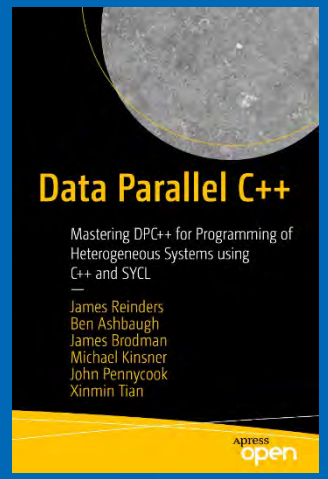
SYCL nuances not found in SYCL tutorials or our SYCL book

James R. Reinders
engineer
Intel



intel®

IXPUG
@ISC21

Warning:

FUTURE (under development)

Marketing calls it:

"Sneak Preview"

"Hot off the Presses"

Possibly insightful, definitely random, collection of things that…

- you won't find in our SYCL book

- you won't hear mentioned even in a full day SYCL tutorial
- insights into "what we were thinking" or "what we are puzzling about"

DISCLAIMER: This is NOT good presentation for a general audience.

Most developers never need to know these things. We are not 'most developers.'

Details under the hood are always a little messy.

Everyone has them – we just don't talk about them.

Possibly insightful, definitely random, collection of things that…

- you won't find in our SYCL book
- you won't hear mentioned even in a full day SYCL tutorial
- insights into "what we were thinking" or "what we are puzzling about"

DISCLAIMER: **This is NOT good presentation for a general audience.**

Most developers never need to know these things. We are not 'most developers.'

Details under the hood are always a little messy.

Everyone has them – we just don't talk about them.

Vocabulary level-setting…. What is…

- an XPU?
- our shared goals?
- SYCL?
- DPC++?
- an awesome book to read?

What is an XPU?

XPU ≈ ∗.∗ processing units

- *a name* for a diverse set of architectures
- SYCL calls them *devices*

- for example: CPU GPU FPGA DSP ASIC

Shared goal: make it so we can *really* program XPUs?

1. Freedom: Use any XPU that I choose.
   (regardless of XPU type or vendor)

2. Value: Regardless of my XPU choice, I consistently can obtain a reasonable level of performance.
   (regardless of XPU type or vendor)

3. Trust: My coding choices can be made with confidence, and my code is maintainable.

Note: some XPU-specific coding and tuning is expected and must be well supported!

SYCL…

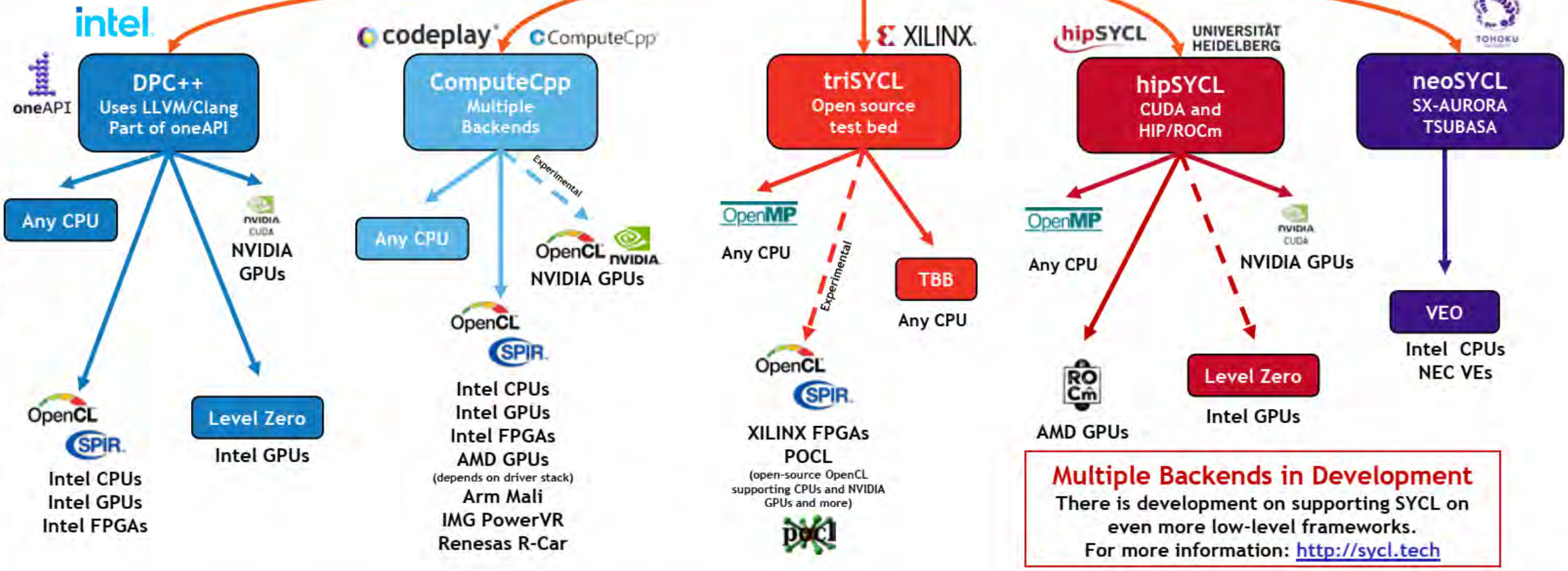
- extends C++ (with templates, a runtime, and libraries) for heterogeneous programming
- This gives us a portable way to query “what XPUs are present?” and a way to process work (code and data) on XPUs
- SYCL calls XPUs by the name “devices”
- SYCL fully support our *shared goals* (freedom, value, trust)

SYCL Implementations in Development

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL Source Code

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

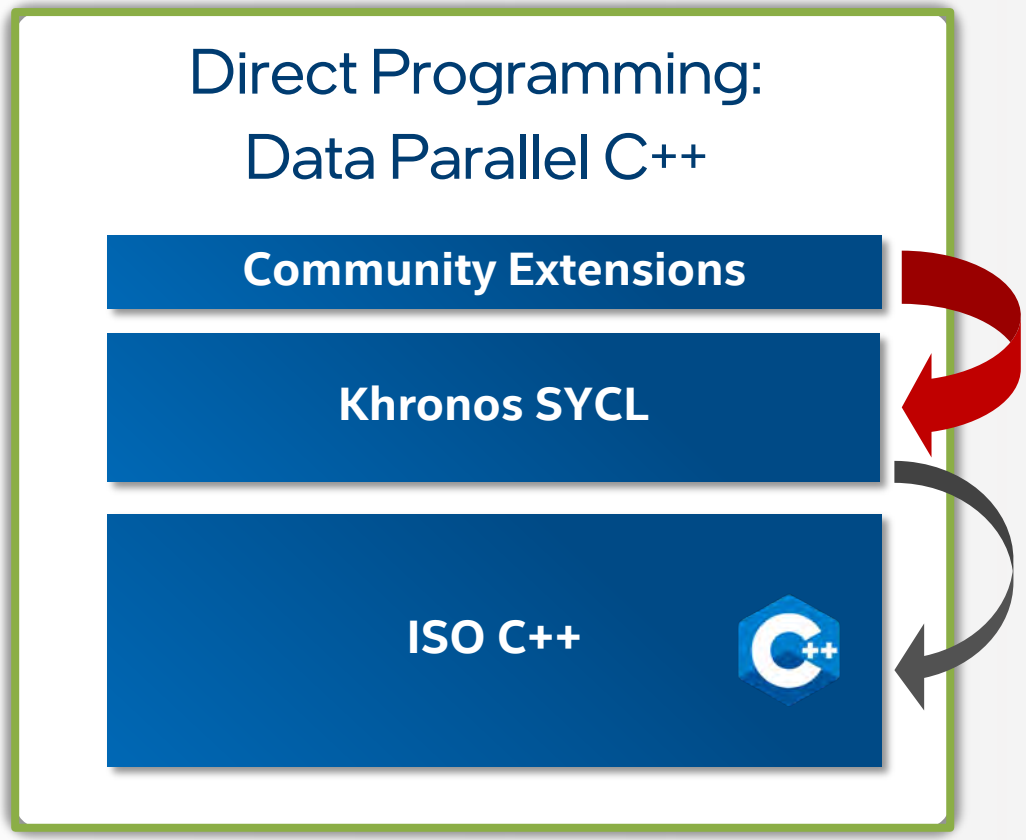


Multiple Backends in Development

There is development on supporting SYCL on even more low-level frameworks.
For more information: http://sycl.tech

https://www.iwocl.org/wp-content/uploads/k04-iwocl-syclcon-2021-wong-slides.pdf

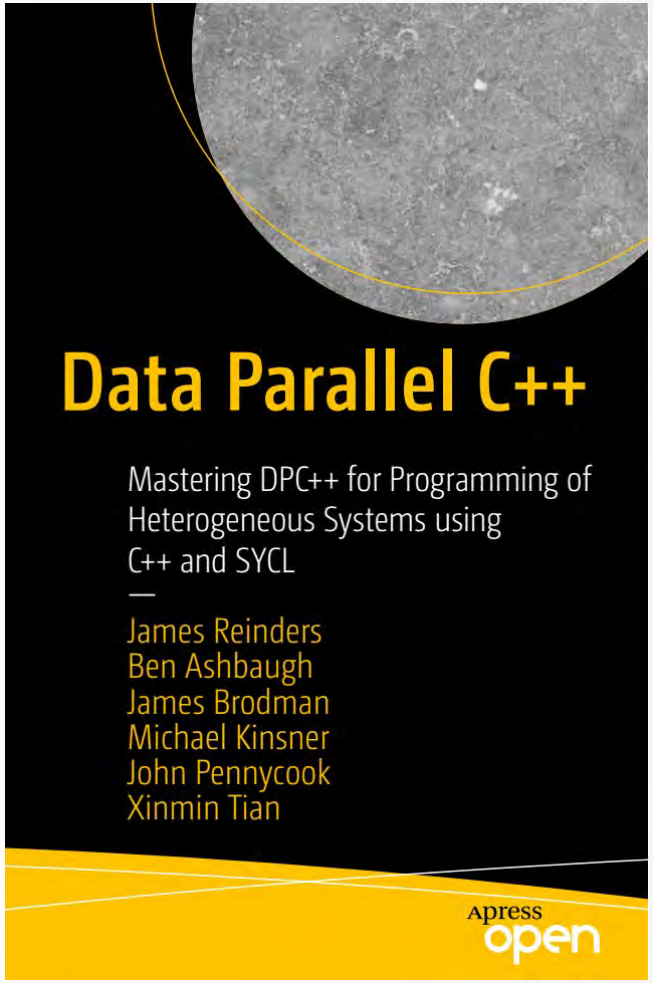
Path to Standardization



Most extensions feed into the SYCL specification
(Often with improvements from experience + generalization)

SYCL motivates proposals + participant positions
(Established best practice)

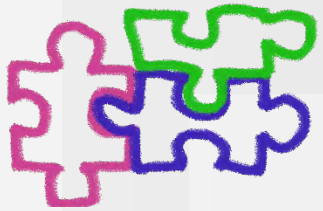
Our SYCL Book



- Free in PDF form! (paper version available for purchase)
  - www.apress.com/book/9781484255735
- First book on SYCL
  - aligned with SYCL 2020

Not taught in our SYCL (DPC++) book, or the ISC SYCL tutorials...

Lots of participation makes a better specification

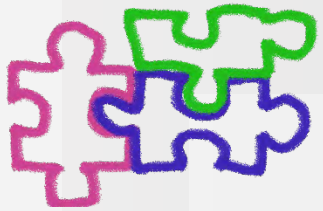


- SYCL 2020 is a huge upgrade to 1.2.1
  - Thanks to very broad industry contributions
  - Orientation very in tune with share goals for effective XPU usage
  - I expect that future SYCL updates will be smaller increments (think: C++11 *big* followed by smaller increments)
- No complete SYCL 2020 implementations exist today

  prototyping did precede the standard – in various compilers
- Future is bright
  - active development of public extensions (e.g., invokeSIMD)

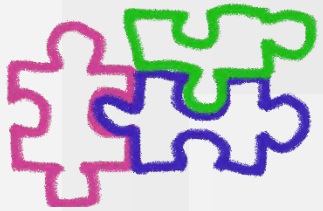

XPUG
@ISC21

SYCL is C++
how much C++ do you need to know?



- Four C++-isms to get used to:
  - **lambda functions** (not required, but preferred, so you'll see them a lot)
  - **use of templates**
    - thankfully – they usually look like function calls with varargs and optional parameters
    - I ❤ CTAD (class template argument deduction – reduces code verbosity)
  - **errno vs. throw/catch**
  - **importance of using scope to control lifetimes**
    - good C hygiene becomes important

SYCL is C++
how much C++ do you need to know?



- I don't think it's too scary – this is my "Hello, SYCL"...

queue submit

```
buffer<uint8_t, 3> frame_buffer(img.data, range<3>(img.rows, img.cols, 3));

q.submit([&](handler& cgh) {
  auto pixels = frame_buffer.get_access<access::mode::read_write>(cgh);
  cgh.parallel_for(range<3>(img.rows, img.cols, 3), [=](item<3> item) {
    uint8_t p = pixels[item];
    pixels[item] = sycl::clamp(p+50,0,255);
  });
});
q.wait_and_throw();
```

3D (R,G,B) using a lambda

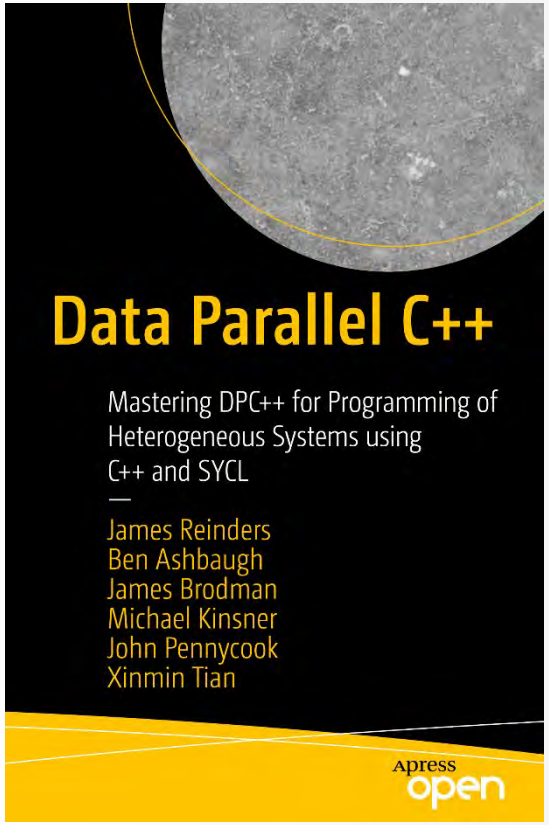
increment by 50
(with clamp to give saturation instead of wraparound)

make sure it is done

- from a blog I did – adds 50 to every R,G,B value in an image

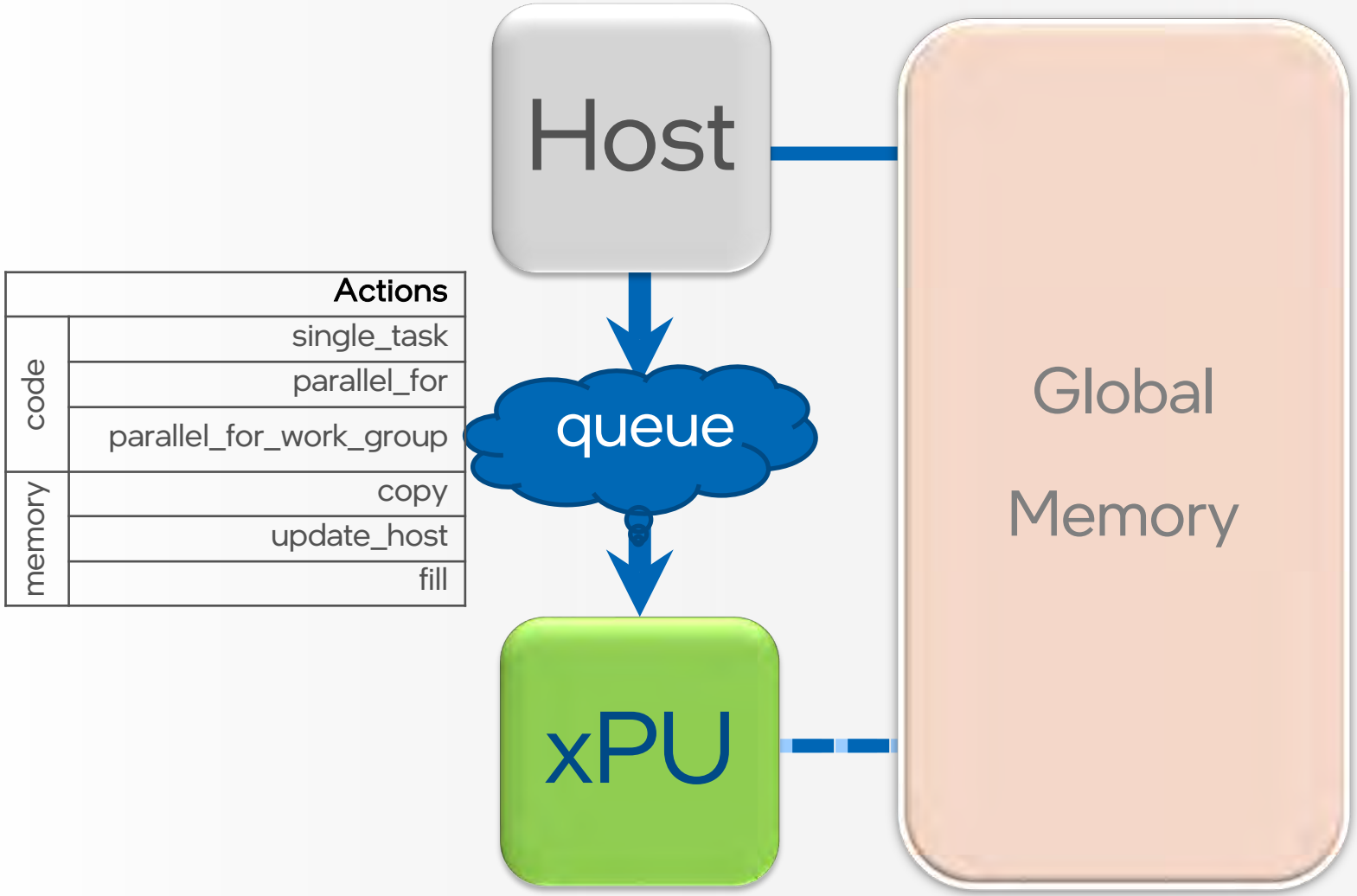

@ISC21
IXPUG

What is a SYCL queue?



Chapter 2

"Where Code Executes"


Host
Global
Memory
queue
xPU
Actions
code
single_task
parallel_for
parallel_for_work_group
memory
copy
update_host
fill

device selection

nonchalant

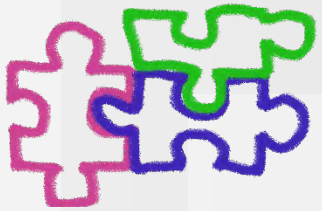
```
queue();
queue(default_selector_v);
```

selective

```
queue(cpu_selector_v);
queue(gpu_selector_v);
queue(accelerator_selector_v);
queue(INTEL::fpga_emulator_selector_v);
queue(INTEL::fpga_selector_v);
```

full unmitigated control freak

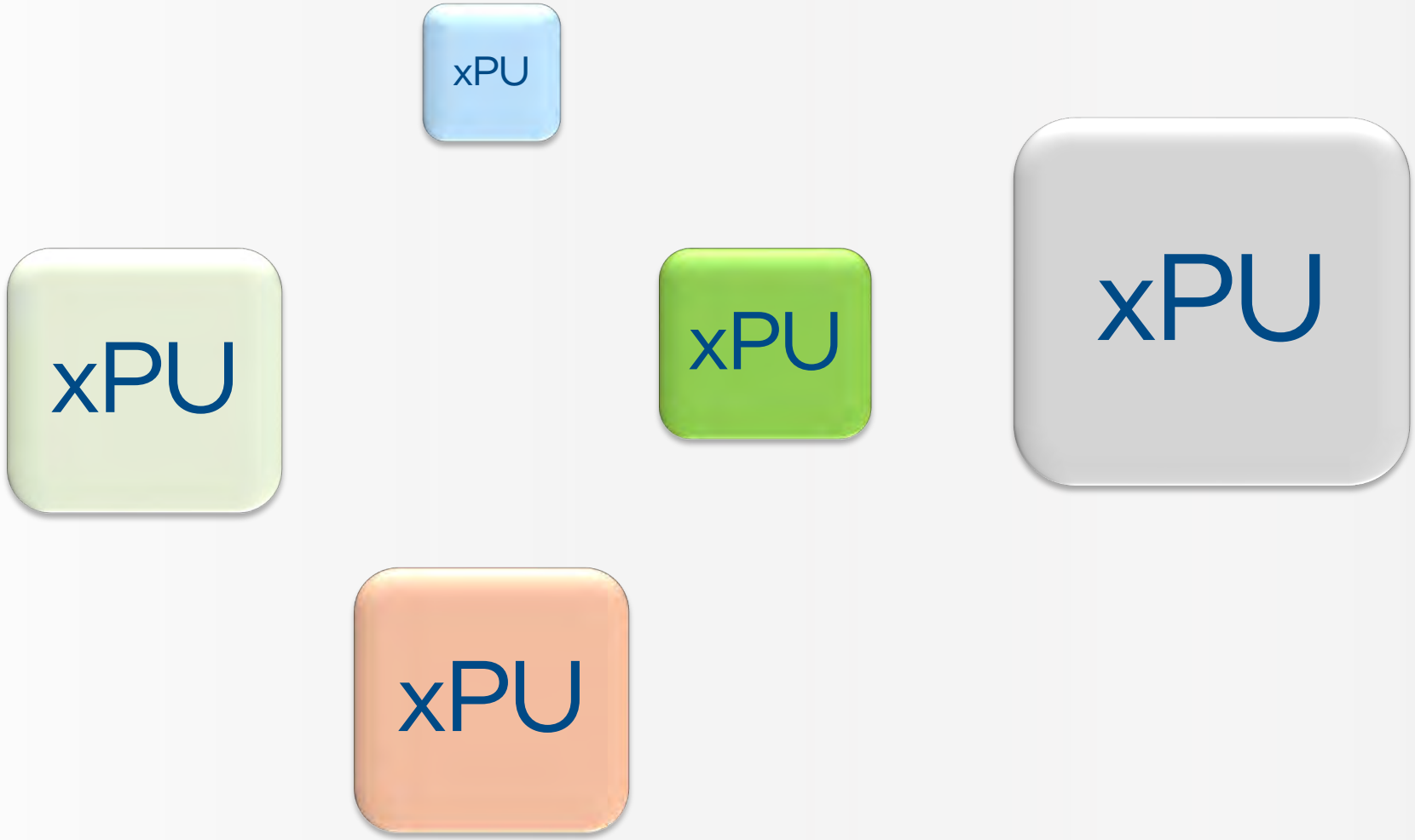
```
queue(my_custom_device_selector_v);
```



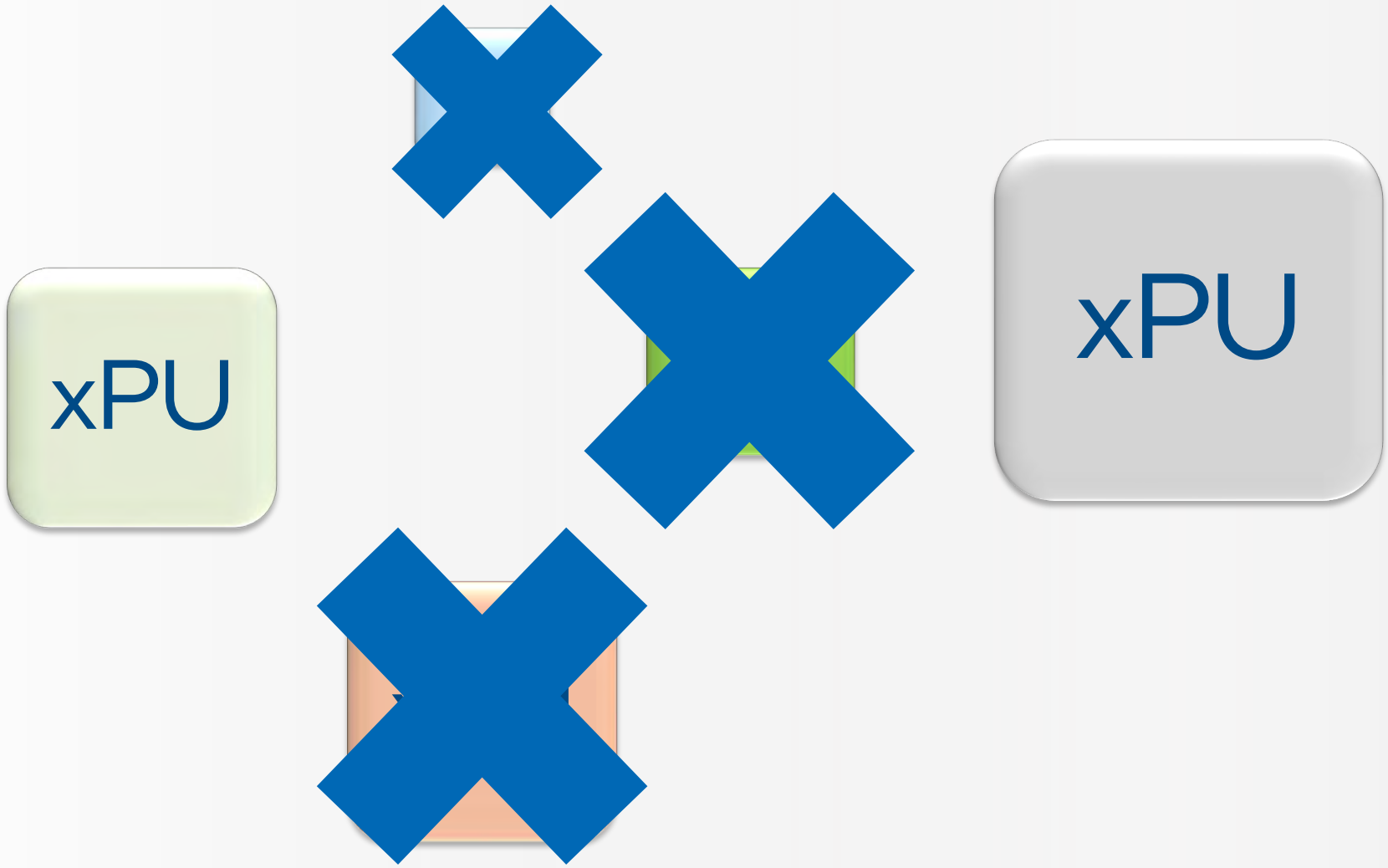
What did we *not tell you?*

The Rest of the Story

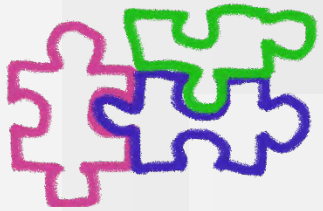
Our application may have many XPUs to select from…



Can we limit an application's choice before it even runs?



Yes… external forces at work: filters



(DPC++ specific example)

SYCL_DEVICE_FILTER

Can limit the SYCL runtime to
only a subset of possible devices.

Affects everything! Specifically:

- platform::get_devices()
- platform::get_platforms())
- all device selectors

https://tinyurl.com/syclfilters

for more on filters and *tracing* options!

Devices

- A device is a software abstraction of some hardware resource.

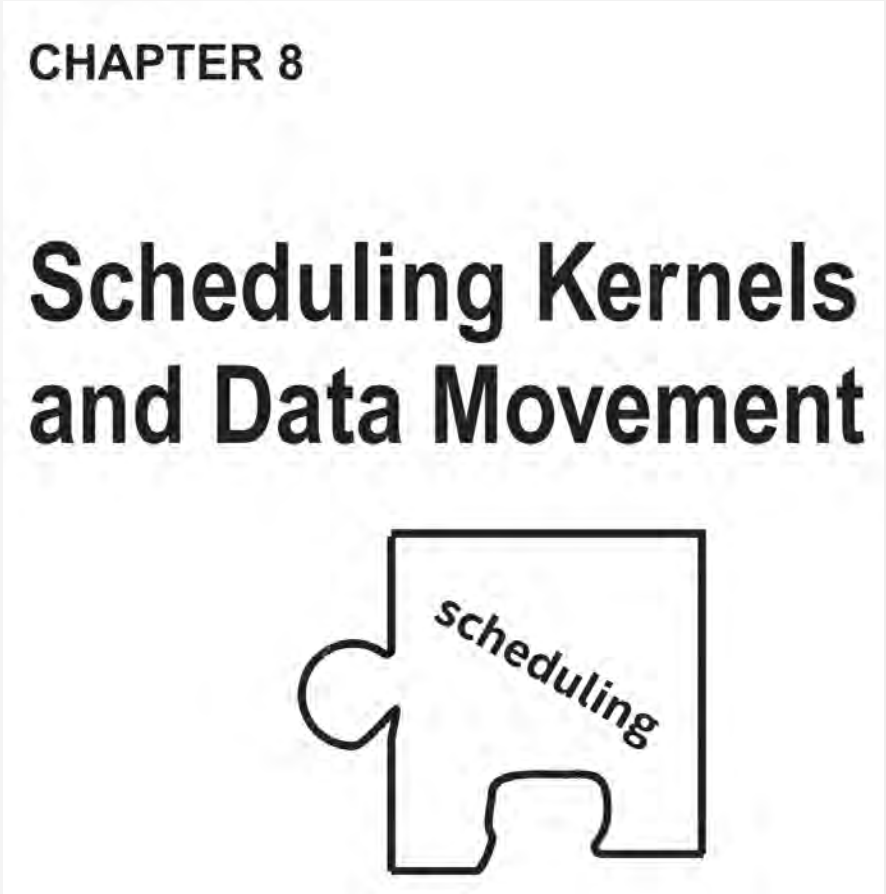
- The same hardware resource may appear in multiple platforms! (e.g., each GPU will be exposed via both OpenCL and Level 0)

```
> SYCL_DEVICE_FILTER=opencl,level_zero sycl-ls
CPU : Intel(R) OpenCL 2.1
GPU : Intel(R) OpenCL HD Graphics 3.0
GPU : Intel(R) Level-Zero 1.0
HOST: SYCL host platform 1.2
```

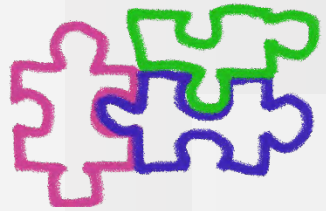

XPUG
@ISC21

What *type* of queue?

- `FIFO?`
- `Priority?`
- `Run?`



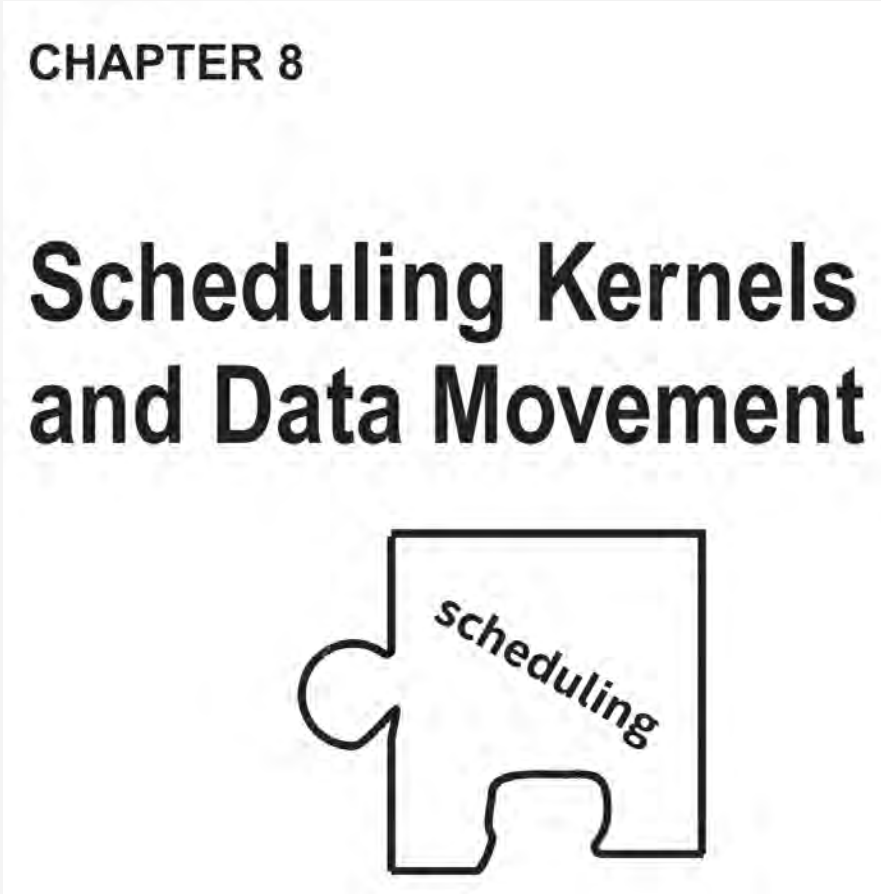
What *type* of queue?



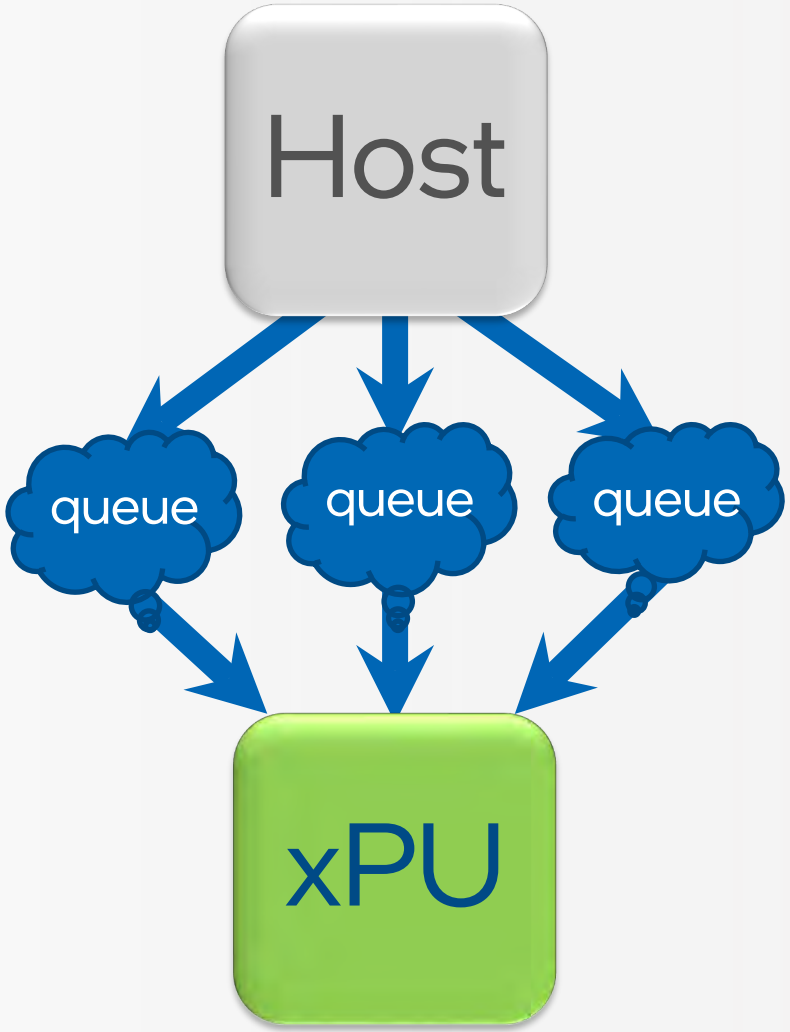
- `FIFO?`
- `Priority?`
- `Run?`

`Get FIFO with:`

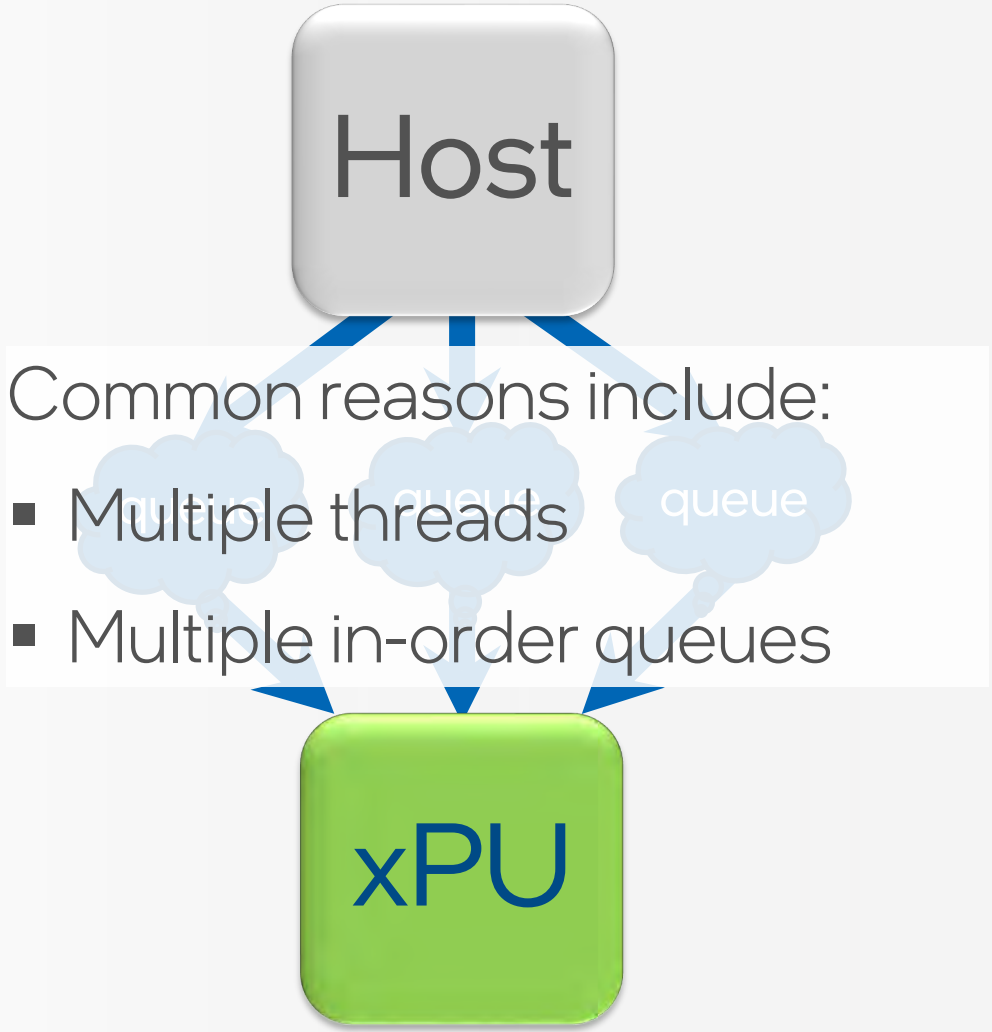
`property::queue::in_order()`



are multiple queues better than one?



are multiple queues better than one?

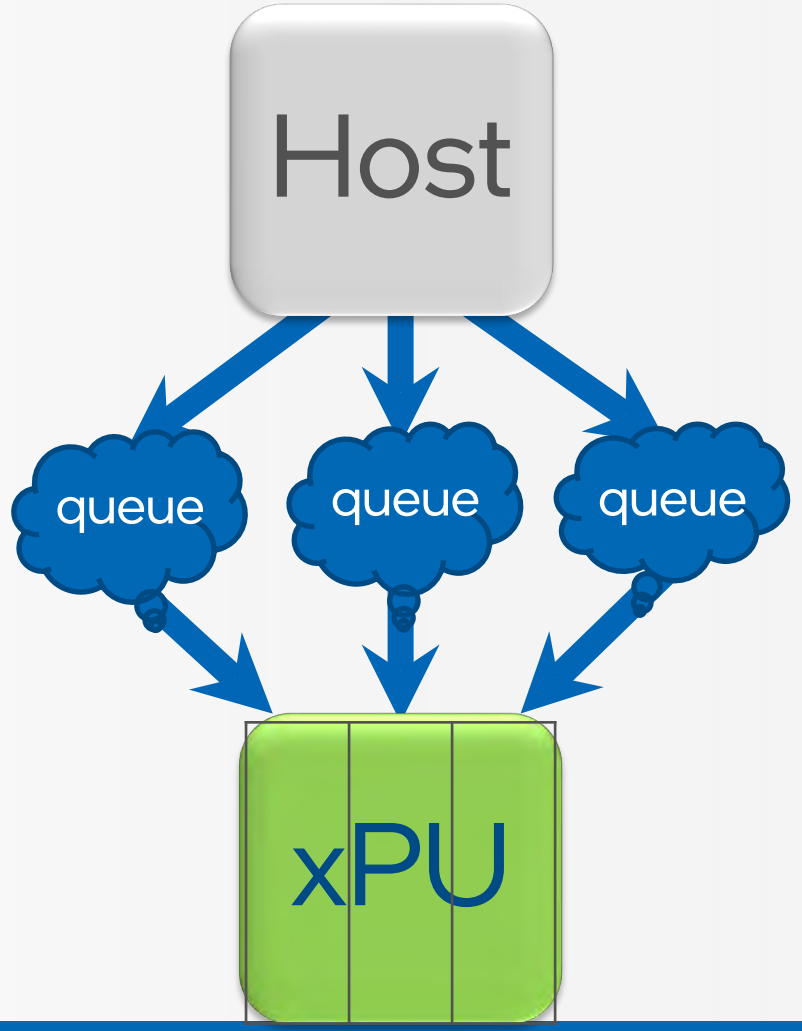


Common reasons include:

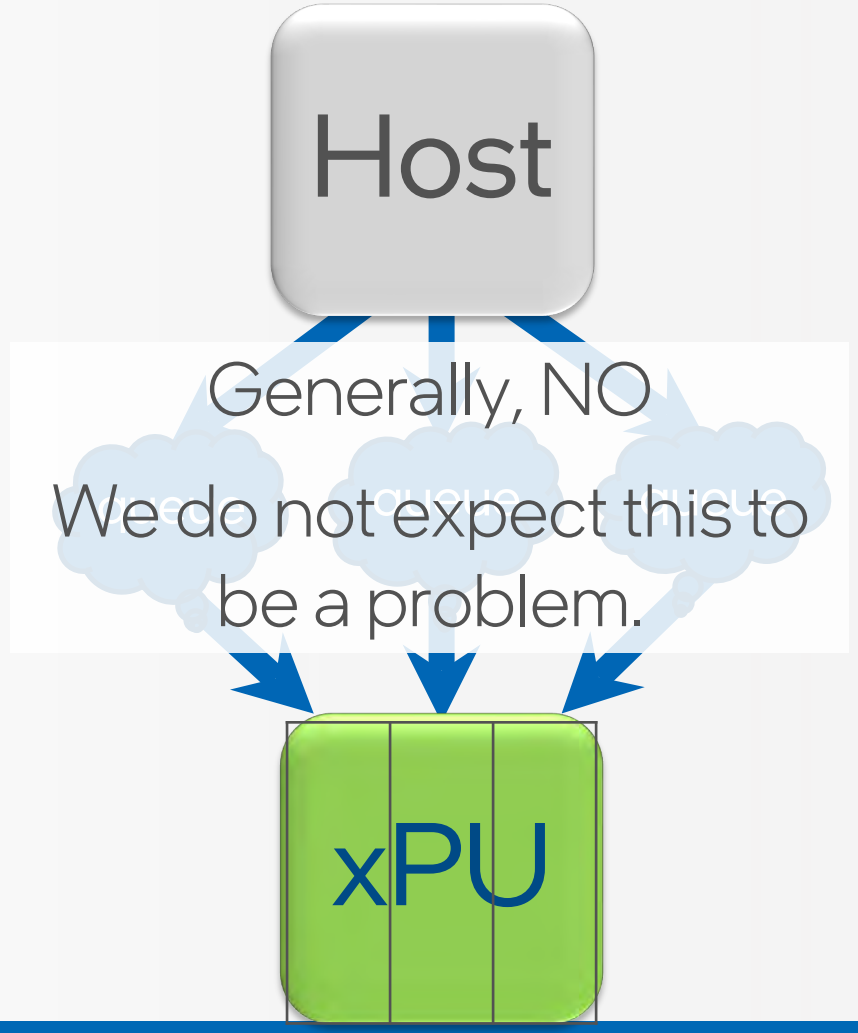
- Multiple threads
- Multiple in-order queues


XPUG
@ISC21

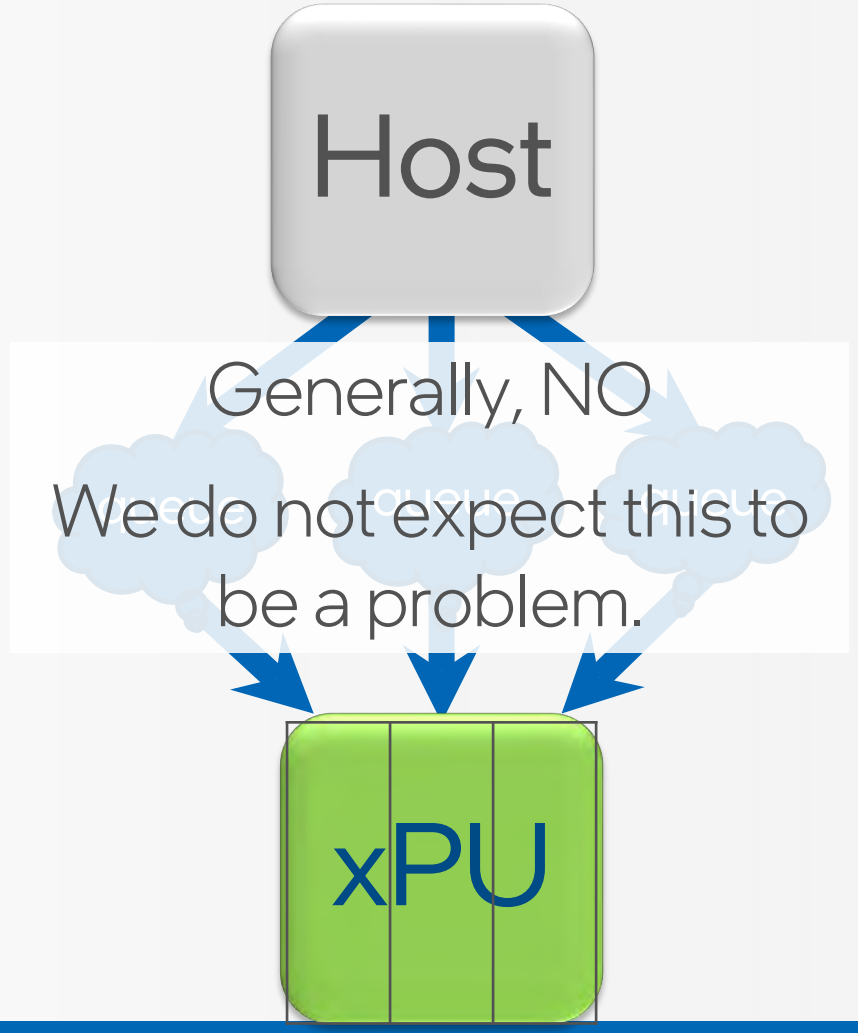
can multiple queues be worse than one?



can multiple queues be worse than one?



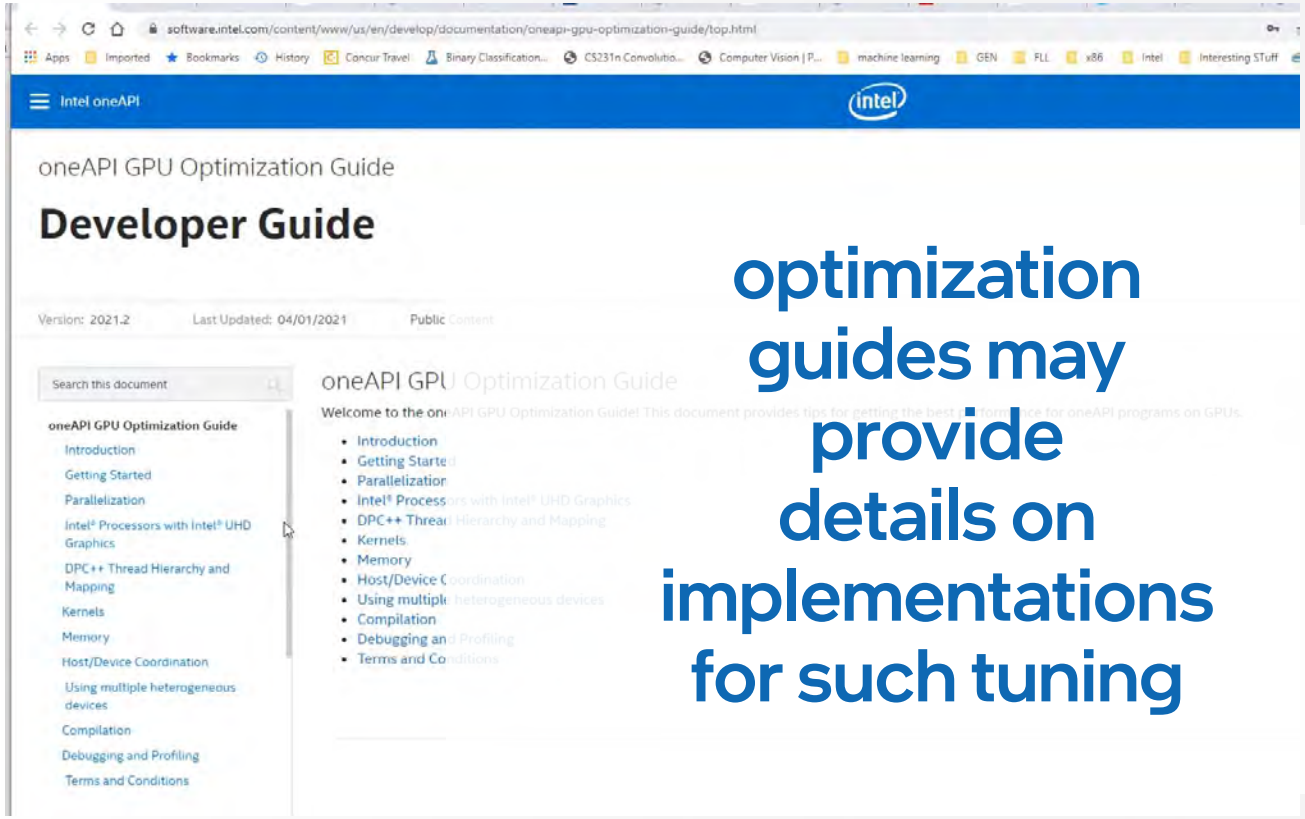
can multiple queues be worse than one?



neither of these should be highly visible:

- contexts are not cheap
- the path to a device may do batching

optimization guides



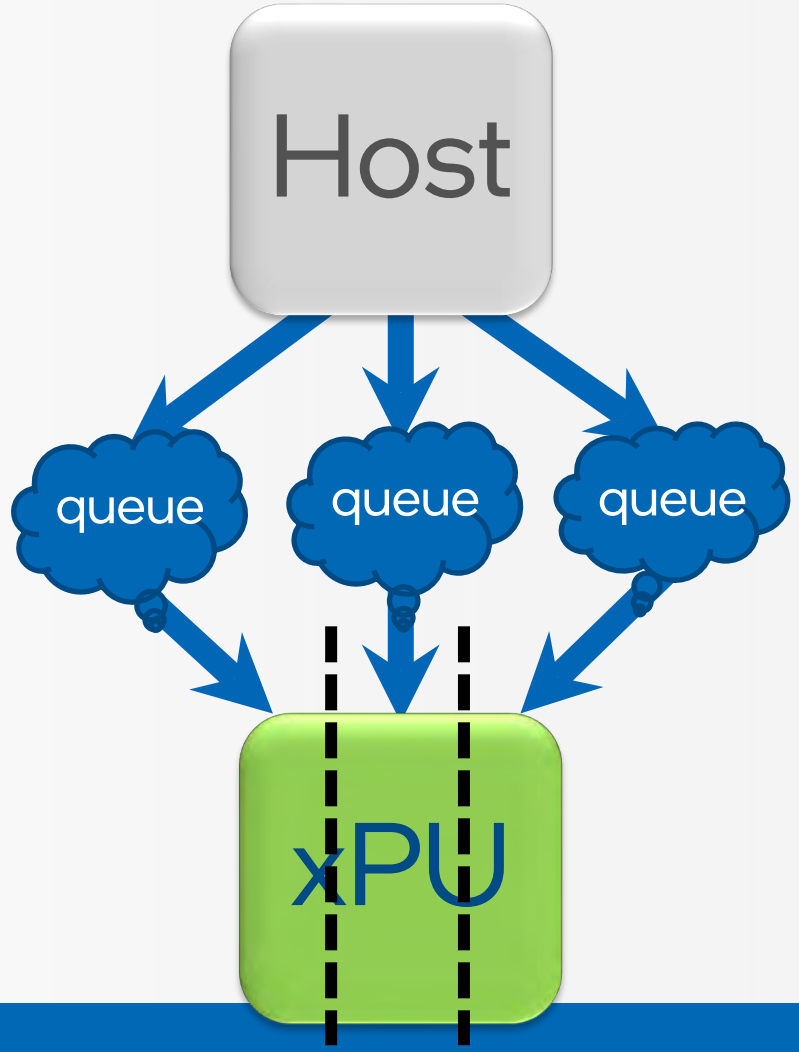
optimization guides may provide details on implementations for such tuning

Set the SYCL_PI_LEVEL_ZERO_BATCH_SIZE=8

https://tinyurl.com/SYCLgpuOPT


IXPUG
@ISC21

can multiple queues divide up a device?



Implicit vs Explicit Scaling

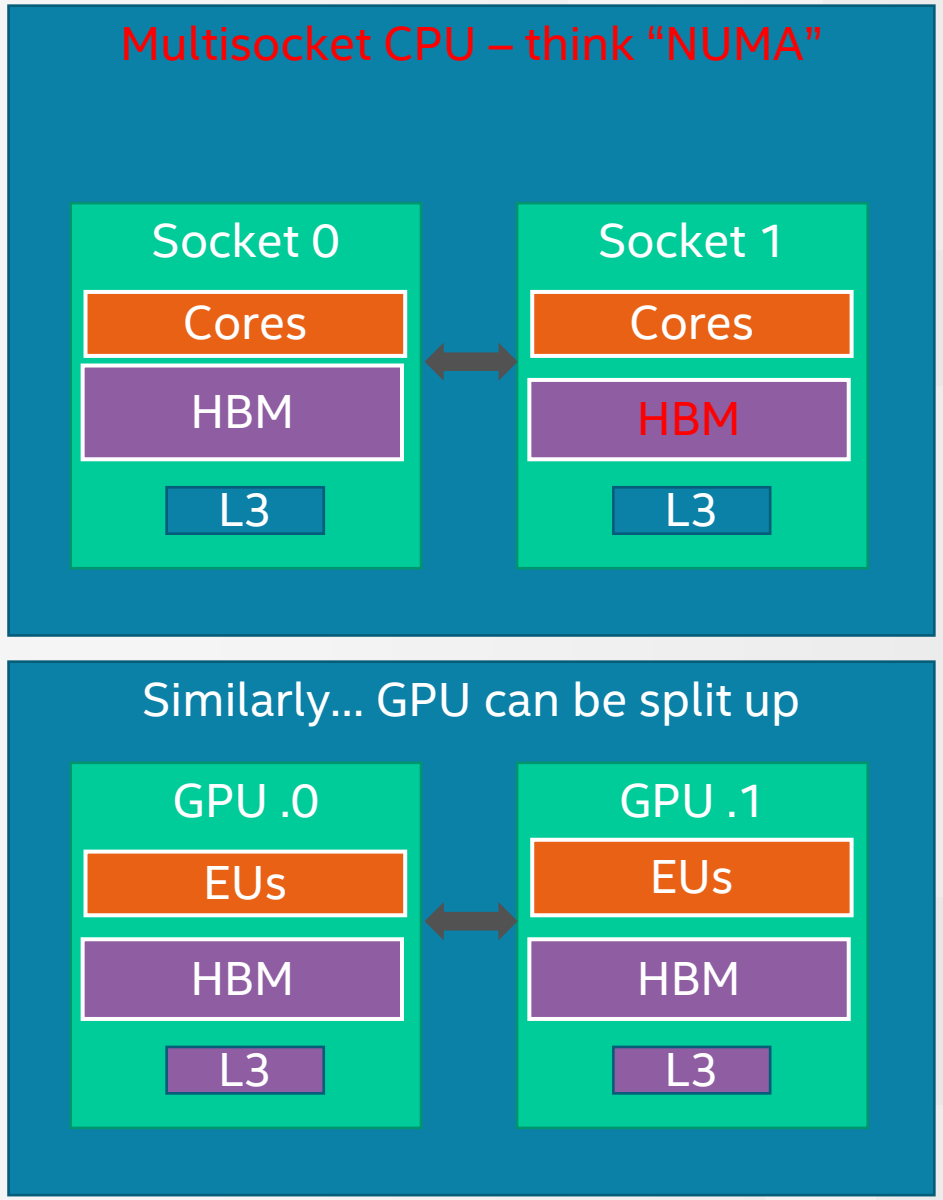
- **Implicit Scaling**: One device – throw everything at it!
  Driver automatically distributes work and allocations across underlying resources
- **Explicit Scaling**: Divide up the device – throw different pieces at each division!
  Application manually distributes work and allocations across underlying resources

- Same trade-off as for multisocket CPU systems:
  - Implicit requires attention to memory placement, work scheduling, etc
  - Explicit requires an "extra" level of decomposition

- If you'd normally launch one MPI rank per socket, use explicit scaling!

Sub-Devices

- A sub-device represents a collection of execution and memory resources.
- A sub-device is still a device.
  - It can do anything a device can do…
  - …including creating sub-(sub-) devices!
- Sub-devices are a very powerful abstraction for CPUs and GPUs.



Sub-devices created at NUMA boundary.

Environmental vs. Programmatic Controls

- **Environmental:**
  Environment modifies definition of "device" and default context.
  - e.g. OpenCL Intercept, `ZE_AFFINITY_MASK`, `LIBOMPTARGET_DEVICES`
  - No code changes required, but error-prone and non-portable

- **Programmatic:**
  Code explicitly sets up sub-devices.
  - e.g. `sycl::create_sub_devices()`,
    `#pragma omp target device(n) subdevice(0, m)`
  - Code changes required, but errors are detectable, and behavior is standardized

1 MPI Rank → Single Device (Programmatic)

```
// If there are multiple devices available, select one of them
auto devices = sycl::device::get_devices(sycl::info::device_type::gpu);
sycl::device root = devices[rank % devices.size()];

...

// Attempt to split the device along NUMA boundaries
sycl::device dev;
try {
  auto sub_devices = root.create_sub_devices
       <sycl::info::partition_property::partition_by_affinity_domain>
       (sycl::info::partition_affinity_domain::numa);
  dev = sub_devices[rank % sub_devices.size()];
} catch (sycl::exception e) {
  dev = root;
}

... // Allocating memory and enqueuing kernels works as before
```

unnecessarily verbose today

1 MPI Rank → Multiple Devices (Many Contexts)

```
// Each rank will offload to all devices that it can see
auto devices = sycl::device::get_devices(sycl::info::device_type::gpu);
...
// Create a queue associated with each device
std::vector<sycl::queue> queues;
for (auto& dev : devices) {
  queues.push_back(sycl::queue(dev));
}
...
// Allocate memory using the context associated with each device
std::vector<float*> as;
for (int d = 0; d < queues.size(); ++d) {
  sycl::queue q = queues[d];
  as.push_back(sycl::malloc_shared<float>(per_device, q.get_device(), q.get_context()));
}
...
// Execute a kernel on each device
for (int d = 0; d < queues.size(); ++d) {
  sycl::queue q = queues[d];
  float* a = as[d];  float* b = bs[d];  float* c = cs[d];
  q.parallel_for(per_device, [=](sycl::id<1> i) {
    c[i] = a[i] + b[i];
  });
}
```

Each device operates on a private allocation created against a per-device context.

1 MPI Rank → Multiple Sub-Devices (Single Context)

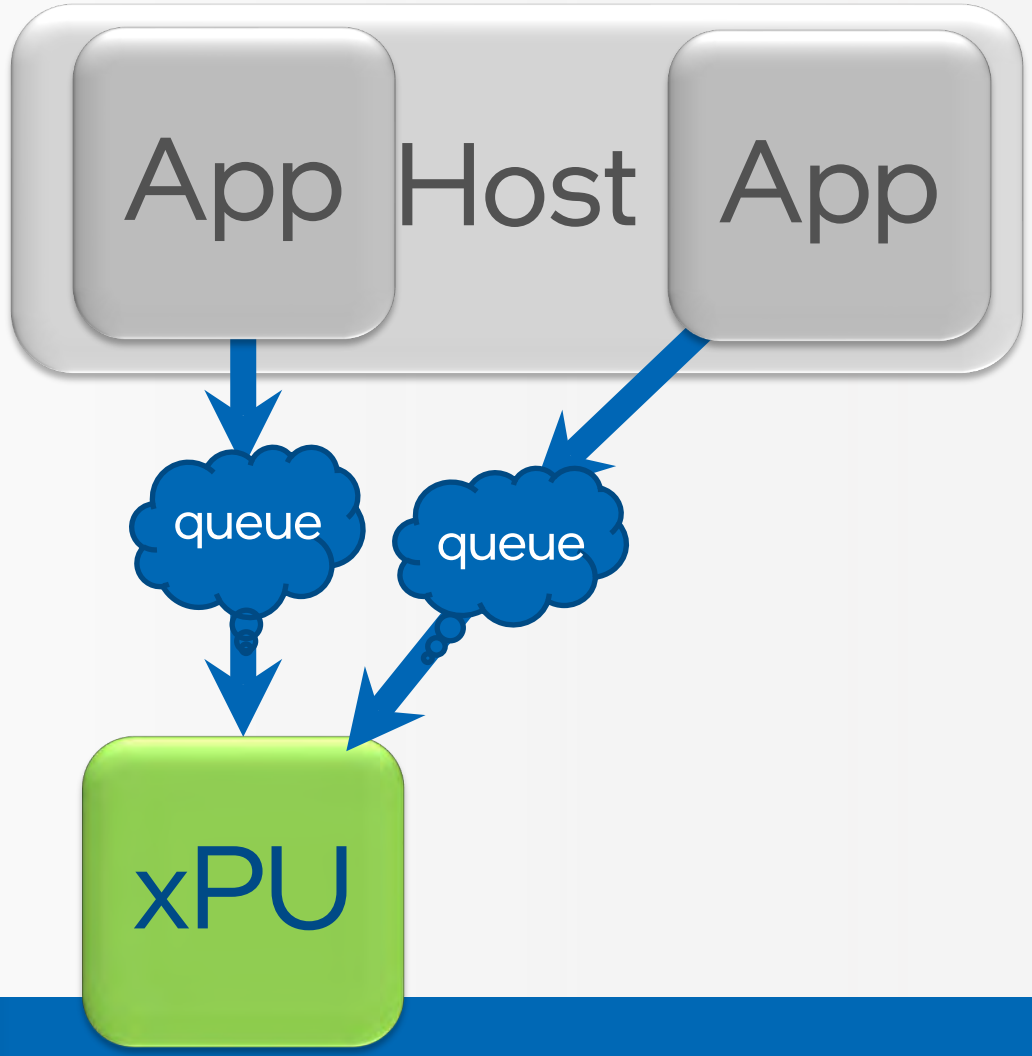
```
// If there are multiple devices available, select one of them
auto available_devices = sycl::device::get_devices(sycl::info::device_type::gpu);
sycl::device root = devices[rank % devices.size()];
...
// Attempt to split the device along NUMA boundaries
std::vector<sycl::device> sub_devices;
sycl::device dev;
...
// Create a single context associated with the root device and all sub-devices
std::vector<sycl::device> devices{root, sub_devices};
sycl::context ctxt(devices);
...
// Create a queue associated with each device
std::vector<sycl::queue> queues;
for (auto& dev : devices) {
  queues.push_back(sycl::queue(ctxt, dev));
}
...
// Allocate memory using the shared context
float* a = sycl::malloc_shared<float>(nelems, root, ctxt);
...
// Execute a kernel on each device
for (int d = 0; d < queues.size(); ++d) {
  sycl::queue q = queues[d];
  q.parallel_for(per_device, [=](sycl::id<1> i) {
    c[offset + i] = a[offset + i] + b[offset + i];
  });
}
```

Allocations are made against the root device (and are visible to all sub-devices).

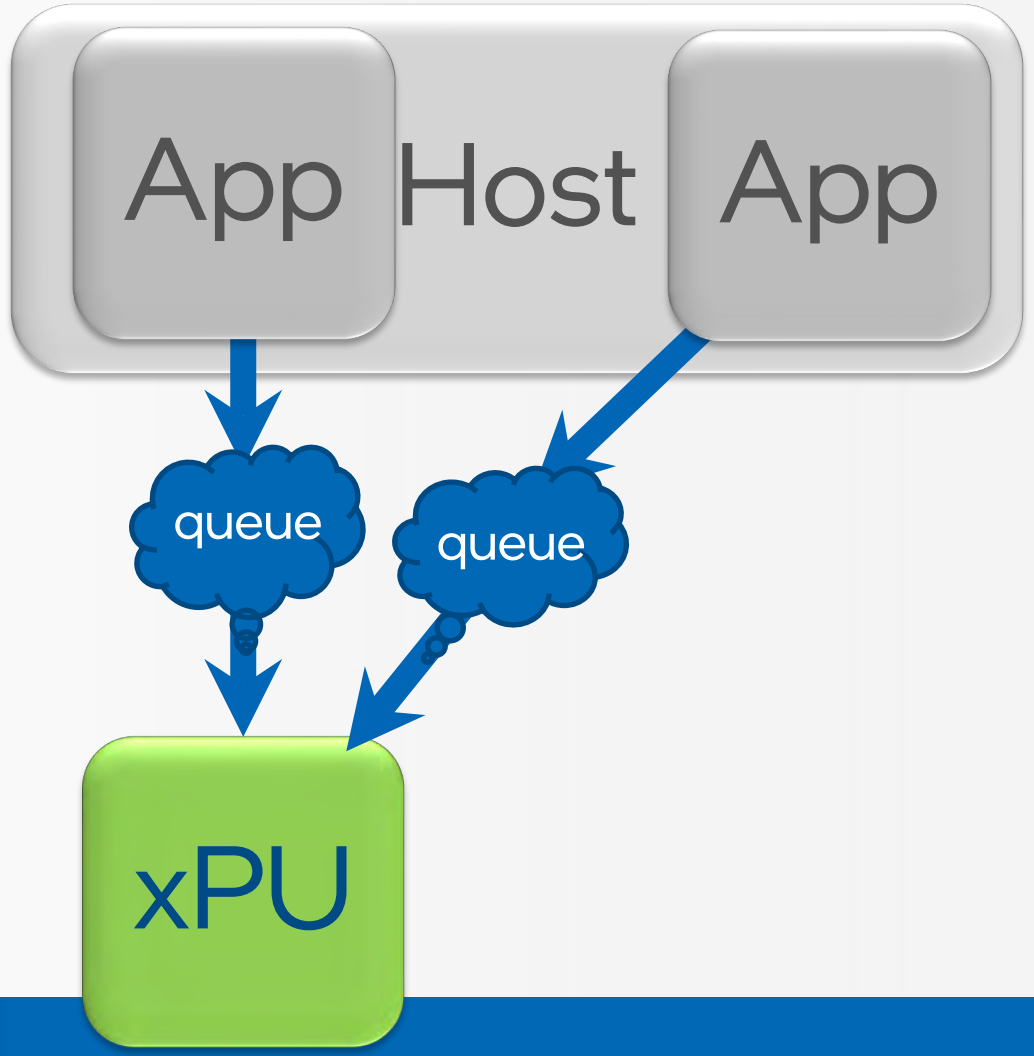
Each sub-device operates on a private chunk of the shared allocation.


XPUG
@ISC21

What about multiple applications?



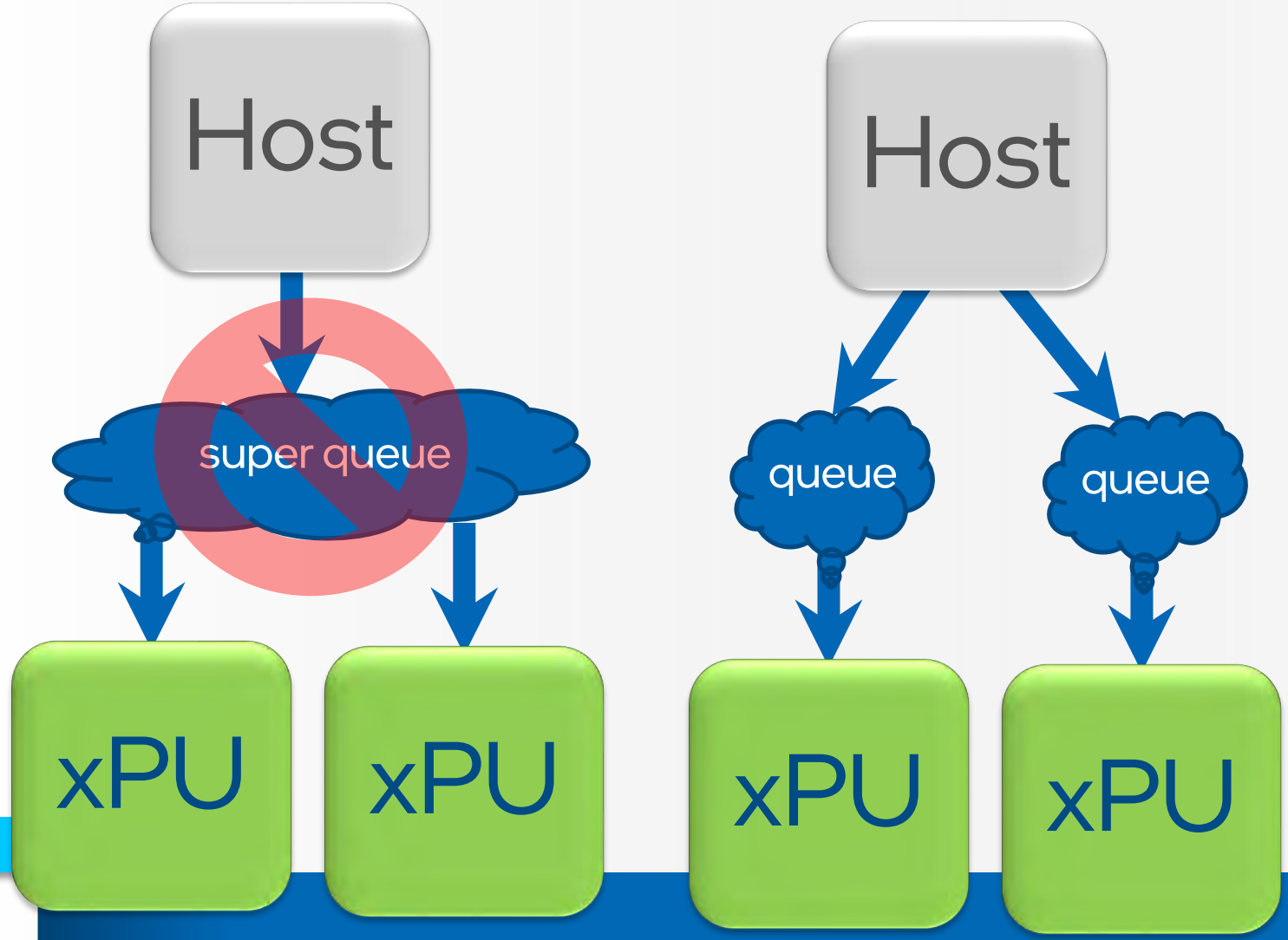
What about multiple applications?



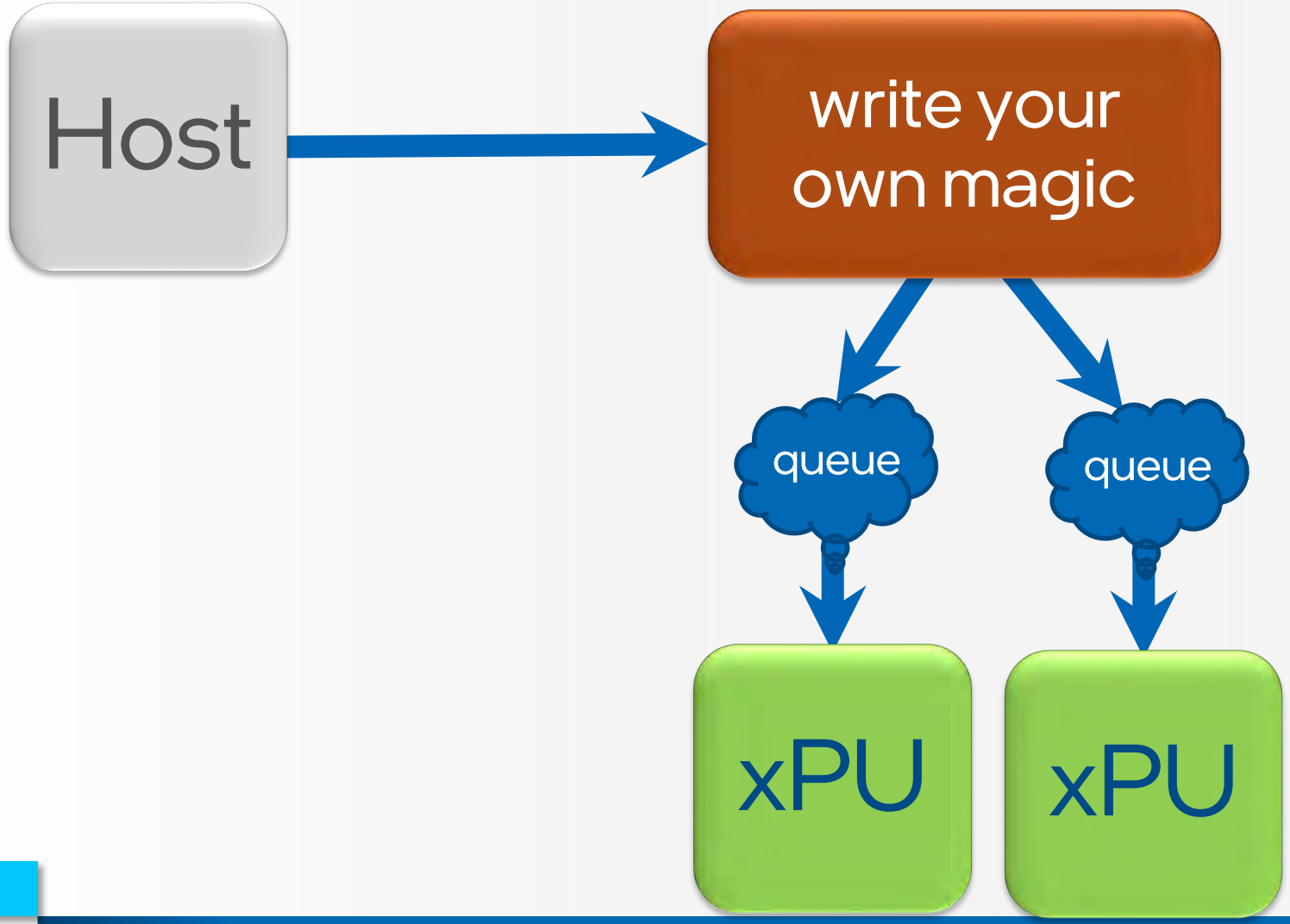
This is what multiple MPI ranks looks like!
Each MPI rank is a process ("App").

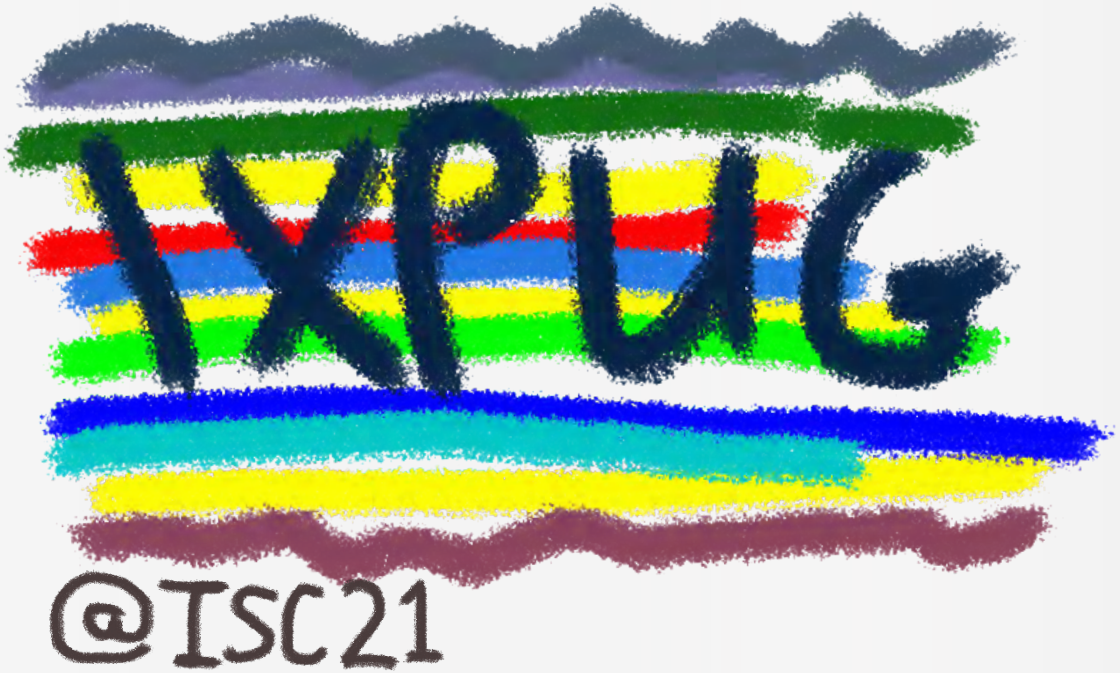

XPUG
@ISC21

one to a customer

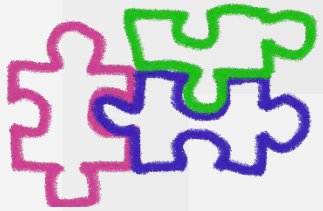


an exercise for the viewer?




XPUG
@ISC21

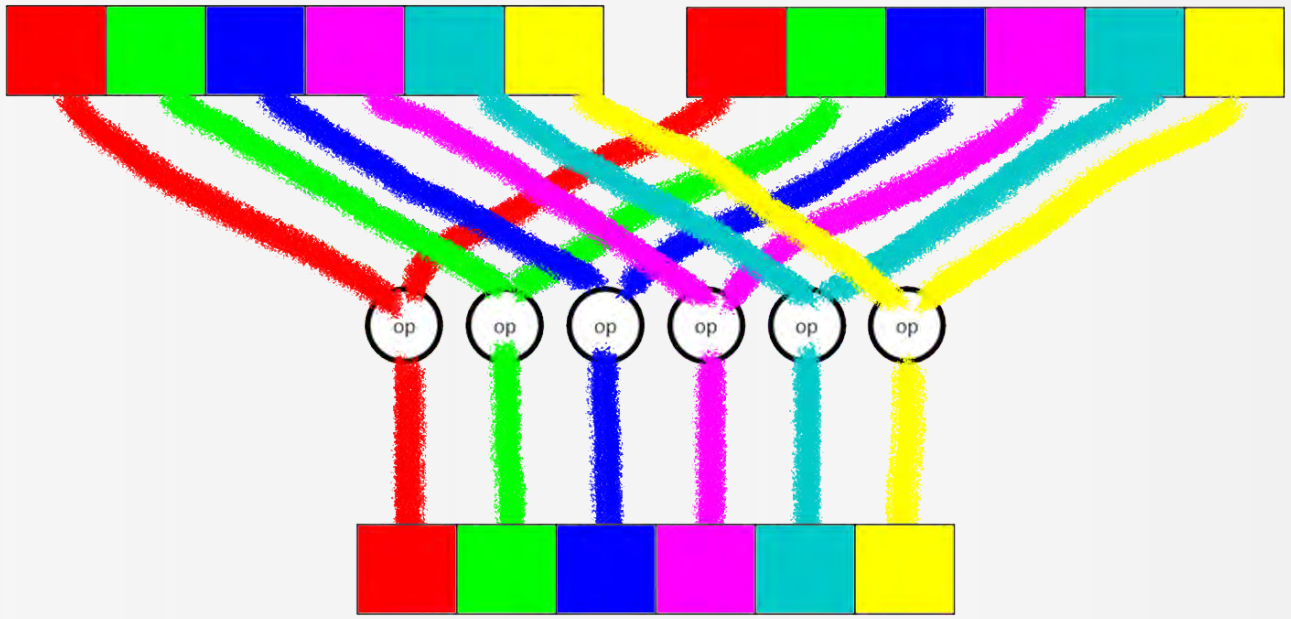
SPMD enough?



- beautiful abstraction,
  works a very large mount of the time!
- access to SIMD hardware can be left as a non-trivial mapping job for compilers or hardware mechanisms (what could go wrong?)

- Good news – we have answers!

SPMD: Color inside your lanes



Every operation on a single lane

Efficient and desirable (highly parallel), but the topic of ‘coloring outside our lanes’ is a recurring question that gets attention.

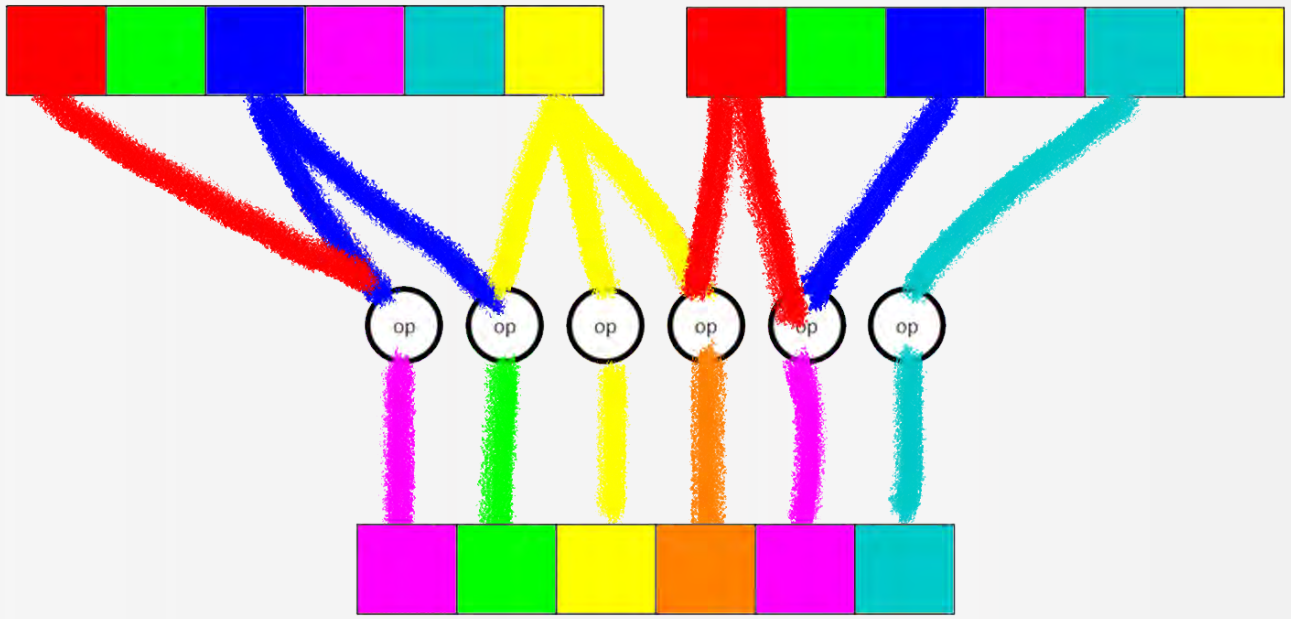
SPMD vs. SIMD

- SPMD models may not take full advantage of SIMD capabilities

We cover “sub-group” in the book, but we leave a lot to the imagination.

The DPC++ project is generalizing sub-group later this year, beyond the current SYCL specification… to help fulfill more of its vision to allow *coloring outside our lanes*.

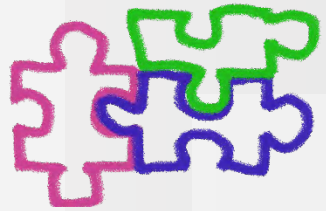
Color ~~inside your lanes~~ anywhere you please



Full SIMD
Whatever we want

Freeing and *can* be very efficient because
many SIMD operations have hardware support *(but not the one shown ☺)*.

What about graphs?



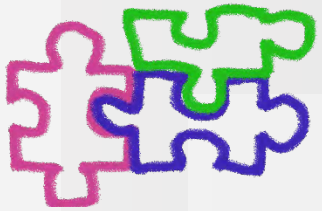
- SYCL supports…

DAG + pipes

- DAG already needed for control flow (to ensure correctness)
- Pipes were added for FPGAs to avoid the default style of "write result to memory, then read it back from memory for the next step"
- Is this well developed? NO – a great area for research and development.


XPUG
@ISC21

Single Source is Cool, and with Ripple effects



I'm used to:

```
$(CPP) -c foo.o foo.cpp
```

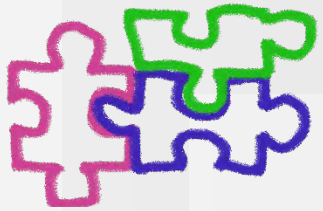
recompiling the entire file.

But now it has code for the host, and code for device(s).

What if I only change the host code?
or just code destined for a single device?

Expect compilers to do interesting things to help,
so we don't have to solve by breaking up source code!


@ISC21
IXPUG



Devices belonging to the same context must be able to access each other's global memory using some implementation-specific mechanism. A given context can only wrap devices owned by a single platform.

Contexts & Queues

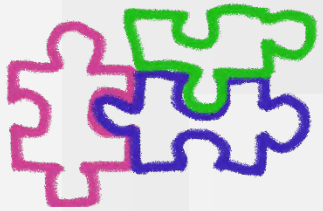
```
// Explicitly create a new context from a device or list of devices
sycl::context(const sycl::device& device);
sycl::context(const std::vector<device> &devices);

// Implicitly create a new context (often by mistake)
sycl::queue();
sycl::queue(const DeviceSelector &selector);
sycl::queue(const sycl::device& device);

// Create a queue associated with a specific context
sycl::queue(const sycl::context& ctxt, const DeviceSelector &selector);
sycl::queue(const sycl::context& ctxt, const sycl::device& device);
```

- OpenMP manages contexts for the user, but they are exposed via interoperability

Contexts



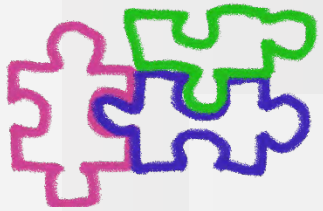
- A context is a collection of one or more (sub-)devices.

- Programs are built per context, and implementations can optimize based on the context.

- Allocations are bound to contexts, not to devices!


IXPUG
@ISC21

Final SYCL 2020 caused a book errata



SYCL 2020
Appendix D: What has changed from previous versions

A SYCL implementation is no longer required to provide a host device. Instead, an implementation is only required to provide at least one device. Implementations are still allowed to provide devices that are implemented on the host, but it is no longer required. The specification no longer defines any special semantics for a "host device" and APIs specific to the host device have been removed.

No such thing as a 'host device'

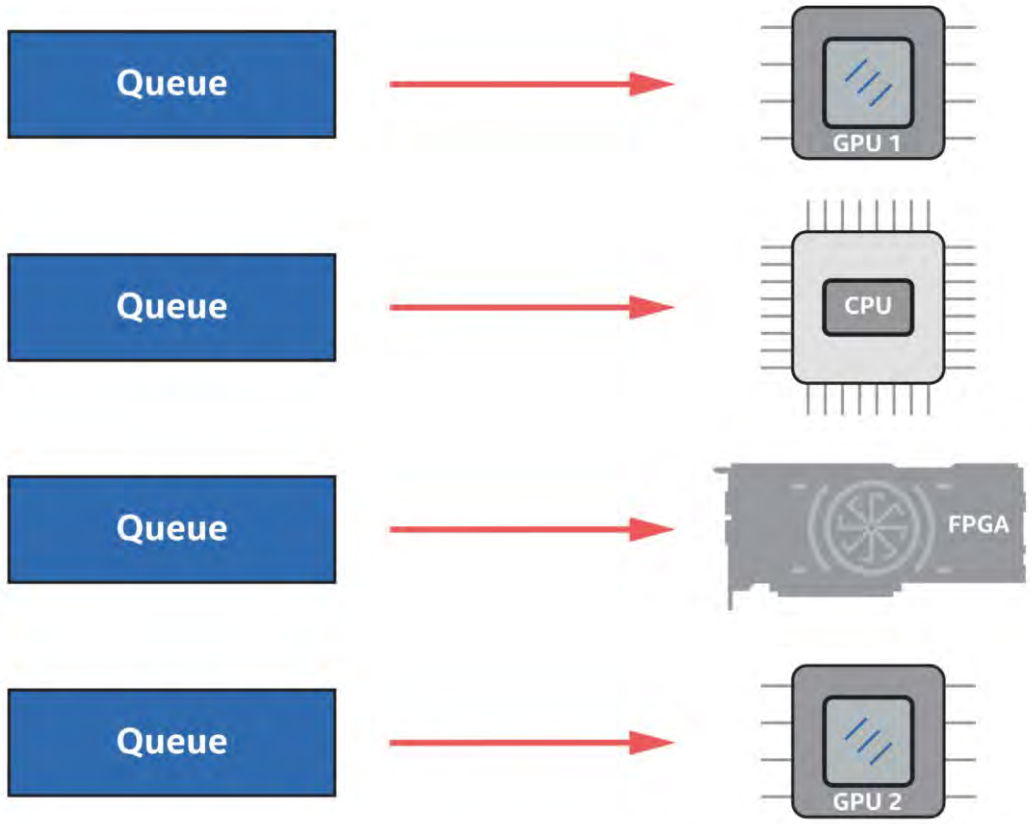


Figure 2-5. *A queue is bound to a single device. Work submitted to the queue executes on that device*

No such thing as a ‘host device’

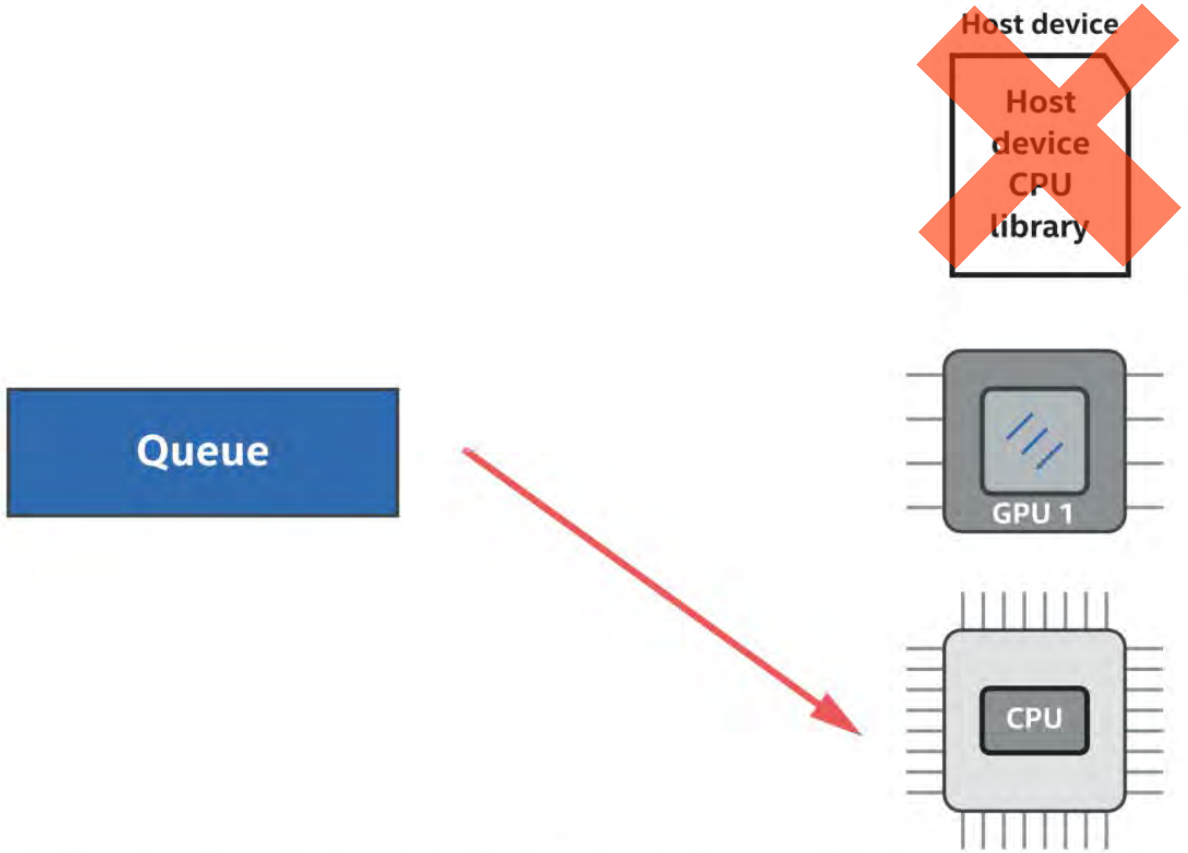


Figure 2-11. *Queue bound to a CPU device available to the application*

No such thing as a 'host device'

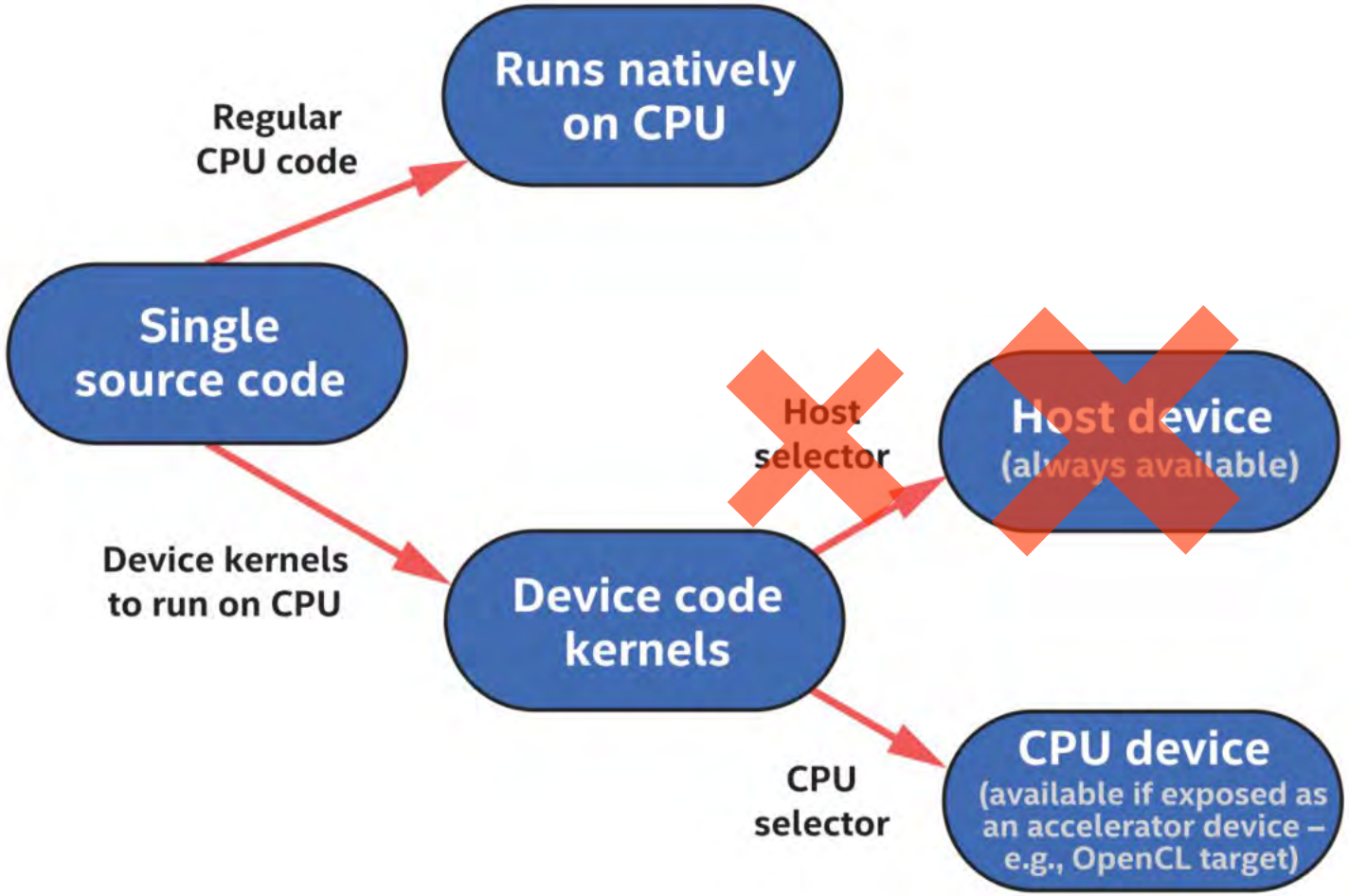
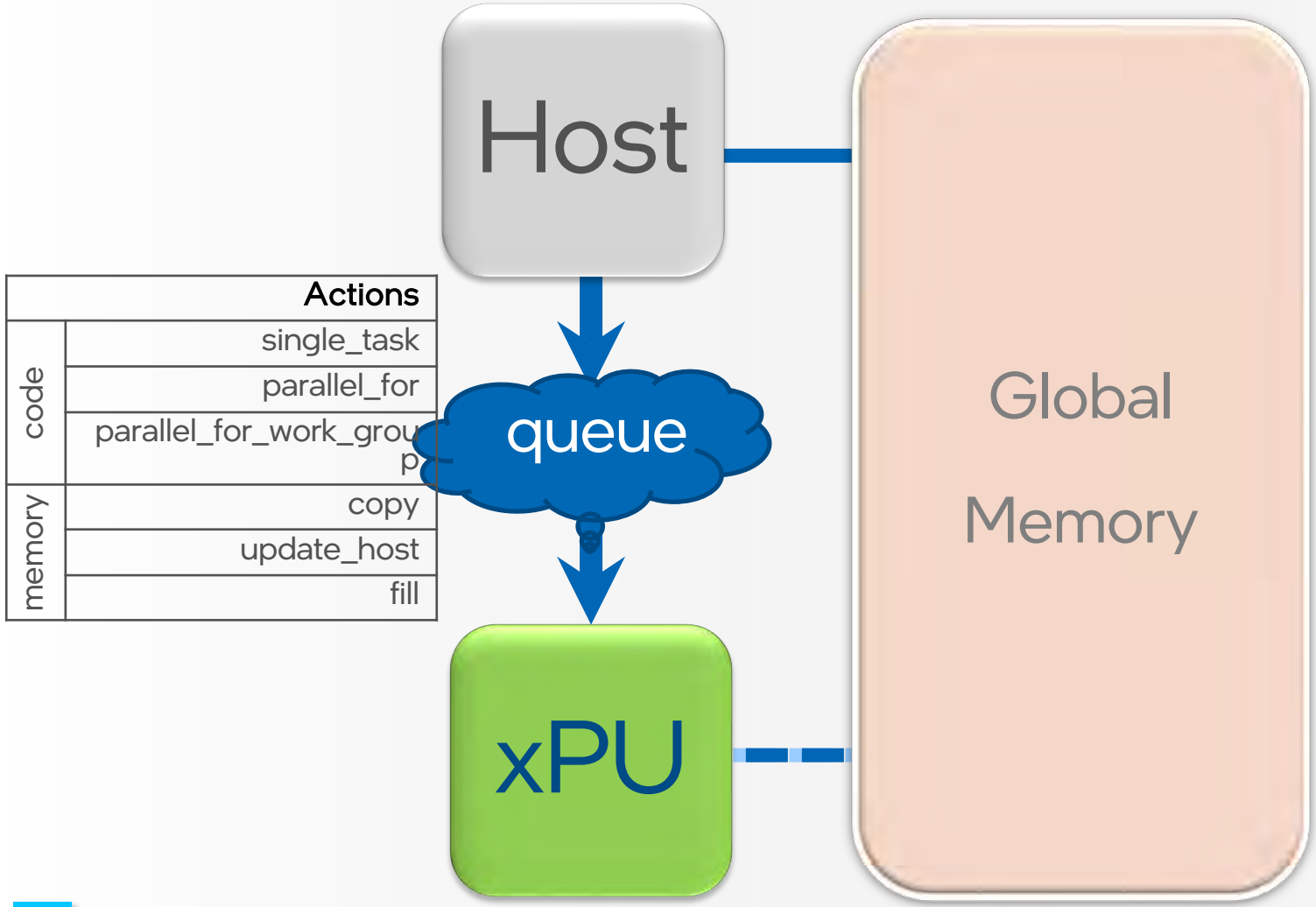


Figure 2-16. *SYCL mechanisms to execute on a CPU*

That's the rest of the story on queues.



device selection – the "_v" is new in SYCL 2020

nonchalant

```
queue();
queue(default_selector_v);
```

selective

```
queue(cpu_selector_v);
queue(gpu_selector_v);
queue(accelerator_selector_v);
queue(INTEL::fpga_emulator_selector_v);
queue(INTEL::fpga_selector_v);
```

full unmitigated control freak

```
queue(my_custom_device_selector_v);
```


XPUG
@ISC21

If anything was of interest you –
esp. if you want more – drop me a note!
(I want more code examples.)

james.r.reinders@intel.com

(LinkedIn jamesreinders)

I am *extremely* interested in feedback regarding ways to convey tips-and-tricks, techniques, insights, etc. that help us all be more effective.

xpublog?4


intel