As I understood from the provided link, there are situations that for some reason, after creating a container including some elements, we are interested to save e.g. a pointer (or reference, or iterator) to some elements for later use. Then we modify or change somehow the container's content, like adding or erasing some elements. Then if we try to reuse that pointer (or reference, iterator), it is not valid anymore and it does not show the same value and the existing elements might move to a totally different location.   
According to the reference link, different containers may behave differently in regarding invalidation and I think the reason is that different functions may change the size of the container. In some containers, a function it may invalidate all the pointers, references and iterators, and in another container it may invalidate only the pointer, reference or iterator related to the referred element in the function.

Although, the C++ standard allows that pointers and references to behave independently for the same function, but it is risky to rely on this kind of distinction between pointer and reference. For example it may happen that a pointer is still valid, but an element that is pointed by the pointer may be it is not the same as the element that we really meant.