

PyData Array Recommendations

Thomas Dimitri

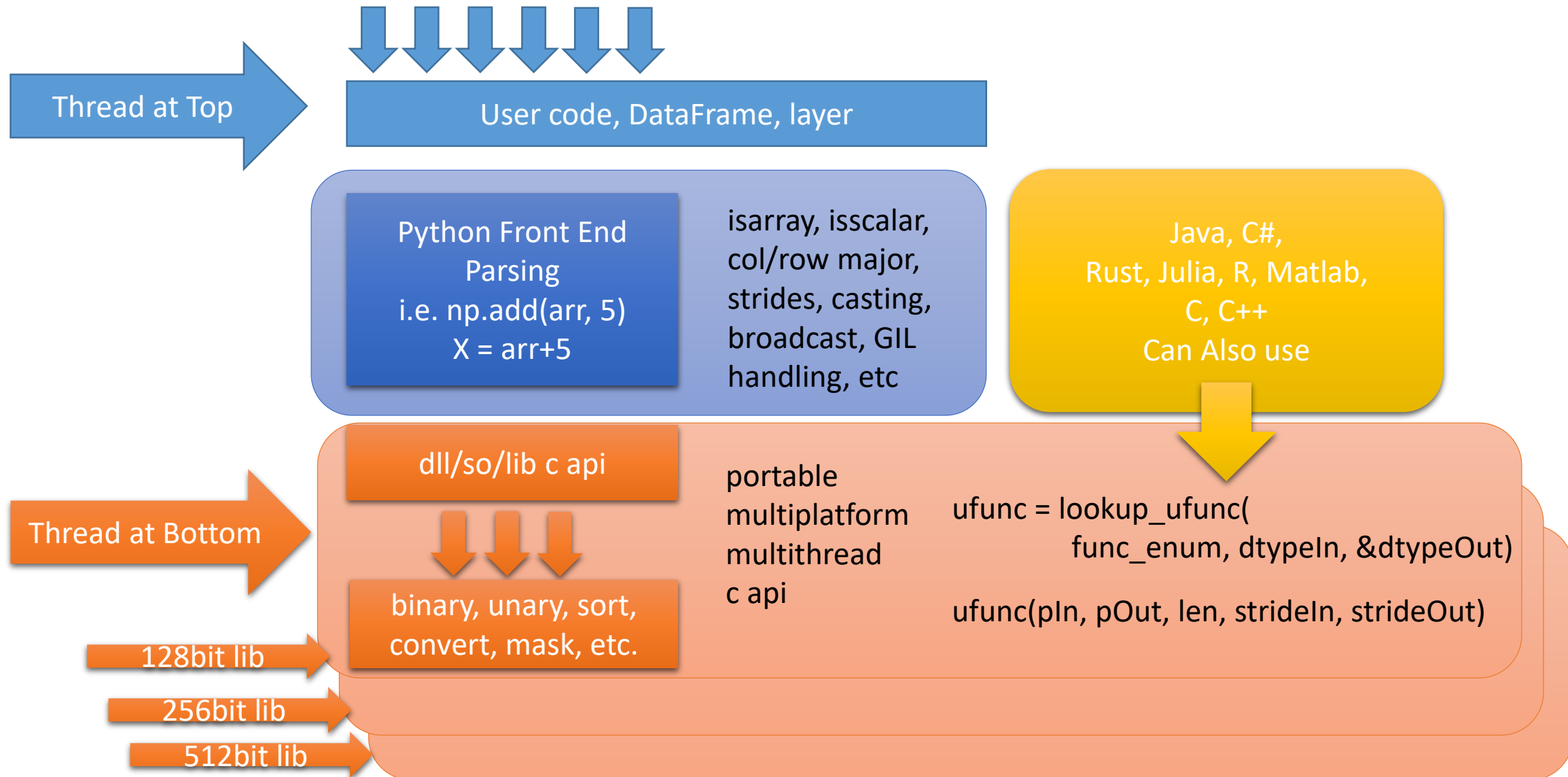
Background

- 30+ years professional software engineer – network protocols, user interfaces, compilers, file formats, pci cards, bios, trading engines, scanners, architectural software, ran 2 software companies, software patents, c/c++, java, c#, python
- 14 year consultant for SIG currently tasked with unifying large data analytics and framework
- SIG has Matlab, Python (numpy, pandas), C++/C# algos
- Desire to move towards one platform over time
- Ongoing beta solution: riptide: Uses Datasets/Structs/NumpyArrays, Multithreaded Numpy, portable FileFormat with stacking

Array Recommendations (in order of importance)

- Multithread back end engine and separate out to portable 'C' lib
- Introduce multikey Categorical class, subclassed from two classes Array and GroupingOps
- Introduce Grouping and GroupingOps class
- New loops: groupby loops/ partition loops/ closer JIT coupling
- Introduce new apis (hashing) + Invalids
- Subclass before making another dtype: Rework Date/Time classes
- Ledger + other ways to analyze performance
- Dtypes for variable length string: ucs1,2,4. Bitmask.
- Hooks for display/storage/setname/getname

Threading Model



Threading Model for Arrays

- Multicore/Shared memory same computer, same process
- Possible tuning: Thread count, NUMA, Process Affinity
- Wakeup (futex) – important: selective thread wake up
 - Some routines just need 2 worker threads, more threads is not going to speed anything up for absolute value. Memory bound problem.
- Calibration of worker threads
- Array Recycling (when large page allocations disabled - for speed to reduce page faults)

NOTE: Distributed/Cloud Computing (not covered)

Portable Multithreaded C (Orange Block)

- Gold Standard and easy to achieve with interface decoupling
- Allows other platforms or languages to call same routines with same results and performance
- Prefer C interface over C++ (a C++ layer can wrap C interface)
- In many low level routines want to get sum of Boolean mask (`np.add.reduce(mask)`) – should really just call `SumBoolean()`
- Threading while threading (have to know inside threaded routine)
- Be able to turn on “Ledger” to see what is getting called, performance

Array Routines We Threaded

- Basic Math both Unary and Binary (abs, add, sqrt, etc.)
- Comparisons (==, !=, <, >, <=, >=)
- Casting Conversions (i.e. int32 to float64)
- Boolean mask, Fancy Index mask set/get (i.e., `a = b[filter]`)
- Reduce: all reduce functions sped up
- Sorting (espec lexsort, multithreaded multikey indirect merge sort)
- putmask, where, searchsorted, hstack, interp
- + Added Hashing, ismember, group creation

Speed ups from multi-threading

- Varies from 1.5x to 30x speedups (large arrays)
- Some users thread at top (with multiple processes), and use 2 threads at bottom for maximum performance
- Most basic math sped up by 3x
- Some reduction functions like sum,min,max,std up to 30x
- Unary functions sped up (isnotnan 10x faster than ~np.isnan())
- Many comparisons sped up by 10x
- `getitem[fancyindex]` sped up by 20x
- Can always turn off – thus no harm from multithreading

Example dir of FastArray which subclasses np.ndarray (inherits)

FastArray

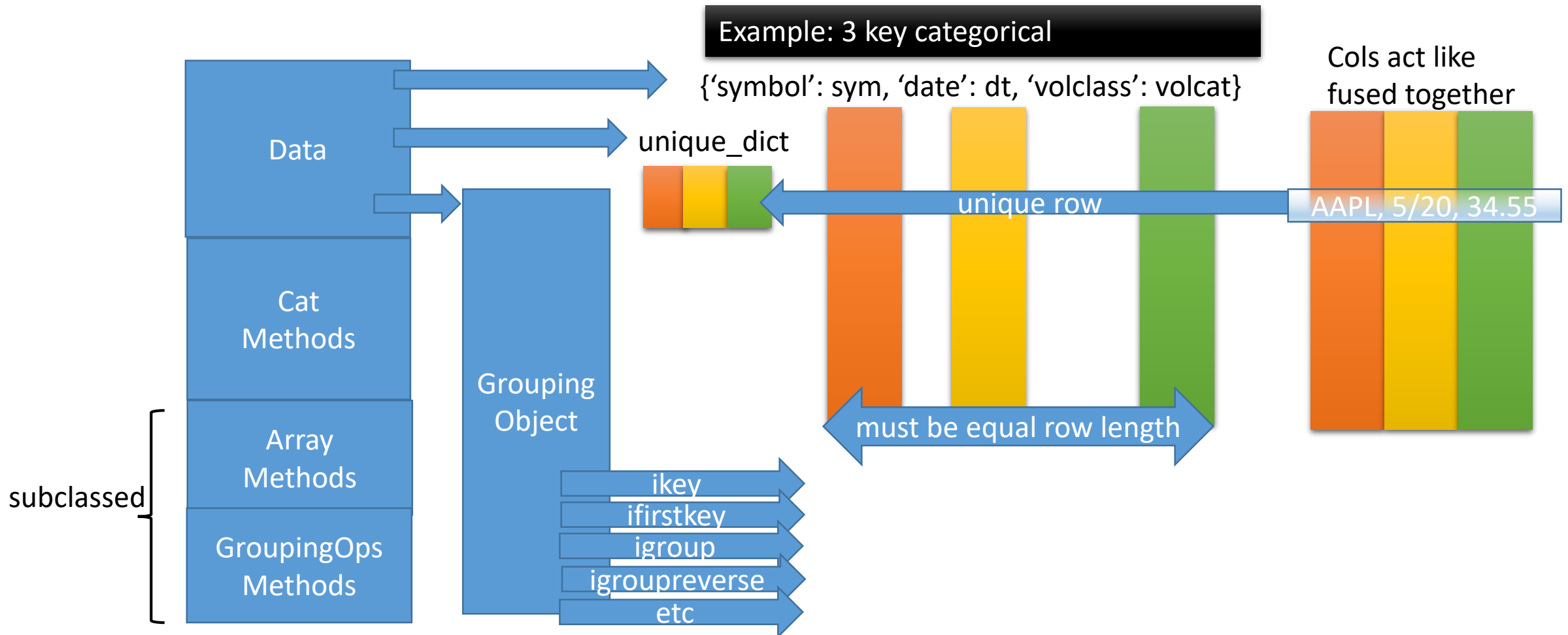
```
'abs', 'all', 'any', 'apply', 'apply_numba', 'argmax', 'argmin', 'argpartition', 'argpartition2', 'argsort', 'astype',  
'base', 'between', 'byteswap', 'choose', 'clip', 'clip_lower', 'clip_upper', 'compress', 'conj', 'conjugate', 'copy', 'copy_invalid', 'count',  
'crc', 'ctypes', 'cummax', 'cummin', 'cumprod', 'cumsum', 'data', 'describe', 'diagonal', 'diff', 'differs', 'display_query_property',  
'dot', 'dtype', 'dump', 'dumps', 'duplicated', 'ema_decay', 'fill', 'fill_backward', 'fill_forward', 'fill_invalid', 'fillna', 'flags', 'flat',  
'flatten', 'get_name', 'getfield', 'imag', 'info', 'inv', 'iscomputable', 'isfinite', 'isin', 'isinf', 'isna', 'isnan', 'isnanorzero',  
'isnormal', 'isnotfinite', 'isnotinf', 'isnotnan', 'isnotnormal', 'issorted', 'item', 'itemset', 'itemsizes', 'map', 'map_old', 'max', 'mean',  
'median', 'min', 'move_argmax', 'move_argmin', 'move_max', 'move_mean', 'move_median', 'move_min', 'move_rank', 'move_std', 'move_sum',  
'move_var', 'nanargmax', 'nanargmin', 'nanmax', 'nanmean', 'nanmin', 'nanrankdata', 'nanstd', 'nansum', 'nanvar', 'nbytes', 'ndim',  
'newbyteorder', 'nonzero', 'normalize_minmax', 'normalize_zscore', 'notna', 'numbastring', 'nunique', 'partition', 'partition2', 'prod', 'ptp',  
'push', 'put', 'rankdata', 'ravel', 'real', 'register_function', 'repeat', 'replace', 'replacena', 'reshape', 'resize', 'rolling_mean', 'rolling_nanmean',  
'rolling_nanstd', 'rolling_nansum', 'rolling_nanvar', 'rolling_std', 'rolling_sum',  
'rolling_var', 'round', 'sample', 'save', 'searchsorted', 'set_name', 'setfield', 'setflags', 'shape', 'shift', 'sign', 'size', 'sort',  
'squeeze', 'std', 'str', 'str_append', 'strides', 'sum', 'swapaxes', 'take', 'tile', 'timewindow_prod', 'timewindow_sum', 'tobytes', 'tofile',  
'tolist', 'tostring', 'trace', 'transitions', 'transpose', 'trunc', 'unique', 'var', 'view', 'where'
```

Note the added functionality for display (get_name, set_name, display_query)

Categorical Class (not a new dtype)

- Subclasses from (FastArray, GroupByOps)
 - Base array is int8/16/32/64
- Take most common categorical, of single string array
 - mycat=Cat(stringarray)
 - Often looks like string, sometimes integer bins
 - We chain `._fa` or `._np` to view FastArray or np.ndarray if we want to force integer bin mode
 - Can chain `.str` for string or `.date` for Dateclass to
- Creation can take two paths:
 - Via a lexsort or hashing
 - hashing (faster for low cardinality), also preserves order of first occurrence
- Once uniques are discovered, the high/low unique ratio count known important for some algorithms
- Have to be able to filter out (we reserve 0) so base 1 indexing

Categorical Class



Categorical has Grouping object also inherits from GroupingOps

- Grouping object:

'apply', 'apply_helper', 'as_filter', 'base_index', 'catinstance', 'copy', 'copy_from', 'count', 'gbkeys', 'get_name', 'ifirstgroup', 'ifirstkey', 'igroup', 'igroupreverse', 'ikey', 'ilastkey', 'inextkey', 'iprevkey', 'iscategorical', 'isdirty', 'isdisplaysorted', 'isenum', 'isin', 'ismember', 'ismultikey', 'isordered', 'isortrows', 'issinglekey', 'ncountgroup', 'ncountkey', 'newclassfrominstance', 'newgroupfrominstance', 'onedict', 'pack_by_group', 'packed', 'possibly_recast', 'register_functions', 'regroup', 'set_dirty', 'set_name', 'shrink', 'sort', 'unique_count', 'uniquedict', 'uniquelist'

dir of Categorical class (inherits from FastArray and GroupingOps)

'abs', 'agg', 'aggregate', 'align', 'all', 'any', 'apply', 'apply_nonreduce', 'apply_numba', 'apply_reduce', 'argmax', 'argmin', 'argpartition', 'argpartition2', 'argsort', 'as_filter', 'as_singlekey', 'as_string_array', 'astype', 'auto_add_off', 'auto_add_on', 'base', 'base_index', 'between', 'byteswap', 'categories', 'categories_equal', 'category_add', 'category_array', 'category_codes', 'category_dict', 'category_mapping', 'category_mode', 'category_remove', 'category_replace', 'choose', 'clip', 'clip_lower', 'clip_upper', 'compress', 'conj', 'conjugate', 'contains_np_arrays', 'copy', 'copy_invalid', 'count', 'count_uniques', 'crc', 'ctypes', 'cumcount', 'cummax', 'cummin', 'cumprod', 'cumsum', 'data', 'describe', 'diagonal', 'diff', 'differs', 'display_convert_func', 'display_query_properties', 'doc', 'dot', 'dtype', 'dump', 'dumps', 'duplicated', 'ema_decay', 'ema_normal', 'ema_weighted', 'expand_any', 'expand_array', 'expand_dict', 'fill', 'fill_backward', 'fill_forward', 'fill_invalid', 'fillna', 'filter', 'filtered_name', 'filtered_set_name', 'filtered_string', 'findnth', 'first', 'first_bool', 'first_fancy', 'flags', 'flat', 'flatten', 'from_bin', 'from_category', 'gb_keychain', 'get_groupings', 'get_header_names', 'get_name', 'getfield', 'groupby_data', 'groupby_data_clear', 'groupby_data_set', 'groupby_reset', 'grouping', 'grouping_dict', 'groups', 'head', 'hstack', 'ifirstkey', 'ikey', 'ilastkey', 'imag', 'info', 'inv', 'invalid_category', 'invalid_set', 'iscomputable', 'isenum', 'isfiltered', 'isfinite', 'isin', 'isinf', 'ismultikey', 'isna', 'isnan', 'isnanorzero', 'isnormal', 'isnotfinite', 'isnotin', 'isnotnan', 'isnotnormal', 'issinglekey', 'isorted', 'item', 'itemset', 'itemsizes', 'iter_groups', 'key_from_bin', 'last', 'lock', 'map', 'map_old', 'mapping_add', 'mapping_new', 'mapping_remove', 'mapping_replace', 'max', 'mean', 'median', 'min', 'mode', 'move_argmax', 'move_argmin', 'move_max', 'move_mean', 'move_median', 'move_min', 'move_rank', 'move_std', 'move_sum', 'move_var', 'nan_index', 'nanargmax', 'nanargmin', 'nanmax', 'nanmean', 'nanmedian', 'nanmin', 'nanrankdata', 'nanstd', 'nansum', 'nanvar', 'nb_ema', 'nb_min', 'nb_sum', 'nb_sum_punt', 'nbytes', 'ndim', 'newbyteorder', 'newclassfrominstance', 'ngroup', 'nonzero', 'normalize_minmax', 'normalize_zscore', 'notna', 'nth', 'null', 'numbastring', 'nunique', 'ohlc', 'one_hot_encode', 'ordered', 'partition', 'partition2', 'prod', 'ptp', 'push', 'put', 'rank', 'rankdata', 'ravel', 'real', 'register_function', 'register_functions', 'repeat', 'replace', 'replacena', 'resample', 'reshape', 'resize', 'rolling_count', 'rolling_diff', 'rolling_mean', 'rolling_nanmean', 'rolling_nanstd', 'rolling_nansum', 'rolling_nanvar', 'rolling_shift', 'rolling_std', 'rolling_sum', 'rolling_var', 'round', 'sample', 'save', 'searchsorted', 'sem', 'set_name', 'setfield', 'setflags', 'shape', 'shift', 'shift_cat', 'shrink', 'sign', 'size', 'sort', 'sort_gb', 'sorted', 'squeeze', 'std', 'str', 'str_append', 'strides', 'sum', 'swapaxes', 'tail', 'take', 'tile', 'timewindow_prod', 'timewindow_sum', 'tobytes', 'tofile', 'tolist', 'tostring', 'trace', 'transform', 'transitions', 'transpose', 'trimbr', 'trunc', 'unique', 'unique_count', 'unique_repr', 'unlock', 'var', 'view', 'where'

Cat example with filter and GroupingOps

example str Cat

isin filter

can trim

very fast str ops

has groupbyops

```
In [41]: symbol
Categorical([GOLD, GOLD, GOLD, GOLD, GOLD, ..., QURE, QURE, QURE, QURE, QURE]) Length: 82212305
FastArray([1249, 1249, 1249, 1249, 1249, ..., 2336, 2336, 2336, 2336, 2336], dtype=int16) Base Index: 1
FastArray([b'A', b'AA', b'AAL', b'AAN', b'AAOI', ..., b'ZUO', b'ZVO', b'ZYME', b'ZYNE', b'ZYXI'], dtype='|S16') Unique count: 3201

In [42]: symbol[['GOLD', 'AAPL']]
FastArray([ True,  True,  True, ..., False, False, False])

In [43]: symbol.shrink(['AAPL', 'IBM', 'QURE'])
Categorical([Filtered, Filtered, Filtered, Filtered, Filtered, ..., QURE, QURE, QURE, QURE, QURE]) Length: 82212305
FastArray([0, 0, 0, 0, 0, ..., 3, 3, 3, 3, 3]) Base Index: 1
FastArray([b'AAPL', b'IBM', b'QURE'], dtype='|S4') Unique count: 3

In [44]: symbol.str.lower
FastArray([b'gold', b'gold', b'gold', ..., b'qure', b'qure', b'qure'],
dtype='|S16')

In [45]: symbol.sum(arange(len(symbol)))
#b7o_Header_Symbol      col_0
-----
A          966489024152
AA         2063505633016
AAL        26129178449080
AAN         316412972621
AAOI        252219031198
AAN         16269136409
AAP         866498166991
AAPL        287286361753363
AAT         137698846
```

Cat example with base array, grouping, reduction

can flip view

get uniques

Grouping object

igroup is fancy
index

has apply power

```
In [53]: symbol._np
array([1249, 1249, 1249, ..., 2336, 2336, 2336], dtype=int16)

In [54]: symbol.categories()
FastArray([b'Filtered', b'A', b'AA', ..., b'ZYME', b'ZYNE', b'ZYXI'],
          dtype='<S16')

In [55]: symbol.grouping
_ikKey: [1249 1249 1249 ... 2336 2336 2336]
_iFirstKey: None
_iLastKey: None
_iNextKey: None
_unique_count: 3201
_grouping_dict: None
_grouping_unique_dict: {'symbol': FastArray([b'A', b'AA', b'AAL', ..., b'ZYME', b'ZYNE', b'ZYXI'],
          dtype='<S16'))}
_enum: None
_categorical: False
_isenum: False
_isdirty: False
_catinstance: None

_packed: True
_iGroup: [34051548 34051549 34051550 ... 40172985 40172986 40172987]
_iFirstGroup: [ 0 0 17272 ... 82202231 82208329 82210691]
_nCountGroup: [ 0 17272 38129 ... 6098 2362 1614]

_gbkeys: None
_sort_display: False
_Ordered: True
_base_index: 1

In [56]: symbol.apply_reduce(lambda x: x.mean(), {'arange': arange(len(symbol)), 'ones': ones(len(symbol))})
*b7o_Header_Symbol      arange      ones
-----
A      5.596e+07      1.00
AA     5.412e+07      1.00
AAL    5.046e+07      1.00
AAN    4.919e+07      1.00
AAOI   4.725e+07      1.00
AAON   5.959e+07      1.00
AAP    5.322e+07      1.00
AAPL   4.676e+07      1.00
```

igroup + ncountgroup
needed for grouping loop

Enum Cat Example (pre defined bins)

User has predefined bins

Wants to see 33 for 'stop'

Duality: can use strings

Duality: can use int

Has same functionality as
normal categorical

user bin referenced, not copied

subclasses from np.ndarray

isin and other functionality works

```
In [6]: mycat = Cat([44,33,44,55, 55], {'stop':33, 'start':44, 'pause':55})

In [7]: mycat
Categorical([start, stop, start, pause, pause]) Length: 5
FastArray([44, 33, 44, 55, 55]) Base Index: None
{33:'stop', 44:'start', 55:'pause'} Unique count: 3

In [8]: mycat=='stop'
FastArray([False,  True, False, False, False])

In [9]: mycat==33
FastArray([False,  True, False, False, False])

In [10]: mycat.count()
#key_0  Count
-----  ----
start      2
stop       1
pause      2

[3 rows x 2 columns] total bytes: 72.0 B

In [11]: mycat._np
array([44, 33, 44, 55, 55])

In [12]: mycat.dtype
dtype('int32')

In [13]: isinstance(mycat, np.ndarray)
True

In [14]: mycat[['start','stop']]
FastArray([ True,  True,  True, False, False])
```

Multikey Cat Example (Date + String)

Can have many keys, ordered or not

```
In [36]: x=utcnow()
In [37]: x
DateTimeNano(['20200520 11:39:00.087683900', '20200520 11:39:00.087684200', '20200520 11:39:00.087684200', '20200520 11:39:00.087684600', '20200520 11:39:00.087684600'], to_tz='NYC')
In [38]: y=FA(['sam','sam','sam','jack','jack'])
mycat = Cat([x,y])
In [40]: mycat
Categorical([('20200520 11:39:00.087683900', sam), (20200520 11:39:00.087684200, sam), (20200520 11:39:00.087684200, sam), (20200520 11:39:00.087684600, jack), (20200520 11:39:00.087684600, jack)]) Length: 5
  *key_0: DateTimeNano(['20200520 11:39:00.087683900', '20200520 11:39:00.087684200', '20200520 11:39:00.087684600'], to_tz='NYC'), 'key_1': FastArray([b'sam', b'sam', b'jack'], dtype='|S4']) Unique count
In [41]: mycat.count()
-----
*key_0  *key_1  Count
-----
20200520 11:39:00.087683900  sam      1
20200520 11:39:00.087684200  sam      2
20200520 11:39:00.087684600  jack      2
[3 rows x 3 columns] total bytes: 48.0 B
In [42]: mycat == (x[1], y[1])
FastArray([False, True, True, False, False])
In [61]: x.set_name('x')
DateTimeNano(['20200520 11:39:00.087683900', '20200520 11:39:00.087684200', '20200520 11:39:00.087684200', '20200520 11:39:00.087684600', '20200520 11:39:00.087684600'], to_tz='NYC')
In [62]: y.set_name('y')
FastArray([b'sam', b'sam', b'sam', b'jack', b'jack'], dtype='|S4')
In [63]: z=Cat([x,y])
z.count()
-----
*x      *y      Count
-----
20200520 11:39:00.087683900  sam      1
20200520 11:39:00.087684200  sam      2
20200520 11:39:00.087684600  jack      2
[3 rows x 3 columns] total bytes: 48.0 B
In [64]: z.min({'First':arange(5), 'Second': arange(5.0)+5})
-----
*x      *y      First  Second
-----
20200520 11:39:00.087683900  sam      0      5.00
20200520 11:39:00.087684200  sam      1      6.00
20200520 11:39:00.087684600  jack      3      8.00
[3 rows x 4 columns] total bytes: 72.0 B
In [65]: z._np
FastArray([1, 2, 2, 3, 3], dtype=int8)
In [66]: z._np
FastArray([1, 2, 2, 3, 3], dtype=int8)
```

ez to create multikey

still uses int bin

use tuple to lookup

arrays needs names

now names correct

groupops functionality

int8/16/32/64
based on uniq count

Common Categorical Operations

- Categoricals can
 - Be compared to one another
 - Add, Remove, Edit, Reorder, Regroup uniques
 - Filtered (reserve 0, base 1 indexing)
 - Saved and Loaded
 - Convert from multikey to single key
 - Revert back to original array construction
 - Multiple categoricals can be Stacked
 - Be created from one or more other categoricals
 - Can be ordered in various ways: lexicographical/first appearance

Grouping and GroupingOps class

- Contains common routines that users like to perform over groups
 - Reduce functions such as sum, min, max
 - Cumulative such as cumsum, cumprod
 - Rolling or sliding windows, moving averages (ema functions)
 - Repopulating original array after a reduce function (e.g. 'transform')
 - Any operation performed over a single array, can be performed over a group of single arrays
 - Custom functions or lambda functions which can be jitted
 - Allowing multiple functions (from above) to be applied in one command
- Contains helper arrays:
 - Have fancy index helpers:
 - ifirstkey – an array of row numbers where first member of group appeared
 - ilastkey – where last member of group appeared
 - igroup – a fancy index which when applied, would reorder groups contiguously
 - igroupreverse – fancy index to get reverse igroup
 - Other arrays
 - ngrouppcount (how many items in the group)
 - more

New Loops

- Existing:
 - Contig math vector loops (stride == itemsize)
 - Able to hit mm_256_add_ routines, etc.
 - Strided loops
 - Can also use vector routines via gather intrinsic
- New: Groupby Loops
 - See “igroup” – fancy index in order of uniques
 - For example if ‘AAPL’ is first category then it might exist in 10,000 rows scattered about 1 million row array. Similar to what lexsort returns now to sum up all ‘AAPL’ can just use the index to get to the data (no need to copy or reshuffle data)
- New: Partition Loops
 - Often used after sort_inplace. Now groups are in order – might be many groups, perhaps 1 million... want to ‘fake’ slice arrays fast to do operations (real slice too much overhead)
- Loop over JIT kernels
 - Numba has nb.prange and threads.. Can use that to create own grouping kernel

Suggested Additions to Array

- Hashing Class
 - Example “groupbyhash” and ismember
 - Many hashing algos – we use linear probing hash
 - Also quadratic hash, robin hood, cuckoo, etc, + red/black tree
 - Or crc, “smarhash”, “murmur”, and many others
 - Like np.add.reduce... np.hash.reduce
 - What is fast way to drop_duplicates? hash
 - Find out if all the unique categories are the same? hash
- Searchsorted with kwarg ‘exact’
 - Must be exact match or give other index, like -1
 - Searchsorted is easily multithreadable, thus it can be used in ismember
- Merge variations
 - Enrichment style
 - Left/Right Inner/Outer Join
- Think about isnot routines like isnotnan very popular
- Other ideas like fma (fused multiply add)

Hashing speed plus invalids and fancy indexing

Very common big array, 99 uniques

compare ismember to np.isin

ismember also returns fancy index

20x faster with more information

fancy idx understand invalids

Dataset/Frame understands also

astype understands invalids

25x faster with invalid capability

```
In [119]: z.astype(np.float64)
FastArray([nan, nan, 40., ..., nan, nan, 39.])

In [120]: npa = y._np

In [121]: y
FastArray([28, 40, 29, 39])

In [122]: fi = arange(1_000_000) % 4

In [123]: npa[fi]
array([28, 40, 29, ..., 40, 29, 39])

In [124]: y[fi]
FastArray([28, 40, 29, ..., 40, 29, 39])

In [125]: %timeit npa[fi]
3.78 ms ± 76.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

In [126]: %timeit y[fi]
141 µs ± 1.34 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
In [90]: x = np.random.randint(1,100, 10_000_000)

In [91]: x
array([27, 84, 40, ..., 96, 72, 39])

In [92]: y=FA([28,40,29,39])

In [93]: np.isin(x,y)
array([False, False,  True, ..., False, False,  True])

In [94]: rc.IsMember32(x,y)
(FastArray([False, False,  True, ..., False, False,  True]),
 FastArray([-128, -128,   1, ..., -128, -128,   3], dtype=int8))

In [95]: %timeit np.isin(x,y)
42.1 ms ± 2.44 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [96]: %timeit rc.IsMember32(x,y)
1.86 ms ± 114 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

In [97]: fi = rc.IsMember32(x,y)[1]

In [98]: fi
FastArray([-128, -128,   1, ..., -128, -128,   3], dtype=int8)

In [99]: y[fi]
FastArray([-2147483648, -2147483648,   40, ..., -2147483648,
-2147483648,   39])

In [100]: Dataset({'test': y[fi]})
# test
-----
0      Inv
1      Inv
2      40
3      Inv
4      Inv
5      Inv
6      Inv
7      Inv
8      Inv
9      Inv
10     Inv
11     Inv
12     Inv
13     Inv
14     Inv
...
9999985  Inv
9999986  Inv
9999987  Inv
9999988  Inv
9999989  Inv
9999990  Inv
9999991  Inv
9999992  Inv
9999993  Inv
9999994  Inv
9999995  Inv
9999996  Inv
9999997  Inv
9999998  Inv
9999999  39

[10000000 rows x 1 columns] total bytes: 38.1 MB

In [101]: z=y[fi]

In [102]: z.inv
-2147483648
```

Subclass before creating a new dtype

- Example: Datetime or ipv4 class
 - We have DateTimeNano, TimeSpan, Date, DateSpan classes
 - Date class is int32, days since epoch and is timezone agnostic
 - Subtracting two Dates produces a DateSpan
 - Also based on int32
 - Subtracting two Times produces a TimeSpan
 - Can be EITHER int64 or float64 based
 - Each has pros and cons
 - Is that 5 new dtypes? Now a TimeSeries class.. 6 new dtypes? Ipv4 – 7 new dtypes? When does it end?
 - Rather just subclass
 - `class Date(FastArray, otherclass):`
 - `def __new__(cls, arr, ..):`
 - `return arr.view(Date)`
 - Use `isinstance()` to detect, may need centralized Class registry (we have a TypeRegister)
 - No new ufunc loops required:
 - If I want to add 3 to DateTimeNano to add 3 nanoseconds, just +3
 - Hits sped up vectorized addition code – no new work
 - No `switch(dtype)`:
 - case int64:
 - case datetime64: ← no need to add this new dtype everywhere anymore

Ledger

- Records all operations, can time them
- Can see when upcasting occurs, which routines take longest, order of operations, how many threads used
- Built into portable C code (so all languages can use)
- Useful for debugging performance (espec due to C code hiding from most python profilers)
- Similar info with Recycler

New dtypes

- Variable length strings
- Bitmask Boolean
- Otherwise subclass from common dtypes
 - Use `isinstance()` checks
 - May want central “TypeRegister” to hold all known types
 - Class has to know how to save itself and reload itself so can be serialized to any file format
 - Class has to know how to display itself, right/left justification or long short display format, strftime format, precision, ipv4 etc.

Display/Serialization Hooks

- Known `__` attributes to get/set meta data for serialization
- Hooks to help with autocomplete
- Or other display – left justified, abbreviated, etc.

```
[11]: d = Struct({"alpha": 1, "beta": [2, 3], "gamma": ['2', '3'], "delta": arange(10),  
               "epsilon": Struct(  
                 "theta": Struct(  
                   "kappa": 3,  
                   "zeta": 4,  
                 }),  
               "iota": 2 })
```

```
[12]: d["AAAA"]="AAAA"
```

```
[13]: d.|
```

s	AAAA	str
i	alpha	int
l	beta	list
a	delta	array i32
s	epsilon	struct(0, 2)
l	gamma	list
f	all	function
i	AllNames	instance
i	AllowAnyName	instance
f	any	function

```
[9]:  
[10]:  
ons, *args, _debug=False):
```