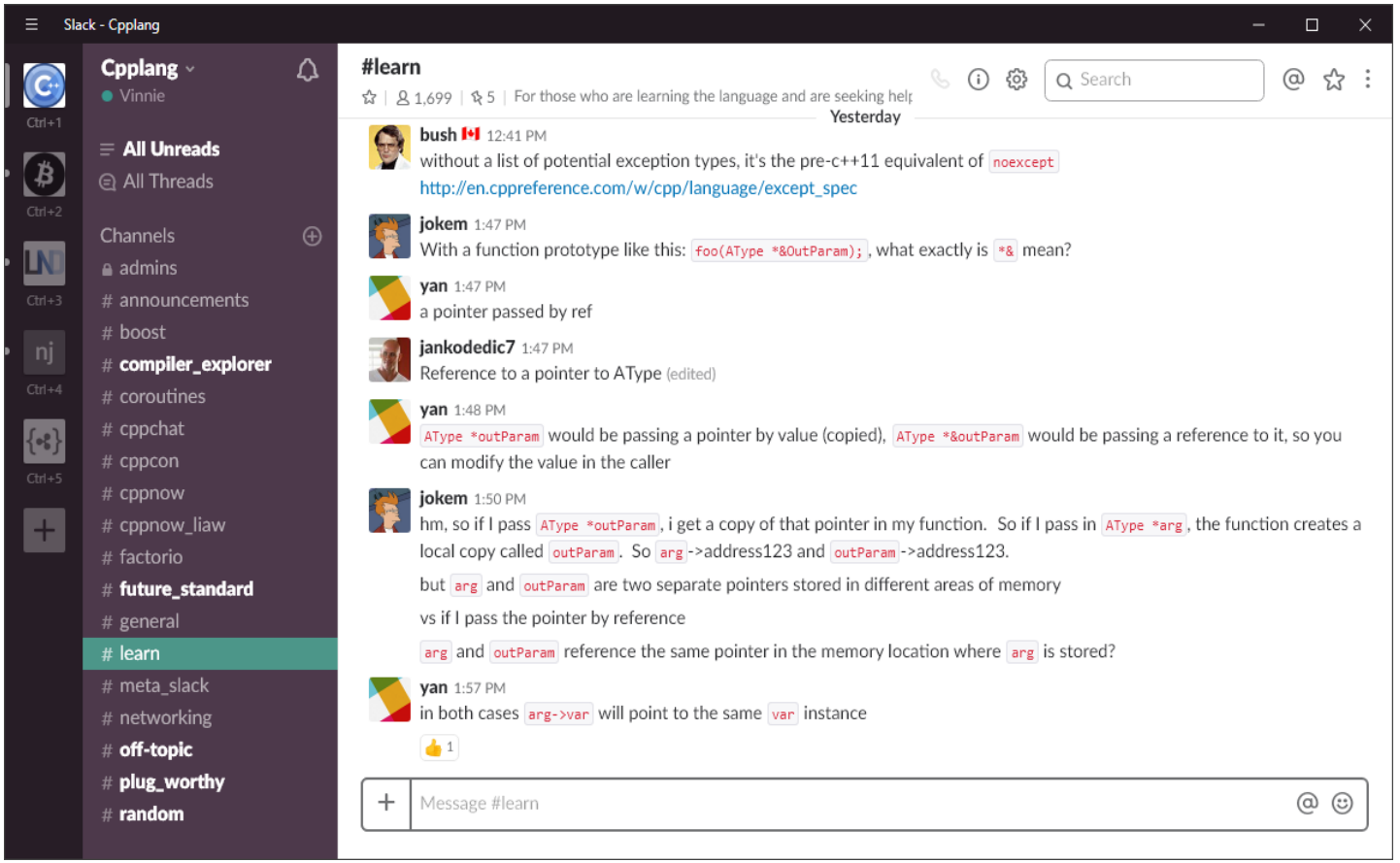
Interactive Websites:
Using Boost.Beast WebSockets and Networking TS

Vinnie Falco
Author of Boost.Beast

CppCon 2018
Sep 23-29

http://cppalliance.com

C++ Community



https://cpplang.slack.com

Sign up here:
https://cpplang.now.sh/

Boost.Beast

- HTTP and WebSocket protocols
- Using Boost.Asio
- Header-only C++11
- Part of Boost 1.66.0 and later
- Goal: Standardization

https://github.com/boostorg/beast

Boost C++ Libraries

- Establish existing practice
- Become part of C++

```
boost::shared_ptr
boost::optional
boost::bind
boost::mutex
boost::chrono
BOOST_FOR_EACH
```

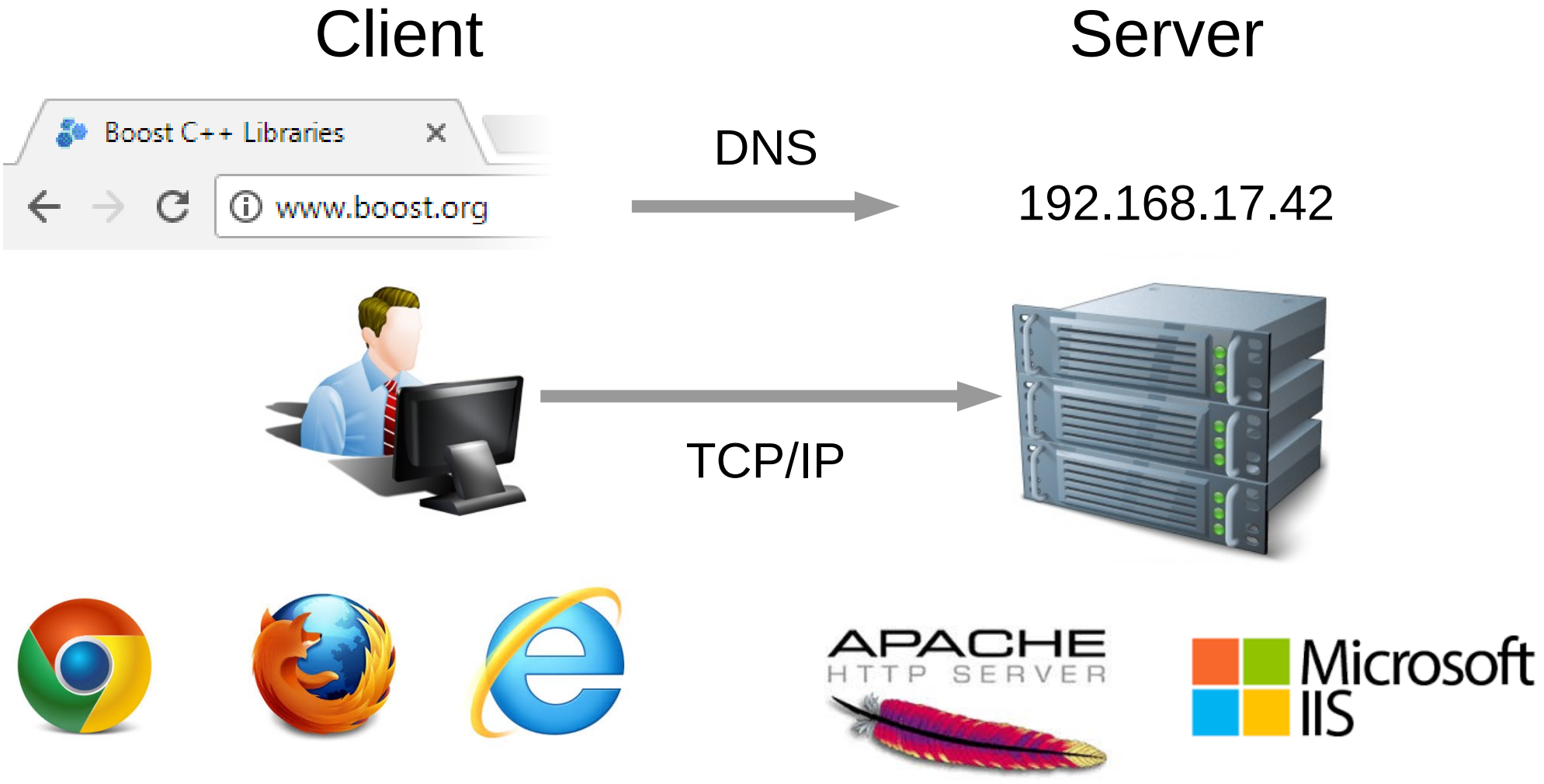
```
boost::asio
boost::filesystem
boost::thread
boost::shared_mutex
boost::function
BOOST_STATIC_ASSERT
```

Outline

1. HTTP/HTML
2. Interactive Web
3. WebSockets
4. Asio
5. Chat Server

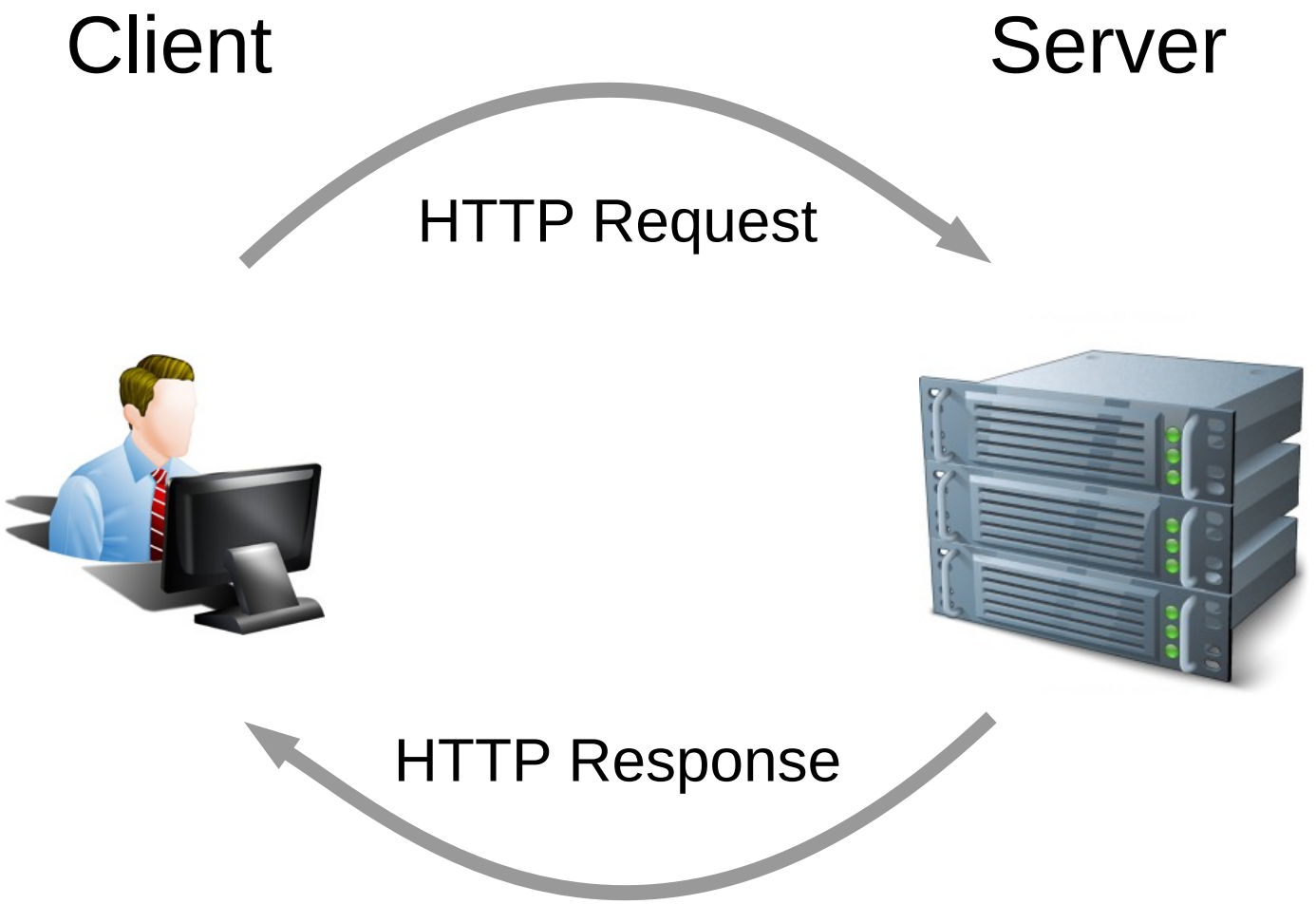
HTTP

- “Hypertext Transfer Protocol”



HTTP

- Half-Duplex



WebSocket

- WebSocket Upgrade HTTP request

```
GET /chat.cgi HTTP/1.1
Host: www.example.org
Upgrade: websocket
Connection: upgrade
Sec-WebSocket-Key: 2pGeTR08MA==
Sec-WebSocket-Version: 13
User-Agent: Beast
```

WebSocket

TARGET VERSION

```
GET /chat.cgi HTTP/1.1
Host: www.example.org
Upgrade: websocket
Connection: upgrade
Sec-WebSocket-Key: 2pGeTR08MA==
Sec-WebSocket-Version: 13
User-Agent: Beast
```

WebSocket

- WebSocket Upgrade HTTP response

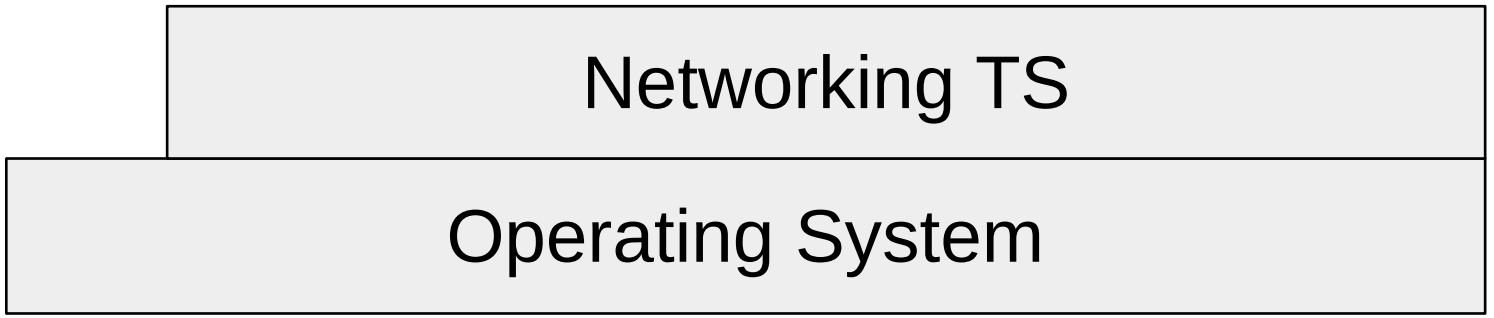
```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: upgrade
Sec-WebSocket-Accept: shZRK+xOo=
Server: Beast
```

Networking

- WinSock (Windows)
- epoll (Linux)
- kqueue (BSD, OSX)

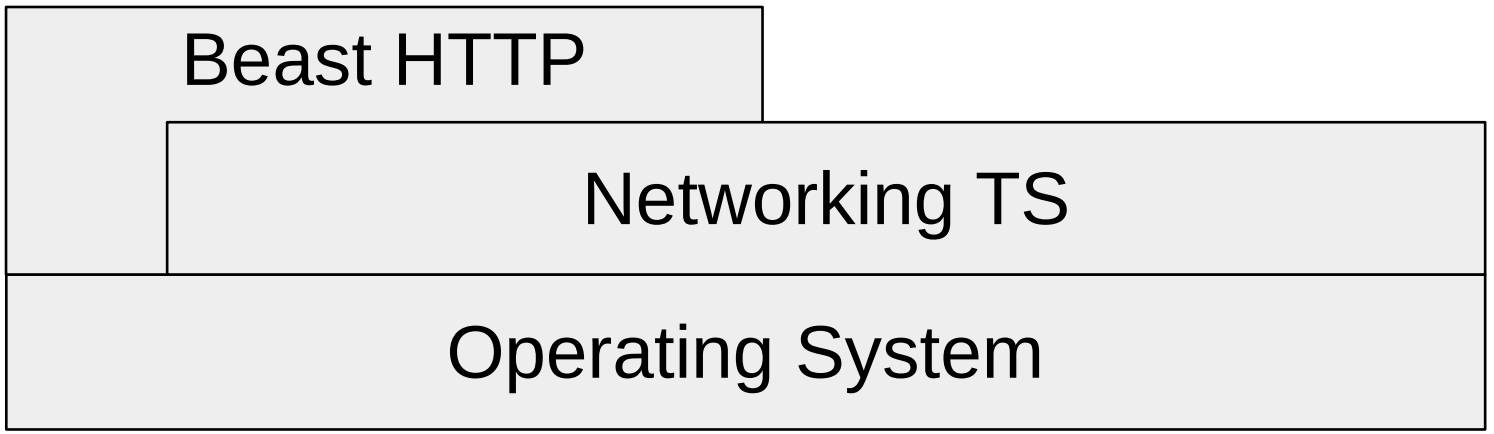
Networking

- Networking TS
- Net-TS flavored Boost.Asio
- Net-TS flavored Standalone Asio



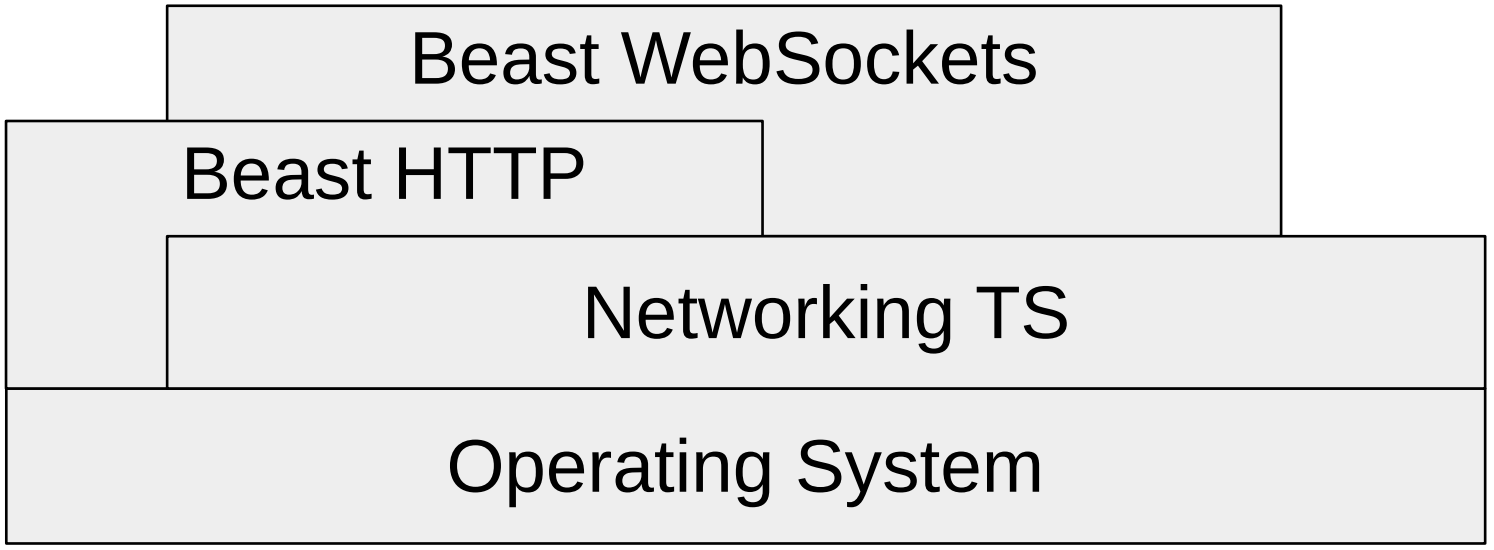
Networking

- Beast HTTP



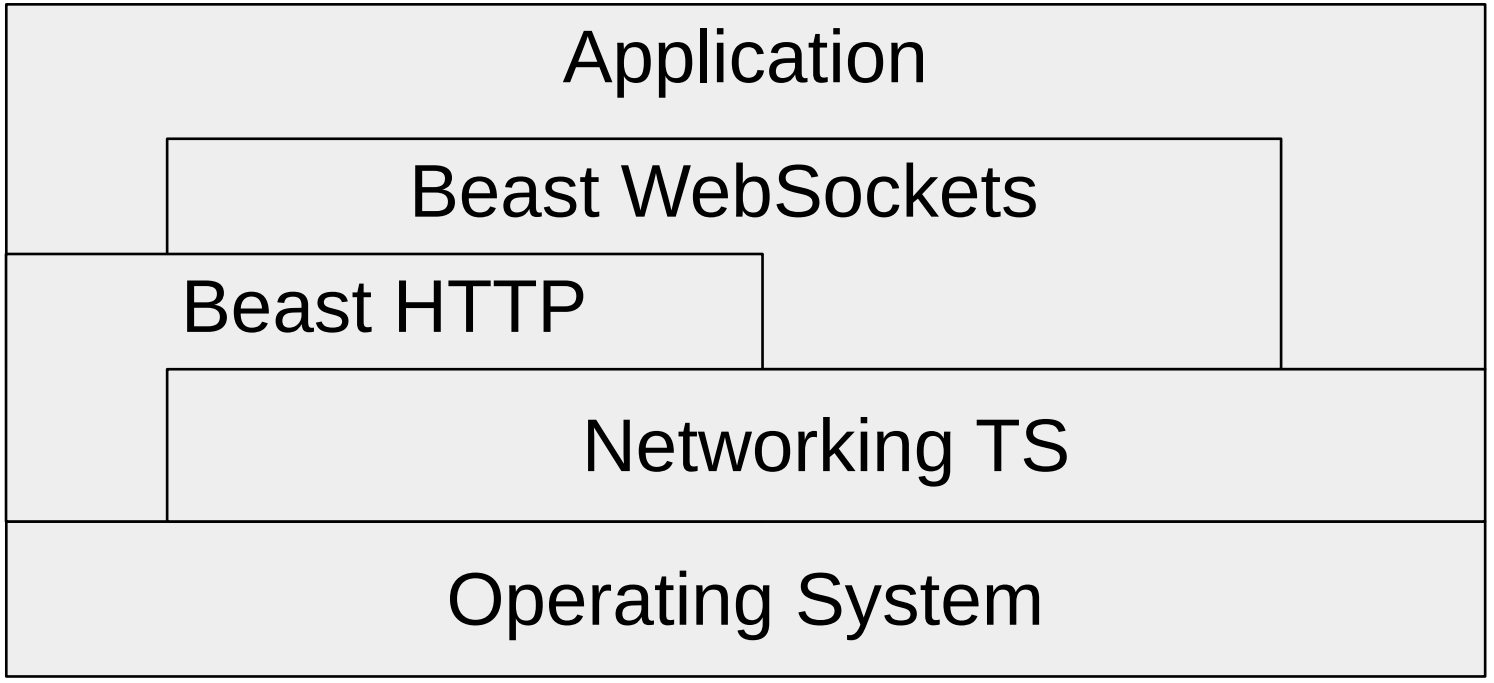
Networking

- Beast WebSockets



Networking

- Application uses HTTP, WebSockets, Net TS



Asio

- Asynchronous Model

```
/// Initiate a read
async_read(sock, buffers, handler);
```

Shared Pointer

```
/// Create a shared pointer to a modifiable T
shared_ptr<T> sp = make_shared<T>;

/// csp is cheaply constructible from sp
shared_ptr<T const> csp = sp;

/// vsp is cheaply constructible from any shared_ptr
shared_ptr<void> vsp = sp; // or csp

/// Going from csp to sp is disallowed
sp = csp; // const violation
```

Asio

- Socket: Not thread-safe

websocket-chat-server

```
// Creates and runs the server
int main(int argc, char* argv[]);

// Accepts incoming connections
class listener;

// Handles HTTP requests on a connection
class http_session;

// Maintains an active WebSocket session
class websocket_session;

// Holds the chat room data
class shared_state;
```

websocket-chat-server

```
int main(int argc, char* argv[])
{
    // Usage:
    // websocket-chat-server <address> <port> <doc_root>

    auto const address =
        asio::ip::make_address(argv[1]);

    auto const port =
        static_cast<unsigned short>(std::atoi(argv[2]));

    auto const doc_root = argv[3];

    // The io_context is required for all I/O
    asio::io_context ioc;

    ...
```

websocket-chat-server

```
...

// Create and launch a listening port
std::make_shared<listener>(
    ioc,
    tcp::endpoint{address, port},
    std::make_shared<shared_state>(doc_root))→run();

...
```

websocket-chat-server

```
    ...

    // Capture SIGINT and SIGTERM to perform a clean shutdown
    asio::signal_set signals(ioc, SIGINT, SIGTERM);
    signals.async_wait(
        [&](boost::system::error_code const&, int)
        {
            // Stop the `io_context`. This will cause `run()`
            // to return immediately, eventually destroying the
            // `io_context` and all of the handlers in it.
            ioc.stop();
        });

    // Run the I/O service on the main thread
    ioc.run();

    // (If we get here, it means we got a SIGINT or SIGTERM)

    return EXIT_SUCCESS;
}
```

websocket-chat-server

```
class shared_state
{
    std::string const doc_root_;
    std::unordered_set<websocket_session*> sessions_;

public:
    explicit
    shared_state(std::string doc_root);

    std::string const&
    doc_root() const noexcept
    {
        return doc_root_;
    }

    void join  (websocket_session& session);
    void leave (websocket_session& session);
    void send  (std::string message);
};
```

websocket-chat-server

```
class listener : public std::enable_shared_from_this<listener>
{
    tcp::acceptor acceptor_;
    tcp::socket socket_;
    std::shared_ptr<shared_state> state_;

    void fail(error_code ec, char const* what);
    void on_accept(error_code ec);

public:
    listener(
        asio::io_context& ioc,
        tcp::endpoint endpoint,
        std::shared_ptr<shared_state> const& state);

    // Start accepting incoming connections
    void run();
};
```

websocket-chat-server

```
// Start accepting incoming connections
void listener::run()
{
    acceptor_.async_accept(
        socket_,
        bind(
            &listener::on_accept,
            shared_from_this(),
            std::laceholders::_1));
}
```

websocket-chat-server

```
void listener::on_accept(error_code ec)
{
    if(ec)
        fail(ec, "accept");
    else
        std::make_shared<http_session>(
            std::move(socket_),
            state_)->run();

    // Accept another connection
    acceptor_.async_accept(
        socket_,
        std::bind(
            &listener::on_accept,
            shared_from_this(),
            std::placeholders::_1));
}
```

websocket-chat-server

```
// Report a failure
void
listener::
fail(error_code ec, char const* what)
{
    // Don't report on canceled operations
    if(ec != asio::error::operation_aborted)
        std::cerr <<
          what << ": " <<
          ec.message() << "\n";
}
```