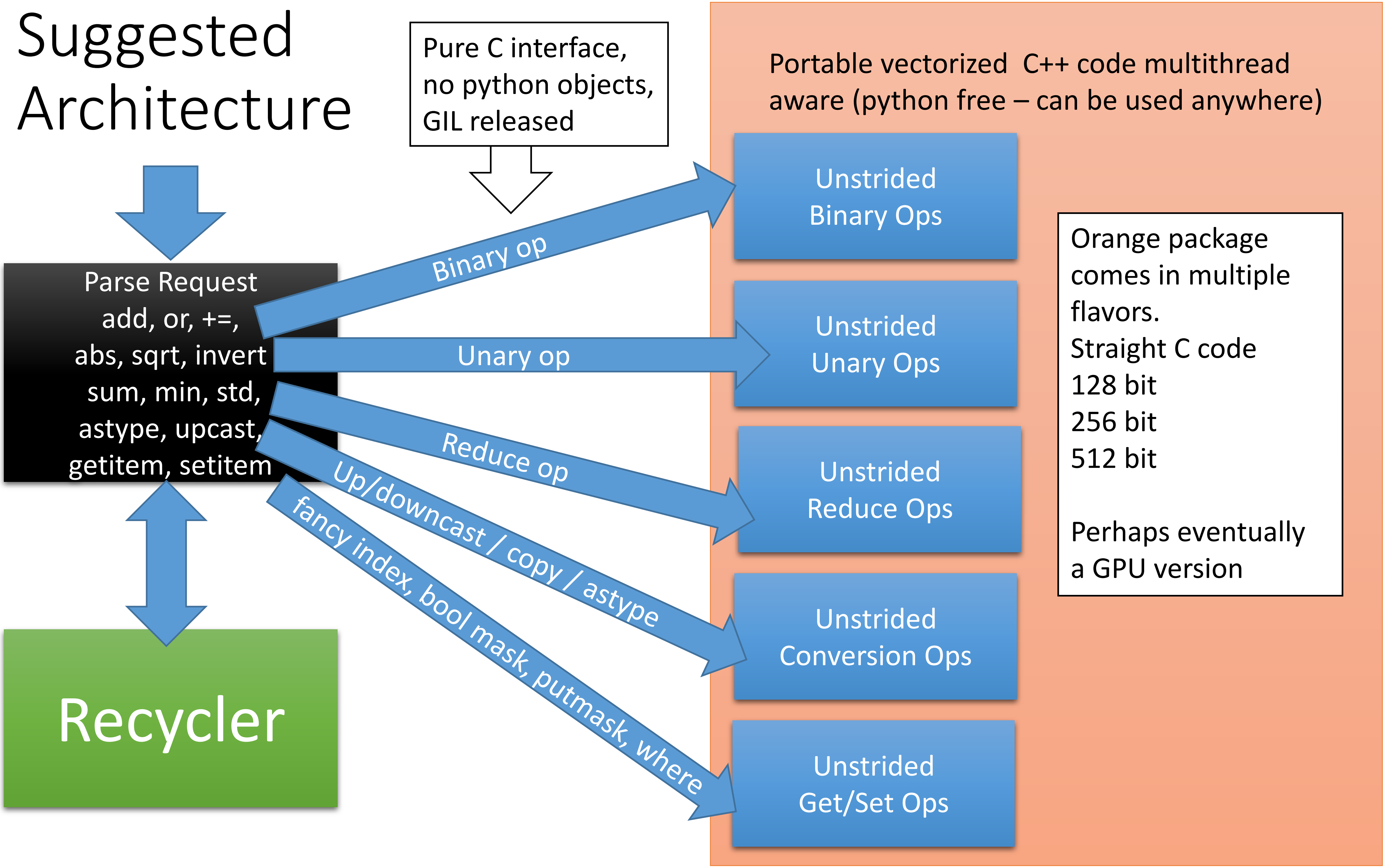
Suggested Architecture



Pure C interface, no python objects, GIL released

Parse Request
add, or, +=,
abs, sqrt, invert
sum, min, std,
astype, upcast,
getitem, setitem

Binary op

Unary op

Reduce op

Up/downcast / copy / astype

fancy index, bool mask, putmask, where

Recycler

Portable vectorized C++ code multithread aware (python free – can be used anywhere)

Unstrided Binary Ops

Unstrided Unary Ops

Unstrided Reduce Ops

Unstrided Conversion Ops

Unstrided Get/Set Ops

Orange package comes in multiple flavors.
Straight C code
128 bit
256 bit
512 bit

Perhaps eventually a GPU version

Suggested approach

- Recycling – off by default if large memory supported, else default on, universal to all platforms, can tune (see write up)
- Threading – off by default, universal to all platforms, can turn on, assign numa affinity, assign max count, etc.
- Vectorization – on by default for computers that support it

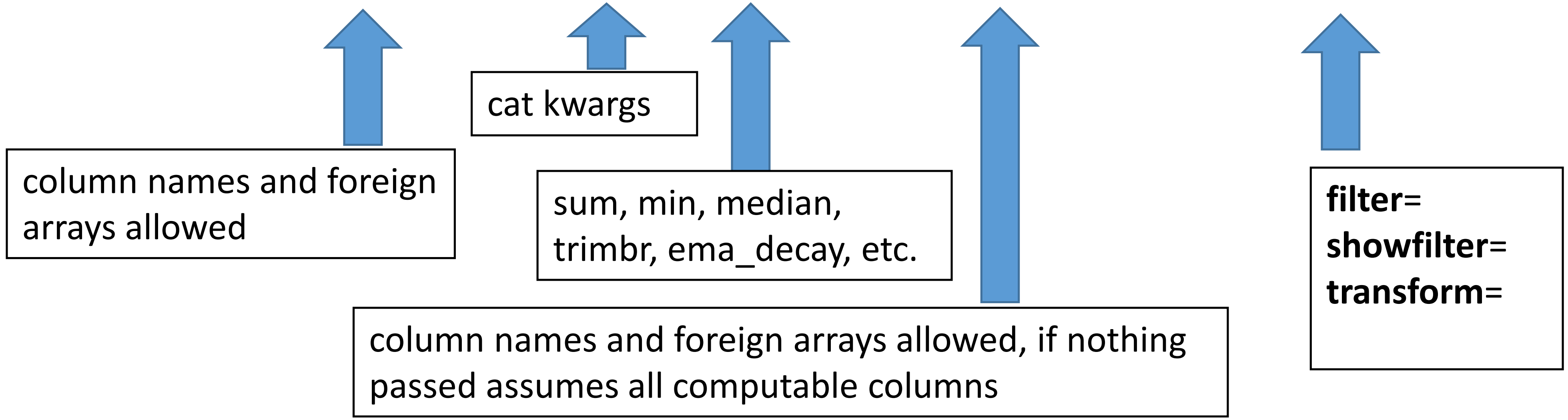
Vectorization is chip dependent

- 128 / 256/ 512 flavors
  - Start with 256. If computer only supports 128 – do they really need high performance since that chip is 8+ years old? Not worth effort initially.
- Requires plug in package so others can contribute – for instance, once a user sees how 256 bit package is written, the 512 package is derivable (and so is 128 or some future chip).
- Newer chips have more features that can be exploited (and thus the pluggable package approach).

First step

- Write the recycler first otherwise – you will not be able to tell when CPU is hitting max speeds due to page faults.
- Write threading second

ds.groupby(<ONE OR MORE KEYS>, kwargs).OPERATION(<ONE OR MORE INPUTS>, kwargs)



column names and foreign arrays allowed

cat kwargs

sum, min, median, trimbr, ema_decay, etc.

column names and foreign arrays allowed, if nothing passed assumes all computable columns

filter=
showfilter=
transform=

ds.groupby(<ONE OR MORE KEYS>).agg(<ONE OR MORE OPS>, <ONE OR MORE INPUTS>, kwargs)

ds.groupby(<ONE OR MORE KEYS>).apply_reduce(<ONE OR MORE OPS>, <ONE OR MORE INPUTS>, kwargs)

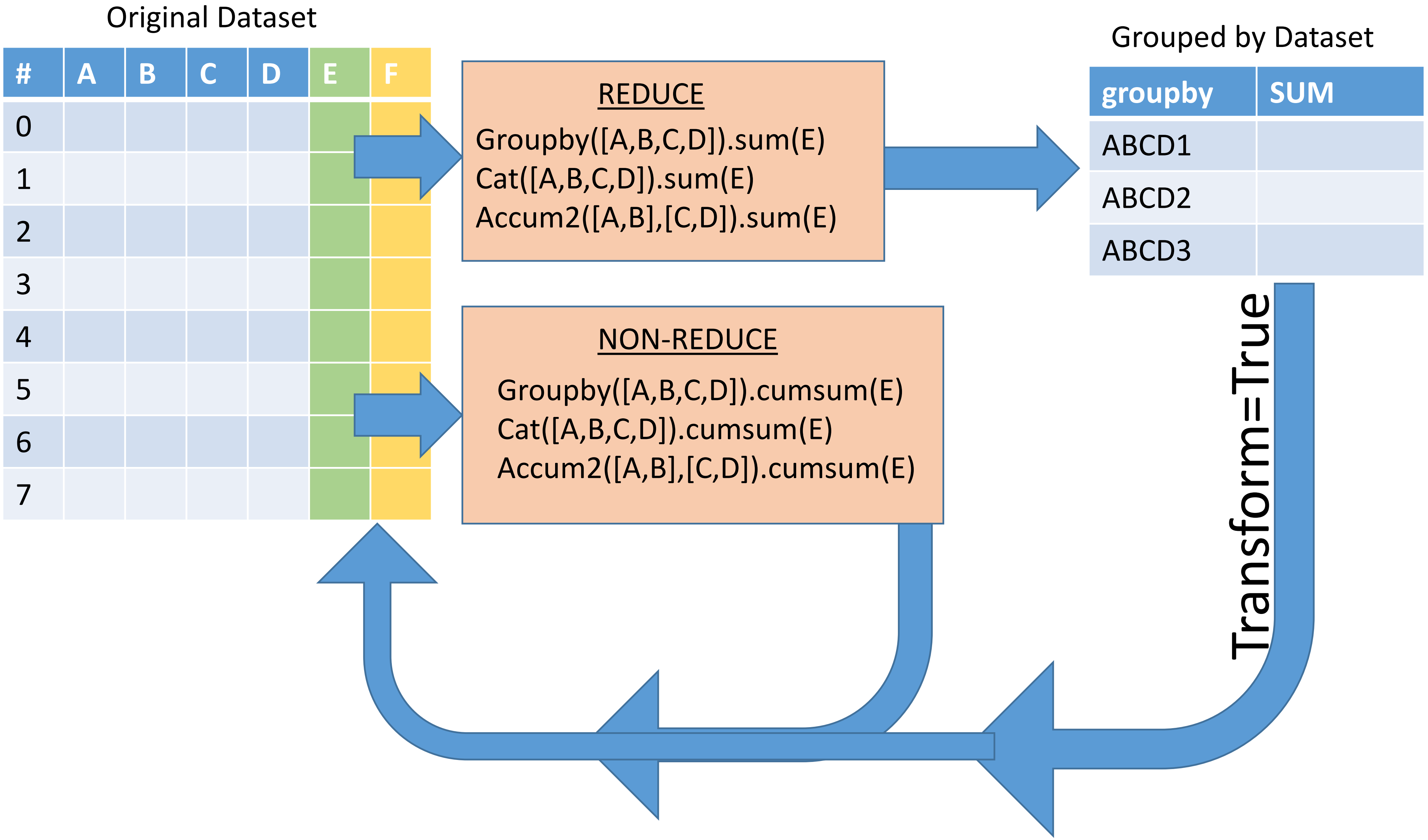
Groupby shorthand methods

ds.gb: does pandas style – first occurrence binning, but display sorted
ds.gbu: ‘u’ = unordered. First occurrence ordering and binning. Bins displayed in first occurrence.
ds.gbs: ‘s’ = sorted. All keys sorted (lexsort). Bins are sorted and displayed that way. (lex=True)

Categorical Form then matches identically:

cat.OPERATION(<ONE OR MORE INPUTS>, kwargs)
cat.agg(<ONE OR MORE OPS>, <ONE OR MORE INPUTS>, kwargs)
cat.apply_reduce(<ONE OR MORE OPS>, <ONE OR MORE INPUTS>, kwargs)

Multiple ways to produce a new column ‘F’ which can be fed back in



Original Dataset

#	A	B	C	D	E	F
0						
1						
2						
3						
4						
5						
6						
7						

REDUCE

Groupby([A,B,C,D]).sum(E)
Cat([A,B,C,D]).sum(E)
Accum2([A,B],[C,D]).sum(E)

Grouped by Dataset

groupby	SUM
ABCD1	
ABCD2	
ABCD3	

NON-REDUCE

Groupby([A,B,C,D]).cumsum(E)
Cat([A,B,C,D]).cumsum(E)
Accum2([A,B],[C,D]).cumsum(E)