

Project 3: HTTP Server & CGI

NP TA 柏維

Important Dates

- **Deadline: 5/15 (Wed.) 23:55**
- **Demo: 5/18 (Sat.) 9:20 - 18:20**
 - We will announce demo slots 2 - 3 days before
 - Tasks:
 - Correct format and compile.
 - QA.
 - Pass test cases.
 - Implement 1 extra function with limit time.

Info

- You are **HIGHLY** encouraged to publish your questions on Project3 討論區
 - Check the spec and other questions first.
- You can contact TAs by E3. (Mails sent to other addresses will NOT be replied)
- TA hours (Tuesday: 15:00 - 17:00) on **4/30**, **5/7** will be held at online.
- You **MUST** make a reservation by email in advance.
- TAs will **NOT** debug for you.

NP Servers

- **Your projects will be run on NP servers during demo**
- NP projects should run on NP servers
- Any abuse of NP servers will be recorded
- Do not leave any zombie process in the system

Requirements

- This project is divided into 2 parts.
 - Part 1 should run on **NP Linux Workstation**.
 - Part 2 should run on **Windows 10**.
- The requirements in both parts are roughly the same with slightly different conditions.

Requirements

- In part 1, you are asked to implement
 - 1-1: A simple HTTP server called **http_server**
 - 1-2: A CGI program called **console.cgi**

NP Project 3 Sample Console

npnlinux12.cs.nycu.edu.tw:12345/console.cgi?h0=npnlinux12.cs.nycu.edu.tw&p0=12344&f0=t1.txt&h1=npnlinux12.cs.nycu.edu.tw&p1=12343&f1=t2.txt&h2=&p2=&f2=&h3=&p3=&f3=&h4=&p4=&f4=

npnlinux12.cs.nycu.edu.tw:12344

```
*****
** Welcome to the information server. **
*****
*** User '(no name)' entered from 140.113.235.43012. ***
% ls
bin
np_single_golden
test.html
% exit
```

npnlinux12.cs.nycu.edu.tw:12343

```
*****
** Welcome to the information server. **
*****
*** User '(no name)' entered from 140.113.235.56710. ***
% ls |1
% number
  1 bin
  2 np_single_golden
  3 test.html
% exit
```

Requirements

- In part 2, you are asked to implement
 - A special HTTP server called **cgi_server**

```
NP Project 3 Sample Console x +  
← → ↻ 不安全 nplinux12.cs.nycu.edu.tw:12345/console.cgi?h0=nplinux12.cs.nycu.edu.tw&p0=12344&f0=t1.txt&h1=nplinux12.cs.nycu.edu.tw&p1=12343&f1=t2.txt&h2=&p2=&f2=&h3=&p3=&f3=&h4=&p4=&f4=  
  
nplinux12.cs.nycu.edu.tw:12344  
*****  
** Welcome to the information server. **  
*****  
*** User '(no name)' entered from 140.113.235.235:43012. ***  
% ls  
bin  
np_single_golden  
test.html  
% exit  
  
nplinux12.cs.nycu.edu.tw:12343  
*****  
** Welcome to the information server. **  
*****  
*** User '(no name)' entered from 140.113.235.235:56710. ***  
% ls |1  
% number  
1 bin  
2 np_single_golden  
3 test.html  
% exit
```

Scenario 1: on NP Linux Workstation

Run HTTP Server

client



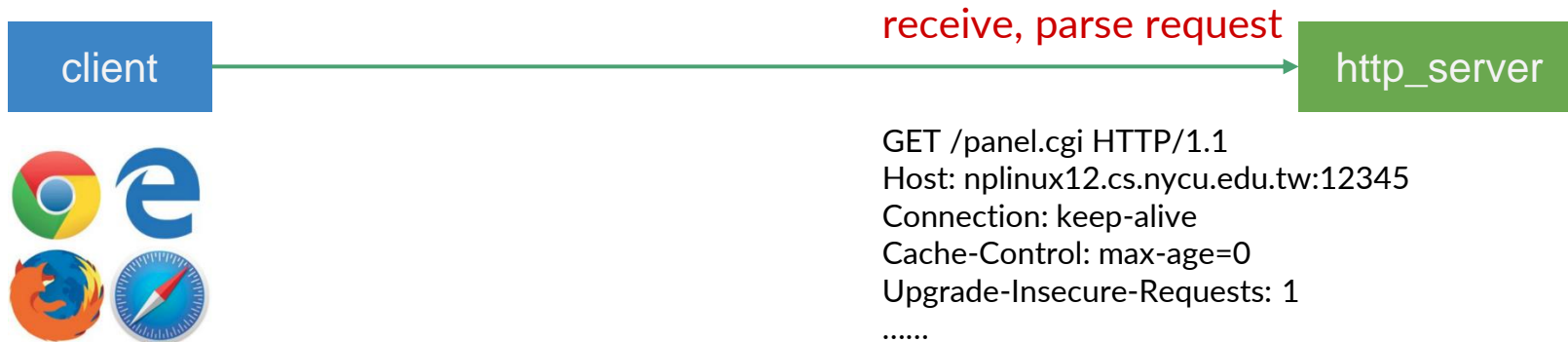
http_server

nplinux12
port: 12345

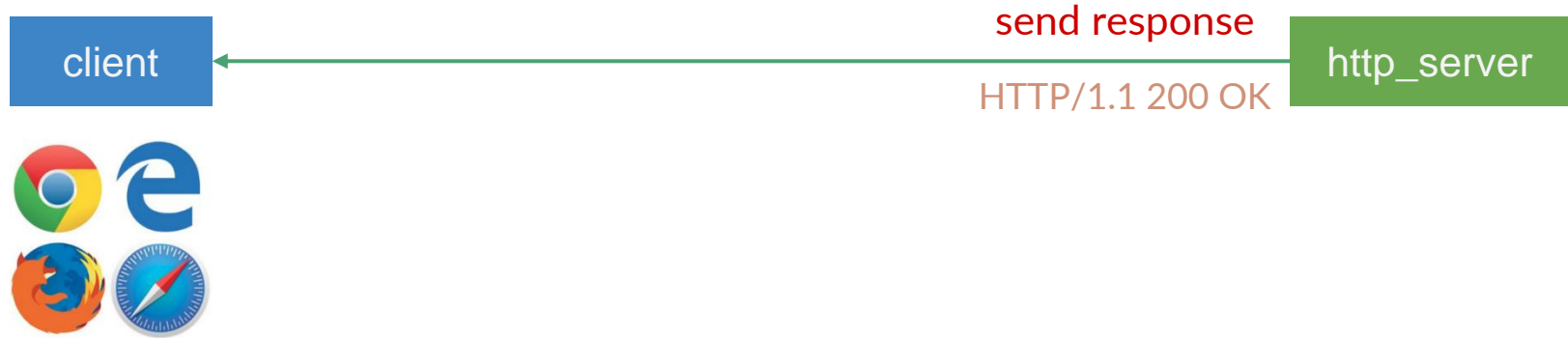
Request 1 - Send



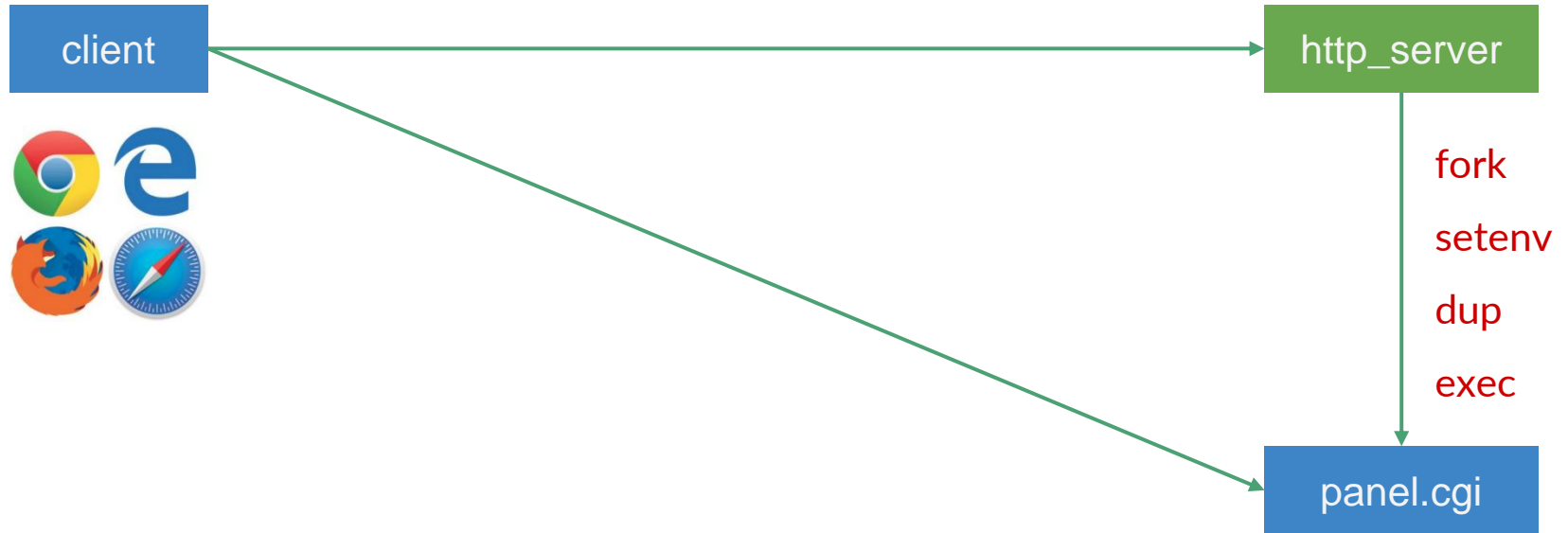
Request 1 - Receive



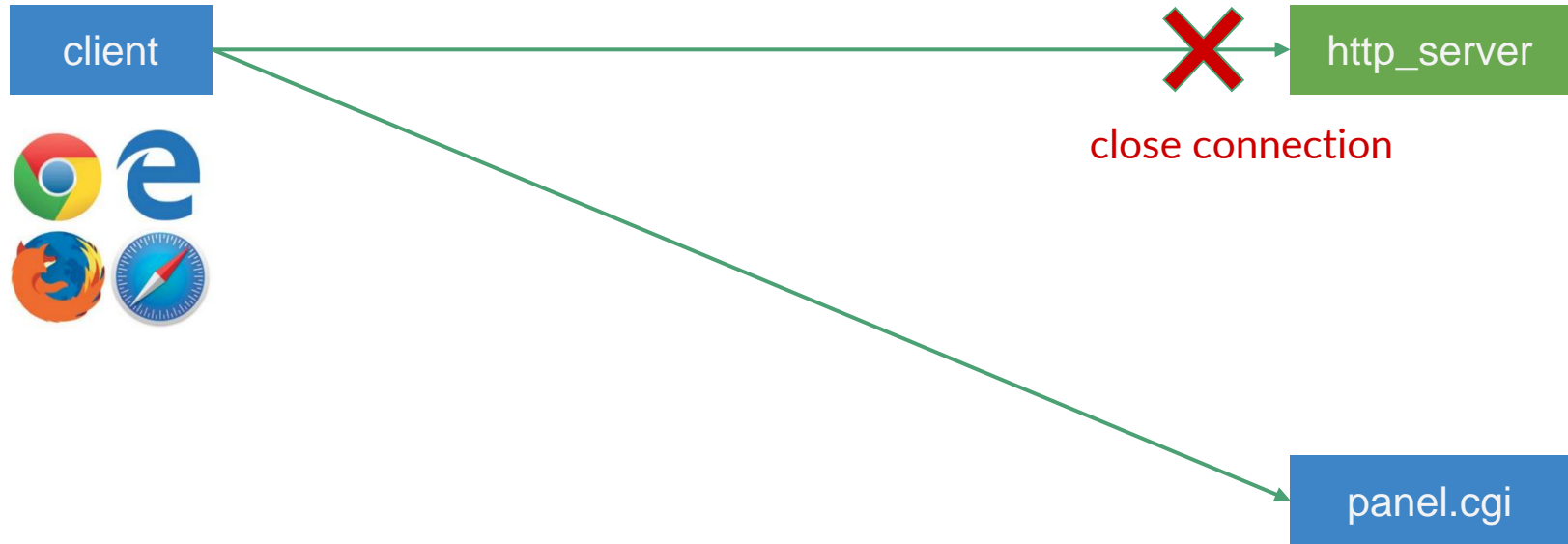
Request 2 - Close Connection



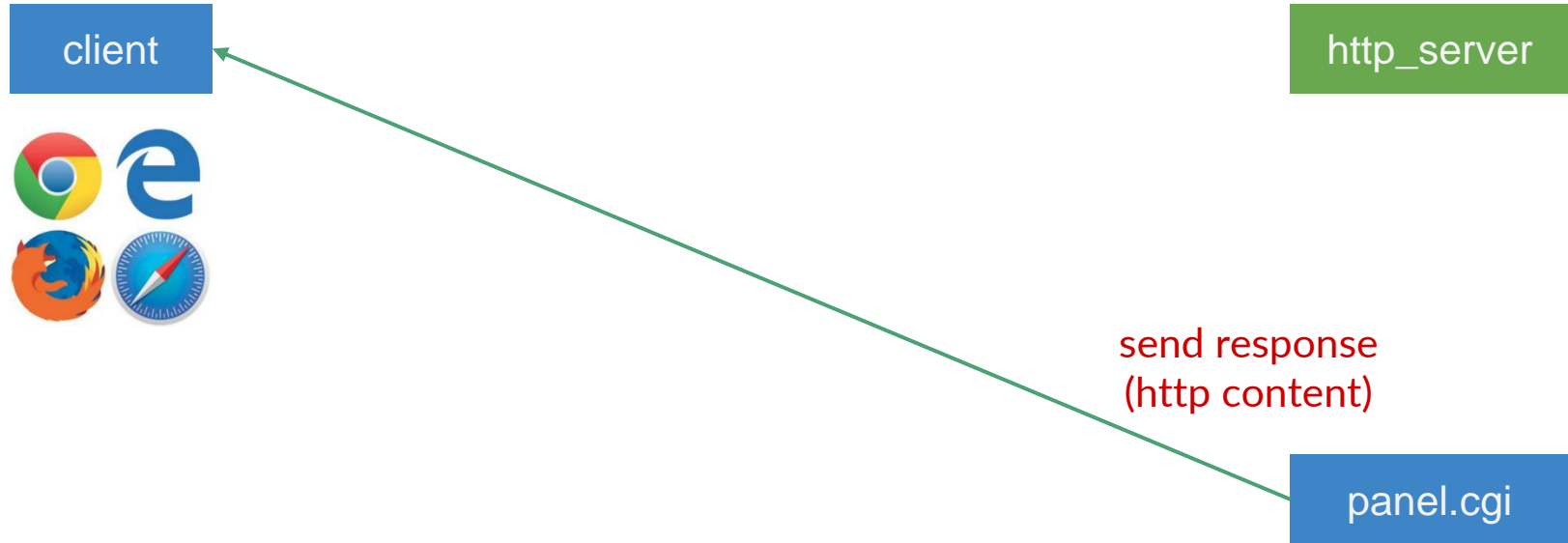
Request 1 - Execute CGI Program



Request 1 - Close Connection



Request 1 - Response



Request 1 - Terminate

client













http_server

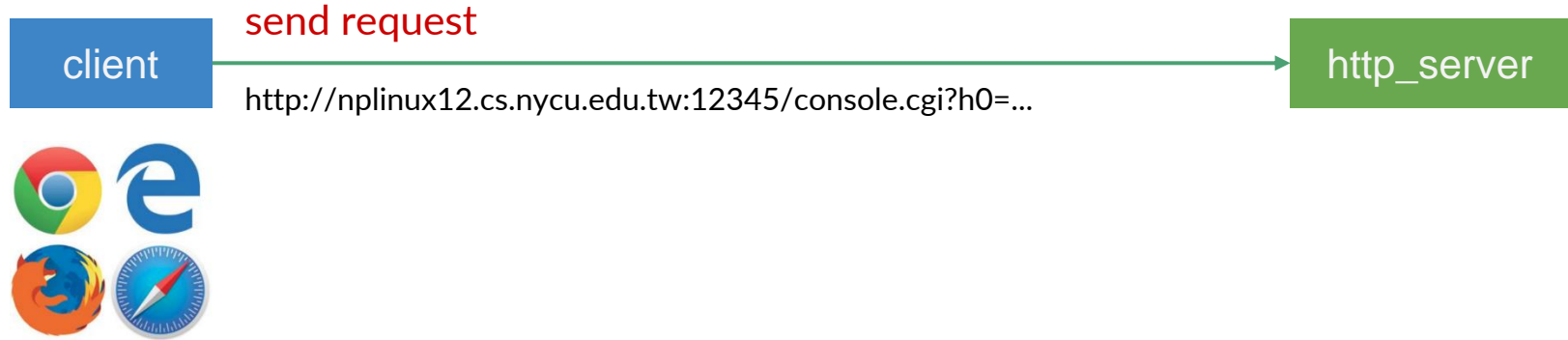
child exit

panel.cgi

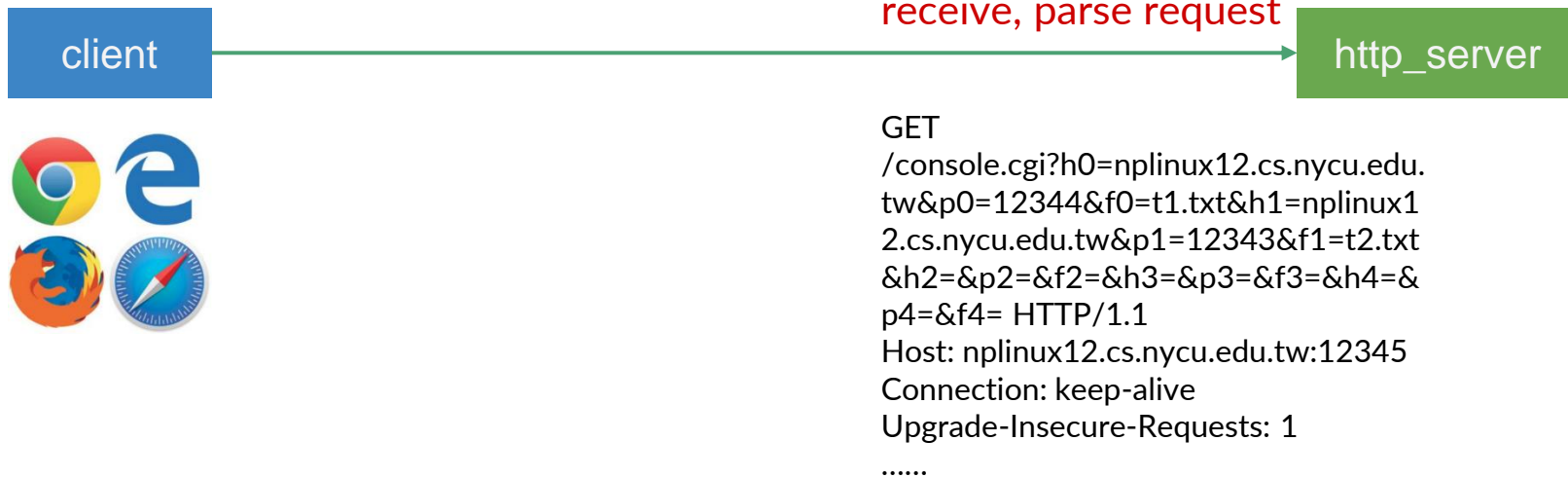
Page Generated by panel.cgi

#	Host	Port	Input File	
Session 1	<input type="text" value="nplinux12"/> 	<input type="text" value=".cs.nycu.edu.tw"/>	<input type="text" value="12344"/>	<input type="text" value="t1.txt"/> 
Session 2	<input type="text" value="nplinux12"/> 	<input type="text" value=".cs.nycu.edu.tw"/>	<input type="text" value="12343"/>	<input type="text" value="t2.txt"/> 
Session 3	<input type="text"/> 	<input type="text" value=".cs.nycu.edu.tw"/>	<input type="text"/>	<input type="text"/> 
Session 4	<input type="text"/> 	<input type="text" value=".cs.nycu.edu.tw"/>	<input type="text"/>	<input type="text"/> 
Session 5	<input type="text"/> 	<input type="text" value=".cs.nycu.edu.tw"/>	<input type="text"/>	<input type="text"/> 
<input type="button" value="Run"/>				

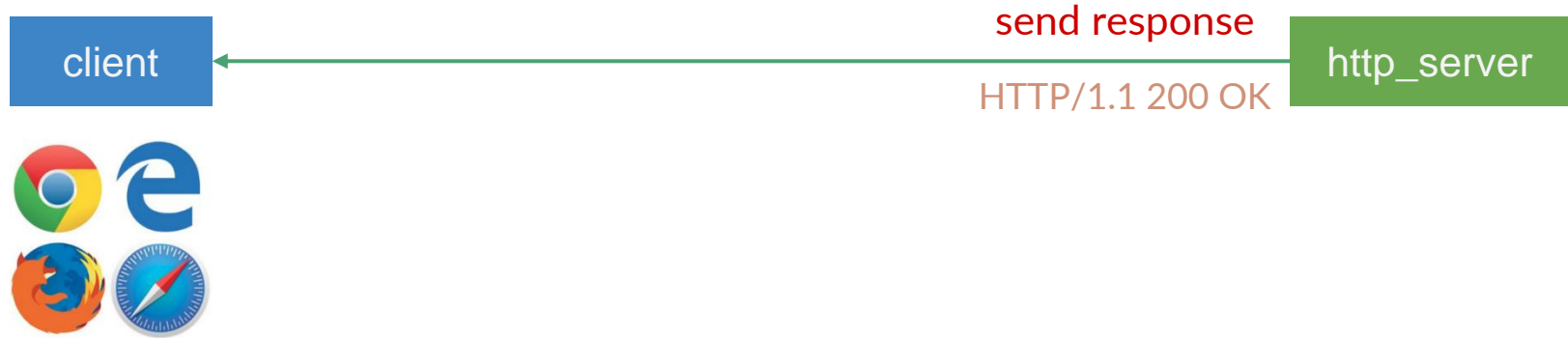
Request 2 - Send



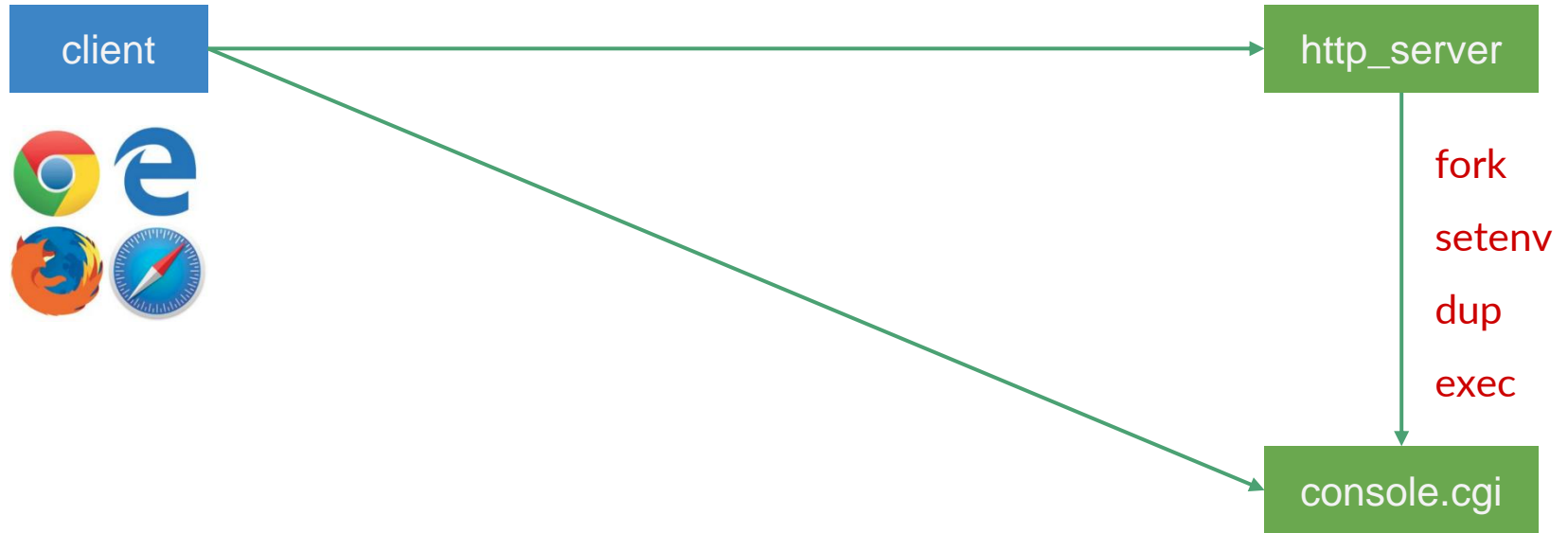
Request 2 - Receive



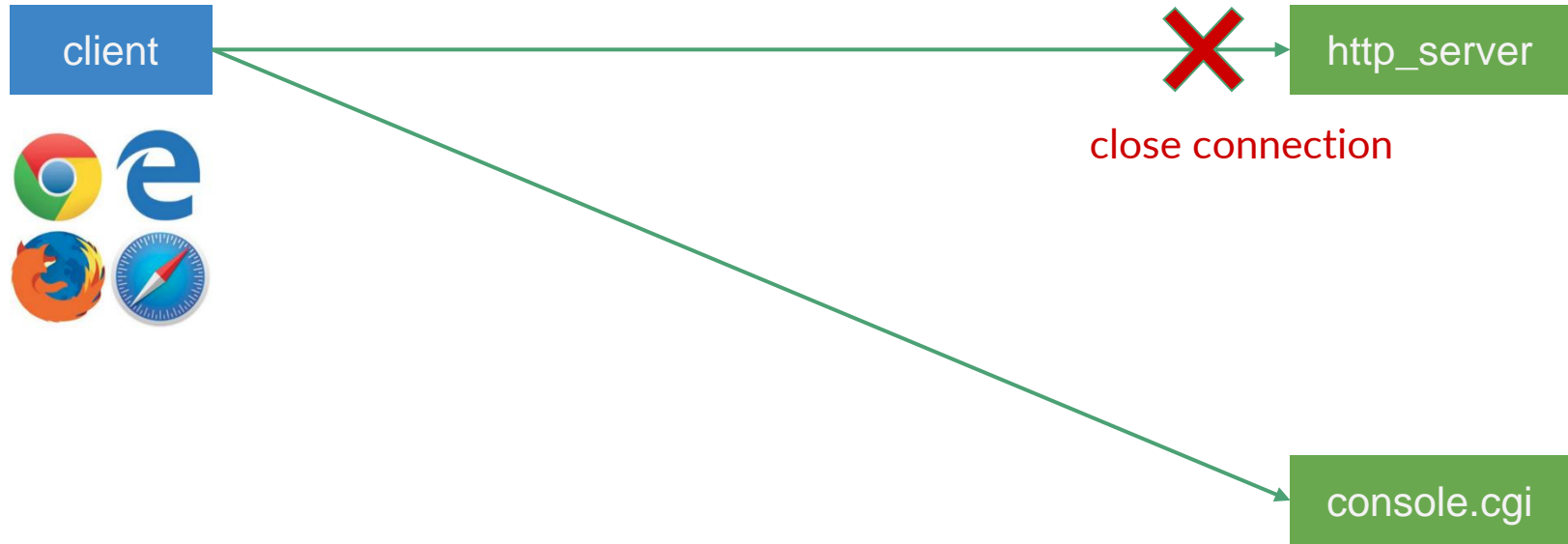
Request 2 - Close Connection



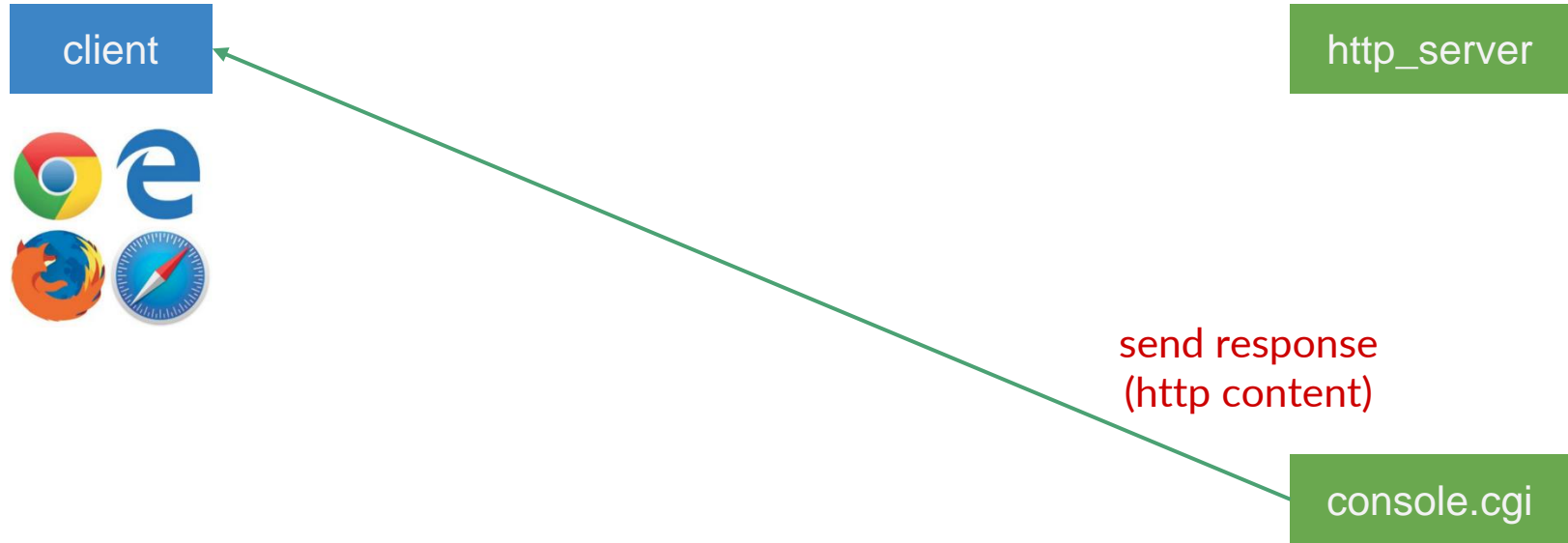
Request 2 - Execute CGI Program



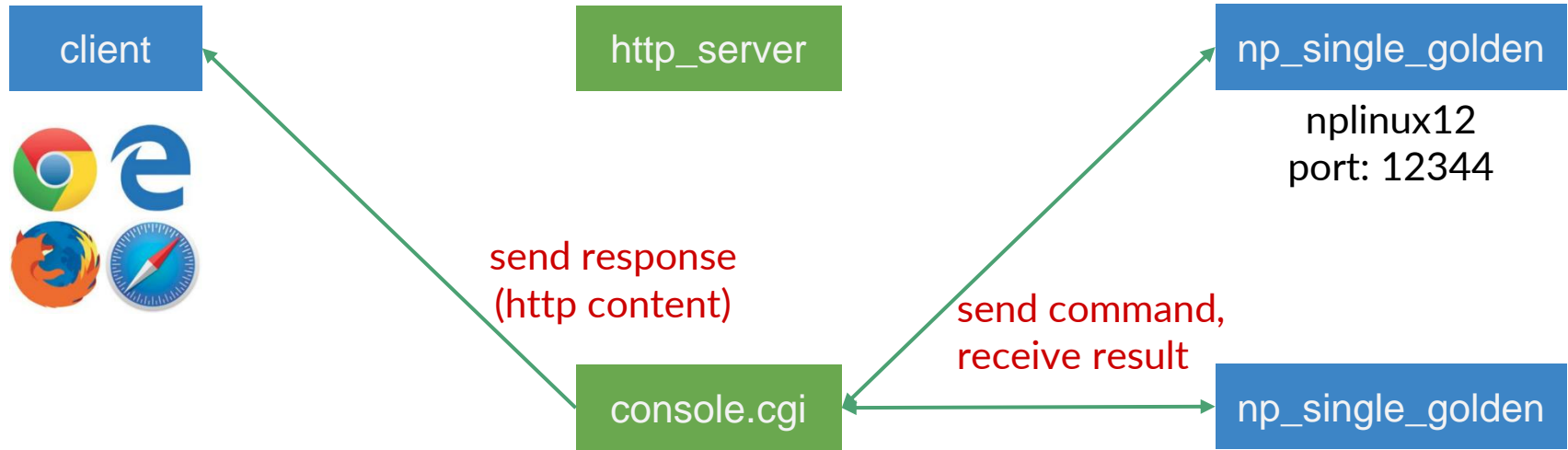
Request 2 - Close Connection



Request 2 - Response



Request 2 - Connect Servers



QUERY_STRING=
h0=nplinux12.cs.nycu.edu.tw&p0=12344&f0=t1.txt&
h1=nplinux12.cs.nycu.edu.tw&p1=12343&f1=t2.txt&
h2=&p2=&f2=&h3=&p3=&f3=&h4=&p4=&f4=

Page Generated by console.cgi

- Display hostname and port of the remote server in each session
- Make sure the commands are displayed in the right order, at the right time, and can be easily distinguished from the outputs

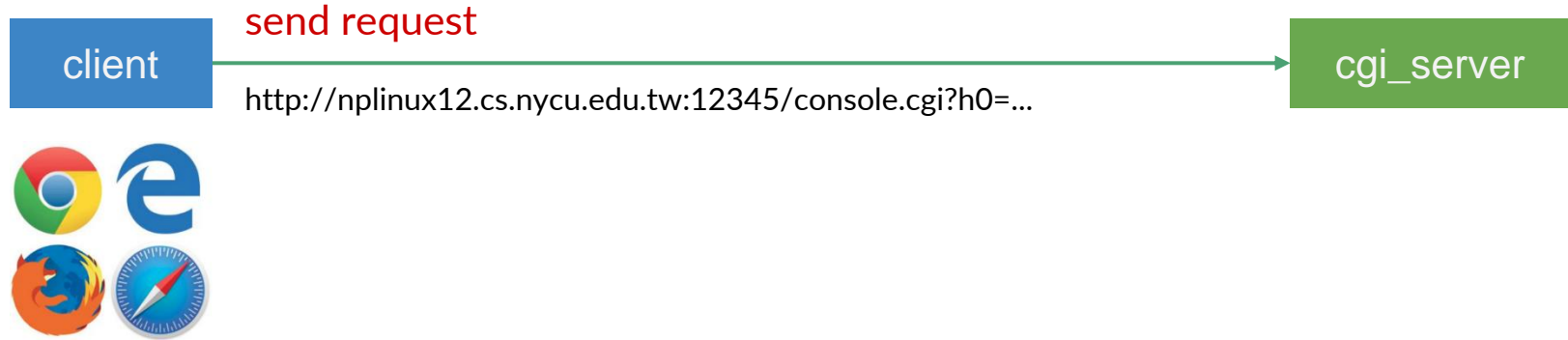
NP Project 3 Sample Console

nplinux12.cs.nycu.edu.tw:12345/console.cgi?h0=nplinux12.cs.nycu.edu.tw&p0=12344&f0=t1.txt&h1=nplinux12.cs.nycu.edu.tw&p1=12343&f1=t2.txt&h2=&p2=&f2=&h3=&p3=&f3=&h4=&p4=&f4=

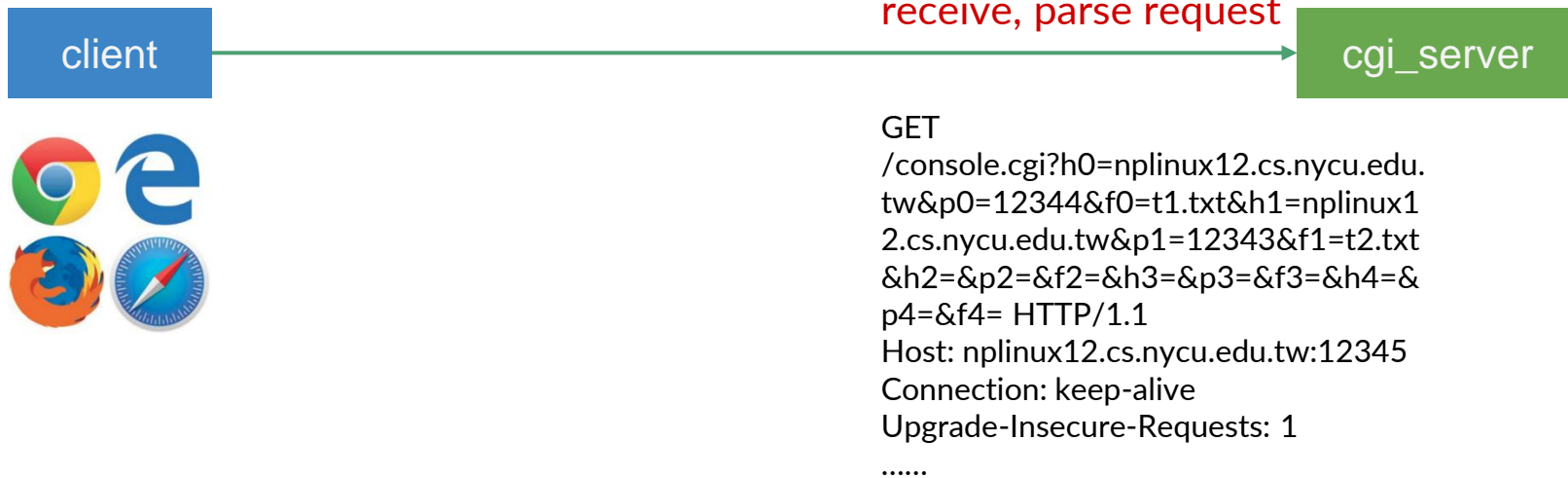
nplinux12.cs.nycu.edu.tw:12344	nplinux12.cs.nycu.edu.tw:12343
***** ** Welcome to the information server. ** ***** *** User '(no name)' entered from 140.113.235.43012. *** % ls bin np_single_golden test.html % exit	***** ** Welcome to the information server. ** ***** *** User '(no name)' entered from 140.113.235.56710. *** % ls 1 % number 1 bin 2 np_single_golden 3 test.html % exit

Scenario 2: on Windows

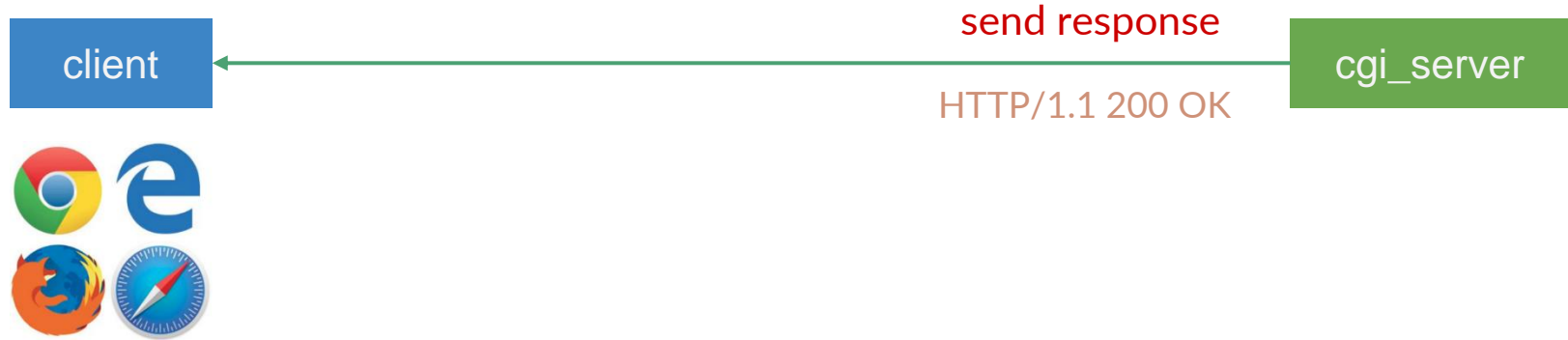
Request 2 - Send



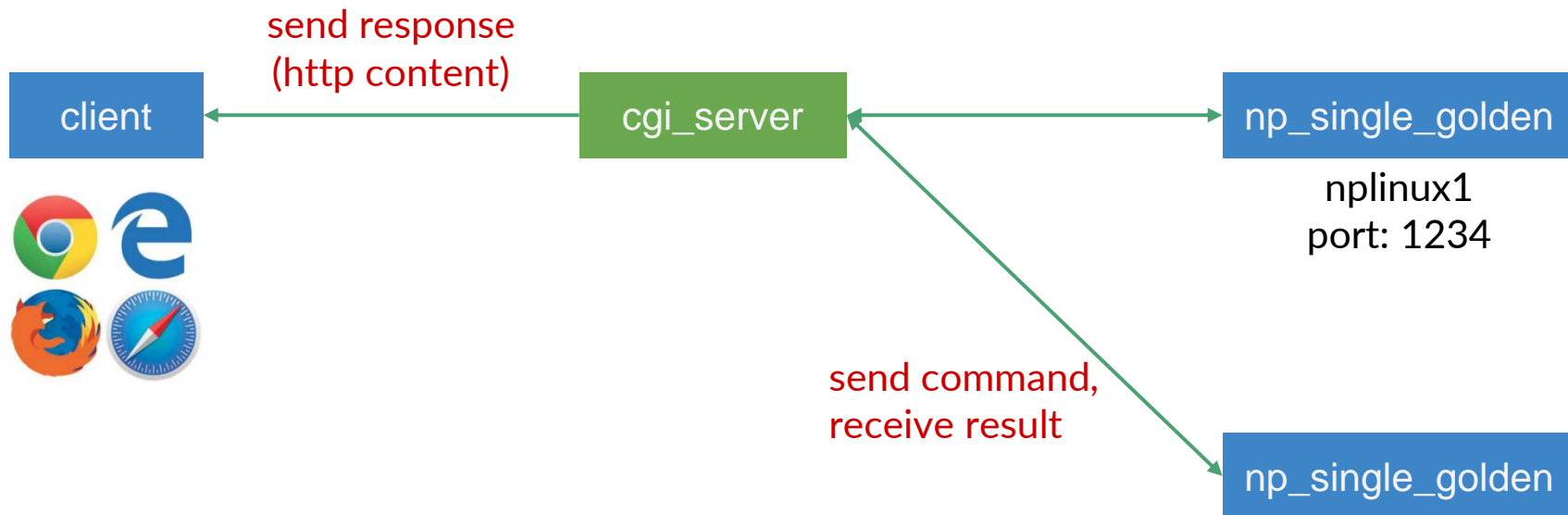
Request 2 - Receive



Request 2 - Response



Request 2 - Connect Servers



QUERY_STRING=
h0=nplinux1.cs.nycu.edu.tw&p0=1234&f0=t1.txt&
h1=nplinux2.cs.nycu.edu.tw&p1=5678&f1=t2.txt&
h2=&p2=&f2=&h3=&p3=&f3=&h4=&p4=&f4=

Reminder

Extra Files

- You should check the files in extra_files.zip first
 - Boost.Asio Example
 - CGI programs
 - np_single_golden
 - etc.

Supplementary

Outline

- Lambda Expressions
- Auto Specifier
- Shared Pointer
- Move
- Boost.Asio Example
- Printing HTML Content in C++

Lambda Expressions (since C++11)

- An unnamed function object capable of capturing variables in scope
- A lambda expression consists of three parts

`[]()``{ }`

- captures
- params
- body

```
/* without capture */  
function<int(int)> square = [] (int x) { return x * x; };  
cout << square(5) << endl; /* output: 25 */
```

Lambda Expressions (since C++11)

- An unnamed function object capable of capturing variables in scope

```
/* capture by reference */
```

```
int x = 0;
```

```
function<int(int)> add = [&x](int y) { x = 1; return x + y; };
```

```
cout << add(3) << endl; /* output: 4 */
```

```
cout << x << endl;      /* output: 1 */
```

```
/* capture by value */
```

```
int x = 0;
```

```
function<int(int)> add = [x](int y) mutable { x = 1; return x + y; };
```

```
cout << add(3) << endl; /* output: 4 */
```

```
cout << x << endl;      /* output: 0 */
```

Lambda Expressions (since C++11)

- Without lambda expression

```
bool by_name(Person a, Person b) {  
    return a.name < b.name;  
}  
  
bool by_age(Person a, Person b) {  
    return a.age < b.age;  
}  
  
vector<Person> employees;  
  
/* sort employees ordered by name */  
sort(employees.begin(), employees.end(), by_name);  
  
/* sort employees ordered by age */  
sort(employees.begin(), employees.end(), by_age);
```

Lambda Expressions (since C++11)

- With lambda expression

```
vector<Person> employees;
```

```
/* sort employees ordered by name */
```

```
sort(employees.begin(), employees.end(), [](Person a, Person b) {  
    return a.name < b.name;  
});
```

```
/* sort employees ordered by age */
```

```
sort(employees.begin(), employees.end(), [](Person a, Person b) {  
    return a.age < b.age;  
});
```

Auto Specifier (since C++11)

- Let compiler automatically deduce types

```
// int  
auto a = 1 + 2;
```

```
// int  
auto b = a;
```

```
/* function<int(int)> */  
auto square = [](int x) { return x * x; };
```

```
vector<int> arr;  
/* vector<int>::iterator */  
auto begin_it = arr.begin();
```

Shared Pointer (since C++11)

- **std::shared_ptr** is a smart pointer that retains shared ownership of an object through a pointer
- You do not have to **free** and **delete** manually

```
std::shared_ptr<int> sp(new int);  
*sp = 5;  
cout << *sp;    /* output: 5 */
```

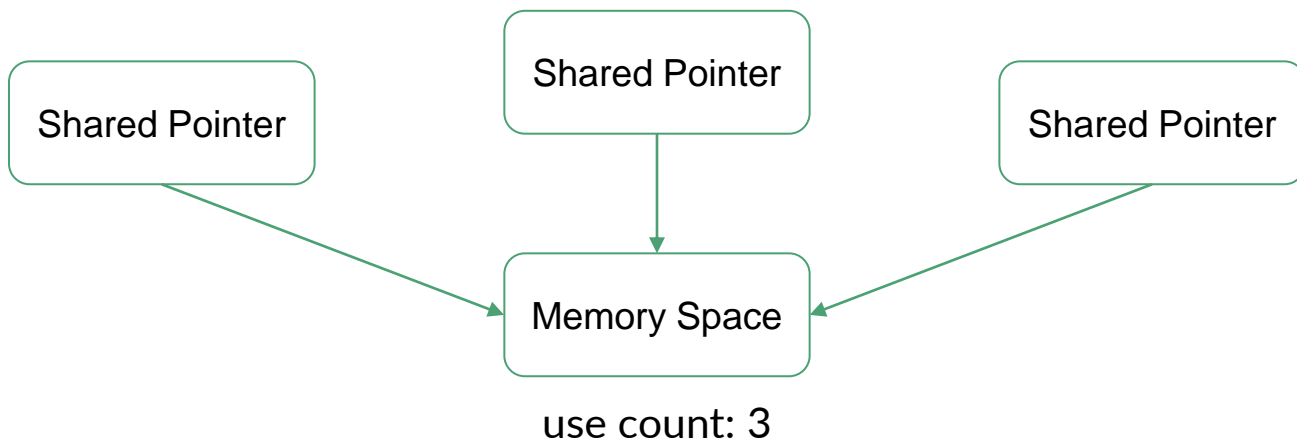
```
auto sp2 = std::make_shared<int>(10);  
cout << *sp2;    /* output: 10 */
```

C++ smart pointers

1. std::unique_ptr
2. std::shared_ptr
3. std::weak_ptr

Shared Pointer (since C++11)

- When will the allocated resource be destroyed?
 - When the last remaining `shared_ptr` owning the object is destroyed (when use count is 0)



Shared Pointer (since C++11)

- When will the allocated resource be destroyed?
 - When the last remaining `shared_ptr` owning the object is destroyed (when use count is 0)

```
{
    std::shared_ptr<int> sp(new int);
    {
        std::shared_ptr<int> sp2(sp);
        cout << sp.use_count(); /* output: 2 */
    }
    cout << sp.use_count();      /* output: 1 */
} /* free the space */
```

enable_shared_from_this

- Allows an object that is currently managed by a `shared_ptr` **safely generate additional `shared_ptr` instances**

```
class MyClass : std::enable_shared_from_this<MyClass>
{
    std::shared_ptr<MyClass> get_ptr() {
        return shared_from_this(); // Good
        return this;                // Bad
    }
};
```

Move (since C++11)

- **std::move** is used to indicate that an object may be "moved from", i.e. allowing the efficient transfer of resources from one object to another.

```
string a = "Hello";
```

```
/* extra cost of copying string a */
```

```
string b = a;
```

```
/* the content of string a will be moved into string c */
```

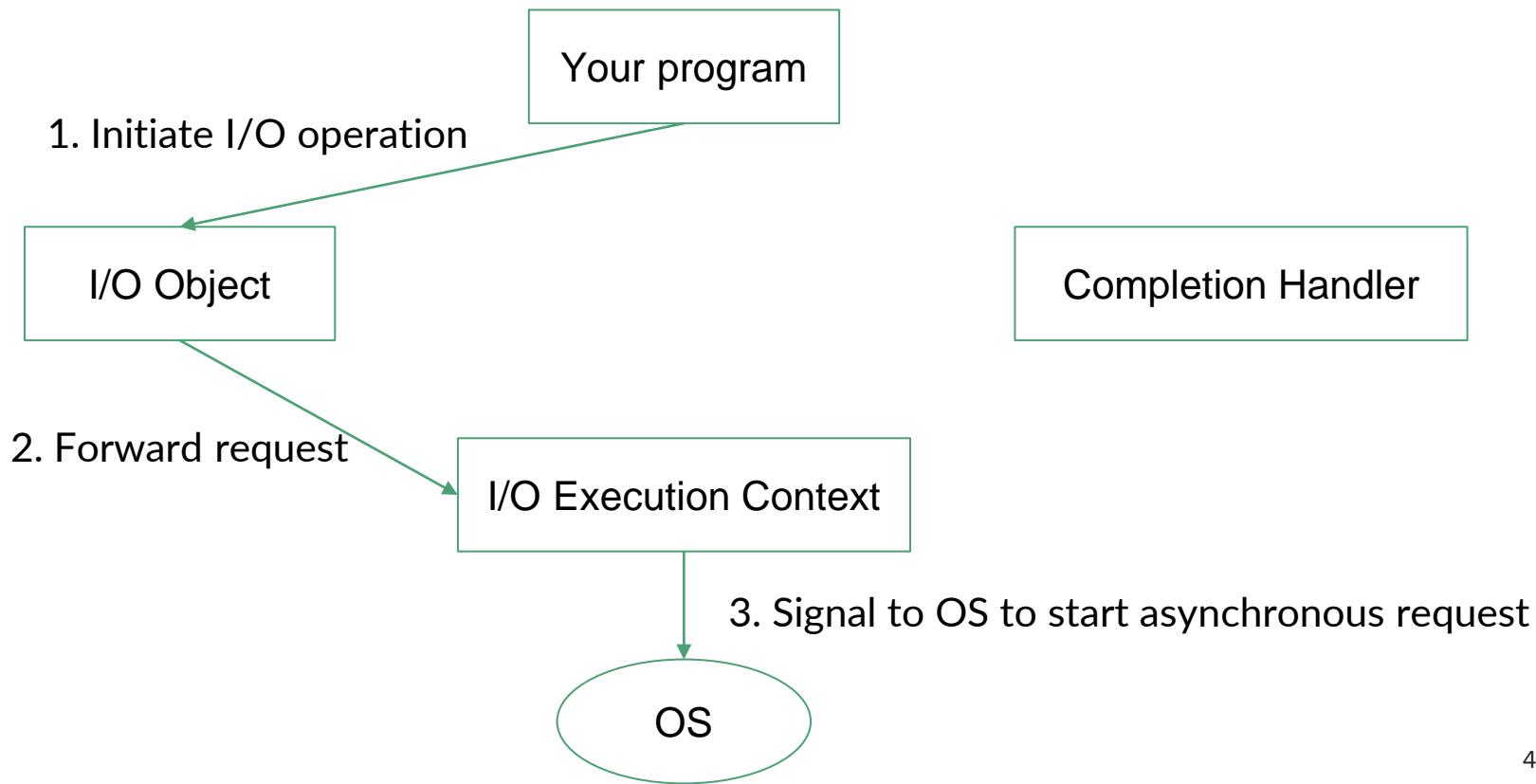
```
string c = move(a);
```

```
cout << "'" << a << "'" << endl; // output: ""
```

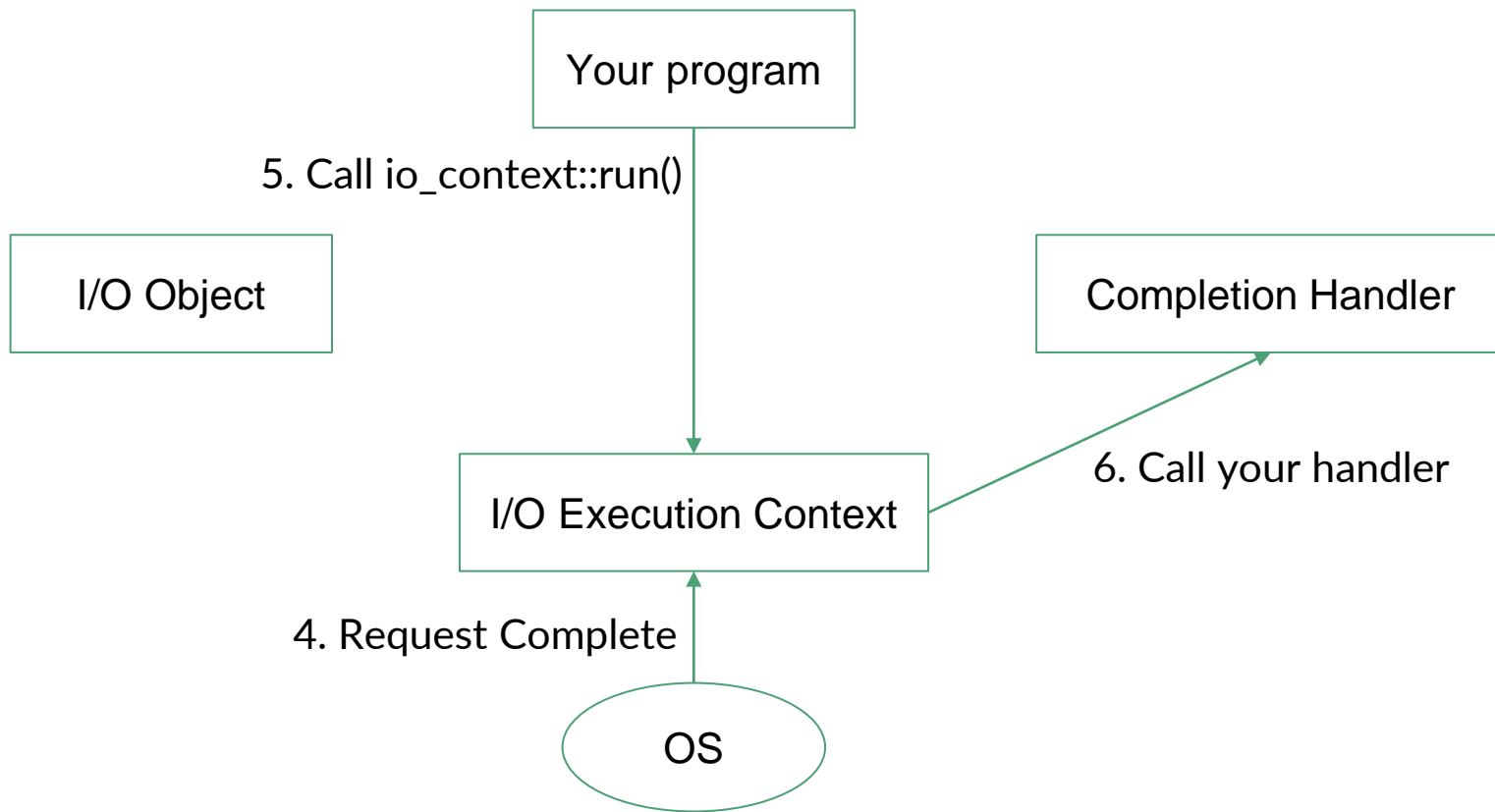
```
cout << "'" << b << "'" << endl; // output: "Hello"
```

```
cout << "'" << c << "'" << endl; // output: "Hello"
```

Boost.Asio Overview



Boost.Asio Overview



Boost.Asio Example

- A **c++11** asynchronous echo server example from Boost.Asio documentation
- The following codes are simplified because of space limitation

Boost.Asio Example

- Call `io_context::run()`

```
int main(int argc, char* argv[])
{
    boost::asio::io_context io_context;
    server s(io_context, std::atoi(argv[1]));

    /* VERY IMPORTANT! */
    io_context.run();

    return 0;
}
```


Boost.Asio Example

```
class server {  
private:  
    tcp::acceptor acceptor_;  
  
public:  
    server(boost::asio::io_context& io_context, short port)  
        : acceptor_(io_context, tcp::endpoint(tcp::v4(), port)) {  
        do_accept();  
    }  
  
    .  
    .  
    .  
  
};
```

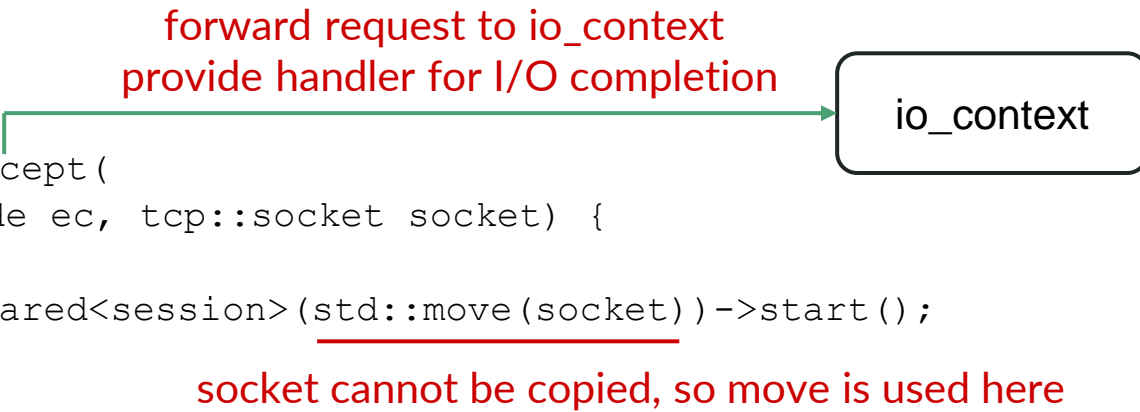
Boost.Asio Example

```
void do_accept() {  
    acceptor_.async_accept(  
        [this](error_code ec, tcp::socket socket) {  
            if (!ec) {  
                std::make_shared<session>(std::move(socket))->start();  
            }  
            do_accept();  
        });  
}
```

forward request to io_context
provide handler for I/O completion

io_context

socket cannot be copied, so move is used here



Boost.Asio Example

```
class session : public std::enable_shared_from_this<session> {  
private:
```

```
    tcp::socket socket_;  
    enum { max_length = 1024 };  
    char data_[max_length];
```

```
public:
```

```
    session(tcp::socket socket) : socket_(std::move(socket)) {}
```

```
    void start() { do_read(); }      socket cannot be copied, so move is used here
```

```
.  
.   
.
```

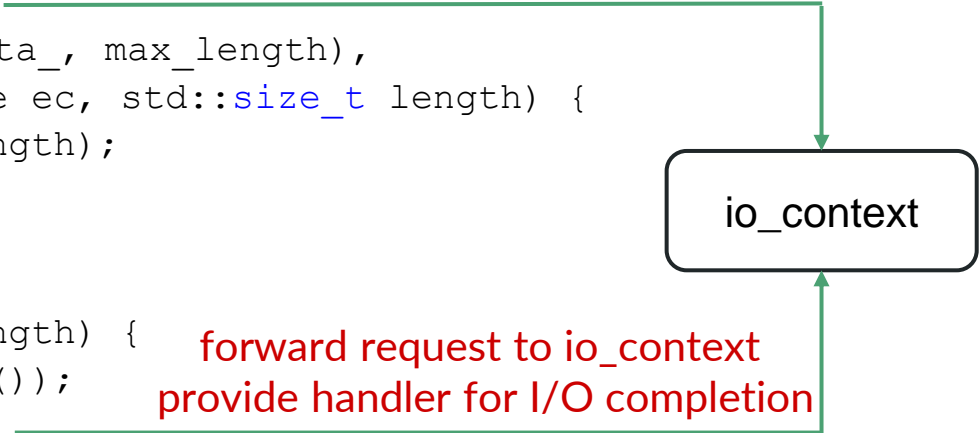
Boost.Asio Example

```
void do_read() {  
    auto self(shared_from_this());  
    socket_.async_read_some(  
        boost::asio::buffer(data_, max_length),  
        [this, self](error_code ec, std::size_t length) {  
            if (!ec) do_write(length);  
        });  
}  
  
void do_write(std::size_t length) {  
    auto self(shared_from_this());  
    boost::asio::async_write(  
        socket_, boost::asio::buffer(data_, length),  
        [this, self](error_code ec, std::size_t length) {  
            if (!ec) do_read();  
        });  
}
```

forward request to io_context
provide handler for I/O completion

io_context

forward request to io_context
provide handler for I/O completion



Printing HTML Content in C++

console.cpp

```
std::string html_content =  
    "<!DOCTYPE html>\n"  
    "<html lang='en'>\n"  
    "  <head>\n"  
    "    <meta charset='UTF-8' />\n"  
    "    <title>NP Project 3 Sample Console</title>\n"  
    "...  
    "</head>\n"  
    "<body>\n"  
    "...  
    "</body>\n"  
    "</html>";  
std::cout << html_content << std::flush;
```

Printing HTML Content in C++

`console.cpp`

```
std::cout <<  
    #include "console.html"  
<< std::flush;
```

`console.html`

```
R"NP(  
<!DOCTYPE html>  
<html lang="en">  
    <head>  
        <meta charset="UTF-8" />  
        <title>NP Project 3 Sample Console</title>  
        ...  
    </head>  
    <body>  
        ...  
    </body>  
</html>  
)NP"
```

Reference

- <https://en.cppreference.com/w/cpp/language/lambda>
- <https://en.cppreference.com/w/cpp/language/auto>
- https://en.cppreference.com/w/cpp/memory/shared_ptr
- https://en.cppreference.com/w/cpp/memory/enable_shared_from_this
- <https://en.cppreference.com/w/cpp/utility/move>
- https://www.boost.org/doc/libs/1_77_0/doc/html/boost_asio.html