Microsecond-scale Preemption for Concurrent GPU-accelerated DNN Inferences

Mingcong Han, Hanze Zhang, Rong Chen, Haibo Chen

Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University

Shanghai AI Laboratory

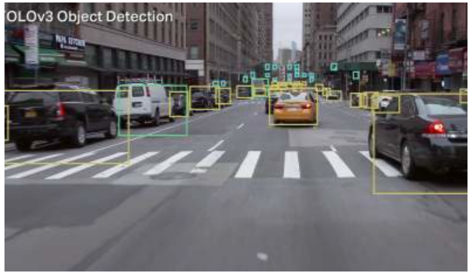
Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China

MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University, China

Motivation

DNNs are widely adopted by modern intelligent applications

Motivation



Obstacle Detection

Fatigue Detection

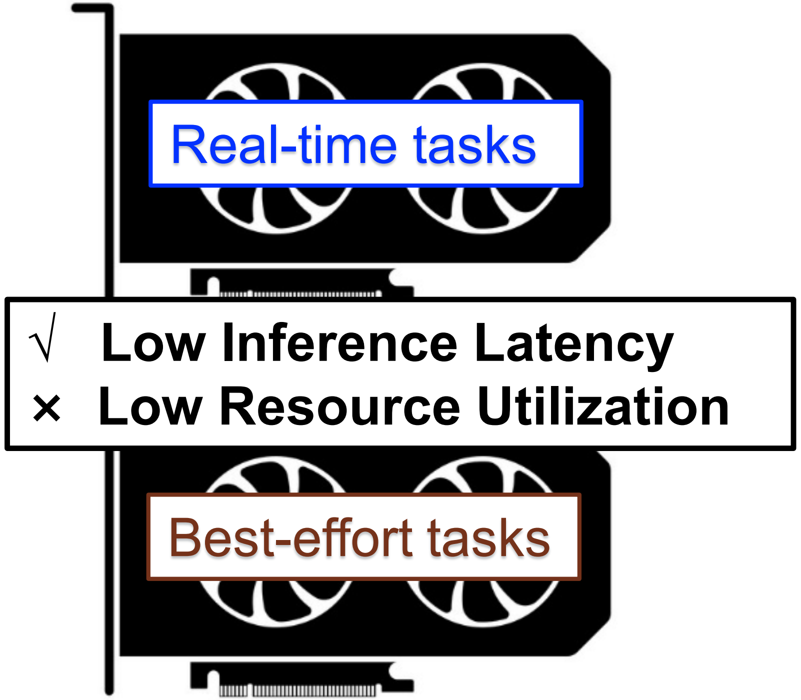
Real-time tasks

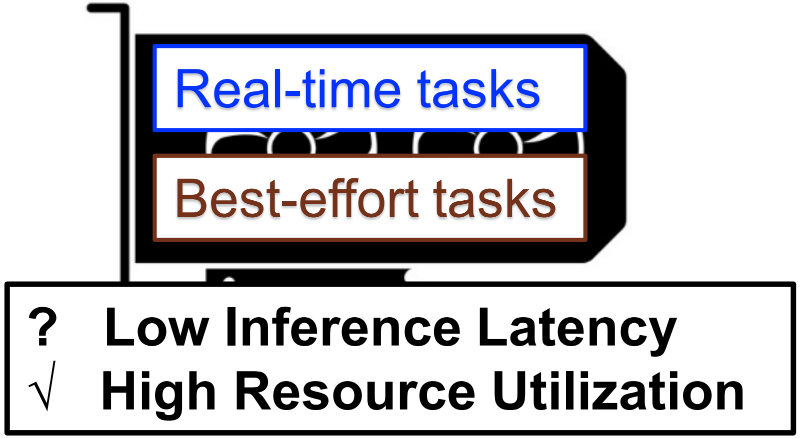
- Latency critical

Best-effort tasks

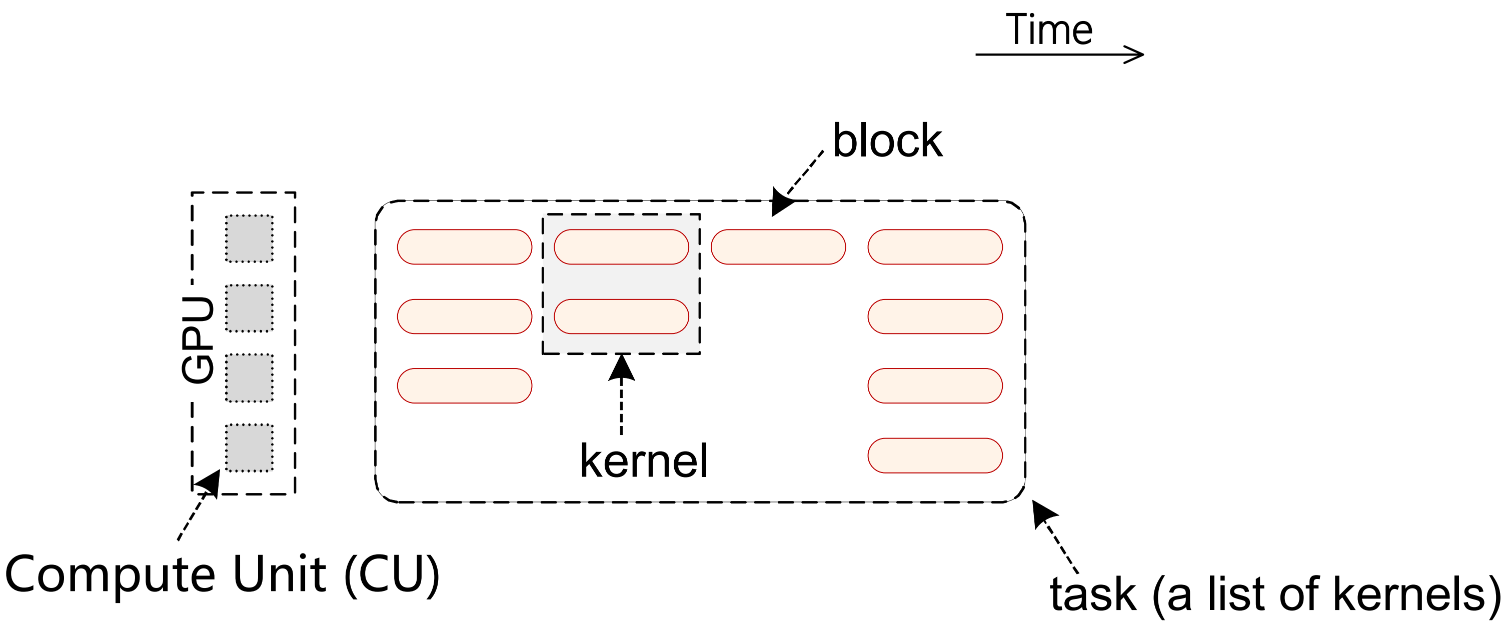
- No hard real-time requirement

Motivation

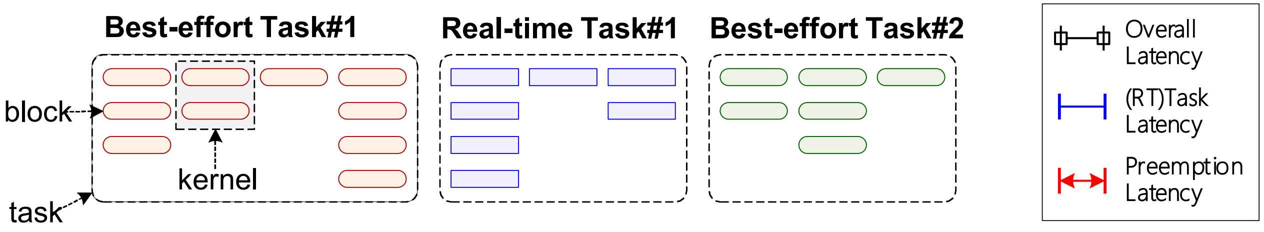

Real-time tasks
√ Low Inference Latency
× Low Resource Utilization
Best-effort tasks


Real-time tasks
Best-effort tasks
? Low Inference Latency
√ High Resource Utilization

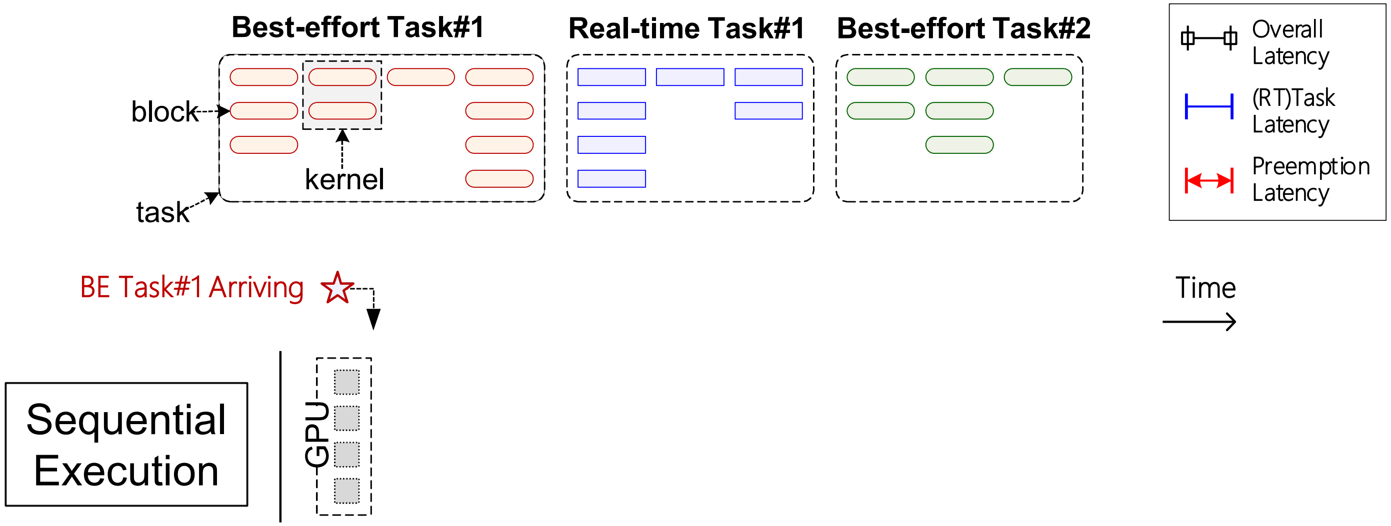
GPU-accelerated DNN inference



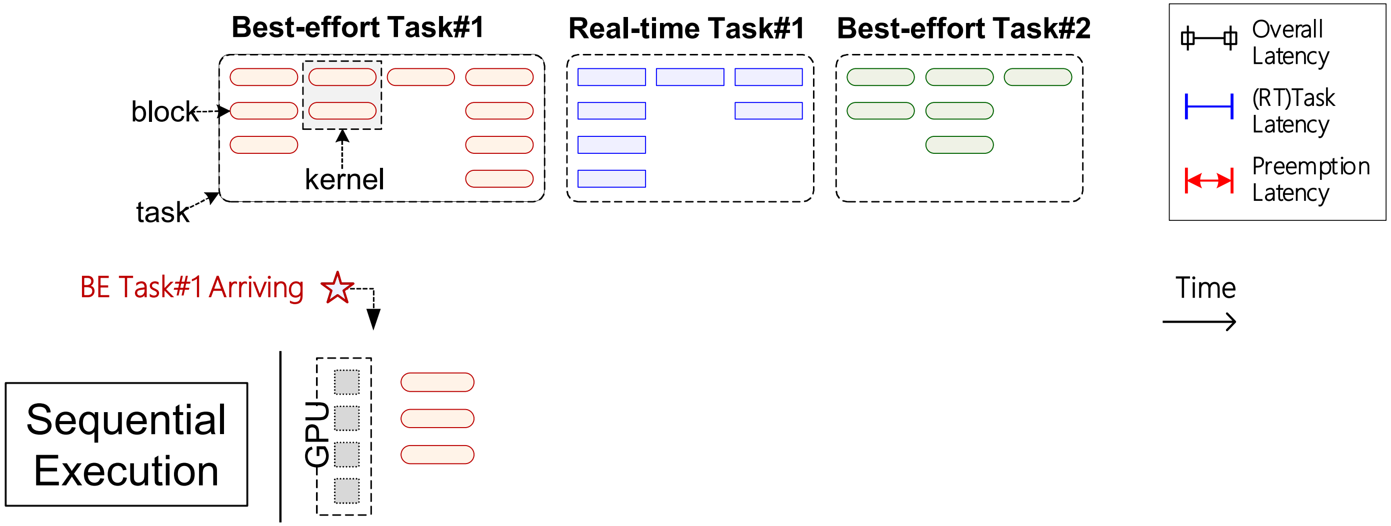
Existing GPU Task Scheduling



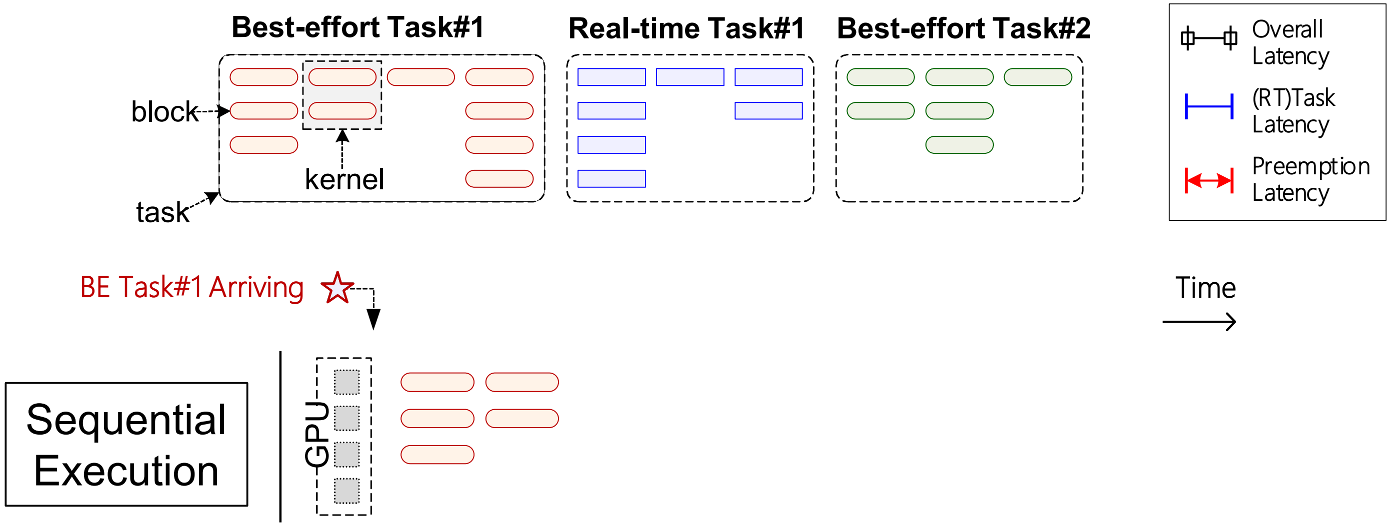
Existing GPU Task Scheduling



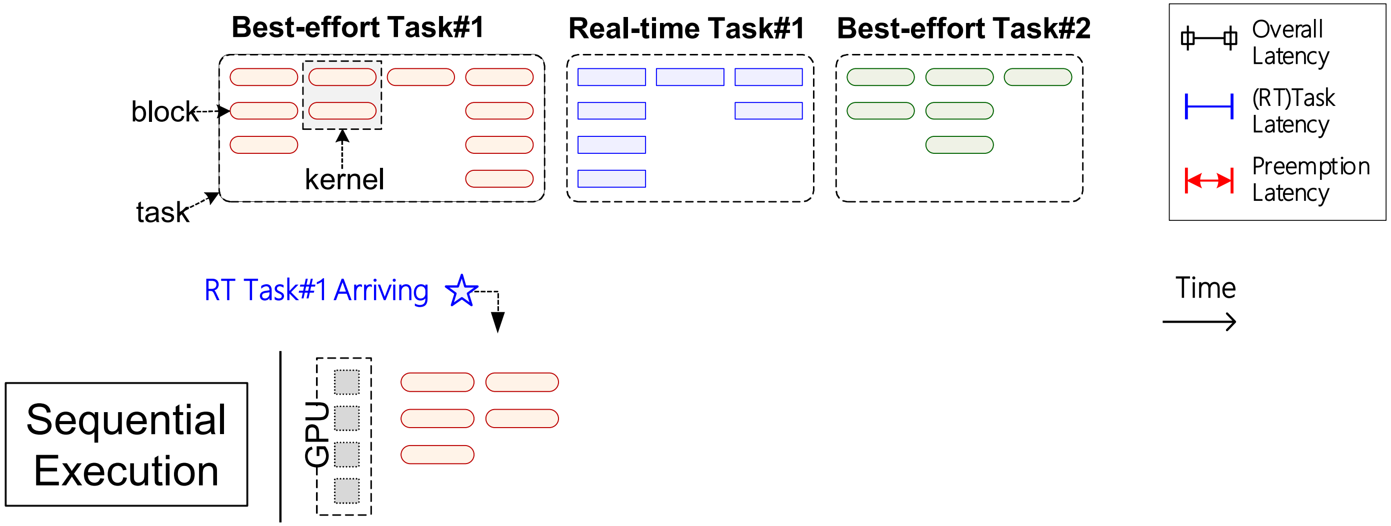
Existing GPU Task Scheduling



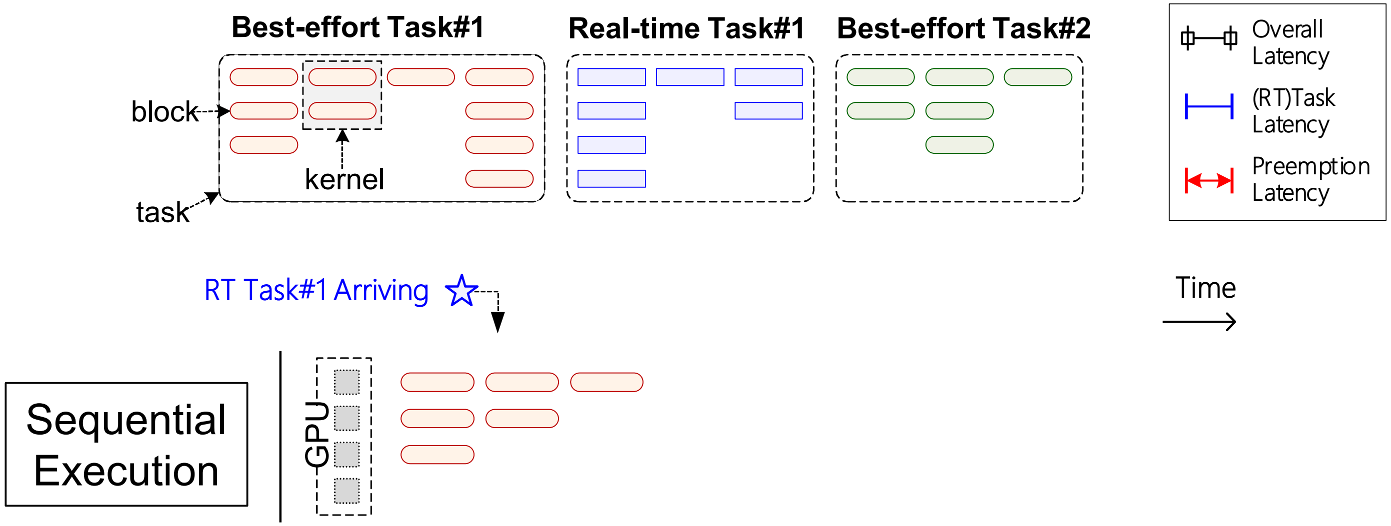
Existing GPU Task Scheduling



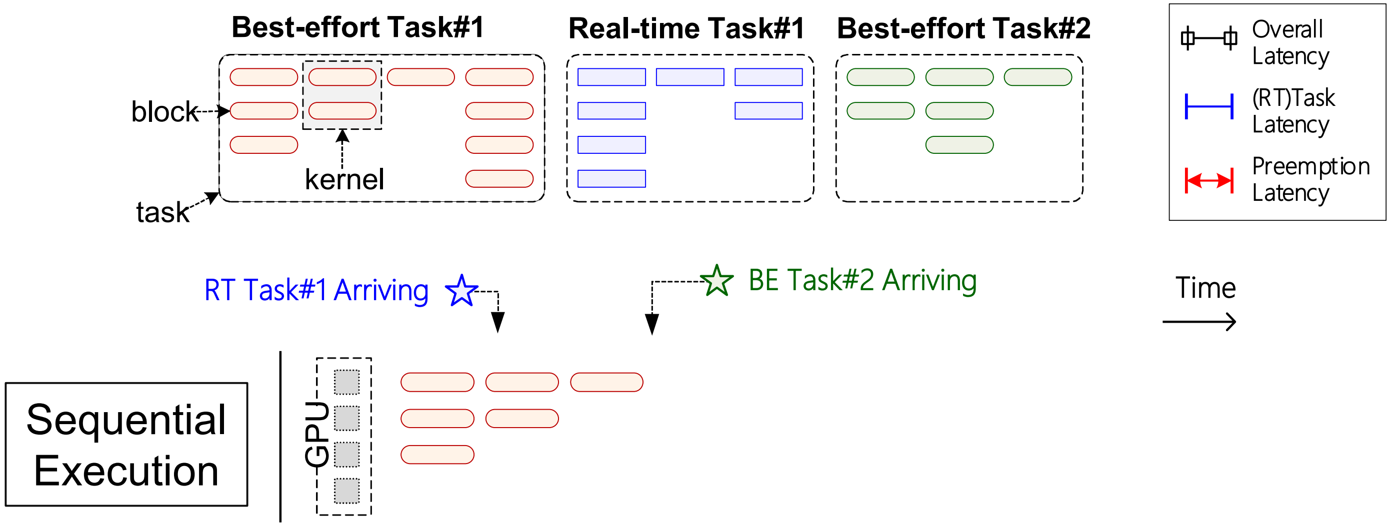
Existing GPU Task Scheduling



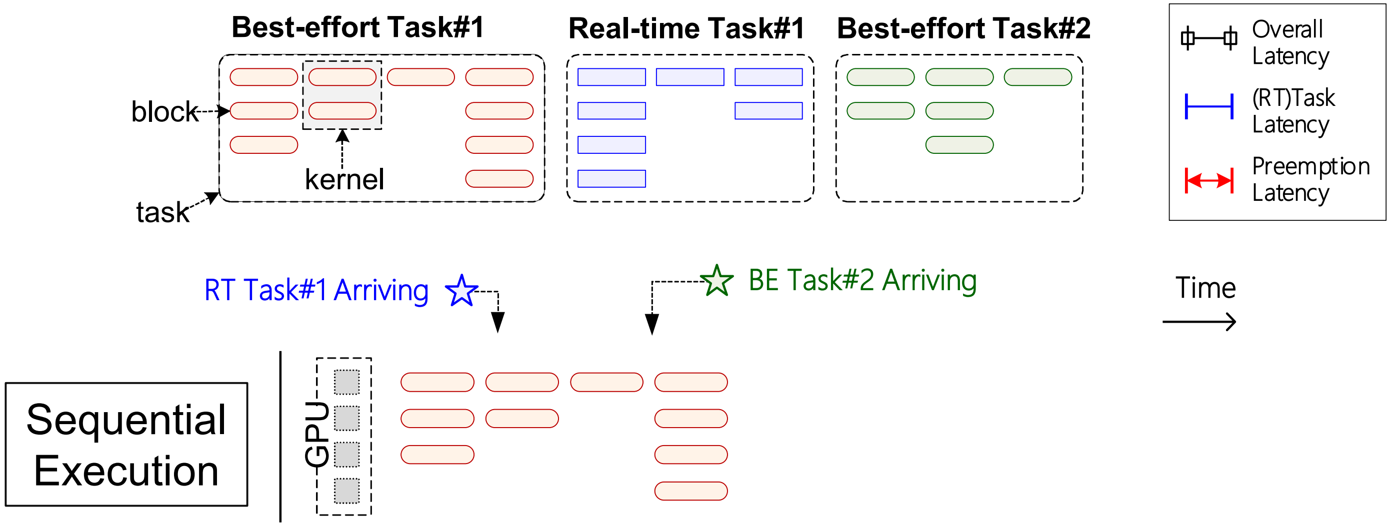
Existing GPU Task Scheduling



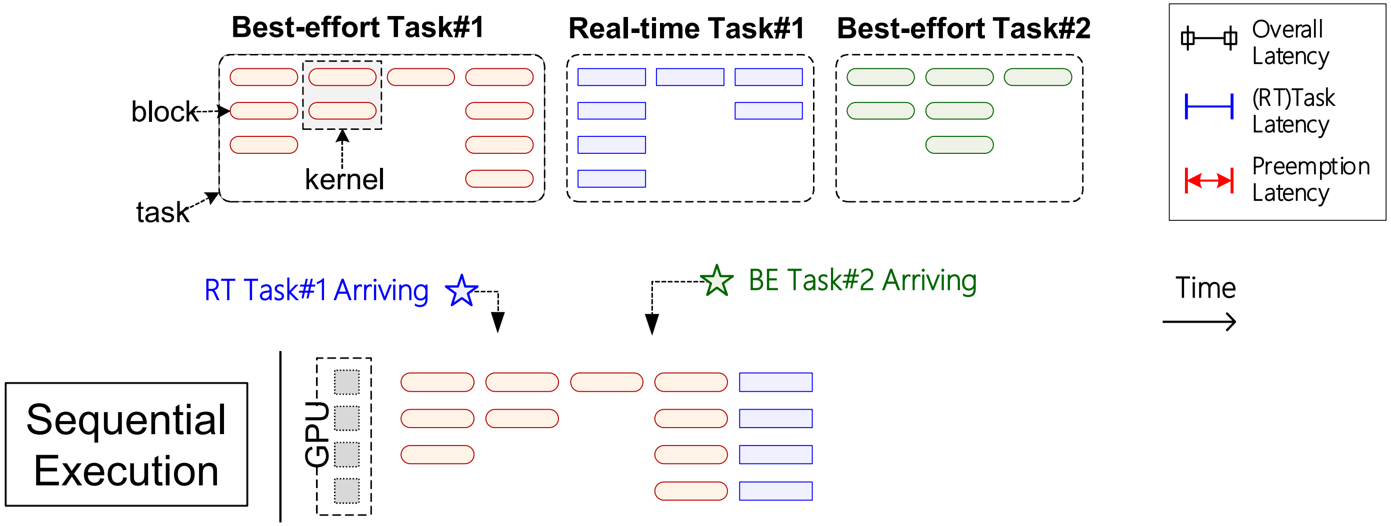
Existing GPU Task Scheduling



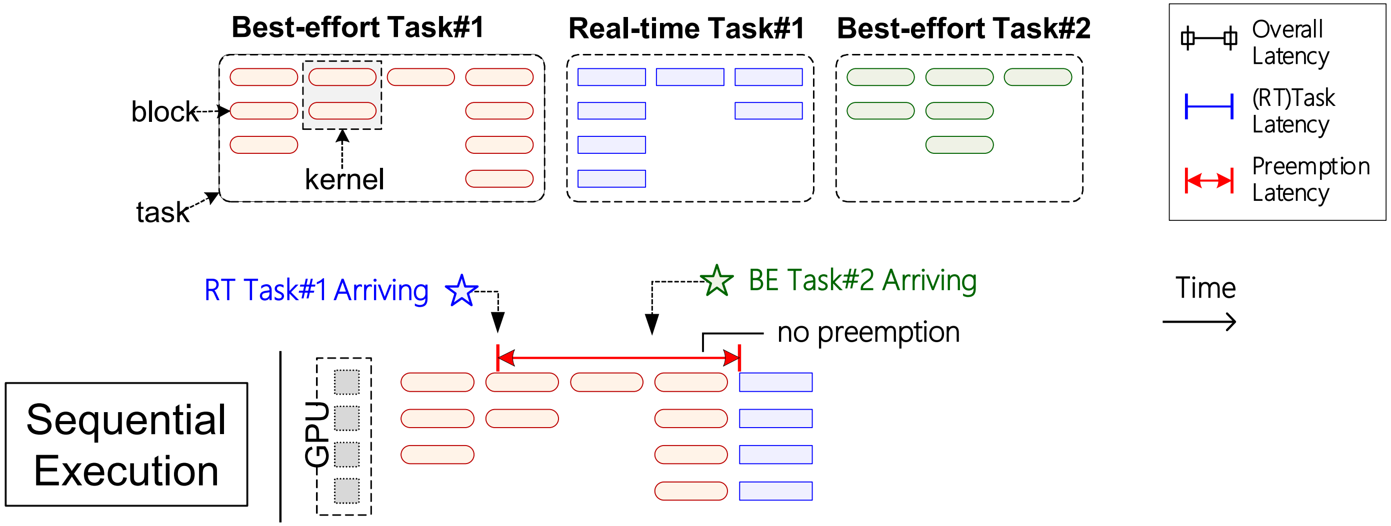
Existing GPU Task Scheduling



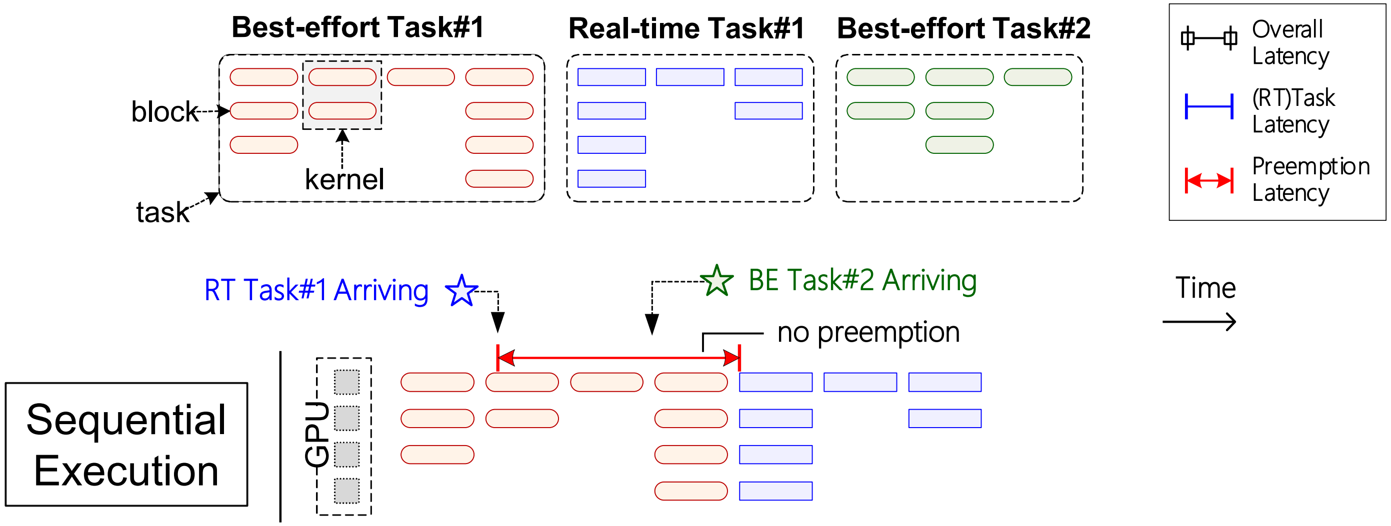
Existing GPU Task Scheduling



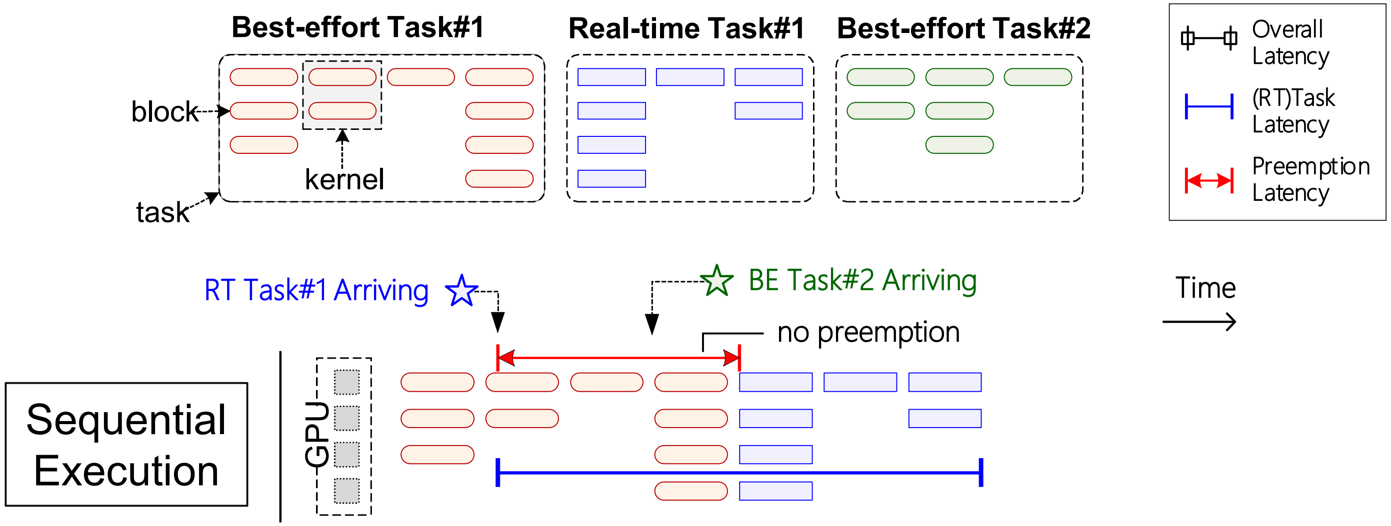
Existing GPU Task Scheduling



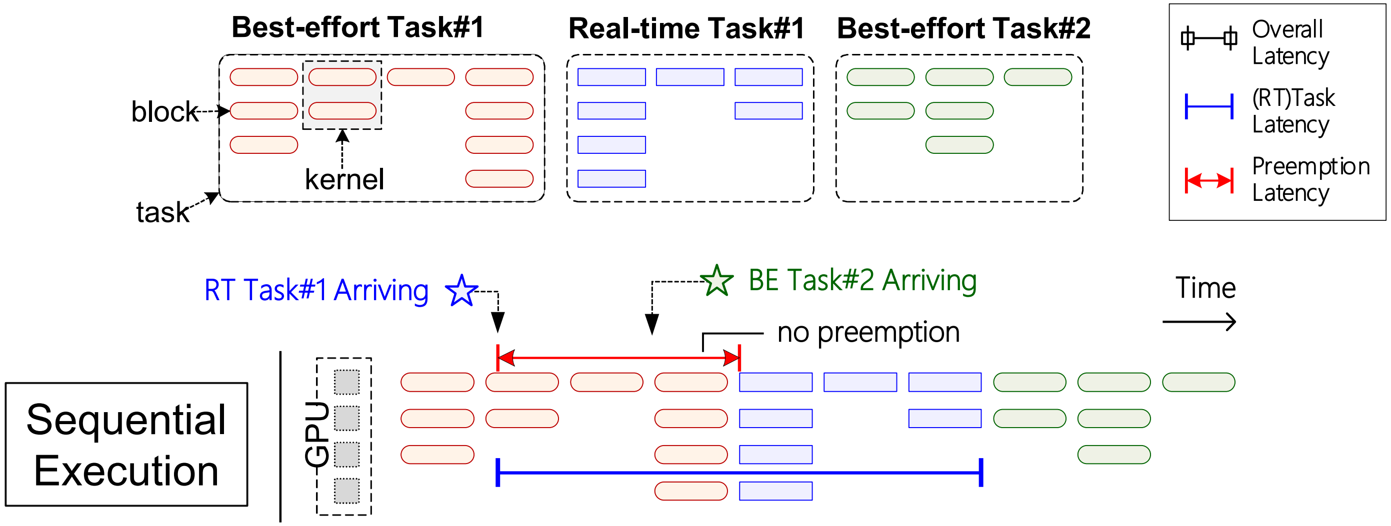
Existing GPU Task Scheduling



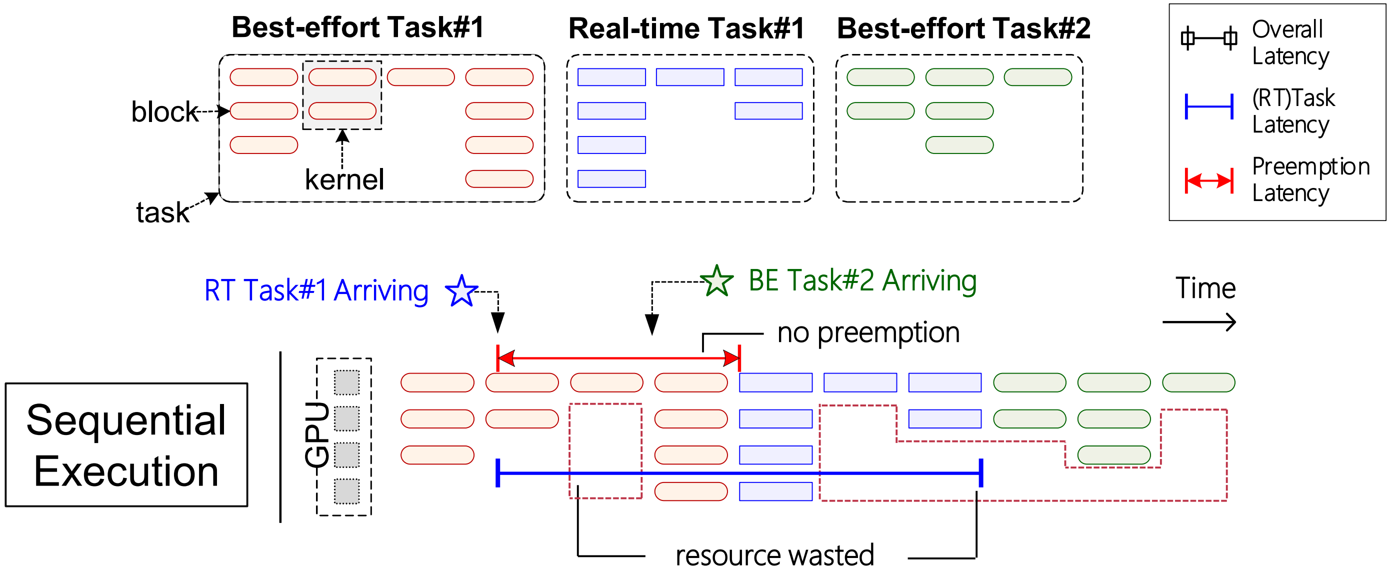
Existing GPU Task Scheduling



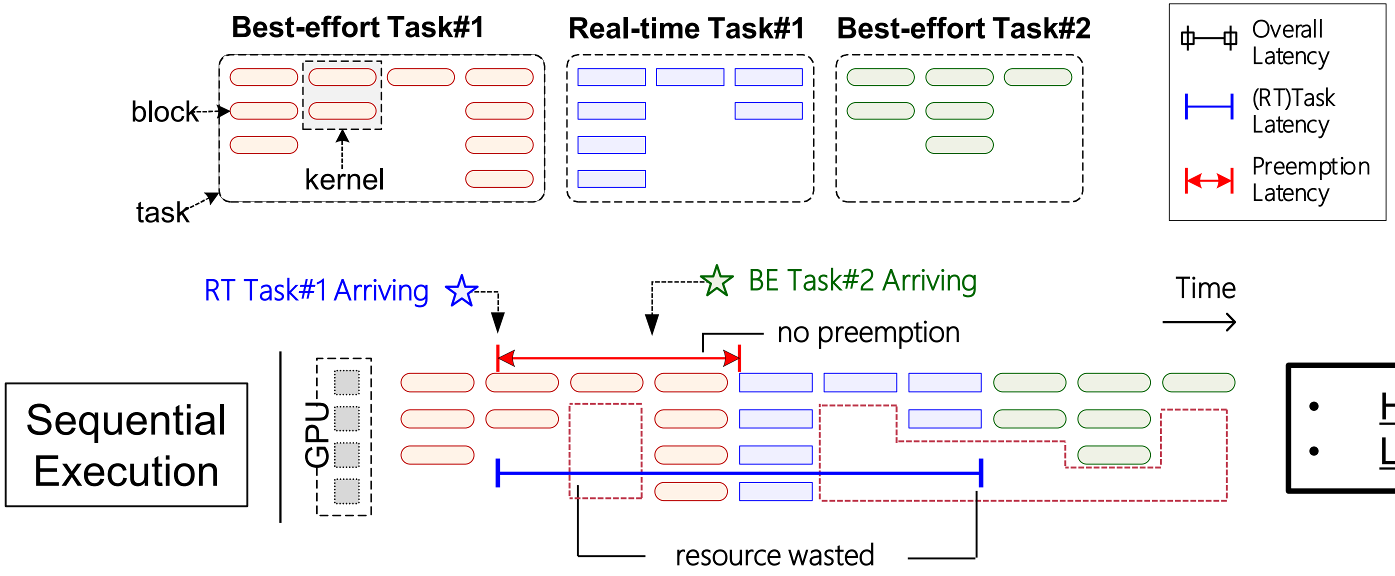
Existing GPU Task Scheduling



Existing GPU Task Scheduling

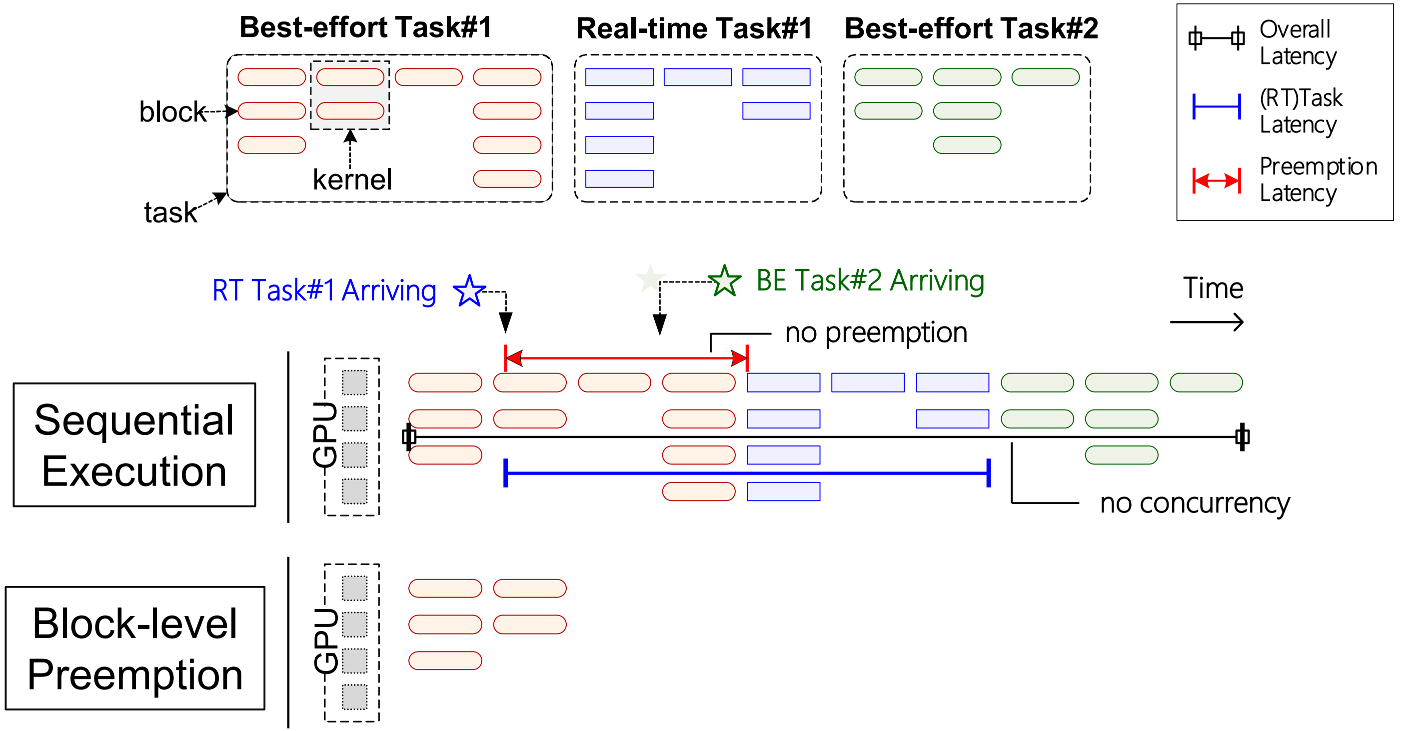


Existing GPU Task Scheduling

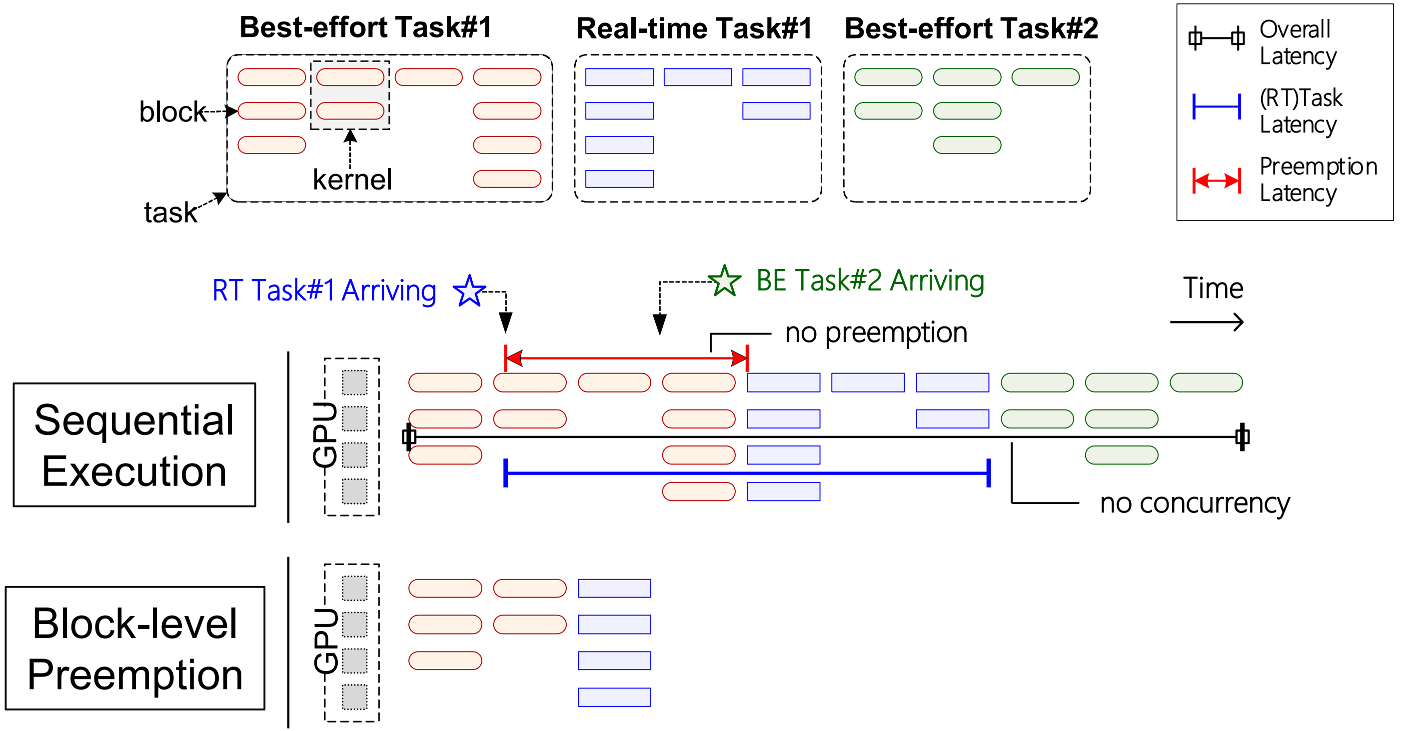


- High latency for **RT** tasks
- Low throughput (not work-conserving)

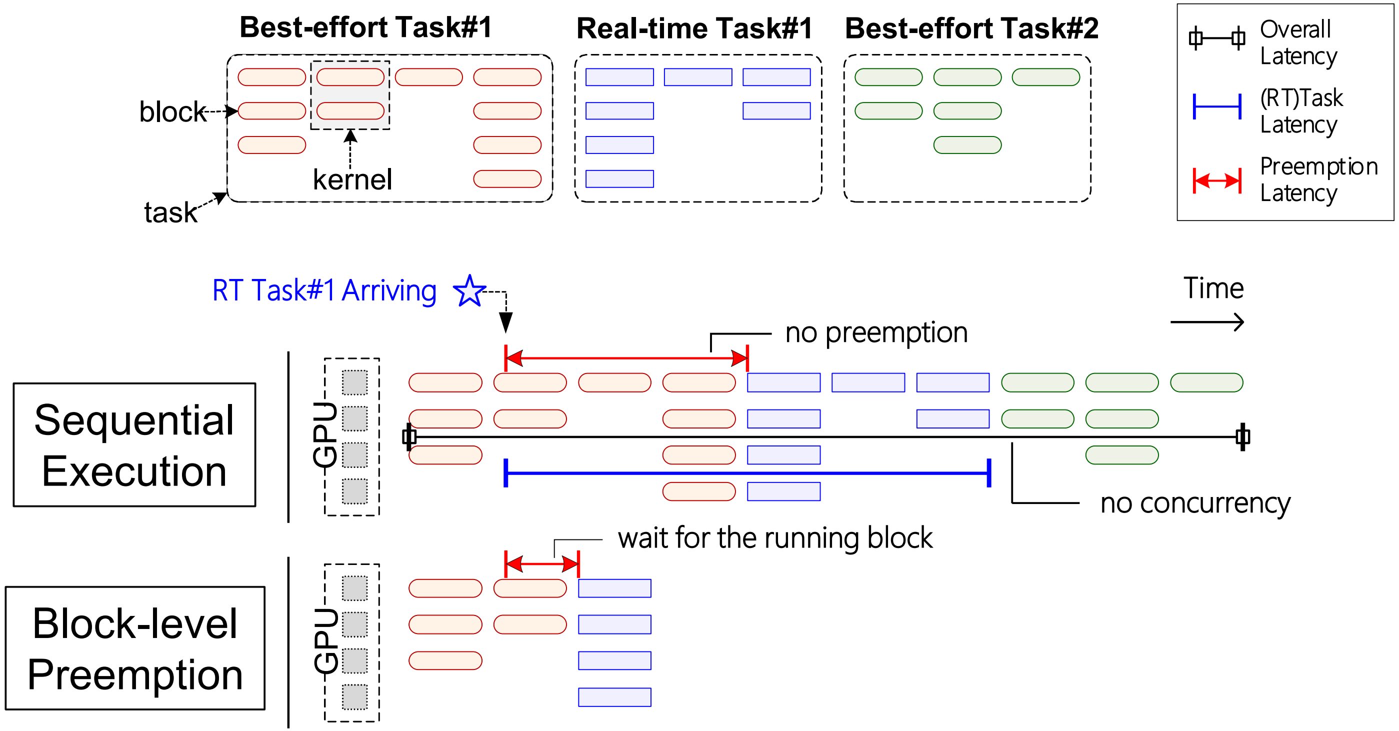
Existing GPU Task Scheduling



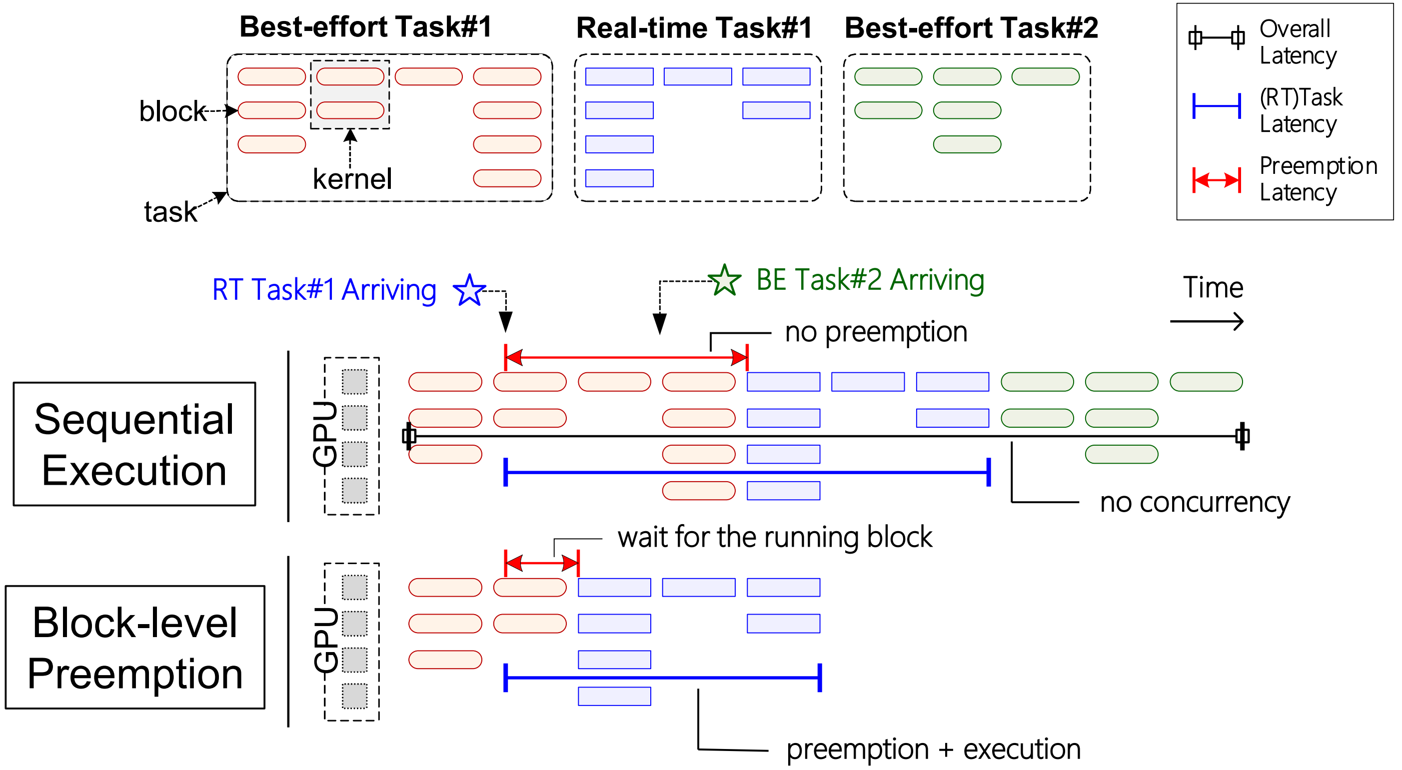
Existing GPU Task Scheduling



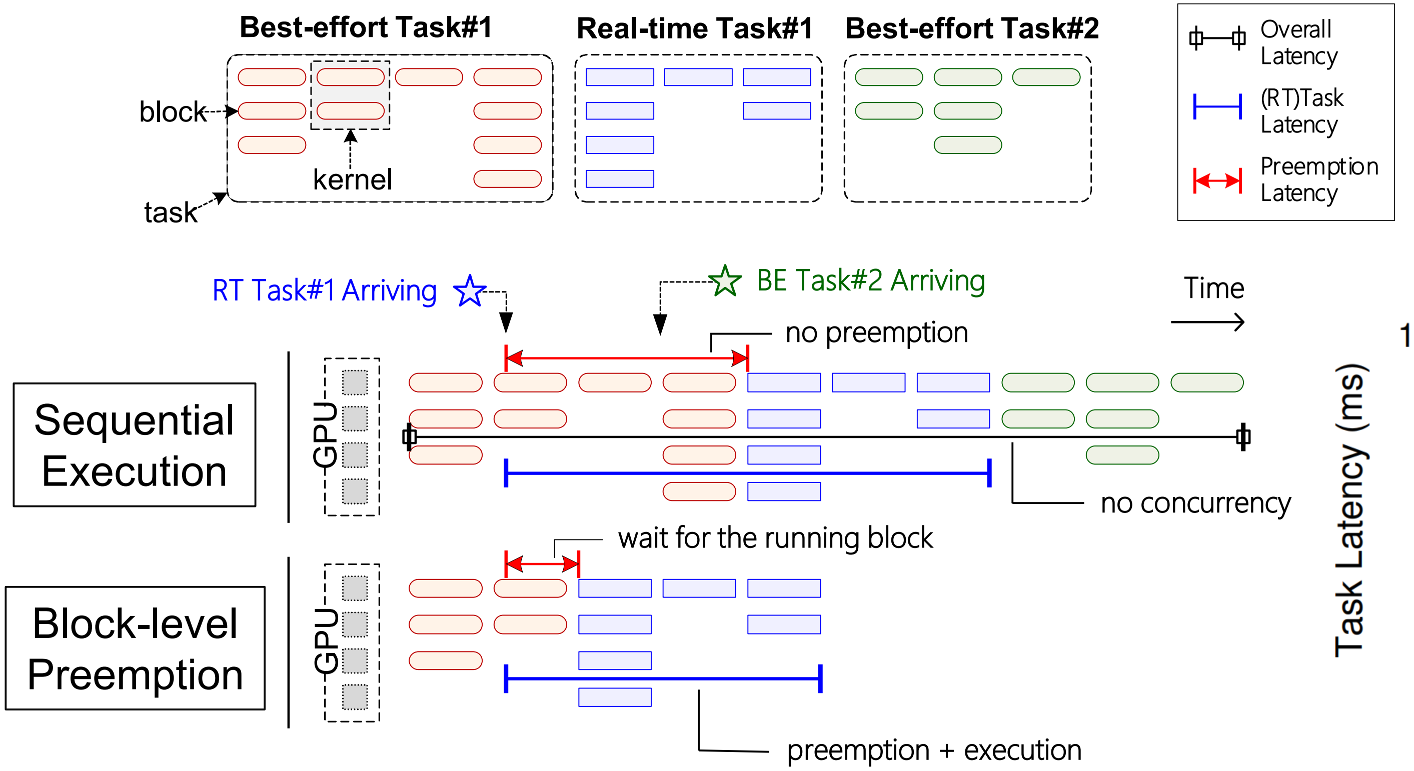
Existing GPU Task Scheduling



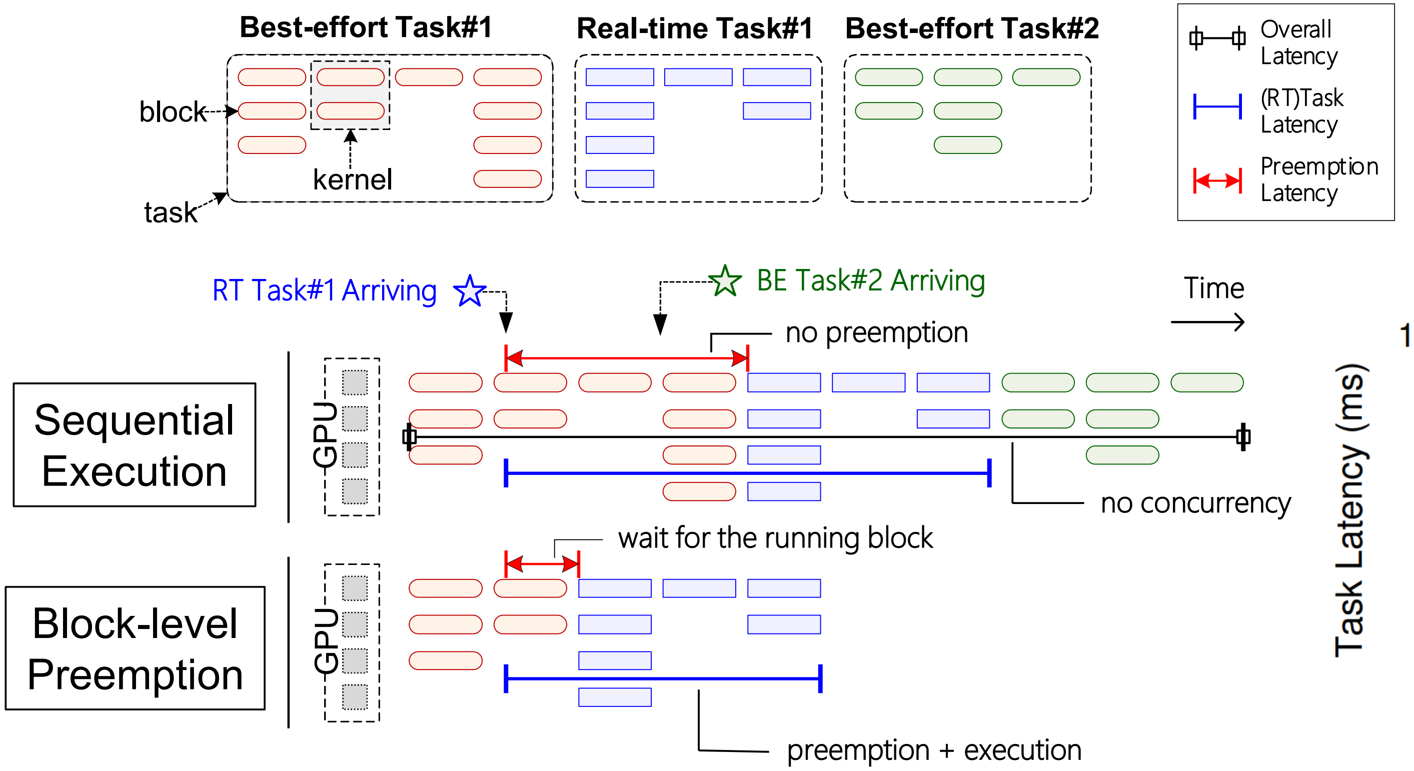
Existing GPU Task Scheduling



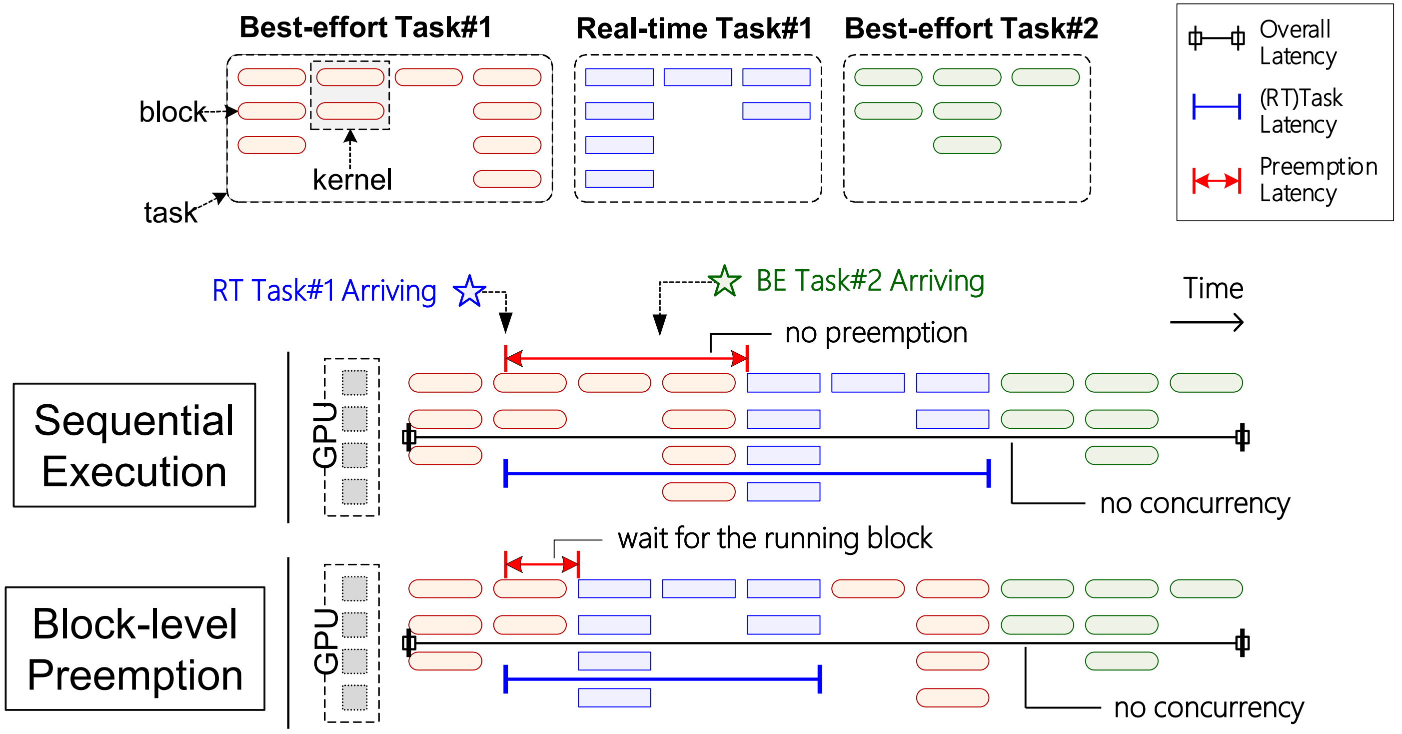
Existing GPU Task Scheduling



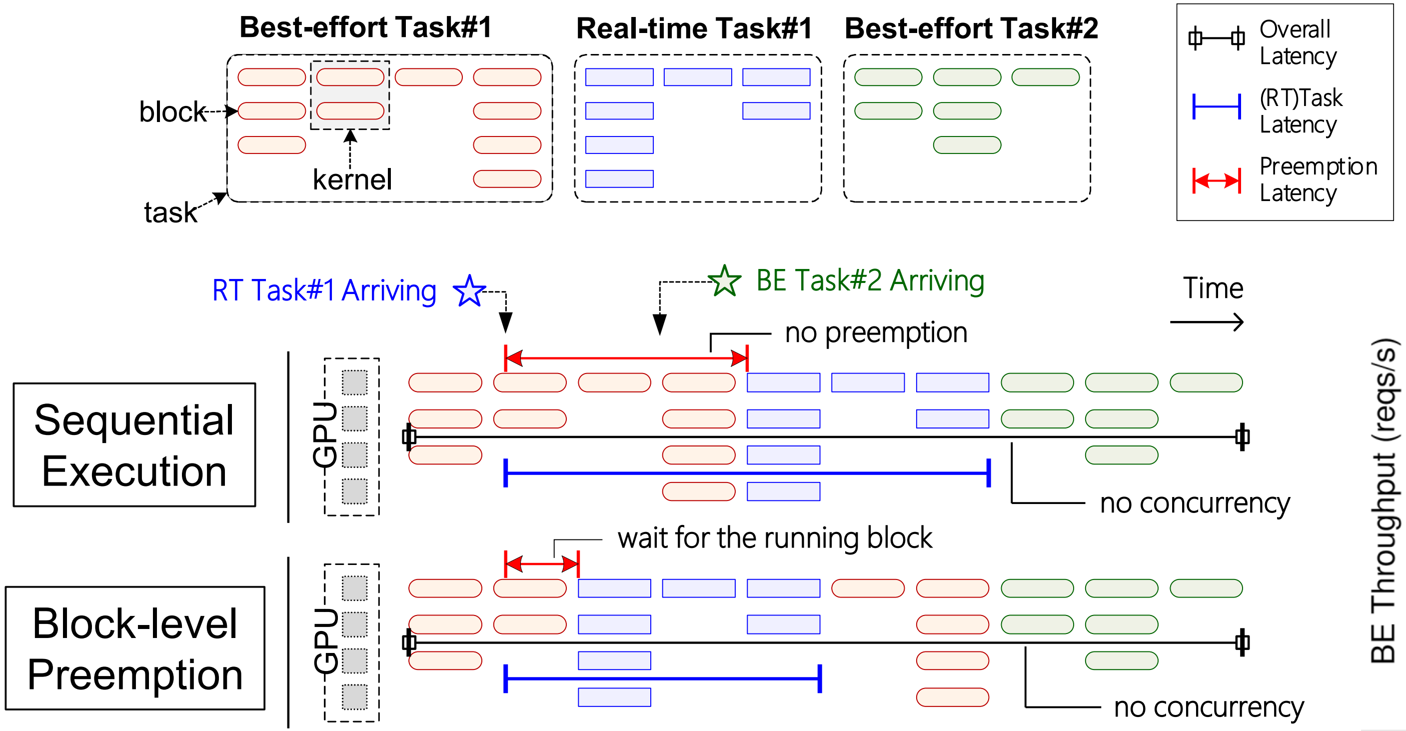
Existing GPU Task Scheduling

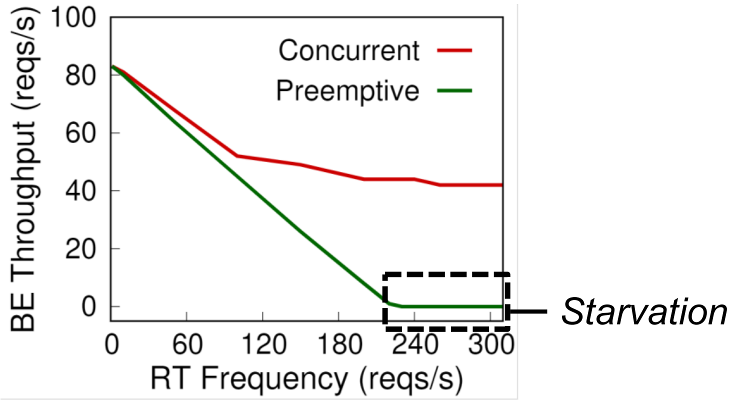


Existing GPU Task Scheduling

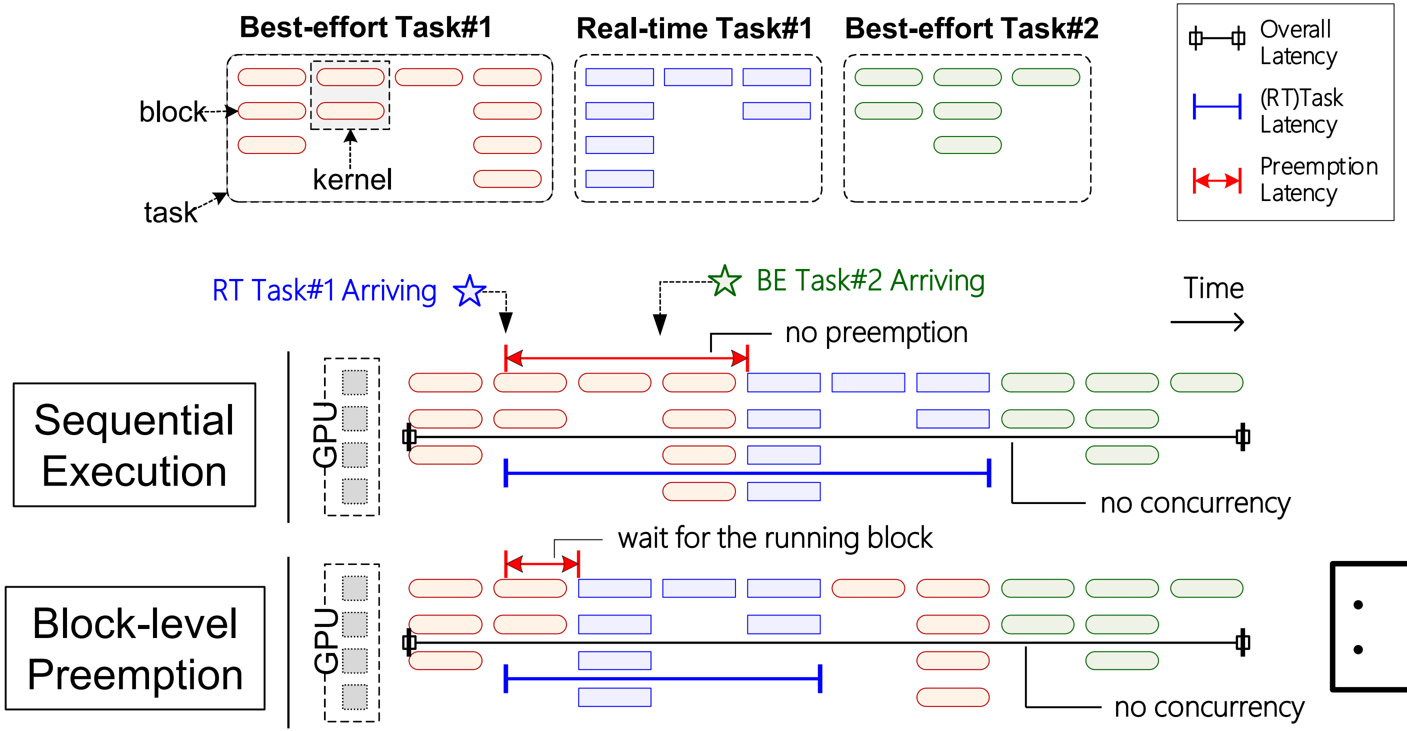


Existing GPU Task Scheduling



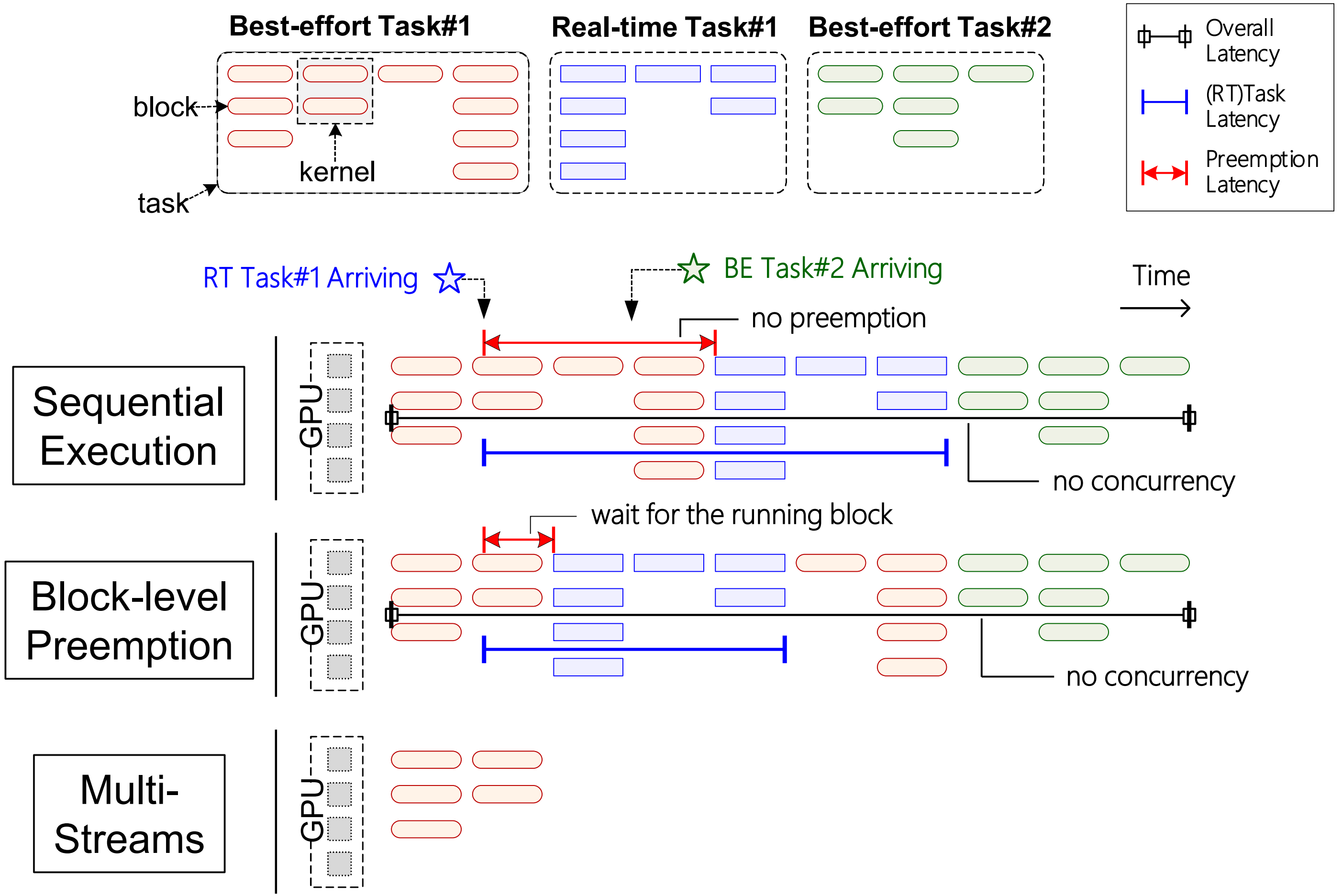


Existing GPU Task Scheduling

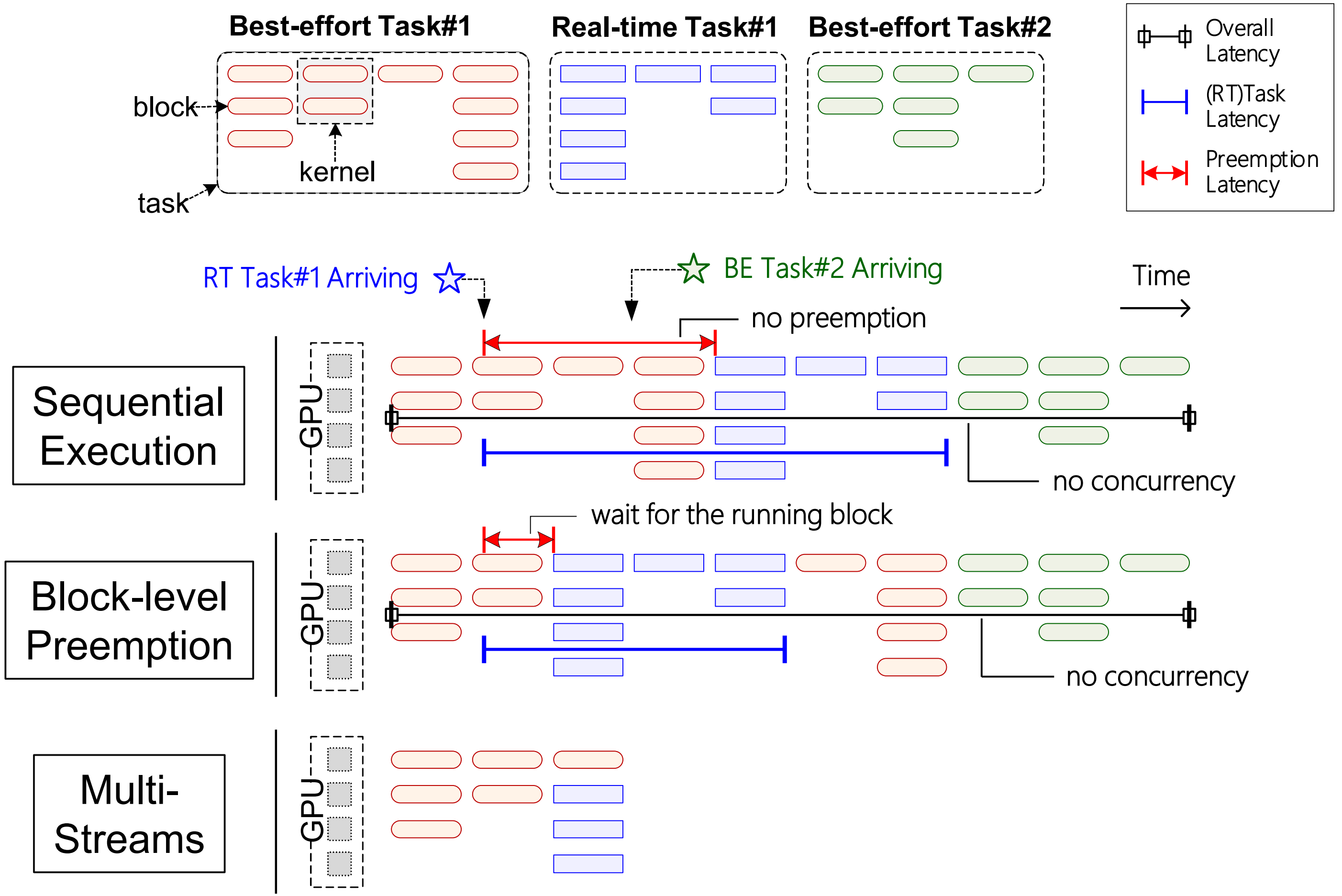


- (Not always) low latency for **RT** tasks
- <u>Low throughput</u> (not work-conserving)

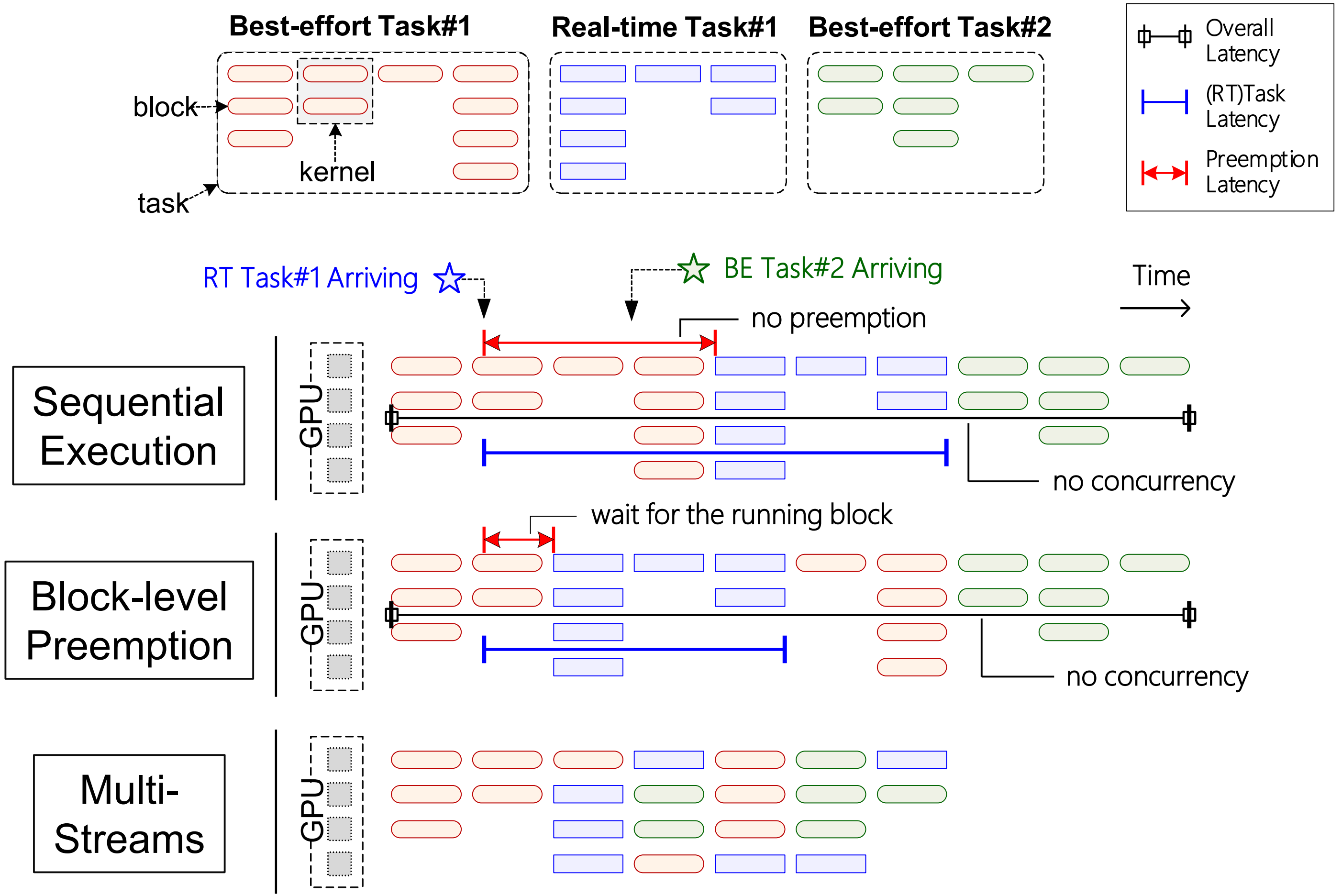
Existing GPU Task Scheduling



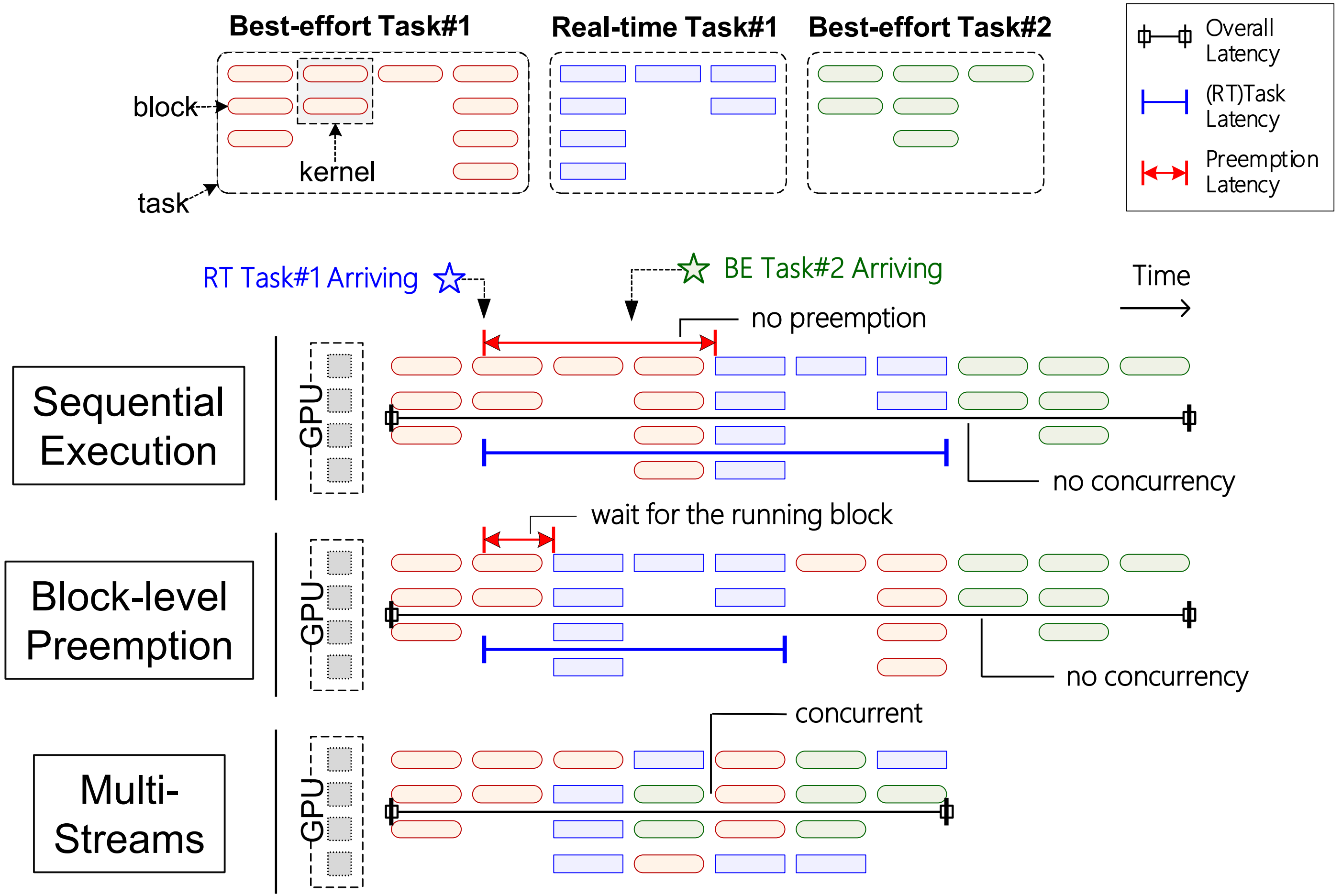
Existing GPU Task Scheduling



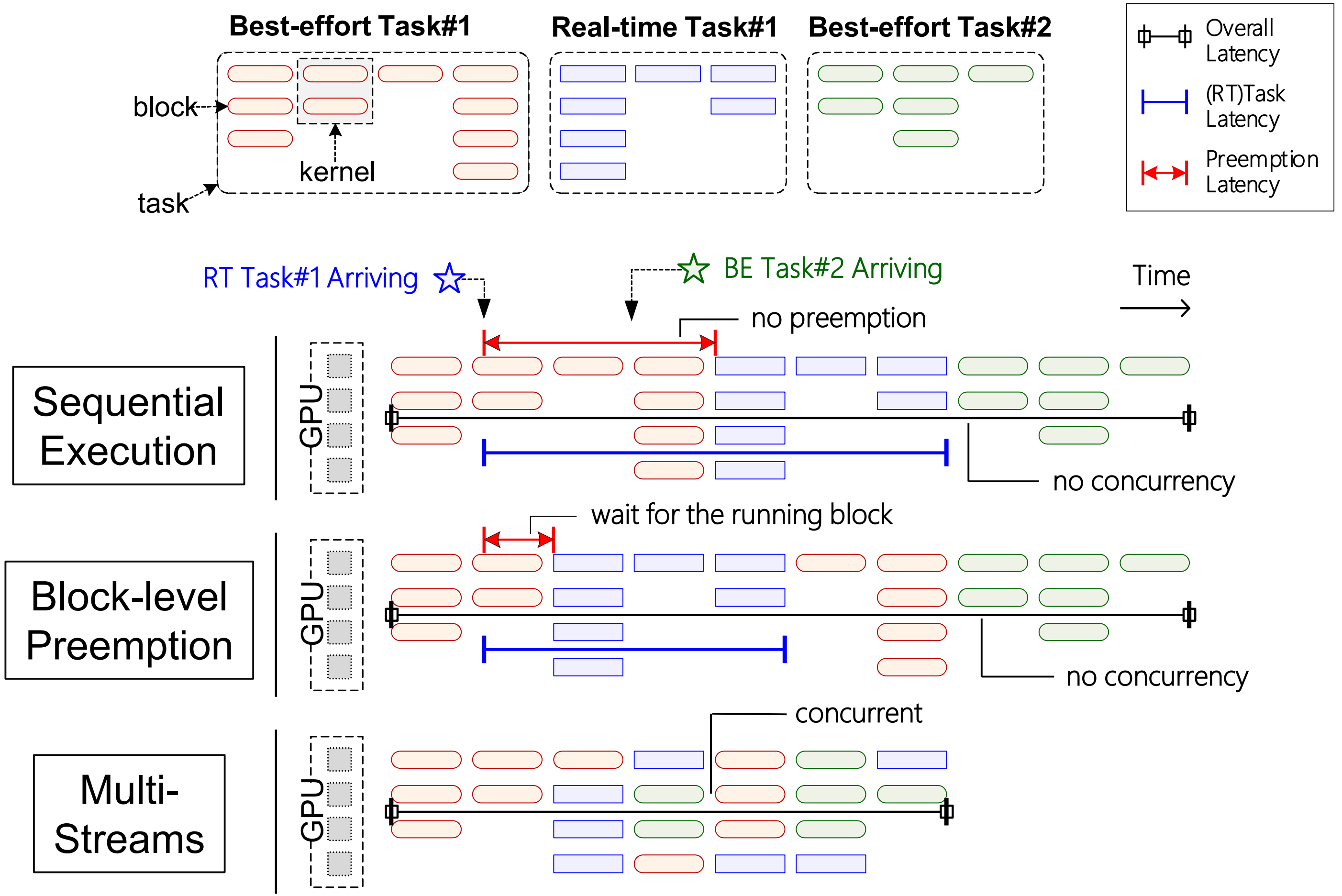
Existing GPU Task Scheduling

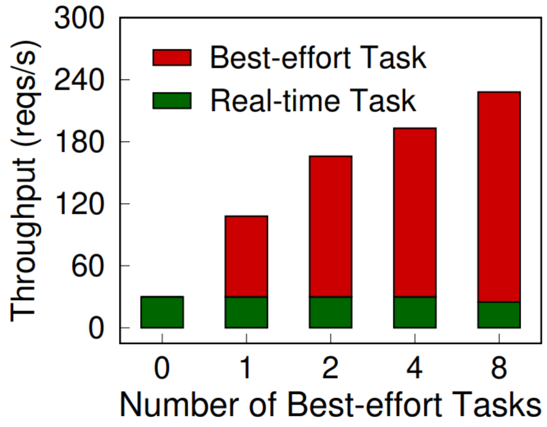


Existing GPU Task Scheduling

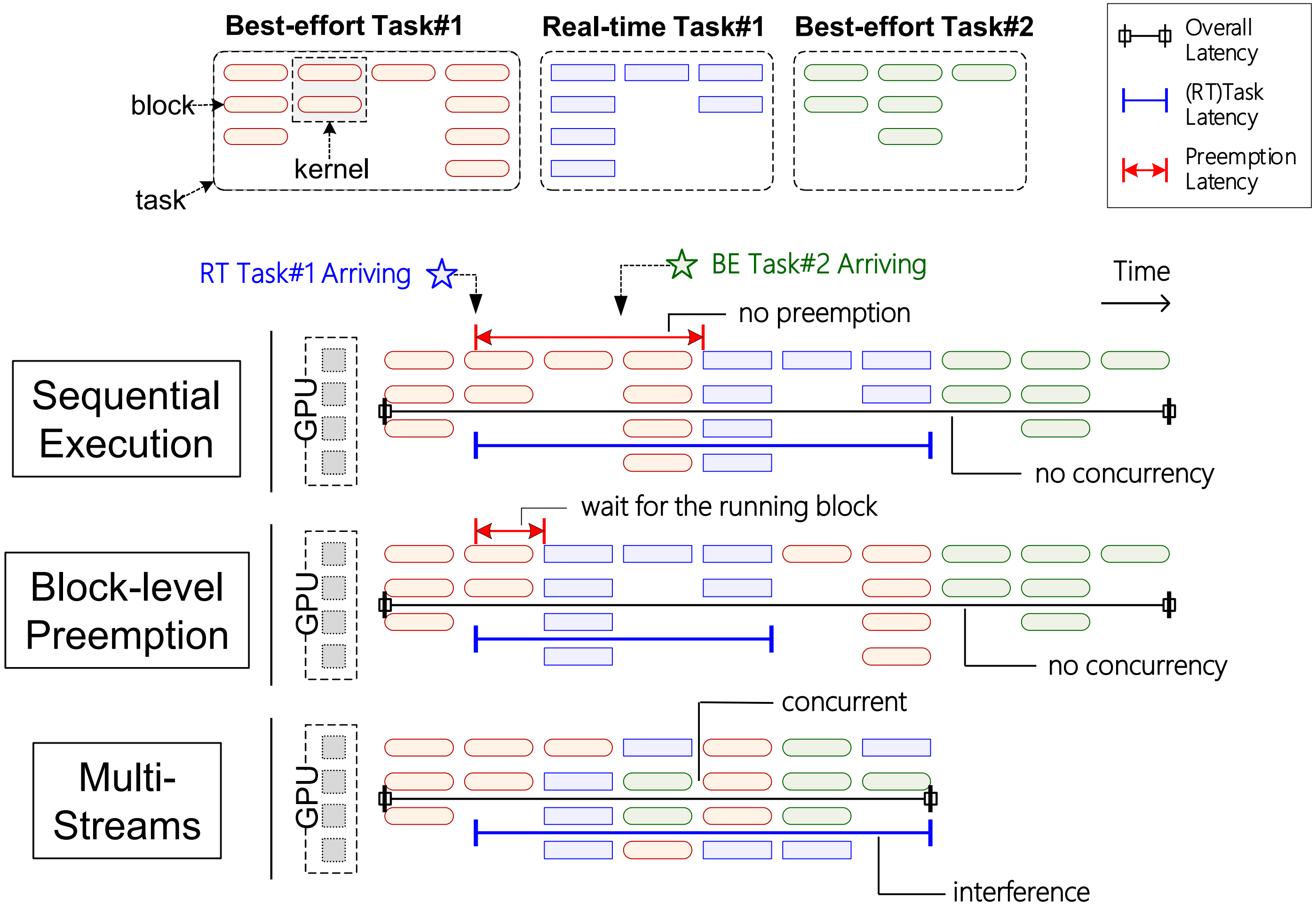


Existing GPU Task Scheduling

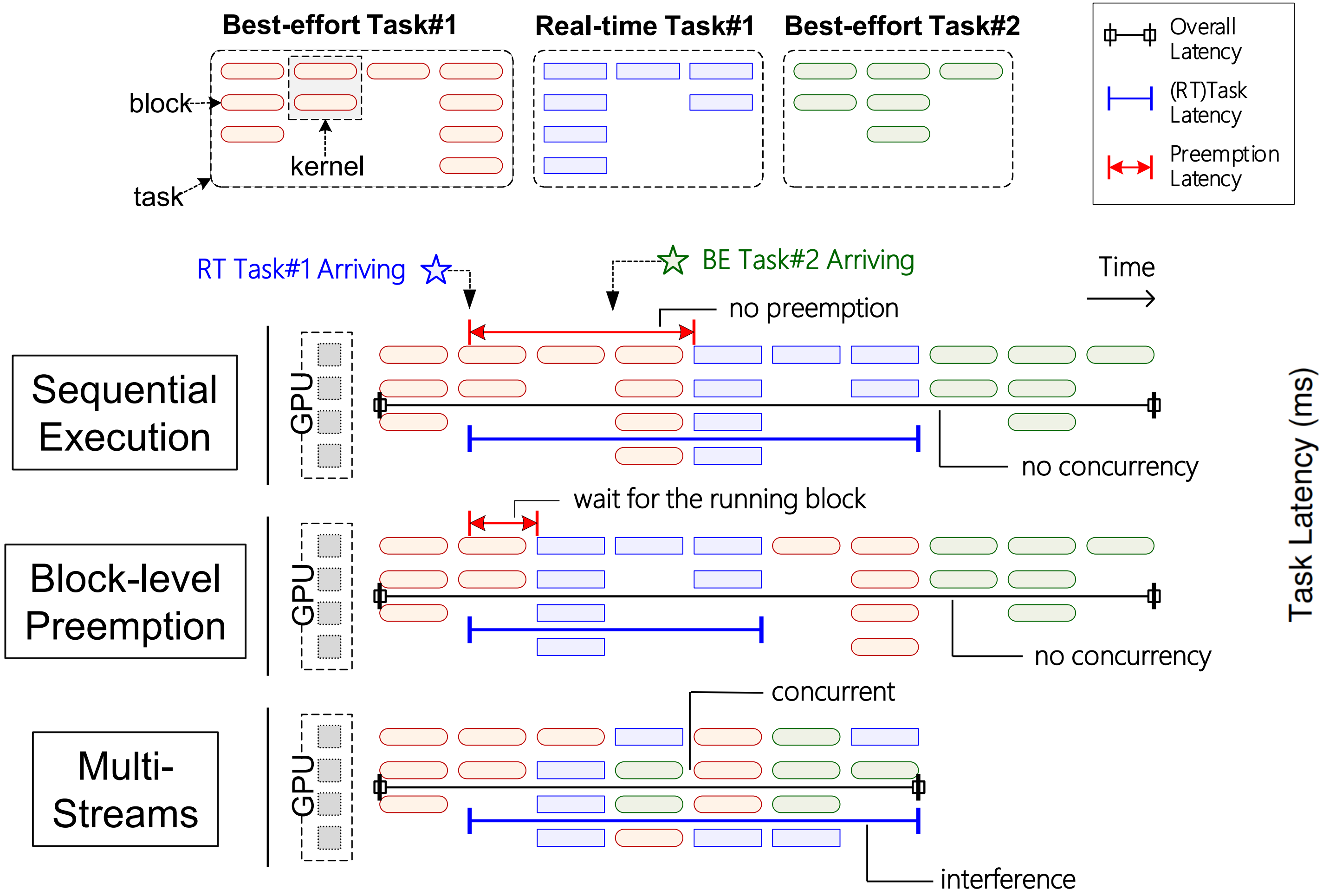


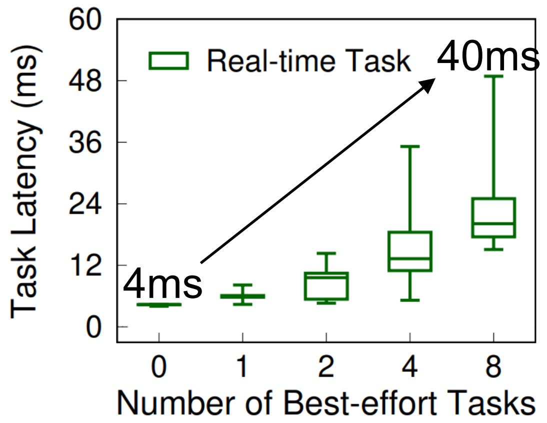


Existing GPU Task Scheduling

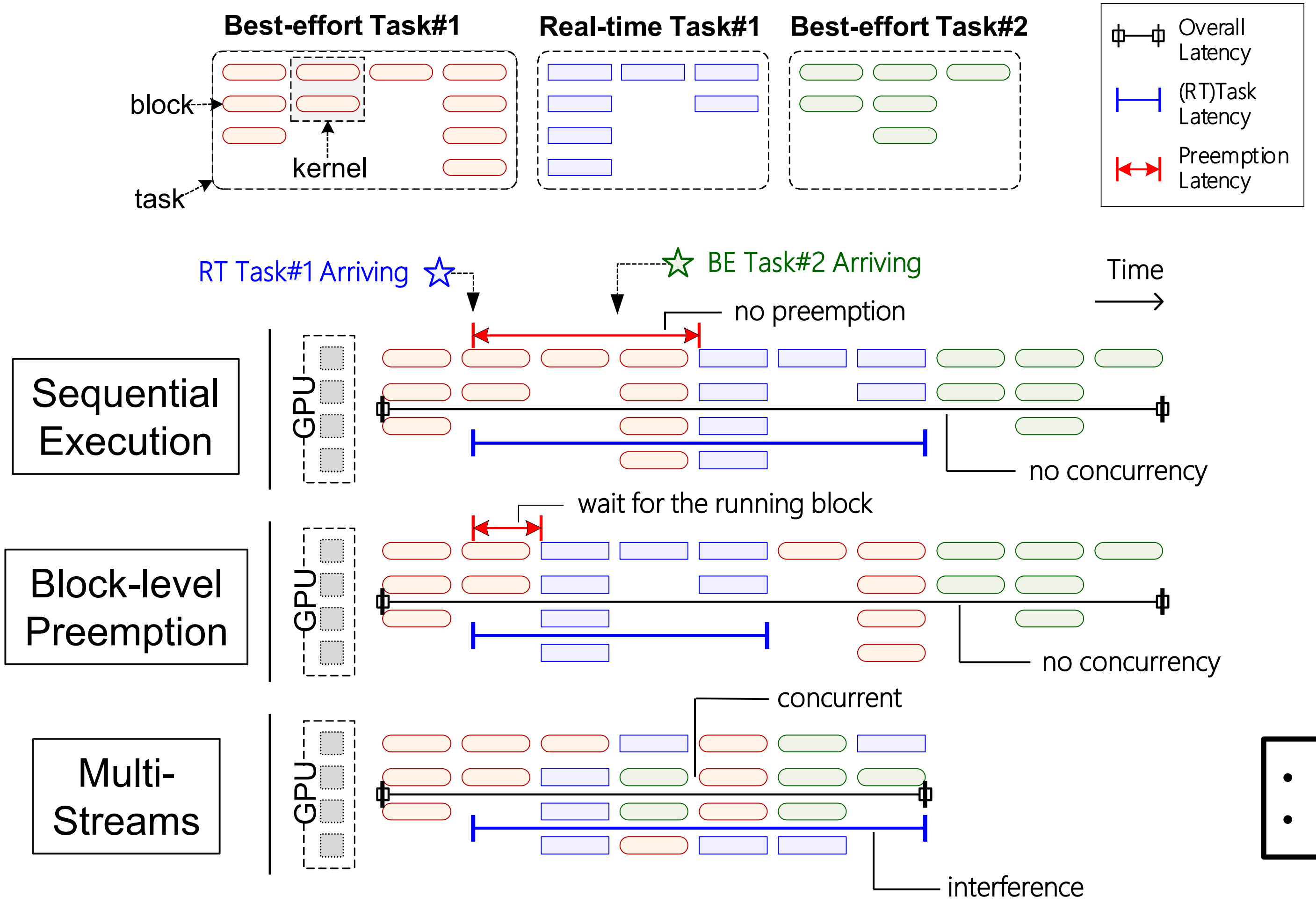


Existing GPU Task Scheduling



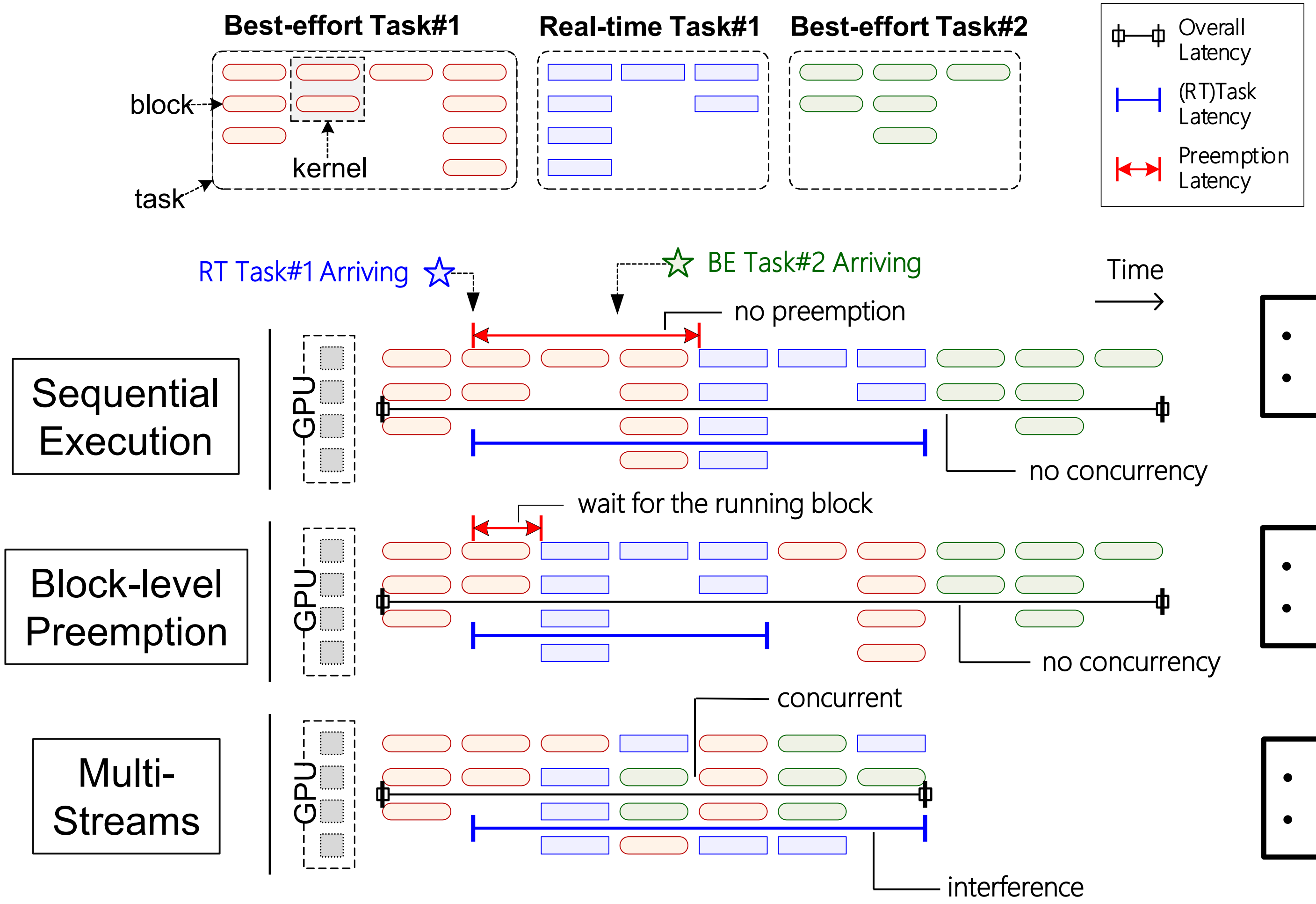


Existing GPU Task Scheduling



- High latency for **RT** tasks
- High throughput (work-conserving)

Existing GPU Task Scheduling

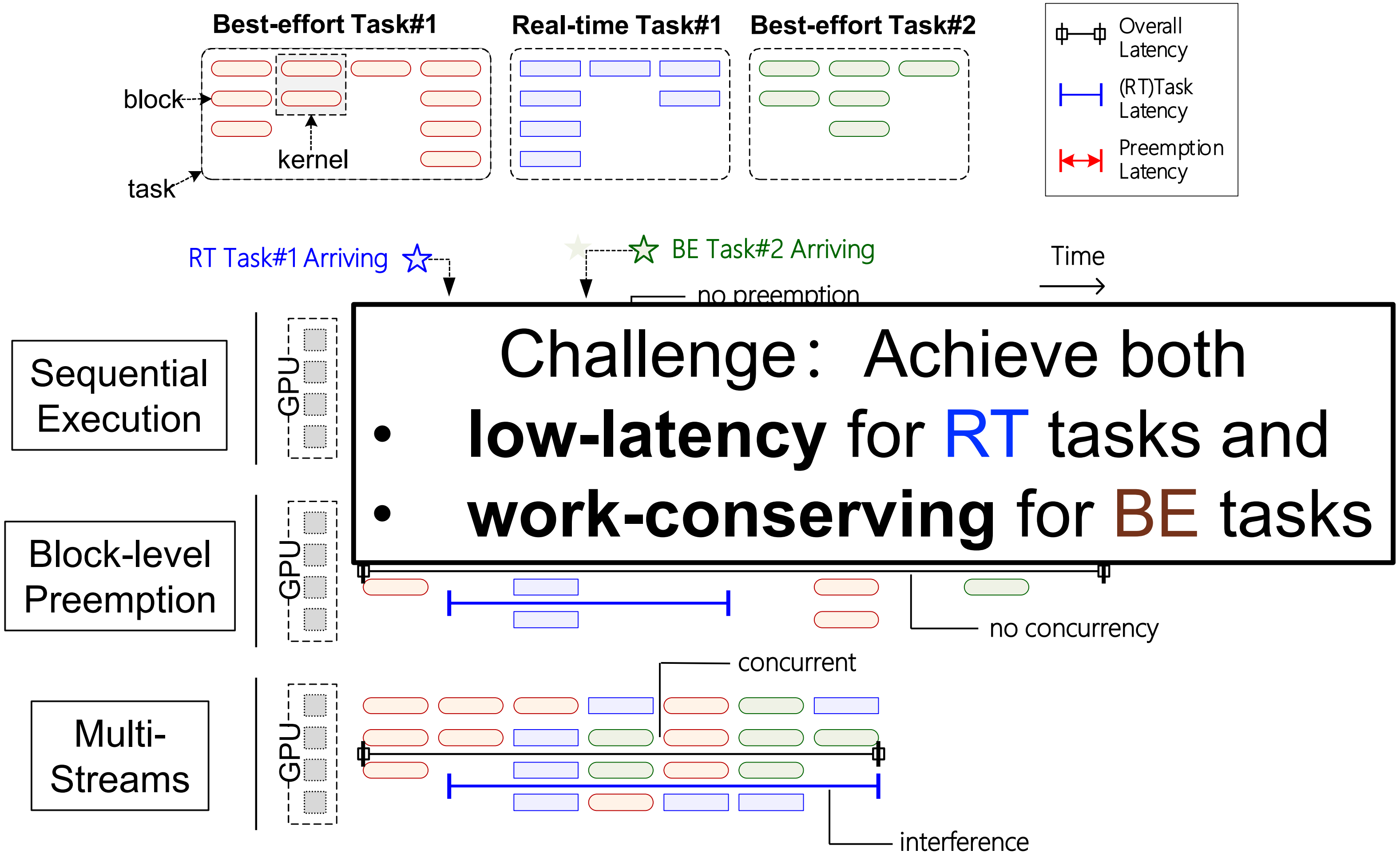


- High latency for **RT** tasks
- Low throughput (not work-conserving)

- (Not always) low latency for **RT** tasks
- Low throughput (not work-conserving)

- High latency for **RT** tasks
- High throughput (work-conserving)

Existing GPU Task Scheduling



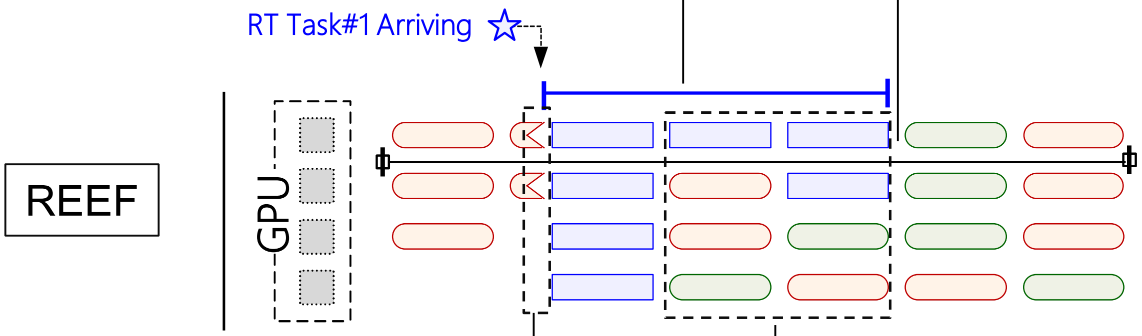
Challenge： Achieve both

- **low-latency** for RT tasks and
- **work-conserving** for BE tasks

REEF: GPU-accelerated DNN Inference System

Design Goal 1:
- Low-latency for real-time tasks

Design Goal 2:
- Work-conserving for best-effort tasks

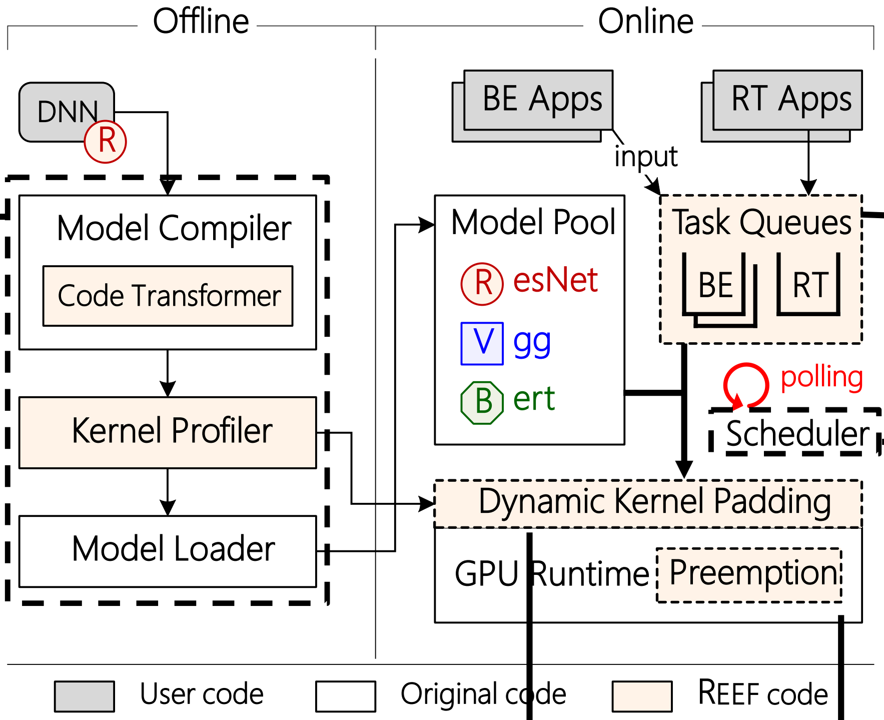


Reset-based Preemption:
- μs-scale preemption based on *idempotence*

Dynamic Kernel Padding:
- controlled concurrent execution based on *latency predictability*

REEF overview: architecture

generate DNN models



Task queues:

- one FIFO queue for **RT** tasks
- several FIFO queues for **BE** tasks

Execution modes:

normal mode:

- no RT task in RT queue
- execute BE tasks concurrently

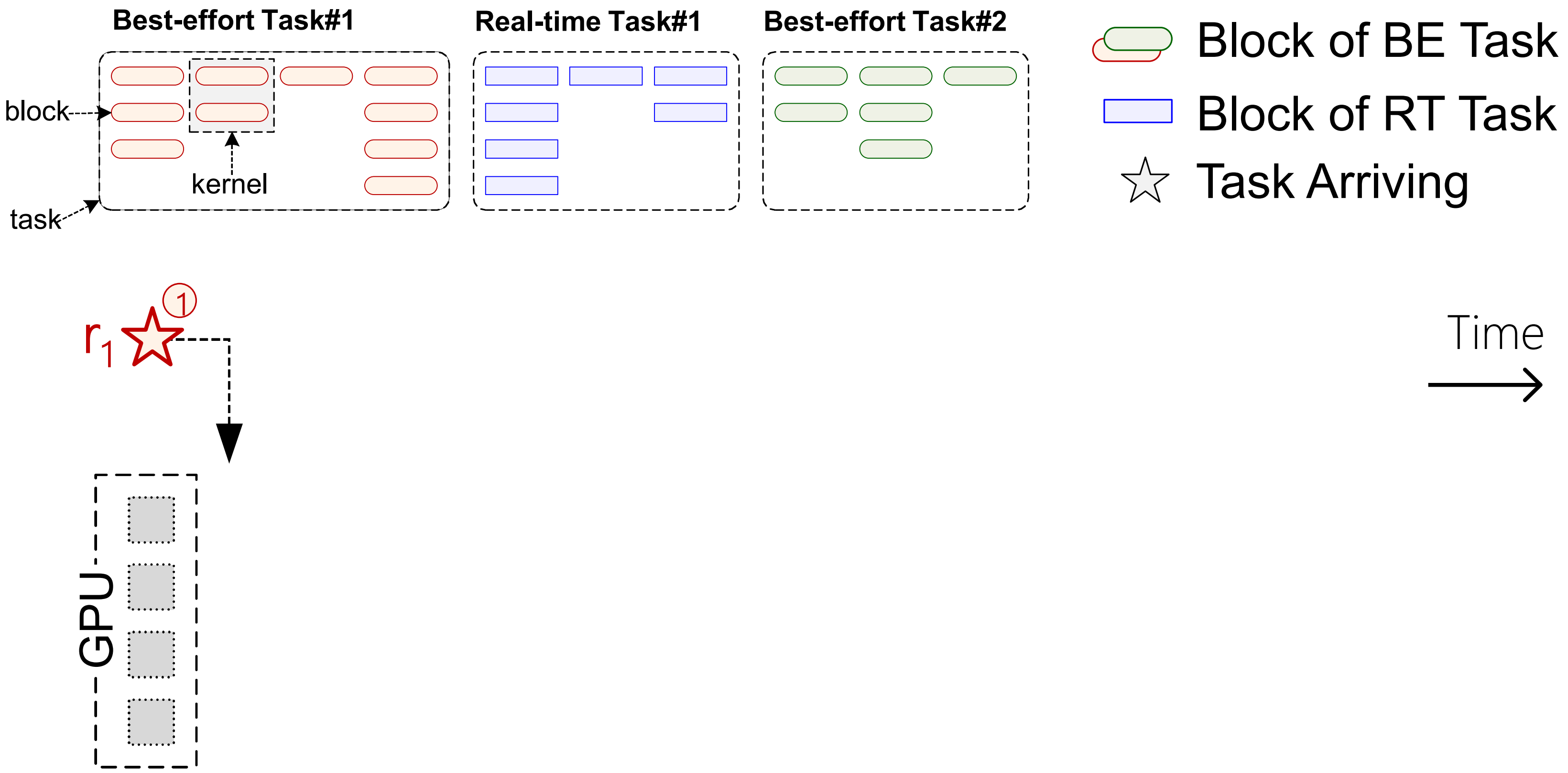
real-time mode:

- at least one RT task in RT queue
- execute one RT task a time

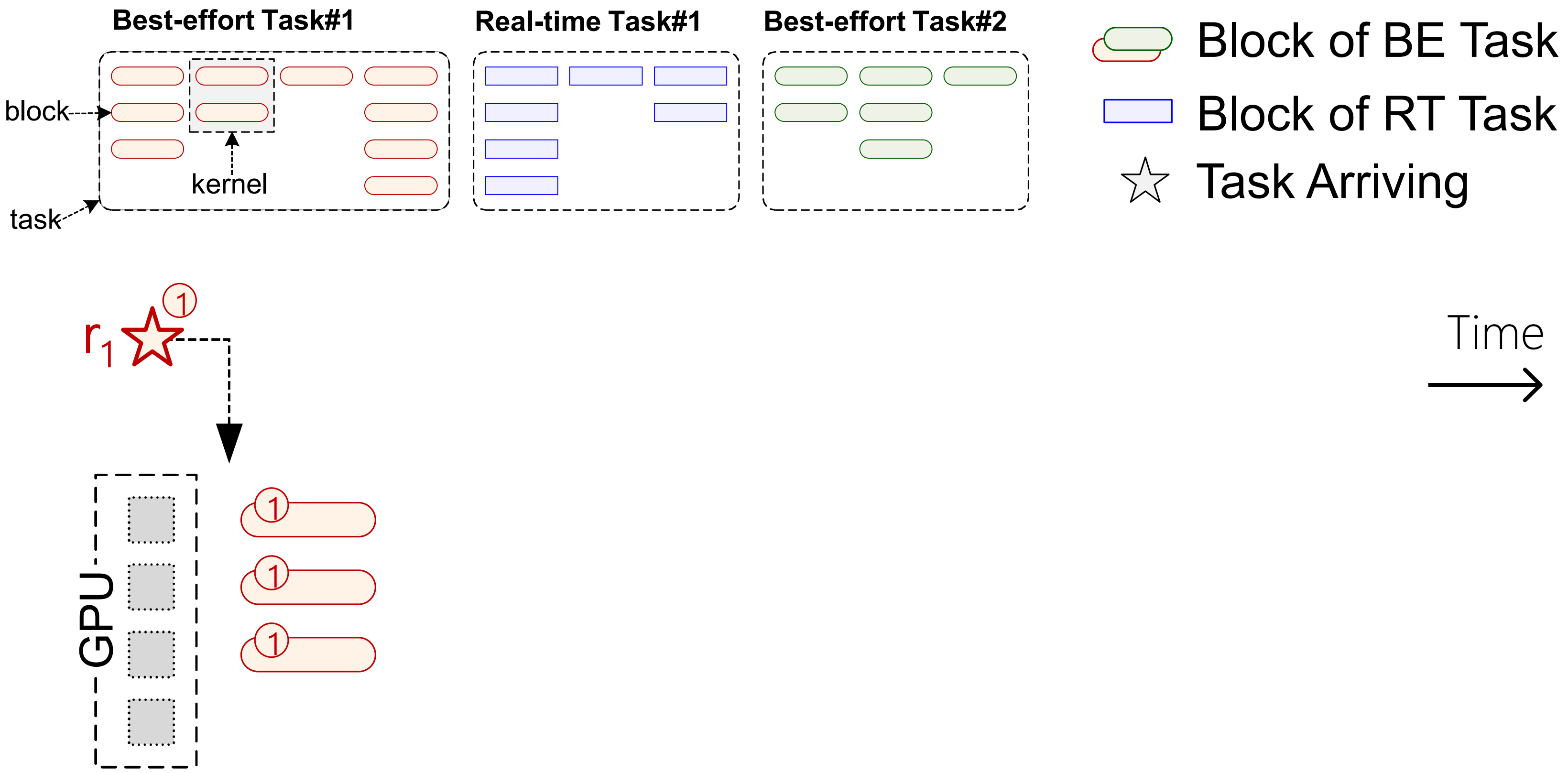
execute both **RT** and **BE** tasks concurrently in ***real-time mode***

preempt **BE** tasks when **RT** task arrives in ***normal mode***

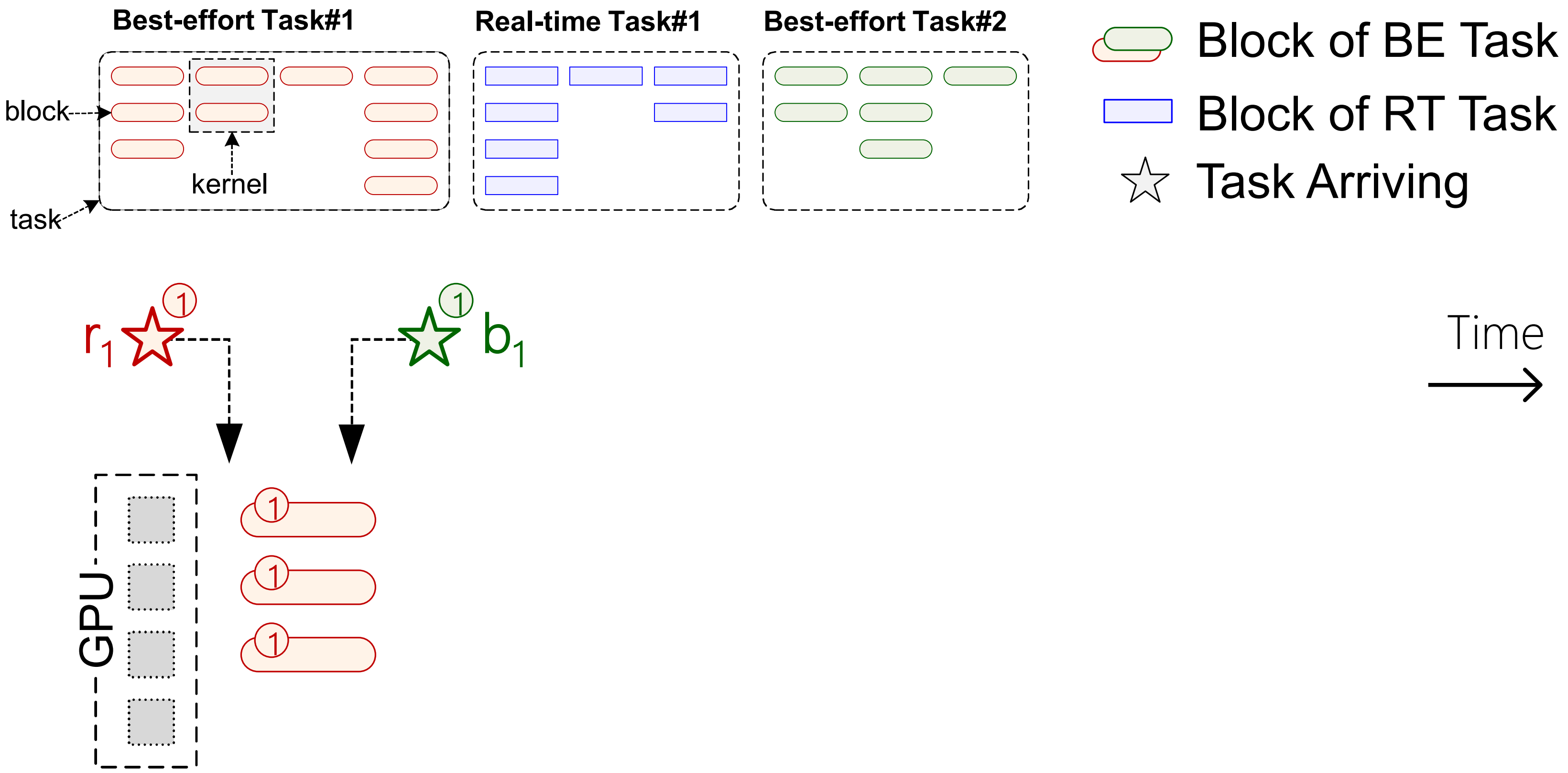
REEF overview: scheduling example



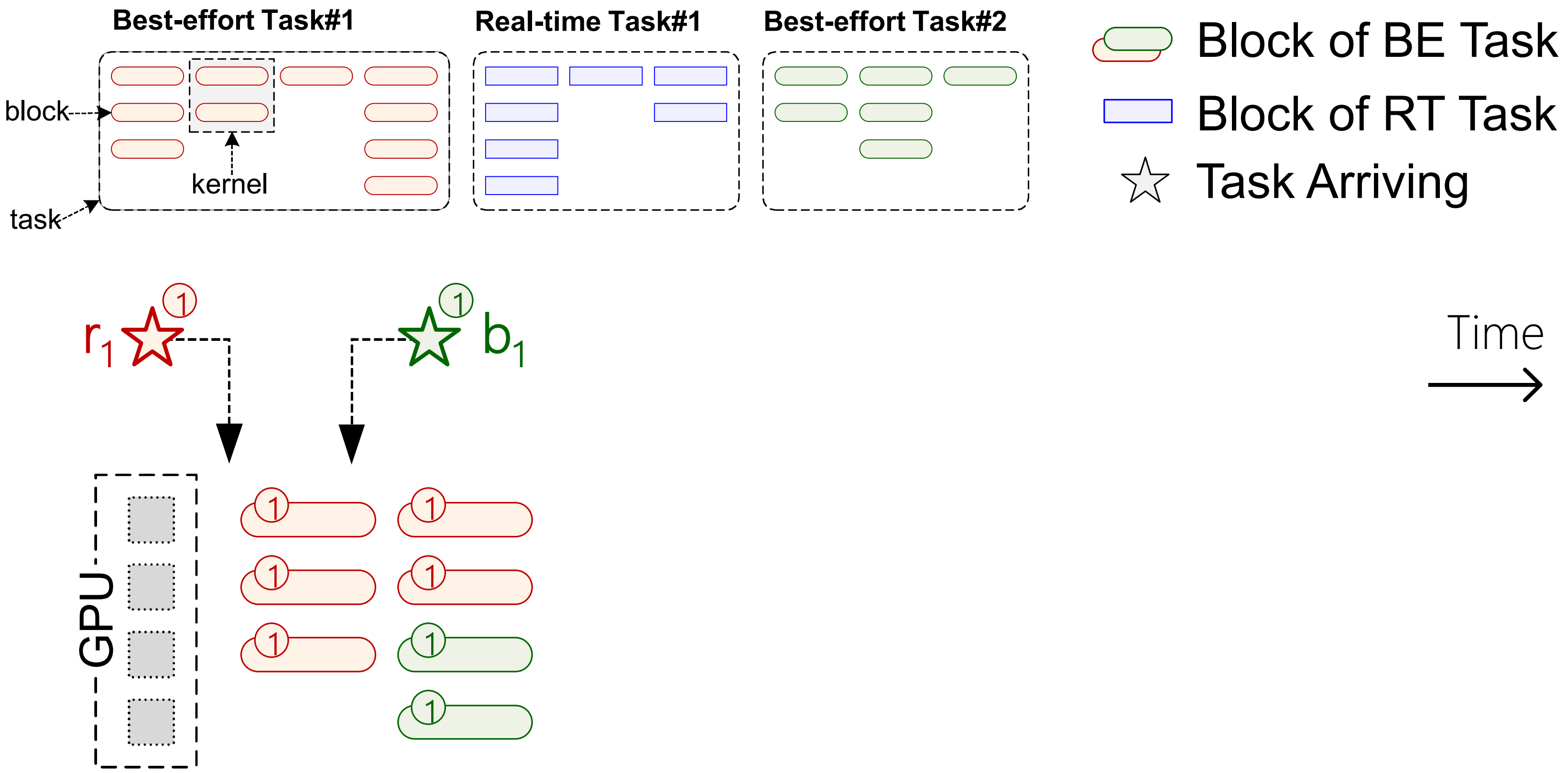
REEF overview: scheduling example



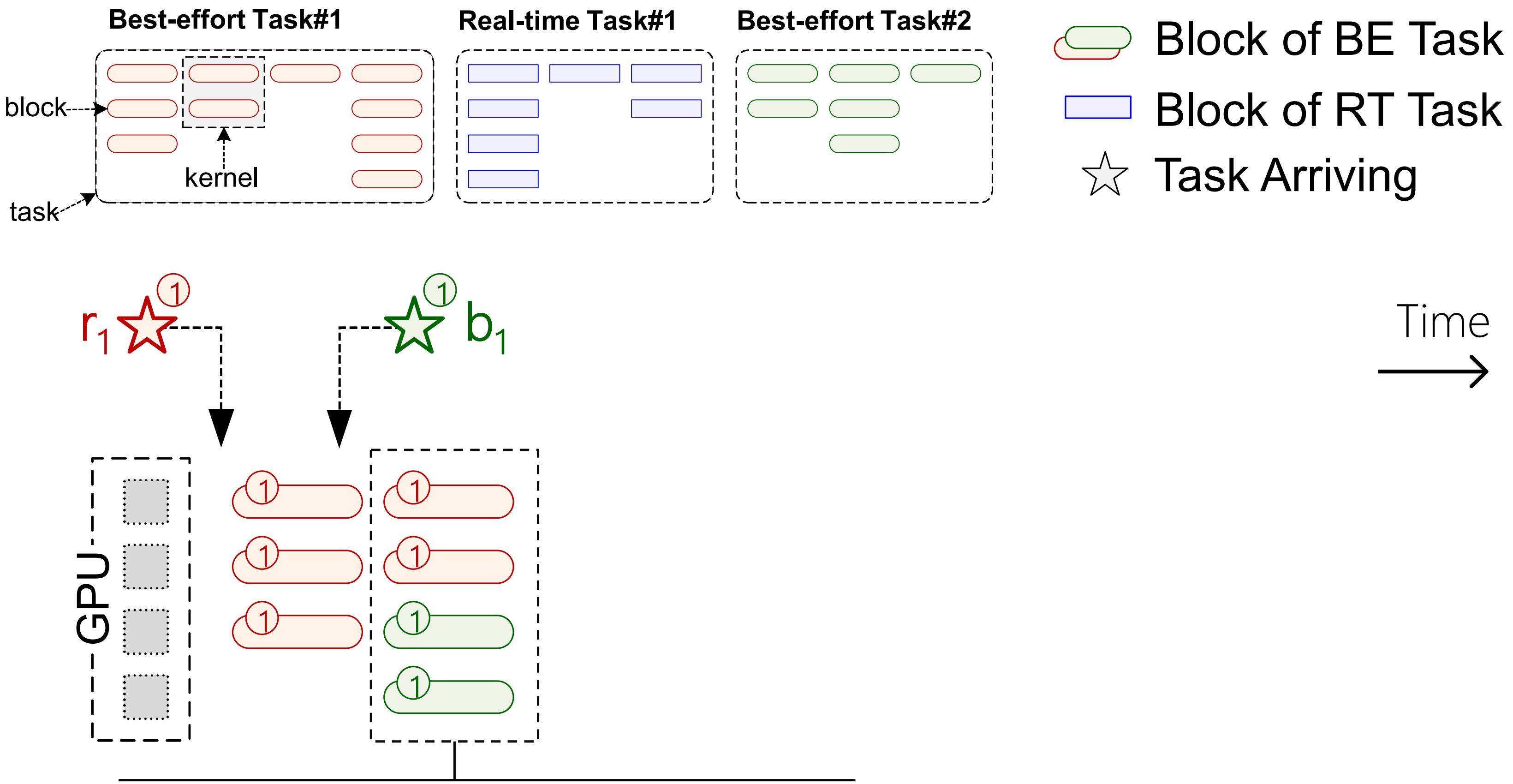
REEF overview: scheduling example



REEF overview: scheduling example

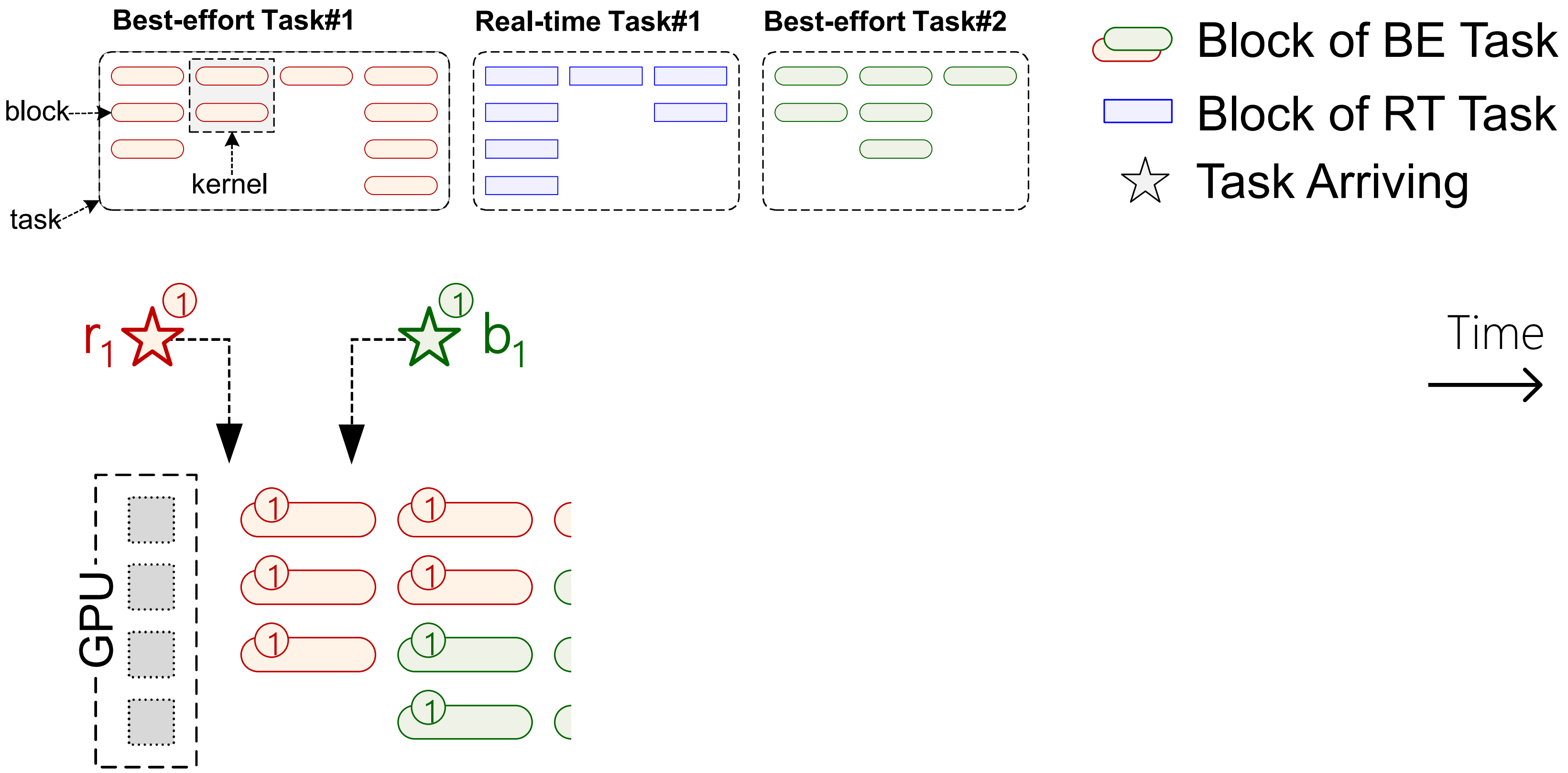


REEF overview: scheduling example

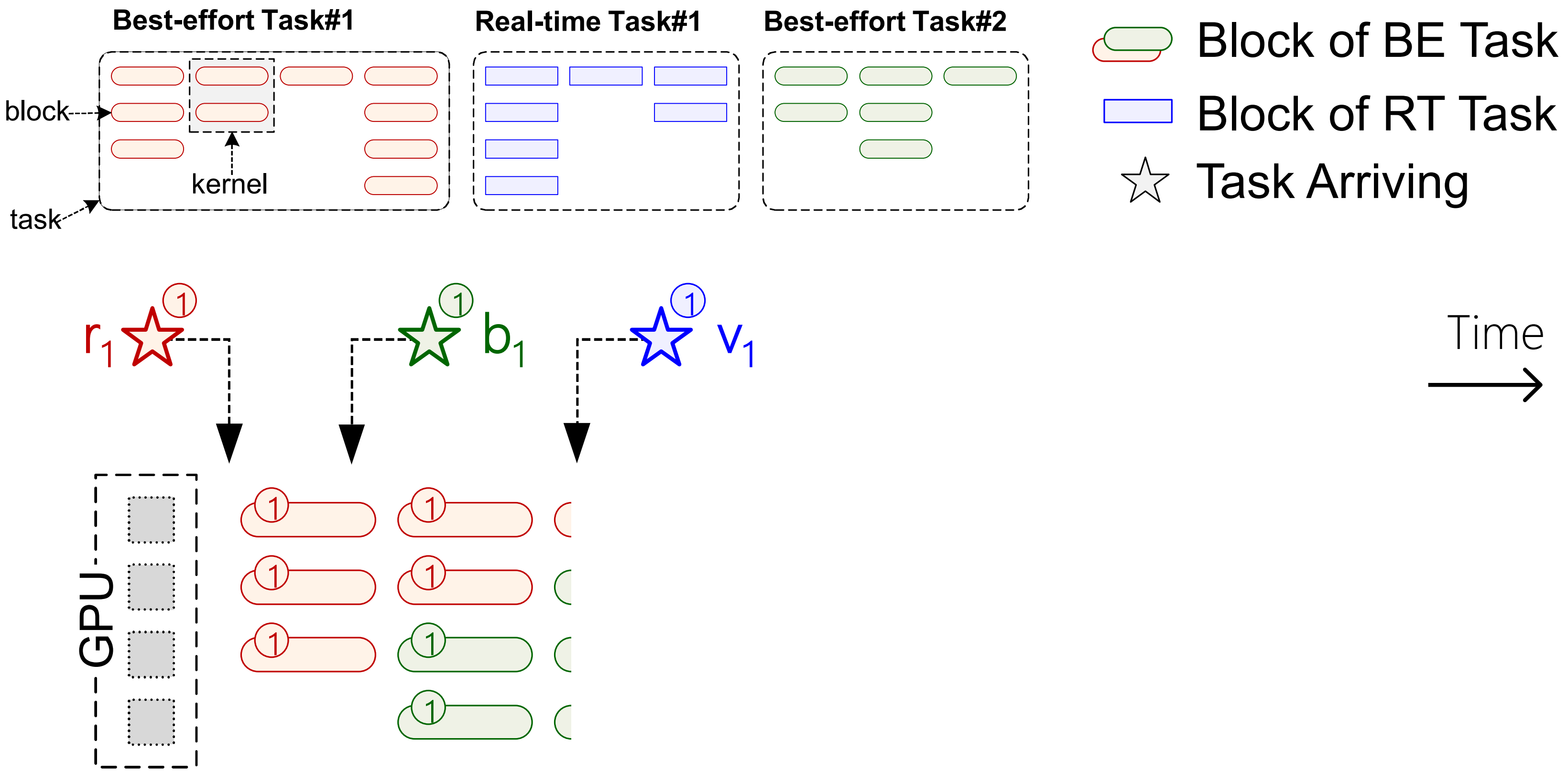


In normal mode, kernels are executed concurrently in **multiple GPU streams**

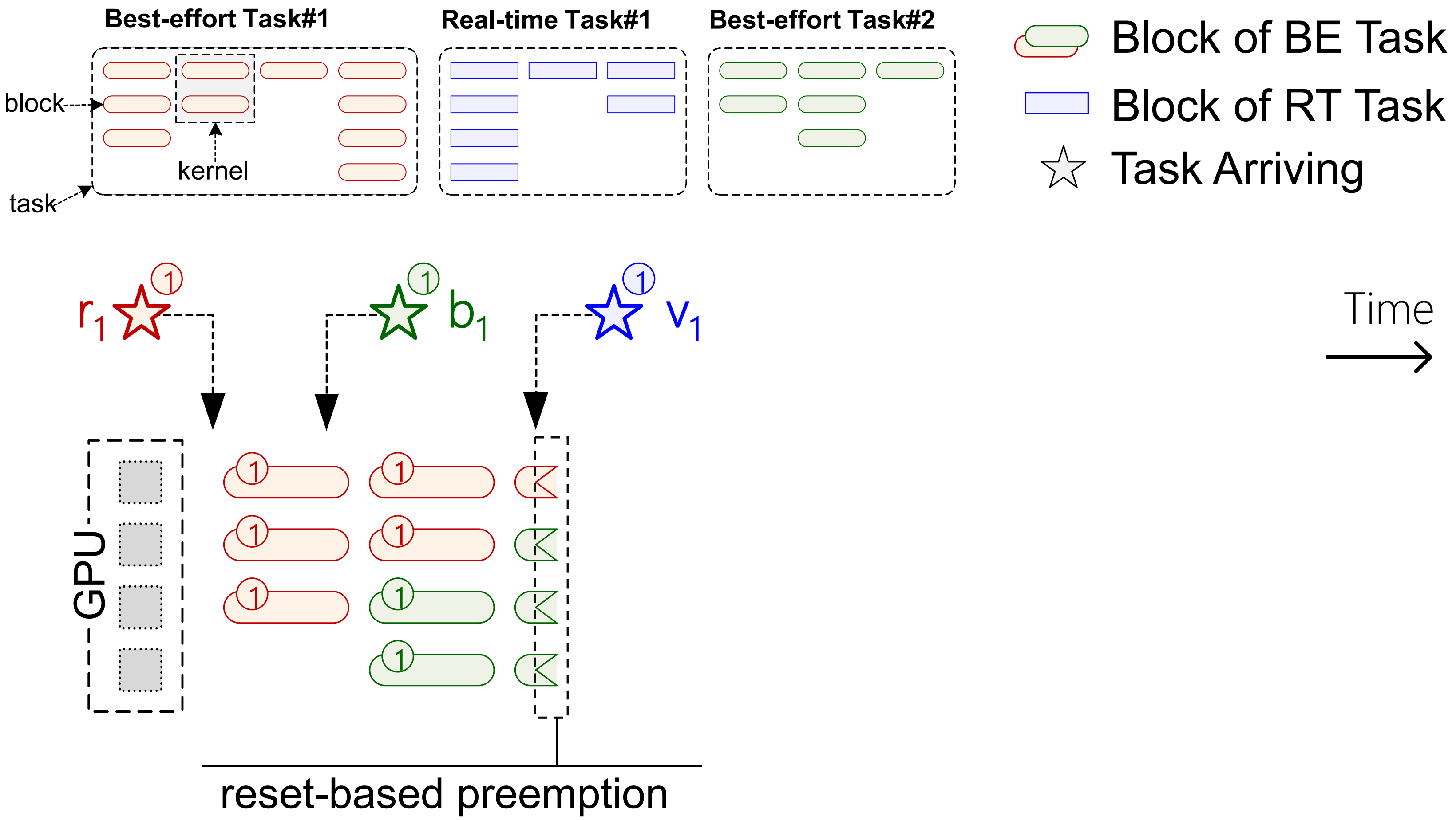
REEF overview: scheduling example



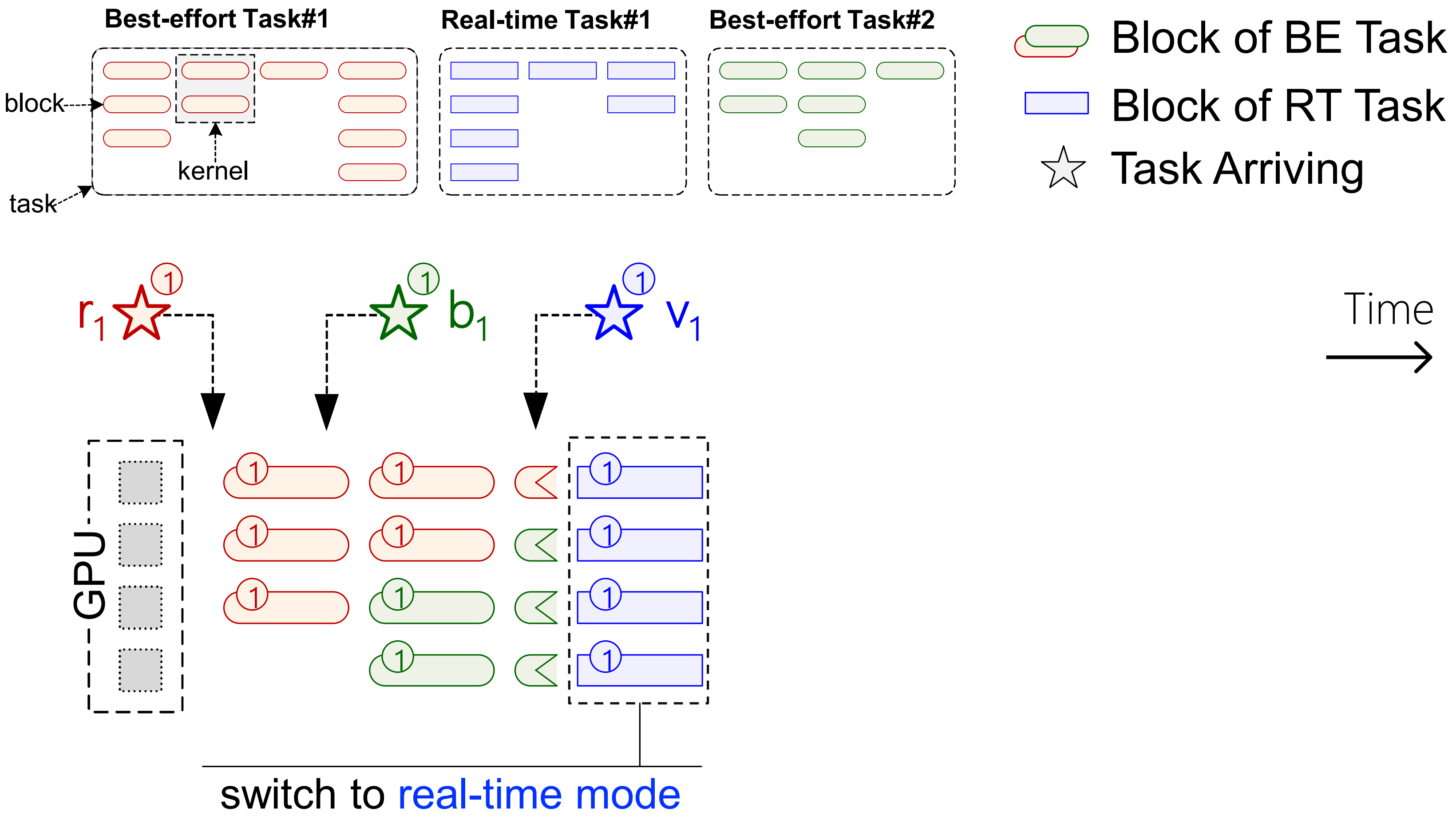
REEF overview: scheduling example



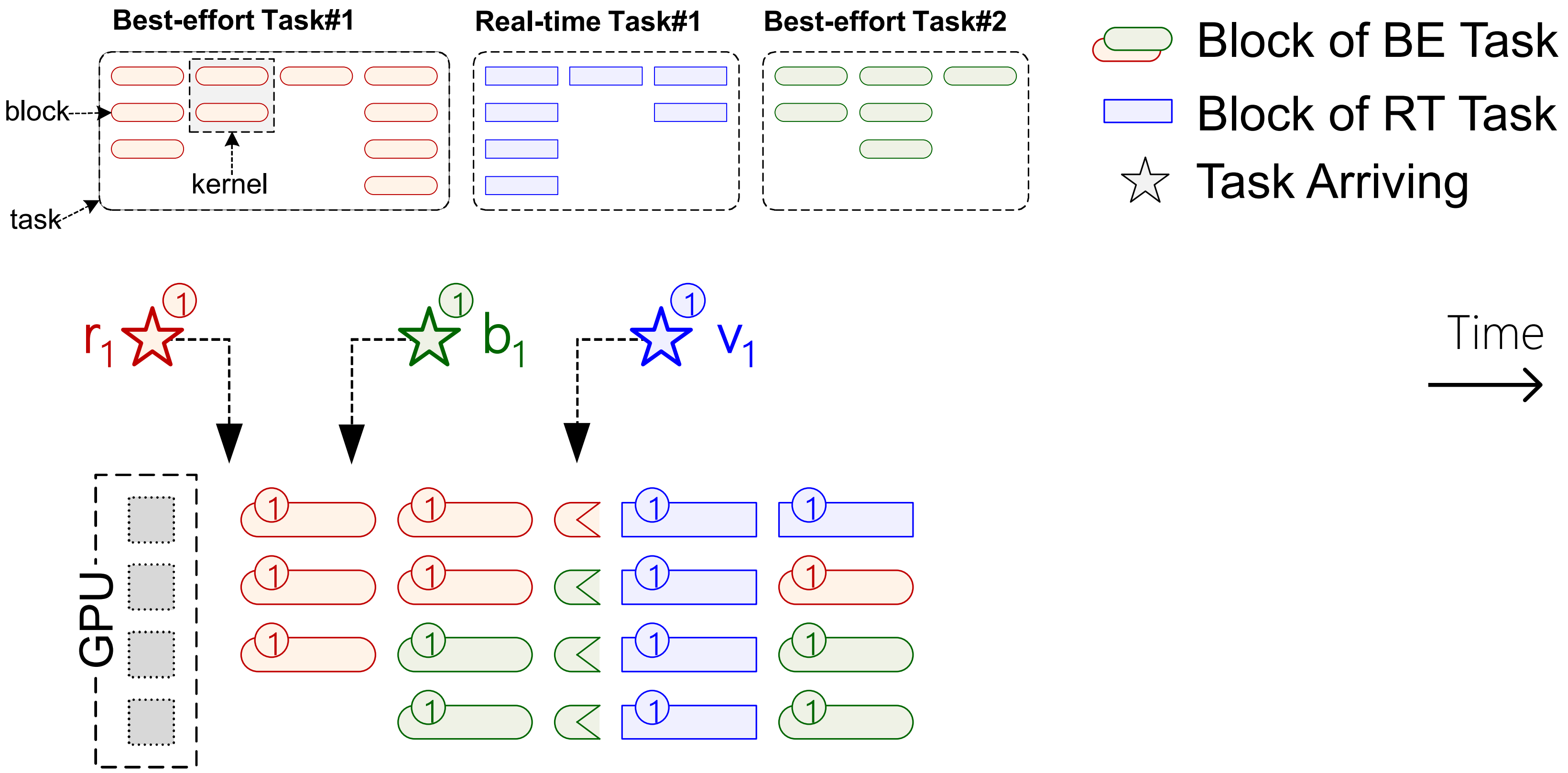
REEF overview: scheduling example



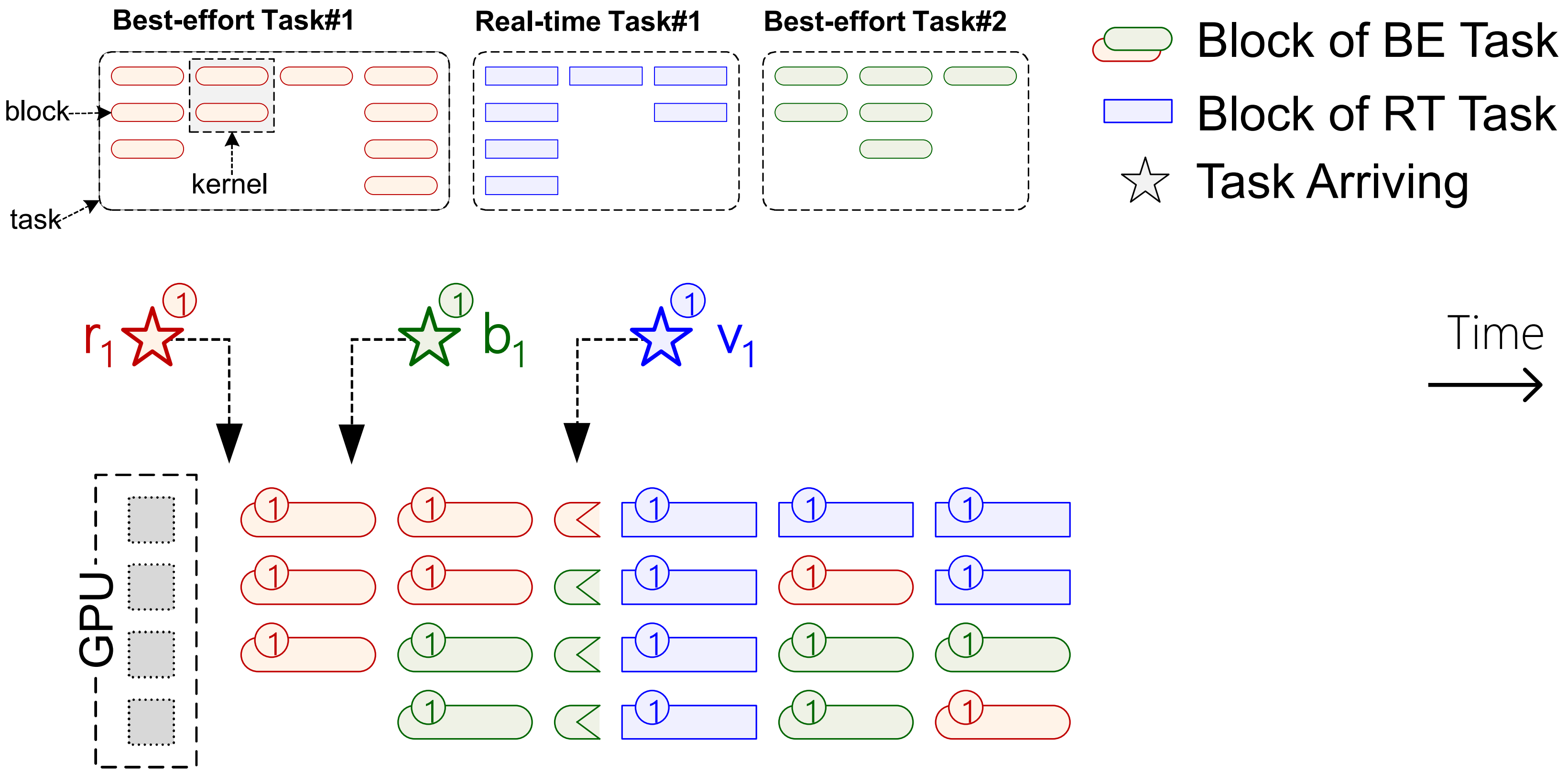
REEF overview: scheduling example



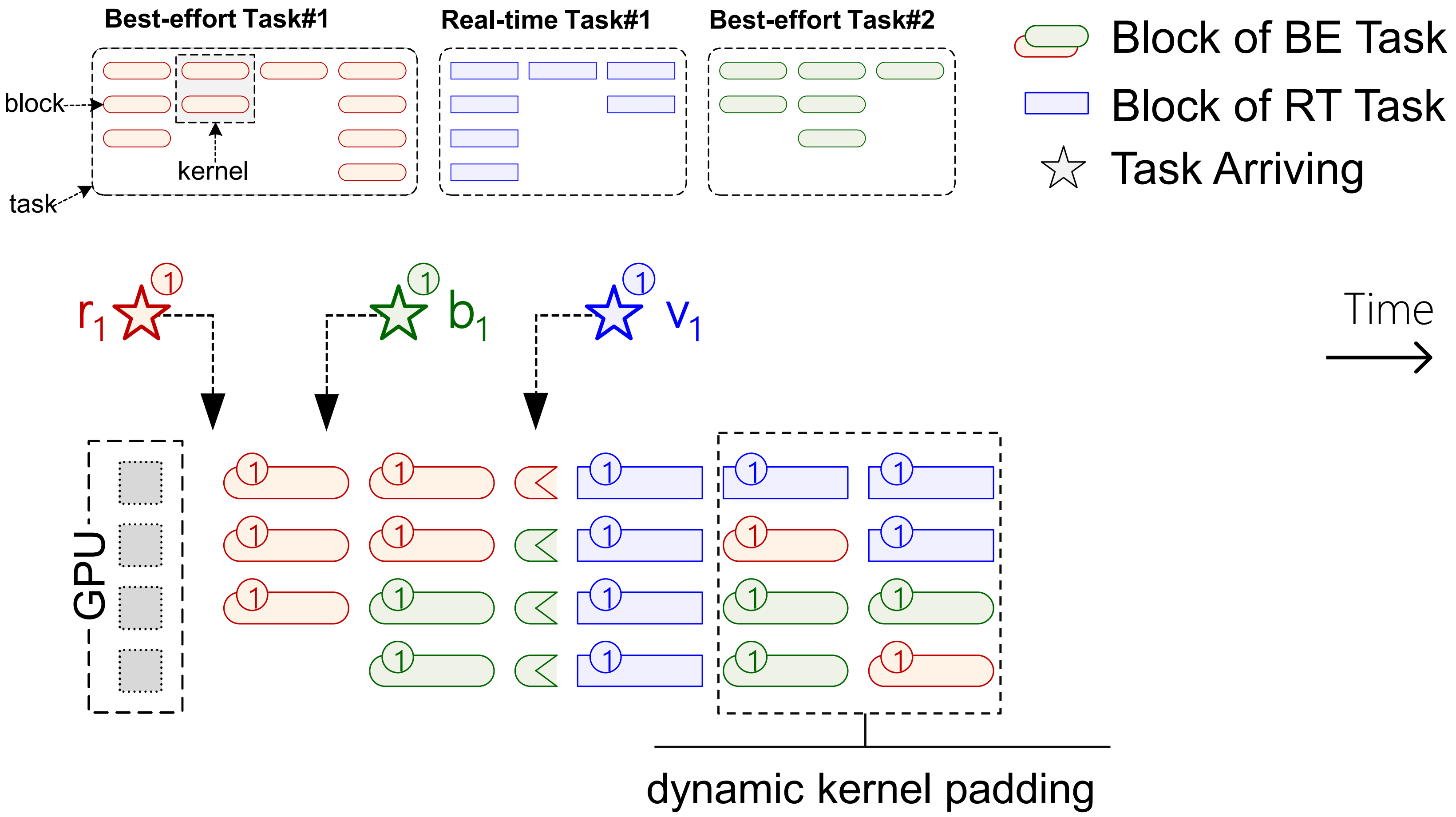
REEF overview: scheduling example



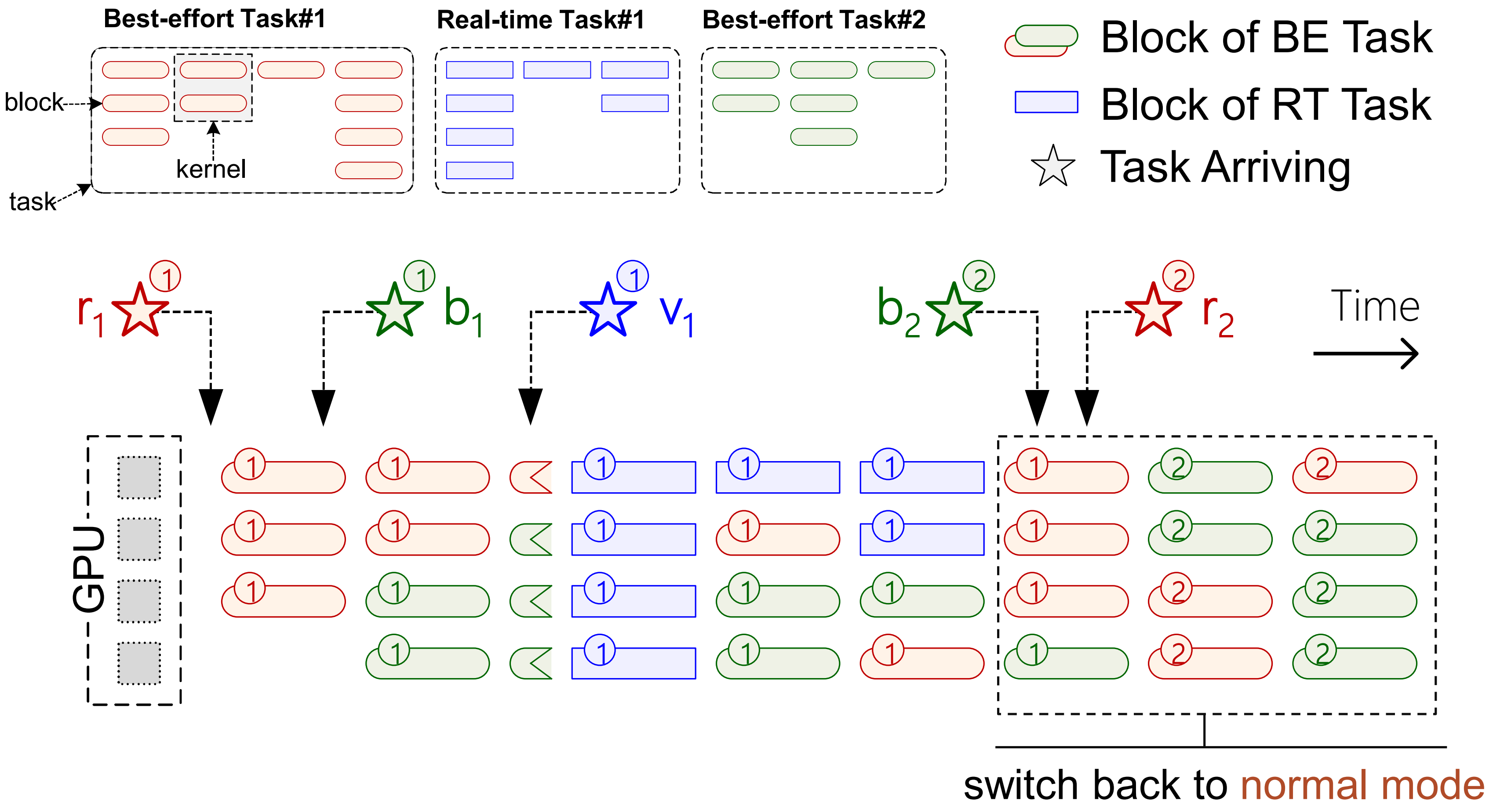
REEF overview: scheduling example



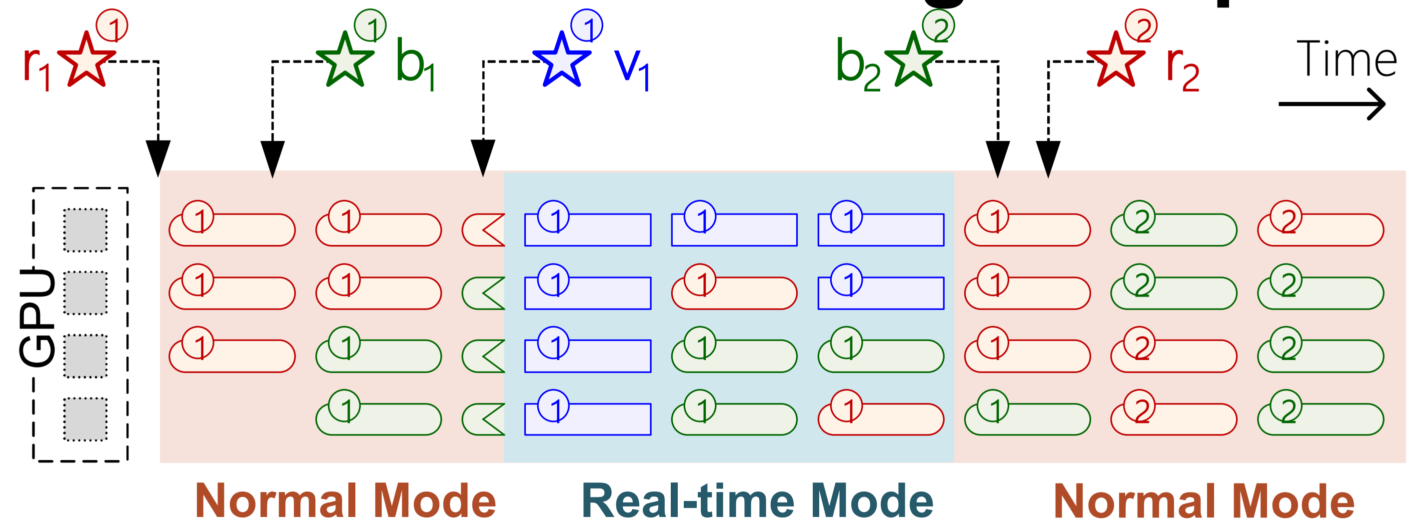
REEF overview: scheduling example



REEF overview: scheduling example

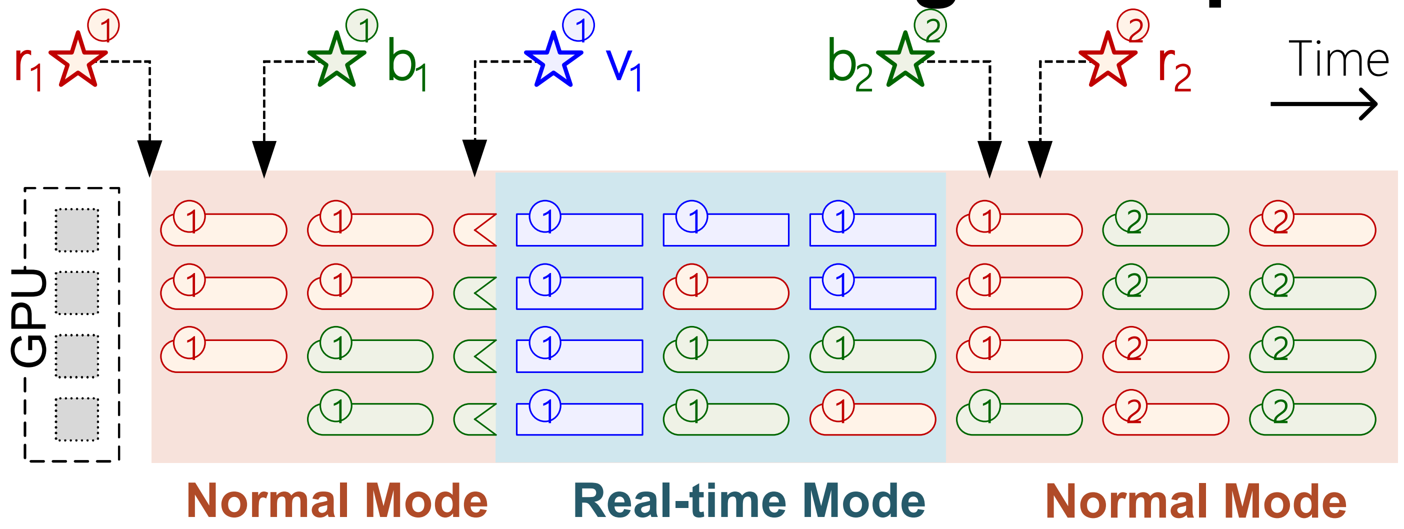


REEF overview: scheduling example



- **Low latency** for real-time tasks
  - **Normal Mode**: preempt best-effort tasks in a few μs.
  - **Real-time Mode**: get the GPU resources as many as possible.
- **Work conserving** for best-effort tasks
  - **Normal Mode**: fully utilize GPU resources by using GPU streams.
  - **Real-time Mode**: use the GPU resources leftover by real-time tasks.

REEF overview: scheduling example



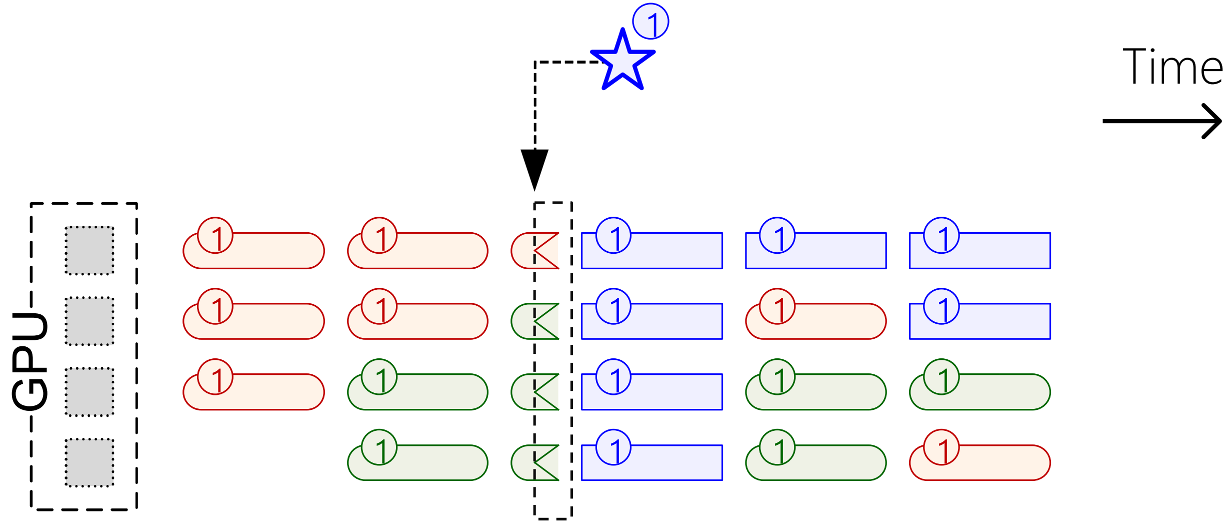
- **Low latency** for real-time tasks
  - **Normal Mode**: preempt best-effort tasks in a few μs
  - **Real-time Mode**: get the GPU resources as many as possible
- **Work conserving** for best-effort tasks
  - **Normal Mode**: fully utilize GPU resources by using GPU streams
  - **Real-time Mode**: use the GPU resources leftover by real-time tasks

Reset-based Preemption

Dynamic Kernel Padding

Reset-Based Preemption

Design Goal:
Preempt concurrent BE tasks in a few μs



Key Observation

Idempotence

```
# device codes
__global__ void conv_relu(in, weight, out):
    sum = 0;
    for i in range(0,3)
        for j in range(0,3)
            sum += in[..] × weight[..]
    out[..] = ReLU(sum)

__global__ void dense(in, weight, bias, out):
    sum = 0;
    for i in range(0,512)
        sum += in[..] × weight[..]
    out[..] = sum + bias[..]
```

Key Observation

Idempotence

Input (read-only)

Output (write-only)

```
# device codes
__global__ void conv_relu(in, weight, out):
    sum = 0;
    for i in range(0,3)
        for j in range(0,3)
            sum += in[..] × weight[..]
    out[..] = ReLU(sum)
__global__ void dense(in, weight, bias, out):
    sum = 0;
    for i in range(0,512)
        sum += in[..] × weight[..]
    out[..] = sum + bias[..]
```

Key Observation

Idempotence

Input (read-only)

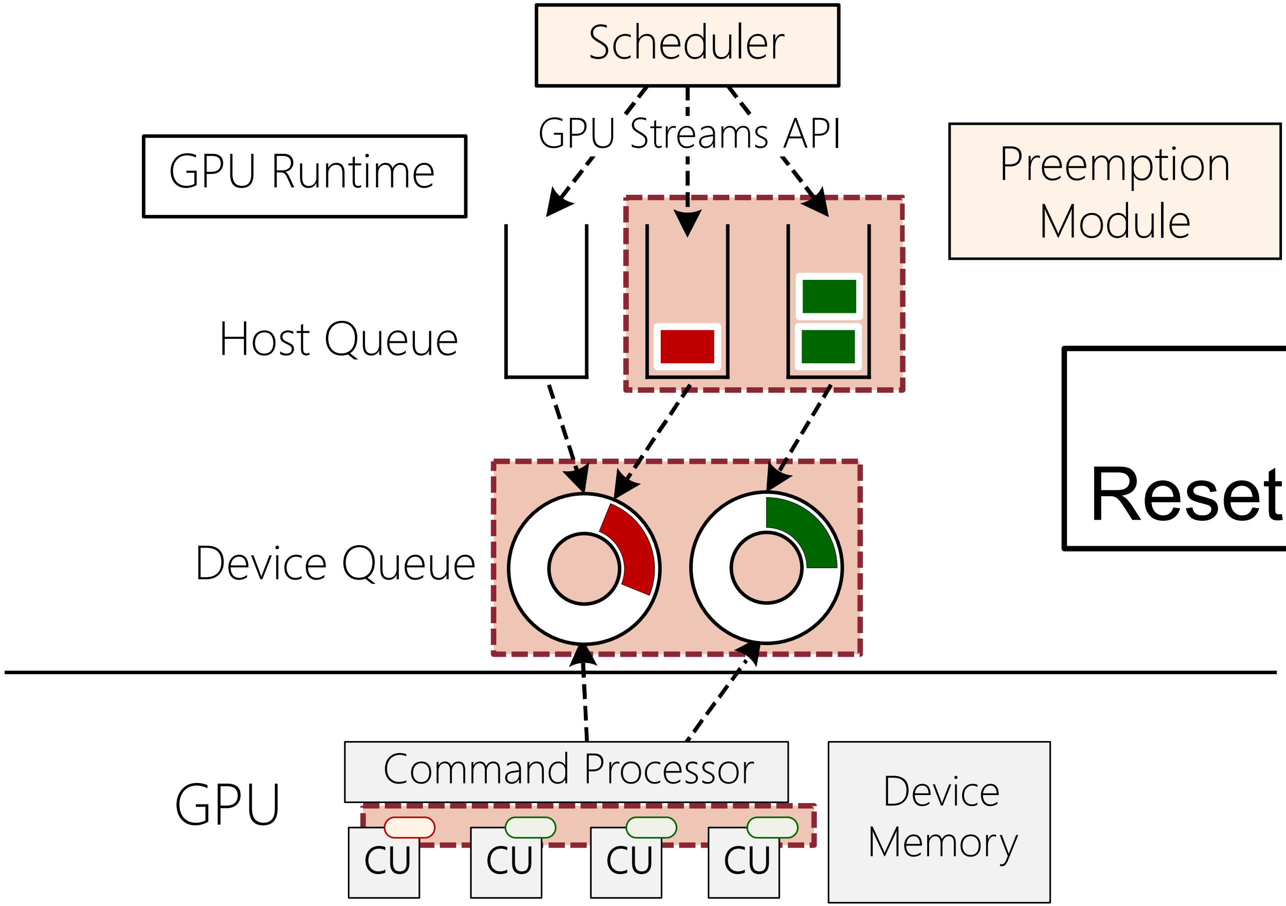
Output (write-only)

```
# device codes
__global__ void conv_relu(in, weight, out):
    sum = 0;
    for i in range(0,3)
       for j in range(0,3)
          sum += in[..] × weight[..]
    out[..] = ReLU(sum)
...bias, out):
...
       sum += in[..] × weight[..]
    out[..] = sum + bias[..]
```

Basic idea:

- *Preempt* a task by *killing* the running kernels
- *Restore* a task by *re-executing* the preempted kernels

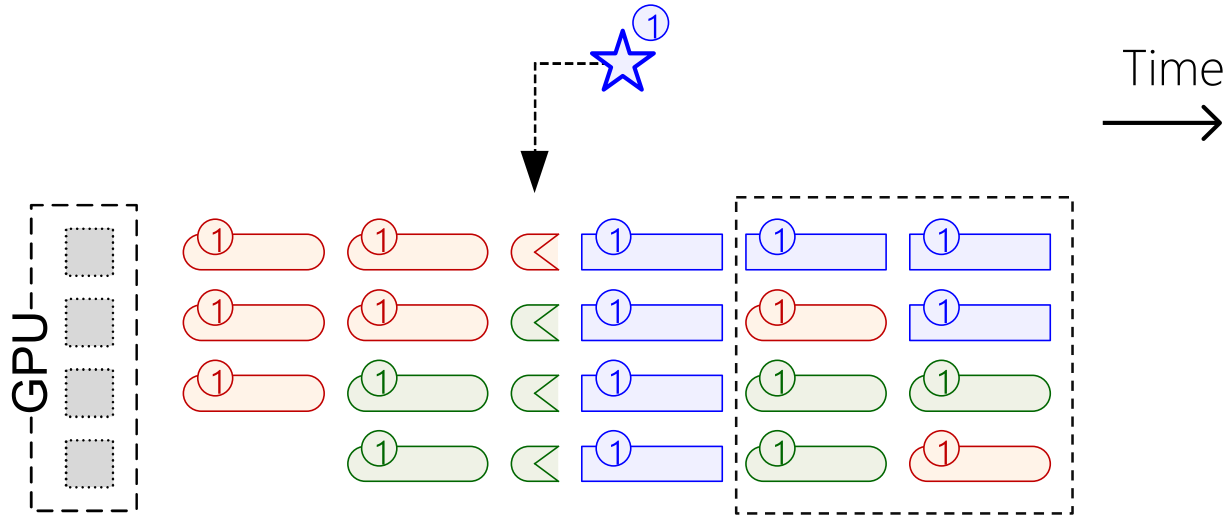
Reset-based Preemption



Key Idea:
Reset kernels in everywhere

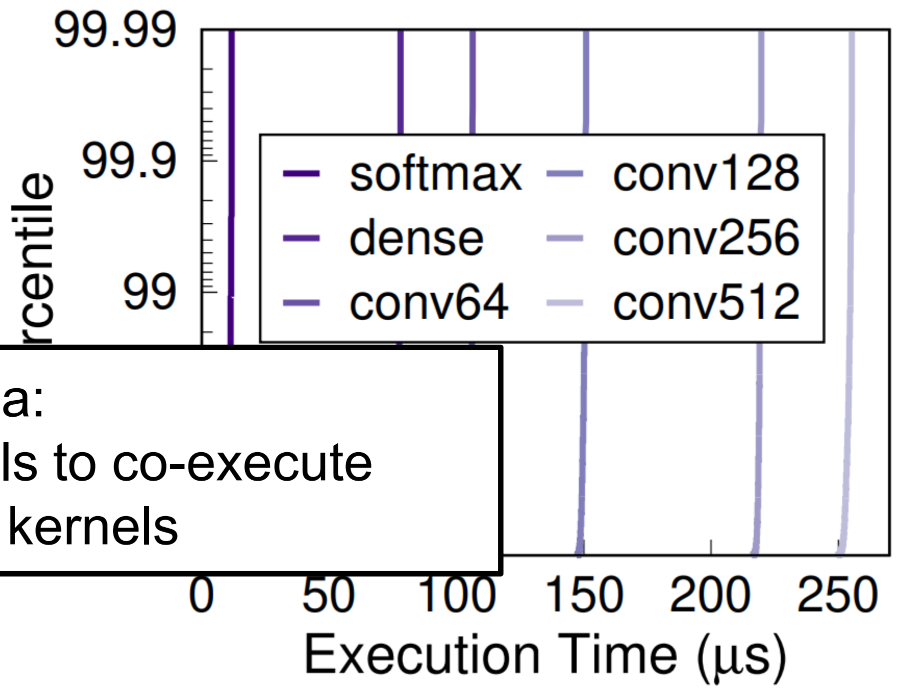
Dynamic Kernel Padding

Design Goal:
Allow RT/BE tasks to execute concurrently
without interference to RT tasks



Key Observation

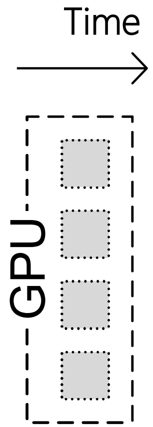
Latency Predictability



Basic Idea:
Allow *shorter* **BE** kernels to co-execute with *longer* **RT** kernels

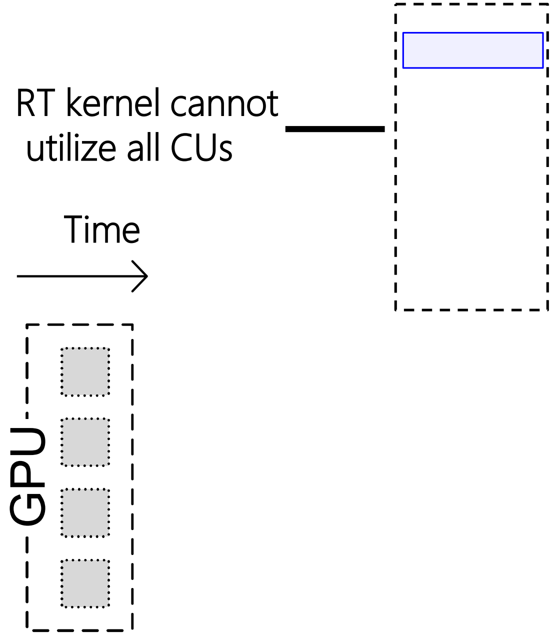
Dynamic Kernel Padding

Key Idea:
Dynamically pad RT kernels with BE kernels



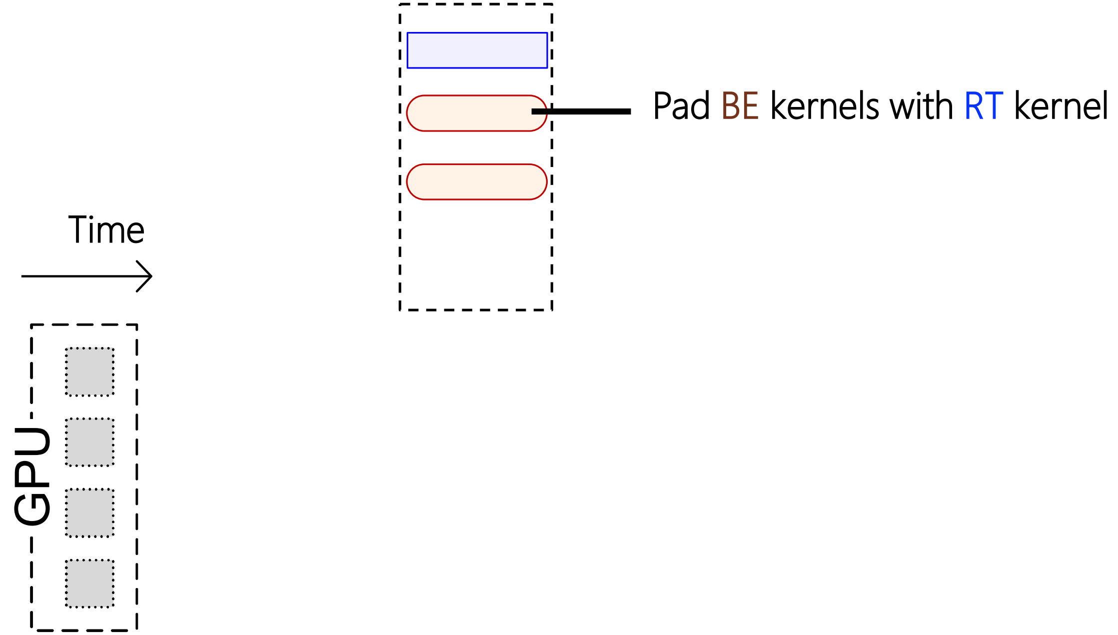
Dynamic Kernel Padding

Key Idea:
Dynamically pad RT kernels with BE kernels



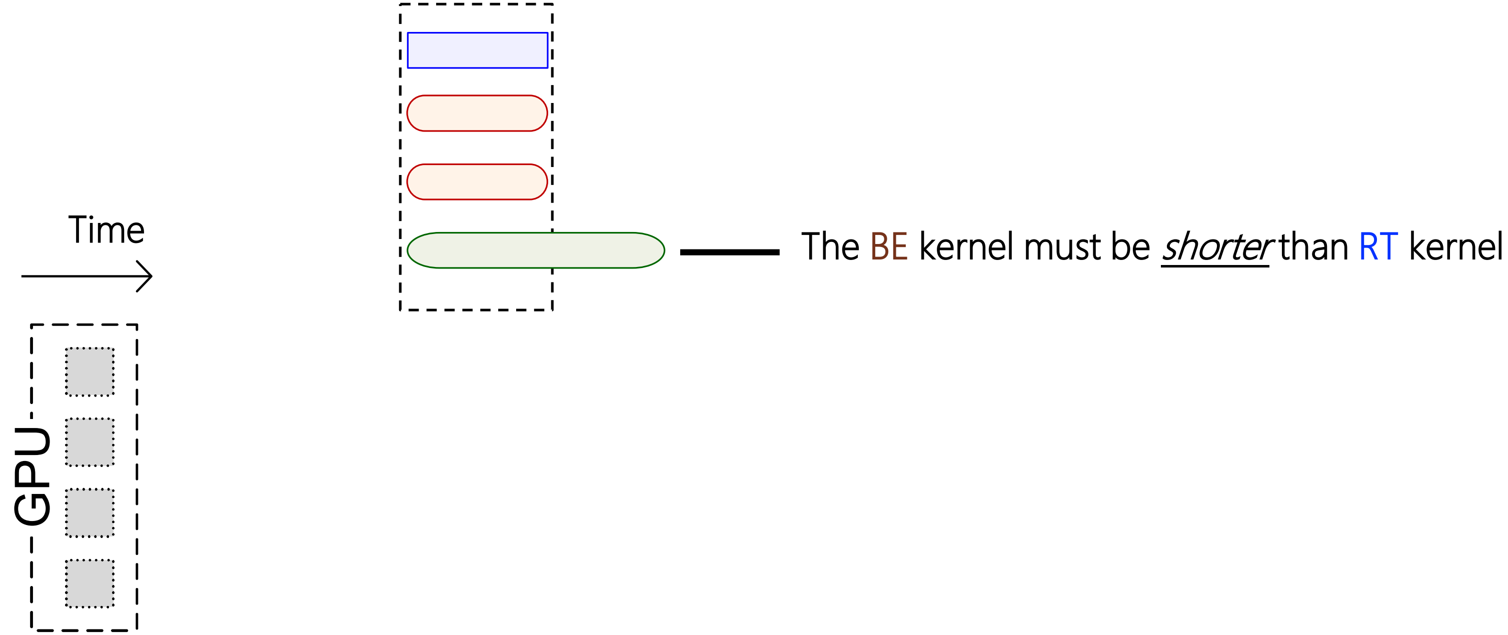
Dynamic Kernel Padding

Key Idea:
Dynamically pad RT kernels with BE kernels



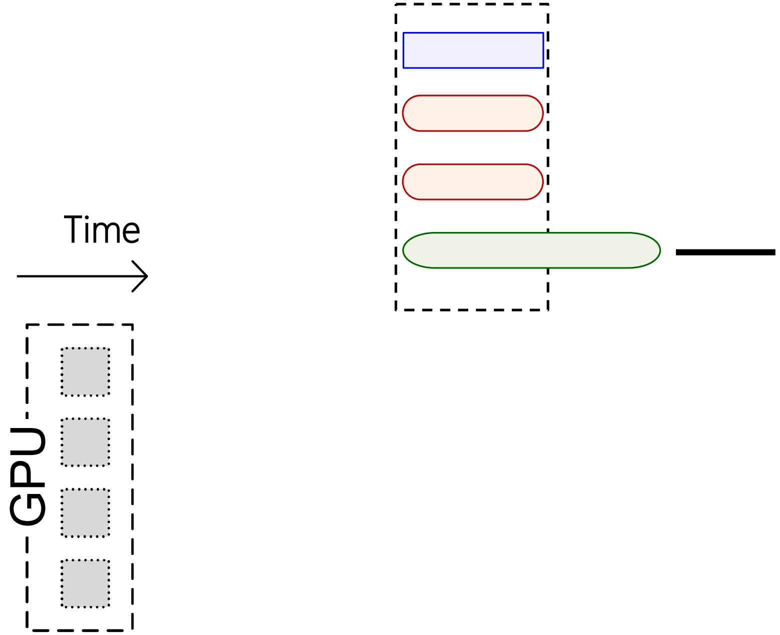
Dynamic Kernel Padding

Key Idea:
Dynamically pad RT kernels with BE kernels



Dynamic Kernel Padding

Key Idea:
Dynamically pad RT kernels with BE kernels

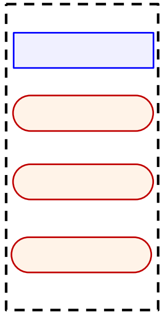


Latency Predictability

The BE kernel must be *shorter* than RT kernel

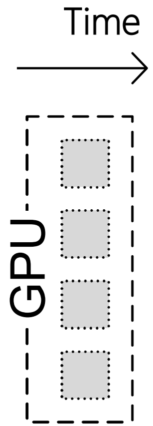
Dynamic Kernel Padding

Key Idea:
Dynamically <u>pad</u> RT kernels with BE kernels



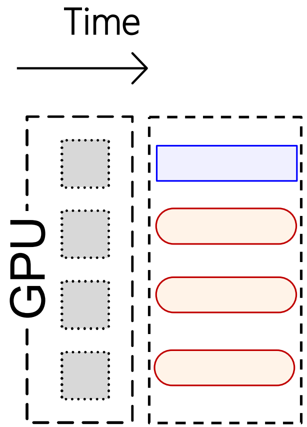
Latency Predictability

The BE kernel must be *<u>shorter</u>* than RT kernel



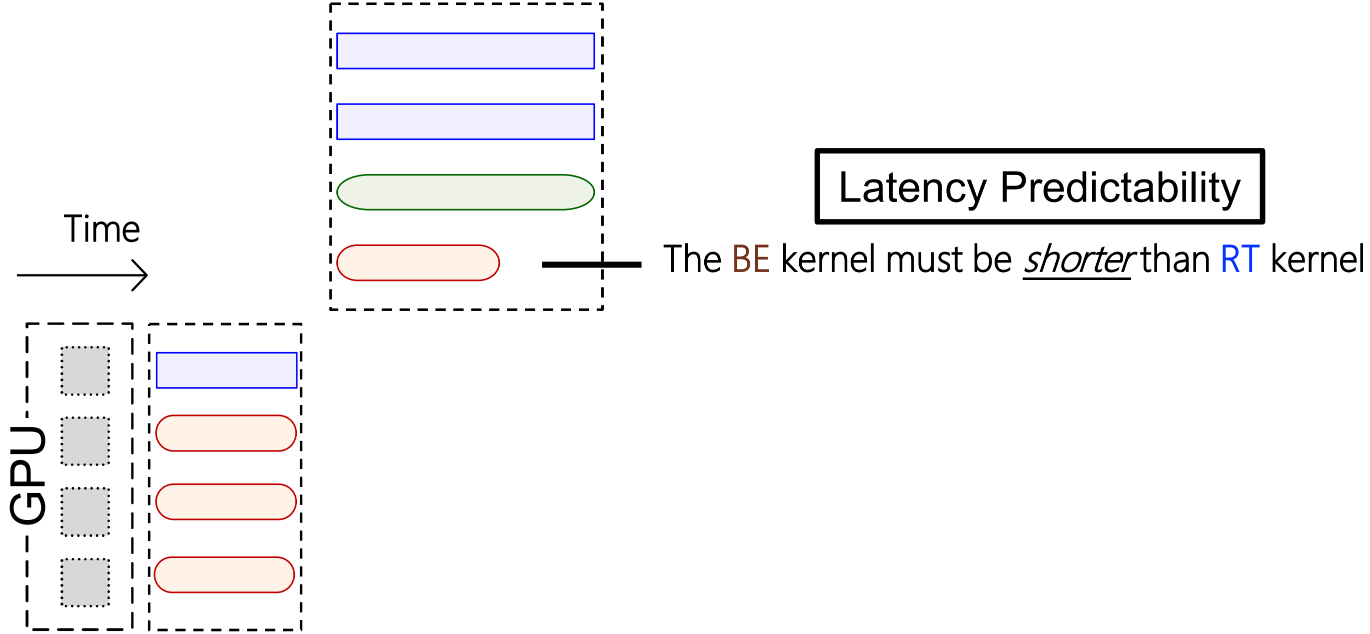
Dynamic Kernel Padding

Key Idea:
Dynamically pad RT kernels with BE kernels



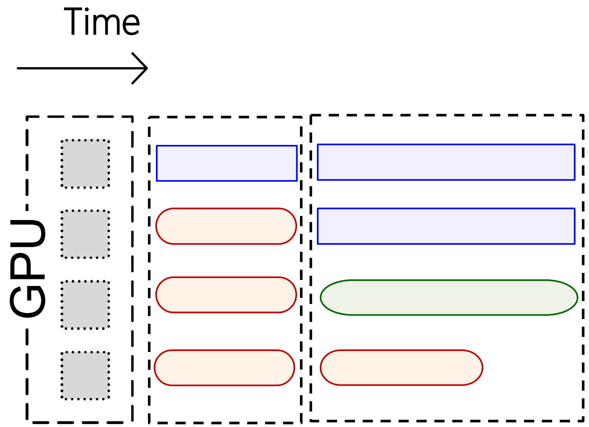
Dynamic Kernel Padding

Key Idea:
Dynamically pad RT kernels with BE kernels



Dynamic Kernel Padding

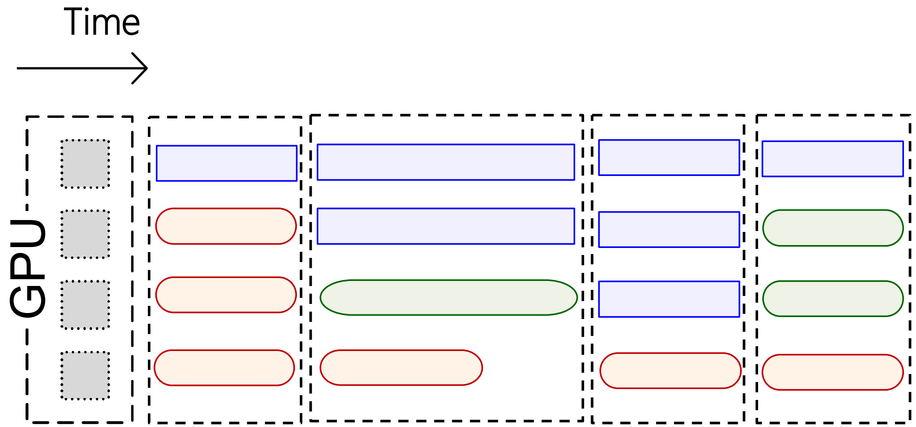
Key Idea:
Dynamically <u>pad</u> RT kernels with BE kernels



Dynamic Kernel Padding

Key Idea:

Dynamically pad RT kernels with BE kernels



Evaluation

- **Hardware Environments**
  - AMD Instinct MI50 GPU (60 CUs and 16 GB memory)
  - Intel Core i7-10700 CPU (8 cores) + 16 GB of DRAM
- **Software Environments**
  - ROCm 4.3.0
  - Apache TVM 0.8.0

Evaluation

- **DNN Inference Serving Benchmark (DISB)**
  - A new benchmark for DNN inferences in real-time scenarios
  - Five representative DNN models:
    - ResNet-152 (RNET), DenseNet-201(DNET) ,VGG-19 (VGG), Inception-v3(IN3), DistilBert(BERT)
  - Five workloads

Issue **RT** inference requests (fixed frequency)

DISB	**A**	**B**	**C**	**D**	**E**
Num. of RT clients	1/VGG	1/VGG	1/VGG	5/ALL	5/ALL
Frequency (reqs/s)	100 [U]	220 [U]	100 [U]	20 [U]	20 [P]
Num. of BE clients	1/RNET	1/RNET	5/ALL	5/ALL	5/ALL

saturate our testbed

Issue **BE** inference requests (close-loop)

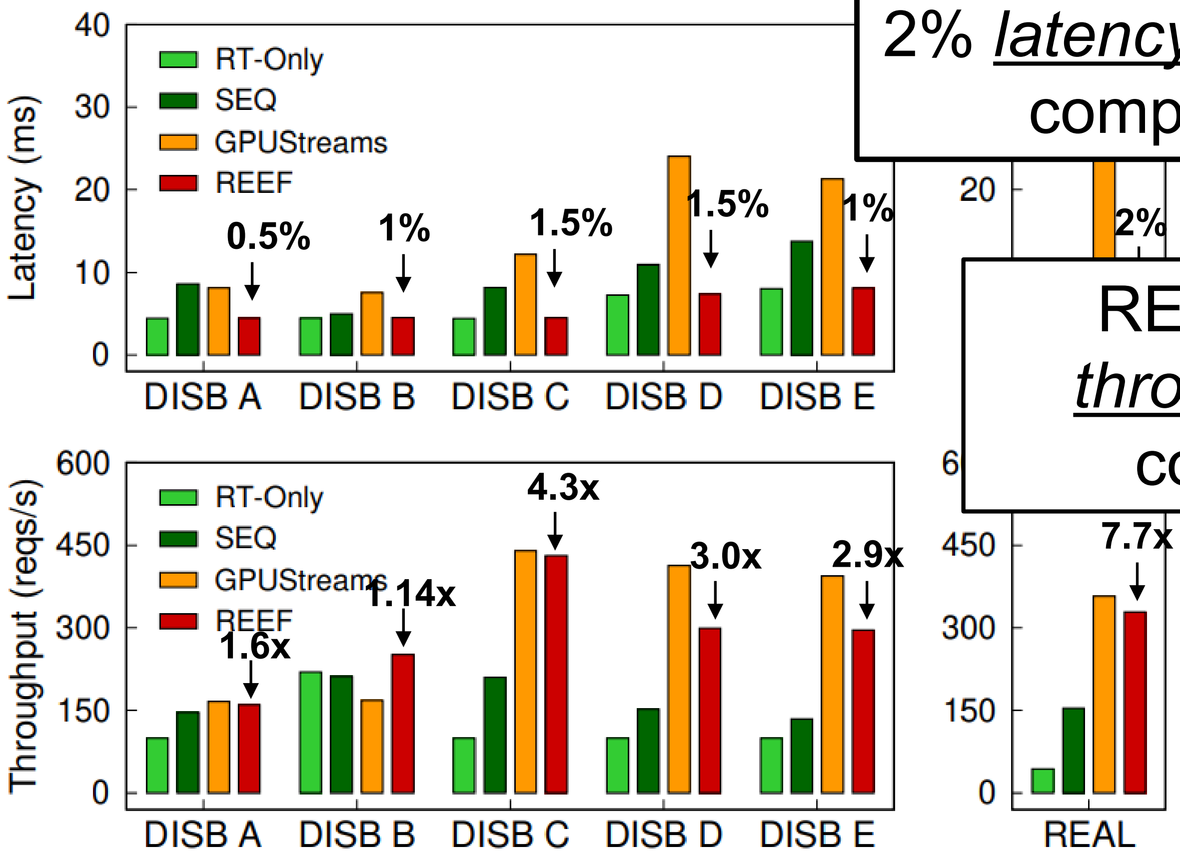
Evaluation

- **DNN Inference Serving Benchmark (DISB)**
  - A new benchmark for DNN inferences in real-time scenarios
  - Five representative DNN models:
    - ResNet-152 (RNET), DenseNet-201(DNET) ,VGG-19 (VGG), Inception-v3(IN3), DistilBert(BERT)
  - Five workloads
- **Real-world Trace**
  - From an open autonomous driving platform (i.e., ApolloAuto)

Evaluation

- **Comparing targets**
  - **RT-Only**: dedicate the GPU for RT tasks
  - **SEQ**: sequentially execute tasks without preemption
  - **GPUStreams**: execute RT/BE tasks concurrently in multiple GPU streams

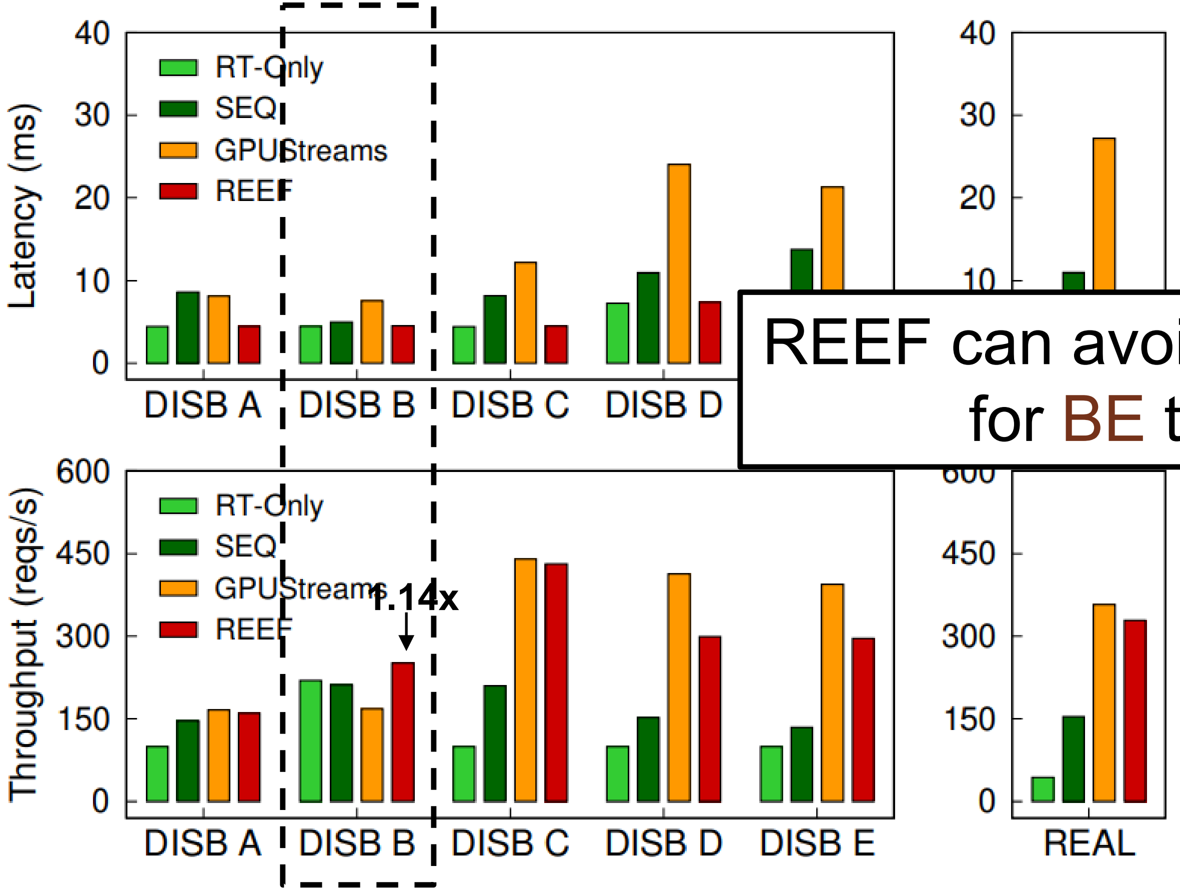
Evaluation



REEF only incurs at most 2% *latency overhead* for RT tasks compared with RT-Only

REEF can improve *overall throughput* by 1.14× ~ 7.7× compared with RT-Only

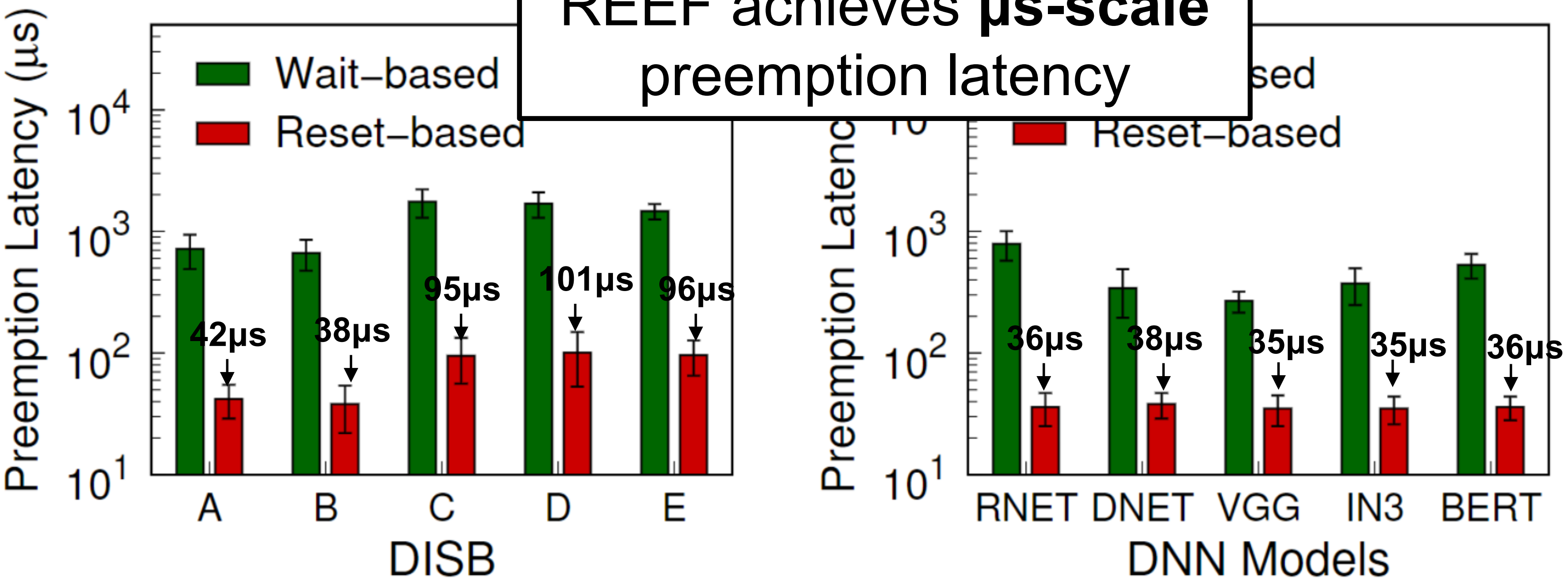
Evaluation



REEF can avoid starvation for BE tasks

Evaluation

REEF achieves **µs-scale** preemption latency



Conclusion

- **REEF: a GPU-accelerated DNN inference serving system**
  - Achieve both low-latency (2% latency overhead for real-time tasks) and work-conserving (1.14x – 7.7x throughput improvement)
  - Reset-based preemption: μs-scale preemption based on ***idempotence***
  - Dynamic kernel padding: controlled concurrent execution based on ***latency predictability***

Thanks & QA