Prototyping a 2D Physics-Based Animation with Python

Vincent Bonnet

Python Meetup - March 2019

Content

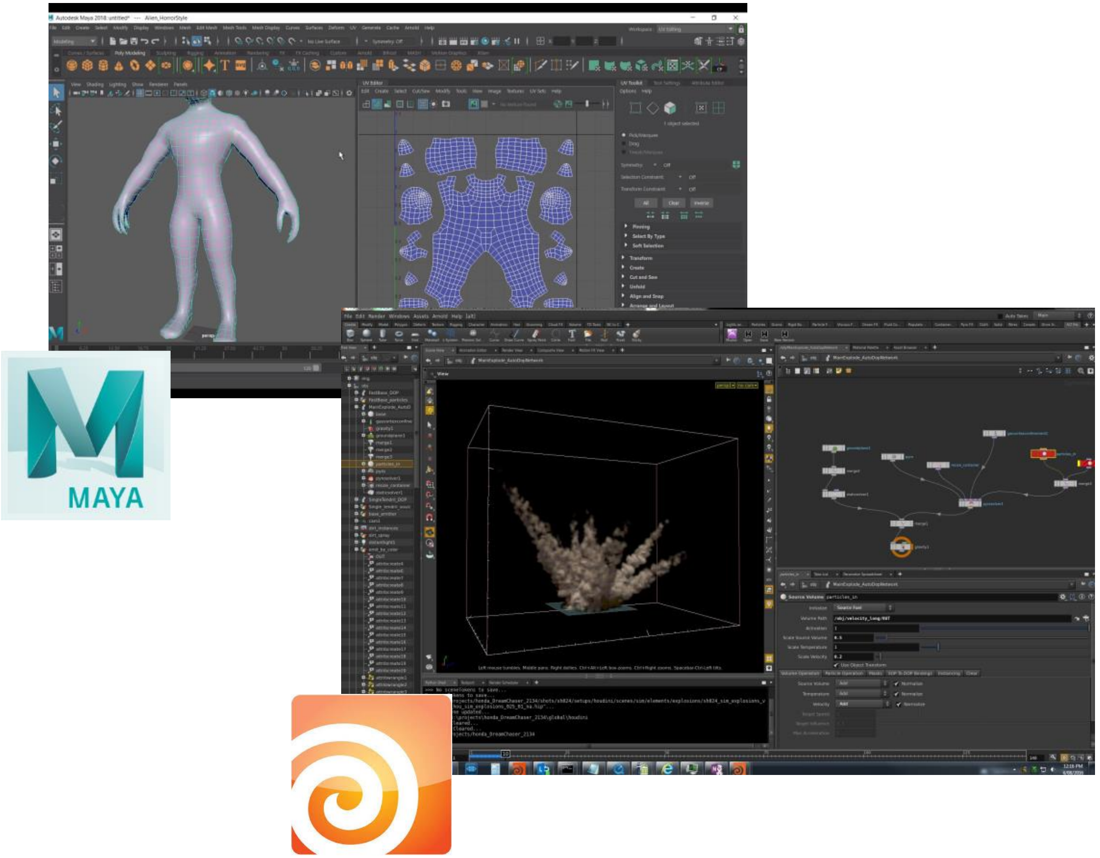
- Integration
  - Standard Plugin
  - Inter-process communication
- Solver
  - Solver in a Nutshell
  - Python Ecosystem

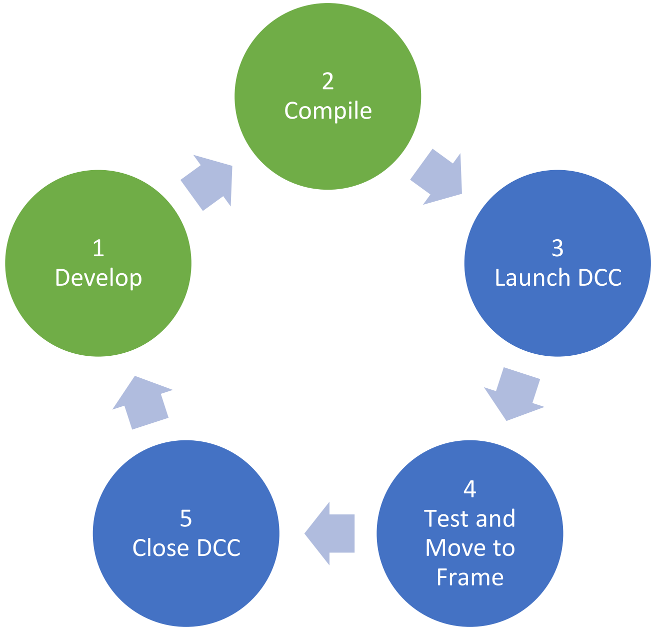
Physics-Based Animation

Part 1 - Integration

Traditional Integration

Plugin System





Inter-process communication

Process A - Client

Inter-process communication

Process A - Client

Python API

Inter-process communication

Process A - Client

Python API

Process B - Client

Inter-process communication

Process A - Client

Python API

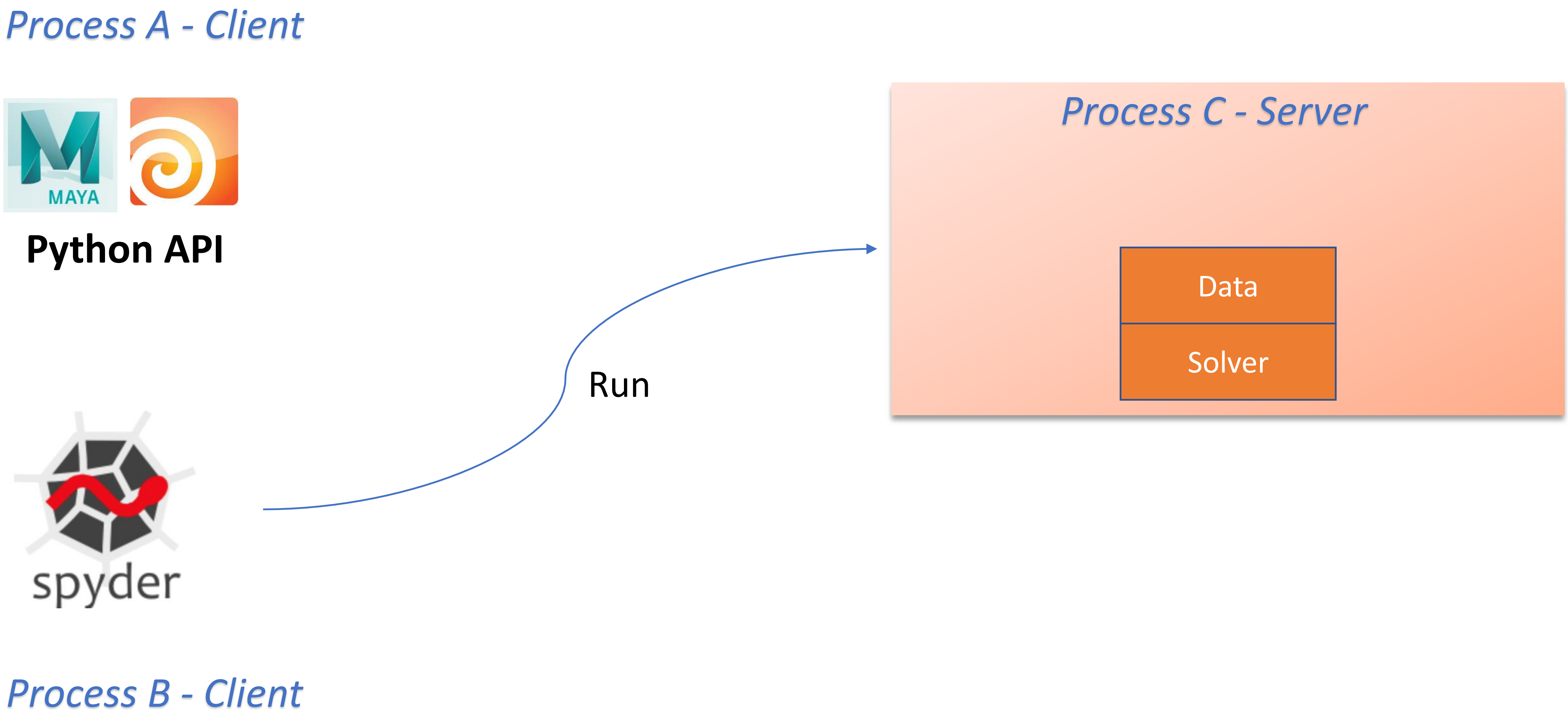
Process C - Server

Data

Solver

Process B - Client

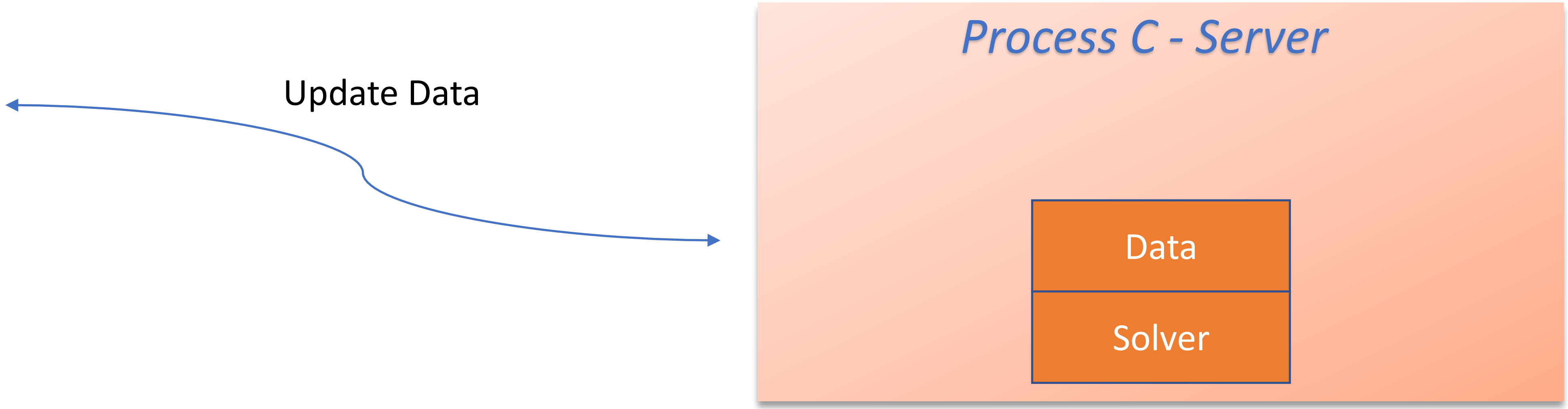
Inter-process communication



Inter-process communication

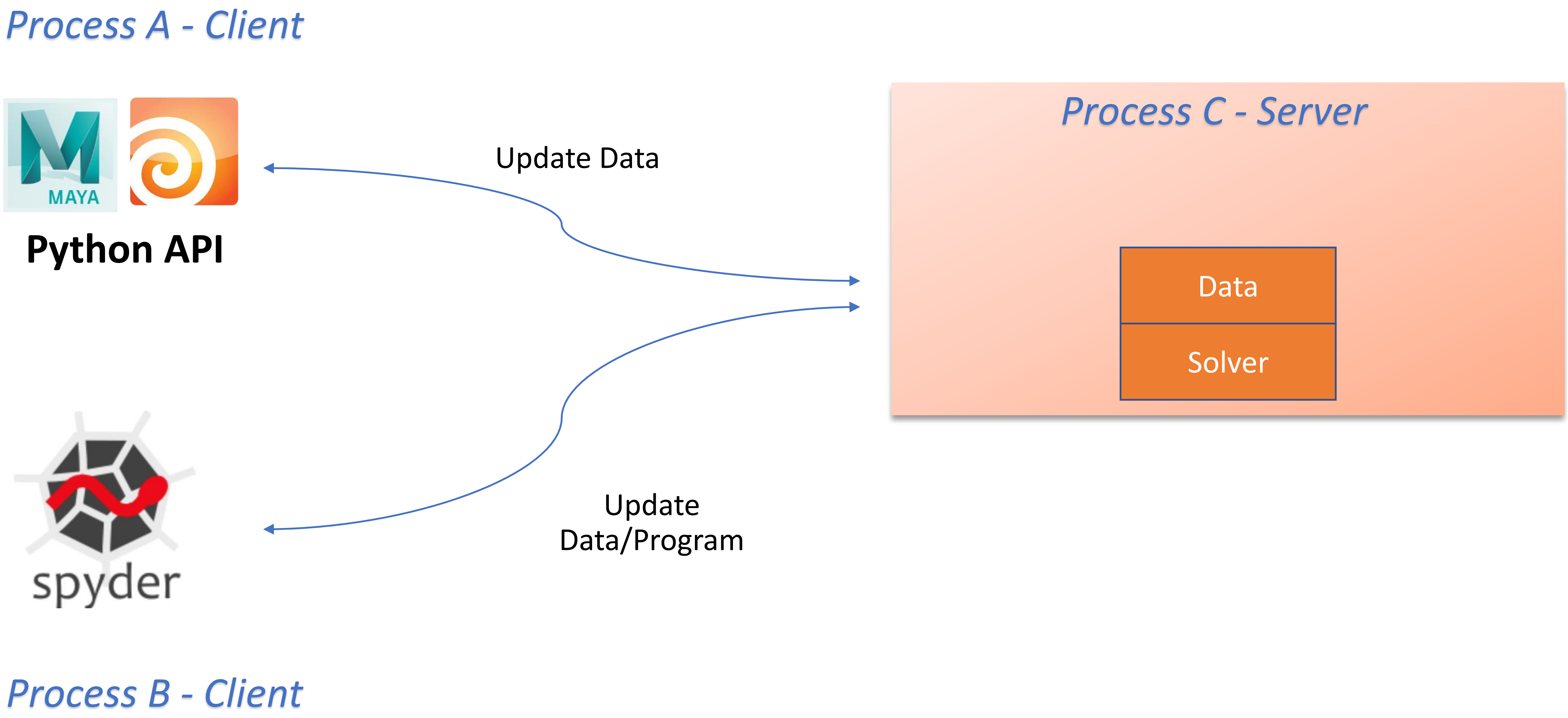
Process A - Client

Python API



Process B - Client

Inter-process communication



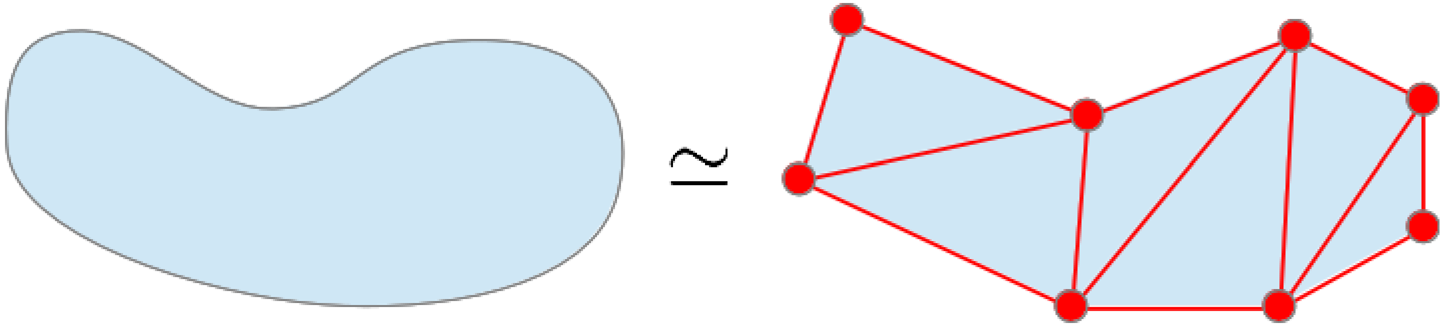

Waiting for
Python 3 ...

Part 2 - Solver

Solver in a Nutshell

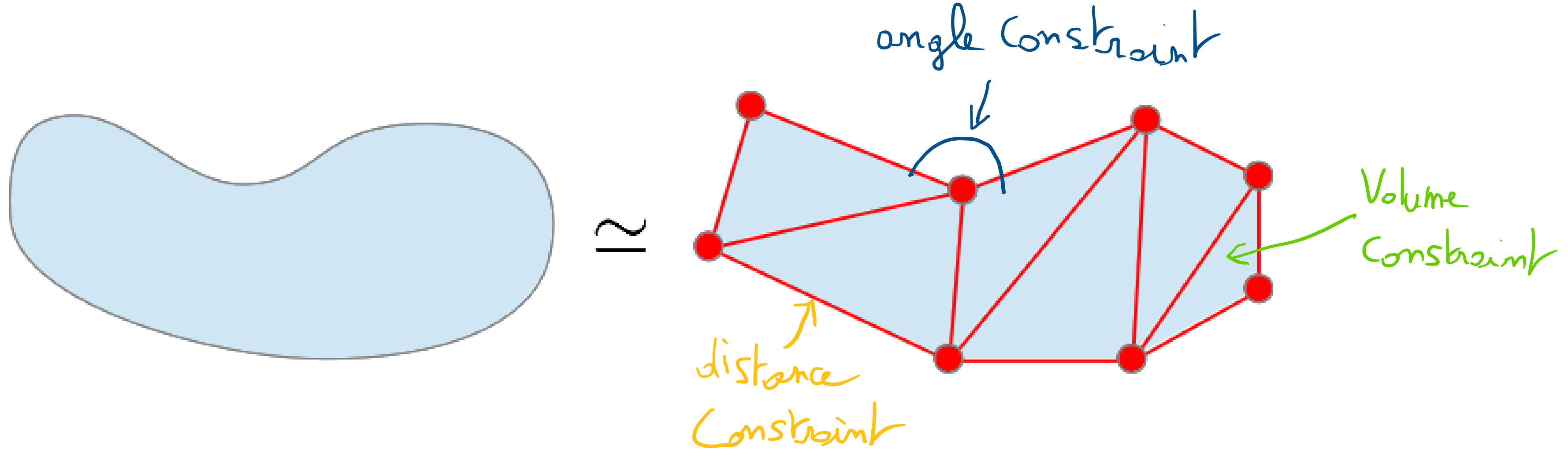
Solver Requirements

Discretization



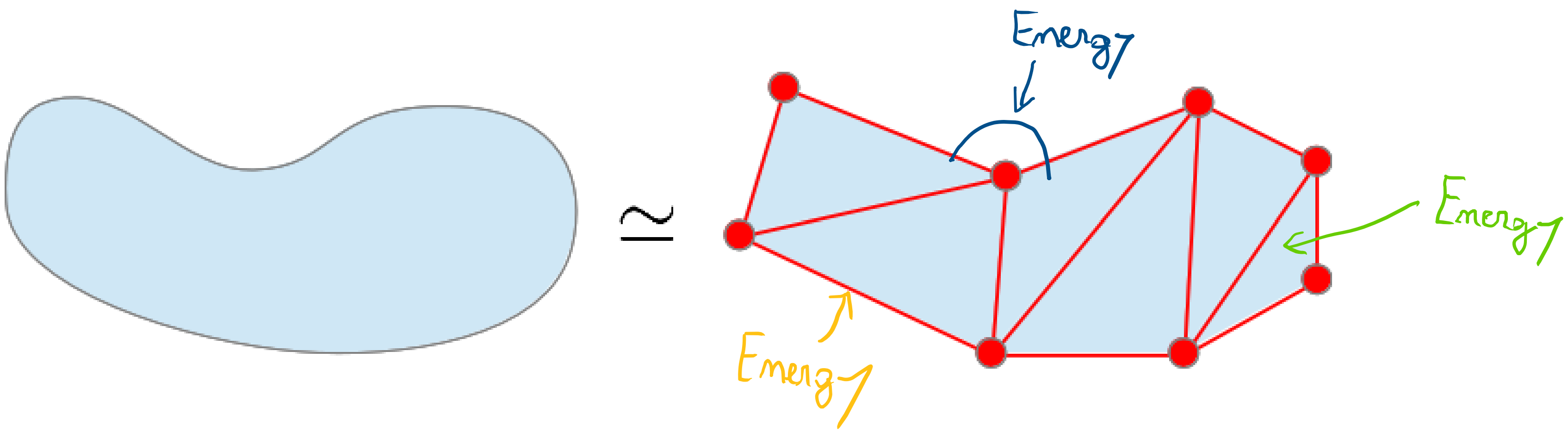
Solver Requirements

Force Computation



Solver Requirements

Force Computation

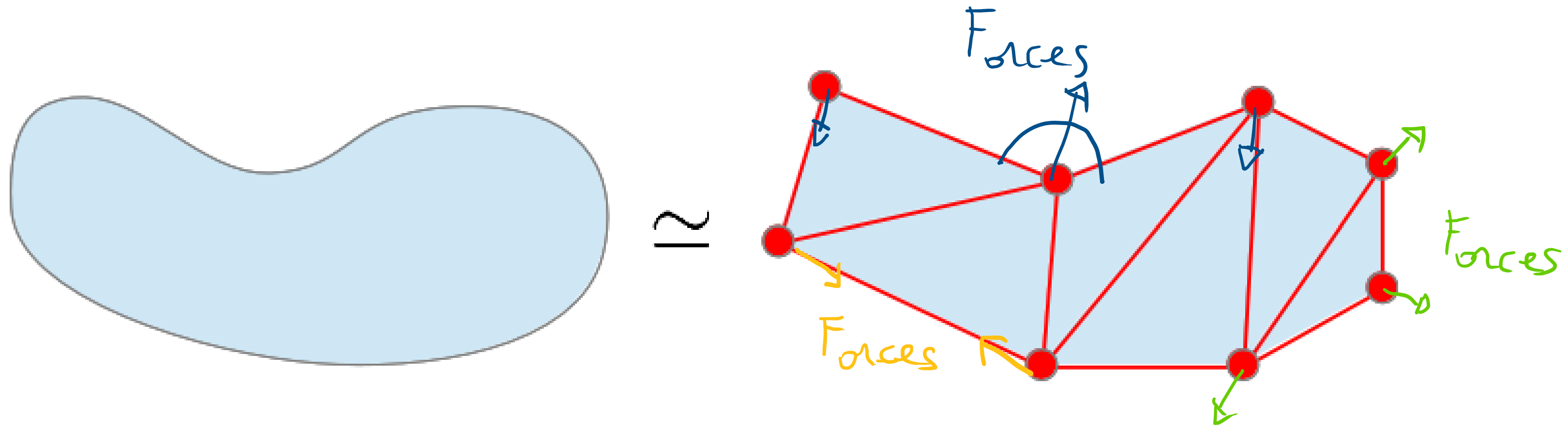


potential energy E based on constraint C

$$E(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \frac{1}{2} k C(\mathbf{x}_1, \ldots, \mathbf{x}_n)^2$$

Solver Requirements

Force Computation



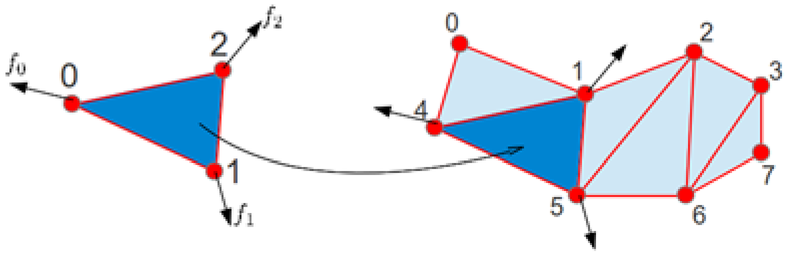
$$\mathbf{F}_j(\mathbf{x}_1,\ldots,\mathbf{x}_n) = -\frac{\partial}{\partial \mathbf{x}_j} E(\mathbf{x}_1,\ldots,\mathbf{x}_n)$$

$$= -kC(\mathbf{x}_1,\ldots,\mathbf{x}_n)\frac{\partial C(\mathbf{x}_1,\ldots,\mathbf{x}_n)}{\partial \mathbf{x}_j}$$

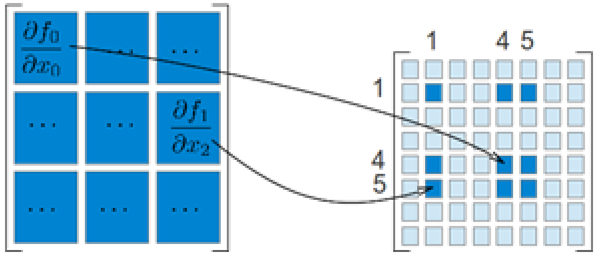
Solver Requirements

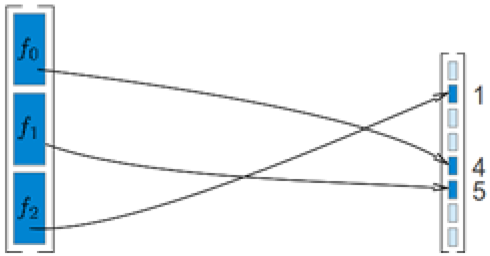
Forward in Time

$$\left(M - \frac{\mathrm{d}f}{\mathrm{d}x} h^2\right) \Delta_v = h\left(f_0 + h\frac{\mathrm{d}f}{\mathrm{d}x} v_0\right)$$

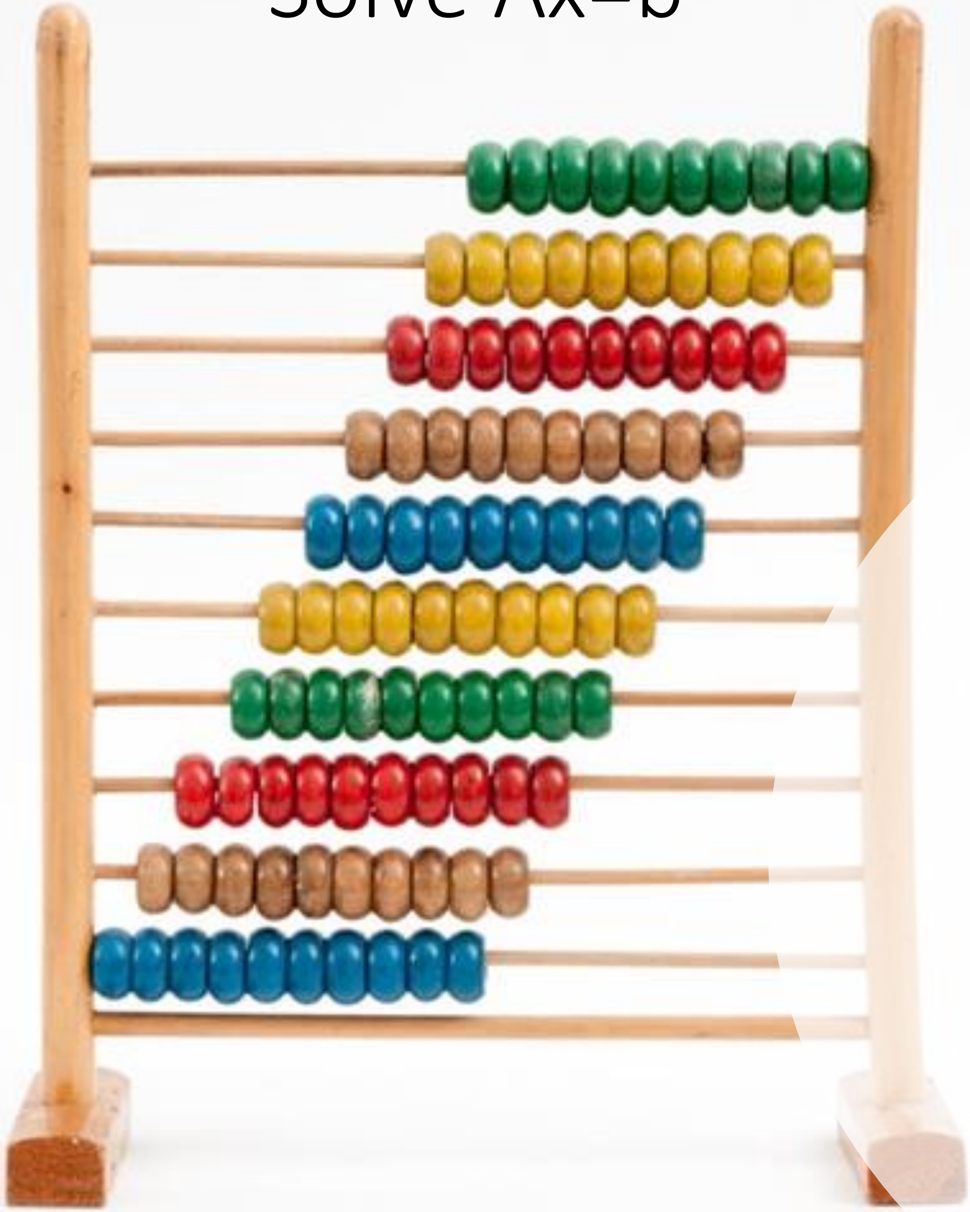


$A_{n \times n}$ $\qquad x_n$ $\qquad b_n$





Solve Ax=b



Do It Again

Implementation

Python + Numba / Julia

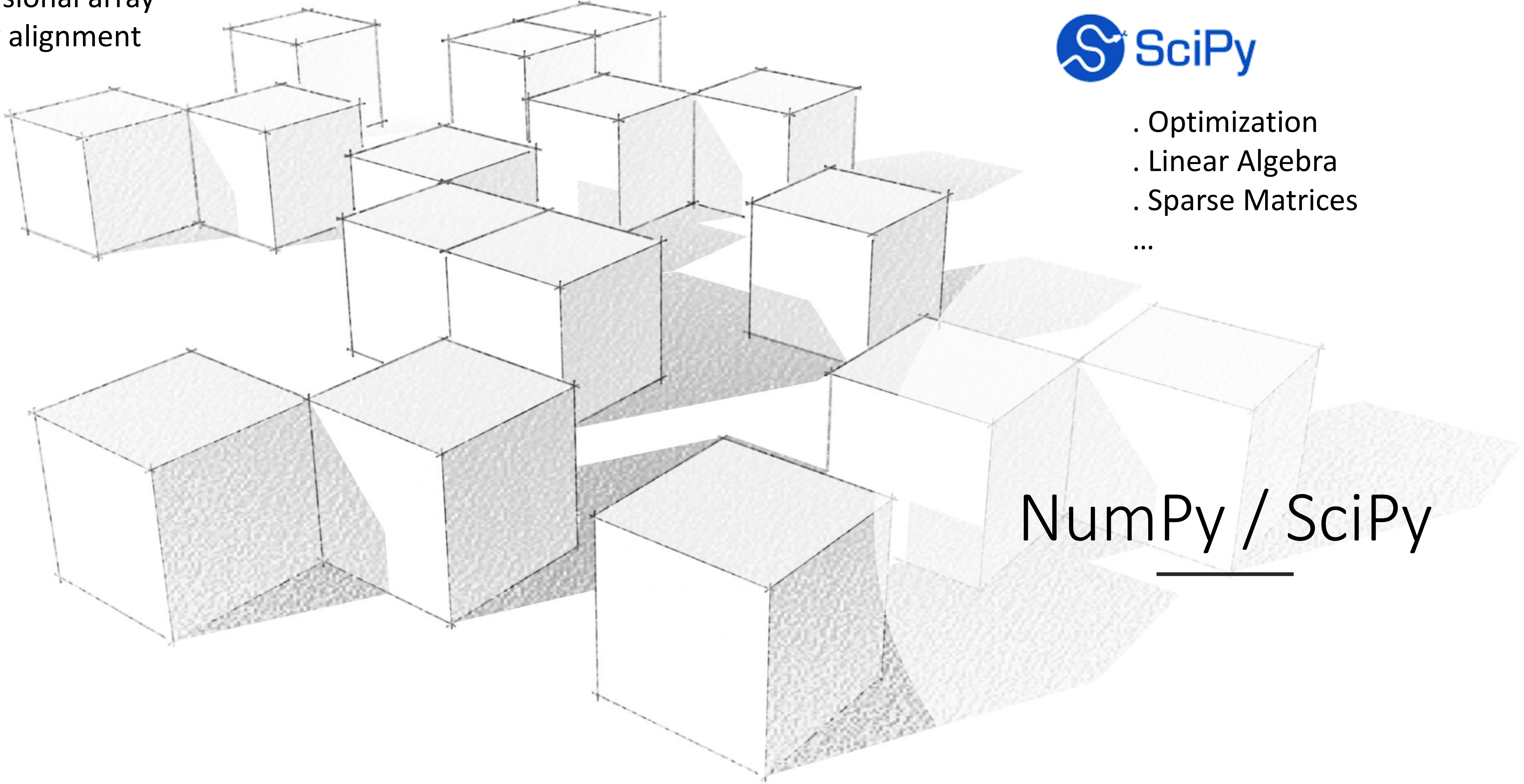
NumPy / SciPy

NumPy

. N-dimensional array
. Memory alignment

SciPy

. Optimization
. Linear Algebra
. Sparse Matrices
…



Numba

- Translates Python functions to optimized machine code at runtime
- CPU and GPU support
- Minimal code changes - @njit decorator

Numba

Benchmark

Apply $\sqrt{tan(x)\,cos(x)}$ on a 1-D array

	Python	Numba	Numba MT
1e6	0.295 sec	0.009 sec	0.002 sec
1e7	3.01 sec	0.069 sec	0.016 sec
1e8	29.50 sec	0.689 sec	0.165 sec
1e9	-	10.939 sec	1.681 sec

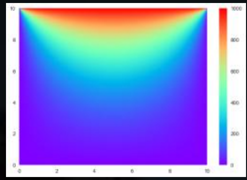
```
@njit(parallel=True)
def run(array):
    for i in prange(array.shape[0]):
        array[i] = function(array[i])
    return array
```

- Translates Python functions to optimized machine code at runtime
- CPU and GPU support
- Minimal code changes - @njit decorator

Numba

Benchmark

Solve $\nabla^2 f = 0$ on a 2-D array



	Python	Numba	Numba MT
64x64 2000 iter.	9.696 sec	0.007 sec	0.012 sec
128x128 10000 iter.	-	0.133 sec	0.131 sec
256x256 50000 iter.	-	2.459 sec	1.401 sec
512x512 250000 iter.	-	50.343 sec	19.872 sec

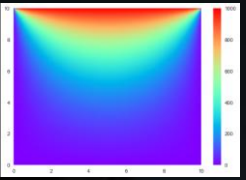
- Translates Python functions to optimized machine code at runtime
- CPU and GPU support
- Minimal code changes - @njit decorator

```
@njit(parallel=True)
def iteration(array):
for i in prange(1, array.shape[0] - 1):
    for j in range(1, array.shape[1] - 1):
        next_x[i][j] = (x[i - 1][j] + x[i + 1][j] +
                        x[i][j - 1] + x[i][j + 1]) * 0.25
```

Numba

Benchmark

Solve $\nabla^2 f = 0$ on a 2-D array



Stencil Op.	Python	Numba	Numba MT
8.192 millions	9.696 sec	0.007 sec	0.012 sec
163.840 millions	-	0.133 sec	0.131 sec
3.2768 billions	-	2.459 sec	1.401 sec
65.536 billions	-	50.343 sec	19.872 sec

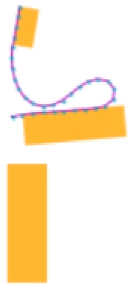
- Translates Python functions to optimized machine code at runtime
- CPU and GPU support
- Minimal code changes - @njit decorator

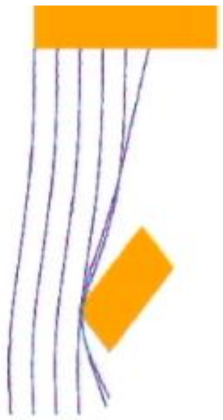
```
@njit(parallel=True)
def iteration(array):
for i in prange(1, array.shape[0] - 1):
    for j in range(1, array.shape[1] - 1):
        next_x[i][j] = (x[i - 1][j] + x[i + 1][j] +
                        x[i][j - 1] + x[i][j + 1]) * 0.25
```

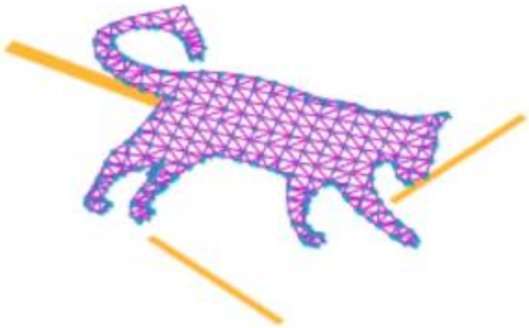
INTEL® DISTRIBUTION FOR PYTHON

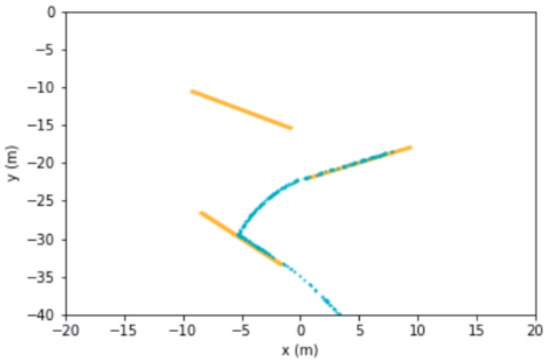
intel Software

- Accelerate NumPy*, SciPy*, and scikit-learn* with integrated Intel® Performance Libraries
- Latest vectorization and multithreading instructions, Numba* and Cython*, composable parallelism with Threading Building Blocks
- No Changes to code









Results