can be **captured by copy** or **captured by reference**. Orthogonally, each variable can be **explicitly captured** or **implicitly captured**. We'll examine each of these aspects of lambda capture in turn.

Syntactically, the lambda capture consists of an optional **capture default** followed by a comma-separated list of zero or more identifiers (or the keyword **this**), which are explicitly captured. The capture default can be one of `=` or `&` for capture by copy or capture by reference, respectively. If there is a capture default, then **this** and any local variables in scope that are ODR-used within the lambda body and not explicitly captured will be implicitly captured.

```
void f1()
{
    int a = 0, b = 1, c = 2;
    auto c1 = [a, b]{ return a + b; };
        // a and b are explicitly captured.
    auto c2 = [&]{ return a + b; };
        // a and b are implicitly captured.
    auto c3 = [&, b]{ return a + b; };
        // a is implicitly captured, and b is explicitly captured.
    auto c4 = [a]{ return a + b; }
        // Error, b is ODR-used but not captured.
}
```

The Standard defines the **lambda introducer** as the lambda capture together with its surrounding `[` and `]`. If the lambda introducer is an empty pair of brackets, no variables will be captured, and the lambda is stateless:

```
auto c1 = []{ /*...*/ };  // empty lambda capture
```

The lambda capture enables access to portions of the local **stack frame**. As such, only variables with **automatic storage duration** — i.e., nonstatic local variables — can be captured, as we'll see in detail later in this section and the lambda body section. An explicitly captured variable whose name is immediately preceded by an `&` symbol in the lambda capture is captured by reference; without the `&`, it is captured by copy. If the capture default is `&`, then all implicitly captured variables are captured by reference. Otherwise, if the capture default is `=`, all implicitly captured variables are captured by copy:

```
void f2()
{
    int a = 0, b = 1;
    auto c1 = [&a]{ /*...*/ return a; };  // a captured by reference
    auto c2 = [a] { /*...*/ return a; };  // a captured by copy
    auto c3 = [a, &b] { return a + b; };
        // a is explicitly captured by copy, and b is explicitly
        // captured by reference.
    auto c4 = [=]{ return a + b; };
        // a and b are implicitly captured by copy.
    auto c5 = [&]{ return &a; };
        // a is implicitly captured by reference.
```