ACM Symposium on Cloud Computing

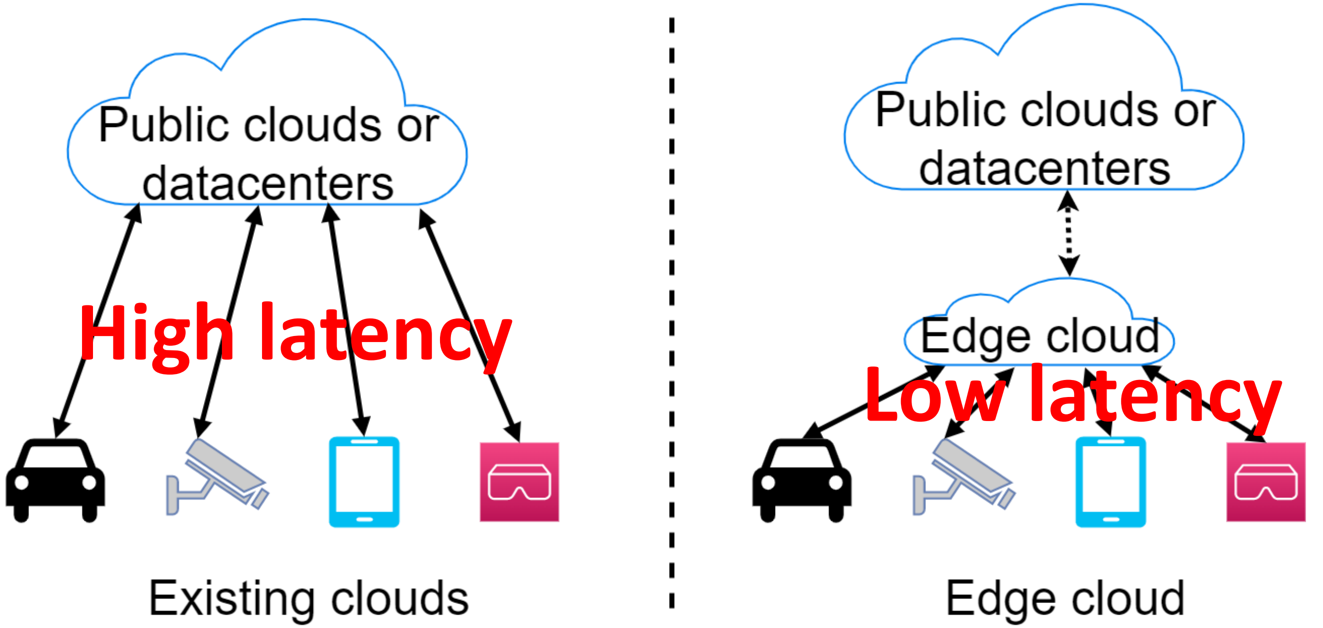
Achieving Low Latency in Public Edges by Hiding Workloads Mutual Interference

Weiwei Jia, Jiyuan Zhang, Jianchen Shan, Jing Li, Xiaoning Ding

The University of Rhode Island New Jersey Institute of Technology Hofstra University

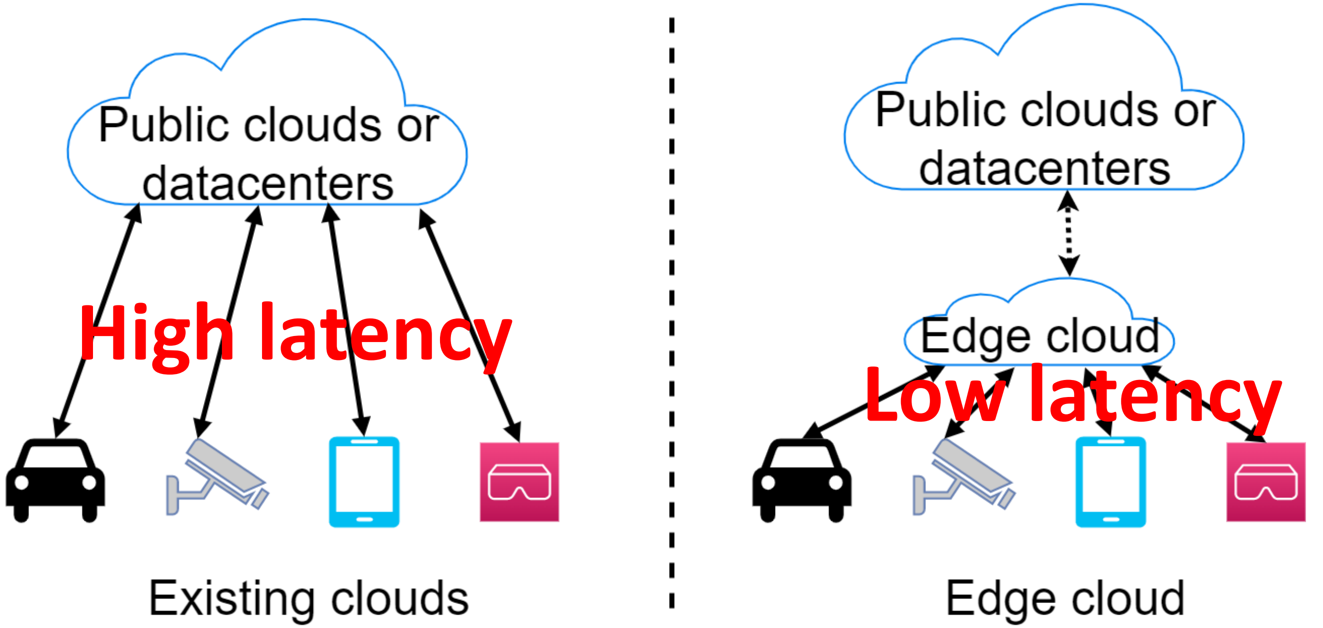
Background: edge cloud and its applications

- Edge cloud is a tiny cloud deployed close to end users.
  - Constrained computer resources.
  - Low latency.
  - E.g., AWS local zones.



Background: edge cloud and its applications

- Edge cloud is a tiny cloud deployed close to end users.
  - Constrained computer resources.
  - Low latency.
  - E.g., AWS local zones.



- Edge cloud is dominated by latency sensitive apps that require low latency and high resource demanding.
  - E.g., microseconds for autonomous vehicles/robots and AR/VR.
  - Edge applications usually generate large amount of data for computation.
    - Resource-constrained end devices cannot afford the computation.

Resource sharing increases edge app mutual interference and latency

- Multiple applications from different users share the same edge server.
  - Execution of an app contends resources and interferes with execution of other apps.
    - Interference may cause significant performance penalties if not well controlled.

Resource sharing increases edge app mutual interference and latency

- Multiple applications from different users share the same edge server.
  - Execution of an app contends resources and interferes with execution of other apps.
    - Interference may cause significant performance penalties if not well controlled.
- Resource contention and apps mutual interference are unavoidable in edge cloud!
  - Limited number of edge servers host large number of apps.
    - Hard to effectively distribute and separate interfering apps across different servers.
  - Resource usage patterns of edge applications may change dynamically.
    - Hard to predict which apps may interfere with each other.

Resource sharing increases edge app mutual interference and latency

- Multiple applications from different users share the same edge server.
  - Execution of an app contends resources and interferes with execution of other apps.
    - Interference may cause significant performance penalties if not well controlled.
- Resource contention and apps mutual interference are unavoidable in edge cloud!
  - Limited number of edge servers host large number of apps.
    - Hard to effectively distribute and separate interfering apps across different servers.
  - Resource usage patterns of edge applications may change dynamically.
    - Hard to predict which apps may interfere with each other.
- Problem: how to efficiently schedule latency sensitive apps to reduce their mutual interference and latencies in edge cloud.
  - This problem is under-studied in edge cloud scenarios.

Existing cloud solutions are not effective for edge cloud

- Resource over-provisioning (Mengwei Xu et. al.[IMC '21]).
  - Significant resource waste.
    - Average CPU utilization is about 6x lower on edge servers than on cloud servers.

Existing cloud solutions are not effective for edge cloud

- Resource over-provisioning (Mengwei Xu et. al.[IMC ‘21]).
  - Significant resource waste.
    - Average CPU utilization is about 6x lower on edge servers than on cloud servers.
- Collocating best-effort apps with latency sensitive apps (PARTIES[ASPLOS ‘19]).
  - Edge cloud is dominated by latency sensitive applications.

Existing cloud solutions are not effective for edge cloud

- Resource over-provisioning (Mengwei Xu et. al.[IMC ‘21]).
  - Significant resource waste.
    - Average CPU utilization is about 6x lower on edge servers than on cloud servers.
- Collocating best-effort apps with latency sensitive apps (PARTIES[ASPLOS ‘19]).
  - Edge cloud is dominated by latency sensitive applications.
- Avoid collocating applications that interfere with each other (Bolt[ASPLOS ‘17]).
  - Hard to predict which apps may interfere with each other due to app execution dynamics.

Existing cloud solutions are not effective for edge cloud

- Resource over-provisioning (Mengwei Xu et. al.[IMC ‘21]).
  - Significant resource waste.
    - Average CPU utilization is about 6x lower on edge servers than on cloud servers.
- Collocating best-effort apps with latency sensitive apps (PARTIES[ASPLOS ‘19]).
  - Edge cloud is dominated by latency sensitive applications.
- Avoid collocating applications that interfere with each other (Bolt[ASPLOS ‘17]).
  - Hard to predict which apps may interfere with each other due to app execution dynamics.
- Resource partitioning (Heracles[ISCA ‘15]) can barely help due to app dynamic resource usage.
  - Infrequent adjustment of resource partitions may not help.
    - Resources are not adapted to resource demand of the workload.
  - Frequent adjustment may lead to high overhead.

Outline

- Problem: how to schedule latency sensitive apps to reduce their mutual interference and latencies in edge cloud.

✓ DASEC: dynamic asymmetric scheduling for edge computing.
  - Basic idea: move the interference off the tasks on the critical paths of the workloads.
  - Key issues and solutions.

- Evaluation.
  - DASEC has been implemented in Linux/KVM, Linux CFS, and Google user-level scheduler (i.e., ghost [SOSP '21]).
  - Compared to vanilla Linux/KVM, DASEC reduces mean latency and 99^{th} tail latency by 46% and 52%, respectively.

Basic idea: move interference off the tasks on the critical paths of the workloads

- Critical path of a workload is the longest execution path from starting executing the workload to finishing the execution.
  - Tasks on the workload's critical path must be finished as quickly as possible to avoid delaying the progress of the workload.

Basic idea: move interference off the tasks on the critical paths of the workloads

- Critical path of a workload is the longest execution path from starting executing the workload to finishing the execution.
  - Tasks on the workload’s critical path must be finished as quickly as possible to avoid delaying the progress of the workload.
- How to judge whether a task is on the critical path.
  - All the other tasks of the workload depend on the task to make progress.

Basic idea: move interference off the tasks on the critical paths of the workloads

- Critical path of a workload is the longest execution path from starting executing the workload to finishing the execution.
  - Tasks on the workload's critical path must be finished as quickly as possible to avoid delaying the progress of the workload.
- How to judge whether a task is on the critical path.
  - All the other tasks of the workload depend on the task to make progress.
- Making interference affect mostly the tasks on non-critical paths and rarely the tasks on critical paths to reduce latency.
  - Tasks on non-critical paths to yield resources to tasks on critical paths.

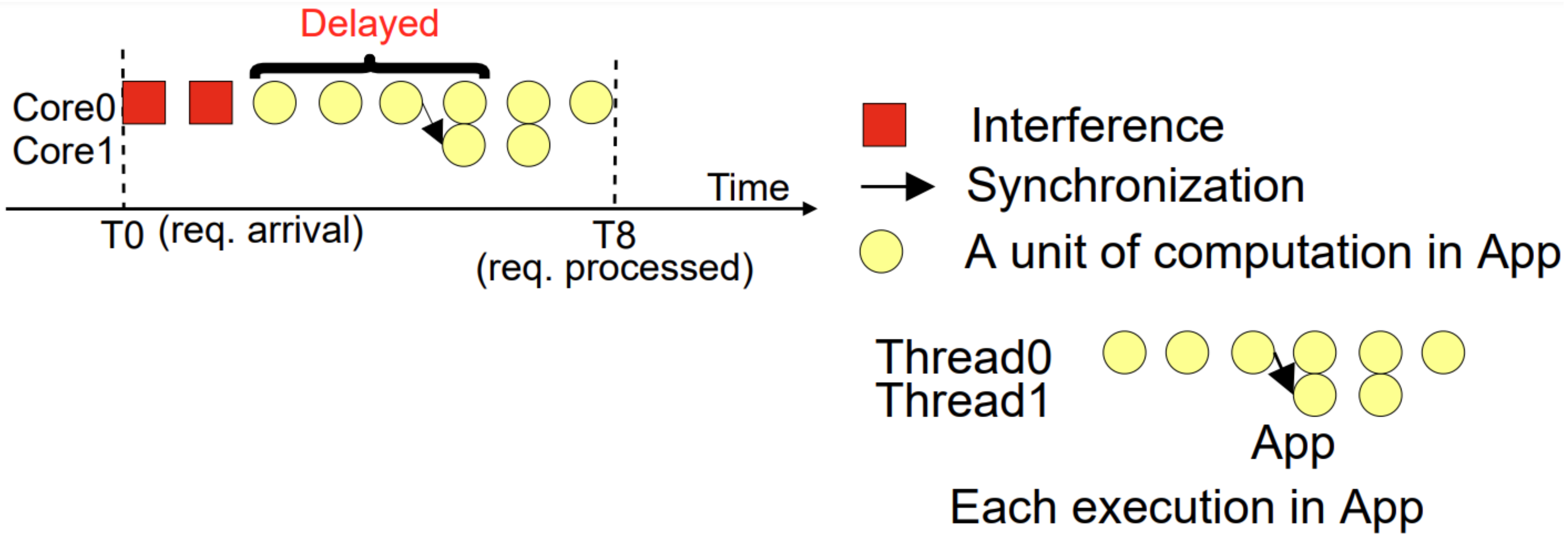
Basic idea: move interference off the tasks on the critical paths of the workloads

- Critical path of a workload is the longest execution path from starting executing the workload to finishing the execution.
  - Tasks on the workload's critical path must be finished as quickly as possible to avoid delaying the progress of the workload.
- How to judge whether a task is on the critical path.
  - All the other tasks of the workload depend on the task to make progress.
- Making interference affect mostly the tasks on non-critical paths and rarely the tasks on critical paths to reduce latency.
  - Tasks on non-critical paths to yield resources to tasks on critical paths.
- This work focuses on the interference caused by sharing CPU cores.
  - CPU resources, as the most important resource type, have the largest impact on perf.

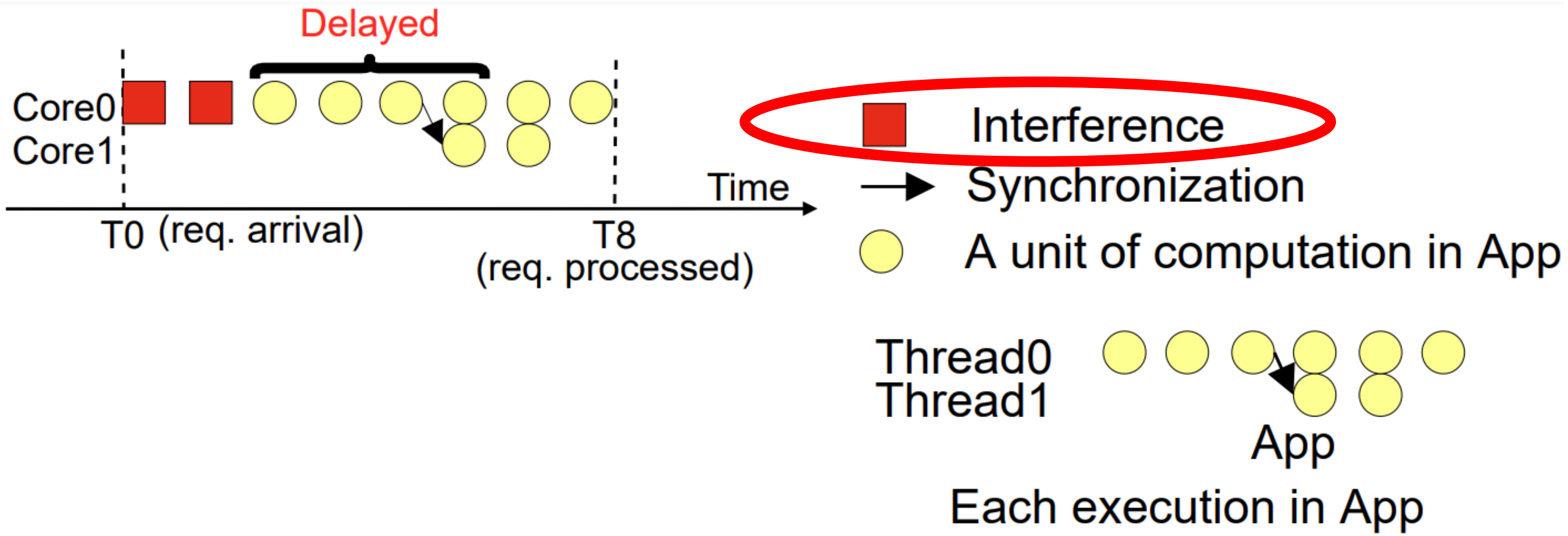
Basic idea: move interference off the tasks on the critical paths of the workloads

- Critical path of a workload is the longest execution path from starting executing the workload to finishing the execution.
  - Tasks on the workload's critical path must be finished as quickly as possible to avoid delaying the progress of the workload.
- How to judge whether a task is on the critical path.
  - All the other tasks of the workload depend on the task to make progress.
- Making interference affect mostly the tasks on non-critical paths and rarely the tasks on critical paths to reduce latency.
  - Tasks on non-critical paths to yield resources to tasks on critical paths.
- This work focuses on the interference caused by sharing CPU cores.
  - CPU resources, as the most important resource type, have the largest impact on perf.
- Tasks in workloads interfere with each other in three ways.
  - Tasks on app's critical path are delayed, interrupted, or lack CPU share.

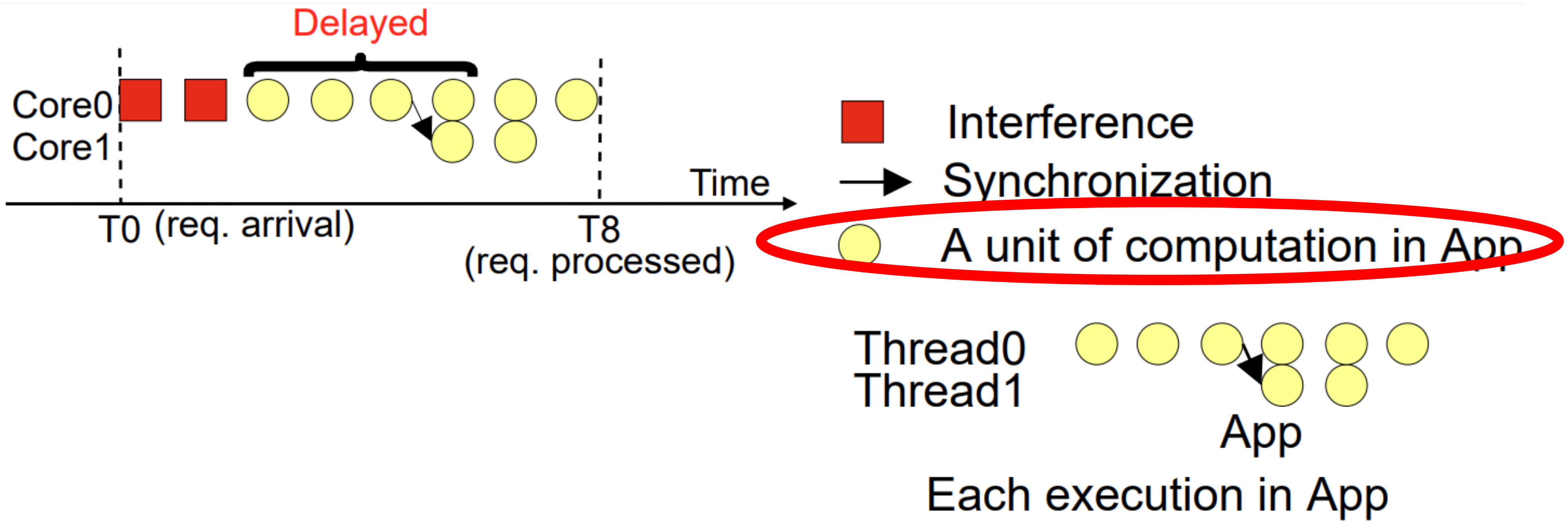
Interference #1: tasks on app's critical path are delayed and start late



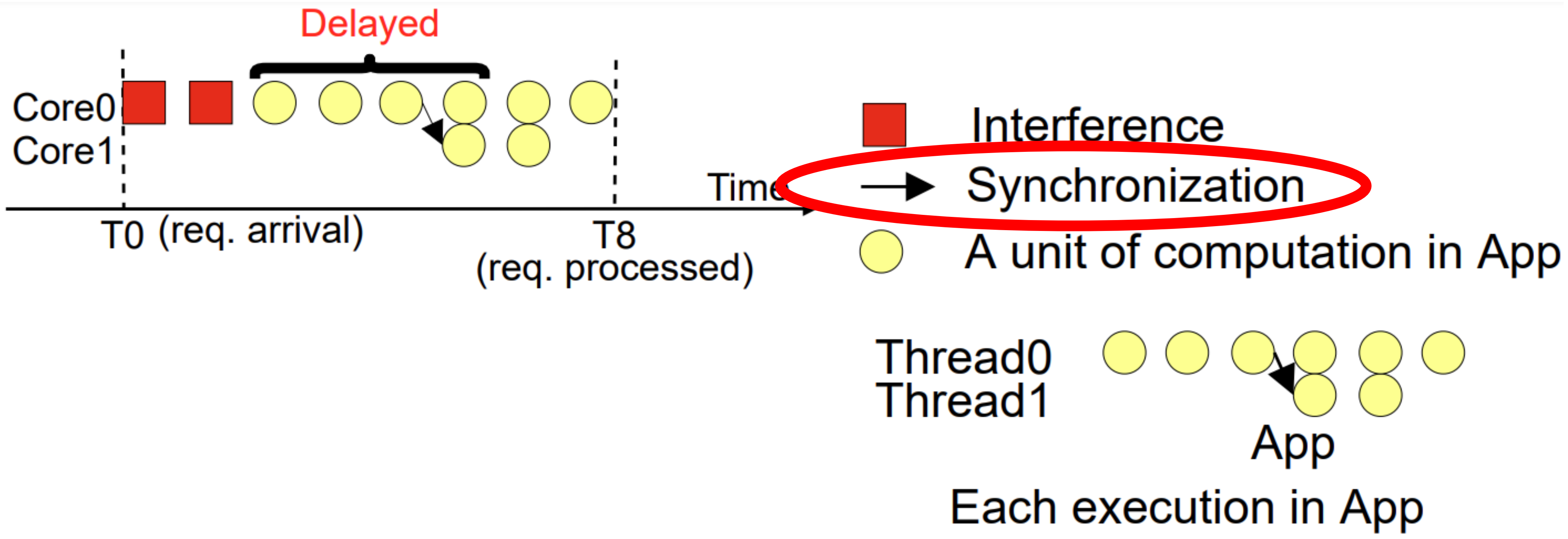
Interference #1: tasks on app's critical path are delayed and start late



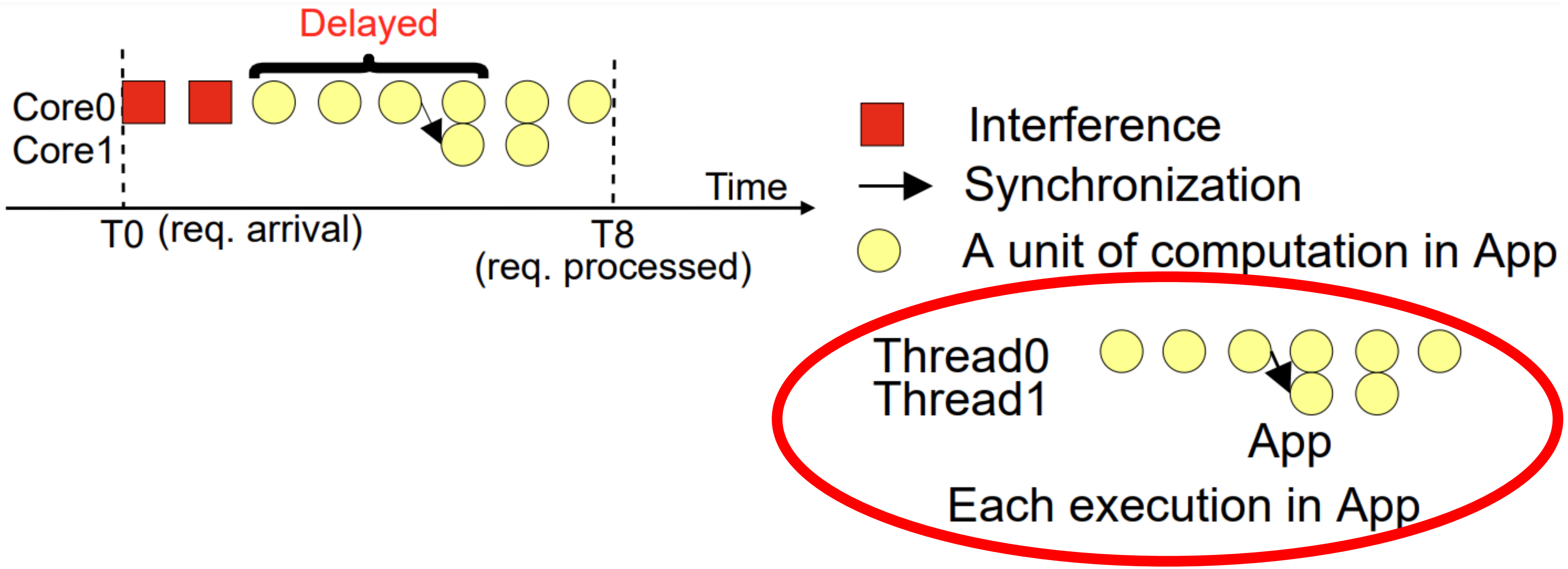
Interference #1: tasks on app's critical path are delayed and start late



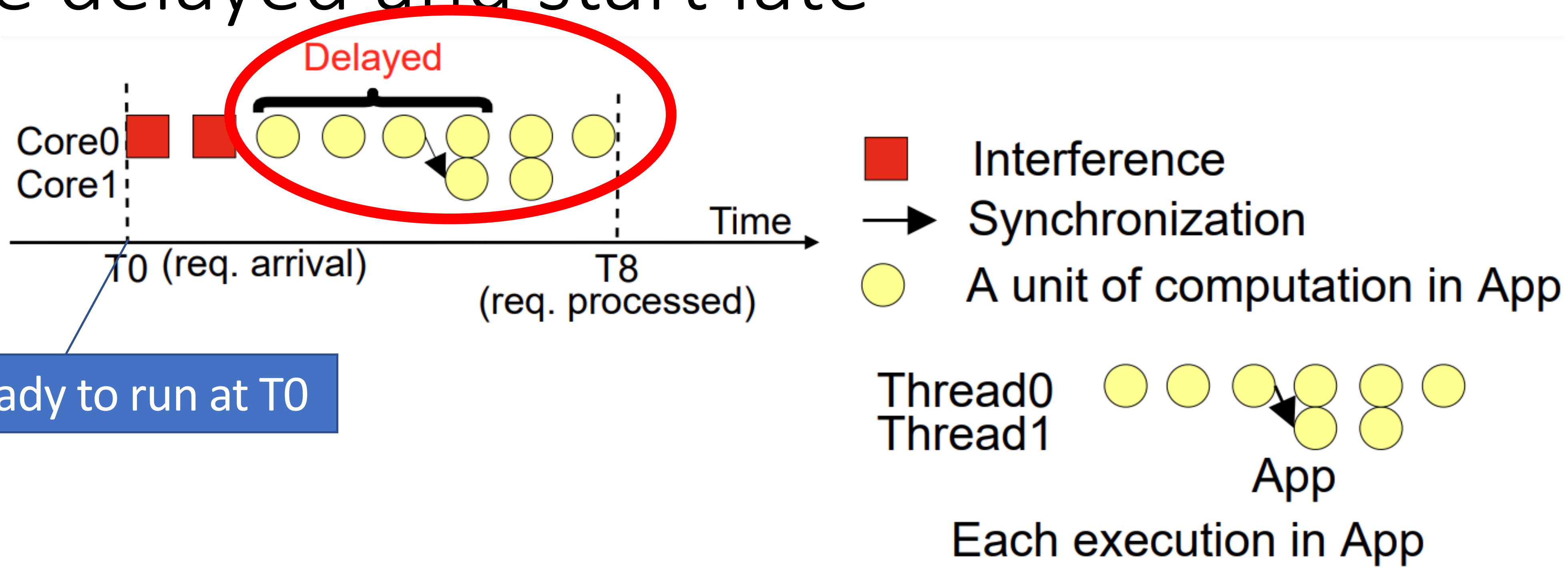
Interference #1: tasks on app's critical path are delayed and start late



Interference #1: tasks on app's critical path are delayed and start late

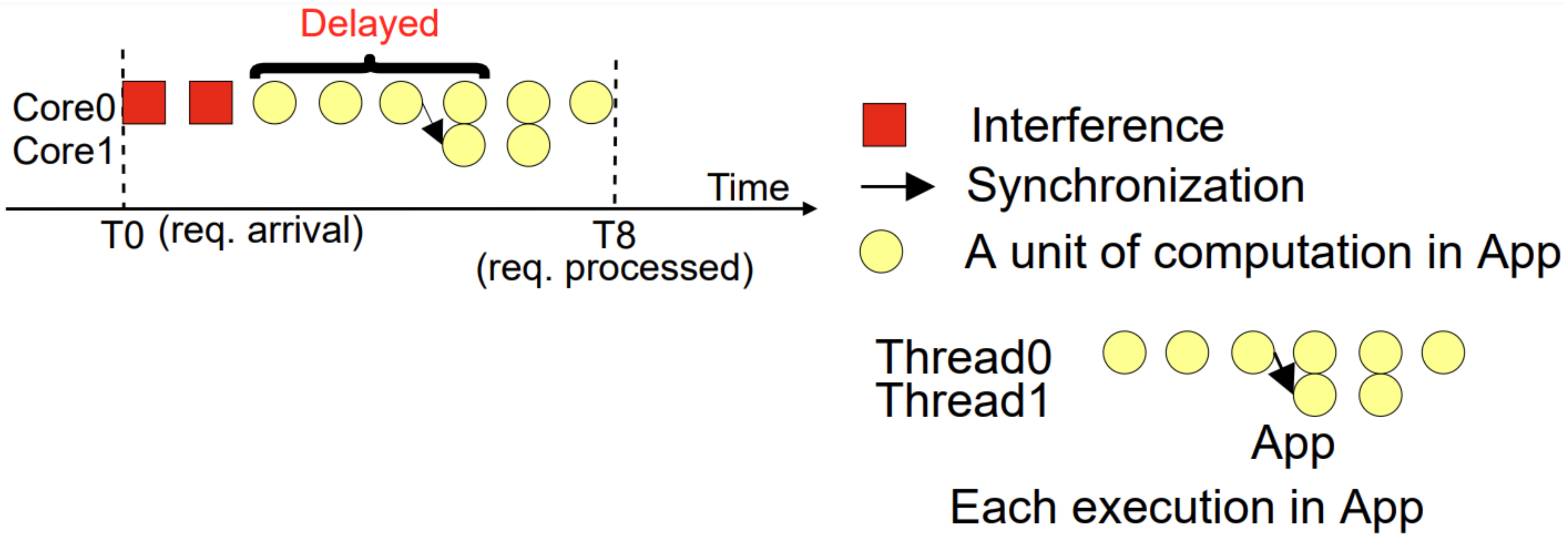


Interference #1: tasks on app's critical path are delayed and start late



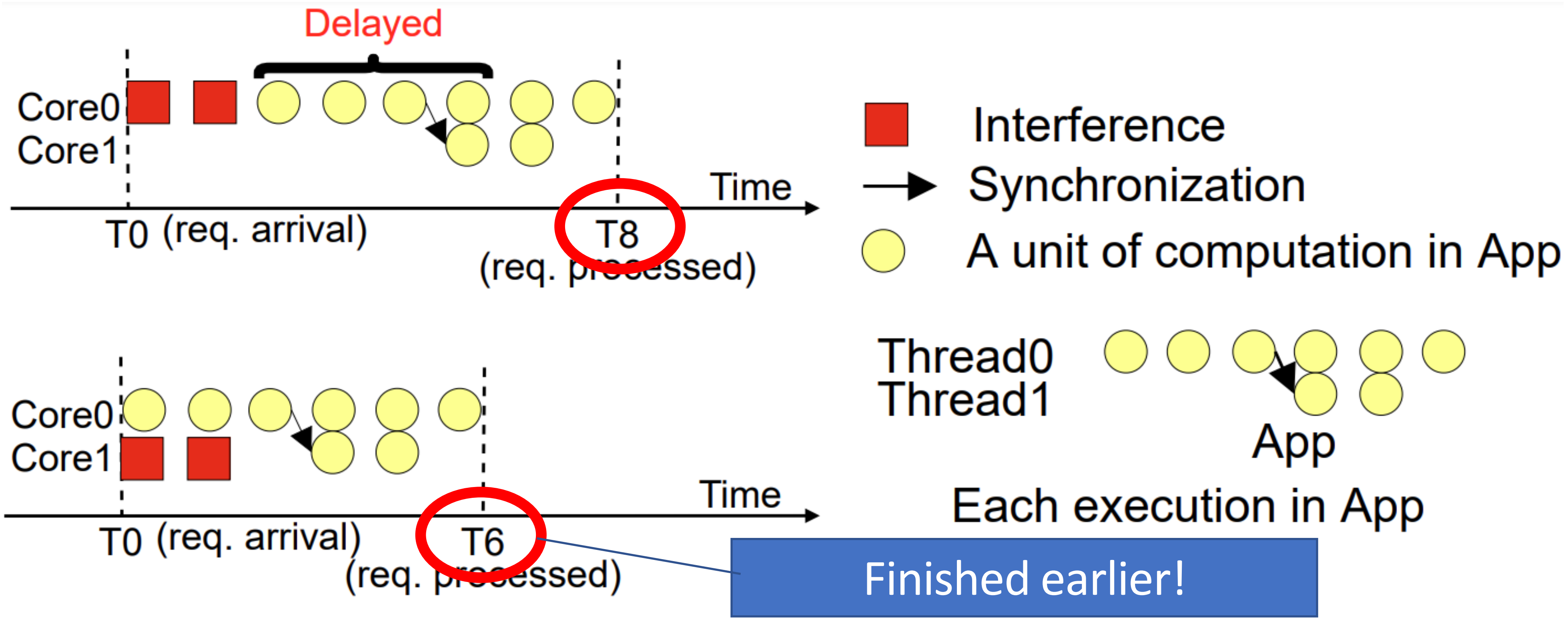
App is ready to run at T0

Interference #1: tasks on app's critical path are delayed and start late



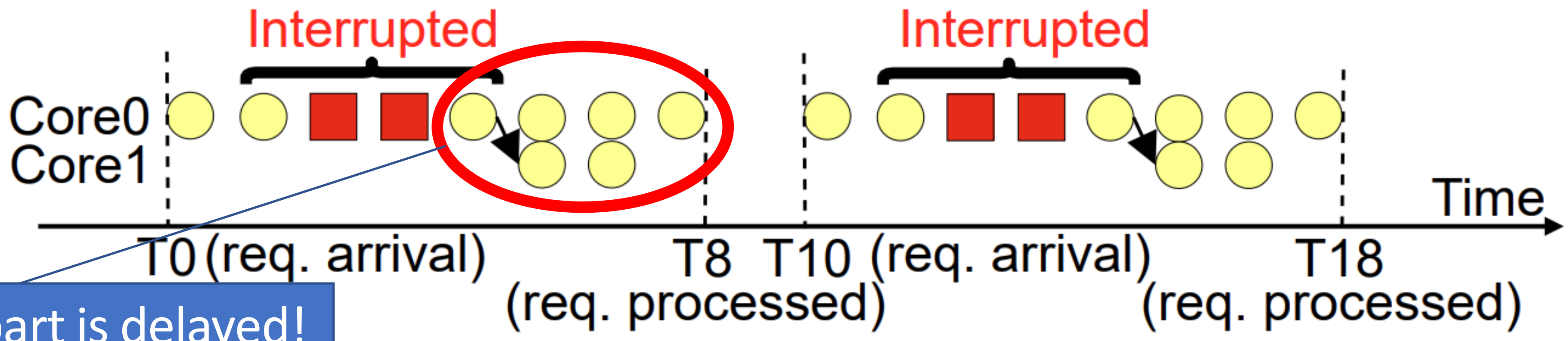
- Detection: at the end of previous time period, a thread/process was in "ready" or "running" status and its timeslice was not used up.

Interference #1: tasks on app's critical path are delayed and start late



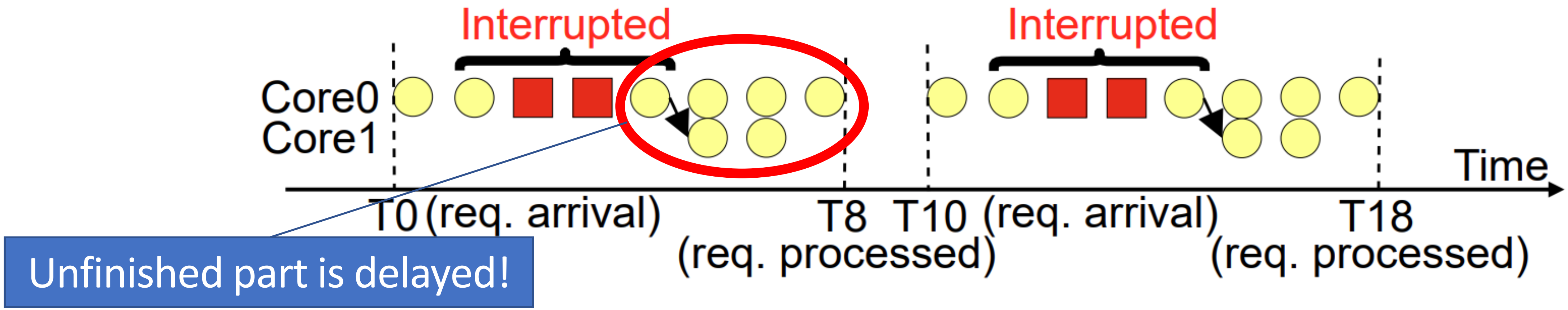
- Detection: at the end of previous time period, a thread/process was in "ready" or "running" status and its timeslice was not used up.
- Solution: reduce rescheduling latency of the thread/process to let it start early.

Interference #2: tasks on app's critical path are interrupted



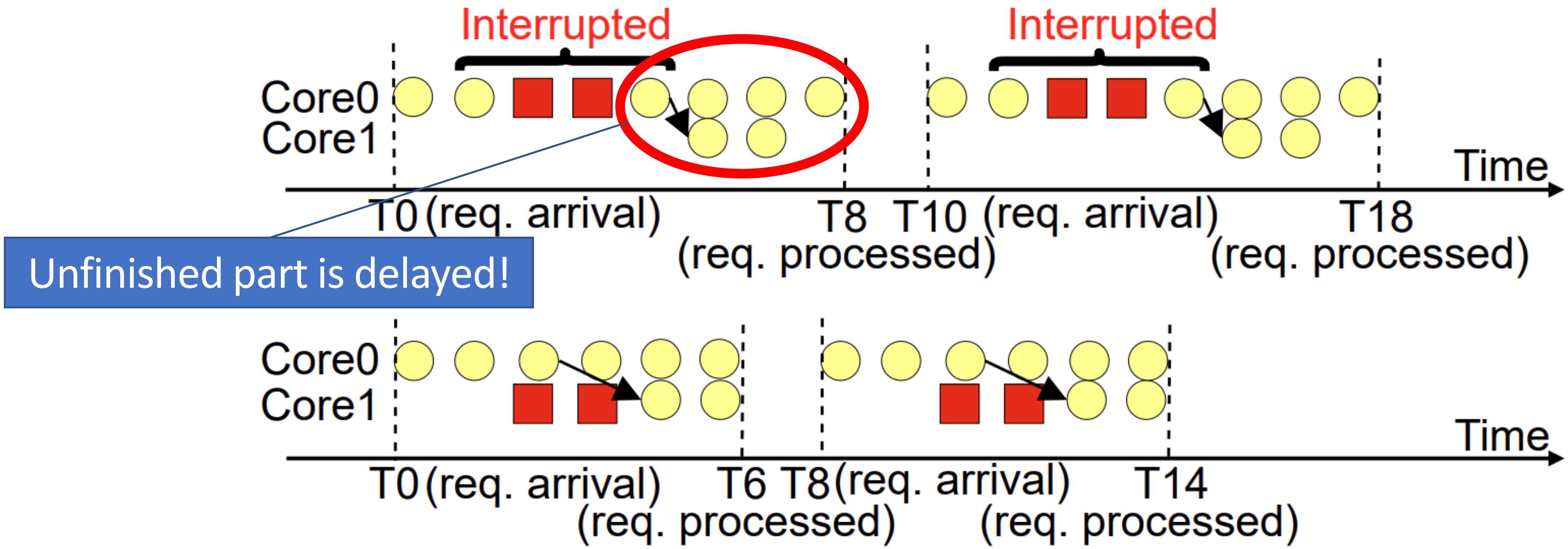
Unfinished part is delayed!

Interference #2: tasks on app's critical path are interrupted



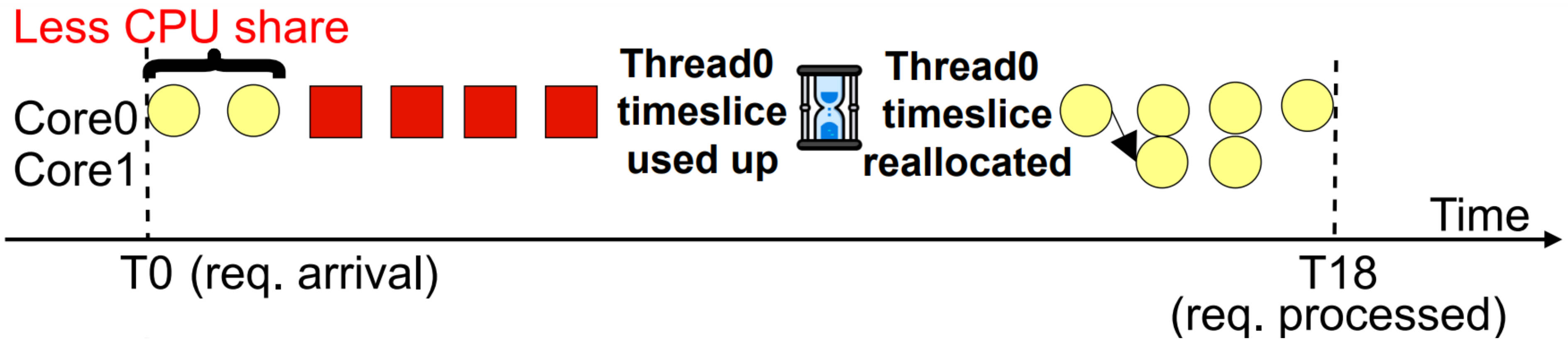
- Detection: threads/processes with low rescheduling latencies are scheduled on the same core or their total timeslices exceed core's capacity.

Interference #2: tasks on app's critical path are interrupted

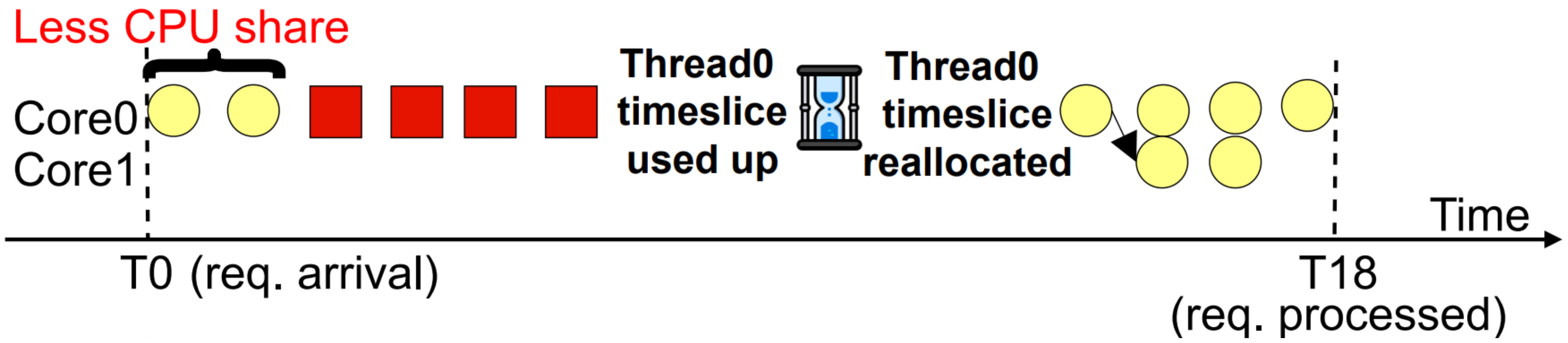


- Detection: threads/processes with low rescheduling latencies are scheduled on the same core or their total timeslices exceed core's capacity.
- Solution: adjust the layout of threads/processes on cores in a conservative way.

Interference #3: tasks on app's critical path lack CPU share

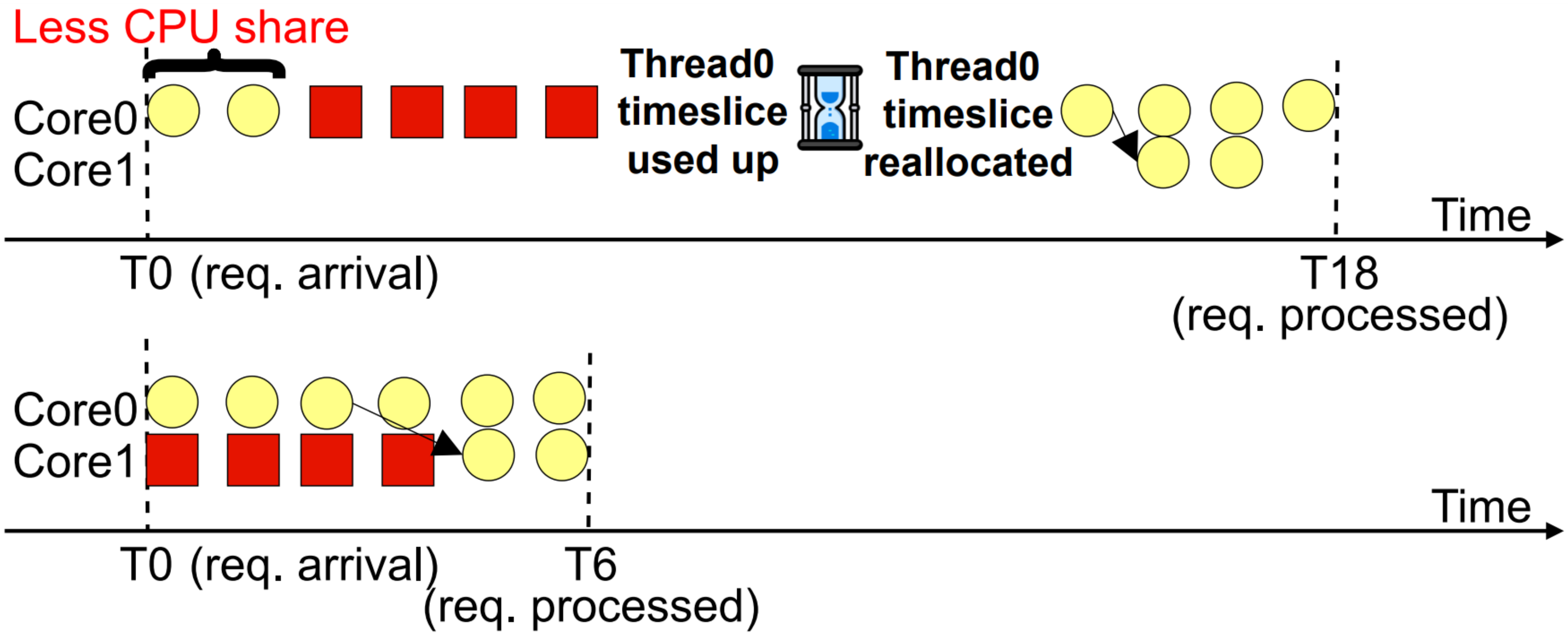


Interference #3: tasks on app's critical path lack CPU share



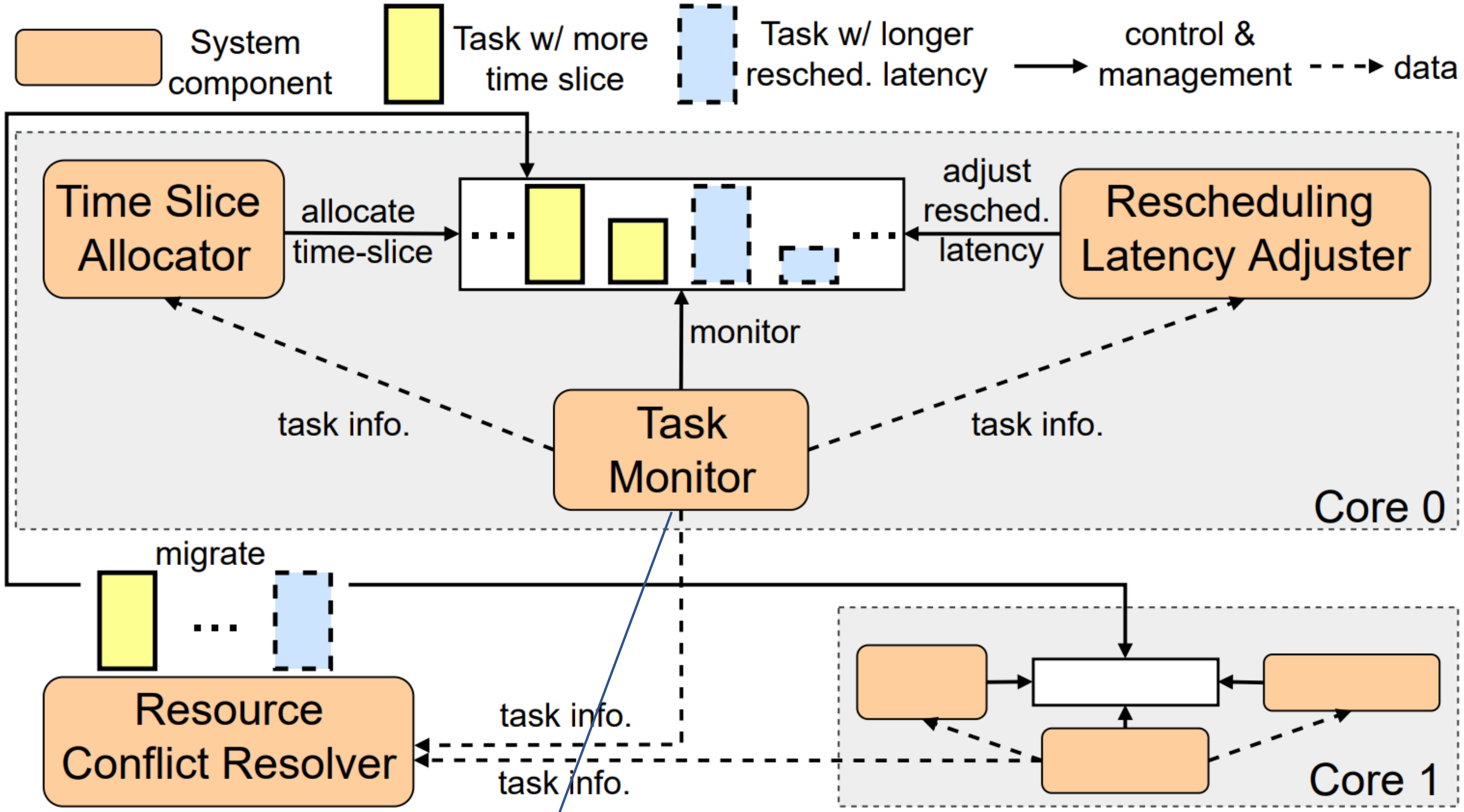
- Detection: threads/processes were preempted due to depletion of timeslice in previous time period.

Interference #3: tasks on app's critical path lack CPU share



- Detection: threads/processes were preempted due to depletion of timeslice in previous time period.
- Solution: keep total timeslice of the app fixed and allocate more time share to the threads/processes on app's critical path.

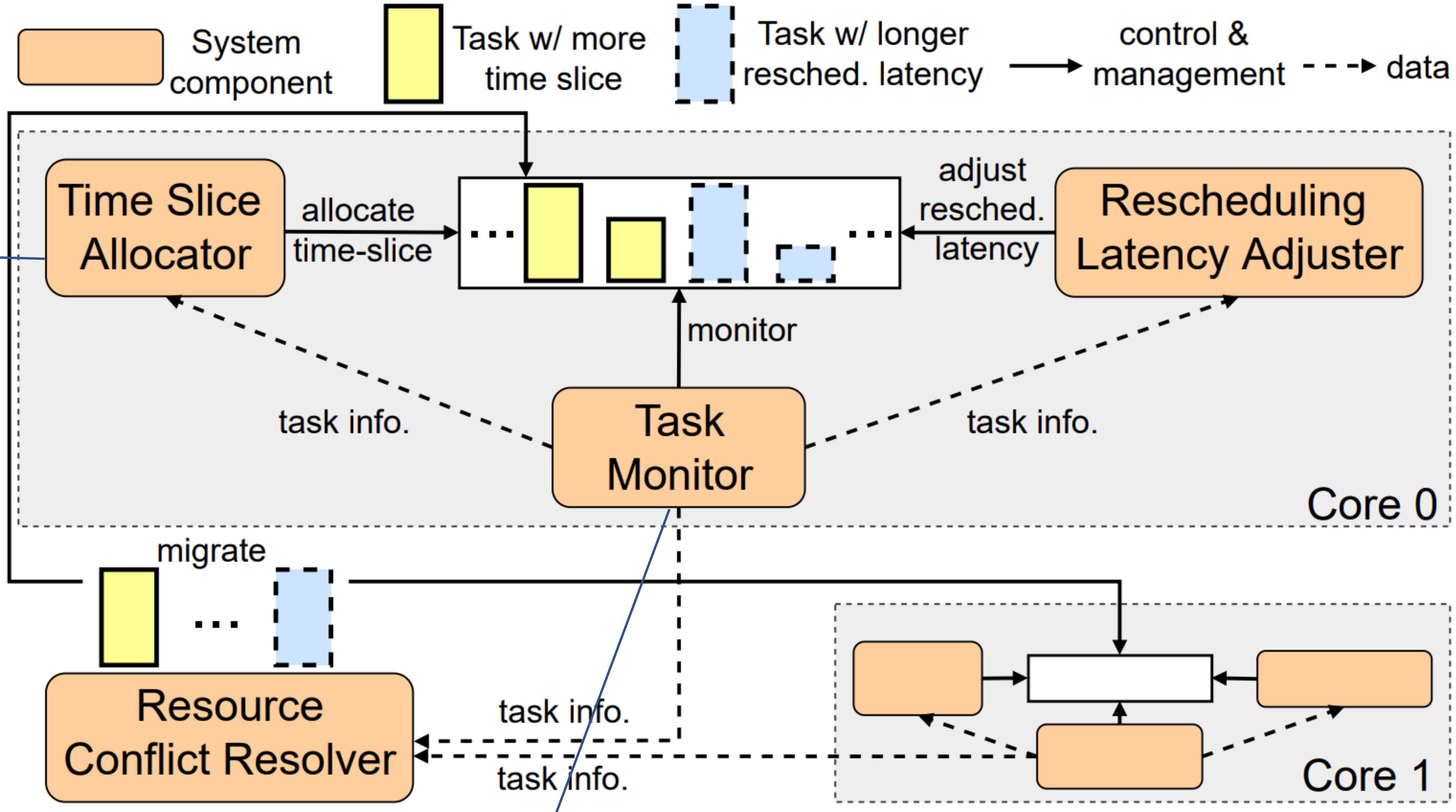
DASEC implementation



Periodically monitors tasks' states and events, e.g., remaining timeslice.
(Implemented in Linux Proc FS)

DASEC implementation

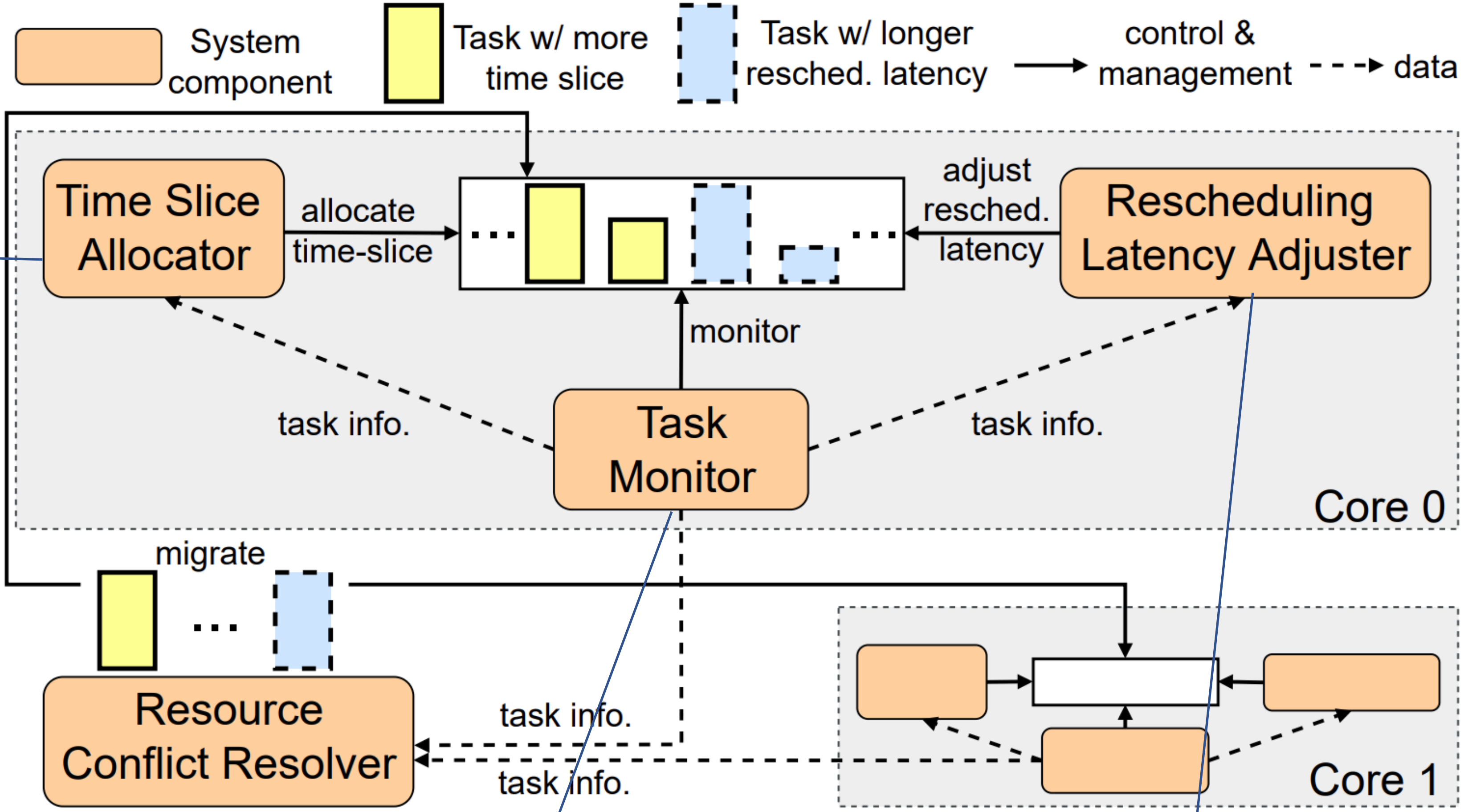
Assign timeslice to each task for the upcoming time period through changing tasks' weights.
(Implemented in Linux CFS)



Periodically monitors tasks' states and events, e.g., remaining timeslice.
(Implemented in Linux Proc FS)

DASEC implementation

Assign timeslice to each task for the upcoming time period through changing tasks' weights.
(Implemented in Linux CFS)

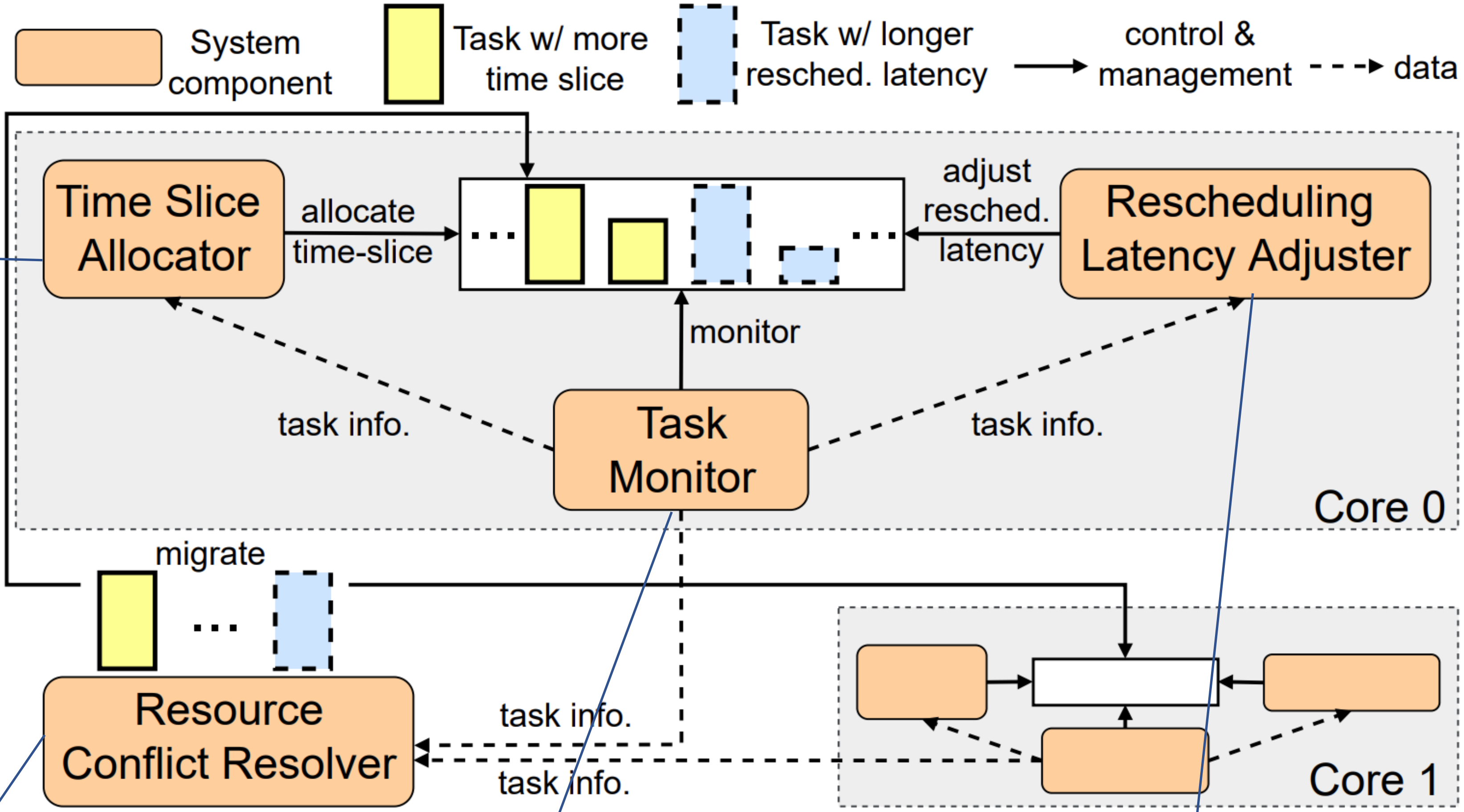


Periodically monitors tasks' states and events, e.g., remaining timeslice.
(Implemented in Linux Proc FS)

Adjust wakeup latency to change each task's rescheduling latency.
(Implemented in Linux CFS)

DASEC implementation

Assign timeslice to each task for the upcoming time period through changing tasks' weights.
(Implemented in Linux CFS)



Resolve time slice conflicts and rescheduling latency conflicts on cores through changing tasks' layout on cores.
(Implemented with Linux set affinity interfaces)

Periodically monitors tasks' states and events, e.g., remaining timeslice.
(Implemented in Linux Proc FS)

Adjust wakeup latency to change each task's rescheduling latency.
(Implemented in Linux CFS)

Outline

- Problem: how to schedule latency sensitive apps to reduce their mutual interference and latencies in edge cloud.
- DASEC: dynamic asymmetric scheduling for edge computing.
  - Basic idea: move the interference off the tasks on the critical paths of the workloads.
  - Key issues and solutions.

✓ Evaluation.
  - DASEC has been implemented in Linux/KVM, Linux CFS, and Google user-level scheduler (i.e., ghost [SOSP '21]).
  - Compared to vanilla Linux/KVM, DASEC reduces mean latency and 99th tail latency by 46% and 52%, respectively.

Experimental setup

- HPE ProLiant DL580 Gen10 server with 80 cores, 256GB DRAM, and two 2TB SSDs.
- Both VMs and VMM (Linux QEMU/KVM) use Ubuntu Linux 18.04 with the same Linux 5.3 kernel and software configuration.
- Each VM has 16 vCPUs and 16GB memory.
- Compared with vanilla Linux/KVM, PARTIES ([ASPLOS ‘19]), and BVT ([SOSP ‘99] and [EuroSys ‘14]).
- Test under two settings.
  - Multiple VM instances of the same workload.
  - Multiple VM instances of different workloads.

Evaluation applications and workloads

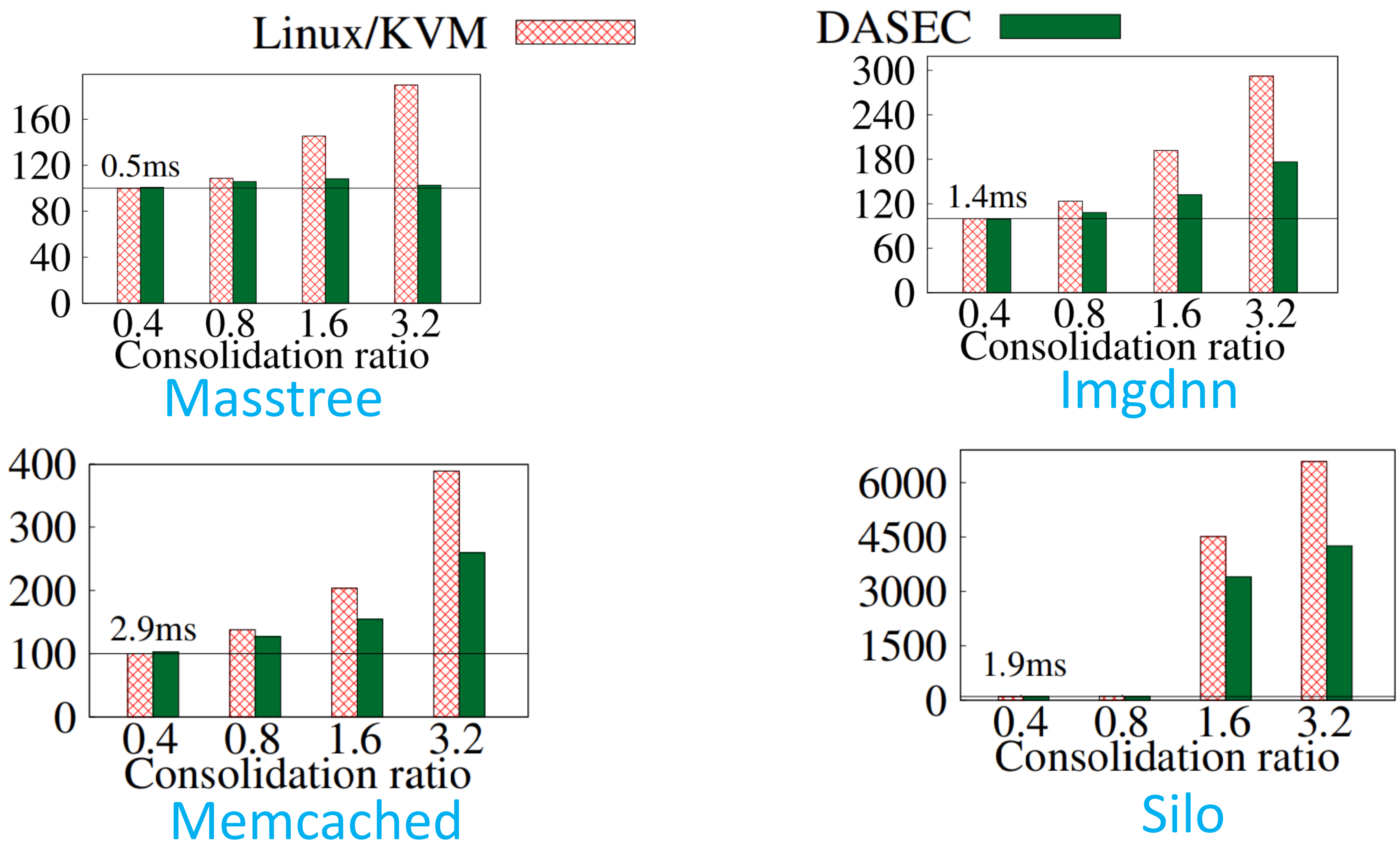
Application	Workload description
Image-classify	Image classification on ImageNet
Action-recognize	Video action recognition
Img-dnn	Handwriting recognition based on OpenCV
Masstree	In memory Key/Value store with 50% GET and 50% SET
Silo	In-memory transactional database with TPCC
Memcached	Serve requests (random keys, 50% SET, 50% GET)

Evaluation objectives

- What is DASEC's performance?
- How much performance improvement can be achieved with DASEC, compared with PARTIES?
- How effective is each technique in DASEC?
- What is DASEC's applicability and overhead?

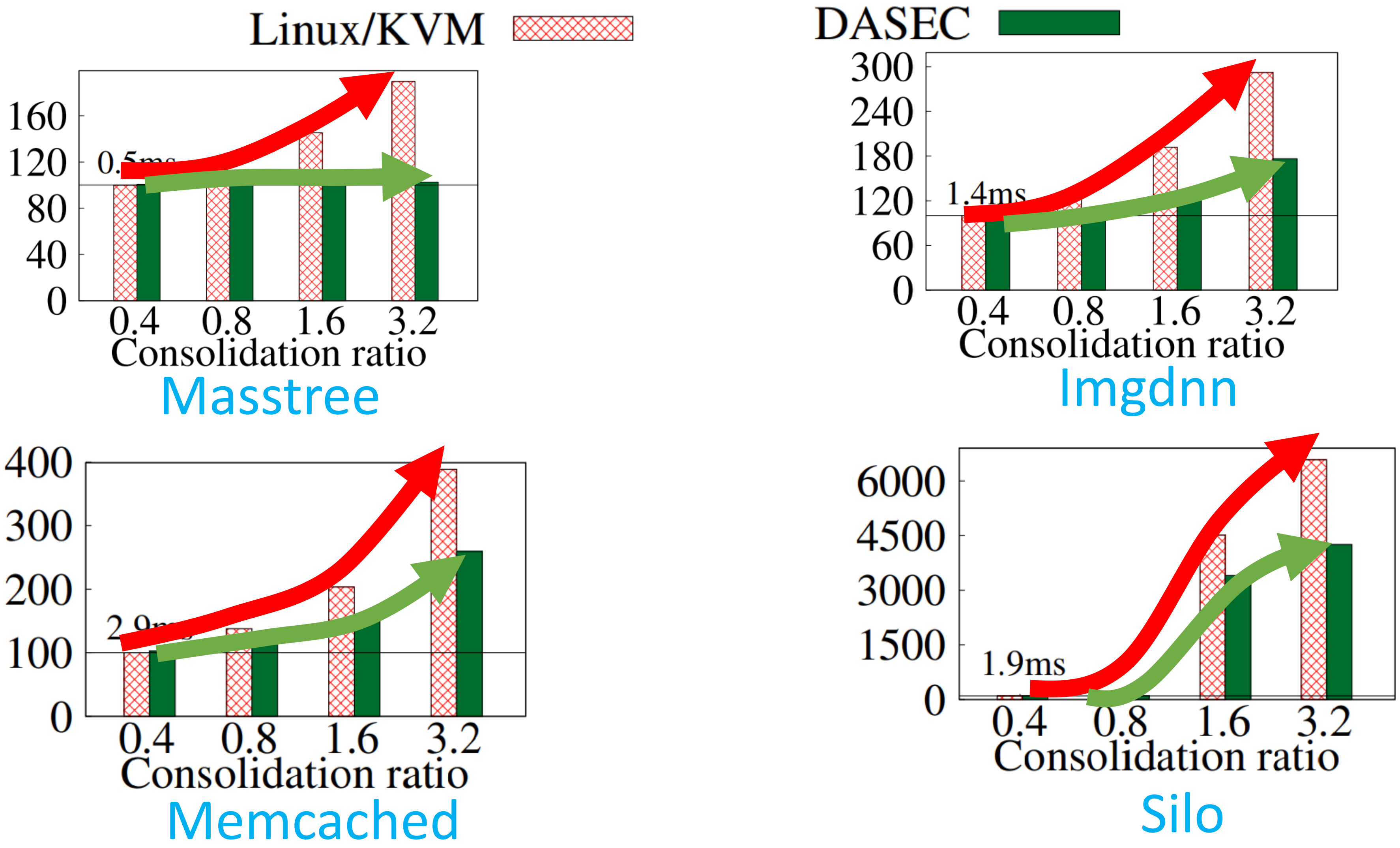
Mean latency (DASEC vs Linux/KVM)

*Latencies relative to Linux/KVM when consolidation ratio is 0.4.



Mean latency (DASEC vs Linux/KVM)

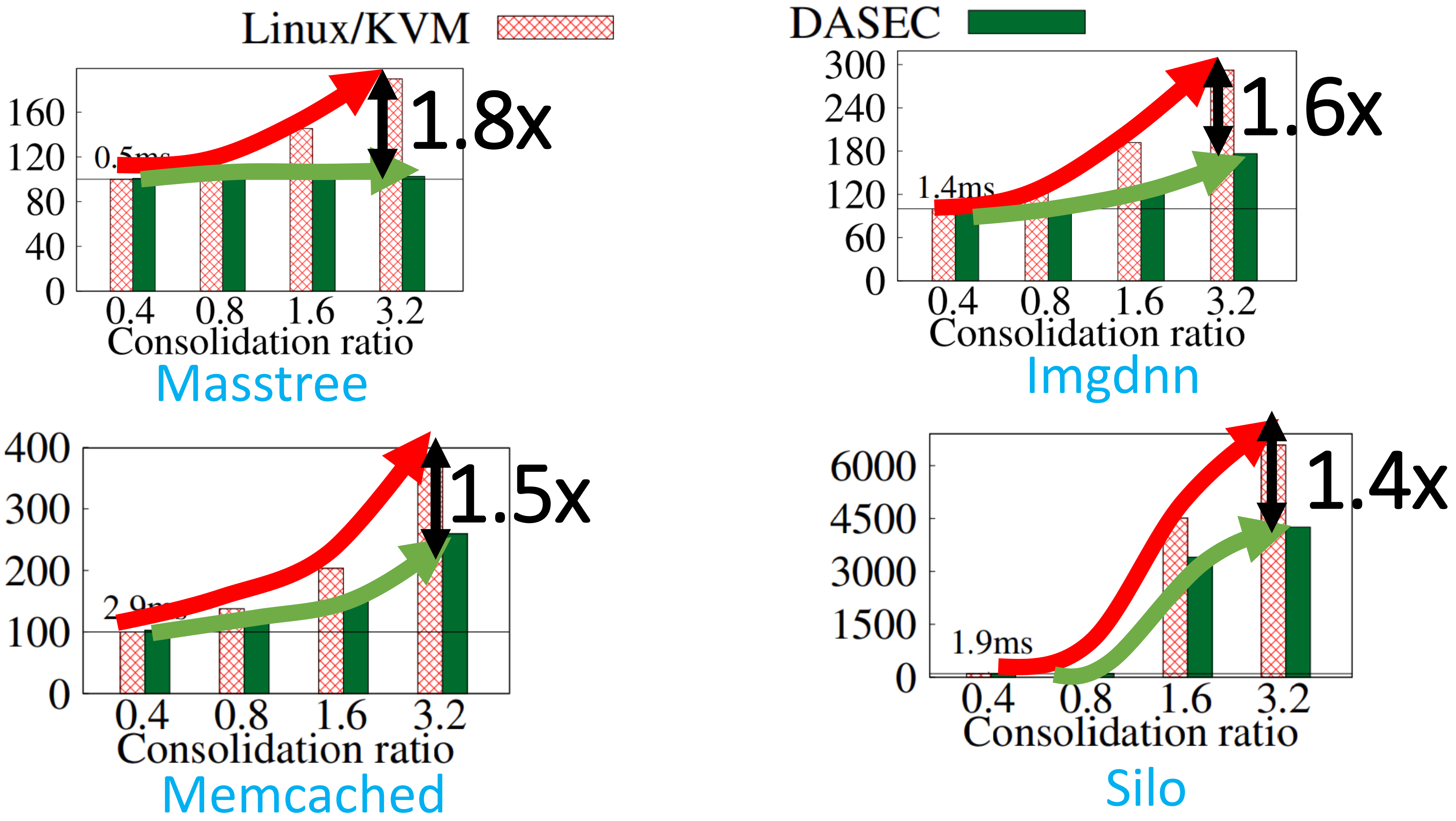
*Latencies relative to Linux/KVM when consolidation ratio is 0.4.



- As consolidation ratio increases, Linux/KVM's mean latency increases much more compared to DASEC.

Mean latency (DASEC vs Linux/KVM)

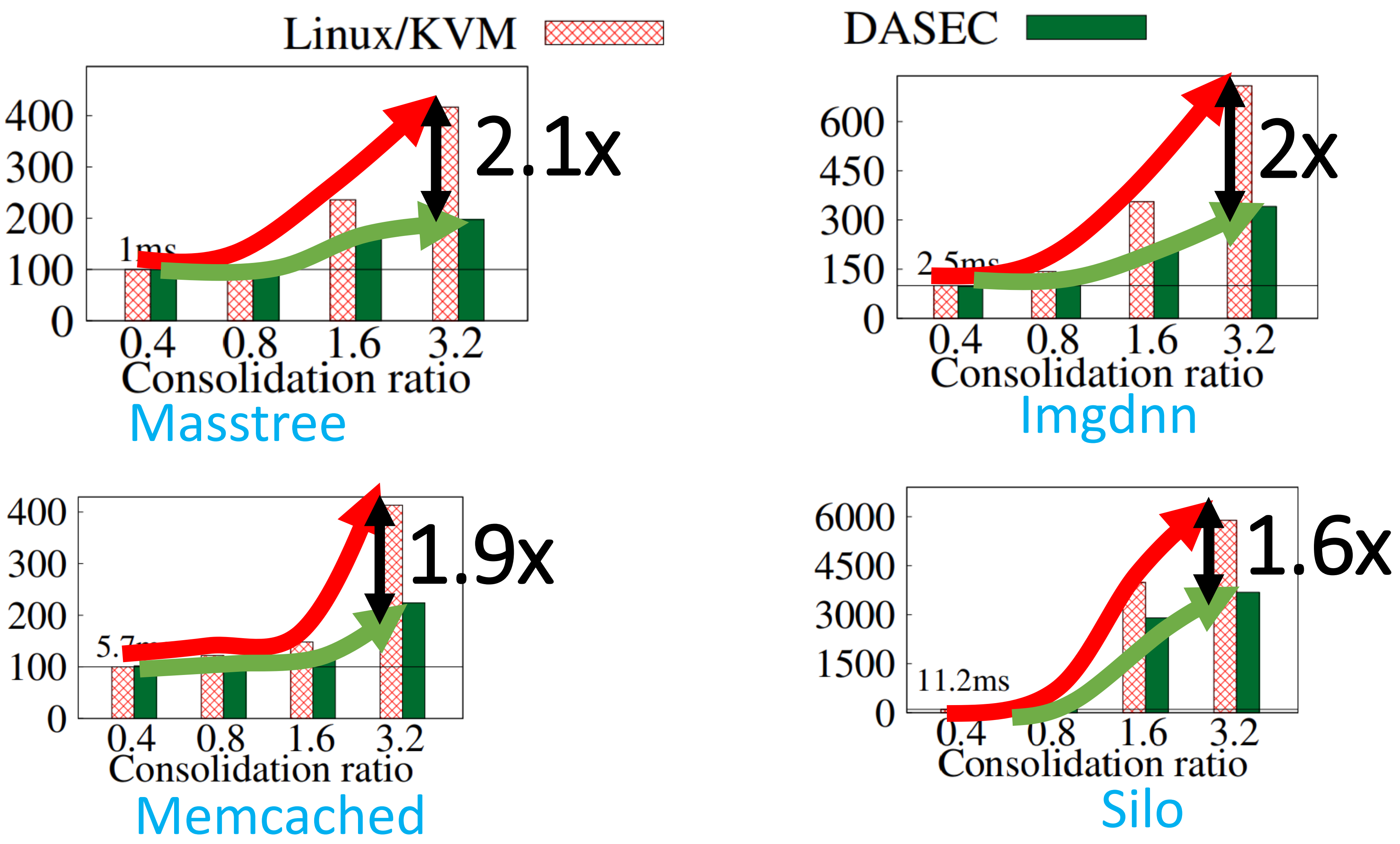
*Latencies relative to Linux/KVM when consolidation ratio is 0.4.



- Compared to Linux/KVM, DASEC reduces mean latencies by 46% on average.

99th tail latency (DASEC vs Linux/KVM)

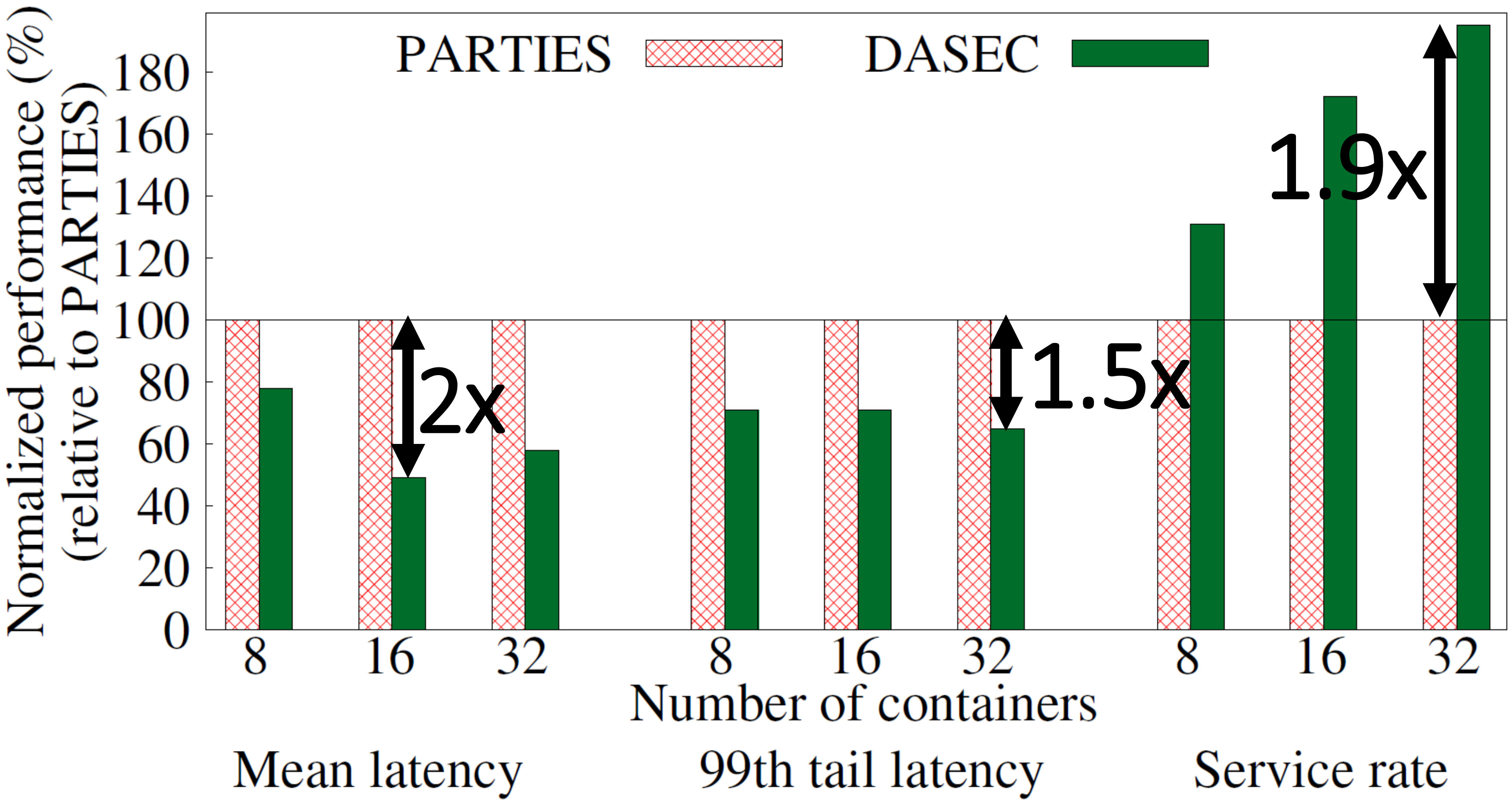
*99th tail Latencies relative to Linux/KVM when consolidation ratio is 0.4.



- Compared to Linux/KVM, DASEC reduces 99th tail latencies by 52% on average.

Performance (DASEC vs PARTIES)

*All containers run the same Masstree workload.



- Compared to PARTIES, DASEC offers up to 51% lower mean latencies, 35% lower 99th tail latency, and 95% more service rate.

Conclusions

- How to efficiently schedule latency sensitive applications with low latency by reducing their mutual interference in edge cloud.
  - Edge cloud is resource constrained and dominated by latency sensitive workloads.
  - Such applications are resource demanding and have dynamic resource usage.
  - Existing cloud approaches are not effective to reduce latency in edge cloud.

Conclusions

- How to efficiently schedule latency sensitive applications with low latency by reducing their mutual interference in edge cloud.
  - Edge cloud is resource constrained and dominated by latency sensitive workloads.
  - Such applications are resource demanding and have dynamic resource usage.
  - Existing cloud approaches are not effective to reduce latency in edge cloud.
- DASEC is an efficient solution for reducing workload mutual interference and latency in edge cloud.
  - Move the interference off the tasks on the critical paths of the workload.
  - Detections and solutions for workload mutual interference in three ways.

Conclusions

- How to efficiently schedule latency sensitive applications with low latency by reducing their mutual interference in edge cloud.
  - Edge cloud is resource constrained and dominated by latency sensitive workloads.
  - Such applications are resource demanding and have dynamic resource usage.
  - Existing cloud approaches are not effective to reduce latency in edge cloud.
- DASEC is an efficient solution for reducing workload mutual interference and latency in edge cloud.
  - Move the interference off the tasks on the critical paths of the workload.
  - Detections and solutions for workload mutual interference in three ways.
- Evaluation shows DASEC can substantially reduce latency compared to related systems in edge cloud.

References

[1] Xu, Mengwei, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. "From cloud to edge: a first look at public edge platforms." In Proceedings of the 21st ACM Internet Measurement Conference, pp. 37-53. 2021.

[2] Chen, Shuang, Christina Delimitrou, and José F. Martínez. "Parties: Qos-aware resource partitioning for multiple interactive services." In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 107-120. 2019.

[3] Duda, Kenneth J., and David R. Cheriton. "Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler." In Proceedings of the seventeenth ACM symposium on Operating systems principles, pp. 261-276. 1999.

[4] Leverich, Jacob, and Christos Kozyrakis. "Reconciling high server utilization and sub-millisecond quality-of-service." In Proceedings of the Ninth European Conference on Computer Systems, pp. 1-14. 2014.

[5] Humphries, Jack Tigar, Neel Natu, Ashwin Chaugule, Ofir Weisse, Barret Rhoden, Josh Don, Luigi Rizzo, Oleg Rombakh, Paul Turner, and Christos Kozyrakis. "ghost: Fast & flexible user-space delegation of linux scheduling." In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, pp. 588-604. 2021.

[6] Delimitrou, Christina, and Christos Kozyrakis. "Bolt: I know what you did last summer... in the cloud." ACM SIGARCH Computer Architecture News 45, no. 1 (2017): 599-613.

Thank you!
Questions?

weiwei.jia@uri.edu
https://www.ele.uri.edu/faculty/weiwei/