Operating

The newer and more robust way of processing your chains is through the process called operating.

C++ Workflow

Using a Lambda

You can easily operate on your chain via a C++ lambda and this is how you do it:

```
Chain->Operate([](const FChain::FCursor& Cursor, FMyTrait Trait)
{
    ...
});
```

The type of cursor here must match the type of the chain used. Note that you're not allowed to acquire a reference to the trait while processing a non-solid chain, only its copy. So in order to operate on a solid chain, you could do something like this:

```
SolidChain->Operate([](const FSolidChain::FCursor& Cursor, FMyTrait& Trait)
{
    ...
});
```

Now you can change the properties (fields) of the trait directly, without copying involved.

Concurrency

Solid Chains also support a special type of operating - a multi-threaded one. The function to call is explicitly named with a `Concurrently` prefix and accepts two more arguments: the maximum number of tasks to utilize and the minimum number of slots per each such task. For example:

```
SolidChain->OperateConcurrently([](const FSolidChain::FCursor& Cursor,
FMyTrait& Trait)
{
    ...
}, 4, 32);
```

The second parameter helps to also limit the number of tasks. If there are too little slots available,

excessive tasks won’t be launched.