```
// ...

enum { k_CACHE_LINE_SIZE = 64 };  // A cache line on this platform is 64 bytes.

void f()
{
    alignas(k_CACHE_LINE_SIZE) int i0 = 0; // i1 and i2 are on separate
    alignas(k_CACHE_LINE_SIZE) int i1 = 0; // cache lines.

    // ...
}
```

As an empirical demonstration of the effects of false sharing, a benchmark program repeatedly calling `f` completed its execution seven times faster on average when compared to the same program without use of **`alignas`**.[5] Note that because supported extended alignments are implementation defined, using **`alignas`** is not a strictly portable solution. Opting for less elegant and more wasteful padding approach instead of **`alignas`** might be preferrable for portability.

Avoiding false sharing within a single-thread-aware object

A real-world scenario where the need for preventing false sharing is fundamental occurs in the implementation of high-performance concurrent data structures. As an example, a thread-safe ring buffer might make use of **`alignas`** to ensure that the indices of the head and tail of the buffer are aligned at the start of a cache line (typically 64, 128, or 256 bytes),[6] thereby preventing them from occupying the same one.

```
#include <atomic>  // std::atomic
class ThreadSafeRingBuffer
{
    alignas(k_CACHE_LINE_SIZE) std::atomic<std::size_t> d_head;
    alignas(k_CACHE_LINE_SIZE) std::atomic<std::size_t> d_tail;

    // ...
};
```

Not aligning `d_head` and `d_tail` in the code snippet above to the CPU cache size might result in poor performance of the `ThreadSafeRingBuffer` because CPU cores that need to access only one of the variables will inadvertently load the other one as well, triggering expensive hardware-level coherency mechanisms between the cores' caches. On the other hand,

[5] The benchmark program was compiled using Clang 11.0.0 (c. 2020) using `-Ofast`, `-march=native`, and `-std=c++11`. The program was then executed on a machine running Windows 10 x64, equipped with an Intel Core i7-9700k CPU (8 cores, 64-byte cache line size). Over the course of multiple runs, the version of the benchmark without **`alignas`** took an average of 18.5967ms to complete, while the version with **`alignas`** took an average of 2.45333ms to complete.

[6] In C++17, one can portably retrieve the minimum offset between two objects to avoid false sharing through the `std::hardware_destructive_interference_size` constant defined in the **`<new>`** header.