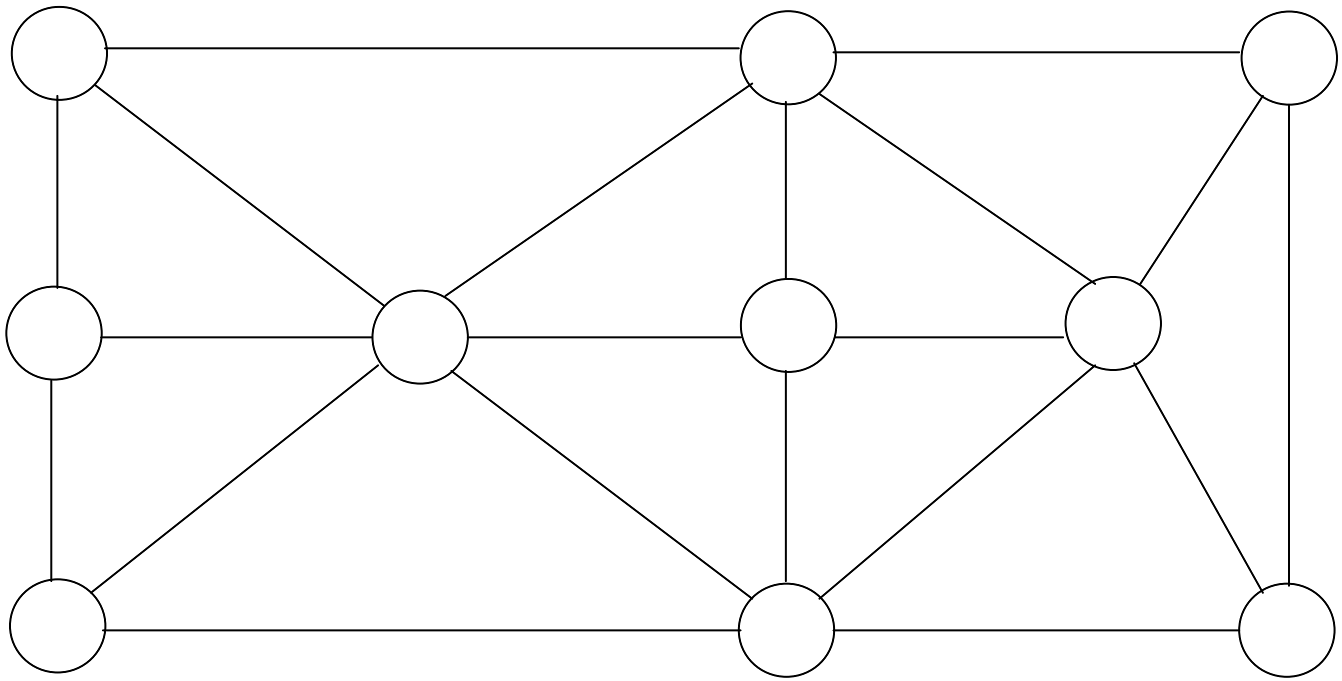
Concurrent Counting is harder than Queuing

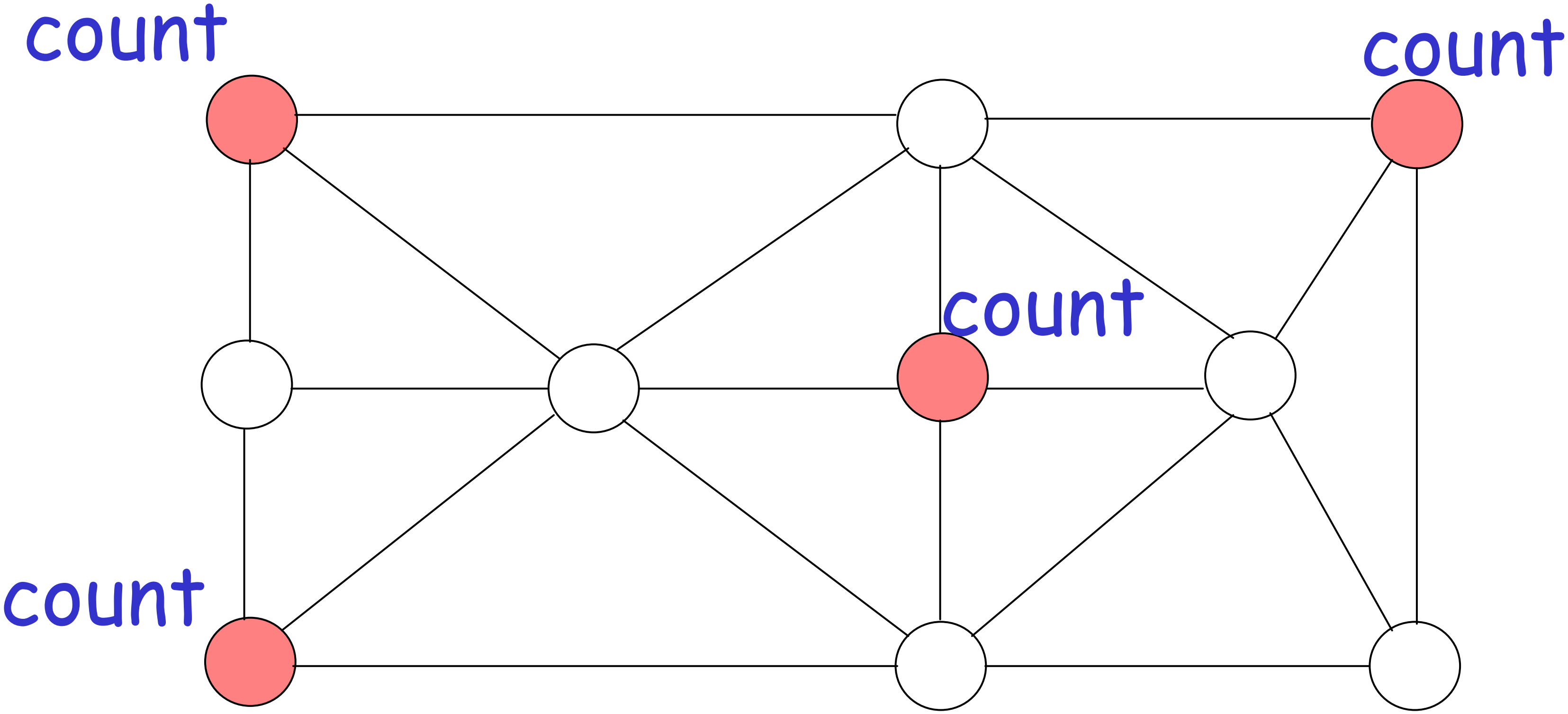
Costas Busch
Rensselaer Polytechnic Intitute

Srikanta Tirthapura
Iowa State University

Arbitrary graph

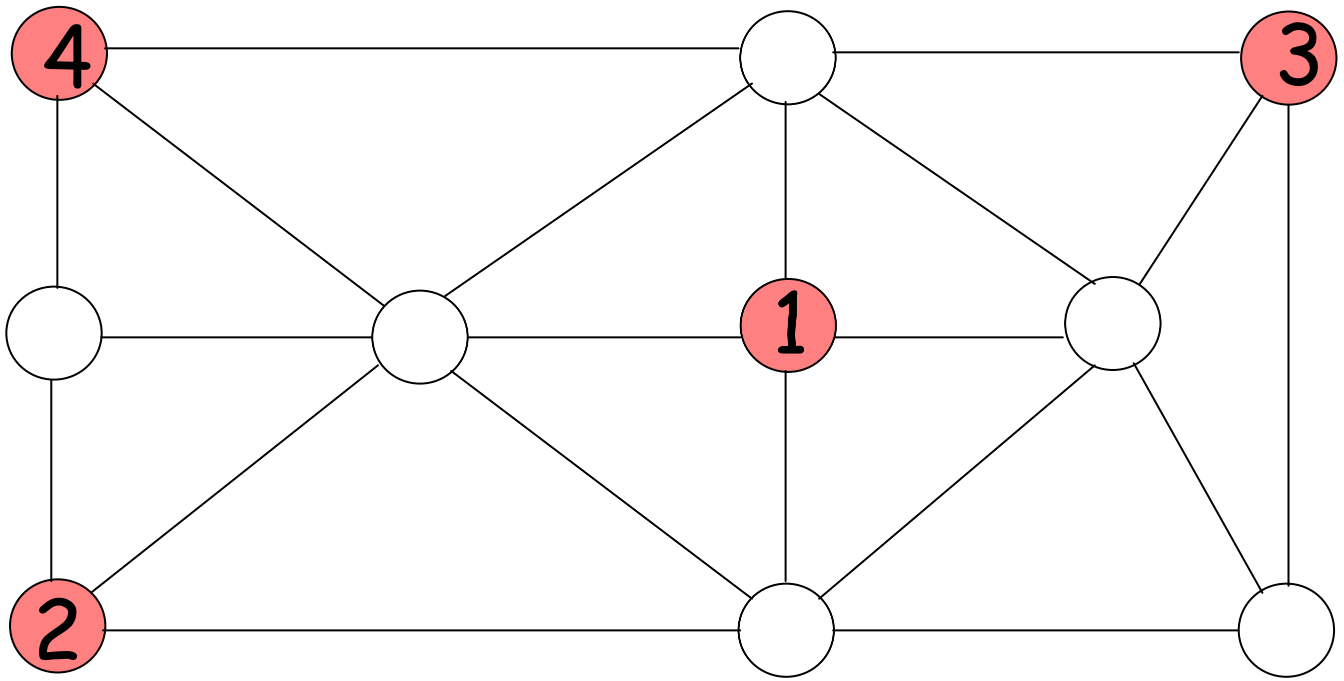


Distributed Counting



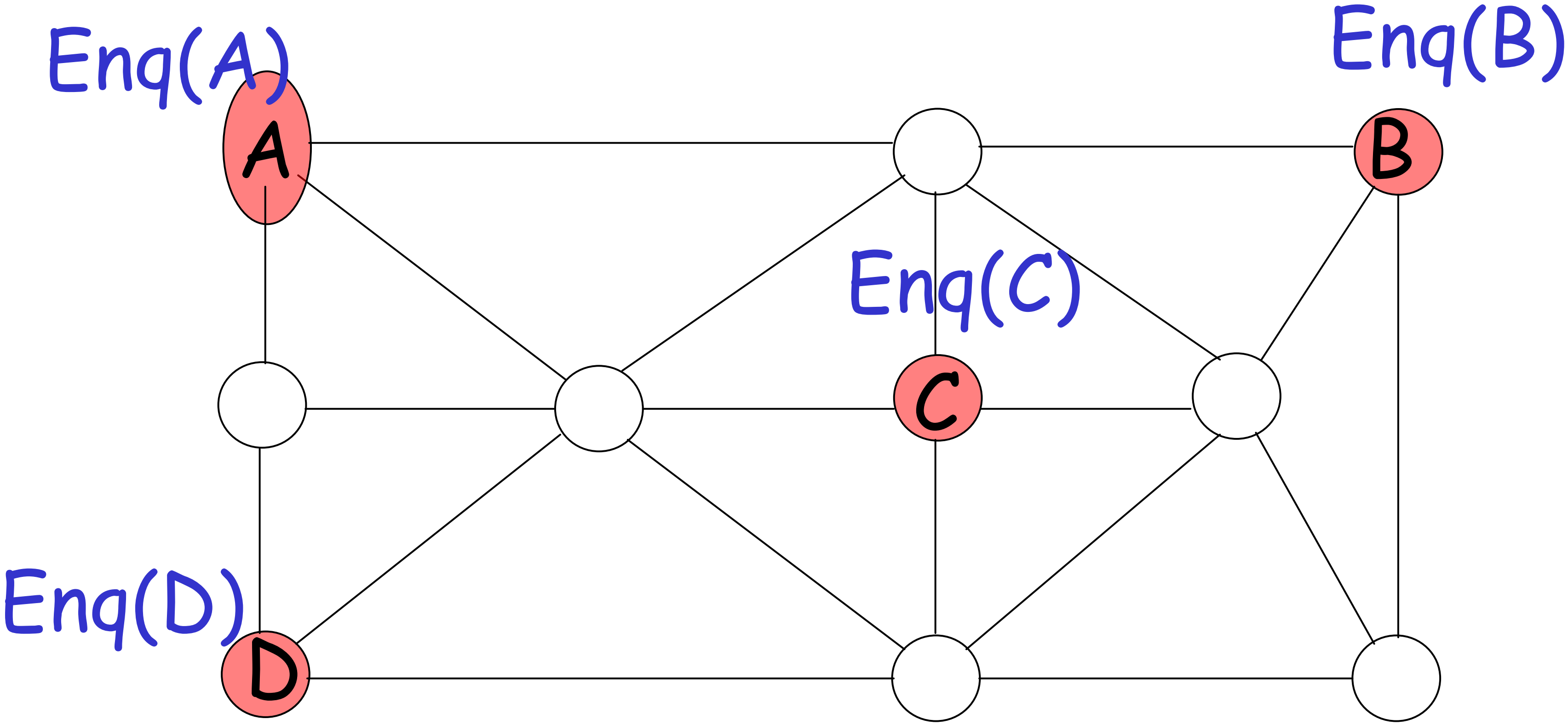
Some processors request a counter value

Distributed Counting



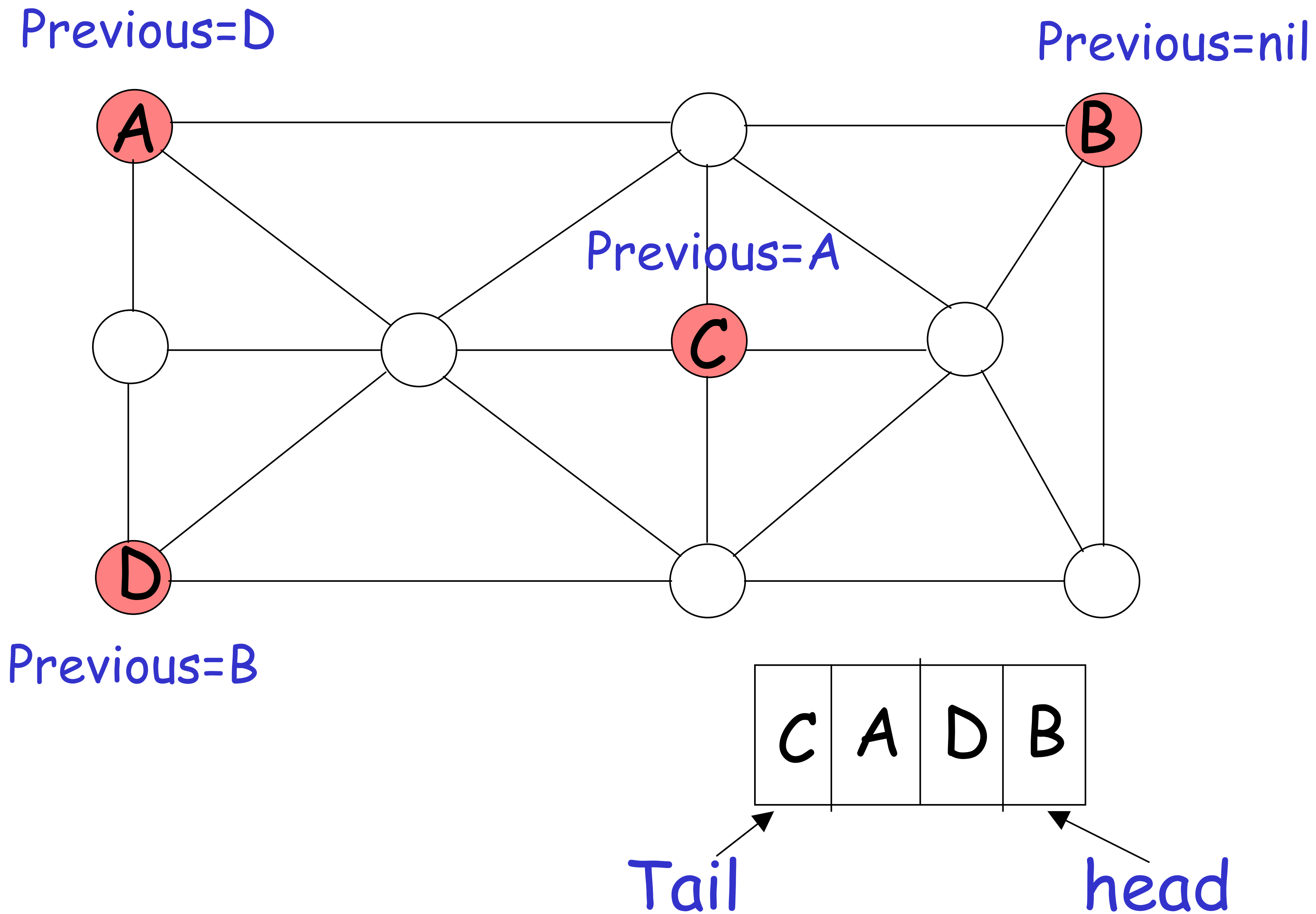
Final state

Distributed Queuing



Some processors perform enqueue operations

Distributed Queuing



Applications

Counting:

- parallel scientific applications
- load balancing (counting networks)

Queuing:

- distributed directories for mobile objects
- distributed mutual exclusion

Ordered Multicast

Multicast with the condition:
all messages received at all nodes in the same order

Either Queuing or Counting will do

Which is more efficient?

Queuing vs Counting?

Total orders

- Queuing = finding predecessor
- Needs local knowledge

- Counting = finding rank
- Needs global knowledge

Problem

Is there a formal sense in which Counting is harder problem than Queuing?

Reductions don't seem to help

Our Result

Concurrent Counting is harder than Concurrent Queuing

on a variety of graphs including:

many common interconnection topologies

- complete graph,
- mesh
- hypercube
- perfect binary trees

Model

Synchronous system $G=(V,E)$
- edges of unit delay

Congestion: Each node can process only one message in a single time step

Concurrent one-shot scenario:
- a set R subset V of nodes issue queuing (or counting) operations at time zero
- No more operations added later

Cost Model

$C_Q(v)$: delay till v gets back queuing result

Cost of algorithm A on request set R is $C_Q(A,R) = \sum_{v \in R} C_Q(v)$

Queuing Complexity = $\min_A \{\max_{R \subset V} C_Q(A,R)\}$

Define Counting Complexity Similarly

Lower Bounds on Counting

For arbitrary graphs:

Counting Cost = $\Omega(n \log^* n)$

For graphs with diameter D:

Counting Cost = $\Omega(D^2)$

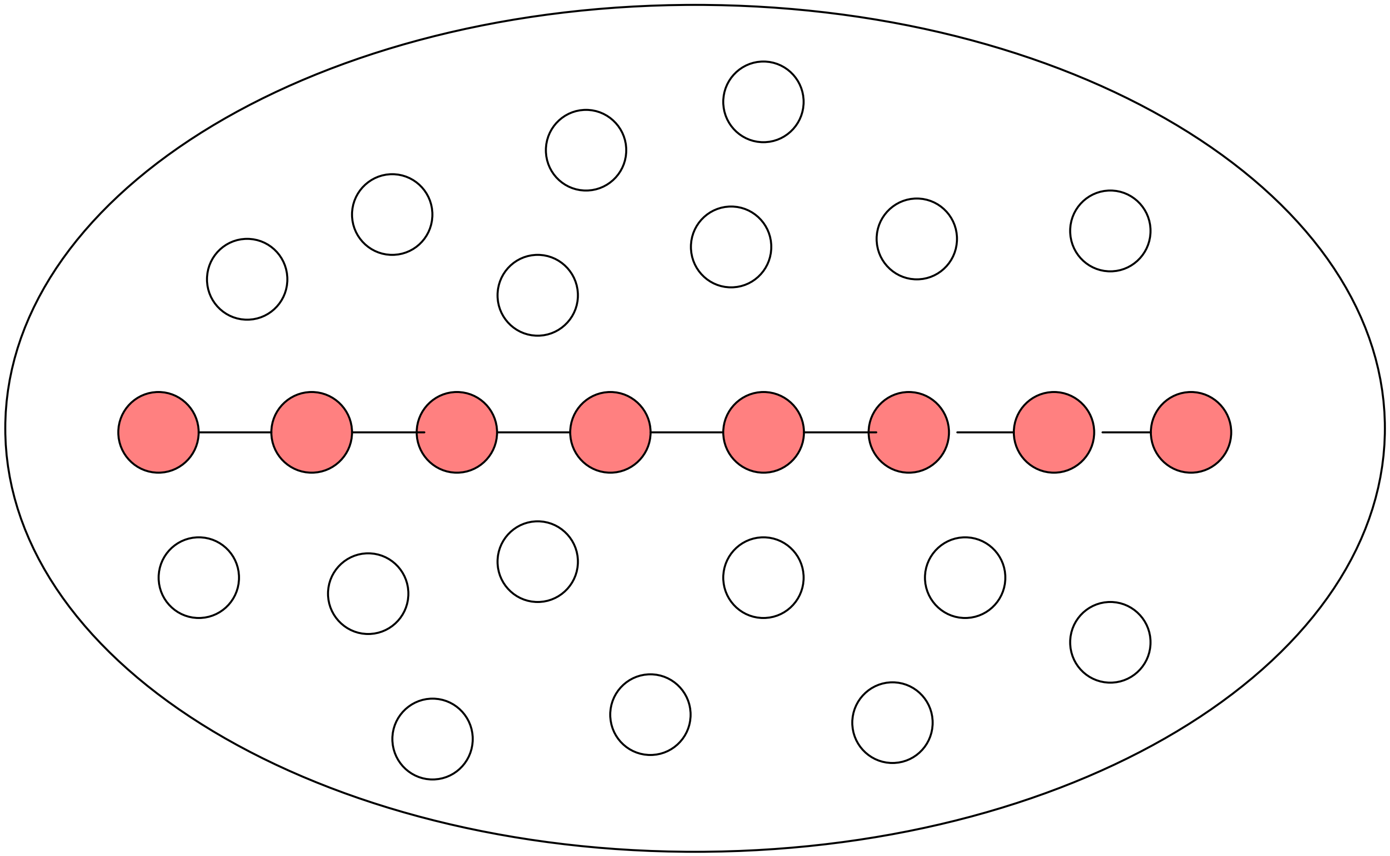
Theorem: For graphs with diameter D:

Counting Cost = $\Omega(D^2)$

Proof:

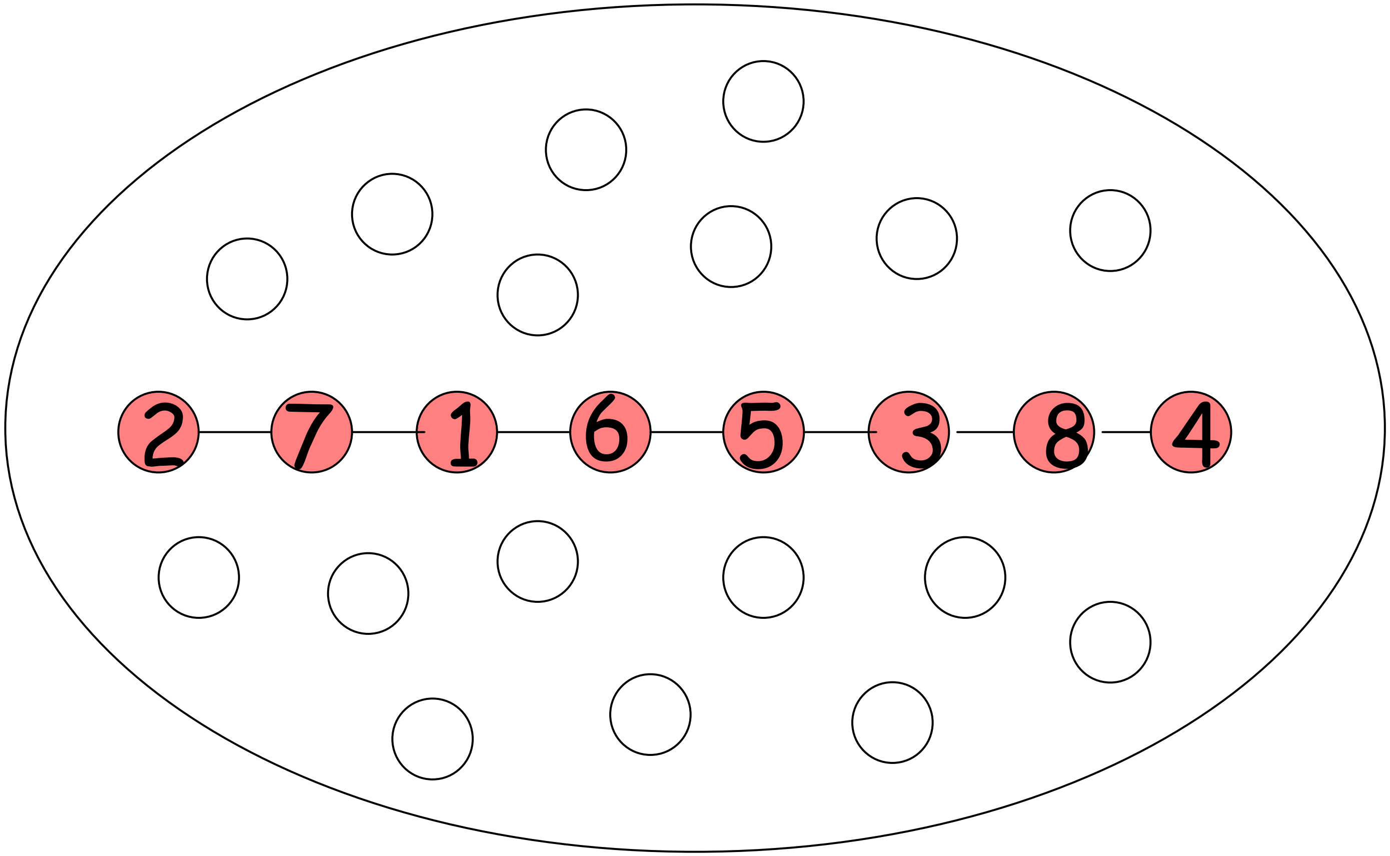
Consider some arbitrary algorithm for counting

Graph



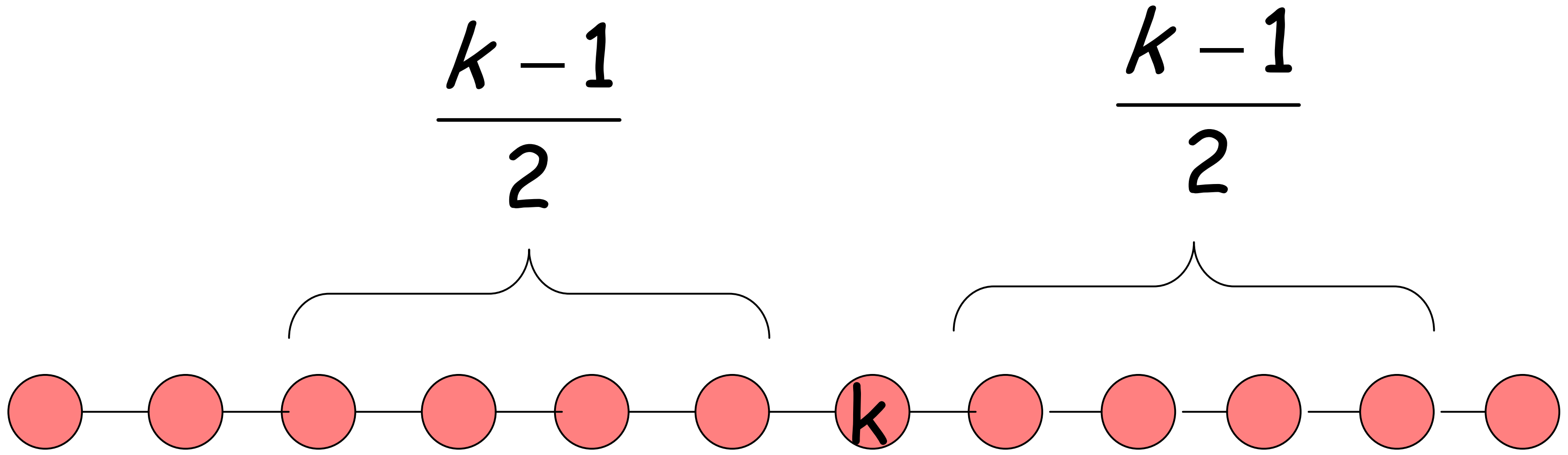
Take shortest path of length D

Graph



make these nodes to count

Node of count k decides after at least $\dfrac{k-1}{2}$ time steps



Needs to be aware of k-1 other processors

Counting Cost: $\sum_{k=1}^{D} \frac{k-1}{2} = \Omega(D^2)$

End of Proof

Theorem: For arbitrary graphs:

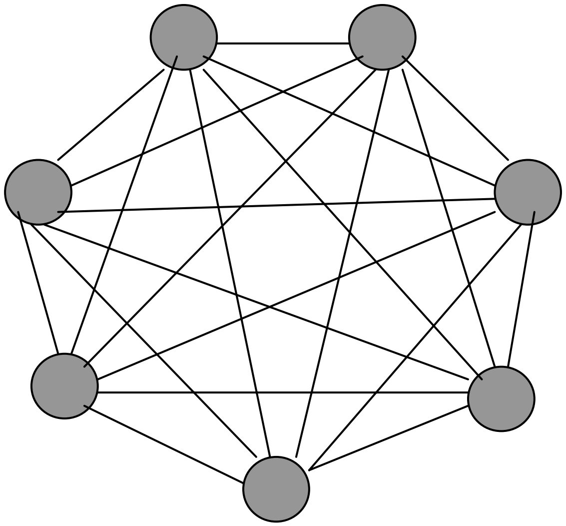
Counting Cost = $\Omega(n \log^* n)$

Proof:

Consider some arbitrary algorithm for counting

Prove it for a complete graph with n nodes:

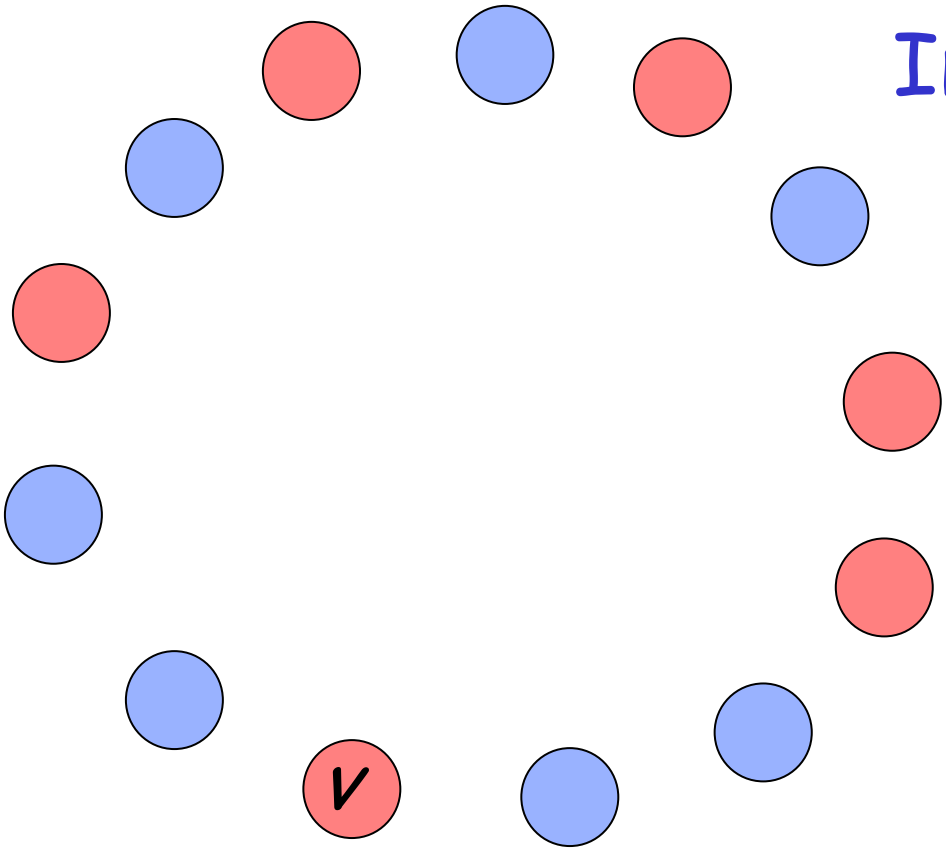
any algorithm on any graph with n nodes can be simulated on the complete graph



The initial state affects the outcome

Red: count

Blue: don't count

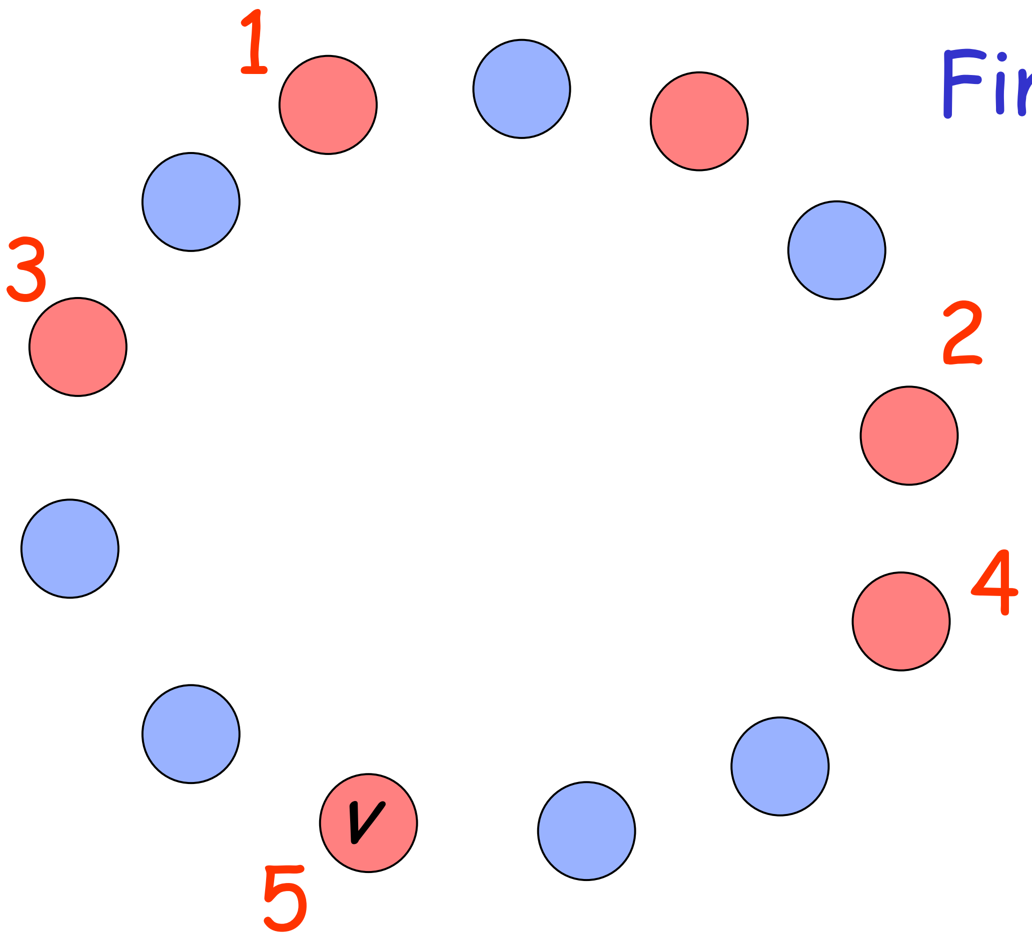


Initial State

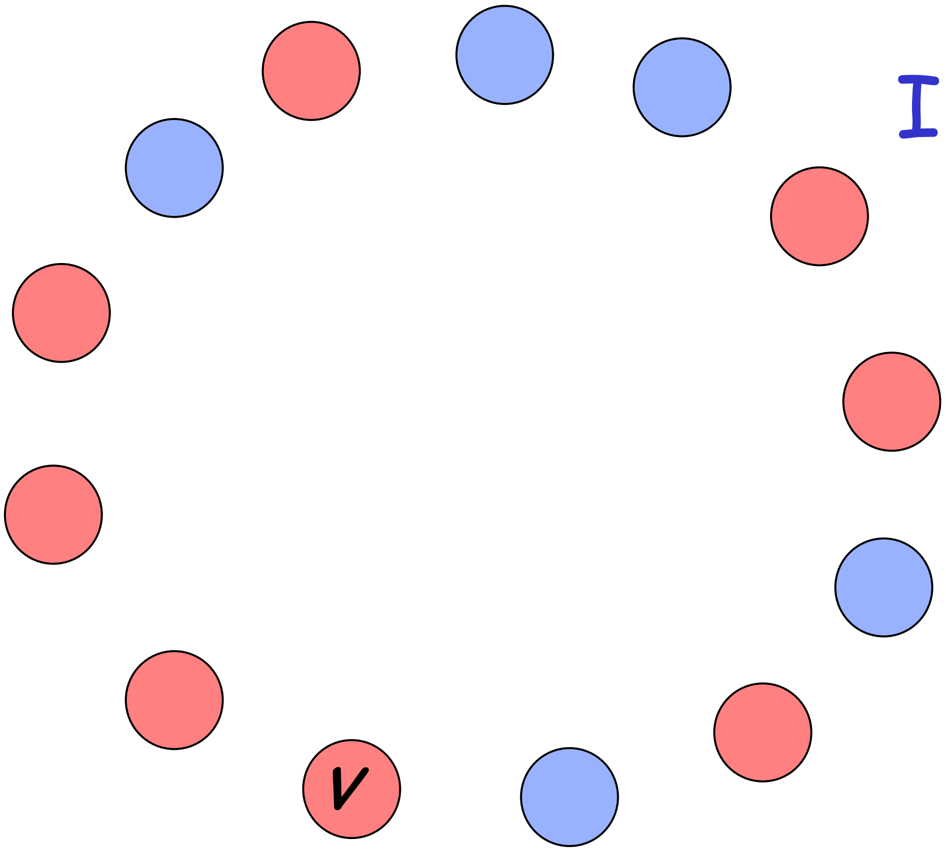
Red: count

Blue: don't count

Final state



Red: count
Blue: don't count

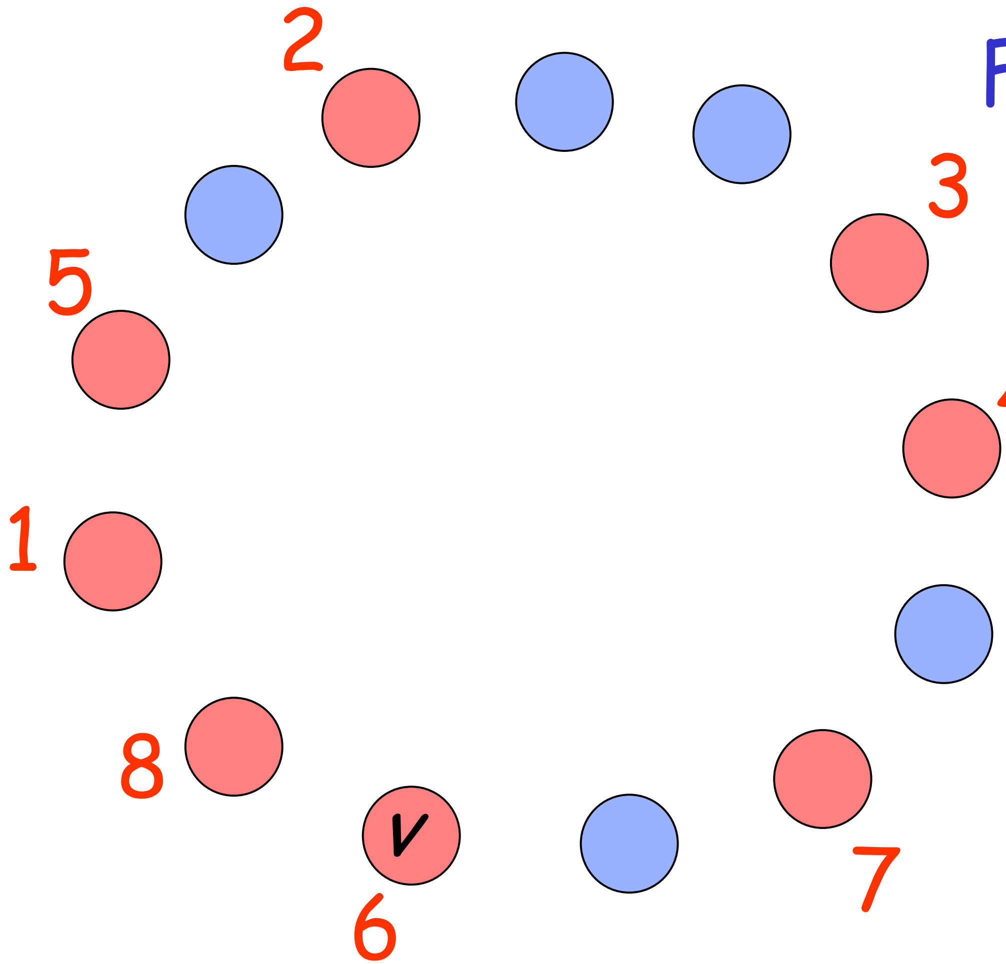


Initial state

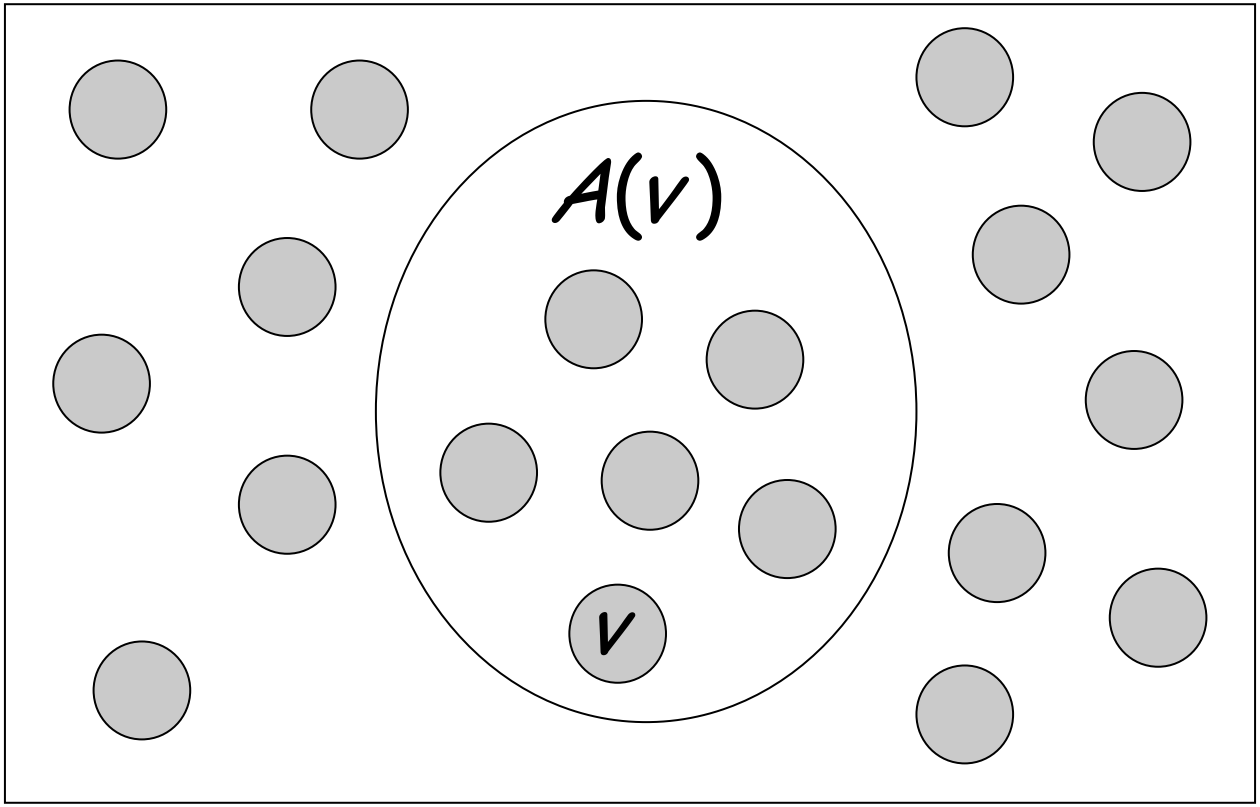
Red: count

Blue: don't count

Final state

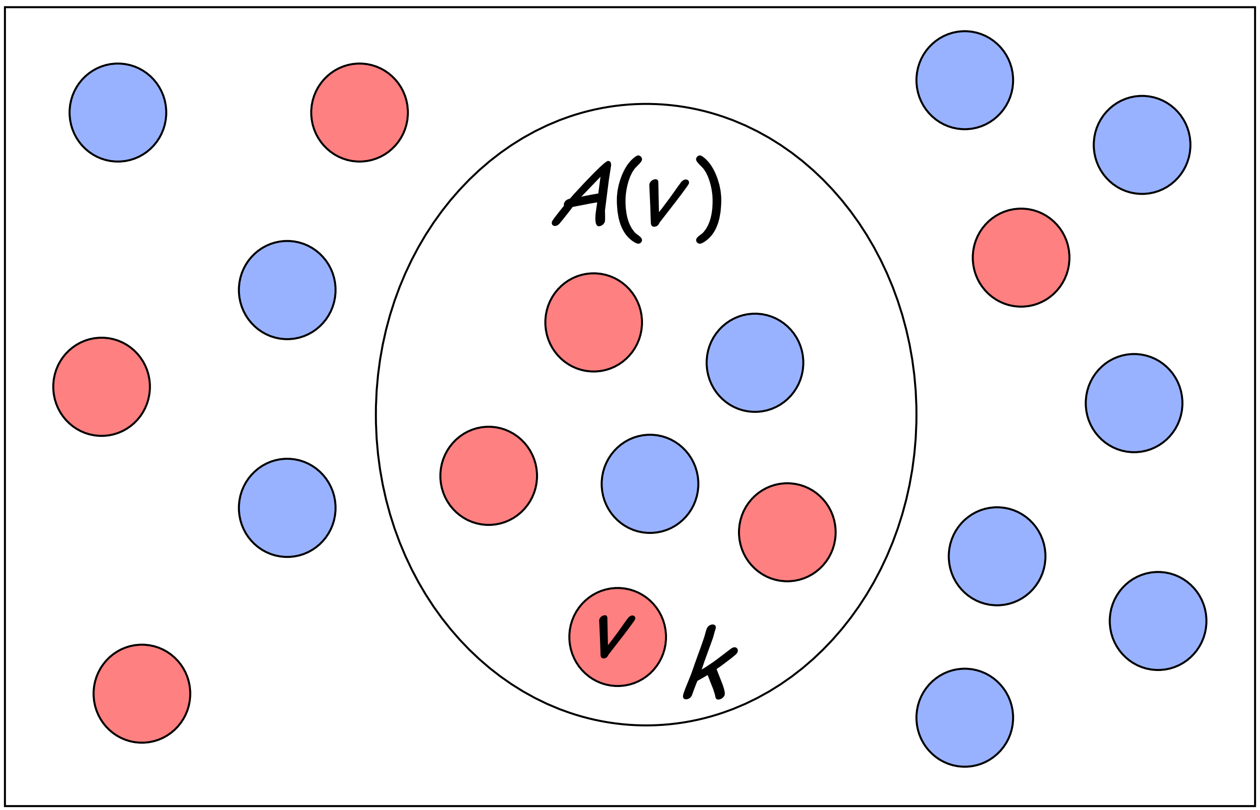


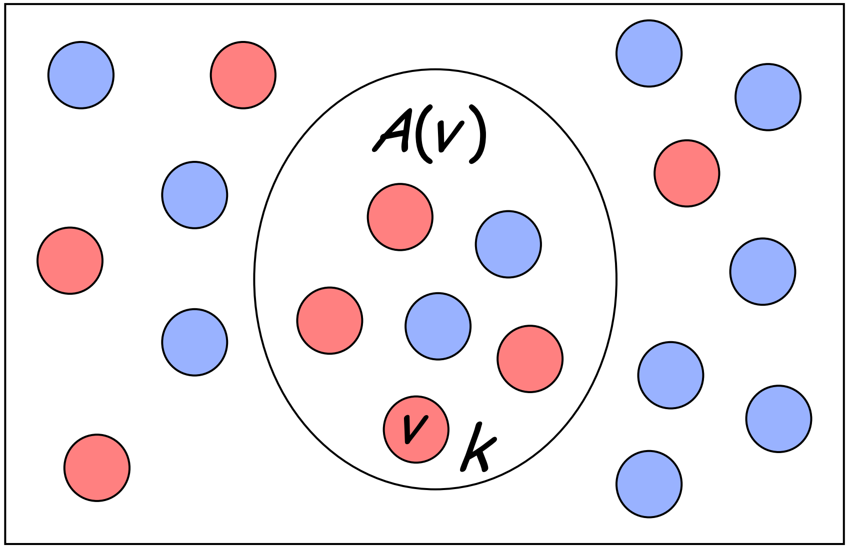
Let $A(v)$ be the set of nodes whose input may affect the decision of v



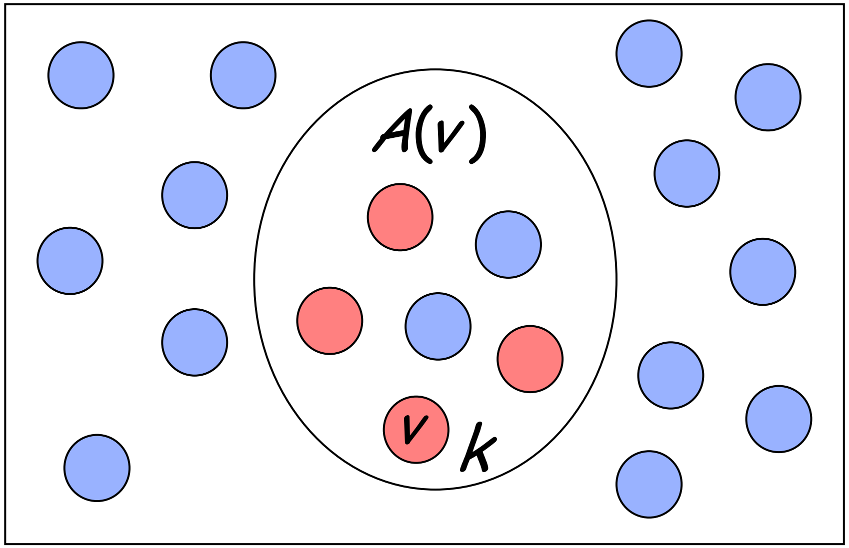
Suppose that there is an initial state for which v decides k

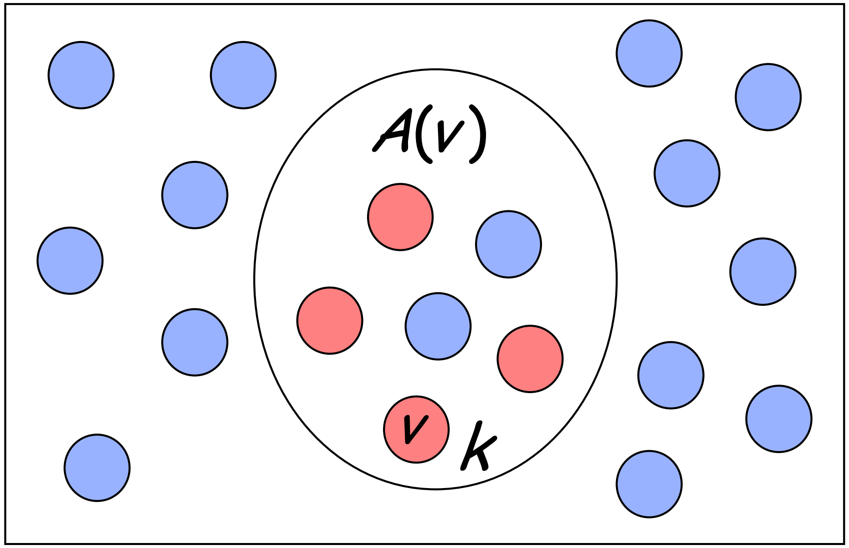
Then: $|A(v)| \geq k$





These two initial states give same result for v





If $|A(v)| < k$, then v would decide less than k

Thus, $|A(v)| \geq k$

Suppose that v decides at time t

We show:

$$|A(v)| \le \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\right\} t \text{ times}$$

Suppose that v decides at time t

$$|A(v)| \leq \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\right\} t \text{ times}$$

$$|A(v)| \geq k$$

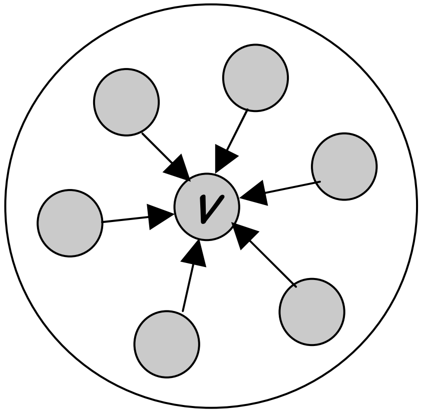
$$\Rightarrow t \geq \log^* k$$

Cost of node v : $t \geq \log^* k$

If n nodes wish to count:

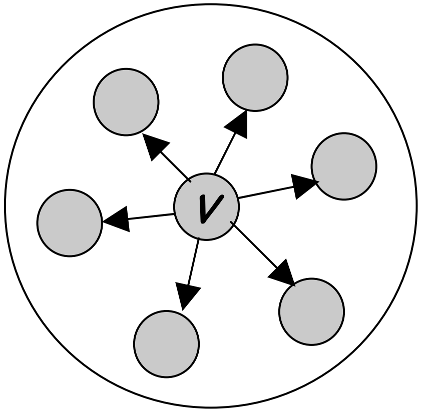
Counting Cost = $\sum_{k=1}^{n} \log^* k = \Omega(n \log^* n)$

$A(v,t)$



Nodes that
affect v
up to time t

$B(v,t)$



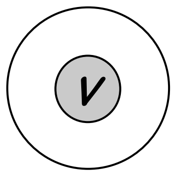
Nodes that
v affects
up to time t

$$a(t) = \max_x |A(x,t)|$$

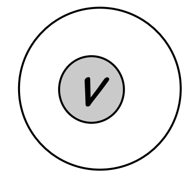
$$b(t) = \max_x |B(x,t)|$$

$A(v, t=1)$

$B(v, t=1)$

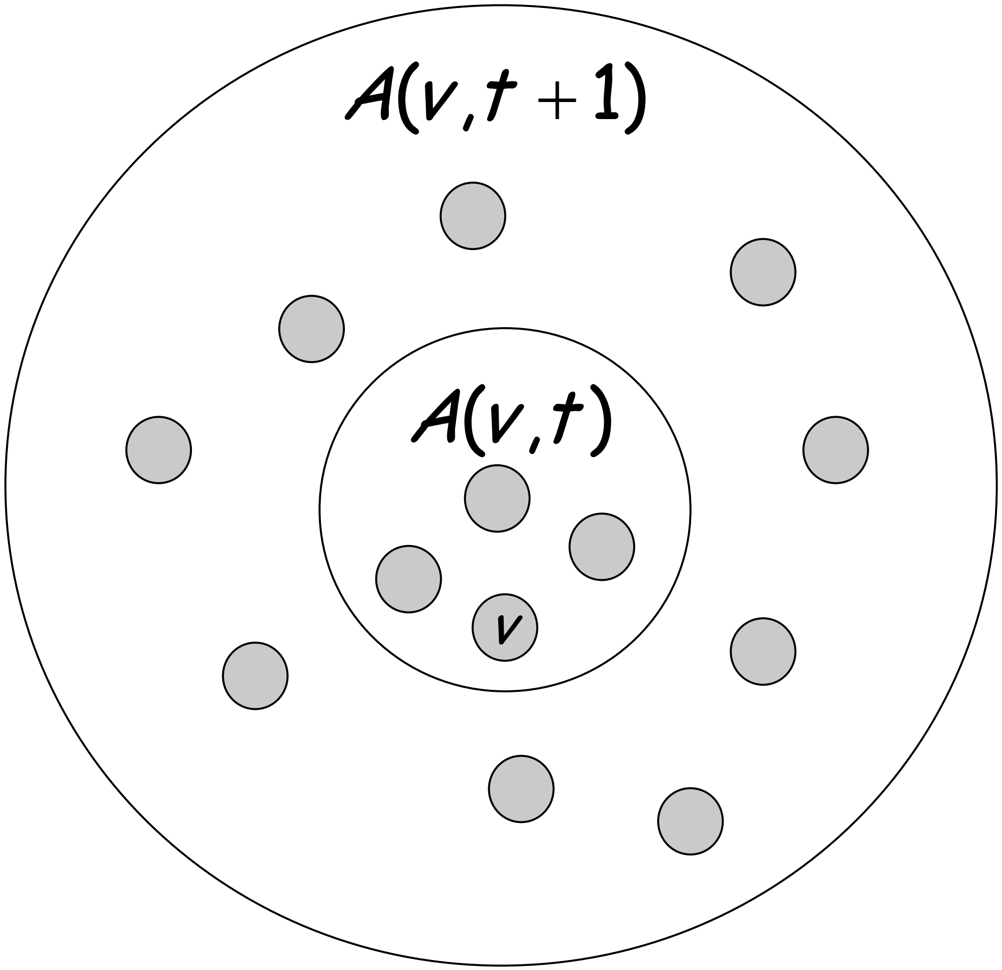


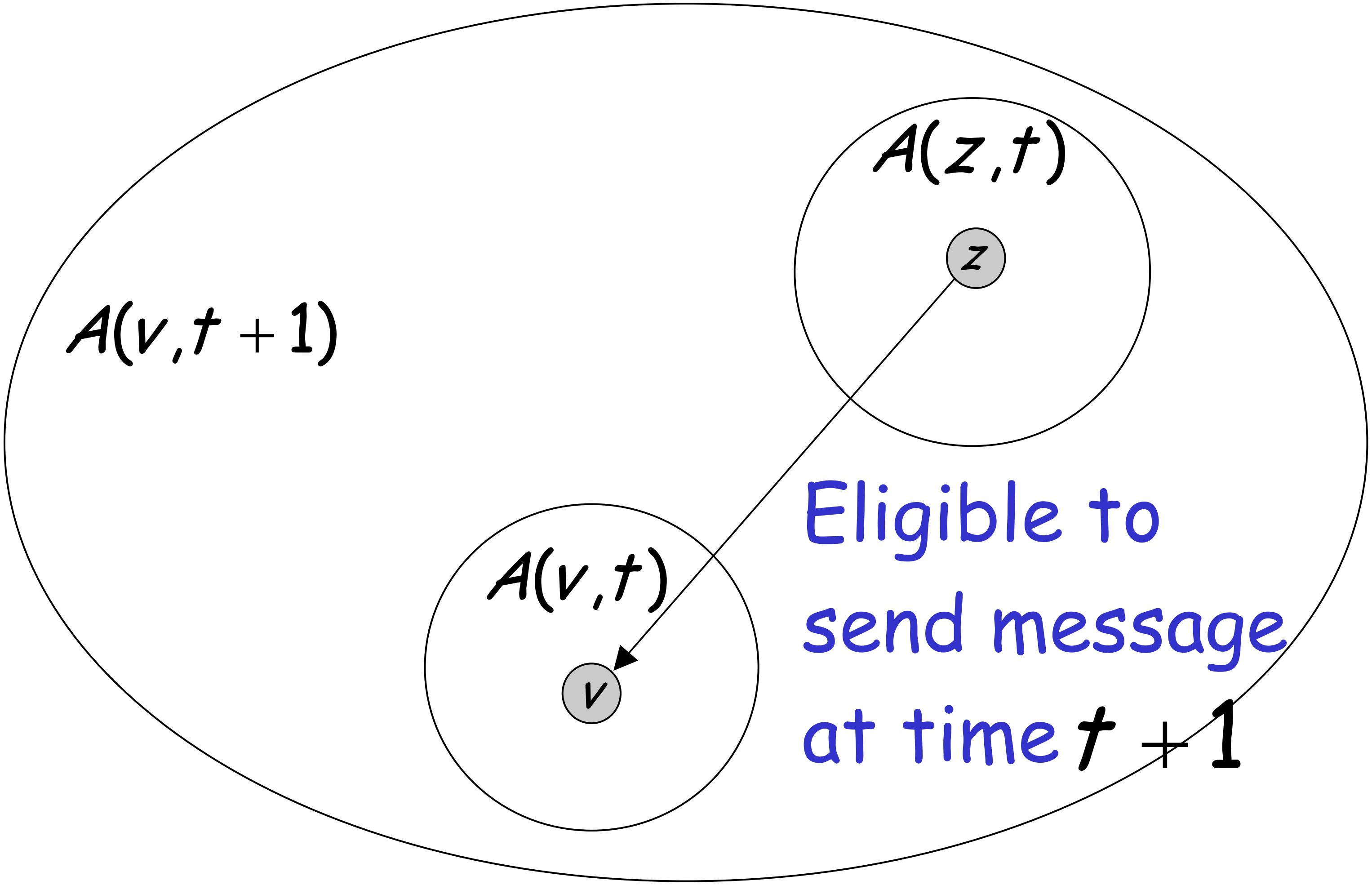
$a(t) = 1$



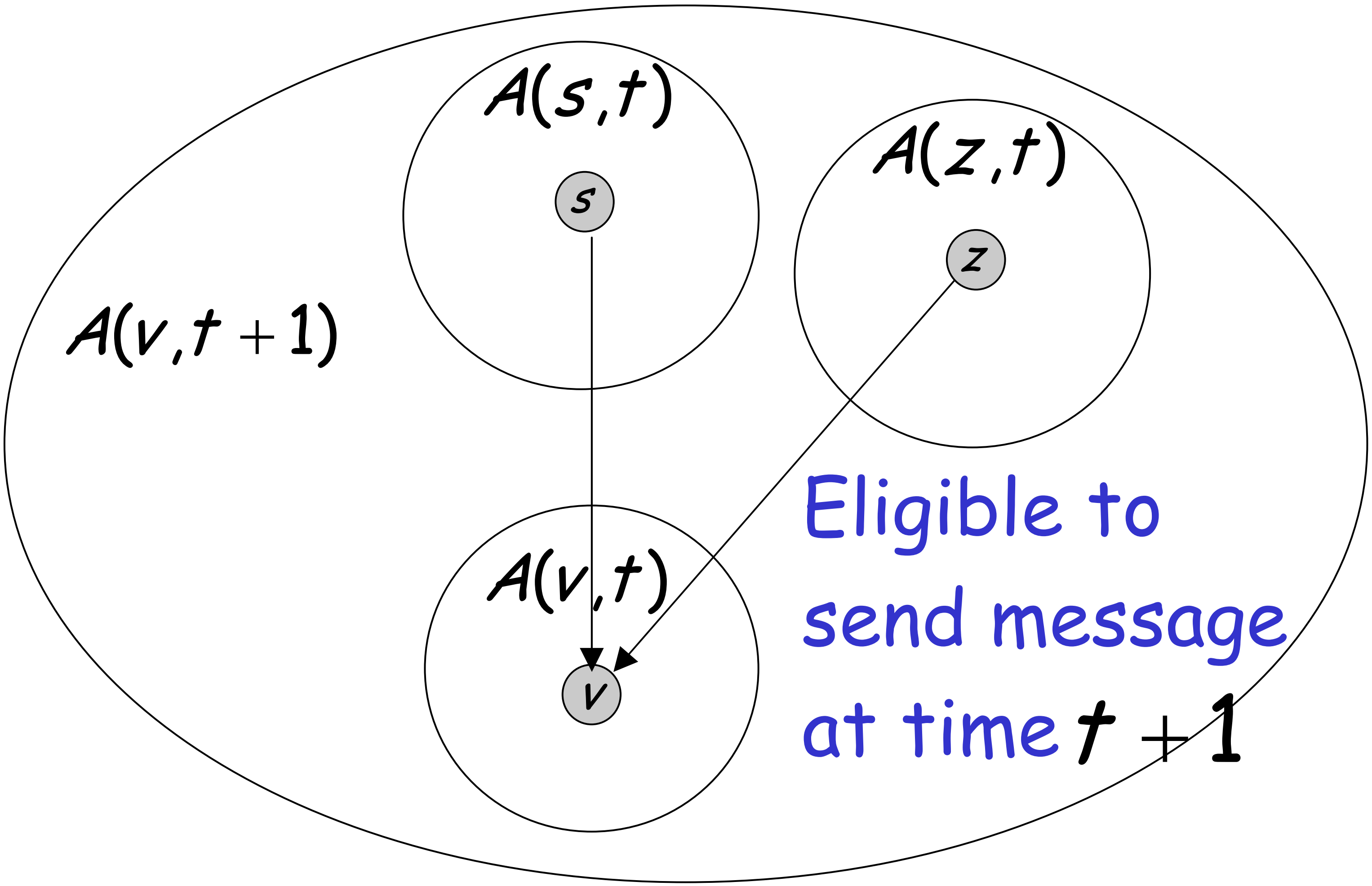
$b(t) = 1$

After $t = 1$, the sets grow


$A(v, t+1)$
$A(v, t)$
v

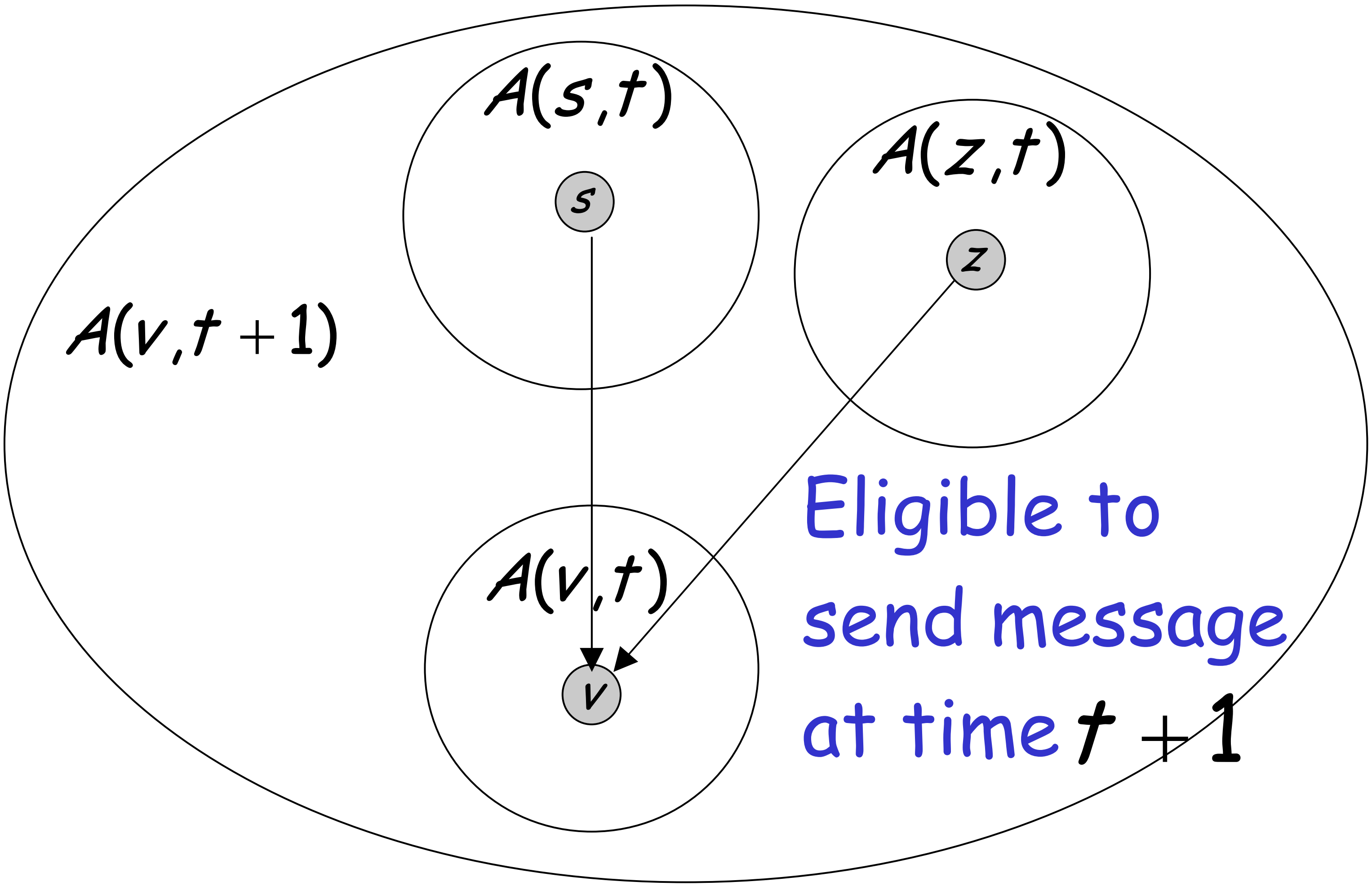


There is an initial state such that that z sends a message to v



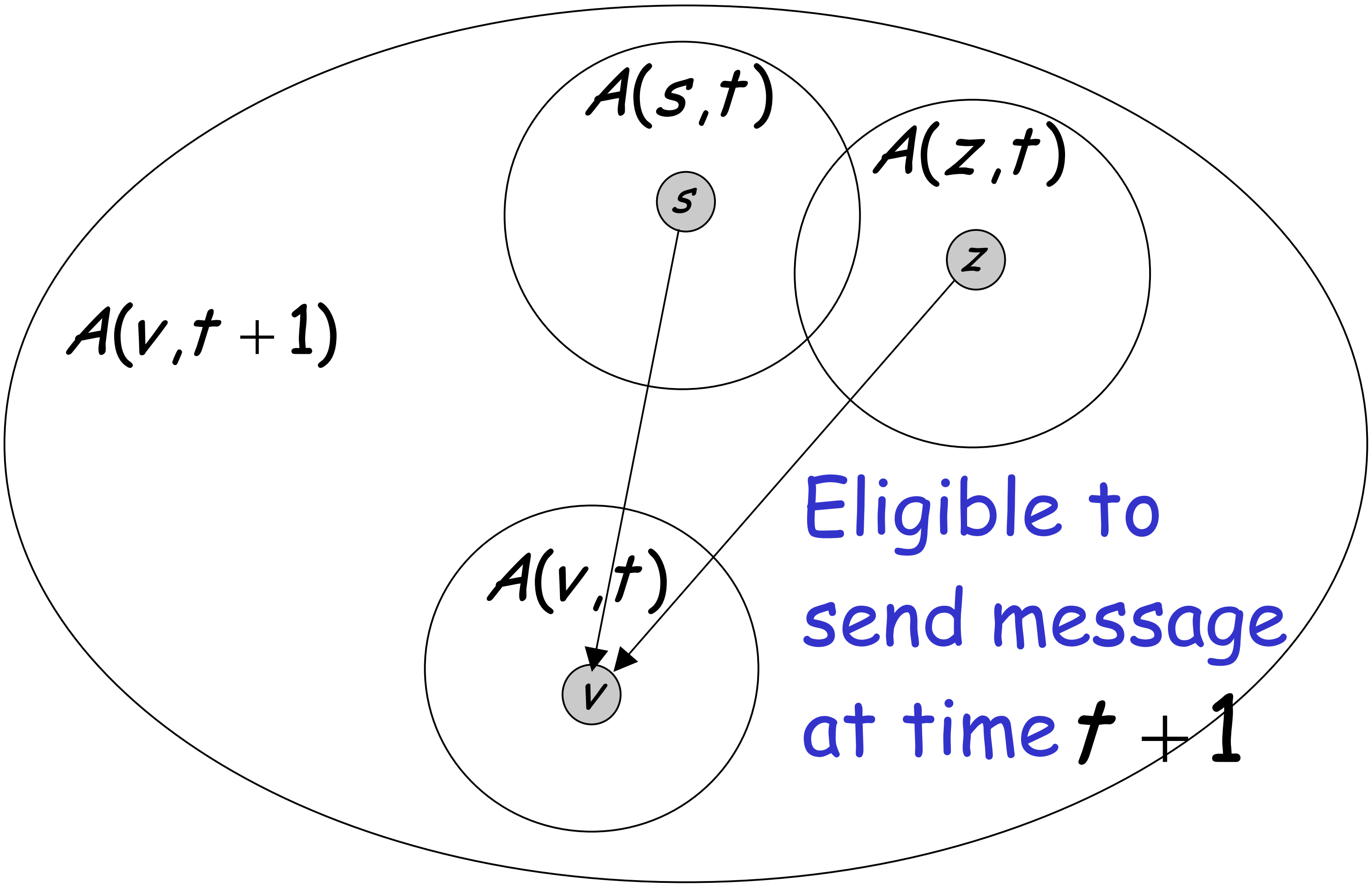
Suppose that $A(s,t) \cap A(z,t) = \emptyset$

Then, there is an initial state such that both send message to v

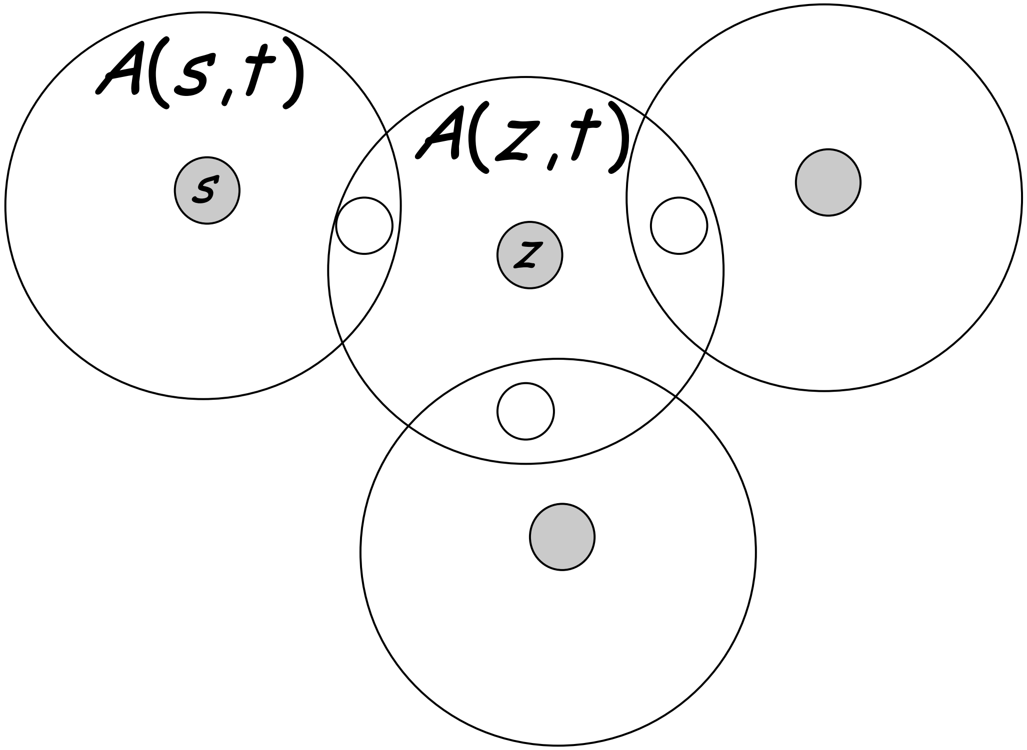


Eligible to send message at time $t+1$

However, v can receive one message at a time

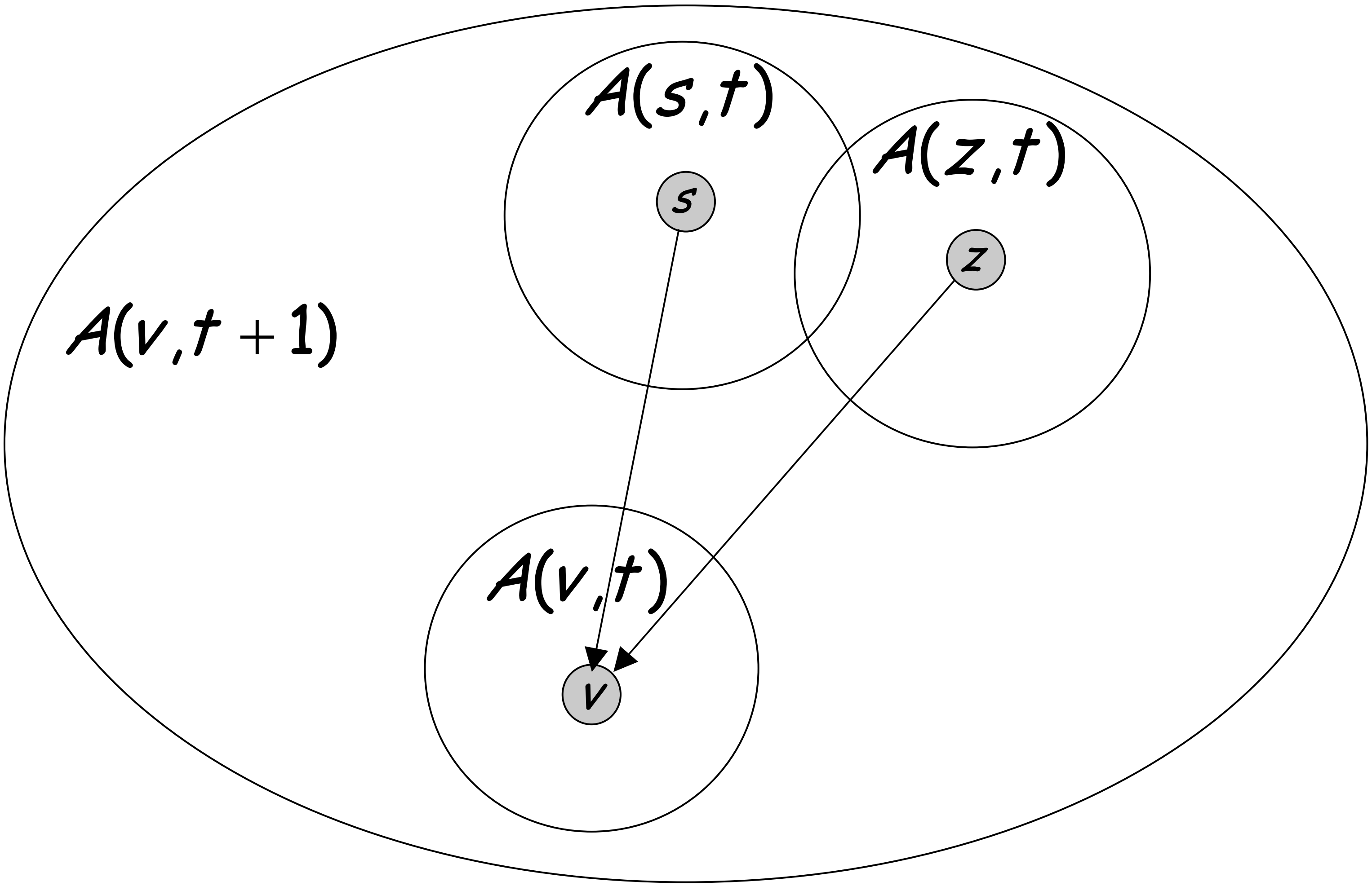


Therefore: $A(s,t) \cap A(z,t) \neq \varnothing$



Number of nodes like s : $\quad a(t) \cdot b(t)$

$\max_x |A(x,t)| \qquad \max_x |B(x,t)|$



Therefore: $\left|A(v,t+1)\right| \le \left|A(v,t)\right| + a(t)\cdot\left(a(t)\cdot b(t)\right)$

Thus: $a(t+1) \le a(t)\big(1 + a(t)\big)$

We can also show: $b(t+1) \le b(t)\big(1 + 2^{a(t)}\big)$

Which give:

$$a(\tau) \le \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\right\} \begin{matrix}\tau \\ \text{times}\end{matrix}$$

End of Proof

Upper Bound on Queuing

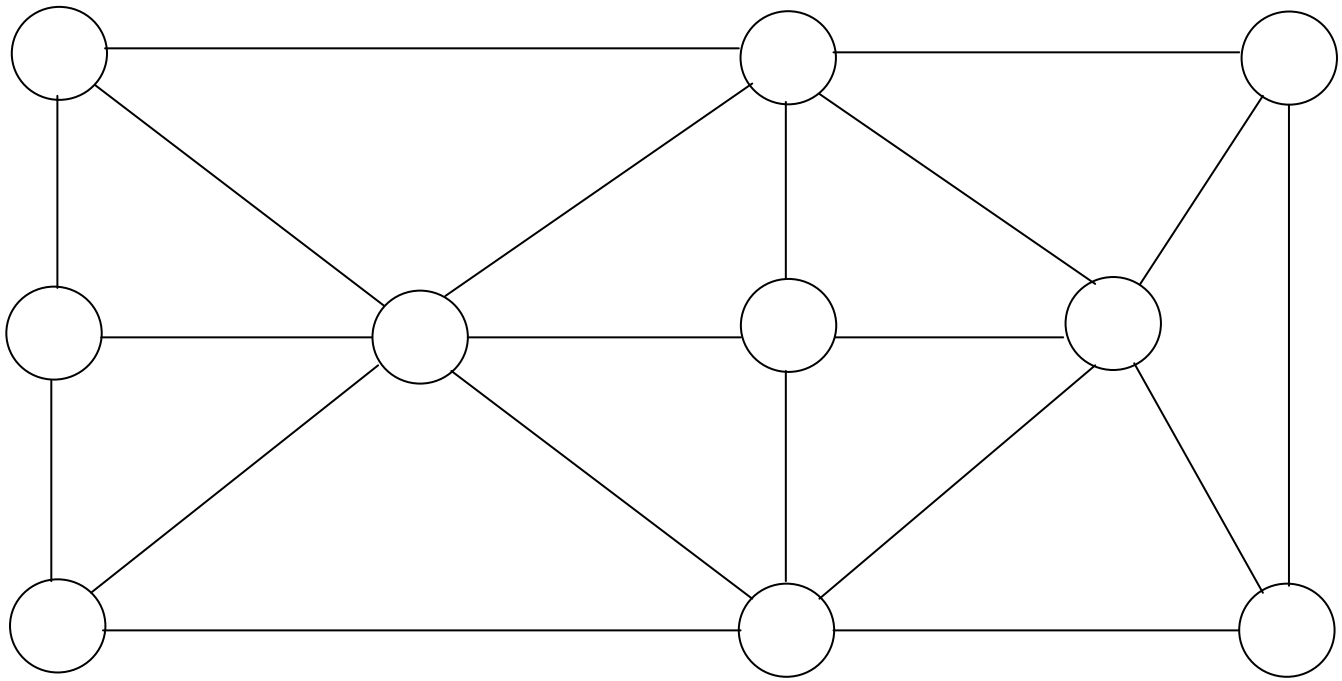
For graphs with spanning trees of constant degree:

Queuing Cost = $O(n\log n)$

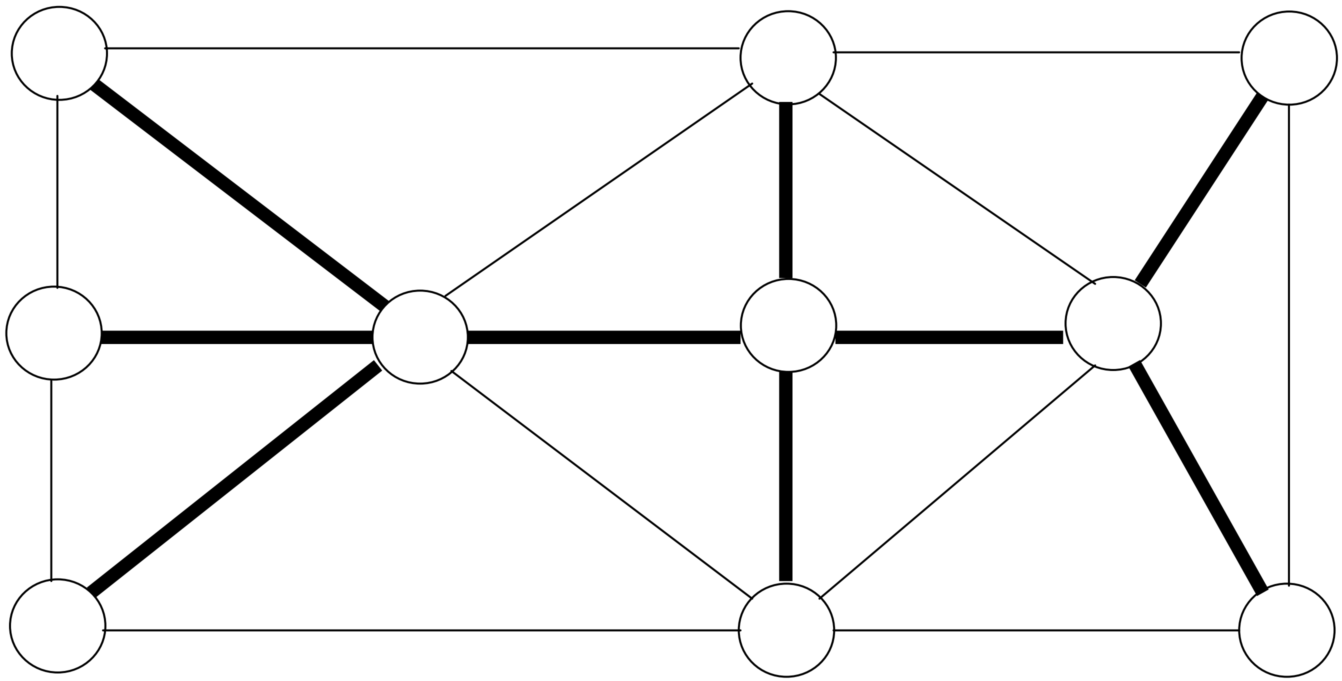
For graphs whose spanning trees are lists or perfect binary trees:

Queuing Cost = $O(n)$

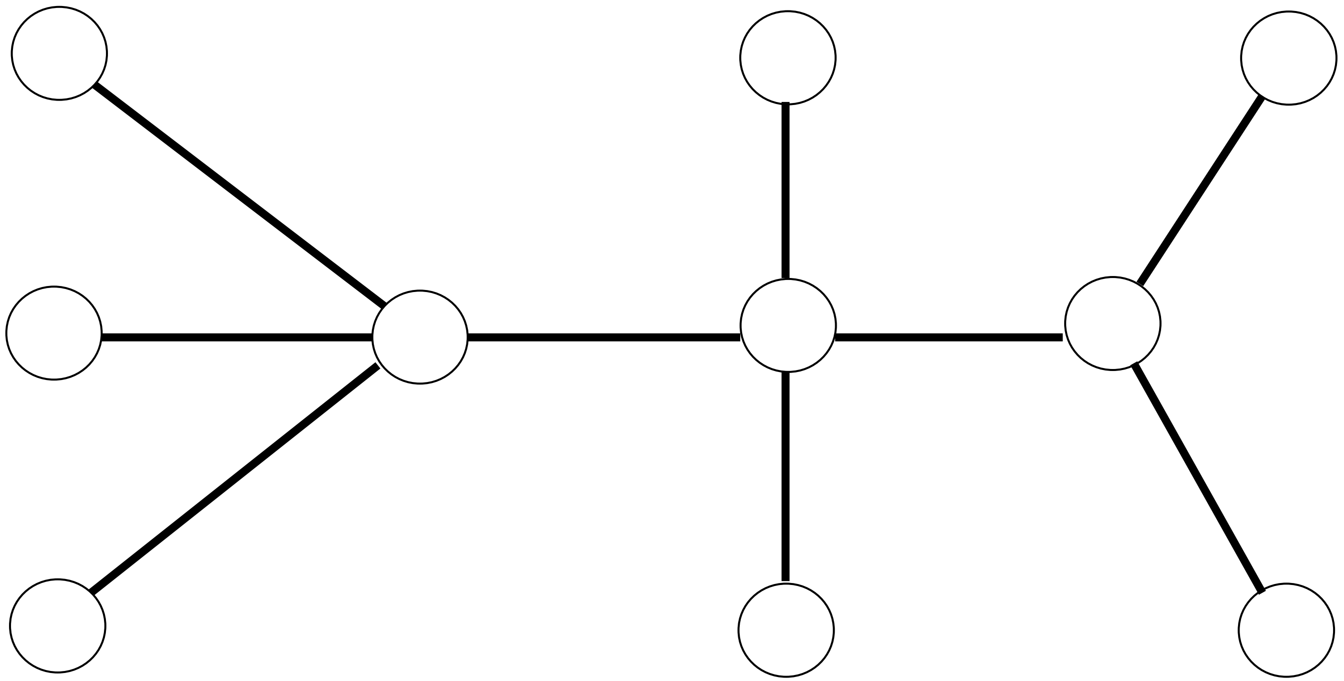
An arbitrary graph



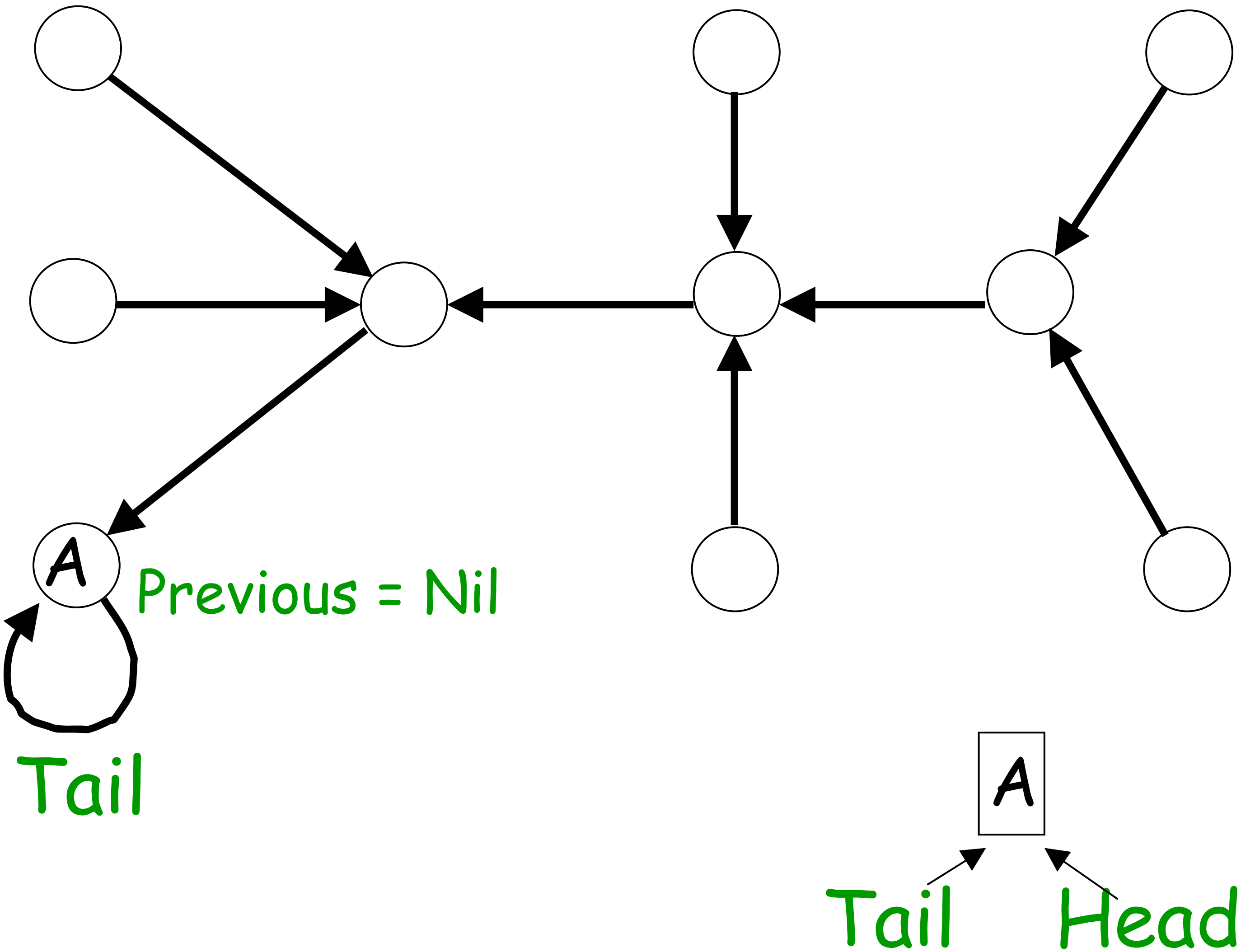
Spanning tree

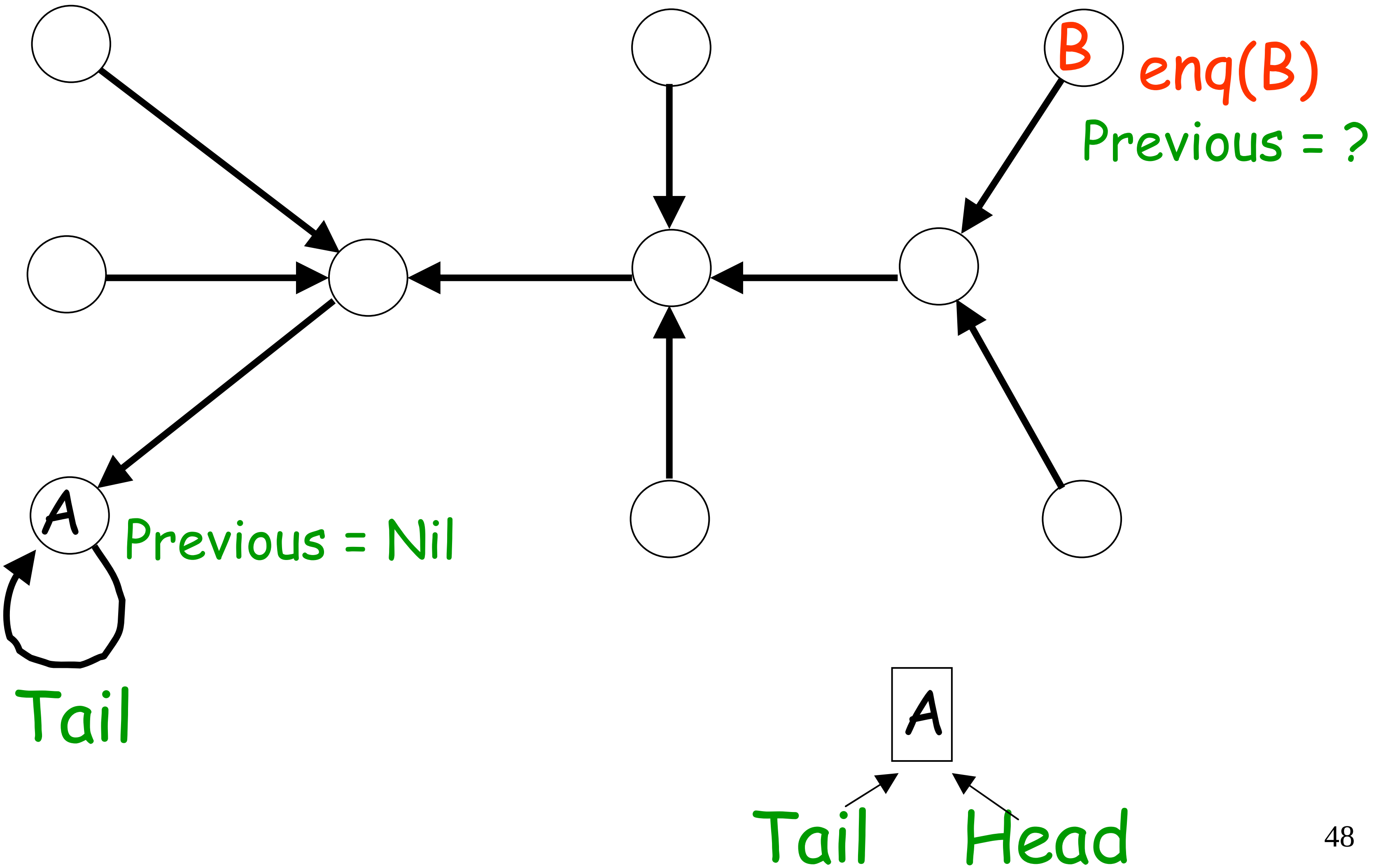


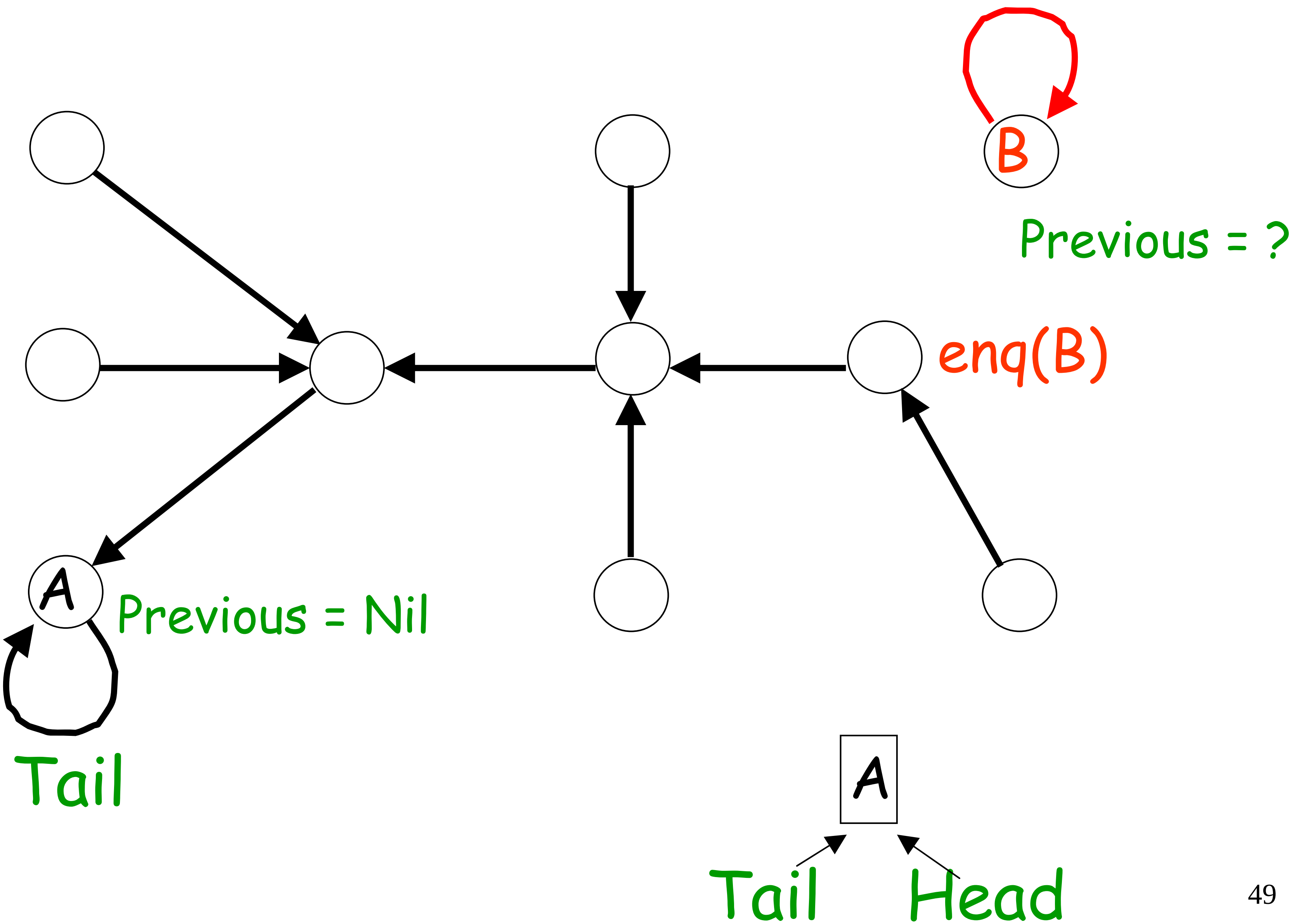
Spanning tree



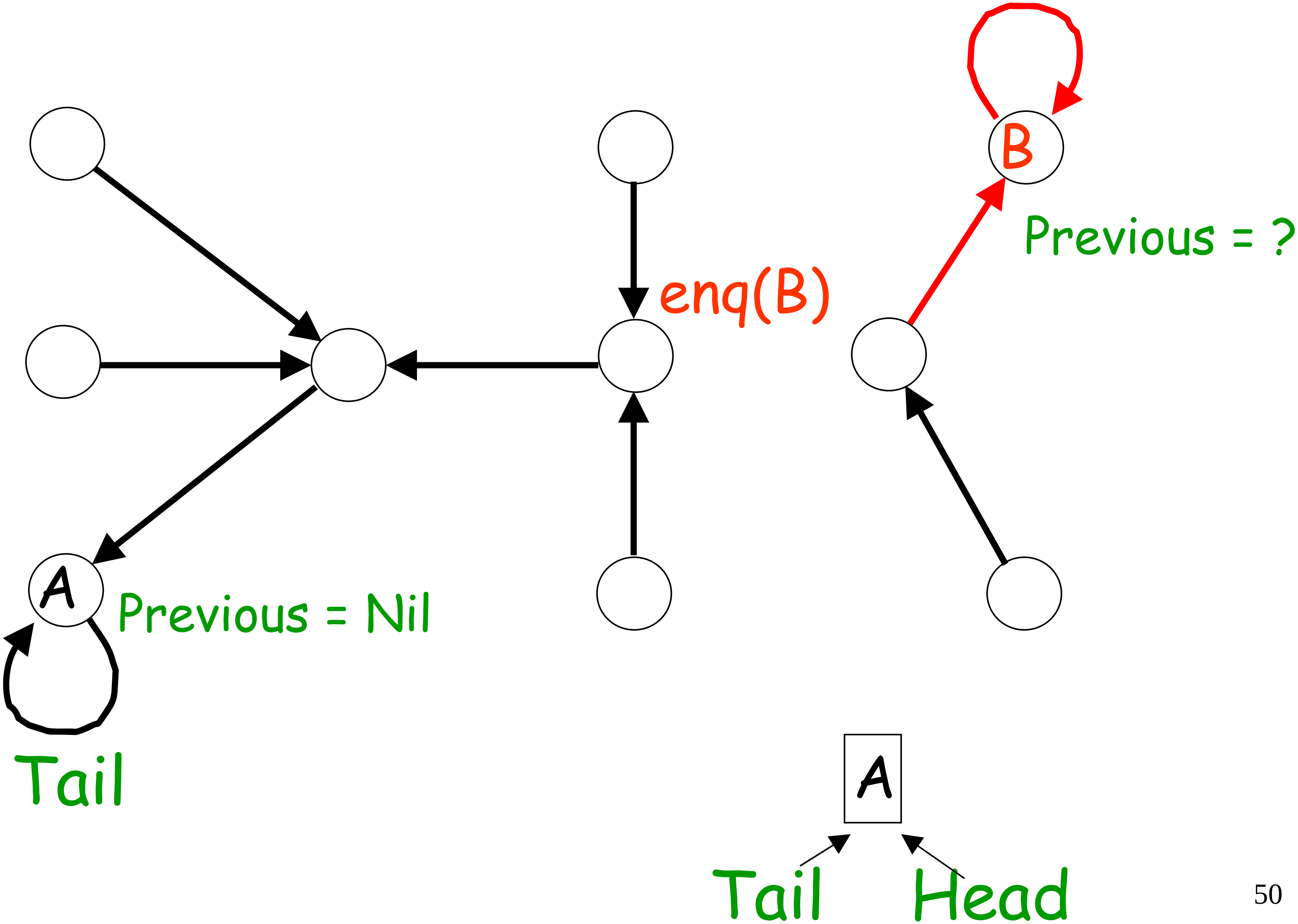
Distributed Queue

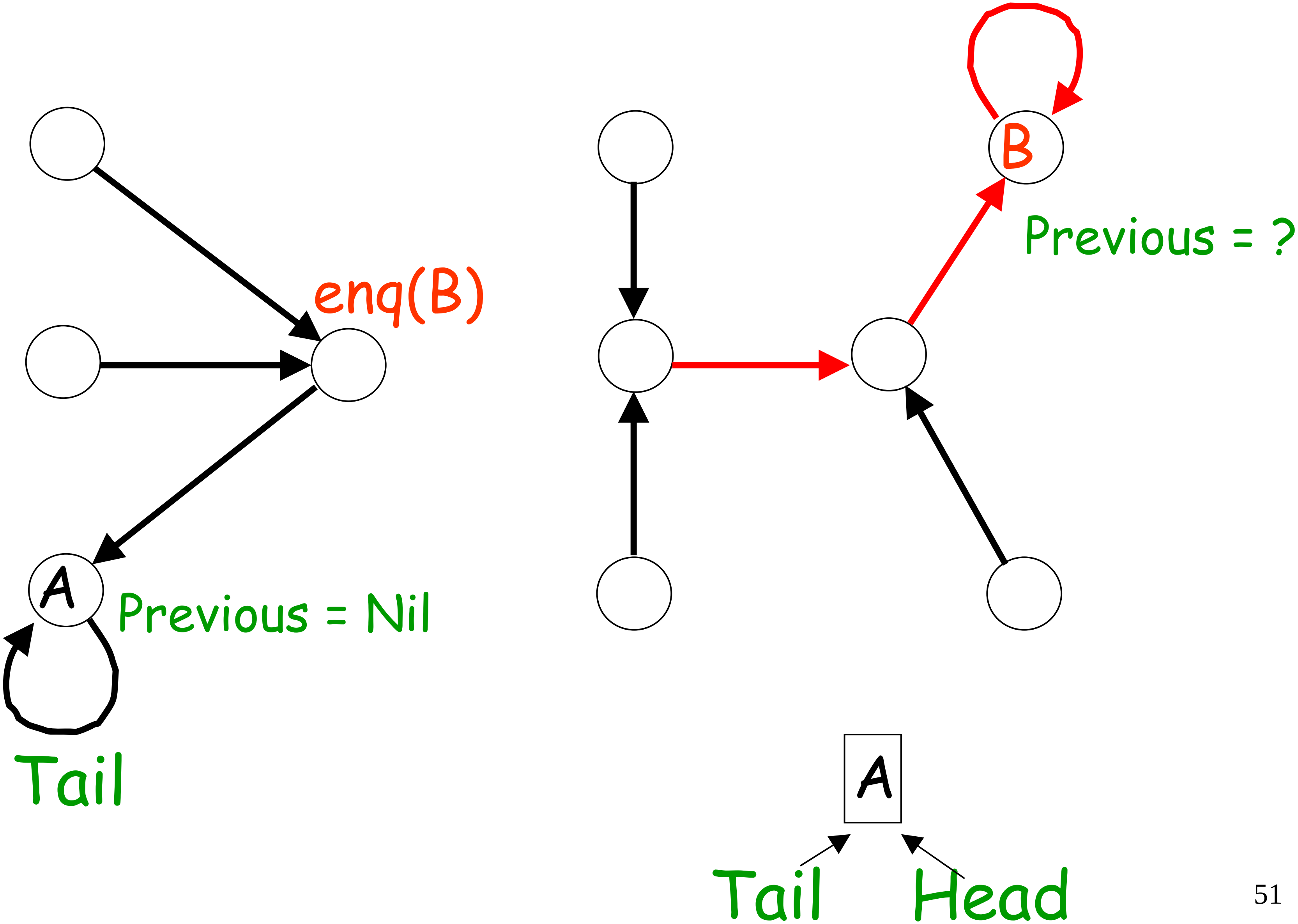


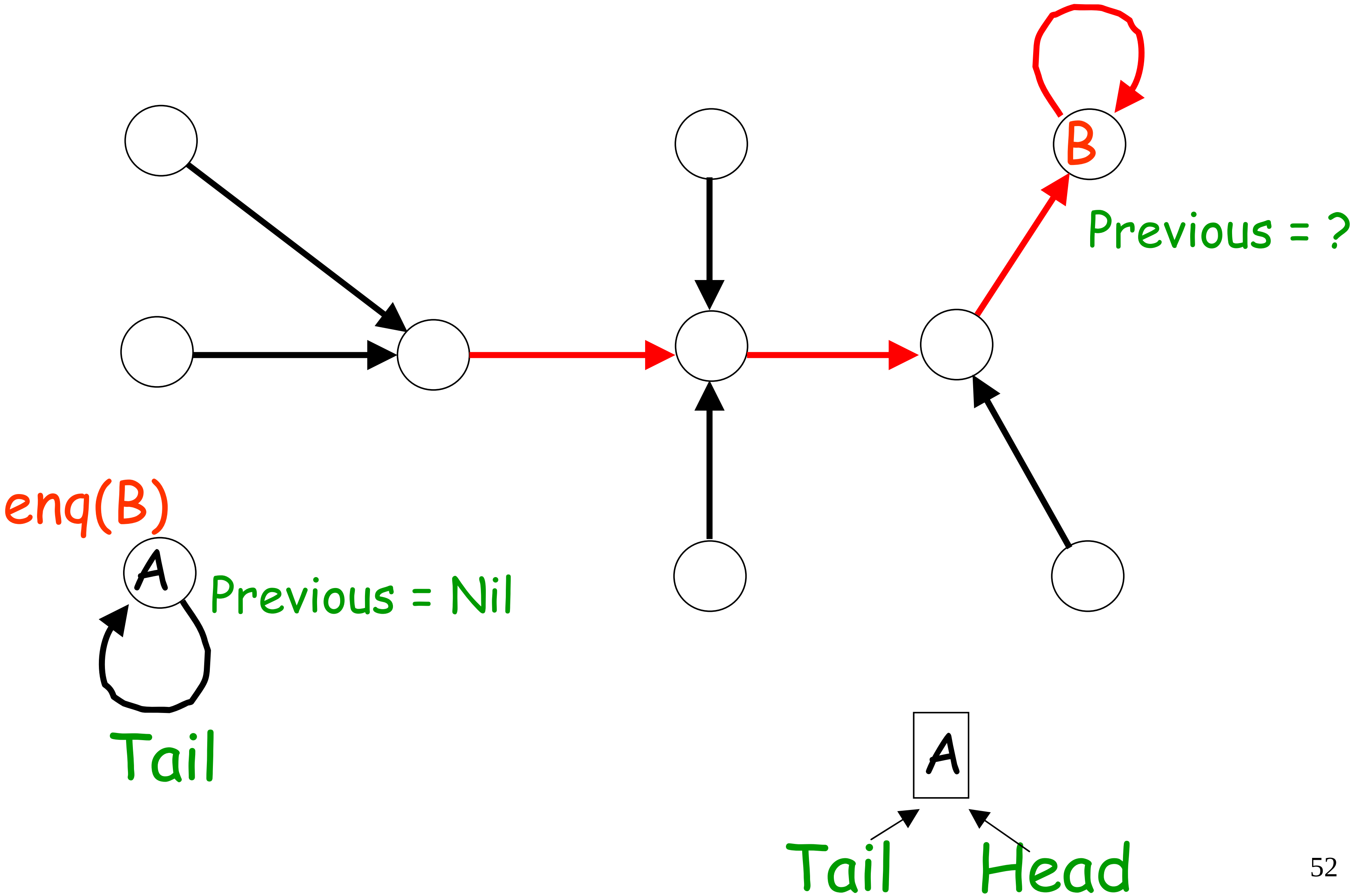

B
enq(B)
Previous = ?
A
Previous = Nil
Tail
A
Tail
Head

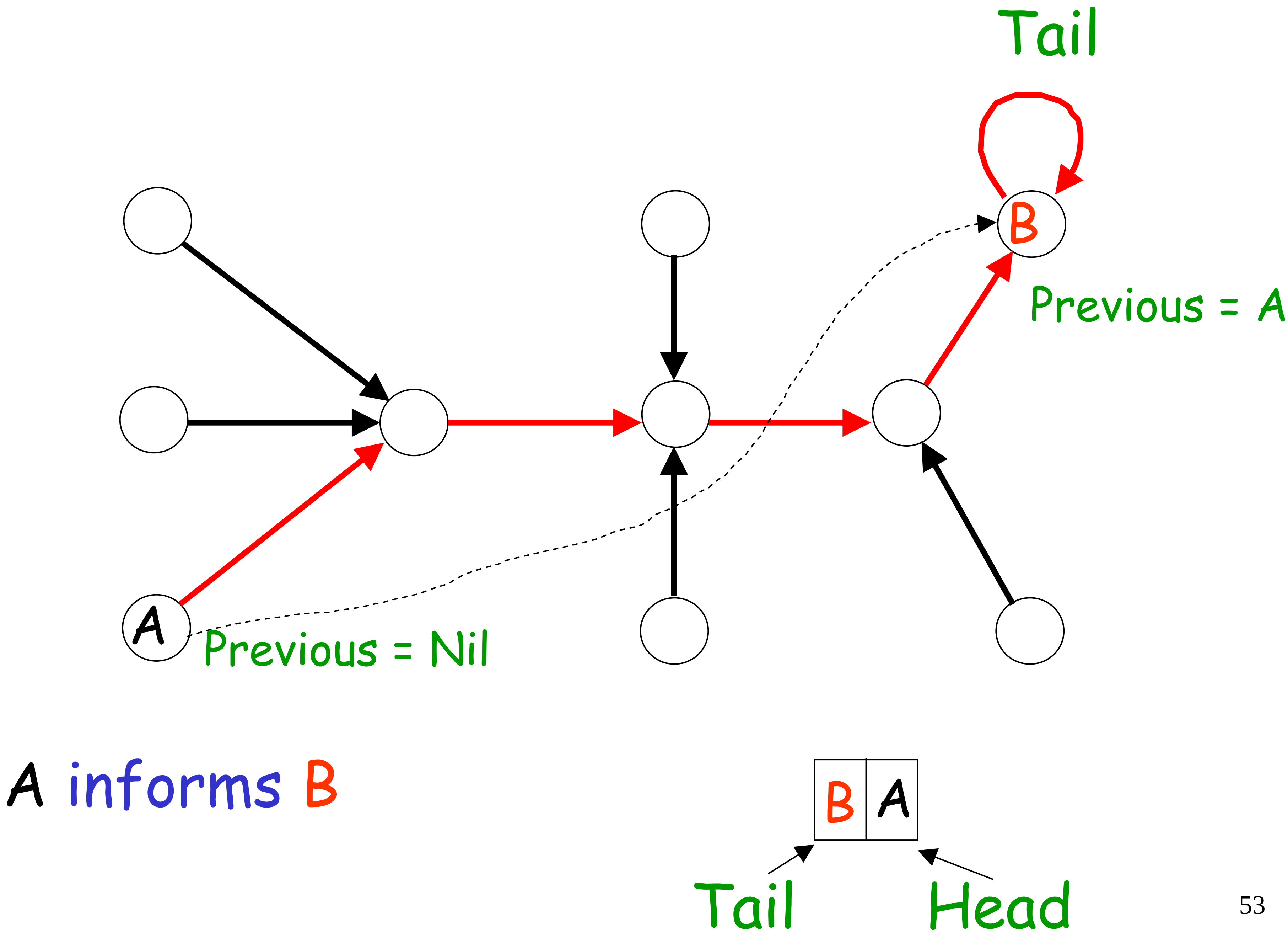


B
Previous = ?
enq(B)
A
Previous = Nil
Tail
A
Tail
Head

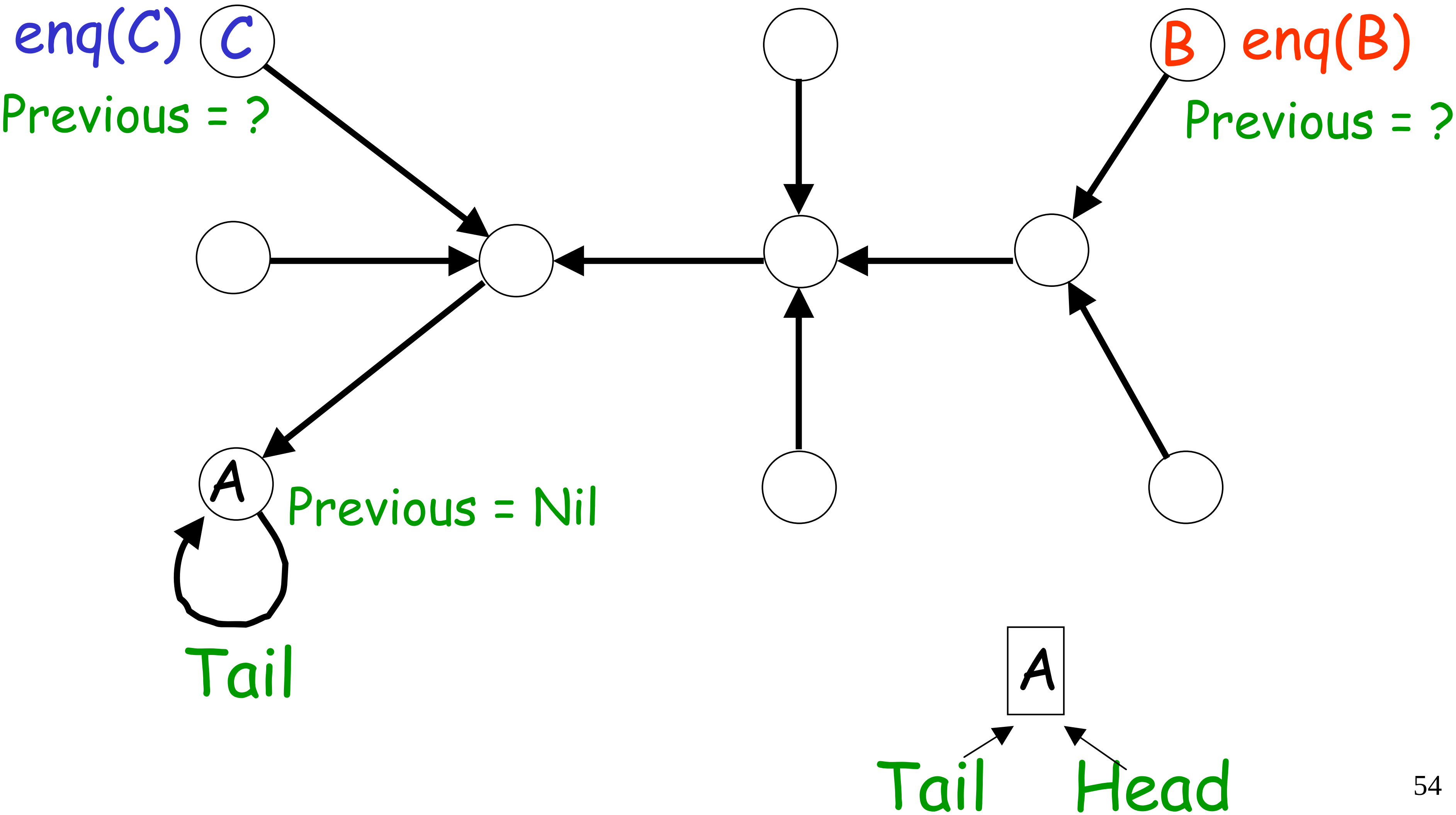

B
Previous = ?
enq(B)
A
Previous = Nil
Tail
A
Tail
Head

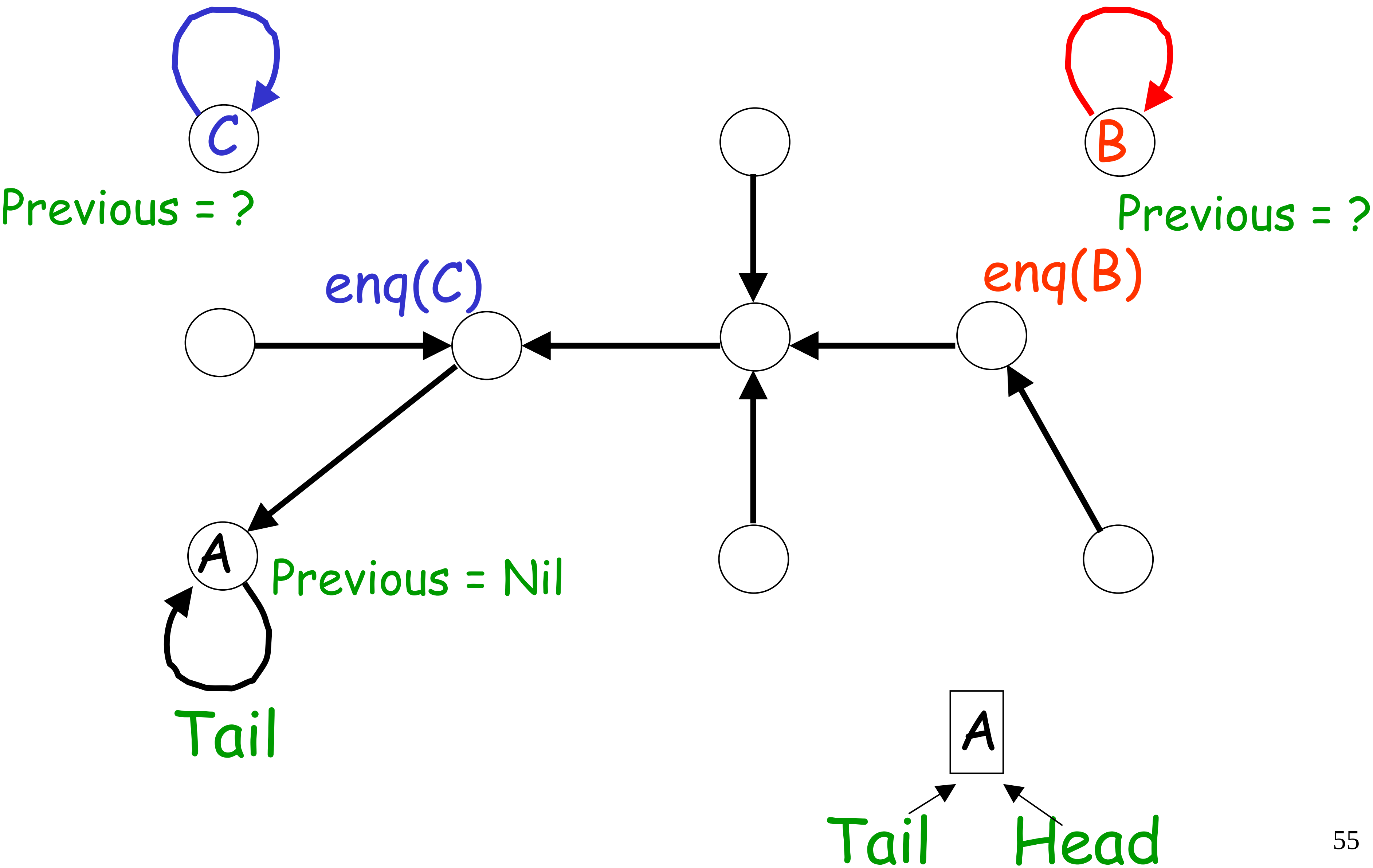

enq(B)
A
Previous = Nil
Tail
B
Previous = ?
A
Tail
Head

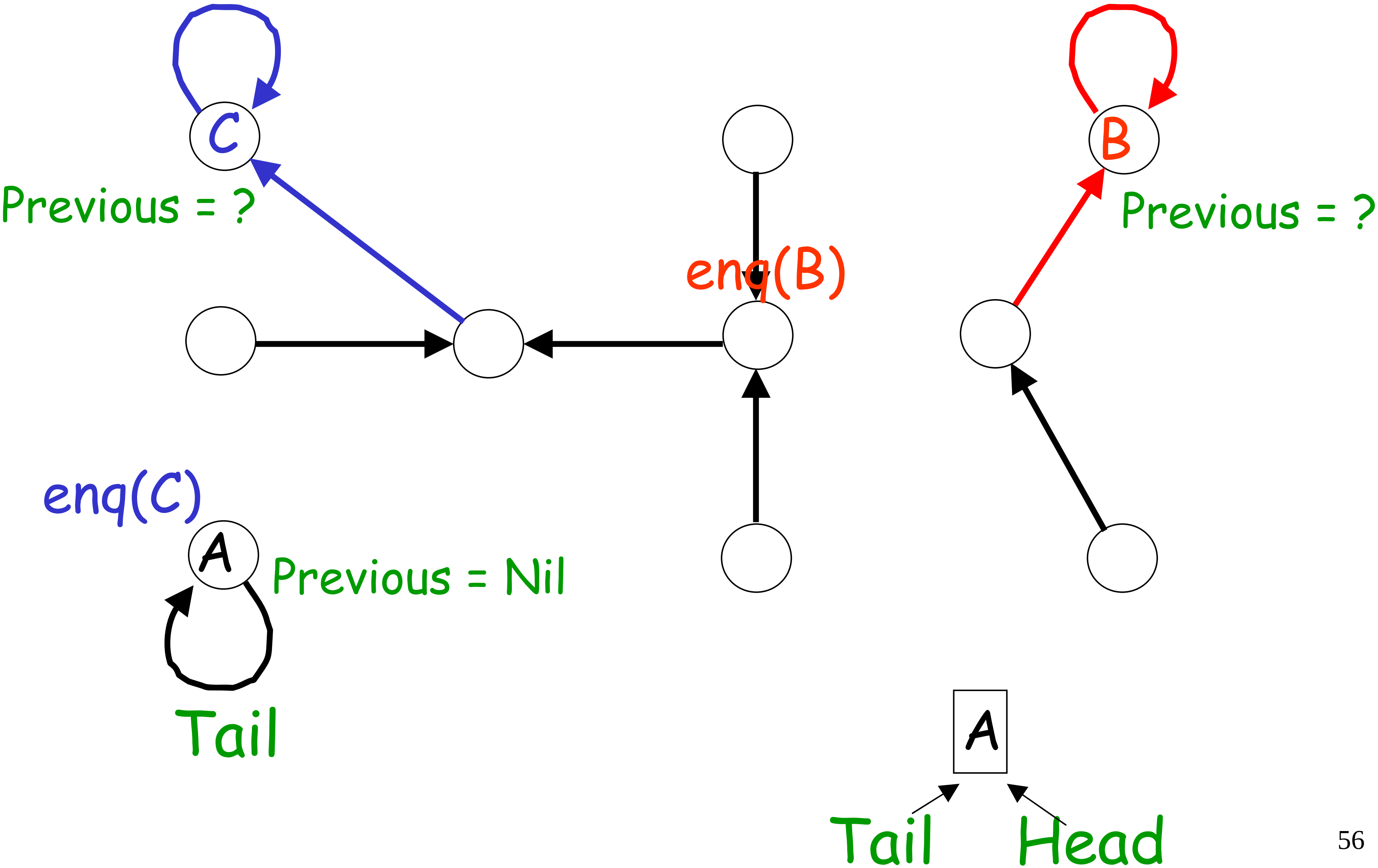

B
Previous = ?
enq(B)
A
Previous = Nil
Tail
A
Tail
Head

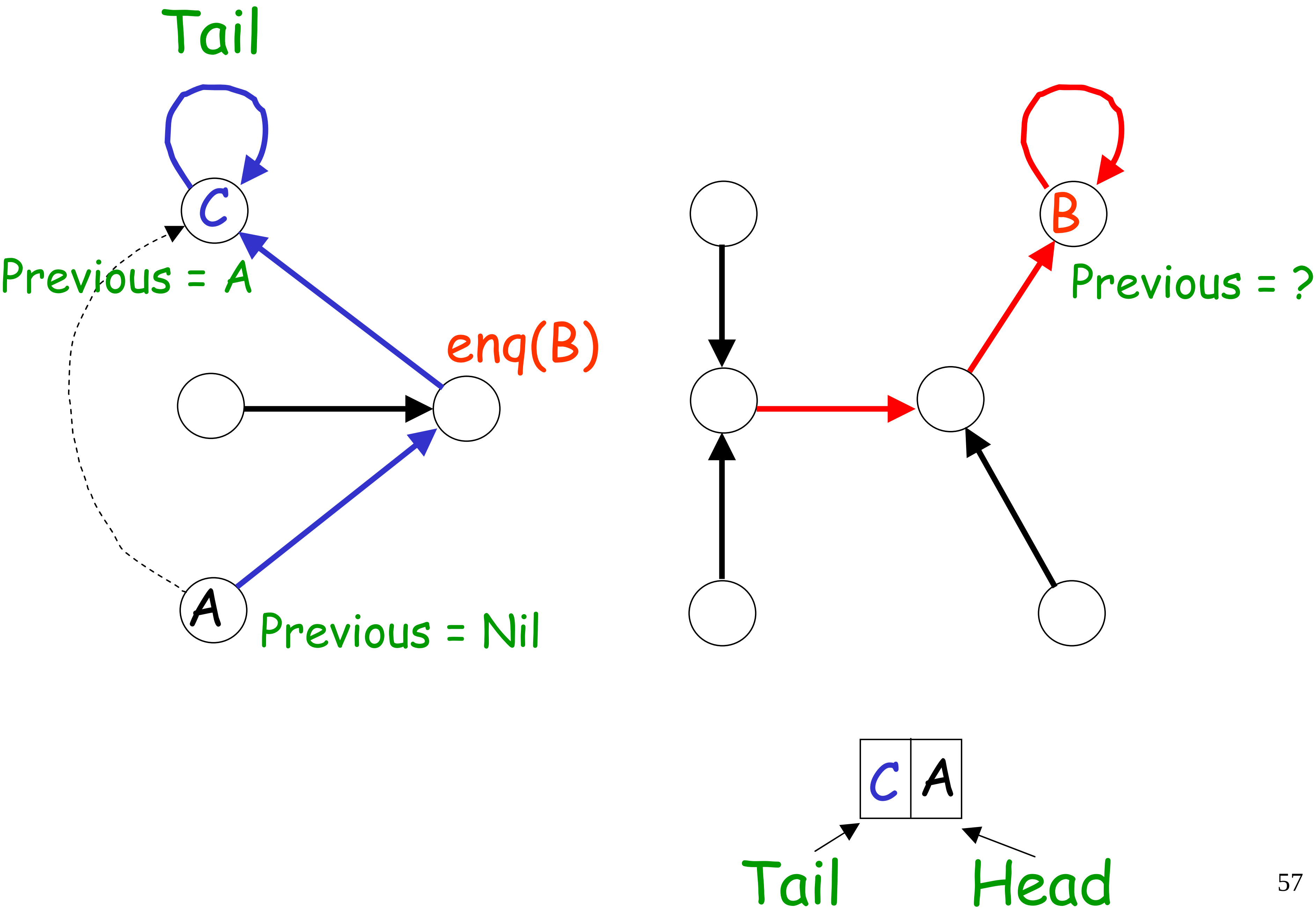

Tail
B
Previous = A
A
Previous = Nil
A informs B
B
A
Tail
Head

Concurrent Enqueue Requests

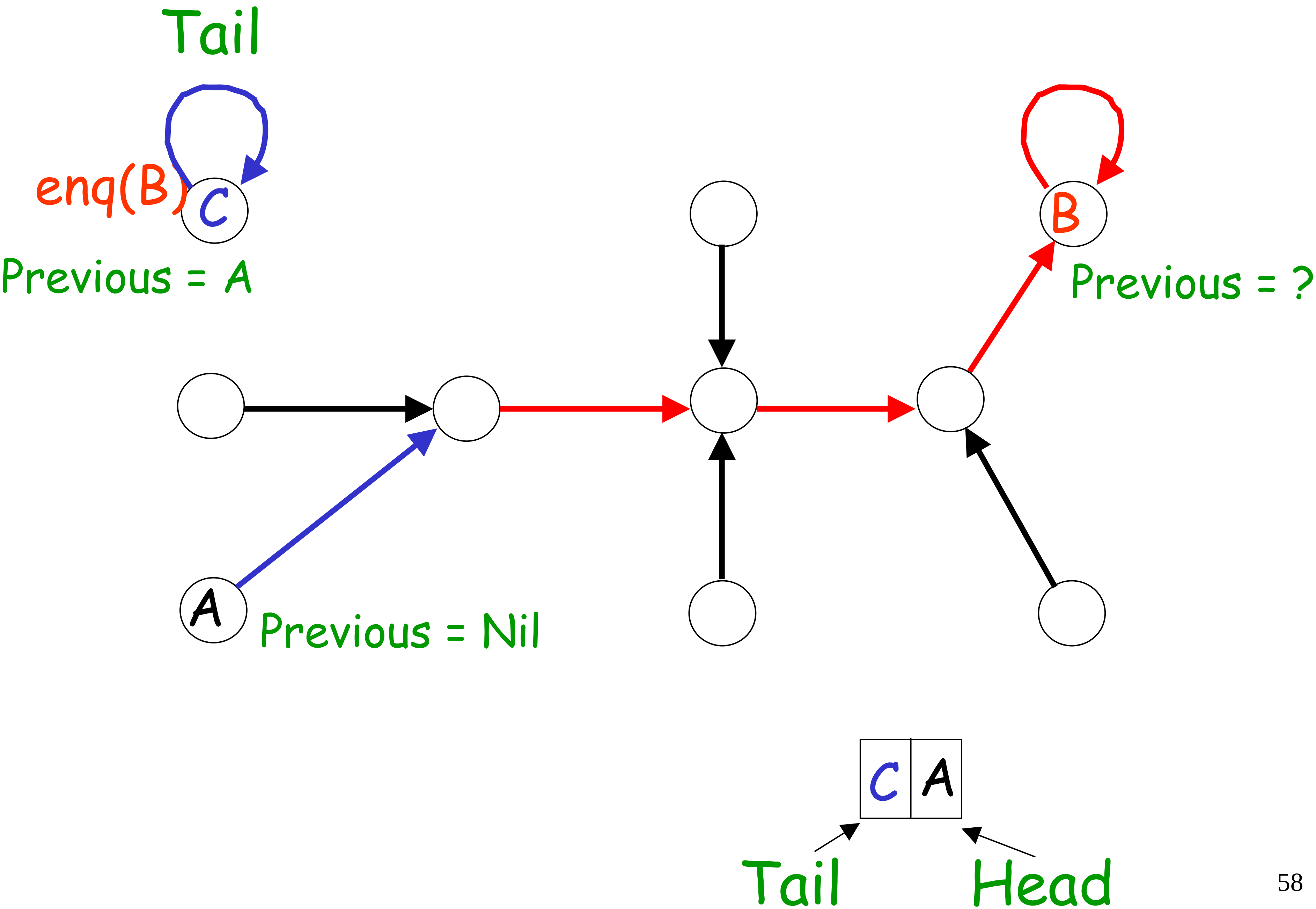



C
Previous = ?
enq(C)
B
Previous = ?
enq(B)
A
Previous = Nil
Tail
A
Tail
Head

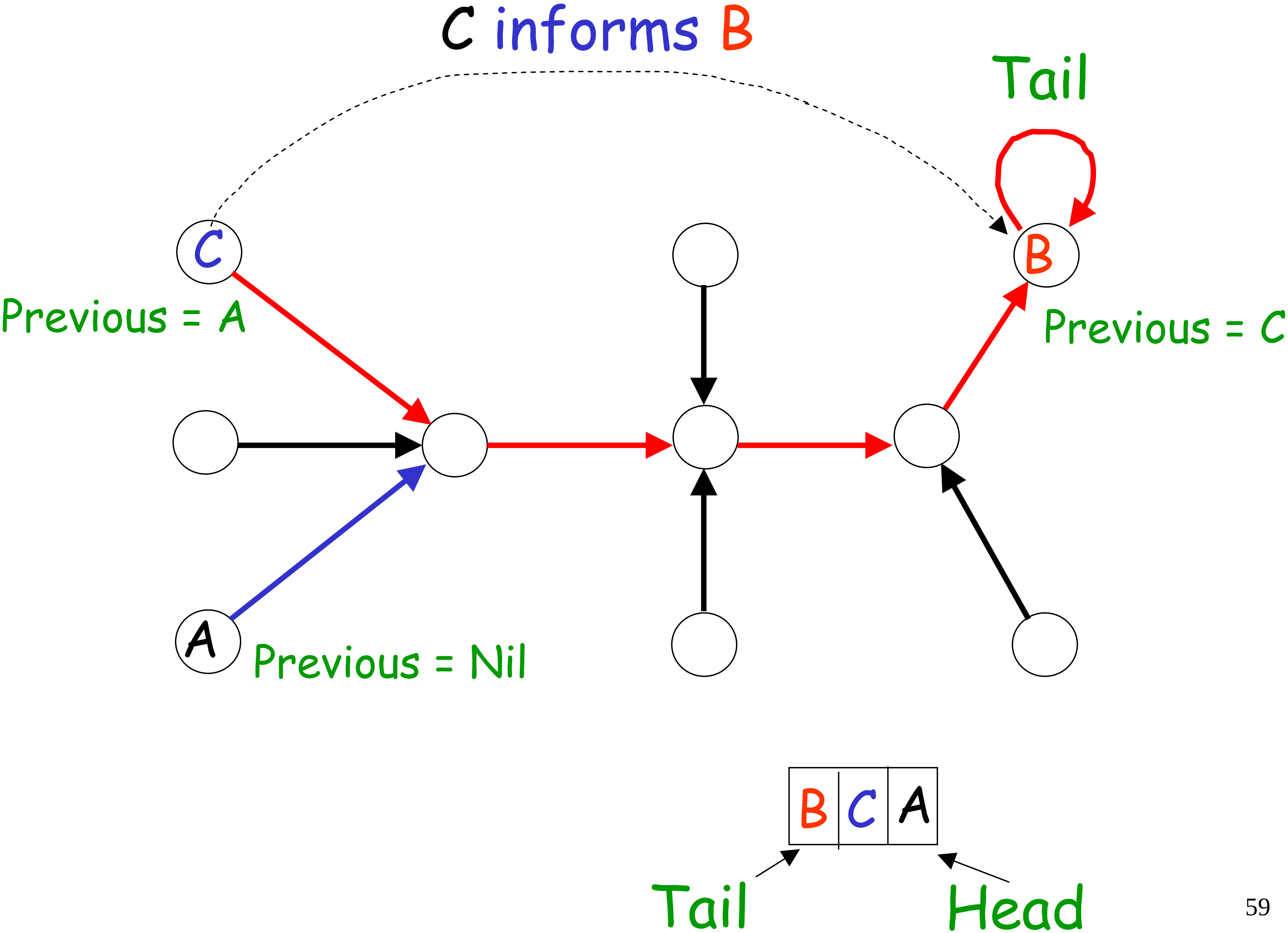

C
Previous = ?
enq(B)
B
Previous = ?
enq(C)
A
Previous = Nil
Tail
A
Tail
Head

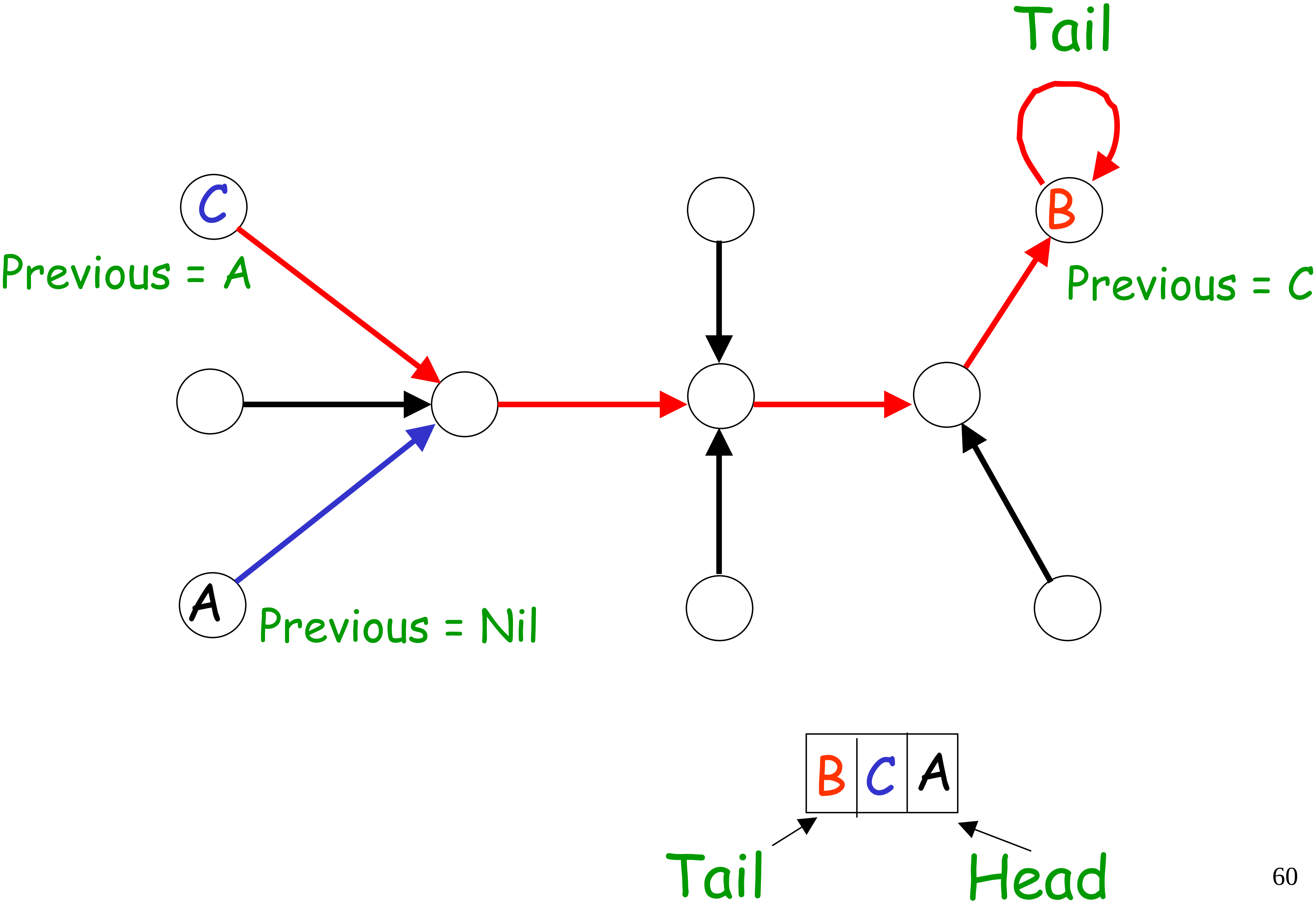


Tail
C
Previous = A
enq(B)
A
Previous = Nil
B
Previous = ?
C
A
Tail
Head

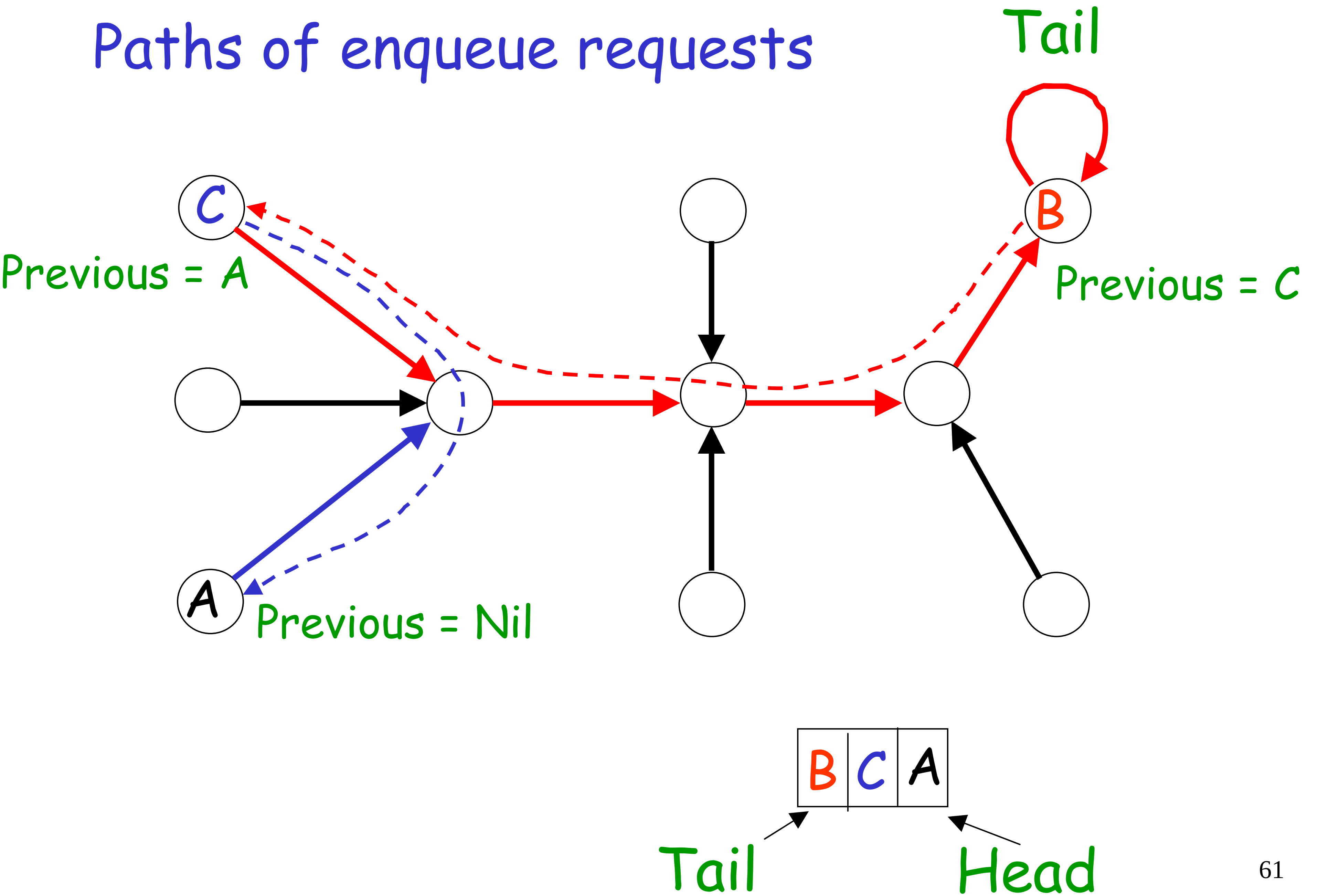


Tail
enq(B)
C
Previous = A
B
Previous = ?
A
Previous = Nil
C
A
Tail
Head

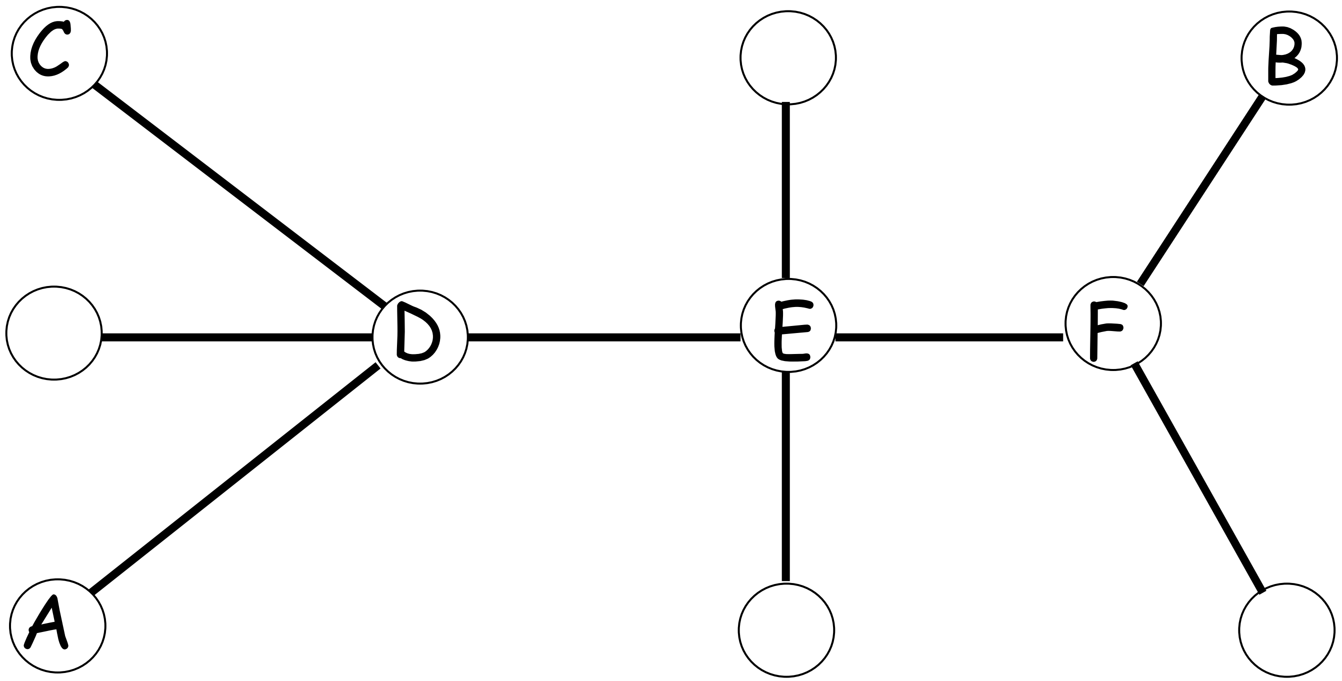

C informs B
Tail
C
B
Previous = A
Previous = C
A
Previous = Nil
B
C
A
Tail
Head


Tail
C
B
Previous = A
Previous = C
A
Previous = Nil
B
C
A
Tail
Head

Paths of enqueue requests

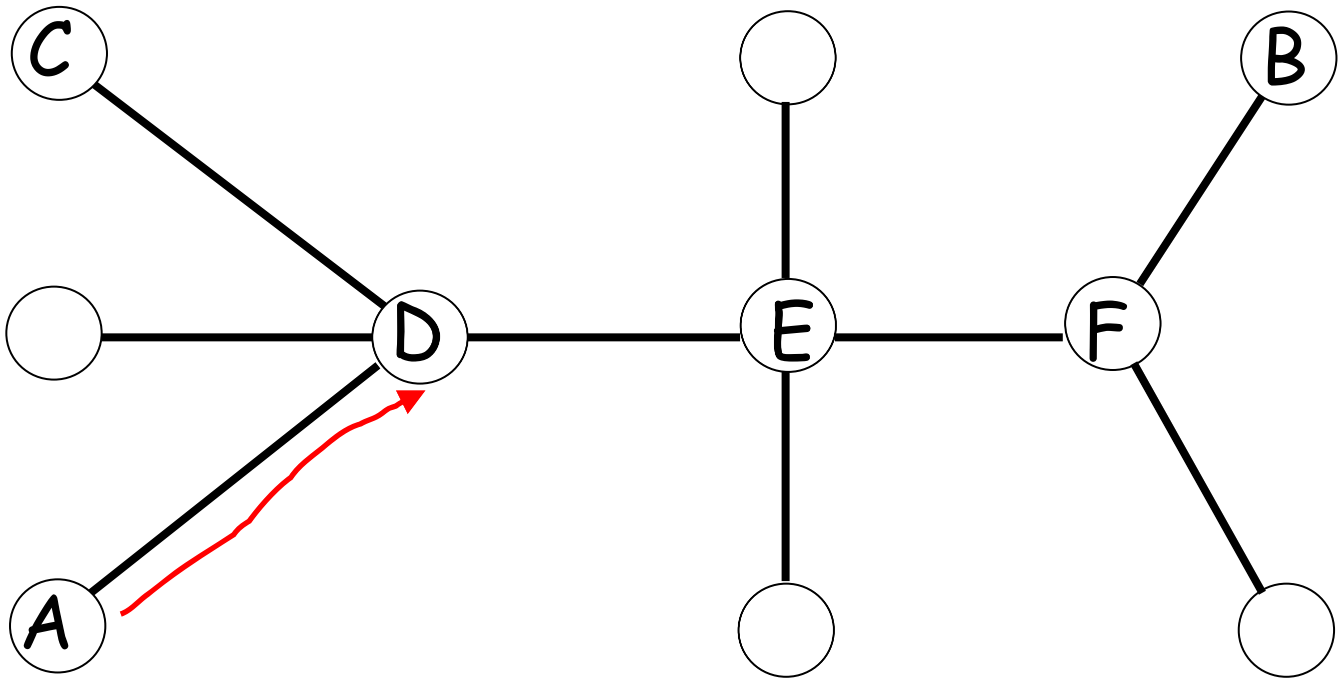


Nearest-Neighbor TSP tour on Spanning tree

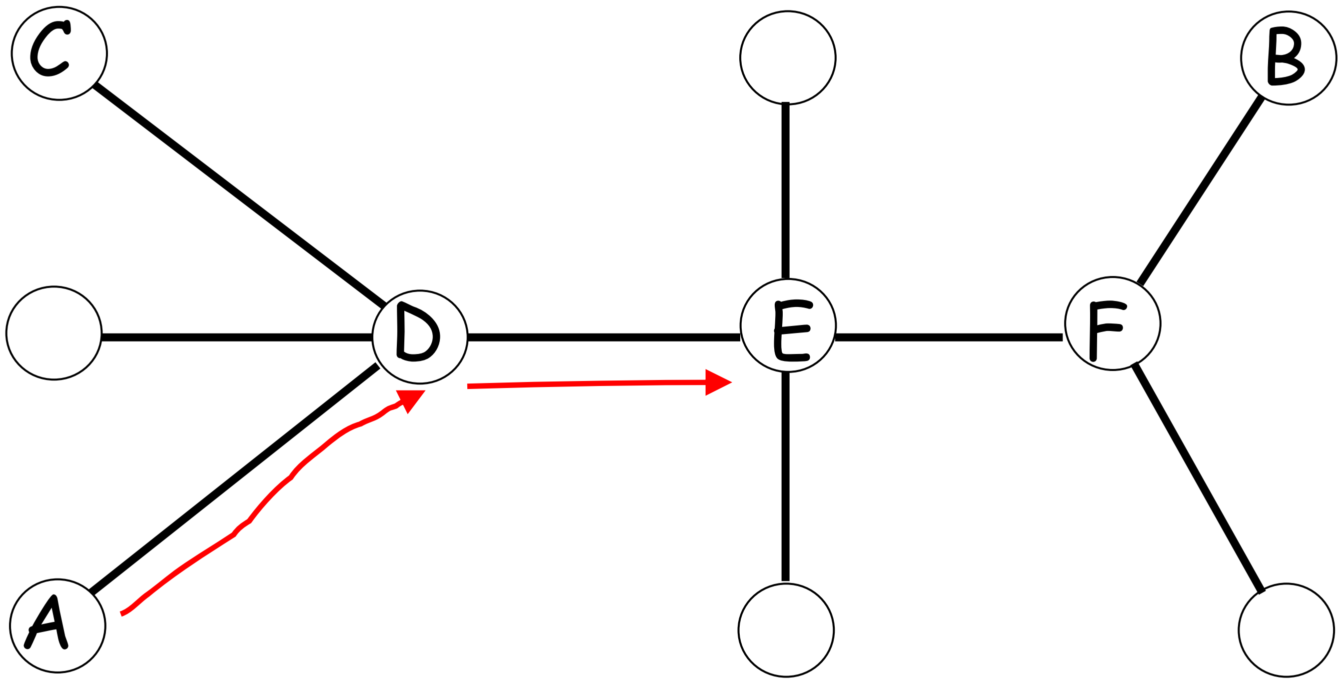


Origin
(first element in queue)

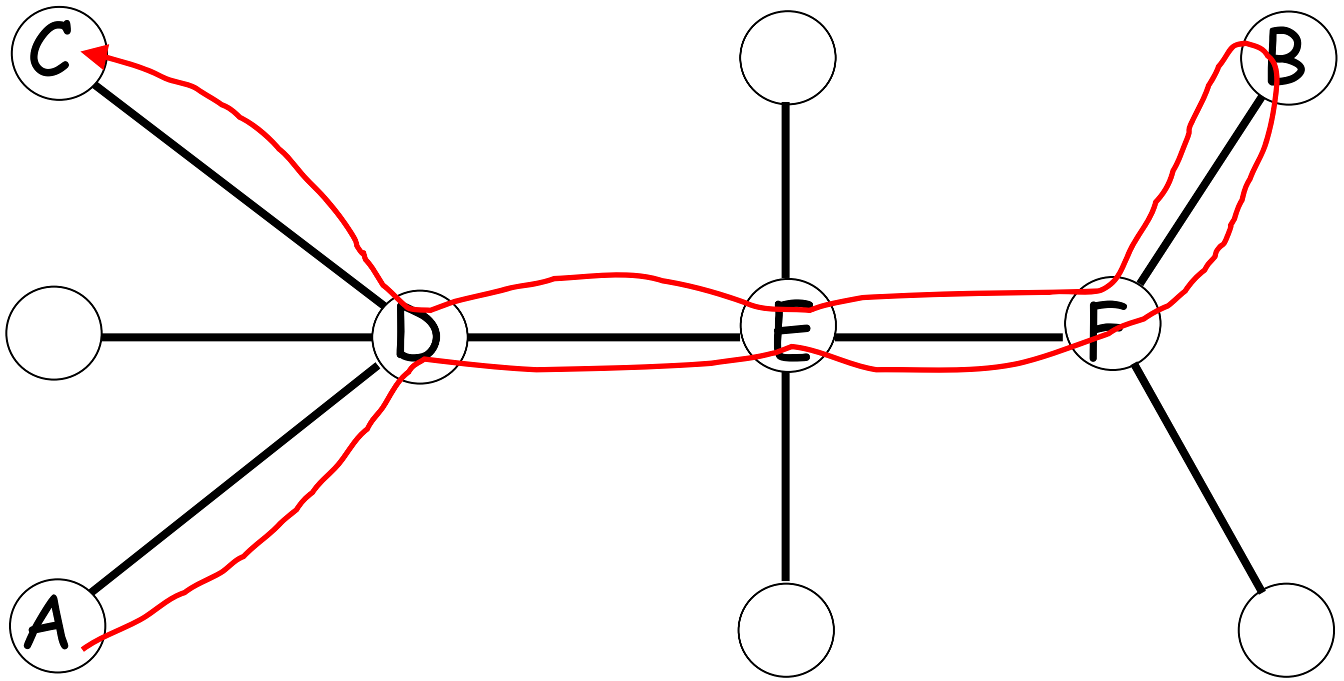
Visit closest unused node in Tree



Visit closest unused node in Tree



Nearest-Neighbor TSP tour



[Herlihy, Tirthapura, Wattenhofer PODC'01]

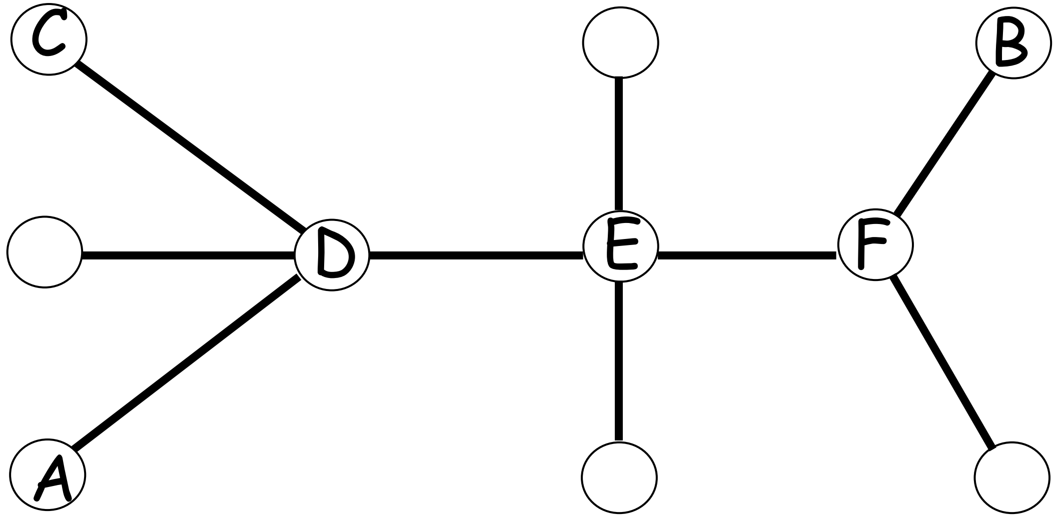
For spanning tree of constant degree:

Queuing Cost $\leq$ 2 x Nearest-Neighbor TSP length

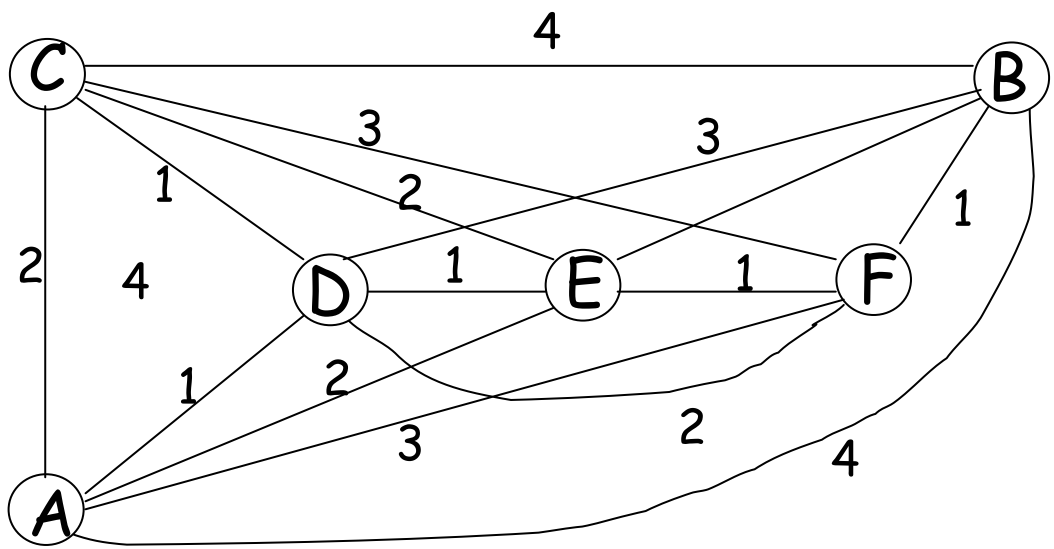
[Rosenkratz, Stearns, Lewis SICOMP1977]

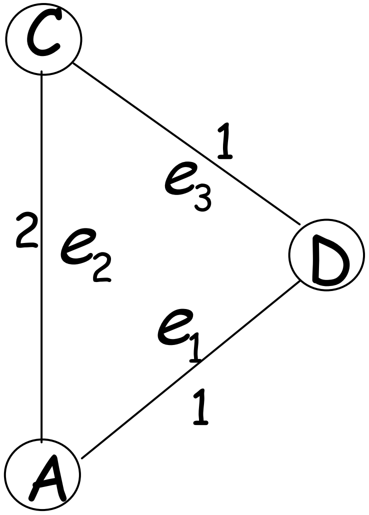
If a weighted graph satisfies triangular inequality:

$$\text{Nearest-Neighbor TSP length} \leq \text{Optimal TSP length} \times \log n$$



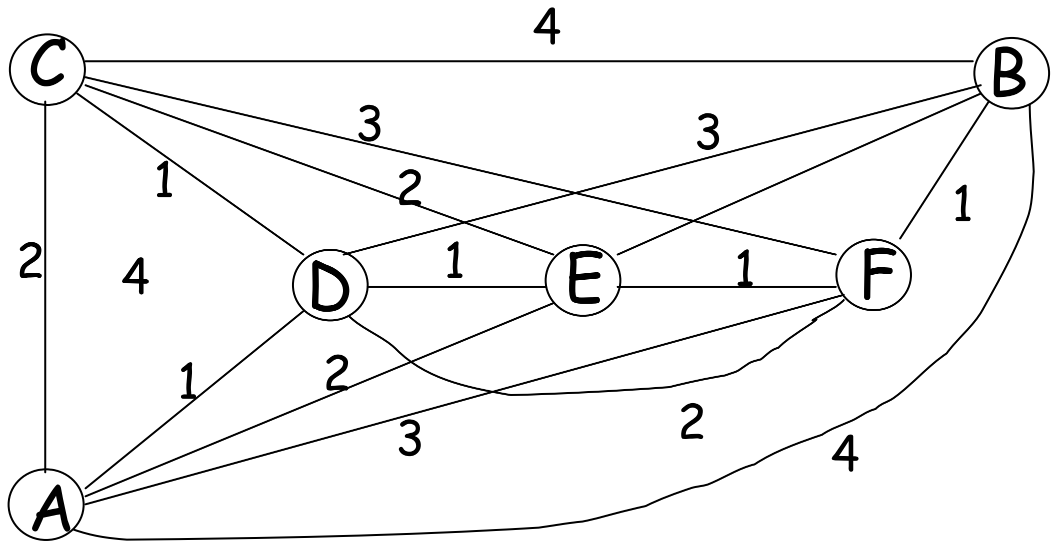
weighted graph of distances

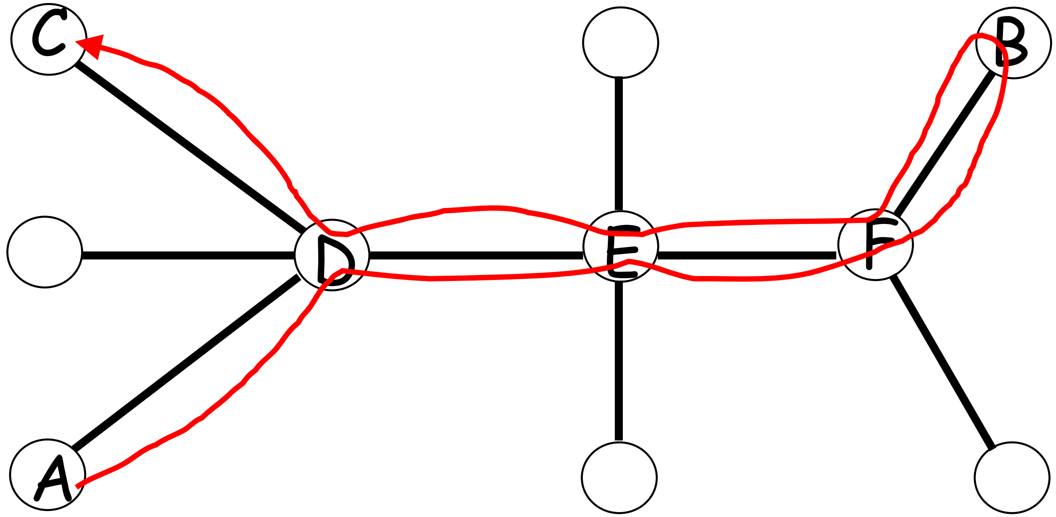




$$w(e_1) \leq w(e_2) + w(e_3)$$

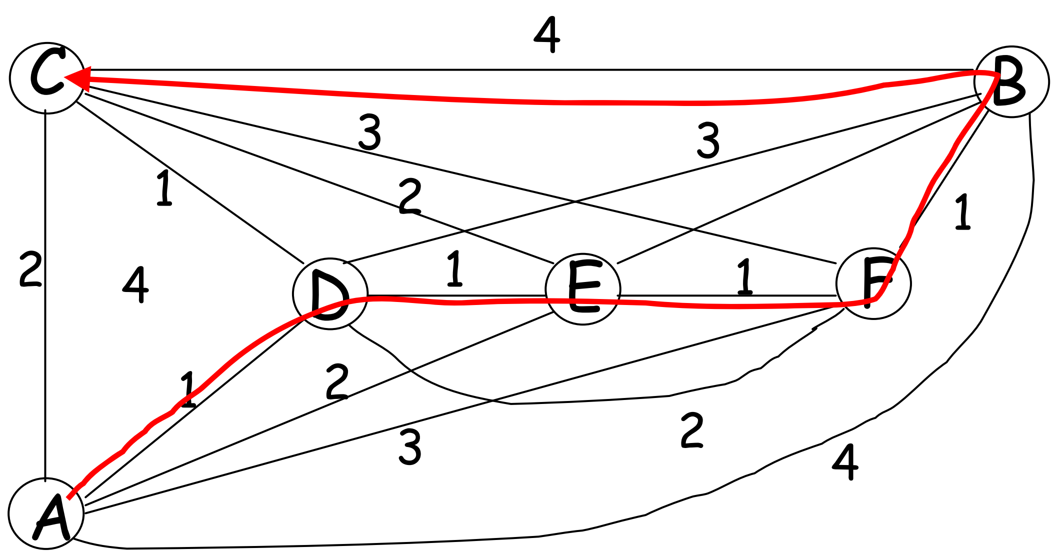
Satisfies triangular inequality

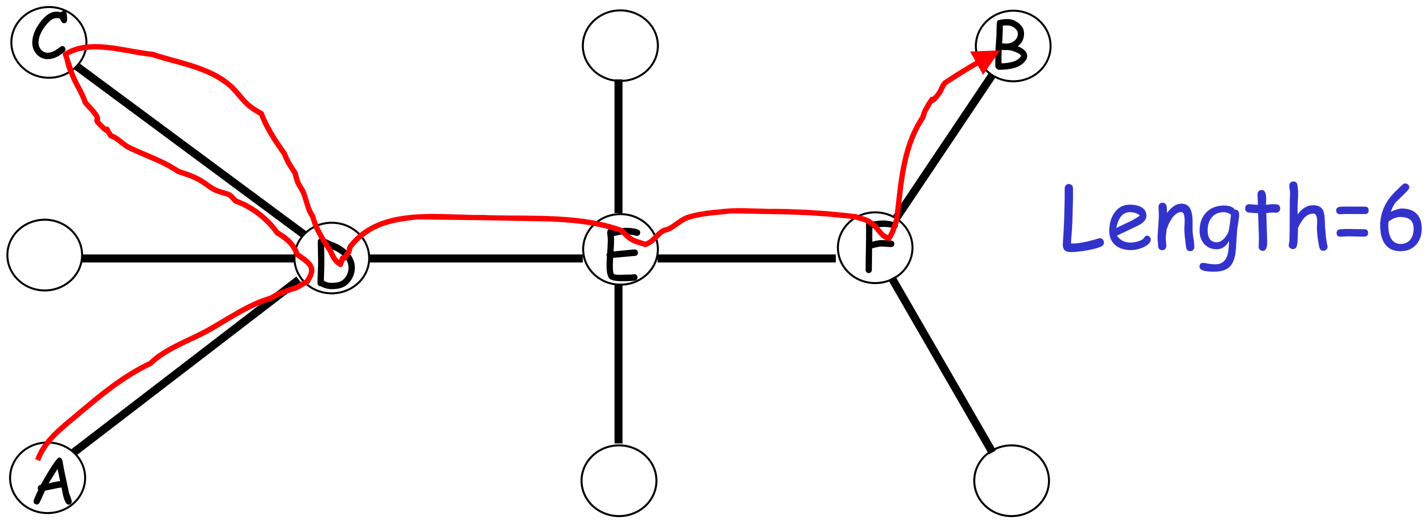




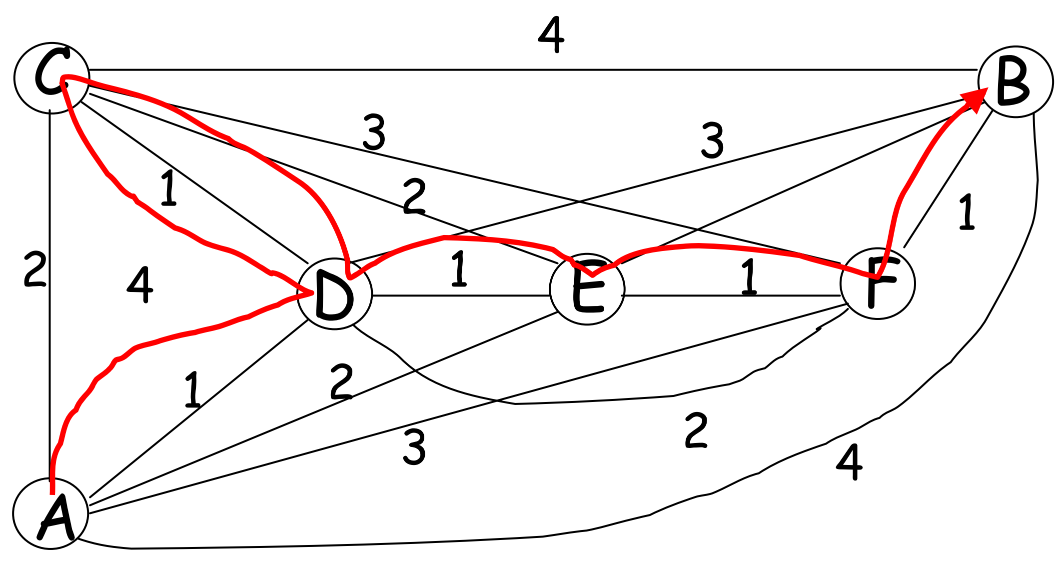
Length=8

Nearest Neighbor TSP tour



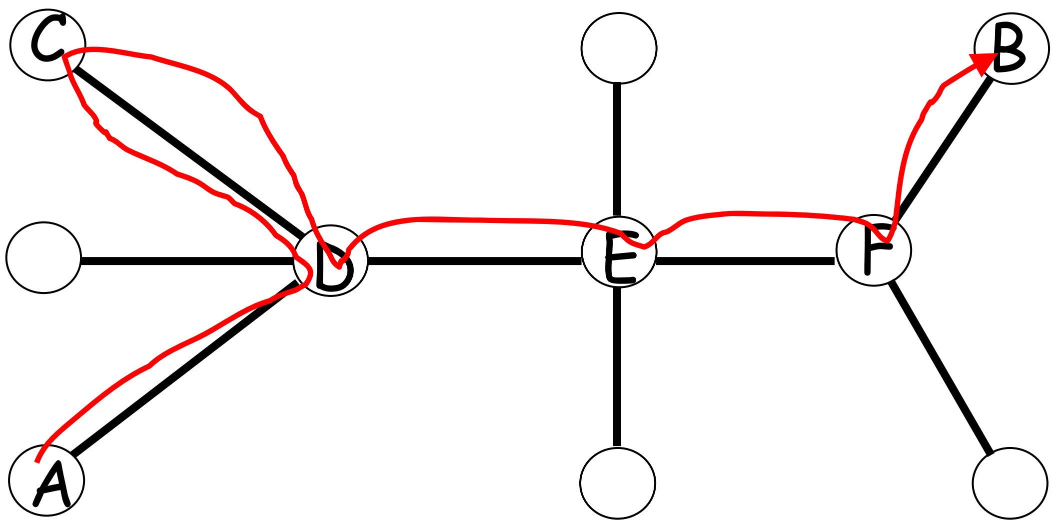


Optimal TSP tour



It can be shown that:

Optimal TSP length $\leq 2n$ (Nodes in graph)



Since every edge is visited twice

Therefore,
for constant degree spanning tree:

Queuing Cost = O(Nearest-Neighbor TSP)

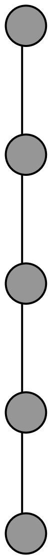
= O(Optimal TSP x $\log n$)

= $O(n \log n)$

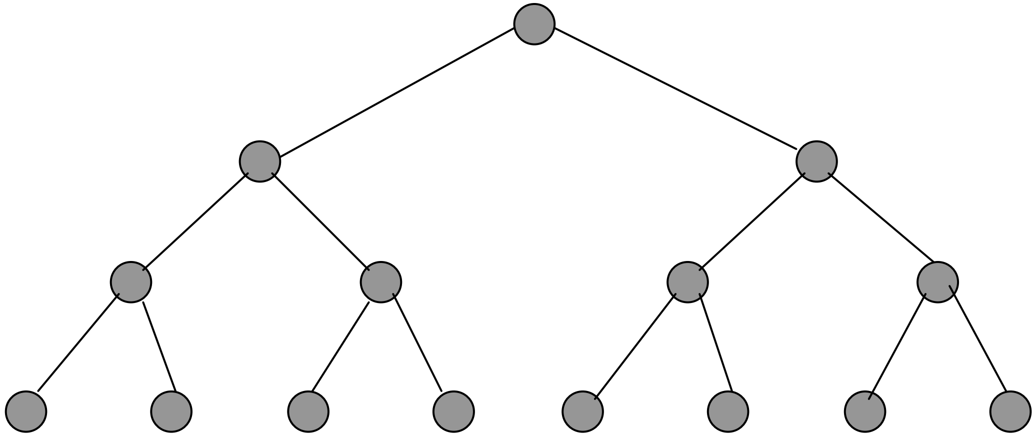
For special cases we can do better:

Spanning Tree is

List

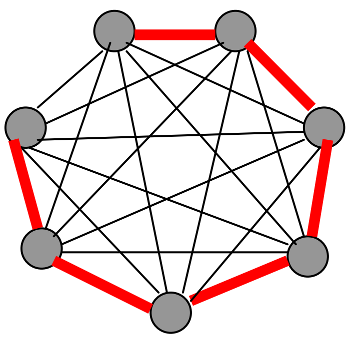


balanced binary tree

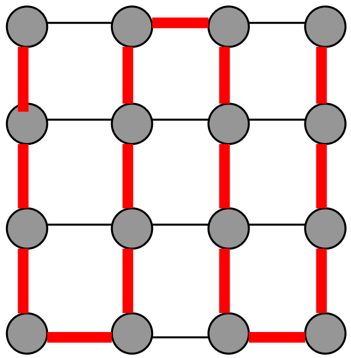


Queuing Cost = $O(n)$

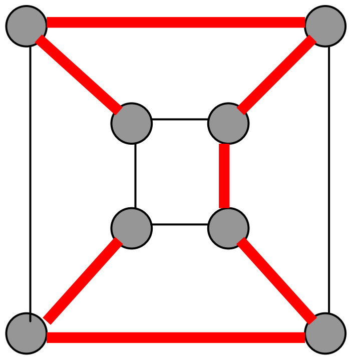
Graphs with Hamiltonian path, have spanning trees which are lists



Complete graph



Mesh



Hypercube

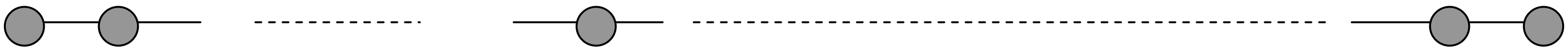
Queuing Cost = $O(n)$

Counting Cost = $\Omega(n \log^* n)$

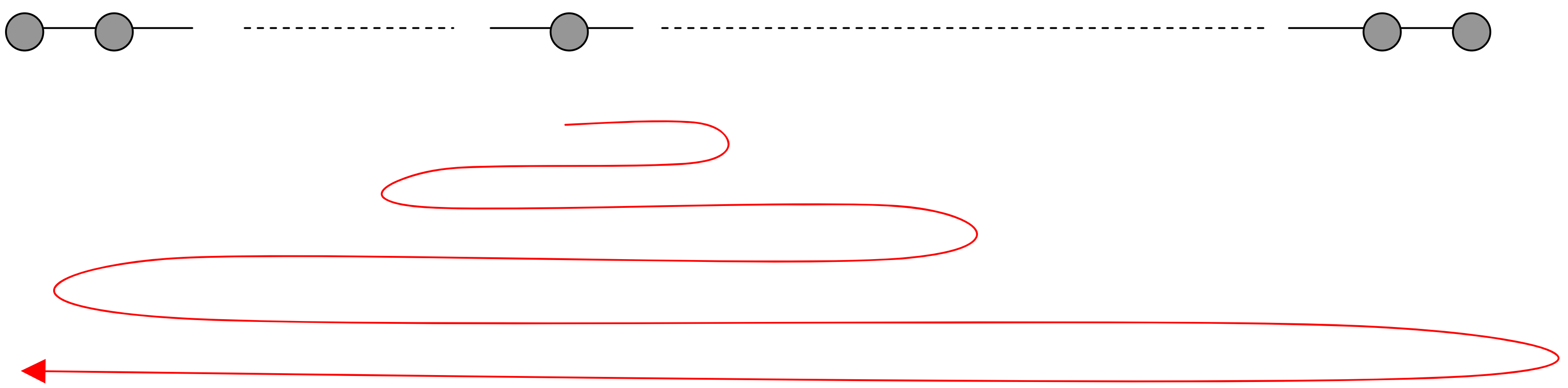
Theorem:

If the spanning tree is a list, then

Queuing Cost = $O(n)$



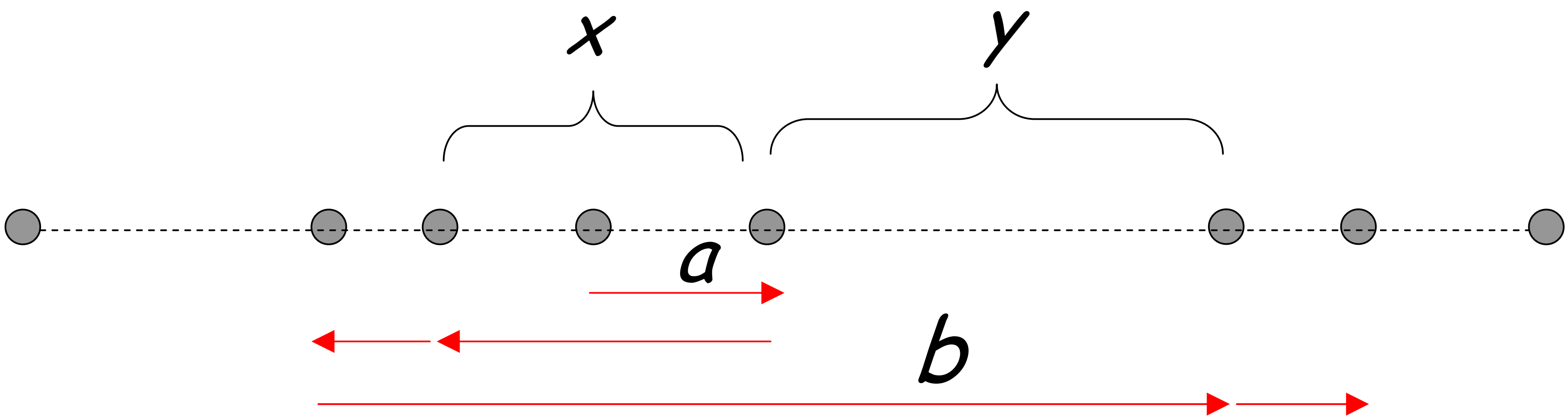
Proof:



Nearest Neighbor TSP

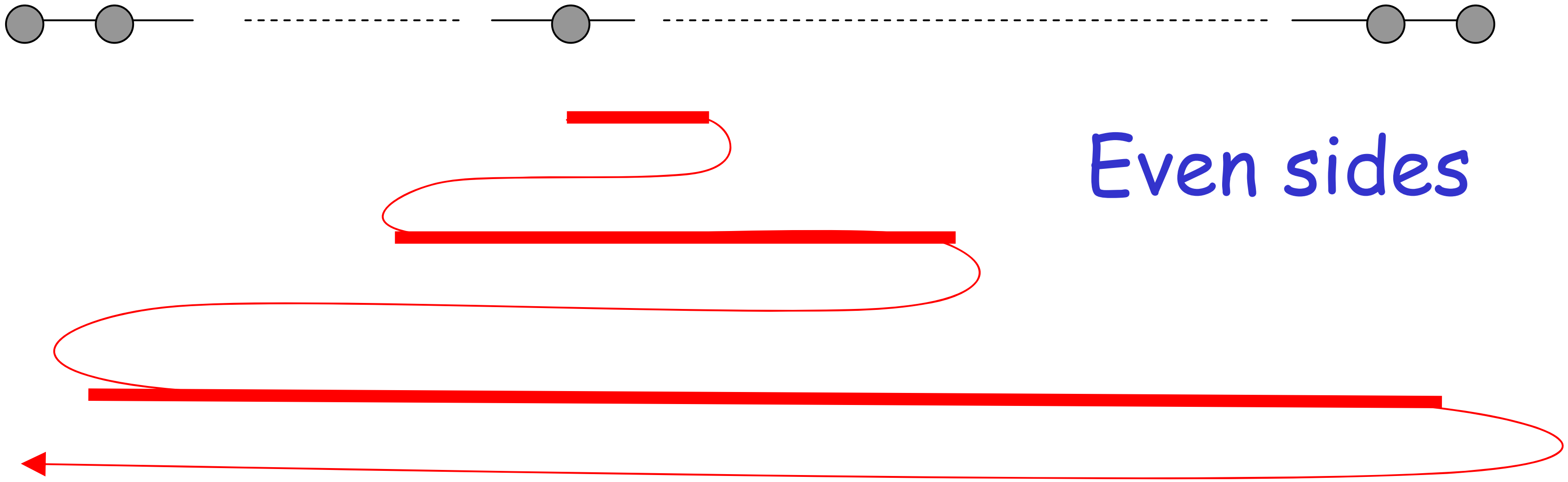
Queuing Cost = O(Nearest-Neighbor TSP)

$$x < y$$



$$b > 2a$$

n nodes

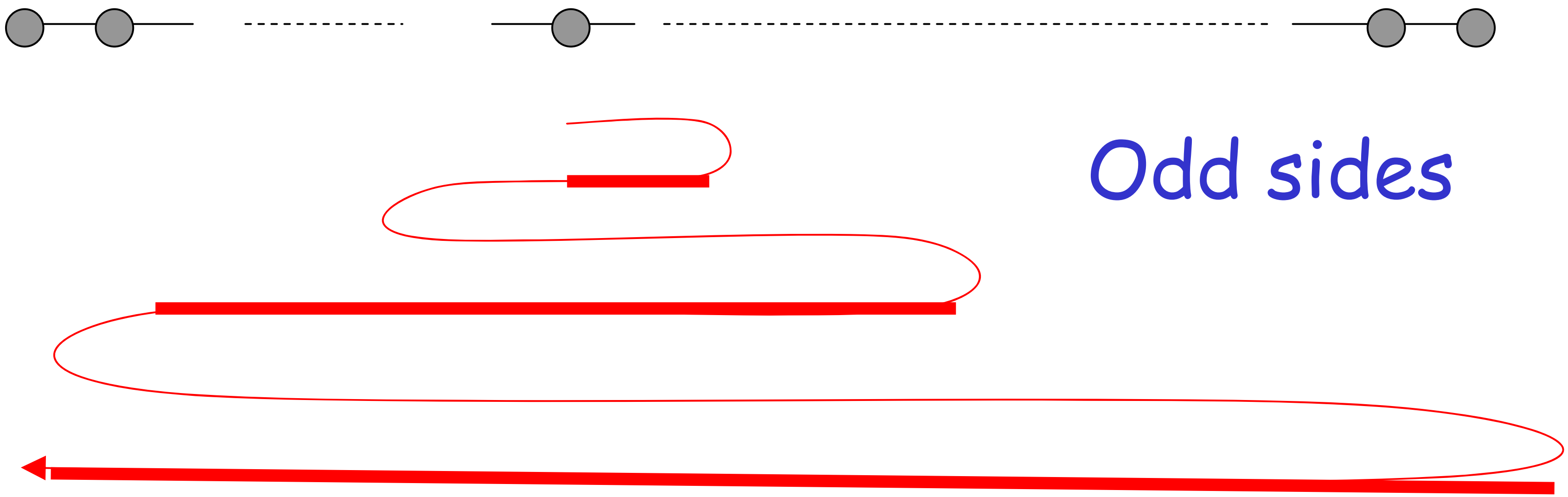


Even sides

Length doubles

Total length $\leq 2n$

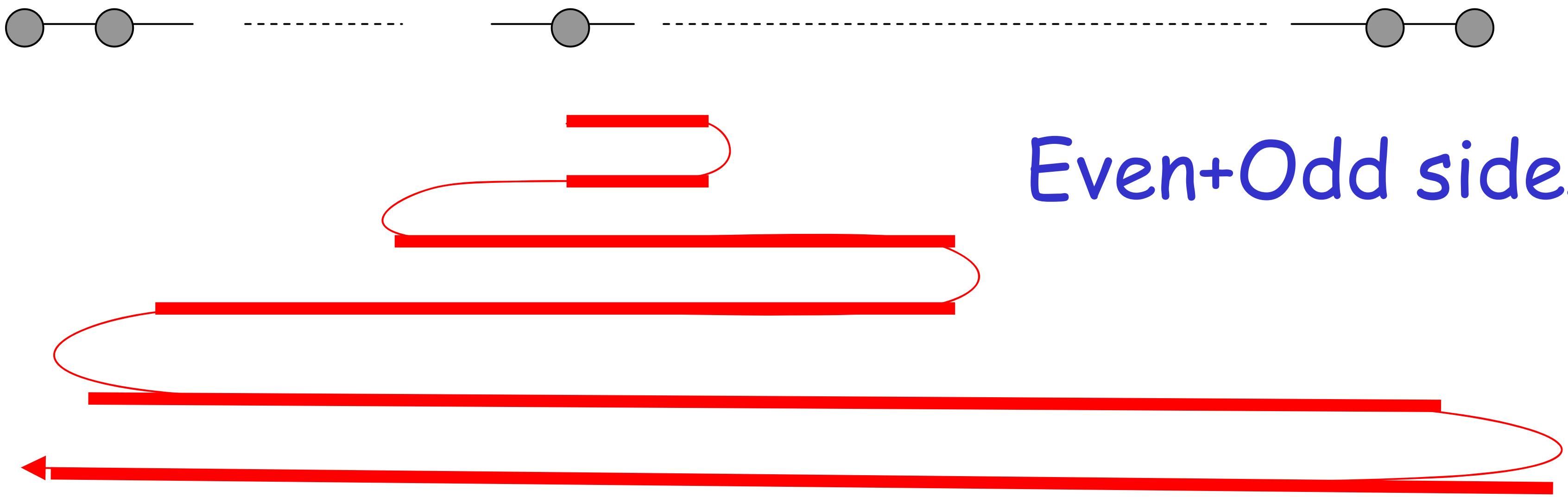
n nodes



Length doubles

Total length $\leq 2n$

n nodes



Even+Odd sides

Total length $\leq 2n + 2n = 4n$

End of proof