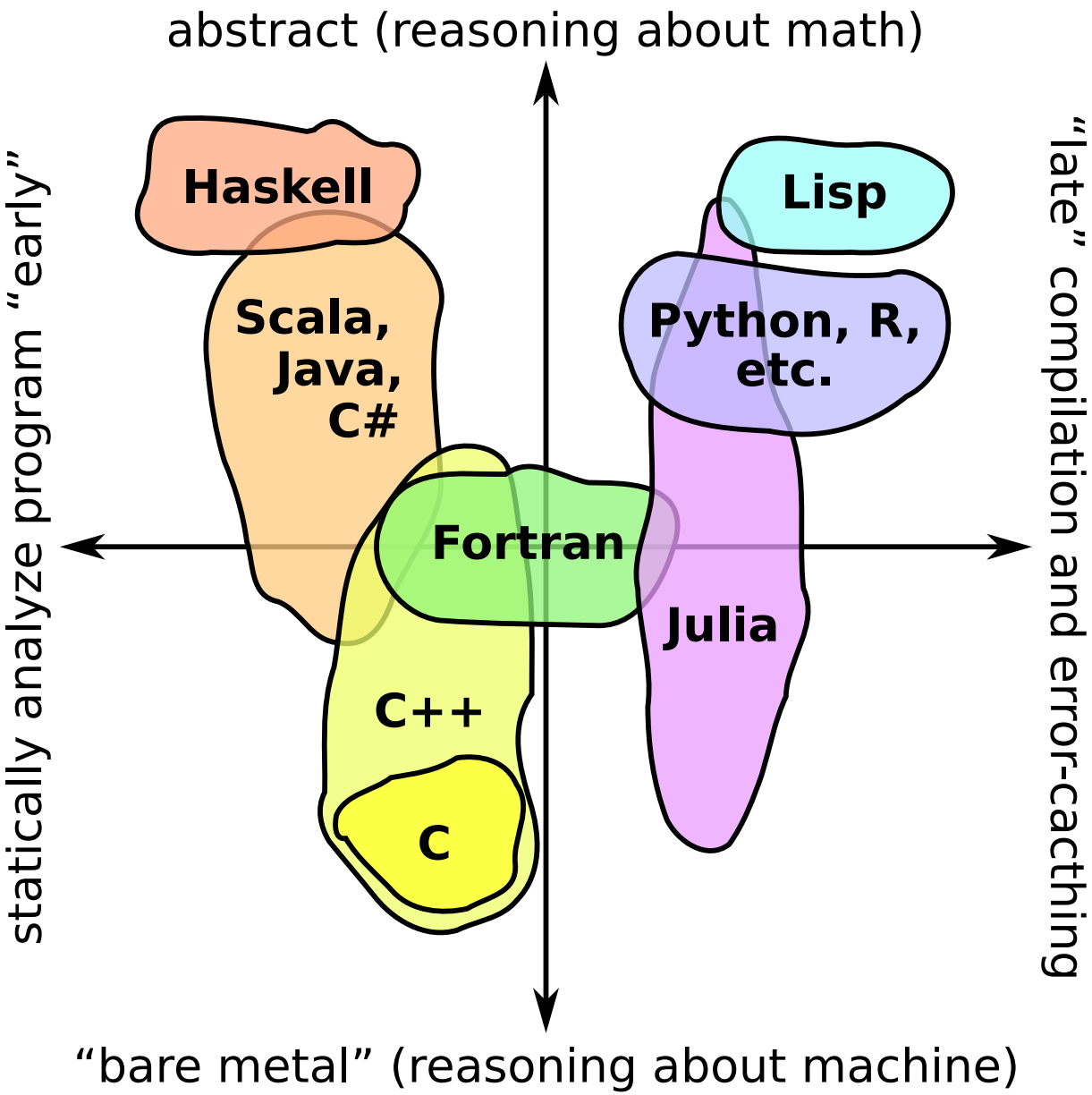
Julia: a strange place in language space



Julia is almost everything I wanted

My wish-list from last time:

- define a subset of a well-known syntax, such as Python
- that JIT-compiles to CPU , GPU, and maybe FPGA
- that can be freely mixed with surrounding code
- that is immutable, maybe total-functional
- with special handling of Python's imperative statements and restrictive lambdas
- using interval arithmetic as data types
- separate mathematics from performance annotations in CSS

Julia is almost everything I wanted

diana hep

My wish-list from last time:

- define a subset of a ~~well~~-known syntax, such as ~~Python~~ Julia
- that JIT-compiles to CPU, GPU, and maybe FPGA
- that can be freely mixed with surrounding code √
- that is immutable, maybe total-functional: check for Julia functions ending in "!"
- ~~with special handling of Python's imperative statements and restrictive lambdas~~: Julia syntax is expression-only and easily defines complex lambdas
- using interval arithmetic as data types: Julia's metaprogramming supports this kind of inspection
- separate mathematics from performance annotations in CSS Julia doesn't do this, but its metaprogramming would help build a demonstration project

However. . .

Potential stumbling blocks for HEP analysts

diana hep

Julia behavior	**HEP expectation**
arrays range from 1 to N	0 to $N-1$ from C arrays, C++ vectors, and Python lists
histogram (built-in `hist()`) is exclusive on low edge and inclusive on high edge	inclusive on low edge and exclusive on high edge (HBOOK, PAW, ROOT, Numpy)
multidimensional arrays are column-major	row-major C/C++, Numpy default, GPU
no OOP-like fluent syntax: `event.track(2).hit(12)` → `hit(track(event, 2), 12)`	physicists have become accustomed to thinking in terms of objects
`elseif` (no space)	C/C++ `else if`; Python `elif`

Potential issues for building large projects

- Since everything is JIT-compiled, some errors are discovered only when they're encountered, such as `UndefVarError`.

  Unit tests need to fill in for missing "compiler checks," and it's hard to guarantee coverage of all potential uses (just like Python programming or C++ templates).
- `Function` is not a parameterized type, so functions passed as arguments can't be constrained by method signature.
- Without methods in data structures (classes), the namespace fills up: I might think I'm defining a new function for my custom data structure, but I'm actually adding methods to a global function of the same name.
- Would need formal interfaces (contracts) to get an error if an implementation is incomplete.
- Do all packages need to be GitHub repositories ending in `.jl`?