SpZip: Architectural Support for Effective Data Compression In Irregular Applications

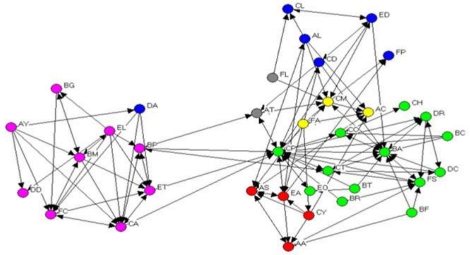
Yifan Yang, Joel S. Emer, Daniel Sanchez

ISCA 2021

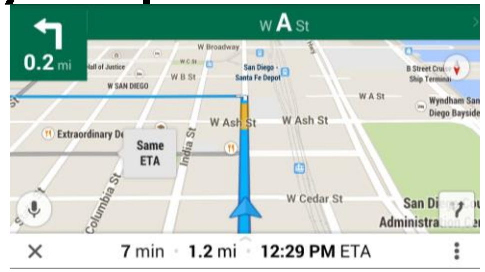
Session 12B (June 16, 2021 at 8 PM EDT)

Irregular applications are memory bound

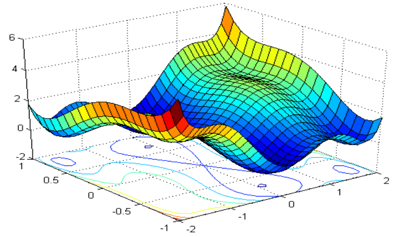
- Irregular applications, such as graph analytics and sparse linear algebra, are an increasingly important workload domain



Social network analysis



Navigation



Scientific computing

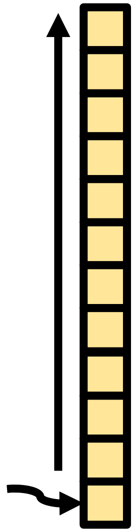
- Irregular applications are often *memory bound*

- Data compression is an attractive approach to accelerate irregular applications

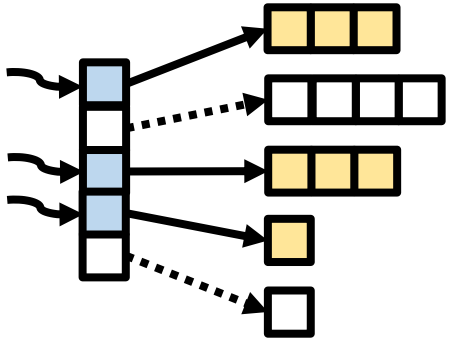
Existing data compression solutions are not tailored for irregular applications

Hardware compression units for **sequentially accessed long streams**

e.g., IBM z15 [ISCA'20]

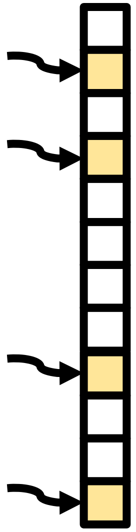


This work is optimized for **indirect, data-dependent accesses to short streams**



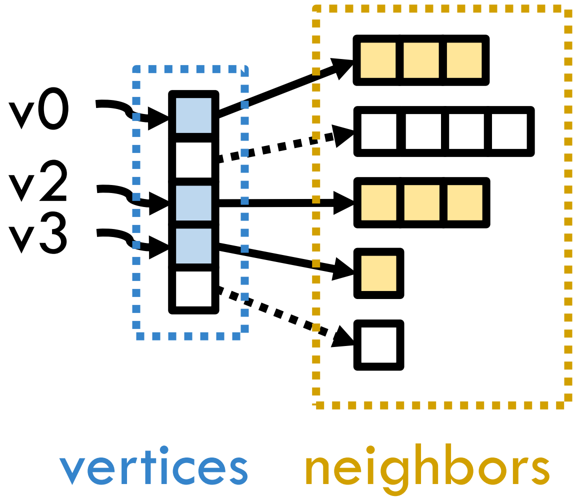
Compressed memory hierarchies support **random accesses**

e.g., VSC [ISCA'04]



Understanding access patterns in irregular applications

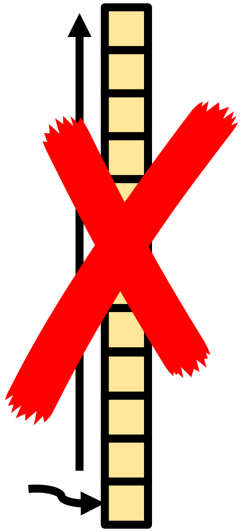
Graph Adjacency List



Existing data compression solutions are not tailored for irregular applications

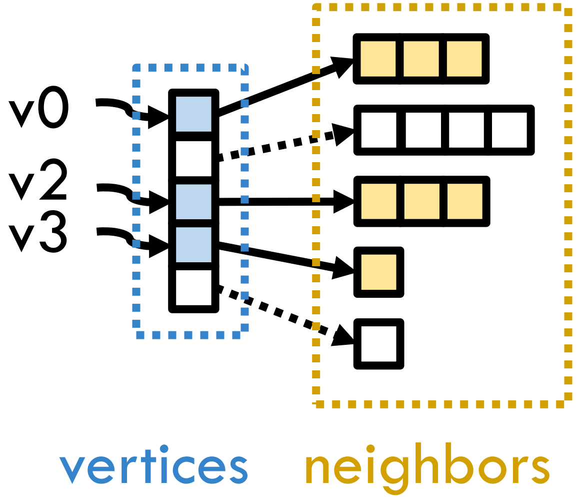
Hardware compression units for **sequentially accessed long streams**

e.g., IBM z15 [ISCA'20]



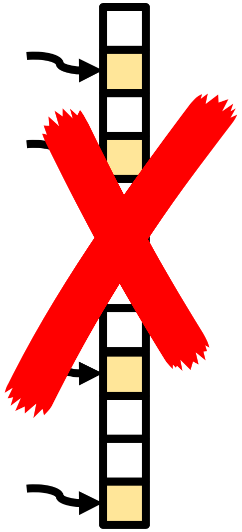
Limited compression gain on short streams

This work is optimized for **indirect, data-dependent accesses to short streams**



Compressed memory hierarchies support **random accesses**

e.g., VSC [ISCA'04]

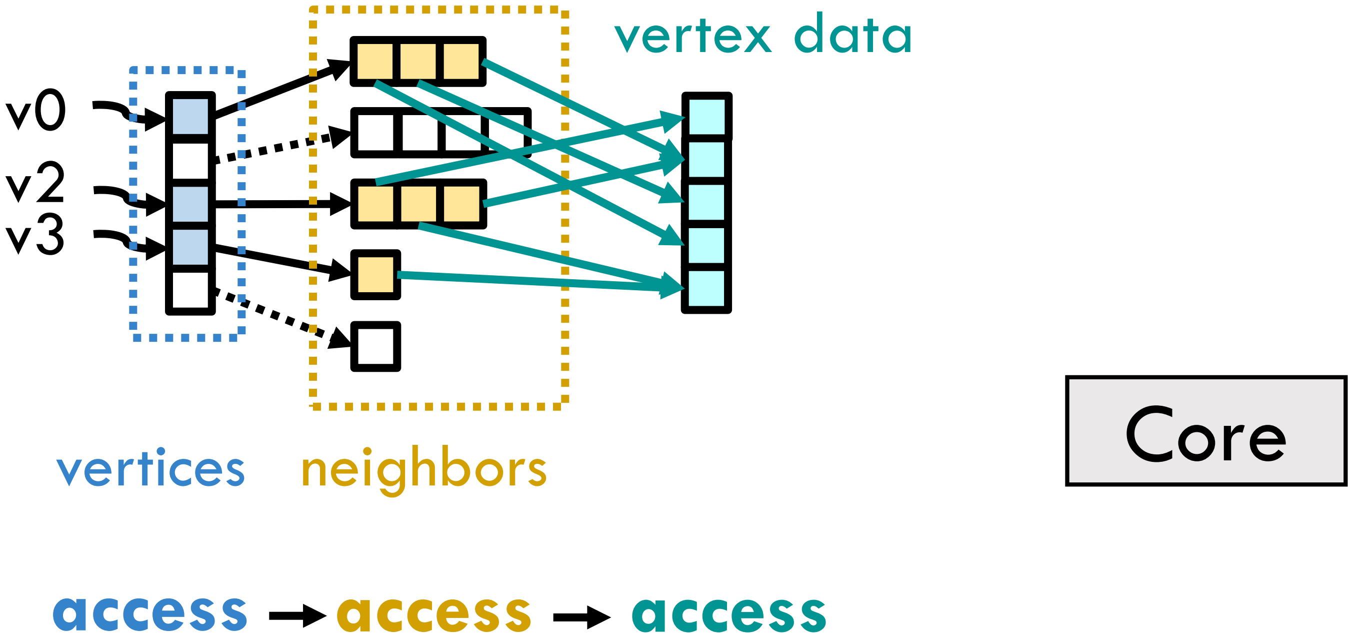


Data decompression increases critical path latency

Compressing data structures in irregular applications is hard

- Challenge 1: Access and decompression are interleaved

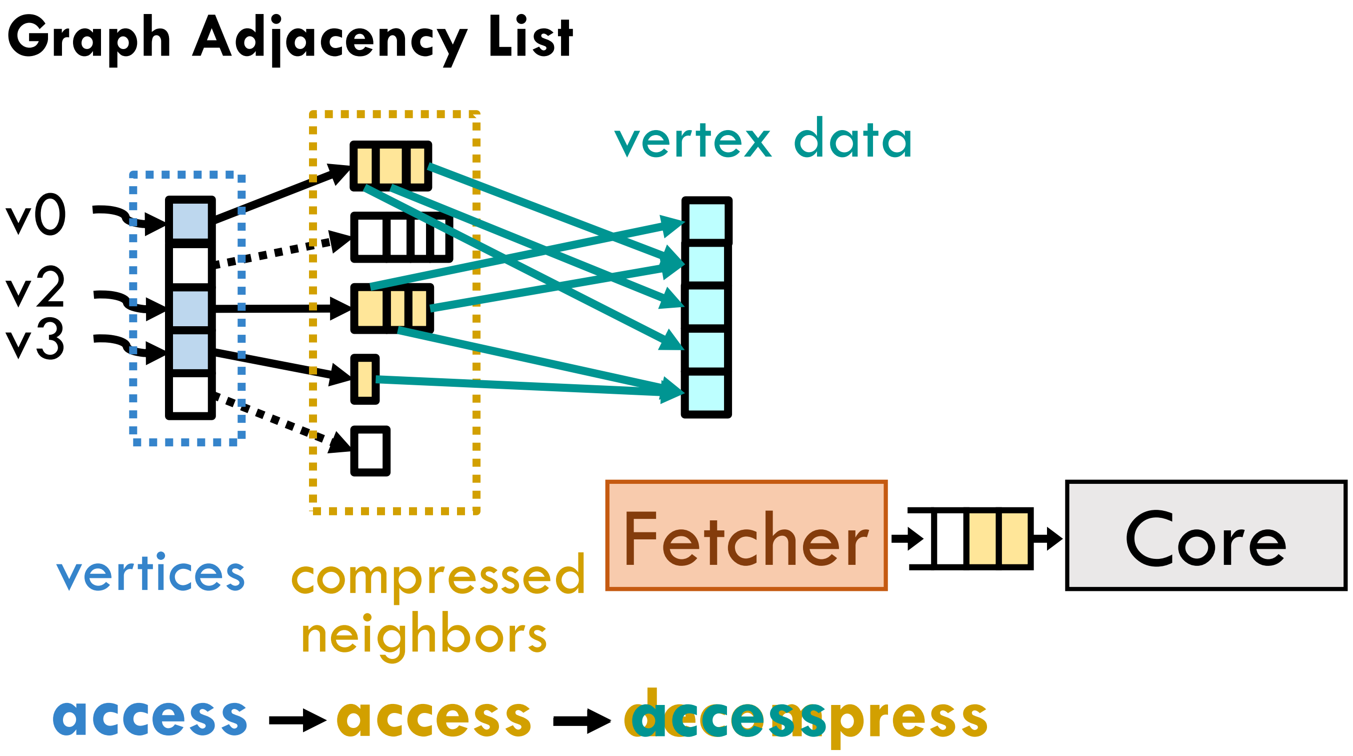
Graph Adjacency List



Compressing data structures in irregular applications is hard

- Challenge 1: Access and decompression are interleaved

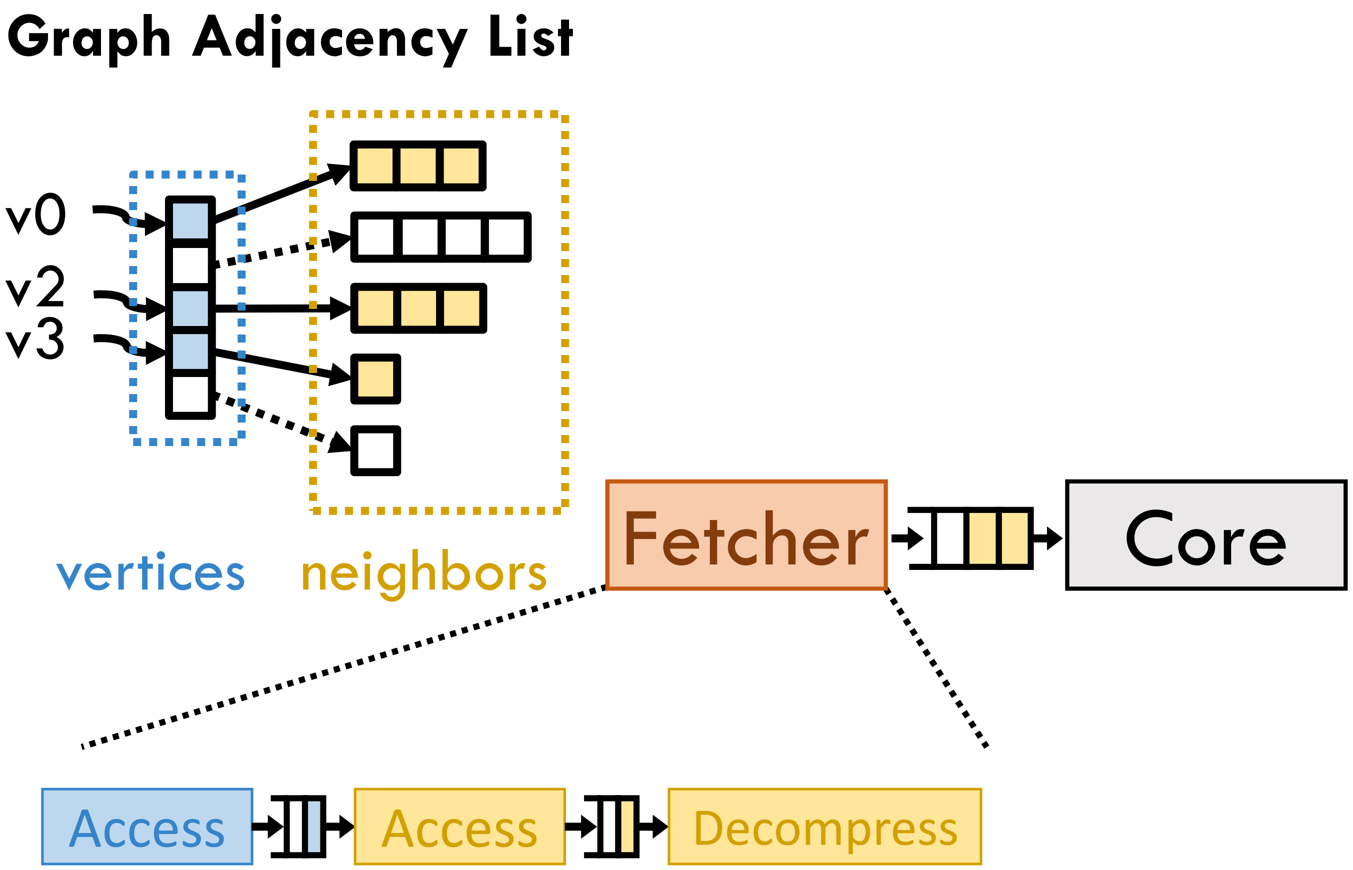
- Insight 1: Specialized hw to accelerate data access and decompression
  - Exploit **decoupled execution** to hide memory access and decompression latencies



Compressing data structures in irregular applications is hard

- Challenge 2: Need to support various access patterns and compression formats

- Insight 2: **Programmable** hardware
  - A pipeline consists of a set of **composable** operators expressing the traversal and decompression of data structures
  - Dataflow Configuration Language (DCL)

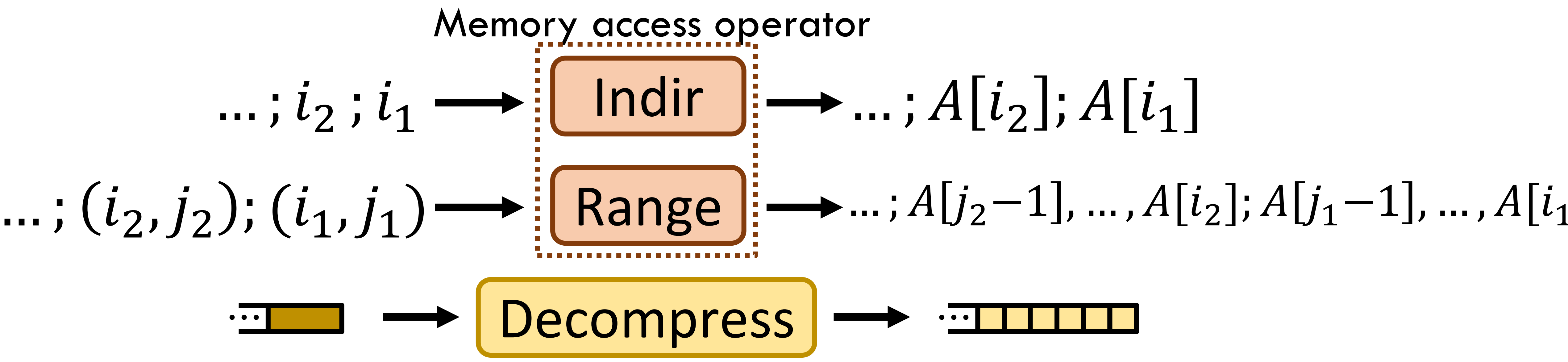


Agenda

- Motivation
- SpZip Dataflow Configuration Language (DCL)
- SpZip Design
- Evaluation

Dataflow Configuration Language (DCL) overview

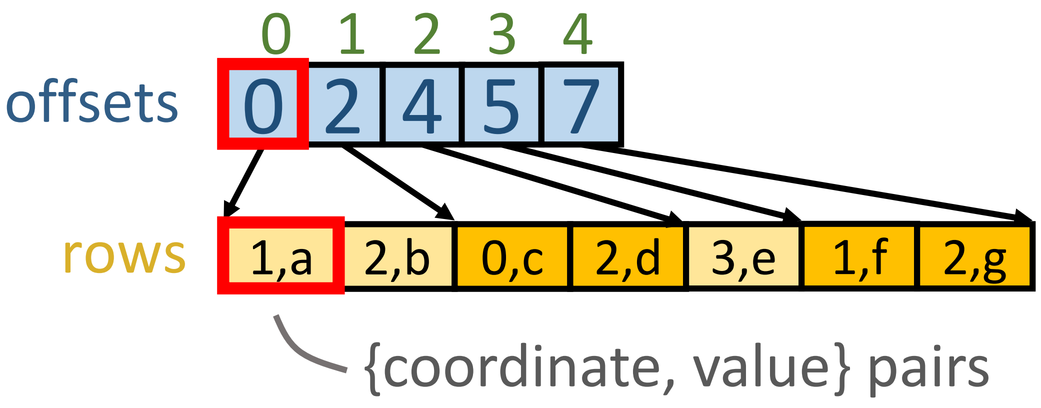
- A DCL program expresses the traversal, decompression and compression of data structures in irregular applications
- DCL program is an acyclic graph of composable operators
- Operators are connected by queues to exploit pipeline parallelism



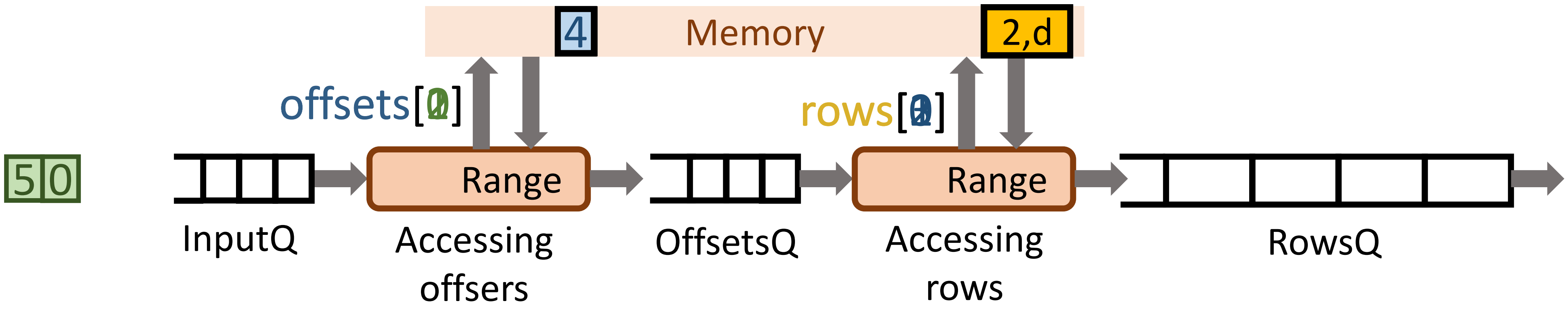
Traversing a sparse matrix in DCL

rows \ cols	0	1	2	3
0	0	a	b	0
1	c	0	d	0
2	0	0	0	e
3	0	f	g	0

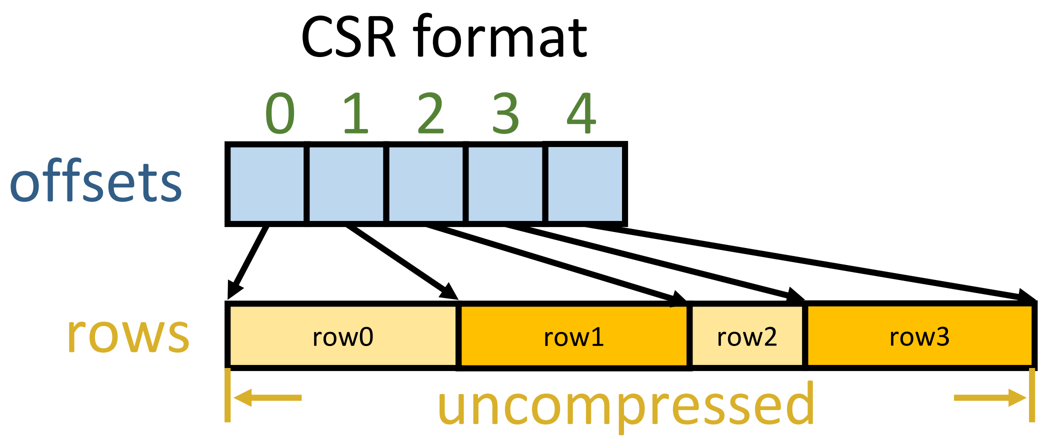
Compressed Sparse Row (CSR) format



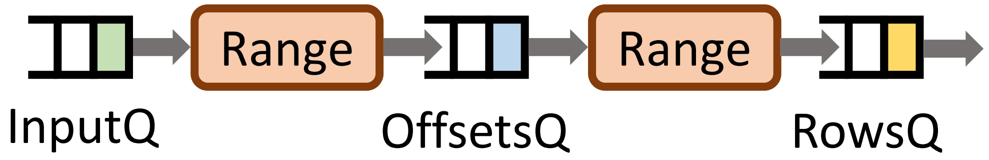

```
for i in range(numRows):                          in DCL
  for {col, val} in rows[offsets[i]:
                         offsets[i+1]]:
      visit({col, val})                           in Core
```



Traversing a sparse matrix in DCL

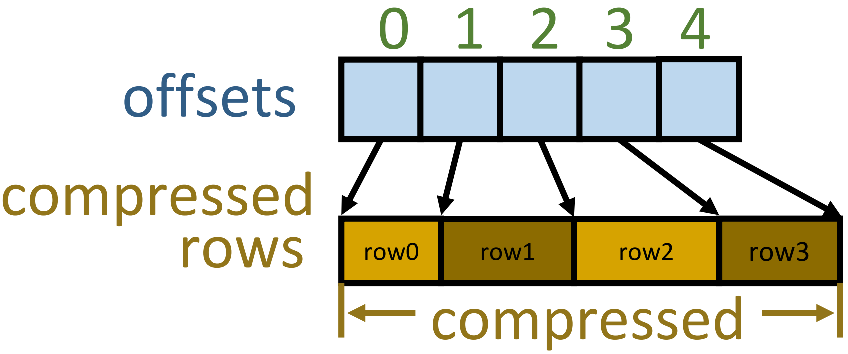



```
for i in range(numRows):                          in DCL
  for {col, val} in rows[offsets[i]:
                         offsets[i+1]]:
    visit({col, val})                             in Core
```

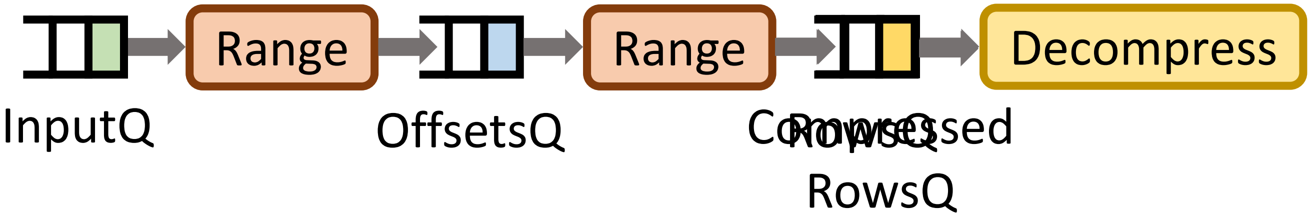


Data decompression support in DCL

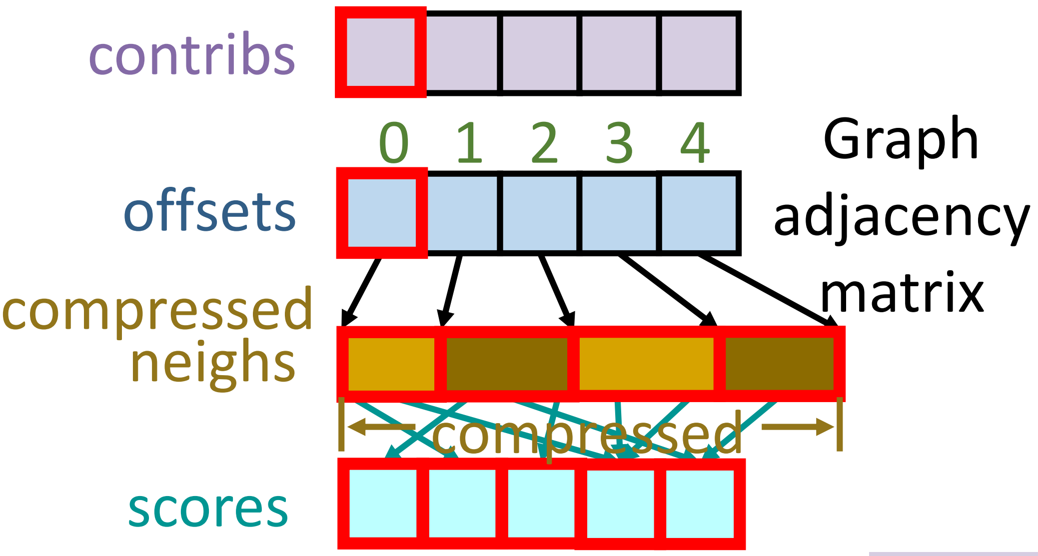
CSR with individually compressed rows



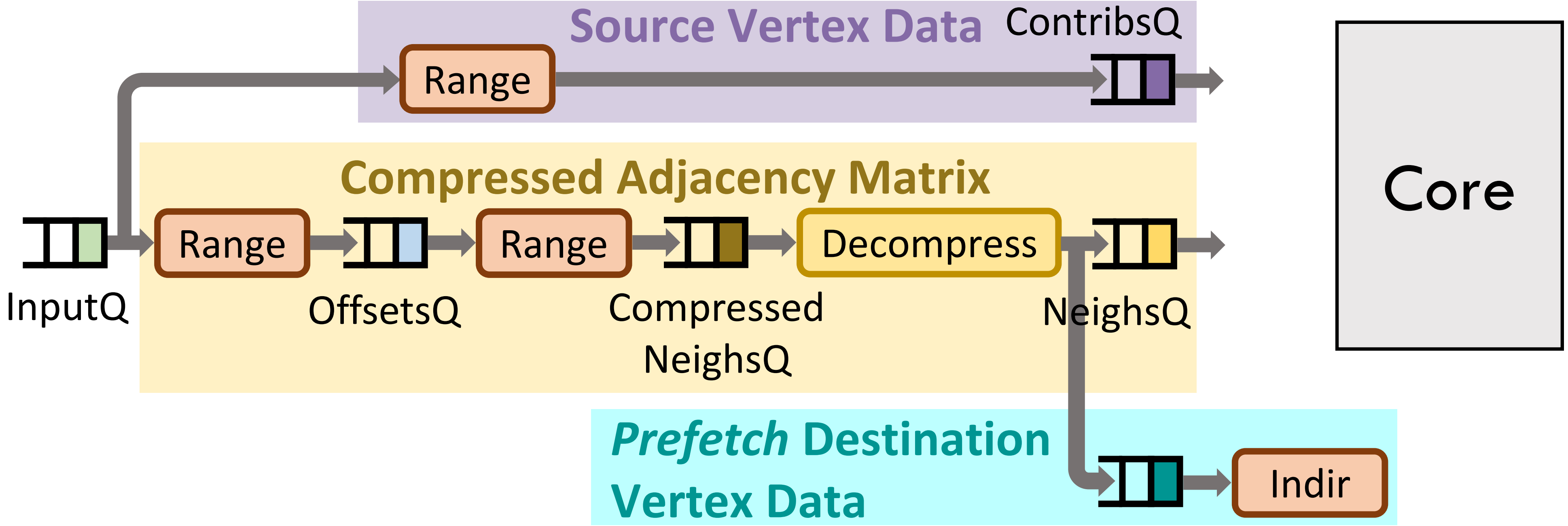

```
for i in range(numRows):                      in DCL
  for {col, val} in rows[offsets[i]:
                         offsets[i+1]]:
    visit({col, val})                         in Core
```



Using DCL in PageRank traversing multiple data structures




```
def PageRankIter(Graph g, Array scores,
                          Array contribs)
  for src in range(g.numVertices):
    for dst in decompress(g.neighs[g.offsets[src]:
                                   g.offsets[src+1]]):
      scores[dst] += contribs[src]
```

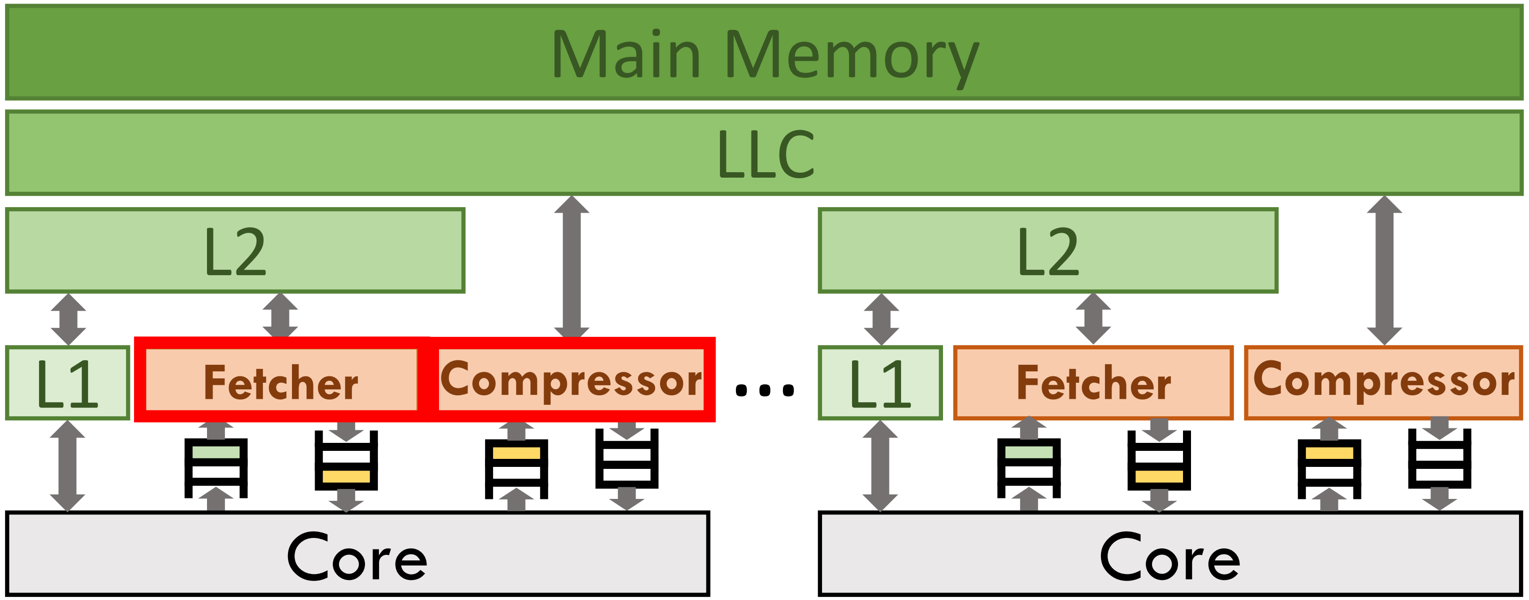


Agenda

- Motivation
- SpZip Dataflow Configuration Language (DCL)
- SpZip Design
- Evaluation

SpZip Overview

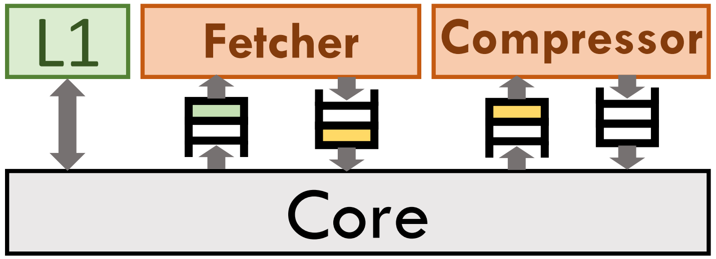
- SpZip augments each CPU core with a programmable fetcher and compressor



- The fetcher accelerates data structure traversal and decompression
- The compressor compresses newly generated data before storing it off-chip
- Fetcher and compressor issue conventional cache line requests

SpZip exploits decoupled execution

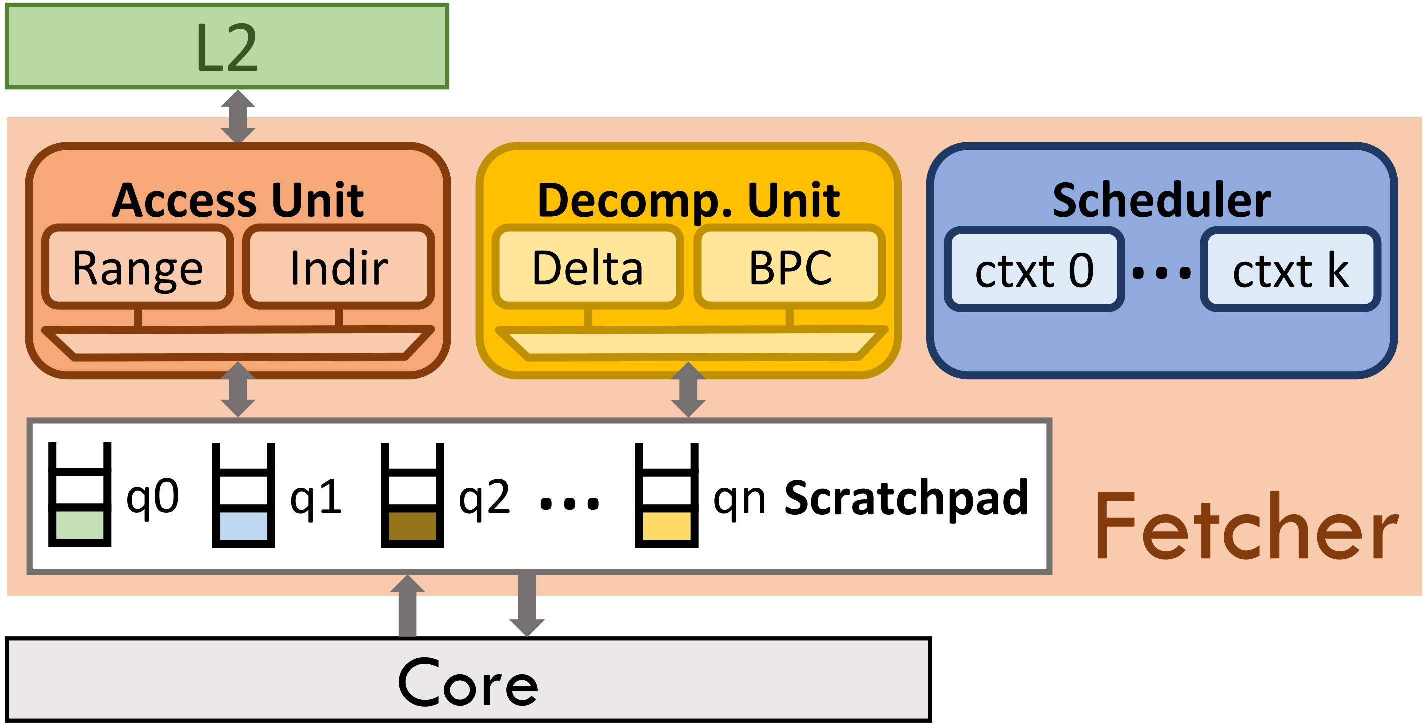
- The fetcher and compressor communicate with core through queues to exploit decoupled execution



- The fetcher runs ahead of the core to traverse and decompress data, hiding memory access and decompression latencies

SpZip fetcher microarchitecture

- Access Unit and Decompression Unit implement DCL operators
- Scratchpad holds queues between operators
- Queues between operators allow pipeline parallelism

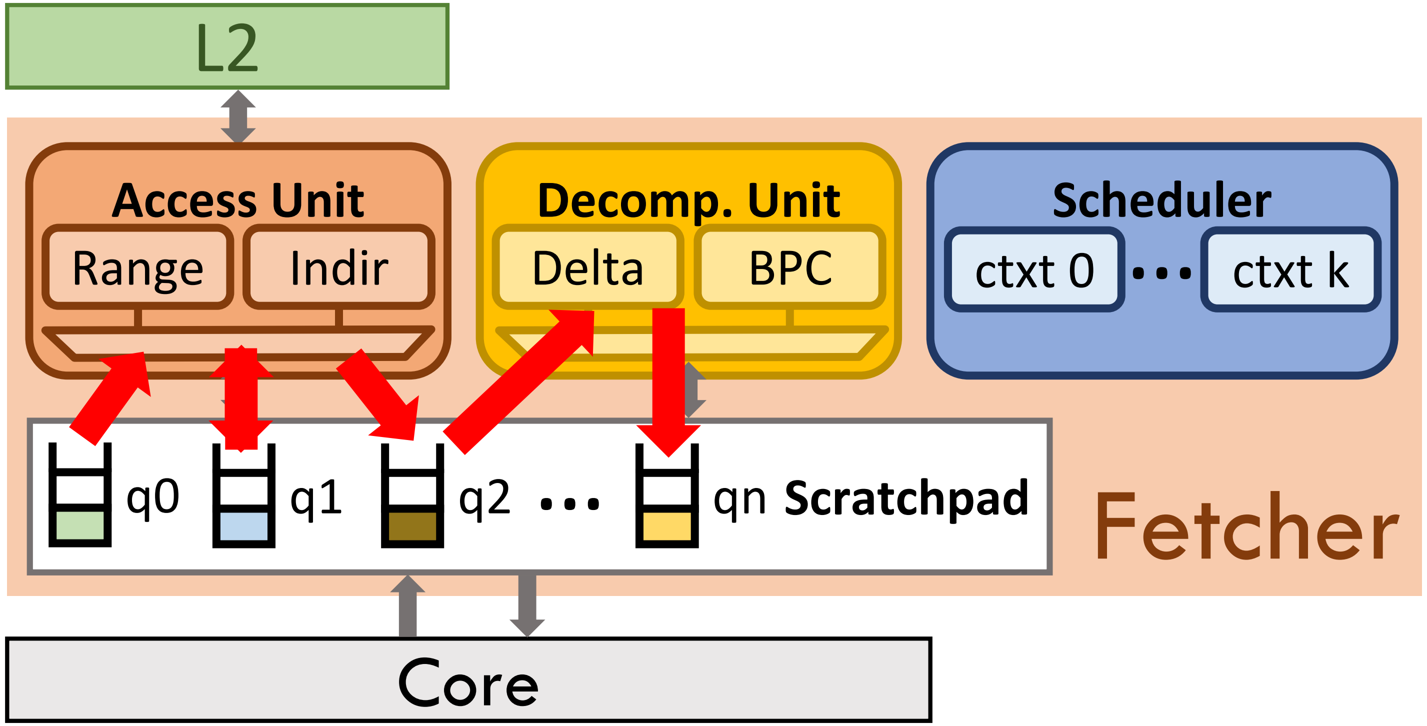


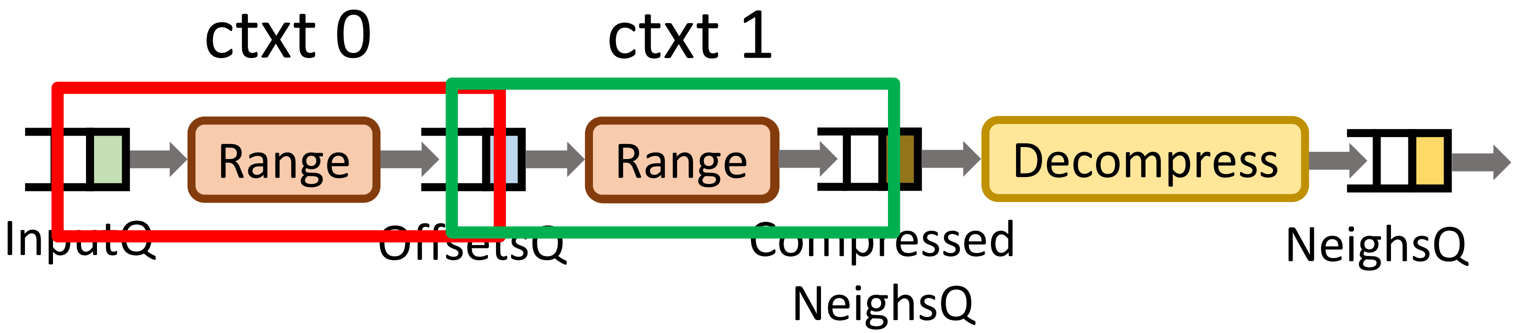
SpZip fetcher is programmable

- Scratchpad is configurable to support variable numbers and sizes of queues

- DCL operators are time-multiplexed on the same physical unit

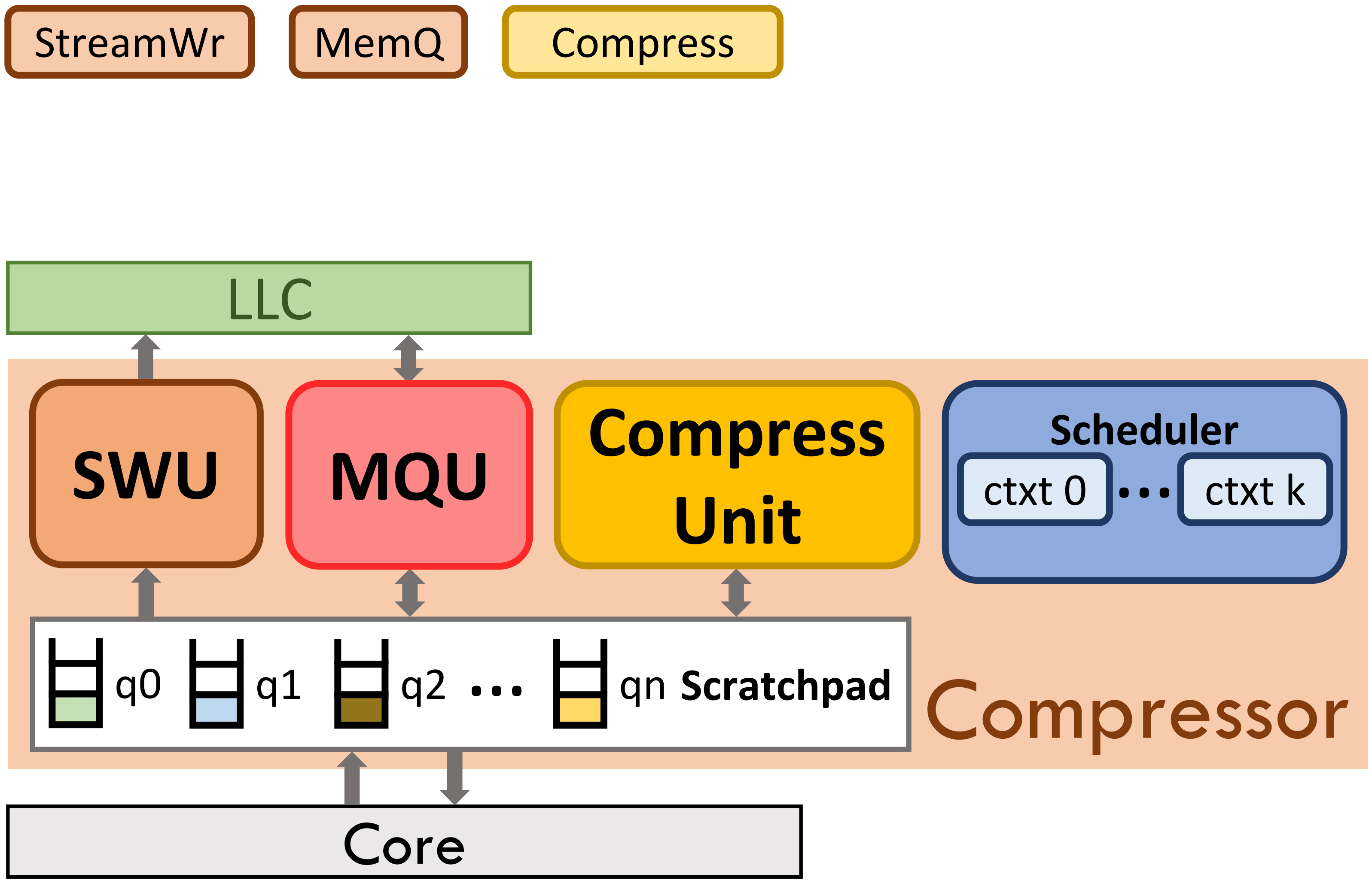
- Scheduler holds operator contexts and chooses which operator to fire each cycle





SpZip compressor overview

- Compression uses a different set of DCL operators
- Similar decoupled and programmable design as the fetcher
- See paper for more details



Agenda

- Motivation
- SpZip Dataflow Configuration Language (DCL)
- SpZip Design
- Evaluation

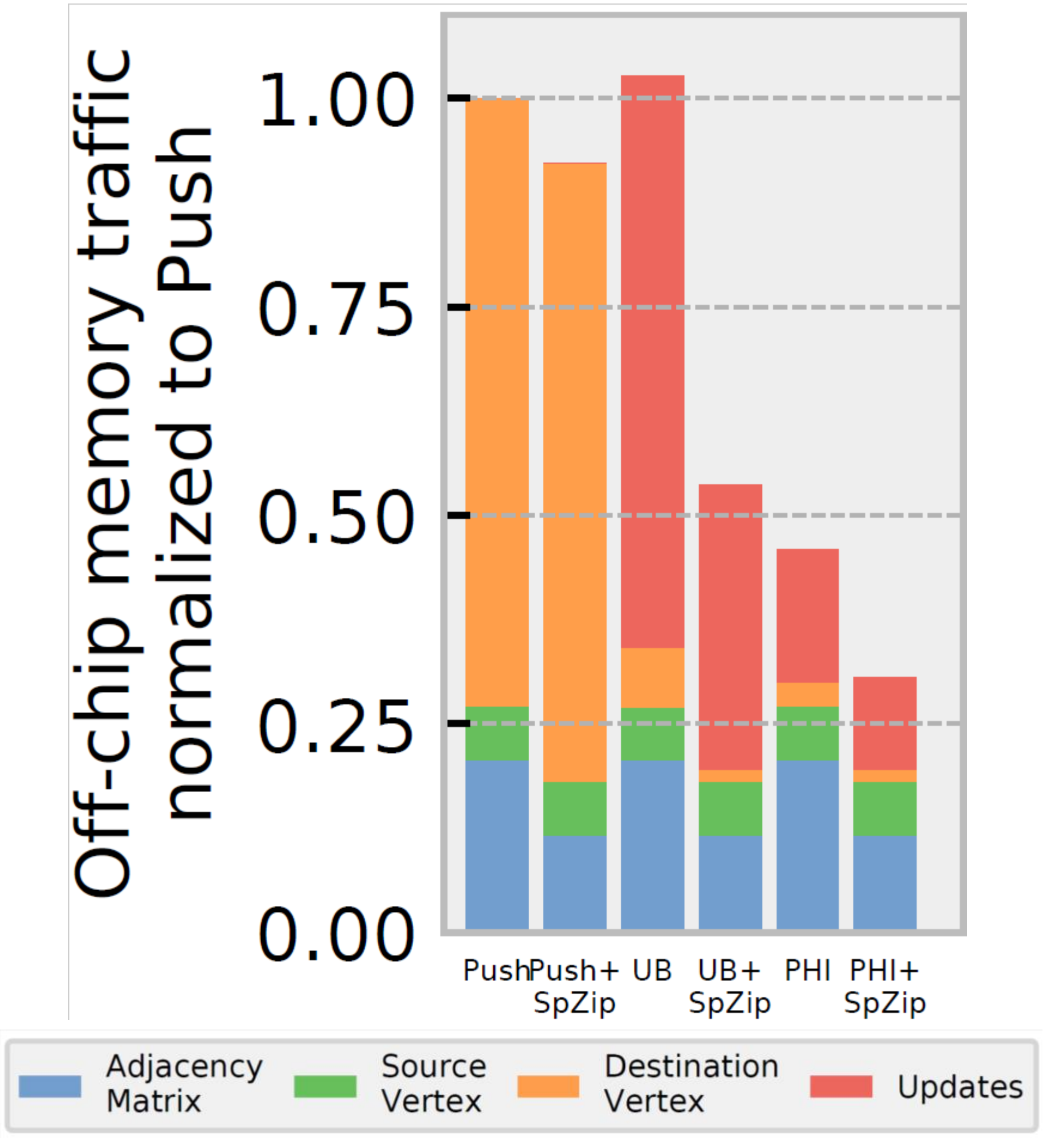
Evaluation Methodology

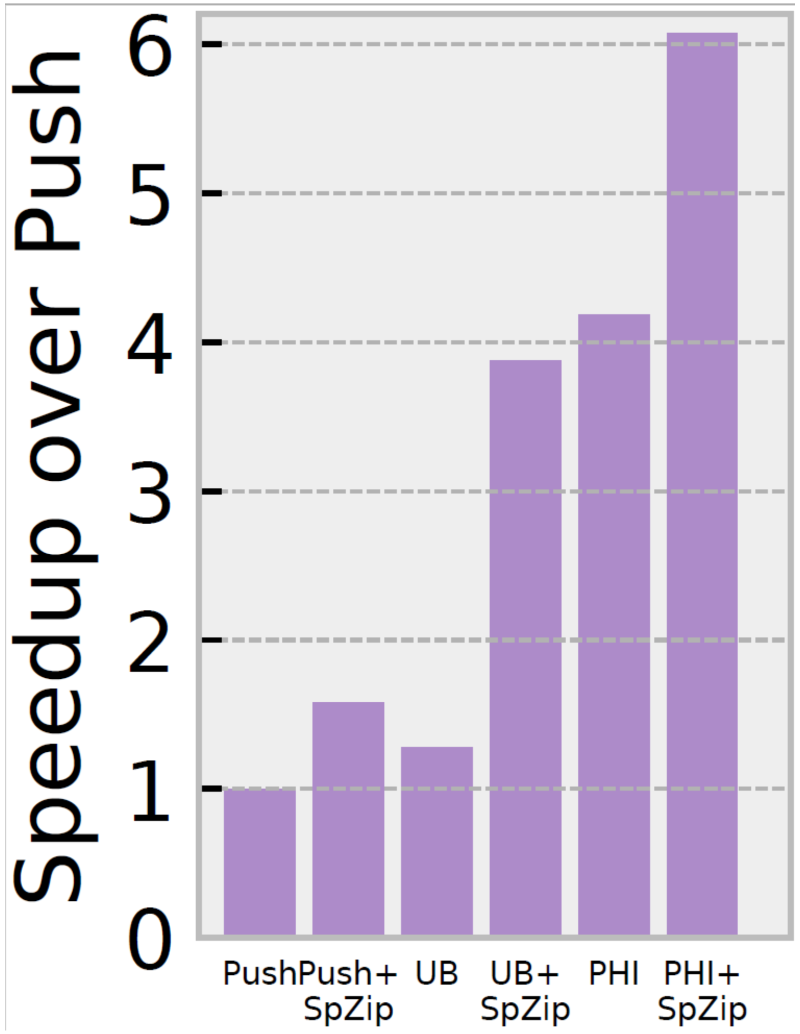
- Event-driven simulation using ZSim
- SpZip system
  - 16 Haswell-like OOO cores
  - 32 MB L3 cache
  - 4 memory controllers (51.2GB/s)
  - SpZip adds 0.2% area overhead of per-core fetcher and compressor

- Irregular applications
  - PageRank, PageRank Delta, Connected Components, Radii Estimation, BFS, Degree Counting, SPMV

- Large real world inputs
  - Up to 100 million vertices
  - Up to 1 billion edges

SpZip improves performance and reduces traffic





See paper for

- DCL support for compressing data structures
- Programmable compressor design
- Additional evaluation results
  - Impact of preprocessing
  - Benefits over compressed memory hierarchies
  - Impact of decoupled fetching vs data compression

Conclusions

- Irregular applications have indirect, data-dependent memory access patterns that make compression challenging

- SpZip makes data compression practical for irregular applications
  - Decoupled execution hides memory access and decompression latencies
  - DCL and programmable design support wide range of data structures and compression formats

- SpZip achieves significant speedups and memory traffic reductions on irregular applications

Thanks For Your Attention!

ISCA 2021

Session 12B (June 16, 2021 at 8 PM EDT)

CSAIL