Data Structures & Algorithms

Subodh Kumar

(subodh@iitd.ac.in, Bharti 422)

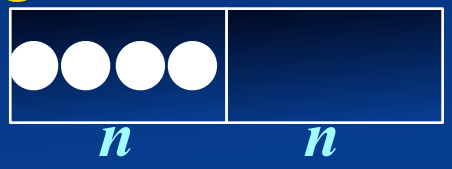
Dept of Computer Sc. & Engg.

Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth

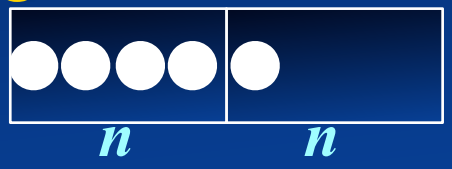
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



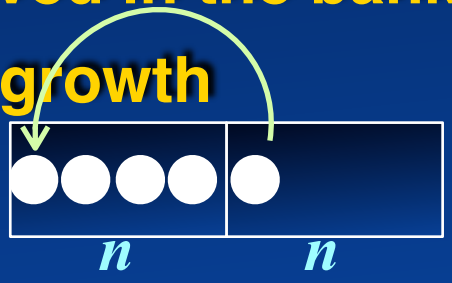
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



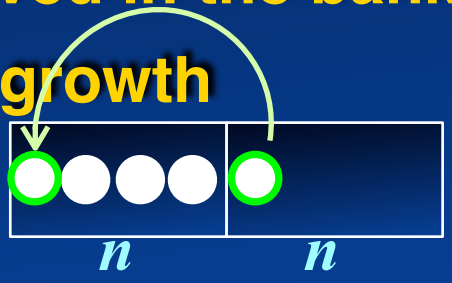
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



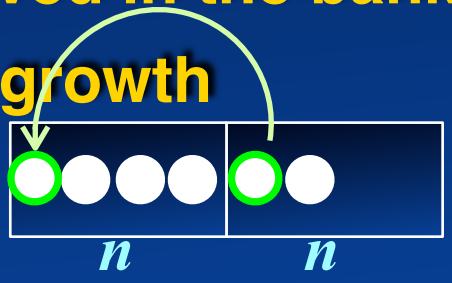
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



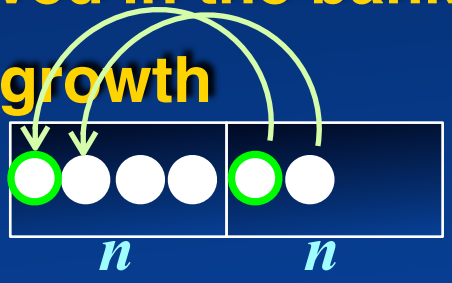
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



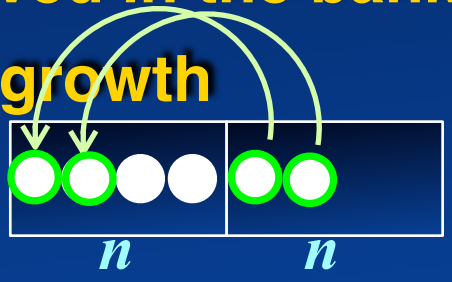
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



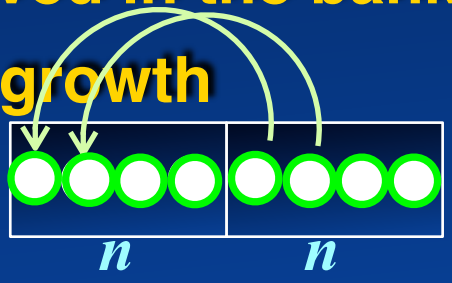
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



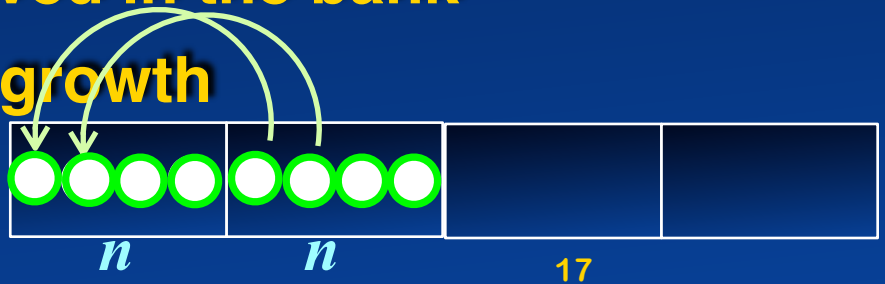
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



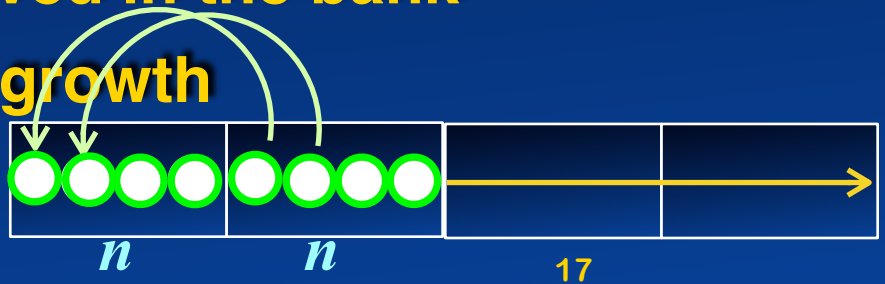
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



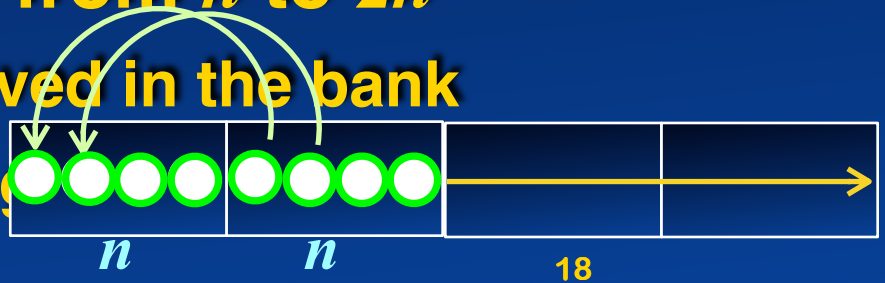
Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count cost of each usage = 2 steps
  - 1 unit pays for the usage
  - 1 unit banked for later utilization
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth



Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count 3 units for usage of each index i
  - 1 unit pays for the usage
  - 1 unit banked for later copy of index i
  - 1 unit banked for later copy of index i - $n/2$
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size $



Amortized Analysis

- Vector:
  - Array, with size doubled when filled
- Count 3 units for usage of each index i
  - 1 unit pays for the usage
  - 1 unit banked for later copy of index i
  - 1 unit banked for later copy of index i - $n/2$
  - Each operation is still O(1)
- When doubling the size from n to $2n$
  - There must be n units saved in the bank
  - Use it to pay for the size growth

Insertion in 2-3 Tree

Insertion in 2-3 Tree

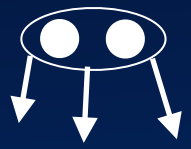
- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree

Insertion in 2-3 Tree

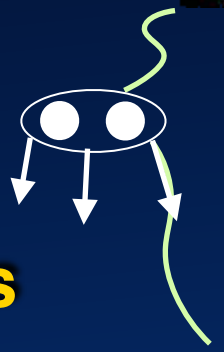
- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits

Insertion in 2-3 Tree

- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits

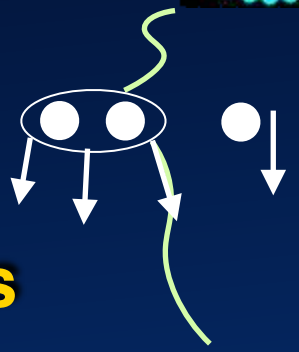


Insertion in 2-3 Tree



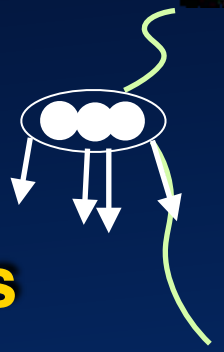
- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits

Insertion in 2-3 Tree



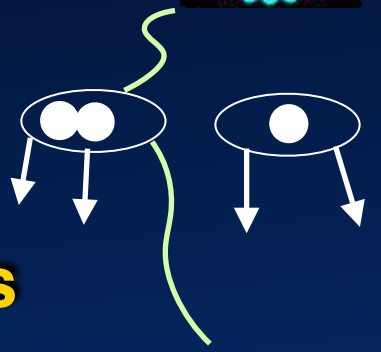
- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits

Insertion in 2-3 Tree



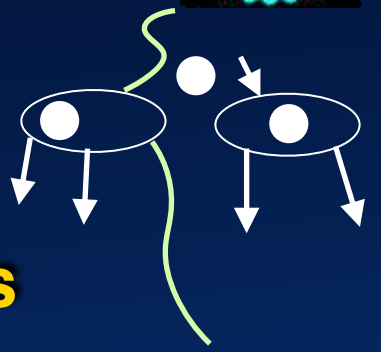
- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits

Insertion in 2-3 Tree



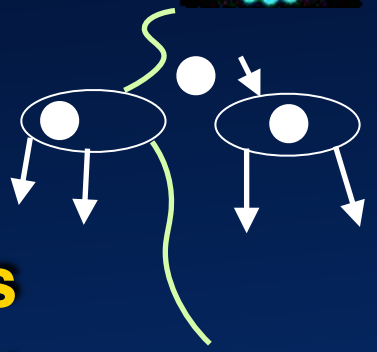
- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits

Insertion in 2-3 Tree



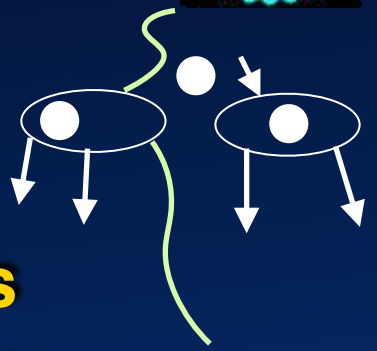
- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits

Insertion in 2-3 Tree



- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits
- After each insertion, $\Delta\Phi \leq 1$ – *No. of splits*
  - Insert into a 3-node
  - Splits it into two 2-nodes: -1
  - At most one node becomes 3-node

Insertion in 2-3 Tree

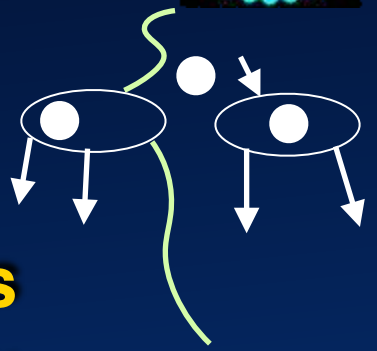


- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits
- After each insertion, $\Delta\Phi \leq 1$ – *No. of splits*
  - Insert into a 3-node
  - Splits it into two 2-nodes: -1
  - At most one node becomes 3-node
- After n insertions:
  - $\Delta\Phi \leq n$ - No. of splits = Φ
    - Assume $\Phi = 0$ initially (empty tree)

  $\Rightarrow$No. of splits $\leq n - \Phi \leq n$

  $\Rightarrow$averages to O(1) split per insert

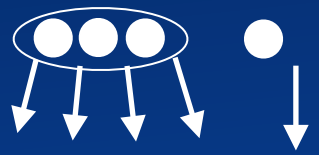
Insertion in 2-3 Tree



- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits
- After each insertion, $\Delta\Phi \le 1$ – *No. of splits*
  - Insert into a 3-node
  - Splits it into two 2-nodes: -1
  - At most one node becomes 3-node
- After n insertions:
  - $\Delta\Phi \le n$ - No. of splits = Φ
    - Assume $\Phi = 0$ initially (empty tree)

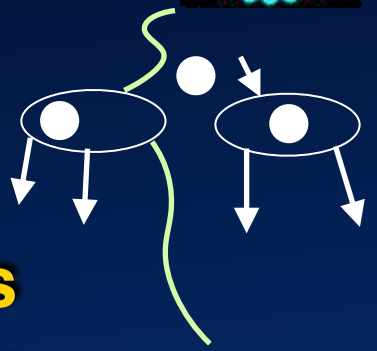
  $\Rightarrow$No. of splits $\le n - \Phi \le n$

  $\Rightarrow$averages to O(1) split per insert



2-4 Tree

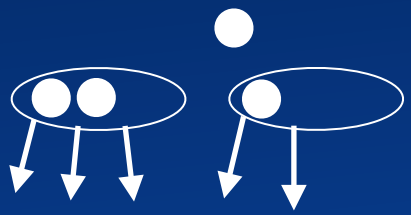
Insertion in 2-3 Tree



- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits
- After each insertion, $\Delta\Phi \le 1$ – *No. of splits*
  - Insert into a 3-node
  - Splits it into two 2-nodes: -1
  - At most one node becomes 3-node
- After n insertions:
  - $\Delta\Phi \le n$ - No. of splits = Φ
    - Assume $\Phi = 0$ initially (empty tree)

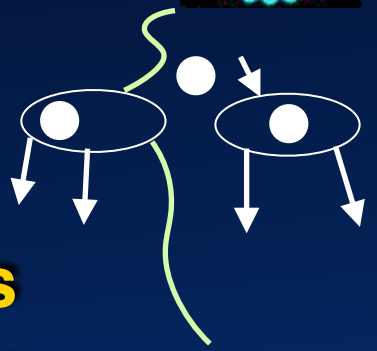
  $\Rightarrow$No. of splits $\le n - \Phi \le n$

  $\Rightarrow$averages to O(1) split per insert



2-4 Tree

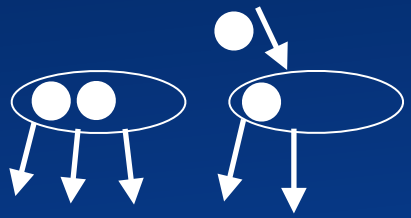
Insertion in 2-3 Tree



- Let tree potential Φ = *No. of 3-nodes*
  - No. of b-nodes in an a-b tree
- One insertion can cause up to lg n splits
- After each insertion, $\Delta\Phi \le 1$ – *No. of splits*
  - Insert into a 3-node
  - Splits it into two 2-nodes: -1
  - At most one node becomes 3-node
- After n insertions:
  - $\Delta\Phi \le n$ - No. of splits = Φ
    - Assume $\Phi = 0$ initially (empty tree)

  $\Rightarrow$No. of splits $\le n - \Phi \le n$

  $\Rightarrow$averages to O(1) split per insert



2-4 Tree

Insert/Delete in 2-5 Tree

- 5-nodes can split on insertion
- 2 nodes can merge on deletion
- Φ = No. of 5 nodes + No. of 2 nodes
- After insertion, ΔΦ ≤ 1 – *No. of splits*
- After deletion, ΔΦ ≤ 1 – 2x *No. of merges*
  - ΔΦ ≤ n – *No. of splits* – 2x *No. of merges* = Φ
    - Assume Φ = 0 initially (empty tree)

  ⇒No. of splits + merges ≤ n – Φ ≤ n

  ⇒averages to O(1) split per *update*
- Holds for any a-b tree if $b > 2a$

Insert/Delete in 2-5 Tree

- 5-nodes can split on insertion
- 2 nodes can merge on deletion
- Φ = No. of 5 nodes + No. of 2 nodes
- After insertion, ΔΦ ≤ 1 – *No. of splits*
- After deletion, ΔΦ ≤ 1 – 2x *No. of merges*
  - ΔΦ ≤ n – *No. of splits* – 2x *No. of merges* = Φ
    - Assume Φ = 0 initially (empty tree)

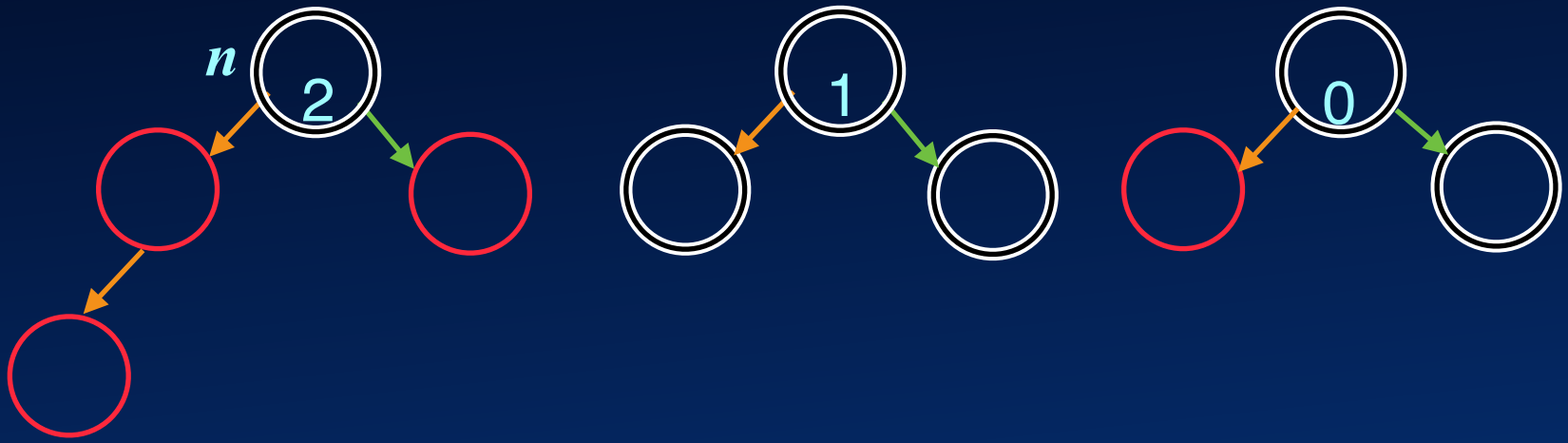
  ⇒No. of splits + merges ≤ n – Φ ≤ n

  ⇒averages to O(1) split per *update*
- Holds for any a-b tree if $b > 2a$

$$b = 2a + 1$$

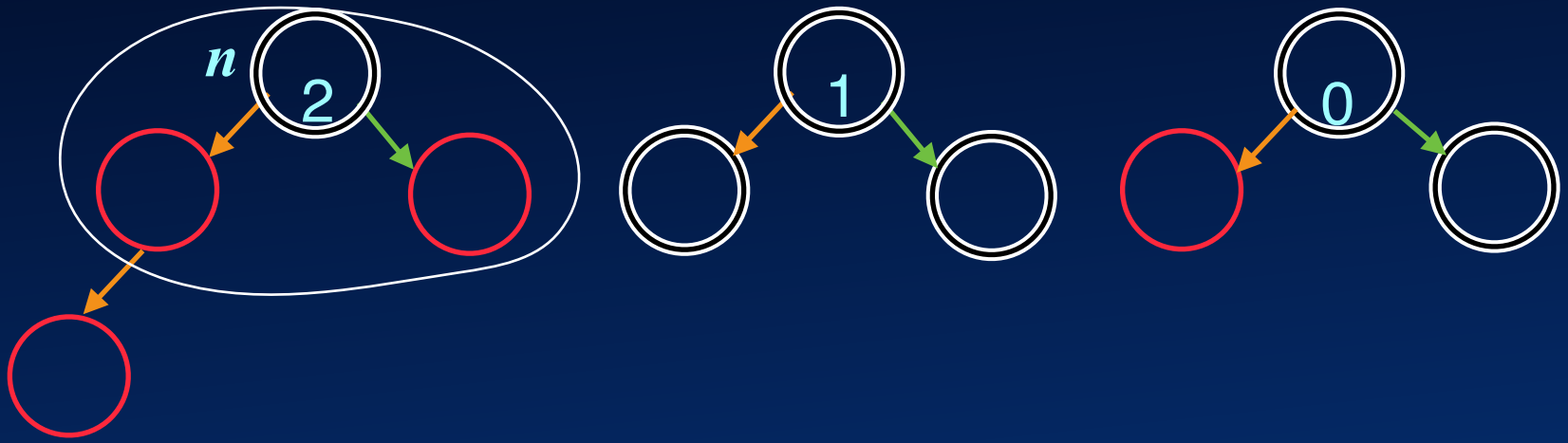
R-B Tree Color changes

Insert



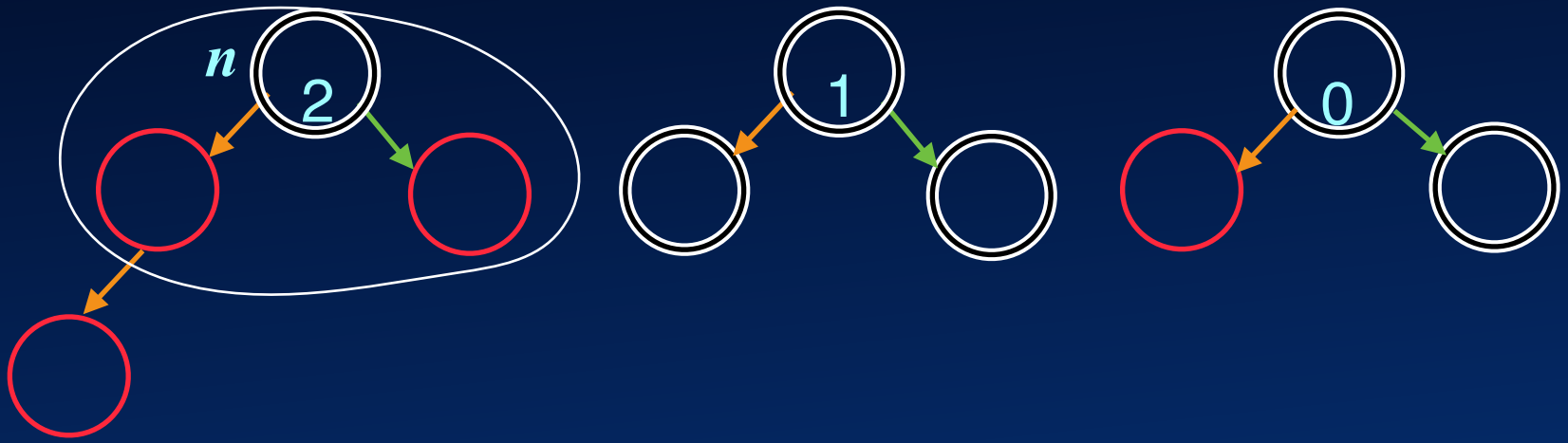
R-B Tree Color changes

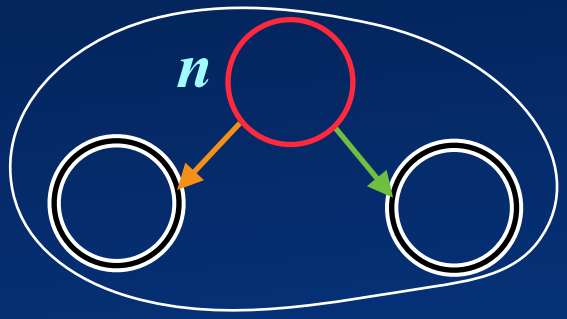
Insert



R-B Tree Color changes

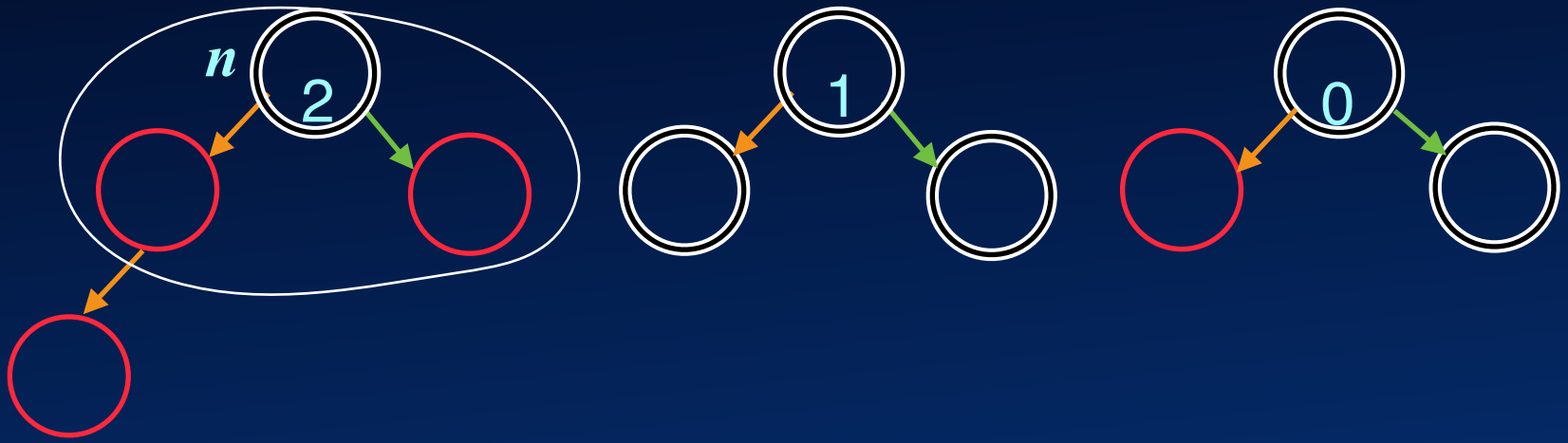
Insert

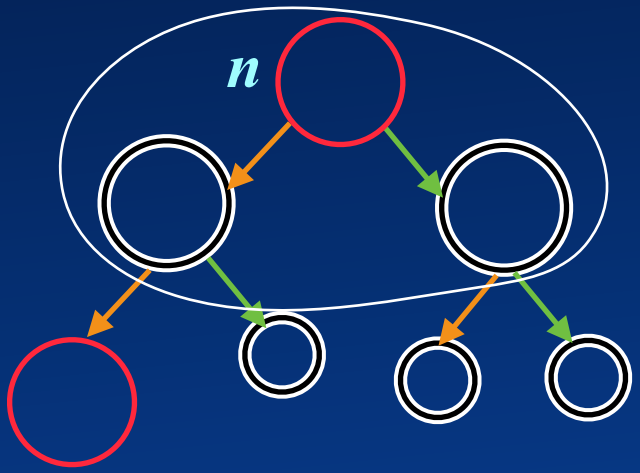




R-B Tree Color changes

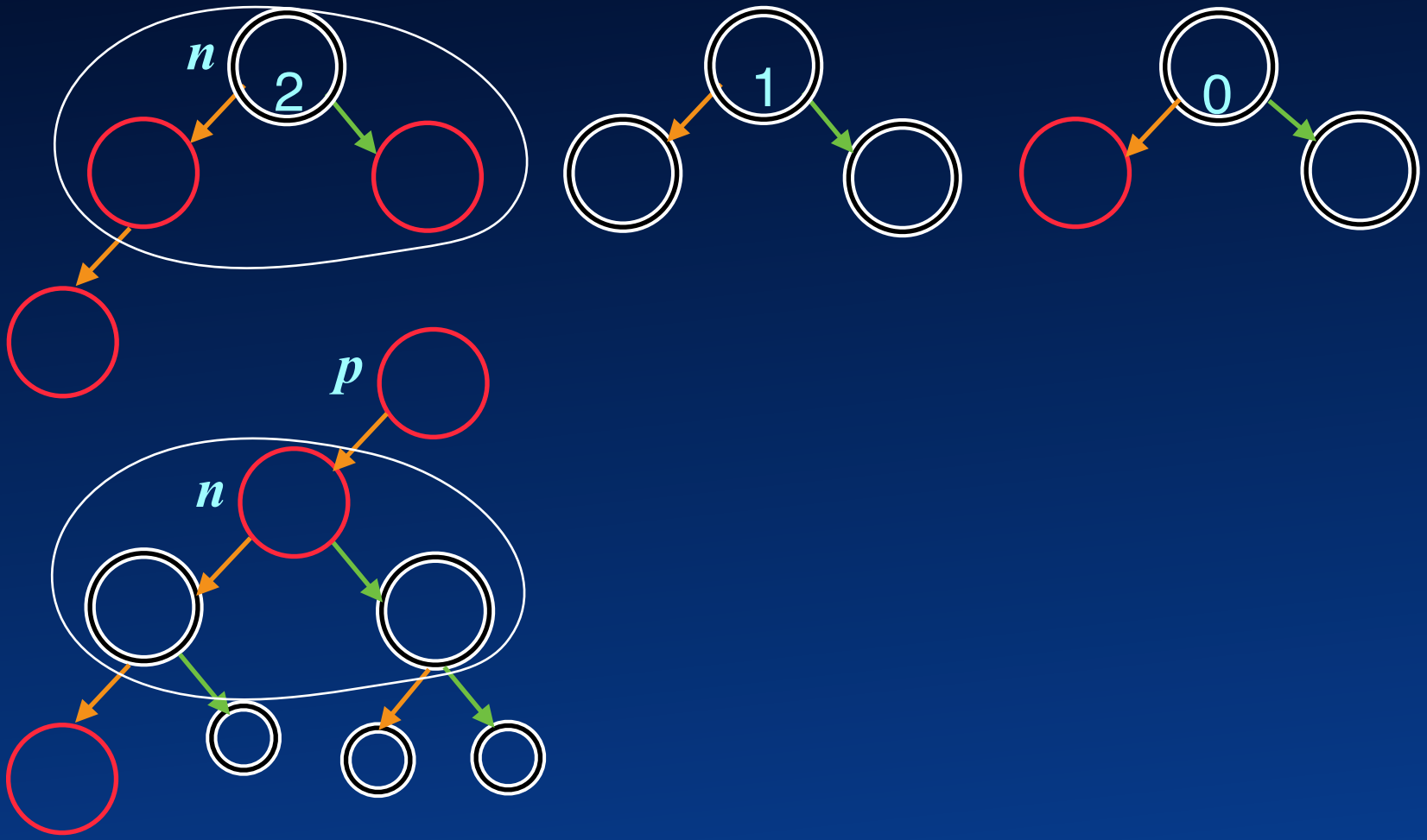
Insert





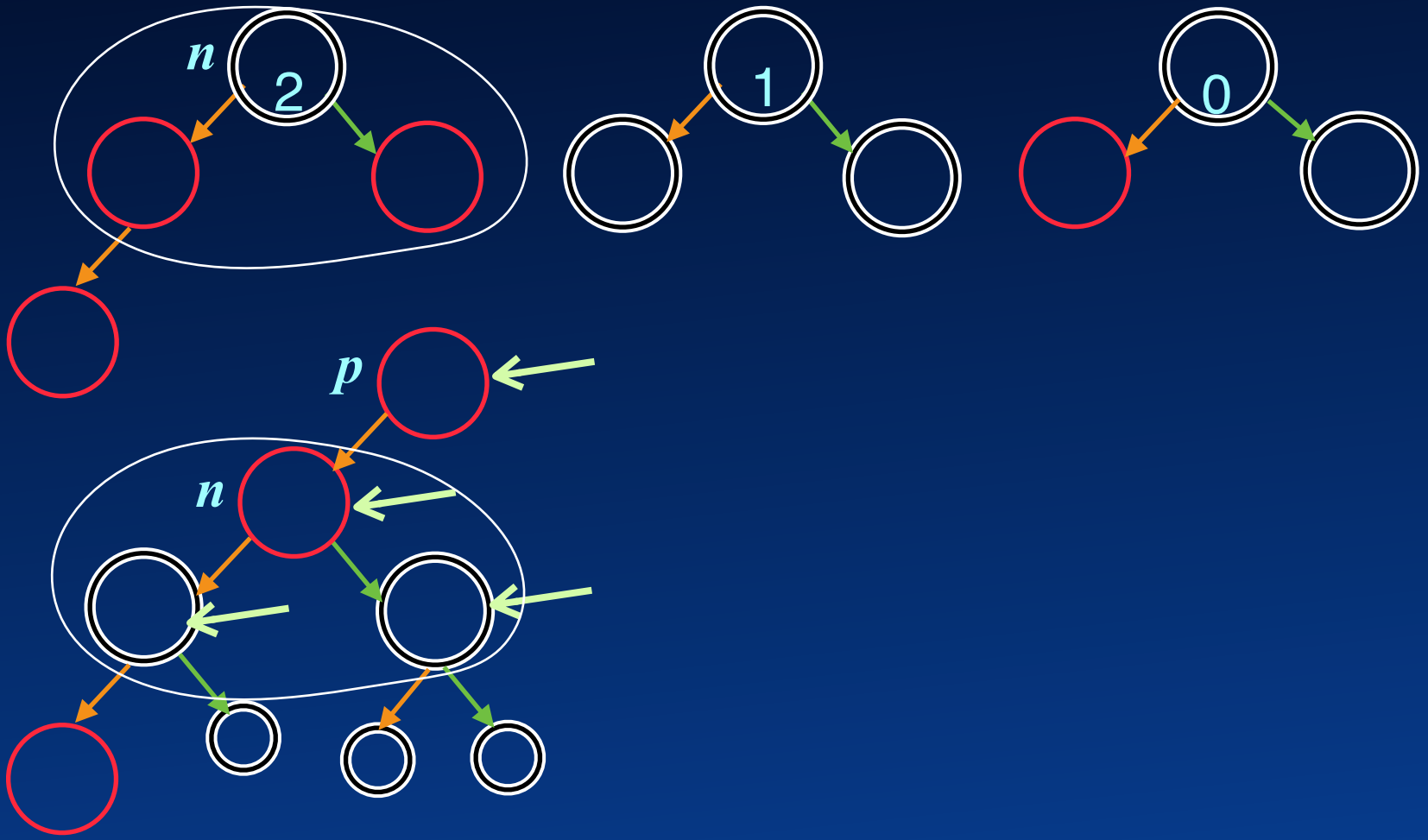
R-B Tree Color changes

Insert



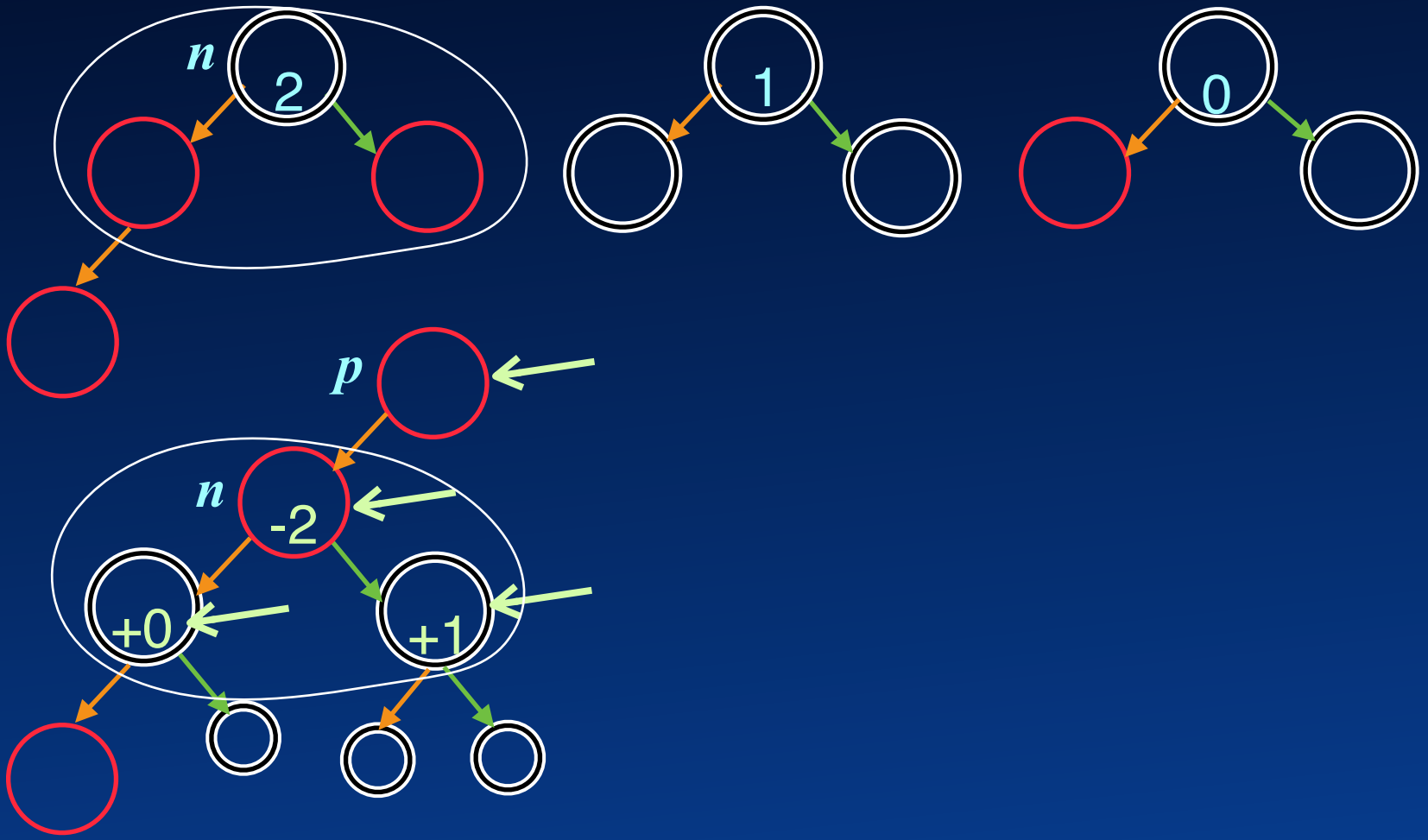
R-B Tree Color changes

Insert



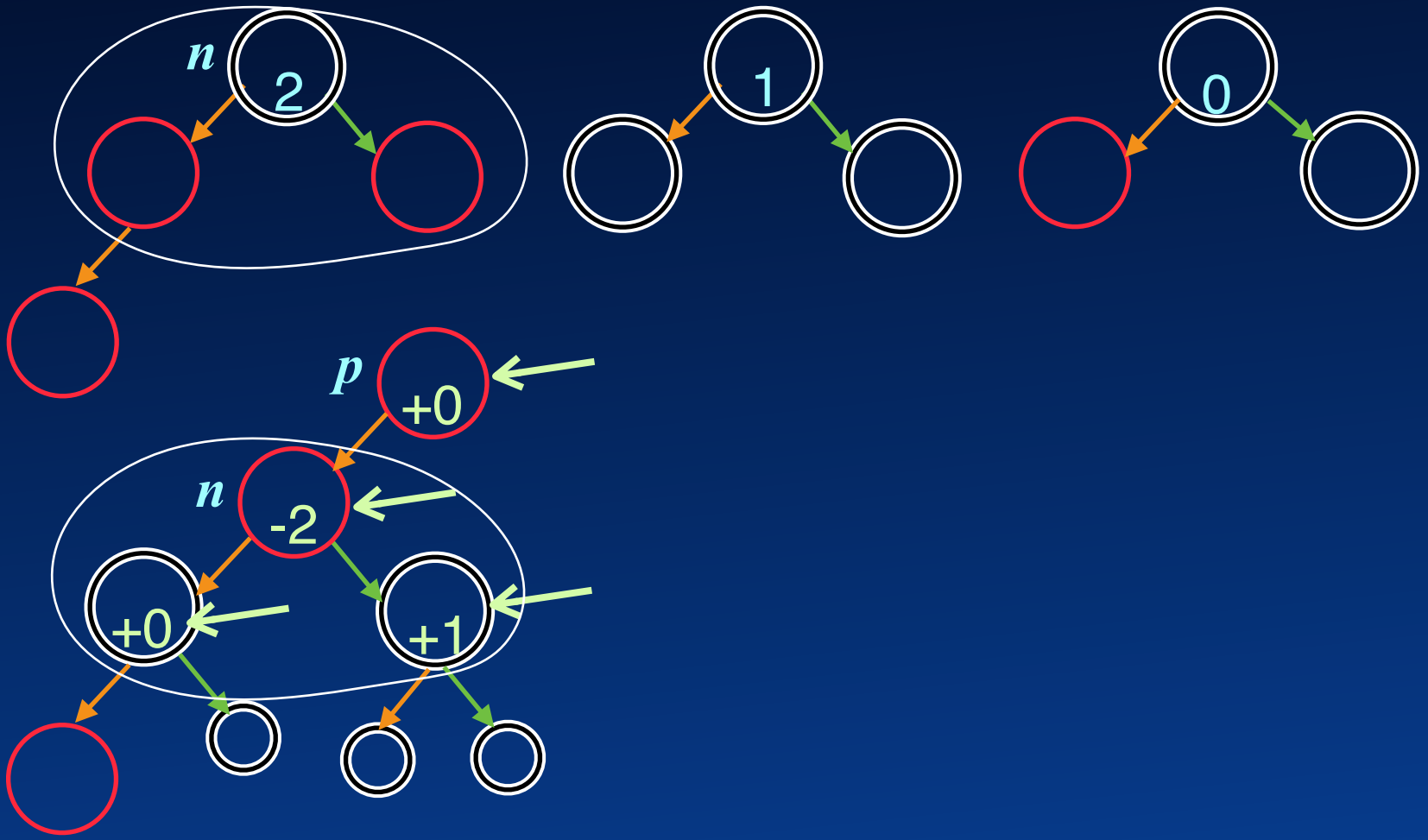
R-B Tree Color changes

Insert



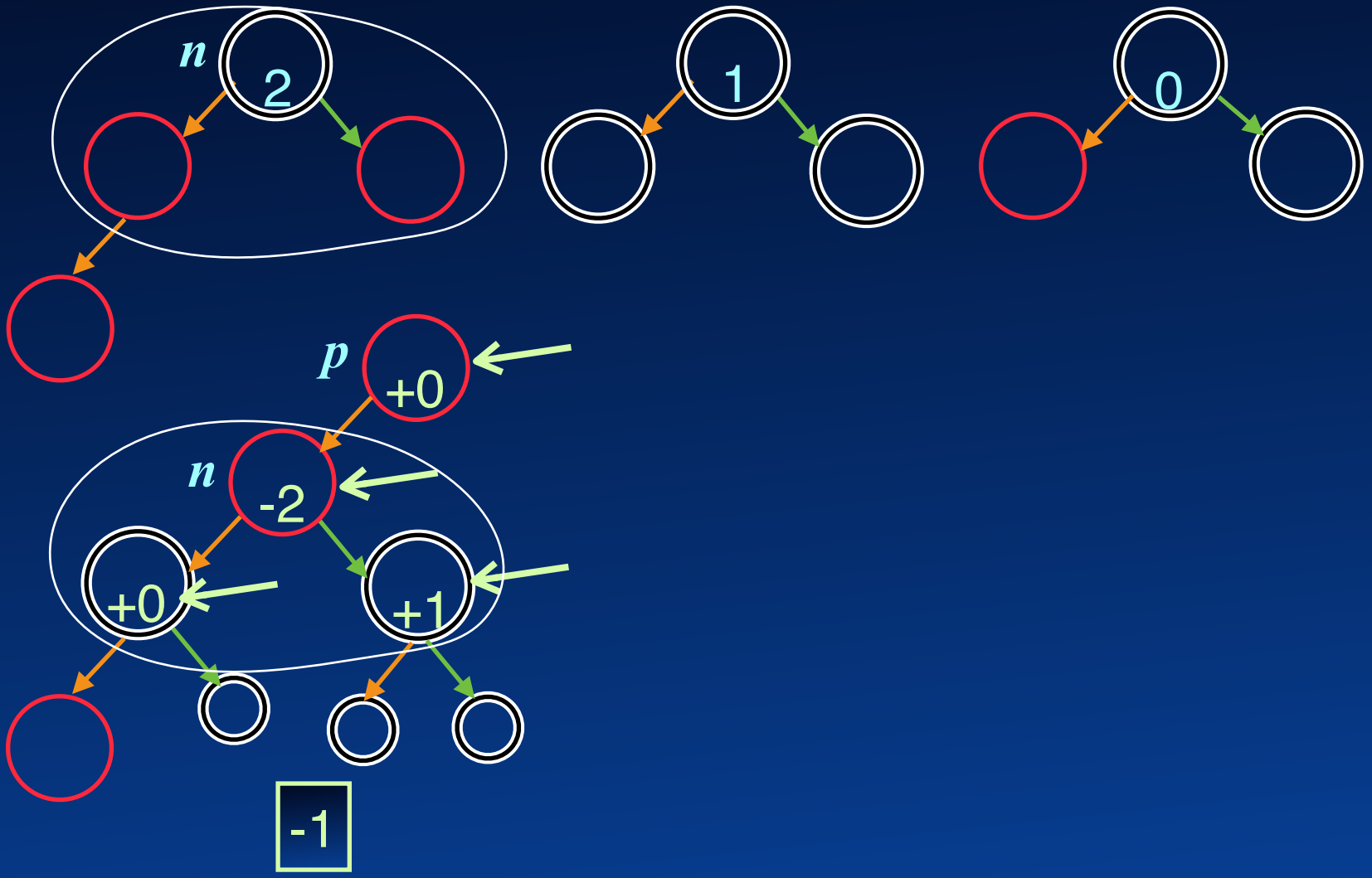
R-B Tree Color changes

Insert



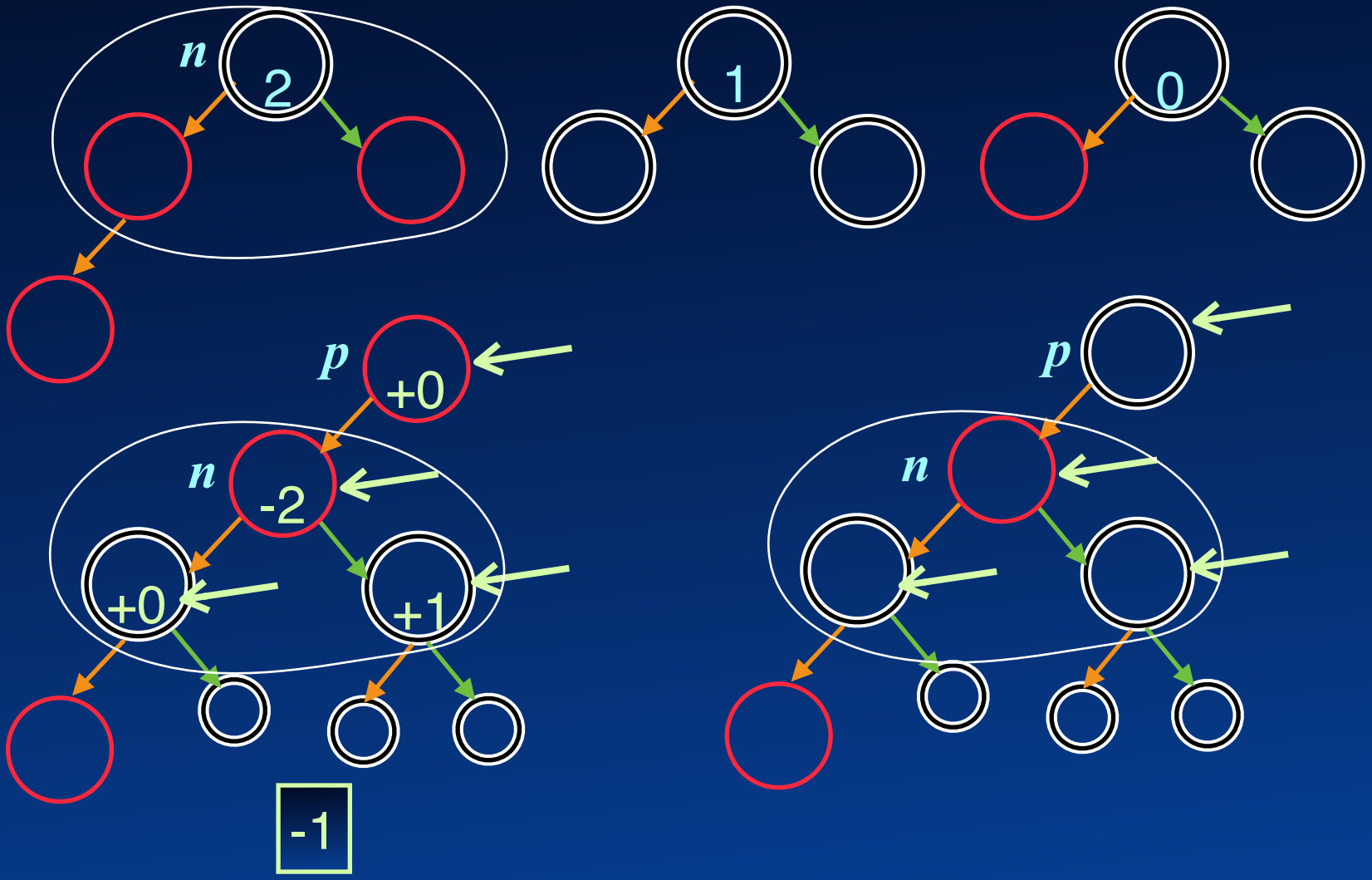
R-B Tree Color changes

Insert



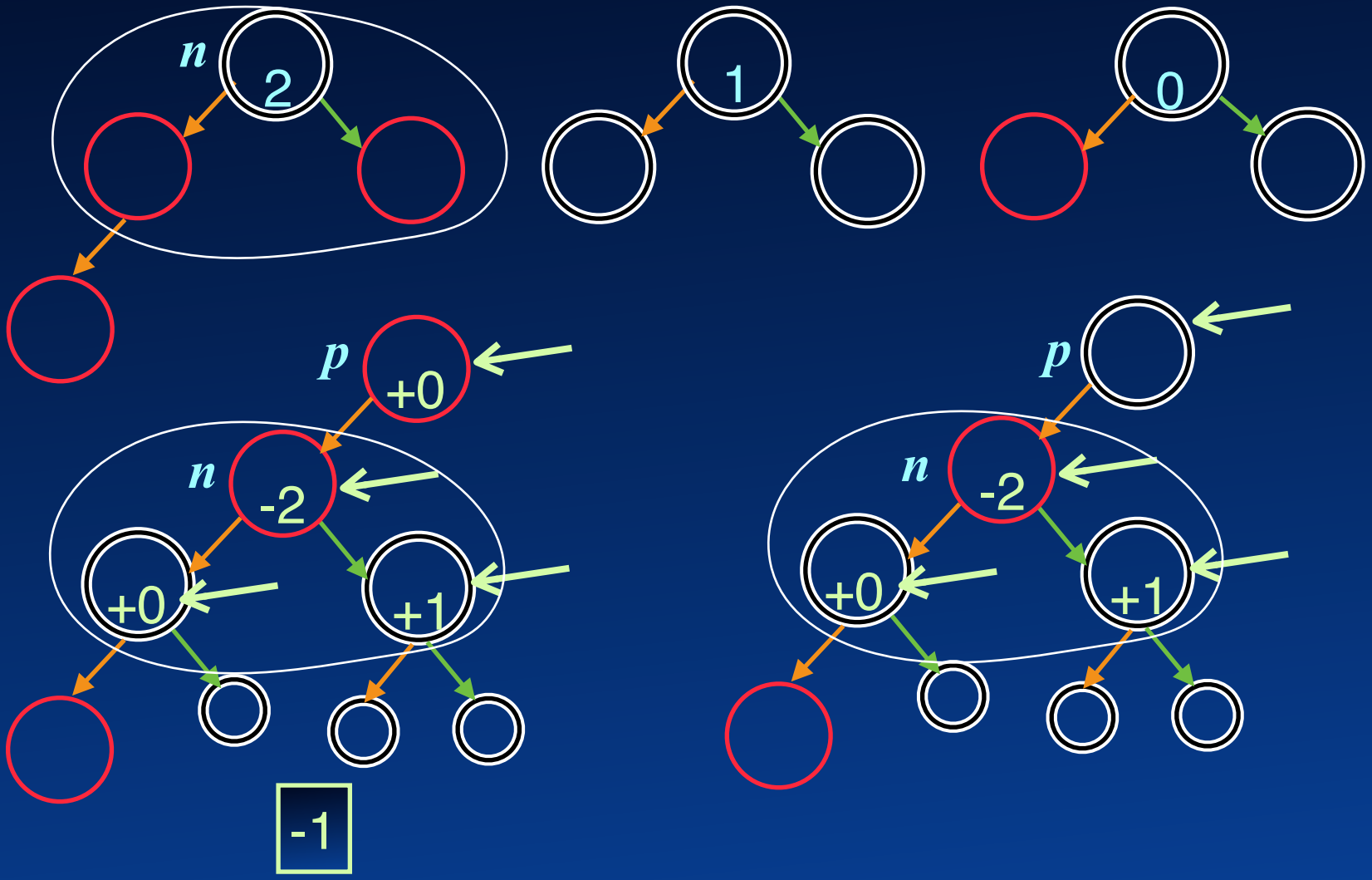
R-B Tree Color changes

Insert



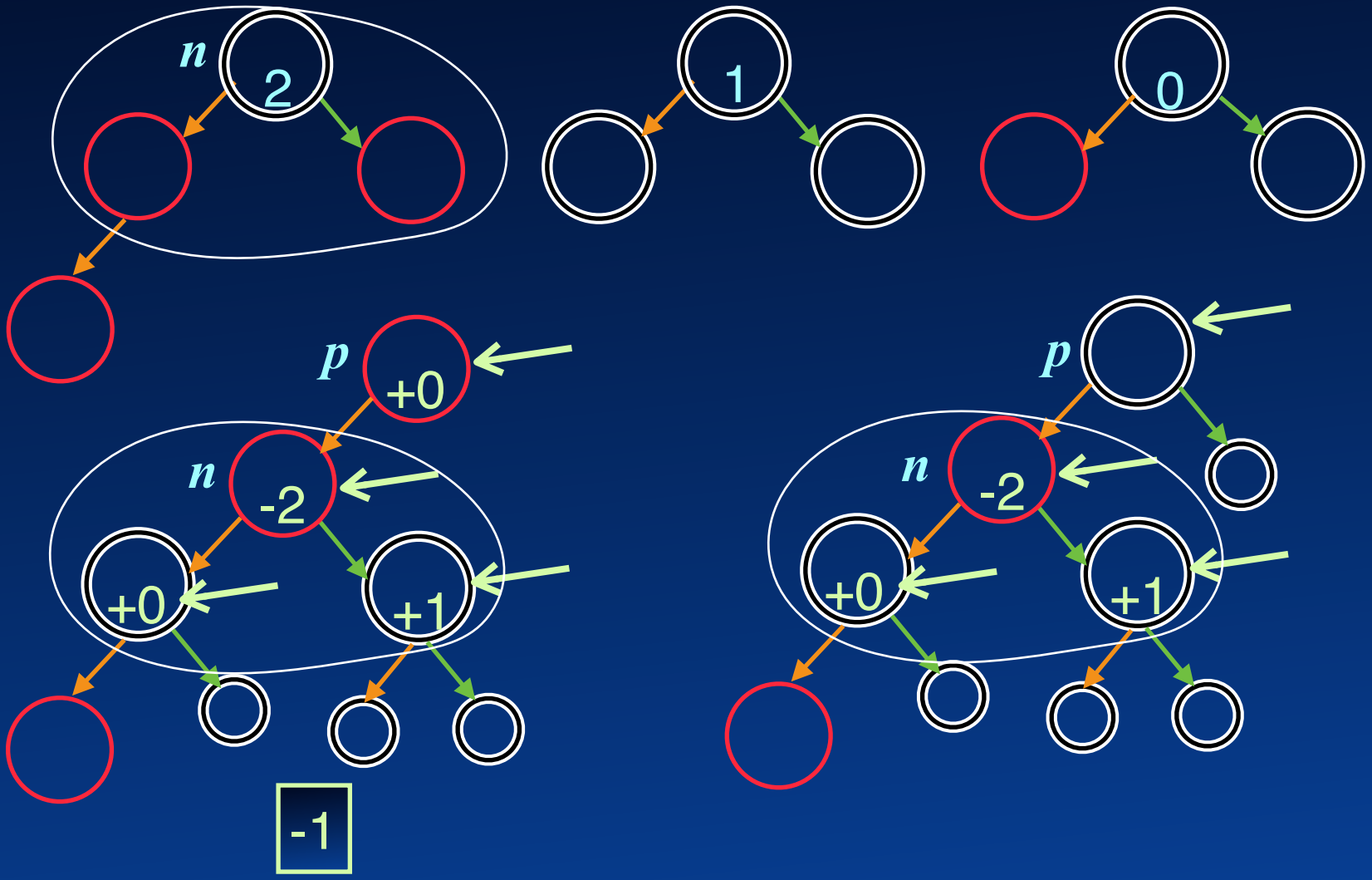
R-B Tree Color changes

Insert



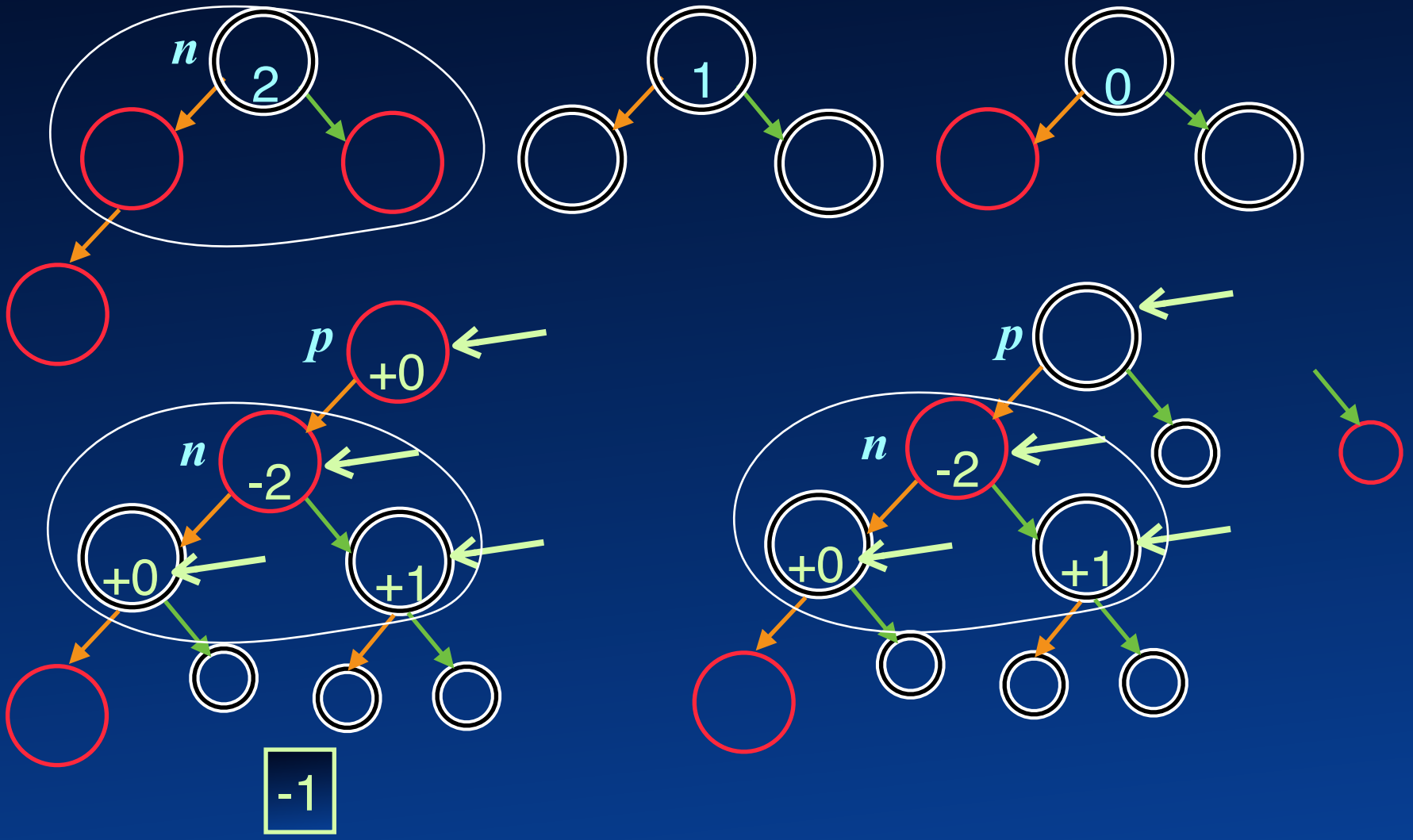
R-B Tree Color changes

Insert



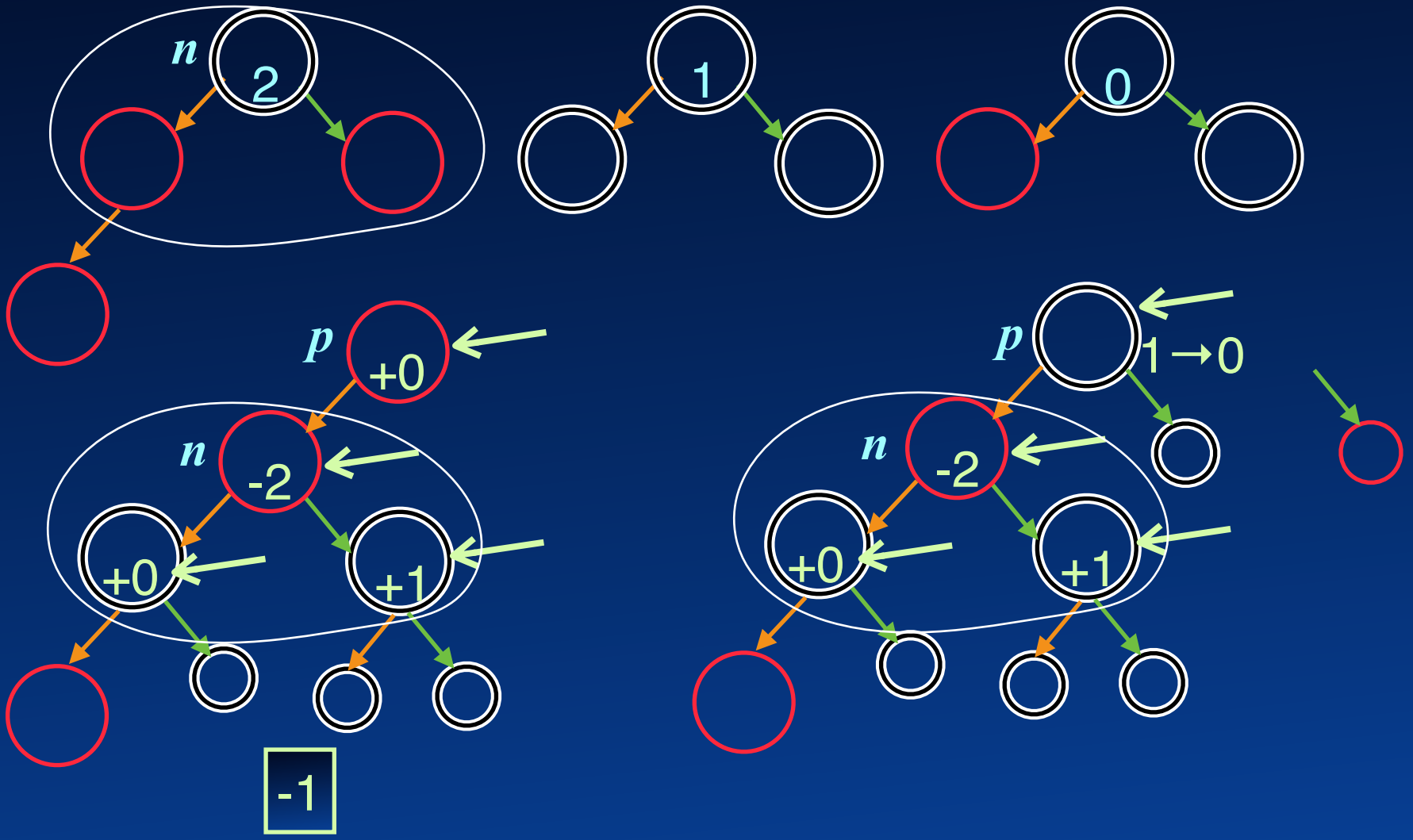
R-B Tree Color changes

Insert



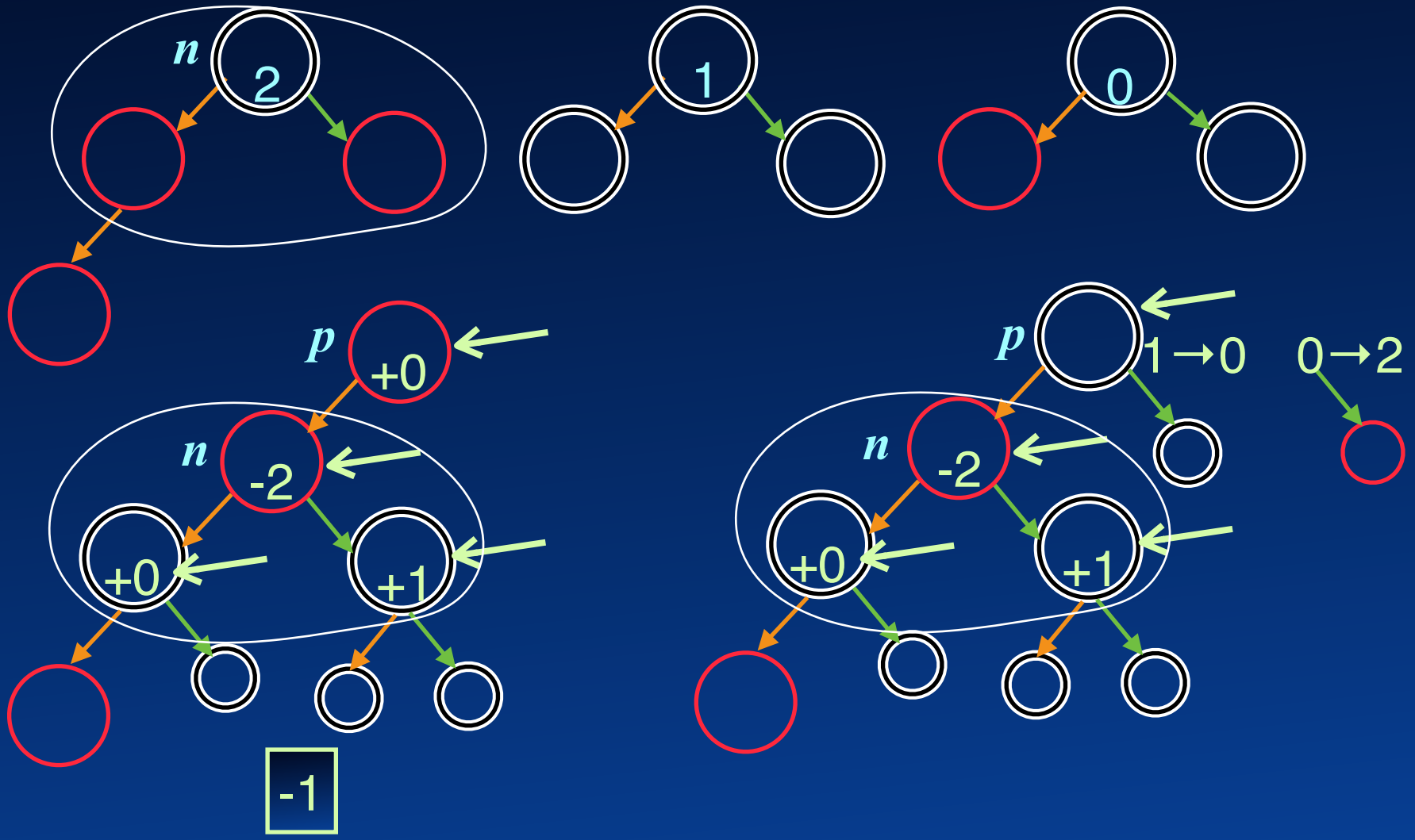
R-B Tree Color changes

Insert



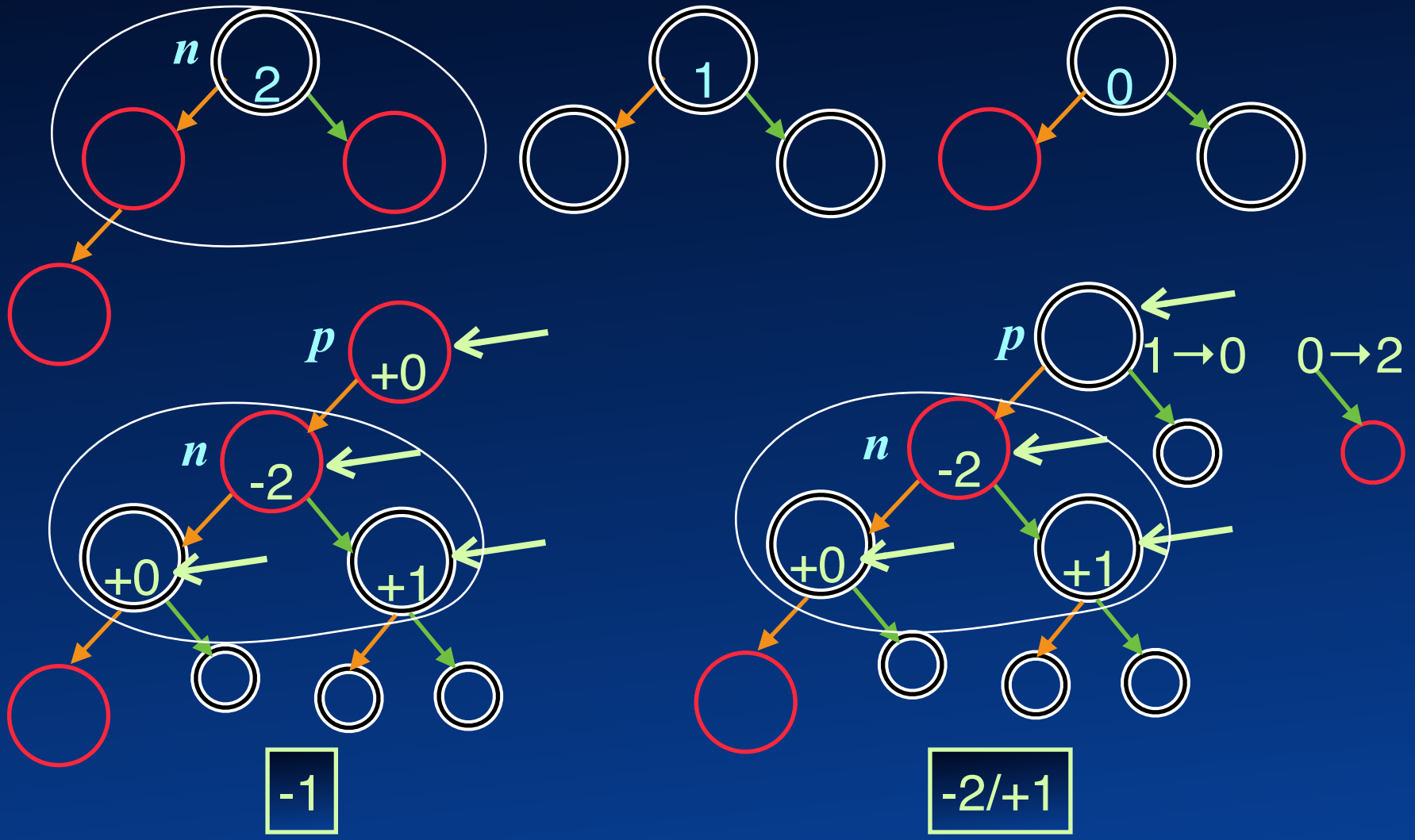
R-B Tree Color changes

Insert



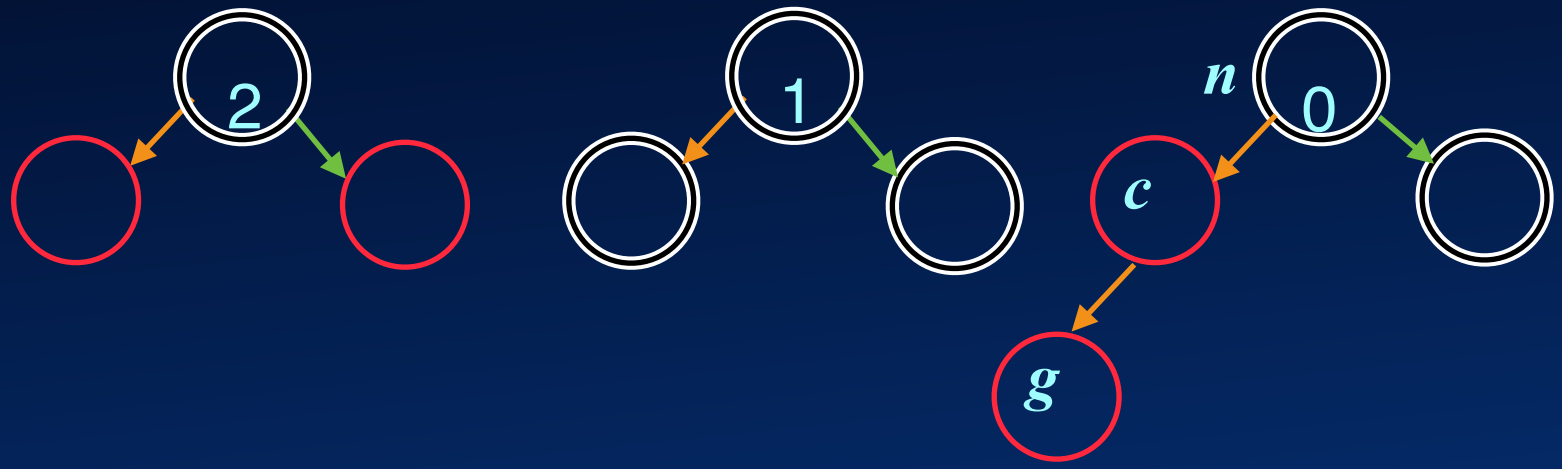
R-B Tree Color changes

Insert



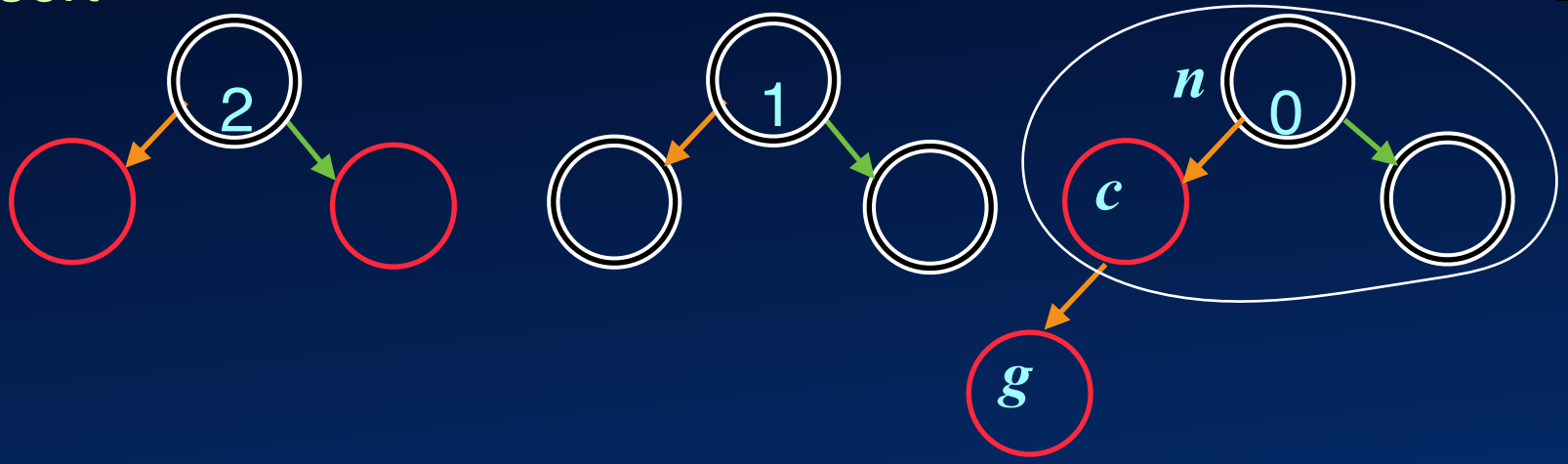
R-B Tree Color changes

Insert



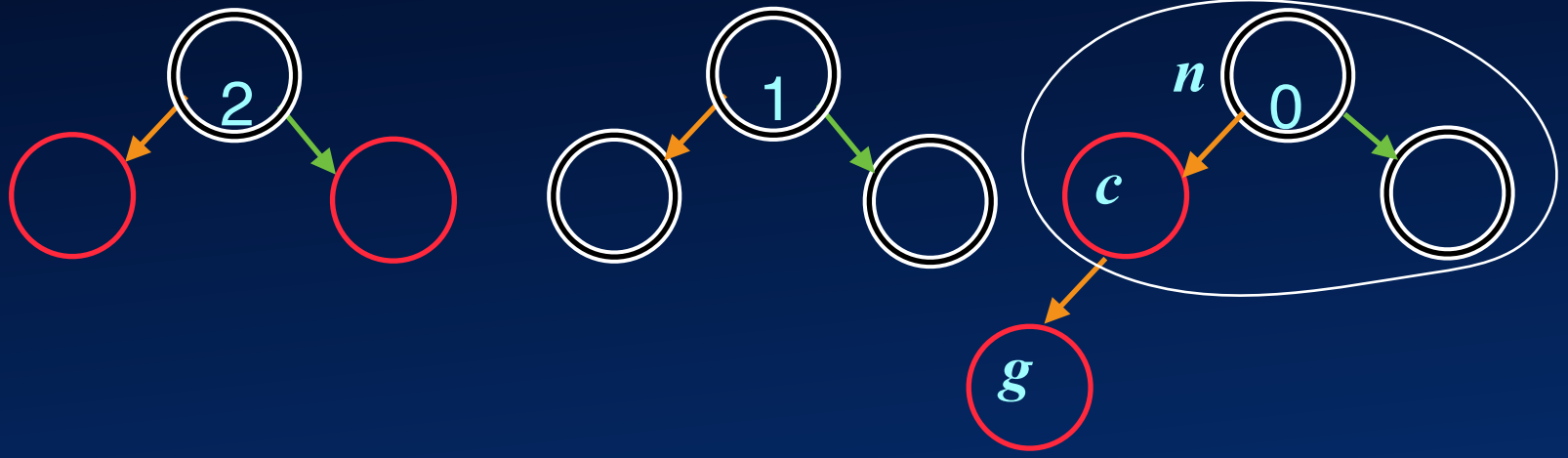
R-B Tree Color changes

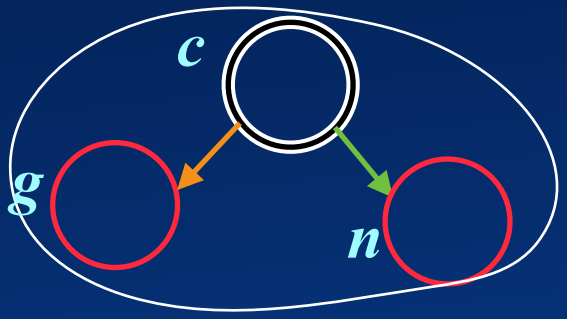
Insert



R-B Tree Color changes

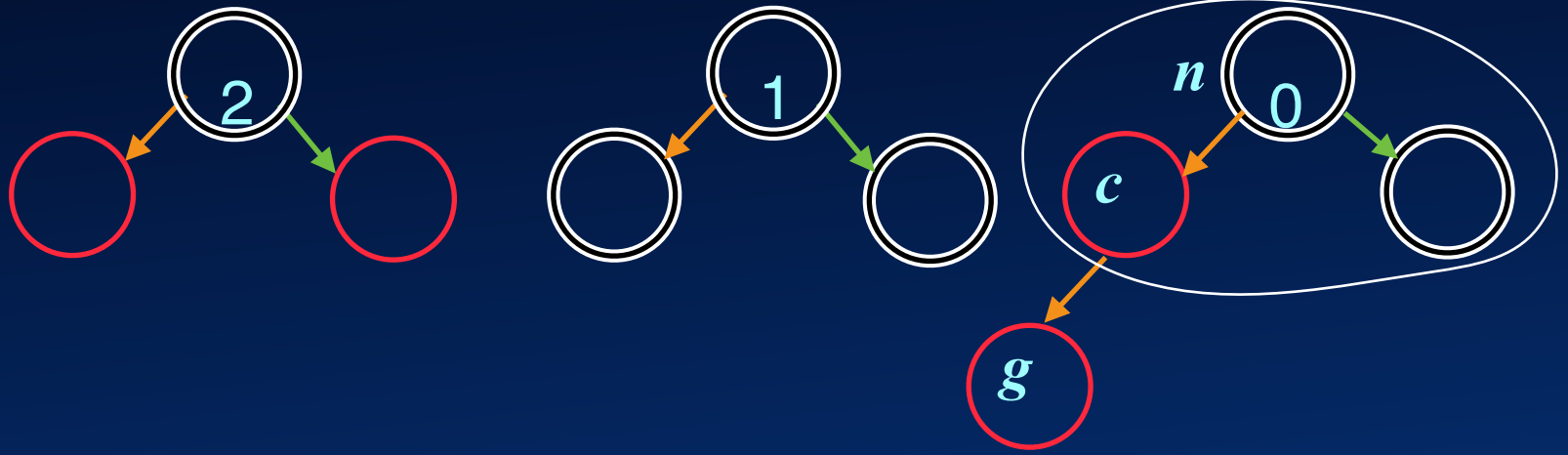
Insert

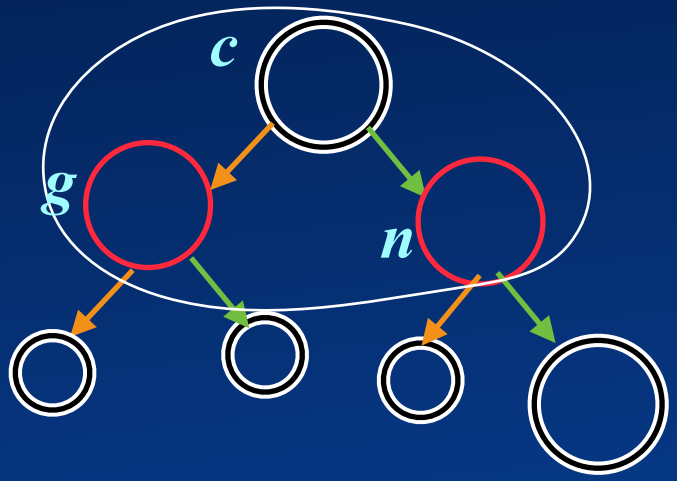




R-B Tree Color changes

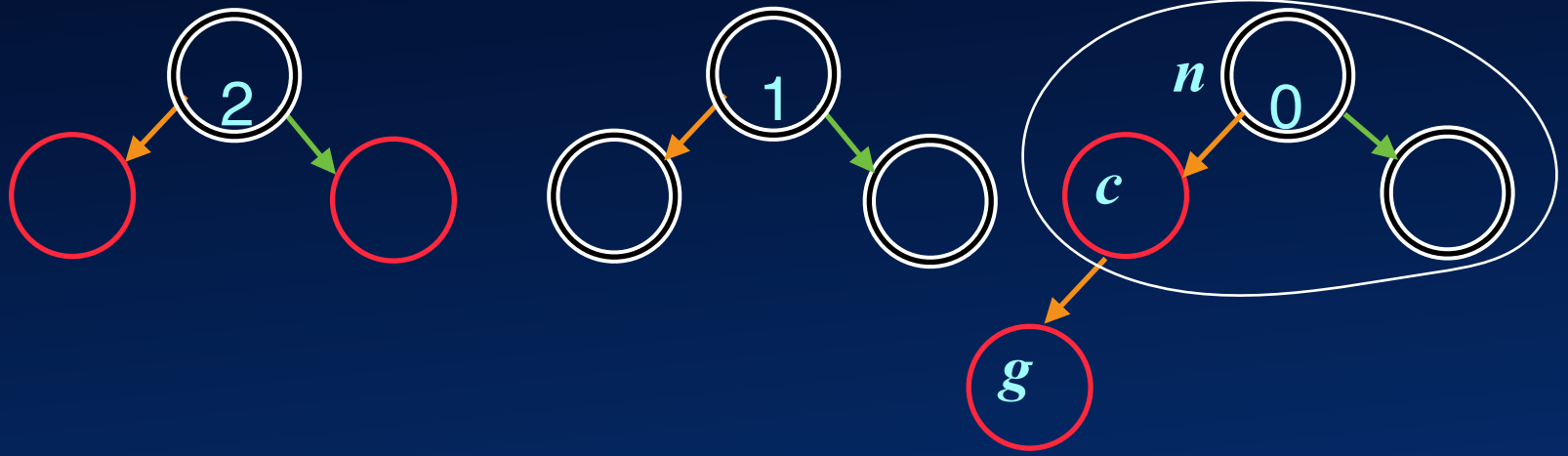
Insert

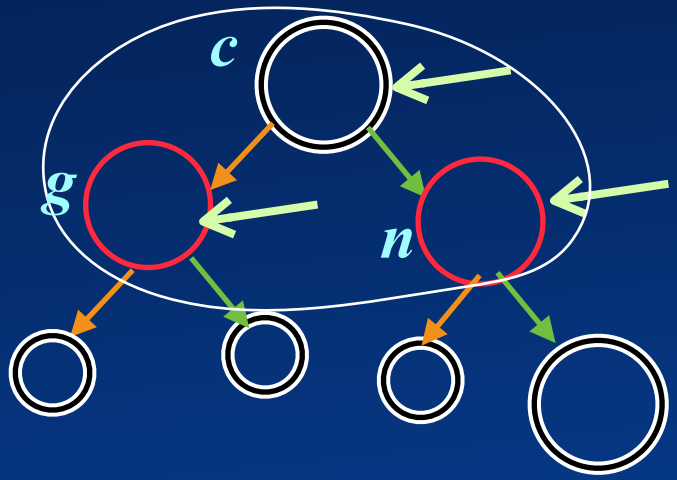




R-B Tree Color changes

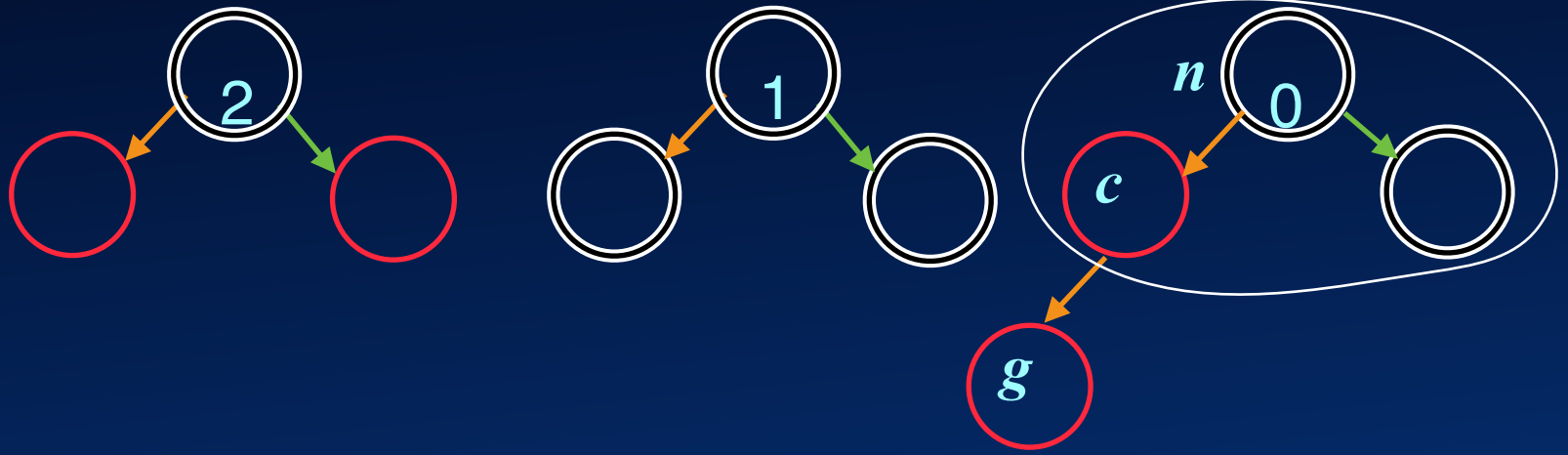
Insert

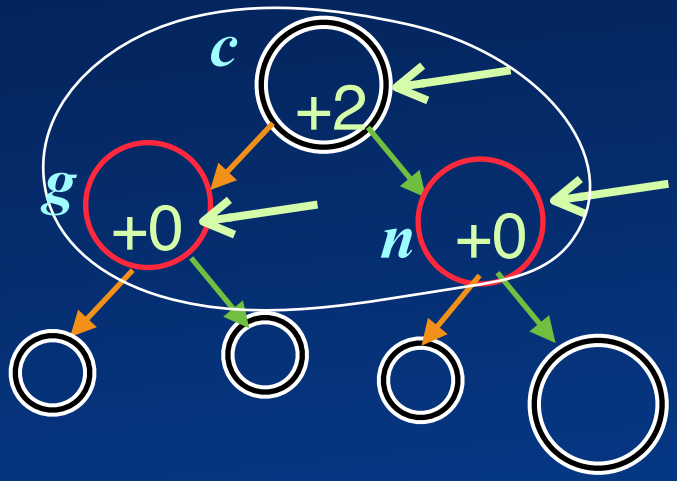




R-B Tree Color changes

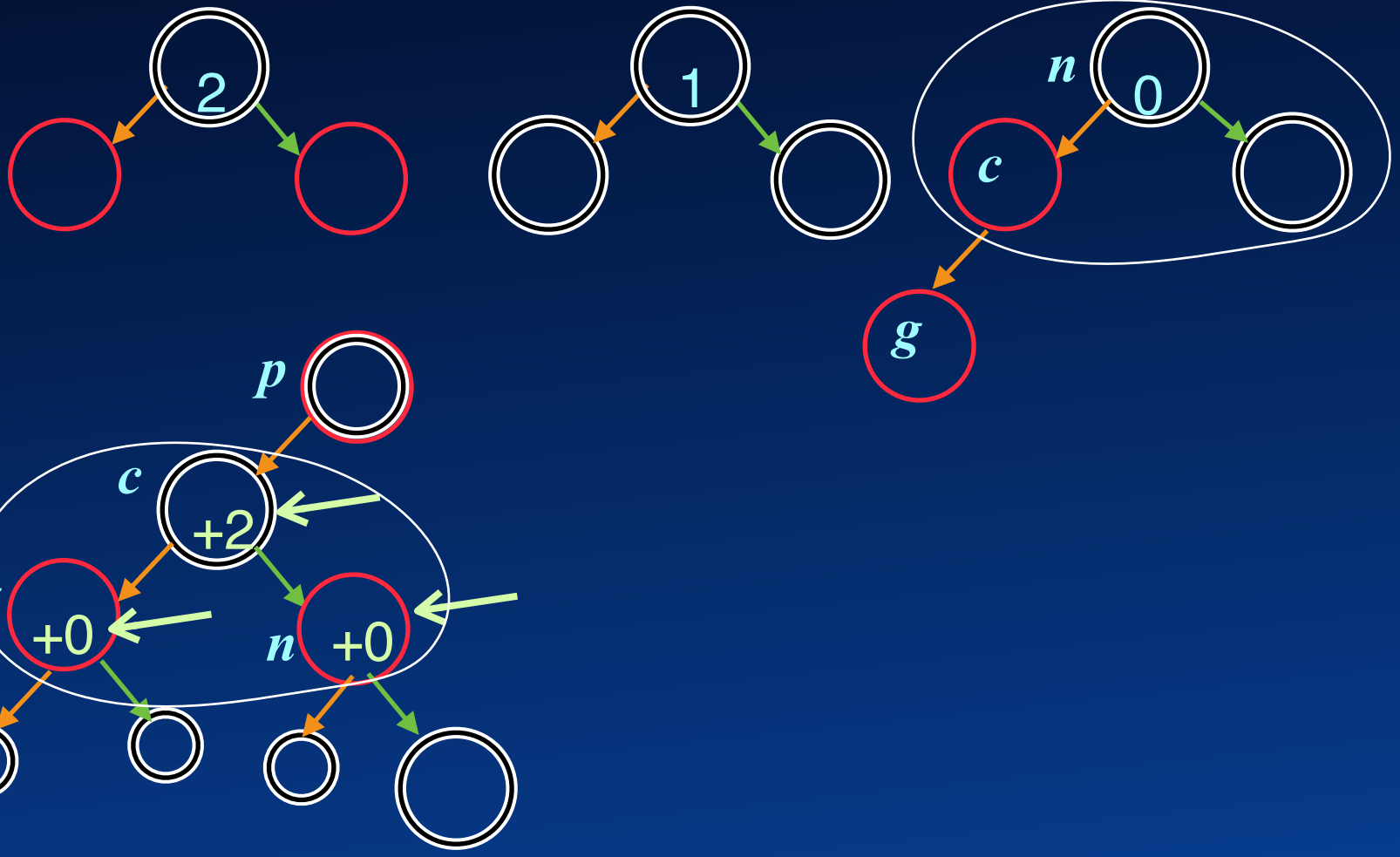
Insert





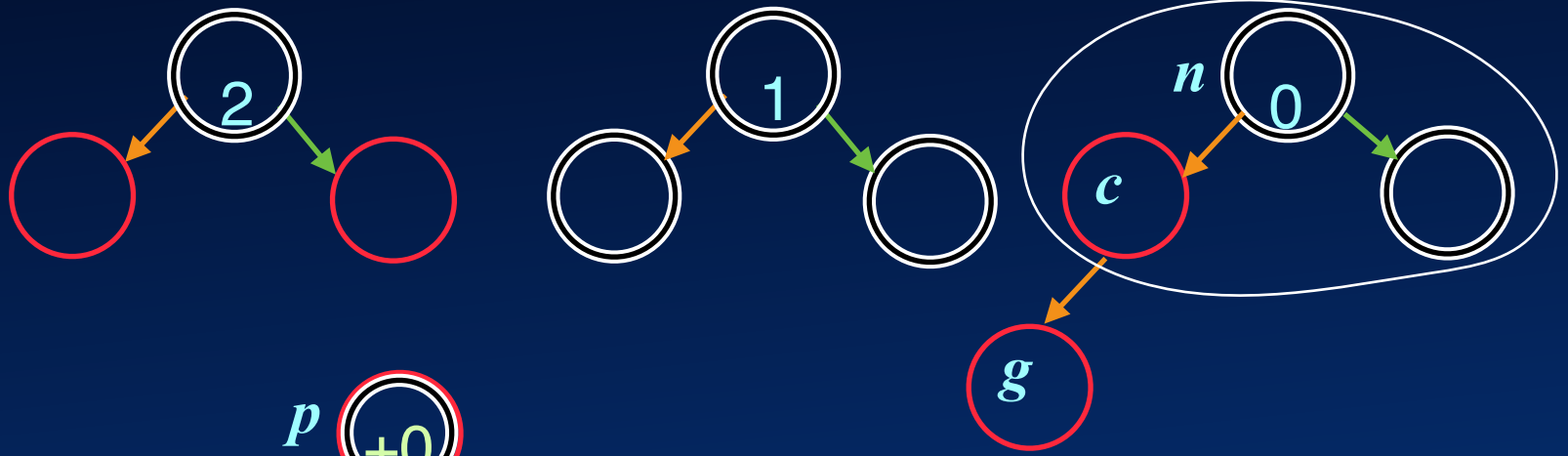
R-B Tree Color changes

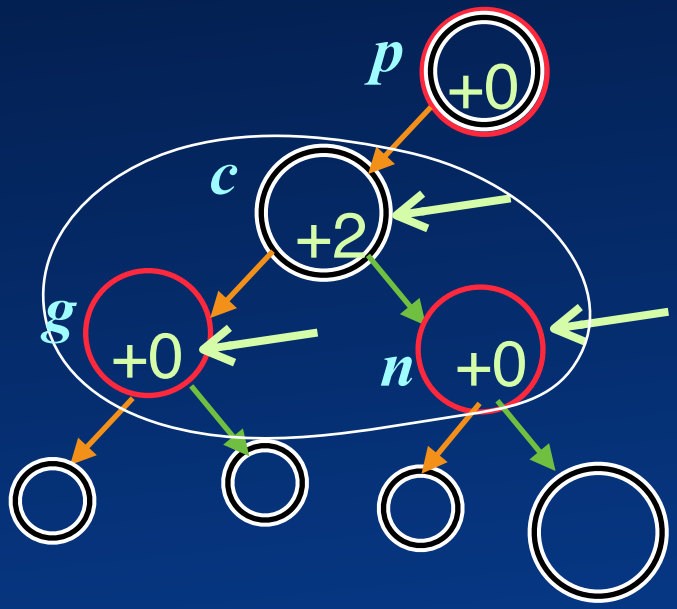
Insert



R-B Tree Color changes

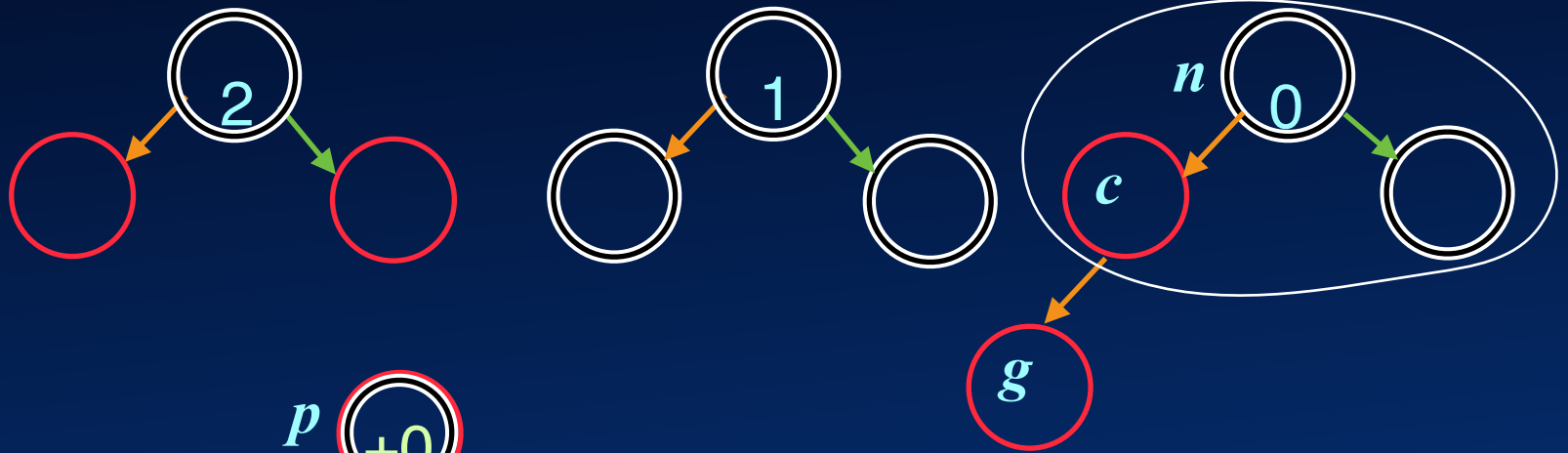
Insert

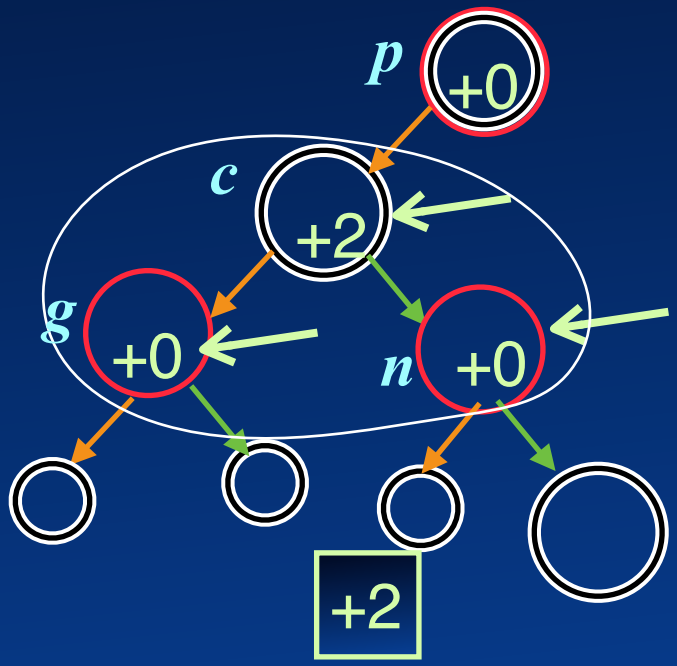




R-B Tree Color changes

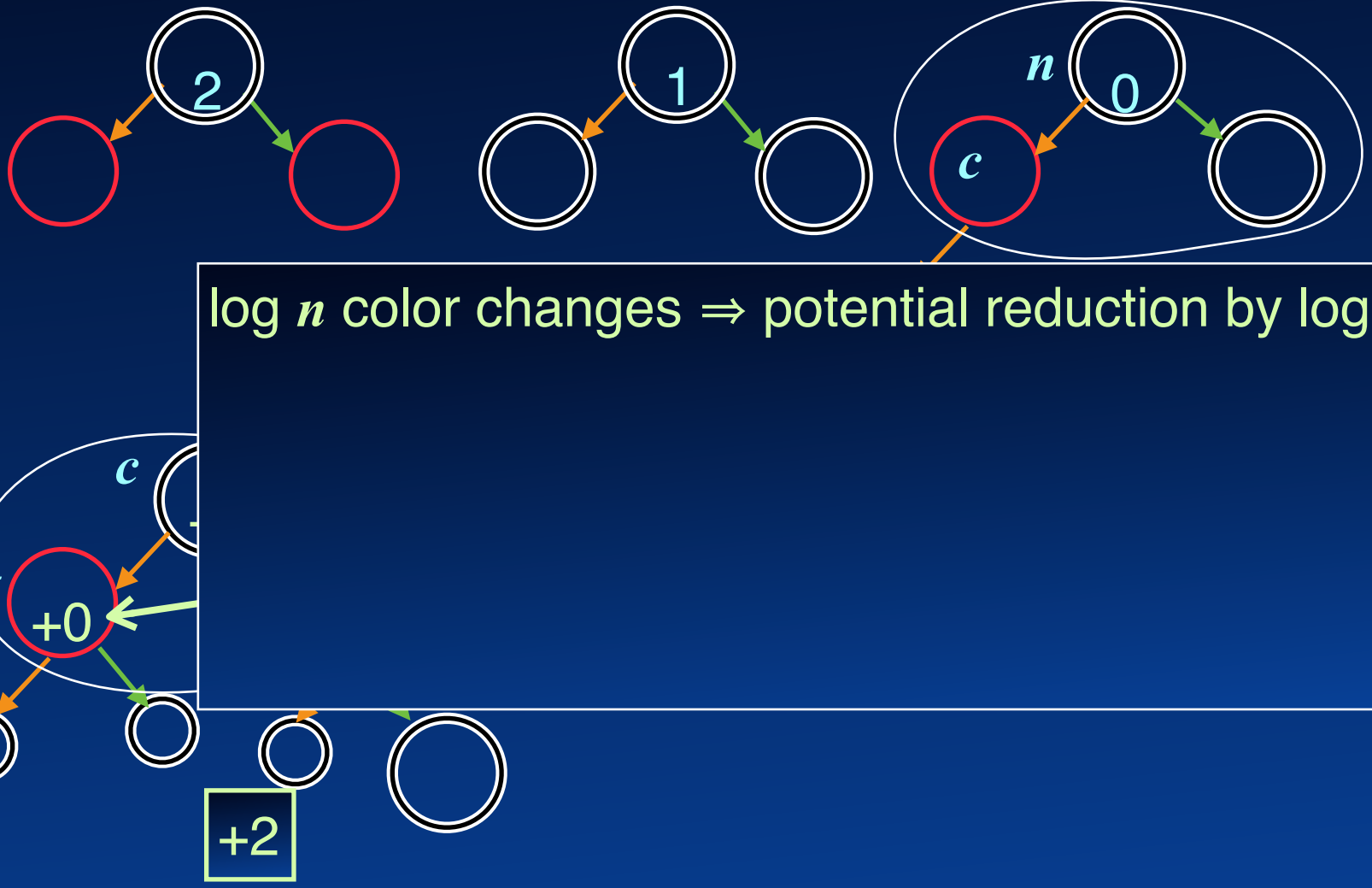
Insert





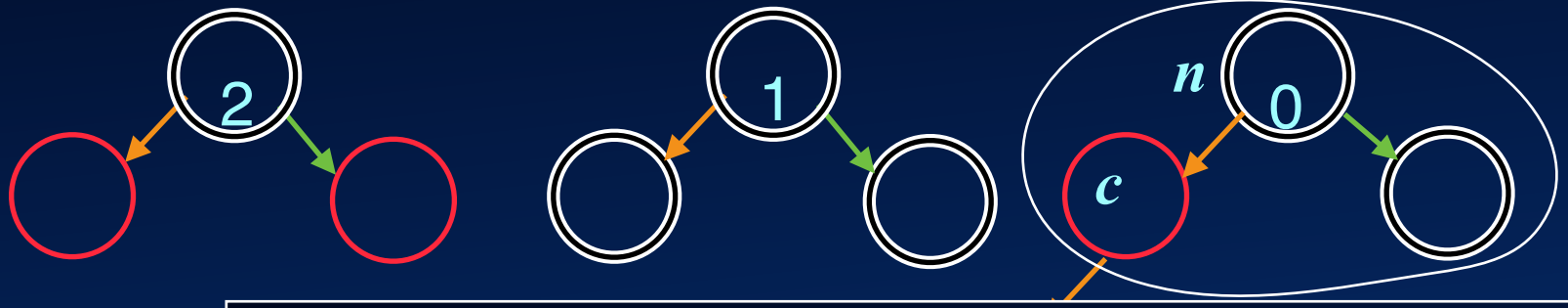
R-B Tree Color changes

Insert



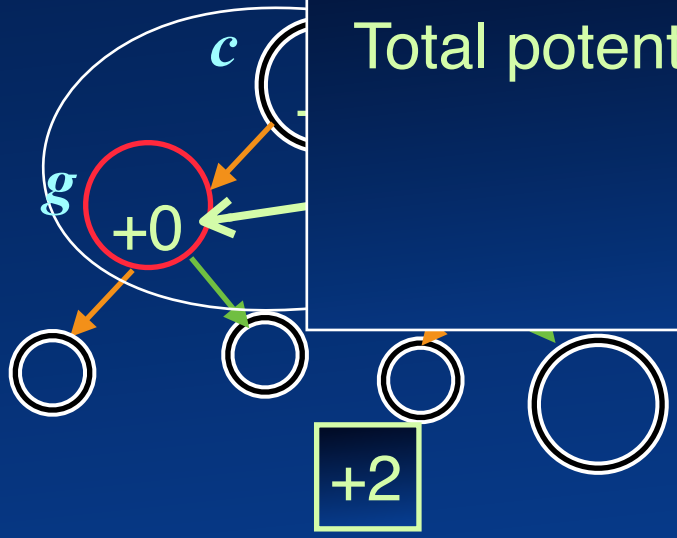
R-B Tree Color changes

Insert



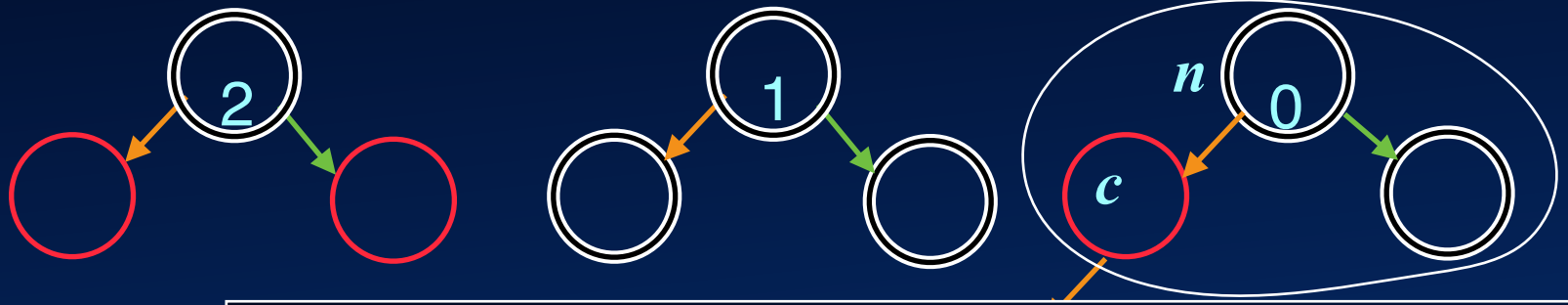
log n color changes ⇒ potential reduction by log n

After n insertions (starting from a null tree):
Total potential ≤ n



R-B Tree Color changes

Insert

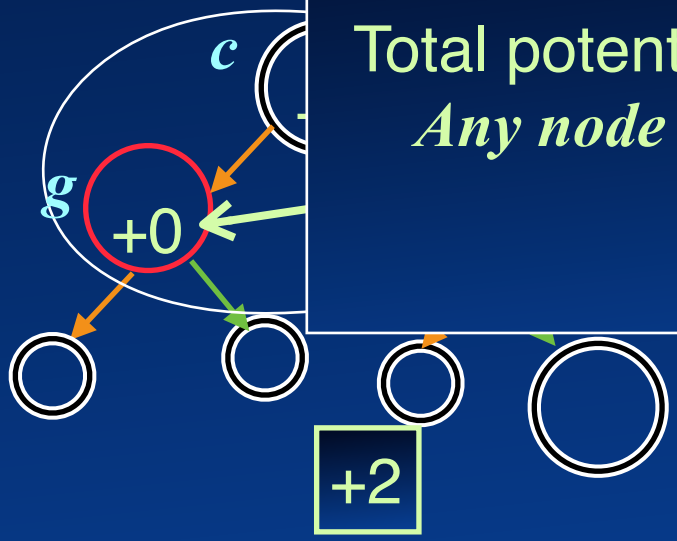


log n color changes ⇒ potential reduction by log n

After n insertions (starting from a null tree):

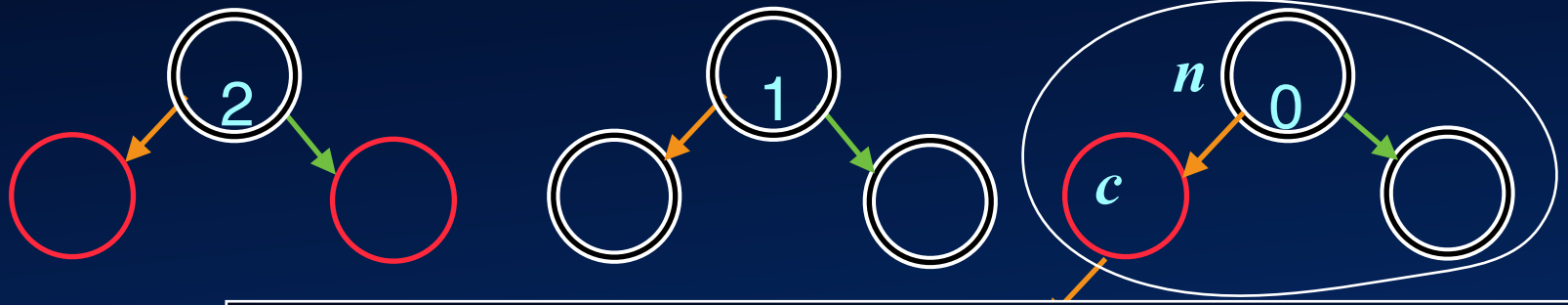
Total potential ≤ n

Any node with $\Phi = 2$ implies 2 red nodes



R-B Tree Color changes

Insert



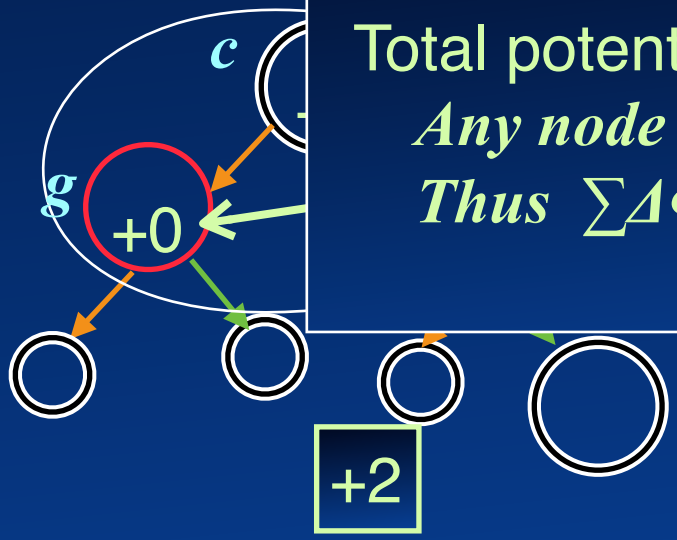
log n color changes ⇒ potential reduction by log n

After n insertions (starting from a null tree):

Total potential ≤ n

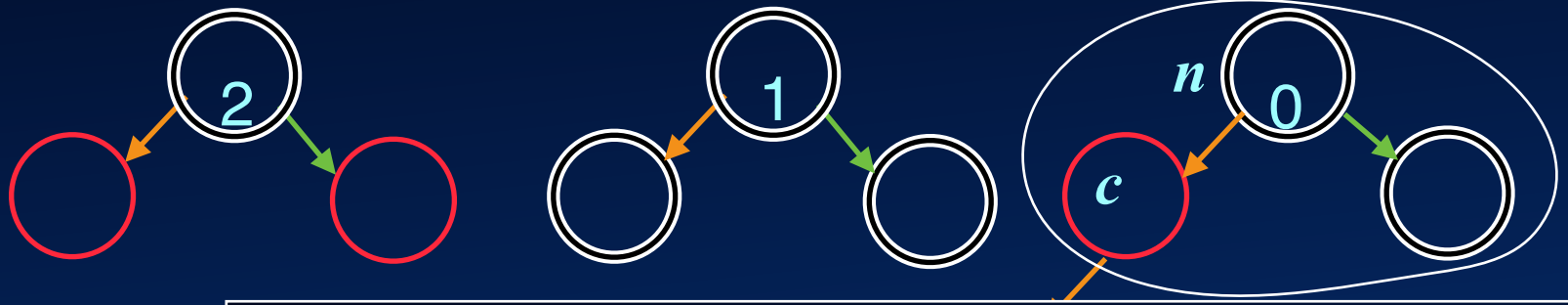
Any node with $\Phi = 2$ implies 2 red nodes

Thus $\sum \Delta\Phi \le n \Rightarrow$ +2 .. -Recolors - Recolors.. $\le n$



R-B Tree Color changes

Insert



log n color changes ⇒ potential reduction by log n

After n insertions (starting from a null tree):

Total potential ≤ n

Any node with $\Phi = 2$ implies 2 red nodes

Thus $\sum \Delta\Phi \le n \Rightarrow$ +2 .. -Recolors - Recolors.. $\le n$

$\sum$ other cases $\le 2n \Rightarrow \sum$ bad cases = O(n)