UVM based Generic Interrupt Service Routine (gISR)

Ritesh Mehta ,
Design Verification Engineer , Google

Nakul Sharma
Senior Design Verification Engineer , Google

Agenda

- Motivation
- gISR
  - The Idea
  - The Algo - Tree Traversal
  - Exclusive checks
  - UVM Implementation
- Proof of Concept
  - Issues Reported
- Future Scope

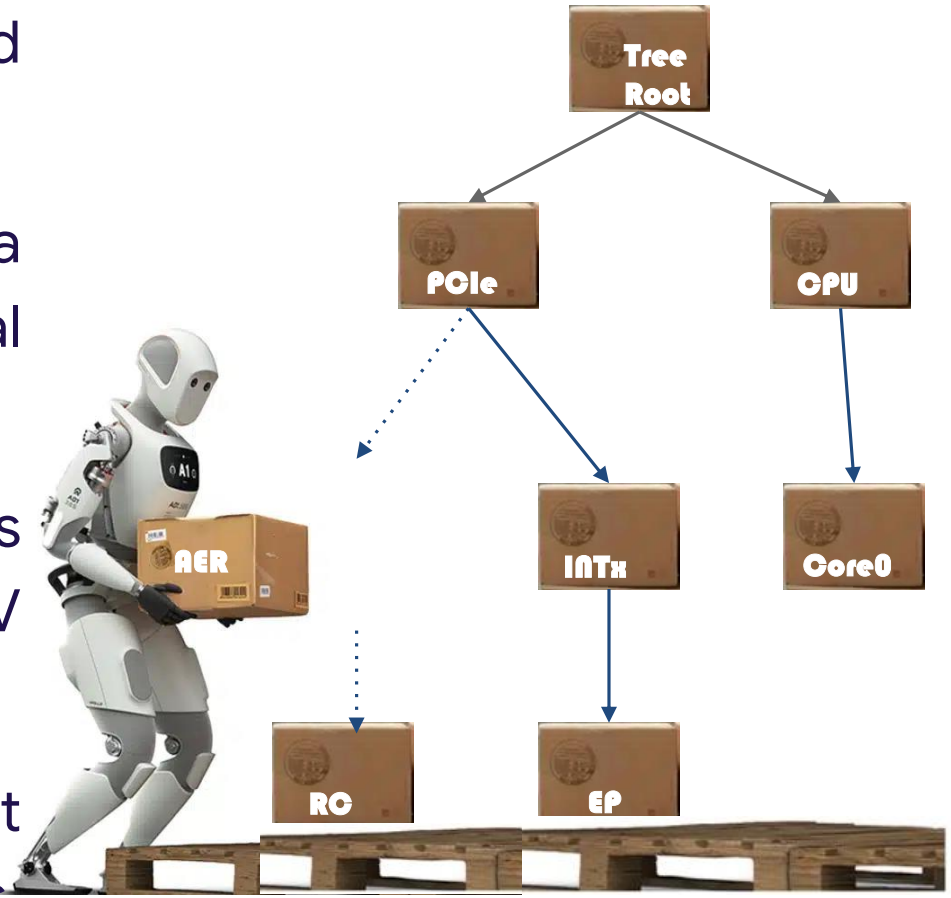
Motivation

The traditional reliance on manual techniques and ad-hoc test cases for interrupt tree verification is plagued by inherent limitations and drawbacks :

- Time-Consuming and Error-Prone
- Limited Coverage
- Difficulty in Scaling & reduced Reusability
- Lack of Standardization
- Limited Visibility into Interrupt Behavior

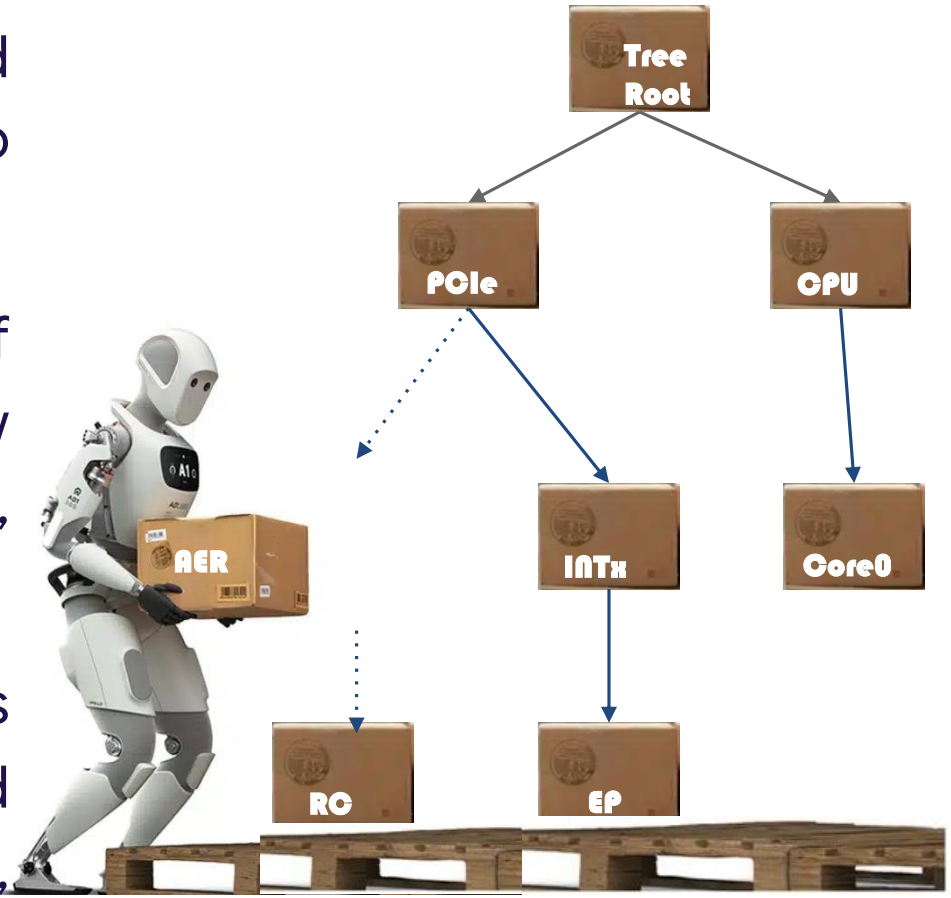
gISR

- **gISR (generic ISR):** A ready-made, UVM based , software-like interrupt handler, eliminating the need for users to write custom code from scratch.
- **Automatic Interrupt Tree:** Builds a representation of the interrupt system with minimal effort from user.
- **Easy to Add Interrupts:** Simple UVM APIs (functions) to add new interrupt sources to the ENV infrastructure.
- **Customizable Interrupts:** Ability to set properties for each interrupt like Status, Masks, Severity or any other custom properties.

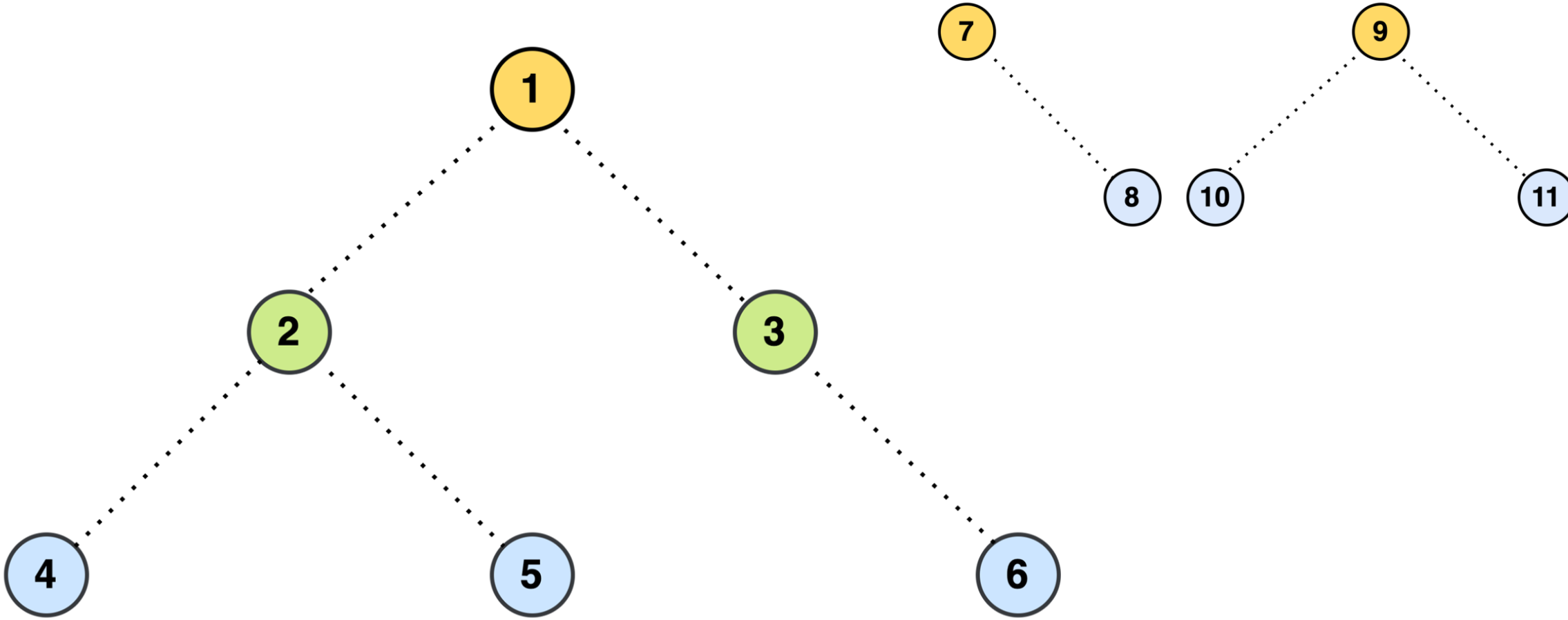


gISR

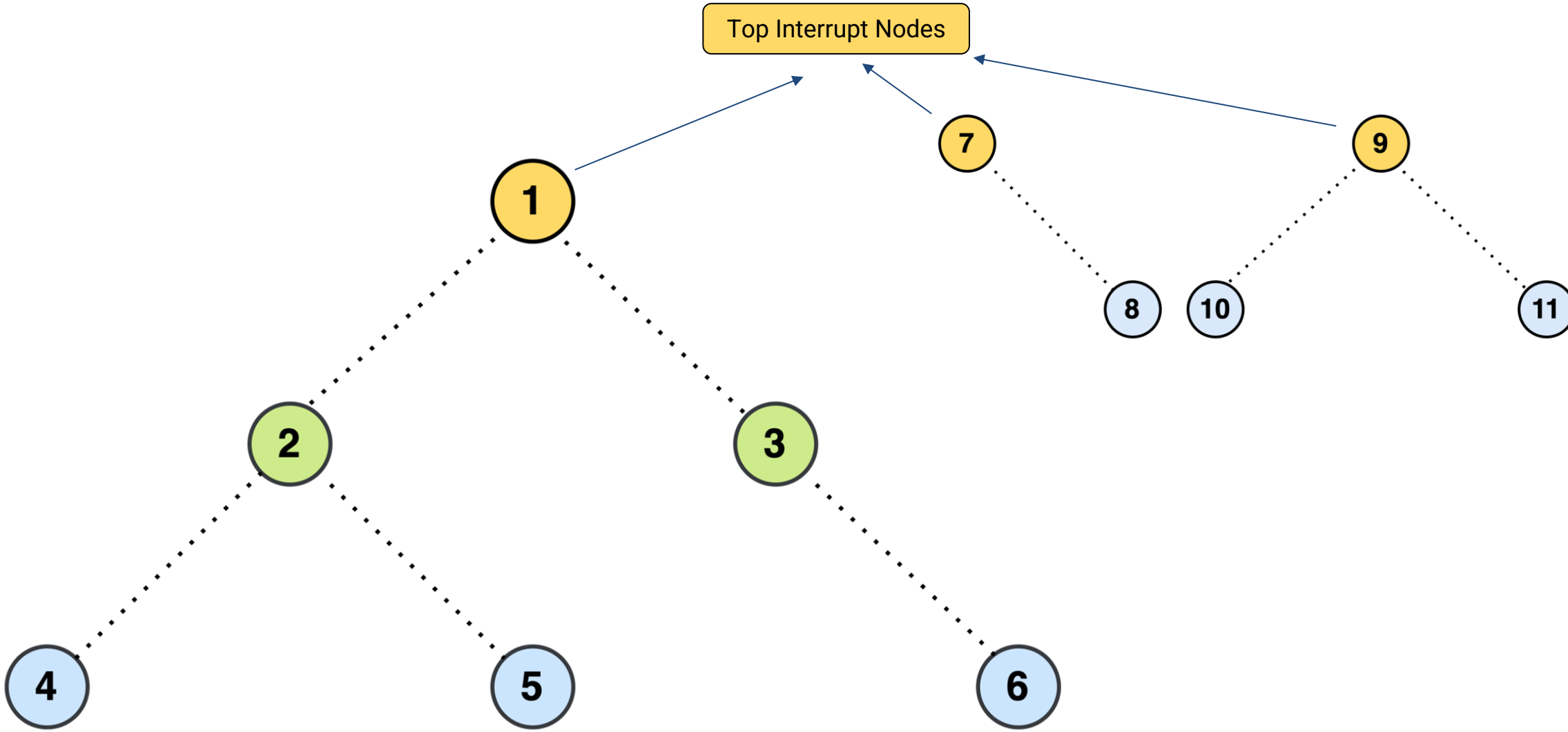
- **Smart Checking:** Automated holistic checks to validate if interrupts are behaving correctly based on the design. Provision of standardized reports to make debugging easy.
- **Support for Variety:** Handles different types of interrupts (level-triggered, pulse-based) and how they are cleared (Write 1 to Clear (W1C), Auto Clear, etc.).
- **Helper APIs :** Curated library of easy-to-use APIs to help user customizing the test sequences around interrupts. For ex : set_expected_interrupt(), wait_for_interrupt(), etc.



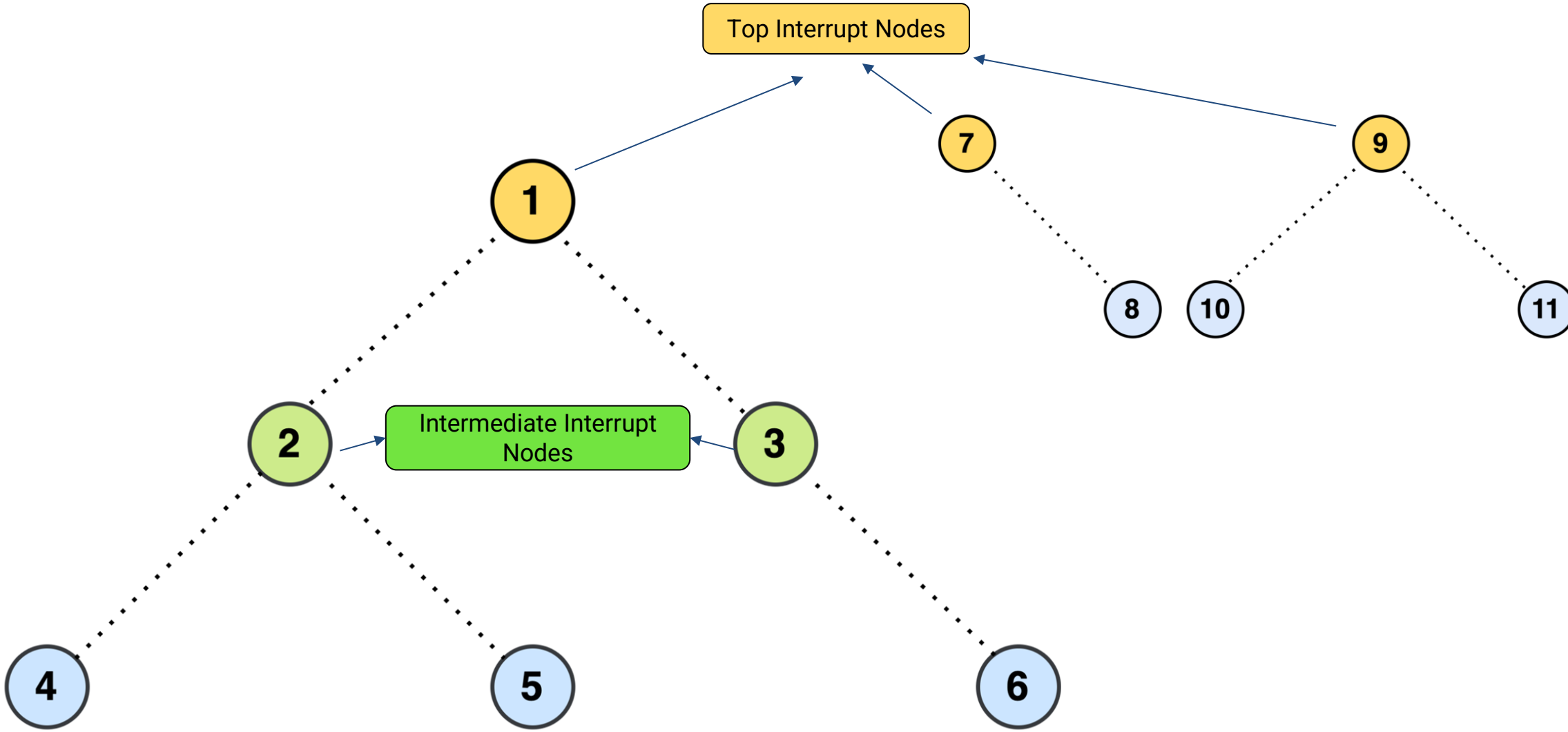
The Idea



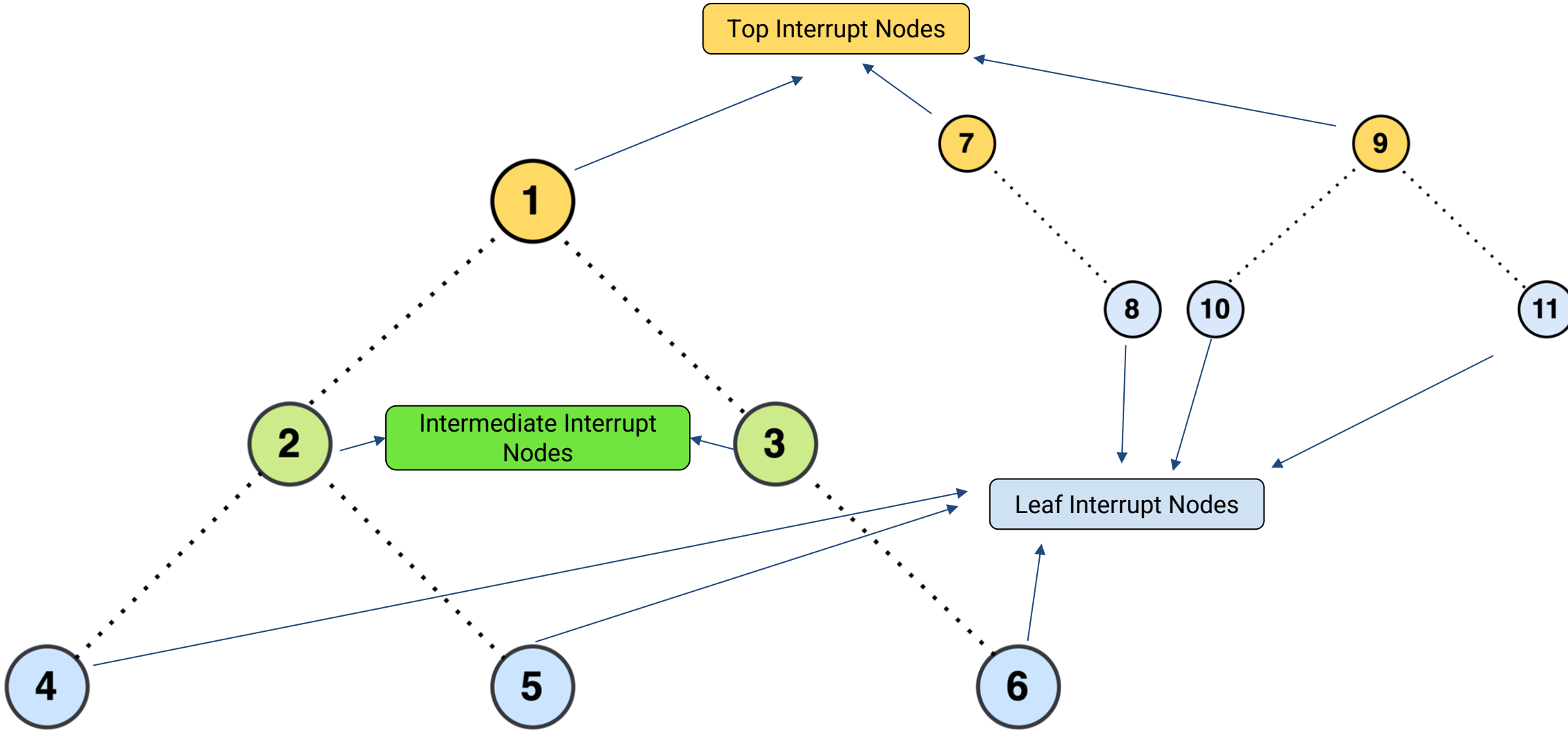
The Idea



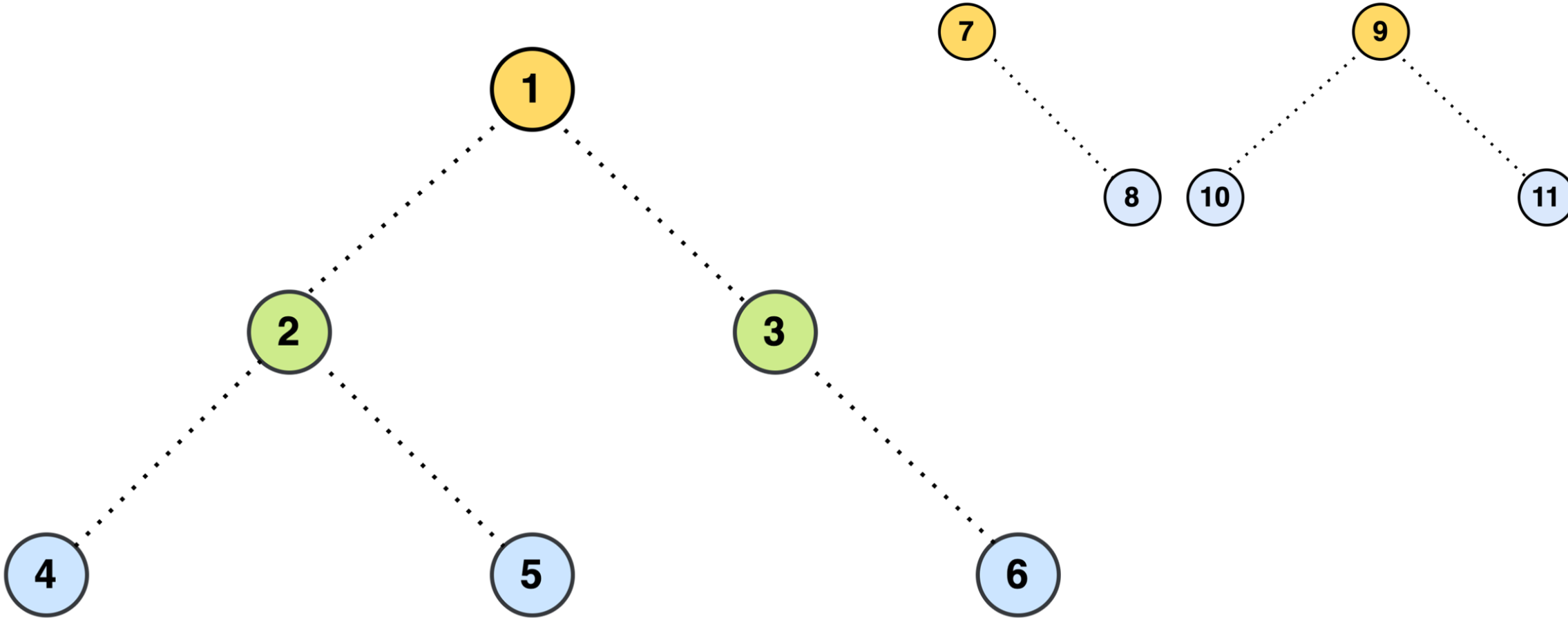
The Idea


Top Interrupt Nodes
1
7
9
8
10
11
2
Intermediate Interrupt Nodes
3
4
5
6

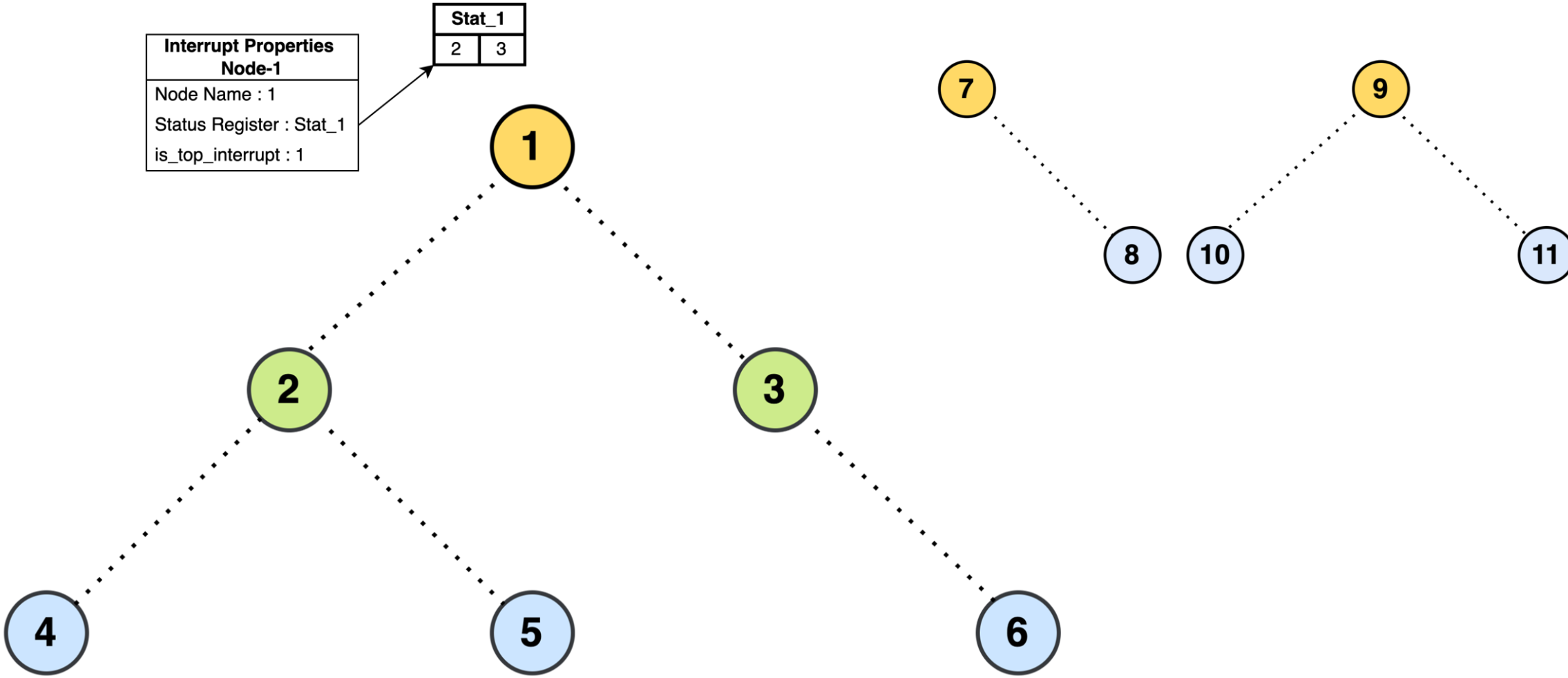
The Idea


Top Interrupt Nodes
1
2
3
4
5
6
7
8
9
10
11
Intermediate Interrupt Nodes
Leaf Interrupt Nodes

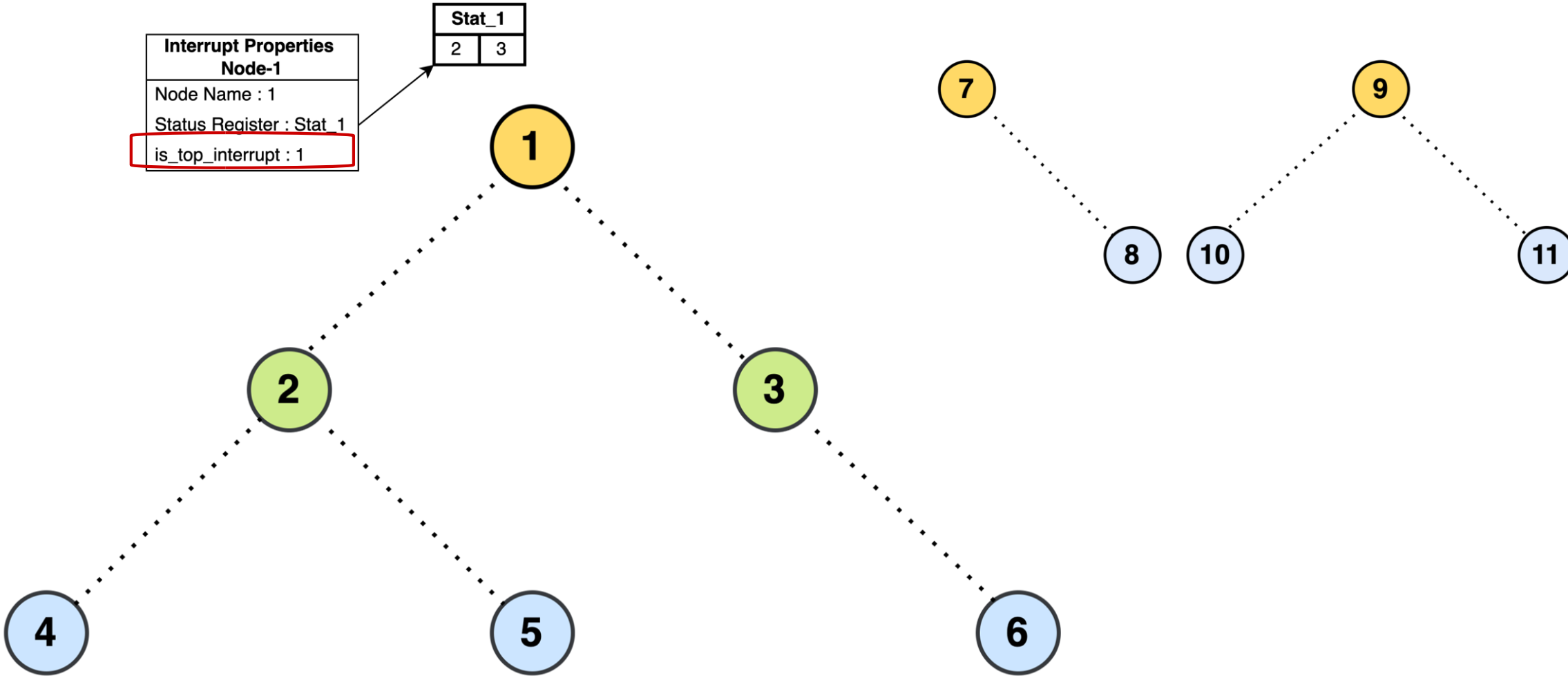
The Idea



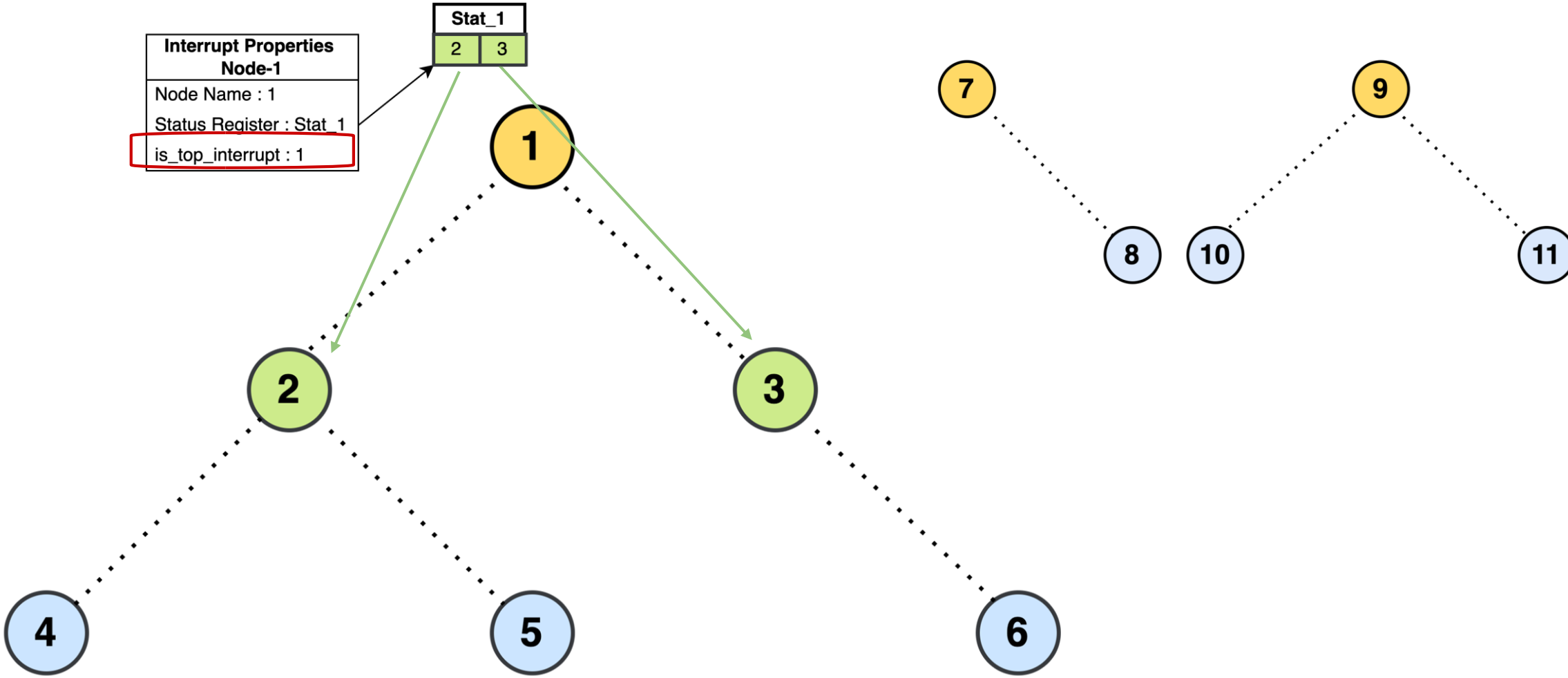
The Idea


Interrupt Properties Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
1
2
3
4
5
6
7
8
9
10
11

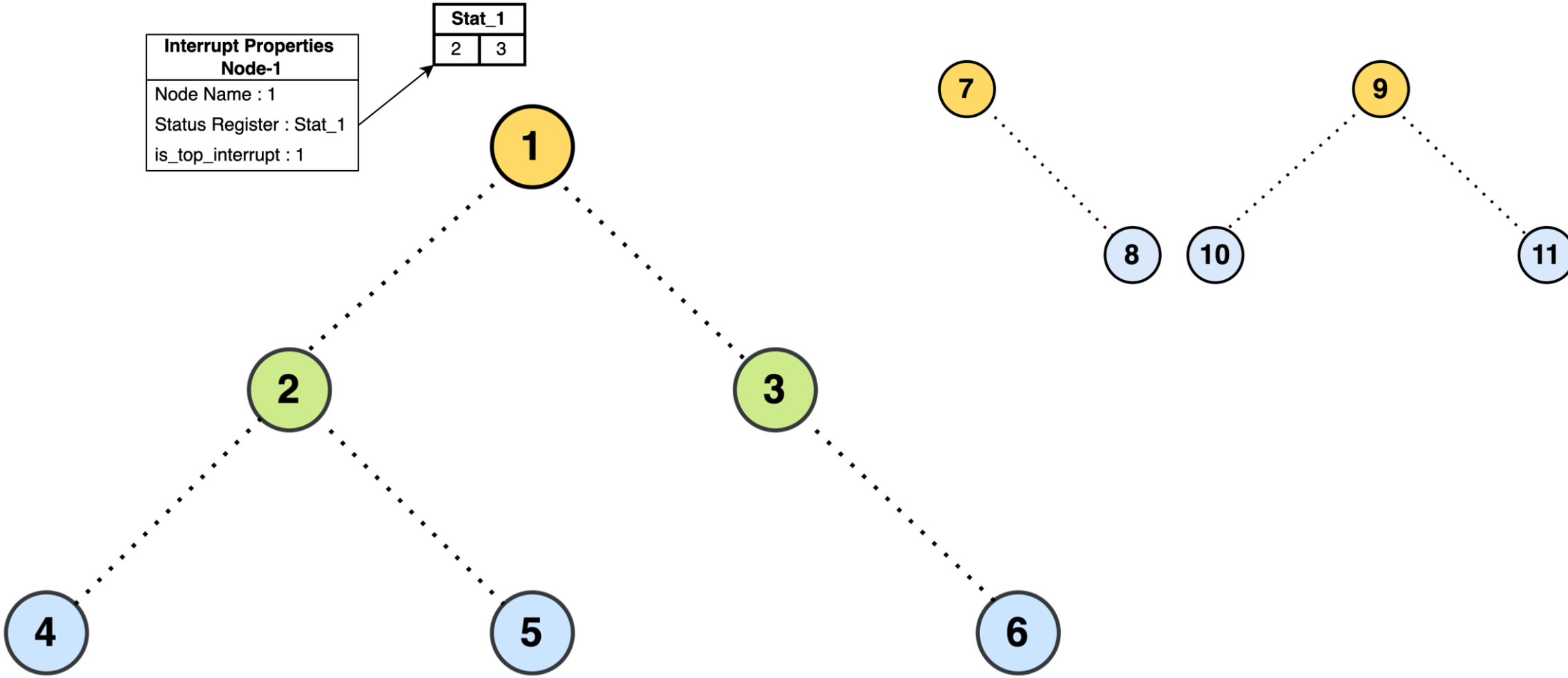
The Idea


Interrupt Properties
Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
1
2
3
4
5
6
7
8
9
10
11

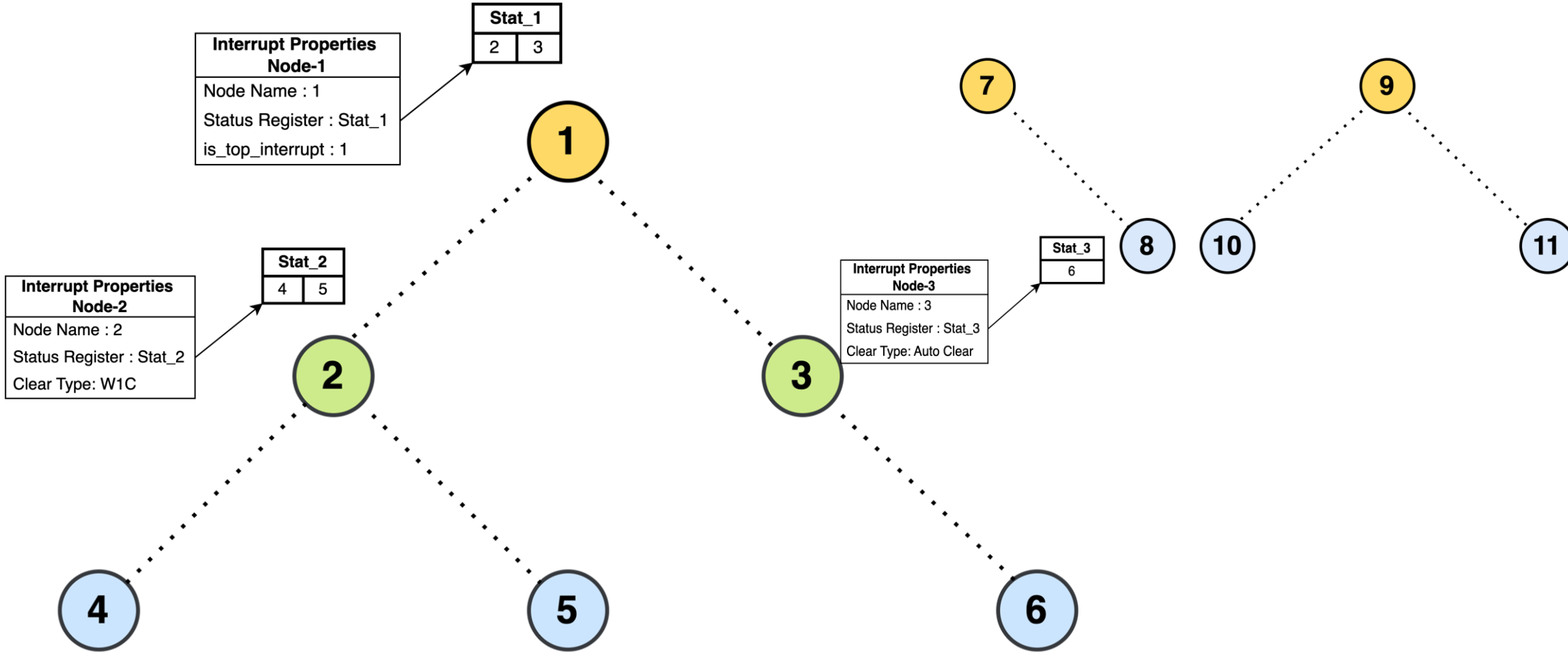
The Idea


Interrupt Properties Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
1
2
3
4
5
6
7
8
9
10
11

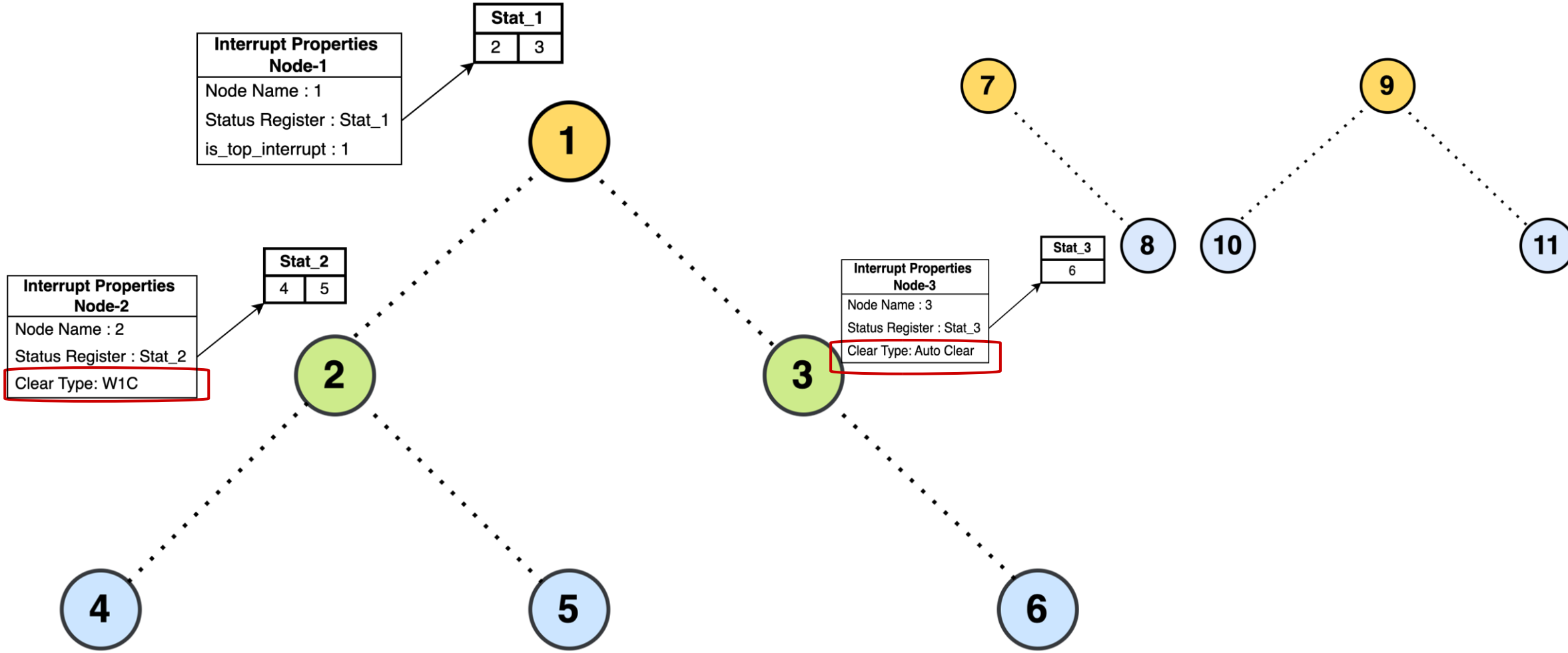
The Idea


Interrupt Properties Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
1
2
3
4
5
6
7
8
9
10
11

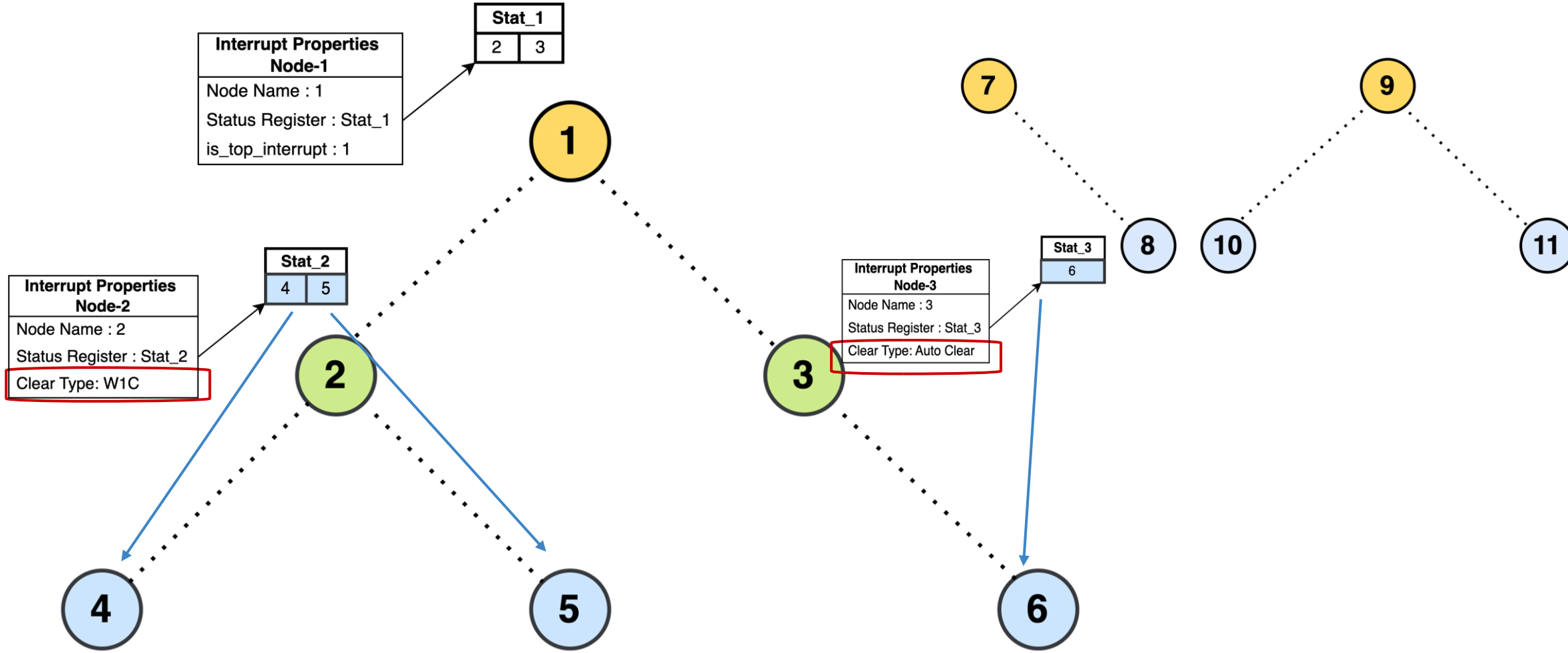
The Idea


Interrupt Properties
Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
Interrupt Properties
Node-2
Node Name : 2
Status Register : Stat_2
Clear Type: W1C
Stat_2
4
5
Interrupt Properties
Node-3
Node Name : 3
Status Register : Stat_3
Clear Type: Auto Clear
Stat_3
6
1
2
3
4
5
6
7
8
9
10
11

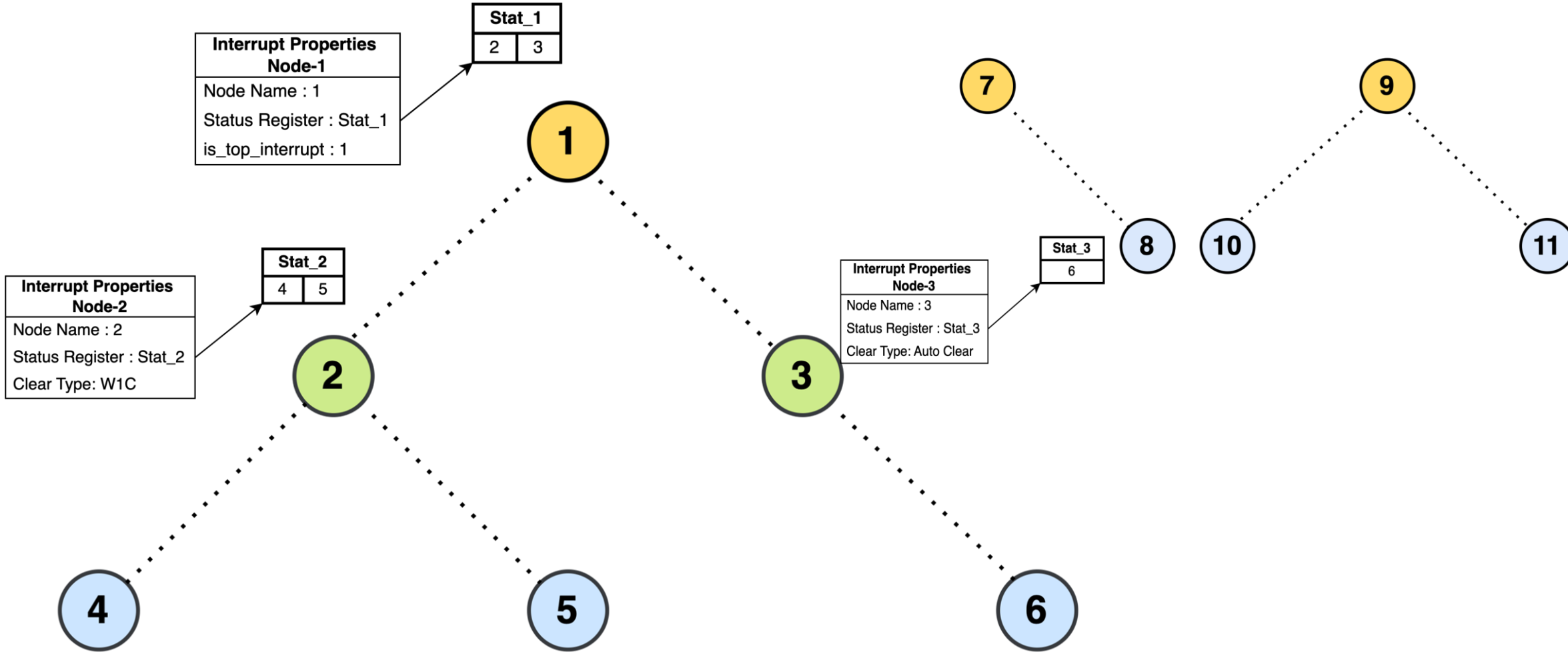
The Idea


Interrupt Properties Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
Interrupt Properties Node-2
Node Name : 2
Status Register : Stat_2
Clear Type: W1C
Stat_2
4
5
Interrupt Properties Node-3
Node Name : 3
Status Register : Stat_3
Clear Type: Auto Clear
Stat_3
6
1
2
3
4
5
6
7
8
9
10
11

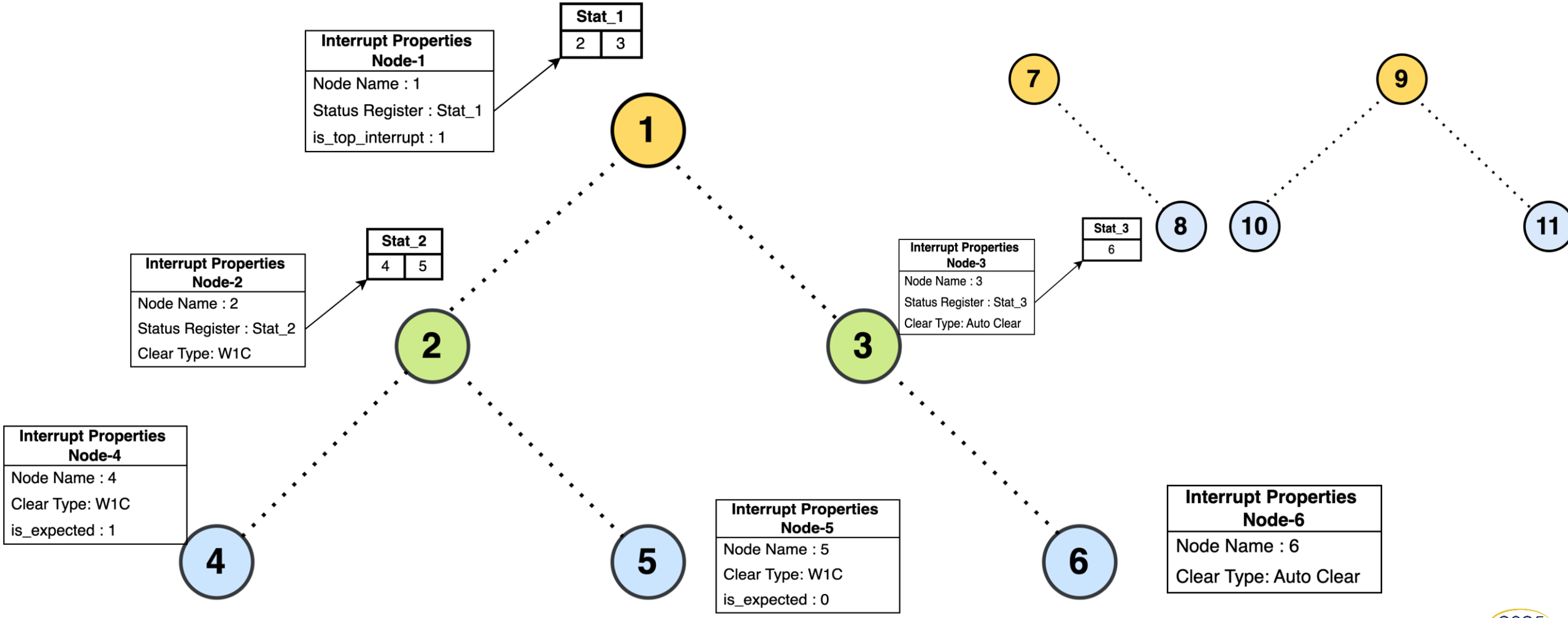
The Idea


Interrupt Properties
Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
Interrupt Properties
Node-2
Node Name : 2
Status Register : Stat_2
Clear Type: W1C
Stat_2
4
5
Interrupt Properties
Node-3
Node Name : 3
Status Register : Stat_3
Clear Type: Auto Clear
Stat_3
6
1
2
3
4
5
6
7
8
9
10
11

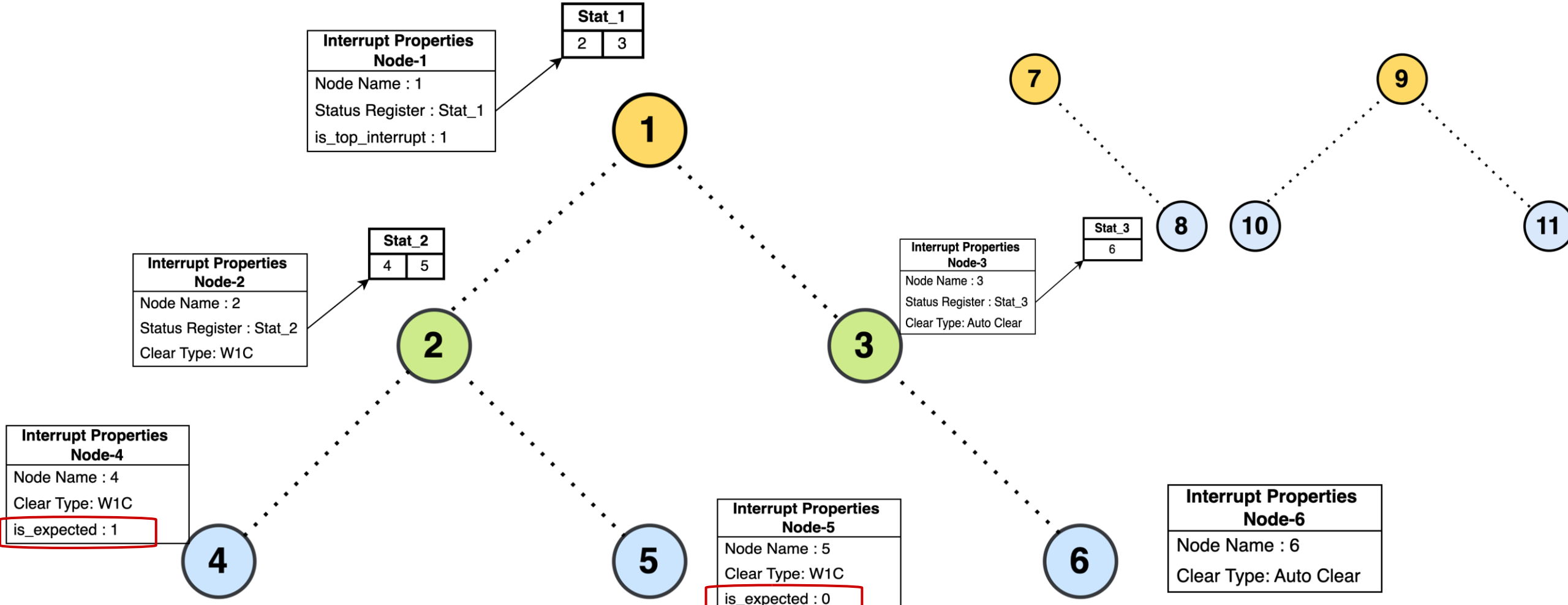
The Idea


Interrupt Properties Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
Interrupt Properties Node-2
Node Name : 2
Status Register : Stat_2
Clear Type: W1C
Stat_2
4
5
Interrupt Properties Node-3
Node Name : 3
Status Register : Stat_3
Clear Type: Auto Clear
Stat_3
6
1
2
3
4
5
6
7
8
9
10
11

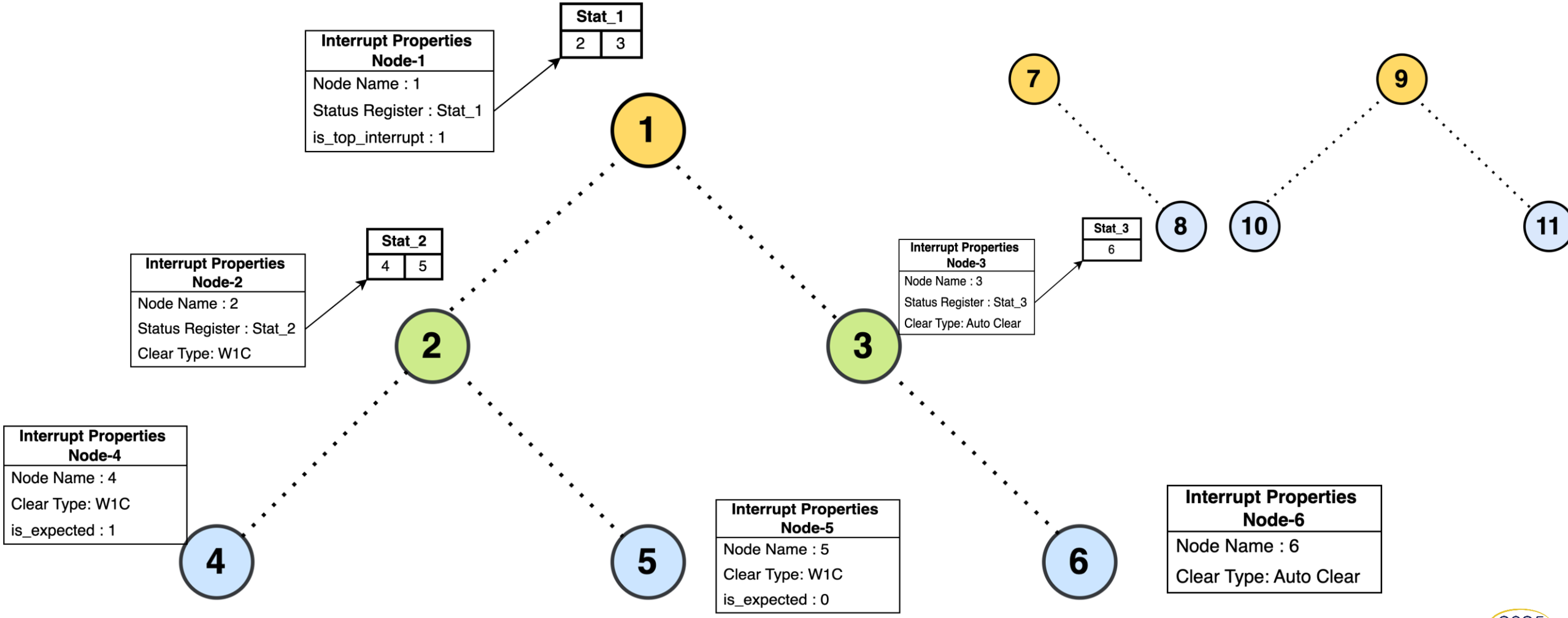
The Idea


Interrupt Properties Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
Interrupt Properties Node-2
Node Name : 2
Status Register : Stat_2
Clear Type: W1C
Stat_2
4
5
Interrupt Properties Node-3
Node Name : 3
Status Register : Stat_3
Clear Type: Auto Clear
Stat_3
6
Interrupt Properties Node-4
Node Name : 4
Clear Type: W1C
is_expected : 1
Interrupt Properties Node-5
Node Name : 5
Clear Type: W1C
is_expected : 0
Interrupt Properties Node-6
Node Name : 6
Clear Type: Auto Clear
1
2
3
4
5
6
7
8
9
10
11

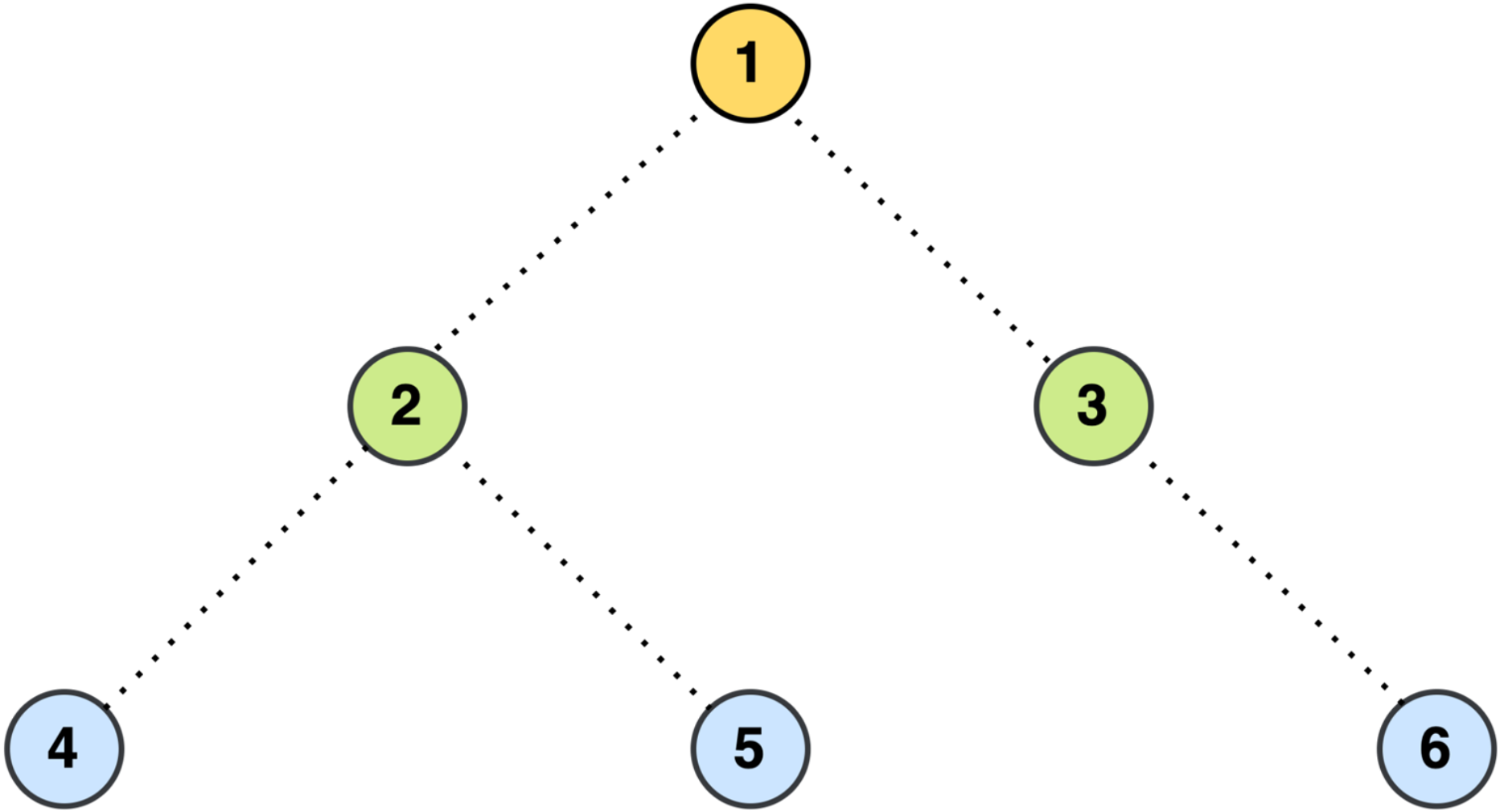
The Idea


Interrupt Properties Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
1
Interrupt Properties Node-2
Node Name : 2
Status Register : Stat_2
Clear Type: W1C
Stat_2
4
5
2
Interrupt Properties Node-3
Node Name : 3
Status Register : Stat_3
Clear Type: Auto Clear
Stat_3
6
3
7
8
9
10
11
Interrupt Properties Node-4
Node Name : 4
Clear Type: W1C
is_expected : 1
4
5
Interrupt Properties Node-5
Node Name : 5
Clear Type: W1C
is_expected : 0
6
Interrupt Properties Node-6
Node Name : 6
Clear Type: Auto Clear

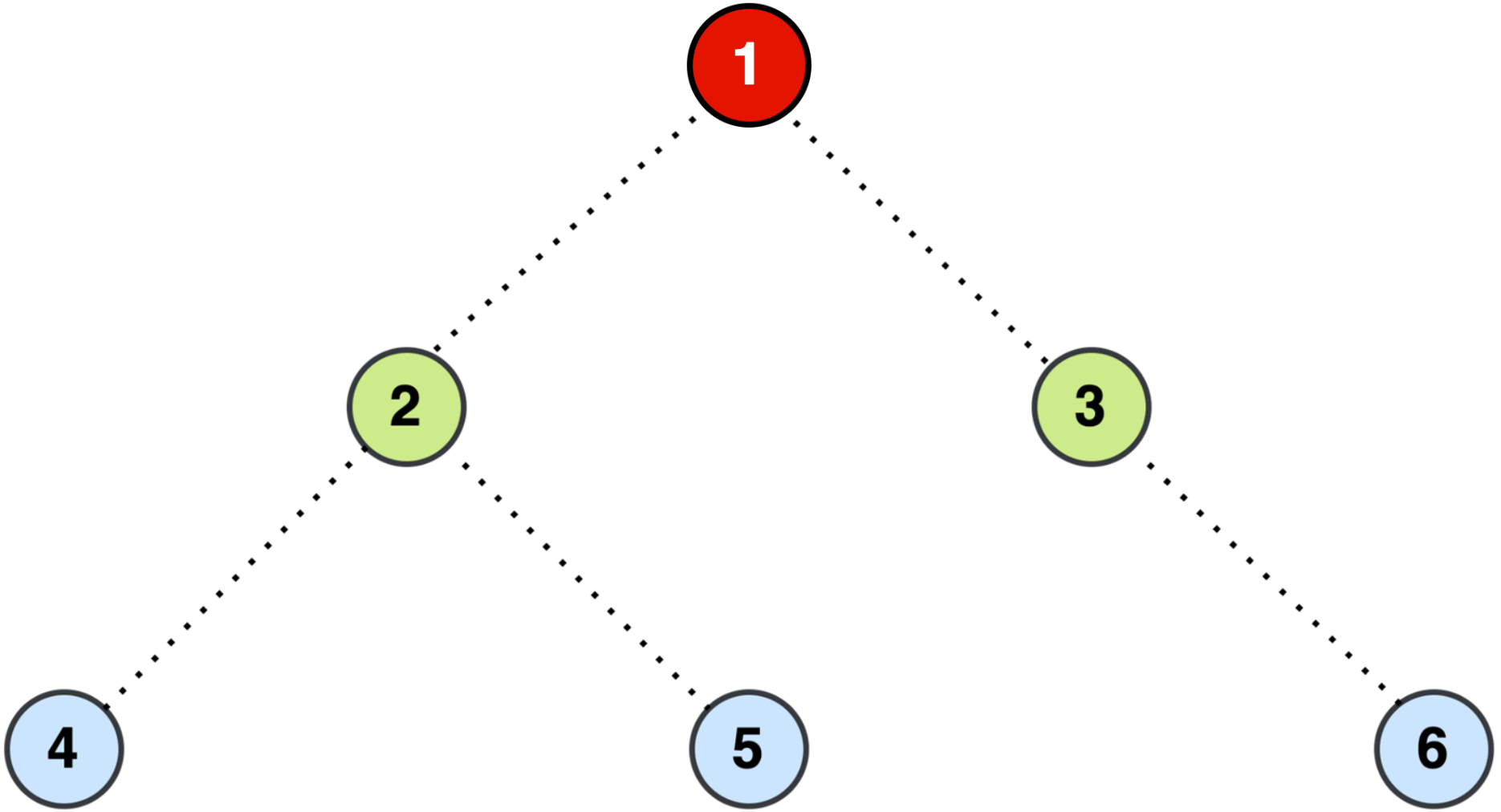
The Idea


Interrupt Properties Node-1
Node Name : 1
Status Register : Stat_1
is_top_interrupt : 1
Stat_1
2
3
Interrupt Properties Node-2
Node Name : 2
Status Register : Stat_2
Clear Type: W1C
Stat_2
4
5
Interrupt Properties Node-3
Node Name : 3
Status Register : Stat_3
Clear Type: Auto Clear
Stat_3
6
Interrupt Properties Node-4
Node Name : 4
Clear Type: W1C
is_expected : 1
Interrupt Properties Node-5
Node Name : 5
Clear Type: W1C
is_expected : 0
Interrupt Properties Node-6
Node Name : 6
Clear Type: Auto Clear
1
2
3
4
5
6
7
8
9
10
11

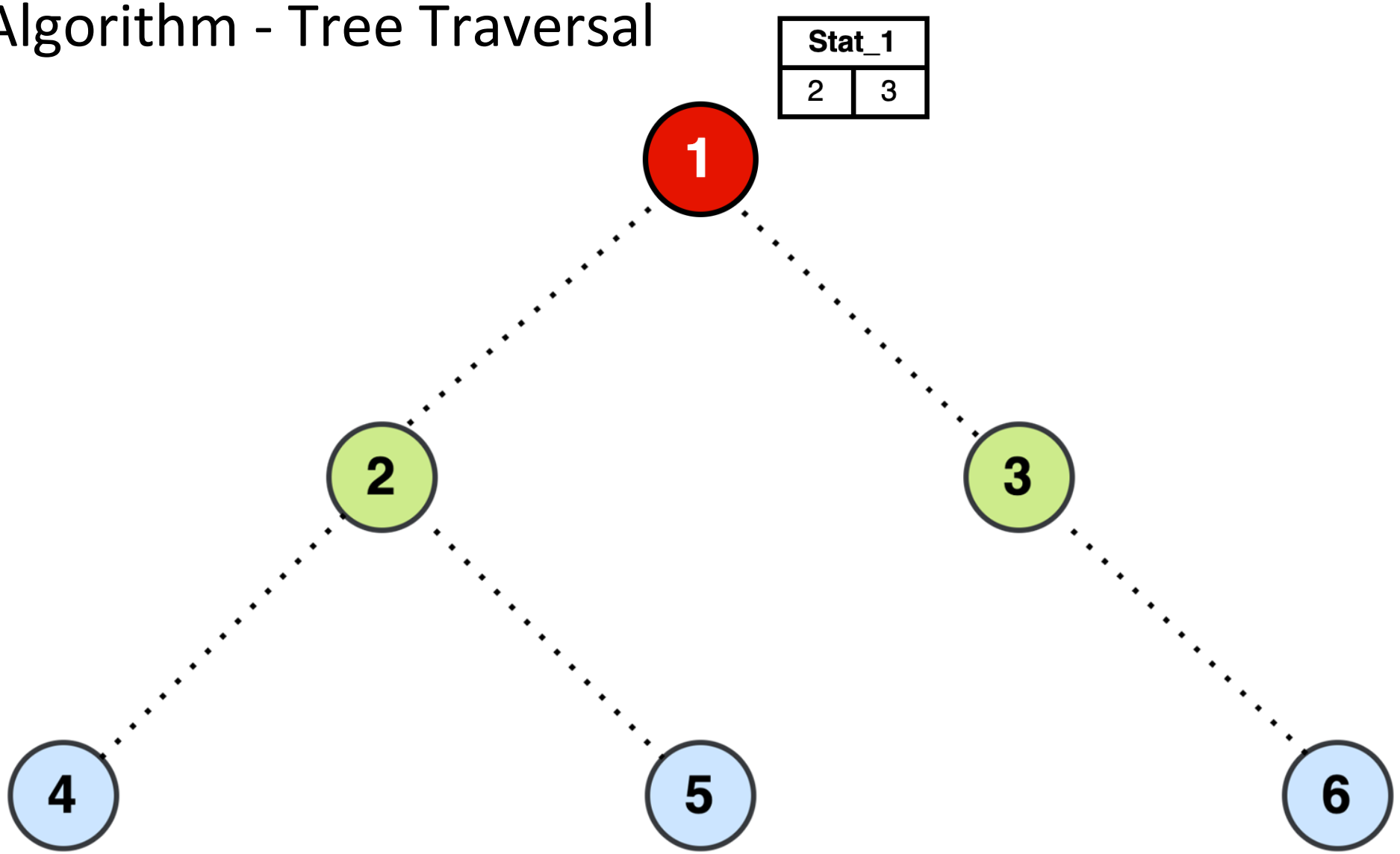
The Algorithm - Tree Traversal



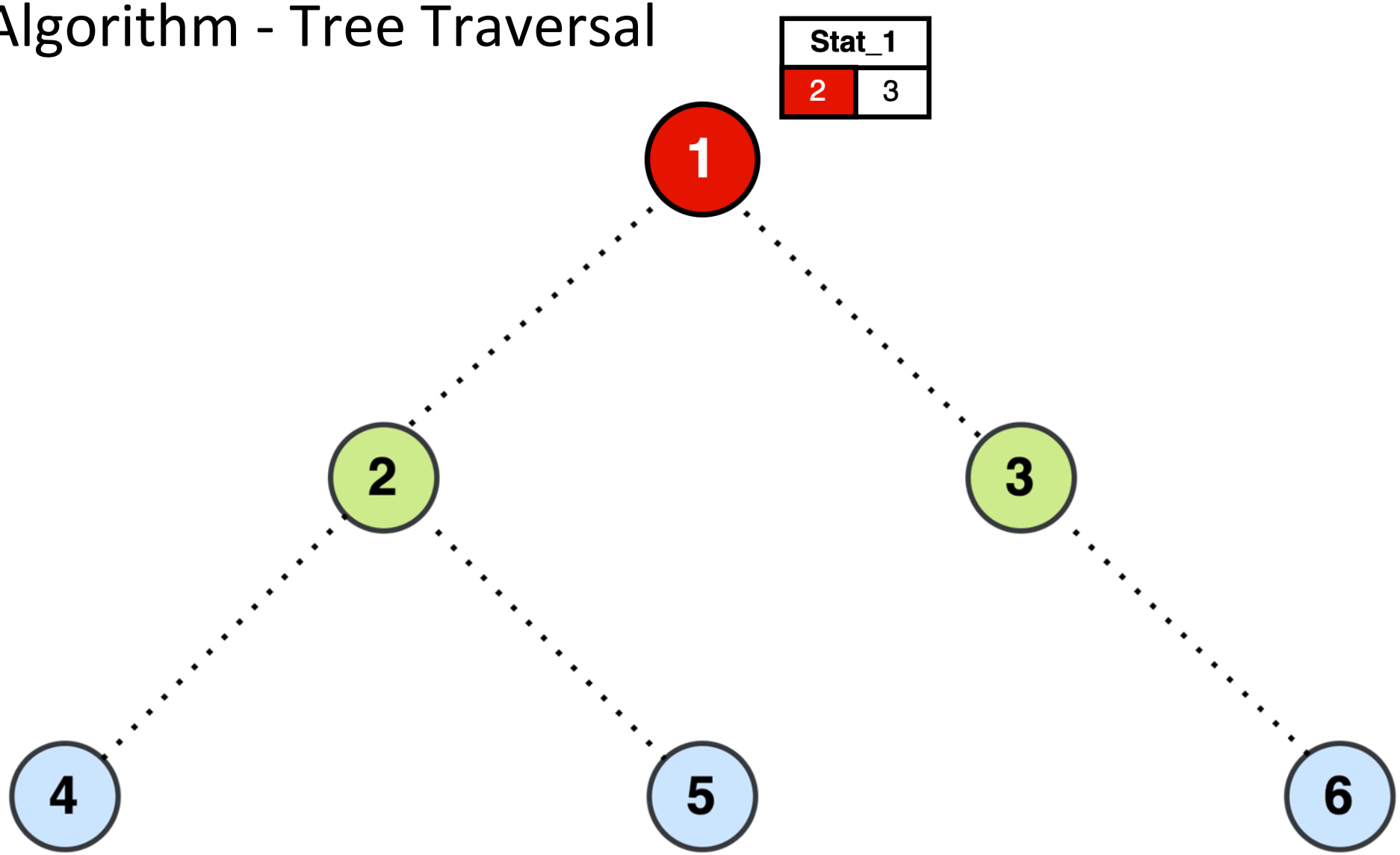
The Algorithm - Tree Traversal



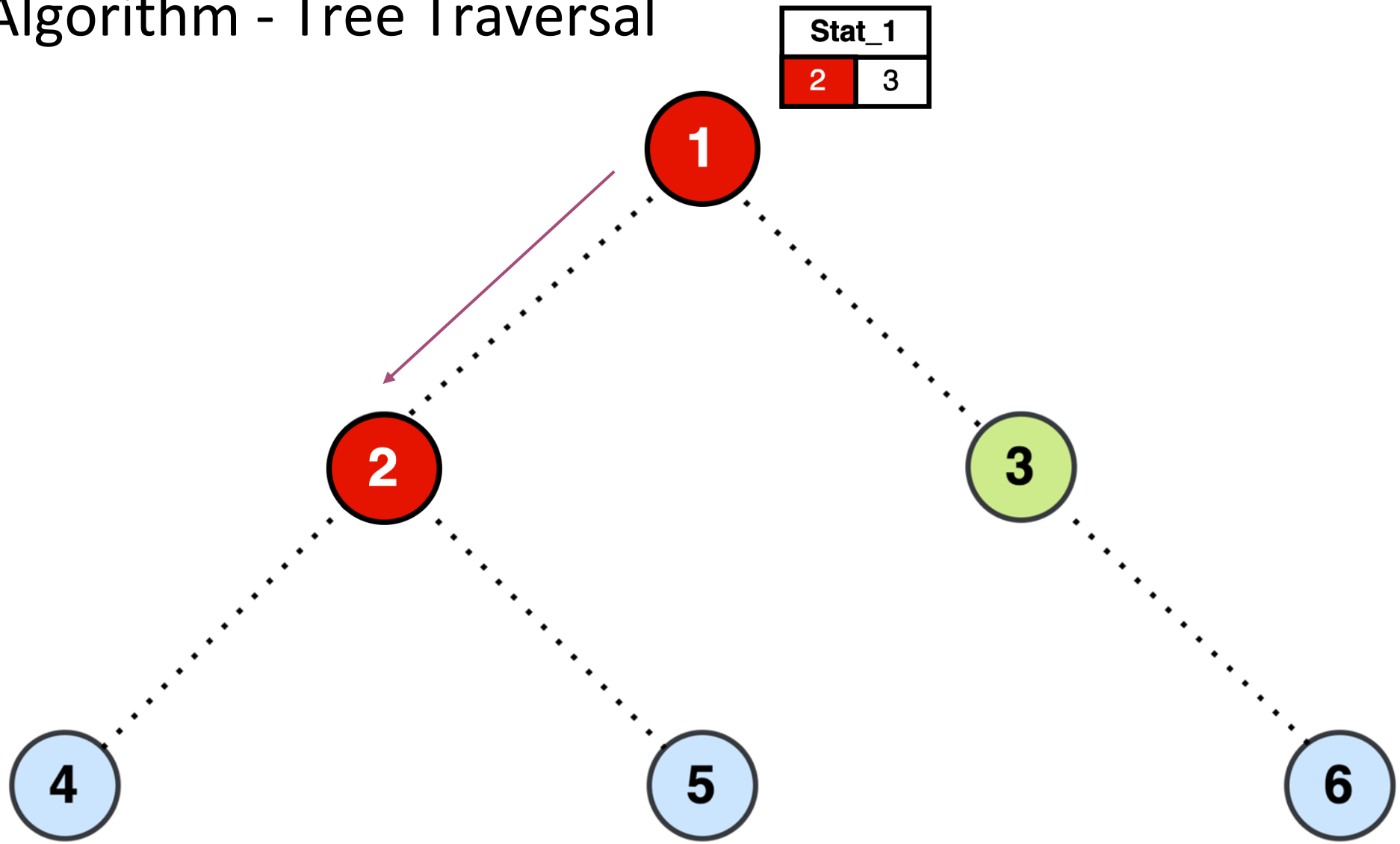
The Algorithm - Tree Traversal



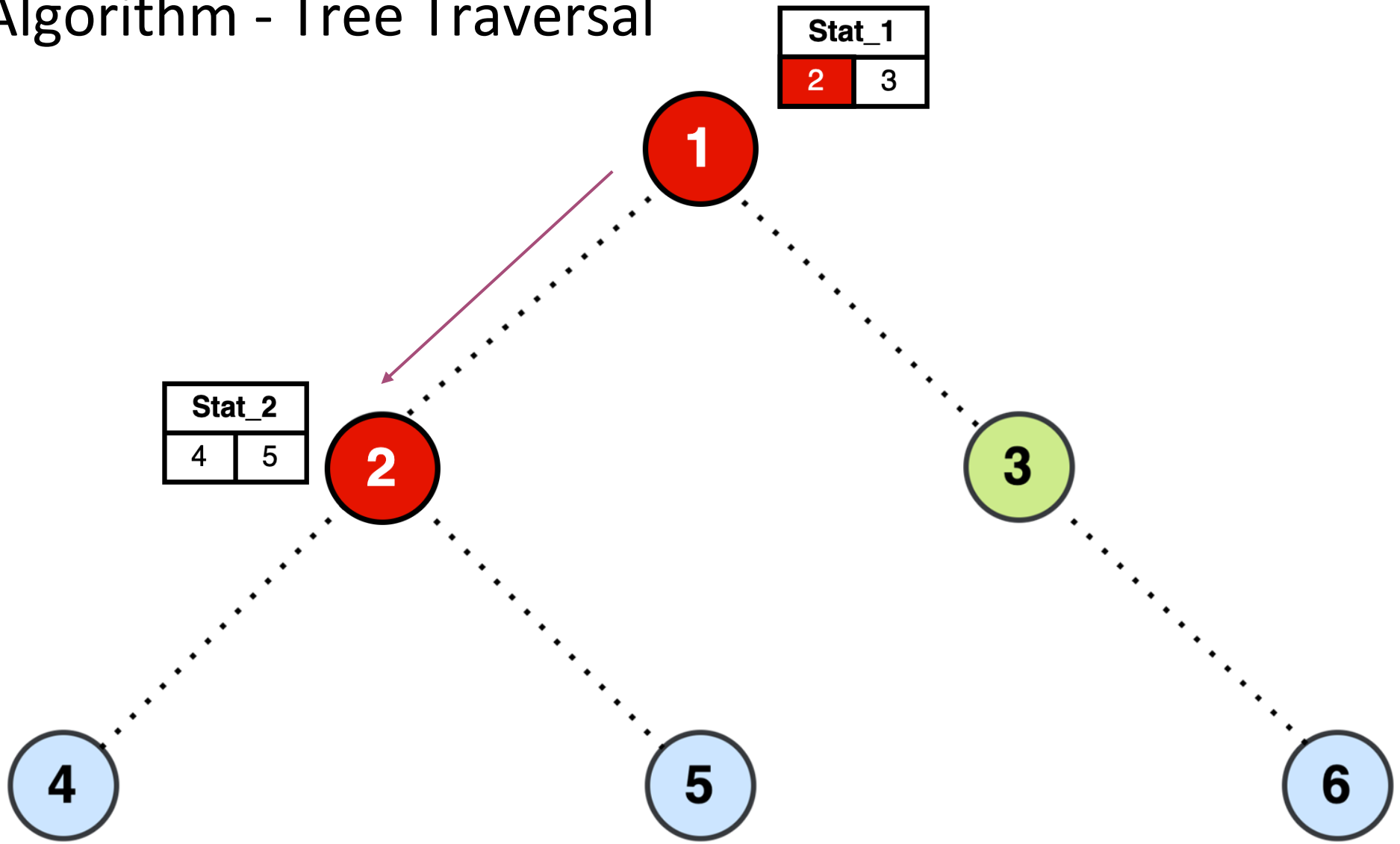
The Algorithm - Tree Traversal



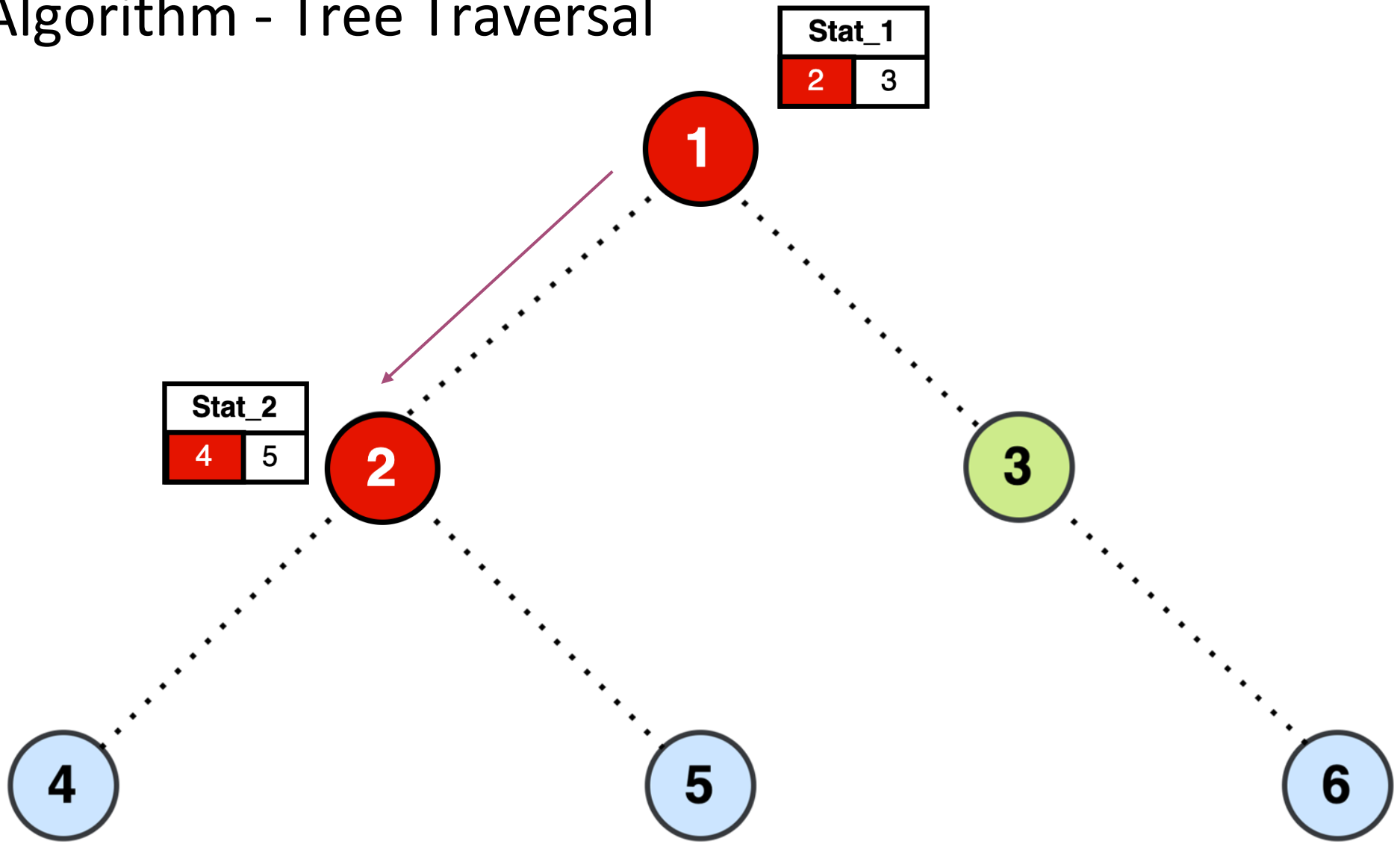
The Algorithm - Tree Traversal



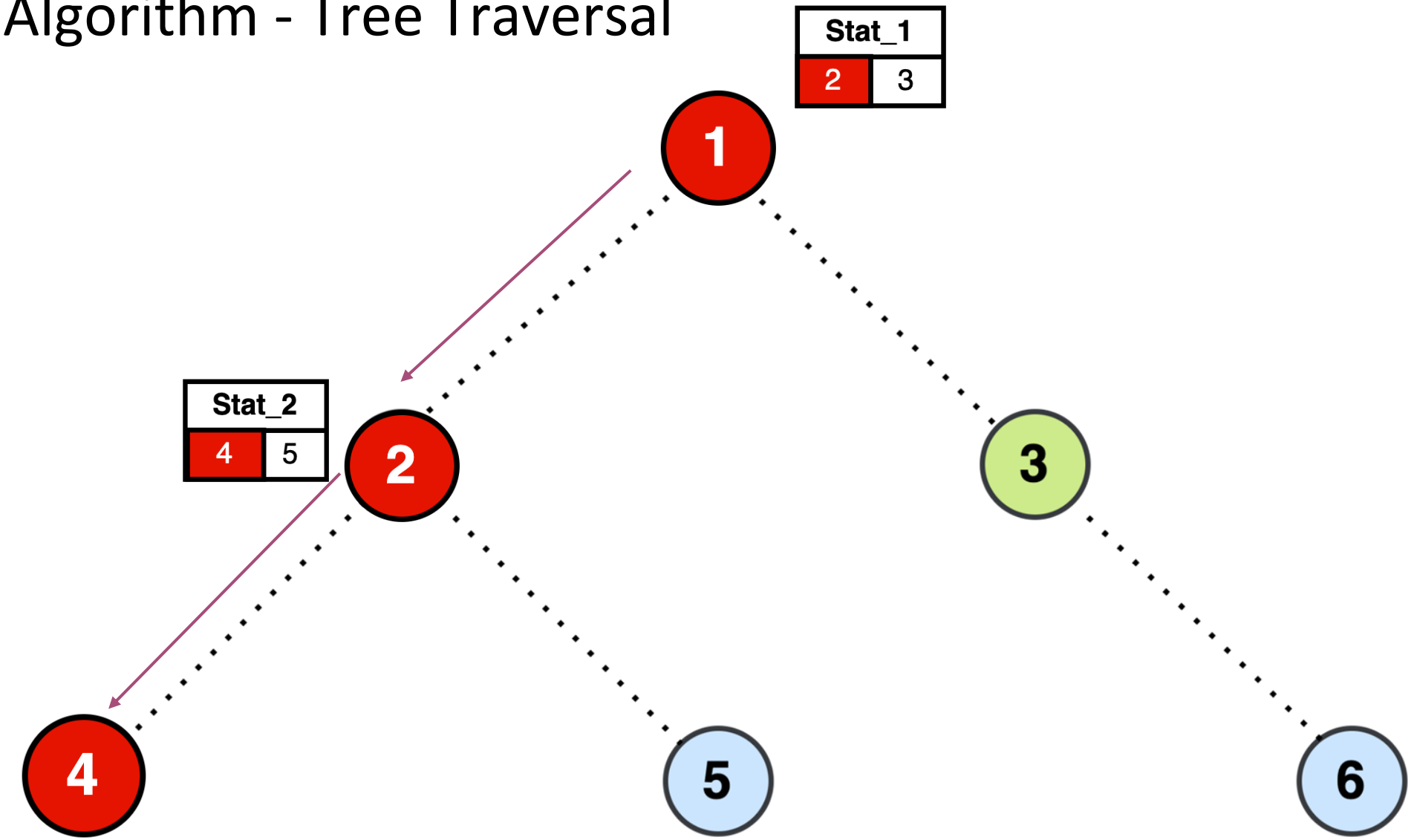
The Algorithm - Tree Traversal



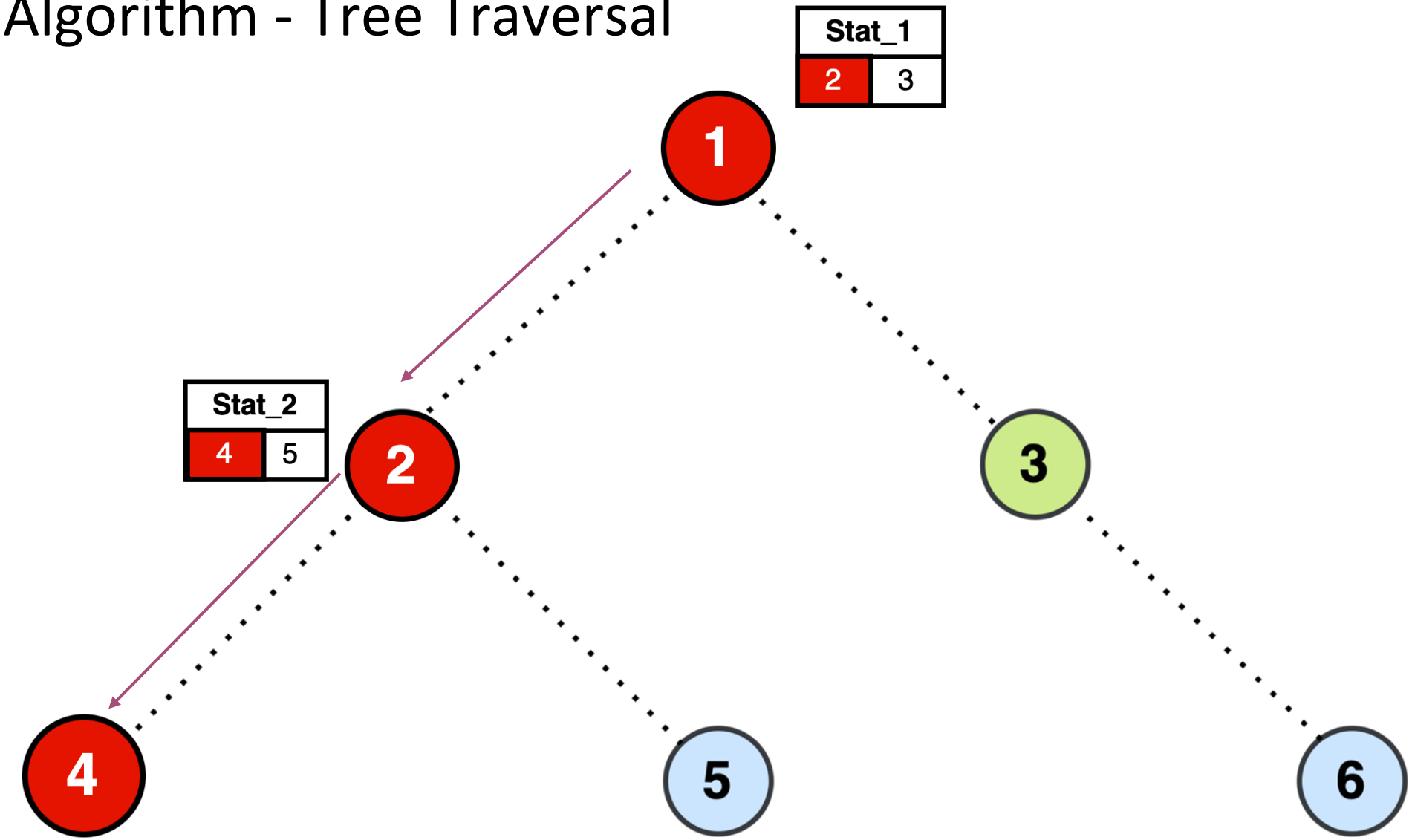
The Algorithm - Tree Traversal



The Algorithm - Tree Traversal

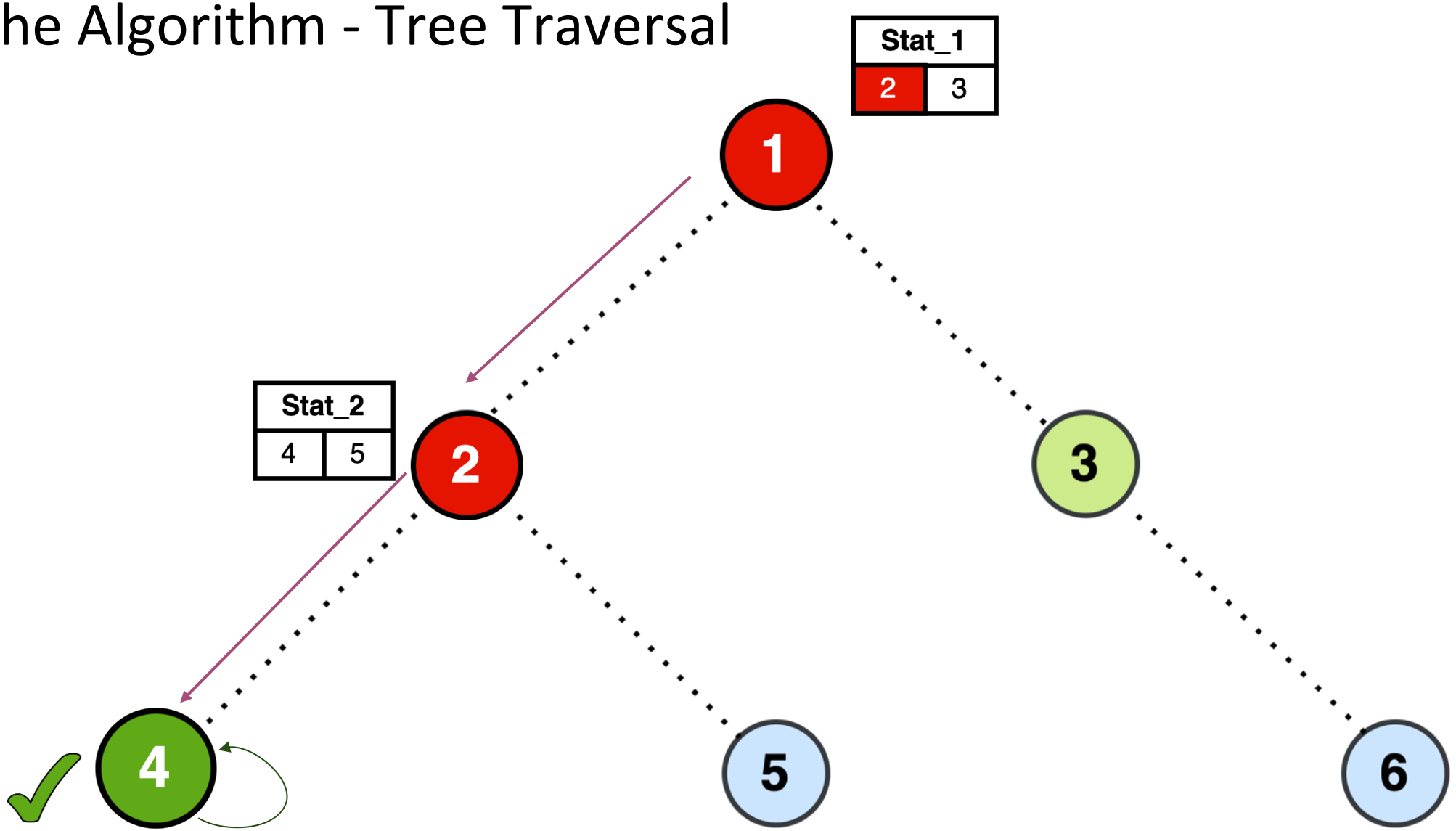


The Algorithm - Tree Traversal



is_expected = 1

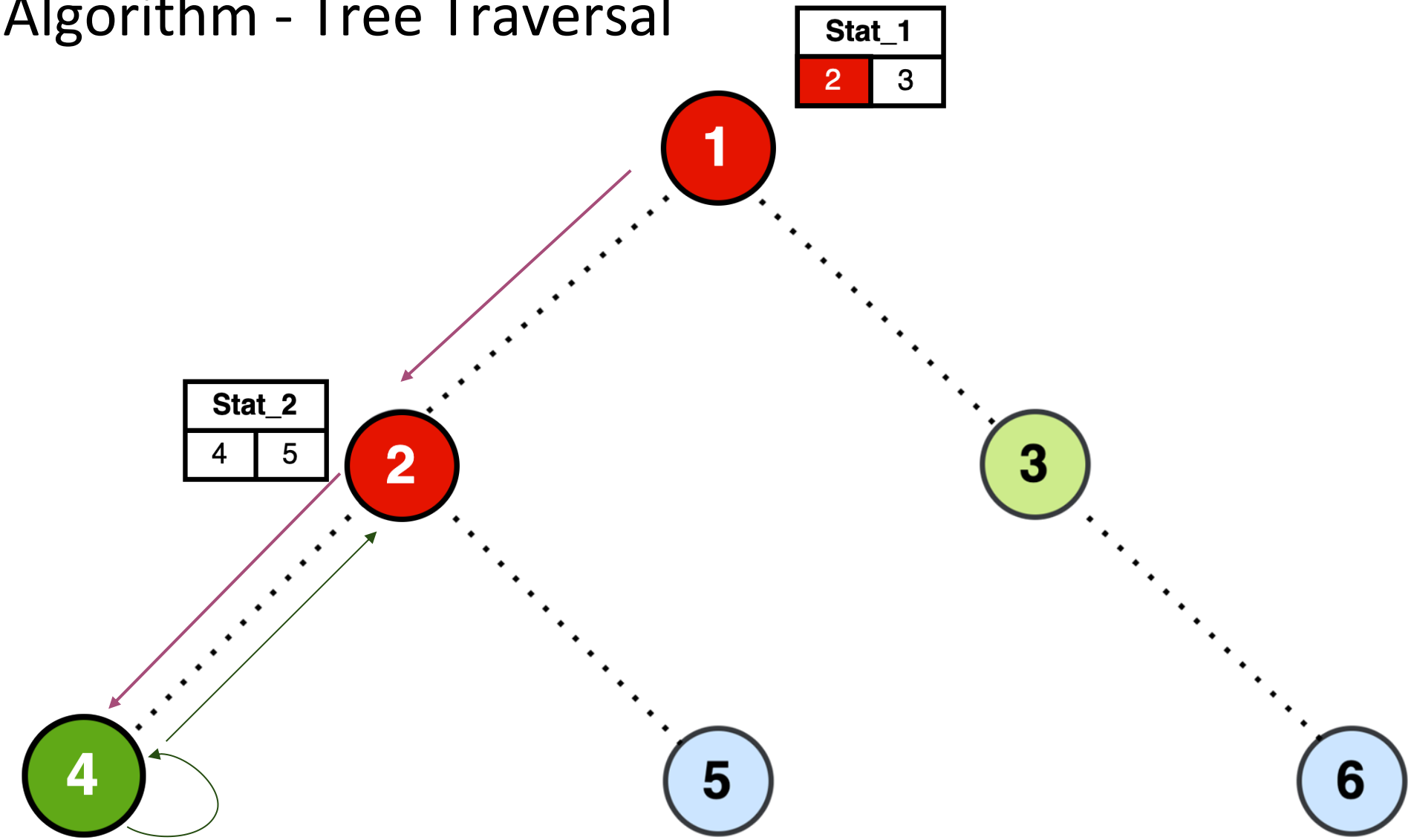
The Algorithm - Tree Traversal



is_expected = 1

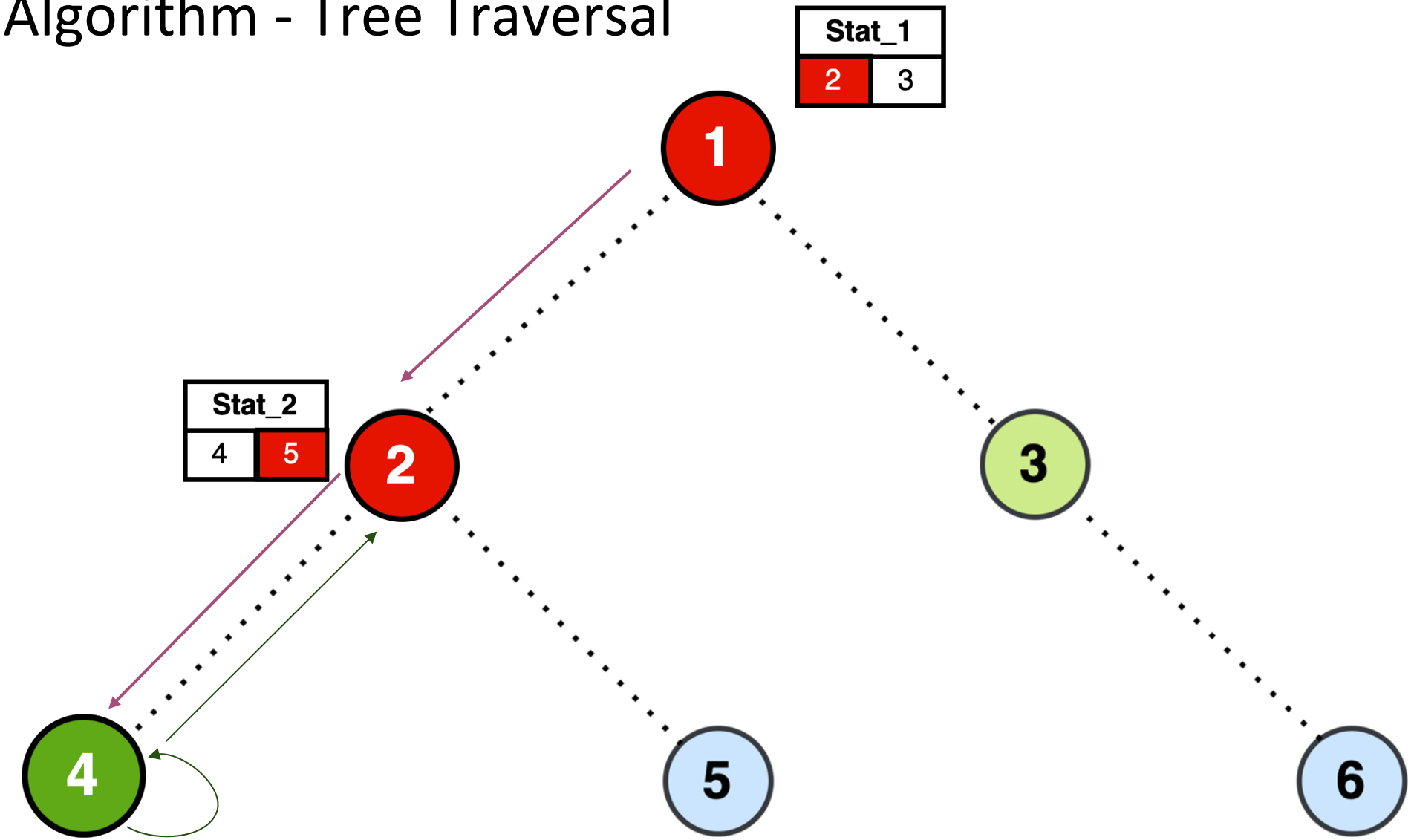
Expected interrupt asserted 1 --> 2 --> 4

The Algorithm - Tree Traversal



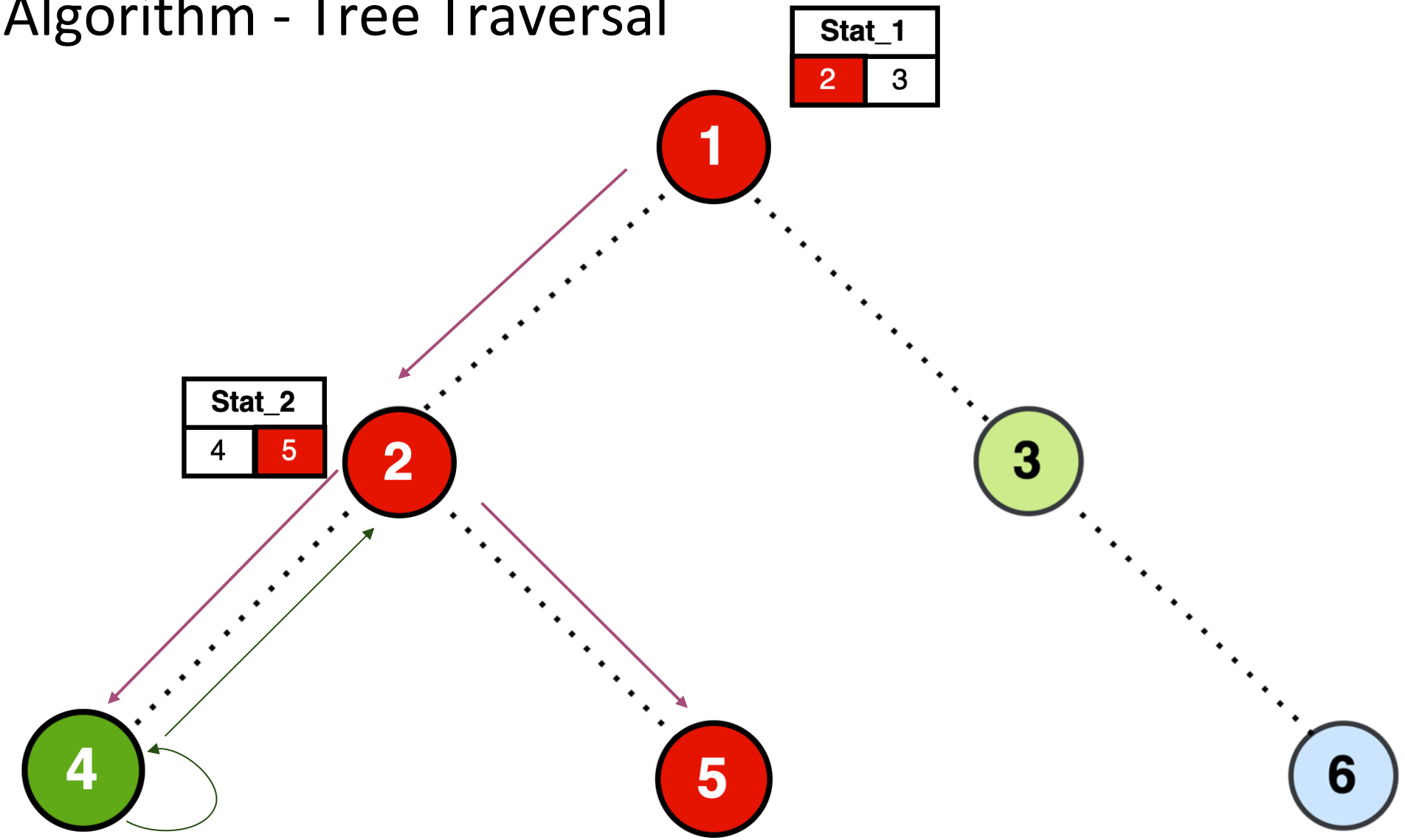
is_expected = 1

The Algorithm - Tree Traversal



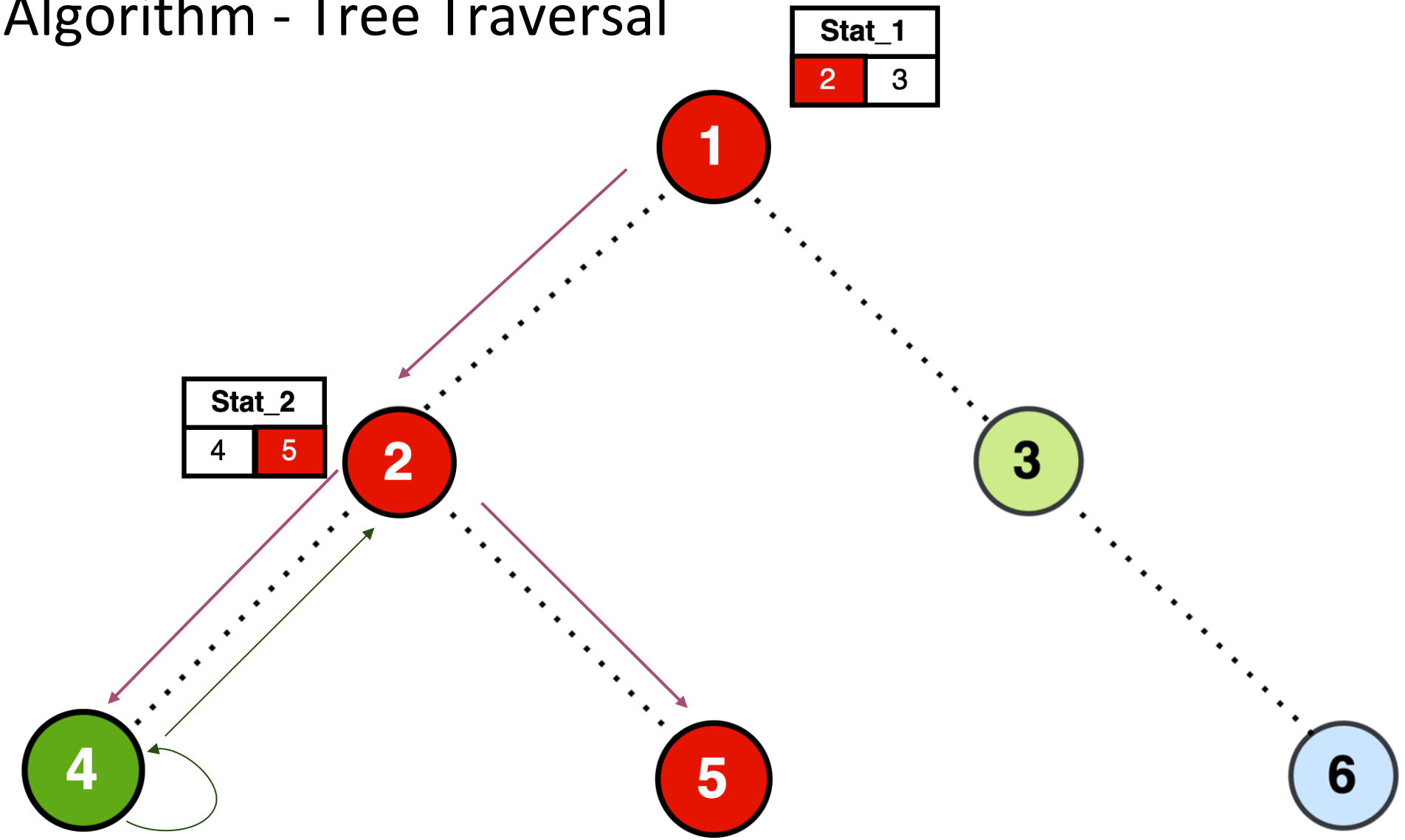
is_expected = 1

The Algorithm - Tree Traversal

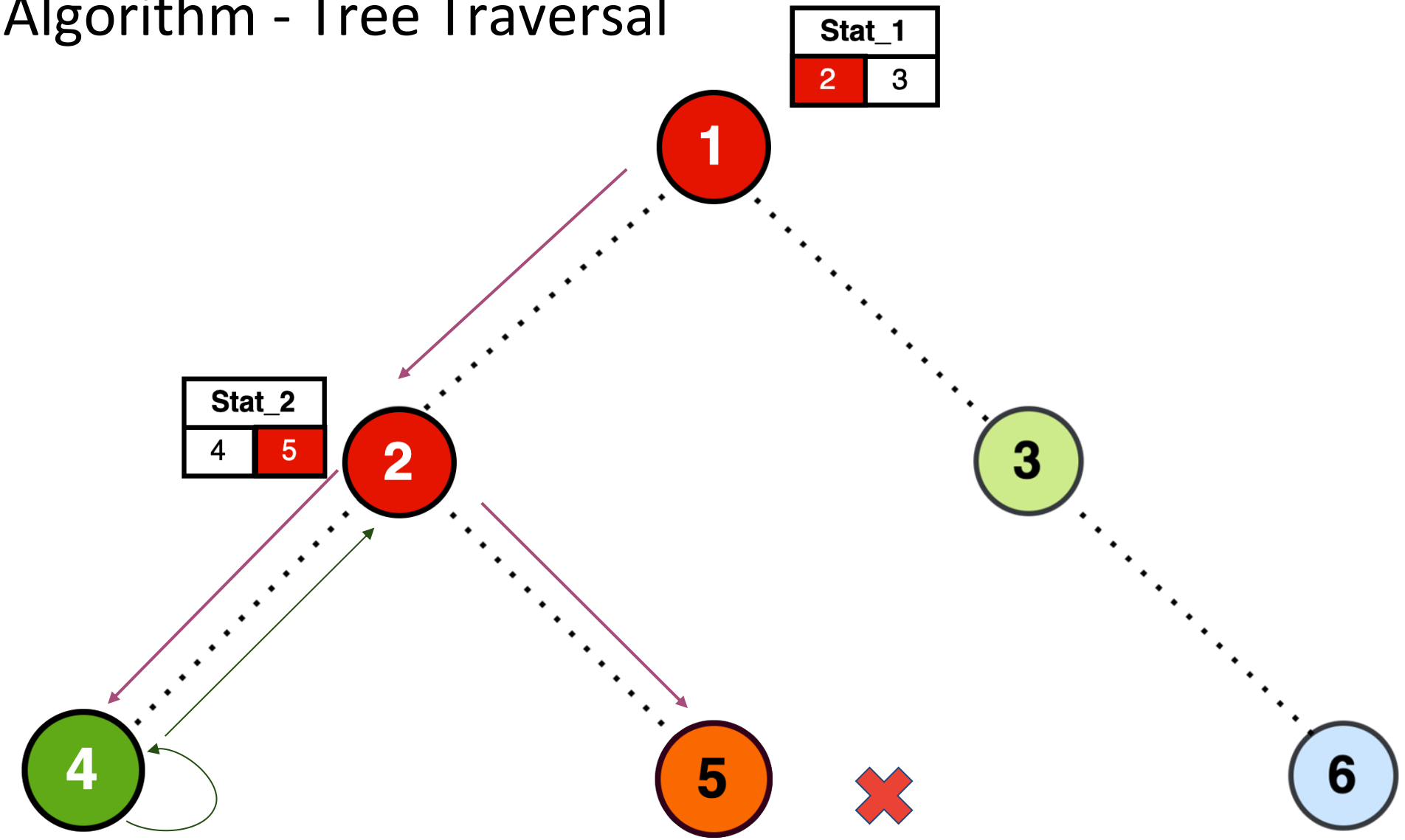


is_expected = 1

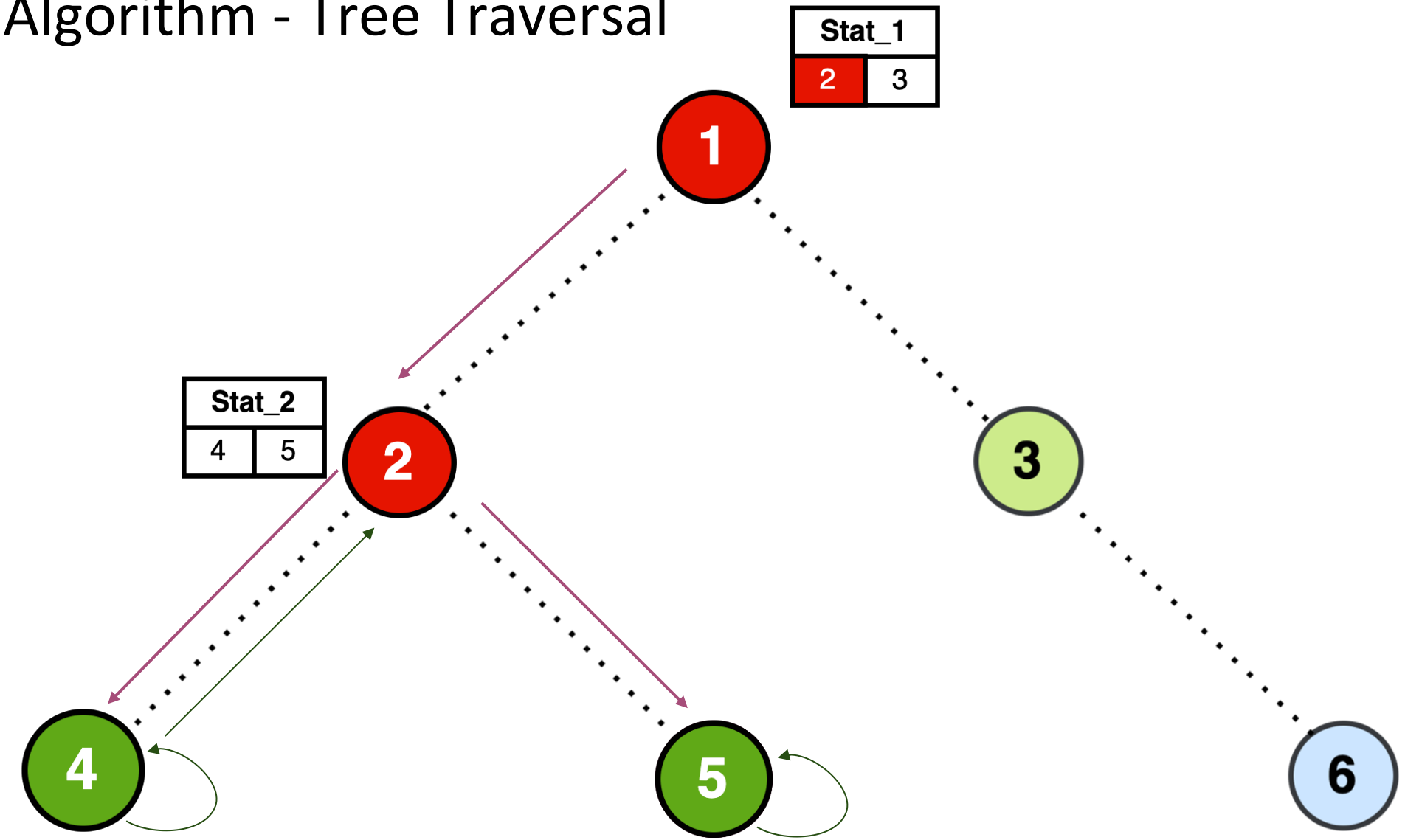
The Algorithm - Tree Traversal



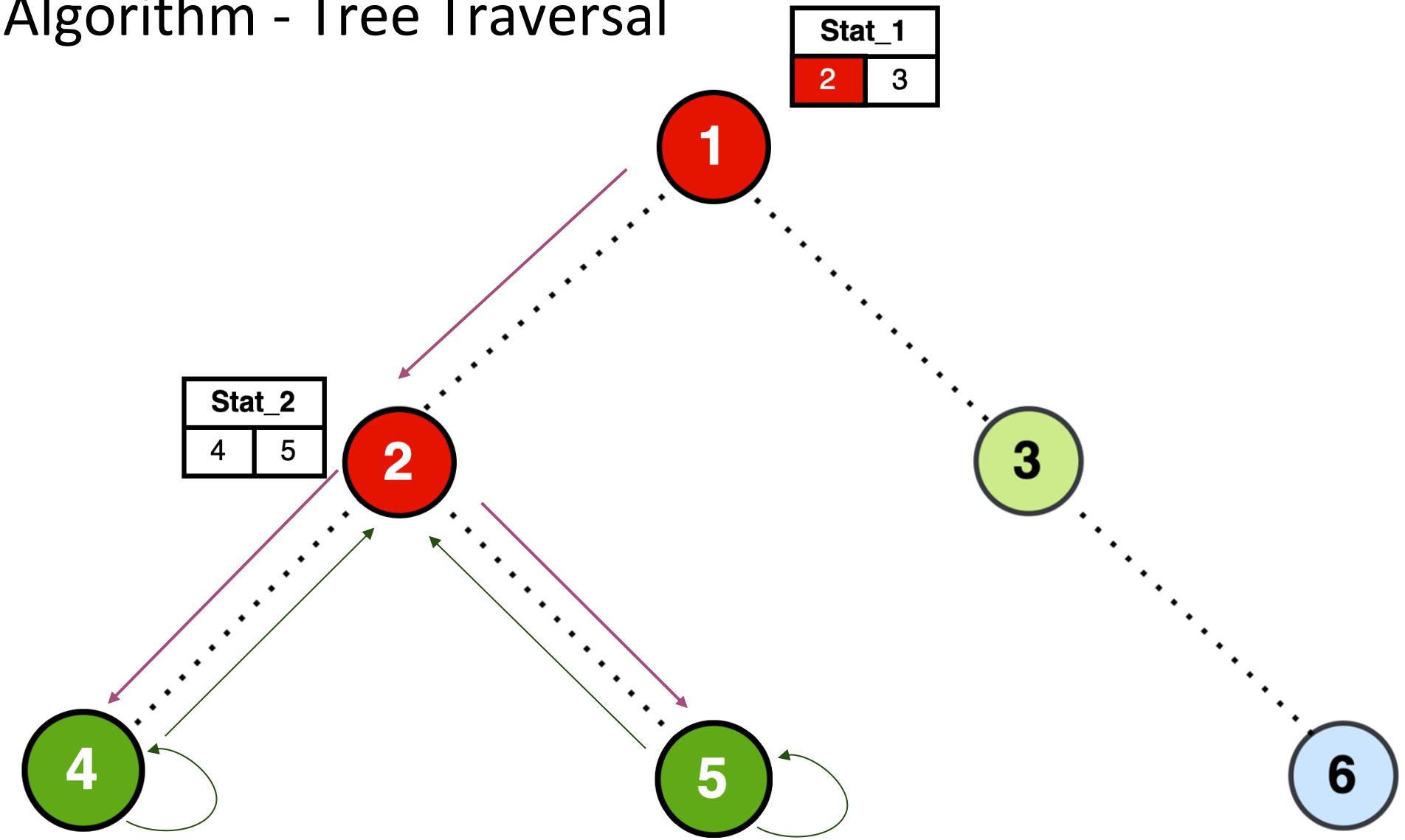
The Algorithm - Tree Traversal



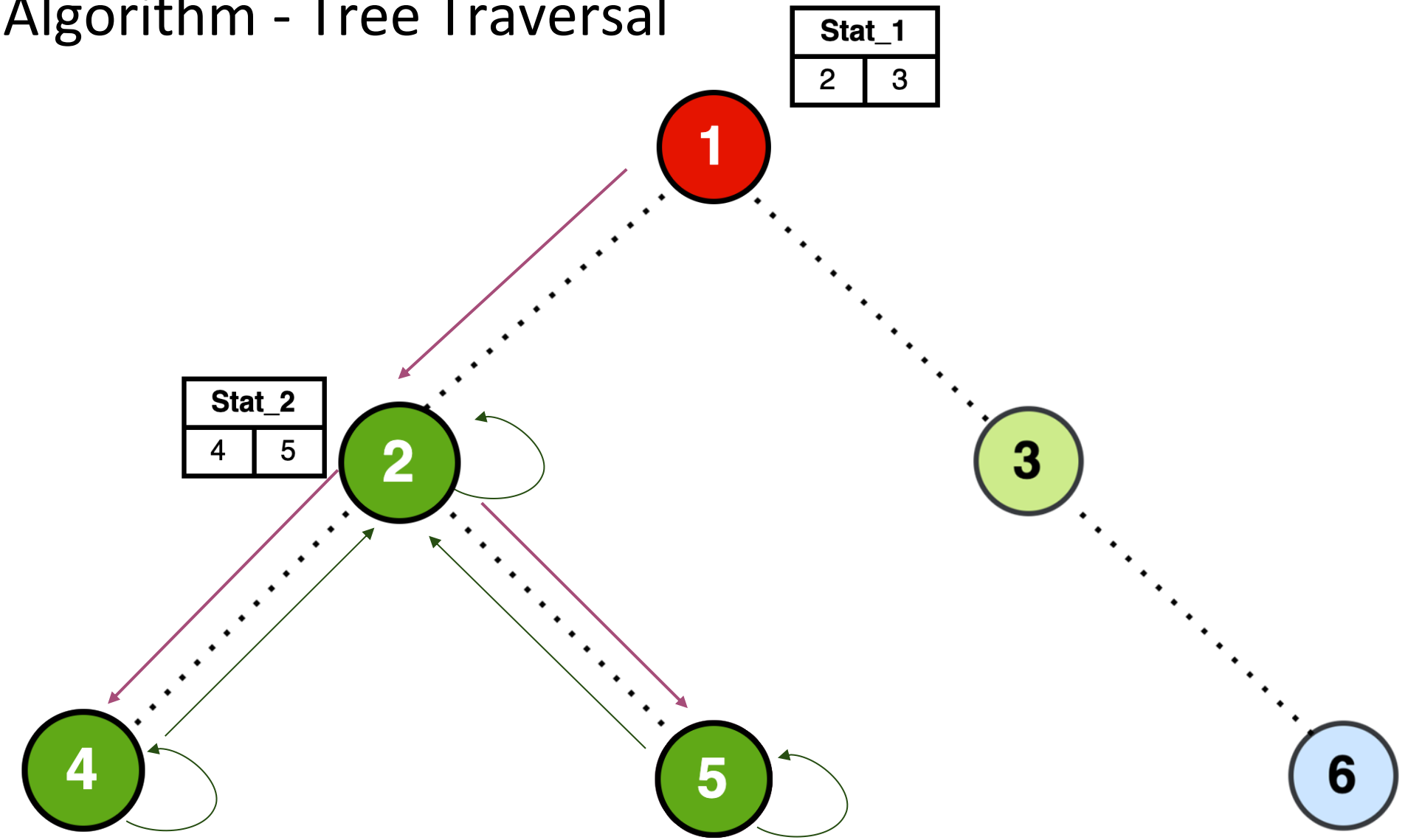
The Algorithm - Tree Traversal



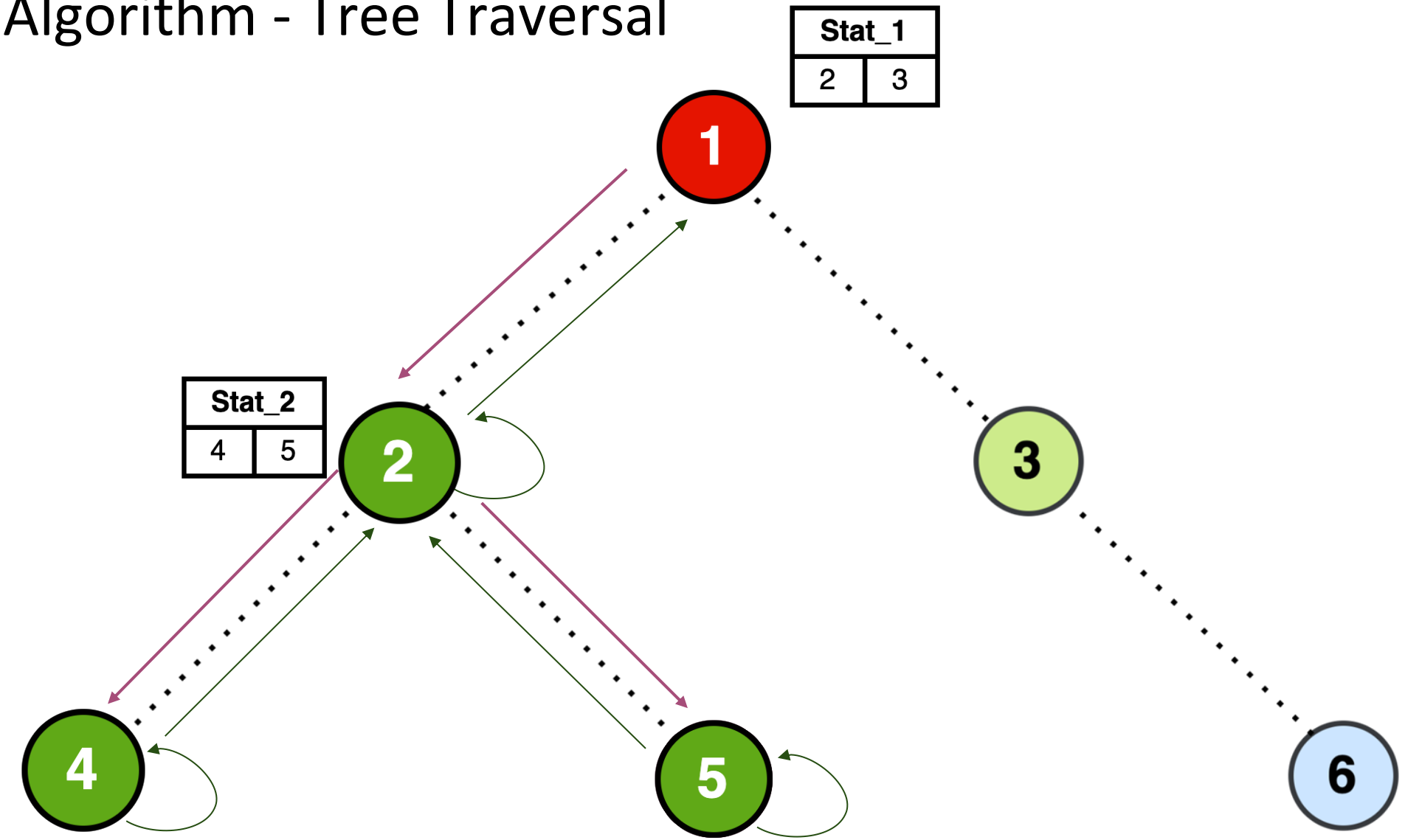
The Algorithm - Tree Traversal



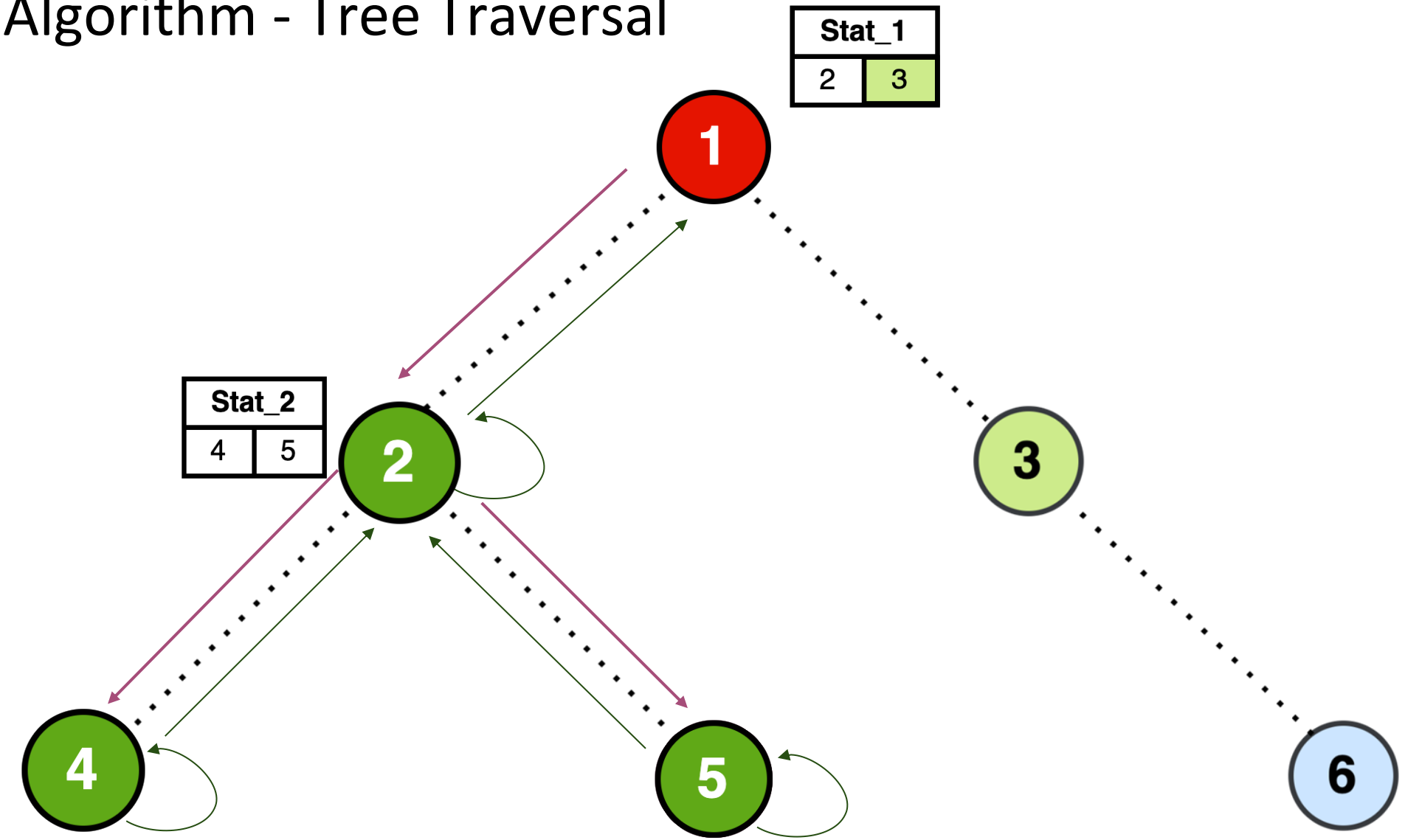
The Algorithm - Tree Traversal



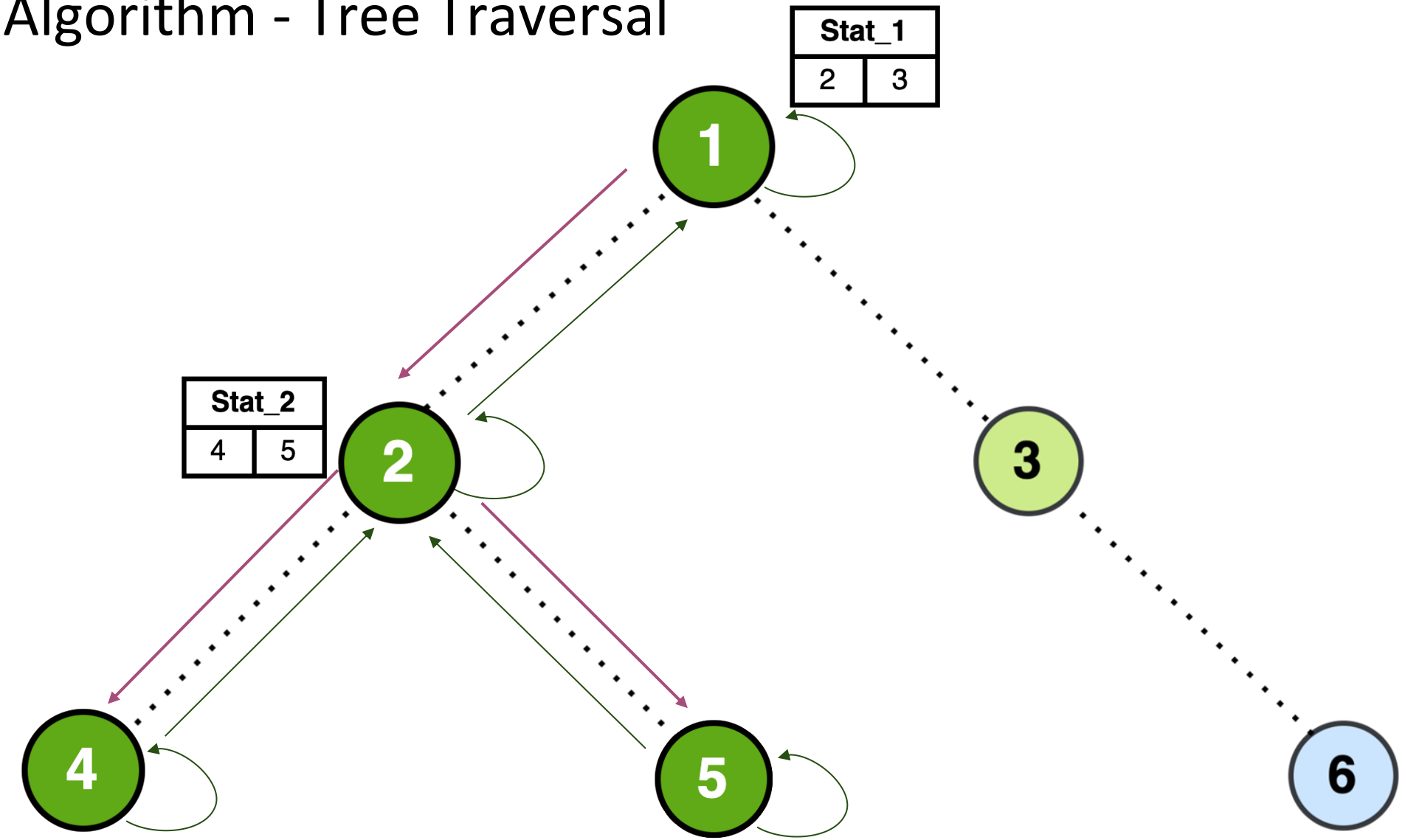
The Algorithm - Tree Traversal



The Algorithm - Tree Traversal



The Algorithm - Tree Traversal



UVM Implementation

There are three base classes :

- **Interrupt Node properties**
  - Contains all the properties related to a particular node like
    - Node name
    - Is_top_interrupt
    - Clear Type
    - Is_expected etc.

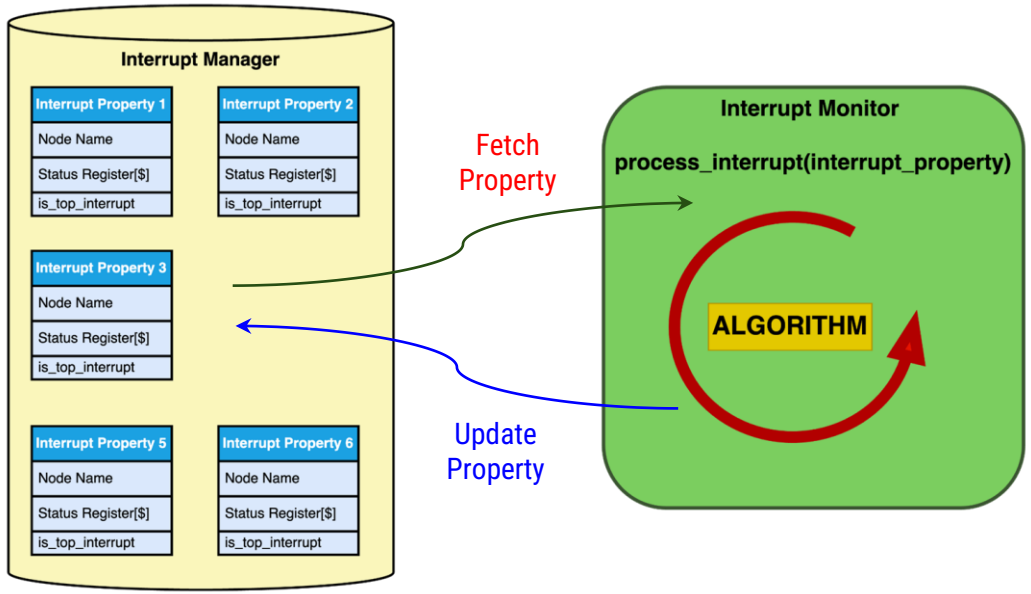
generic_interrupt_properties
node_name
status_register_q
clear type
parent_node_handle
child_node_handle[$]
is_expected
.....

- **Interrupts Manager**
  - Contains all the interrupt nodes created and links them.
    - This class contains all of the APIs required to create a node and assign corresponding properties.

UVM Implementation

There are three base classes :

- **Interrupt Monitor**
  - Contains a handle to interrupt manager (hence all of the interrupt nodes)
  - Starts to monitor all of the top(pin) interrupts and initiates the process (as described previously) for each top interrupt when asserted.



Exclusive Checks

Initialization:

- **Interrupt Tree Sanity Check:** Performs a sanity check by ensuring every interrupt node correctly maps to its designated status register.

Runtime Monitoring:

- **Hierarchical Assertion Check:** The hierarchy is validated by confirming an asserted parent always has an asserted child, which ensures proper signal propagation.
- **Leaf Node Validation:** Leaf nodes are examined to verify that their expected assertion status aligns with the actual state.

Exclusive Checks

Runtime Monitoring:

- **Auto-Clear and Re-assertion:** Interrupts are cleared based on their defined clear type, and a subsequent check confirms the corresponding status register field and thus the interrupt has been cleared.
- **Status Register Monitoring:** An uncleared status register in the monitored interrupt branch triggers a re-initiation of the entire process.

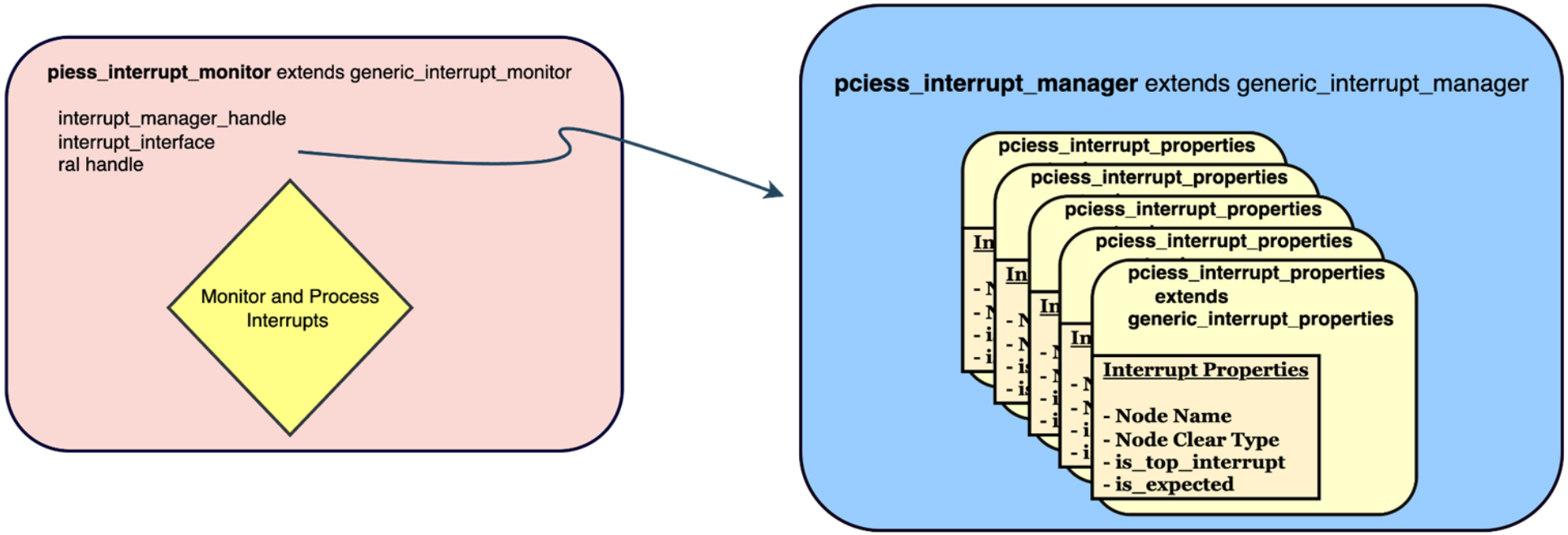
End of Simulation Analysis:

- **Expected Assertion Verification:** The check confirms that all interrupts configured as "expected" were asserted during the simulation.
- **Global Status Register Check:** Finally, the framework verifies all interrupt status registers are clear, guaranteeing a clean end-of-test state.

Proof of Concept

PCIe Subsystem is used as a test vehicle to check the validity of gISR. Reason being -

- 150+ interrupts present in PCIeSS generated collectively from in-house blocks as well as from encapsulated third party IPs.
- 5+ levels of interrupt aggregation points present in this sub-system.



Issues Reported

- Unexpected Interrupt Assertion
- Incorrect Interrupt Routing
  - Interrupt didn’t trigger when expected
  - Interrupt reached different node than expected
- Asserted Interrupts not getting cleared
  - No support for clearing
  - Status Register Attributes mismatch between RTL & RDL.
- Unconnected Interrupts
- Same interrupt multiple times
  - One time Expected interrupt triggered multiple times.

Future Scope

- Input google sheet/text file parsing for auto generation of interrupt nodes.
- Google sheet based covered interrupts dumping to aid tracking and review.
- Pulse interrupt checking mechanism with enhanced pulse-width checks.
- Support for counter based interrupts' assertions check.
- Test notify bit support for structural testing.
- Support for ML-based Interrupt behavior Anomaly Detection.

Thank you

Questions ?