



Montreal Code Sprint

LIBPC – DESIGN

mpg
Flaxen Geo
15 March 2011



Contents

- libPC Goals
- Design
 - Pipeline Architecture
 - The Classes
- Open Issues

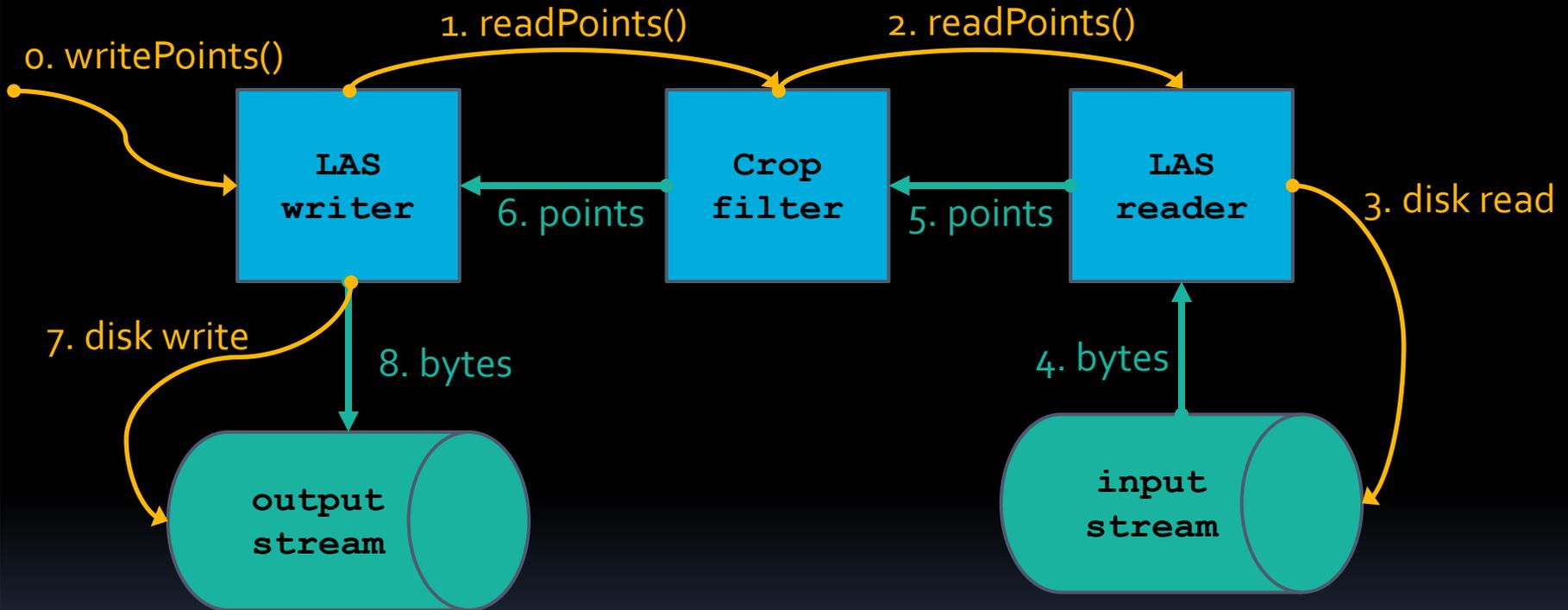
libPC Goals (1-3)

1. libPC is a library which provides APIs for reading, writing, and processing point cloud data of various formats. Additionally, some command line tools are provided. As GDAL is to 2D pixels, libPC is to multidimensional points.
2. From a market perspective, libPC is "version 2" of libLAS. The actual code base will be different, however, and the APIs will not be compatible.
3. The libPC implementation has high performance, yet the API remains flexible. We recognize that these two goals will conflict at times and will weigh the tradeoffs pragmatically.

libPC Goals (4-6)

4. The architecture of a libPC-based workflow will be a pipeline of connected stages, each stage being either a data source (such as a file reader), a filter (such as a point thinner), or data sink (such as a file writer).
5. The libPC library will be in C++, but will also include a C API and will have SWIG bindings for languages like Python and C#. libPC will support multiple platforms, specifically Windows, Linux, and Mac.
6. libPC is open source and is released under a BSD license.

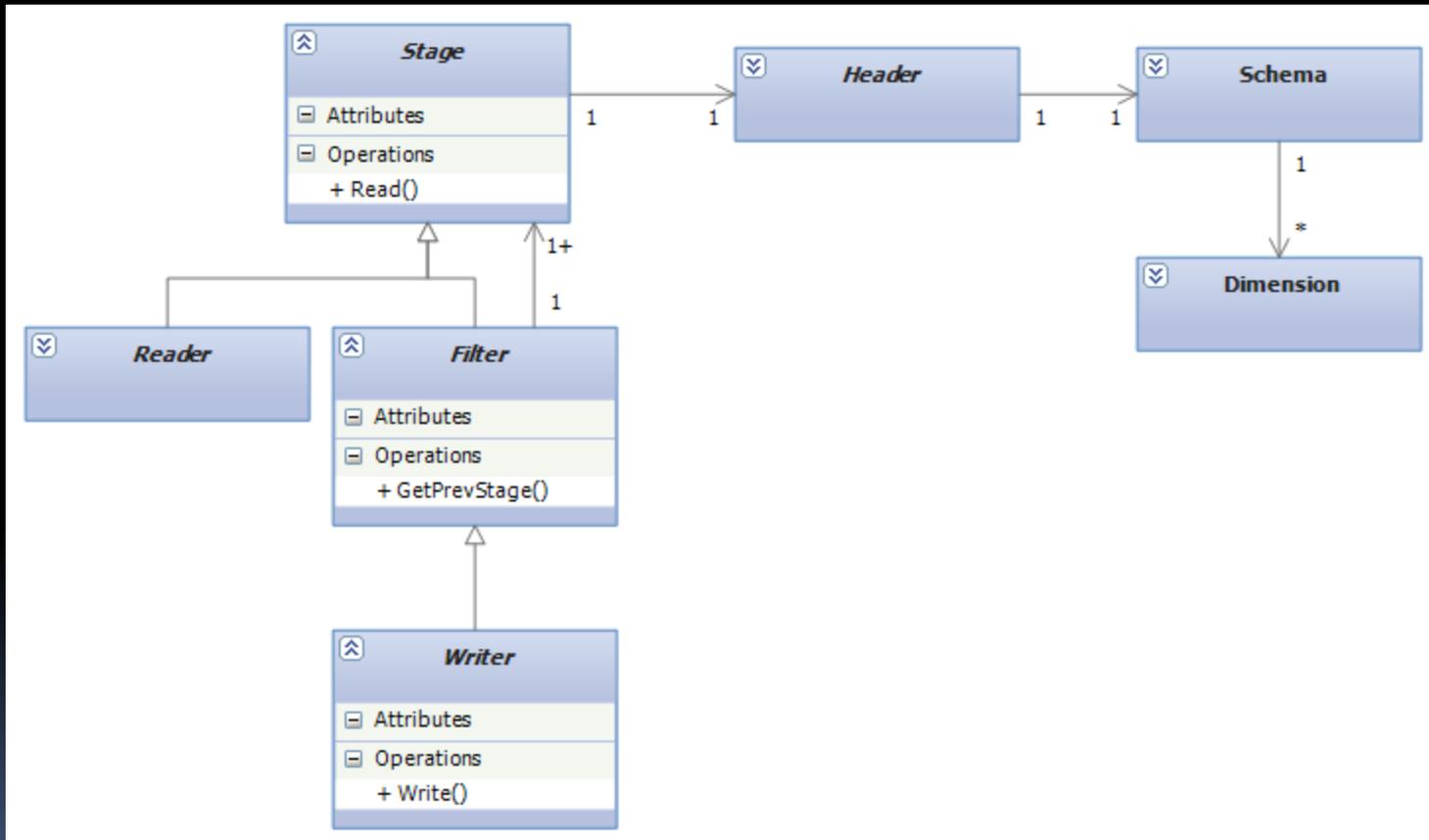
Pipeline Design



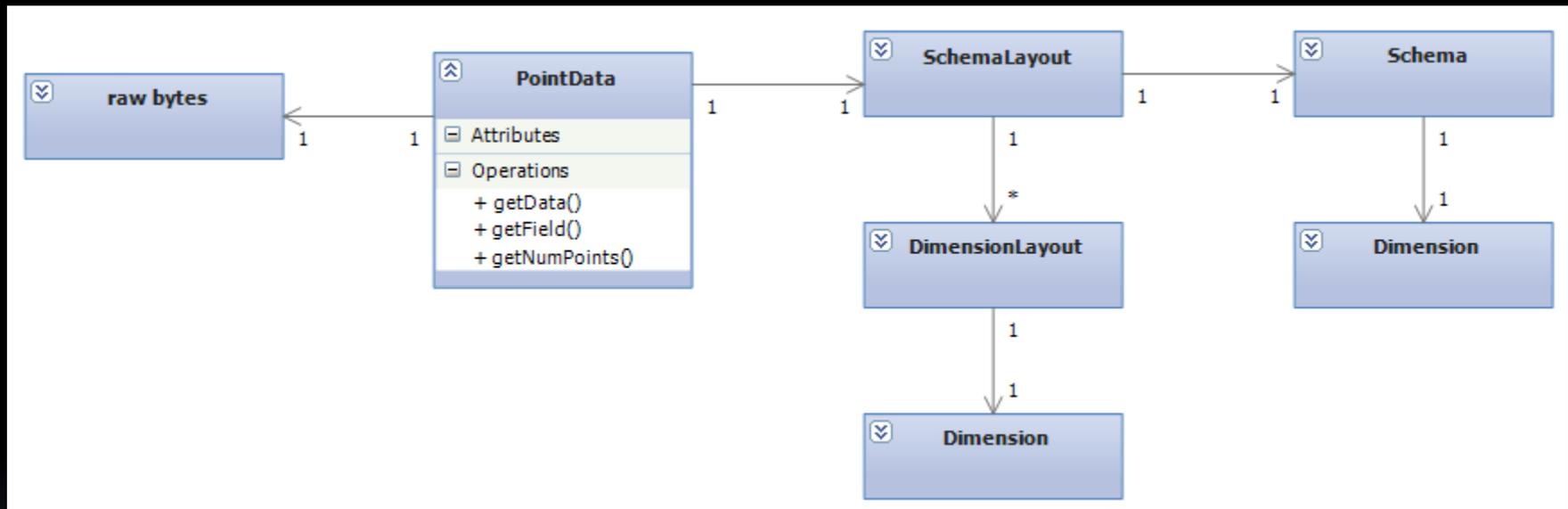
Pipeline Design Goals

- Aimed at:
 - command line apps for data processing
 - secondarily, viewing
- “Readers” and “Writers”
 - Producers/Consumers, Sources/Sinks, ...
 - Composition of stages
- Pipeline is static
 - Setup once at run time
 - Then use (read from) repeatedly

Key Classes (1)



Key Classes (2)



Groups of Classes

- **Math**
 - Range, Vector, Bounds
- **Schemata**
 - Dimension, DimensionLayout, Schema, SchemaLayout, PointData
- **Pipeline**
 - Header, Stage, Reader, Writer, Filter
- **Other**
 - Utils, Signaller, exceptions, Color, Metadata, SpatialReference

Math Range<T>

- A minimum/maximum pair
- Features
 - get/set min/max
 - predicates: equality, overlaps, contains, ...
 - clip, grow, scale, ...



Math Vector<T>

- A vector of values, in the mathematical sense
- Features
 - length fixed at ctor
 - element accessors
 - equality testing

Math Bounds<T>

- A list of Ranges
 - such as an (x,y,z) bounding box
- Features:
 - get/set min/max
 - predicates: equality, empty, contains, overlaps, ...
 - grow, clip, ...

Schemata Dimension

- Strongly typed data field
 - such as "X Position" or "GPS Time"
- Features:
 - field name enum
 - XPosition, GPSTime, ReturnNumber
 - data type enum
 - Int8, ..., Uint64, ..., double
 - getNumBytes, isSigned, isNumeric, ...
 - supports "scale/offset" uint32/float concept

Schemata Schema

- A set of Dimensions
 - conceptually like a database schema
 - unordered; no associated physical layout
- Features:
 - `void addDimension(const Dimension&)`
 - `const Dimension& getDimension(size_t index)`
 - `size_t getDimensionIndex(Field field)`

Schemata DimensionLayout

- Represents a Dimension, as physically stored
 - conceptually, an array of raw bytes
 - the schema is overlaid/union'd over that
- Features:
 - `getByteOffset()` // starting byte in raw bytes array
 - `getPosition()` // index into Dimensions array

Schemata SchemaLayout

- An ordered list of DimensionLayouts, to represent the physical layout on disk
- Features
 - getSchema
 - getByteSize() // sum of all dimensions
 - getDimensionLayout(size_t index)

Pipeline Header

- Basic data associated with all stages
 - Number of points, bounds, ...
- Features
 - get/set Schema, NumPoints, Bounds, ...
 - get/set Metadata, SpatialReference

Pipeline Stage

- Stage is abstract base class for a pipeline unit
 - for Readers, Writers, Filters
- Features:
 - `uint32_t read(PointData&)`
 - `virtual void readBegin(uint32 numPointsToRead)`
 - `virtual uint32 readBuffer(PointData&)`
 - `virtual void readEnd(uint32 numPointsRead)`
 - `seekToPoint(), getCurrentPosition(), bool atEnd()`
 - `getHeader()`

The read() protocol

- user calls `Stage::read(PointData&)`
 - user supplies the `PointData` object
 - this is not virtual, do not override
- `read()` does this for you:
 - `readBegin`
 - `readBuffer()`
 - `readEnd`
- you override these in your derived classes

Pipeline Reader

- Derives from Stage
 - only difference is it supplies code to manage the current point number for you
- Features
 - adds `m_currentPointNumber`

Pipeline Filter

- Derives from Stage
 - ctor takes a Stage& (the “previous” stage)
 - only difference is it supplies default implementations of read functions, etc.
- Features
 - readBegin(): m_prevStage.readBegin()

...

Pipeline Writer

- Derived from Filter, designed for file-based writers
- Features:
 - `write(size_t numPoints)`
 - `protected writeBegin, writeBuffer, writeEnd`
 - `get/set ChunkSize`

The write() protocol

- User calls write(numPointsToWrite)
 - not virtual
- writeBegin/Buffer/End are virtual
 - Derived classes provide these
- write() does this:
 - writeBegin()
 - numChunks = numPoints / chunkSize
 - for each chunk do
 - PointData data(chunkSize)
 - prevStage.readBuffer(data)
 - writeBuffer()
 - writeEnd()

Other Utils

- Provides a dumping ground for static helper functions
- Features:
 - `random()`
 - `compare_epsilon`
 - iostream helpers: `open/create/close` file
 - file helpers: `rename`, `getSize`, `delete`
 - `read/writeField<T>(uint8*)`

Other Signaller

- Provides callbacks for progress reporting and requesting interrupts during long-running pipeline operations
 - class done, but not yet implemented anywhere
- Features
 - virtual void setPercentComplete(double)
 - virtual bool isInterruptRequested() const

Other exceptions

- Provides some libPC-named exceptions for specific error situations
- Features
 - `libpc_error` // base class
 - `invalid_point_data`
 - `invalid_format`
 - `configuration_error`
 - `not_yet_implemented`

Other Color

- Provides a simple holder for an R,G,B triplet
- Features
 - get/set Red/Green/Blue
 - interpolate(value, rangeMin, rangeMax)
 - // provides a mapping into a color ramp



Other Metadata

- Provides a holder for an arbitrary array of bytes
 - class not yet implemented

Other SpatialReference

- Provides a holder for some sort of SRS representation
 - class not yet implemented
 - will provide WKT, EPSG code, reprojection, etc.
 - likely heavily dependent on LIBPC_HAVE_GDAL

Concrete pipeline classes

- Things derived from Stage
- Readers/Writer (Drivers)
 - LasReader, LasWriter
 - LiblasReader, LiblasWriter
 - OCI
 - FauxReader, FauxWriter
- Filters
 - CacheFilter, ColorFilter, DecimationFilter, CropFilter, MosaicFilter



Issues

Some things are still open and need resolution

- like, this week

Issue Capabilities

- It would be a Good Thing if some readers could advertise certain features
 - I_Support_Spatial_Indexing
 - I_Don't_Like_Doing_Random_Seeks
 - ...?
- Knowing this information would help a pipeline-creator be able to omit a filter, for example
- Questions
 - What is the set of capabilities offered?
 - Should they be exposed from Stage?
 - How should they be expressed?
 - Boost-style traits?

Issue PointData Templating

- The Dimension class uses an enum for the data type
 - It is not `Dimension<T>`, because then there is no base class to allow for `std::vector<Dimension>`
 - plus, virtual function overhead?
- But PCL does templates, sayeth Hobu
- Task: give a 5-10 min talk on PCL
 - how/why it is different from libPC
 - what ideas can we adopt?

Issue Sequential or Random?

- Two worlds
 - LAS (files) present a sequential list of points
 - OCI (queries) present a spatially indexed set of points
- But:
 - Stage::read() is really a sequential/file model
- What can/should we do about this?

Issue Stage Class Hierarchy

The naming of classes is a difficult matter
It isn't just one of your holiday games

- Reader, Writer, Filter, Producer, Consumer, Source, Sink, ...
- What is a Writer?
 - The Chipper ain't one.
 - Should it really be it's own class?

Issue IsValid?

- It seemed like a good idea at the time
 - But doesn't seem useful at all now.
 - Just one more thing for developers to need to remember to do
 - The default action goes the wrong way

Issue Miscellany

- Get rid of header class? (fits into stage)
- Boost equivalents for Range, Vector, etc.?
 - (where are we actually using these?)
- Bit fields in dimensions?
 - Needed for direct mapping to disk only