# std::variant

- **std::variant** - a polymorphic type containing one of a fixed set of types
  - Safe replacement of C unions
  - plus a data element to remember which type it is

```
using VT = std::variant< T0, T1, T2>;
```

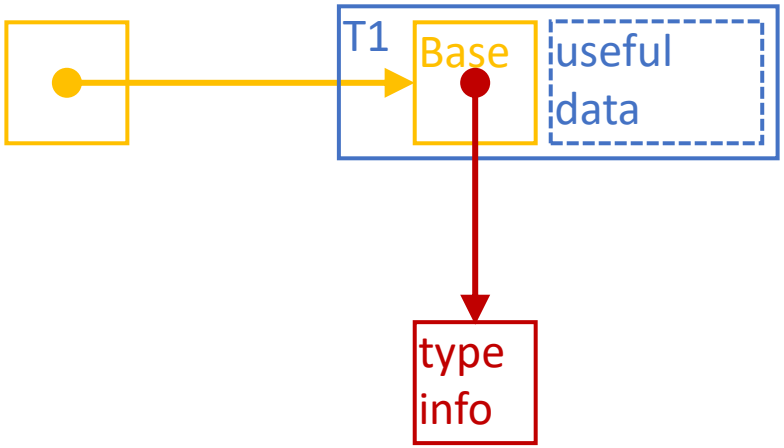
  - More space-effective than a pointer to a dynamically-allocated object with inheritance
    - If the types are of similar sizes
    - But not extensible
  - There is no common interface (base class) required in the types
  - Allows assignment from any of the types

```
T0 v0 = /*...*/;
VT a = v0;
VT b(std::in_place_type<T1>, /*...*/);   // calls T1::T1(/*...*/)
VT c(std::in_place_index<2>, /*...*/);   // calls T2::T2(/*...*/)
b = v0;                  // calls T1::~T1(), T0::T0(v0)
c.emplace<T1>(/*...*/); // calls T2::~T2(), T1::T1(/*...*/)
a.emplace<2>(/*...*/);  // calls T0::~T0(), T2::T2(/*...*/)
```

  - Access to the contained data:

```
void action( VT & vo)
{
  switch ( vo.index() ) {
  case 0: { T0 & v0 = std::get< 0>(vo); v0.f(); } break;
  case 1: { T1 & v1 = std::get< 1>(vo); v1.g(); } break;
  case 2: { T2 & v2 = std::get< 2>(vo); v2.h(); } break;
}
```

## Using inheritance




```
std::unique_ptr<Base> x =
std::make_unique<T1>(/*...*/);
```

- Higher run-time cost
  - Pointer, dynamic allocation
- Extensible set of concrete types

## Using std::variant


```
std::variant<T1,T2,T3> x =
T1(/*...*/);
```

- Lower run-time cost
  - No dynamic allocation
- Always requires maximum space
- Not intrusive
  - Does not use any base class
- Fixed set of alternative types

# std::variant and static visitor

```
using VT = std::variant< T0, T1, T2>;
```

- std::visit - Usage through a polymorphic functor

```
struct VisitorA {
  void operator()( T0 & x) { /*...*/ }
  void operator()( T1 & x) { /*...*/ }
  void operator()( T2 & x) { /*...*/ }
};
void action( VT & vo)
{
  VisitorA va;
  std::visit(va, vo);
}
```

- It may be used with a polymorphic lambda

```
std::visit([](auto && a){ a.something(); }, vo);
```

  - All the types need a common interface
    - The interface is not explicitly declared
  - This is a static equivalent of inheritance and virtual functions
    - At higher cost – visit requires a tricky implementation similar to visitors
    - But the memory footprint may still be smaller – no dynamic allocation