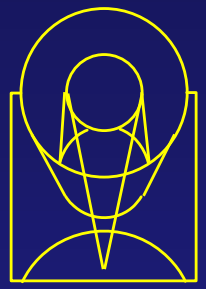
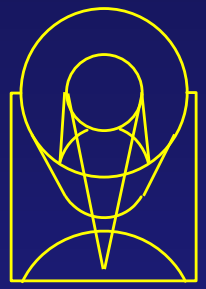


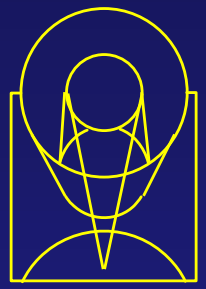
numarray

Perry Greenfield
Space Telescope Science Institute



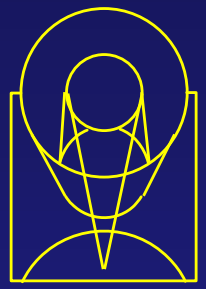
Who: Cast of Developers

- Perry Greenfield: original framework
- Todd Miller: most of past and ongoing development, chararray, memmap.
- JC Hsu: recarray
- Rick White: design and early coding
- Jochen Krupper: updates to manual, conversion to Python doc format, work on libraries.
- Phil Hodge: original adaptation of manual
- Paul Dubois: Porting of MA package (soon)
- The past Numeric Developers!
  - Jim Hugunin, Konrad Hinsen, David Ascher, Paul Dubois, Travis Oliphant

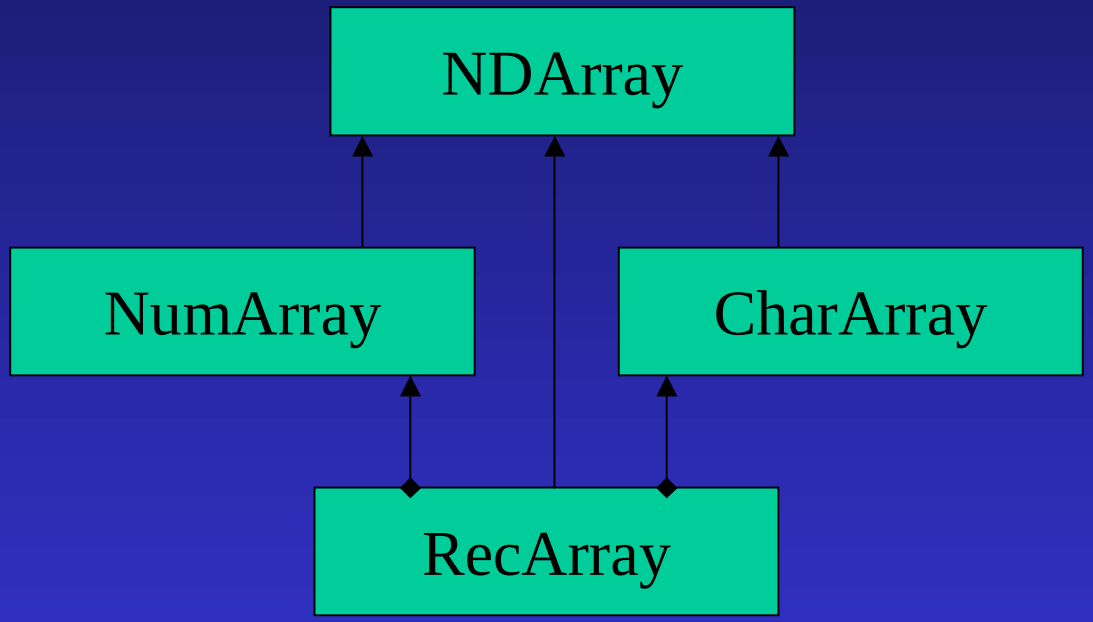


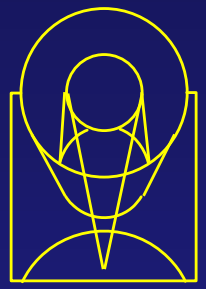
What is numarray?

- Numeric descendant and replacement
- Features:
  - Sub-classable arrays
  - User specifiable buffer, offset, and basic stride.
  - Operation on misaligned or byte-swapped arrays.
  - Operation on memory-mapped arrays.
  - Generation of C-code from templates
  - Increased use of Python in implementation
  - More flexible IEEE error handling
  - Index arrays



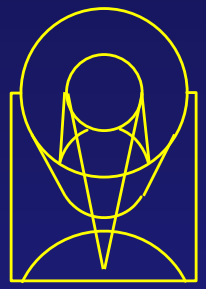
Numarray Class Hierarchy





Where is numarray now?

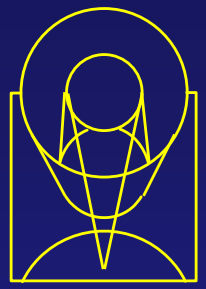
- Virtually all the base functionality of Numeric exists in numarray.
- Has nearly all the desired new functionality.
- Generally as fast (if not faster) for megabyte sized arrays (but much slower for small--< 10K elements).
- FFT, RandomArray, LinearAlgebra libraries ported.
- Corresponding documentation (except for C-API) exists (i.e., manual).



The BIG Question: Why?

We (STScI) needed capabilities that Numeric could not provide…easily anyway. Some background:

- Astronomical community has two large analysis contingents: IRAF and IDL-based.
- STScI trying to develop an environment that takes the best from both areas.
  - PyRAF to run IRAF tasks
  - Python/Numeric for IDL-style development
- But astronomers deal with big data sets: Memory issues!

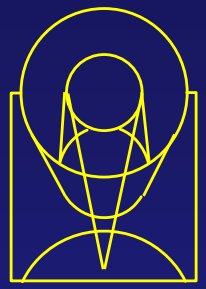


The BIG Question: Why? (cont.)

Manipulating large astronomical images is very memory intensive.

- Hubble Space Telescope now generates 4Kx4K images (64MB).
- Ground-based telescopes have 8Kx8K detectors or larger (256MB!).
- We also have much data in record format (“tables”) with large data files as well (>100MB)

We wish to write Python programs to access such data.



The BIG Question: Why? (cont.)

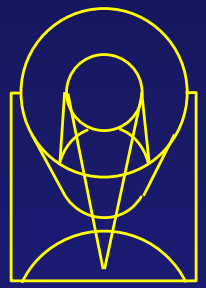
Accessing tables cuts both ways. (Columns may be of different numeric types, or even strings.)

- By column
- By row

Numeric forces you to copy columns to individual arrays.

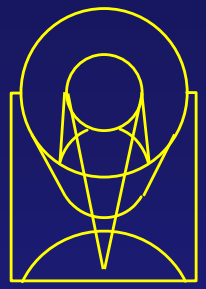
- Wastes memory
- Makes changing rows difficult

Desire generalization: arrays of records



Example RecArray Usage

```
>>> import recarray
>>> a=recarray.array("a"*75, "r,3i,5a", shape=(3,), names="theReal,theInts,theString")
>>> print a
RecArray[
(2.5984589414244182e+020, array([1633771873, 1633771873, 1633771873]), 'aaaaa'),
(2.5984589414244182e+020, array([1633771873, 1633771873, 1633771873]), 'aaaaa'),
(2.5984589414244182e+020, array([1633771873, 1633771873, 1633771873]), 'aaaaa'),
]
>>> a.field("theReal")
array([  2.59845894e+20,   2.59845894e+20,   2.59845894e+20], type=Float32)
>>> a.field("theInts")
array([[1633771873, 1633771873, 1633771873],
       [1633771873, 1633771873, 1633771873],
       [1633771873, 1633771873, 1633771873]])
>>> a.field("theString")
CharArray(['aaaaa', 'aaaaa', 'aaaaa'])
```



The BIG Question: Why? (cont.)

Use memory mapping to reduce memory demands.

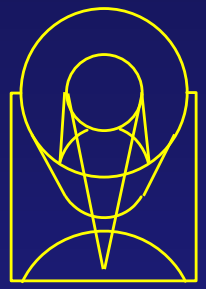
- But that brings new problems, I.e., byte order.
- Very difficult to handle with Numeric.

Record arrays (tables) require being able to construct numeric arrays with odd offsets between elements.

- Alignment problems.
- Also very difficult to handle with Numeric.

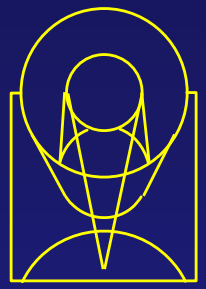
Numeric type conversion creates large temporaries.

These three were the killer issues that ruled Numeric out. But there are others.



Why: Other Numeric issues

- Guido won't accept Numeric into Standard Library
  - Code too hard to understand and maintain.
- Scalar/Array type coercion wastes memory.
- Wasn't subclassible (well, not when we started, anyway).
- Missing unsigned ints.
- More convenient use of index arrays
- Weak IEEE support



How: Implementation

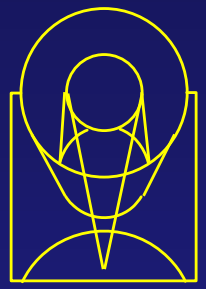
How to handle various representations (type, byteswap, alignment) without impacting speed and memory use?

Studied:

- Functional – easy to implement, but too slow
- Combinatorial – efficient, but enormous code bloat
- Temporaries – fast, but far too memory intensive

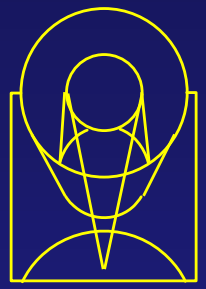
Instead:

- We chose a hybrid of functional and temporaries. When transformations are needed, we do so in blocks, not too big, not too small.



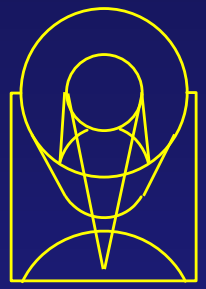
How: Philosophy

- Do as much in Python while optimizing large array performance (our need, after all).
- Defer all other optimizations as late as possible.
  - Small array performance suffers (for now)
  - Indexing performance suffers
- Initially we intended for many incompatibilities
  - “To get things right”
  - But no changes for the sake of changes
  - Eventually most incompatiblities (but not all) were removed or will be removed.
    - There were good reasons for most of Numeric’s design decisions.



How: Planned incompatibilities

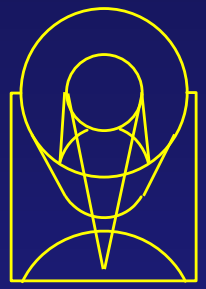
- Scalar-Array coercion rules
  - Operation with scalar of same kind does not automatically promote to scalar's equivalent type.
  - e.g. (Arange(10, type=Int16) * 5).type() à Int16 not Int32
- C API
  - Native API supports byte-swapped, misaligned arrays
  - Emulation API exists for porting Numeric extensions
- Consistent return type for single item indexing
  - Unlike what "manifesto" stated, now leaning towards returning Python scalars, not arrays
    - But simple interface to get rank-0 arrays always
- Types are objects, not just string codes
  - e.g. Int32 vs. 'i'
  - But old code should work in almost all cases



Future Work

(in rough order of our current priorities)

- Backward compatibility problems (as discovered)
- Rank-0 returns from single item indexing [started]
- Document C-API (with many examples) [started]
- IEEE special values getting/setting. [mostly done]
- MA port
- Index array revisions/enhancements
- Mac OS X port
- Add more 3rd party libraries
- Optimization
- Future division support
- Python object array support
- Threading support
- Long double support



Discussion

- What are the community's priorities?
  - If out of whack with ours, who wants to work on them?
- What's missing from the to-do list?
  - Integrating with weave and other scipy tools.
- Timing of switch to numarray
  - Criteria for switching?
    - Speed?
    - Compatibility?
    - Library support?
    - Documentation?