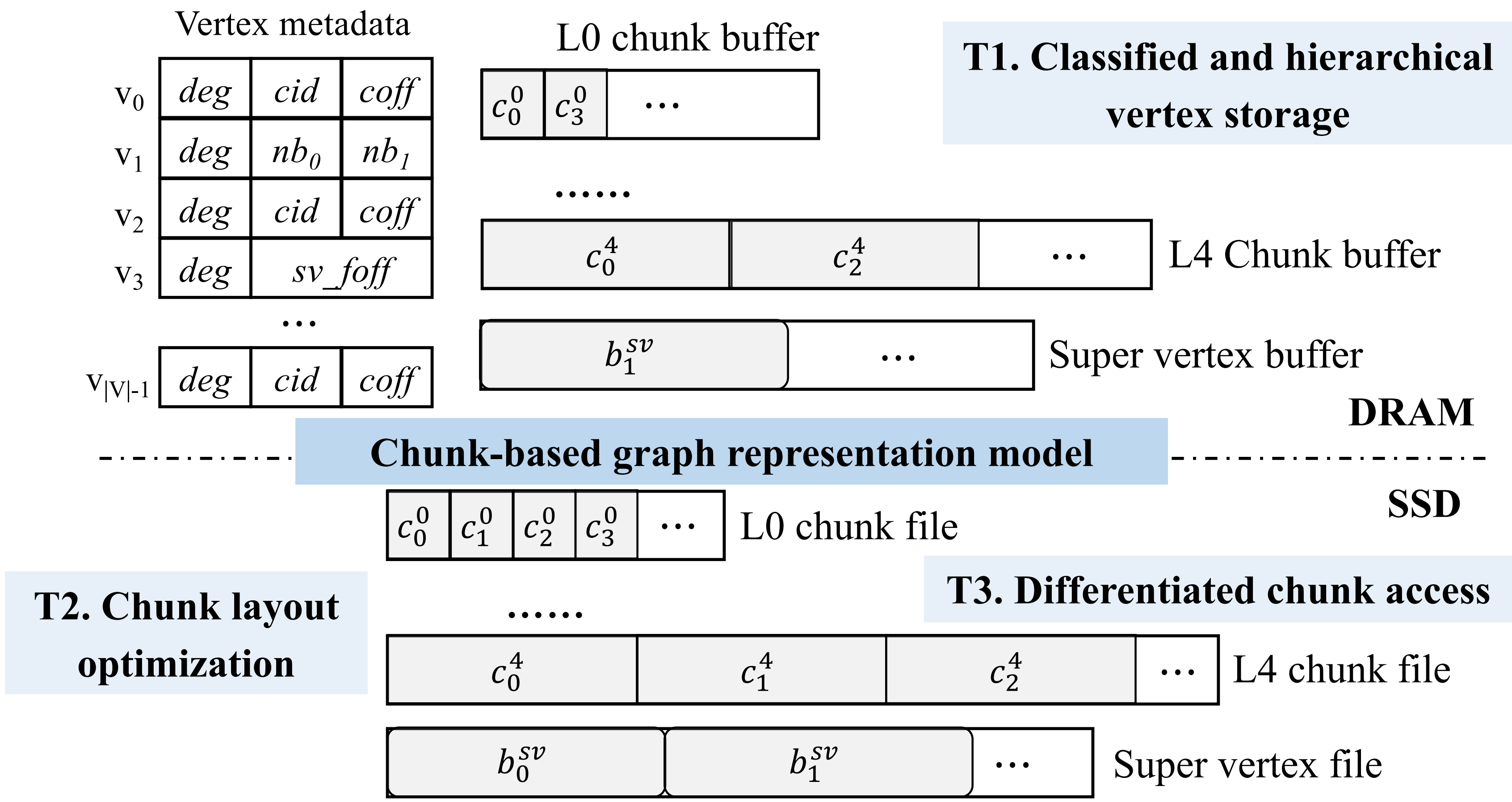
Limitations for Memory-Storage Cache Subsystems

- **Problem 2: Vertex cut**
  - A vertex smaller than one page is placed **across two adjacent pages** (e.g. v_3,v_5)



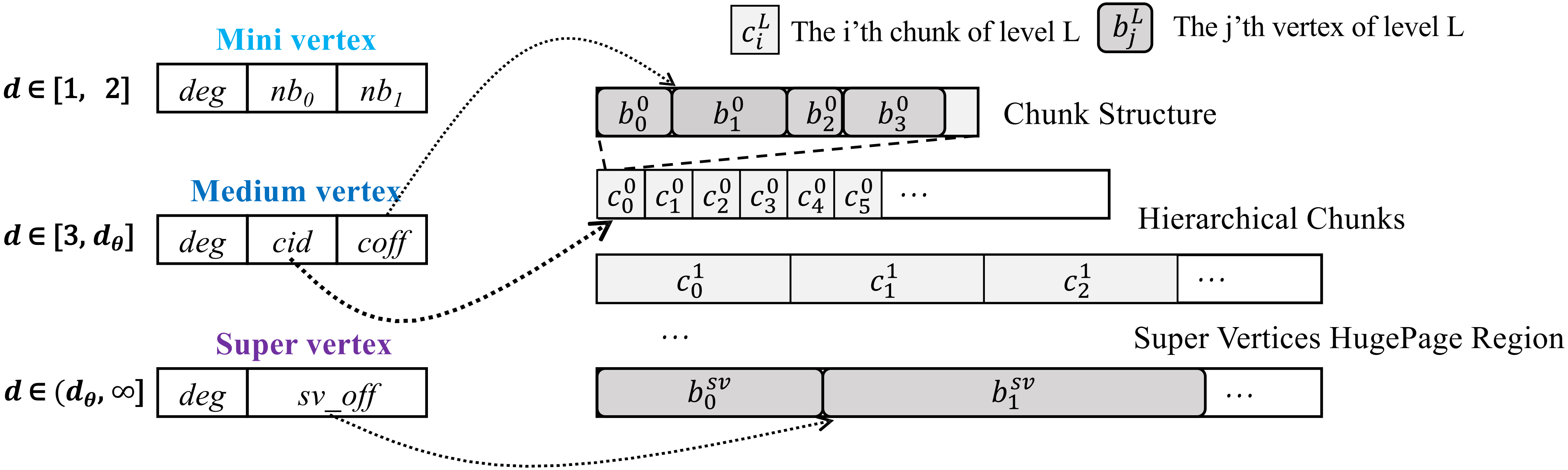
Target: Minimize the number of page accesses for each query

ChunkGraph: I/O efficient chunk-based graph representation model



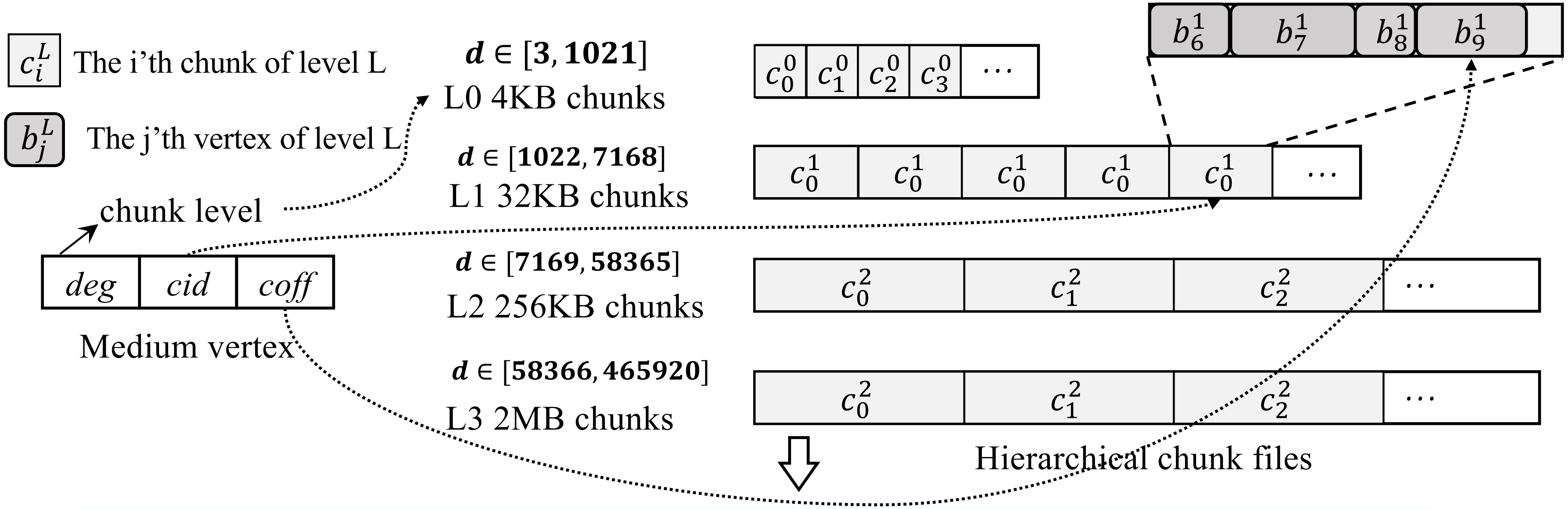
Technique 1.1: Classified Hierarchical Vertex Storage

- All vertices are classified into **three categories** according to their degrees
  - **Mini vertex: in-index storing** without additional storage cost and indirect addressing
  - **Medium vertex:** chunk-based storing **without vertex cutting issues**
  - **Super vertex: HugePage-based** storing with lower page table and TLB overhead



Technique 1.2: Hierarchical Chunk Management

- **Hierarchical** chunk size according to medium vertices' degree
  - E.g. a four-layer hierarchical chunk implementation



Differentiated chunk buffer sizes according to each layer proportion

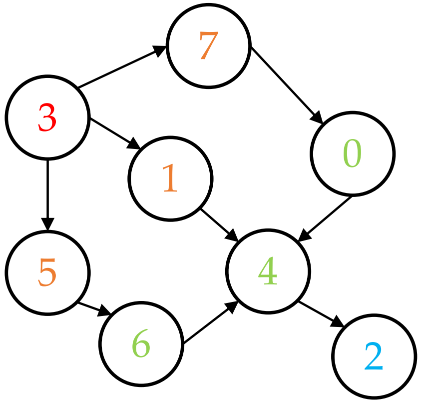
$$S_i = \frac{S_i \times M}{\sum_1^L S_j}$$

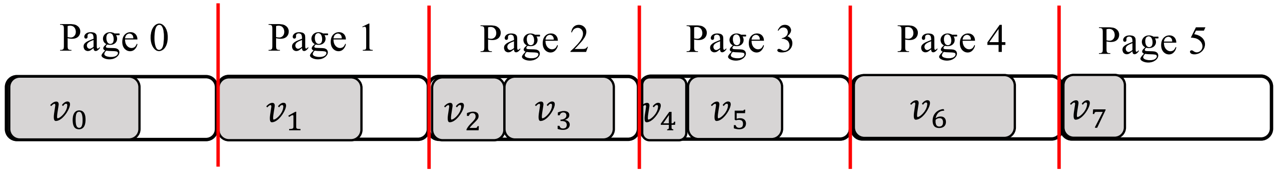
M: total available memory size

S_i: chunk file size of layer i

Technique 2.1: Reordering based Chunk Layout Optimization

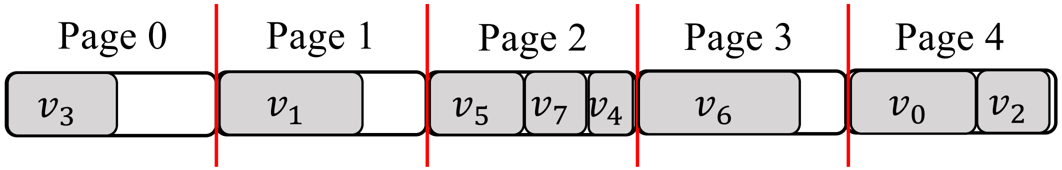
- Observation: A vertex is likely to be accessed after its neighbors or sibling vertices accessed
  - E,g. Run BFS on root 3. (Vertex access order: 3, 1, 5, 7, 4, 6, 0, 2)





(a) Current vertex-id based chunk layout

Iter1: P2
Iter2: P1, P3, P5
Iter3: P0, P3, P4
Iter4: P2



(b) Reordering based chunk layout

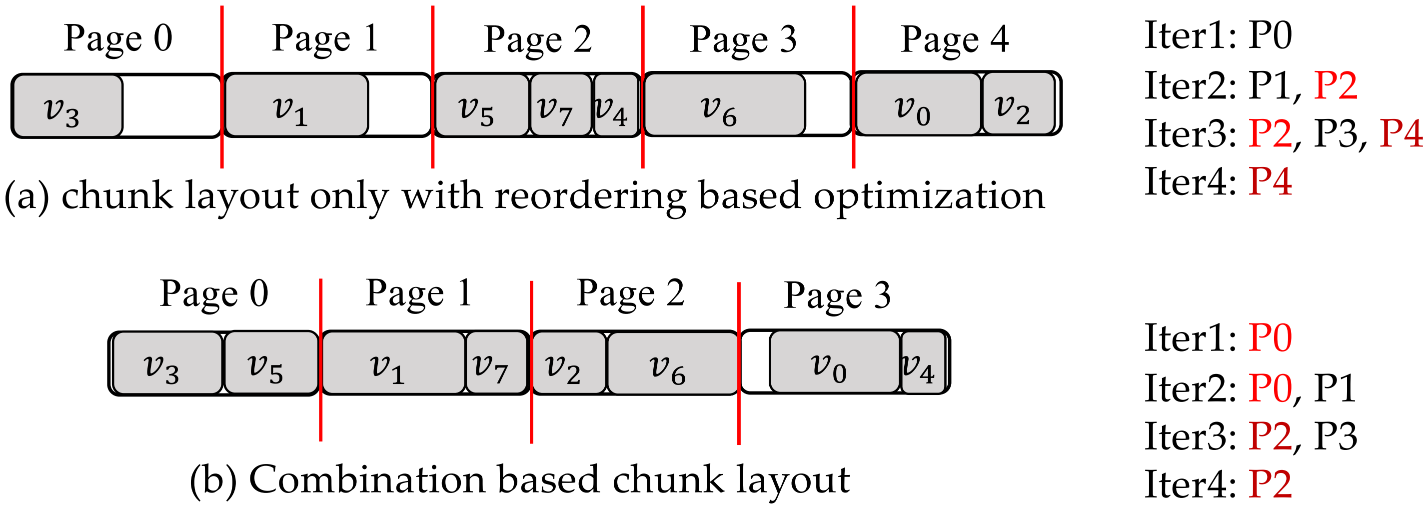
Iter1: P0
Iter2: P1, P2
Iter3: P2, P3, P4
Iter4: P4

Better temporal locality, buffer cache hit ratio, providing opportunities for **sequential I/O**

Problem: reordered optimization still suffers from **inter-fragmentation** within chunk

Technique 2.2: Vertex-combination Chunk Layout Optimization

- Solution: combine the vertices **with complementary degree** into one chunk

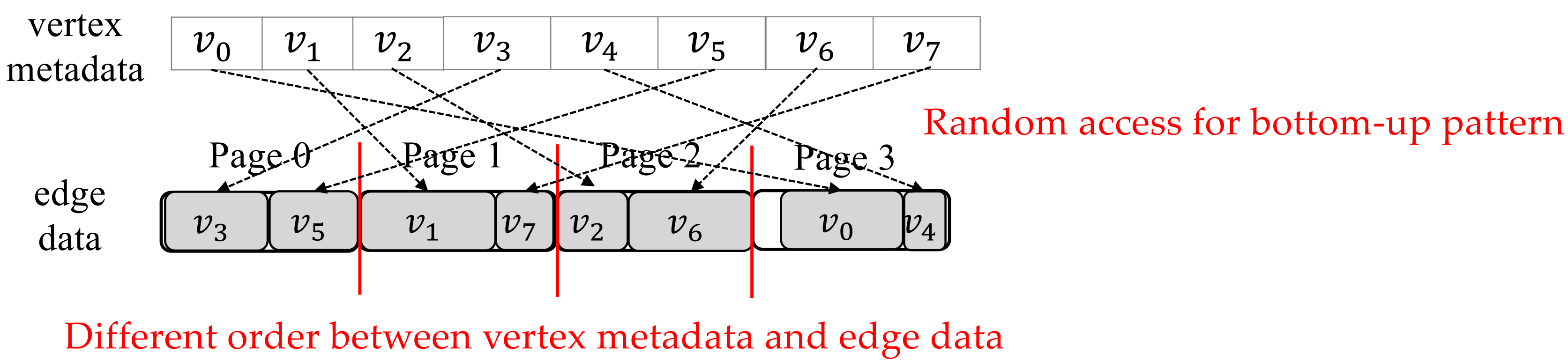


Less intra-chunk fragmentation and **better spatial locality, minimizing chunk access**

#Fragmentation chunk is reduced from **95.24% to 52.77%** on YahooWeb dataset.

Technique 3: Differentiated Chunk Access Optimization

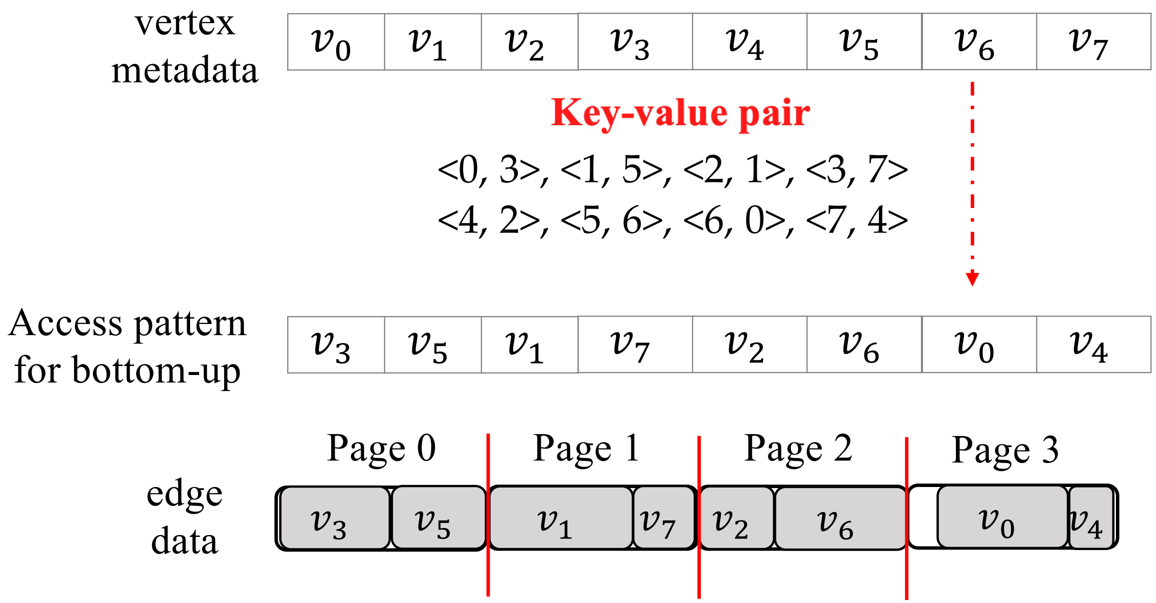
- Graph algorithms usually involves different graph access pattern
  - **Top-down**: sparse access, only activated vertices
  - **Bottom-up**: dense access, scan the whole graph in vertex ID order



For bottom-up algorithm, we should traverse all vertices according to **edge data order**

Technique 3: Differentiated Chunk Access Optimization

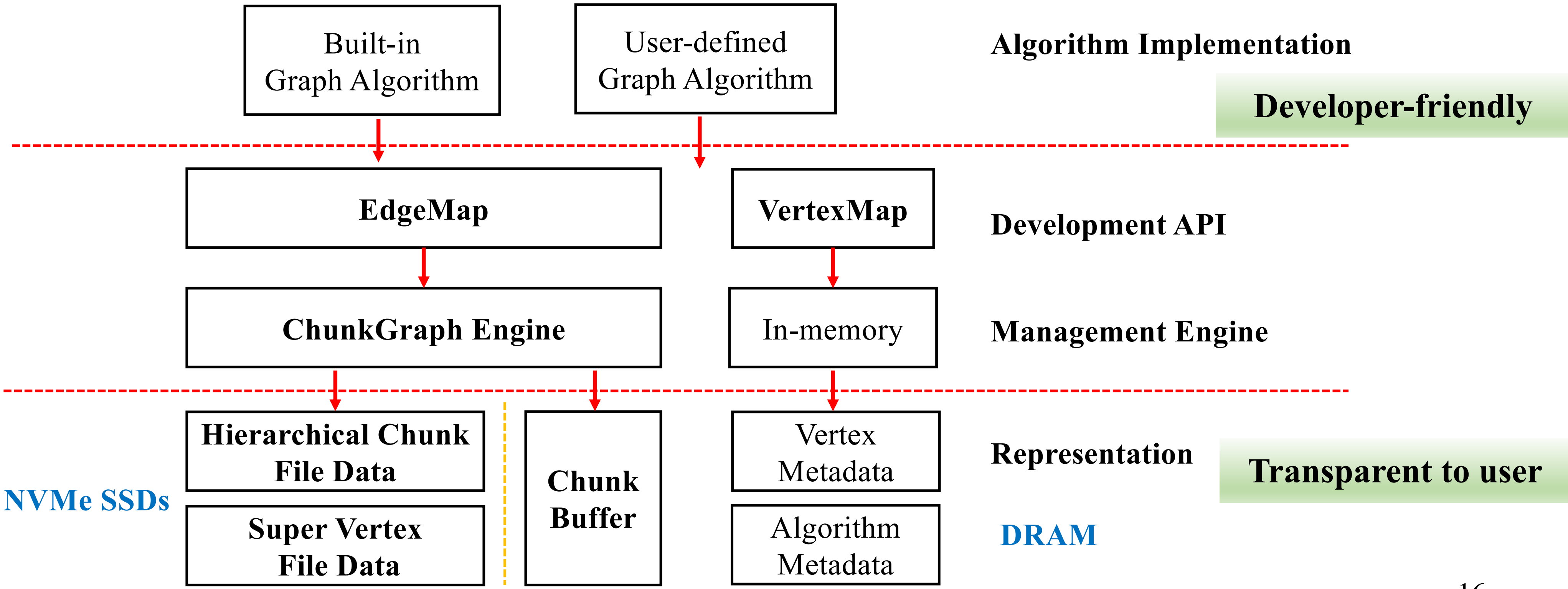
- **Differentiated chunk access** pattern for bottom-up access
  - Store **key-value pair** <reordered_id, vid> to support chunk order access



Avoid random access for vertices due to reordering optimization

Prototype System and Implementations

- **ChunkGraph** is implemented based on Ligra's graph interface



Evaluation settings

- Testbed
  - A server with 2 sockets, each with 24 physical cores
  - **8 * 16GB = 128GB** DRAM + **2 * 3.84TB** SSD

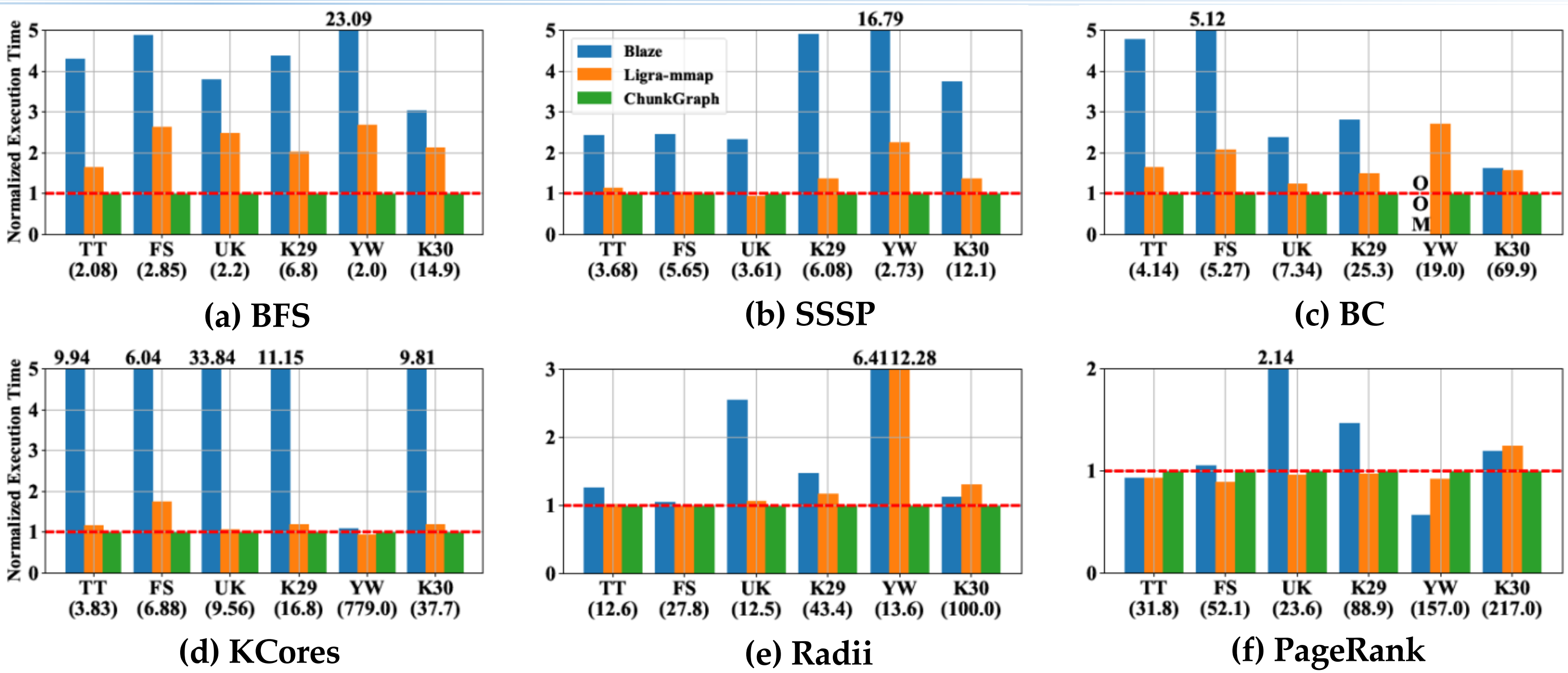
- Graph datasets

Dataset	$\|V\|$	$\|E\|$	CSR Size	Chunk Size	
Twitter (TT)	61.6M	1.5B	11.9GB	13.5GB	Real World Graph
Friendster (FS)	68.3M	2.6B	20.3GB	21.2GB	Real World Graph
UKdomain (UK)	101.7M	3.1B	26.2GB	27.5GB	Real World Graph
YahooWeb (YW)	1.4B	6.6B	70.5GB	77.8GB	Real World Graph
Kron29 (K29)	512M	8B	72GB	78.2GB	Synthetic Graph
Kron30 (K30)	1B	16B	144GB	156.3GB	Synthetic Graph

Evaluation settings

- Comparison systems
  - **Blaze**
    - ✓ **The SOTA out-of-core graph system** optimized for modern fast SSDs
  - **Ligra-mmap**
    - ✓ **Ligra's variant using mmap** to map the graph data files into the virtual memory space
- Evaluation metrics
  - Graph query performance
    - ✓ BFS, SSSP, BC, Kcores, Radii, PageRank
  - I/O overhead
  - Computation overhead

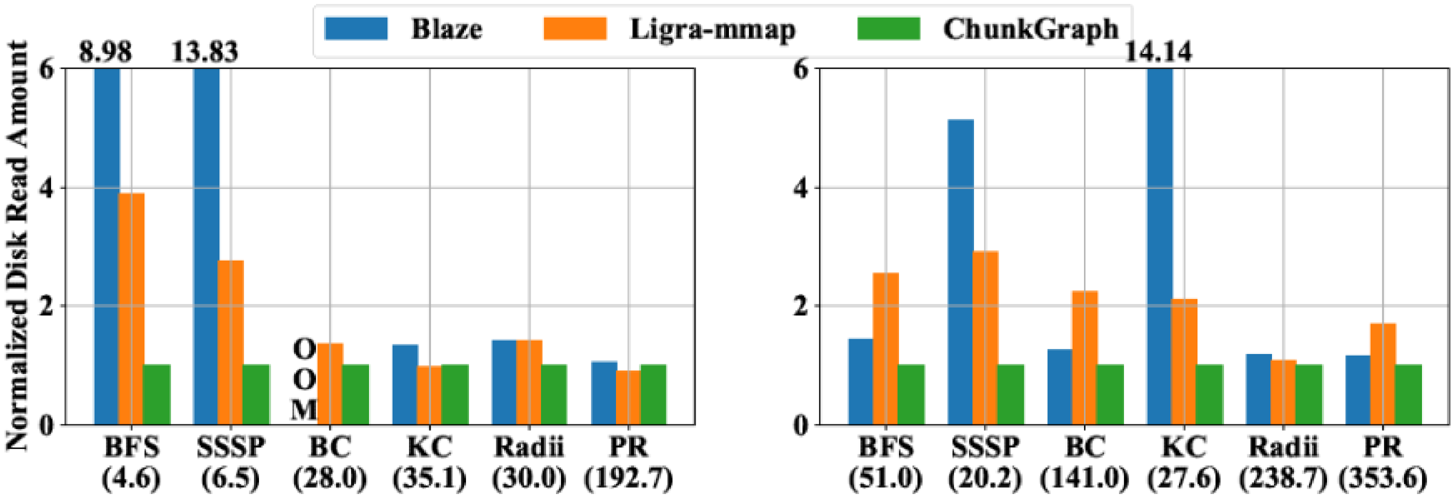
Evaluation 1: Graph query performance



ChunkGraph achieves **1.62x-23.09x** speedup upon Blaze, and **1.08x-2.94x** compared to Ligra-mmap on sparsely accessed algorithms **BFS, SSSP, BC.**

Evaluation 2: I/O overhead

- Disk read amount of different algorithms on Yahoo and Kron30

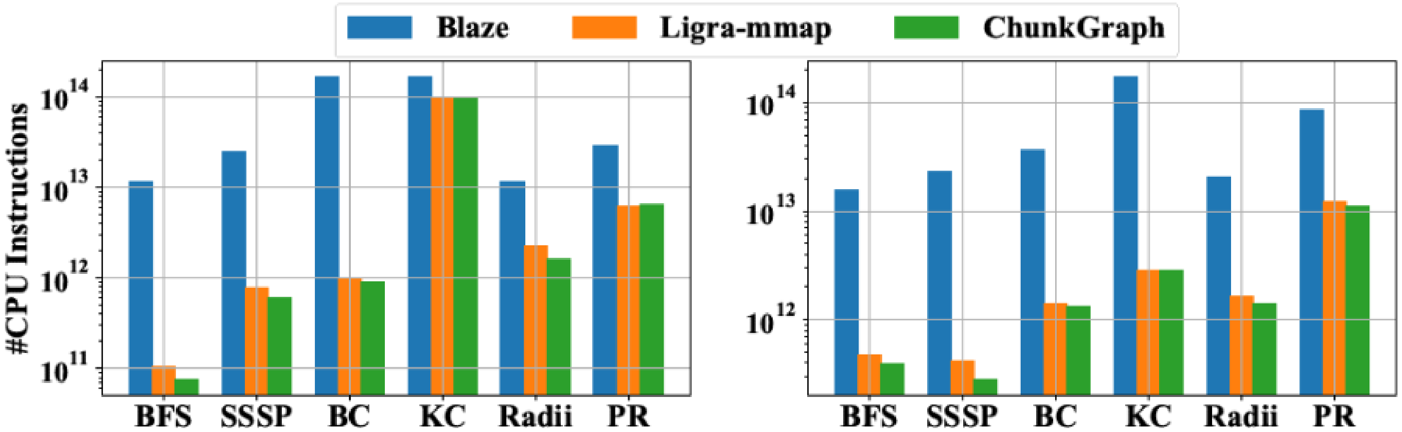


(a) Yahoo (b) Kron30

ChunkGraph reduces disk read amount by **4.68×** and **1.98×** on average, compared to Blaze and Ligra-mmap respectively.

Evaluation 3: Computation overhead

- CPU instructions executed during different algorithms' execution



(a) Yahoo (b) Kron30

ChunkGraph reduces the number of CPU instructions by **185.01×** compared to the external memory graph system Blaze.

Conclusion

- **ChunkGraph**: an I/O efficient external graph system for processing large-scale graphs
  - **Classified and hierarchical** vertex storage strategy
  - Chunk layout optimization based on **vertex reordering and combination**
  - **Differentiated chunk access** optimization
  - Encompass both out-of-core systems and memory-storage cache subsystems

- More evaluation results and analysis are in the paper
- The source code is at https://github.com/ZoRax-A5/ChunkGraph

Thanks for your attention!

Weixu Zong @ ZJU
zorax@zju.edu.cn