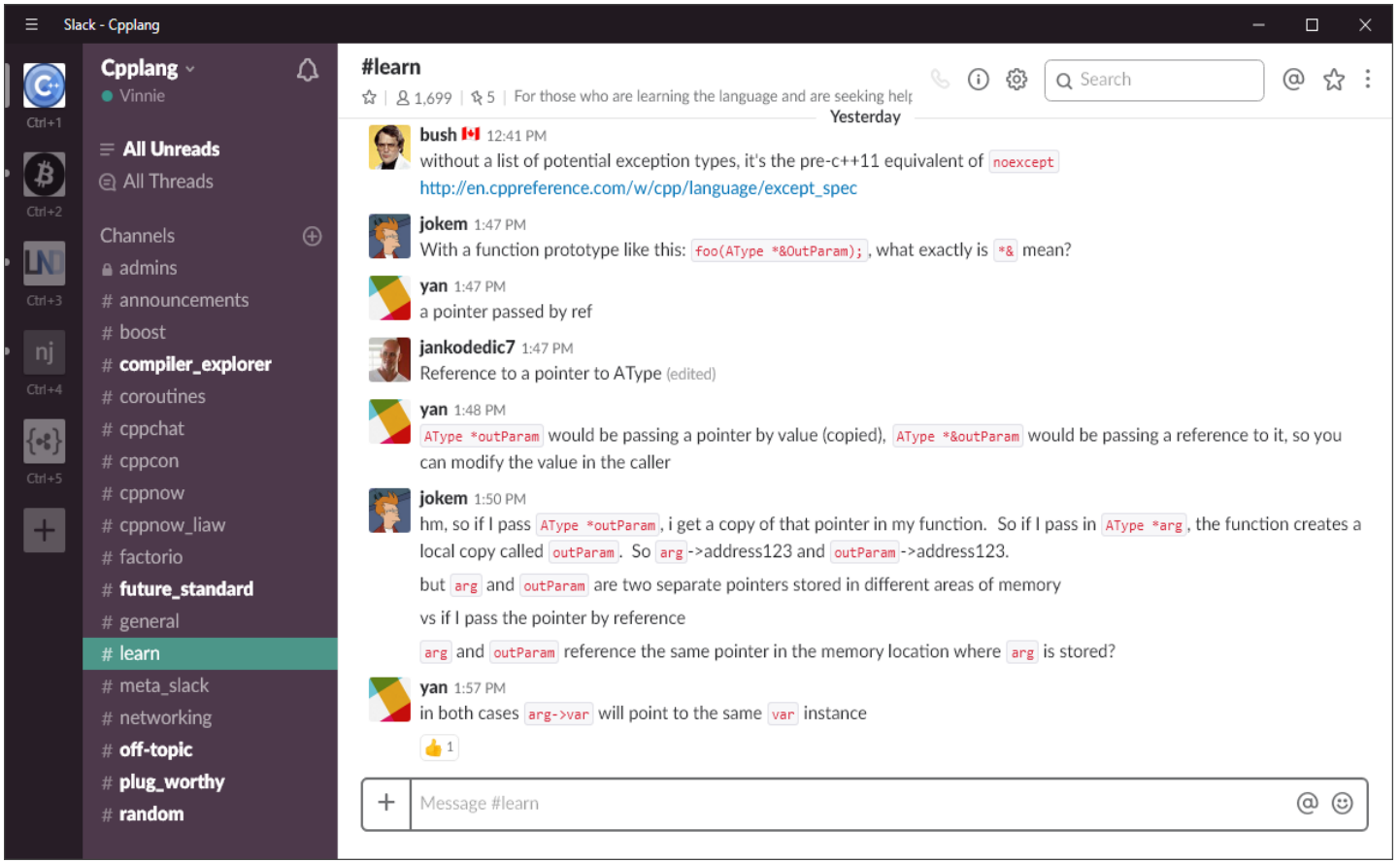
Interactive Websites:
Using Boost.Beast WebSockets and Networking TS

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

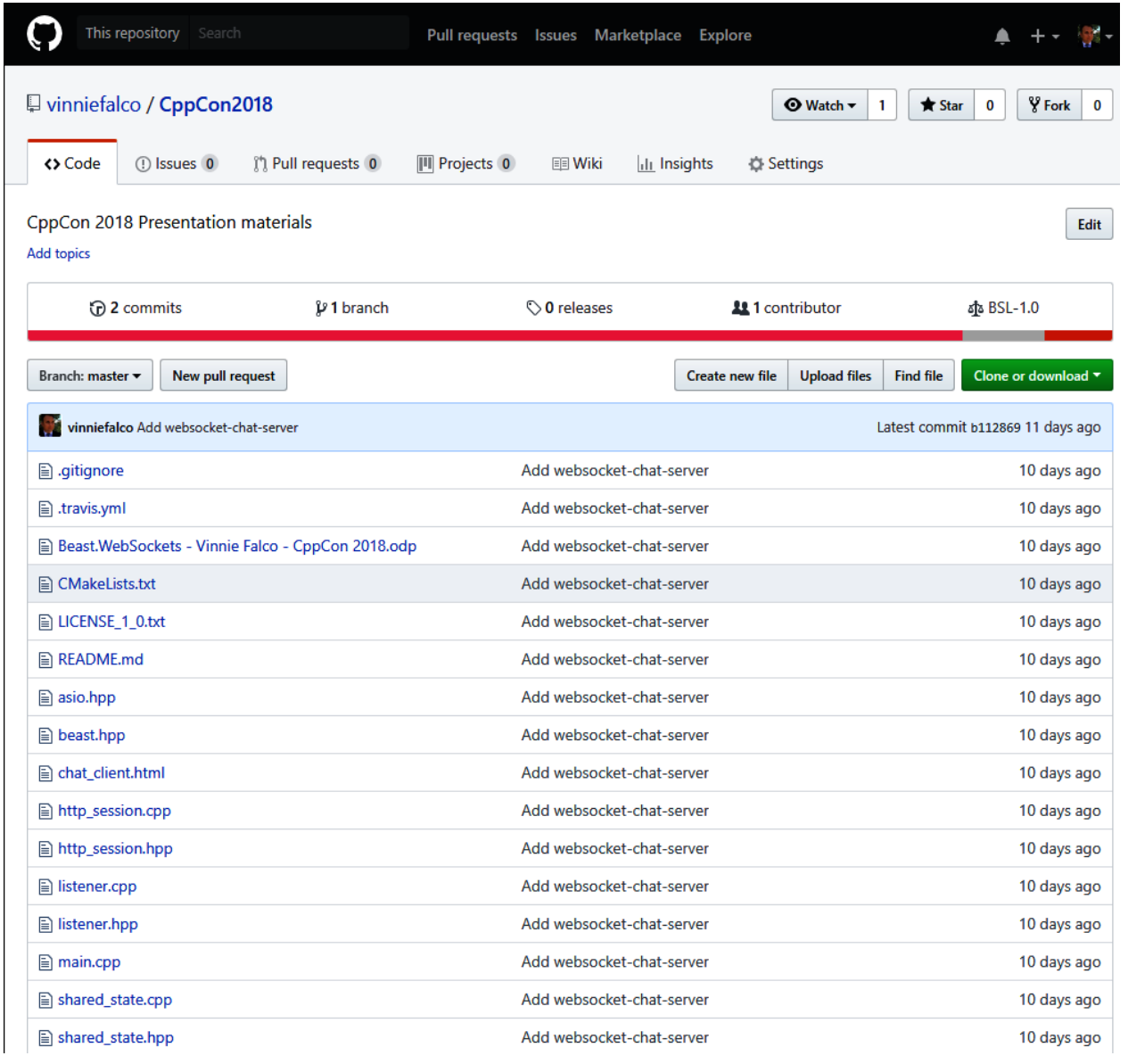
- Establish existing practice
- Become part of C++

```
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
```

Repository

https://github.com/vinniefalco/CppCon2018



SPOILER ALERT!


WebSocket
HTTP

Contents

- Web Protocols
- Interactive Web
- WebSockets
- Networking TS
- Chat Server
- Chat Client
- Education

Web Protocols

HTTP

- “Hypertext Transfer Protocol”

Client

DNS

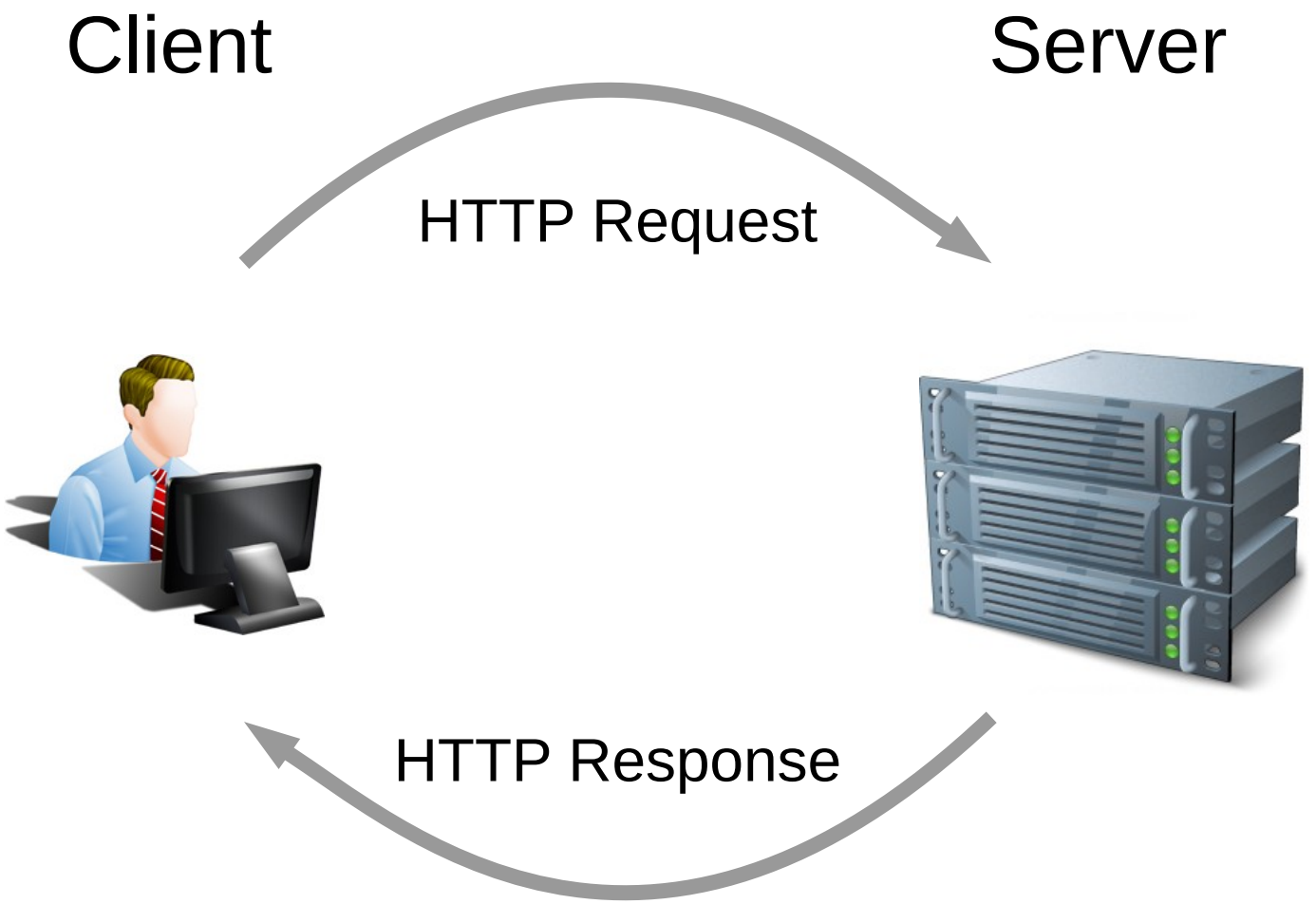
Server

192.168.17.42

TCP/IP

HTTP

- Half-Duplex



WebSockets

WebSockets

- Message oriented
- Routable
- Symmetric, full-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

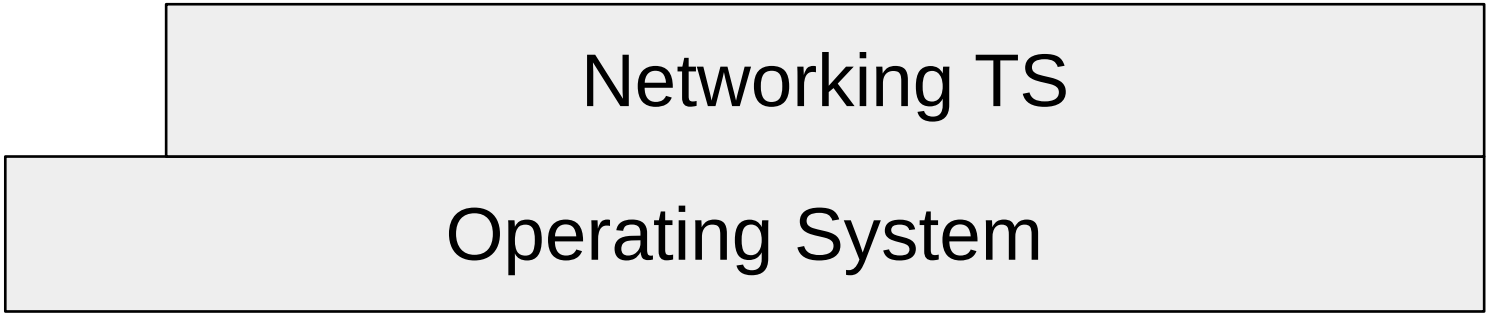
```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: upgrade
Sec-WebSocket-Accept: shZRK+xOo=
Server: Beast
```

Networking

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

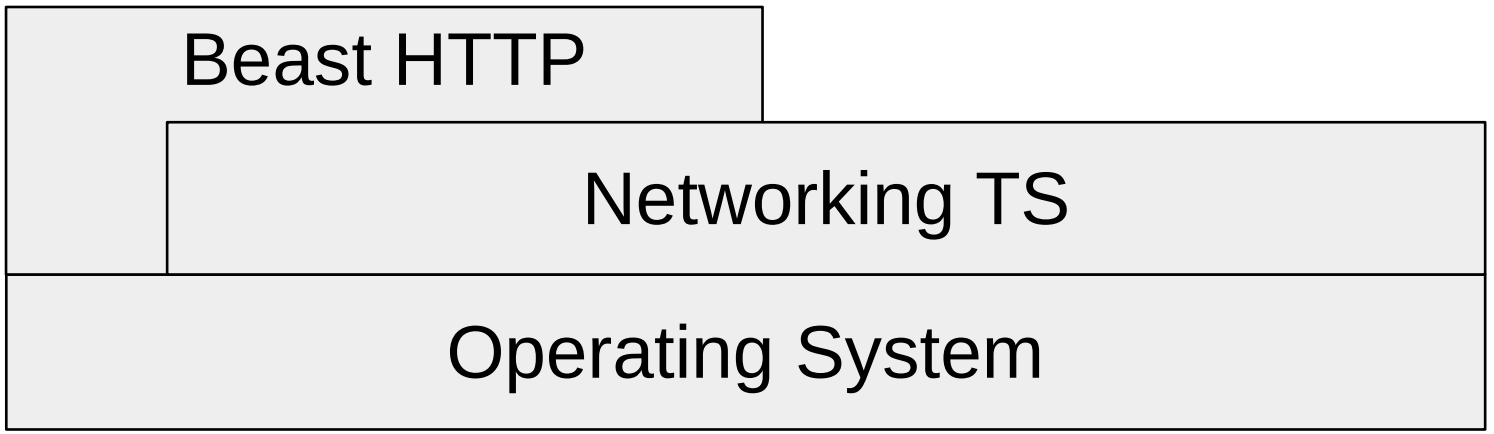
Networking

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



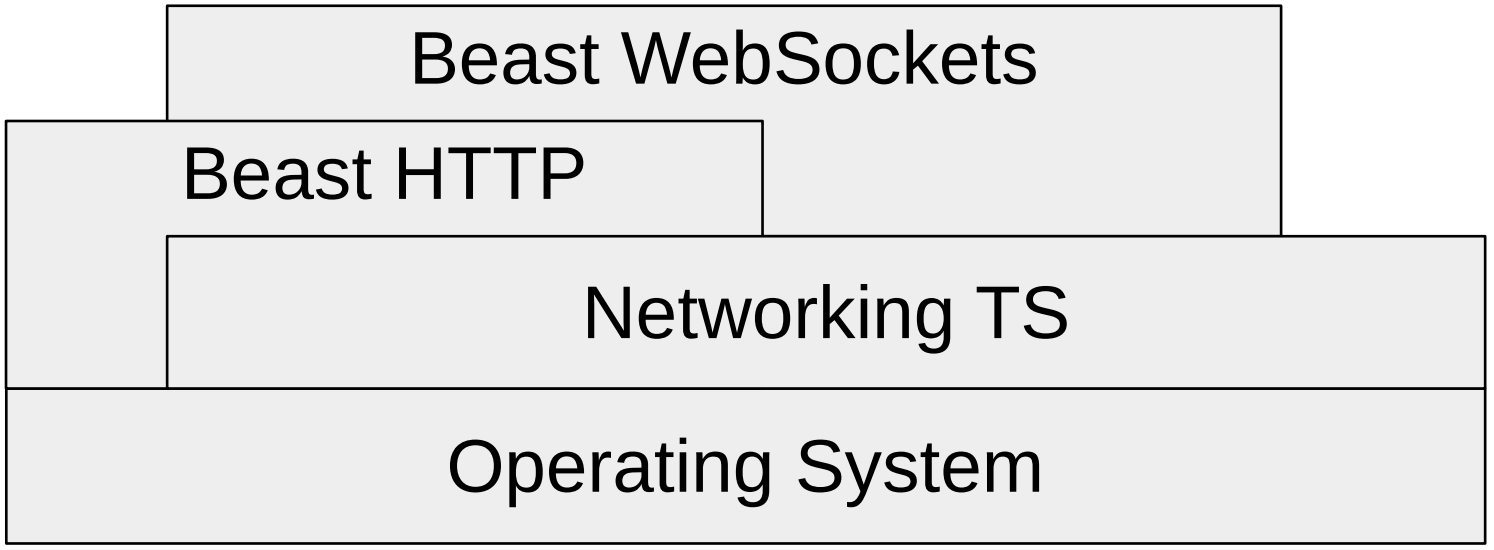
Networking

- Beast HTTP



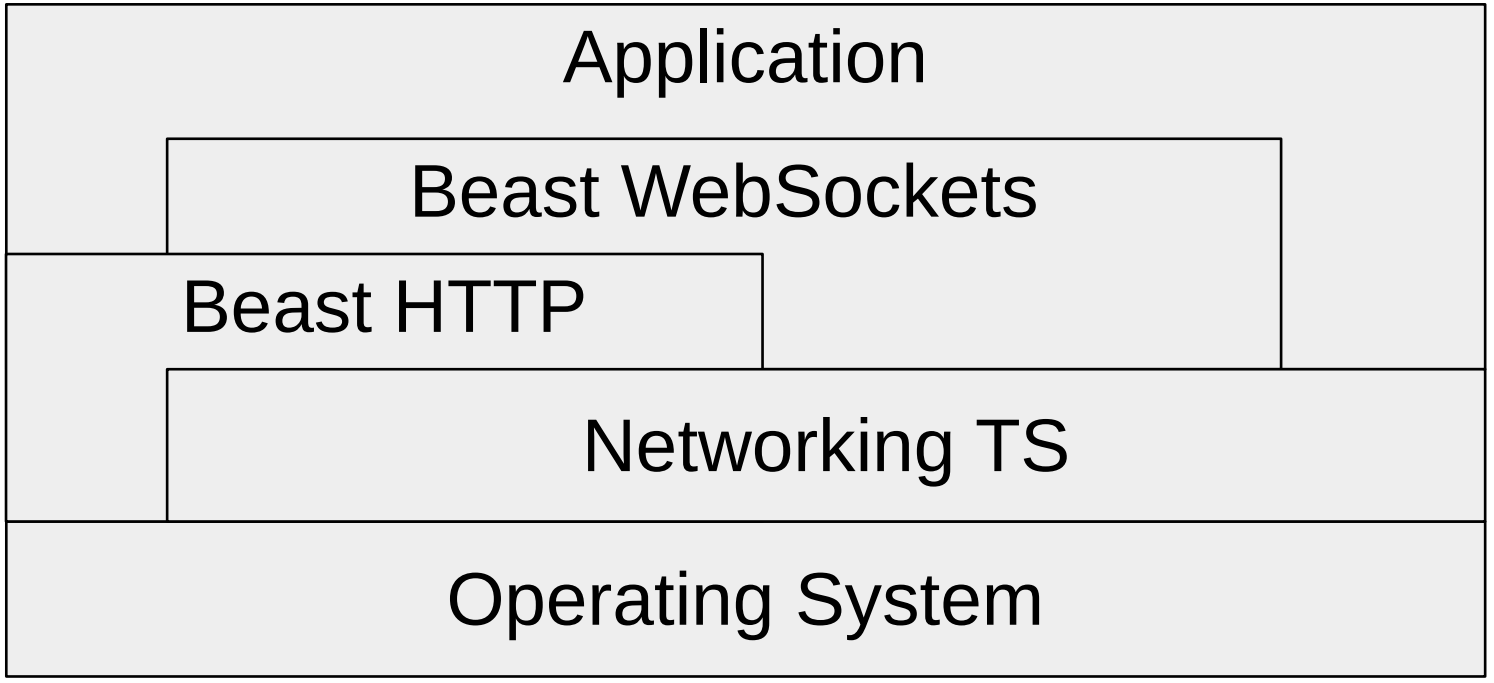
Networking

- Beast WebSockets



Networking

- Application uses HTTP, WebSockets, Net TS



Asio

- Asynchronous Model

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

Asio

- Socket: Not thread-safe

Boost.Beast HTTP

- `boost::beast::http::message` container

https://youtu.be/WsUnnYEKPnI

Networking.TS

Networking TS

Library	Header file location and namespace
Networking TS	`<experimental/io_context>` `std::experimental::net::io_context`
Boost.Asio	`<boost/asio/io_context.hpp>` `boost::asio::io_context`
Asio	`<asio/io_context.hpp>` `asio::io_context`

Networking TS

```
// Required for ALL I/O activities
io_context ioc;

// Declare a TCP/IP socket
ip::tcp::socket sock(ioc);

// Write some data
auto bytes_transferred = sock.write_some(b);
```

Networking TS

```
template<class ConstBufferSequence>
size_t
socket::write_some(
  ConstBufferSequence const& b);

template<class MutableBufferSequence>
size_t
socket::read_some(
  MutableBufferSequence const& b);
```

Buffers

```
class mutable_buffer
{
  ...
public:
  mutable_buffer();

  mutable_buffer(
    void*, size_t);




  size_t size();

  void const* data();
};
```

```
class const_buffer
{
  ...
public:
  const_buffer();

  const_buffer(
    void const*, size_t);

  const_buffer(
    mutable_buffer);

  size_t size();

  void const* data();
};
```

ConstBufferSequence

A *ConstBufferSequence* is a non-owning range of read-only memory regions.

In this table:
- **X** is a type meeting the requirements of *ConstBufferSequence*
- **a** is a value of type **X**

Expression	Type	Description
`X::value_type`	`T`	`T` is convertible to `const_buffer`.
`X::const_iterator`	`U`	`U` is a bidirectional iterator whose `reference` type is convertible to `const_buffer`.
`X(a)`	`X`	`X` is *CopyConstructible*. The copy will reference the same memory regions as the original sequence.
`a.begin();` `a.end();`	`const_iterator` or convertible to `const_iterator`	
`const_buffer` `mutable_buffer`		Also a *ConstBufferSequence*.

MutableBufferSequence

A *MutableBufferSequence* is a non-owning range of mutable memory regions.

In this table:
- **X** is a type meeting the requirements of *MutableBufferSequence*
- **a** is a value of type **X**

Expression	Type	Description
`X::value_type`	`T`	`T` is convertible to `mutable_buffer`.
`X::const_iterator`	`U`	`U` is a bidirectional iterator whose `reference` type is convertible to `mutable_buffer`.
`X(a)`	`X`	`X` is *CopyConstructible*. The copy will reference the same memory regions as the original sequence.
`a.begin();` `a.end();`	`const_iterator` or convertible to `const_iterator`	
`mutable_buffer`		Also a *MutableBufferSequence*.

Buffers

```
string s = “Hello, world!”;

sock.write_some(
    const_buffer(s.data(), s.size()));

// Better
sock.write_some(buffer(s));
```

Buffers

```
template<typename PodType, size_t N>
mutable_buffer buffer(array<PodType, N>& data);

template<typename PodType, size_t N>
const_buffer buffer(array<PodType, N> const& data);

template<typename PodType, typename Allocator>
mutable_buffer buffer(vector<PodType, Allocator>& data);

template<typename PodType, typename Allocator>
const_buffer buffer(vector<PodType, Allocator> const& data);

template<typename Elem, typename Traits, typename Allocator>
mutable_buffer buffer(basic_string<Elem, Traits, Allocator>& data);

template<typename Elem, typename Traits, typename Allocator>
const_buffer buffer(basic_string<Elem, Traits, Allocator> const& data);

// ...26 more overloads
```

Buffers

```
// Read data into a buffer until
// it contains a specified string
auto matching_bytes = read_until(sock, b, “\r\n\r\n”);
```

Buffers

```
// Read data into a buffer until
// it contains a specified string
auto matching_bytes = read_until(sock, b, “\r\n\r\n”);

template<class SyncReadStream, class DynamicBuffer>
size_t
read_until(
    SyncReadStream& s,
    DynamicBuffer&& buffers,
    string_view match);
```

Threading Models

Model	Features	Notes
single threaded, single `io_context`	• Fastest • Limited capacity • Implicit strand	Easiest to write
multi-threaded, single `io_context`	• Highest capacity • Explicit strand	Capacity for overhead
multi-threaded, `io_context` per thread	• Fastest • Highest capacity • Implicit strand • Needs balancing	Most complex

Chat Server

Chat Server

```
// Boost/Asio
namespace net = boost::asio;
using tcp = net::ip::tcp;
using error_code = boost::system::error_code;

// Beast
namespace beast = boost::beast;
namespace http = boost::beast::http;
namespace websocket = boost::beast::websocket;
```

Chat Server

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

// Holds the server data
class shared_state;

// Accepts incoming connections
class listener;

// Handles HTTP requests on a connection
class http_session;

// Maintains an active WebSocket session
class websocket_session;
```

Chat Server

```
class shared_state
{
   std::string 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);
};
```

Chat Server

```
void shared_state::join(websocket_session& session)
{
    sessions_.insert(&session);
}

void shared_state::leave(websocket_session& session)
{
    sessions_.erase(&session);
}

void shared_state::send(std::string message)
{
    auto const ss = std::make_shared<
        std::string const>(std::move(message));

    for(auto session : sessions_)
        session->send(ss);
}
```

Chat Server

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

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

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

    auto doc_root = argv[3];

    ...
```

Chat Server

```
int main(int argc, char* argv[])
   ...

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

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

   ...
```

Chat Server

```
int main(int argc, char* argv[])
   ...

   // Capture SIGINT and SIGTERM for clean shutdown
   asio::signal_set signals(ioc, SIGINT, SIGTERM);
   signals.async_wait(
      [&ioc](error_code const&, int)
      {
         // Stop the io_context. This will cause run()
         // to return immediately,  destroying the
         // io_context and any remaining handlers
         ioc.stop();
      });

   ioc.run();

   return EXIT_SUCCESS;
}
```

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);

    void run();
};
```

Chat Server

```
void listener::run()
{
   // Start accepting a connection.
   acceptor_.async_accept(
      socket_,
      [self = shared_from_this()](error_code ec)
      {
         self->on_accept(ec);
      });
}
```

Chat Server

```
void listener::on_accept(error_code ec)
{
    if(ec)
        fail(ec, "accept");
    else
        // Launch a new session for this connection
        std::make_shared<http_session>(
            std::move(socket_), state_)->run();

    // Accept another connection
    acceptor_.async_accept(
        socket_,
        [self = shared_from_this()](error_code ec)
        {
            self->on_accept(ec);
        });
}
```

Chat Server

```
void listener::fail(
    error_code ec, char const* what)
{
    // Don't report on canceled operations
    if(ec == asio::error::operation_aborted)
        return;

    std::cerr <<
        what << ": " <<
        ec.message() << "\n";
}
```

Chat Server

```
class http_session
    : public std::enable_shared_from_this<http_session>
{
    tcp::socket socket_;
    beast::flat_buffer buffer_;
    std::shared_ptr<shared_state> state_;
    http::request<http::string_body> req_;

    void fail(error_code ec, char const* what);
    void on_read(error_code ec, std::size_t);
    void on_write(error_code ec, std::size_t, bool close);

public:
    http_session(tcp::socket socket,
                 std::shared_ptr<shared_state> const& state);

     void run();
};
```

Chat Server

```
http_session::http_session(
    tcp::socket socket,
    std::shared_ptr<shared_state> const& state)
    : socket_(std::move(socket))
    , state_(state)
{
}

void http_session::run()
{
    // Read a request
    http::async_read(socket_, buffer_, req_,
        [self = shared_from_this()]
            (error_code ec, std::size_t bytes)
        {
            self->on_read(ec, bytes);
        });
}
```

Chat Server

```
void http_session::on_read(error_code ec, std::size_t)
{
    // This means they closed the connection
    if(ec == http::error::end_of_stream)
    {
        socket_.shutdown(tcp::socket::shutdown_send, ec);
        return;
    }

    if(ec)
        return fail(ec, "read");

    // Handle WebSocket Upgrade
    if(websocket::is_upgrade(req_))
    {
        // Create a WebSocket session by transferring the socket
        std::make_shared<websocket_session>(
            std::move(socket_), state_)->run(std::move(req_));
        return;
    }

    ...
```

Chat Server

```
void http_session::on_read(error_code ec, std::size_t)
    ...

    handle_request(state_->doc_root(), std::move(req_),
        [this](auto&& response)
        {
            using response_type = typename
                std::decay<decltype(response)>::type;

            auto sp = std::make_shared<response_type>(
                std::move(response));

            http::async_write(this->socket_, *sp,
                [self = shared_from_this(), sp](
                    error_code ec, std::size_t bytes)
                {
                    self->on_write(ec, bytes, sp->need_eof());
                });
        });
}
```

Chat Server

```
void http_session::on_write(
    error_code ec, std::size_t, bool close)
{
    if(ec)
        return fail(ec, "write");

    if(close)
    {
        // This means we should close the connection,
        socket_.shutdown(tcp::socket::shutdown_send, ec);
        return;
    }

    // Clear contents of the request message,
    // otherwise the read behavior is undefined.
    req_ = {};

    // Read another request
    http::async_read(socket_, buffer_, req_,
        [self = shared_from_this()]
            (error_code ec, std::size_t bytes)
        {
            self->on_read(ec, bytes);
        });
}
```

Chat Server

```
class websocket_session
    : public std::enable_shared_from_this<websocket_session>
{
    beast::flat_buffer buffer_;
    websocket::stream<tcp::socket> ws_;
    std::shared_ptr<shared_state> state_;
    std::vector<std::shared_ptr<std::string const>> queue_;

    void fail(error_code ec, char const* what);
    void on_accept(error_code ec);
    void on_read(error_code ec, std::size_t);
    void on_write(error_code ec, std::size_t);

public:
    ~websocket_session();
    websocket_session(tcp::socket socket,
                      std::shared_ptr<shared_state> const& state);

    template<class Body, class Allocator>
    void run(http::request<Body, http::basic_fields<Allocator>> req);

    void send(std::shared_ptr<std::string const> const& ss);
};
```

Chat Server

```
template<class Body, class Allocator>
void websocket_session::
run(
    http::request<Body, http::basic_fields<Allocator>> req)
{
     // Accept the websocket handshake
     ws_.async_accept(
         req,
       [self = shared_from_this](error_code ec)
       {
          self->on_accept(ec);
       });
}
```

Chat Server

```
void websocket_session::
on_accept(error_code ec)
{
    if(ec)
        return fail(ec, "accept");

    // Add this session to the list
    state_->join(*this);

    // Read a message
    ws_.async_read(
        buffer_,
        [sp = shared_from_this()](
            error_code ec, std::size_t bytes)
        {
            sp->on_read(ec, bytes);
        });
}
```

Chat Server

```
void websocket_session::
on_read(error_code ec, std::size_t)
{
    if(ec)
        return fail(ec, "read");

    // Send to all connections
    state_->send(
        beast::buffers_to_string(buffer_.data()));

    // Clear the buffer
    buffer_.consume(buffer_.size());

    ws_.async_read(buffer_, [sp = shared_from_this()](
            error_code ec, std::size_t bytes)
        {
            sp->on_read(ec, bytes);
        });
}
```

Chat Server

```
void websocket_session::
send(std::shared_ptr<std::string const> const& ss)
{
    // Always add to queue
    queue_.push_back(ss);

    // Are we already writing?
    if(queue_.size() > 1)
        return;

    ws_.async_write(
        net::buffer(*queue_.front()),
        [sp = shared_from_this()](
            error_code ec, std::size_t bytes)
        {
            sp->on_write(ec, bytes);
        });
}
```

Chat Server

```
void websocket_session::
on_write(error_code ec, std::size_t)
{
    if(ec)
        return fail(ec, "write");

    queue_.erase(queue_.begin());

    if(! queue_.empty())
        ws_.async_write(
            net::buffer(*queue_.front()),
            [sp = shared_from_this()](
                error_code ec, std::size_t bytes)
            {
                sp->on_write(ec, bytes);
            });
}
```

Chat Server

```
void
websocket_session::
fail(error_code ec, char const* what)
{
   // Don't report these
   if(ec == net::error::operation_aborted ||
       ec == websocket::error::closed)
      return;

   std::cerr << what << ": " <<
      ec.message() << "\n";
}
```

Chat Server

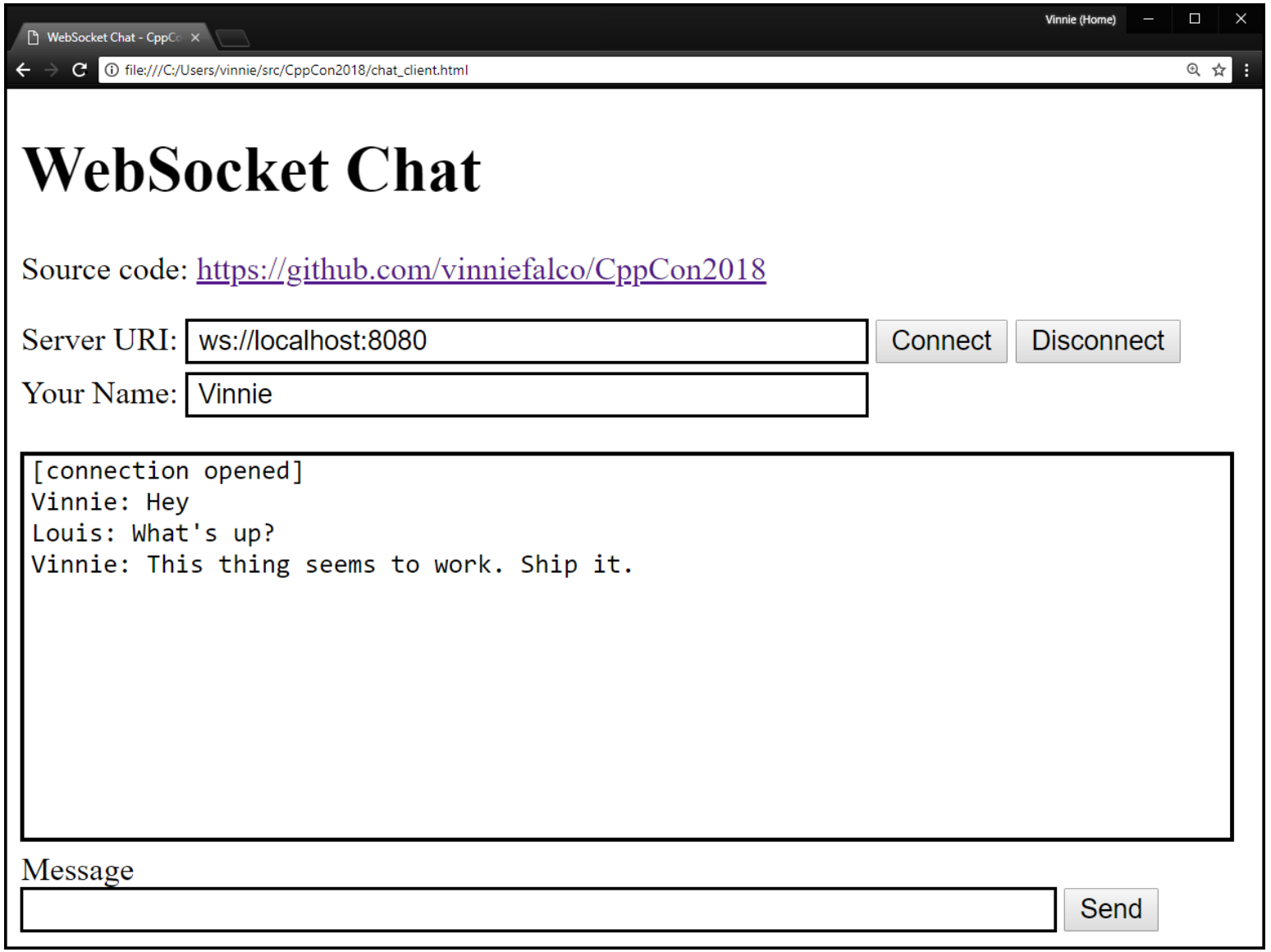
```
websocket_session::
~websocket_session()
{
    state_->leave(*this);
}
```

Web Server

- See if the socket was closed
- Check for WebSocket Upgrade request
- Calculate HTTP response
- Send HTTP response
- Read the next request...
- ...or close the connection.

Chat Client

Chat Client


WebSocket Chat - CppCo
Vinnie (Home)
file:///C:/Users/vinnie/src/CppCon2018/chat_client.html
WebSocket Chat
Source code: https://github.com/vinniefalco/CppCon2018
Server URI: ws://localhost:8080
Connect
Disconnect
Your Name: Vinnie
[connection opened]
Vinnie: Hey
Louis: What's up?
Vinnie: This thing seems to work. Ship it.
Message
Send

Chat Client

```
<html>
<head>
<title>WebSocket Chat - CppCon2018</title>
</head>
<body>
<h1>WebSocket Chat</h1>
<!-- UI and app -->
</body>
</html>
```

Chat Client

```
Server URI:
<input
   id="uri"
   size="47"
   class="draw-border"
   style="margin-bottom: 5px;"
   value="ws://localhost:8080">
```

Server URI: ws://localhost:8080

Chat Client

```
<button
   id="connect"
   class="echo-button">
Connect
</button>
```

```
<button
   id="disconnect"
   class="echo-button">
Disconnect
</button>
```

Chat Client

```
Your Name:
<input
   id="userName"
   class="draw-border"
   size=47>
```

Your Name: 

Chat Client

```
<pre
  id="messages"
  style="border: solid 1px #cccccc;">
</pre>
```

Chat Client

```
Message<br>
<input
   id="sendMessage"
   class="draw-border">
<button
   id="send"
   class="echo-button">Send</button>
```

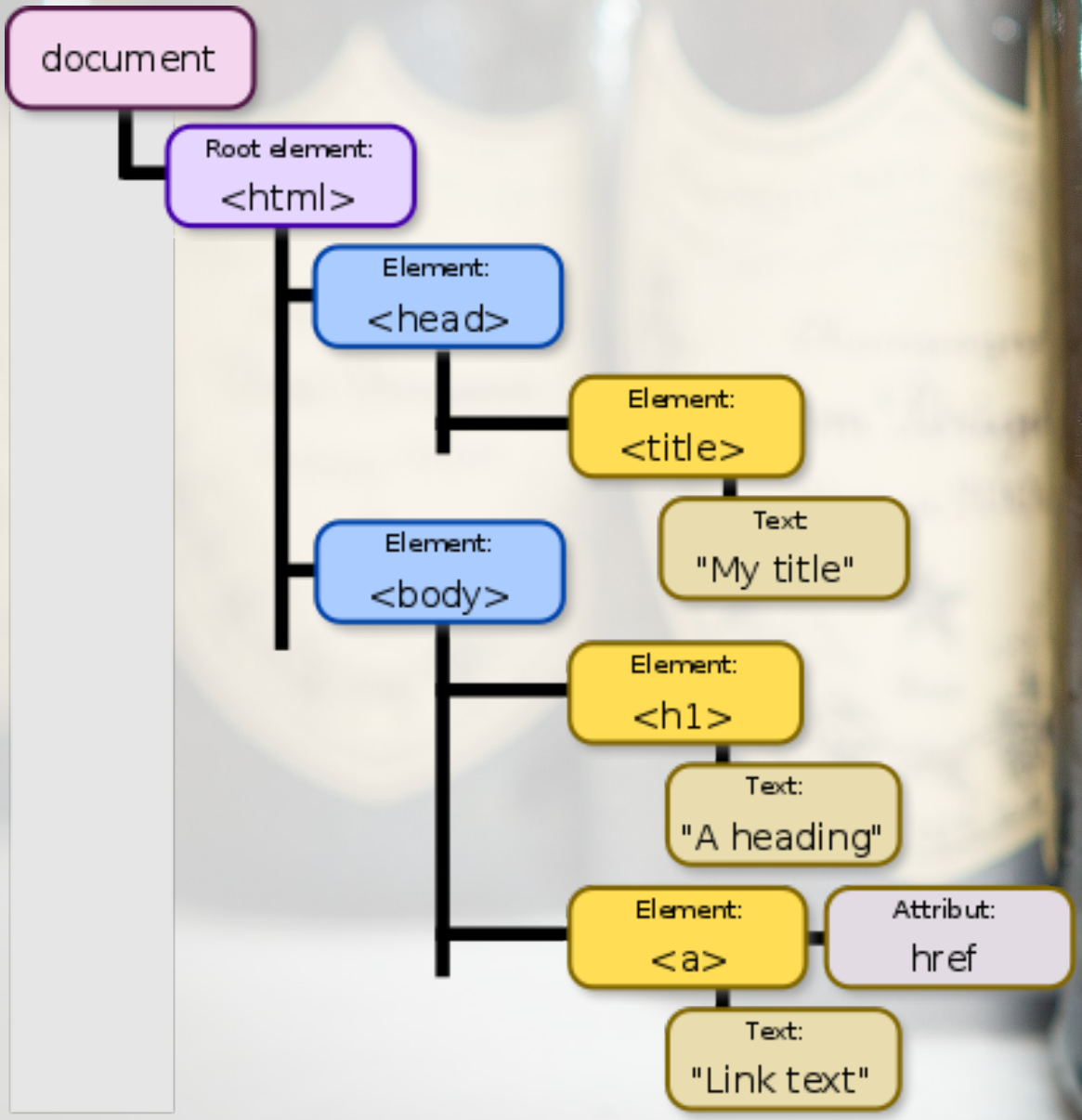
Message

Send


Dom PERIGNON · MILLÉSIME
Altum villare
Champagne
Dom Pérignon
Vintage 2004
Sparkling Wine
Vin Mousseux
Brut
750ml
12.5% alc./vol.

DOM

Document Object Model

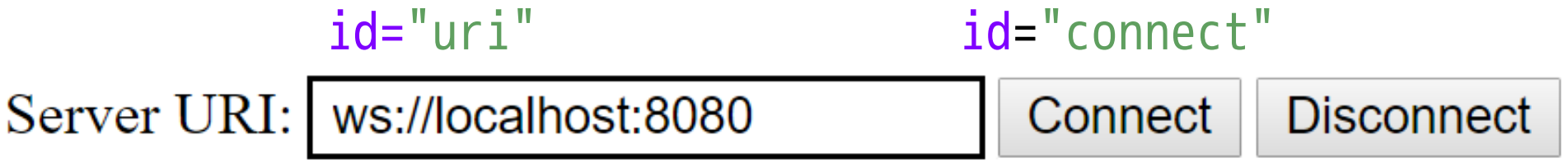


Chat Client

```
<script>
   var ws = null;
   connect.onclick = function() {
      ws = new WebSocket(uri.value);

      ...

   };
</script>
```



Chat Client

```
connect.onclick = function() {
   ...

   ws.onopen = function(ev) {
      messages.innerText +=
         "[connection opened]\n";
   };

   ws.onclose = function(ev) {
      messages.innerText +=
         "[connection closed]\n";
   };
```

Chat Client

```
connect.onclick = function() {
   ...

   ws.onmessage = function(ev) {
      messages.innerText += ev.data + "\n";
   };

   ws.onerror = function(ev) {
      messages.innerText += "[error]\n";
      console.log(ev);
   };
```

Chat Client

```
<script>
   var ws = null;
   connect.onclick = function() {
      ws = new WebSocket(uri.value);
      ...
   };
</script>
```

Education

Summary

```
// Code and slides from the talk
https://github.com/vinniefalco/CppCon2018

// Boost.Beast library
https://github.com/boostorg/beast

// List of Boost libraries, including Boost.Asio
https://www.boost.org/doc/libs/
```

Speaker's Dinner