NOAA Modeling Fair
Python Session

– September 11, 2018 –

Sam Trahan
Todd Spindler
Hyun-Sook Kim
Deanna Spindler

NOAA NCEP EMC / IM Systems Group

Session Overview

- Talk 1: *Python: What it is and What it is Not.*
  - Sam Trahan
- Talk 2: *Reading Scientific Datasets in Python*
  - Todd Spindler
- Talk 3: *Graphical Diagnostic Tools*
  - Hyun-Sook Kim
- Talk 4: *Time Series Data Analysis in Python*
  - Deanna Spindler

What is Python and What is it *Not*?

- From python.org website. This is half true:

  Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together... more text …

- This is half true:
  - Python is used for numerical computing, machine learning, data visualization, and much more. It is an ecosystem.

What is Python and What is it *Not*?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together... more text …

Python is used for numerical computing, machine learning, data visualization, and much more. It is an ecosystem.

What is Python and What is it *Not*?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together... more text …

Python is used for numerical computing, machine learning, data visualization, and many more. It is an ecosystem.

“Python is”
“Language *XYZ* is”

- Formal, international, standards:
  - C++ – ISO/IEC 14882:2014
  - Fortran – ISO/IEC 1539-1:2010
  - C – ISO/IEC 9899:2018
- Proprietary standards:
  - Bash – GNU Project
  - Visual Basic – Microsoft
  - Matlab – MathWorks

~~“Python is”~~
Pythons are

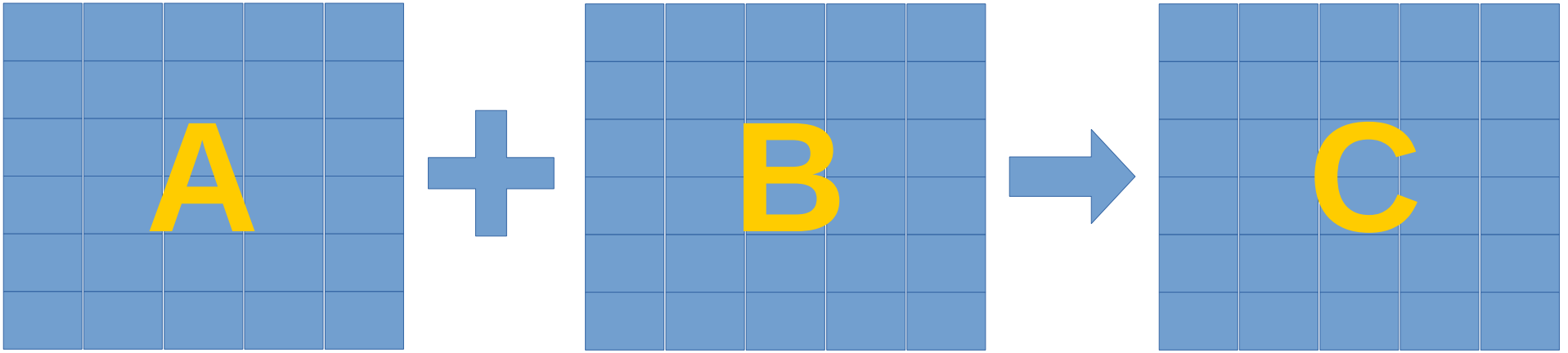
- A list of recommendations (PEP) with a reference implementation (CPython) and no standard.
- Pythons are.
  - CPython – reference implementation from python.org
  - PyPy – Just In Time (JIT) compiler; usually faster than CPython
  - Jython – compiles to Java bytecode; usually faster than CPython
  - Cython – compiles Python to C; compatible with CPython
  - Numba – JIT compiler using LLVM, sits within CPython
  - IronPython – integrates Python into Visual Studio
- Similar language, different set of supported libraries.

What is Python and What is it *Not*?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together... more text …

Python is used for numerical computing, machine learning, data visualization, and many more. It is an ecosystem.

Languages: High vs Low
Low-Level Languages



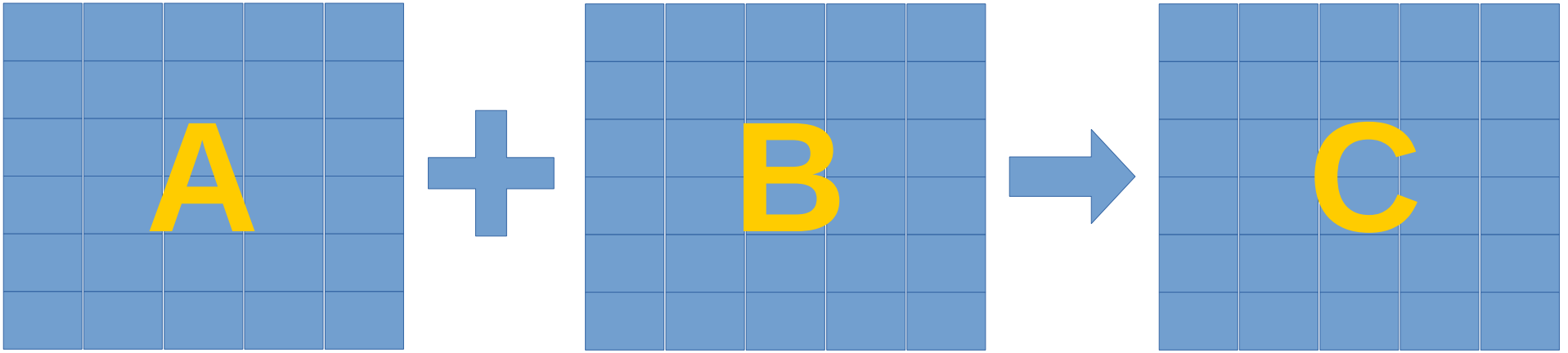
- Low-level Fortran

```
do j=1,m
  do i=1,n
    c(i,j) = a(i,j) + b(i,j)
  enddo
enddo
```

- Low-level C

```
for(j=0;j<m;j++)
  for(i=0;i<n;i++)
    c[j][i]=a[j][i]+b[j][i]
```

Languages: High vs Low
High-Level Languages



- High-level Fortran

  C = A + B

- High-level C++

  C = A + B;

- Python with numpy

  C = A + B

- High-level R

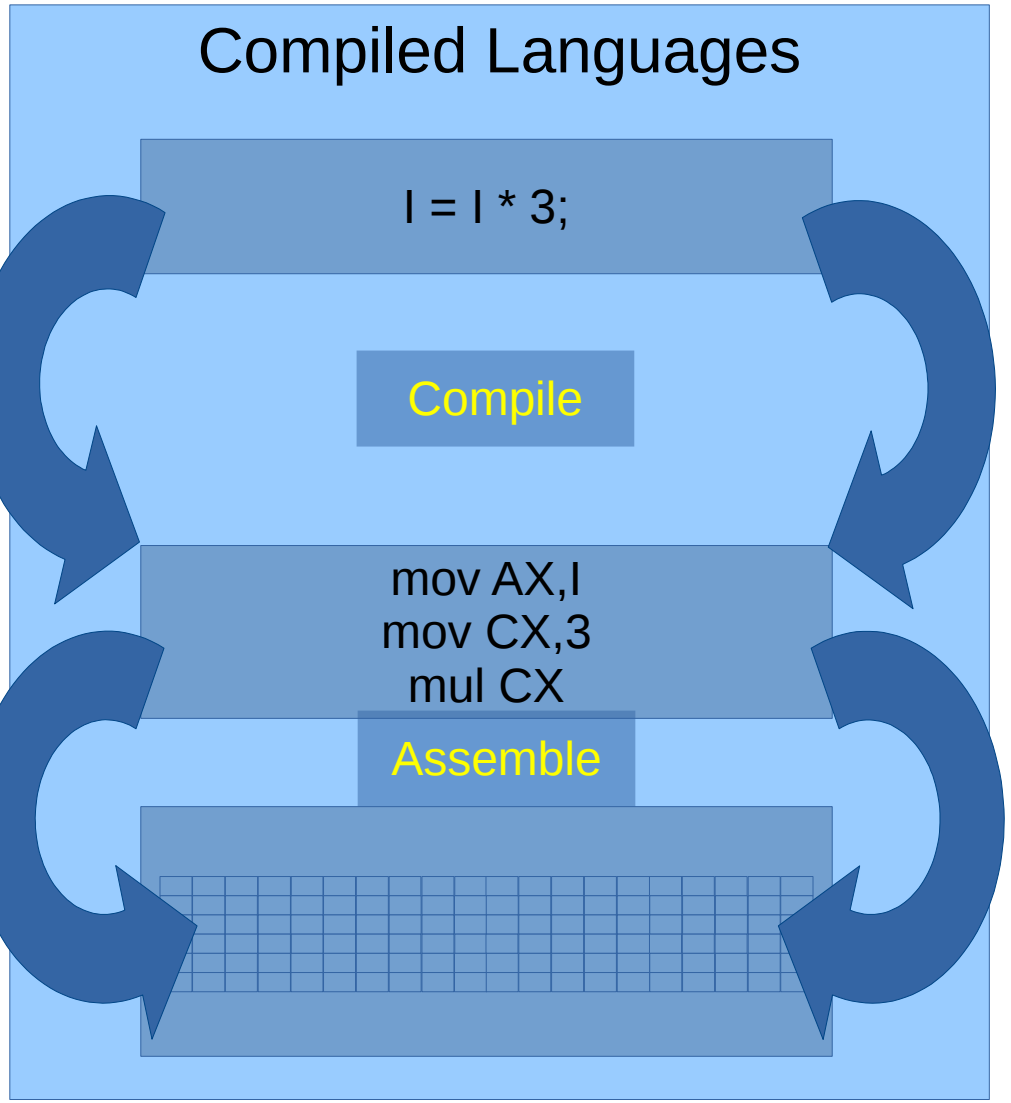
  C <- A + B

What is Python and What is it *Not*?

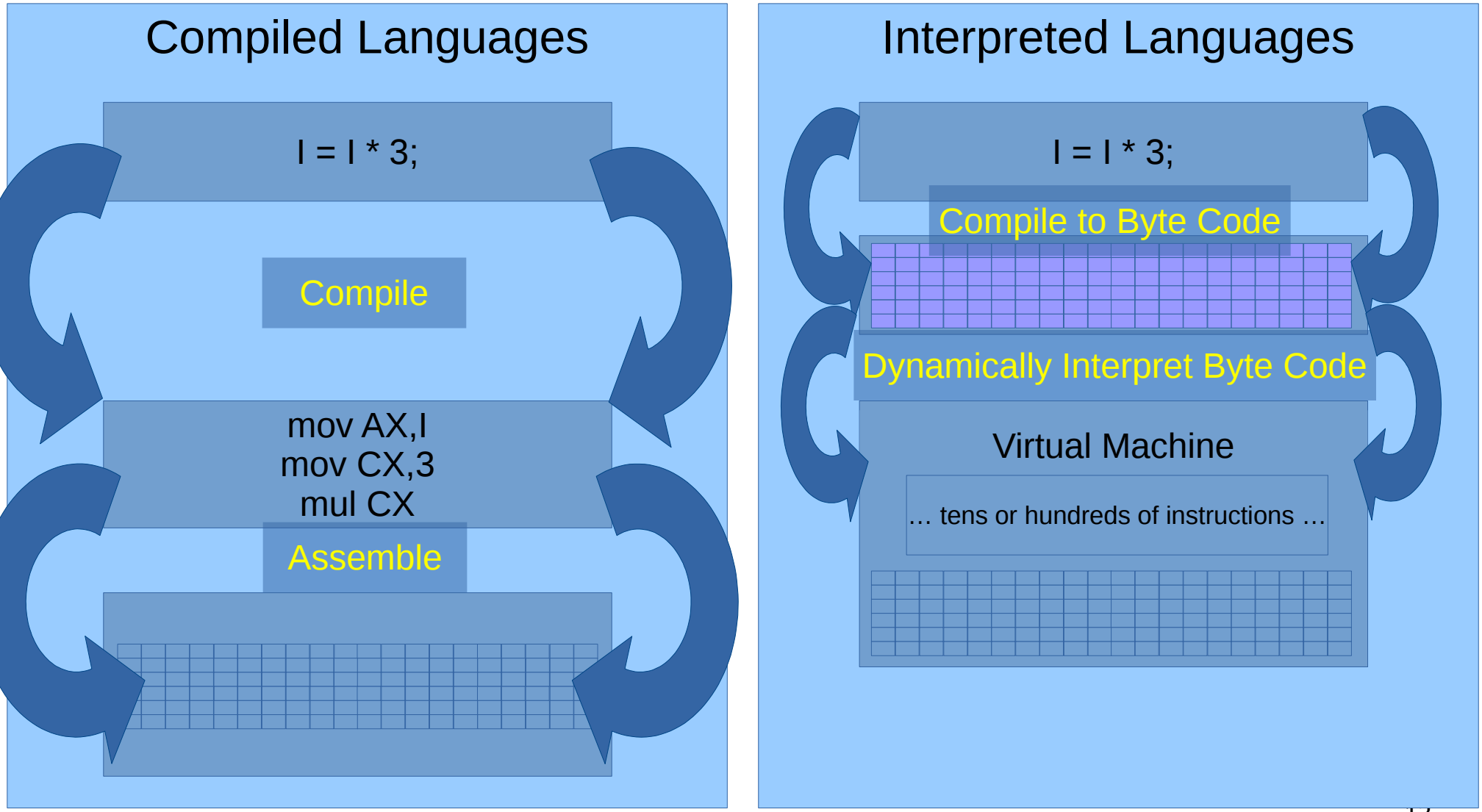
Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together... more text …

Python is used for numerical computing, machine learning, data visualization, and many more. It is an ecosystem.

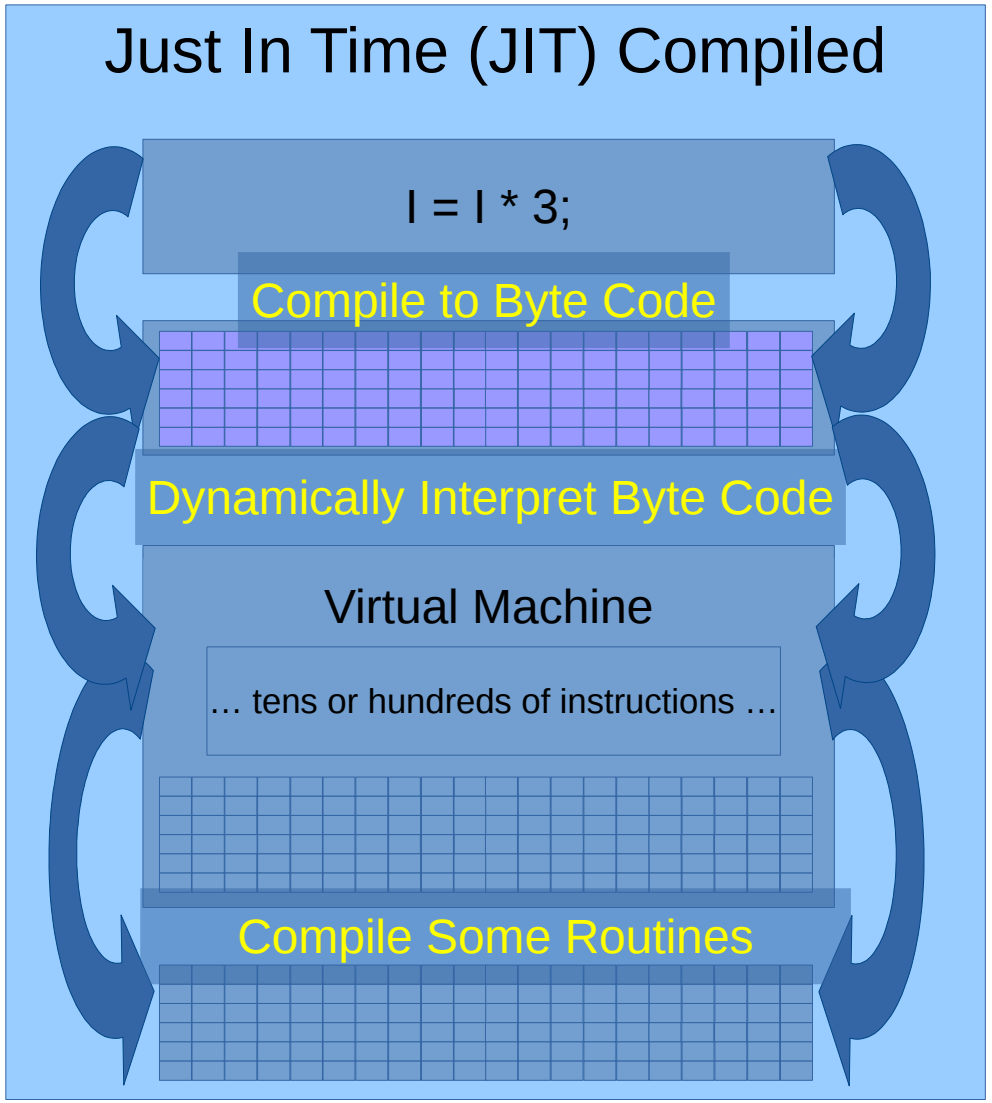
Compiled, Interpreted, and JIT
Compiled Languages

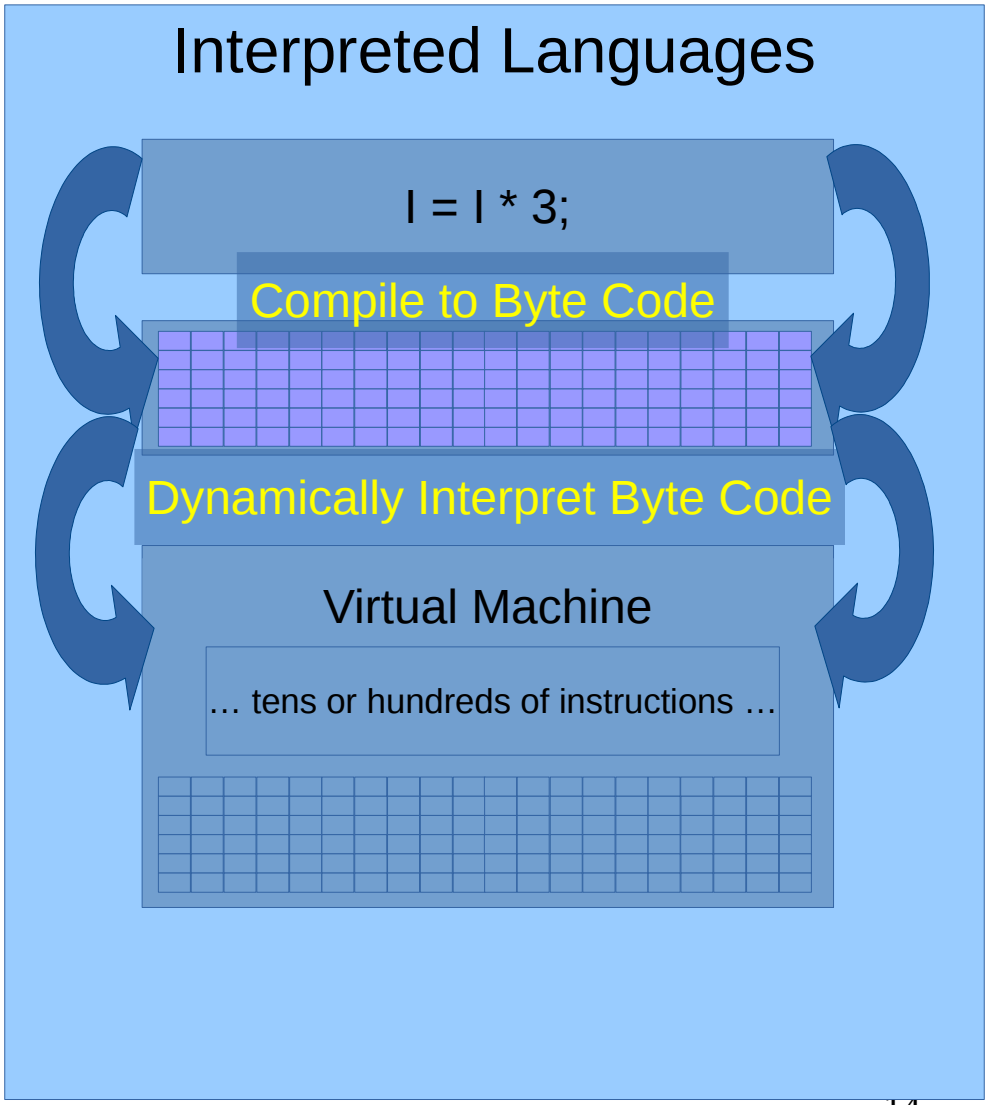


Compiled, Interpreted, and JIT
Interpreted Languages



Compiled, Interpreted, and JIT
Just In Time (JIT) Compiled



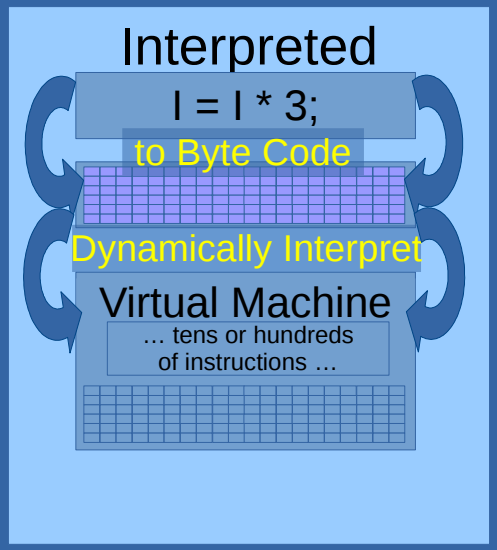


Compiled, Interpreted, and JIT

Pythons are

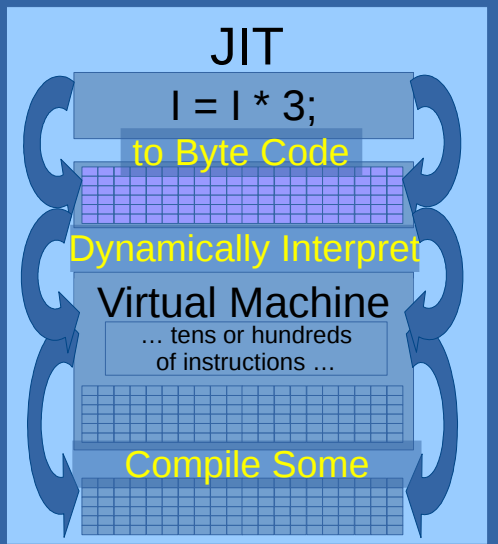
Interpreted

- CPython
  - Reference implementation



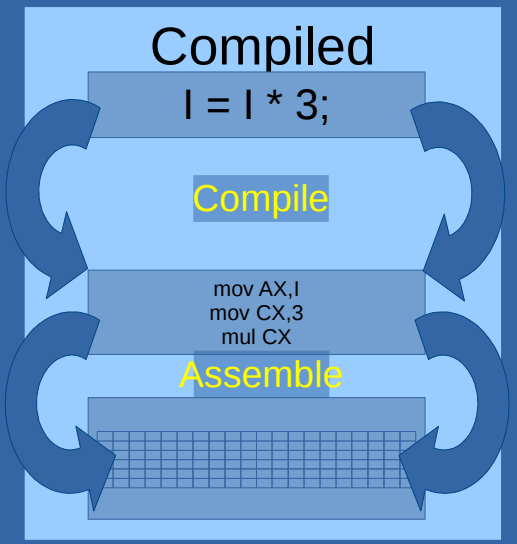
JIT+Interpreted

- Numba
- Jython
  - Uses Java Virtual Machine
- PyPy



Compiled

- Cython
  - Python to C converter



What is Python and What is it *Not*?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together... more text …

Python is used for numerical computing, machine learning, data visualization, and many more. It is an ecosystem.

Dynamic vs. Static
Example Python Program

```
A=1
B=1
print("I will add 1 and 1")
def print_x_plus_y(x, y):
    result = x + y
    print(f"{result}={x+y}")
print_x_plus_y(A,B)
```

- Will print:

  I will add 1 and 1

  2=2

Dynamic vs. Static
Dynamic Typing

```
A=1
B="apple"
print("I will add 1 and 1")
def print_x_plus_y(x, y):
    result = x + y
    print(f"{result}={x+y}")
print_x_plus_y(A,B)
```

- Will fail!
  - "1 + apple" is meaningless
- CPython does not know this until the program reaches:

```
result = x + y
```

Dynamic vs. Static
Static Typing

```
A=1
B="apple"
print("I will add 1 and 1")
def print_x_plus_y(
        x: int, y: int):
    result = x + y
    print(f"{result}={x+y}")
print_x_plus_y(A,B)
```

- Python will find the error as soon as it compiles the file!

Dynamic vs. Static
The Power of Dynamic Typing

```
def print_x_plus_y(x, y):
    result = x + y
```

- Same code for different types:
  - “abc” + “def” = “abcdef”
  - 1 + 1 = 2
  - [1,2,3] + [4,5,6] = [1,2,3,4,5,6]
  - … and many more …
  - All from: result = x + y

What is Python and What is it *Not*?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together... more text …

Python is used for numerical computing, machine learning, data visualization, and many more. It is an ecosystem.

Python is
an Ecosystem

- Python does:
  - Machine learning
  - Graphics
  - Numerics
  - etc.
- Python is an ecosystem

Python is
Glue

- ~~Python~~ does:
  - Machine learning
  - Graphics
  - Numerics
  - etc.
- Python is an ecosystem

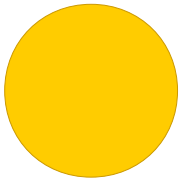
- Many “Python” packages are wrappers around C, CUDA, Fortran, C++, etc.
  - Cython facilitates this.
- Python is glue

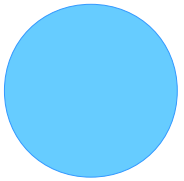
What is Python and What is it *Not*?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together... more text …

Python is used for numerical computing, machine learning, data visualization, and many more. It is an ecosystem.

Object-Oriented Programming
Without





float x
float y
float r
float red
float green
float blue

float x
float y
float r
float red
float green
float blue

brightness_temperature(
red,green,blue)

minimum_covering_circle(
x1,y1,r1,x2,y2,r2)

saturation(
red,green,blue)

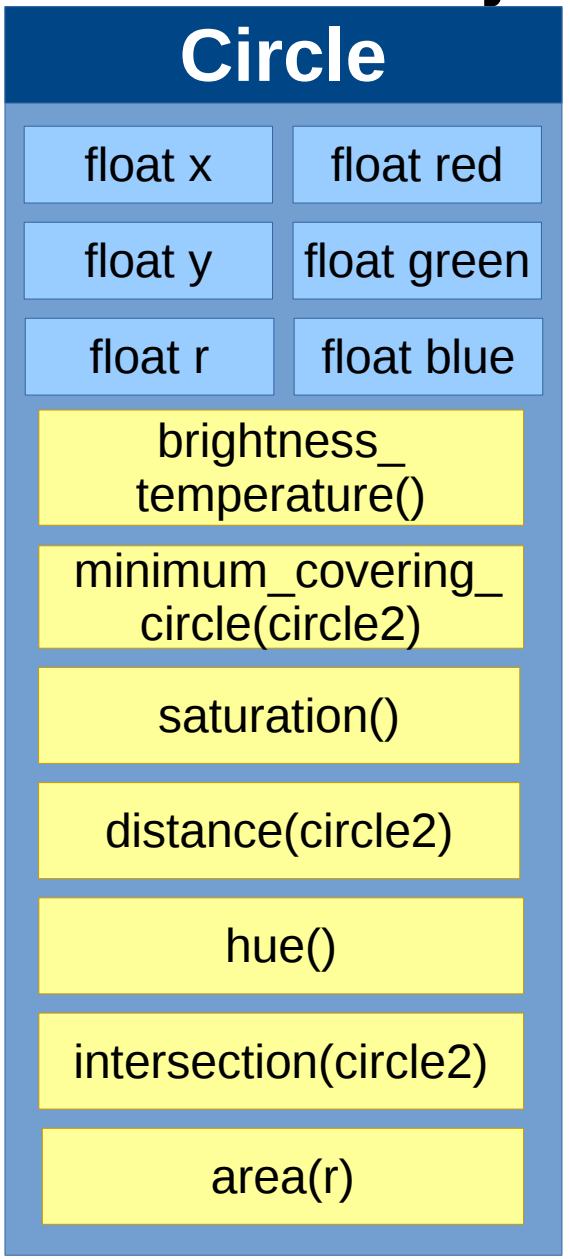
distance(
x1,y1,r1,x2,y2,r2)

hue(
red,green,blue)

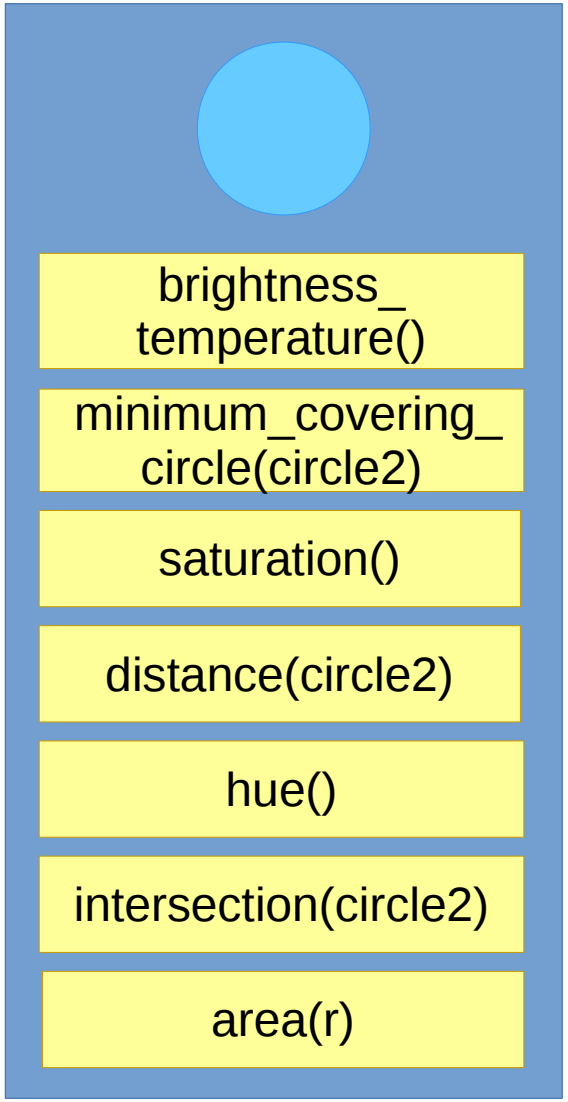
intersection(
x1,y1,r1,x2,y2,r2)

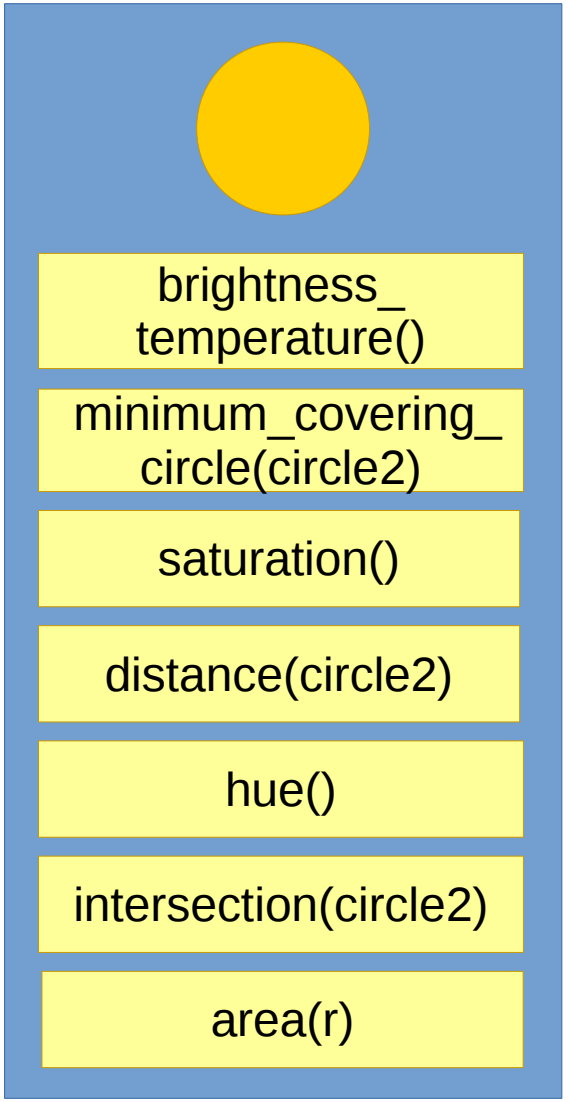
area(r)

Object-Oriented Programming
Implementation – Object's Class



Object-Oriented Programming
Interface – Object's Instance





Running Python

- Note to self:
  - ssh to jet-rsa.rdhpcs.noaa.gov
  - Use the screen session on fe4
  - See windows #3 (interactive) and #4 (script)
  - Interactive commands are in script test.py.

What is Python and What is it *Not*?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together... more text …

Python is used for numerical computing, machine learning, data visualization, and much more. It is an ecosystem.

Conclusion
Python is...

- An ever-changing ecosystem:
  - Multiple implementations.
  - Peer-reviewed recommendation process.
  - Numerous, redundant, actively-developed, libraries.
- Flexible:
  - Compiled, interpreted, or just-in-time.
  - A high-level language, low-level if needed.
  - Dynamically typed, scoped, etc. but can be static (to some extent).
  - Object-oriented, or not, as desired
- Glue
  - Easy to plug other languages into Python.
  - Easy to pass data between many libraries.

End.

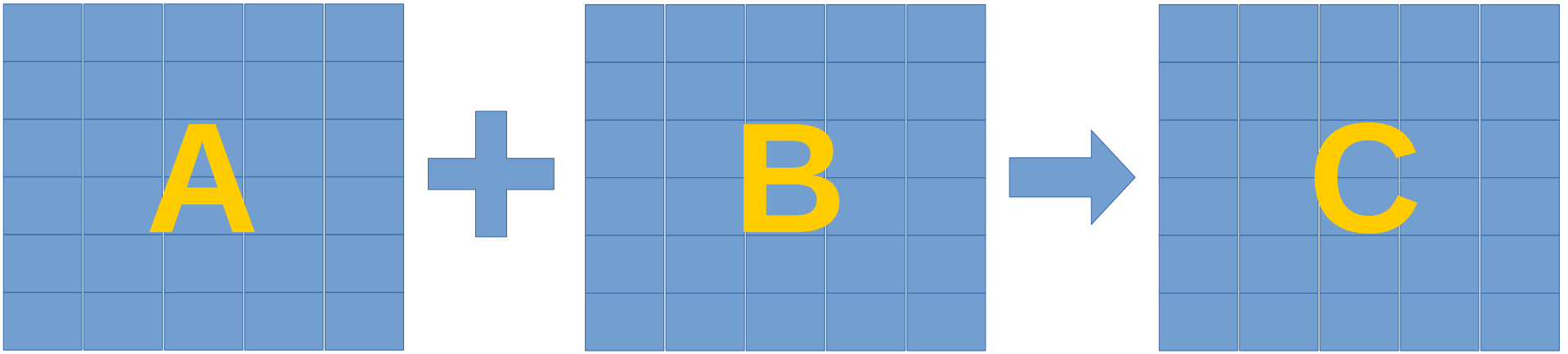
Backup Slides

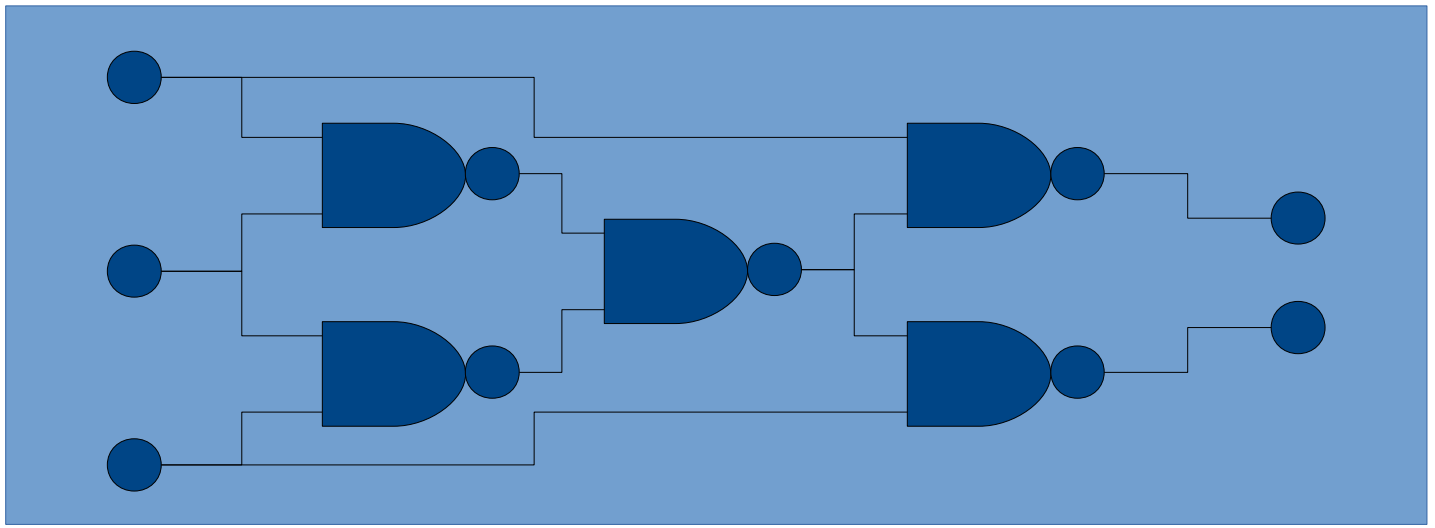
- Assembly: A Lower Level than Low-Level Languages
- NCEP Language Review, Unified Workflow Project
  - Python 2 vs. 3
  - Language Choices
  - Issues in csh
  - Other Languages

Assembly

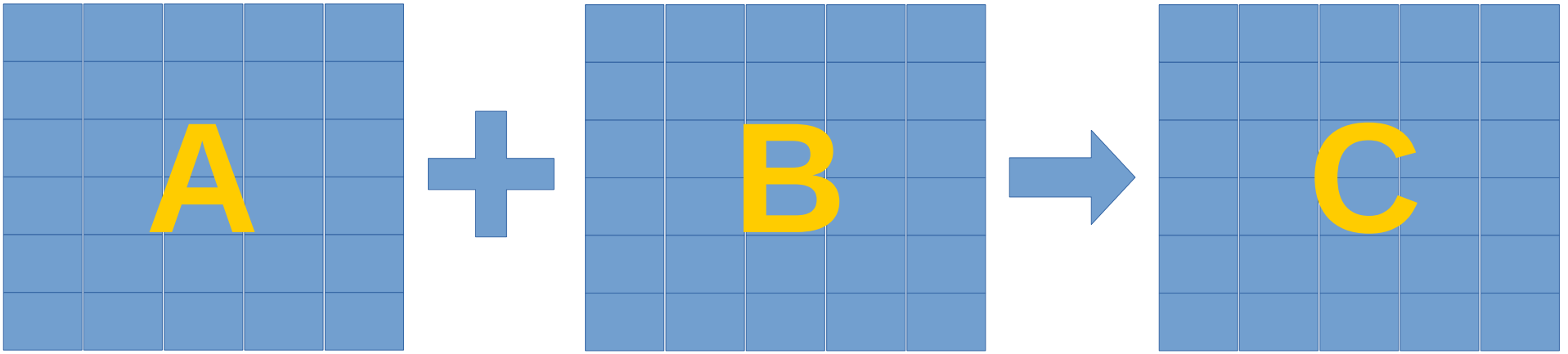
A Lower Level than Low-Level Languages

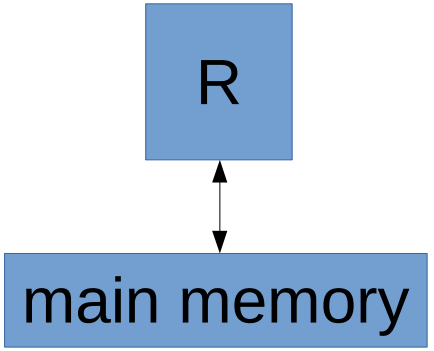
What is Python and What is it *Not*?
High-Level vs. Low-Level Languages





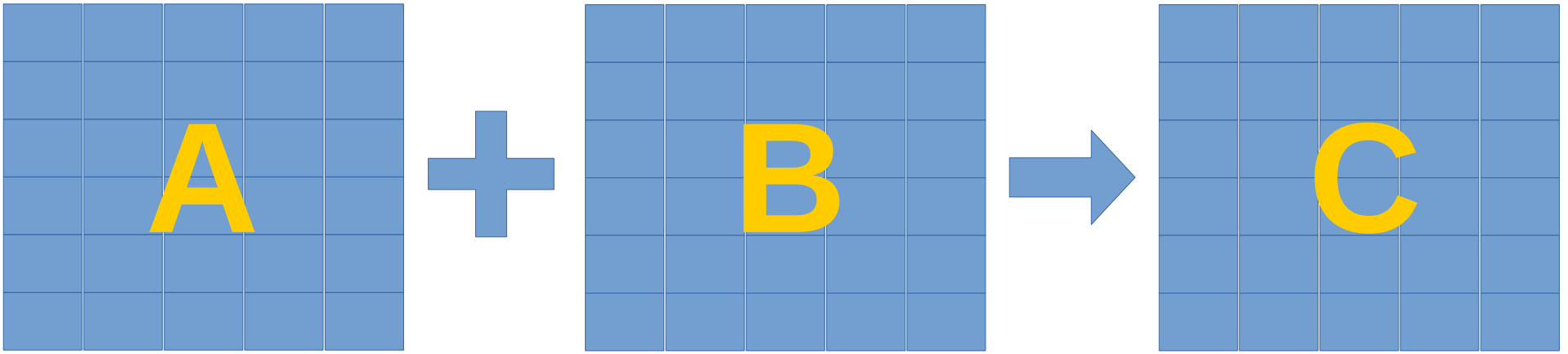
What is Python and What is it *Not*?
High-Level vs. Low-Level Languages

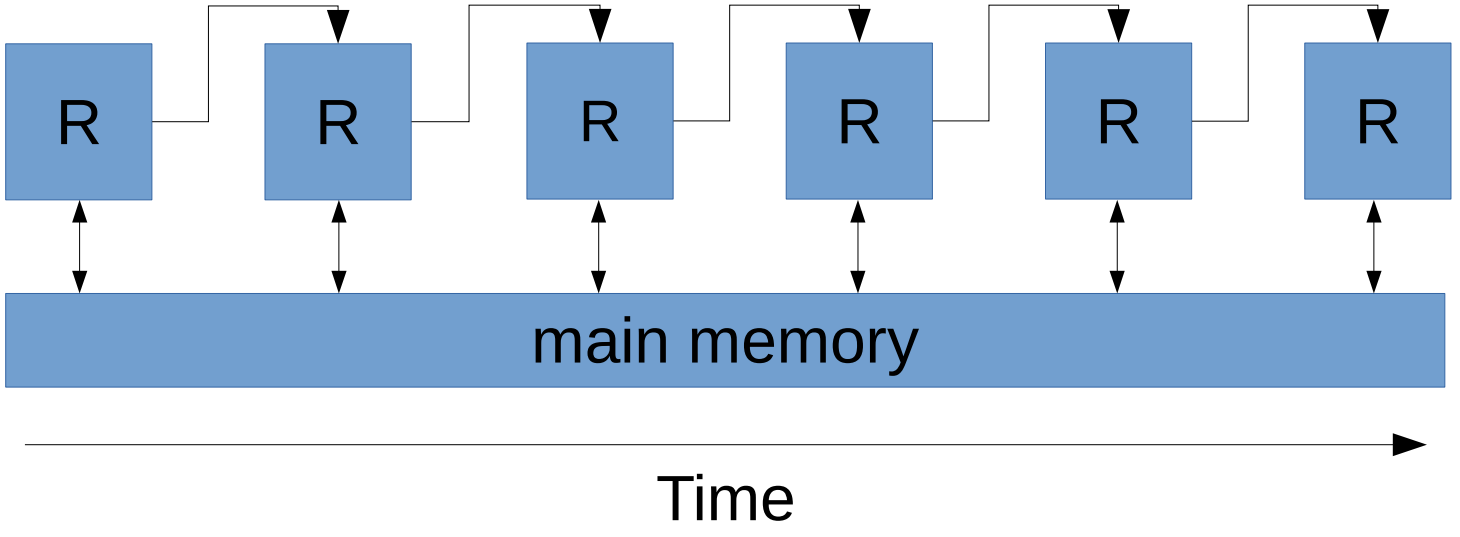




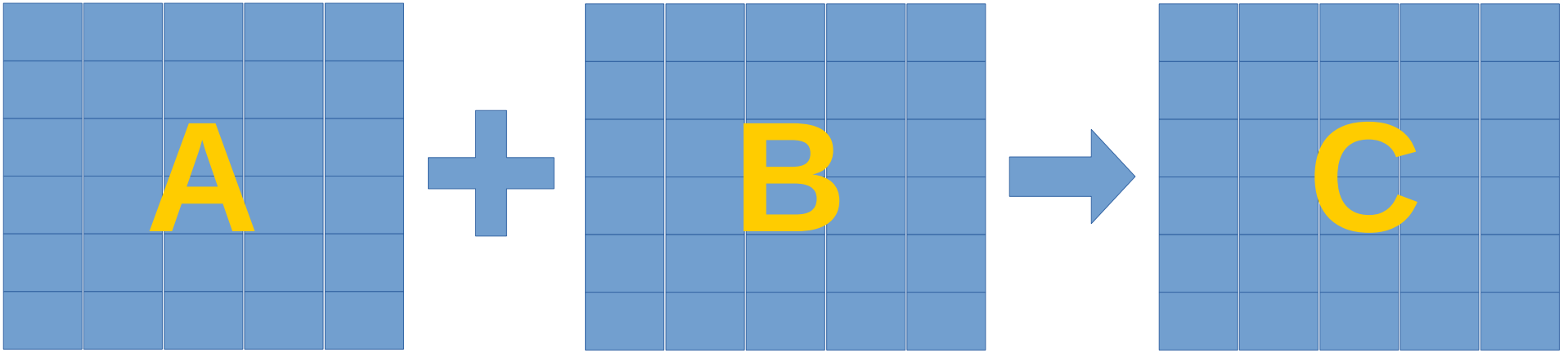
What is Python and What is it *Not*?

High-Level vs. Low-Level Languages



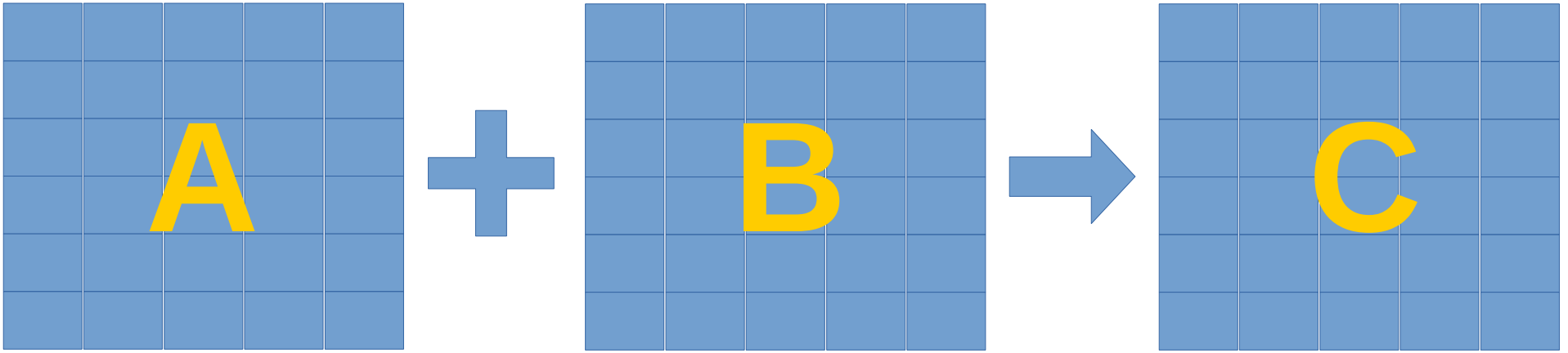


What is Python and What is it *Not*?
High-Level vs. Low-Level Languages



- Assembly
  - mov RAX,n
  - mov RDX,m
  - mul RDX
  - push RAX # n * m
  - mov RBX, array_a_start
  - mov RSI, array_b_start
  - mov RDI, array_c_start
  - mov RCX,0
  - loop_top:
  - mov RAX, [RBX,RCX,8]
  - mov RDX, [RSI,RCX,8]
  - add RAX,RDX
  - mov [RDI,RCX,8],RAX
  - inc RCX
  - mov RAX,[RSP]
  - cmp RAX,RCX
  - jnz loop_top

Languages: High vs Low
Low-Level Languages



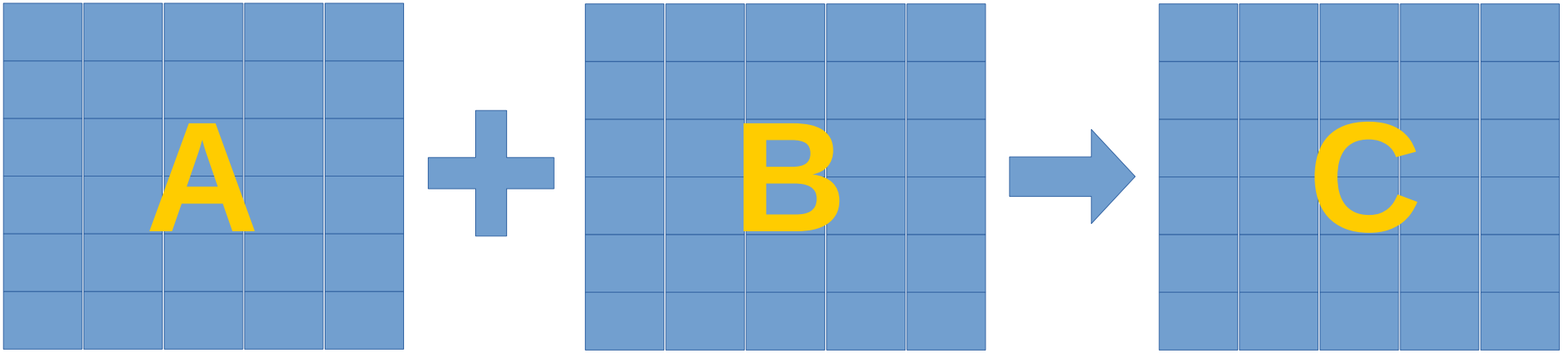
- Low-level Fortran

```
do j=1,m
  do i=1,n
    c(i,j) = a(i,j) + b(i,j)
  enddo
enddo
```

- Low-level C

```
for(j=0;j<m;j++)
  for(i=0;i<n;i++)
    c[j][i]=a[j][i]+b[j][i]
```

Languages: High vs Low
High-Level Languages



- High-level Fortran

  C = A + B

- High-level C++

  C = A + B;

- Python with numpy

  C = A + B

- High-level R

  C <- A + B

Taken, with permission, from NCEP language evaluation

Python 2 vs. 3

Python 2 vs. 3
Language Evaluation for NCEP, 2017

CPython 2.7 vs. CPython 3.6 (by NCEP priorities)

Python 2: **+0**	Python 3: **+3**	– Better exception handling → better logging
Python 2: **+0**	Python 3: **+1**	– Improved to prevent stupid mistakes
Python 2: **+2**	Python 3: **+0**	– Native ASCII becomes native Unicode
Python 2: **+0**	Python 3: **+2**	– Concise, clear, code (~1.5-5x fewer lines)
Python 2: **+0**	Python 3: **+2**	– Python 2.7 end-of-life date 2020
Python 2: **+0**	Python 3: **+0**	– Installation – trivial for either
Python 2: **+0**	Python 3: **+1**	– Training – usually python 3 these days
Python 2: **+2**	Python 3: **+9**	Totals

Python 3 Improvements
A few short examples

- Simple container classes trivial to declare:
  - Point=namedtuple(
    'Point',["x","y","z"])
- vs Python 2:

```
class Point(object):
    def __init__(self,x,y,z):
        super(Point,self).__init__(self)
        self._x,self._y,self._z = (x,y,z)
    def getx(self): return self._x
    def setx(self,x): self._x=x
    def delx(self,x): self._x=None
```

  - Python 2 example continued

```
x=property(getx,setx,delx)
def gety(self): return self._y
def sety(self,y):
    self._y=y
def dely(self,y):
    self._y=None
y=property(gety,sety,dely)
def getz(self): return self._z
def setz(self,z):
    self._z=z
def delz(self,z):
    self._z=None
z=property(getz,setz,delz)
```

Python 3 Improvements
A few short examples

- Prevent common errors:
  - Indentation tabs are parse-time syntax errors
  - super() and new/old style classes improved
    - Python 2: have to derive from “object” and pass class, self to superclass constructor

      ```
      class Point(object):
          def __init__(self,x,y,z):
              super(Point,self).__init__(self)
      ```

    - Python 3: all classes are new style, simpler super()

      ```
      class Point:
          def __init__(self,x,y,z):
              super().__init__()
      ```

Python 3 Improvements
A few short examples

- Python 3.6 added literal string interpolation
  - critical functionality present in all other scripting languages
  - Major flaw in python until 3.6
- Trivial example:
  - shells: var=(expression) ; echo “$var”
  - Ruby: var=(expression) ; puts “#{var}”
  - Perl: var=(expression) ; print “$var\n”
- And now in python 3.6:
  - Python 3.6: var=(expression) ; print(f’{var}\n’)
- Note:
  - In more complex code, this functionality dramatically reduces code complexity
  - In this trivial example, it doesn’t; this is just to demonstrate the feature.

Taken, with permission, from NCEP language evaluation

Language Choices

Capability Comparison
Categories

- Raking: DANGER - LOW – MED – HIGH

  -2 -1 +0 +1

- Portability – will my code work everywhere?
- Learning curve – for people with no knowledge
- NCEP Knowledge– what NCEP knows
- Outside Knowledge – in CS and geosciences
- Versatile – Can it simplify development and maintenance?
  - Core – standard distribution only
  - All – with common, high-reliability packages
  - (core+all) / 2
- Other – considerations specific to that language

Capability Comparison
Special Notes

- bash, ksh: *advanced language extensions* beyond sh
- sh = 100% POSIX-compliant sh
- csh lacks basic language functionality
  - Not versatile: Large parts of production suite would need to be re-implemented in executables, or call bash/ksh scripts
- python, ruby, perl – can replace many small executables with simple functions ("versatile")
- ruby "versatile" category
  - "Versatile core" is for ISO/IEC-compliant Ruby (1.8.7)
  - "Versatile all" - common extensions

Capability Comparison
Operational Languages

	Portability	Learning Curve	Existing Knowledge		Versatile (core+all)/2		Other	Total
			NCEP	Outside	core	all		
advanced ksh	Conflicting implementations	Steep	ample	minimal	med	med		**-3**
csh	Conflicting implementations	Medium	some	some	near zero	near zero		**-4**
advanced bash	Major version variance	Steep	ample	minimal	med	med		**-2**
POSIX sh	ISO Standard (POSIX)	Medium	ample	some	med	med	Always installed	**+3**
Python 2	Uniform across platforms	Teaching Language	some	ample	high	high	later	**+2**
Python 3							slide	**+4**
perl	Uniform across platforms	Steep	some	some	high	high	cryptic concise	**-1**
ruby	ISO Standard ISO/IEC 30170:2012	Teaching Language	minimal	ample	med	high		**+2.5**

Capability Comparison

What if we trained NCEP in a new language?

	Portability	Learning Curve	Existing Knowledge		Versatile (core+all)/2		Other	Total
			NCEP	Outside	core	all		
advanced ksh	Conflicting implementations	Steep	ample	minimal	med	med		**-3**
csh	Conflicting implementations	Medium	With training	some	near zero	near zero		**-3**
advanced bash	Major version variance	Steep	ample	minimal	med	med		**-2**
POSIX sh	ISO Standard (POSIX)	Medium	ample	some	med	med	Always installed	**+3**
Python 2	Uniform across platforms	Teaching Language	With training	ample	high	high	later	**+3**
Python 3							slide	**+5**
perl	Uniform across platforms	Steep	With training	some	high	high	cryptic concise	**0**
ruby	ISO Standard ISO/IEC 30170:2012	Teaching Language	With training	ample	med	high		**+4.5**

Capability Comparison
Python 2.7 vs. 3.6

Final python 2 vs. current release python 3

Python 2: **+0**	Python 3: **+3**	– Better exception handling → better logging
Python 2: **+0**	Python 3: **+1**	– Improved to prevent stupid mistakes
Python 2: **+2**	Python 3: **+0**	– Stuck with 7-bit ASCII (good for NCEP)
Python 2: **+0**	Python 3: **+2**	– Concise, clear, code (~1.5-5x fewer lines)
Python 2: **+0**	Python 3: **+2**	– Python 2.7 end-of-life date 2020
Python 2: **+0**	Python 3: **+0**	– Installation – trivial for either
Python 2: **+0**	Python 3: **+1**	– Training – usually python 3 these days
Python 2: **+2**	Python 3: **+9**	Totals

Taken, with permission, from NCEP language evaluation

Issues in csh

Issues in csh
Missing Basic Functionality

- Extremely abridged version.
  - No signal handlers
    - Cannot clean up after failed job
    - Cannot contact ecFlow server to report failed job
  - Inconsistent handling of strings with spaces
    - Special syntax needed to handle strings; syntax varies depending on context
    - Effectively, this makes it unusable for such strings
  - Cannot redirect stdout and stderr to different files

Issues in csh
Major Design Flaws

- Extremely abridged version.
  - Ad-hoc parser – must execute statements to parse them
    - More on later slides
  - Inefficient syntax for complicated expressions
    - More on later slides
  - No functions
    - Aliases are simply pasted code; they lack most capabilities of functions such as arguments, nested scopes, separate return values

Issues in csh
Ad-hoc Parser

- Inconsistent syntax – makes it error-prone
  - Excellent example from wikipedia:
    - Makes an empty file:

```
if ( ! -e myfile ) echo mytext > myfile
```

    - Puts “mytext” in a file

```
if ( ! -e myfile ) then
   echo mytext > myfile
endif
```

Issues in csh
Ad-hoc Parser

- Inconsistent syntax – makes it error-prone
  - From Berret, et.al. 2009
    - Suppose $A is undefined.
    - Statement has no effect, as it should:
      - if ($?A) echo A is defined
    - Statement fails because $A is undefined
      - if ($?A) set B = $A
    - $A is evaluated even though that statement should not be executed.

Issues in csh
Ad-hoc Parser

- From Berret, et.al. 2009
  - Error: “Variable name must contain alphanumeric characters”
    - grep "var" < file
    - grep "$var\$" <file
  - \ is not quoting the $
  - Works:
    - grep "$var"'$' < file
    - set dollar='$'
    - grep "vardollar" < file
  - Trivial in sh-like shells:
    - grep "$var\$" < file
  - Or:
    - grep "var" < file

Taken, with permission, from NCEP language evaluation

Other Languages

BASH

- GNU Bash project – only one implementation
- Generally backward-compatible, but:
  - Major syntactic additions make version dependence problems hard to detect
  - Built-in commands vary from version to version
  - Built-in commands added in later versions
    - Prior bash version used /bin program
    - Now it doesn’t! Functionality changed, maybe not backward-compatible

ksh

- Originated in AT&T but has multiple implementations now.
  - Significant syntactic differences
  - Differences in handling datatypes.
    - Is 013=11 or 13?
  - Built-in commands differ between versions
    - (See bash slides for details.)

Ruby

- Slower than other scripting systems, but
  - Can compile to JVM byte code for faster execution
- Fewer books and forums than Python
- More limited standard library than Python, but similar to Perl
- String processing speed comparable to Perl
- Less usage in AMS, AGU community

Perl

- Extremely concise language.
  - Great for rapid prototyping.
  - Tremendous reduction in code length for many tasks.
  - String processing speed comparable to compiled languages
- Example. Calculate pi in Perl 5:

```
$.=".$]";
$\=2/$.++-$\ for $...1e6;
print
```