Simple Length-Constrained Minimum Spanning Trees

Ellis Hershkowitz & Richard Huang
Brown University

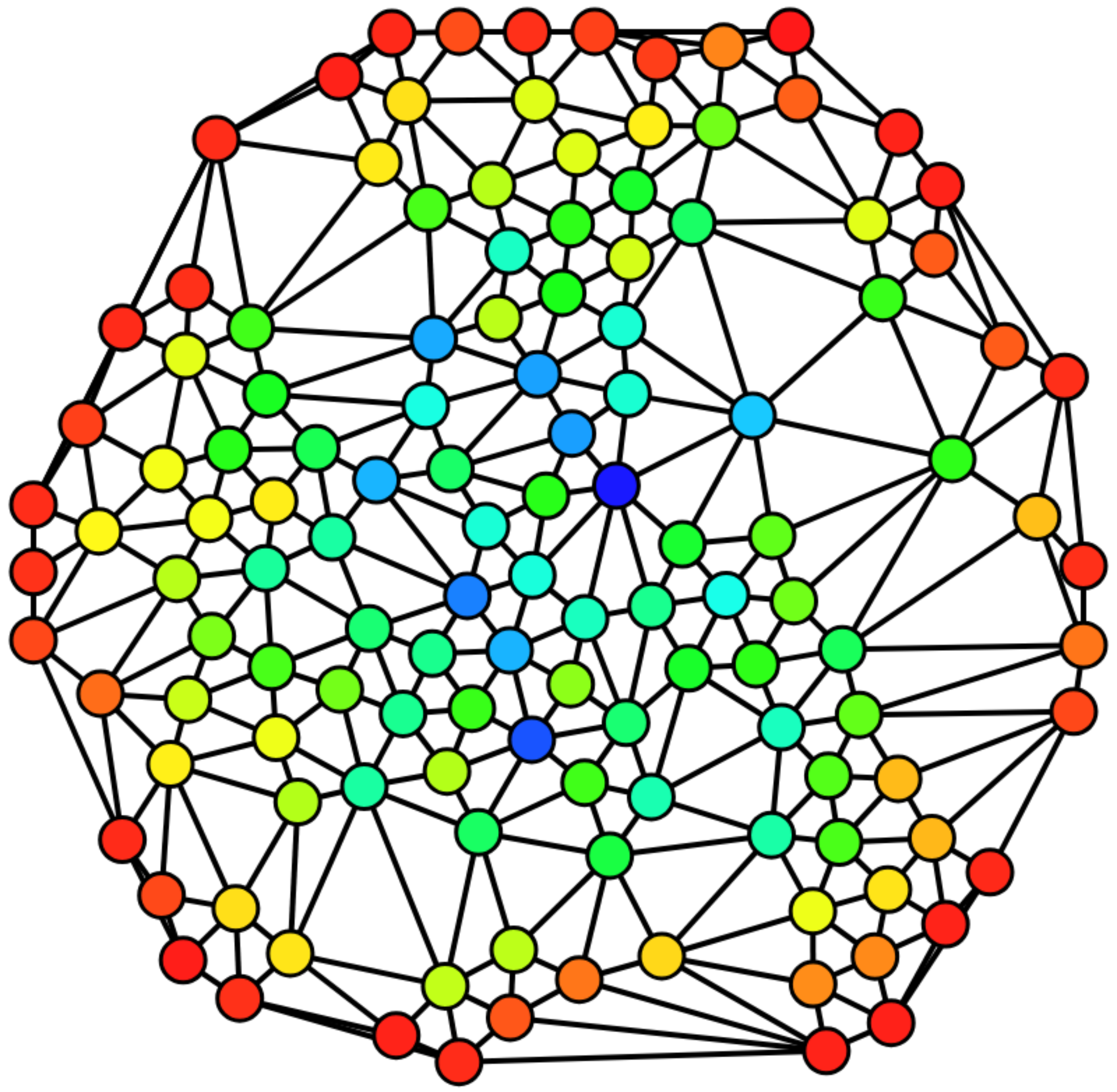
Minimum Spanning Tree (MST)

Minimum Spanning Tree (MST)

- Given a (connected) edge-weighted graph, find a spanning tree that minimizes the sum of edge weights

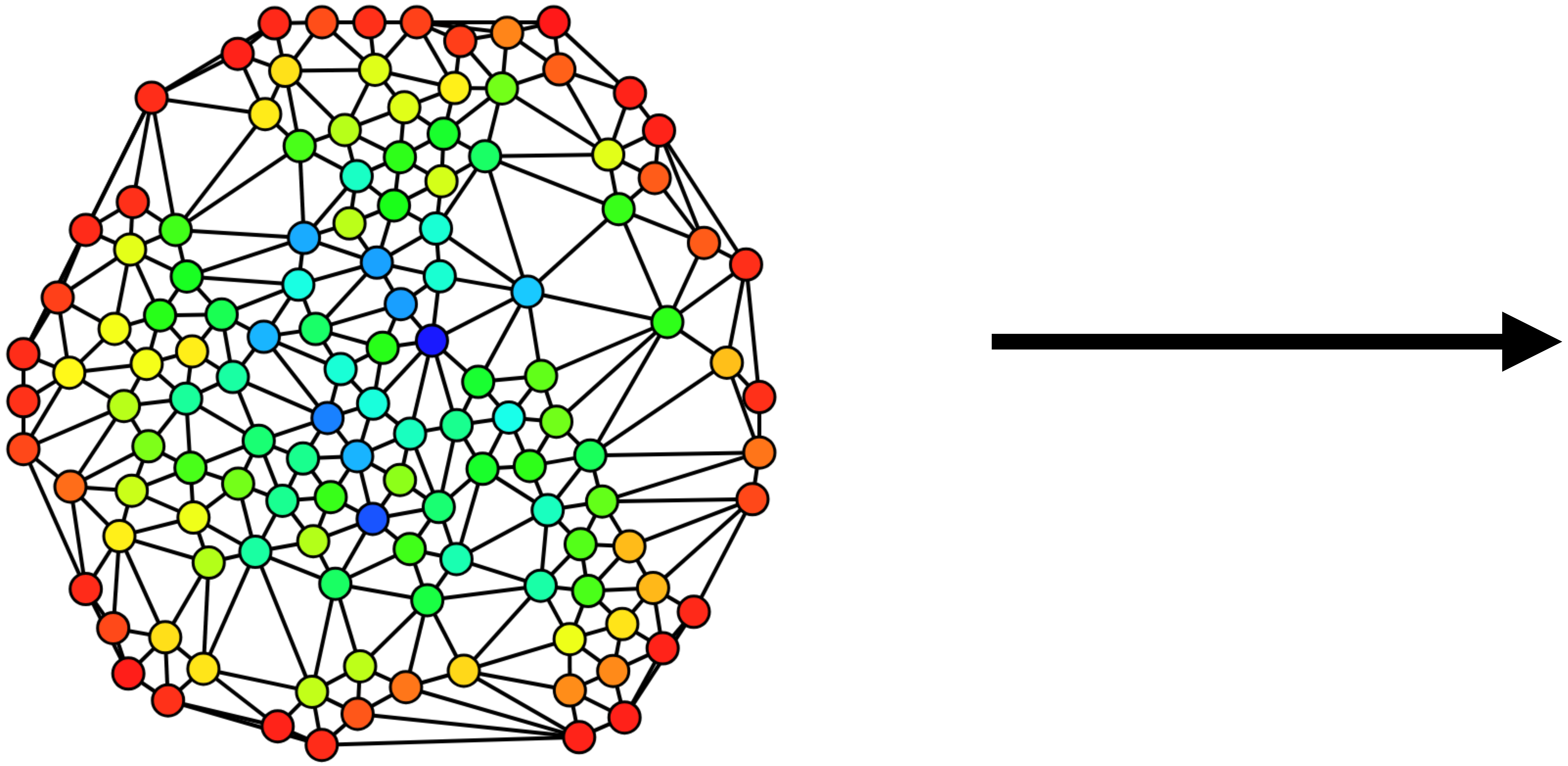
Minimum Spanning Tree (MST)

- Given a (connected) edge-weighted graph, find a spanning tree that minimizes the sum of edge weights



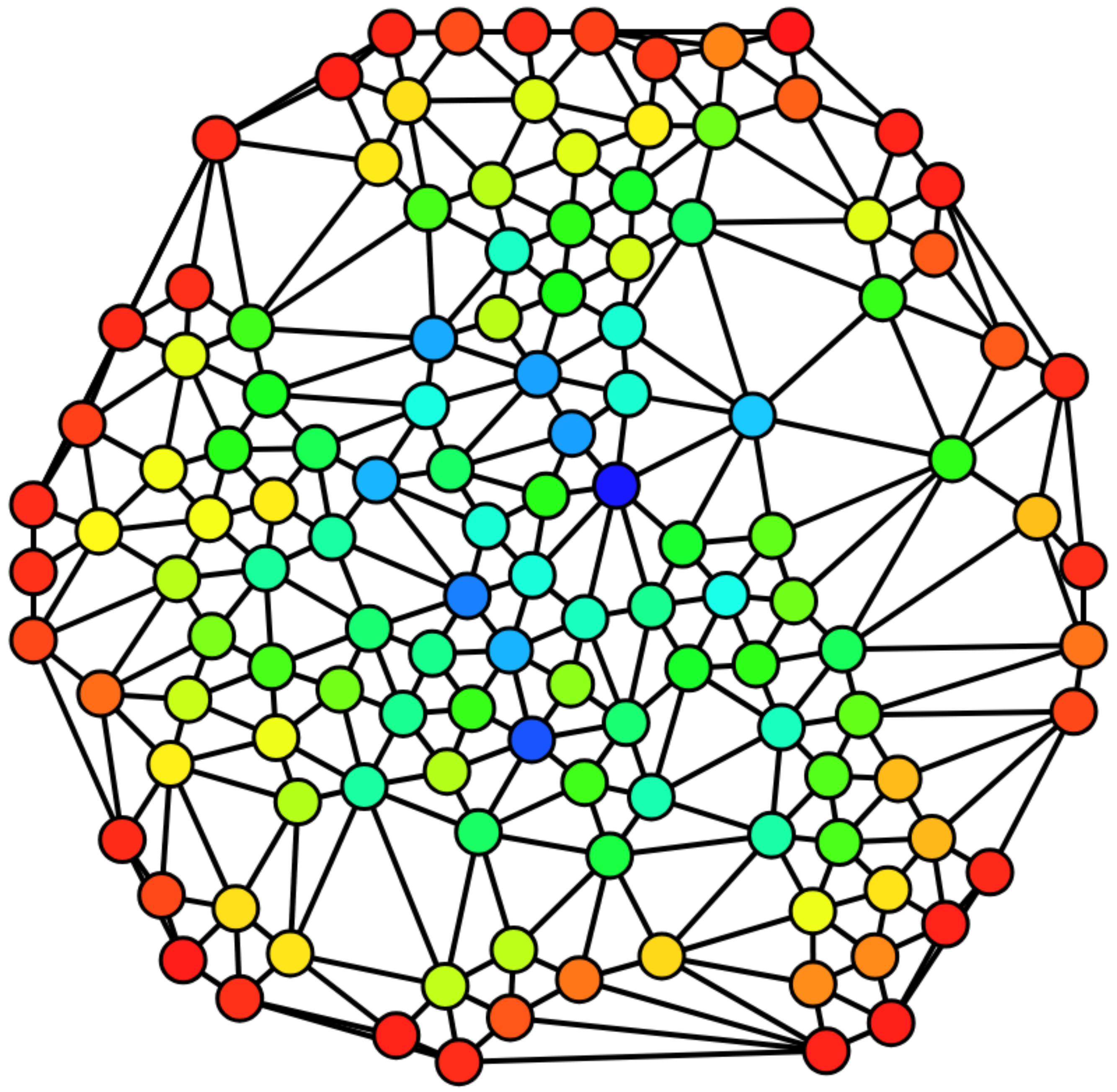
Minimum Spanning Tree (MST)

- Given a (connected) edge-weighted graph, find a spanning tree that minimizes the sum of edge weights



Minimum Spanning Tree (MST)

- Given a (connected) edge-weighted graph, find a spanning tree that minimizes the sum of edge weights



Length-Constrained
Minimum Spanning Tree (MST)

Length-Constrained Minimum Spanning Tree (MST)

- Introduce an edge-length function and a length (diameter) constraint

Length-Constrained Minimum Spanning Tree (MST)

- Introduce an edge-length function and a length (diameter) constraint
- Given another input $L > 0$ find a spanning tree with diameter at most L that minimizes the sum of edge weights

Length-Constrained Minimum Spanning Tree (MST)

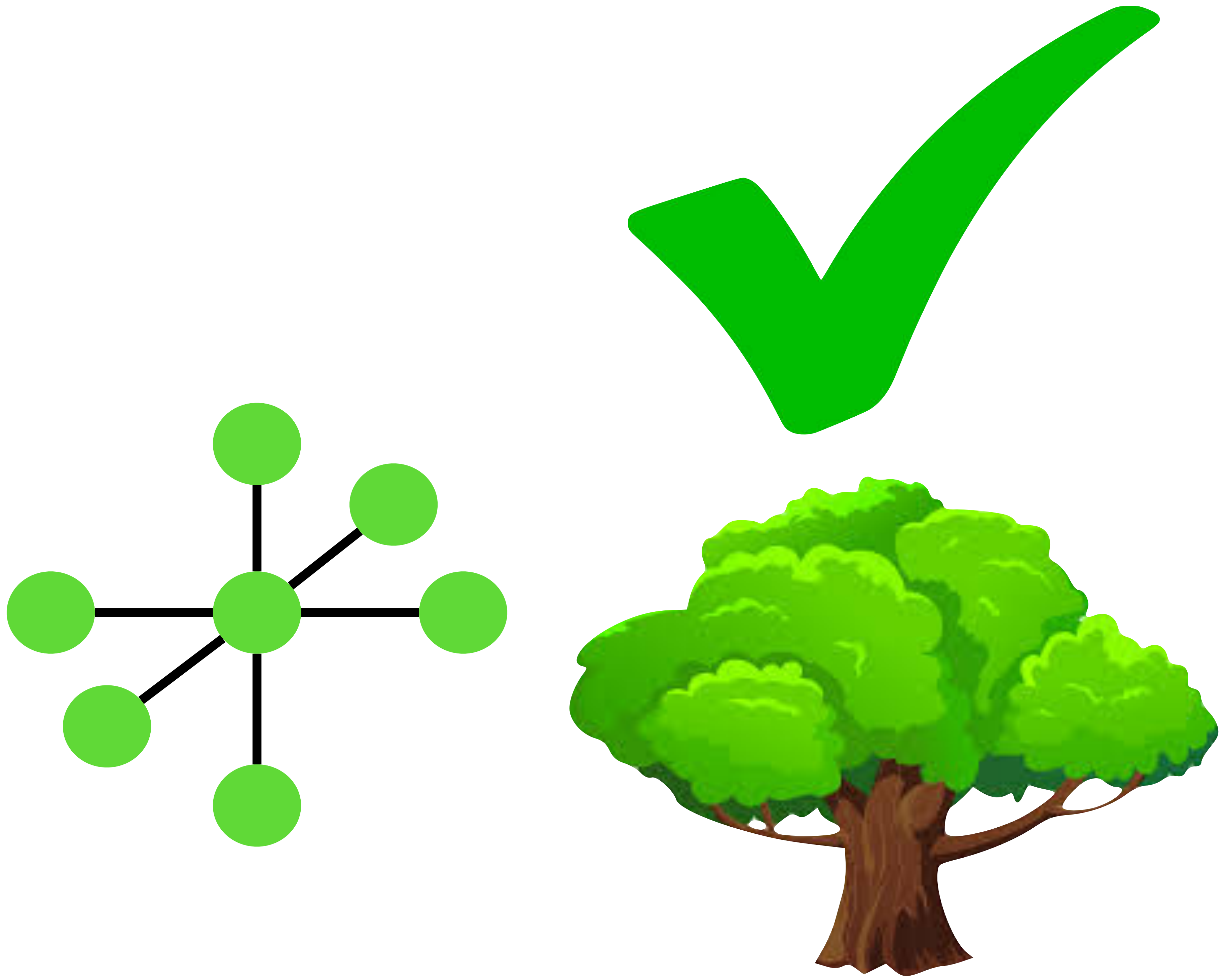
- Introduce an edge-length function and a length (diameter) constraint
- Given another input $L > 0$ find a spanning tree with diameter at most L that minimizes the sum of edge weights

Length-Constrained
Minimum Spanning Tree (MST)

- Introduce an edge-length function and a length (diameter) constraint
- Given another input $L > 0$ find a spanning tree with diameter at most L that minimizes the sum of edge weights

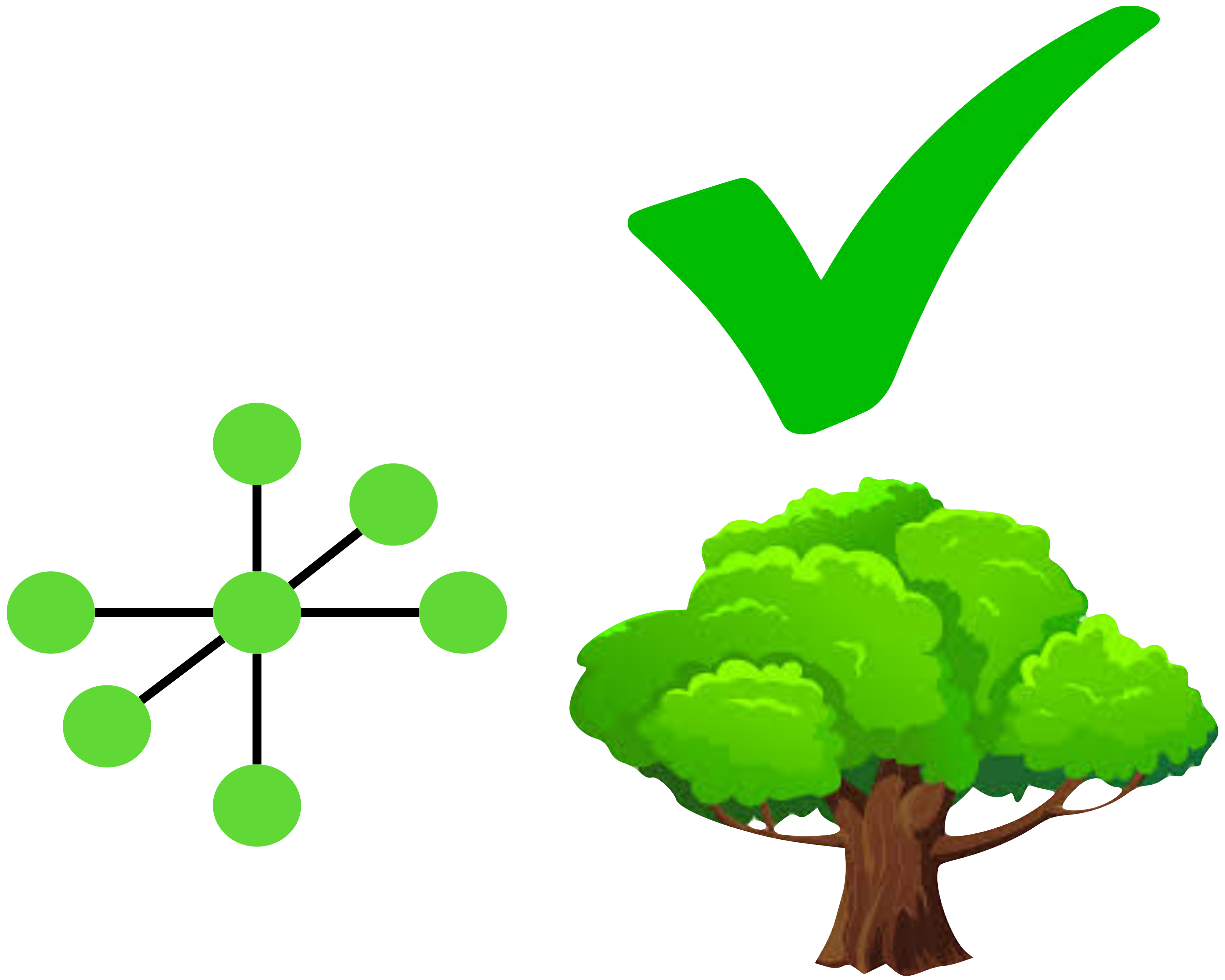
Length-Constrained
Minimum Spanning Tree (MST)

- Introduce an edge-length function and a length (diameter) constraint
- Given another input $L > 0$ find a spanning tree with diameter at most L that minimizes the sum of edge weights



Length-Constrained
Minimum Spanning Tree (MST)

- Introduce an edge-length function and a length (diameter) constraint
- Given another input $L > 0$ find a spanning tree with diameter at most L that minimizes the sum of edge weights



Length-Constrained Minimum Spanning Tree (MST)

- NP-hard; there mainly exist **bicriteria** approximation algorithms

Length-Constrained Minimum Spanning Tree (MST)

- NP-hard; there mainly exist **bicriteria** approximation algorithms
  - Let OPT_L be the weight of a min-weight spanning tree with diameter L

Length-Constrained Minimum Spanning Tree (MST)

- NP-hard; there mainly exist **bicriteria** approximation algorithms
  - Let OPT_L be the weight of a min-weight spanning tree with diameter L
  - Need to approximate L **and** OPT_L

Our Result

Our Result

Our Result

There is a (simple) algorithm that given any $\epsilon \geq 1/\text{poly}(n)$,

Our Result

There is a (simple) algorithm that given any $\epsilon \geq 1/\text{poly}(n)$,

outputs a spanning tree with

Our Result

There is a (simple) algorithm that given any $\epsilon \geq 1/\text{poly}(n)$,

outputs a spanning tree with

length: $O(1/\epsilon) \cdot L$

Our Result

There is a (simple) algorithm that given any $\epsilon \geq 1/\text{poly}(n)$,

outputs a spanning tree with

length: $O(1/\epsilon) \cdot L$

weight: $O(n^{\epsilon}/\epsilon) \cdot \text{OPT}_L$

Our Result

There is a (simple) algorithm that given any $\epsilon \geq 1/\text{poly}(n)$,

outputs a spanning tree with

length: $O(1/\epsilon) \cdot L$

weight: $O(n^\epsilon/\epsilon) \cdot \text{OPT}_L$

(with high probability)

Different ϵ's give different results:

ϵ	Length	Weight

Different ϵ's give different results:

ϵ	**Length**	**Weight**
$1/\log n$	$O(\log n)$	$O(\log n)$

Different ϵ's give different results:

ϵ	**Length**	**Weight**
$1/\log n$	$O(\log n)$	$O(\log n)$
$1/c$	$O(1)$	$O(n^{1/c})$

Different ϵ's give different results:

ϵ	**Length**	**Weight**
$1/\log n$	$O(\log n)$	$O(\log n)$
$1/c$	$O(1)$	$O(n^{1/c})$
$\log \log n/\log n$	$o(\log n)$	poly$(\log n)$

Related Work

Related Work

Lower bound for length-constrained MST by (Naor/Schieber, 1997):

Related Work

Lower bound for length-constrained MST by (Naor/Schieber, 1997):

Related Work

Lower bound for length-constrained MST by (Naor/Schieber, 1997):

If you want to preserve L exactly,

Related Work

Lower bound for length-constrained MST by (Naor/Schieber, 1997):

If you want to preserve L exactly,

then you must pay an $\Omega(\log n)$ weight approximation.

Related Work

Weight	Length	Comments	Citation

Related Work

Weight	Length	Comments	Citation
$O(\log n)$	$O(\log n)$	Repeatedly computes min-weight max matchings (complicated)	Marathe/Ravi/Sundaram/ Ravi/Rosenkrantz/Hunt III, 1998

Related Work

Weight	Length	Comments	Citation
$O(\log n)$	$O(\log n)$	Repeatedly computes min-weight max matchings (complicated)	Marathe/Ravi/Sundaram/Ravi/Rosenkrantz/Hunt III, 1998
$O(n^{\epsilon} e^{1/\epsilon})$	1	Running time is $n^{1/\epsilon} \cdot \text{poly}(n)$	Kortsarz/Peleg, 1999

Related Work

Weight	Length	Comments	Citation
$O(\log n)$	$O(\log n)$	Repeatedly computes min-weight max matchings (complicated)	Marathe/Ravi/Sundaram/Ravi/Rosenkrantz/Hunt III, 1998
$O(n^{\epsilon} e^{1/\epsilon})$	1	Running time is $n^{1/\epsilon} \cdot \text{poly}(n)$	Kortsarz/Peleg, 1999
$O(n^{\epsilon}/\epsilon)$	$O(1/\epsilon)$	Cool	Us

Algorithm

Algorithm

Initially all vertices are **active**

Algorithm

Initially all vertices are **active**

For $O(1/\epsilon)$ rounds…

Algorithm

Initially all vertices are **active**

For $O(1/\epsilon)$ rounds…

1. Sample each active vertex independently with probability $n^{-\epsilon}$

Algorithm

Initially all vertices are **active**

For $O(1/\epsilon)$ rounds…

1. Sample each active vertex independently with probability $n^{-\epsilon}$
2. For each non-sampled vertex u, add the cheapest L-bounded path from u to a sampled vertex to our subgraph

Algorithm

Initially all vertices are **active**

For $O(1/\epsilon)$ rounds…

1. Sample each active vertex independently with probability $n^{-\epsilon}$
2. For each non-sampled vertex u, add the cheapest L-bounded path from u to a sampled vertex to our subgraph
3. Deactivate all non-sampled vertices

Algorithm

Initially all vertices are **active**

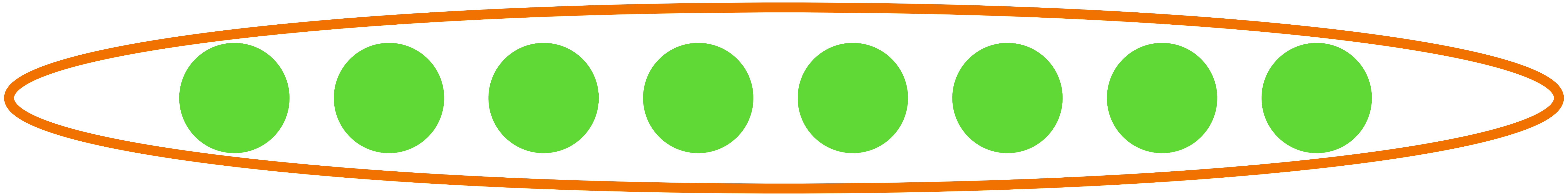
For $O(1/\epsilon)$ rounds…

1. Sample each active vertex independently with probability $n^{-\epsilon}$
2. For each non-sampled vertex u, add the cheapest L-bounded path from u to a sampled vertex to our subgraph
3. Deactivate all non-sampled vertices

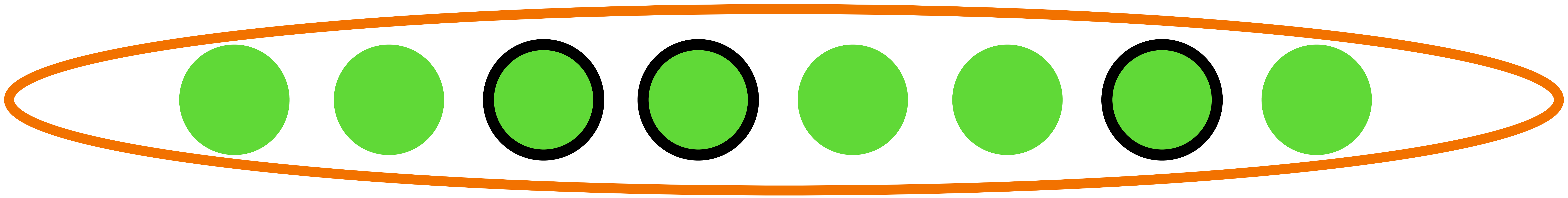
Return a shortest-path tree of our subgraph

Algorithm

Algorithm

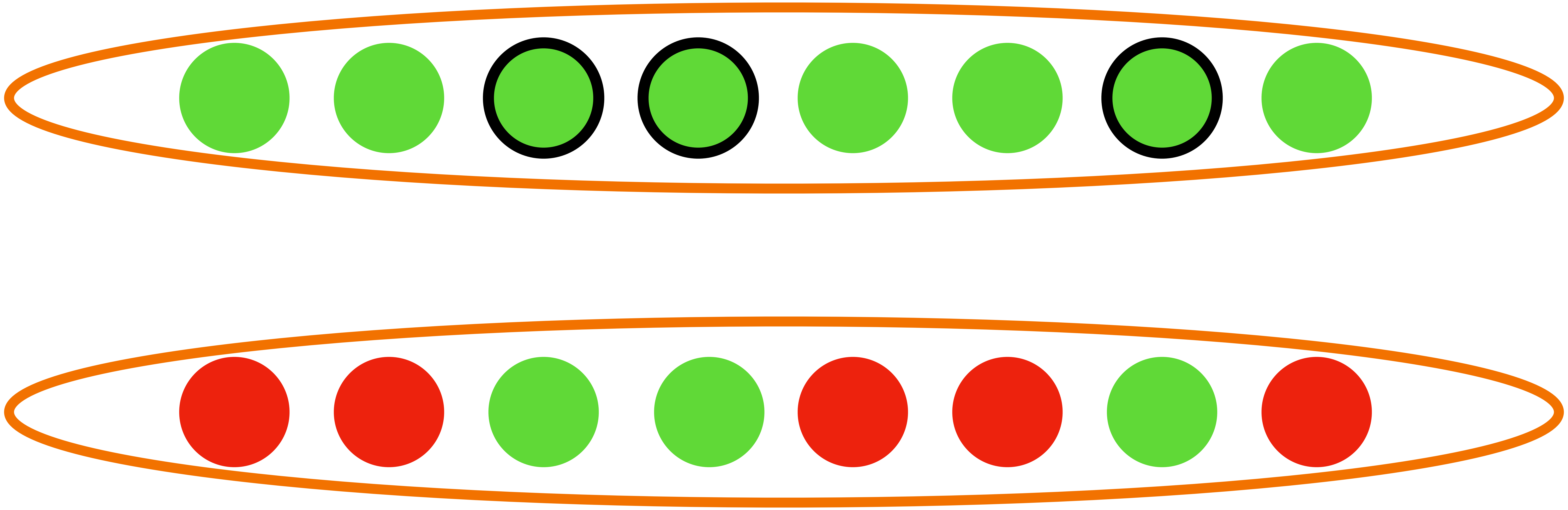


Algorithm



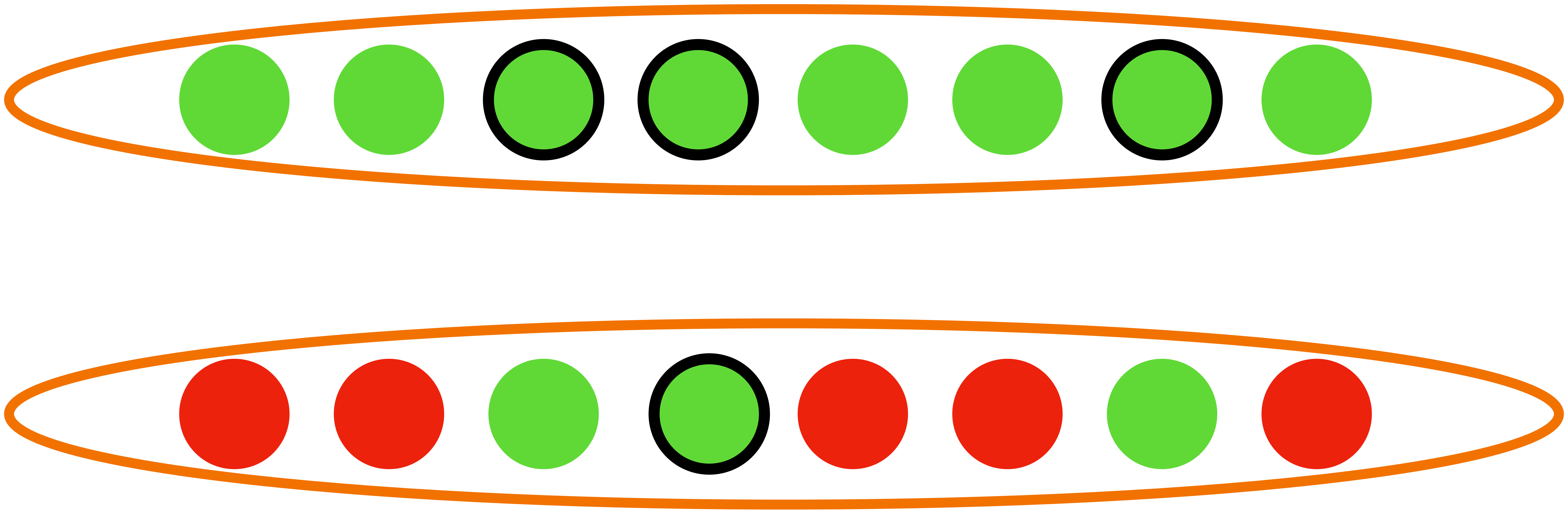
Algorithm

Red = deactivated
Bolded = sampled



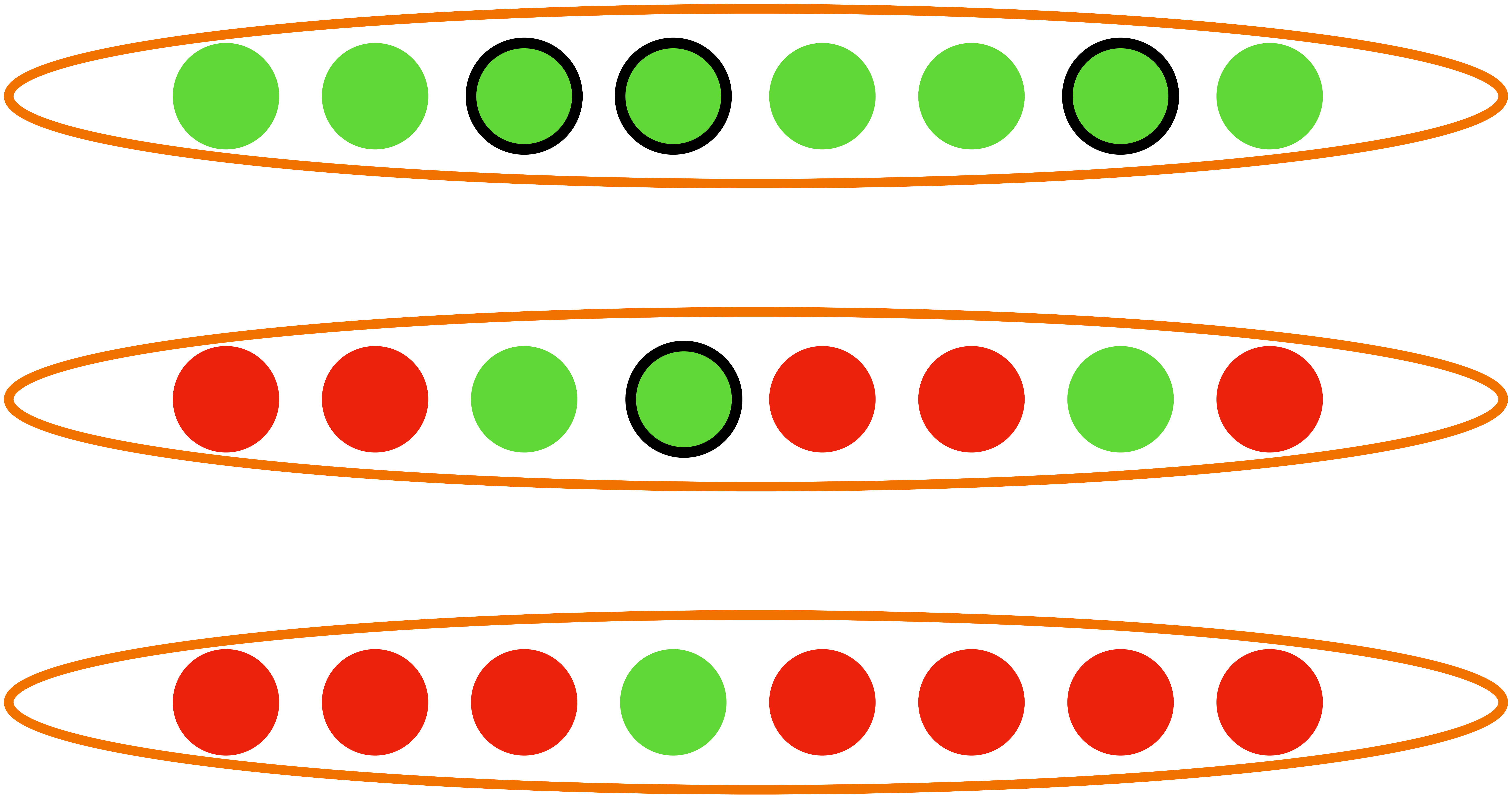
Algorithm

Red = deactivated
Bolded = sampled



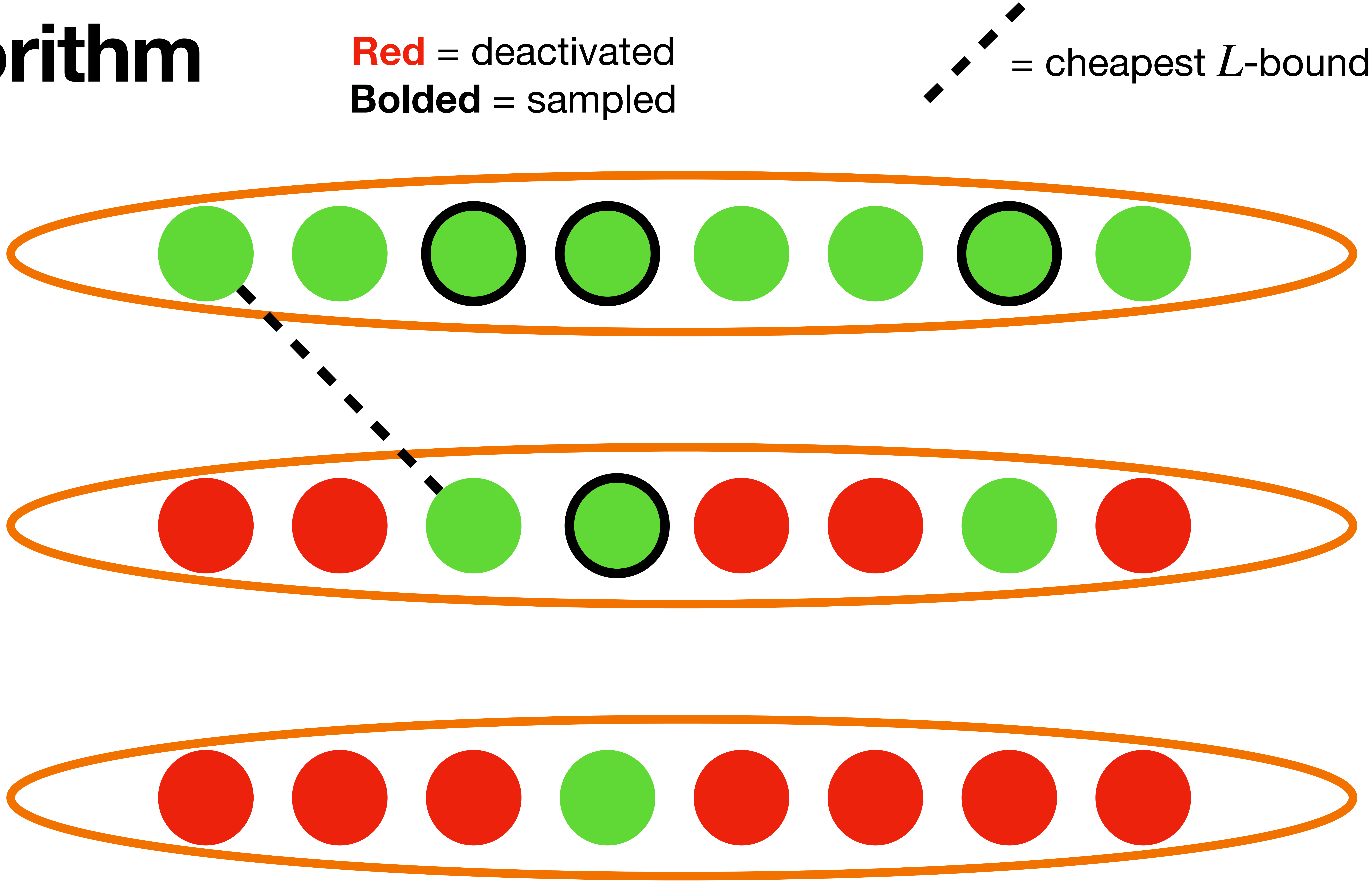
Algorithm

Red = deactivated
Bolded = sampled



Algorithm

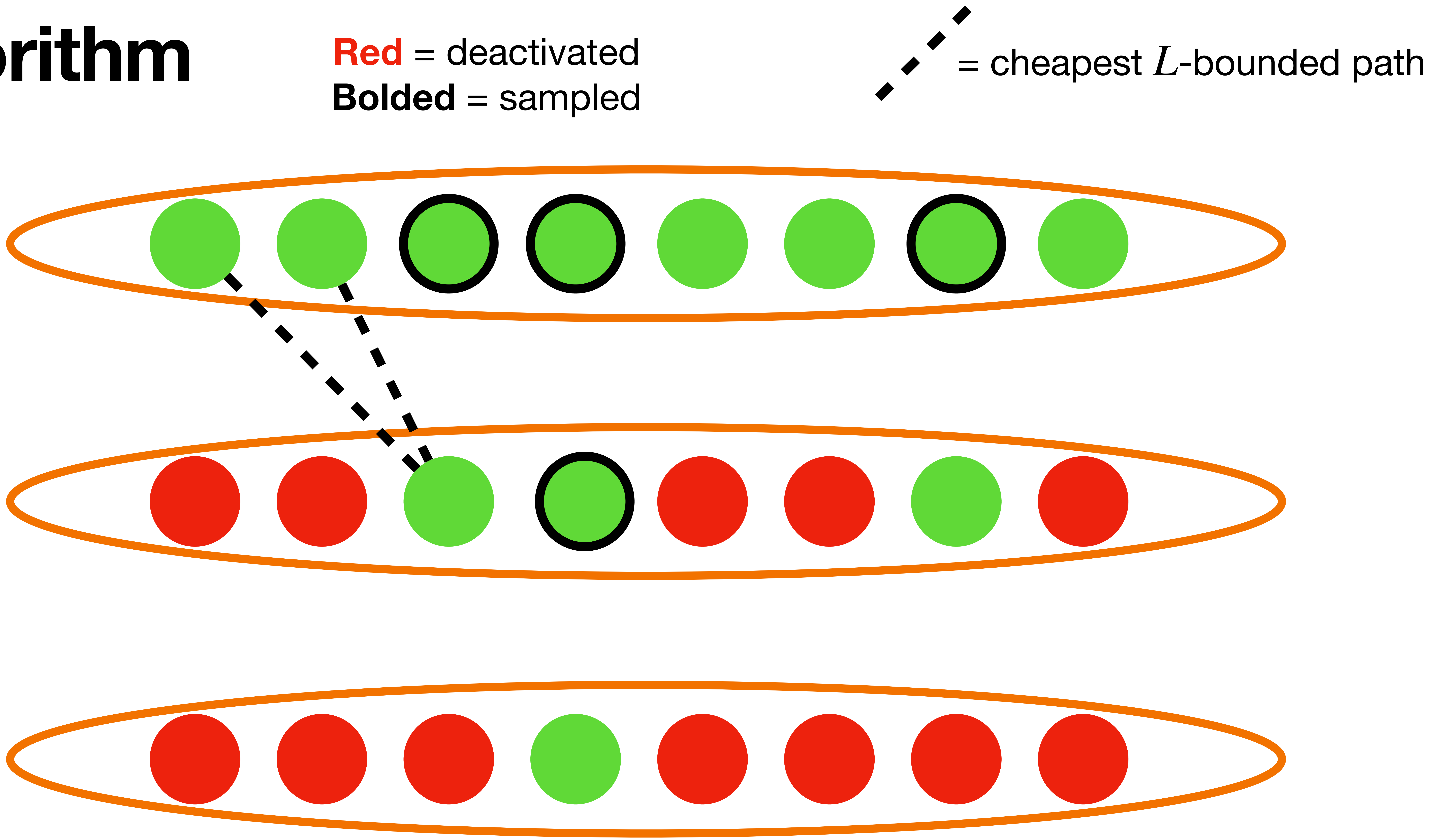
Red = deactivated
Bolded = sampled



= cheapest L-bounded path

Algorithm

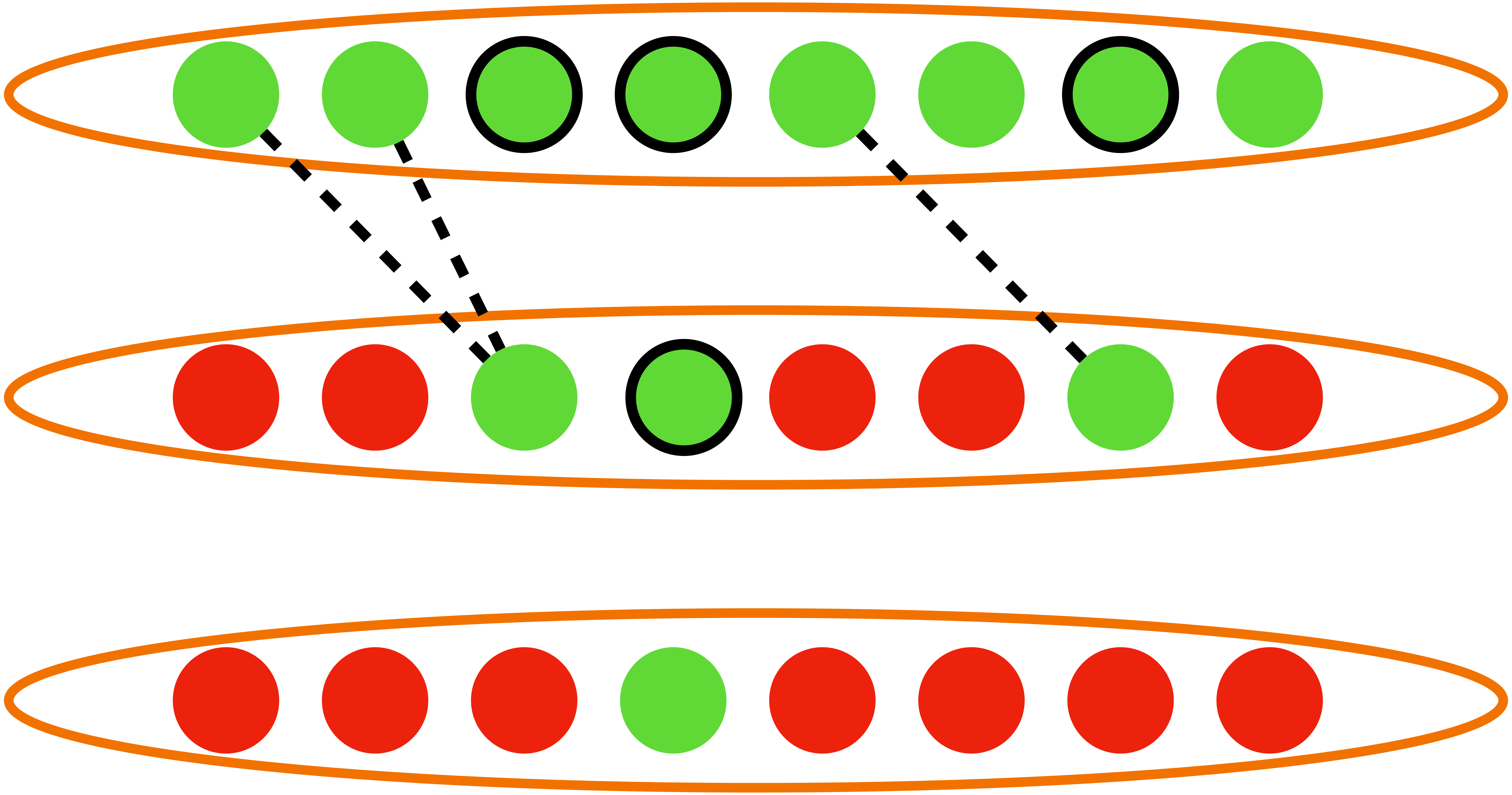
Red = deactivated
Bolded = sampled



Algorithm

Red = deactivated
Bolded = sampled

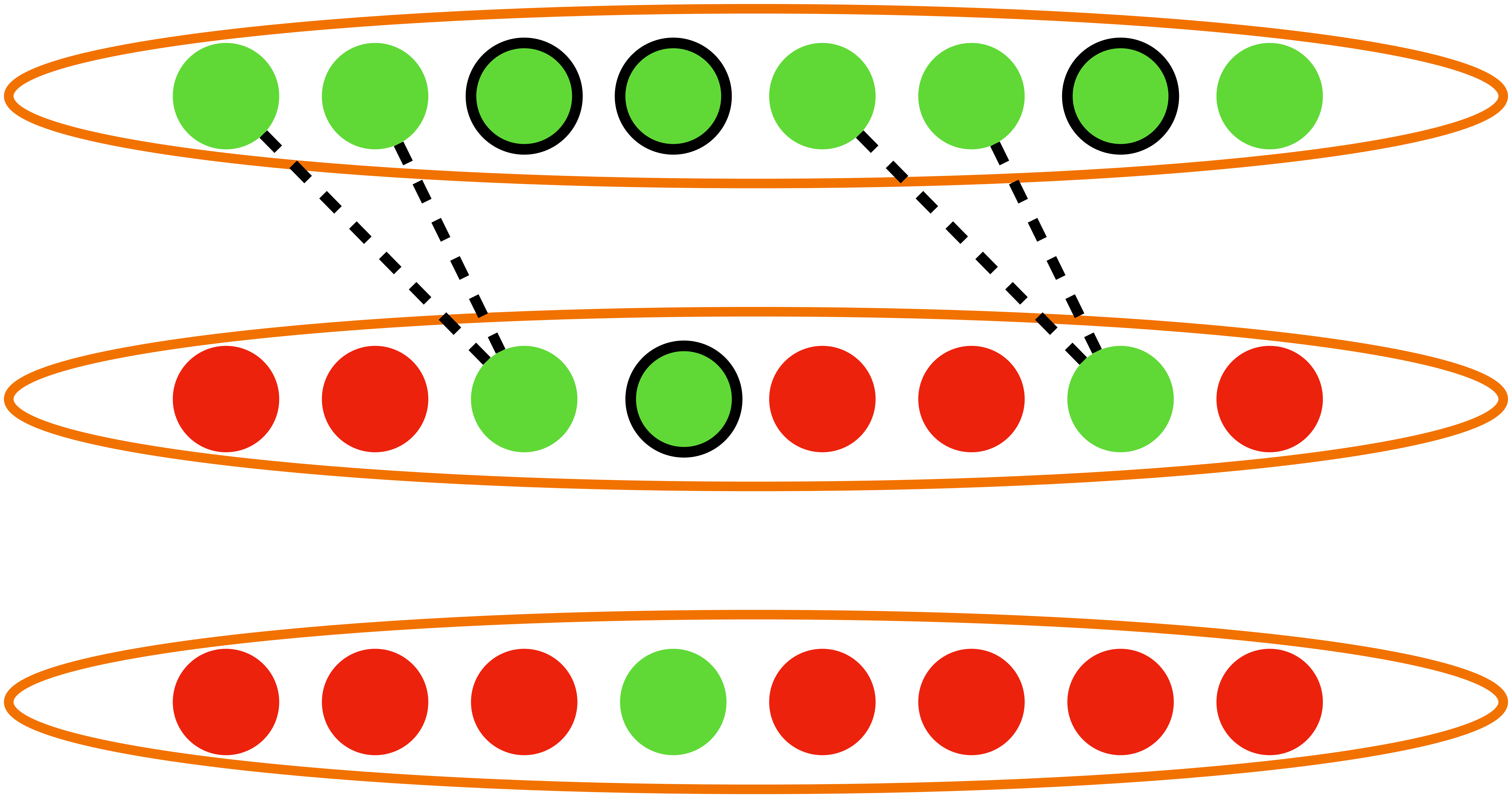
= cheapest L-bounded path



Algorithm

Red = deactivated
Bolded = sampled

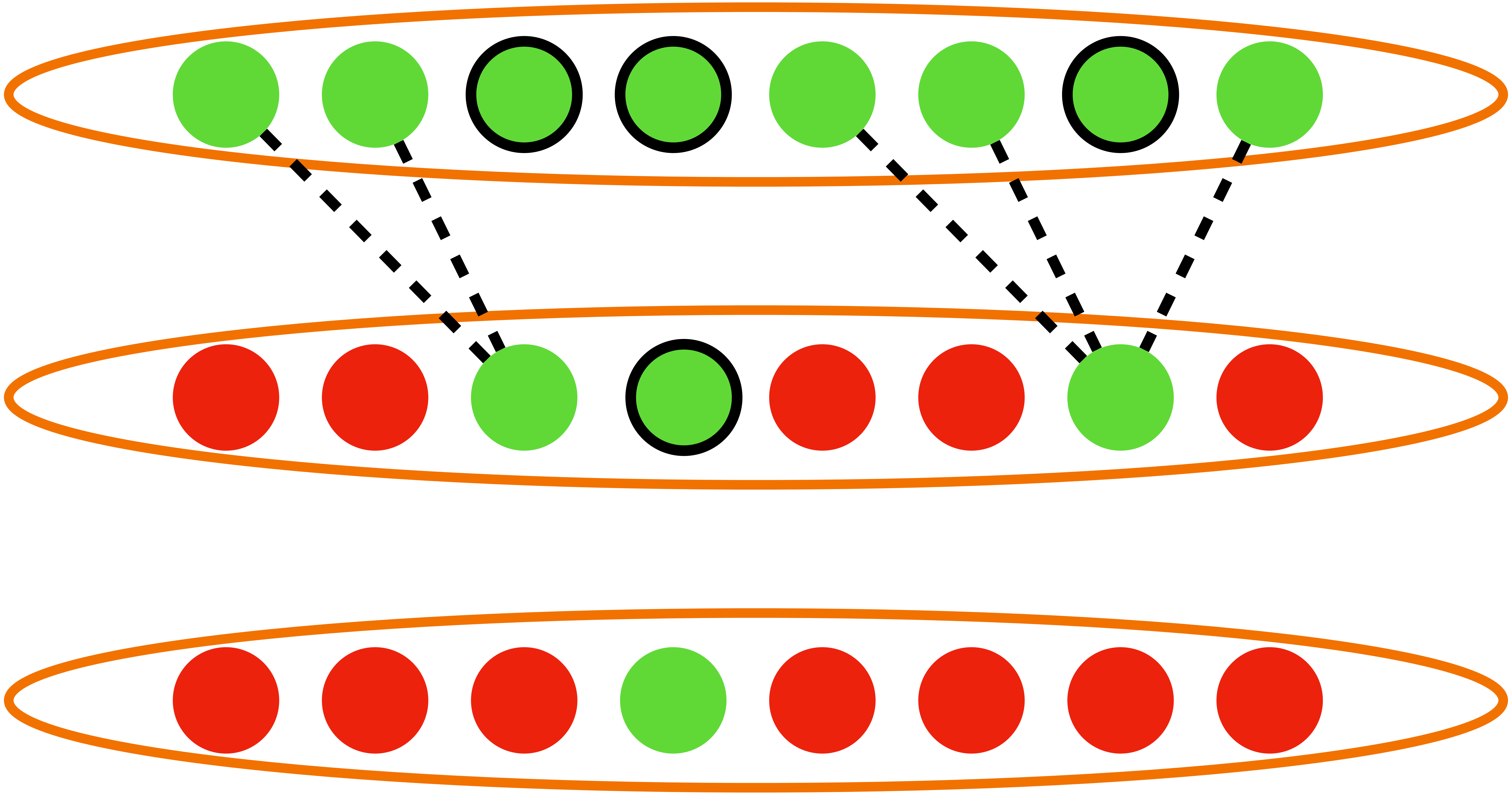
= cheapest L-bounded path



Algorithm

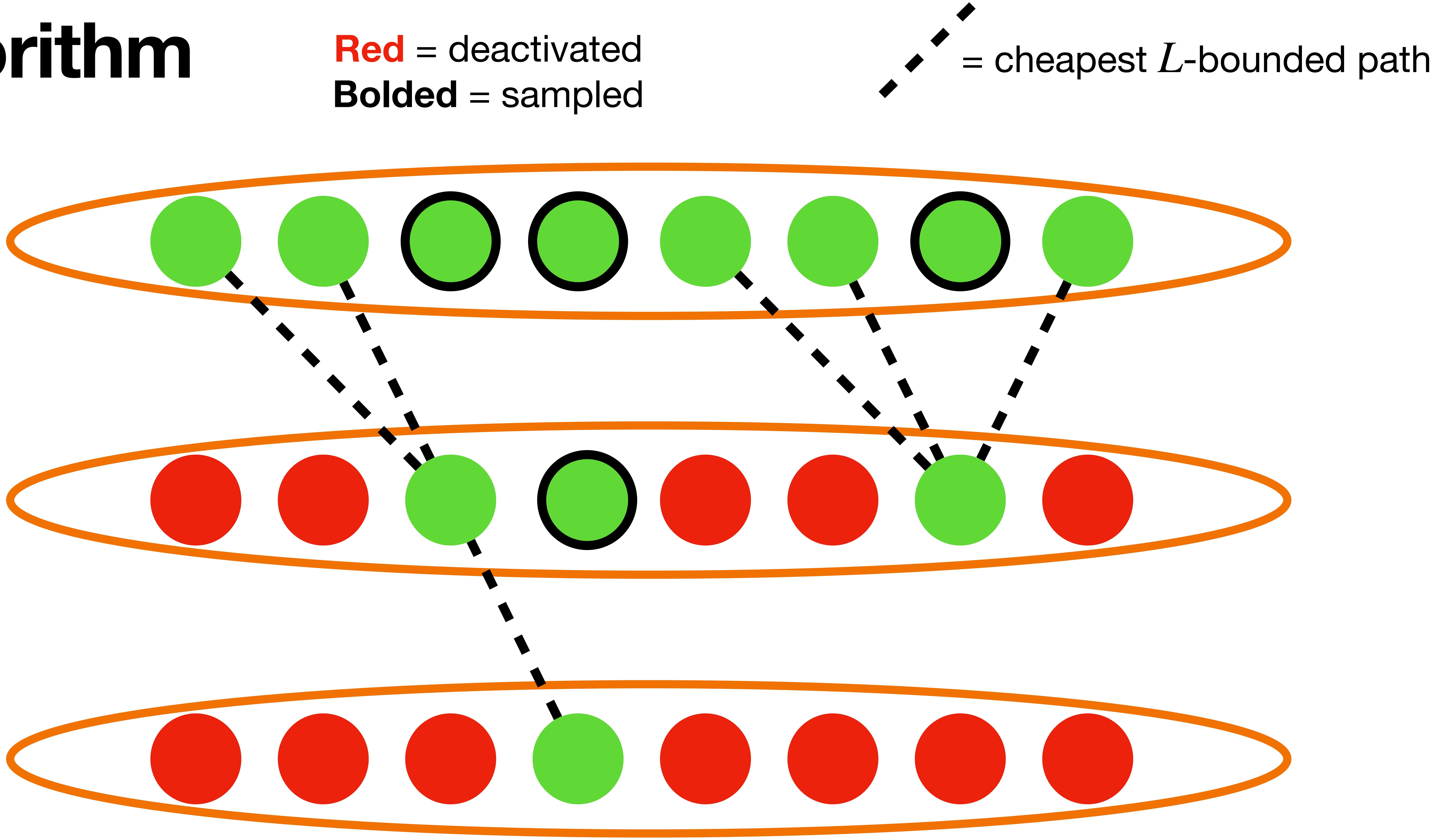
Red = deactivated
Bolded = sampled

= cheapest L-bounded path



Algorithm

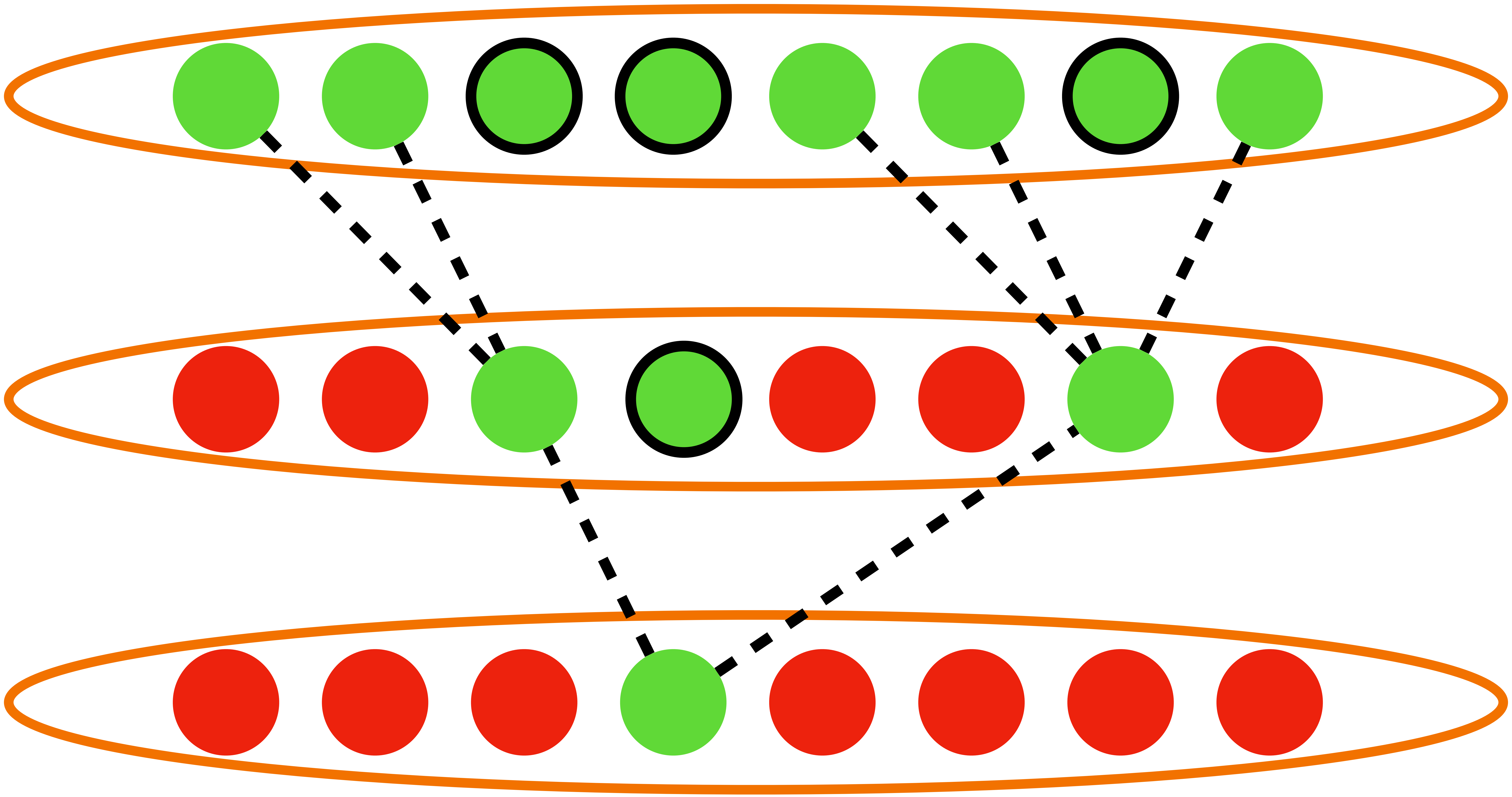
Red = deactivated
Bolded = sampled



Algorithm

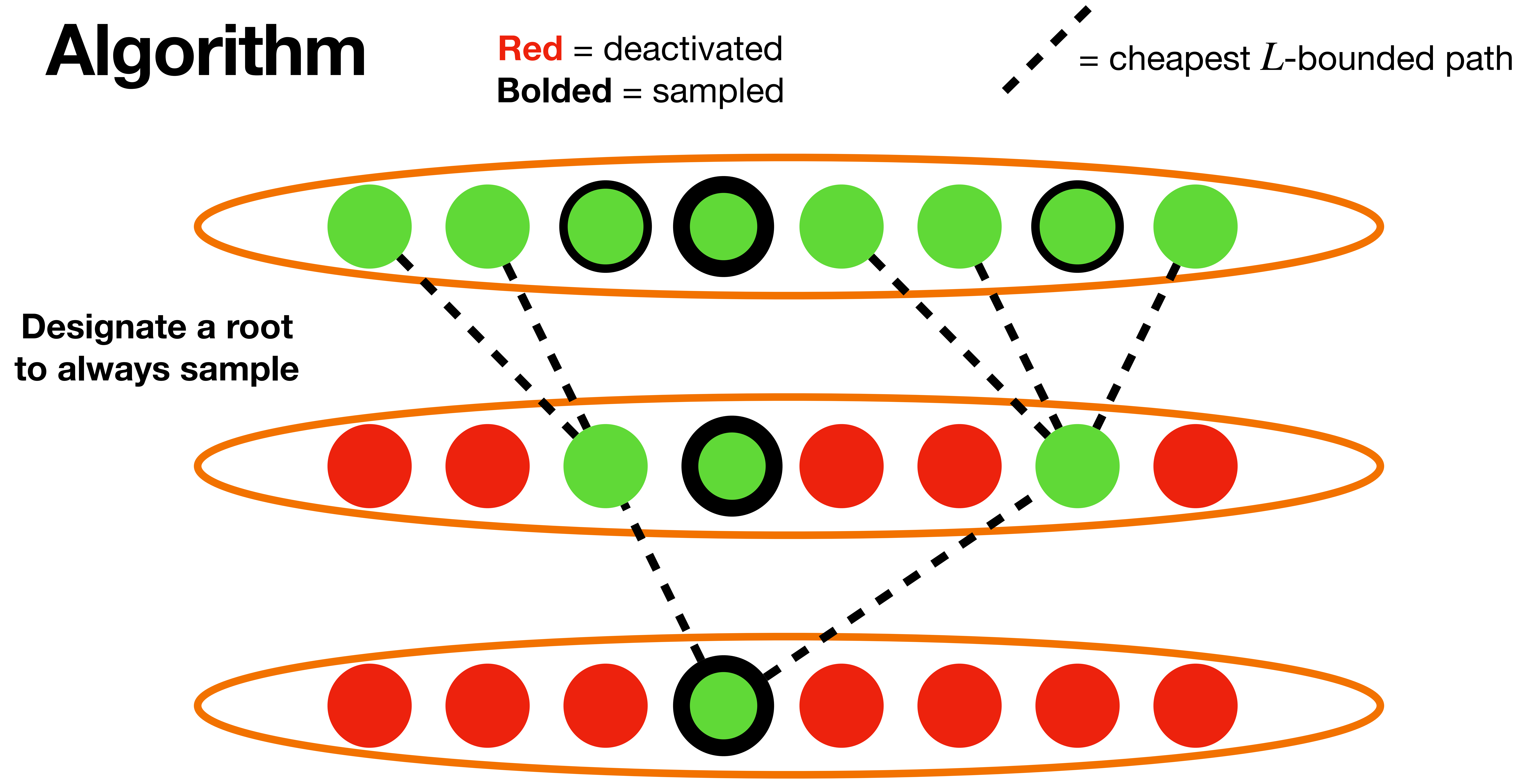
Red = deactivated
Bolded = sampled

= cheapest L-bounded path



Algorithm

Red = deactivated
Bolded = sampled



Is it feasible = Is it a tree?

Is it feasible = Is it a tree?

All vertices are deactivated after $O(1/\epsilon)$ rounds with high probability

Is it feasible = Is it a tree?

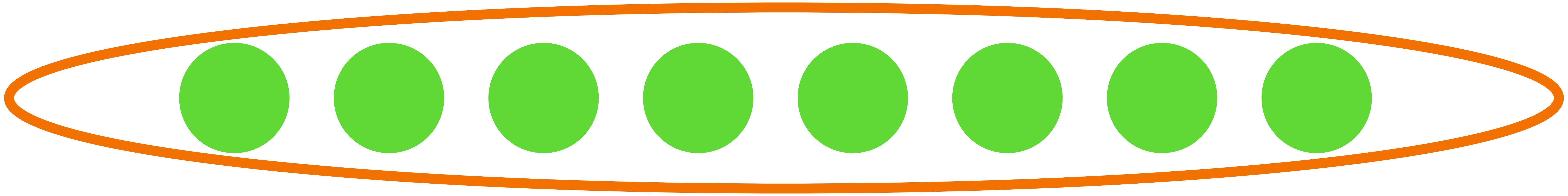
All vertices are deactivated after $O(1/\epsilon)$ rounds with high probability

Is it feasible = Is it a tree?

All vertices are deactivated after $O(1/\epsilon)$ rounds with high probability

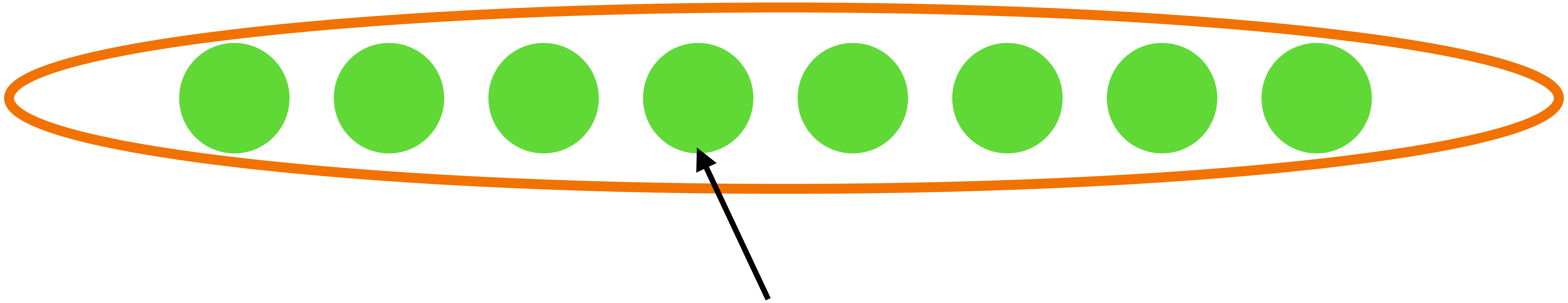
Is it feasible = Is it a tree?

All vertices are deactivated after $O(1/\epsilon)$ rounds with high probability



Is it feasible = Is it a tree?

All vertices are deactivated after $O(1/\epsilon)$ rounds with high probability



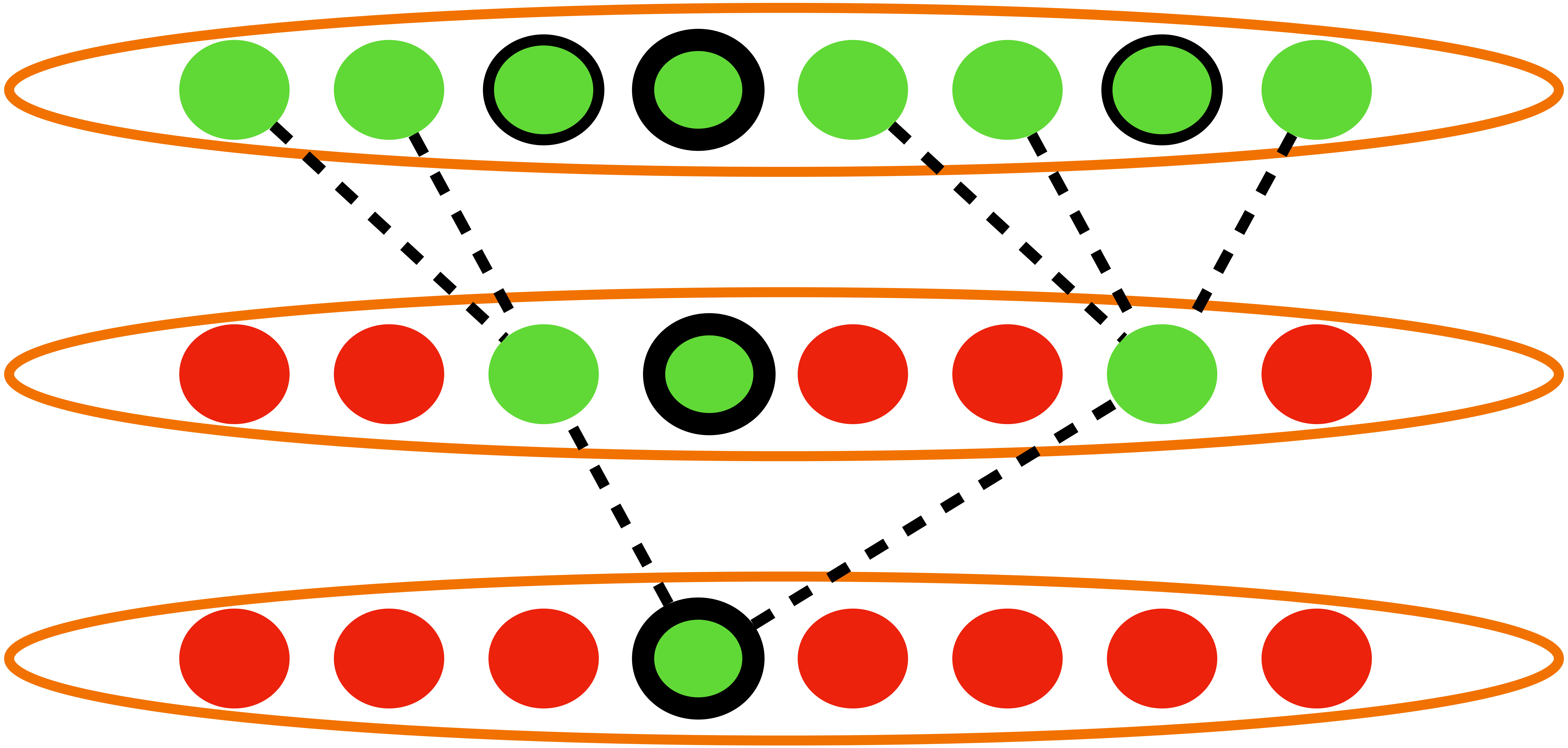
Sampled independently w.p. $n^{-\epsilon}$, so
after enough rounds it will be
nonsampled + deactivated w.h.p.

Is it feasible = Is it a tree?

Then all vertices are connected by some path to the root

Is it feasible = Is it a tree?

Then all vertices are connected by some path to the root



Length Bound

Length Bound

We add paths of length at most L for $O(1/\epsilon)$ rounds

Length Bound

We add paths of length at most L for $O(1/\epsilon)$ rounds

$\implies$ Subgraph has length $O(1/\epsilon) \cdot L$

Length Bound

We add paths of length at most L for $O(1/\epsilon)$ rounds

$\implies$ Subgraph has length $O(1/\epsilon) \cdot L$

🤯

Weight Bound

Weight Bound

Idea: compare how a **worse** algorithm does on a **structured** graph.

Weight Bound

Idea: compare how a **worse** algorithm does on a **structured** graph.

our alg weight $\leq$ worse alg weight $\leq O(n^\epsilon/\epsilon) \cdot \mathrm{OPT}_L$

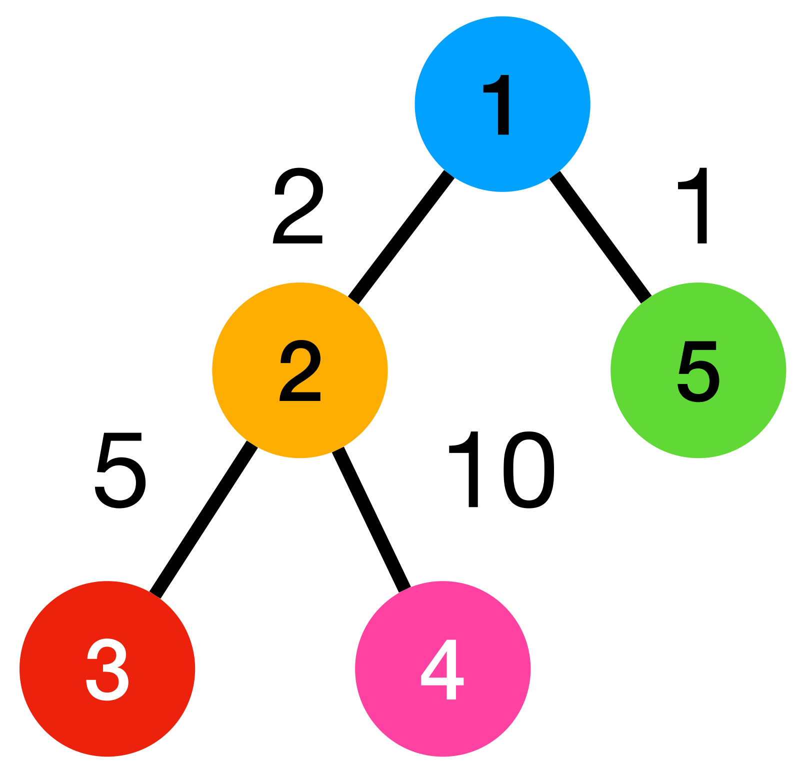
Weight Bound

Structured graph: a contracted Euler tour of an optimal solution

Weight Bound

Structured graph: a contracted Euler tour of an optimal solution

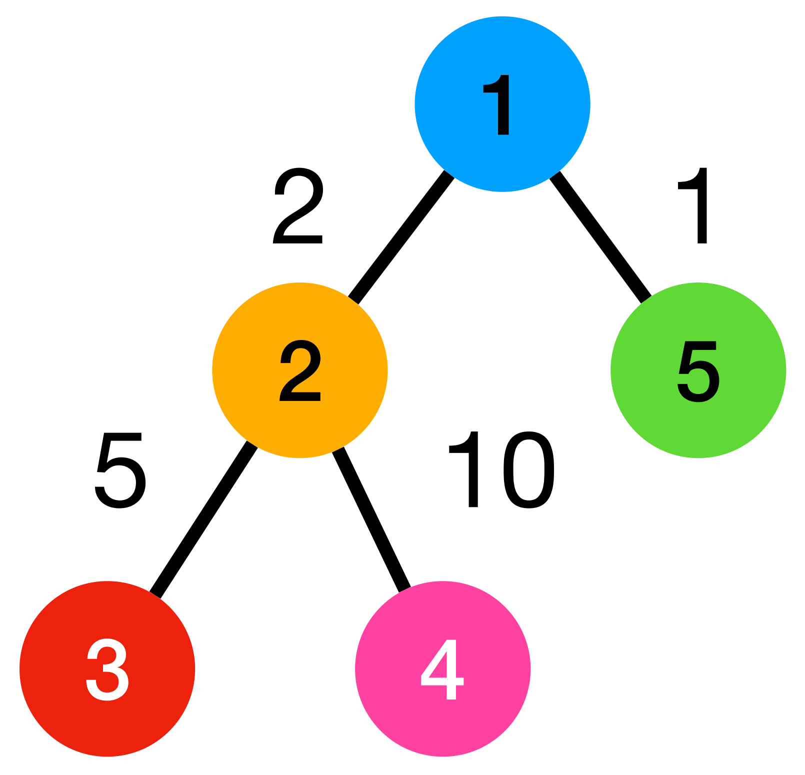
Optimal tree



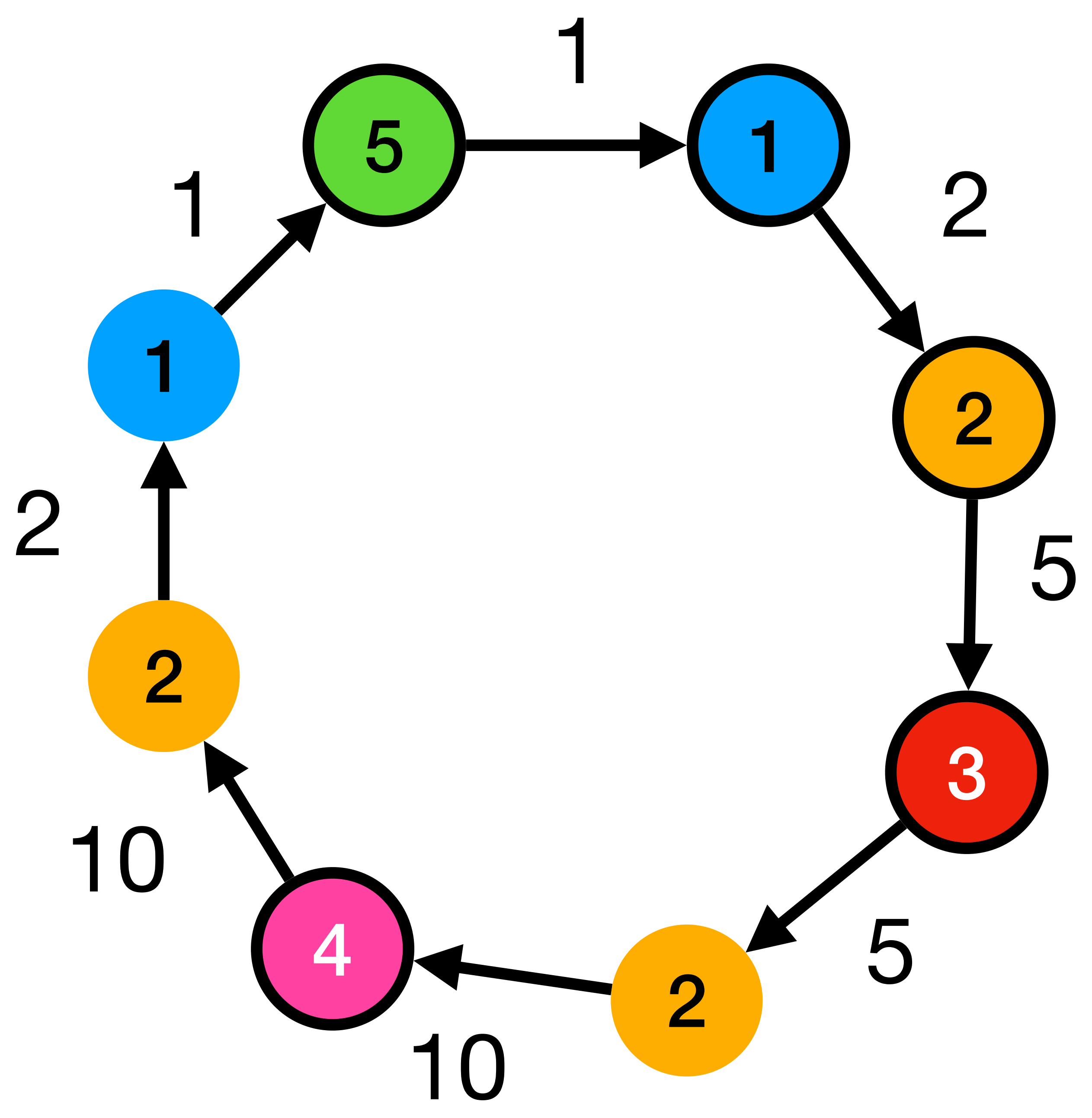
Weight Bound

Structured graph: a contracted Euler tour of an optimal solution

Optimal tree



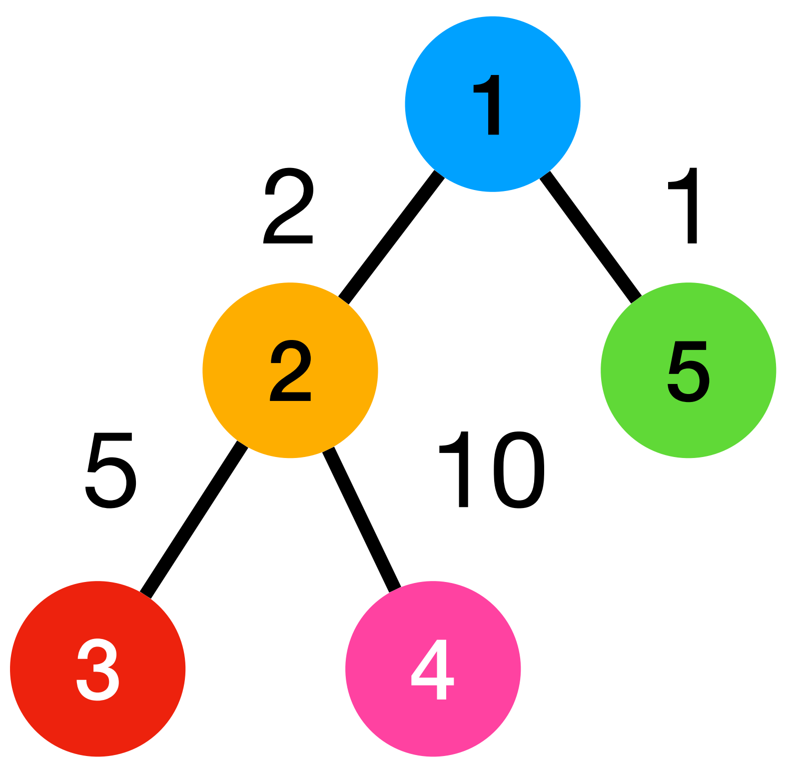
Euler tour



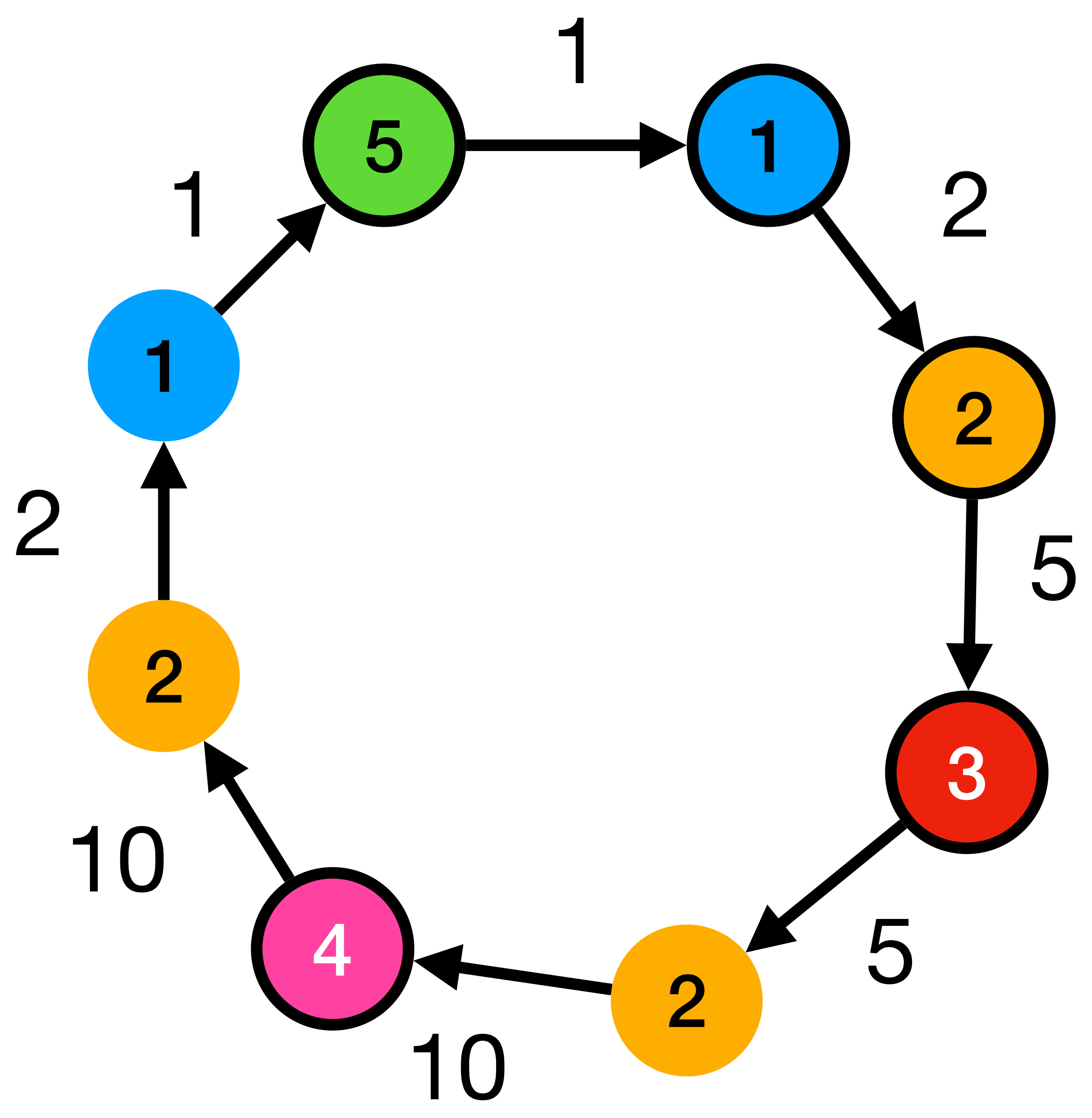
Weight Bound

Structured graph: a contracted Euler tour of an optimal solution

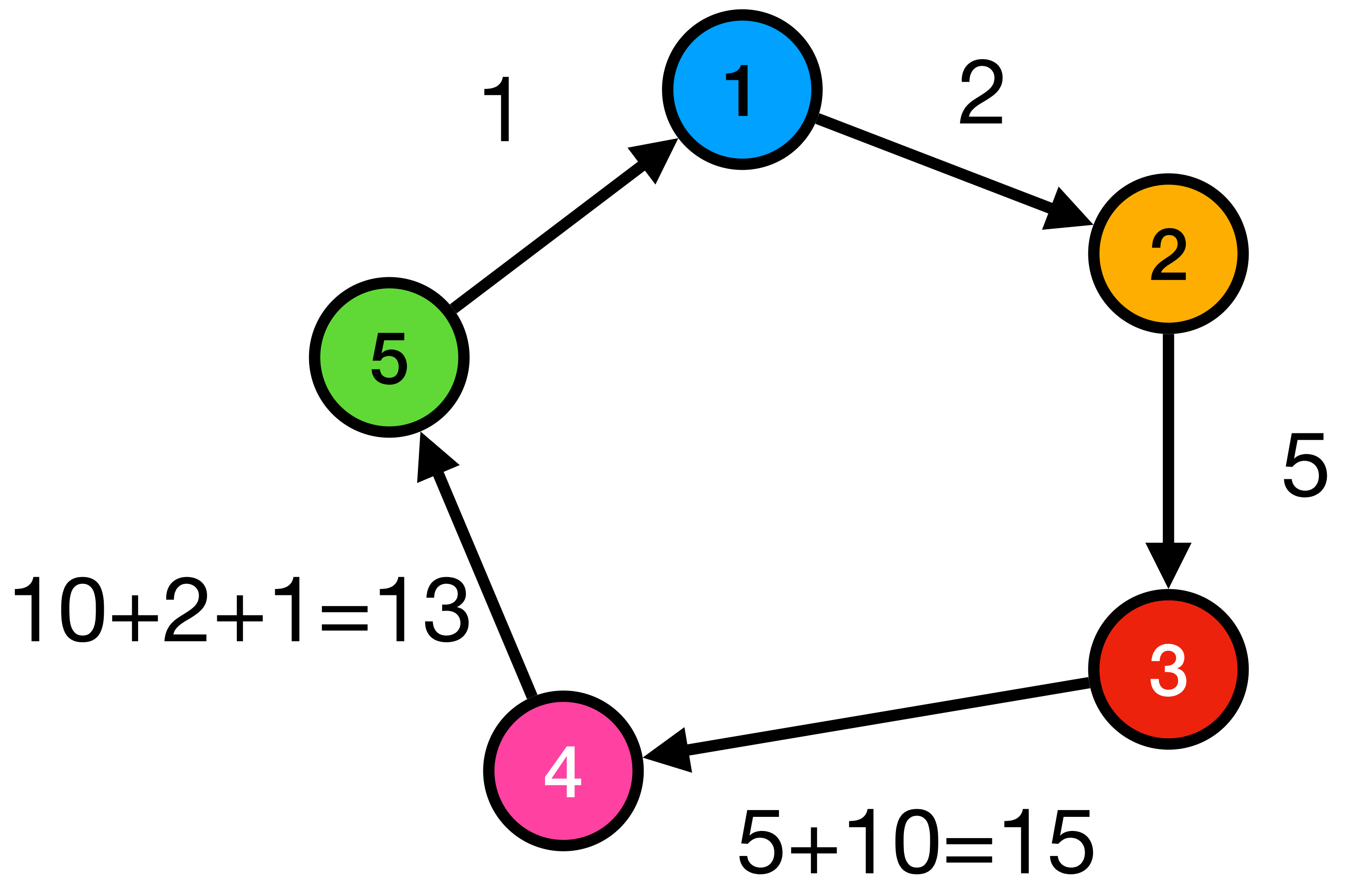
Optimal tree



Euler tour



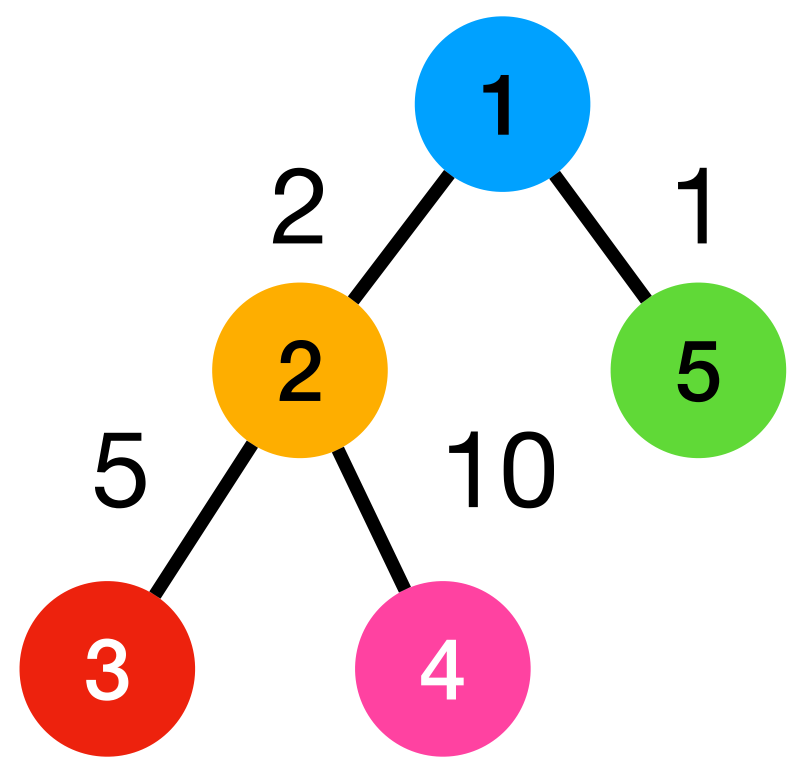
Contracted



Weight Bound

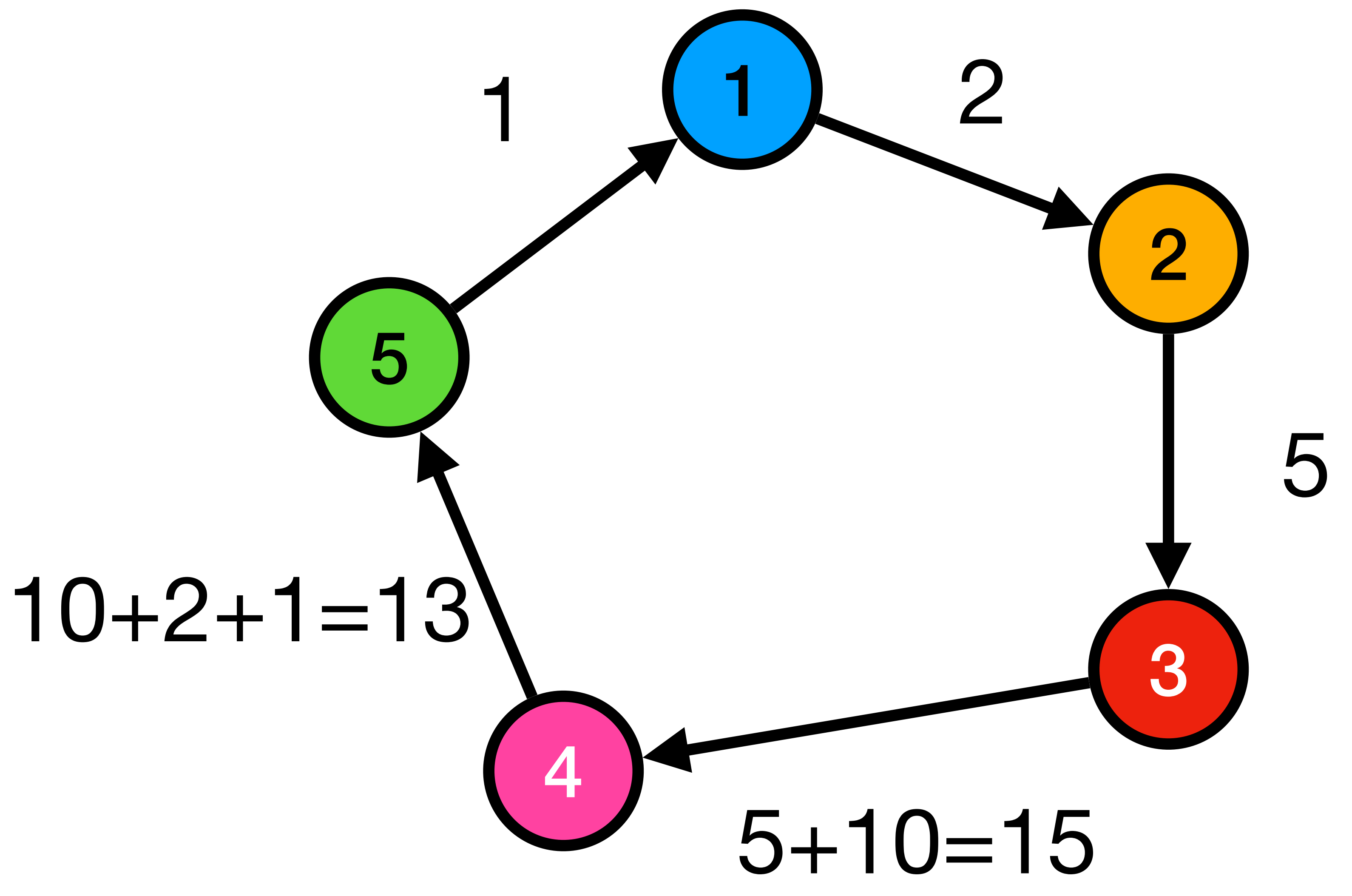
Structured graph: a contracted Euler tour of an optimal solution

Optimal tree



Total sum of edge weights in contracted Euler tour is $O(\mathbf{OPT}_L)$!!

Contracted



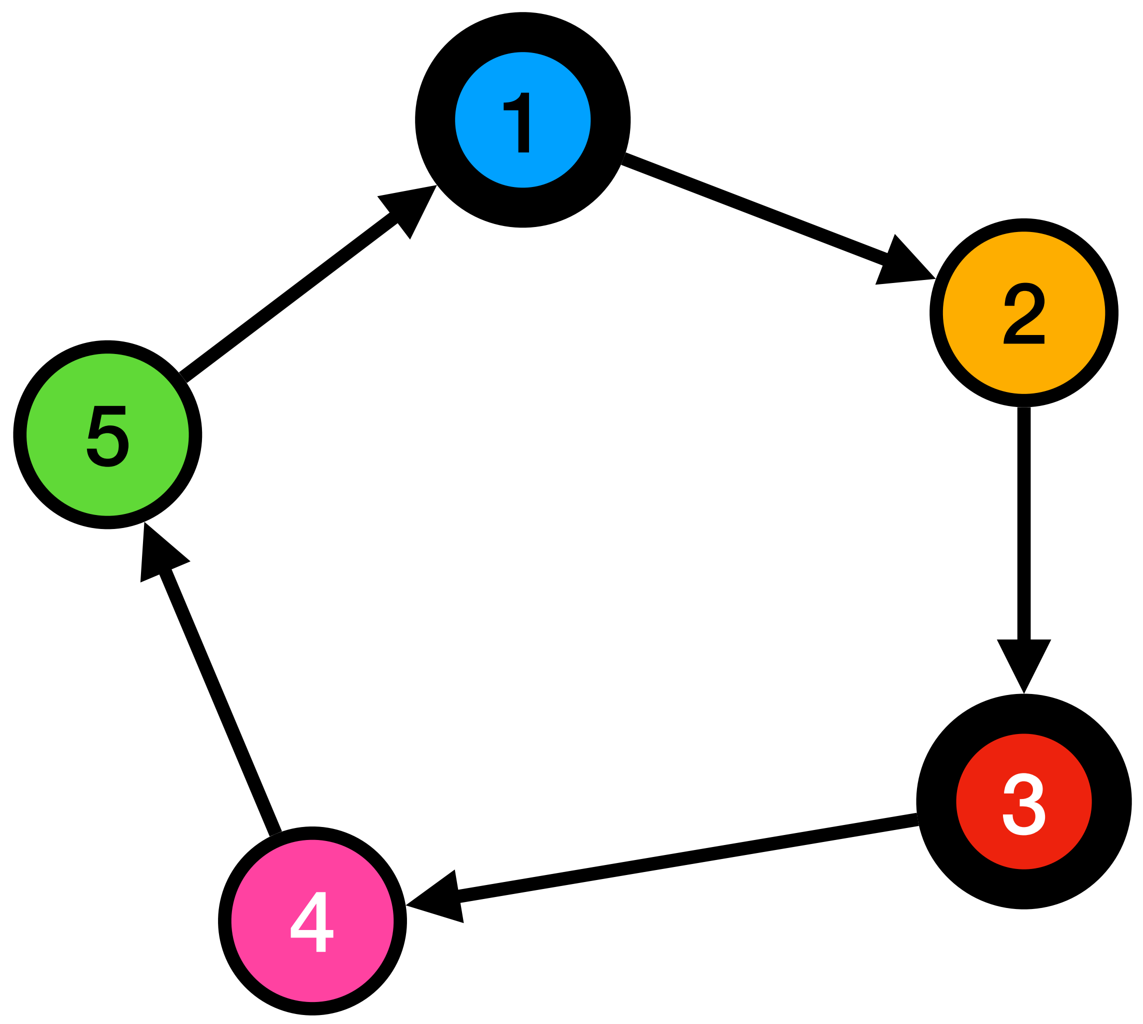
Weight Bound

Worse algorithm: charge the weight of the path from each non-sampled vertex to its nearest sampled vertex in the contracted Euler tour

Weight Bound

Worse algorithm: charge the weight of the path from each non-sampled vertex to its nearest sampled vertex in the contracted Euler tour

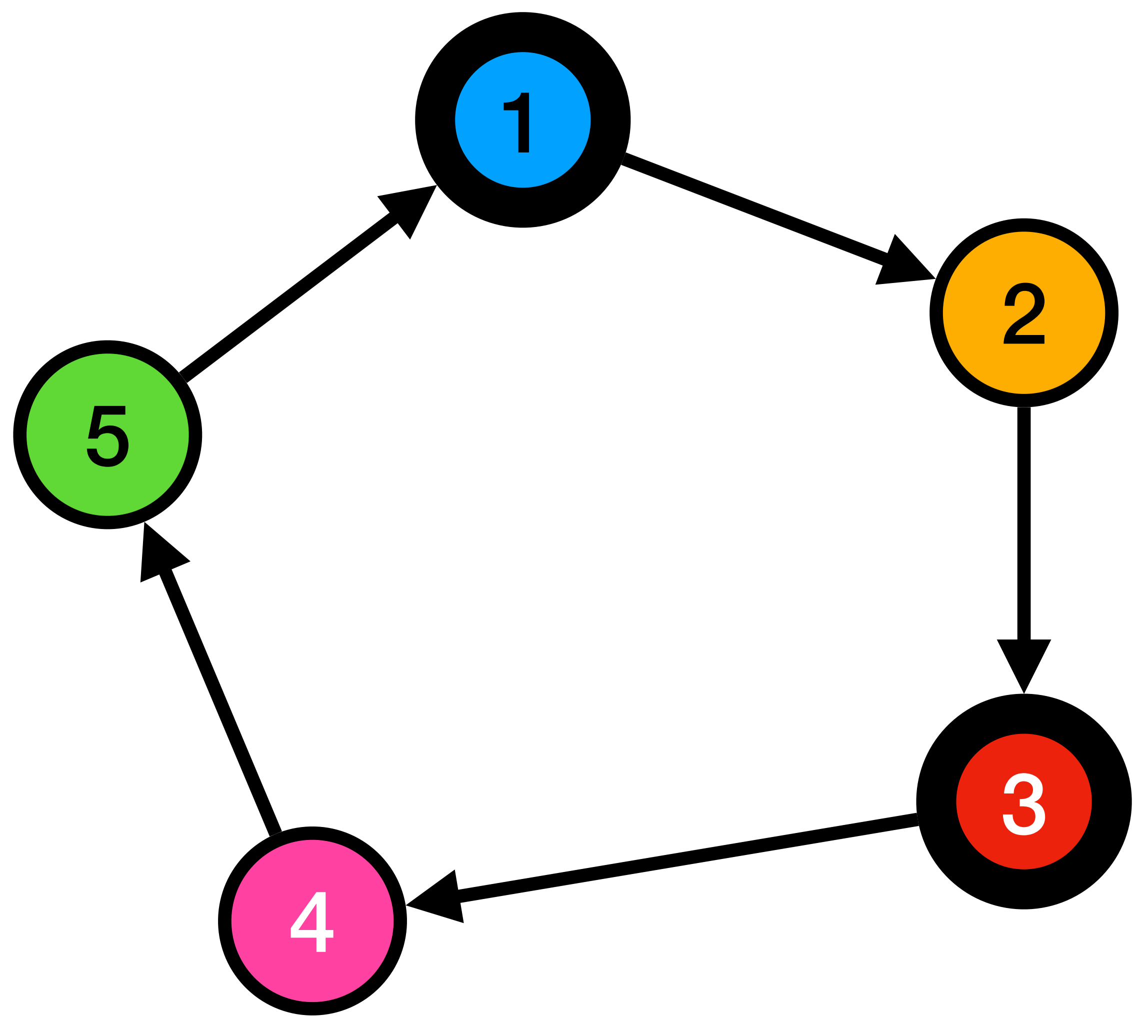
1, 3 are sampled



Weight Bound

Worse algorithm: charge the weight of the path from each non-sampled vertex to its nearest sampled vertex in the contracted Euler tour

1, 3 are sampled

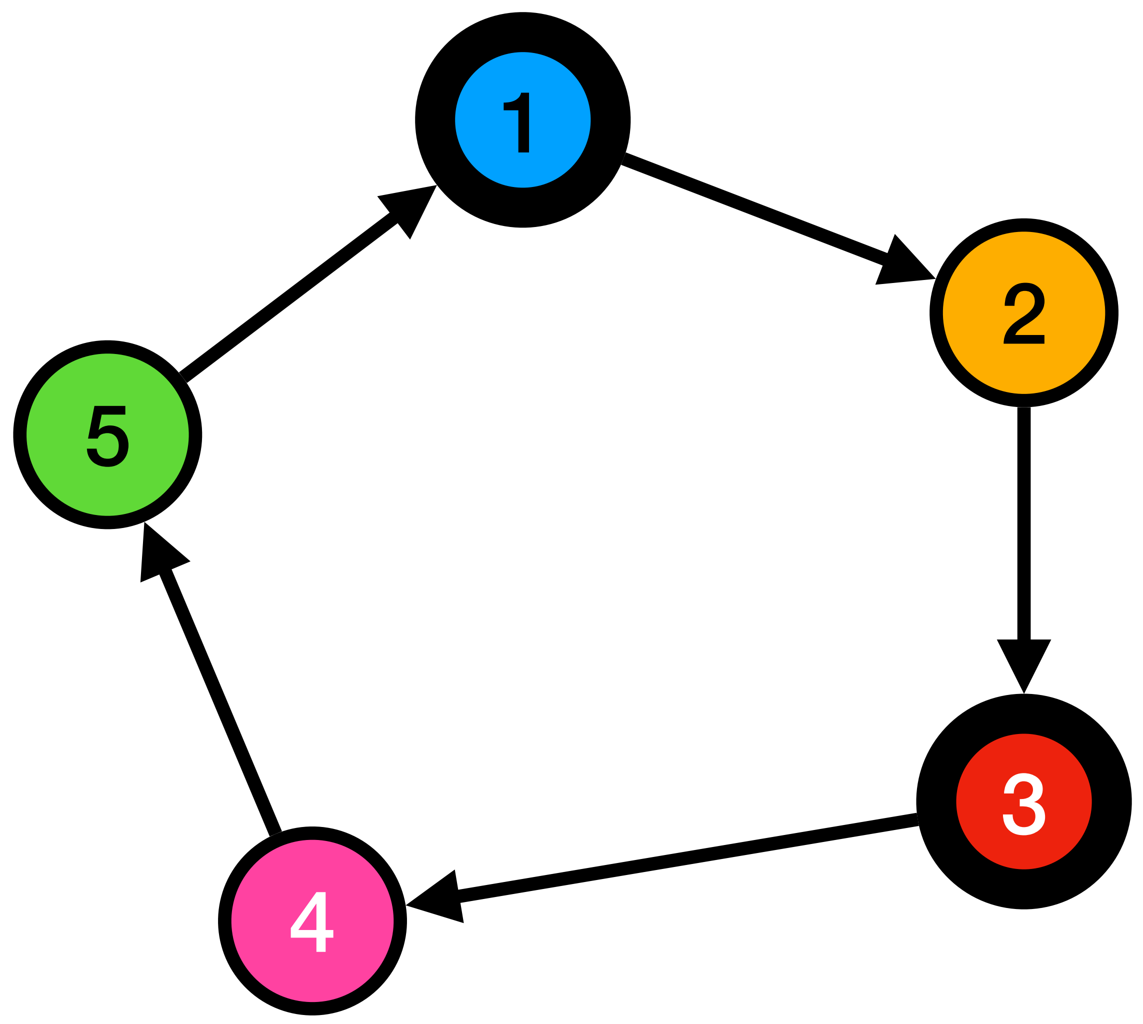


Charge the paths:
(2,3)
(4,1)
(5,1)

Weight Bound

Worse algorithm: charge the weight of the path from each non-sampled vertex to its nearest sampled vertex in the contracted Euler tour

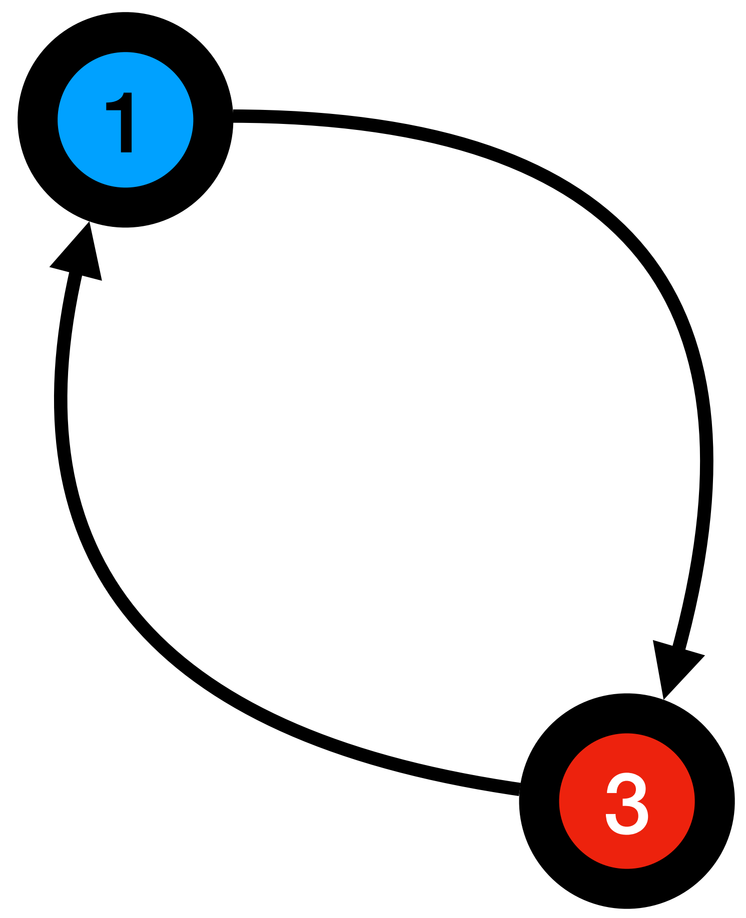
Deactivate non-sampled vertices and repeat



Weight Bound

Worse algorithm: charge the weight of the path from each non-sampled vertex to its nearest sampled vertex in the contracted Euler tour

Deactivate non-sampled vertices and repeat



Weight Bound

Why worse?

Weight Bound

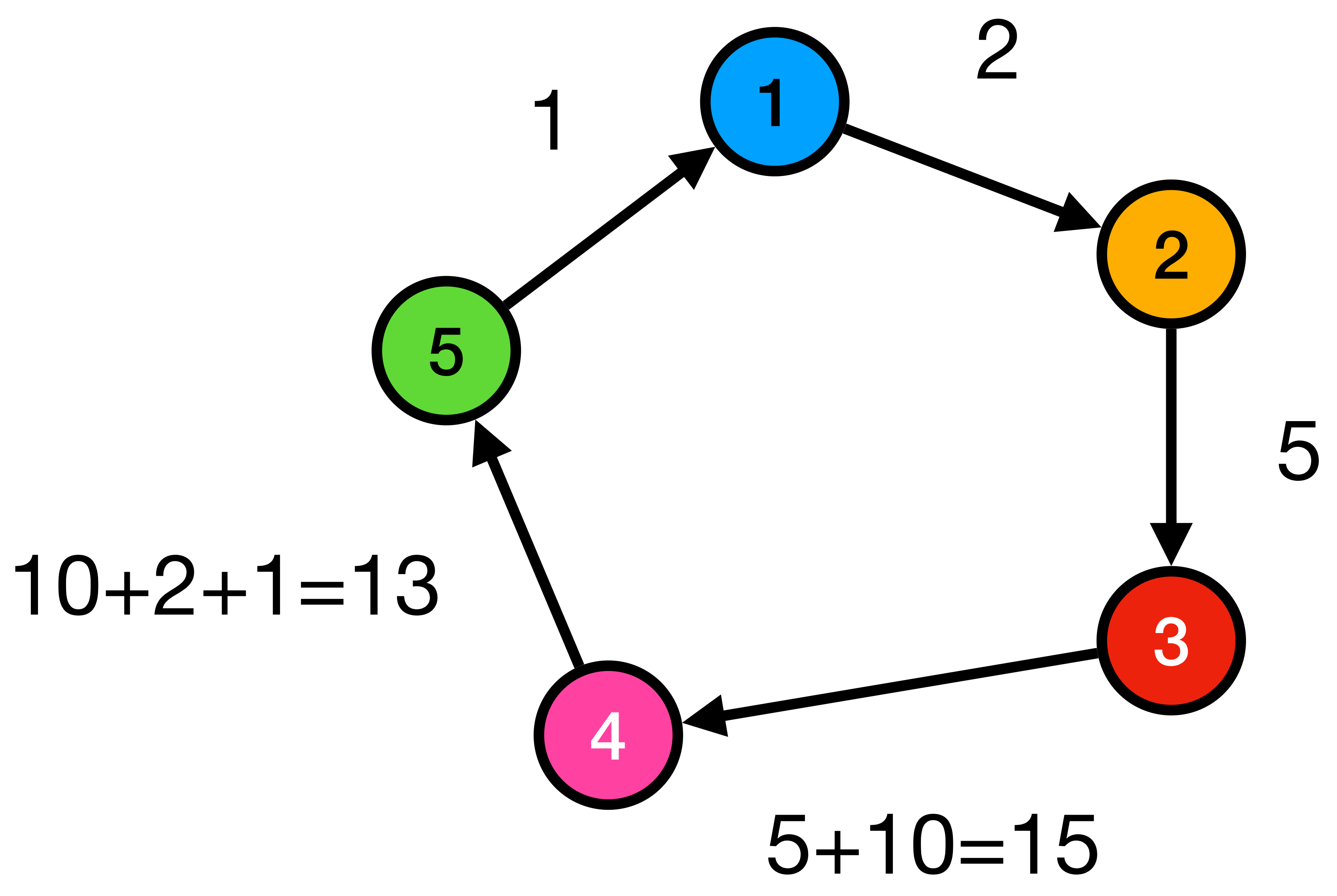
Why worse?

Worse alg charges L-bounded paths from the ET (could be arbitrarily high in weight)

Weight Bound

Why worse?

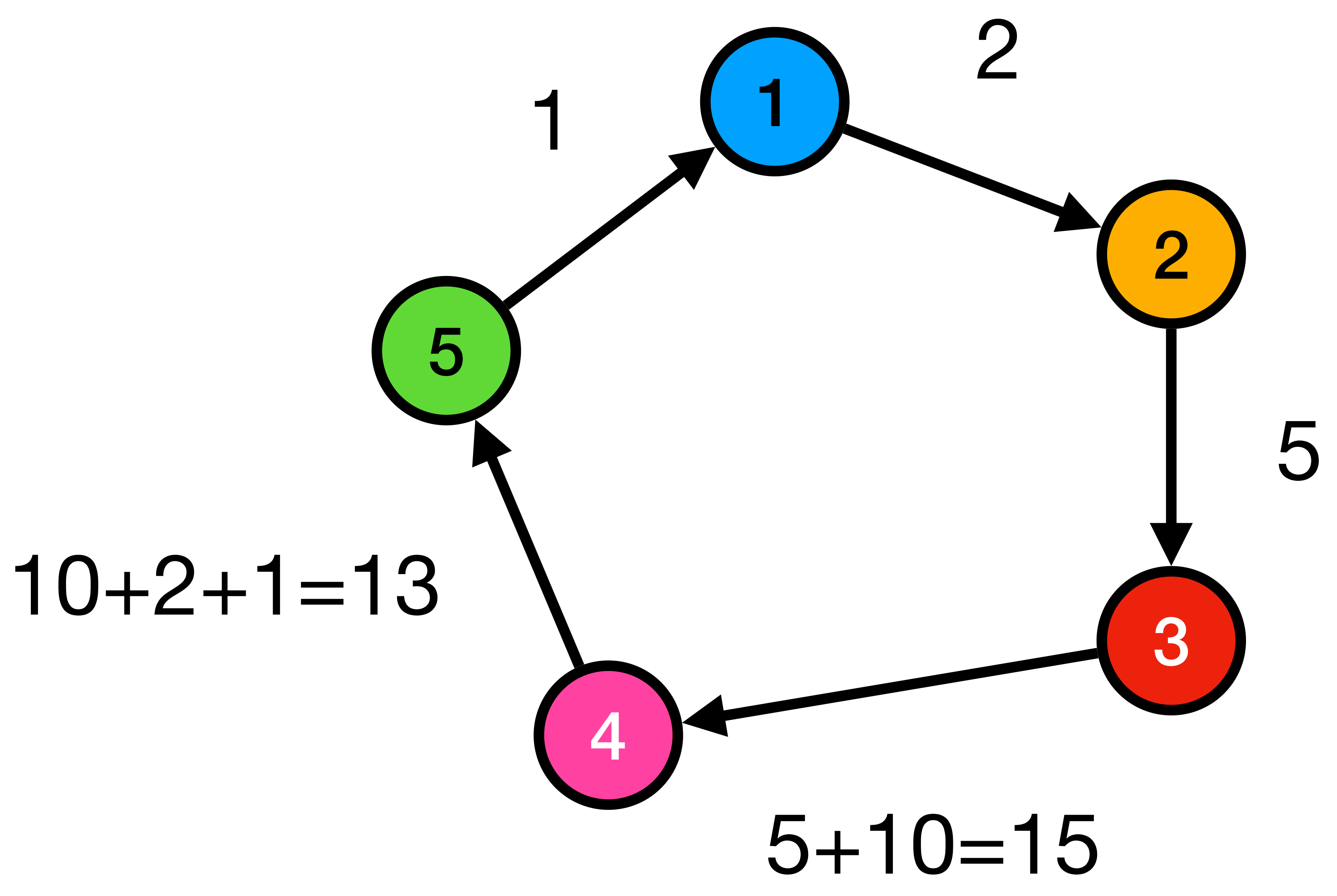
Worse alg charges L-bounded paths from the ET (could be arbitrarily high in weight)



Weight Bound

Why worse?

Worse alg charges L-bounded paths from the ET (could be arbitrarily high in weight)

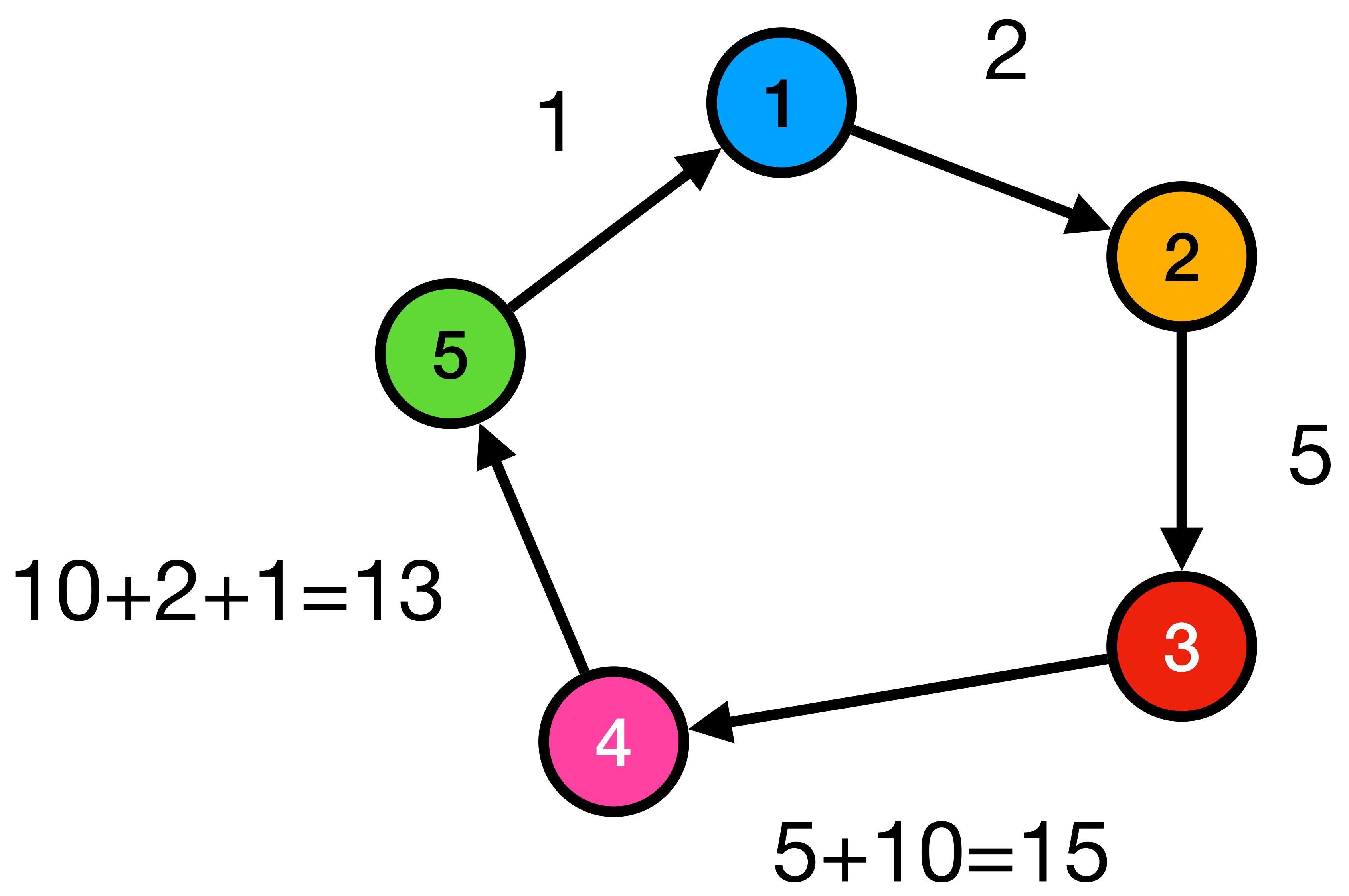


Weight Bound

Why worse?

Worse alg charges L-bounded paths from the ET (could be arbitrarily high in weight)

But our alg chooses the **min-weight** L-bounded paths

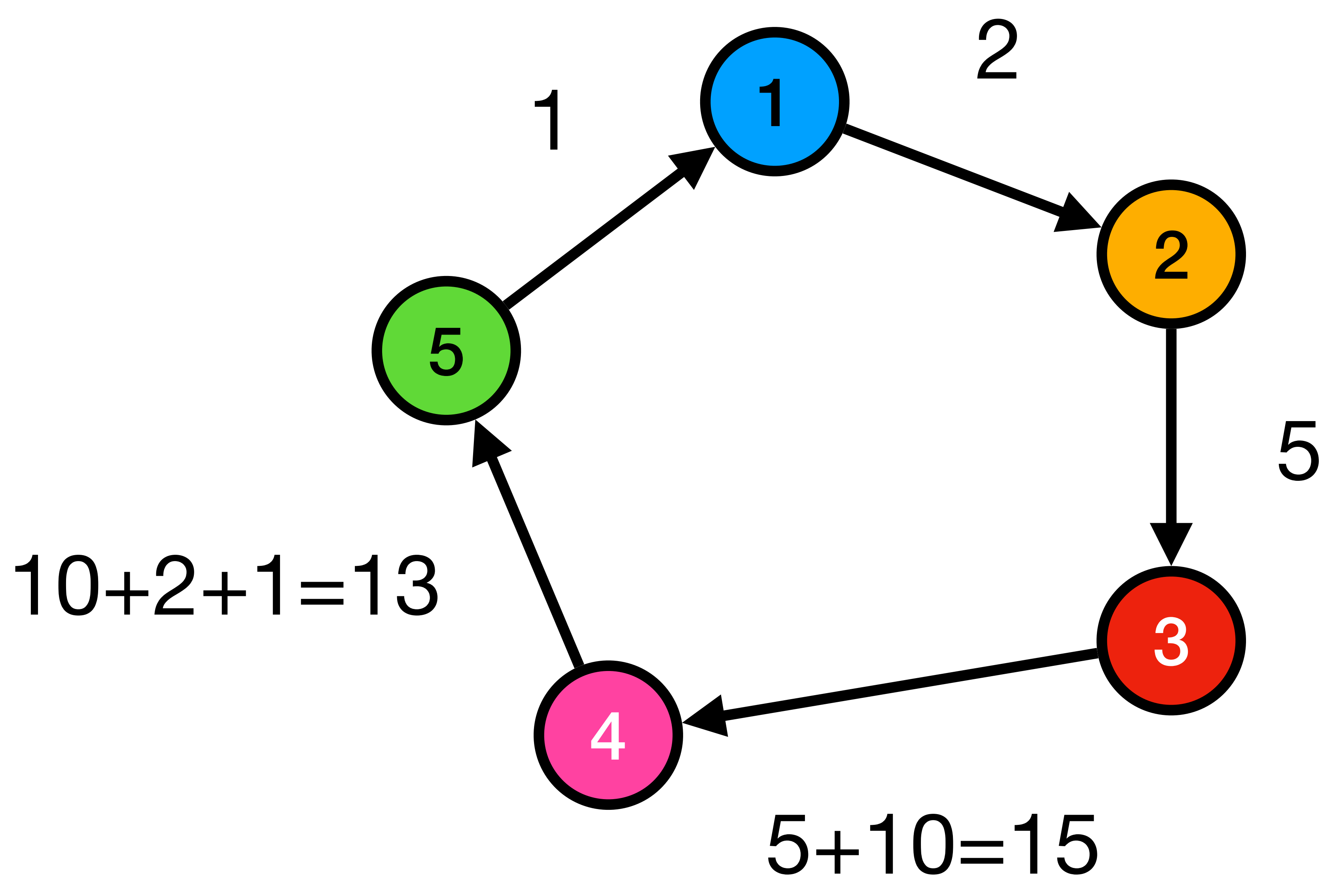


Weight Bound

Why worse?

Worse alg charges L-bounded paths from the ET (could be arbitrarily high in weight)

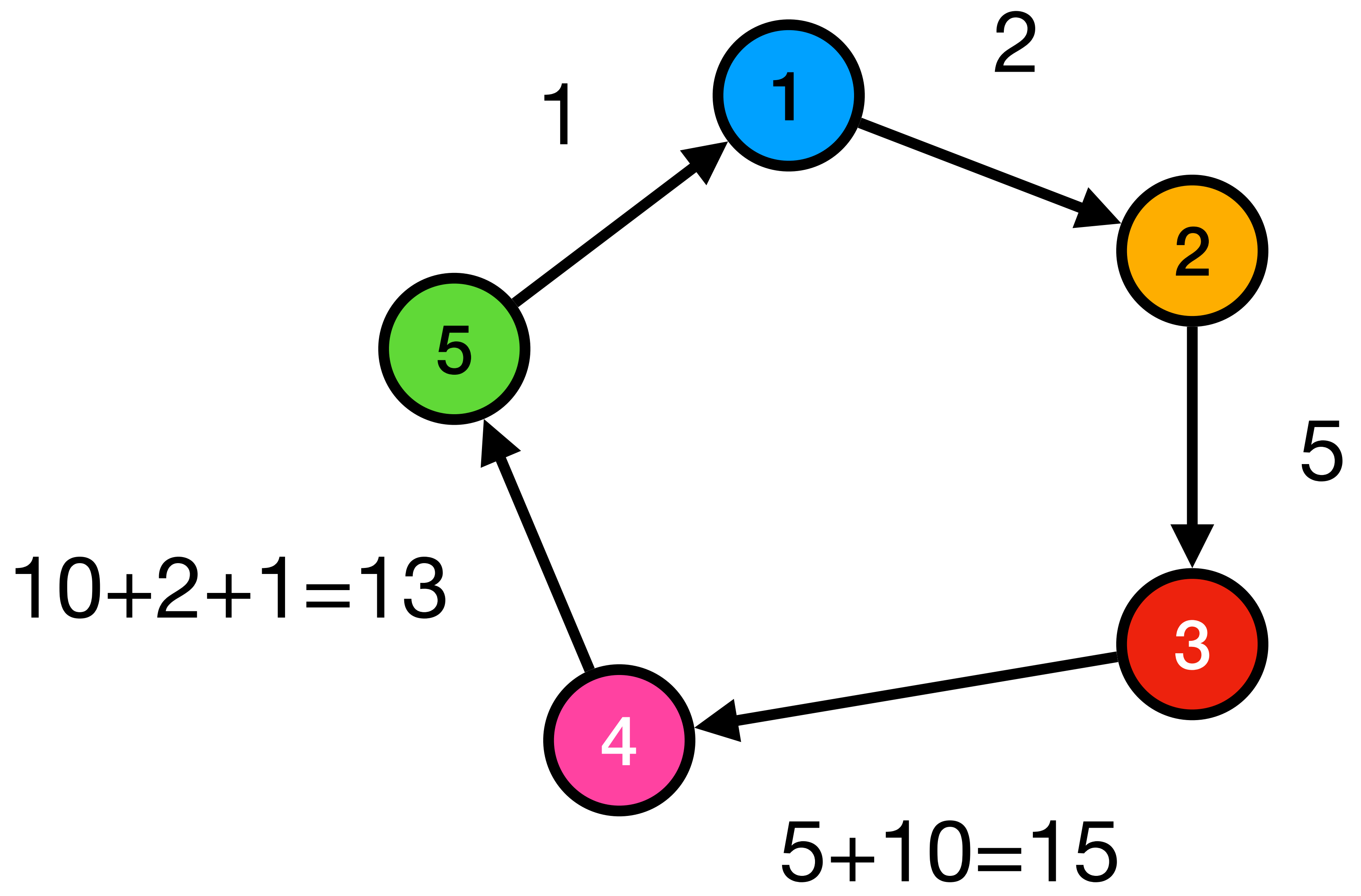
But our alg chooses the **min-weight** L-bounded paths



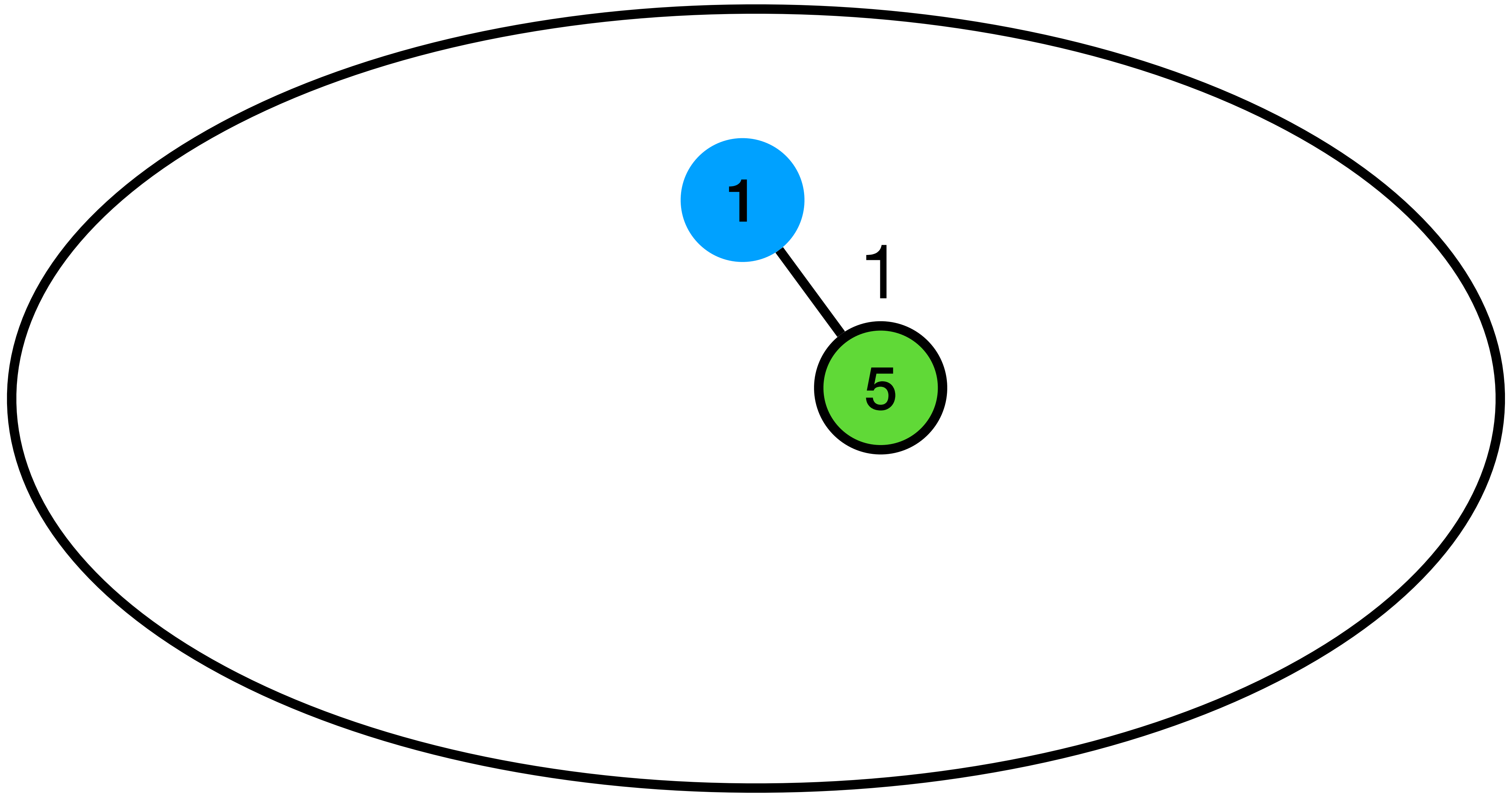
Weight Bound

Why worse?

Worse alg charges L-bounded paths from the ET (could be arbitrarily high in weight)



But our alg chooses the **min-weight** L-bounded paths



Weight Bound

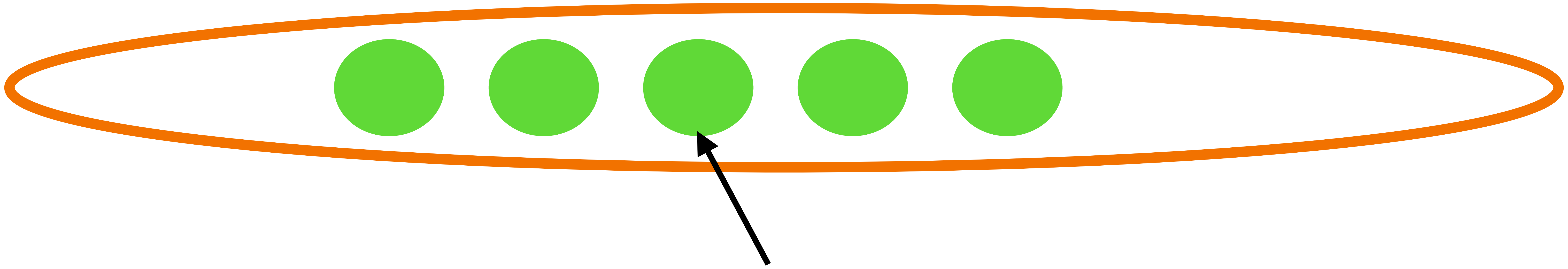
So our algorithm's weight is at most the worse algorithm's weight

Weight Bound

So our algorithm's weight is at most the worse algorithm's weight

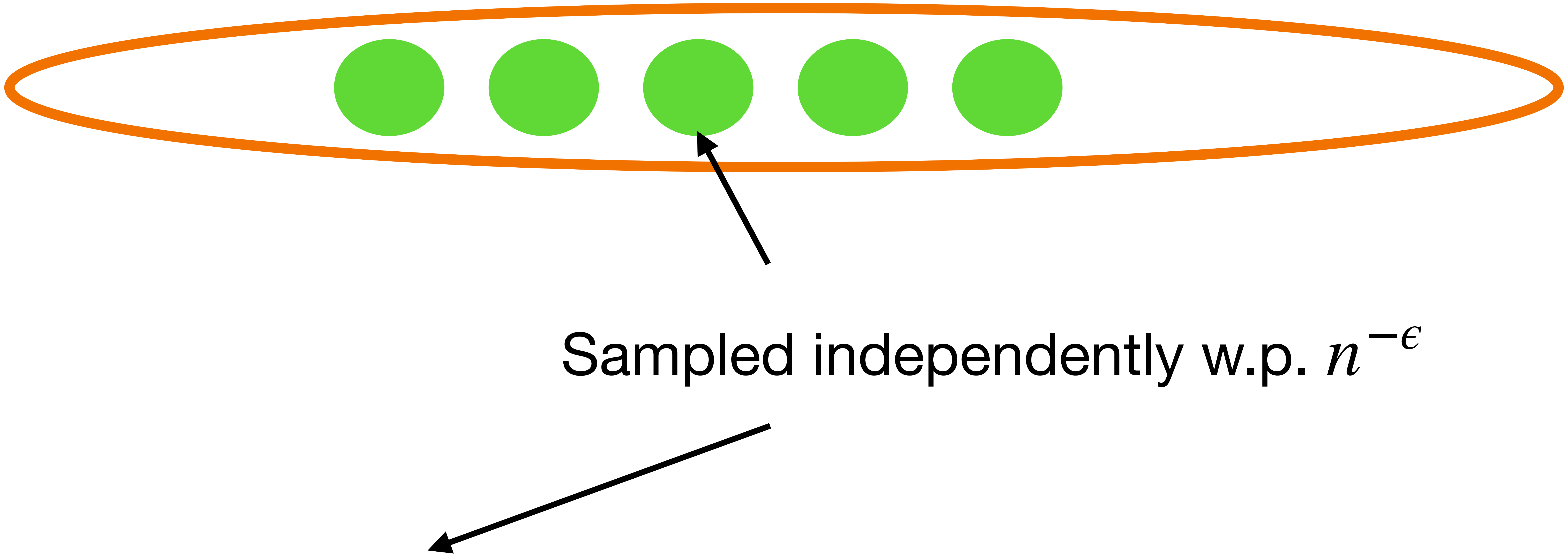
Now just show that the worse algorithm's weight is at most
$O(n^\epsilon/\epsilon) \cdot \text{OPT}_L$

Weight Bound

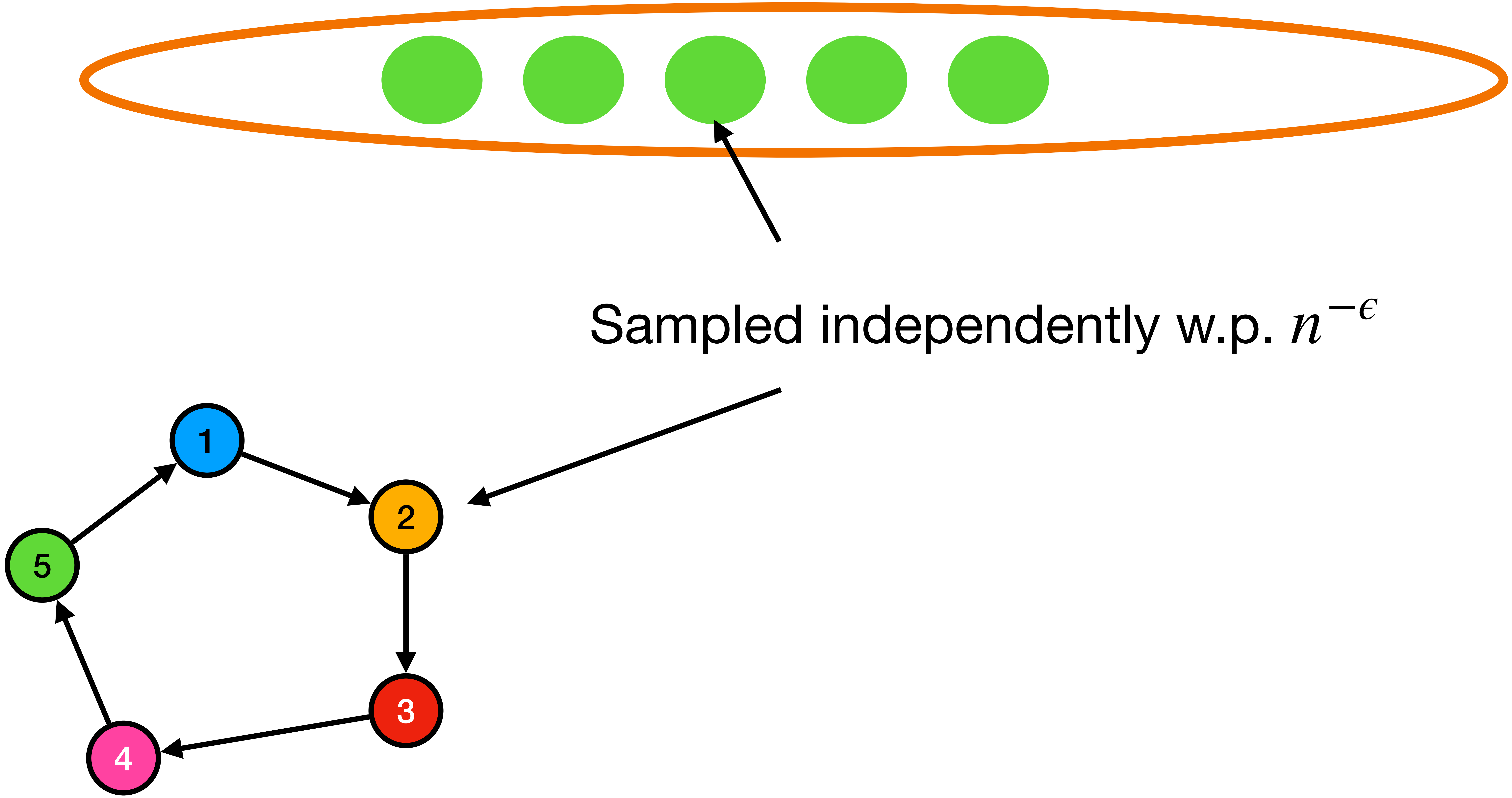


Sampled independently w.p. $n^{-\epsilon}$

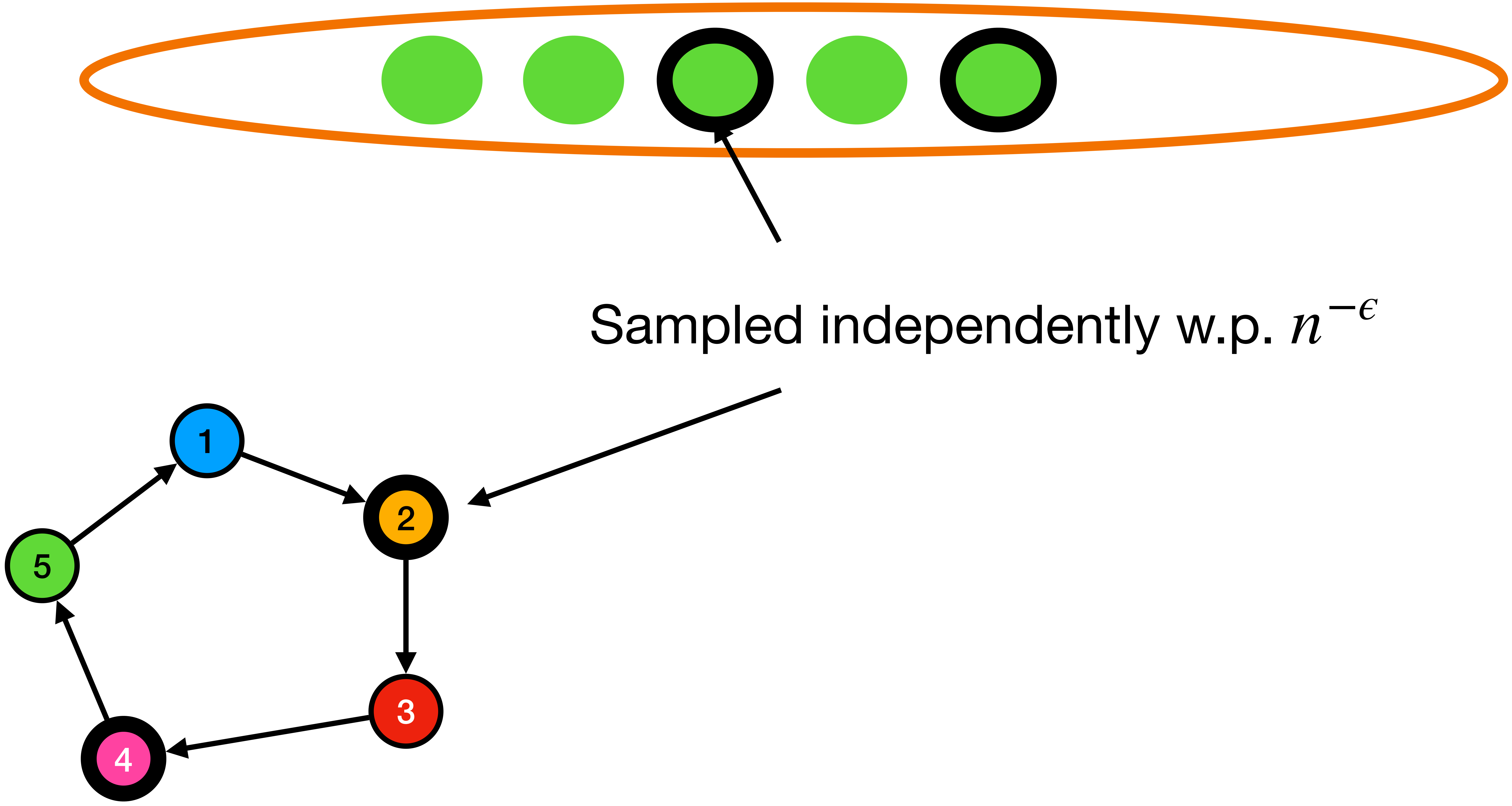
Weight Bound



Weight Bound



Weight Bound

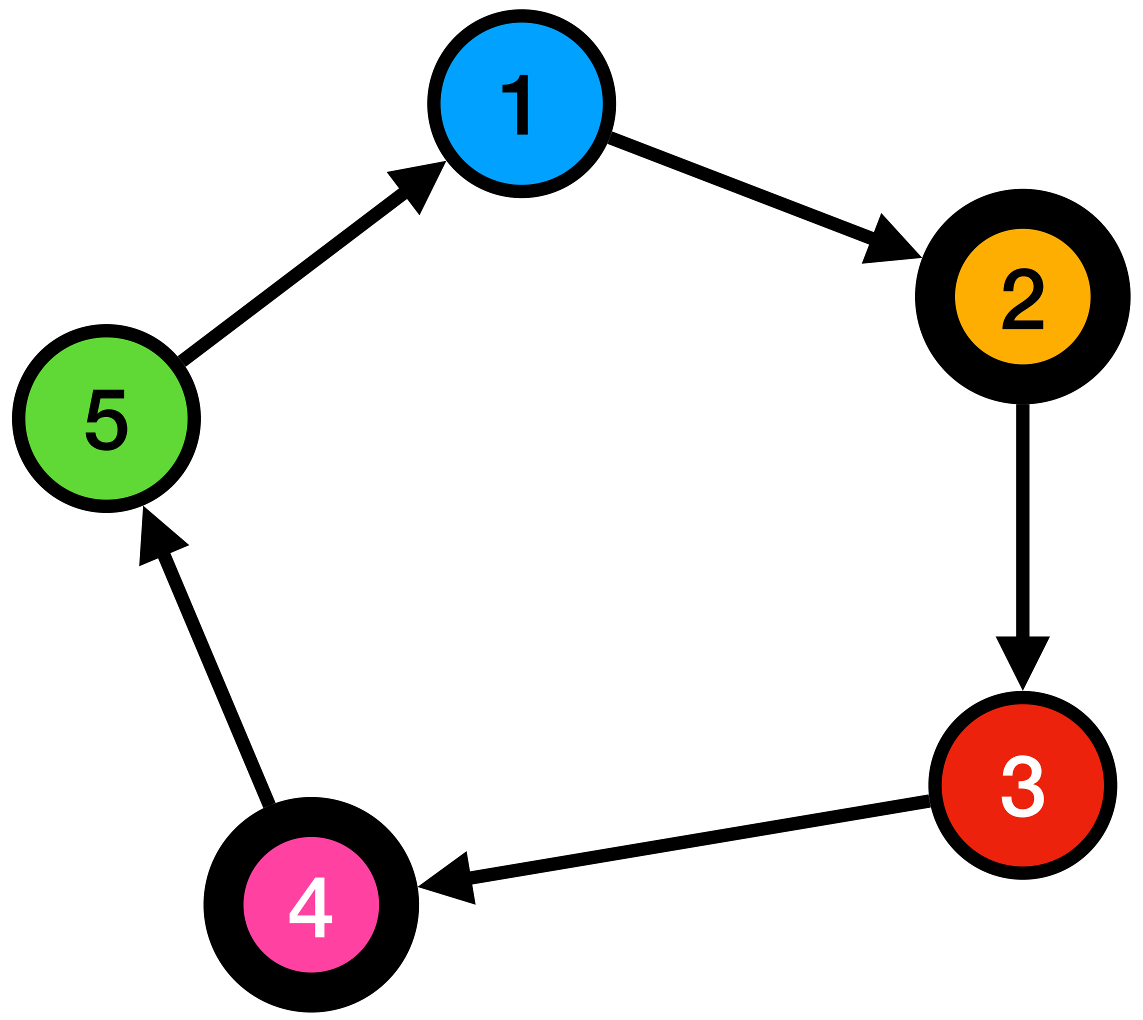


Weight Bound

In the tour, an edge is charged $O(n^{\epsilon})$ times (in expectation)

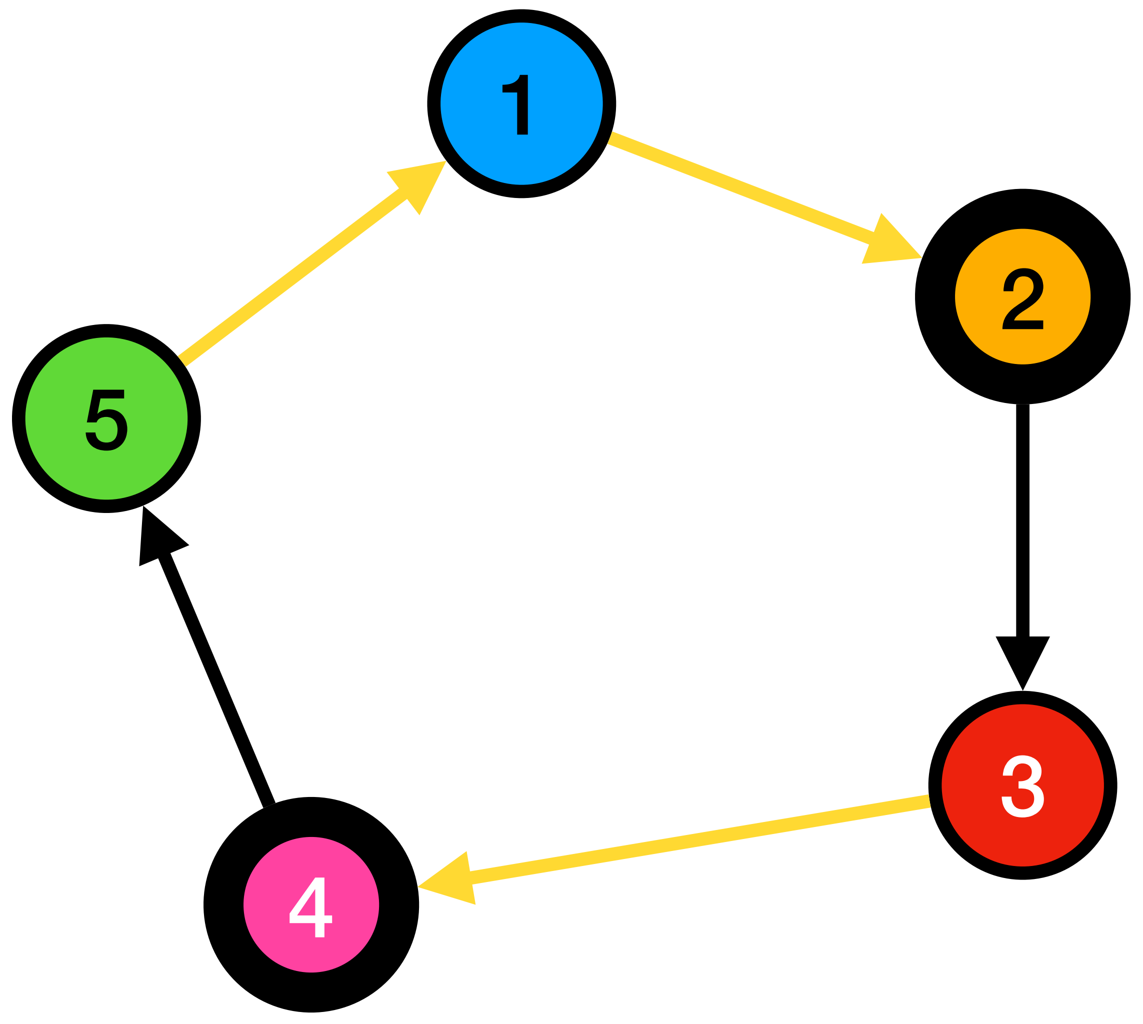
Weight Bound

In the tour, an edge is charged $O(n^{\epsilon})$ times (in expectation)



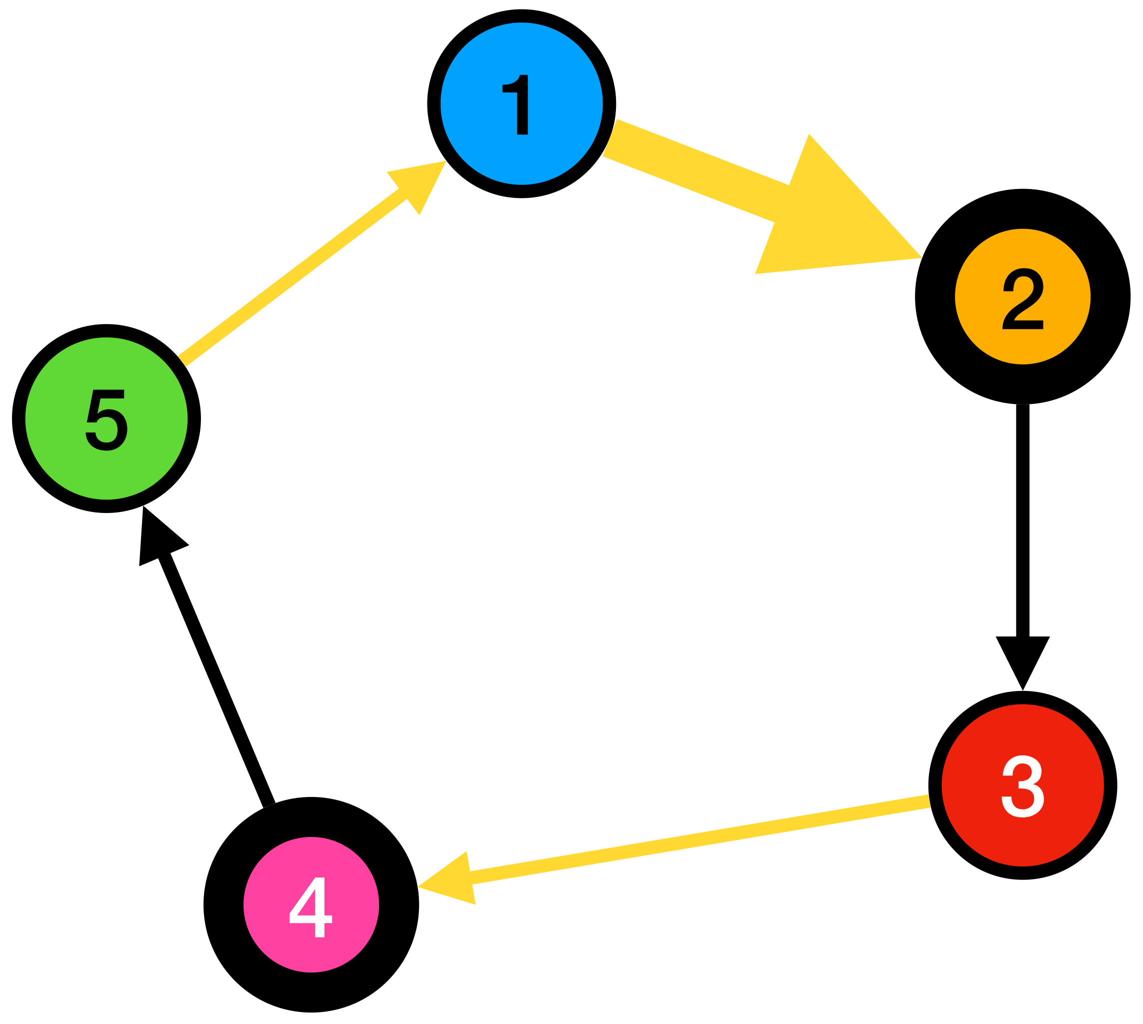
Weight Bound

In the tour, an edge is charged $O(n^{\epsilon})$ times (in expectation)



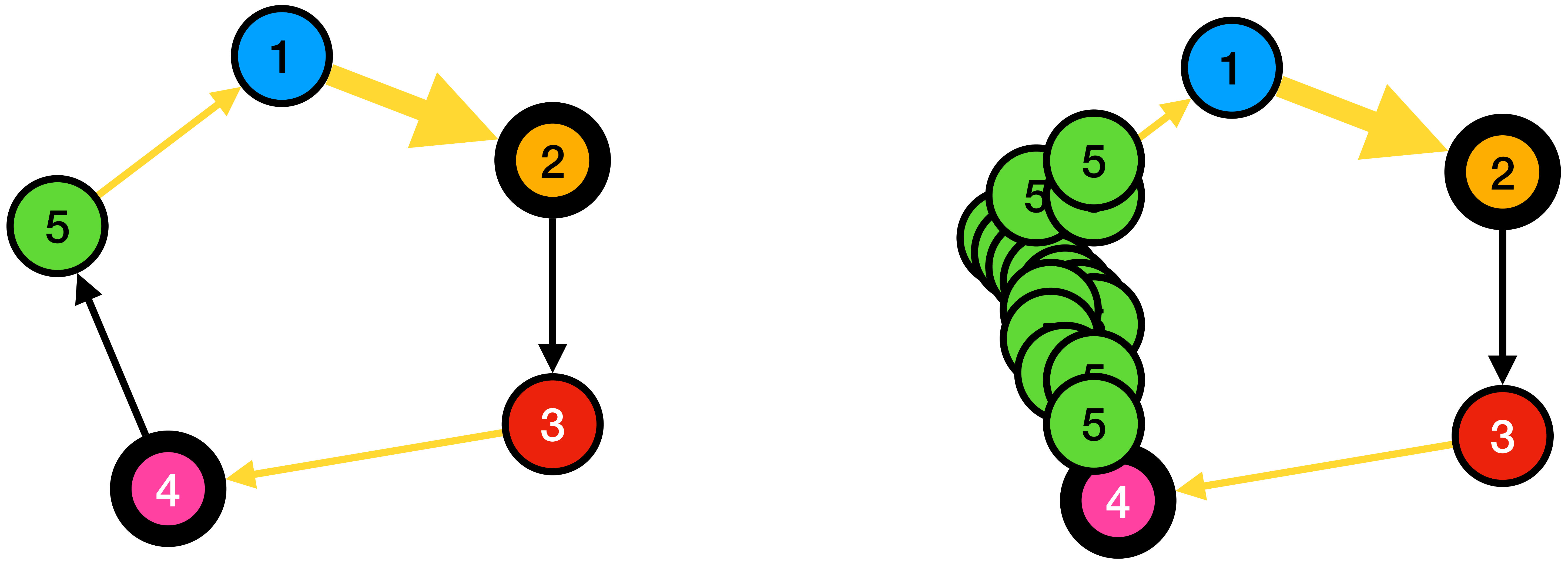
Weight Bound

In the tour, an edge is charged $O(n^{\epsilon})$ times (in expectation)



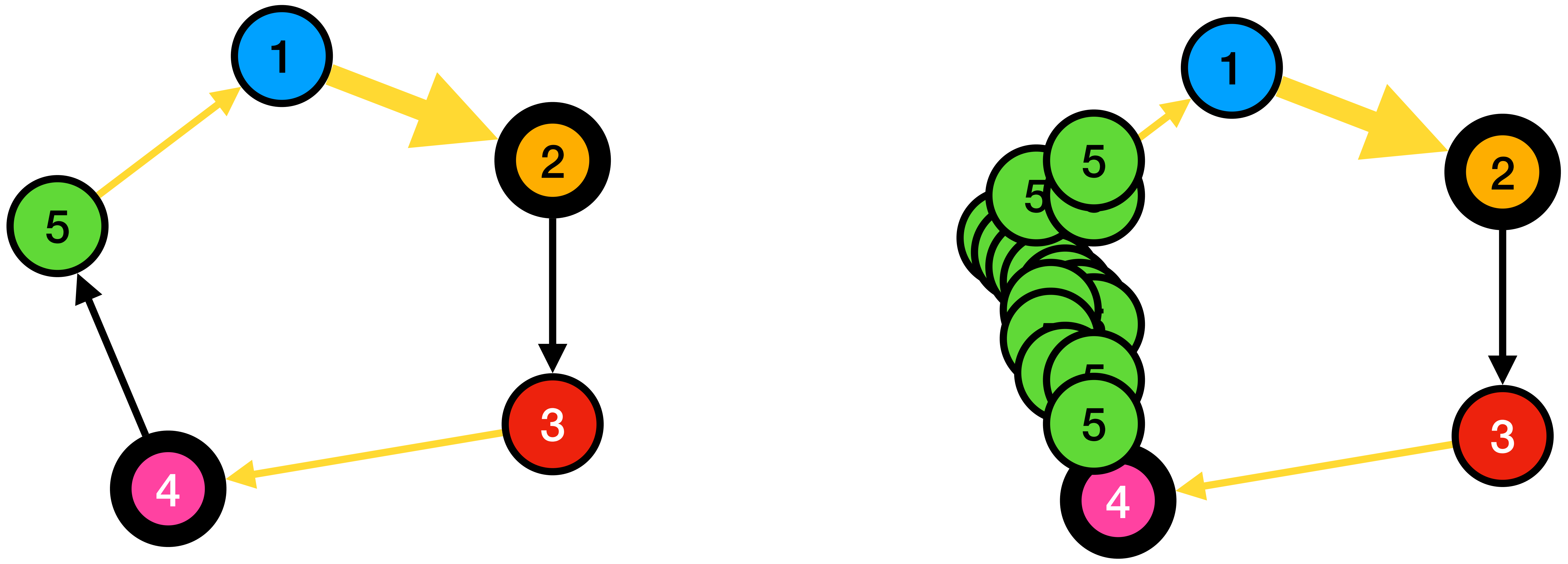
Weight Bound

In the tour, an edge is charged $O(n^{\epsilon})$ times (in expectation)



Weight Bound

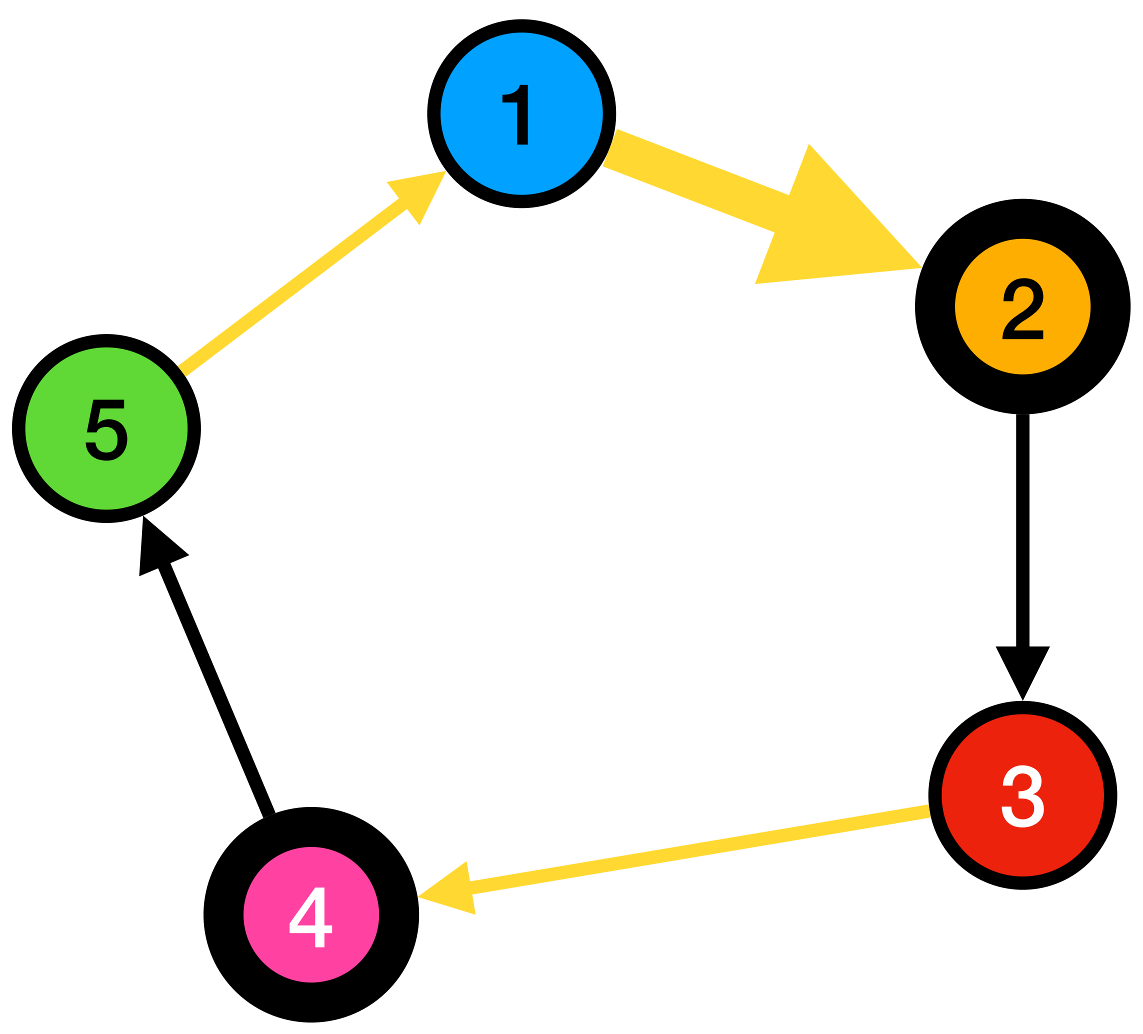
In the tour, an edge is charged $O(n^{\epsilon})$ times (in expectation)

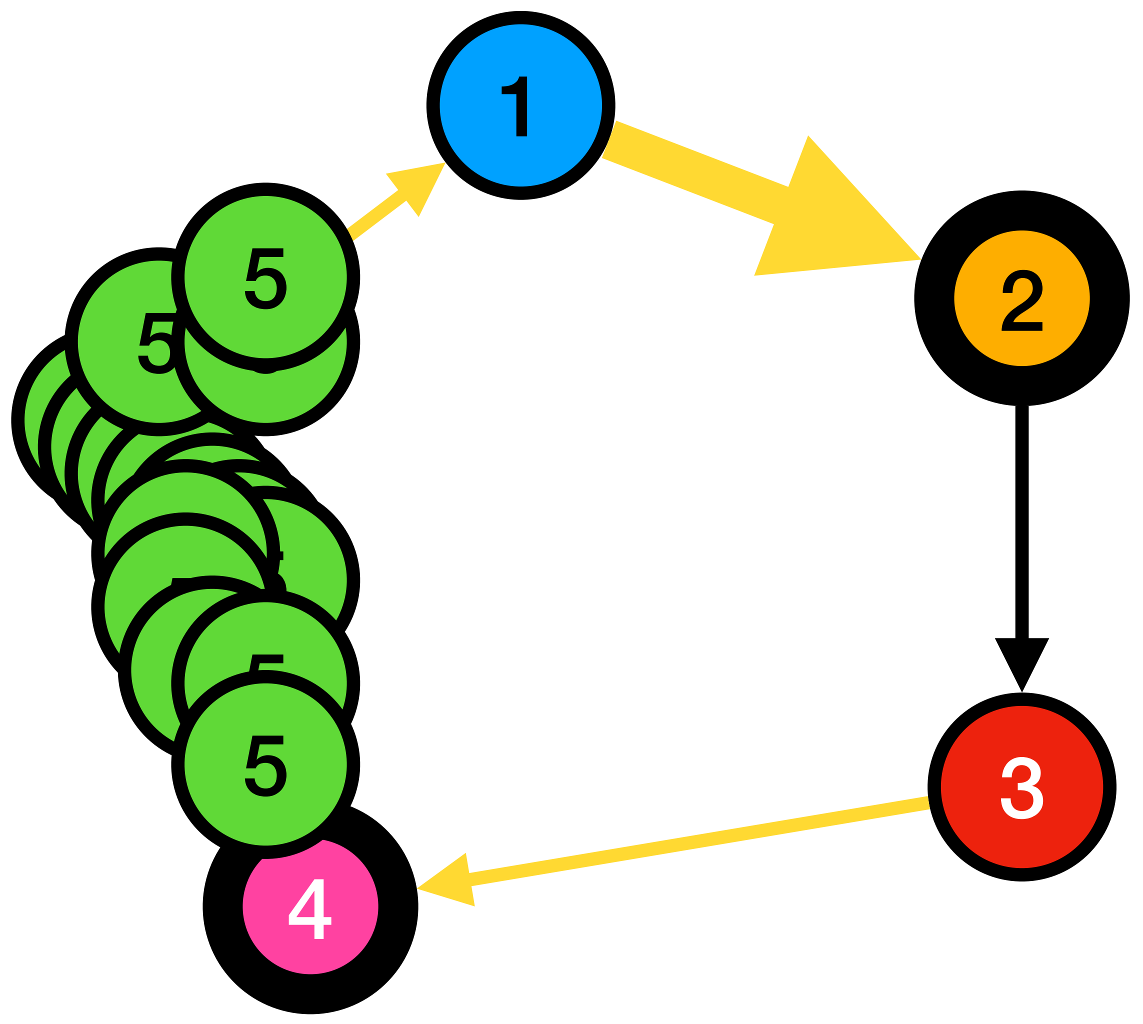


Weight Bound

In the tour, an edge is charged $O(n^\epsilon)$ times (in expectation)

+ $O(1/\epsilon)$ rounds + tour is over an optimal tree



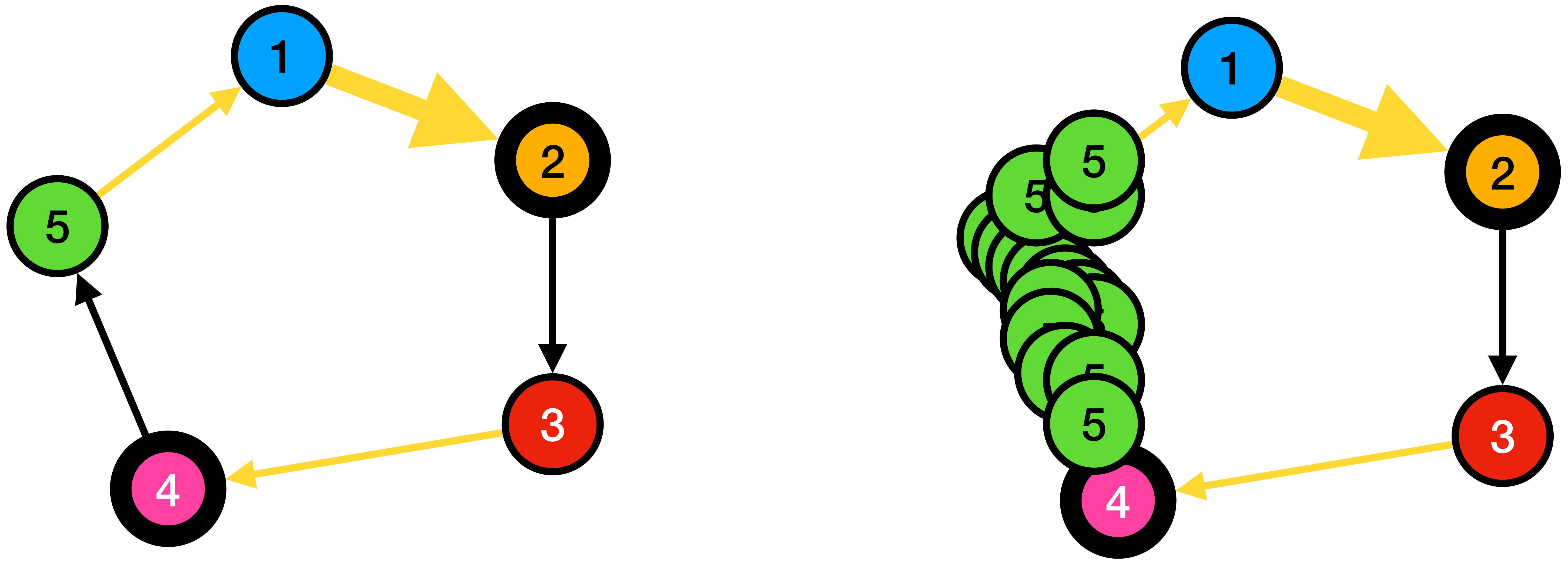


Weight Bound

In the tour, an edge is charged $O(n^{\epsilon})$ times (in expectation)

+ $O(1/\epsilon)$ rounds + tour is over an optimal tree

$\Longrightarrow O(n^{\epsilon}/\epsilon)$ **weight approximation**



Conclusion

Conclusion

A simple **random sampling** + **greedily adding cheapest** L**-bounded paths**

Conclusion

A simple **random sampling** + **greedily adding cheapest L-bounded paths**

algorithm gets a spanning tree of

Conclusion

A simple **random sampling** + **greedily adding cheapest L-bounded paths**

algorithm gets a spanning tree of

length: $O(1/\epsilon) \cdot L$

Conclusion

A simple **random sampling** + **greedily adding cheapest L-bounded paths**

algorithm gets a spanning tree of

length: $O(1/\epsilon) \cdot L$

weight: $O(n^\epsilon/\epsilon) \cdot \text{OPT}_L$

Conclusion

A simple **random sampling** + **greedily adding cheapest L-bounded paths**

algorithm gets a spanning tree of

length: $O(1/\epsilon) \cdot L$

weight: $O(n^{\epsilon}/\epsilon) \cdot \mathrm{OPT}_L$

Can further tradeoff between length and weight using ϵ!

Conclusion

A simple **random sampling** + **greedily adding cheapest L-bounded paths**

algorithm gets a spanning tree of

length: $O(1/\epsilon) \cdot L$

weight: $O(n^\epsilon/\epsilon) \cdot \mathrm{OPT}_L$

Can further tradeoff between length and weight using ϵ!

Thank you