Portability

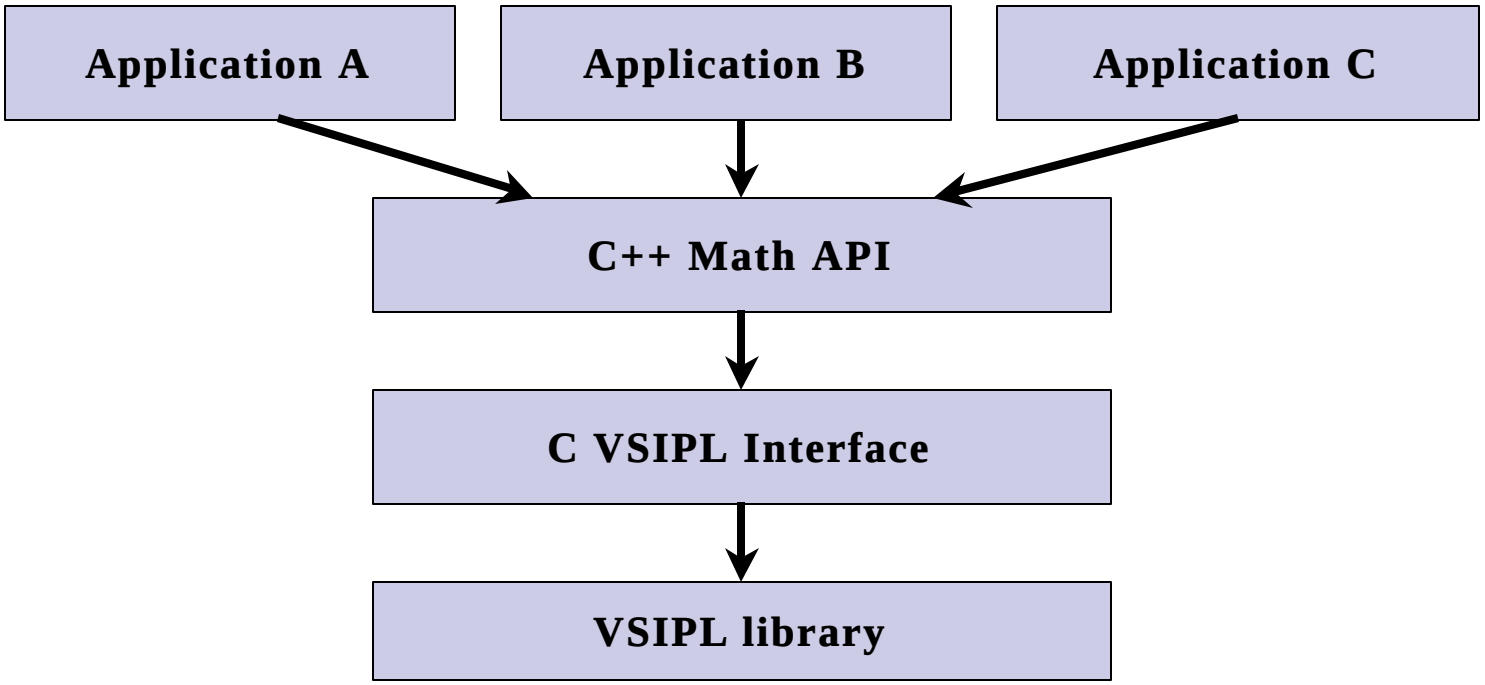
- **Operating System and Architecture Independence**
  - **Solaris - Ultra Sparc**
  - **Linux - PPC/Pentium**
  - **VxWorks - PPC-Altivec**
  - **MAC OS X - PPC-Altivec**
  - **Windows - Pentium**

- **If the machine supports a C++ compiler….**

PENNSTATE ARL

Reusability

- First signal processing application – 6 months development
- Second (comparable) application -- 6 weeks development
  - Original development of second application in ADA - minimum of 6 months.



PENNSTATE ARL

Rapid/Stable Applications Development

Powerful Expressibility:

```
D = C / 2.0 - A + B * A;
A = B ^ 3.5;
A = B.abs();
A = B.sin();
A = B.var();
A = B.fft().fftshift().abs();              // A = abs(fftshift(fft(B)));
A = (B.fft() * (C.fft().conj())).ifft();
A = B.xcorr(C );
```

Memory Management:

most dynamic memory usage, including VSIPL, is transparent

PENNSTATE ARL

Shared Objects

- **Signal processing objects are generated once and are automatically shared:**
  - **FFT coefficients**
  - **Window Functions**
  - **Filters**
  - **Tuners**

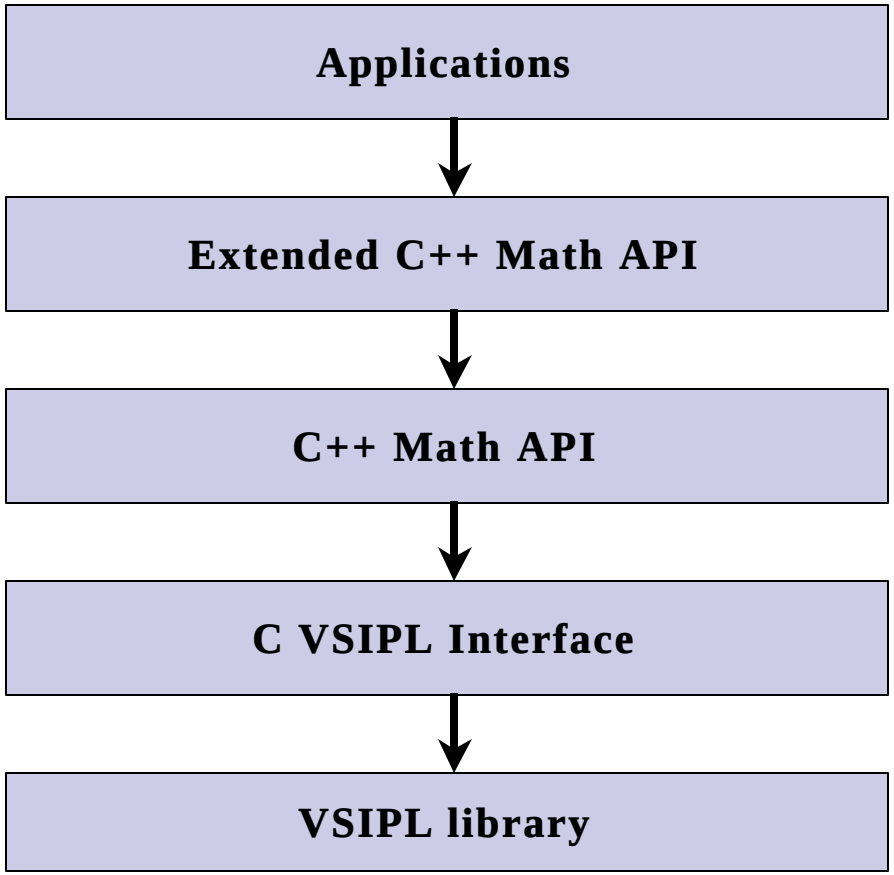
PENNSTATE ARL

Standard Template Library

- **The C++ STL containers provide an efficient means to organize, access, and process information/data.**
- **Most modern large-scale C++ signal processing development efforts will use the STL extensively.**
- **The math API interface can be made to mirror STL operation to provide a more intuitive use of basic API operations.**

Extensibility

Inherit most functionality, modify some, add other functions



This could be Company Proprietary

This could be unlimited distribution

VSIPL Transparency .... Memory Management

- VSIPL codelet to perform A = B * C for vectors of 512 samples

```
float_complex A[512], B[512], C[512];
        // data must be placed in B and C
vsip_cblock_f  *Ab, *Bb, *Cb;
vsip_cvview_f  *Av, *Bv, *Cv;
        // must bind user data to blocks
Ab = vsip_cblockbind_f(A, 0, 512, 0);   // omitting error checking
Bb = vsip_cblockbind_f(B, 0, 512, 0);
Cb = vsip_cblockbind_f(C, 0, 512, 0);
        // must create view to blocks
Av = vsip_cvbind_f(Ab, 0, 1, 512);       // omitting error checking
Bv = vsip_cvbind_f(Bb, 0, 1, 512);
Cv = vsip_cvbind_f(Cb, 0, 1, 512);
```

PENNSTATE ARL

VSIPL Transparency .... Memory Management (cont)

- VSIPL codelet to perform A = B * C for vectors of 512 samples (cont)

```
            // must admit blocks to VSIPL memory space
 vsip_cblockadmit_f(Ab, 0);              // omitting error checking
 vsip_cblockadmit_f(Bb, 0);
 vsip_cblockadmit_f(Cb, 0);
            // finally,  we get to the multiply
vsip_vmul_f(Bv, Cv, Av);
            // must destroy blocks, views, etc:
vsip_cvalldestroy_f(Av);
vsip_cvalldestroy_f(Bv);
vsip_cvalldestroy_f(Cv);
```

VSIPL Transparency .... Memory Management (cont)

- VSIPL codelet to perform A = B * C for vectors of 512 samples (cont)

  Clearly, VSIPL != VSIMPLE
  Clearly, direct VSIPL coding is prone to memory leaks/errors

- The "CVector" equivalent code is:

```
CVector A(512), B(512), C(512);
        // something puts data in B and C
A = B * C;
```

PENNSTATE ARL

Performance (complex data)

Function	Size	Application	VSIPL kernel	Efficiency
A = B * C	256	13.4 usec	5.3 usec	40%
	512	38	18	47%
	1024	59	32	54%
	4096	351	205	58%
A = B.fft()	256	81.5	67.7	80%
	512	192	163	85%
	1024	426	383	90%
	4096	1949	1746	90%

PENNSTATE ARL

Performance (cont)

Function	Size	Application	VSIPL kernel	Efficiency
A = B.fir(F)	256	524 usec	498 usec	95%
	512	1011	974	96%
	1024	2040	1972	97%
	4096	9048	8702	97%
A = B.xcorr(C)	256	306	230	75%
	512	616	486	79%
	1024	1529	1228	80%
	4096	7643	6090	80%

PENNSTATE ARL

Debugging/Profiling/Tuning

- **The API can provide:**
  - **Development and Performance modes of operation**
  - **An ability to view detailed state information for math objects**
  - **Easy mobility of object data to/from objects and the file system**
  - **Profiling of the application software to facilitate performance tuning**
  - **Exception handling interface**