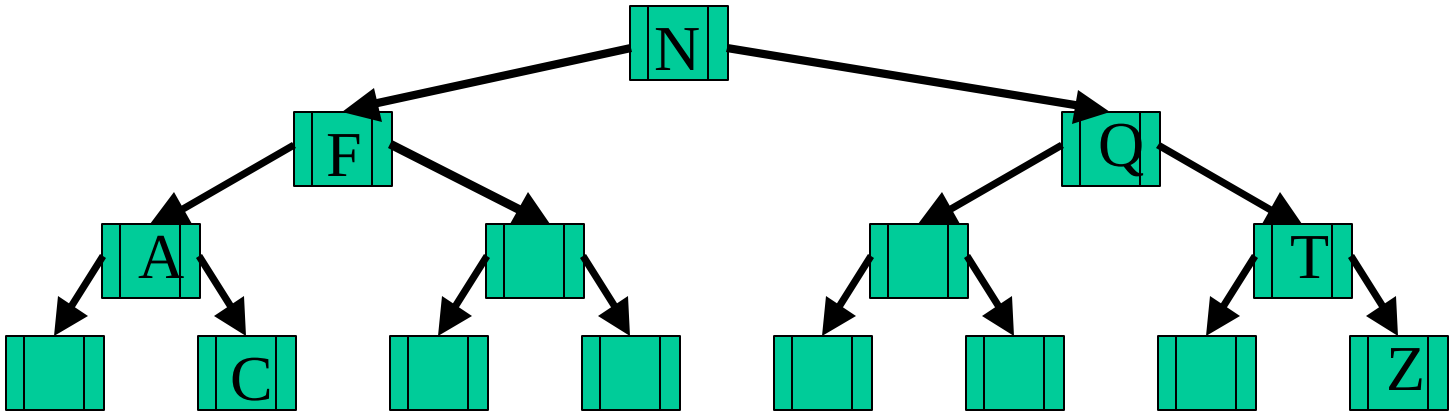
Binary Tree

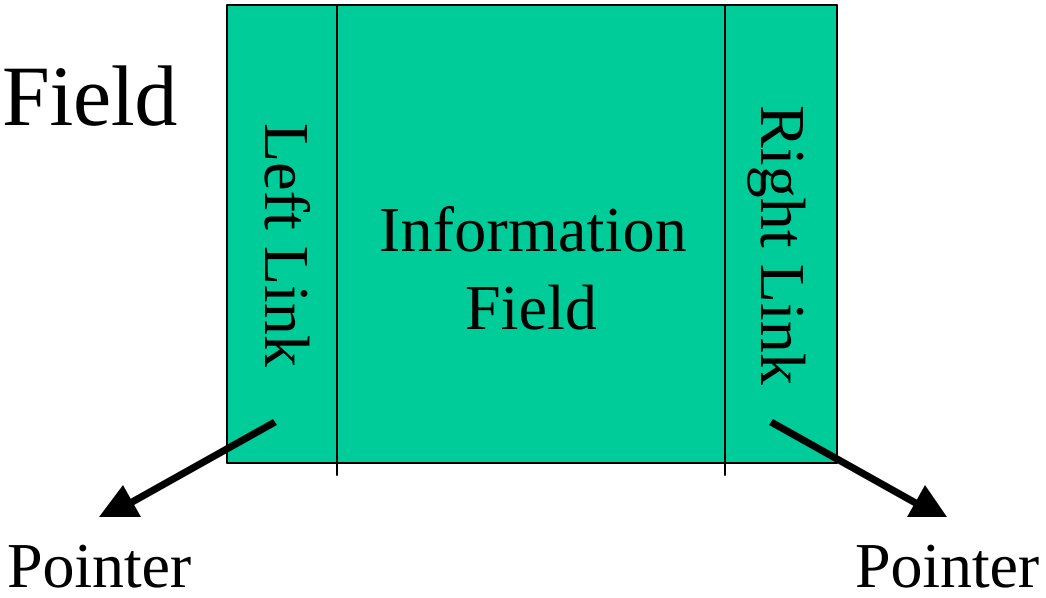


- A binary tree is a data structure consisting of
  - Nodes
  - Pointers or Branches

Node Structure

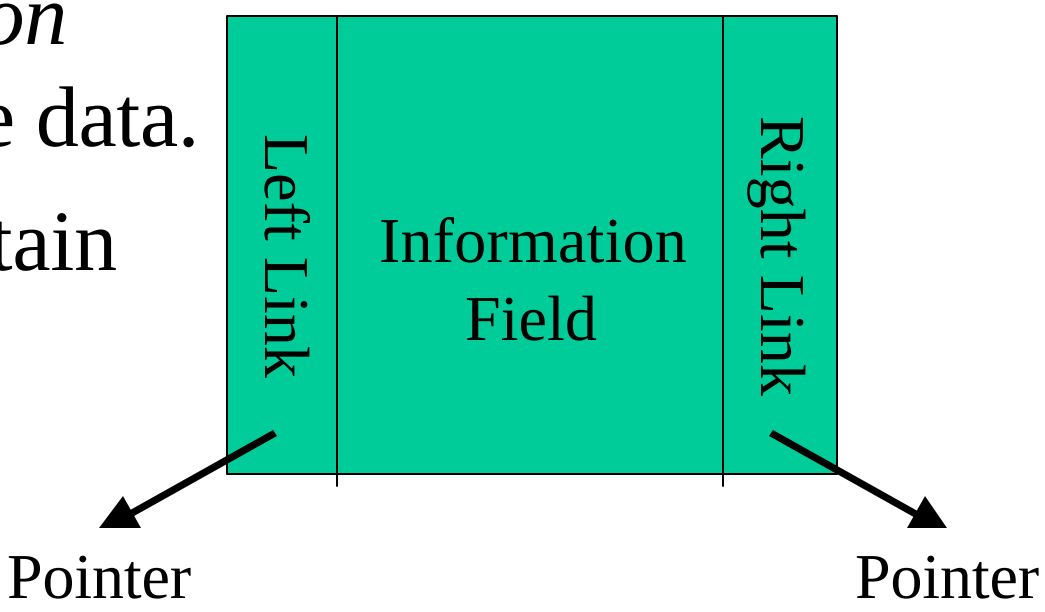
A node in a binary tree has three components:

- Information Field
- Left Link
- Right Link



Node Contents

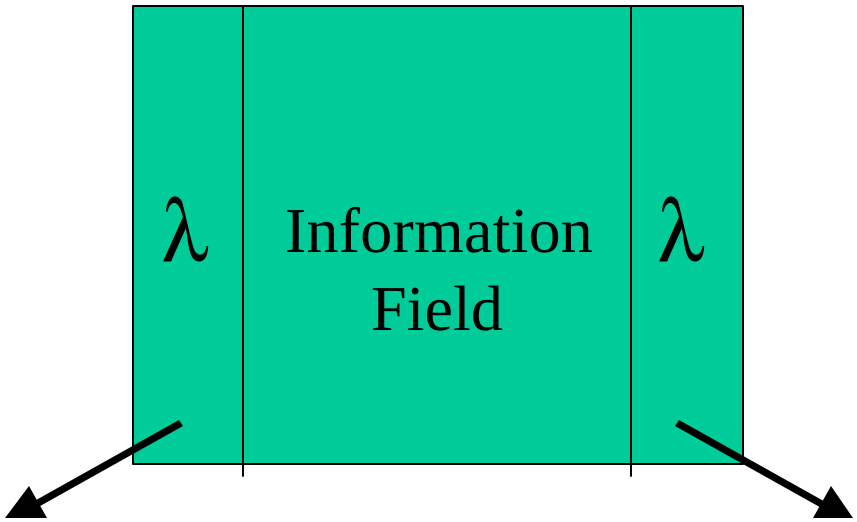
- The *Information Field* holds the data.
- The *Links* contain a *Pointer* to related data.



Null Pointer

λ

The lower case Greek letter lambda (for “link”) is used to indicate a null (empty, unused) link.



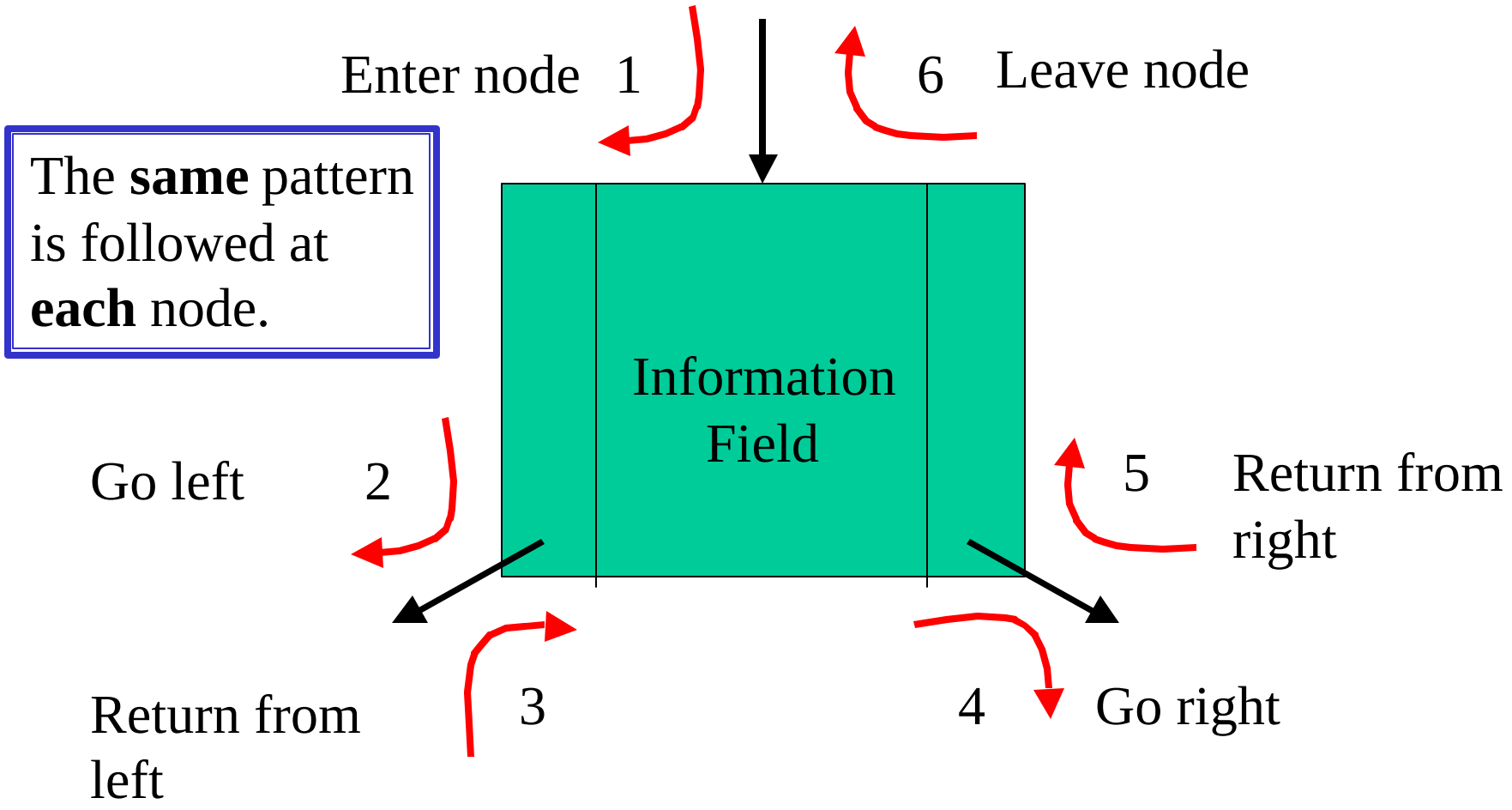
Traversals

Definition of ***Traverse***:

- To pass over, across, or through
- To go back and forth over or along

Traversing

The act of following a path through a binary tree.

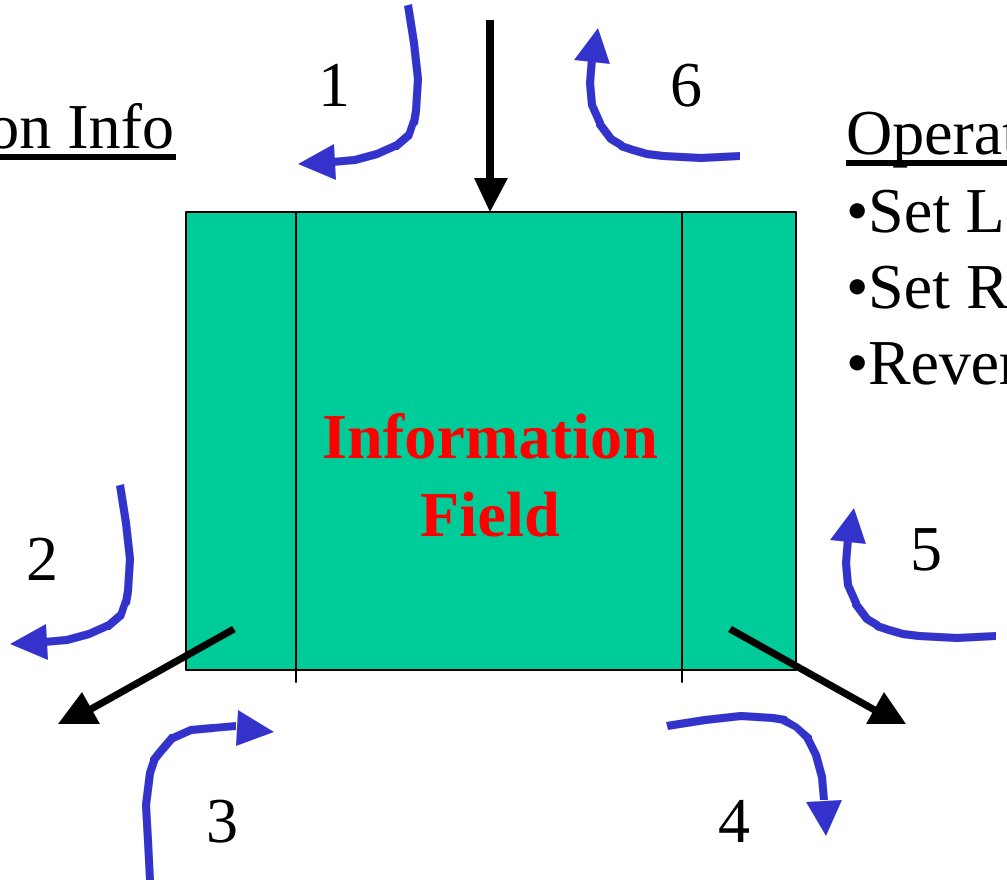


Visiting a Node

The Act of **Operating on the Node**.

<u>Operations on Info</u>

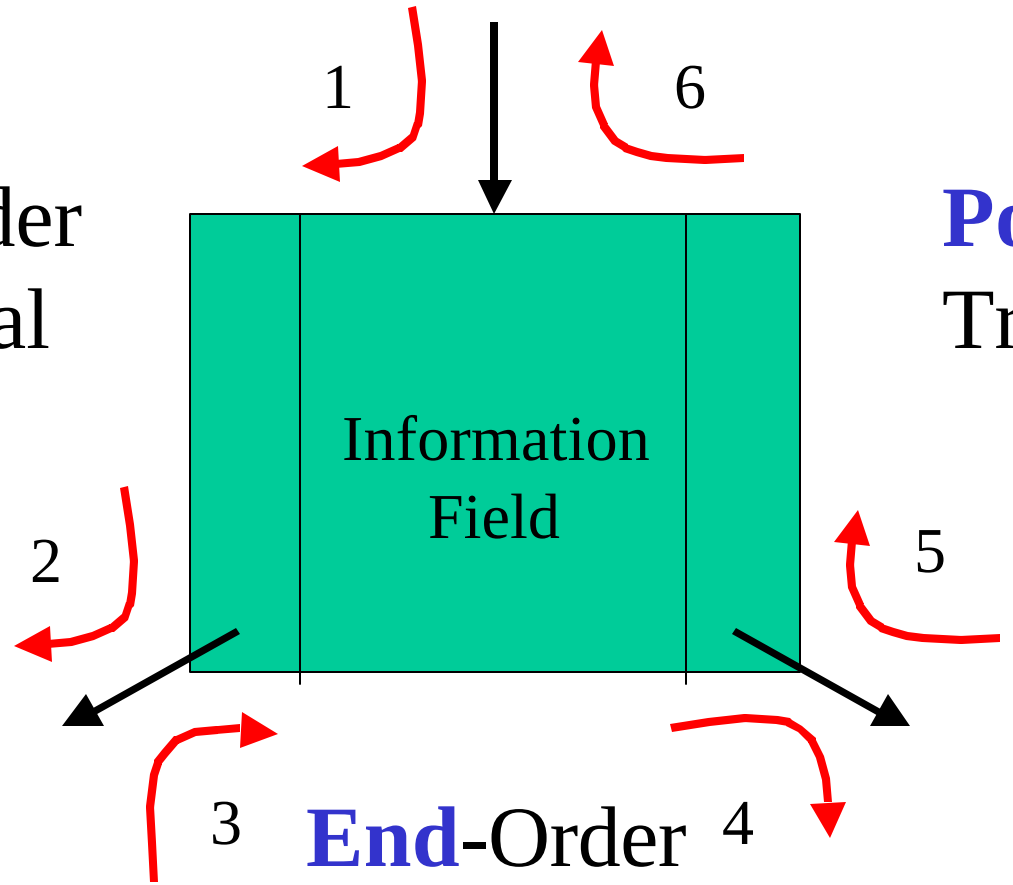
- Insert
- Copy
- Exchange
- Calculate



<u>Operations on Links</u>

- Set Left Link
- Set Right Link
- Reverse Links

Three Kinds of Traversals



Pre-Order Traversal

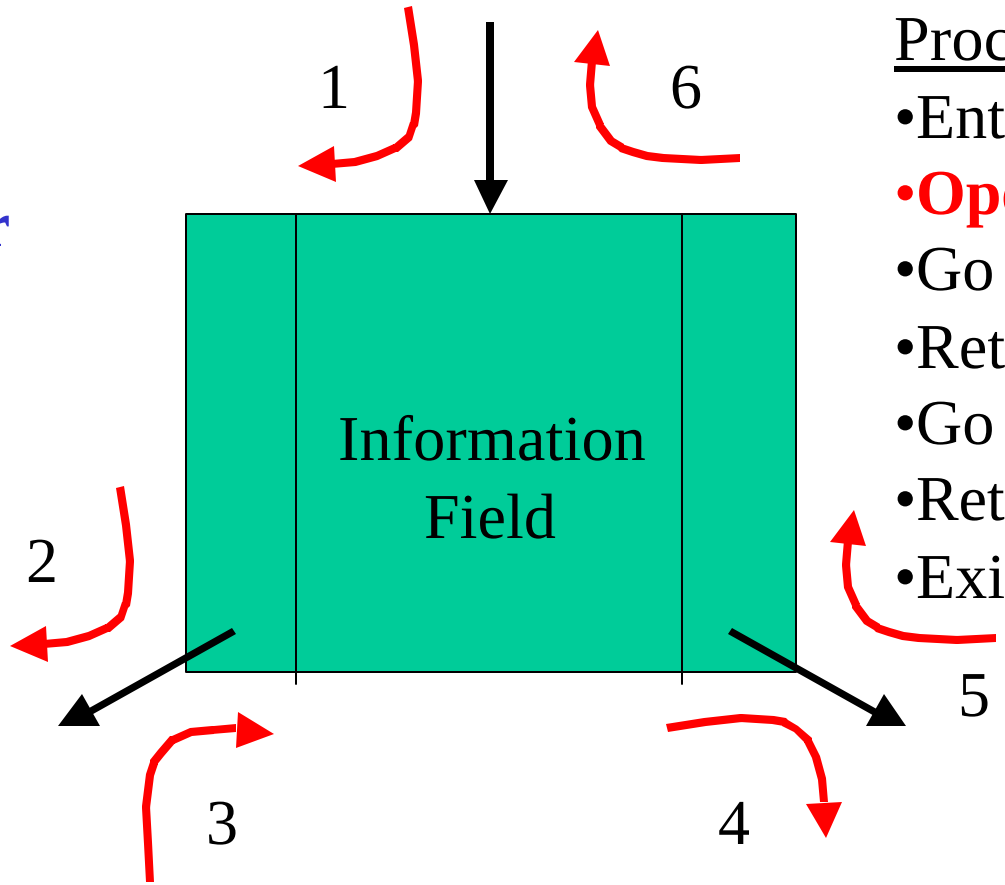
Post-Order Traversal

End-Order Traversal

Pre-Order Traversal

Operate on the Node between steps 1 and 2

Pre-Order Traversal

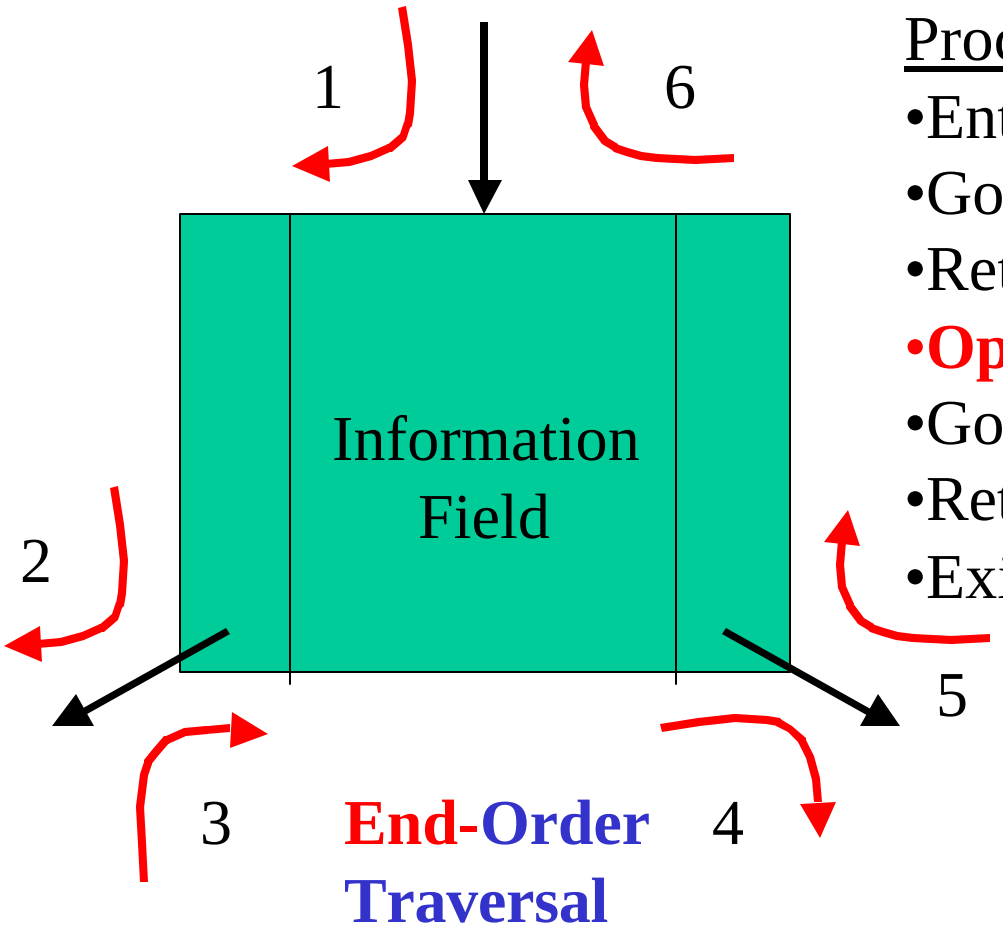


Procedure

- Enter node
- **Operate on Node**
- Go left
- Return from left
- Go right
- Return from right
- Exit node

End-Order Traversal

Operate on the Node between steps 3 and 4.



Procedure

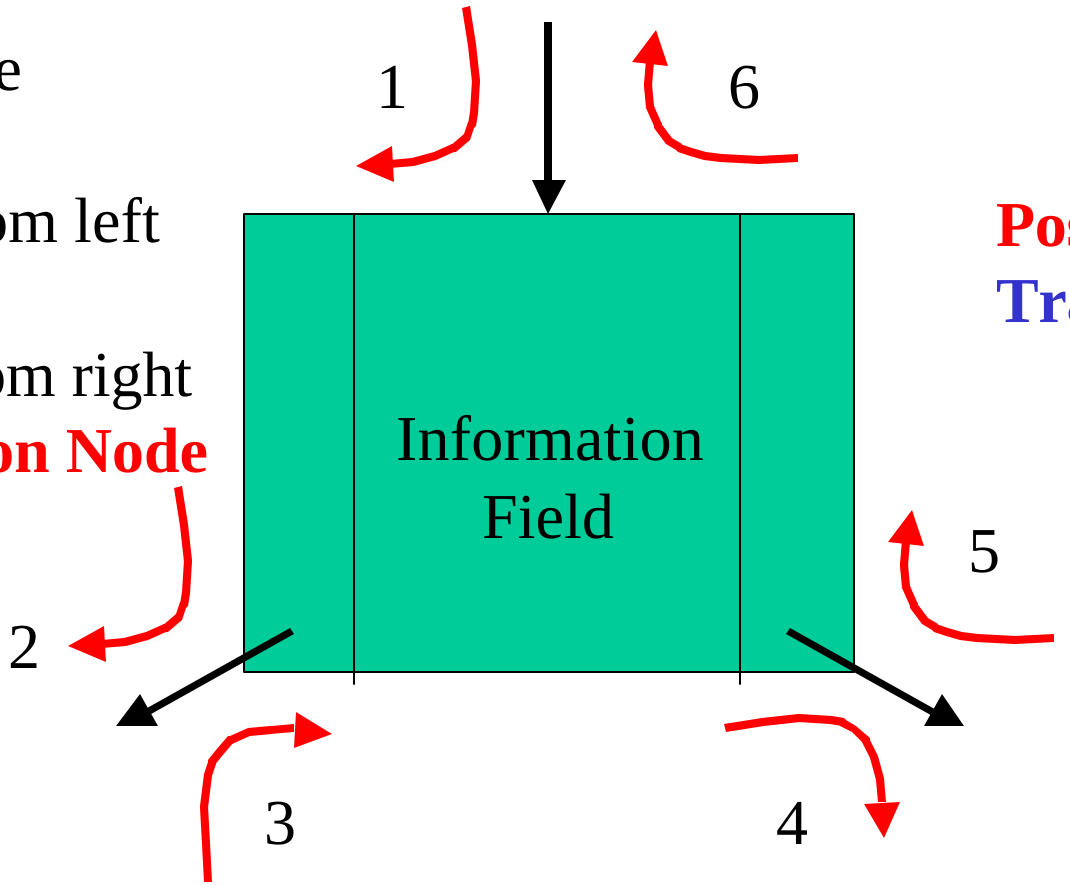
- Enter node
- Go left
- Return from left
- **Operate on Node**
- Go right
- Return from right
- Exit node

Post-Order Traversal

Operate on the Node between steps 5 and 6.

Procedure

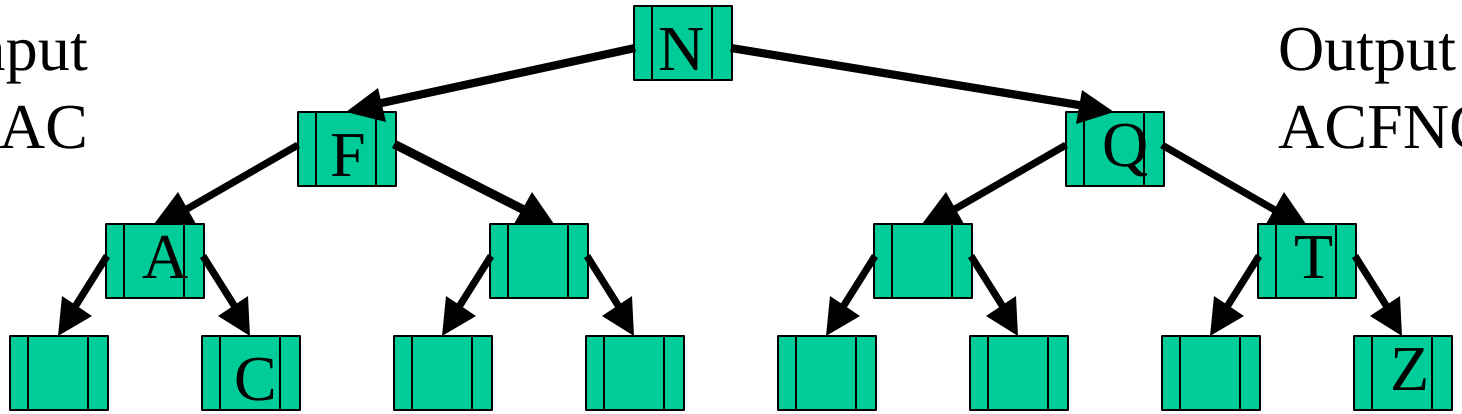
- Enter node
- Go left
- Return from left
- Go right
- Return from right
- **Operate on Node**
- Exit node



Post-Order Traversal

Binary Tree for Sorting

Input
NQFTZAC

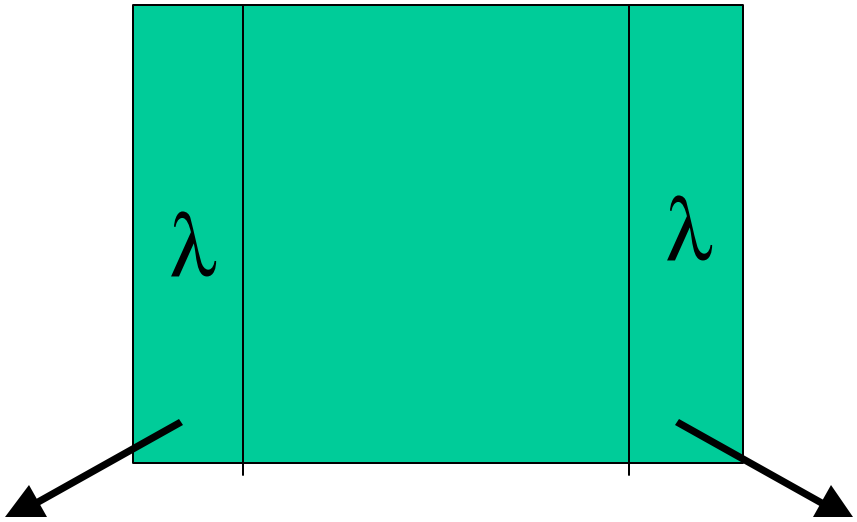


Output
ACFNQTZ

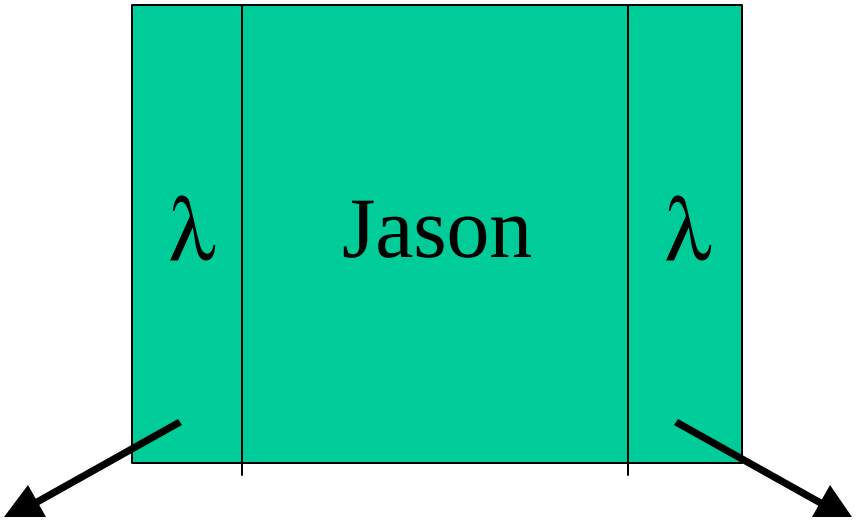
- The binary tree sort is an ***insertion*** sort.
- New data is entered into its final position immediately upon arrival of the data.
- The sorted list can be obtained at any time by ***visiting*** all nodes in the tree using the ***end-order traversal***.

If the Node is Empty Insert the Data

Jason

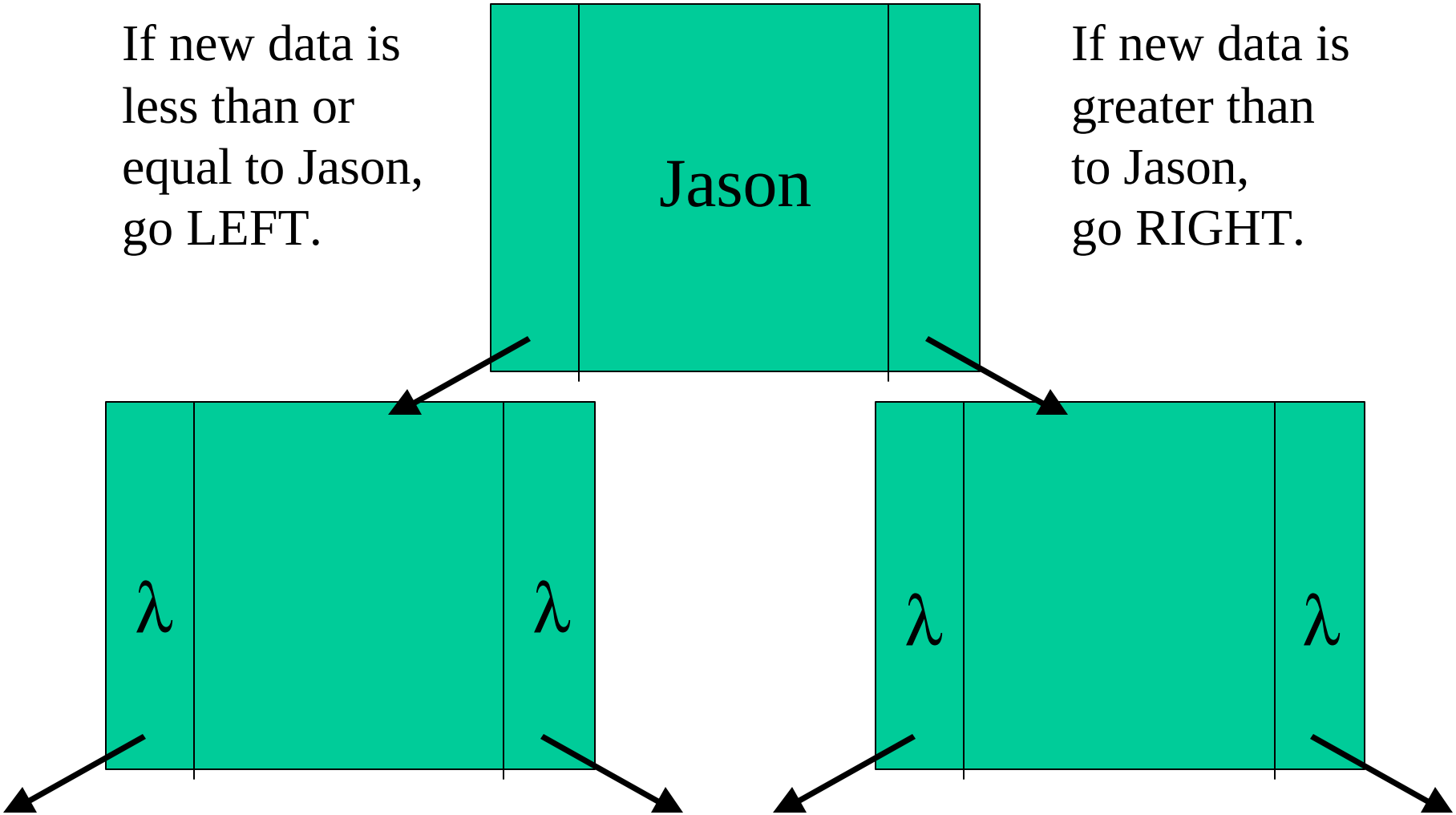


Data Inserted



If a Node Already has Data

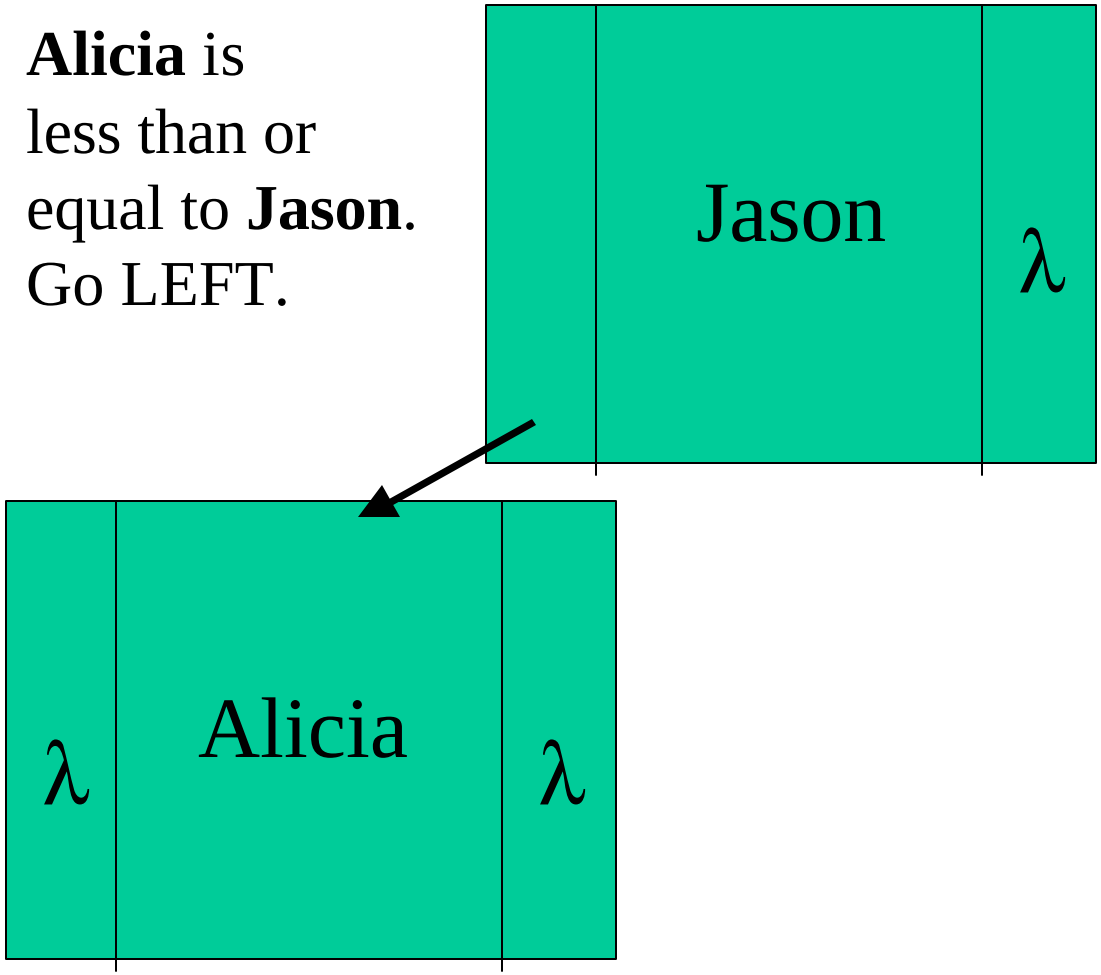
If new data is less than or equal to Jason, go LEFT.



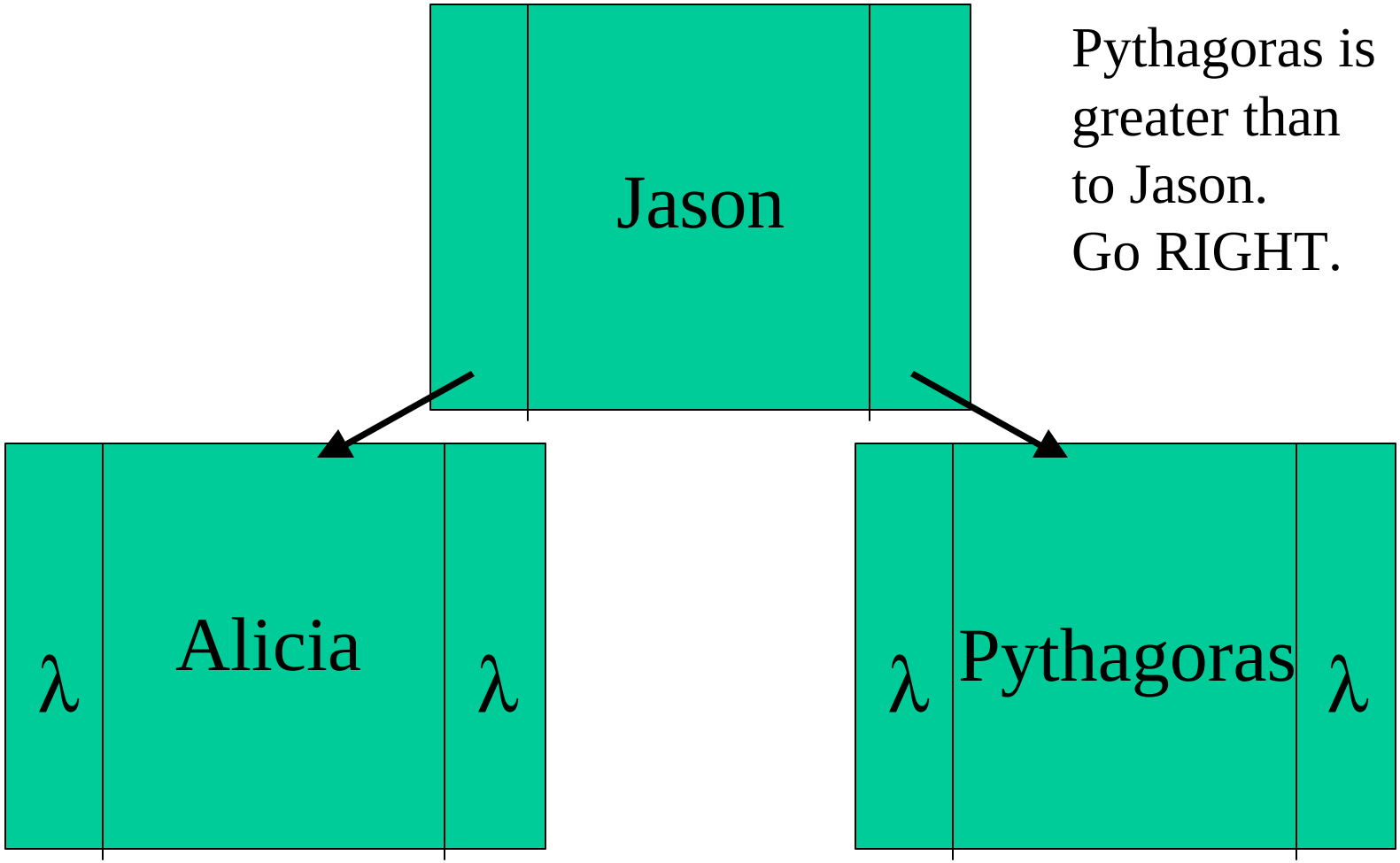
If new data is greater than to Jason, go RIGHT.

Insert Alicia

Alicia is
less than or
equal to **Jason**.
Go LEFT.

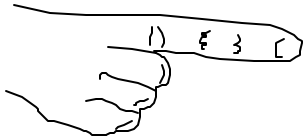


Insert Pythagoras



New Data: Jackson

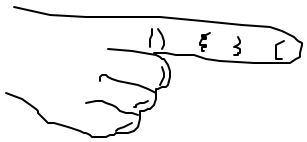
Compare
Jackson
to Jason.

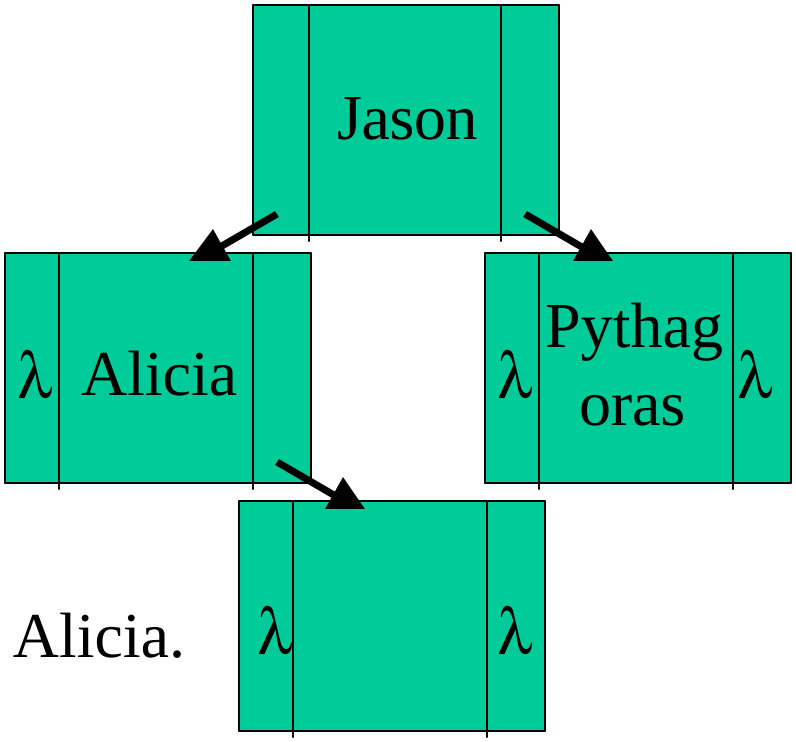


Jackson is
less than or equal to
Jason. Go Left.

Jackson (Continued)

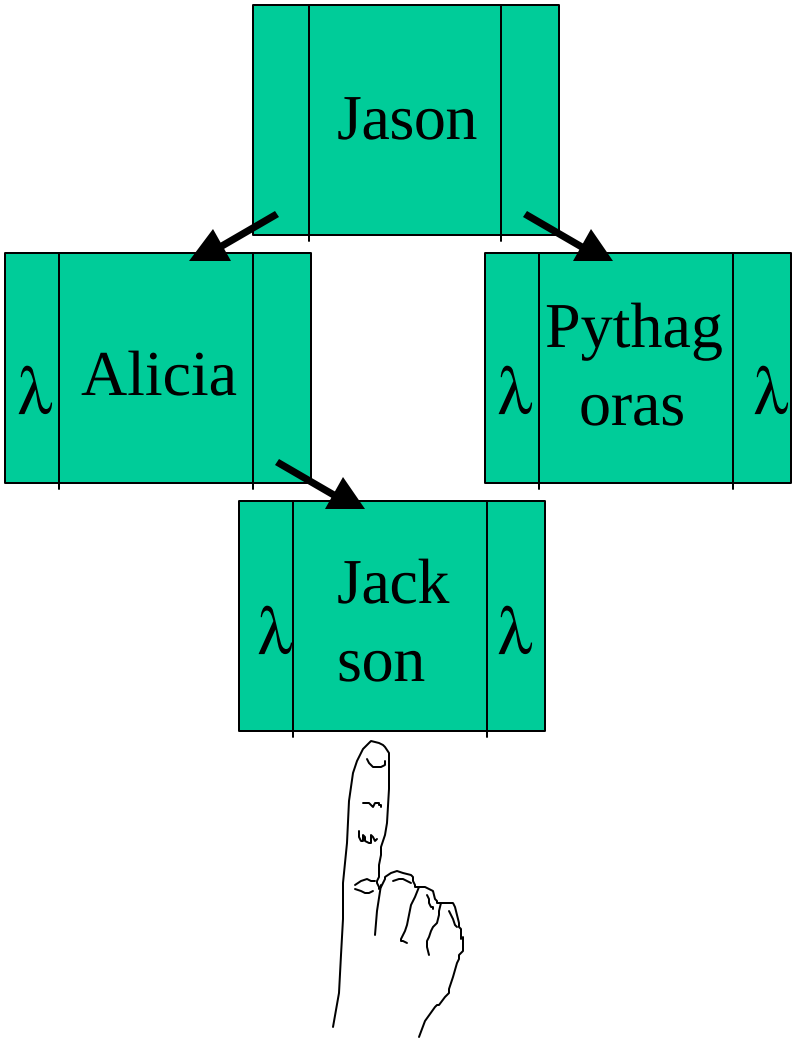
Compare
Jackson
to Alicia.





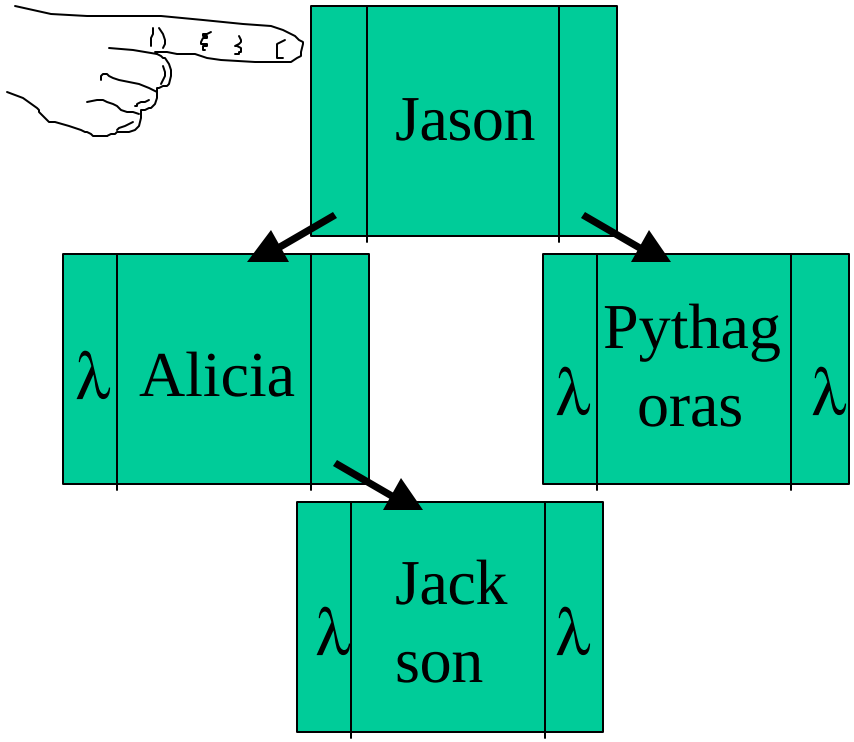
Jackson is
greater than Alicia.
Go Right.

Insert Jackson



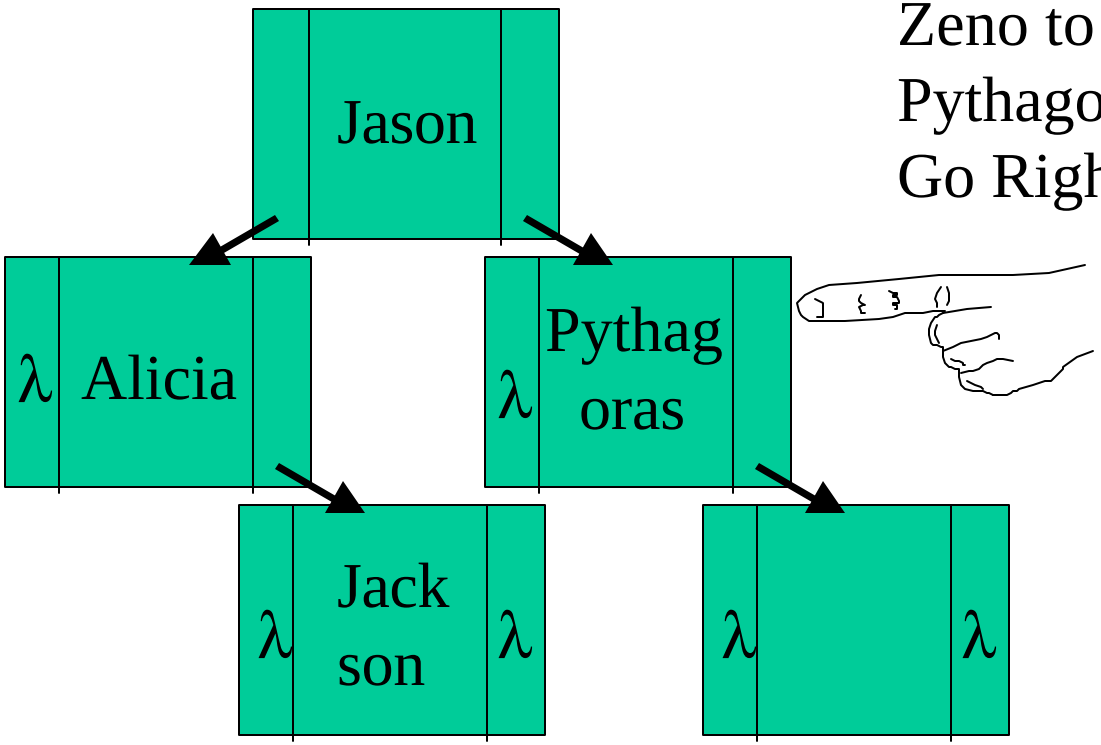
New Data: Zeno

Compare Zeno to Jason.
Go Right.

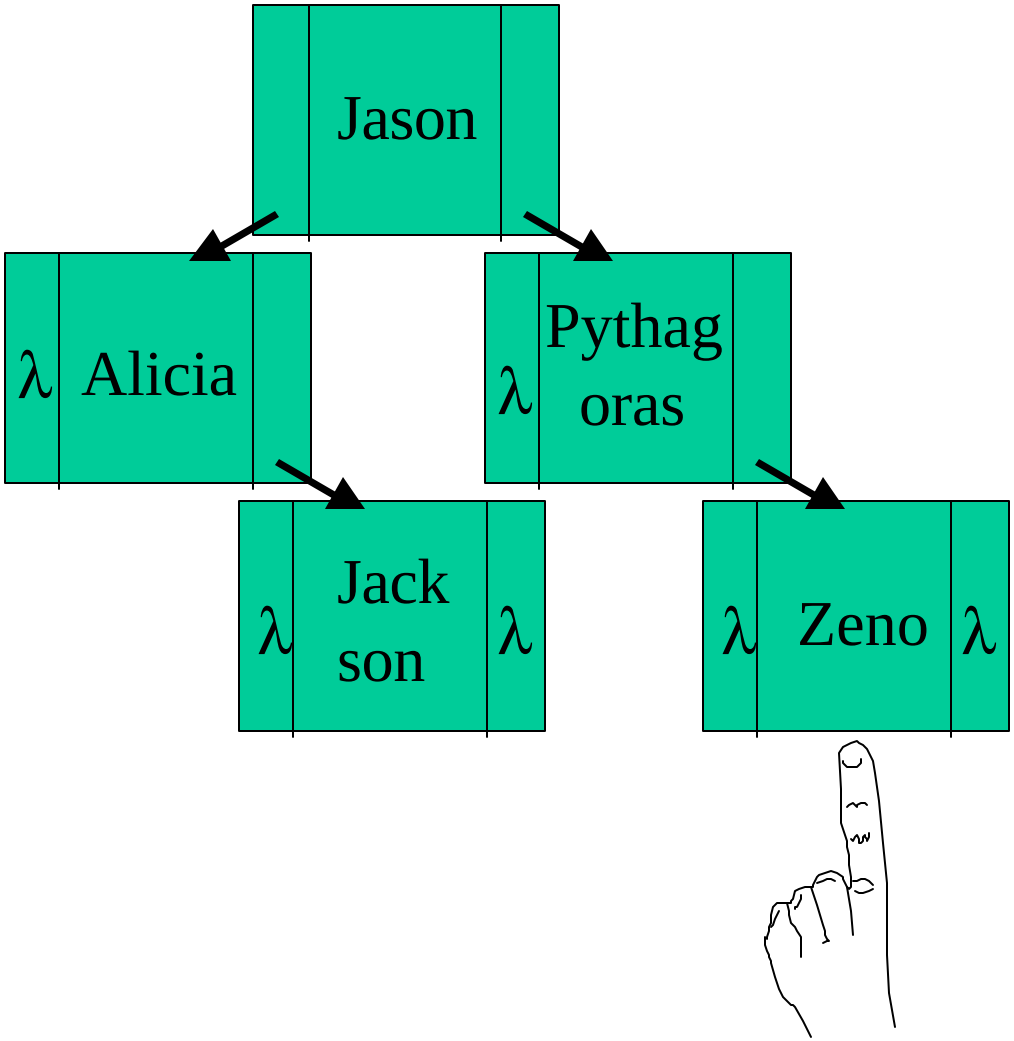


Zeno (Continued)

Compare Zeno to Pythagoras. Go Right.



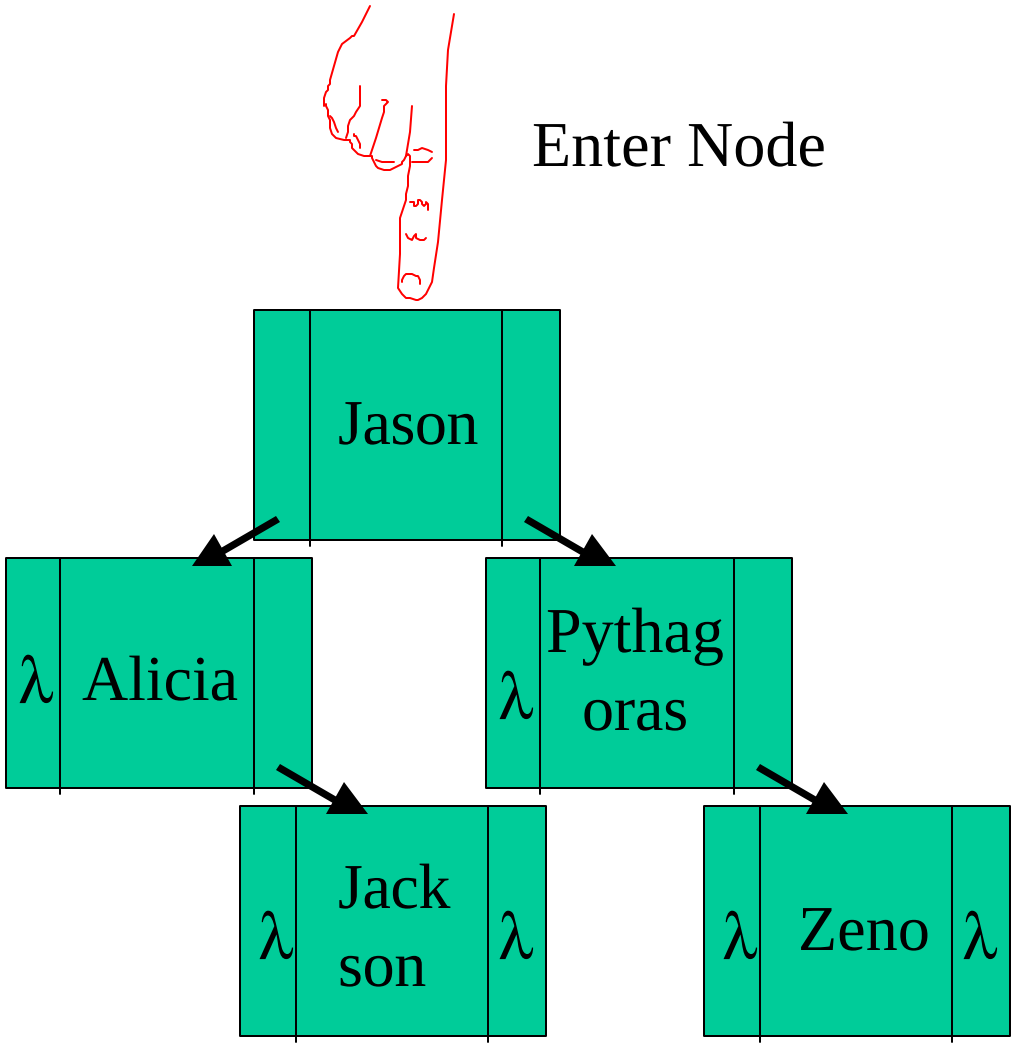
Insert Zeno



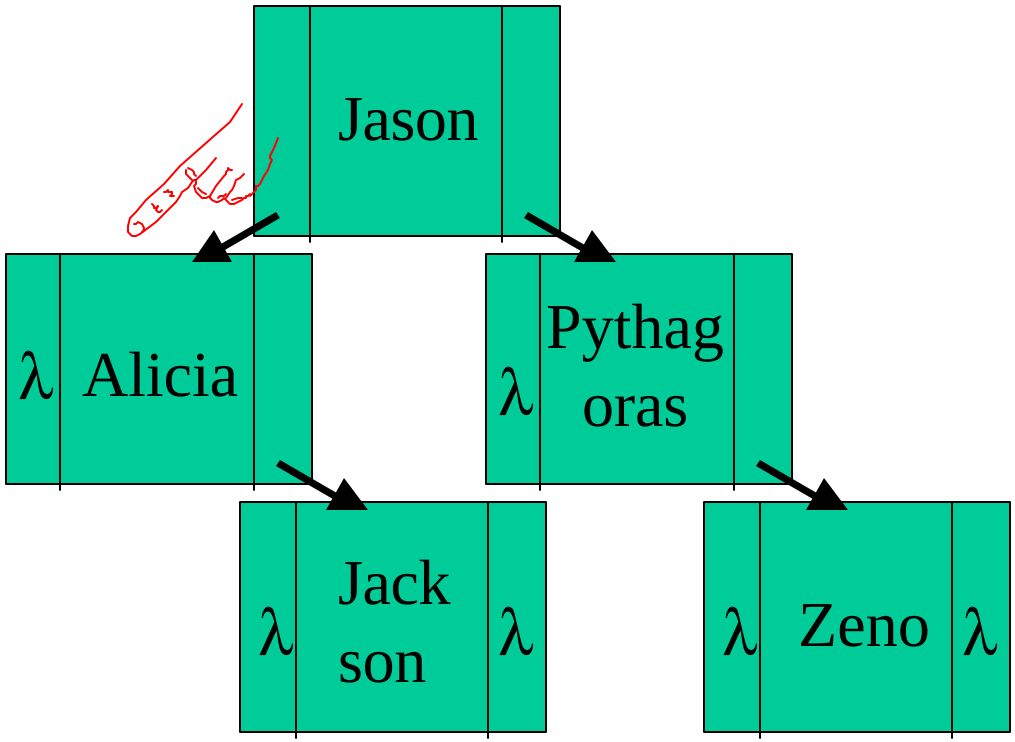
Creating the Sorted List

- Do an End-Order Traversal
- The Operation on the Node is: append a copy of the information field to the end of the sorted list.

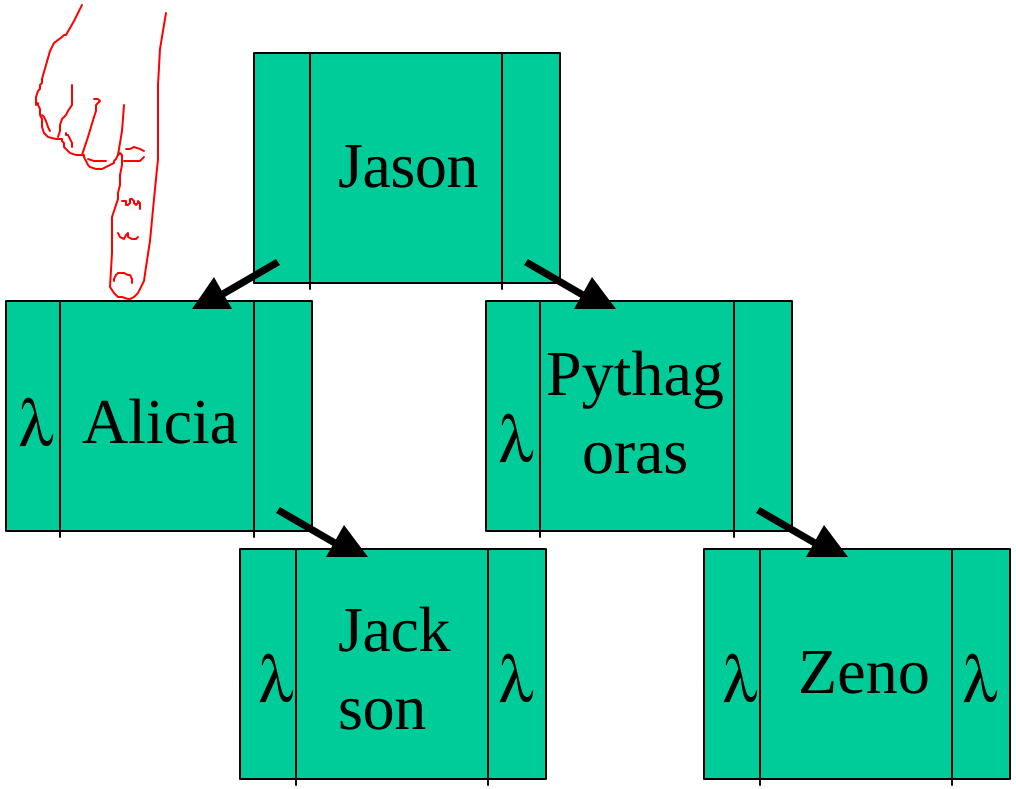
Begin End-Order Traversal



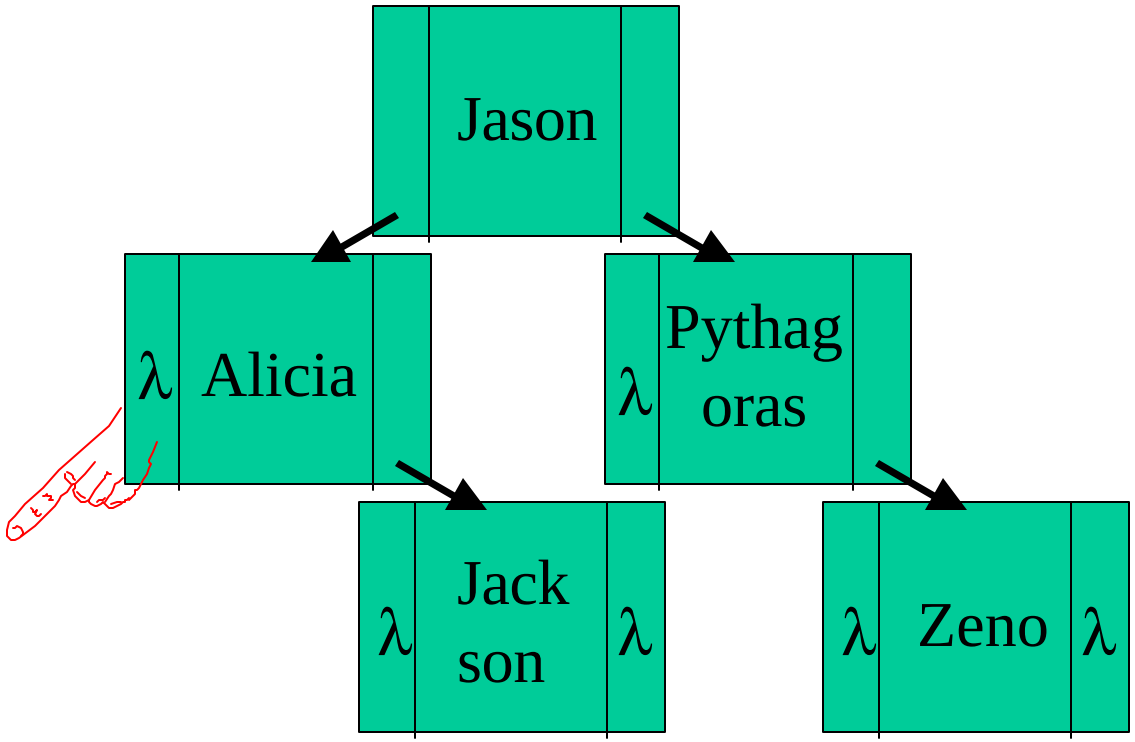
Go Left from Jason



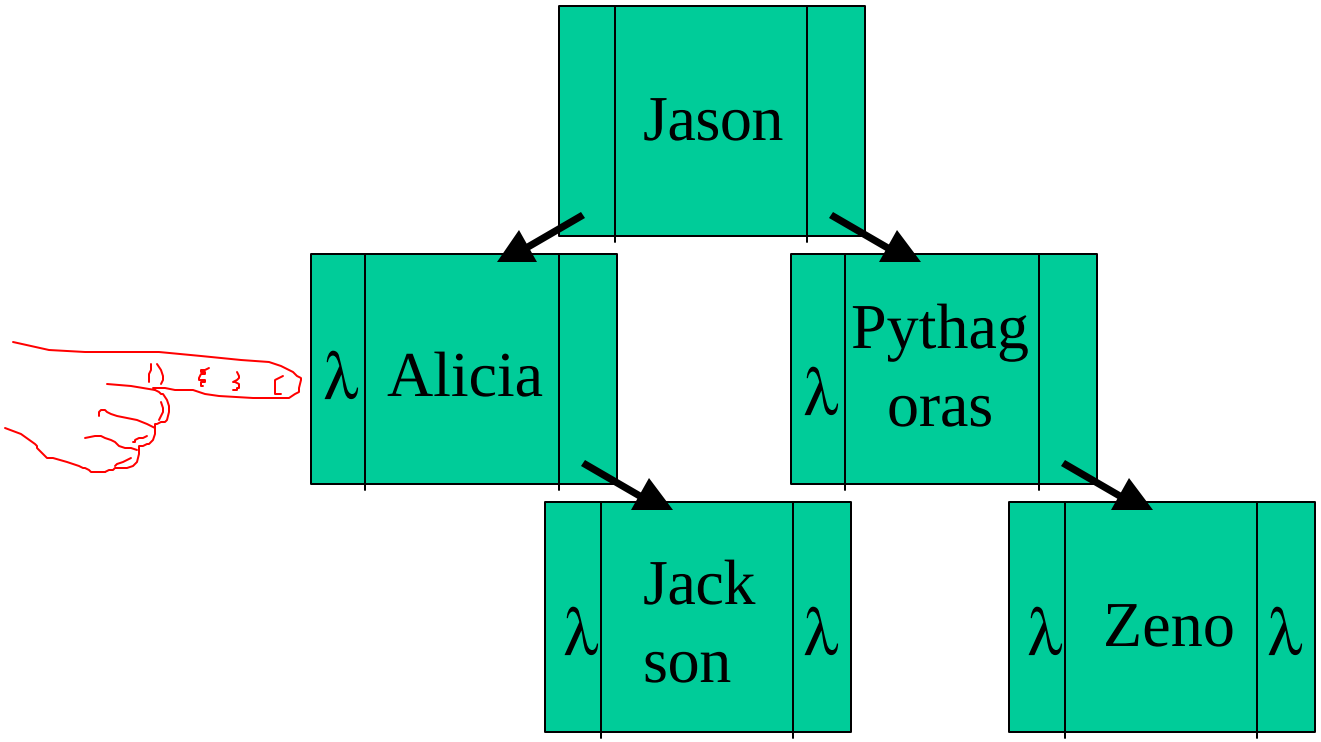
Enter Node Alicia



Go Left from Alicia

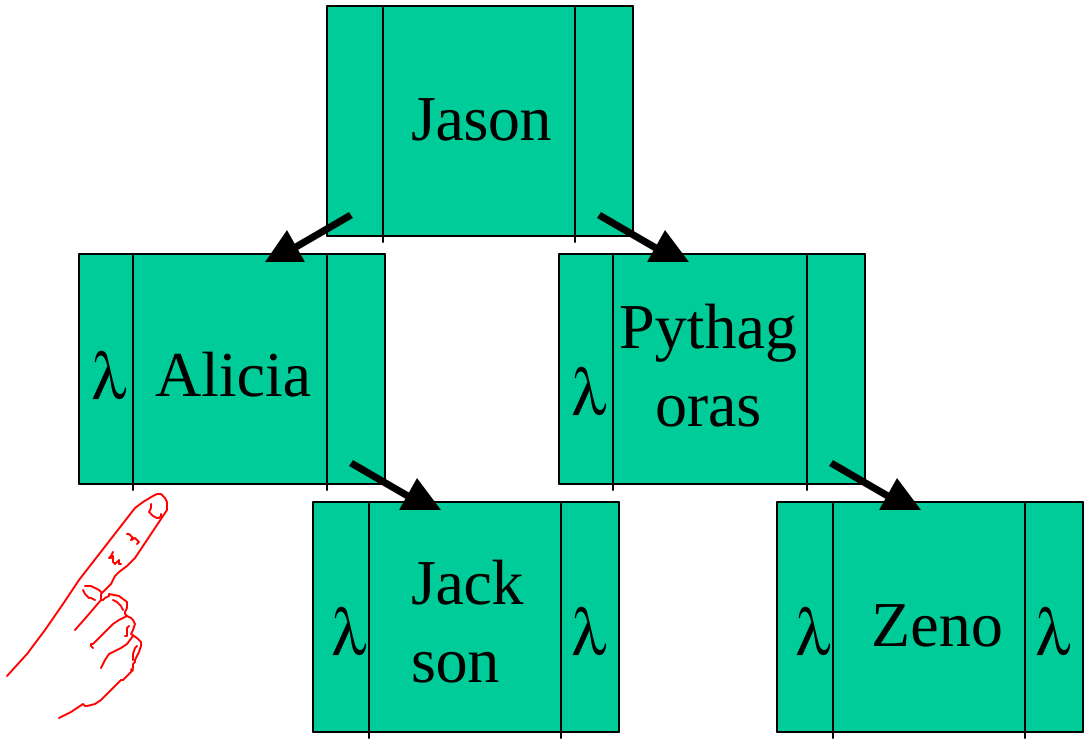


Alicia Left Link is Null



Since the link is null, return to Alicia immediately.

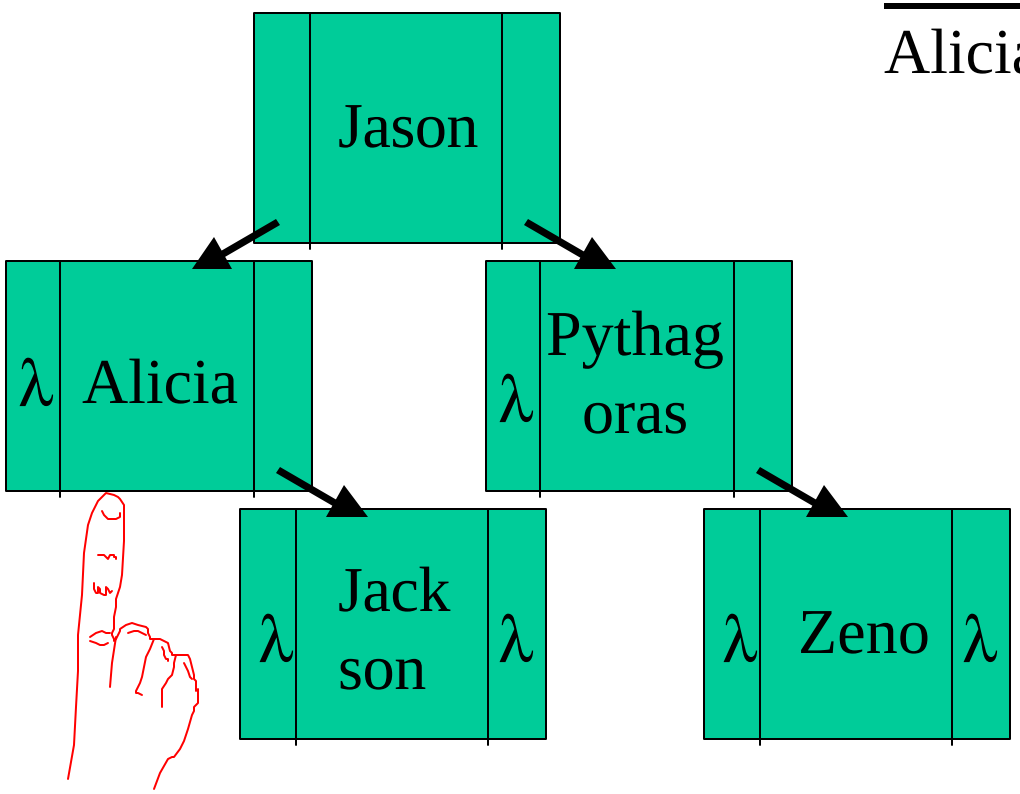
Return to Alicia from Left Link



Since the link is null, we return to Alicia immediately.

Visit Alicia

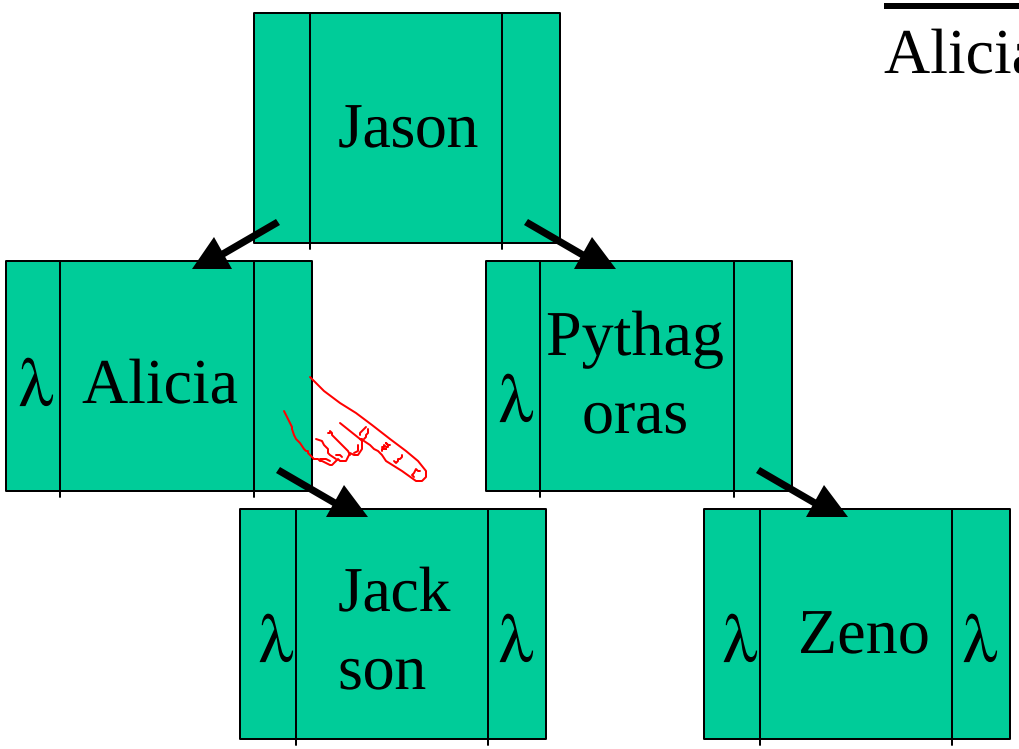
Sorted Names
Alicia



Copy Alicia to the list of sorted names.

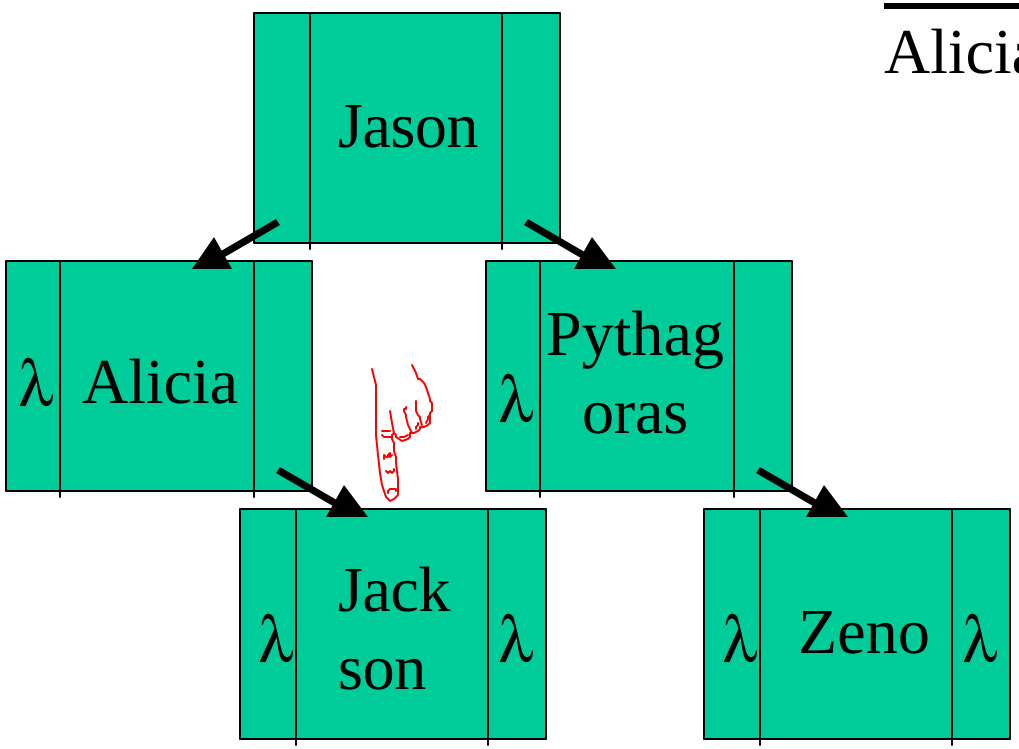
Go Right from Alicia

<u>Sorted Names</u>
Alicia



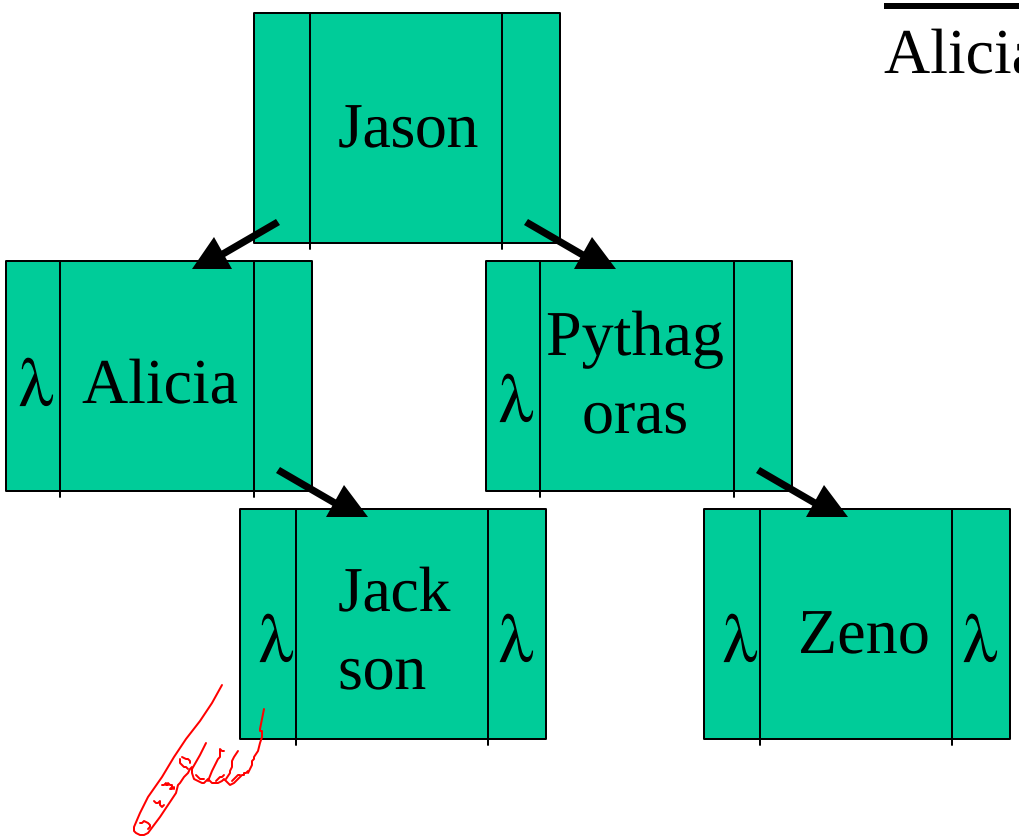
Enter Node Jackson

<u>Sorted Names</u>
Alicia



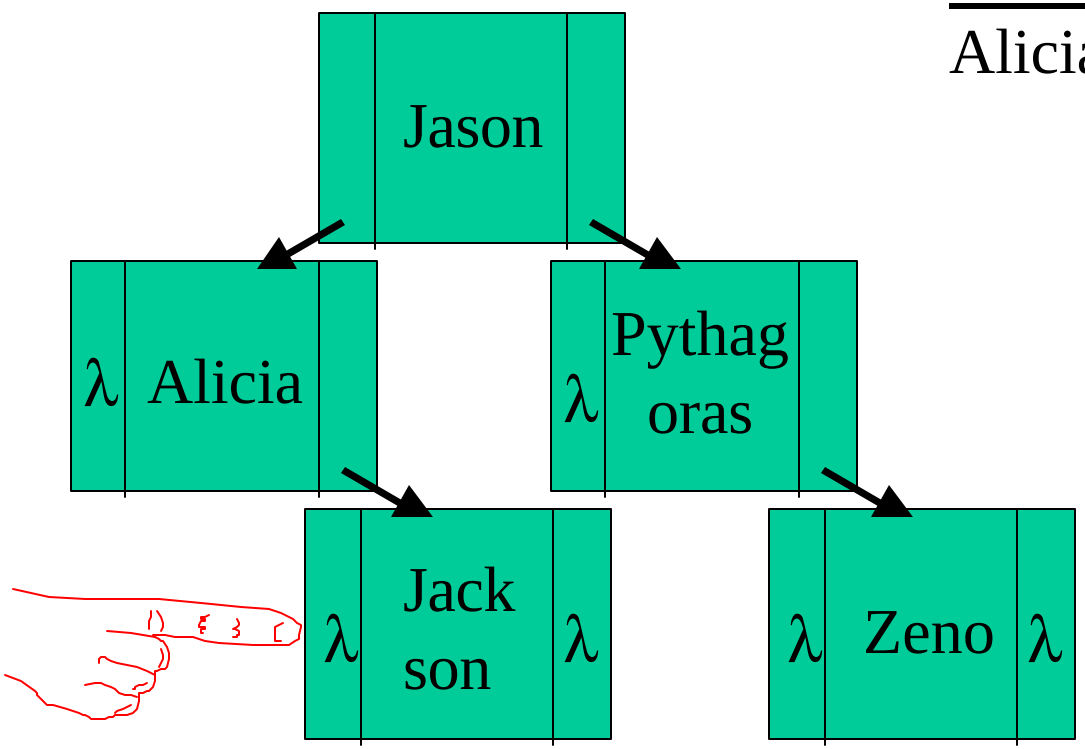
Go Left from Jackson

<u>Sorted Names</u>
Alicia



Jackson Left Link is Null

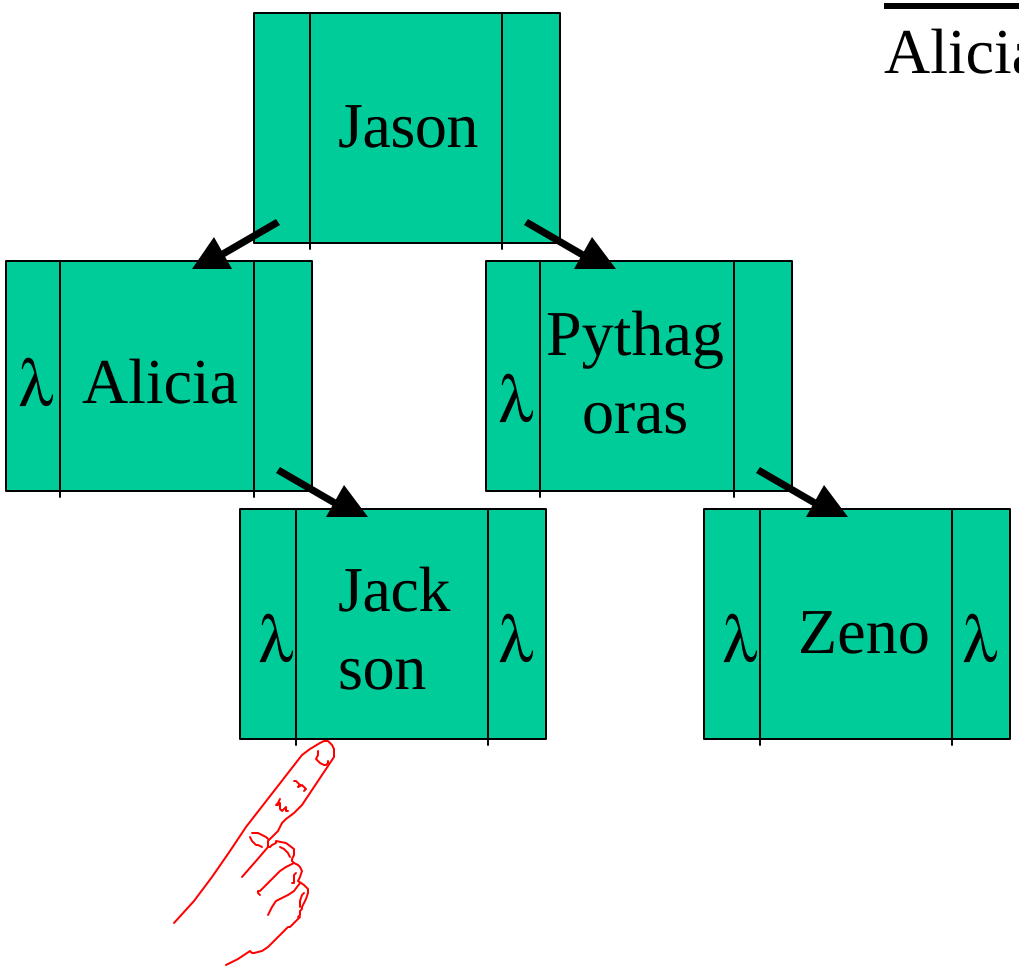
Sorted Names
Alicia



Since the left link is null, return to Jackson immediately.

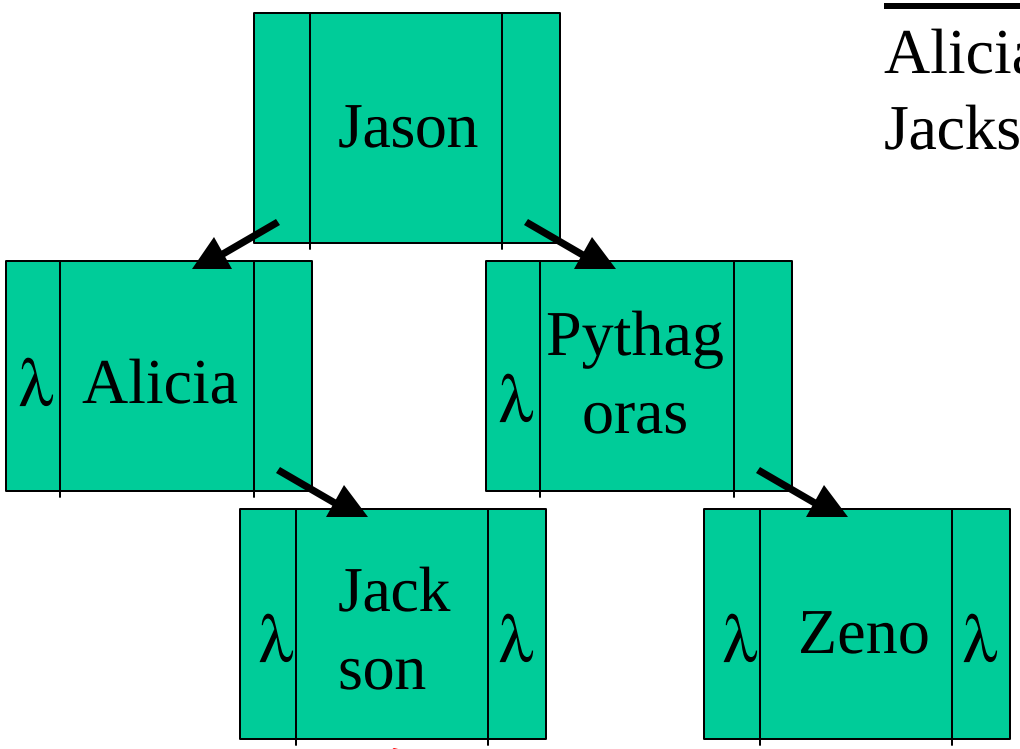
Return to Jackson from Left Link

<u>Sorted Names</u>
Alicia



Visit Jackson

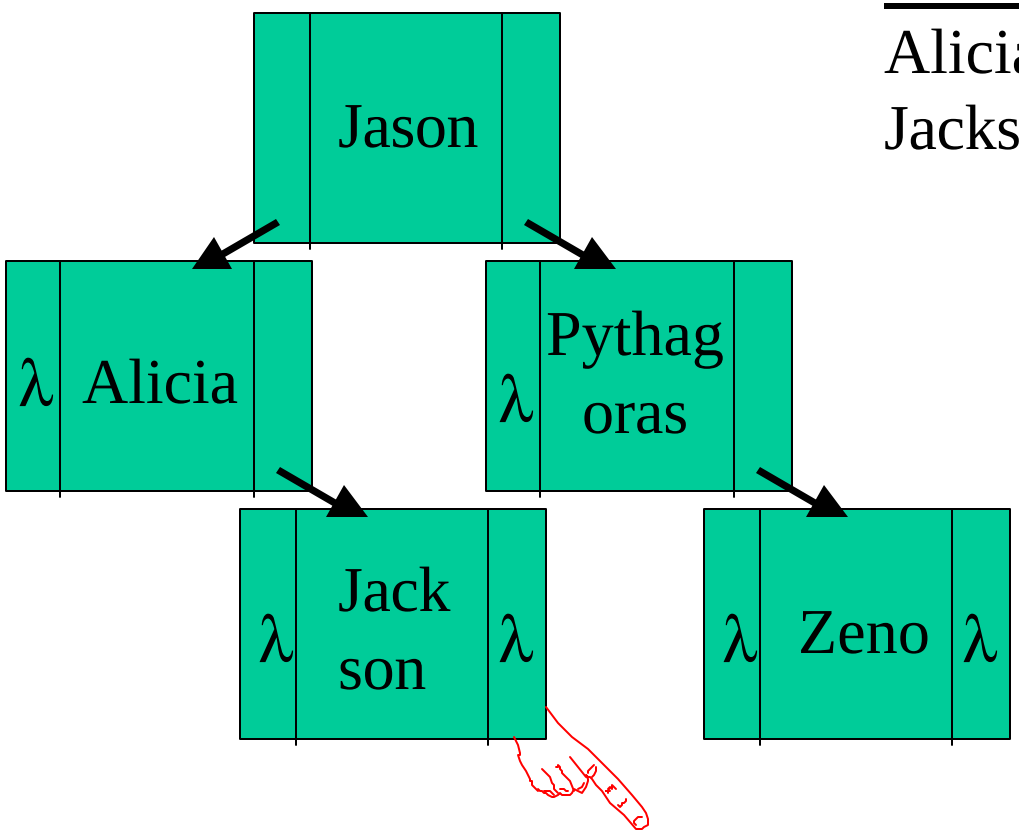
<u>Sorted Names</u>
Alicia
Jackson



Copy Jackson to the list of sorted names.

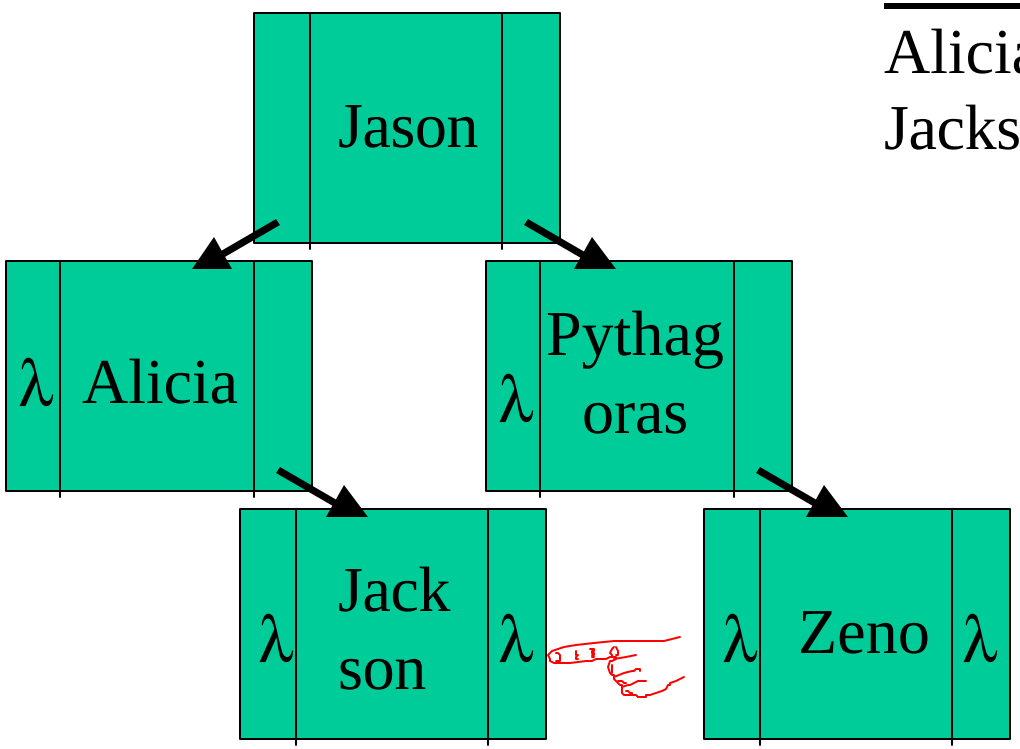
Go Right from Jackson

<u>Sorted Names</u>
Alicia
Jackson



Jackson Right Link is Null

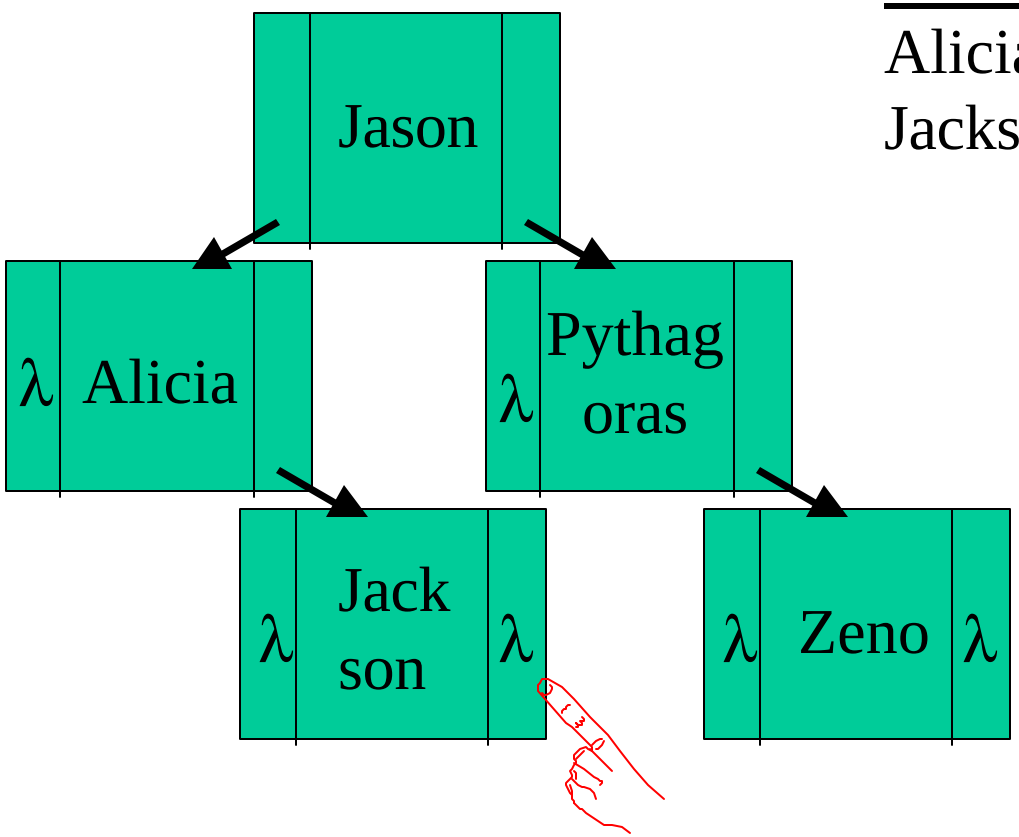
<u>Sorted Names</u>
Alicia
Jackson



Since the right link is null, return to Jackson immediately.

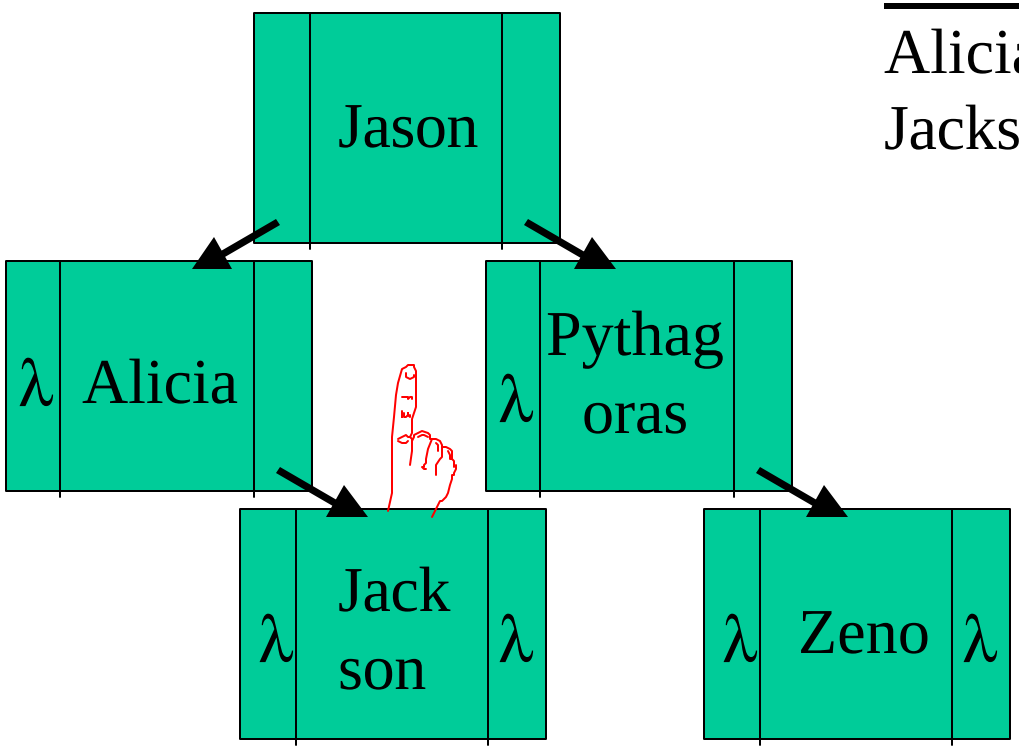
Return to Jackson from Right Link

<u>Sorted Names</u>
Alicia
Jackson



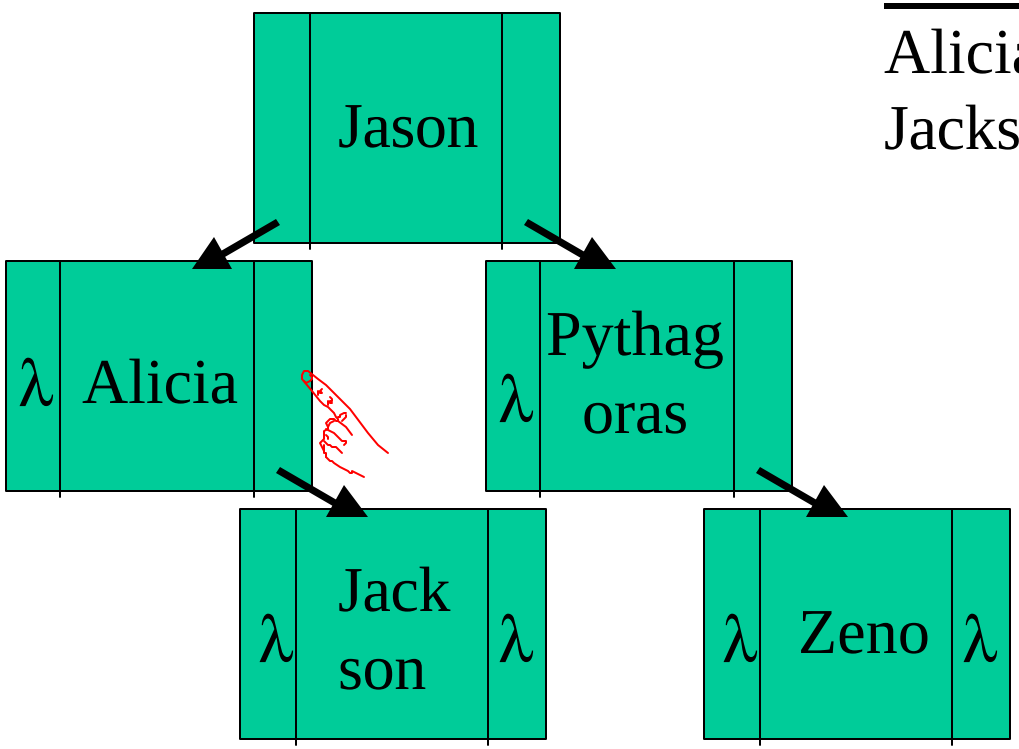
Leave Jackson

<u>Sorted Names</u>
Alicia
Jackson

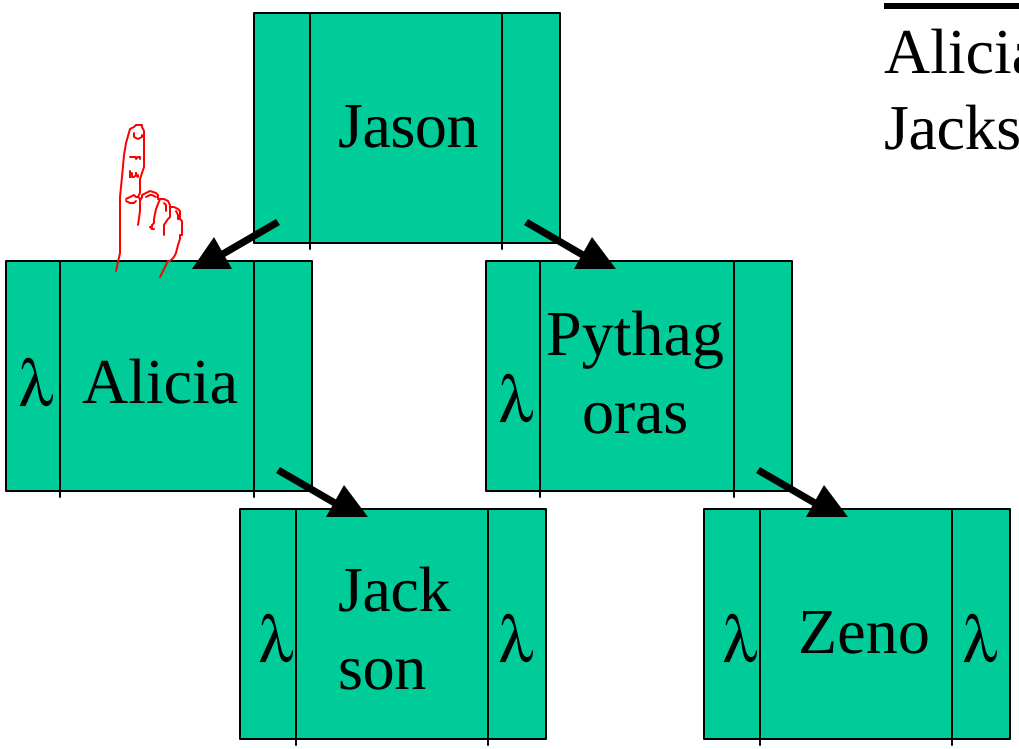


Return to Alicia from Right Link

<u>Sorted Names</u>
Alicia
Jackson



Leave Alicia



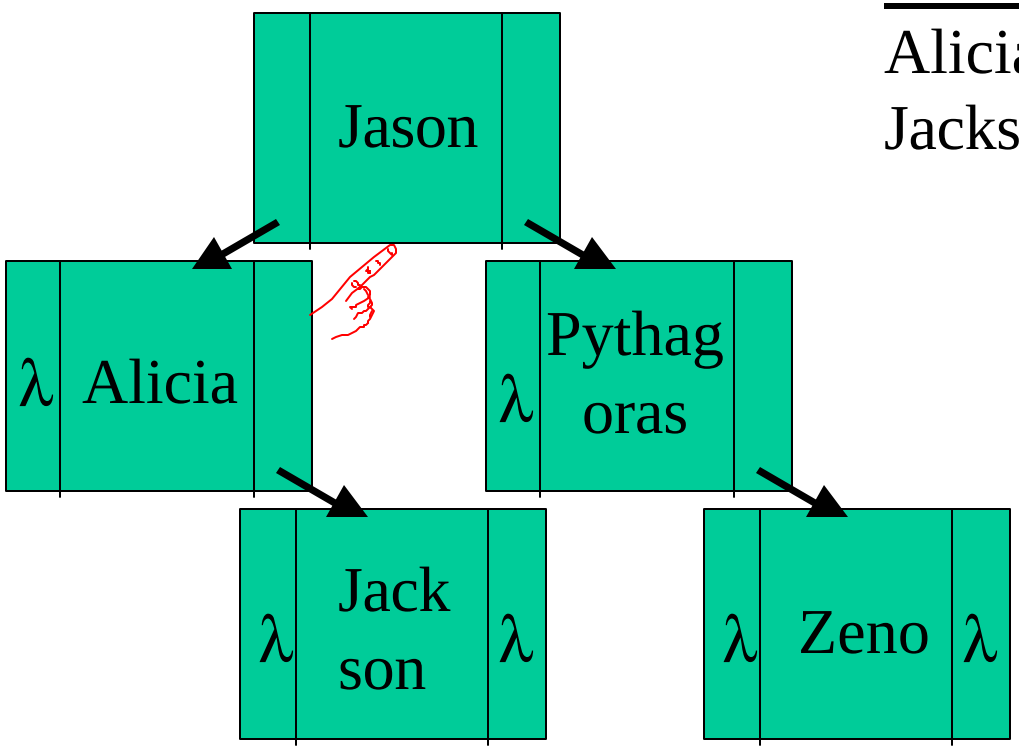
Sorted Names

Alicia

Jackson

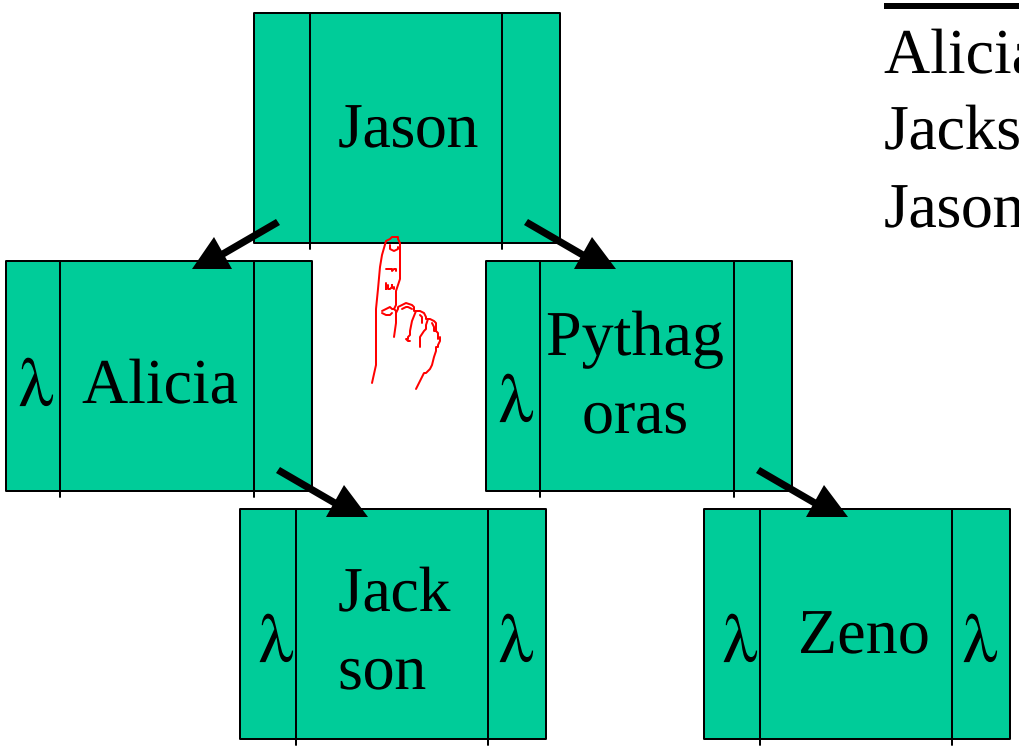
Return to Jason from Left Link

<u>Sorted Names</u>
Alicia
Jackson



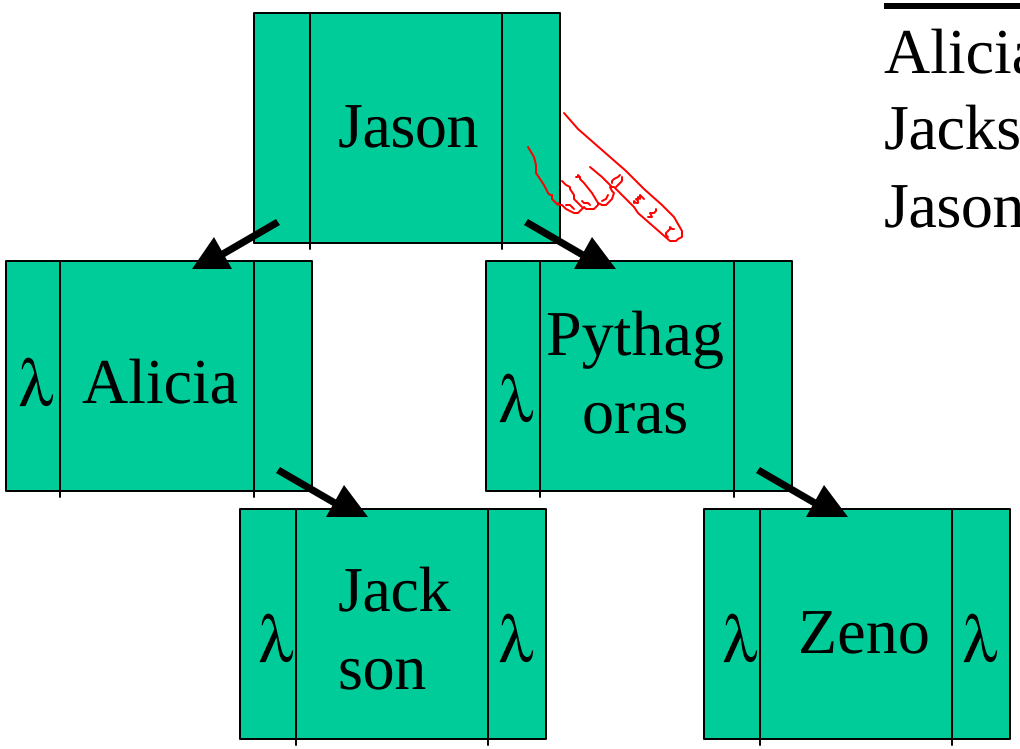
Visit Jason

<u>Sorted Names</u>
Alicia
Jackson
Jason



Copy Jason to the list of sorted names.

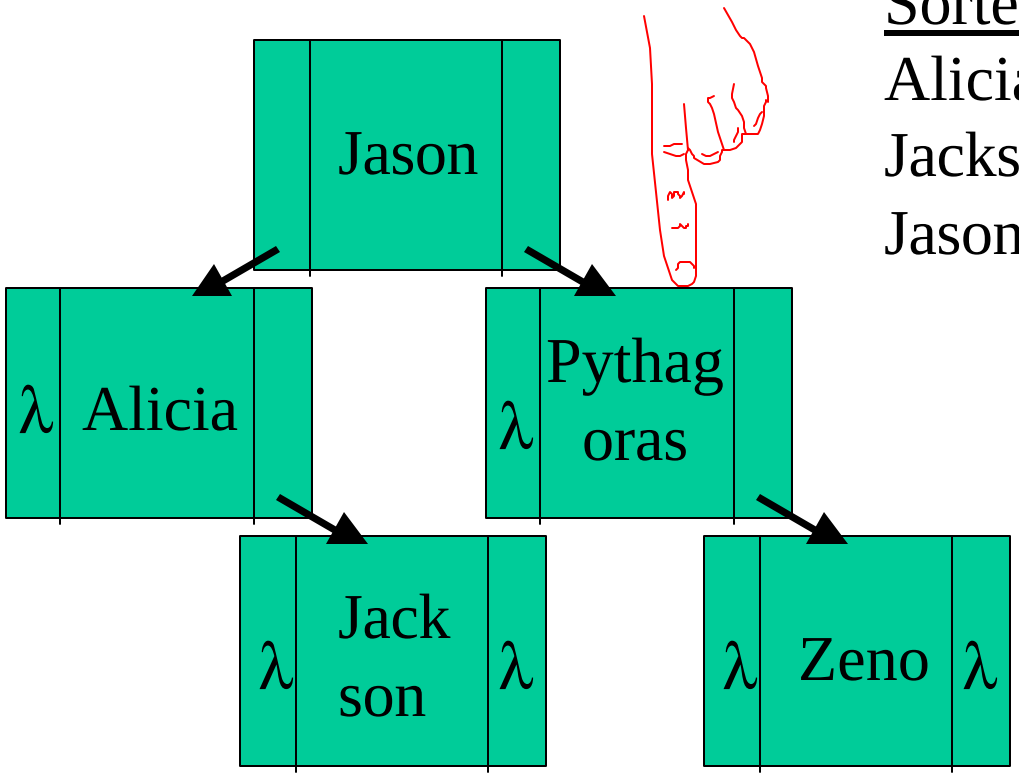
Go Right from Jason



Sorted Names
Alicia
Jackson
Jason

Copy Jason to the list of sorted names.

Enter Node Pythagoras



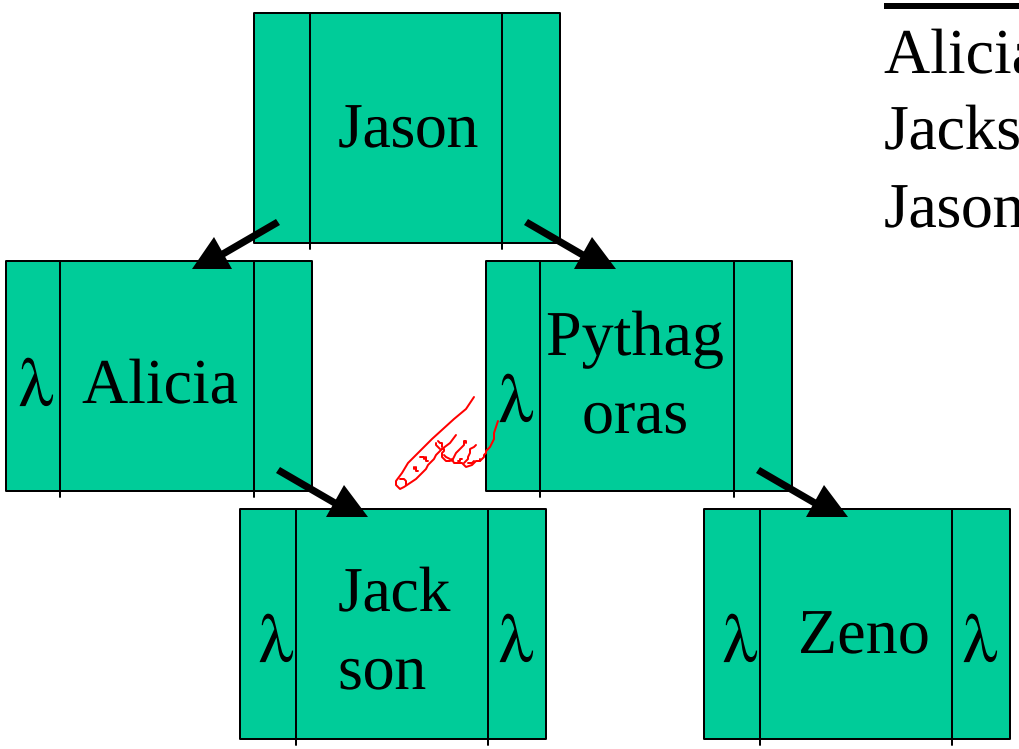
Sorted Names

Alicia

Jackson

Jason

Go Left from Pythagoras



Sorted Names

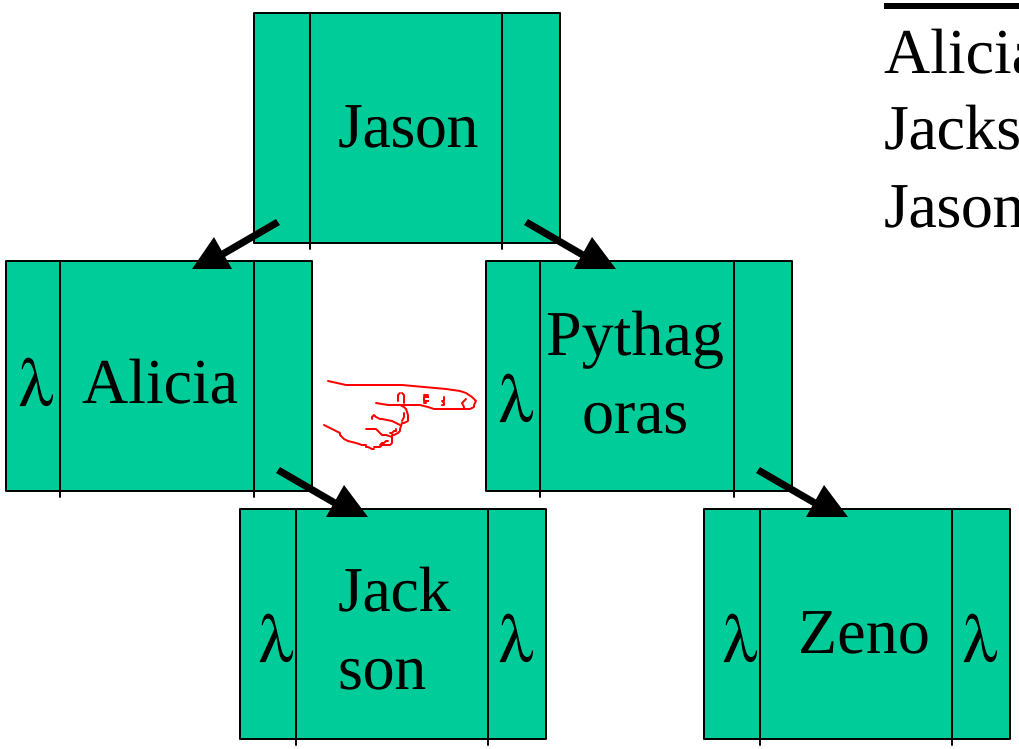
Alicia

Jackson

Jason

Pythagoras Left Link is Null

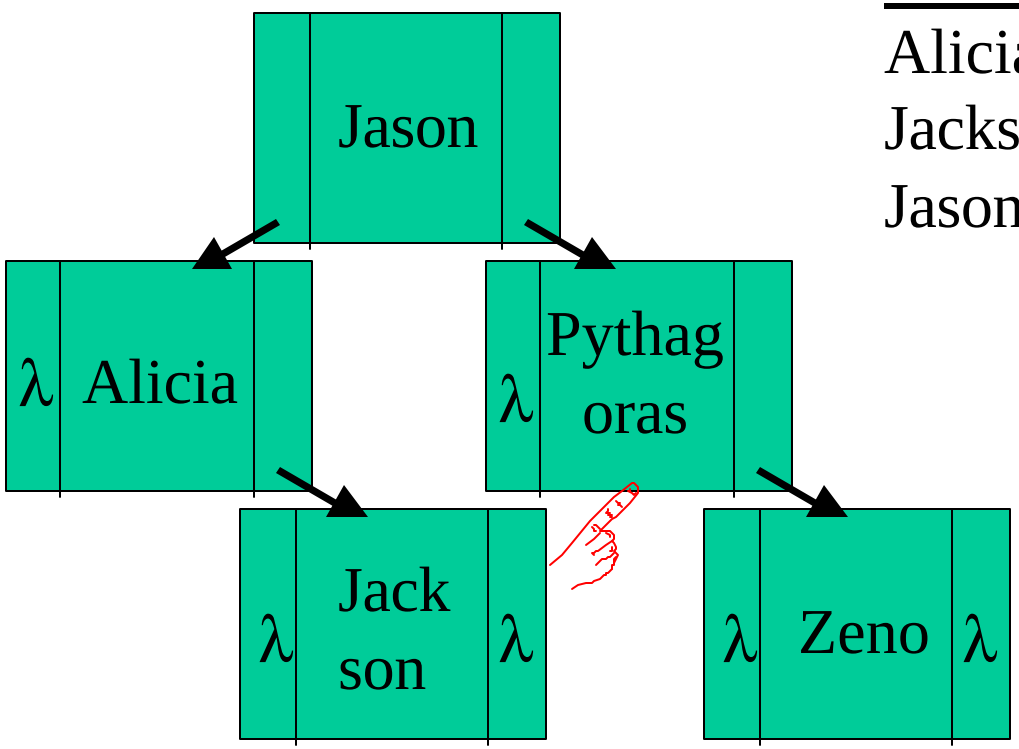
<u>Sorted Names</u>
Alicia
Jackson
Jason



Since the left link is null, return to Pythagoras immediately.

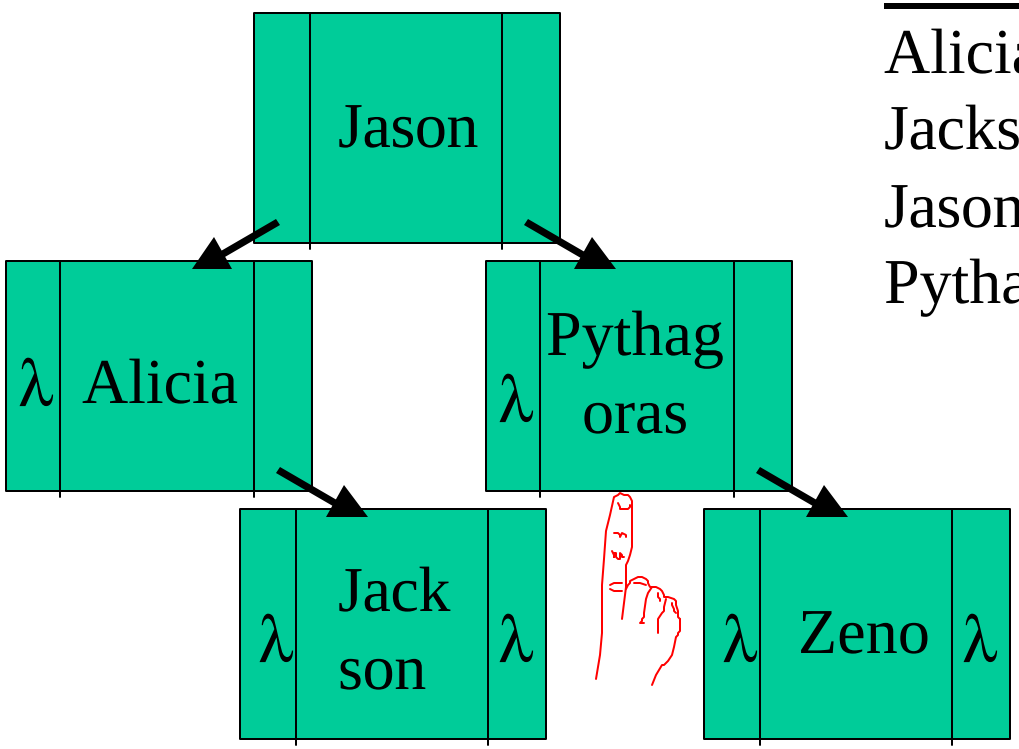
Return to Pythagoras from Left Link

<u>Sorted Names</u>
Alicia
Jackson
Jason



Since the left link is null, return to Pythagoras immediately.

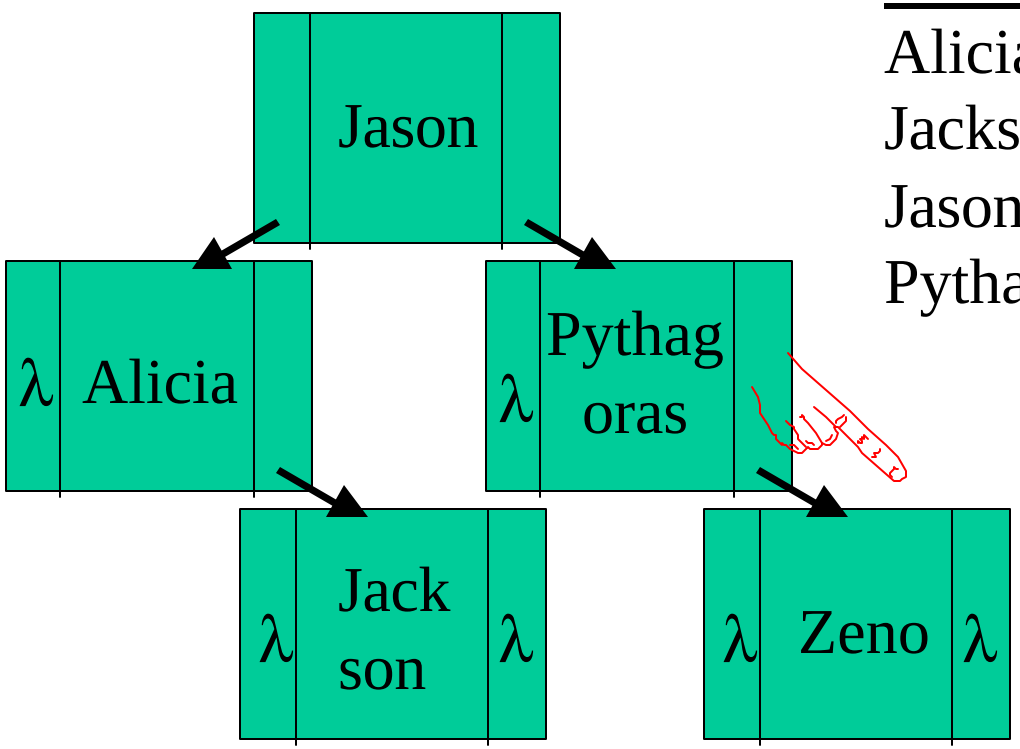
Visit Pythagoras



Sorted Names
Alicia
Jackson
Jason
Pythagoras

Copy Pythagoras to the list of sorted names.

Go Right from Pythagoras

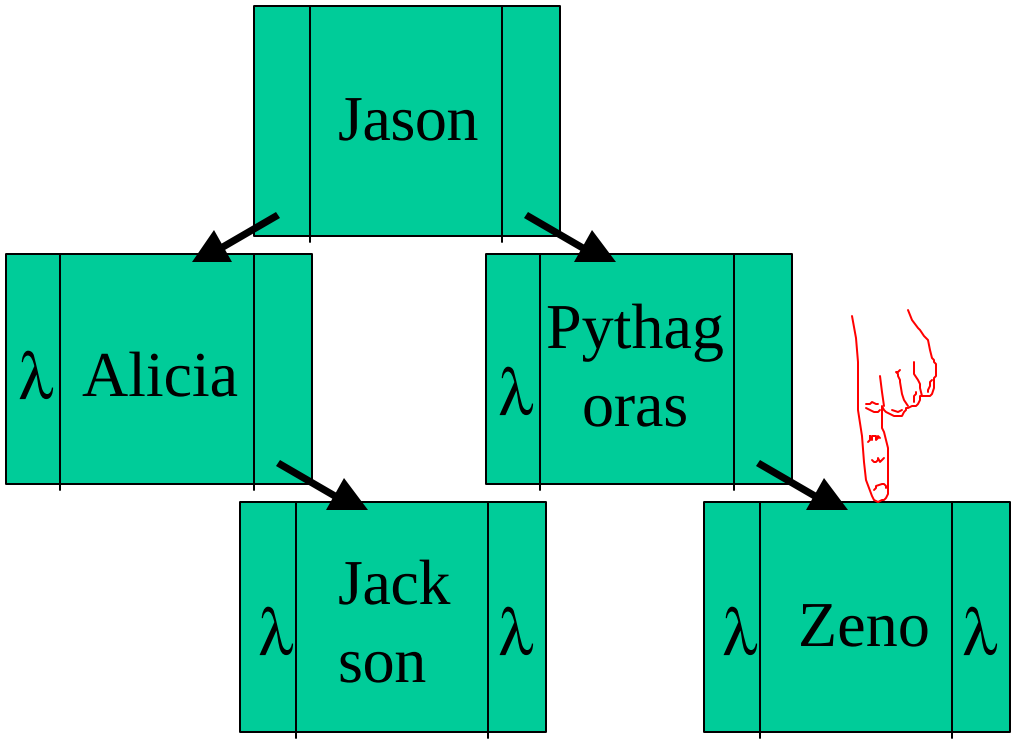


Sorted Names
Alicia
Jackson
Jason
Pythagoras

Copy Pythagoras to the list of sorted names.

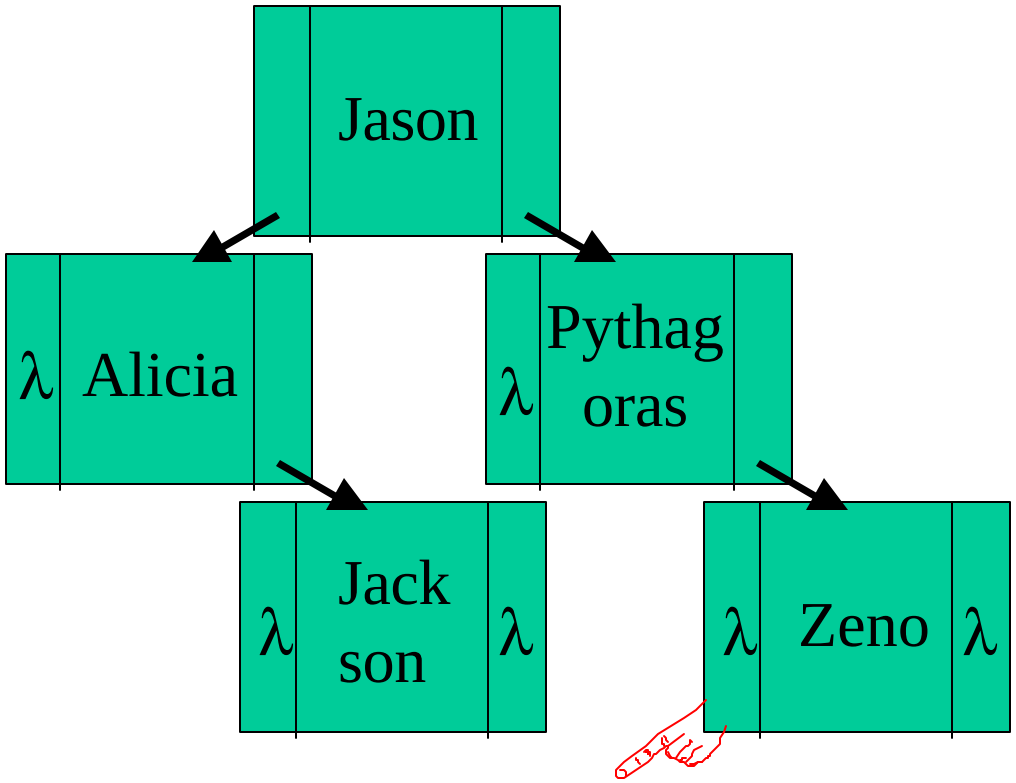
Enter Zeno

Sorted Names
Alicia
Jackson
Jason
Pythagoras



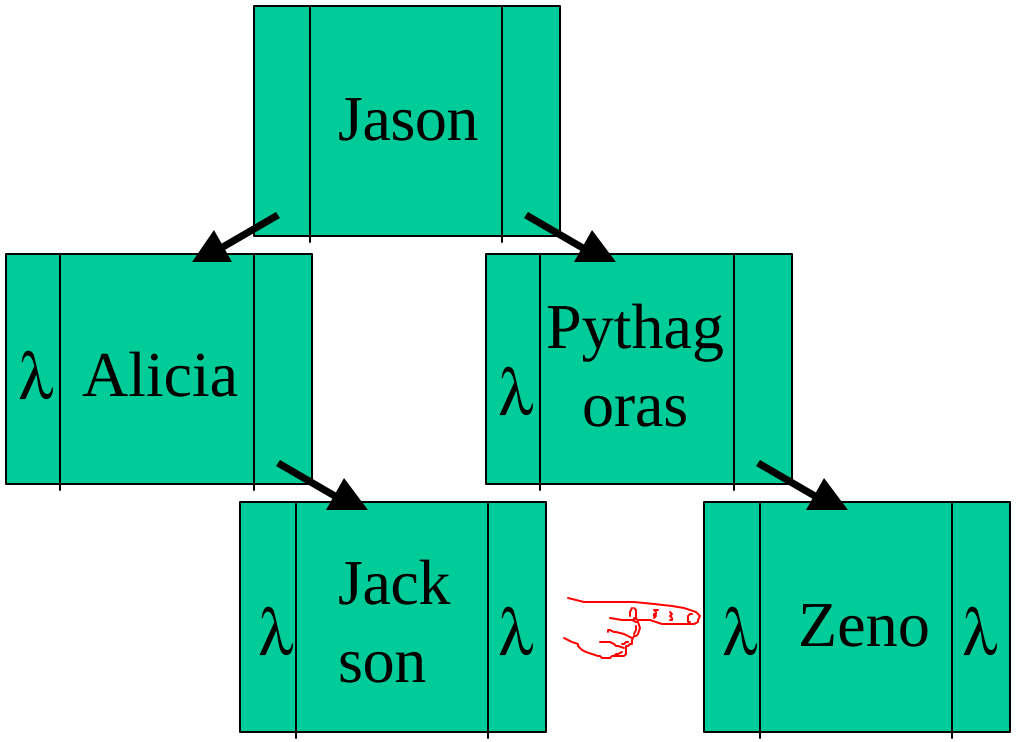
Go Left from Zeno

Sorted Names
Alicia
Jackson
Jason
Pythagoras



Zeno Left Link is Null

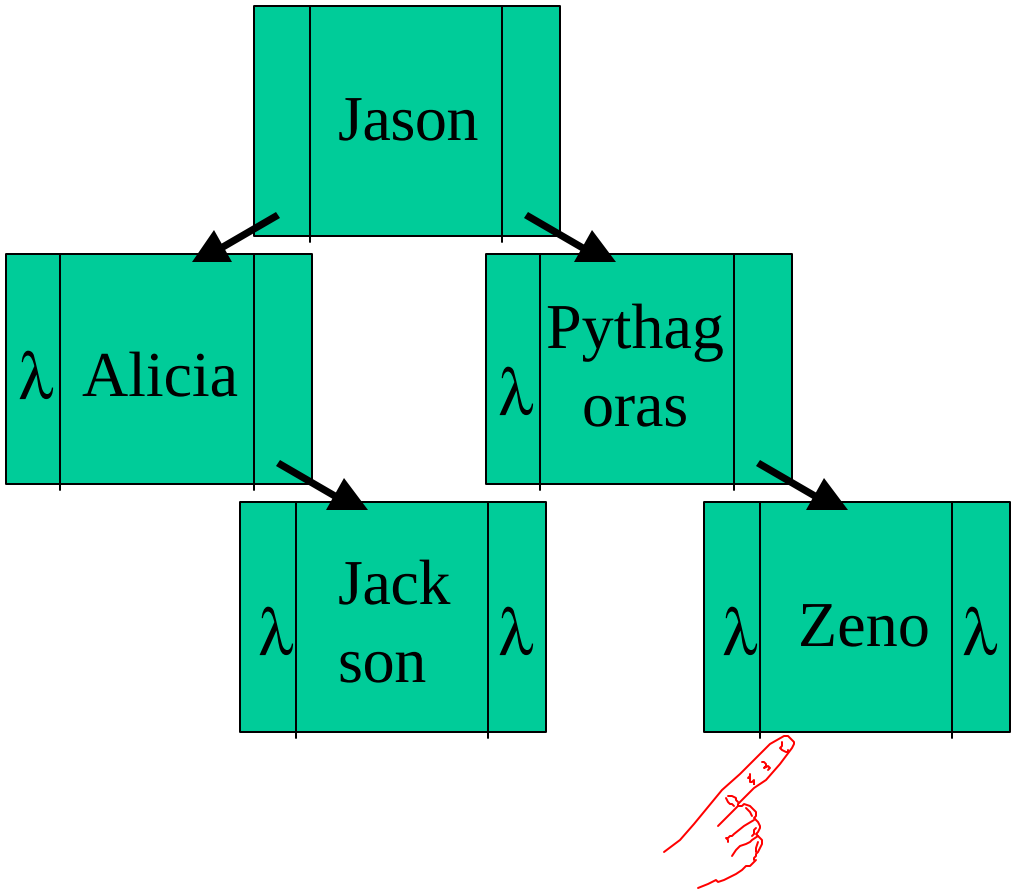
Sorted Names
Alicia
Jackson
Jason
Pythagoras



Since the left link is null, return to Zeno immediately.

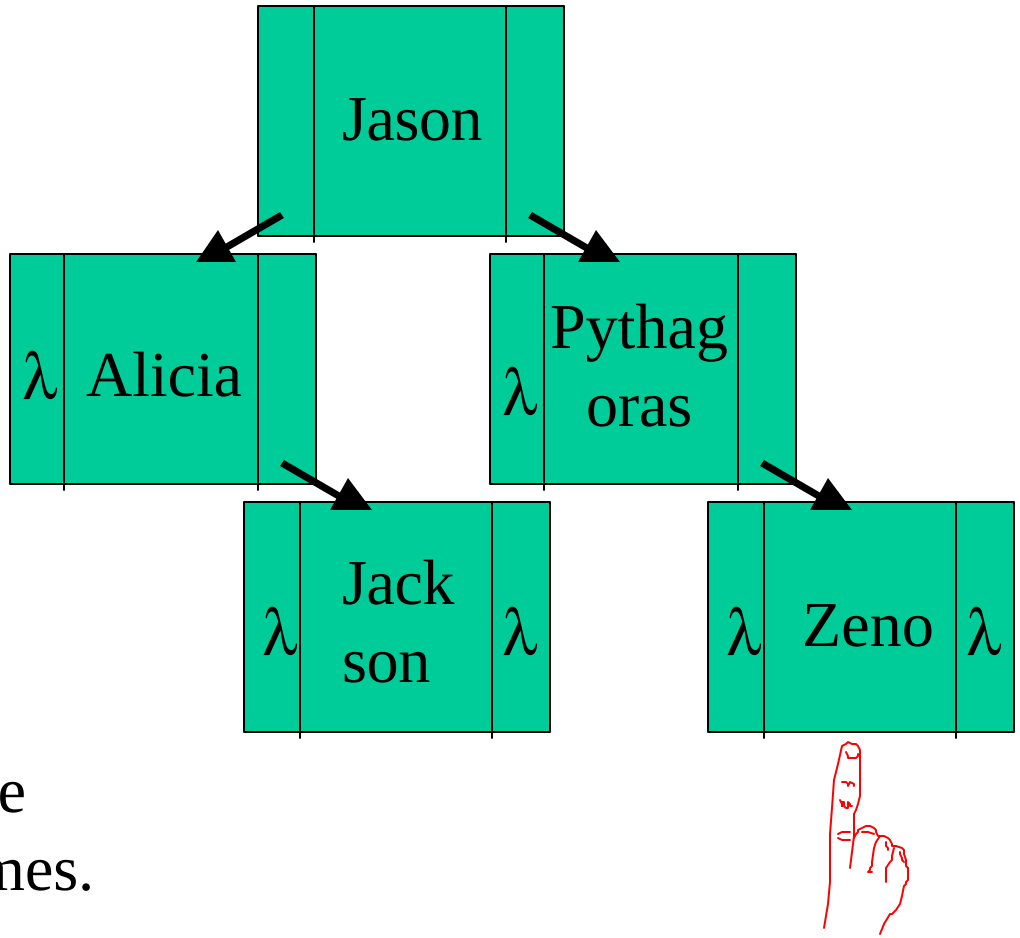
Return to Zeno from Left Link

Sorted Names
Alicia
Jackson
Jason
Pythagoras



Visit Zeno

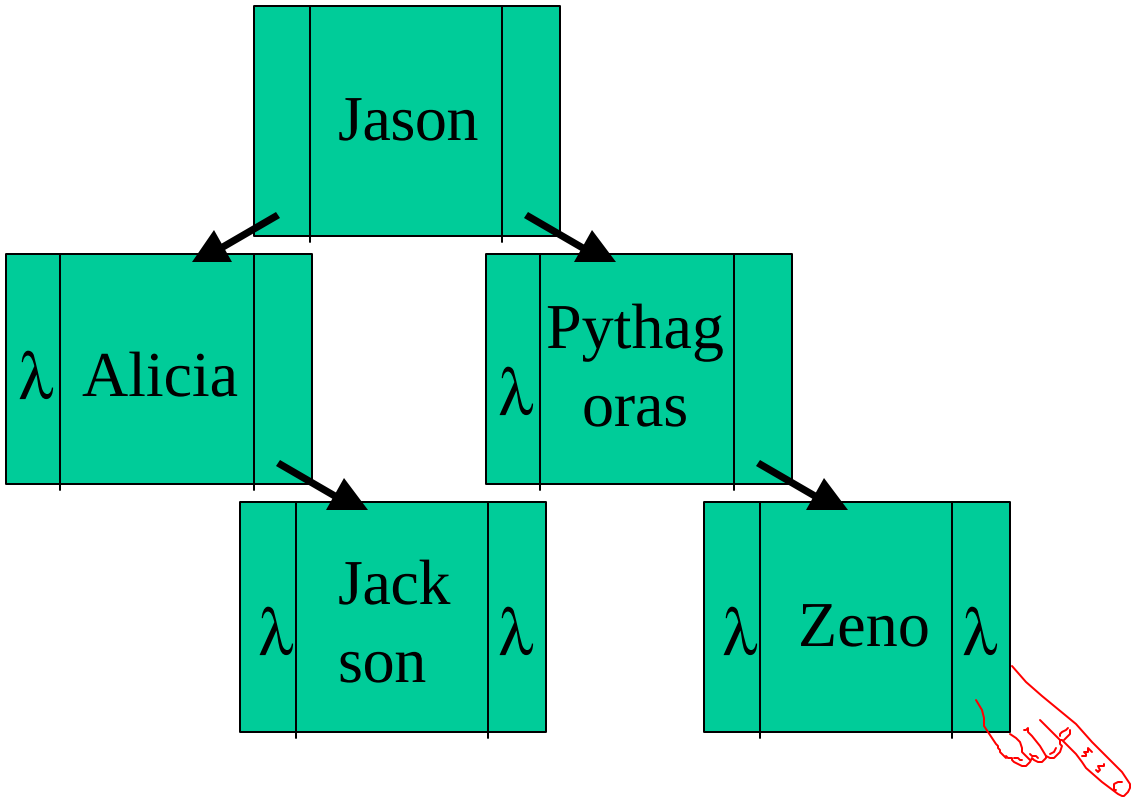
<u>Sorted Names</u>
Alicia
Jackson
Jason
Pythagoras
Zeno



Copy Zeno to the
list of sorted names.

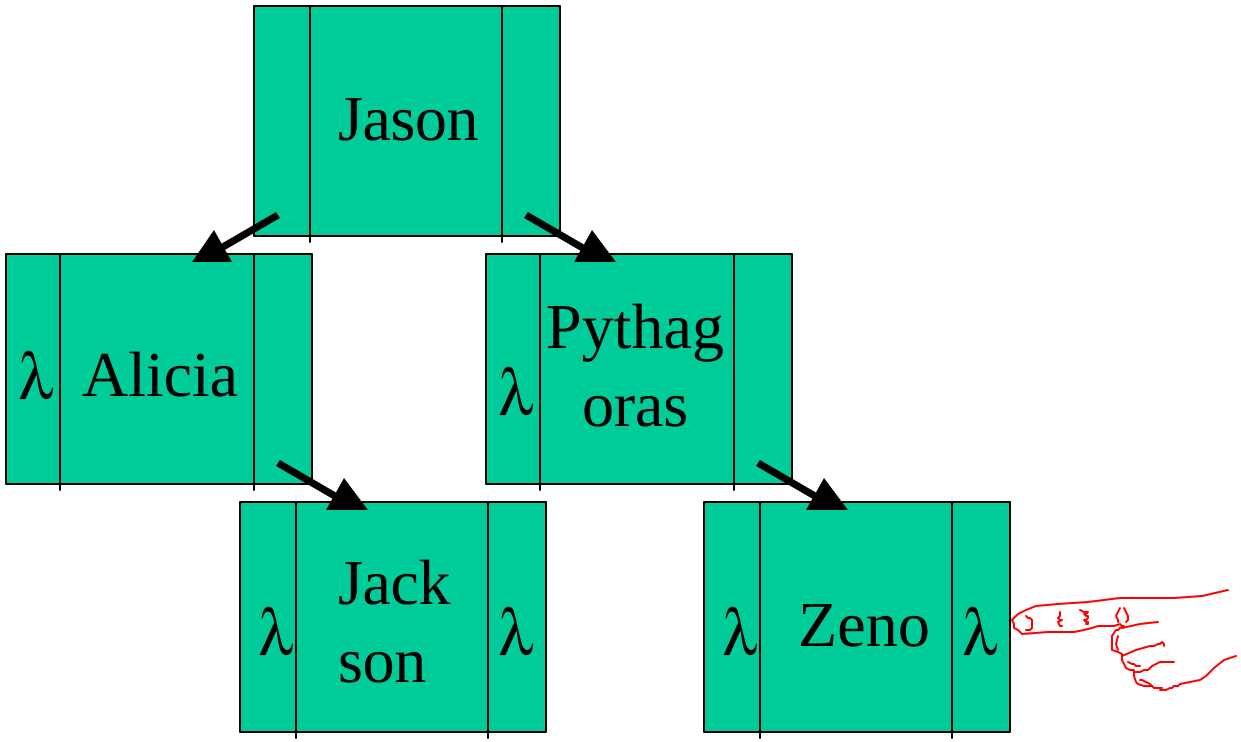
Go Right from Zeno

Sorted Names
Alicia
Jackson
Jason
Pythagoras
Zeno



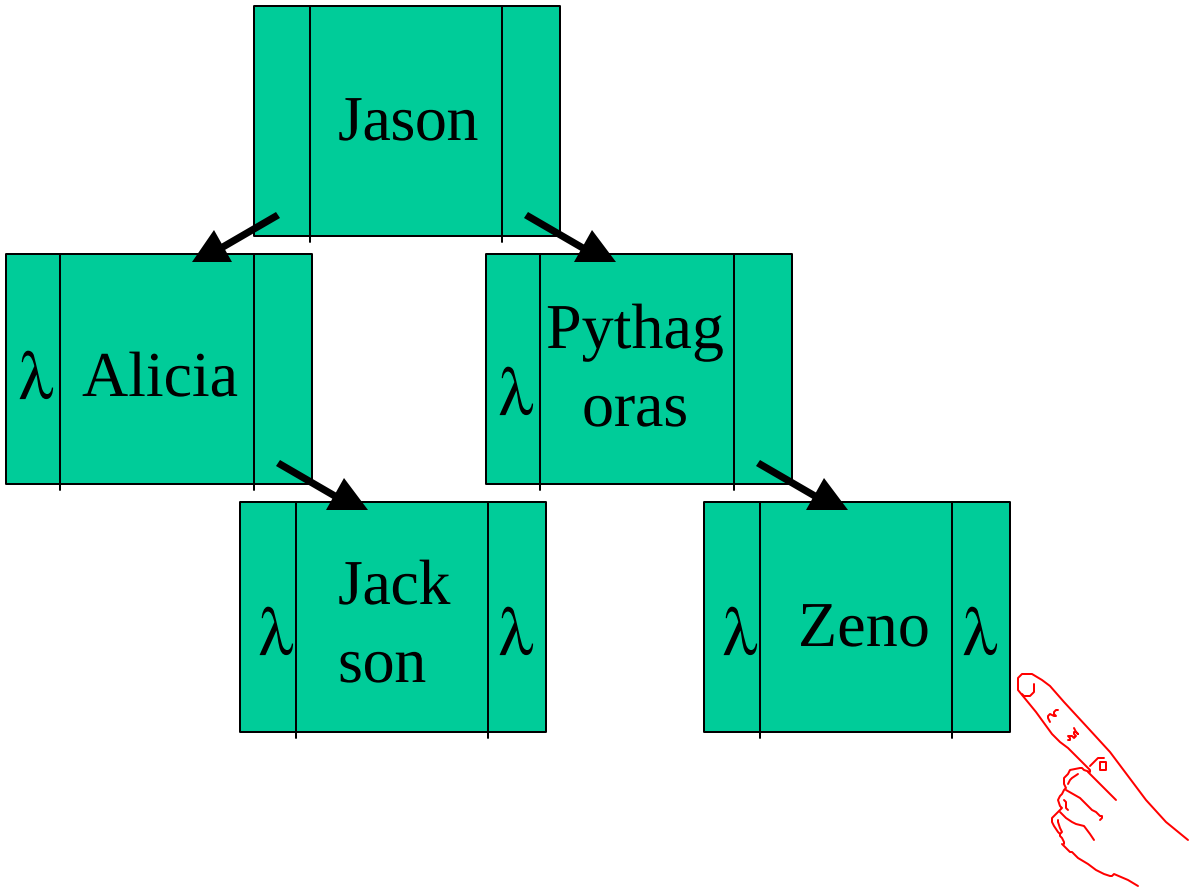
Zeno Right Link is Null

Sorted Names
Alicia
Jackson
Jason
Pythagoras
Zeno



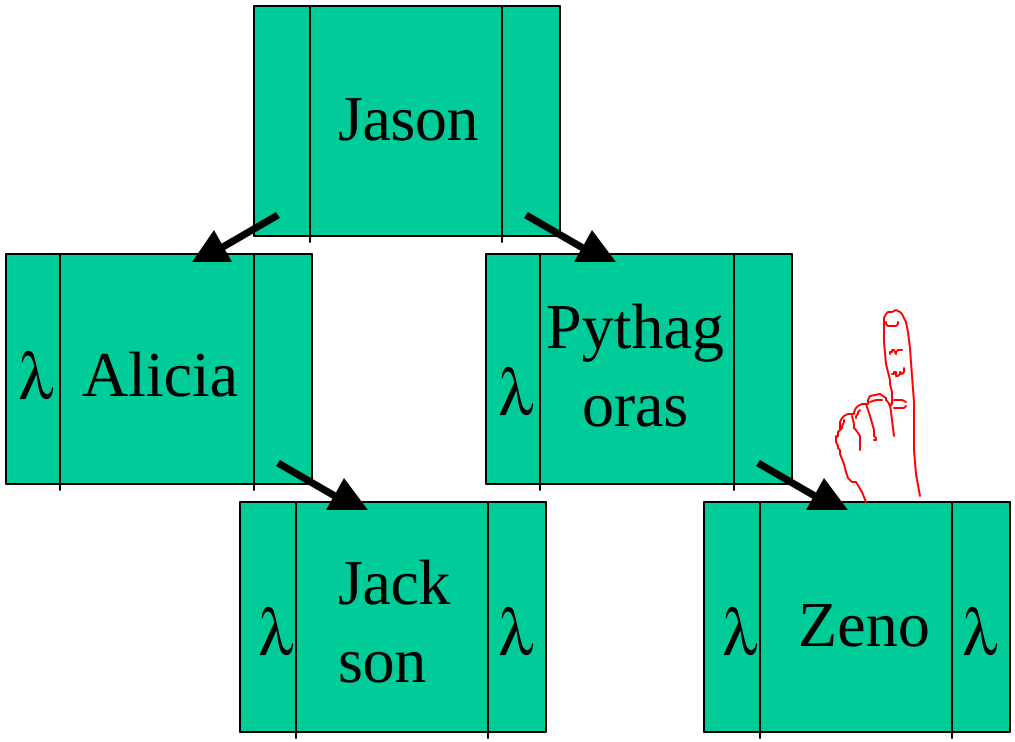
Return to Zeno from Right Link

Sorted Names
Alicia
Jackson
Jason
Pythagoras
Zeno



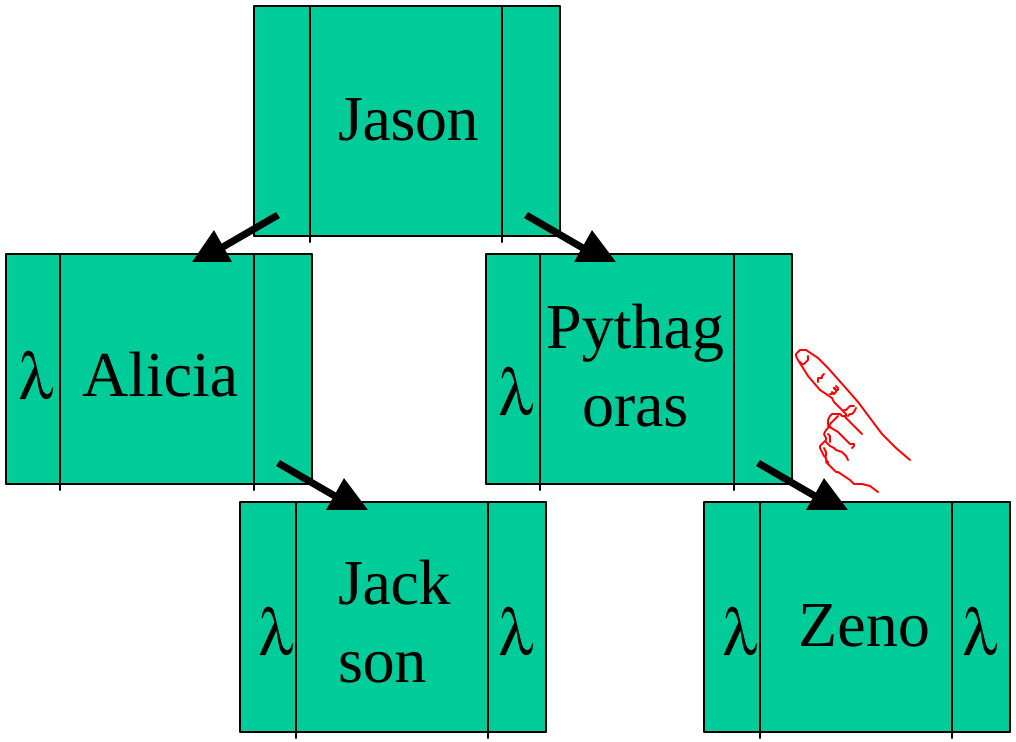
Leave Zeno

Sorted Names
Alicia
Jackson
Jason
Pythagoras
Zeno



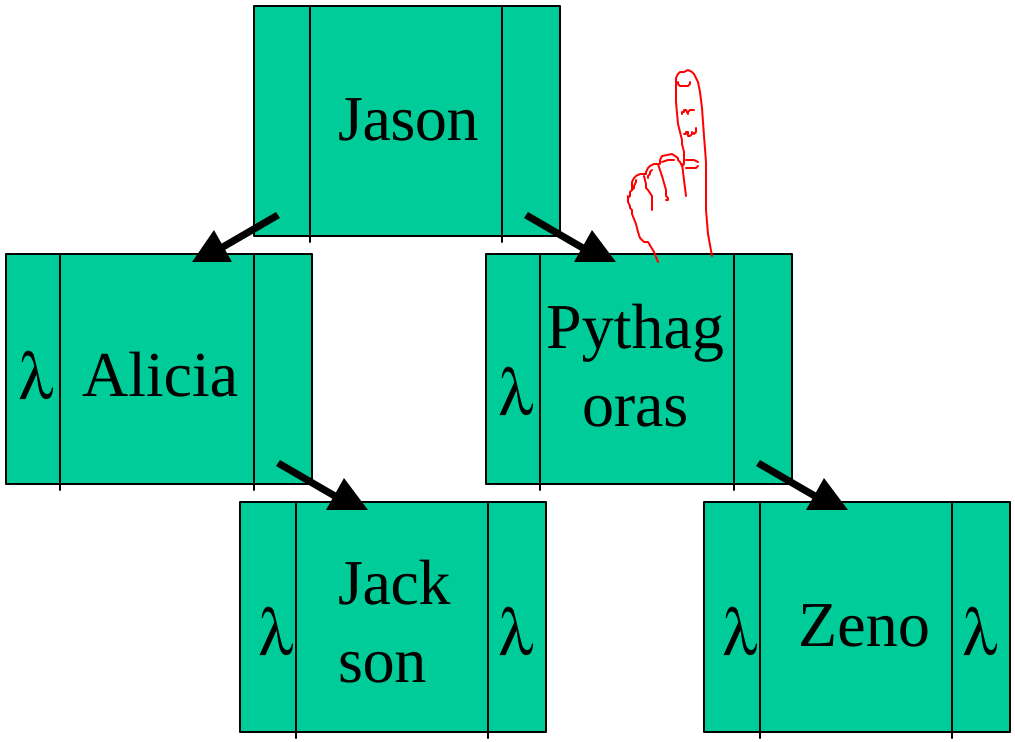
Return to Pythagoras from Right Link

Sorted Names
Alicia
Jackson
Jason
Pythagoras
Zeno



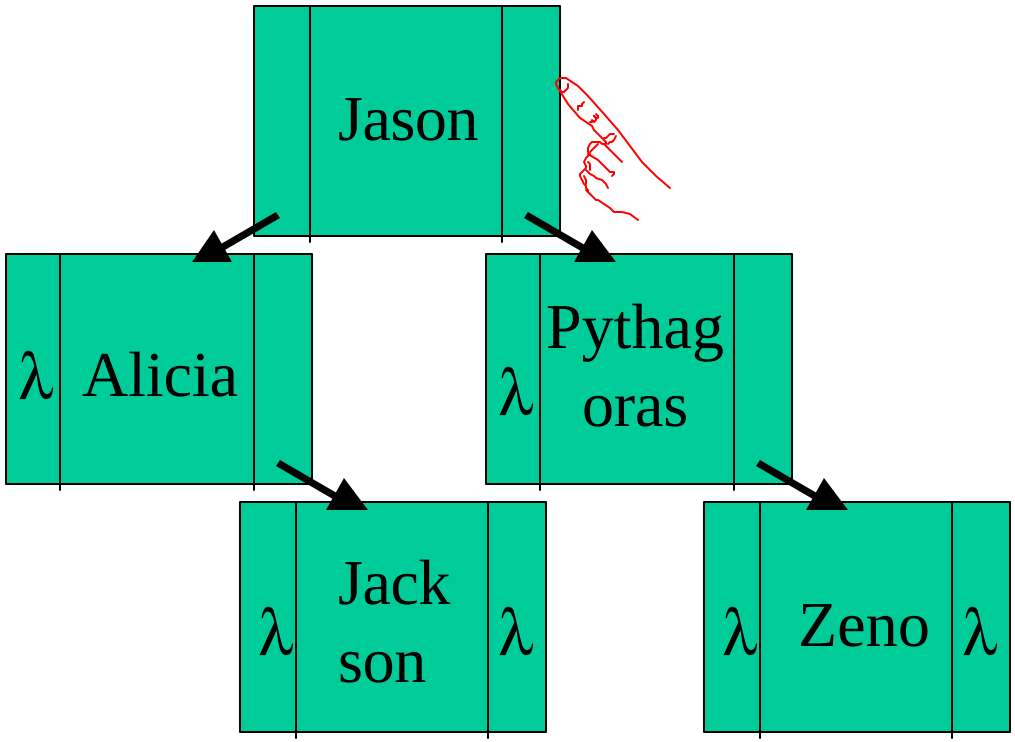
Leave Pythagoras

<u>Sorted Names</u>
Alicia
Jackson
Jason
Pythagoras
Zeno



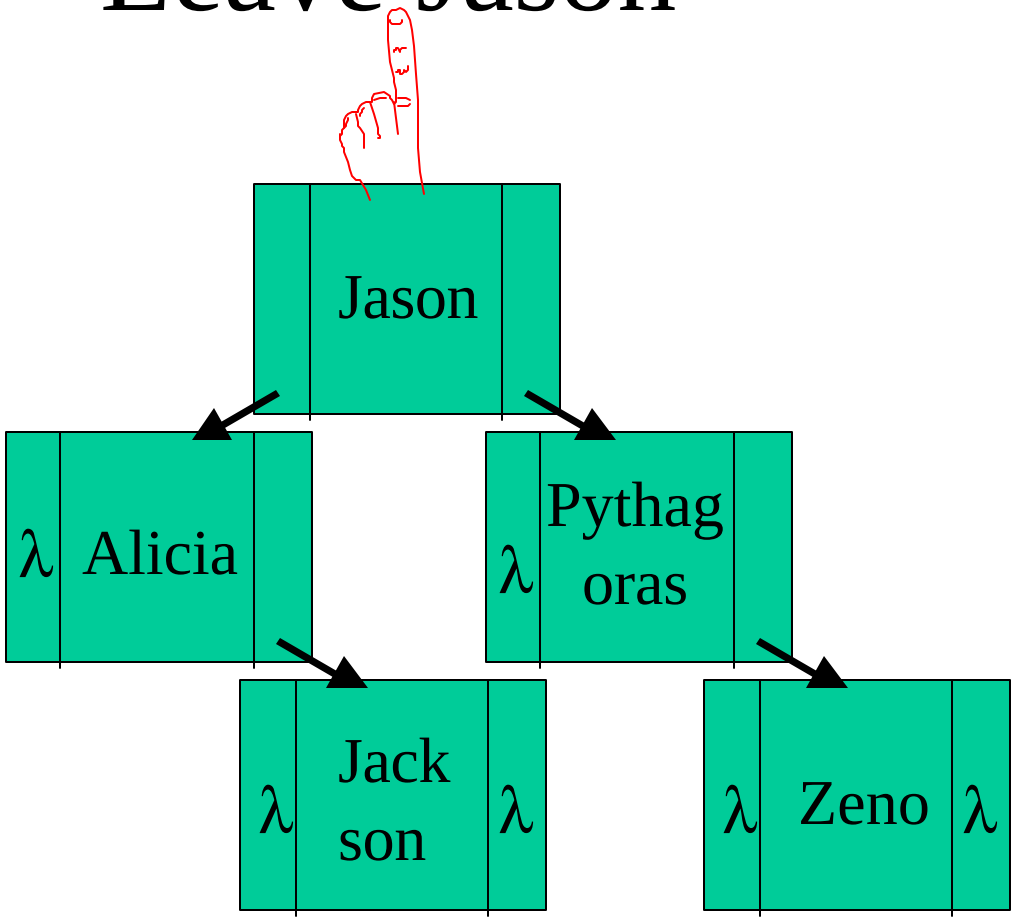
Return to Jason from Right Link

Sorted Names
Alicia
Jackson
Jason
Pythagoras
Zeno



Leave Jason

Sorted Names
Alicia
Jackson
Jason
Pythagoras
Zeno

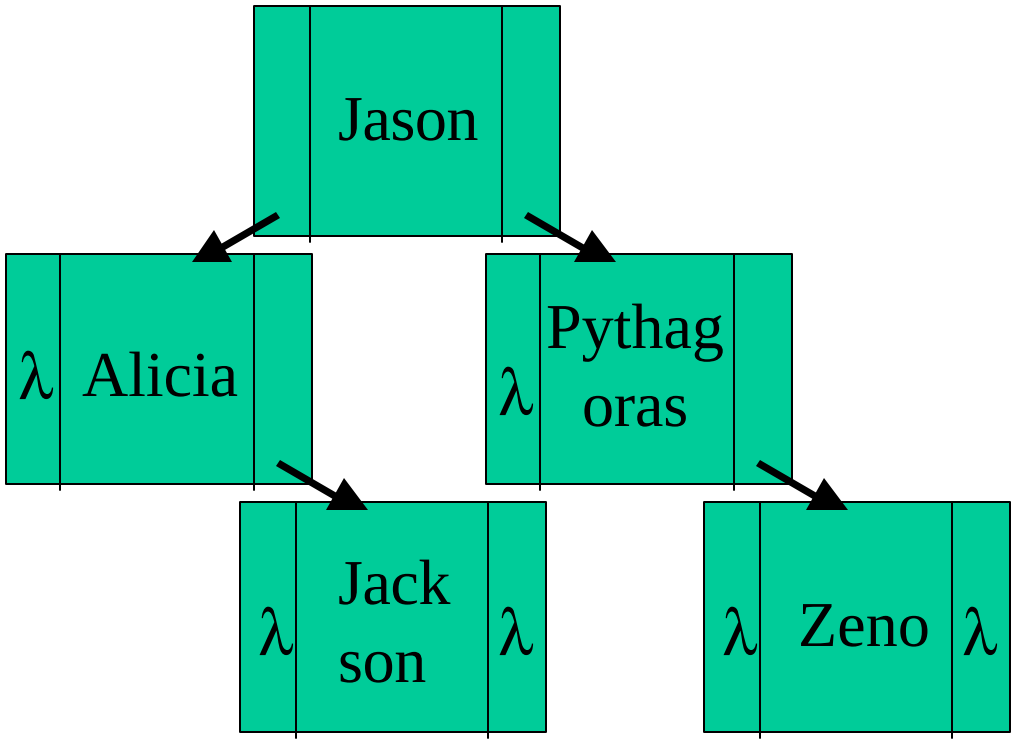


Done

Report Alphabetical List

Sorted Names

Alicia
Jackson
Jason
Pythagoras
Zeno



Implementation

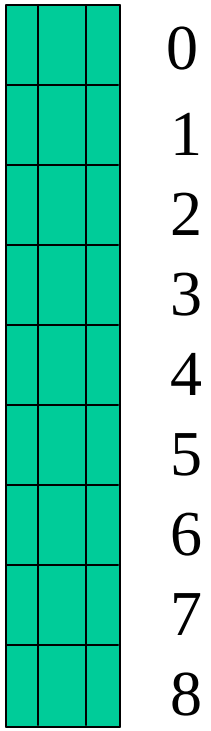
In real life, we do not move boxes around.

We change the pointers.

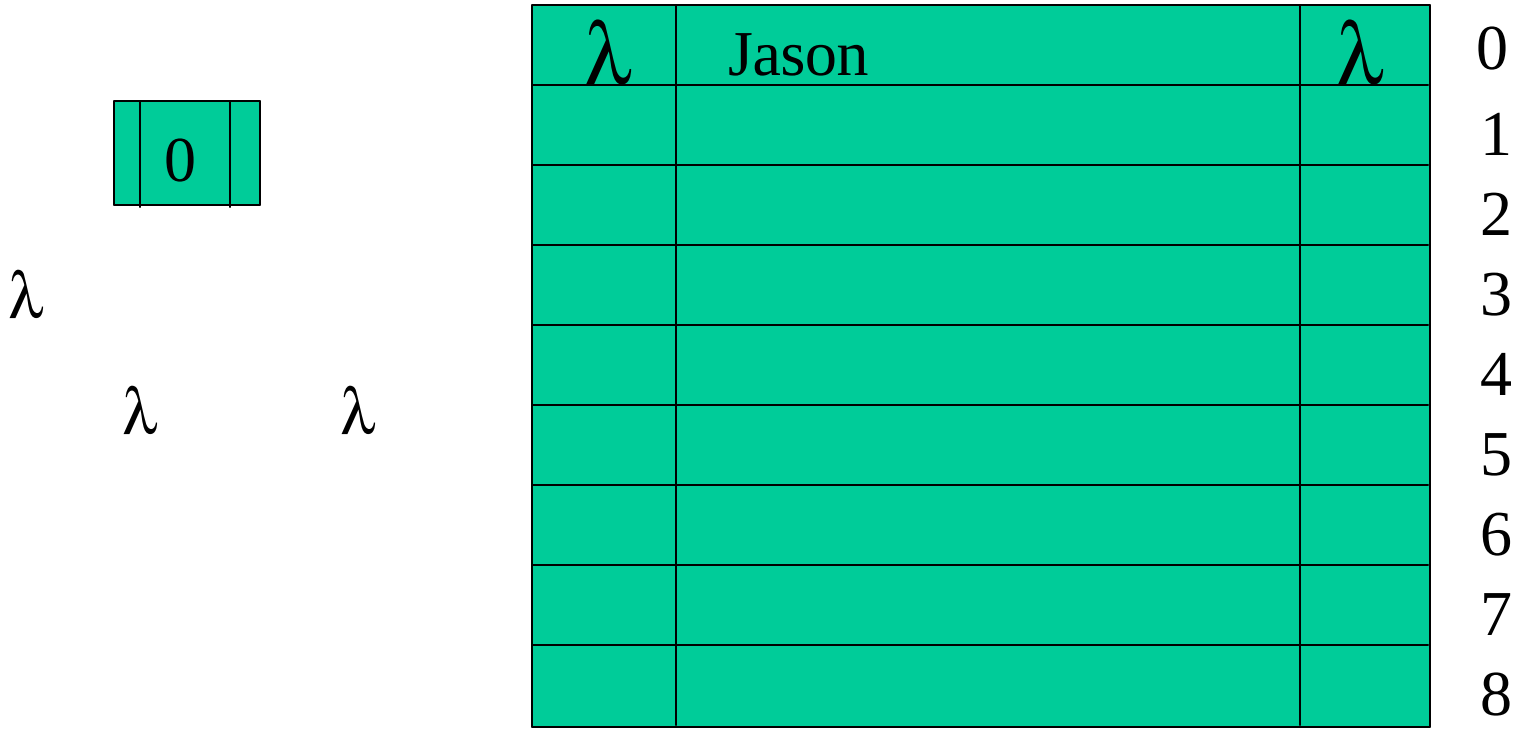
Pointers are addresses, or numbers that represent locations in memory where a node is stored.

Memory

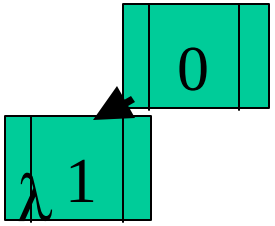
- Set aside a region of memory to be used for storing data. This is called the ***Available Stack***.
- As data arrives, it is immediately stored.
- A small additional amount of memory is used to establish the relationship of new data to data already present.
- The structure used to store data and accommodate the bookkeeping needed is called a ***Node***.



Repeat Example with Jason



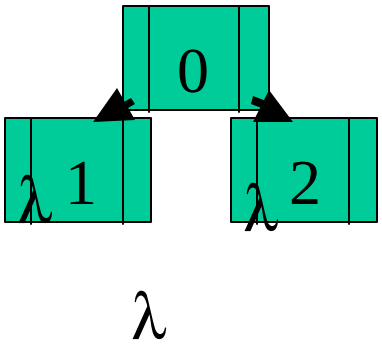
Repeat Example with Alicia



λ λ

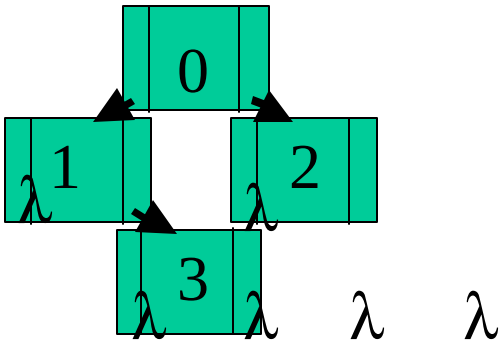
1	Jason	λ	0
λ	Alicia	λ	1
			2
			3
			4
			5
			6
			7
			8

Repeat Example with Pythagoras



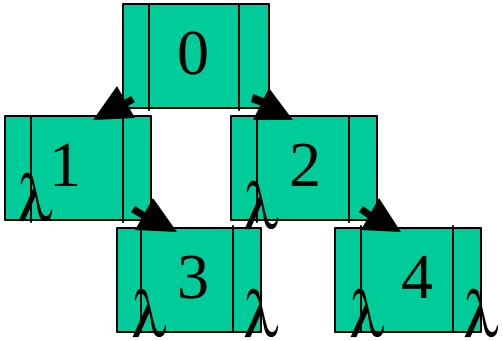
1	Jason	2	0
λ	Alicia	λ	1
λ	Pythagoras	λ	2
			3
			4
			5
			6
			7
			8

Repeat Example with Jackson



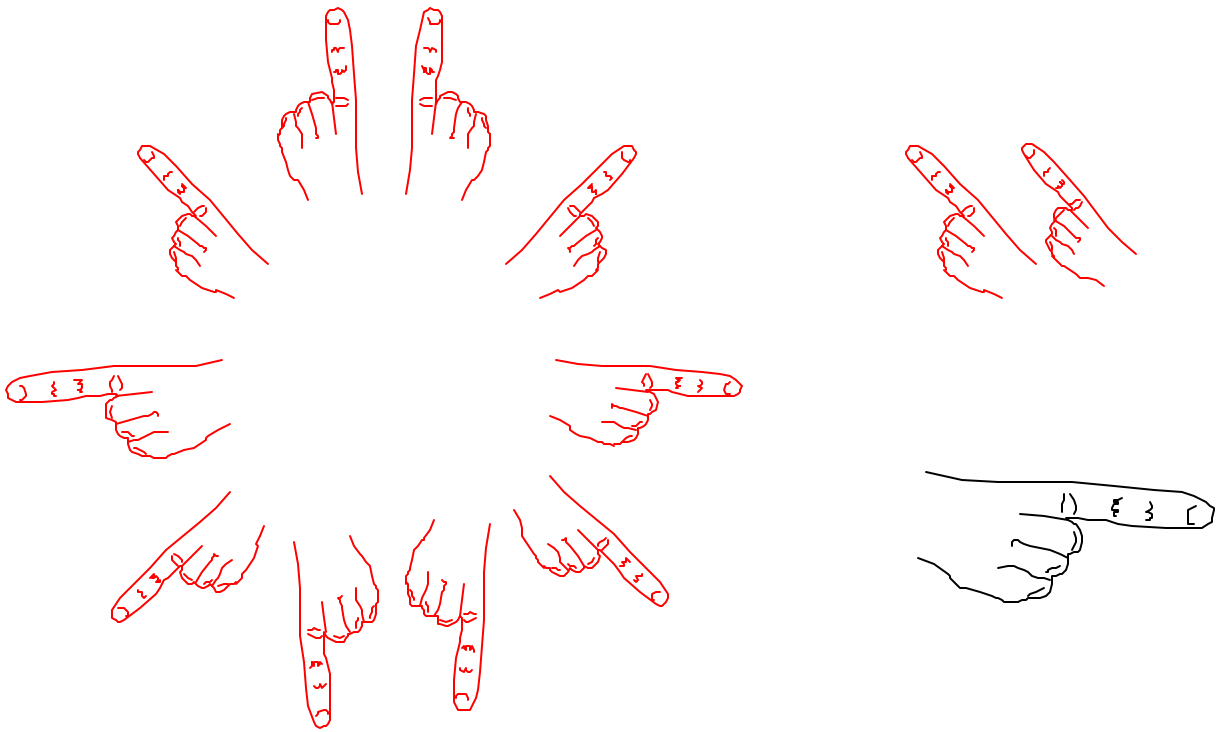
1	Jason	2	0
λ	Alicia	3	1
λ	Pythagoras	λ	2
λ	Jackson	λ	3
			4
			5
			6
			7
			8

Repeat Example with Zeno



1	Jason	2	0
λ	Alicia	3	1
λ	Pythagoras	4	2
λ	Jackson	λ	3
λ	Zeno	λ	4
			5
			6
			7
			8

The End



Hands Across America