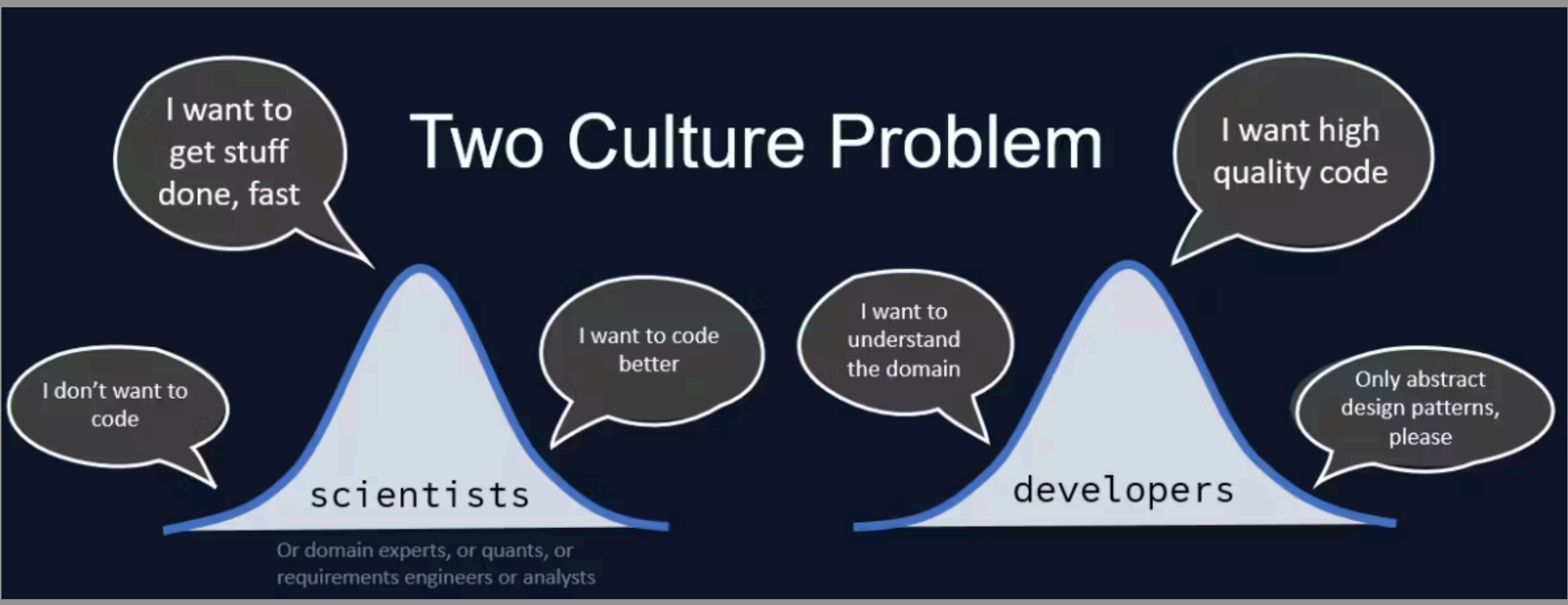
Bridging the Gap: Python Performance Tools for Scientists and Developers



The "Two Culture Problem" has always existed between scientists (domain experts), who prioritize quick results, and developers, who value clean, maintainable code. In the following slides, we’ll explore how Python can be optimized with tools and techniques to bridge this gap and serve both effectively.

Next Slide

This is the Python loop we want to optimize using different tools.

test-python.py

```python
def python_sum(n):
    total = 0
    for i in range(n):
        total += i
    return total

start = time.time()
result = python_sum(10**7)
print(f"Python result: {result}, Time: {time.time() - start}s")
```

Next Slide

Cython compiles Python to C for performance.

example.pyx

```
def cython_sum(int n):
    cdef long int i, total = 0
    for i in range(n):
        total += i
    return total
```

bash

```
cythonize -i example.pyx
```

test.py

```
from example import cython_sum
import time

start = time.time()
result = cython_sum(10**7)
print(f"Cython result: {result}, Time: {time.time() - start}s")
```

Next Slide

Numba

Numba is an LLVM code generator that integrates directly with Python

test-numba.py

```python
from numba import jit
import time

@jit(nopython=True)
def numba_sum(n):
    total = 0
    for i in range(n):
        total += i
    return total

start = time.time()
result = numba_sum(10**7)
print(f"Numba result: {result}, Time: {time.time() - start}s")
```

Next Slide

test-julia.jl

```julia
function julia_sum_optimized(n)
    total::Int64 = 0
    for i in 1:n-1
        total += i
    end
    return total
end

n = 10^7
@time julia_sum_optimized(n)
```

Next Slide

Results and Conclusion

```
Python result: 49999995000000, Time: 0.39797449111938477s

Cython result: 49999995000000, Time: 0.007426023483276367s

Numba  result: 49999995000000, Time: 0.0013298988342285156s

Julia  result: 49999995000000, Time: 0.008507013320922852s
```

The results demonstrate that the calculations are correct, and we can observe the time differences across the implementations. Python, as expected, has the slowest execution time at 397 ms. Cython follows with 7.4 ms, while Numba performs the best at 1.3 ms. Julia completes the same calculation in 8.5 ms. However, Julia can show significantly better performance in vectorized calculations, as highlighted in this link.