

Lecture: 14

Distributed Minimum Spanning Tree



Rajiv Misra

Dept. of Computer Science &
Engineering

Indian Institute of Technology Patna

rajivm@iitp.ac.in

Preface

Recap of Previous Lecture:

- In the previous lecture, we have discussed the basic concepts of '**Distributed Shared Memory**', its consistency models and algorithms for shared memory mutual exclusion.

Content of this Lecture:

- In this lecture, we will discuss about '**Distributed Minimum Spanning Tree**' and
- Also discuss the well known **GHS algorithm** from the following reference:

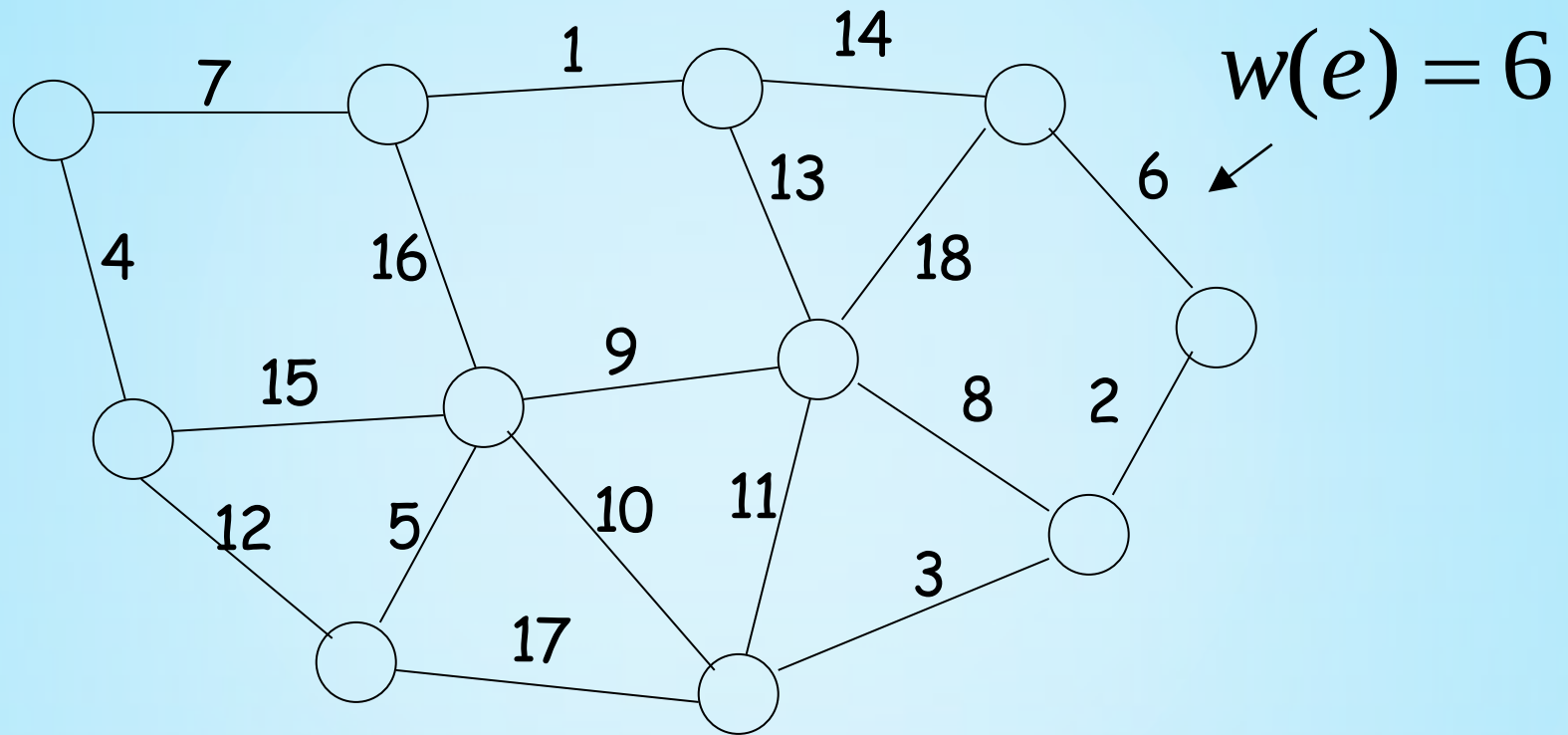
"R.G. Gallager, P.A. Humblet, P.M. Spira, A distributed algorithm for minimum-weight spanning trees, ACM Trans. Program. Lang. Syst. 5 (1983) 66–77."

Introduction

- The **'Distributed Minimum Spanning Tree (MST)'** problem involves the construction of a minimum spanning tree by a distributed algorithm, in a network where nodes communicate by message passing.
- One important application of this problem is to **find a tree that can be used for broadcasting.**
- In particular, if the cost for a message to pass through an edge in a graph is significant, a MST can minimize the total cost for a source process to communicate with all the other processes in the network.

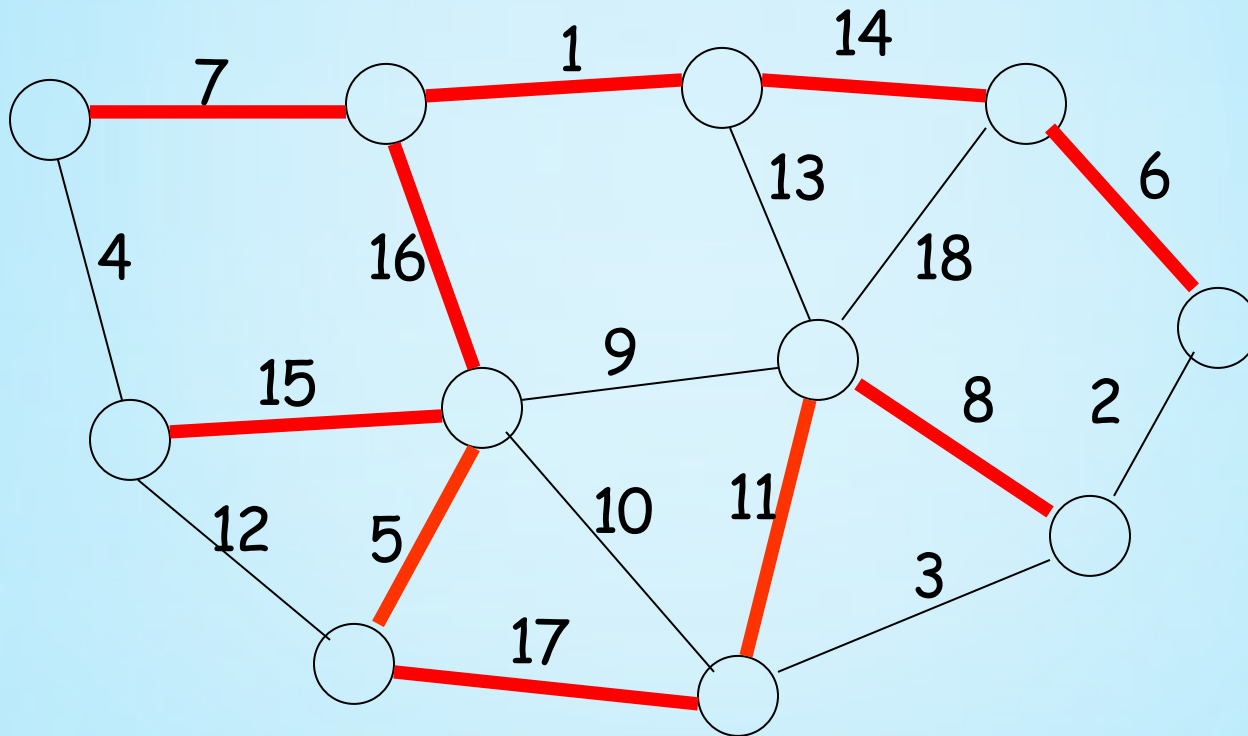
Preliminaries

Weighted Graph $G=(V,E)$, $|V|=n$, $|E|=m$



Spanning tree

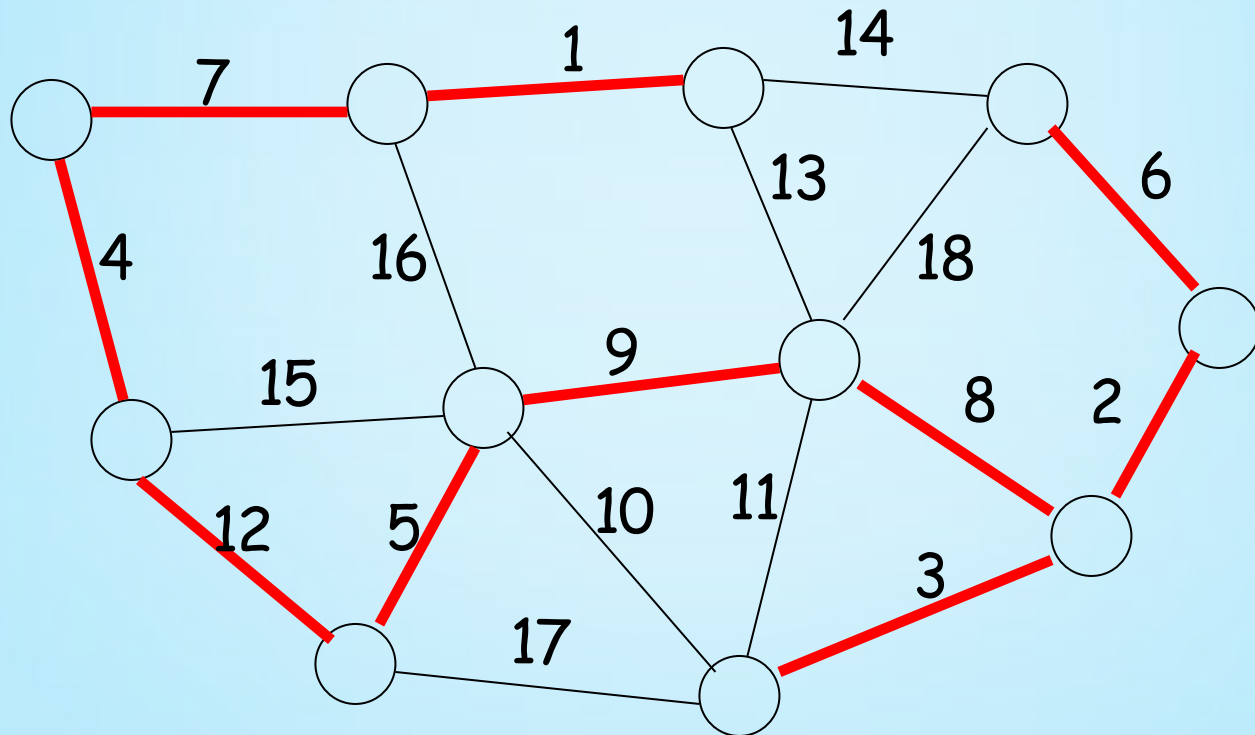
Any tree $T=(V,E')$ (connected acyclic graph)
spanning all the nodes of G



Minimum-weight spanning tree

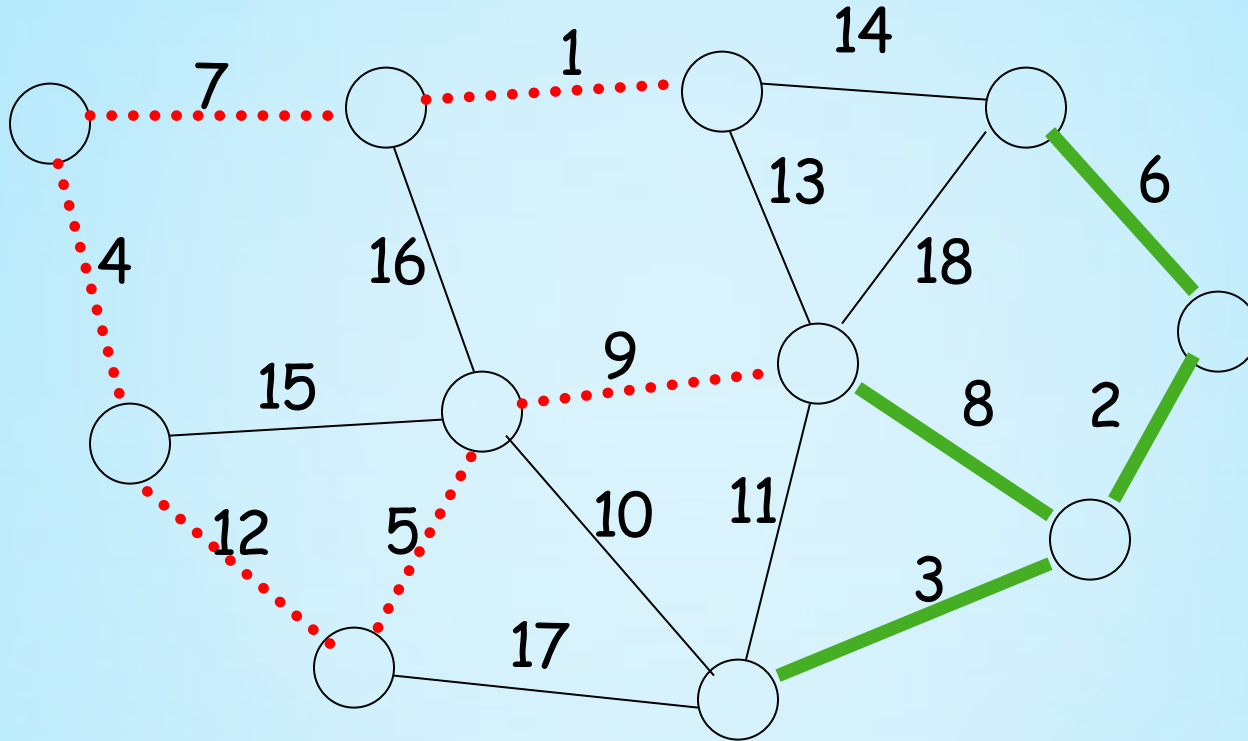
A spanning tree subject to the sum of its weights is minimized:

For MST : $\mathcal{T}$ $w(\mathcal{T}) = \sum_{e \in \mathcal{T}} w(e)$ is minimized



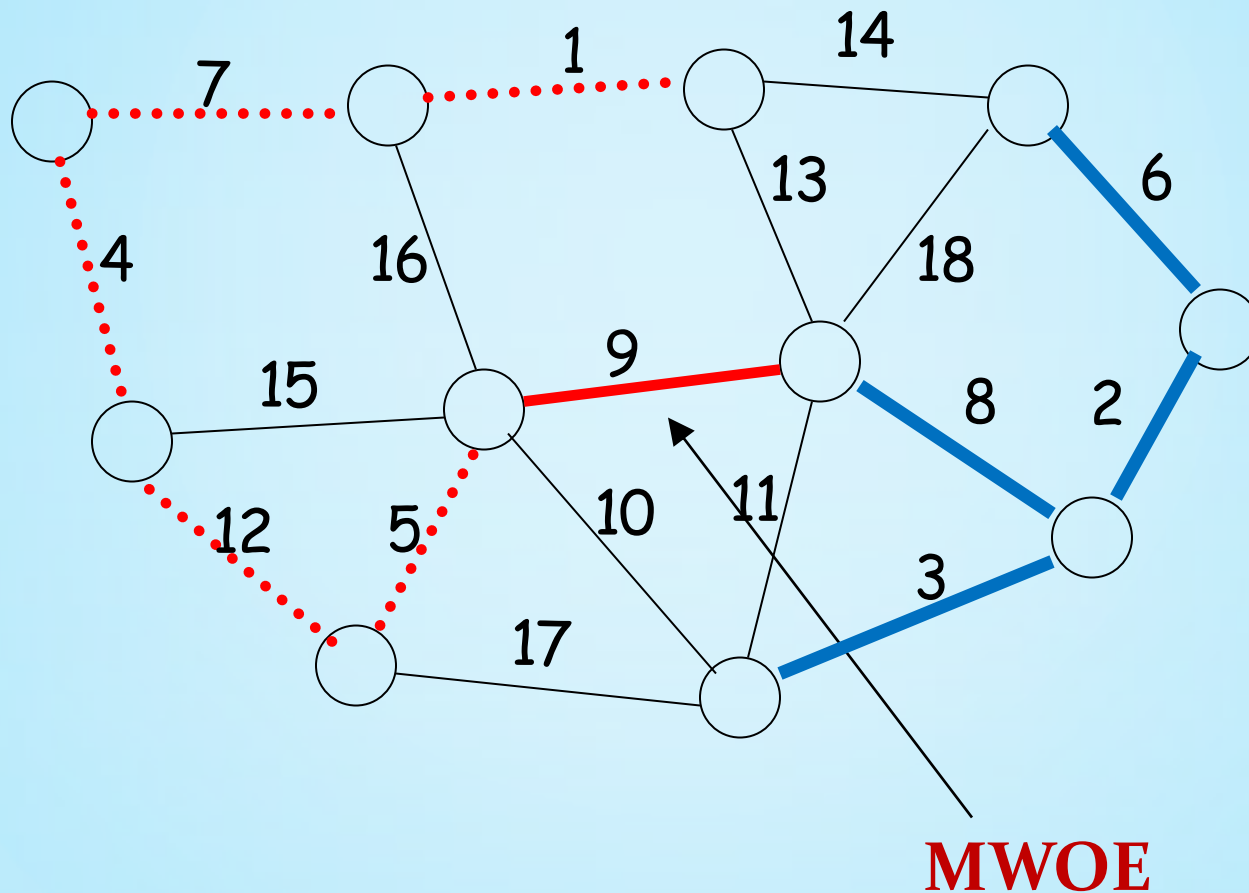
Spanning tree fragment

Any (connected) sub-tree of a MST



Minimum weight outgoing edge (MWOE)

An edge adjacent to the fragment with smallest weight and that does not create a cycle

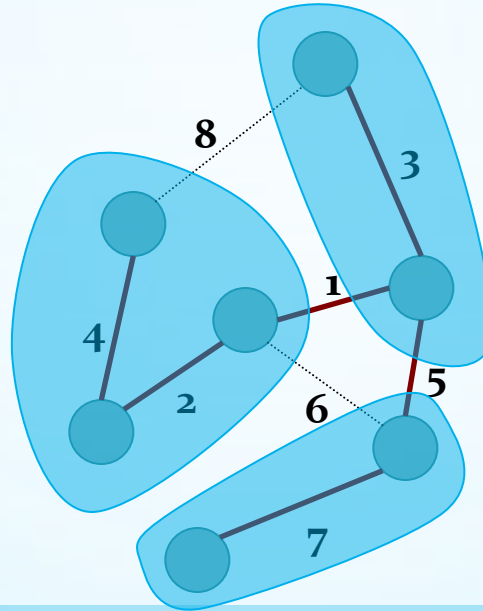


MST Fragment

- Definition – MST *fragment*

- A connected sub-tree of MST

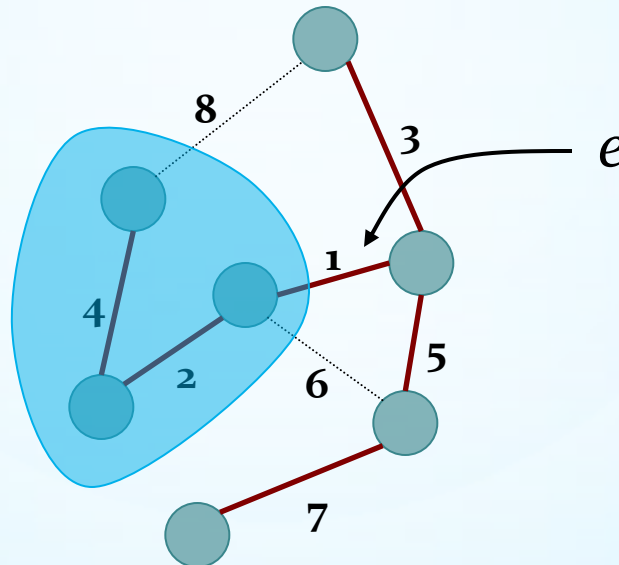
Example of possible fragments:



MST Properties

- MST Property 1

Given a fragment of an MST, let e be a minimum-weight outgoing edge of the fragment. Then joining e and its adjacent non-fragment node to the fragment yields another fragment of an MST.

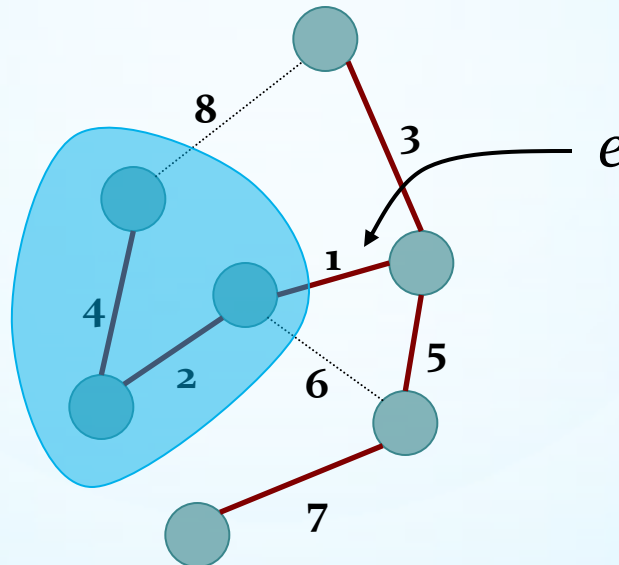


MST Properties

- MST Property 1

Given a fragment of an MST, let e be a minimum-weight outgoing edge of the fragment. Then joining e and its adjacent non-fragment node to the fragment yields another fragment of an MST.

- *Proof:*



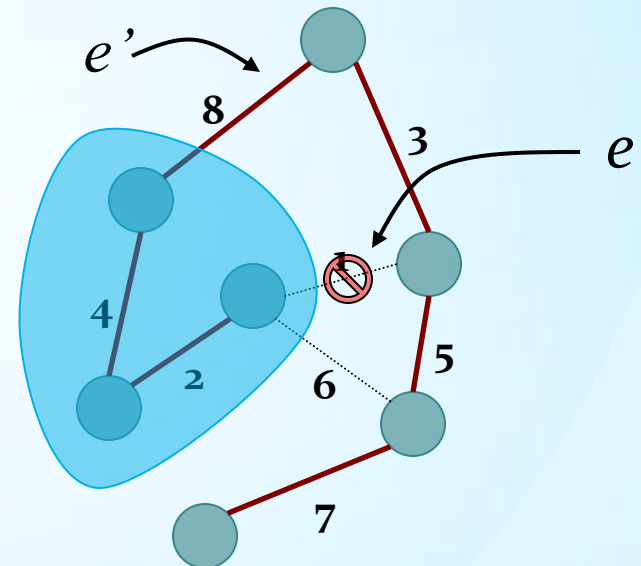
MST Properties

- MST Property 1

Given a fragment of an MST, let e be a minimum-weight outgoing edge of the fragment. Then joining e and its adjacent non-fragment node to the fragment yields another fragment of an MST.

- *Proof:*

Suppose e is not in MST, but some e' instead.



MST Properties

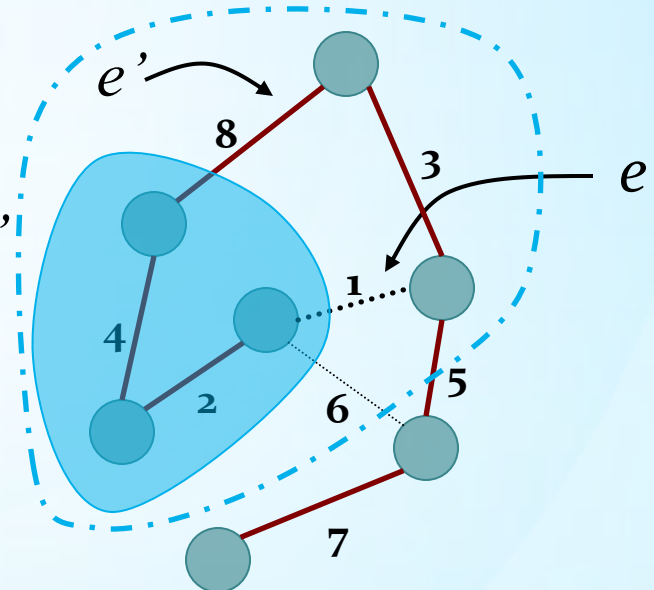
- MST Property 1

Given a fragment of an MST, let e be a minimum-weight outgoing edge of the fragment. Then joining e and its adjacent non-fragment node to the fragment yields another fragment of an MST.

- *Proof:*

Suppose e is not in MST, but some e' instead.

MST with e forms a cycle.



MST Properties

- MST Property 1

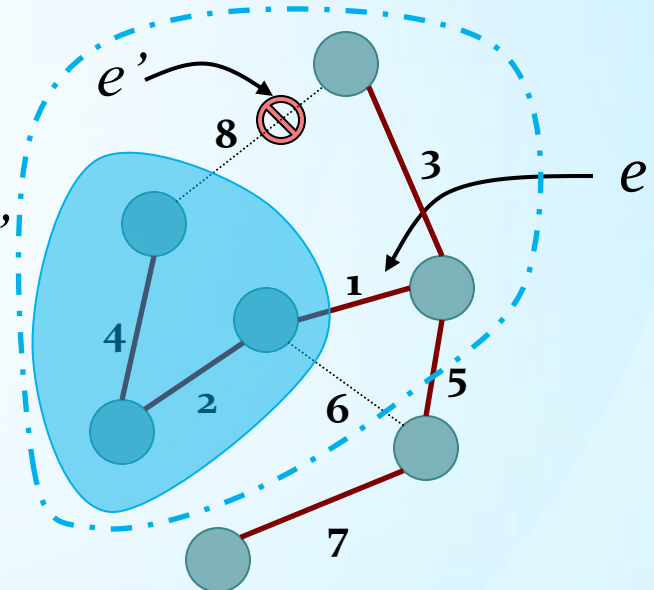
Given a fragment of an MST, let e be a minimum-weight outgoing edge of the fragment. Then joining e and its adjacent non-fragment node to the fragment yields another fragment of an MST.

- *Proof:*

Suppose e is not in MST, but some e' instead.

MST with e forms a cycle.

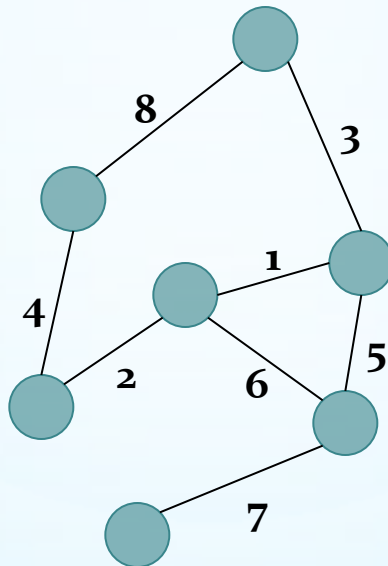
We obtain a cheaper MST with e instead of e' .



MST Properties

- MST Property 2

If all the edges of a connected graph have different weights, then the MST is unique.

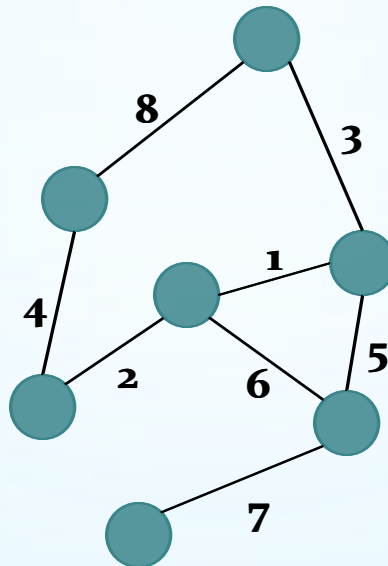


MST Properties

- MST Property 2

If all the edges of a connected graph have different weights, then the MST is unique.

- *Proof:*



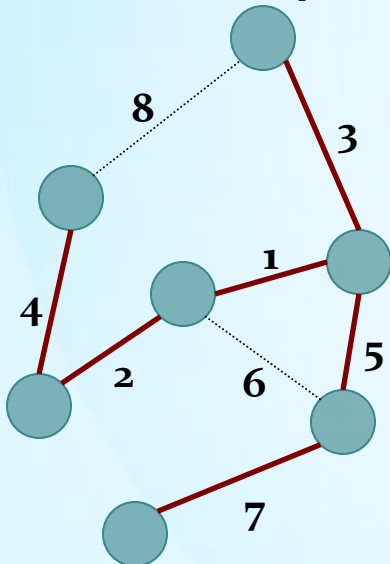
MST Properties

- MST Property 2

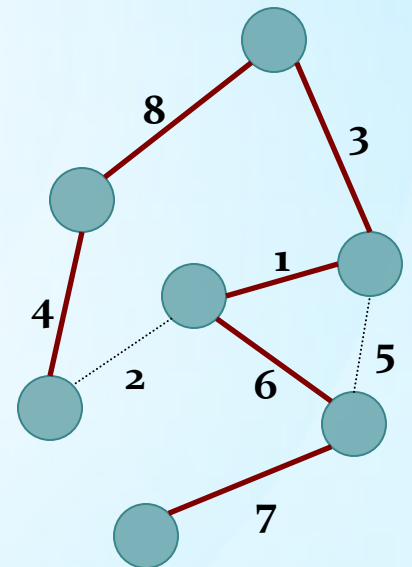
If all the edges of a connected graph have different weights, then the MST is unique.

- *Proof:*

Suppose existence of two MSTs.



T



T'

MST Properties

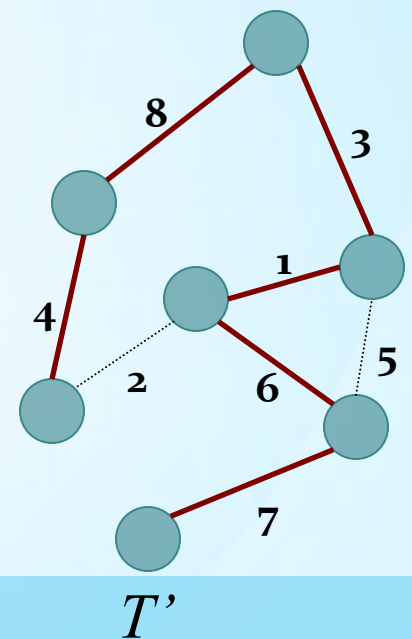
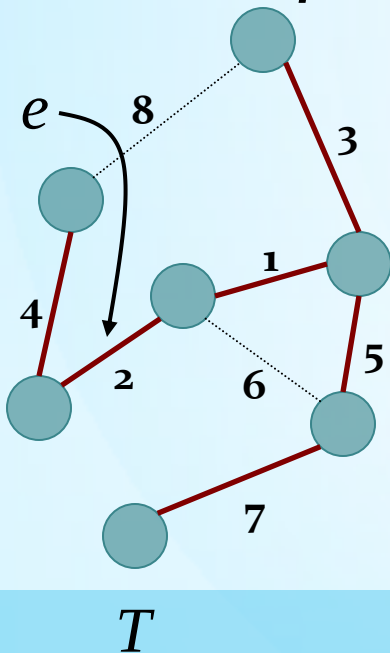
- MST Property 2

If all the edges of a connected graph have different weights, then the MST is unique.

- *Proof:*

Suppose existence of two MSTs.

Let e be the minimal-weight edge not in both MSTs (wlog e in T).

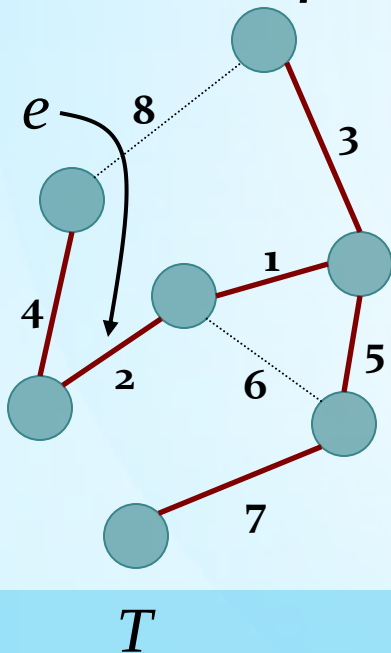


MST Properties

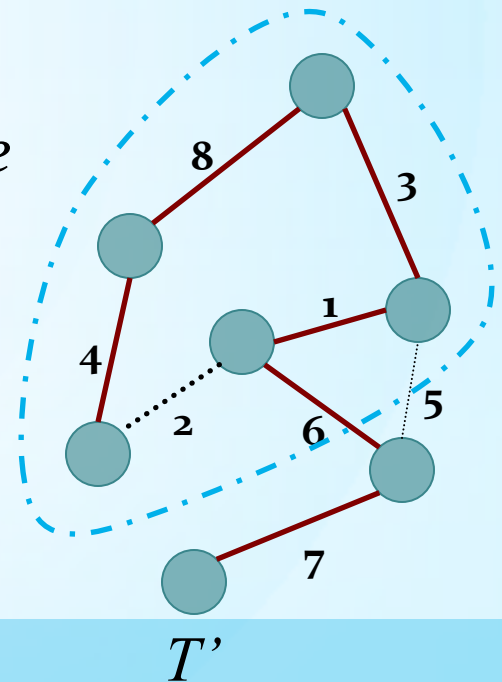
- MST Property 2

If all the edges of a connected graph have different weights, then the MST is unique.

- *Proof:*



*Suppose existence of two MSTs.
Let e be the minimal-weight edge not in both MSTs (wlog e in T).
 T' with e has a cycle*

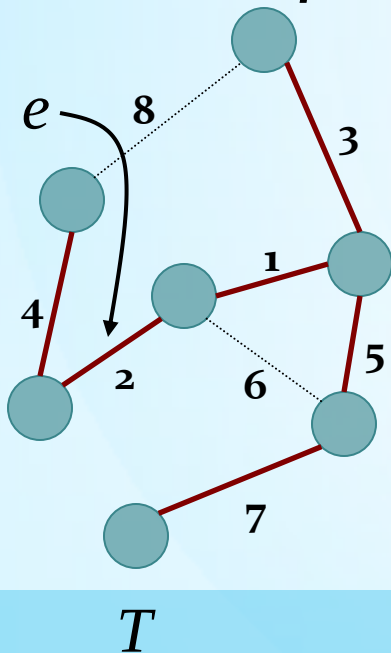


MST Properties

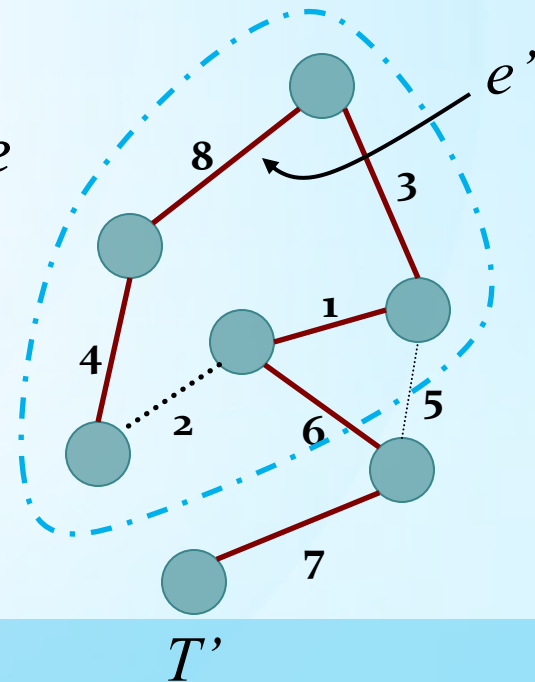
- MST Property 2

If all the edges of a connected graph have different weights, then the MST is unique.

- *Proof:*



*Suppose existence of two MSTs.
Let e be the minimal-weight edge not in both MSTs (wlog e in T).
 T' with e has a cycle
At least cycle edge e' is not in T .*

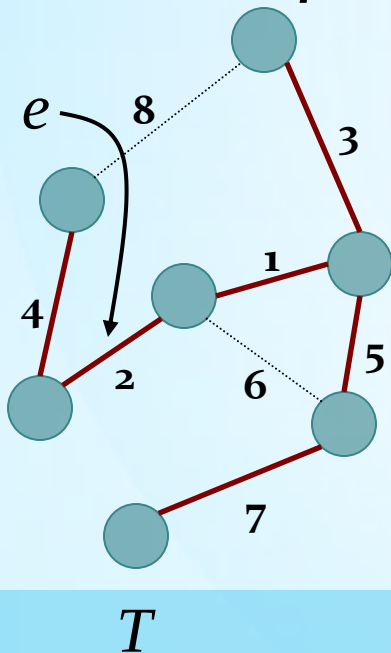


MST Properties

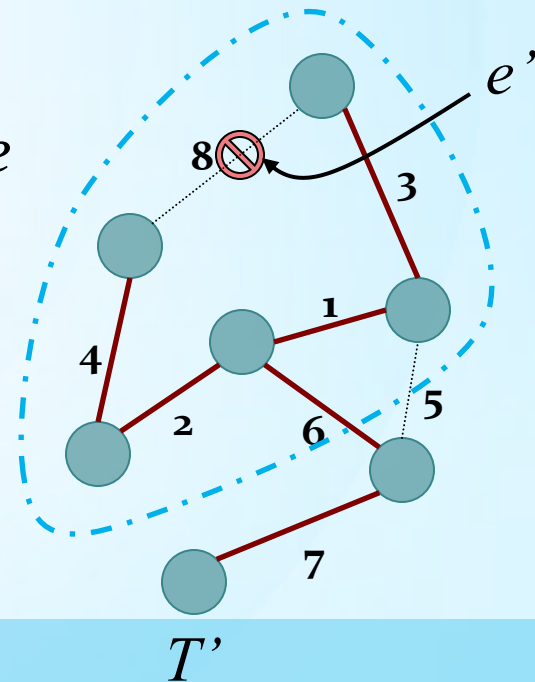
- MST Property 2

If all the edges of a connected graph have different weights, then the MST is unique.

- Proof:

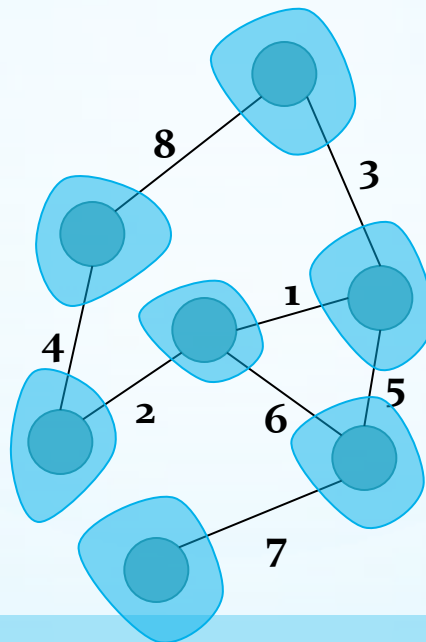


Suppose existence of two MSTs.
Let e be the minimal-weight edge not in both MSTs (wlog e in T).
 T' with e has a cycle
At least cycle edge e' is not in T .
Since $w(e) < w(e')$ we conclude that T' with e and without e' is a smaller MST than T' .



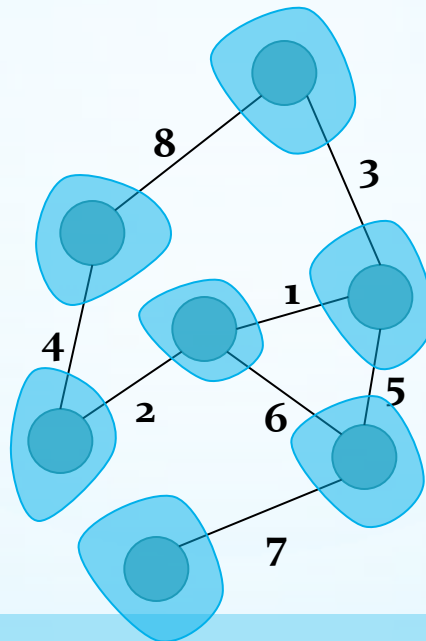
MST Properties

- Idea of MST based on properties 1 & 2
 - Start with fragments of one node.



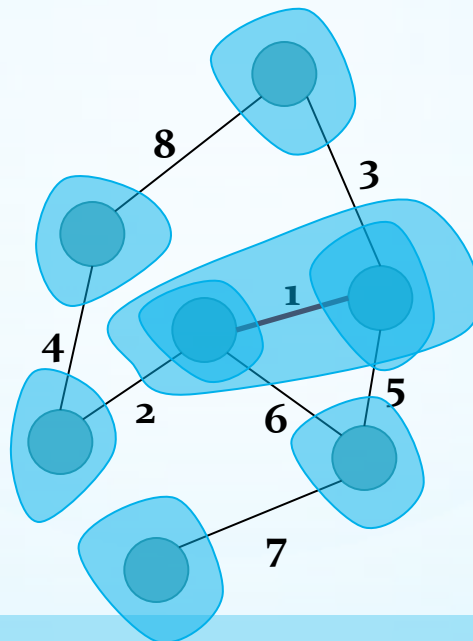
MST Properties

- Idea of MST based on properties 1 & 2
 - Enlarge fragments in any order (property 1)
 - Combine fragments with a common node
 - (property 2)



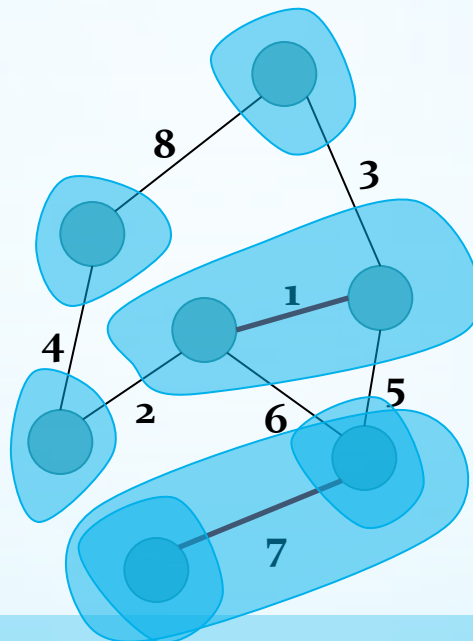
MST Properties

- Idea of MST based on properties 1 & 2
 - Enlarge fragments in any order (property 1)
 - Combine fragments with a common node
 - (property 2)



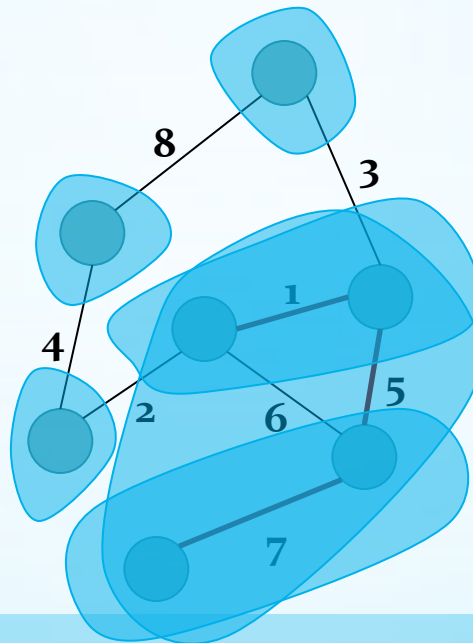
MST Properties

- Idea of MST based on properties 1 & 2
 - Enlarge fragments in any order (property 1)
 - Combine fragments with a common node
 - (property 2)



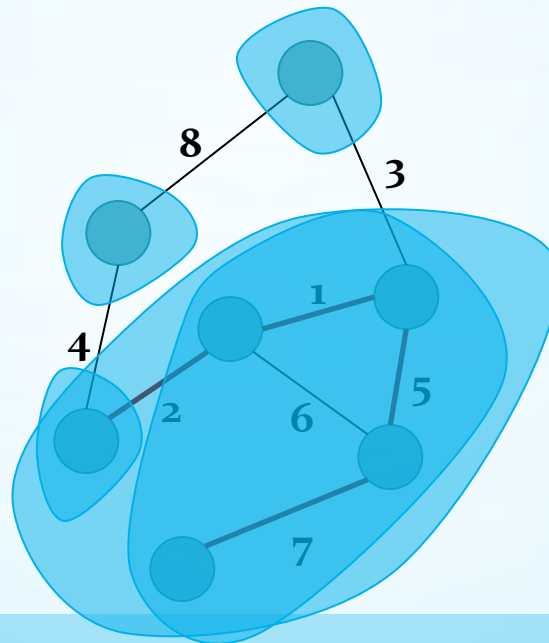
MST Properties

- Idea of MST based on properties 1 & 2
 - Enlarge fragments in any order (property 1)
 - Combine fragments with a common node
 - (property 2)



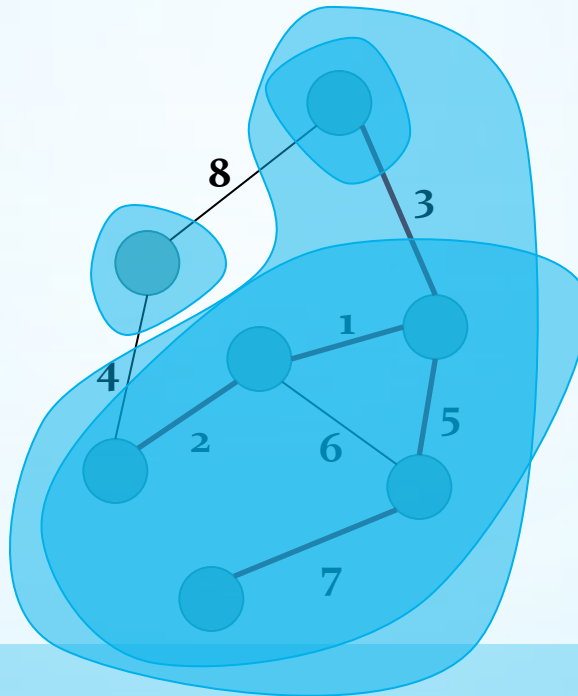
MST Properties

- Idea of MST based on properties 1 & 2
 - Enlarge fragments in any order (property 1)
 - Combine fragments with a common node
 - (property 2)



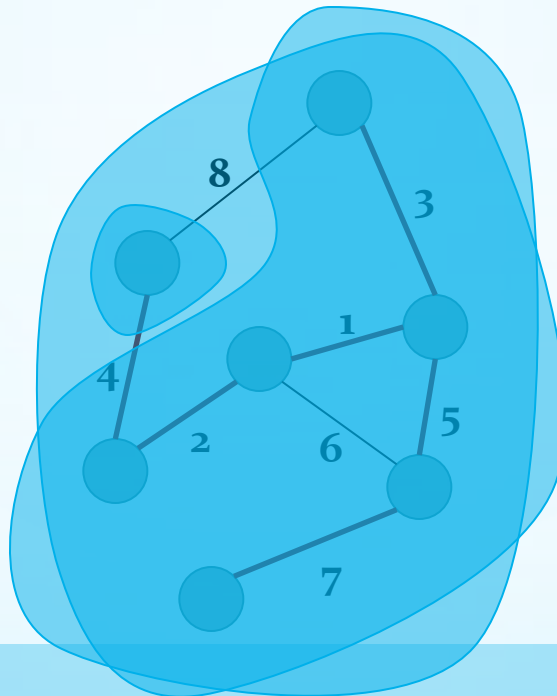
MST Properties

- Idea of MST based on properties 1 & 2
 - Enlarge fragments in any order (property 1)
 - Combine fragments with a common node
 - (property 2)



MST Properties

- Idea of MST based on properties 1 & 2
 - Enlarge fragments in any order (property 1)
 - Combine fragments with a common node
 - (property 2)



MST in Message Passing Model

- The **message-passing model** is one of the most commonly used models in distributed computing. In this model, **each process is modeled as a node** of a graph. The **communication channel between two processes is an edge** of the graph.
- Two commonly used algorithms for the **classical minimum spanning tree problem** are **Prim's algorithm and kruskal's algorithm**. However, it is difficult to apply these two algorithms in the distributed message-passing model. The **main challenges** are:
 - i. Both Prim's algorithm and Kruskal's algorithm require **processing one node or vertex at a time**, making it difficult to make them run in parallel.
 - ii. Both Prim's algorithm and Kruskal's algorithm require **processes to know the state of the whole graph**, which is very difficult to discover in the message-passing model.
- Due to these difficulties, new techniques were needed for **distributed MST algorithms** in the message-passing model.

GHS Algorithm

GHS Algorithm

- The **GHS algorithm of 'Gallager, Humblet and Spira' (1983)** is one of the best-known algorithms in distributed computing theory.
- GHS is a distributed algorithm based on Kruskal's algorithm** that constructs the minimum-weight spanning tree in a **connected undirected graph with distinct edge weights**.
- A processor exists at each node of the graph, knowing initially only the weights of the adjacent edges. The processors obey the same algorithm and exchange messages with neighbors until the tree is constructed.
- This algorithm can construct the **minimum spanning tree in asynchronous Message-passing model**.

GHS Algorithm: Synchronous vs. Asynchronous

1) Synchronous GHS:

- Works with non-uniform models, distinct weights

Steps:

1. Initially, each node is a fragment
2. **Repeat in parallel:** (Synchronous Phase)
 - i. Each fragment – coordinated by a fragment root node - finds its Minimum Weight Outgoing Edge (MWOE)
 - ii. Merge fragments adjacent to MWOE's
3. Until there is only one fragment left

2) Asynchronous GHS:

- Simulates the synchronous version
- Works with uniform models, distinct weights

Steps:

1. Every fragment F has a level $L(F) \geq 0$: at the beginning, each node is a fragment of level 0
2. Two types of merges: absorption and join

Note: We will work with the 'Asynchronous GHS Algorithm'

Overview

- The input graph $G(V,E)$ is considered to be a network, where **vertices V** are independent computing nodes and **edges E** are communication links. Links are weighted as in the classical problem.
- At the beginning of the algorithm, **nodes know only the weights of the links which are connected to them.** (It is possible to consider models in which they know more, for example their neighbors' links.)
- As the output of the algorithm, **every node knows which of its links belong to the Minimum Spanning Tree and which do not.**

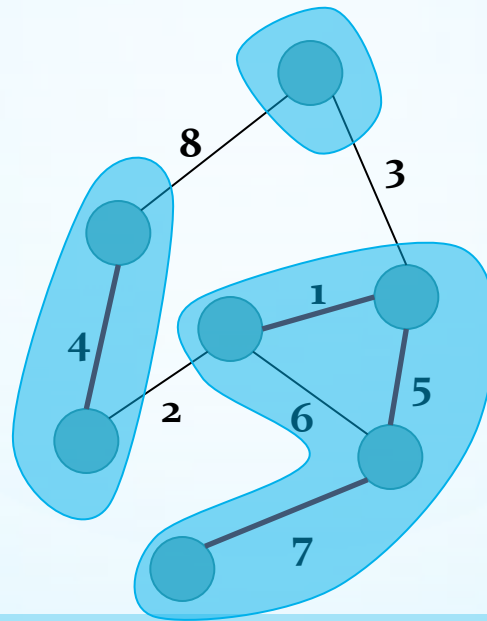
Preconditions

1. The algorithm should run on a **connected undirected graph**.
2. The graph should have **distinct finite weights** assigned to each edge.
3. Each node initially knows the weight for each edge incident to that node.
4. Initially, each node is in a quiescent state and it either spontaneously awakens or is awakened by receipt of any message from another node.
5. Messages can be transmitted independently in both directions on an edge and arrive after an unpredictable but finite delay, without error.
6. Each edge delivers messages in **FIFO order**.

The Idea of Distributed MST

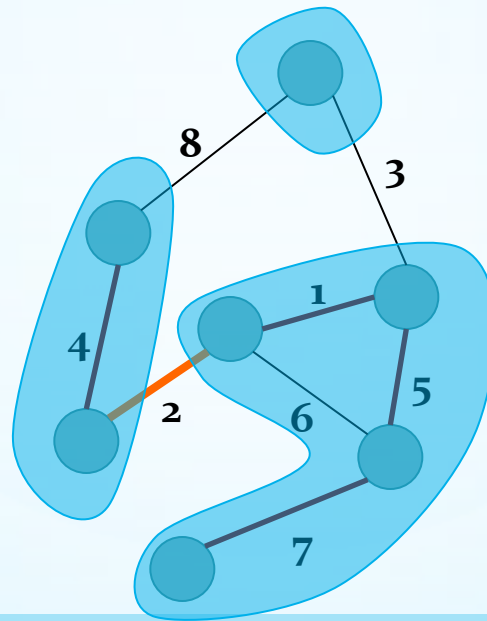
GHS Notation: Fragments

- Fragments
 - Every node starts as a single fragment.



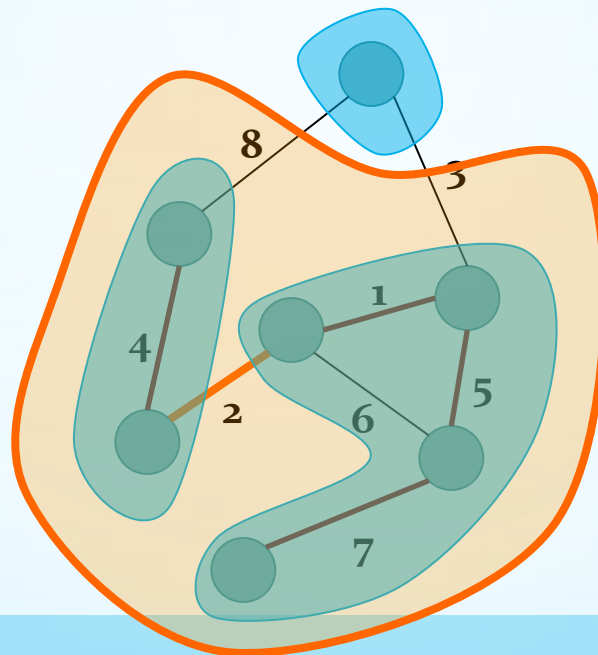
GHS Notation: Fragments

- Fragments
 - Each fragment finds its minimum outgoing edge.



GHS Notation: Fragments

- Fragments
 - Each fragment finds its minimum outgoing edge.
 - Then it tries to combine with the adjacent fragment.

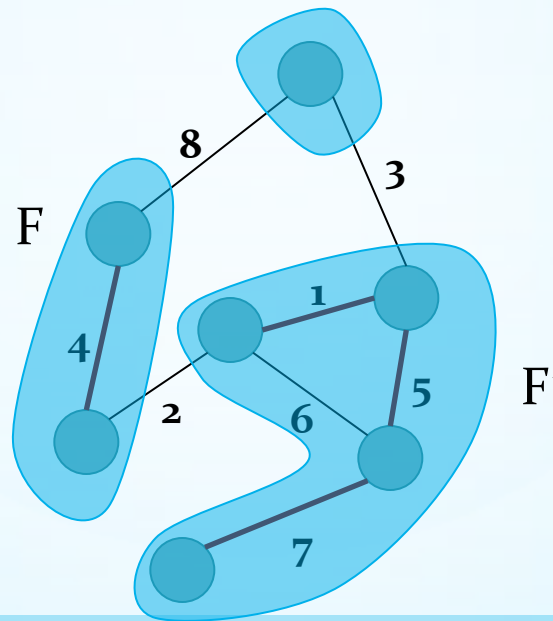


GHS Notation: Levels

- Levels
 - Every fragment has an associated level that
 - has impact on combining fragments.
 - A fragment with a single node is defined to
 - to be at level 0.

GHS Notation: Levels

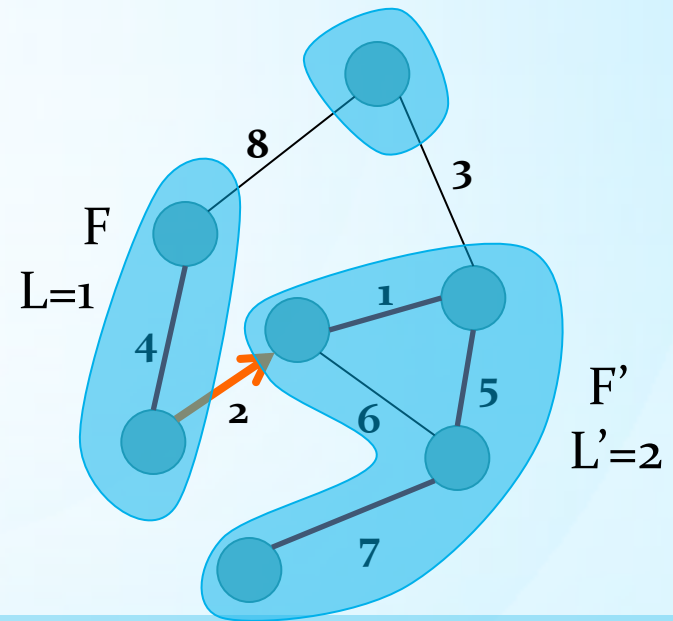
- Levels
 - The combination of two fragments depends on
 - the levels of fragments.



GHS Notation: Levels

- Levels
 - The combination of two fragments depends on
 - the levels of fragments.

If a fragment F wishes to connect to a fragment F' and $L < L'$ then:

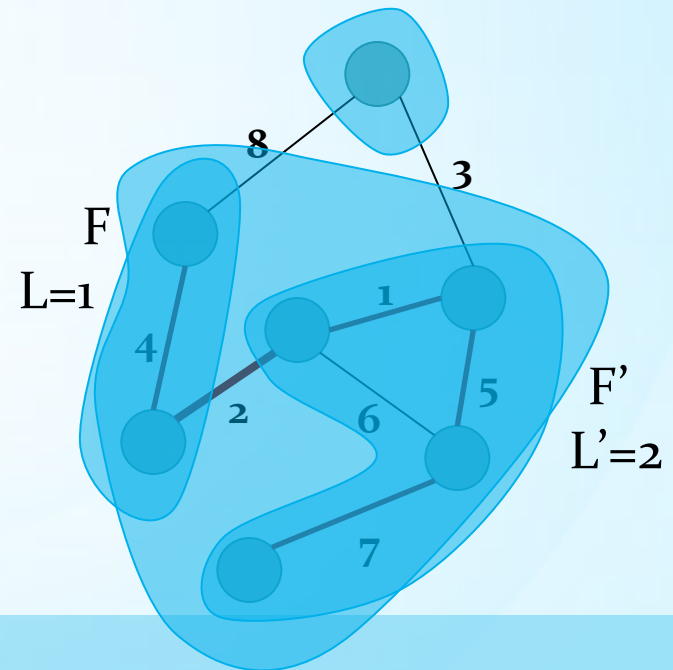


GHS Notation: Levels

- Levels
 - The combination of two fragments depends on
 - the levels of fragments.

If a fragment F wishes to connect to a fragment F' and $L < L'$ then:

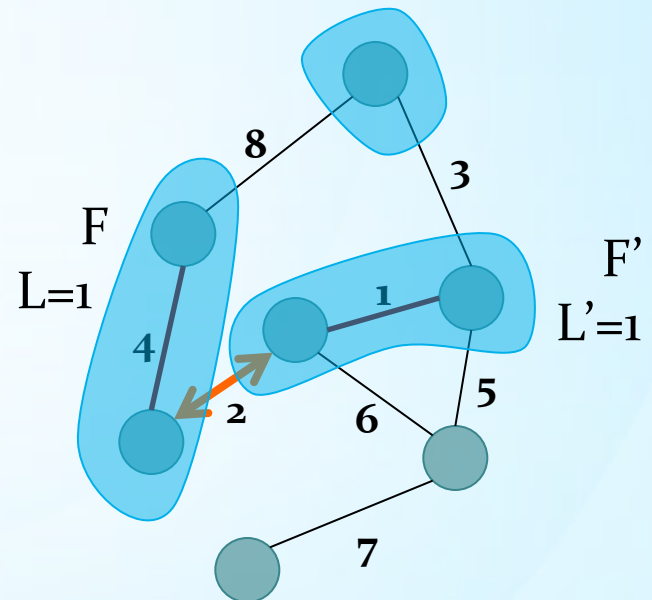
F is absorbed in F' and the resulting fragment is at level L' .



GHS Notation: Levels

- Levels
 - The combination of two fragments depends on
 - the levels of fragments.

If fragments F and F' have the same minimum outgoing edge and $L = L'$ then:

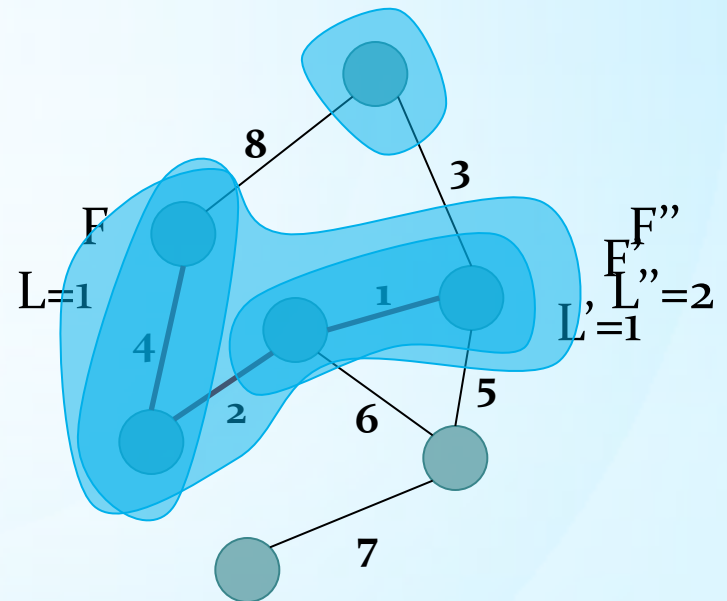


GHS Notation: Levels

- Levels
 - The combination of two fragments depends on
 - the levels of fragments.

If fragments F and F' have the same minimum outgoing edge and $L = L'$ then:

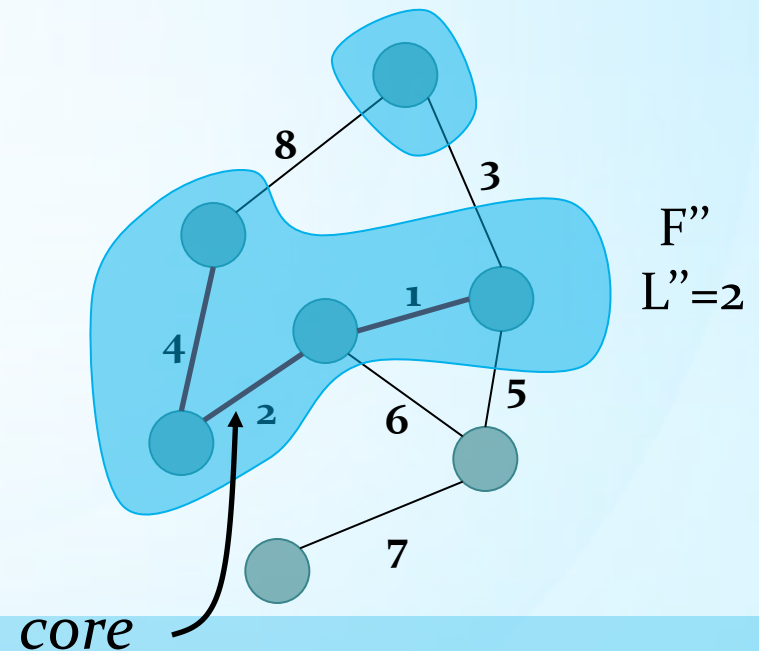
The fragments combine into a new fragment F'' at level $L'' = L + 1$.



GHS Notation: Levels

- Levels
 - The identity of a fragment is the weight of its
 - core.

If fragments F and F' with same level were combined, the combining edge is called the core of the new segment.



GHS Notation: Node States

- State
 - Each node has a state

Sleeping - initial state

Find - during fragment's search
for a minimal outgoing edge

Found - otherwise (when a
minimal outgoing edge was
found)

The Algorithm

Description of the Algorithm (1)

- The GHS algorithm assigns a level to each fragment, which is a non decreasing integer with initial value 0.
- Each non-zero level fragment has an ID, which is the ID of the core edge in the fragment, which is selected when the fragment is constructed.
- During the execution of the algorithm, each node can classify each of its incident edges into three categories:
 - (i) **Branch edges** are those that have already been determined to be part of the MST.
 - (ii) **Rejected edges** are those that have already been determined not to be part of the MST.
 - (iii) **Basic edges** are neither branch edges nor rejected edges.

Description of the Algorithm (2)

- **For level-0 fragments**, each **awakened node** will do the following:

1. Choose its minimum-weight incident edge and **marks that edge as a branch edge**.

2. Send a **message via the branch edge** to notify the node on the other side.

3. **Wait for a message** from the other end of the edge.

- The edge chosen by both nodes **it connects becomes the core** with level 1.

Description of the Algorithm (3)

- **For a non-zero level fragment**, an execution of the algorithm can be separated into three stages in each level:

(i) Broadcast

- The two nodes adjacent to the core **broadcast messages to the rest of the nodes in the fragment**. The messages are **sent via the branch edge** but not via the core.
- Each broadcast message contains the ID and level of the fragment. At the end of this stage, each node has received the new fragment ID and level.

Description of the Algorithm (4)

(ii) Convergecast

- In this stage, all nodes in the fragment cooperate to **find the minimum weight outgoing edge of the fragment**. Outgoing edges are edges connecting to other fragments.
- The messages sent in this stage are in the **opposite direction of the broadcast stage**. Initialized by all the leaves (the nodes that have only one branch edge), a message is sent through the branch edge.
- The message contains the minimum weight of the incident outgoing edge it found (or infinity if no such edge was found). For each non-leaf node, (let the number of its branch edges be n) after receiving $n-1$ convergecast messages, it will pick the minimum weight from the messages and compare it to the weights of its incident outgoing edges. The smallest weight will be sent toward the branch it received the broadcast from.

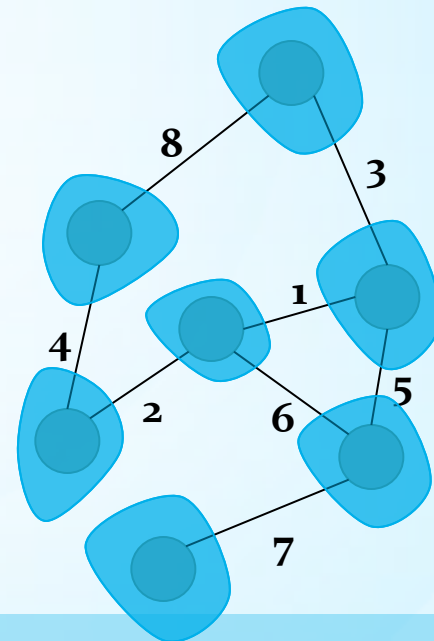
Description of the Algorithm (5)

(iii) Change core

- After the completion of the previous stage, the two nodes connected by the core can **inform each other of the best edges they received.**
- Then they can identify the **minimum outgoing edge from the entire fragment.** A message will be sent from the core to the minimum outgoing edge via a path of branch edges.
- Finally, a message will be sent out via the chosen outgoing edge to request to combine the two fragments that the edge connects.

The Algorithm: Execution

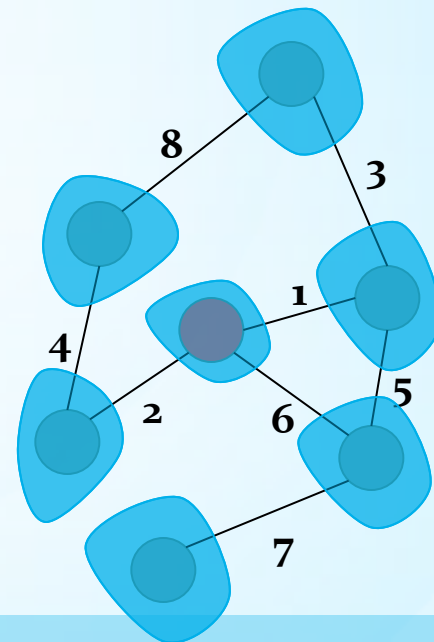
- Fragment minimum outgoing edge discovery
 - Special case of zero-level fragment (*Sleeping*).



The Algorithm: Execution

- Fragment minimum outgoing edge discovery
 - Special case of zero-level fragment (***Sleeping***).

*When a node awakes from the state **Sleeping**, it finds a minimum edge connected.*

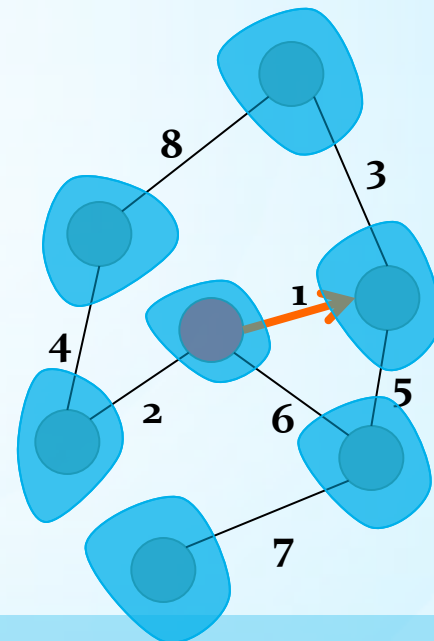


The Algorithm: Execution

- Minimum outgoing edge discovery
 - Special case of zero-level fragment (***Sleeping***).

*When a node awakes from the state ***Sleeping***, it finds a minimum edge connected.*

*Marks it as a **branch** of MST and sends a **Connect** message over this edge.*



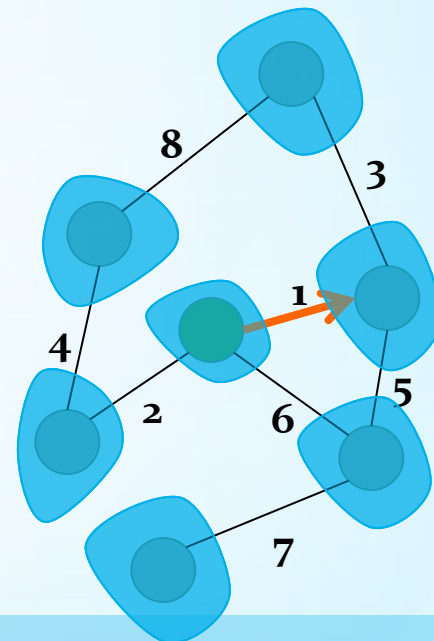
The Algorithm: Execution

- Minimum outgoing edge discovery
 - Special case of zero-level fragment (***Sleeping***).

*When a node awakes from the state ***Sleeping***, it finds a minimum edge connected.*

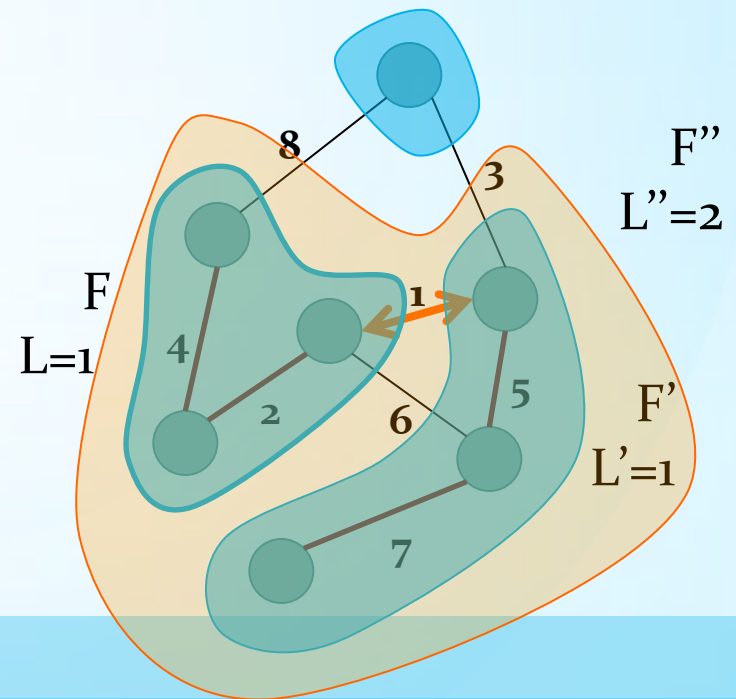
*Marks it as a **branch** of MST and sends a **Connect** message over this edge.*

*Goes into a **Found** state.*



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Take a fragment at level L that was just combined
 - out of two level $L-1$ fragments.

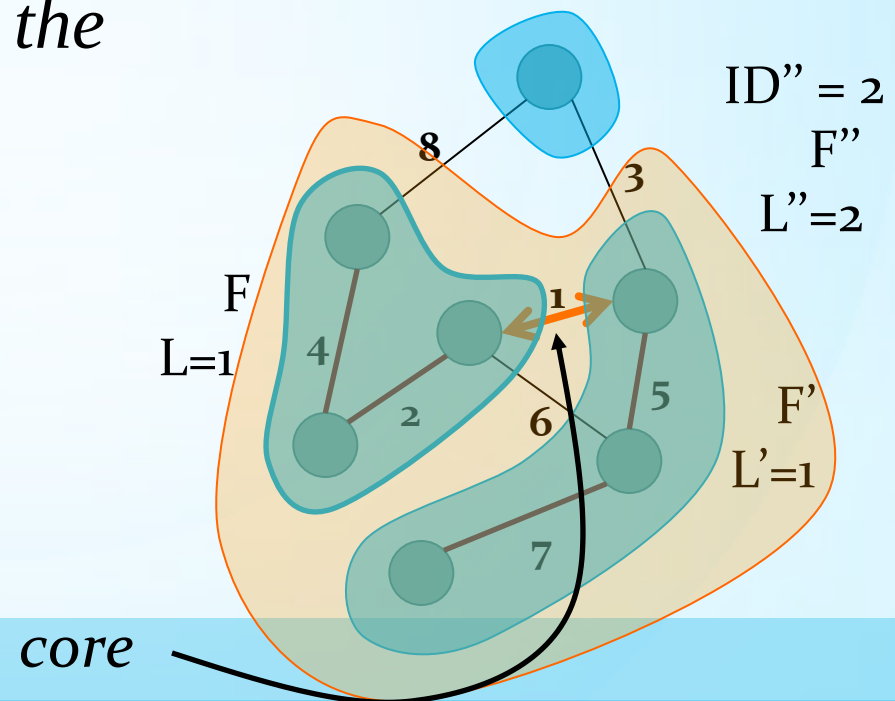


The Algorithm: Execution

- Minimum outgoing edge discovery
 - Take a fragment at level L that was just combined
 - out of two level L-1 fragments.

The weight of the core is the identity of the fragment.

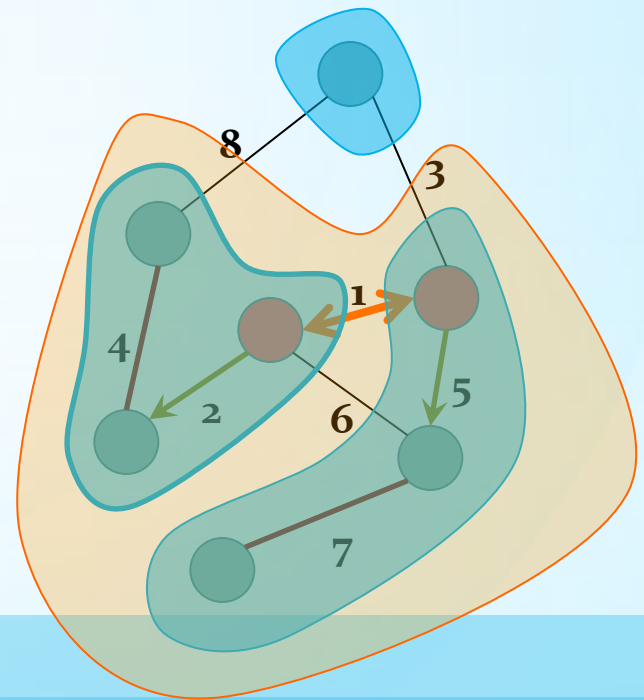
It acts as a root of a fragment tree.



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Take a fragment at level L that was just combined
 - out of two level L-1 fragments.

*Nodes adjacent to core send an **Initiate** message to the borders.*

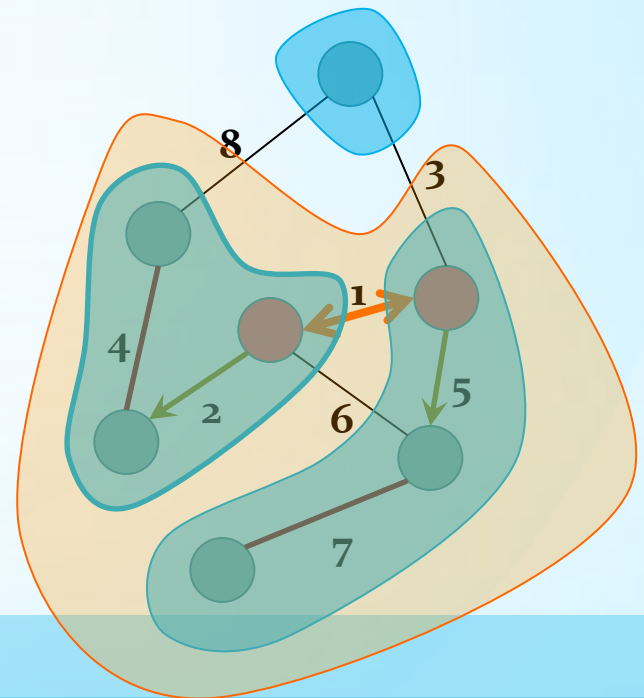


The Algorithm: Execution

- Minimum outgoing edge discovery
 - Take a fragment at level L that was just combined
 - out of two level L-1 fragments.

*Nodes adjacent to core send an **Initiate** message to the borders.*

Relayed by the intermediate nodes in the fragment.



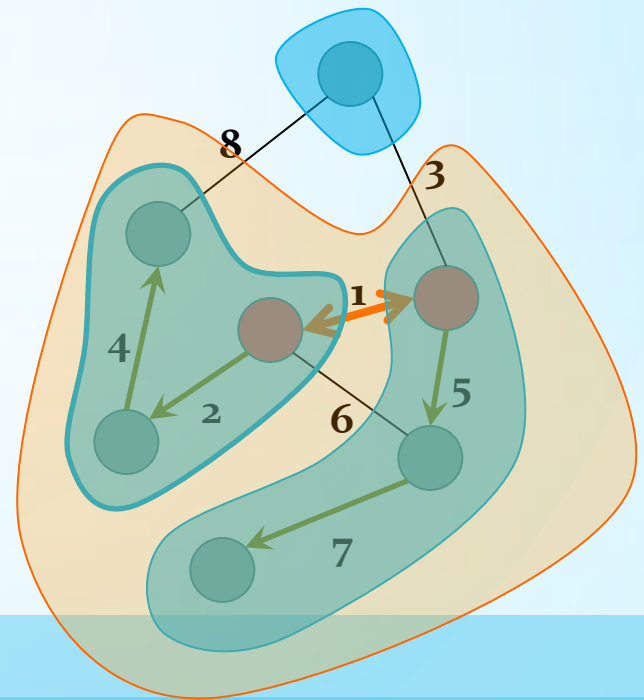
The Algorithm: Execution

- Minimum outgoing edge discovery
 - Take a fragment at level L that was just combined
 - out of two level L-1 fragments.

*Nodes adjacent to core send an **Initiate** message to the borders.*

Relayed by the intermediate nodes in the fragment.

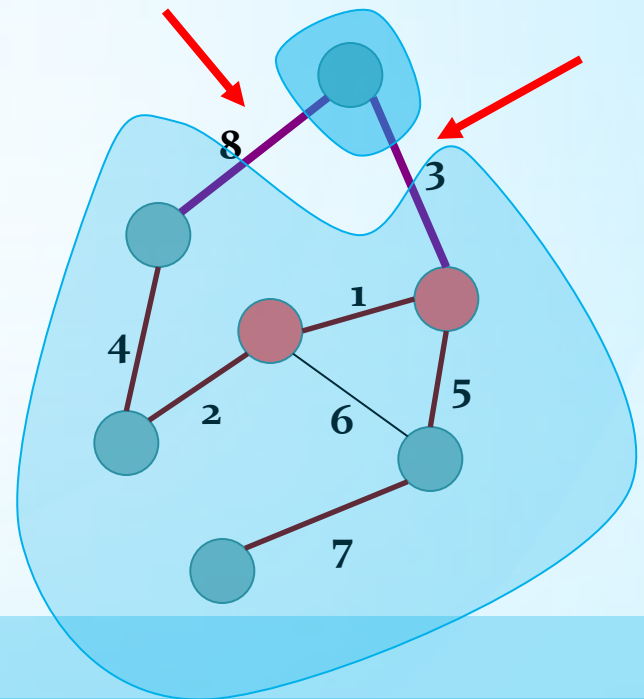
*Puts the nodes in the **Find** state.*



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Edge classification/

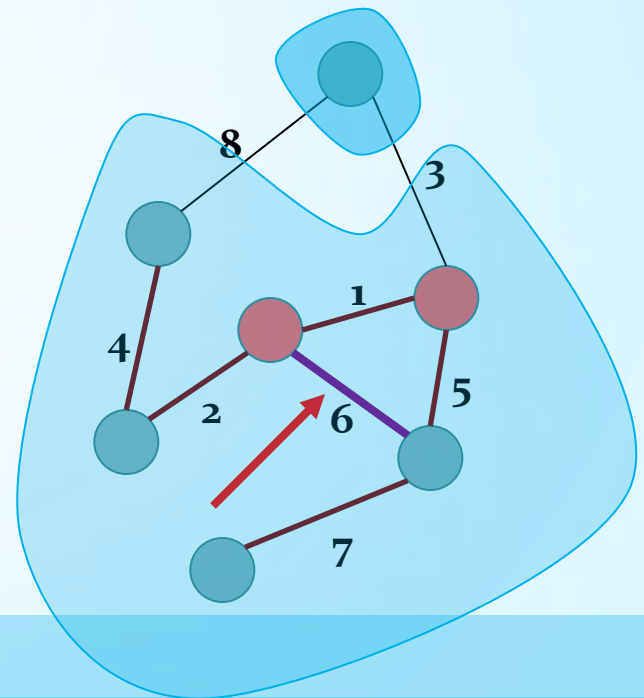
Basic - yet to be classified, can be inside fragment or outgoing edges



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Edge classification.

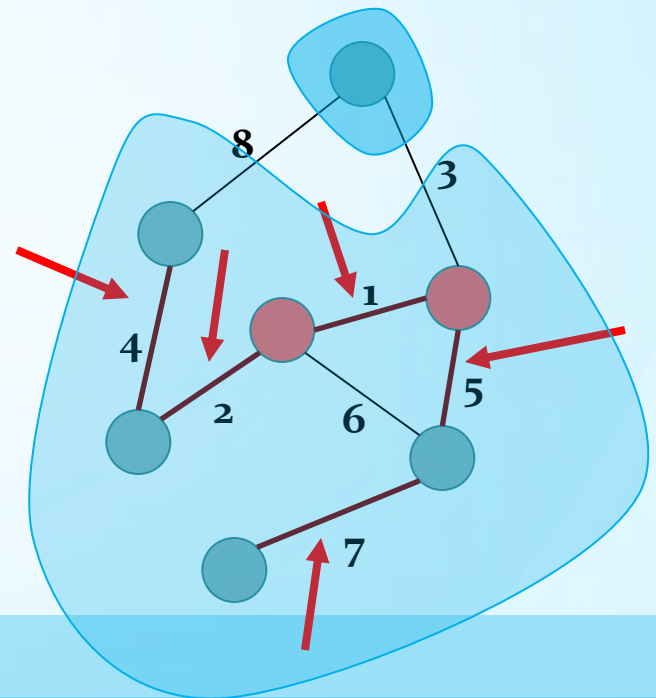
Rejected – an inside fragment edge



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Edge classification.

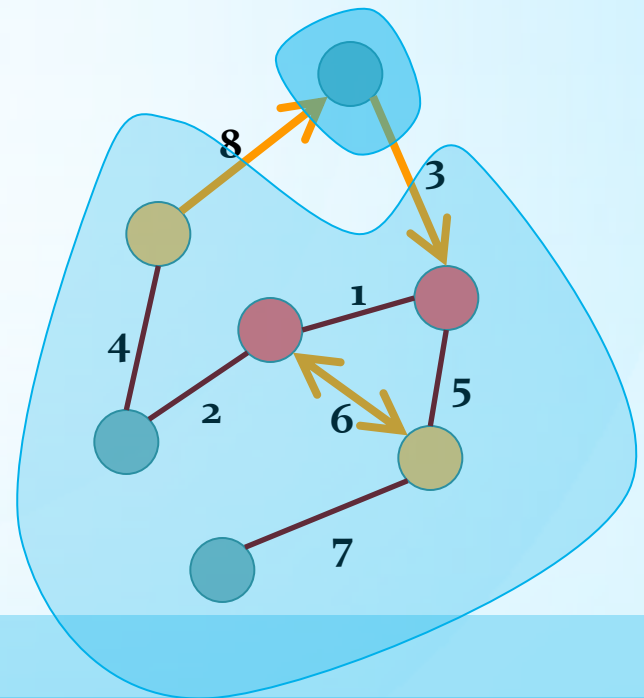
Branch – an MST edge



The Algorithm: Execution

- Minimum outgoing edge discovery
 - On receiving the *Initiate* message a node tries
 - to find a minimum outgoing edge.

*Sends a **Test** message on
Basic edges (minimal first)*

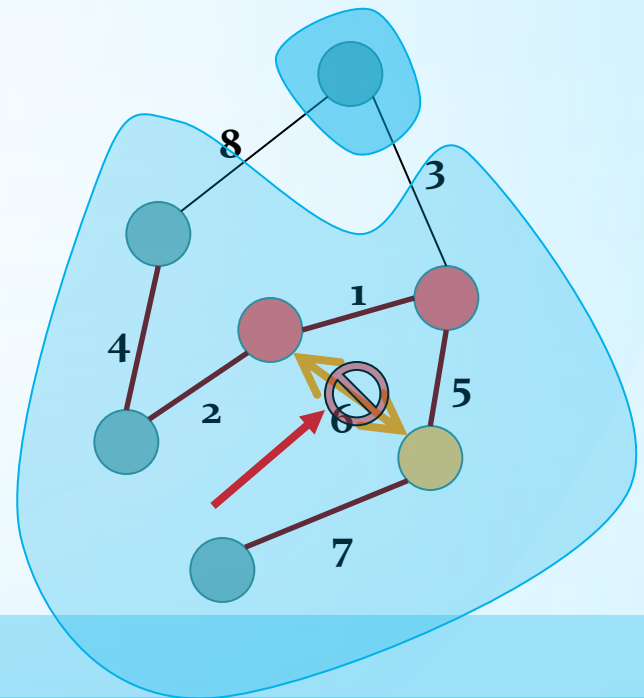


The Algorithm: Execution

- Minimum outgoing edge discovery
 - On receiving the **Test** message.

*In case of same identity:
send a **Reject** message, the
edge is **Rejected**.*

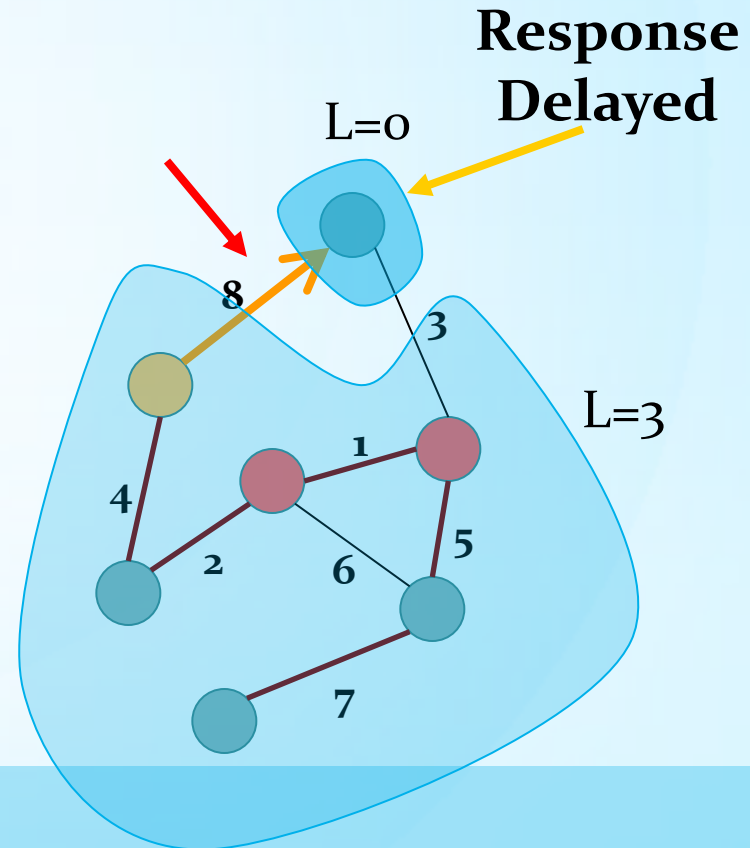
*In case **Test** was sent in
both directions, the edge is
Rejected automatically
without a **Reject** message.*



The Algorithm: Execution

- Minimum outgoing edge discovery
 - On receiving the **Test** message.

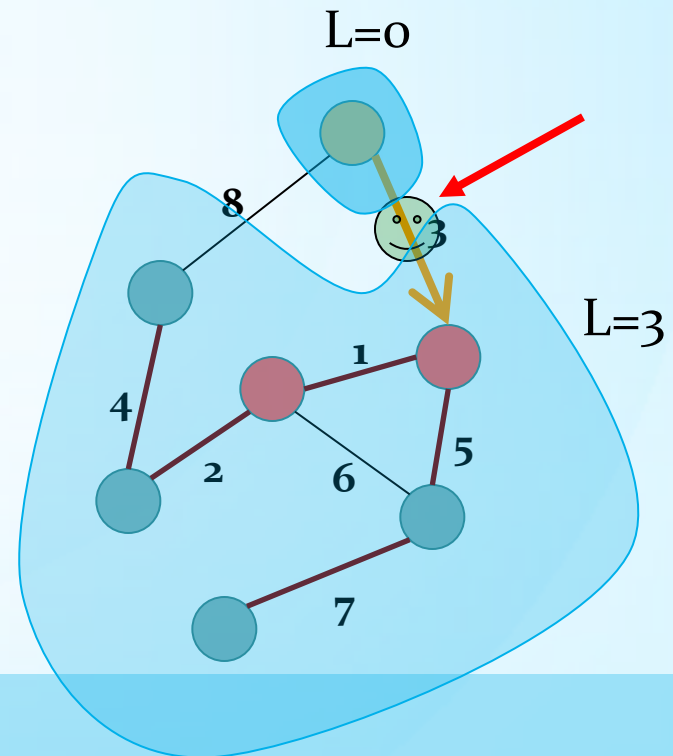
*In case of a self lower level:
Delay the response until the
identity rises sufficiently.*



The Algorithm: Execution

- Minimum outgoing edge discovery
 - On receiving the **Test** message.

*In case of a self higher level:
Send an **Accept** message
The edge is accepted as a
candidate.*

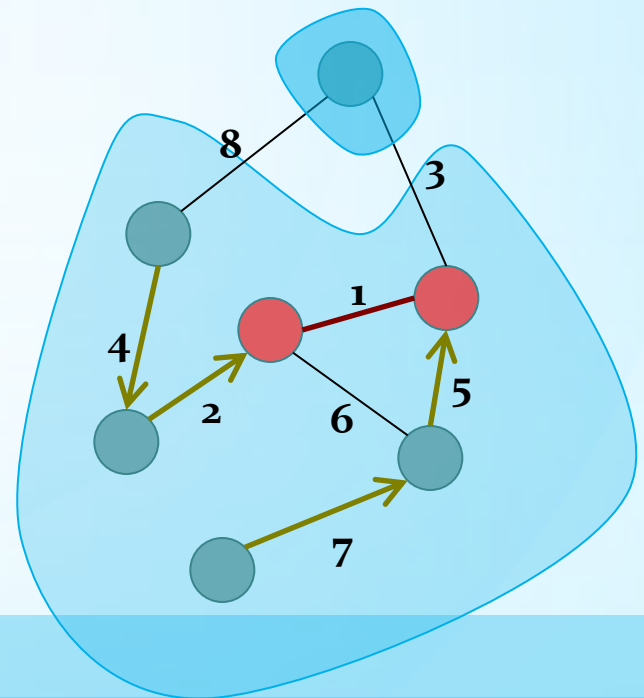


The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

*Nodes send **Report** messages along the branches of the MST.*

If no outgoing edge was found the algorithm is complete.



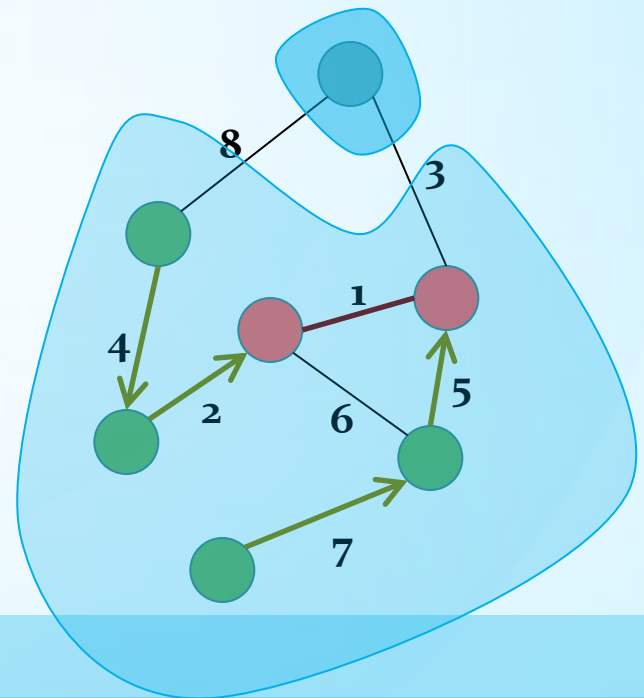
The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

Nodes send **Report** messages along the branches of the MST.

If no outgoing edge was found the algorithm is complete.

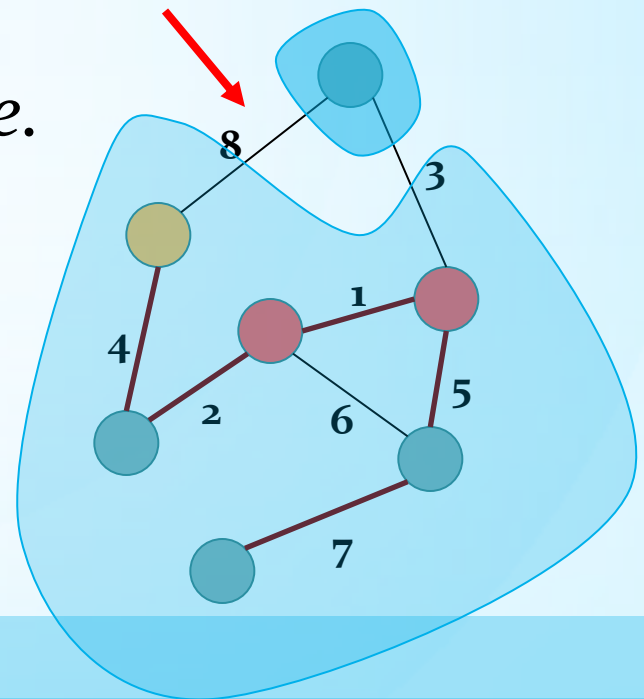
After sending they go into **Found** mode.



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

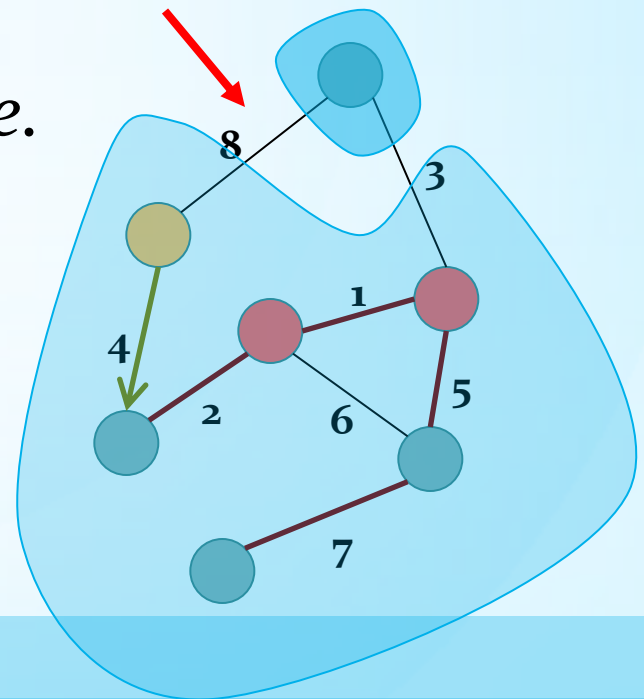
*Every leaf sends the **Report** when resolved its outgoing edge.*



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

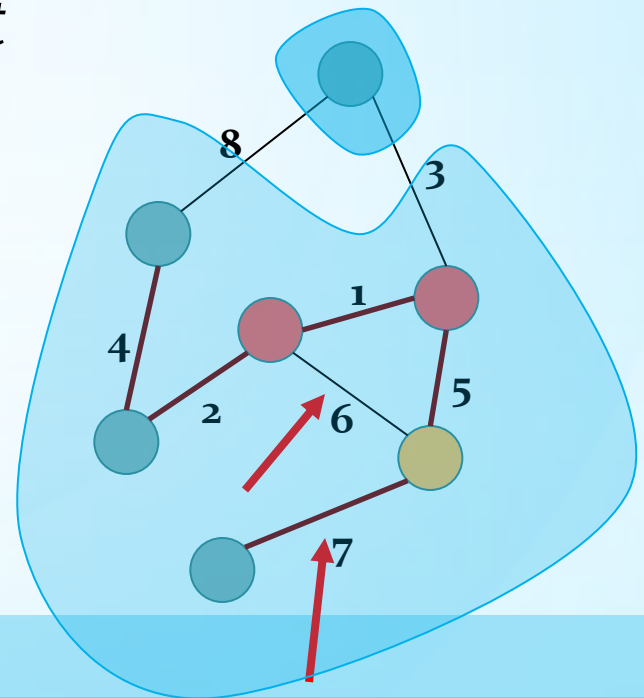
*Every leaf sends the **Report** when resolved its outgoing edge.*



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

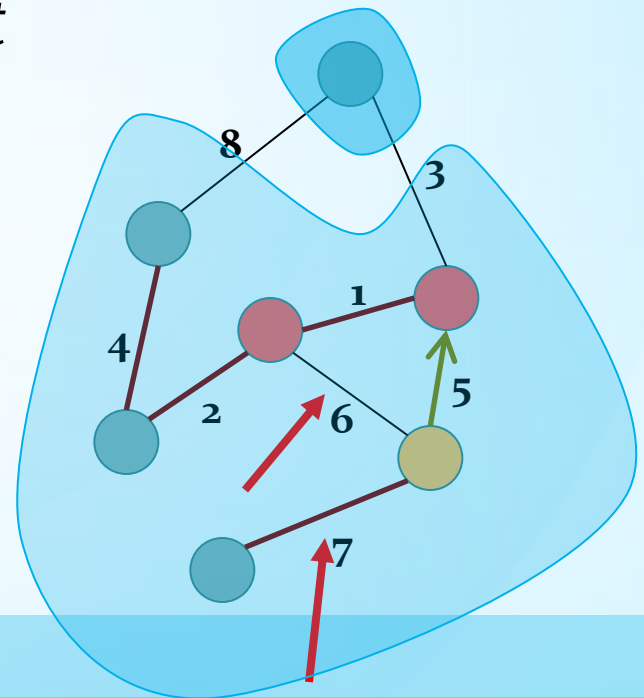
*Every interior sends the **Report** when resolved its outgoing and all its children sent theirs.*



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

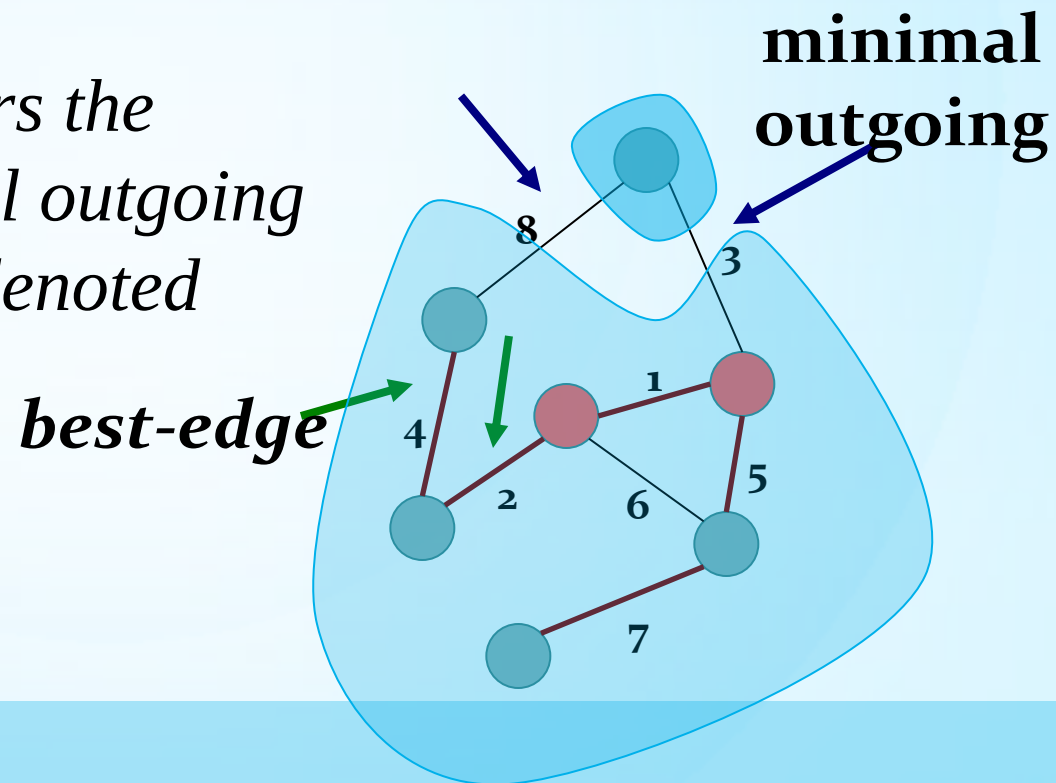
*Every interior sends the **Report** when resolved its outgoing and all its children sent theirs.*



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

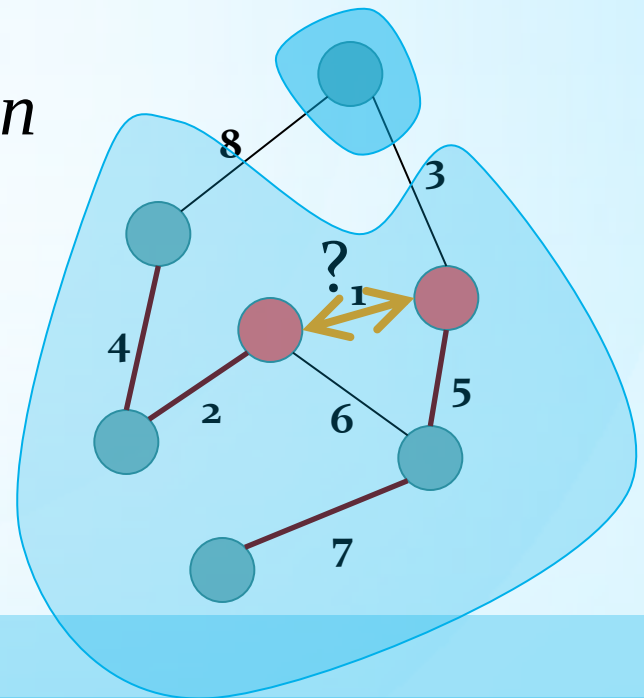
*Every node remembers the branch to the minimal outgoing edge of its sub-tree, denoted **best-edge**.*



The Algorithm: Execution

- Minimum outgoing edge discovery
 - Agreeing on the minimal outgoing edge.

*The core adjacent nodes exchange **Reports** and decide on the minimal outgoing edge.*

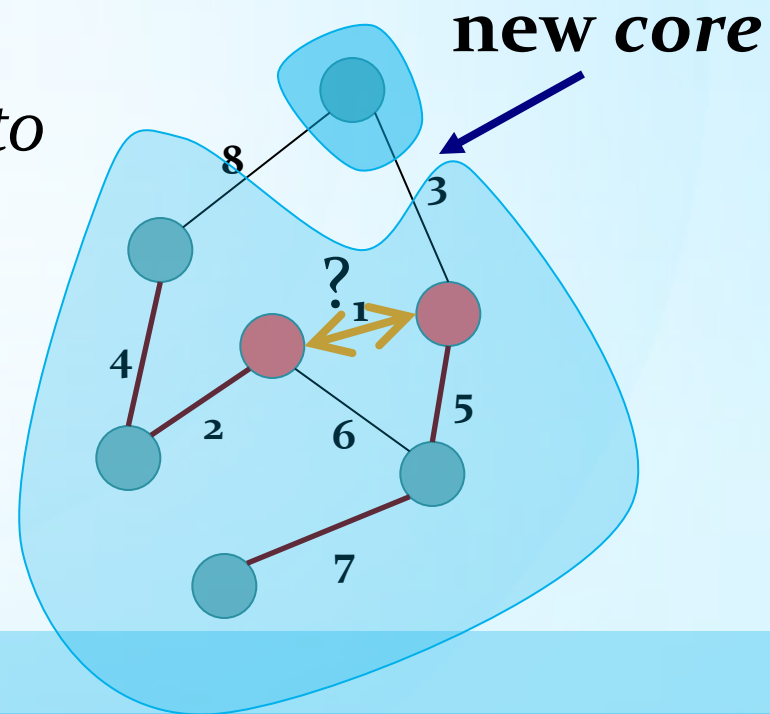


The Algorithm: Execution

- Combining segments
 - Changing core.

*When decided a **Change-core** message is sent over branches to the minimal outgoing edge.*

*The tree **branches** point to the new core.*



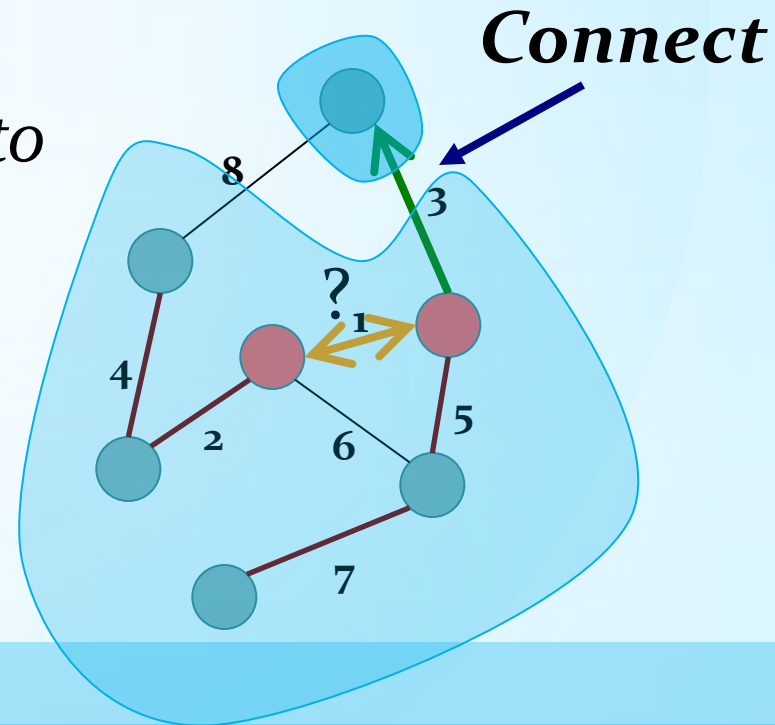
The Algorithm: Execution

- Combining segments
 - Changing core

*When decided a **Change-core** message is sent over branches to the minimal outgoing edge.*

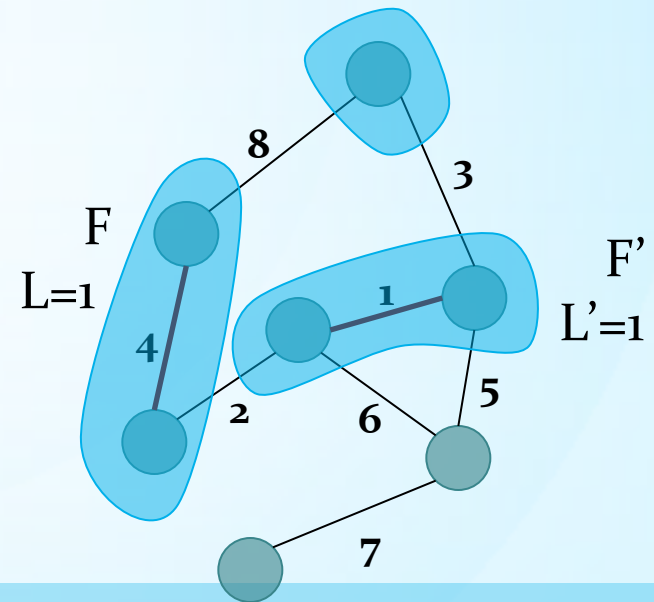
*The tree **branches** point to the new core.*

*Finally a **Connect** message is sent over the minimal edge.*



The Algorithm: Execution

- Final notes
 - Connecting same level fragments.

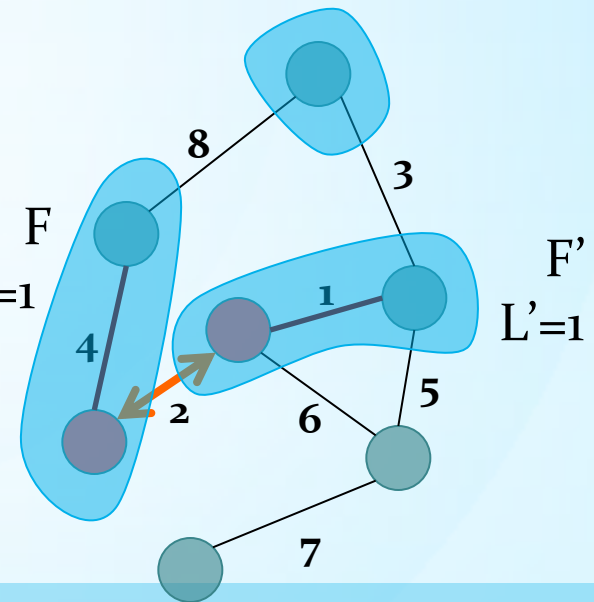


The Algorithm: Execution

- Final notes
 - Connecting same level fragments.

*Both core adjacent nodes send a **Connect** message, which causes the level to be increased.*

*As a result, core is changed and new **Initiate** messages are sent.*



The Algorithm: Execution

- Final notes
 - Connecting lower level fragments.

*When lower level fragment F' at node n' joins some fragment F at node n before n sent its **Report**.*

*We can send n' an **Initiate** message with the **Find** state so it joins the search.*

The Algorithm: Execution

- Final notes
 - Connecting lower level fragments.

*When lower level fragment F' at node n' joins some fragment F at node n after n sent its **Report**.*

*It means that n already found a lower edge and therefore we can send n' an **Initiate** message with the **Found** state so it doesn't join the search.*

The Algorithm: Execution

- Final notes
 - Forwarding the *Initiate* message at level L .

*When forwarding an **Initiate** message to the leafs, it is also forwarded to any pending fragments at level $L-1$, as they might be delayed with response.*

The Algorithm: Execution

- Final notes
 - Upper bound on fragment levels.

Level $L+1$ contains at least 2 fragments at level L .

Level L contains at least 2^L nodes.

$\log_2 N$ is an upper bound on fragment levels.

The Algorithm: Correctness

- Proof outline
 - Correct MST build-up

*The **Connect** message is sent on the minimal outgoing edge only.*

As a result of properties 1 & 2 we should obtain an MST.

The Algorithm: Correctness

- Proof outline
 - No deadlocks

Assume there is a deadlock.

*Choose a fragment from the lowest level set
and with minimal outgoing edge in that set.*

*Its **Test/Connect** message surely will be replied.*

There will always be a working fragment.

The Algorithm: Complexity

- Complexity
 - Communication

*At most E **Reject** messages (with corresponding E **Test** messages, because each edge can be rejected only once.*

The Algorithm: Complexity

- Complexity
 - Communication

*At every but the zero or last levels, each node can accept up to 1 **Initiate**, **Accept** messages. It can transmit up to 1 **Test (successful)**, **Report**, **ChangeRoot**, **Connect**. Since the number of levels is bounded by $\log_2 N$ number of such messages is at most $5N(\log_2 N - 1)$.*

The Algorithm: Complexity

- Complexity
 - Communication

*At level zero, each node receives at most one **Initiate** and transmits at most one **Connect**. At the last level a node can send at most one **Report** message, as a result at most $3N$ such messages.*

The Algorithm: Complexity

- Complexity
 - Communication

As a result, the upper bound is: $5N \log_2 N + 2E$.

The Algorithm: Complexity

- Complexity
 - Time (under assumption of initial awakening it is $5N \log_2 N$)

We prove by induction that one needs $5lN - 3N$ time units for every node to reach level l .

The Algorithm: Complexity

- Complexity

- Time (under assumption of initial awakening it is $5N \log_2 N$)

$l=1 \rightarrow$ Each node is awakened and sends a **Connect** message. By time $2N$ all nodes should be at level 1.

The Algorithm: Complexity

- Complexity

- Time (under assumption of initial awakening it is $5N \log_2 N$)

*Assume $l \rightarrow$ At level l , each node can send at most N **Test** messages which will be answered before time $5lN - N$. The propagation of the **Report** messages, **ChangeRoot**, **Connect**, and **Initiate** messages can take at most $3N$ units, so that by time $5(l + 1)N - 3N$ all nodes are at level $l + 1$.*

The Algorithm: Complexity

- Complexity

- Time (under assumption of initial awakening it is $5N \log_2 N$)

*At the highest level only **Test**, **Reject**, and **Report** messages are used.*

The Algorithm: Complexity

- Complexity
 - Time (under assumption of initial awakening it is $5N \log_2 N$)

As a result we have the algorithm complete under $5N \log_2 N$ time units.

Conclusion

- **Distributed MST algorithms** are useful in communication networks when one wishes to broadcast information from one node to all other nodes and there is a cost associated with each channel of the network.
- In addition to the broadcast application, there are many potential control problems for networks whose communication complexities are reduced by having a known spanning tree.
- In this lecture, we have presented an **asynchronous distributed algorithm i.e. GHS algorithm** which determines the minimum-weight spanning tree (MST) of the graph.
- In upcoming lecture we will discuss about '**Termination Detection**' and set of representative termination detection algorithms.