Popular Conjectures and Dynamic Problems

Thanks to Amir Abboud for some of his slides!

Plan

- ➡ Overview of some lower bounds for dynamic problems
- ➡ Simple and powerful proofs

Dynamic graph algorithms

Given initial graph G, can preprocess it.
Edge updates: insert(u,v), delete(u,v)

Queries: (depend on the problem)
How many SCCs are there? Can u reach v? …

Dynamic graph algorithms

Given initial graph G, can preprocess it.
Edge updates: insert(u,v), delete(u,v)

Queries: (depend on the problem)
How many SCCs are there? Can u reach v? …

Want to minimize the preprocessing, *update* and *query* times.

- Worst case time
- Amortized time
- Total time (over all updates)

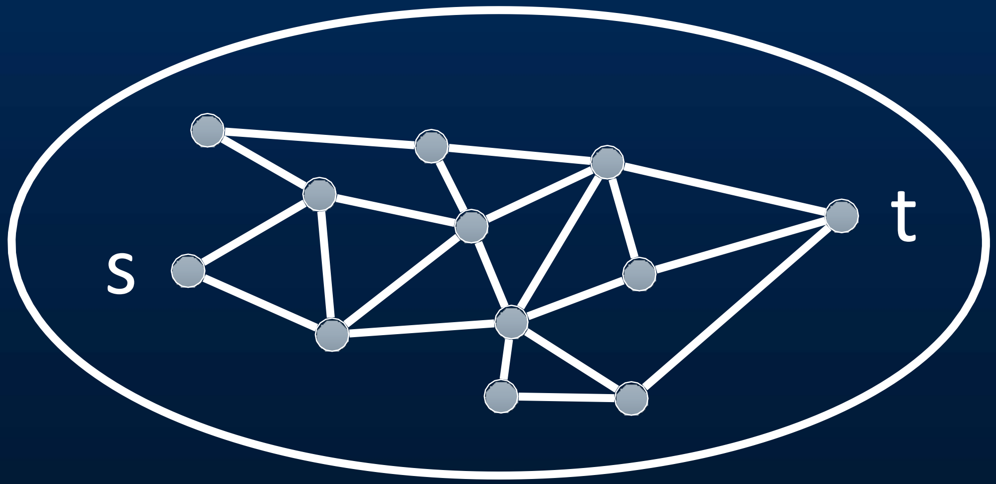
Dynamic Problems

Dynamic (undirected) Connectivity

<u>Input</u>: an undirected graph G

<u>Updates:</u> Add or remove edges.

<u>Query:</u> Are s and t connected?



Trivial algorithm: O(m) time per update.
[Thorup STOC 01]: O(log m (log log m)^3) amortized time per update.

[Pătraşcu - Demaine STOC 05]:
$\Omega(\textit{log}\ m)$ Cell-probe lower bound.

Great!

Dynamic Problems

Dynamic (directed) Reachability

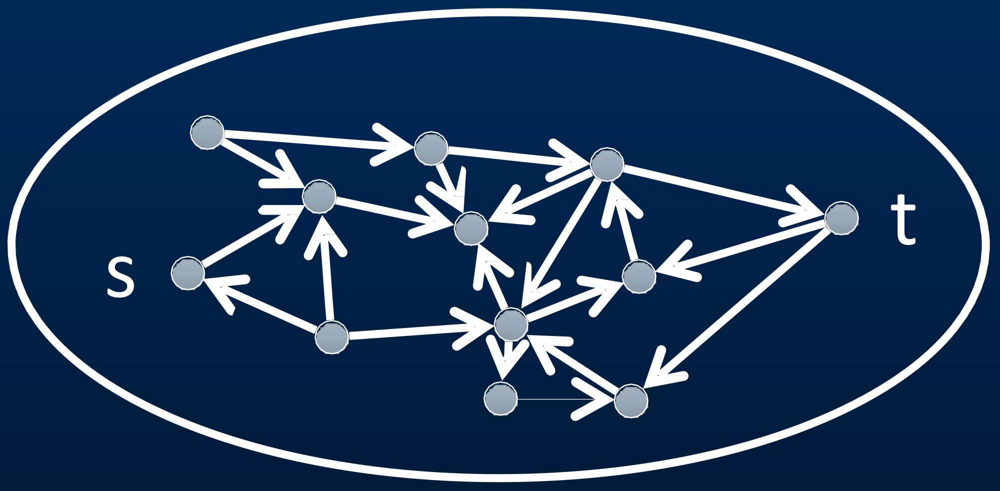
Input: A directed graph G.

Updates: Add or remove edges.

Query:

s,t-Reach: Is there a path from s to t?

#SSR: How many nodes can s reach?



Trivial algorithm: O(m) time updates

Using fast matrix multiplication
[Sankowski FOCS 04'] $O(n^{1.58})$

Not great.

Best cell probe lower bound still $\Omega(\mathit{log}\ m)$

Many Examples

<table>
<tr><th>Problem</th><th>Upper bound</th><th>(Unconditional) Lower bound</th></tr>
<tr><td>s,t--Reach</td><td rowspan="4">O(m) or O(n)</td><td rowspan="6">Ω(log m)</td></tr>
<tr><td>#SSR</td></tr>
<tr><td>Strongly Connected Components</td></tr>
<tr><td>Maximum Matching</td></tr>
<tr><td>Connectivity with node updates</td><td>O(m)</td></tr>
<tr><td>Approximate Diameter</td><td>O(mn)</td></tr>
</table>

Many successes for the partially dynamic setting and related problems.

Huge gaps –what is the right answer?

Today:

Much higher lower bounds via the fine-grained approach

3SUM Lower Bounds

Theorem [Pătraşcu STOC10]: The 3-SUM conjecture implies polynomial lower bounds for many dynamic problems.

3-SUM: Given n integers, are there 3 that sum to 0?

The 3-SUM Conjecture: “No O(n^{2-eps}) time algorithm”

A very cool series of reductions…

Problem	Upper bound	(3-SUM) Lower bound
s,t--Reach	O(m) or O(n)	m^a *for some a>0*
#SSR		
Connectivity with node updates	O(m)	

No poly log updates for Reachability!

3SUM Lower Bounds

[Abboud-VW FOCS '14], [Kopelowitz - Pettie - Porat. SODA '16]
Optimized Pătraşcu's reductions and added problems to the list

Problem	Upper bound	(3-SUM) Lower bound
s,t-Reach	O(m) or O(n)	$m^{1/3}$
#SSR		
Strongly Connected Components		
Maximum Matching		
Connectivity with node updates	O(m)	
Approximate Diameter	O(mn)	

Some steps in the reduction are lossy –stuck at $m^{1/3}$.

3SUM might not be the most appropriate…

BMM Lower Bounds

[Abboud-VW FOCS 14']

The BMM conjecture implies tight lower bounds for combinatorial algorithms

The BMM conjecture:
"No $O(n^{3-eps})$ time combinatorial algorithm
for Boolean Matrix Multiplication"

<table>
<tr><th>Problem</th><th>(combinatorial) Upper bound</th><th>(BMM) Lower bound</th><th>(3-SUM) Lower bound</th></tr>
<tr><td>#SSR</td><td rowspan="4">$O(m)$</td><td rowspan="4">m</td><td rowspan="5">$m^{1/3}$</td></tr>
<tr><td>Strongly Connected Components</td></tr>
<tr><td>s,t-Reach</td></tr>
<tr><td>Maximum Matching</td></tr>
<tr><td>Approximate Diameter</td><td>$O(mn)$</td><td>n</td></tr>
</table>

Any improvement for these problems will probably have to use
fast matrix mult.

OMv Lower Bounds

[Henzinger - Krinninger - Nanongkai - Saranurak STOC'15]

Most BMM lower bounds hold for non-combinatorial algorithms as well, under the Online Matrix Vector Multiplication Conjecture.

OMv Lower Bounds

[Henzinger - Krinninger - Nanongkai - Saranurak STOC'15]
Most BMM lower bounds hold for non-combinatorial algorithms as well, under the Online Matrix Vector Multiplication Conjecture.

OMv problem: Given n x n Boolean matrix A and n Boolean vectors $v_1,\ldots,v_n$, given online, return each $A \cdot v_i$ right after v_i has been given.

OMv Lower Bounds

[Henzinger - Krinninger - Nanongkai - Saranurak STOC'15]
Most BMM lower bounds hold for non-combinatorial algorithms as well, under the Online Matrix Vector Multiplication Conjecture.

OMv problem: Given n x n Boolean matrix A and n Boolean vectors $v_1,\ldots,v_n$, given online, return each $A \cdot v_i$ right after v_i has been given.

[Green-Larsen, Williams'17]: One can compute $A \cdot v_i$ for all i online, in $n^3/2^{\Omega(\sqrt{\log n})}$ total time.

OMv Lower Bounds

[Henzinger - Krinninger - Nanongkai - Saranurak STOC '15]
Most BMM lower bounds hold for non-combinatorial algorithms as well, under the Online Matrix Vector Multiplication Conjecture.

OMv problem: Given n x n Boolean matrix A and n Boolean vectors $v_1, \ldots, v_n$, given online, return each $A \cdot v_i$ right after v_i has been given.

[Green-Larsen, Williams'17]: One can compute $A \cdot v_i$ for all i online, in $n^3/2^{\Omega(\sqrt{\log n})}$ total time.

OMv Conjecture: OMv requires $n^{3-o(1)}$ total time.

OMv Lower Bounds

[Henzinger - Krinninger - Nanongkai - Saranurak STOC'15]
Most BMM lower bounds hold for non-combinatorial algorithms as well, under the Online Matrix Vector Multiplication Conjecture.

OMv problem: Given n x n Boolean matrix A and n Boolean vectors $v_1,\ldots,v_n$, given online, return each $A \cdot v_i$ right after v_i has been given.

[Green-Larsen, Williams'17]: One can compute $A \cdot v_i$ for all i online, in $n^3/2^{\Omega(\sqrt{\log n})}$ total time.

OMv Conjecture: OMv requires $n^{3-o(1)}$ total time.

[Cl-Gr-L'15] : Cell probe lower bounds for OMv problem over very large finite fields F, space usage S = min (n log |F|, n^2) when $|F|=n^{\Omega(1)}$, S=O(n).

OMv Lower Bounds

[Henzinger - Krinninger - Nanongkai – Saranurak 2015]:
Most BMM lower bounds hold for non-combinatorial algorithms as well, under the OMv Conjecture.

<table>
<tr><th>Problem</th><th>(combinatorial) Upper bound</th><th>(BMM, OMv) Lower bound</th><th>(3-SUM) Lower bound</th></tr>
<tr><td>#SSR</td><td rowspan="4">O(m)</td><td rowspan="4">m</td><td rowspan="5">$m^{1/3}$</td></tr>
<tr><td>Strongly Connected Components</td></tr>
<tr><td>s,t-Reach</td></tr>
<tr><td>Maximum Matching</td></tr>
<tr><td>Approximate Diameter</td><td>O(mn)</td><td>n</td></tr>
</table>

OMv Lower Bounds

[Henzinger - Krinninger - Nanongkai – Saranurak 2015]:
Most BMM lower bounds hold for non-combinatorial algorithms as well, under the OMv Conjecture.

<table>
<tr><th>Problem</th><th>(combinatorial) Upper bound</th><th>(BMM, OMv) Lower bound</th><th>(3-SUM) Lower bound</th></tr>
<tr><td>#SSR</td><td rowspan="4">O(m)</td><td rowspan="4">m</td><td rowspan="5">$m^{1/3}$</td></tr>
<tr><td>Strongly Connected Components</td></tr>
<tr><td>s,t-Reach</td></tr>
<tr><td>Maximum Matching</td></tr>
<tr><td>Approximate Diameter</td><td>O(mn)</td><td>n</td></tr>
</table>

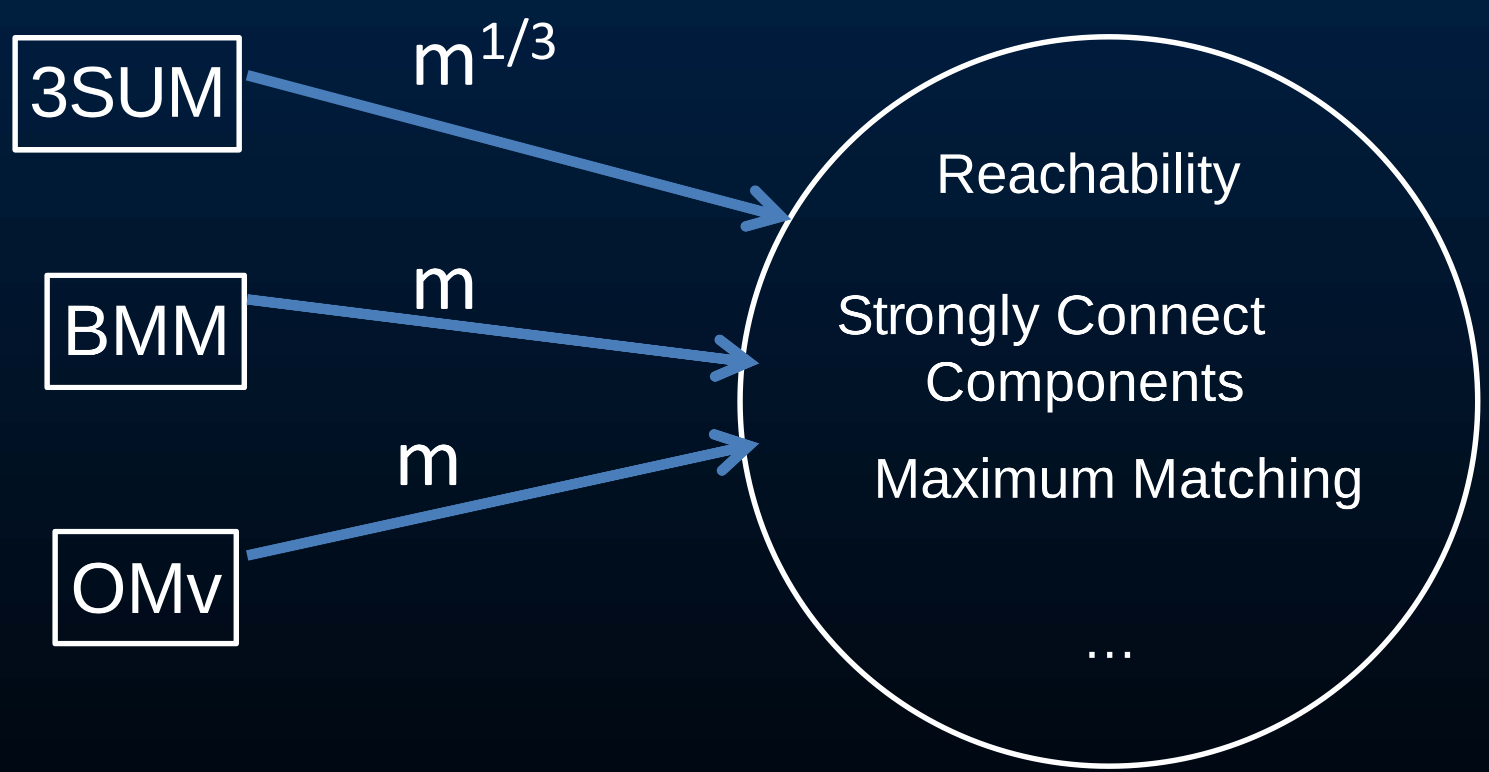
What about diameter? Another conjecture?

SETH / OVC Lower Bounds

[A-VW FOCS 14] OVC, SETH imply very high lower bounds!

SETH: "For all $\varepsilon>0$, there's a k s.t. k-SAT cannot be solved in $(2-\varepsilon)^n$ time"

OVC: "Checking if a set of n vectors over $\{0,1\}^d$ contains an orthog. pair requires $n^{2-o(1)}$ poly(d) time"

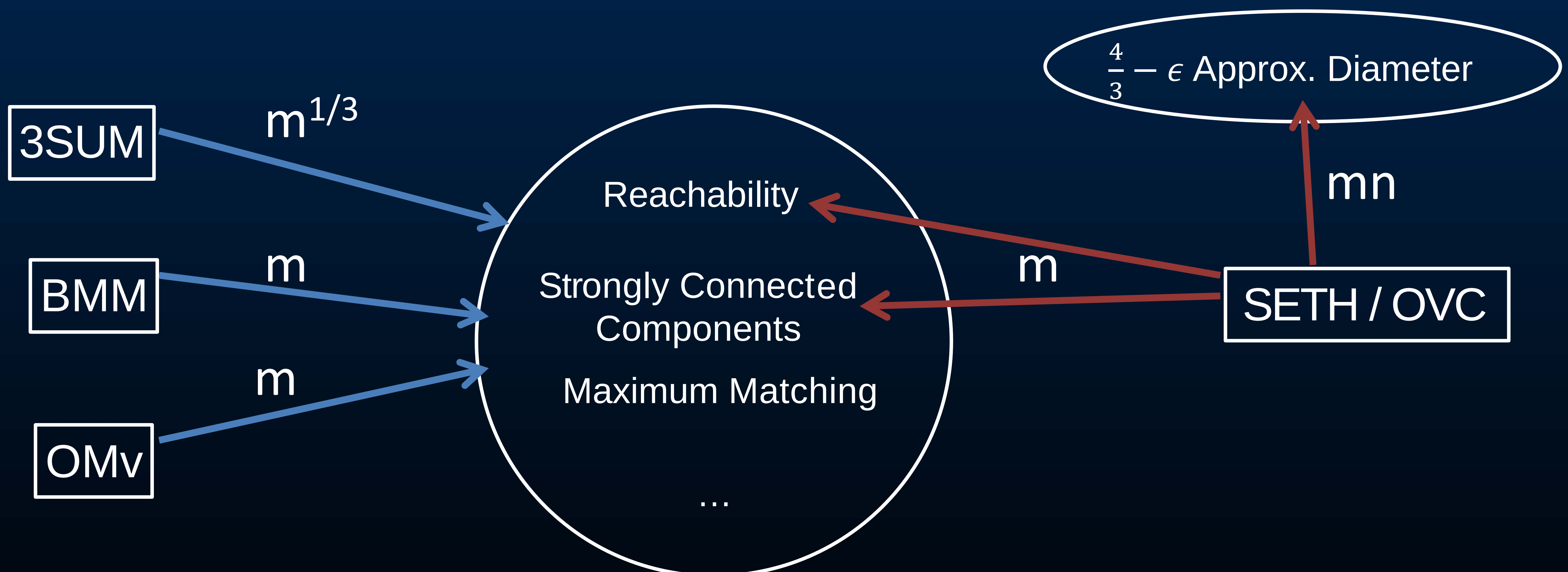


SETH / OVC Lower Bounds

[A-VW FOCS 14] OVC, SETH imply very high lower bounds!

SETH: "For all ε>0, there's a k s.t. k-SAT cannot be solved in $(2-\varepsilon)^n$ time"

OVC: "Checking if a set of n vectors over $\{0,1\}^d$ contains an orthog. pair requires $n^{2-o(1)}$ poly(d) time"

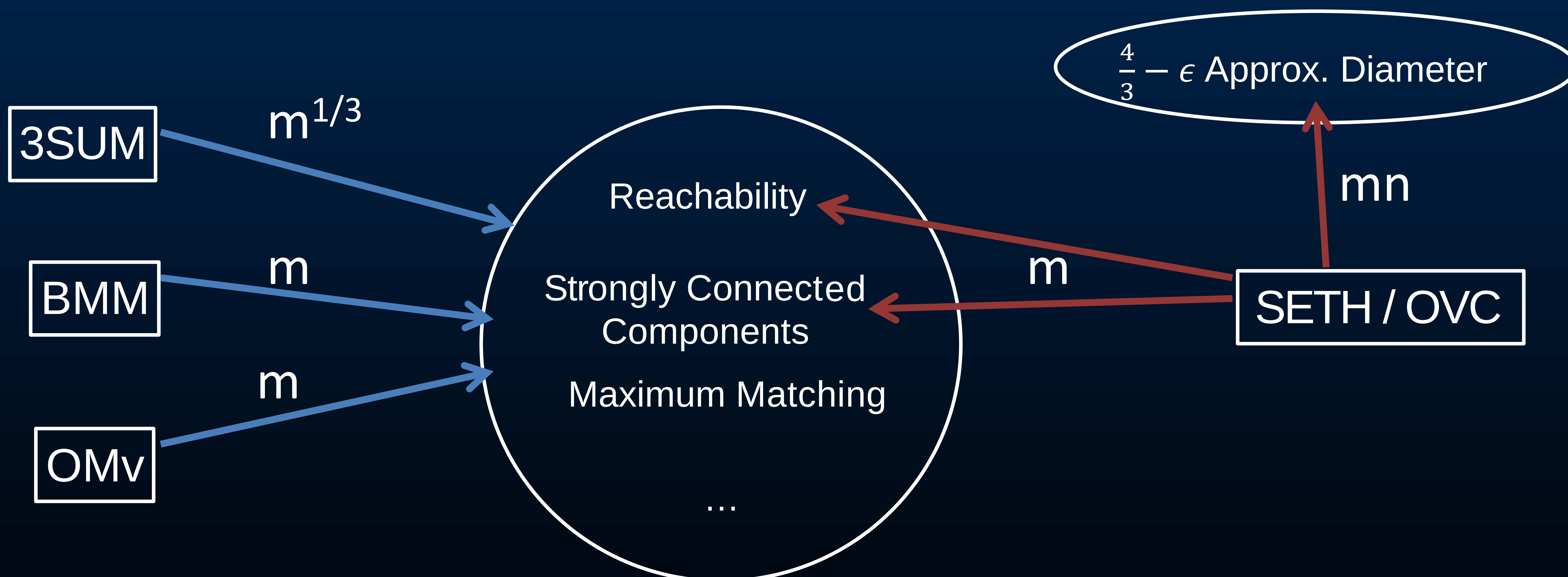


SETH / OVC Lower Bounds

[A-VW FOCS 14] OVC, SETH imply very high lower bounds!

SETH: "For all ε>0, there's a k s.t. k-SAT cannot be solved in $(2-\varepsilon)^n$ time"

OVC: "Checking if a set of n vectors over $\{0,1\}^d$ contains an orthog. pair requires $n^{2-o(1)}$ poly(d) time"



Different conjectures are better for explaining different barriers

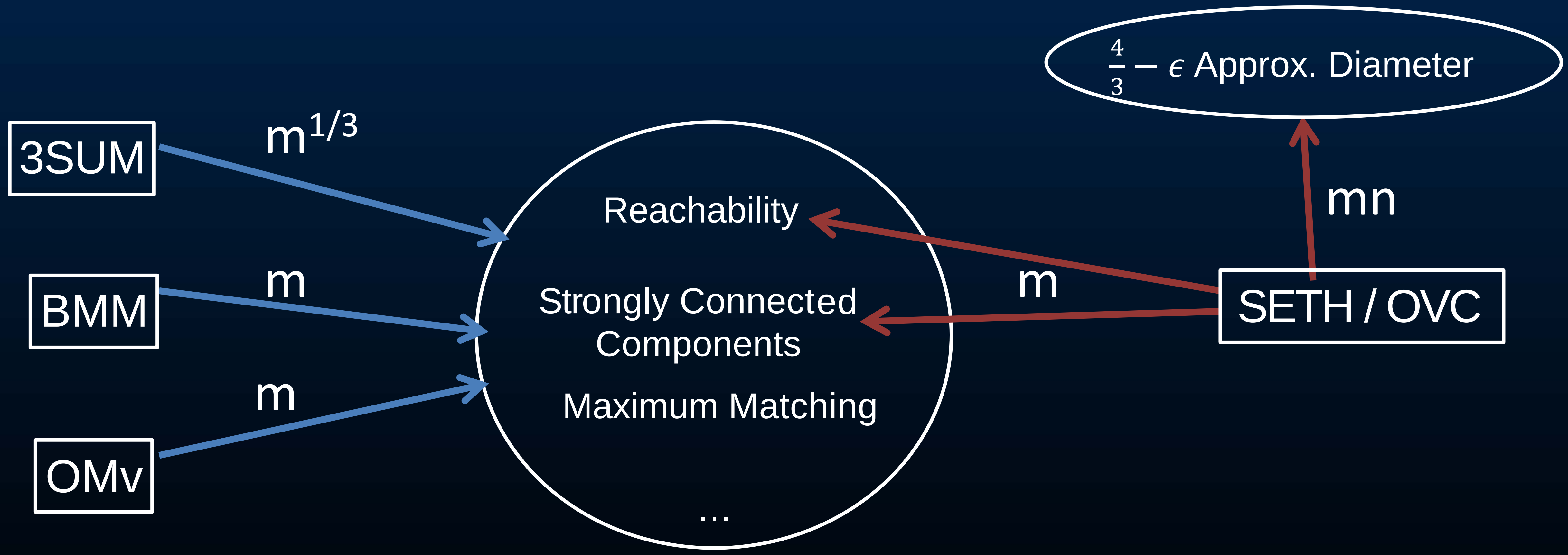
APSP Lower Bounds

[Abboud-VW FOCS 14']

The APSP conjecture implies tight lower bounds for some weighted problems.

The APSP conjecture:

"No $O(n^{3-\epsilon})$ time algorithm for All-Pairs-Shortest-Paths"



Different conjectures are better for explaining different barriers

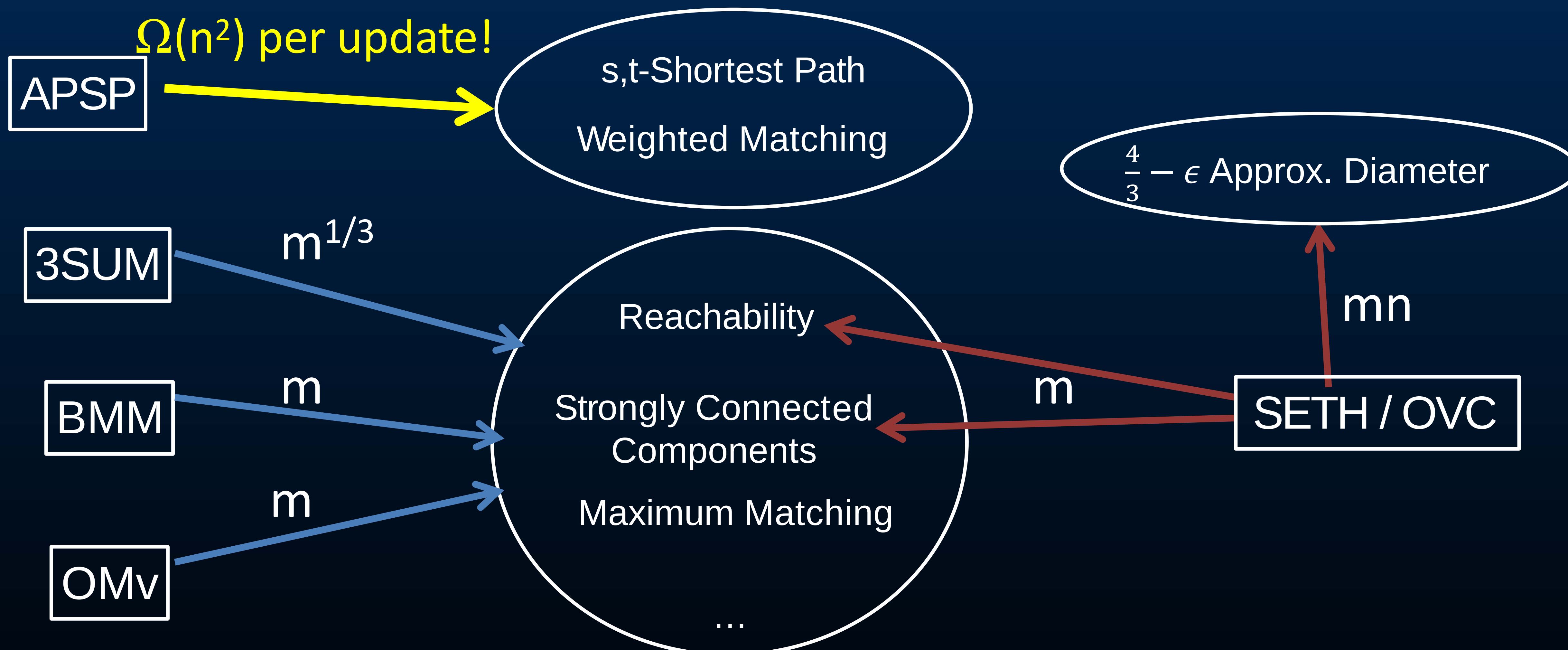
APSP Lower Bounds

[Abboud-VW FOCS 14']

The APSP conjecture implies tight lower bounds for some weighted problems.

The APSP conjecture:

"No $O(n^{3-\epsilon})$ time algorithm for All-Pairs-Shortest-Paths"



Different conjectures are better for explaining different barriers

Plan

➡ Overview of some lower bounds for dynamic problems

➡ Simple and powerful proofs

- Single Source Reachability
- #ss-Reach
- Strongly Connected Components
- Diameter
- s-t Shortest Path

Dynamic single source reachability

Single source reachability: given a source s, which nodes can s reach? O(m+n) time, DFS

Dynamic single source reachability

Single source reachability: given a source s, which nodes can s reach? O(m+n) time, DFS

Dynamic #SS-reachability:
Updates: delete/insert edge
Query: how many nodes can s reach?

Dynamic single source reachability

Single source reachability: given a source s, which nodes can s reach?   O(m+n) time, DFS

Dynamic #SS-reachability:
Updates: delete/insert edge
Query: how many nodes can s reach?

Trivial solution:

Dynamic single source reachability

Single source reachability: given a source s, which nodes can s reach? O(m+n) time, DFS

Dynamic #SS-reachability:
Updates: delete/insert edge
Query: how many nodes can s reach?

Trivial solution:
O(m + n) time updates or O(m + n) time queries

Dynamic single source reachability

Single source reachability: given a source s, which nodes can s reach?   O(m+n) time, DFS

Dynamic #SS-reachability:
Updates: delete/insert edge
Query: how many nodes can s reach?

Trivial solution:
O(m + n) time updates or O(m + n) time queries
[Sankowski'04]: $O(n^{1.495})$ update and query time

Dynamic single source reachability

Single source reachability: given a source s, which nodes can s reach? O(m+n) time, DFS

Dynamic #SS-reachability:
Updates: delete/insert edge
Query: how many nodes can s reach?

Trivial solution:
O(m + n) time updates or O(m + n) time queries
[Sankowski'04]: $O(n^{1.495})$ update and query time

No nontrivial solution for sparse graphs!

Thm: $O(m^{1 - \varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2 - \varepsilon'})$ time and hence SETH is false.

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

Reduction from OV, vector dimension d

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

Reduction from OV, vector dimension d

Preprocessing: create a special graph G

Then a **stage** for each vector v in OV instance:

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

Reduction from OV, vector dimension d

Preprocessing: create a special graph G

Then a **stage** for each vector v in OV instance:

(1) Insert ≤ d edges into G

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

Reduction from OV, vector dimension d

Preprocessing: create a special graph G

Then a **stage** for each vector v in OV instance:

(1) Insert ≤ d edges into G

(2) Query #SS-reach

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

Reduction from OV, vector dimension d

Preprocessing: create a special graph G

Then a **stage** for each vector v in OV instance:

(1) Insert ≤ d edges into G
(2) Query #SS-reach
(3) Remove the ≤ d inserted edges

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

Reduction from OV, vector dimension d

Preprocessing: create a special graph G

Then a **stage** for each vector v in OV instance:

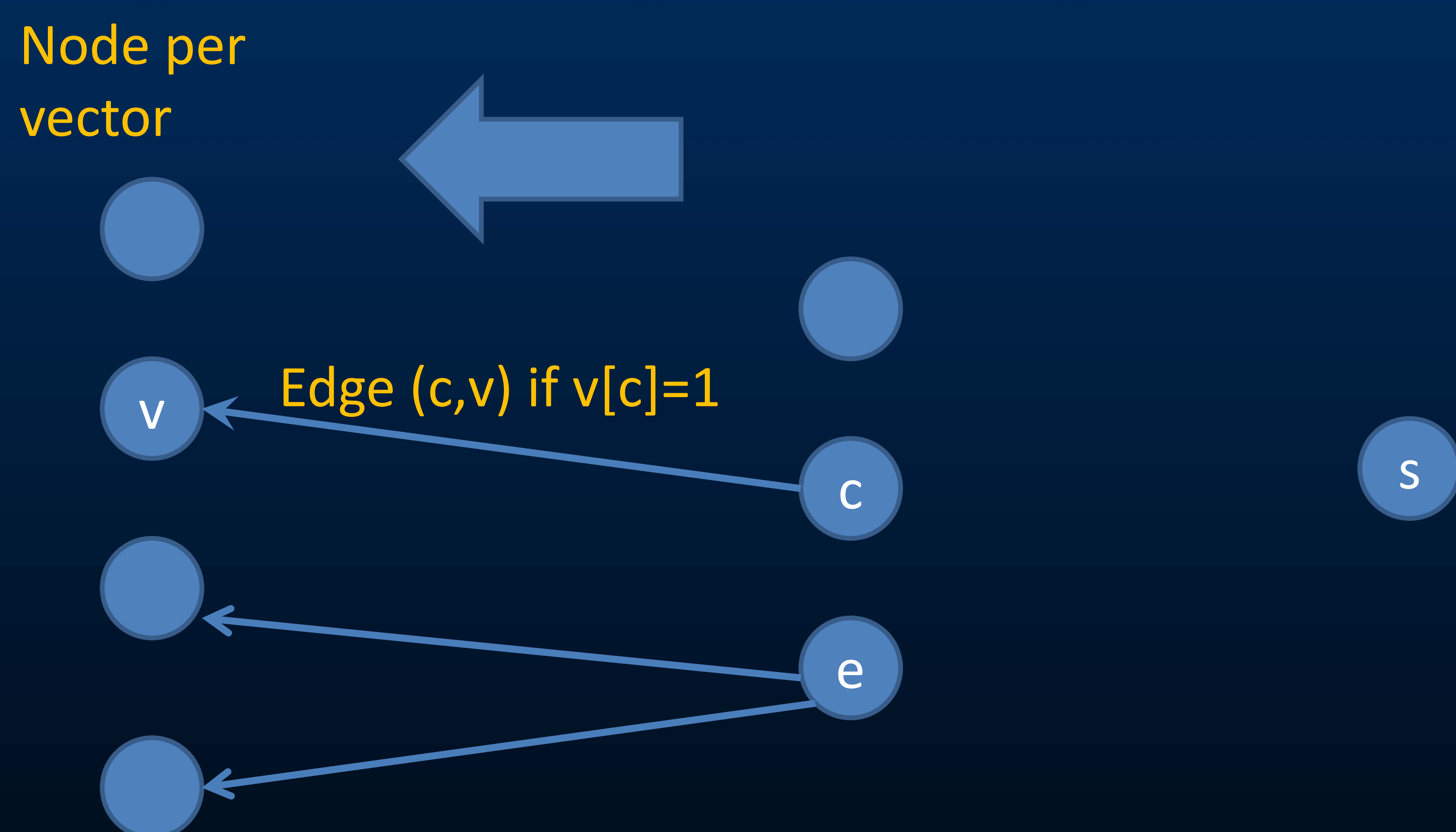
(1) Insert ≤ d edges into G

(2) Query #SS-reach

(3) Remove the ≤ d inserted edges

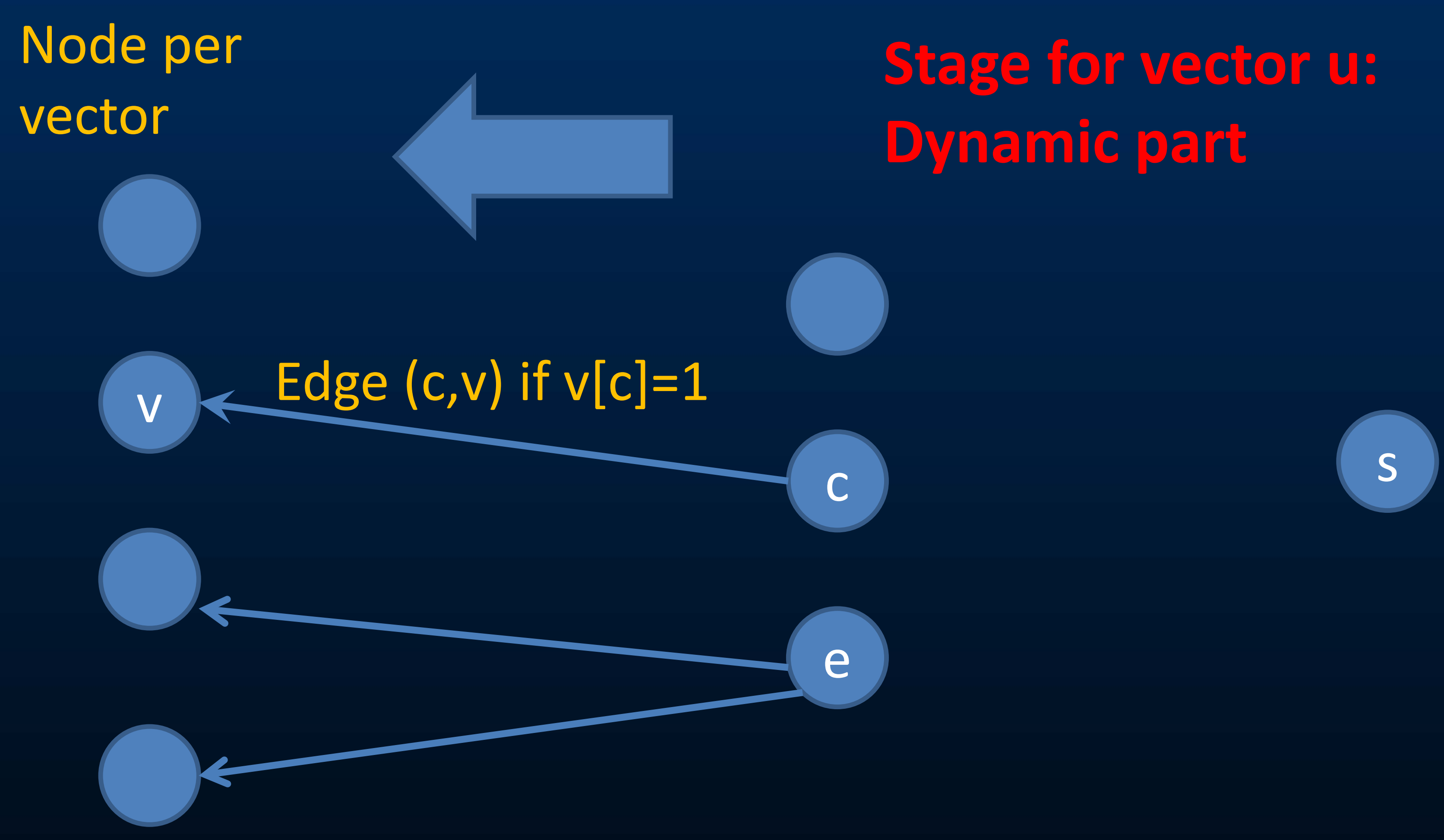
n queries, O(n d) updates

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

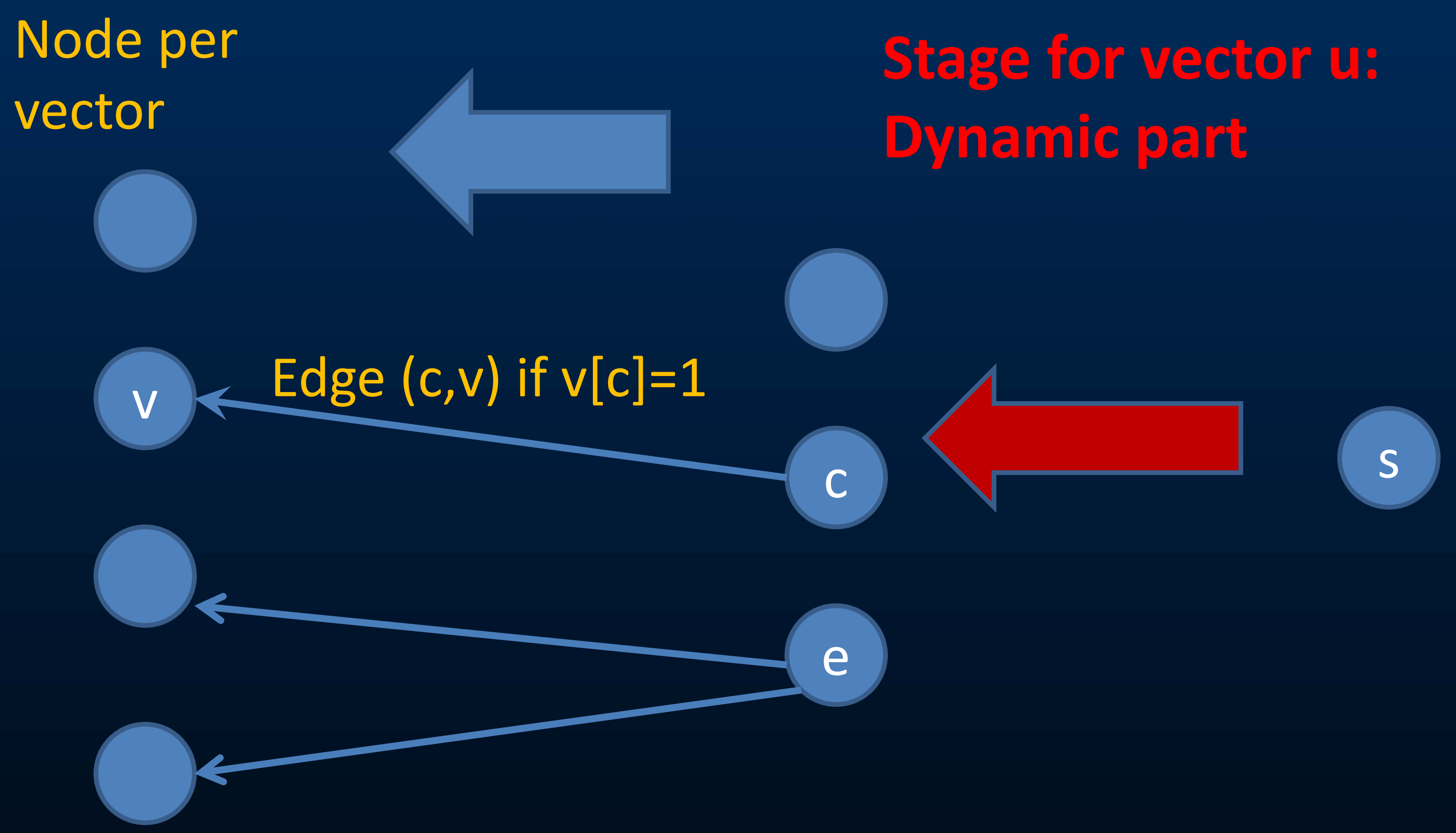


Graph after preprocessing (static)

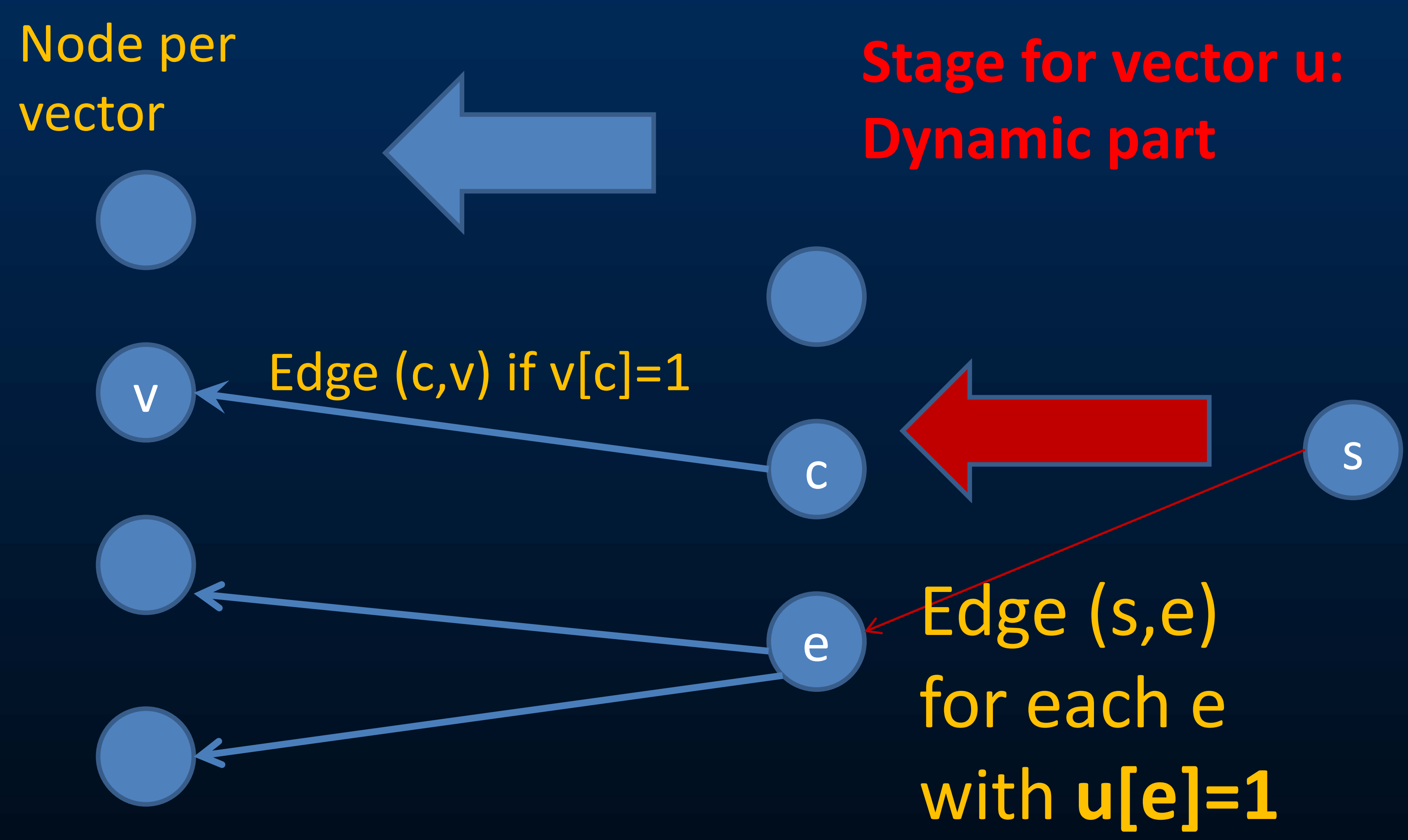
Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.



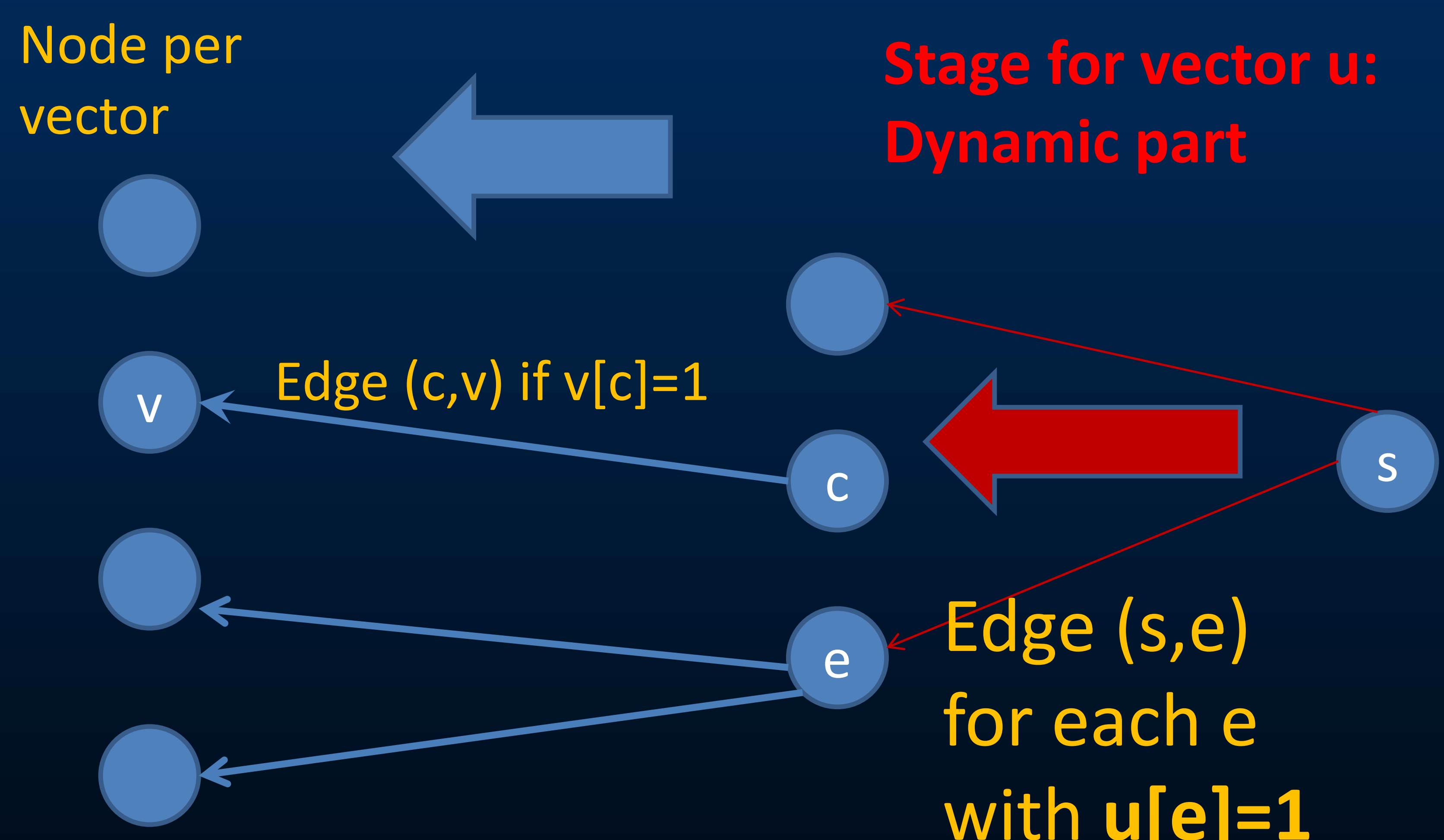
Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.



Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

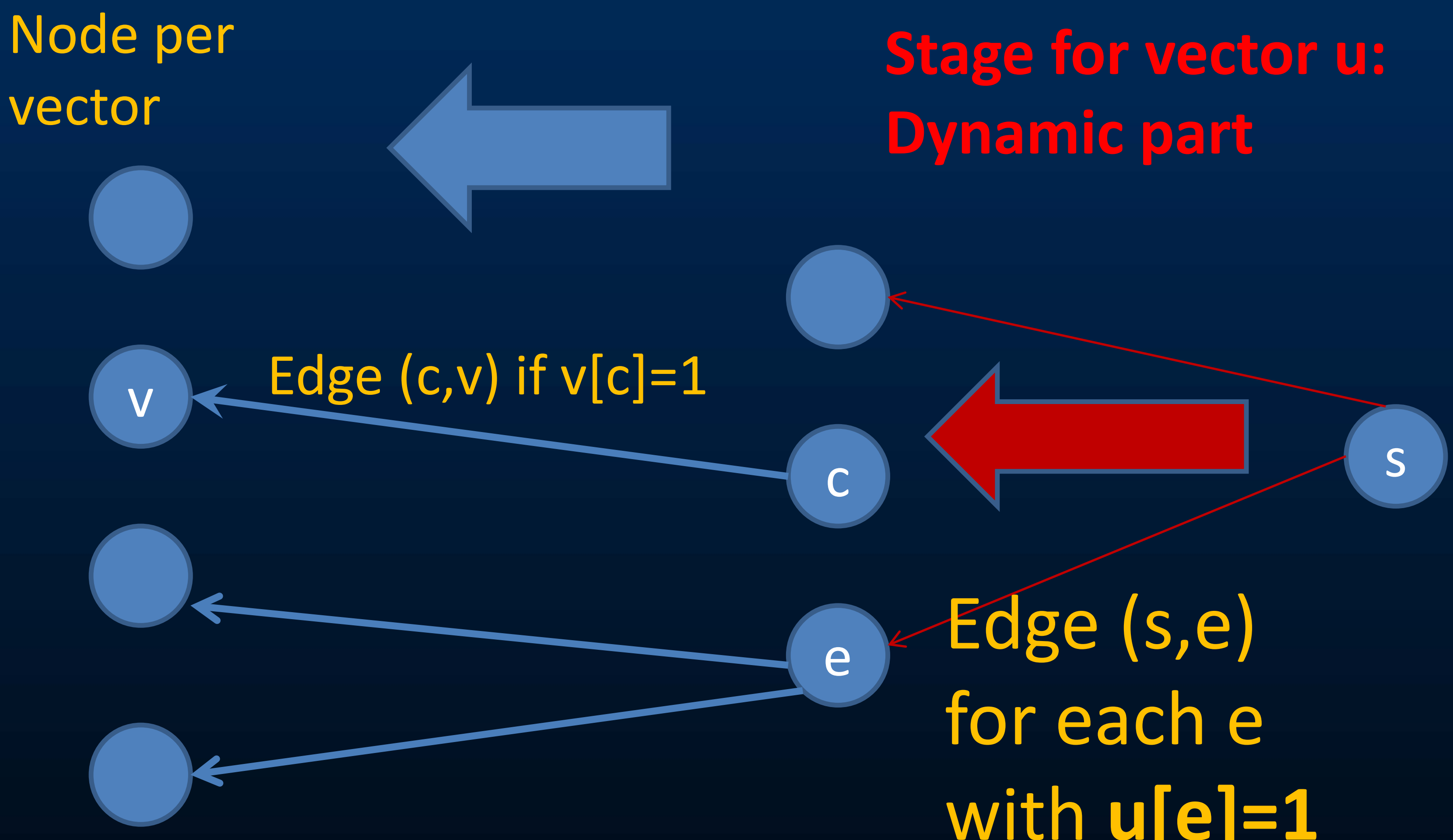


Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.



Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

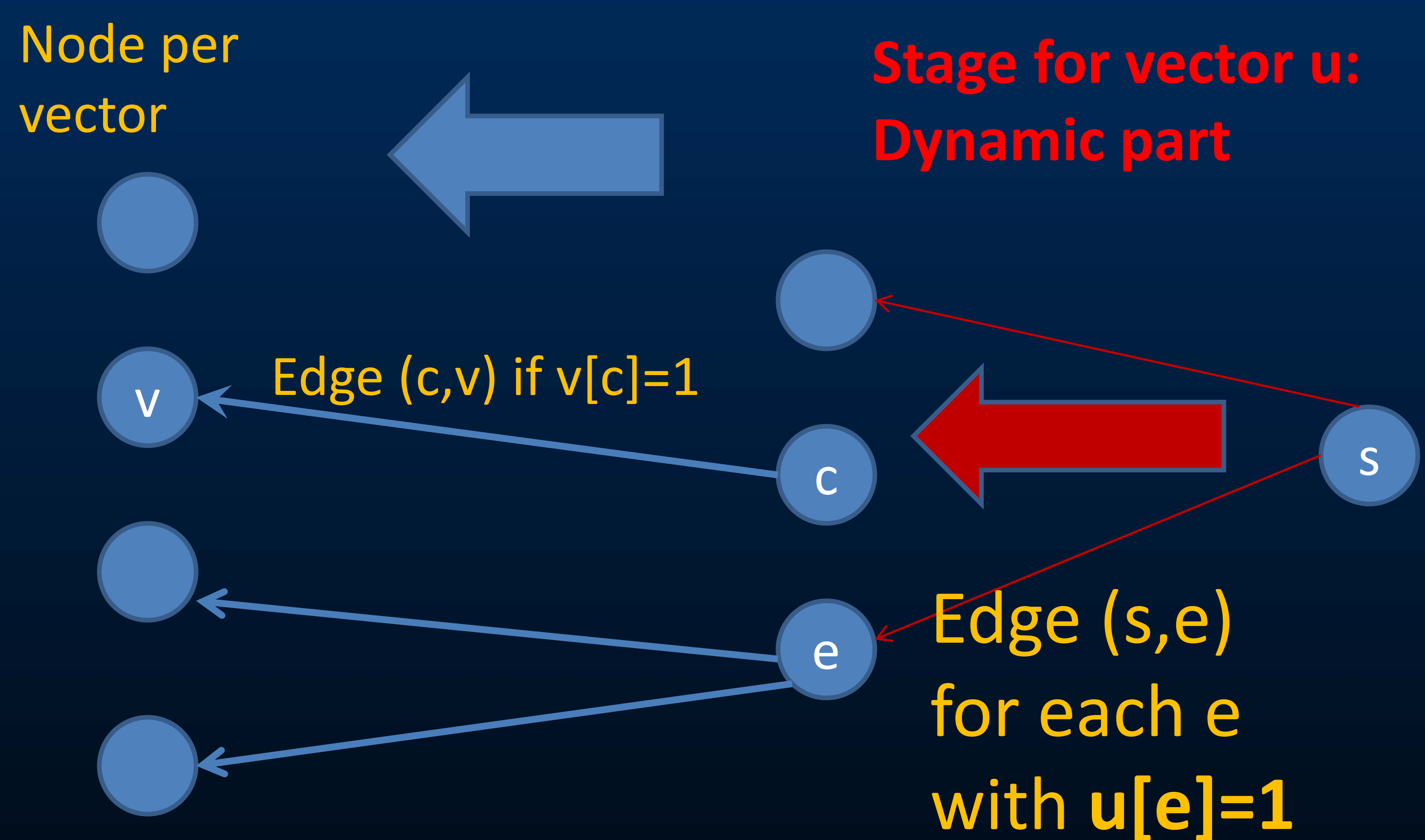
(1) s can reach itself



Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

(1) s can reach itself

(2) s can reach all coords e with **u[e]=1**. Say X such.

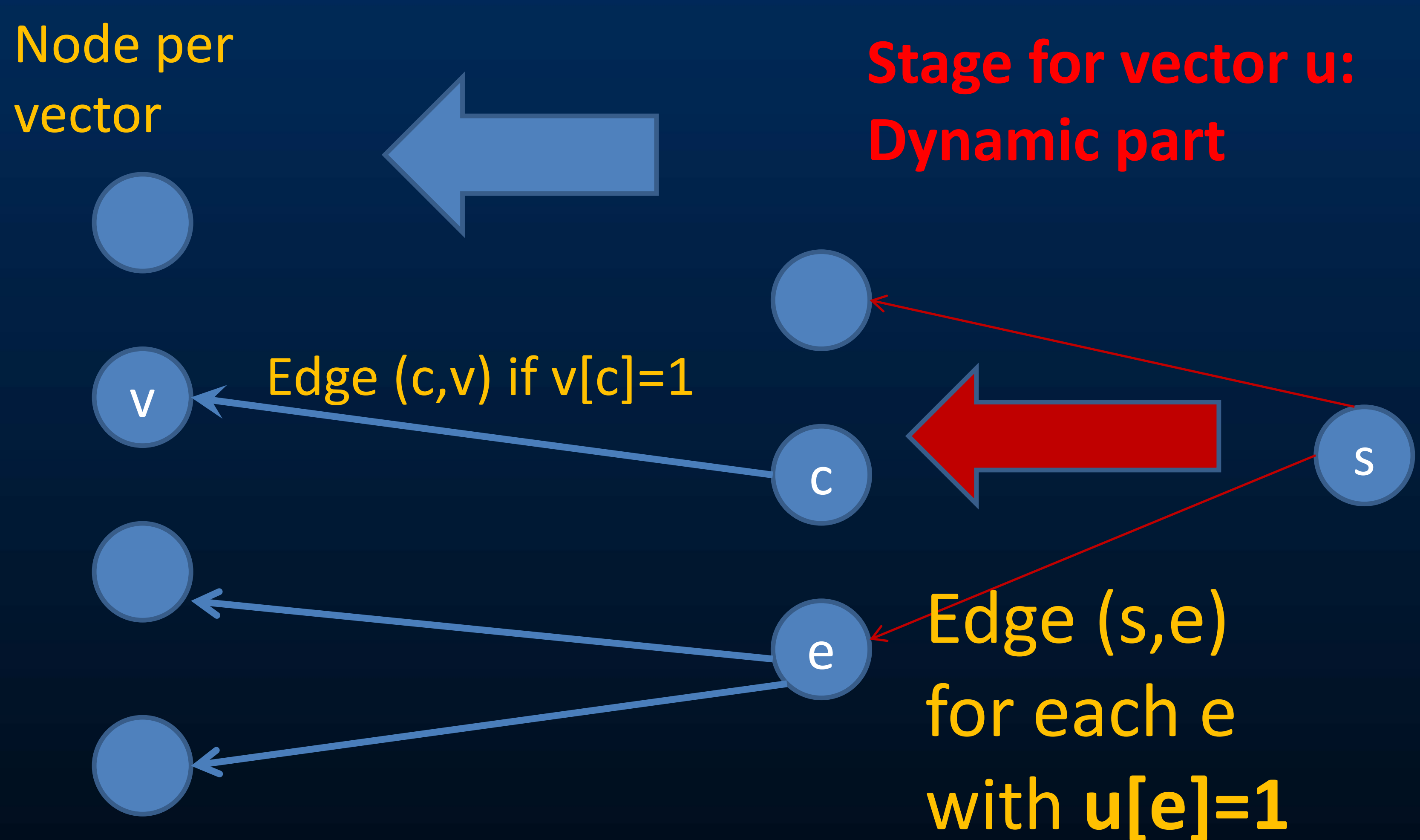


Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

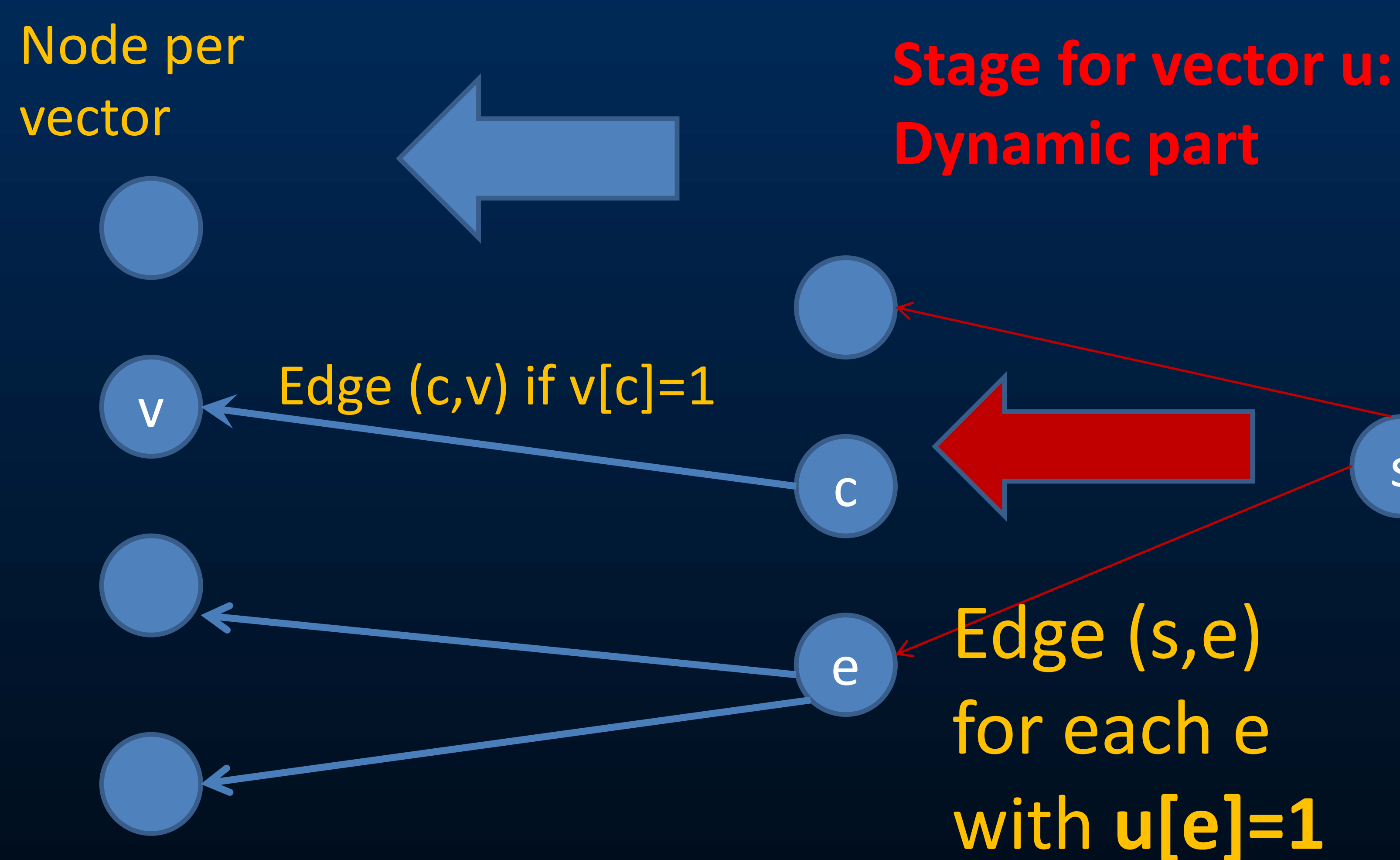
(1) s can reach itself

(2) s can reach all coords e with **u[e]=1**. Say X such.

(3) s can reach all vectors that are not orthog to u



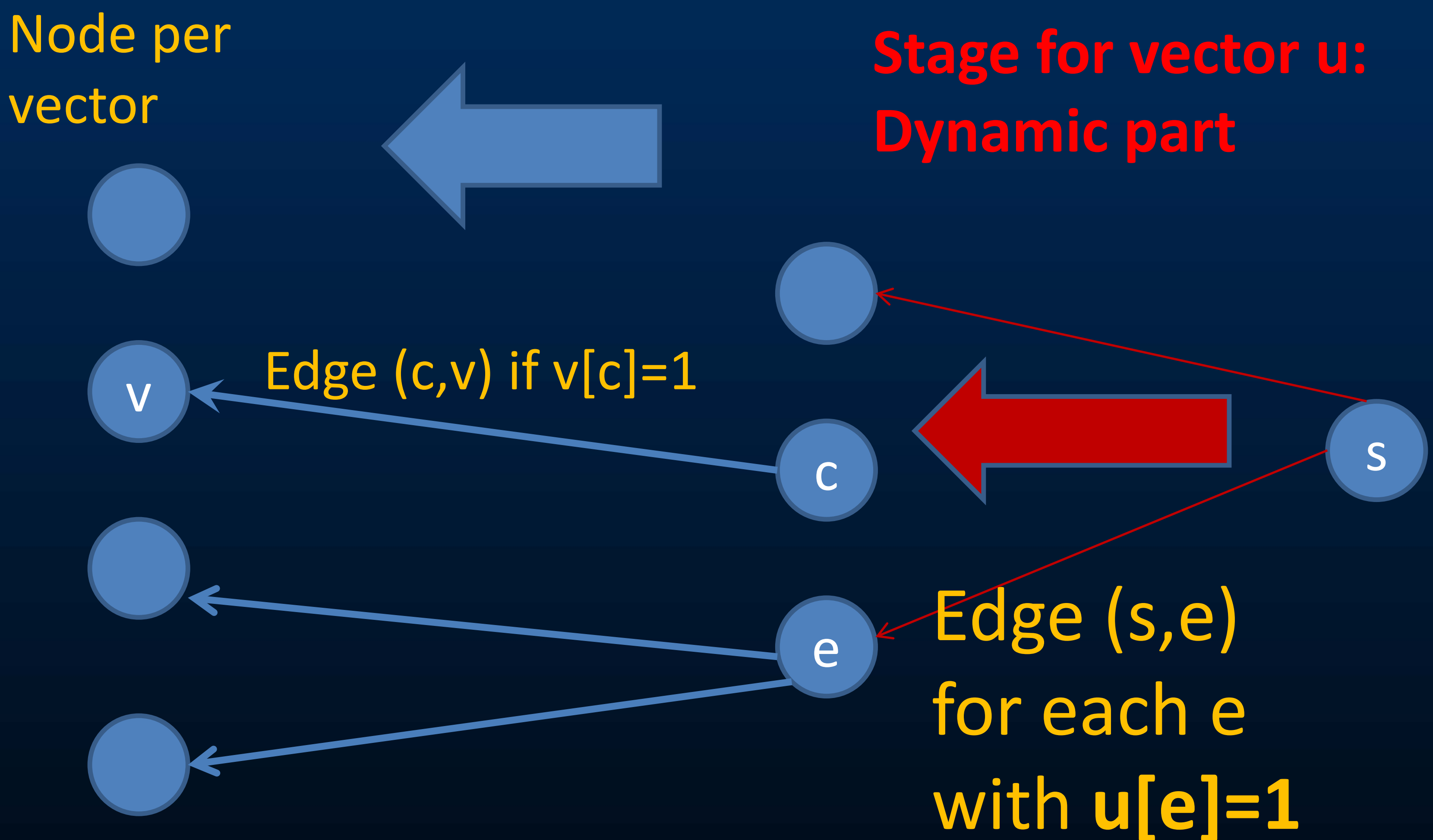
Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.



(1) s can reach itself
(2) s can reach all coords e with **u[e]=1**. Say X such.
(3) s can reach all vectors that are not orthog to u

There is **some v orthog to u** iff the # of reachable nodes from s **is < X + n + 1**

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.

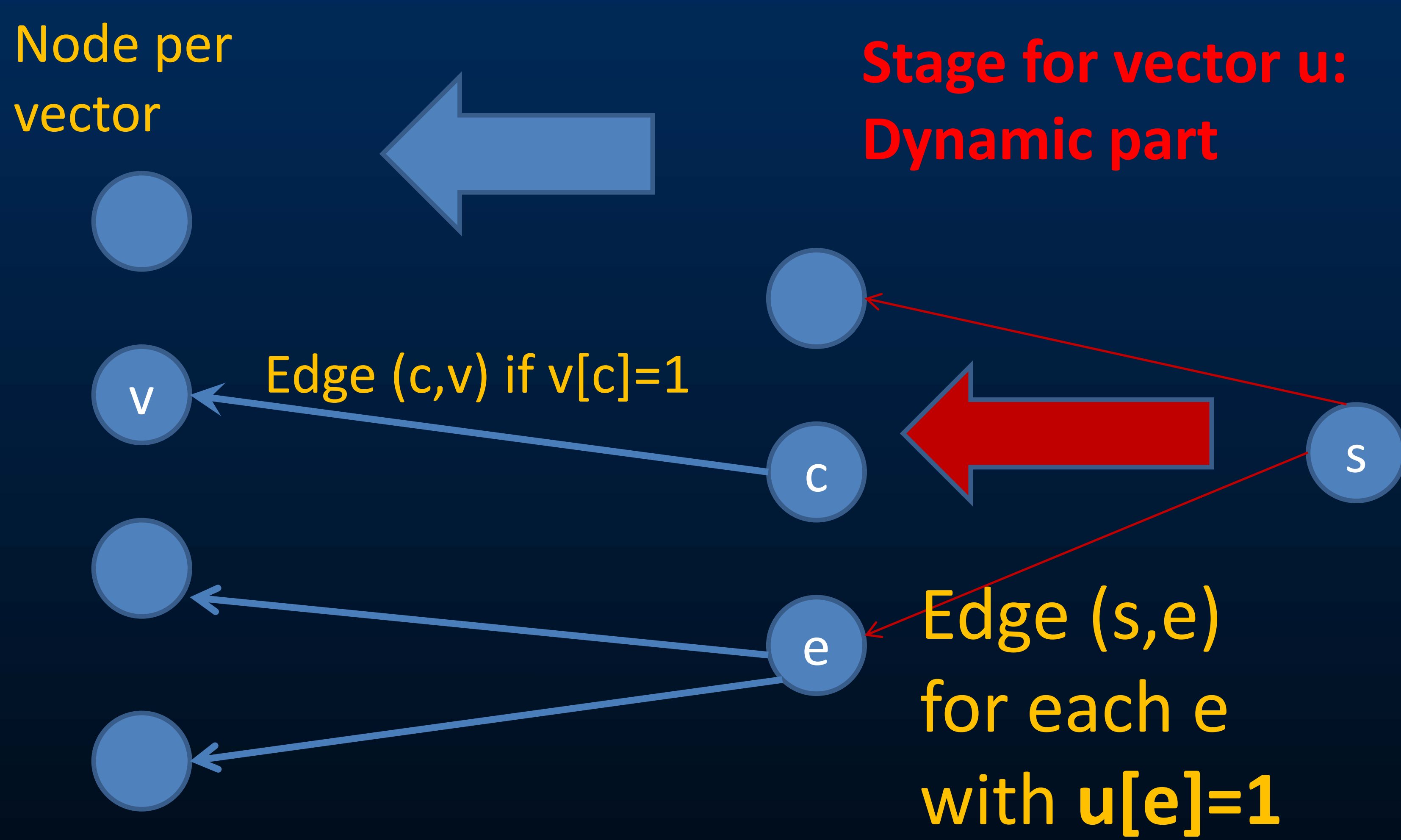


(1) s can reach itself
(2) s can reach all coords e with **u[e]=1**. Say X such.
(3) s can reach all vectors that are not orthog to u

There is **some v orthog to u** iff the # of reachable nodes from s **is < X + n + 1**

m ~ n d

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.



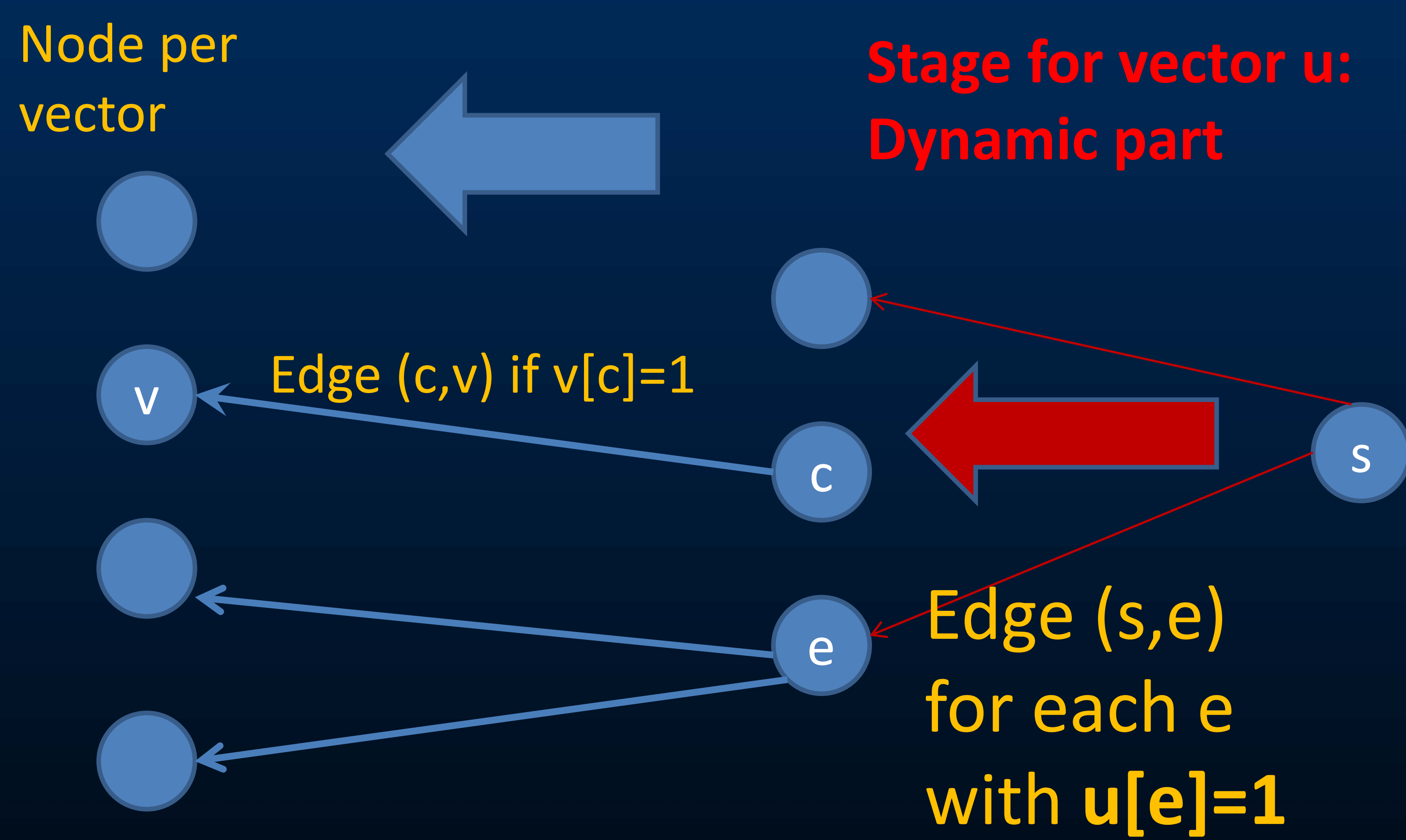
(1) s can reach itself
(2) s can reach all coords e with **u[e]=1**. Say X such.
(3) s can reach all vectors that are not orthog to u

There is **some v orthog to u** iff the # of reachable nodes from s **is < X + n + 1**

m ~ n d

O(n d) updates, n queries

Thm: $O(m^{1-\varepsilon})$ queries and updates for #SS-reach imply OV in $O(n^{2-\varepsilon'})$ time and hence SETH is false.



(1) s can reach itself
(2) s can reach all coords e with **u[e]=1**. Say X such.
(3) s can reach all vectors that are not orthog to u

There is **some v orthog to u** iff the # of reachable nodes from s **is < X + n + 1**

m ~ n d

O(n d) updates, n queries

So $\mathbf{m^{1-o(1)}}$ lower bound from OV and SETH.

Plan

➡ Overview of some lower bounds for dynamic problems

➡ Simple and powerful proofs

- Single Source Reachability
- #ss-Reach
- Strongly Connected Components
- Diameter
- s-t Shortest Path

Dynamic maintenance of SCCs

Strongly connected components:
Can find them in O(m) time in a graph with m edges.

Dynamic algorithms: maintain graph G under
insert(u,v), delete(u,v) supporting:

Dynamic maintenance of SCCs

Strongly connected components:
Can find them in O(m) time in a graph with m edges.

Dynamic algorithms: maintain graph G under
insert(u,v), delete(u,v) supporting:

- **query1**(u,v): are u and v in the same SCC?

Dynamic maintenance of SCCs

Strongly connected components:
Can find them in O(m) time in a graph with m edges.

Dynamic algorithms: maintain graph G under
insert(u,v), delete(u,v) supporting:

- **query1**(u,v): are u and v in the same SCC?
- **query2**: how many SCCs does G have?

Dynamic maintenance of SCCs

Strongly connected components:
Can find them in O(m) time in a graph with m edges.

Dynamic algorithms: maintain graph G under
insert(u,v), delete(u,v) supporting:

- **query1**(u,v): are u and v in the same SCC?
- **query2**: how many SCCs does G have?

(All known algorithms for query1 also solve query2.)

Dynamic SCC: prior work

If only inserts or only deletes allowed, can answer both types of queries in constant time and

update time is “small”.

Dynamic SCC: prior work

If only inserts or only deletes allowed, can answer both types of queries in constant time and

update time is "small".

T = Sum over all m update times.

Dynamic SCC: prior work

If only inserts or only deletes allowed, can answer both types of queries in constant time and
update time is "small".
T = Sum over all m update times.

Inserts only:
BFGT'09: T ~ $n^2 \log n$,
HKMST.'08: T ~ $\min\{m^{3/2}, mn^{2/3}\}$
Bernstein, Chechik'18: T~ $mn^{1/2}$, rand.
Bhattacharya Kulkarni'20: T~ $m^{4/3}$, rand.

Dynamic SCC: prior work

If only inserts or only deletes allowed, can answer both types of queries in constant time and
update time is "small".
T = Sum over all m update times.

Inserts only:
BFGT'09: T ~ $n^2 \log n$,
HKMST.'08: T ~ $\min\{m^{3/2}, mn^{2/3}\}$
Bernstein, Chechik'18: T~ $mn^{1/2}$, rand.
Bhattacharya Kulkarni'20: T~ $m^{4/3}$, rand.
Deletes only: R.Z.'02, Lacki'11, Roditty'12: T ~ mn.

Dynamic SCC: prior work

If only inserts or only deletes allowed, can answer both types of queries in constant time and
update time is "small".
T = Sum over all m update times.

Inserts only:
BFGT'09: T ~ $n^2 \log n$,
HKMST.'08: T ~ $\min\{m^{3/2}, mn^{2/3}\}$
Bernstein, Chechik'18: T~ $mn^{1/2}$, rand.
Bhattacharya Kulkarni'20: T~ $m^{4/3}$, rand.
Deletes only: R.Z.'02, Lacki'11, Roditty'12: T ~ mn.

Amortized update time is
n for deletes only, $\min(m^{1/3}, n^2/m)$ for inserts only.

Fully dynamic SCC

If both inserts and deletes allowed: best known solution is to recompute SCCs after each update!

Fully dynamic SCC

If both inserts and deletes allowed: best known solution is to recompute SCCs after each update!

Thm: Under OVC, any fully dynamic algorithm that can answer queries "Is the number of SCCs>2?" requires $m^{1-o(1)}$ update or query time.

Fully dynamic SCC

If both inserts and deletes allowed: best known solution is to recompute SCCs after each update!

Thm: Under OVC, any fully dynamic algorithm that can answer queries "Is the number of SCCs>2?" requires $m^{1-o(1)}$ update or query time.

If SETH is true, might as well recompute the SCCs after each update!

Dynamic #SCC>2 is hard

Reduce from OV with vector dimension d

For each vector v, have a stage:

Dynamic #SCC>2 is hard

Reduce from OV with vector dimension d

For each vector v, have a stage:

- Insert ≤ d edges

Dynamic #SCC>2 is hard

Reduce from OV with vector dimension d

For each vector v, have a stage:

- Insert ≤ d edges
- Query #SCC>2.

Dynamic #SCC>2 is hard

Reduce from OV with vector dimension d

For each vector v, have a stage:

- Insert ≤ d edges
- Query #SCC>2.
  - *If #SCC>2 is yes, return that some u is orthogonal to v.*

Dynamic #SCC>2 is hard

Reduce from OV with vector dimension d

For each vector v, have a stage:

- Insert ≤ d edges
- Query #SCC>2.
    - *If #SCC>2 is yes, return that some u is orthogonal to v.*
- Delete the ≤ d edges

Dynamic #SCC>2 is hard

Reduce from OV with vector dimension d

For each vector v, have a stage:

- Insert ≤ d edges
- Query #SCC>2.
  - *If #SCC>2 is yes, return that some u is orthogonal to v.*
- Delete the ≤ d edges

O(n d) updates, n queries, m ~ nd edges

Dynamic #SCC>2 is hard

Reduce from OV with vector dimension d

For each vector v, have a stage:

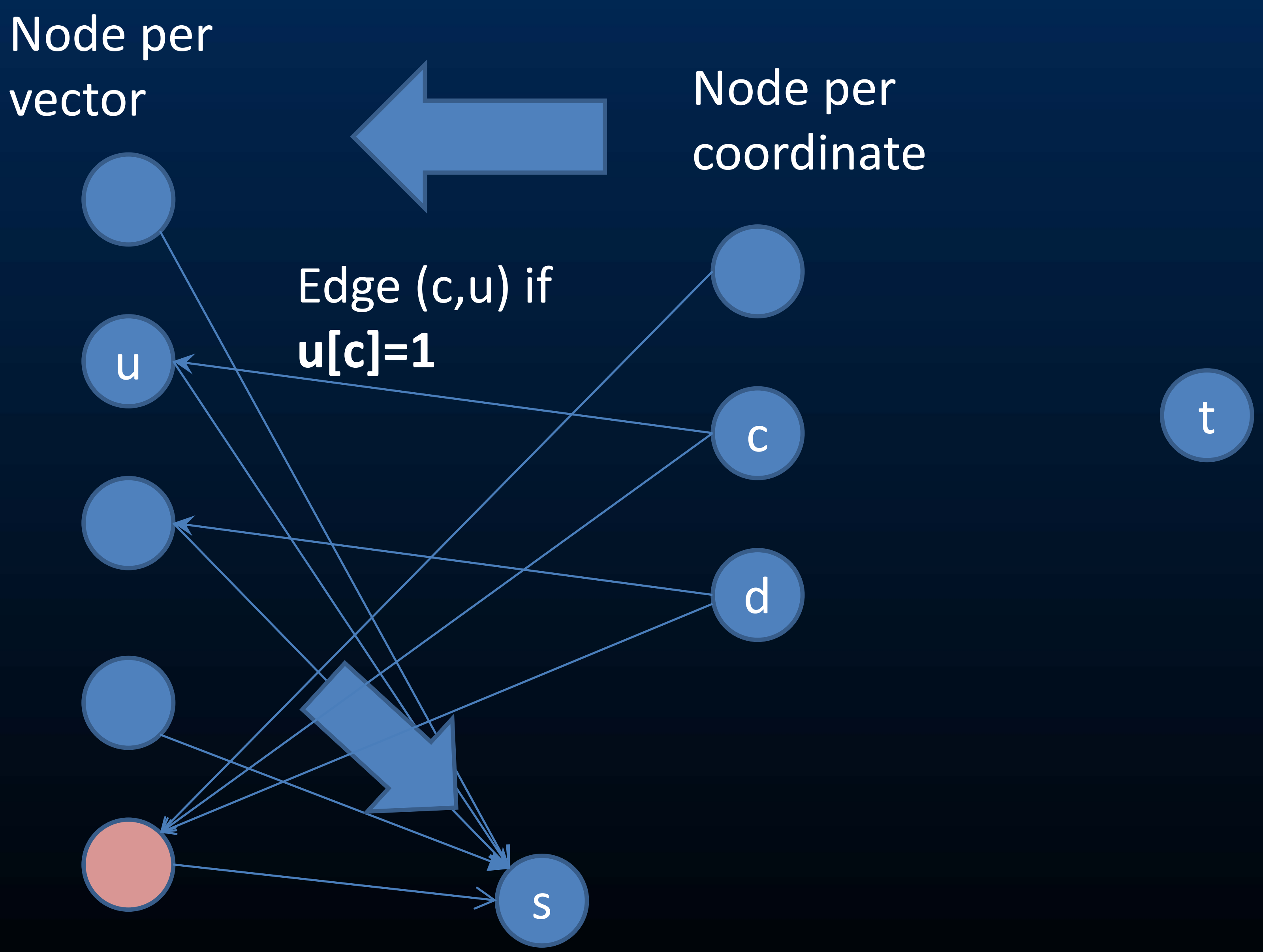
- Insert ≤ d edges
- Query #SCC>2.
  - *If #SCC>2 is yes, return that some u is orthogonal to v.*
- Delete the ≤ d edges

O(n d) updates, n queries, m ~ nd edges

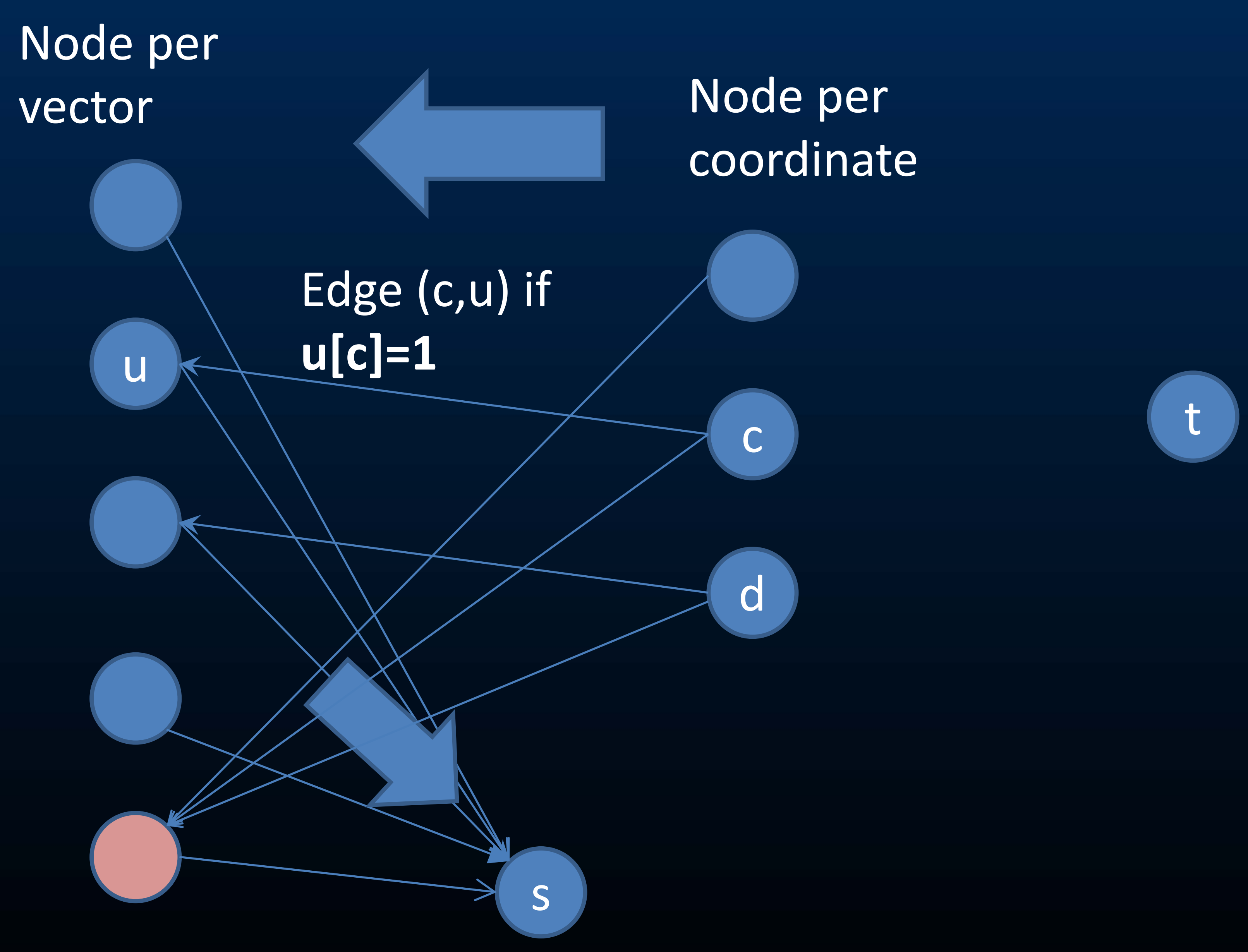
- OV/SETH lower bound of $m^{1-o(1)}$ for query or update

Dynamic #SCC>2 is hard

Graph after preprocessing

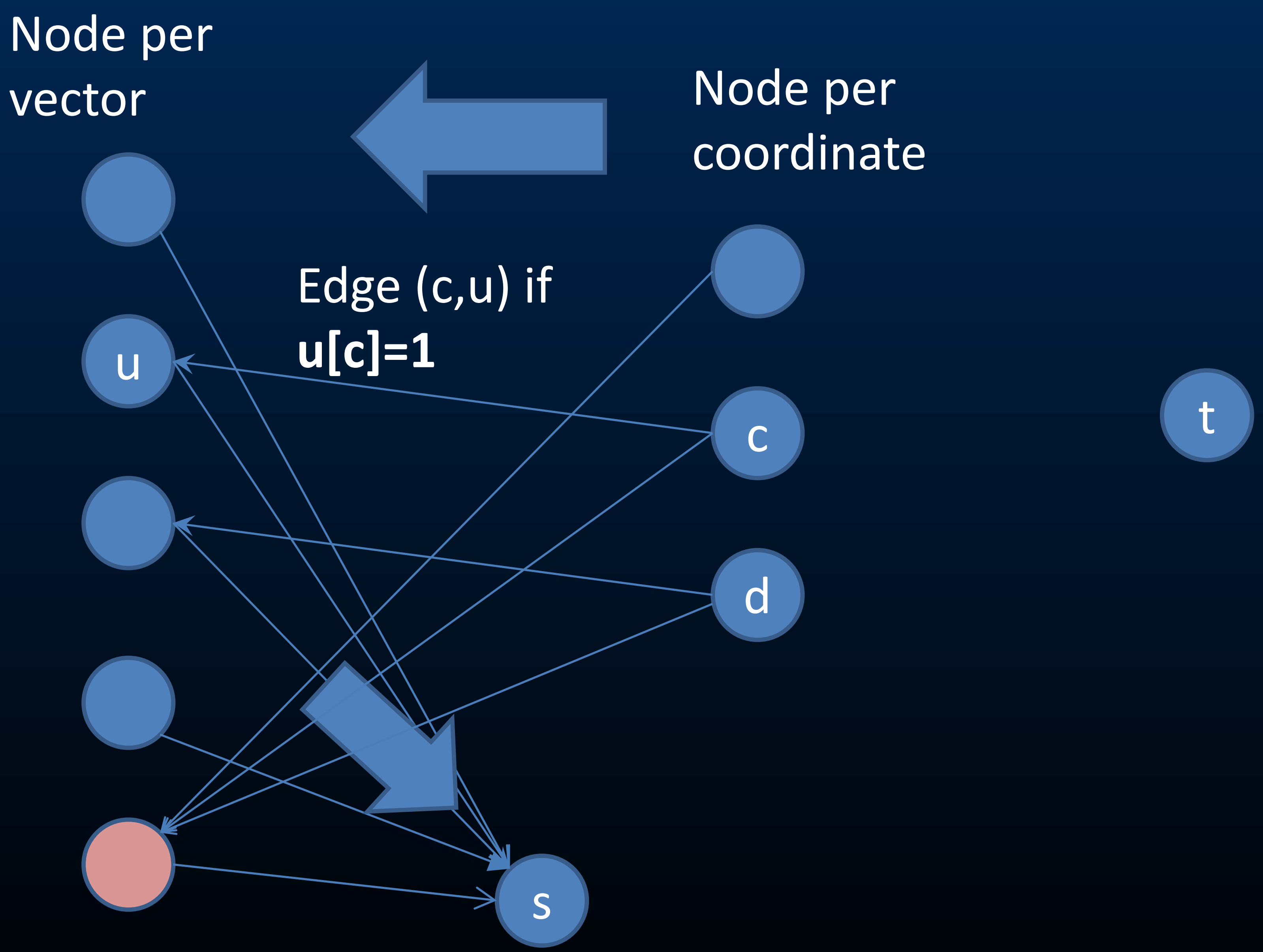


Dynamic #SCC>2 is hard



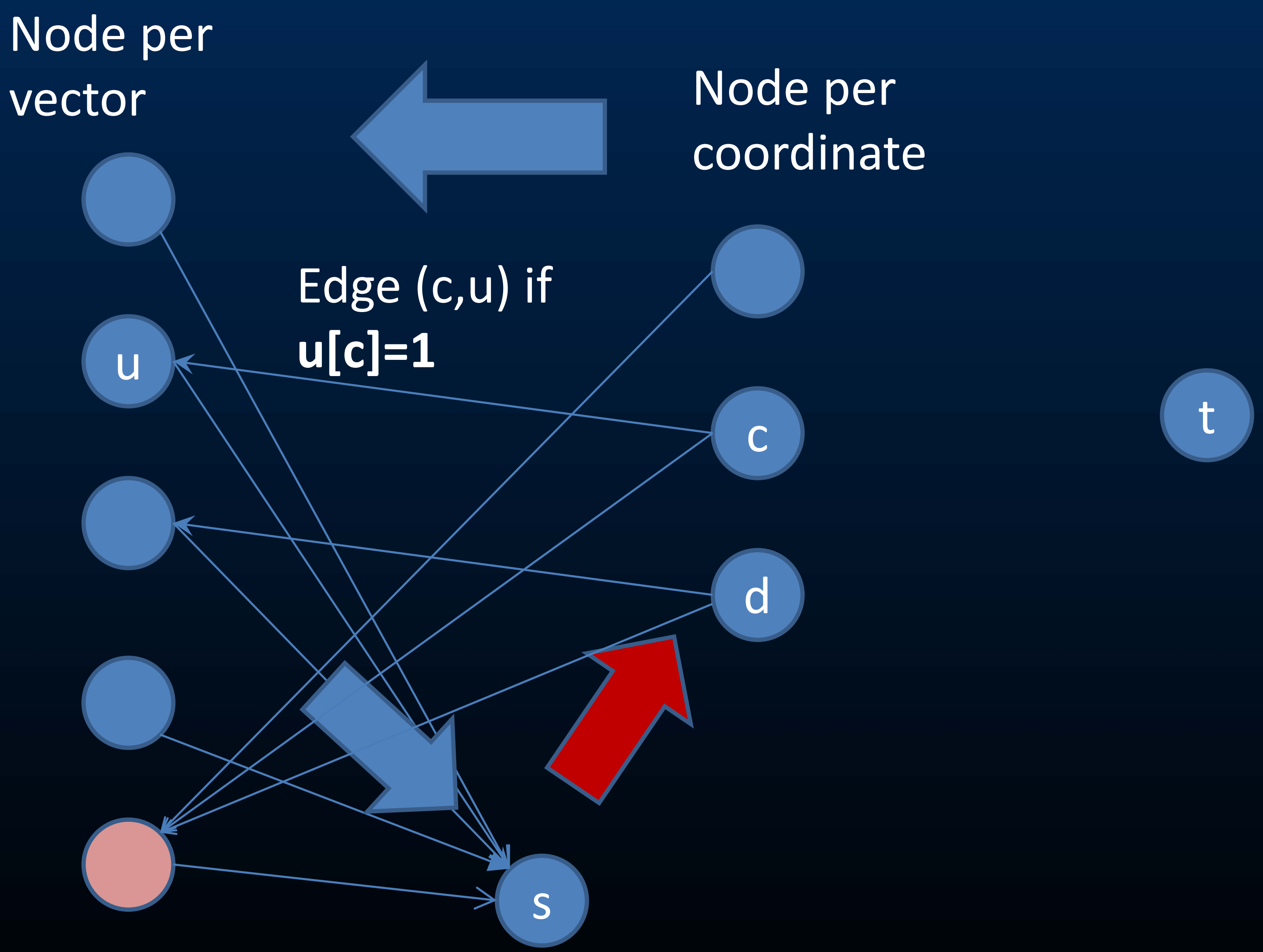
Dynamic #SCC>2 is hard

Stage for vector v (updates red):



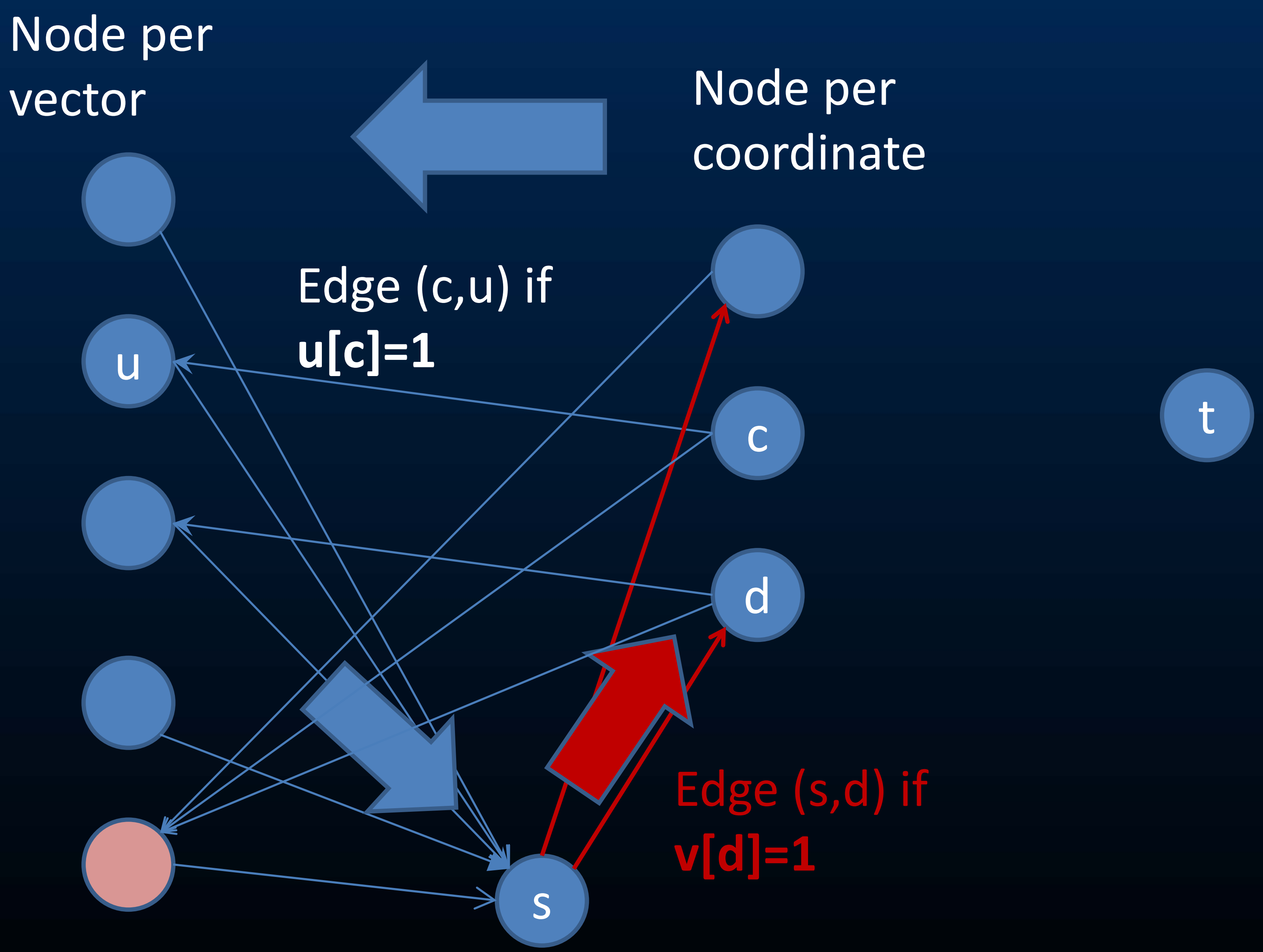
Dynamic #SCC>2 is hard

Stage for vector v (updates red):



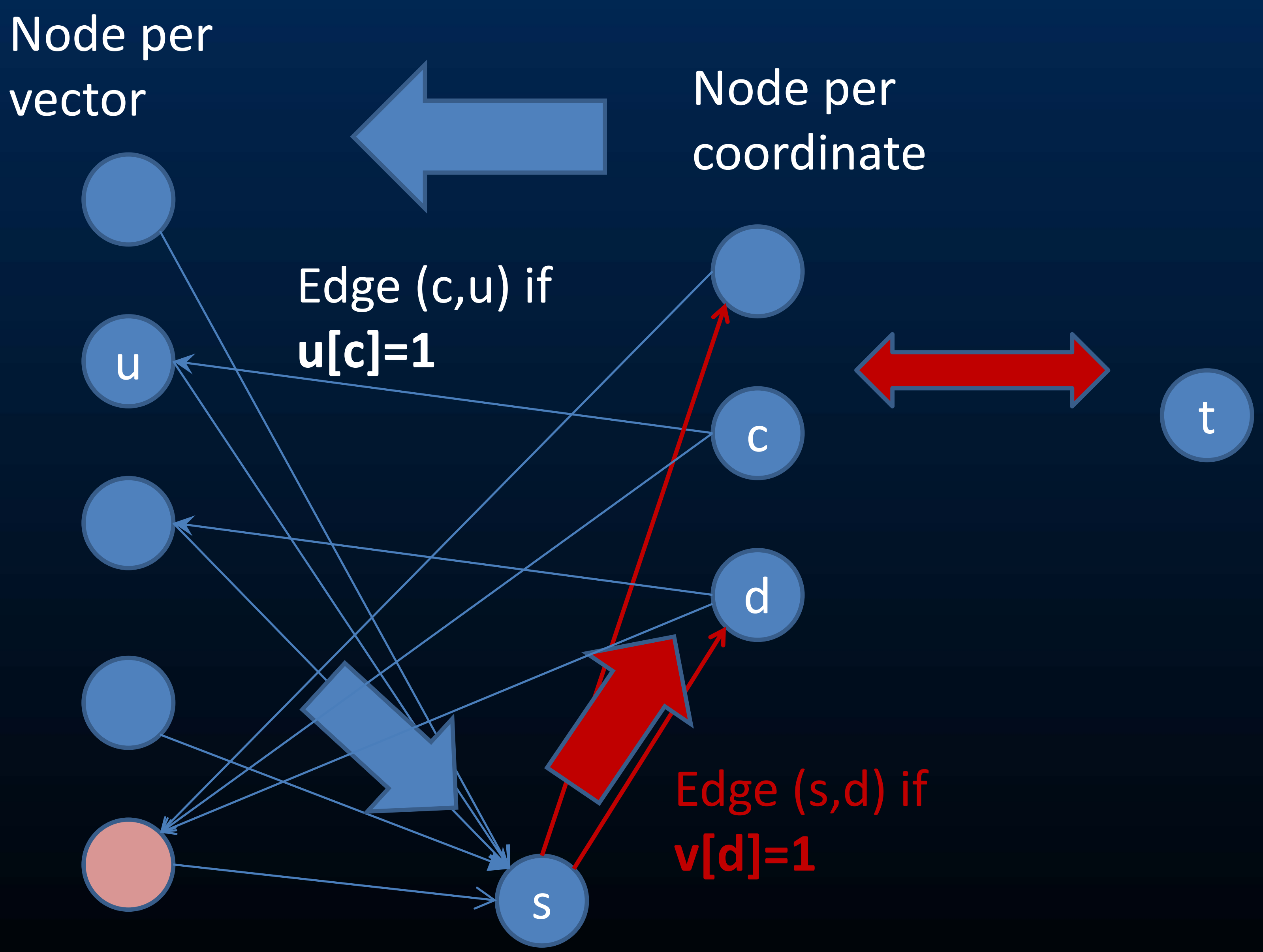
Dynamic #SCC>2 is hard

Stage for vector v (updates red):



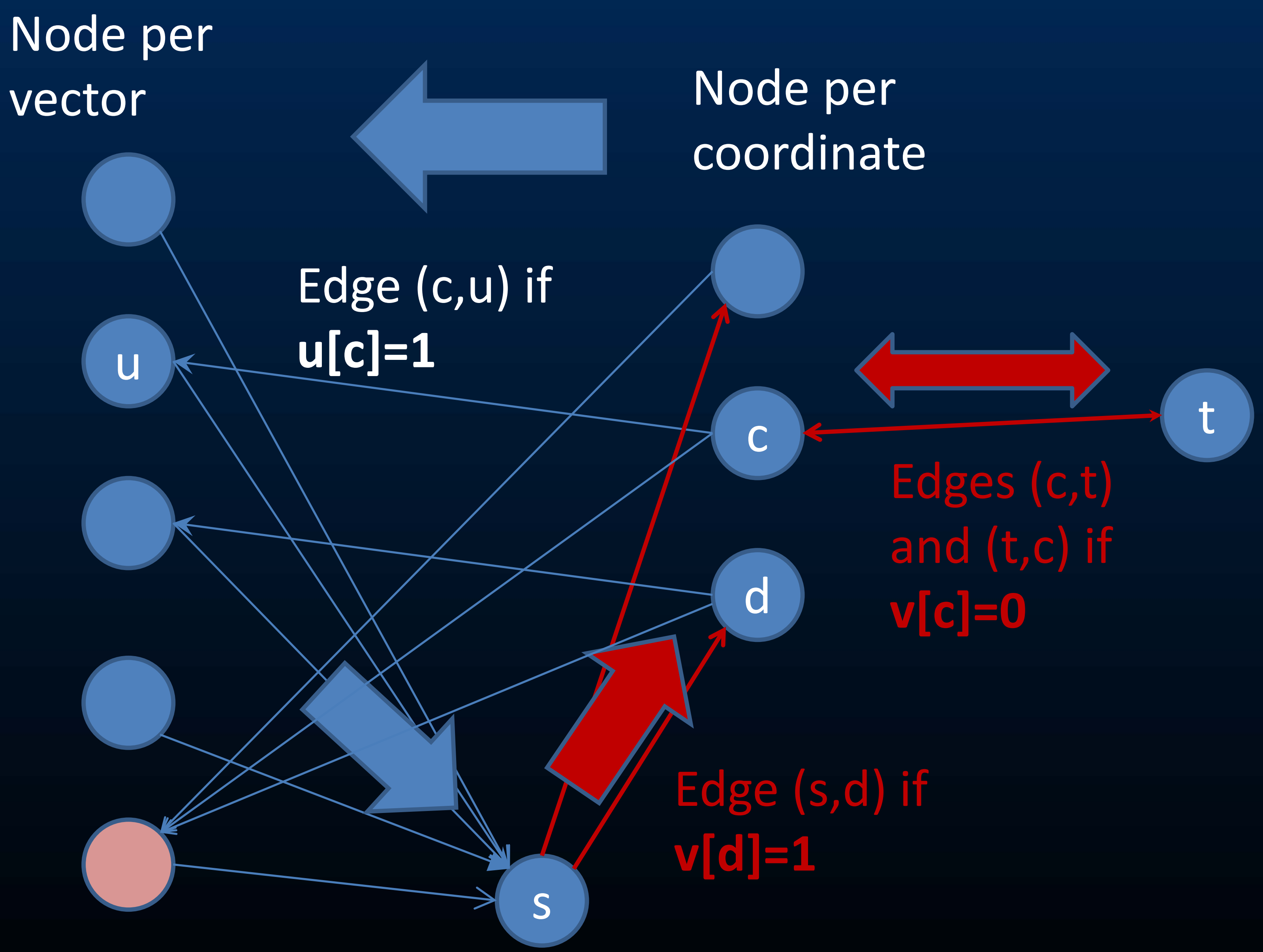
Dynamic #SCC>2 is hard

Stage for vector v (updates red):



Dynamic #SCC>2 is hard

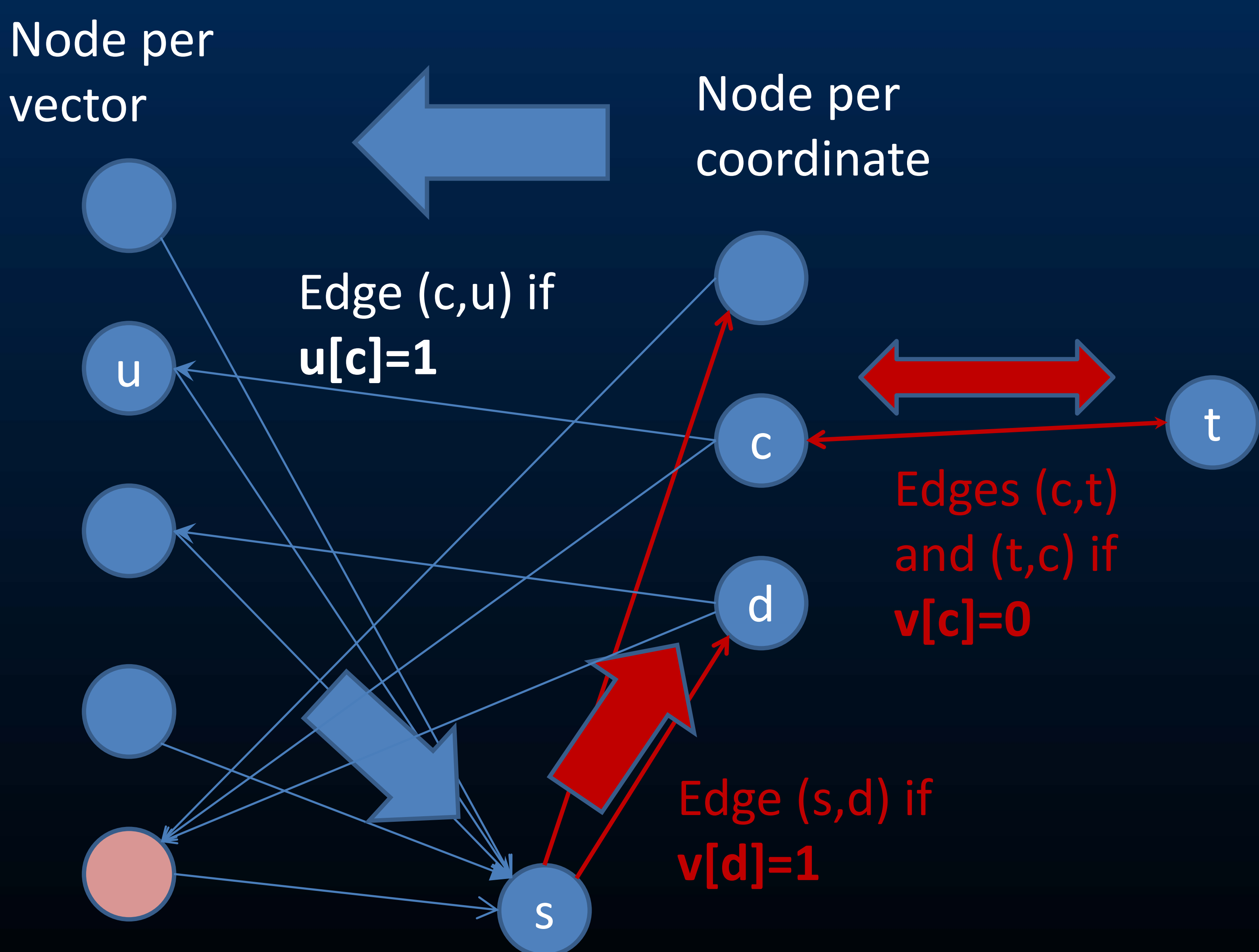
Stage for vector v (updates red):



Dynamic #SCC>2 is hard

Stage for vector v (updates red):

(1) No path from s to c if v[c]=0.

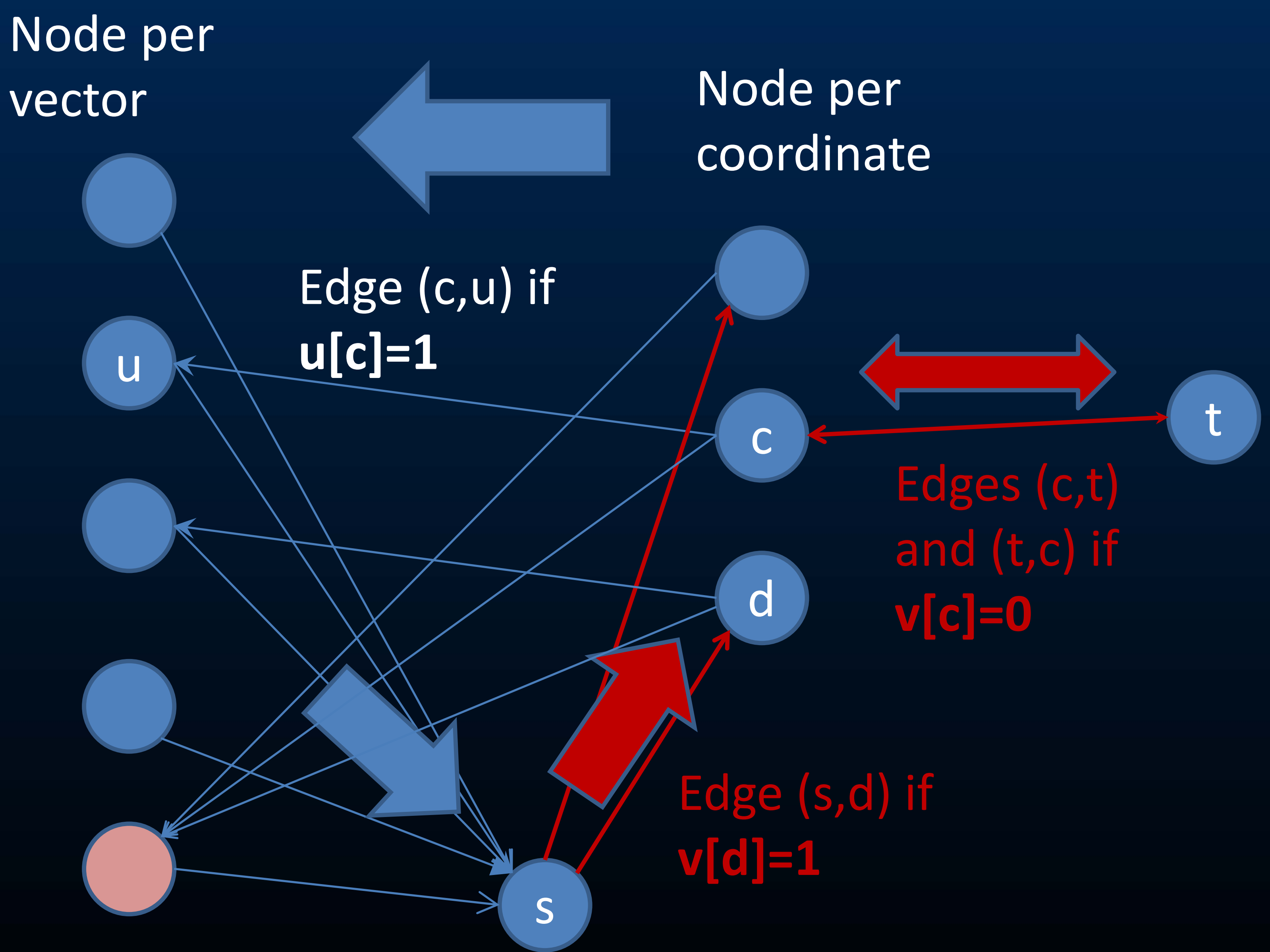


Dynamic #SCC>2 is hard

Stage for vector v (updates red):

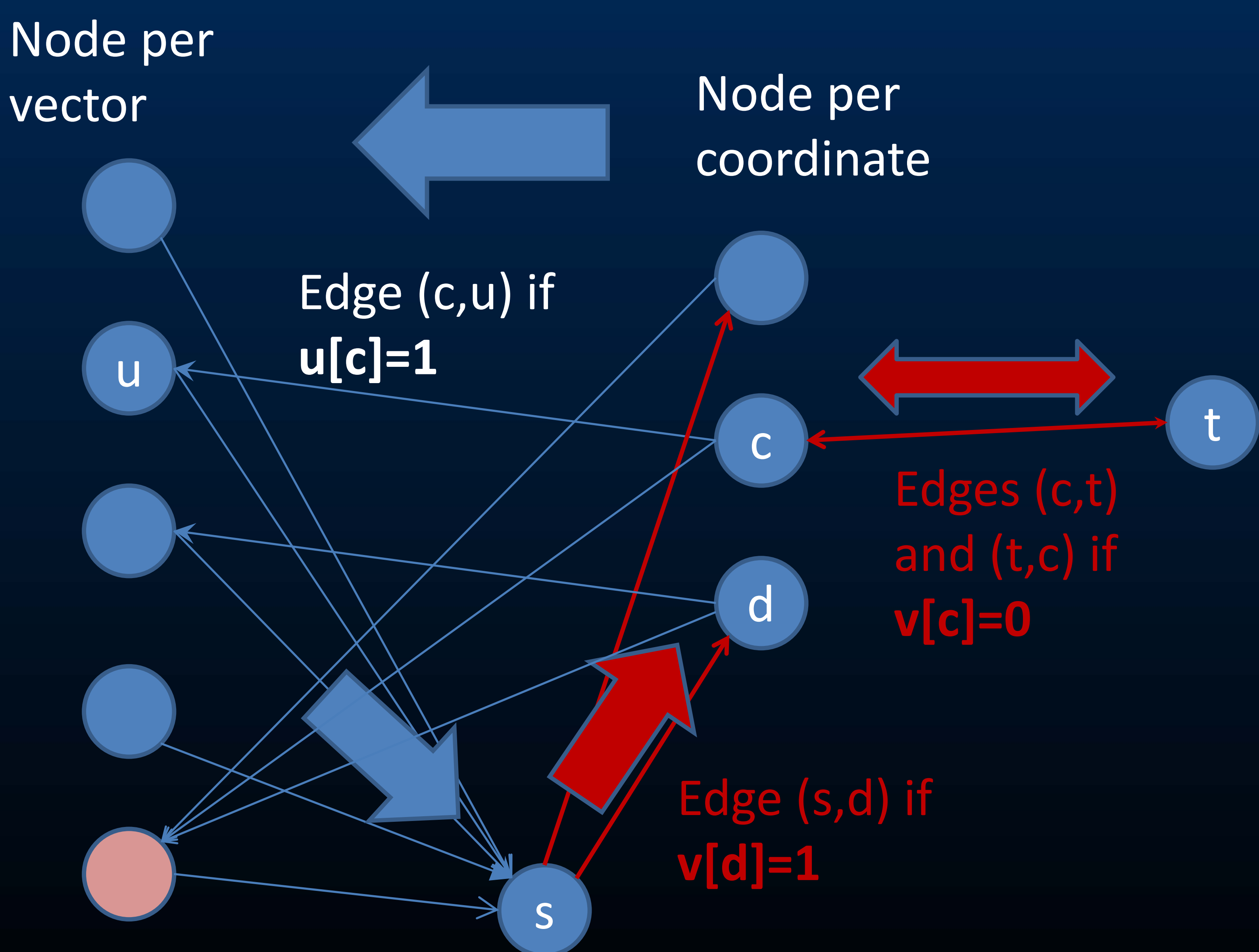
(1) No path from s to c if v[c]=0.

(2) No path from s to t.



Dynamic #SCC>2 is hard

Stage for vector v (updates red):



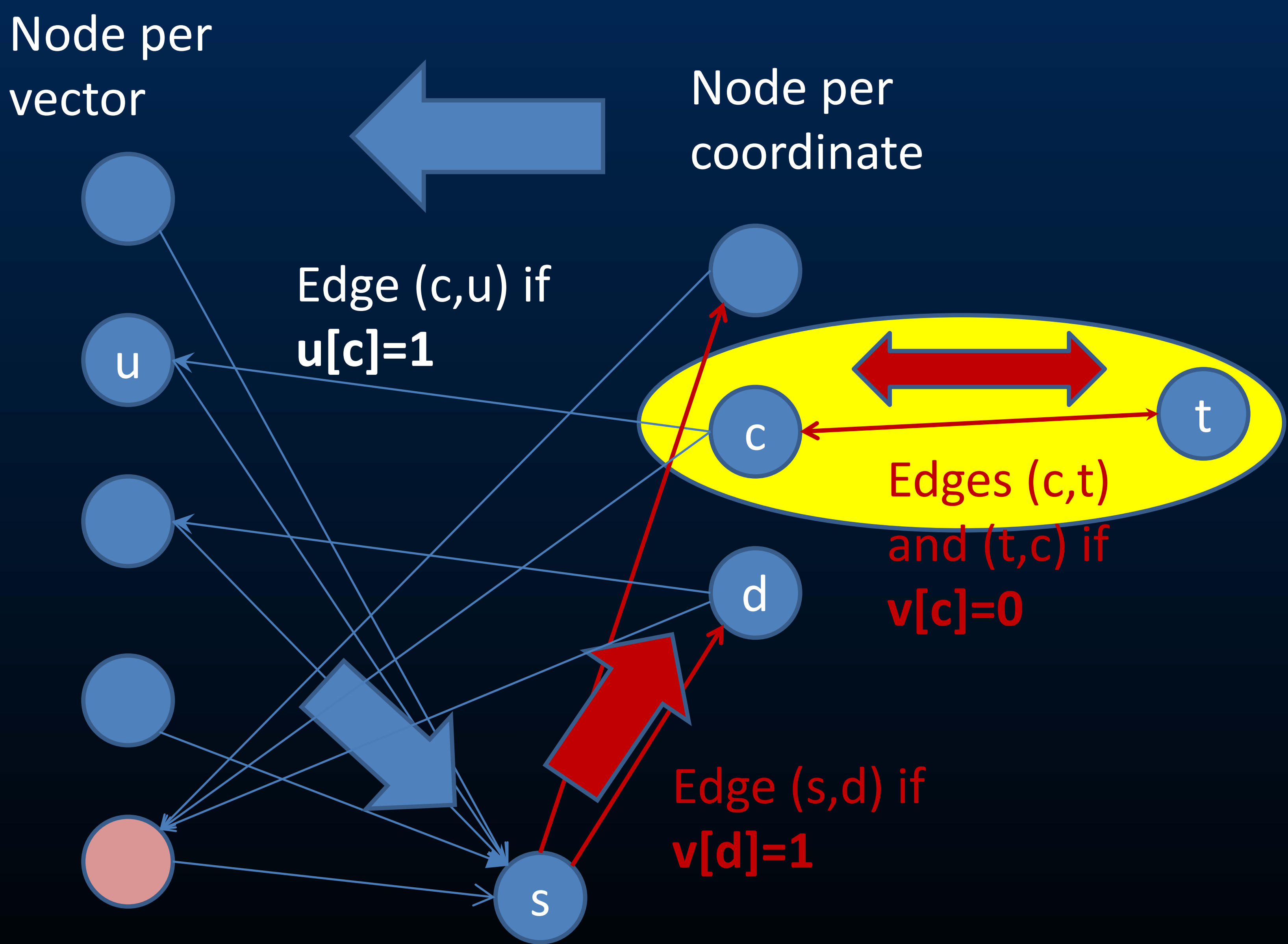
(1) No path from s to c if v[c]=0.

(2) No path from s to t.

(3) t is in an SCC with all c s.t. v[c]=0.

Dynamic #SCC>2 is hard

Stage for vector v (updates red):



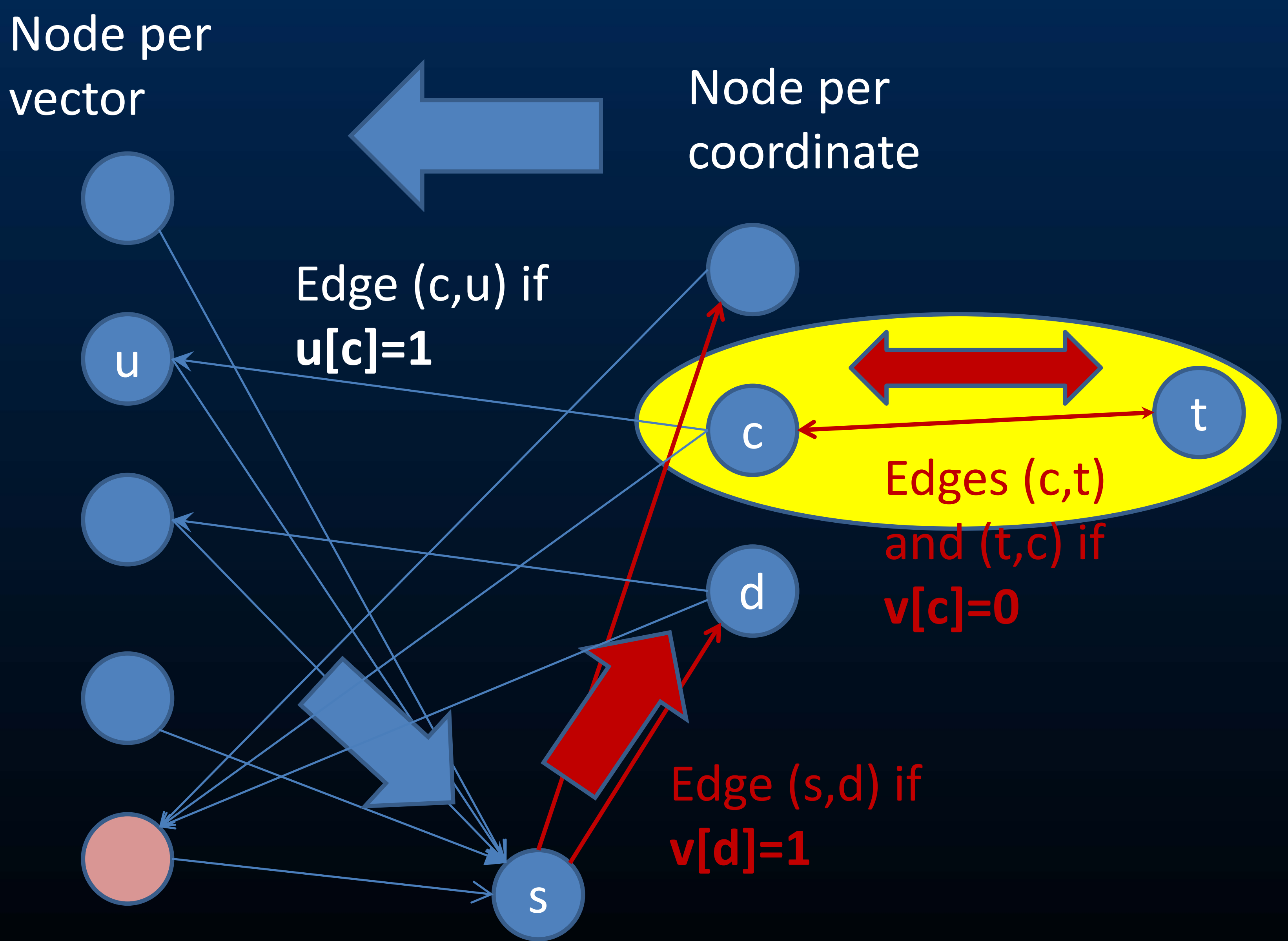
(1) No path from s to c if v[c]=0.

(2) No path from s to t.

(3) t is in an SCC with all c s.t. v[c]=0.

Dynamic #SCC>2 is hard

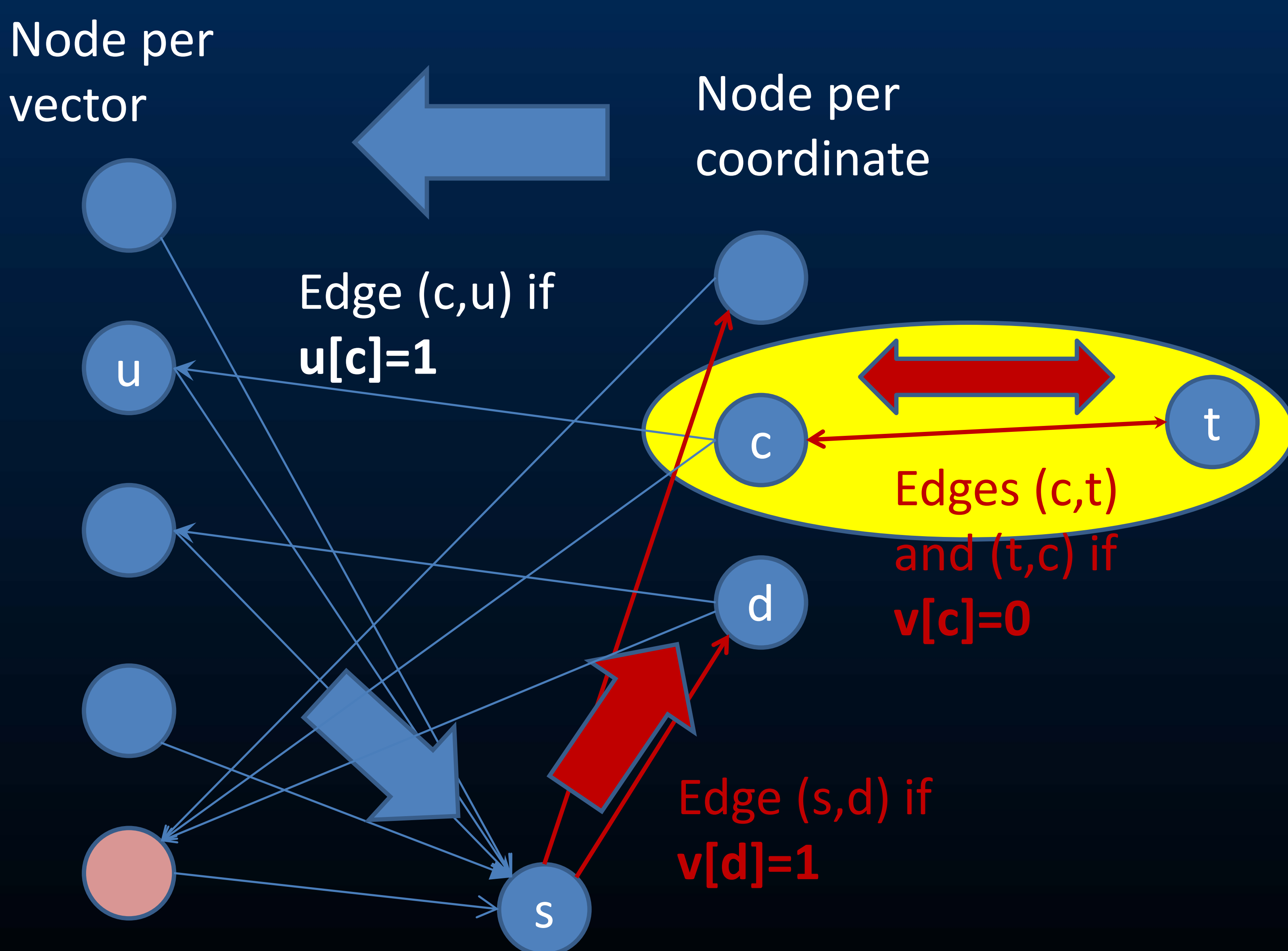
Stage for vector v (updates red):



(1) No path from s to c if v[c]=0.
(2) No path from s to t.
(3) t is in an SCC with all c s.t. v[c]=0.
(4) s is in an SCC with all c s.t. v[c]=1.

Dynamic #SCC>2 is hard

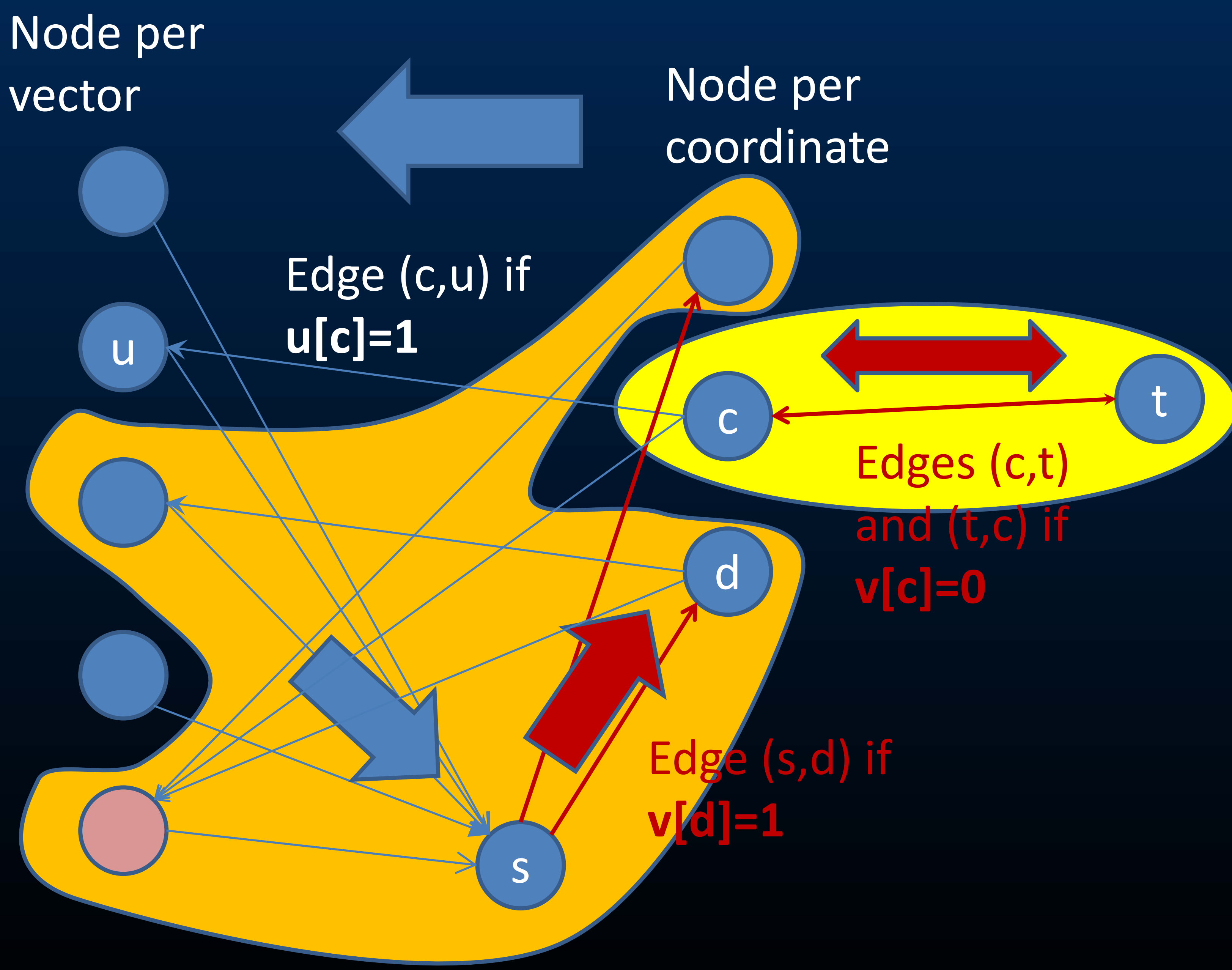
Stage for vector v (updates red):



(1) No path from s to c if v[c]=0.
(2) No path from s to t.
(3) t is in an SCC with all c s.t. v[c]=0.
(4) s is in an SCC with all c s.t. v[c]=1.
(5) u and s are in the same SCC iff there is a c with u[c]=v[c]=1, i.e. **iff u and v are not orthog**.

Dynamic #SCC>2 is hard

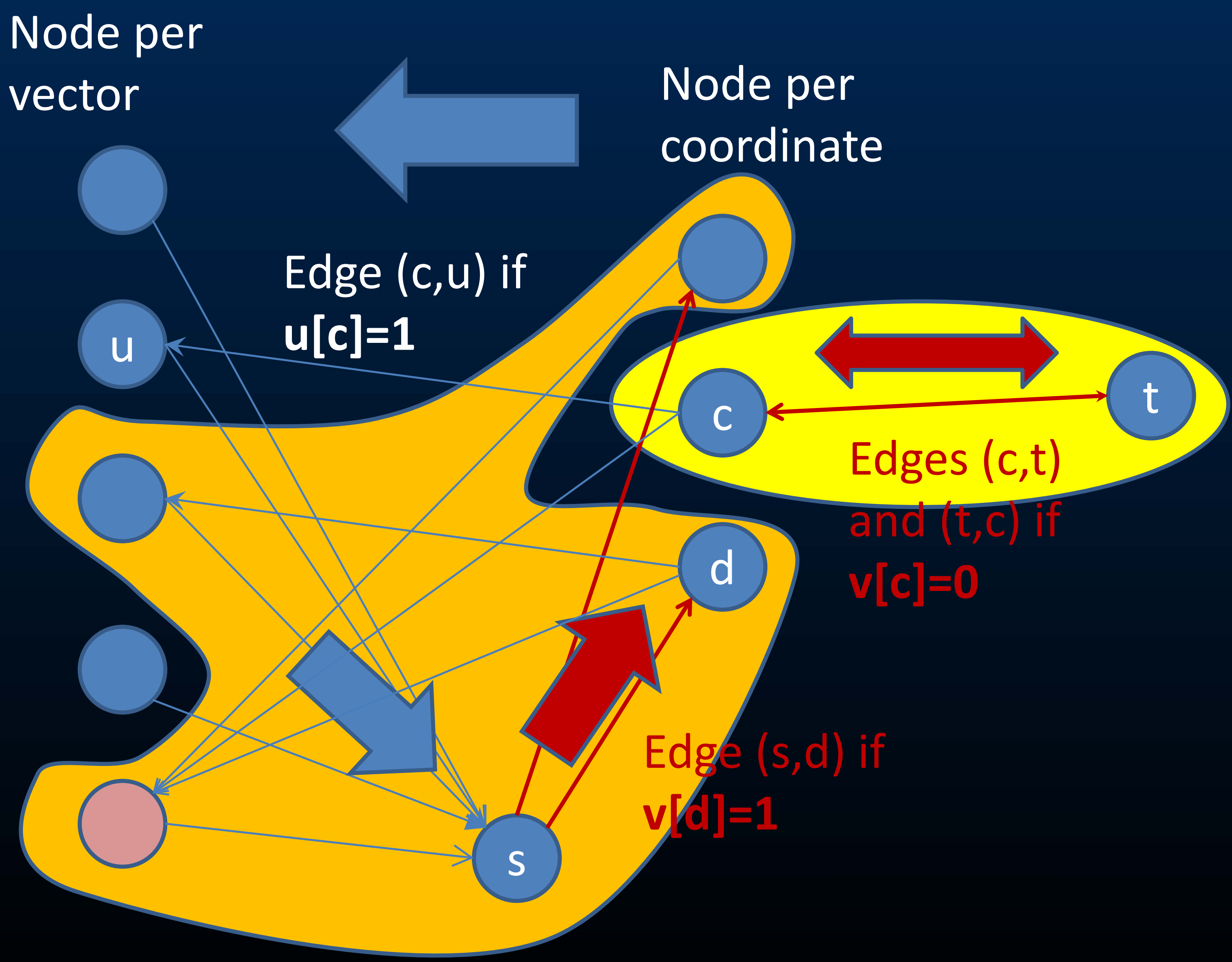
Stage for vector v (updates red):



(1) No path from s to c if v[c]=0.
(2) No path from s to t.
(3) t is in an SCC with all c s.t. v[c]=0.
(4) s is in an SCC with all c s.t. v[c]=1.
(5) u and s are in the same SCC iff there is a c with u[c]=v[c]=1, i.e. **iff u and v are not orthog**.

Dynamic #SCC>2 is hard

Stage for vector v (updates red):



(1) No path from s to c if v[c]=0.

(2) No path from s to t.

(3) t is in an SCC with all c s.t. v[c]=0.

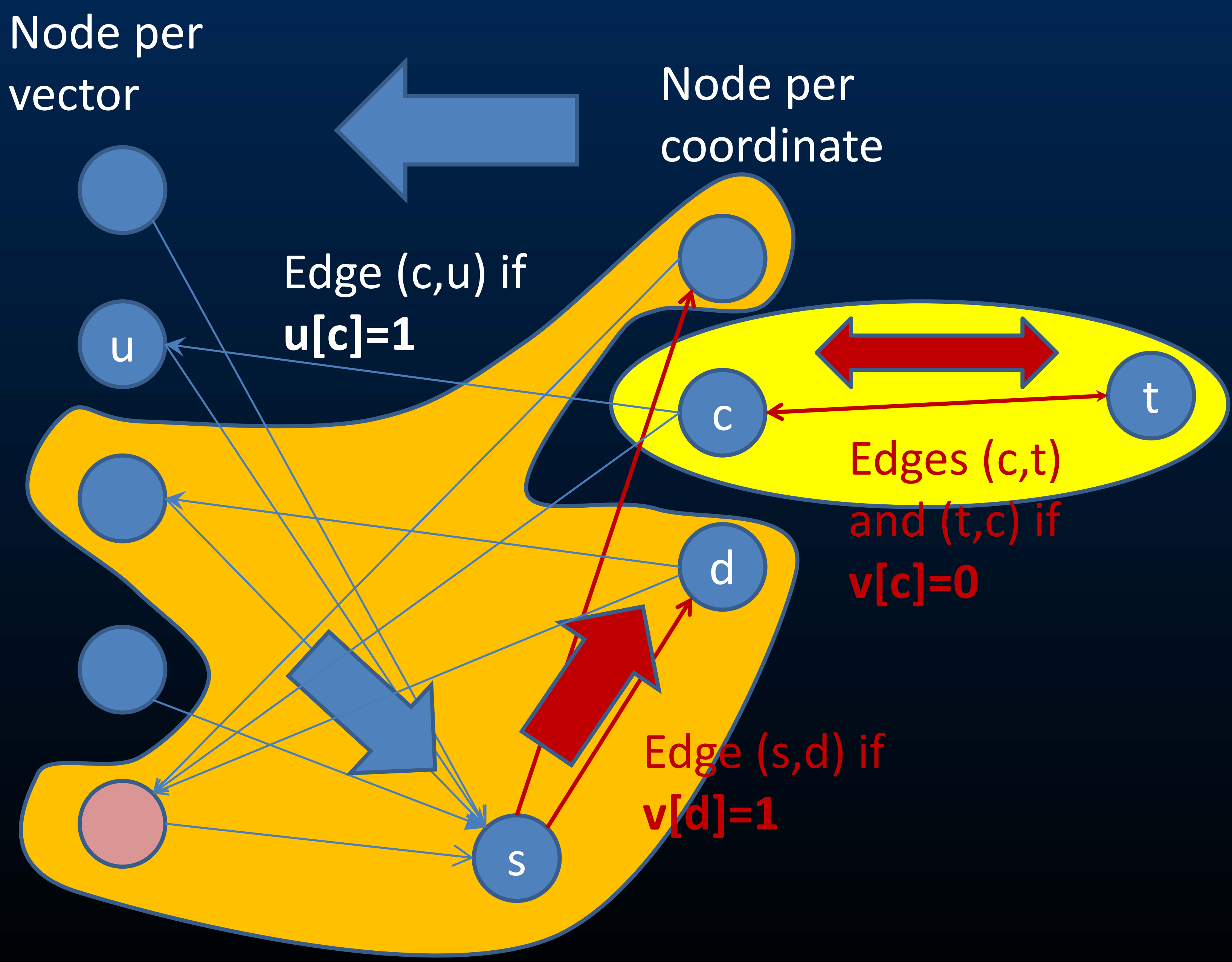
(4) s is in an SCC with all c s.t. v[c]=1.

(5) u and s are in the same SCC iff there is a c with u[c]=v[c]=1, i.e. **iff u and v are not orthog**.

Thus #SCC is 2 iff there is *no vector orthogonal to v*.

Dynamic #SCC>2 is hard

Stage for vector v (updates red):



(1) No path from s to c if v[c]=0.

(2) No path from s to t.

(3) t is in an SCC with all c s.t. v[c]=0.

(4) s is in an SCC with all c s.t. v[c]=1.

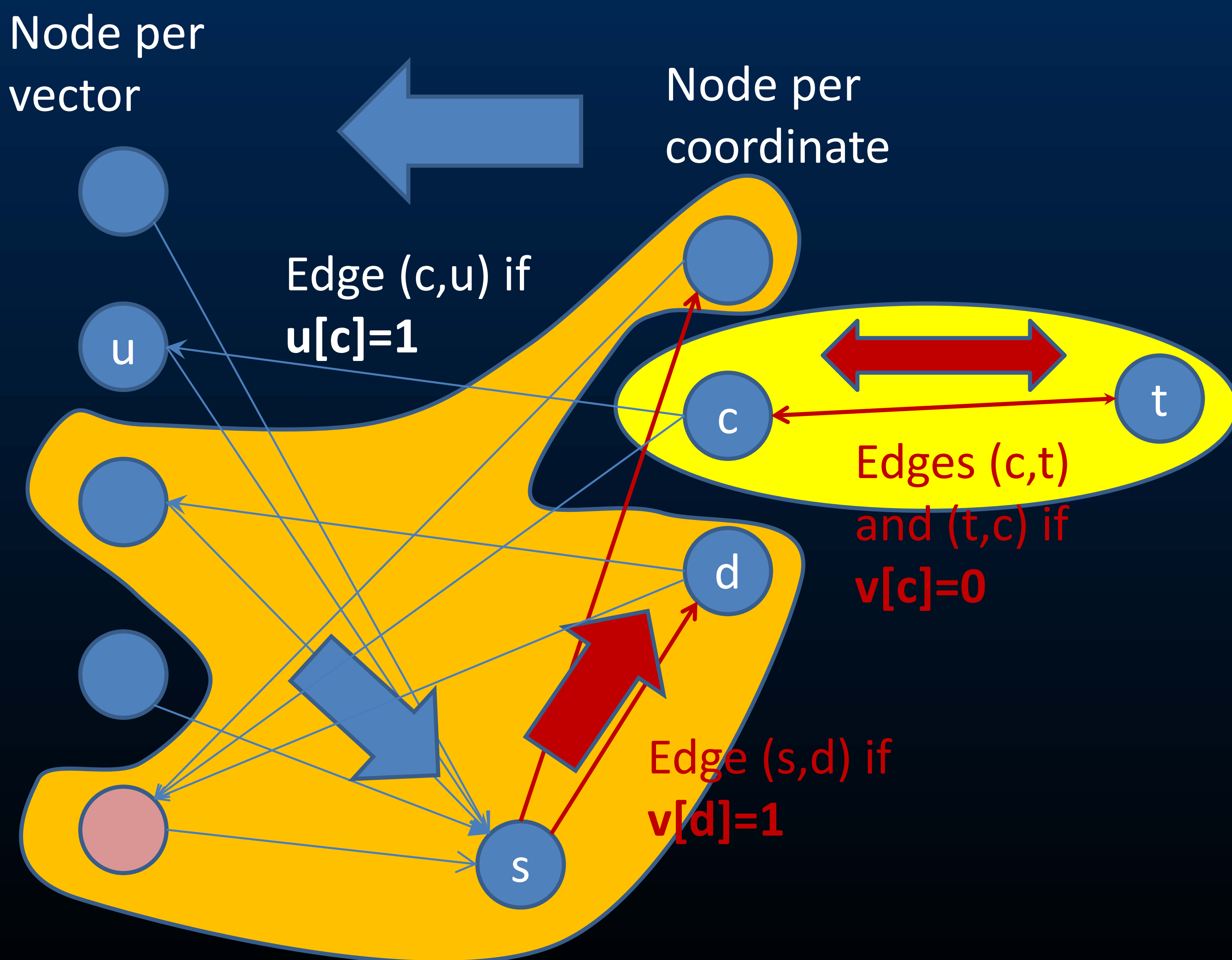
(5) u and s are in the same SCC iff there is a c with u[c]=v[c]=1, i.e. **iff u and v are not orthog**.

Thus #SCC is 2 iff there is *no vector orthogonal to v*.

O(n d) updates, n queries

Dynamic #SCC>2 is hard

Stage for vector v (updates red):



(1) No path from s to c if v[c]=0.
(2) No path from s to t.
(3) t is in an SCC with all c s.t. v[c]=0.
(4) s is in an SCC with all c s.t. v[c]=1.
(5) u and s are in the same SCC iff there is a c with u[c]=v[c]=1, i.e. **iff u and v are not orthog**.

Thus #SCC is 2 iff there is *no vector orthogonal to v*.

O(n d) updates, n queries
So a $\mathbf{n}^{1-o(1)}$ lower bound.

With additional gadgets, lower bounds for:
(more) Strongly Connected Components
Undirected Connectivity with node updates and more.

Next: even higher lower bounds!

Plan

➡ Overview of some lower bounds for dynamic problems

➡ Simple and powerful proofs

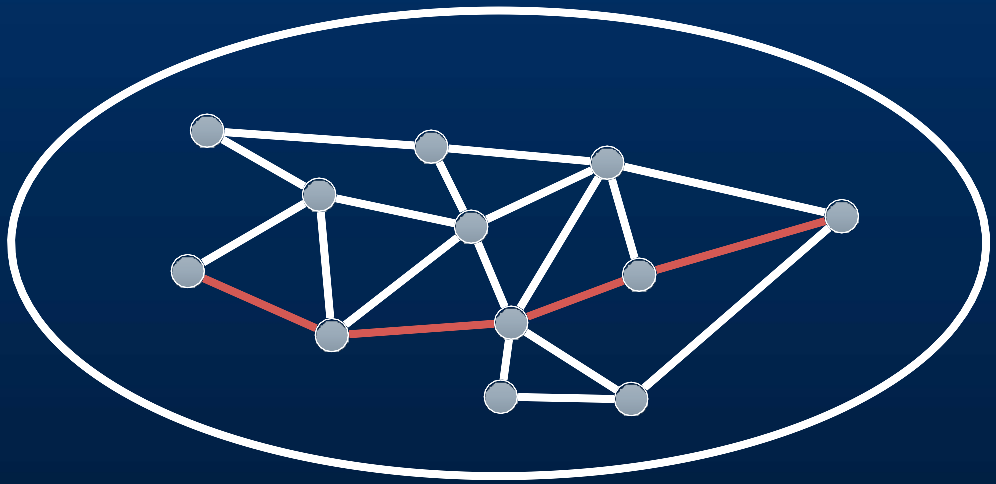
- Single Source Reachability
- #ss-Reach
- Strongly Connected Components
- Diameter
- s-t Shortest Path

Dynamic Diameter

<u>Input</u>: an undirected graph G

<u>Updates:</u> Add or remove edges.

<u>Query:</u> What is the <u>diameter</u> of G?



Upper bounds for dynamic All-Pairs-Shortest-Paths:
Naive: ~*O(mn)* per update.
[Demetrescu-Italiano 03', Thorup 04']: amortized ~$O(n^2)$.

<u>Theorem</u> [Abboud –VW FOCS 14']:
A $\frac{4}{3} - \epsilon$ approximation for the diameter of a sparse graph under edge updates with amortized $O(n^{2-\delta})$ update time for $\epsilon, \delta > 0$ refutes SETH!

Theorem [**Abboud** –VW FOCS 14']:
1.33-approximation for the diameter of a sparse graph under edge updates
with amortized $O(n^{2-\varepsilon})$ update time refutes SETH!

Proof outline:

Three Orthogonal Vectors (3-OV)

(0,0,…,1)	(1,0,…,1)	(1,0,…,1)
(0,1,…,1)	(0,1,…,0)	(0,0,…,1)
…	…	…
(1,0,…,0)	(1,0,…,1)	(1,1,…,0)

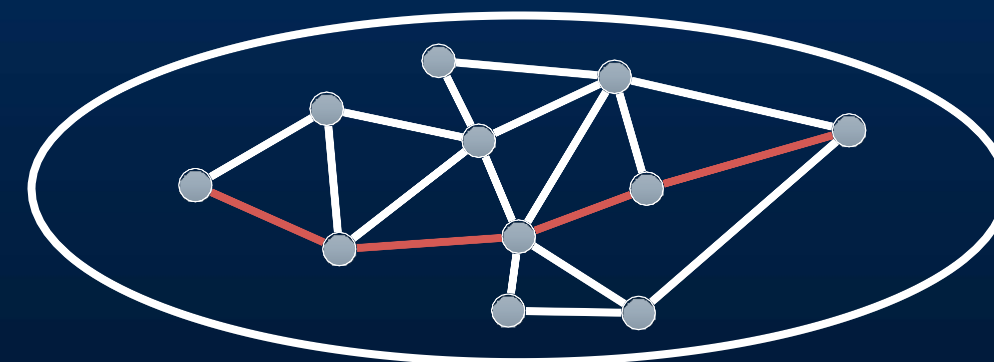
(1,0,1,…,0)

(0,1,1,…,0)

(1,1,1,…,0)

Given three lists of n vectors in $\{0,1\}^d$ is there an "orthogonal" triple?

dynamic Diameter



d = polylog(n)

Recall: **3-OV** in $n^{3-\varepsilon}$ ***poly d*** time refutes **SETH**

Theorem [A –VW FOCS 14']:

A $\frac{4}{3}-\epsilon$ approximation for the diameter of a sparse graph under edge updates with amortized $O(n^{2-\delta})$ update time for $\epsilon, \delta > 0$ refutes SETH!

Proof outline:

Three Orthogonal Vectors (3-OV)

(0,0,…,1)	(1,0,…,1)	(1,0,…,1)
(0,1,…,1)	(0,1,…,0)	(0,0,…,1)
…	…	…
(1,0,…,0)	(1,0,…,1)	(1,1,…,0)

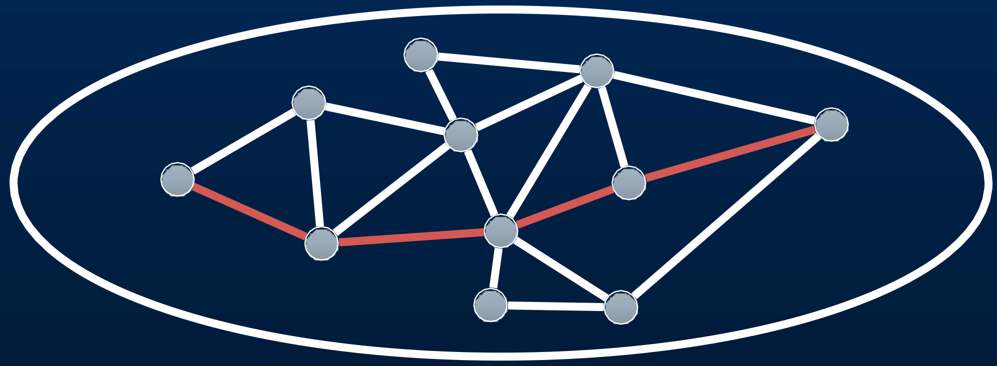
(1,0,1,…,0)

(0,1,1,…,0)

(1,1,1,…,0)

Given three lists of n vectors in $\{0,1\}^d$ is there an "orthogonal" triple?

dynamic Diameter



is the diameter 3 or more?

Graph G on ***m=O(nd)*** *nodes and edges*, **O(nd)** updates and queries

3-OV in ***$n^{2.9}$ poly d*** time

(refutes SETH)

O(nd) updates/queries in ***$n^{2.9}$poly d*** time

d=polylog(n), m=~O(n)

Amortized ***$O(m^{1.9})$*** update/query time

Theorem [Abboud -VW FOCS 14']:

A $\frac{4}{3}-\epsilon$ approximation for the diameter of a sparse graph under edge updates with amortized $O(n^{2-\delta})$ update time for $\epsilon, \delta > 0$ refutes SETH!

Proof:

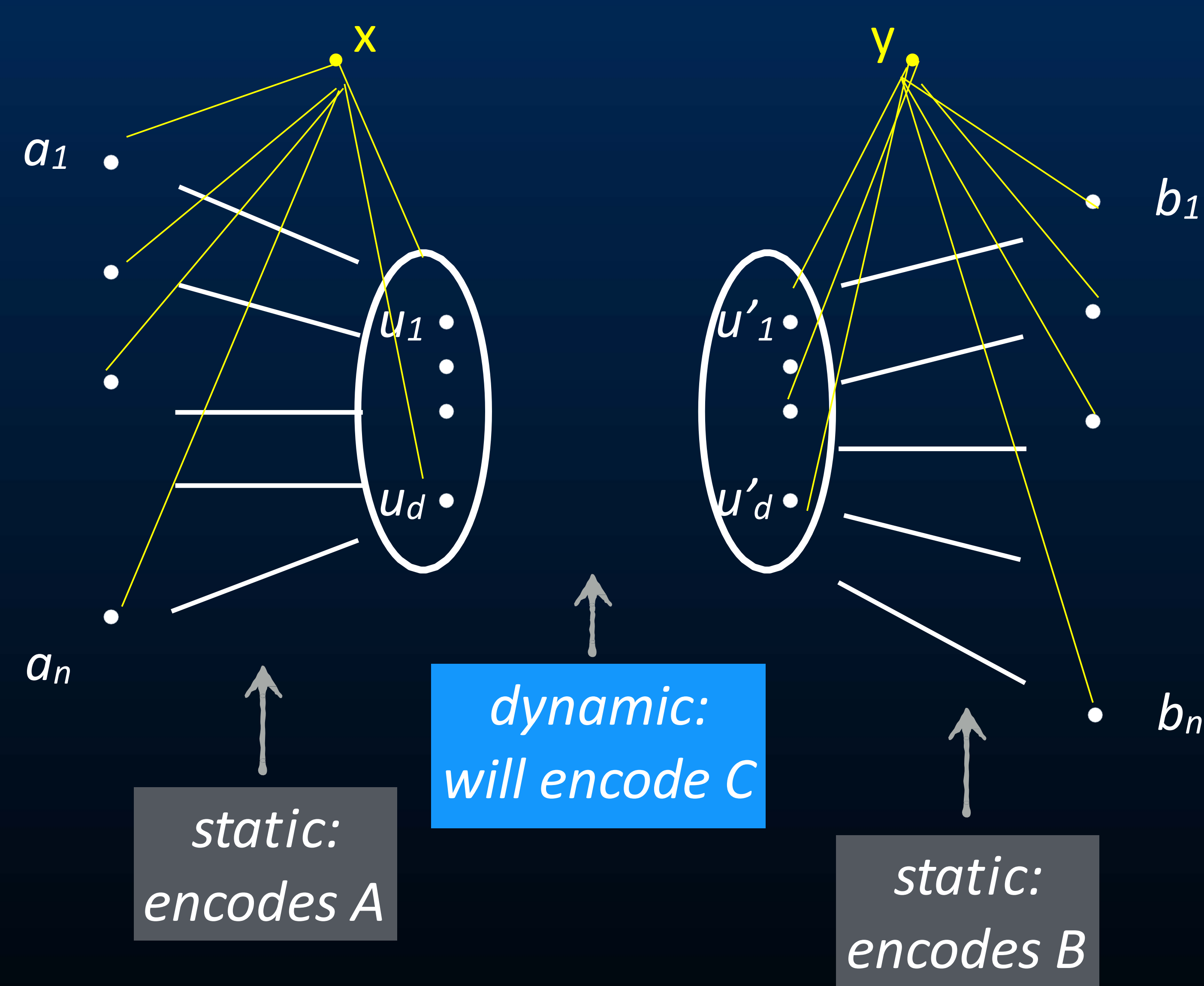
Three Orthogonal Vectors ➡ dynamic Diameter

A	B	C
(0,0,…,1)	(1,0,…,1)	(1,0,…,1)
(0,1,…,1)	(0,1,…,0)	(0,0,…,1)
…	…	…
(1,0,…,0)	(1,0,…,1)	(1,1,…,0)

(1,0,1,…,0)

(0,1,1,…,0)

(1,1,1,…,0)



add edge

u'_j •——• b_i

iff $b_i[j]=1$

Theorem [**A** –VW FOCS 14']:

A $\frac{4}{3}-\epsilon$ approximation for the diameter of a sparse graph under edge updates with amortized $O(n^{2-\delta})$ update time for $\epsilon, \delta > 0$ refutes SETH!

Proof:

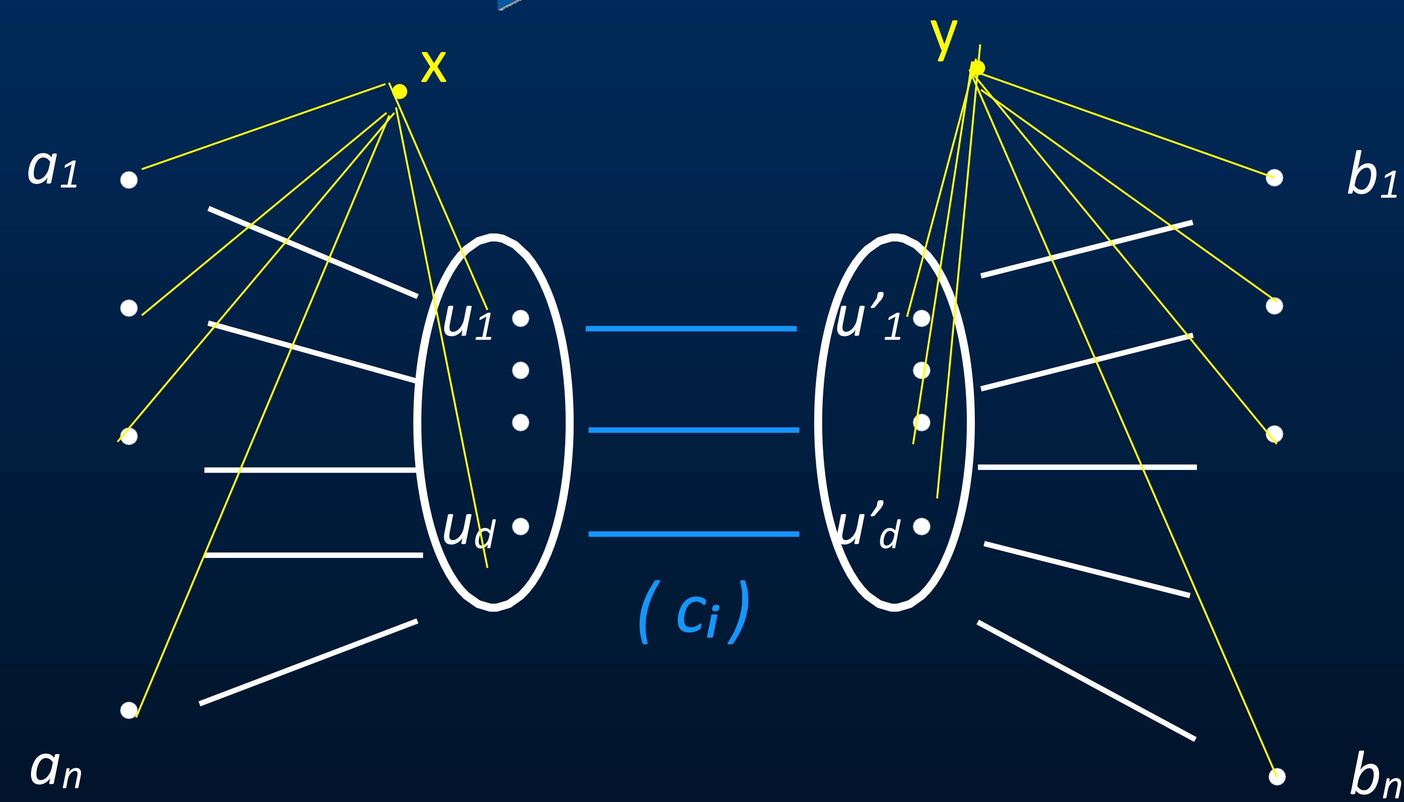
Three Orthogonal Vectors → **dynamic Diameter**

A	B	C
(0,0,…,1)	(1,0,…,1)	(1,0,…,1)
(0,1,…,1)	(0,1,…,0)	(0,0,…,1)
…	…	…
(1,0,…,0)	(1,0,…,1)	(1,1,…,0)

(1,0,1,…,0)

(0,1,1,…,0)

(1,1,1,…,0)



add edge

u'_j •——• b_i

iff $b_i[j]=1$

For each c_i:

1. add edges u_j •——• u'_j iff $c_i[j]=1$
2. ask Diameter query.

<u>Theorem</u> [**A** –VW FOCS 14']:

A $\frac{4}{3}-\epsilon$ approximation for the diameter of a sparse graph under edge updates with amortized $O(n^{2-\delta})$ update time for $\epsilon,\delta>0$ refutes SETH!

<u>Proof:</u>

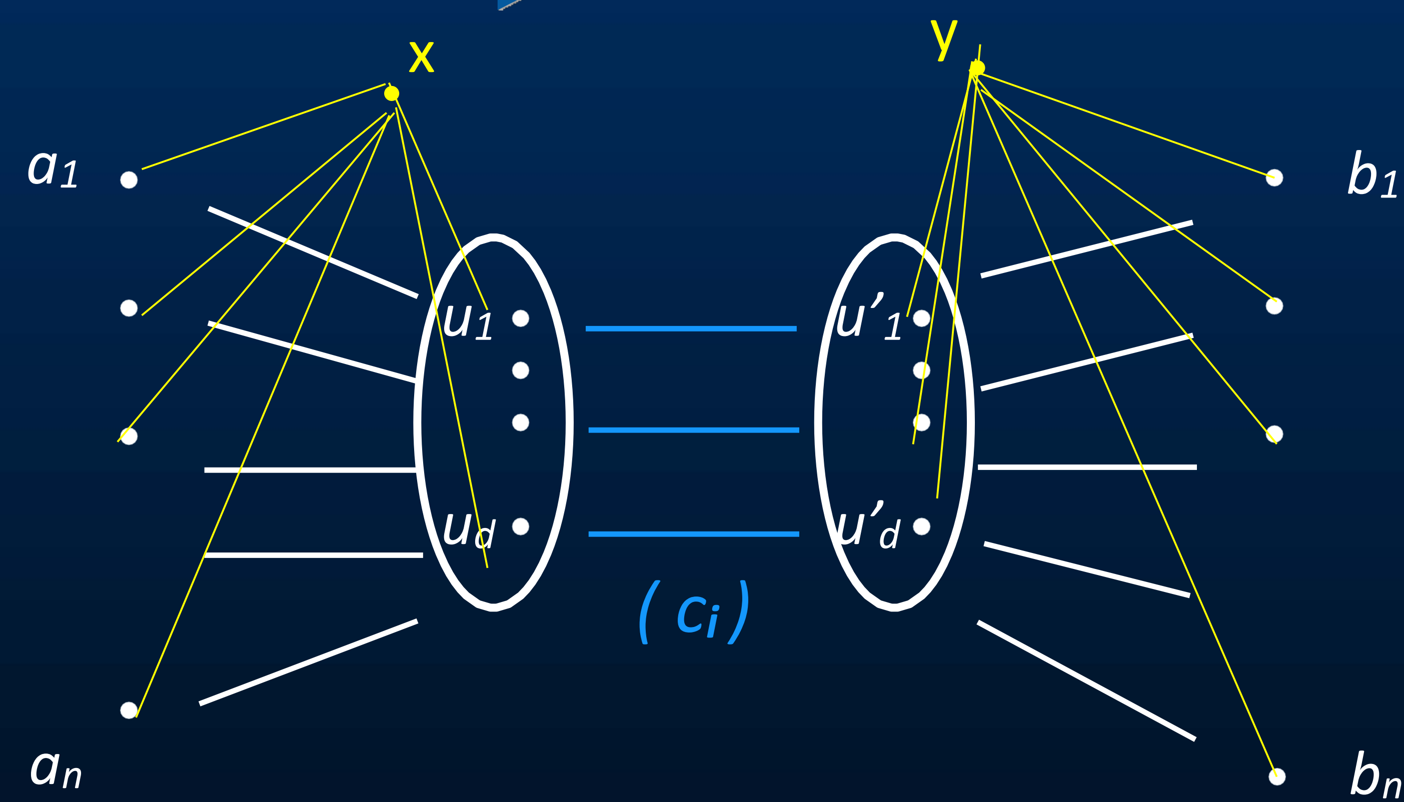
Three Orthogonal Vectors → dynamic Diameter

A	B	C
(0,0,…,1)	(1,0,…,1)	(1,0,…,1)
(0,1,…,1)	(0,1,…,0)	(0,0,…,1)
…	…	…
(1,0,…,0)	(1,0,…,1)	(1,1,…,0)

(1,0,1,…,0)

(0,1,1,…,0)

(1,1,1,…,0)



add edge

u'_j •——• b_i

iff $b_i[j]=1$

<u>Observation</u>:

The distance from *a* to *b* is more than ***3*** iff *a,b,c_i* are an orthogonal triple.

(no coordinate with all three 1's)

Theorem [**Abboud** –VW FOCS 14']:

A $\frac{4}{3}-\epsilon$ approximation for the diameter of a sparse graph under edge updates with amortized $O(n^{2-\delta})$ update time for $\epsilon, \delta > 0$ refutes SETH!

Proof:

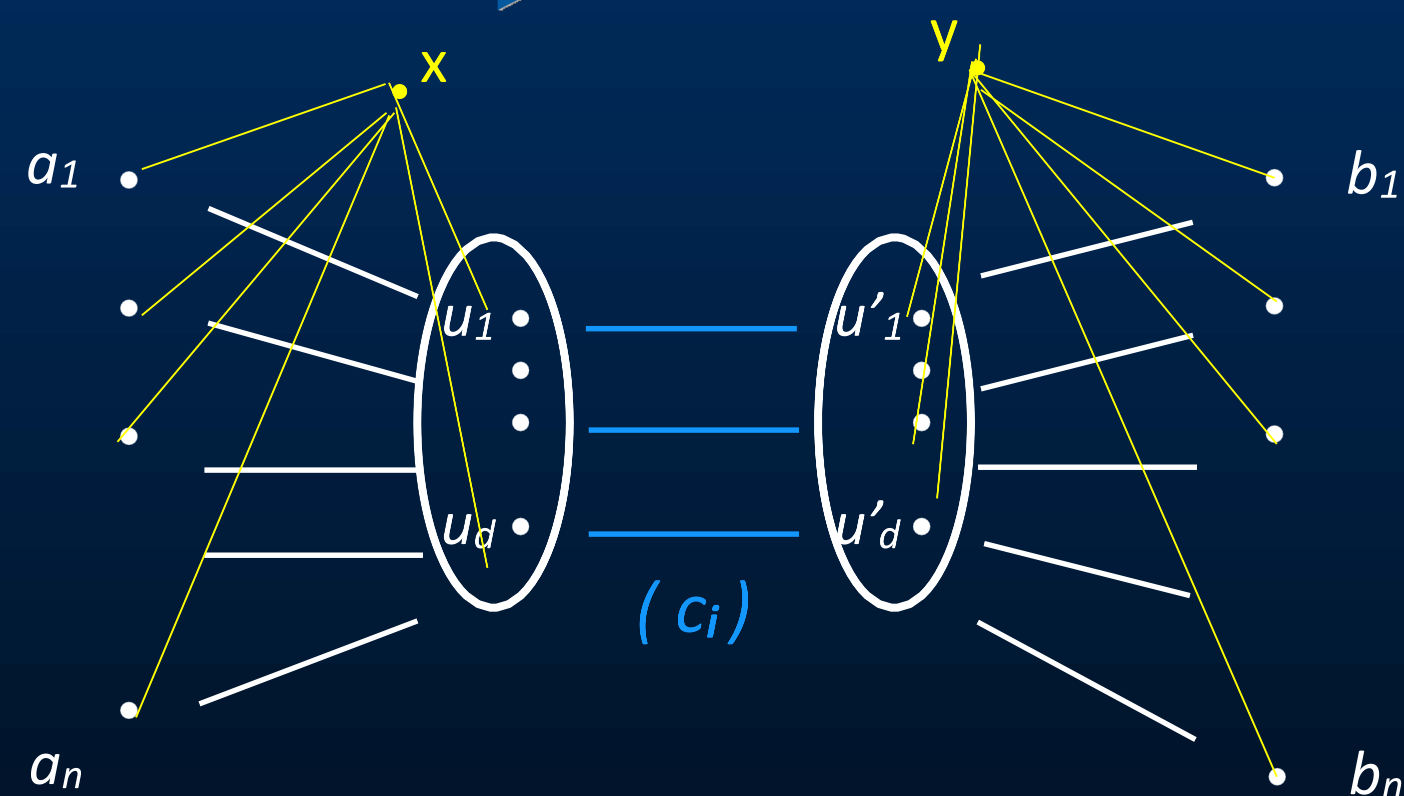
Three Orthogonal Vectors → dynamic Diameter

A	B	C
(0,0,…,1)	(1,0,…,1)	(1,0,…,1)
(0,1,…,1)	(0,1,…,0)	(0,0,…,1)
…	…	…
(1,0,…,0)	(1,0,…,1)	(1,1,…,0)

(1,0,1,…,0)

(0,1,1,…,0)

(1,1,1,…,0)



$O(nd)$ updates,
$m = O(nd)$ edges

$n^{2-o(1)}$ per update!

For each c_i:

1. add edges u_j •——• u'_j iff $c_i[j]=1$
2. Query. If Diameter > 3, output "yes".
3. remove edges and move on to next c_i

Plan

➡ Overview of some lower bounds for dynamic problems

➡ Simple and powerful proofs

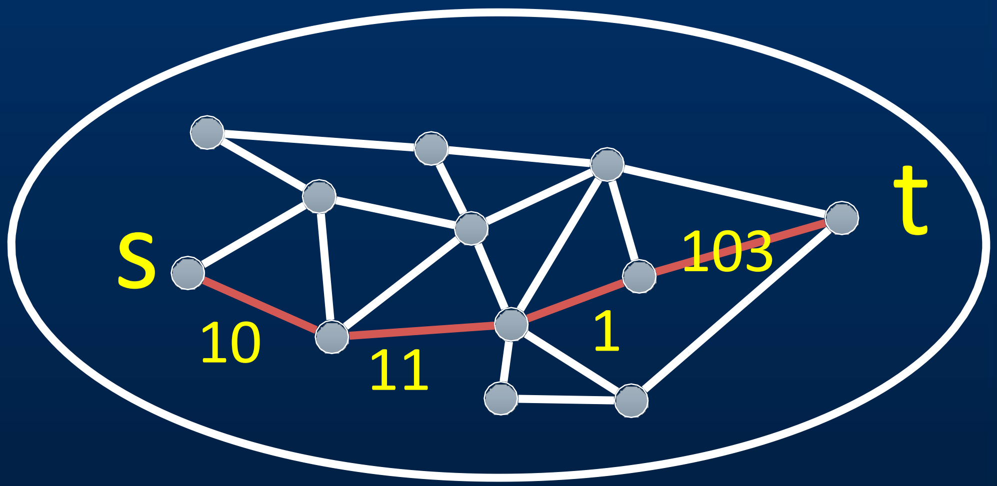
- Single Source Reachability
- #ss-Reach
- Strongly Connected Components
- Diameter
- s-t Shortest Path

Decremental s-t Shortest Path

Input: an weighted graph G, nodes s,t

Updates: Remove weighted edges.

Query: What is $d(s,t)$?



Upper bounds:
Naive: $\tilde{O}(m)$ per update. $\tilde{O}(n^2)$ for dense graphs

Theorem [RZ'04, A VW'14]
If s-t Shortest Path in dense m edge graphs can be supported with $O(m^{1-\epsilon})$ time per update, after $O(n^{3-\epsilon})$ preprocessing time for $\epsilon > 0$, then APSP in n node graphs is in $O(n^{3-\epsilon})$ time.

Theorem [RZ'04, A VW'14]

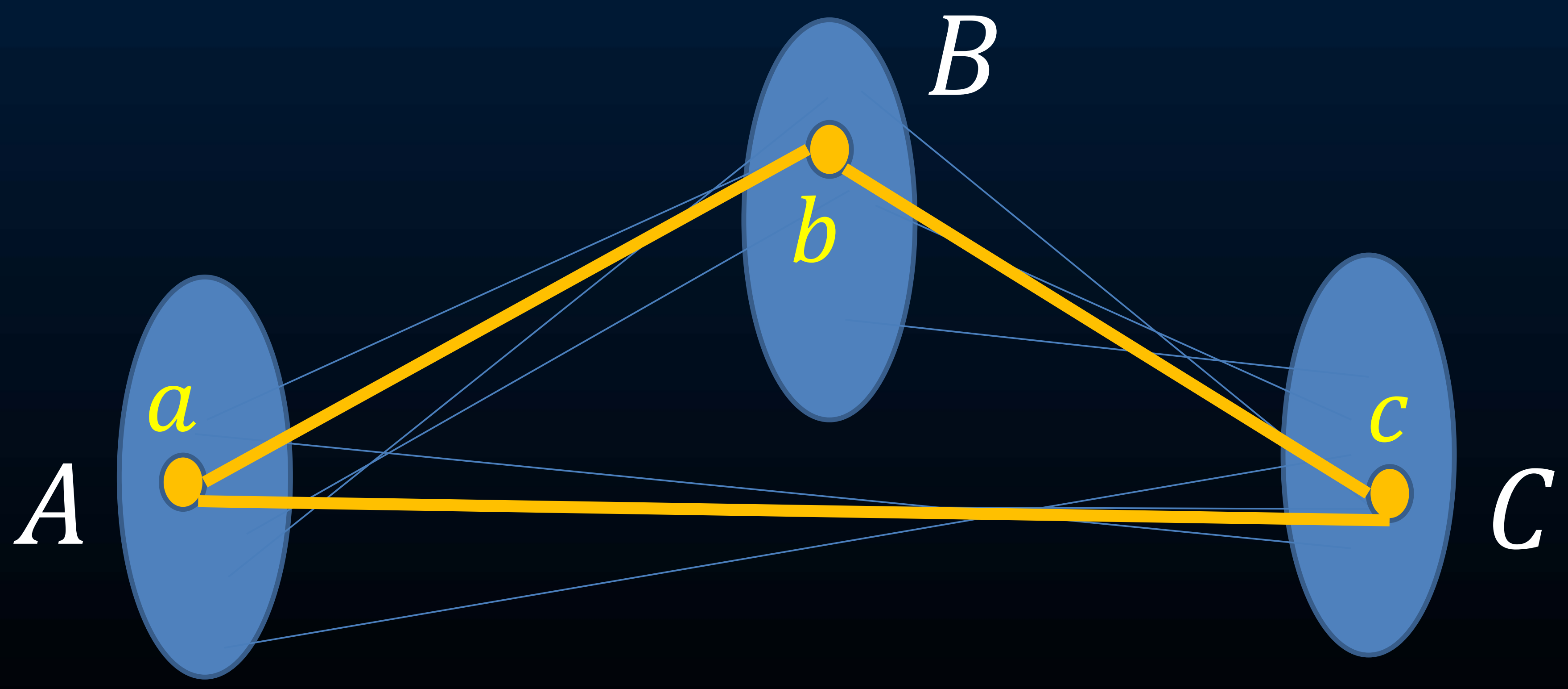
If s-t Shortest Path in dense m edge graphs can be supported with $O\left(m^{1-\epsilon}\right)$ time per update, after $O\left(n^{3-\epsilon}\right)$ preprocessing time for $\epsilon > 0$, then APSP in n node graphs is in $O(n^{3-\epsilon})$ time.

Reduction from Negative Triangle:

Theorem [RZ'04, A VW'14]
If s-t Shortest Path in dense m edge graphs can be supported with $O(m^{1-\epsilon})$ time per update, after $O(n^{3-\epsilon})$ preprocessing time for $\epsilon > 0$, then APSP in n node graphs is in $O(n^{3-\epsilon})$ time.

Reduction from Negative Triangle:

We are given tripartite G with parts A,B,C and want to know if $\exists a \in A, b \in B, c \in C$: $w(a,b) + w(b,c) + w(c,a) < 0$.

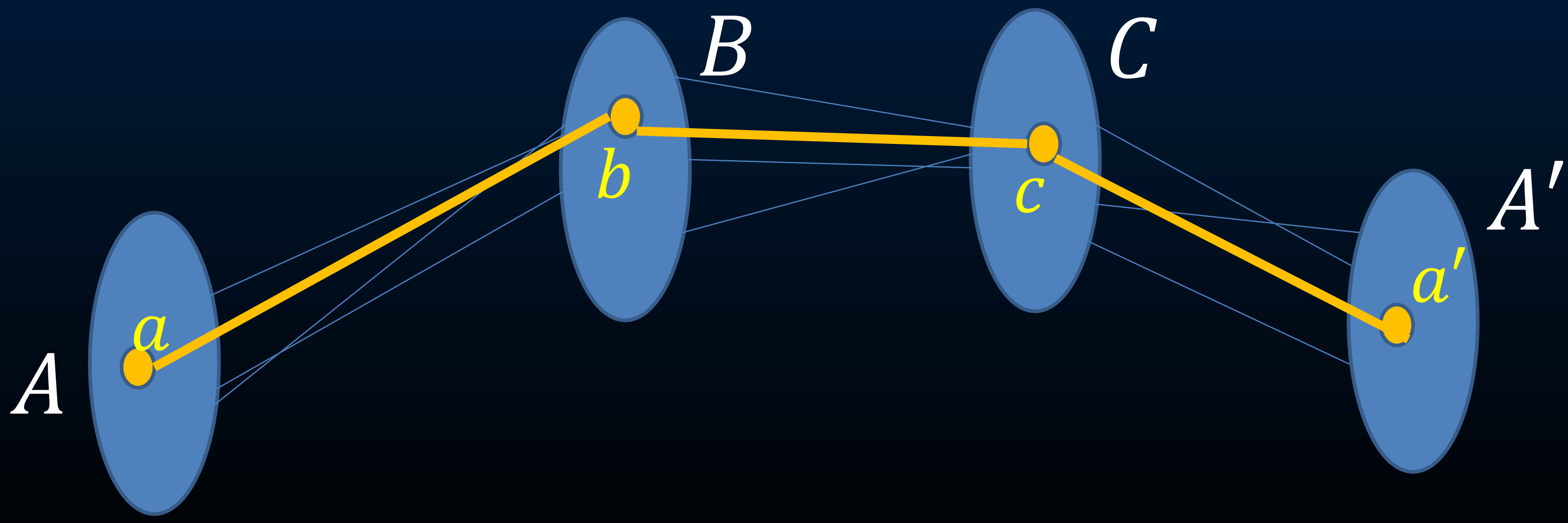


Theorem [RZ'04, A VW'14]

If s-t Shortest Path in dense m edge graphs can be supported with $O(m^{1-\epsilon})$ time per update, after $O(n^{3-\epsilon})$ preprocessing time for $\epsilon > 0$, then APSP in n node graphs is in $O(n^{3-\epsilon})$ time.

This is the same as: Given G' with parts A,B,C, A' and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', w(a,b) + w(b,c) + w(c,a') < 0.$$

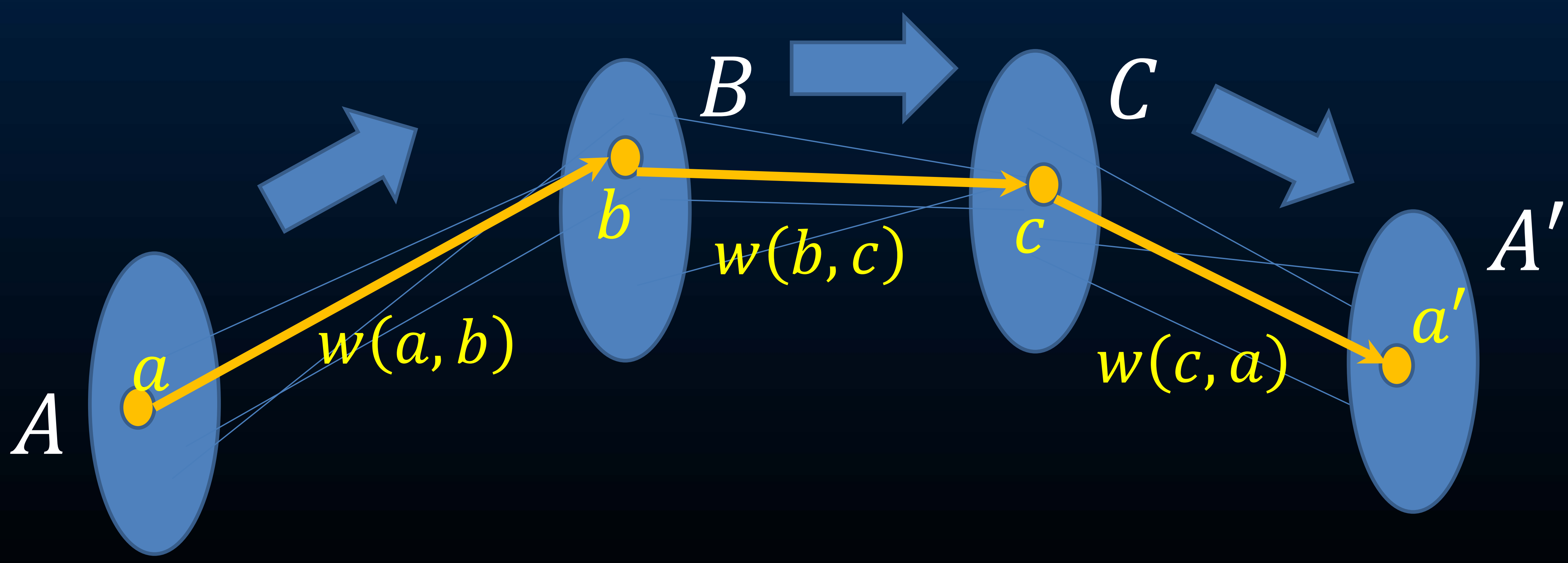


Theorem [RZ'04, A VW'14]

If s-t Shortest Path in dense m edge graphs can be supported with $O(m^{1-\epsilon})$ time per update, after $O(n^{3-\epsilon})$ preprocessing time for $\epsilon > 0$, then APSP in n node graphs is in $O(n^{3-\epsilon})$ time.

This is the same as: Given directed layered G' with parts A,B,C, A' and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

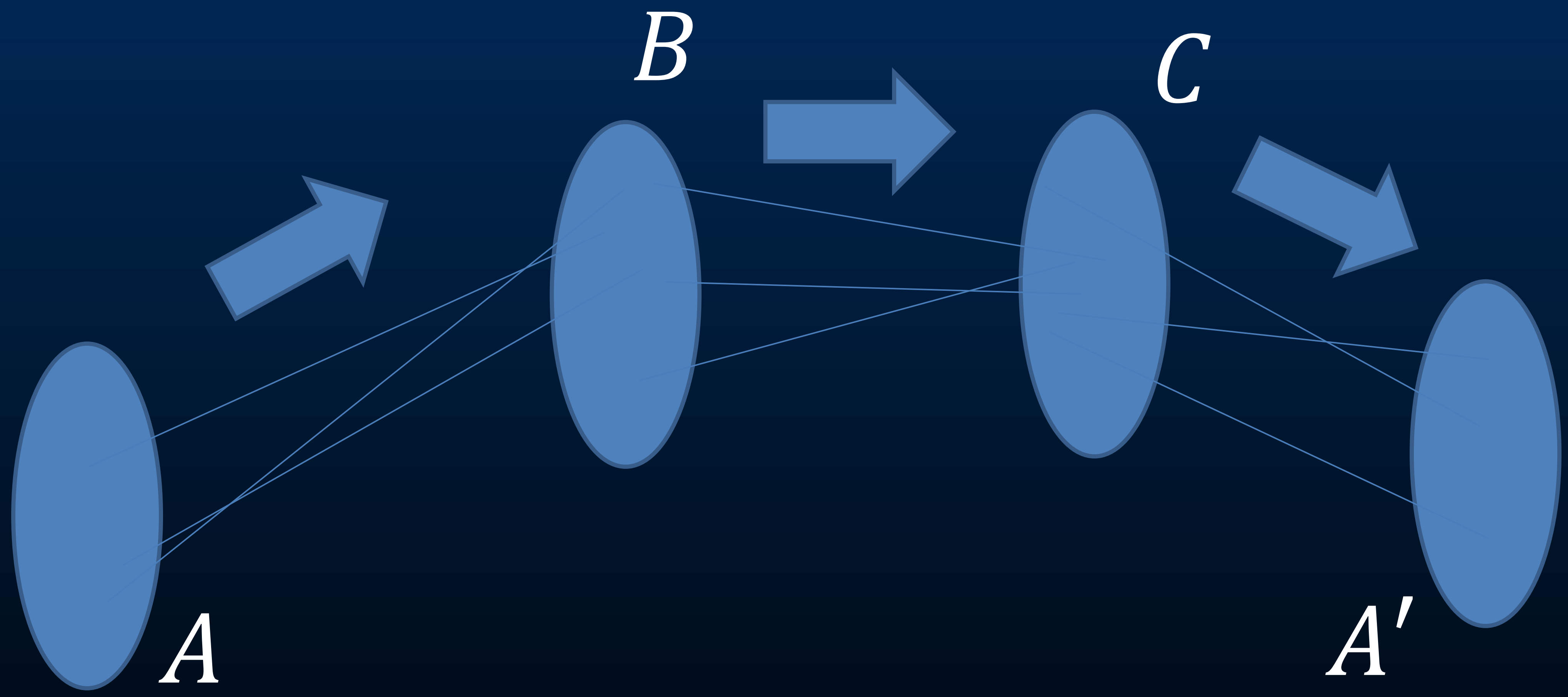
$$a = a', d(a, a') < 0.$$



Given directed layered G' with parts A,B,C, A' and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

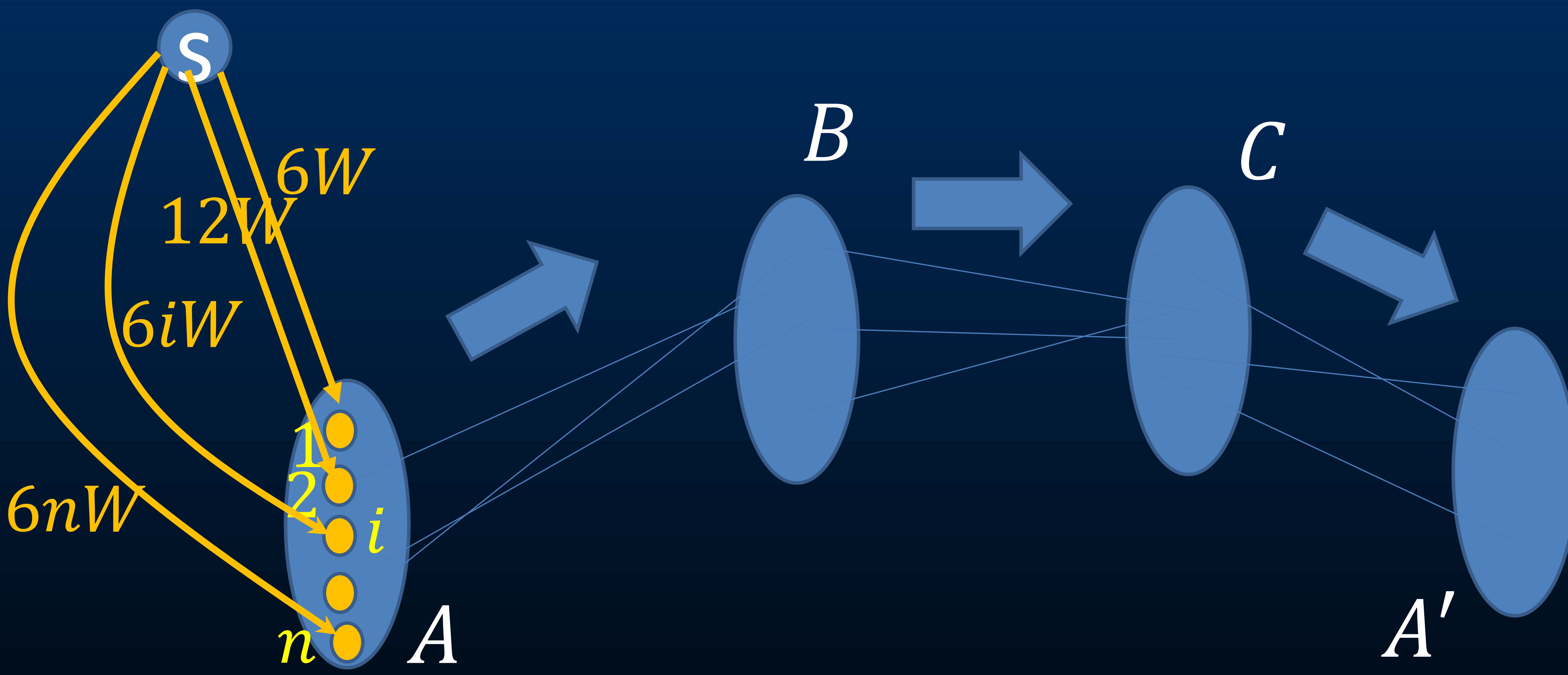
All edge weights lie in $\{-W, \ldots, W\}$.



Given directed layered G' with parts A,B,C, A' and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

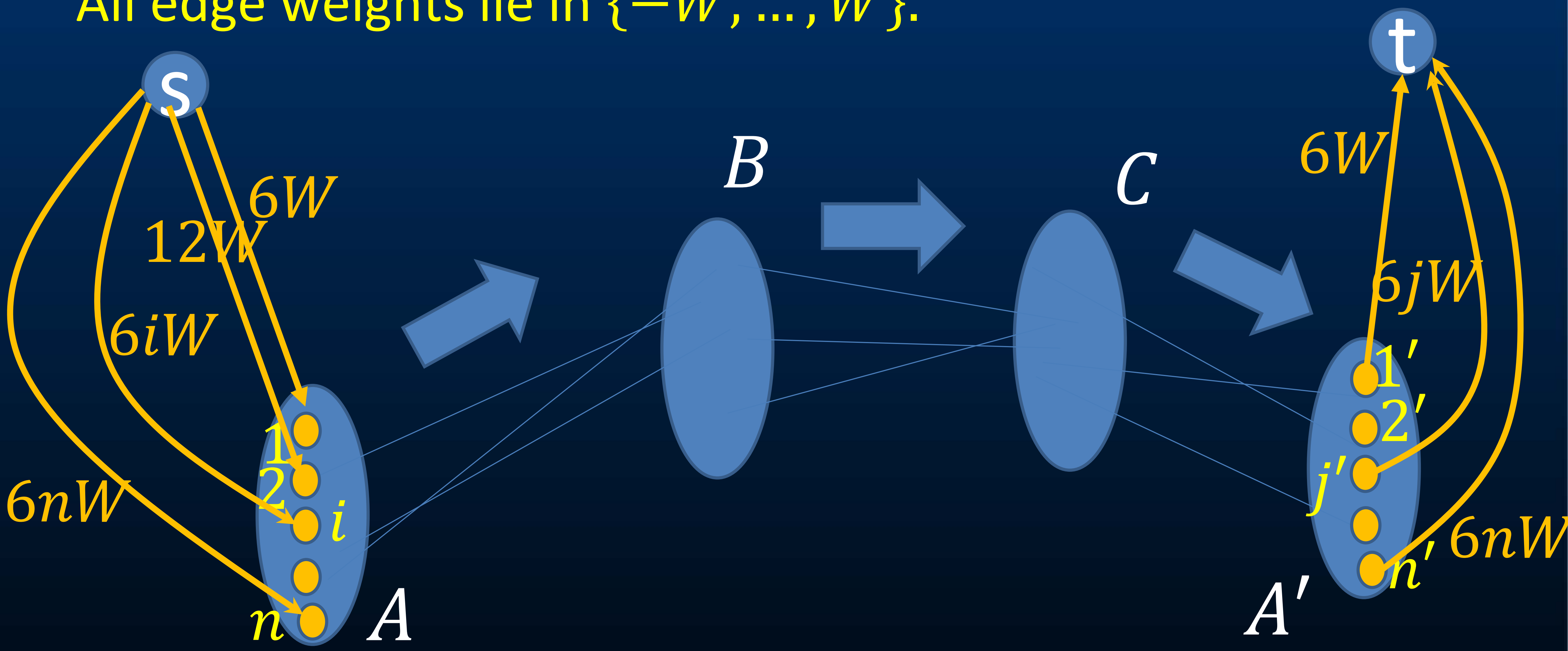
All edge weights lie in $\{-W, \ldots, W\}$.



Given directed layered G’ with parts A,B,C, A’ and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

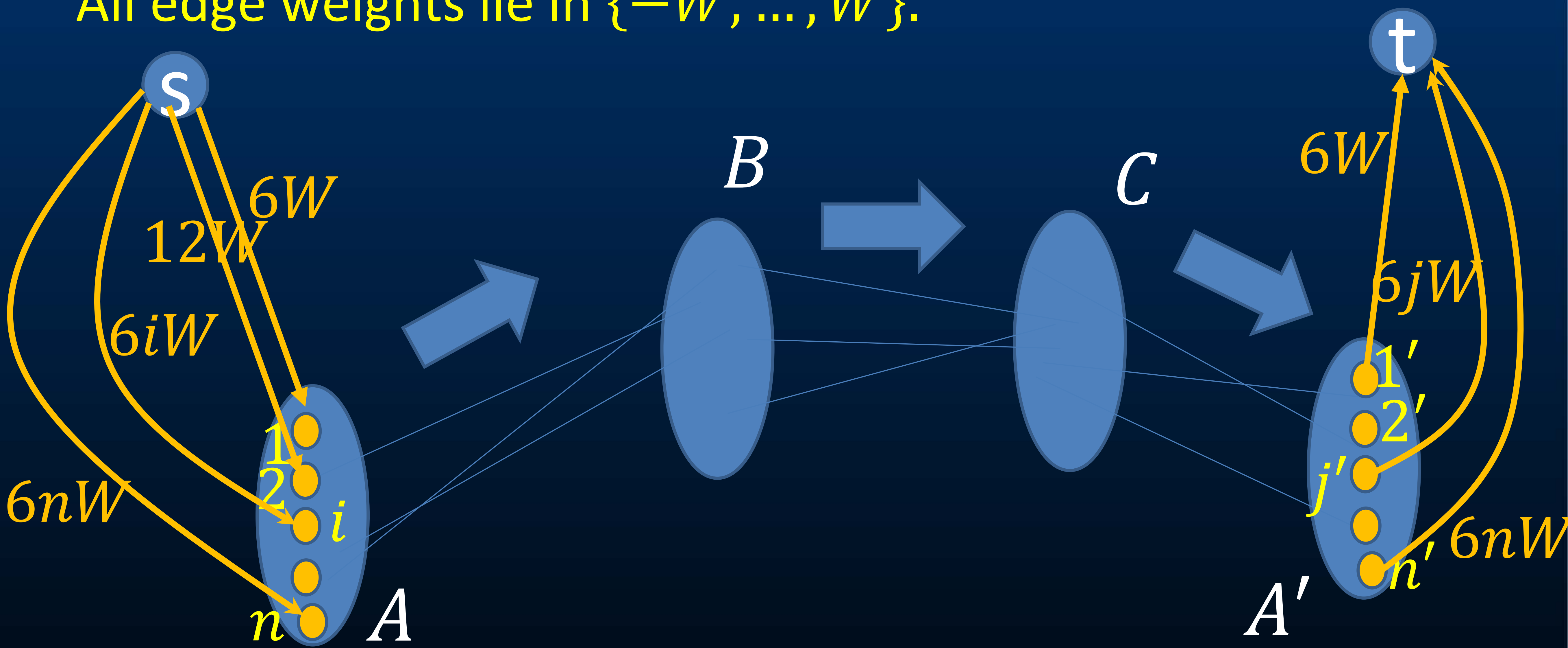
All edge weights lie in $\{-W, \ldots, W\}$.



Given directed layered G' with parts A,B,C, A' and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

All edge weights lie in $\{-W, \dots, W\}$.



Claim: $d(s, t) = 12W +$ distance between 1 in A and 1' in A'.

Pf: If $i > 1$ or $j > 1$, dist through i, j' is $\geq 18W - 3W = 15W$.

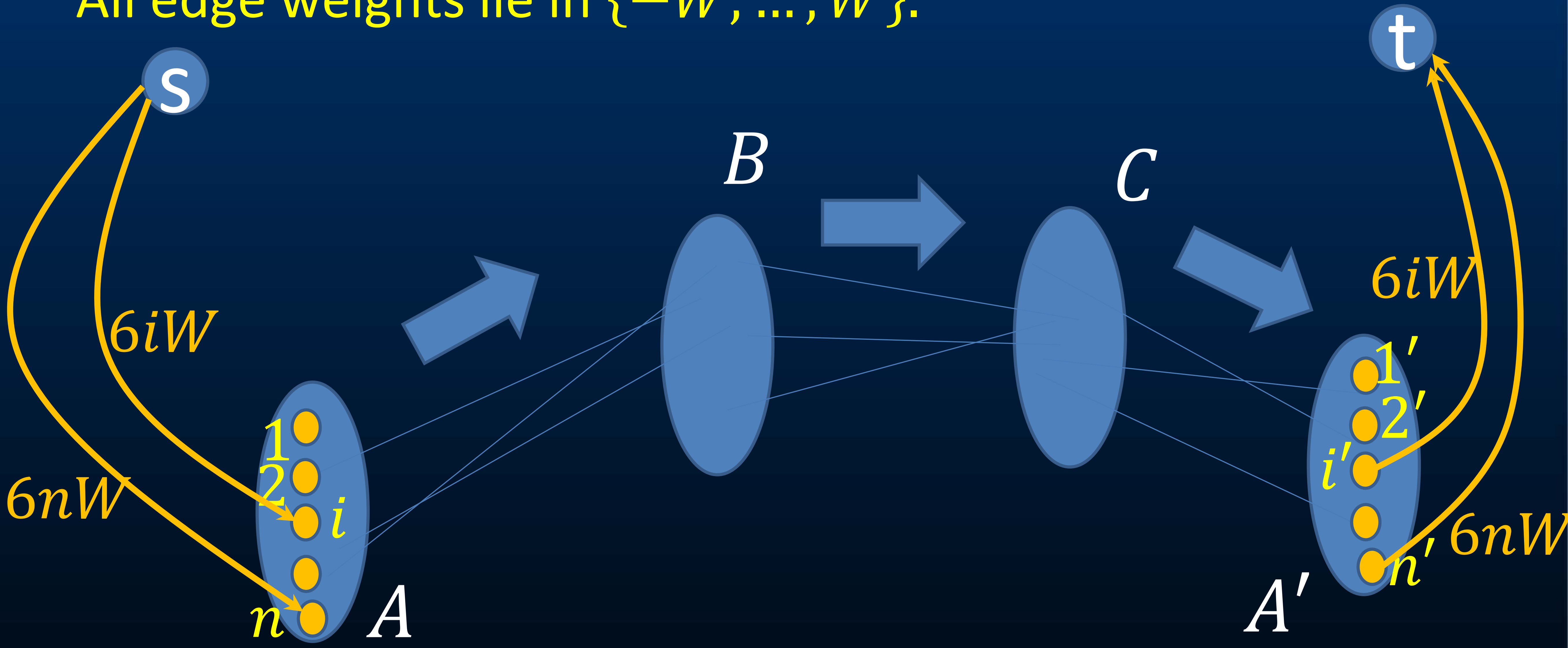
Dist through $1, 1'$ is $\leq 12W + 3W = 15W$.

Given directed layered G' with parts A,B,C, A' and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

All edge weights lie in $\{-W, \ldots, W\}$.

Remove $(s,j),(j',t)$ for all $j < i$.

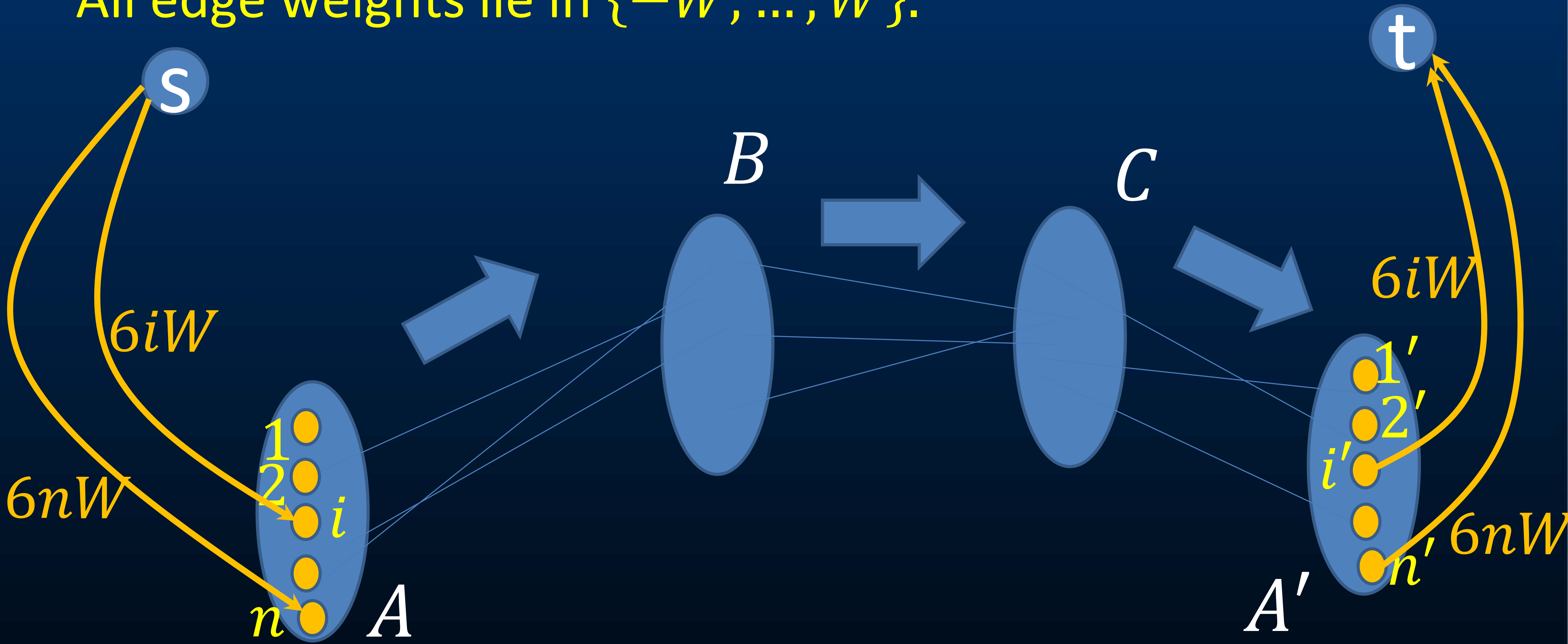


Given directed layered G’ with parts A,B,C, A’ and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

All edge weights lie in $\{-W, \ldots, W\}$.

Remove $(s, j), (j', t)$ for all $j < i$.



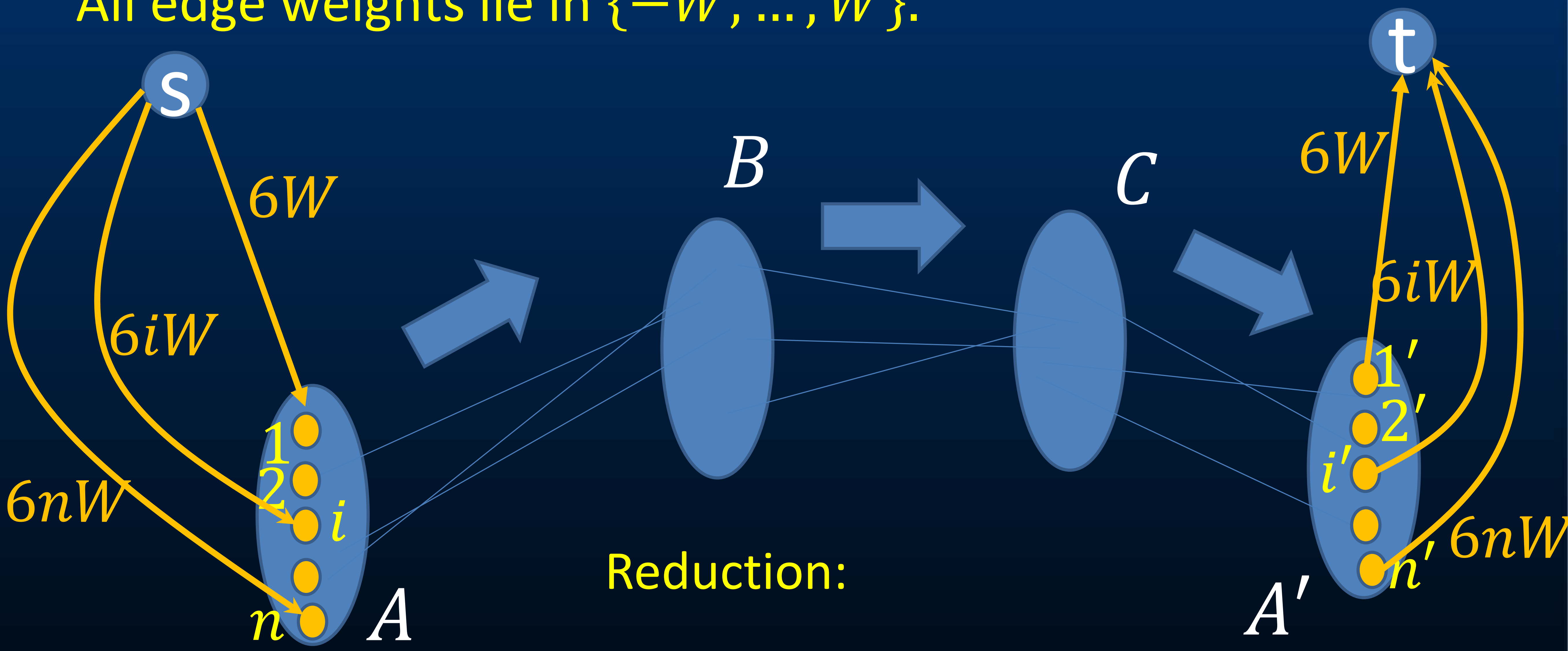
Claim: $d(s, t) = 12iW +$ distance between i in A and i' in A’.

Pf: If a $> i$ or $b > i$, dist through a, b' is $\geq 12iW + 6W - 3W = (12i + 3)W$. Dist through i, i' is $\leq 12iW + 3W$.

Given directed layered G' with parts A,B,C, A' and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

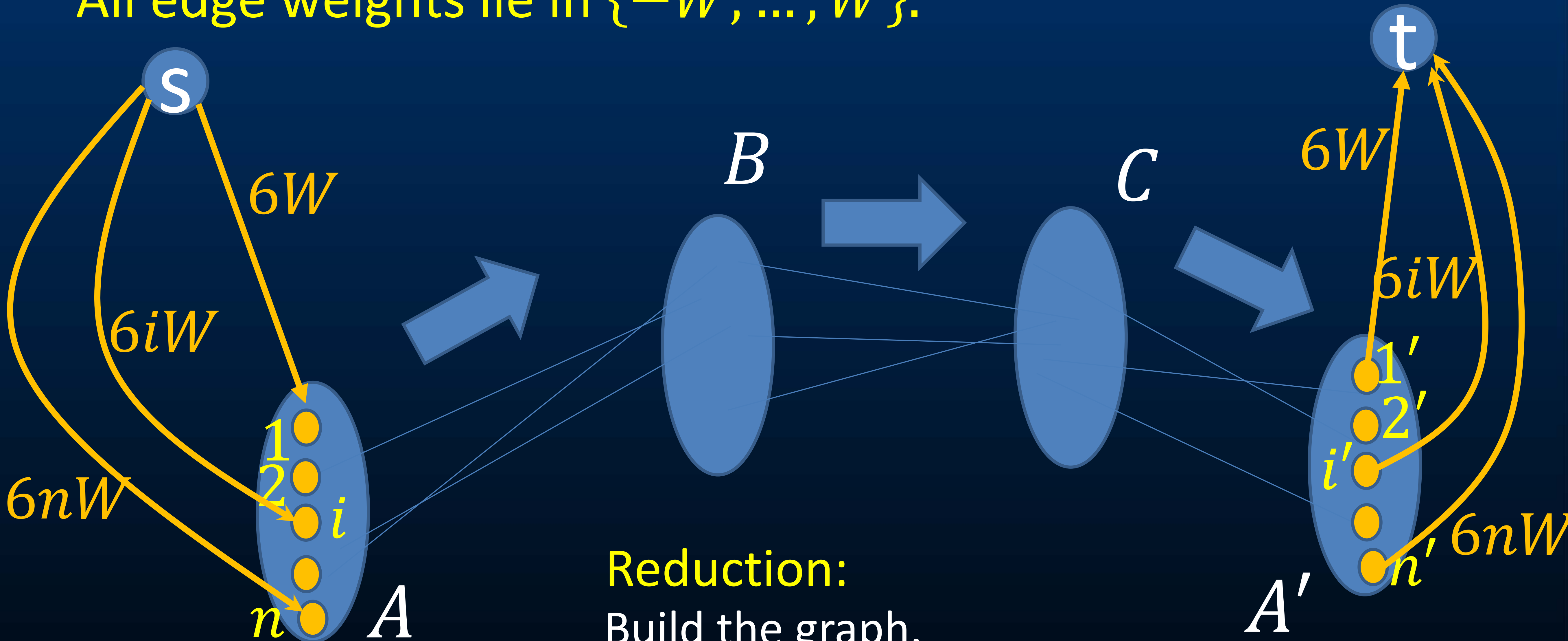
All edge weights lie in $\{-W, \dots, W\}$.



Given directed layered G’ with parts A,B,C, A’ and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

All edge weights lie in $\{-W, \dots, W\}$.



Reduction:

Build the graph.

For i from 1 to n:

  if $d(s,t) < 12iW$, return “Neg Triangle!”

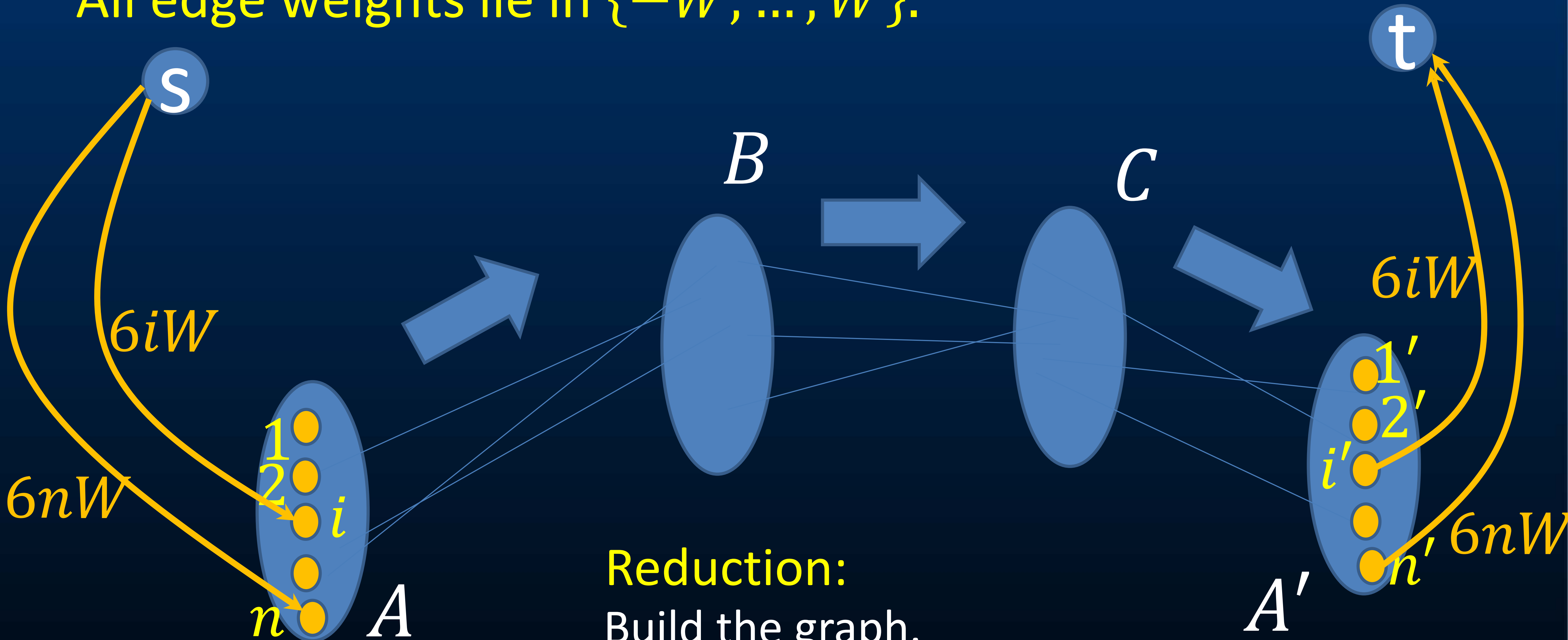
  else remove edges (s, i) and (i', t)

Return “No Neg Triangle!”

Given directed layered G’ with parts A,B,C, A’ and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

All edge weights lie in $\{-W, \ldots, W\}$.



Reduction:

Build the graph.

For i from 1 to n:

  if $d(s,t) < 12iW$, return “Neg Triangle!”

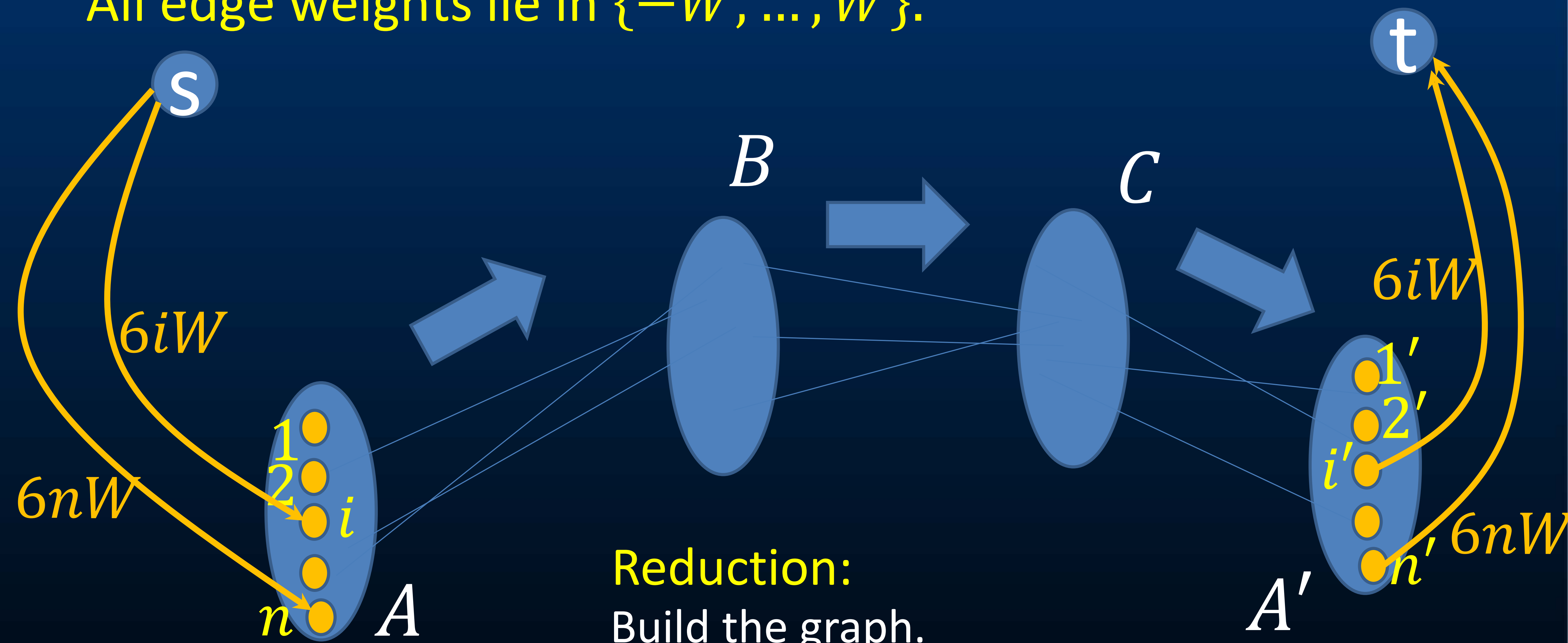
  else remove edges (s, i) and (i', t)

Return “No Neg Triangle!”

Given directed layered G’ with parts A,B,C, A’ and want to know if $\exists a \in A, b \in B, c \in C, a' \in A'$:

$$a = a', d(a, a') < 0.$$

All edge weights lie in $\{-W, \ldots, W\}$.



Claim: $d(s,t) = 12iW +$ distance between i in A and i' in A’.

Reduction:

Build the graph.

For i from 1 to n:

  if $d(s,t) < 12iW$, return “Neg Triangle!”

  else remove edges (s, i) and (i', t)

Return “No Neg Triangle!”

Theorem [RZ'04, A VW'14]
If s-t Shortest Path in dense m edge graphs can be supported with $O(m^{1-\epsilon})$ time per update, after $O(n^{3-\epsilon})$ preprocessing time for $\epsilon > 0$, then APSP in n node graphs is in $O(n^{3-\epsilon})$ time.

The graph we build has $N = O(n)$ nodes and $M = O(n^2)$ edges.
We then perform $2n$ deletions.

If s-t Shortest Path preprocessing is $O(N^{3-\epsilon})$ time, the amortized deletion time is $O(M^{1-\epsilon}) = O(n^{2-2\epsilon})$, then we can solve Neg. Triangle in $O(n^{3-\epsilon})$ time.

Exercise: Show how to modify the reduction so that it works for undirected graphs as well.

Summary:

Very high lower bounds for fundamental problems

After identifying the conjecture,
the proofs are often very simple!