Average Attendee

First

Second

Third

Python Buffer Interface

Travis E. Oliphant

NumPy in Python

- Getting NumPy into Python has been a long-term goal
- We have not wanted to commit to the release schedule
- Nobody has stepped up to argue our case with other Python developers.
- Now NumPy is even “bigger” than it was in the past

NumPy in Python

- Tactical change
  - Get the "structure" of NumPy into Python 3.0 via the buffer interface
  - Start with changes to Python 3.0 and then backport additions to Python 2.6
  - Eventually, the demand for some of the rest of NumPy will probably increase

Array Interface

- Numeric, Numarray, NumPy all use the array interface to share data
- An attribute-based interface without any direct support in the language
- We realized it could act as a replacement of the buffer protocol (interface)

PEP 3118

- After SciPy 2006, the idea of the buffer protocol was hatched.
- I was side tracked for much of 2006-2007 academic year on other issues
- With the help of Carl Banks and Greg Ewing and others on py3k-dev, PEP 3118 grew out of my early efforts.

PEP 3118 Overview

- Re-defines the tp_as_buffer function pointer table for every PyTypeObject
- Adds PyMemoryViewObject (memoryview in Python) --- will be the first object in Python to support multi-dimensional slicing.
- Expands the struct module with new character-based syntax.
- Creates new C-API functions to make common things simple.

Timeline

- Happening now. If you'd like to help the Google Sprint is next week (but I'm moving next week).
- MemoryViewObject needs work
- Struct module needs work
- Bug-fixes on what's already implemented
- Python 3.0 is due for alpha release at the end of August.

tp_as_buffer

```
typedef struct {
    readbufferproc bf_getbuffer;
    writebufferproc bf_writebuffer;
    segcountproc bf_getsegcount;
    charbufferproc bf_getcharbuf;
} PyBufferProcs
```

```
typedef struct {
    getbufferproc bf_getbuffer;
    releasebufferproc bf_releasebuffer;
} PyBufferProcs
```

GetBuffer

```
typedef int (*getbufferproc)
    (PyObject *obj, PyBuffer *view, int flags)
```

Argument	Explanation
obj	Object being queried
view	View structure to fill
flags	What kind of buffer is requested
return	-1 if error; 0 if success

```
typedef void (*releasebufferproc)
    (PyObject *obj, PyBuffer *view)
```

PyBuffer structure

```
struct bufferinfo {
     void *buf;
     Py_ssize_t len;
     Py_ssize_t itemsize;
     int readonly;
     int ndim;
     char *format;
     Py_ssize_t *shape;
     Py_ssize_t *strides;
     Py_ssize_t *suboffsets;
     void *internal;
} PyBuffer;
```

PyBuffer Explanation

Member	Description
buf	Pointer to start of memory
len	Total number of bytes
itemsize	Number of bytes per element
readonly	Is memory read-only?
ndim	Number of dimensions (>=0)
format	Struct-style syntax describing memory
shape	Size in each dimension
strides	Number of bytes to skip to get to the next element in each dimension
suboffset	If >=0, then value is a pointer in this dimension. This tells how many bytes to skip after de-referencing. to get to the start of the next dimension.
internal	For use by object.

Flags

Flag	Description
PyBUF_SIMPLE	Only simple (ptr, len) interface is requested
PyBUF_CHARACTER	Character buffer requested
PyBUF_WRITEABLE	A writeable buffer is needed
PyBUF_LOCKDATA	A locked, read-only buffer is needed
PyBUF_FORMAT	Make sure format is provided
PyBUF_ND	Make sure shape information is provided
PyBUF_STRIDES	Make sure stride information is provided
PyBUF_INDIRECT	Provide sub-offsets if available
PyBUF_{C,F,ANY}_CONTIGUOUS	Make sure buffer is C, Fortran, or either-one contiguous

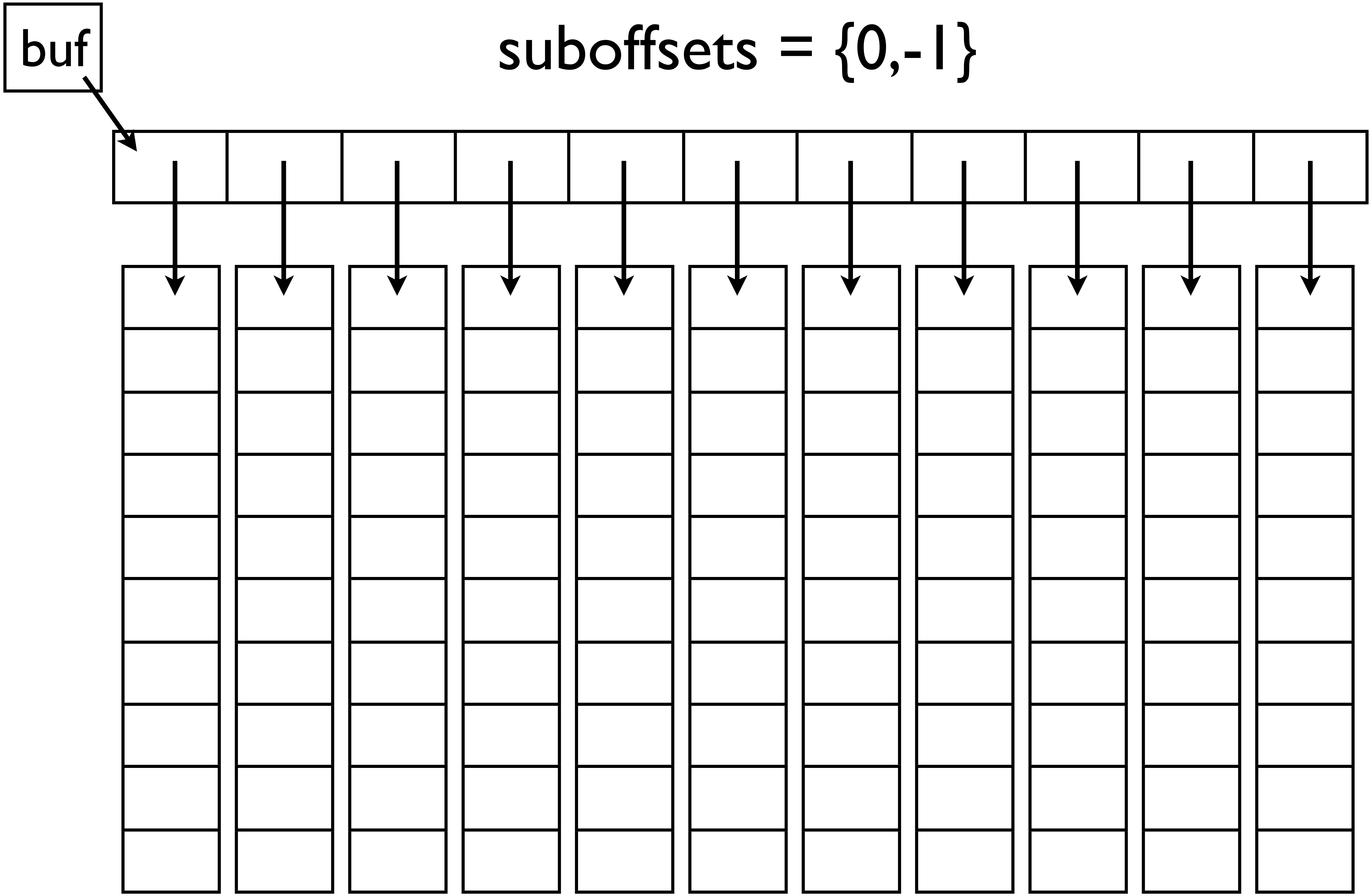
Pointer in-direction

```
void *get_item_pointer(int ndim, void *buf, Py_ssize_t *strides,
                       Py_ssize_t *suboffsets, Py_ssize_t *indices) {
    char *pointer = (char*)buf;
    int i;
    for (i = 0; i < ndim; i++) {
        pointer += strides[i] * indices[i];
        if (suboffsets[i] >=0 ) {
            pointer = *((char**)pointer) + suboffsets[i];
        }
    }
    return (void*)pointer;
}
```

```
(void *) PyBuffer_GetPointer
          (PyBuffer *view, Py_ssize_t *indices);
```

Suboffsets

suboffsets = {0,-1}



New C-API

Name	Purpose
PyObject_CheckBuffer	Make sure getbuffer is present
PyObject_GetBuffer	Call getbuffer if available
PyObject_ReleaseBuffer	Call releasebuffer if available
PyBuffer_FromContiguous	Copy to a buffered memory from contiguous memory
PyBuffer_ToContiguous	Copy from a buffered memory to contiguous memory
PyObject_CopyData	Copy data between two objects with the buffer interface
PyBuffer_IsContiguous	True if buffer is contiguous (either C or Fortran depending on argument)
PyBuffer_FillContiguousStrides	Fill a strides array belonging to a contiguous N-d array.
PyBuffer_FillInfo	Fill the PyBuffer structure for simple 1-d buffer
PyMemoryView_Check	Make sure the object is a MemoryView object
PyMemoryView_GetContiguous	Get a contiguous MemoryView object from another object
PyMemoryView_FromObject	Get a MemoryView object from an object using the buffer interface
PyMemoryView_FromMemory	Get a MemoryView object from a PyBuffer struct.

MemoryView Object

```
typedef struct {
    PyObject_HEAD
    PyObject *base;
    PyBuffer view;
} PyMemoryViewObject;
```

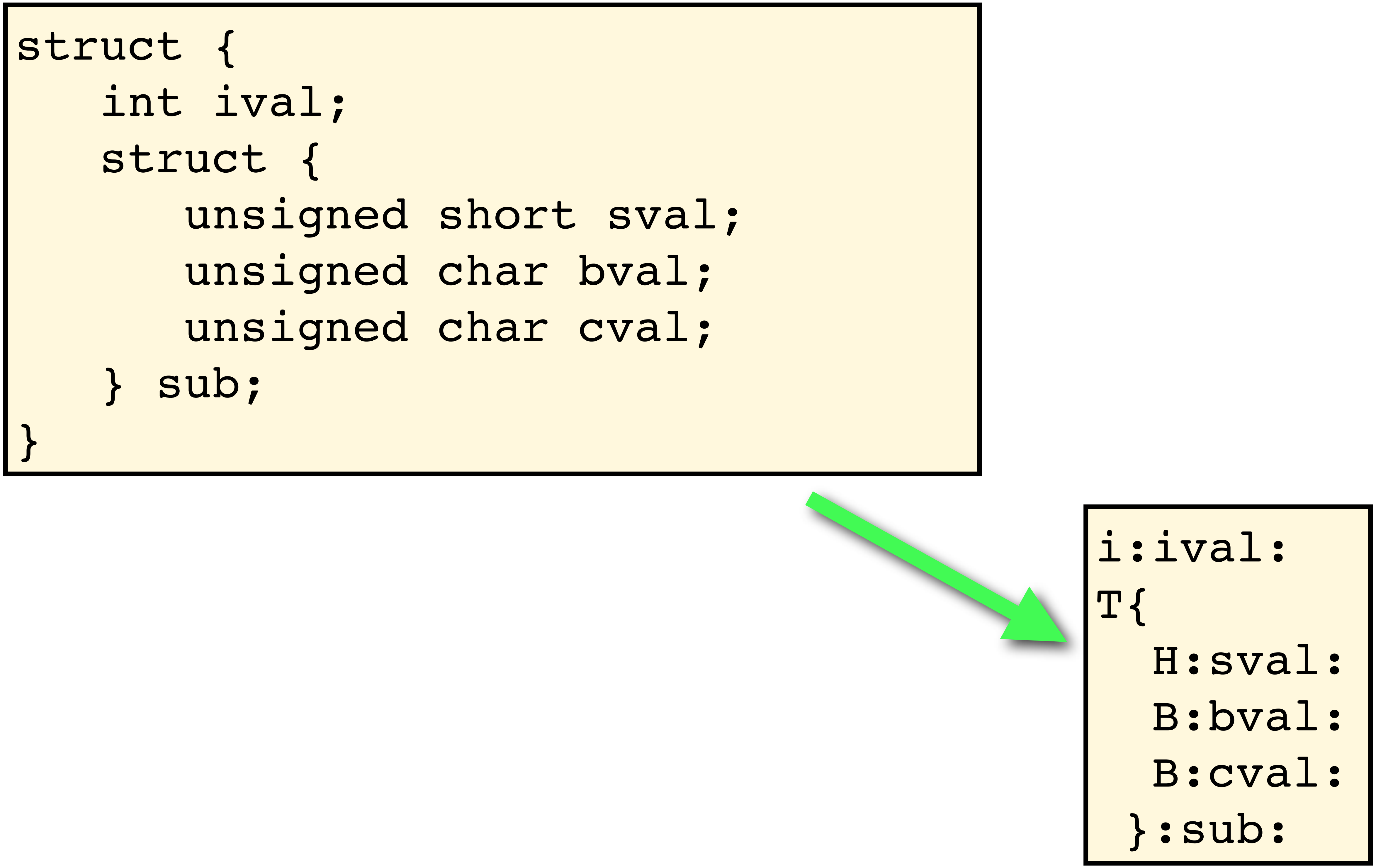
Methods	Purpose
__getitem__	Multi-dimensional slicing
__setitem__	Multi-dimensional sliced setting
tobytes	Create contiguous bytes
tolist	Create a (nested) list

Attributes
format
itemsize
shape
strides
suboffsets
size
readonly
ndim

Struct-string syntax

Char.	Description
t	bit (number before states how many bits)
?	platform _Bool
g	long double (unpacks to ctypes object)
c	ucs-1 (latin-1) (unpacks to unicode)
u	ucs-2 (unpacks to unicode)
w	ucs-4 (unpacks to unicode)
O	Python Object pointer
Z	Complex of whatever the next specifier is (unpacks to complex)
&	Pointer to whatever the next specifier is (unpacks to ctypes void_p)
T{}	Structure (detailed layout should be inside {}) (unpacks to ctypes)
(k1,k2,...,kn)	Multi-dimensional array of whatever comes next (unpacks to nested list)
:name:	Optional name of the preceeding element
X{}	Pointer to a function (optional signature inside of {} with any return value preeceeded by -> and placed at the end)

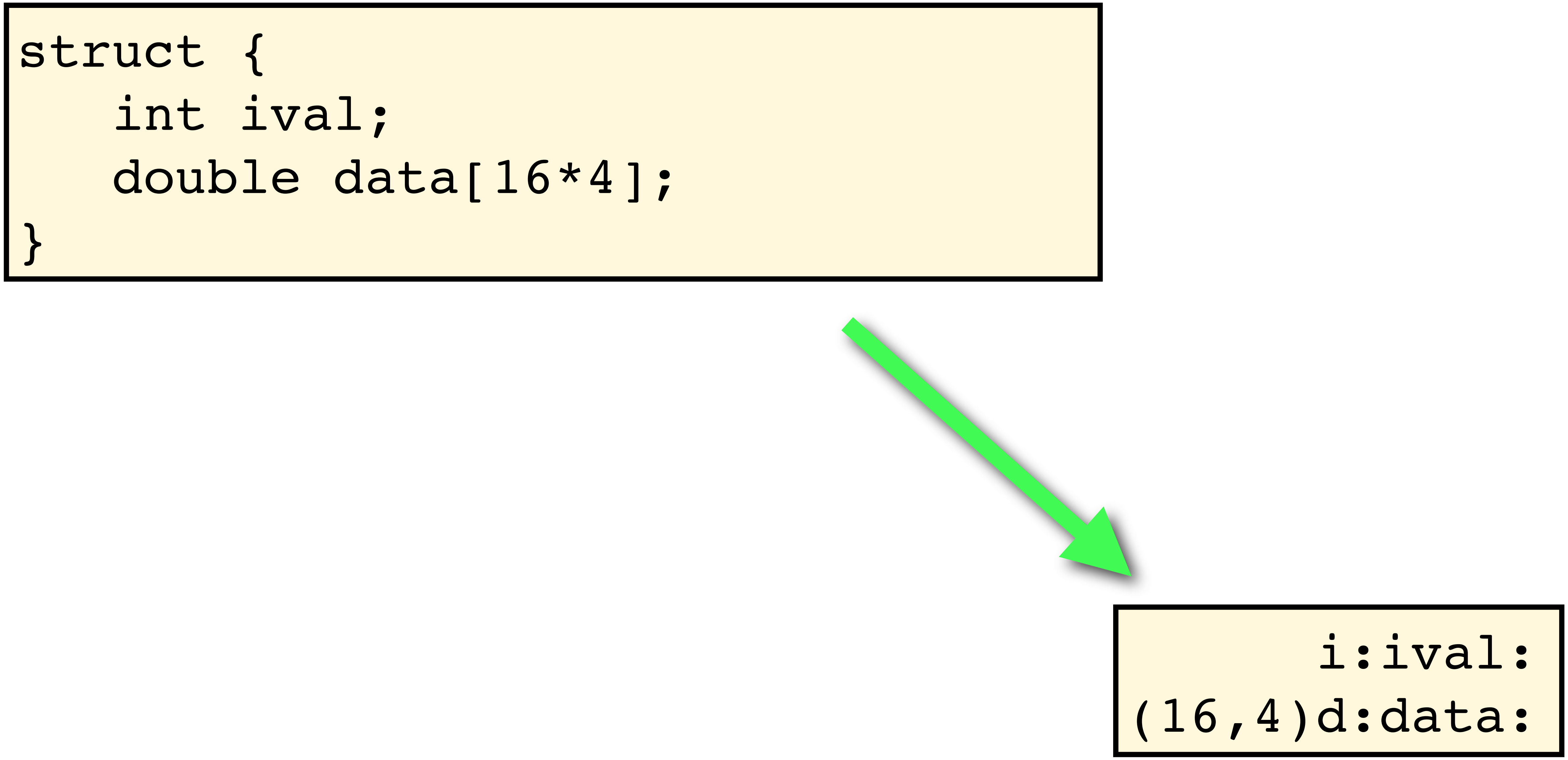
Struct examples



```
struct {
   int ival;
   struct {
      unsigned short sval;
      unsigned char bval;
      unsigned char cval;
   } sub;
}
```

```
i:ival:
T{
  H:sval:
  B:bval:
  B:cval:
 }:sub:
```

Struct examples



```
struct {
    int ival;
    double data[16*4];
}
```

```
      i:ival:
(16,4)d:data:
```

Implications

- Should have standard way to share data among media libraries
- Should have standard way to share arrays among GUIs
- Should increase adoption of NumPy-like features by wider Python community
- Powerful struct/ctypes connection
- Maybe automatic compiled function call-backs using function-pointer data

Interested?

- Google code Sprints (Aug. 22-25)
- Contact me for some Guidance before Tuesday morning (Aug. 21)