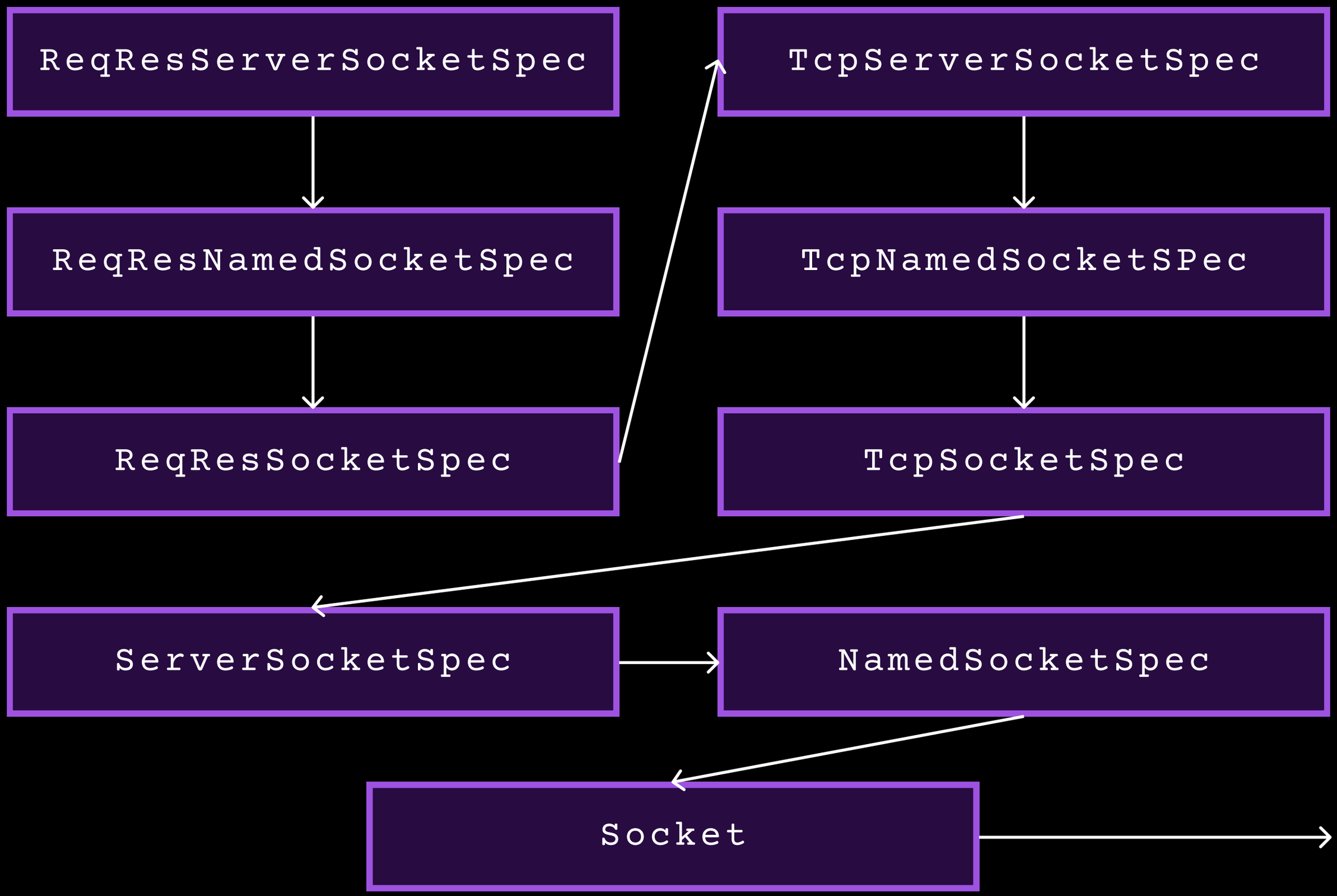
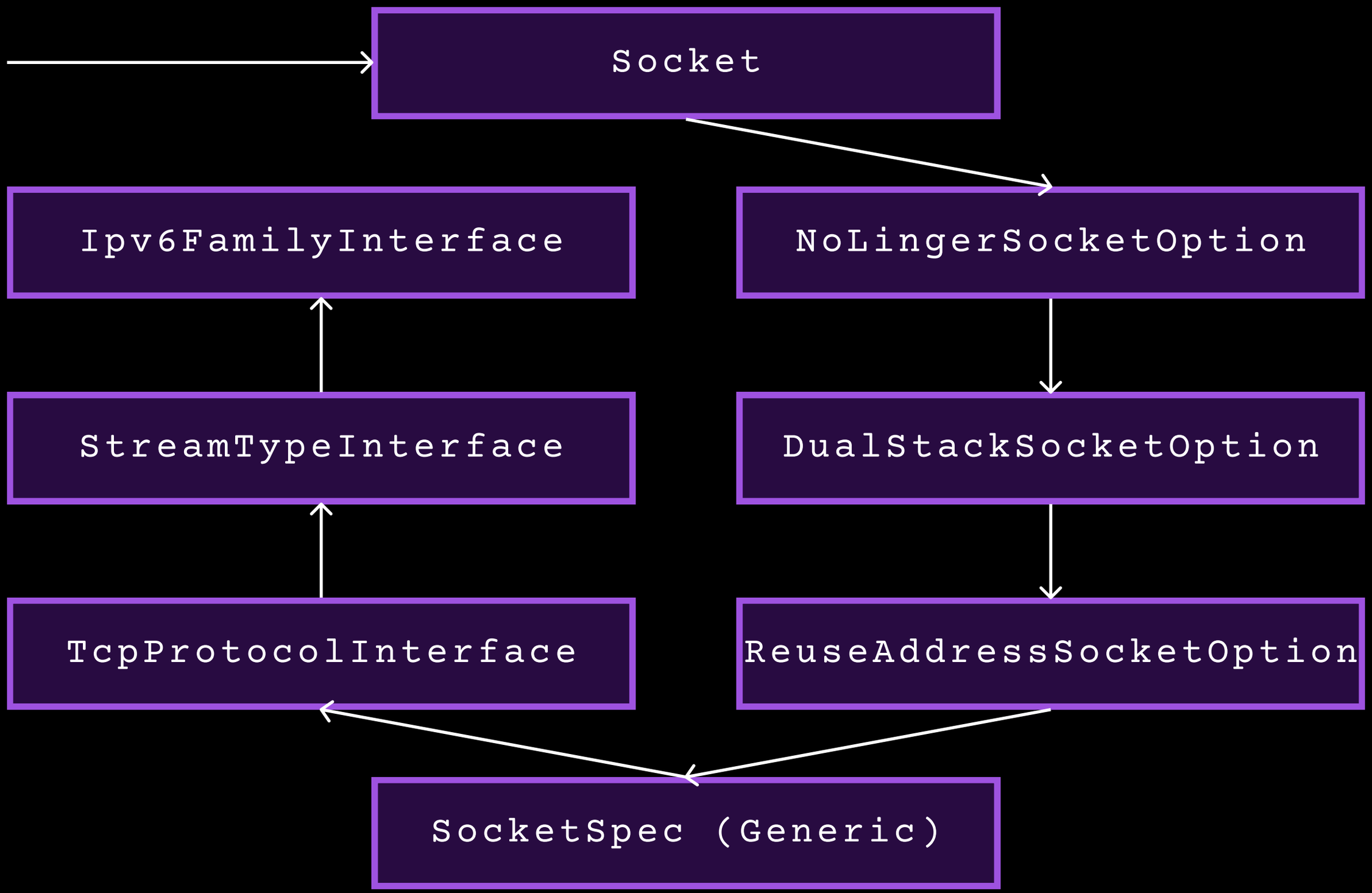
Rain::Networking Inheritance

Goals, Methodology

Example Inheritance



Example Inheritance



Inheritance: Goals

1. Strongly type `Socket` behavior.
2. Guarantee RAII on `Socket` resources.
3. Reuse shared protocol behavior code.
4. Solve for diamond inheritance.

```
std::unique_ptr<Emilia::Smtp::Server>
    smtpServer(new Emilia::Smtp::Server(...)),
    httpServer(new Emilia::Http::Server(...));

[...]

httpServer.reset(new Emilia::Http::Server(...));
```

There are 5 types of objects, and inheritance works differently between them.

Object Layer 1: Option/FTP

1. Options may be specified with variadic templated inheritance.
2. Family/Type/Protocol is specified with every Socket.
3. `SocketOption` CRTPs a `Generic` object for constructor calling order.

```
template <
    typename SocketFamilyInterface,
    typename SocketTypeInterface,
    typename SocketProtocolInterface,
    template <typename>
    class SocketOption,
    template <typename>
    class... SocketOptions>
class Socket<...>
      : public SocketOption<Socket<
          SocketFamilyInterface,
          SocketTypeInterface,
          SocketProtocolInterface,
          SocketOptions...>> {
    using...
};
```

Object Layer 1: Option/FTP

`Ipv4FamilyInterface` `NoLingerSocketOption` `...`

```
class Ipv4FamilyInterface :
    virtual public SocketFamilyInterface {
    public:
    virtual Family family() const noexcept
        final override { return Family::INET; }
};

template <typename Socket>
class NoLingerSocketOption : public Socket,
    virtual public NoLingerSocketOptionInterface {

    [...]

    virtual void
        alreadyNoLingerSocketOption() final {}
};
```

Object Layer 2: Generic

1. The only object which holds resources (i.e. system socket ID), and guarantees RAII in doing so. Cannot copy/move.
2. Template-inheritied by other layers to implement behavior.
3. Only one Generic object: `SocketSpec` (and its Interface).

```
template <
    typename SocketFamilyInterface,
    typename SocketTypeInterface,
    typename SocketProtocolInterface>
class SocketSpec<[...]> : virtual public [...] {
    [...]

    public:
    SocketSpec() :
        _nativeSocket(validateSystemCall(::socket(
        static_cast<int>(this->family()),
        static_cast<int>(this->type()),
        static_cast<int>(this->protocol())))) {
        this->unblock();
    }
};
```

Object Layer 3: Interface

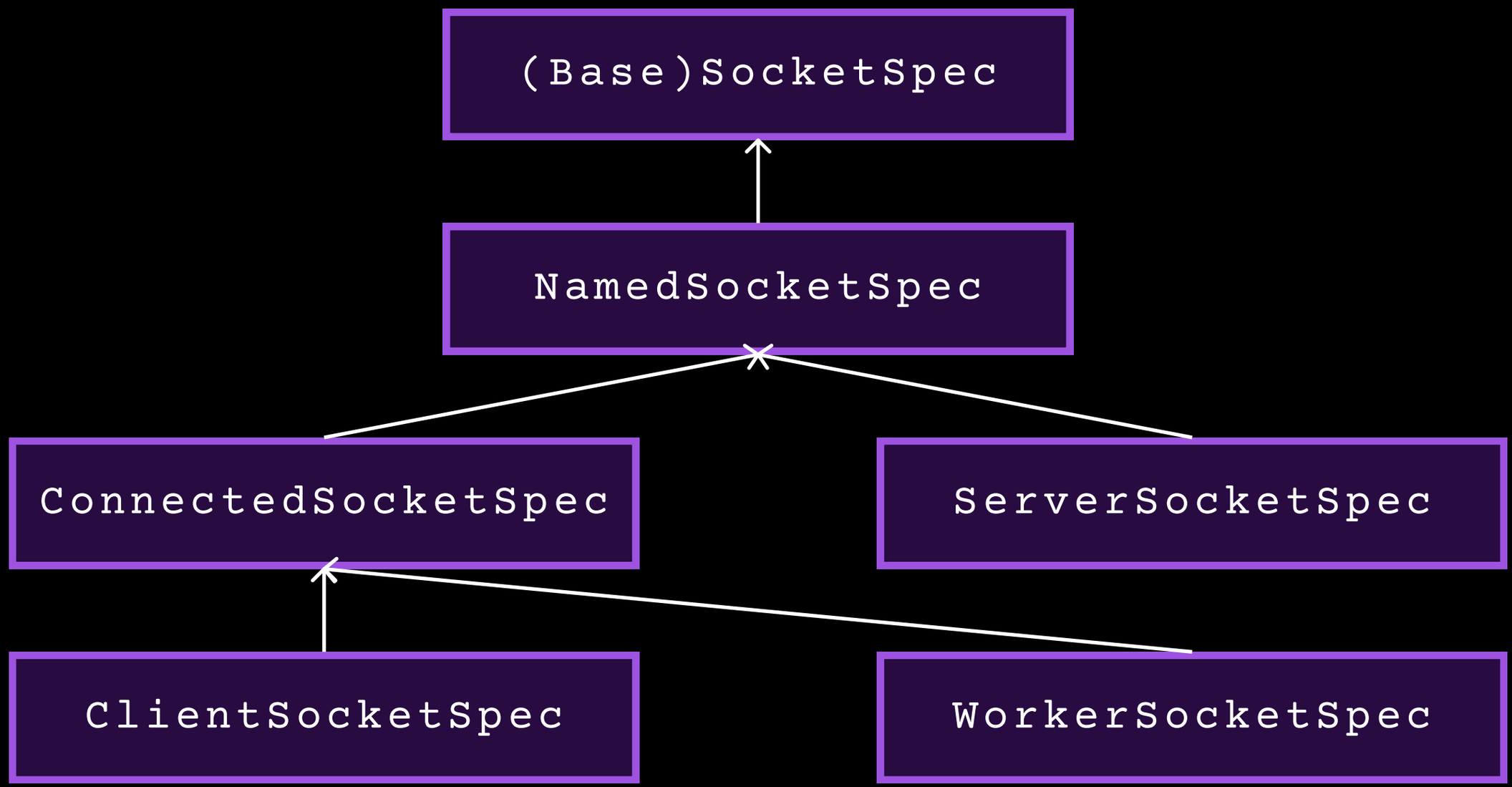
1. Pure virtual classes which do not hold resources nor template.
2. Used to inherit behavior which will be implemented by Protocol, Specification.
3. Almost every other object has an associated Interface.

```
class SocketInterface :
    virtual public [...] {
    [...]

    public:
    SocketInterface() = default;
    virtual ~SocketInterface() {}
    SocketInterface(
        SocketInterface const &) = delete;
    SocketInterface &operator=(
        SocketInterface const &) = delete;
    SocketInterface(SocketInterface &&) = delete;
    SocketInterface &operator=(
        SocketInterface &&) = delete;
    [...]
};
```

Object Layer 4: Specialization

1. A specialized socket is used for a specific purpose.
    a. (Base): most general and has no specific purpose.
    b. Named: a socket which is RAII-guaranteed to be bound to a hostname:port, and may listen on it.
    c. Connected: a Named socket which has been bound to a remote hostname:port.
    d. Worker: a Connected socket spawned from a listening server.
    e. Client: a Connected socket connected to a remote server.
    f. Server: a Named socket which listens on its bound hostname:port, and is templated by its Worker.
2. Specializations and Protocols are orthogonal.
3. Template-inherits Generic to hold resources, and inherits other Specializations to share behavior.
4. Inter-Specialization inheritance is not templated and fixed.
5. Remember: inheritance of Specializations is via the Specialization Interface, and not the Specialization itself.

Object Layer 4: Specialization



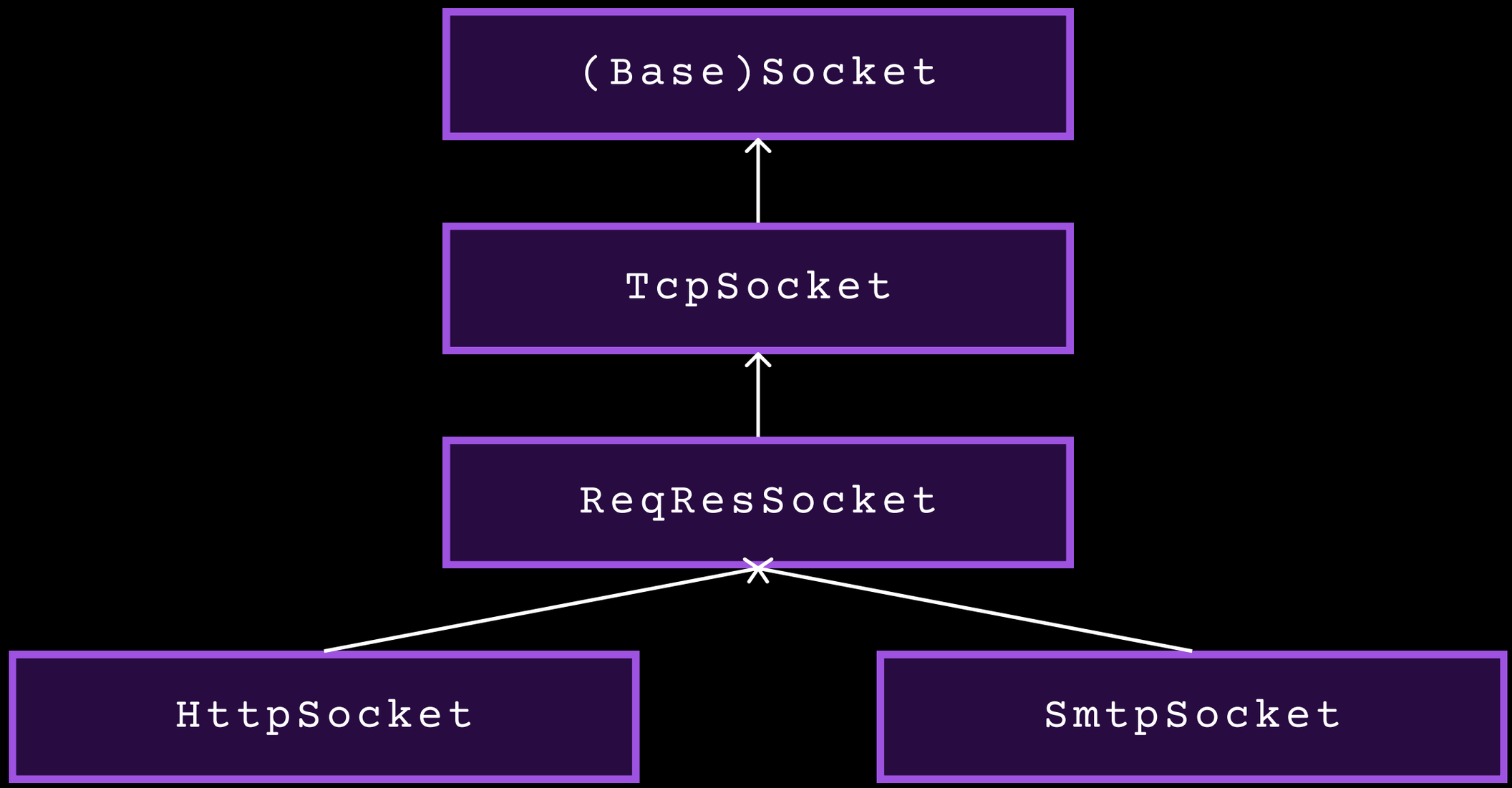
Object Layer 4: Specialization

```
class ClientSocketSpecInterfaceInterface :
    virtual public ConnectedSocketSpecInterface,
    virtual public Tcp::ClientSocketSpecInterface
{};

template <
    typename RequestMessageSpec,
    typename ResponseMessageSpec>
class ClientSocketSpecInterface : virtual public
    ClientSocketSpecInterfaceInterface {
  public:
  using ClientSocketSpecInterfaceInterface::send;
  using ClientSocketSpecInterfaceInterface::recv;

    [...]
};
```

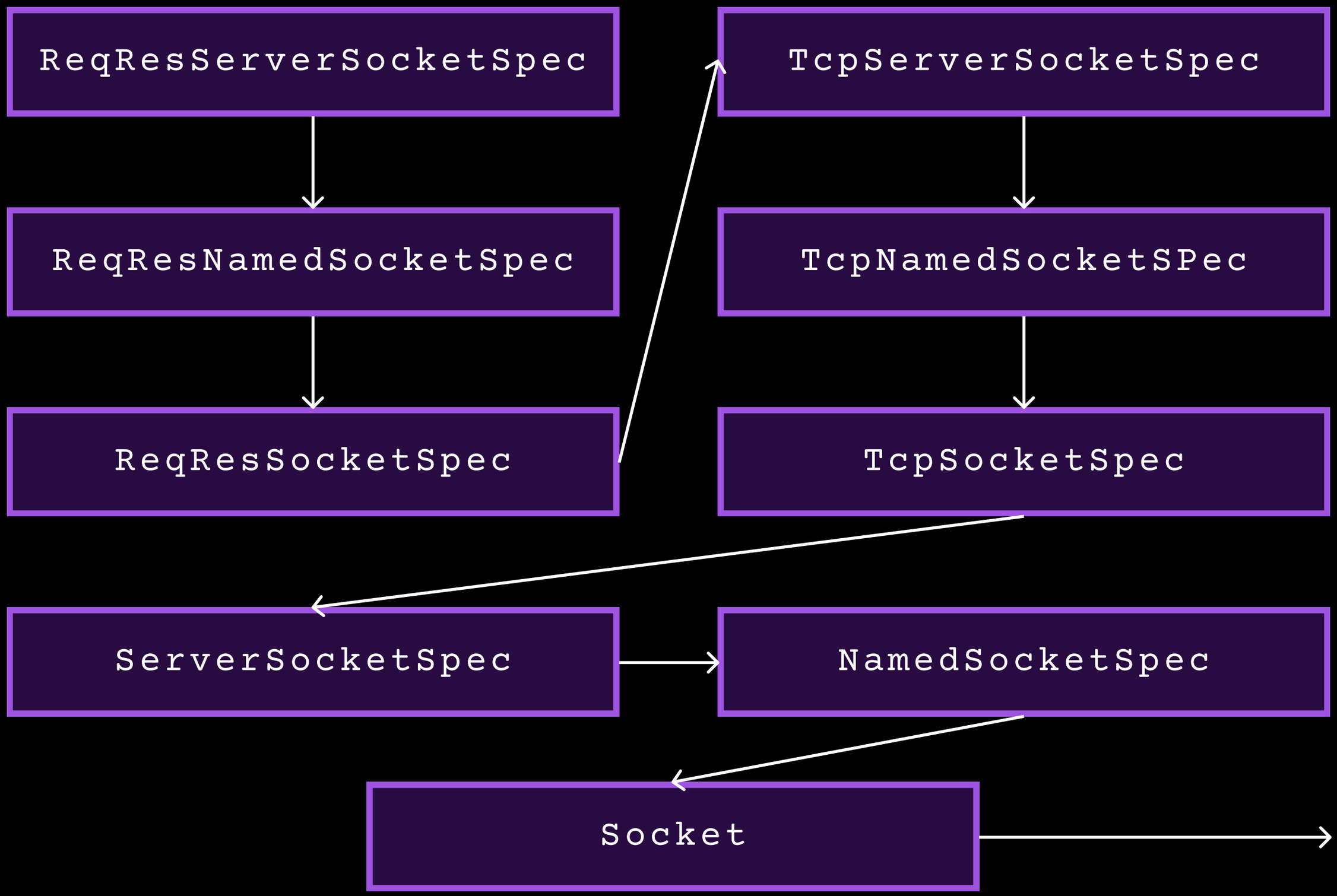
Object Layer 5: Protocol

1. Implements behavior specific to a particular socket communication protocol.
   a. (Base): no specific behavior.
   b. TCP: TCP sockets may read/write at any time, and are implemented with an underlying `std::streambuf`.
   c. ReqRes: a TCP socket with distinct chunks serving as Requests and Responses between a distinct Server and Client.
   d. HTTP: HTTP sockets additionally enforce certain formats on the request/response.
   e. SMTP: likewise, SMTP sockets also have their request/response requirements.
2. Inheritance of the single, resource-holding Generic will first traverse the inheritance of Specializations, before traversing the inheritance of Protocols. This solves the diamond inheritance problem.
3. Template-inherits a Specialization which template-inherits a resource-holding Generic.
4. Inter-Protocol inheritance is not templated and fixed.

Object Layer 5: Protocol



Example Inheritance



Example Inheritance