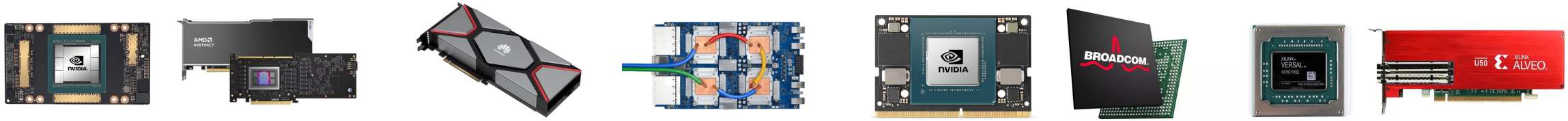
# **XSched**: Preemptive Scheduling for Diverse XPUs

**Weihang Shen** Mingcong Han Jialong Liu Rong Chen Haibo Chen

IPADS, Shanghai Jiao Tong University


# XPUs are Widely Adopted

GPUs NPUs TPUs ASICs FPGAs …




*Typical scenarios:*

Cloud Services

Autonomous Vehicles

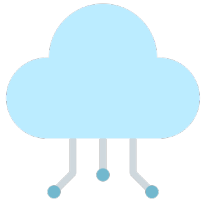
Copilot + PC

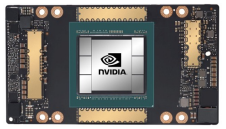
Edge Devices

# XPU Multitasking is Common

**Cloud**: improve utilization, save cost

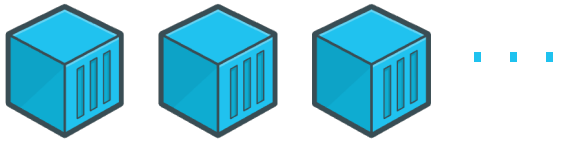
*One XPU can serve tens of job instances*





GPU/NPU Cloud

Serverless Functions



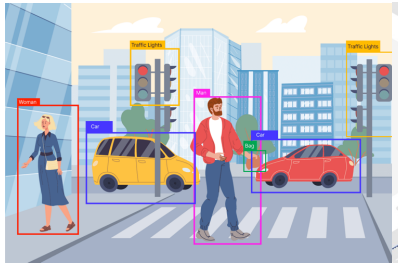
Tenants / Container Jobs

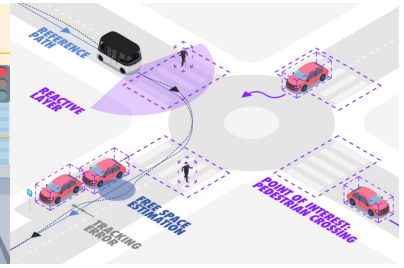
**Edge**: limited XPU resources

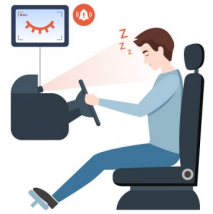
*30+ tasks are deployed on 1~2 Orin GPU*

NV
Jetson Orin

Autonomous
Vehicles



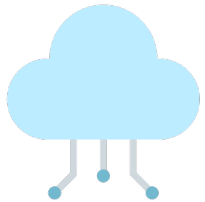


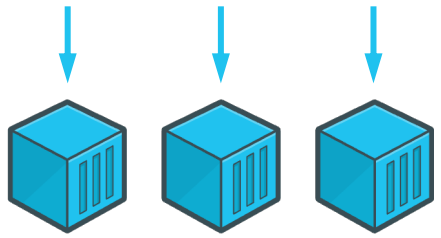
Perception  Planning & Control  Fatigue Detection

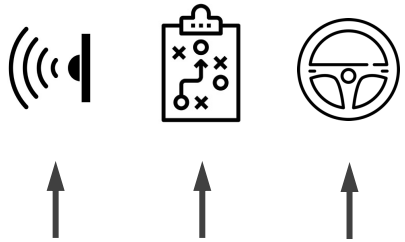
# Diverse Scheduling Requirements of Apps

SLO / DDL



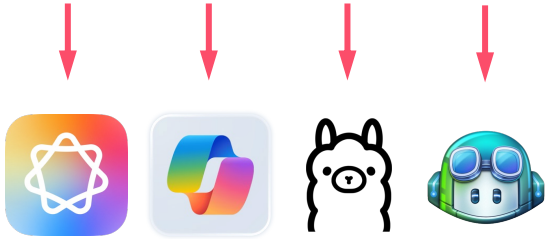
Fairness / Throughput





Realtime Latency

Responsiveness



# Diverse Scheduling Requirements of Apps





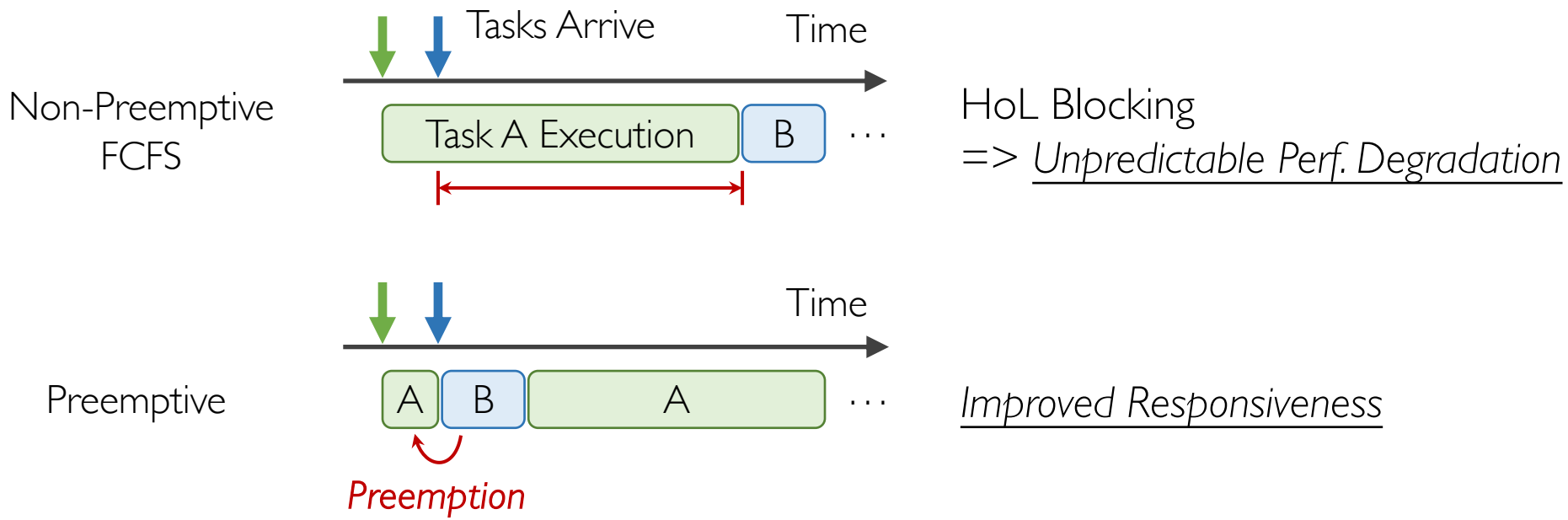
Realtime Latency

*Task preemption* is a crucial mechanism to meet *diverse* application needs

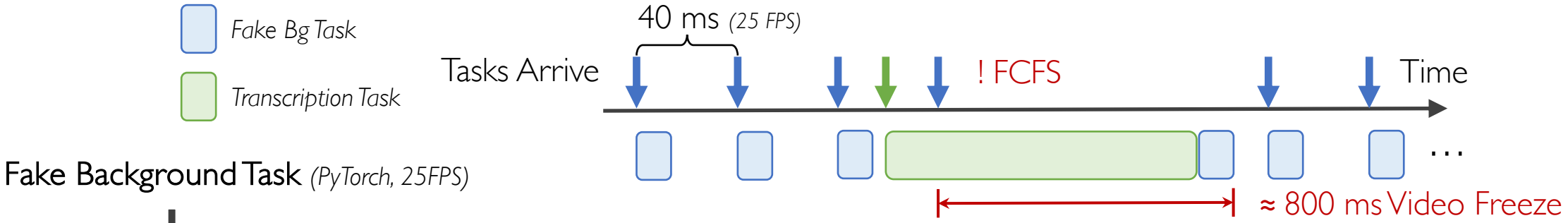
# Issues of Native Hardware Scheduling

## #1 Most XPUs *lack preemption* support

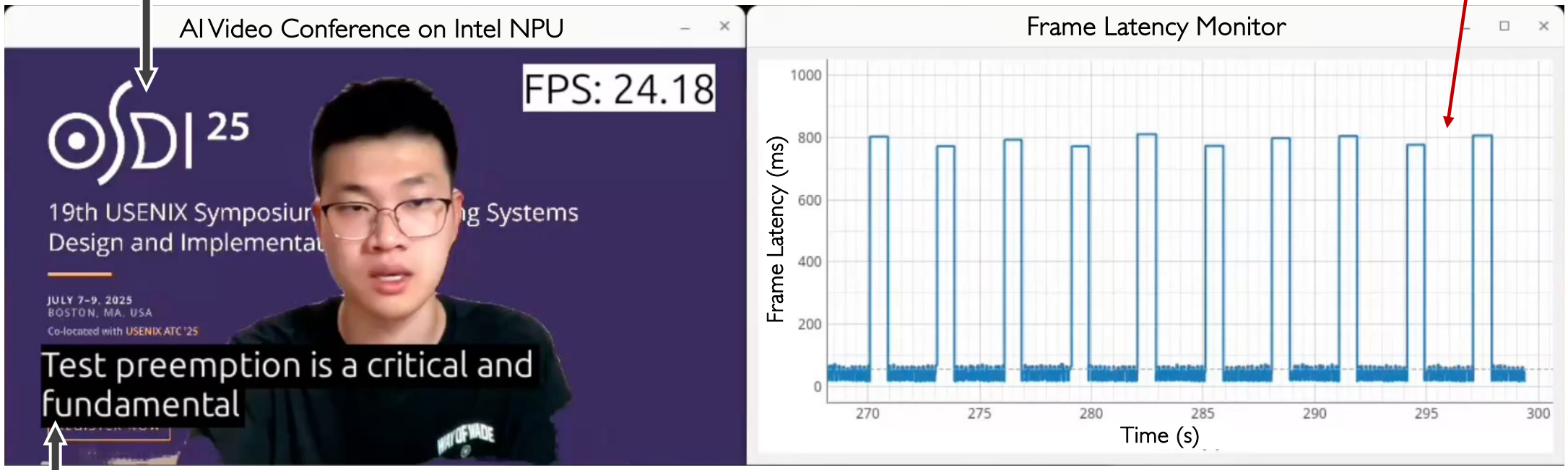
Non-preemptive FCFS policy is naturally embedded in the HW, FW, and drivers.



# Example: AI Conference on Intel NPU




Fake Background Task *(PyTorch, 25FPS)*

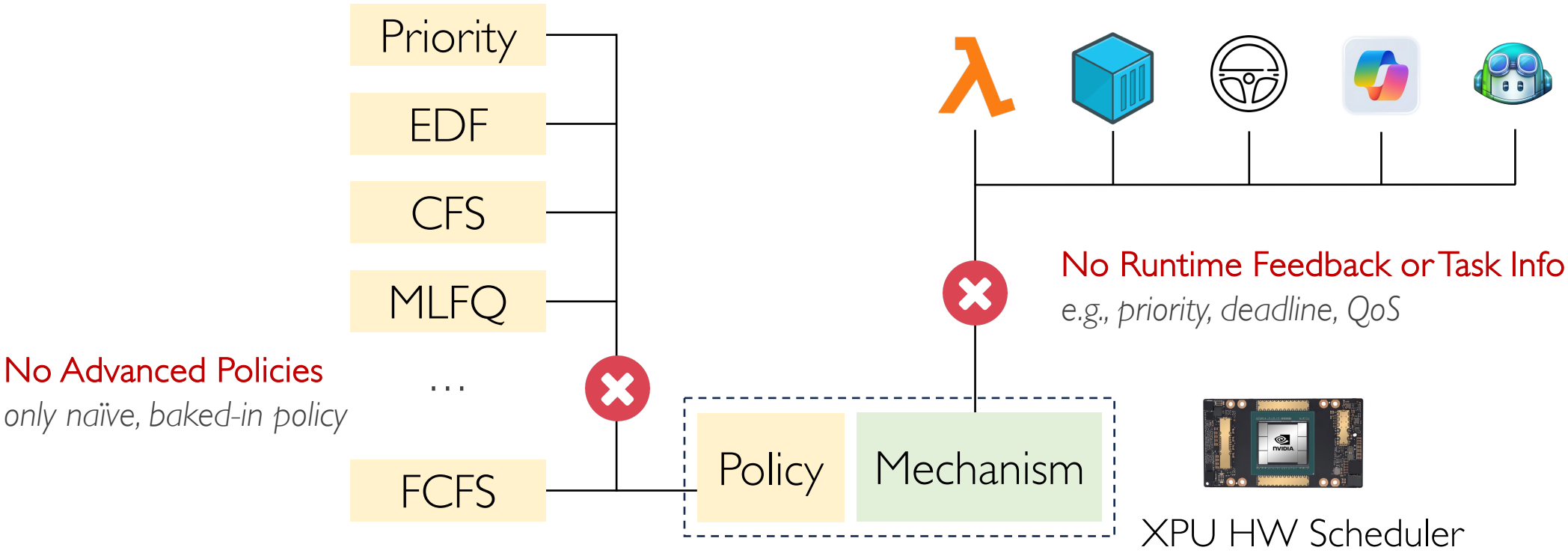



Transcription Task *(whisper.cpp, every 3s)*

**Tested on Intel Core Ultra 185H NPU, Driver Version 1.5.1*

# Issues of Native Hardware Scheduling

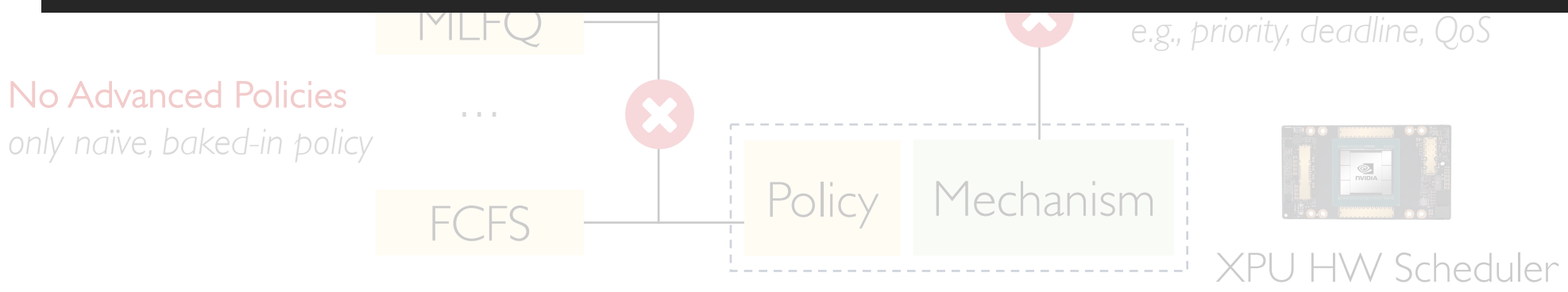
## #2 Hardware schedulers are *not flexible*



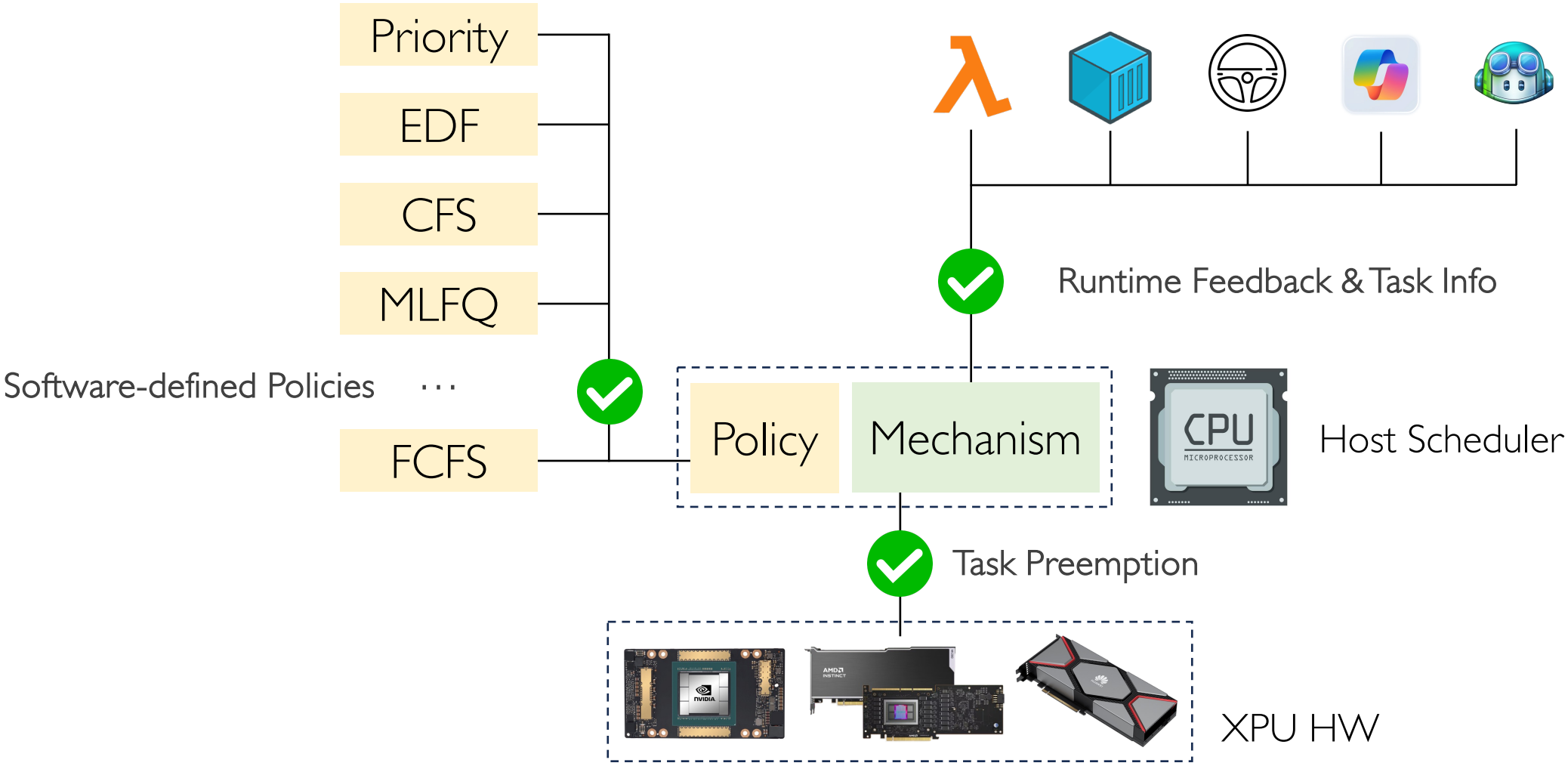
# Issues of Native Hardware Scheduling

#2 Hardware schedulers are *not flexible*

XPU hardware scheduling
*mismatches* application requirements



# Approach: Host-Managed Scheduling



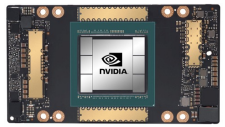
# Existing Solutions are not GENERAL

EDF CFS SJF STCF MLFQ …

#3 Not Uniform

*hinder hardware-agnostic policies & cooperative scheduling*

Existing Solutions, *e.g., Effisha [PPoPP'17], FLEP [ASPLOS'17], REEF [OSDI'22], …*



#1 Not Portable

*designed for certain GPU, while new XPUs emerging*

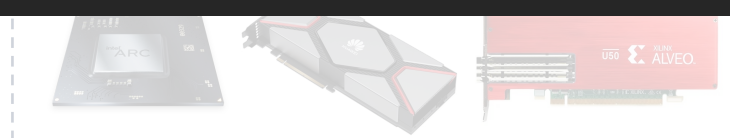
#2 Not Evolvable

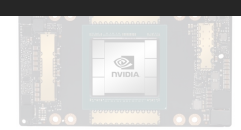
*for new / deprecated HW features, while XPUs keep advancing*

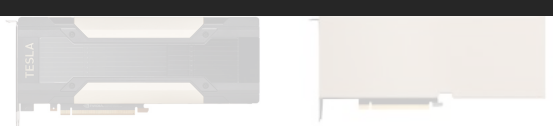
# Existing Solutions are not GENERAL

EDF CFS SJF STCF MLFQ ···

#3 Not Uniform
*hinder hardware-agnostic policies & cooperative scheduling*







#1 Not Portable
*designed for certain GPU, while new XPUs emerging*

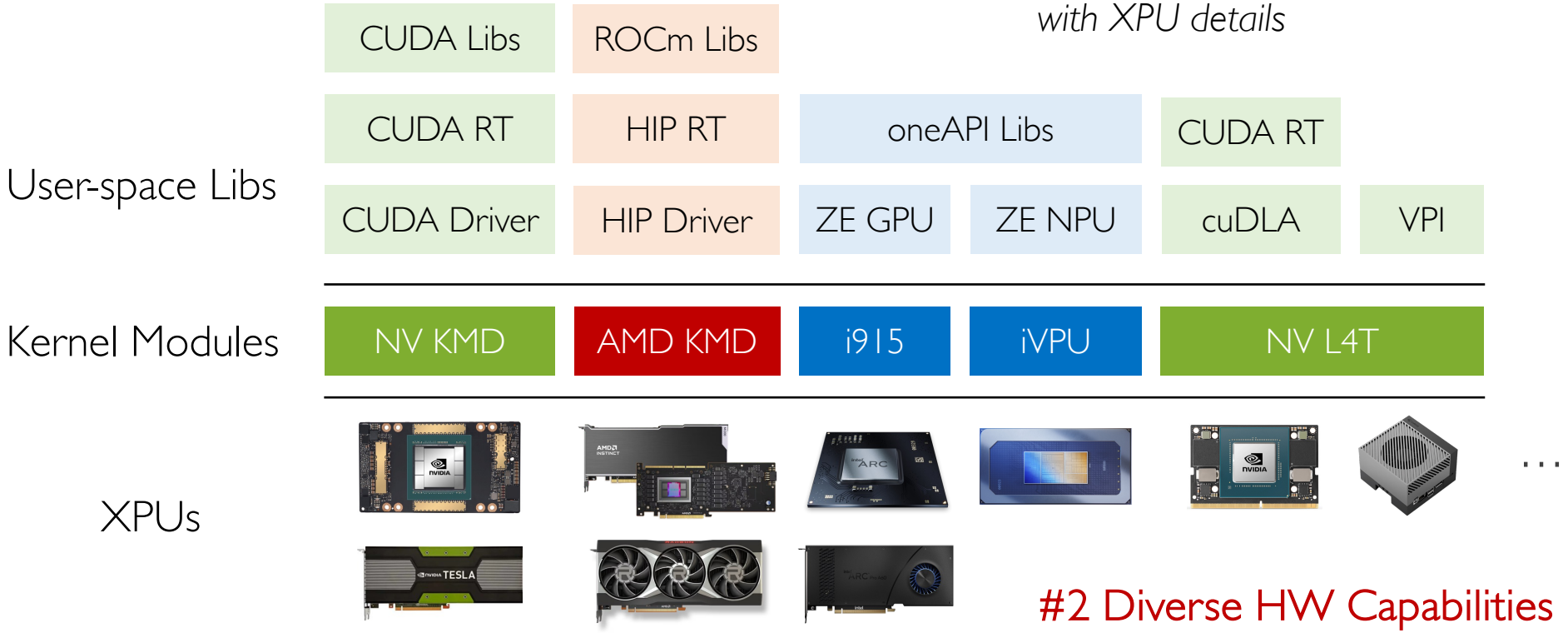
#2 Not Evolvable
*for new / deprecated HW features, while XPUs keep advancing*

**Our Goal: *GENERAL* & *PREEMPTIVE***
**Host-managed XPU Scheduling**

# Challenges for General XPU Scheduling

**#1 Complex SW Stack**

*scheduling could be tightly coupled with XPU details*

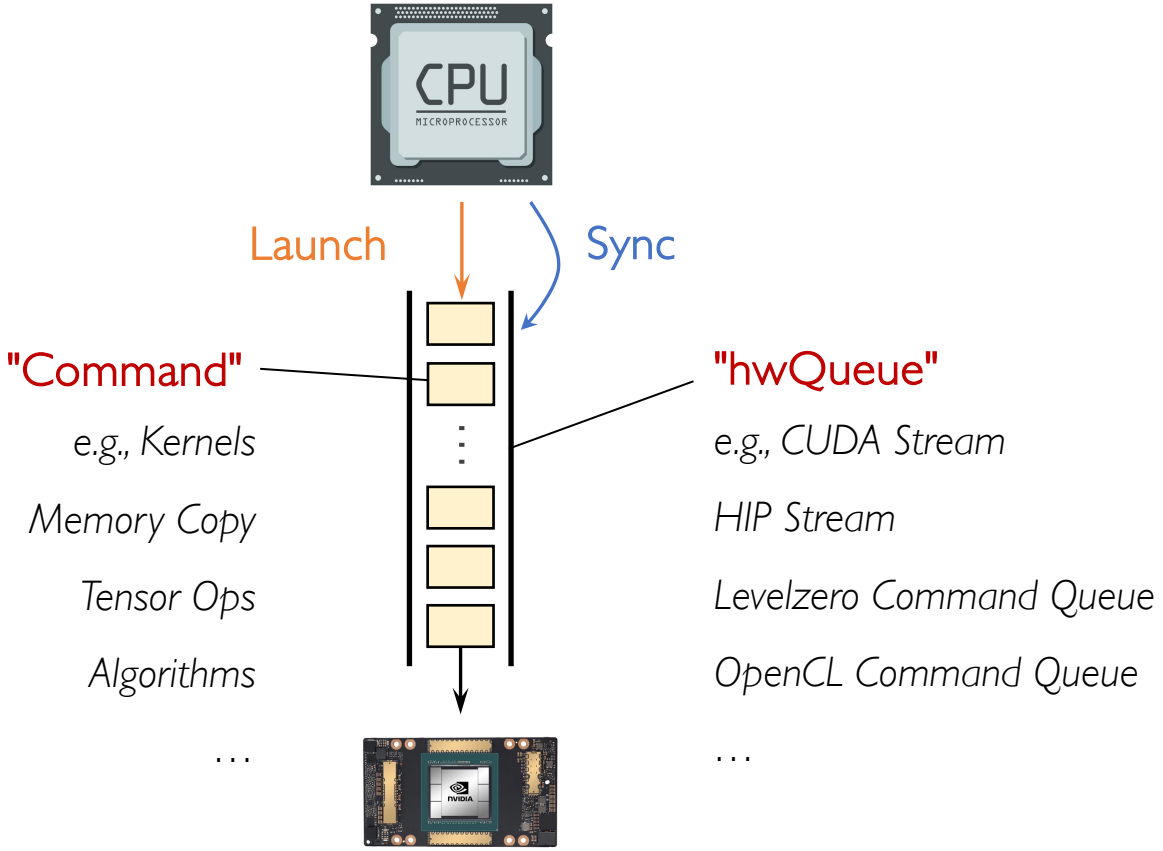



**#2 Diverse HW Capabilities**

*scheduling could rely on specific functionalities e.g., programmability, CU reset, timeslice control*

# Opportunity: XPU Programming Paradigm

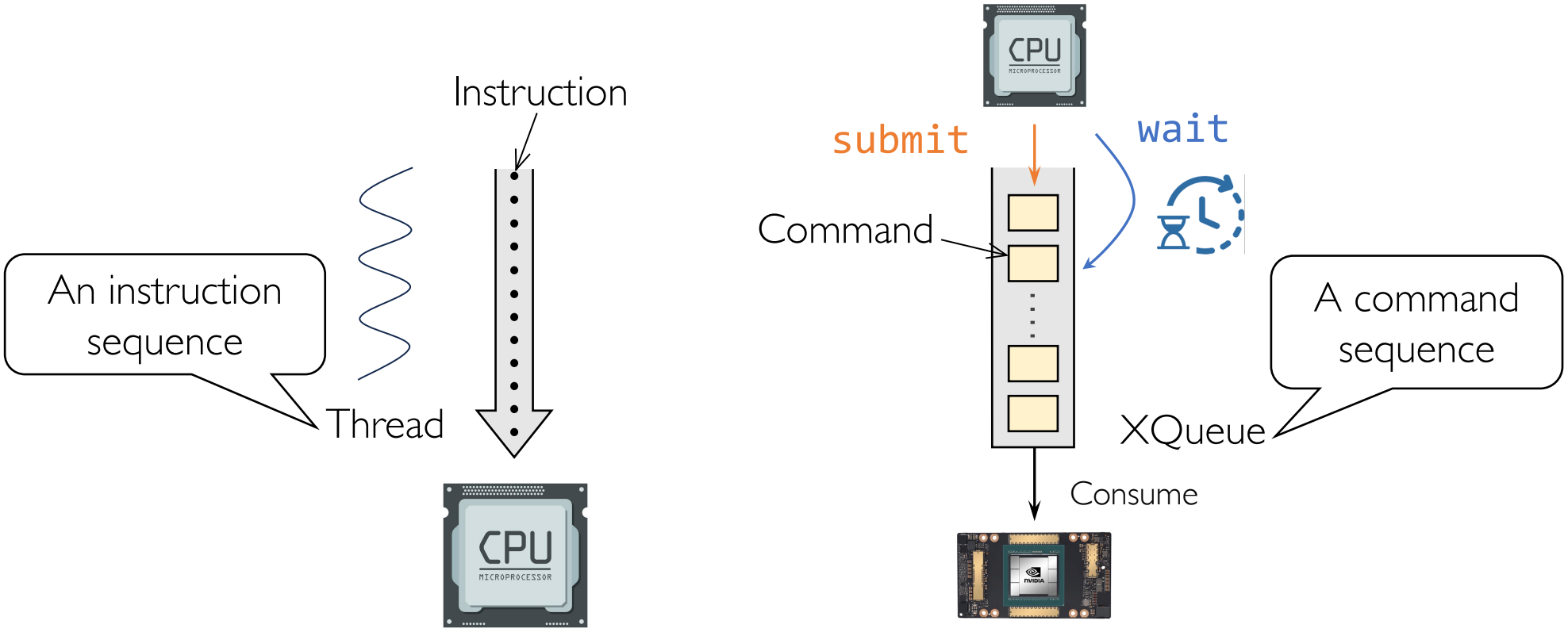
```
def infer_task():
  memcpyH2D(…) # memcpy cmd
  conv(…)      # kernel cmds
  relu(…)
  ...
  softmax(…)
  memcpyD2H(…) # memcpy cmd
  sync(…) # wait for completion
```



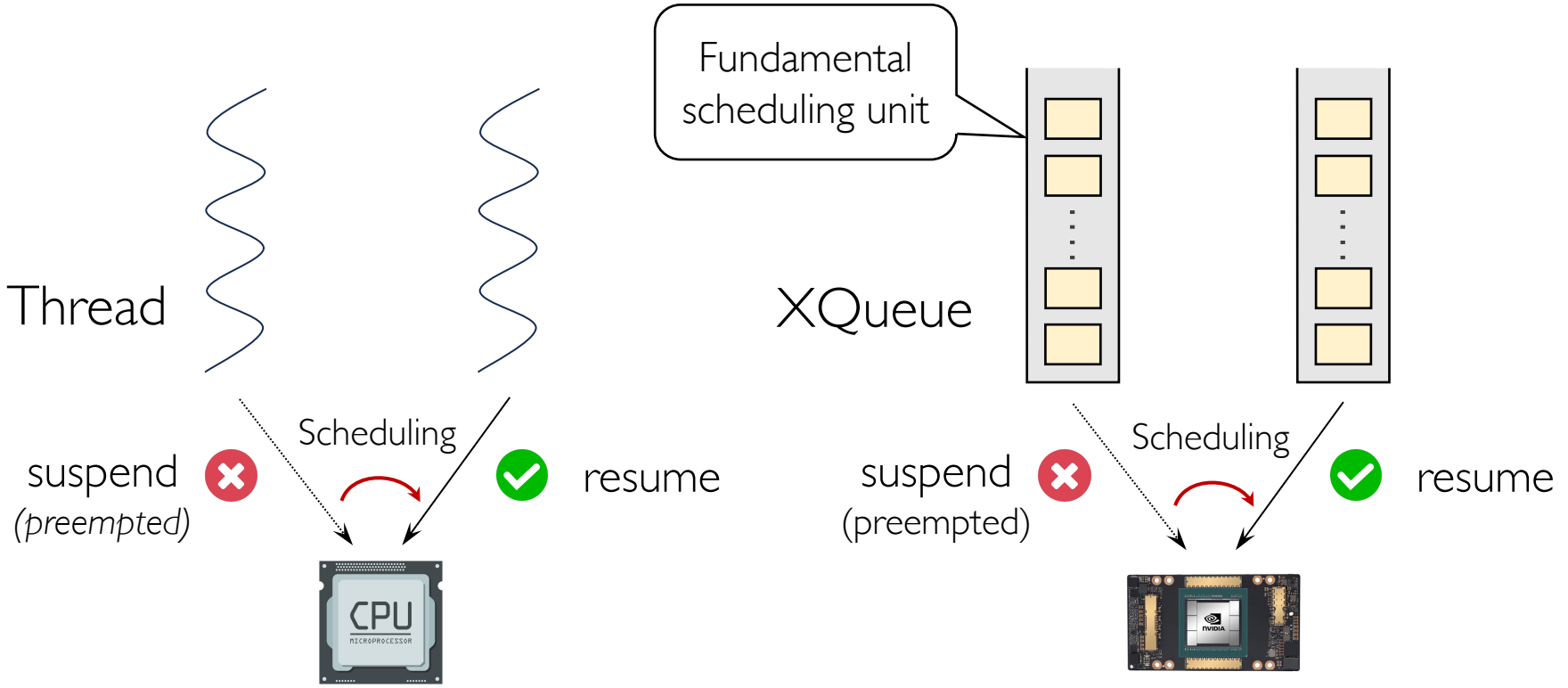
# Opportunity: XPU Programming Paradigm


An XPU task can be abstracted as a sequence of *commands* executed on a *command queue*

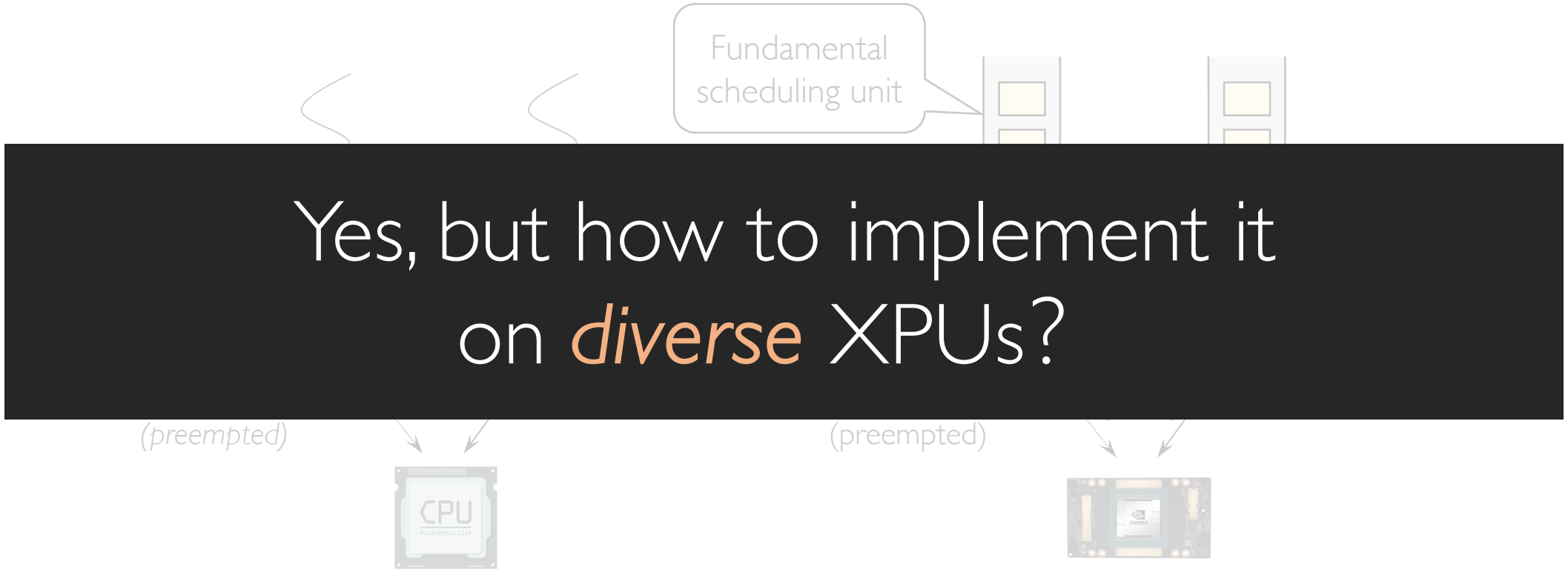
# XQueue Abstraction: XPU Command Queue



# XQueue: *Preemptible* XPU Command Queue

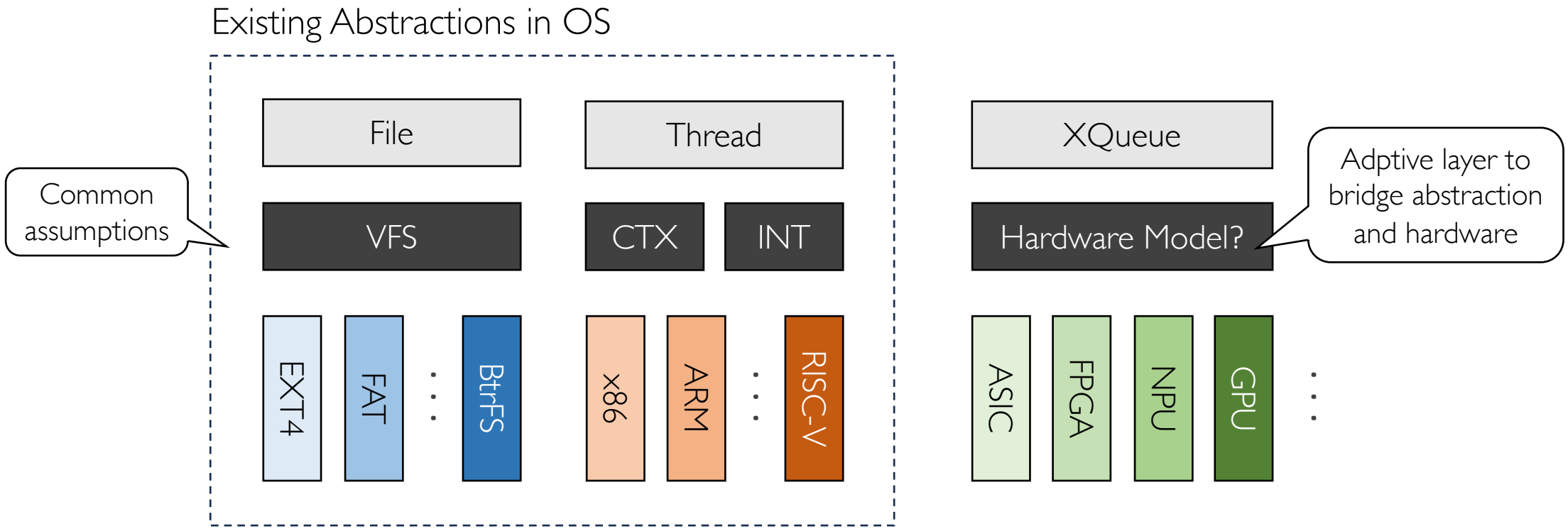


# XQueue: *Preemptible* XPU Command Queue

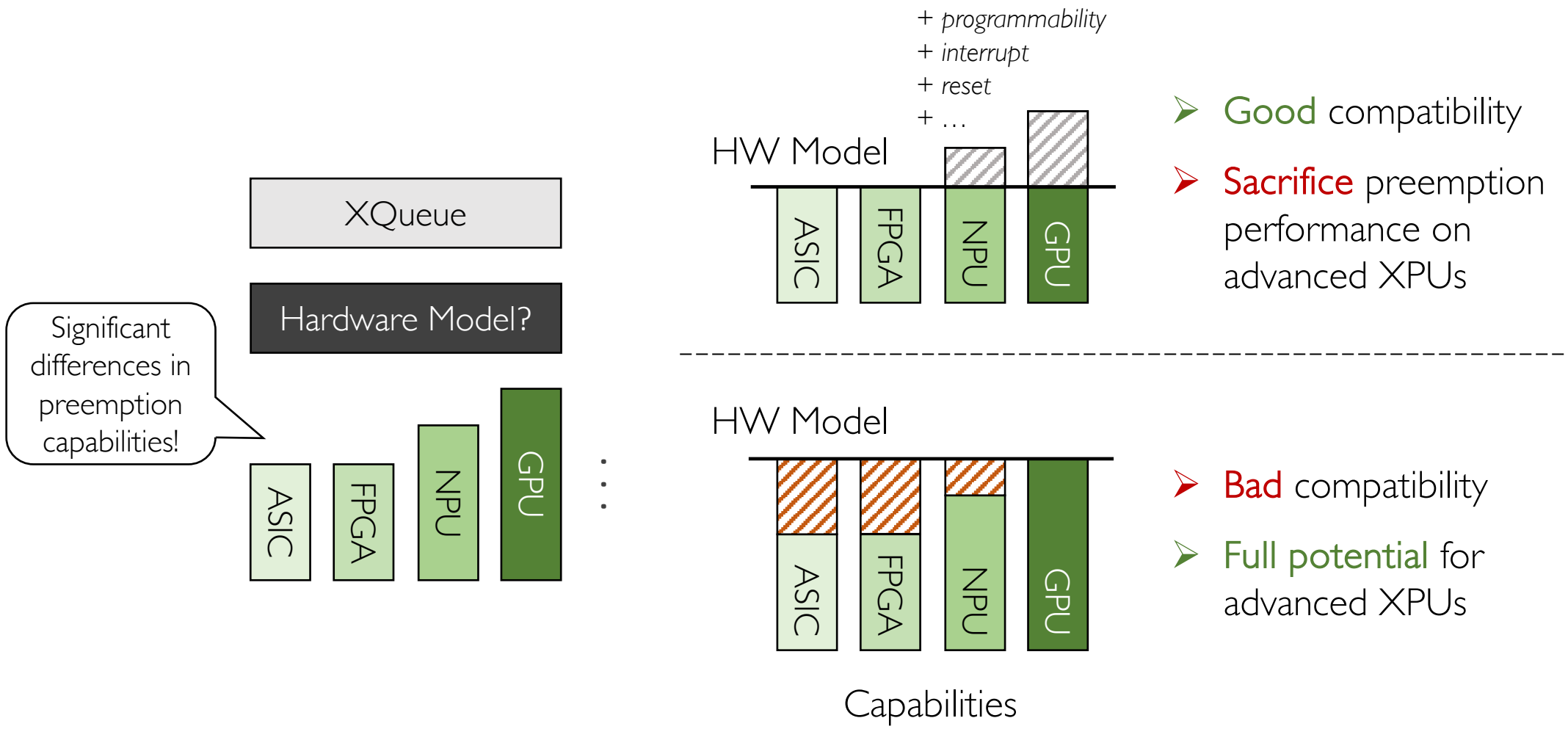



Yes, but how to implement it
on *diverse* XPUs?

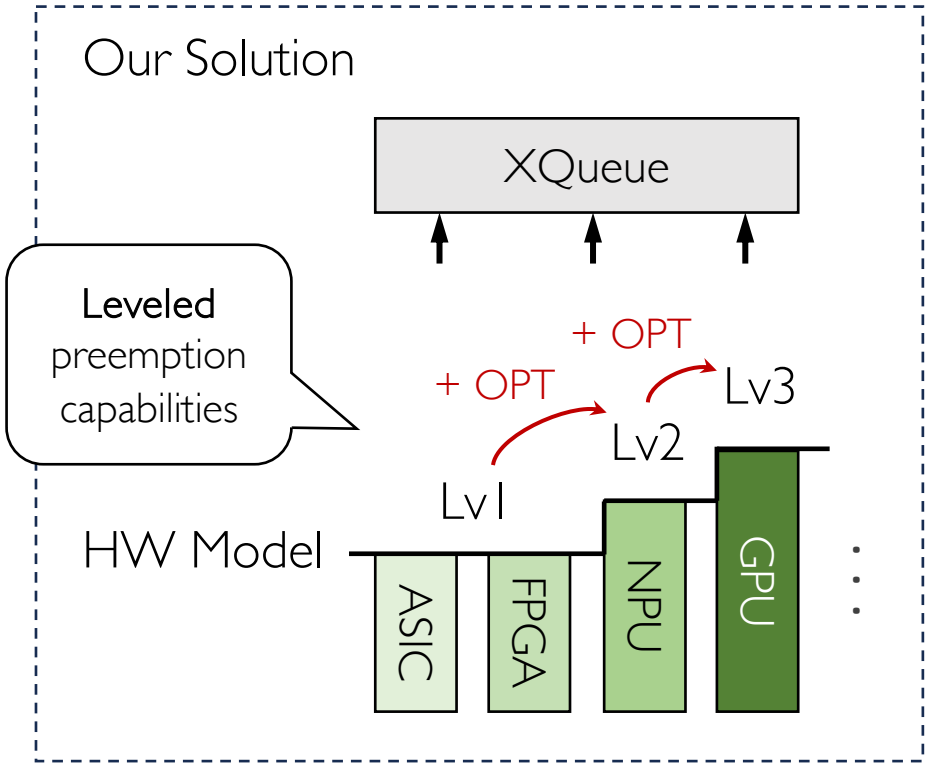
# Implementing XQueue on Diverse XPUs



# Dilemma: Compatibility or Performance?

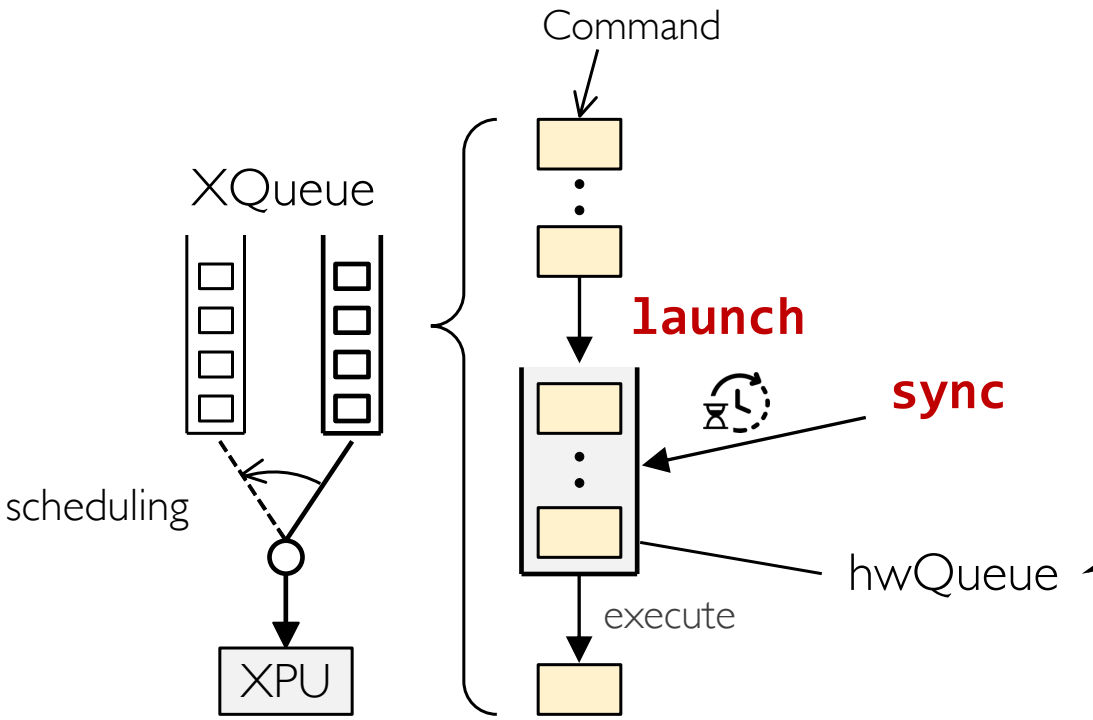


# Our Solution: Multi-level Hardware Model



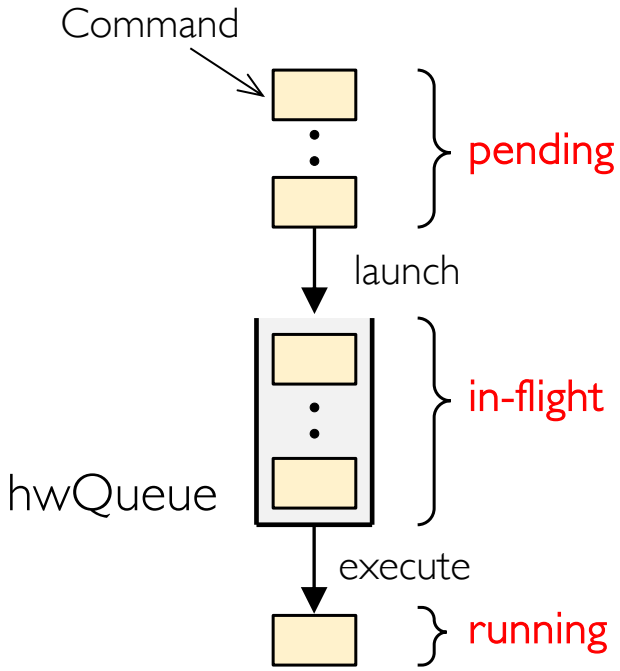

- Categorize preemption-related capabilities into three incremental levels (Lv1—Lv3)
- Higher levels posses all capabilities of lower ones and introduce advanced capabilities
- The lowest level (Lv1) is supported by ALL XPUs
- The highest level (Lv3) unlocks full potentials of advanced XPUs

# Basic Assumption: hwQueue




XPUs should provide hwQueues with basic **launch** and **sync** interface to support task execution

# From the Perspective of Command States




- **Pending** commands

  *Created and waiting on the host side*

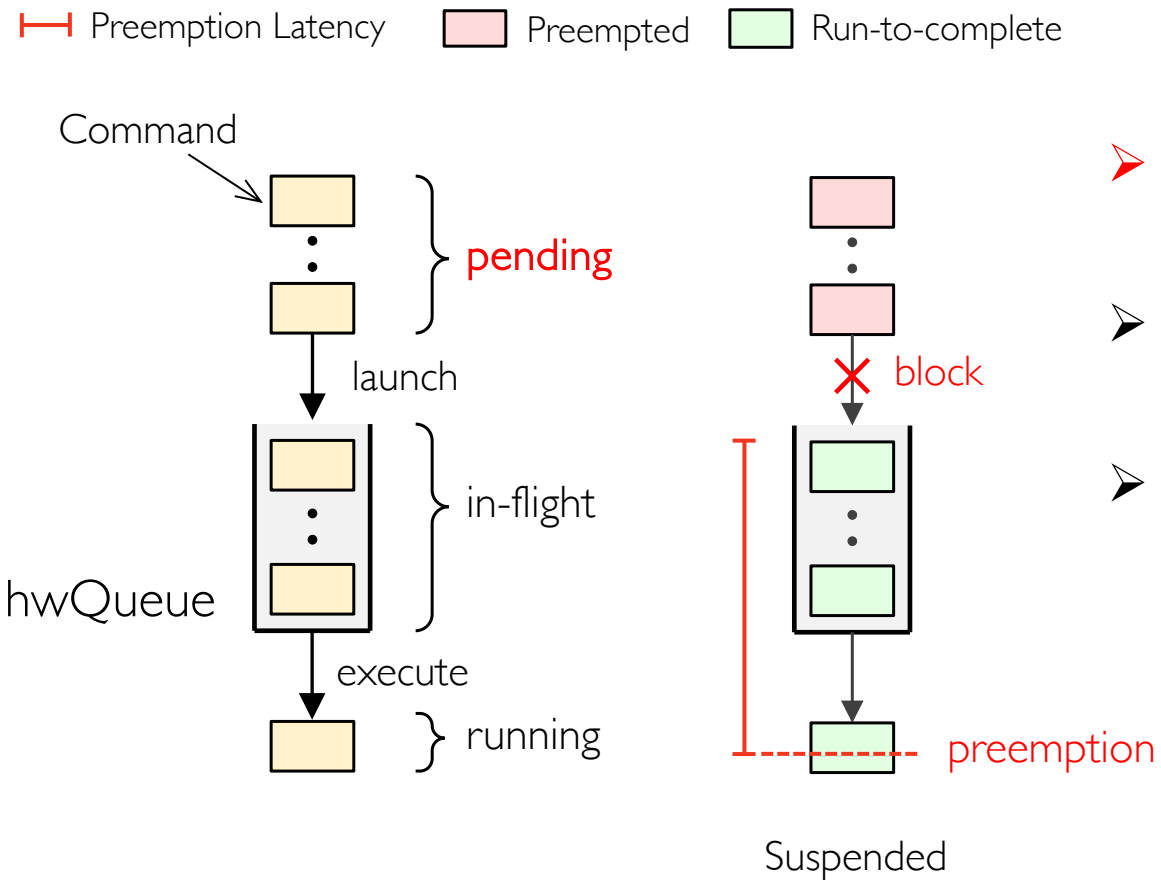
- **In-flight** commands

  *Launched to a hwQueue, escapes host control*

- **Running** commands

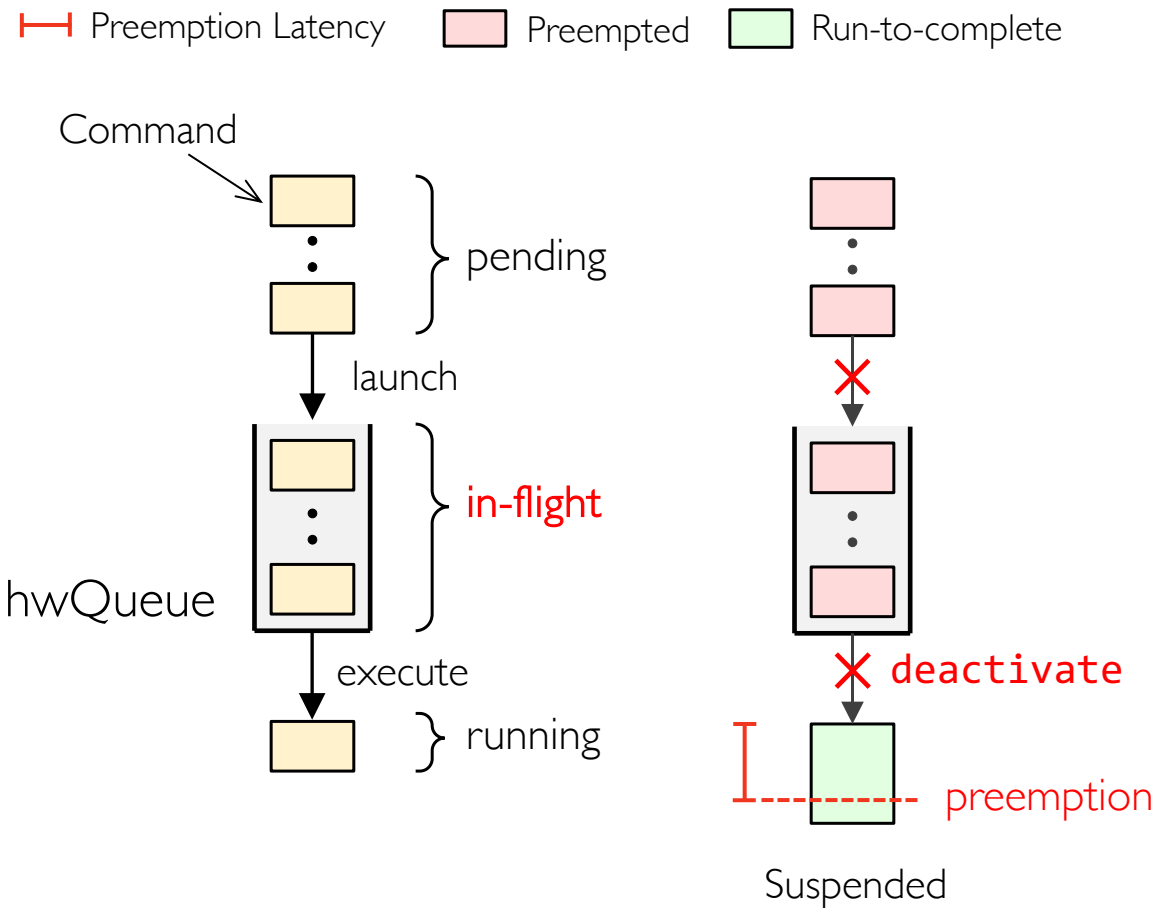
  *Being executed on the XPU*

# Level-1: Preempt Pending Command



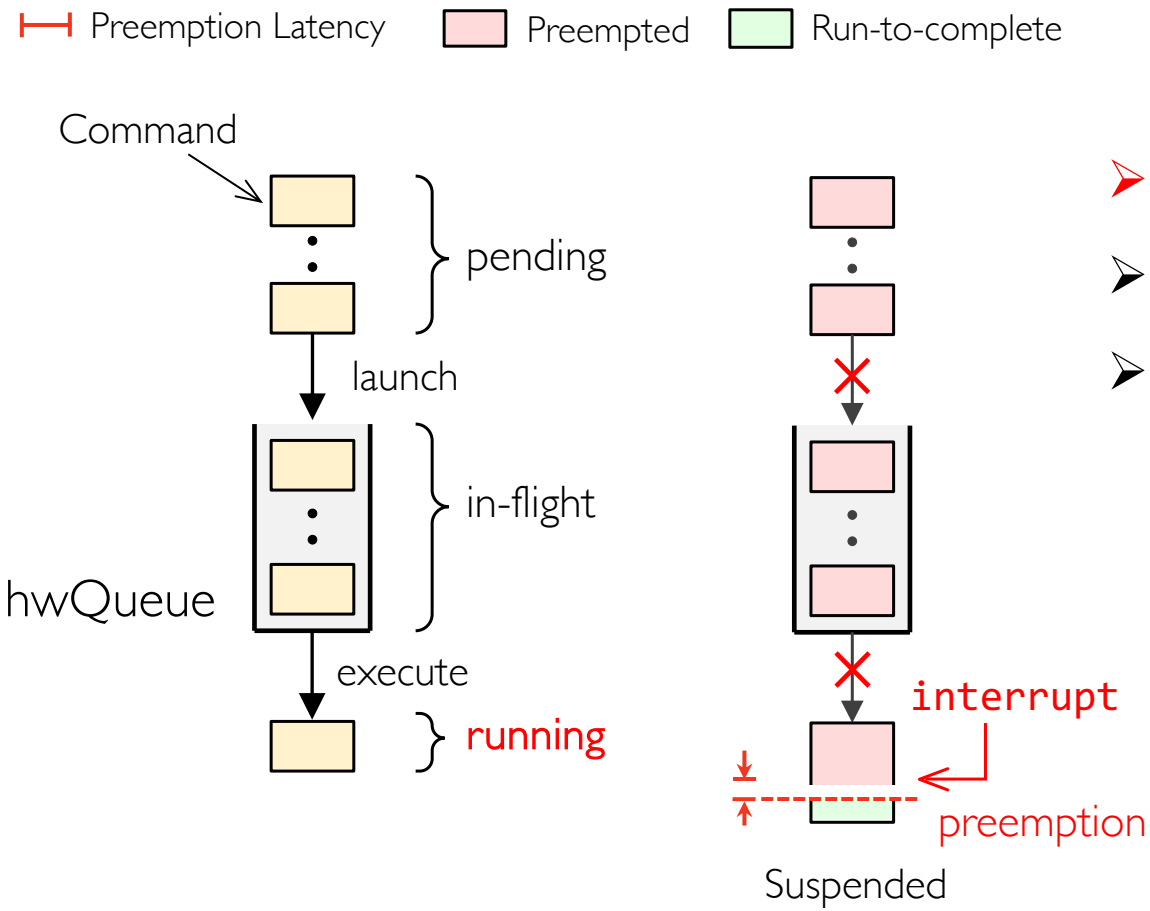

- **Block** host CPU from launching new commands, no hardware requirements
- Preemption done until in-flight and running commands are finished
- OPT: Progressively launches the pending commands and limits the number of in-flight commands to reduce preemption latency

# Level-2: Preempt In-flight Command




- **Deactivate** an hwQueue so that its commands will not be executed further
- Significantly reduced preemption latency
- Approach 1: instruct the *μ-controllers* in modern XPUs, e.g., *Intel NPU, NV & AMD GPU* to stall command dispatching
- Approach 2: leverage command *programmability* to abort in-flight commands

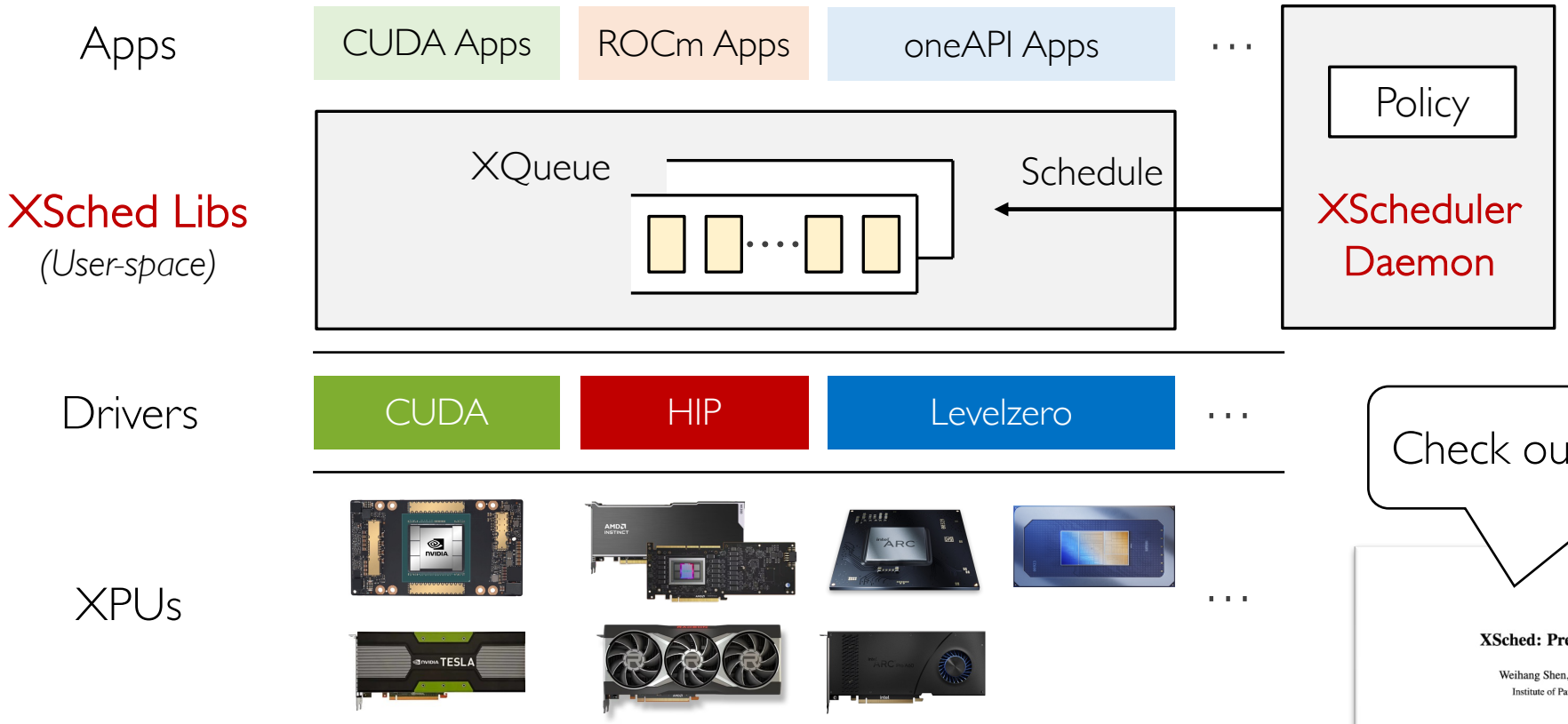
# Level-3: Preempt Running Command



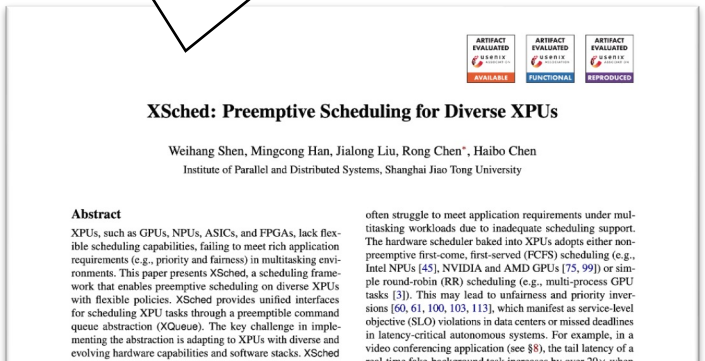

- Interrupt the running command
- Ultra-low & stable preemption latency
- Supported in modern GPUs

# XSched: A Preemptive XPU Scheduling Framework

*based-on the XQueue Abstraction and the Multi-level Hardware Model*




Check our paper for details!



XSched: Preemptive Scheduling for Diverse XPUs

Weihang Shen, Mingcong Han, Jialong Liu, Rong Chen*, Haibo Chen

Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University

**Abstract**

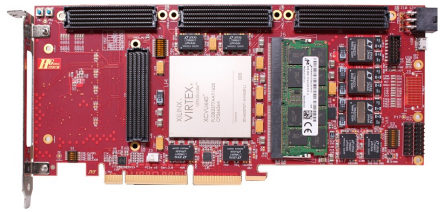
XPUs, such as GPUs, NPUs, ASICs, and FPGAs, lack flexible scheduling capabilities, failing to meet rich application requirements (e.g., priority and fairness) in multitasking environments. This paper presents XSched, a scheduling framework that enables preemptive scheduling on diverse XPUs with flexible policies. XSched provides unified interfaces for scheduling XPU tasks through a preemptible command queue abstraction (XQueue). The key challenge in implementing the abstraction is adapting to XPUs with diverse and evolving hardware capabilities and software stacks. XSched often struggle to meet application requirements under multitasking workloads due to inadequate scheduling support. The hardware scheduler baked into XPUs adopts either non-preemptive first-come, first-served (FCFS) scheduling (e.g., Intel NPUs [45], NVIDIA and AMD GPUs [75, 99]) or simple round-robin (RR) scheduling (e.g., multi-process GPU tasks [3]). This may lead to unfairness and priority inversions [60, 61, 100, 103, 113], which manifest as service-level objective (SLO) violations in data centers or missed deadlines in latency-critical autonomous systems. For example, in a video conferencing application (see §8), the tail latency of a

# Supported & Evaluated XPUs


NV GV100 GPU


AMD MI50 GPU

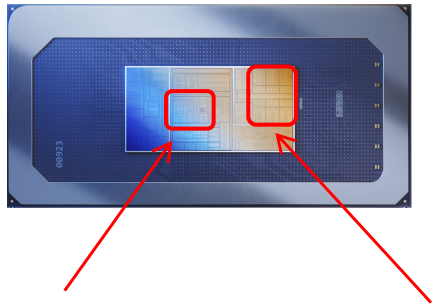
Ascend 910b NPU



Xilinx VU9P FPGA

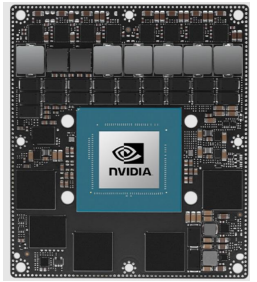
**integrated with AWS-F1*


NV K40m GPU



Intel NPU3720 + Arc iGPU

**integrated with Core Ultra 185H*



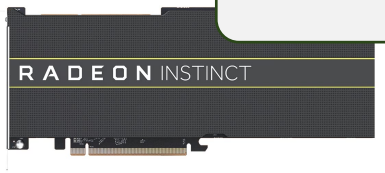

NV DLA *(NPU)* + OFA *(ASIC)* + PVA *(ASIC)*

**integrated with Jetson AGX Orin*

# Level-1: *General* Design for *Effortless* Integration

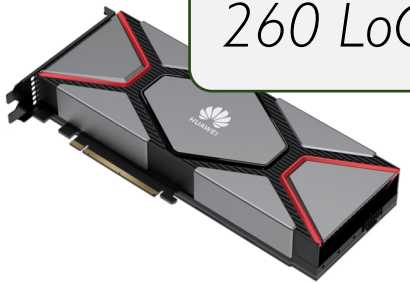

511 LoC

NV GV100 GPU



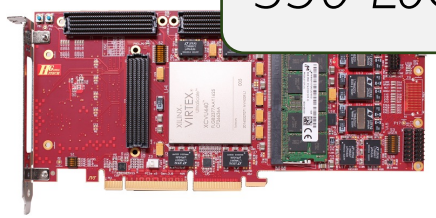

841 LoC

AMD MI50 GPU



260 LoC

Ascend 910b NPU

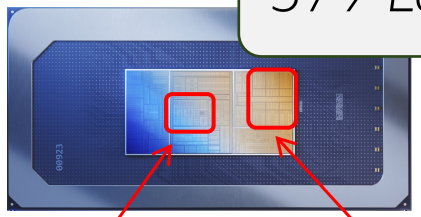


350 LoC

Xilinx VU9P FPGA

**integrated with AWS-F1*

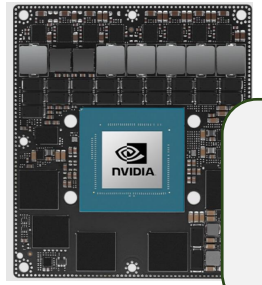

NV K40m GPU



379 LoC

Intel NPU3720 + Arc iGPU

**integrated with Core Ultra 185H*




247 (NPU)
214 (ASIC)

NV DLA *(NPU)* + OFA *(ASIC)* + PVA *(ASIC)*

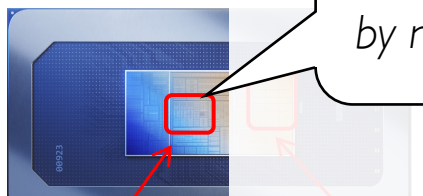
**integrated with Jetson AGX Orin*

# Level-2 Preemption Support


NV GV100 GPU


NV K40m GPU

Pure software approach
*by modifying GPU kernel binary*



Hardware-assisted approach
*by modifying driver to leverage the* ***μ-controller***

Intel NPU3720 + Arc iGPU
**integrated with Core Ultra 185H*

# Level-3 Preemption Support


Modify GPU *trap handler* and proactively trigger GPU *interrupts*

NV GV100 GPU



AMD MI50 GPU

Ascend 910b NPU



Xilinx VU9P FPGA

*integrated with AWS-F1




NV K40m GPU



Intel NPU3720 + Arc iGPU

*integrated with Core Ultra 185H



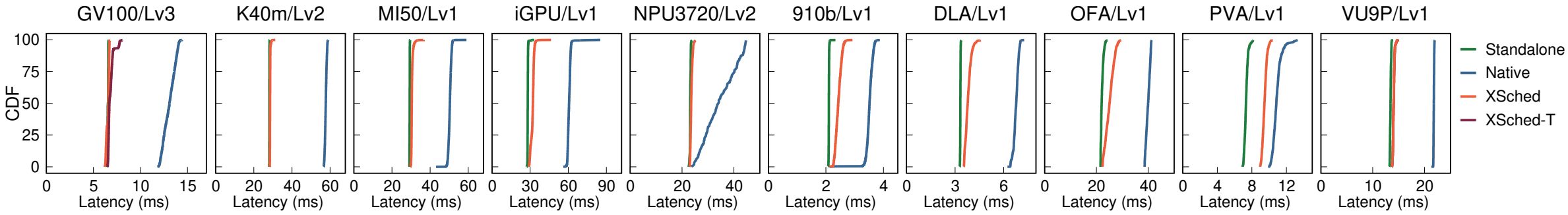
NV DLA *(NPU)* + OFA *(ASIC)* + PVA *(ASIC)*

*integrated with Jetson AGX Orin

# Is XSched Effective?

- Policy: Fixed-priority policy
- Result: With XSched, latency close to standalone, reduce the P99 latency by up to 2.11× compared with native

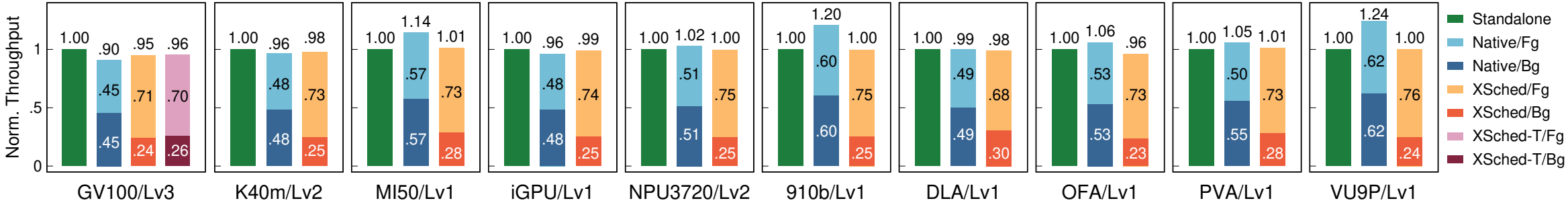
Latency CDF of the high-priority task



# Is XSched Effective?

- Policy: Bandwidth partition policy, partition ratio = 75% : 25%
- Result: Guarantee desired throughput ratio with 1.5% overhead

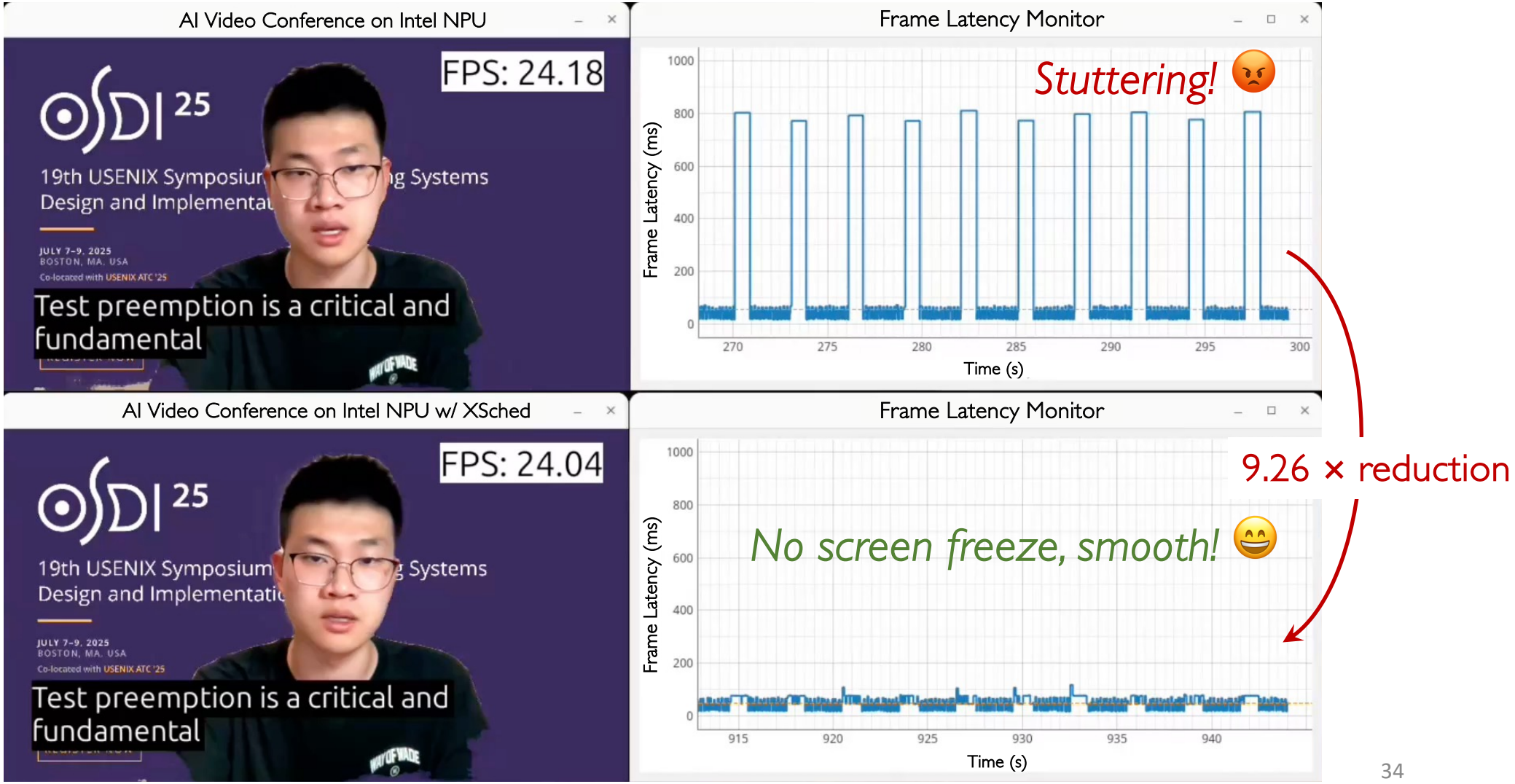
Normalized throughput of the two process



# Application: AI Video Conference with XSched

Native

XSched
*Laxity-based policy*

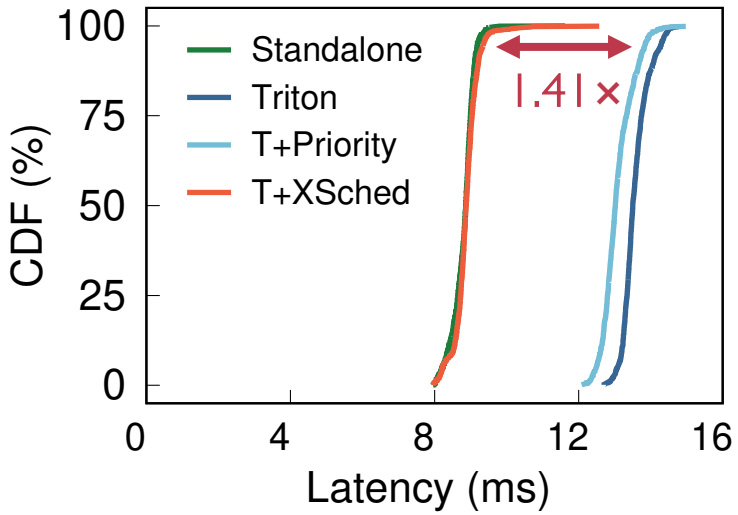


# Application: Inference Serving with XSched

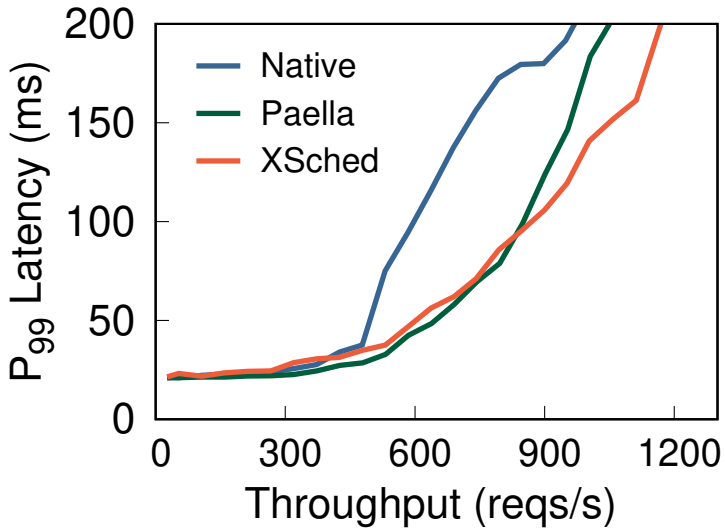
- Integrated with Triton
- 1.41× reduction on P99

- Comparable with ad-hoc solutions, Paella [SOSP'23]

Latency CDF of the high-priority requests.




Throughput-latency curve.



# Conclusion


XSched — A preemptive scheduling framework that supports diverse XPUs

Challenges

- Complex XPU software stacks
- Diverse and evolving hardware capabilities

Key Idea

- XQueue — A preemptible command queue abstraction
- Multi-level hardware model (optimal scheduling performance and compatibility)

Code

Publicly available at https://github.com/XpuOS/xsched