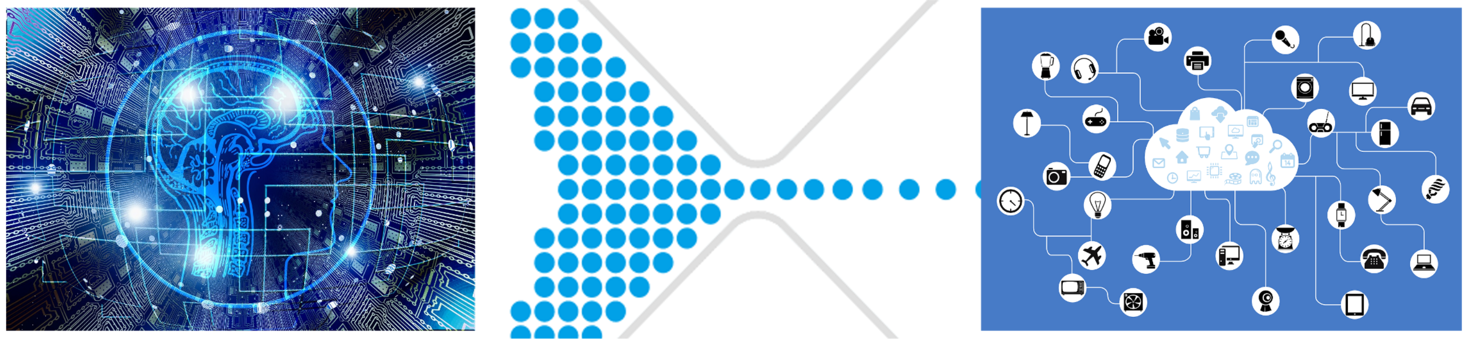
XSP: Across-Stack Profiling and Analysis of Machine Learning Models on GPUs

Cheng Li*[1], Abdul Dakkak*[1], Jinjun Xiong[2], Wei Wei[3], Lingjie Xu[3], Wen-mei Hwu[1]
University of Illinois Urbana-Champaign[1], IBM Research[2], Alibaba Group[3]
{cli99, dakkak, w-hwu}@illinois.edu, jinjun@us.ibm.com, {w.wei, lingjie.xu}@alibaba-inc.com

Video: https://youtu.be/v95JfmM66eE

Background

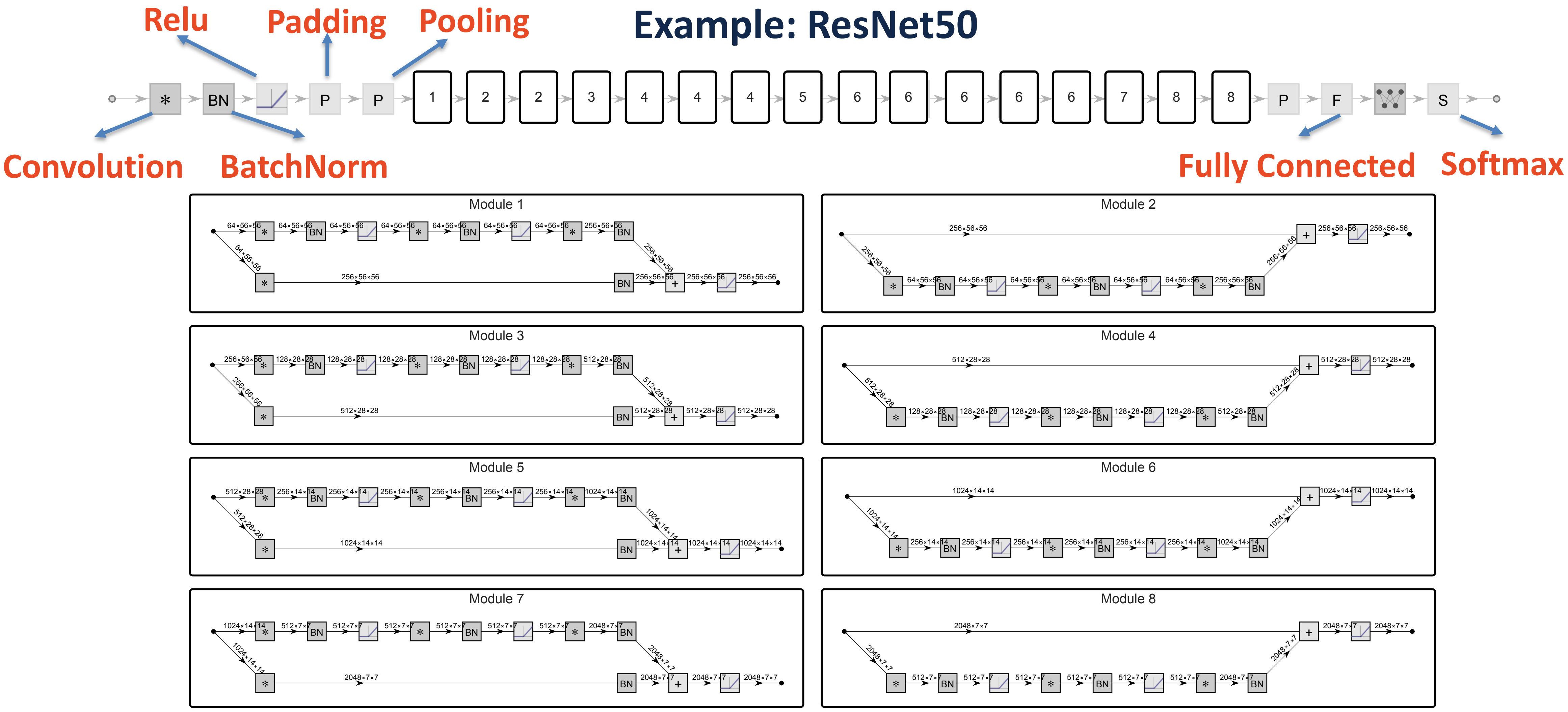
- Machine Learning (ML) models are used in many application domains
- Understanding ML inference performance is an increasingly pressing but challenging task



Slow adoption of DL innovations

ML Model

A graph where each vertex is a layer (or operator) and an edge represents data transfer



ML Inference Pipeline

Input Image

Pre-processing

- Image decoding
- Resizing
- Normalization
- Type conversion

Input Tensor

Prediction

Model prediction using framework API

Output Tensor

Post-processing

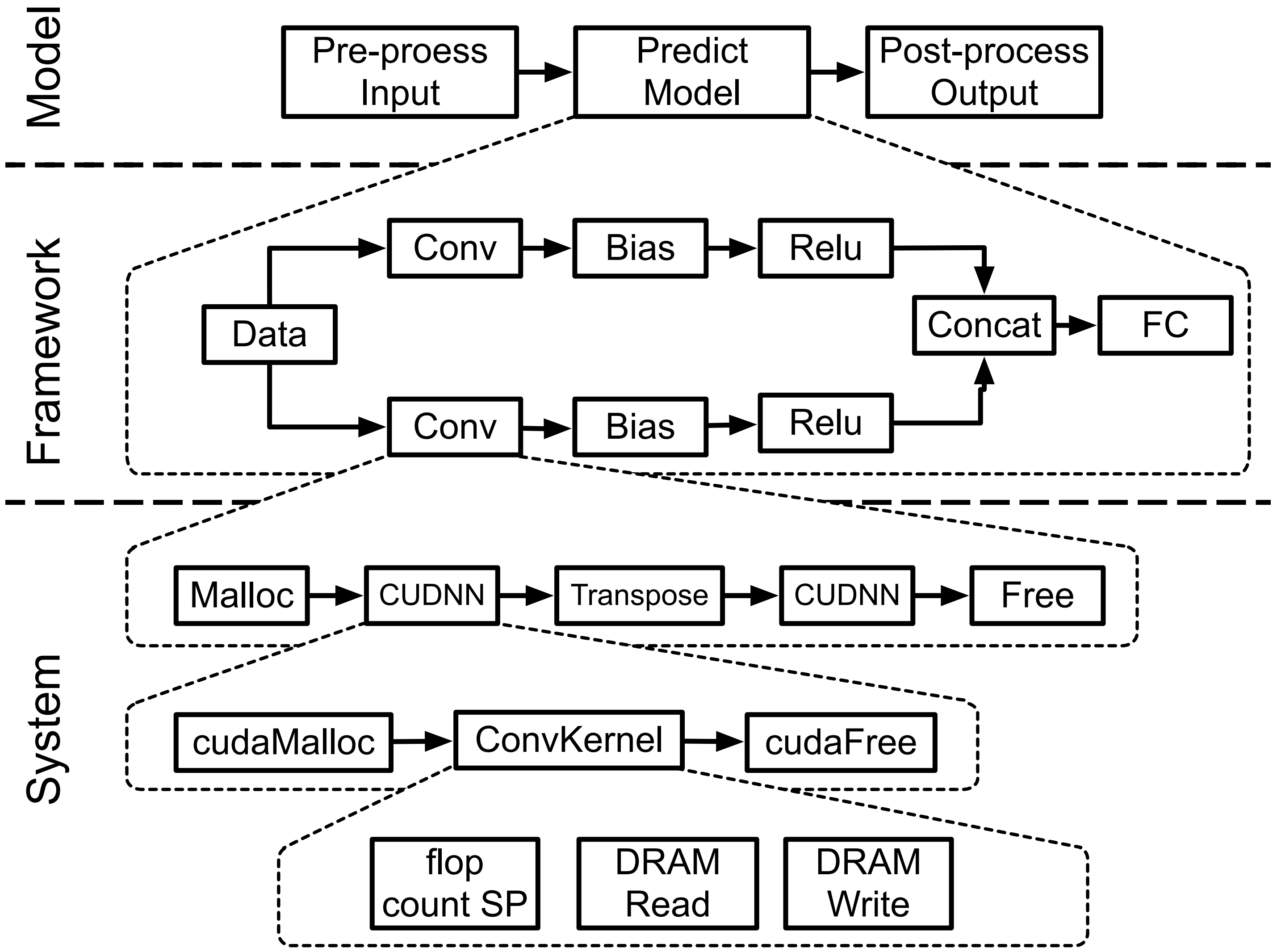
Unpacking into pairs (label, probabilities) and sorting

Top1

(dog, 0.99)

XSP Motivation

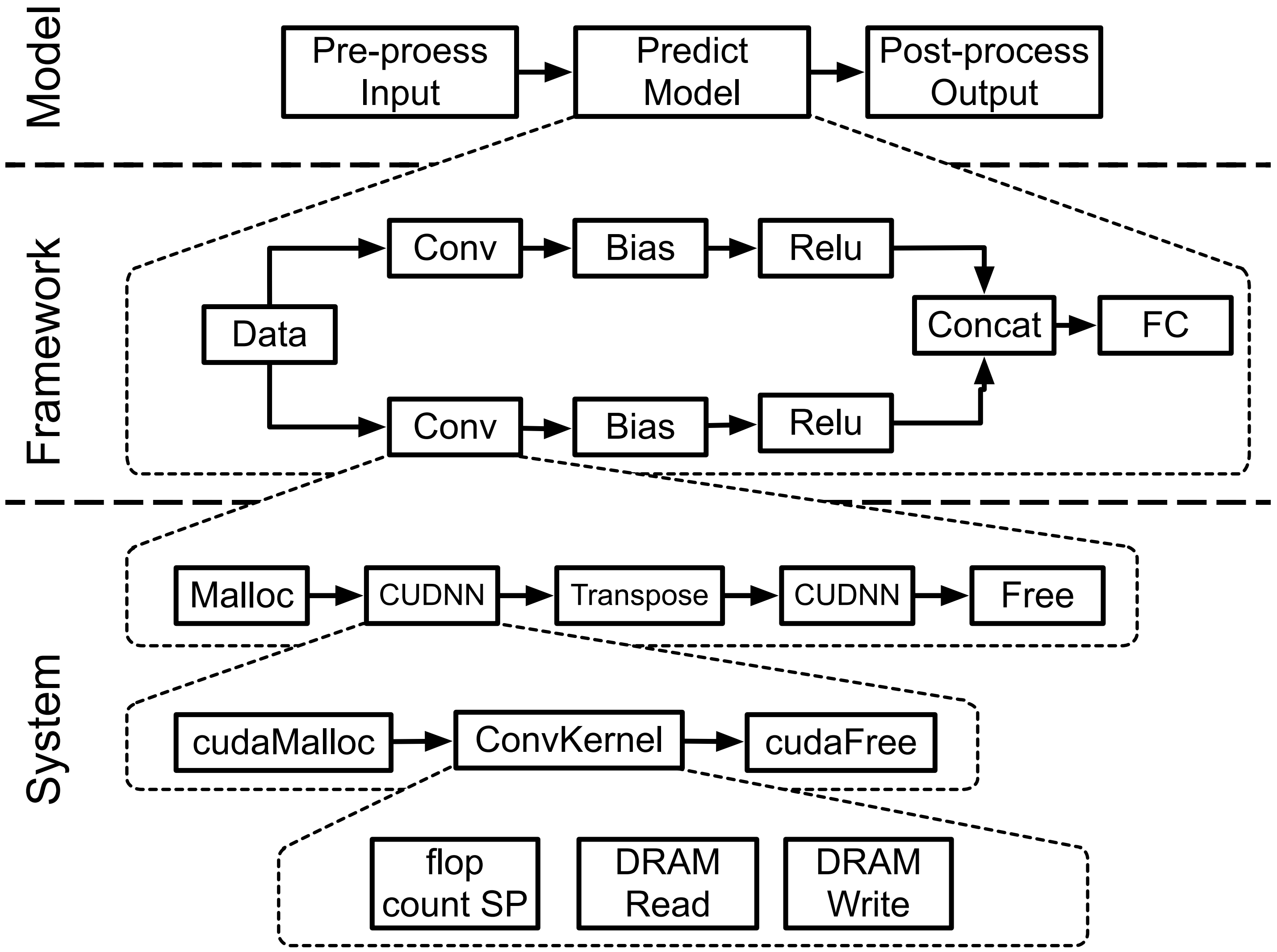
- A holistic view of the model execution is needed
- Existing profiling tools are disjoint
  - Profiling at different granularities means switching between tools
  - No correlation between profiles



Levels of the HW/SW stack

XSP Motivation

- Inference is impacted by the interplay between levels of the HW/SW stack
- Any of them can be a bottleneck



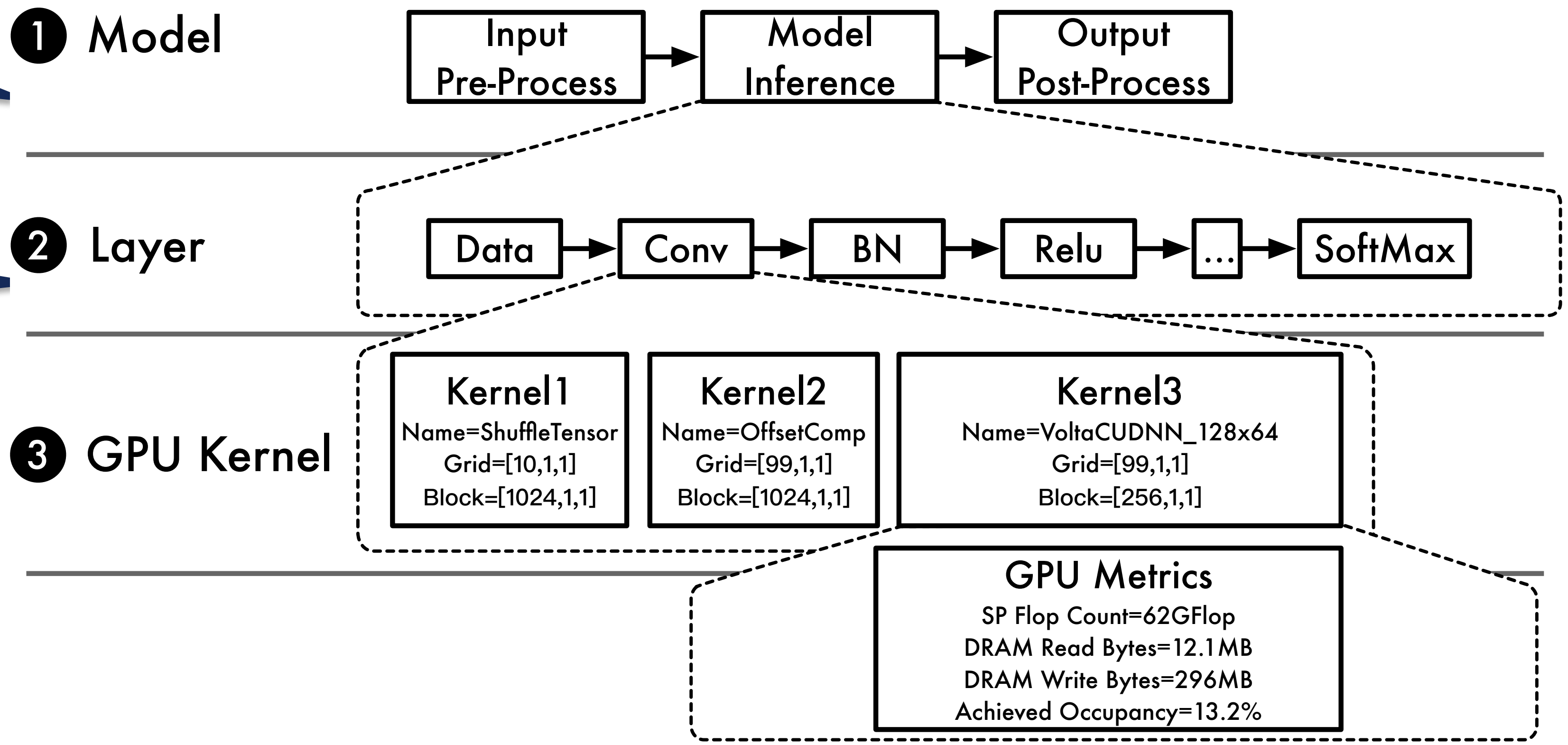
Levels of the HW/SW stack

Current DL Profiling on GPUs

Using code insertion

Using framework profiler

Using nvprof or Nsight



One has to manually perform the difficult task of correlating these disjoint profiles

Model-, layer-, and GPU kernel-level profiles of MLPerf ResNet50 v1.5 with batch size 256 on a Volta GPU

An Approach - Modifying Frameworks

- NGC frameworks (TensorFlow, PyTorch, etc.) are instrumented with NVTX markers
  - GPU profile with layer annotations, lacks framework profiling
  - May inhibit frameworks from performing some optimizations
  - Does not work for DL models that use customized frameworks
- TensorFlow profiler
  - framework profile with some GPU profiling
  - Does not work for other frameworks
- **Vendor lock-in & limited applicability**

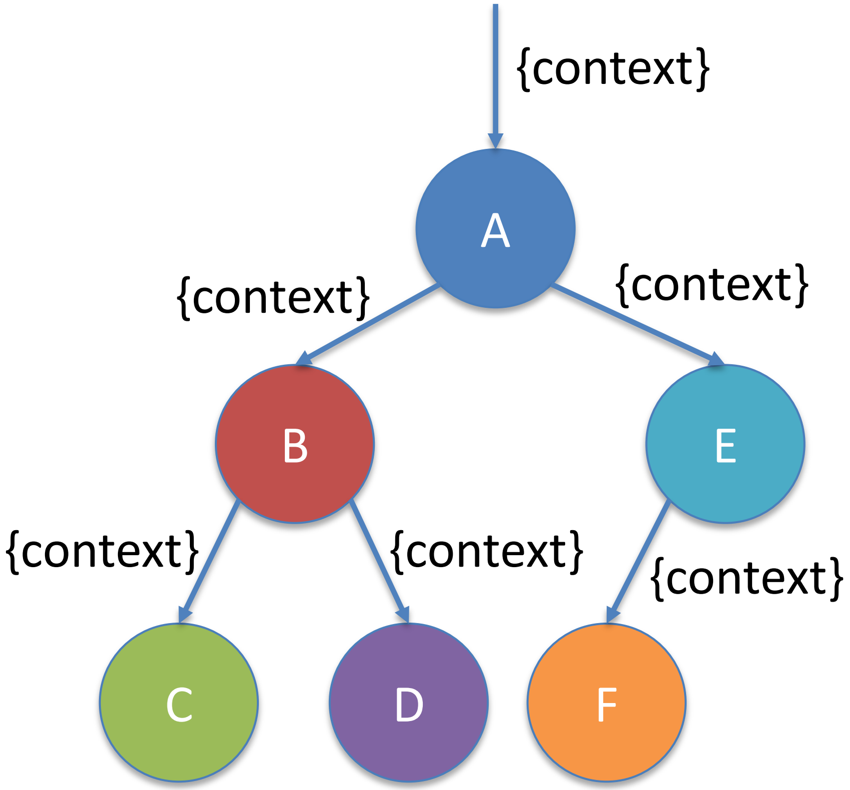
XSP: Across-stack Profiling

- Incorporates profile data from different sources to obtain a holistic and hierarchical view of DL workloads
  - Innovatively leverages distributed tracing
- Accurately captures the profiles at each HW/SW stack level despite the profiling overhead
  - Leveled experimentation methodology
- Coupled with an automated analysis pipeline
- Reveals insights that would otherwise be difficult to discern

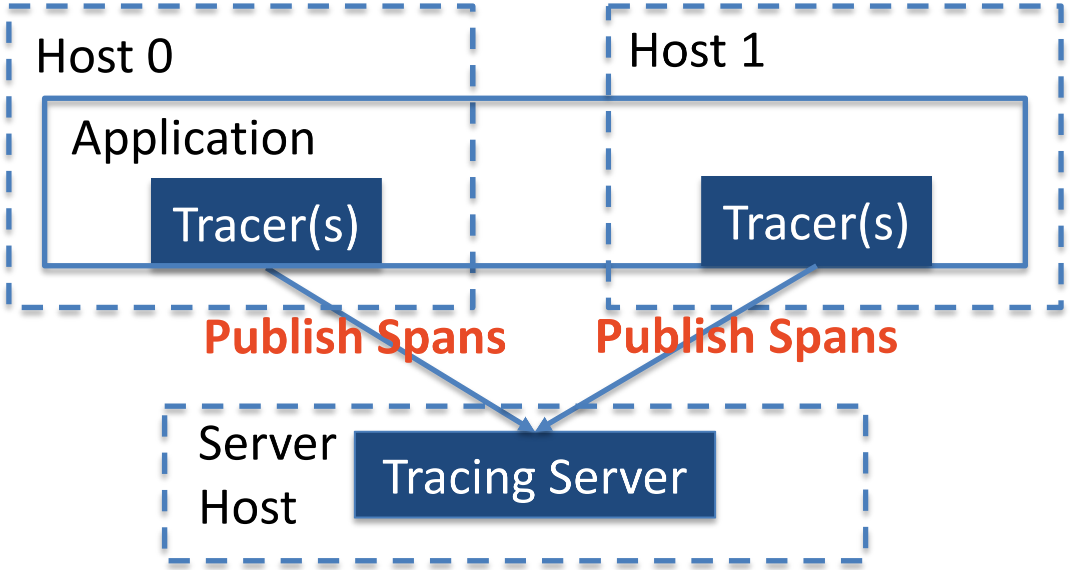
Distributed Tracing

- Designed to monitor distributed applications (e.g. microservices)
- Key Concepts
  - **Span**: a named, timed operation representing a piece of the workflow
    - **Start & end timestamps**
    - **Tags & Logs**: key-value pairs of user-defined annotation or logging messages for spans
    - **SpanContext**: a state to refer to a distinct span
  - **Trace**: a tree of spans
  - **Tracer**: an object that creates and publishes spans

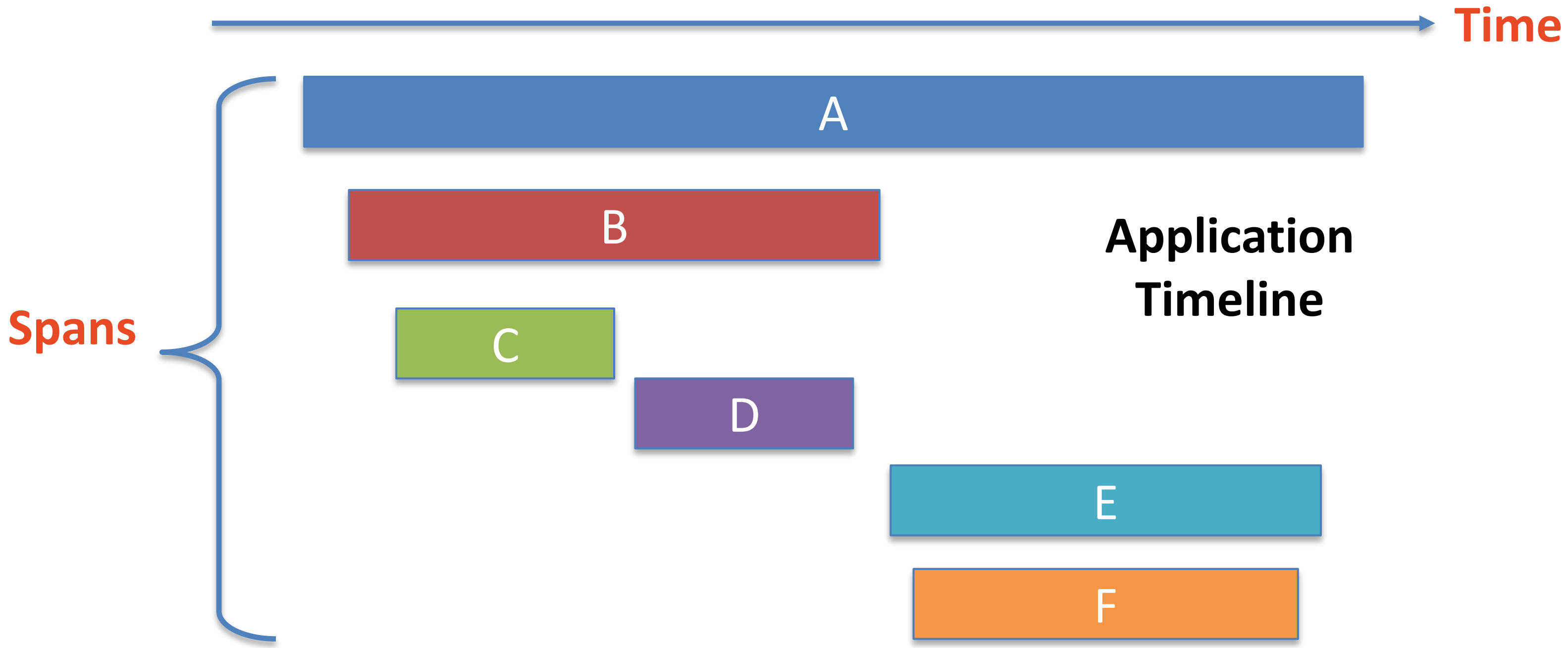
An Example



An application with services (A, B, C, D, E, F) that have causal relationships



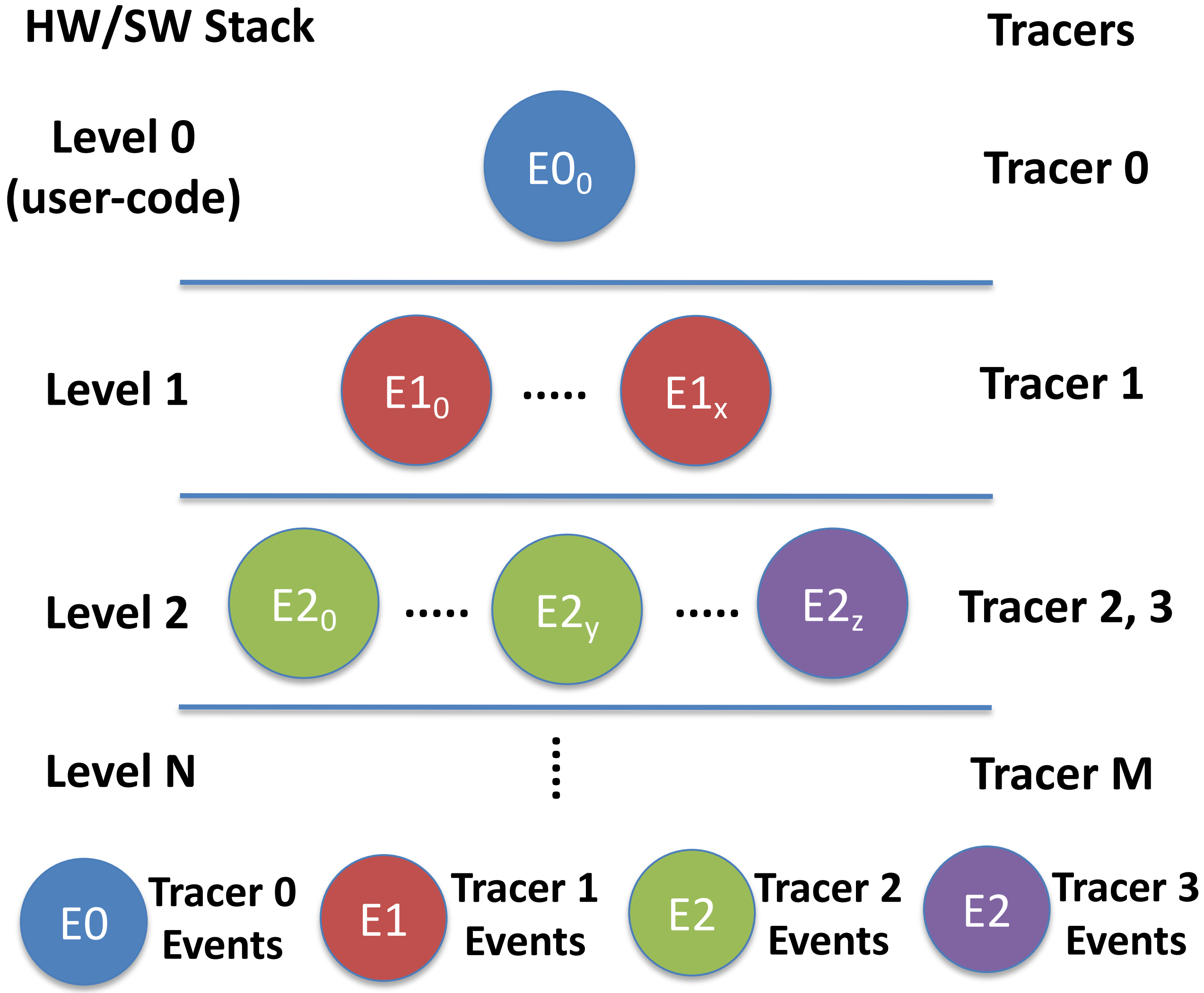
Tracing Workflow



Application Timeline

Leveraging Distributed Tracing in XSP

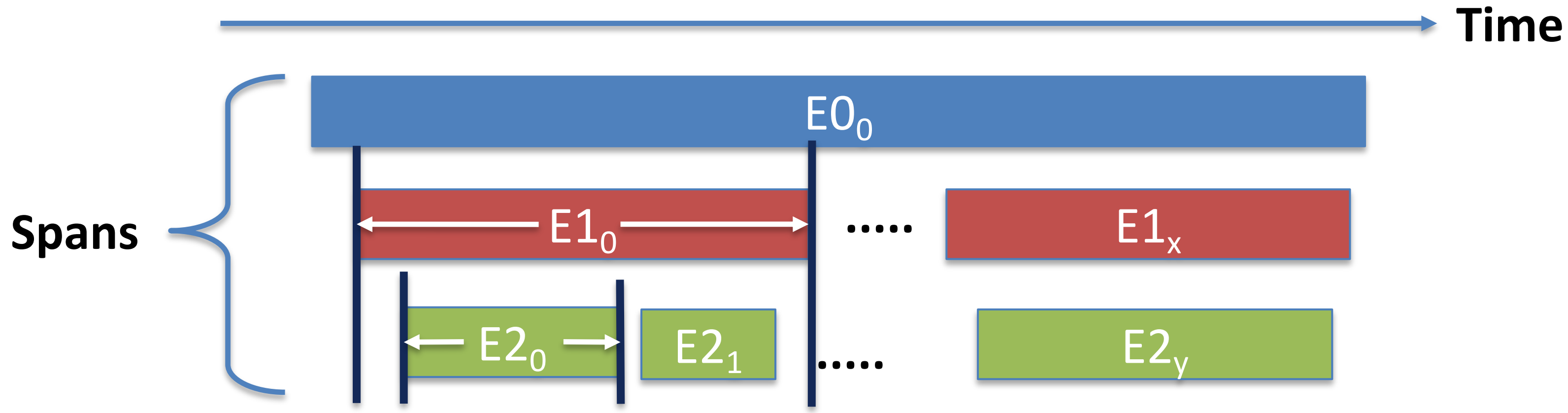
- Observe the similarity between profiling and distributed tracing
- Turn profilers into tracers
- Convert profiled events into spans
- Multiple tracers can exist within a stack level
- Tracers can be enabled/disabled



XSP Design

Constructing Parent/Child Relationships

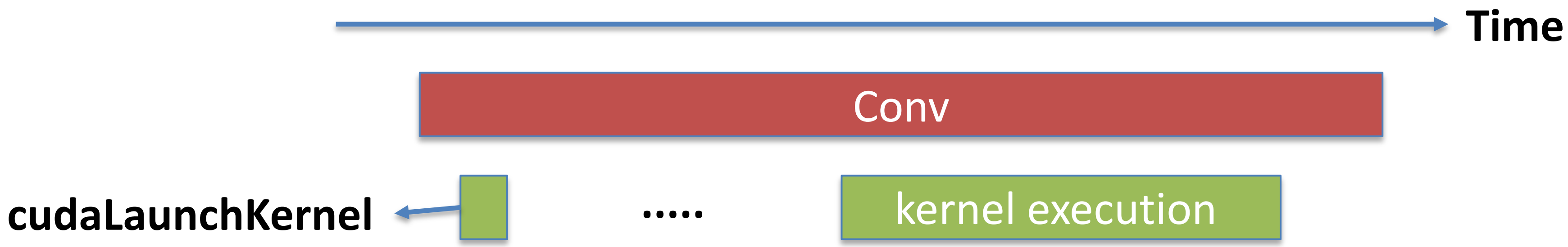
- Tracers use the system clock
- Spans are time intervals and assigned with levels
- During the profile analysis, check interval inclusion
  - If interval s1 contains interval s2 and s1 is a level higher than s2, then s1 is a parent of s2



Time Interval Inclusion

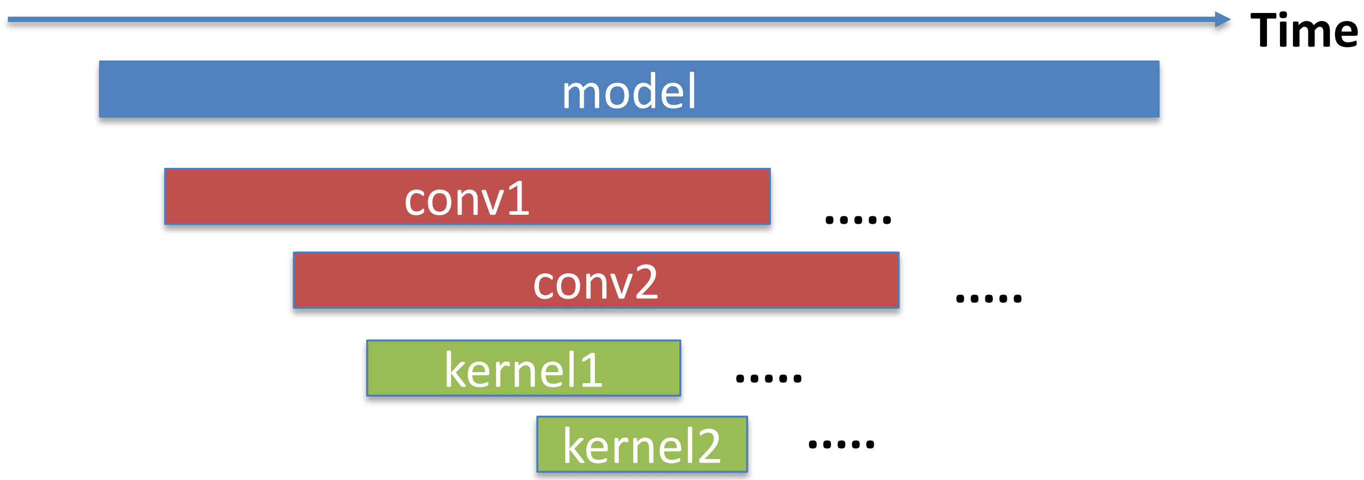
Capturing Asynchronous Events

- E.g. Asynchronous GPU kernel launches
- Capture both the kernel launch and execution spans
  - Use the kernel launch span to figure out the parent span
  - Use the kernel execution span to get performance information or figure out its children spans



Capturing Parallel Events

- E.g. Two conv layers overlap, and each invokes GPU kernels
- Serialize the conv layers to get their correlations to GPU kernels
- Or more complex post-processing



Two conv layers overlap

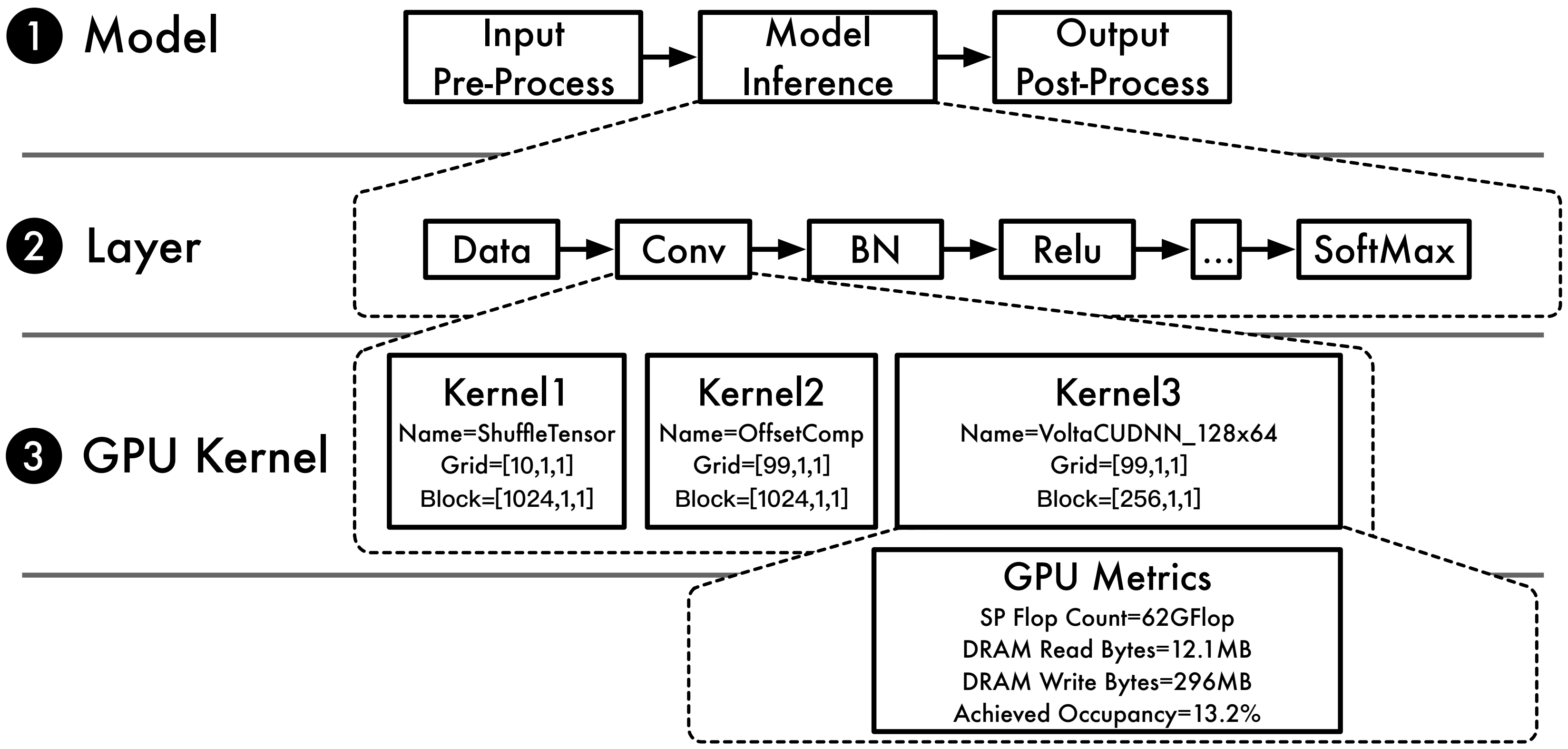
XSP for ML Inference on GPUs

Global Tracer:
User inserts tracing API (startSpan & finishSpan) to capture code sections

Framework Tracer:
Built on top of the framework profiling capability to capture layer level information

GPU Tracer:
Built on top of CUPTI to capture CUDA runtime API, GPU activities, GPU metrics

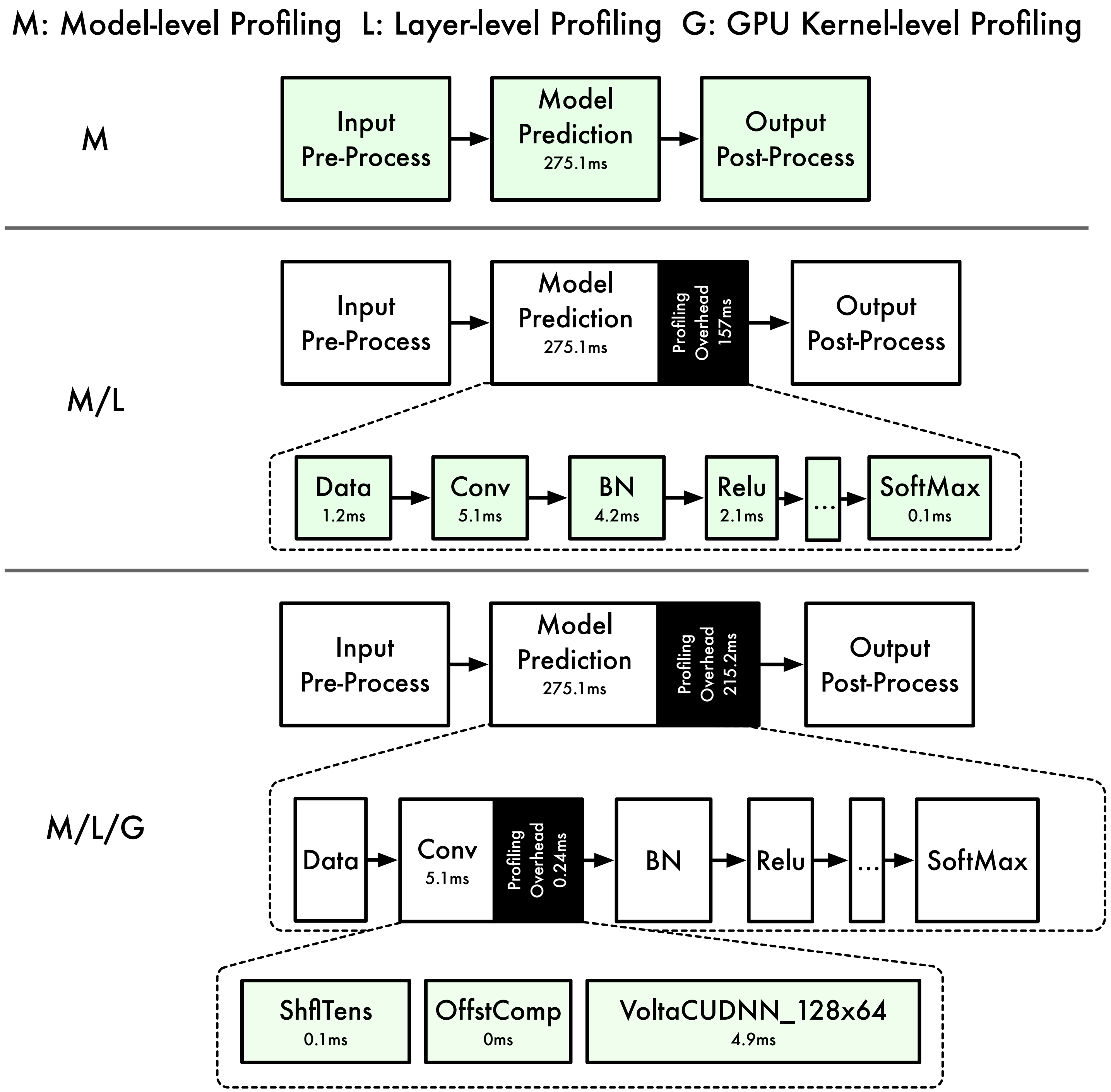
No change to DL frameworks or libraries



Model-, layer-, and GPU kernel-level profiles of MLPerf ResNet50 v1.5 with batch size 256 on a Volta GPU

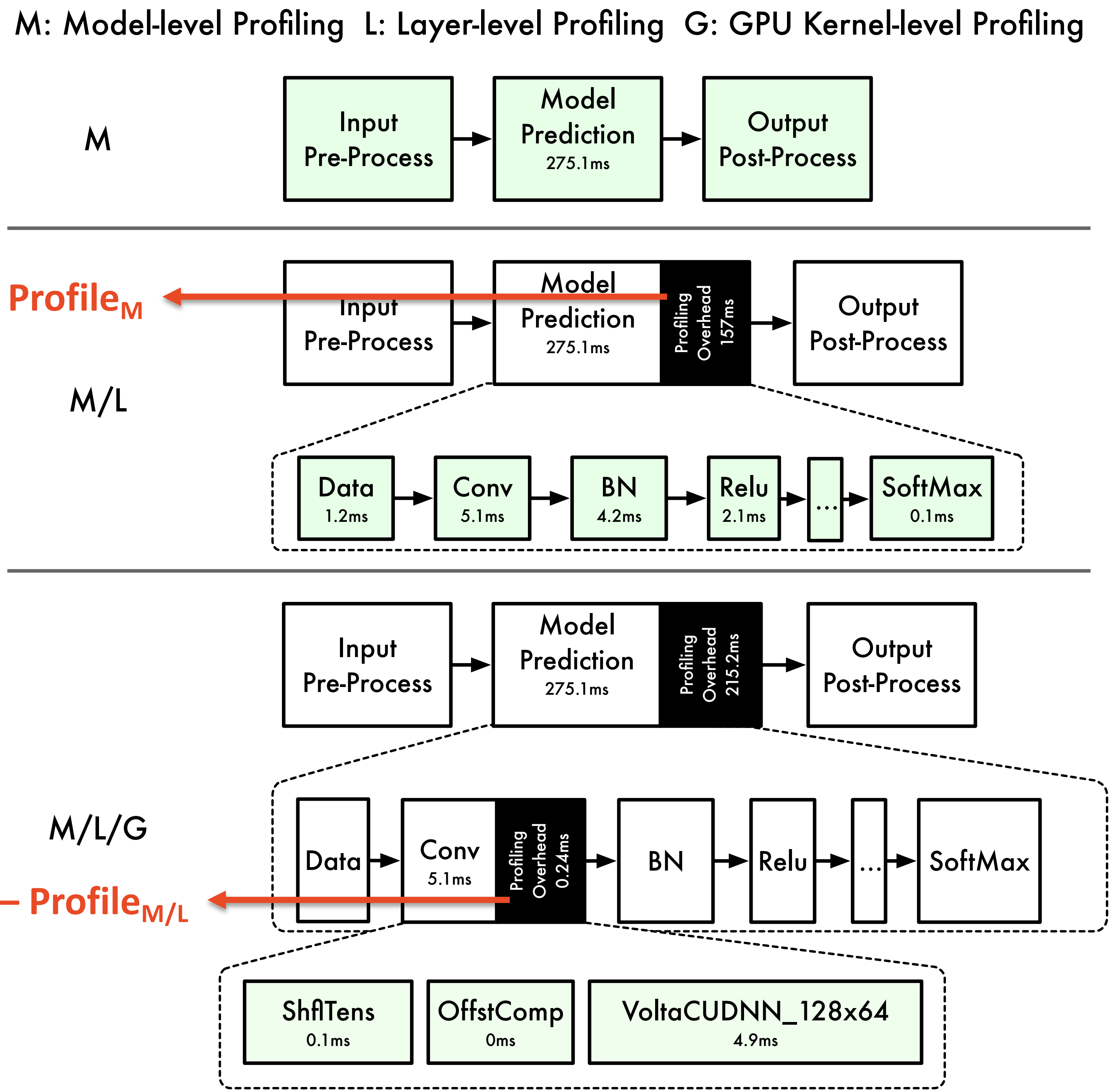
Dealing with Profiling Overhead

- Profiling always comes with overhead
- XSP uses leveled experimentation to get accurate timing for all levels



Leveled Experimentation

- Profilers at level n accurately capture events at level n
- Use traces from runs with different profiling levels enabled
  - Overhead$_n$ = Profile$_{0/\ldots/n}$ – Profile$_{0/\ldots/n-1}$



Automated Across-stack Analysis

M: Model-level profiling
L: Framework-level profiling
G: GPU-level profiling

The 15 analyses performed by XSP using profiles from one or more levels

	Analysis	Profiling Provider	End-to-End Benchmarking	Framework Profilers	NVIDIA Profilers	XSP
A1	Model throughput and latency	M	✓	✗	✗	✓
A2	Layer information	L	✗	✓	✗	✓
A3	Layer latency	L	✗	✓	✗	✓
A4	Layer allocated memory	L	✗	✓	✗	✓
A5	Layer type distribution	L	✗	✓	✗	✓
A6	Layer aggregated latency	L	✗	✓	✗	✓
A7	Layer aggregated allocated memory	L	✗	✓	✗	✓
A8	GPU information	G	✗	✗	✓	✓
A9	GPU roofline	G	✗	✗	✓	✓
A10	GPU aggregated information	G	✗	✗	✓	✓
A11	Layer aggregated GPU information	L/G	✗	✗	✗	✓
A12	Layer aggregated GPU metrics	L/G	✗	✗	✗	✓
A13	GPU vs CPU latency	L/G	✗	✗	✗	✓
A14	Layer roofline	L/G	✗	✗	✗	✓
A15	Model roofline	M/L/G	✗	✗	✓	✓

Example Analysis

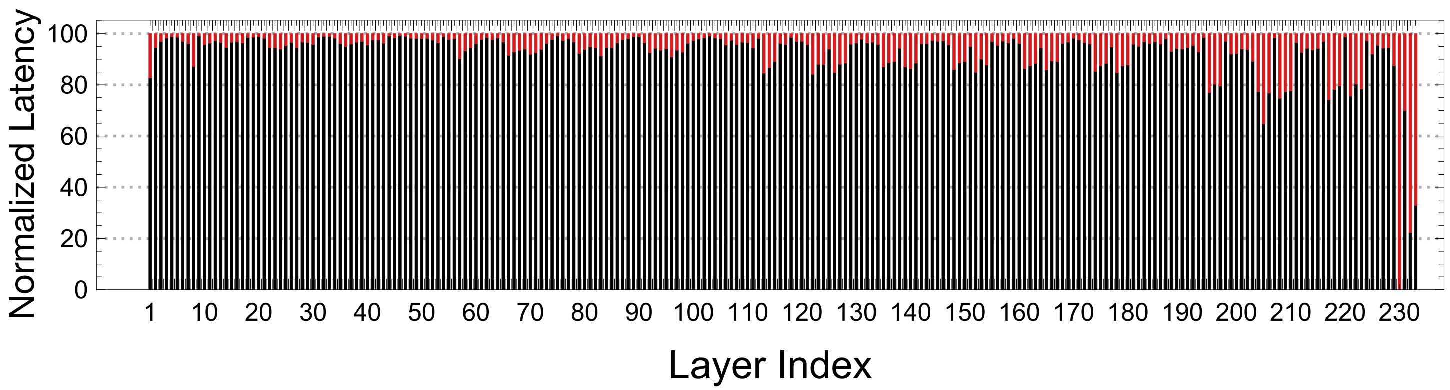
https://ipdps20.netlify.com/tensorflow/mlperf_resnet50_v1.5/

A8

The top 5 most time-consuming GPU kernel invocations

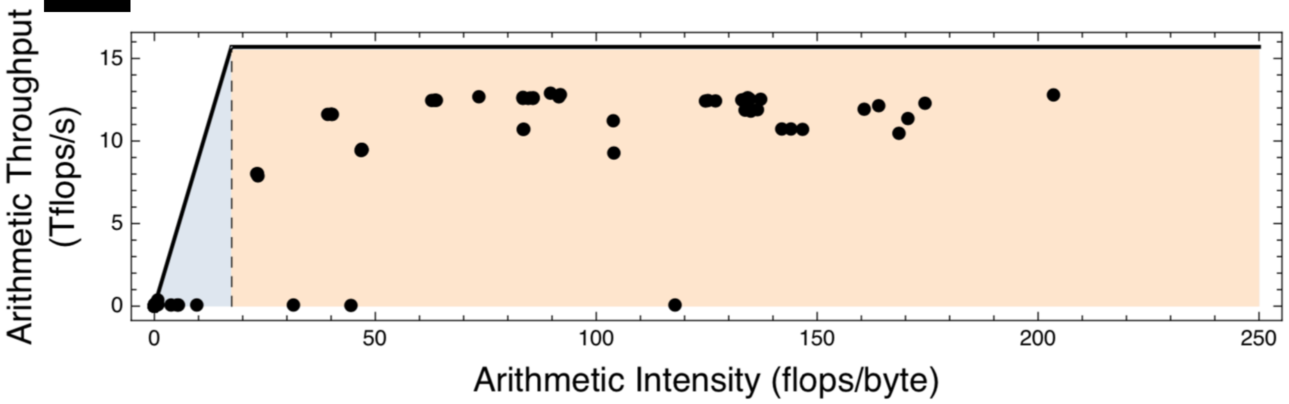
Kernel Name	Layer Index	Layer Kernel Latency (ms)	Kernel Gflops	Kernel DRAM Reads (MB)	Kernel DRAM Writes (MB)	Kernel Achieved Occupancy (%)	Kernel Arithmetic Intensity (flops/byte)	Kernel Arithmetic Throughput (Tflops/s)	Memory Bound?
volta_cgemm_32x32_tn	221	6.04	77.42	40.33	43.86	12.18	876.97	12.82	✗
volta_cgemm_32x32_tn	208	6.03	77.42	43.93	43.81	12.19	841.59	12.83	✗
volta_scudnn_128x128_relu_interior_nn_v1	195	5.48	59.20	27.71	8.40	15.49	1,563.30	10.80	✗
volta_scudnn_128x64_relu_interior_nn_v1	3	4.91	62.89	11.55	283.05	13.20	203.58	12.81	✗
volta_scudnn_128x128_relu_interior_nn_v1	57	4.56	59.24	34.83	37.64	15.15	779.55	12.99	✗

A13



GPU vs Non-GPU Normalized latency

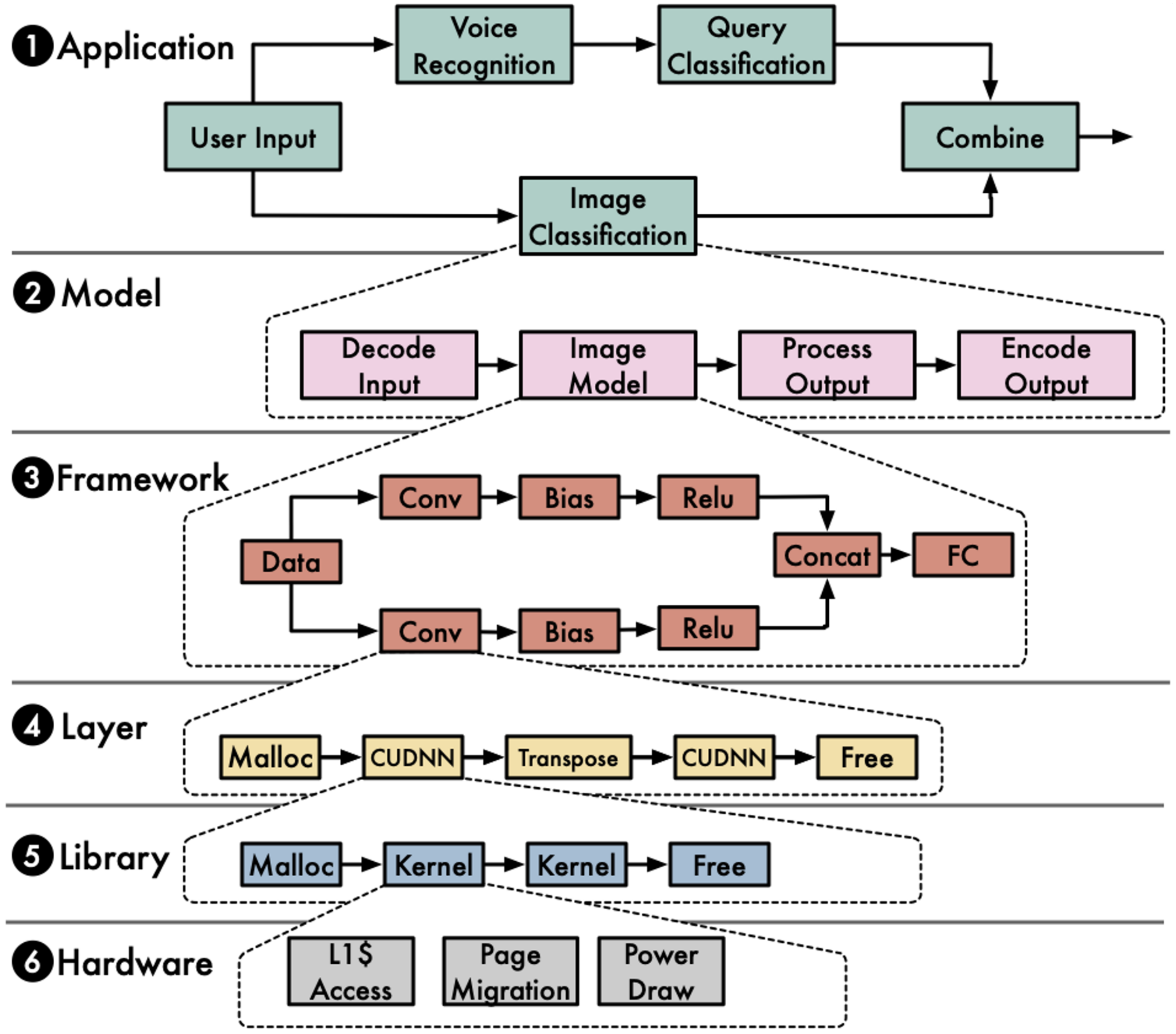
A14



Layer roofline analysis

XSP Extensibility

- Other profiling tools or methods can be integrated
  - More tracers at each stack level, e.g. CPU+GPU
  - Capture more stack levels, e.g. ML library level and application level
  - Work with accelerators and simulators
- Add more types of analyses
- Add ML training support



Conclusion

- XSP is an across-stack profiling design that aggregates profile data from different sources and correlates them to construct a holistic and hierarchical view of ML model execution
  - A smooth hierarchical step-through of model performance at different levels within the HW/SW stack to identify bottlenecks
  - Systematic comparisons of models, frameworks, and hardware through the consistent profiling and automated analysis workflows
  - Extensible to accommodate different use cases

Thank you

More information in the paper

Cheng Li*[1], Abdul Dakkak*[1], Jinjun Xiong[2], Wei Wei[3], Lingjie Xu[3], Wen-mei Hwu[1]

University of Illinois Urbana-Champaign[1], IBM Research[2], Alibaba Group