Course: Mathematics for IT Professionals

Lecture 9

Trees

By

Solomon Mensah

Outline

The topics to be treated in this lecture are:

- Definition of trees
- Binary trees
- Properties of Binary trees
- Abstract Data Type for Trees
- Binary Tree Traversal
  - Pre-order traversal
  - In-order traversal
  - Post-order traversal
- Applications of Trees

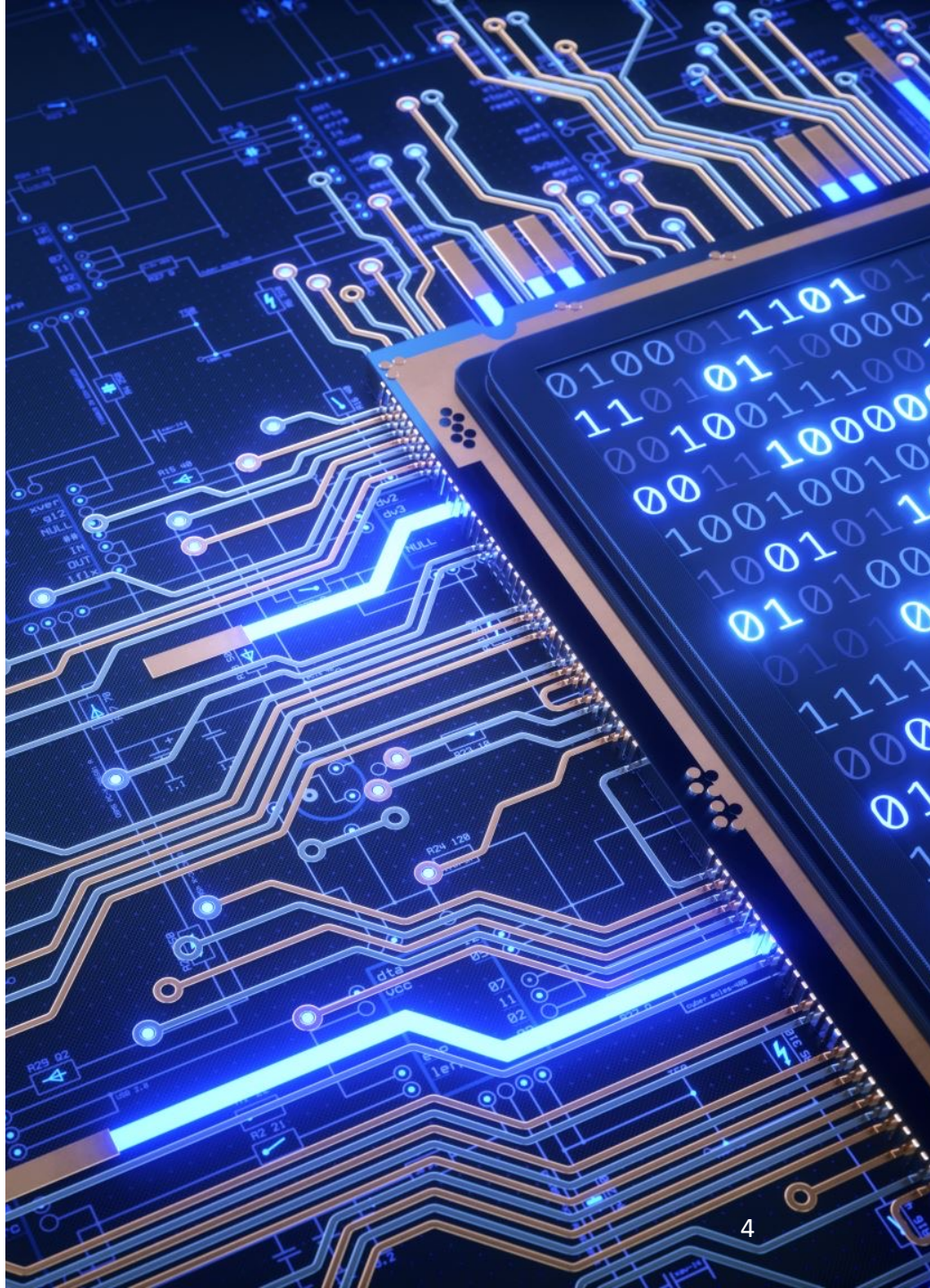
Lecture Learning Outcomes

At the end of the session, you will be able to

- differentiate between a tree and non-tree structure
- understand basic properties of tree data structure
- know the forms of binary trees
- execute tree traversals within a binary tree structure
- know some applications of trees

Preliminaries

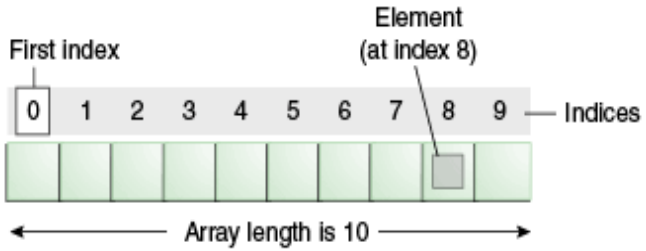
- A data structure is a way to store and organize data in a computer memory
- Data structures can be broadly classified as;
  - Linear data structures
  - Non-linear data structures

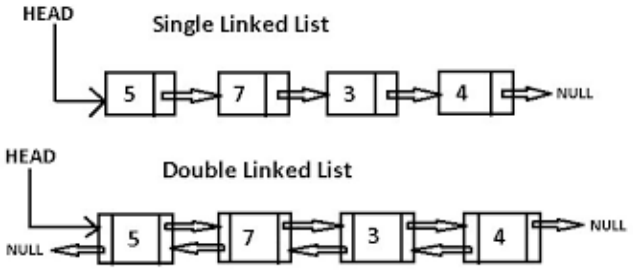


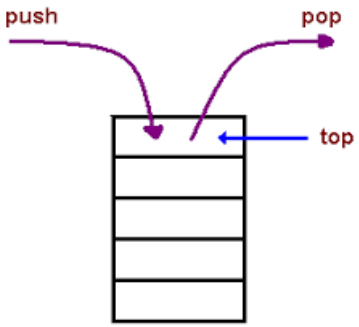
Rosen, K. H. (2012). *Discrete mathematics and its applications (7th Edition)*. McGraw-Hill.

Preliminaries

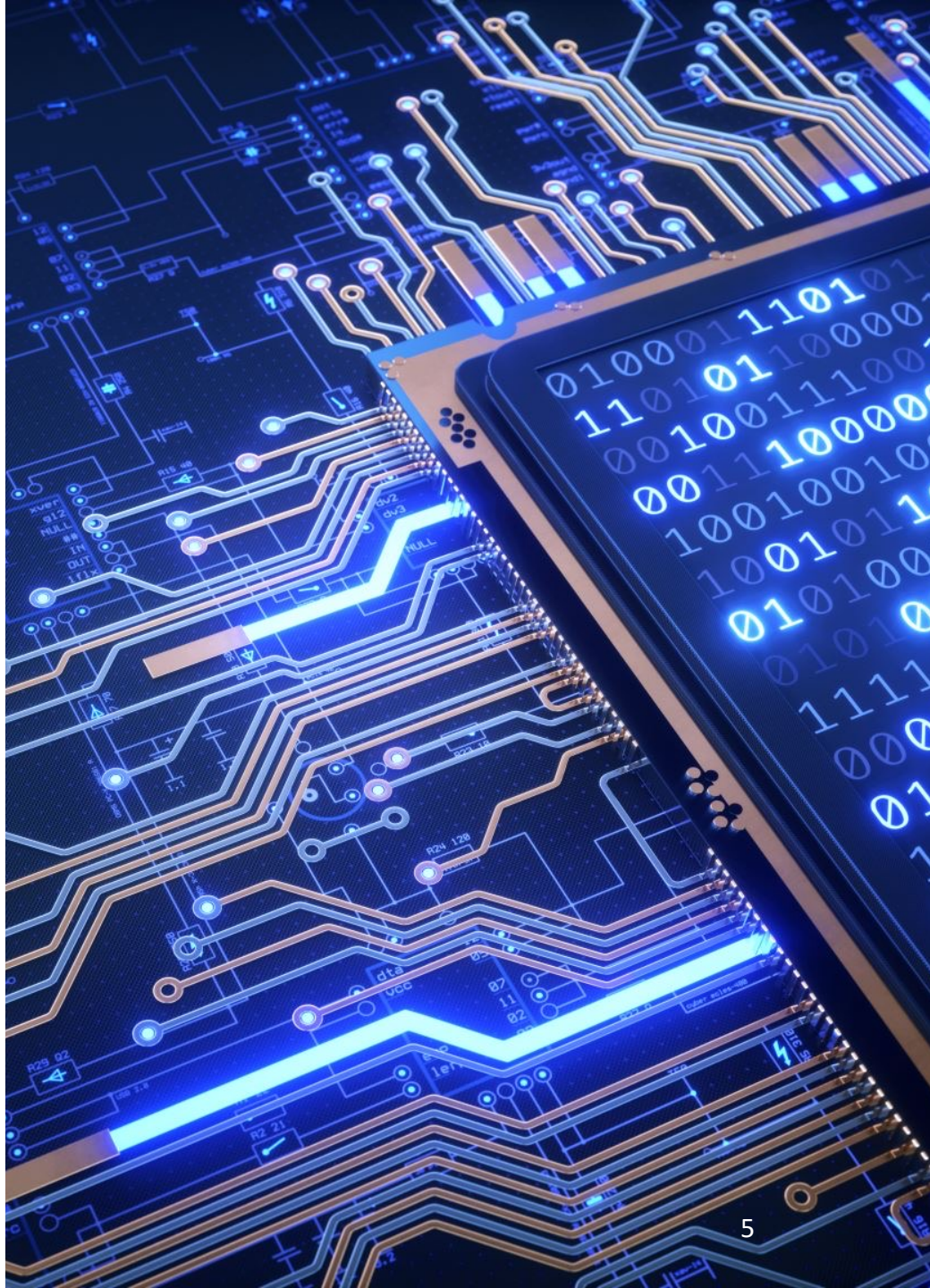
- **Linear data structures** are those in which the data elements are accessed in a sequential manner





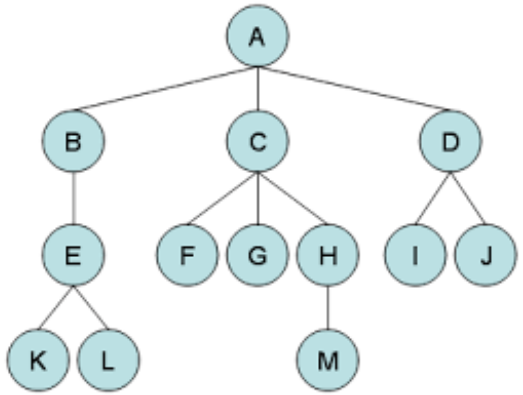


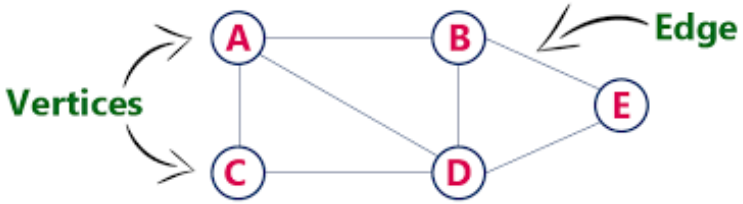
Rosen, K. H. (2012). *Discrete mathematics and its applications (7th Edition)*. McGraw-Hill.

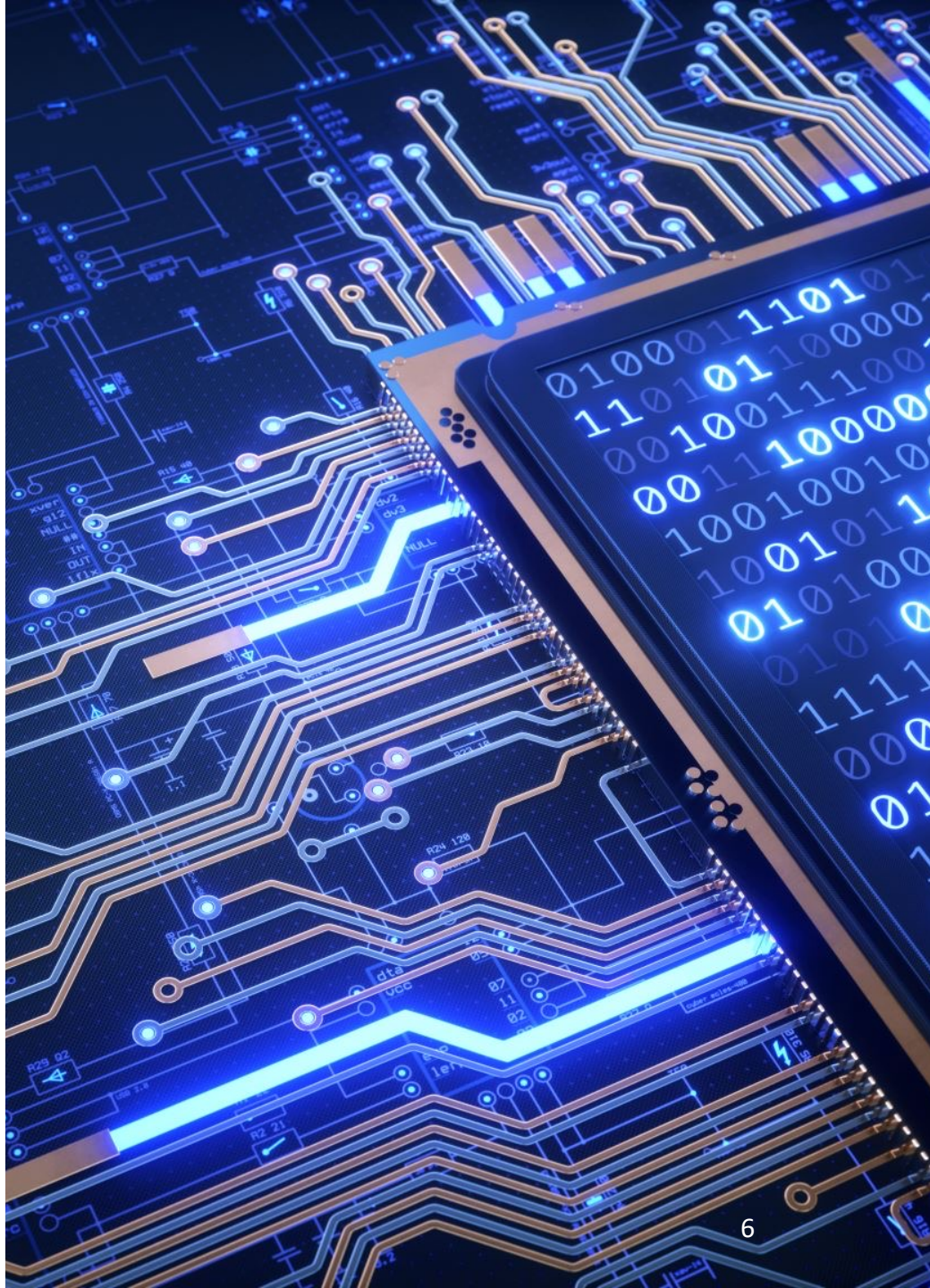


Preliminaries

- **Non-linear data structures** are those in which the data elements are connected and accessed non-sequentially based on certain property or relationship among them



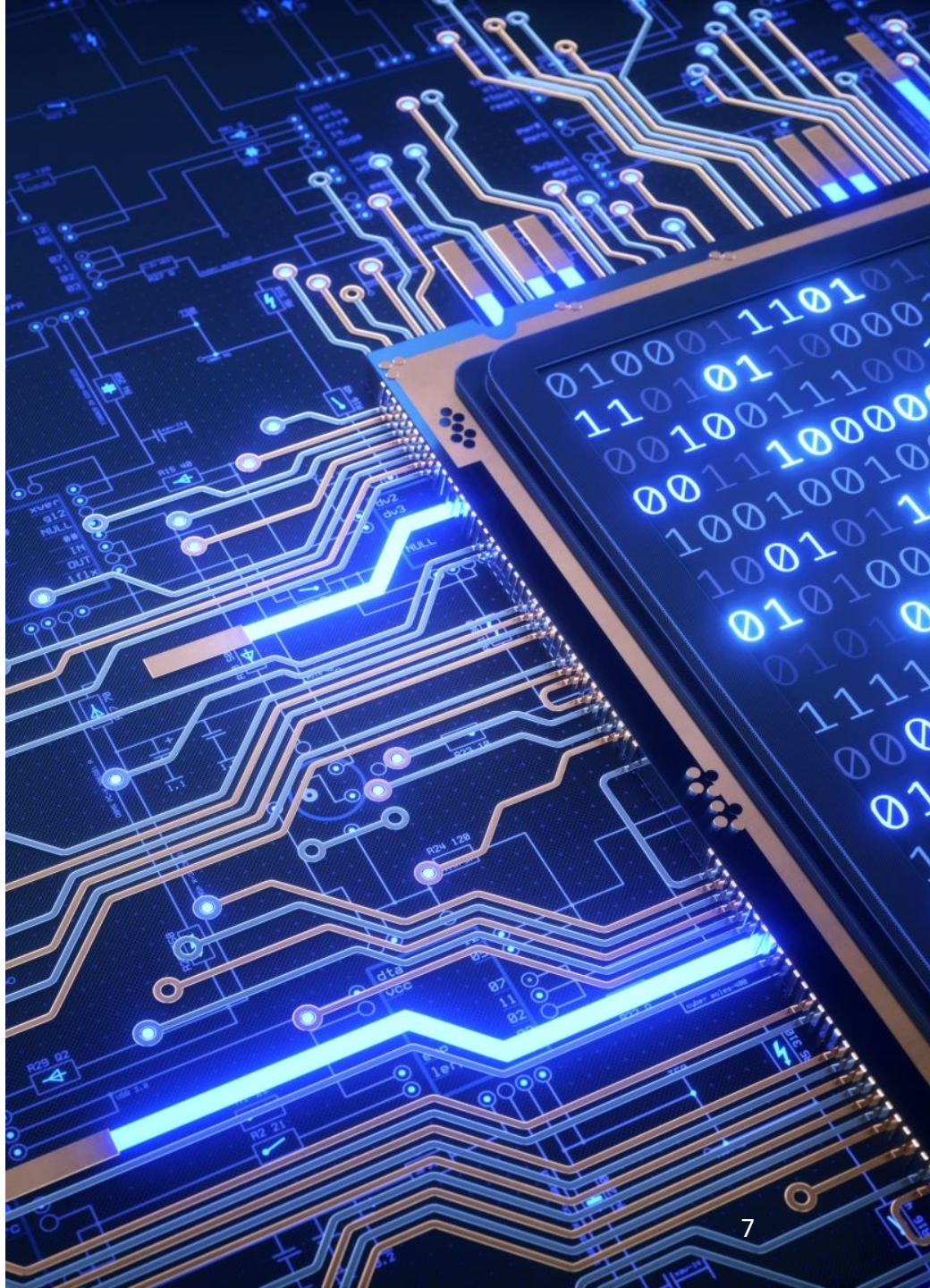




Rosen, K. H. (2012). *Discrete mathematics and its applications (7th Edition)*. McGraw-Hill.

Preliminaries

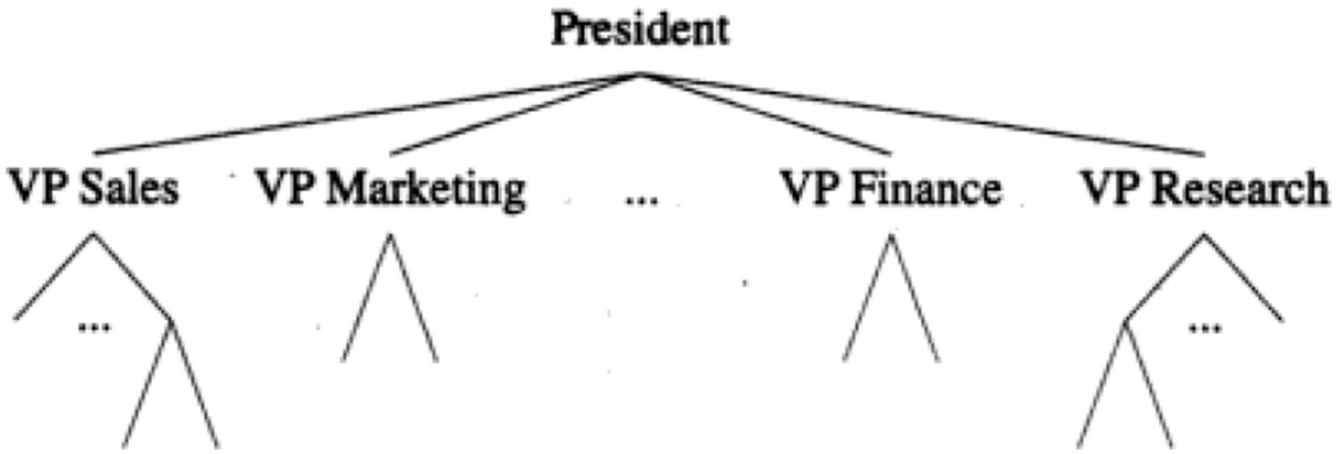
- Most common operations performed over a data structure are; Insert, Delete, and Search
- Performing the required operation over a data structure involves visiting the elements (data) of it in a particular order called ***traversing***
- Choosing a data structure for an application depends on
  - The type and amount of data to be stored
  - Cost (time complexity) of various operations
  - Memory occupation
  - Ease of implementation (least preferred)

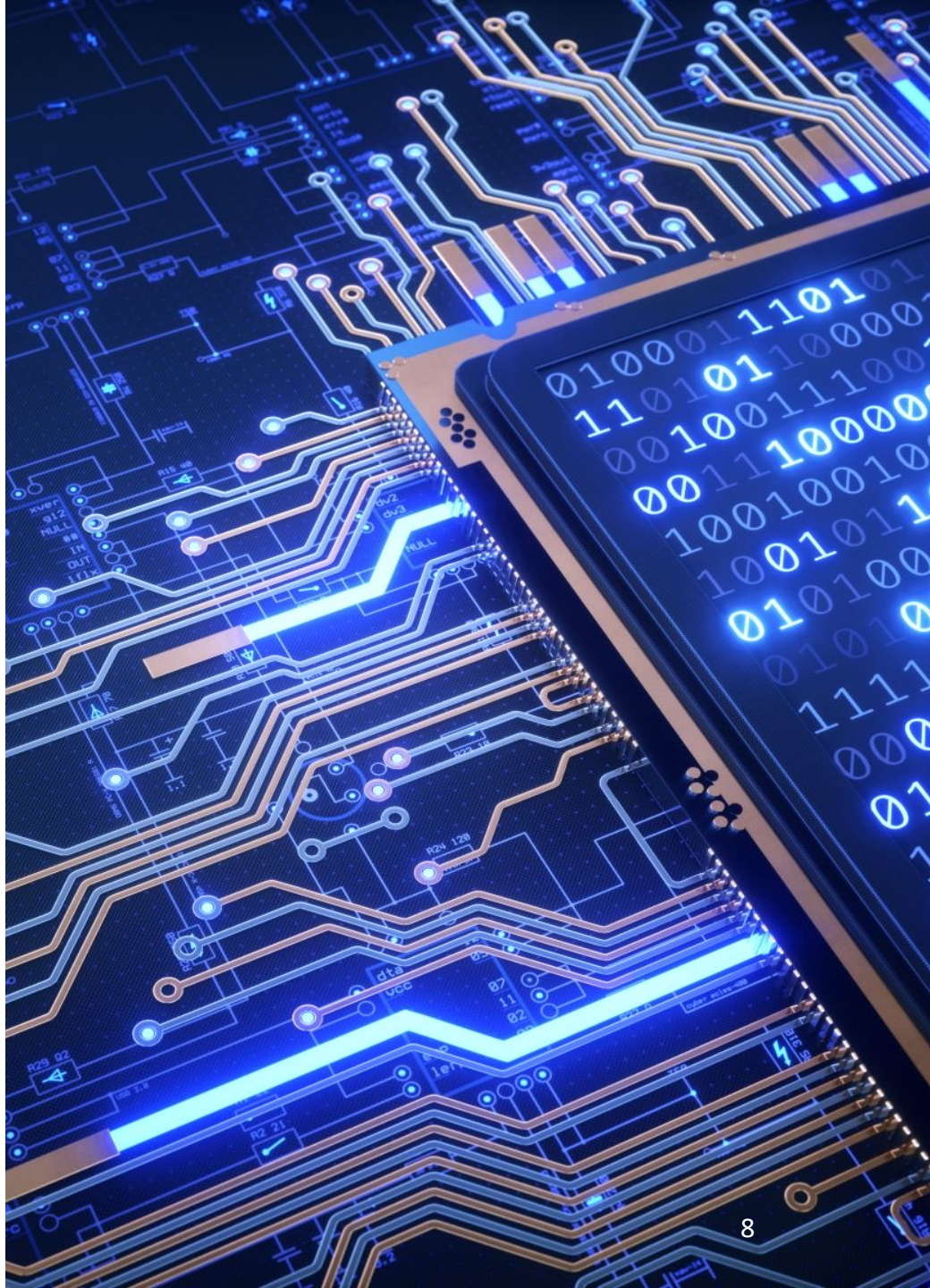


Rosen, K. H. (2012). *Discrete mathematics and its applications (7th Edition)*. McGraw-Hill.

Introduction

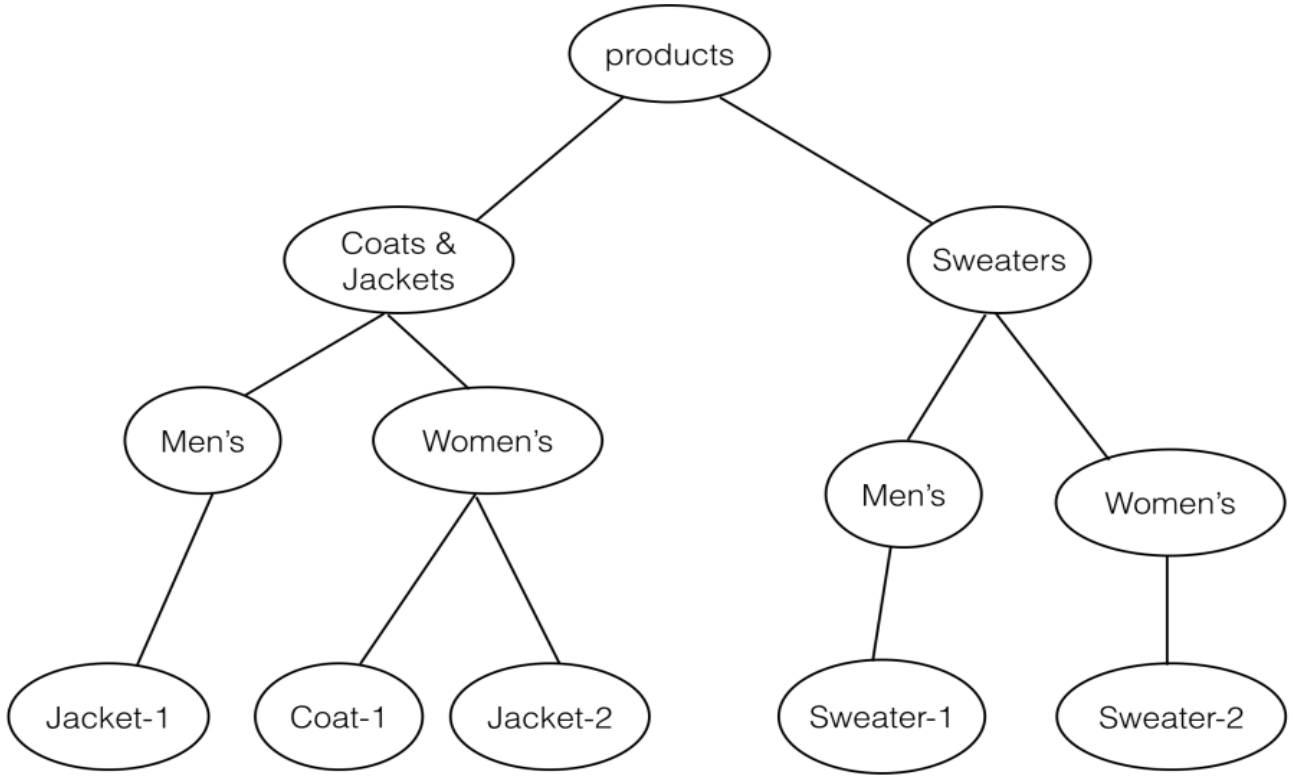
- A tree is a **non-linear** data structure that is used to store data in a **hierarchical** manner
- Example 1: Hierarchical administrative structure of a corporation

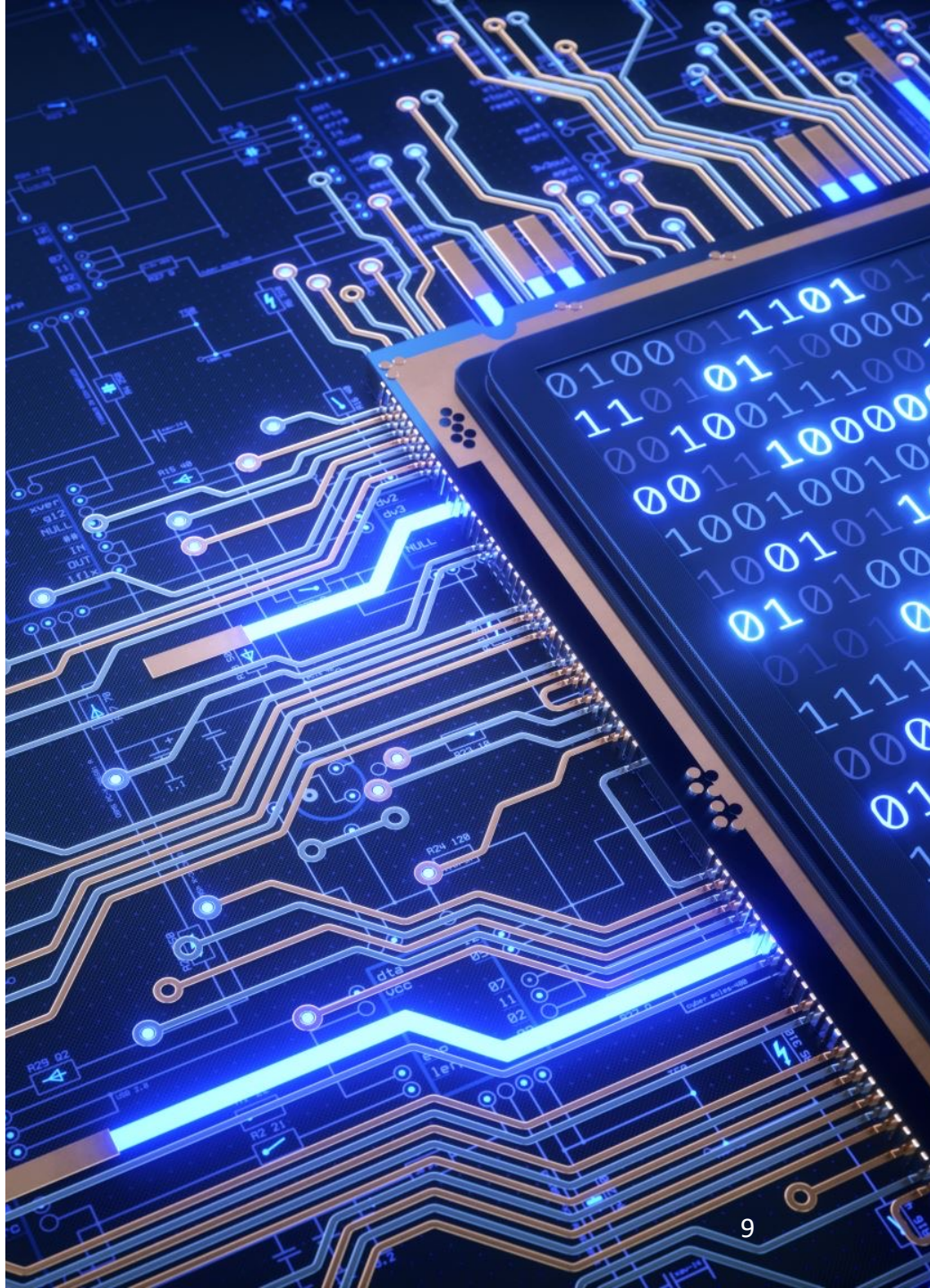




Introduction

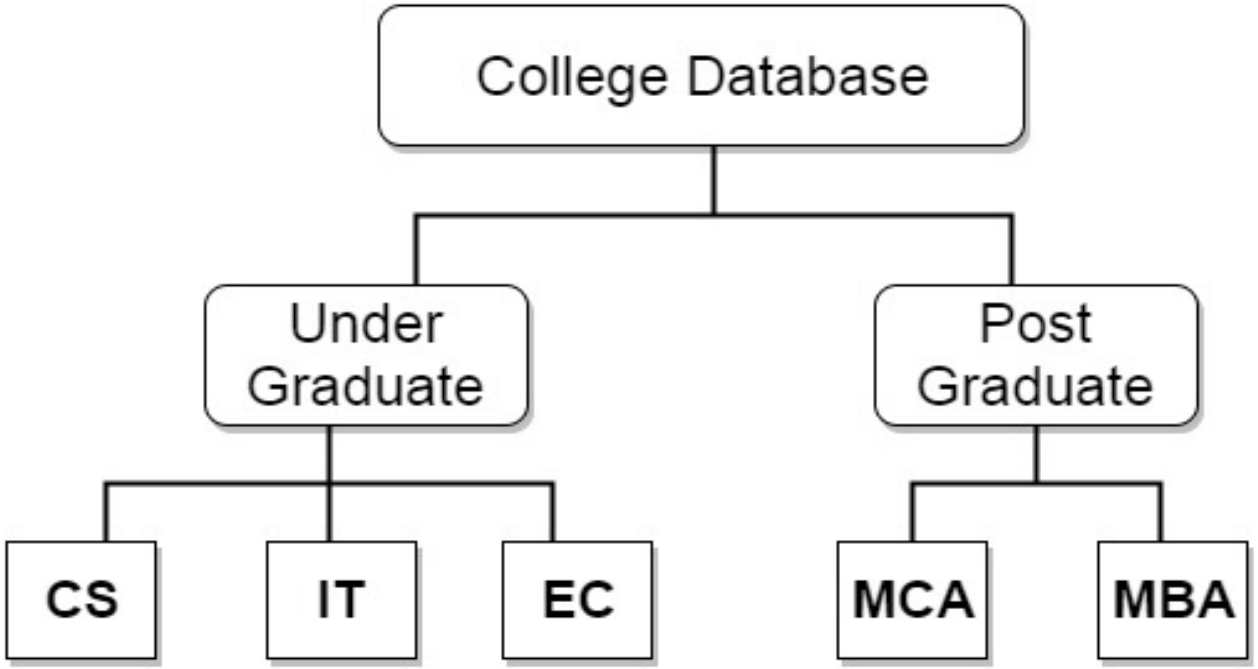
- Example 2: Product information categorization

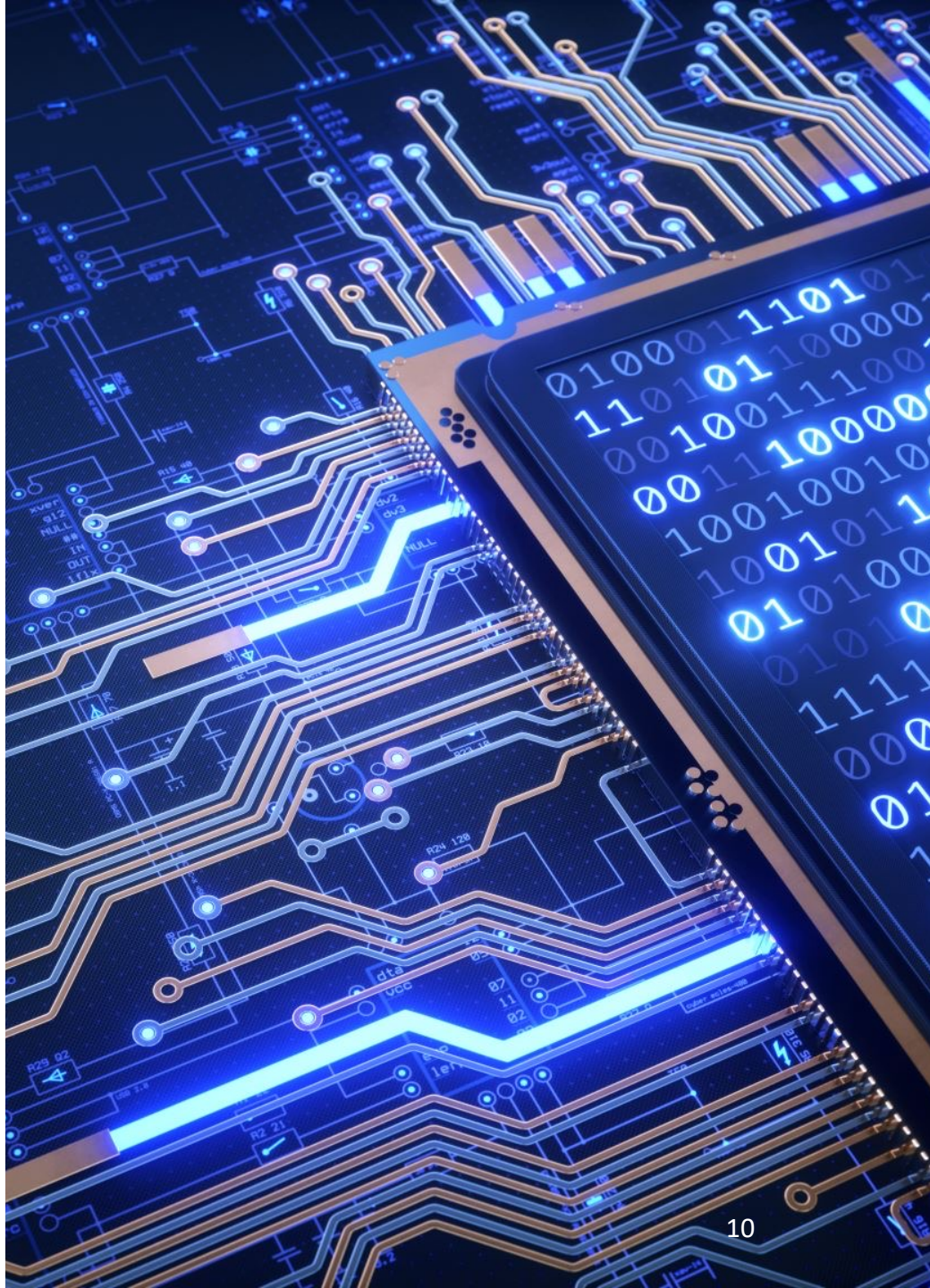




Introduction

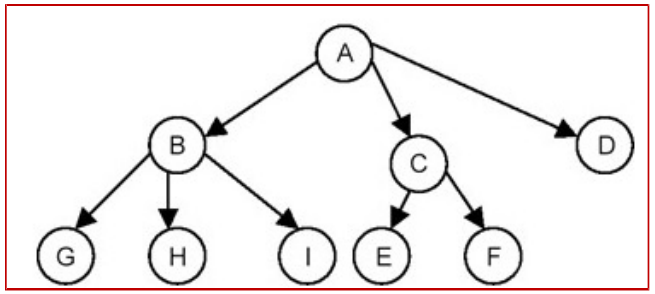
- Example 3: Student information categorization in a University



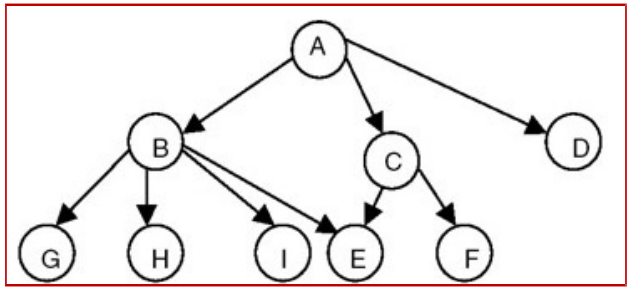


What is a Tree?

- A tree is a finite set of one or more nodes such that there is a specially designated node called the root and the remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1,\ldots, T_n$ where each of these sets is a tree.
- The sets $T_1,\ldots, T_n$ are called the subtrees of the root.

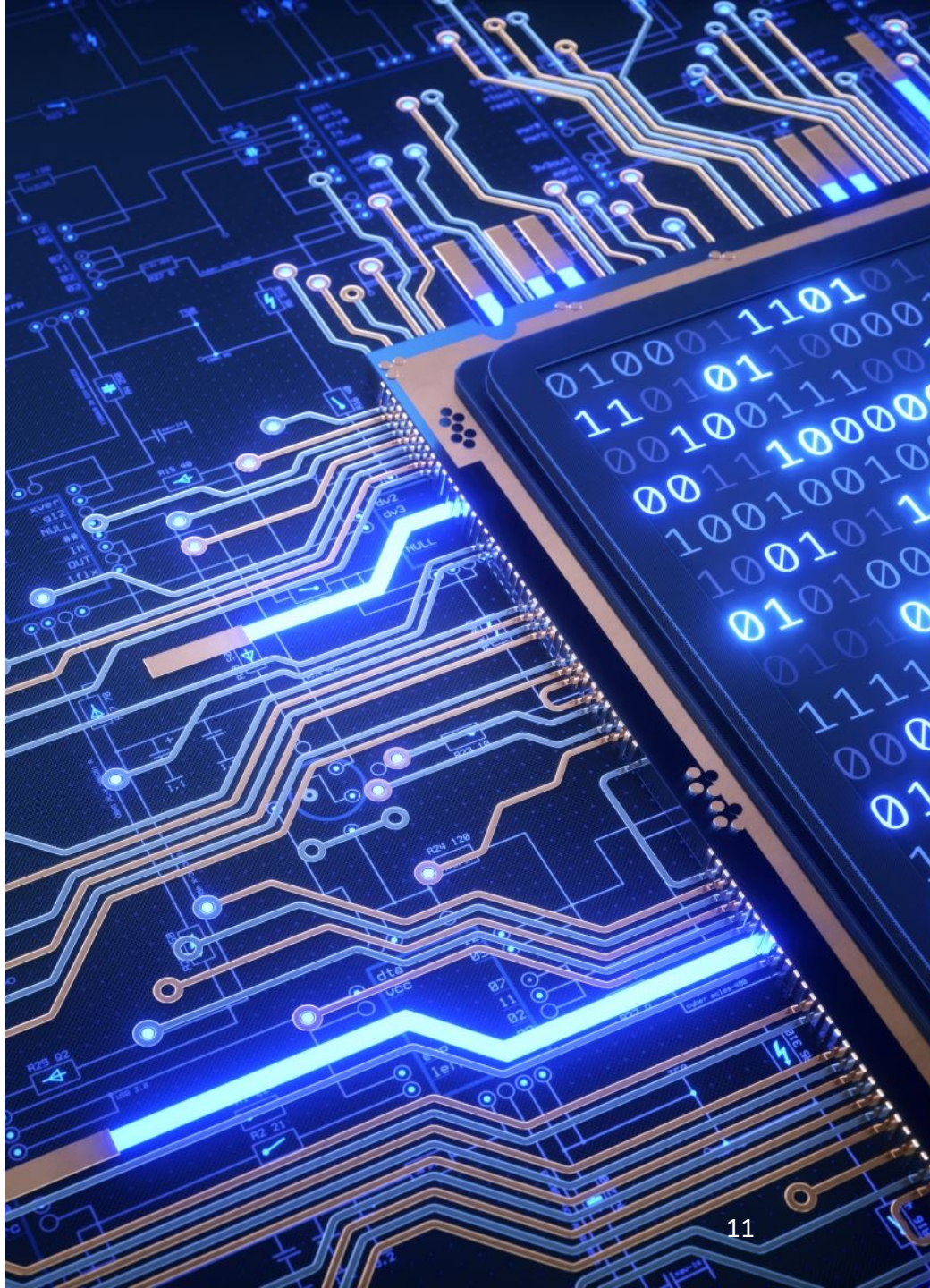


A tree structure



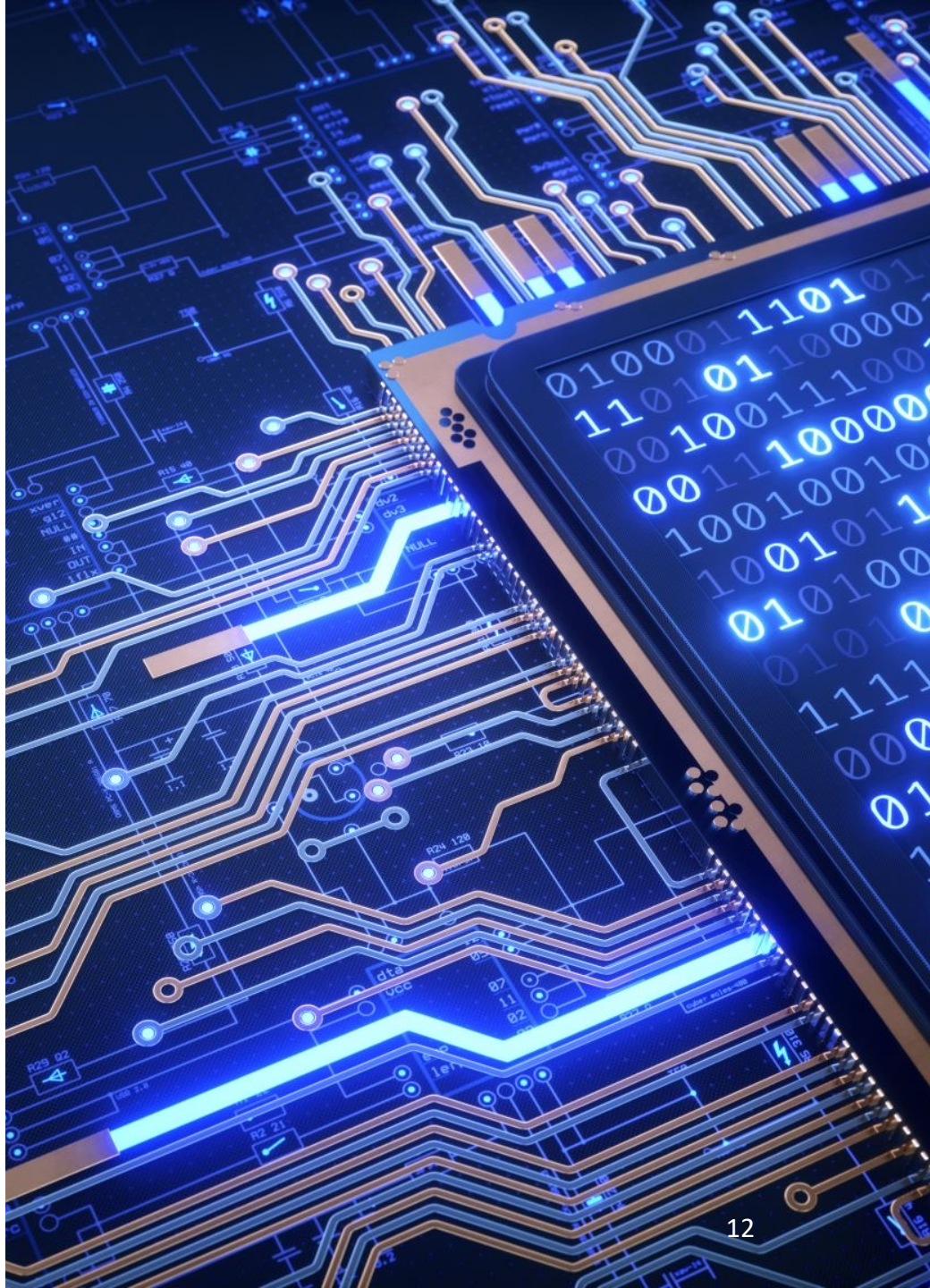
A non-tree structure
(Nodes B, G, H, I can't be made disjoint)

Rosen, K. H. (2012). *Discrete mathematics and its applications (7th Edition)*. McGraw-Hill.



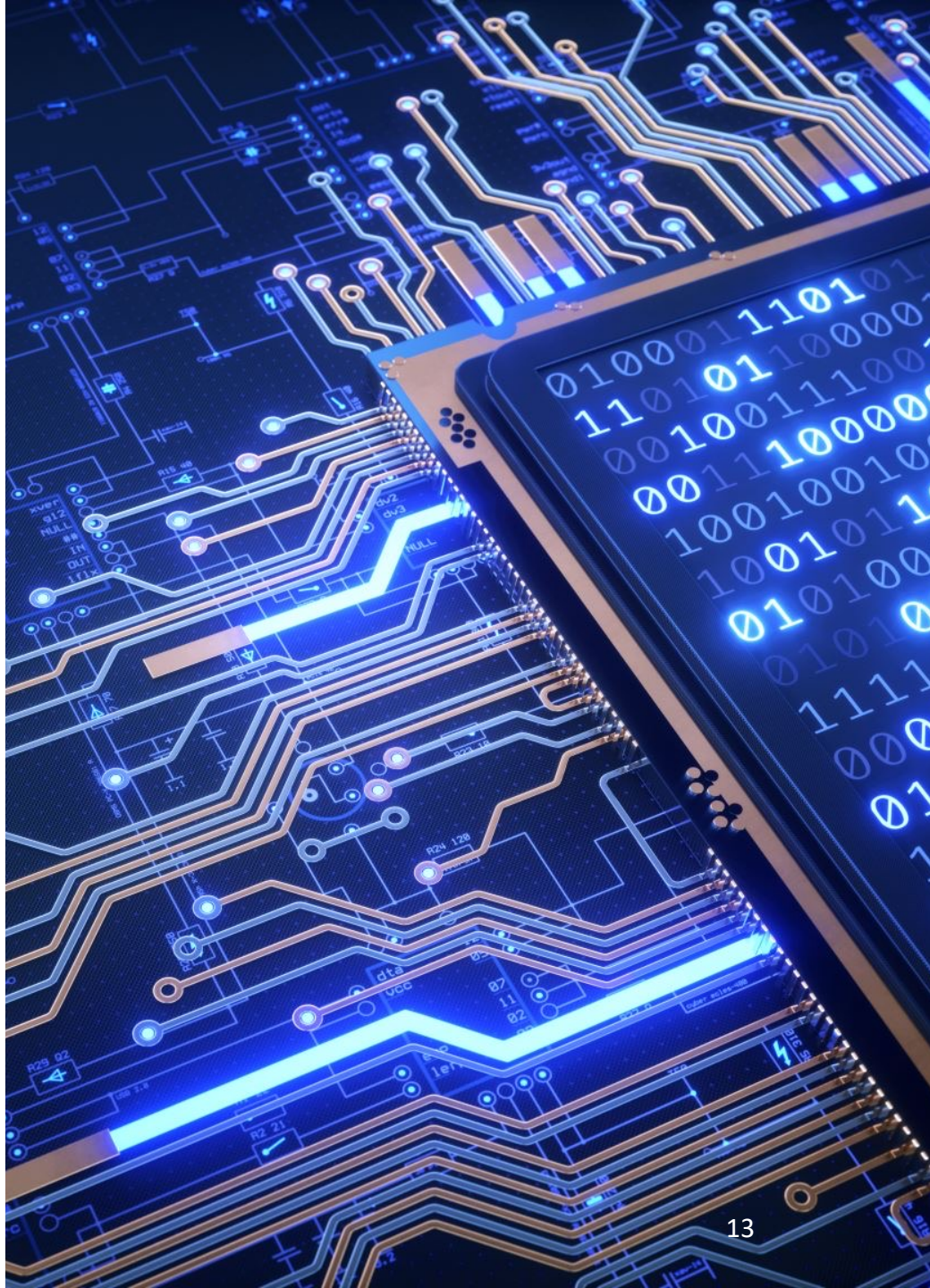
Terminology

- We normally draw trees with their roots at the top
- If a node is connected to other nodes below it, the top node is called the **parent** and the nodes below it are called its **children**
- The number of subtrees of a node is called its **degree**
- Nodes that have degree zero (or no children) are called **leaf** or terminal nodes or external nodes.
- Nodes other than leaf nodes are called **internal** or nonterminal nodes
- Children of the same parent are called **siblings.**



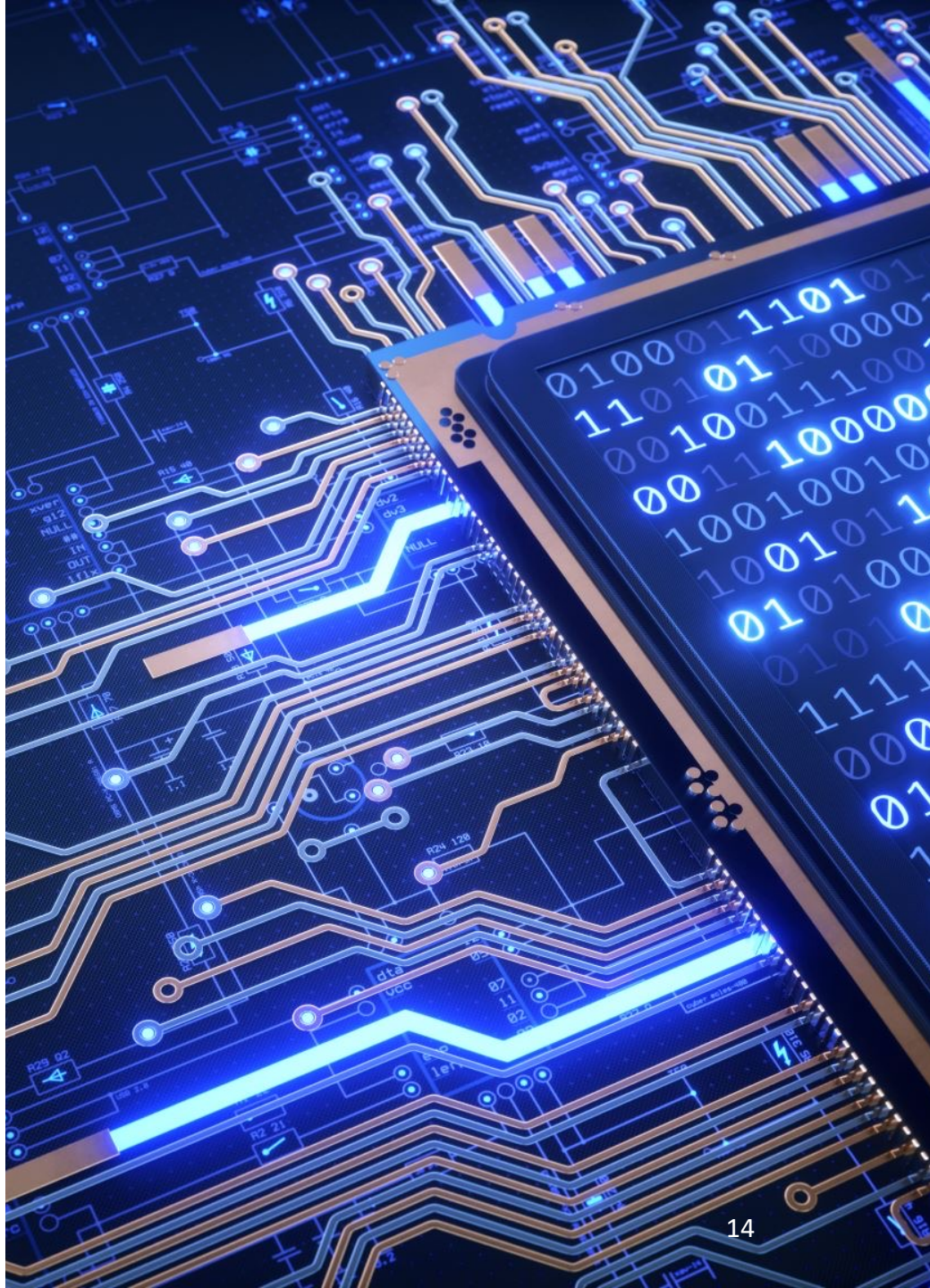
Terminology

- The **degree of a tree** is the maximum degree of the nodes in the tree
- The **ancestors** of a node are all the nodes along the path from the root to that node.
- The **level of a node** is defined by initially letting the root be at level 0 (some books use root level as 1)
- If a node is at level p, its children are at level p + 1.
- The **height** or **depth** of a tree is defined to be the maximum level of any node in the tree.



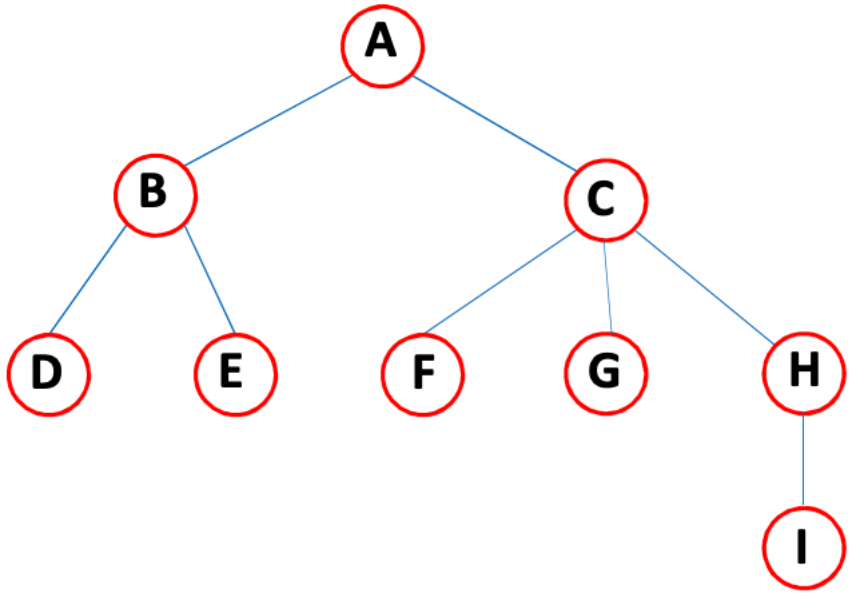
Terminology

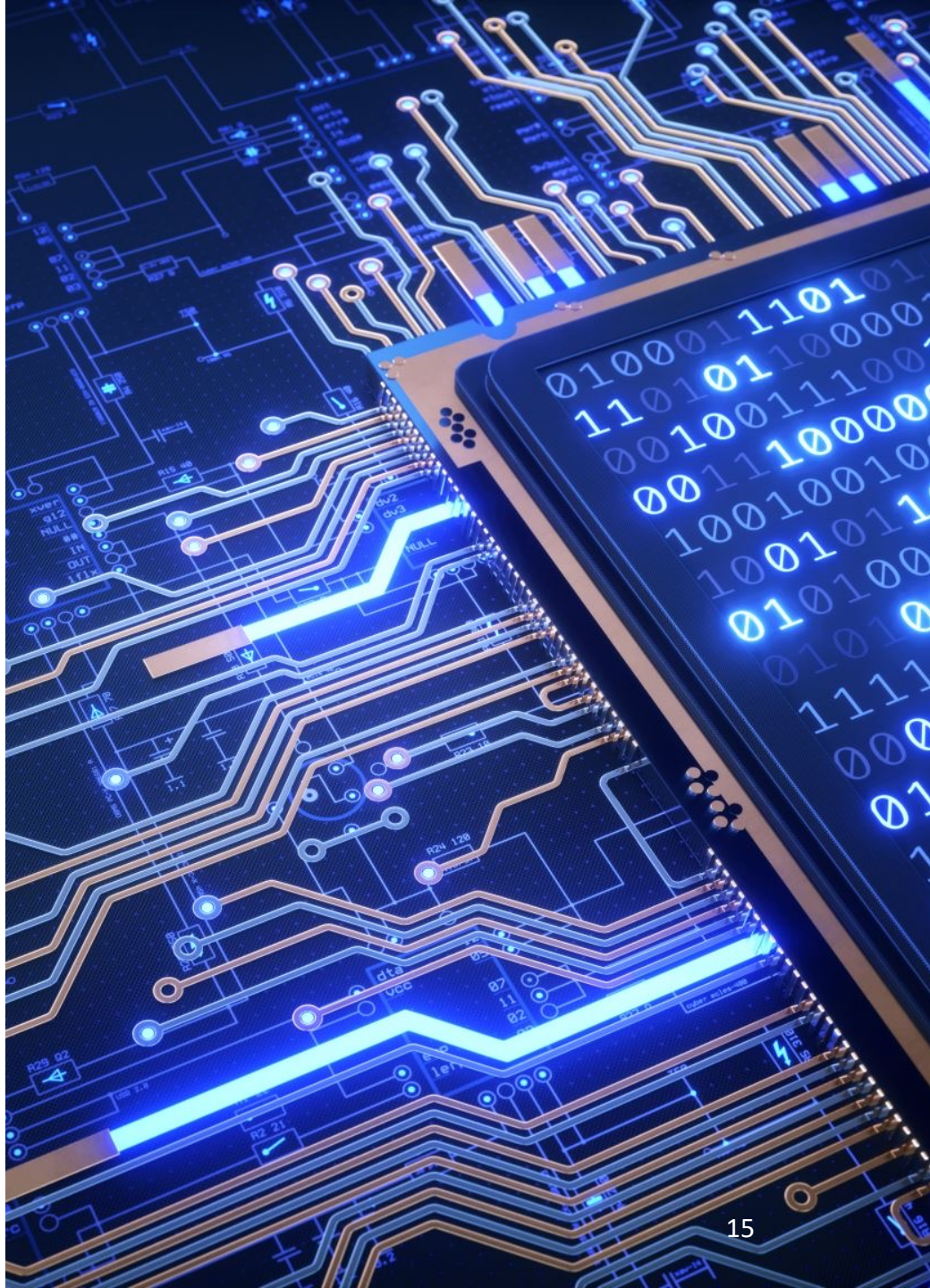
- A **path** is a series of edges from a higher level node to a lower level node. Thus, **path length** or **distance** of a node from an ancestor of it, is the number of edges in between.
- A node at level *i* is at a distance of *i* from the root.
- The **size** of a tree is the total number of nodes in it.
- A forest is a set of n ≥ 0 disjoint trees. For example, if we
- remove the root of a tree, we get a forest.



Terminology

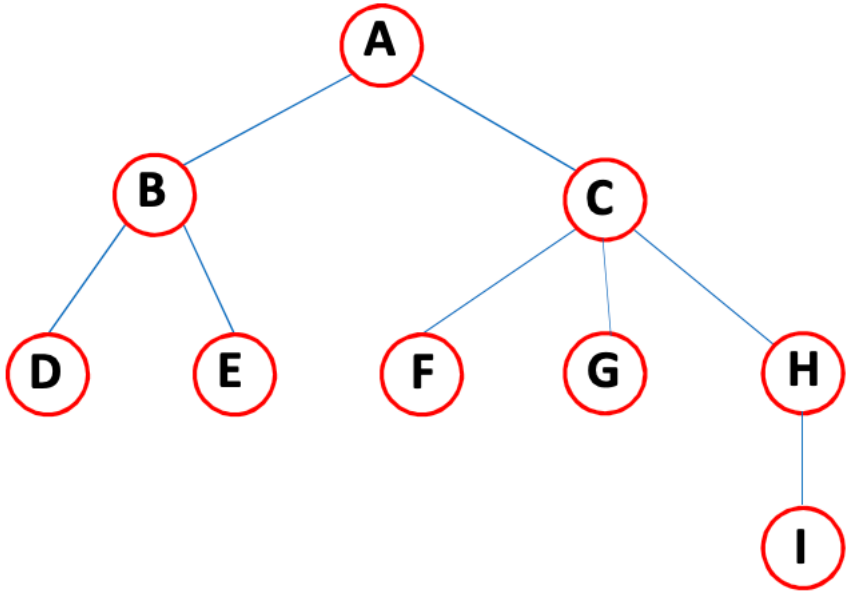
- A is the root node
- B is the parent of D & E
- D & E are the children of B
- A is ancestor of D & E
- D & E are descendants of A
- C is the sibling of B
- D, E, F, G, I are leaf nodes or external nodes
- A, B, C, H are internal nodes or nonterminal nodes

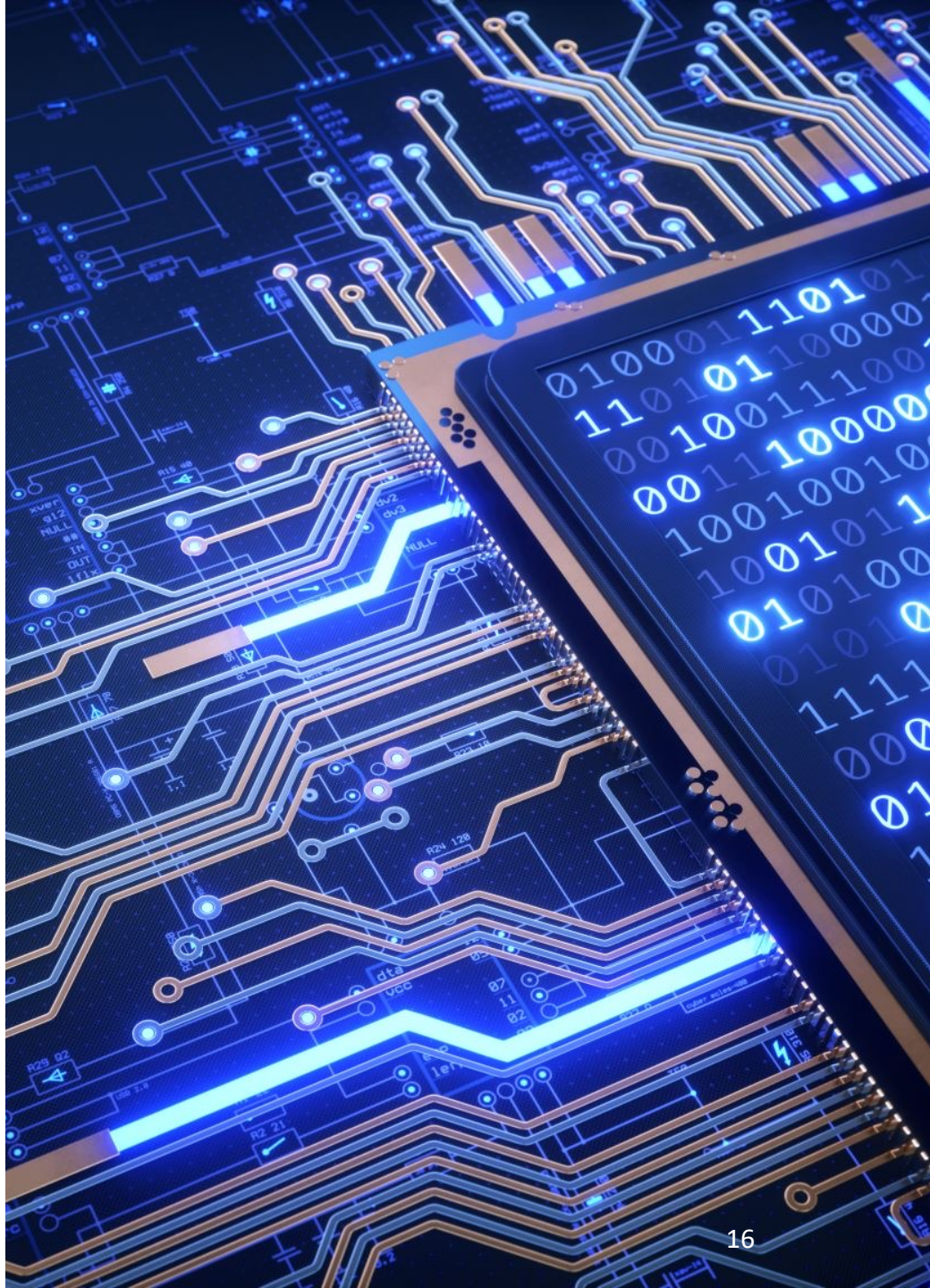




Terminology

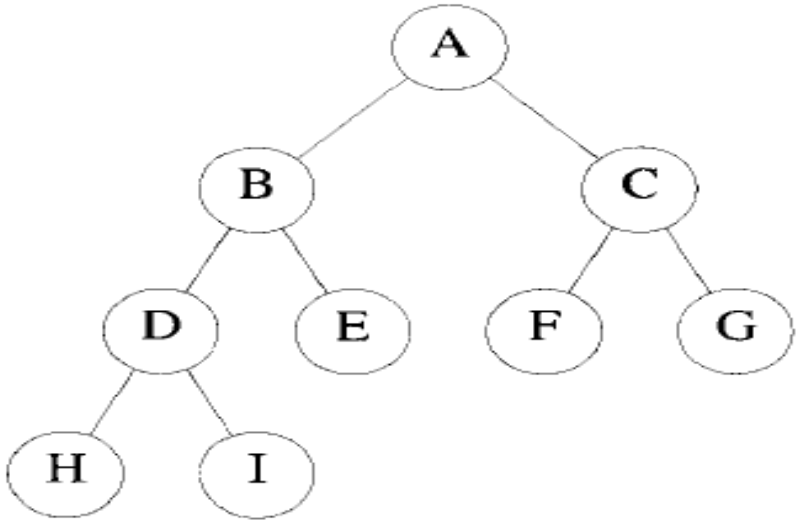
- The level of D is 2
- The level of I is 3
- The height or depth of the tree is 3
  - (maximum level)
- The degree of B is 2
- The degree of C is 3
- The degree of the tree is 3
- The size of the tree is 9

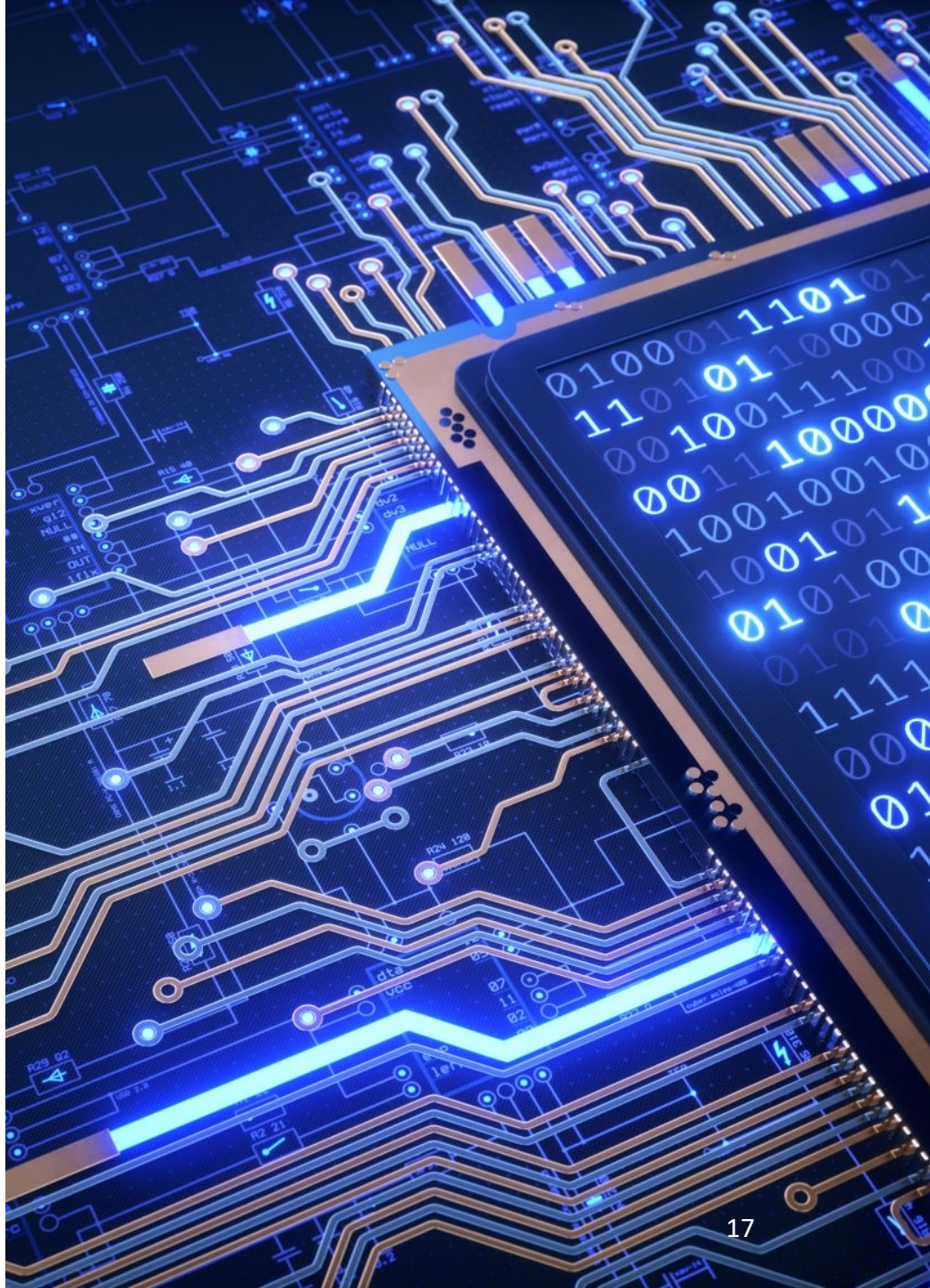




What is a Binary Tree?

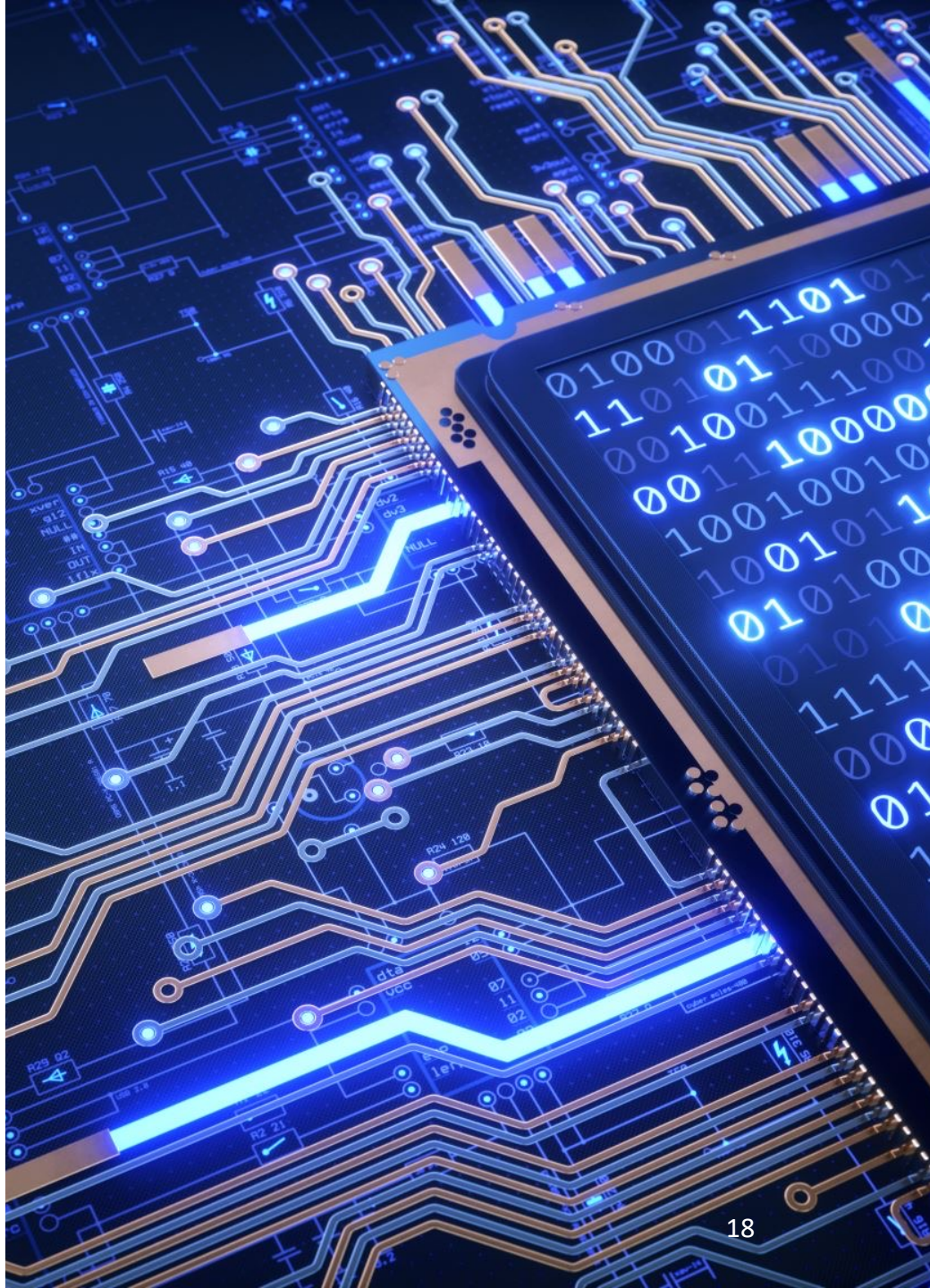
- A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left and right subtrees.
- A binary tree can be defined recursively as: A single node with either no children, or with two children.



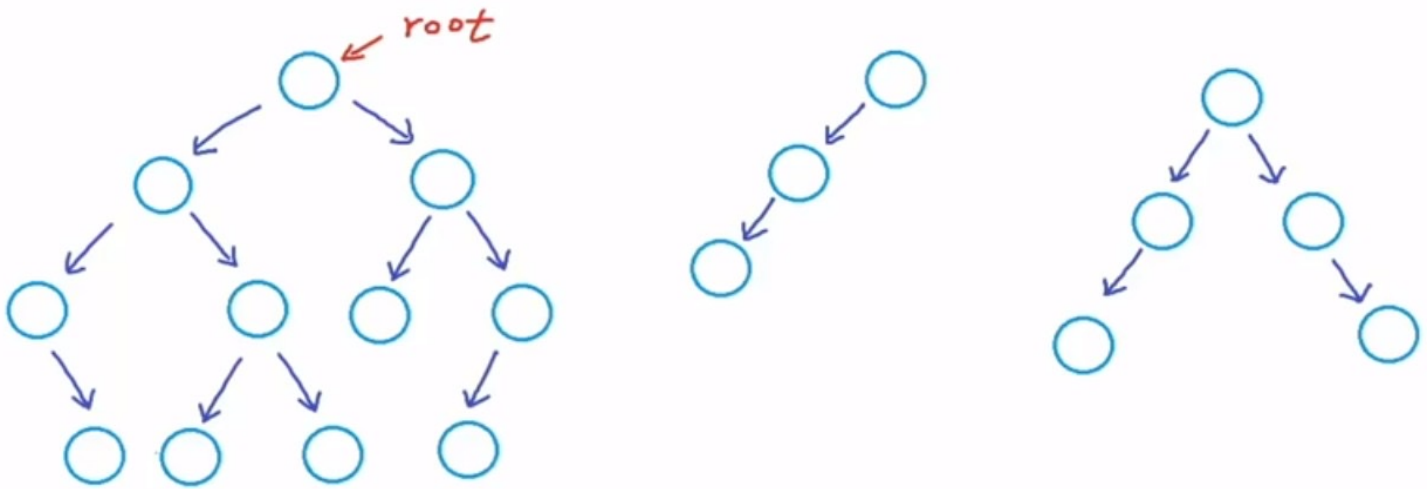


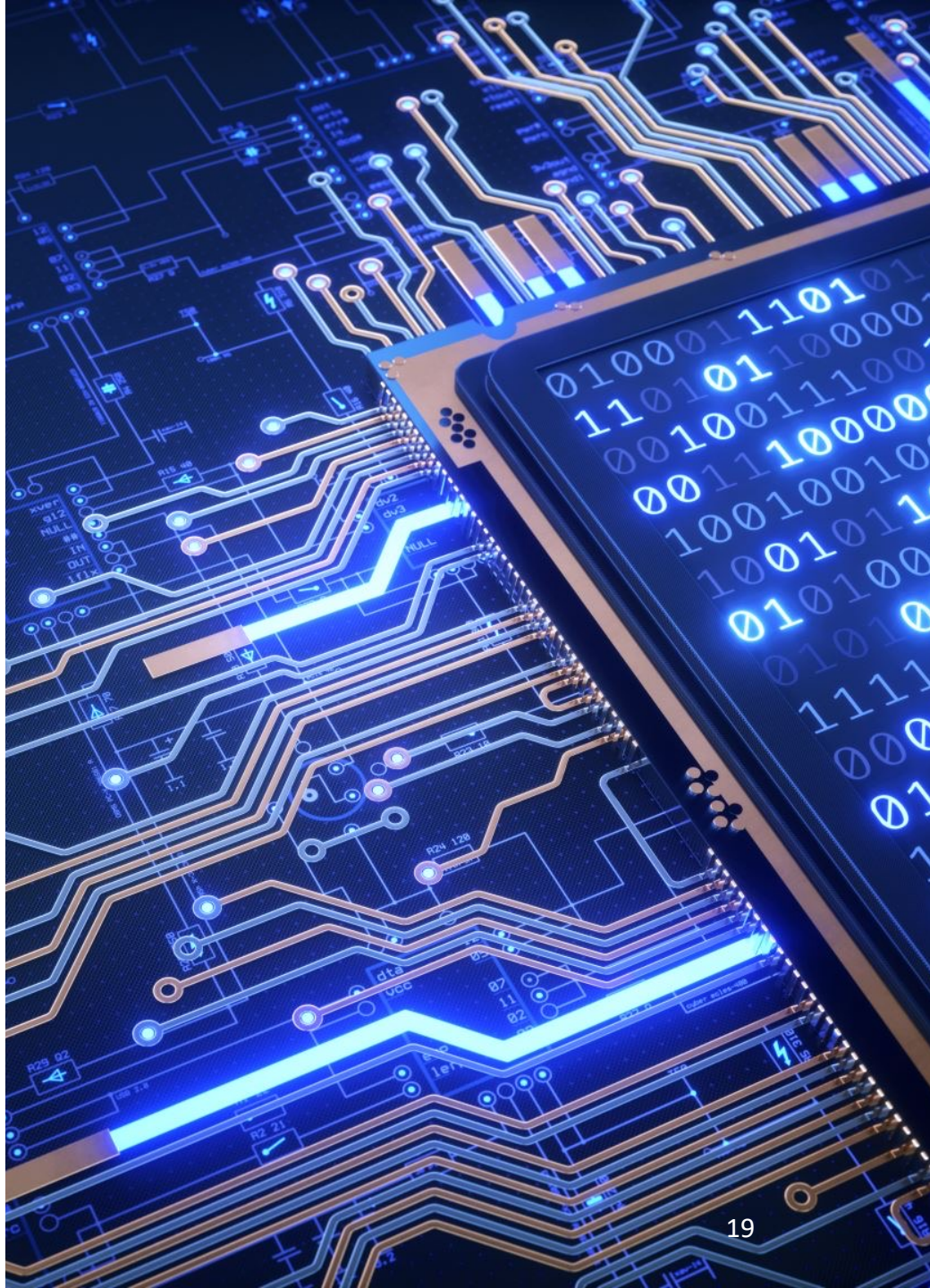
What is a Binary Tree?

- A binary tree is a special and important tree data structure
- The most important characteristic of binary tree is that any node in the tree can have at most two children. In other words, no node will have degree greater than 2.
- The order of the left and right subtrees is significant in binary tree, whereas in a general tree order of subtrees is insignificant
- A binary tree is allowed to have zero nodes, whereas in other trees, there must be at least one node



Are these Binary Trees?

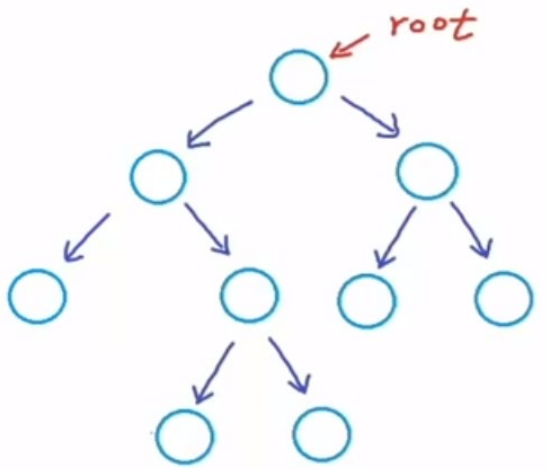




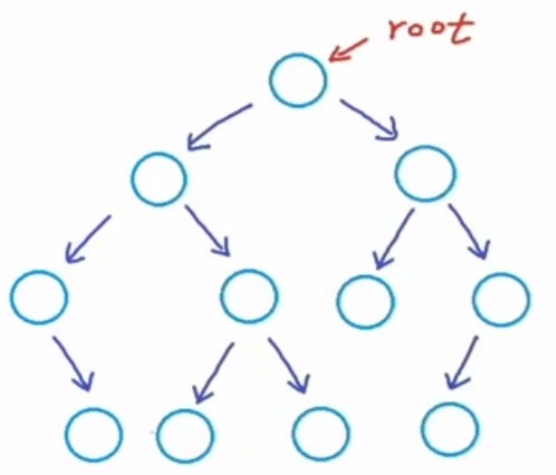
Properties of Binary Trees

- **Strict / proper binary tree**: Each node can have either two or zero children

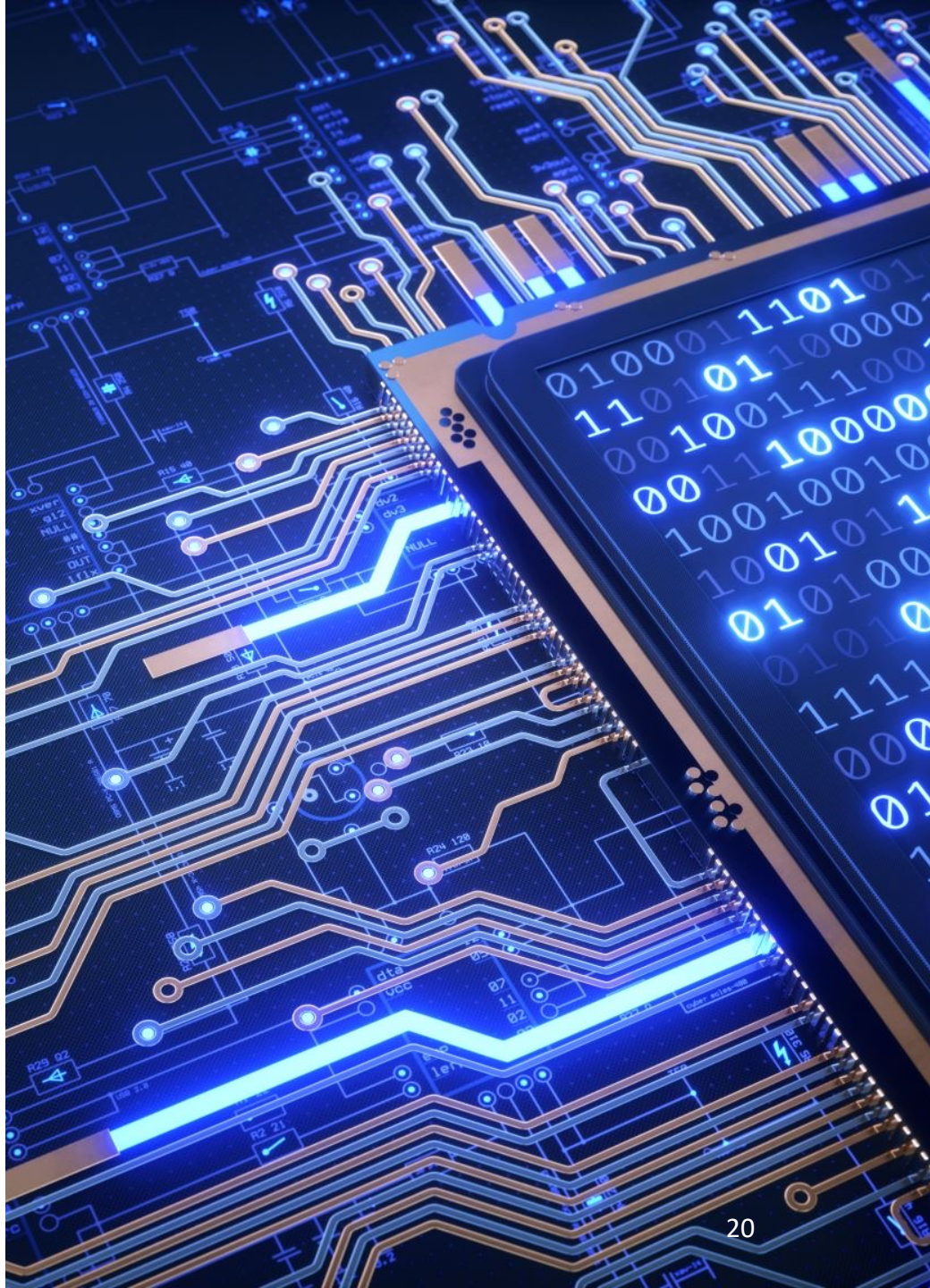
- Is this a proper binary tree?



Proper binary tree

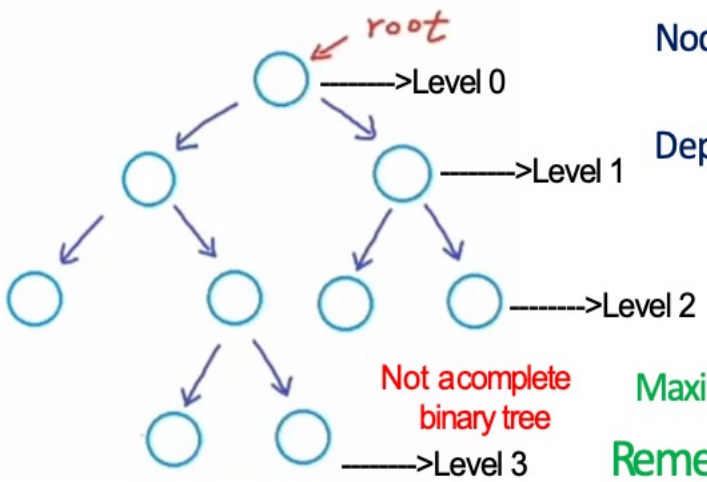


Not a proper binary tree



Properties of Binary Trees

- Complete binary tree: All levels except the last level are completely filled, and all nodes are as left as possible
- Is this a complete binary tree?



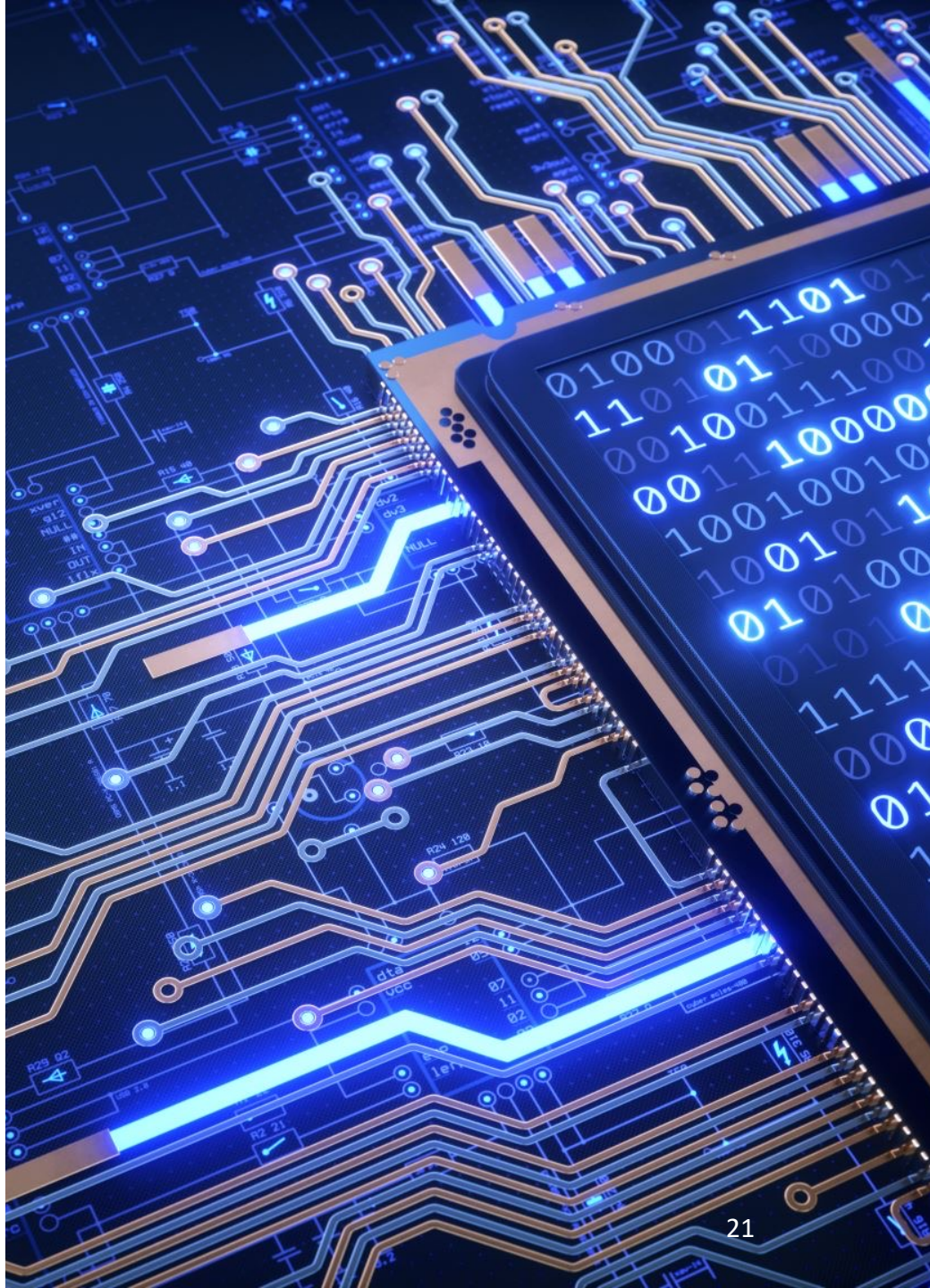
Nodes at the same depth are at the same level

Depth of a node is the length of the path from root to that node

Number of edges from root to a node

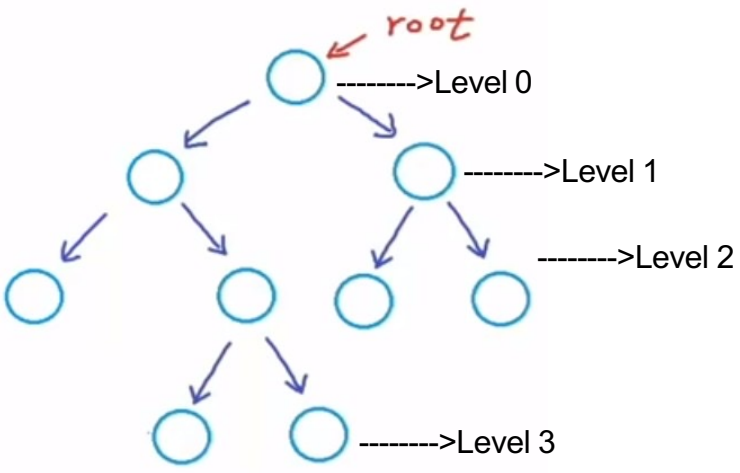
Maximum depth of a tree is equal to its height

Remember: depth of a node is not same as its height



Properties of Binary Trees

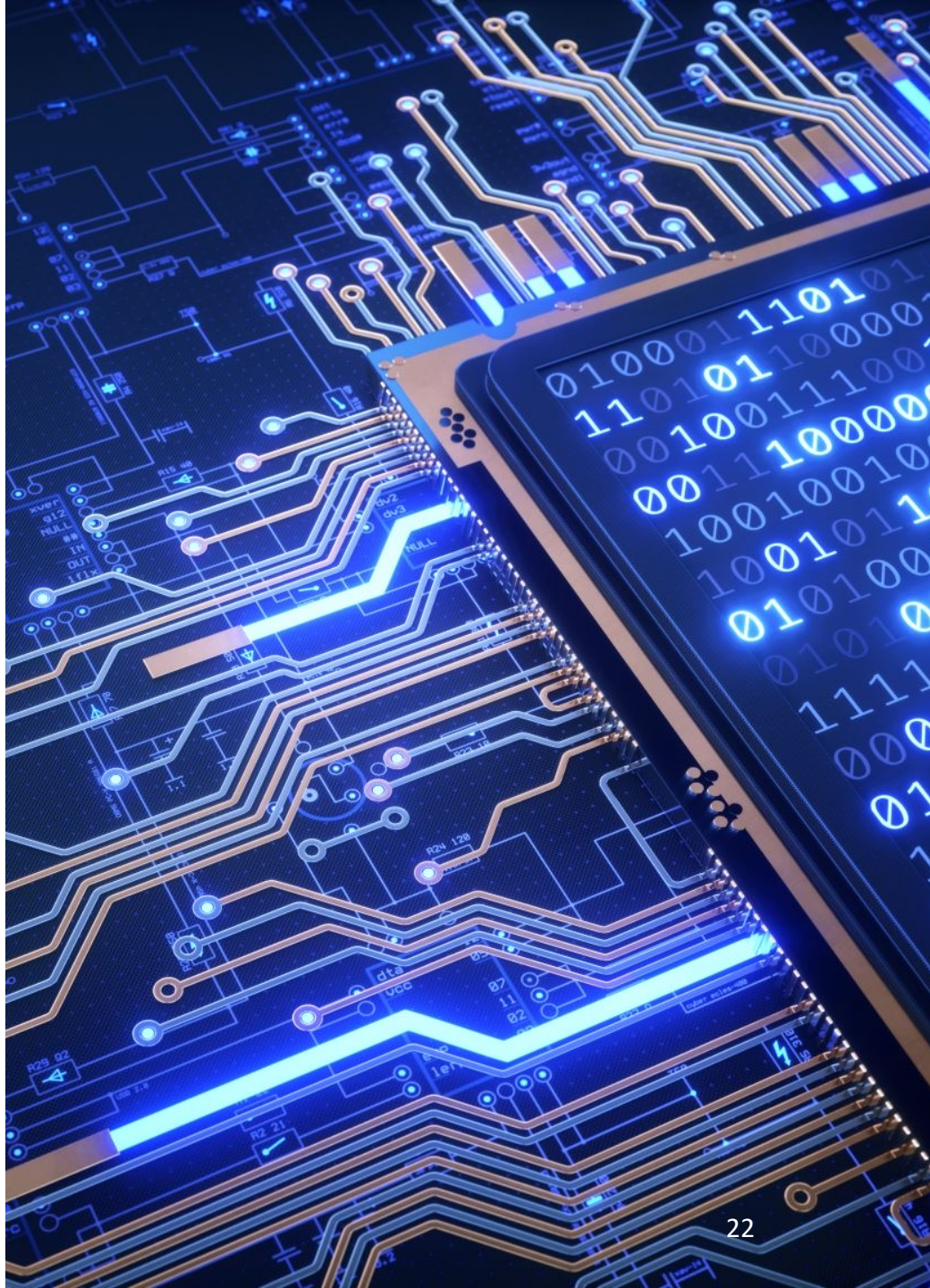
- **Complete binary tree**: All levels except the last level are completely filled, and all nodes are as left as possible



Not a complete binary tree

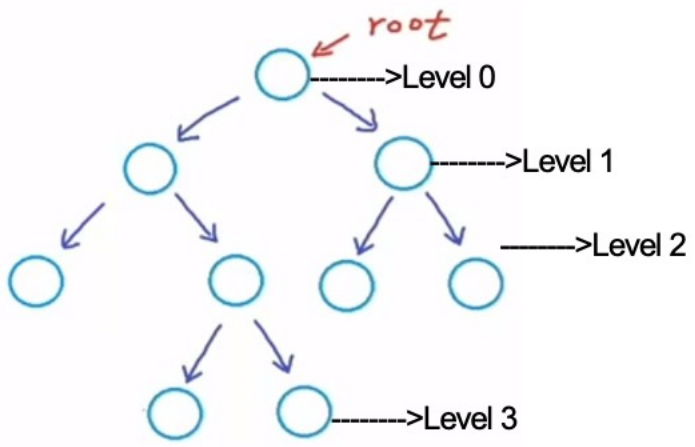
- Height of a node is the longest path from leaf to that node

Maximum number of nodes at some level $i = 2^i$

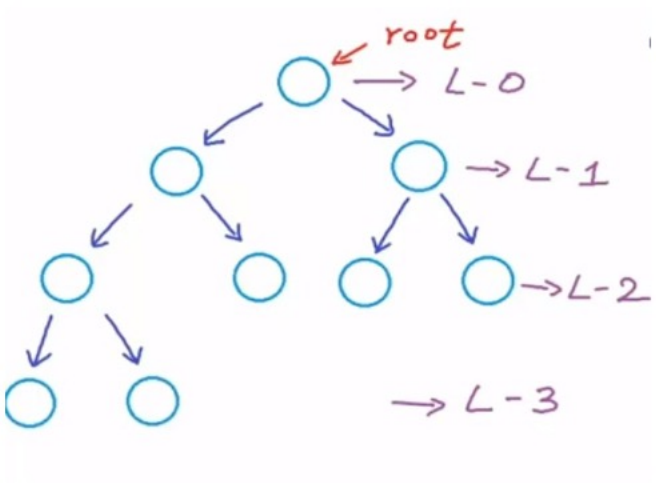


Properties of Binary Trees

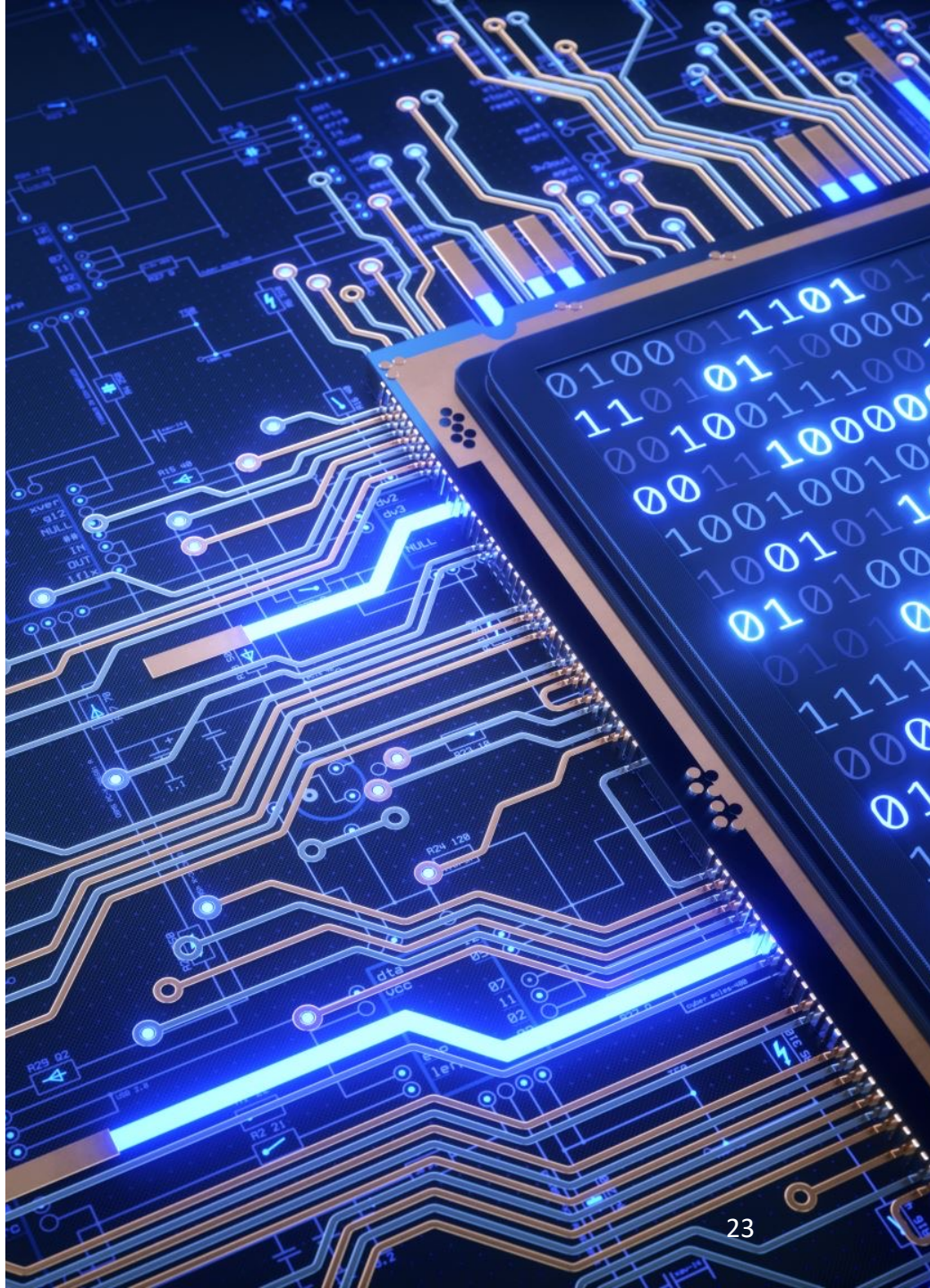
- **Complete binary tree**: All levels except the last level are completely filled, and all nodes are as left as possible



Not a complete binary tree

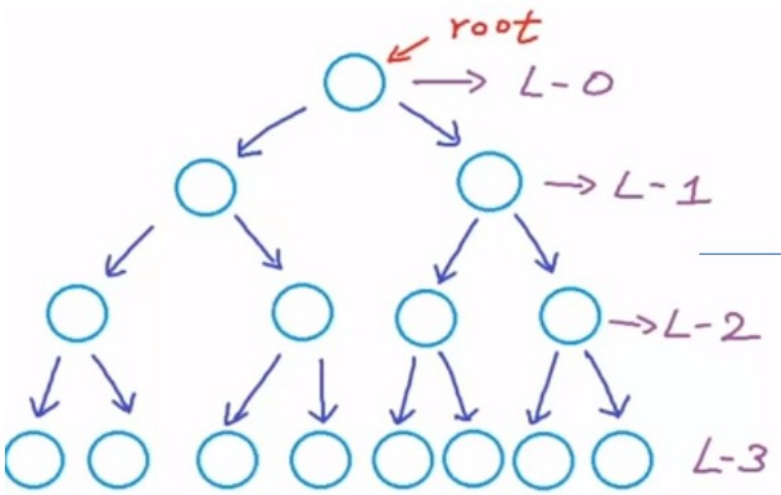


Complete binary tree



Properties of Binary Trees

Perfect binary tree: All levels are completely filled



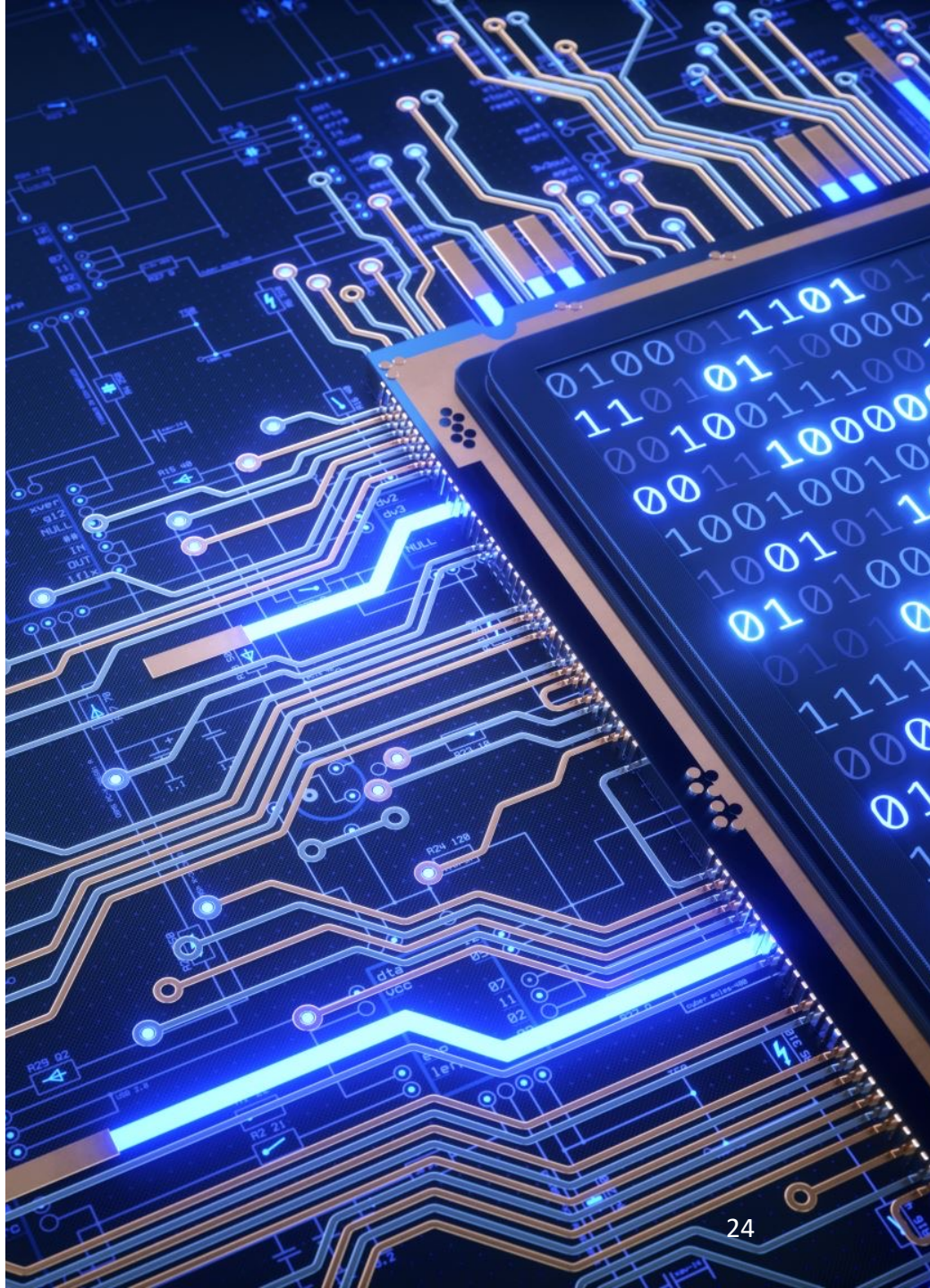
Maximum number of nodes in a binary tree of height *h* is, $2^{h+1} - 1$

(i.e., $2^{Number\ of\ levels} - 1$)

Height = 3

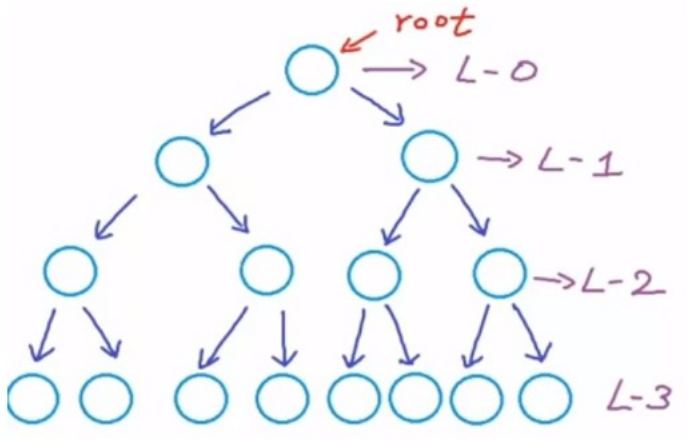
Number of nodes = $2^{3+1}-1 = 15$

So, a perfect binary tree is a binary tree having maximum number of possible nodes



Properties of Binary Trees

Perfect binary tree: All levels are completely filled



A perfect binary tree is a complete binary tree as well, but, not vice-versa

Maximum number of nodes in a binary tree of height h is, $2^{h+1}-1$

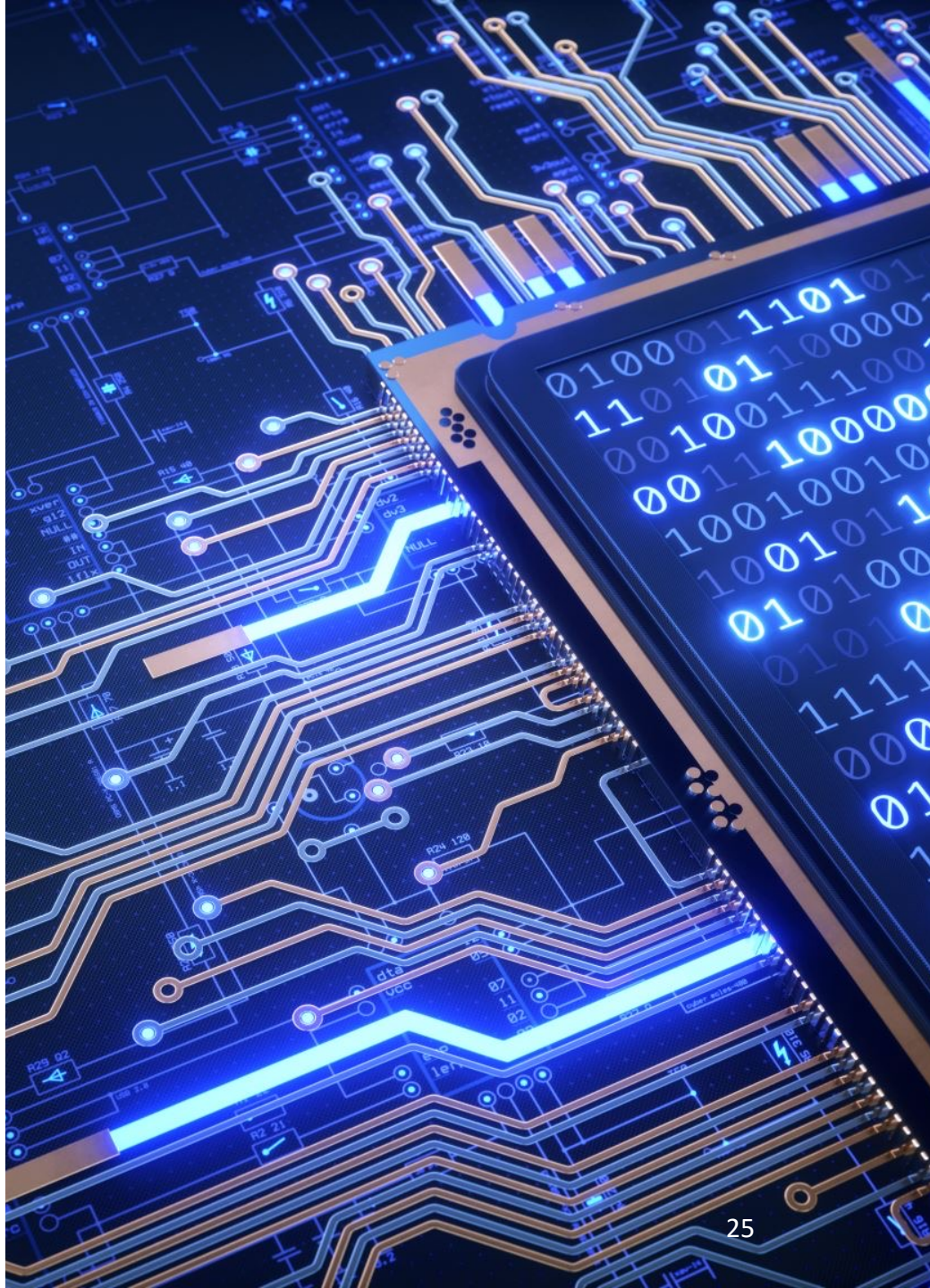
(i.e., $2^{Number\ of\ levels} - 1$)

What is the height of a perfect binary tree having n nodes?

Solve the equation, $n = 2^{h+1}-1$ for h

We get, $h = log_2(n+1) - 1$

For a complete binary tree we get, $h = \lfloor log_2 n \rfloor$



Binary Trees - ADT

AbstractDataType *binaryTree*
{
instances
collection of elements; if not empty, the collection is partitioned into a root, left subtree, and right subtree; each subtree is also a binary tree;

operations

empty() : return **true** if empty, return **false** otherwise;

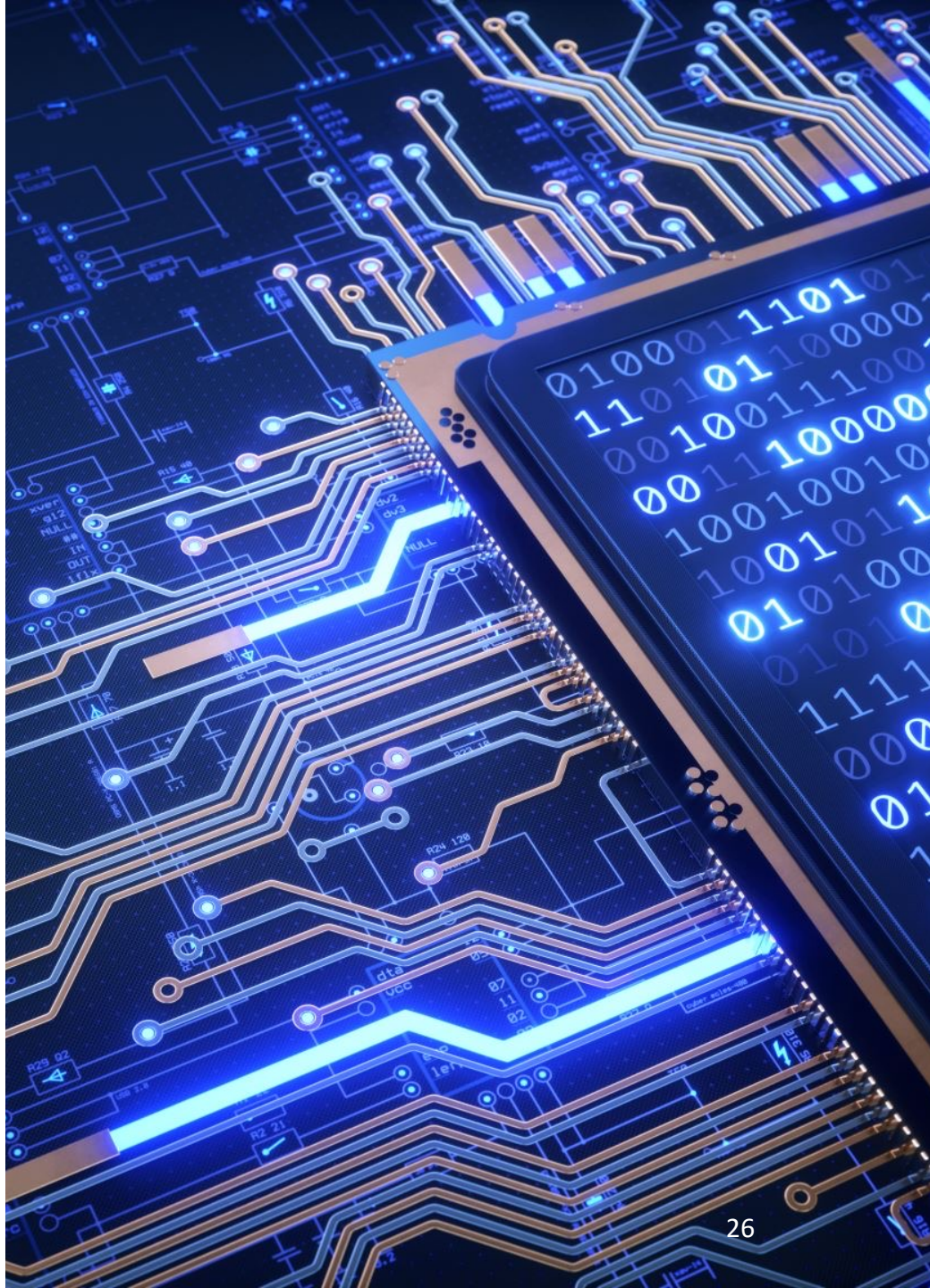
size() : return number of elements/nodes in the tree;

preOrder(*visit*) : preorder traversal of binary tree; *visit* is the visit function to use;

inOrder(*visit*) : inorder traversal of binary tree;

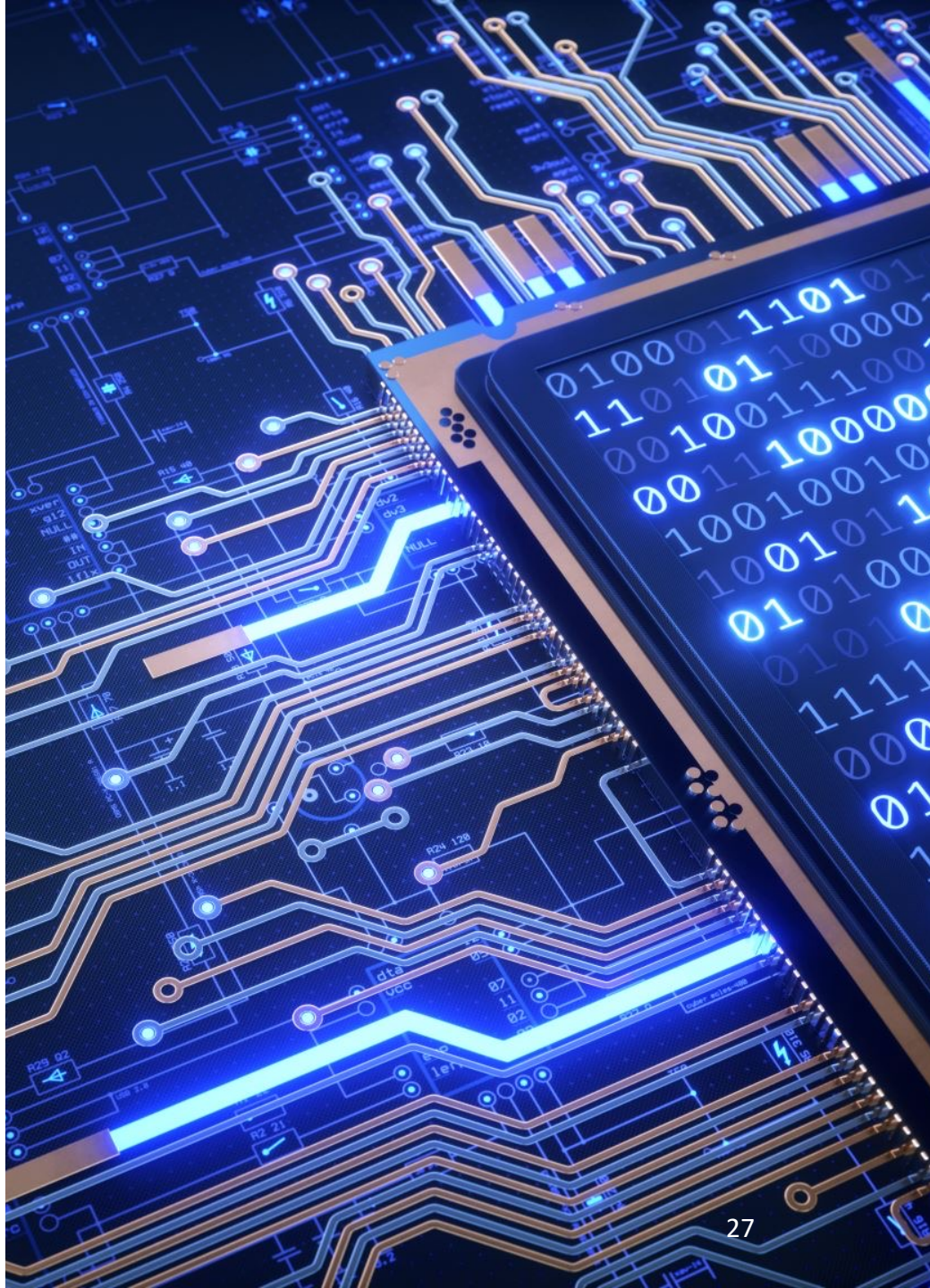
postOrder(*visit*) : postorder traversal of binary tree;

}



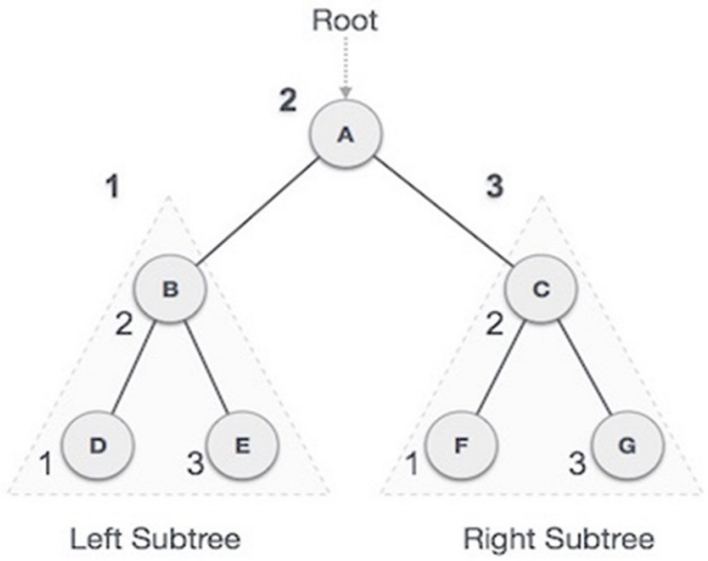
Binary Tree Traversal

- Visiting/examining every node in the tree is called **traversing**
- In general, traversing is done to search a given item (or key) in the tree or to print all the values it contains.
- There are three ways to traverse a binary tree
  - In-order Traversal
  - Pre-order Traversal
  - Post-order Traversal

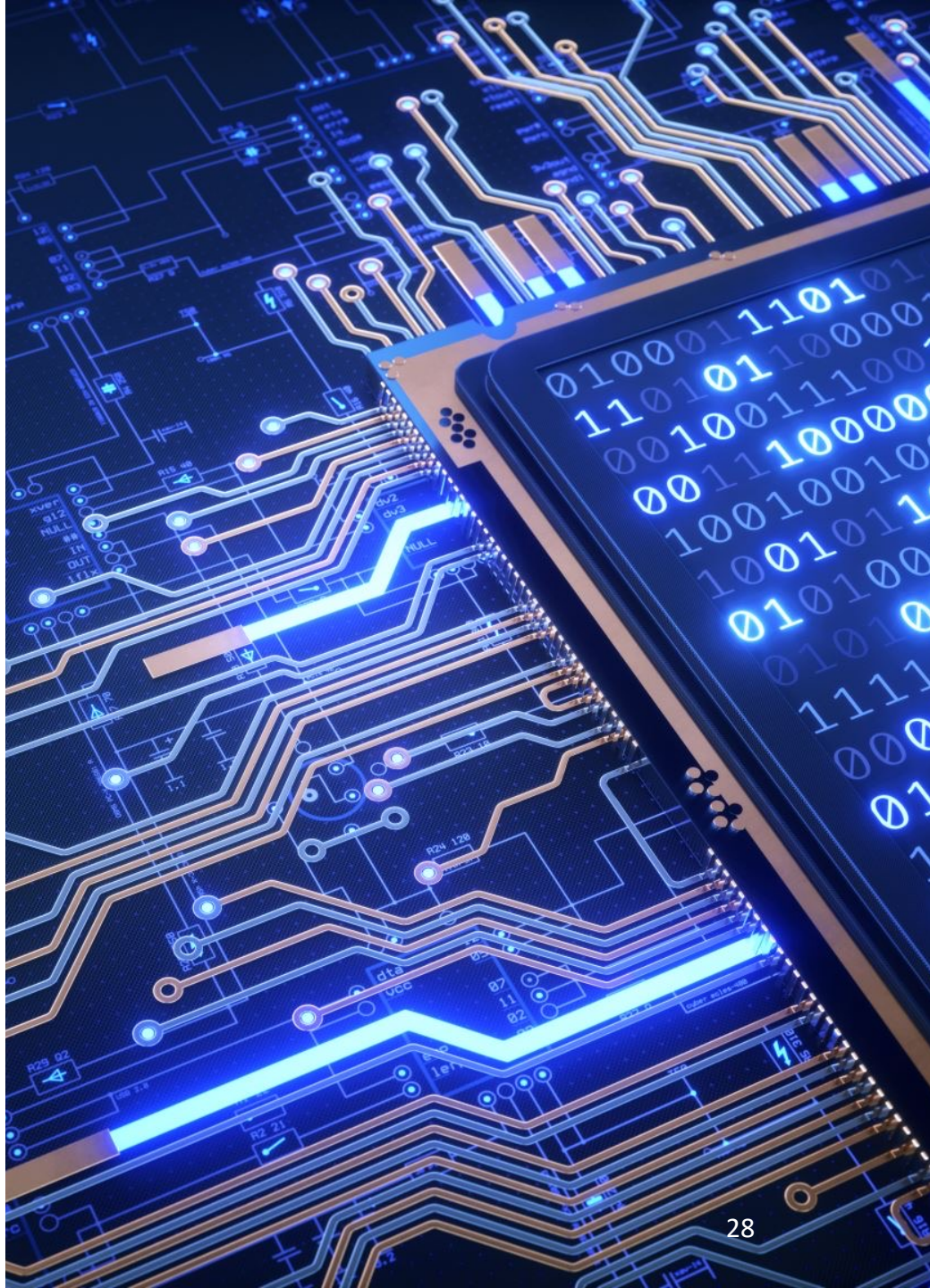


Binary Tree Traversal

- **In-Order Traversal:**
  - Left-sub-tree is visited first, then root and then the right sub-tree.

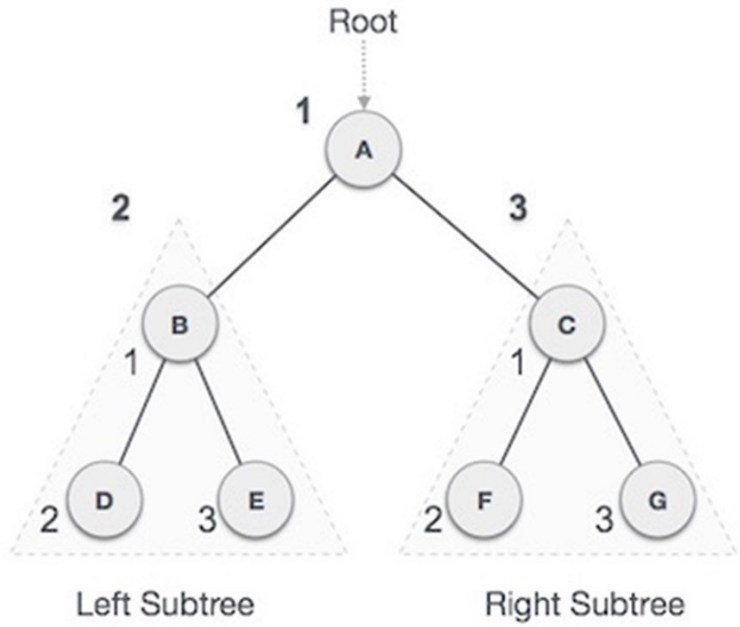


Output: D → B → E → A → F → C → G

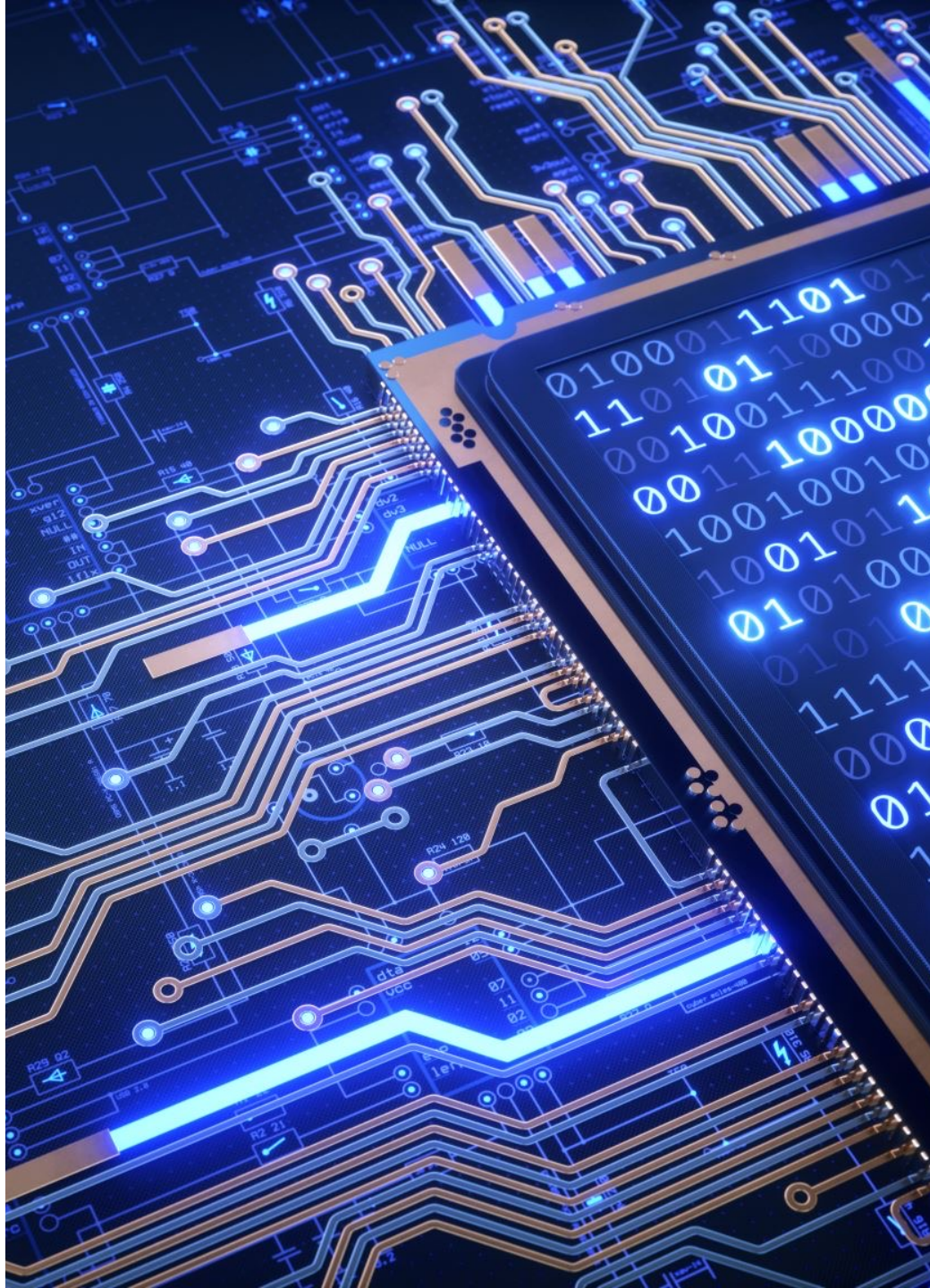


Binary Tree Traversal

- **Pre-Order Traversal:**
  - Root node is visited first, then left sub-tree and finally right sub- tree

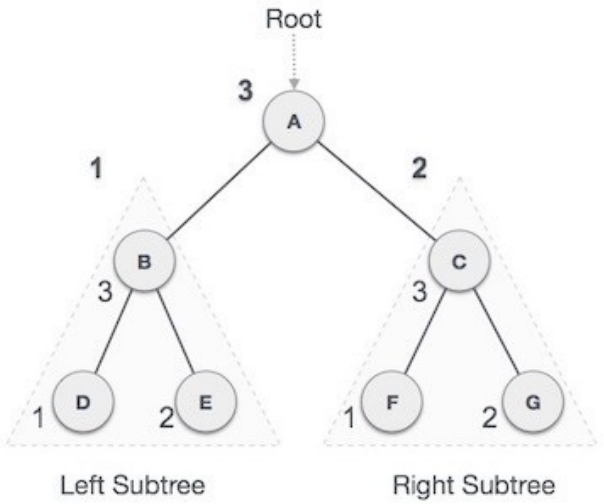


Output: A → B → D → E → C → F → G

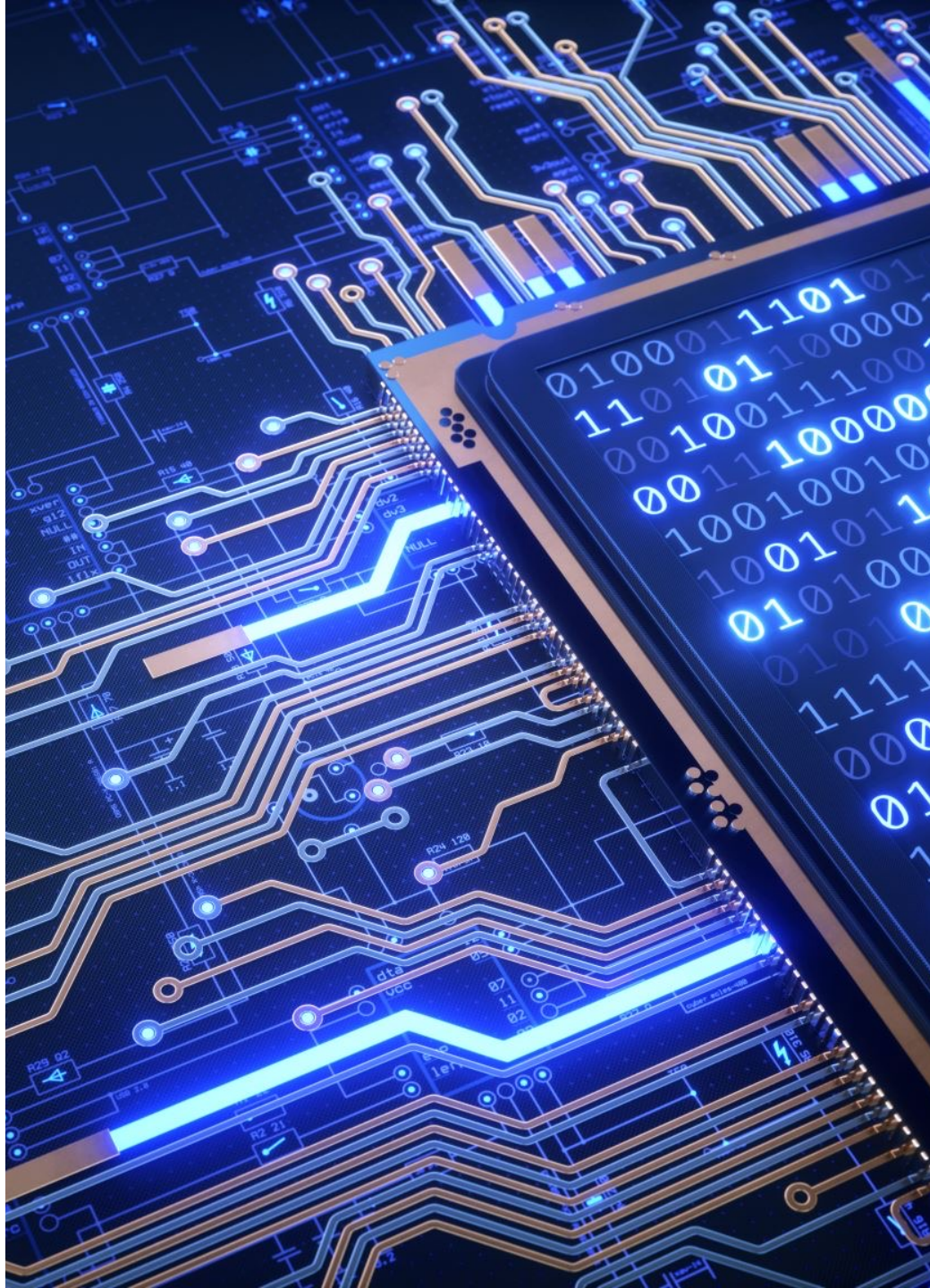


Binary Tree Traversal

- **Post-Order Traversal (Algorithm):**
  - Root node is visited last. First we traverse left sub-tree, then right sub-tree and finally root

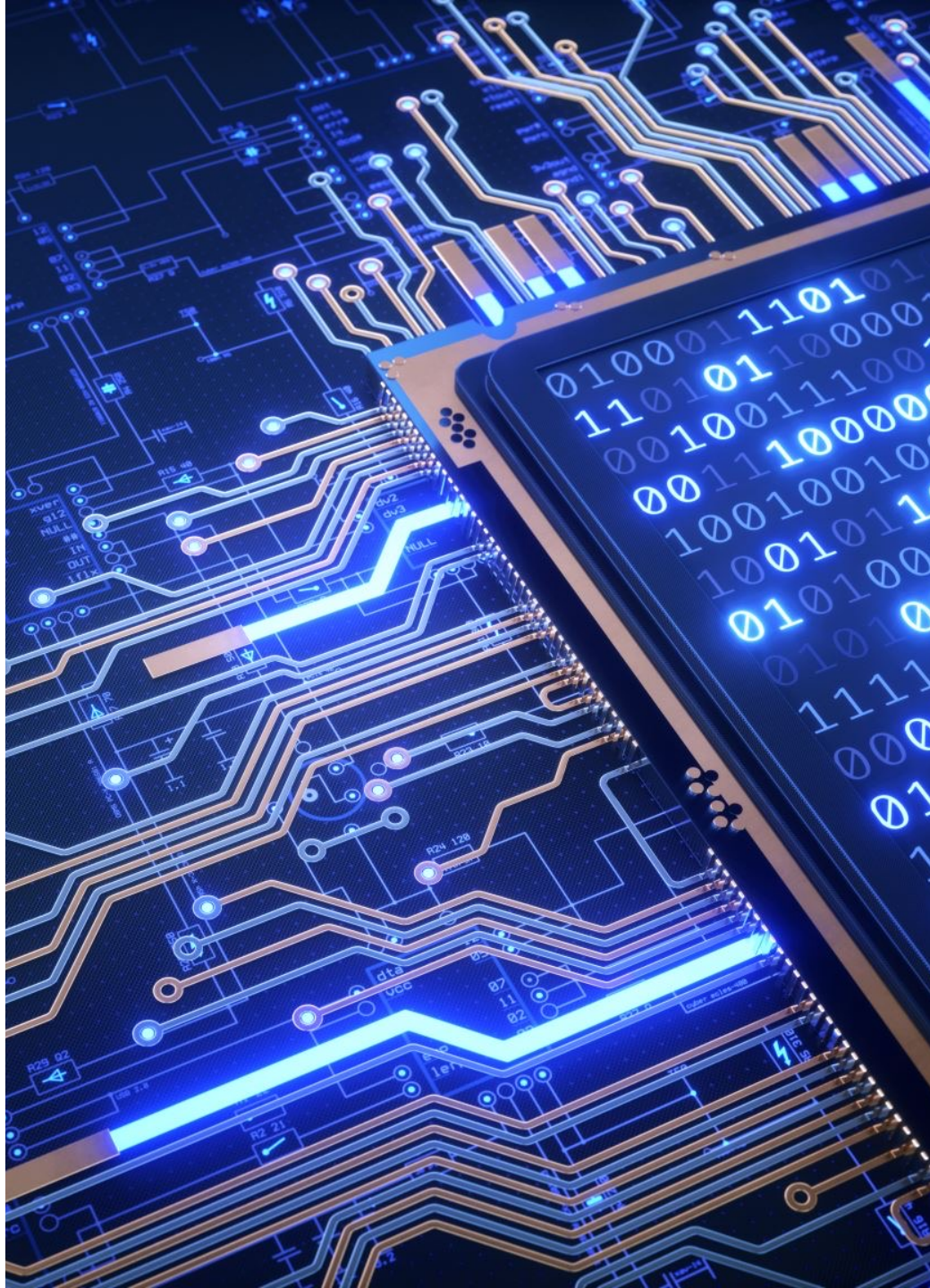


Output: D → E → B → F → G → C → A



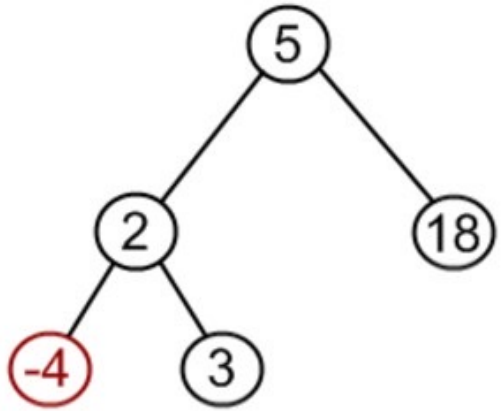
Binary Tree

- **Deleting a node:**
  - Deletion of a node is complicated
  - Two steps:
  1) Search for the given node and identify its parent
  2) Delete the node:
     - Three cases:
     a) Node to be deleted has no children
     b) Node to be deleted has one child
     c) Node to be deleted has two children

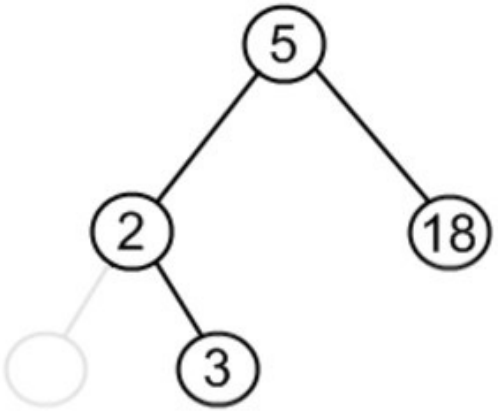


Binary Tree

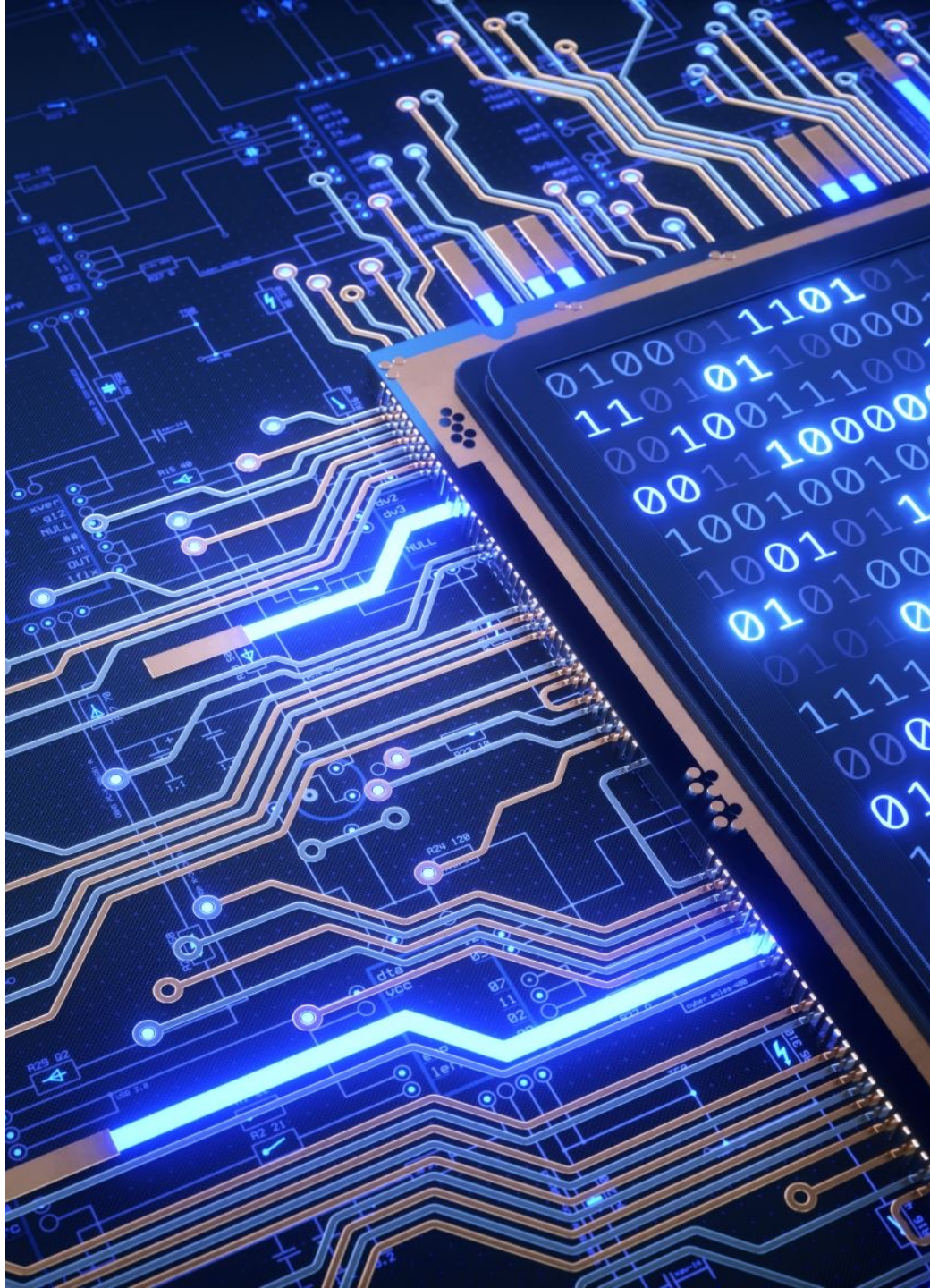
- **Deleting a node:** (a) Node to be deleted has no children
  - Set corresponding link of the parent to NULL



Node to be deleted

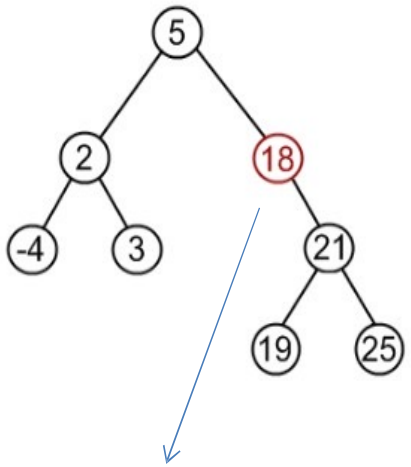


After deletion

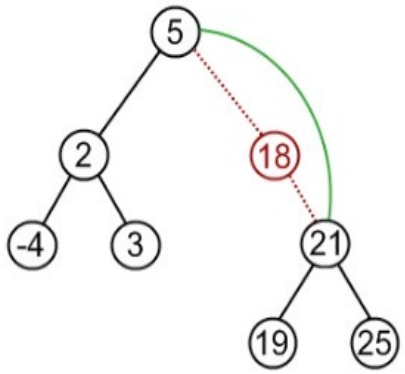


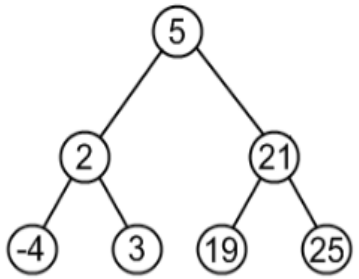
Binary Tree

- **Deleting a node:** (b) Node to be deleted has one child
  - Link the single child (with it's subtree) to the parent of the node to be deleted

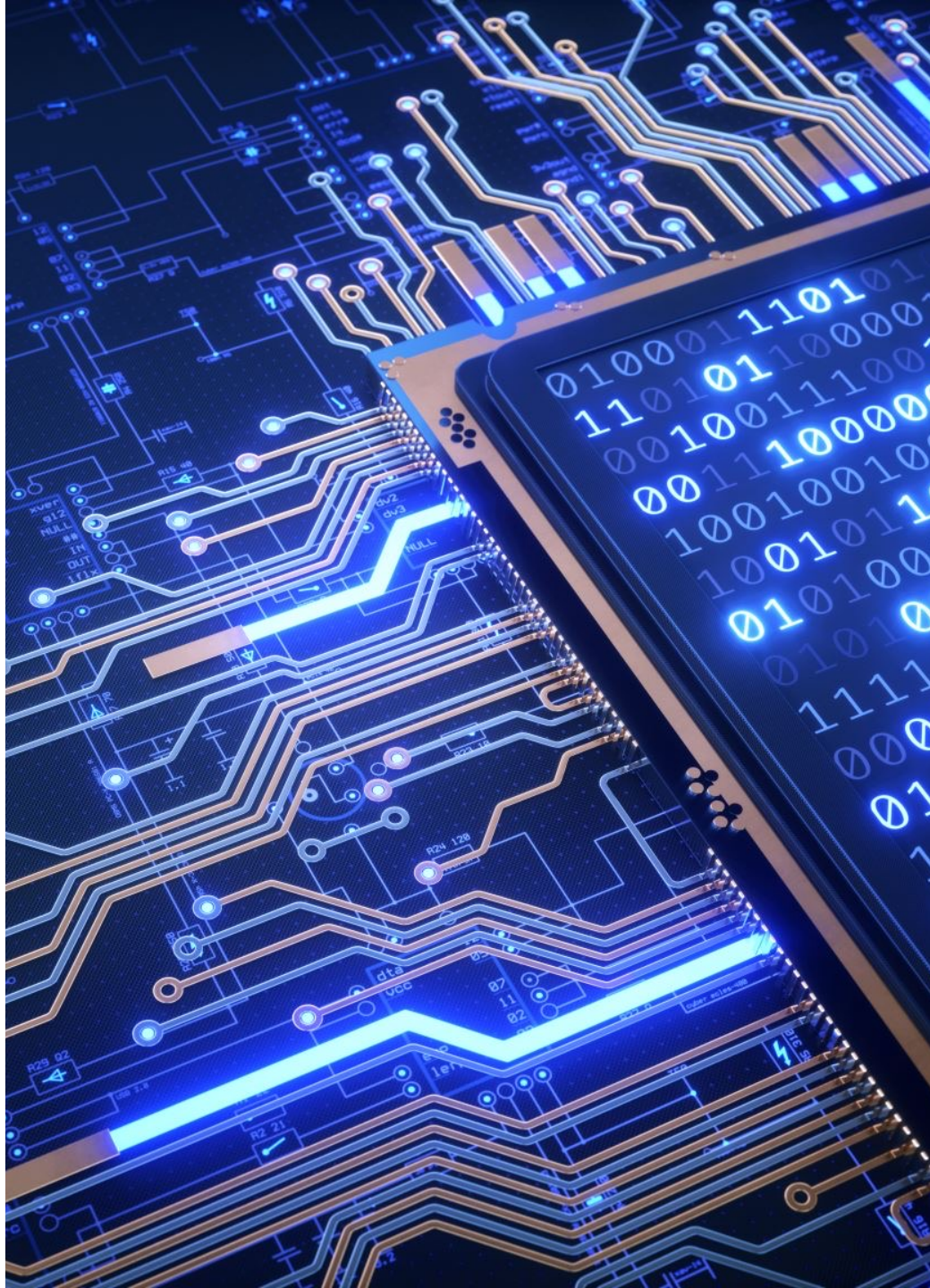


Node to be deleted





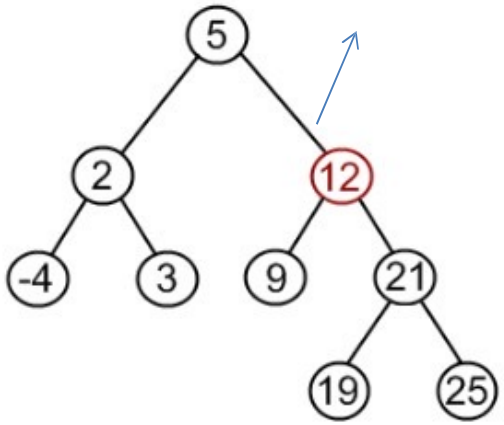
After deletion

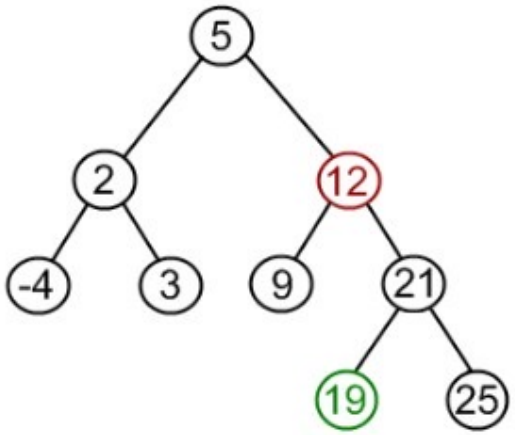


Binary Tree

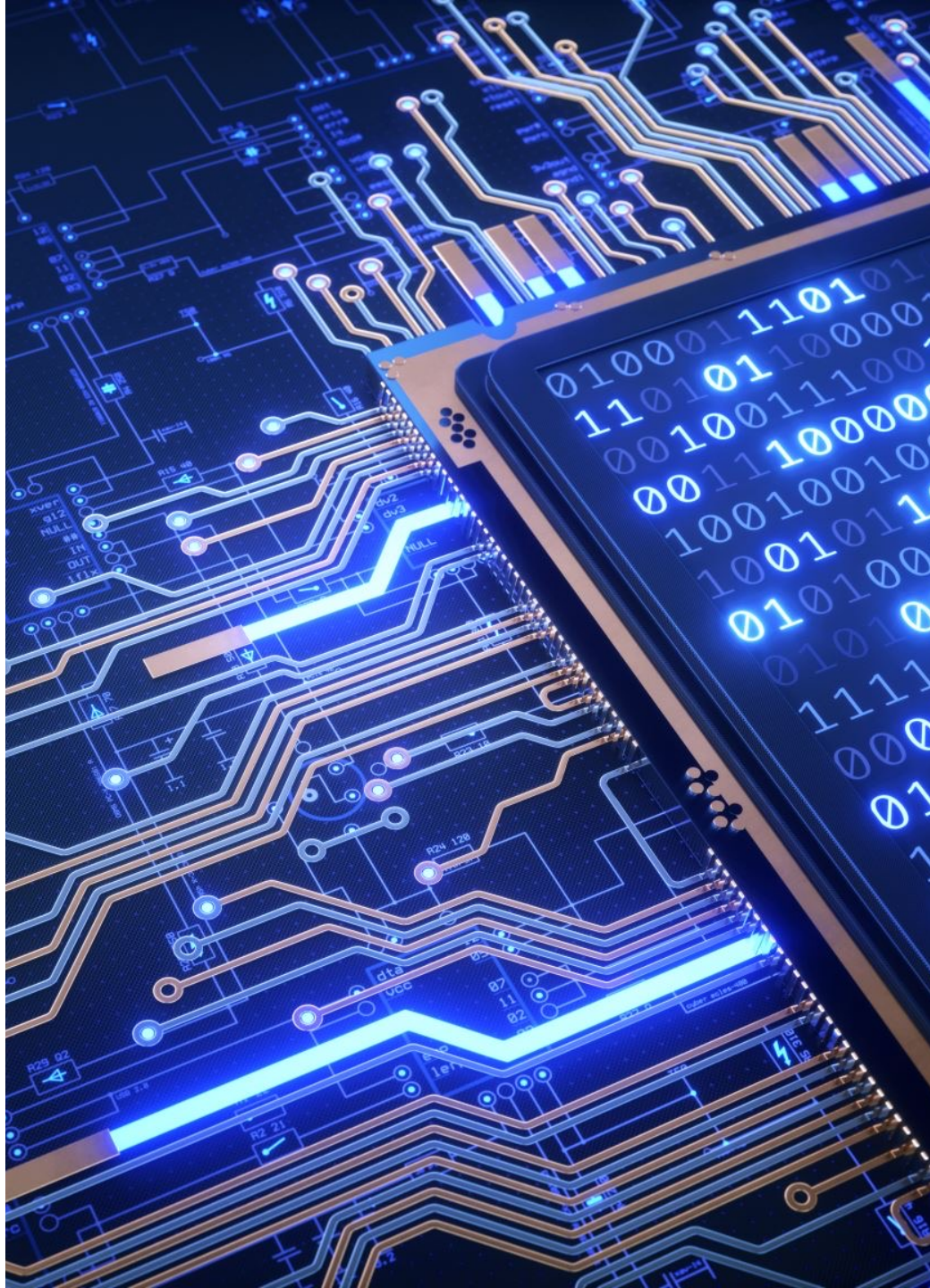
- **Deleting a node:** (c) Node to be deleted has two children
  - find the minimum element in the right subtree, and replace the node to be deleted with it (now, right subtree has a duplicate)
  - apply delete to the right subtree to remove the duplicate.

Node to be deleted with two children



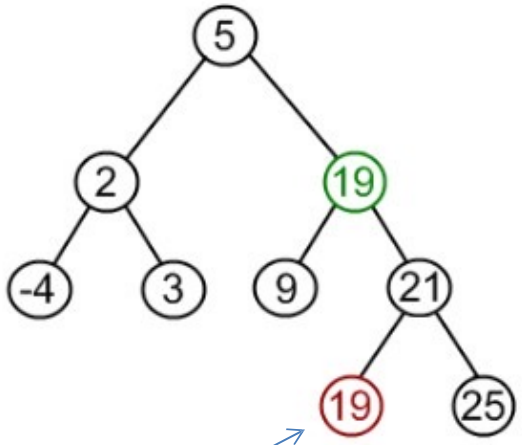


Minimum in the right subtree of 12

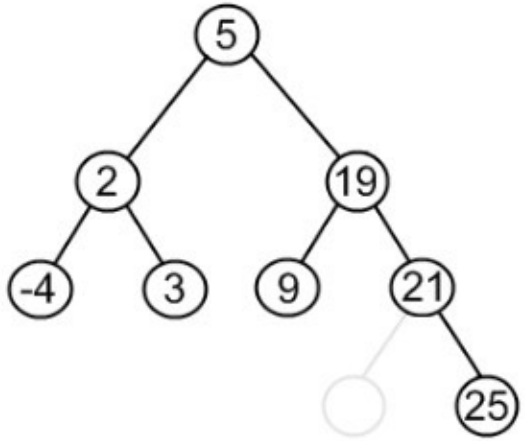


Binary Tree

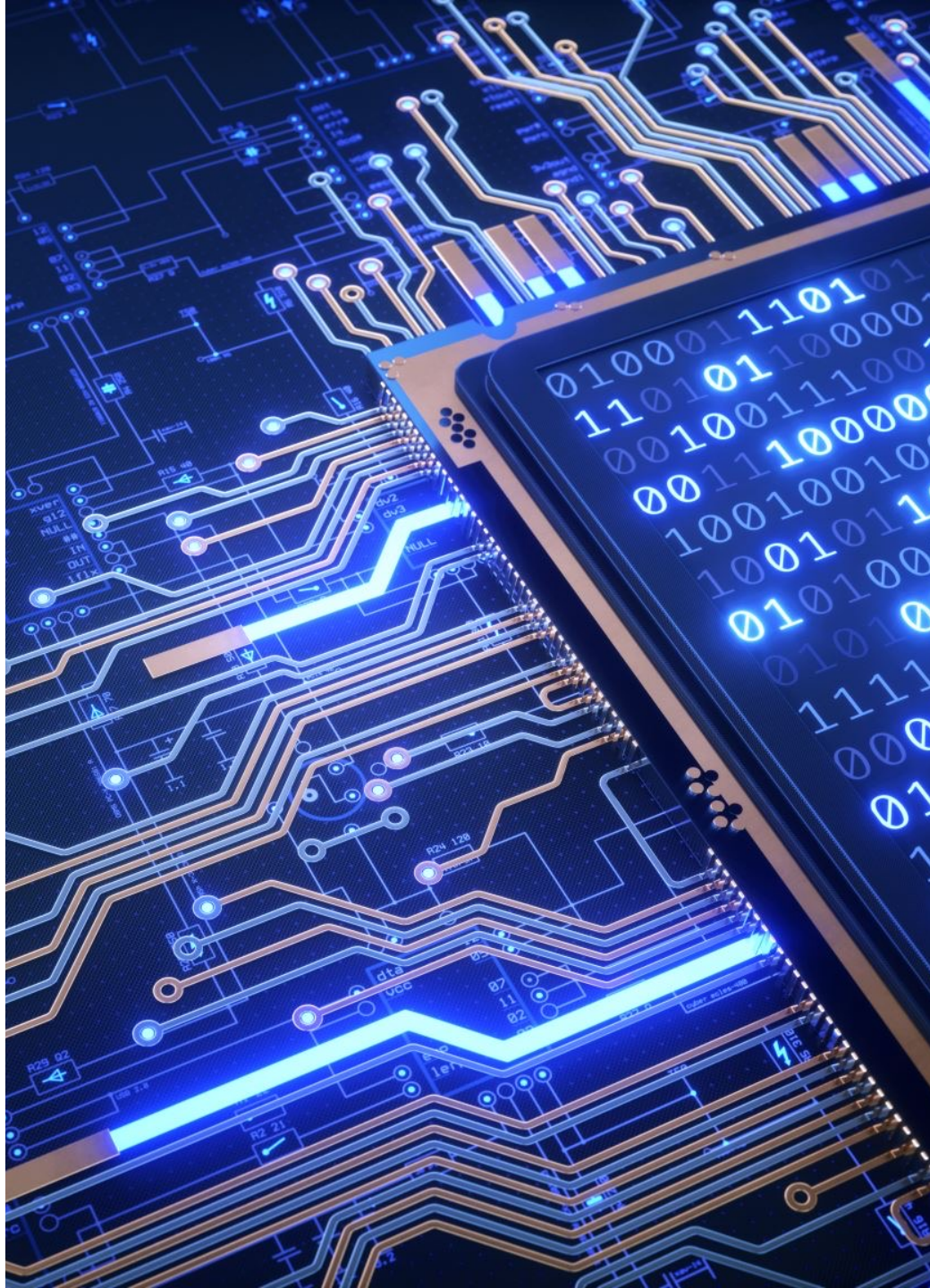
- **Deleting a node:** (c) Node to be deleted has two children
  - find the minimum element in the right subtree, and replace the node to be deleted with it (now, right subtree has a duplicate)
  - apply delete to the right subtree to remove the duplicate.



Duplicate to be removed

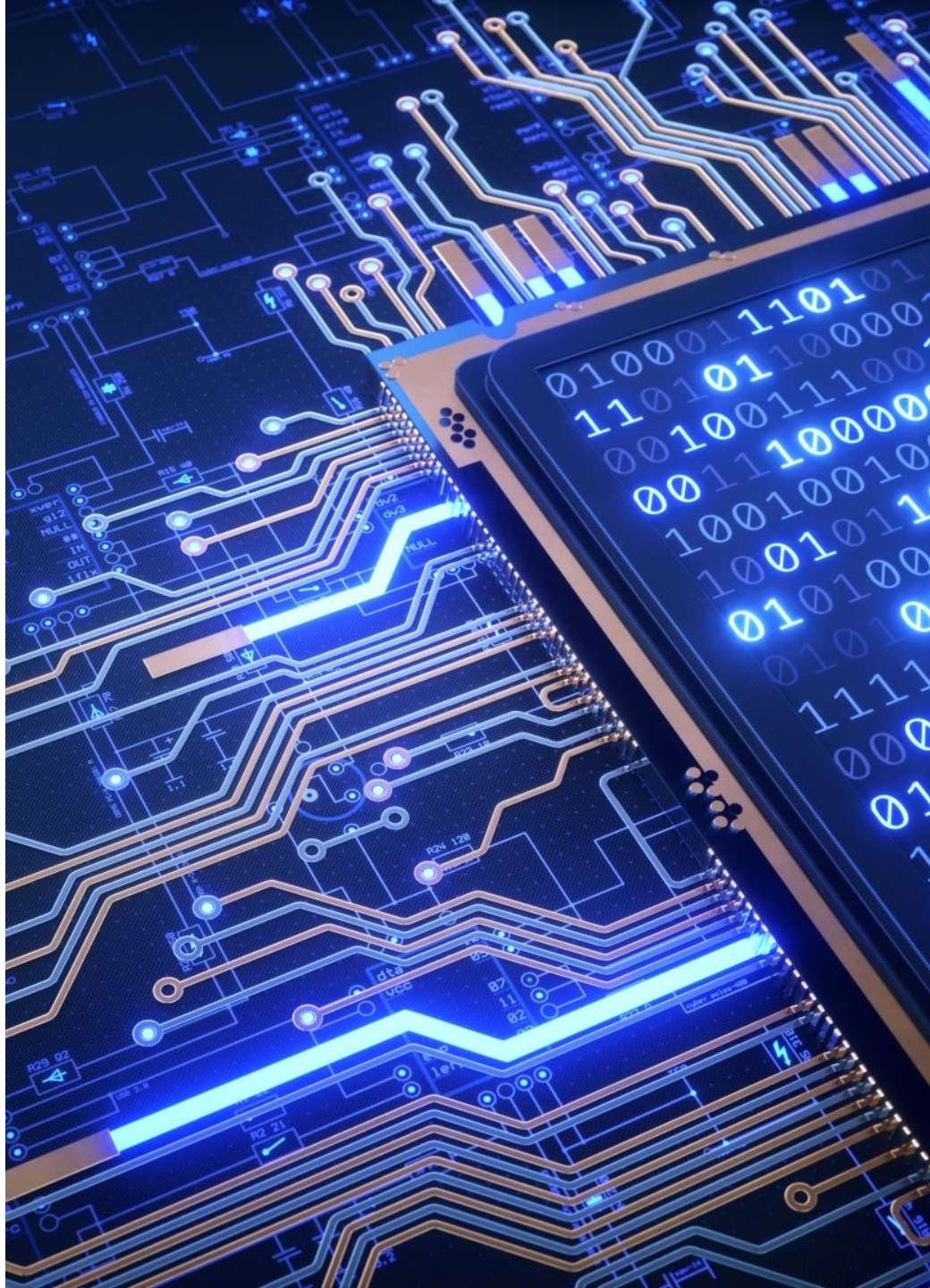


After deletion



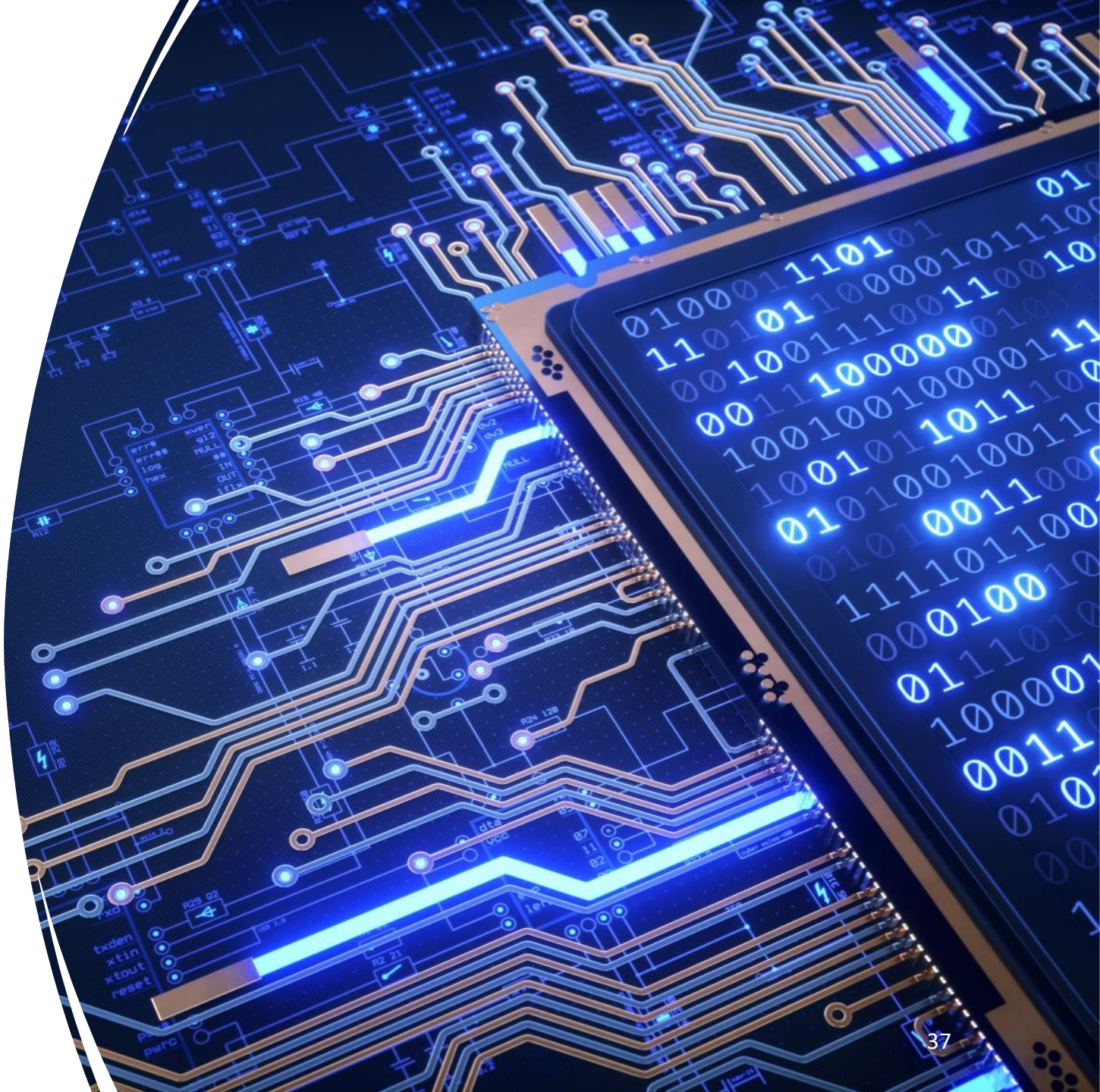
Applications of Trees

- Hierarchical Data Representation
- Searching and Sorting
- Expression Parsing and Evaluation
- Network Routing
- Machine Learning and Artificial Intelligence
- Compilers and Interpreters
- Bioinformatics
- Game Development
- Natural Language Processing



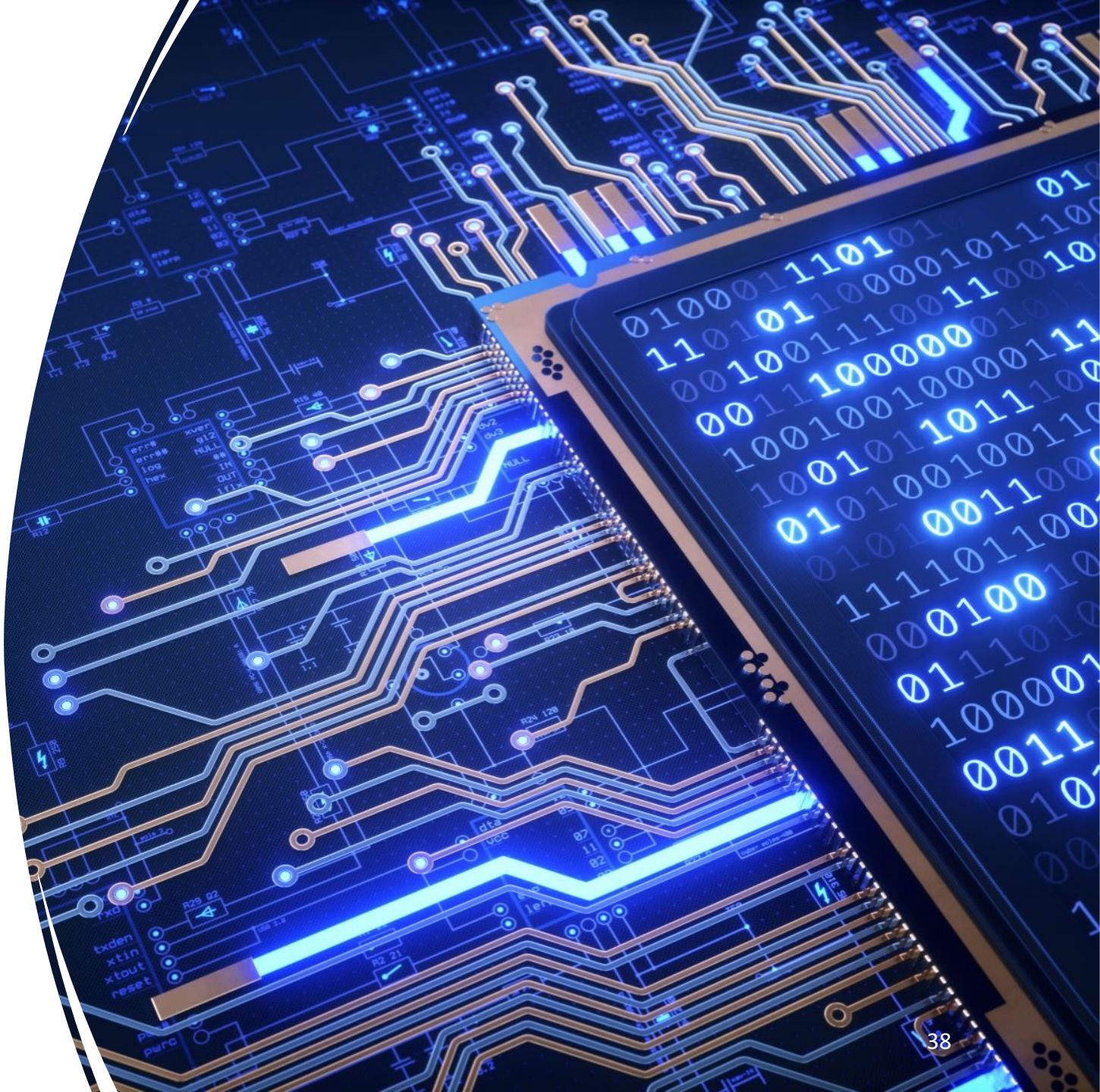
Summary

- Tree is a non-linear data structure
- Definition of a tree data structure
- Tree properties and basic terminologies
- Binary tree
  - Pre-order
  - In-order
  - Post-order
- Applications



Reference

Rosen, K. H. (2012). *Discrete mathematics and its applications (7th Edition)*. McGraw-Hill.
Chapter 11



See you next time!