Graphics Architecture

Windows Subsystem for Linux

About us

Jesse Natalie

Developer on Direct3D

Steve Pronovost

Lead Windows Graphics Kernel

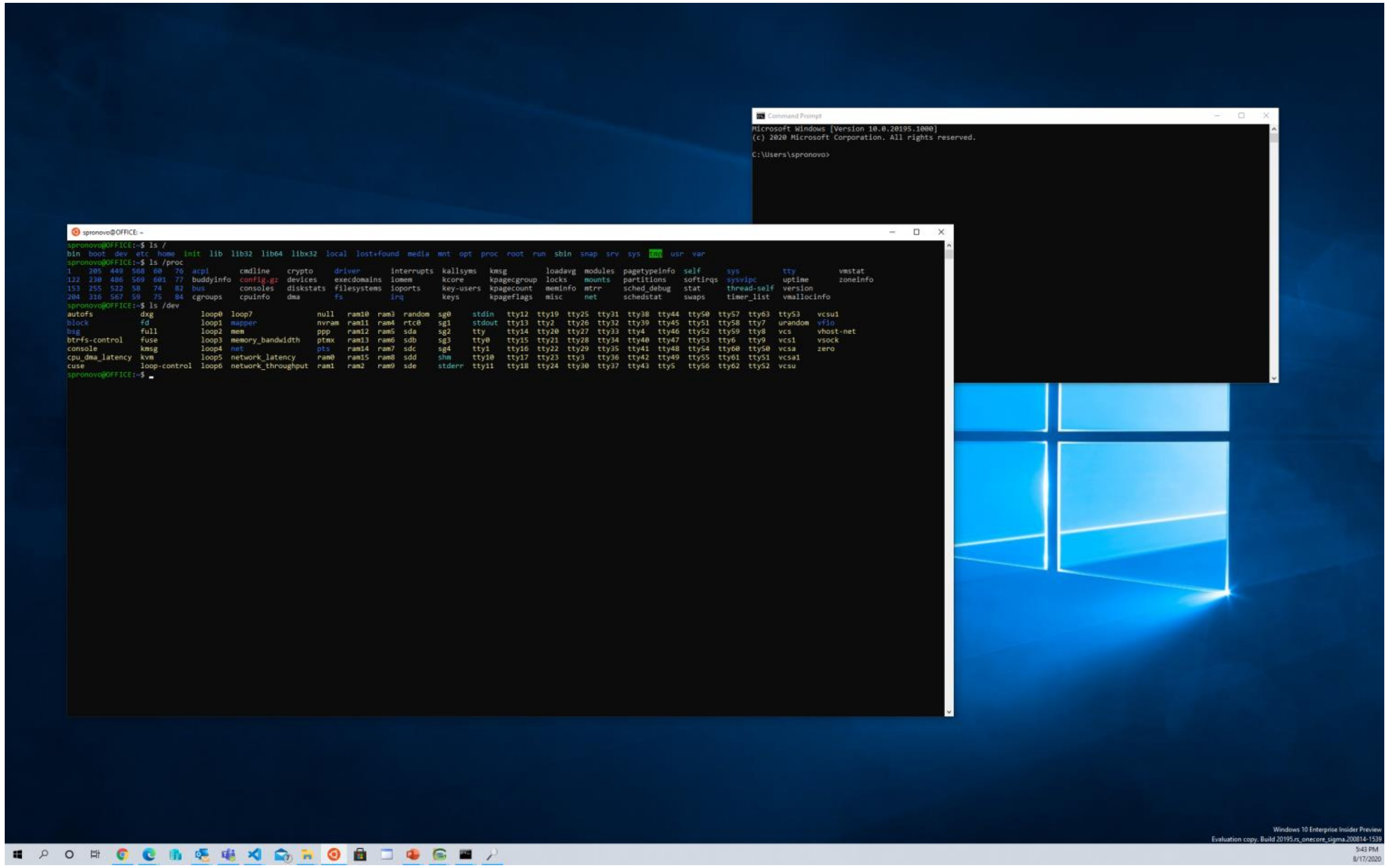
What is WSL?

- **W**indows **S**ubsystem for **L**inux
  - Infrastructure to run Linux applications inside of Windows
  - Today only terminal applications are supported

Why WSL?

- But can’t you just run Linux inside of a VM already?
  - Yes, but managing a VM is a pain and not user friendly
- WSL is all about developers
  - Creating a friendly and integrated experience for developers that needs both a Windows and Linux development environment
    - Some tools run best or only on Windows
    - Some tools run best or only on Linux
  - Get the best of both worlds
    - Developer can run their Windows and Linux development workflow from a single PC
      - No clunky dual boot setup
      - No need for multiple PCs
      - No need for manually managed VM

WSL



- Terminal integration
- Filesystem integration
- Windows / Linux interop
- … and many more

- Limited to terminal applications today

WSL 1 vs WSL 2

- WSL 1
  - Linux userspace running against an emulated Linux Kernel
  - Linux userspace isolated in a pico process
  - Linux userspace call to kernel trap and emulated on top of ntos
- WSL 2
  - Full Linux userspace and Linux kernel running in a VM
  - Same integrated experience
  - Better compat (no more kernel emulation)

Most requested WSL features

- Access to the GPU from within WSL
  - Mostly for compute
  - Most requested is access to NVIDIA CUDA API
  - Subject of this talk
- Ability to run GUI applications
  - Going beyond a terminal only experience and the ability to run X11 and Wayland applications
  - This is the subject of our other XDC talk
    - *X11 and Wayland applications in WSL*

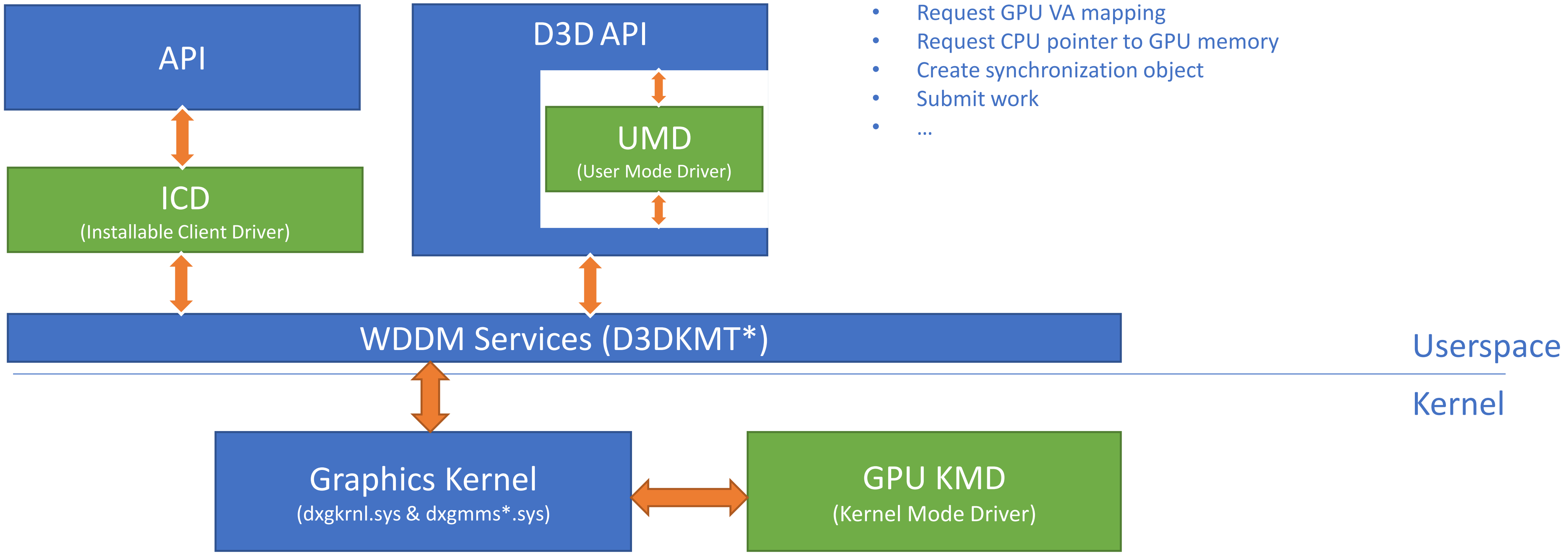
Bringing GPU to WSL

- We want to share the GPU(s) with the host
  - Not dedicated assignment
  - All host GPU available to WSL VM
  - Both host and WSL VM can submit work simultaneously to the GPU
- We want to flexibly manage the resources
  - No partition of video memory or fix scheduling quantum
  - Resource assignment based on applications need
- We want to enable a broad set of APIs
  - CUDA, OpenCL, OpenGL, and more
    - DirectX 12 is an implementation details that allow us to get there.

WDDM GPU Para-Virtualization (GPU-PV)

- WDDM – Windows Display Driver Model
  - Thin abstractions for the GPU that all graphics and compute APIs are based on
  - Abstract and manage GPU access for multiple clients
  - Think about it as DRM & KMS
- Para-Virtualization
  - Level of abstraction is the WDDM interface
  - Project the compute/rendering portion of the WDDM interface in a VM so driver can interact with it as if the GPU was local
- Was designed precisely for these usage scenarios
  - Windows Defender Application Guard for Edge
  - Windows Sandbox
  - Device Emulator (e.g. Hololens emulator)
- Extending to support Linux Guest, including WSL

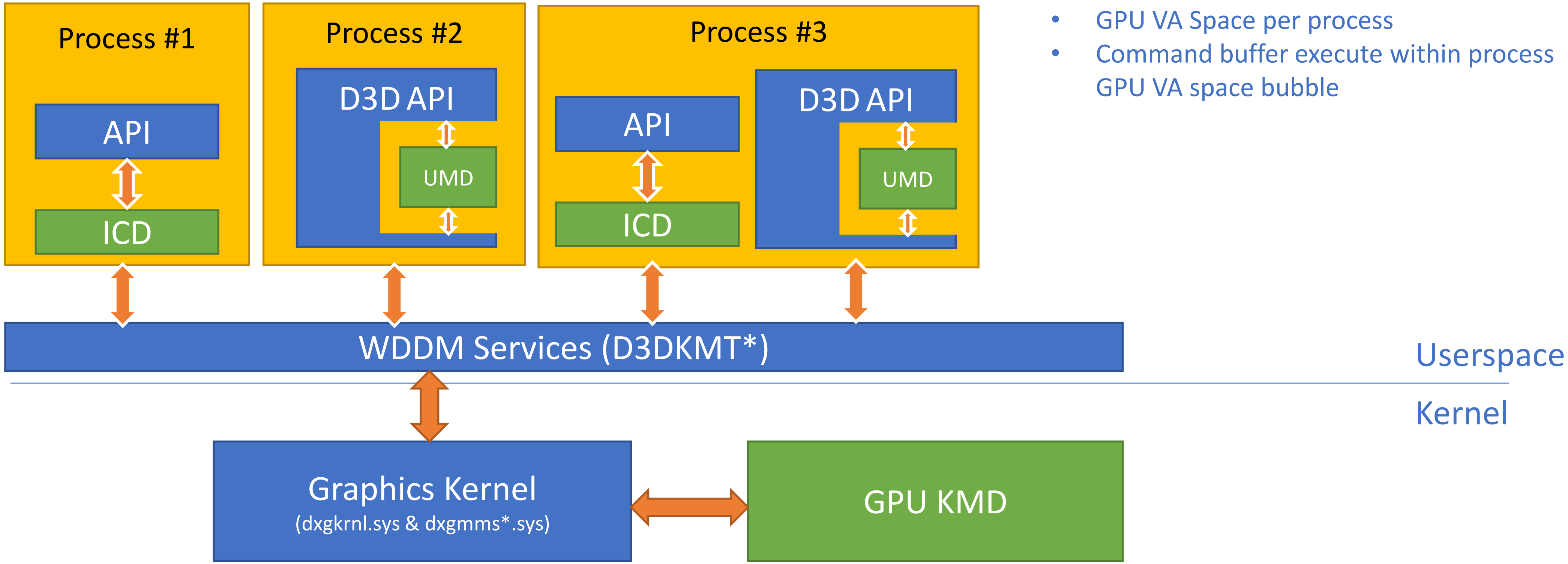
WDDM Architecture



WDDM Services low level and API agnostic

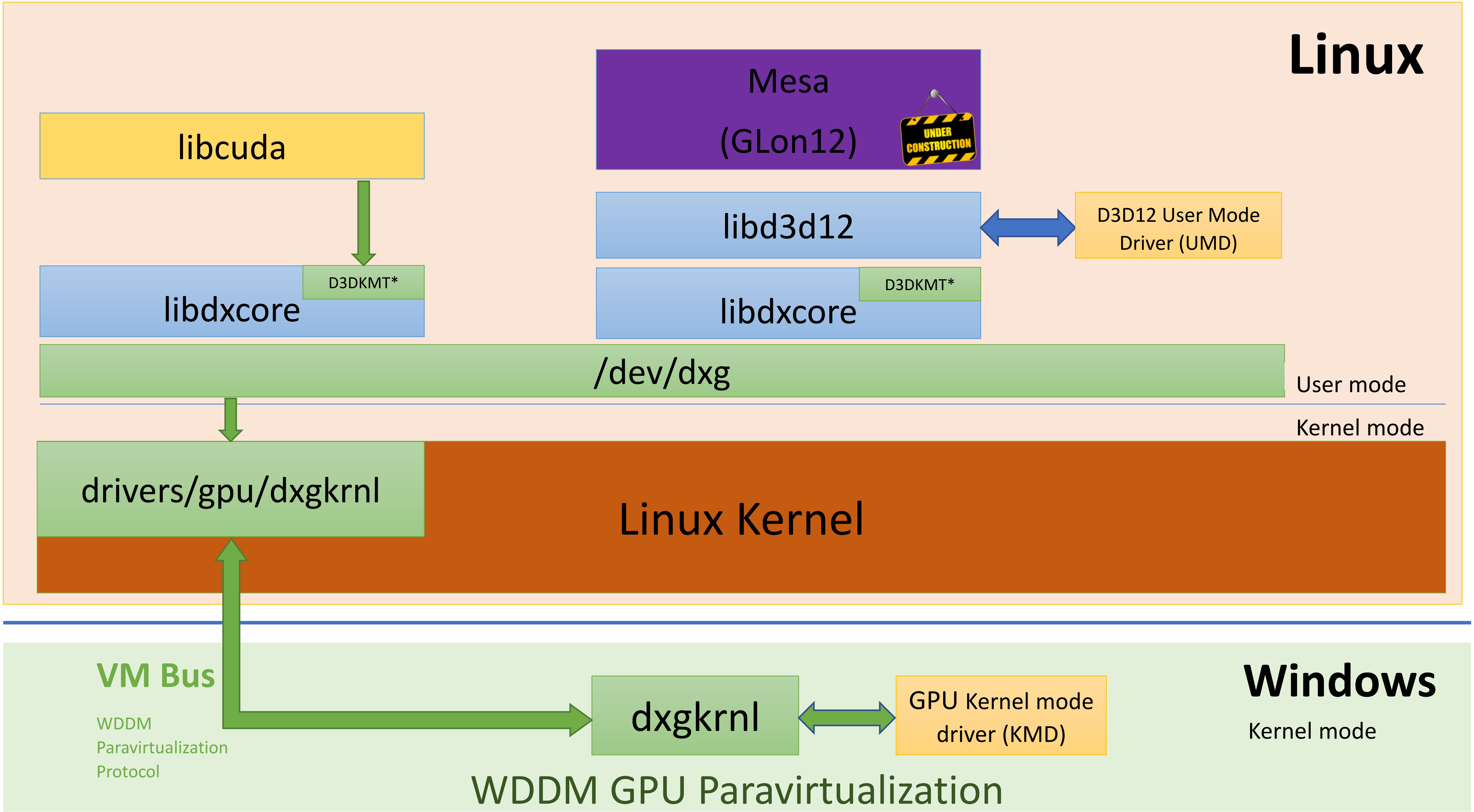
- Enumerate GPU
- Create Device
- Create Context / HwQueues
- Allocate GPU memory
- Request GPU VA mapping
- Request CPU pointer to GPU memory
- Create synchronization object
- Submit work
- …

WDDM Architecture



Isolation between processes

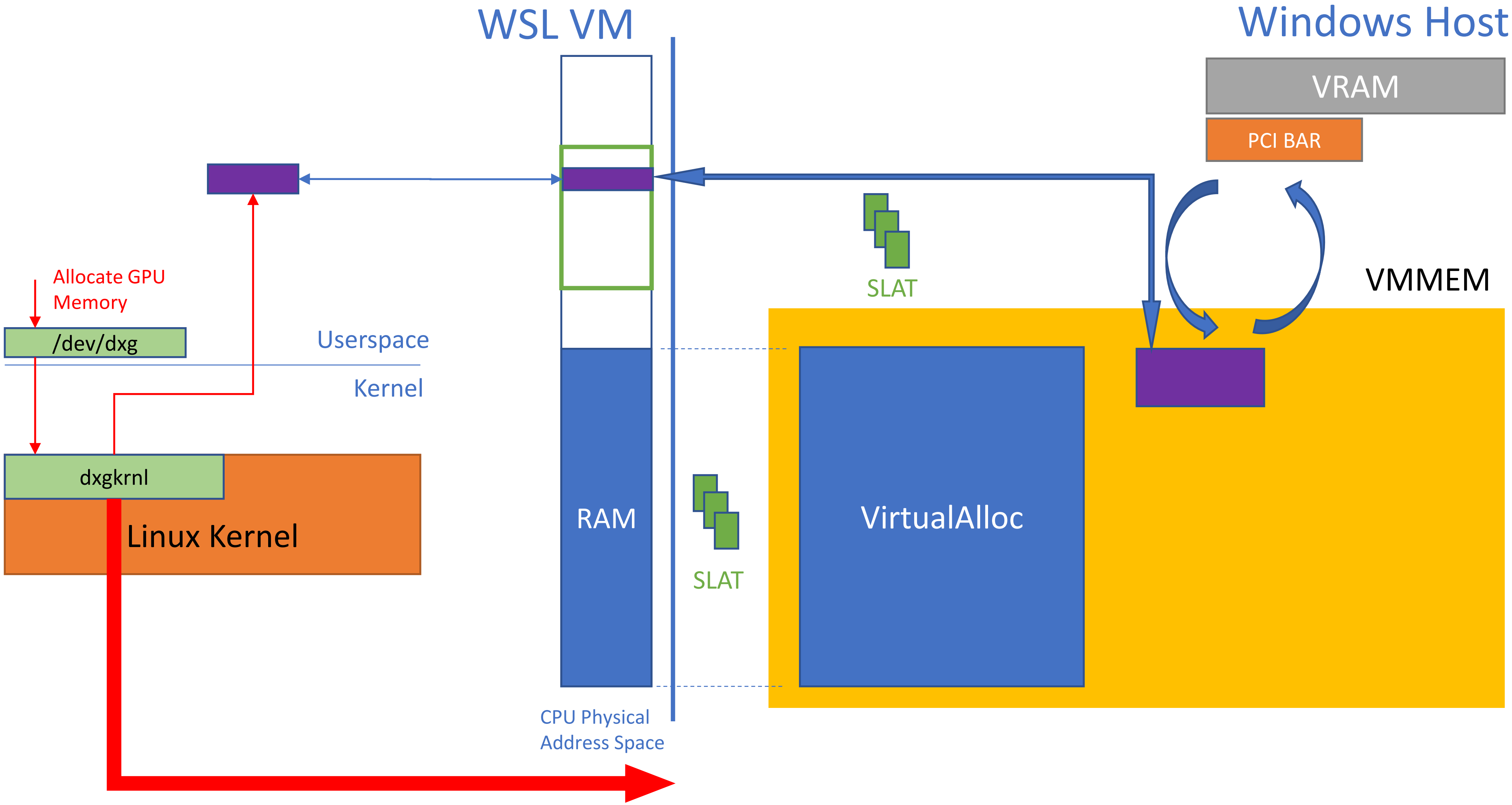
- GPU VA Space per process
- Command buffer execute within process
  GPU VA space bubble


Linux
Mesa
(GLon12)
UNDER CONSTRUCTION
libcuda
libd3d12
D3D12 User Mode Driver (UMD)
D3DKMT*
libdxcore
D3DKMT*
libdxcore
/dev/dxg
User mode
Kernel mode
drivers/gpu/dxgkrnl
Linux Kernel
VM Bus
WDDM Paravirtualization Protocol
dxgkrnl
GPU Kernel mode driver (KMD)
Windows
Kernel mode
WDDM GPU Paravirtualization

Dxgkrnl Linux Edition

- Open source
  - https://github.com/microsoft/WSL2-Linux-Kernel/tree/linux-msft-wsl-4.19.y/drivers/gpu/dxgkrnl
- Not a straight pass-through
  - Some WDDM API implemented locally
  - Some a combination of local and messages to the host
  - Fundamentally memory manager, scheduler and GPU are on the host
- No data copy
  - Only control information exchanged over VM bus
  - Data in command buffers or GPU surfaces shared between guest and host

Guest CPU Access to GPU Memory



WDDM 3.0

- Seamless support in WDDM3.0+
  - User mode driver compiled for Linux included in driver package
  - Host driver store mounted in Linux
  - Works out of the box
- Integrated into the Windows Driver Certification process
  - IHV Partner adding WSL 2 configured system to their test pool
  - HLK contains WSL 2 specific test validating driver

WSL Graphics Userspace

Goals

- Support breadth of existing Linux compute APIs
  - CUDA
  - OpenCL
  - Eventually graphics APIs like OpenGL/Vulkan too
- Minimize redundant/unnecessary work from driver vendors
- Support hardware-accelerated ML like TensorFlow

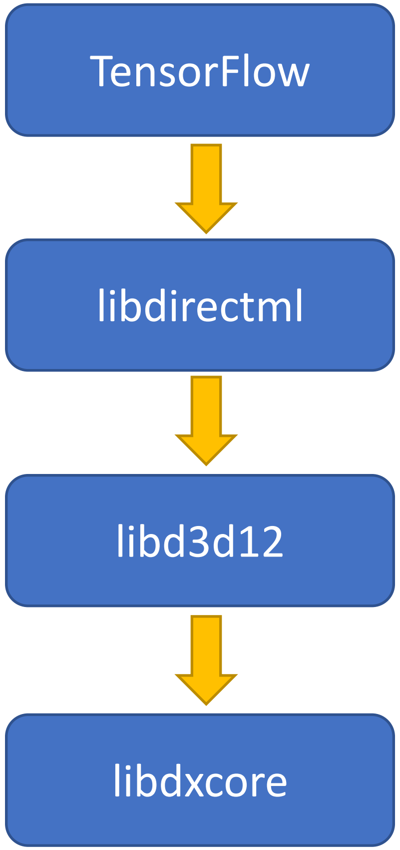
- NOT trying to introduce new competing APIs

How to get compute acceleration in WSL

- Two possible approaches
  - Ask driver vendors to port ICDs for APIs apps are using
  - Ask driver vendors to port UMD, we port D3D, we build layers to support APIs in terms of D3D
- ICD approach means continued asks on driver vendors for new APIs
  - E.g. 3+ APIs across 4+ vendors
- Mapping layer approach improves both Windows + WSL
  - 1 UMD per vendor, 1 mapping layer per API
  - Enables us to leverage DirectML as backend for ML frameworks
  - Mapping layers can be used to decrease vendor burden for supporting Windows
- Also possible for ICDs to be ported
  - CUDA in WSL works this way

What exists today

- DXCore
  - APIs for enumerating GPUs and querying properties
  - Similar role to DRM render nodes
- D3D12
  - Requires D3D12 UMD to be ported as well
  - UMDs available or in development from all Windows GPU vendors
- DirectML
  - Layer on top of D3D12 to provide highly optimized GPU-accelerated ML operators
- TensorFlow
  - Uses DirectML backend in WSL



What exists today - notes

- Compute-only functionality
  - Rasterization pipeline is available, but no swapchains / window integration
- Intention of D3D in WSL is implementation detail for GPU access
  - Not trying to introduce a new API for apps – no SDK planned
  - Added only to allow GPU access for higher level frameworks / APIs
- D3D stack is same code that runs on Windows
  - All components involved modified to dual-compile
  - Fixed lots of non-conformant code depending on MSVC quirks
  - Replaced Windows-specific constructs with cross-platform code
  - Wrote header shim with #defines/typedefs for things that come from Windows SDK
  - Clang caught several real bugs with its better warnings

What exists today – TensorFlow

- TensorFlow on DirectML
  - Runs on a wide variety of hardware
    - CUDA is NVIDIA
    - ROCm is a limited set of newer AMD hardware
  - Consistency/conformance
    - We test and work with all hardware vendors for consistent compute results
  - Easy to set up (just pip install tensorflow-directml)
- https://github.com/microsoft/tensorflow-directml
  - Working closely with the TensorFlow community to bring this feature upstream so that it's available in the official build of TensorFlow going forward

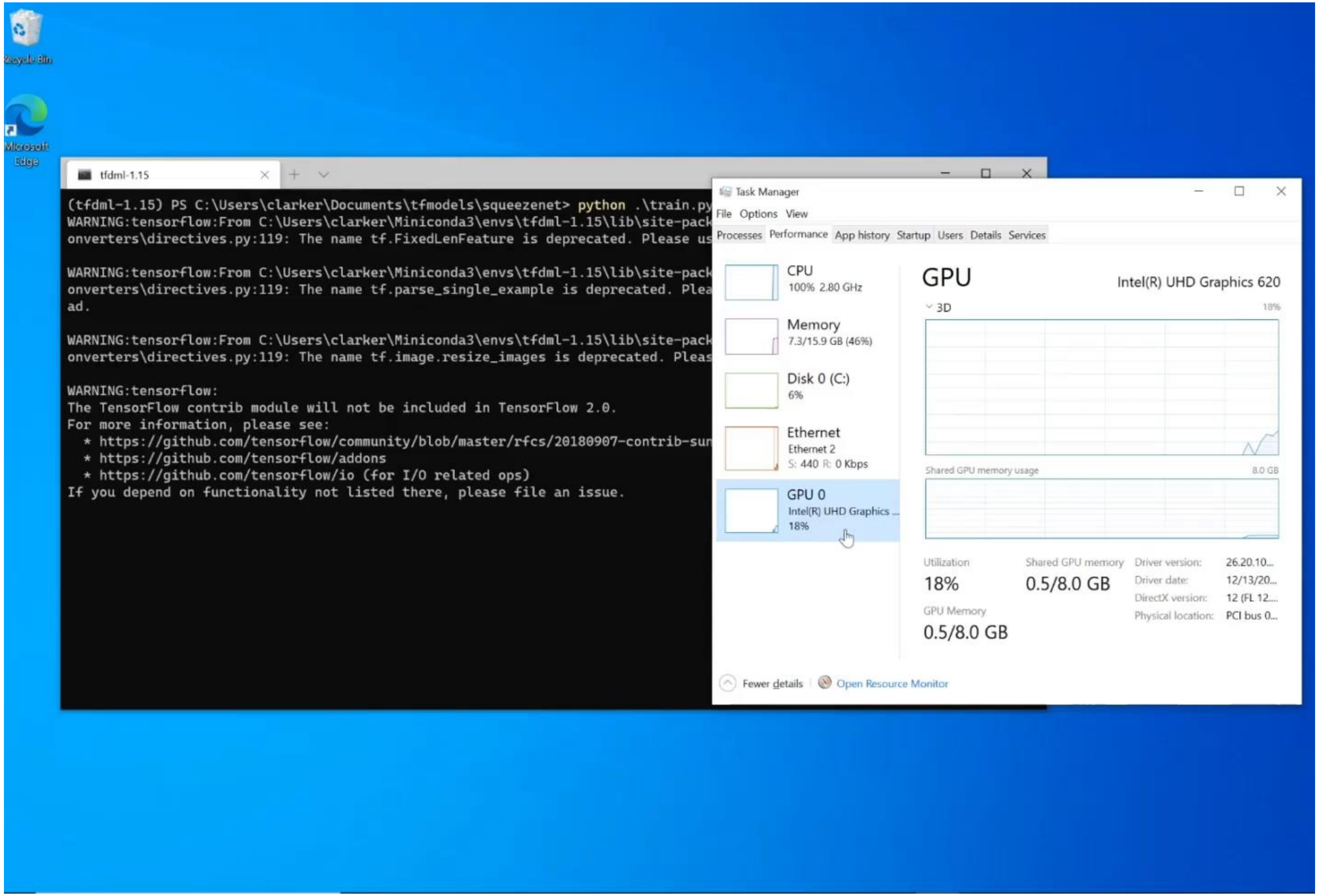
Shipping as binaries

- Attempting to be distro-agnostic
  - Both Microsoft code (D3D, DML) and WSL drivers
- Note quite statically-linked, but close
  - Only external dependencies on libc
  - C++ runtime and other dependencies included
  - No exceptions crossing module boundaries
  - Technically linking against musl in our build, but in a glibc-compatible way

What’s in the works

- OpenCLOn12
- OpenGLOn12
  - Both leveraging Mesa
  - Both currently working on Windows – WSL efforts not yet started
  - OpenGL requires solving window integration: hard problem
    - Lots of open design questions, not intrinsically hard due to Mesa/GL
    - Some work here underway – see later talk for non-accelerated WSL window integration: “X11 and Wayland Applications in WSL”

Demo



Recycle Bin
Microsoft Edge
tfdml-1.15
(tfdml-1.15) PS C:\Users\clarker\Documents\tfmodels\squeezenet> python .\train.py
WARNING:tensorflow:From C:\Users\clarker\Miniconda3\envs\tfdml-1.15\lib\site-pack
onverters\directives.py:119: The name tf.FixedLenFeature is deprecated. Please us
WARNING:tensorflow:From C:\Users\clarker\Miniconda3\envs\tfdml-1.15\lib\site-pack
onverters\directives.py:119: The name tf.parse_single_example is deprecated. Plea
ad.
WARNING:tensorflow:From C:\Users\clarker\Miniconda3\envs\tfdml-1.15\lib\site-pack
onverters\directives.py:119: The name tf.image.resize_images is deprecated. Pleas
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
* https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sun
* https://github.com/tensorflow/addons
* https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.
Task Manager
File Options View
Processes Performance App history Startup Users Details Services
CPU
100% 2.80 GHz
Memory
7.3/15.9 GB (46%)
Disk 0 (C:)
6%
Ethernet
Ethernet 2
S: 440 R: 0 Kbps
GPU 0
Intel(R) UHD Graphics ...
18%
GPU
Intel(R) UHD Graphics 620
3D
18%
Shared GPU memory usage
8.0 GB
Utilization
18%
Shared GPU memory
0.5/8.0 GB
GPU Memory
0.5/8.0 GB
Driver version: 26.20.10...
Driver date: 12/13/20...
DirectX version: 12 (FL 12...
Physical location: PCI bus 0...
Fewer details
Open Resource Monitor

More info / how to try it out

- https://aka.ms/gpuinwsldocs