

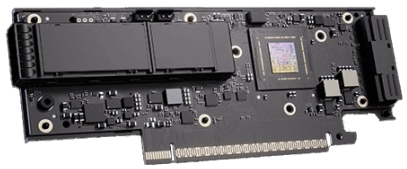
SBB: Eliminating Centralized Bottlenecks in Userspace Network Runtimes

Kang Hu, Shuqi Dong, Chuandong Li, Ran Yi, Zonghao Zhang, Yiming Yao, Bo An, Jie Zhang, Xiaolin Wang, Yingwei Luo, Zhenlin Wang, Diyu Zhou

PEKING UNIVERSITY

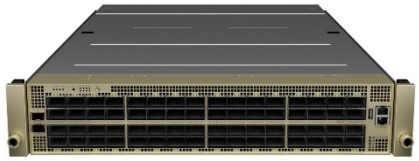
Michigan Tech

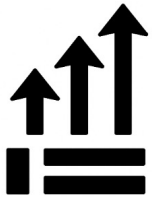
Modern Network Hardware



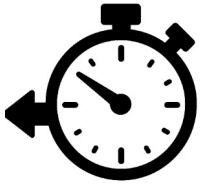






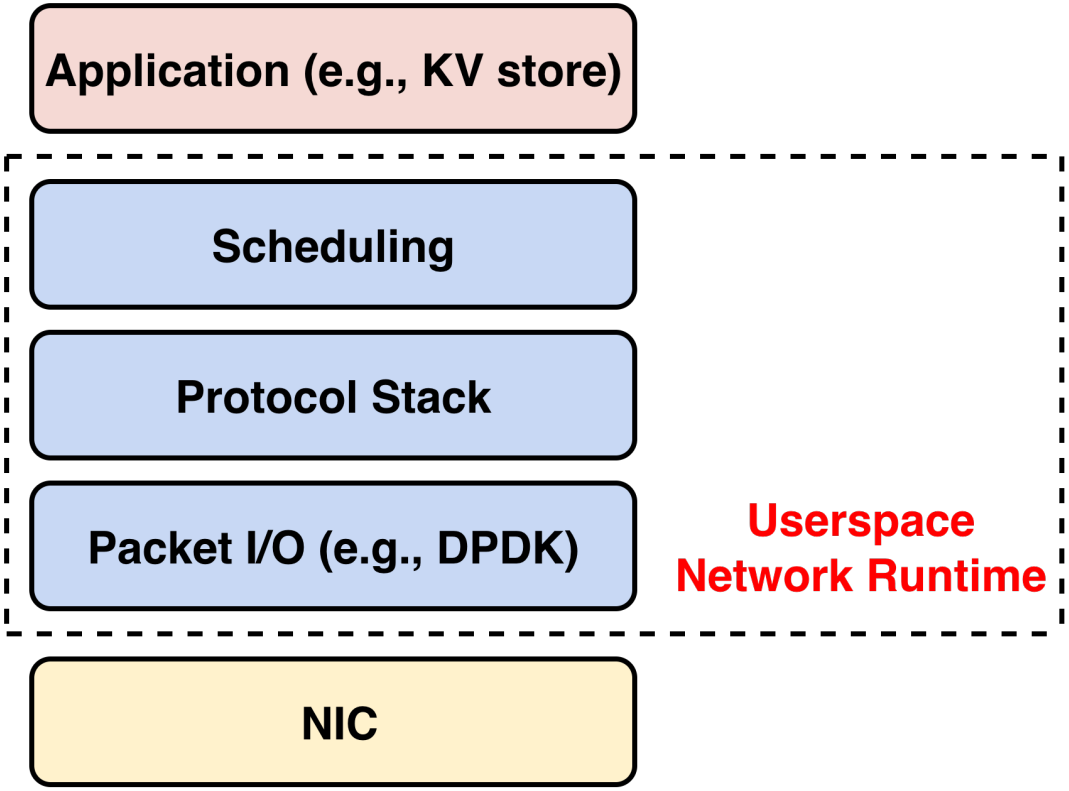


Hundreds of Gb/s

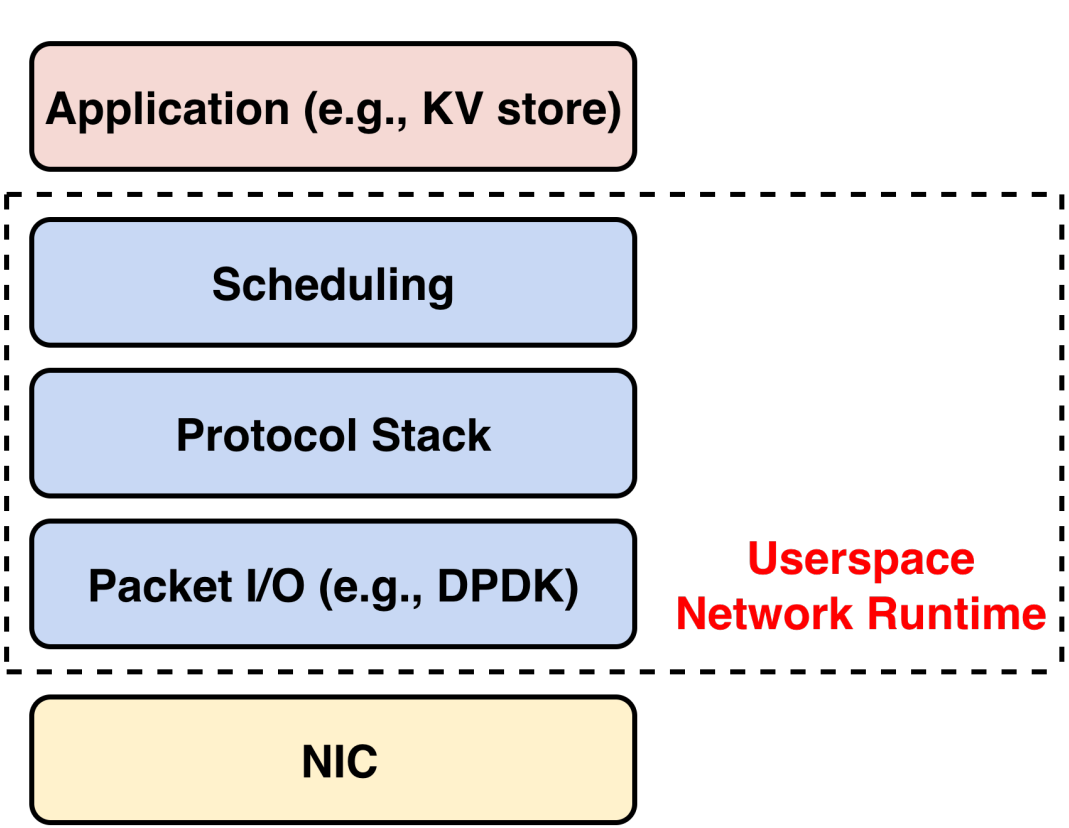


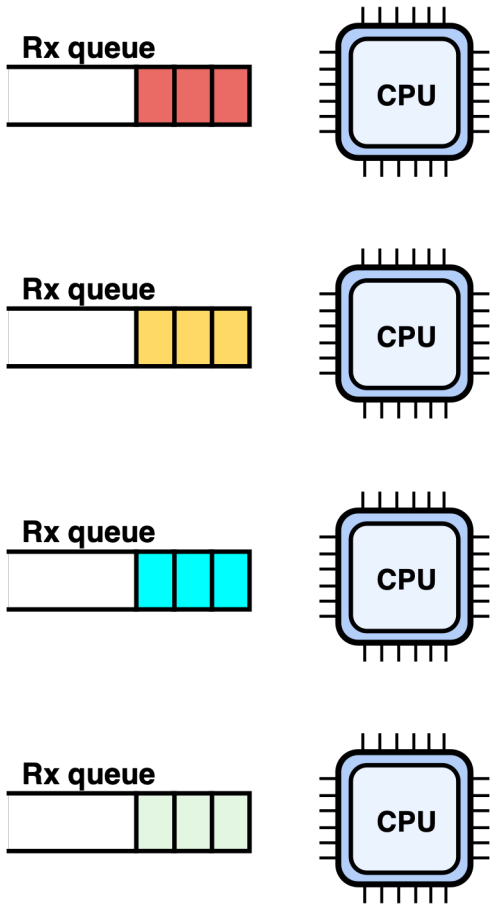
Single-digit Microsecond Latency

Userspace Network Runtime

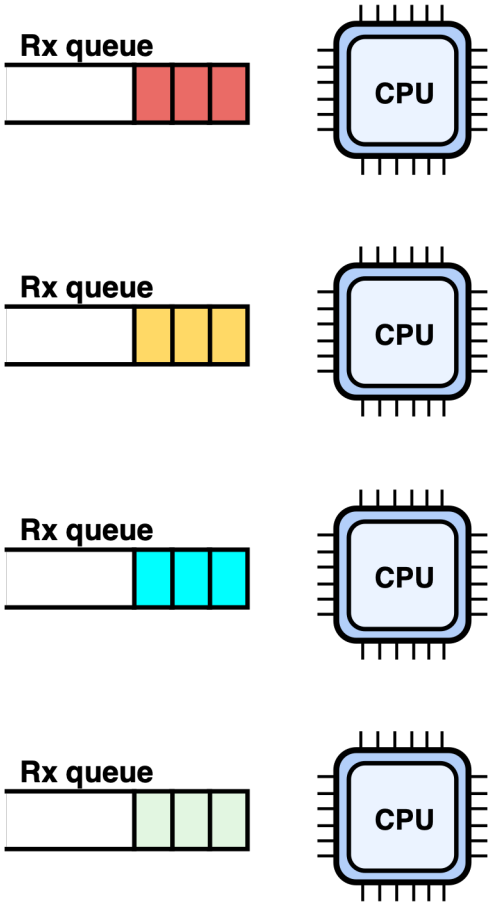


Userspace Network Runtime

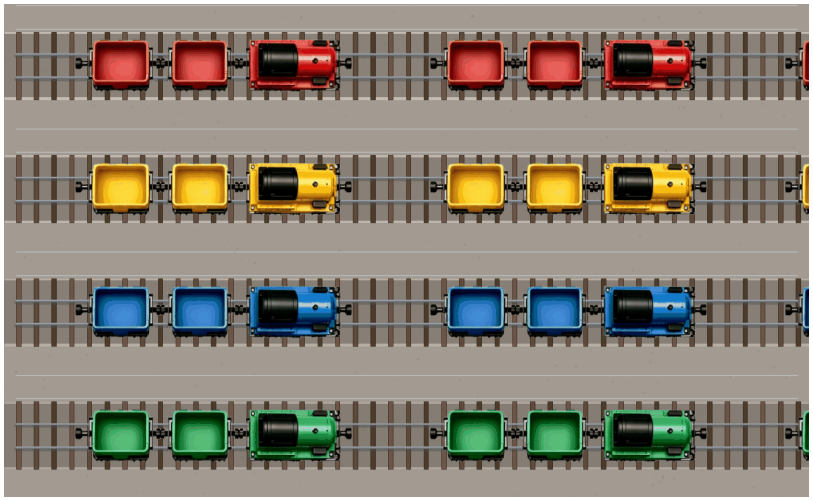




Analogy: Railway System

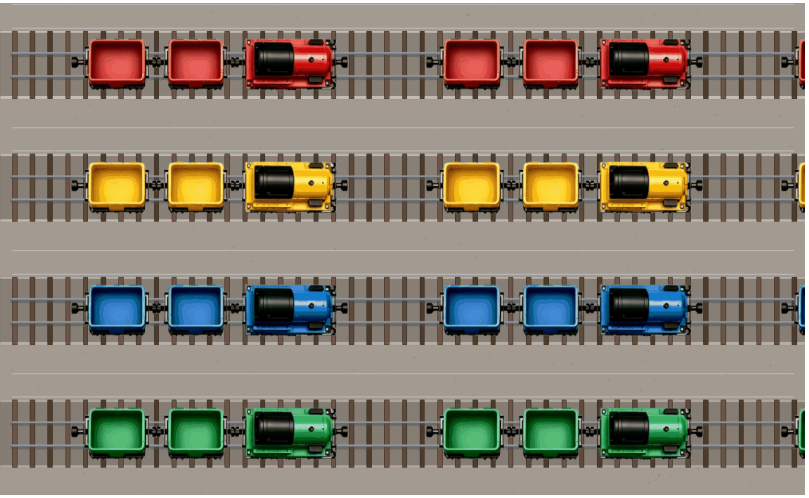


Similar

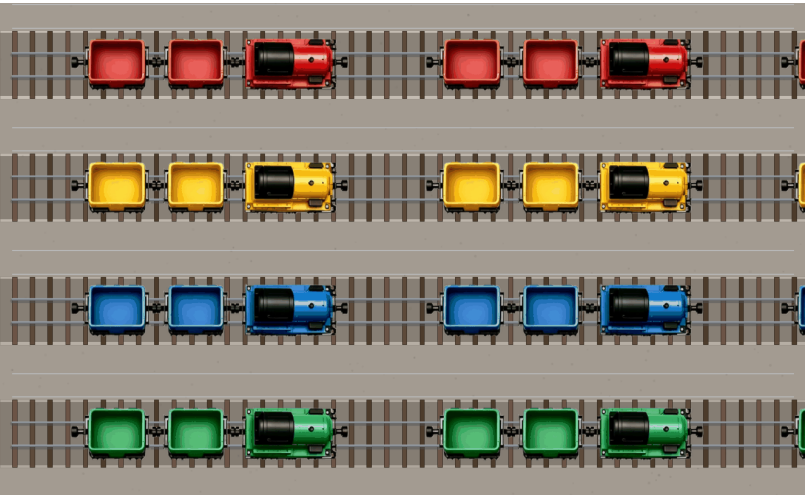


Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

Analogy: Railway System



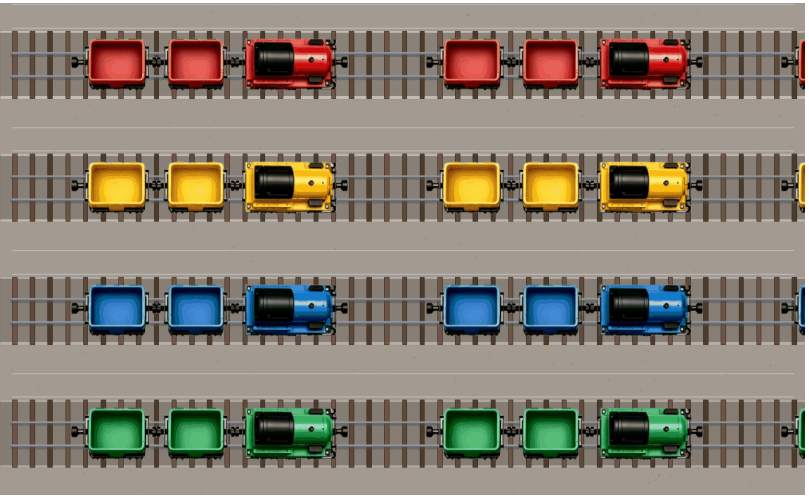
Analogy: Railway System

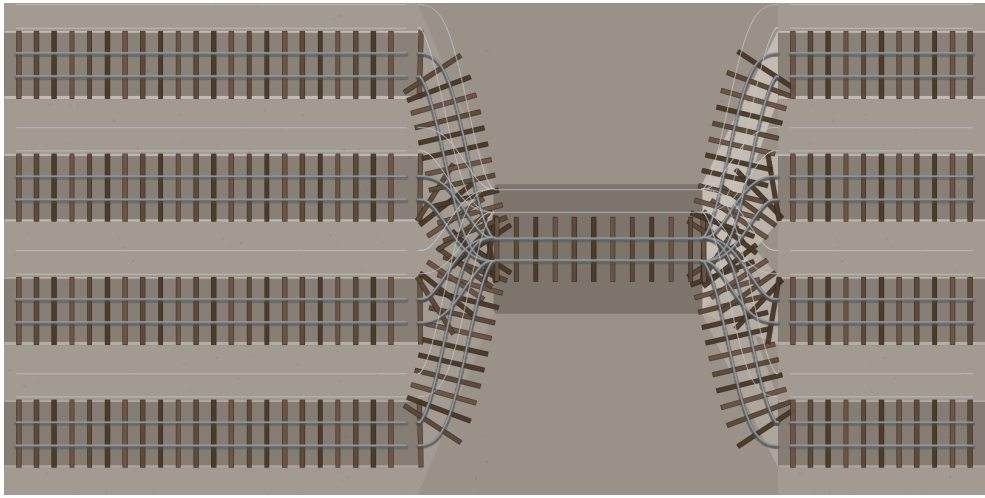


Scale



Analogy: Railway System

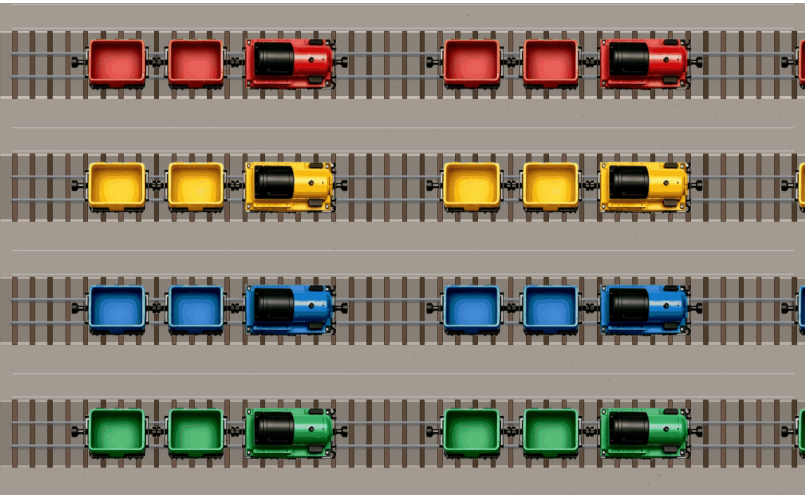




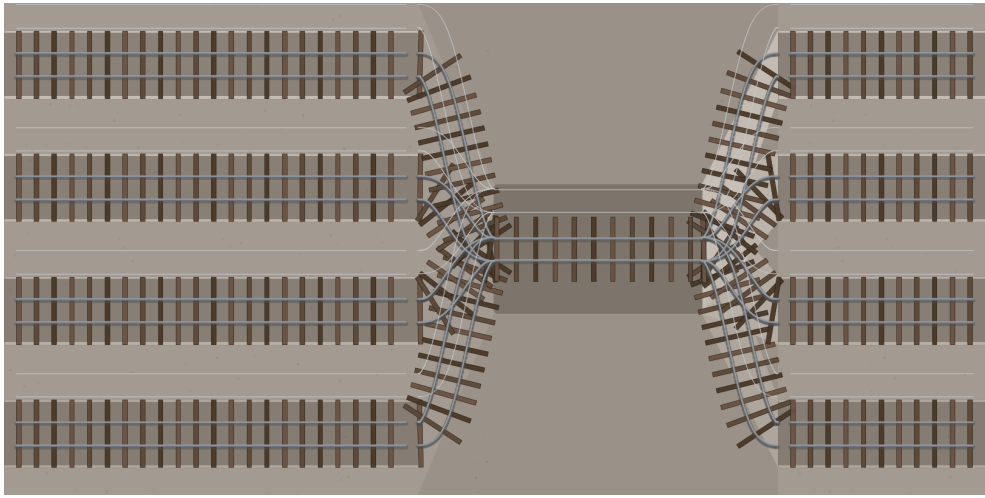
Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

Analogy: Railway System

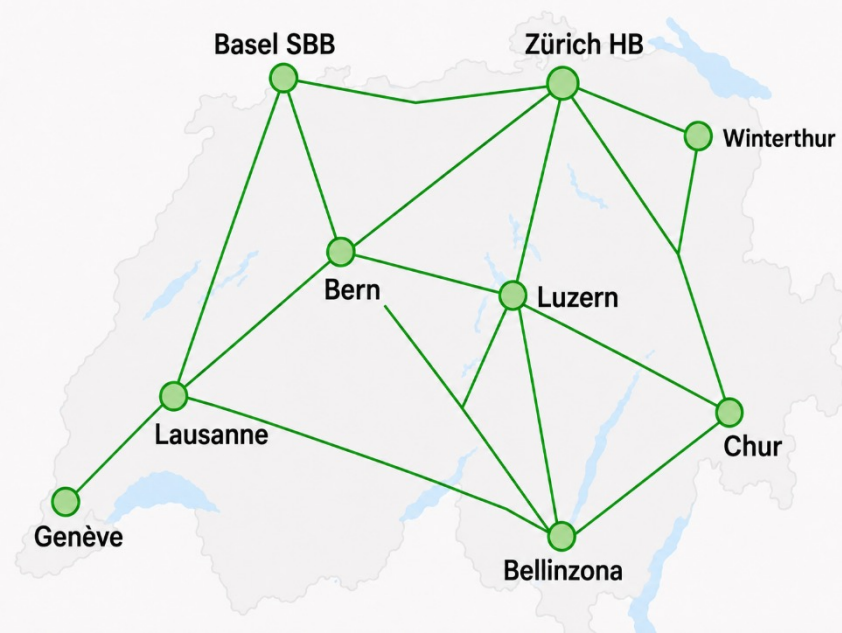
Decentralized



Centralized



Decentralized



Centralized



SBB, Our work

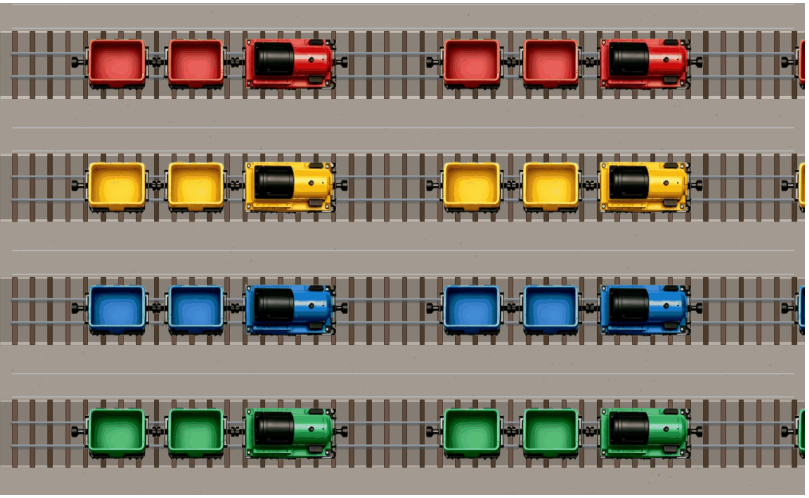
Shinjuku, NSDI 2019

Picture credit: ChatGPT

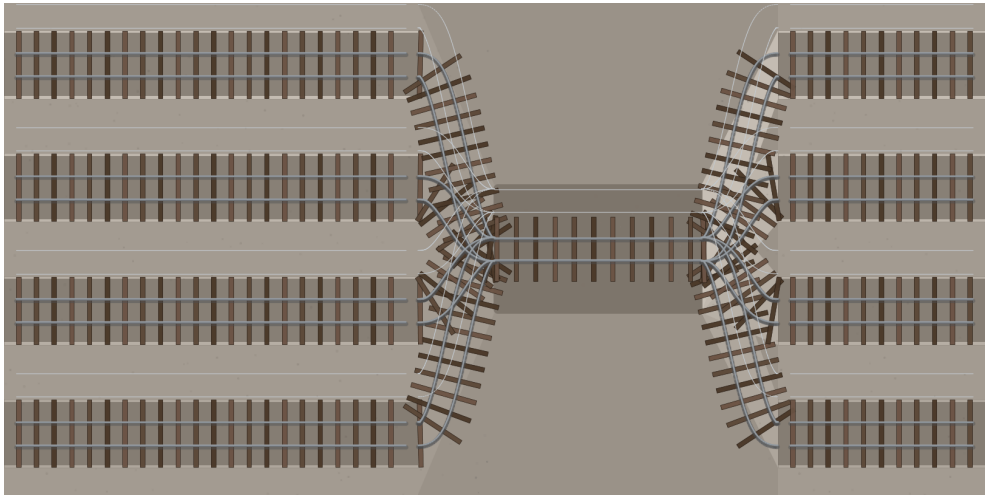
[1]Shinjuku (新宿駅) is a major train station in Tokyo that serves millions traveling on 12 lines of various types and speeds.

Analogy: Railway System

Decentralized



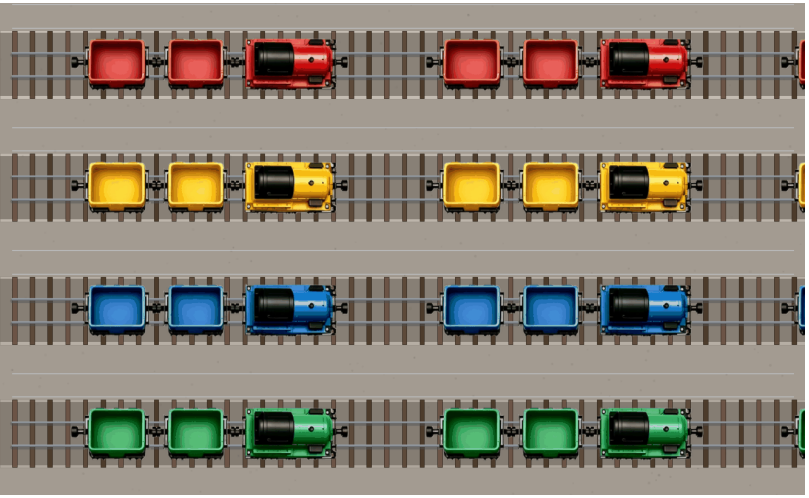
Centralized



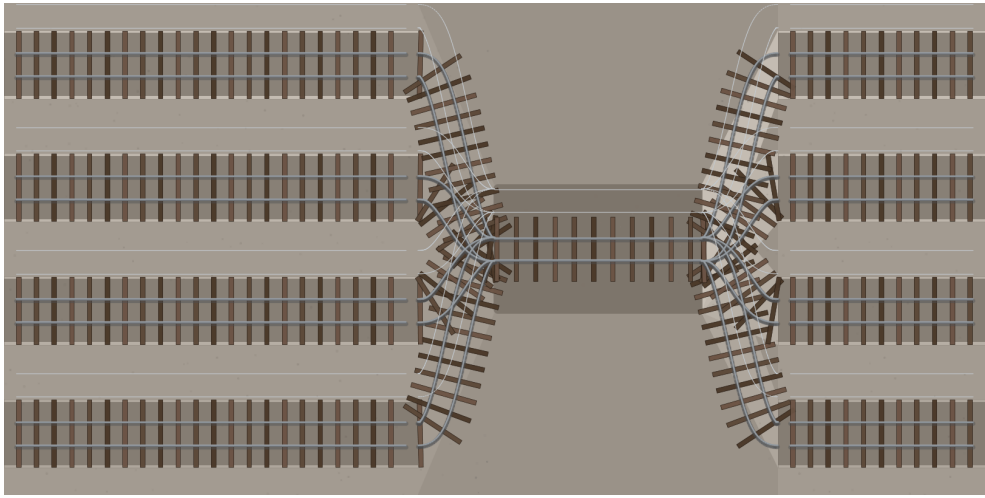
Question: What is the disadvantage of centralized railway system?

Analogy: Railway System

Decentralized



Centralized



Question: What is the disadvantage of centralized railway system?

scalability!

Analogy: Railway System

Conclusion:
Centralized network runtimes may have scalability bottlenecks.

Analogy: Railway System

Conclusion:
Centralized network runtimes may have scalability bottlenecks.

Reality:
Almost all current userspace network runtimes are centralized.

Analogy: Railway System

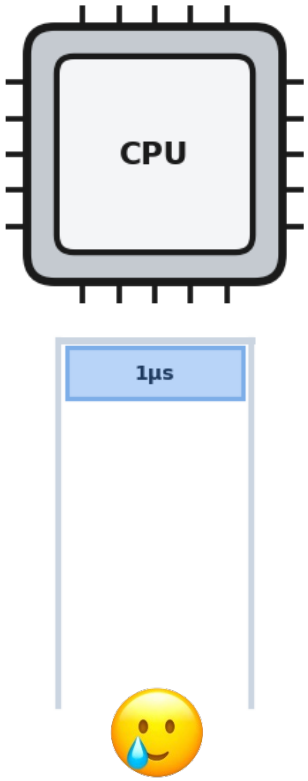
Conclusion:
Centralized network runtimes may have scalability bottlenecks.

Reality:
Almost all current userspace network runtimes are centralized.

Why?
There exists three types of scheduling challenges.

Scheduling #1: Request Preemption

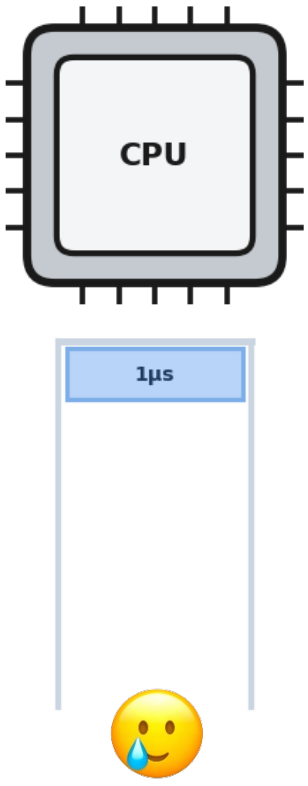
Request preemption is used to avoid head-of-line blocking.

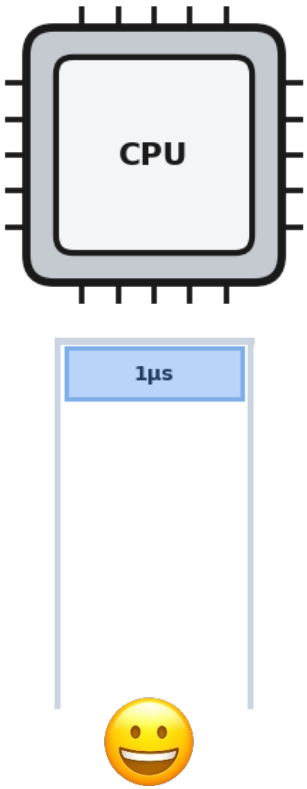


Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

Scheduling #1: Request Preemption

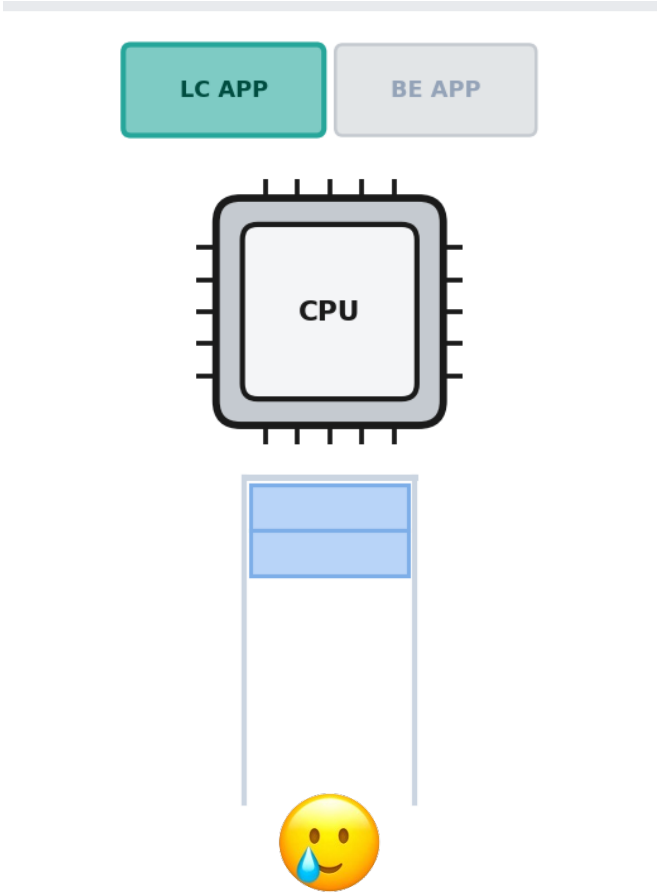
Request preemption is used to avoid head-of-line blocking.





Scheduling #2: CPU Allocation

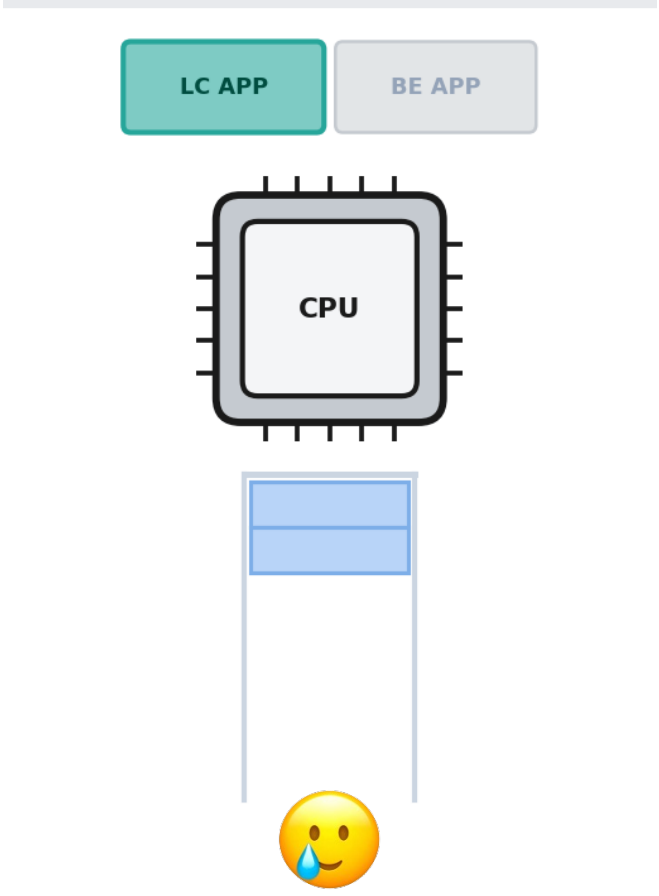
CPU allocation wakes up LC promptly when multi-task co-location.

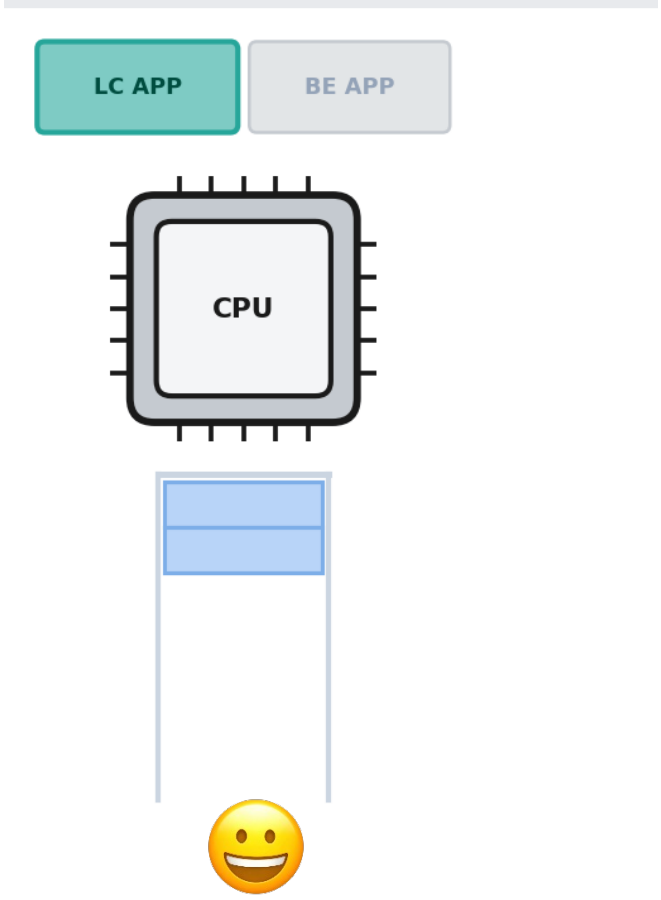


Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

Scheduling #2: CPU Allocation

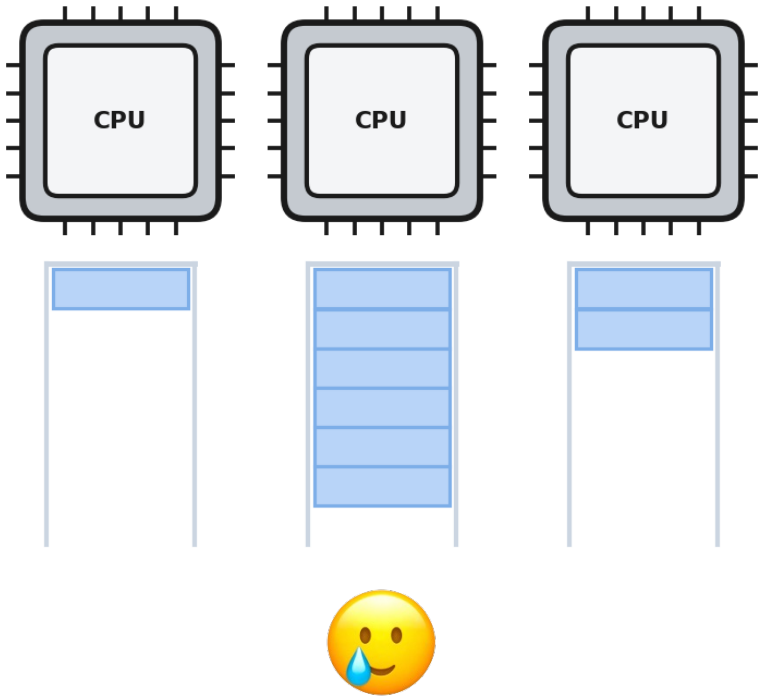
CPU allocation wakes up LC promptly when multi-task co-location.





Scheduling #3: Request Load Balancing

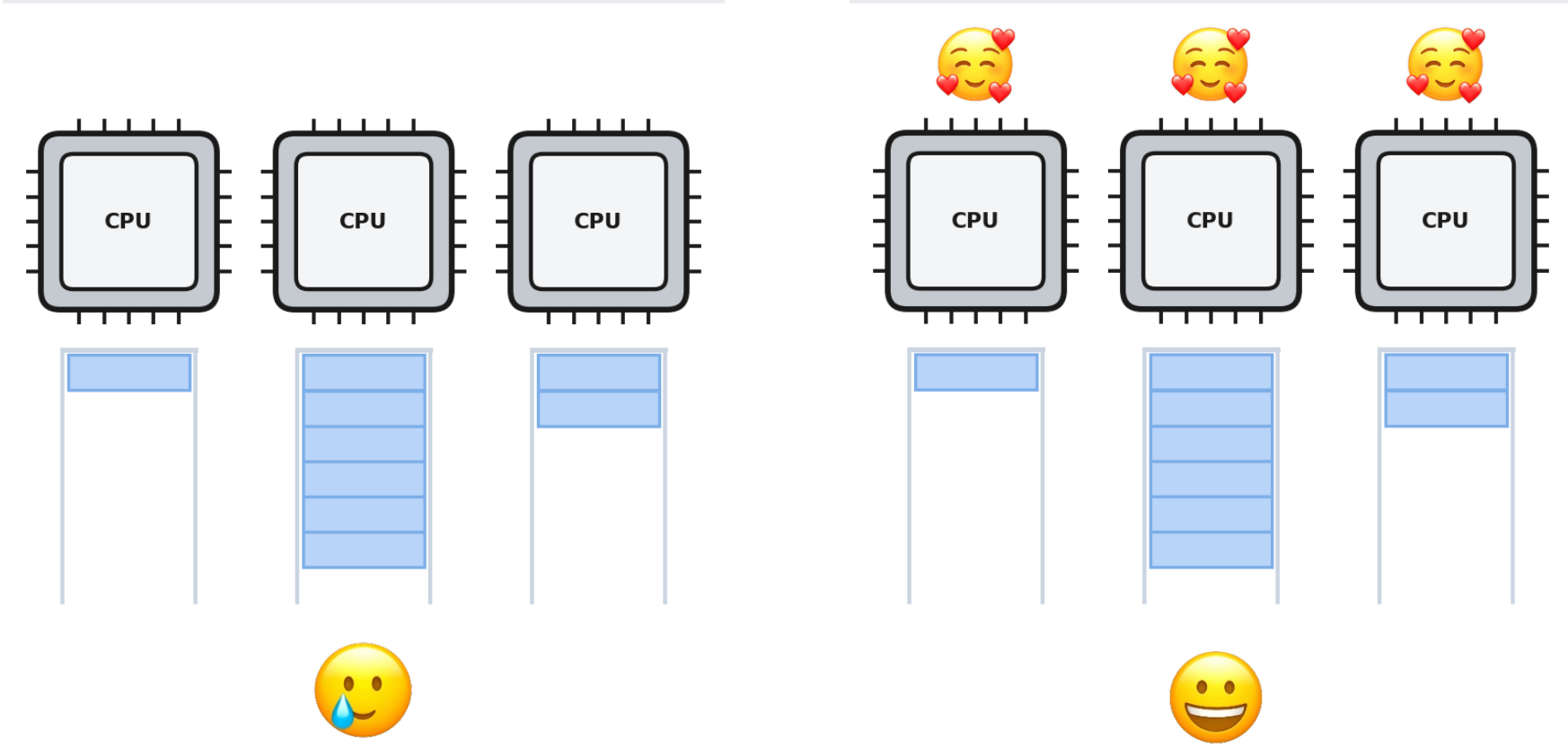
Request load balancing mitigates imbalance among workers.



Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

Scheduling #3: Request Load Balancing

Request load balancing mitigates imbalance among workers.



Prior Work's Solutions

	Request load balancing	Request preemption	CPU allocation
IX, ZygOS, Arrakis	✓	✗	✗
Shinjuku, Concord	✓✗	✓✗	✗
Perséphone	✓✗	✗	✗
TQ	✓✗	✓	✗
Libpreemptible	✓✗	✓✗	✗
Aspen-KB	✓	✓✗	✗
Shenango	✓✗	✗	✓✗
Caladan	✓	✗	✓✗
Junction, Vessel	✓	✗	✓✗
Skyloft	✓✗	✓	✓✗

✗: Not supported.

✓✗: Supported with a centralized entity.

✓: Supported with decentralized design.

Prior Work's Solutions

	Request load balancing	Request preemption	CPU allocation
IX, ZygOS, Arrakis	✓	✗	✗
Shinjuku, Concord	✓✗	✓✗	✗
Perséphone	✓✗	✗	✗
TQ	✓✗	✓	✗
Libpreemptible	✓✗	✓✗	✗
Aspen-KB	✓	✓✗	✗
Shenango	✓✗	✗	✓✗
Caladan	✓	✗	✓✗
Junction, Vessel	✓	✗	✓✗
Skyloft	✓✗	✓	✓✗

✗: Not supported.

✓✗: Supported with a centralized entity.

✓: Supported with decentralized design.

Most prior systems do not support all three scheduling functions.

Prior Work's Solutions

	Request load balancing	Request preemption	CPU allocation
IX, ZygOS, Arrakis	✓	✗	✗
Shinjuku, Concord	✓✗	✓✗	✗
Perséphone	✓✗	✗	✗
TQ	✓✗	✓	✗
Libpreemptible	✓✗	✓✗	✗
Aspen-KB	✓	✓✗	✗
Shenango	✓✗	✗	✓✗
Caladan	✓	✗	✓✗
Junction, Vessel	✓	✗	✓✗
Skyloft	✓✗	✓	✓✗

✗: Not supported.

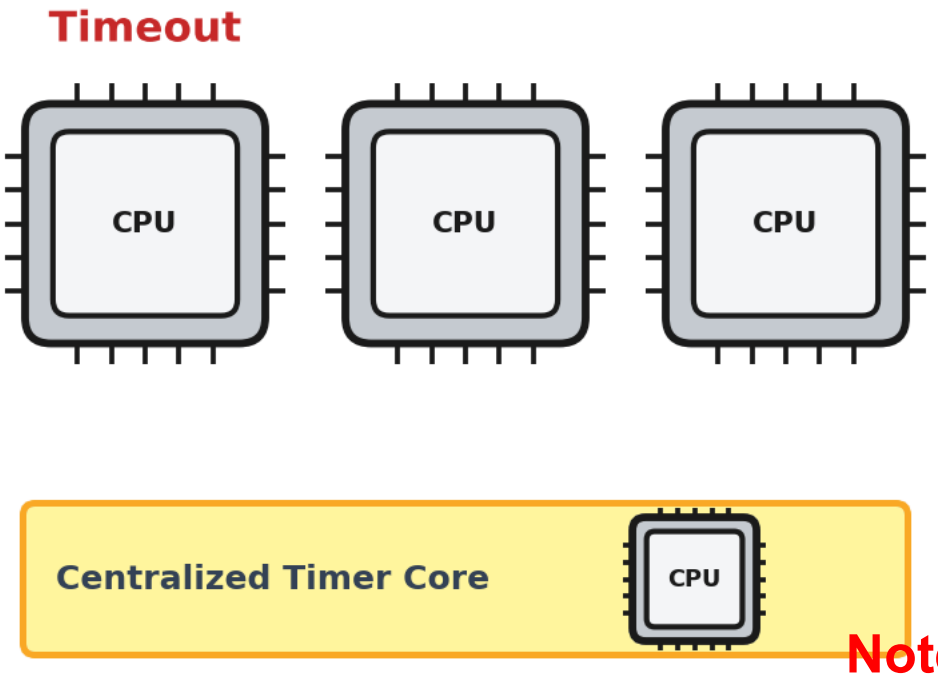
✓✗: Supported with a centralized entity.

✓: Supported with decentralized design.

Most prior systems do not support all three scheduling functions.

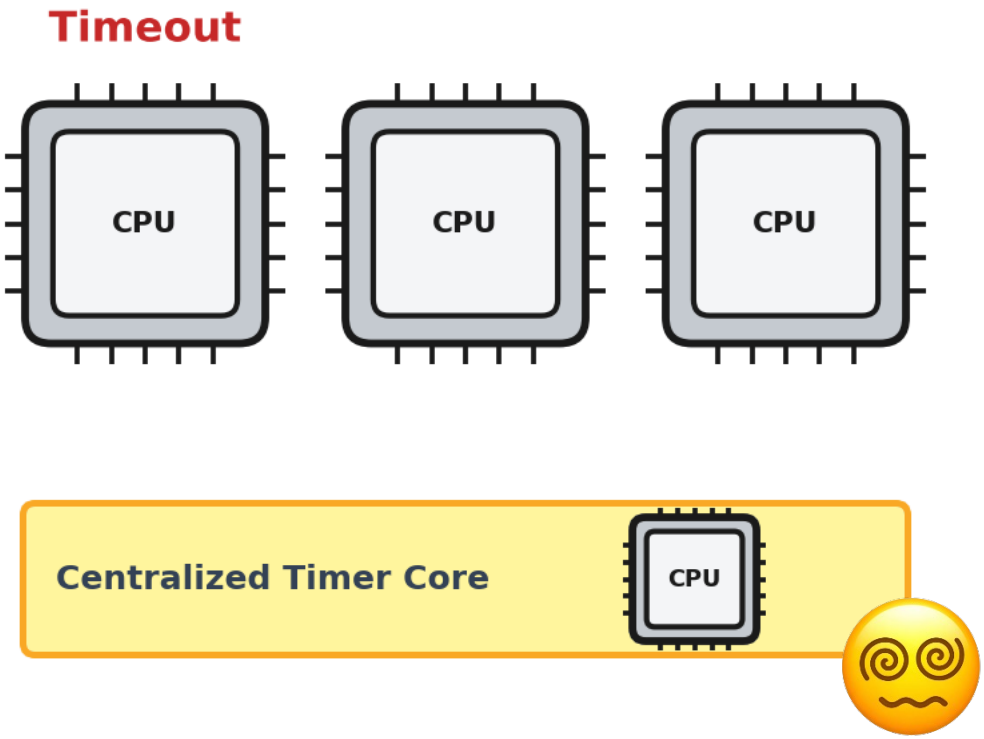
All prior work uses centralized entities for at least one scheduling.

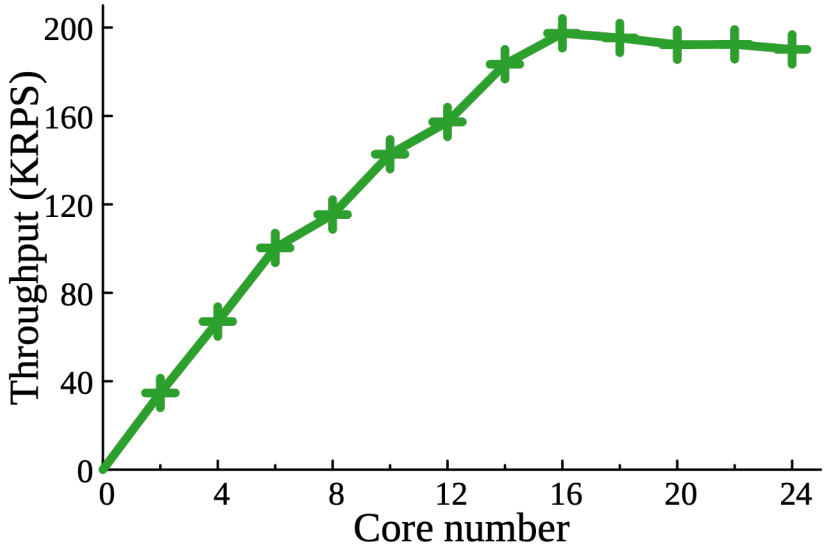
Centralized Component #1: Timer



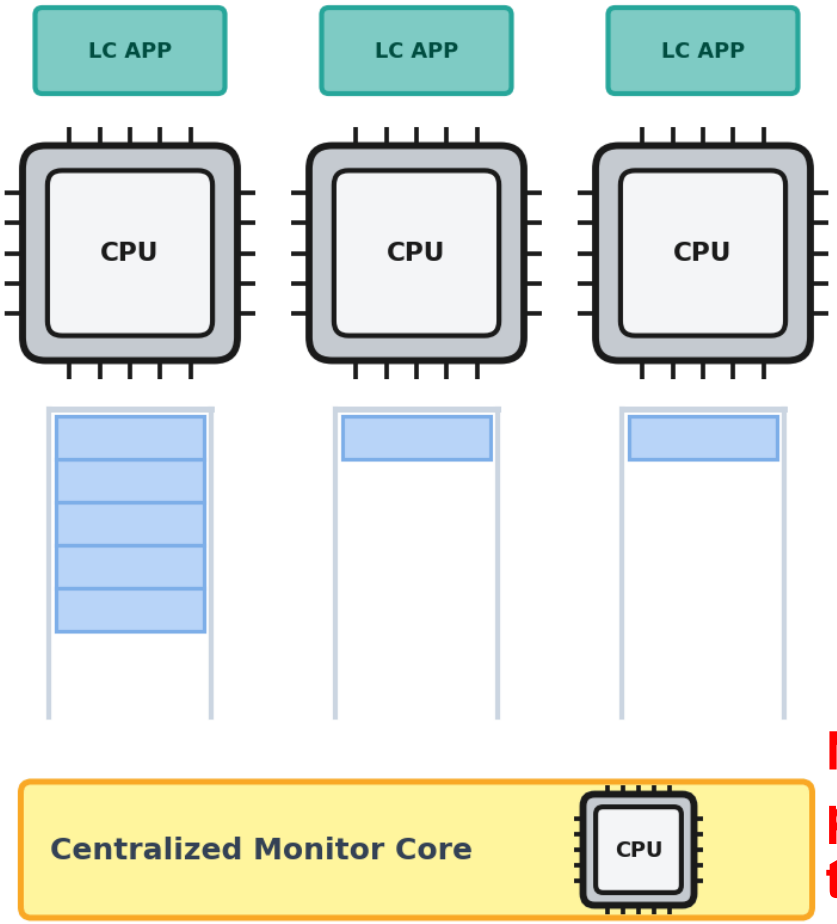
Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

Centralized Component #1: Timer



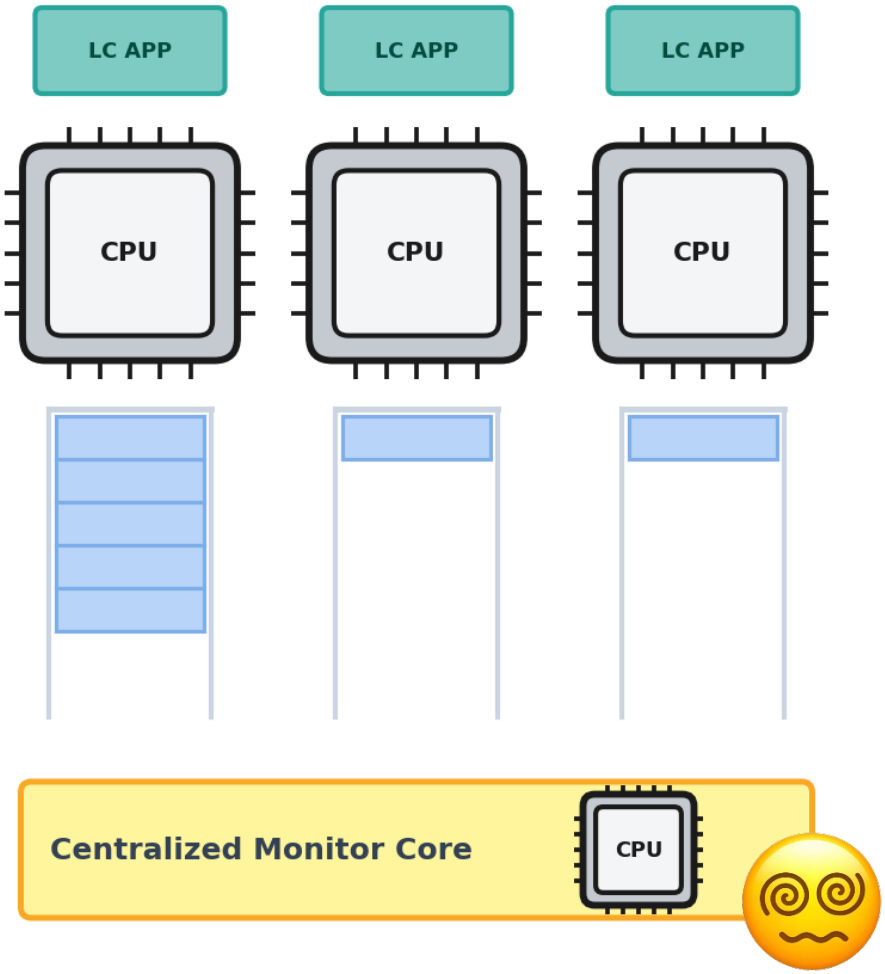


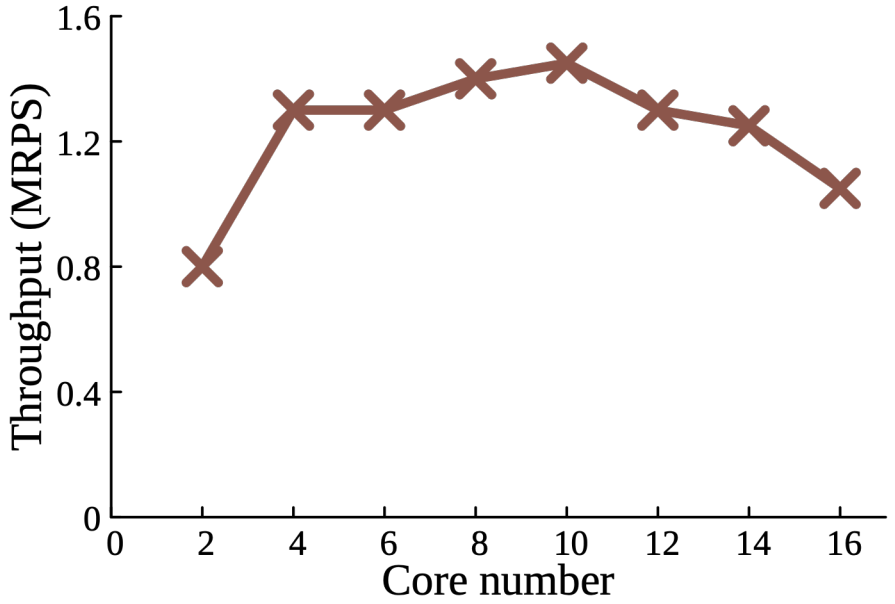
Centralized Component #2: Monitor



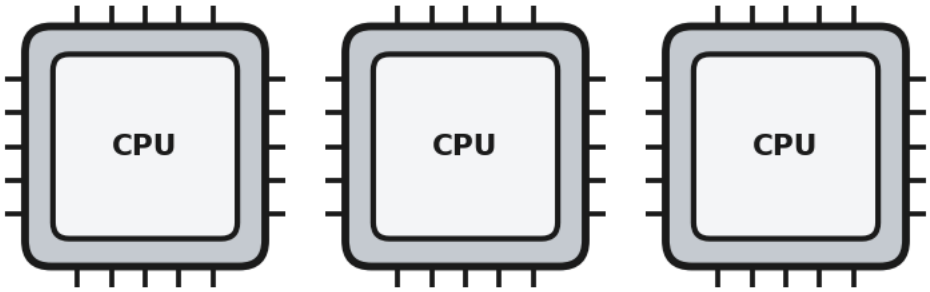
Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

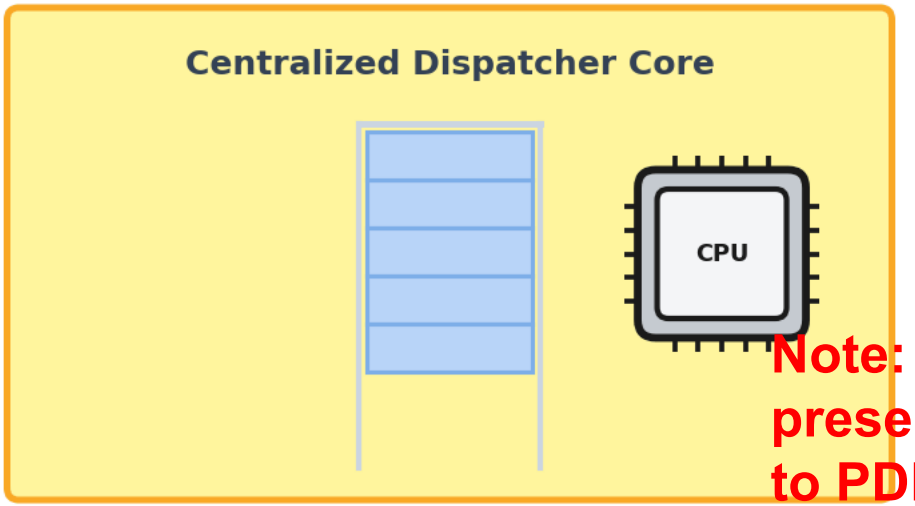
Centralized Component #2: Monitor





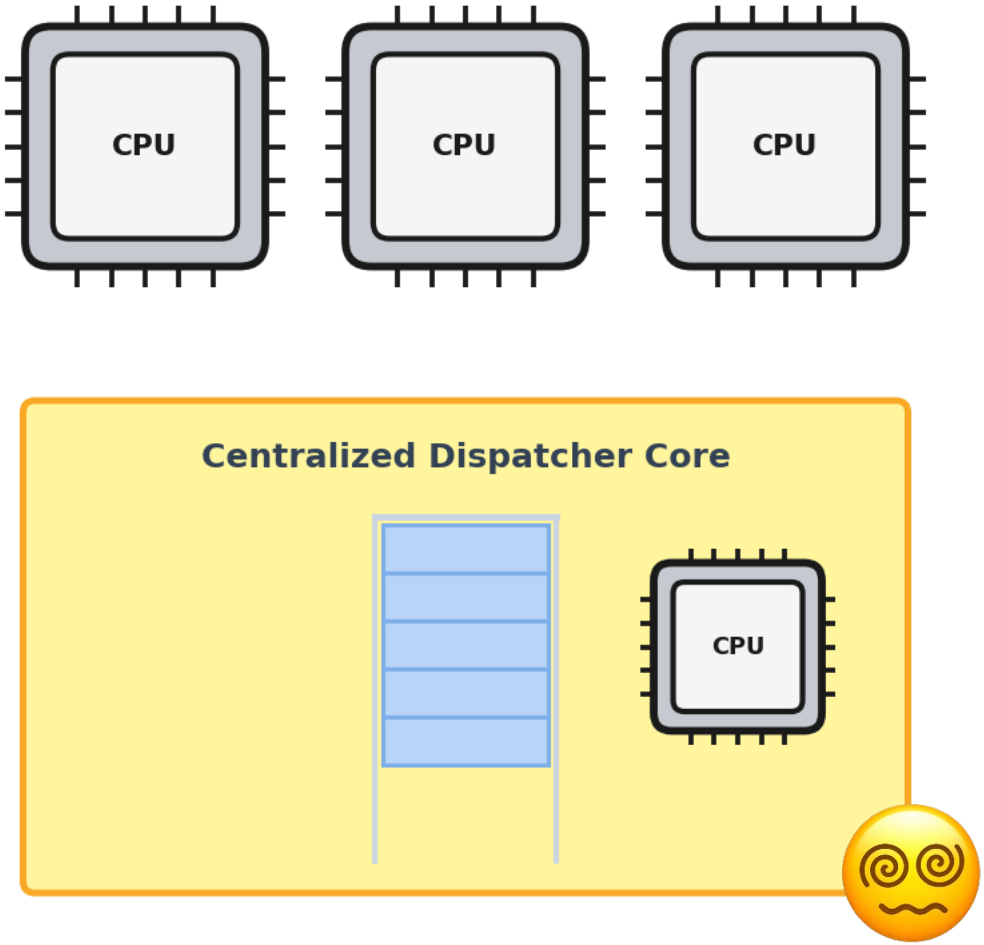
Centralized Component #3: Dispatcher

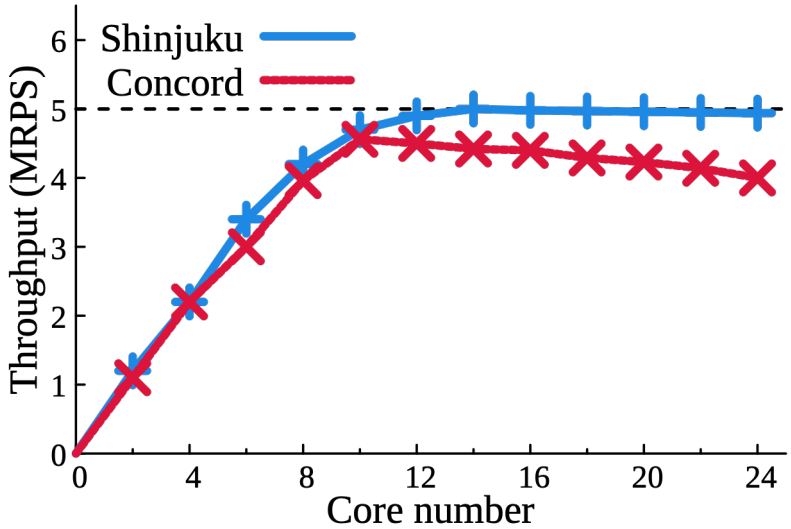




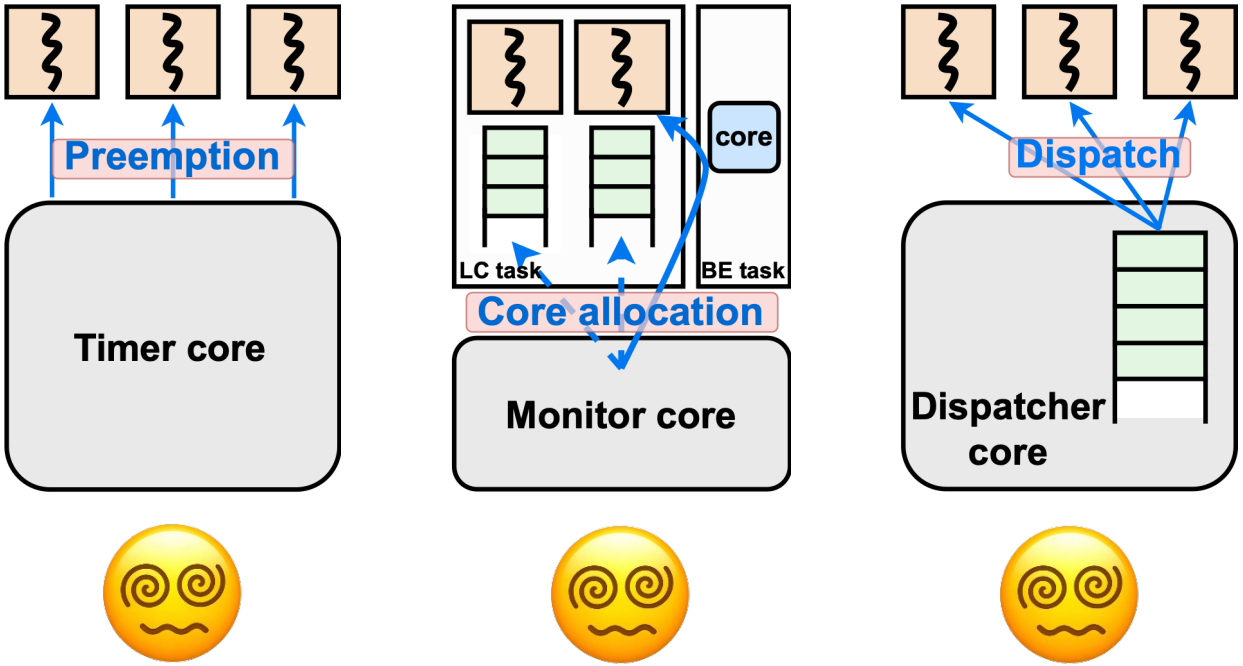
Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

Centralized Component #3: Dispatcher



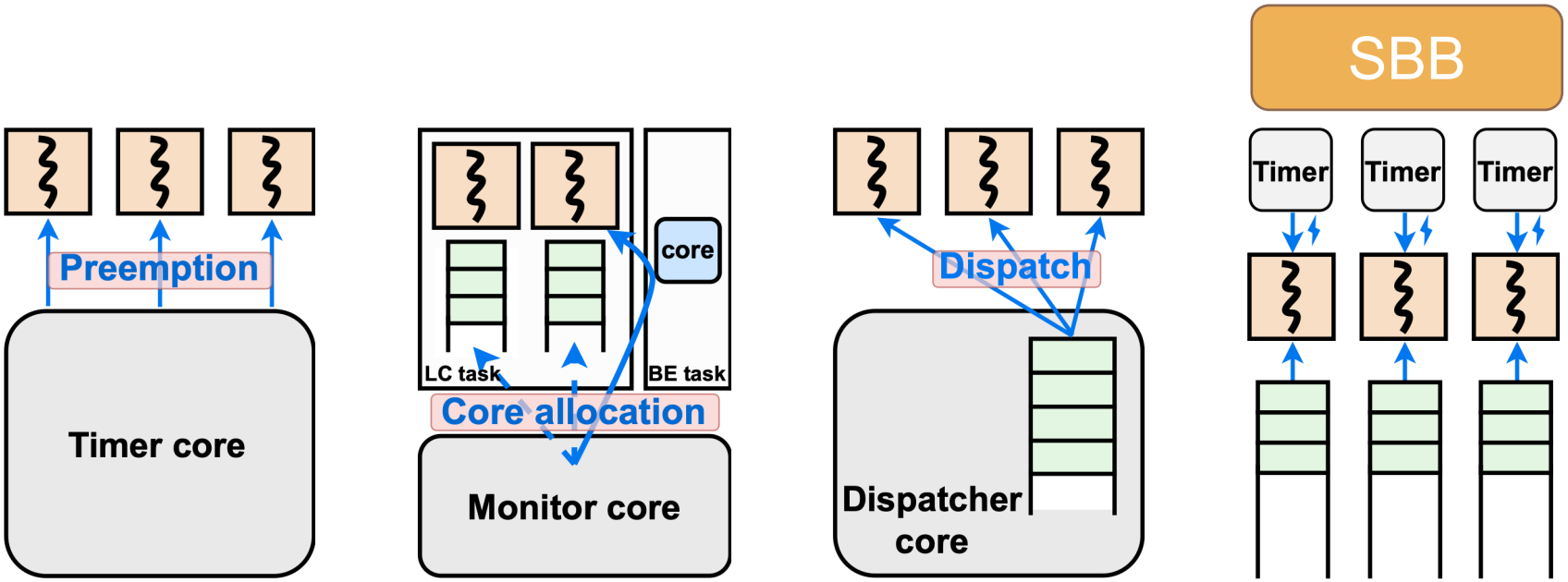


Centralized Components



Centralized components will become the scalability bottlenecks.

Design Principle



Centralized components will become the scalability bottlenecks.

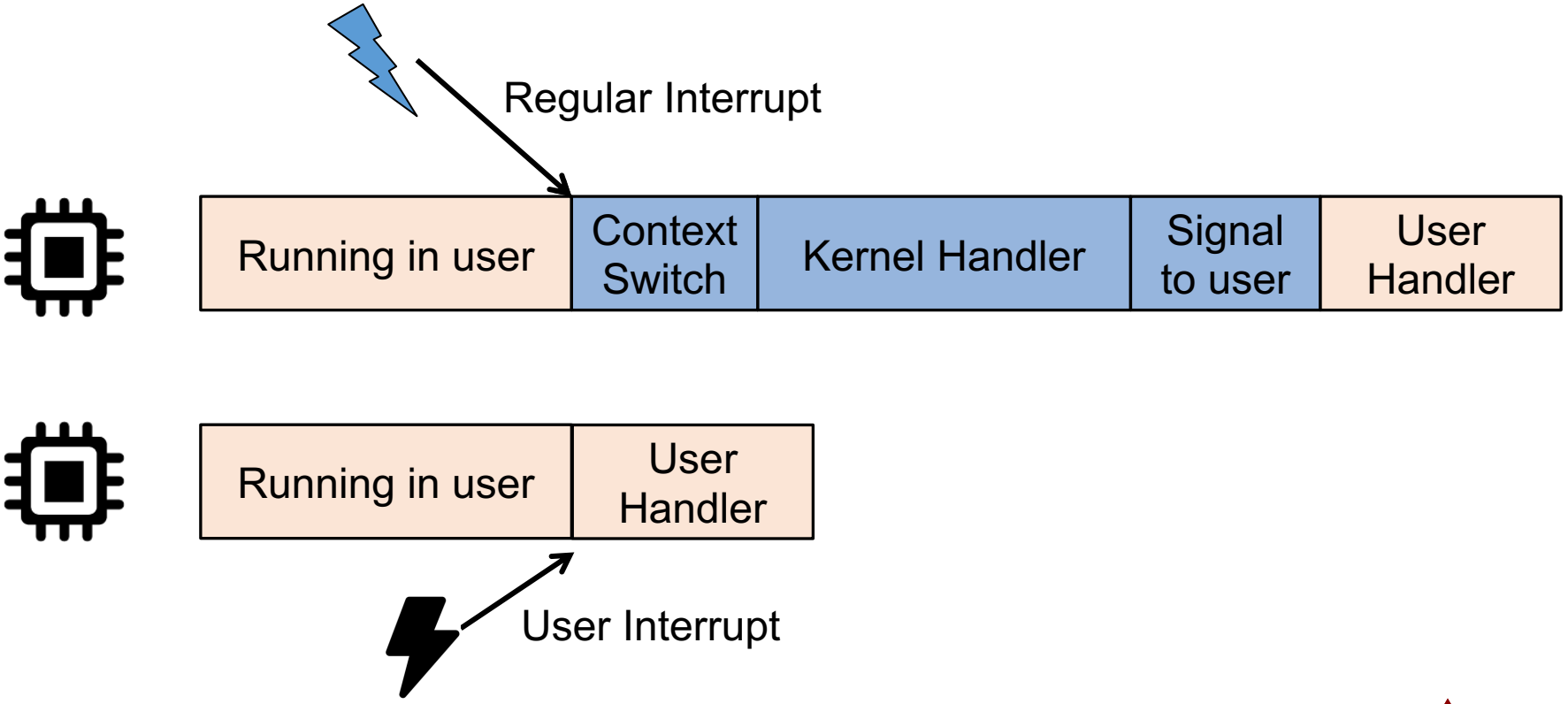
SBB's Design Principle: Build a decentralized architecture

Design Overview

Three Challenges	Three Solutions	
Request Preemption	Timer User Interrupt	
		Mechanism
CPU Allocation	NIC User Interrupt	
Request Load Balancing	Two-Level Balancing	Policy

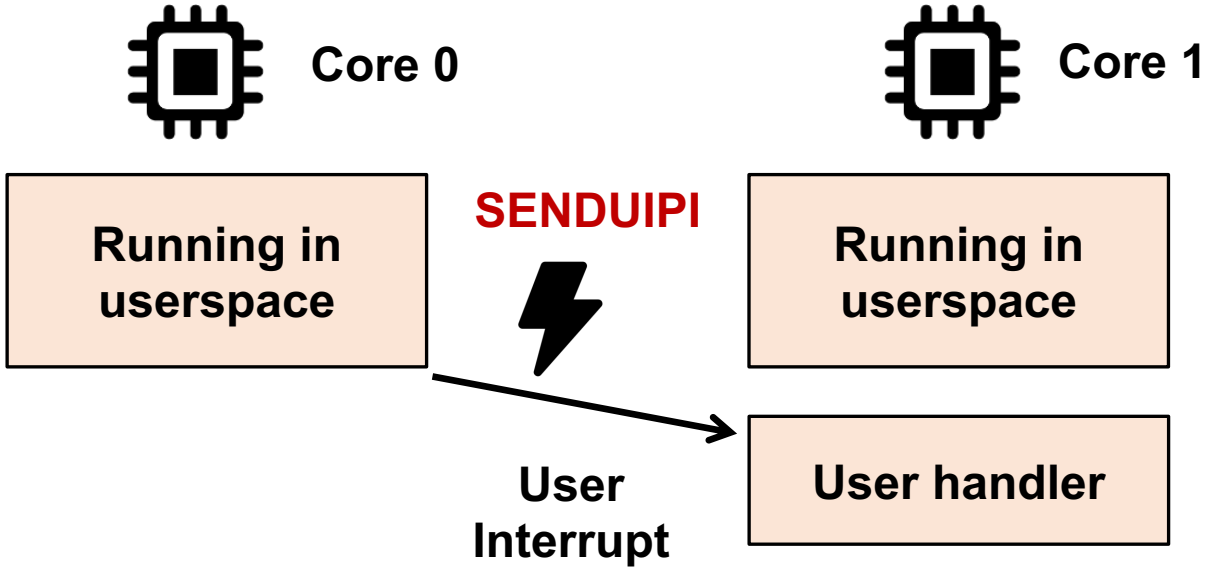
User Interrupt

User Interrupt: New hardware feature
(from Intel Sapphire Rapids, 2023)



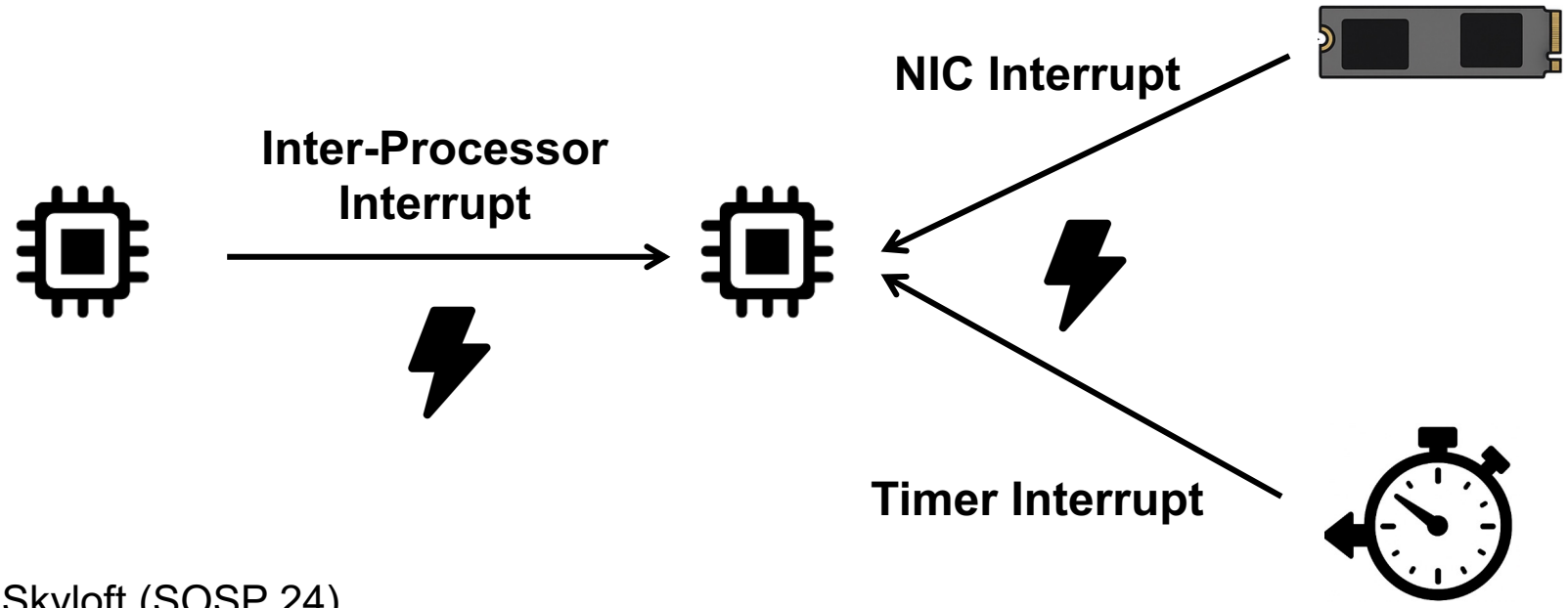
User Interrupt

Normal use: Inter-Processor Interrupt



User Interrupt

New use: Device Interrupt (NIC, LAPIC timer, SSD, etc.) [1,2]



[1] Skyloft (SOSP 24)
[2] Aeolia (SOSP 25)

SBB's Request Preemption

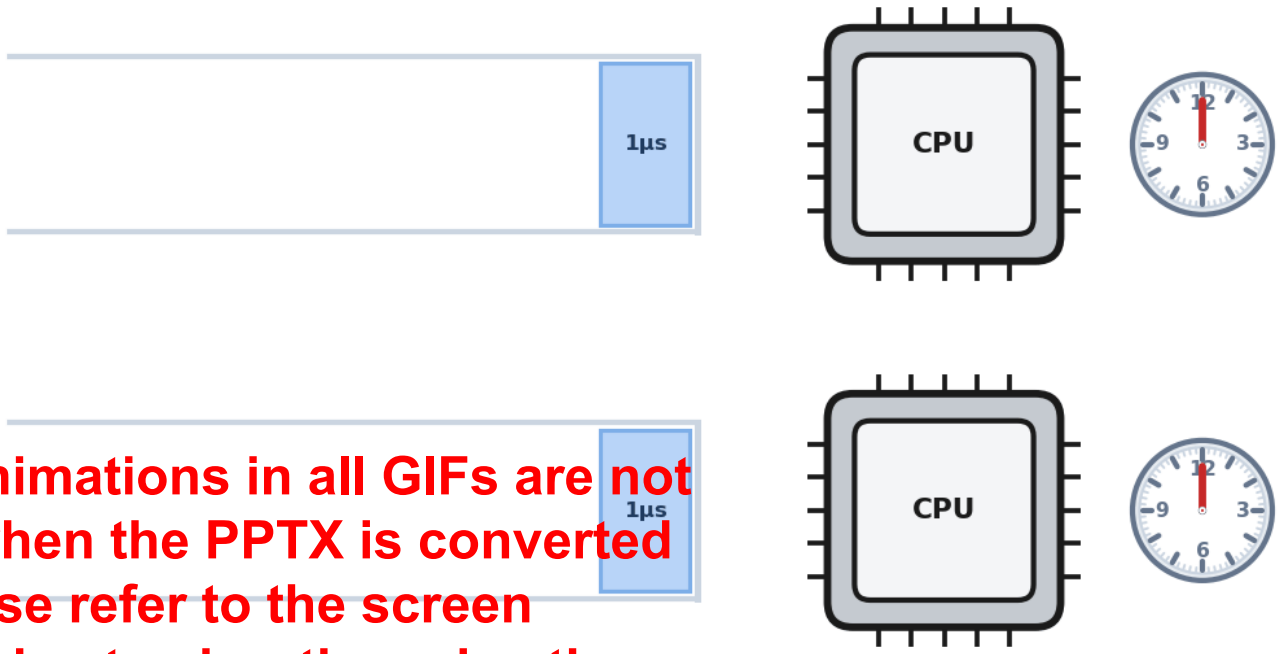
LAPIC timer:

- Per-CPU hardware
- Supports various timing modes
- Triggers timer interrupt

SBB's Request Preemption

The worker arms a one-shot timer interrupt for each request:

- Short request: no timer interrupt
- Long request: timer interrupt triggers requeue



Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

SBB's Request Preemption

The worker arms a one-shot timer interrupt for each request:

- Short request: no timer interrupt
- Long request: timer interrupt triggers requeue

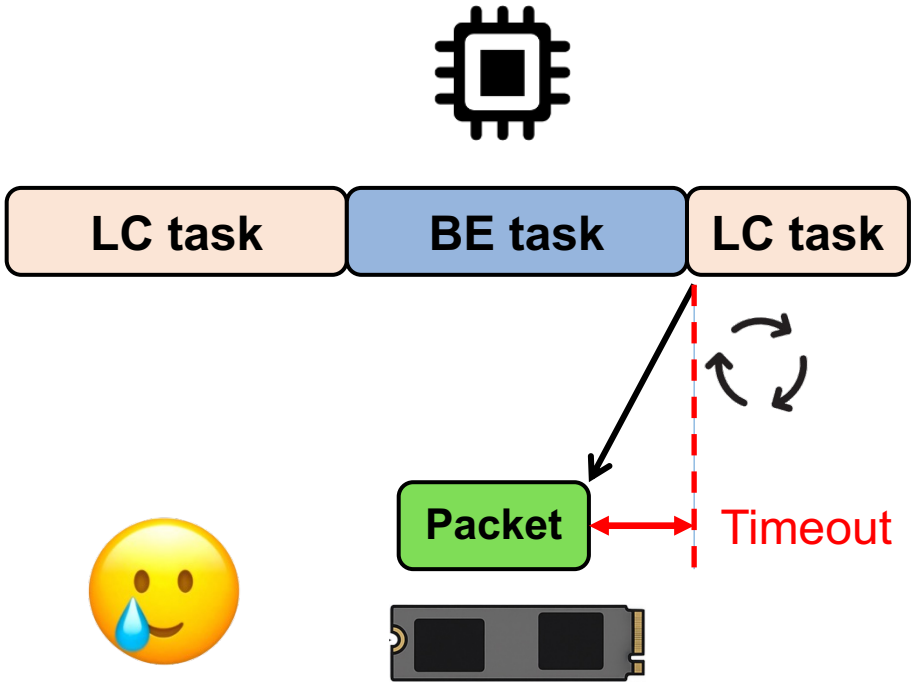
Preemption Mechanism	**System**	**Latency (cycles)**
Signal-based timer	Linux	7697
Custom IPI	Shinjuku	1878
User IPI	Aspen-KB	1381
Timer User Interrupt	SBB	688

SBB's CPU Allocation

Observation: inherent conflict between polling and co-location

SBB's CPU Allocation

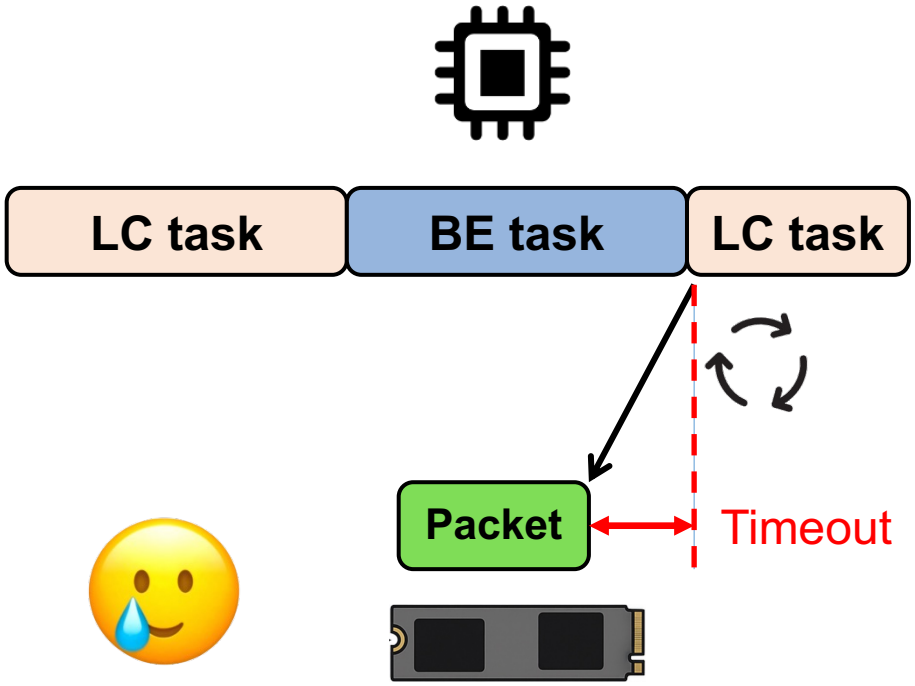
Observation: inherent conflict between polling and co-location

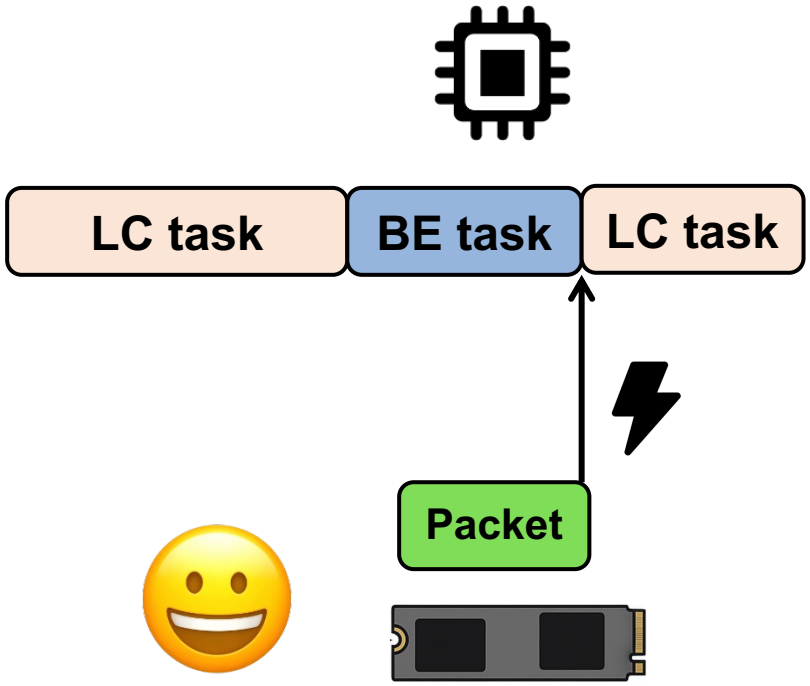


Polling: De-scheduled LC tasks cannot detect incoming packets

SBB's CPU Allocation

Observation: inherent conflict between polling and co-location





Polling: De-scheduled LC tasks cannot detect incoming packets

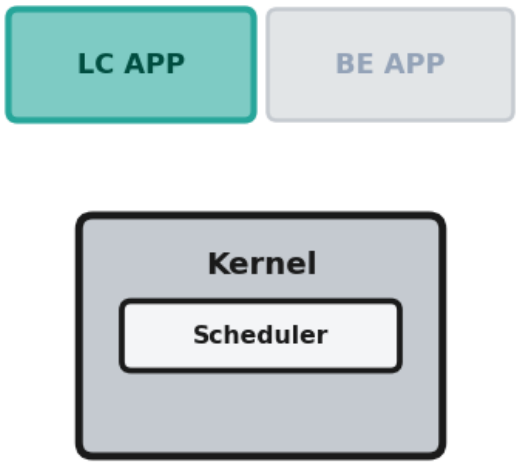
Interrupt: De-scheduled LC task gets notification for new packets

SBB's CPU Allocation

When a new packet arrives:

- LC is running: the worker thread will handle it directly in userspace

- BE is running: the interrupt will trigger the kernel scheduler, which will reschedule the LC task thread immediately

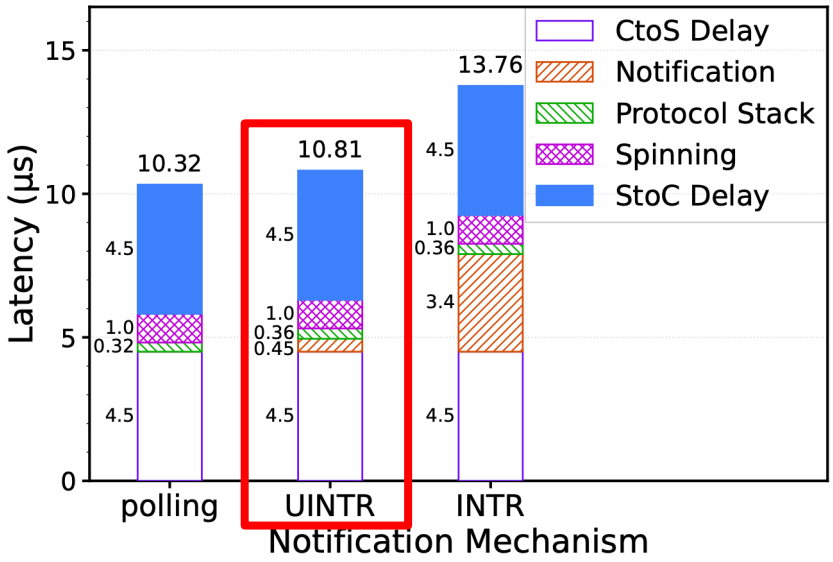


Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

SBB's CPU Allocation

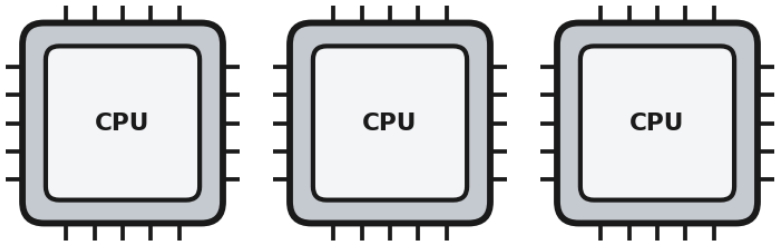
When a new packet arrives:

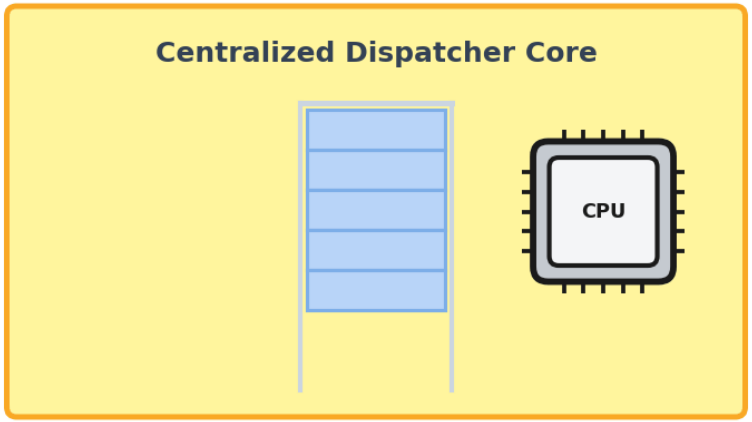
- LC is running: the worker thread will handle it directly in userspace
- BE is running: the interrupt will trigger the kernel scheduler, which will reschedule the LC task thread immediately



SBB's Request Load Balancing

Centralized: Dispatcher Core

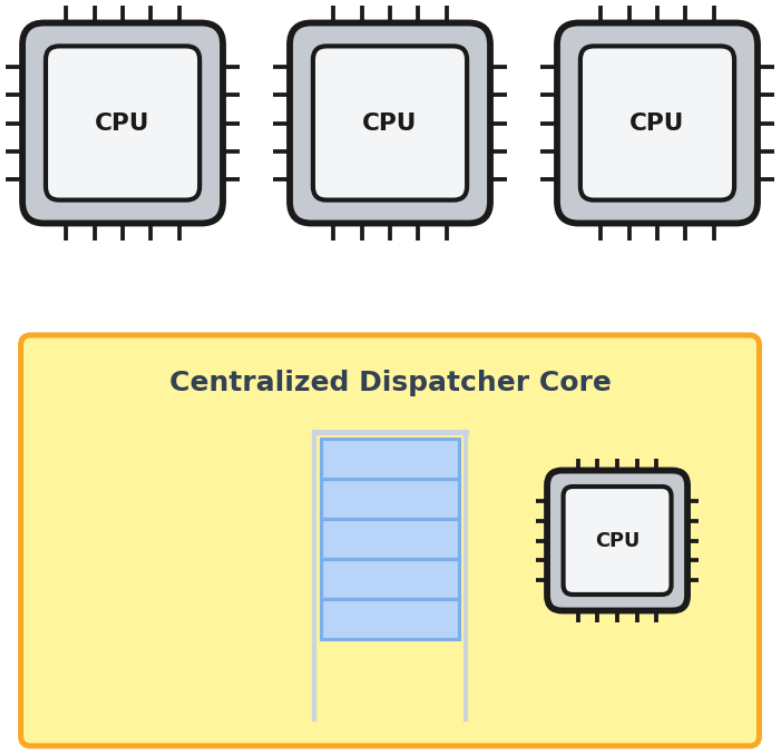




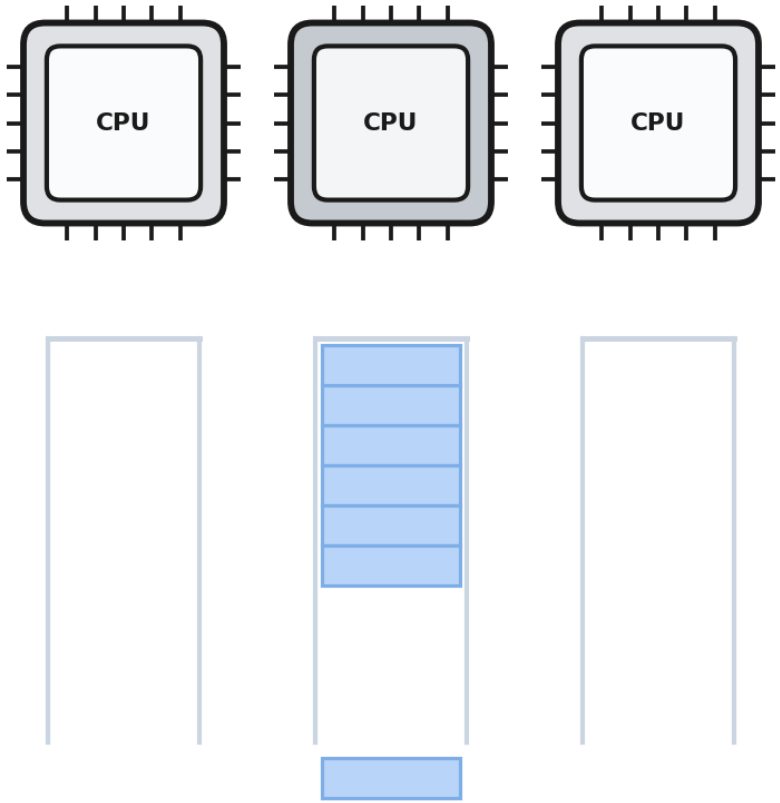
Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

SBB's Request Load Balancing

Centralized: Dispatcher Core

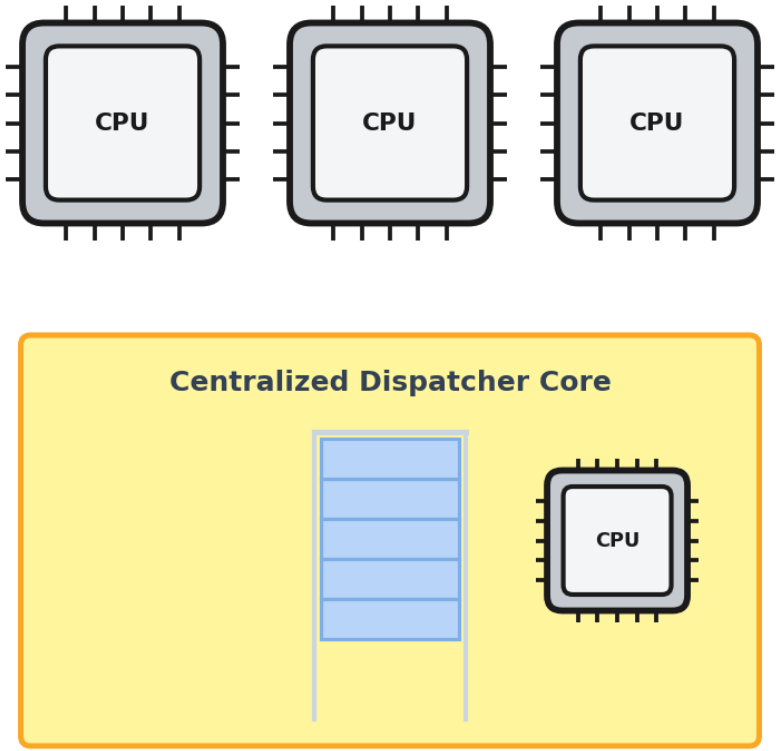


Decentralized: Task Stealing

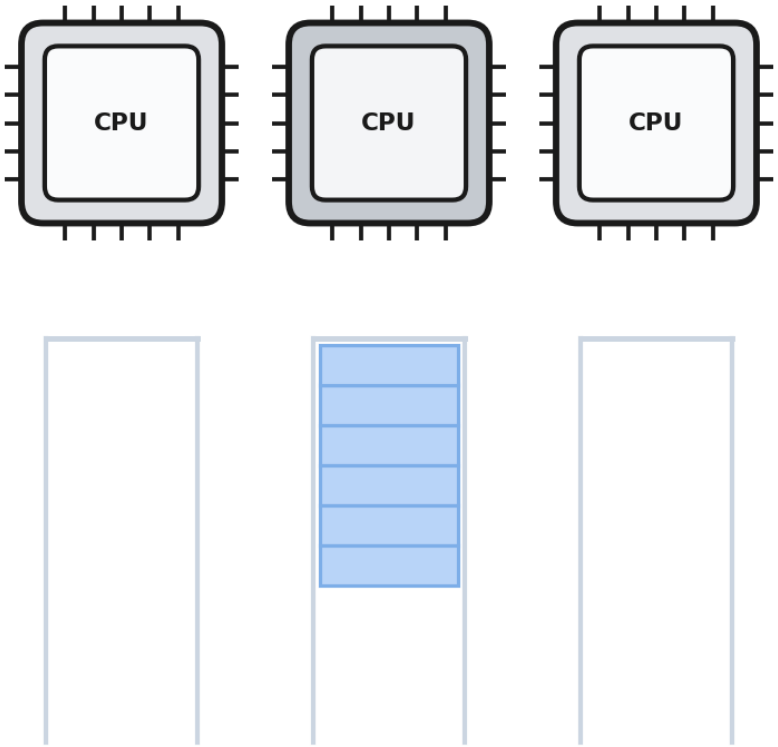


SBB's Request Load Balancing

Centralized: Dispatcher Core



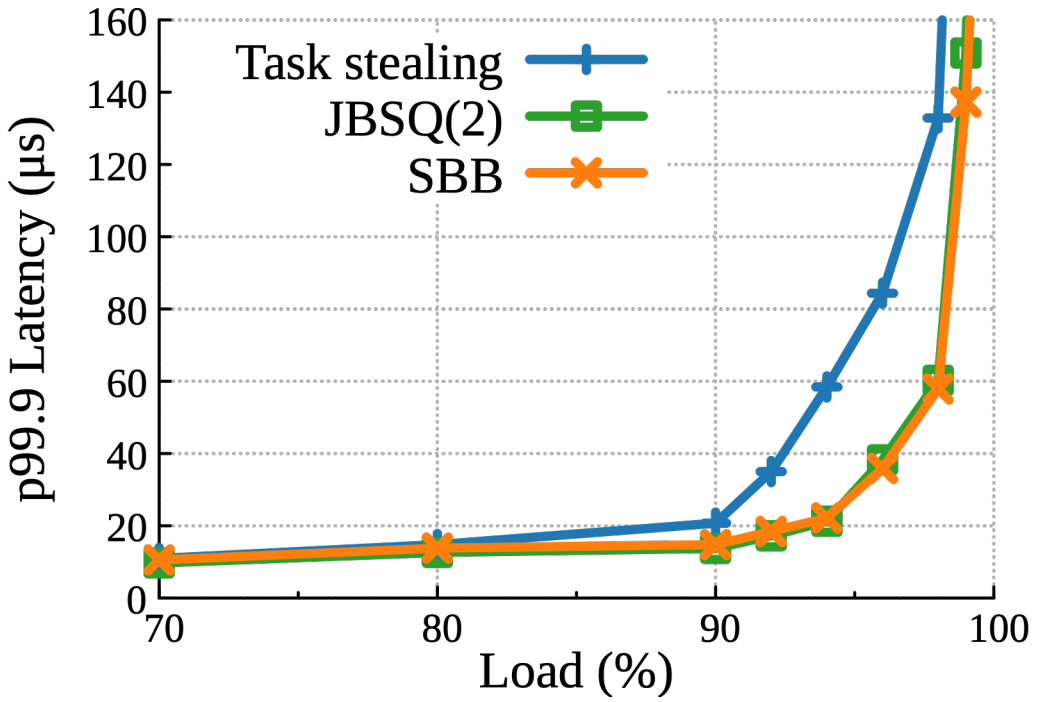
Decentralized: Task Stealing



Conventional wisdom: Decentralized load balancing performs inherently worse.

SBB's Request Load Balancing

Observation:
Existing policies fail to consider both causes of imbalance.



- **Task stealing**: prior work's decentralized approach
- **JBSQ(2)**: prior work's centralized approach
- **SBB**: Our two-level decentralized approach

SBB's Request Load Balancing

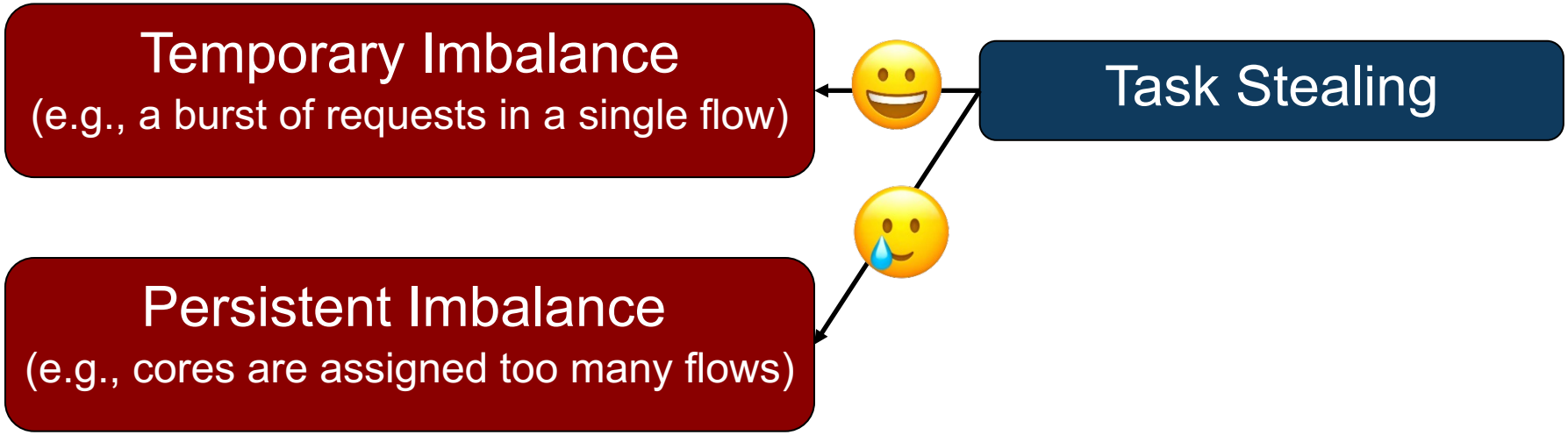
Observation:
Existing policies fail to consider both causes of imbalance.

Temporary Imbalance
(e.g., a burst of requests in a single flow)

Persistent Imbalance
(e.g., cores are assigned too many flows)

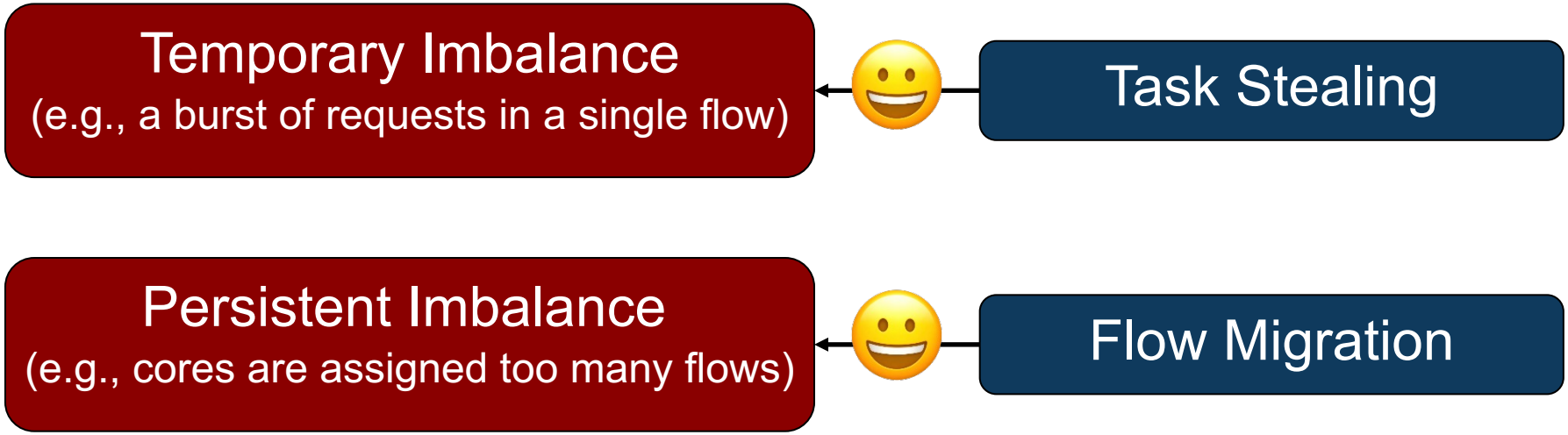
SBB's Request Load Balancing

Observation:
Existing policies fail to consider both causes of imbalance.



SBB's Request Load Balancing

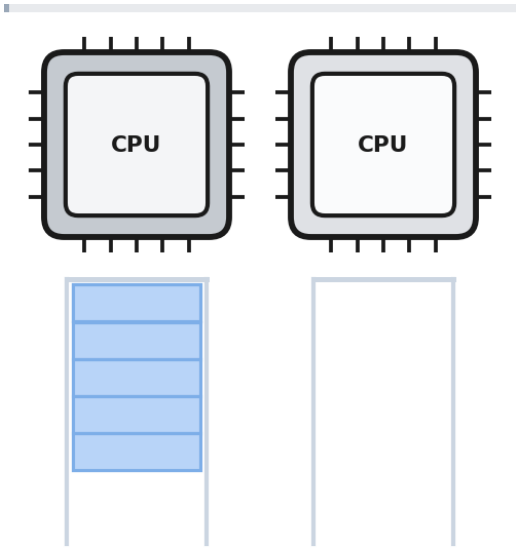
Observation:
Existing policies fail to consider both causes of imbalance.



SBB's Request Load Balancing

Temporary Imbalance

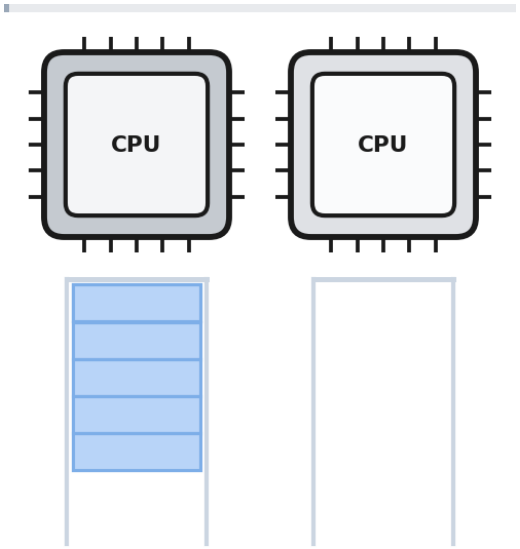
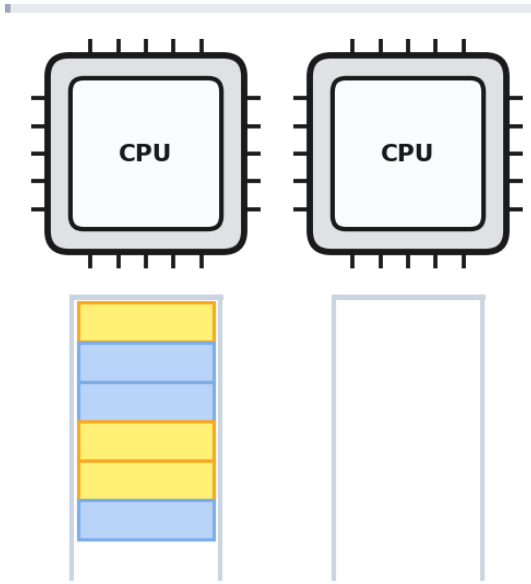
Task Stealing



Packet Level

Note: The animations in all GIFs are not preserved when the PPTX is converted to PDF. Please refer to the screen recording video to view the animations.

SBB's Request Load Balancing

Temporary Imbalance	Persistent Imbalance
Task Stealing	Flow Migration
	
Packet Level	Flow Level

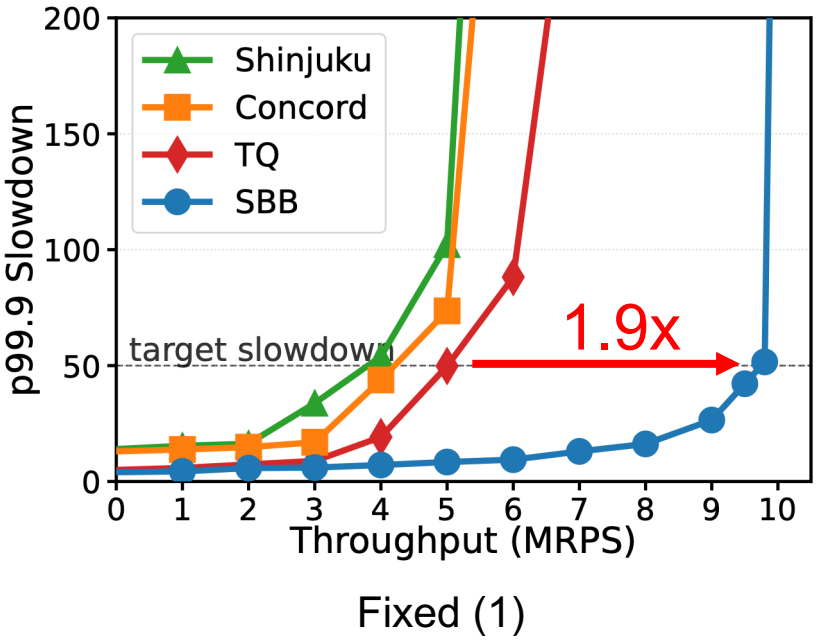
Evaluation

- Setup:
  - 128-core Xeon Platinum 8592 processors
  - 100 Gbps Intel E810 NIC

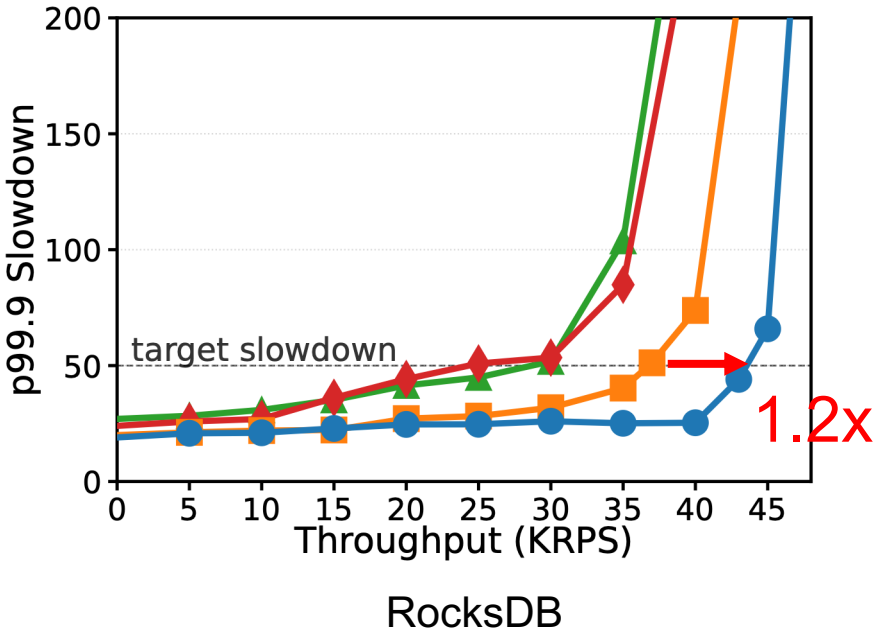
- Baselines:
  - Shinjuku, Concord, TQ, Caladan, Caladan-DL, Skyloft

- **How does SBB perform for a single networking application?**
- **How does SBB behave under multi-application co-location?**
- **How does SBB scale with an increasing number of worker cores?**

Evaluation #1: Dedicated Cores



Fixed (1)

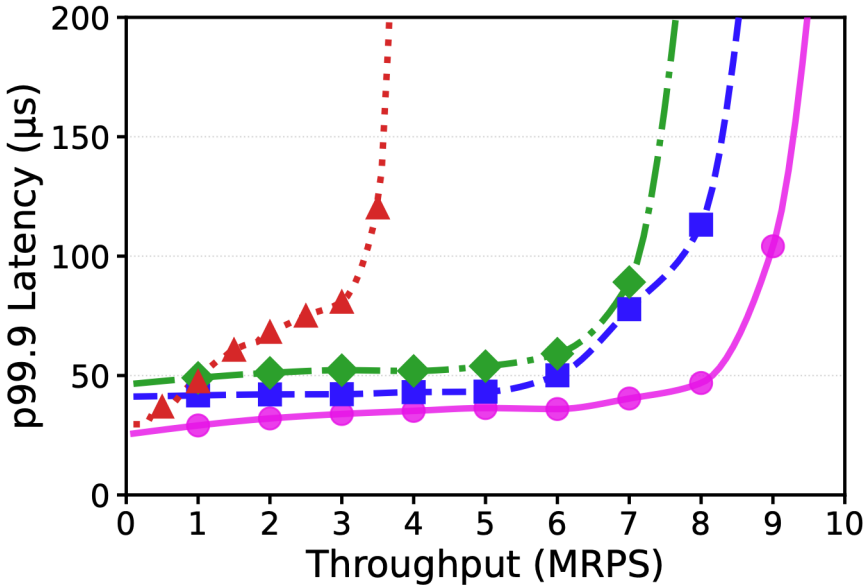


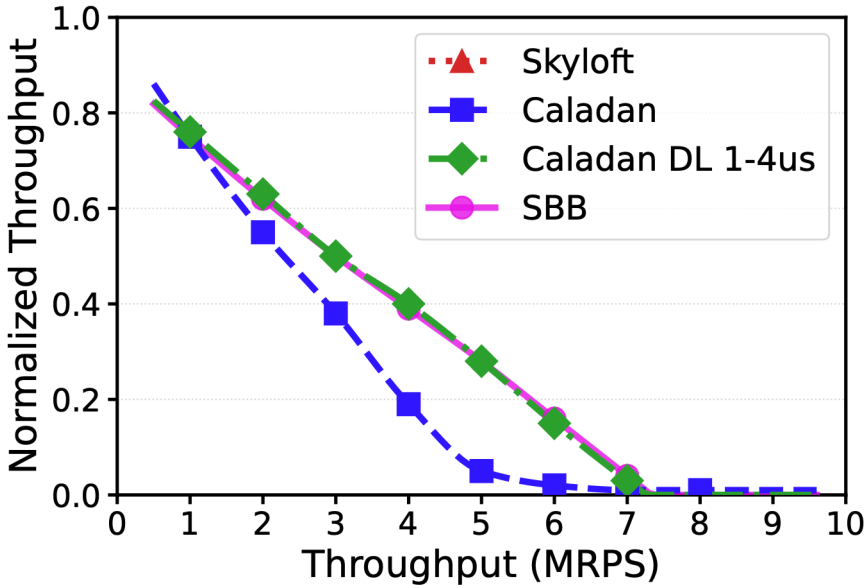
RocksDB

SBB achieves high performance with dedicated core for various workloads.

Evaluation #2: Multi-task Co-location

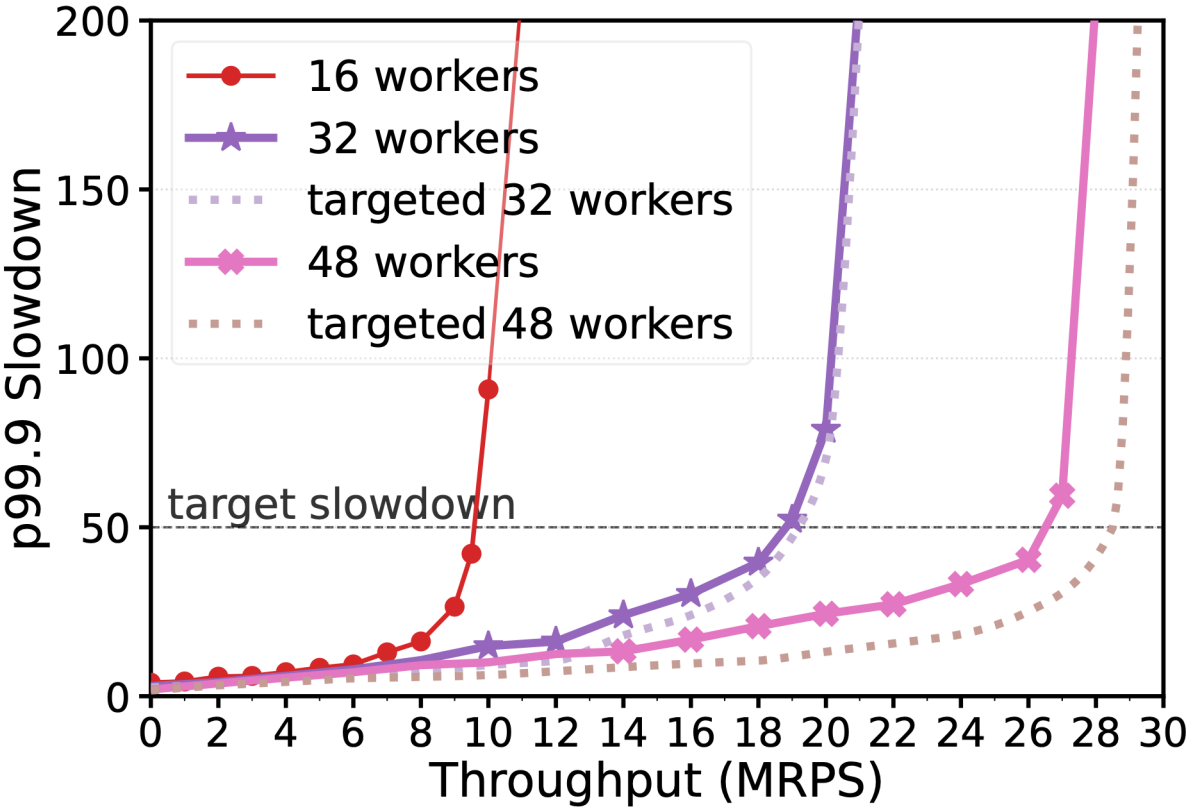
- LC task: Memcached
- BE task: Swaptions





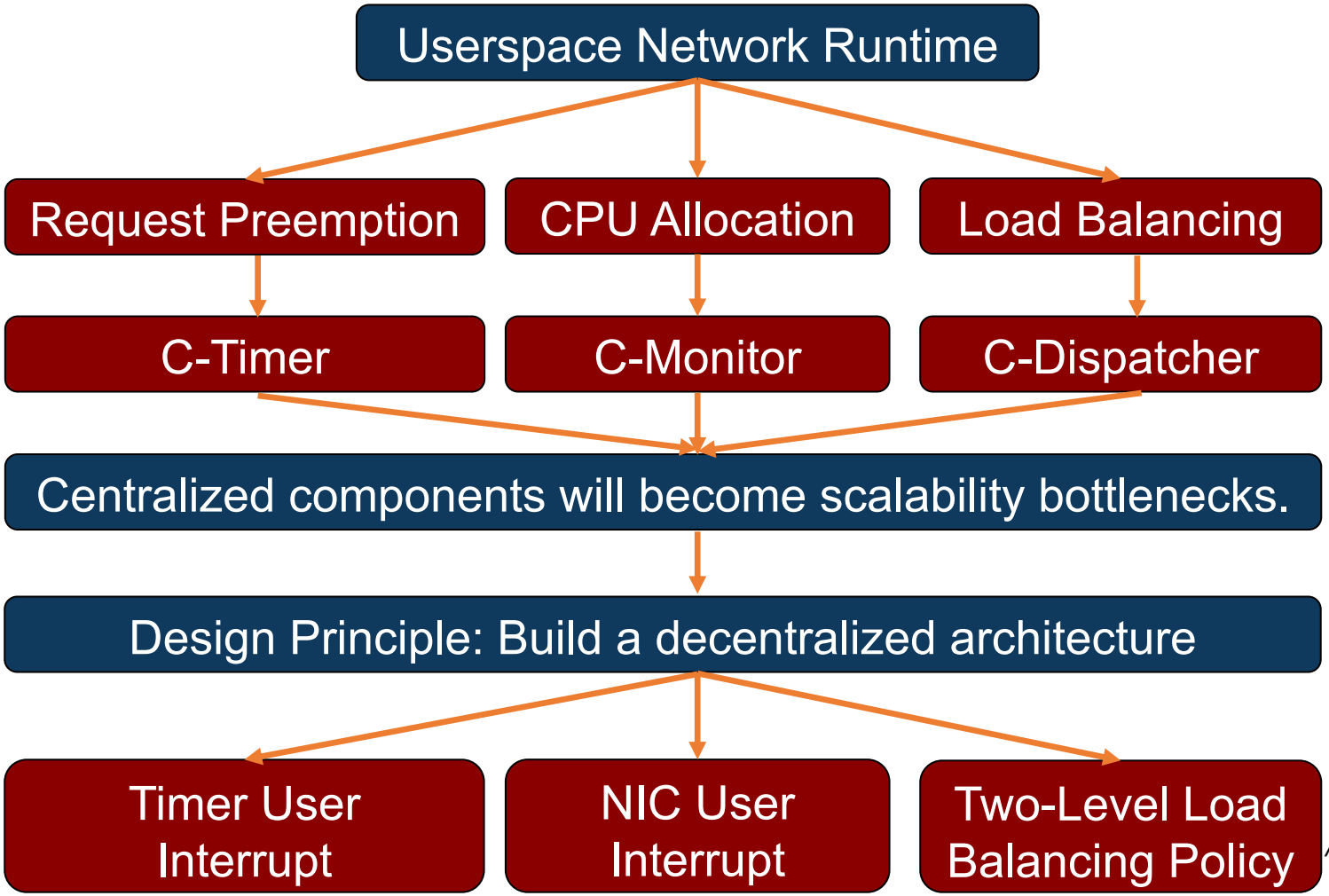
SBB achieves both lower latency and higher efficiency when co-locating.

Evaluation #3: Scalability



SBB scales well for all three types of scheduling across various workloads.

Summary



Our lab:
https://telos.top/

Source code:
https://github.com/TELOS-syslab/SBB-Artifact