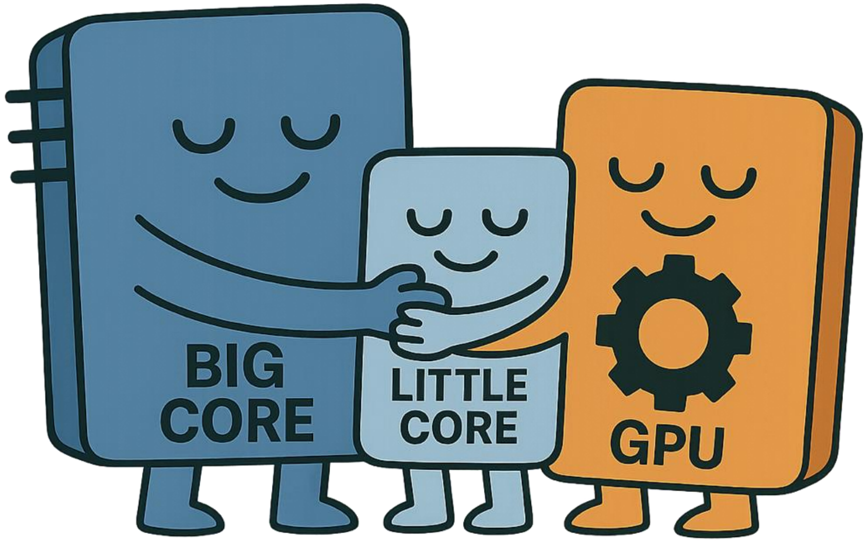
*2025 IEEE International Symposium on Workload Characterization | October 12-14, 2025 | Irvine, USA*

# BetterTogether




# An Interference-Aware Framework for Fine-grained Software Pipelining on Heterogeneous SoCs

***Yanwen Xu***, *Rithik Sharma, Zheyuan Chen, Shaan Mistry, Tyler Sorensen*

UC SANTA CRUZ | Baskin Engineering

Microsoft

** The views and opinions expressed in this presentation are solely those of the authors and do not necessarily reflect the official policy or position of Microsoft.*

# Motivation: Accelerating Computation at **Edge**

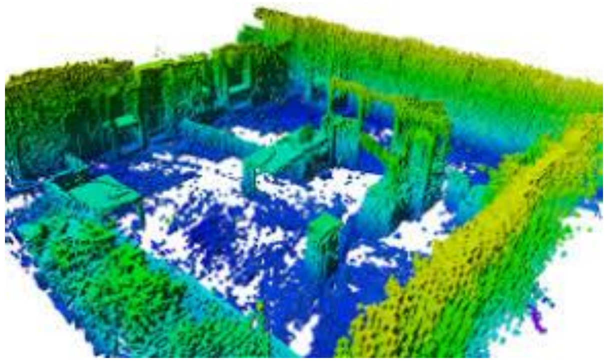
- Lower latency
- Energy efficiency
- Privacy benefits

NVIDIA Jetson Thor


Google Pixel

NVIDIA Jetson Orin Nano

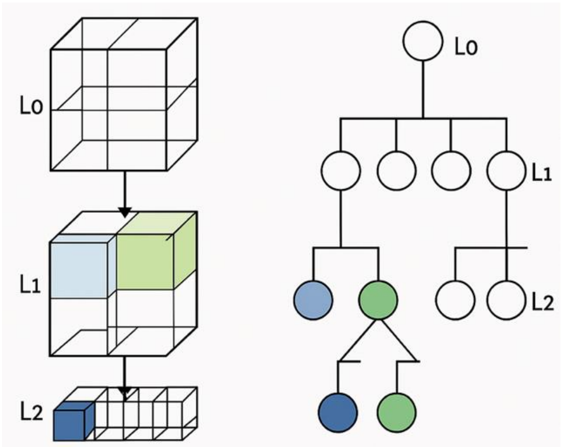
# Motivation: Accelerating Computation at Edge



**Input:** 3D point cloud

*e.g., Classic Algorithm from NVIDIA*

**3D Octree Reconstruction**



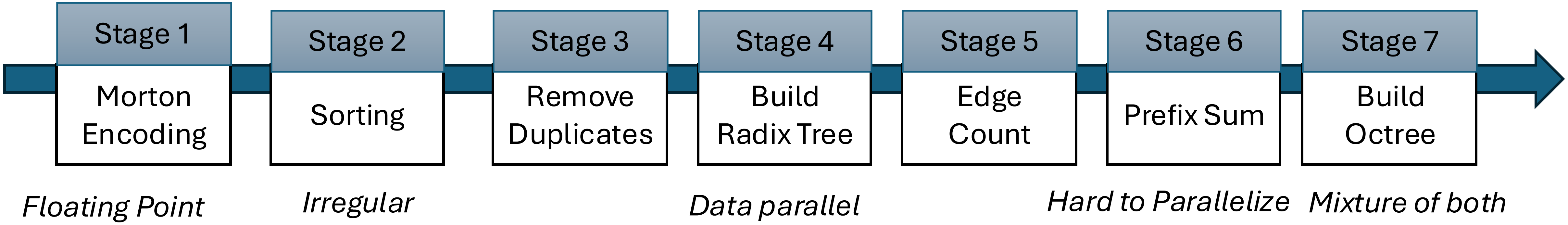

**Output:** octree data structure

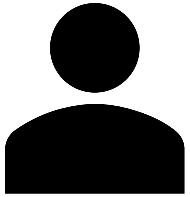
*Decomposed into*

| Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 |
|---|---|---|---|---|---|---|
| Morton Encoding | Sorting | Remove Duplicates | Build Radix Tree | Edge Count | Prefix Sum | Build Octree |

*<u>Stages depends on data from previous stages</u>*

# Motivation: Accelerating Computation at Edge



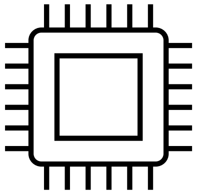
Developer

```cpp
// -----------------------------------------------------------------------
// Encode
// -----------------------------------------------------------------------

[[nodiscard]] constexpr uint32_t morton3D_SplitBy3bits(const uint32_t a) {
  auto x = static_cast<uint32_t>(a) & 0x000003ff;
  x = (x | x << 16) & 0x30000ff;
  x = (x | x << 8) & 0x0300f00f;
  x = (x | x << 4) & 0x30c30c3;
  x = (x | x << 2) & 0x9249249;
  return x;
}

[[nodiscard]] constexpr uint32_t m3D_e_magicbits(const uint32_t x,
                                                 const uint32_t y,
                                                 const uint32_t z) {
  return morton3D_SplitBy3bits(x) | (morton3D_SplitBy3bits(y) << 1) |
         (morton3D_SplitBy3bits(z) << 2);
}

[[nodiscard]] constexpr uint32_t xyz_to_morton32(const glm::vec4 &xyz,
                                                 const float min_coord,
                                                 const float range) {
  constexpr auto bit_scale = 1024;
  const auto i = static_cast<uint32_t>((xyz.x - min_coord) / range * bit_scale);
  const auto j = static_cast<uint32_t>((xyz.y - min_coord) / range * bit_scale);
  const auto k = static_cast<uint32_t>((xyz.z - min_coord) / range * bit_scale);
  return m3D_e_magicbits(i, j, k);
}
```



CPU

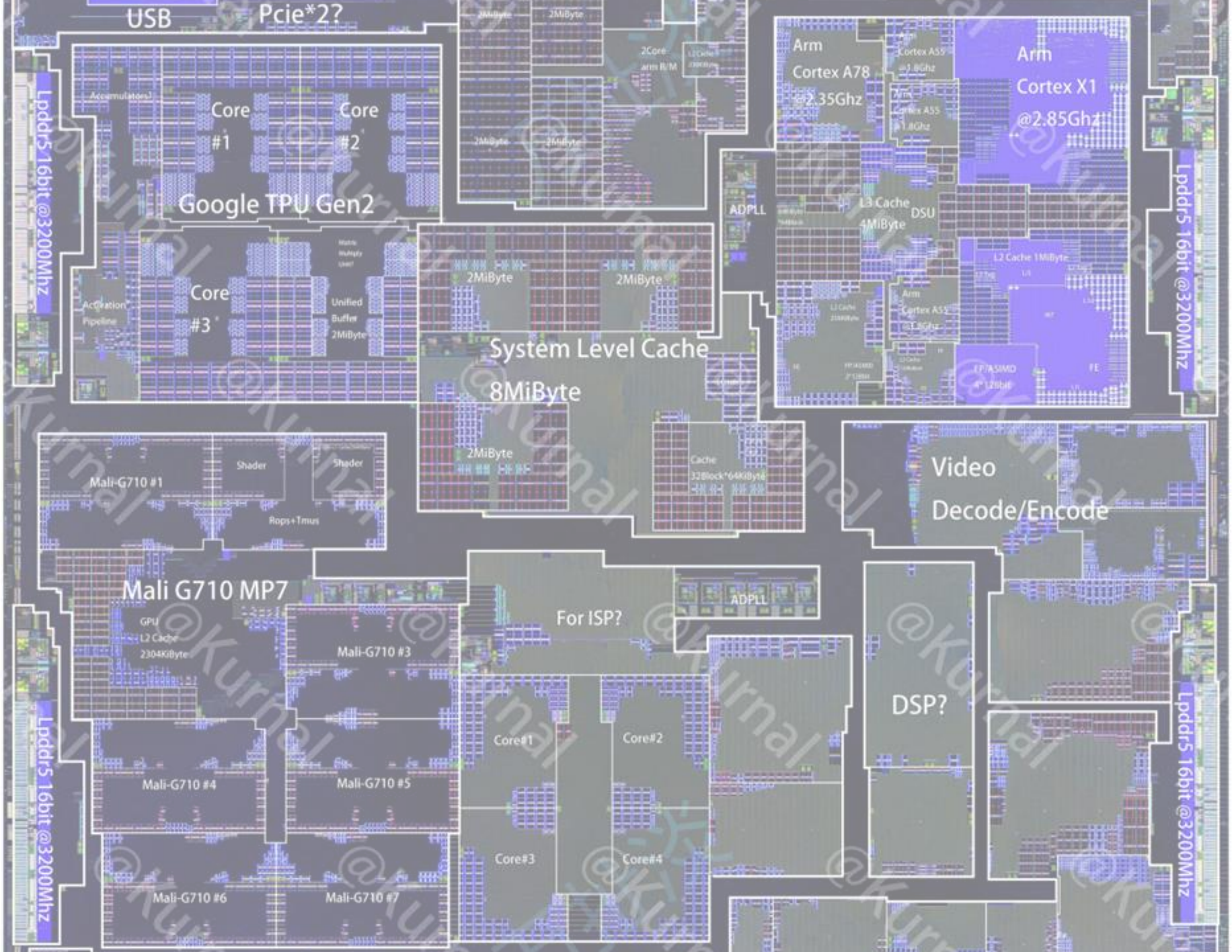
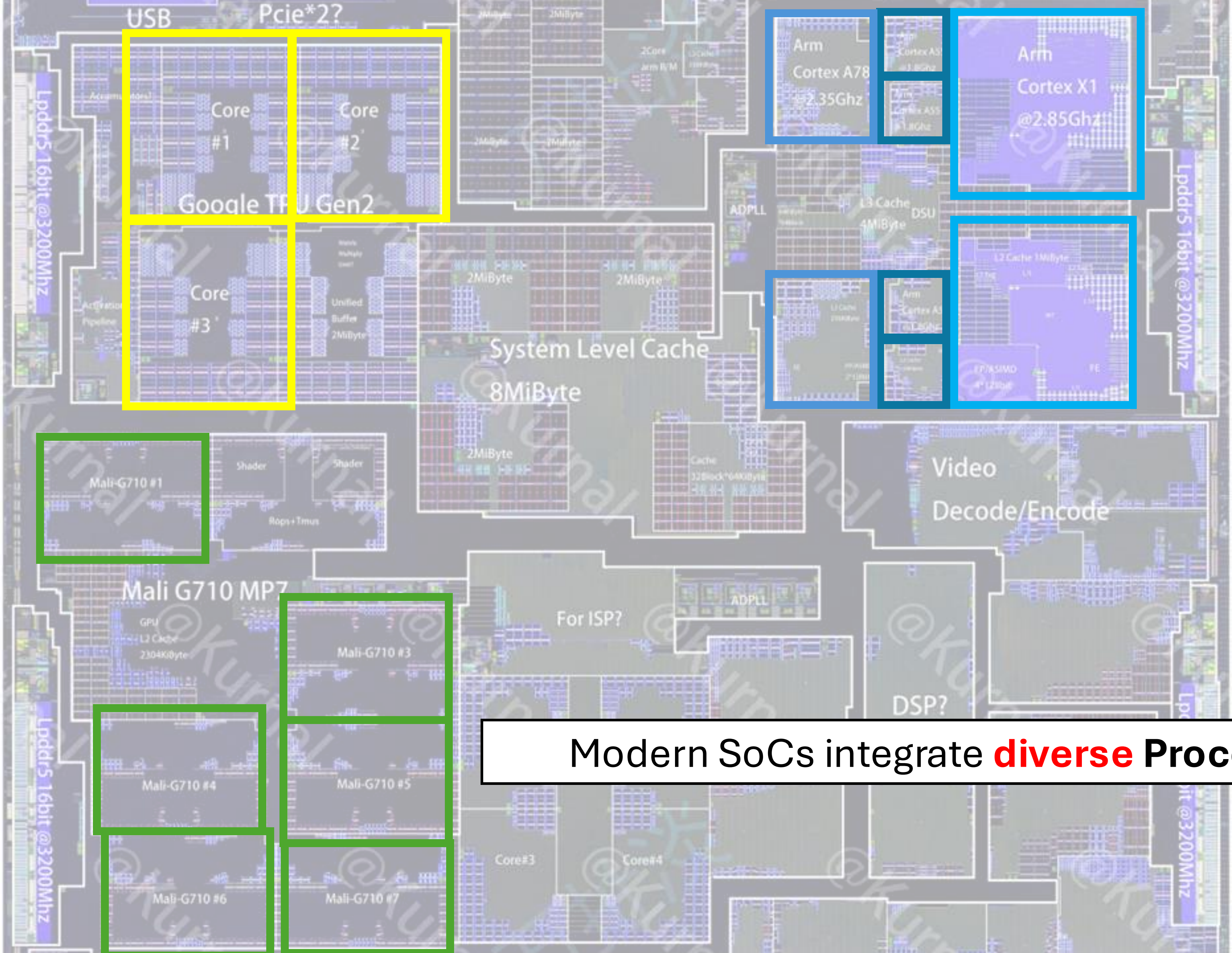



*Image credit: Kurnal on X*

Google Tensor G2

*Image credit: Kurnal on X*




| Google Tensor G2 |
| --- |
| Mali-G710 MP7 |
| 3 TPU |
| 2 Cortex-X1 (P-cores) |
| 2 Cortex-A78 (medium) |
| 4 Cortex-A55 (E-cores) |

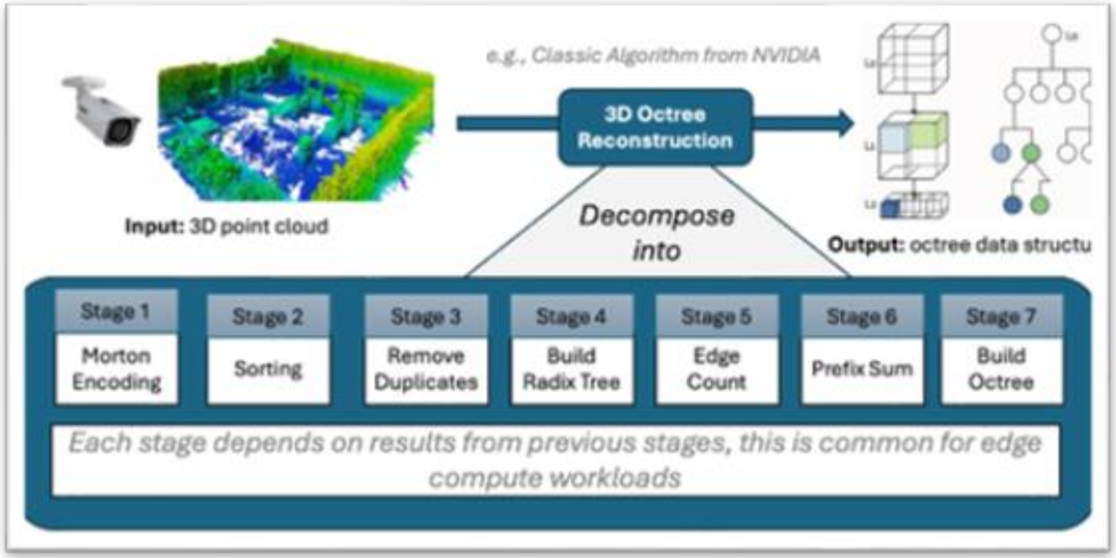
Modern SoCs integrate **diverse** **Processing Units** (**PU**)

# PU Profiling

Mali-G710 MP7
2 Cortex-X1
2 Cortex-A78
4 Cortex-A55

*All Available PUs on Pixel*

- Ran each stage on each available PUs



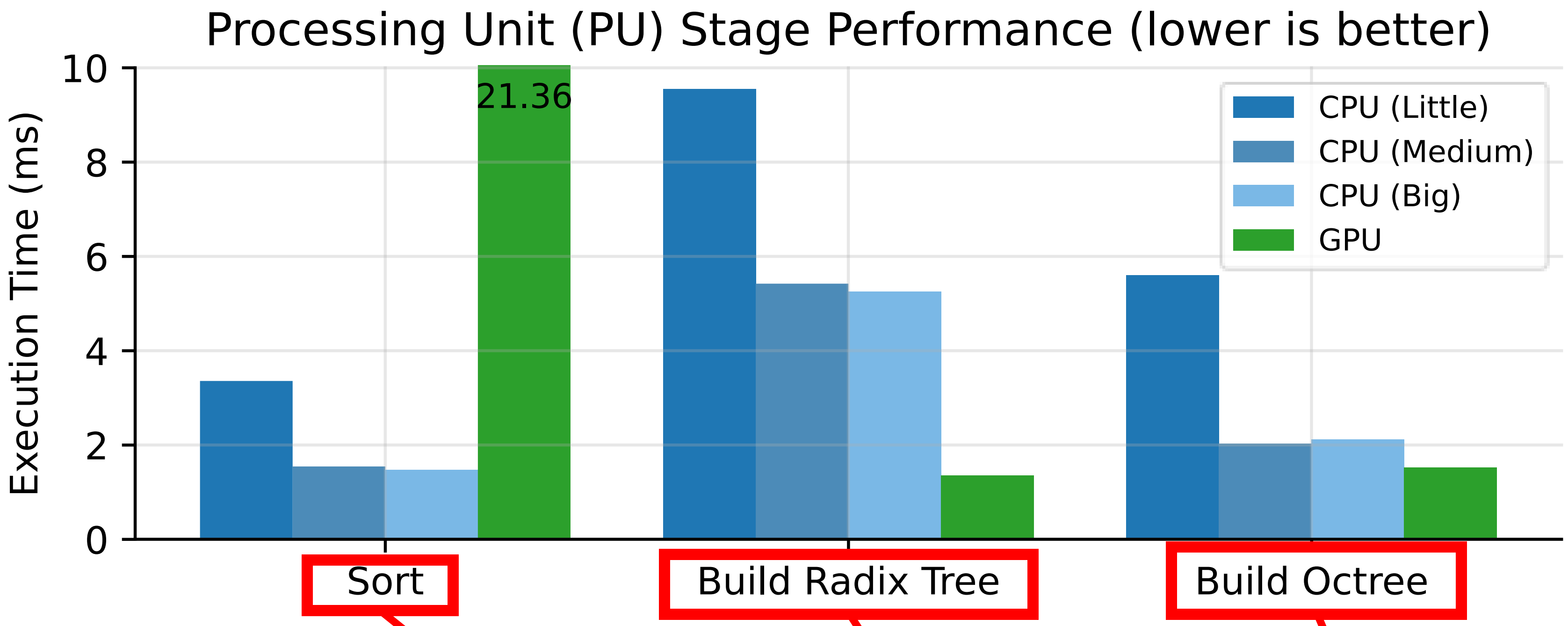

*All Stages*

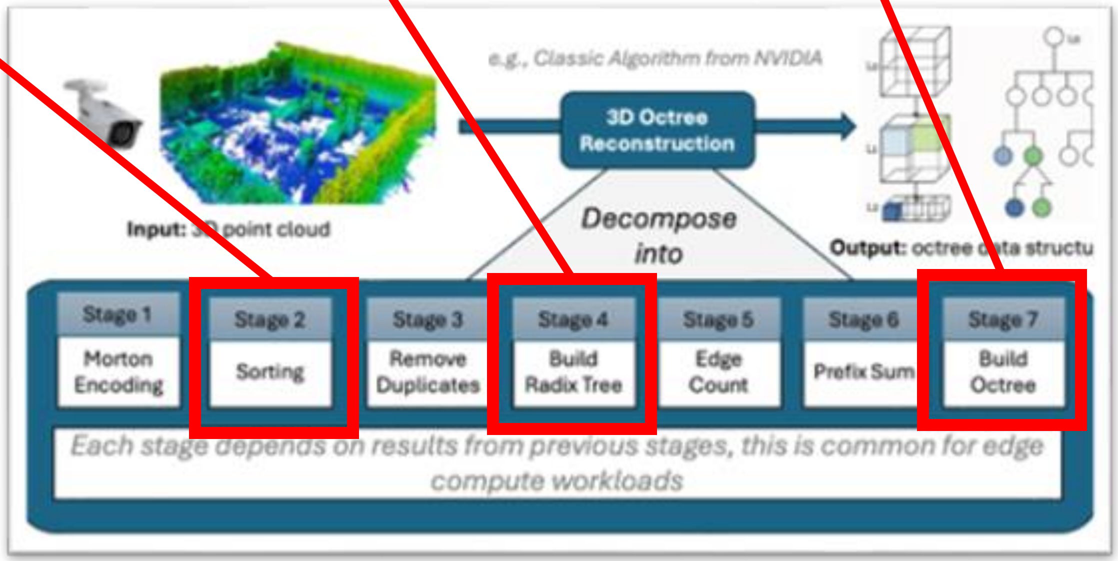
# PU Profiling

Mali-G710 MP7
2 Cortex-X1
2 Cortex-A78
4 Cortex-A55

*All Available PUs on Pixel*

- Ran each stage on each available PUs



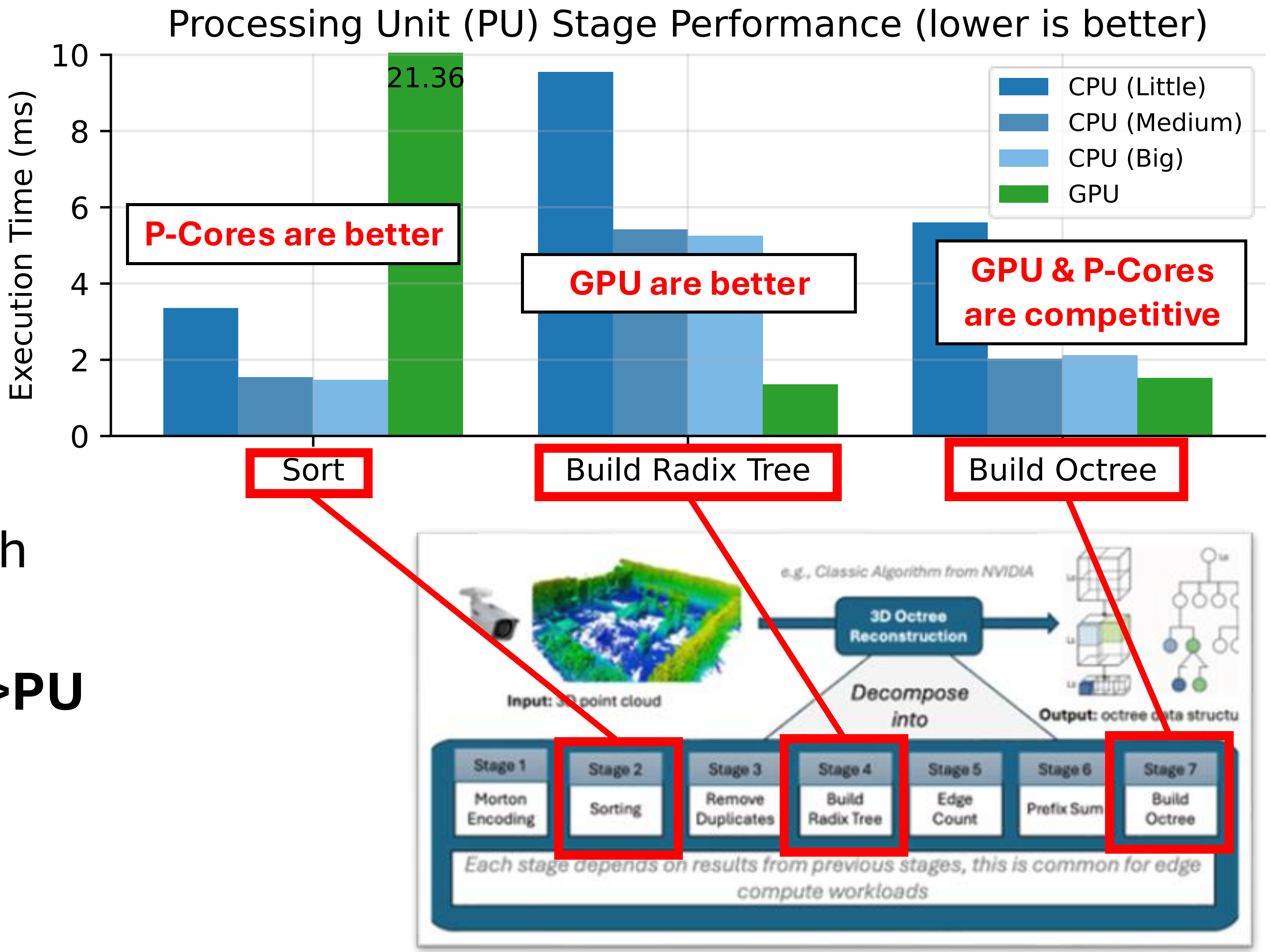




*All Stages*

# PU Profiling

Mali-G710 MP7
2 Cortex-X1
2 Cortex-A78
4 Cortex-A55

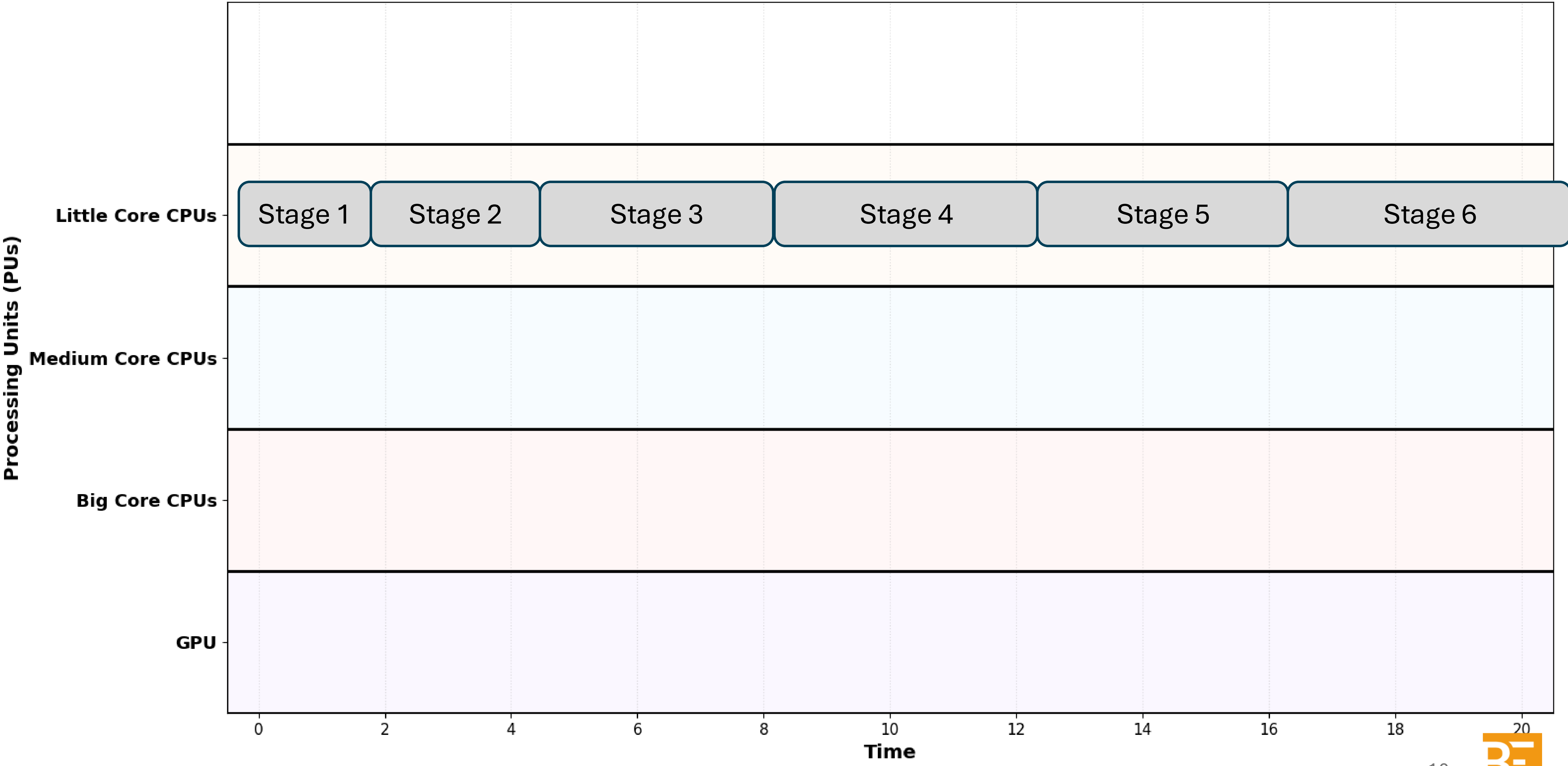
*All Available PUs on Pixel*



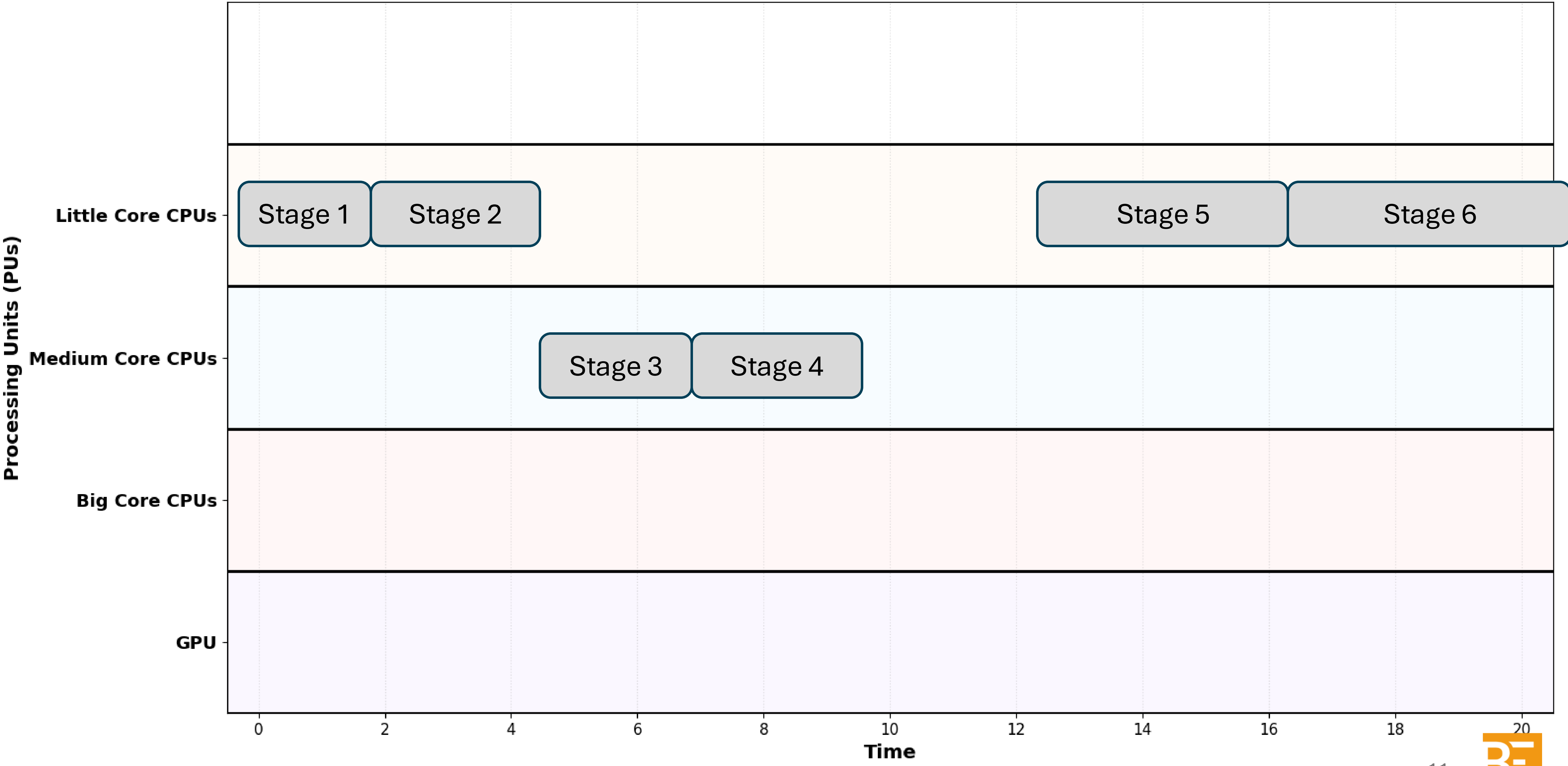

*All Stages*

- Ran each stage on each available PUs
- Found optimal **Stage->PU** mapping

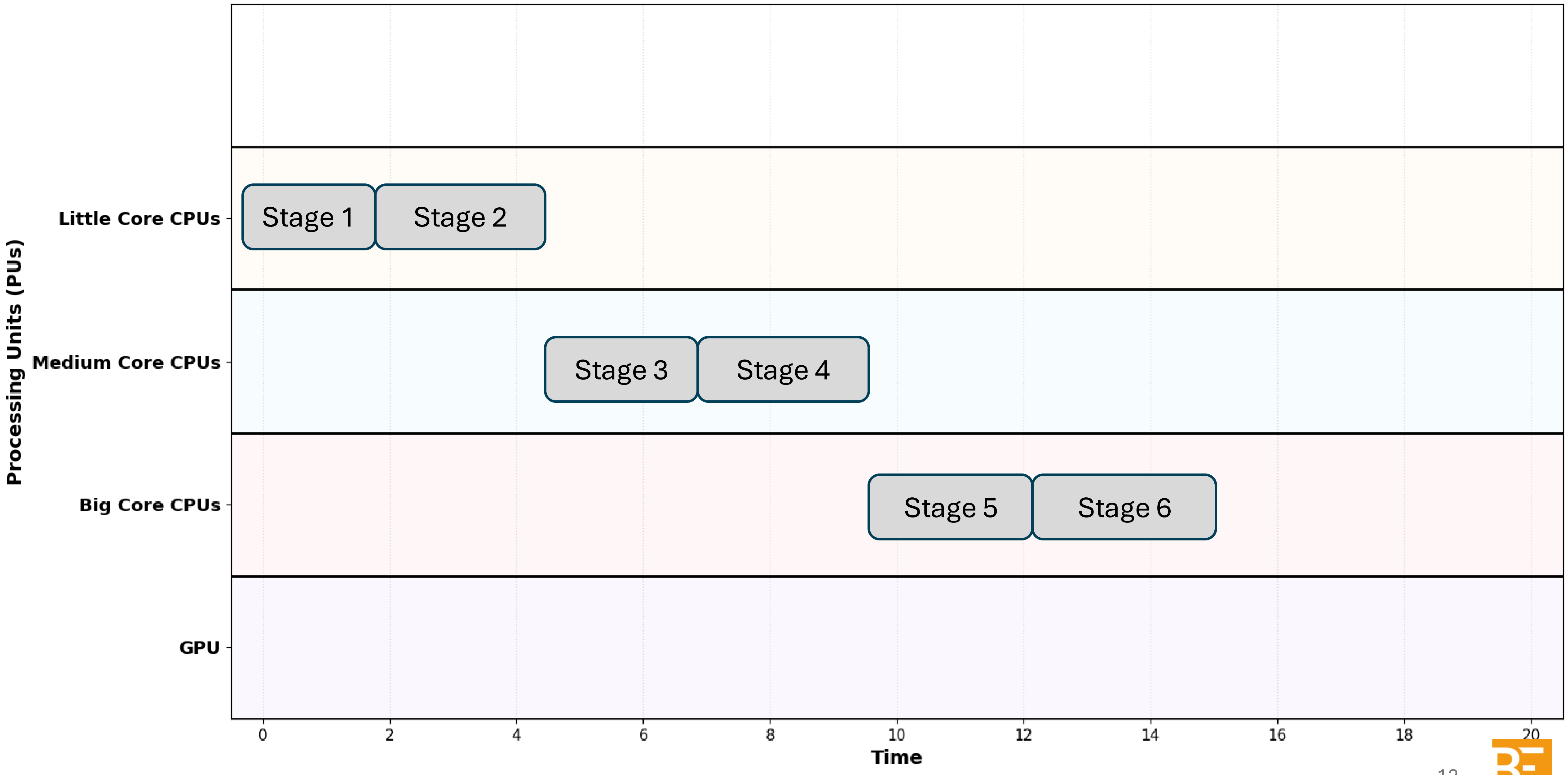
# Efficient Pipelined Scheduling



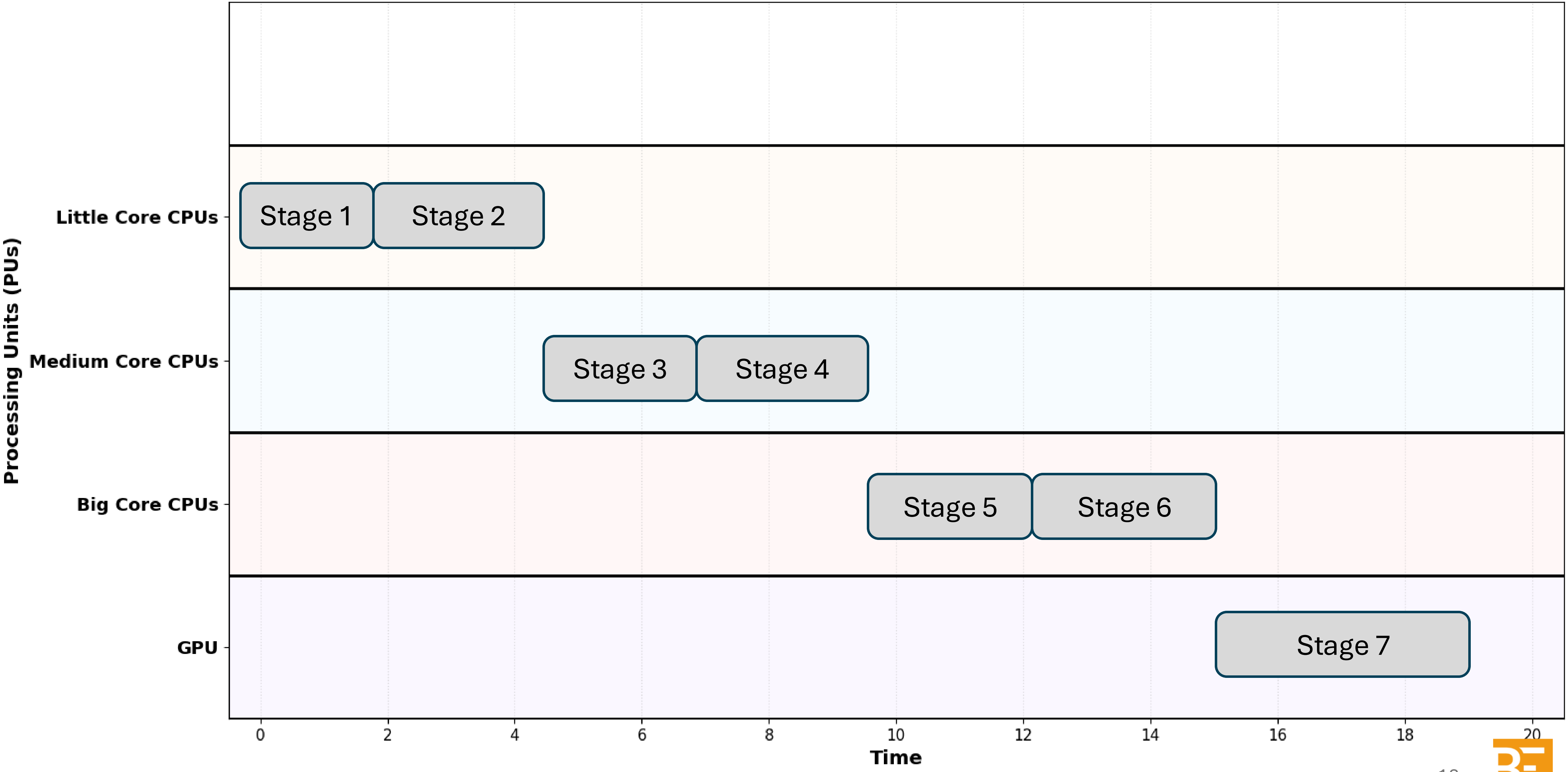
# Efficient Pipelined Scheduling



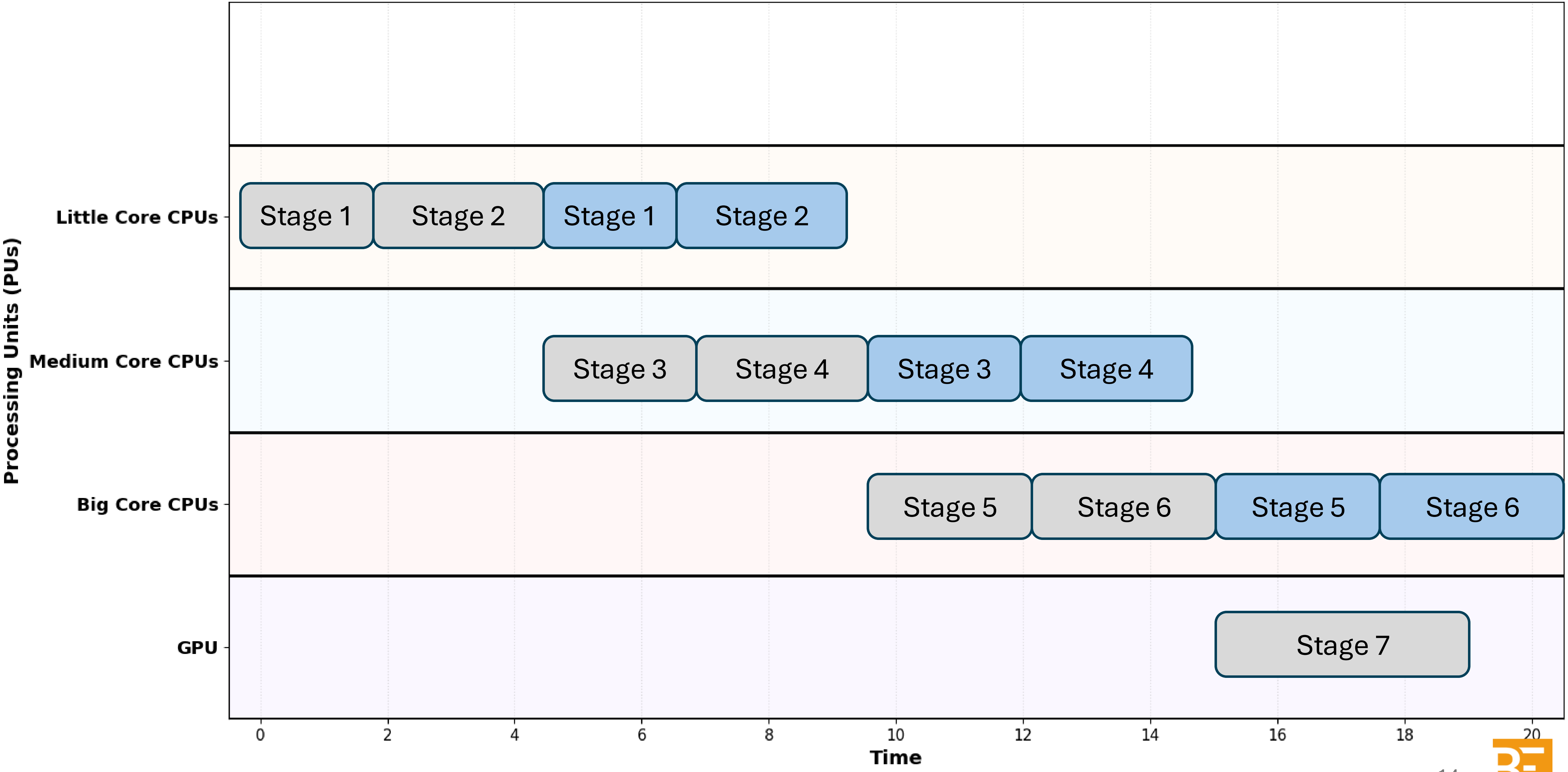
# Efficient Pipelined Scheduling



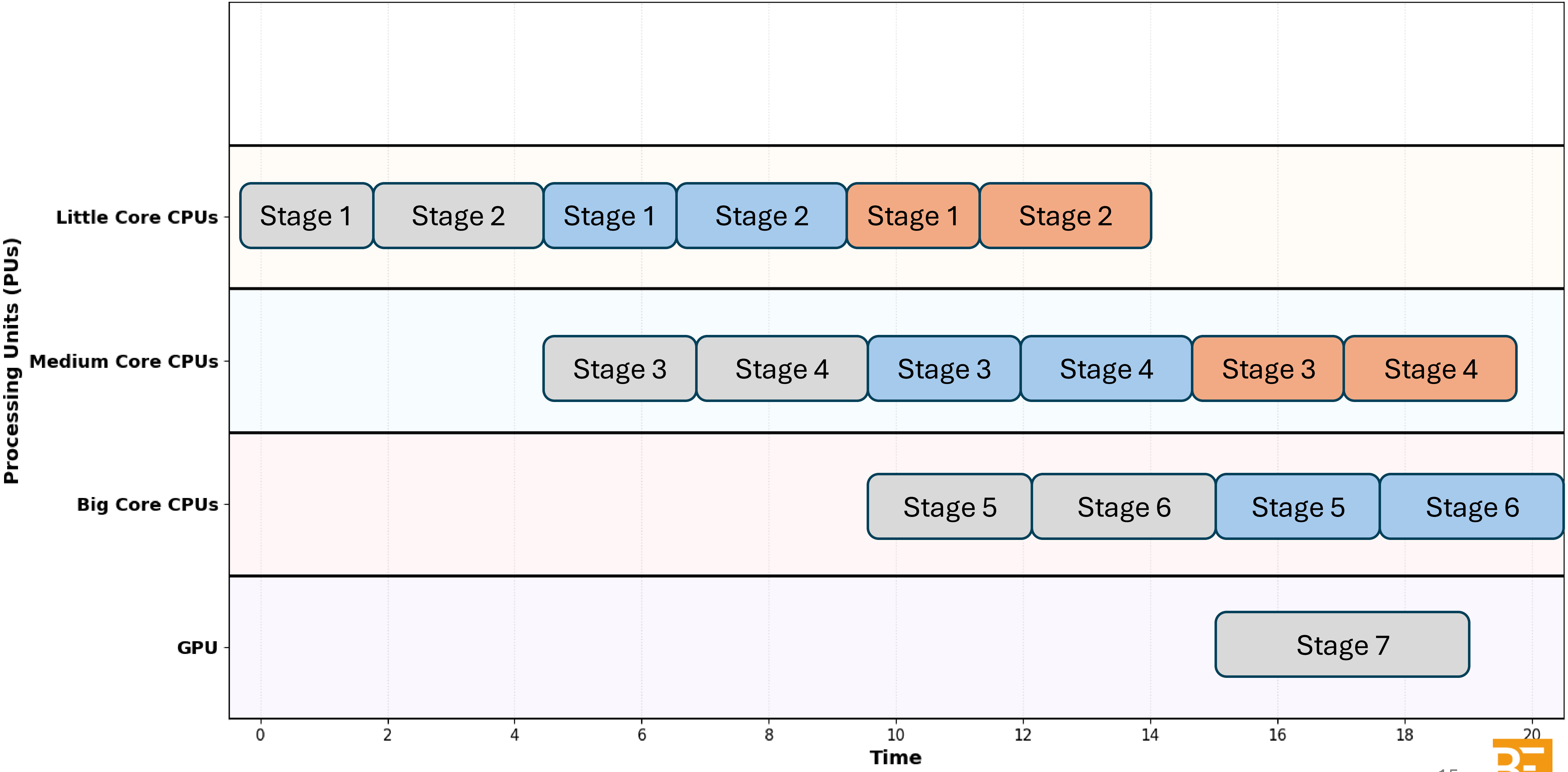
# Efficient Pipelined Scheduling



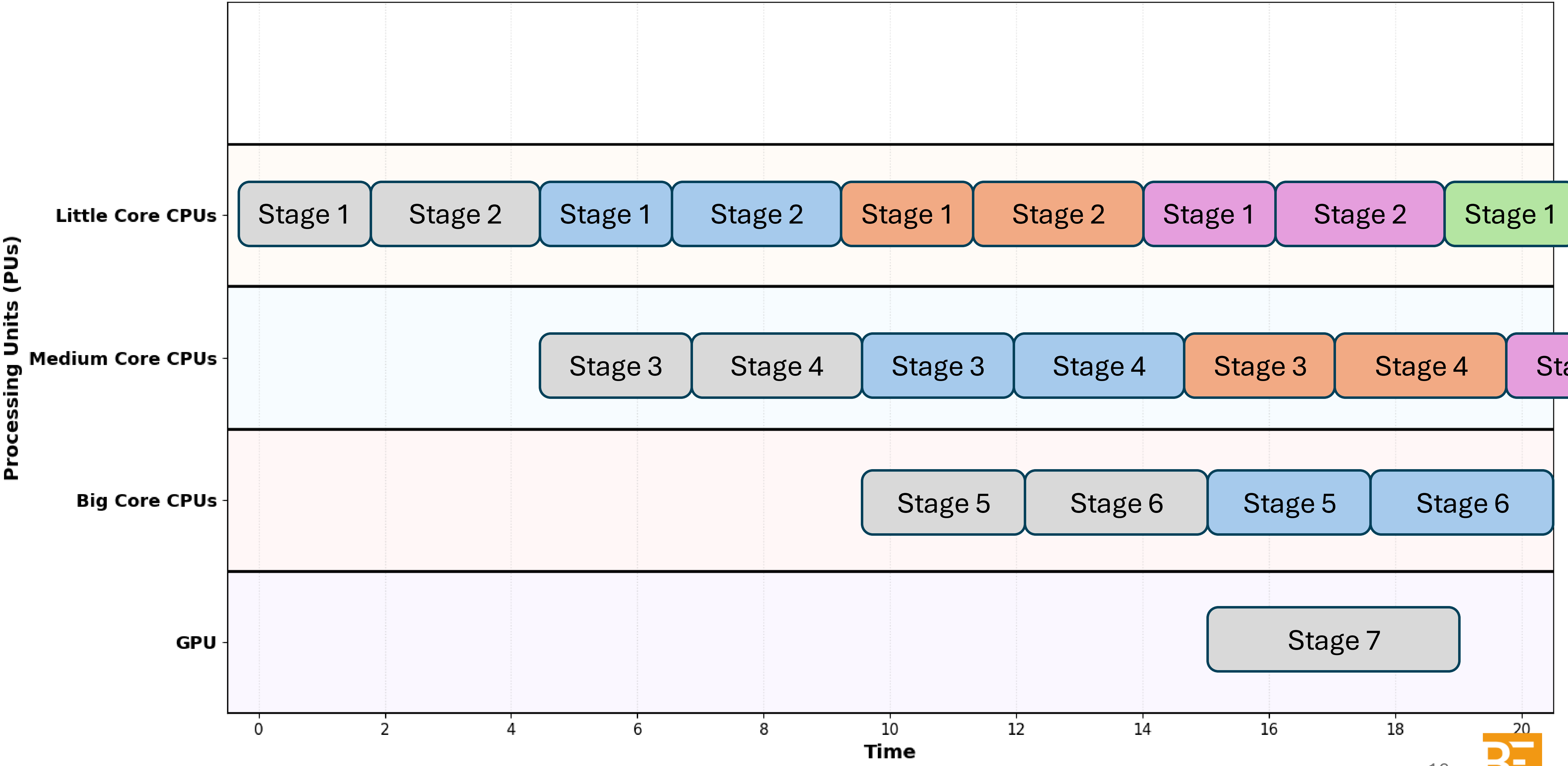
# Efficient Pipelined Scheduling



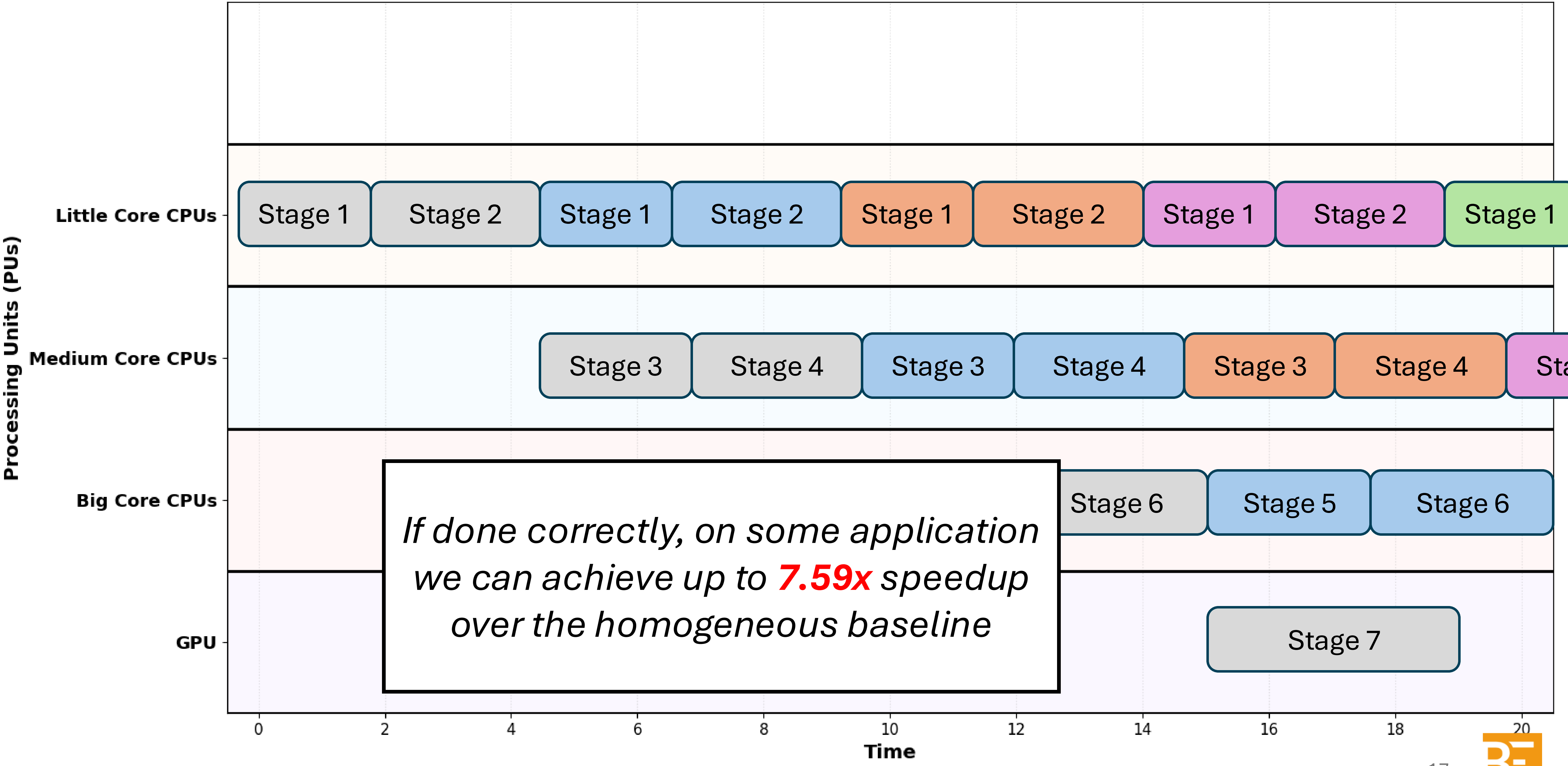
# Efficient Pipelined Scheduling



# Efficient Pipelined Scheduling

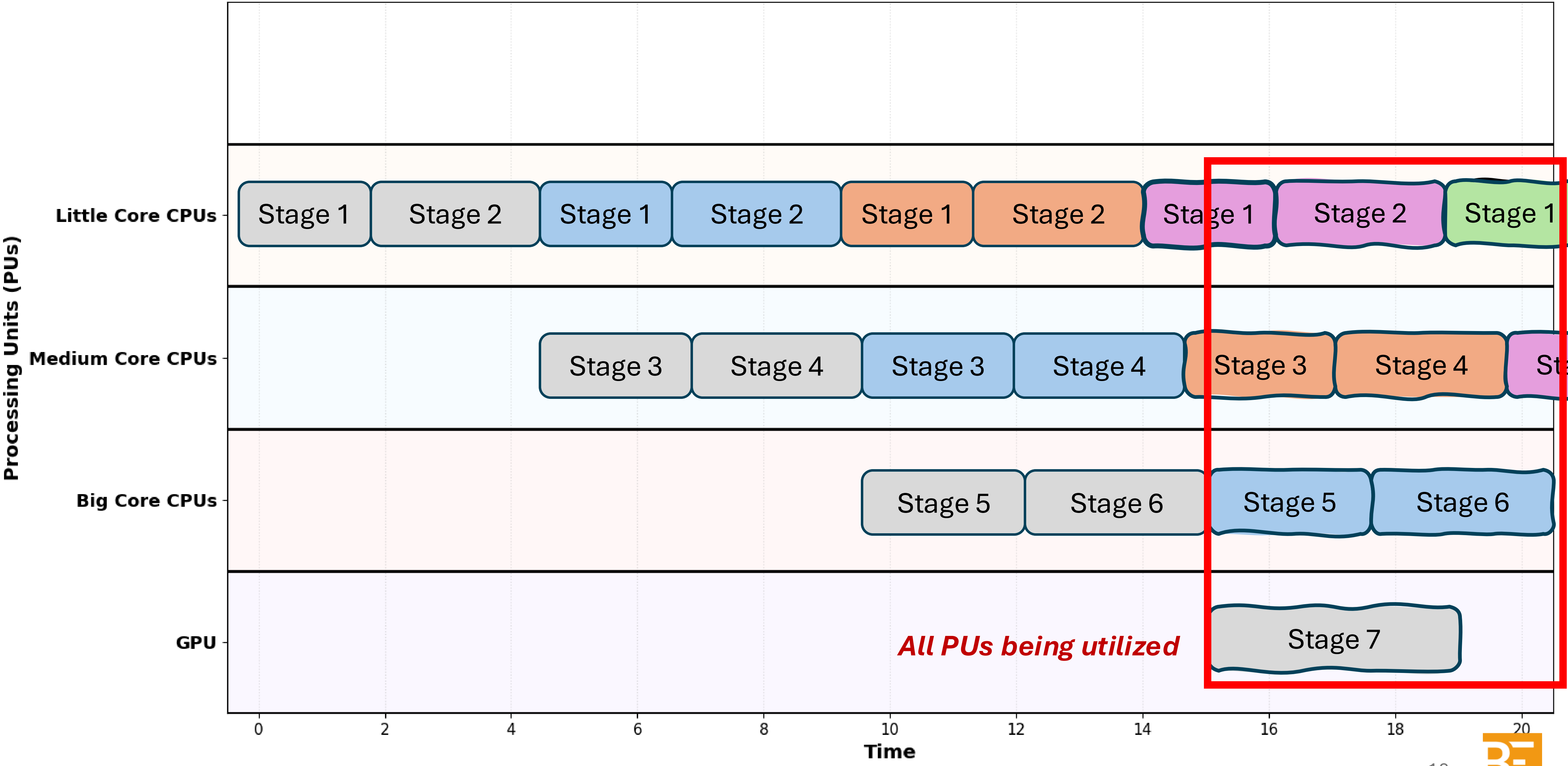


# Efficient Pipelined Scheduling

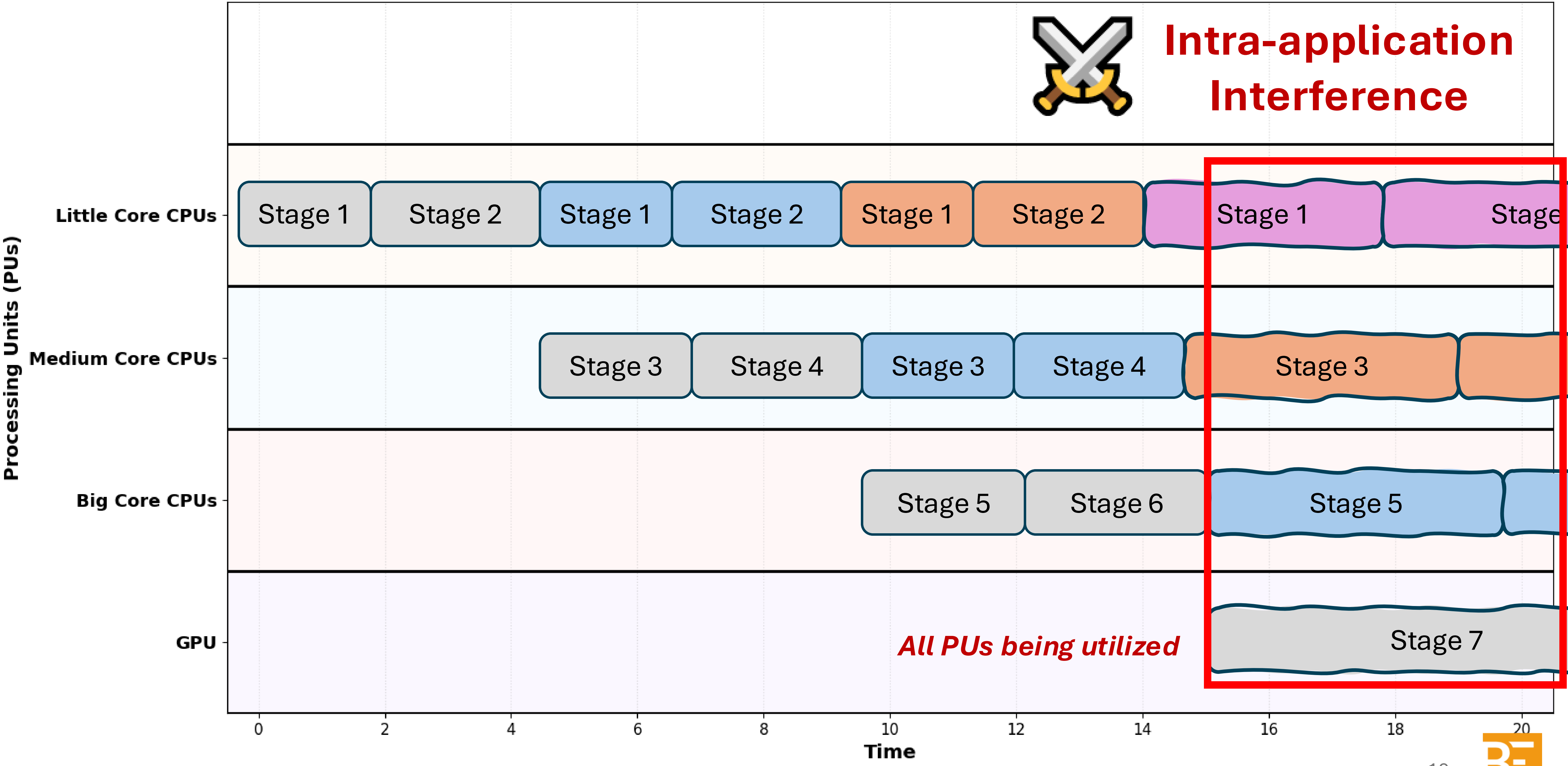



*If done correctly, on some application we can achieve up to* ***7.59x*** *speedup over the homogeneous baseline*

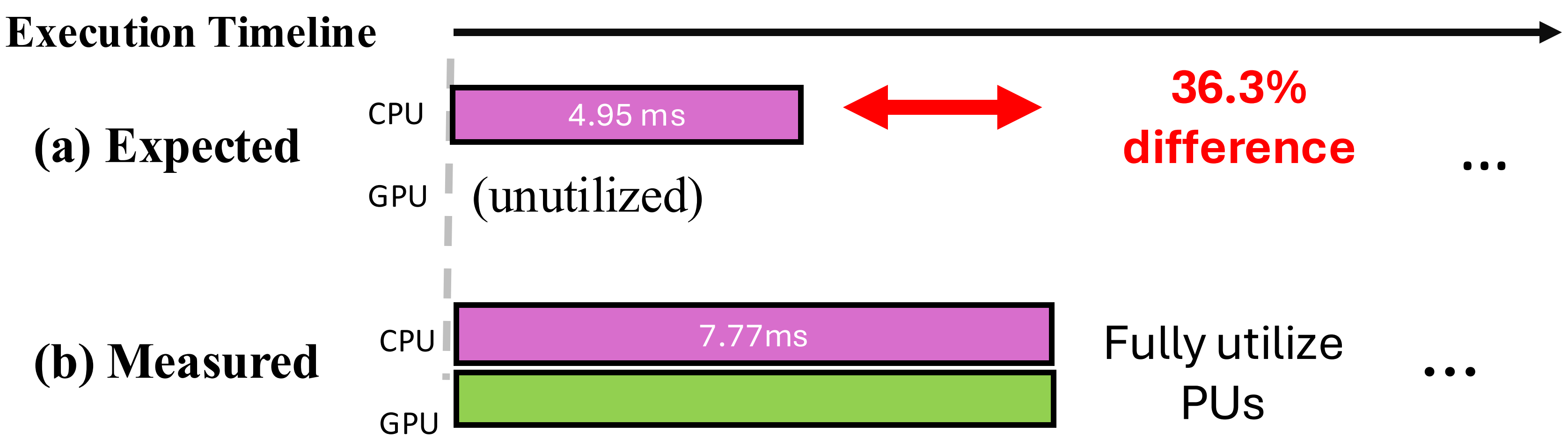
# Efficient Pipelined Scheduling



# Efficient Pipelined Scheduling



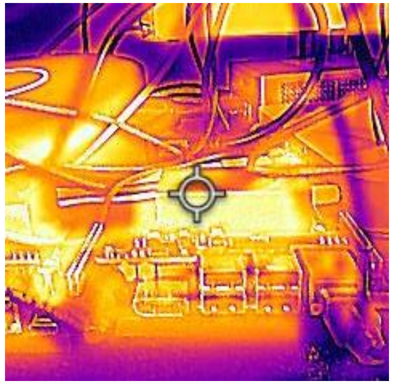
- Mobile systems are prone to **intra-application interferences**
  - E.g., we think **4.95** ms, but real measurement was **7.77** ms, ~**57% slower**

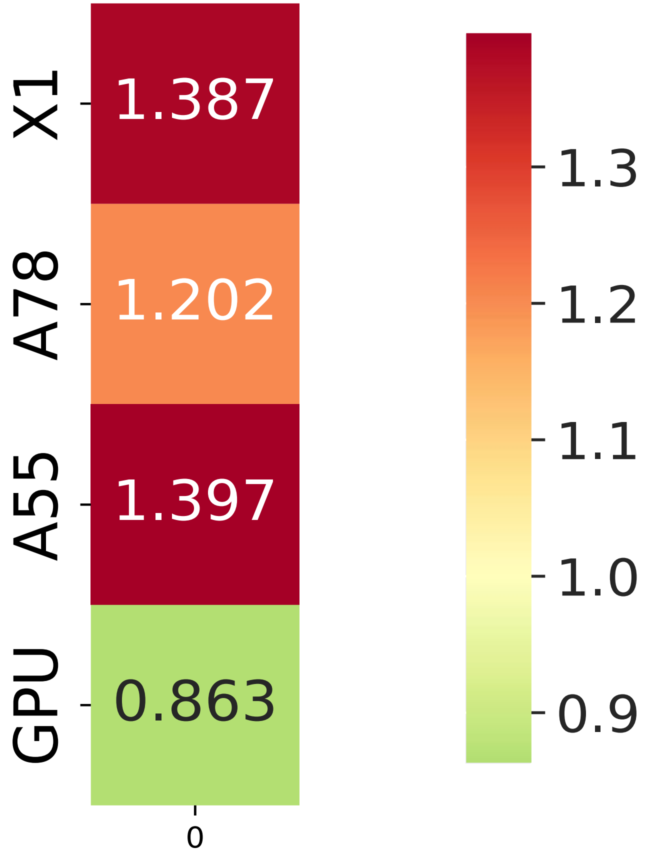


# **Challenge I**: Interference

- When PUs fully utilized

- **Slowdowns** and **Speedups*** due to
  - *Resource contention*
  - *Dynamic voltage and frequency scaling (DVFS)*
  - *Thermal throttling*
  - *Power management*
  - …






**Red = Slower down**
**Green = Speedup**

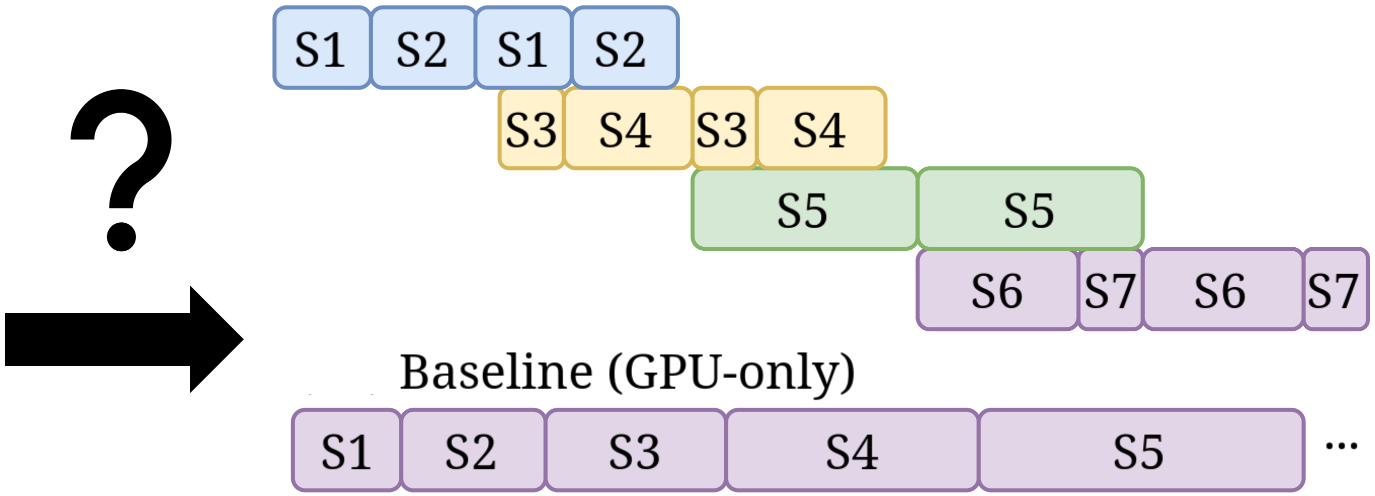
*We consulted with engineers from a major mobile vendor, whose insights were consistent with our observations*

# Finding the Optimal Schedule is Hard

- **<u>Schedule</u> = mapping** from program **stages** to appropriate **PU**

| Application Stages | |
|---|---|
| **S1** | *Morton Encoding* |
| **S2** | *Sort* |
| **S3** | *Build Radix Tree* |
| **...** | |
| **S7** | *Link Nodes* |

| Affinity Map |
|---|
| 4x Little Cores **(PU1)** |
| 2x Medium Cores **(PU2)** |
| 2x Big Cores **(PU3)** |
| 1x iGPU **(PU4)** |



# **Challenge II**: Portability

- Large design exploration space
  - e.g., 9 stage AlexNet $5^9 \approx$ **1.9 M** potential schedules
  - **~37 years** for Google Pixel 7a
- **Schedules** are not portable
  - Optimal schedule on Pixel does not work on NVIDIA Jetson

The scheduling framework need to be **portable** and **flexible**, and **suitable** for rapid development

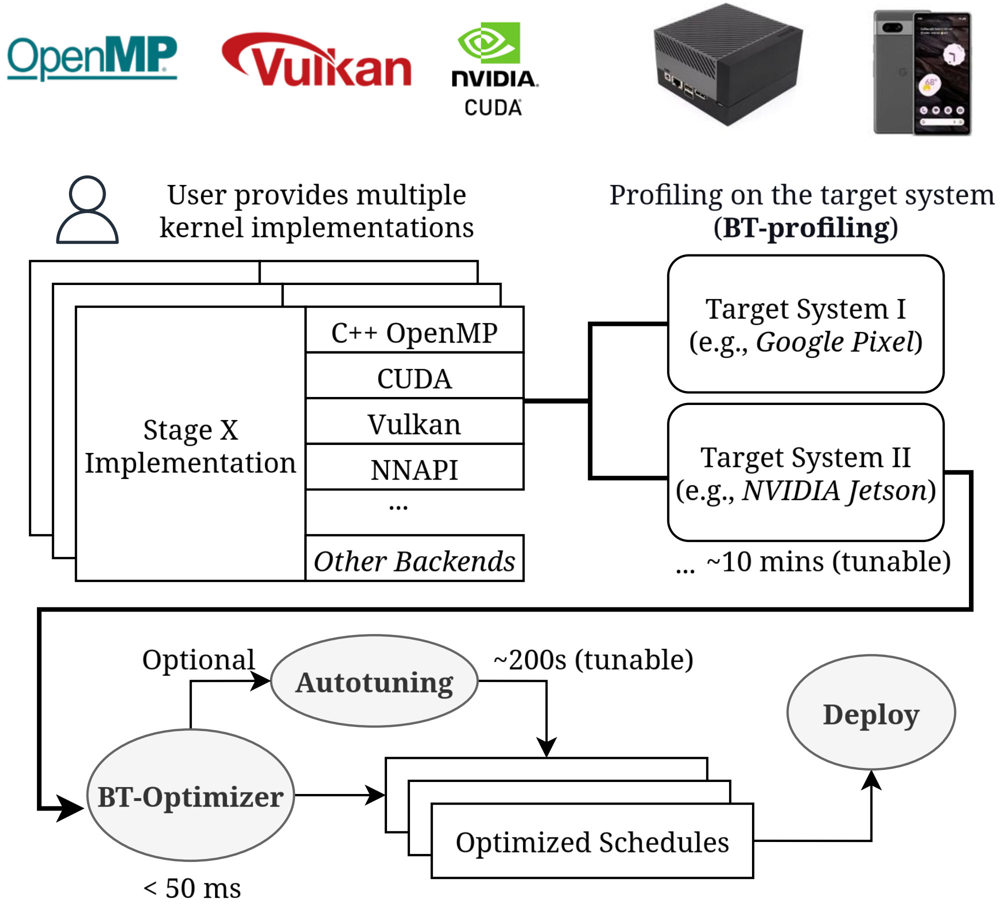
| NVIDIA Jetson |
|---|
| 8 GPU-SMs |
| 6 E-cores |

| Apple A18 SoC |
|---|
| 5 GPU-cores |
| 8 NPU-cores |
| 2 P-cores |
| 4 E-cores |

| Google Pixel 7a |
|---|
| 7 GPU-cores |
| 1 TPU |
| 2 P-cores |
| 2 M-cores |
| 4 E-cores |

# We present ***BetterTogether***

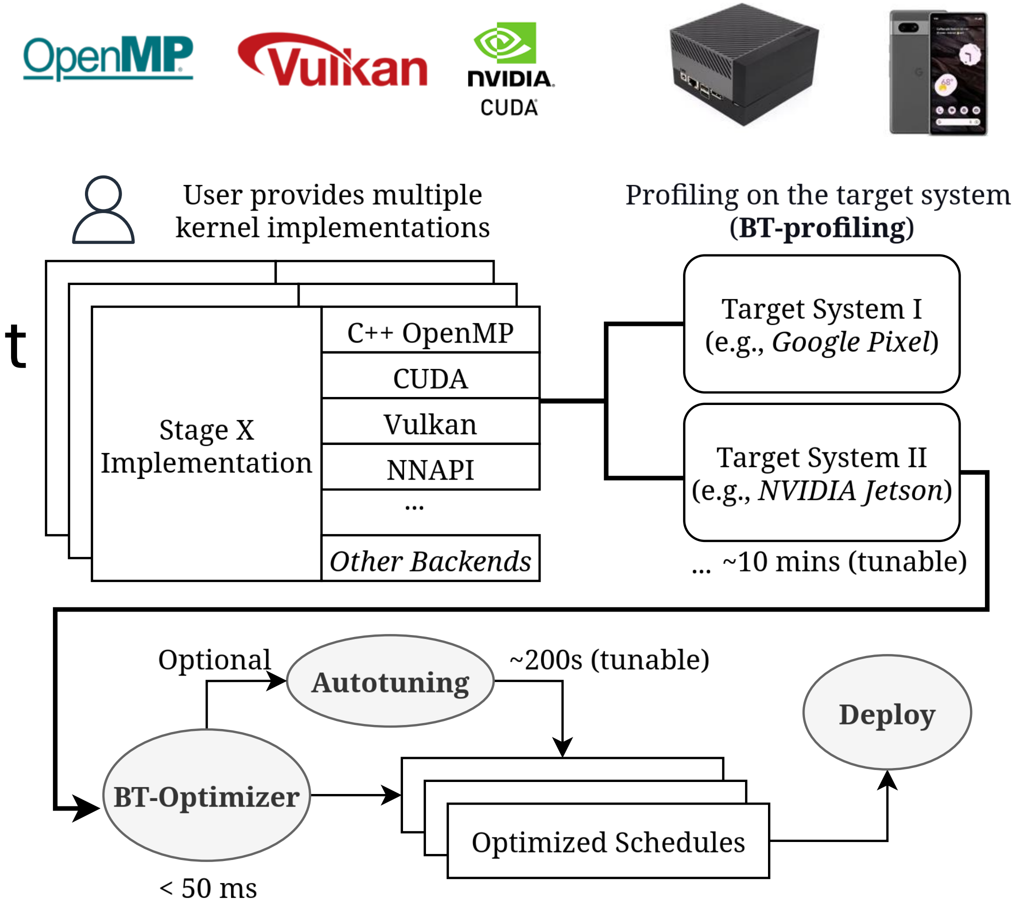
- A performance modeling approach that accounts for intra-application interference




Up to **7.59x** speedup
w/ Avg **2.14x** speedup

# We present ***BetterTogether***

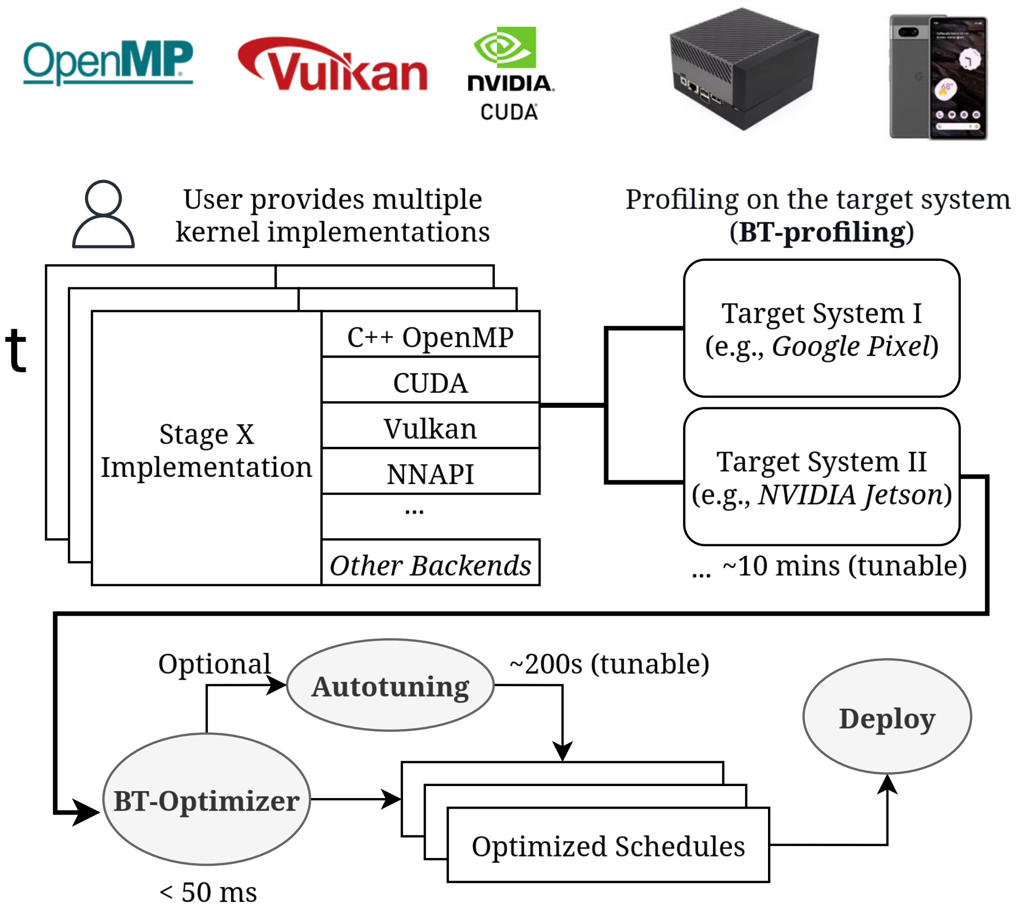
- A performance modeling approach that accounts for intra-application interference
- An end-to-end static pipeline generator that works across a variety of devices




Up to **7.59x** speedup
w/ Avg **2.14x** speedup

# We present ***BetterTogether***

- A performance modeling approach that accounts for intra-application interference
- An end-to-end static pipeline generator that works across a variety of devices
- Consists of 3 major components



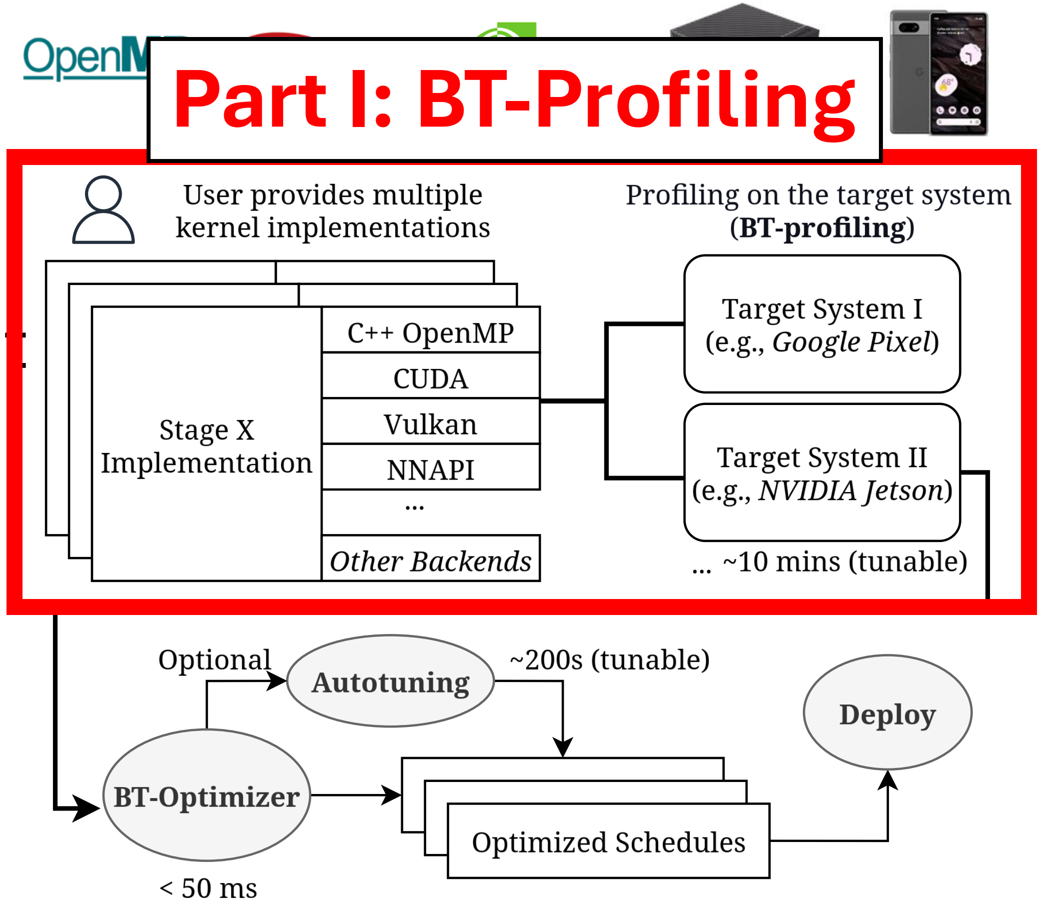

Up to **7.59x** speedup
w/ Avg **2.14x** speedup

# We present *BetterTogether*

- A performance modeling approach that accounts for intra-application interference
- An end-to-end static pipeline generator that works across a variety of devices

- Consists of 3 major components
  - *BetterTogether* **Profiling**
    - Attack ***Challenge I (Interference)***

## Part I: BT-Profiling

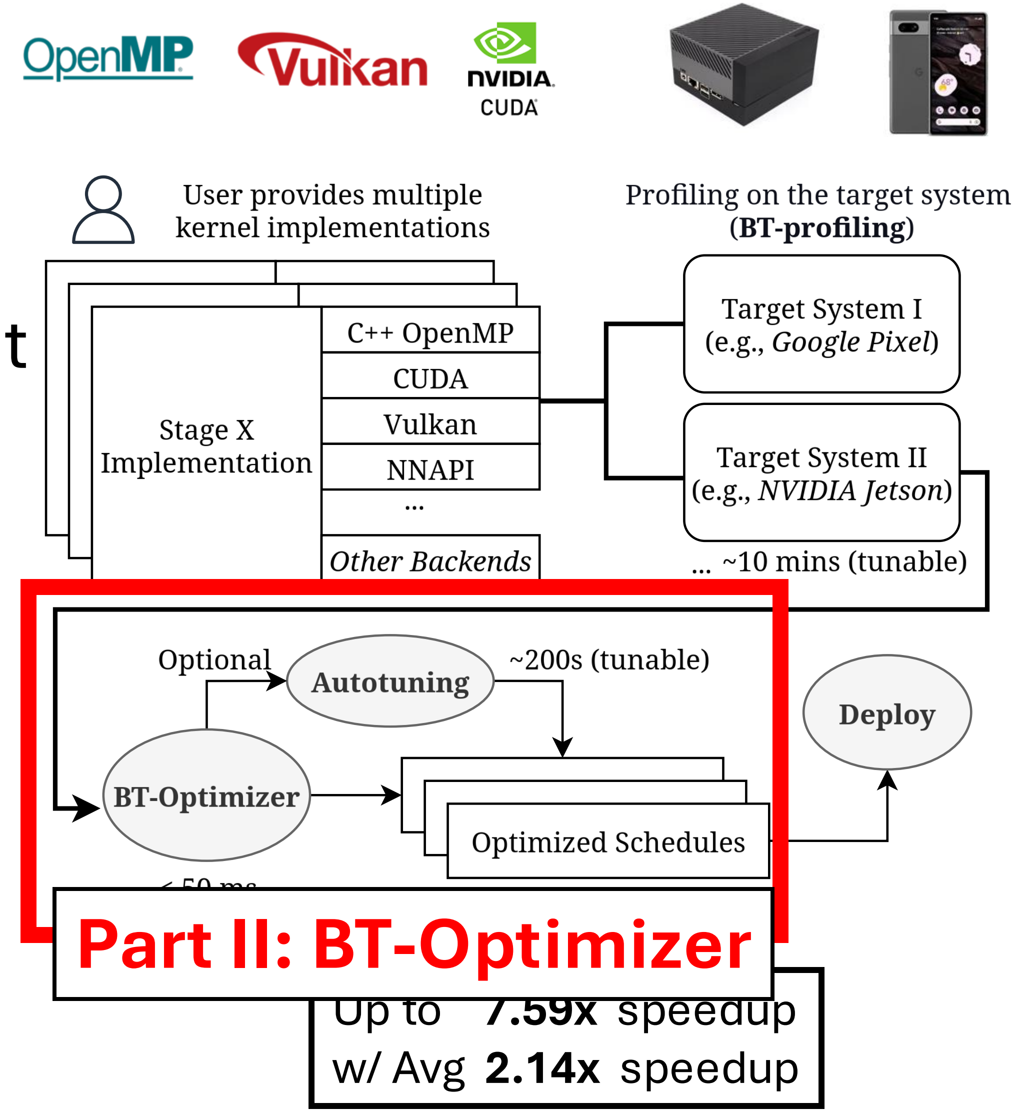



Up to **7.59x** speedup
w/ Avg **2.14x** speedup

# We present ***BetterTogether***

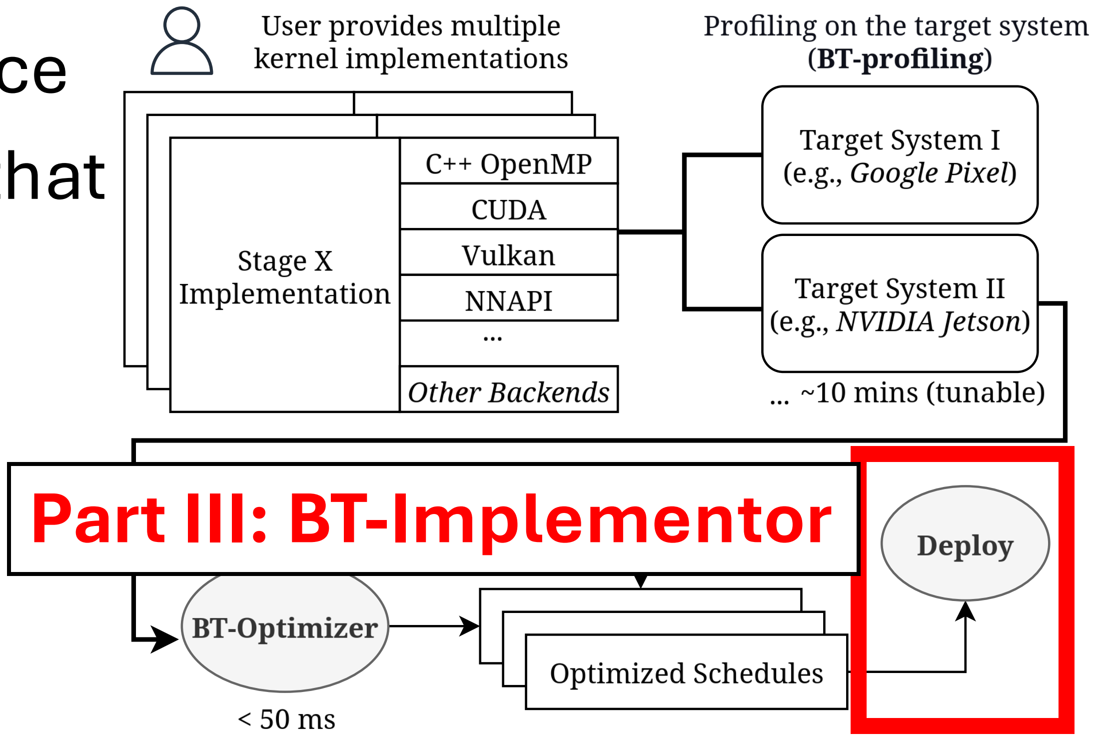
- A performance modeling approach that accounts for intra-application interference
- An end-to-end static pipeline generator that works across a variety of devices

- Consists of 3 major components
  - *BetterTogether* **Profiling**
    - Attack ***Challenge I (Interference)***
  - *BetterTogether* **Optimizer**
    - Attack ***Challenge II (Portability)***



# We present *BetterTogether*

- A performance modeling approach that accounts for intra-application interference
- An end-to-end static pipeline generator that works across a variety of devices




- Consists of 3 major components
  - *BetterTogether* **Profiling**
    - Attack ***Challenge I (Interference)***
  - *BetterTogether* **Optimizer**
    - Attack ***Challenge II (Portability)***
  - *BetterTogether* **Implementor**
    - Efficient static heterogenous pipeline execution

Up to **7.59x** speedup
w/ Avg **2.14x** speedup

# This Work: *BetterTogether* Overview

**Affinity Map**

- 4x Little Cores (**PU1**)
- 2x Medium Cores (**PU2**)
- 2x Big Cores (**PU3**)
- 1x iGPU (**PU4**)

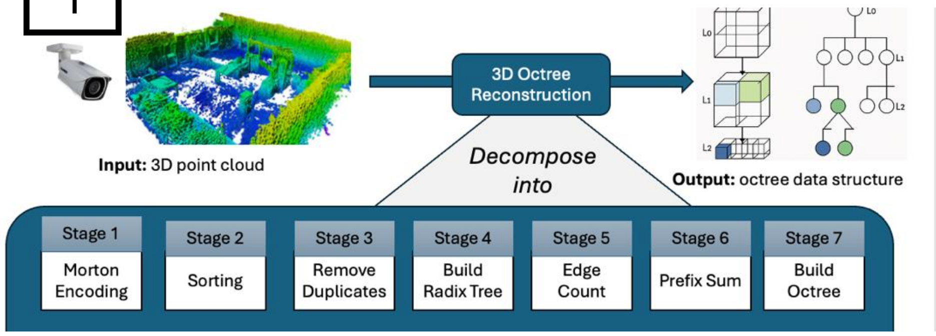
# This Work: *BetterTogether* Overview

| Affinity Map |
| --- |
| 4x Little Cores (**PU1**) |
| 2x Medium Cores (**PU2**) |
| 2x Big Cores (**PU3**) |
| 1x iGPU (**PU4**) |

1

User decompose the workloads into Stages

1



# This Work: *BetterTogether* Overview

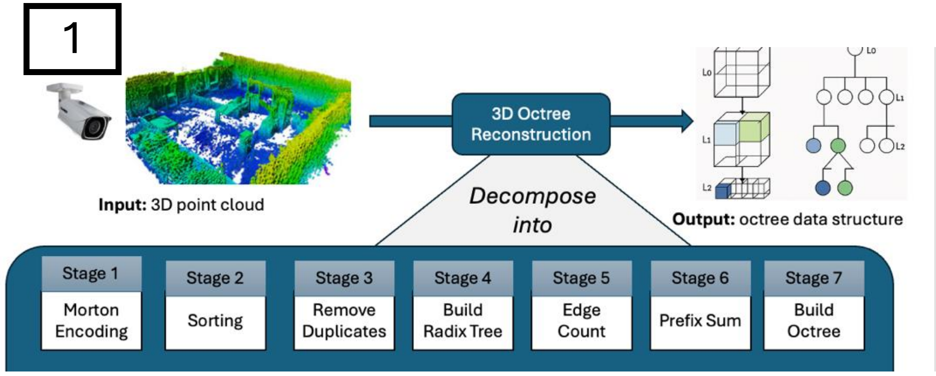
| Affinity Map |
| --- |
| 4x Little Cores **(PU1)** |
| 2x Medium Cores **(PU2)** |
| 2x Big Cores **(PU3)** |
| 1x iGPU **(PU4)** |

1

User decompose the workloads into Stages

2

User provide implementations for each PU




2

```
void run_stage_1_cpu(in, out, N) {
  #pragma omp parallel for
  for (int i = 0; i < N; ++i)
    out[i] = morton32(in[i]);
}
```

*CPU Code (e.g., OpenMP)*

```
__global__ void run_stage_1_gpu(in, out, N) {
  int idx = threadIdx.x + blockDim.x * blockIdx.x;
  int stride = blockDim.x * gridDim.x;
  for (int i = idx; i < N; i += stride)
    out[i] = morton32(in[i]);
}
```

*GPU Code (e.g., CUDA)*

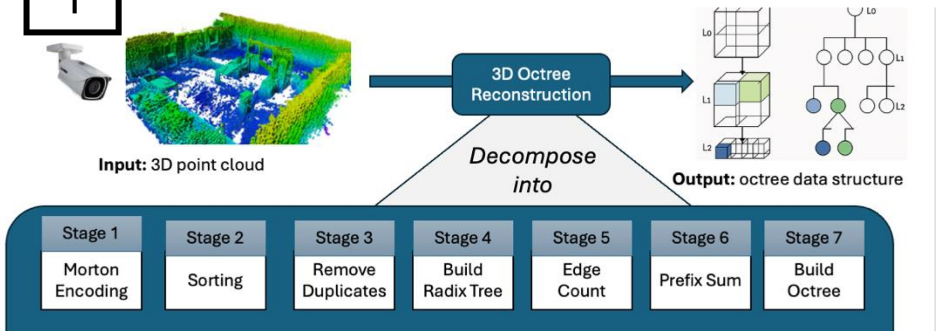
# This Work: *BetterTogether* Overview

| Affinity Map |
|---|
| 4x Little Cores (**PU1**) |
| 2x Medium Cores (**PU2**) |
| 2x Big Cores (**PU3**) |
| 1x iGPU (**PU4**) |

1

User decompose the workloads into Stages

1




2

User provide implementations for each PU

2

```
void run_stage_1_cpu(in, out, N) {
  #pragma omp parallel for
  for (int i = 0; i < N; ++i)
    out[i] = morton32(in[i]);
}
```

*CPU Code (e.g., OpenMP)*

```
__global__ void run_stage_1_gpu(in, out, N) {
  int idx = threadIdx.x + blockDim.x * blockIdx.x;
  int stride = blockDim.x * gridDim.x;
  for (int i = idx; i < N; i += stride)
    out[i] = morton32(in[i]);
}
```

*GPU Code (e.g., CUDA)*

Interference-aware Profiling Table

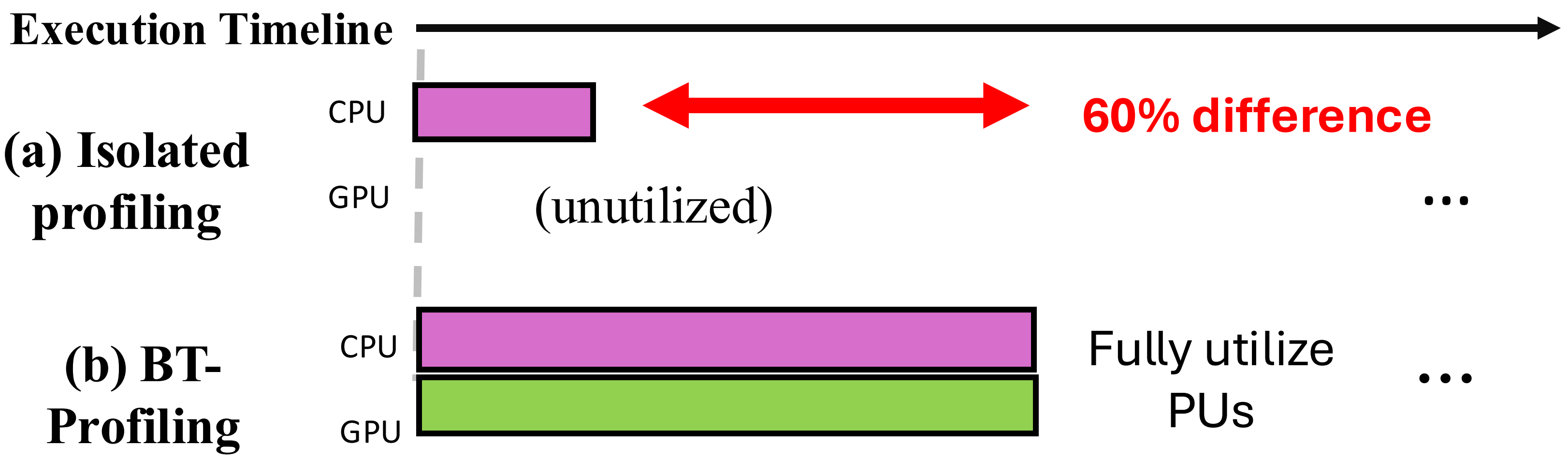
| | **S1** | **S2** | .. | **S7** |
|---|---|---|---|---|
| **PU0** | 2.6 | 3.3 | | 5.8 |
| **PU1** | 0.8 | 1.5 | | 1.9 |
| **PU2** | 0.6 | 1.4 | | 2.1 |
| **PU3** | 0.8 | 9.0 | | 1.5 |

Interference aware

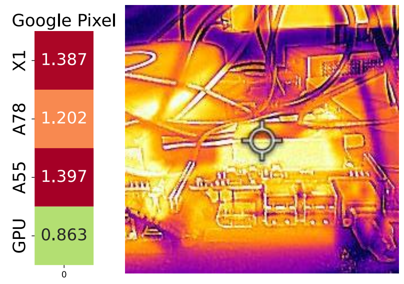
**BT-Profiling**

# **BT-Profiling** - *Interference aware profiling*

- While profiling each {PU × Stage} pair:
  - Concurrently execute similar stages on other PUs
  - Simulate **whole-application** execution to capture resource contention




**More Accurate**




**Overcome**

# BT-Optimizer

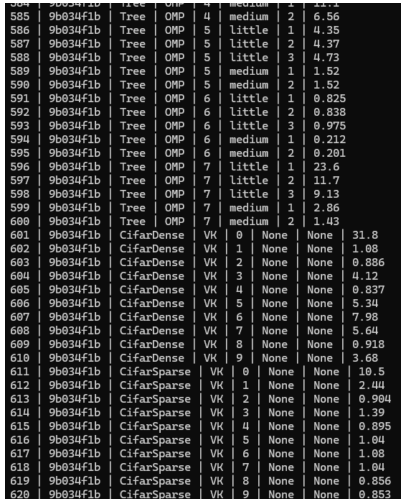
- We express our optimization problem as linear constrains

*d) Notation and Decision Variables:*

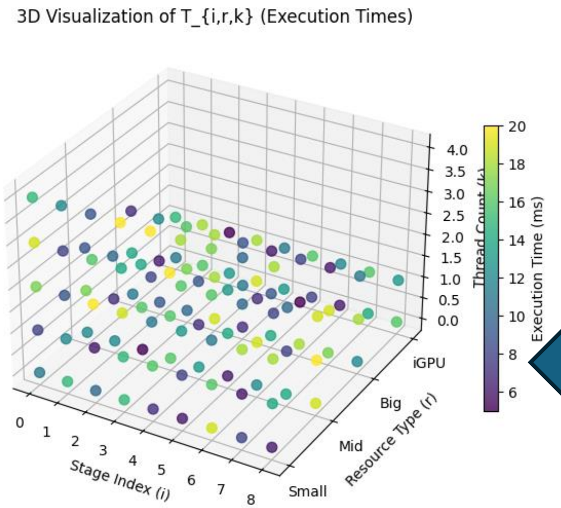
| | |
|---|---|
| $N$ | Total number of pipeline stages |
| $N_i$ | The pipeline stage $i$, with $i \in \mathcal{N} = \{0, \ldots, N-1\}$ |
| $\mathcal{C}$ | PU classes: $\mathcal{C} = \{c_1, \ldots, c_M\}$ |
| $t_{i,c}$ | Profiled latency of stage $i$ on PU $c$ |
| $x_{i,c}$ | Decision variable: $x_{i,c} \in \{0, 1\}$; |
| | $x_{i,c} = 1 \Leftrightarrow$ stage $i$ runs on PU $c$ |

- We propose a **three-step** optimization approach

*Raw BT Profiling results*



```
585 | 9b034f1b | Tree | OMP | 4 | medium | 2 | 6.56
586 | 9b034f1b | Tree | OMP | 5 | little | 1 | 4.35
587 | 9b034f1b | Tree | OMP | 5 | little | 2 | 4.37
588 | 9b034f1b | Tree | OMP | 5 | little | 3 | 4.73
589 | 9b034f1b | Tree | OMP | 5 | medium | 1 | 1.52
590 | 9b034f1b | Tree | OMP | 5 | medium | 2 | 1.52
591 | 9b034f1b | Tree | OMP | 6 | little | 1 | 0.825
592 | 9b034f1b | Tree | OMP | 6 | little | 2 | 0.838
593 | 9b034f1b | Tree | OMP | 6 | little | 3 | 0.975
594 | 9b034f1b | Tree | OMP | 6 | medium | 1 | 0.212
595 | 9b034f1b | Tree | OMP | 6 | medium | 2 | 0.201
596 | 9b034f1b | Tree | OMP | 7 | little | 1 | 23.6
597 | 9b034f1b | Tree | OMP | 7 | little | 2 | 11.7
598 | 9b034f1b | Tree | OMP | 7 | little | 3 | 9.13
599 | 9b034f1b | Tree | OMP | 7 | medium | 1 | 2.86
600 | 9b034f1b | Tree | OMP | 7 | medium | 2 | 1.43
601 | 9b034f1b | CifarDense | VK | 0 | None | None | 31.8
602 | 9b034f1b | CifarDense | VK | 1 | None | None | 1.08
603 | 9b034f1b | CifarDense | VK | 2 | None | None | 0.886
604 | 9b034f1b | CifarDense | VK | 3 | None | None | 4.12
605 | 9b034f1b | CifarDense | VK | 4 | None | None | 0.837
606 | 9b034f1b | CifarDense | VK | 5 | None | None | 5.34
607 | 9b034f1b | CifarDense | VK | 6 | None | None | 7.98
608 | 9b034f1b | CifarDense | VK | 7 | None | None | 5.64
609 | 9b034f1b | CifarDense | VK | 8 | None | None | 0.918
610 | 9b034f1b | CifarDense | VK | 9 | None | None | 3.68
611 | 9b034f1b | CifarSparse | VK | 0 | None | None | 10.5
612 | 9b034f1b | CifarSparse | VK | 1 | None | None | 2.44
613 | 9b034f1b | CifarSparse | VK | 2 | None | None | 0.904
614 | 9b034f1b | CifarSparse | VK | 3 | None | None | 1.39
615 | 9b034f1b | CifarSparse | VK | 4 | None | None | 0.895
616 | 9b034f1b | CifarSparse | VK | 5 | None | None | 1.04
617 | 9b034f1b | CifarSparse | VK | 6 | None | None | 1.08
618 | 9b034f1b | CifarSparse | VK | 7 | None | None | 1.04
619 | 9b034f1b | CifarSparse | VK | 8 | None | None | 0.856
620 | 9b034f1b | CifarSparse | VK | 9 | None | None | 0.853
```



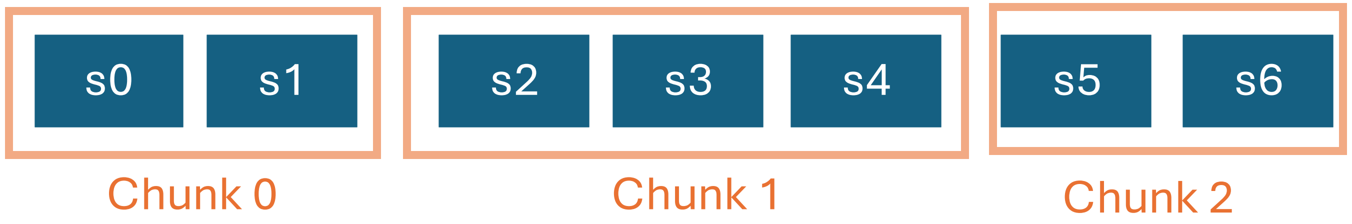

Interference-aware Profiling Table

| | S1 | S2 | .. | S7 |
|---|---|---|---|---|
| PU0 | 2.6 | 3.3 | | 5.8 |
| PU1 | 0.8 | 1.5 | | 1.9 |
| PU2 | 0.6 | 1.4 | | 2.1 |
| PU3 | 0.8 | 9.0 | | 1.5 |

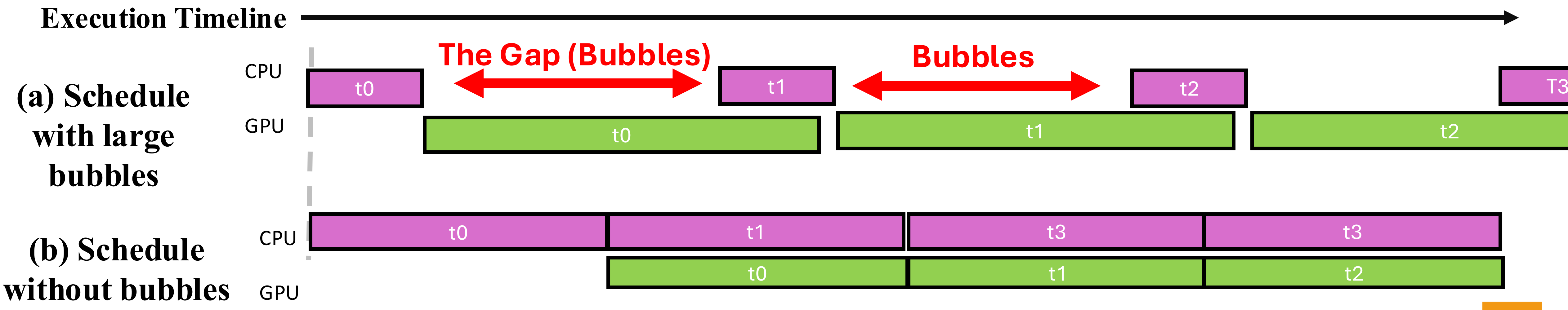
*BT Profiling Table*

**SMT Solver**

# BT-Optimizer **Step 1 -** Minimizing Pipeline Bubbles

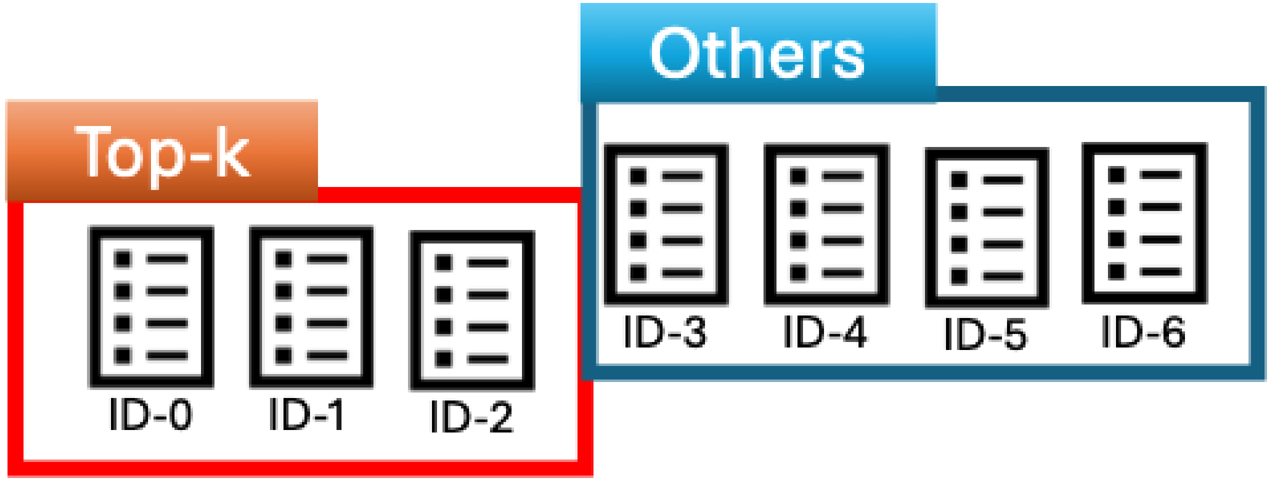



- Improve utilization by reducing idle gaps (*pipeline bubbles*) across PUs
- By **reducing bubbles**, we **keep all PUs busy** and improve pipeline throughput.

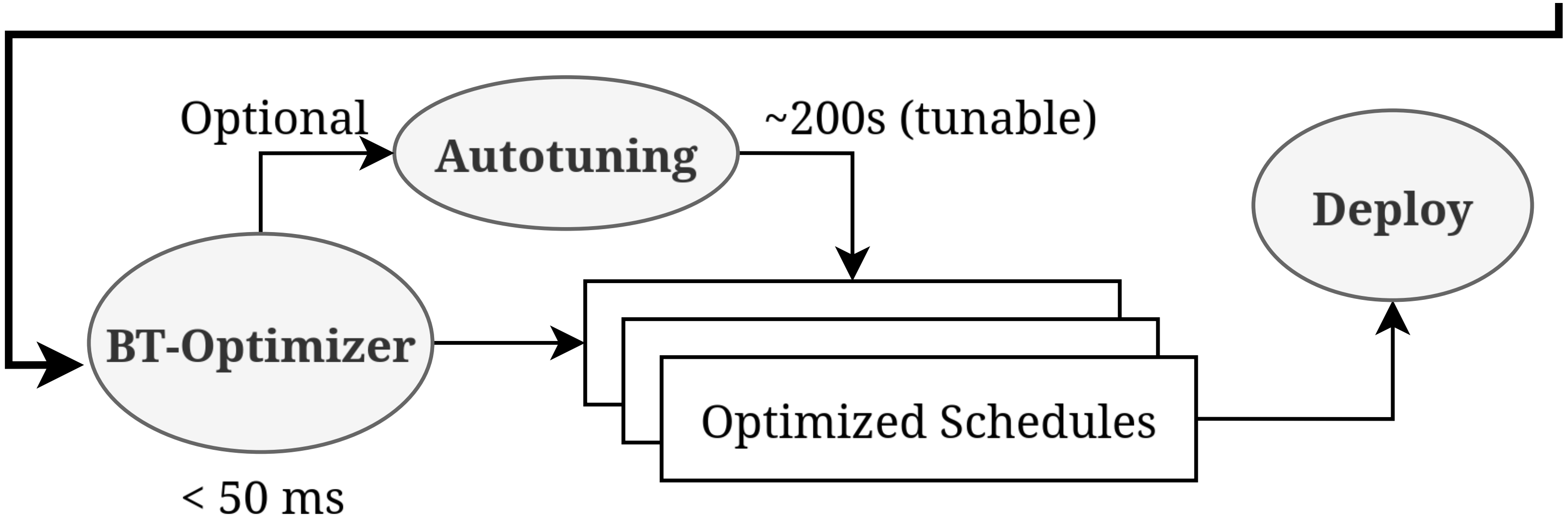


# BT-Optimizer **Step 2: Optimizing Latency**

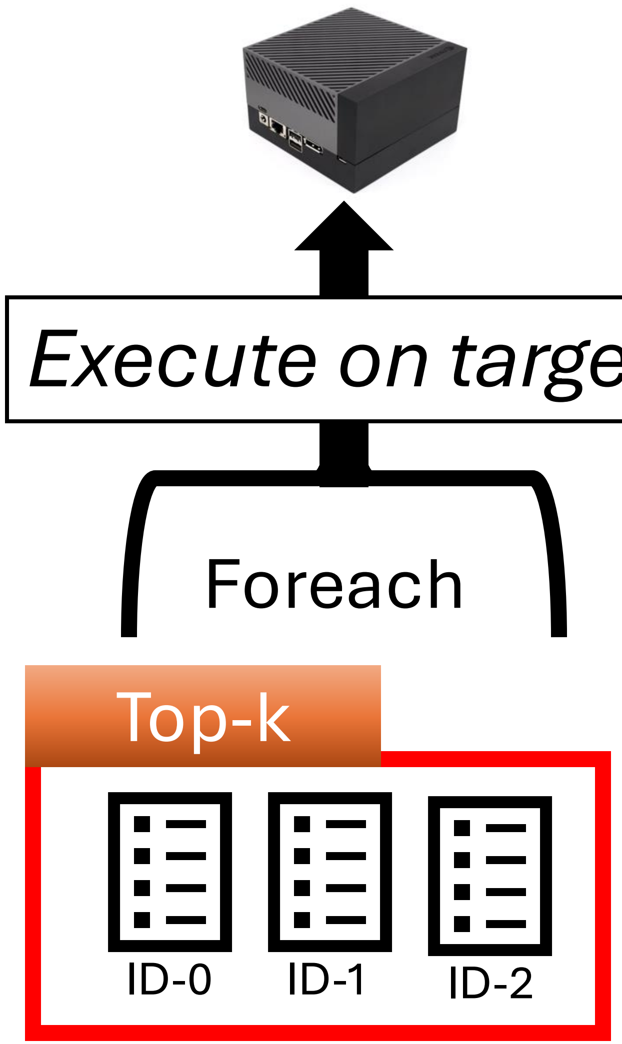
- High utilization ≠ **low latency**
- Generate K schedules (e.g., k = 20),
  - each with a different assignment of stages to PUs.
- Minimize latency

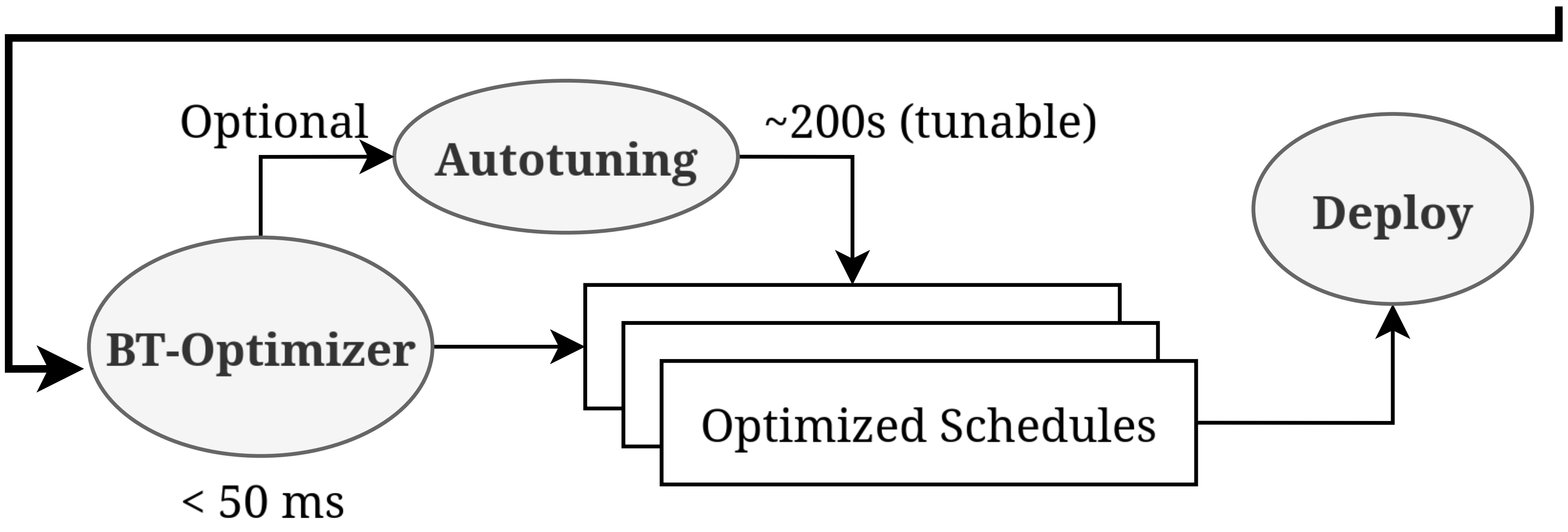


# BT-Optimizer **Optional Step 3: Autotuning**



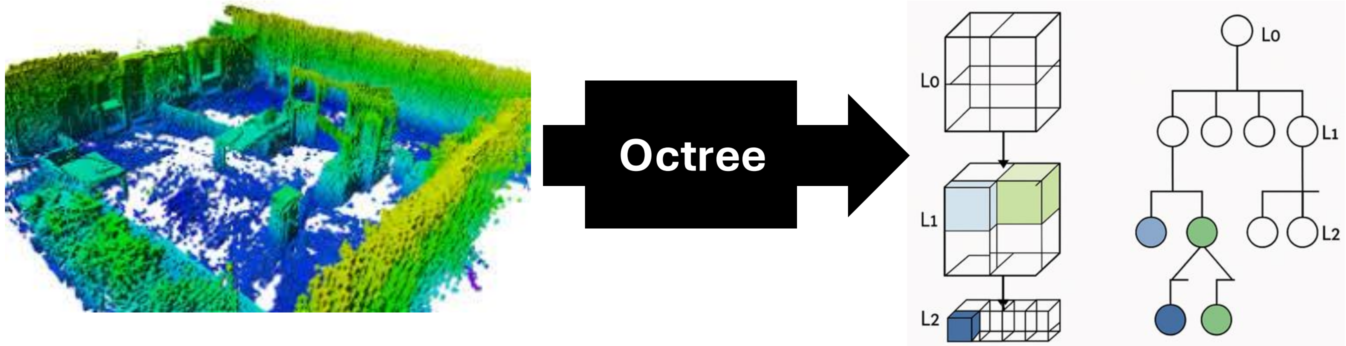
# BT-Optimizer **Optional Step 3: Autotuning**

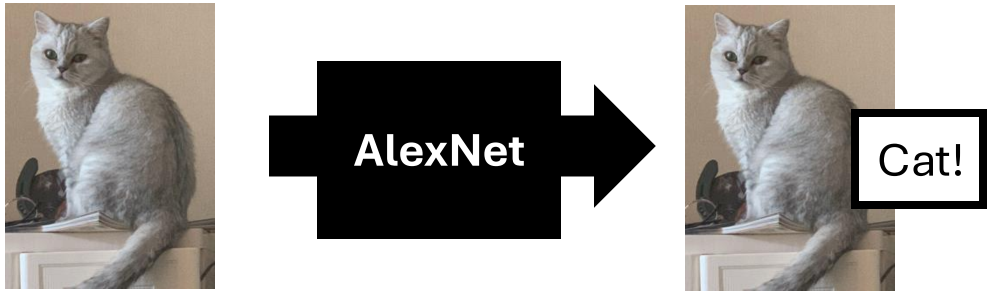


# Workloads Evaluated

- Three edge compute vision tasks
    1. **AlexNet-dense**
    2. **AlexNet-sparse**
    3. **Octree Construction**
- Common in resource-constrained environments




- *AlexNet-dense*
    - Dense linear algebra
    - CIFAR-10 dataset
- *AlexNet-sparse*
    - Sparse linear algebra
    - Pruned w/ CONDENSA*
    - Stored in CSR
- *Octree*
    - Tree traversals
    - Irregular memory access
    - Sorting
    - Prefix Sum

* *https://github.com/NVlabs/condensa*

# Platforms Evaluated

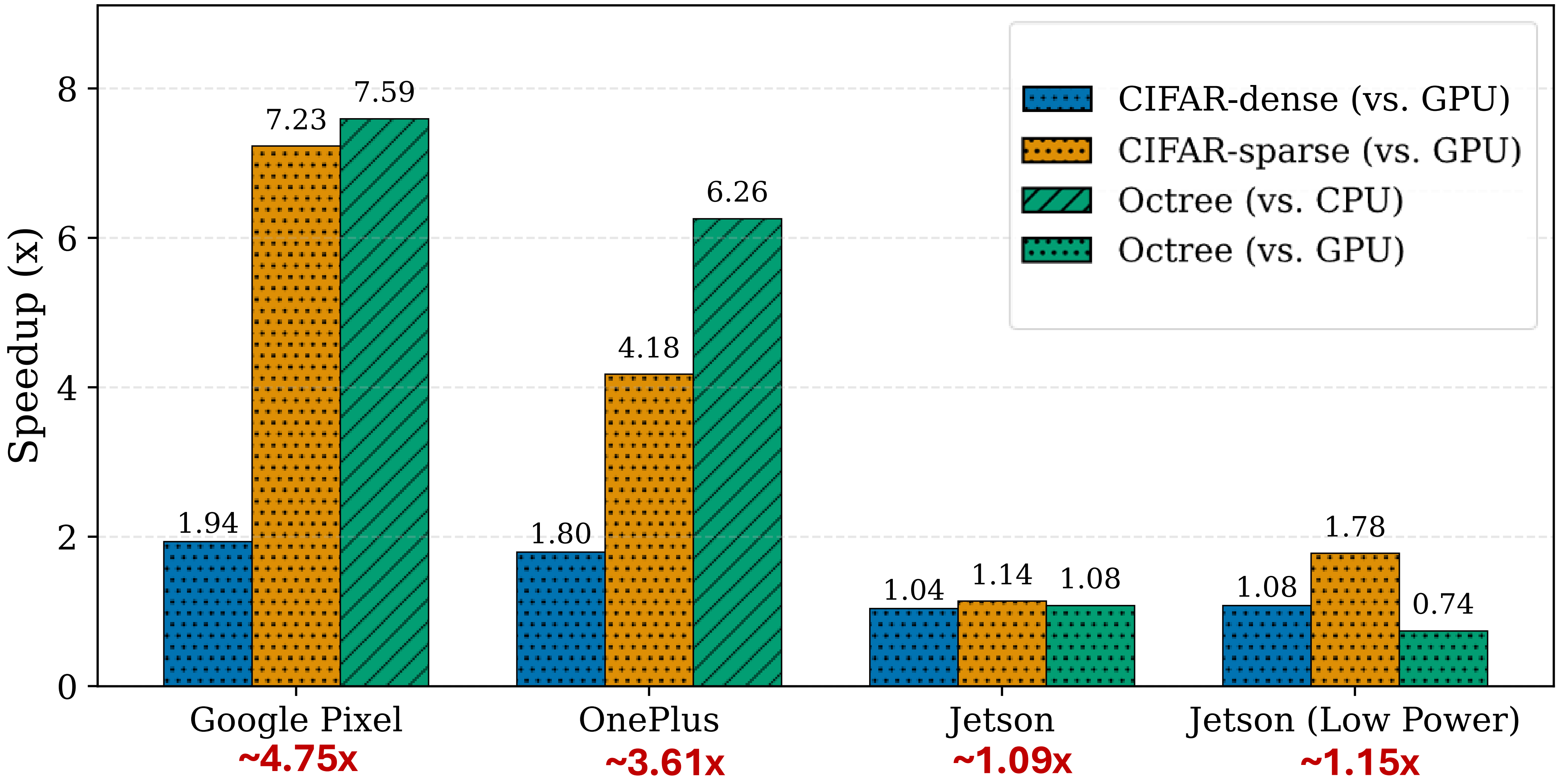
Less Powerful GPUs
Mix CPUs

Powerful GPUs
Little ARM CPUs

| Platform | Backe nd | CPU | CPU Frequency | Integrated GPU |
|---|---|---|---|---|
| **Google Pixel 7A** | Vulkan | 2x Cortex-X1<br>2x Cortex-A78<br>4x Cortex-A55 | 2.85 GHz<br>2.35 GHz<br>1.80 GHz | Mali-G710 MP7 |
| **OnePlus 11** | Vulkan | 1x Coretex-X3<br>2x Coretex-A715<br>2x Cortex-A710<br>3x Cortex-A510 | 3.2 GHz<br>2.8 GHz<br>2.8 GHz<br>2.0 GHz | Adreno 740 |
| **NVIDIA Jetson Orin Nano** | CUDA<br>Vulkan | 6x Cortex-A78AE | 1.7 GHz | Ampere GPU |
| ***NVIDIA Jetson Orin Nano (low-power mode)** | CUDA<br>Vulkan | 4x Cortex-A78AE | ~0.85 GHz | Ampere GPU |

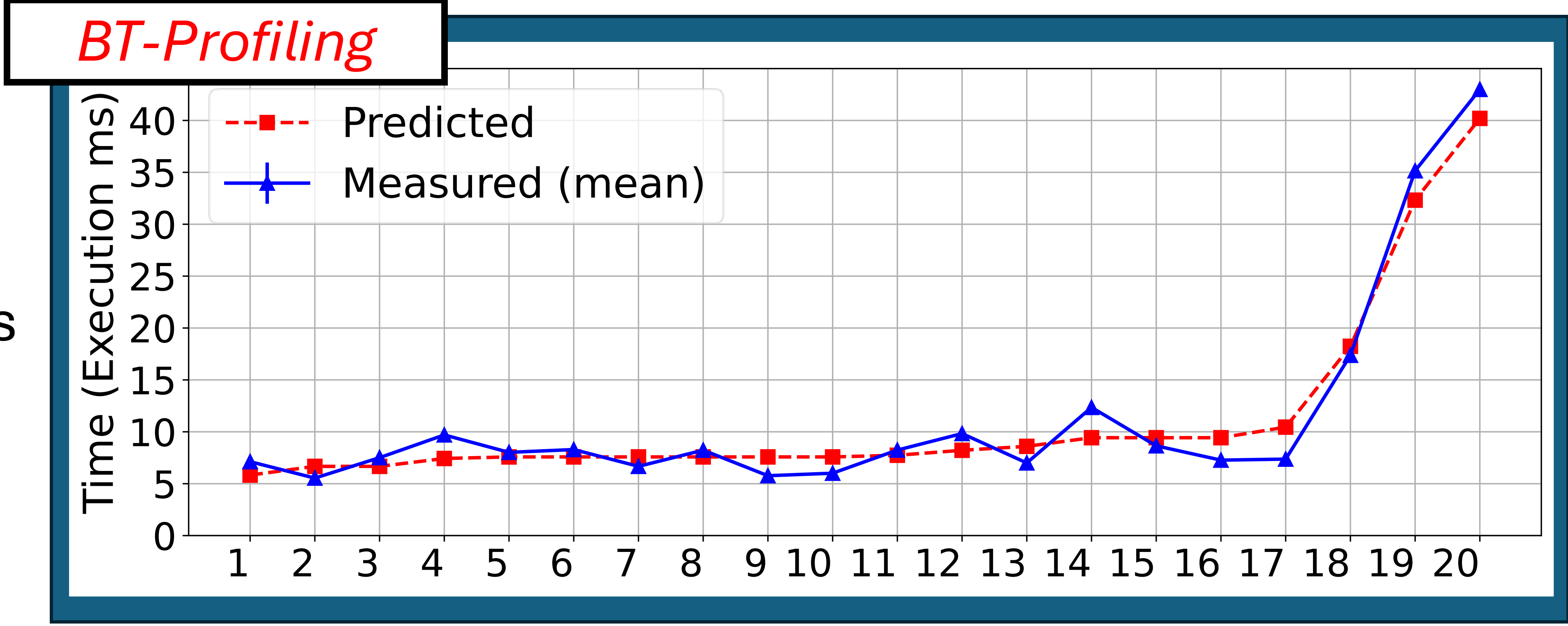
** In low-power mode, 2 cores are shutdown, and overall CPU frequence is reduced by half*

# Results Overview

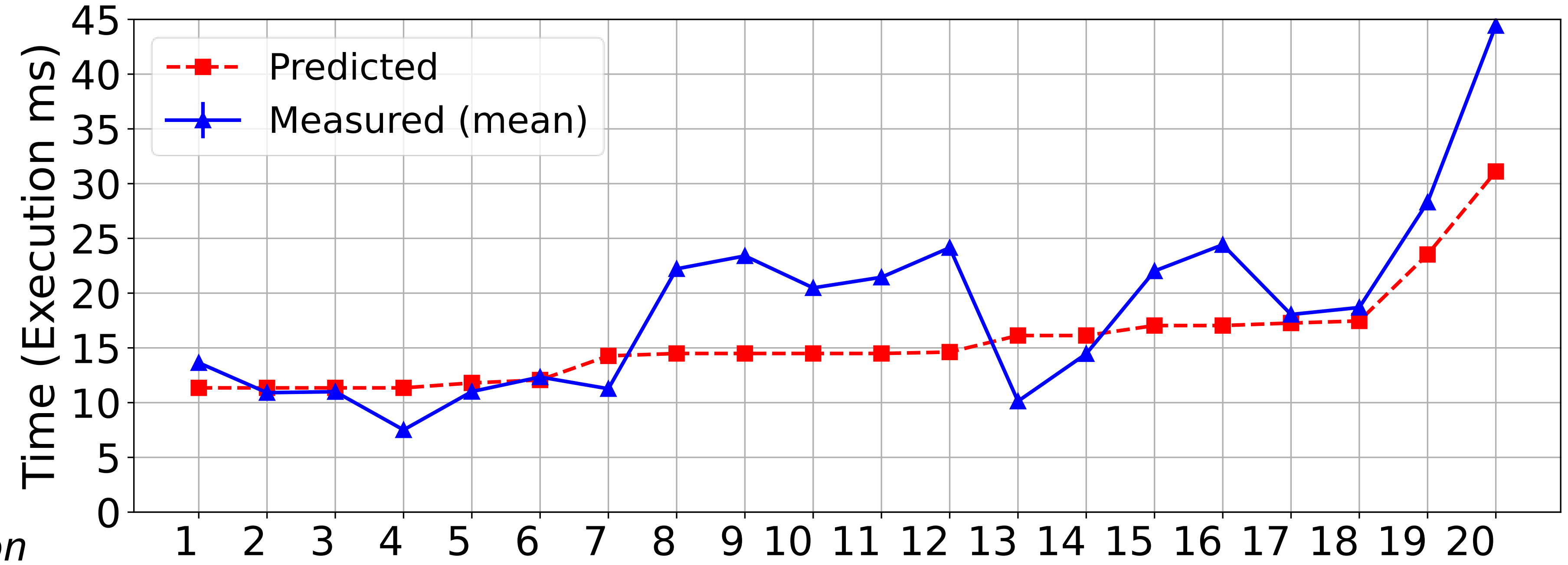



- Geomean speedup of **2.14x** across all workloads, w/ peak of **7.59x**
- In 1 case, slowdown

*BetterTogether* produces predictions that closely match measured execution time



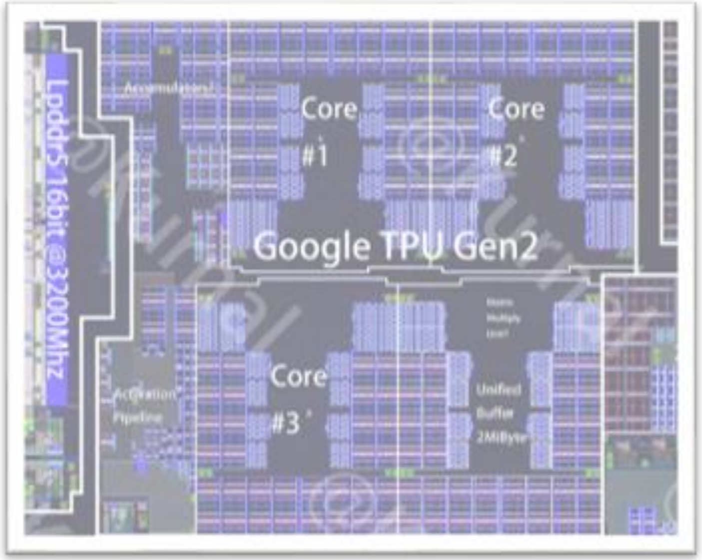

Without BT show discrepancies




*predicted and measured execution*

# We have additional insights in the paper

- Preliminary results on
  - Using Google's **EdgeTPU**
  - Implemented using *nnapi*
  - Showing a 1.25x speedup for AlexNet-dense application on top of existing results
  - Showcasing the flexibility and extensibility of *BetterTogether*

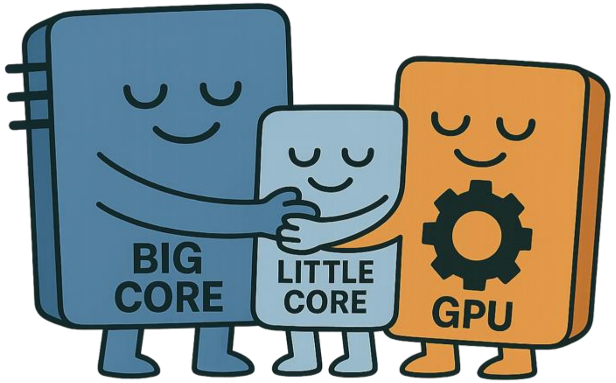


# Conclusion

- We propose ***BetterTogether***
  - An **interference-aware profiling** method that produce accurate profiling tables by accounting for *intra-application interference*
  - an end-to-end static pipeline generator for edge SoCs
- Using *BetterTogether*, we implemented **3** class of applications
- Evaluated across **4** diverse devices, achieving up to **7.59x** **(geo. 2.14x)**


**Team**

**Yanwen Xu**, Rithik Sharma, Zheyuan Chen
Shaan Mistry, Tyler Sorensen
*{**yxu83**, riksharm, zchen406, sdmistry, tyler.sorensen}**@ucsc.edu***

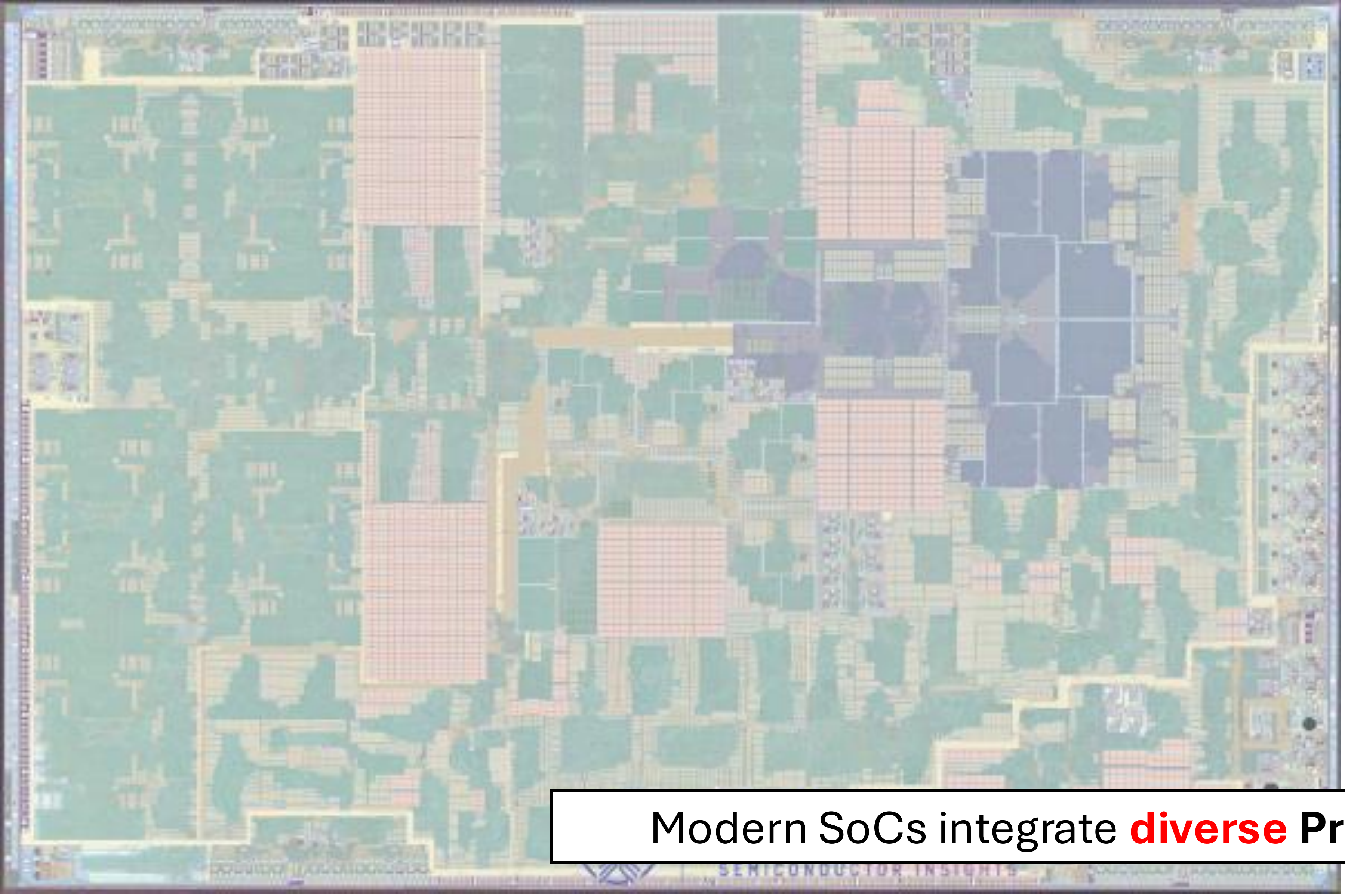




**Open-Source Repo**

github.com/ucsc-redwood/better-together

Baskin Engineering
Microsoft

# Backup slides

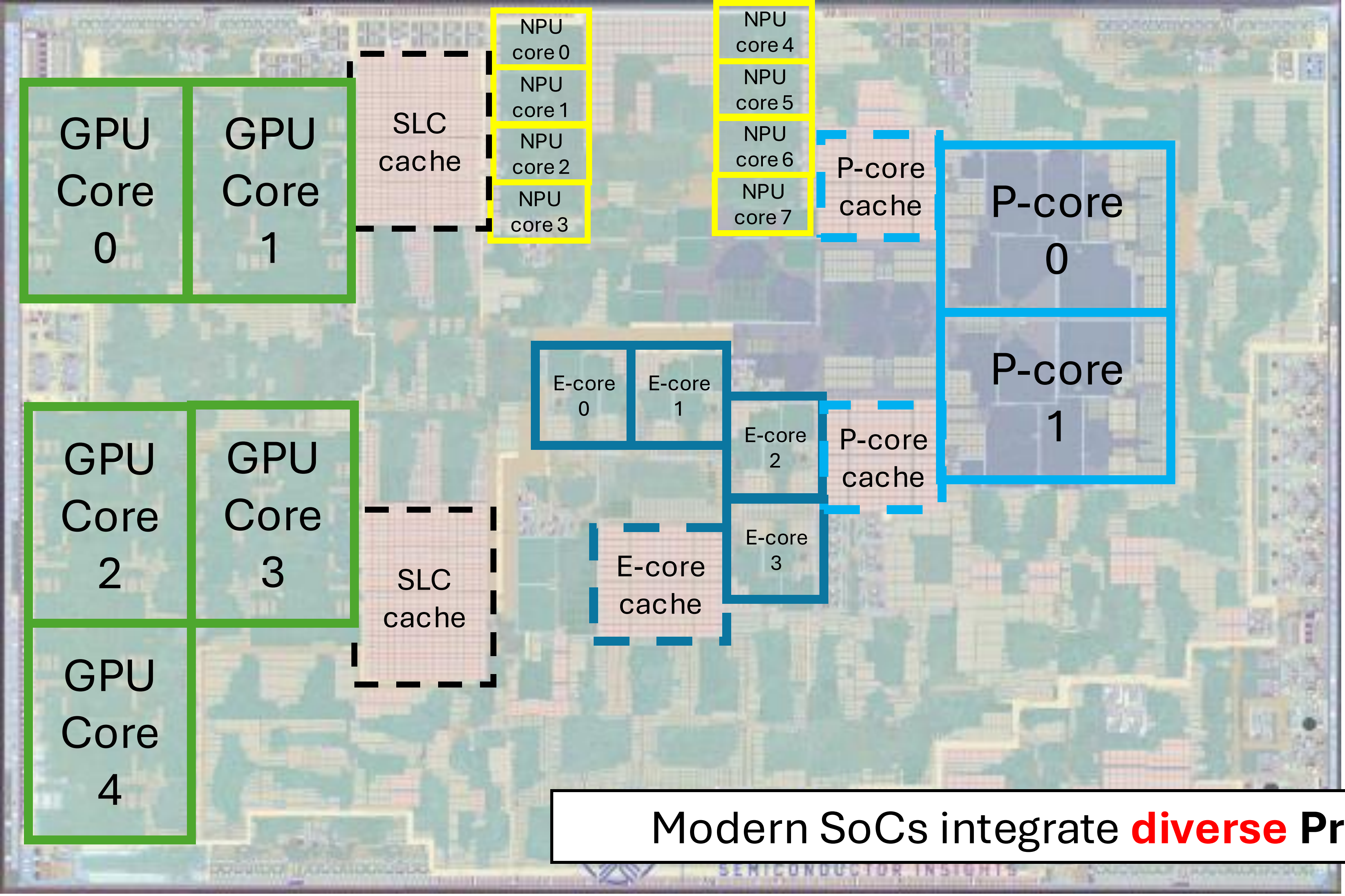



Apple A18 SoC
90mm2 @ TSMC N3E

- *big.LITTLE CPU*
- *Integrated GPU*
- *Domain Specific Accelerators (DSA)*
- …

Modern SoCs integrate **diverse** **Processing Units** (**PU**)

*Image by ChipWise: https://chipwise.tech/our-portfolio/apple-a18-a18-pro-die-shot/*




Apple A18 SoC
90mm2 @ TSMC N3E

5 GPU-cores
8 NPU-cores
2 P-cores
4 E-cores

- *big.LITTLE CPU*
- *Integrated GPU*
- *Domain Specific Accelerators (DSA)*
- *...*

Modern SoCs integrate **diverse** **Processing Units** (**PU**)

*Image by ChipWise: https://chipwise.tech/our-portfolio/apple-a18-a18-pro-die-shot/*

*We want to utilize all system resources (i.e., PUs)*

# Profiling-guided approach (isolated benchmarking)

Developer

- Given $N$ stages
- Given $M$ types of PU

Benchmarking each {*PU x stage*} *pair* on the target system

*Get a N x M profiling table*

| | $P_0$ | … | $P_m$ |
|---|---|---|---|
| $S_0$ | 1.2 | | 2.5 |
| … | | … | |
| $S_N$ | 6.1 | | 4.5 |

Optimize

**Optimal Schedule**

*implementation*

Deploy

*Expect **X** speedup*

# Profiling-guided approach (isolated benchmarking)

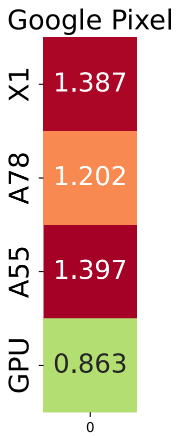
Developer

- Given $N$ stages
- Given $M$ types of PU

Benchmarking each {*PU x stage*} *pair* on the target system

*Get a N x M profiling table*

**Inaccurate profiling**




| | $P_0$ | … | $P_m$ |
|---|---|---|---|
| $S_0$ | ~~1.2~~ | | ~~2.5~~ |
| … | | ~~…~~ | |
| $S_N$ | ~~6.1~~ | | ~~4.5~~ |

Optimize

**Inaccuracy propagate**

**Suboptimal Schedule**

*implementation*

Deploy

*Expect ~~X~~ speedup*

**e.g., Only X/2 speedup**

# BT Implementor

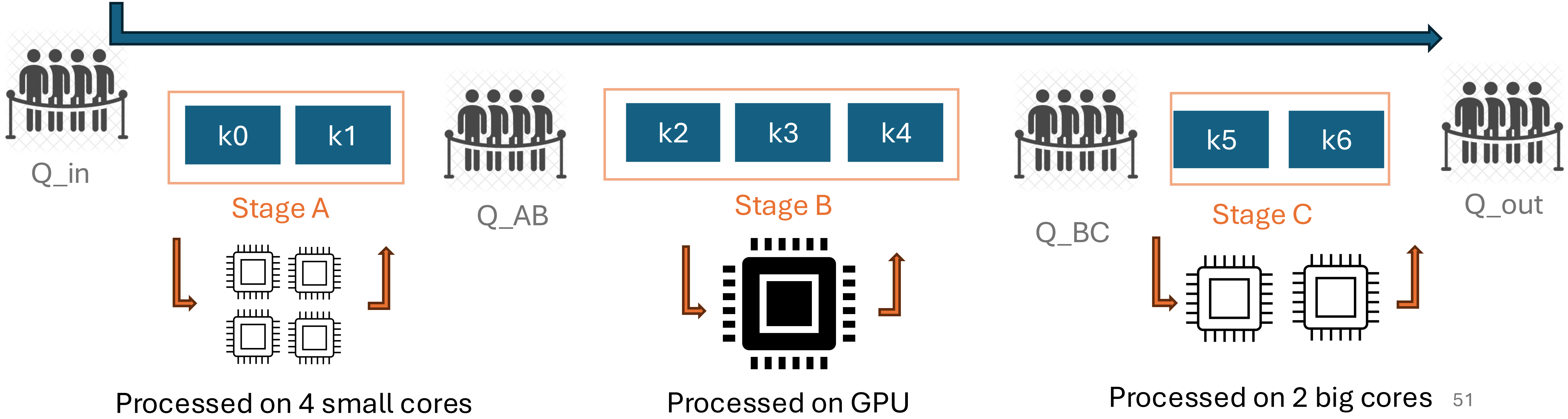
- We define <u>Task</u>
  - light weight, pointing to CPU/iGPU shared memory

```
struct Task {
        uint32_t uid;
        float* u_input;
        float* u_output;
        size_t n;
};
```

- Using concurrent Queue to pass Tasks around stages.

  Each chunk process the incoming tasks in respective type for Cores

CPU use thread affinity



- BetterTogether yields higher correlations

*BetterTogether*

| | OnePlus | Google | Jetson | Jetson (LP) | Avg. |
|---|---|---|---|---|---|
| CIFAR-D | 0.9968 | 0.9990 | 0.9491 | 0.9548 | 0.9749 |
| CIFAR-S | 0.9684 | 0.9441 | 0.8668 | 0.8926 | 0.9180 |
| Tree | 0.9418 | 0.8450 | 0.8283 | 0.8886 | 0.8759 |
| Avg. | 0.9690 | 0.9294 | 0.8814 | 0.9120 | 0.9229 |

| | OnePlus | Google | Jetson | Jetson (LP) | Avg. |
|---|---|---|---|---|---|
| CIFAR-D | 0.9740 | 0.9497 | 0.9481 | 0.9472 | 0.9547 |
| CIFAR-S | 0.9678 | 0.8887 | 0.7005 | 0.7325 | 0.8224 |
| Tree | 0.9816 | 0.8220 | 0.6532 | 0.6839 | 0.7852 |
| Avg. | 0.9745 | 0.8868 | 0.7673 | 0.7879 | 0.8541 |

*Correlation (0.0–1.0) between predicted and actual times across all applications and platforms.* ***Higher is better.***