PhD Dissertation Defense: Systems Abstractions for Big Data Processing on a Single Machine

Steffen Maass

Advisor: Taesoo Kim

April 3, 2019

Large-scale data processing is ubiquitous

One Trillion Edges: Graph Processing at Facebook-Scale

Avery Ching
Facebook
1 Hacker Lane
Menlo Park, California
aching@fb.com

Sergey Edunov
Facebook
1 Hacker Lane
Menlo Park, California
edunov@fb.com

Maja Kabiljo
Facebook
1 Hacker Lane
Menlo Park, California
majakabiljo@fb.com

Dionysios Logothetis
Facebook
1 Hacker Lane
Menlo Park, California
dionysios@fb.com

Sambavi Muthukrishnan
Facebook
1 Hacker Lane
Menlo Park, California
sambavim@fb.com

ABSTRACT

Analyzing large graphs provides valuable insights for social networking and web companies in content ranking and recommendations. While numerous graph processing systems have been developed and evaluated on available benchmark graphs of up to 6.6B edges, they often face significant difficulties in scaling to much larger graphs. Industry graphs can be two orders of magnitude larger - hundreds of billions or up to one trillion edges. In addition to scalability challenges, real world applications often require much more complex graph processing workflows than previously evalu-

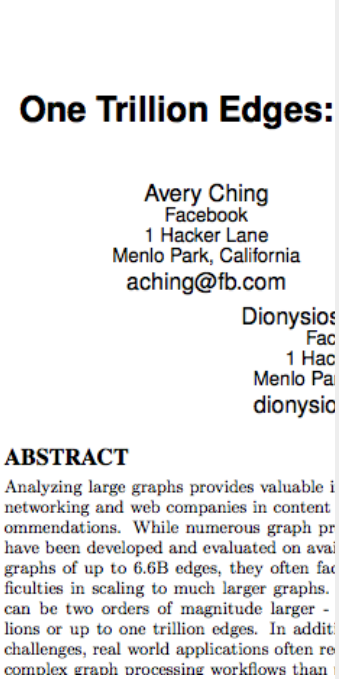
a project to run Facebook-scale graph applications in the summer of 2012 and is still the case today.

Table 1: Popular benchmark graphs.

Graph	Vertices	Edges
LiveJournal [9]	4.8M	69M
Twitter 2010 [31]	42M	1.5B
UK web graph 2007 [10]	109M	3.7B
Yahoo web [8]	1.4B	6.6B

Social networks

Large-scale data processing is ubiquitous



One Trillion Edges:

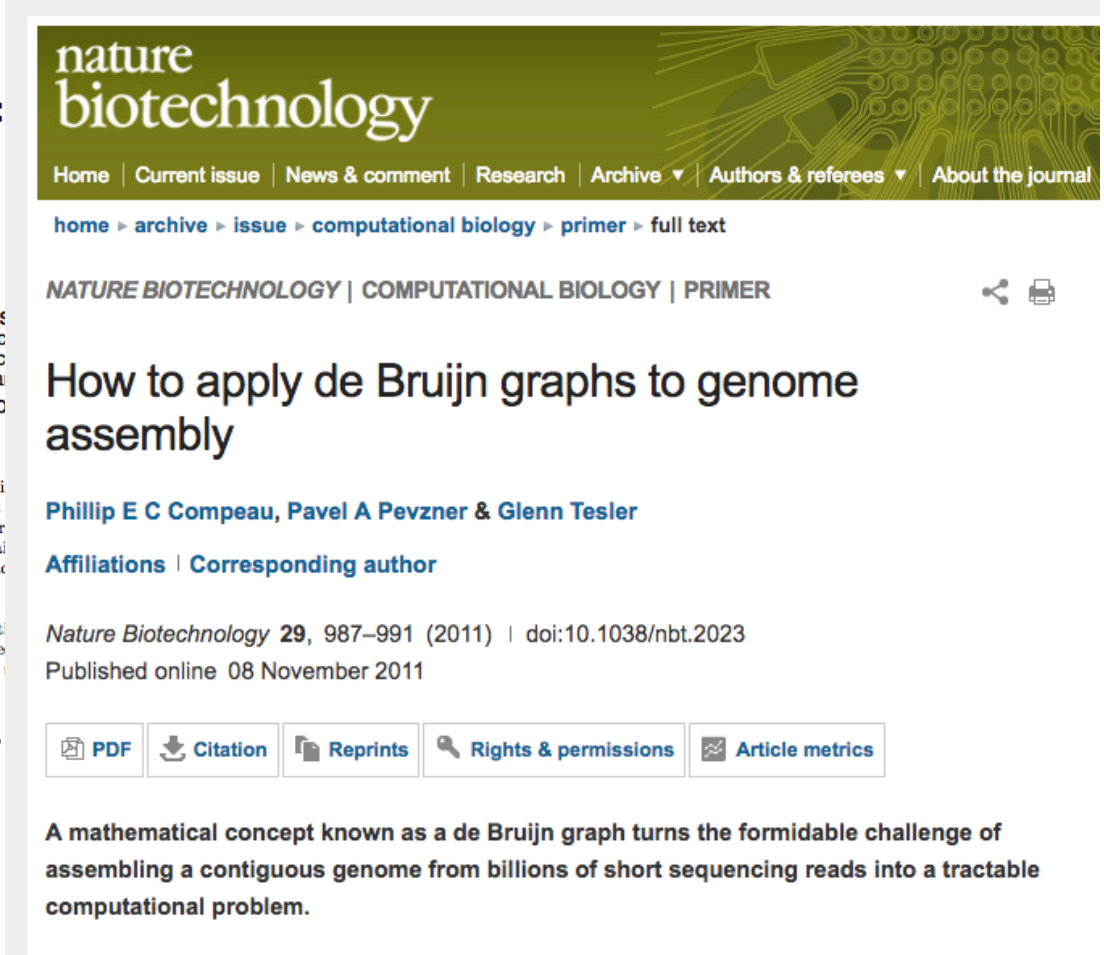
Avery Ching
Facebook
1 Hacker Lane
Menlo Park, California
aching@fb.com

Dionysios
Fac
1 Hac
Menlo Pa
dionysio

ABSTRACT

Analyzing large graphs provides valuable i
networking and web companies in content
ommendations. While numerous graph pr
have been developed and evaluated on avai
graphs of up to 6.6B edges, they often fac
ficulties in scaling to much larger graphs.
can be two orders of magnitude larger -
lions or up to one trillion edges. In additi
challenges, real world applications often re
complex graph processing workflows than

Social networks



nature biotechnology

Home | Current issue | News & comment | Research | Archive ▾ | Authors & referees ▾ | About the journal

home ▸ archive ▸ issue ▸ computational biology ▸ primer ▸ full text

NATURE BIOTECHNOLOGY | COMPUTATIONAL BIOLOGY | PRIMER

How to apply de Bruijn graphs to genome assembly

Phillip E C Compeau, Pavel A Pevzner & Glenn Tesler

Affiliations | Corresponding author

Nature Biotechnology **29**, 987–991 (2011) | doi:10.1038/nbt.2023
Published online 08 November 2011

PDF | Citation | Reprints | Rights & permissions | Article metrics

A mathematical concept known as a de Bruijn graph turns the formidable challenge of assembling a contiguous genome from billions of short sequencing reads into a tractable computational problem.

Genome analysis

Large-scale data processing is ubiquitous

One Trillion Edges:

Avery Ching
Facebook
1 Hacker Lane
Menlo Park, California
aching@fb.com

Dionysios
Fac
1 Hac
Menlo Pa
dionysio

ABSTRACT

Analyzing large graphs provides valuable i
networking and web companies in content
ommendations. While numerous graph pr
have been developed and evaluated on avai
graphs of up to 6.6B edges, they often fac
ficulties in scaling to much larger graphs.
can be two orders of magnitude larger -
lions or up to one trillion edges. In additi
challenges, real world applications often re
complex graph processing workflows than

Social networks

nature biotechnology

Home | Current issue | News & con

home ▸ archive ▸ issue ▸ computa

NATURE BIOTECHNOLOGY | CO

How to apply de E
assembly

Phillip E C Compeau, Pavel A P

Affiliations | Corresponding aut

Nature Biotechnology 29, 987–99
Published online 08 November 20

PDF | Citation | Reprint

A mathematical concept known
assembling a contiguous genor
computational problem.

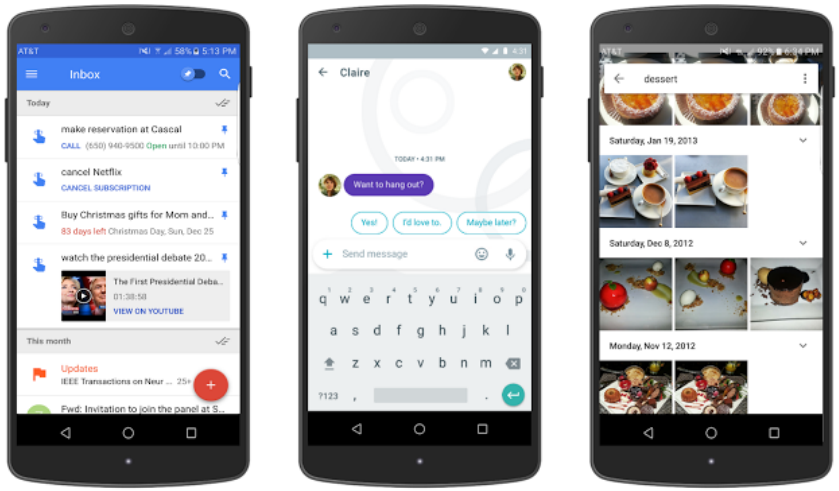
Genome analysi

Graph-powered Machine Learning at Google

Thursday, October 06, 2016

Posted by Sujith Ravi, Staff Research Scientist, Google Research

Recently, there have been significant advances in Machine Learning that enable computer systems to solve complex real-world problems. One of those advances is Google's large scale, graph-based machine learning platform, built by the Expander team in Google Research. A technology that is behind many of the Google products and features you may use everyday, graph-based machine learning is a powerful tool that can be used to power useful features such as reminders in Inbox and smart messaging in Allo, or used in conjunction with deep neural networks to power the latest image recognition system in Google Photos.



Learning with Minimal Supervision

Graphs enable Machine Learning

Powerful single machines available

sgi The Trusted Leader in High Performance Computing

Products Solutions Services Partners About SGI

More sockets. More memory. More SAP HANA.

by Cori Pasinetti on July 29, 2015

Tweet 0 Share 0 Share 1 +1 2

SGI UV 300H 20-Socket Appliance Certified by SAP to Run SAP HANA® Under Controlled Availability
Announcing the first 20-socket SAP HANA-certified in-memory server!

SGI announced today that the SGI® UV™ 300H is now SAP®-certified to run the SAP HANA® platform in controlled availability at 20-sockets–delivering up to 15 terabytes (TB) of in-memory computing capacity in a single node. Asserting the value of key enhancements in support package stack 10 (SPS10) for SAP HANA and SAP's close collaboration with system providers, SGI UV 300H delivers outstanding single-node performance and simplicity for enterprises moving to SAP HANA to gain business breakthroughs.

SGI UV 300H is a specialized offering in the SGI® UV™ server line for in-memory computing that enables enterprises to further unlock value from information in real-time, boost innovation, and lower IT costs with SAP HANA. Featuring a highly differentiated single-node architecture, the system delivers significant performance advantages for businesses running SAP® Business Suite 4 SAP HANA (SAP S/4HANA) and complex analytics at extreme scale. The single-node simplicity also helps enterprises eliminate overhead associated with clustered environments, streamline high availability, and scale-up seamlessly as data volumes grow with near-linear performance.

Integrated with the recently announced SAP HANA SPS10, SGI UV 300H capitalizes on deep collaboration between SAP, Intel and SGI to optimize SAP HANA-based workloads on multicore NUMA (non-uniform memory access) systems. This enables enterprises to leverage single-node systems with very large memory capacity and

Gigabytes of RAM

Powerful single machines available

sgi The Trusted Leader in High Performance Computing

Products Solutions

More sockets. More

by Cori Pasinetti on July 29, 2015

Tweet 0 Share 0

SGI UV 300H 20-Socket Applian
Announcing the first 20-socket

SGI announced today that the SGI
controlled availability at 20-sockets
single node. Asserting the value of
SAP's close collaboration with syste
simplicity for enterprises moving to

SGI UV 300H is a specialized offerir
server line for in-memory computin
enterprises to further unlock value
real-time, boost innovation, and lov
HANA. Featuring a highly differentia
architecture, the system delivers si
advantages for businesses running
4 SAP HANA (SAP S/4HANA) and cc
extreme scale. The single-node sim
enterprises eliminate overhead assc
environments, streamline high avai
seamlessly as data volumes grow w
performance.

Integrated with the recently annour
SPS10, SGI UV 300H capitalizes on
between SAP, Intel and SGI to optir
workloads on multicore NUMA (non
access) systems. This enables ente

Gigabytes of

Intel Unveils Plans for Knights Mill, a Xeon Phi for Deep Learning

Michael Feldman | August 18, 2016 01:33 CEST

At the Intel Developer Forum (IDF) this week in San Francisco, Intel revealed it is working on a new Xeon Phi processor aimed at deep learning applications. Diane Bryant, executive VP and GM of Intel's Data Center Group, unveiled the new chip, known as Knights Mill, during her IDF keynote address on Wednesday.

Powerful many-core coprocessors

Powerful single machines available

sgi The Trusted Leader in High Performance Computing

Products Solutions

More sockets. More

by Cori Pasinetti on July 29, 2015

Tweet 0 Share 0

SGI UV 300H 20-Socket Applian
Announcing the first 20-socket

SGI announced today that the SGI(controlled availability at 20-sockets single node. Asserting the value of SAP's close collaboration with syste simplicity for enterprises moving to

SGI UV 300H is a specialized offerir server line for in-memory computin enterprises to further unlock value real-time, boost innovation, and lov HANA. Featuring a highly differentia architecture, the system delivers si advantages for businesses running 4 SAP HANA (SAP S/4HANA) and cc extreme scale. The single-node sim enterprises eliminate overhead assc environments, streamline high avai seamlessly as data volumes grow w performance.

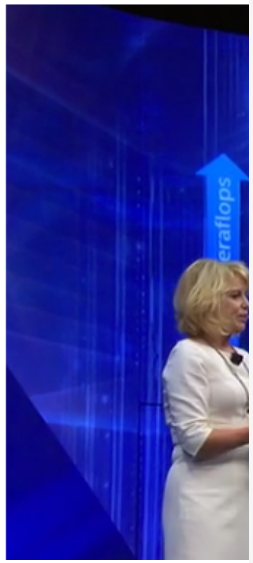
Integrated with the recently annour SPS10, SGI UV 300H capitalizes on between SAP, Intel and SGI to optir workloads on multicore NUMA (non access) systems. This enables ente

Gigabytes of

Intel Unveils P
Learning

Michael Feldman | August 18, 2016

At the Intel Developer Forum (I
at deep learning applications. D
as Knights Mill, during her IDF

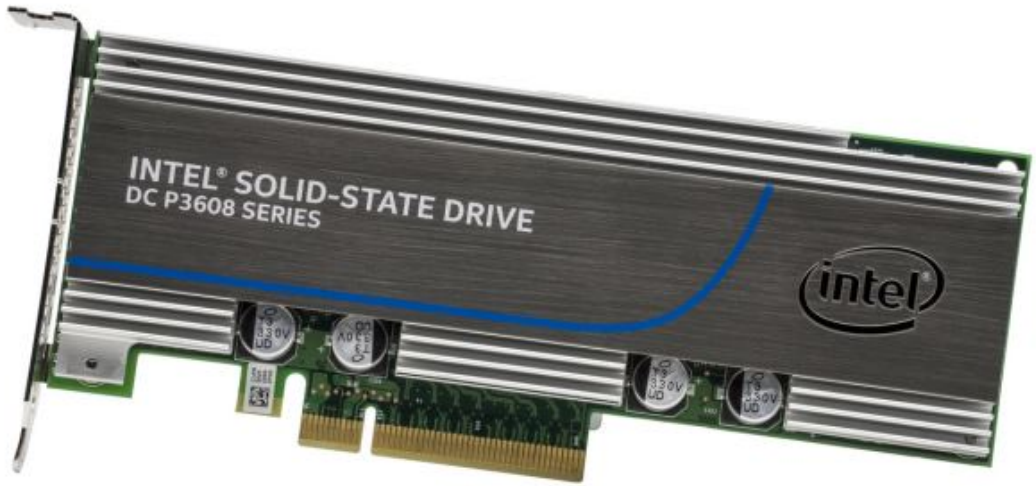


Powerful ma

Intel Announces SSD DC P3608 Series

by Billy Tallis on September 23, 2015 12:00 PM EST

Posted in Intel Storage SSDs PCIe SSD Enterprise SSDs



Intel is introducing a new family of enterprise PCIe SSDs with the aim of outperforming their existing DC P3600 series and even beating the DC P3700 series in many metrics. To do this, they've essentially put two P3600 SSDs on to one expansion card and widened the interface to 8 lanes of PCIe 3.0. While this does come across as a bit of a quick and dirty solution, it is a very straightforward way for Intel to deliver higher performance, albeit at the cost of sharply increased power consumption.

Fast, large-capacity non-volatile Memory

Powerful single machines available

sgi The Trusted Leader in High Performance Computing

Products Solutions

More sockets. More

by Cori Pasinetti on July 29, 2015

Tweet 0 Share 0

SGI UV 300H 20-Socket Applian
Announcing the first 20-socket

SGI announced today that the SGI
controlled availability at 20-sockets
single node. Asserting the value of
SAP's close collaboration with syste

Intel Unveils P
Learning

Michael Feldman | August 18, 2016

At the Intel Developer Forum (I
at deep learning applications. D
as Knights Mill, during her IDF

Intel Announces SSD DC P3608 Series

by Billy Tallis on September 23, 2015 12:00 PM EST

Posted in Intel Storage SSDs PCIe SSD Enterprise SSDs

⇒ Enable tera-scale data processing on a single machine
However: design and systems-level mechanisms required!

workloads on multicore NUMA (non
access) systems. This enables ente

Gigabytes of

Powerful ma

Intel is introducing a new family of enterprise PCIe SSDs with the aim of outperforming their existing DC P3600 series and even beating the DC P3700 series in many metrics. To do this, they've essentially put two P3600 SSDs on to one expansion card and widened the interface to 8 lanes of PCIe 3.0. While this does come across as a bit of a quick and dirty solution, it is a very straightforward way for Intel to deliver higher performance, albeit at the cost of sharply increased power consumption.

Fast, large-capacity non-volatile Memory

Thesis statement and approaches

Large-scale big data analytics is possible on a single machine using systems and design-level abstractions for commodity and heterogeneous single machines.

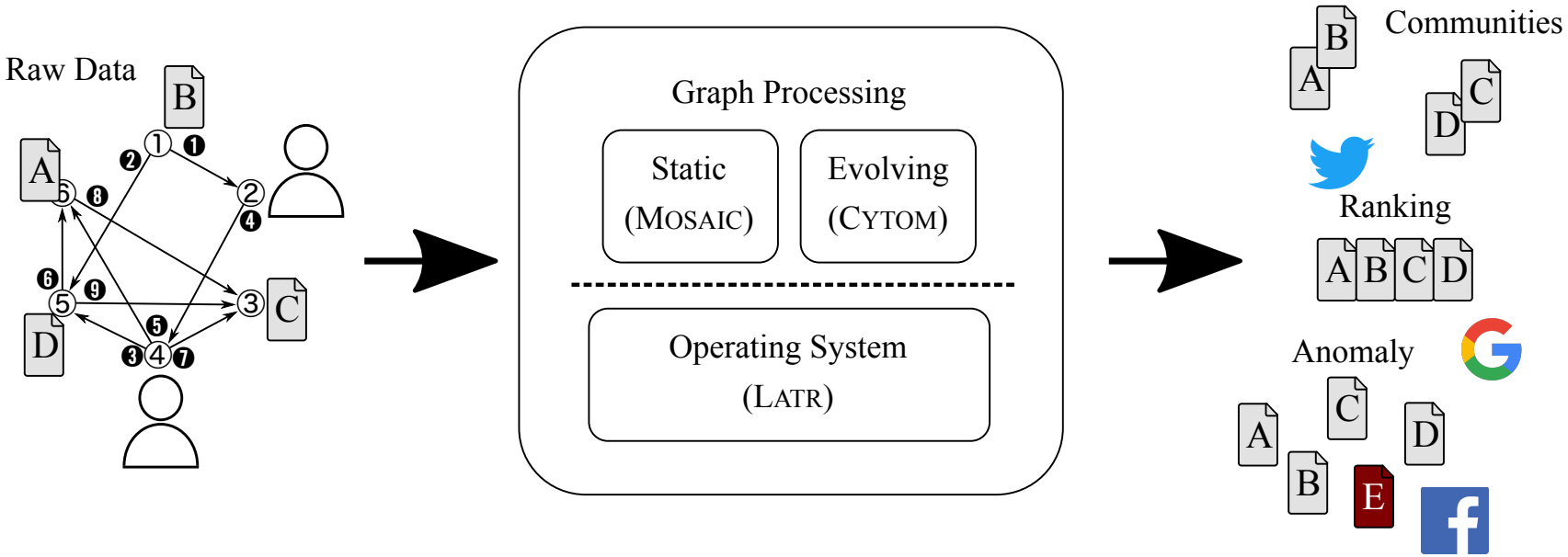
Thesis statement and approaches

Large-scale big data analytics is possible on a single machine using systems and design-level abstractions for commodity and heterogeneous single machines.

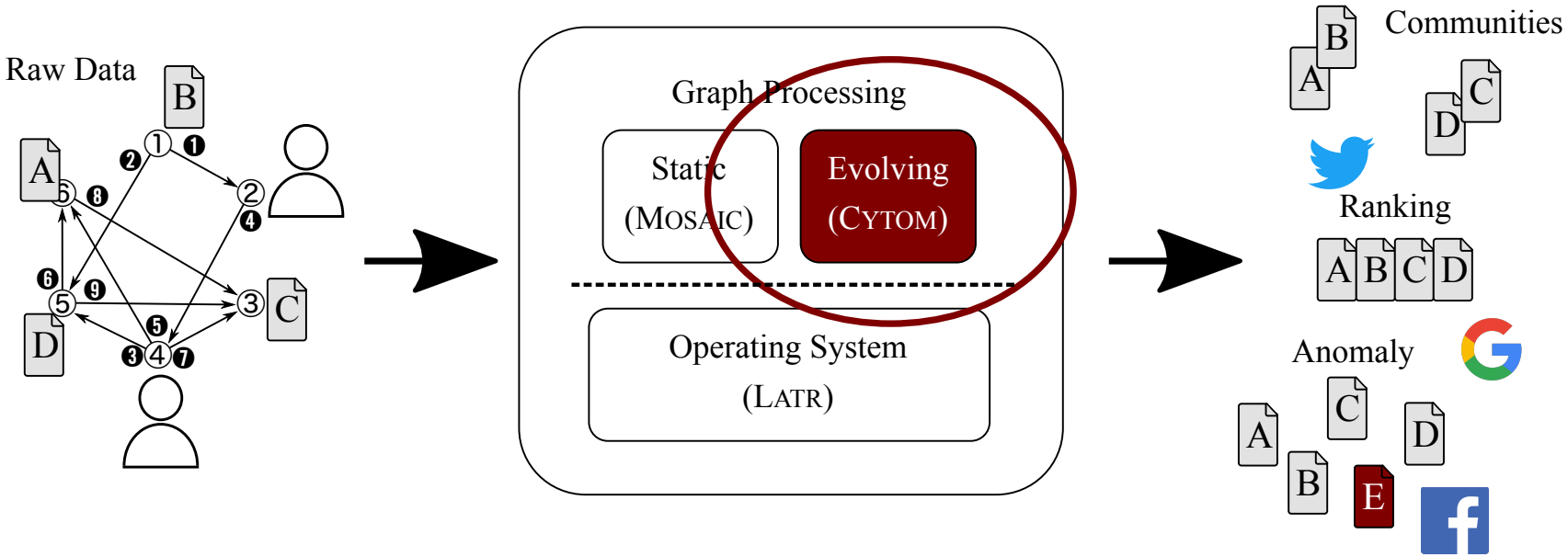
Approaches:

- **Systems-level** analysis and mechanism for improving virtual memory performance with LATR
- **Design-level** abstractions for trillion-scale graph analytics with MOSAIC
- **Design-level** abstractions and data structures for billion-scale evolving graphs with CYTOM

High-level structure

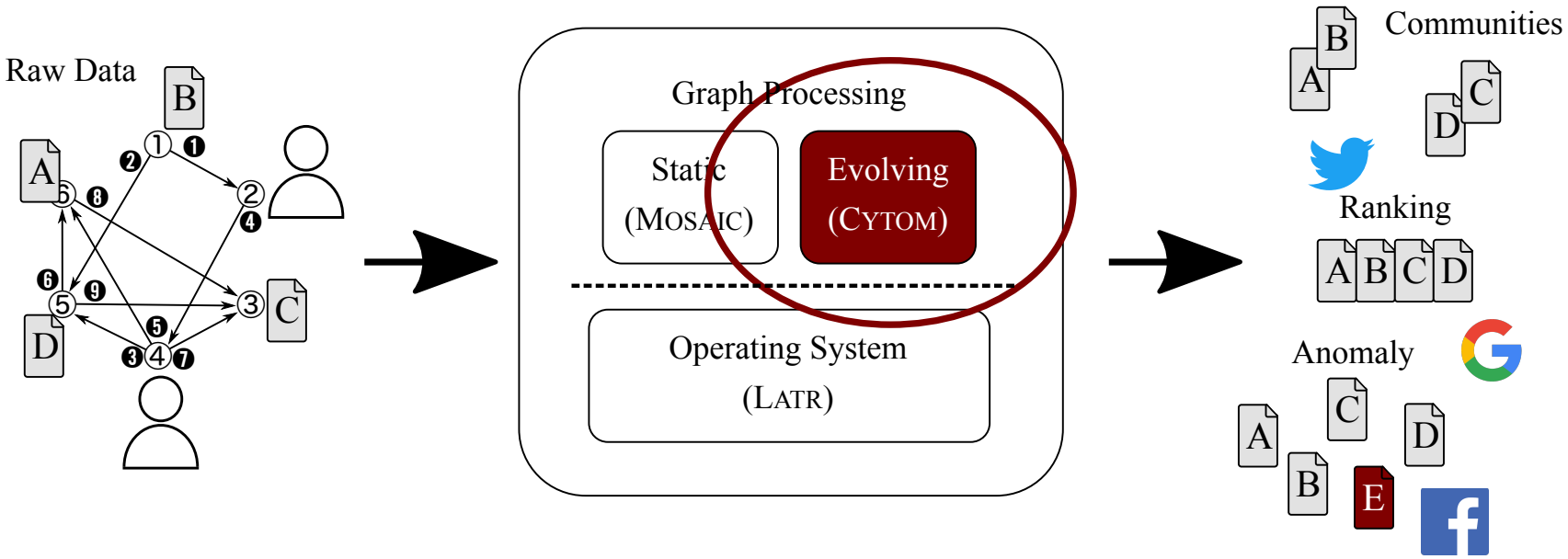


High-level structure



⇒ Focus of this talk: Processing evolving graphs with CYTOM

High-level structure



- CYTOM builds on lessons learned with MOSAIC
- However: Different challenges for setting of evolving graphs

Roadmap

1. Thesis Statement and Approaches

2. CYTOM: Processing Billion-Scale Evolving Graphs
   - Applications & Challenges
   - CYTOM's Cell-based Graph Representation
   - Handling Deletions
   - Programming Interface

3. Conclusion

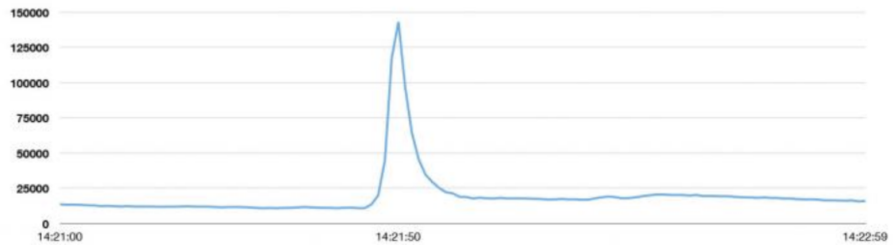
Importance of evolving graphs

New Tweets per second record, and how!

By @raffi
Friday, 16 August 2013

Recently, something remarkable happened on Twitter: On Saturday, August 3 in Japan, people watched an airing of Castle in the Sky, and at one moment they took to Twitter so much that we hit a one-second peak of 143,199 Tweets per second. (August 2 at 7:21:50 PDT; August 3 at 11:21:50 JST)

To give you some context of how that compares to typical numbers, we normally take in more than 500 million Tweets a day which means about 5,700 Tweets a second, on average. This particular spike was around 25 times greater than our steady state.



Social networks

Importance of evolving graphs

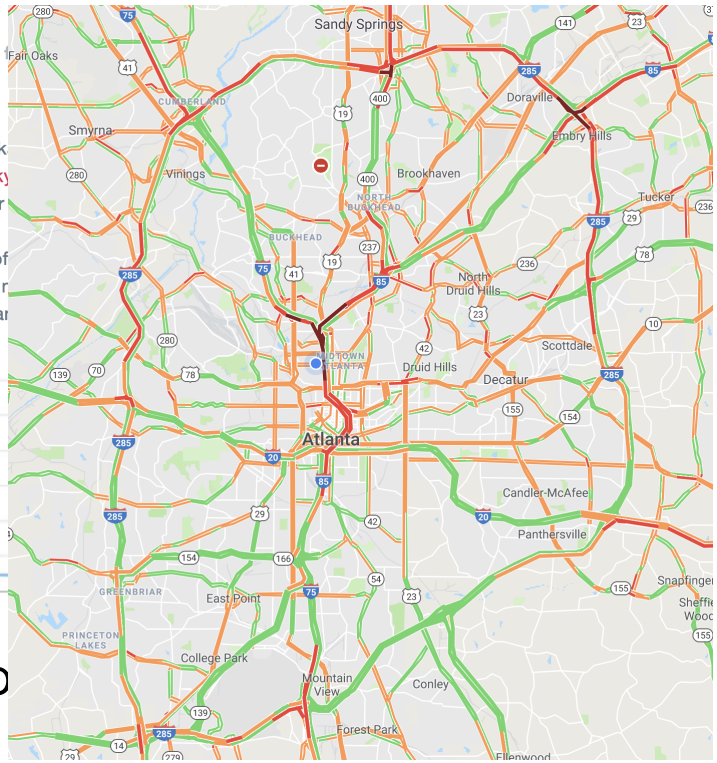
New Tweets per second record, and how!

By @raffi
Friday, 16 August 2013

Recently, something remark
an airing of Castle in the Sky
peak of 143,199 Tweets per

To give you some context of
million Tweets a day which r
around 25 times greater than

Social netwo



Live traffic & route calculation

Importance of evolving graphs

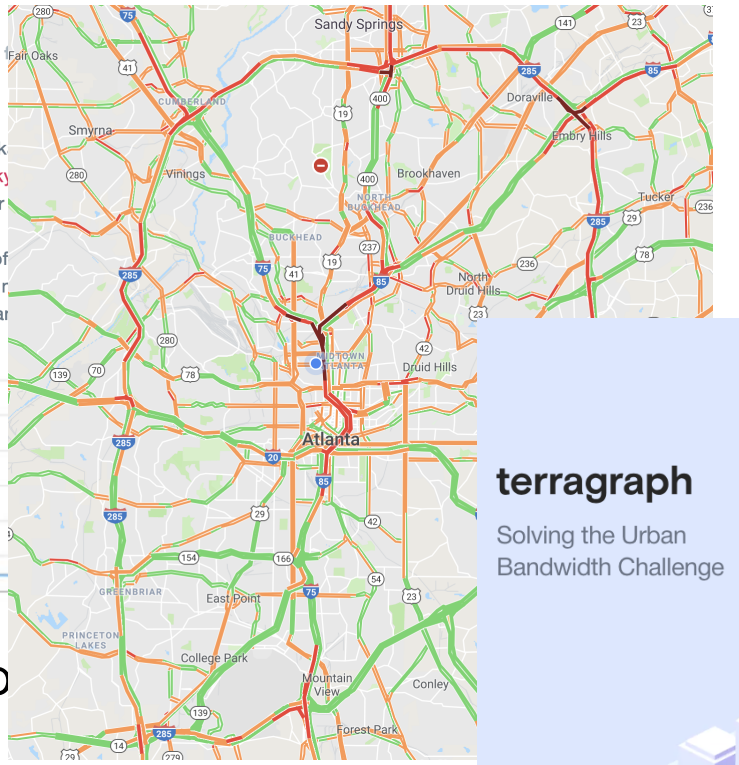
New Tweets per second record, and how!

By @raffi
Friday, 16 August 2013

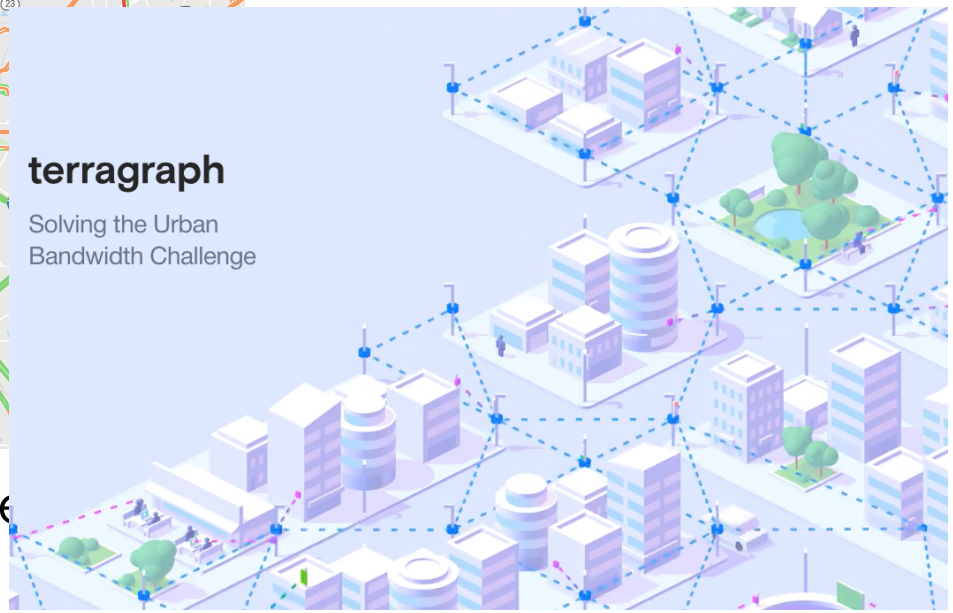
Recently, something remark
an airing of Castle in the Sky
peak of 143,199 Tweets per

To give you some context of
million Tweets a day which r
around 25 times greater than

Social netwo



Live traffic & route



Analysis of next-gen Cellular Data, Connected Cars, ...

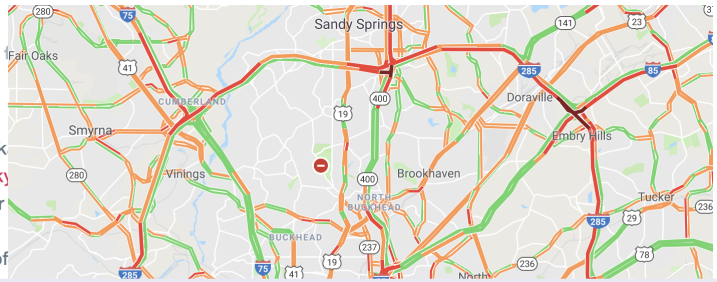
Importance of evolving graphs

New Tweets per second record, and how!

By @raffi
Friday, 16 August 2013

Recently, something remark
an airing of Castle in the Sky
peak of 143,199 Tweets per

To give you some context of



⇒ Applications enable by processing evolving graphs

- Online route calculation
- Online Anomaly detection
- Online Ranking
- Online Community detection

Live traffic & route

Analysis of next-gen Cellular Data, Connected Cars, . . .

Challenges of billion-scale evolving graphs

- Constant stream of updates
- Optimizing graph storage format not possible on every update
  - Graph data structures supporting updates
- Co-running graph updates with algorithm
- Special challenges for single machines (compared to distributed systems):
  - Limited memory & compute
  - Fault tolerance

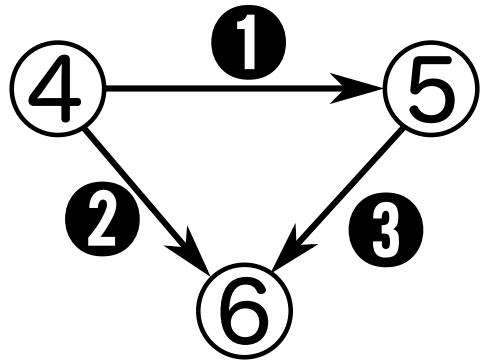
Background: Graph data structures for evolving graphs

- Goal: Efficient update support and computational efficiency
- Four options:
  - Adjacency matrix
  - Adjacency lists
  - Compressed sparse rows (CSR)
  - Edge lists

Background: Graph data structures for evolving graphs

- Goal: Efficient update support and computational efficiency
- Four options:
  - Adjacency matrix
  - Adjacency lists
  - Compressed sparse rows (CSR)
  - Edge lists

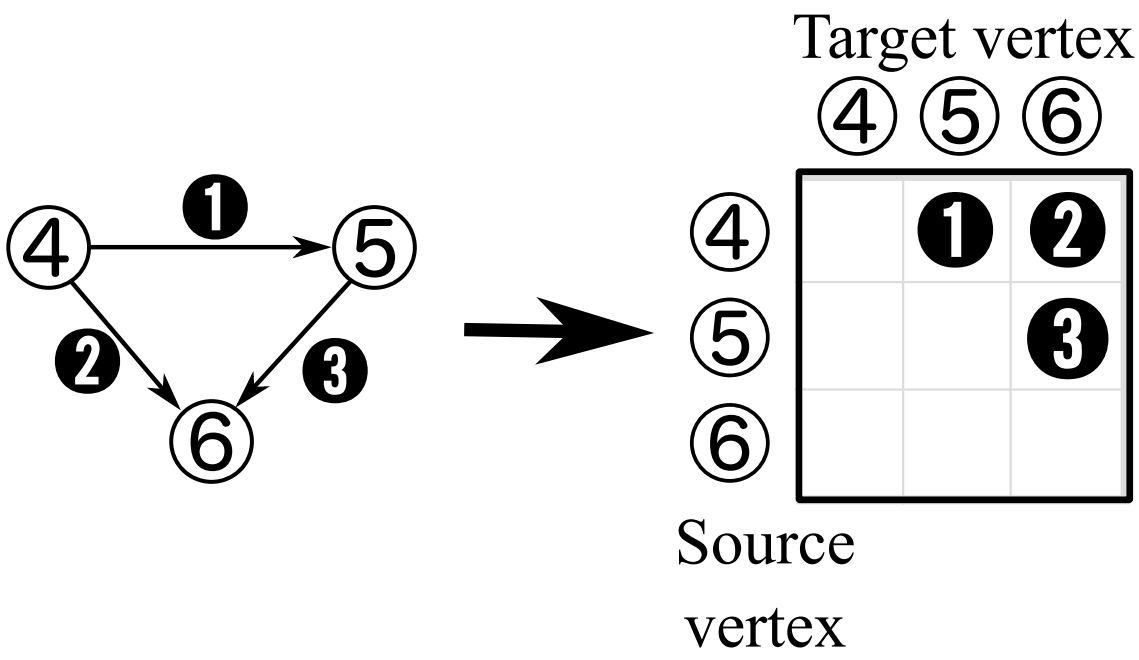
Sample graph:



Background: Adjacency matrix

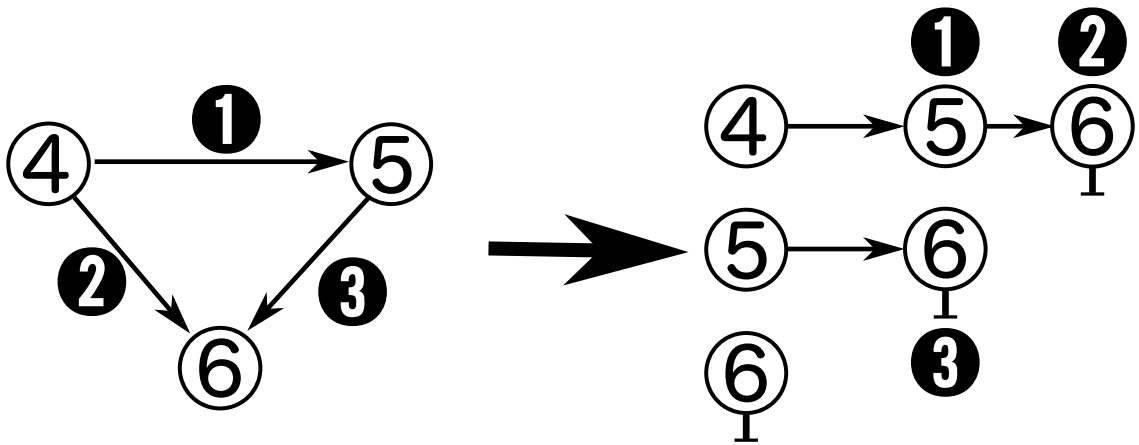
- Minimal storage per edge (1 bit)
- Great for dense graphs
- Limitation: Most real-world graphs are sparse

Example:



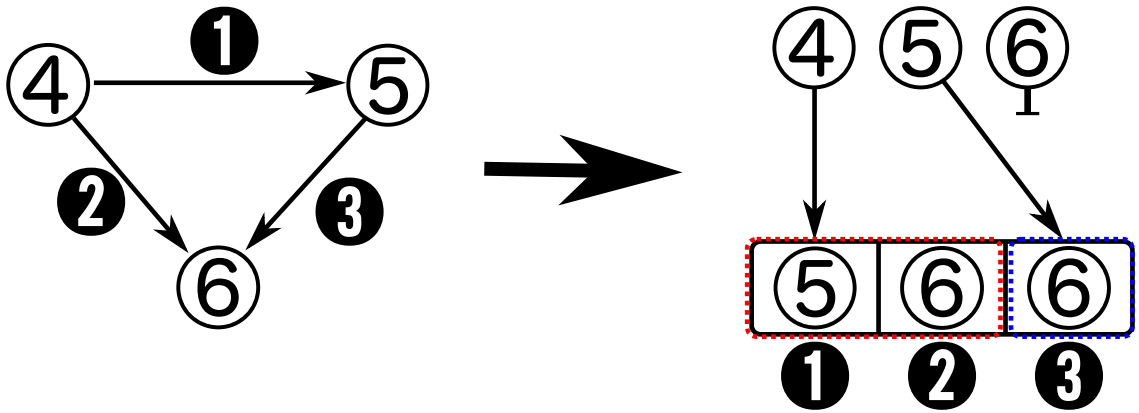
Background: Adjacency lists

- For best case, storage cost per edge: 8 bytes
- Relatively simple construction
- Limitation: Overhead on traversal and allocation of new edges
- Used by Stinger



Background: Compressed sparse rows (CSR)

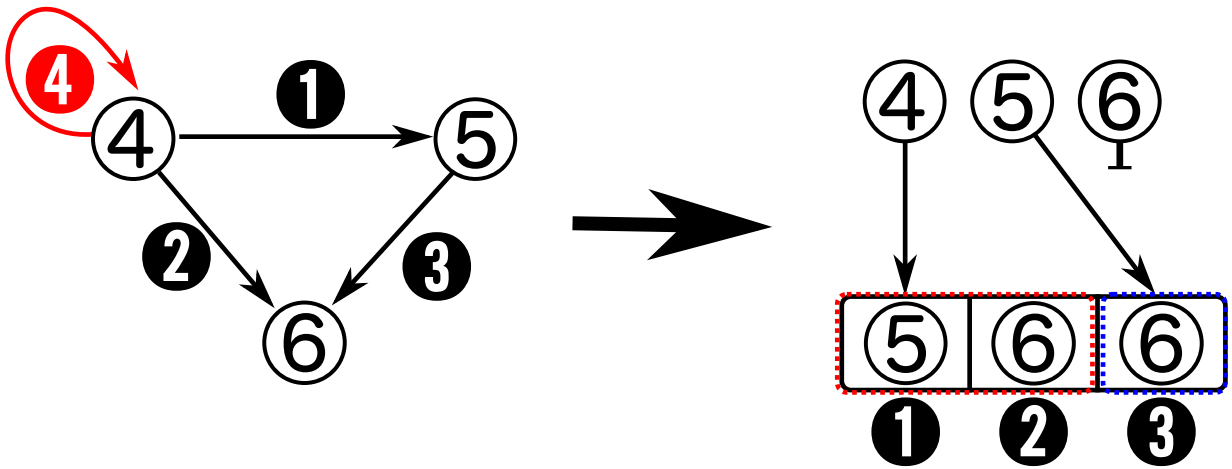
- For best case, storage cost per edge: 8 bytes
- Efficient access and lower storage overhead
- Limitation: Changes require rebuilding or complicated overflow areas
- Used by Llama



Background: Compressed sparse rows (CSR)

- For best case, storage cost per edge: 8 bytes
- Efficient access and lower storage overhead
- Limitation: Changes require rebuilding or complicated overflow areas
- Used by Llama

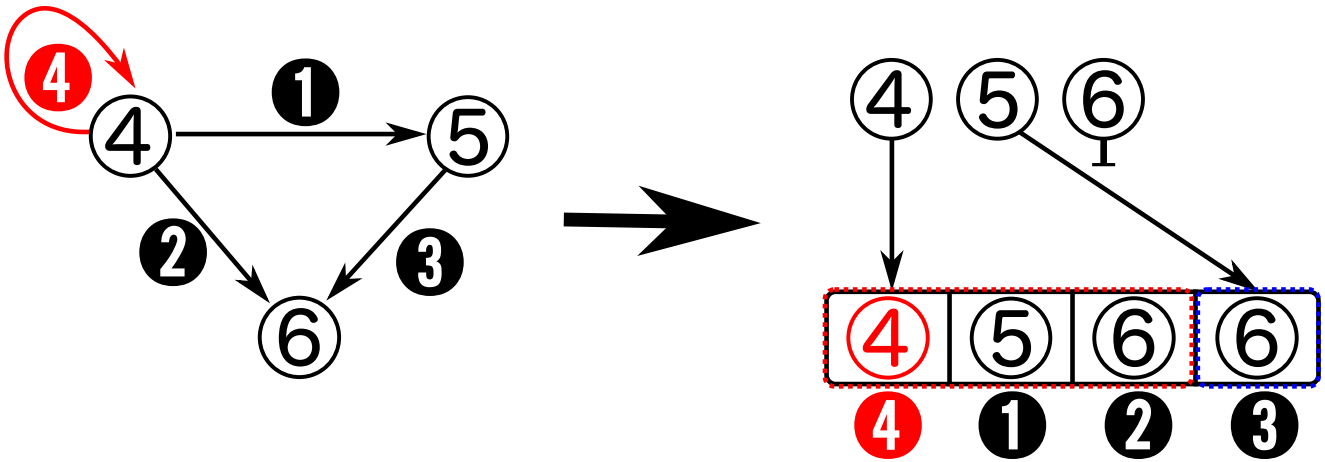
Example: Add edge ❹



Background: Compressed sparse rows (CSR)

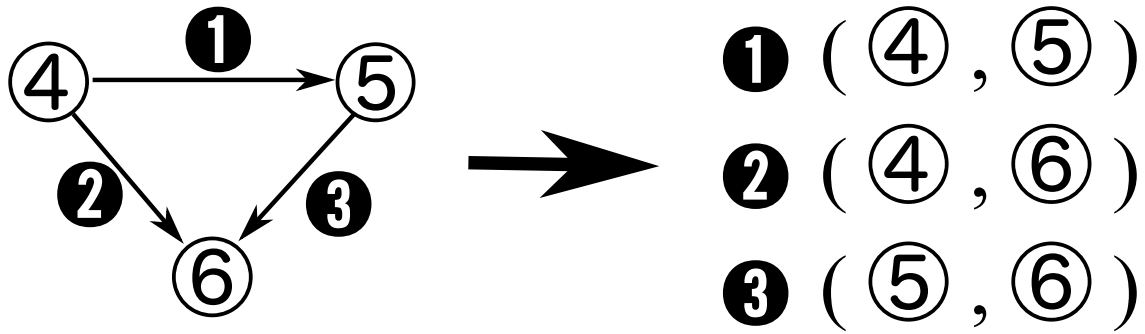
- For best case, storage cost per edge: 8 bytes
- Efficient access and lower storage overhead
- Limitation: Changes require rebuilding or complicated overflow areas
- Used by Llama

Problem: Array has to be shifted



Background: Edge lists

- Simple construction
- Adding new edges possible
- Limitation: Overhead for storage cost, 16 bytes per edge
- Used by GraphIn, GraphOne



Data structures summary

- Comparison of presented data structures

Representation	**Sparse graphs**	**Updates**	**Traversals**	**Storage**
Adj. matrix	-	✓	✓	1 bit
Adj. lists	✓	✓	-	8 bytes
CSR	✓	-	✓	8 bytes
Edge list	✓	✓	✓	16 bytes

Data structures summary

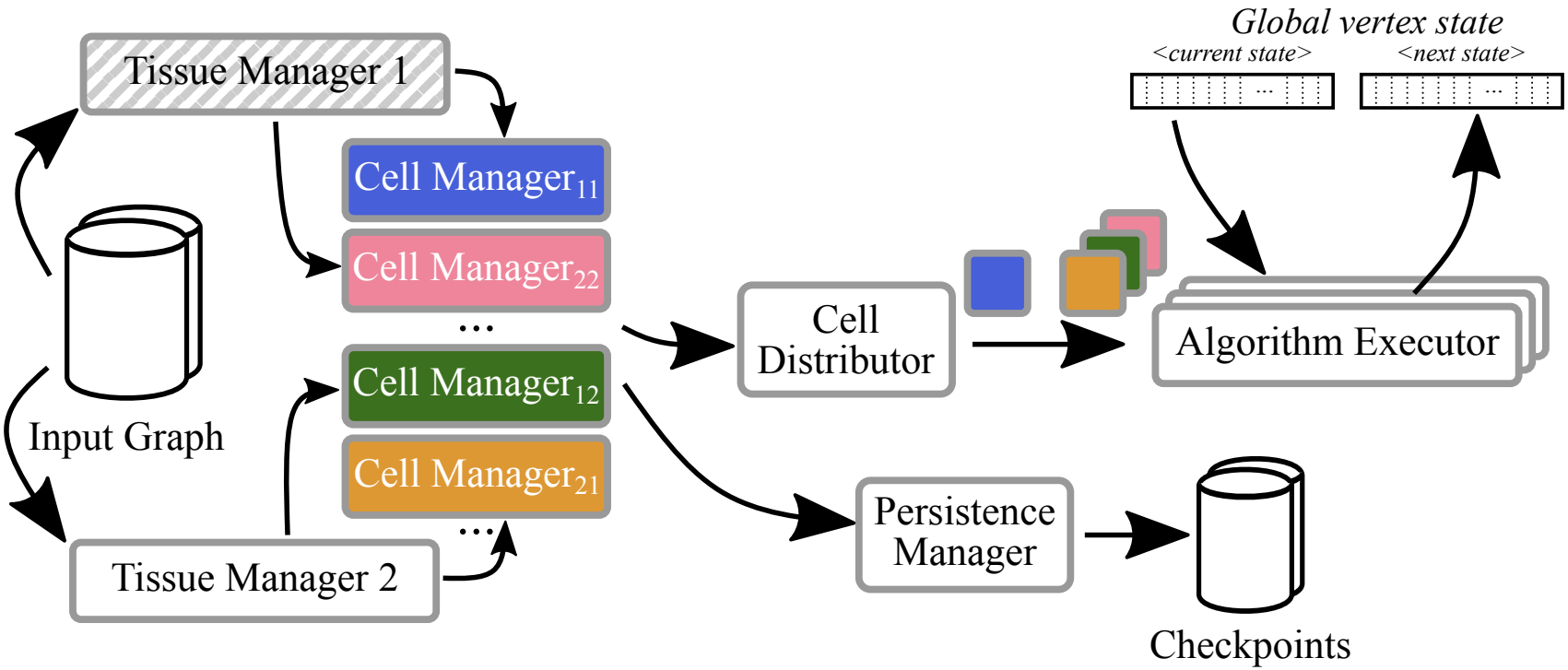
- Comparison of presented data structures

Representation	**Sparse graphs**	**Updates**	**Traversals**	**Storage**
Adj. matrix	-	✓	✓	1 bit
Adj. lists	✓	✓	-	8 bytes
CSR	✓	-	✓	8 bytes
Edge list	✓	✓	✓	16 bytes
Edge list$_{\text{CYTOM}}$	✓	✓	✓	4 bytes

- Our choice for evolving graphs: *edge lists*
- However: Mitigation for storage overhead needed

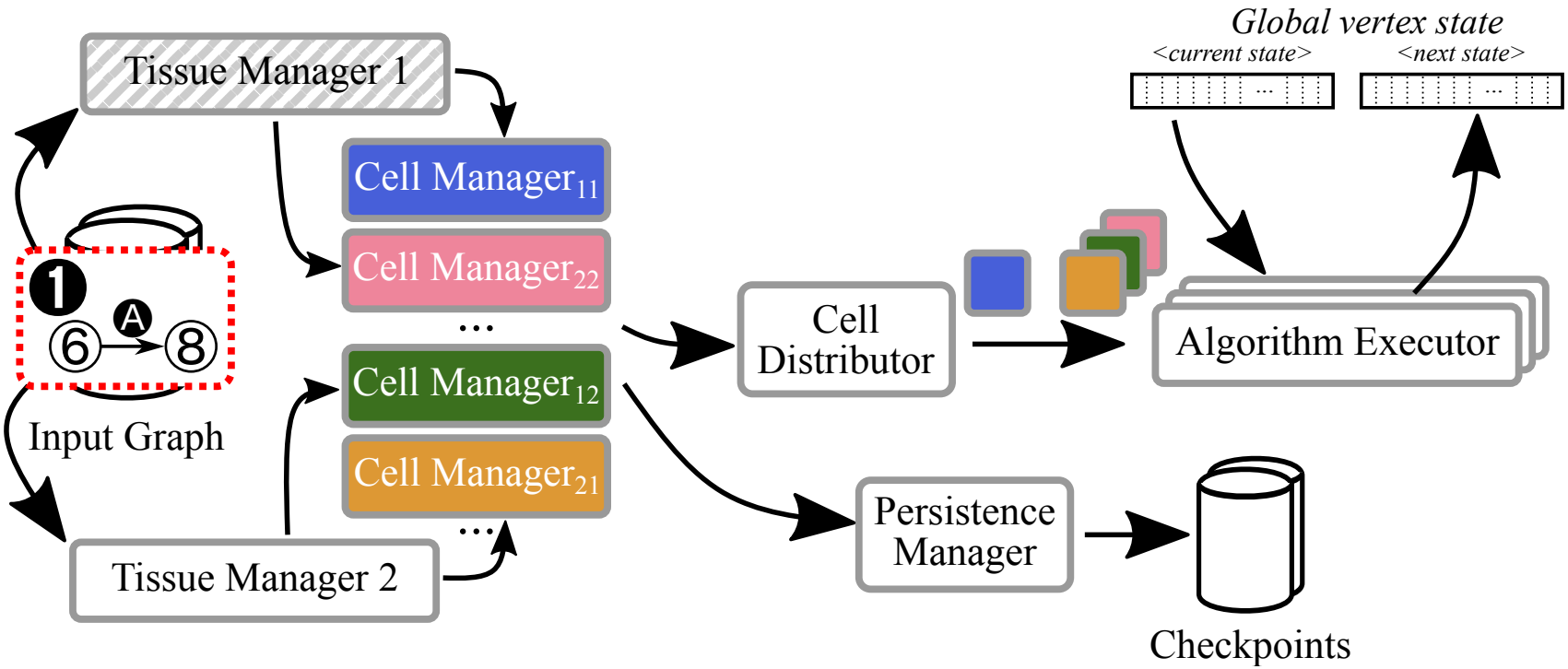
High-level architecture

From raw input graph to algorithmic output:



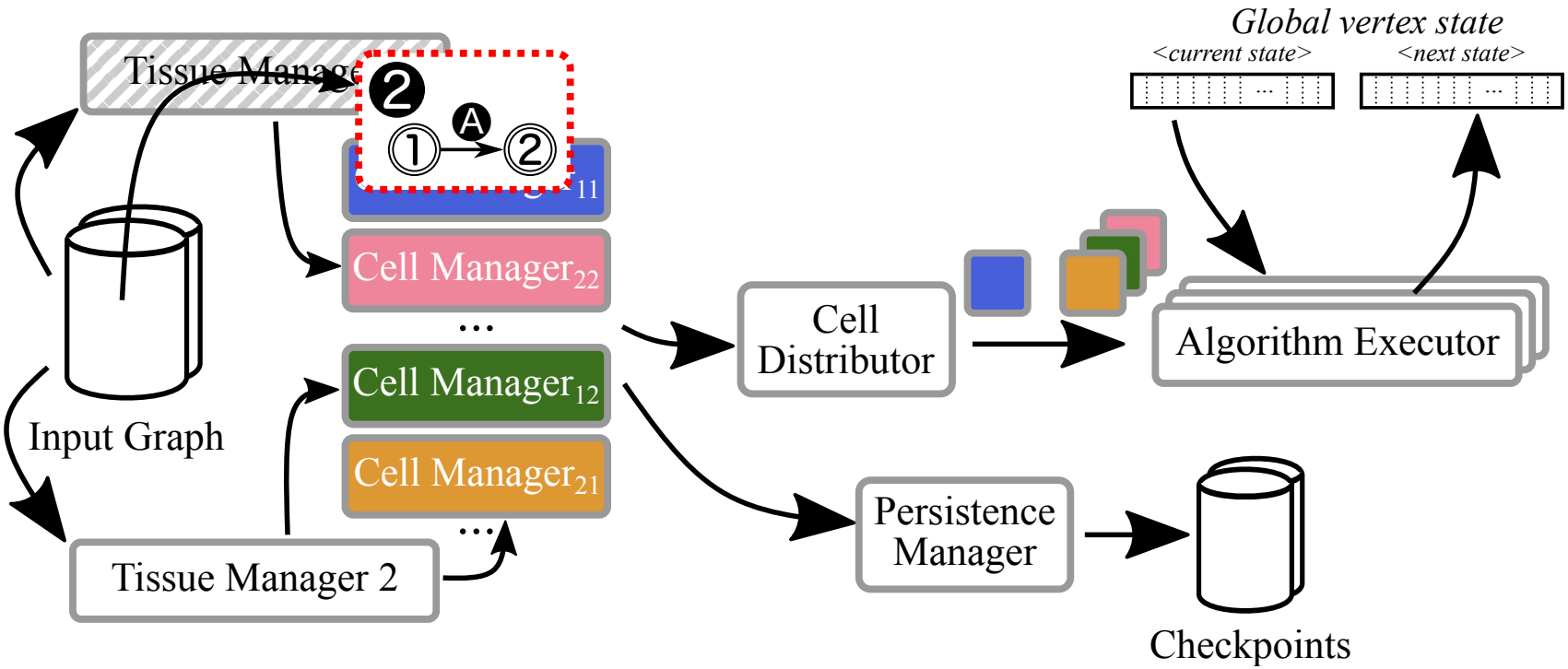
High-level architecture

Step ❶ : Update to input graph



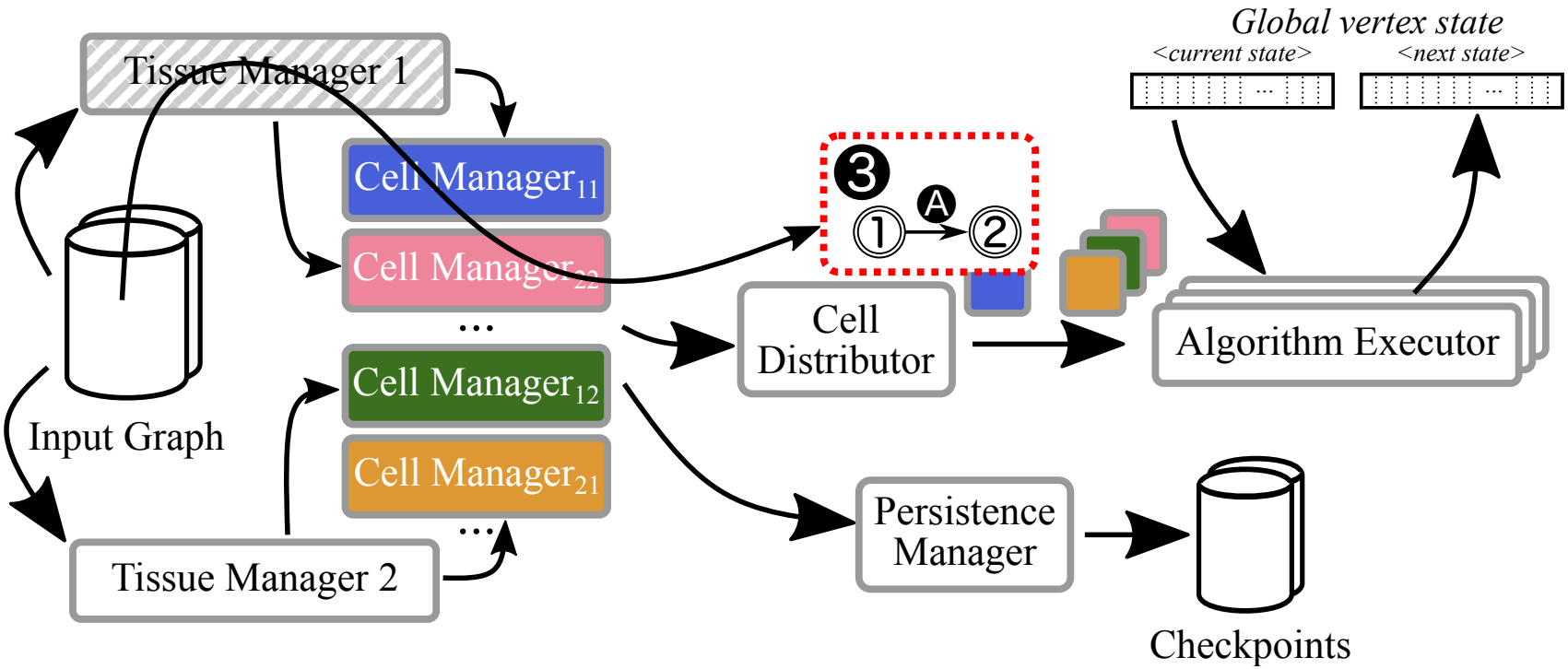
High-level architecture

Step ❷ : Update CYTOM's internal structure



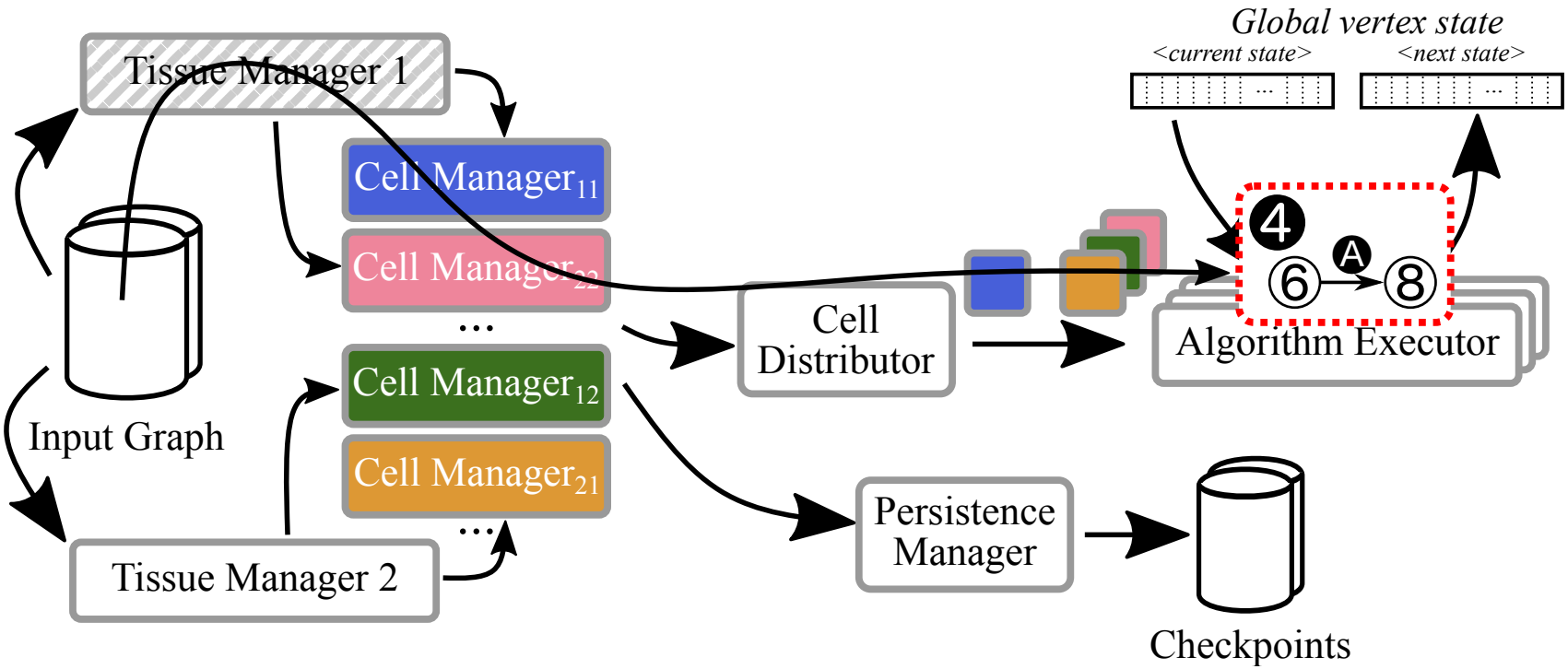
High-level architecture

Step ❸ : Distribute update

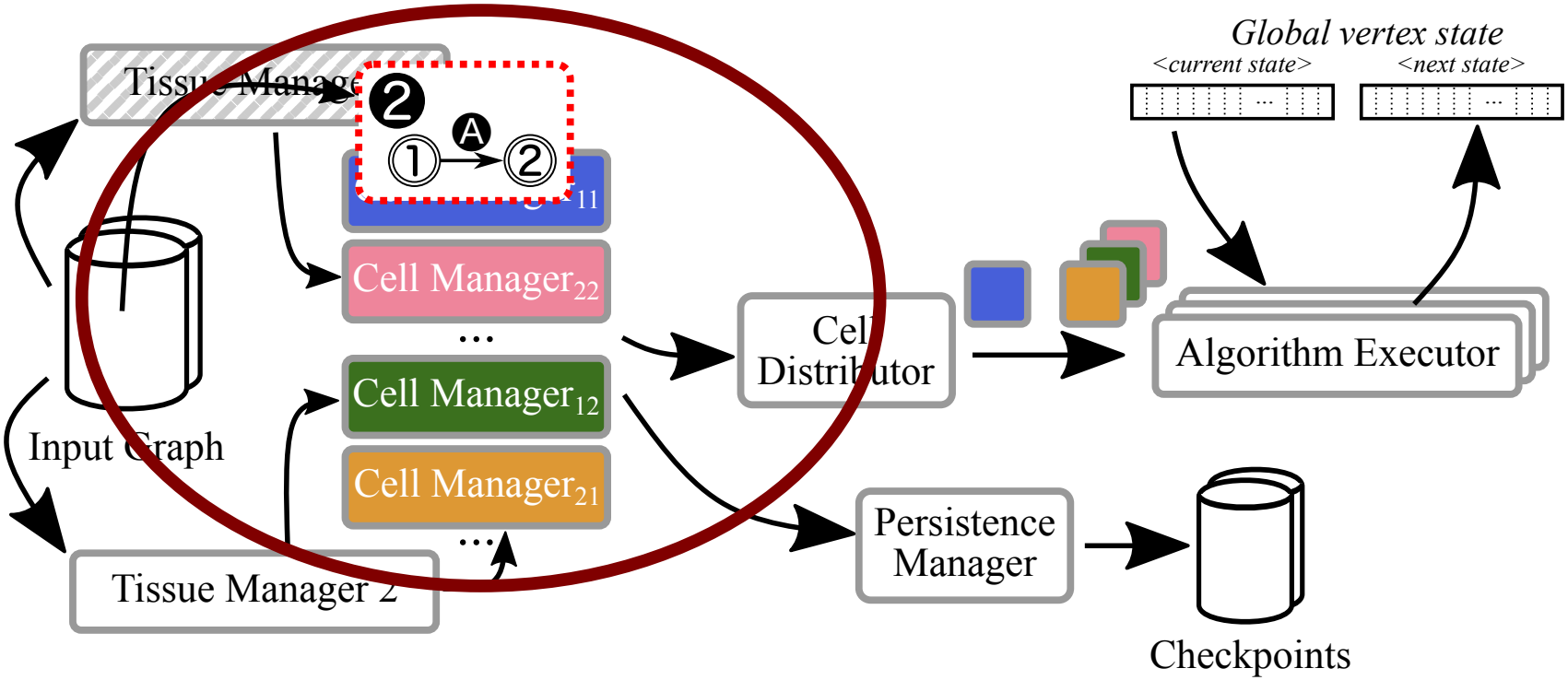


High-level architecture

Step ❹ : Algorithmic processing of update



High-level architecture: Cells

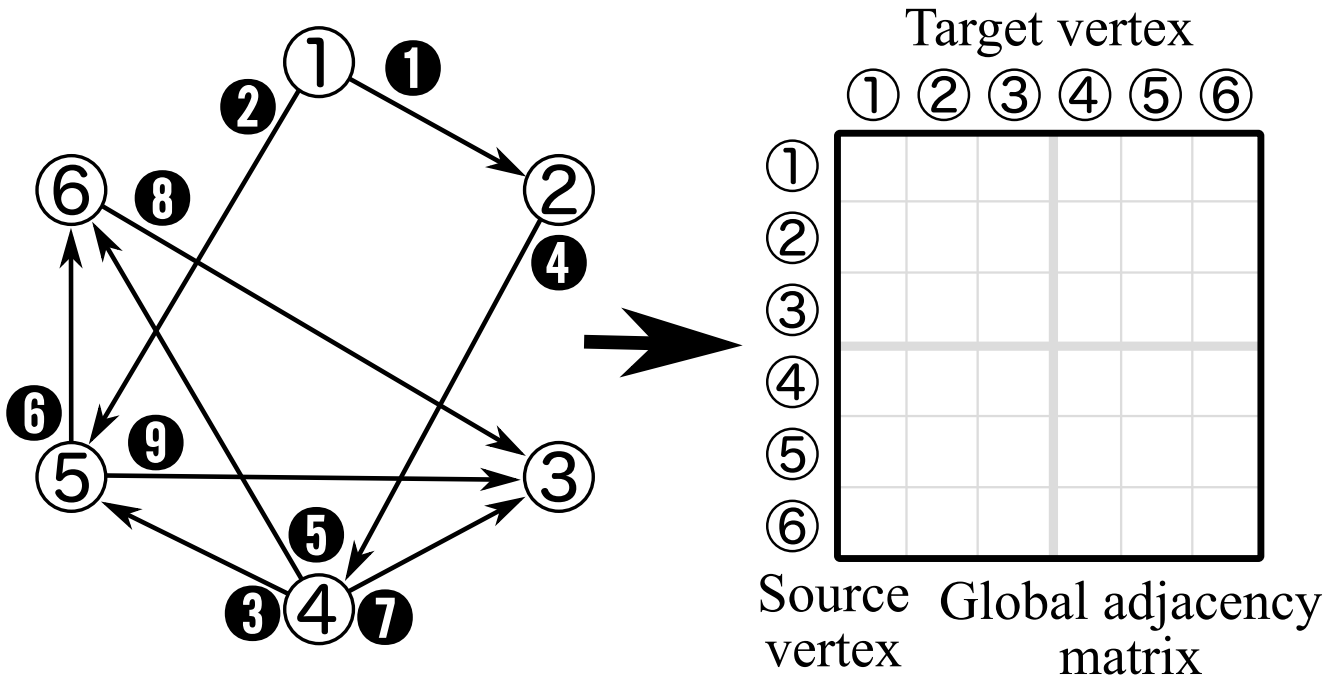


⇒ First focus: CYTOM's cells

Construction of CYTOM's cells

- Lightweight partitioning

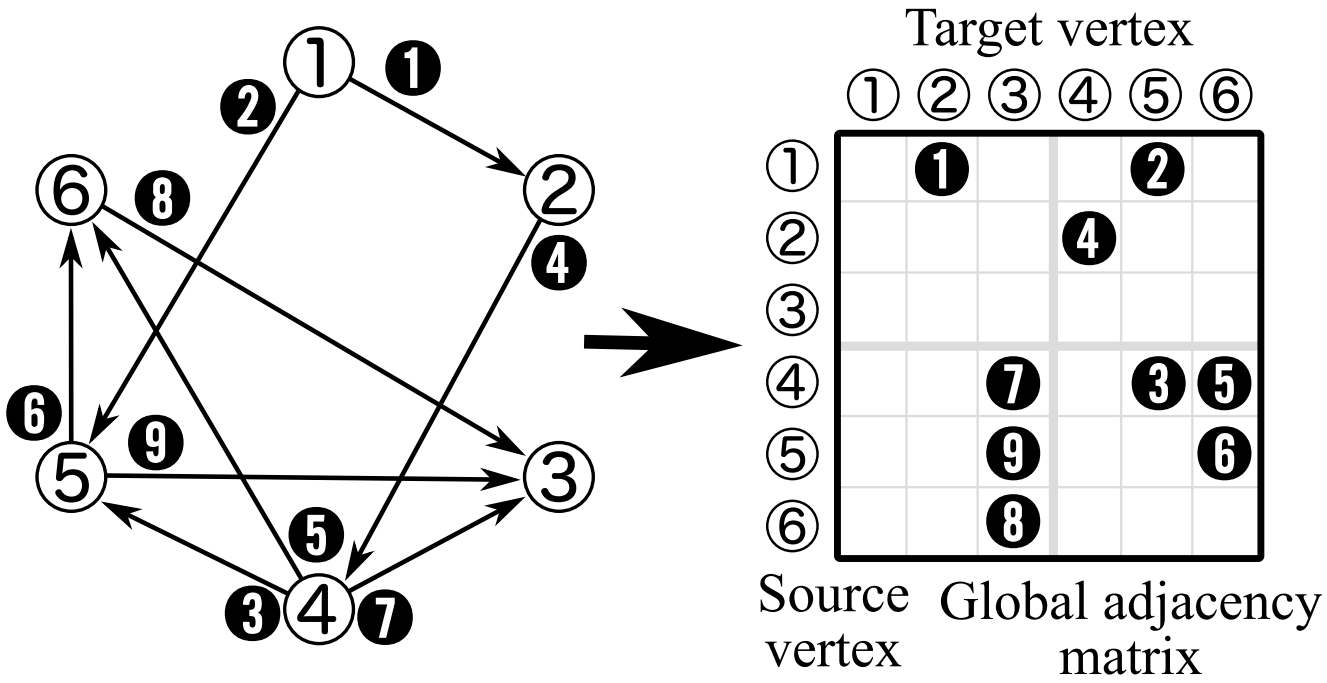
Step 1: Input graph:



Construction of CYTOM's cells

- Lightweight partitioning

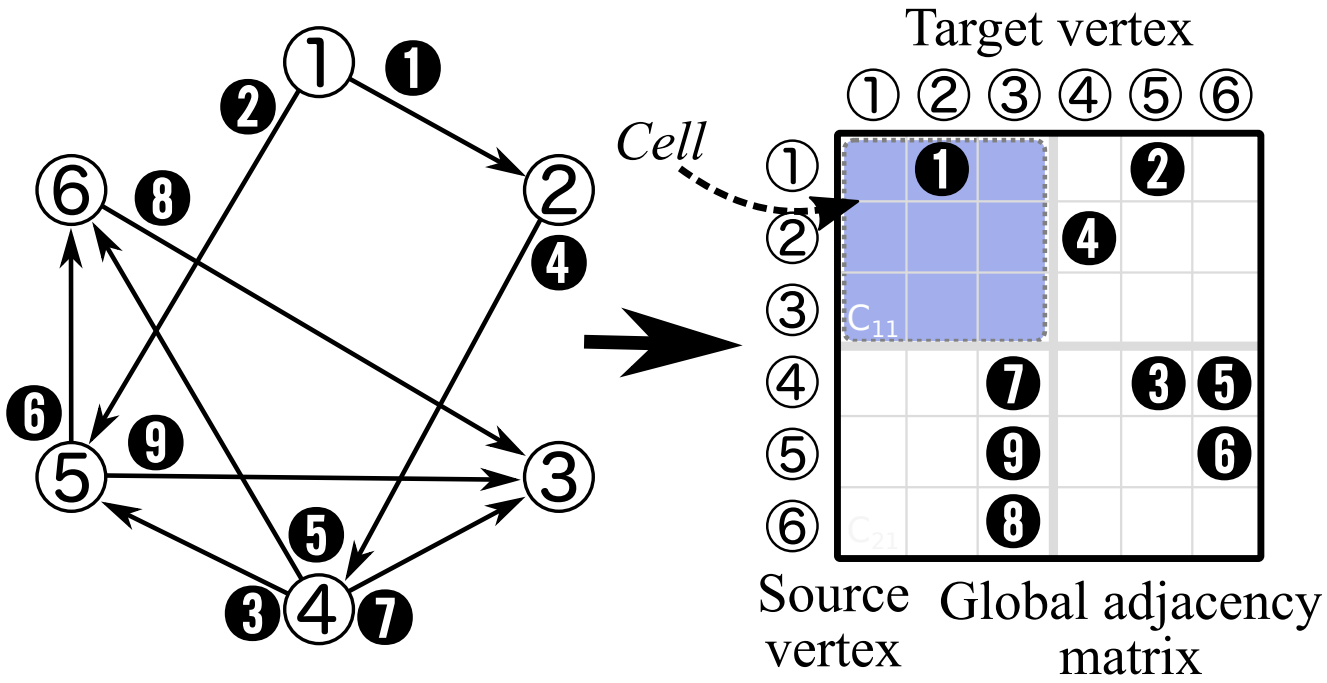
Step 2: Adjacency Matrix



Construction of CYTOM's cells

- Lightweight partitioning

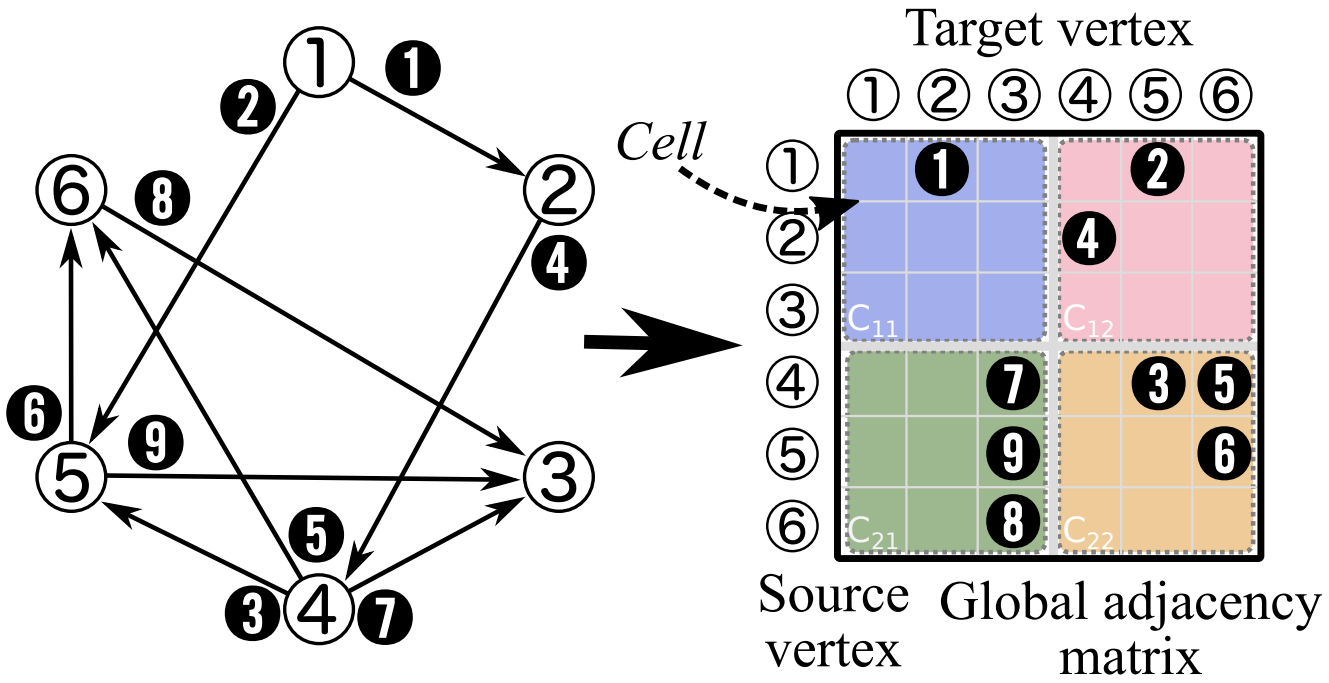
Step 3: Construction of cell C_{11}



Construction of CYTOM's cells

- Lightweight partitioning

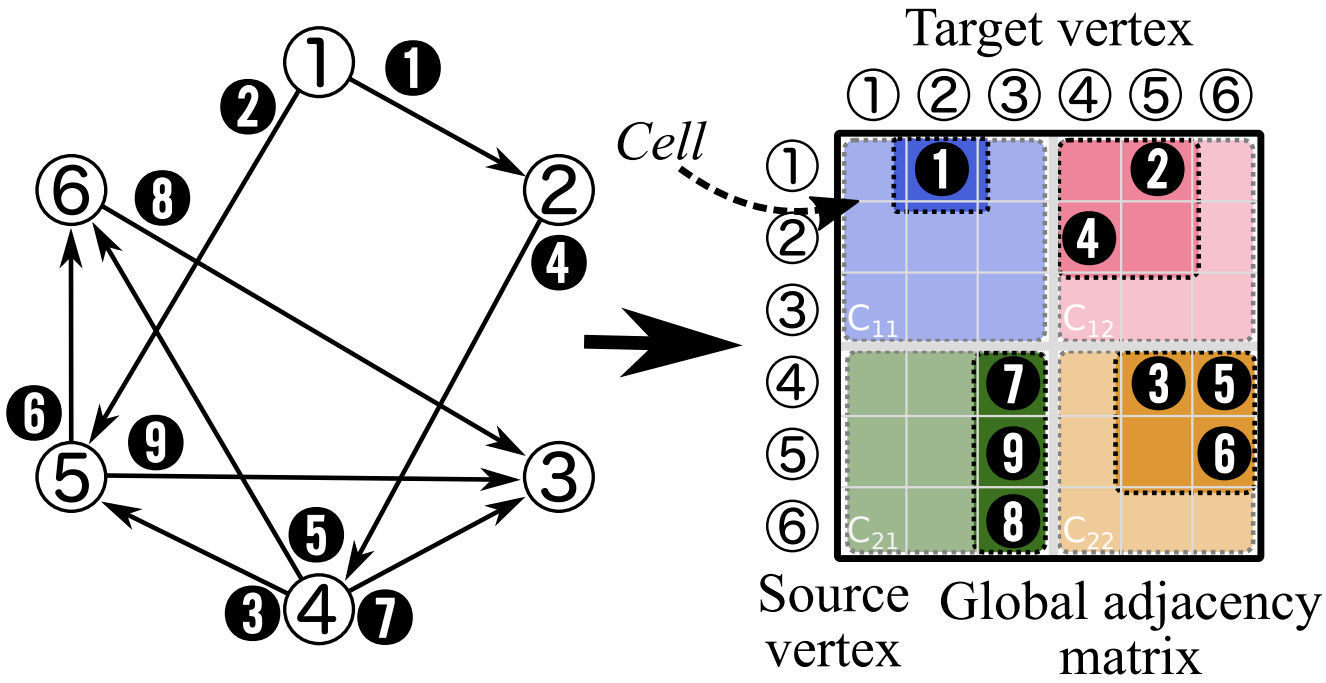
Step 4: Construction of all other cells



Construction of CYTOM's cells

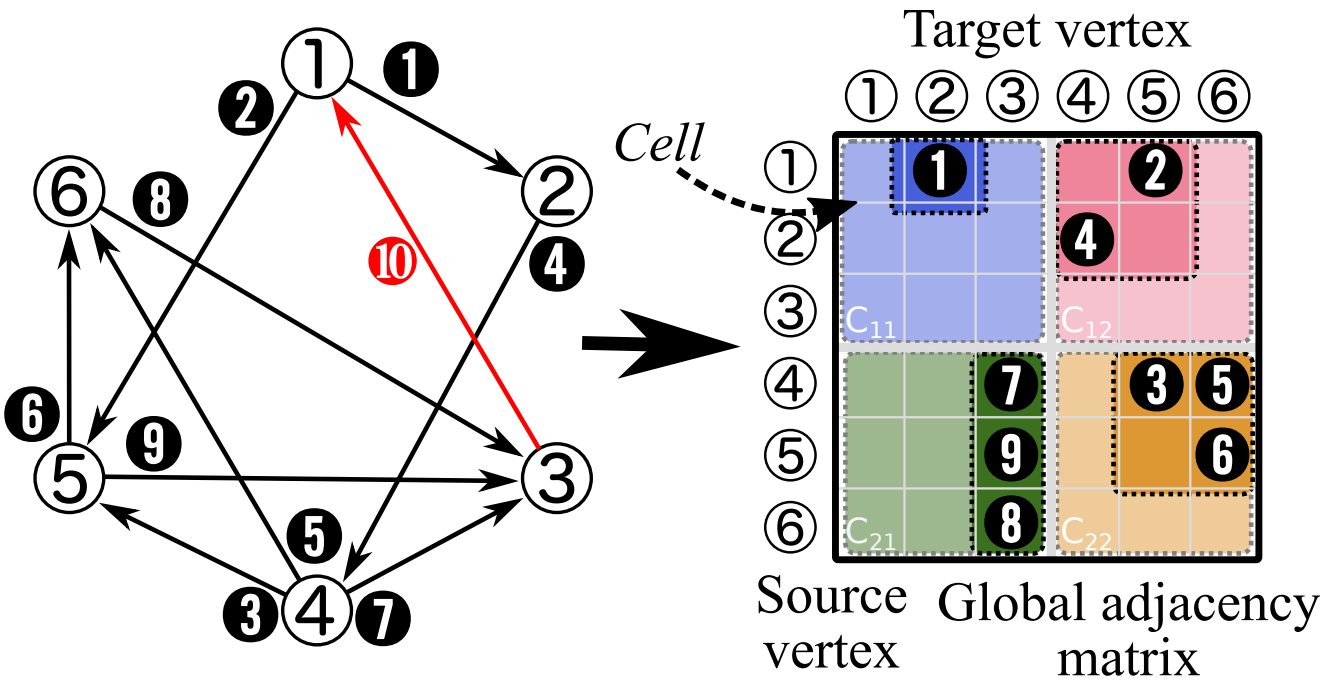
- Lightweight partitioning

Step 5: Overlay current cell size



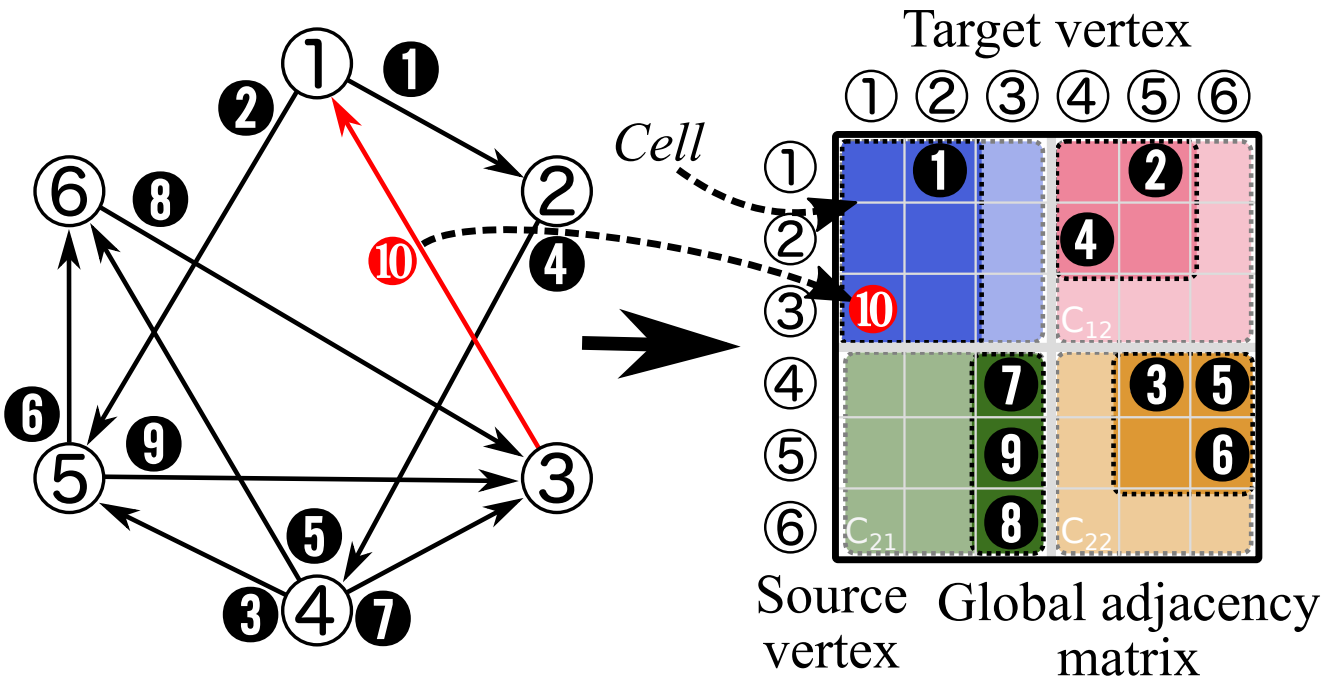
Construction of CYTOM's cells: Adding new edges

- Adding new edges lightweight
- Division to obtain partition



Construction of CYTOM's cells: Adding new edges

- Adding new edges lightweight
- Division to obtain partition



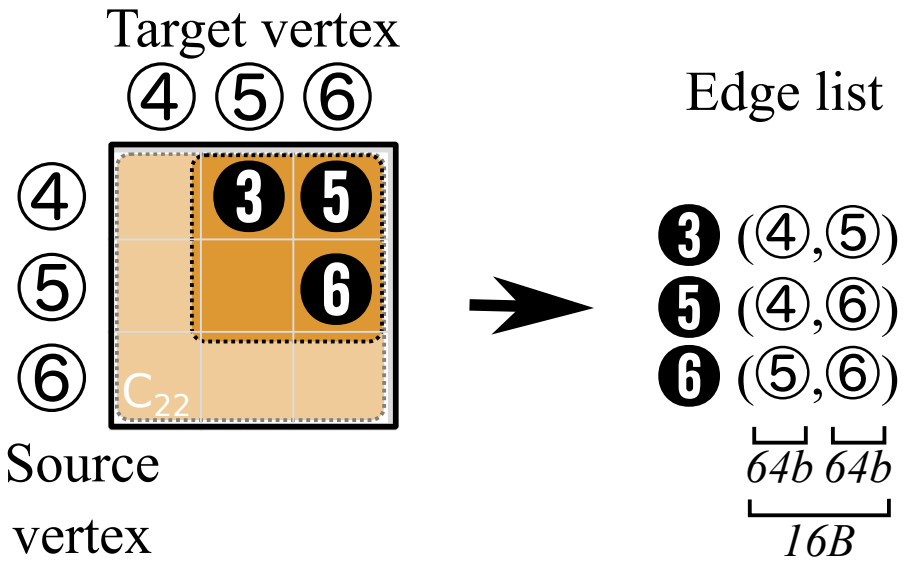
Cytom's cells

- Subgraph-centric
- Difference to *tiles* (Mosaic) or *partitions* (GridGraph)?
  - Both designed for static execution
  - Immutable structures
  - Changes only possible coupled with preprocessing step
  - Cytom's cells can grow (and shrink) as required without preprocessing
- Enables short identifiers (16 bits)
  - 2× to 4× less memory (32 or 64 bit identifiers)
  - Compared to Mosaic: No metadata needed

Benefit of CYTOM's cells

- Enables short identifiers (16 bits)
  - 32 bits (4 bytes) per edge
  - 2× to 4× less memory (32 or 64 bit identifiers)
  - No metadata needed
- Usage of edge lists
  - Insert: Append at end of vector

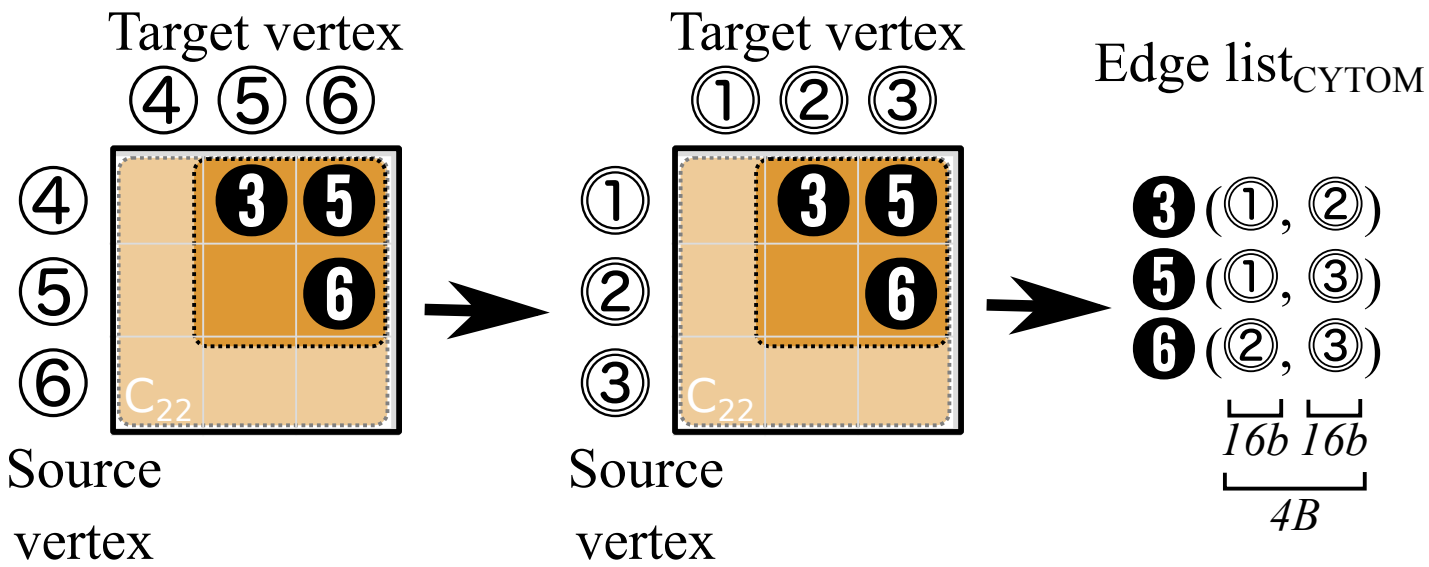
Base case with global edge lists



Benefit of CYTOM's cells

- Enables short identifiers (16 bits)
  - 32 bits (4 bytes) per edge
  - 2× to 4× less memory (32 or 64 bit identifiers)
  - No metadata needed
- Usage of edge lists
  - Insert: Append at end of vector

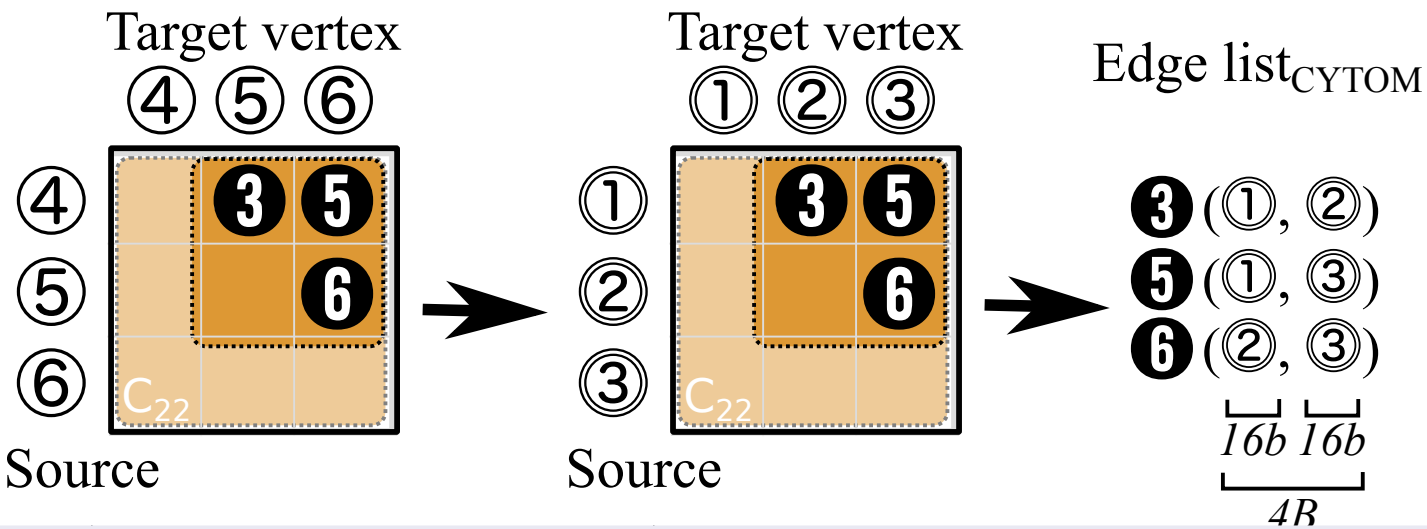
Idea: Translate global IDs into local ones, store in CYTOM's edge list



Benefit of CYTOM's cells

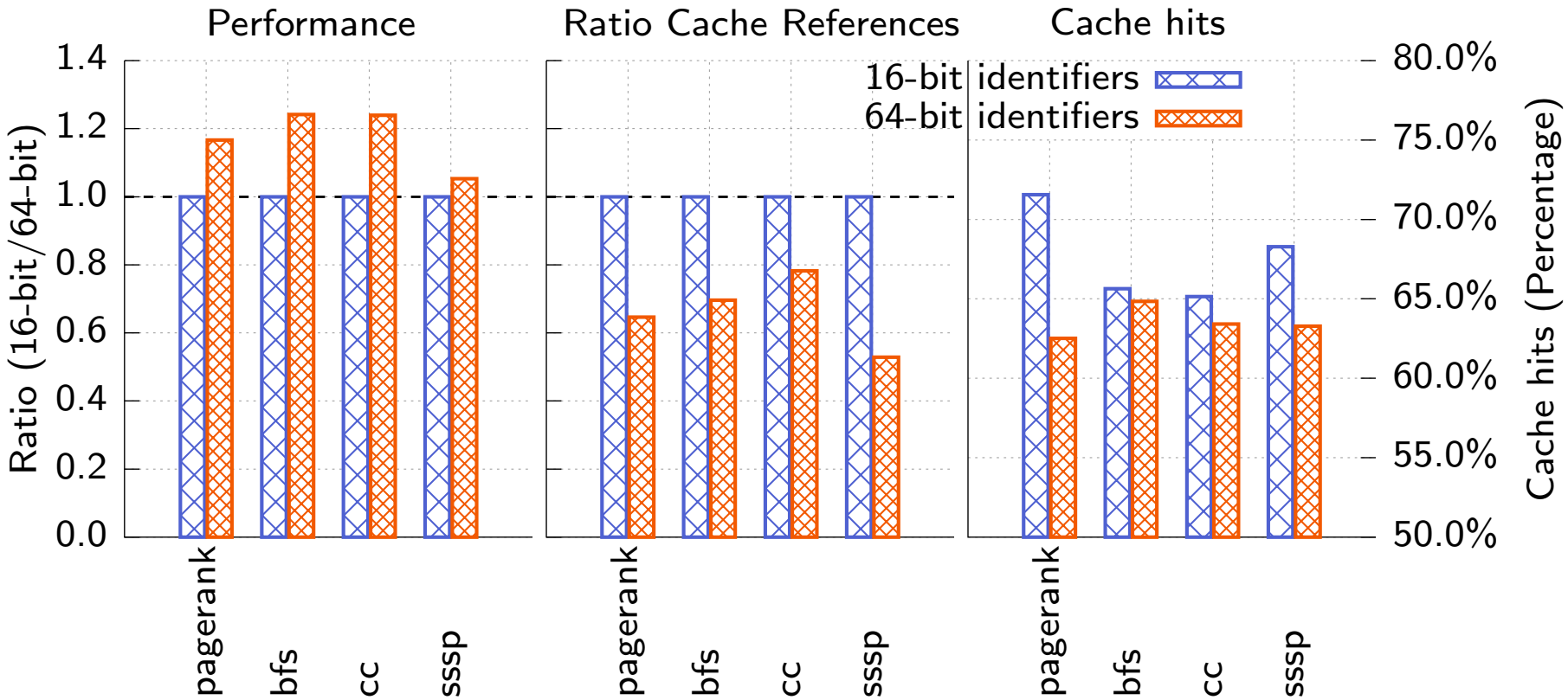
- Enables short identifiers (16 bits)
  - 32 bits (4 bytes) per edge
  - 2× to 4× less memory (32 or 64 bit identifiers)
  - No metadata needed
- Usage of edge lists
  - Insert: Append at end of vector

Idea: Translate global IDs into local ones, store in CYTOM's edge list



⇒ Impact of shortened identifiers?

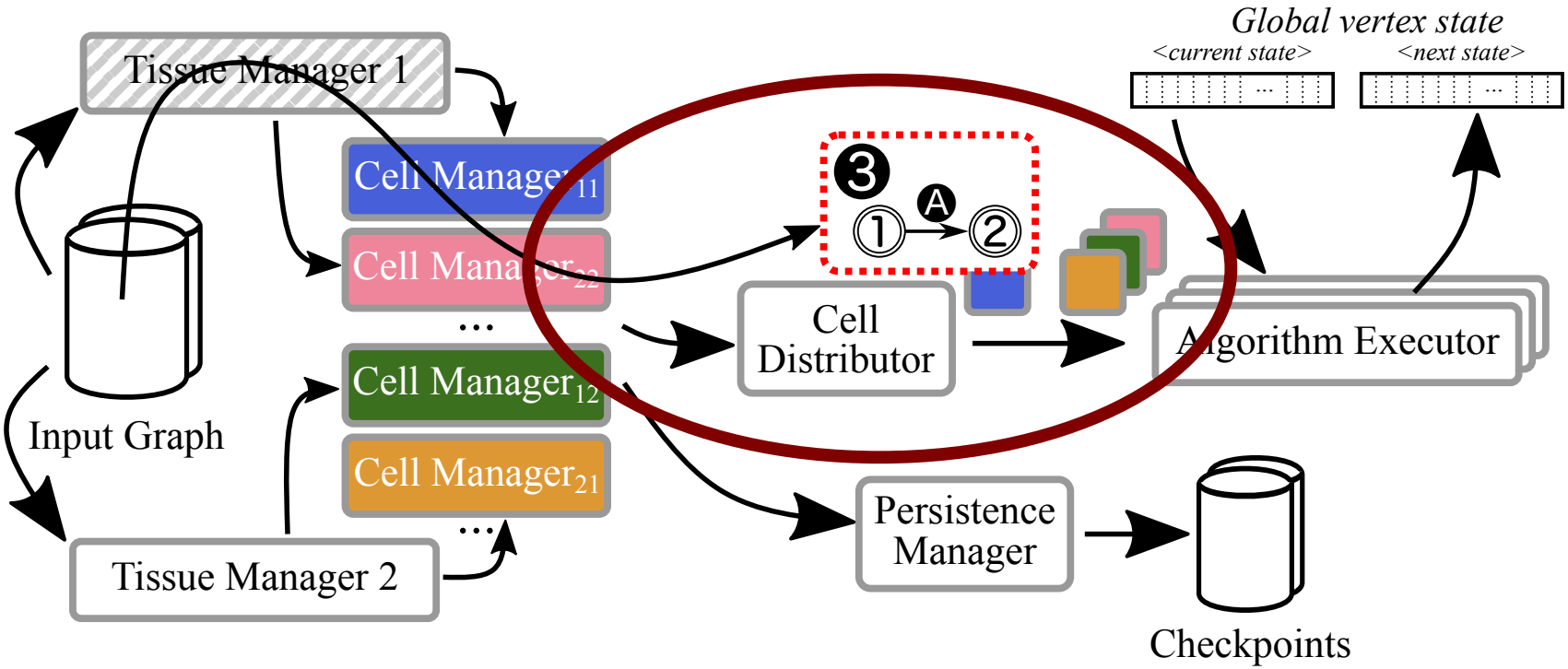
Impact of short identifiers - lower memory footprint



⇒ Up to 24% higher overall performance, 47% fewer cache references

High-level architecture: Cell distribution

How to operate on active cells?

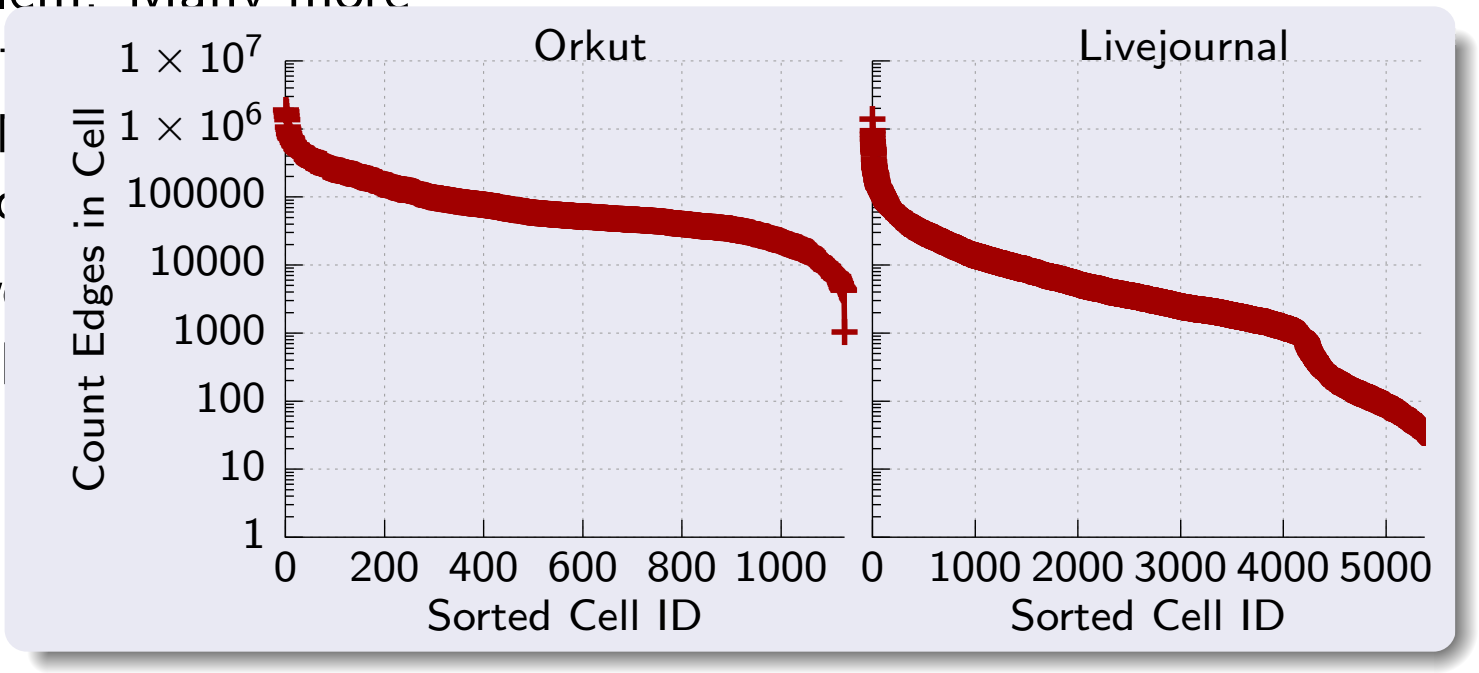


How to load balance cells?

- Problem: Many more cells than threads
  - Millions of cells, dozens of threads
- Skewed cell size distribution

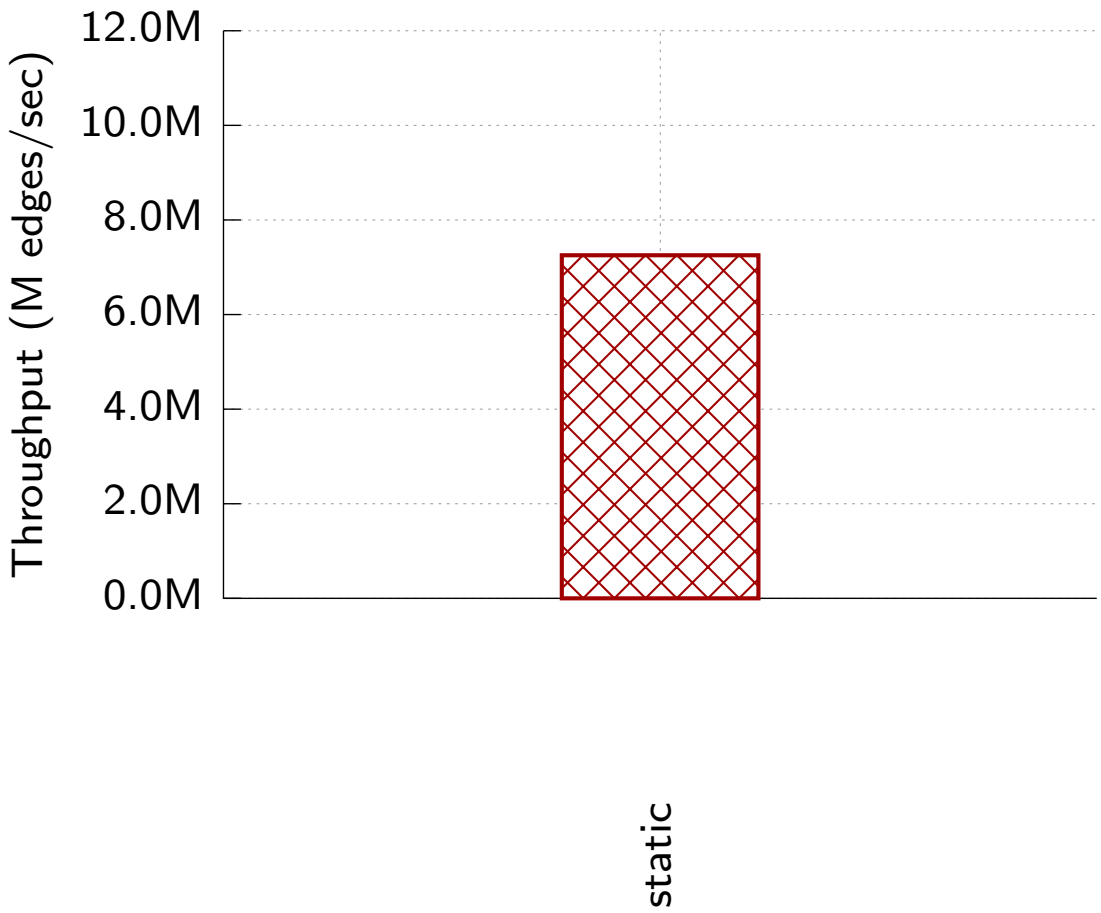
How to load balance cells?

- Problem: Many more cells
  - I
    c
- Skew
  distri



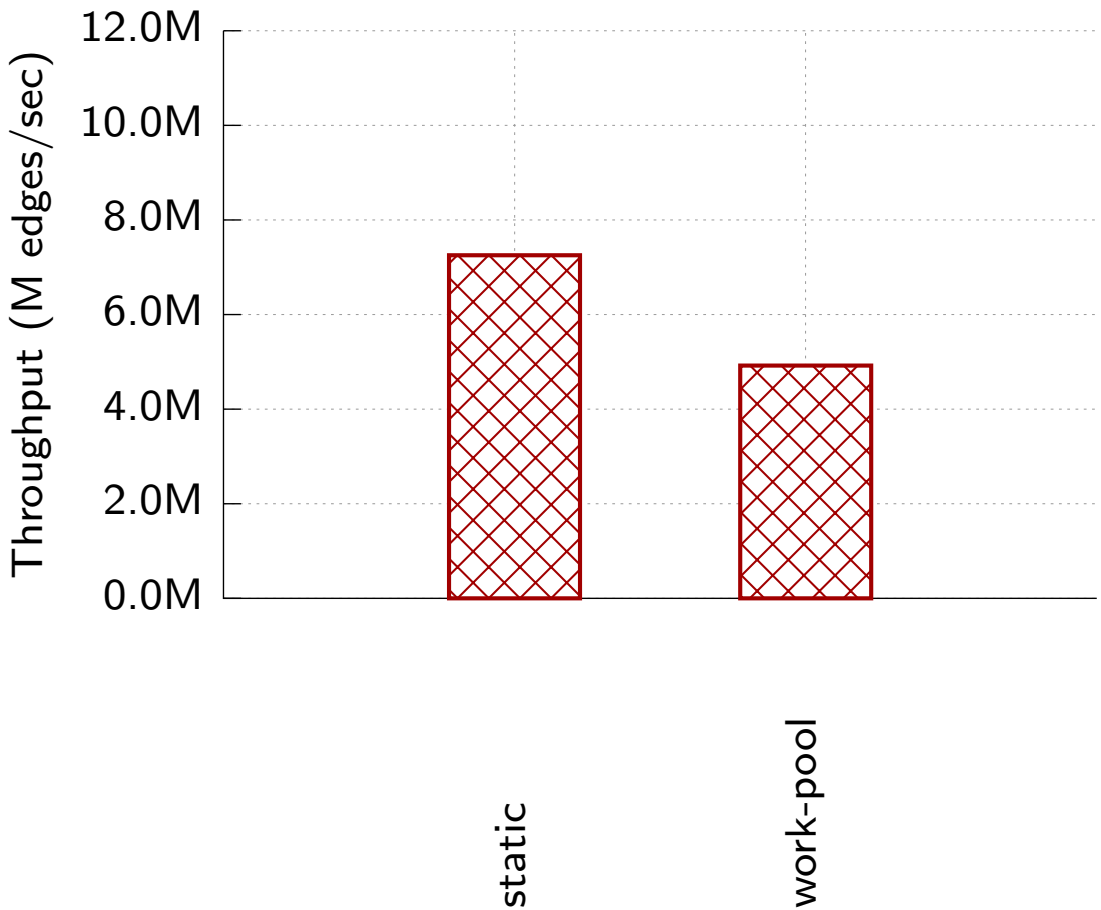
How to load balance cells?

- Problem: Many more cells than threads
  - Millions of cells, dozens of threads
- Skewed cell size distribution
- Four choices implemented:
  - Static partitioning



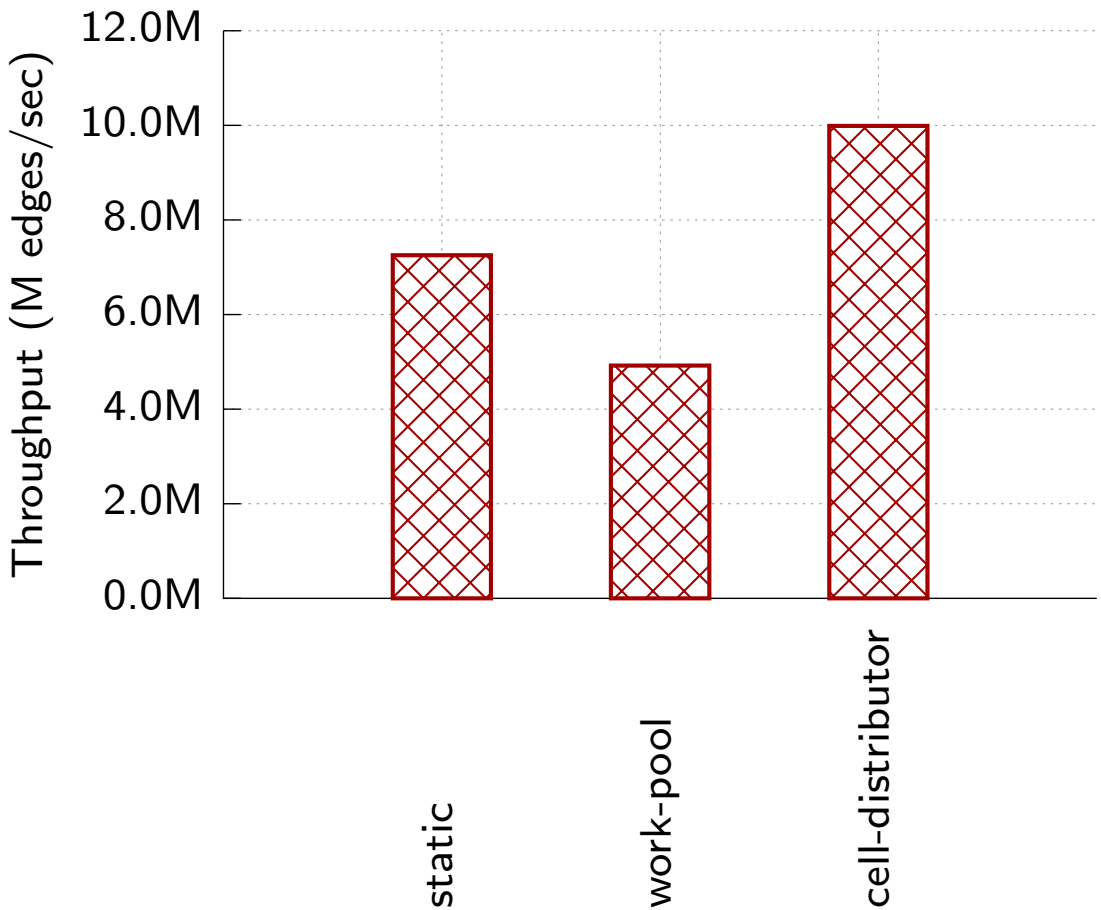
How to load balance cells?

- Problem: Many more cells than threads
  - Millions of cells, dozens of threads
- Skewed cell size distribution
- Four choices implemented:
  - Static partitioning
  - Work pool



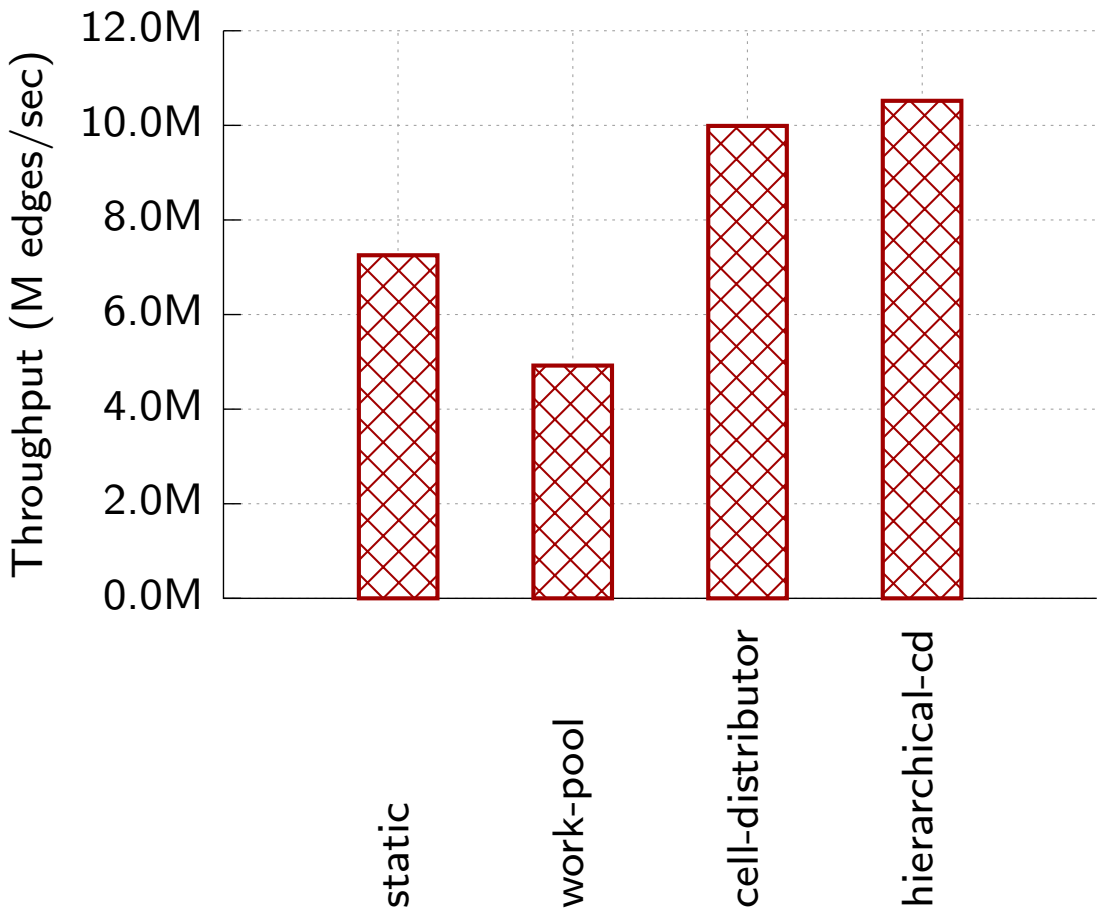
How to load balance cells?

- Problem: Many more cells than threads
  - Millions of cells, dozens of threads
- Skewed cell size distribution
- Four choices implemented:
  - Static partitioning
  - Work pool
  - Cell distributor



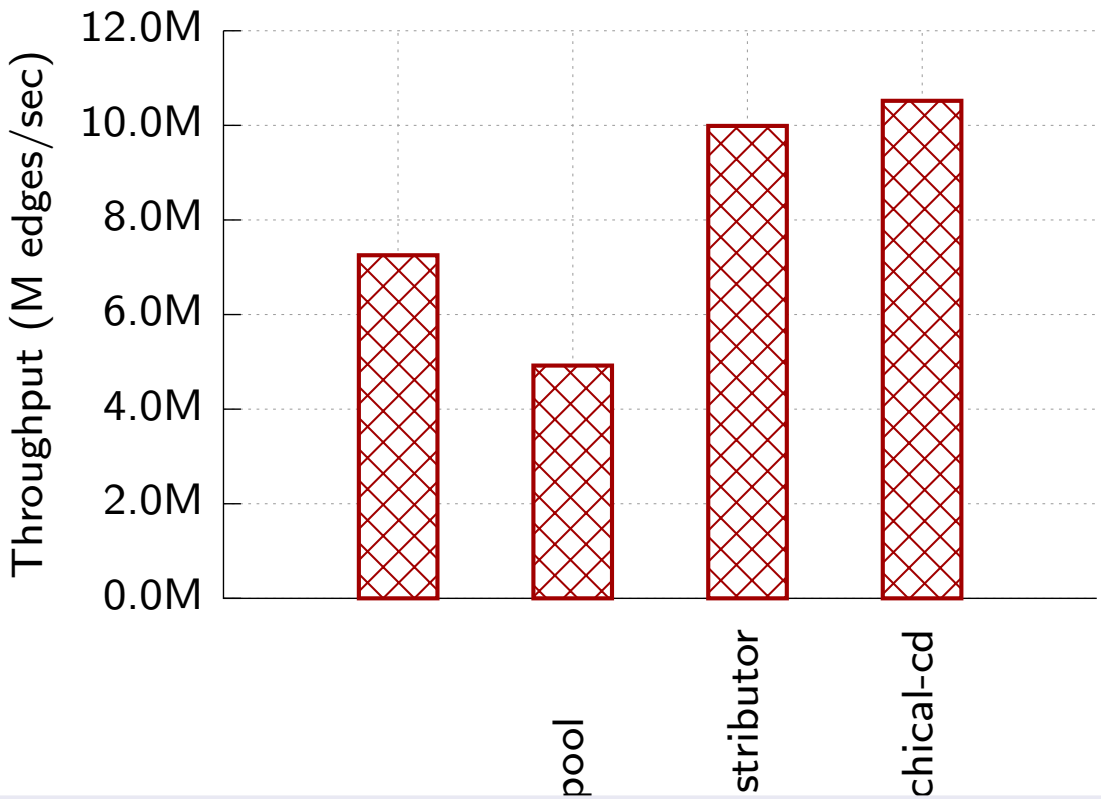
How to load balance cells?

- Problem: Many more cells than threads
  - Millions of cells, dozens of threads
- Skewed cell size distribution
- Four choices implemented:
  - Static partitioning
  - Work pool
  - Cell distributor
  - Hierarchical cell distributor



How to load balance cells?

- Problem: Many more cells than threads
  - Millions of cells, dozens of threads
- Skewed cell size distribution
- Four choices implemented:
  - Static partitioning
  - Work pool
  - Cell distributor
  - Hierarchical cell

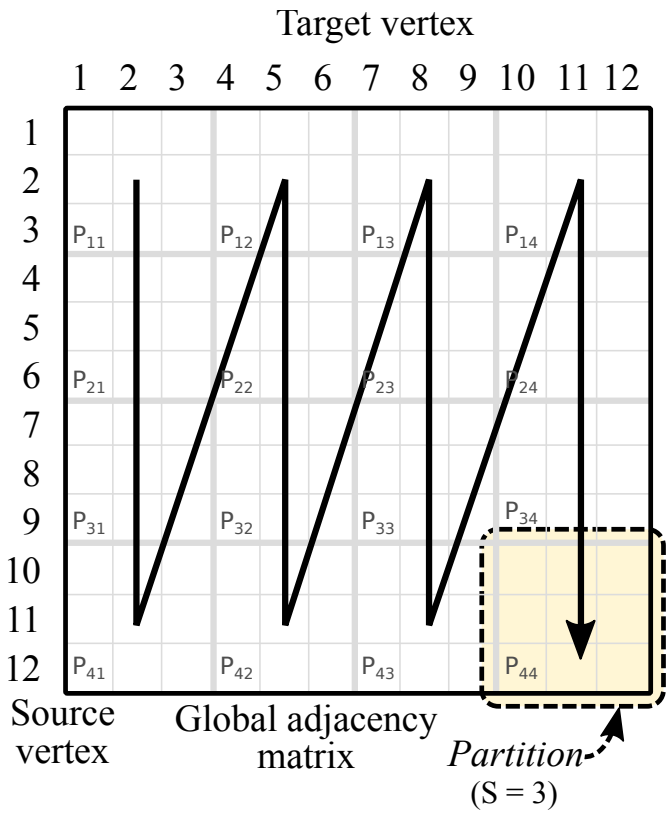


⇒ Hierarchical cell distributor improves throughput by 2.1×
Further optimizations enabled by 2D structure?

2D-partitioning: Opportunity for different traversals

- Multiple traversal strategies are possible with CYTOM

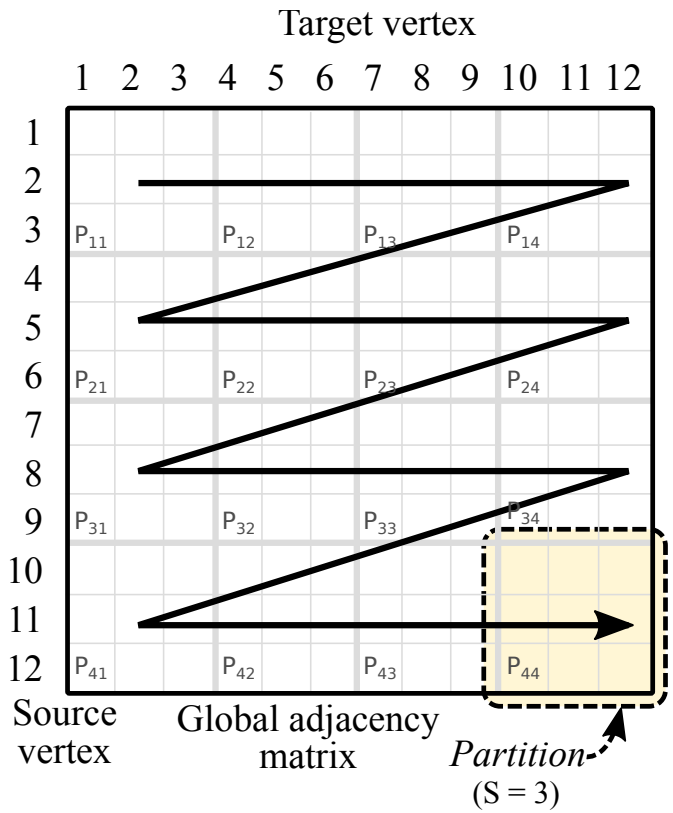
- Evaluated options:
  - Column first
    - **Read** optimized



2D-partitioning: Opportunity for different traversals

- Multiple traversal strategies are possible with CYTOM

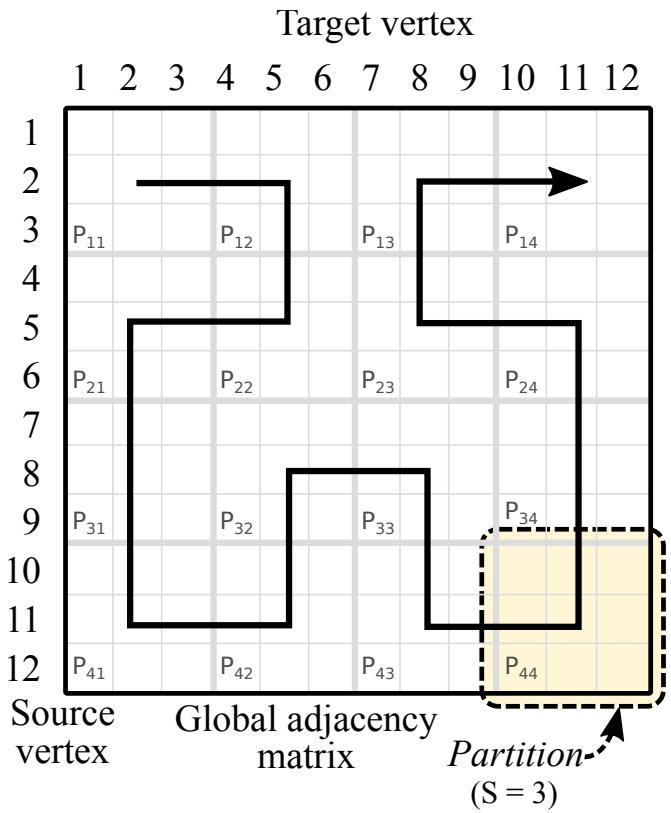
- Evaluated options:
  - Column first
    - **Read** optimized
  - Row first
    - **Write** optimized



2D-partitioning: Opportunity for different traversals

- Multiple traversal strategies are possible with CYTOM

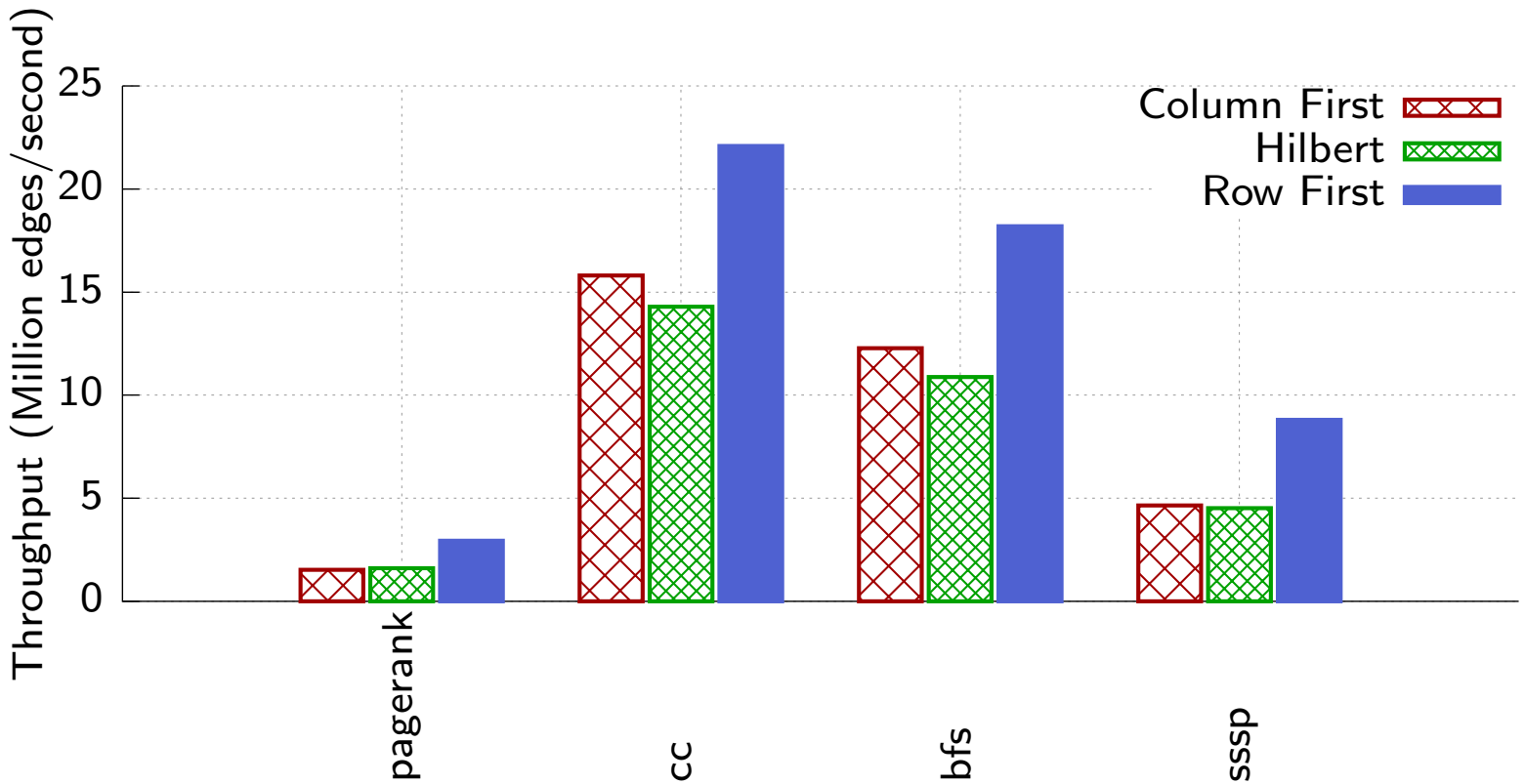
- Evaluated options:
  - Column first
    - **Read** optimized
  - Row first
    - **Write** optimized
  - Hilbert order
    - Mix between **read and write** optimization



Previous results

- With Mosaic: Hilbert order best for locality *and* performance
  - Optimize for **reads** and **writes**
- Mosaic's scenario:
  - Batch-processing of large amounts of edges
  - Entire graph being processed at a time
- However: Cytom's scenario is different:
  - Small sets of changes, e.g., one million edges (maximum: 12 MB of vertex and edge data)
  - Only parts of the graph being processed
- For Cytom: Optimizing **writes** beneficial!

Traversal strategies with CYTOM



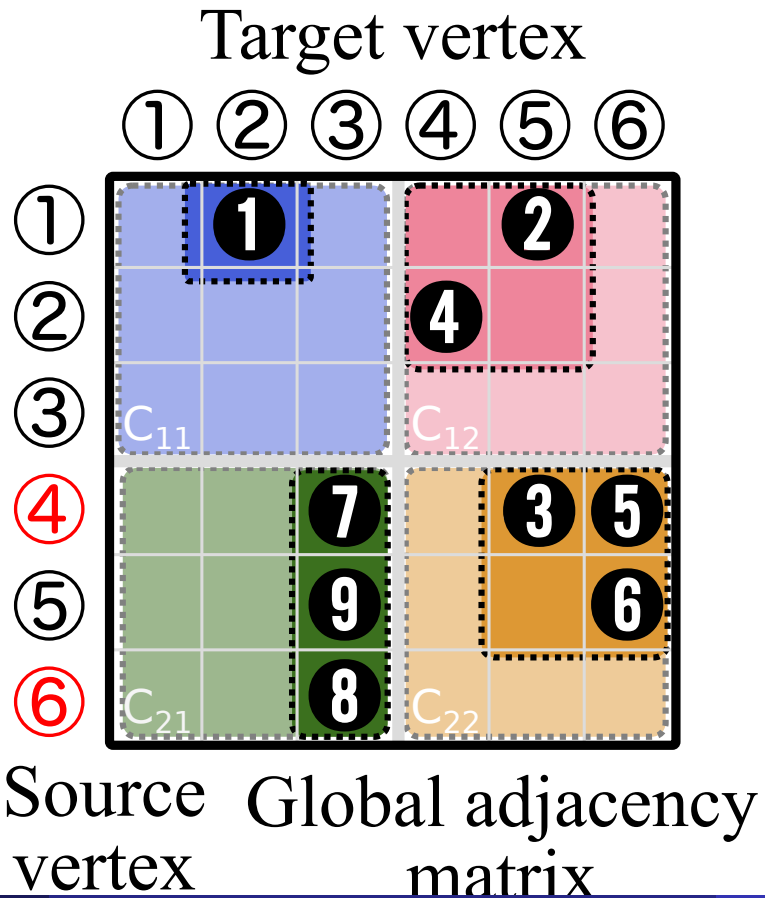
⇒ Row-first, write-optimized strategy performs up to 80% better

Optimization: Selective scheduling

- Important in evolving graphs:
  - Not all vertices active all the time
- Effect: Many inactive edges
- Common in many algorithms
- Idea: Only operate on cells with active vertices

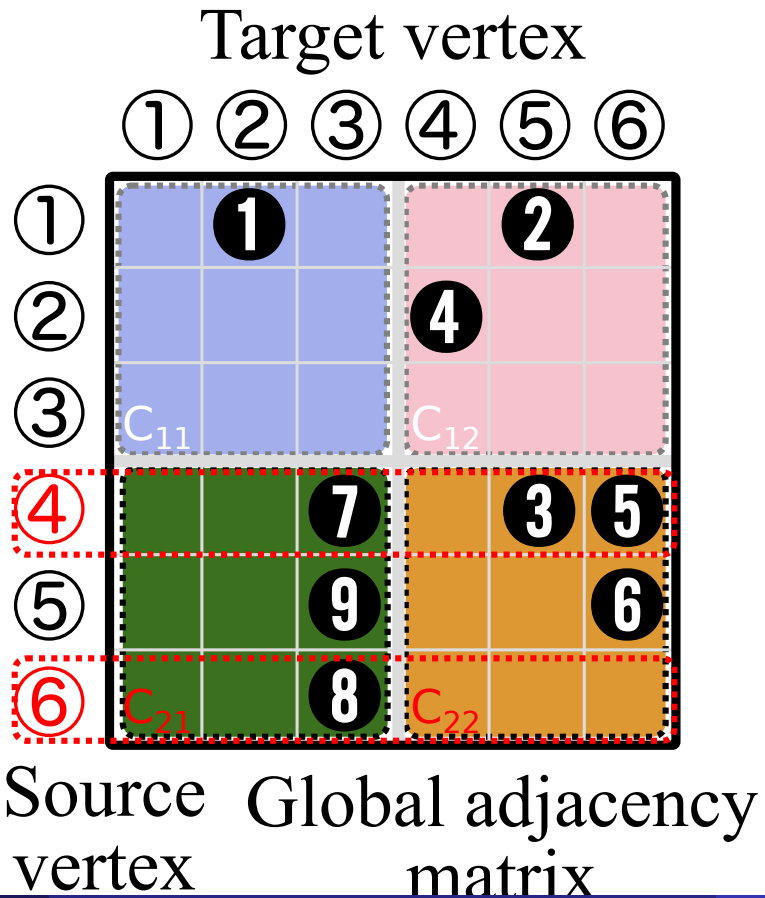
Optimization: Selective scheduling

- Important in evolving graphs:
  - Not all vertices active all the time
- Effect: Many inactive edges
- Common in many algorithms
- Idea: Only operate on cells with active vertices

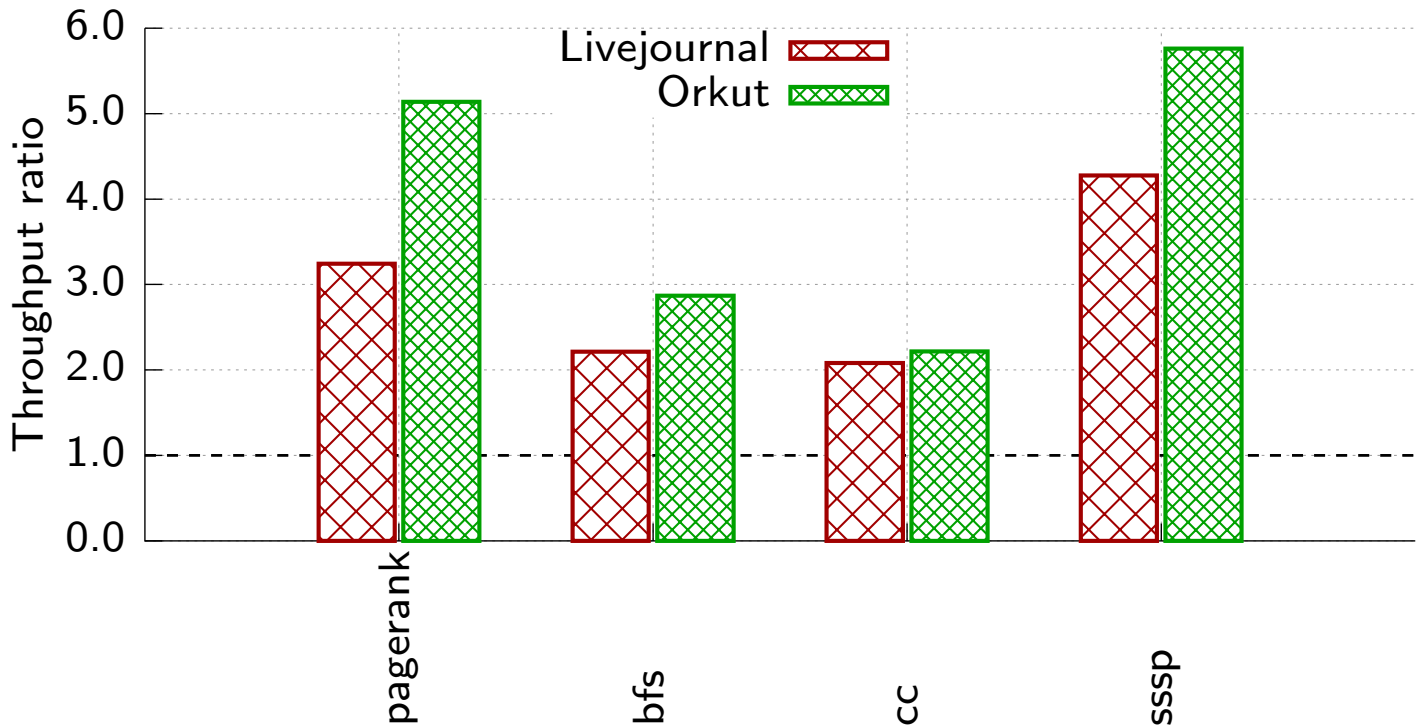


Optimization: Selective scheduling

- Important in evolving graphs:
  - Not all vertices active all the time
- Effect: Many inactive edges
- Common in many algorithms
- Idea: Only operate on cells with active vertices



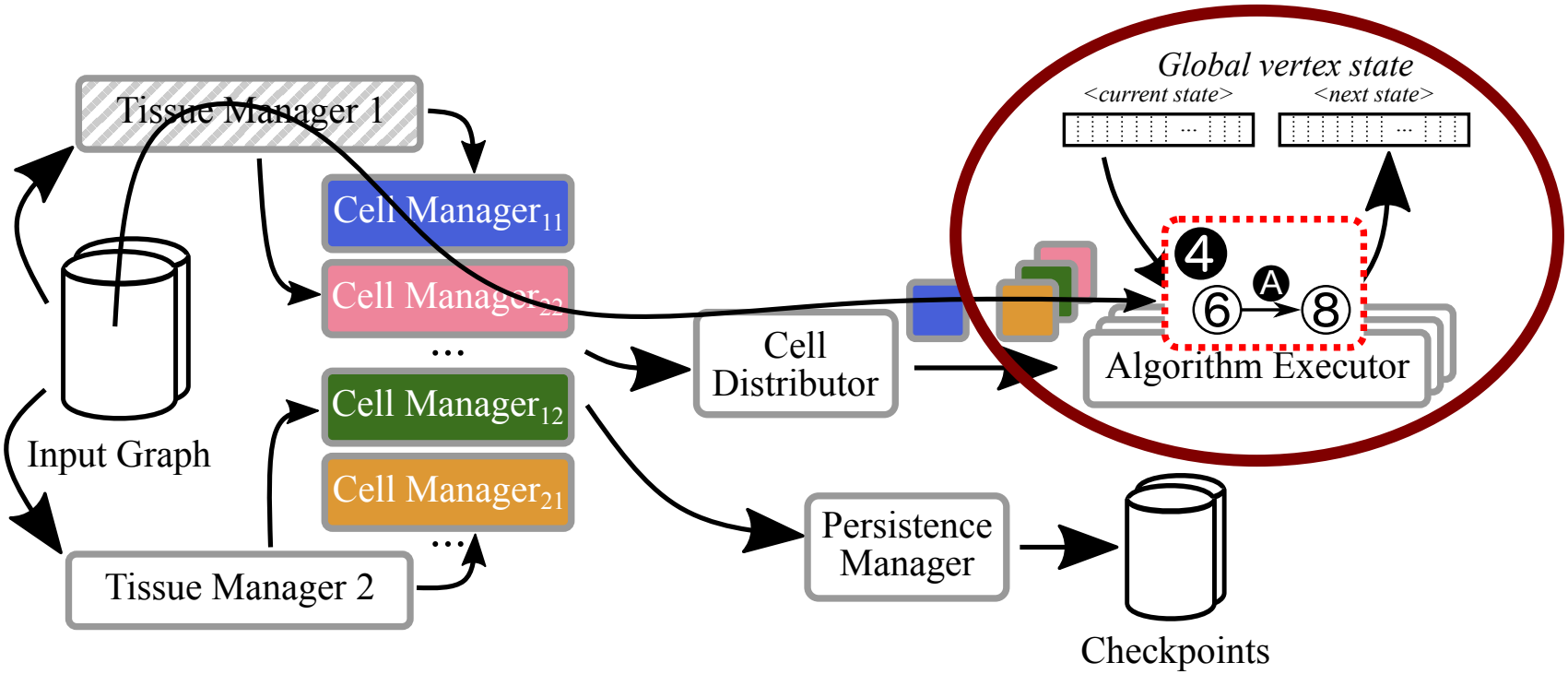
Impact of selective scheduling



$\Rightarrow$ Up to 5.6$\times$ improvement due to reduced number cells

High-level architecture: Algorithmic interface

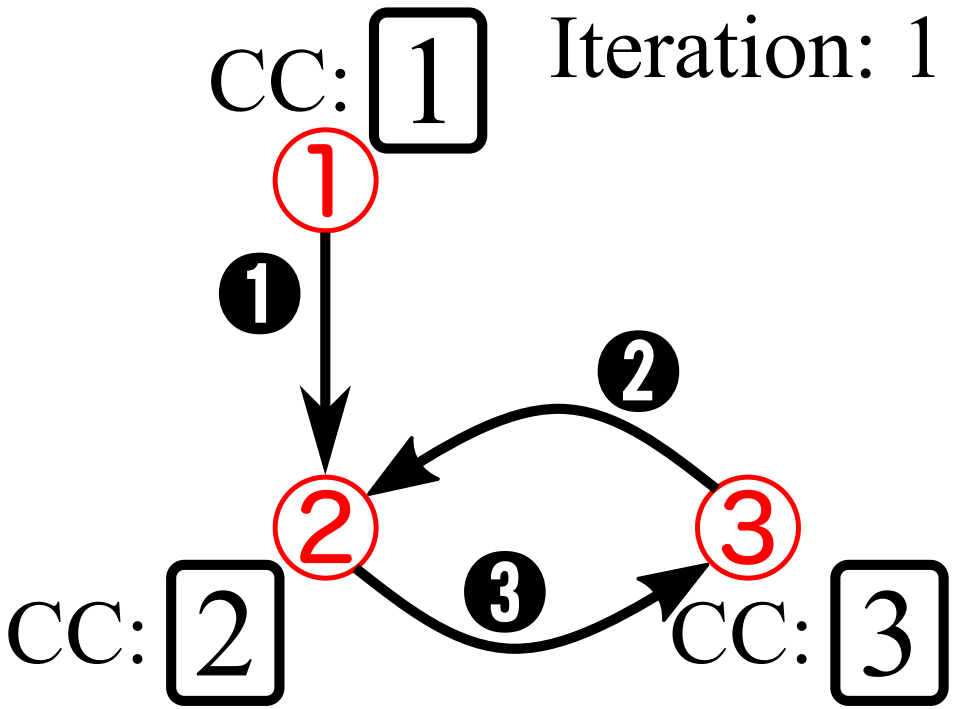
What is CYTOM's algorithmic interface?



Example connected components

- Iterative algorithm
- Minimize connected component over incoming edges

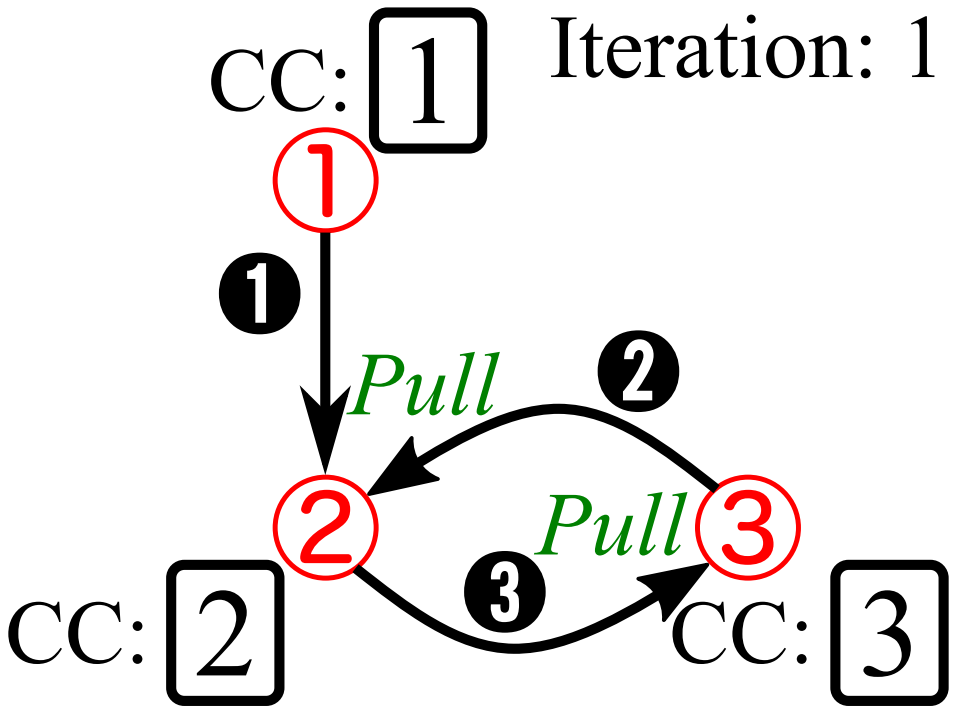
Step 1: Initialize with local vertex ID



Example connected components

- Iterative algorithm
- Minimize connected component over incoming edges

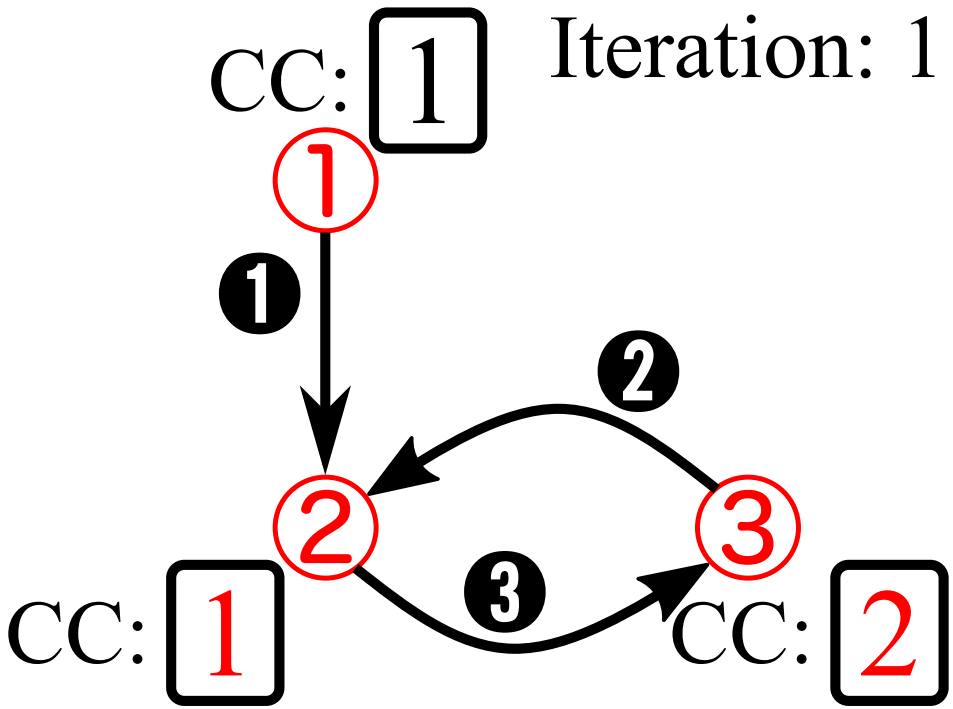
Step 2: Minimize over all incoming edges with *Pull*



Example connected components

- Iterative algorithm
- Minimize connected component over incoming edges

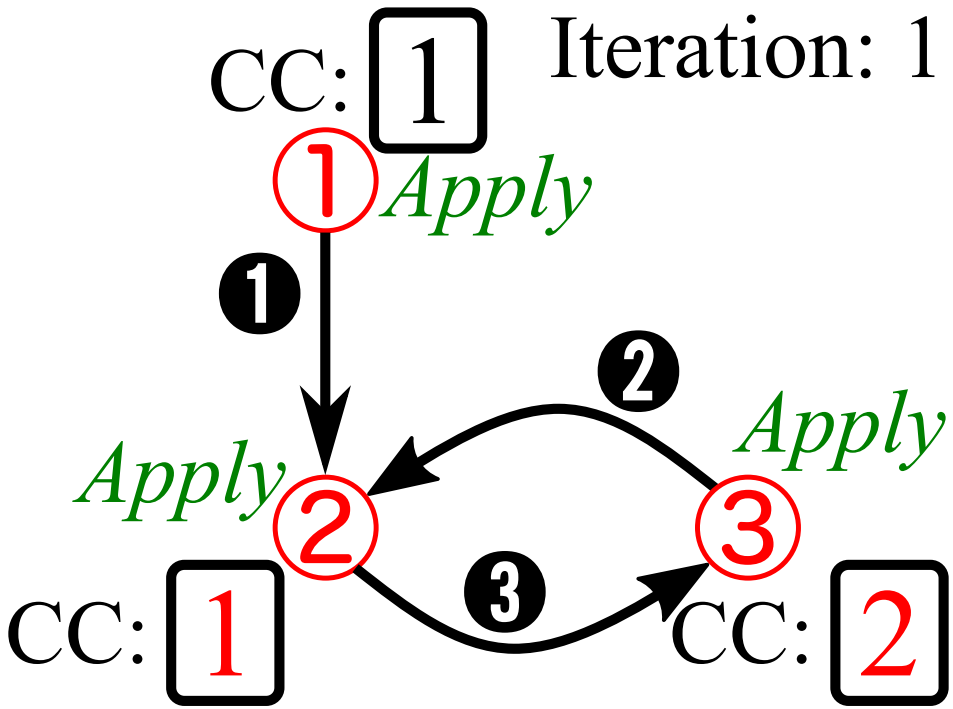
Step 3: Update local connected component value



Example connected components

- Iterative algorithm
- Minimize connected component over incoming edges

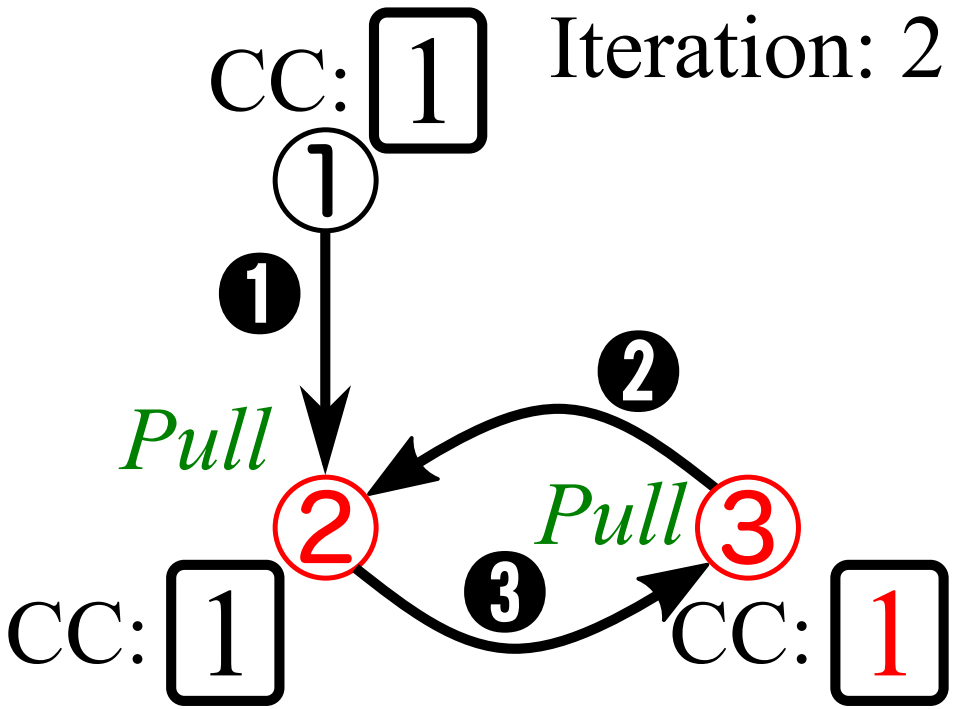
Step 4: Run *apply* and mark active vertices



Example connected components

- Iterative algorithm
- Minimize connected component over incoming edges

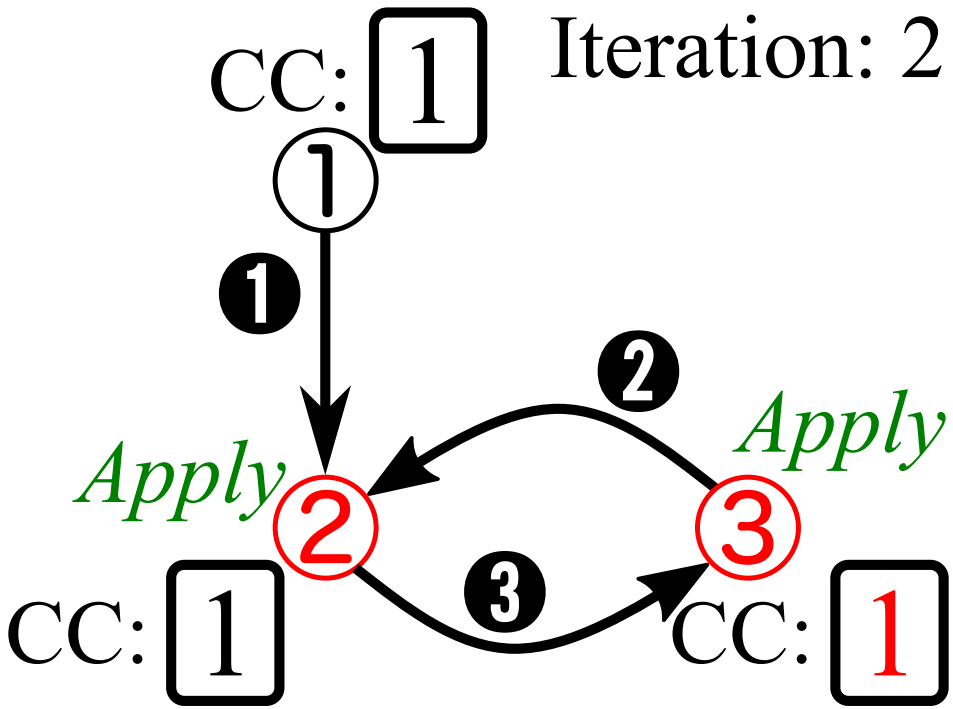
Step 5: New iteration, minimize with *Pull*



Example connected components

- Iterative algorithm
- Minimize connected component over incoming edges

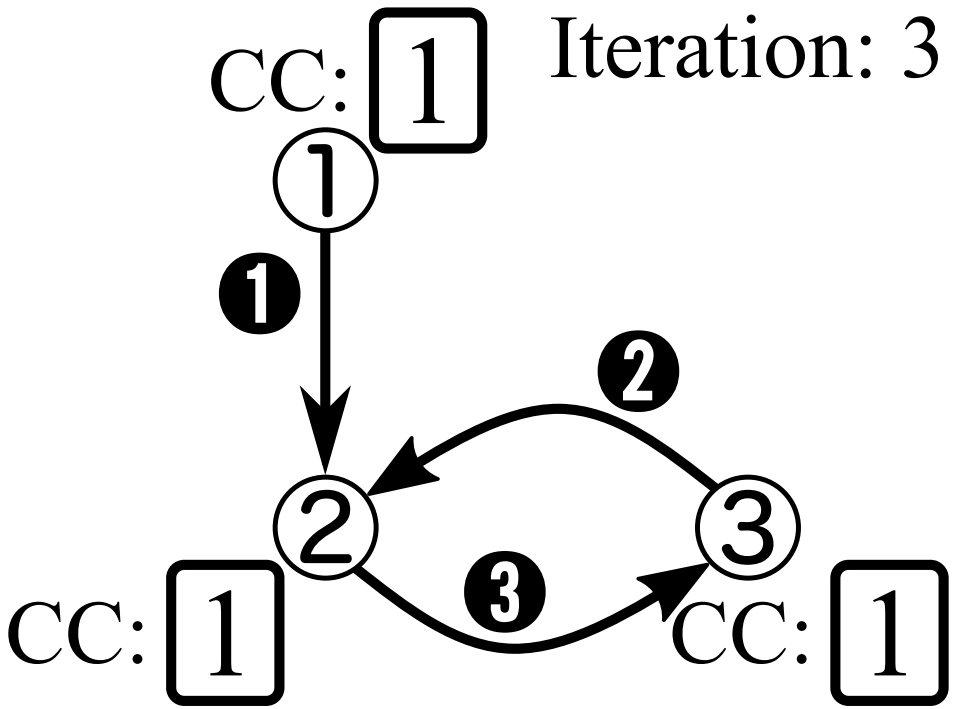
Step 6: Run *Apply* on all active vertices



Example connected components

- Iterative algorithm
- Minimize connected component over incoming edges

Step 7: Convergence reached after one more iteration



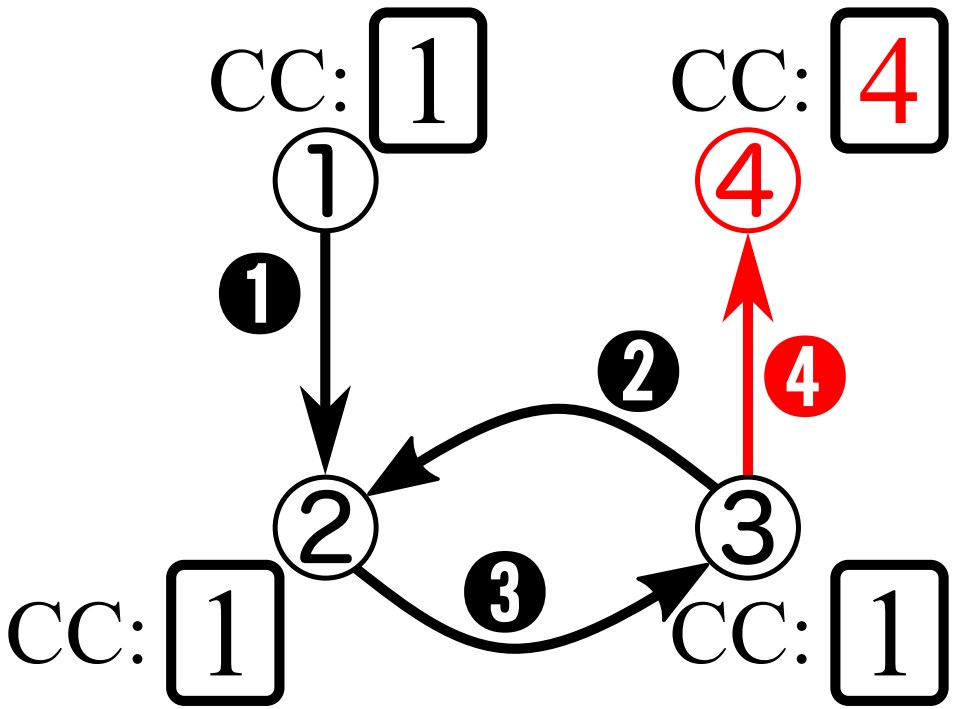
Handling updates

- Add new vertex and edge
- Algorithm has to update connected component

Handling updates

- Add new vertex and edge
- Algorithm has to update connected component

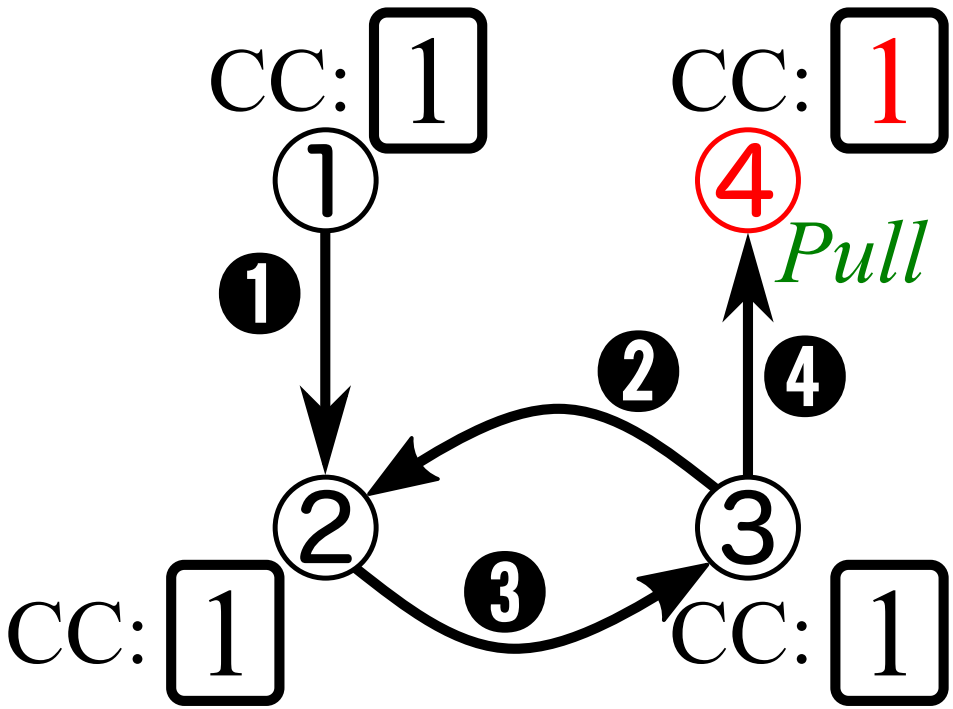
Step 1: Add edge ❹ and vertex ④, init connected component



Handling updates

- Add new vertex and edge
- Algorithm has to update connected component

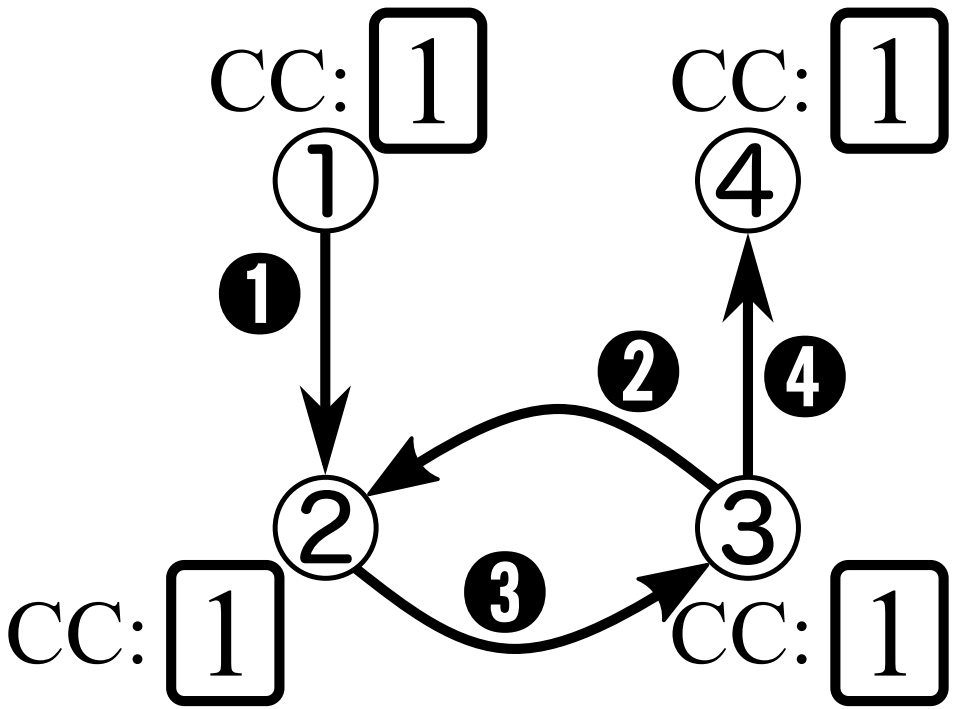
Step 2: *Pull* incoming edge for ④



Handling updates

- Add new vertex and edge
- Algorithm has to update connected component

Step 3: Convergence after next iteration

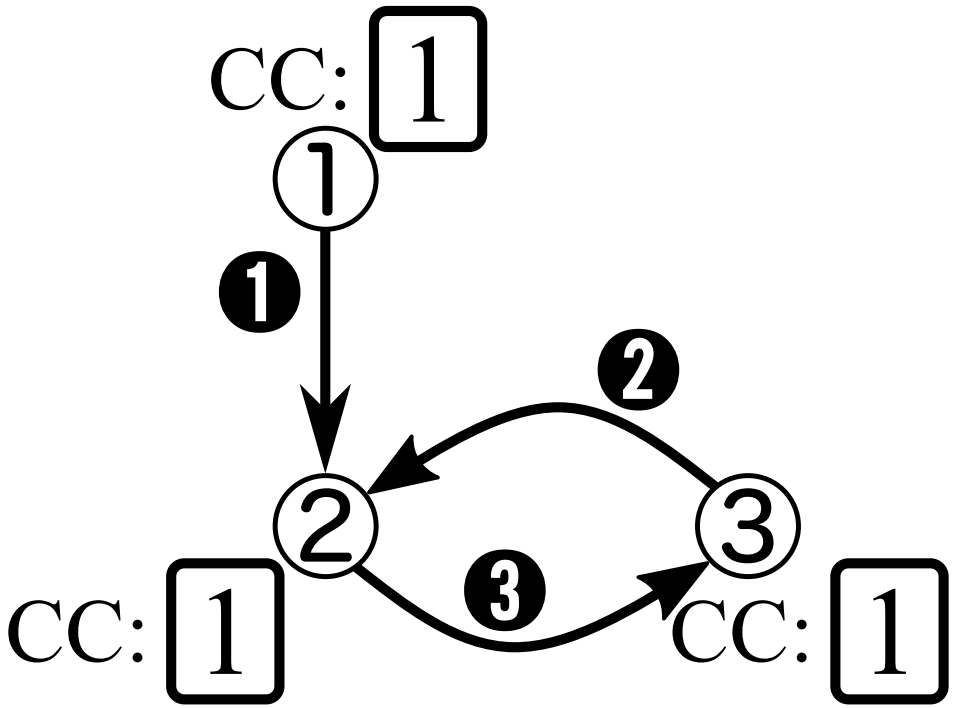


Dealing with deletions: Critical edges

- Potentially devastating impact on algorithmic result
- Example with connected components:

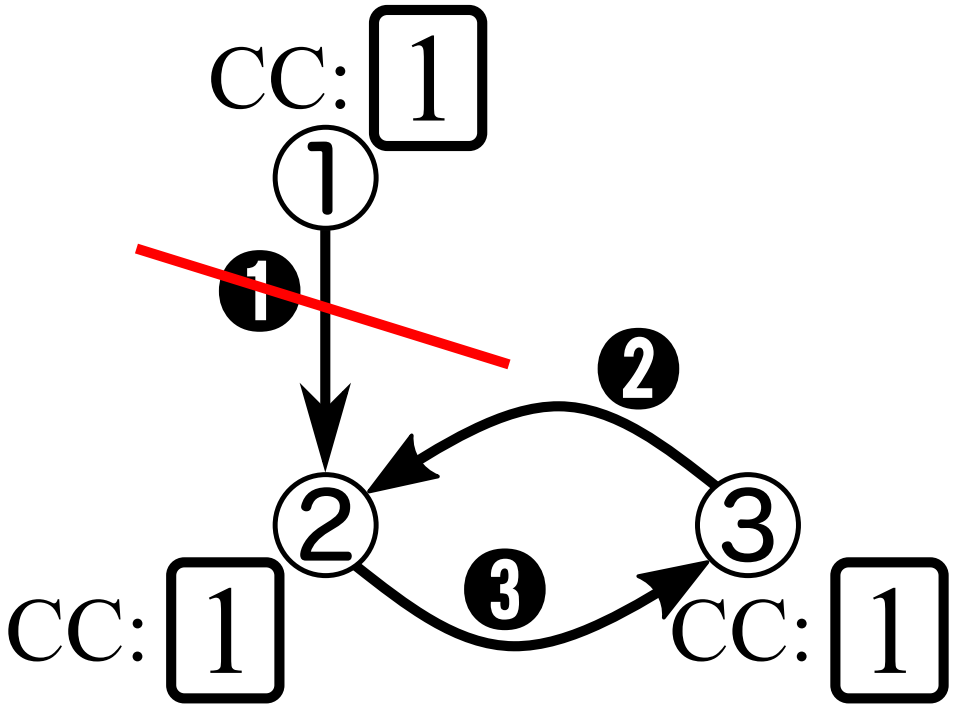
Dealing with deletions: Critical edges

- Potentially devastating impact on algorithmic result
- Example with connected components:



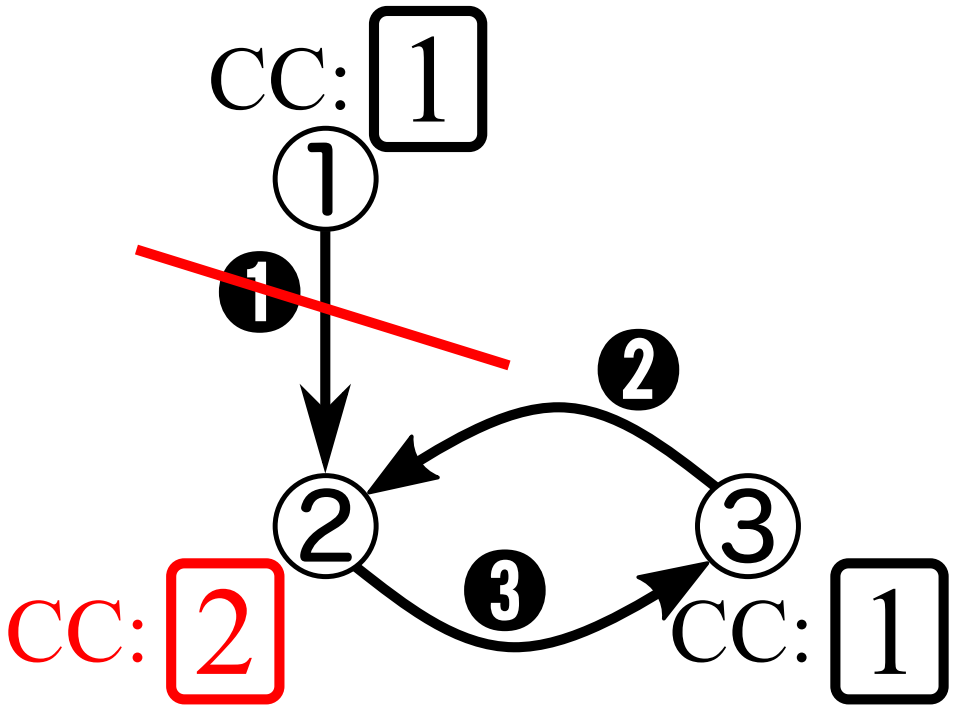
Dealing with deletions: Critical edges

- Potentially devastating impact on algorithmic result
- Example with connected components:



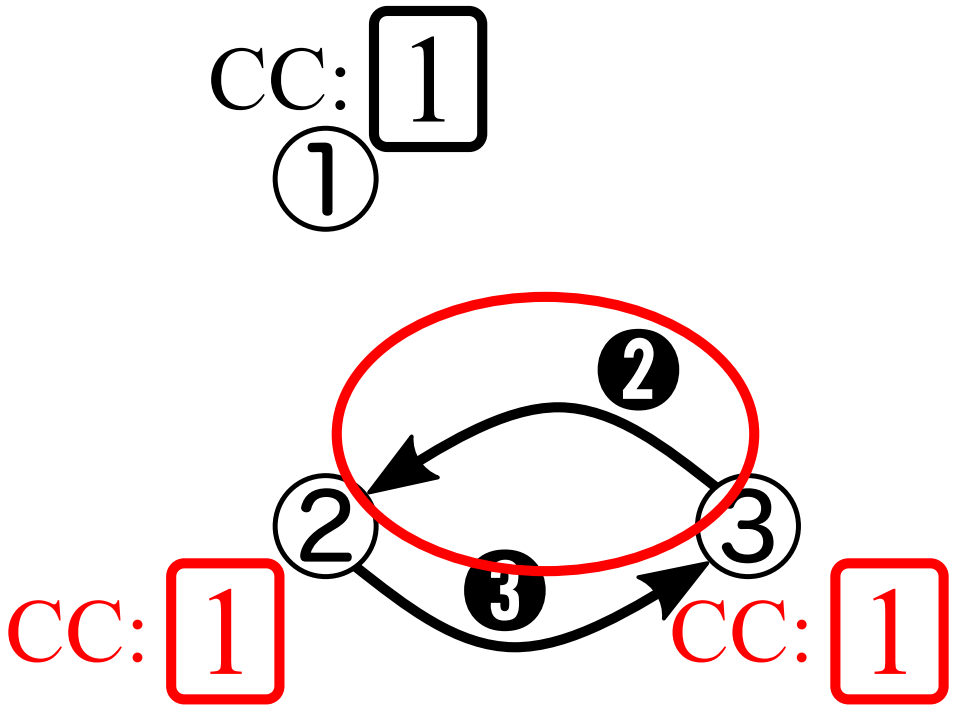
Dealing with deletions: Critical edges

- Potentially devastating impact on algorithmic result
- Example with connected components:



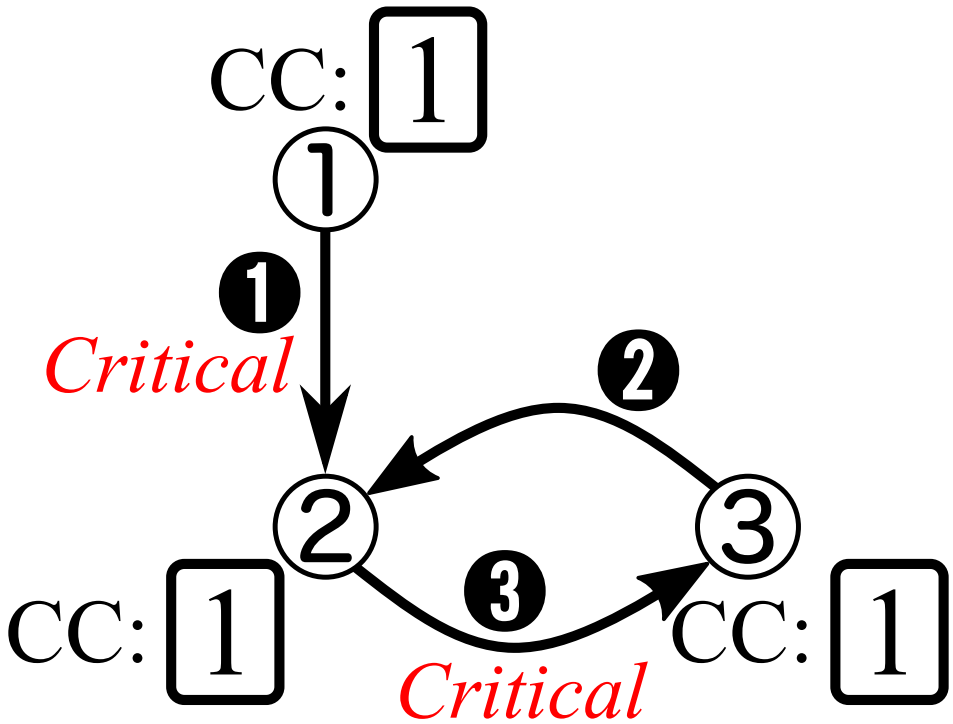
Dealing with deletions: Critical edges

- Potentially devastating impact on algorithmic result
- Example with connected components:



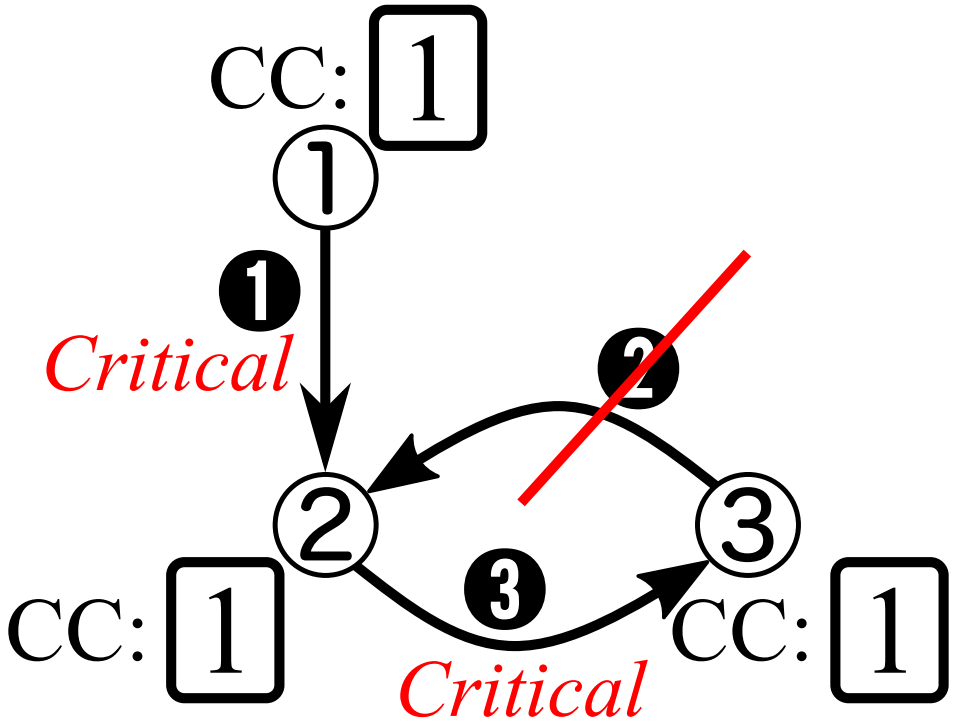
Dealing with deletions: Critical edges

- "Safe" solution: Re-execute algorithm
- CYTOM's approach: Mark *critical* edges carrying results
- If critical edge deleted or updated: Re-execute



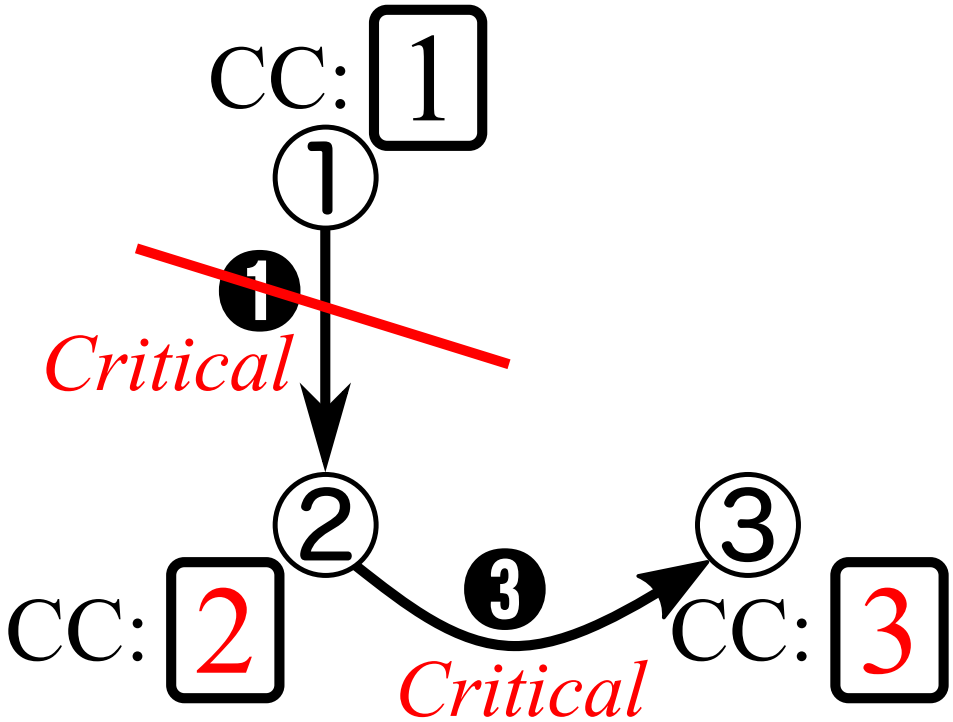
Critical edges: Example

Delete non-critical edge ❷ :



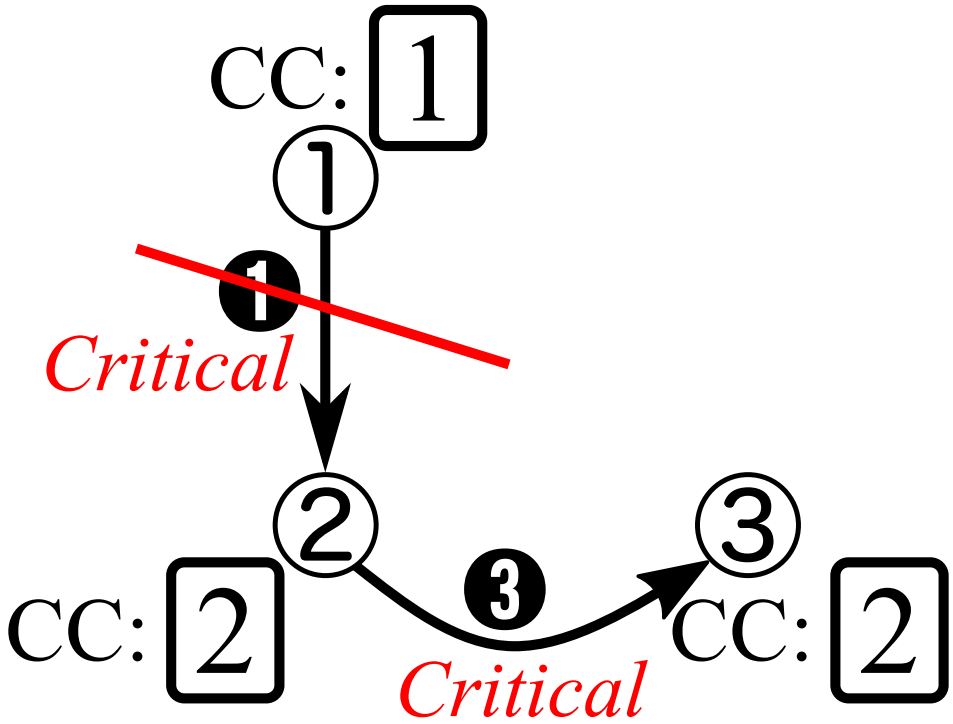
Critical edges: Example

Delete *critical* edge ❶ :



Critical edges: Example

Re-execute algorithm for correct result:



Programming interface

- Scatter-Gather-Apply (GAS) variant
- Algorithm developer not exposed to subgraph-centric aspects
- Special addition for evolving graphs: `edgeChanged` callback
  - React to edge changes
  - Trigger algorithm re-execution if necessary (e.g., deletions)

Example: Connected components

Edge processing callbacks:

Edge-centric operation

Local graph processing on Cell

```
1  // Inside a Cell (local)
2  // Edge e = (Vertex src, Vertex tgt)
3  def Pull(Vertex src, Vertex tgt):
4      if src.value < tgt.value:
5          markCritial(src, tgt)
6          return src.value
7  // Collecting Cell Results (local & global)
8  def Reduce(Vertex v1, Vertex v2):
9      return min(v1.value, v2.value)
10 // On Global Vertices
11 def Apply(Vertex v):
12     if current(v.value) != next(v.value):
13         activate(v)
```

Global graph processing

Vertex-centric operation

Example: Connected components

Special for evolving graphs: edgeChanged callback

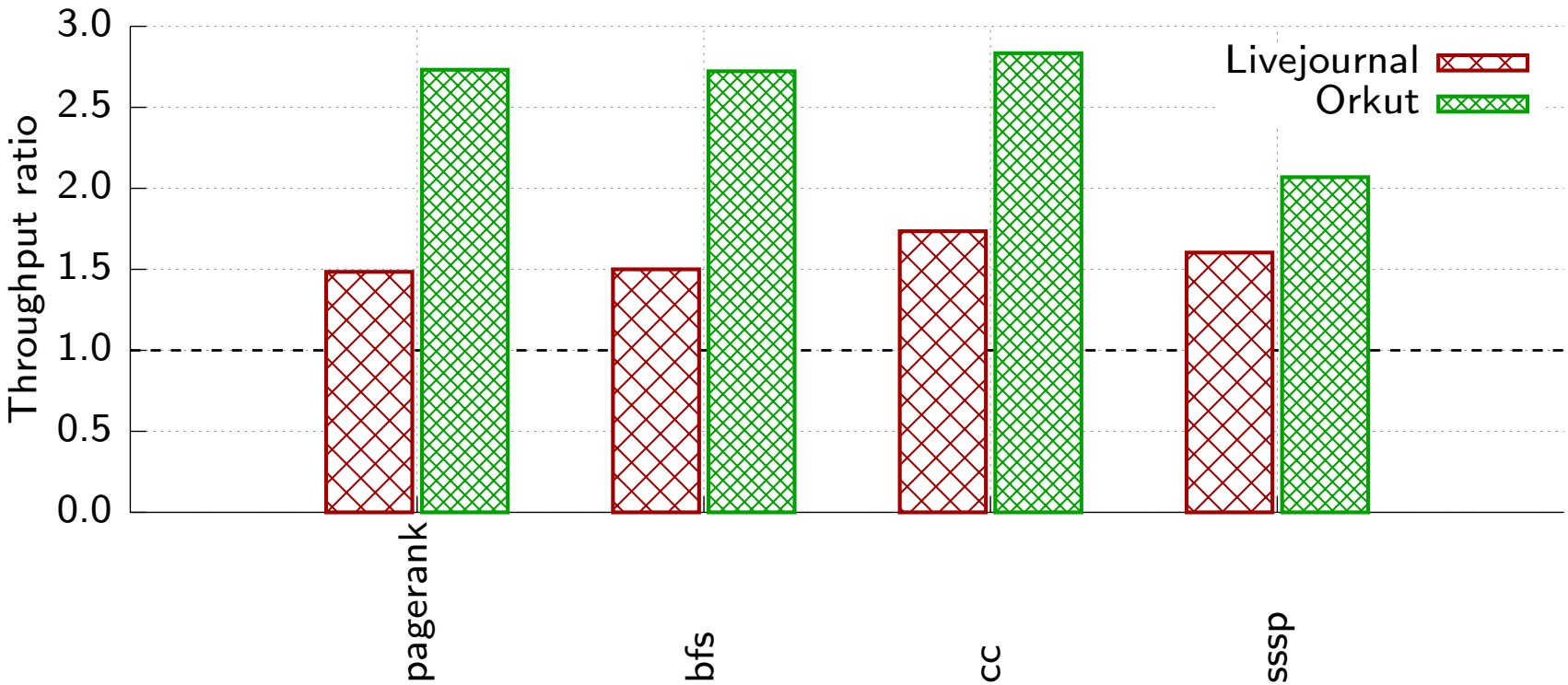
- Filter updates without implied changes
- Handle deletions

```
1  // On Every Edge Update
2  def EdgeChanged(Vertex src, Vertex tgt, Event e):
3      switch(e):
4          case Inserted:
5              // Update if New Path Discovered
6              if src.value < tgt.value:
7                  tgt.value = src.value
8                  activate(tgt)
9                  markCritical(src, tgt)
10         case Deleted:
11             if isCritical(src, tgt):
12                 reExecuteAlgorithm()
```

Edge-centric operation

Impact of edgeChanged callback for insertions

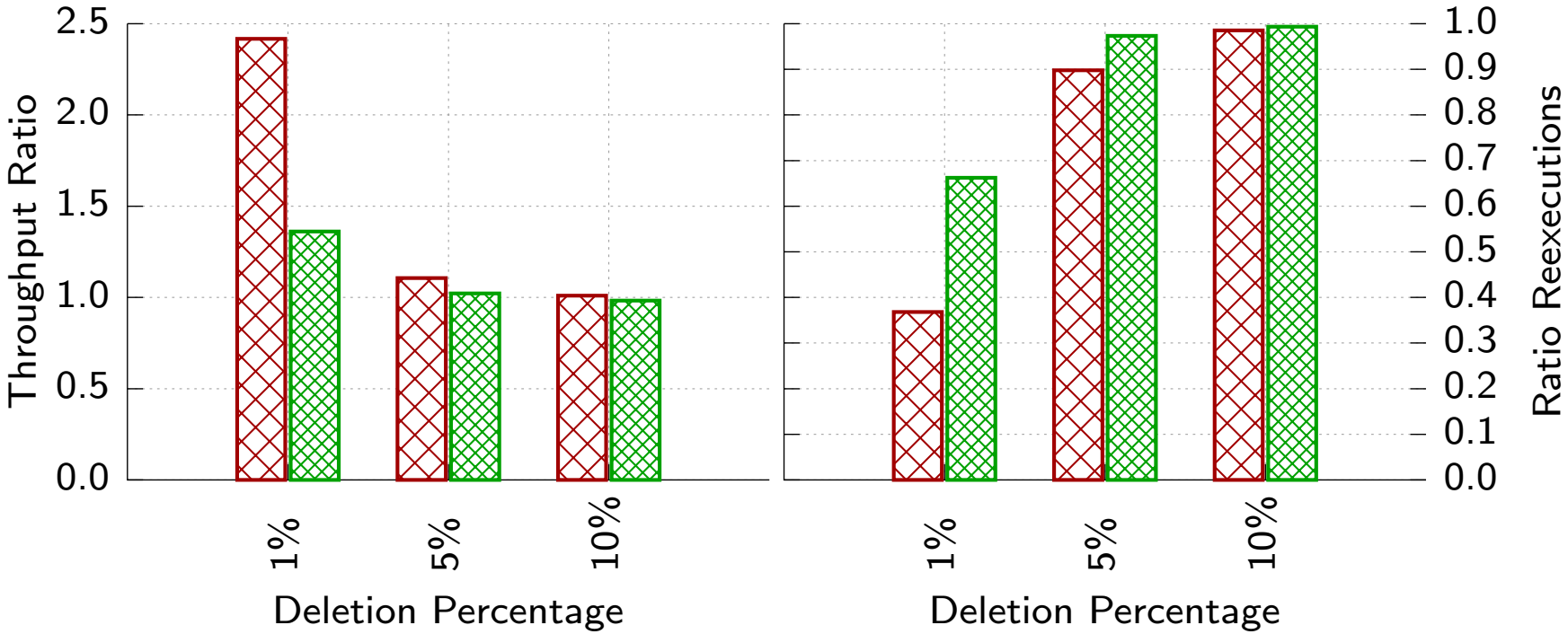
- Benefit: Filter out uninteresting updates



⇒ Up to 2.8× improvement in overall throughput due to reduced work for algorithm

Impact of edgeChanged callback for deletions

Re-execution vs. *critical* edges approach with CC:



⇒ Up to 2.2× improvement, less when more deletions

Comparison to other systems

- How does Cytom perform compared to other systems?
- Metrics:
  - Throughput of graph updates
  - Overall throughput with updates and algorithm processing

High-level comparison

- Features offered by each system

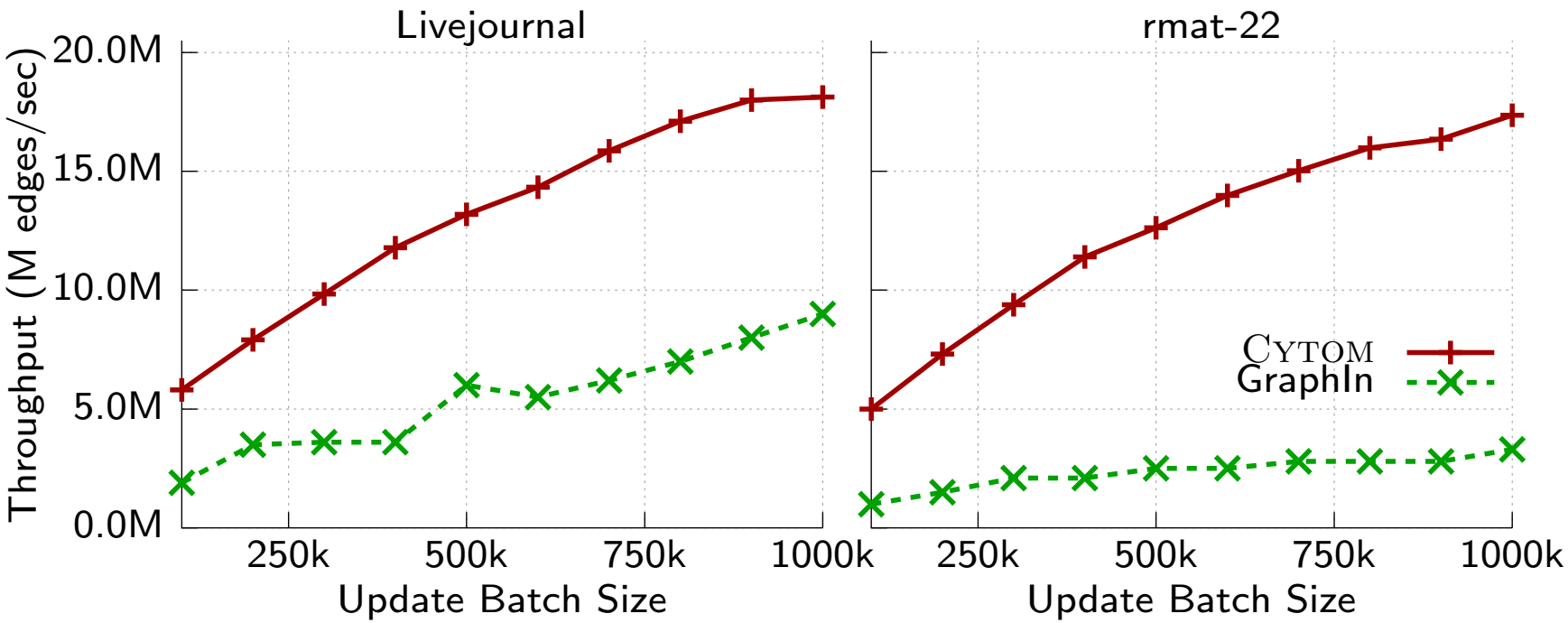
Characteristics	LLAMA	GraphIn	GraphOne	Stinger	Cytom
Incremental proc.	-	✓	✓	✓	✓
Synchronous proc.	✓	✓	✓	✓	✓
Asynchronous proc.	-	-	-	-	✓
Approximation	-	-	-	-	✓
Disk persistence	✓	-	✓	-	✓
Deletions	✓	✓	✓	-	✓
Snapshot view	✓	-	-	-	✓

Throughput for graph updates

- Compared to GraphOne, CYTOM offers 2.8× the throughput
  - Uses CSR for static snapshots, edge lists for new data
  - Log-structured approach
- Compared to Differential Dataflow, CYTOM offers 4.8× the throughput
  - Generalized data flow system
  - Graph support with edge lists

Comparison to GraphIn

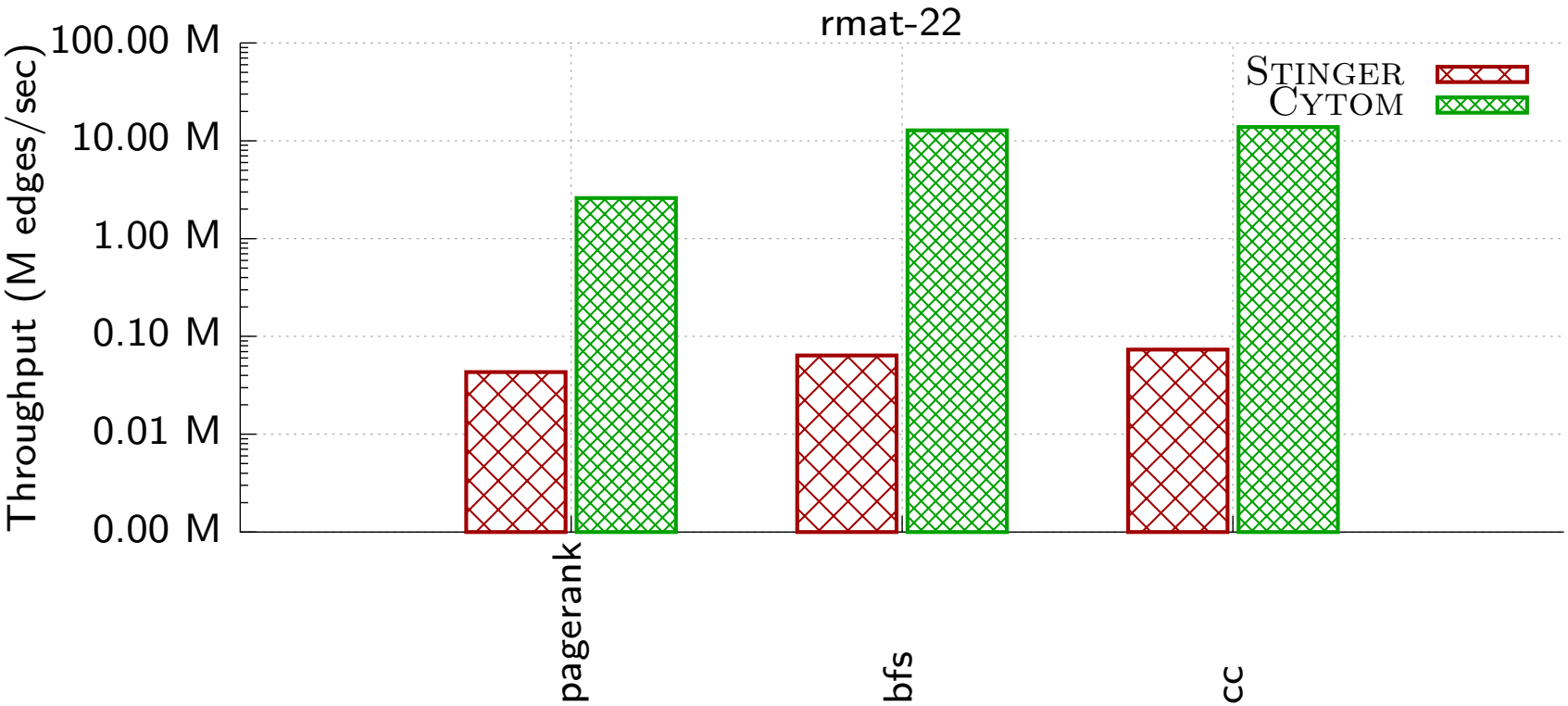
- GraphIn uses edge lists
- Only scales to $\approx$ 100M edges



$\Rightarrow$ Up to 5.5$\times$ improvement in throughput

Comparison to STINGER

- STINGER's focus: Use atomics to insert into adjacency lists
- Earliest systems of all evaluated ones



⇒ At least 60× speedup due to more efficient graph updates and APIs

Conclusion

- Cytom, a billion-scale engine for evolving graphs
  - Cell-based design enables high update rate
  - Selective scheduling enabled by cell-based design
  - edgeChanged callback to quickly react to graph updates

Conclusion

- Cytom, a billion-scale engine for evolving graphs
  - Cell-based design enables high update rate
  - Selective scheduling enabled by cell-based design
  - `edgeChanged` callback to quickly react to graph updates
- Additional components of dissertation:
  - Mosaic, a trillion-scale engine for static graphs
  - Latr, an OS-level approach to reduce overheads of synchronous TLB shootdowns
- Outcome: We can do large-scale data analytics on a single machine

Conclusion

- CYTOM, a billion-scale engine for evolving graphs
  - Cell-based design enables high update rate
  - Selective scheduling enabled by cell-based design
  - **edgeChanged** callback to quickly react to graph updates
- Additional components of dissertation:
  - MOSAIC, a trillion-scale engine for static graphs
  - LATR, an OS-level approach to reduce overheads of synchronous TLB shootdowns
- Outcome: We can do large-scale data analytics on a single machine

Thanks!