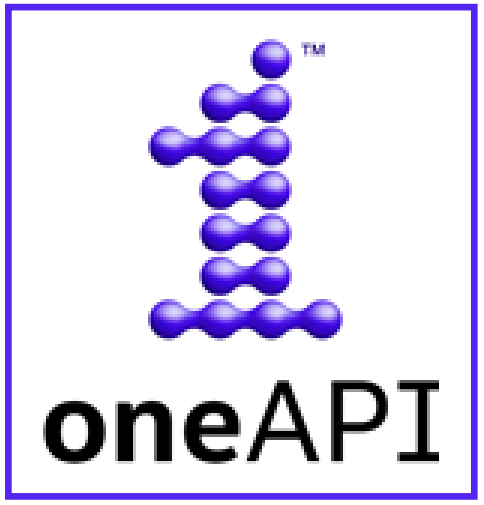
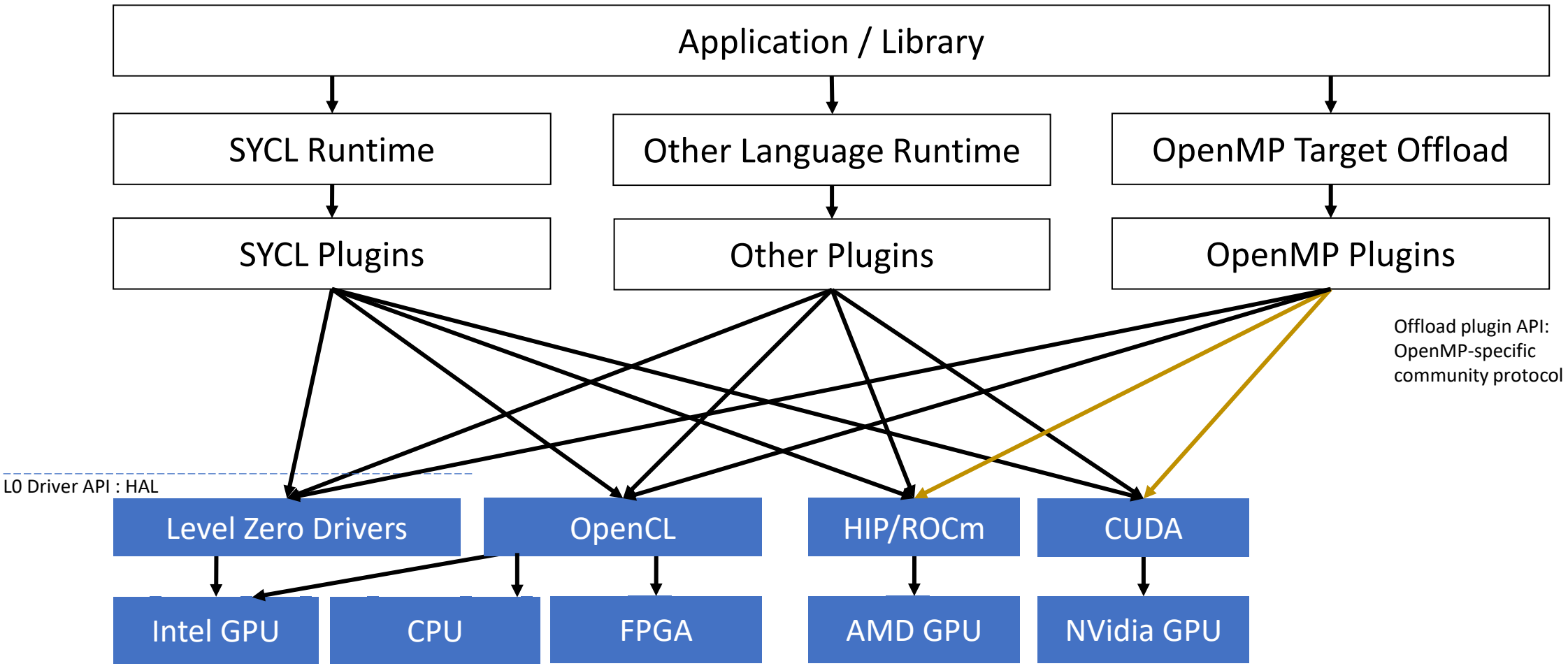
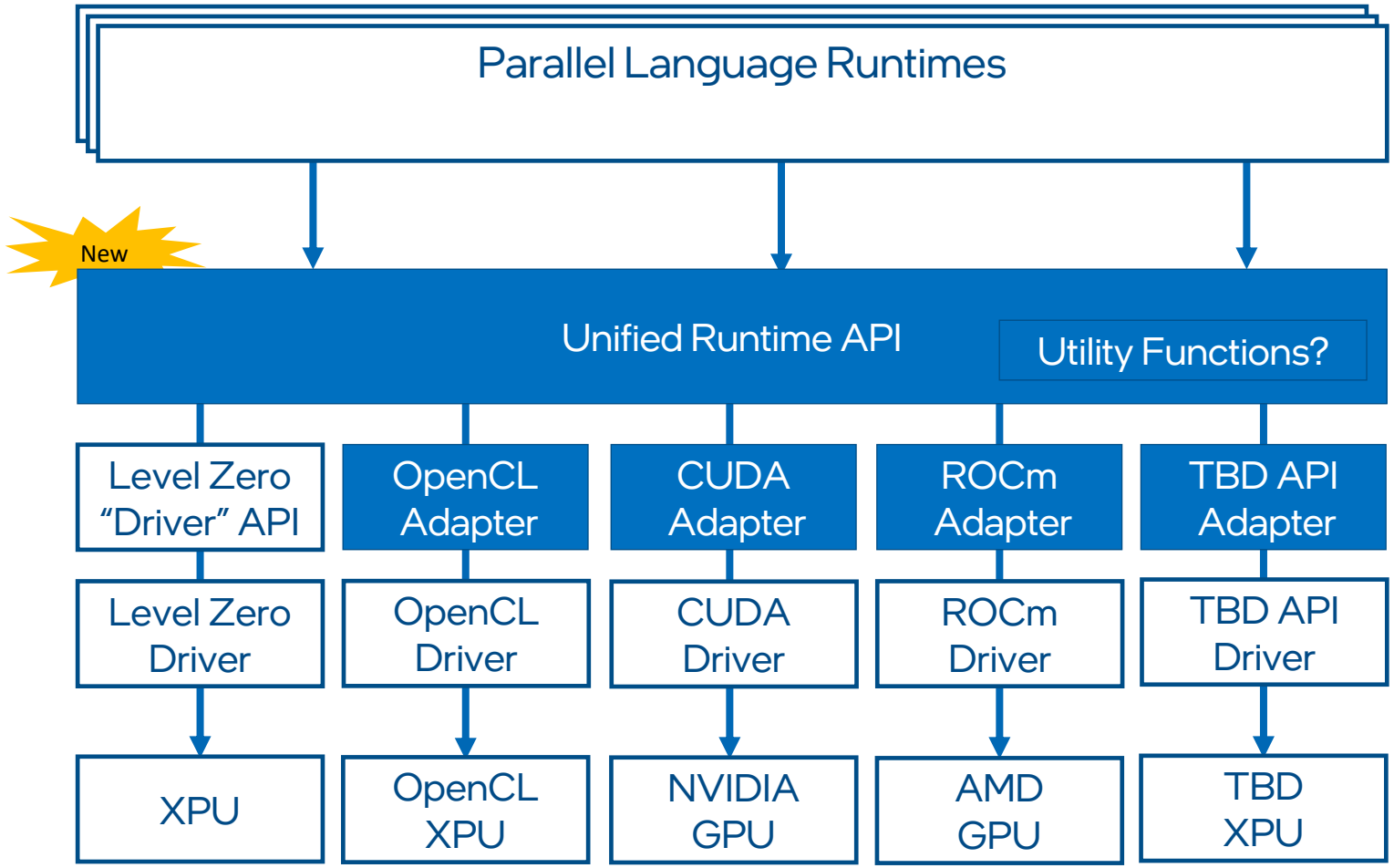


Unified Runtime Direction Discussion

Recap Problem Statement: Today's Language Runtime Stack



Recap from August 18th TAB:



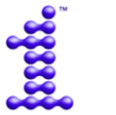
Reviewed Unified Runtime Specification v0.5: https://spec.oneapi.io/unified-runtime/latest/index.html

Received Feedback:

- Can the Unified Runtime build upon OpenCL 3.0 instead?
- Can an application or language runtime call directly into low-level layers if needed?
- Need to clarify the value proposition for the Unified Runtime, especially for existing language runtimes.

A Brief History

- At the time of OpenCL 2.0
  - OpenCL was too large/complicated/low adoption
- Decision to have two layers: SYCL and Level Zero
  - SYCL as language for developers
  - Level Zero as close-to-machine driver
- However, to enable more users and more drivers a middle layer was needed, i.e. Level 1 or Unified Runtime.
  - With adapters that can target L0, CUDA, OpenCL etc.
  - Provide additional APIs needed by OpenMP, Python, Java, etc.
  - The glue between diverse runtimes/frameworks and diverse native APIs
- In the meantime, OpenCL 3.0 was released with smaller minimal feature set, optional APIs, and extensions (e.g. USM).



Options for discussion

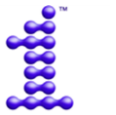
- Continue with Unified Runtime API as an independent API
- Define the Unified Runtime API as OpenCL 3.0 APIs plus extensions

Some pros / cons for each approach

- Unified Runtime as an independent API
  - + Can more quickly / flexibly evolve
  - + Existing oneAPI SYCL plugins can easily be moved from PI to UR
  - + Can ignore features not needed by oneAPI Apps/Middleware
  - - No existing community, cannot leverage mature specification
  - - Likely perceived as not as open (can we fix this perception or is that not an issue?)
- Unified Runtime as OpenCL 3.0 plus extensions
  - + Perceived as more open
  - + Can leverage existing community and specification
  - + Can directly leverage existing OpenCL drivers
  - - Brings in potentially unneeded features
  - - Slower to evolved due to bigger community
  - - Difficult / impossible to build on top of native APIs such as CUDA in a way that is conformant with OpenCL to specification

API Comparison

- Unified Runtime has similar execution model to OpenCL 3.0 and many common APIs
- Main concerns w/ OpenCL 3.0 APIs include:
  - Lack of interoperability and access to native backends (CUDA, L0)
  - Standard vs. vendor-specific extensions for key features (USM)
  - Unified Runtime is leaner, even similar APIs have simpler implementation and fewer options. **TODO: Example?**
- NOTE: The proposal under consideration is NOT to require all vendors to implement a conformant OpenCL driver
  - Some vendors/plugins will implement OpenCL APIs via other native backends



Tradeoffs and Technical Considerations

- Will Unified Runtime diverge or converge with OpenCL 3.0?
  - Will the leaner implementation require the options currently excluded?
  - Will demands of other native backends (CUDA) and other users (OpenMP, Java, Python) require many extensions?
- Will working a large standards body slow down innovation and flexibility?
- How will we support tools like debuggers and profilers?
- How will we support other higher-level utility functionality?

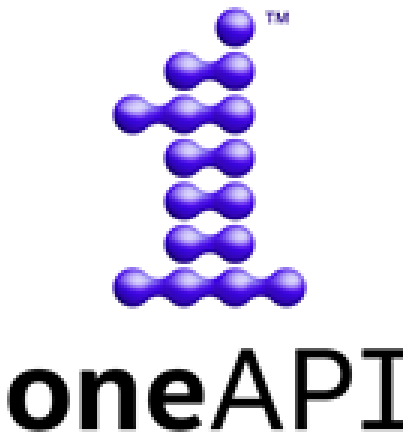
Lessons from PI Layer Extensions

- TODO

Questions for TAB

- TODO: seed questions for discussion

- Non-conforming OpenCL implementations – big deal or no?



Thank You!

http://oneapi.com