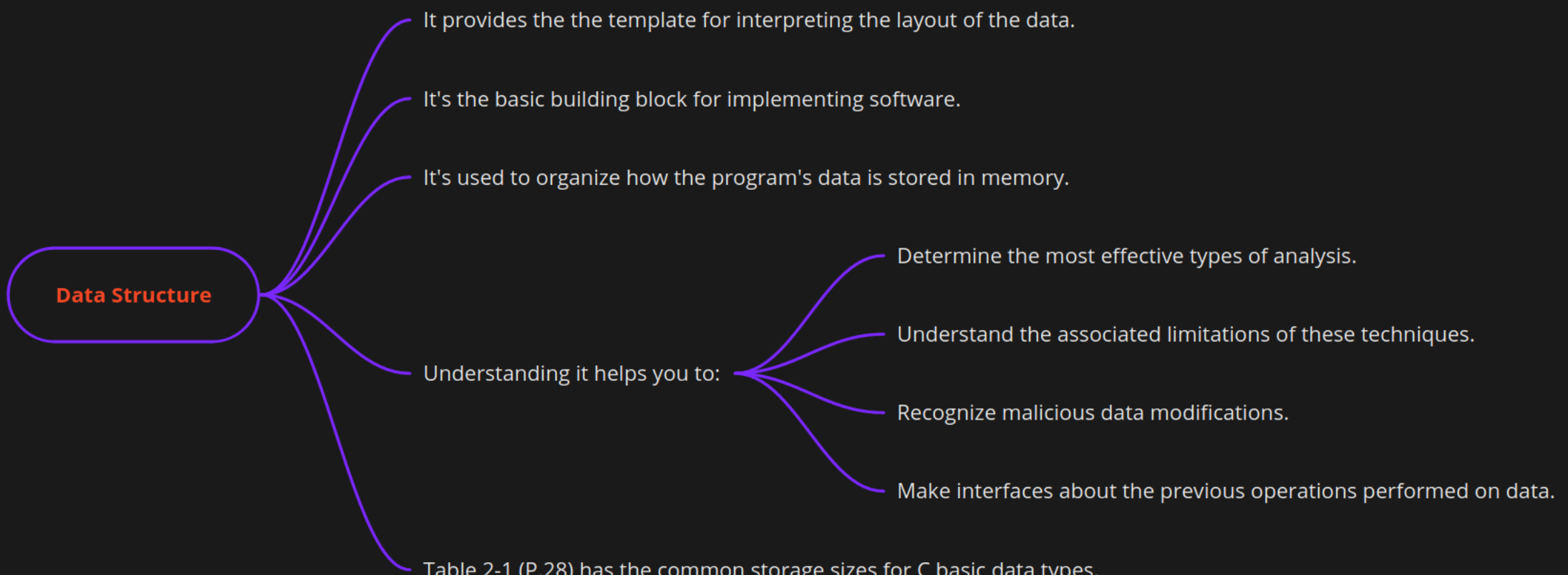
The art of Memory Forensics

Ch.2 - Data Structure

0x1411

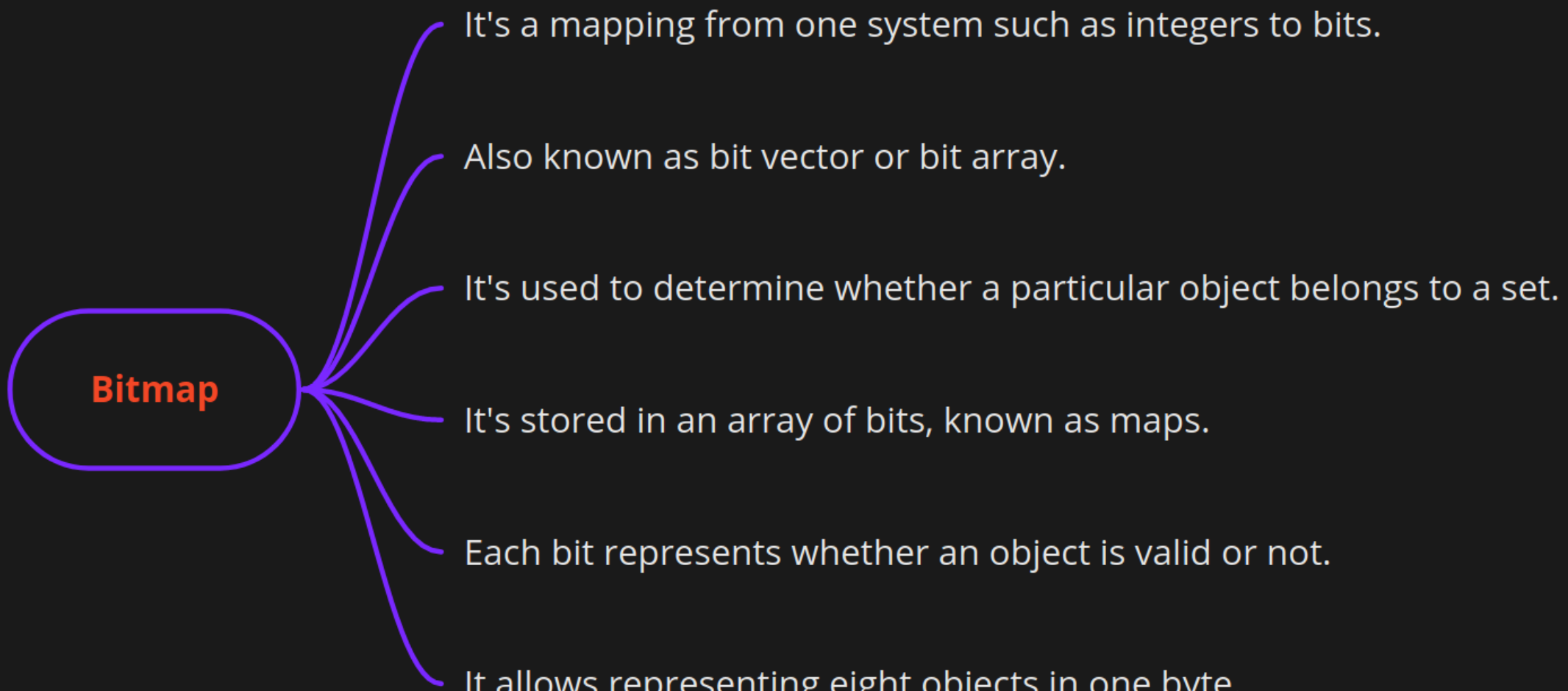

Data Structure
It provides the the template for interpreting the layout of the data.
It's the basic building block for implementing software.
It's used to organize how the program's data is stored in memory.
Understanding it helps you to:
Determine the most effective types of analysis.
Understand the associated limitations of these techniques.
Recognize malicious data modifications.
Make interfaces about the previous operations performed on data.
Table 2-1 (P.28) has the common storage sizes for C basic data types.

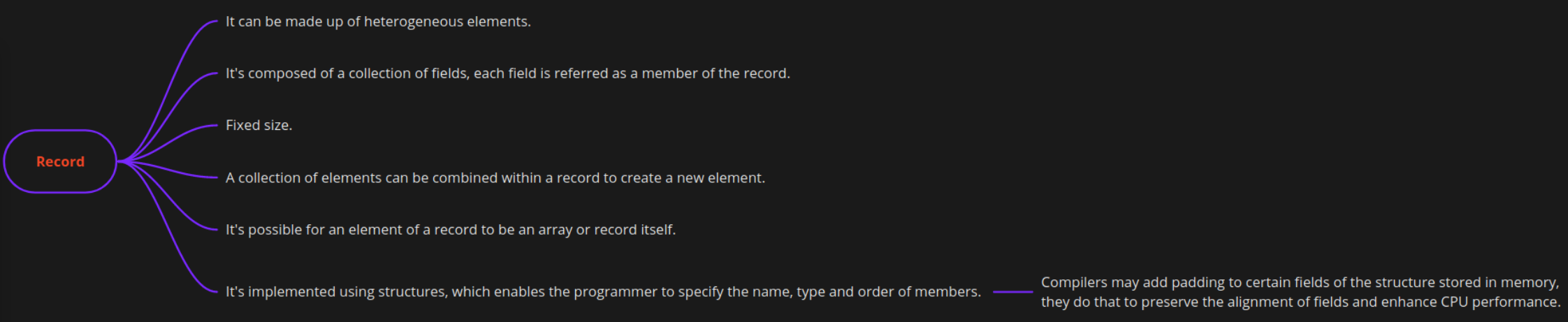
Abstract Data Types

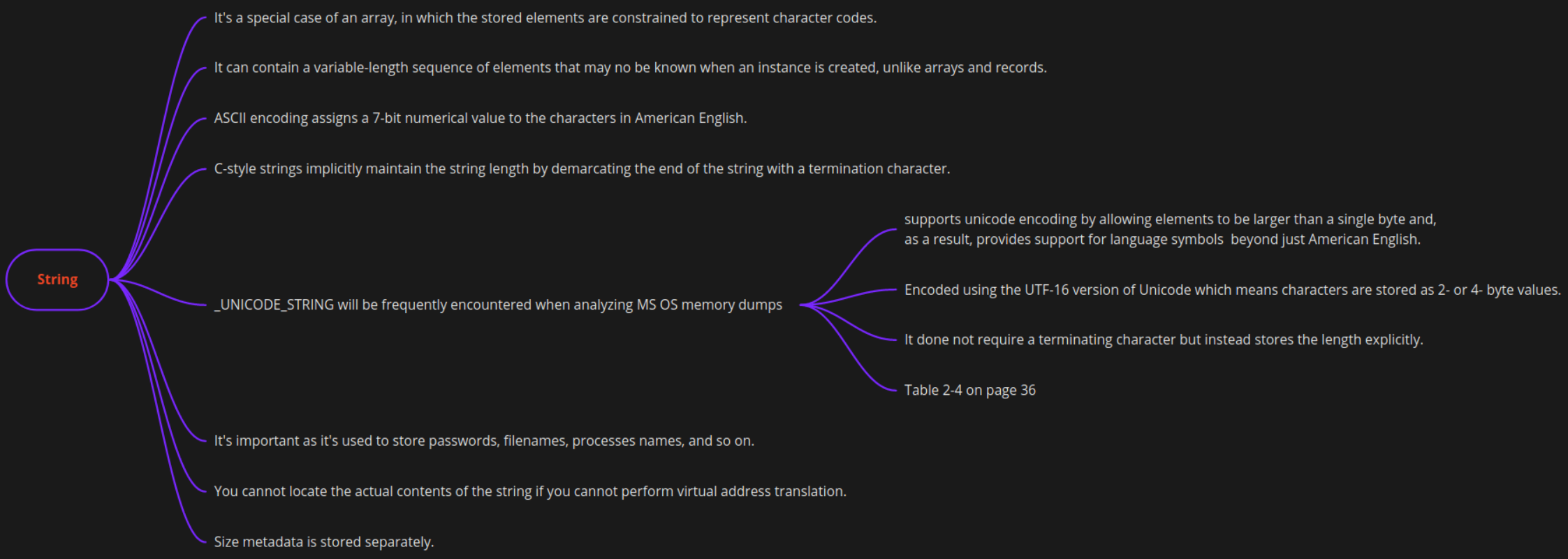
They provide models for both the data and the operations performed on the data.

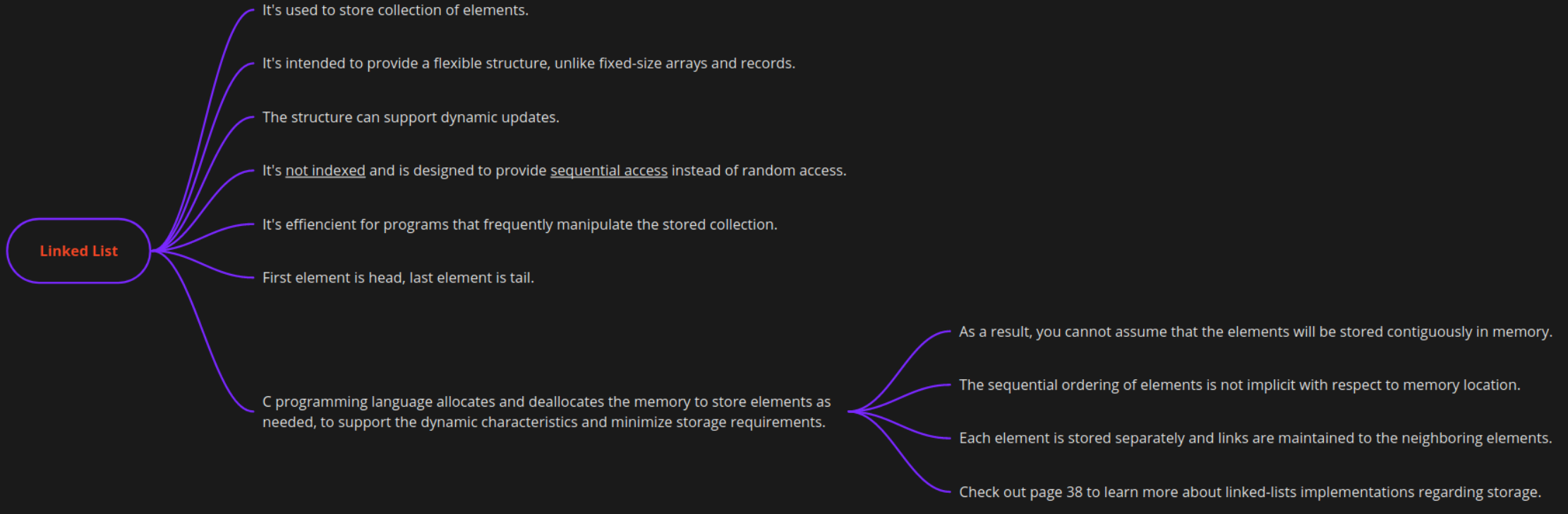
Array

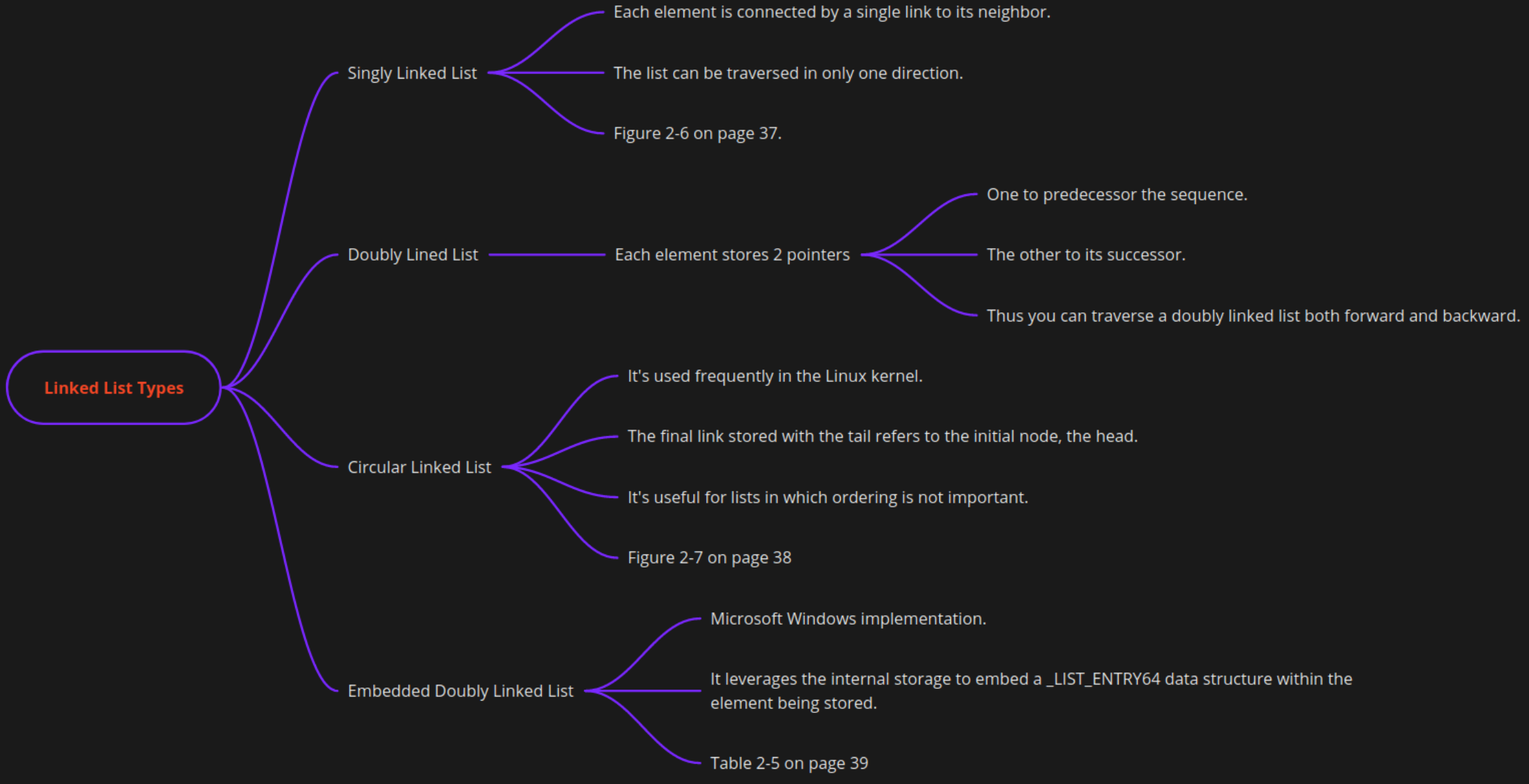
- Its elements are homogeneous data type.
- Array type: data type of the stored elements.
- Fixed size.
- You can access elements by specifying their index that maps to each element's position.
- Starting at index 0.
- The compiler allocates a contiguous block of memory for storing the elements of the array.
- Base address: the first element address in memory which can be used to reference the array location.
- Address(I) = X + (I*S) is used to calculate the address of elements at index I
  - X : base address
  - S: element size
- The previous characteristic is referred to as random access because the time to access an element does not depend on what element you are accessing.

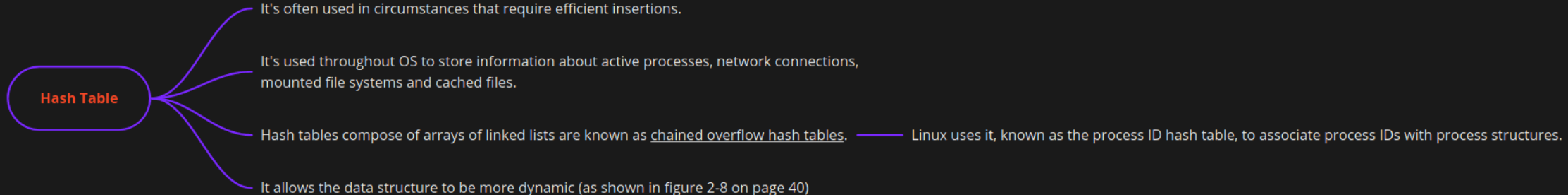

Bitmap
It's a mapping from one system such as integers to bits.
Also known as bit vector or bit array.
It's used to determine whether a particular object belongs to a set.
It's stored in an array of bits, known as maps.
Each bit represents whether an object is valid or not.
It allows representing eight objects in one byte.

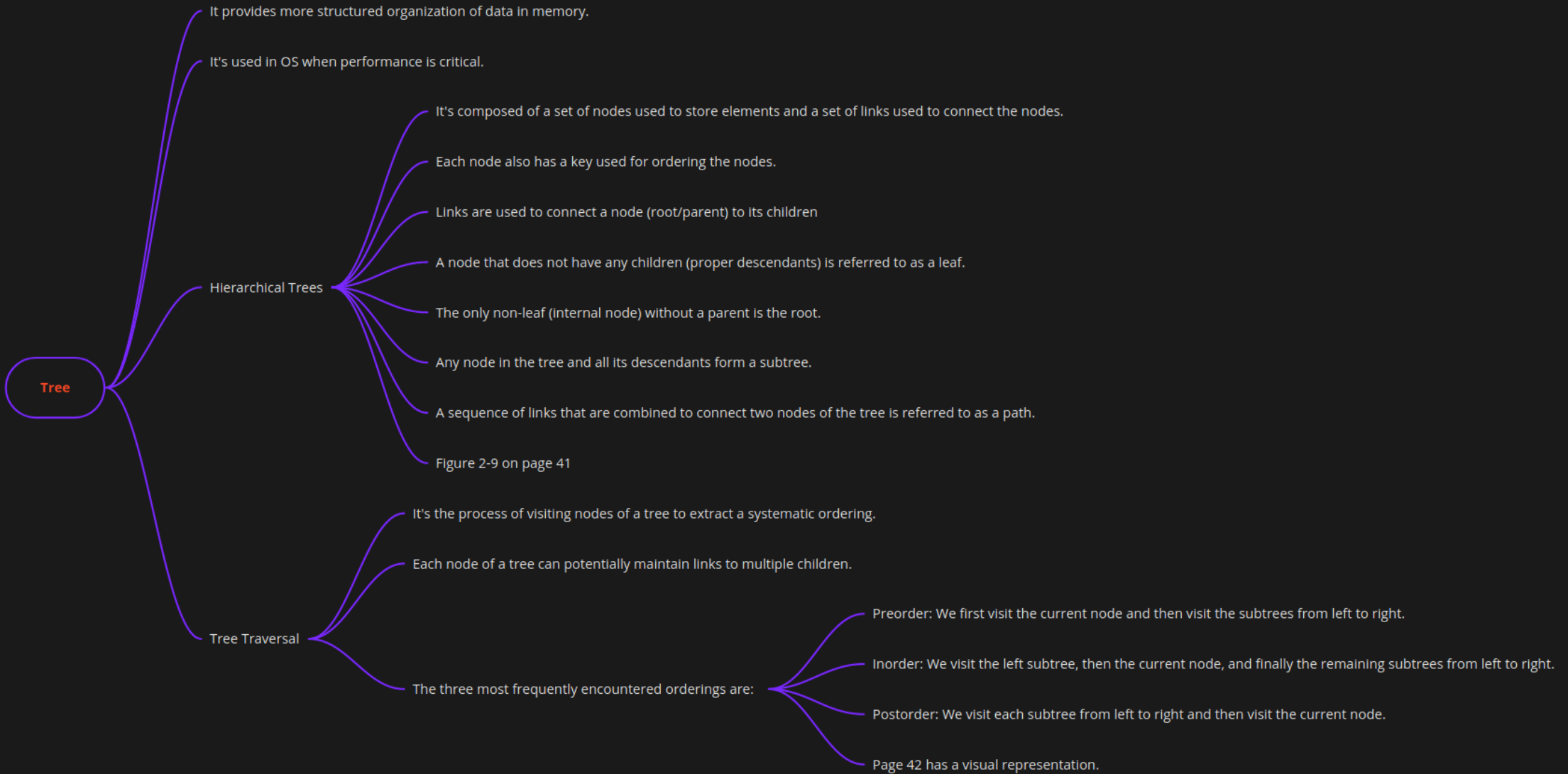

Record
It can be made up of heterogeneous elements.
It's composed of a collection of fields, each field is referred as a member of the record.
Fixed size.
A collection of elements can be combined within a record to create a new element.
It's possible for an element of a record to be an array or record itself.
It's implemented using structures, which enables the programmer to specify the name, type and order of members.
Compilers may add padding to certain fields of the structure stored in memory, they do that to preserve the alignment of fields and enhance CPU performance.


String
It's a special case of an array, in which the stored elements are constrained to represent character codes.
It can contain a variable-length sequence of elements that may no be known when an instance is created, unlike arrays and records.
ASCII encoding assigns a 7-bit numerical value to the characters in American English.
C-style strings implicitly maintain the string length by demarcating the end of the string with a termination character.
_UNICODE_STRING will be frequently encountered when analyzing MS OS memory dumps
supports unicode encoding by allowing elements to be larger than a single byte and, as a result, provides support for language symbols beyond just American English.
Encoded using the UTF-16 version of Unicode which means characters are stored as 2- or 4- byte values.
It done not require a terminating character but instead stores the length explicitly.
Table 2-4 on page 36
It's important as it's used to store passwords, filenames, processes names, and so on.
You cannot locate the actual contents of the string if you cannot perform virtual address translation.
Size metadata is stored separately.


Linked List
It's used to store collection of elements.
It's intended to provide a flexible structure, unlike fixed-size arrays and records.
The structure can support dynamic updates.
It's not indexed and is designed to provide sequential access instead of random access.
It's effiencient for programs that frequently manipulate the stored collection.
First element is head, last element is tail.
C programming language allocates and deallocates the memory to store elements as needed, to support the dynamic characteristics and minimize storage requirements.
As a result, you cannot assume that the elements will be stored contiguously in memory.
The sequential ordering of elements is not implicit with respect to memory location.
Each element is stored separately and links are maintained to the neighboring elements.
Check out page 38 to learn more about linked-lists implementations regarding storage.


Linked List Types
Singly Linked List
Each element is connected by a single link to its neighbor.
The list can be traversed in only one direction.
Figure 2-6 on page 37.
Doubly Lined List
Each element stores 2 pointers
One to predecessor the sequence.
The other to its successor.
Thus you can traverse a doubly linked list both forward and backward.
Circular Linked List
It's used frequently in the Linux kernel.
The final link stored with the tail refers to the initial node, the head.
It's useful for lists in which ordering is not important.
Figure 2-7 on page 38
Embedded Doubly Linked List
Microsoft Windows implementation.
It leverages the internal storage to embed a _LIST_ENTRY64 data structure within the element being stored.
Table 2-5 on page 39


Hash Table
It's often used in circumstances that require efficient insertions.
It's used throughout OS to store information about active processes, network connections, mounted file systems and cached files.
Hash tables compose of arrays of linked lists are known as chained overflow hash tables.
Linux uses it, known as the process ID hash table, to associate process IDs with process structures.
It allows the data structure to be more dynamic (as shown in figure 2-8 on page 40)


Tree
It provides more structured organization of data in memory.
It's used in OS when performance is critical.
Hierarchical Trees
It's composed of a set of nodes used to store elements and a set of links used to connect the nodes.
Each node also has a key used for ordering the nodes.
Links are used to connect a node (root/parent) to its children
A node that does not have any children (proper descendants) is referred to as a leaf.
The only non-leaf (internal node) without a parent is the root.
Any node in the tree and all its descendants form a subtree.
A sequence of links that are combined to connect two nodes of the tree is referred to as a path.
Figure 2-9 on page 41
Tree Traversal
It's the process of visiting nodes of a tree to extract a systematic ordering.
Each node of a tree can potentially maintain links to multiple children.
The three most frequently encountered orderings are:
Preorder: We first visit the current node and then visit the subtrees from left to right.
Inorder: We visit the left subtree, then the current node, and finally the remaining subtrees from left to right.
Postorder: We visit each subtree from left to right and then visit the current node.
Page 42 has a visual representation.