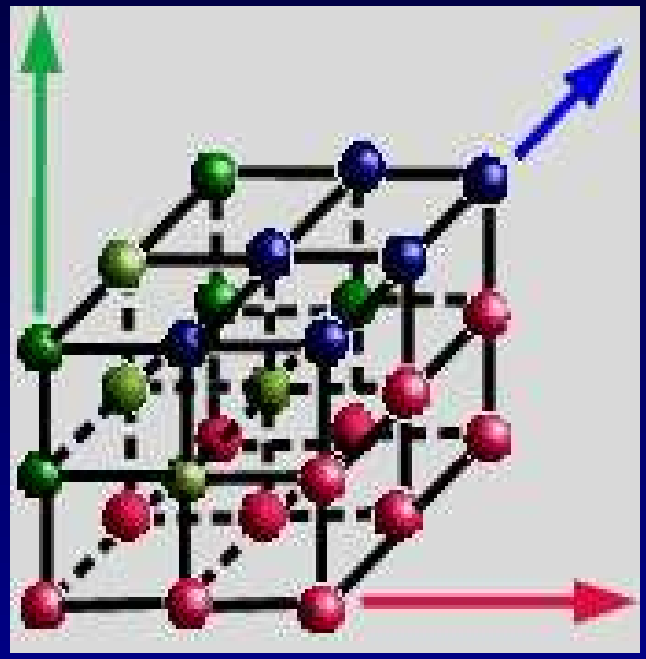
OpenVL

GRC 2003

A Library for Handling and Processing Volumetric Data



Sarang Lakare

Arie Kaufman

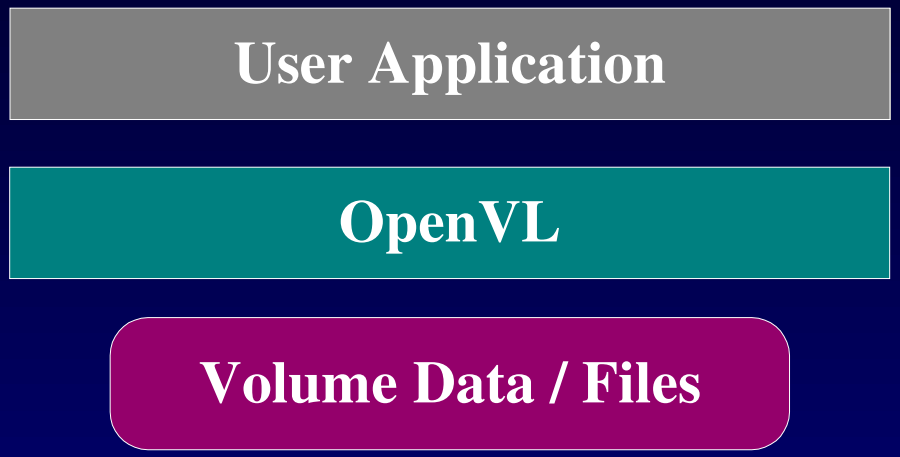
Motivation

- Standard framework for handling volumetric data
  - Accessing the data
  - Reading / Writing to different file formats
- A platform for collaboration

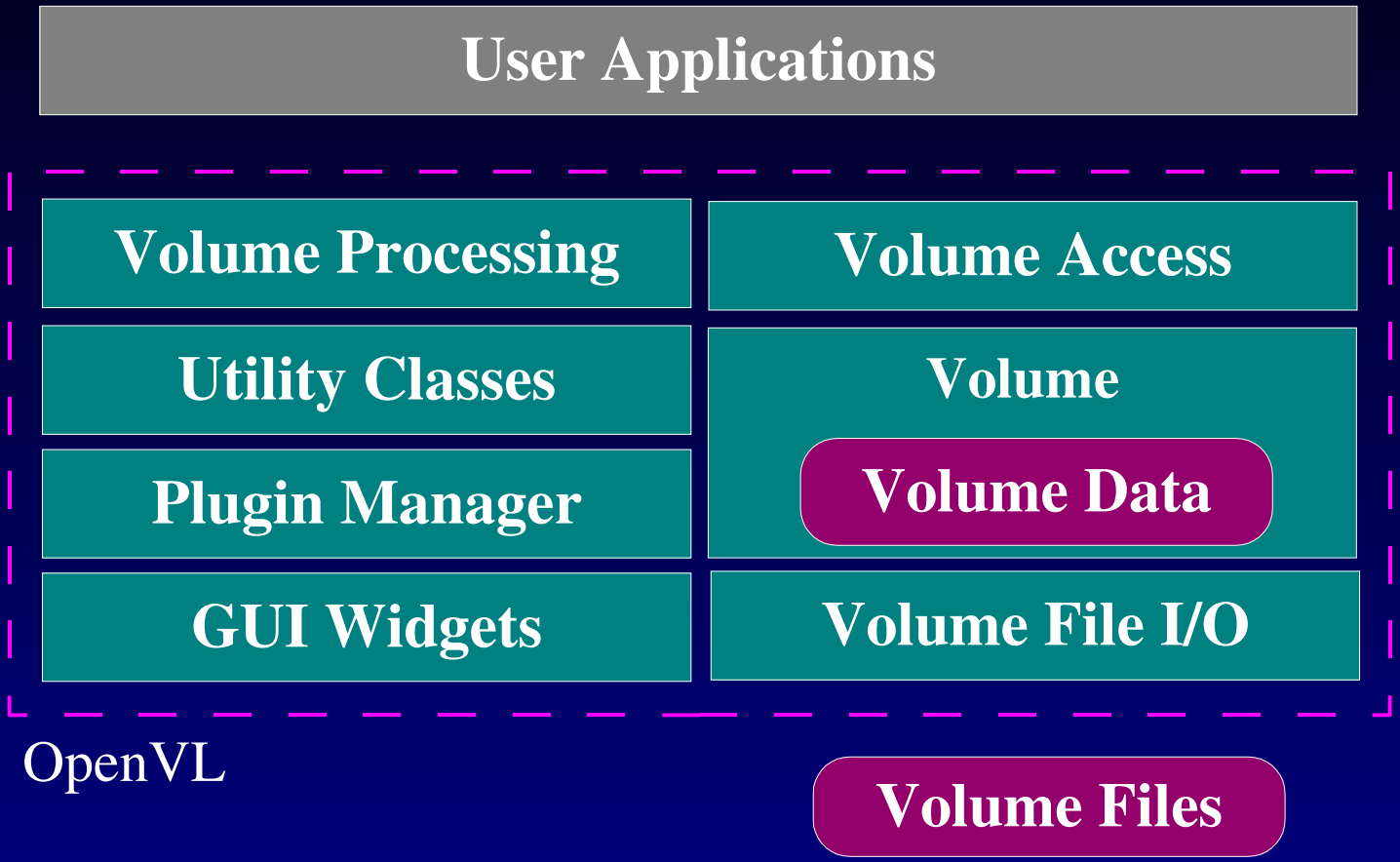
Key Objectives

- Uniform volume access API
- Different layouts for data
- Modular, extensible design
- High performance
- Ease of use
- Open source

Overview



Overview


User Applications
Volume Processing
Volume Access
Utility Classes
Volume
Plugin Manager
Volume Data
GUI Widgets
Volume File I/O
OpenVL
Volume Files

Volume Unit

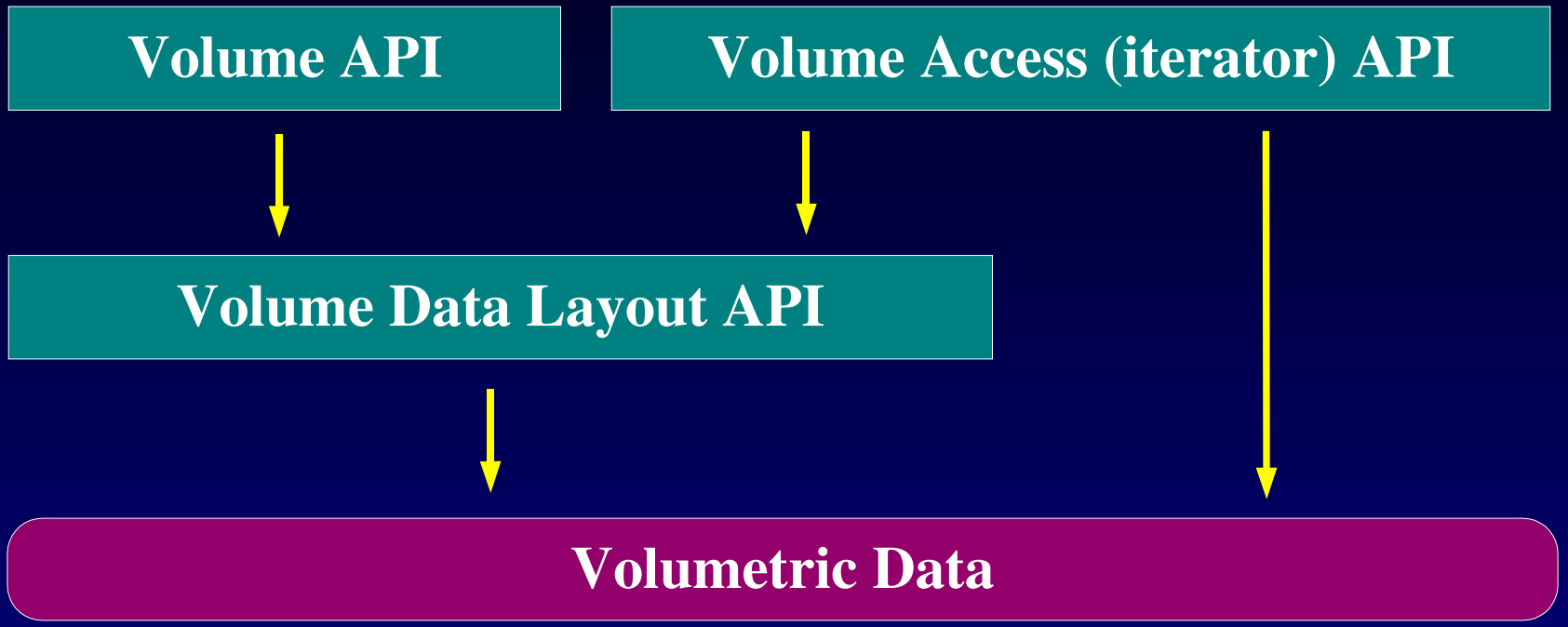
Purpose :

- Holds the volumetric data
- Provides access to the data

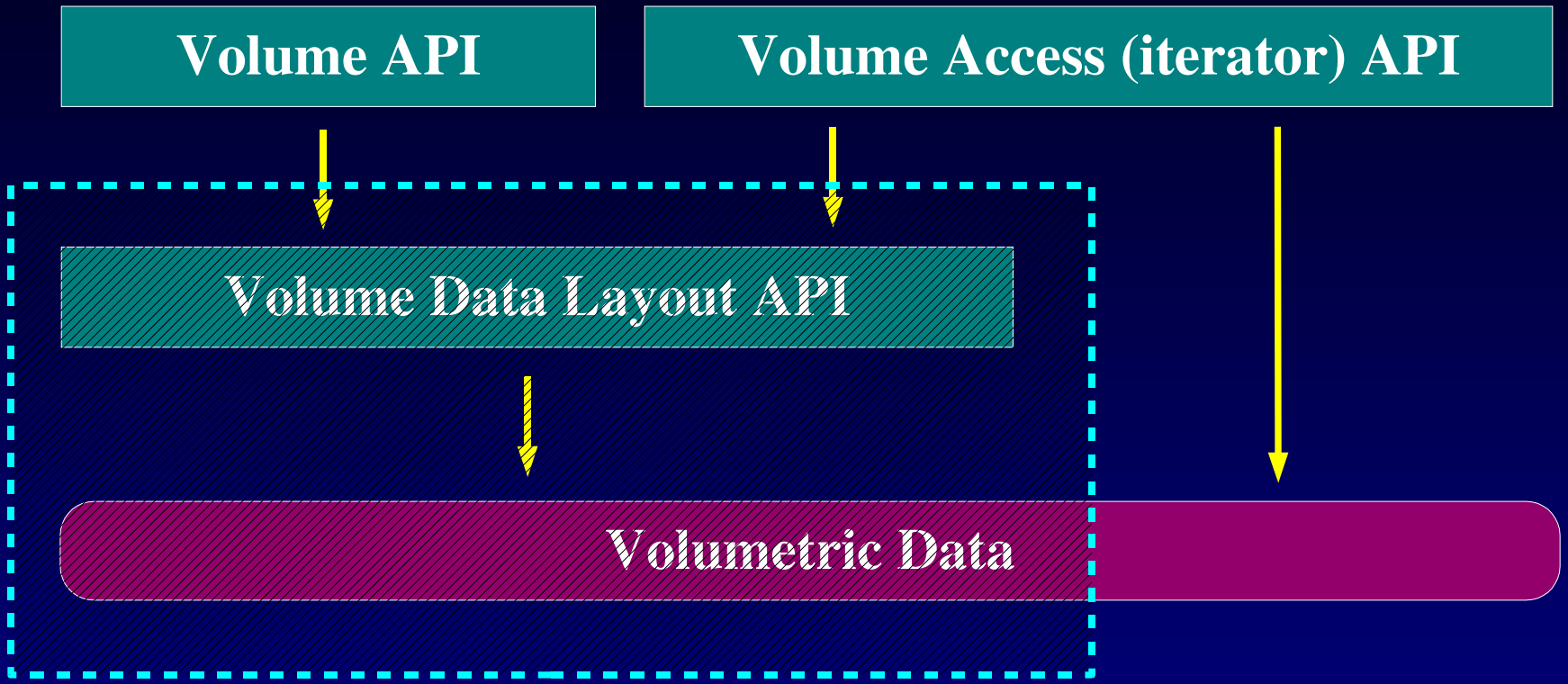
Goals :

- Uniform volume access API
- Allow different data layouts
- Tansparent layout mechanism

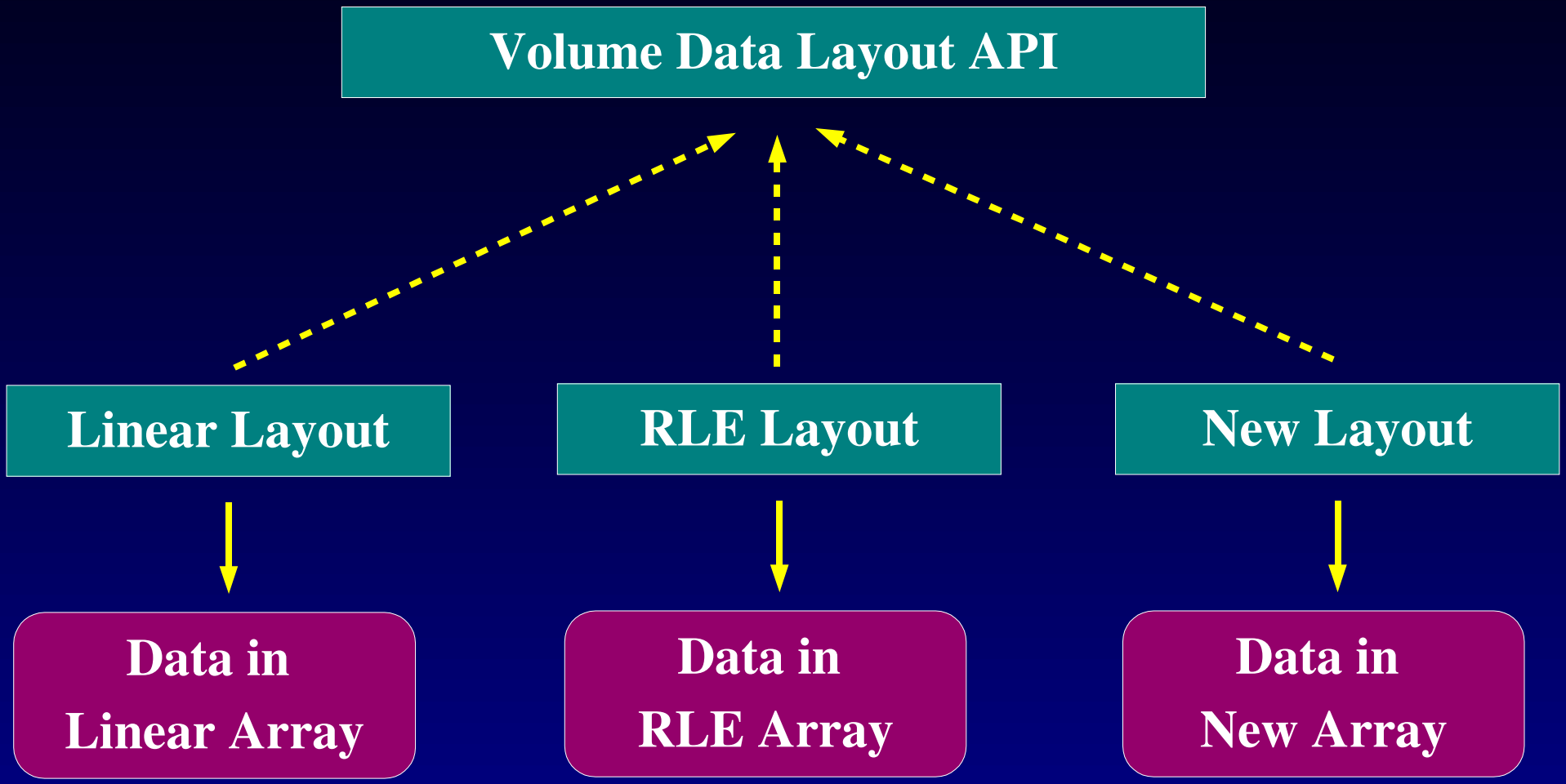
Volume Unit


Volume API
Volume Access (iterator) API
Volume Data Layout API
Volumetric Data

Volume Unit


Volume API
Volume Access (iterator) API
Volume Data Layout API
Volumetric Data

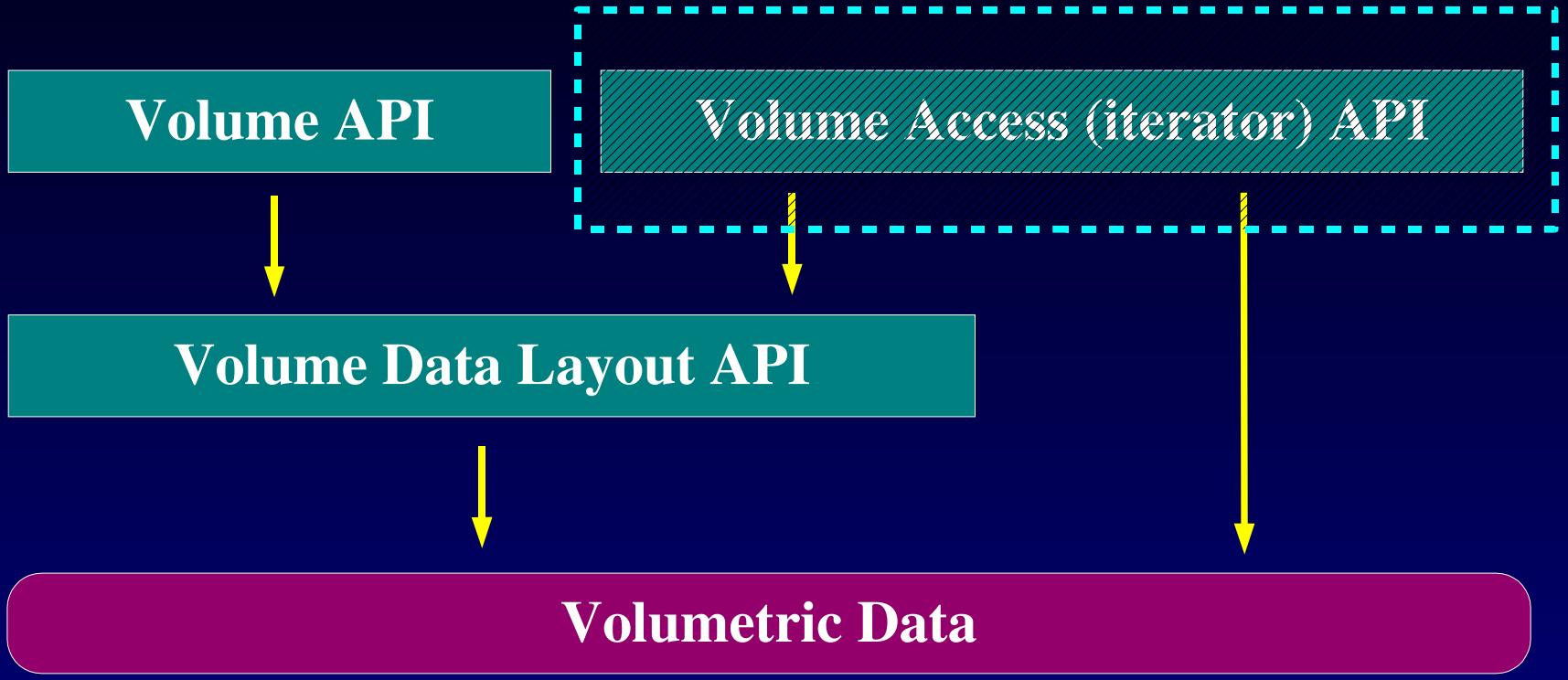
Data Layouts



Data Layout Ideas

- Compressed layouts
- Disk based layouts
- Disk based cached layout
- Network based layout
- Internet based layout
- Texture memory based layout

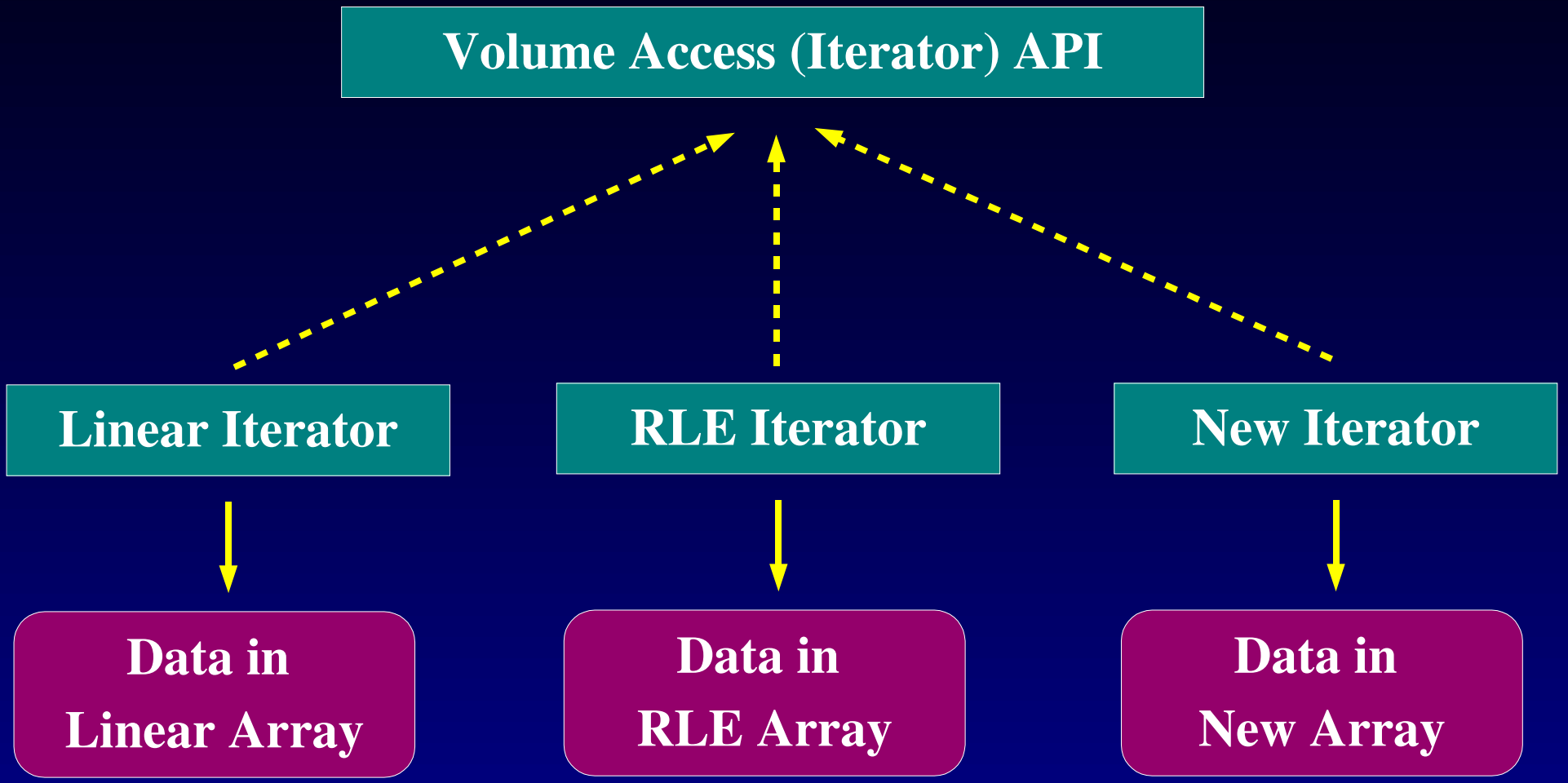
Volume Unit


Volume API
Volume Access (iterator) API
Volume Data Layout API
Volumetric Data

Data Access

- Volume access API : *Iterators*
  - Avoid direct use of pointers
  - More robust code
  - Code is much more readable
  - Easy code profiling

Data Access


Volume Access (Iterator) API
Linear Iterator
RLE Iterator
New Iterator
Data in Linear Array
Data in RLE Array
Data in New Array

Data Access

Iterator properties :

- Always at a grid voxel
- Always inside a volume

Data Access

Iterator API :

- Position change operations
  - `next()`
  - `nextXYZ()`
  - `moveTo(new_position)`
  - `prev()`
  - `prevXYZ()`

Data Access

Iterator API :

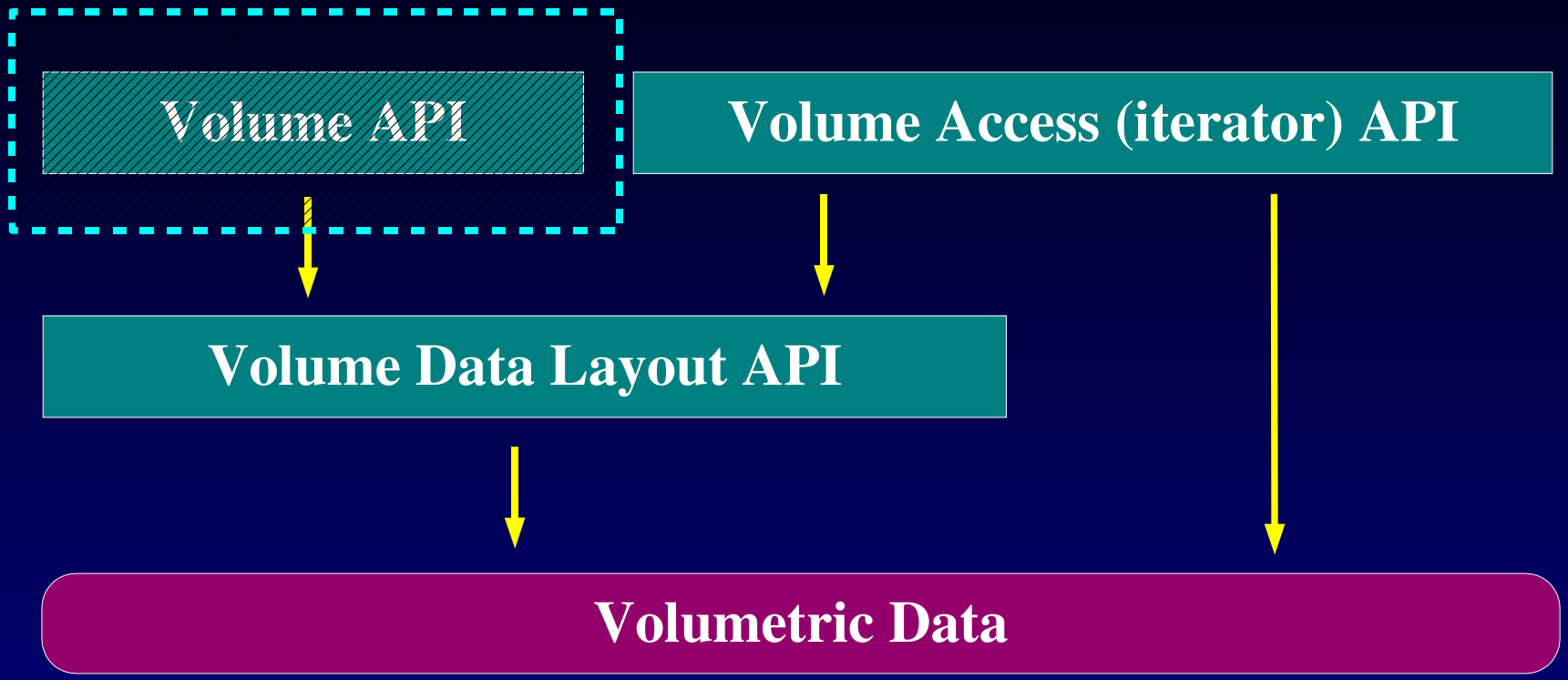
- Value query operations
  - `get()`
  - `getRelative(offset)`
  - `set()`
  - `setRelative(offset)`

Data Access

Iterator API :

- No bounds checking operations
  - `nextNBC()`
  - `prevNBC()`
  - `getRelativeNBC(offset)`
  - `setRelativeNBC(offset)`

Volume Unit


Volume API
Volume Access (iterator) API
Volume Data Layout API
Volumetric Data

Volume Class

- Implements the Volume API
- Class name : `vlVolume`
- Volume objects are the “volumes”

Volume Class

Volume API :

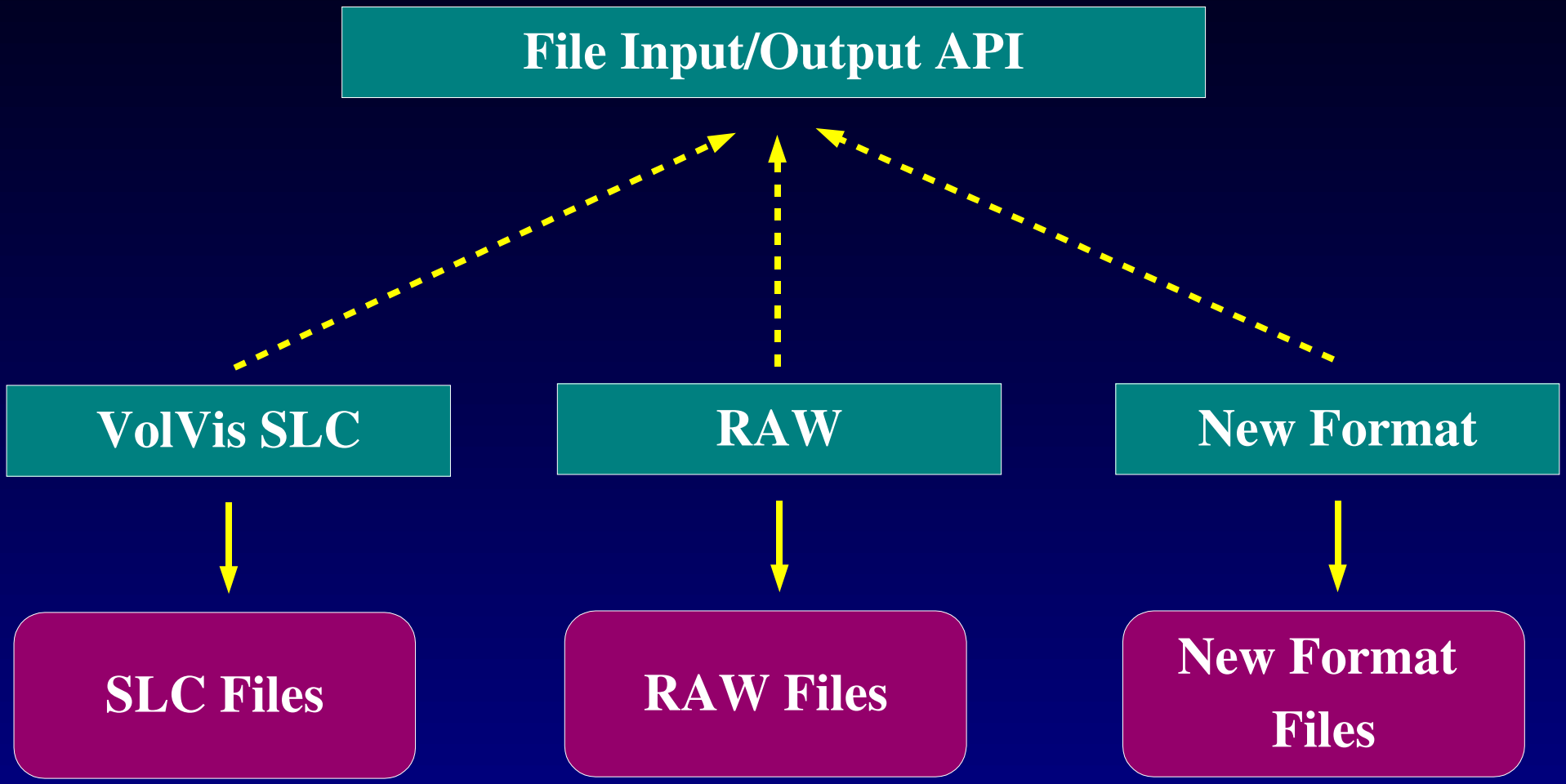
- Volume information
- Metadata for each volume
- File I/O function calls

Volume File I/O

Goals:

- Provide API for reading data from files
- Provide API for writing data to files

Volume File I/O



Volume File I/O

Volume File I/O API :

- For reading
  - `readInfo()`
  - `readData()`
- For writing
  - `writeInfo()`
  - `writeData()`

Volume File I/O

Volume File I/O API :

- File Extensions
  - `getFileExtensions()`

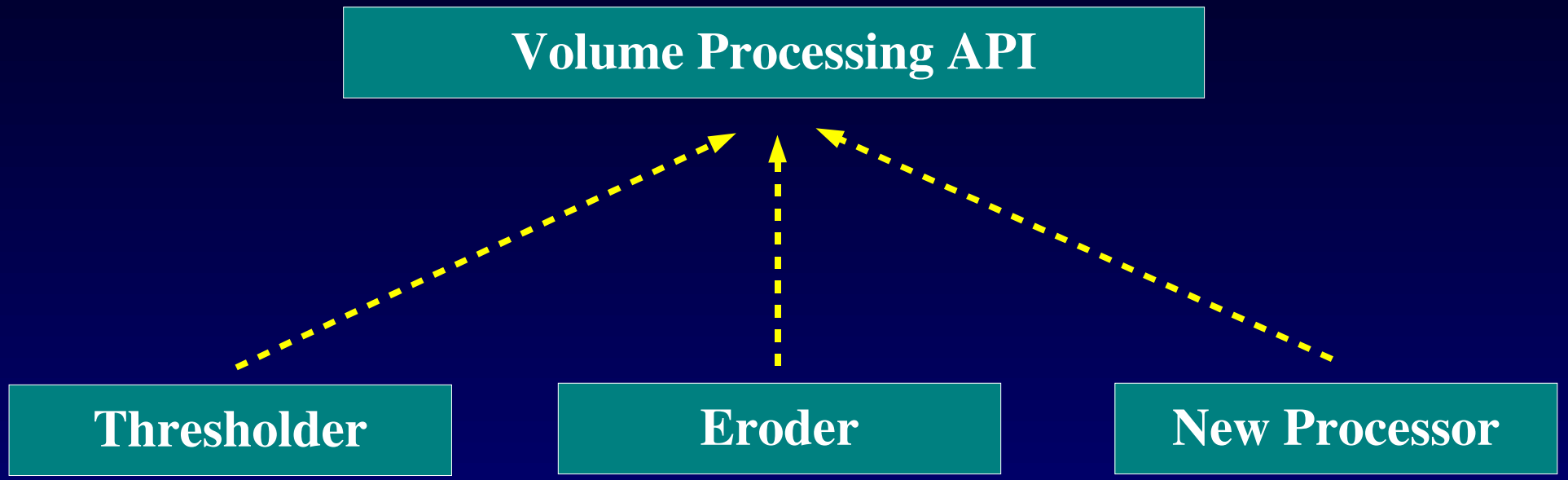
Utility Classes

- vlSlice
- vlTxFunction
- vlHistogram
- vlClock
- Math :
  - vlTriple, vlVector, vlNormal
  - vlMatrix
  - vlDim, vlUnit, vlPoint

Volume Processing

- Any task on a volume
- Image processing
- Volume rendering

Volume Processing



Volume Processing

Volume Processing API:

- `config()`
- `setVolume()`
- `run()`
- `results()`

Volume Processing

`vlVarList` Class

- `set(name, value)`
- `get(name, value)`

Implementation

Goals:

- Fast
- Ease of use
- Hiding templates

Implementation

System :

- Linux OS
- GNU C++ compiler

Implementation

Multiple data types :

- **Use of templates**
- **Volume Data Layout API is templatized**

Implementation

Shared Library (.so files) :

- Dynamically linked library
- Linking at run time
  - Applications can use updated version
  - Reduces memory footprint

Implementation

Plugins:

- Everything in OpenVL is a plugin
- The core library has very little functionality

Implementation

Plugins:

- Dynamic plugins
  - All plugins are dynamic plugins
  - Shared library file (.so file)
  - Loaded at run time when needed

Implementation

Plugins:

- Plugin manager
  - Keeps track of available plugins
  - Loads plugins as and when needed
- Trader
  - Looks for and returns the requested plugin

Example Code

- Creating a volume

```
vlVolume *vol = new vlVolume(vlDim(50,40,20), UnsignedInt8);
```

Example Code

- File input / output:

```
vlVolume *vol = new vlVolume();
vol->read("sample.slc");
vol->write("sample.raw");
```

Example Code

- Accessing volume information:

```
cout << "Dimensions : " << vol->dim() << endl;
cout << "Voxel units : " << vol->units() << endl;
cout << "Bytes per voxel: " << vol->bytesPerVoxel()      <<
  endl;
```

Example Code

- Accessing volume data:

```
vlVolume *vol = new vlVolume(vlDim(50,40,20), UnsignedInt8);
vlVolIter<uint8> iter(vol);
uint32 count(0);
while(!iter.end()) {
  if(iter.get() != 0) ++count;
  iter.next();
}
cout << "No. of non-zero voxels : " << count << endl;
```

Example Code

- Using a volume processor:

```
vlVolume *vol = new vlVolume();
vol->read("sample.slc");
vlVolume *maskVol = new vlVolume(vol->dim());
vlPlugin *proc= vlKernel::trader()->getPlugin("VolProcessor","Thresholder");
if (!proc) return;
proc->setVolume(vol);
proc->config()->set(“maskVol”, maskVol);
proc->config()->set("ThresholdLow", (uint8)(100));
proc->config()->set("ThresholdHigh", (uint8)(150));
if(!proc->run()) return;
std::cout << "Thresholding done." << std::endl;
maskVol->write("sample_thresholded_mask.slc");
```

Future Work

- More plugins!
- Layouts
  - Texture memory based layout
  - Networked layout
  - Disk-Cache layout

Future Work

- Volume Rendering API
  - Unified API for different engines
  - Similar to OpenGL and VLI (Volume Pro's API)

Misc. Stuff

- Source is CVS controlled
- Available freely under LGPL license from the OpenVL website
- Mailing list for users
- Bug tracking
- Encouraged to get involved!

Misc. Stuff

Availability:

- Source
  - You need gcc 2.96 or gcc 3.2
- Binaries
  - Two sets of RPM packages are available for Linux distributions based on
    - » gcc 3.2 (Mandrake 9.x, RedHat 9.0, etc.)
    - » gcc 2.96 (Mandrake 8.x, RedHat 7.3, etc.)

Misc. Stuff

Statistics (4 months):

- ~ 300 downloads
- ~ 5000 page hits
- ~ 25,000 lines of code
- Plugins:
  - 5 File I/O
  - ~ 10 Image processing

Acknowledgements

Contributors:

- S. Yokum-Stover
- K. Hegeman
- M. Lakare

More Information

OpenVL Website:

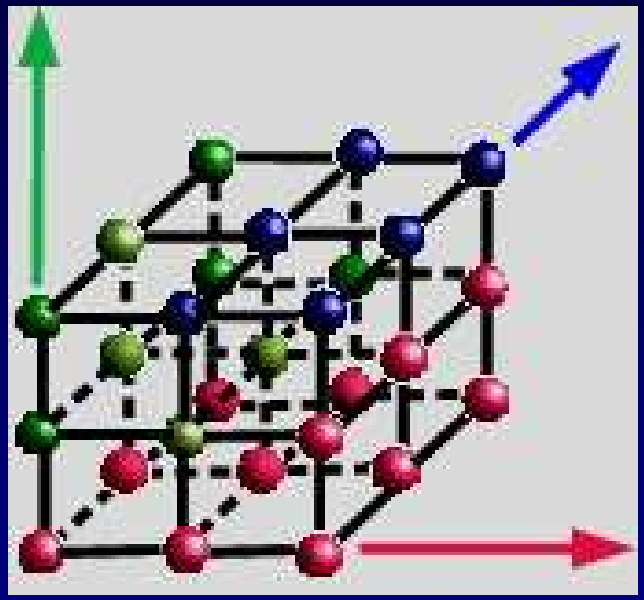
http://openvl.sourceforge.net

&

http://www.cs.sunysb.edu/~vislab/projects/openvl

OpenVL

The Open Volume Library



Thank You!