Day 4 ReCap / Agenda

- Solutions
  - Jeff – More on boost::iostreams
  - Functional team report
  - Krishna – Experience re-writing
- Discussion

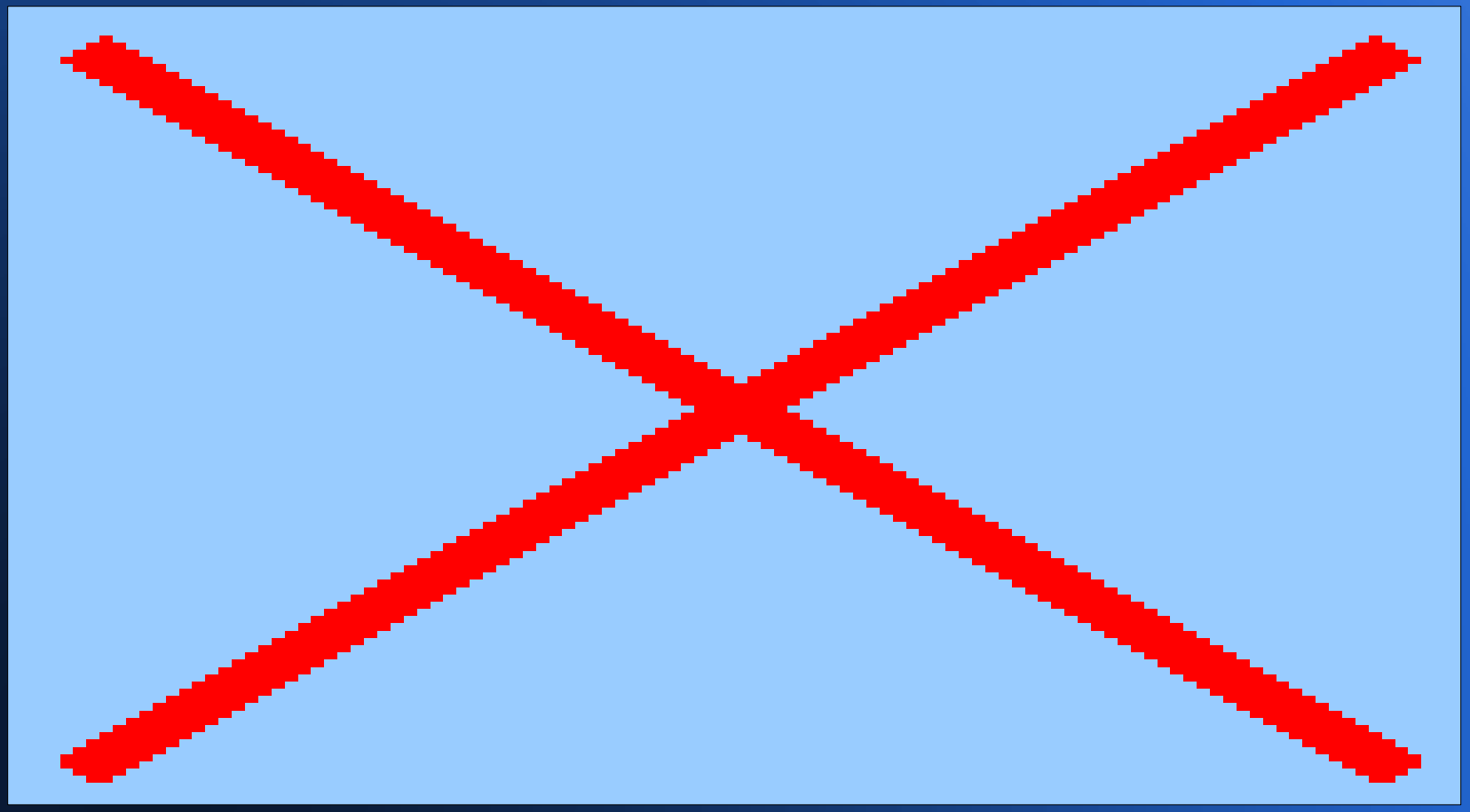
Core Concepts

- Devices
  - Source, Sink
  - Closeable
  - File
- Buffer
  - Always 'in front' of device
- Filter

Core Concepts 2

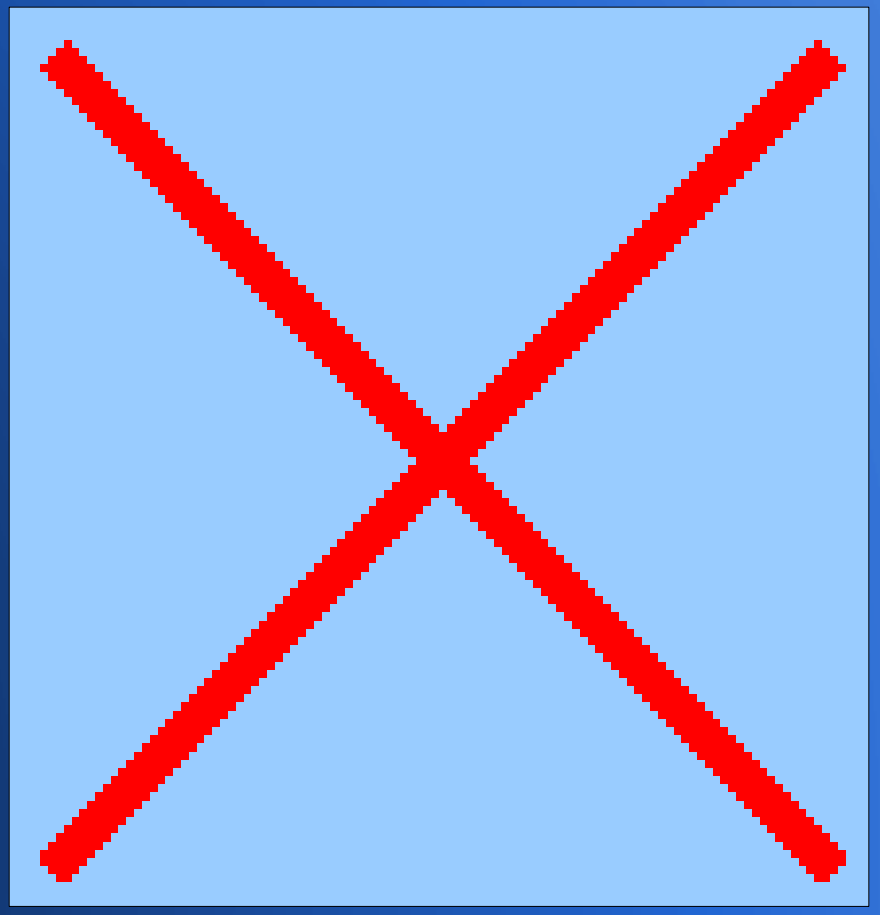
- Stream (was Pipeline)
  - InputStream – Parser,
- Formatter, Parser
- Sync, Async

Concepts Overview



Concepts Overview

- User application needs access to functions on all elements
- Sink and Source
  - Actaully devices (file, socket, memory)
- Filter
  - Can have multiple



Stream (was Pipeline)

- Combines a Device, Filter list, Formatter/Parser
- Derives from Formatter/Paser and Device
- Provides client interface

Streams

```
template <class input_device, class parser, class parser,
  class filters>

class input_stream : public input_device, parser


template <class output_device, class formatter, filters>

class output_stream : public output_device, formatter


template <class io_device, class parser_formatter, class
  filters>

class io_stream : public io_device, parser_formatter
```

Writing ofstream

Compatibility Layer

- What are the requirements?
  - Ideally users won't have to rewrite all code
  - Needing to modify code – ideally compatible with new and old
- What are the approaches?
  - Can't put things in name space std, so will need to rename

Compatibilty Example

```
struct foo
{
foo(int field1, double field2) :
 f1(field1),
     f2(field2)
{}
int f1;
double f2;
};
```

Compatibility Example

```
std::ostream&
operator<<(std::ostream& os,

  const foo& f)
{
   os << f.f1
      << " "
      << f.f2;
   return os;
}
```

```
template<StreamType>
StreamType&
operator<<(StreamType& os,
const foo& f)
{
   os << f.f1
      << " "
      << f.f2;
   return os;
}
```

A 'Real' Extraction Operator

```
template <class CharT, class TraitsT>
inline std::basic_ostream<CharT, TraitsT>&
operator<<(std::basic_ostream<CharT, TraitsT>& os, const boost::gregorian::date& d) {
  boost::io::ios_flags_saver iflags(os);
  typedef boost::date_time::date_facet<date, CharT> custom_date_facet;
  std::ostreambuf_iterator<CharT> output_itr(os);
  if (std::has_facet<custom_date_facet>(os.getloc()))
    std::use_facet<custom_date_facet>(os.getloc()).put(output_itr, os, os.fill(), d);
  else {
    custom_date_facet* f = new custom_date_facet();
    std::locale l = std::locale(os.getloc(), f);
    os.imbue(l);
    f->put(output_itr, os, os.fill(), d);
  }
  return os;
}
```

Compatibility Approaches

- Real need is to support existing paradigm for user code
- Ignore basic_ classes, provide only equivalents for *stringstream, *fstream,
- What about the 'global objects'
  - Perhaps better approach is to export the global buffer interface for cout, clog, etc.

Problems

- Putting in char types/traits – probably not too bad
- Buffering filters may be an issue – again likely not insurmountable