# Smaller and Faster: Parallel Processing of Compressed Graphs with Ligra+

Julian Shun

Based on joint work with Guy Blelloch and Laxman Dhulipala

# Ligra Graph Processing Framework

EdgeMap

VertexMap

Breadth-first search
Betweenness centrality
Connected components
Triangle counting
K-core decomposition
Maximal independent set
Set cover

Single-source shortest paths
Eccentricity estimation
(Personalized) PageRank
Local graph clustering
Biconnected components
Collaborative filtering
…

*Simplicity, Performance, Scalability*

# Steps for Graph Traversal

Graph

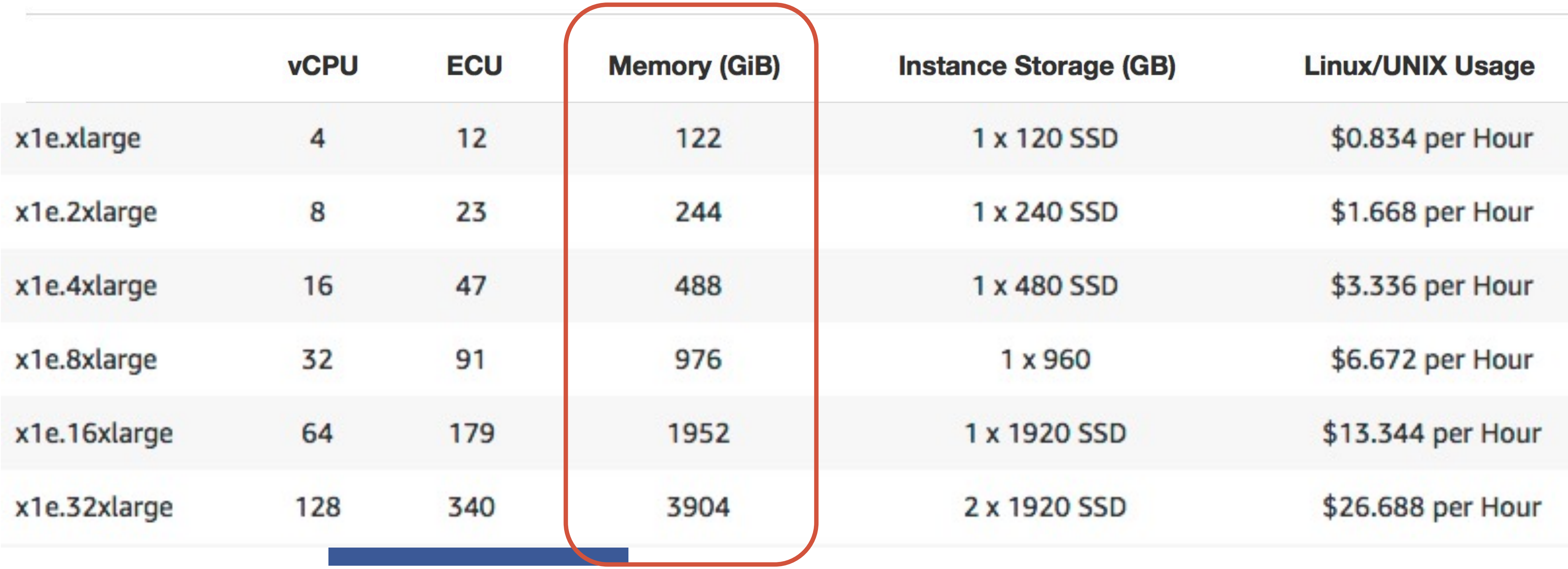
- Operate on a subset of vertices — VertexSubset
- Map computation over subset of edges in parallel
- Return new subset of vertices

} EdgeMap

- Map computation over subset of vertices in parallel — VertexMap

# Large Graphs

## Amazon EC2

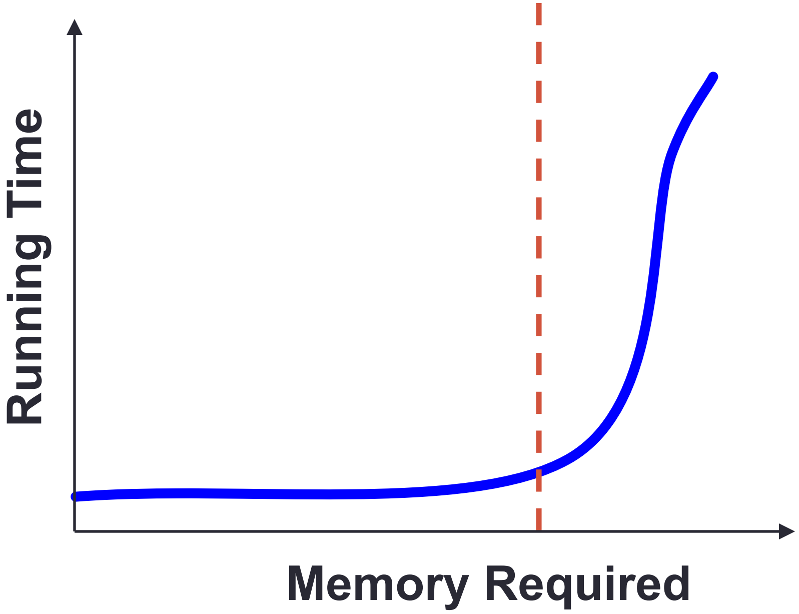


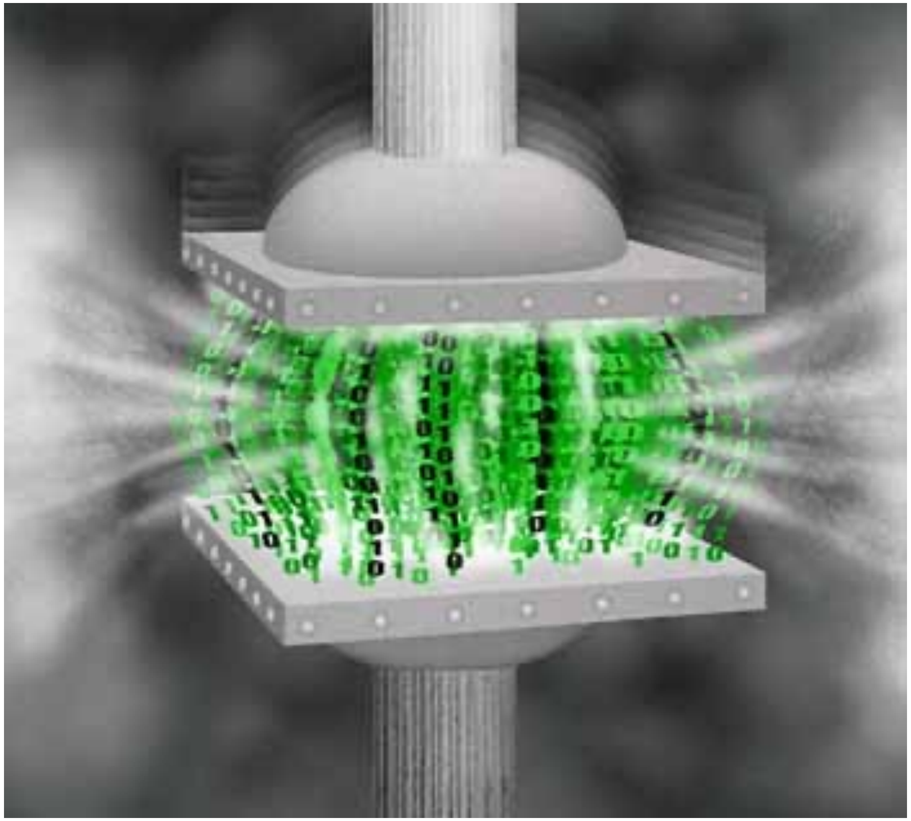
|  | vCPU | ECU | Memory (GiB) | Instance Storage (GB) | Linux/UNIX Usage |
|---|---|---|---|---|---|
| x1e.xlarge | 4 | 12 | 122 | 1 x 120 SSD | $0.834 per Hour |
| x1e.2xlarge | 8 | 23 | 244 | 1 x 240 SSD | $1.668 per Hour |
| x1e.4xlarge | 16 | 47 | 488 | 1 x 480 SSD | $3.336 per Hour |
| x1e.8xlarge | 32 | 91 | 976 | 1 x 960 | $6.672 per Hour |
| x1e.16xlarge | 64 | 179 | 1952 | 1 x 1920 SSD | $13.344 per Hour |
| x1e.32xlarge | 128 | 340 | 3904 | 2 x 1920 SSD | $26.688 per Hour |

- Most can fit on commodity shared memory machine

Example
Dell PowerEdge R930:
Up to 96 cores and 6 TB of RAM

# What if you don't have or can't afford that much memory?

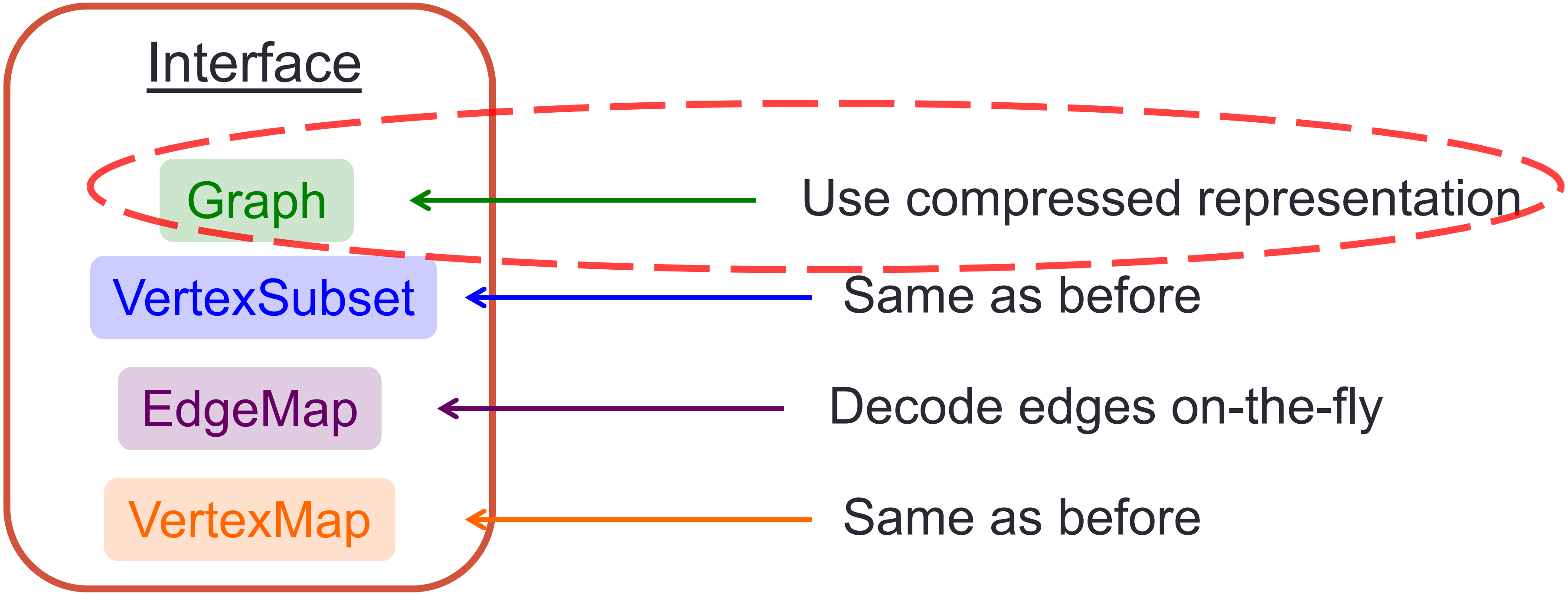




*Graph Compression*

# Ligra+: Adding Graph Compression to Ligra

# Ligra+: Adding Graph Compression to Ligra



Interface

Graph ← Use compressed representation

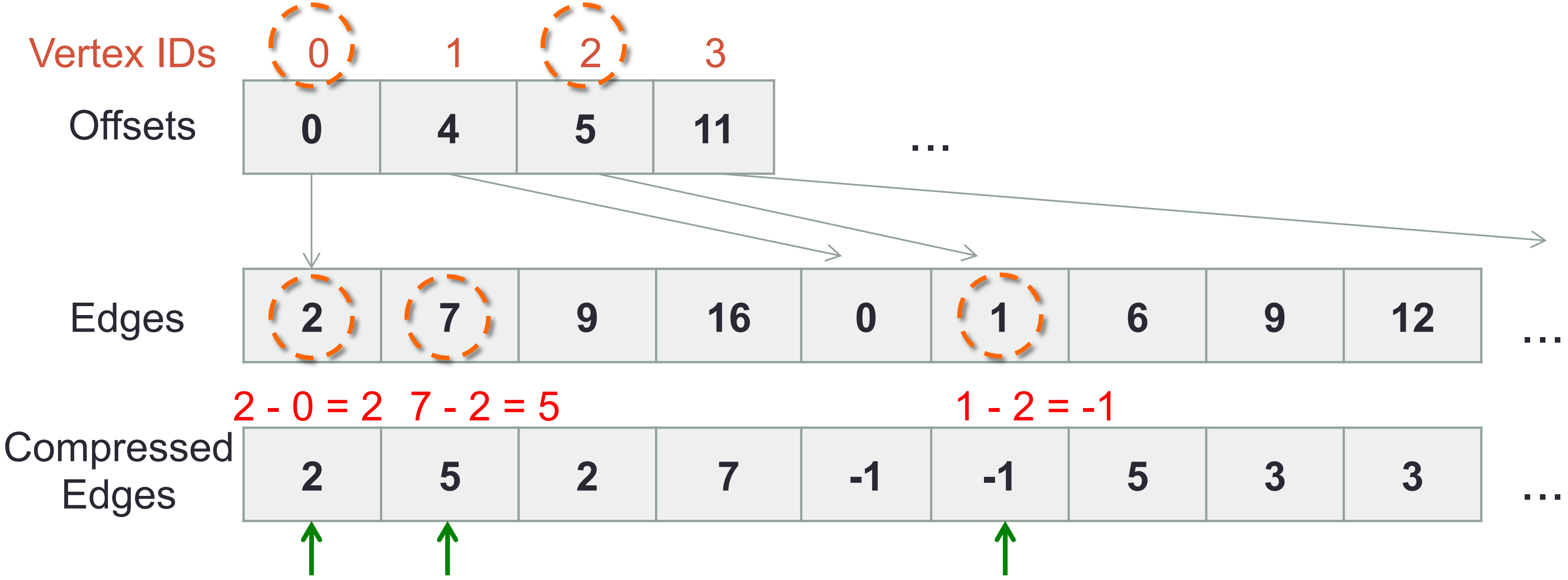
VertexSubset ← Same as before

EdgeMap ← Decode edges on-the-fly

VertexMap ← Same as before

- Same interface as Ligra
- All changes hidden from the user!

# Graph representation



| Vertex IDs | 0 | 1 | 2 | 3 | … |
|---|---|---|---|---|---|
| Offsets | 0 | 4 | 5 | 11 | … |

| Edges | 2 | 7 | 9 | 16 | 0 | 1 | 6 | 9 | 12 | … |
|---|---|---|---|---|---|---|---|---|---|---|
| Compressed Edges | 2 | 5 | 2 | 7 | -1 | -1 | 5 | 3 | 3 | … |

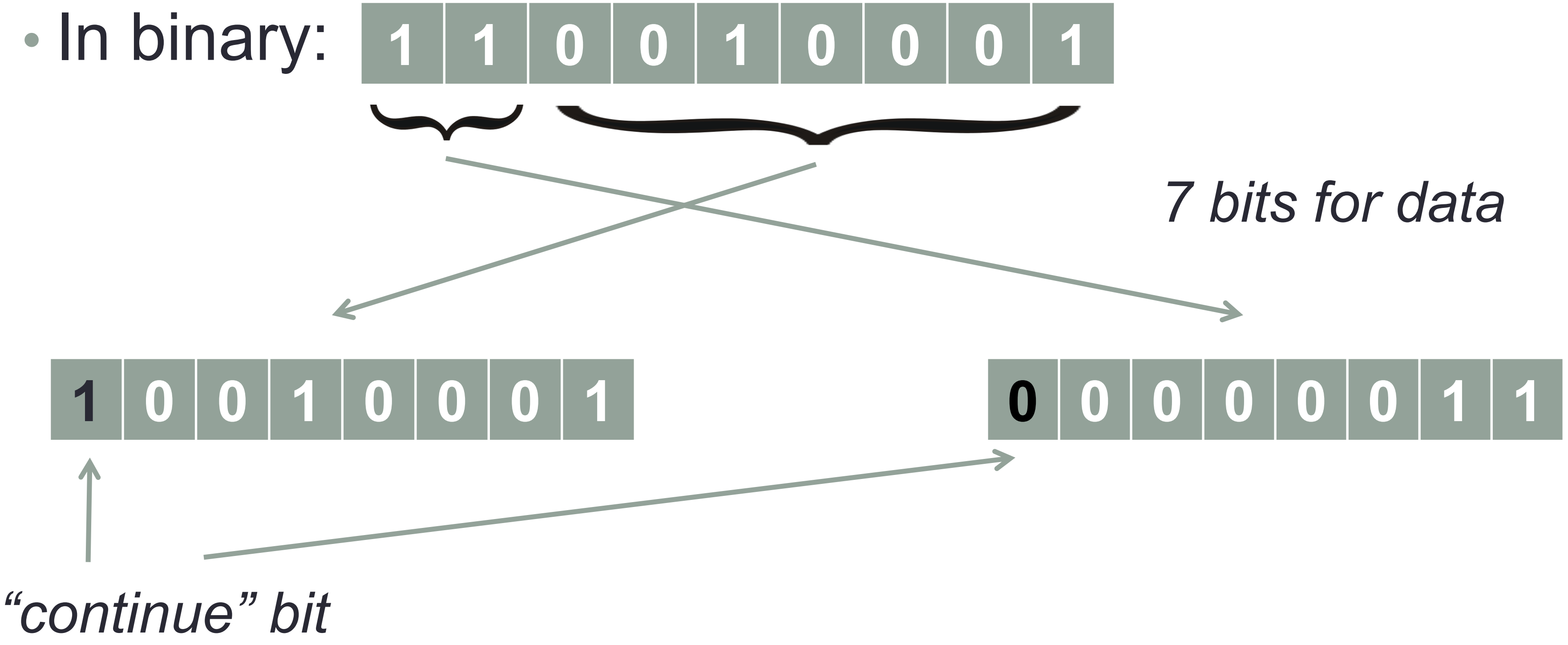
2 - 0 = 2 7 - 2 = 5

1 - 2 = -1

Sort edges and encode differences

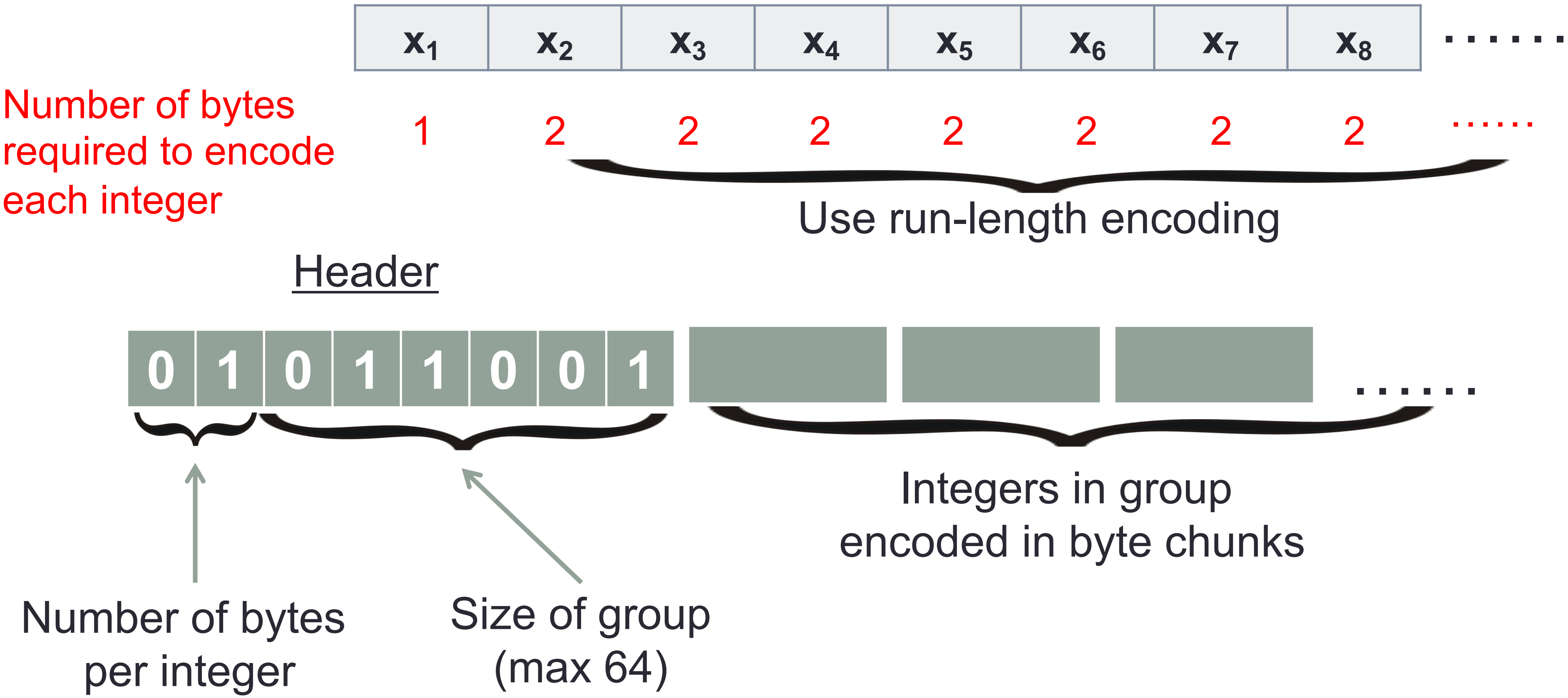
# Variable-length codes

- k-bit codes
  - Encode value in chunks of k bits
  - Use k-1 bits for data, and 1 bit as the “continue” bit
- Example: encode “401” using 8-bit (byte) code
- In binary:



# Encoding optimization

- Another idea: get rid of “continue” bits



$x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$ $x_8$ ……

Number of bytes required to encode each integer

1 2 2 2 2 2 2 2 ……

Use run-length encoding

Header

0 1 0 1 1 0 0 1 ……

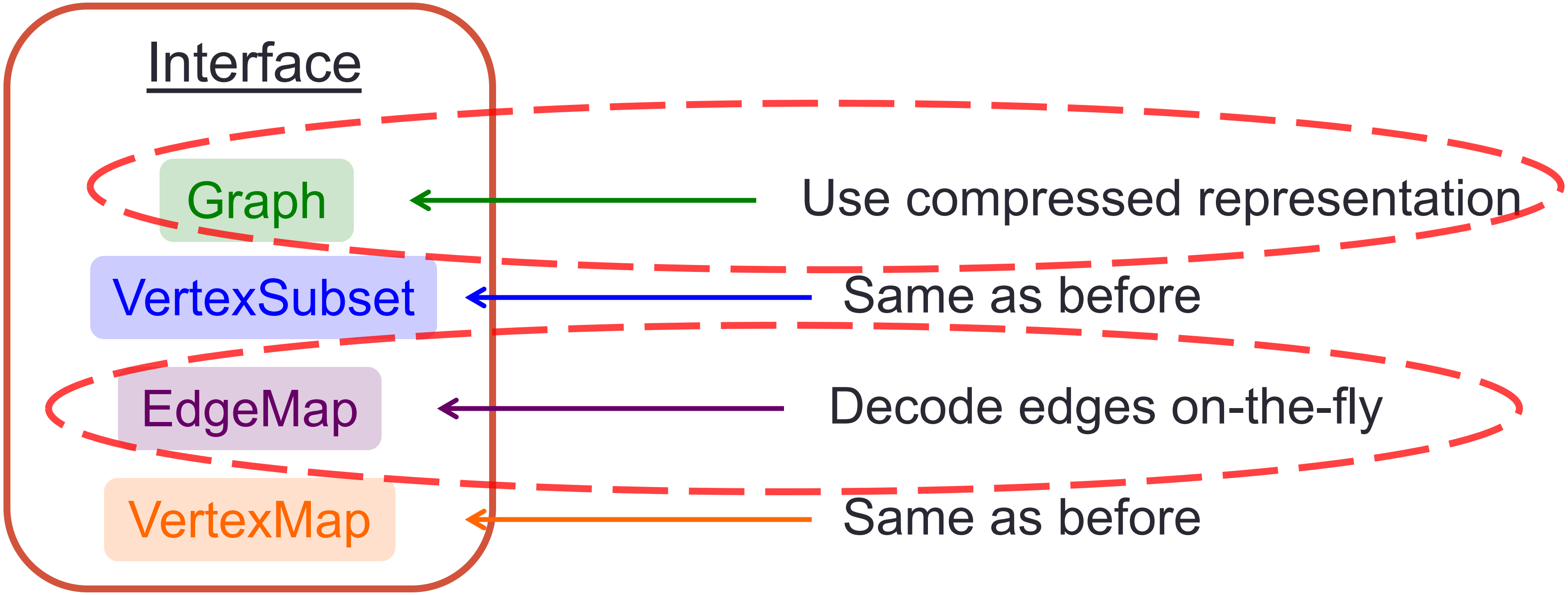
Number of bytes per integer

Size of group (max 64)

Integers in group encoded in byte chunks

- Increases space, but makes decoding cheaper (no branch misprediction from checking “continue” bit)

# Ligra+: Adding Graph Compression to Ligra



Interface

Graph ← Use compressed representation

VertexSubset ← Same as before

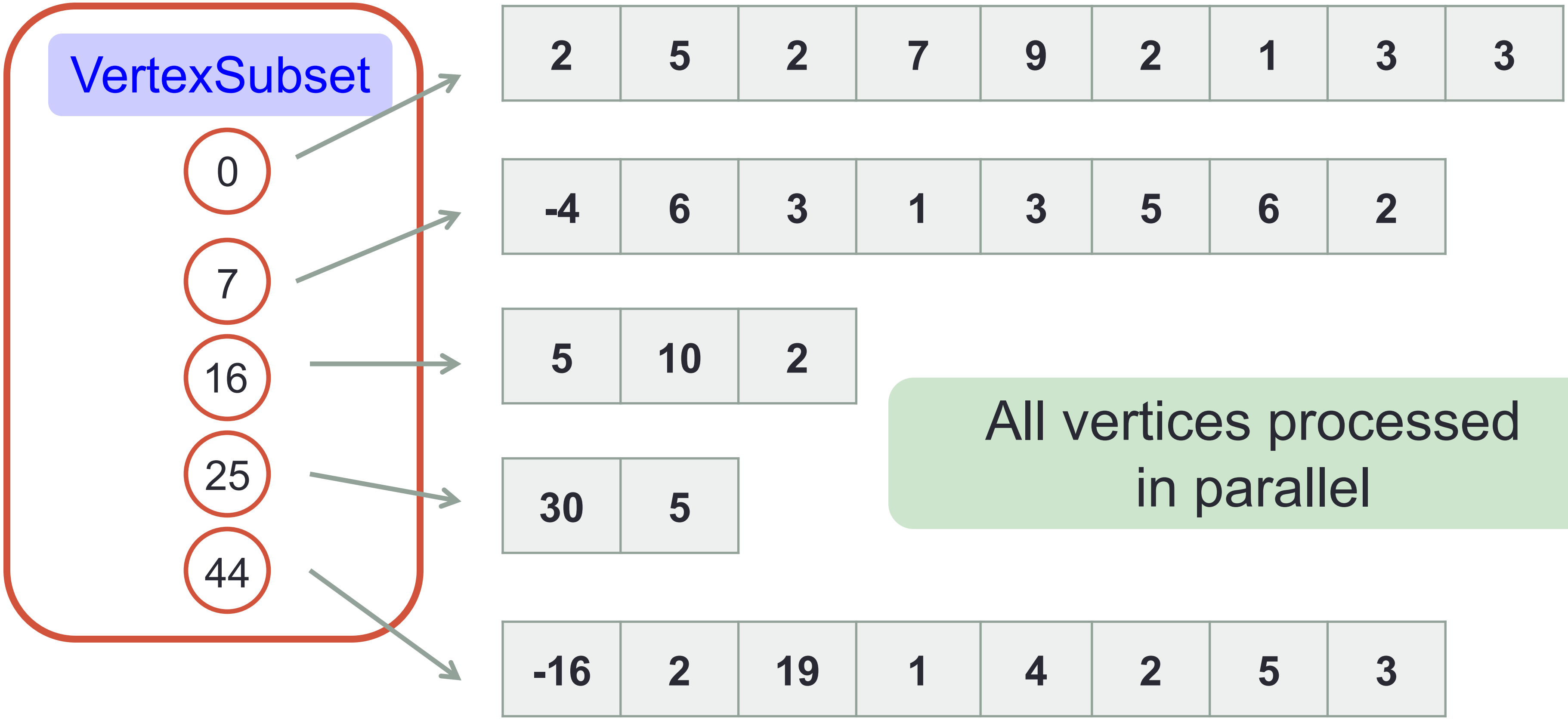
EdgeMap ← Decode edges on-the-fly

VertexMap ← Same as before

- Same interface as Ligra
- All changes hidden from the user!

# Modifying EdgeMap

- Processes outgoing edges of a subset of vertices



VertexSubset

0 7 16 25 44

2 5 2 7 9 2 1 3 3

-4 6 3 1 3 5 6 2

5 10 2

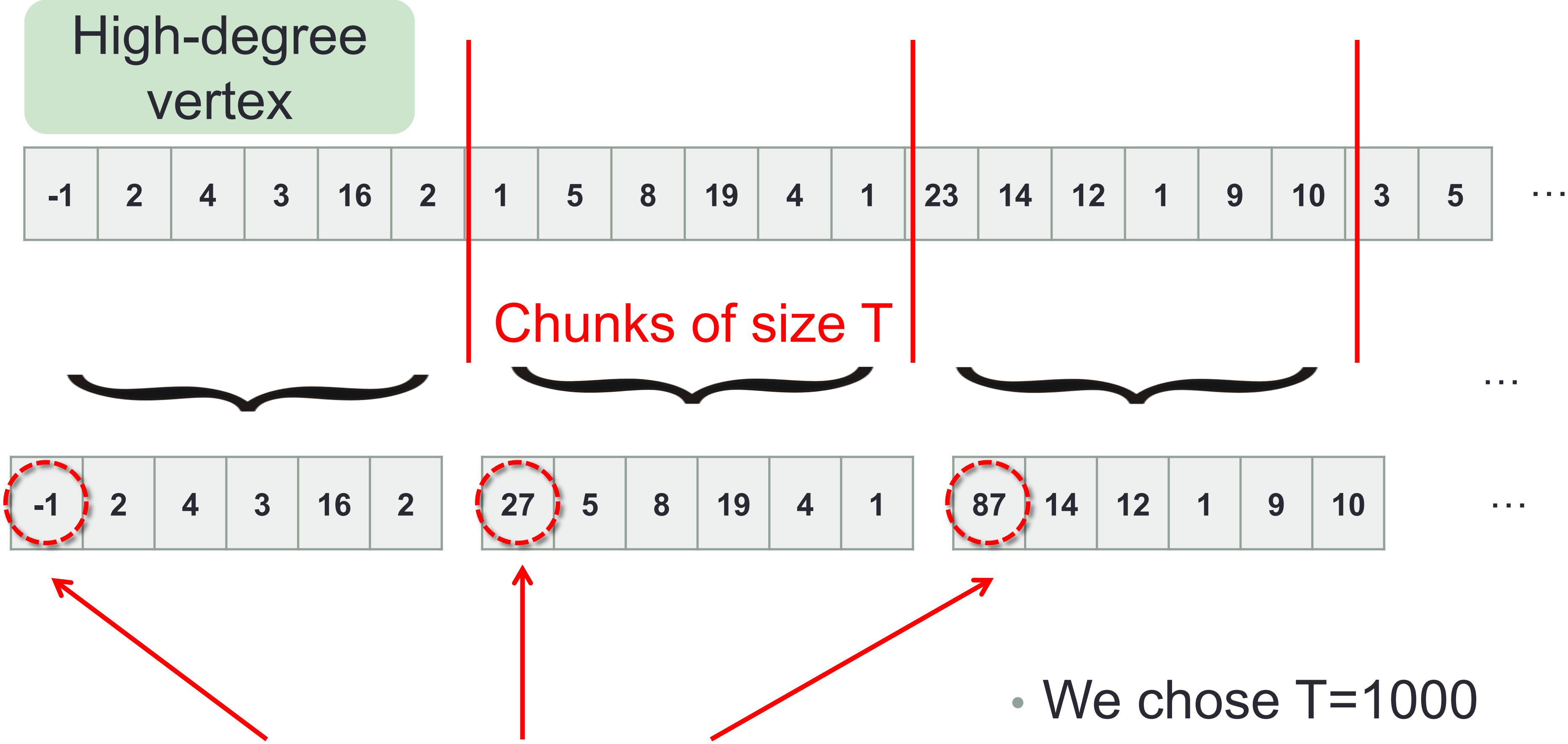
30 5

-16 2 19 1 4 2 5 3

All vertices processed in parallel

*What about high-degree vertices?*

# Handling high-degree vertices



High-degree vertex

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 2 | 4 | 3 | 16 | 2 | 1 | 5 | 8 | 19 | 4 | 1 | 23 | 14 | 12 | 1 | 9 | 10 | 3 | 5 | … |

Chunks of size T

| | | | | | |
|---|---|---|---|---|---|
| -1 | 2 | 4 | 3 | 16 | 2 |

| | | | | | |
|---|---|---|---|---|---|
| 27 | 5 | 8 | 19 | 4 | 1 |

| | | | | | |
|---|---|---|---|---|---|
| 87 | 14 | 12 | 1 | 9 | 10 |

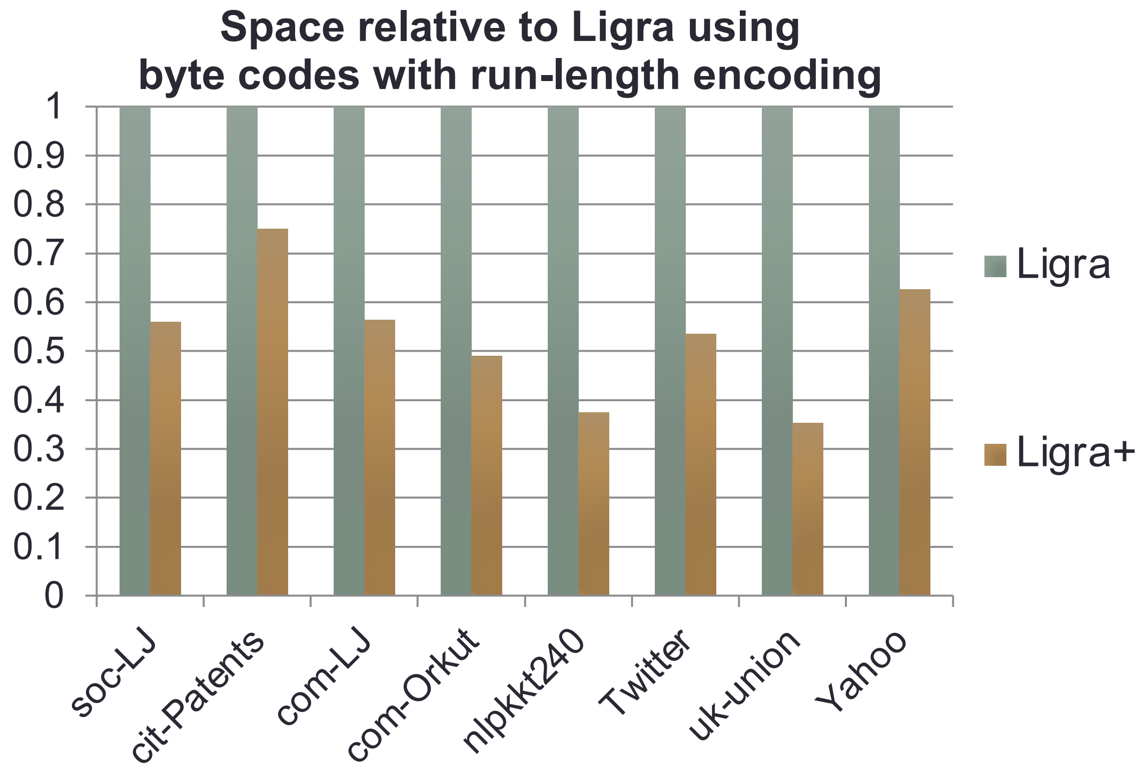
…

Encode first entry relative to source vertex

All chunks can be decoded in parallel!

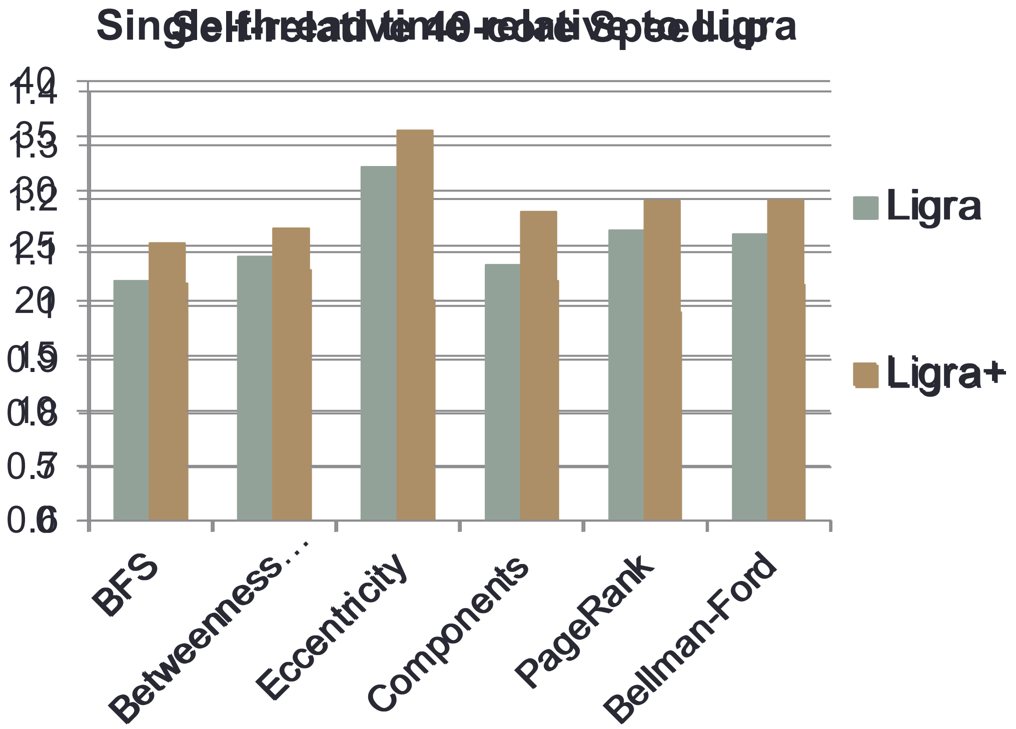
- We chose T=1000
- Similar performance and space usage for a wide range of T

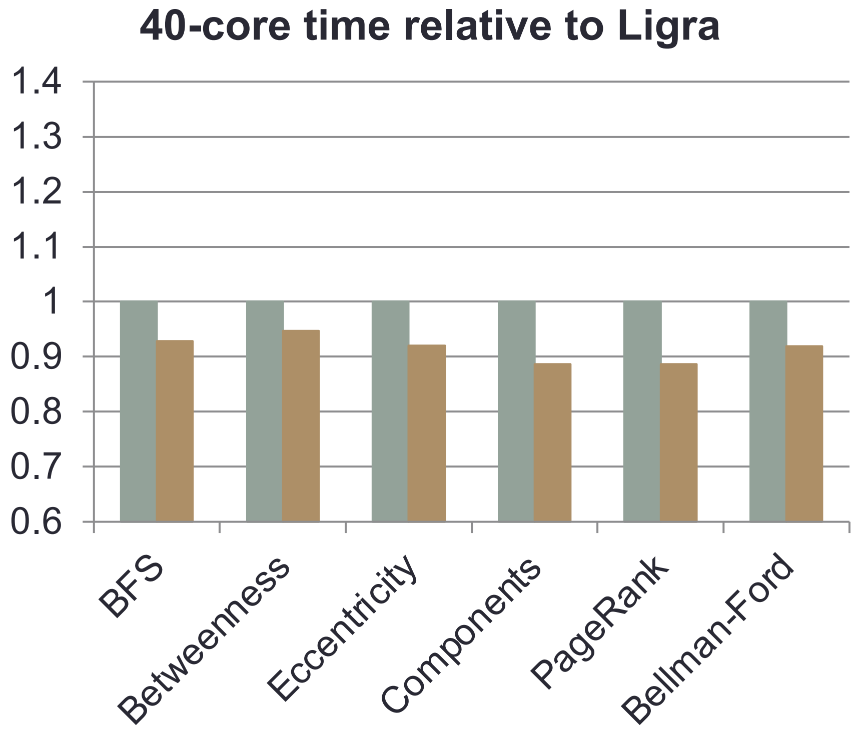
# Ligra+ Space Savings



- Space savings of about 1.3—3x
- Could use more sophisticated schemes to further reduce space, but more expensive to decode
- Cost of decoding on-the-fly?

# Ligra+ Performance





- Cost of decoding on-the-fly?
- Memory subsystem is a scalability bottleneck in parallel as these graph algorithms are memory-bound
- Ligra+ decoding gets better parallel speed up

# Ligra Summary

VertexSubset

VertexMap

EdgeMap

*Optimizations: Hybrid traversal and graph compression*

Breadth-first search
Betweenness centrality
Connected components
Triangle counting
K-core decomposition
Maximal independent set
…

Single-source shortest paths
Eccentricity estimation
(Personalized) PageRank
Local graph clustering
Biconnected components
Collaborative filtering
…

*Simplicity, Performance, Scalability*

# Thank you!

J. Shun and G. E. Blelloch. Ligra: *A Lightweight Graph Processing Framework for Shared Memory*, Principles and Practice of Parallel Programming, 2013.

J. Shun, L. Dhulipala and G. E. Blelloch. Smaller and Faster: *Parallel Processing of Compressed Graphs with Ligra+*, Data Compression Conference, 2015.

Code: https://github.com/jshun/ligra/