it. Preventing a type from being inheritable closes the gap between what is possible with built-in types, such as **int** and **double**, and what can additionally be done with typical user-defined ones — specifically, inherit from them. See *Use Cases — Suppressing derivation to ensure portability* on page 1014, i.e., **Hyrum's law**.

Although other use cases might be plausible, widespread, systemic use can run afoul of stable **reuse** in general and **hierarchical reuse** in particular; see *Potential Pitfalls — Systemic lost opportunities for reuse* on page 1023. Hence, the decision to use **final** on an entire class even rarely — let alone routinely — is not to be taken lightly.

Prior to C++11's introduction of the **final** specifier, there was no convenient way to ensure that a **user-defined type (UDT)** was *uninheritable*, although Byzantine idioms to approximate this restriction existed. For example, a virtual base class needs to be initialized in each constructor of all concrete derived types, and that can be leveraged to prevent useful inheritance. Consider a trio of classes, the first of which, `UninheritableGuard`, has a private constructor and befriends its only intended derived class; the second, `Uninheritable`, derives privately and virtually from `UninheritableGuard`; and the third, `Inheriting`, is a misguided class that tries in vain to inherit from `Uninheritable`:

```
struct UninheritableGuard  // private, virtual base class
{
private:
    UninheritableGuard();         // private constructor
    friend struct Uninheritable;  // constructible only by Uninheritable
};

struct Uninheritable : private virtual UninheritableGuard
{
    Uninheritable() : UninheritableGuard() { /*...*/ }
};

struct Inheriting : Uninheritable  // Uninheritable is effectively final.
{
    Inheriting()
    : Uninheritable()  // Error, Uninheritable() is inaccessible.
    { /*...*/ }
};
```

Any attempt to define — either implicitly or explicitly — a constructor for `Inheriting` will fail with the same error due to the inaccessibility of the constructor for `UninheritableGuard`. Using **virtual** inheritance typically requires each object of type `Uninheritable` to maintain a virtual table pointer; hence, this solution does not come without overhead. Note also that this workaround prior to **final** does not prevent the derivation itself, but merely the instantiation of any ill-fated derived classes.

In the special case where all of the data members of the type are trivial, i.e., have no user-provided **special member functions** (see Section 2.1."Generalized PODs '11" on page 401), we could have instead created a type, e.g., `Uninheritable2`, that is implemented as a **union** consisting of just a single **struct**: