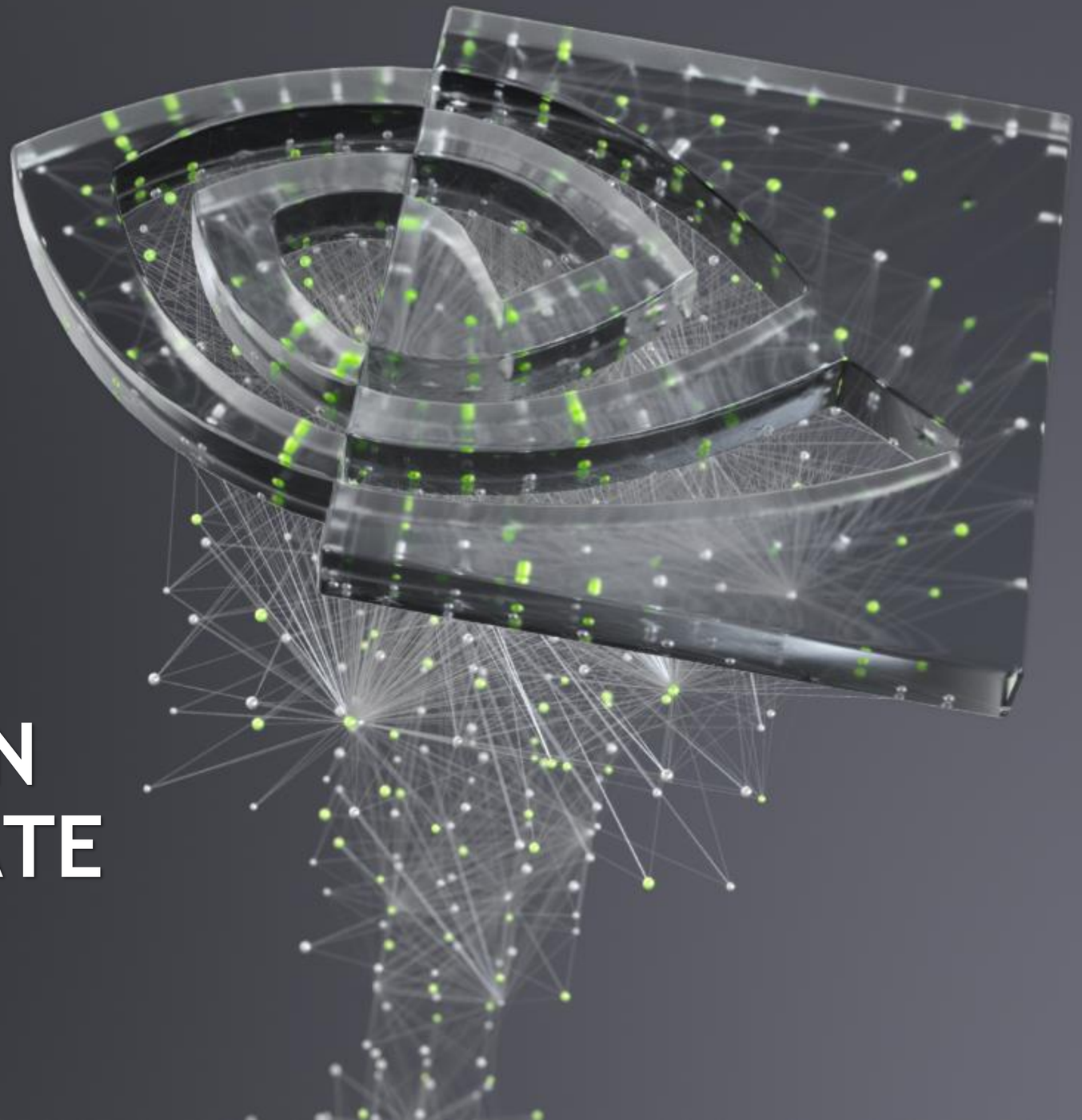
SUPERCOMPUTING IN PYTHON WITH LEGATE

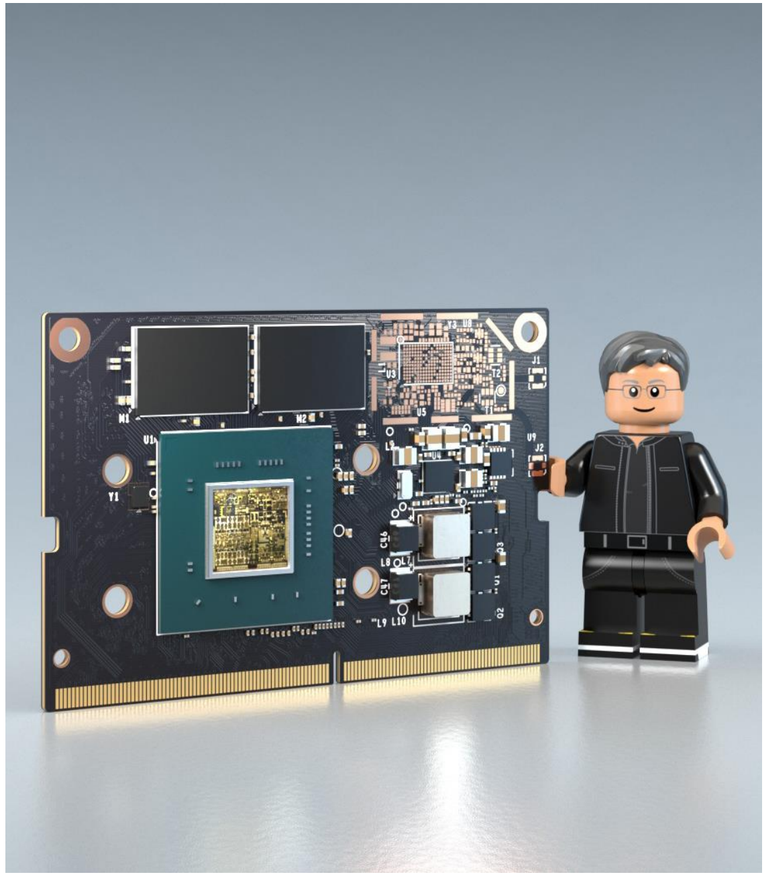
Michael Garland, June 2021



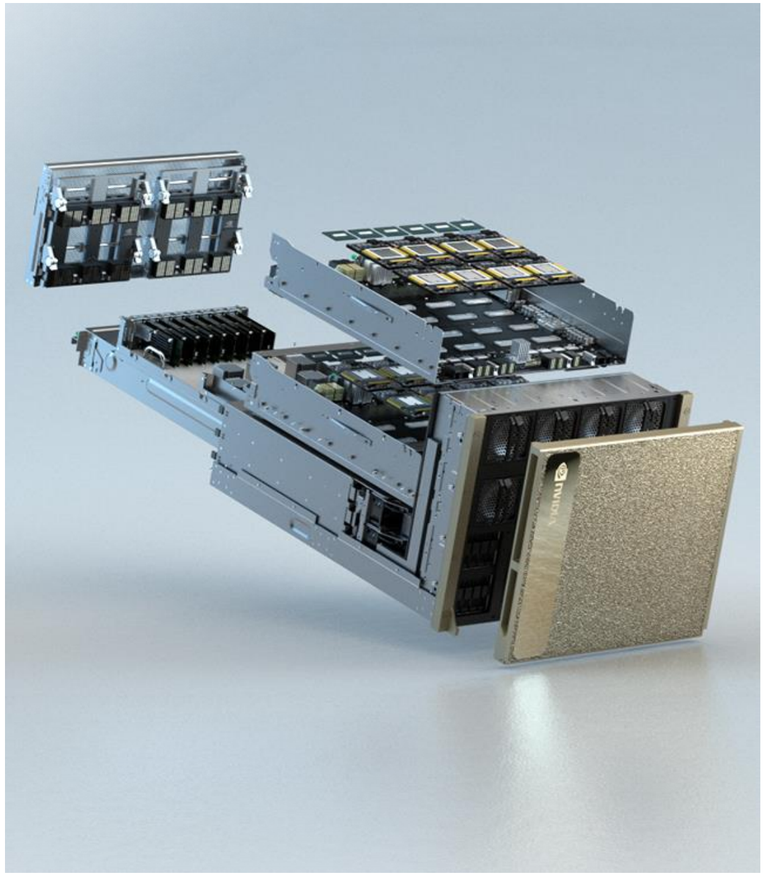
SCALABLE EXECUTION

Achieving high performance at any scale

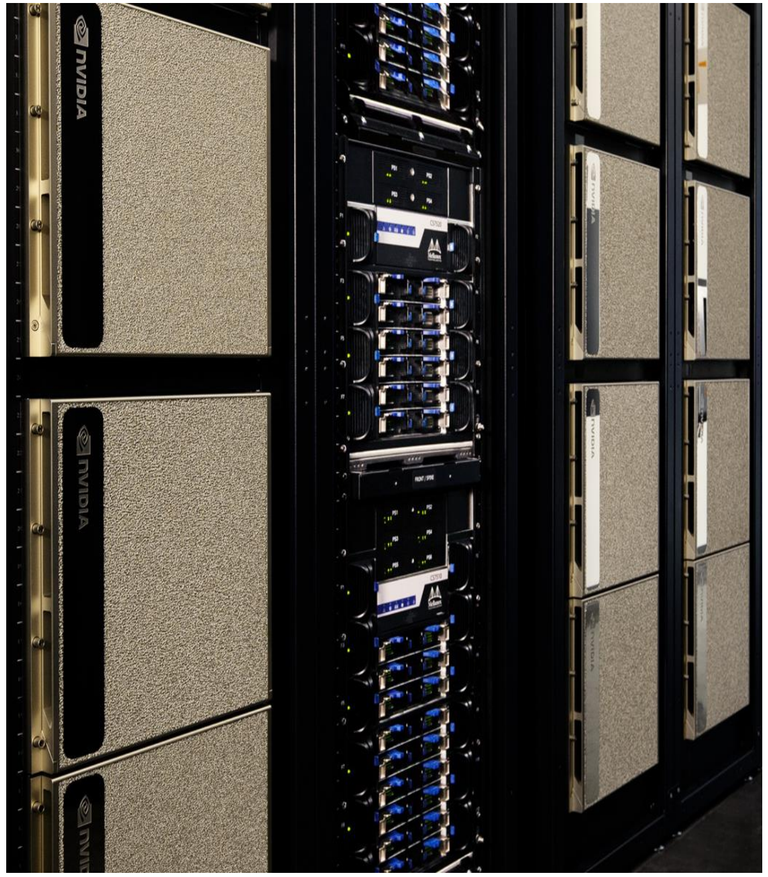
Jetson AGX Xavier



DGX-2



DGX SuperPod



FAMILIAR & CONVENIENT NOTATION

Program at any scale without learning new interfaces

```
import numpy as np

def cg_solve(A, b):
    x = np.zeros(A.shape[1])
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    for i in xrange(b.shape[0]):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)
         if np.sqrt(rsnew) < 1e-10:
            break
        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
     return x
```

- Many existing interfaces expose ample parallelism
- Data often large enough to warrant large machines
- **This code can run on a supercomputer**

MAKE ADOPTION INCREDIBLY SIMPLE

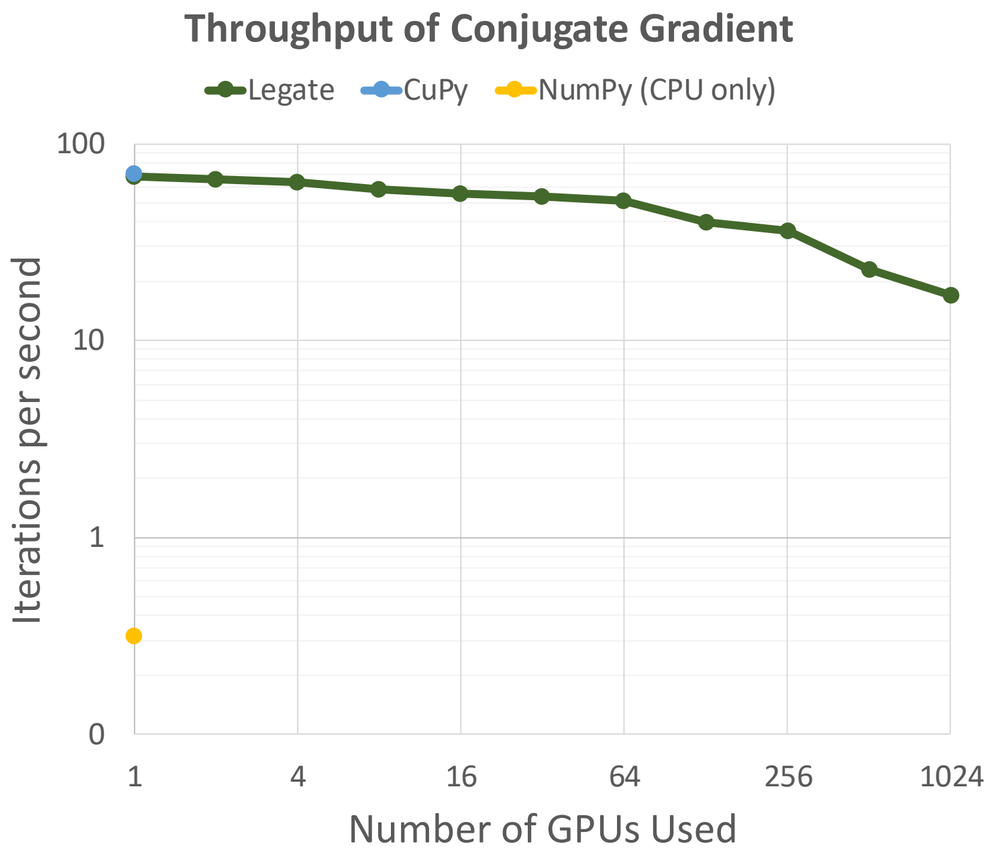
Even if users are a bit arbitrary in their adoption strategy

```
import random, numpy, legate.numpy



# Code written using the NumPy interface provided by “np”:
def step1(np, n):      return np.ones(n), np.ones(n)
def step2(np, x, y):  return np.dot(x, y)



# Unorthodox adoption strategy:
numpy_likes = [numpy, legate.numpy]
x,y = step1(random.choice(numpy_likes), 1_000_000_000)
print( step2(random.choice(numpy_likes), x, y) )
```

PROGRAM 1024 GPUS WITH NUMPY

```
def cg_solve(A, b, conv_iters):
    x = np.zeros_like(b)
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    converged = False
    max_iters =  b.shape[0]

    for i in range(max_iters):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)

        # Check convergence periodically
        if i%conv_iters == 0 and \
                  np.sqrt(rsnew) < 1e-10:
            converged = i
            break

        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew

    return x, converged
```

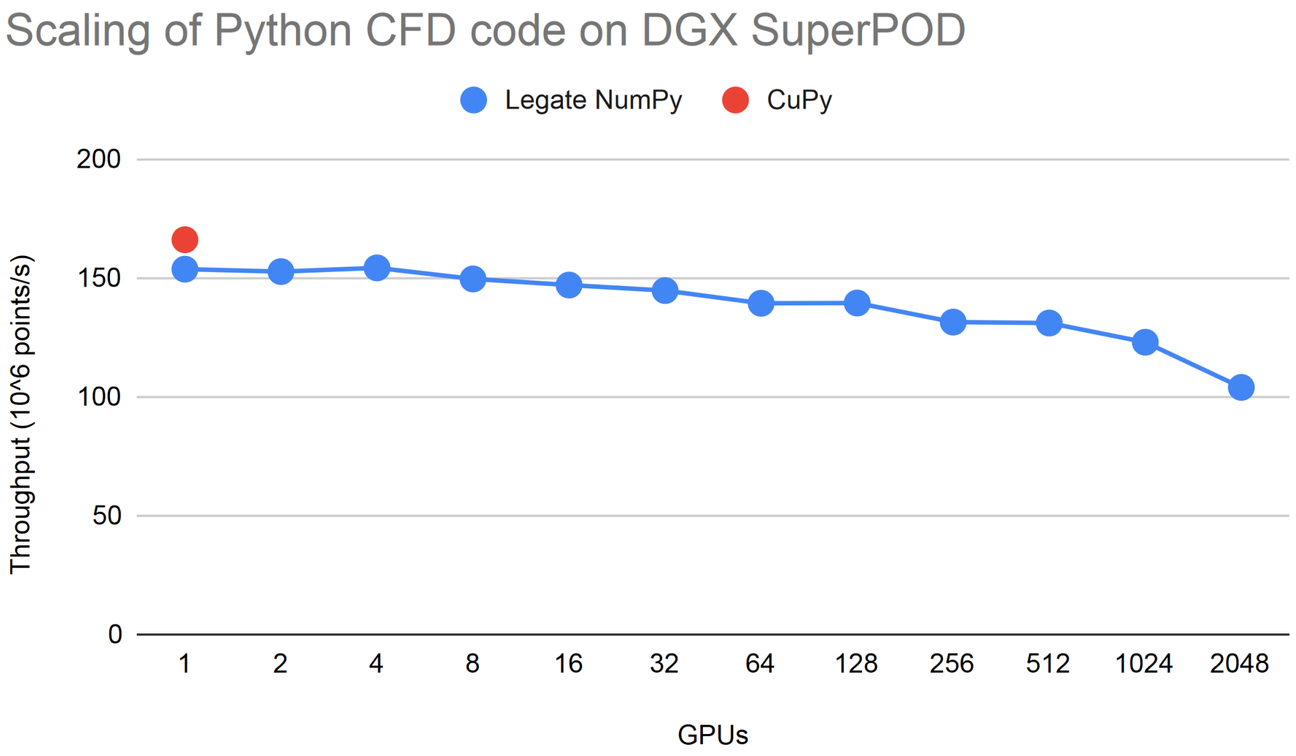
```
# Use Legate NumPy if available
try:    import legate.numpy as np
except: import numpy as np
```



FROM PEDAGOGY TO SUPERCOMPUTING

```python
import legate.numpy as np

for _ in range(iter):
    un = u.copy()

    vn = v.copy()
    b = build_up_b(rho, dt, dx, dy, u, v)
    p = pressure_poisson_periodic(b, nit, p, dx, dy)

    u[1:-1, 1:-1] = (
        un[1:-1, 1:-1]
        - un[1:-1, 1:-1] * dt / dx * (un[1:-1, 1:-1] - un[1:-1, 0:-2])
        - vn[1:-1, 1:-1] * dt / dy * (un[1:-1, 1:-1] - un[0:-2, 1:-1])
        - dt / (2 * rho * dx) * (p[1:-1, 2:] - p[1:-1, 0:-2])
        + nu
        * (
            dt
             / dx ** 2
            * (un[1:-1, 2:] - 2 * un[1:-1, 1:-1] + un[1:-1, 0:-2])
            + dt
             / dy ** 2
            * (un[2:, 1:-1] - 2 * un[1:-1, 1:-1] + un[0:-2, 1:-1])
        )
        + F * dt
    )
```

Running code excerpted from “CFD Python”
https://github.com/barbagroup/CFDPython



Extracted from “CFD Python” course at https://github.com/barbagroup/CFDPython
Barba, Lorena A., and Forsyth, Gilbert F. (2018). CFD Python: the 12 steps to Navier-Stokes equations. *Journal of Open Source Education*, **1**(9), 21, https://doi.org/10.21105/jose.00021

SUMMARY

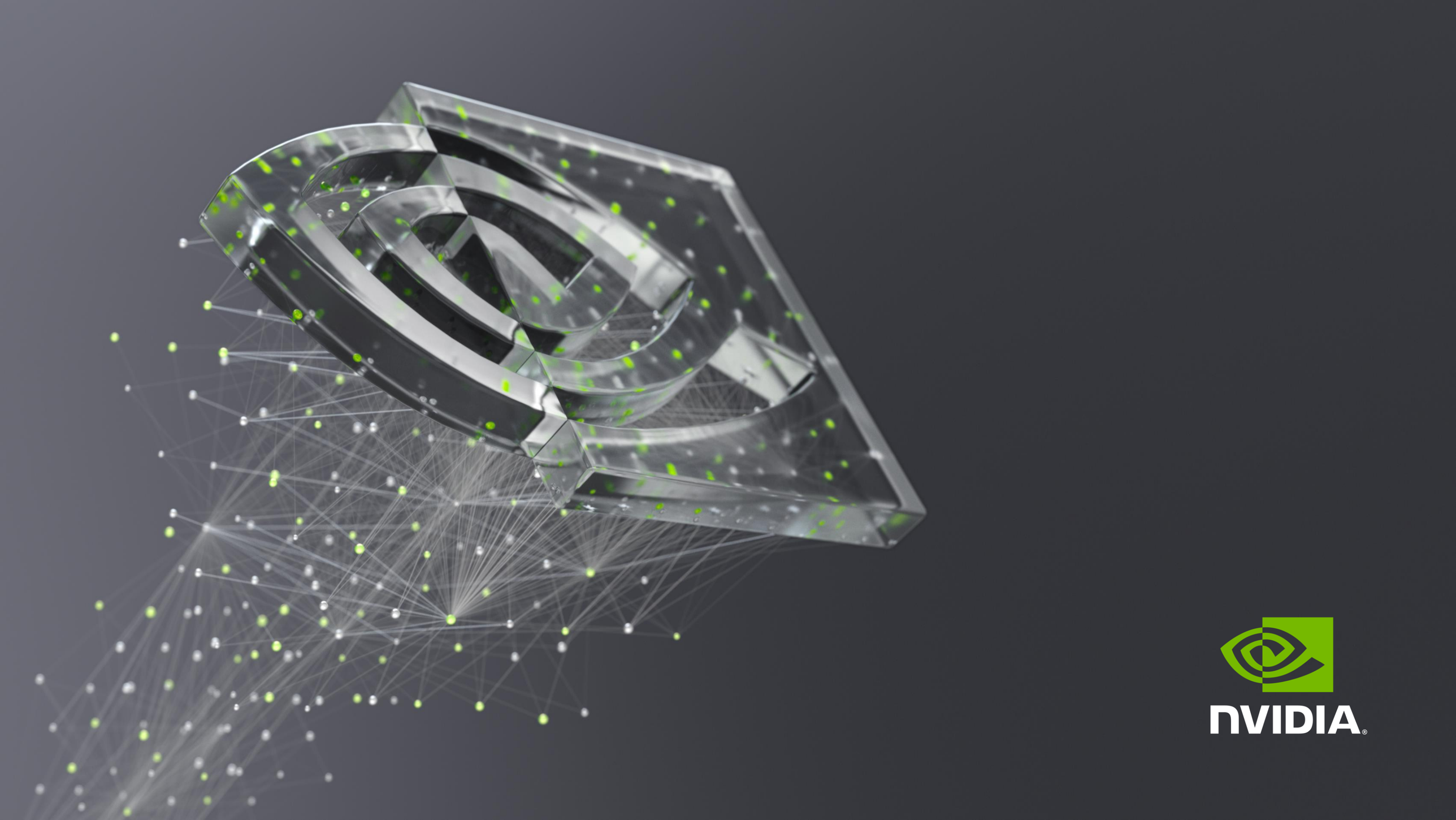
Supercomputing in Python with Legate

Envision an ecosystem of accelerated, distributed computing with convenient notation

Help users program supercomputers as if they were a single processor

Legion runtime system is the essential ingredient that makes this possible

Legate is open source. http://github.com/nv-legate


NVIDIA