IWOCL and SYCLCon 2022

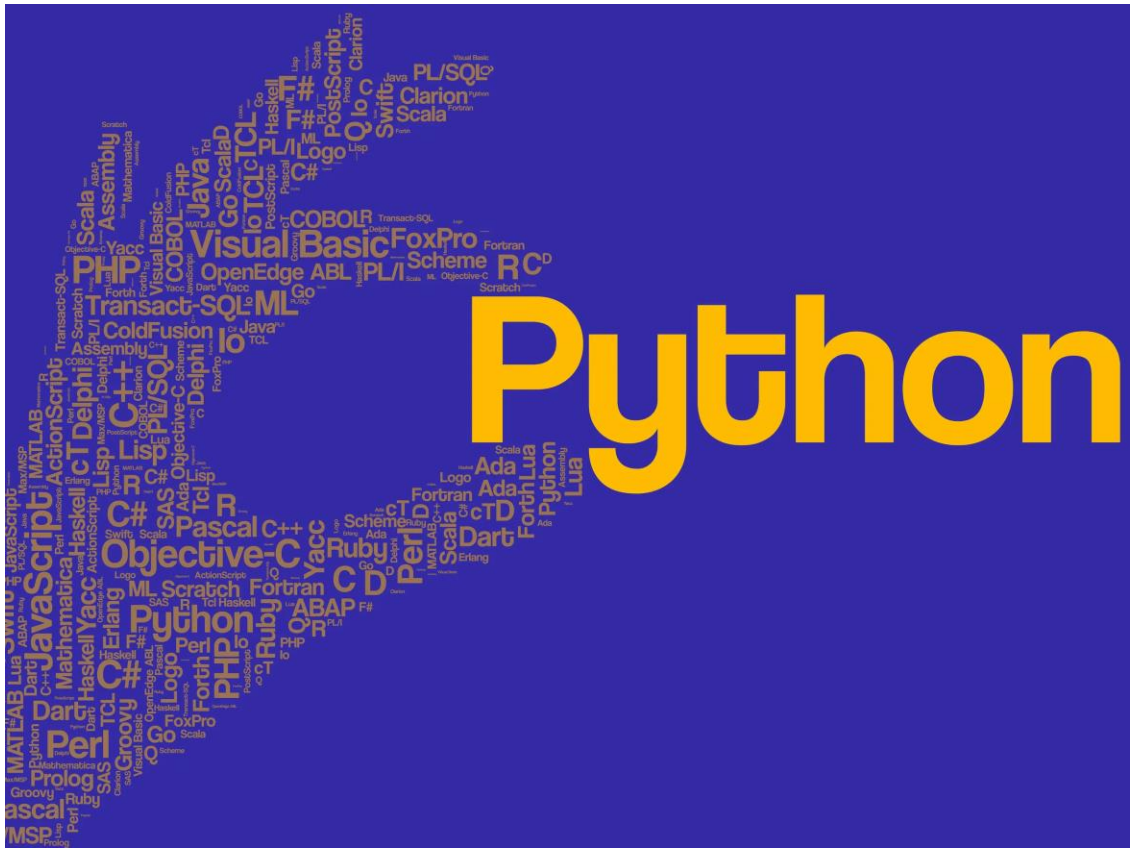
Interfacing Python and SYCL for XPU Programming

Diptorup Deb (presenter)

Oleksandr Pavlyk

intel

Why SYCL matters for Python developers…
… and vice versa



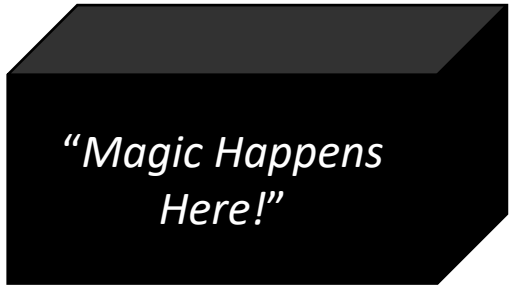
https://spectrum.ieee.org/top-programming-languages-2021

So basically, our goal is to …

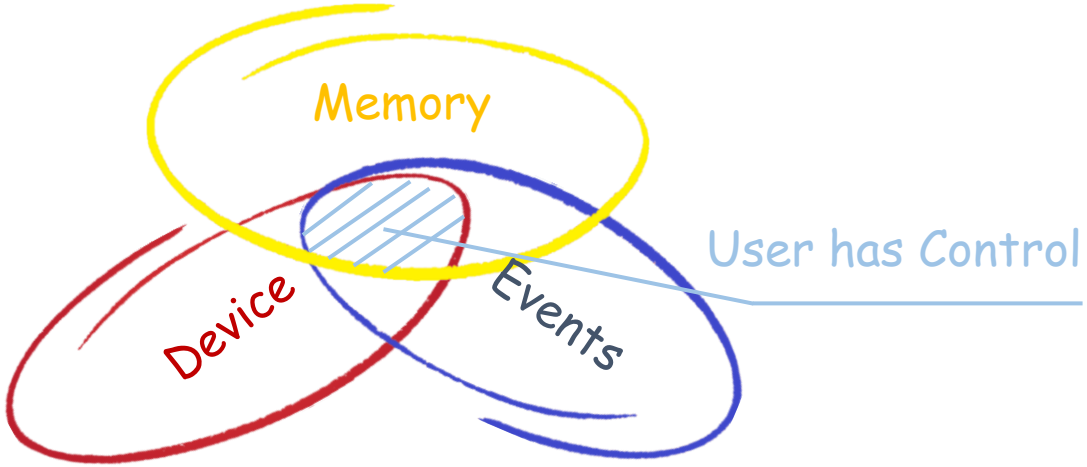
… empower the Python ninja to chop through the overgrown landscape of device programming!

Interfacing SYCL and Python

- Black Box – Call a SYCL library like any other native library using Python C API.



- Holistic – Allow SYCL device and queue management, memory management, synchronization directly from Python



Technical Design Goals

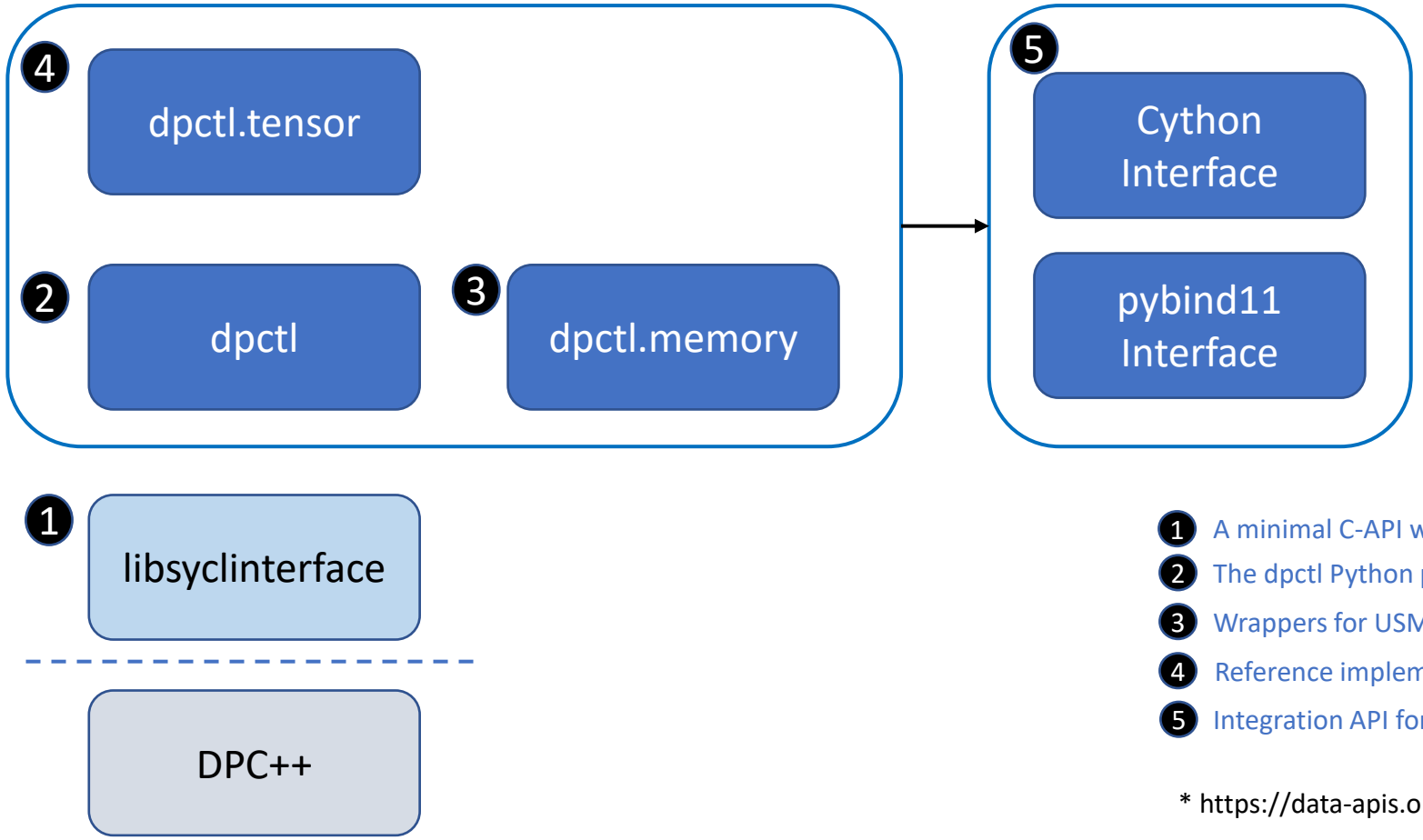
Bring the SYCL programming model to Python

- Dynamic device selection
- Sub-devices
- Async kernel execution
- Device memory management

Simplify using SYCL API-based programming in Python

- Infrastructure to build Python native extensions wrapping SYCL libraries
- Navigate the language boundary between SYCL and Python

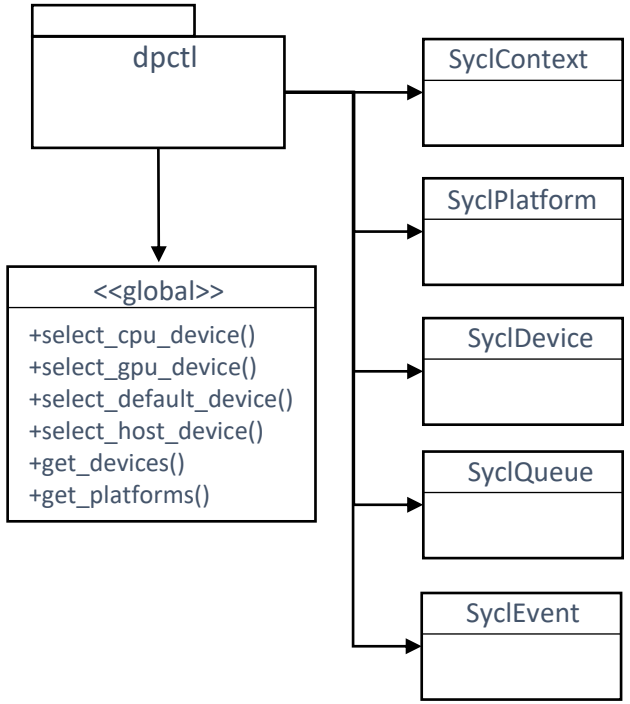
Data Parallel Control (dpctl)



1. A minimal C-API wrapper library of DPC++ SYCL RT
2. The dpctl Python package with wrappers for SYCL RT objects
3. Wrappers for USM allocations
4. Reference implementation of Python Array API* using SYCL
5. Integration API for Python native extension generators

* https://data-apis.org/array-api/latest/

dpctl: SYCL Bindings




```python
import dpctl

def select_device():
    """
    Programmatically try to select a CUDA GPU device
    or else select a CPU device.
    """
    D = None
    cuda_gpus = dpctl.get_devices(backend="cuda")
    if not cuda_gpus:
        D = select_cpu_device()
    else:
        D = cuda_gpus[0]

    return D
```

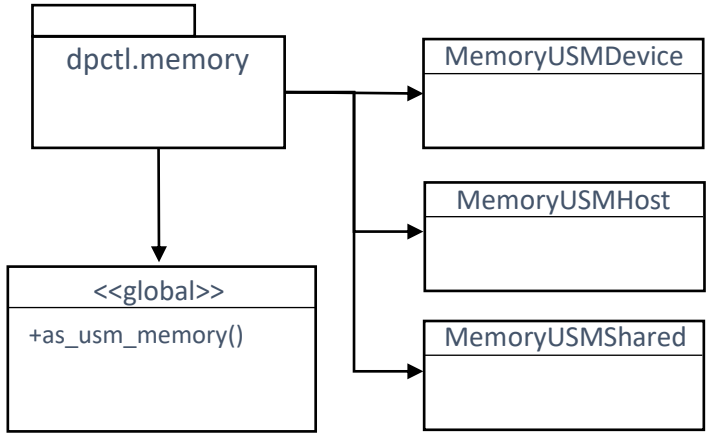
Custom Device Selection

Support for SYCL's Filter Selector Extension

Create Sub-devices

```python
import dpctl;

def subdivide_by_affinity(affinity="numa"):
    """
    Create sub-devices partitioning by affinity. NUMA affinity
    is selected by default.
    """
    cpu_d = dpctl.SyclDevice("cpu")
    try:
        sub_devs = cpu_d.create_sub_devices(partition=affinity)
        print("Partitioned device using " , affinity)
        print(
            "{0} sub-devices were created with respective "
            "#EUs being {1}".format(
                len(sub_devs),
                [d.max_compute_units for d in sub_devs]
            )
        )
    except Exception:
        print("Partitioning device by affinity not possible.")
```

dpctl.memory: USM Bindings

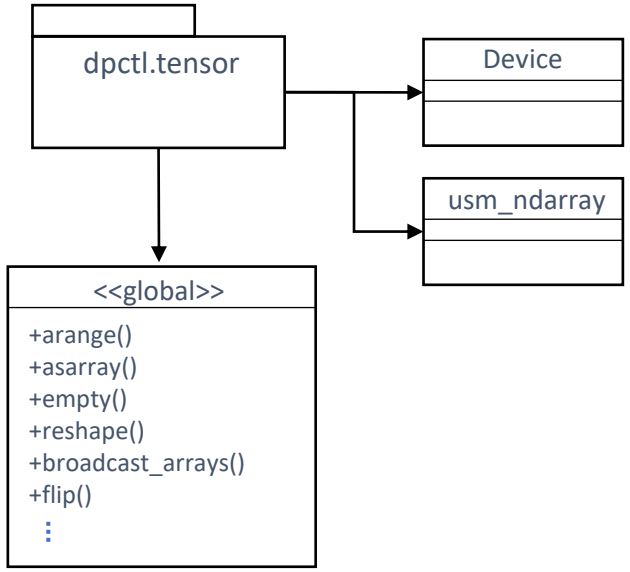



```
import dpctl
import dpctl.memory as dpmem

q = dpctl.SyclQueue()
m_dev = dpmem.MemoryUSMDevice(256, alignment=32, queue=q)

py_ob = bytes(b'abcd' * (m_dev.nbytes // 4))
m_dev.copy_from_host(pyobj)

m_host = dpmem.MemoryUSMHost(m_dev.nbytes)
m_host.copy_from_device(m_dev)

bytearray(m_host)
```

Full control over allocation

Support for Python buffers for host-accessible USM types

dpctl.tensor: Array API



https://data-apis.org/array-api/latest

- Reference implementation of Python Array API standard using SYCL
  - A subset of NumPy-like API (~200 functions)
  - Built on top of dpctl, dpctl.memory
  - Under heavy development

```
import dpctl.tensor as dpt

X = dpt.ones((5, 3), dtype='d', usm_type='device',
    device='opencl:gpu')

# execute on the device where data was allocated
Y = 3 * X

# modify subset of memory
Y[1:4] = dpt.fill(3, 2.7, device=X.device)
```

Extension Interfaces


```cpp
#include "dpctl4pybind11.hpp"
#include <CL/sycl.hpp>
#include <oneapi/mkl.hpp>
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>

void gemv(sycl::queue q,
                    dpt::usm_ndarray m,
                    dpt::usm_ndarray v,
                    dpt::usm_ndarray r,
                    const std::vector<sycl::event> &deps = {})
{
    auto n = m.get_shape(0);
    auto m = m.get_shape(1);
    int mat_typenum = m.get_typenum();
    /* various legality checks omitted */
    sycl::event res_ev;
    if (mat_typenum == UAR_DOUBLE) {
        auto *mat_ptr = m.get_data<double>();
        auto *v_ptr = v.get_data<double>();
        auto *r_ptr = r.get_data<double>();
        res_ev = oneapi::mkl::blas::row_major::gemv(
            q, oneapi::mkl::transpose::nontrans, n, m, 1,
            mat_ptr, m, v_ptr, 1, 0, r_ptr, 1, depends);
    }
    else
        throw std::runtime_error("unsupported");
    // submit the host task keeping arguments alive
    ht_ev = keep_args_alive(q, {m, v, r}, {res_ev});

    // return the pair of host task event and gemv event
    return std::make_pair(ht_ev, res_ev);
}
PYBIND11_MODULE(_onemkl, m)
{
    // Import the dpctl extensions
    import_dpctl();
    m.def("gemv", &gemv, "oneMKL gemv wrapper");
}
```

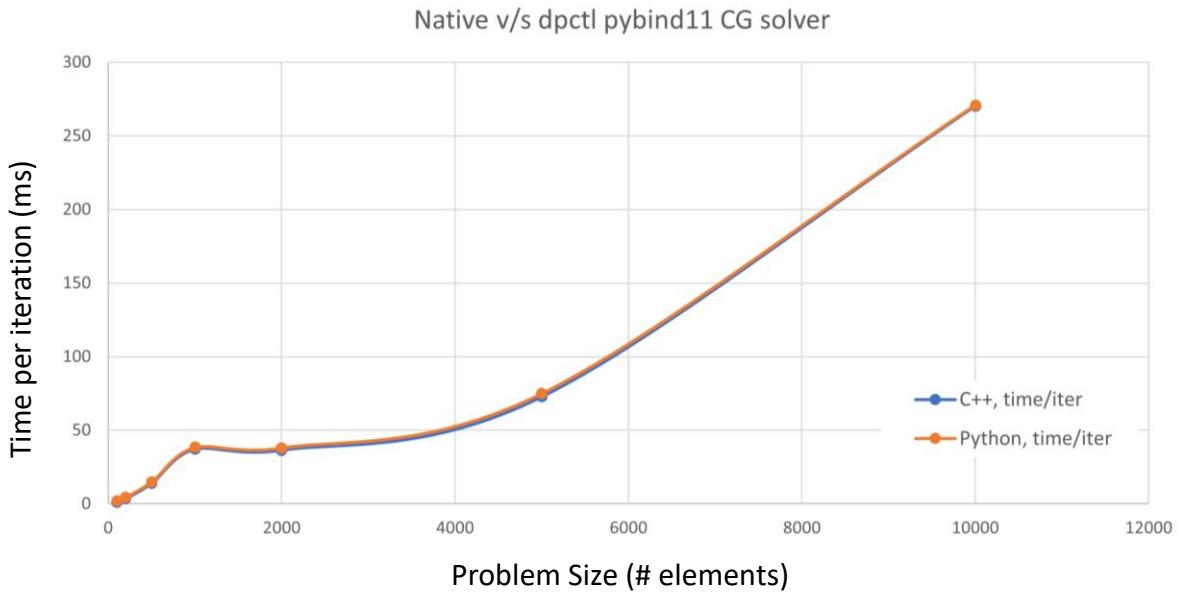
- **Create a Python ext. to call onemkl::gemv in < 40 loc (fits on a slide)**
- **Invoke it seamless from Python using dpctl, dpctl.tensor**

```python
import dpctl;
import numpy as np
import dpctl.tensor as dpt
import onemkl4py

q = dpctl.SyclQueue("level_zero:gpu")
# Allocate matrices and vectors objects using NumPy
Mnp, vnp = np.random.randn(5, 3), np.random.randn(3)

# Copy data to a USM allocation
M = dpt.asarray(Mnp, sycl_queue=q)
v = dpt.asarray(vnp, sycl_queue=q)
r = dpt.empty((5,), dtype="d", sycl_queue=q)

# Invoke a binding for the oneMKL gemv kernel.
he, ce = onemkl4py.gemv_nonblocking(M.sycl_queue, M, v, r)
# ... other computation may be overlapped with kernel execution

# synchronize to finilize the script
q.wait()
```


A Full Example

oneMKL gemv bindings and solvers implementations using gemv

- https://github.com/IntelPython/dpctl/tree/master/examples/pybind11/onemkl_gemv




```python
def cg_solve(A, b):
    """
    Conjugate gradient solver for A @ x == b.
    """
    exec_queue = A.sycl_queue
    x = dpt.zeros(b.shape, dtype=b.dtype)
    Ap = empty_like(x)
    all_host_tasks = []
    r = dpt.copy(b)
    p = dpt.copy(b)
    rsold = sycl_gemm.norm_squared_blocking(exec_queue, r)
    if rsold < 1e-20:
        return (b, 0)
    converged = False
    max_iters = b.shape[0]
    e_p = dpctl.SyclEvent()
    e_x = dpctl.SyclEvent()
    for i in range(max_iters):
        # Ap = A @ p
        he_dot, e_dot = sycl_gemm.gemv(exec_queue, A, p, Ap, depends=[e_p])
        all_host_tasks.append(he_dot)
        # alpha = rsold / dot(p, Ap)
        alpha = rsold / sycl_gemm.dot_blocking(
            exec_queue, p, Ap, depends=[e_dot]
        )
        # x = x + alpha * p
        he1_x_update, e1_x_update = sycl_gemm.axpby_inplace(
            exec_queue, alpha, p, 1, x, depends=[e_p, e_x]
        )
        all_host_tasks.append(he1_x_update)
        e_x = e1_x_update
        # r = r - alpha * Ap
        he2_r_update, e2_r_update = sycl_gemm.axpby_inplace(
            exec_queue, -alpha, Ap, 1, r, depends=[e_p]
        )
        all_host_tasks.append(he2_r_update)
        # rsnew = dot(r, r)
        rsnew = sycl_gemm.norm_squared_blocking(
            exec_queue, r, depends=[e2_r_update]
        )
        if rsnew < 1e-20:
            e1_x_update.wait()
            converged = i
            break
        beta = rsnew / rsold
        # p = r + beta * p
        he3_p_update, e3_p_update = sycl_gemm.axpby_inplace(
            exec_queue, 1, r, beta, p, depends=[e2_r_update]
        )
        rsold = rsnew
        all_host_tasks.append(he3_p_update)
        e_p = e3_p_update
        e_x = e1_x_update
    dpctl.SyclEvent.wait_for(all_host_tasks)
    return x, converged
```

Current Limitations

- DPC++ only: Supporting other SYCL compilers is a TBD for future
  - Depends on DPC++ extensions: *filter_selector, enqueue_barrier, default_context*

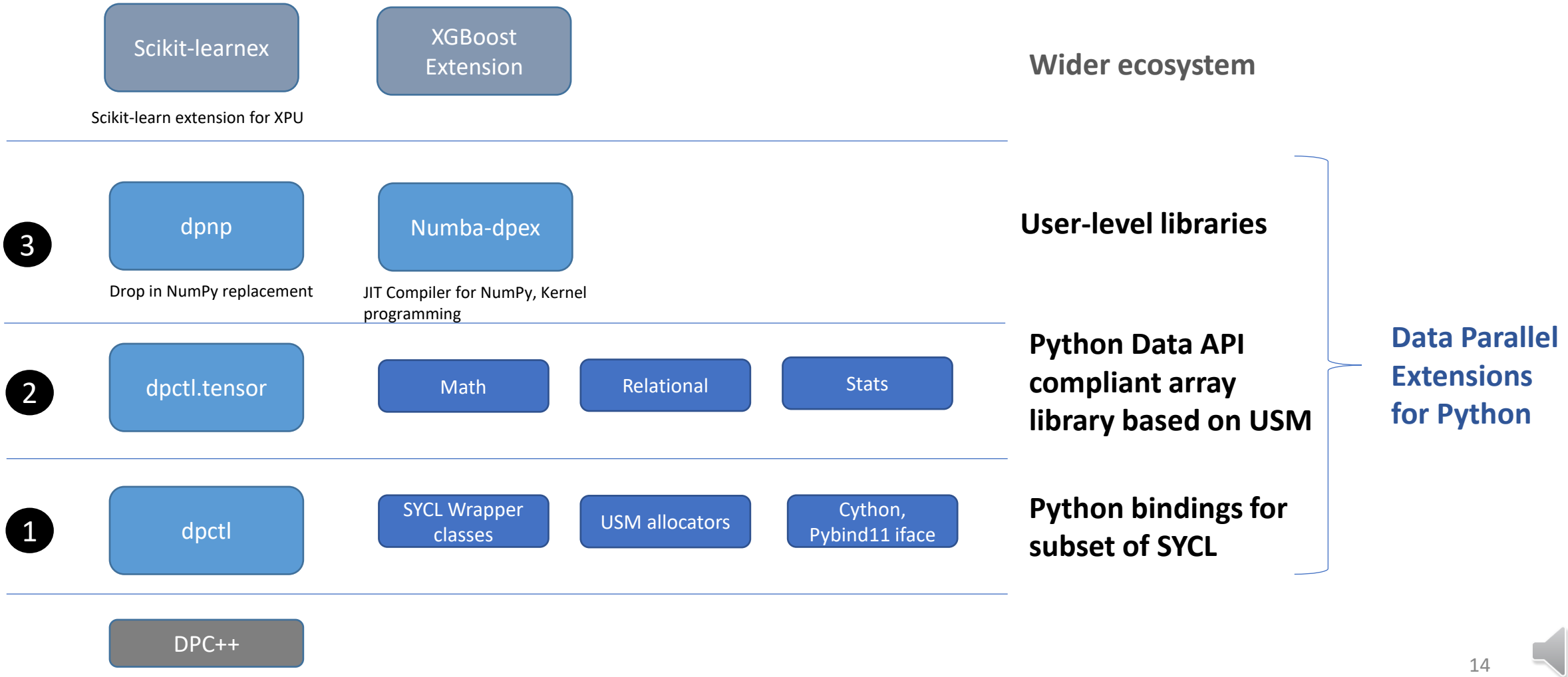
- Only USM: Buffers are not supported. Technical questions need to be sorted out first

Status and Ongoing Work

- Fully open source and currently under heavy development
  - https://github.com/IntelPython/dpctl (Source)
  - https://intelpython.github.io/dpctl/latest/index.html (Docs)
- Install from conda or pip

```
pip3 install dpctl
conda install dpctl -c intel
```

Wider Ecosystem



Thanks!