# Advanced Programming Concepts with C++ CSI2372 – Fall 2019

Jochen Lang &
Mohamed Taleb
EECS

uOttawa
L'Université canadienne
Canada's university

Université d'Ottawa | University of Ottawa

uOttawa.ca

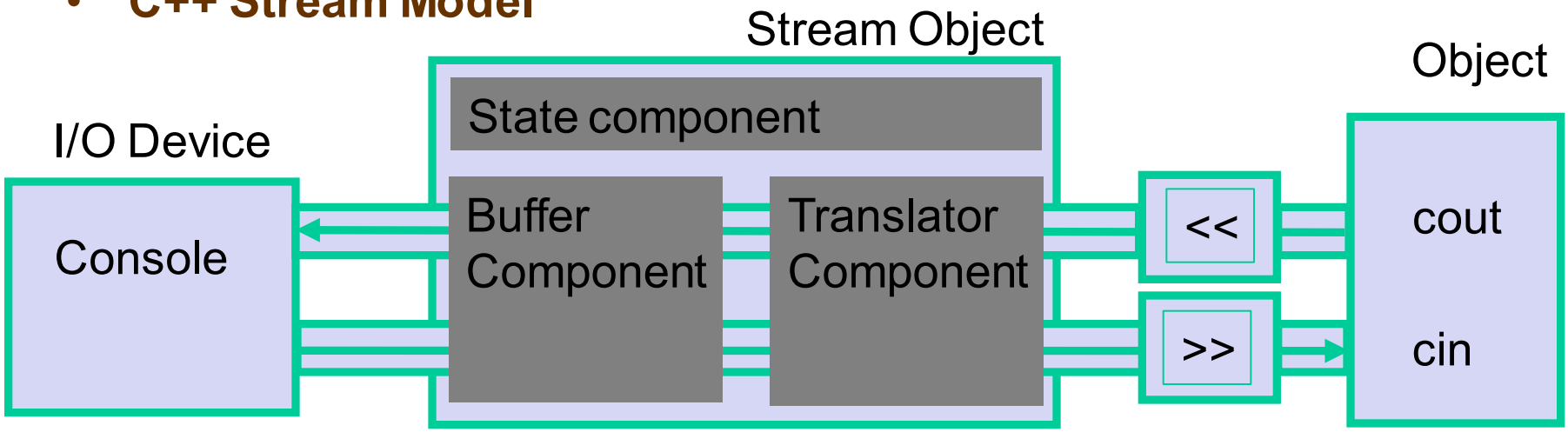
# This lecture

## Text is beautiful

- **Input and output streams**
  - Relevant classes for STL Stream I/O, Ch. 8.1
  - File handling, Ch. 8.2
  - Overloading the insertion and extraction operators 14.2
  - String streams, Ch. 8.3

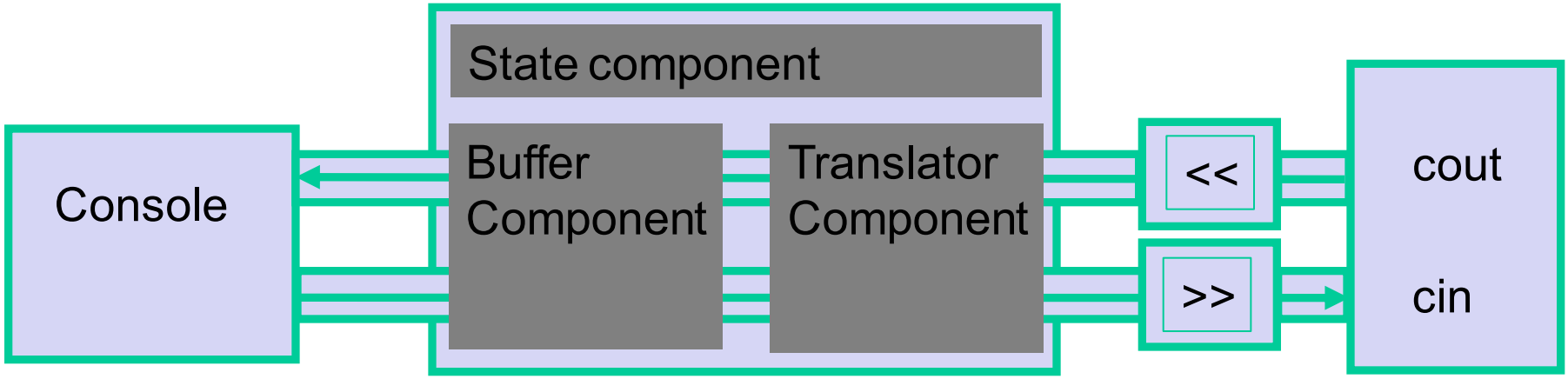
# Input/Output Streams

- **Streams**
  - Sequential data flows from/to devices
- **Examples**
  - Console input/output
  - File input/output
- **C++ Stream Model**



# Class Design

- **Translator classes**
  - `basic_istream, basic_ostream`
- **Buffer classes**
  - `basic_streambuf`
- **State classes**
  - `ios_base, basic_ios`



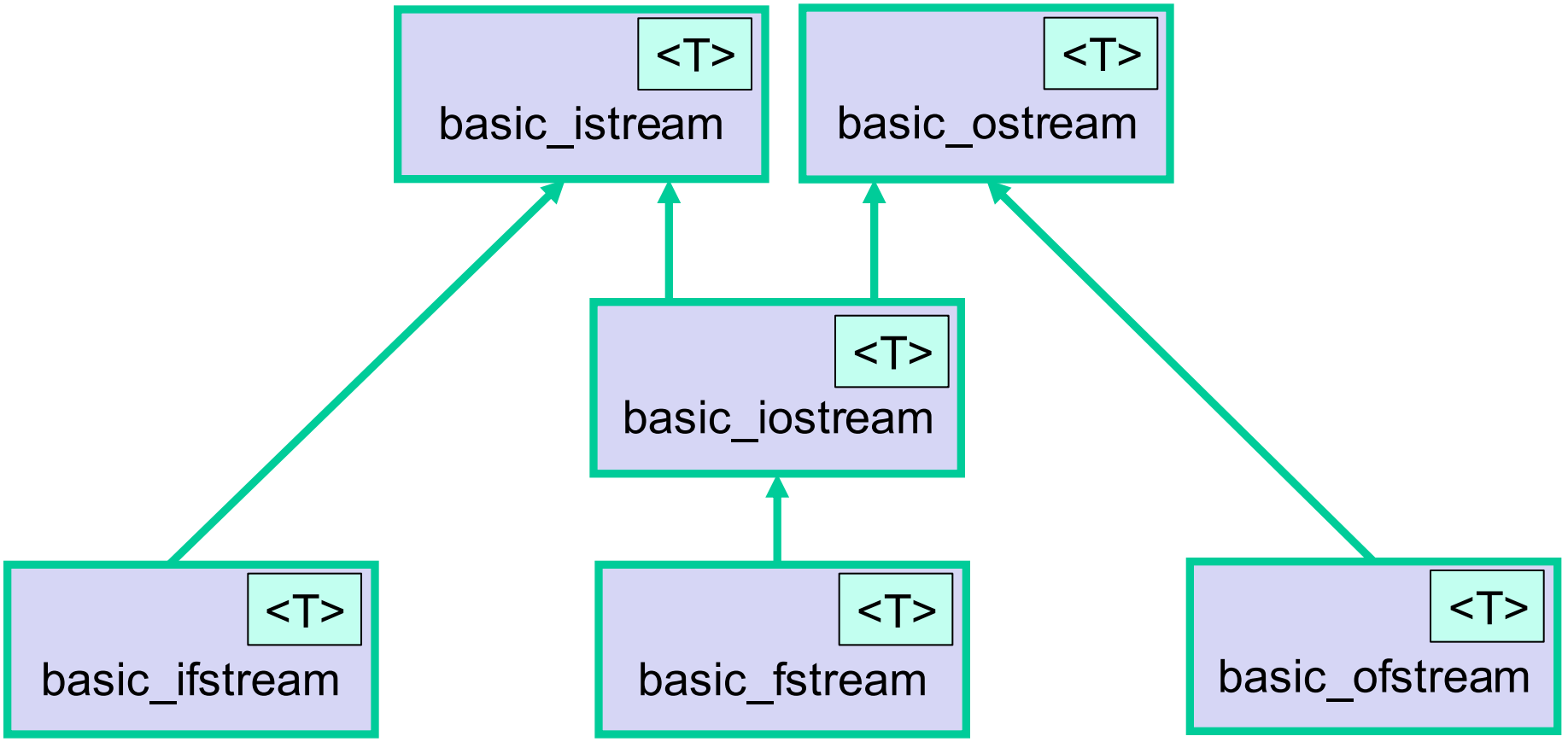
# Console input and output

```
istream cin;
typedef basic_istream<char, char_traits<char> > istream;
```

```
ostream cout;
typedef basic_ostream<char, char_traits<char> > ostream;
```

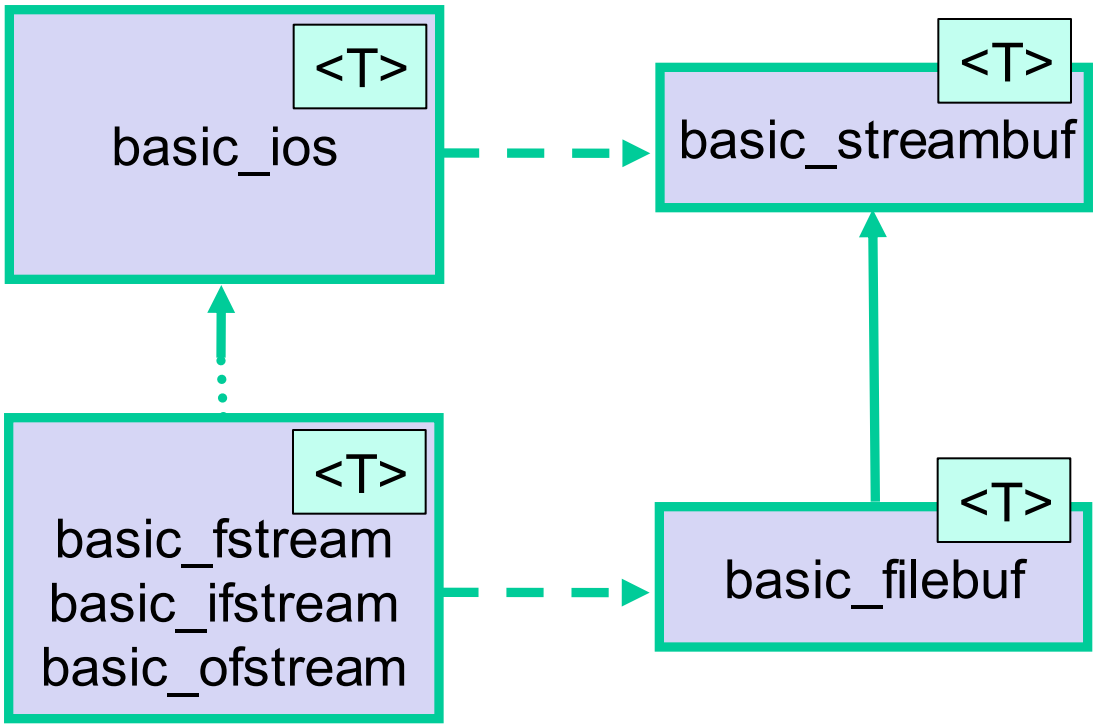
- Console works with char but also wchar_t
  - Templates <> enable the type to be a variable itself
    - Traits are a technique to give templates information about types
  - Languages which do not fit in char need wchar_t but this is not a complete Unicode implementation; only a basic tool to build such a library

# File Streams: Class Layout



# Benefits of Design

- **All streams are part of a common class hierarchy**
  - Streams have a common interface
  - Base classes can often be used instead of derived classes
  - Templates allow instantiation to standard or wide character streams



# Opening a File

- **Example:**

Old style C-string needed until C++11

```
string inFileName; ifstream inFile;
infile.open(inFileName.c_str(), ios::in );
if ( !infile ) {
  cerr << "Error: unable to open: " << inFileName << endl;
  return -1;
}
```

- **Important:**
  - Error checking
  - File modes
  - Note: File modes can be combined

| | |
|---|---|
| in | open for input |
| out | open for output |
| app | append: seek to end before every write |
| ate | seek to end only once after open |
| trunc | truncate stream at open |
| binary | binary mode |

# Combining File IO Modes

- **Possible combinations**

```
ios::out                // default for ofstream
ios::out | ios::app
ios::out | ios::trunc // same as ios::out
ios::in                 // default for ifstream (no truncation)
ios::in | ios::out    // default for fstream (no truncation)
ios::in | ios::out | ios::trunc
```

- **ate (short for at end) can be added to any of those (go to the end of file at initial opening of file but can change later with seek).**
- **app goes always to the end of the file (you cannot seek to earlier file positions).**

# Example: Copying a File

```
bool streamCopy( istream& _inS, ostream& _outS ) {
  char curC;
  if ( !_inS || !_outS ) return false; // check status
  _inS.clear(); _outS.clear(); // clear any errors
  while( !_inS.eof() ) {
    // use get instead of >> to extract white spaces
    _inS.get( curC );
    if ( _inS.fail() ) return false;
    _outS << curC;
  }
  return true;
}
string inFileName; ifstream inFile( inFileName.c_str());
string outFileName; ofstream outFile( outFileName.c_str());
streamCopy( inFile, outFile );
inFile.close(); outFile.close();
```

# Stream Input/Output for Non Built-In Types

- **Stream Output in Java**
  - Object defines a method toString
  - It is recommended that all classes override it
- **Stream Input and Output in C++**
  - Overload the insertion << and extraction operator >>

# Overloading the Insertion Operator <<

```
class Person;
ostream& operator<<( ostream&, const Person& );
class Person {
  string LastName;
  string FirstName;
  int Sin;
  …
  friend ostream& operator<<( ostream&, const Person& );
};
ostream& operator<<( ostream& _os, const Person& _p) {
  _os << _p.FirstName << "\t" << _p.LastName << "\t";
  _os << _p.Sin;
  return _os;
}
…
ofstream outFile; Person john( "John", "Dow" );
cout << "Person: " << john; outFile << john;
```

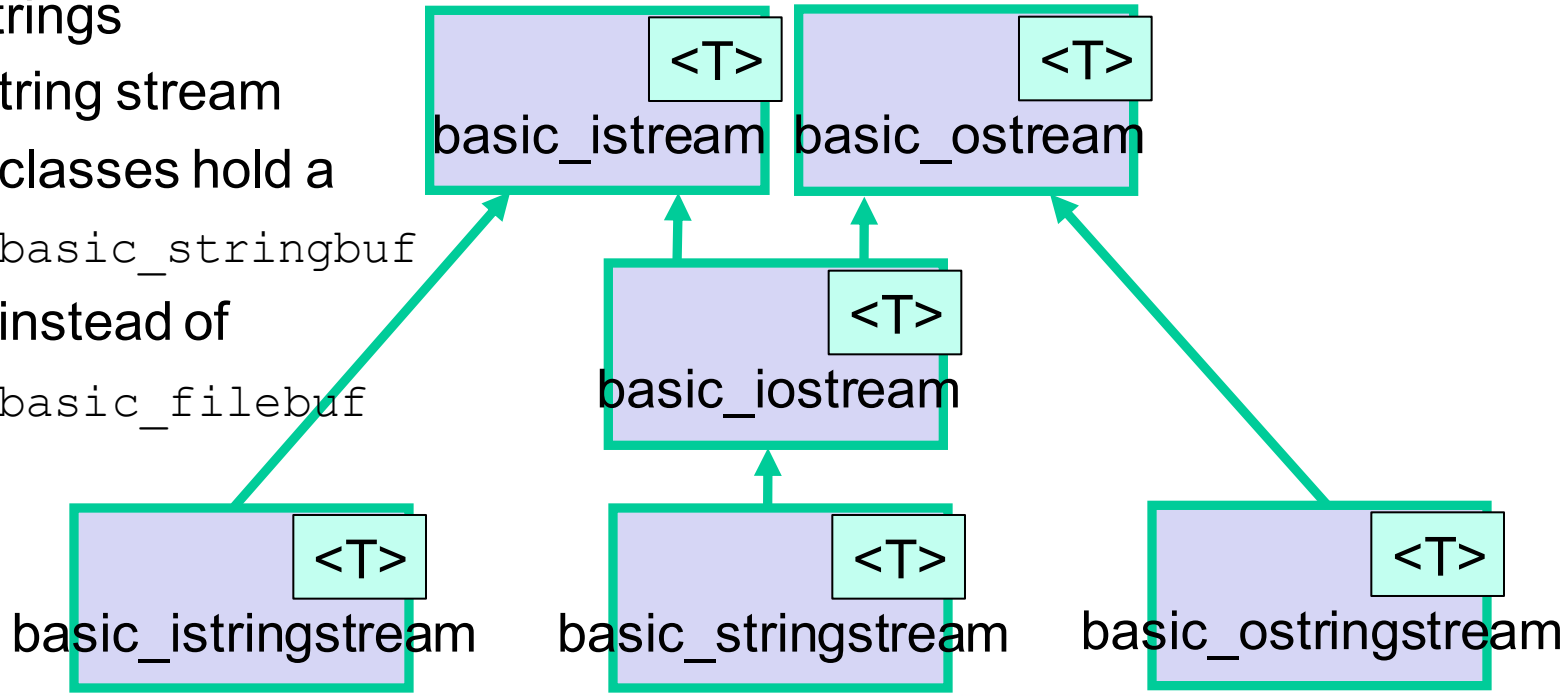
# Overloading the extraction operator >>

- Similar than insertion but need to check errors and leave object in valid state

```
class Person;
istream& operator>>( istream&, Person&);
class Person {  …
  friend istream& operator>>( istream&, Person& );
};
istream& operator>>( istream& _is, Person& _p) {
  _is >> _p.FirstName >> _p.LastName >> _p.Sin;
  // check and make a default Person on failure
  if ( !_is ) _p = Person();
  return _is;
}
…
Person toBeRead;
cin >> toBeRead; cout << "Person: " << toBeRead;
```

# String Streams

- **Same setup than file streams**
  - Read/write to memory rather than file or console
  - Important for parsing text (isstream) or converting types to strings
  - String stream classes hold a `basic_stringbuf` instead of `basic_filebuf`



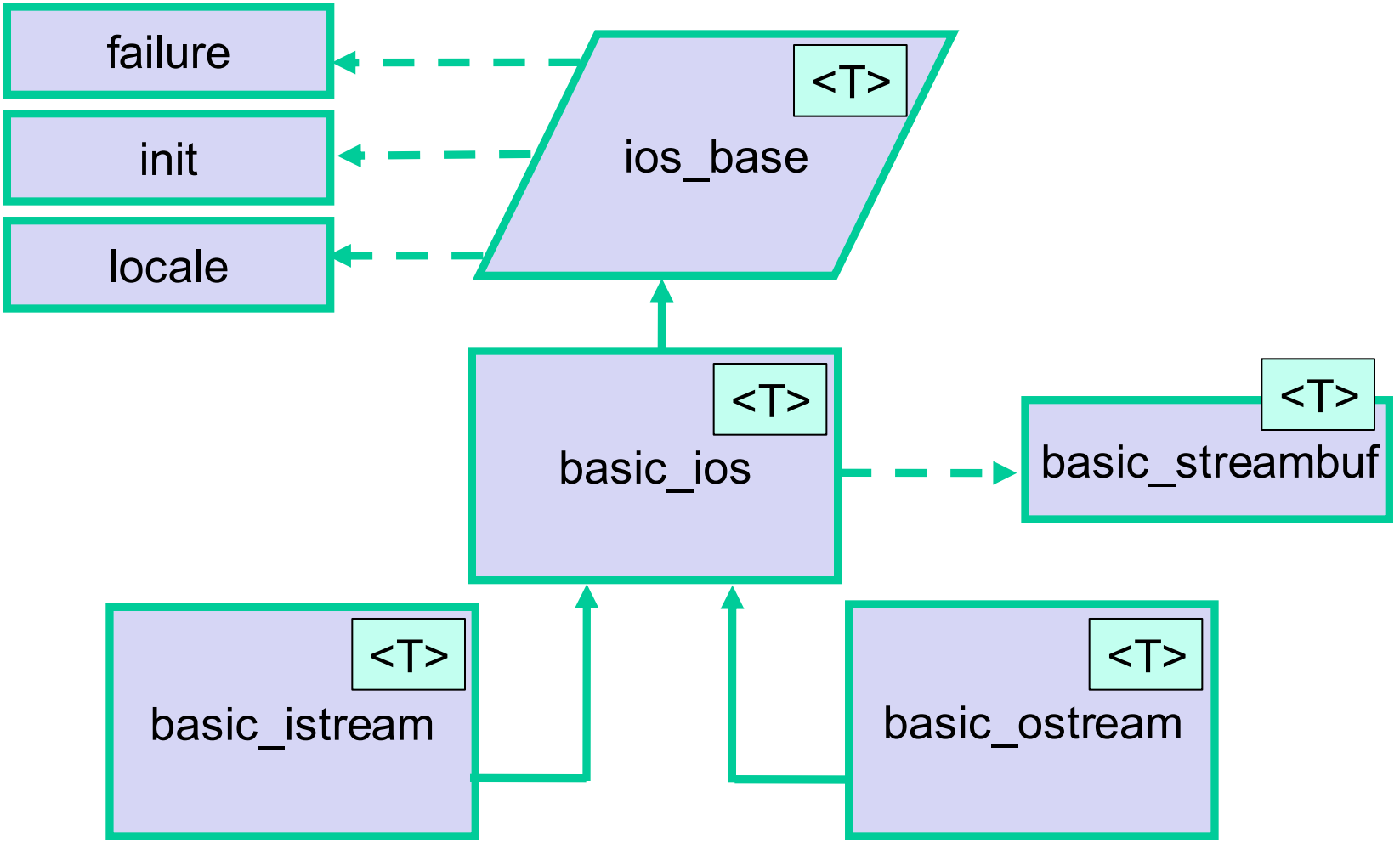
# String Stream Examples

- **Parsing input lines**

```
string line, token;
// Get a line
while (getline(cin, line)) {
istringstream streamLine(line);
// Get individual white space separated tokens
while ( streamLine >> token ) {
   // Process word
} }
```

```
string sNumber;
float fNumber;
ostringstream converter;
converter << fNumber;
sNumber = converter.str();
```

# Details: STL Stream Core Classes



# Working with Streams: Definitions

- **Querying state `ios_base, basic_ios` Examples:**
  - `bool basic_ios::eof();`
  - `bool basic_ios::bad();`
  - `void ios_base::clear(iostate _State = goodbit);`
  - `void ios_base::setstate(iostate _State, bool _Exreraise);`
- **Translation basic_istream, basic_ostream Examples:**
  - `ostream& ostream::operator<<( int val );`
  - `istream& istream::operator>>( int val );`
- **Changing the buffer basic_streambuf Examples:**
  - `int_type basic_streambuf::sungetc();`
  - NOTE: A pointer to a `basic_streambuf` can be retrieved with
  - `basic_streambuf *basic_ios::rdbuf();`

# Working with Streams: Example Usage

```
int myInt = 100;
char myChar = 'x';

cout << myInt << myChar << myFloat << endl;

cout << myString << endl;
cout << dec << myInt << ": "; // manipulator format
cout << oct << myInt << endl;
dec( cout );  // function call
cout.put( myChar ); cout.put(' ');   // put characters
cout.rdbuf()->sputc( myChar );    // into basic_streambuf

cin >> myInt;
if ( !cin.fail() ) { cout << myInt << endl;
} else {
  cin.clear(); cout << cin.rdstate() << endl;
}
// read form basic_streambuf
int curChar = cin.rdbuf()->sgetc();   // stay at position
curChar = cin.rdbuf()->sbumpc();
```

# Next Topic

**Just like int**

- **Abstract data types**
  - Operator overloading
  - Numerical vector and matrix classes in C++
  - Friend operator on classes and functions