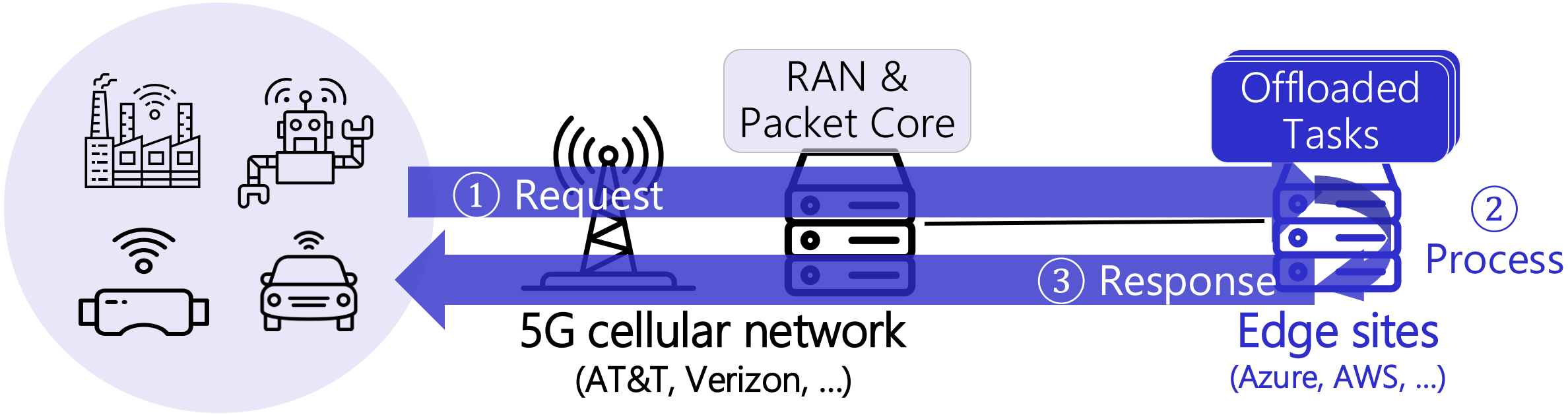
Toward Low-Latency SLO-Aware 5G Multi-Access Edge Computing

Daehyeok Kim

The University of Texas at Austin

Joint work with Xiao Zhang

5G multi-access edge computing (MEC)



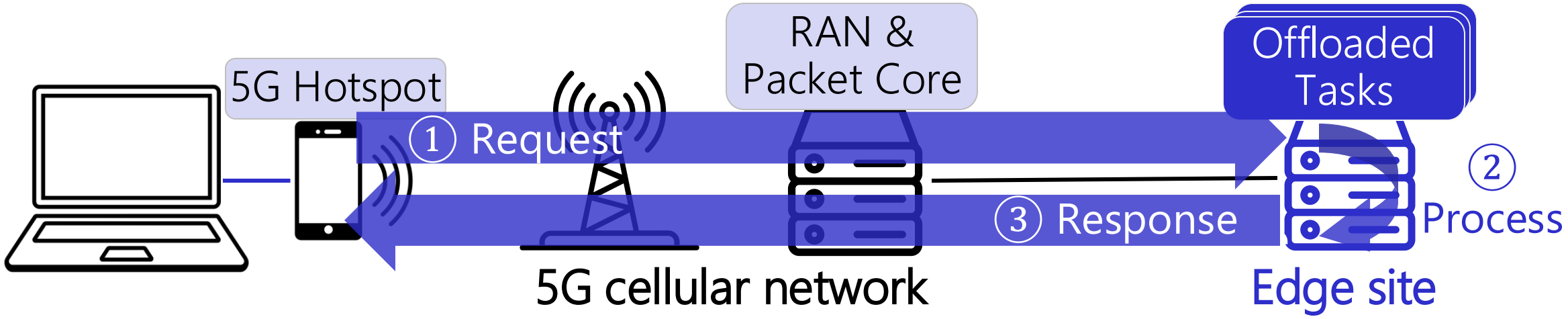
MEC allows for offloading “latency-critical" tasks to the edge

Request-response pattern: Round-trip latency SLO requirements

- E.g., 10-100s ms for AR, smart stadium, video conference, etc.

Does the current MEC deployments provide SLO guarantee?

Measurement study on 5G MEC deployments



5G cellular Networks: Dallas (US), Nanjing (China), and Seoul (S. Korea)

MEC services: AWS Wavelength, Tencent Cloud

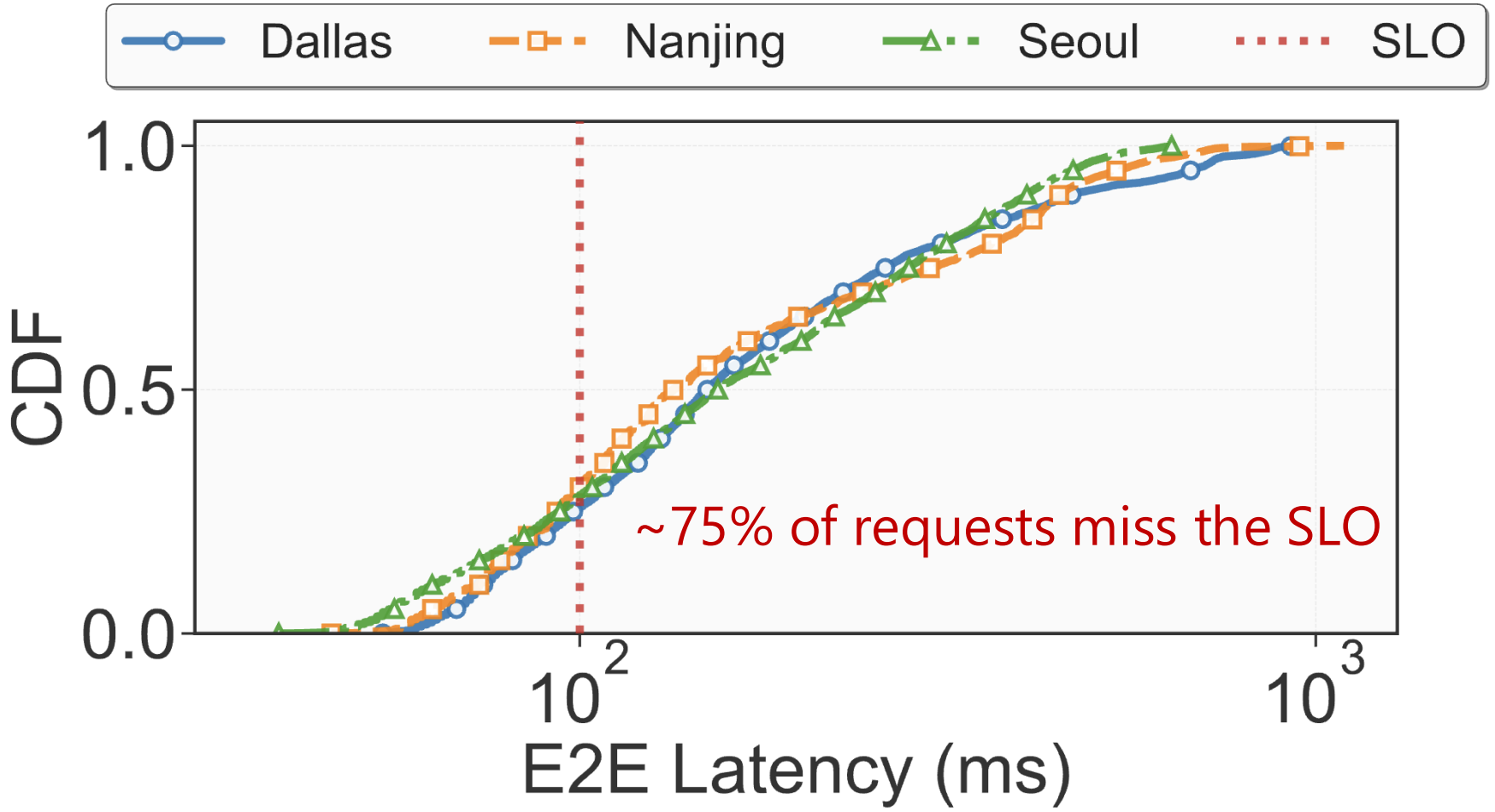
UDP-based for low latency applications

- Smart stadium (SLO 100ms): Offloading "video transcoding"
- Augmented reality (SLO: 100ms): Offloading "video object detection"
- Video conferencing (SLO 150ms): Offloading "video super-resolution"

We measured the end-to-end latency for each application

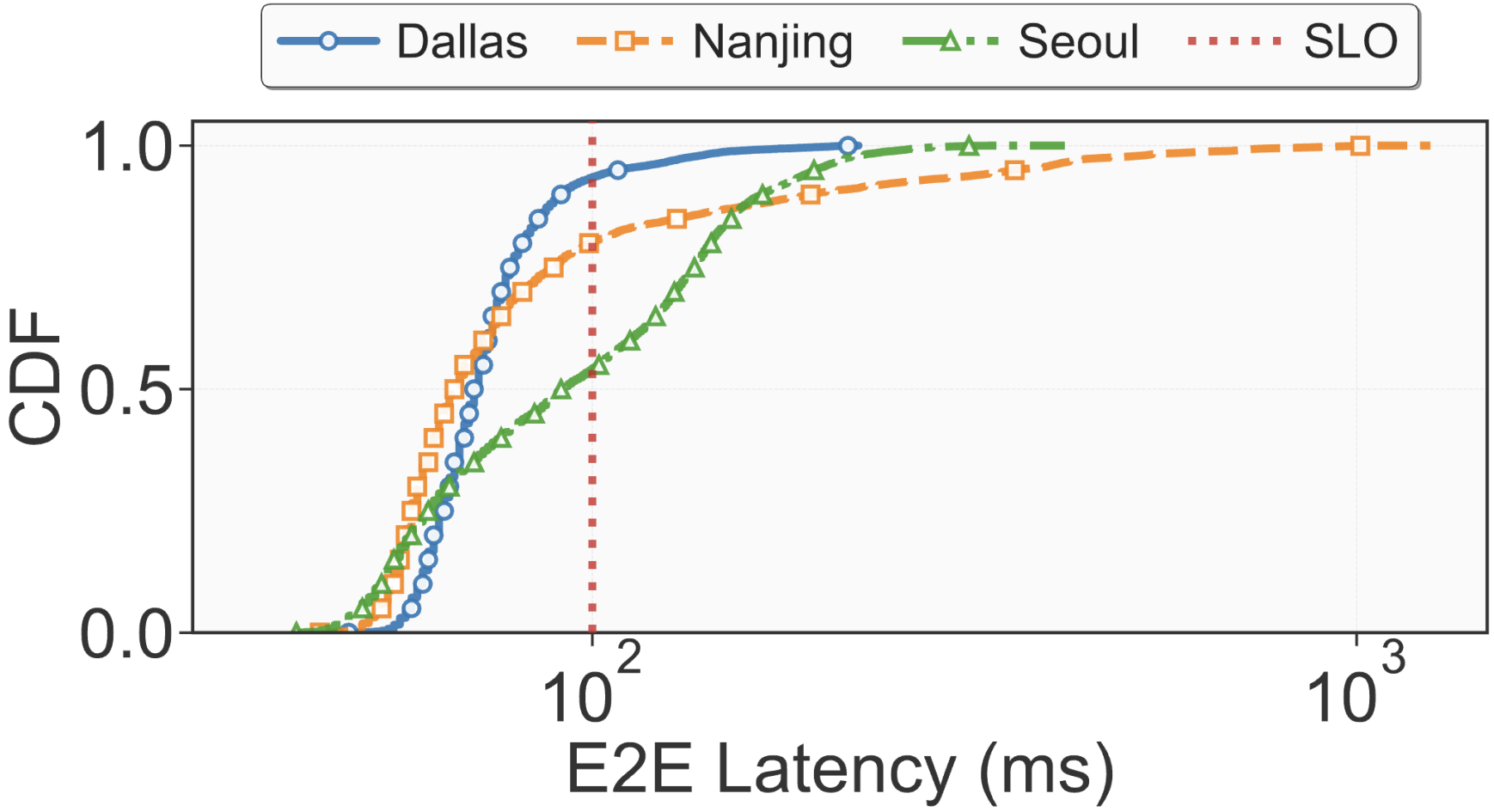
Today's offerings show high latency variability

Smart stadium (video transcoding)

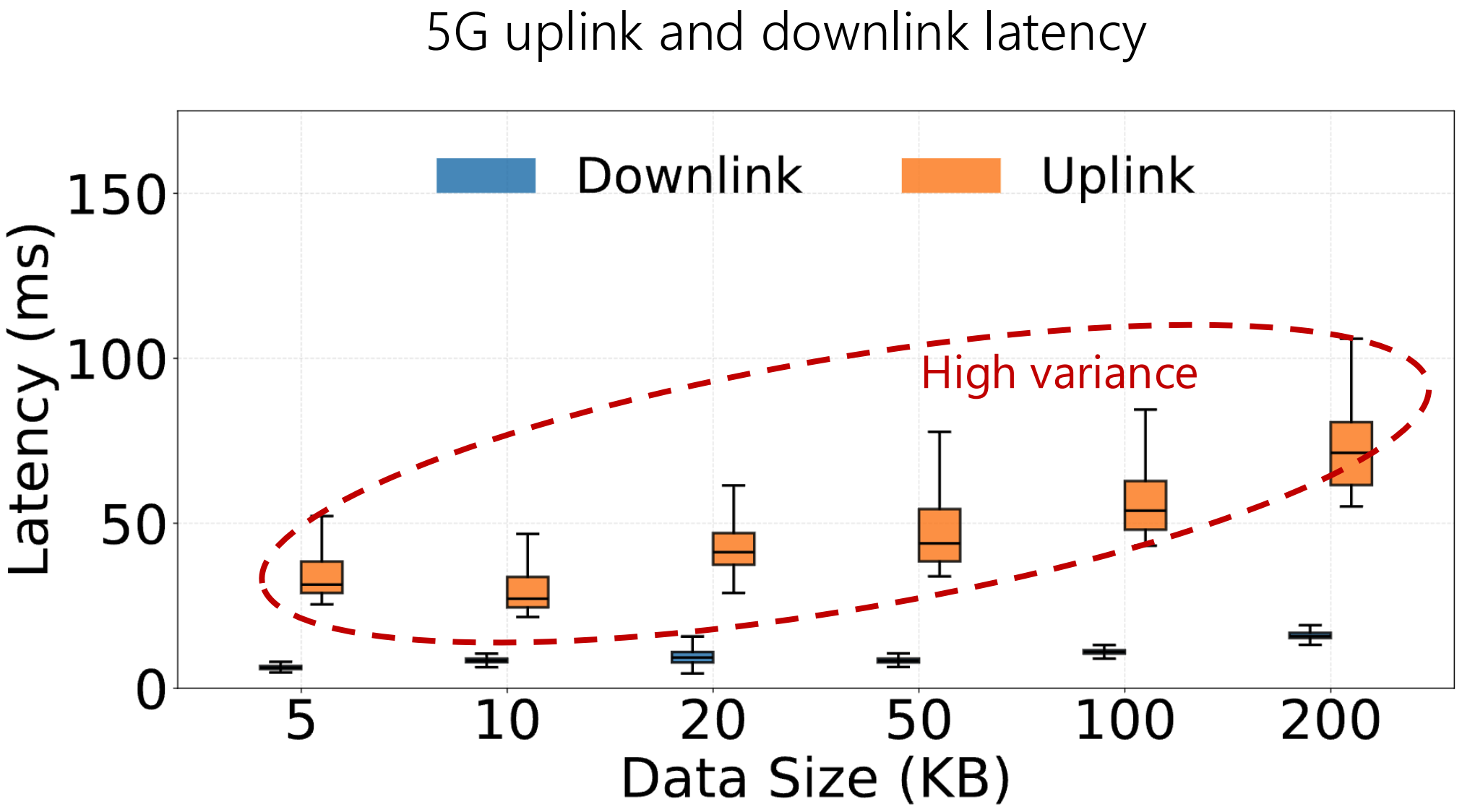


Root cause 1: 5G wireless resource contention

Smart stadium latency *without* edge compute contention

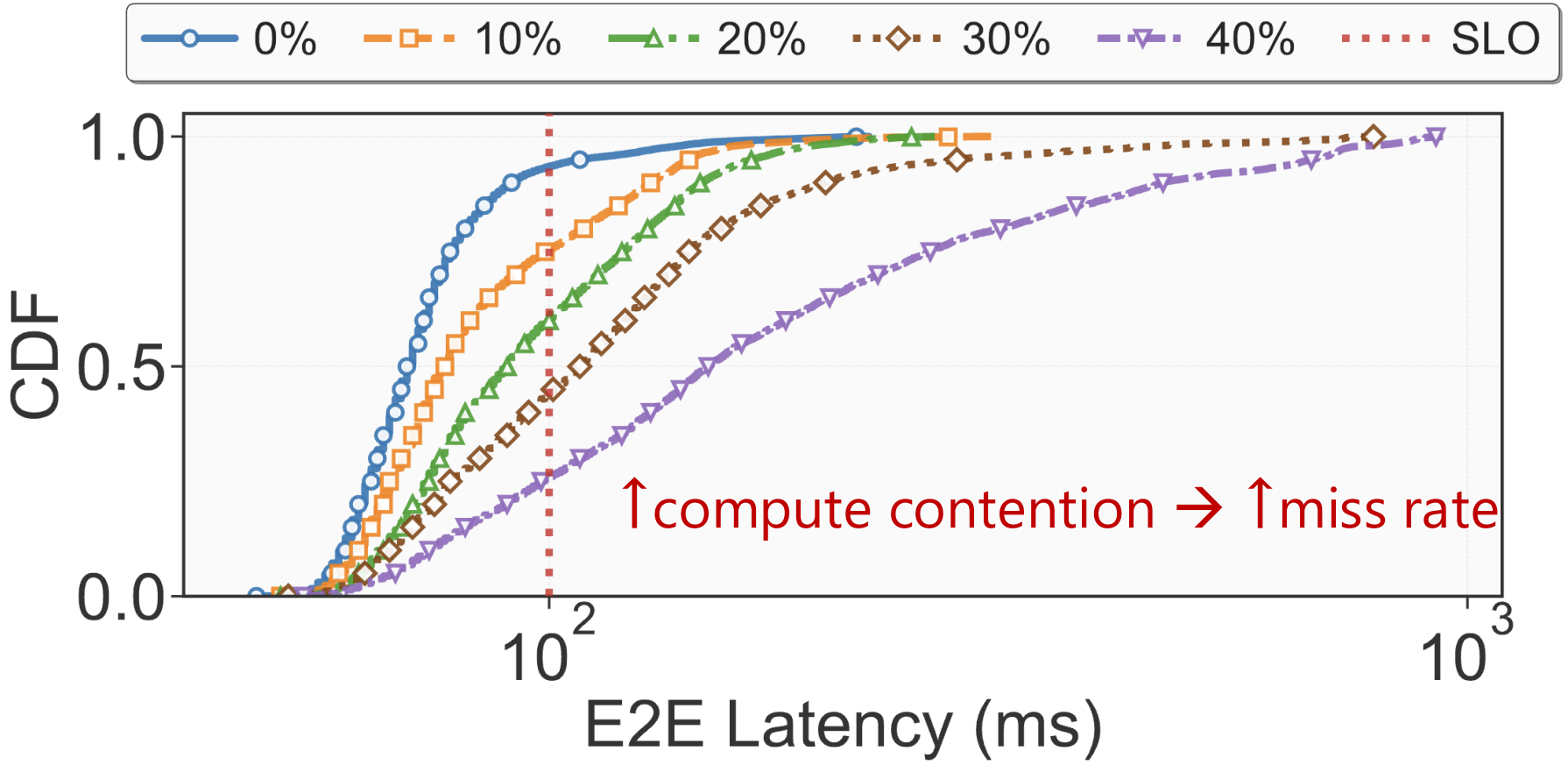


“Uplink” contention is the main factor



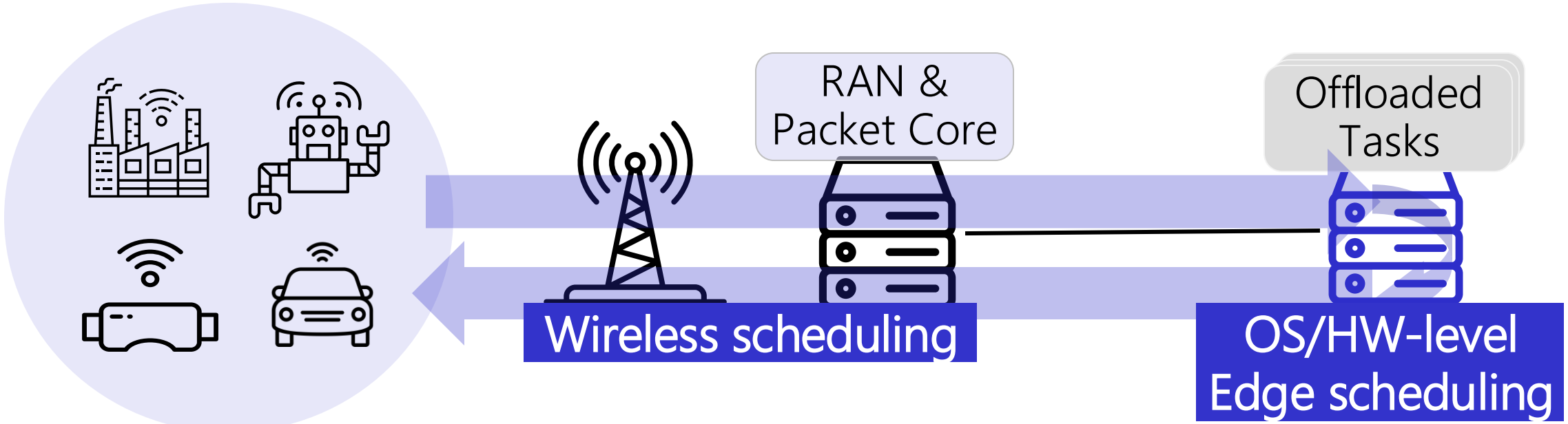
Root cause 2: Edge resource contention

Smart stadium latency under Edge compute contention



Compute resource contention is another key factor for latency variability

SLO-unaware resource scheduling is the key issue

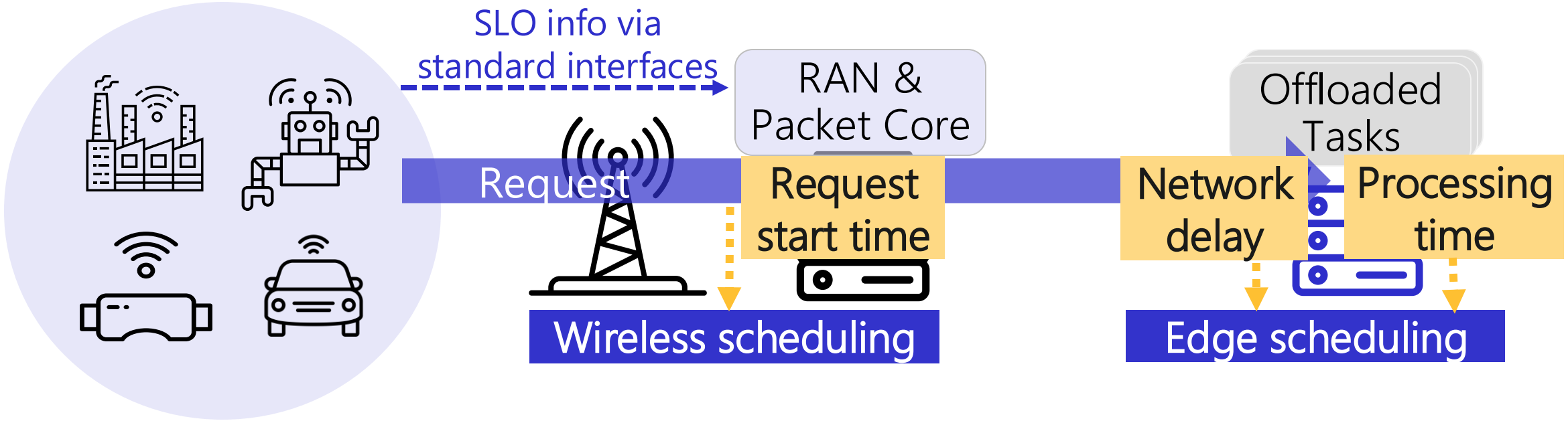


Resource schedulers consider fairness and/or resource utilization, but not application SLOs

E.g., Proportional fair (RAN) and CFS/EEVDF (Edge)

SMEC:
Practical, SLO-aware resource management for 5G MEC

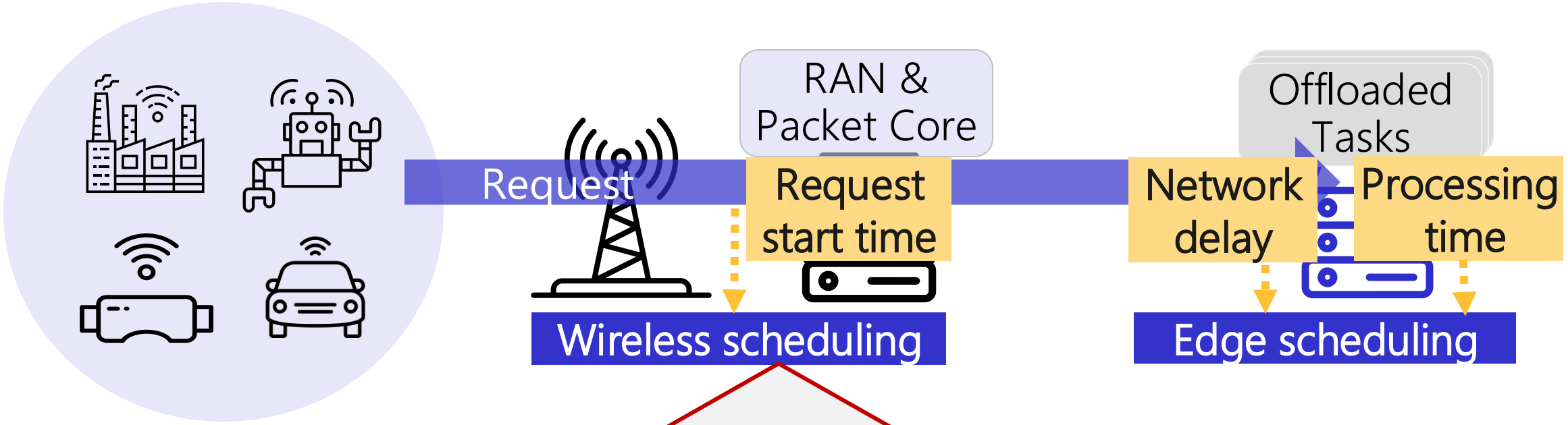
Goal: Improving SLO satisfaction rate *without* RAN-edge coordination



Our approach: RAN and Edge estimate the **deadline** for requests *independently* and accelerate near-deadline requests!

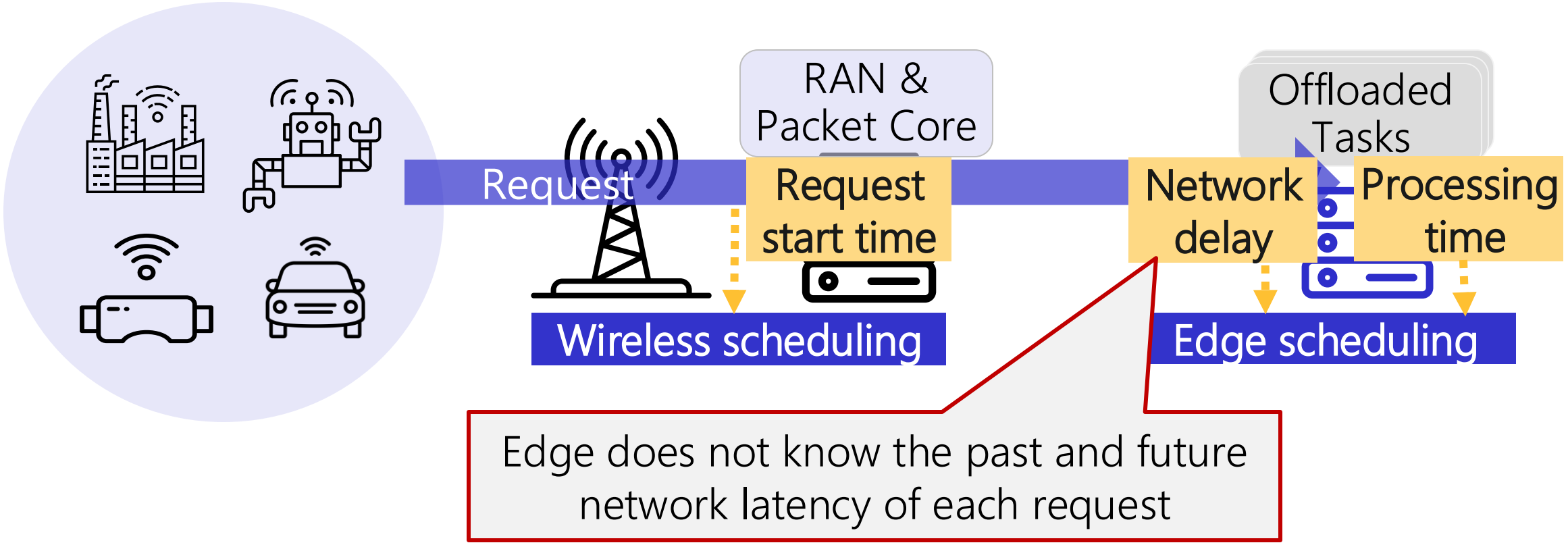
Is this vision achievable?

Challenge 1: How to know request start time?

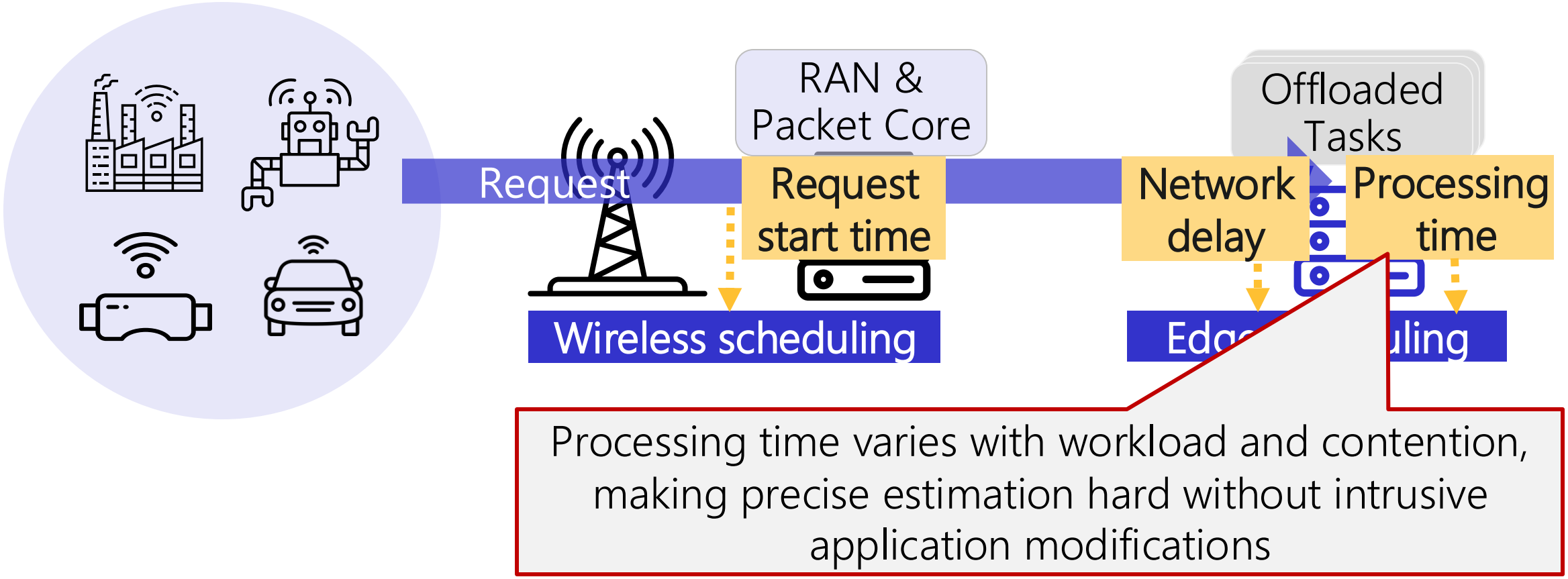


RAN lacks applications' request-level visibility
It has tight timing (500μs) requirements for scheduling

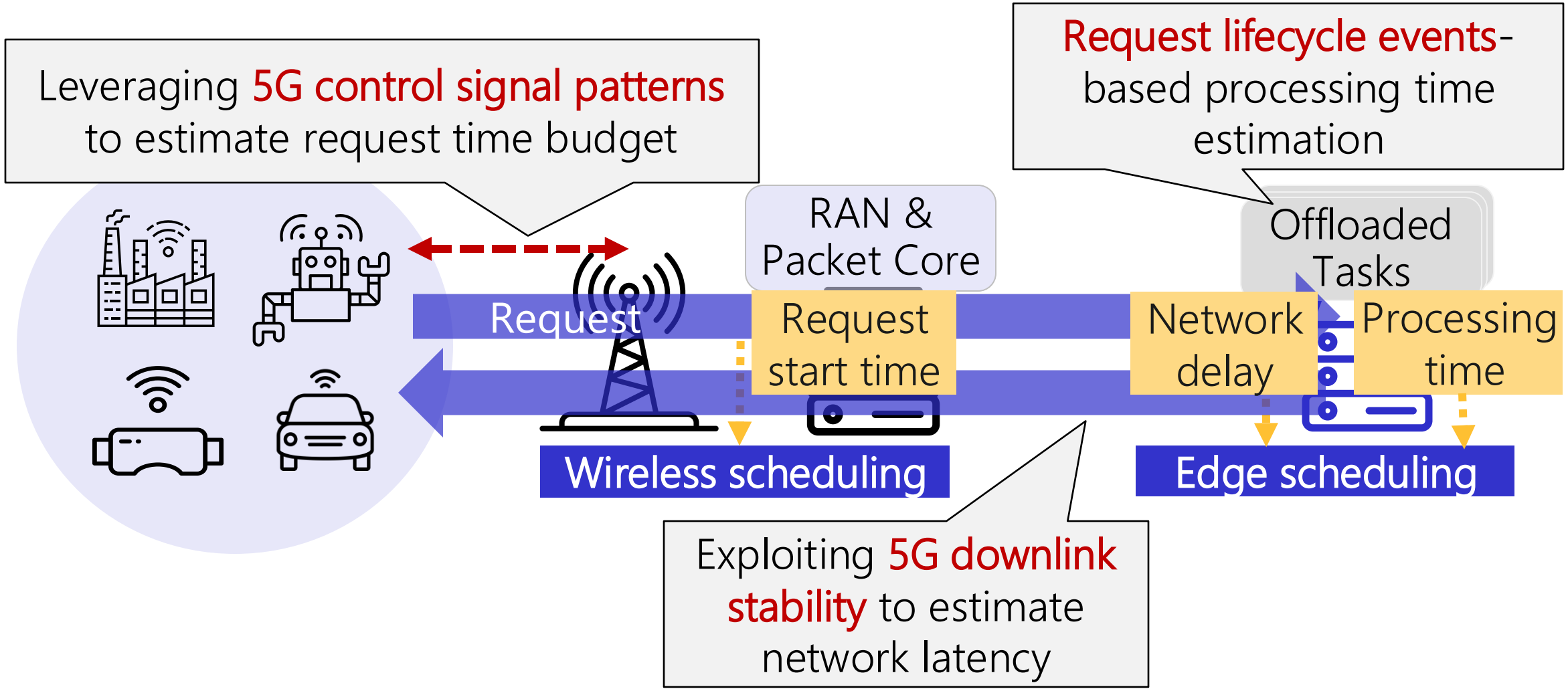
Challenge 2: How to estimate the network delay?



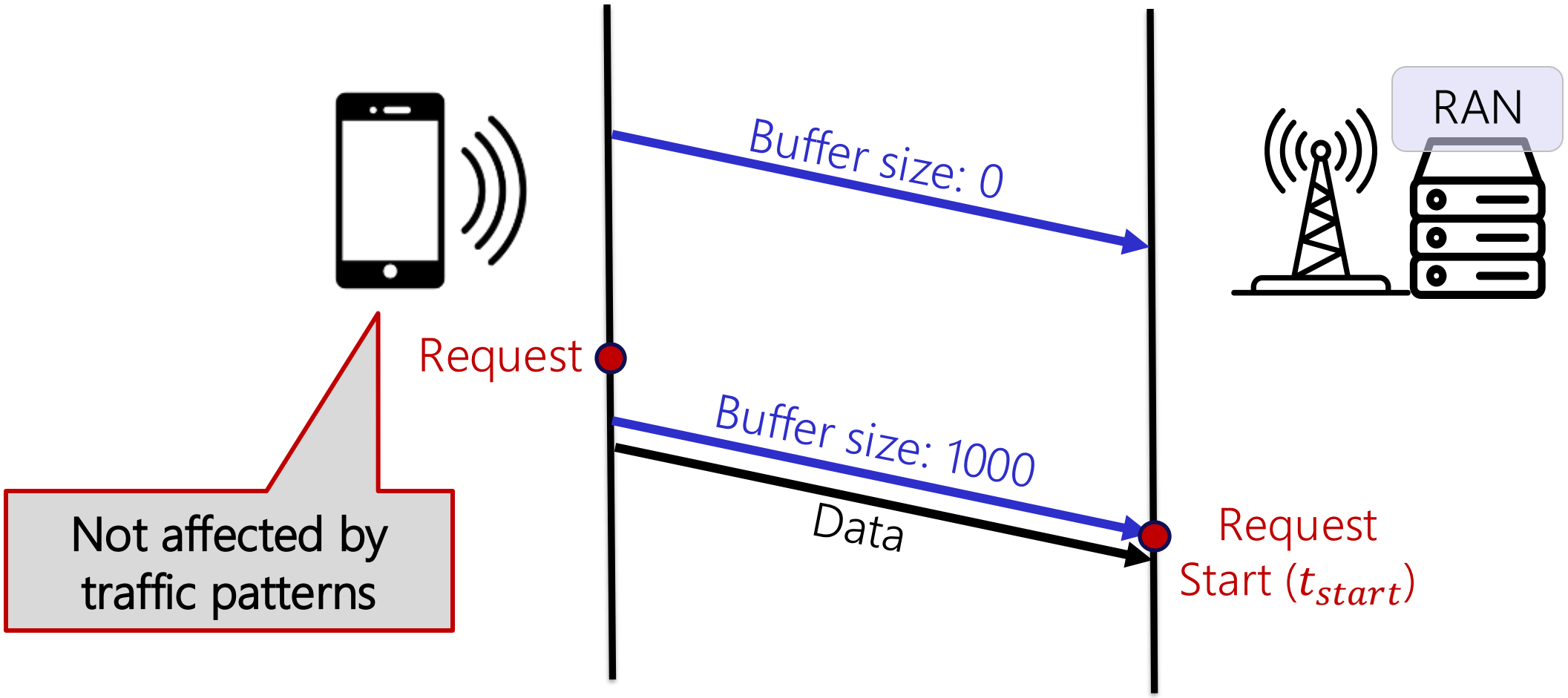
Challenge 3: How to estimate processing time?



Key Insight:
Exploiting natural signals and properties in MEC



Idea 1: Using standard 5G control signals to infer request start time



$$t_{budget}^{RAN} = SLO - (t_{current} - t_{start})$$

Deadline-aware wireless scheduling

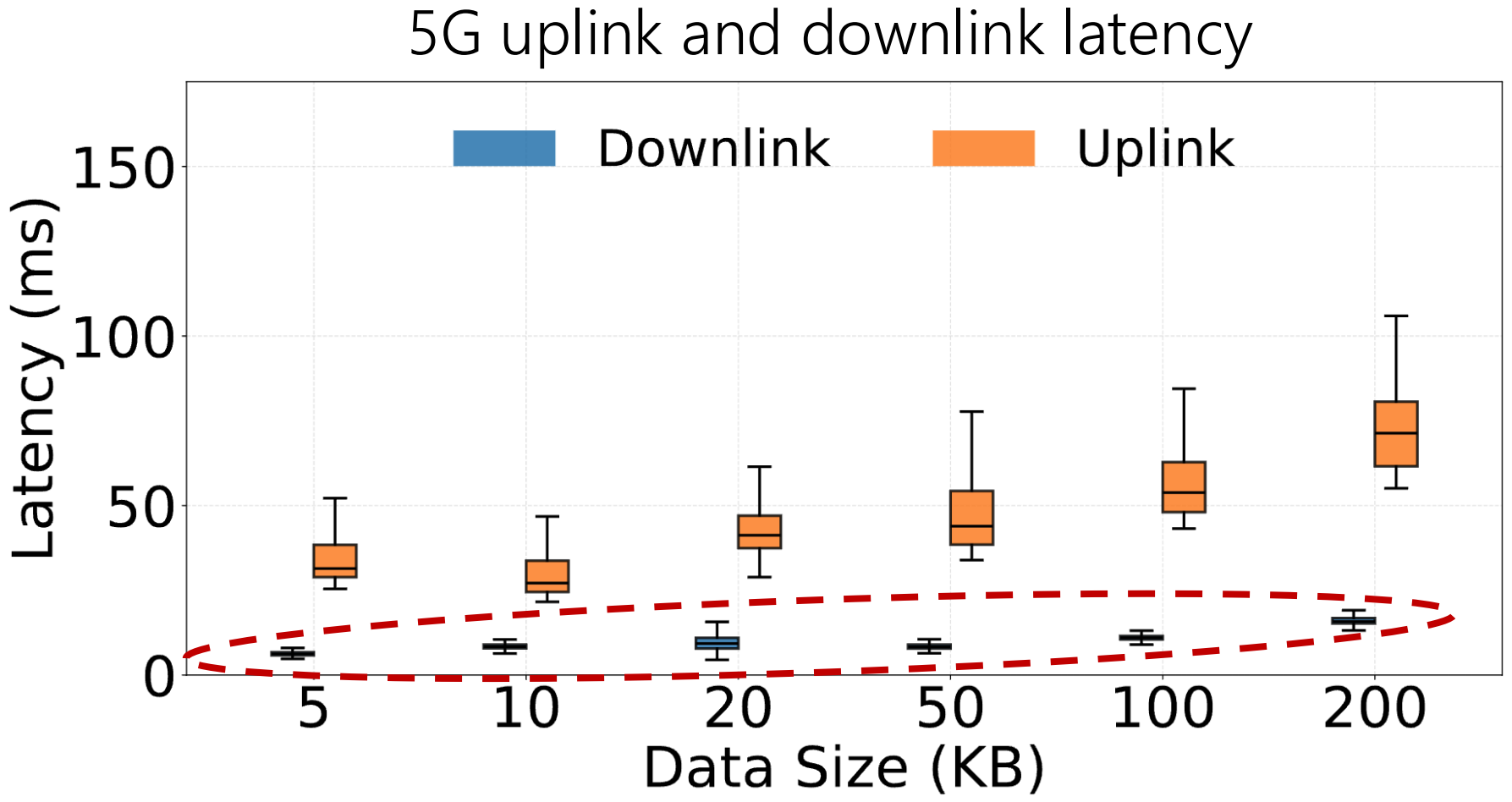
Wireless traffic includes latency-critical (LC) and best-effort (BE) requests

- Among LC requests: Prioritize requests with smaller remaining time budget
- Between LC and BE requests: Prioritize LC requests over BE requests
- Avoid starvation of BE requests: Prioritize SR-triggered resource allocation than normal scheduling

Implemented as a new MAC scheduler in the OAI 5G stack

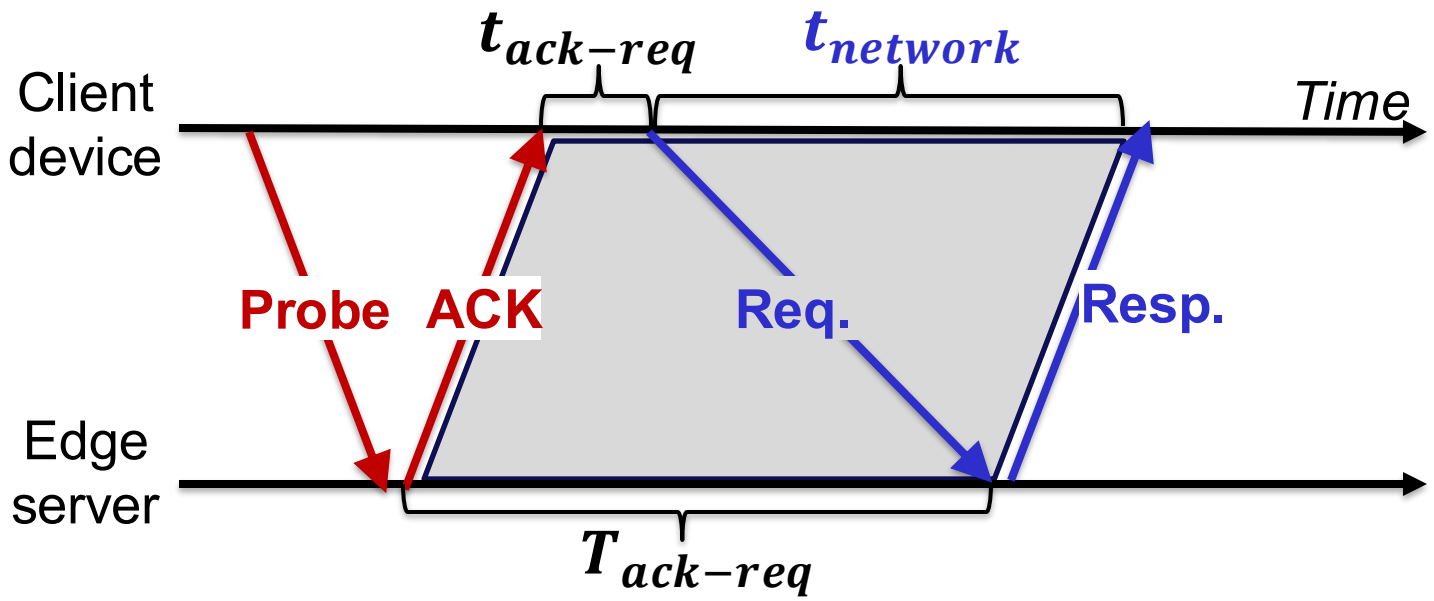
- Can be ported to other RAN stacks

Idea 2:
Exploiting downlink stability for network latency estimation



Our approach: Leveraging downlink stability to estimate network latency via a lightweight probing without time synchronization

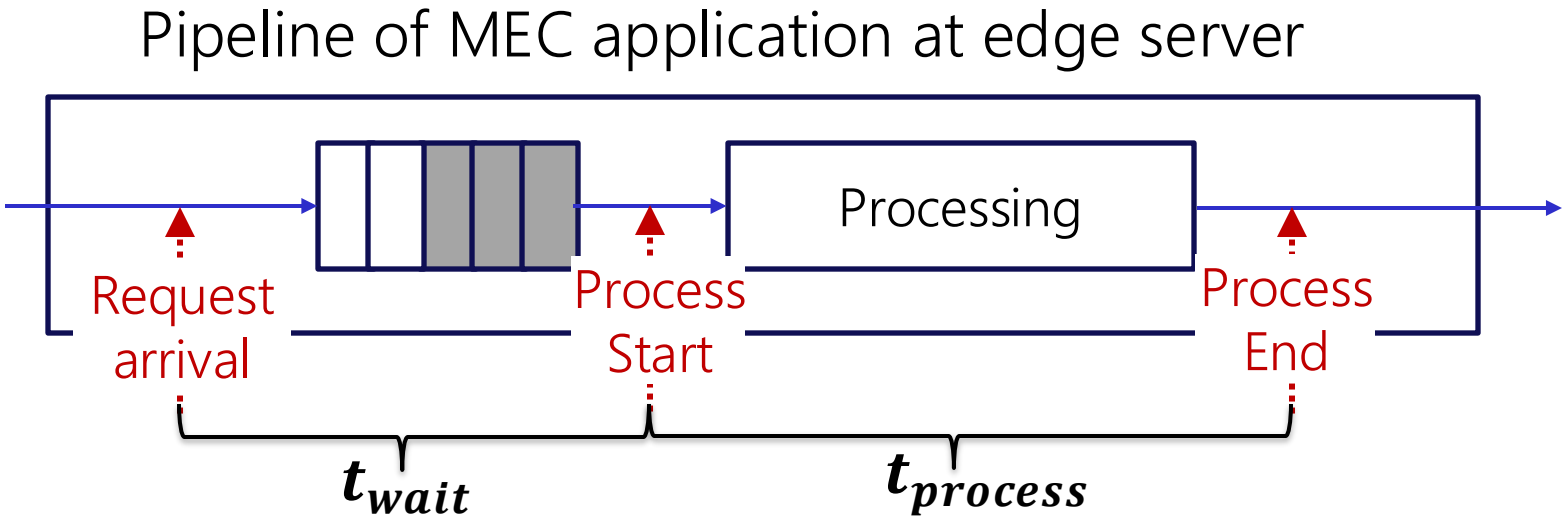
Network latency estimation via SMEC probing protocol



$$t_{network} = T_{ack-req} - t_{ack-req}$$

Low overhead, with Probe–ACK exchanges occurring every few seconds

Idea 3: Leveraging request lifecycle events for processing time estimation



$$t_{process}^{estimate} = median(t_{process}^{0}, t_{process}^{1}, \cdots, t_{process}^{k})$$

$$t_{budget}^{edge} = \mathbf{SLO} - t_{network} - t_{wait} - t_{process}^{estimate}$$

Our approach: SMEC API to report events without intrusive app changes

Deadline-aware proactive edge scheduling

$$t_{budget}^{edge} = \mathbf{SLO} - t_{network} - t_{wait} - t_{process}^{estimate}$$

Decide request urgency by comparing t_{budget}^{edge} with a per-app threshold

CPU Scheduling: Allocate an additional core to apps with urgent requests

- Linux's `sched_setaffinity` system call

GPU scheduling: Increase the priority-level for urgent requests

- CUDA stream priority used by Orion [Eurosys'24]

Early drop: Discard requests with overly tight deadlines

Evaluation setup

Testbed

- 5G network: OAI 5G stack
- Edge server: Intel Xeon CPUs + NVIDIA L4 GPU
- Client emulator: 12 user devices (AmariSoft UE simbox)

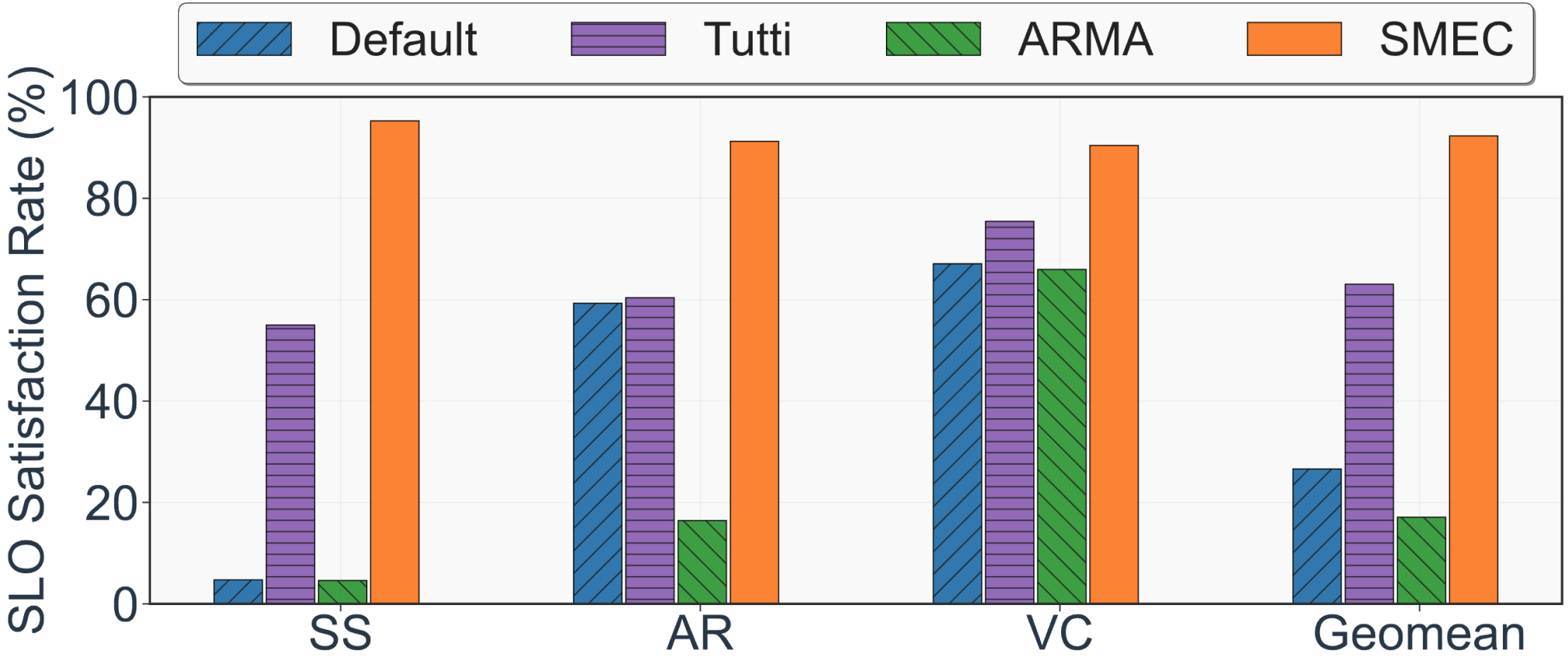
Applications (Three latency-critical & One best-effort)

- Smart Stadium (Video transcoding; CPU-heavy),
- Augmented Reality (Object detection; GPU-heavy)
- Video Conferencing (Super-resolution; GPU-heavy)
- File transfer (Best effort)

Application workloads

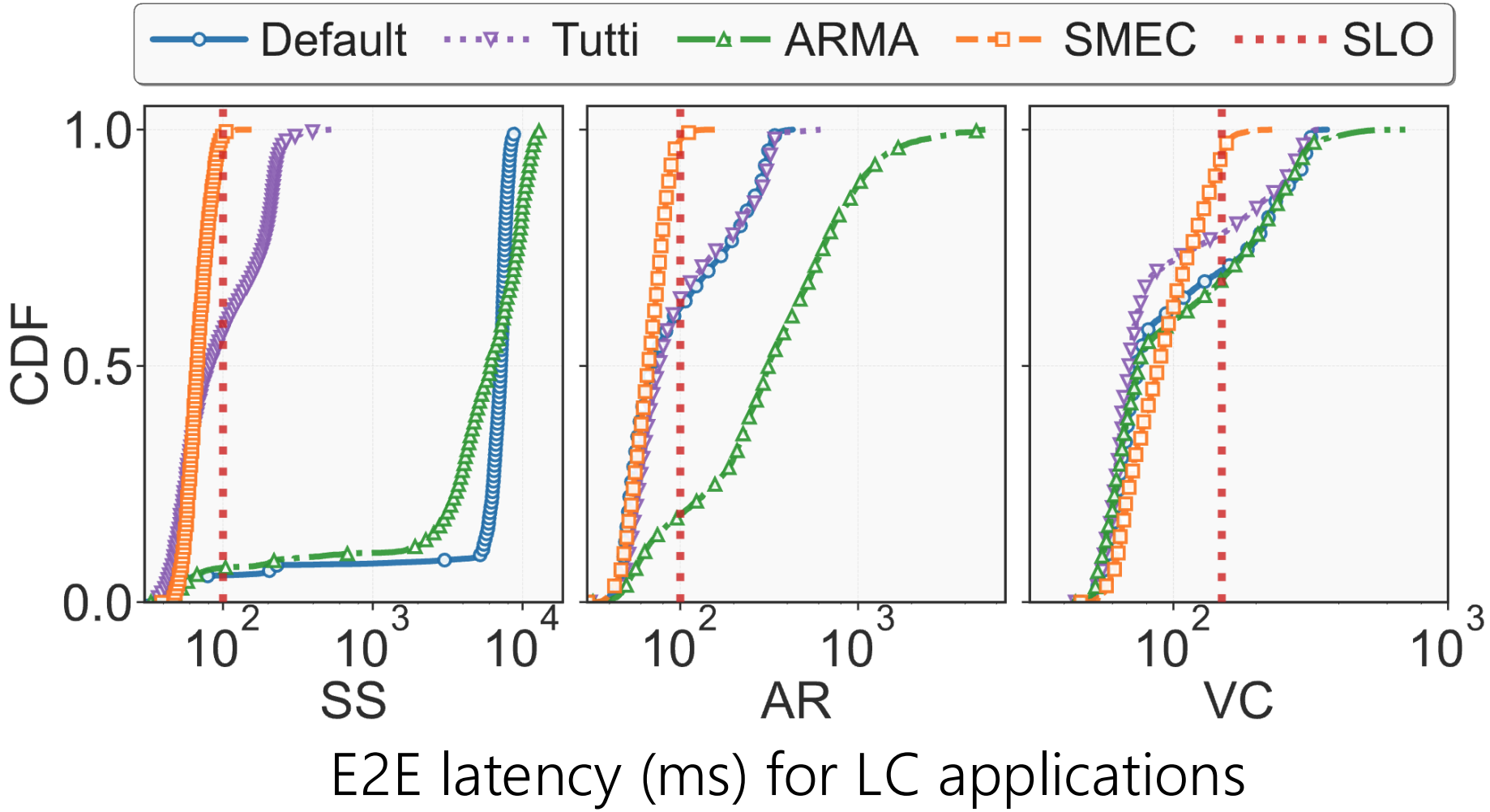
- Static workload: Fixed number of clients sending video streams at a constant rate
- Dynamic workload: Dynamic number of clients sending dynamic video streams

SMEC achieves consistently high SLO satisfaction rate under dynamic workload



SMEC achieves 90-96% SLO satisfaction rate

SMEC achieves low tail latency under dynamic workload



SMEC reduces tail latency by up to 122x!

Summary

Today's 5G MEC deployment suffers from high latency variability

SMEC: SLO-aware resource scheduling framework for MEC

- Exploiting natural signals and properties in 5G MEC to estimate deadline of each request
- Requiring no infrastructure and minimal application changes

SMEC achieves 90-96% SLO satisfaction and reduces tail latency by up to 122x compared to existing approaches