```
  struct S
  {
    long a;
    char b;
   };

   //update a
   auto f = [&s]()
   {
      s.a = 10;
   };

   //upddate b
   auto g = [&s]()
   {
      s.b = 20;
   };

   //sample how to exploit memory model objects layout
   auto fut1 = std::async(std::launch::async,f);
   auto fut2 = std::async(std::launch::async,g);

   fut1.wait();
   fut2.wait();
```

only C++11

is this code thread safe??

only C++11

```
struct S
{
  long a;
  char b;
 };

//update a
auto f = [&s]()
{
   s.a = 10;
};

//upddate b
auto g = [&s]()
{
   s.b = 20;
};

//sample how to exploit memory model objects layout
auto fut1 = std::async(std::launch::async,f);
auto fut2 = std::async(std::launch::async,g);

fut1.wait();
fut2.wait();
```

std::atomic

```
template< class T > struct atomic
template<> struct atomic<Integral>;
template< class T > struct atomic<T*>;
```

- Each instantiation and full specialization of the std::atomic template defines an atomic type.
- Objects of atomic types are the only C++ objects that are free from data races; that is
- if one thread writes to an atomic object while another thread reads from it, the behaviour is well-defined.

from http://en.cppreference.com/w/cpp