Contemporary C++: Learning Modern C++ in a Modern Way

الماس فناوری ابری پاسارگاد- آلفا

مدرس: سعید امراللهی بیوکی

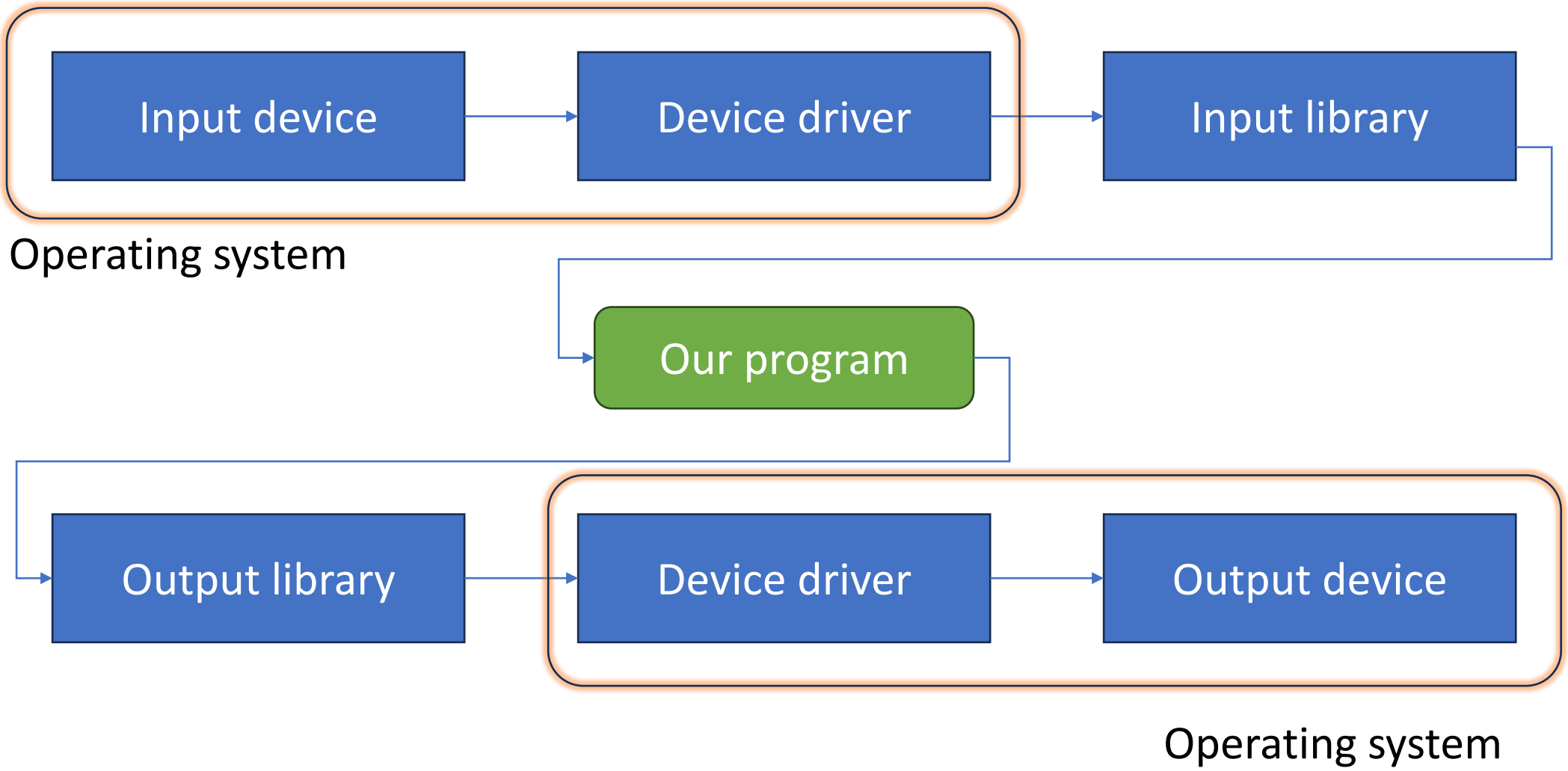
Agenda 22/24

Session 22. Input and Output Streams

- Input and Output
- The I/O Stream Model
- Files
- Opening and Closing files
- Reading and writing a file
- I/O error handling
- Working with files
- String streams
- Writing simple file processing programs

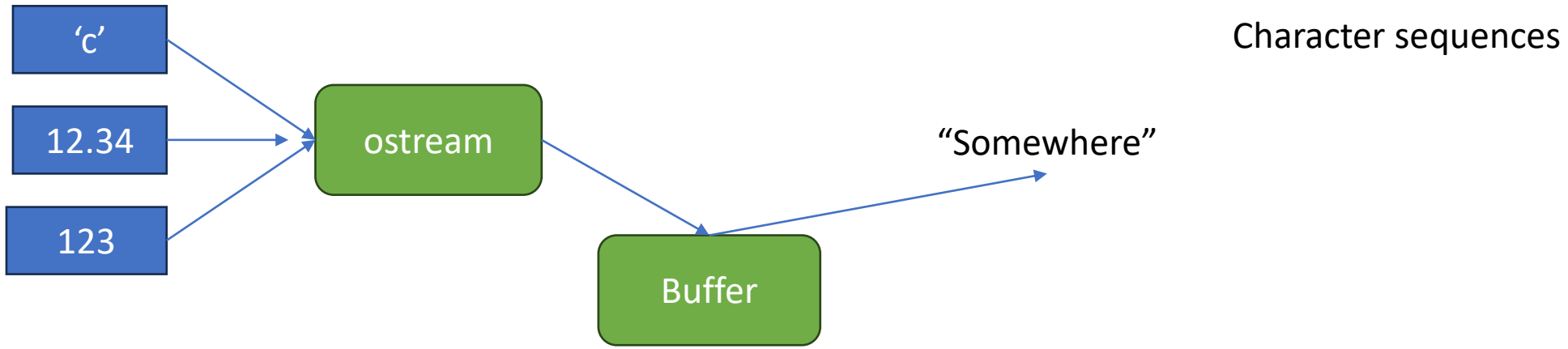
150 min (incl. Q & A)

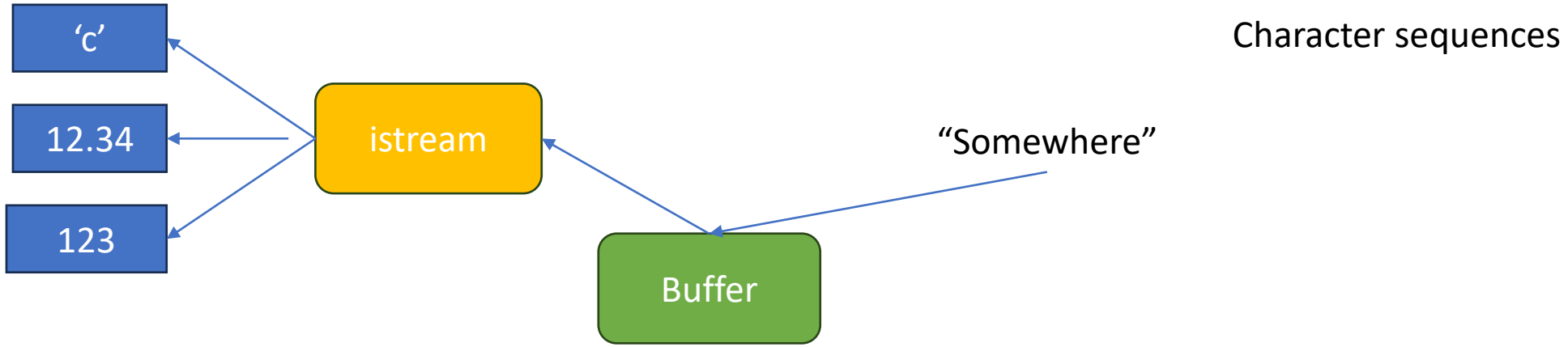
Input and output



the I/O stream model

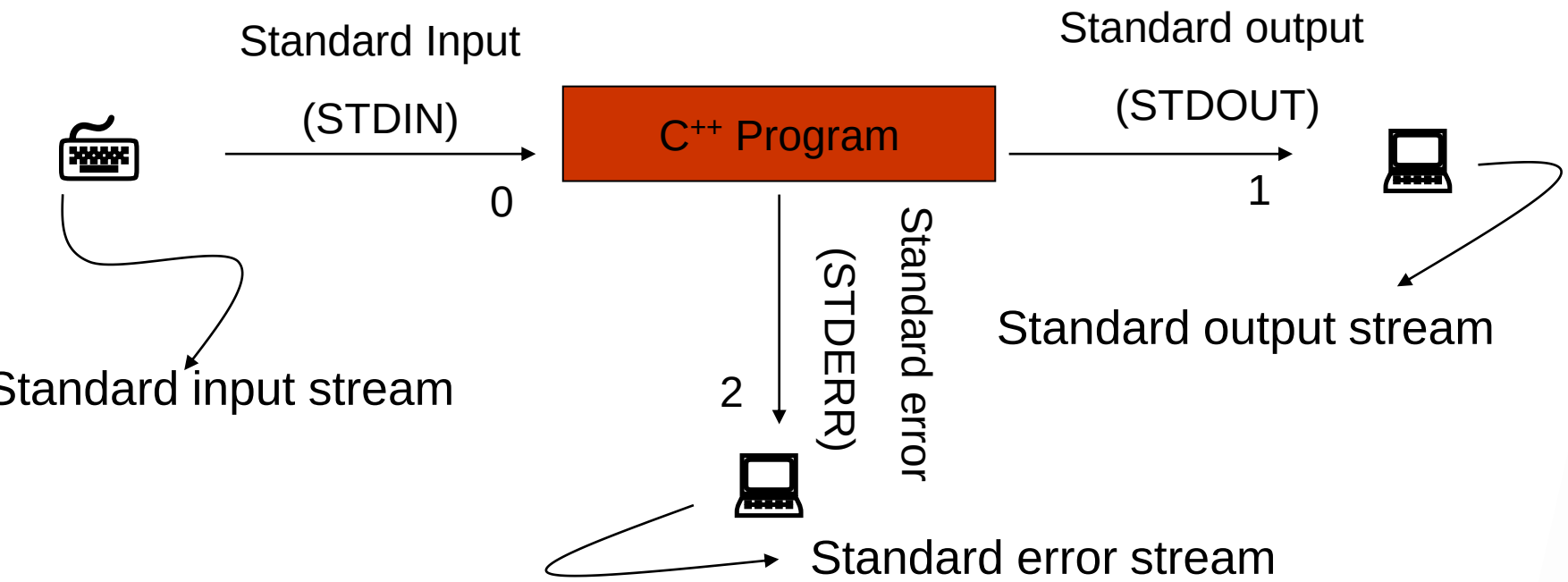
- The C++ standard library provides the type `istream` to deal with streams of input and the type `ostream` to deal with streams of output.



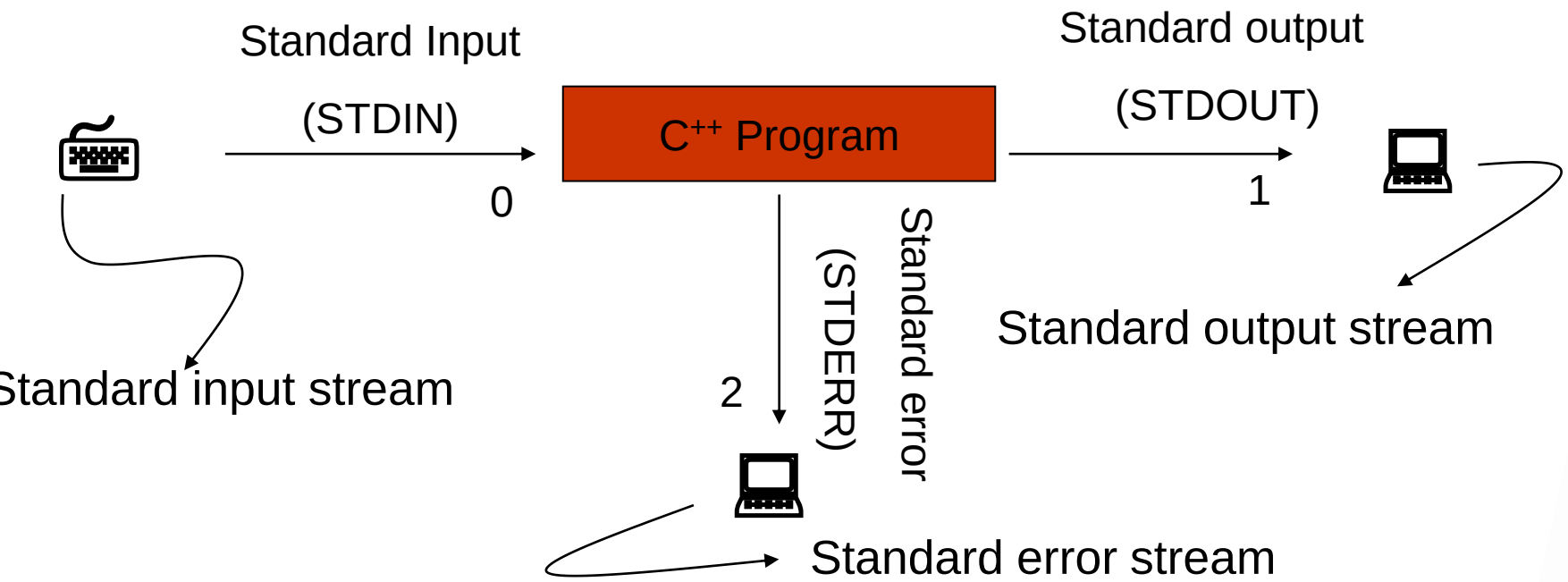


- Buffering


I/O Stream


Standard Input
(STDIN)
0
C++ Program
Standard output
(STDOUT)
1
Standard error
(STDERR)
2
Standard input stream
Standard output stream
Standard error stream

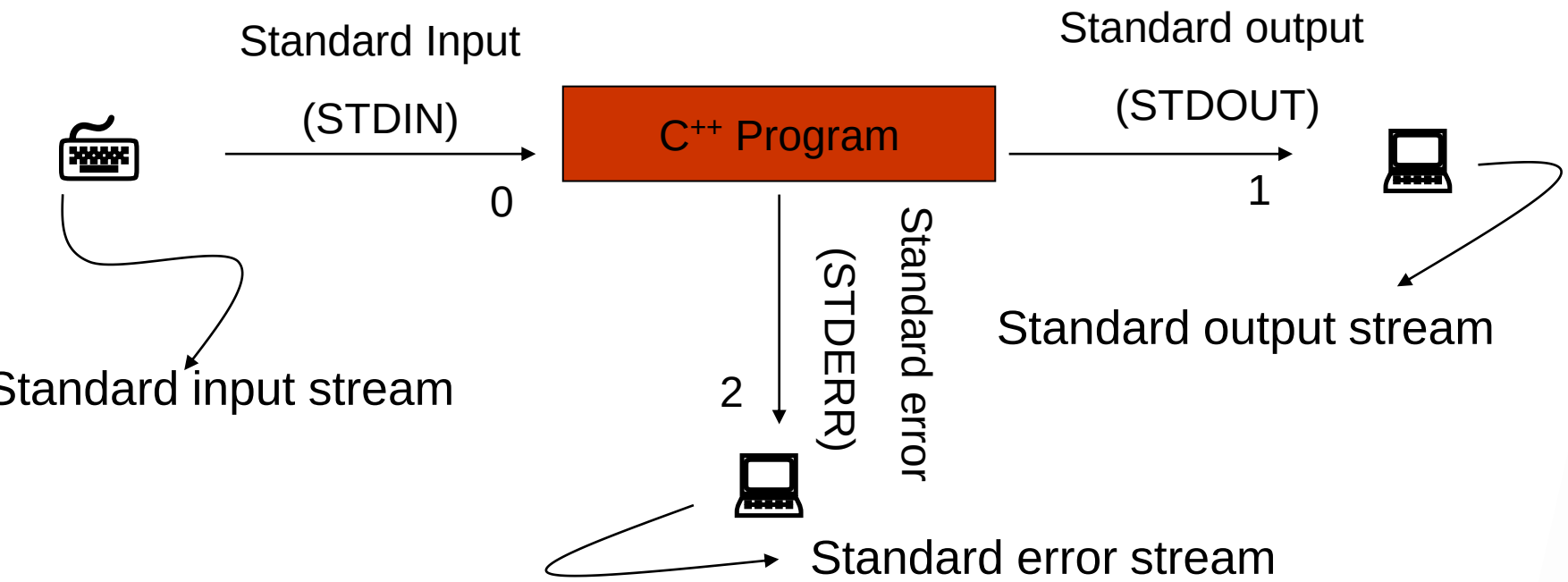

I/O Stream



- Default stream objects
- cin, an istream class object for standard input
- cout, an ostream class object for “ordinary output”
- cerr, an ostream class object for unbuffered “error output”
- clog, an ostream class object for buffered “logging output”

Output operation: “<<“, Insertion operator

Input operation: “>>“, Extraction operator

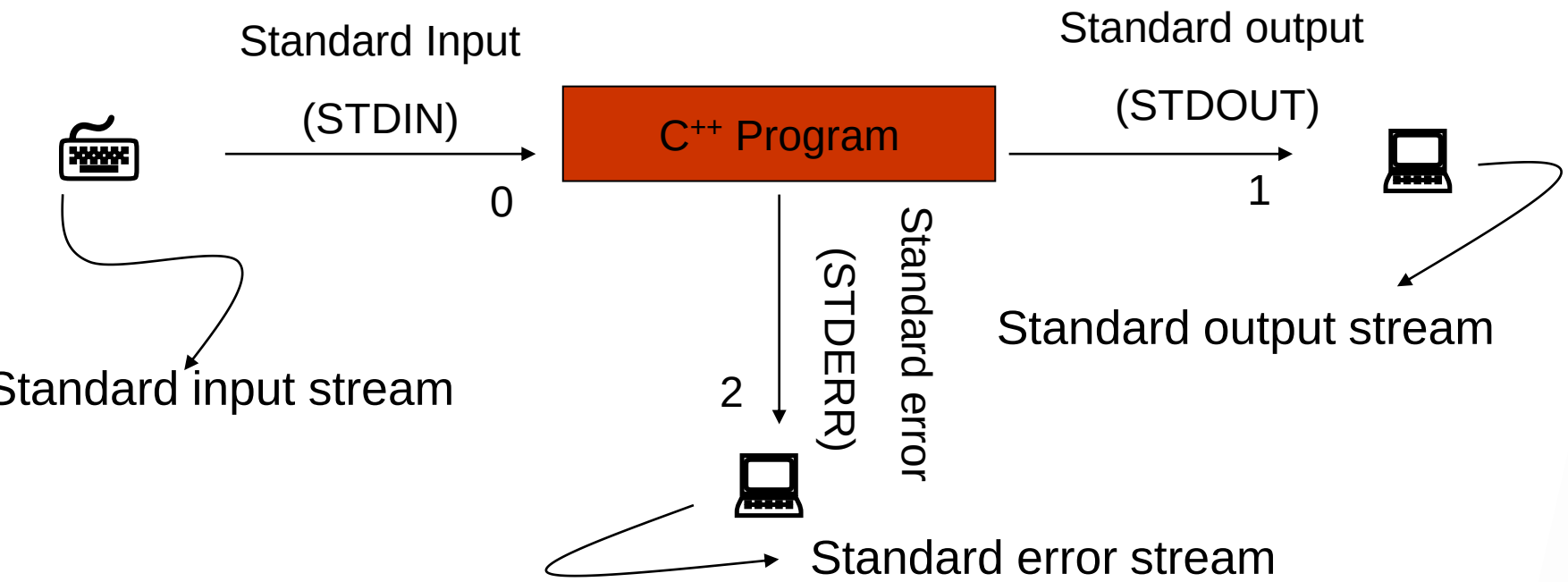


- Default stream objects
- cin, an istream class object for standard input
- cout, an ostream class object for “ordinary output”
- cerr, an ostream class object for unbuffered “error output”
- clog, an ostream class object for buffered “logging output”

```
cout << "What is you name? ";
cin >> name;
cout << "Hello " << name << '\n';
```

Output operation: “<<“, Insertion operator

Input operation: “>>“, Extraction operator



- Default stream objects
  - cin, an istream class object for standard input
  - cout, an ostream class object for “ordinary output”
  - cerr, an ostream class object for unbuffered “error output”
  - clog, an ostream class object for buffered “logging output”

```
cout << "What is you name? ";
cin >> name;
cout << "Hello " << name << '\n';
```

Output operation: “<<“, Insertion operator

Input operation: “>>“, Extraction operator

- Istream and Ostream are symmetric in C++
- Compare with Java

Cerr

- Character error stream
- cerr is exactly like cout except that it is meant for error message.
- by default both cerr and cout write to the screen, but cerr isn't optimized. It is unbuffered.
- Using cerr also has simple effect of documenting that what we write relates to errors.

Consequently, we use cerr for error messages.

```
void f()
{
    vector<char*> v;
    try {
        for (;;) {
            char* p = new char[10000]; // acquire some memory
            v.push_back(p); // make sure the new memory is referenced
            p[0] = 'x'; // use the new memory
        }
    }
    catch(bad_alloc) {
        cerr << "Memory exhausted!\n";
    }
}
```

Clog

- Character log stream
- clog is exactly like cout.
- by default clog writes to the screen.

Given this program:

```
#include <iostream>
int main()
{
  std::cout << "Hello, world!\n";
}
```

modify it to produce this output:

Initialize
Hello, world!
Cleanup

Do not change *main() in any way.*

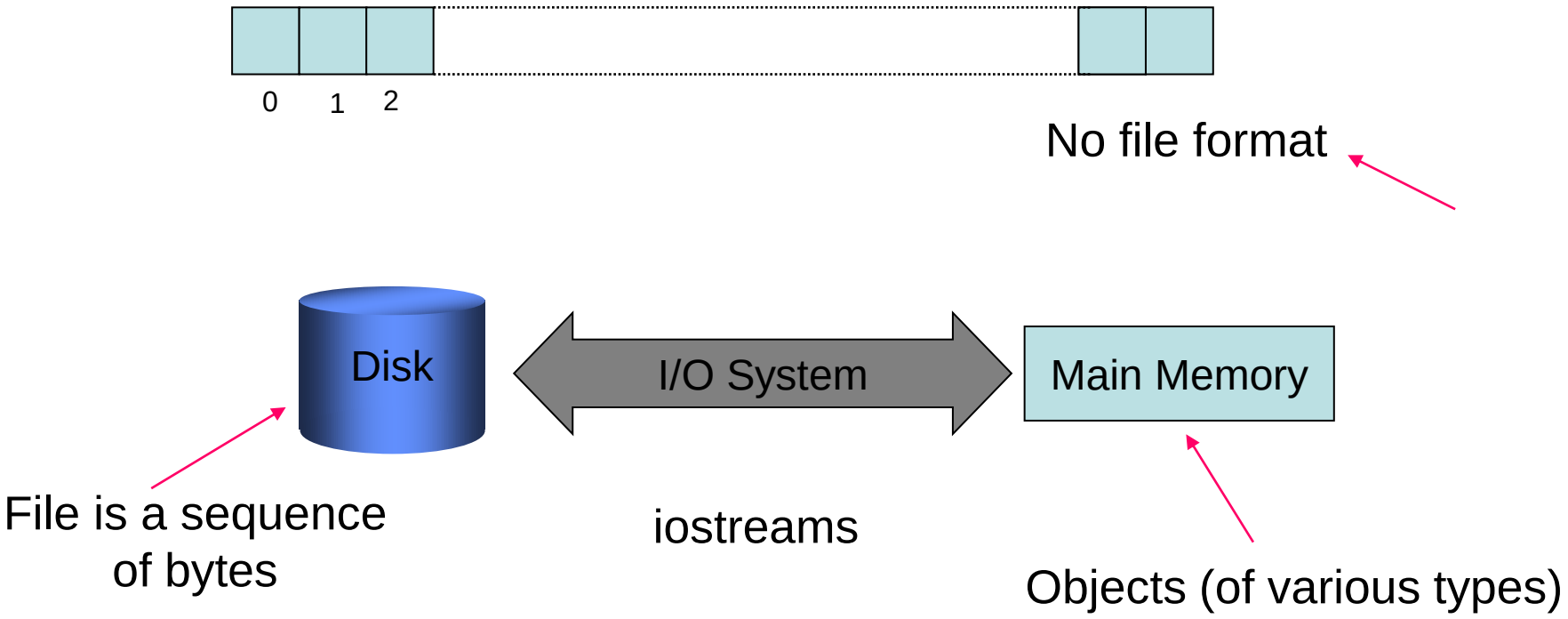
```
#include <iostream>
struct Wrapper { // simple wrapper
  Wrapper() { std::clog << "Initialize\n"; } // constructor
  ~Wrapper() { std::clog << "Cleanup\n"; } // destructor
} main_wrapper; // main function wrapper
int main()
{
  std::cout << "Hello, world!\n";
}
```

Clog Test Prog.

- Cool -10 liners code.

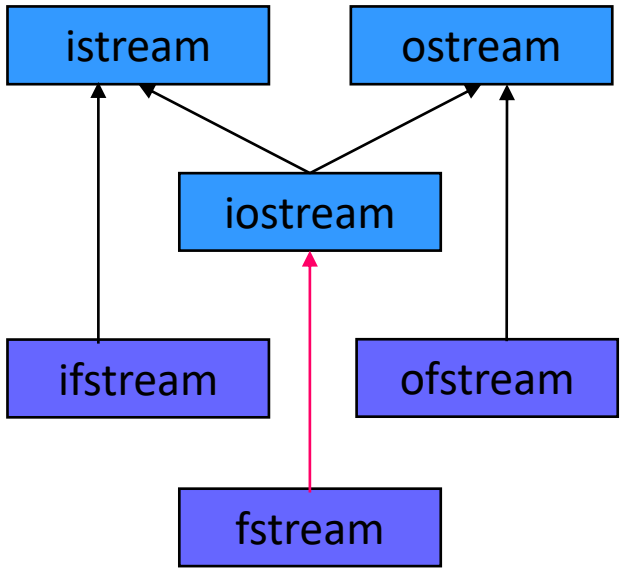
Files

- Main memory is transient storage.
- For keeping data, we should use permanent storage like disk.
- At the fundamental level, a file is a sequence of bytes numbered from 0 upwards. At this level, there is no file format.
- Unix: A file is a *stream* of bytes.



I/O Stream class hierarchy

- An `ifstream` is an `istream` for reading a file, an `ofstream` is an `ostream` for writing to a file, and an `fstream`.



Opening a file for reading

- To read a file
  - We must know its name
  - We must open it (for reading)
  - Then we can read
  - Then we must close it

    that is typically done implicitly

```
void error(const char* p1, const char* p2)
{
  std::cerr << p1 << ' ' << p2 << '\n';
  std::exit(1);
}

std::cout << "Please enter input file name: ";
std::string name;
std::cin >> name;
std::ifstream infile(name.c_str()); // infile is an input stream for the file named name
if (!infile) // check that the file was properly opened
    error("Can't open input file ", name);
```

Working with unknown number of data

```
while (cin >> x) { /* … */ }
```

- std::string: c_str() member function

Opening a file for writing

- To write a file
  - We must know its name
  - We must open it (for writing) Or create a new file of that name
  - Then we can write out our objects
  - Then we must close it
    that is typically done implicitly

```
void error(const char* p1, const char* p2)
{
  std::cerr << p1 << ' ' << p2 << '\n';
  std::exit(1);
}

std::cout << "Please enter input file name: ";
std::string name;
std::cin >> name;
std::ofstream outfile(name.c_str()); // outfile is an input stream for the file named name
if (!outfile) // check that the file was properly opened
    error("Can't open output file ", name);
```

Reading from and writing to files

- Read a string

```
string name;
cin >> name; // input: Dennis Ritchie
cout << name << '\n'; // output: Dennis
```

- Read a line: getline

```
string name;
getline(cin,name); // input: Dennis Ritchie
cout<< name << '\n'; // output: Dennis Ritchie
```

Line-oriented input

- Complete program

```
int main()
{
  ifstream infile("in.txt");
  ofstream outfile("out.txt");

  string s;
  while (getline(infile, s))
    outfile << s << endl;

  return 0;
}
```

Read Write File Using Get Line Prog.

Reading and writing a file: Complete example

- Weather station: Temperature (in Fahrenheit)

0	60.7
1	60.8
2	59.23
…	
23	57.35

ReadingTemperatureFromFile

Prog.

Stream states

Stream states

- Stream state:
  - `good()` : The operation succeeded
  - `eof()` : We hit end-of-input (“end of file”)
  - `fail()` : Something unexpected happened
  - `bad()` : Something unexpected and serious happened

Stream states

- Stream state:
  - `good()` : The operation succeeded
  - `eof()` : We hit end-of-input (“end of file”)
  - `fail()` : Something unexpected happened
  - `bad()` : Something unexpected and serious happened

```
int i = 0;
cin >> i;
if (!cin) { // we get here (only) if an input operation failed
    if (cin.bad())
        cerr << "cin is bad" << '\n'; // stream corrupted: let's get out of here!
    if (cin.eof())
        // no more input
        // this is often how we want a sequence of input operations to end
    }
    if (cin.failed()) { // stream encountered something unexpected
        cin.clear(); // make ready for more input
        // somehow recover
    }
}
```

Stream states

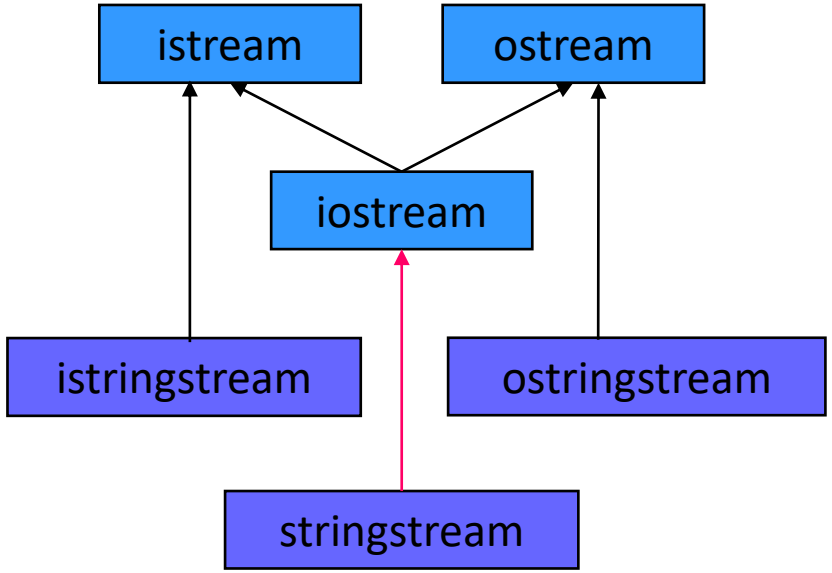
- Stream state:
  - `good()` : The operation succeeded
  - `eof()` : We hit end-of-input ("end of file")
  - `fail()` : Something unexpected happened
  - `bad()` : Something unexpected and serious happened

The state of cin is not good

```
int i = 0;
cin >> i;
if (!cin) { // we get here (only) if an input operation failed
    if (cin.bad())
        cerr << "cin is bad" << '\n'; // stream corrupted: let's get out of here!
    if (cin.eof())
        // no more input
        // this is often how we want a sequence of input operations to end
    }
    if (cin.failed()) { // stream encountered something unexpected
        cin.clear(); // make ready for more input
        // somehow recover
    }
}
```

String Stream

- You can use a `string` as the source of an `istream` or the target for an `ostream`. An `istream` that reads from `string` is called an `istringstream` and an `ostream` that stores characters written to it in a `string` is called an `ostringstream`.




```
double str_to_double(string s)
{
    istringstream is{s};
    double d;
    is >> d;
    if (!is)
        cerr << "double format error: " << s << '\n';

    return d;
}

double d1 = str_to_double("12.4");
double d2 = str_to_double("1.34e-3");
double d3 = str_to_double("twelve point three"); // error
```

I/O Stream class hierarchy

I/O Stream class hierarchy

- The I/O stream library provides formatted and unformatted buffered I/O of text and numeric values.

I/O Stream class hierarchy

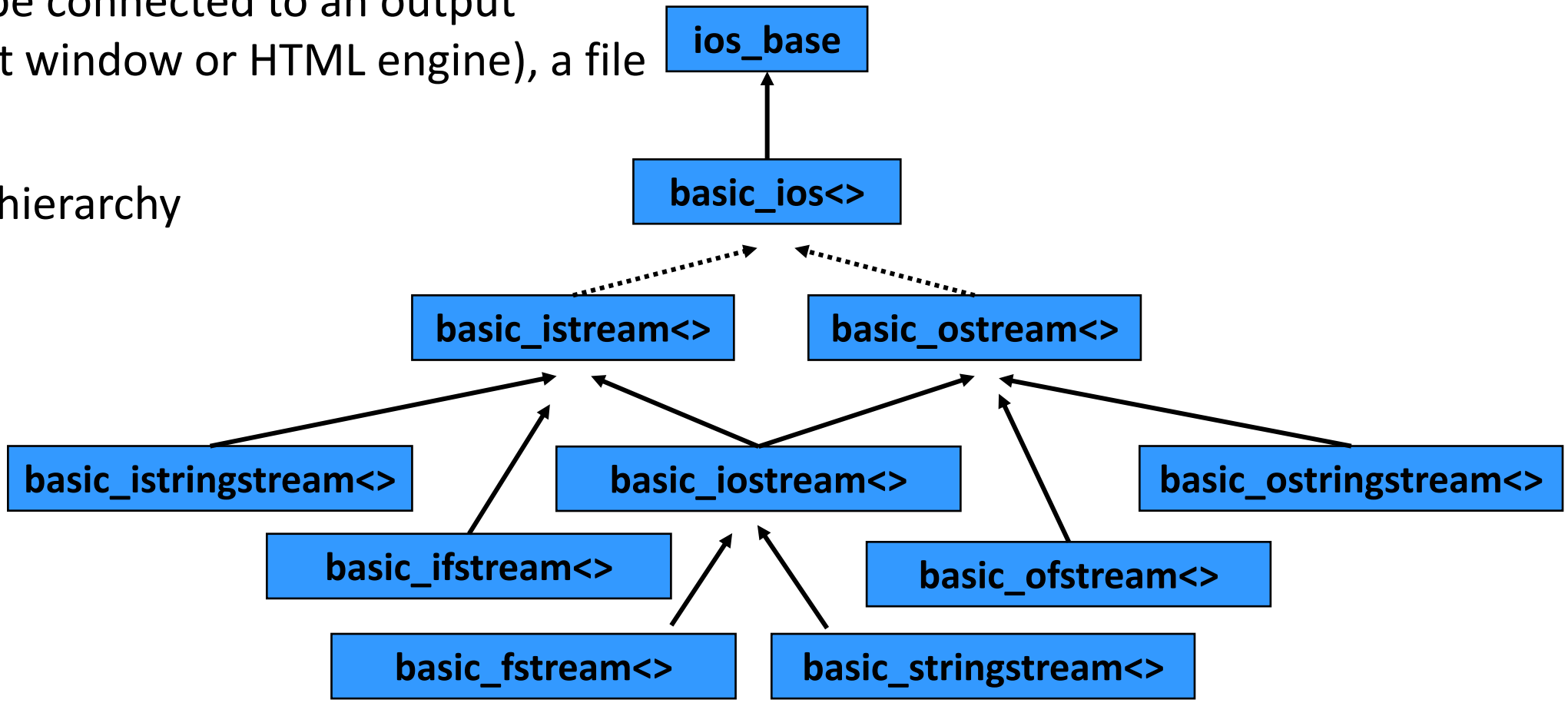
- The I/O stream library provides formatted and unformatted buffered I/O of text and numeric values.
- An istream can be connected to an input device (e.g. a keyboard), a file or a string.

I/O Stream class hierarchy

- The I/O stream library provides formatted and unformatted buffered I/O of text and numeric values.
- An istream can be connected to an input device (e.g. a keyboard), a file or a string.
- An ostream can be connected to an output device (e.g. a text window or HTML engine), a file or a string.

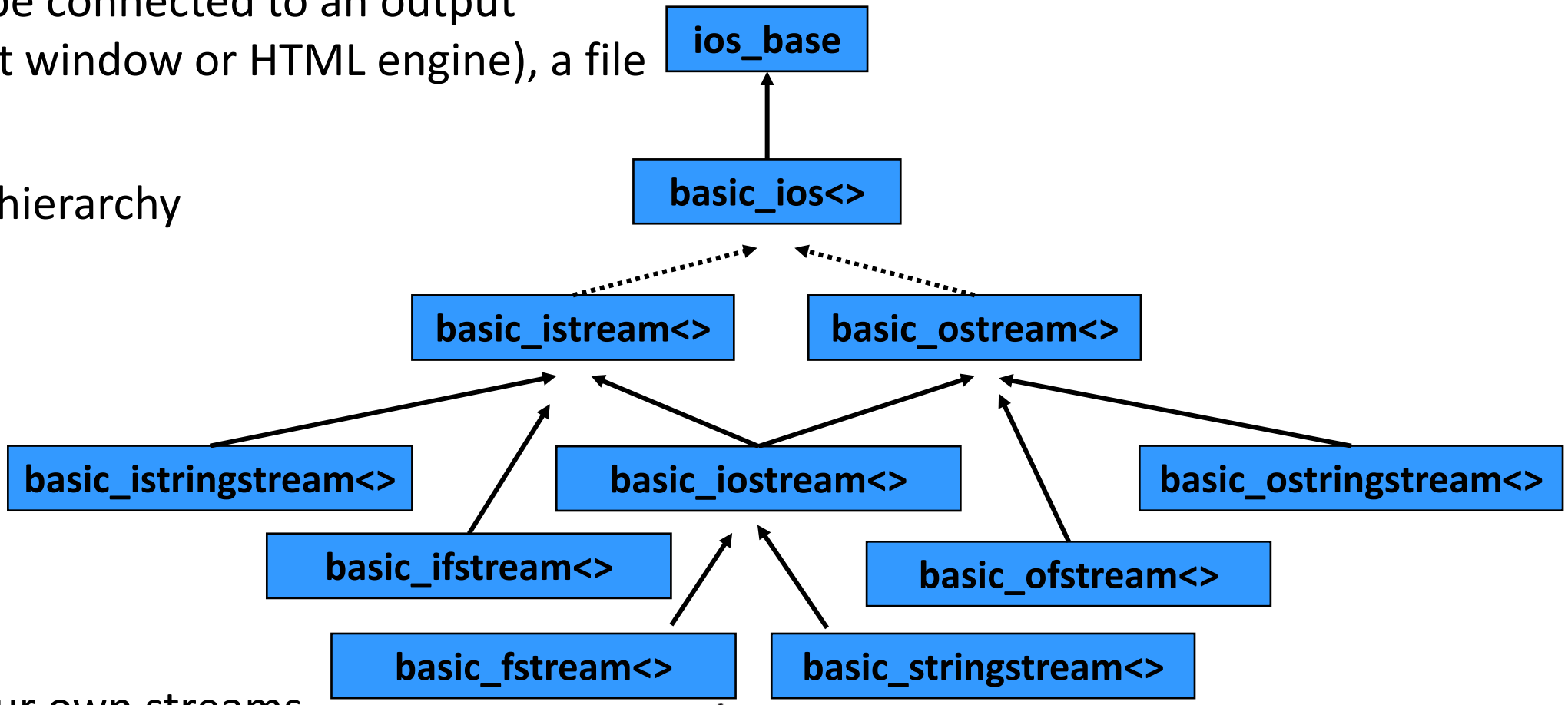
I/O Stream class hierarchy

- The I/O stream library provides formatted and unformatted buffered I/O of text and numeric values.
- An istream can be connected to an input device (e.g. a keyboard), a file or a string.
- An ostream can be connected to an output device (e.g. a text window or HTML engine), a file or a string.
- I/O stream class hierarchy



I/O Stream class hierarchy

- The I/O stream library provides formatted and unformatted buffered I/O of text and numeric values.
- An istream can be connected to an input device (e.g. a keyboard), a file or a string.
- An ostream can be connected to an output device (e.g. a text window or HTML engine), a file or a string.
- I/O stream class hierarchy



- You can create your own streams

Thanks for your patience …

A man who asks a question is a fool for minute,
The man who does not ask, is a fool for a life.
- Confucius

Learning to ask the right (often hard) questions is an essential part of learning to think as a programmer.
- Bjarne Stroustrup *programming Principles and Practice Using C++, page 4.*

There is no stupid question, but there is stupid answer.
- Howard Hinnant