14th USENIX Symposium on Operating Systems Design and Implementation (**OSDI ’20**)

Rammer:

Enabling Holistic Deep Learning Compiler Optimizations with rTasks

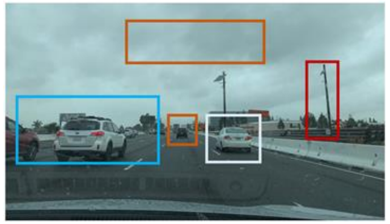
Lingxiao Ma [*†◊], Zhiqiang Xie [*‡◊], Zhi Yang [†], Jilong Xue [◊], Youshan Miao [◊],
Wei Cui [◊], Wenxiang Hu [◊], Fan Yang [◊], Lintao Zhang [◊], Lidong Zhou [◊]

† *Peking University*

‡ *ShanghaiTech University*

◊ *Microsoft Research*

* *Equal contribution*

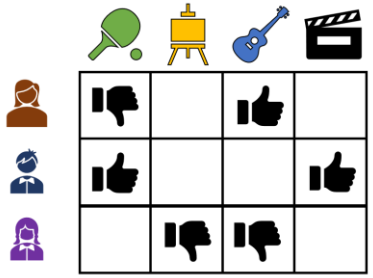


Self-driving

Personal Assistant

Search Engine

Art



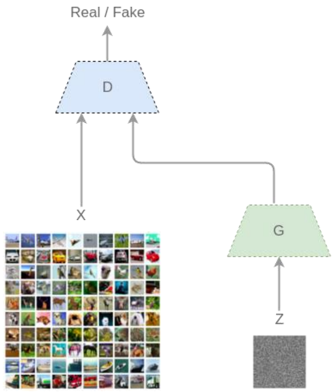
Recommendation

The Rise of Deep Learning

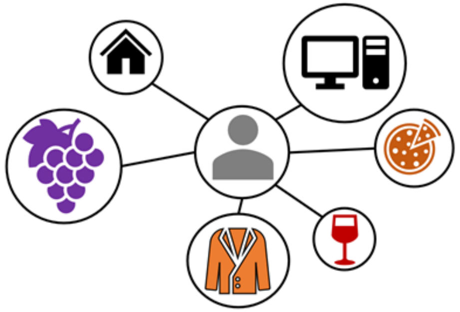
Image Recognition

Speech Recognition

Natural Language



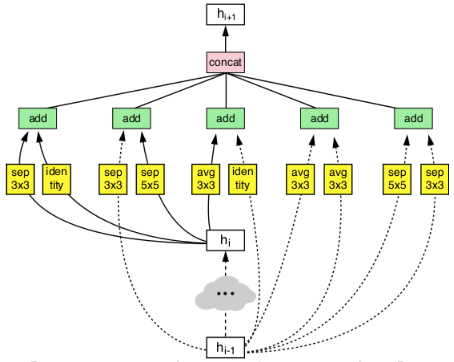
Generative Model



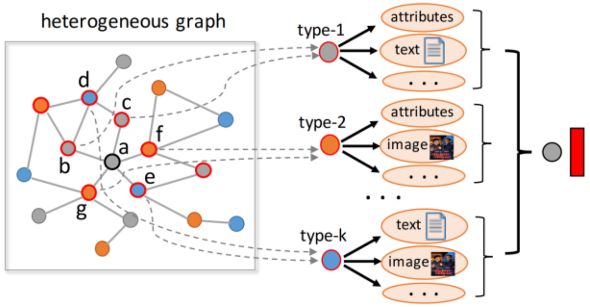
Graph Model

DL Frameworks Bridge the Gap of Models and Hardware

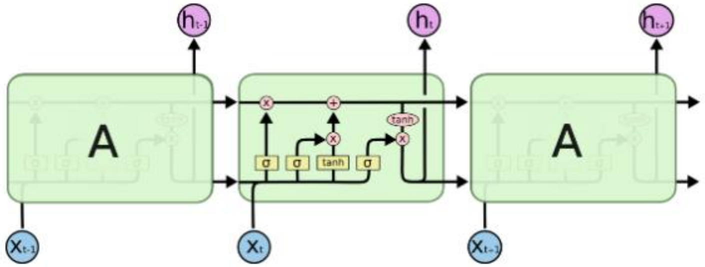
Deep Neural Network (DNN) Models



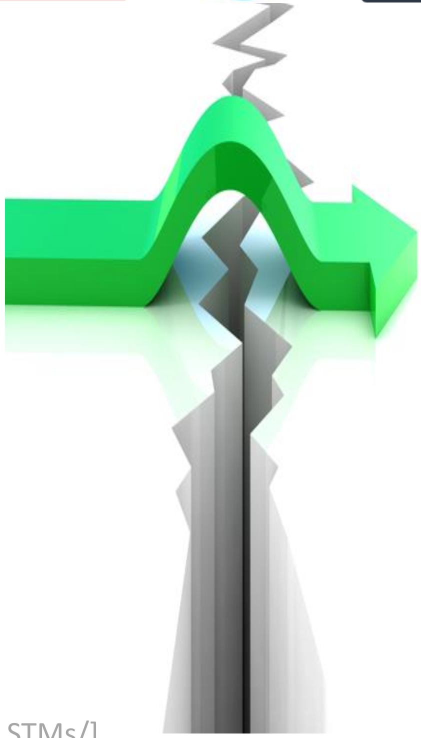
[Barret Zoph, et.al., CVPR'18]



[Chuxu Zhang, et.al., KDD'19]



[Christopher Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/]



Modern Accelerators

NVIDIA GPU

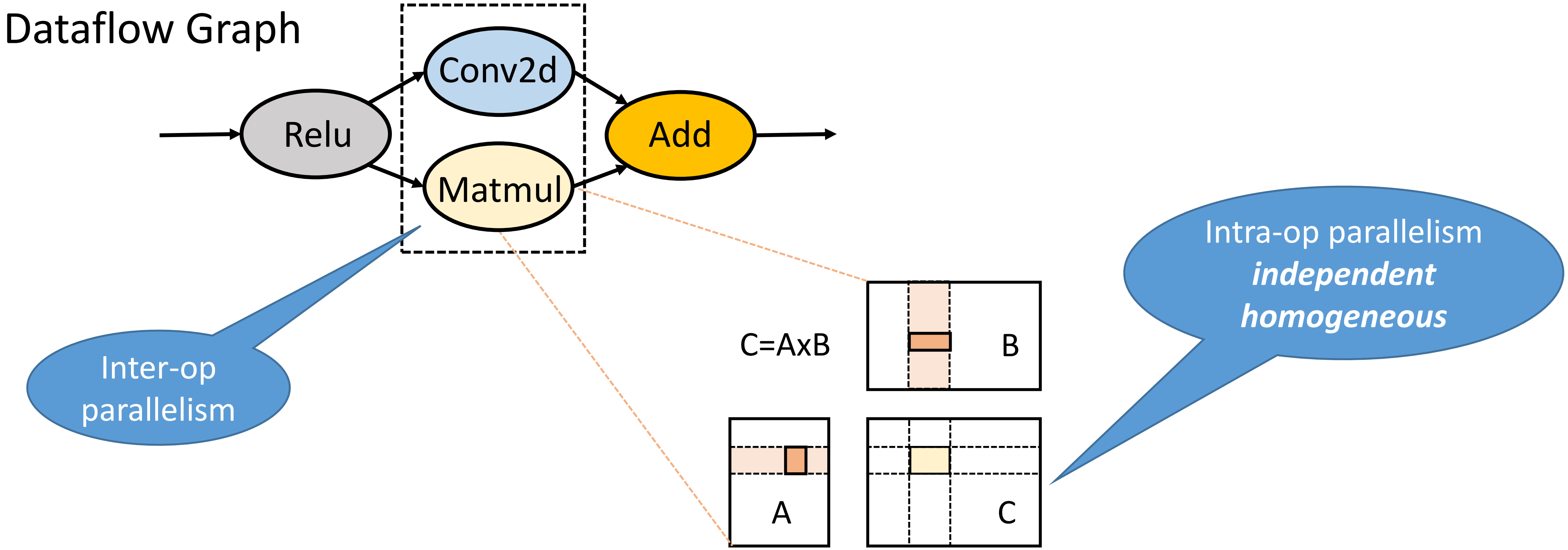
AMD GPU

TPU

Graphcore IPU

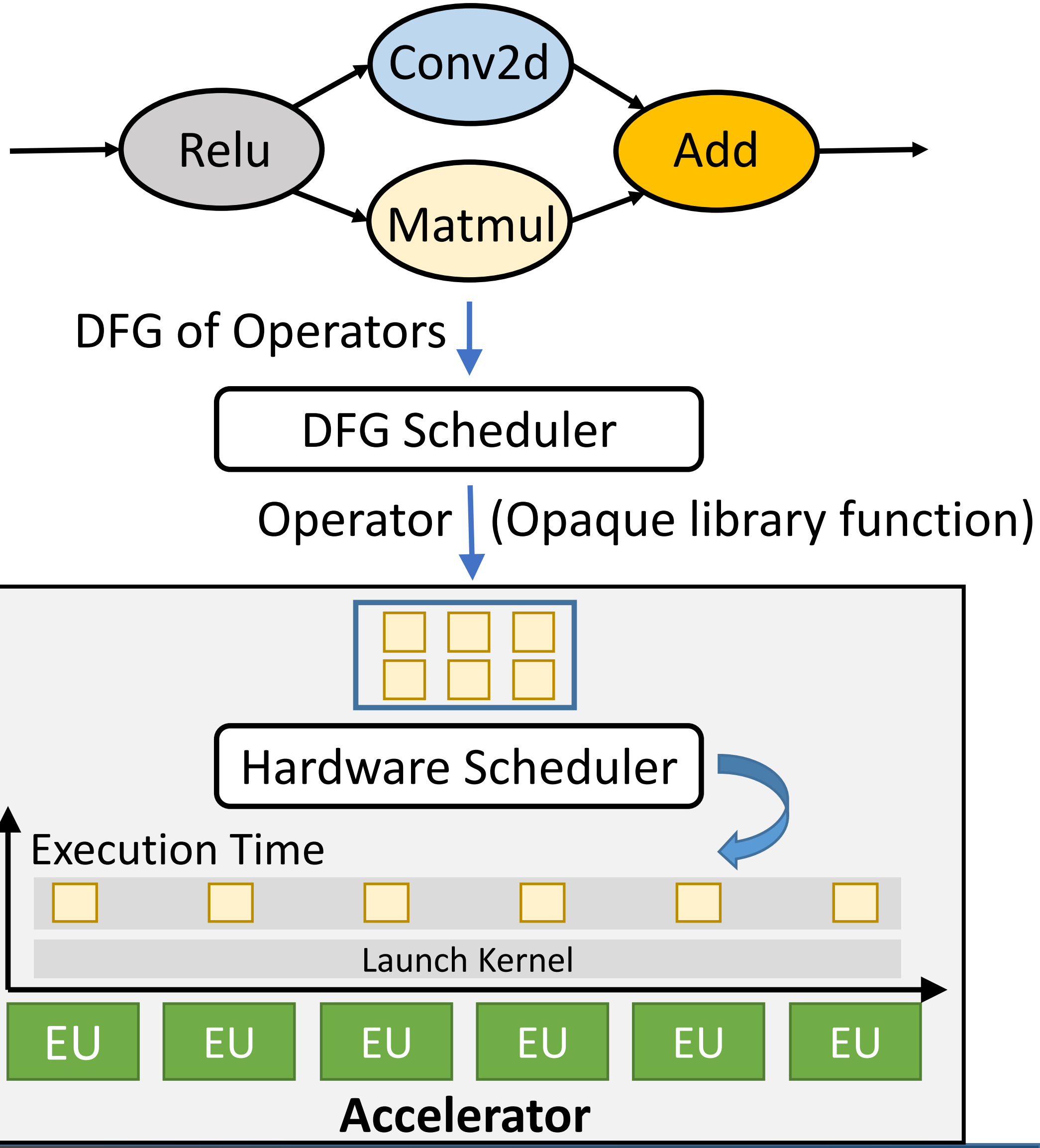
Existing Approach

- DNN is usually modeled as a dataflow graph (DFG)
- DFG naturally contains two levels of parallelism



Existing Approach: Two-Layer Architecture

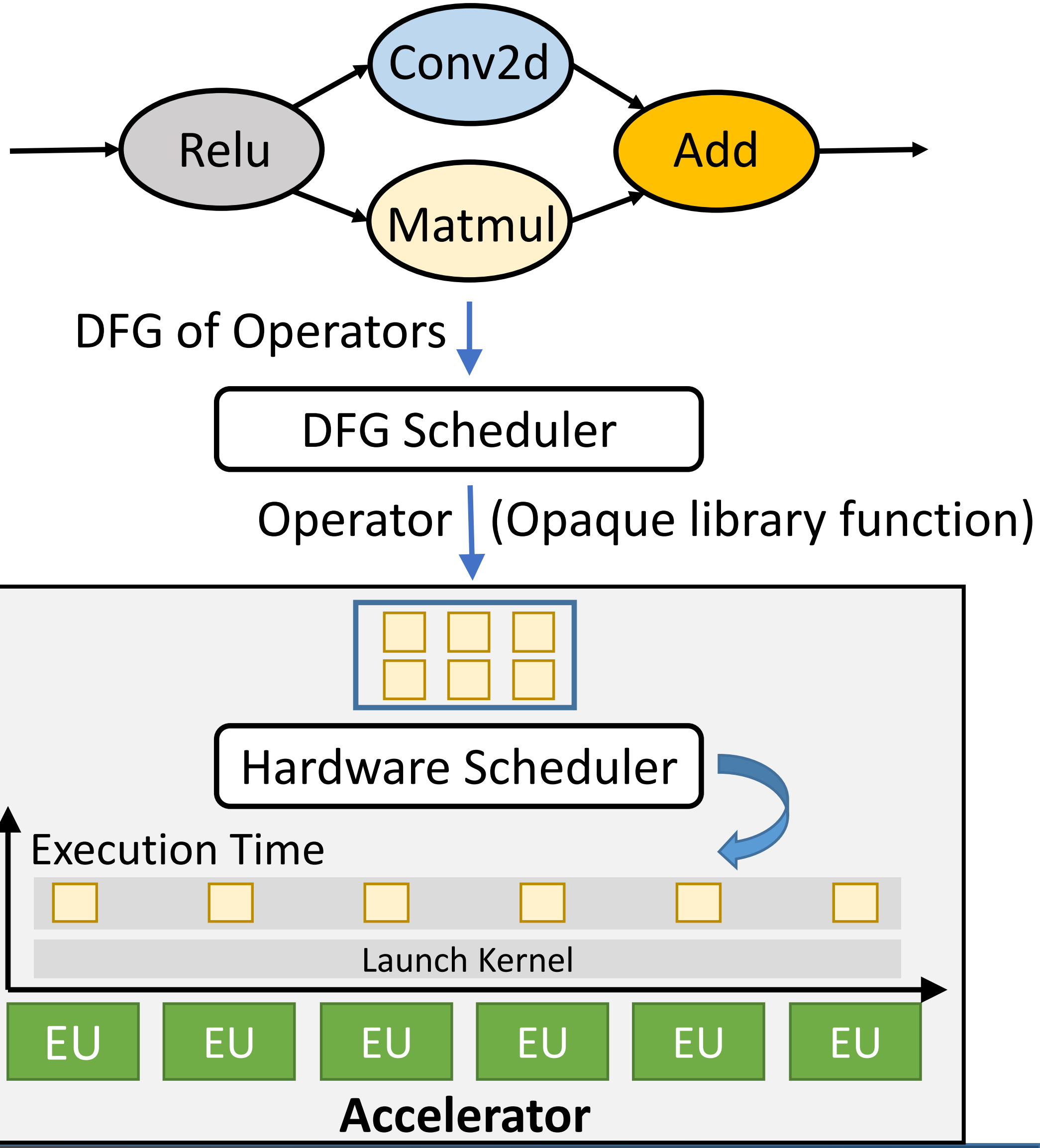
- *DFG scheduler* exploits inter-op parallelism
    - Emit operators that are ready for execution
    - Operators are treated as ***opaque library functions***
- *Hardware scheduler* exploits intra-op parallelism
    - Map intra-op computation to parallel execution units (EUs)



Limitations of Existing Two-Layer Architecture

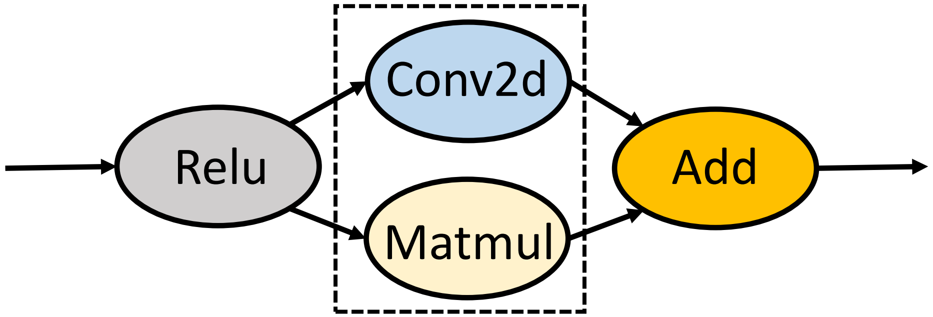
- Two-layer architecture works well when:
  - Schedule overhead is negligible
  - Intra-op parallelism can saturate all EUs
- However, this is often not the case in practice
  - Accelerators are becoming more and more powerful
    - P100 (9.3 Tflops) -> RTX 3090 (35.6 Tflops)
- Low GPU utilization
  - 2% ~ 62% utilization
- High operator scheduling overheads
  - 38% ~ 65% non-kernel time

All data are reported on the inference task (BS=1) of 6 models on a V100 GPU (more details in paper).

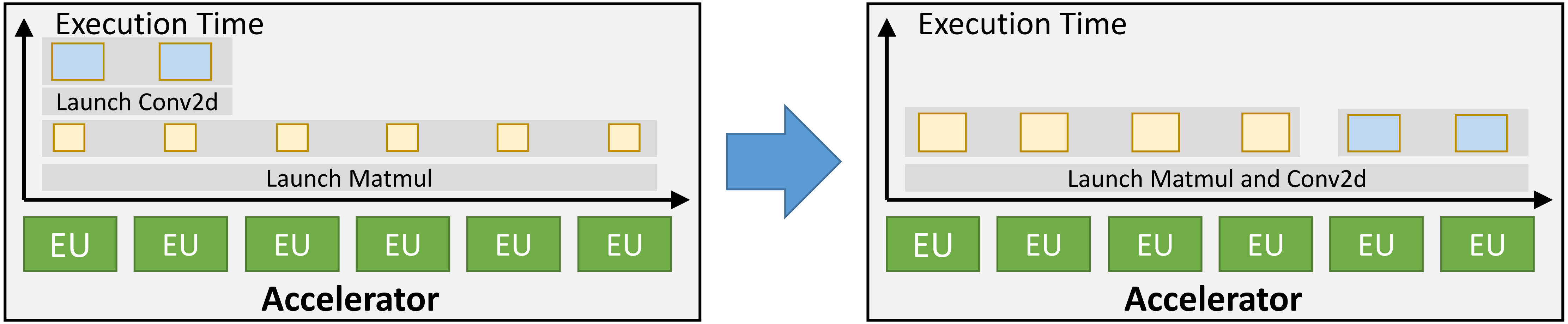


Limitations of Existing Two-Layer Architecture

- Overlook the subtle interplay of inter- and intra- op parallelism



Execution of Two-Layer Architecture **Optimized Execution**



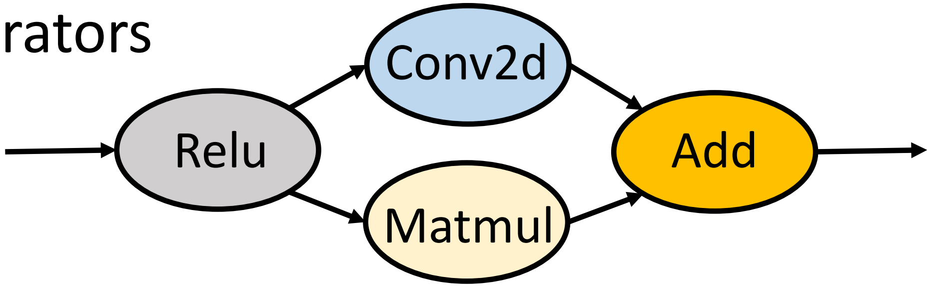
Rammer

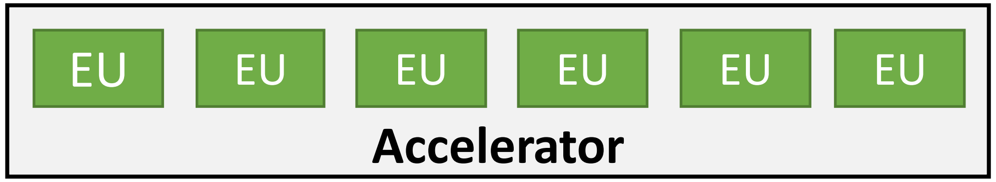
- Key idea: manage the scheduling of ***inter- and intra- operator together***

Rammer

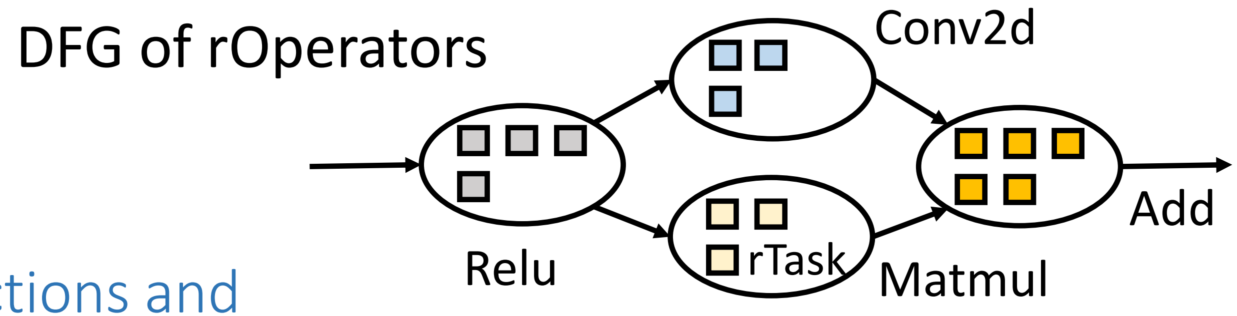
- Challenge 1: operators are opaque functions and do not expose fine-grained intra-op parallelism

DFG of operators



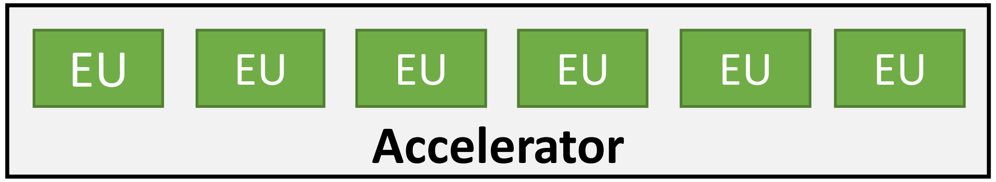


Rammer



- Challenge 1: operators are opaque functions and do not expose fine-grained intra-op parallelism

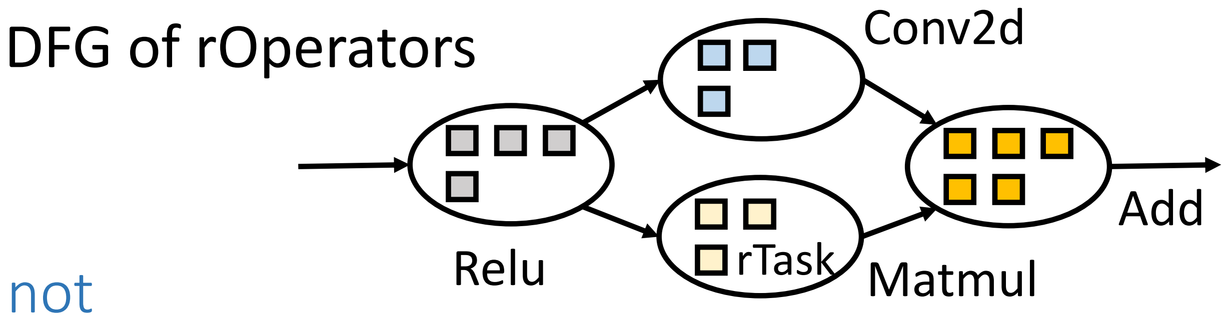
- Solution: ***rTask-Operator (rOperator)*** abstraction
  - Expose fine-grained intra-op parallelism
  - A group of ***independent***, ***homogeneous*** rTasks
  - rTask is the ***minimum computation unit*** on an EU

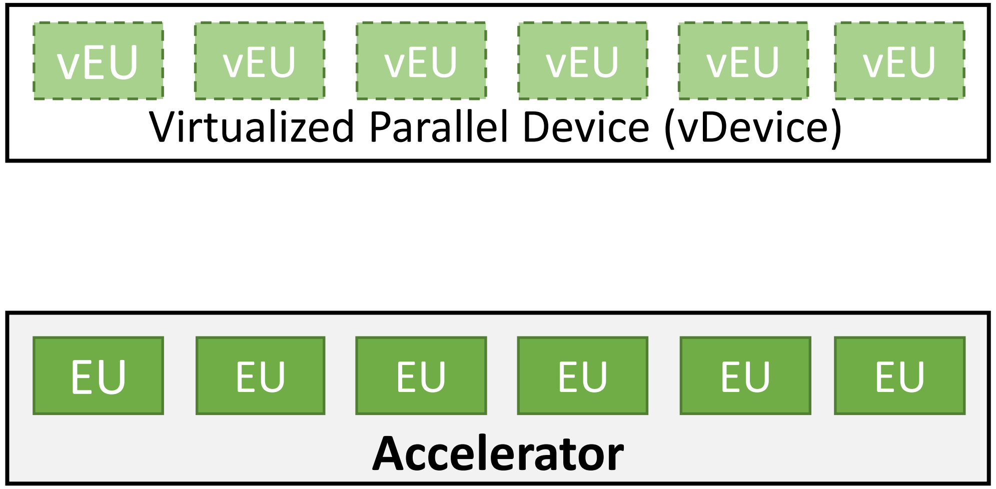


Rammer

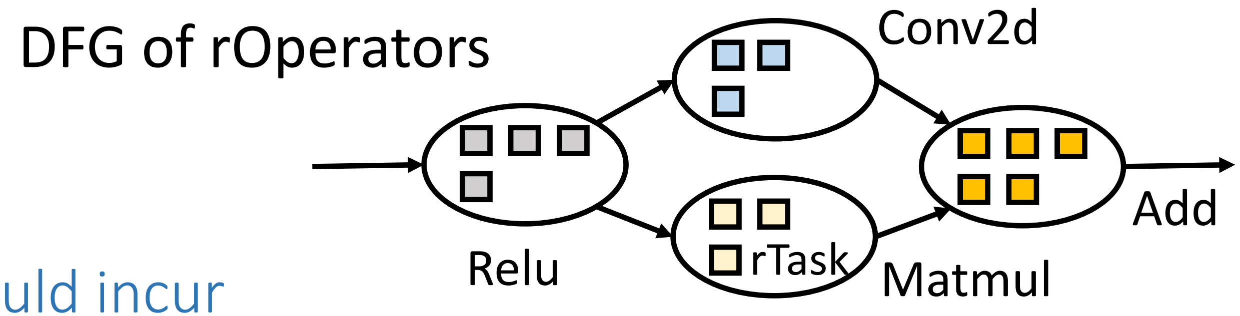
- Challenge 2: accelerators (e.g., GPU) do not expose interfaces for intra-op scheduling

- Solution: ***virtualized parallel device*** abstraction
  - Expose hardwares' fine-grained scheduling capability
  - Decouple scheduling from hardware devices
  - Bypass the hardware scheduler



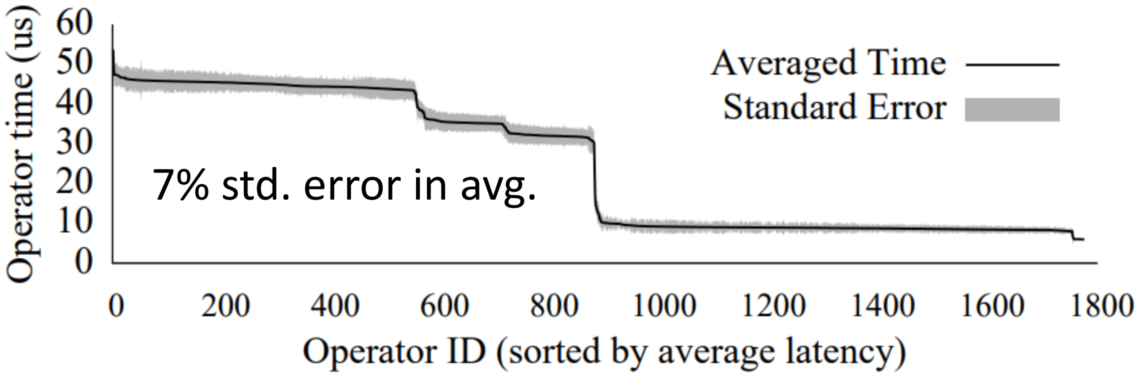


Rammer

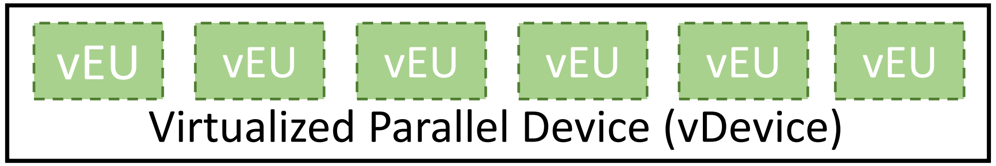


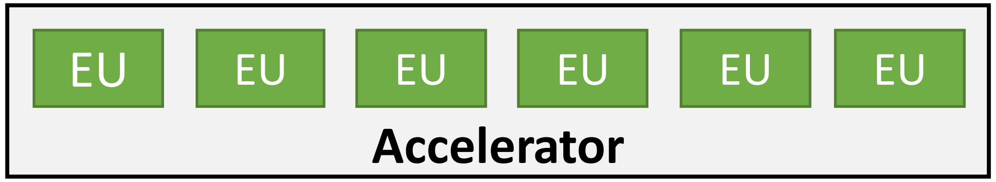
- Challenge 3: fine-grained scheduling could incur even more scheduling overheads

- Observation: ***predictability of DNN computation***
  - Most DNN's DFG is available at the ***compile time***
  - Operators exhibit ***deterministic*** performance



The profiled kernel time of all the operators in ResNeXt model. Each data point ran 1,000 times.



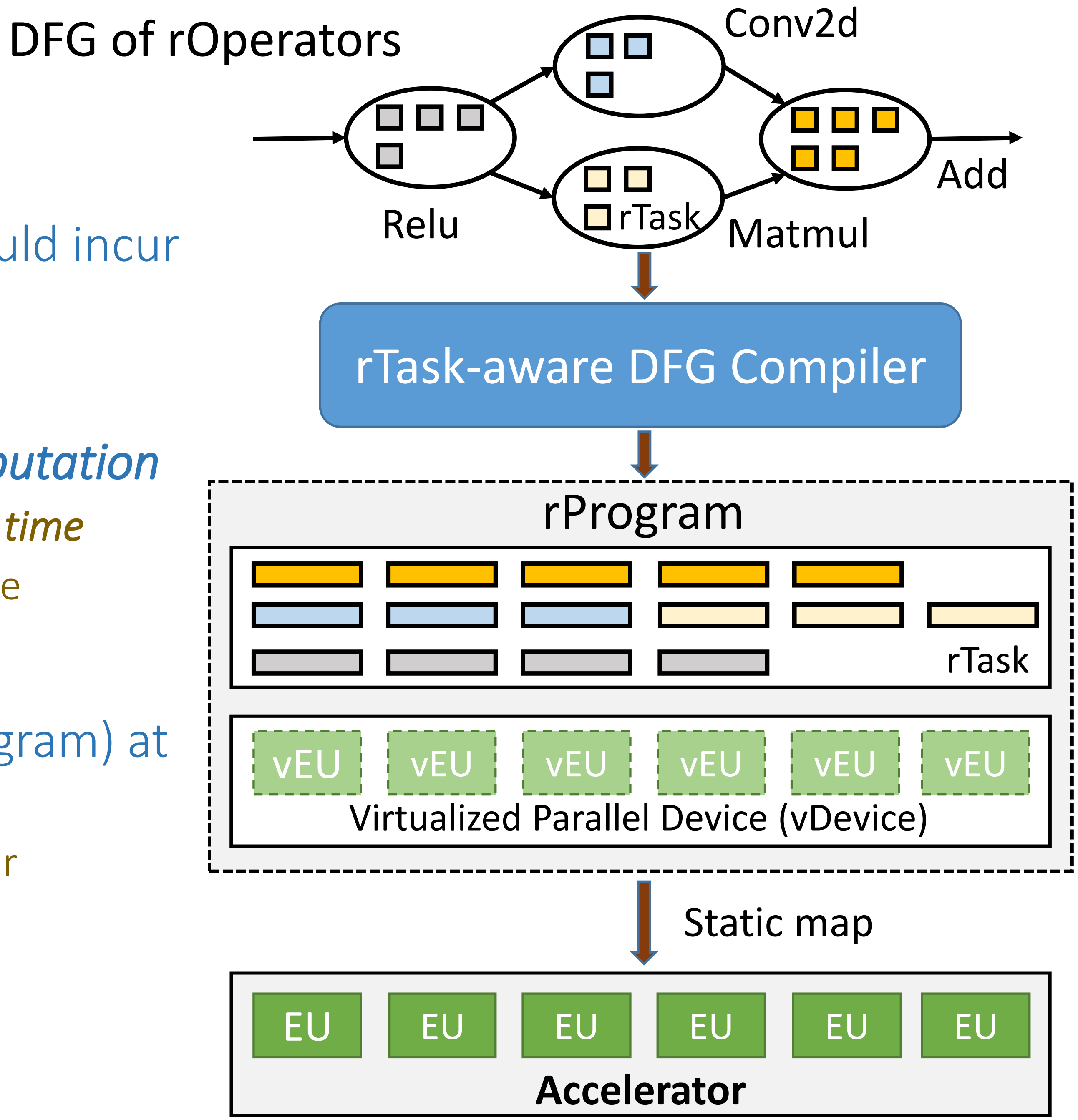


Rammer

- Challenge 3: fine-grained scheduling could incur even more runtime overheads

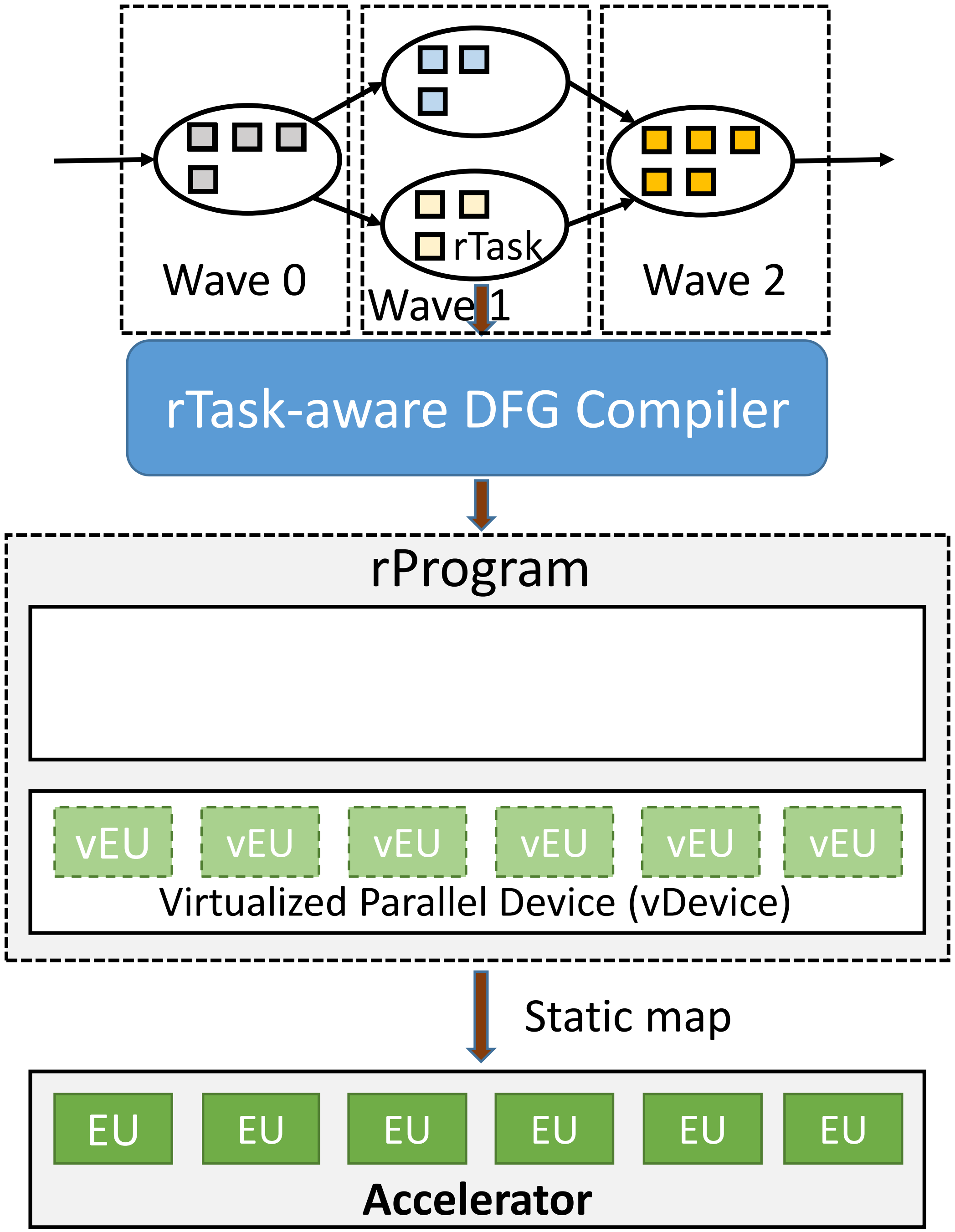
- Observation: ***predictability of DNN computation***
  - Most DNN's DFG is available at the ***compile time***
  - Operators exhibit ***deterministic*** performance

- Solution: generate execution plan (rProgram) at ***compile time***
  - **Mechanism**: scheduling interfaces & profiler
  - **Policy**: wavefront scheduling policy



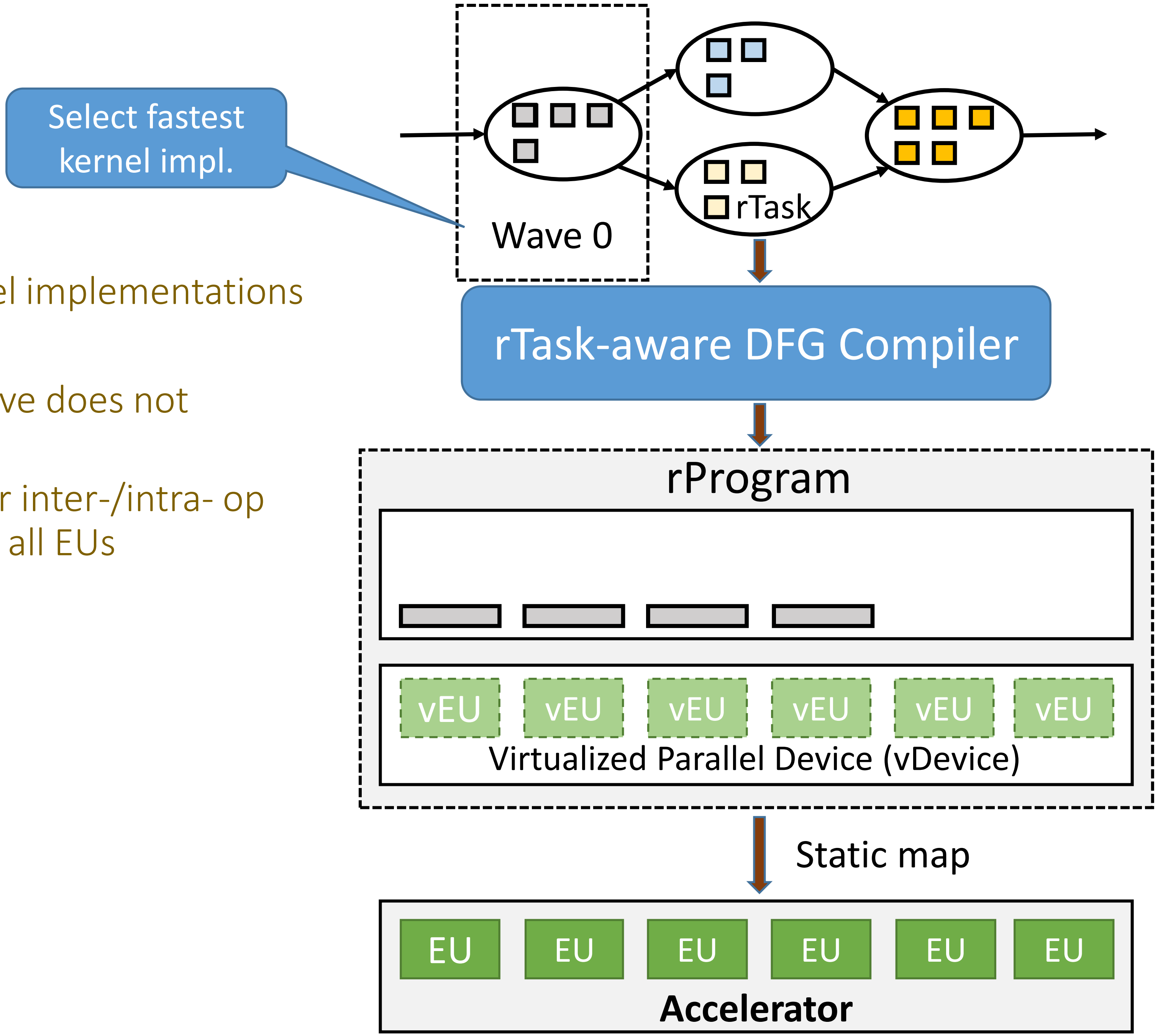
Rammer

- Wavefront scheduling policy
  - Each rOperator has different kernel implementations
  - Partition DFG into waves by BFS
  - Select ***fastest*** kernels if current wave does not saturate all EUs
  - Select ***resource-efficient*** kernels for inter-/intra- op interplay if current wave saturates all EUs



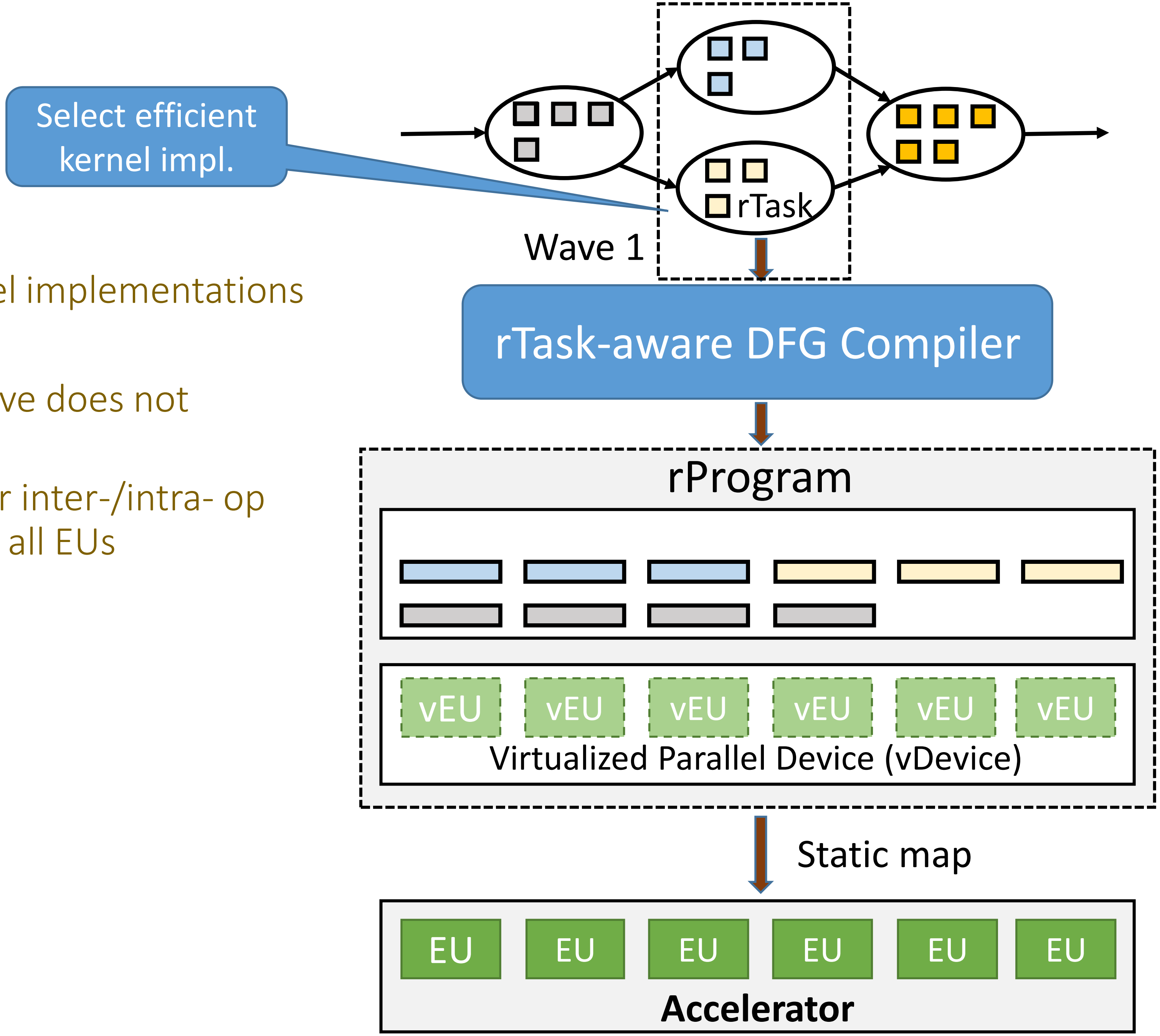
Rammer

- Wavefront scheduling policy
  - Each rOperator has different kernel implementations
  - Partition DFG into waves by BFS
  - Select ***fastest*** kernels if current wave does not saturate all EUs
  - Select ***resource-efficient*** kernels for inter-/intra- op interplay if current wave saturates all EUs



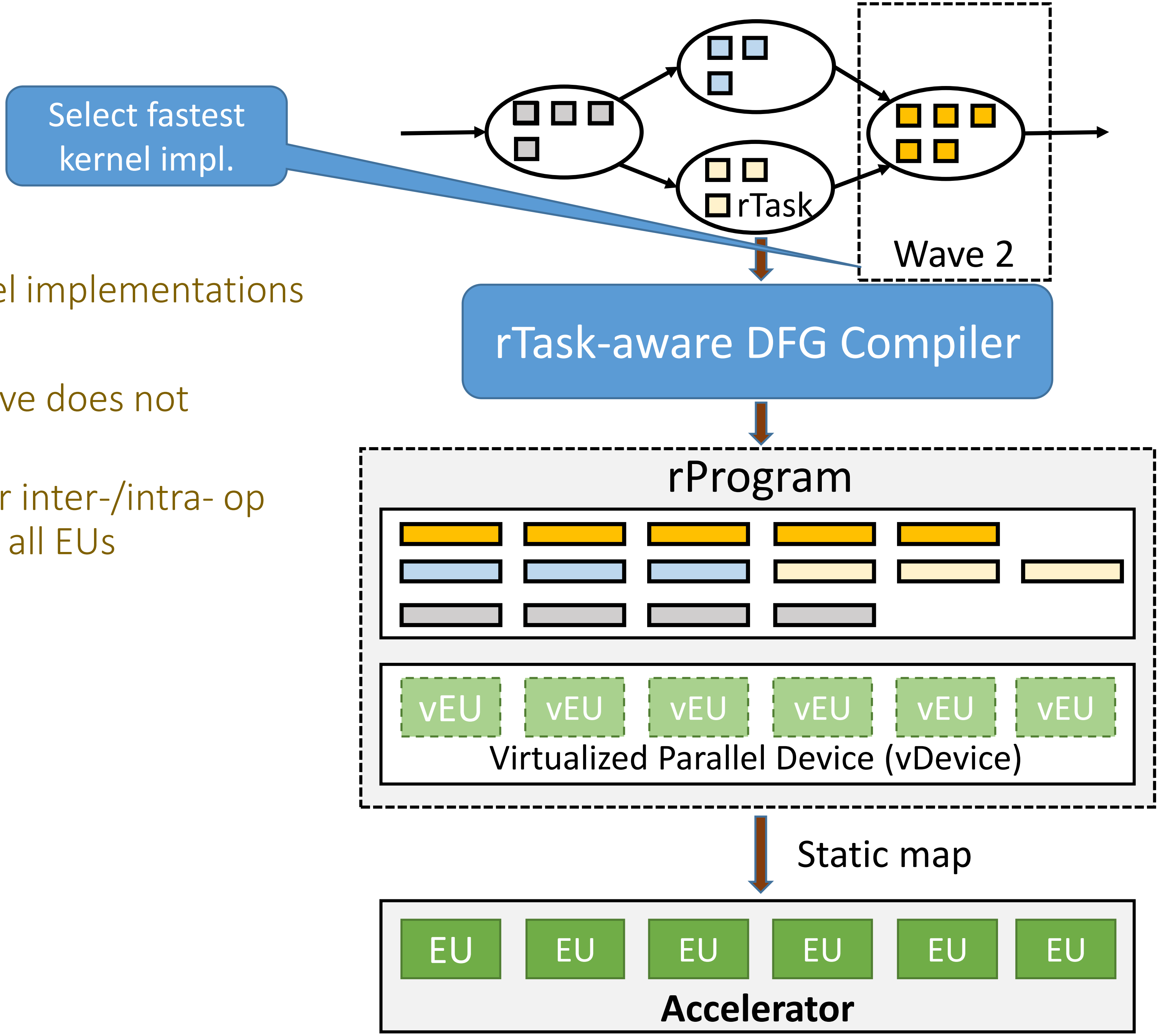
Rammer

- Wavefront scheduling policy
  - Each rOperator has different kernel implementations
  - Partition DFG into waves by BFS
  - Select ***fastest*** kernels if current wave does not saturate all EUs
  - Select ***resource-efficient*** kernels for inter-/intra- op interplay if current wave saturates all EUs

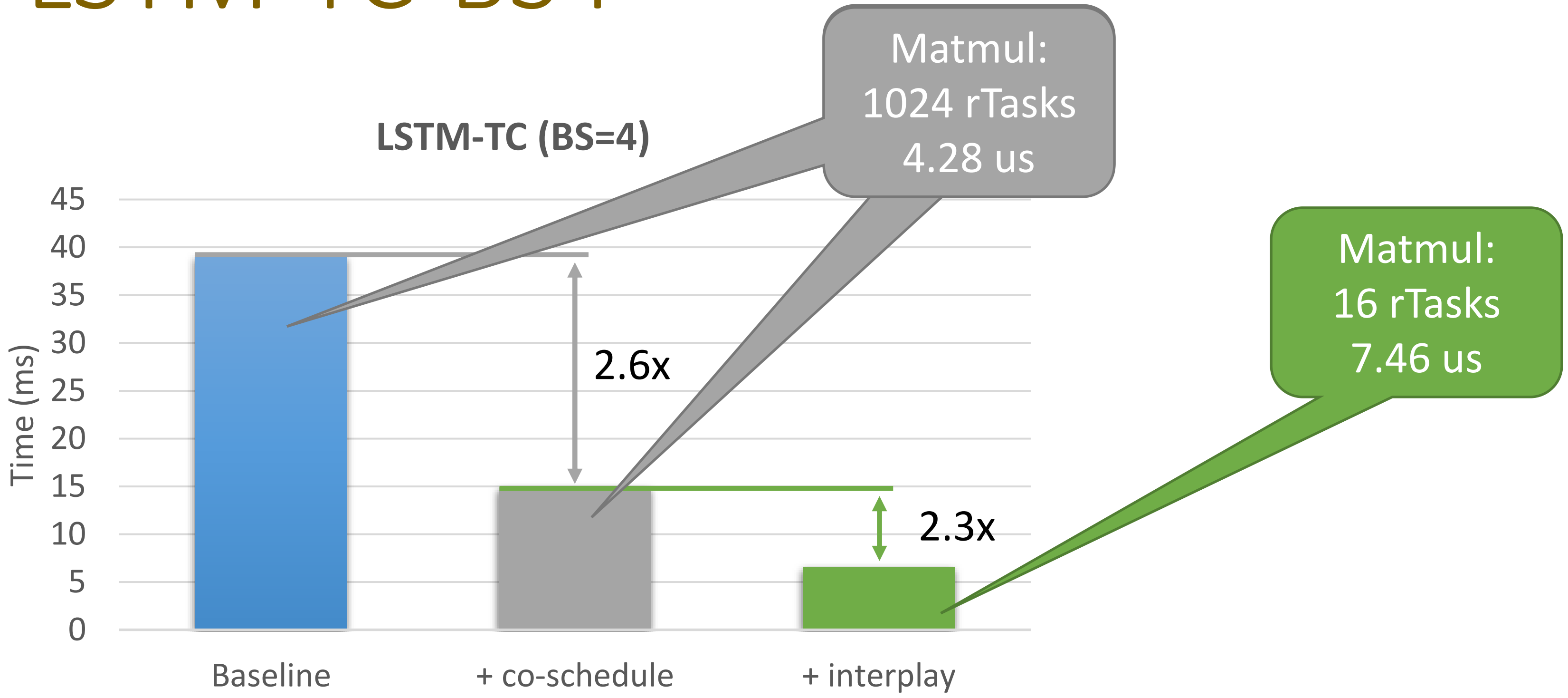


Rammer

- Wavefront scheduling policy
  - Each rOperator has different kernel implementations
  - Partition DFG into waves by BFS
  - Select ***fastest*** kernels if current wave does not saturate all EUs
  - Select ***resource-efficient*** kernels for inter-/intra- op interplay if current wave saturates all EUs



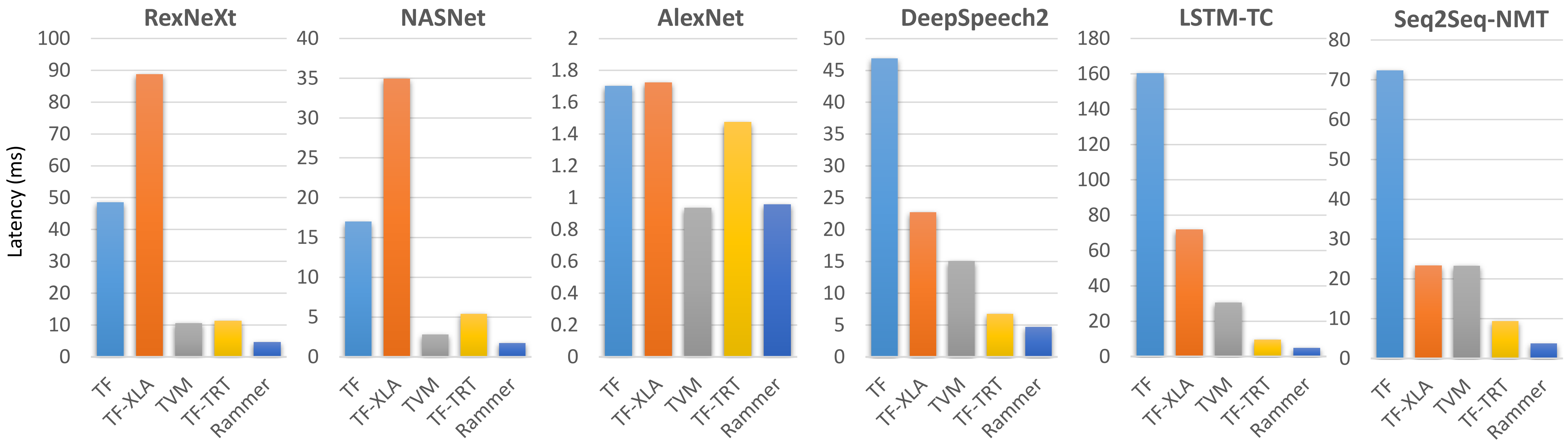
Case Study: LSTM-TC-BS4



- Baseline: two-layer architecture with compiler optimizations (e.g., kernel fusion, kernel tuning)
- + co-schedule (fastest kernels): operator co-scheduling on fastest kernels (same kernels as Baseline)
- + interplay: operator co-scheduling with interplay of inter-/intra- operator parallelism

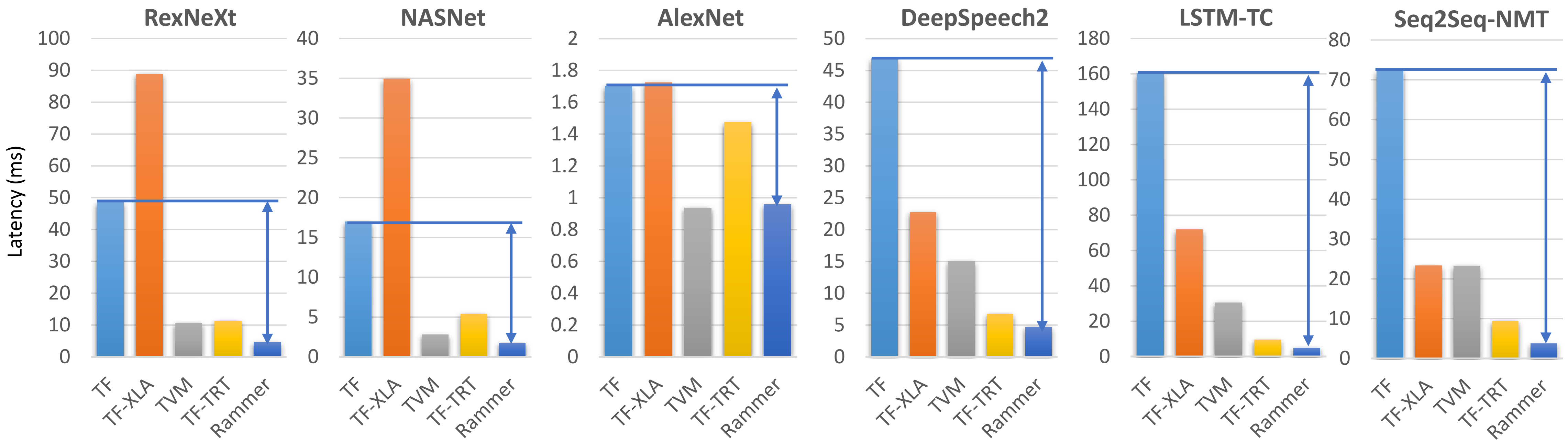
All data are reported on the inference task on a V100 GPU (more details and evaluation in paper).

End-to-end Performance on CUDA GPU



All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

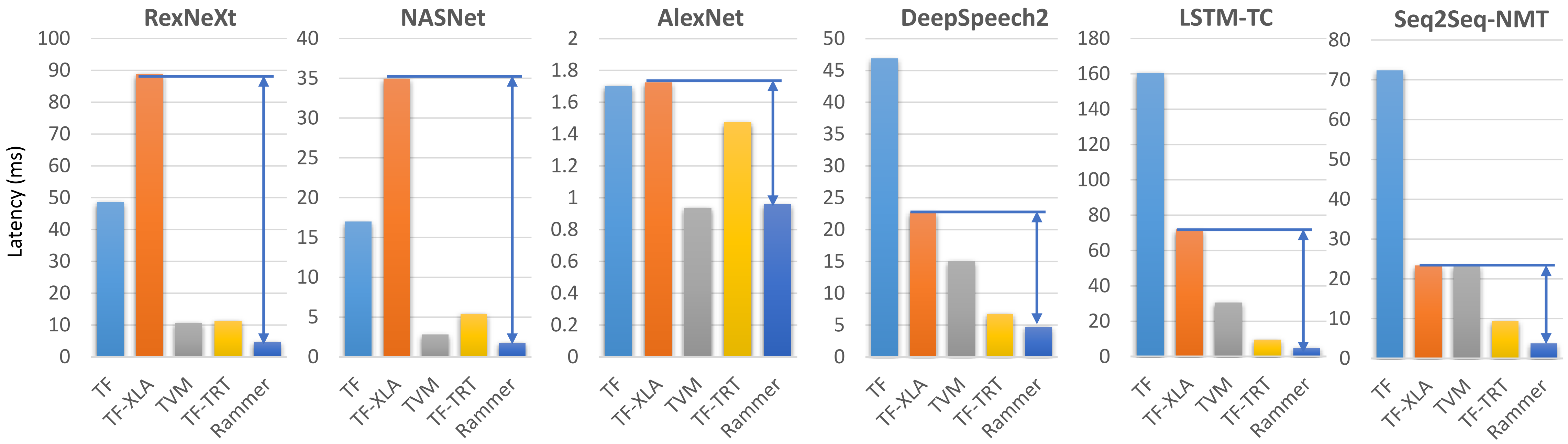
End-to-end Performance on CUDA GPU



- up to ***33.94x*** speedup over TensorFlow-1.15.2 (SOTA DL framework)

All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

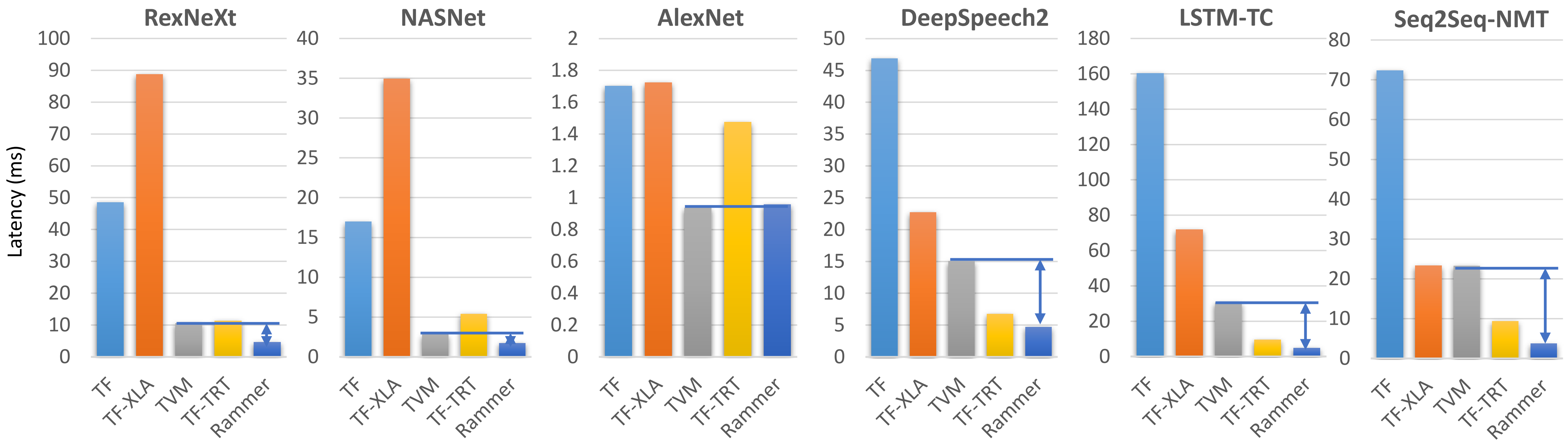
End-to-end Performance on CUDA GPU



- up to ***33.94x*** speedup over TensorFlow-1.15.2 (SOTA DL framework)
- up to ***20.12x*** speedup over TensorFlow-XLA-1.15.2 (SOTA DL compiler)

All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

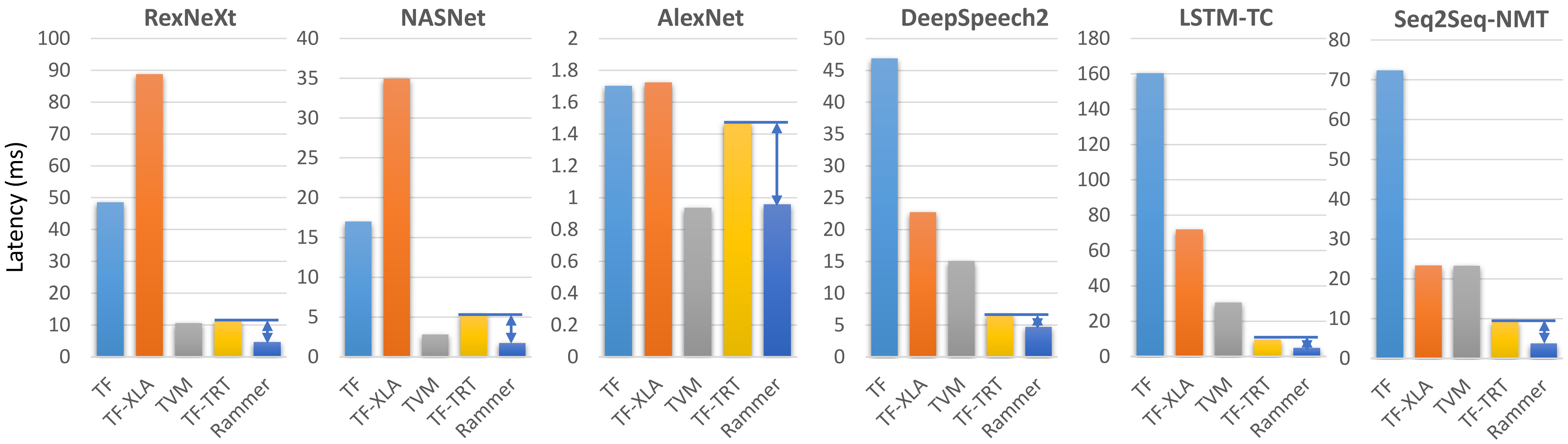
End-to-end Performance on CUDA GPU



- up to *33.94x* speedup over TensorFlow-1.15.2 (SOTA DL framework)
- up to *20.12x* speedup over TensorFlow-XLA-1.15.2 (SOTA DL compiler)
- up to ***6.46x*** speedup over TVM-0.7 (with AutoTVM) (SOTA DL compiler)

All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

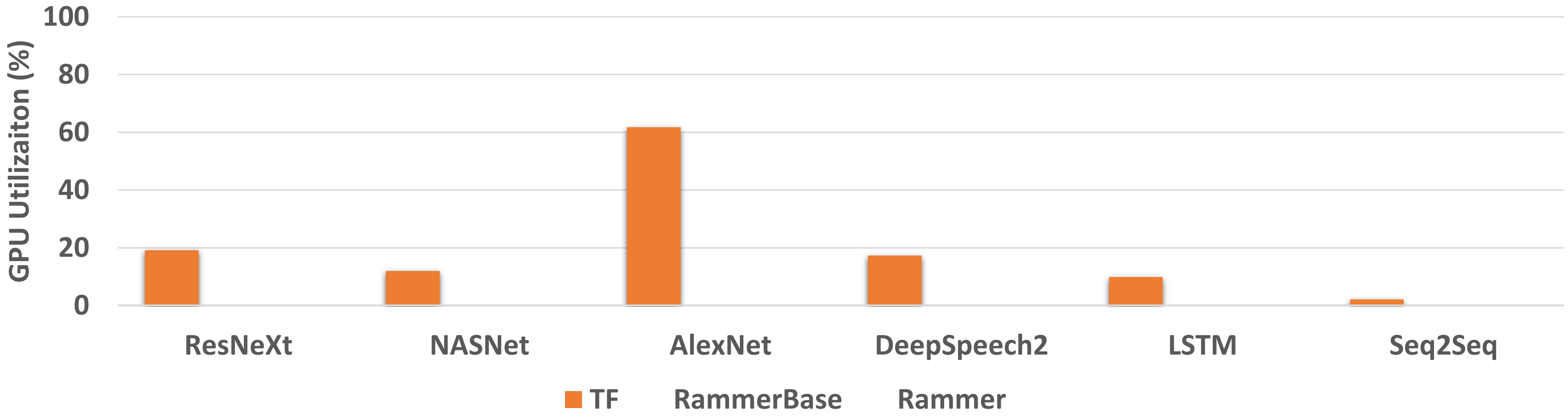
End-to-end Performance on CUDA GPU



- up to *33.94x* speedup over TensorFlow-1.15.2 (SOTA DL framework)
- up to *20.12x* speedup over TensorFlow-XLA-1.15.2 (SOTA DL compiler)
- up to *6.46x* speedup over TVM-0.7 (with AutoTVM) (SOTA DL compiler)
- up to ***3.09x*** speedup over TensorRT-7.0 (SOTA vendor optimized proprietary library)

All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

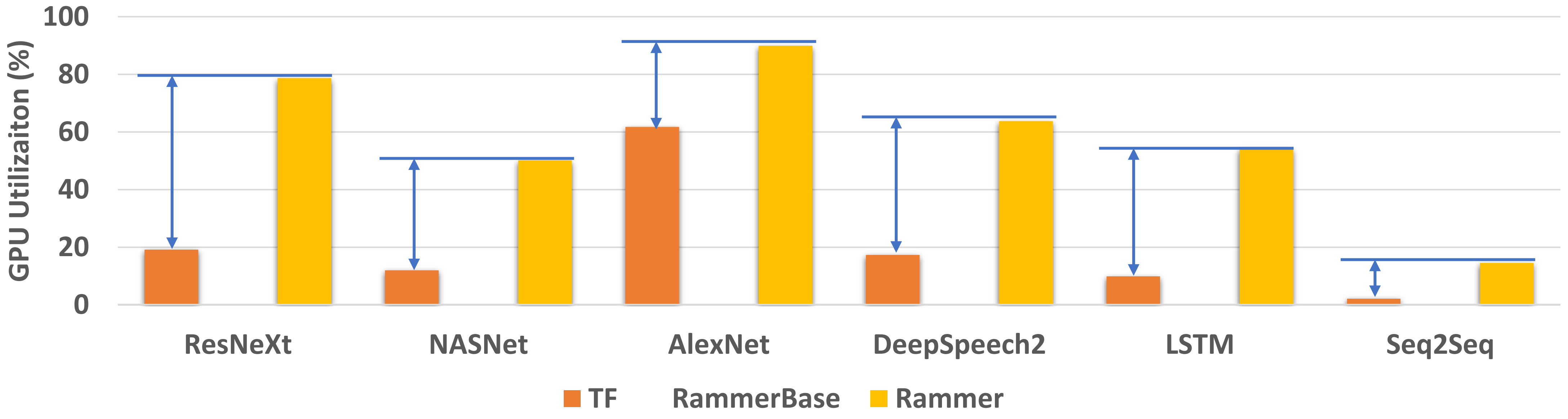
GPU Utilization



- The average GPU utilization of TensorFlow is only 20.3%

All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

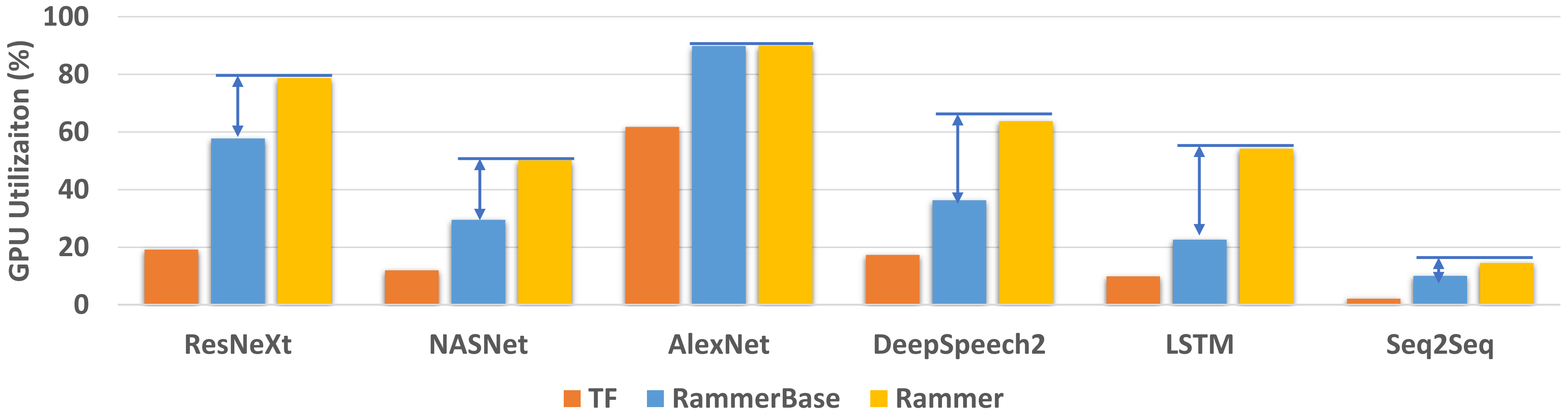
GPU Utilization



- The average GPU utilization of TensorFlow is only **20.3%**
- Rammer can improve the average GPU utilization by 4.32x

All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

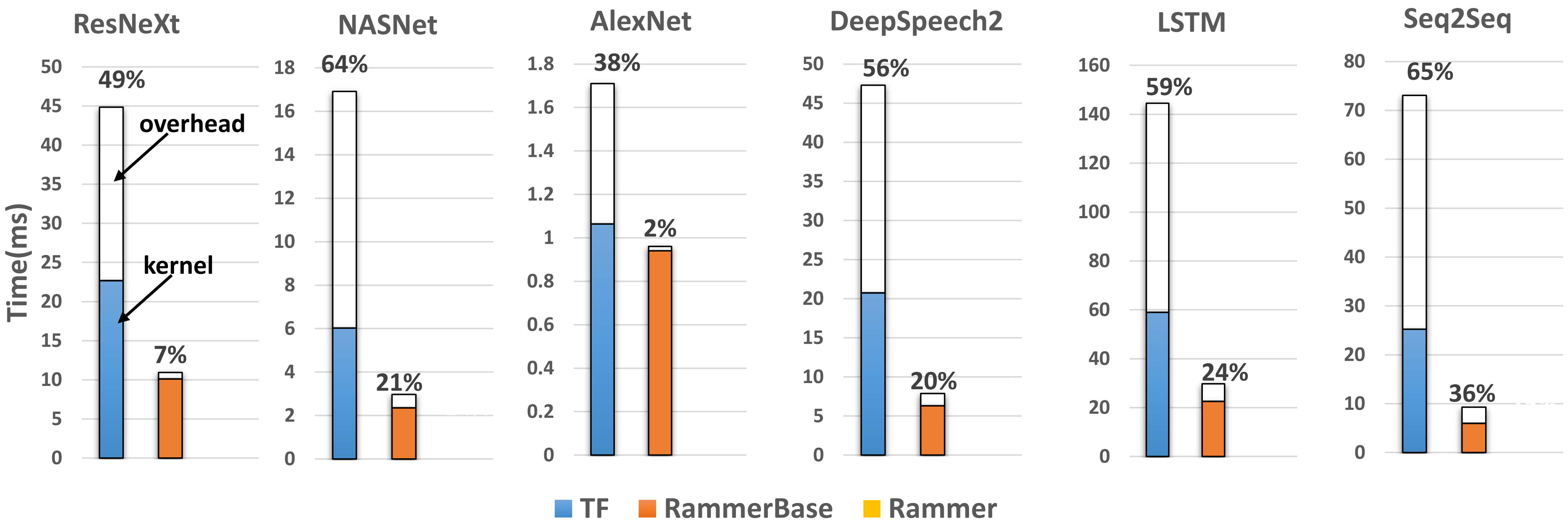
GPU Utilization



- The average GPU utilization of TensorFlow is only **20.3%**
- Rammer can improve the average GPU utilization by **4.32x**
- Compared to RammerBase, Rammer's scheduling by itself can improve the utilization by 1.61x

All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

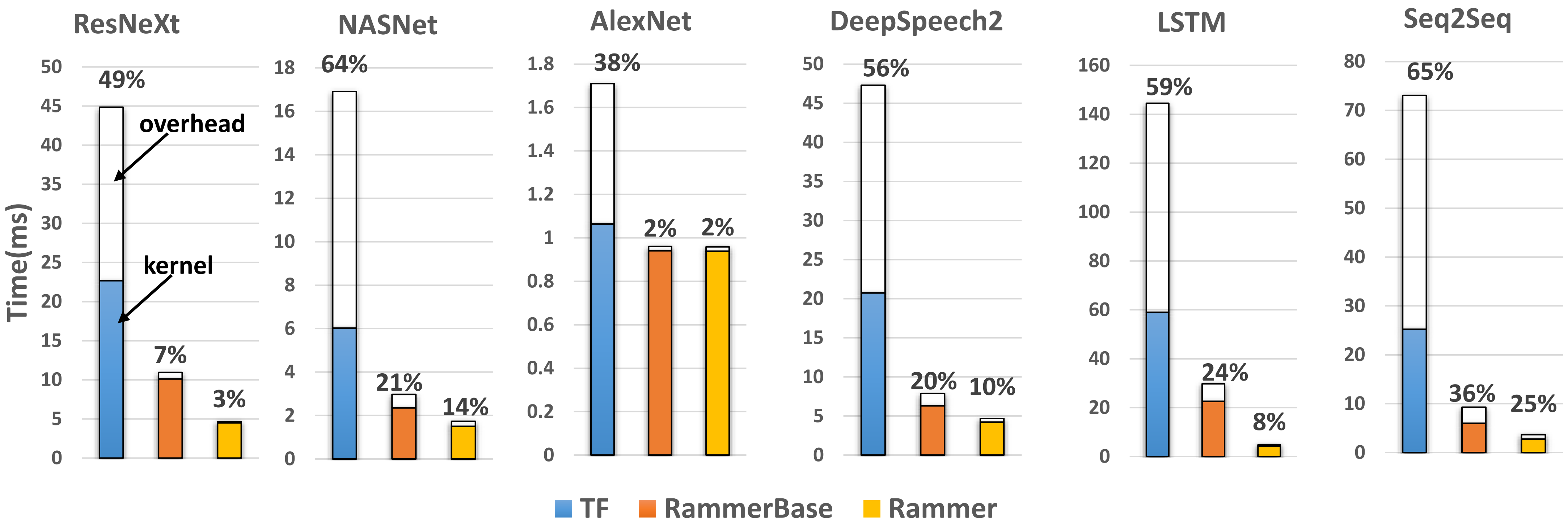
Scheduling Overhead



- RammerBase reduces avg. overhead from 32.29 ms to 2.27 ms over TensorFlow

All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

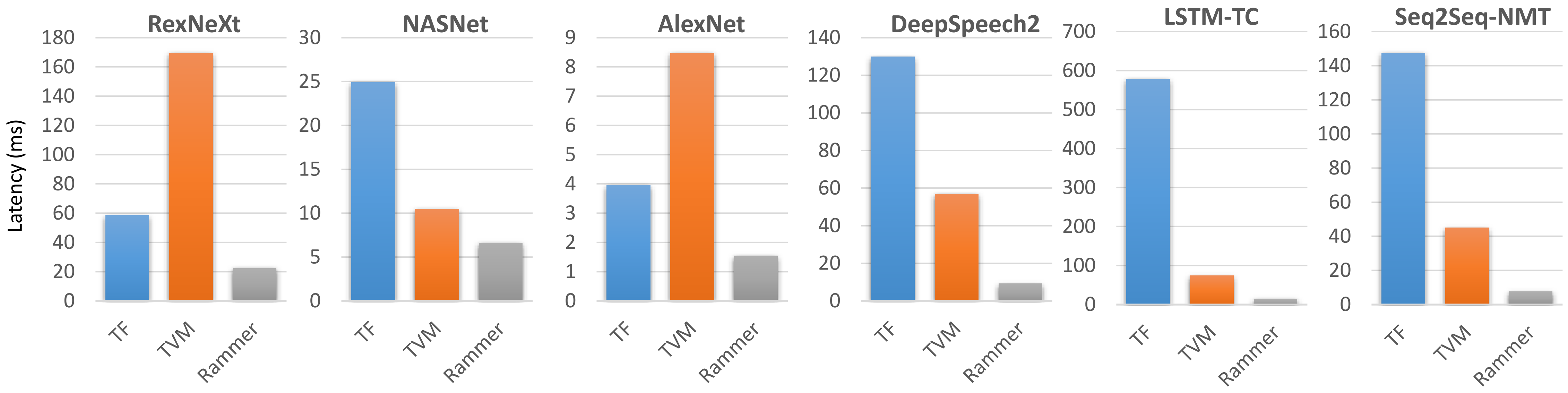
Scheduling Overhead



- RammerBase reduces avg. overhead from 32.29 ms to 2.27 ms over TensorFlow
- Rammer can further reduce avg. overhead to 0.37 ms over RammerBase

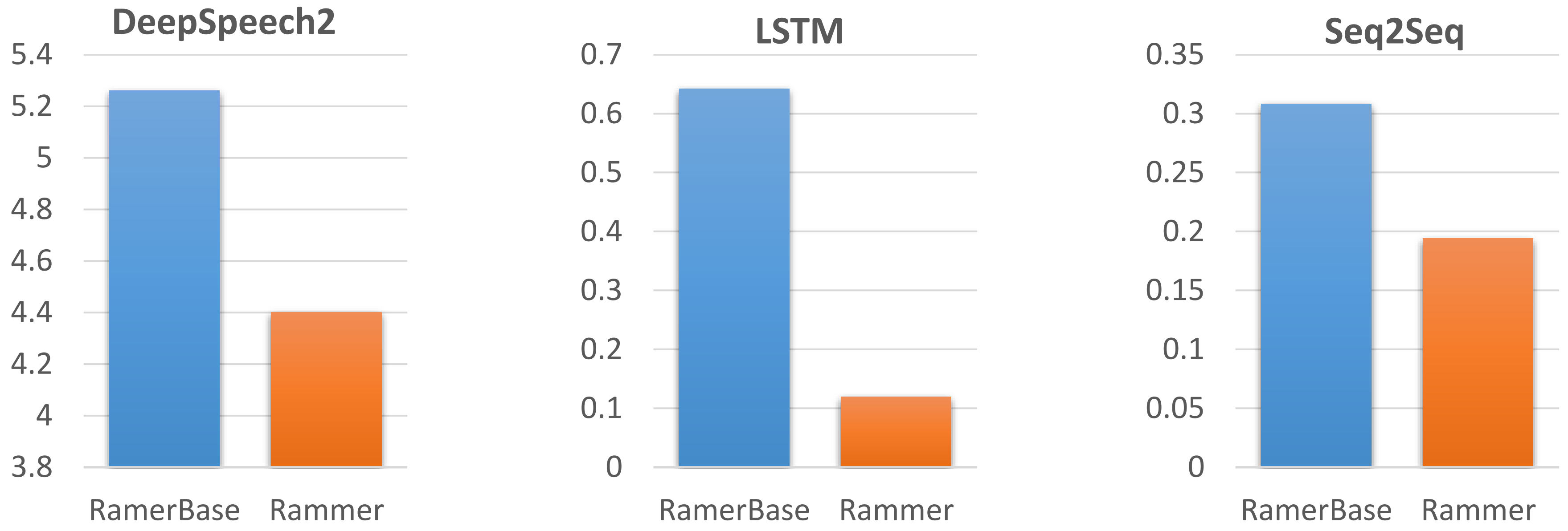
All data are reported on the inference task (BS=1) on a V100 GPU (more details and evaluation in paper).

End-to-end Performance on AMD GPU



- 13.95x speedup over TensorFlow-1.15.2 on average (SOTA DL framework)
- 5.36x speedup over TVM-0.7 on average (with AutoTVM) (SOTA DL compiler)

End-to-end Performance on GraphCore IPU



Our preliminary implementation shows:

- up to 5.37x performance improvement compared with RammerBase

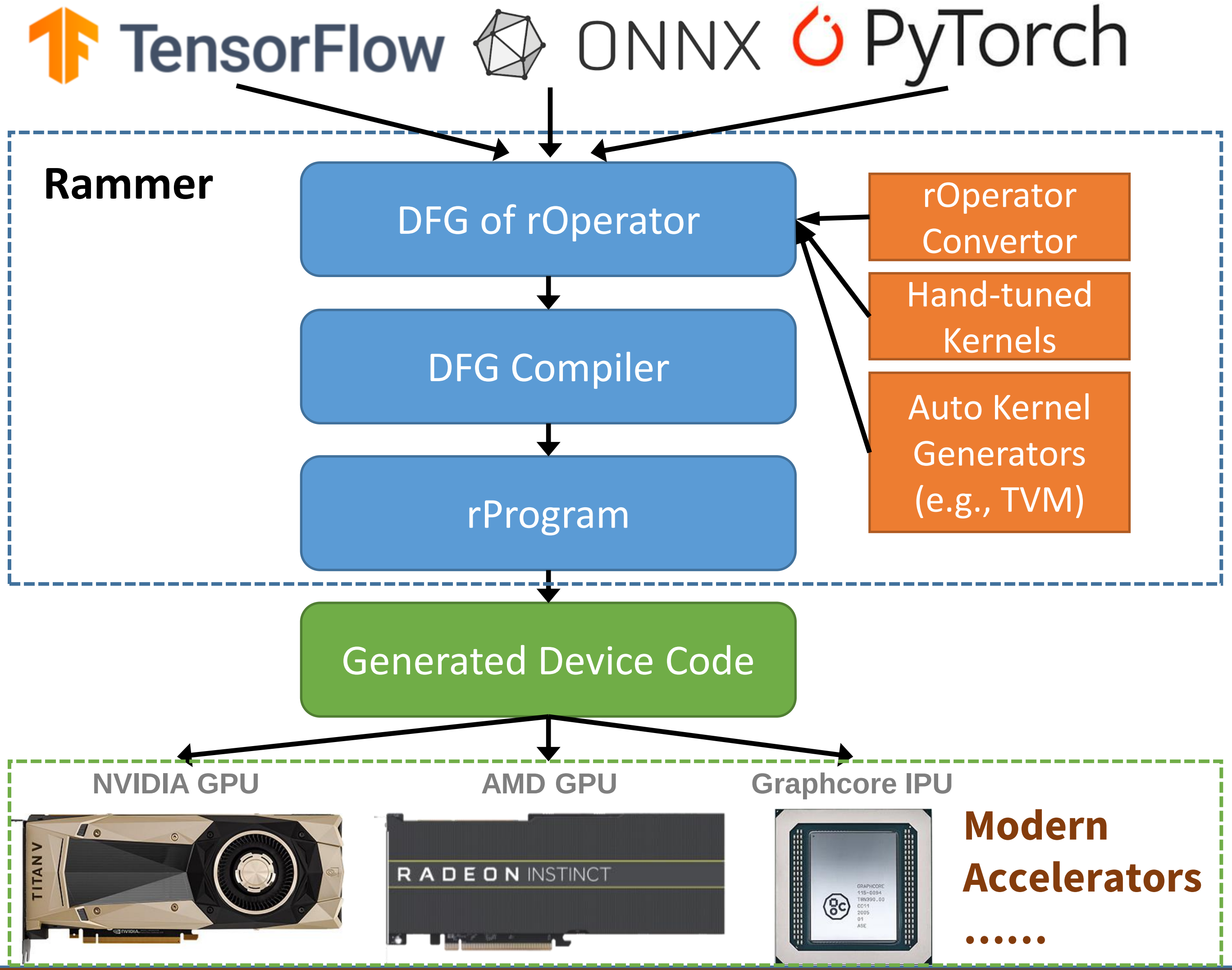
Rammer Open Source Implementation

https://github.com/microsoft/nnfusion

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED



- 52K lines of C++ code
- Support TensorFlow, ONNX, and PyTorch (TorchScript) as frontends
- Support NVIDIA GPU, AMD GPU and Graphcore IPU as backends
- More details in paper:
  - Implementation on CUDA GPU
  - Implementation on AMD ROCm GPU
  - Implementation on Graphcore IPU

Conclusion

- Rammer: holistic approach to manage the parallelism in DNN for scheduling

- Hardware neutral solution
  - ***rTask-Operator Abstraction***: expose fine-grained intra-operator parallelism
  - ***Virtualized Parallel Device***: expose hardwares' fine-grained scheduling capability

Thank you!

Check it out!

https://github.com/microsoft/nnfusion

Contact: NNFusion Team (nnfusion-team@microsoft.com)