With C++11 delegating constructors, the constructor accepting a string can be rewritten to delegate to the one accepting `address` and `port`, avoiding repetition without having to use a private function:

```
#include <cstdint>  // std::uint16_t, std::uint32_t
#include <string>   // std::string

class IPV4Host
{
     // ...
private:
    int connect(std::uint32_t address, std::uint16_t port);

public:
    IPV4Host(std::uint32_t address, std::uint16_t port)
    {
        if(!connect(address, port))
        {
            throw ConnectionException{address, port};
        }
    }

    IPV4Host(const std::string& ip)
        : IPV4Host{extractAddress(ip), extractPort(ip)}
    {
    }
};
```

Using delegating constructors results in less boilerplate and fewer runtime operations, as data members and base classes can be initialized directly through the member initializer list.

Potential Pitfalls

Delegation cycles

If a constructor delegates to itself either directly or indirectly, the program is **ill formed, no diagnostic required (IFNDR)**. While some compilers can, under certain conditions, detect delegation cycles at compile time, they are neither required nor necessarily able to do so. For example, even the simplest delegation cycles might not result in a diagnostic from a compiler[2]:

[2] As of this writing, GCC 11.2 (c. 2021) does not detect this delegation cycle at compile time and produces a binary that, if run, will necessarily exhibit **undefined behavior**. Clang 3.0 (c. 2011) and later, on the other hand, halts compilation with a helpful error message:

```
error: constructor for S creates a delegation cycle
```