oneAPI Technical Advisory Board Meeting:

Proposals for oneAPI Level Zero Feature Evolution

8/25/2021

Paul Petersen, Ben Ashbaugh, Mike Voss, Xinmin Tian

Rules of the Road

- DO NOT share any confidential information or trade secrets with the group
- DO keep the discussion at a High Level
  - Focus on the specific Agenda topics
  - We are asking for feedback on features for the oneAPI specification (e.g. requirements for functionality and performance)
  - We are NOT asking for feedback on any implementation details
- Please submit any implementation feedback in writing on Github in accordance with the Contribution Guidelines at spec.oneapi.com. This will allow Intel to further upstream your feedback to other standards bodies, including The Khronos Group SYCL* specification.

Notices and Disclaimers

The content of this oneAPI Specification is licensed under the Creative Commons Attribution 4.0 International License . Unless stated otherwise, the sample code examples in this document are released to you under the MIT license.

This specification is a continuation of Intel's decades-long history of working with standards groups and industry/academia initiatives such as The Khronos Group*, to create and define specifications in an open and fair process to achieve interoperability and interchangeability. oneAPI is intended to be an open specification and we encourage you to help us make it better. Your feedback is optional, but to enable Intel to incorporate any feedback you may provide to this specification, and to further upstream your feedback to other standards bodies, including The Khronos Group SYCL* specification, please submit your feedback under the terms and conditions below. Any contribution of your feedback to the oneAPI Specification does not prohibit you from also contributing your feedback directly to The Khronos Group or other standard bodies under their respective submission policies.

By opening an issue, providing feedback, or otherwise contributing to the specification, *you agree that Intel will be free to use, disclose, reproduce, modify, license, or otherwise distribute your feedback in its sole discretion without any obligations or restrictions of any kind, including without limitation, intellectual property rights or licensing obligations.* For complete contribution policies and guidelines, see Contribution Guidelines on www.spec.oneapi.com.

Introduction

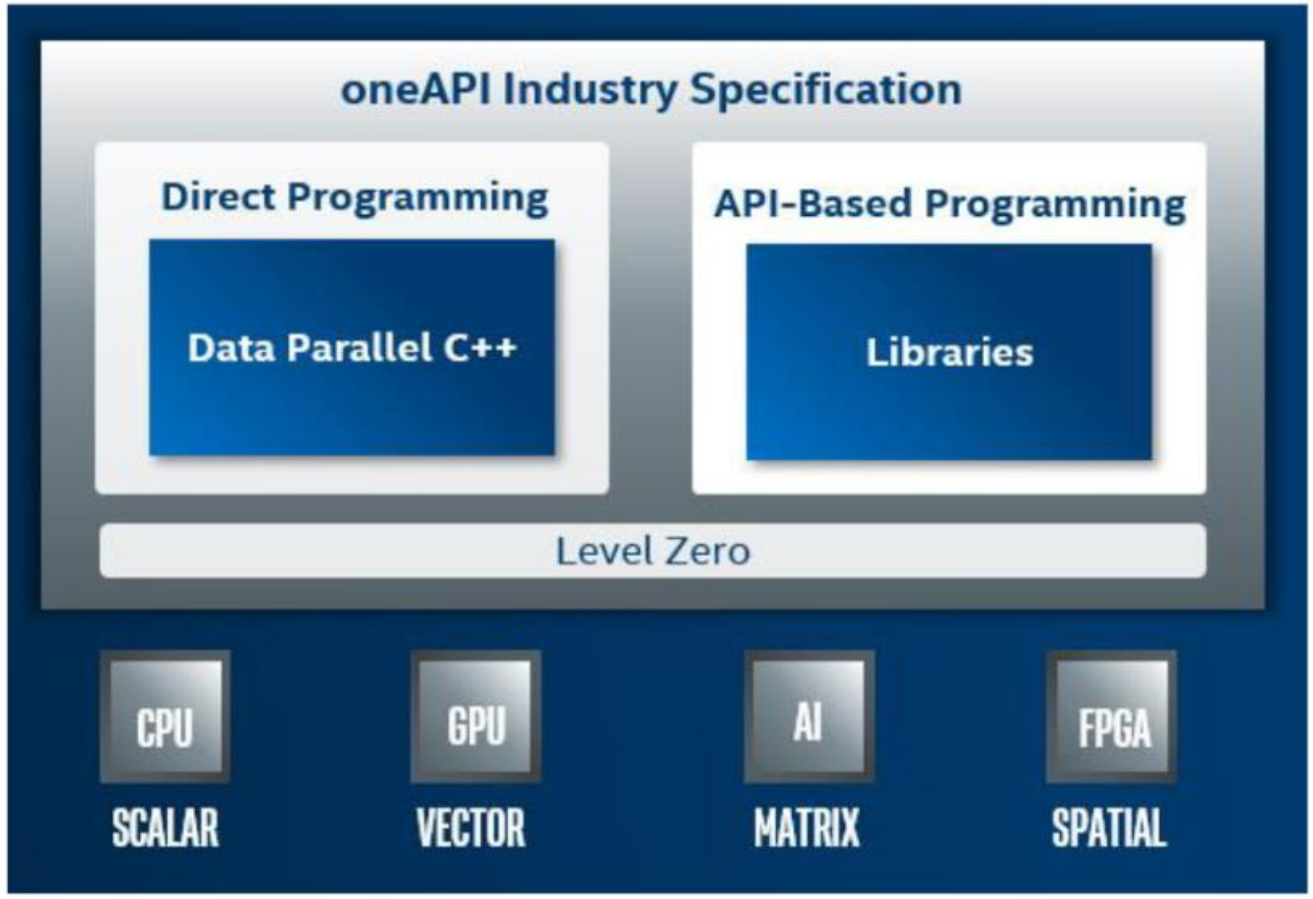
oneAPI Evolution

- The oneAPI specification enables portable access to XPU devices
- We are proposing to evolve the oneAPI specification
- This will include functionality to better integrate the CPU
  - The CPU is an XPU, but also is more widely used by the OS
  - Are any Level Zero API extensions needed for best CPU support?

- We are looking for feedback on the direction, and any need to generalize these concepts further to provide more value

oneAPI Foundation – Level Zero

- Level Zero is defined to be the foundation for our software stack
- We are proposing several extensions to Level Zero to deliver on this definition
- By extending our foundation all higher-level languages and libraries which target Level Zero can directly benefit



Topics for Today

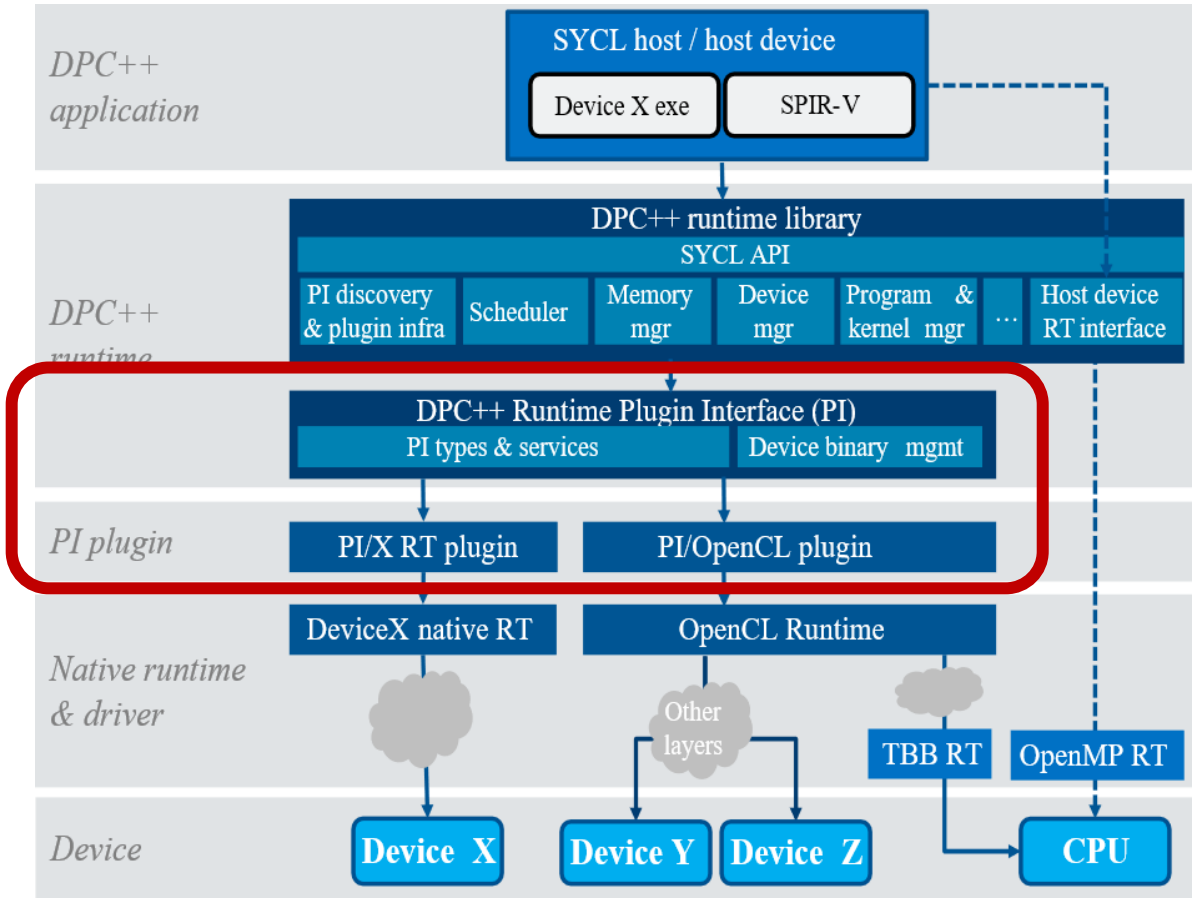
We will be covering 3 proposals

1. Adding higher-level functionality and adaptability to Level Zero
2. Extensions to support CPU resource management in Level Zero
3. Kernel execution on CPU through Level Zero

Level Zero Runtime API

"Adding higher-level functionality and adaptability to Level Zero"

DPC++ Runtime Plugin Interface

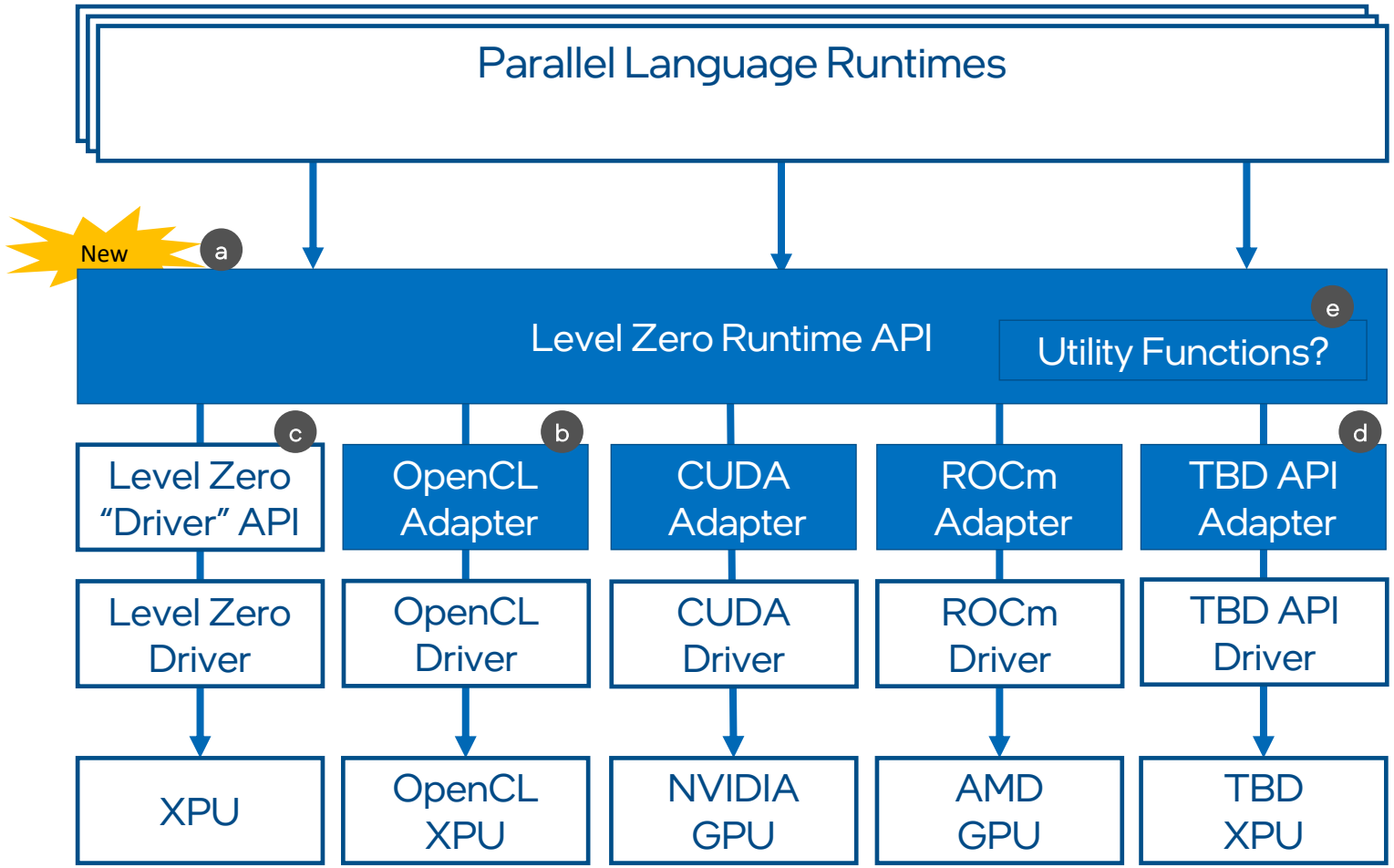


Problem Statement:

- Plugin Interface is an implementation detail
  - No formal specification
- Plugin Interface is only usable by the DPC++ Runtime
  - Other language runtimes must duplicate functionality
- Plugin Interface has deep OpenCL heritage

Diagram Source: https://github.com/intel/llvm/blob/sycl/sycl/doc/PluginInterface.md
(Note: slightly out-of-date!)

Proposal: Level Zero Runtime API



a. Refactor and formalize Plugin Interface into Level Zero Runtime APIs for use by multiple language runtimes

b. Refactor existing plugins into Level Zero Runtime API Adapters

c. Refactor, generalize, and modernize adapter interface

d. Enable new backends by implementing new Level Zero Runtime API Adapters

e. Eventually: Move common utility functionality to Level Zero Runtime API?

oneAPI

Questions for TAB

- Would your language runtime support the Level Zero Runtime API?
- Would a formal Level Zero Runtime API Adapter interface help you to enable additional backend APIs?
- What other backend APIs we should consider as we are designing the Level Zero Runtime API?
- What utility functions should we consider adding to the Level Zero Runtime API?

Resource Manager

"Extensions to support CPU resource management in Level Zero"

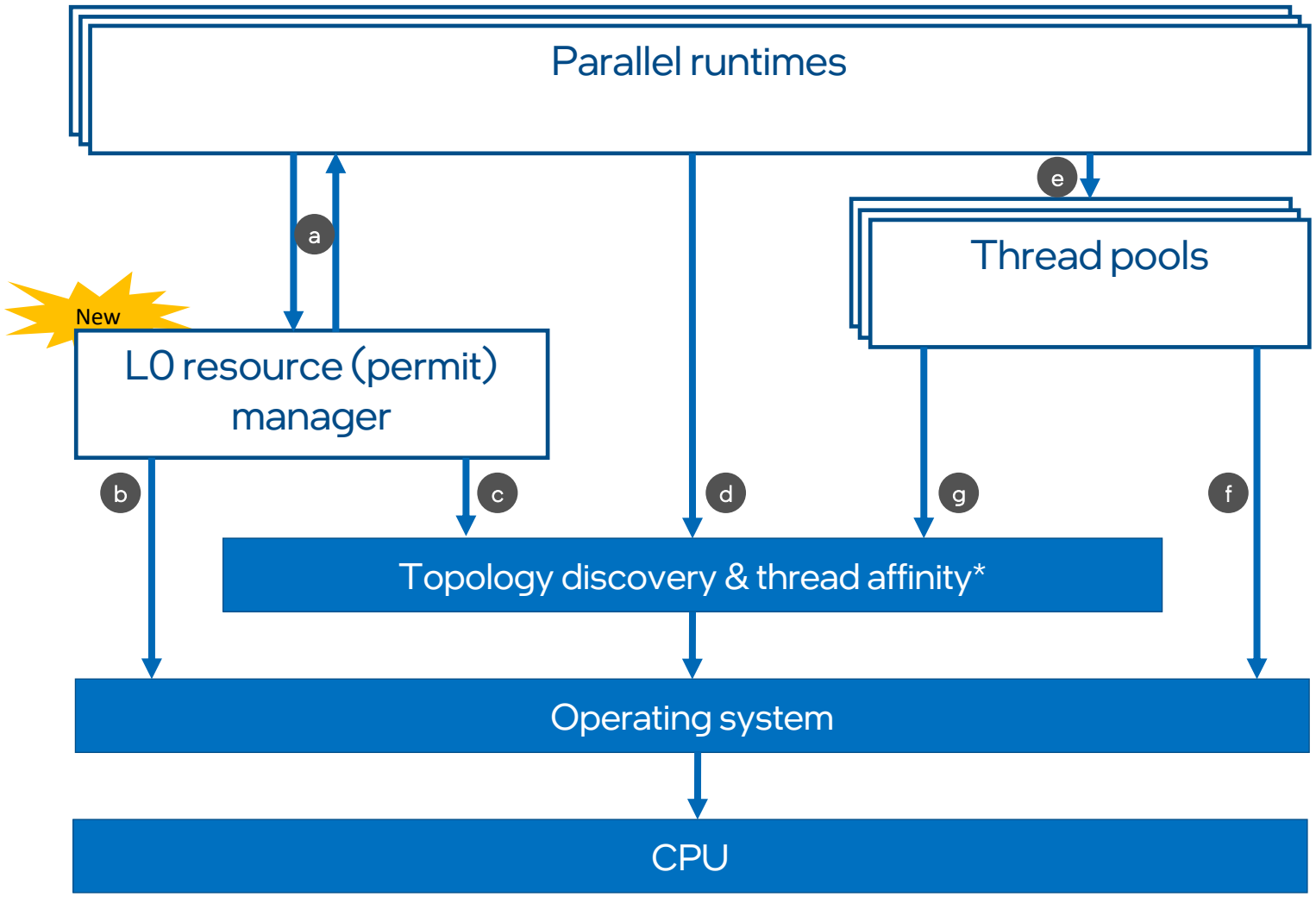
Resource Manager Philosophy

- Goal: Enable coordination and composition among higher-level threading runtimes -- DPC++ on the host, oneTBB and other runtime implementations that opt-in such as OpenMP or thread pools.
- Assumes these higher-level runtimes that opt-in are well-behaved and non-malicious
- Manages permissions for resources, not threads
- Key characteristics:
  - A resource permit is granted for a specific subset of available hardware, not just a thread count
  - Support for (re-)negotiating permits for more resources and fewer resources
  - Support policies for space and time partitioning

Questions for TAB

- Is CPU resource management a problem that should be addressed by oneAPI?
- Does the general approach outlined here seem reasonable?
- What features should be included in a permit request?
- What constraints should be included in a granted permit?
- Should we provide discovery interfaces or expect use of an existing library, like hwloc?
- Will non-oneAPI runtimes opt-in to a resource manager?
- Would other non-CPU devices benefit from resource management?
- Should there be a default management policy?
- How should priorities be managed (if at all)?
  - Some models support priorities while other models do not.
  - Should priorities apply only locally to requests from the same client
  - Or should priorities be applied global across clients?

Resource Management Architecture



a. The resource management protocol is two-way, to allow for efficient renegotiation of permits
b. CPU resources available for the process are set by/via OS
c. Resource manager obtains topology information
d. Parallel runtimes obtain topology information
e. Permit-aware thread pool APIs. Thread pools can be private (to a runtime) or shared
f. Thread pools use OS API to manage threads
g. Thread pools manipulate thread affinities

* There are existing libraries such as hwloc that support topology discover and thread affinity. An implementation might select one of these.

Resource Permit States

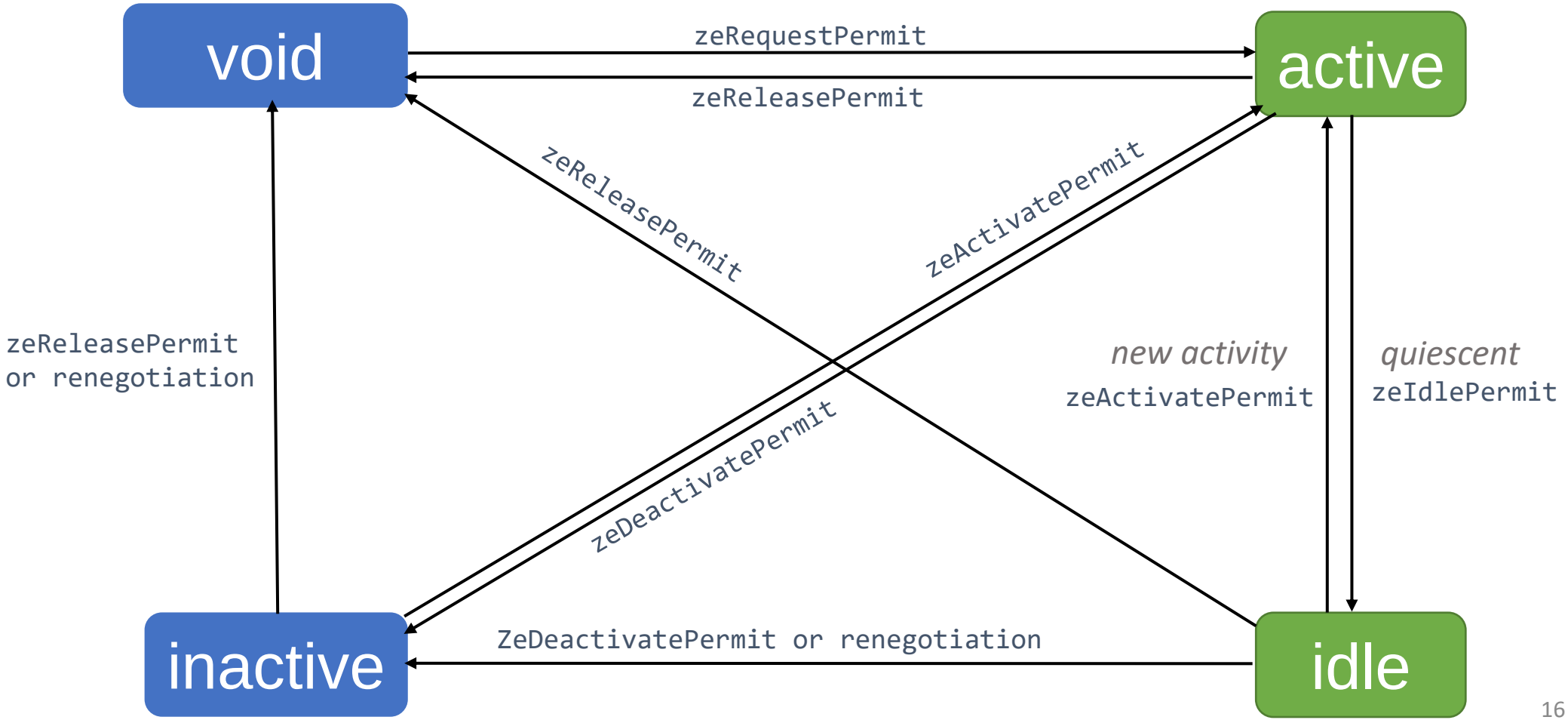
Resource manager

Client (Language RT)

oneAPI

Client initiates connection via zeResourceManagerConnect(callback,constraints,grid)
When connecting, a client can provide constraints for what resources it can use.

Resource manager may call the callback to force a permit renegotiation.



Possible Request and Permit Structures

```
// ---- input: permit request
struct ze_permit_request_t {
  ze_permit_handle_t permit_handle;     // optional: existing permit
  uint32_t min_sw_threads;              // minimum required concurrency
  uint32_t max_sw_threads;              // maximum desired concurrency
  ze_cpu_mask_handle_t cpu_mask_handle; // optional: low-level requested resources
  ze_cpu_constraints_t constraints;     // optional: more abstract constraints such as NUMA ID, core type, etc.
  ze_permit_flags_t props;              // optional: other requested properties
};

// ---- output: granted permit
struct ze_permit_t {
  ze_client_id_t grid,
  uint32_t max_sw_threads;              // granted concurrency
  ze_cpu_mask_handle_t cpu_mask_handle; // granted resources
  ze_permit_flags_t props;              // state, other granted properties
};
```

An example: nested parallelism

```
// OpenMP RTL initialization
// TBB RTL initialization
tbb::parallel_for(0, 100, [](int) {
    /*TBB threads working*/
    #pragma omp parallel for

    for(int i = 0; i < 100; ++i) {
        /*OpenMP threads working*/
    }

});
// end of program
```

```
zeResourceManagerConnect(&omp_cb, &omp_id); // OMP
zeResourceManagerConnect(&tbb_cb, &tbb_id); // oneTBB
zeRequestPermit(tbb_id, …, &tbb_permit);    // oneTBB
zeRegisterThread(tbb_permit);   // each oneTBB thread
zeRequestPermit(omp_id, …, &omp_permit); // OMP, multiple
// OpenMP may wait till callback notifies of a granted permit
zeRegisterThread(omp_permit);   // each OpenMP thread


zeUnregisterThread(); // each OpenMP thread
zeDeactivatePermit(omp_permit);
zeUnregisterThread(); // each oneTBB thread
zeReleasePermit(tbb_permit);
zeReleasePermit(omp_permit);
```

Should there be a default management policy? Management policy choices?

- Is the policy purely a quality of implementation issue?
- If not, what would the default or non-default options be?
  - A greedy first-come, first served policy
  - A policy that tries to achieve fairness among clients?
  - Others…?
- Should the resource manager try to infer nesting and other scenarios from the thread registrations?

What to do about priorities?

- Some runtimes support priorities and others do not.
- oneTBB supports three levels of priorities for its tasks
  - low, normal and high
- Other models, such as OpenMP, do not have priorities
- Should requests from models without priorities be considered as normal? as high?
- Should priorities only apply across requests from the same group/client? Or should they apply globally?

Questions for TAB

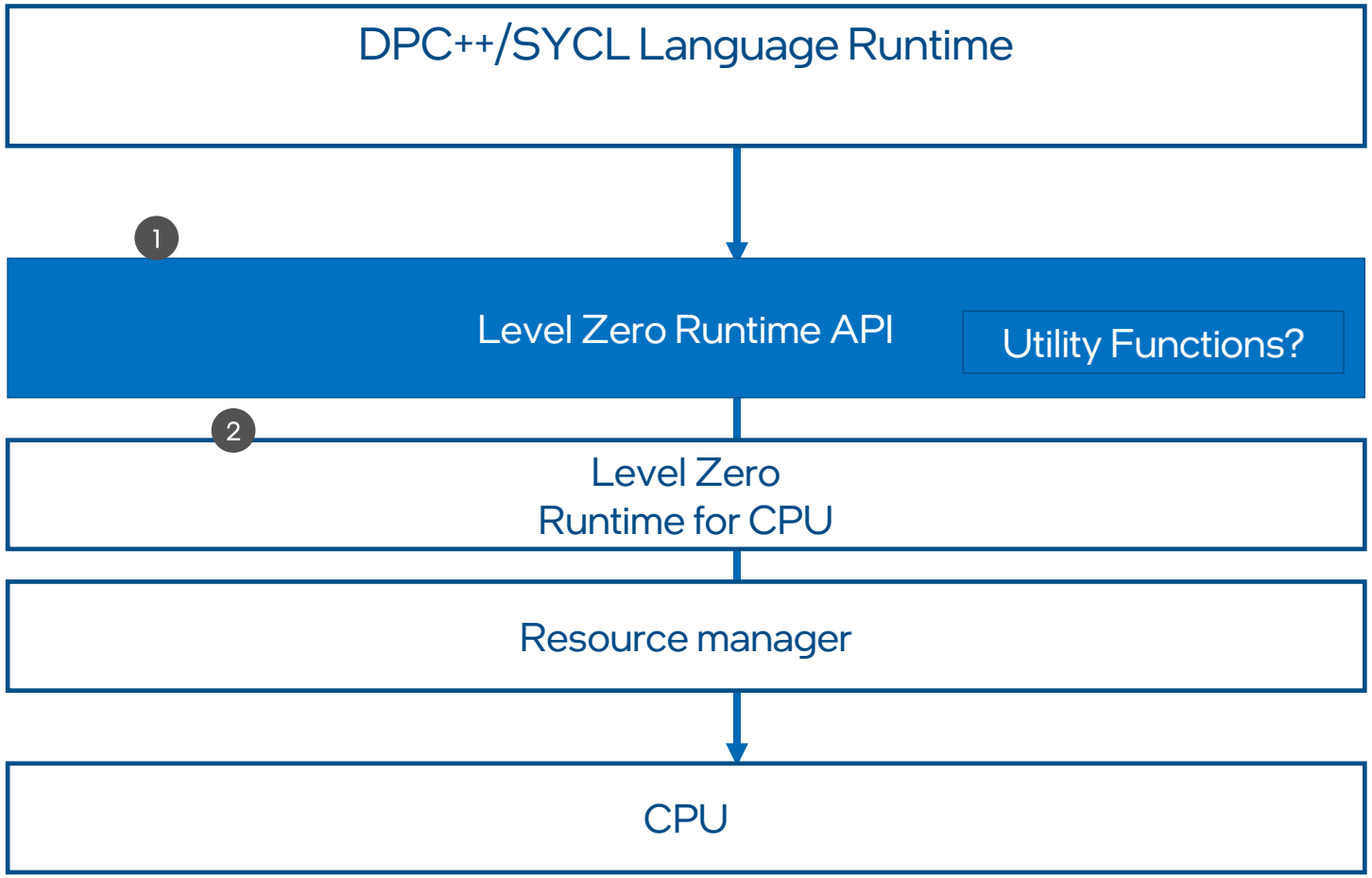
- Is CPU resource management a problem that should be addressed by oneAPI?
- Does the general approach outlined here seem reasonable?
- What features should be included in a permit request?
- What constraints should be included in a granted permit?
- Should we provide discovery interfaces or expect use of an existing library, like hwloc?
- Will non-oneAPI runtimes opt-in to a resource manager?
- Would other non-CPU devices benefit from resource management?
- Should there be a default management policy?
- How should priorities be managed (if at all)?
  - Some models support priorities while other models do not.
  - Should priorities apply only locally to requests from the same client
  - Or should priorities be applied global across clients?

oneAPI

Level Zero CPU Runtime

"Kernel execution on CPU through Level Zero"

Proposal: Level Zero Runtime For CPU



1. Refactor and formalize Plugin Interface into Level Zero Runtime APIs for use by multiple language runtimes
2. Add CPU runtime support based on Level Zero Runtime API

Questions for TAB

- Any requirements you consider needed to add for CPU support?
- Would your language runtime support the Level Zero CPU Runtime?
- What additional functions should we consider adding to the Level Zero CPU Runtime?

Call to Action

- The oneAPI specification enables portable access to XPU devices
- We are proposing to evolve the oneAPI specification
- This will include functionality to better integrate the CPU
  - The CPU is an XPU, but also is more widely used by the OS
  - Are any Level Zero API extensions needed for best CPU support?

- We are looking for feedback on the direction, and any need to generalize these concepts further to provide more value