



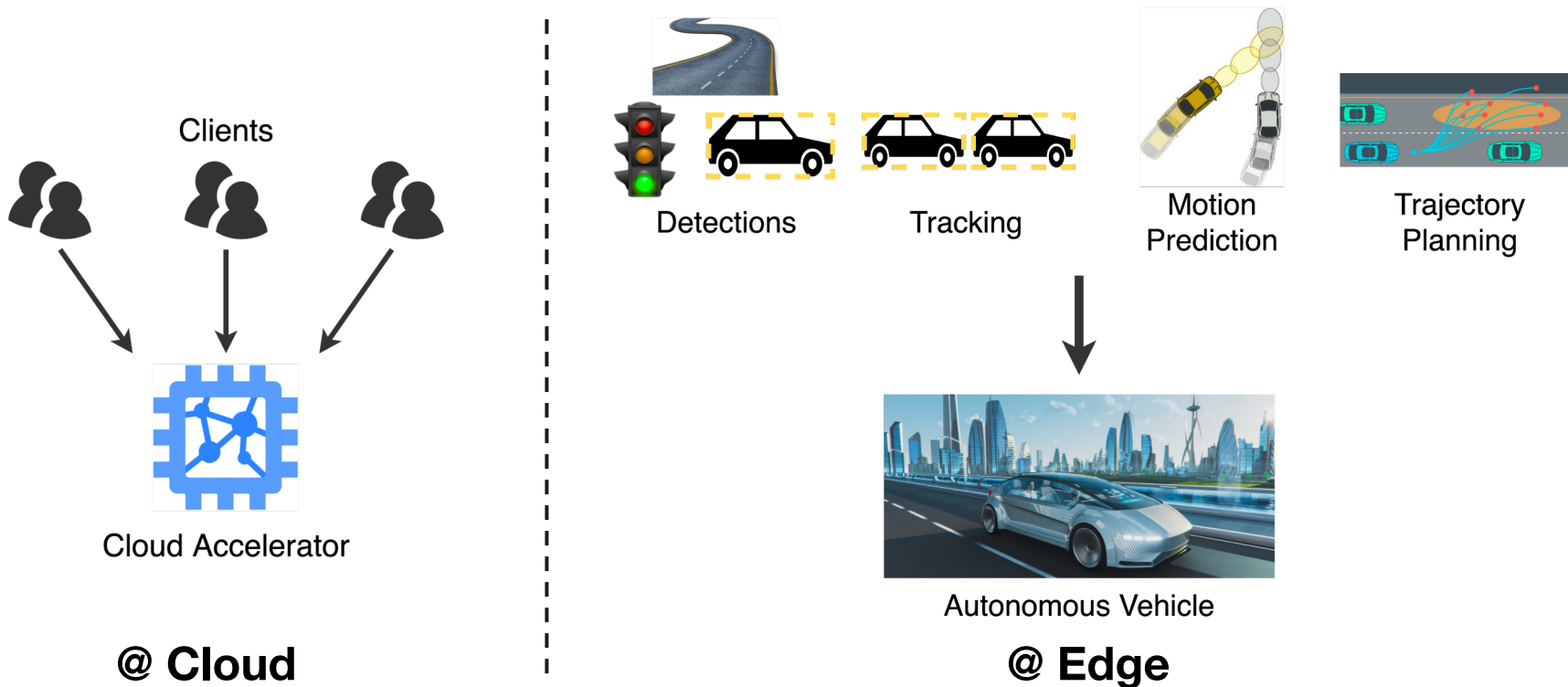
AuRORA: Virtualized Accelerator Orchestration for Multi-Tenant Workloads

Seah Kim, Jerry Zhao,
Krste Asanovic, Borivoje Nikolic, Yakun Sophia Shao



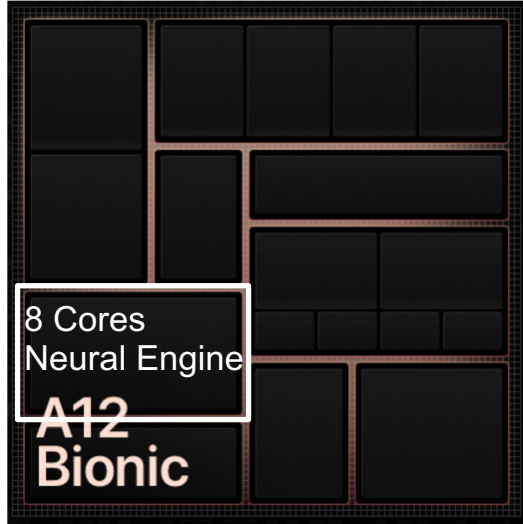
Trends in Modern SoCs: More Applications

Multiple tasks share system resources



Trends in Modern SoCs: Multi-Accelerator

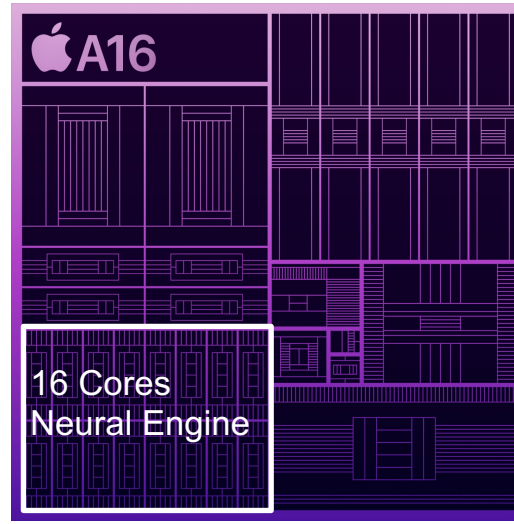
To keep up with application that are becoming more demanding...



A12 - A13: 8 cores

Apple A12 bionic

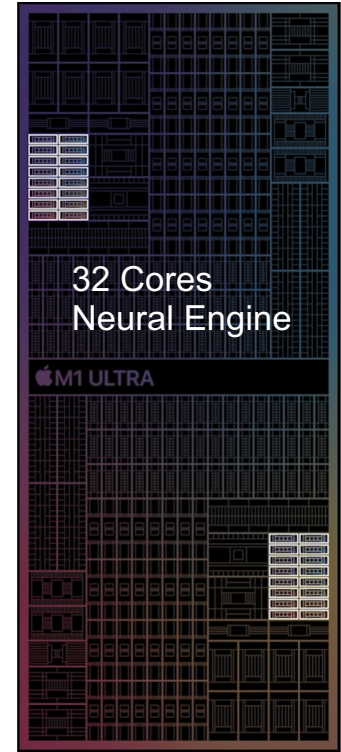
2x
➔



A14 - A16: 16 cores

Apple A16 bionic

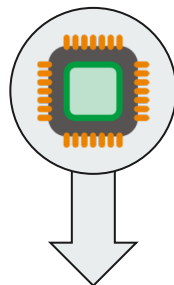
2x
➔



M1/M2 Ultra: 32 Cores

Apple M1 Ultra 3

Trends in Modern SoCs



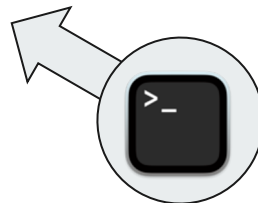
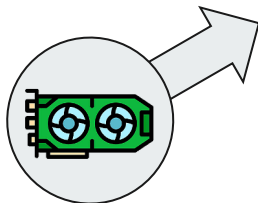
More cores

- End of single-thread performance scaling
- Many-core SoCs to extract TLP

How to scalably architect
many-accelerator SoCs?

More accelerators

- More compute-bound workloads require acceleration

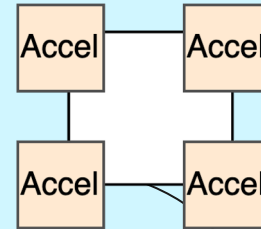


More applications

- Software stacks grow in complexity
- Graphics/multimedia/AI are pervasive

Trends in Modern SoCs

How to architect many-accelerator SoCs?



More accelerators

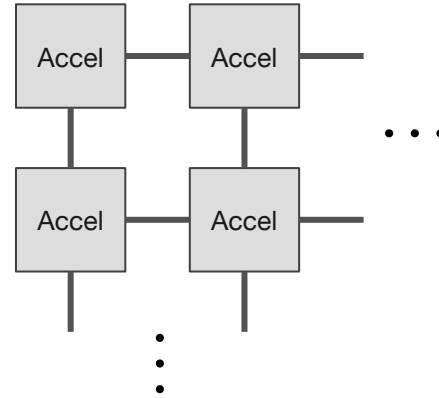
- More compute-bound workloads require acceleration

More applications

- Software stacks grow in complexity
- Graphics/multimedia/AI are pervasive

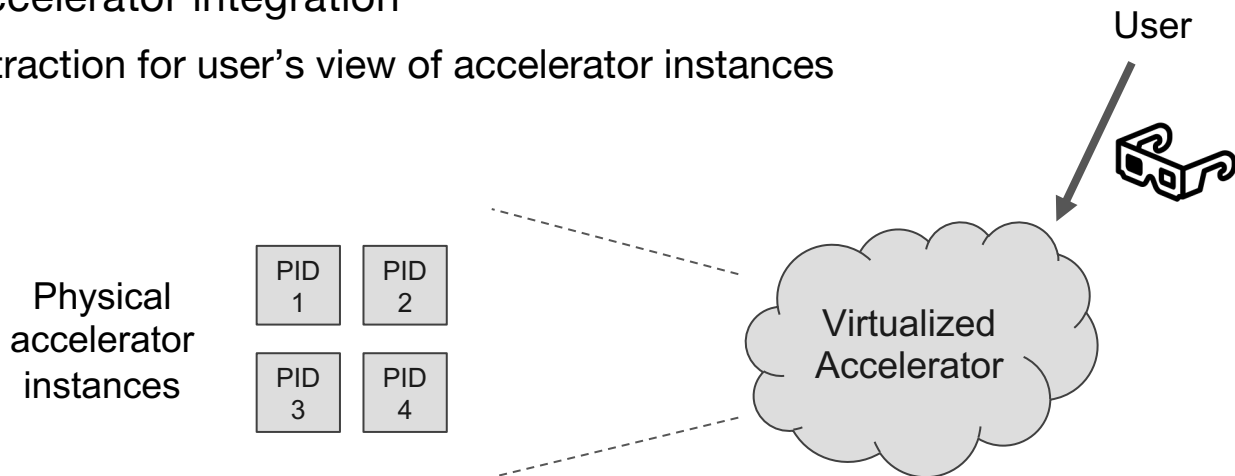
Requirements for Accelerator Integration

- Goal: Enable scalable **many-accelerator** for **multi-tenant execution**
- Requirements:
 - Scalable deployment
 - Many-accelerator integration



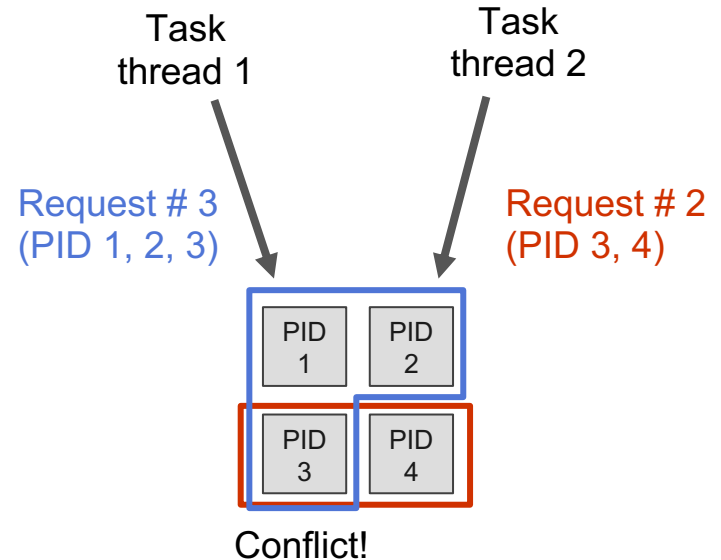
Requirements for Accelerator Integration

- Goal: Enable scalable **many-accelerator** for **multi-tenant execution**
- Requirements:
 - Scalable deployment
 - Many-accelerator integration
 - Virtual accelerator integration
 - Abstraction for user's view of accelerator instances
 - **???**



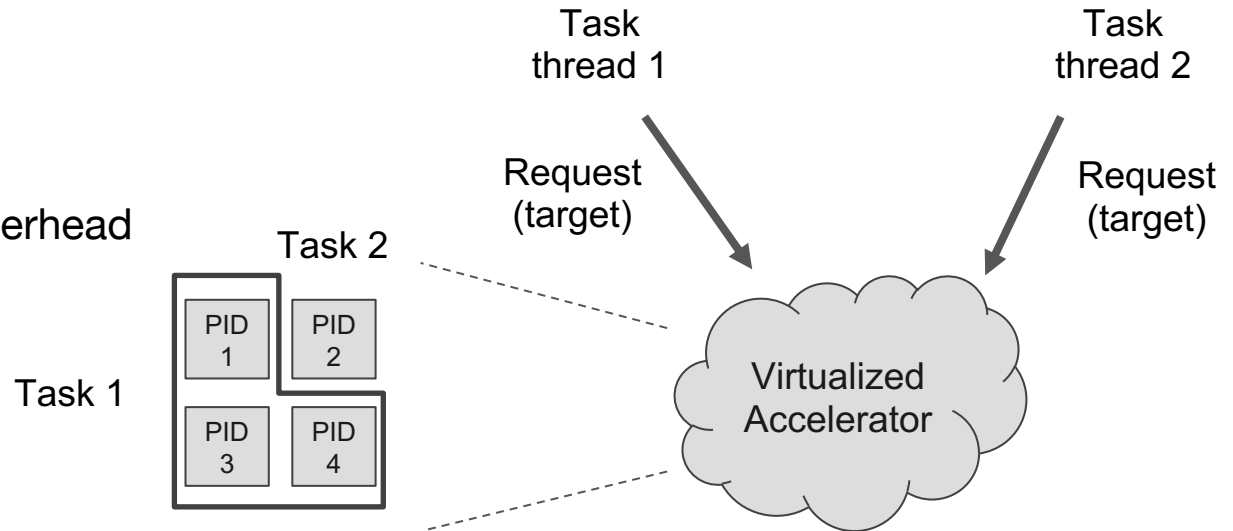
Physical Accelerator Integration

- Program physical accelerator resources
 - Request accelerator using physical ID (PID)
- Issues under multi-tenancy
 - Resource conflict
 - Hard to repartition resource frequently
 - Cause stall: Low utilization
 - Programming burden



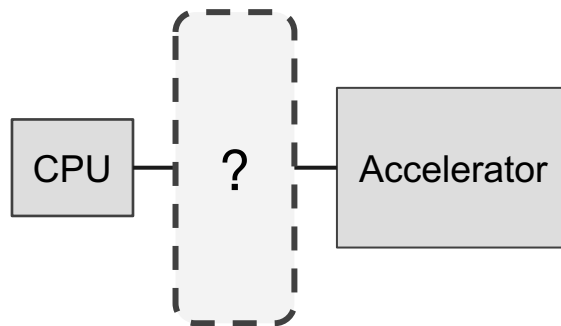
Virtual Accelerator Integration

- Provides an abstraction between ...
 - User's view of accelerator
 - Physical accelerator instances
- Enable scalable many-accelerator for multi-tenancy
- Requirements ...
 - Low latency
 - Programmable
 - Minimize SW overhead



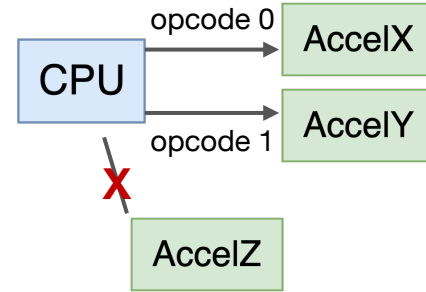
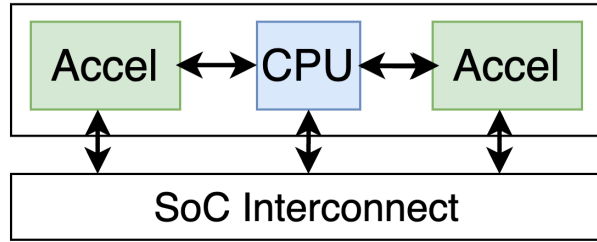
Requirements for Accelerator Integration

- Goal: Enable scalable **many-accelerator** for **multi-tenant execution**
- Requirements:
 - Scalable deployment
 - Virtual accelerator integration
 - Low latency
 - Programmable
 - Minimal programming overhead



- Interface: How accelerator interacts with host CPU

Existing Physical Accelerator Integration Methodologies



Tightly CPU-coupled:

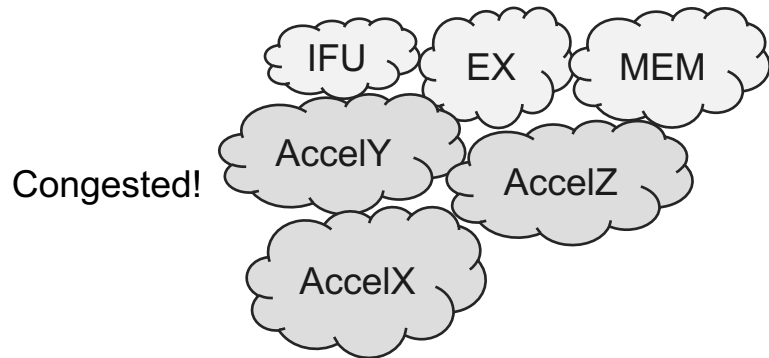
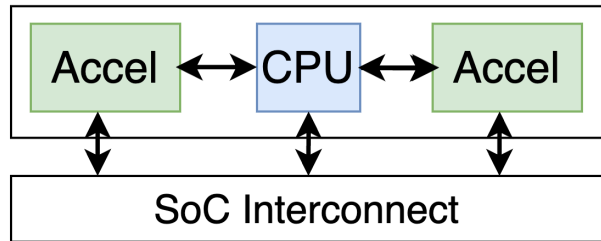
Limited opcode space



Limited accelerator per core

Physically attached to CPU

Existing Physical Accelerator Integration Methodologies



Tightly CPU-coupled:

Limited opcode space



Limited accelerator per core

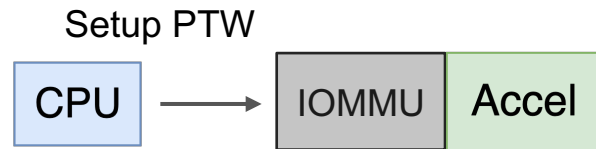
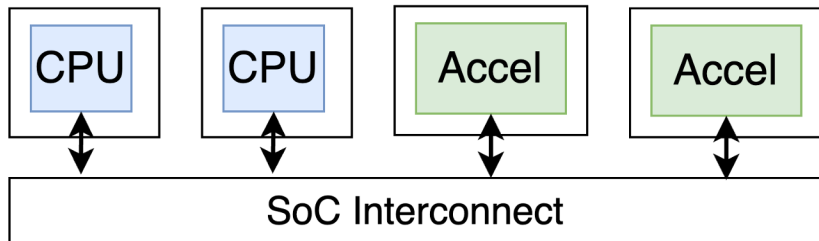
Physically attached to CPU



Physical design challenge

Scalability issue

Existing Physical Accelerator Integration Methodologies



Memory-mapped over interconnect:

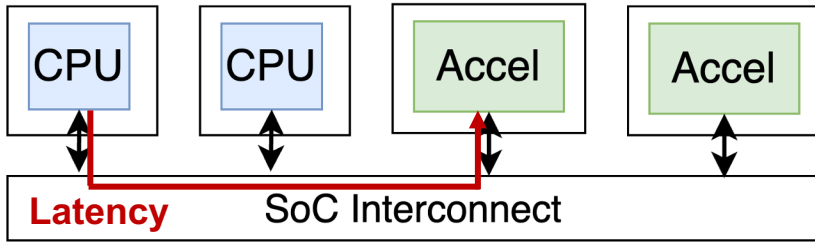
Setup accelerator IOMMU for address translation



Software complexity

Access accelerator over system bus, memory hierarchy

Existing Physical Accelerator Integration Methodologies



Memory-mapped over interconnect:

Setup accelerator IOMMU for address translation



Software complexity

Access accelerator over system bus, memory hierarchy



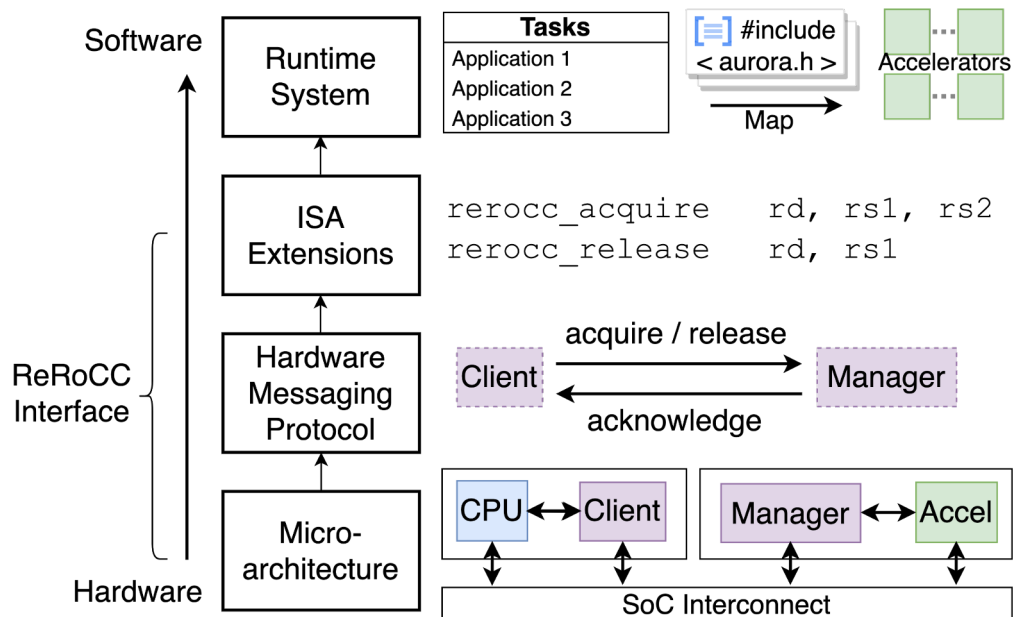
Latency overhead

Virtual integration difficulties

AuRORA: Virtual Accelerator Integration and Orchestration

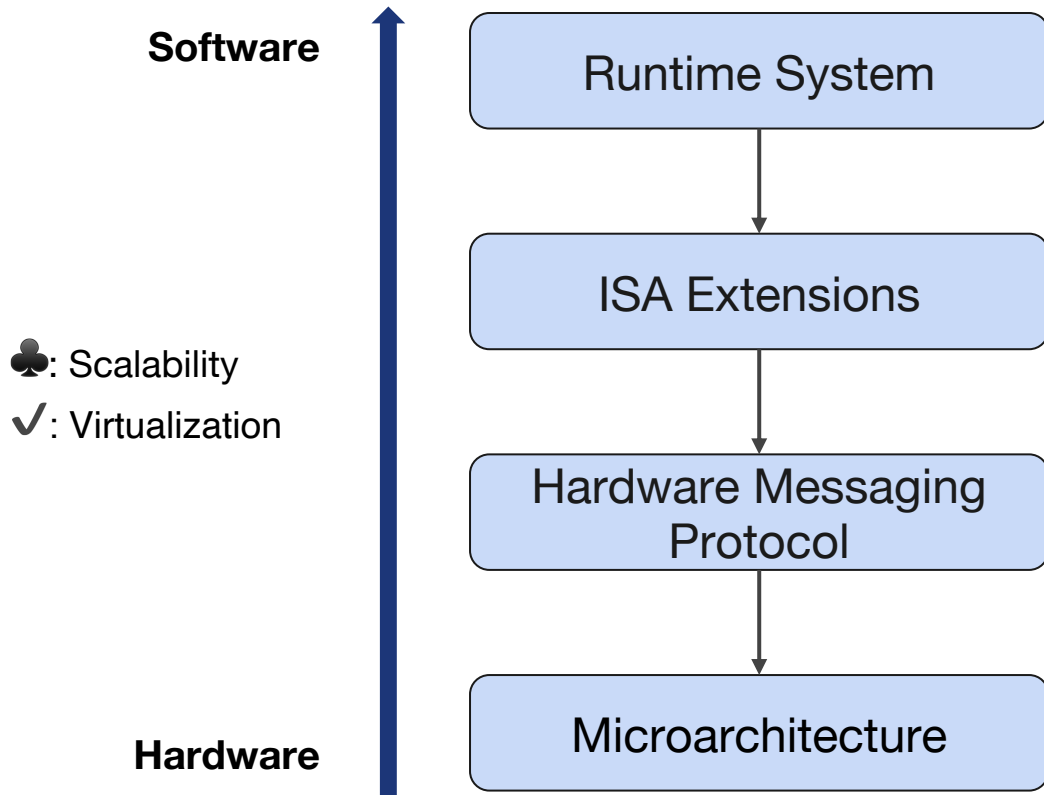
A full-stack system enabling **scalable deployment** and **virtualized integration** of accelerators

♣ : Scalability
✓ : Virtualization

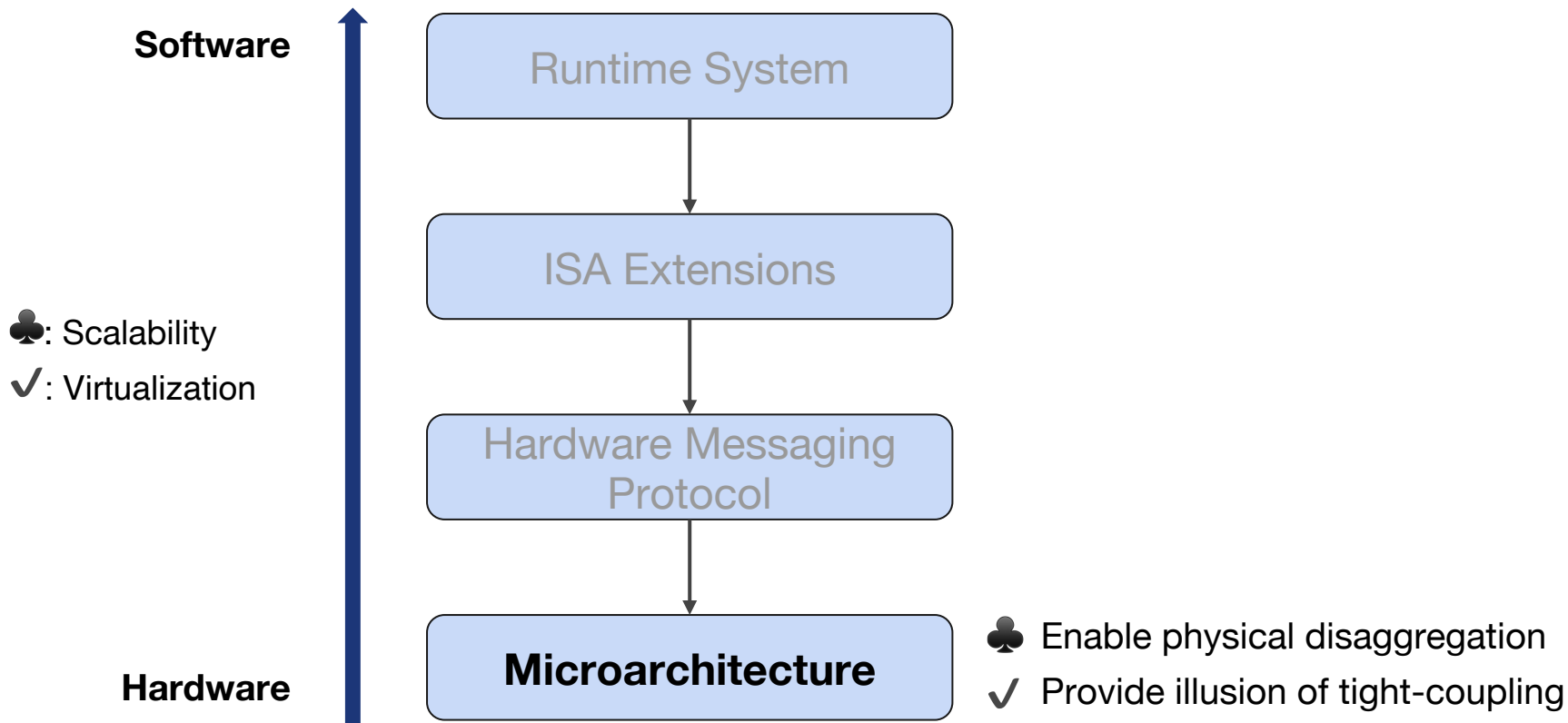


- ✓ Virtualized accelerator management
- ♣ Enable acquiring many-accelerators
- ✓ Enable programmable virtual interface
- ✓ Low latency
- ✓ Enable virtual to physical mapping
- ♣ Enable physical disaggregation
- ✓ Provide illusion of tight-coupling

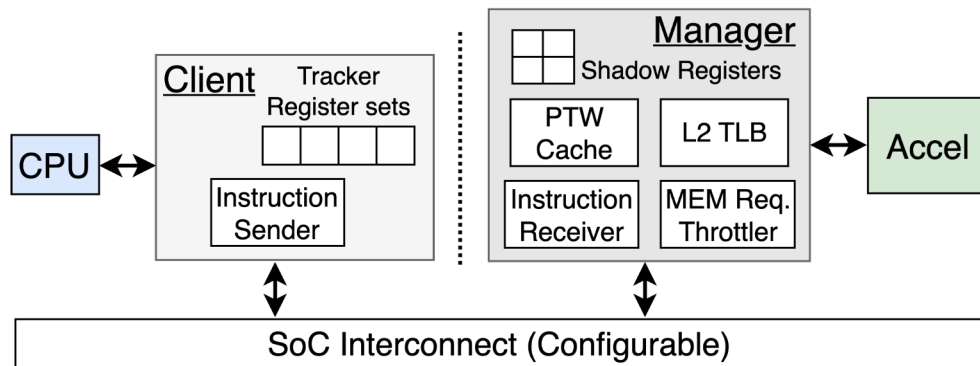
AuRORA Full Stack Implementation



AuRORA Full Stack Implementation



AuRORA Microarchitectural Components



AuRORA Client:

Attaches to CPUs via RoCC

Forwards accelerator instructions to acquired manager

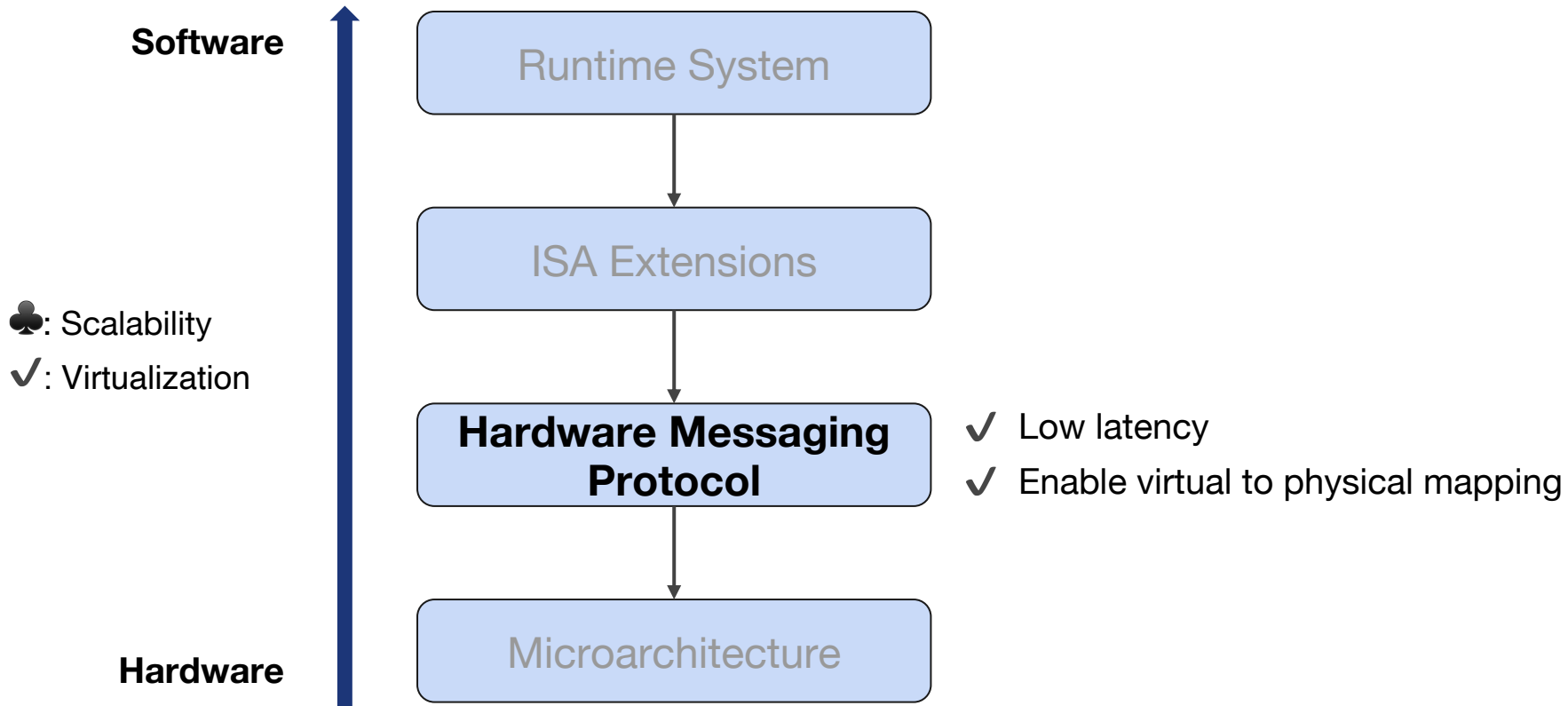
AuRORA Manager:

Attaches to existing RoCC accelerators

Shadow thread architectural state

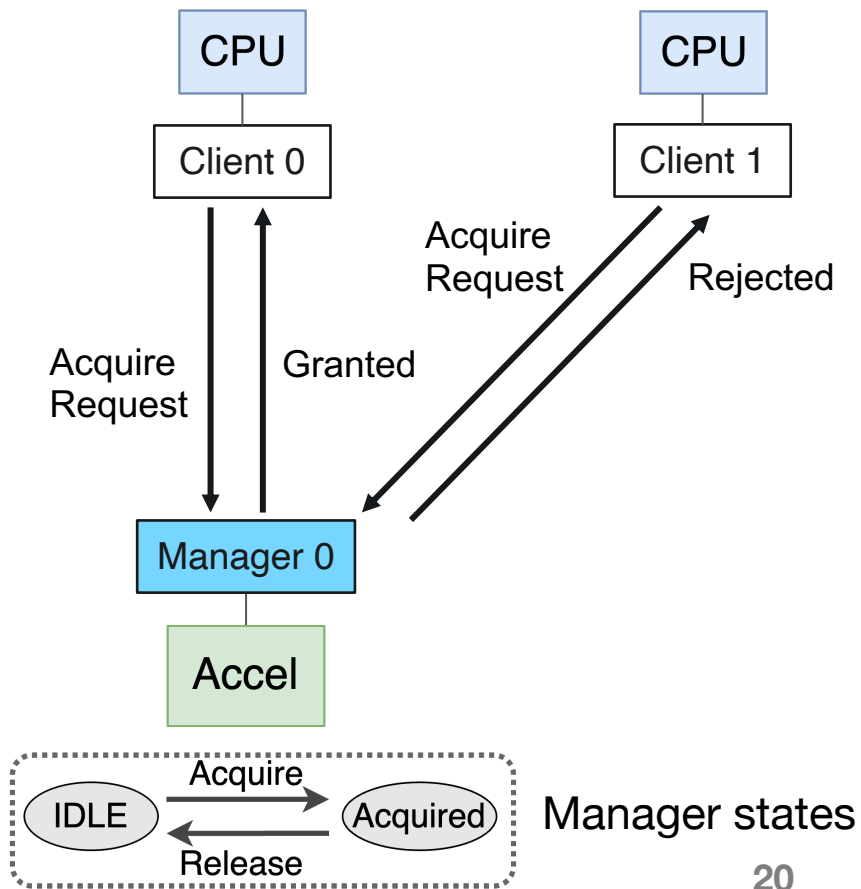
Eliminate need of user-/supervisor-managed IOMMU

AuRORA Full Stack Implementation



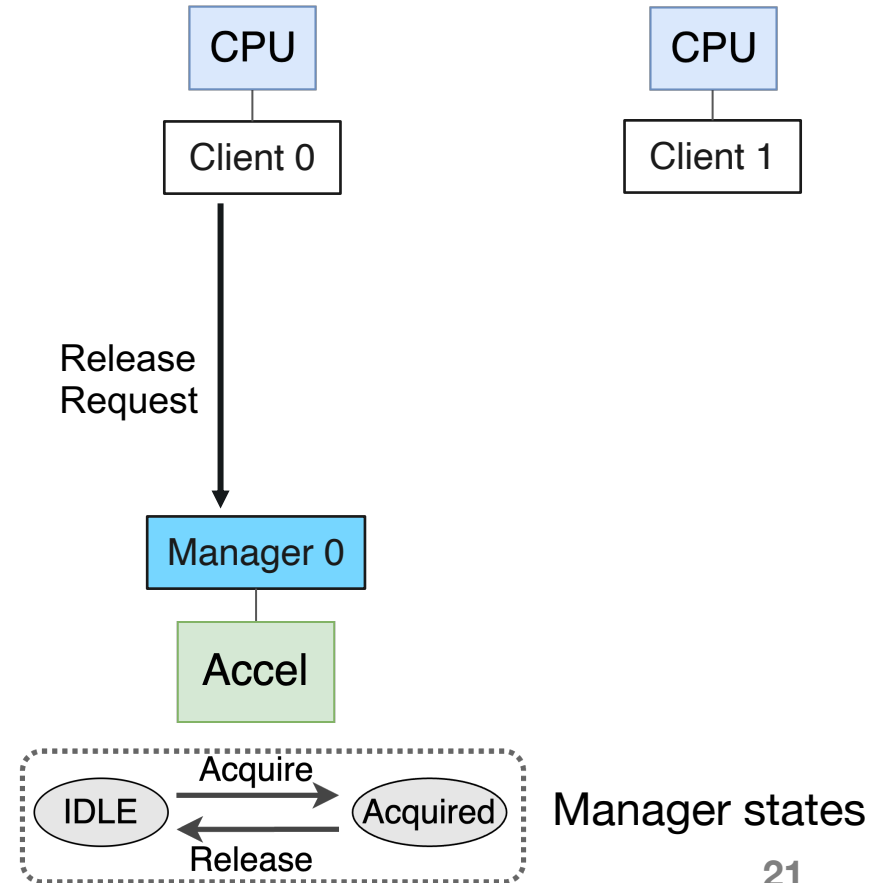
AuRORA Hardware Messaging Protocol

- Manages communication between CPUs and Accelerators
- Protocol supports:
 - Client-manager synchronization
 - Maintenance of shadowed architectural state on managers
 - Client-to-manager instruction forwarding
- Physical transport layer: network-on-chip interconnect

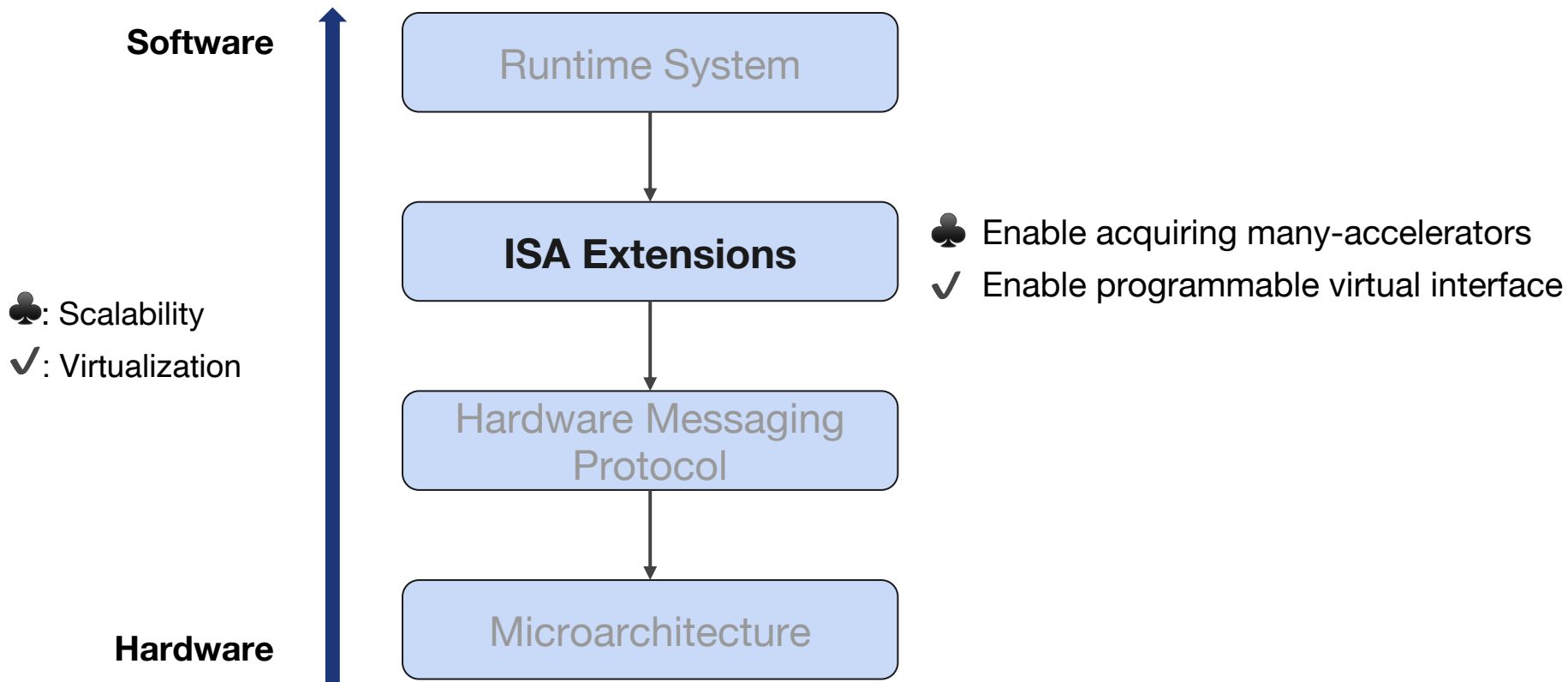


AuRORA Hardware Messaging Protocol

- Manages communication between CPUs and Accelerators
- Protocol supports:
 - Client-manager synchronization
 - Maintenance of shadowed architectural state on managers
 - Client-to-manager instruction forwarding
- Physical transport layer: network-on-chip interconnect



AuRORA Full Stack Implementation



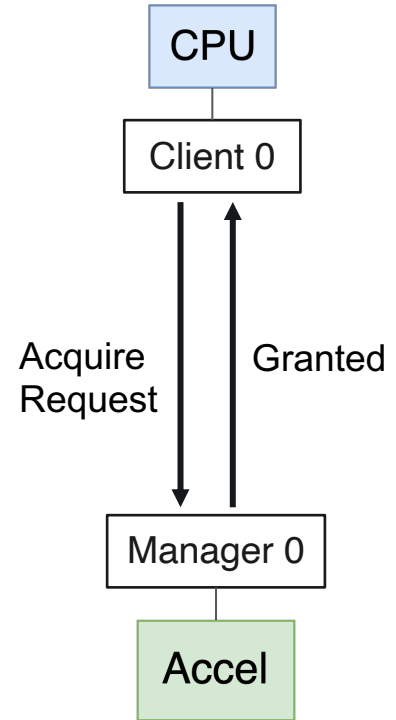
AuRORA ISA Extensions

AuRORA PseudInst.
rerocc_acquire
rerocc_assign
rerocc_release
rerocc_fence
rerocc_memrate

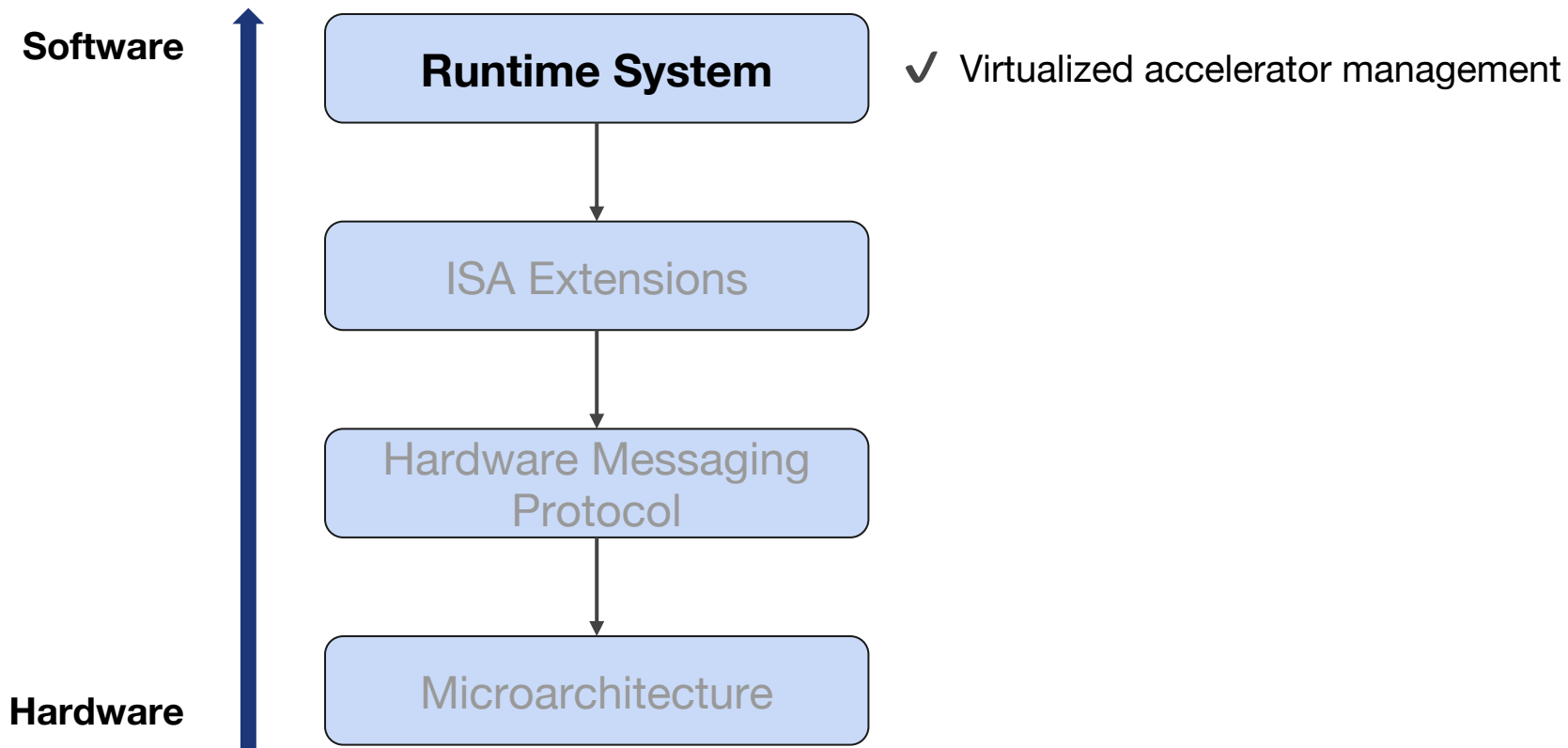
- Allows user thread to interact with HW in programmable fashion
- Low-overhead: bounded by interconnect latency

AuRORA ISA Extensions

- `rerocc_acquire`
 - Maps physical accelerator to virtual accelerator index
 - Return success status
- `rerocc_assign`
 - Maps virtual accelerator to available opcode on its architectural thread
 - Allows an architectural thread to acquire more accelerators than the available opcode space




AuRORA Full Stack Implementation



AuRORA Runtime


- Backwards compatibility with accelerator SW
 - Invoked only before entry of DNN layer execution
- Low overhead
 - Implements in user-space
 - No need to preempt during layer execution
- Dynamically allocate resources for multi-tenant workload
 - Latency target-aware resource allocation

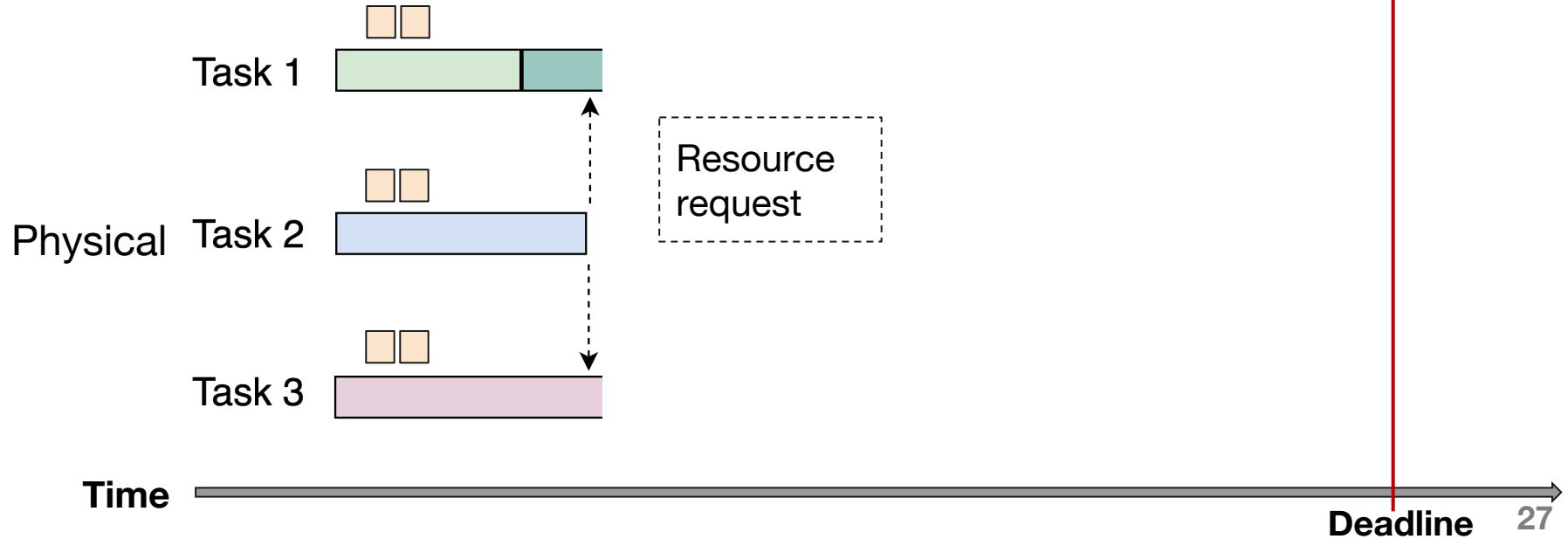
Flow Without AuRORA

 : Scheduling granularity (layer)

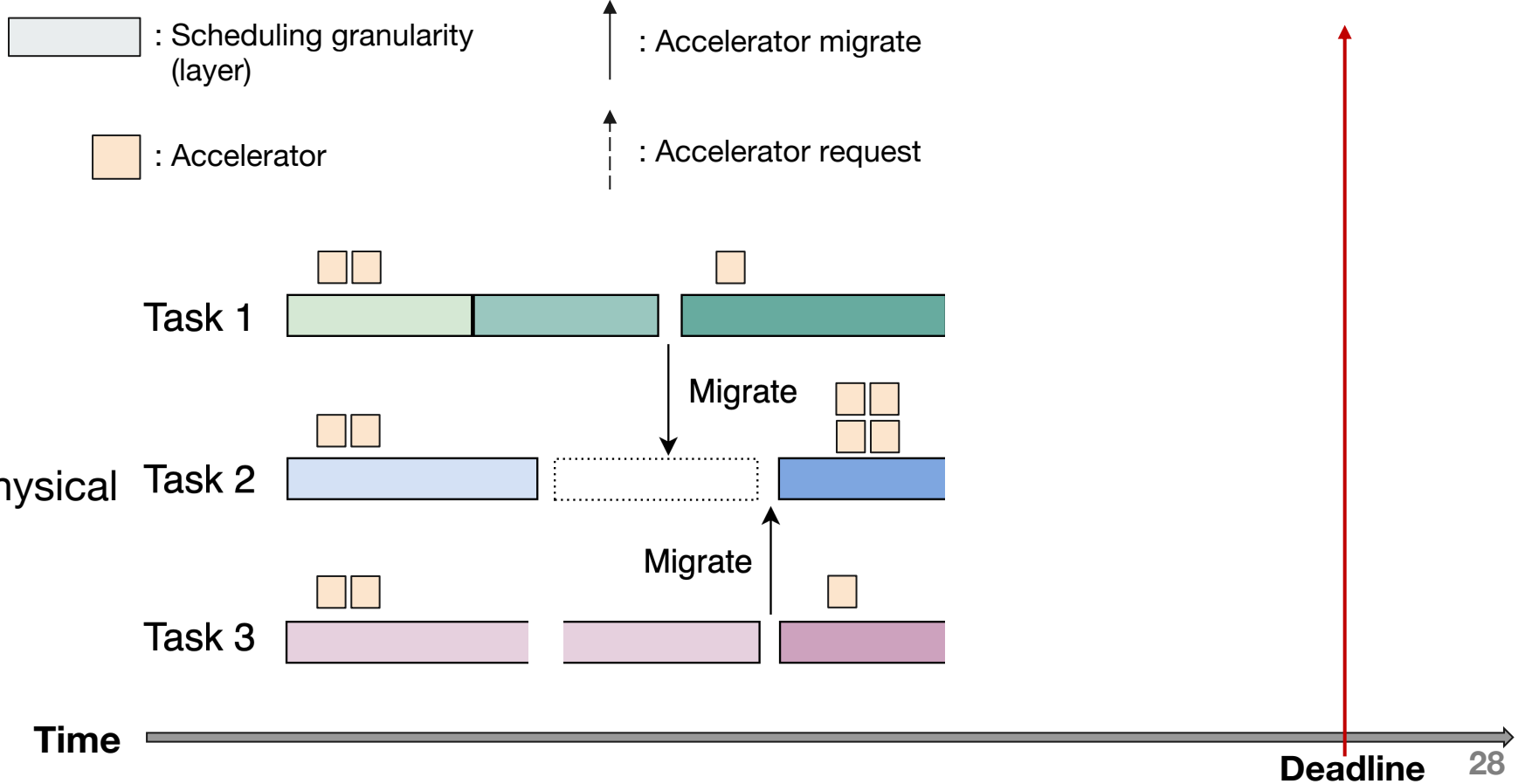
 : Accelerator

 : Accelerator migrate

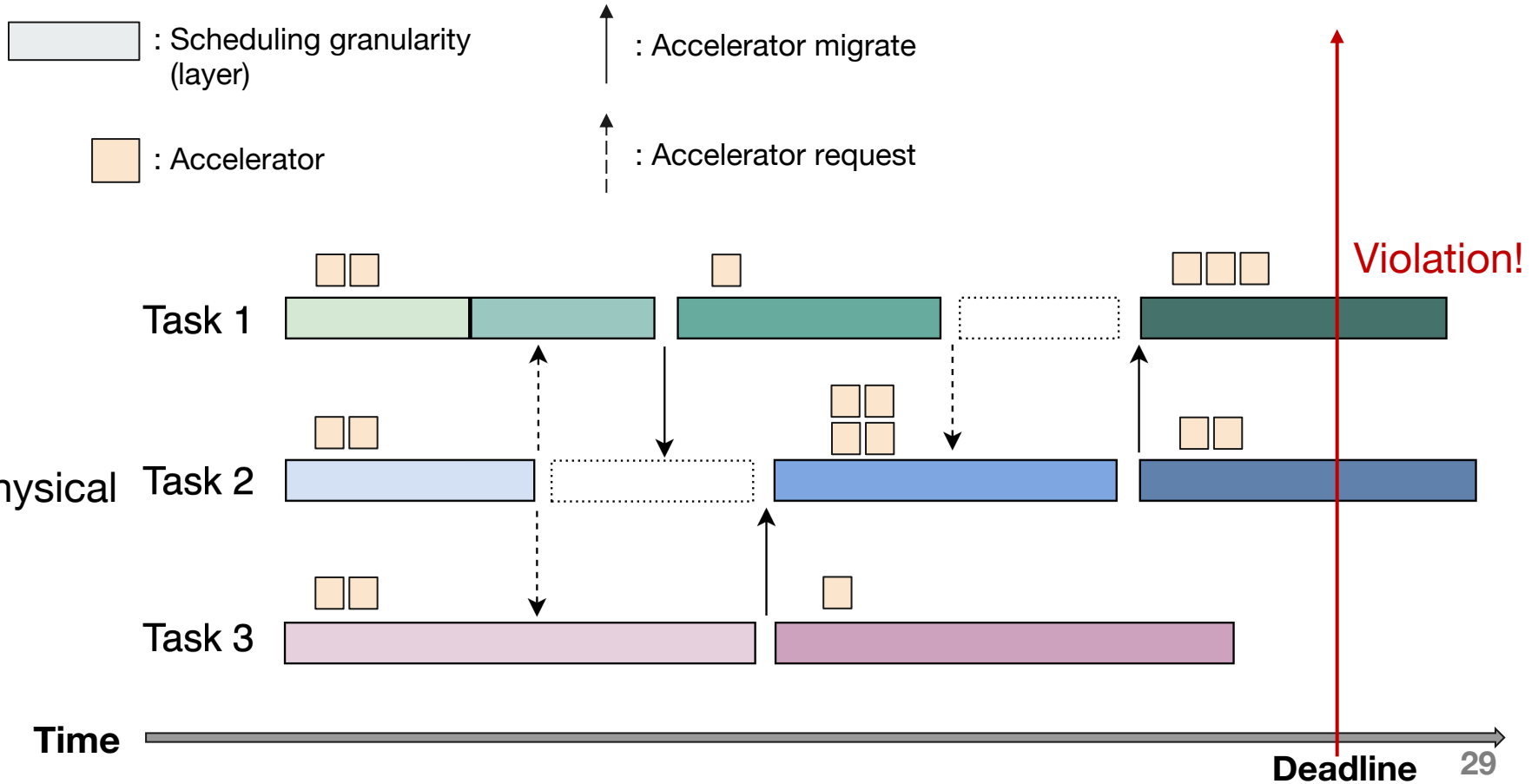
 : Accelerator request




Flow Without AuRORA




Flow Without AuRORA

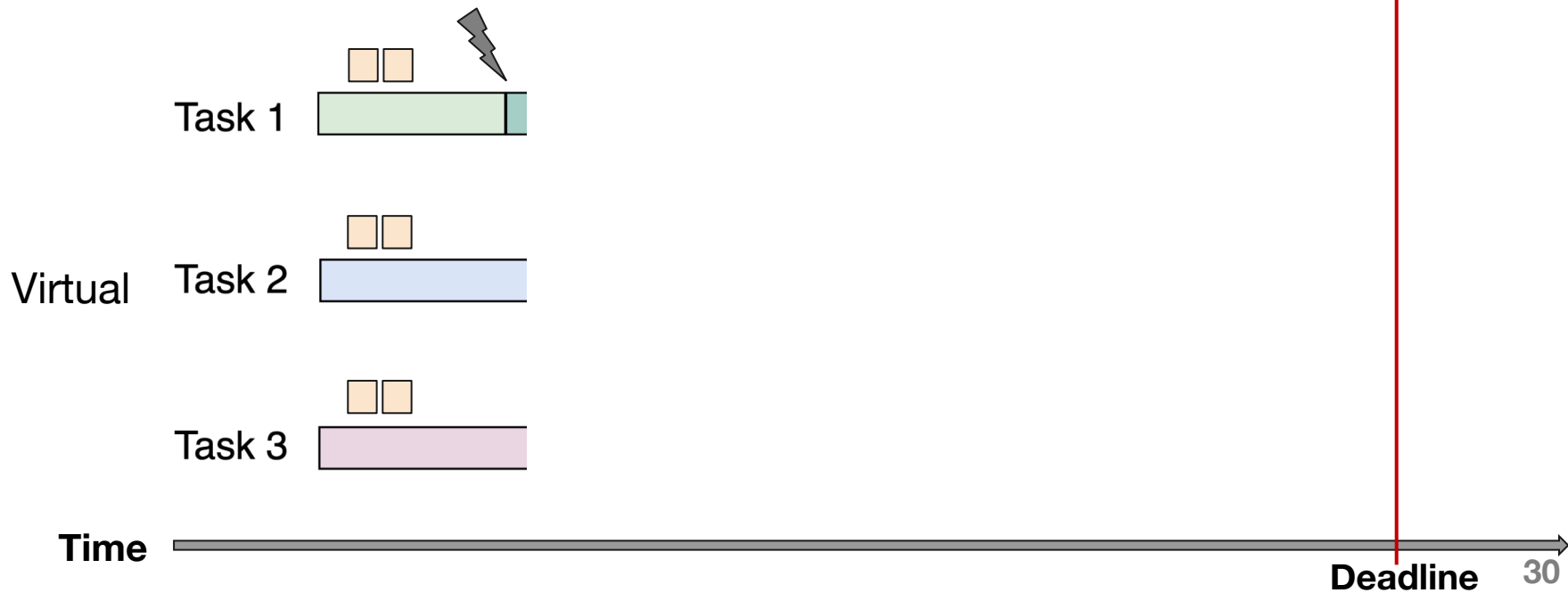


AuRORA Runtime


 : Scheduling granularity (layer)


 : Accelerator release

 : Accelerator

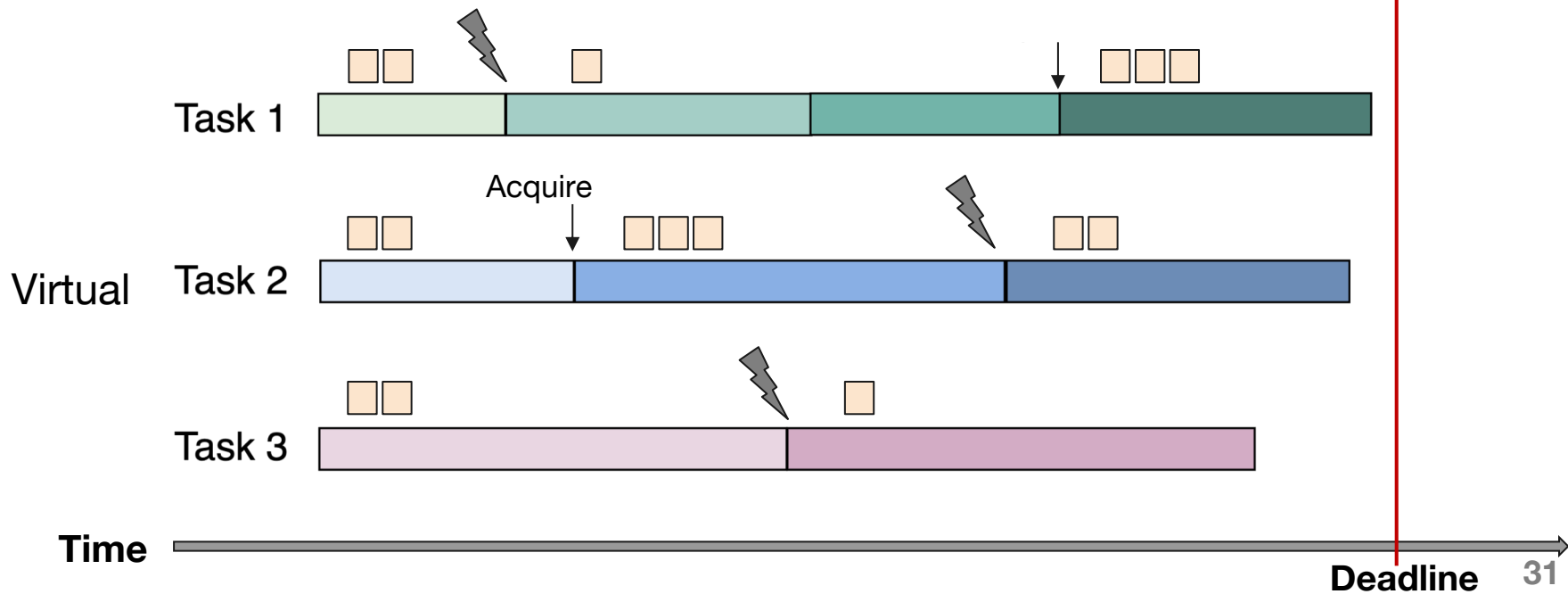


AuRORA Runtime

 : Scheduling granularity (layer)

 : Accelerator release

 : Accelerator



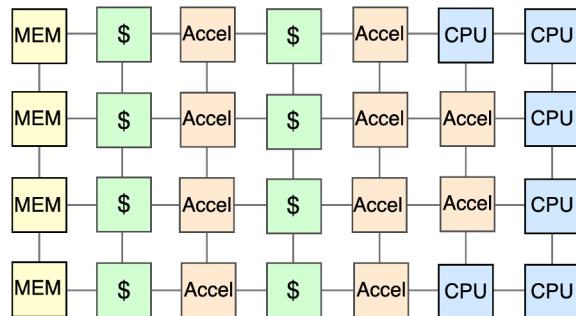
AuRORA Evaluation Methodology

- Implementation details
 - Hardware Manager/Client: Chisel RTL
 - Integrated into Chipyard
 - Software runtime: C++, Linux pthread
 - Runs on top of full Linux stack
- Full-system evaluation details
 - DNN accelerator generator: Gemmini
 - NoC generator: Constellation
 - FPGA evaluation: FireSim



AuRORA Evaluation Methodology

- Implementation details
 - Hardware Manager/Client: Chisel RTL
 - Integrated into Chipyard
 - Software runtime: C++, Linux pthread
 - Runs on top of full Linux stack
- Full-system evaluation details
 - DNN accelerator generator: Gemmini
 - NoC generator: Constellation
 - FPGA evaluation: FireSim
 - 2 integration configuration:
 - Crossbar**
 - NoC



NoC Configuration for Evaluation

Parameter	Value
Systolic array dimension (per tile)	16x16
Scratchpad size (per tile)	128KiB
Accumulator size (per tile)	128KiB
# of accelerator tiles	10
Shared L2 size	2MB
Shared L2 banks	8
DRAM bandwidth	32GB/s
Frequency	1GHz

Chipyard SoC configuration


AuRORA Evaluation Methodology

- Multi-tenant DNN accelerator baselines using physical accelerator
 - **Veltair**: form layer-block to avoid frequent scheduling conflict
 - **MoCA**: groups of layers, dynamic repartition of memory resource

- 2 different configurations of AuRORA
 - **AuRORA-Compute**: dynamic compute resource repartition, without NUMA
 - **AuRORA-All**: enable all optimization (memory partitioning, NUMA-aware accelerator allocation)

AuRORA Evaluation Methodology

- Benchmark DNNs: 10 different DNN inference models
 - Grouped by model size
 - Construct multi-tenant scenario: randomly generated 200-300 inference tasks
- QoS targets
 - 3 different latency targets
 - QoS-H: 1.2x
 - QoS-M: 1x
 - QoS-L: 0.8x



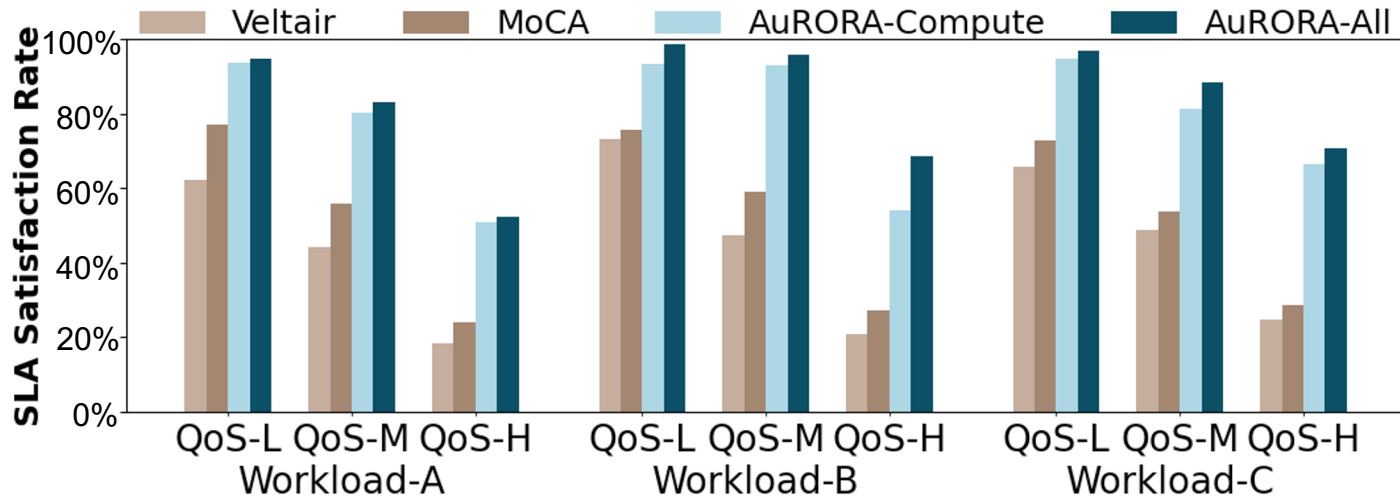
Workload	Model Size	DNN Models
Workload set-A	Light	SqueezeNet, Yolo-LITE, KWS, ResNet18, BERT-small
Workload set-B	Heavy	GoogLeNet, AlexNet, ResNet50, YoloV2, BERT-base
Workload set-C	Mixed	All
Workload set-XR	Mixed	XRBenCh Gaming Scenarios

AuRORA Evaluation Methodology

- Evaluation Metrics:
 - **SLA Satisfaction Rate**
 - **Latency (QoS) target satisfaction ratio**
 - System Throughput (STP)
 - Sum of each program's normalized progress
 - Fairness
 - Evaluates degree to which all programs make equal progress
 - XRBench metrics
 - For Workload set-XR evaluation

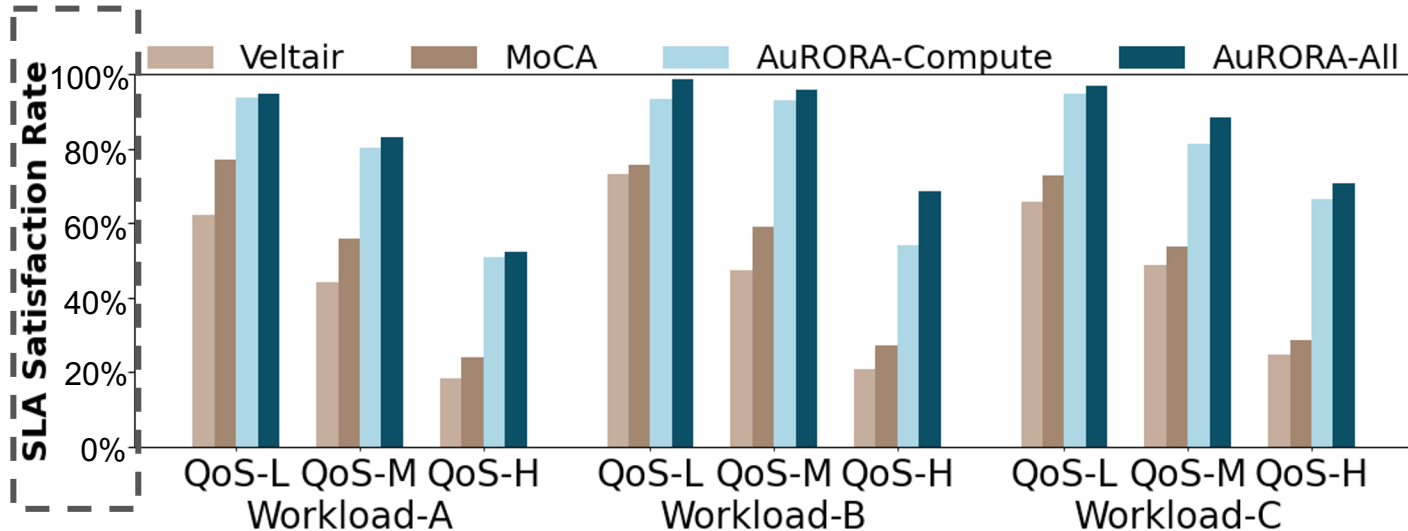
SLA Satisfaction Rate Improvement

- SLA (Service Level Agreement) satisfaction
 - Whether the request meets QoS target



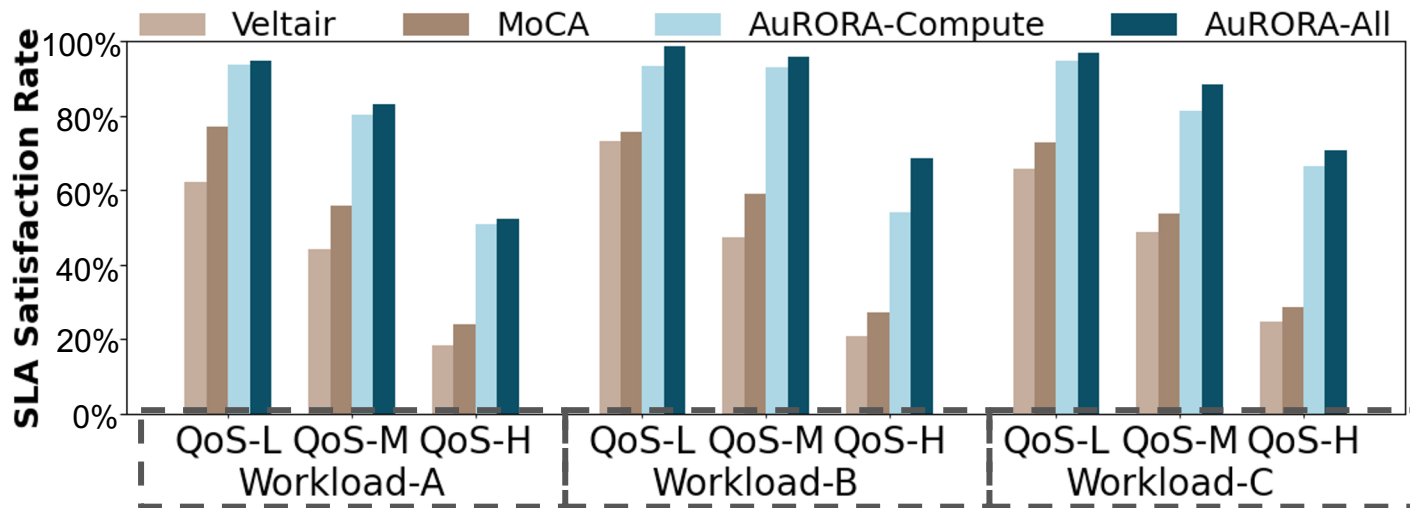
SLA Satisfaction Rate Improvement

- SLA satisfaction rate
 - Range 0% (all fail) ~ 100% (all met QoS)



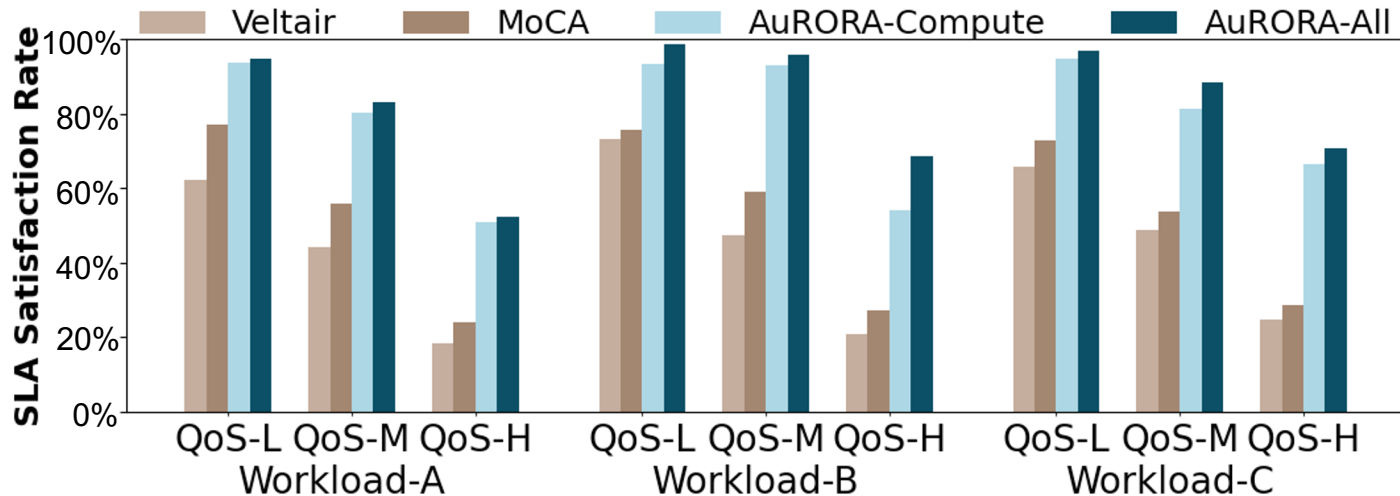
SLA Satisfaction Rate Improvement

- 2-level x-axis
 - Each **Workload** set subdivided into **QoS target level**



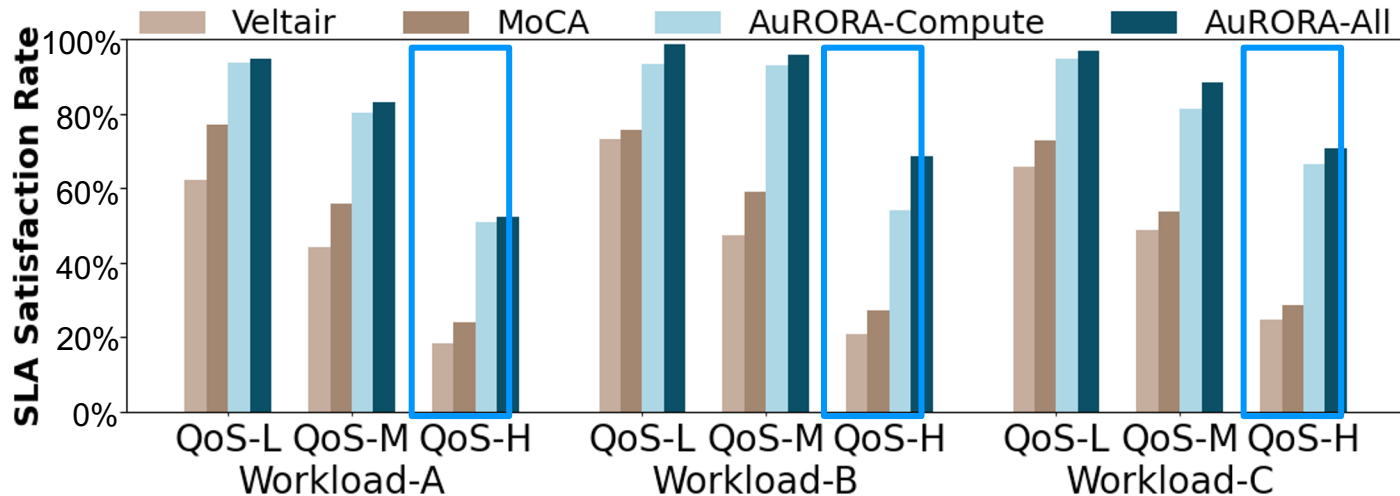
SLA Satisfaction Rate Improvement

- SLA satisfaction rate: Crossbar
 - AuRORA-Compute: 1.9x over Veltair (2.76x max), 1.6x over MoCA (2.33x max)
 - Effectiveness of virtual accelerator management: fast reallocation of compute resources
 - AuRORA-All: improves 1.07x over AuRORA-Compute (1.12x on Workload-B)
 - Effectiveness of dynamic memory resource management



SLA Satisfaction Rate Improvement

- SLA satisfaction rate: Crossbar
 - AuRORA-Compute: QoS-H shows highest improvement over baselines
 - 2.68x over Veltair, 2.14x over MoCA
 - Physical resource partitioning overhead pronounced
 - Baseline's coarser-grained strategy is challenging when fast reconfiguration is needed



Physical Design & Area Analysis

- Synthesize using Intel 16nm
 - Synthesis: Cadence Genus

Component	Area (μm^2)	% of Area
CPU tile	168K	100%
AuRORA Client	2K	1.2%
Rocket CPU	166K	98.8%
Accelerator tile	732K	100%
AuRORA Manager	22K	3%
Accelerator	710K	97%
Mesh	76K	10.4%
Accumulator	260K	35.5%
Scratchpad	150K	20.5%
Total (CPU tile+Accelerator tile)	900K	100%
AuRORA Client + Manager	24K	2.7%

**AuRORA only takes
2.7% of total area**

Summary

AuRORA: A full-stack hardware/software integration approach to support virtualized accelerator orchestration

AuRORA enables scalable many-accelerator system for multi-tenant execution

Full-system evaluation using real SoC, real RISC-V cores and accelerators

Performance/area evaluation using physically realizable RTL

- Multi-tenant scenario evaluation shows 2.02x target satisfaction rate improvement over baseline
- Synthesis result shows $< 2.7\%$ area overhead

Open-sourced, integrated to Chipyard SoC design framework

Open-sourced: <https://github.com/ucb-bar/AuRORA>

