A stack is efficient for temporary storage primarily because its design is a highly constrained, single-access point data structure. This constraint, while limiting its functionality, is exactly what makes its core operations—pushing and popping elements—incredibly fast and simple.

Here are the key reasons for its efficiency:

LIFO (Last-In, First-Out) Principle: The stack's behavior is dictated by this simple rule. There is no need for complex logic to decide where to insert or remove an element. The new item always goes on top, and the item to be removed is always the one on top. This direct, unambiguous process makes operations instantaneous.

Single-Point of Access: All reads and writes occur at only one location: the "top" of the stack. Unlike other data structures that may require traversal or searching through a list or array, a stack's pointer always points directly to where the next operation will take place. This eliminates the need for any searching or indexing, resulting in an O(1) time complexity for both push and pop operations.

Efficient Memory Management: A stack typically uses a contiguous block of memory. When a new item is pushed, the stack simply extends into the next available memory slot. When an item is popped, the memory is freed from the top. This sequential allocation and deallocation is extremely fast and avoids the fragmentation overhead that can occur in other, more dynamic memory structures.

In essence, a stack's efficiency for temporary storage comes from its lack of flexibility. Its strict, linear nature allows it to perform its limited set of operations with maximum speed and minimum computational overhead, making it perfect for tasks like function call management, expression evaluation, and undo/redo functionality where the sequence of events is critical and the most recent data is the most relevant.