Efficient Processing of Large Graphs via Input Reduction

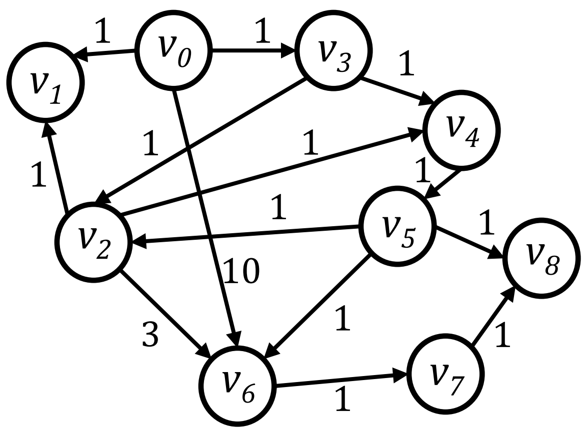
Amlan Kusum, **Keval Vora**,
Rajiv Gupta, Iulian Neamtiu

HPDC'16 – Kyoto, Japan
04 June, 2016

UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution

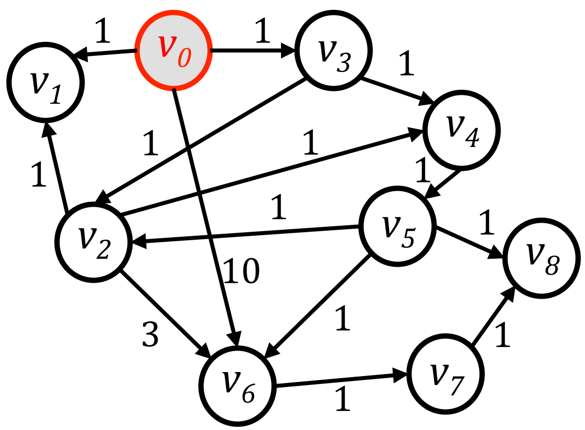


t	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
t_0	0	∞	∞	∞	∞	∞	∞	∞	∞
t_1	0	1	∞	1	∞	∞	10	∞	∞
t_2	0	1	2	1	2	∞	10	11	∞
t_3	0	1	2	1	2	3	5	11	12
t_4	0	1	2	1	2	3	4	6	4
t_5	0	1	2	1	2	3	4	5	4
t_6	0	1	2	1	2	3	4	5	4

UNIVERSITY OF CALIFORNIA, RIVERSIDE

Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution

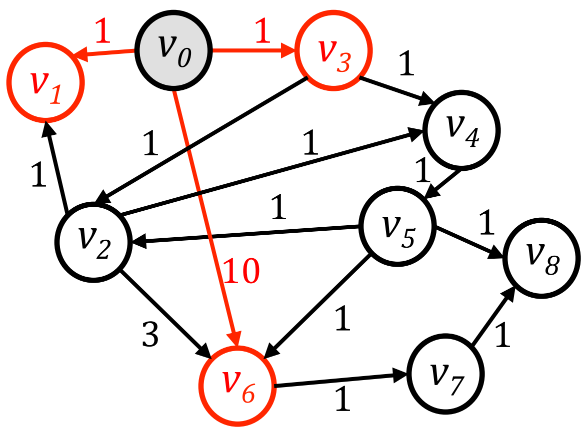


t	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
t_0	**0**	∞	∞	∞	∞	∞	∞	∞	∞
t_1	0	1	∞	1	∞	∞	10	∞	∞
t_2	0	1	2	1	2	∞	10	11	∞
t_3	0	1	2	1	2	3	5	11	12
t_4	0	1	2	1	2	3	4	6	4
t_5	0	1	2	1	2	3	4	5	4
t_6	0	1	2	1	2	3	4	5	4

UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution

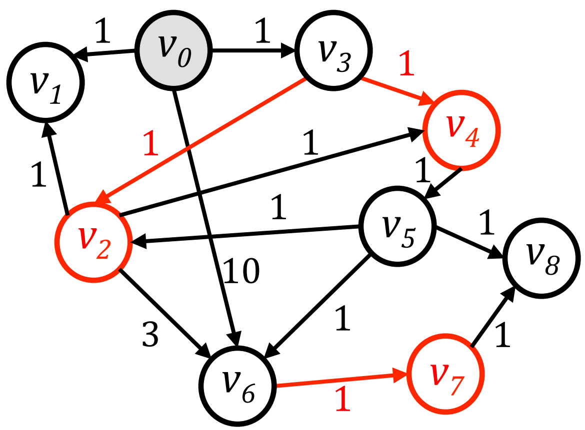


t	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
t_0	0	∞	∞	∞	∞	∞	∞	∞	∞
t_1	0	**1**	∞	**1**	∞	∞	**10**	∞	∞
t_2	0	1	2	1	2	∞	10	11	∞
t_3	0	1	2	1	2	3	5	11	12
t_4	0	1	2	1	2	3	4	6	4
t_5	0	1	2	1	2	3	4	5	4
t_6	0	1	2	1	2	3	4	5	4

UNIVERSITY OF CALIFORNIA, RIVERSIDE

Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution

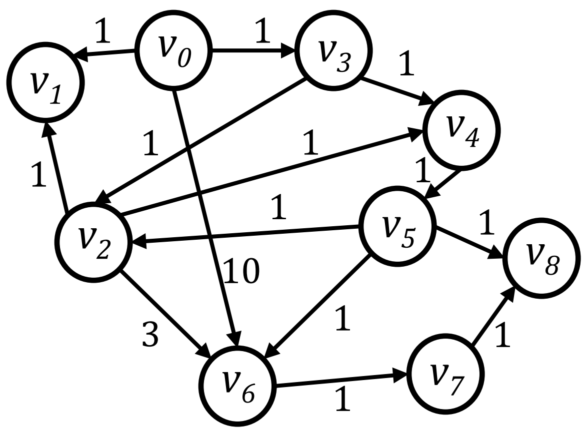


t	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
t_0	0	∞	∞	∞	∞	∞	∞	∞	∞
t_1	0	1	∞	1	∞	∞	10	∞	∞
t_2	0	1	**2**	1	**2**	∞	10	**11**	∞
t_3	0	1	2	1	2	3	5	11	12
t_4	0	1	2	1	2	3	4	6	4
t_5	0	1	2	1	2	3	4	5	4
t_6	0	1	2	1	2	3	4	5	4

UNIVERSITY OF CALIFORNIA, RIVERSIDE

Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution
- Challenging due to ever-growing graph sizes

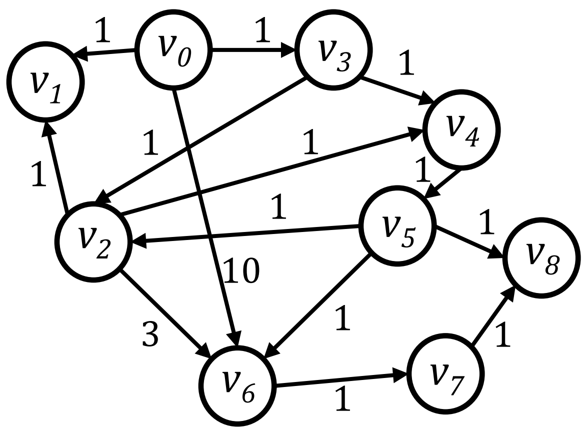


t	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
t_0	0	∞	∞	∞	∞	∞	∞	∞	∞
t_1	0	1	∞	1	∞	∞	10	∞	∞
t_2	0	1	2	1	2	∞	10	11	∞
t_3	0	1	2	1	2	3	5	11	12
t_4	0	1	2	1	2	3	4	6	4
t_5	0	1	2	1	2	3	4	5	4
t_6	0	1	2	1	2	3	4	5	4

UNIVERSITY OF CALIFORNIA, RIVERSIDE

Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution
- Challenging due to ever-growing graph sizes
- Convergence speed is dependent on initializations



t	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
t_0	0	∞	∞	∞	∞	**3**	∞	∞	∞
t_1	0	1	∞	1	∞	3	4	∞	4
t_2	0	1	2	1	2	3	4	6	4
t_3	0	1	2	1	2	3	4	6	4

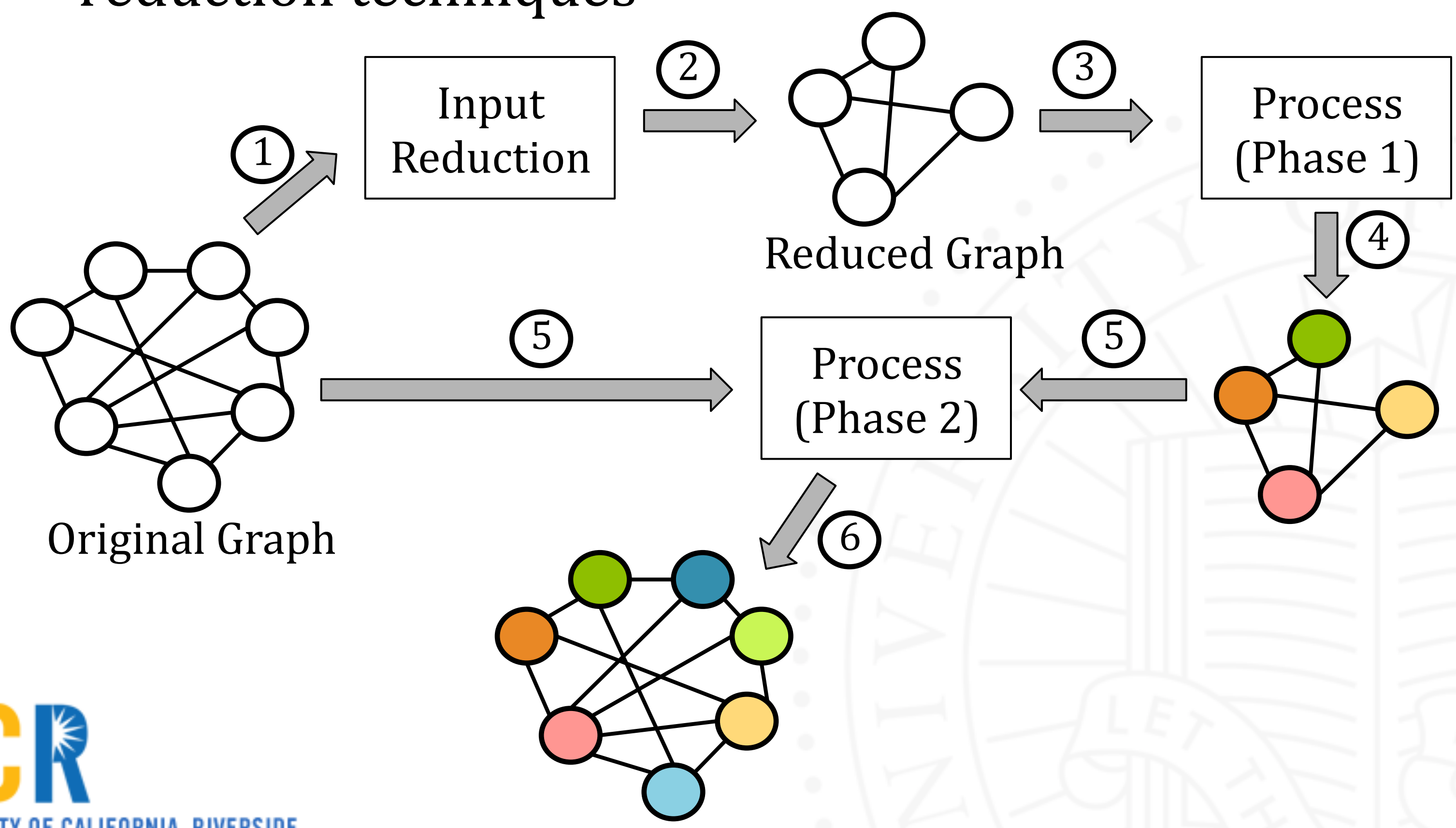
How to find better initializations?

Key Idea

- Compute initial values using a smaller signature of the original graph
  - Generate smaller graph using light-weight input reduction techniques

Key Idea

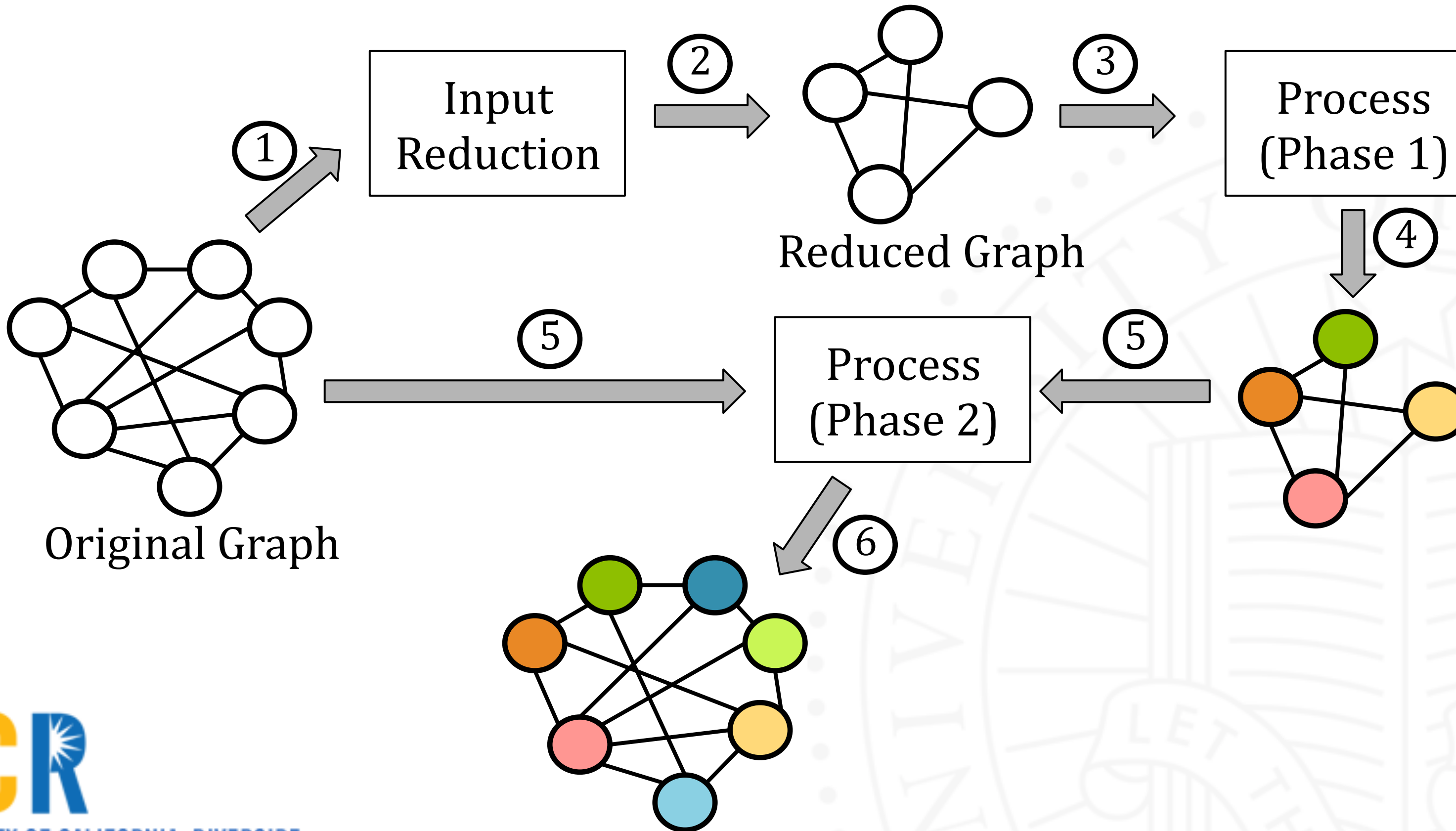
- Compute initial values using a smaller signature of the original graph
  - Generate smaller graph using light-weight input reduction techniques



UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Key Idea

time(Input Reduction)
+ time(Phase 1) **<** **time(Original)**
+ time(Phase 2)



UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Outline

- Input Reduction
- Vertex Transformations
- Correctness of Results
- Evaluation
- Conclusion

UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

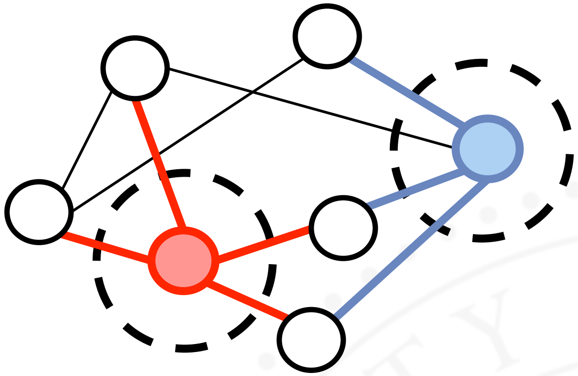
Input Reduction

- Must be light-weight & general
  - Multilevel graph partitioning [SC’95, SC’01]
    - Matching based contraction [ICPP’95, JPDC’98]
    - Pruning based on edge costs affecting paths [ICDM’10]
  - Gate graph for shortest paths problem [ICDM’11]
- Develop vertex level transformations
  - Easily parallelizable using the vertex centric graph processing systems

UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

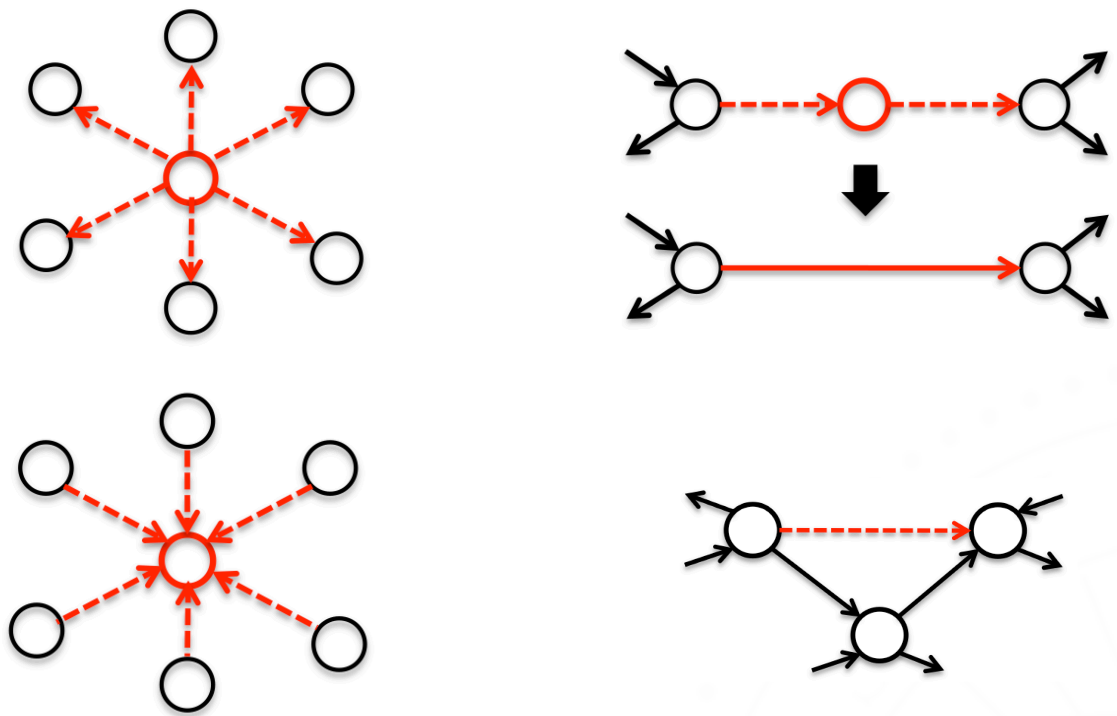
Vertex Transformations

- Maintain structural integrity of the graph
  - Preserve the overall connectivity
- Light-weight
  - Local
  - Non-interfering

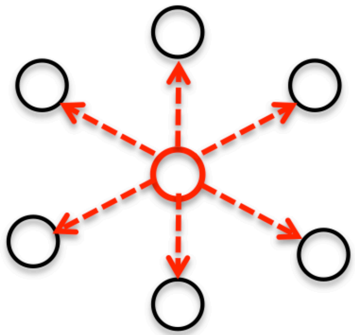


UNIVERSITY OF CALIFORNIA, RIVERSIDE

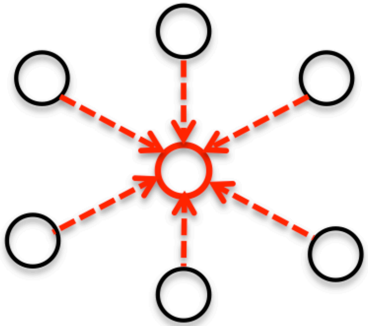
Vertex Transformations



Vertex Transformations

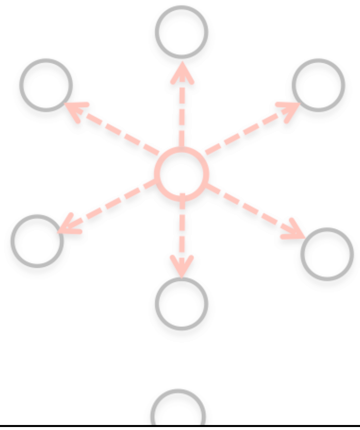


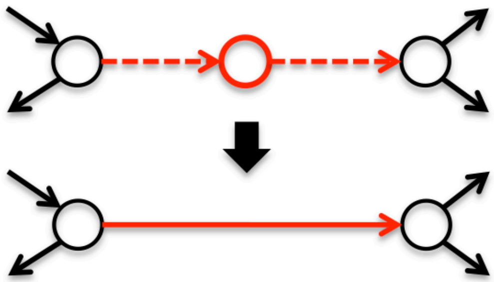
if (inDegree(v) $=0$) **then**
$\triangleright$ apply $\mathcal{T}_1$: drop $v \to *$
$\mathcal{E}' \leftarrow \mathcal{E}' \setminus$ outEdges(v)



if (outDegree(v) $=0$) **then**
$\triangleright$ apply $\mathcal{T}_2$: drop $* \to v$
$\mathcal{E}' \leftarrow \mathcal{E}' \setminus$ inEdges(v)

Vertex Transformations



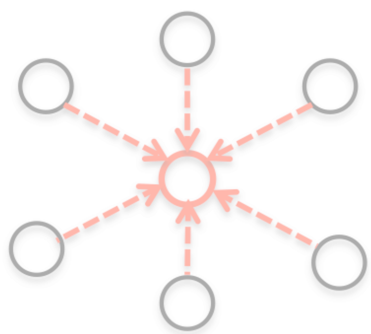


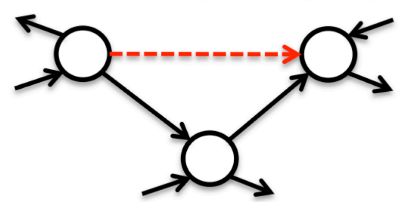
if (inDegree(v) = outDegree(v) = 1) **then**
$\triangleright$ apply $\mathcal{T}_3$: bypass v
$\mathcal{E}' \leftarrow (\mathcal{E}' \setminus \{u \rightarrow v, v \rightarrow w\}) \cup \{u \rightarrow w\}$
where $\{u \rightarrow v, v \rightarrow w\} \subseteq \mathcal{E}'$

UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

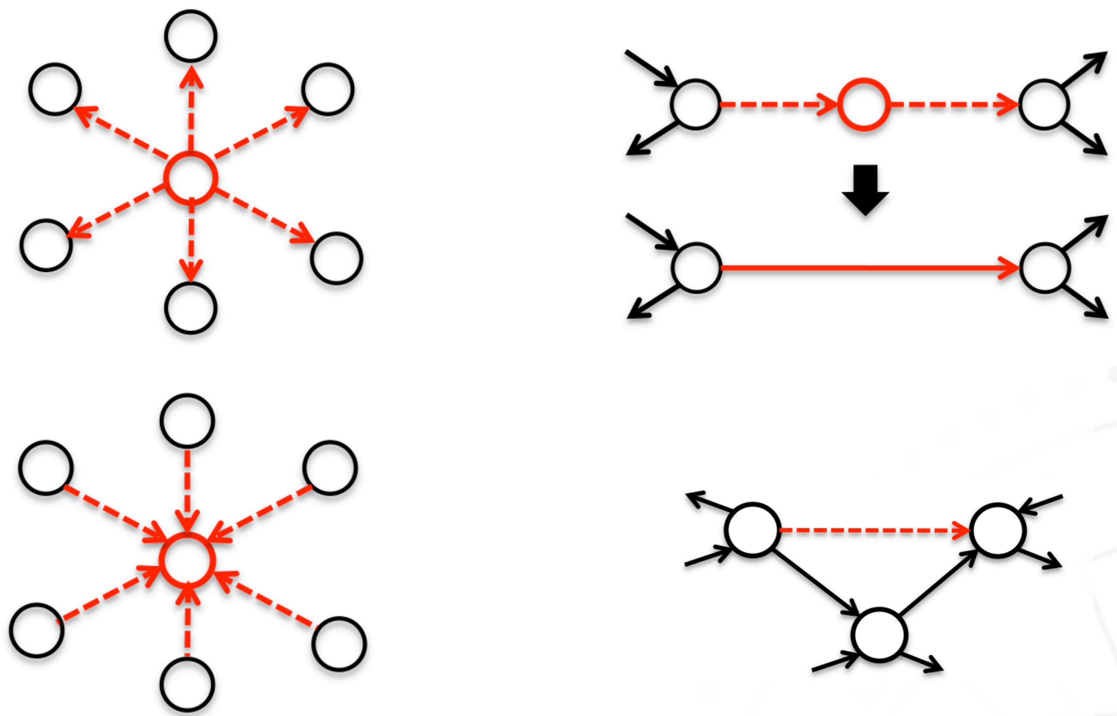
Vertex Transformations

if ($w \in \text{outNeighbors}(v)$ s.t. w is unchanged **and**
outNeighbors(v) $\cap$ inNeighbors(w) $\neq \phi$) **then**
$\triangleright$ apply $\mathcal{T}_5$: drop $v \to w$
$$\mathcal{E}' \leftarrow \mathcal{E}' \setminus \{(v \to w)\}$$





Vertex Transformations

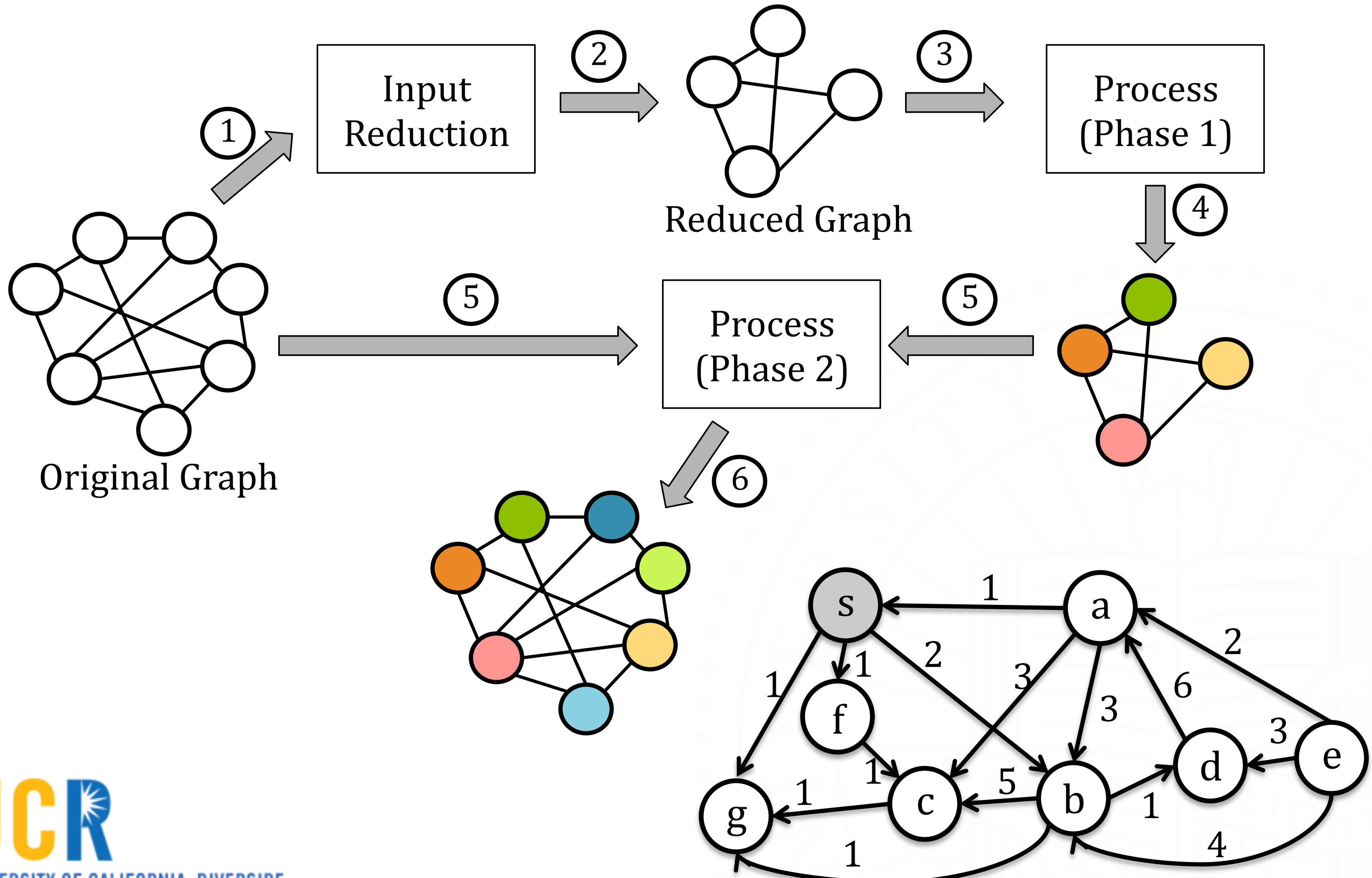


UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Other Details

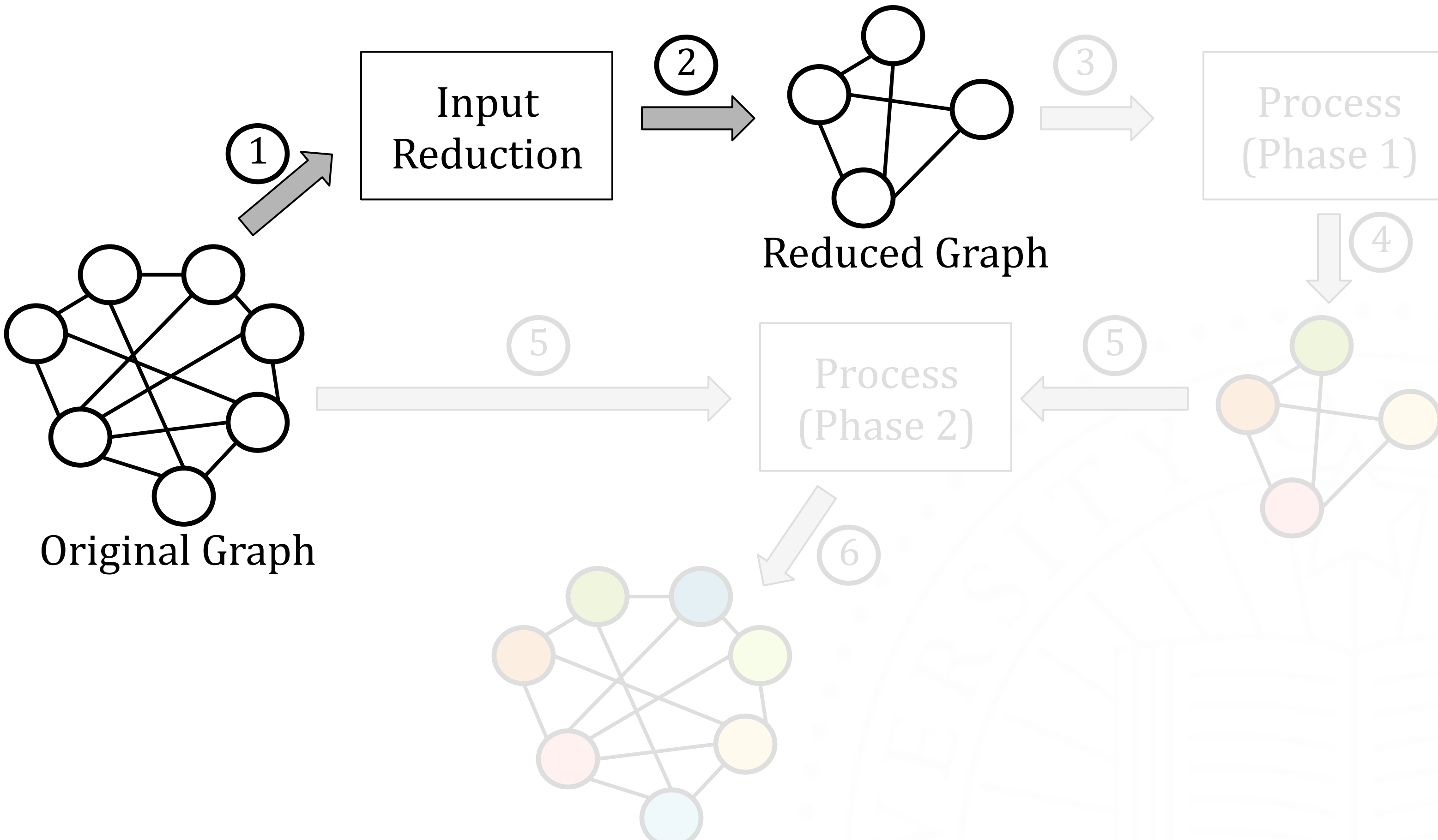
- More vertex transformations
  - Some relax structural integrity
- Order of transformations
  - Unified graph reduction algorithm

Processing workflow



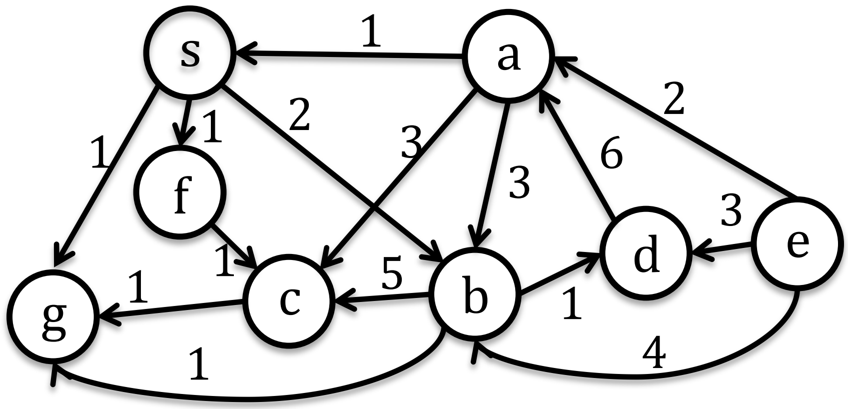
UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

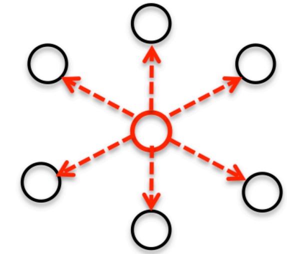
Processing workflow

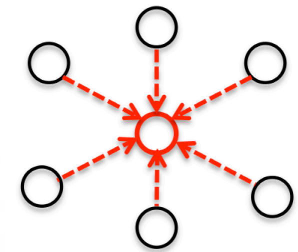


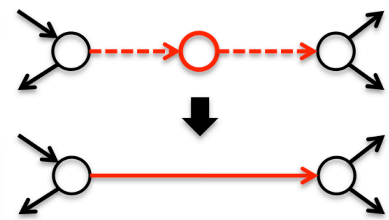
UNIVERSITY OF CALIFORNIA, RIVERSIDE

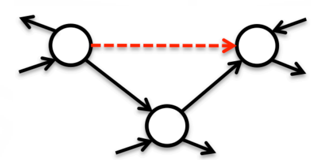
Input Reduction



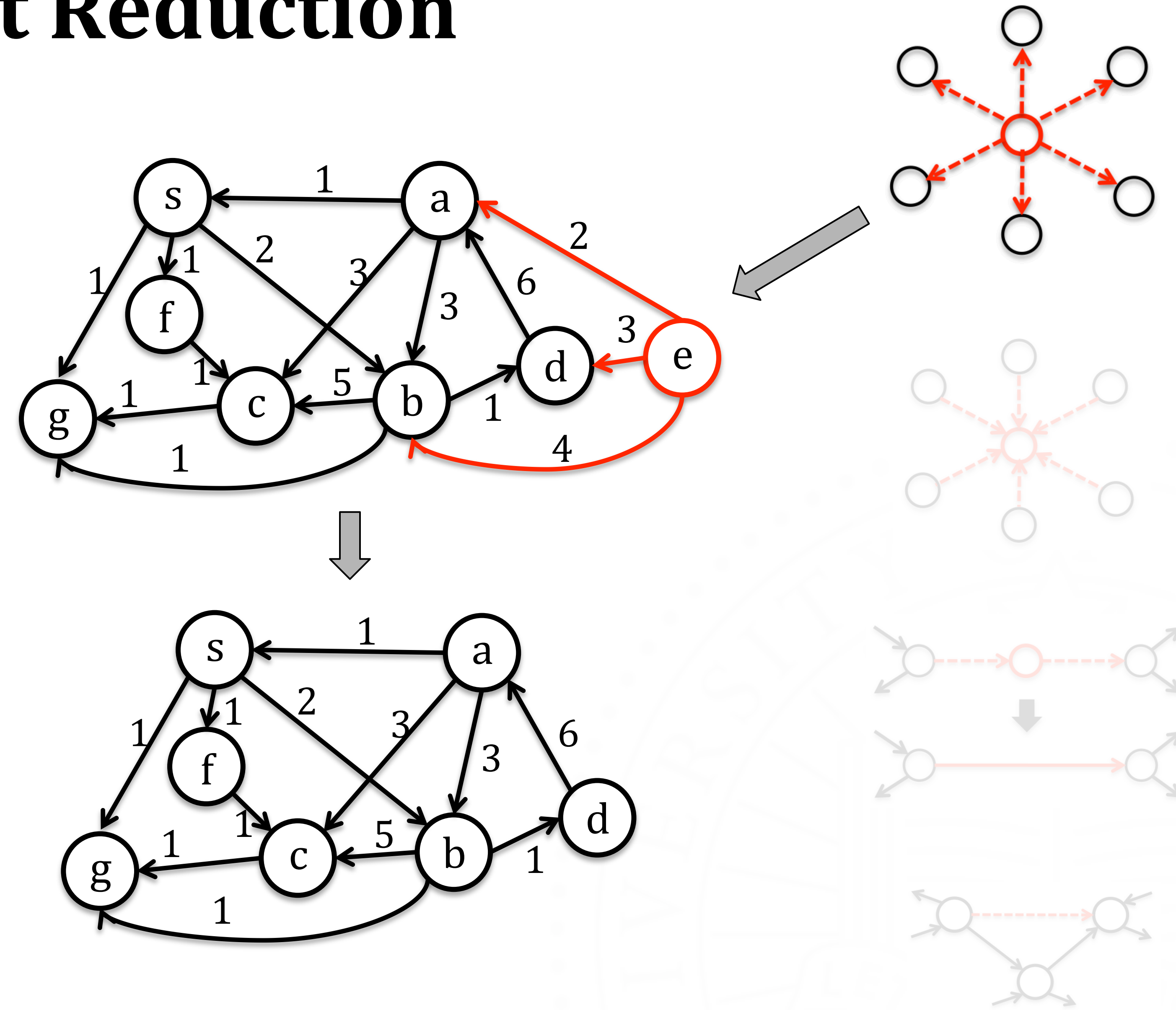




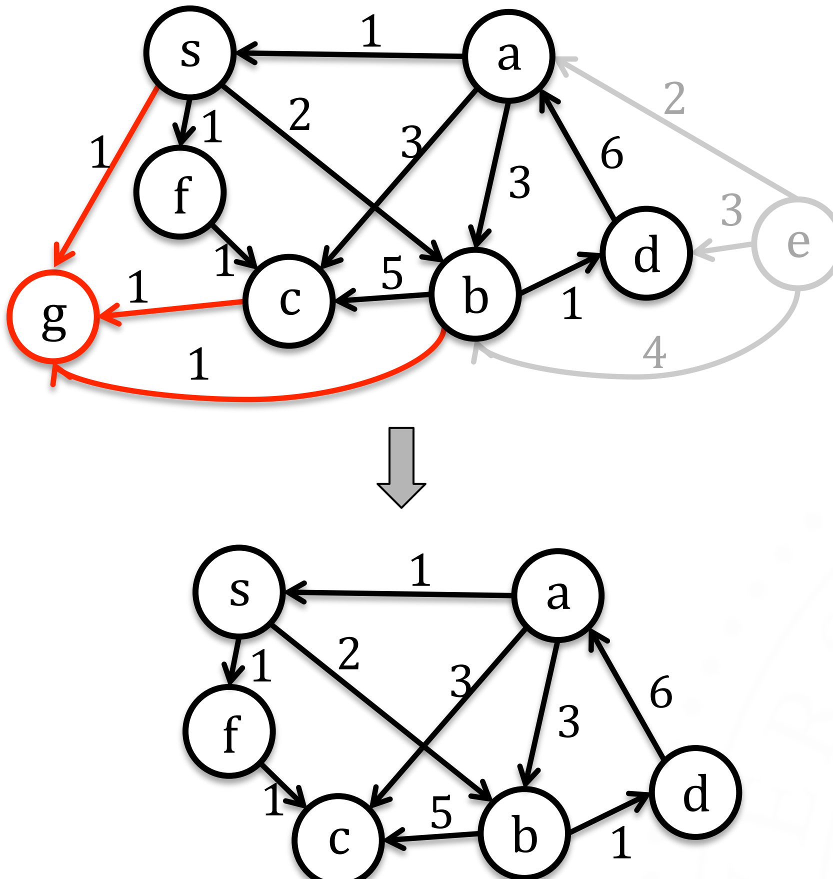


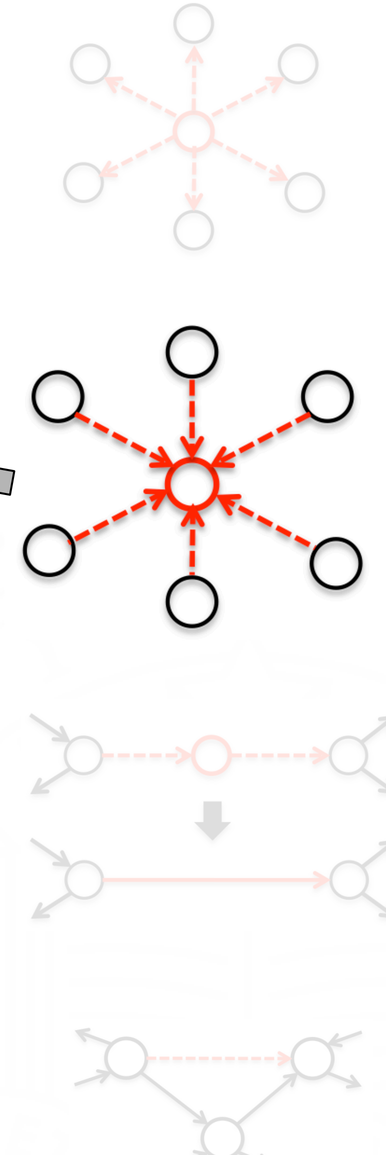


Input Reduction



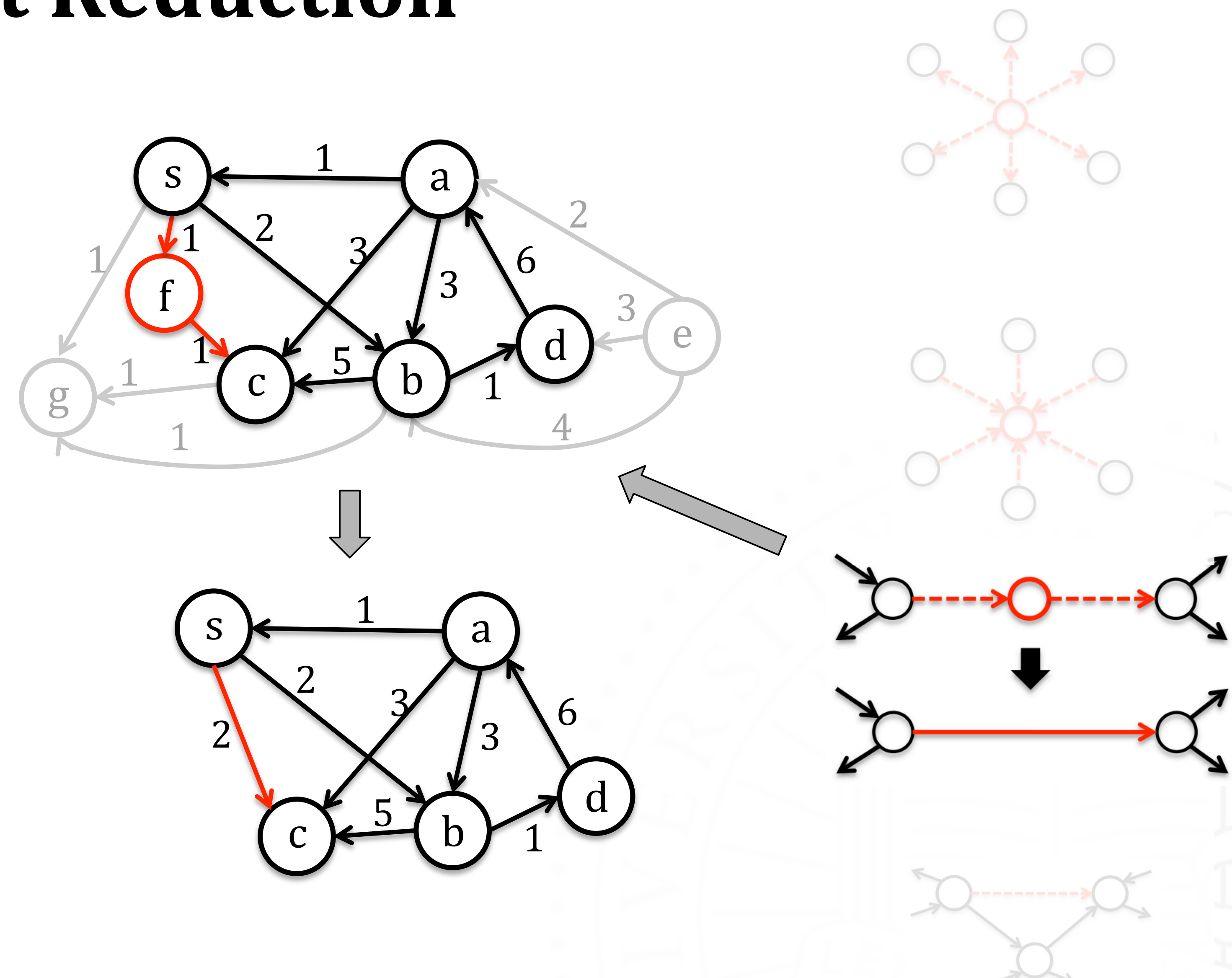
Input Reduction



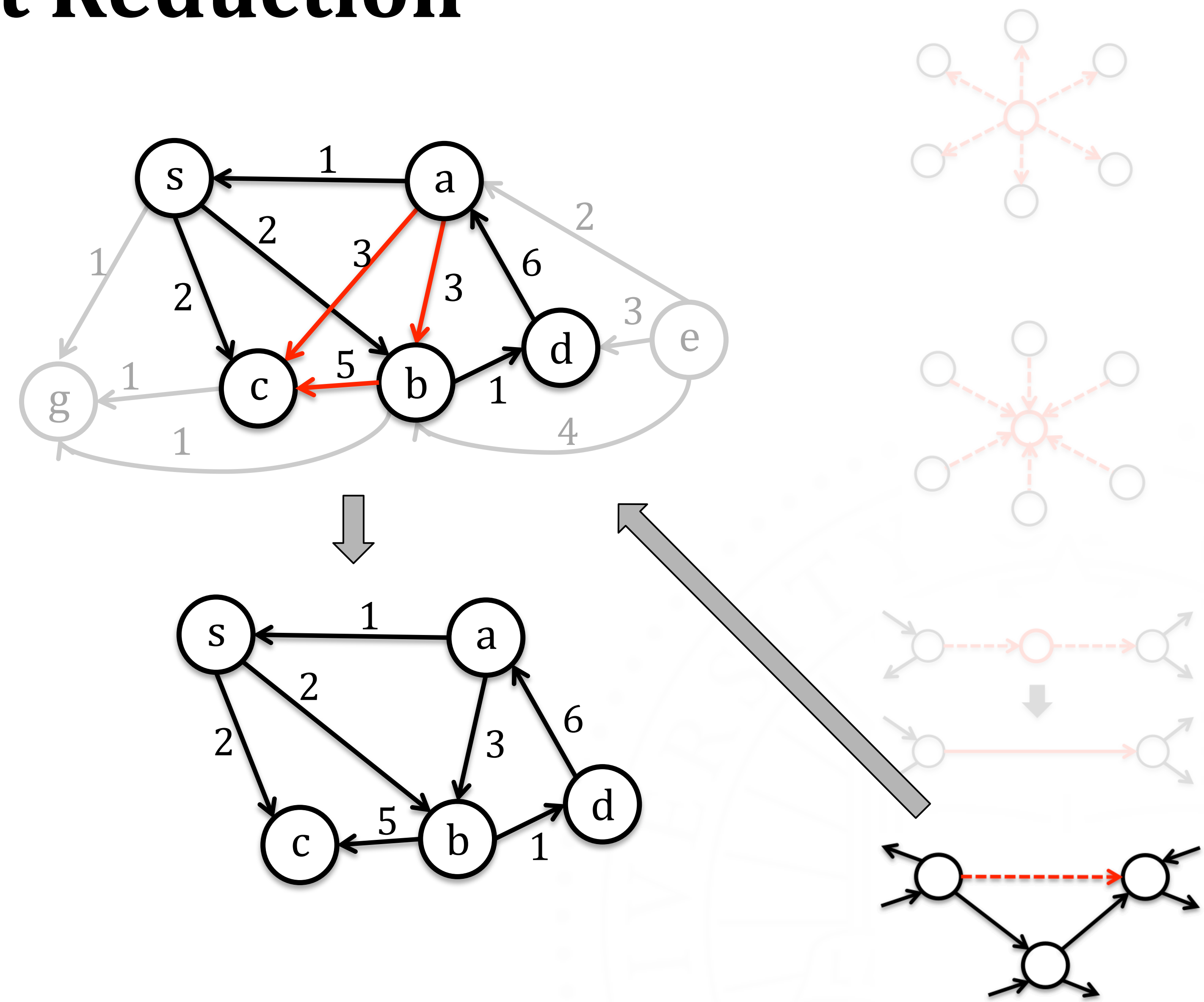


UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

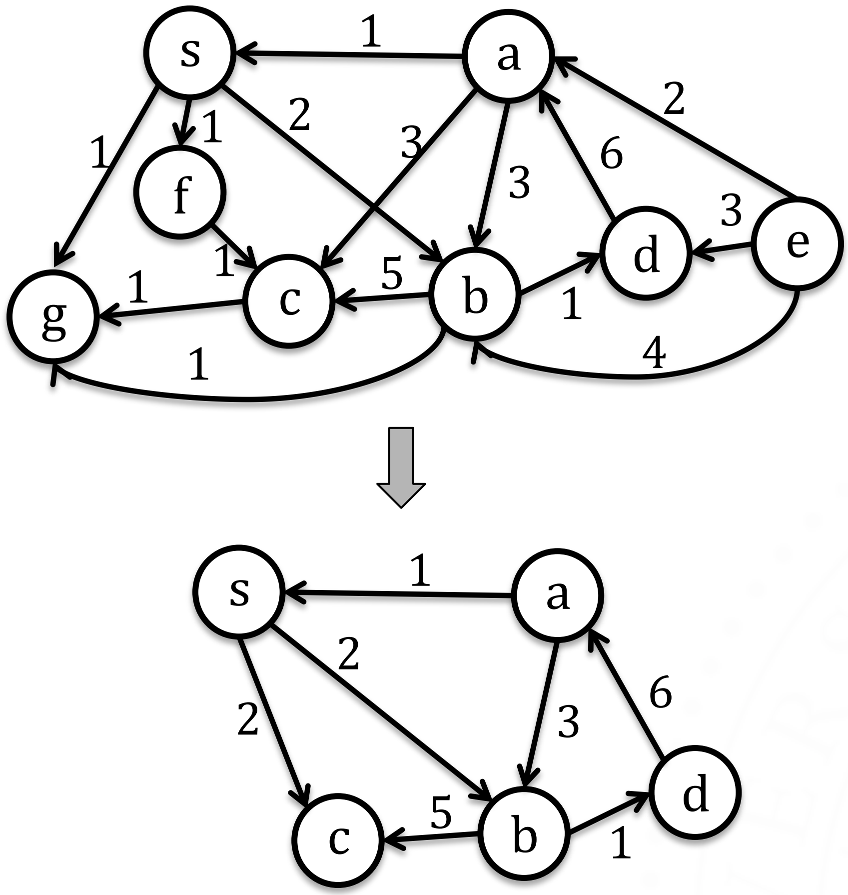
Input Reduction

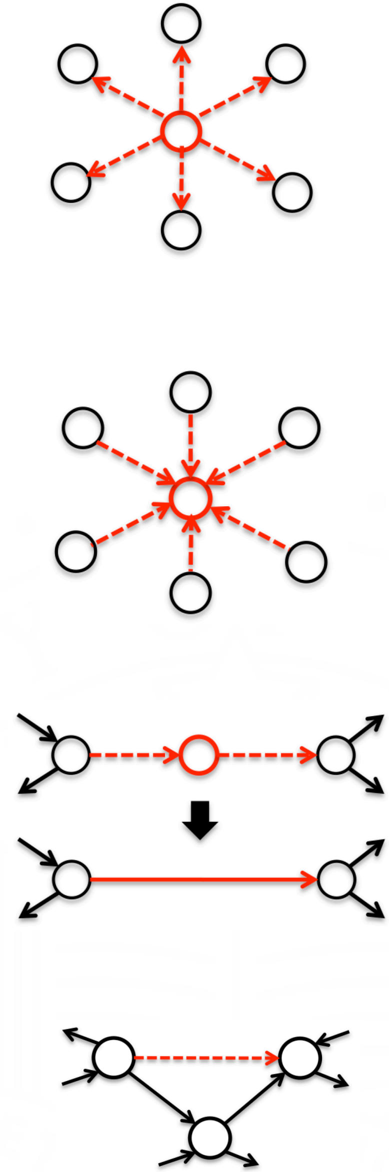


Input Reduction

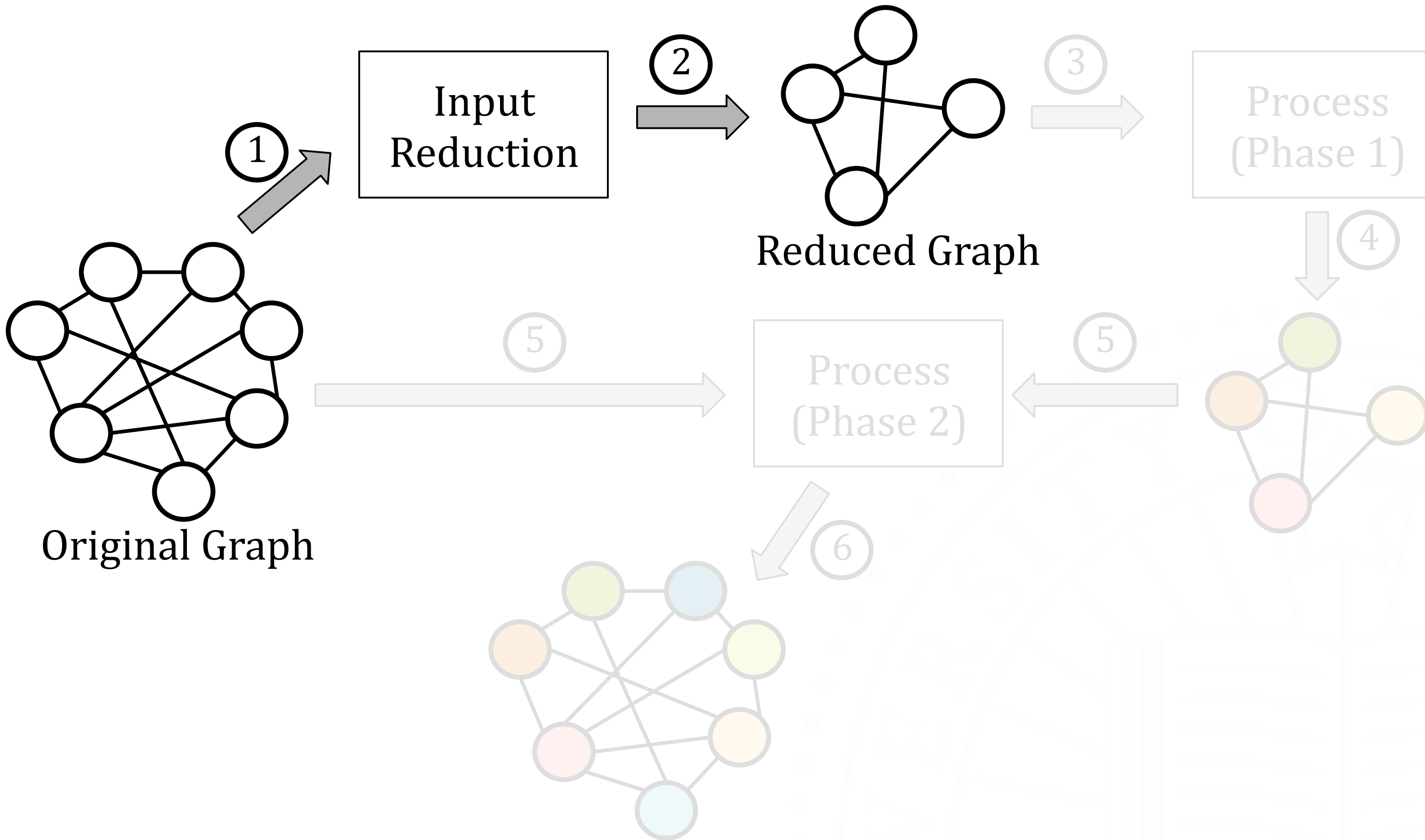


Input Reduction



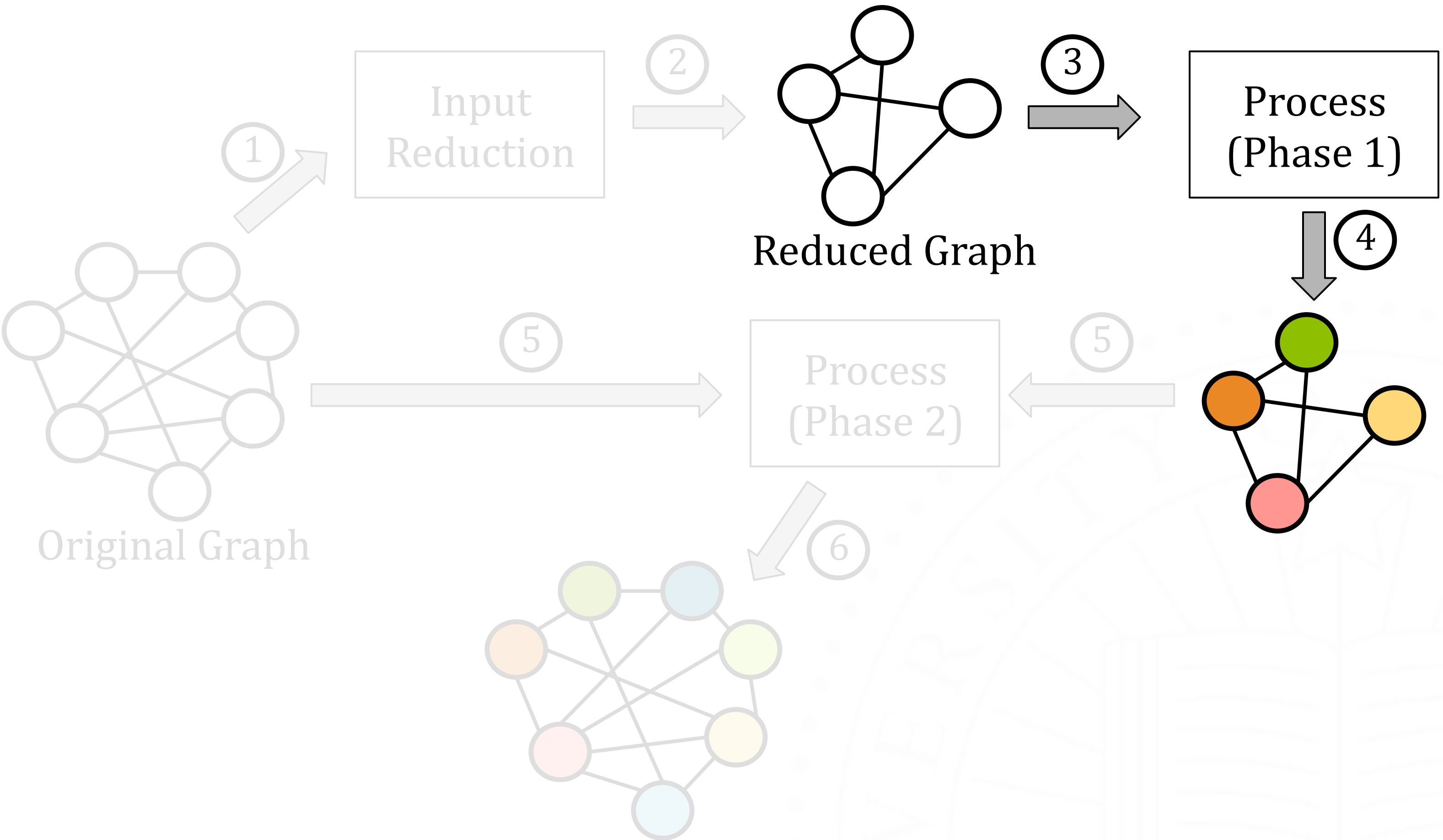


Workflow



UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

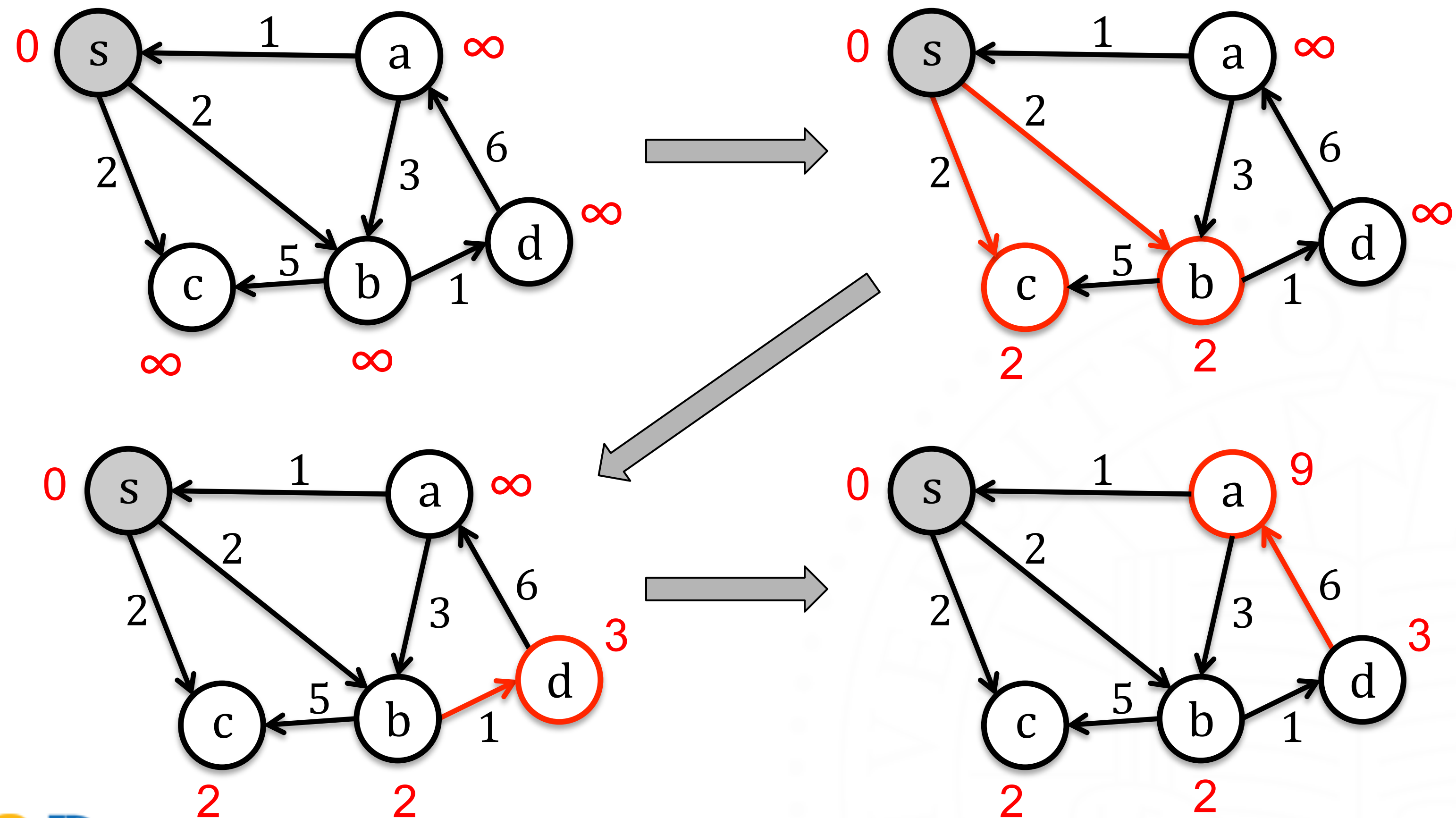
Workflow



UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

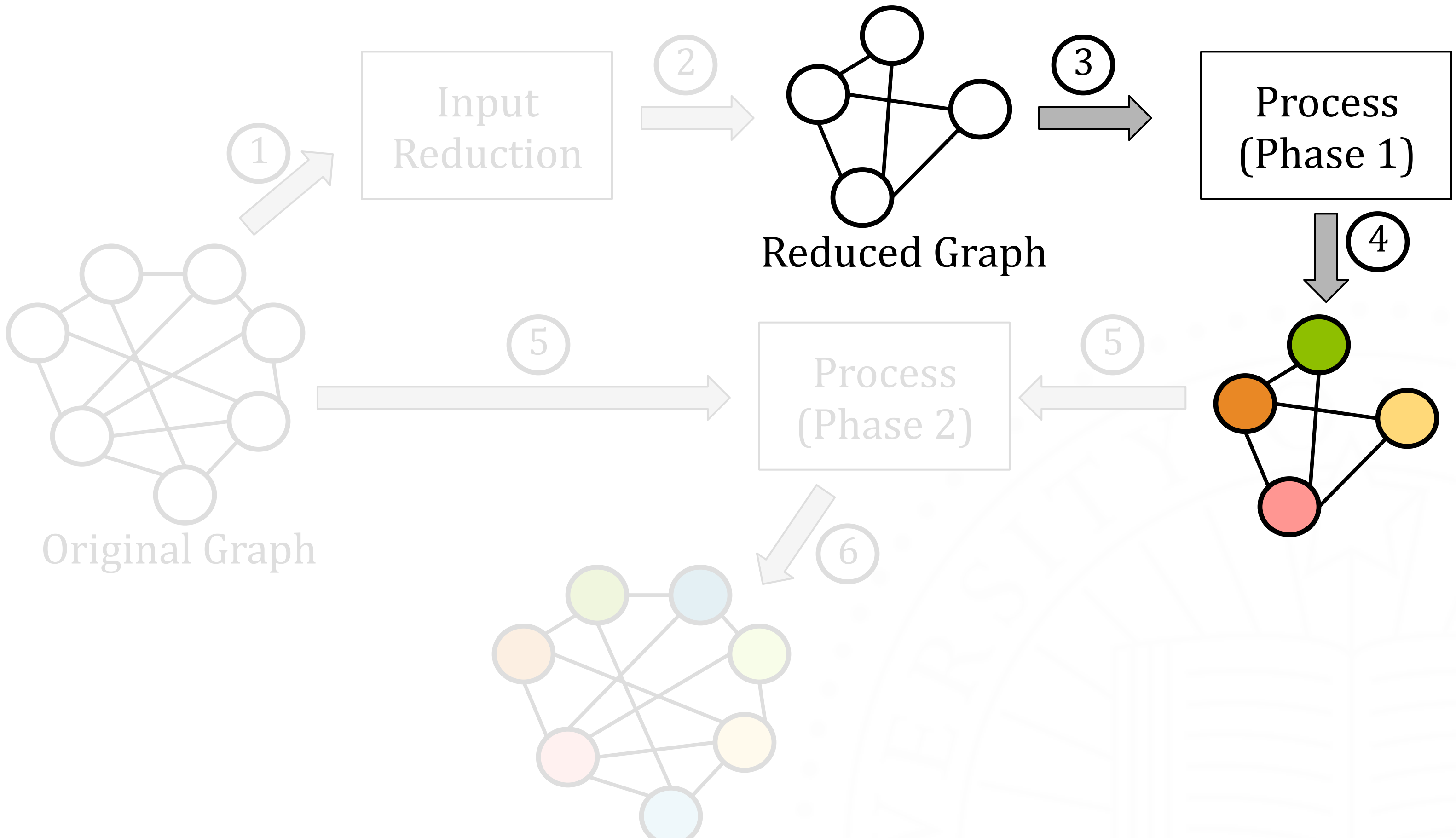
Processing Reduced Graph

- Use the original iterative algorithm



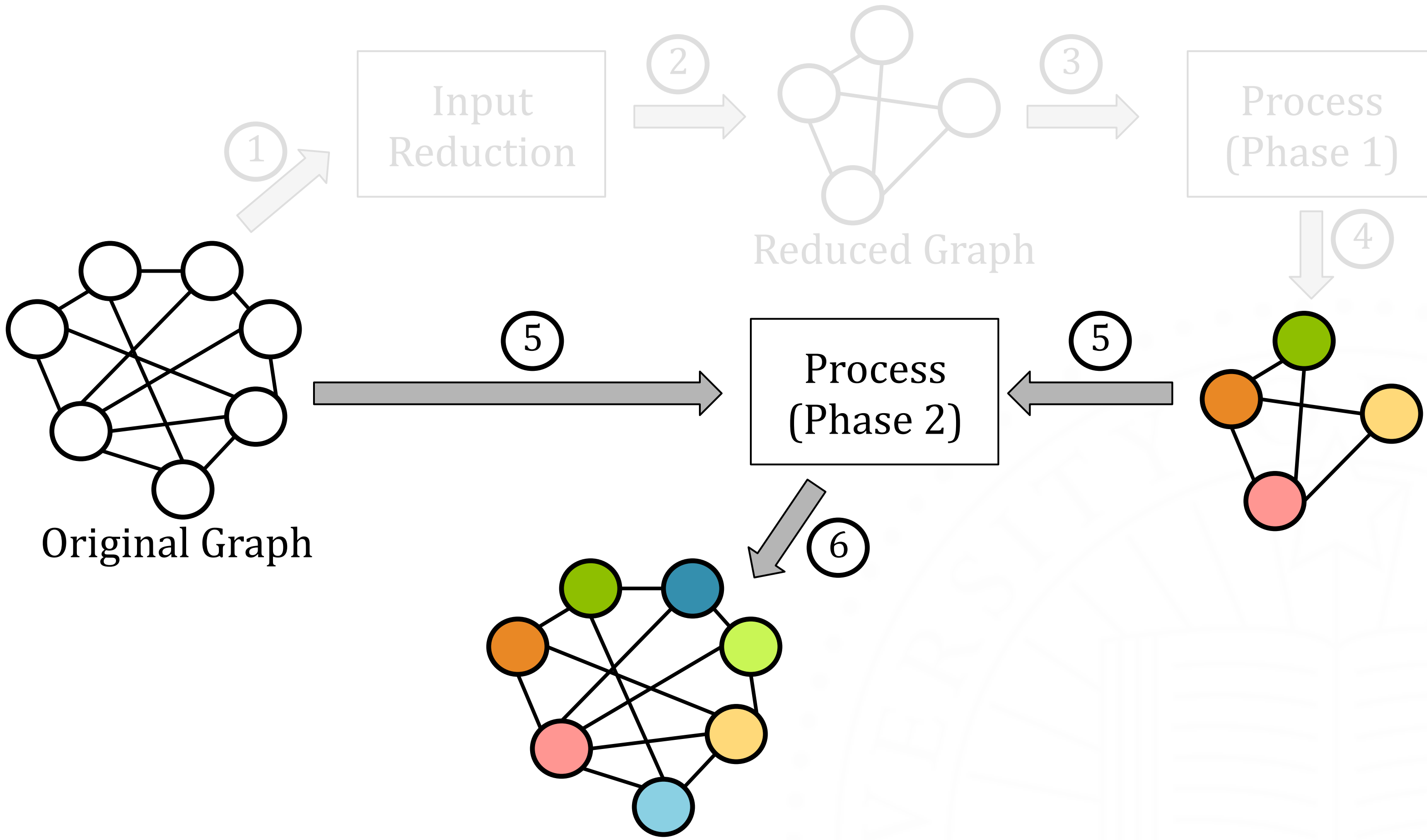
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Workflow



UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

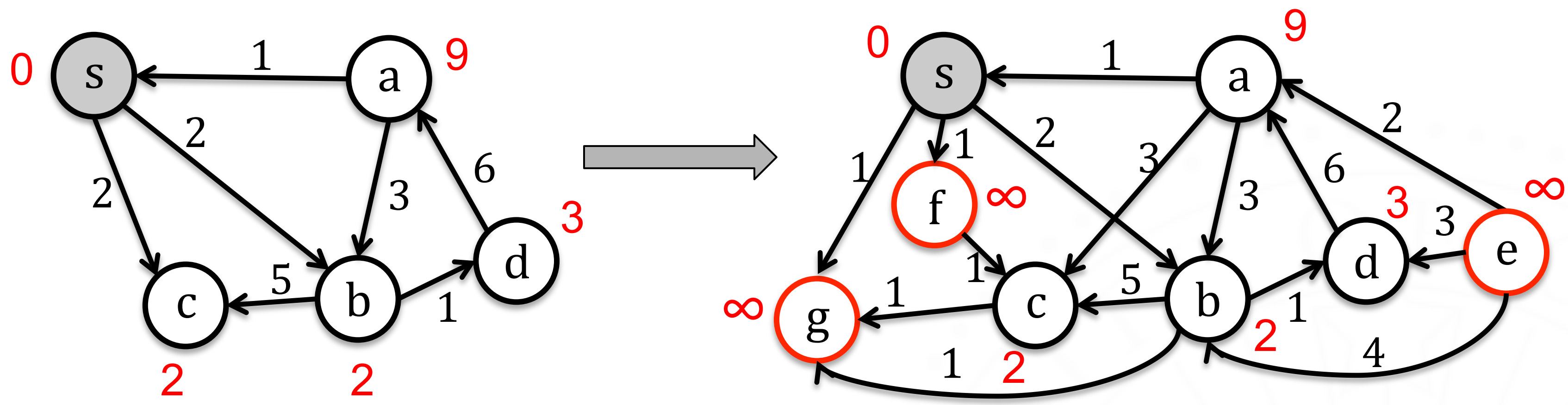
Workflow



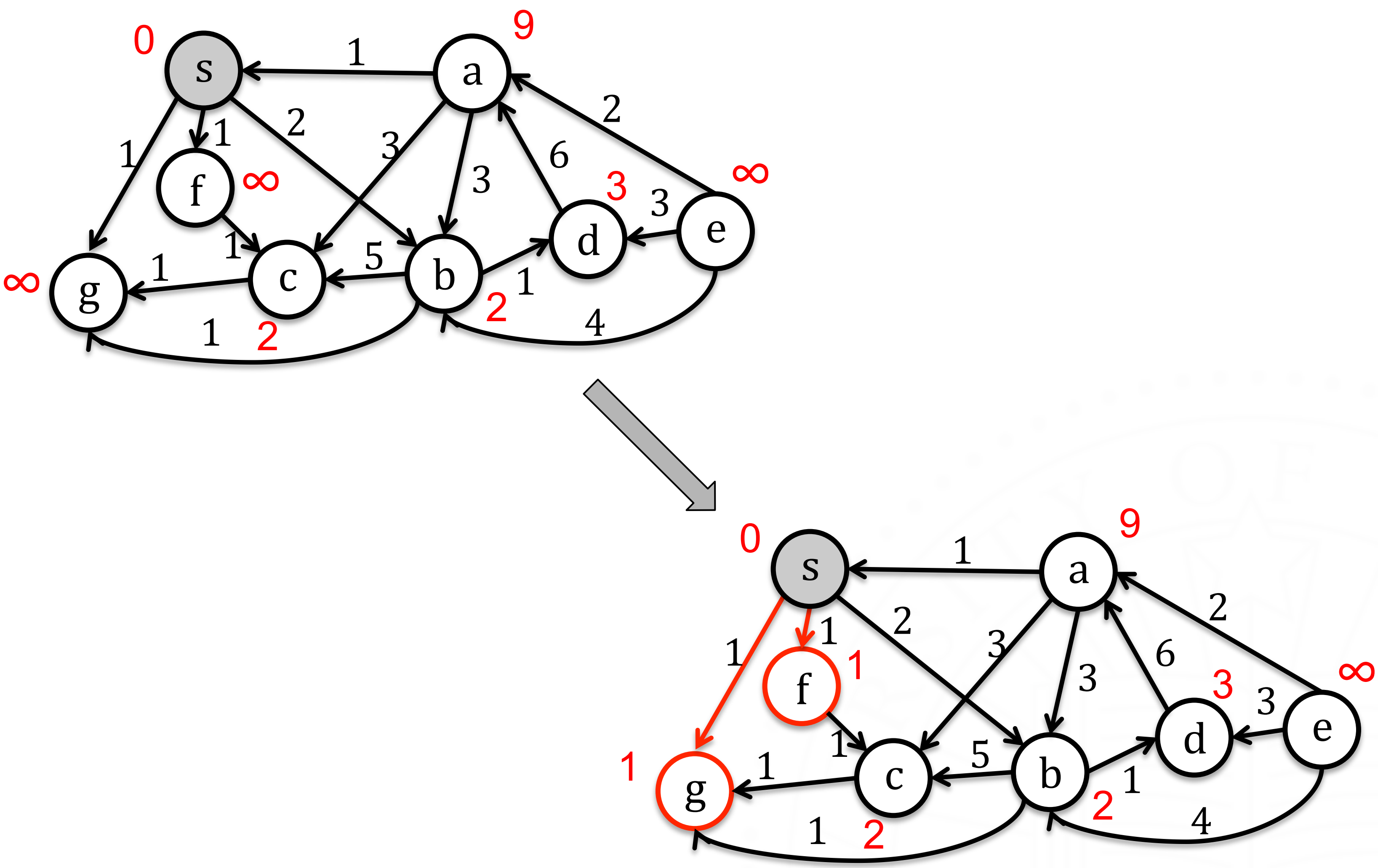
1
Input Reduction
2
Reduced Graph
3
Process (Phase 1)
4
Original Graph
5
Process (Phase 2)
5
6

Mapping Results

- Use default values for missing vertices

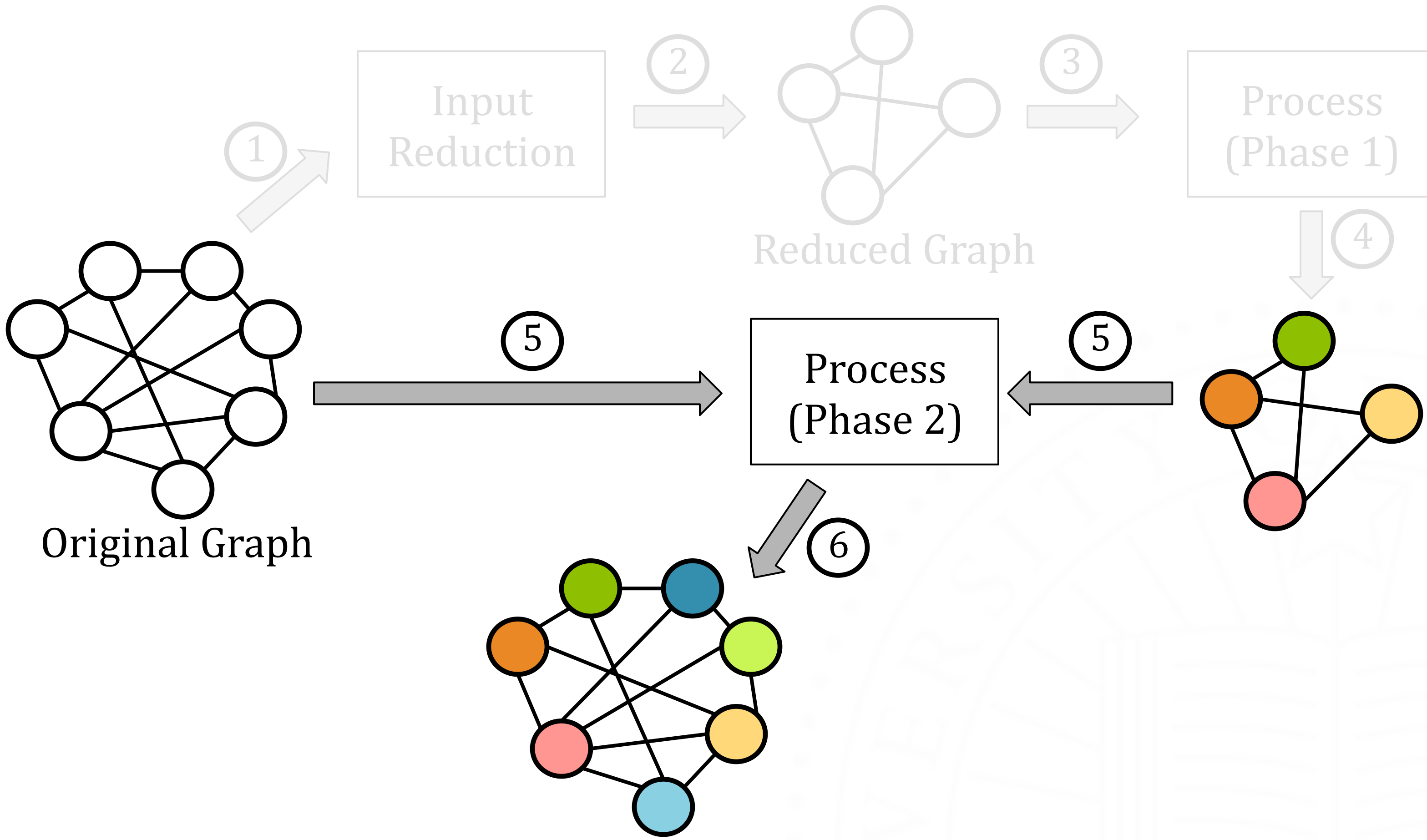


Processing Original Graph



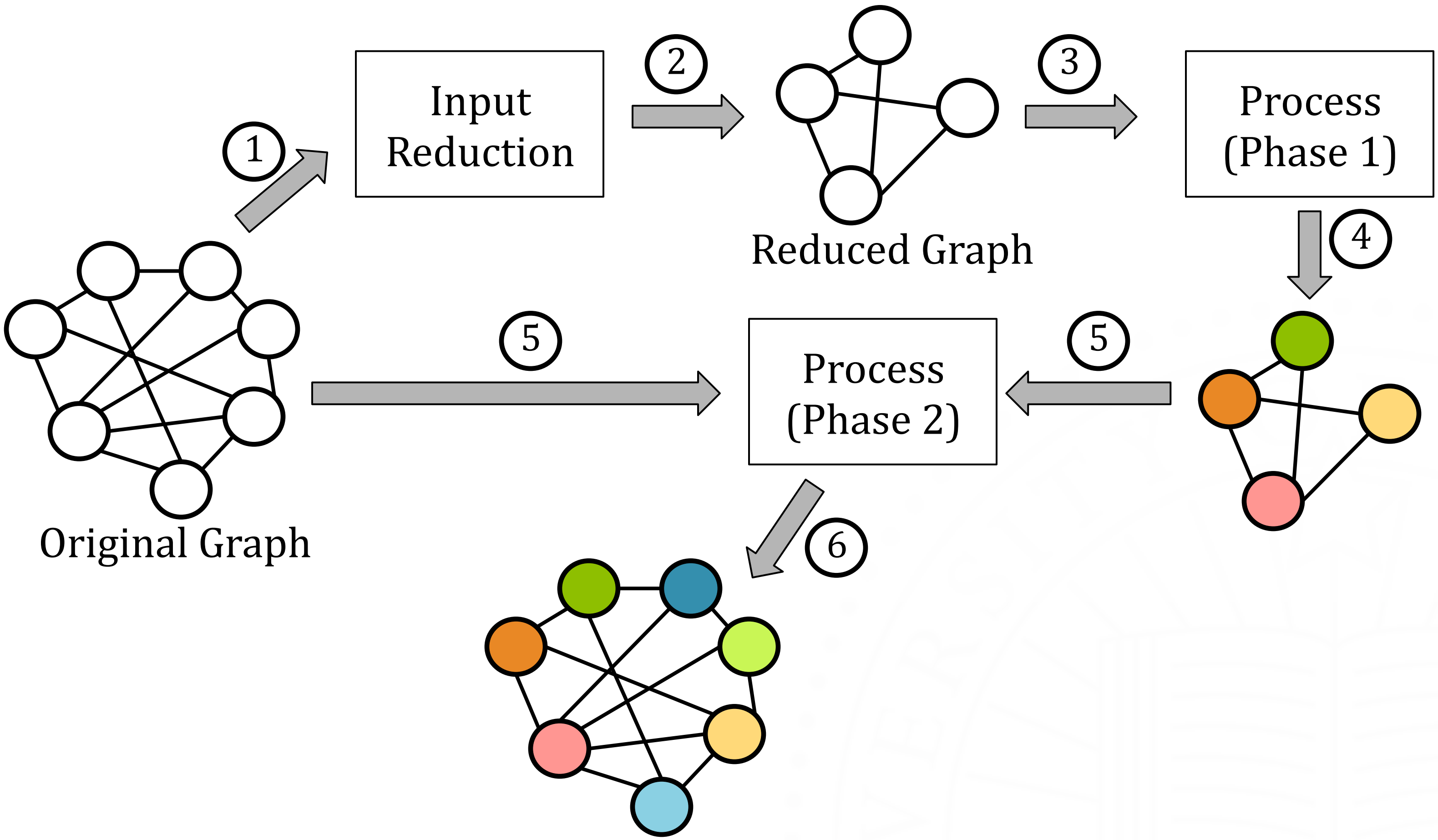
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Workflow



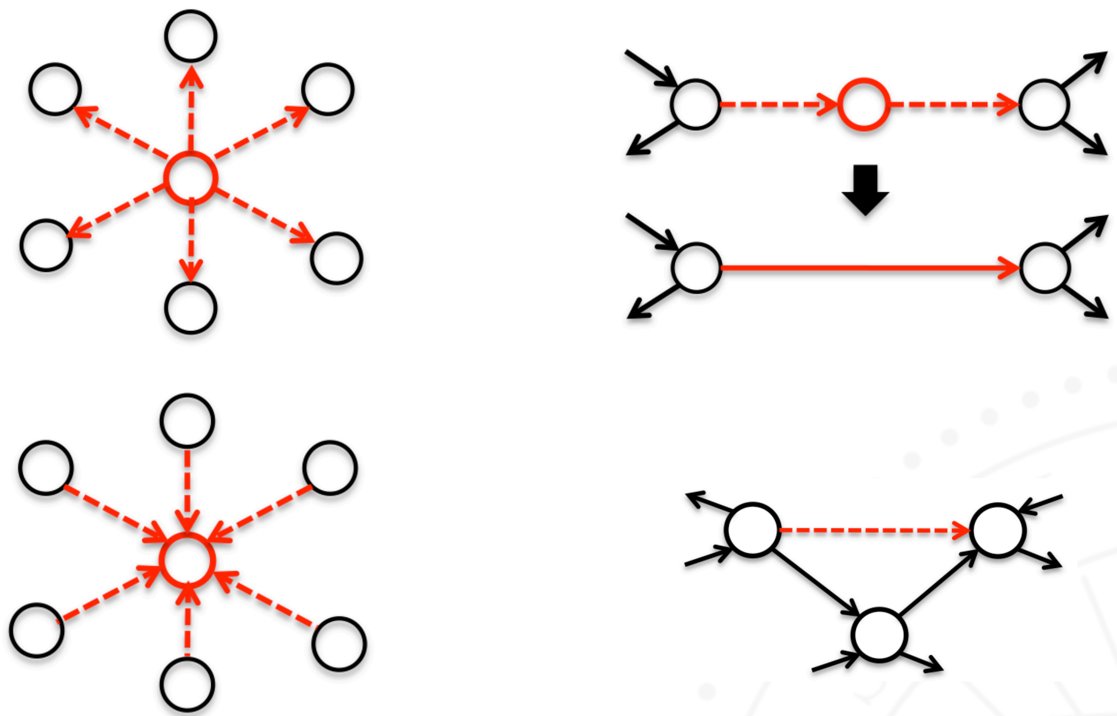
UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Workflow

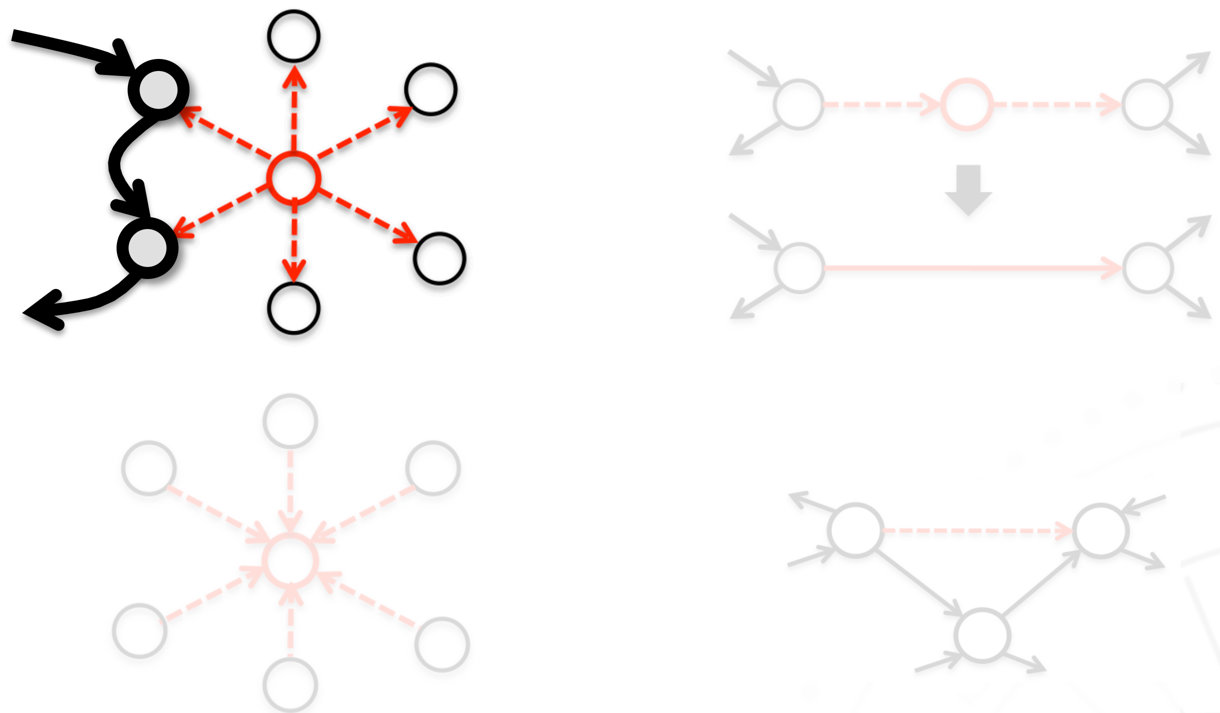


UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Correctness: SSSP Example

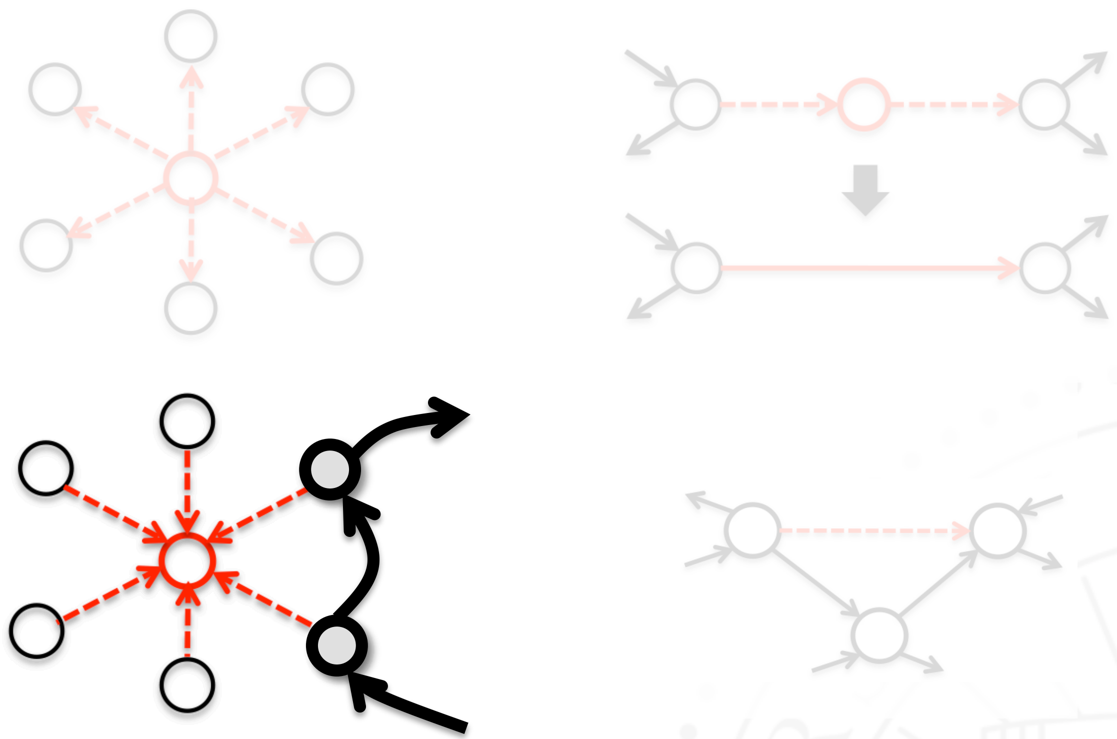


Correctness: SSSP Example



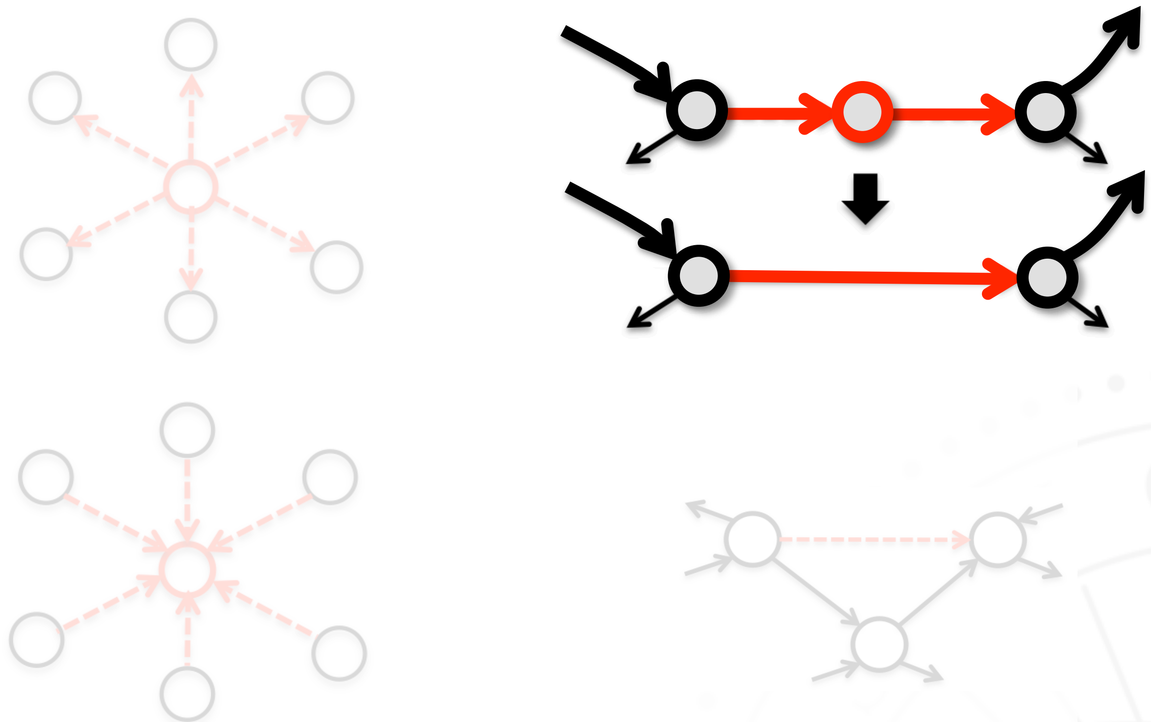
UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Correctness: SSSP Example

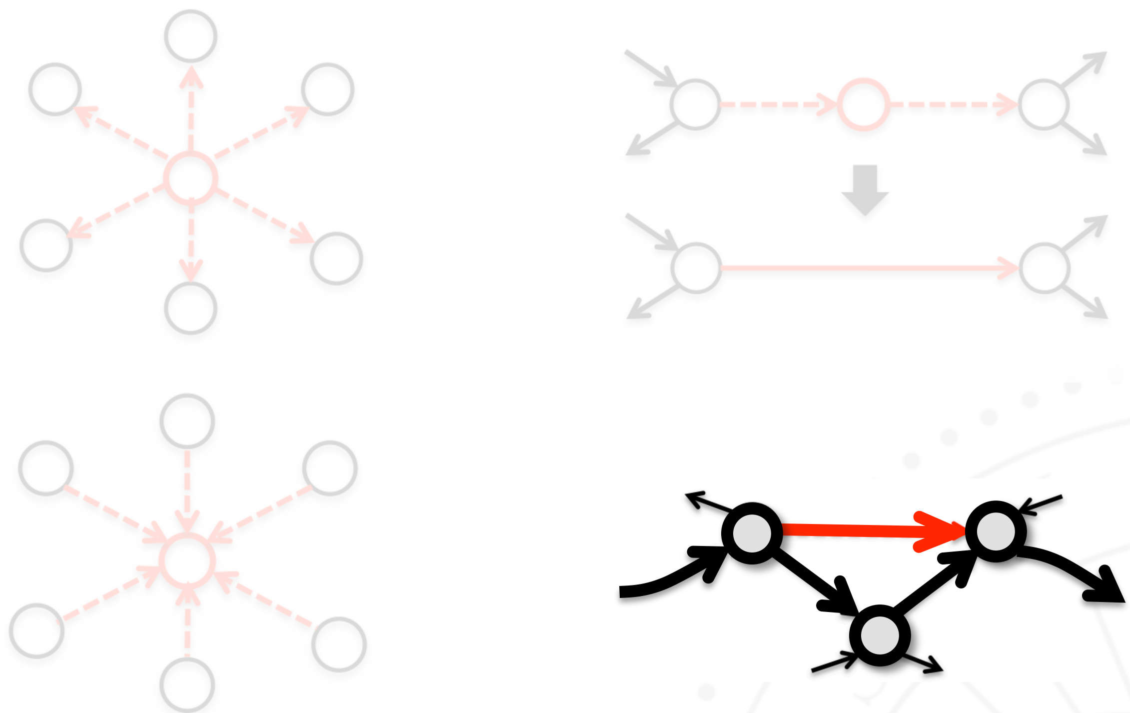


UNIVERSITY OF CALIFORNIA, RIVERSIDE

Correctness: SSSP Example



Correctness: SSSP Example



Correctness

- Transformation properties
  - Level of vertices, edges and components
  - Allow developing & reasoning for new transformations
- Algorithm behavior can be reasoned
  - Phase 2 initializations
  - Properties of aggregation function

- Correctness argued for algorithms used
  - 5 accurate and 1 approximate

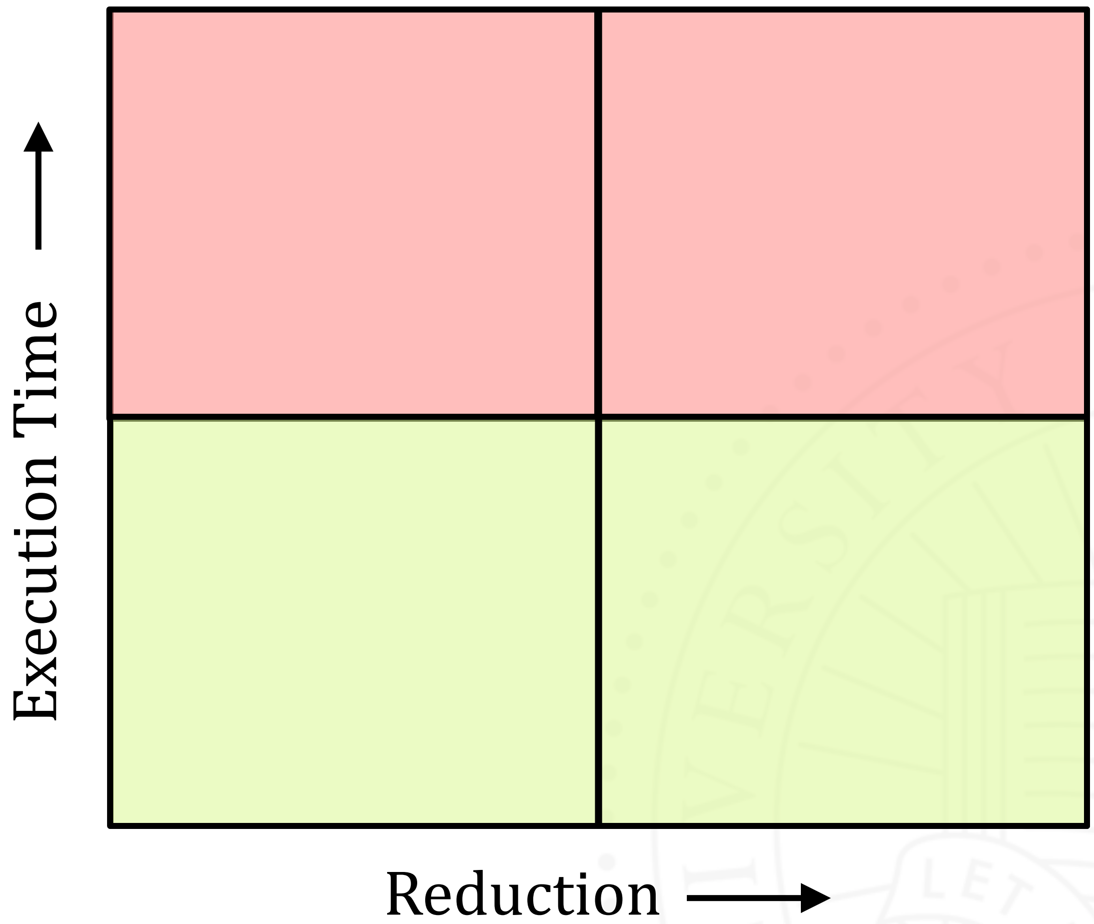
Evaluation

- Techniques independent of frameworks & processing environment
  - Incorporated in Galois [PLDI'11]
  - Single machine: 24-core, 32GB RAM
- 6 benchmarks
  - PR, SSSP, SSWP, CC, GC, CD
- 4 input graphs
  - Friendster (|E| = 2.6B), Twitter (|E| = 1.5B), UKDomain (|E| = 936M), RMAT-24 (|E| = 268M)

UNIVERSITY OF CALIFORNIA, RIVERSIDE

Reduction

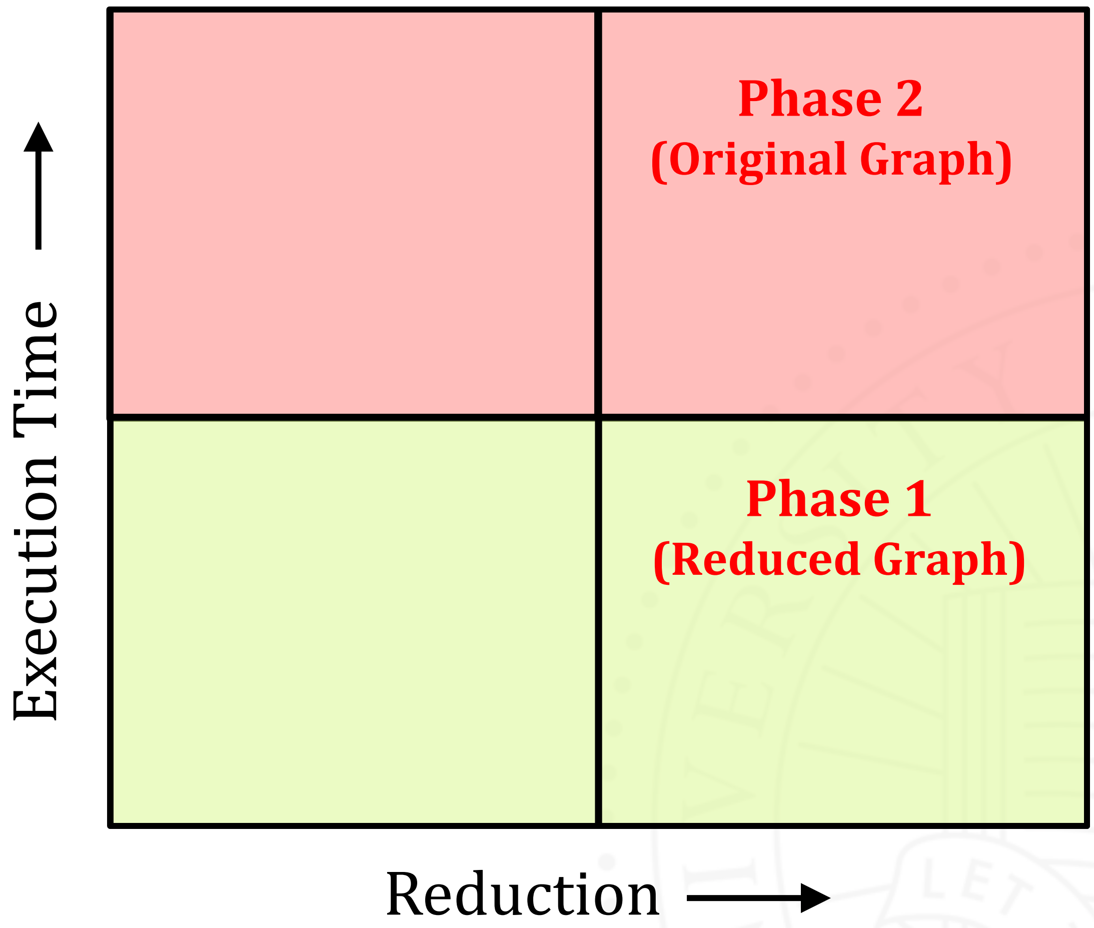
- $$ERP = \frac{|E_{REDUCED}|}{|E_{ORIGINAL}|} \times 100$$



UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

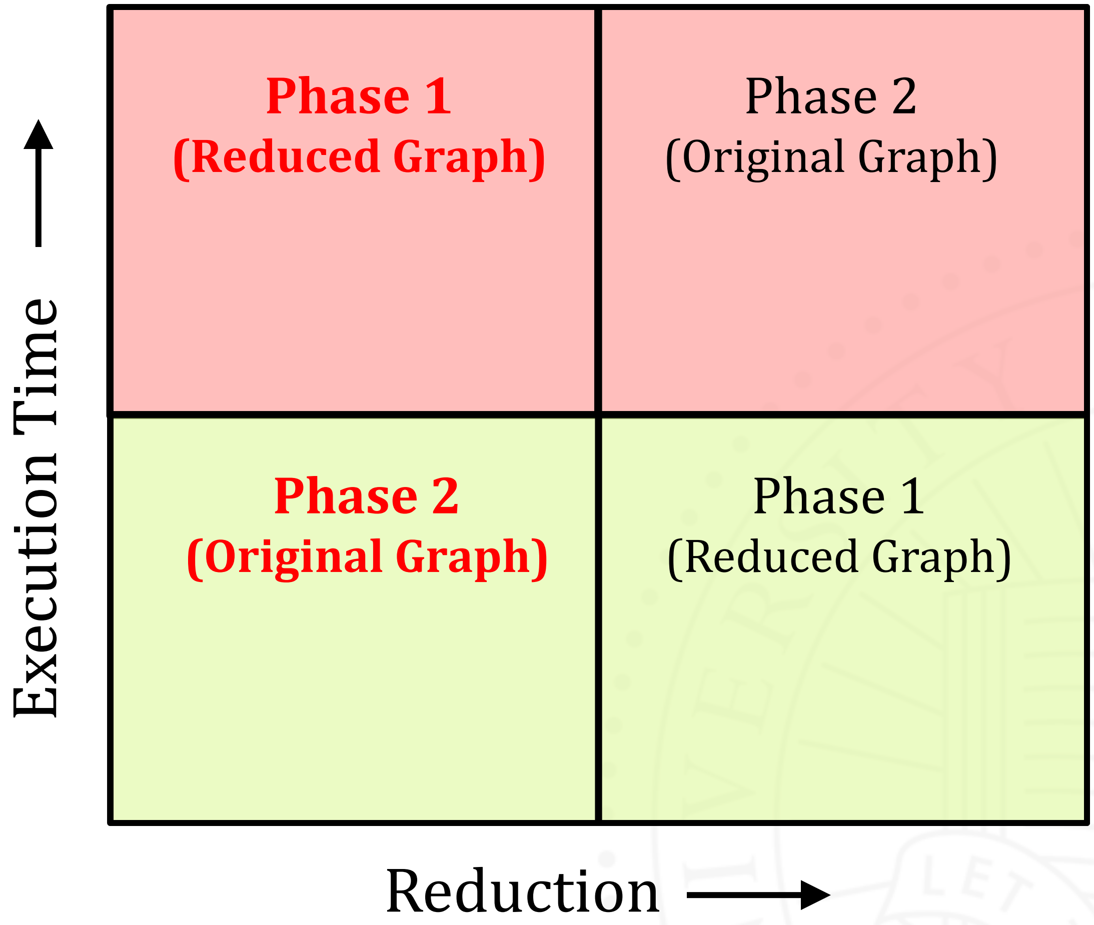
Reduction

- $ERP = \frac{|E_{REDUCED}|}{|E_{ORIGINAL}|} \times 100$



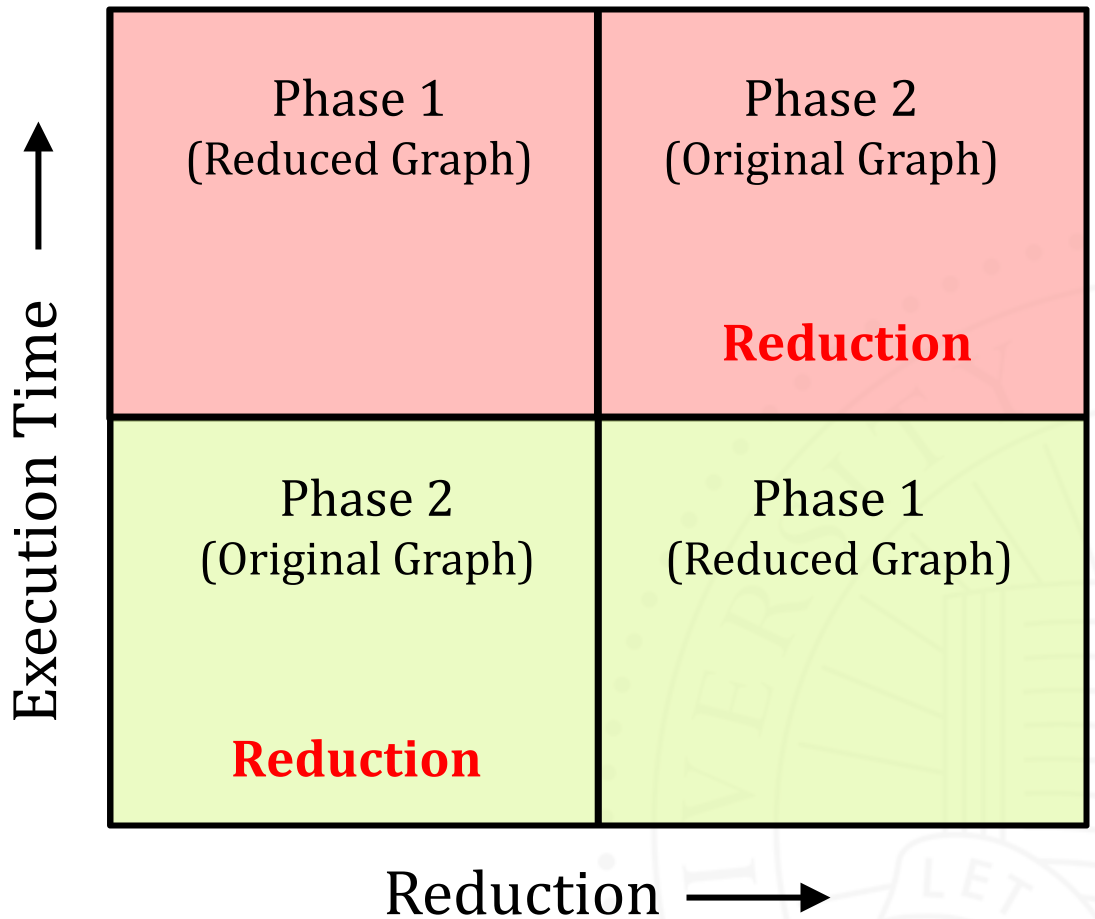
Reduction

- $ERP = \frac{|E_{REDUCED}|}{|E_{ORIGINAL}|} \times 100$



Reduction

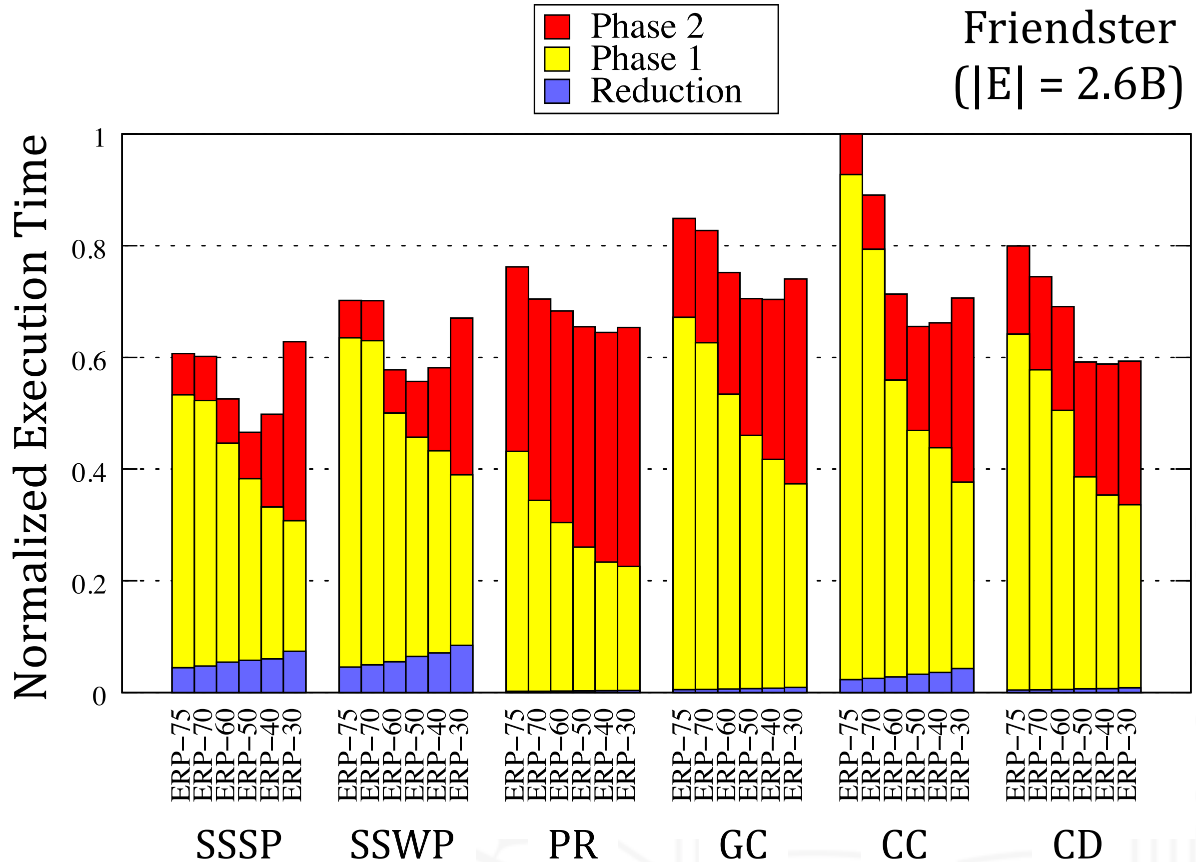
- $ERP = \frac{|E_{REDUCED}|}{|E_{ORIGINAL}|} \times 100$



Execution Time

- Speedups over parallel versions
- Speedups increase as ERP decreases up to an extent
- 1.3x - 1.7x for 75% - 50%
- Structural dissimilarity for very low ERP

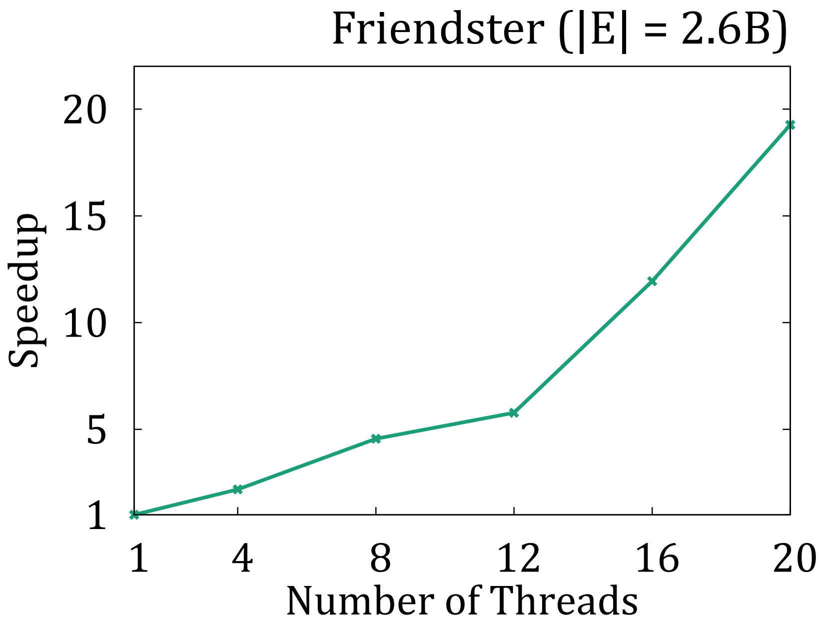
Friendster (|E| = 2.6B)

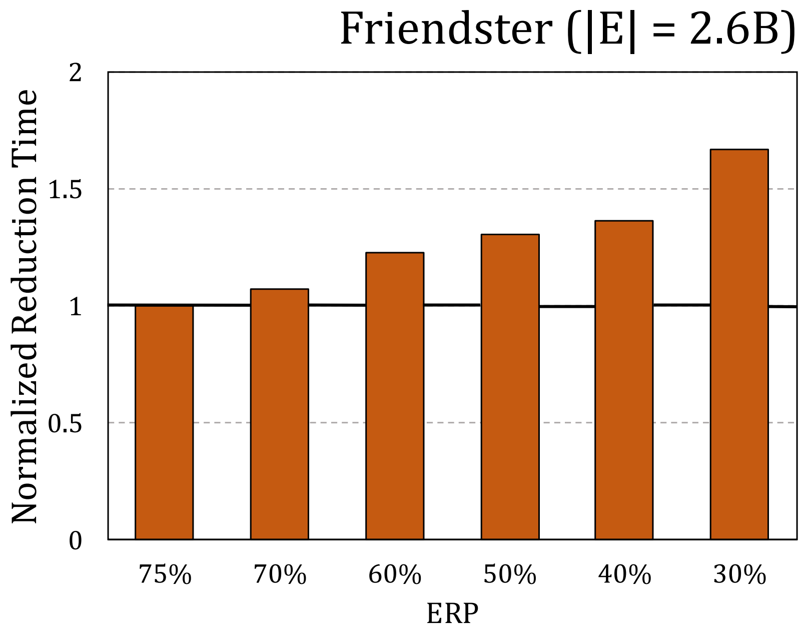


$$ERP = \frac{|E_{REDUCED}|}{|E_{ORIGINAL}|} \times 100$$

Input Reduction

- Transformations are local, i.e., parallelizable
- Higher reduction requires more work

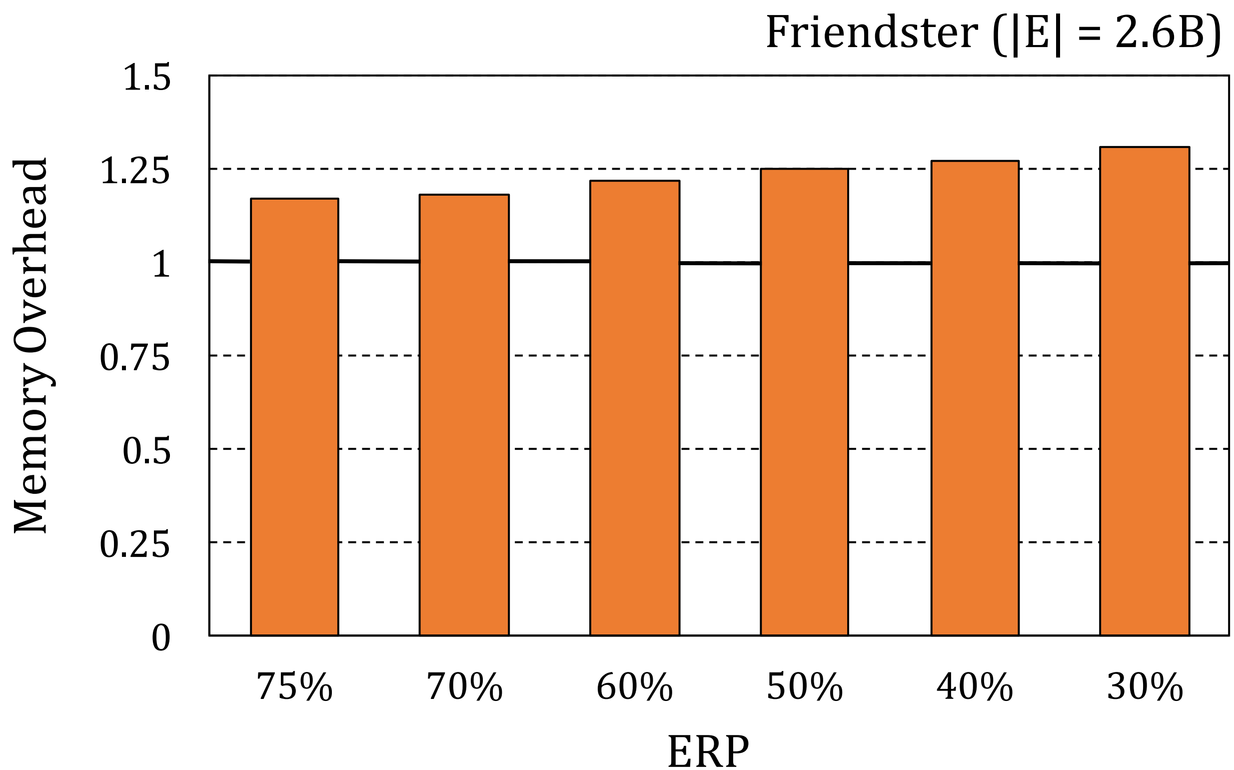




UNIVERSITY OF CALIFORNIA, RIVERSIDE

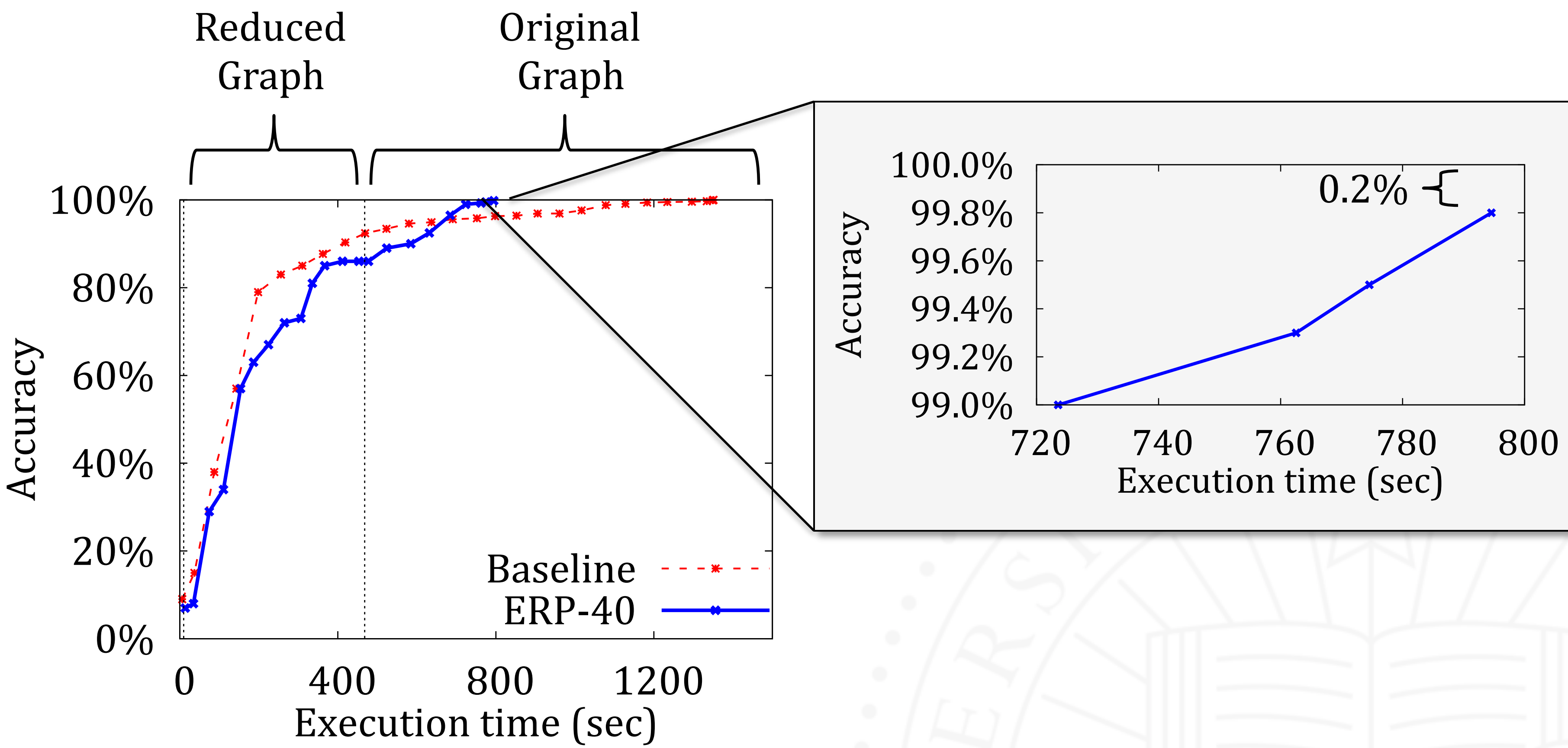
Memory Overhead

- Tracking dissimilar elements
  - Newly added vertices & edges



UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Community Detection



Friendster (|E| = 2.6B)

UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

More Results

- Contribution of individual transformations
  - Some transformations more useful than others
  - Different graphs benefit from different transformations
- Improvement in scalability
- Results for all inputs

Conclusion

- Input reduction using transformations that are
  - Light-weight
  - Parallelizable
  - General
- Correctness reasoned using fine-grained transformation properties
- Achieve 1.25-2.14x speedups

Thanks

- GRASP
  - http://grasp.cs.ucr.edu/

UNIVERSITY OF CALIFORNIA, RIVERSIDE