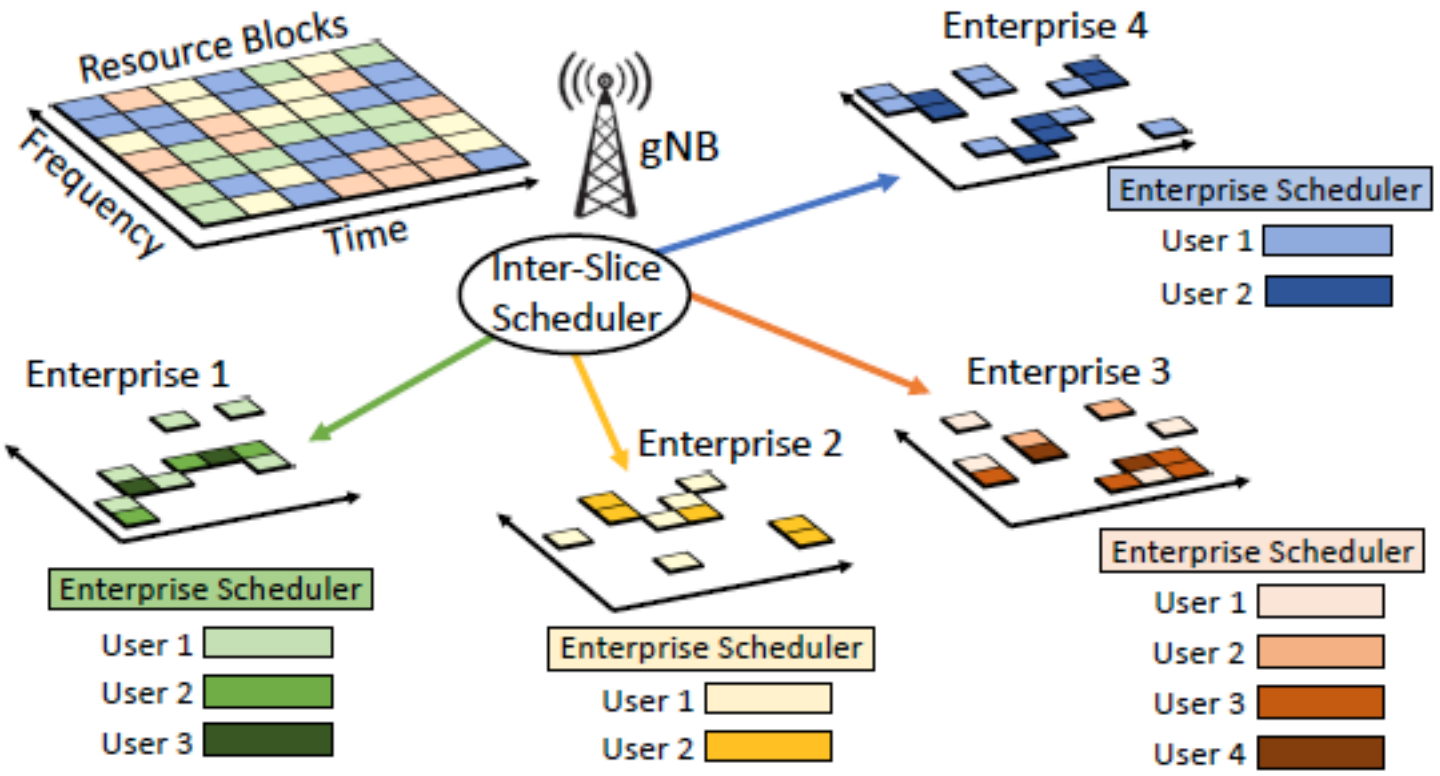
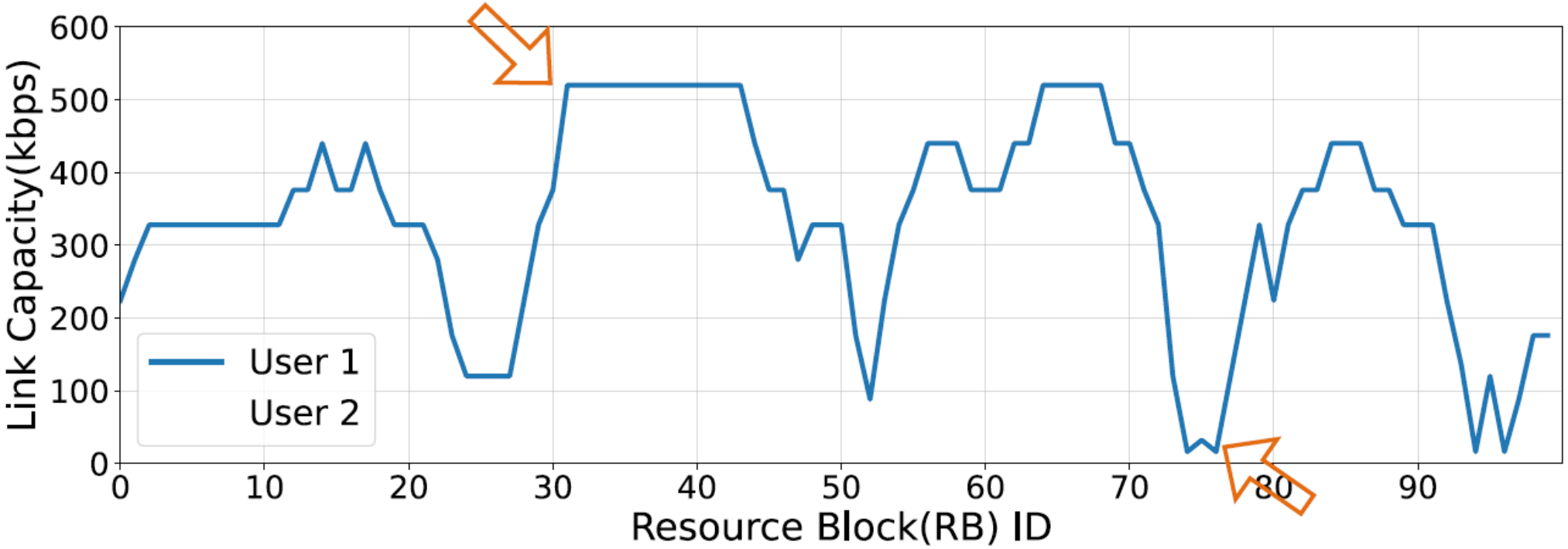
Channel-Aware 5G RAN Slicing with Customizable Schedulers

Zijian Qin

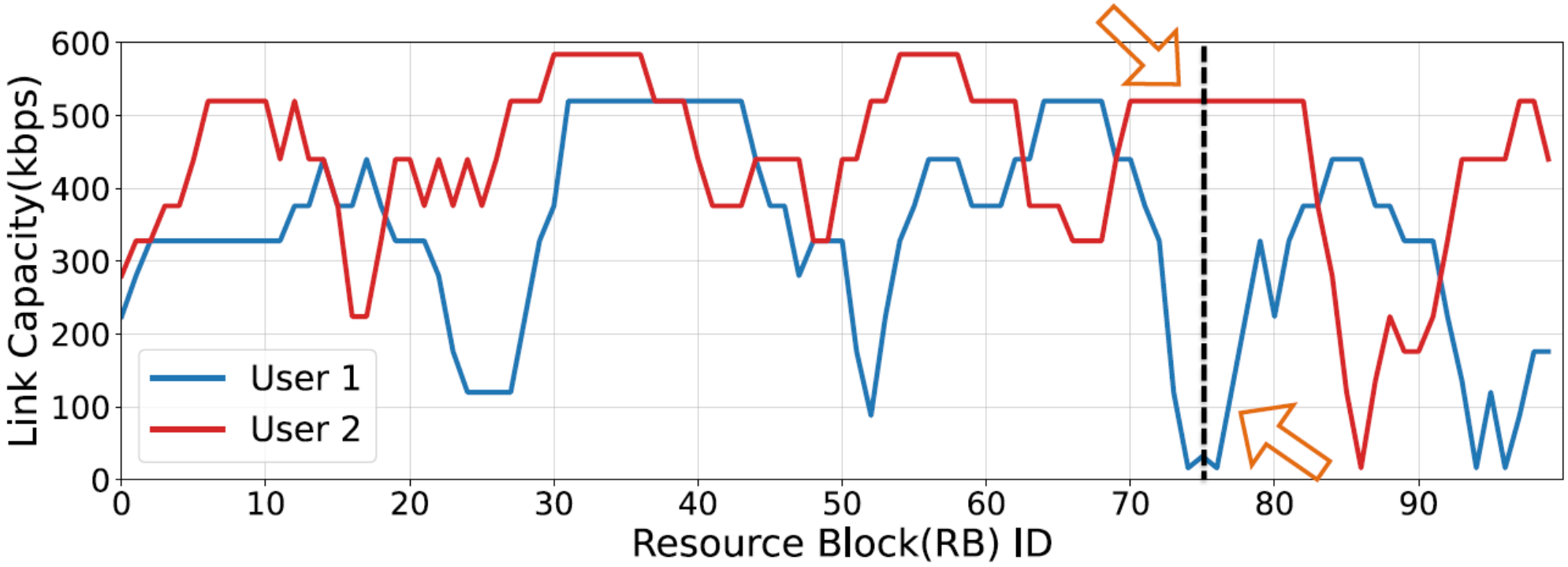
5G RAN Slicing



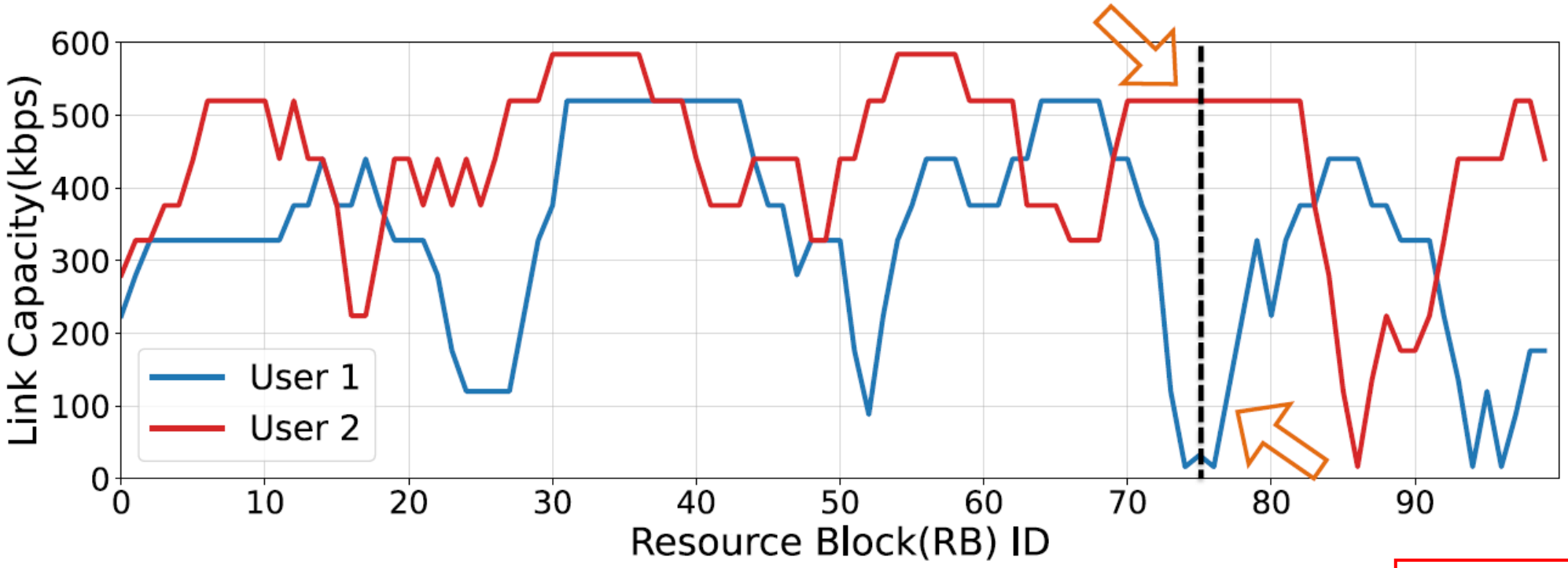
Need for Channel-Awareness



Need for Channel-Awareness



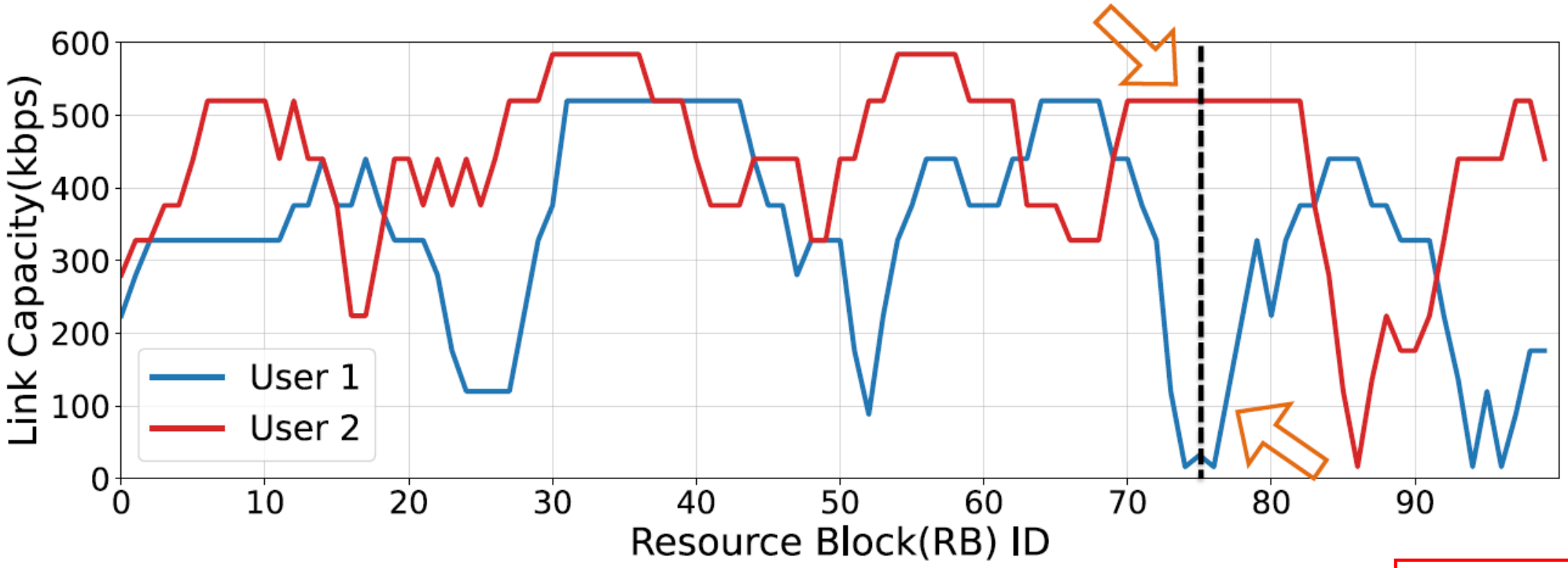
Need for Channel-Awareness



frequency selective fading

· For a UE, the quality of wireless channel varies among different frequency bands.

Need for Channel-Awareness



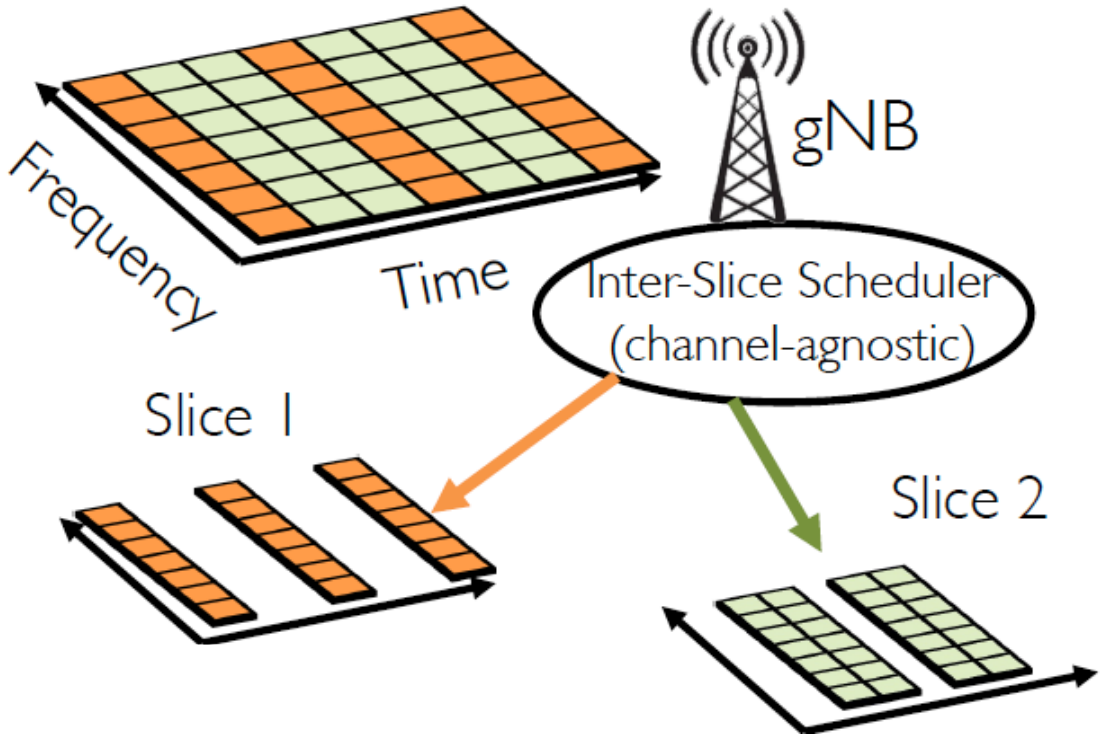
frequency selective fading

· For a UE, the quality of wireless channel varies among different frequency bands.

· The quality of wireless channel may complement among different UEs.

Existing RAN Slicing Techniques

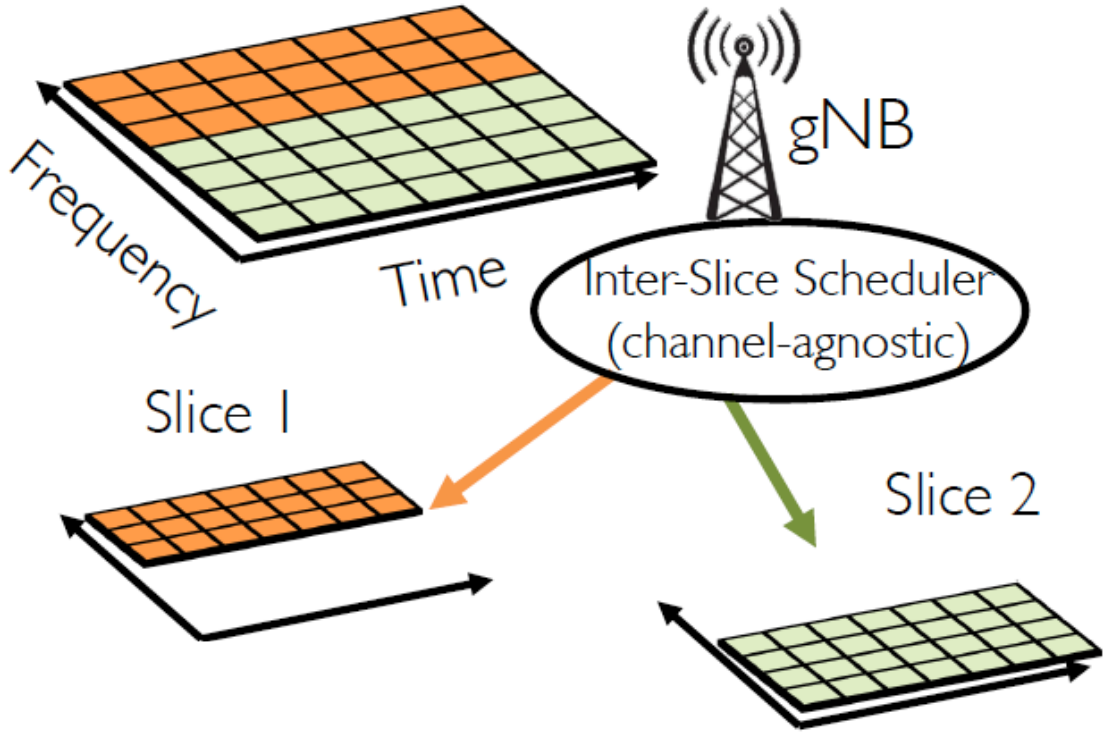
- Inter-slice scheduling--divides RBs across slices: channel-agnostic
- Enterprise scheduling--allocates its RBs to its users: channel-aware



Example1: NVS [21] allocates RBs to slices in the time domain

Existing RAN Slicing Techniques

- Inter-slice scheduling--divides RBs across slices: channel-agnostic
- Enterprise scheduling--allocates its RBs to its users: channel-aware



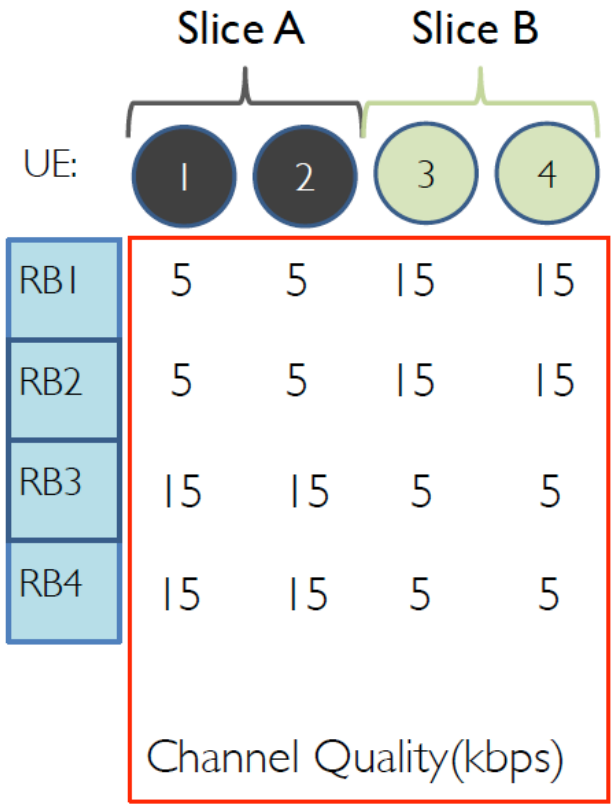
Example2: Flare [29] allocates RBs to slices in the frequency domain

Existing RAN Slicing Techniques

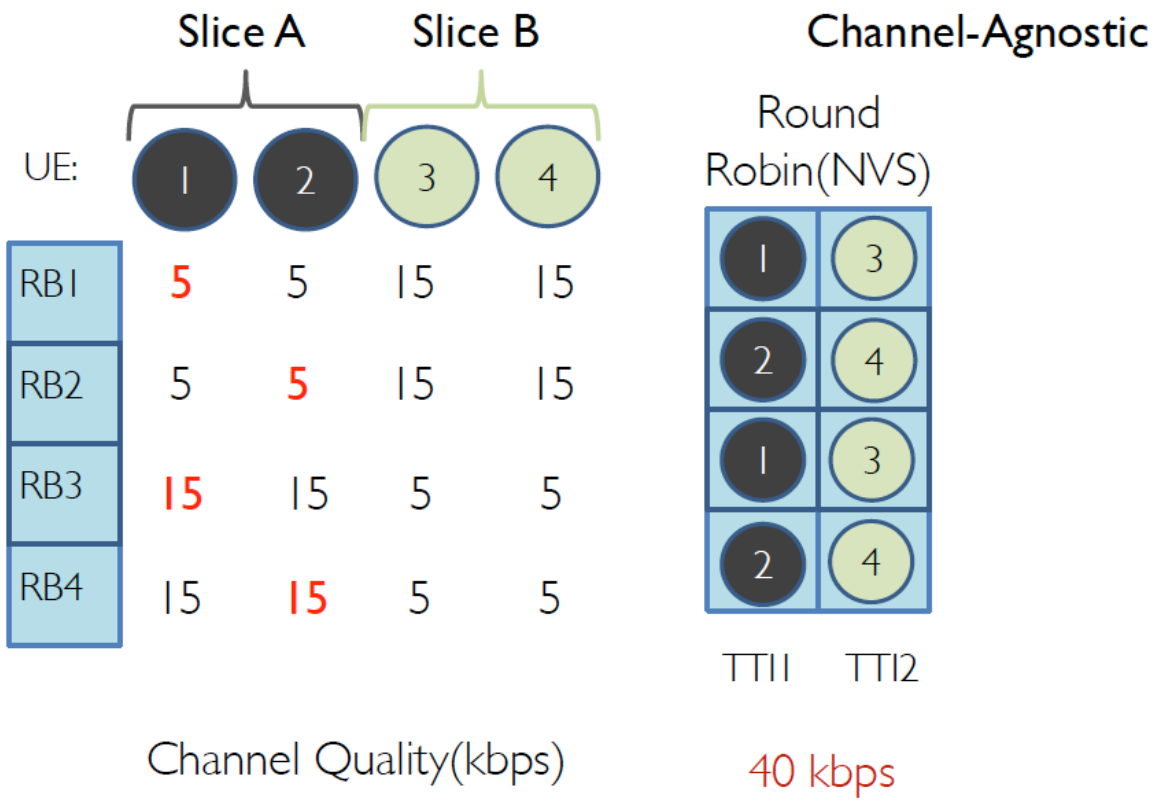
- Inter-slice scheduling--divides RBs across slices: channel-agnostic
- Enterprise scheduling--allocates its RBs to its users: channel-aware

- Pros: decouple inter-slice and enterprise scheduling
- Cons: inter-slice scheduling is channel-agnostic

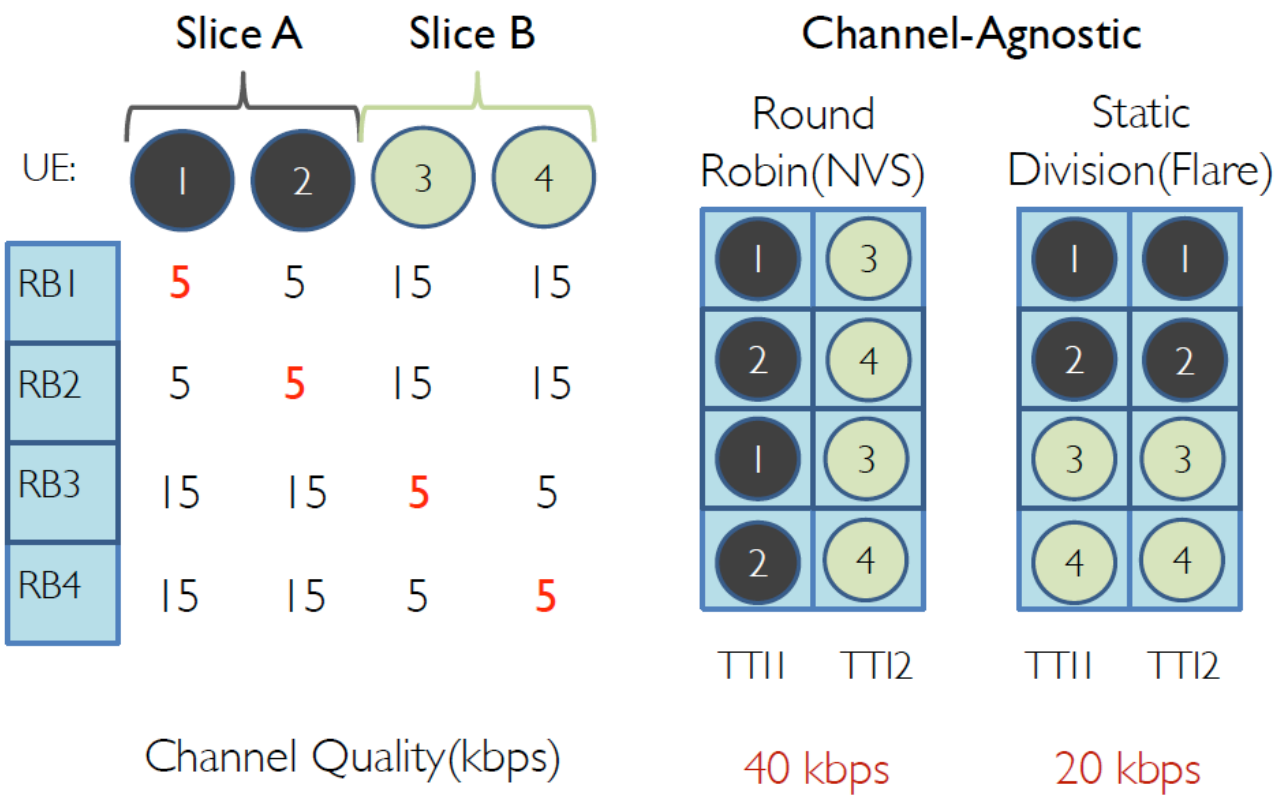
Why Do We Need Channel-Aware Inter-slice Scheduling



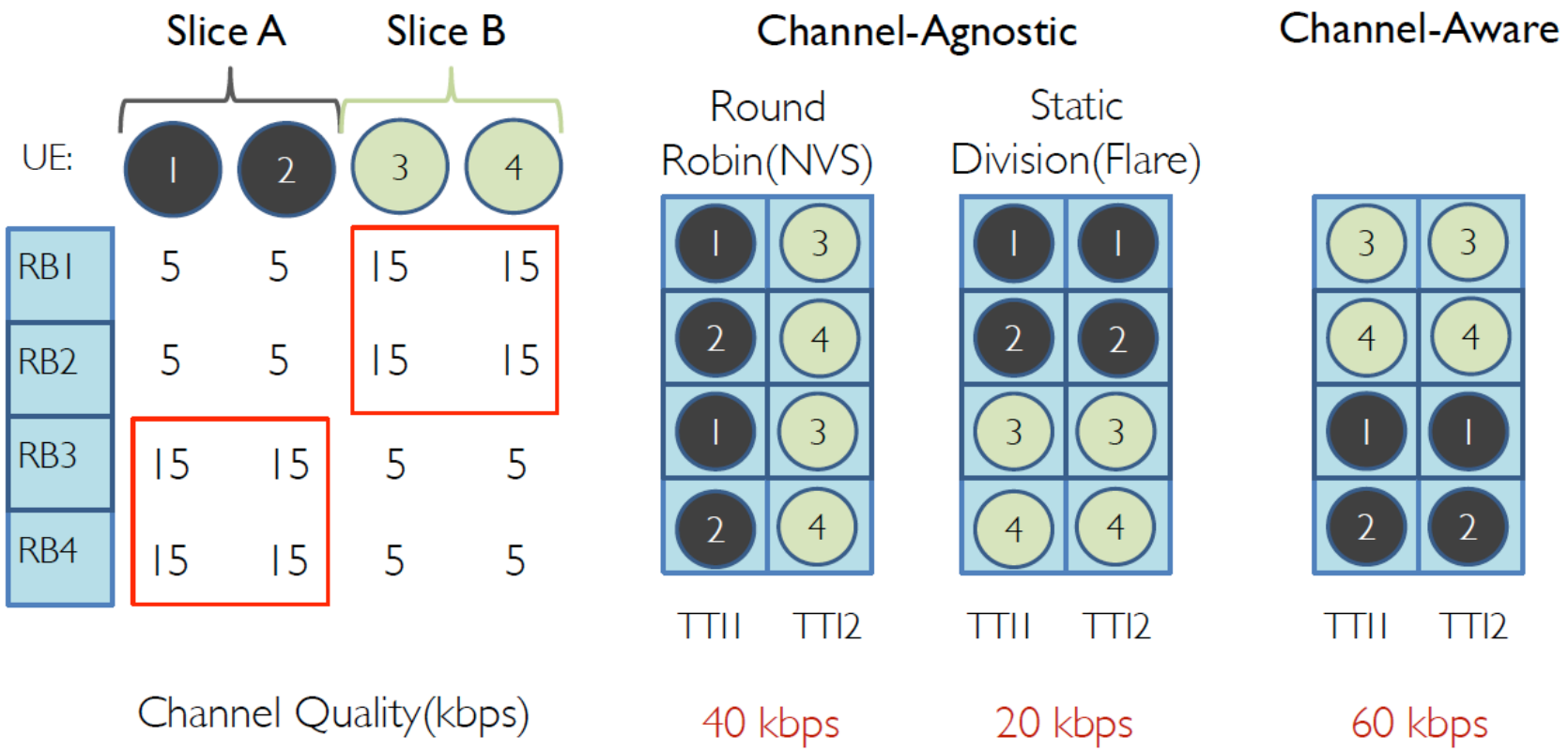
Why Do We Need Channel-Aware Inter-slice Scheduling



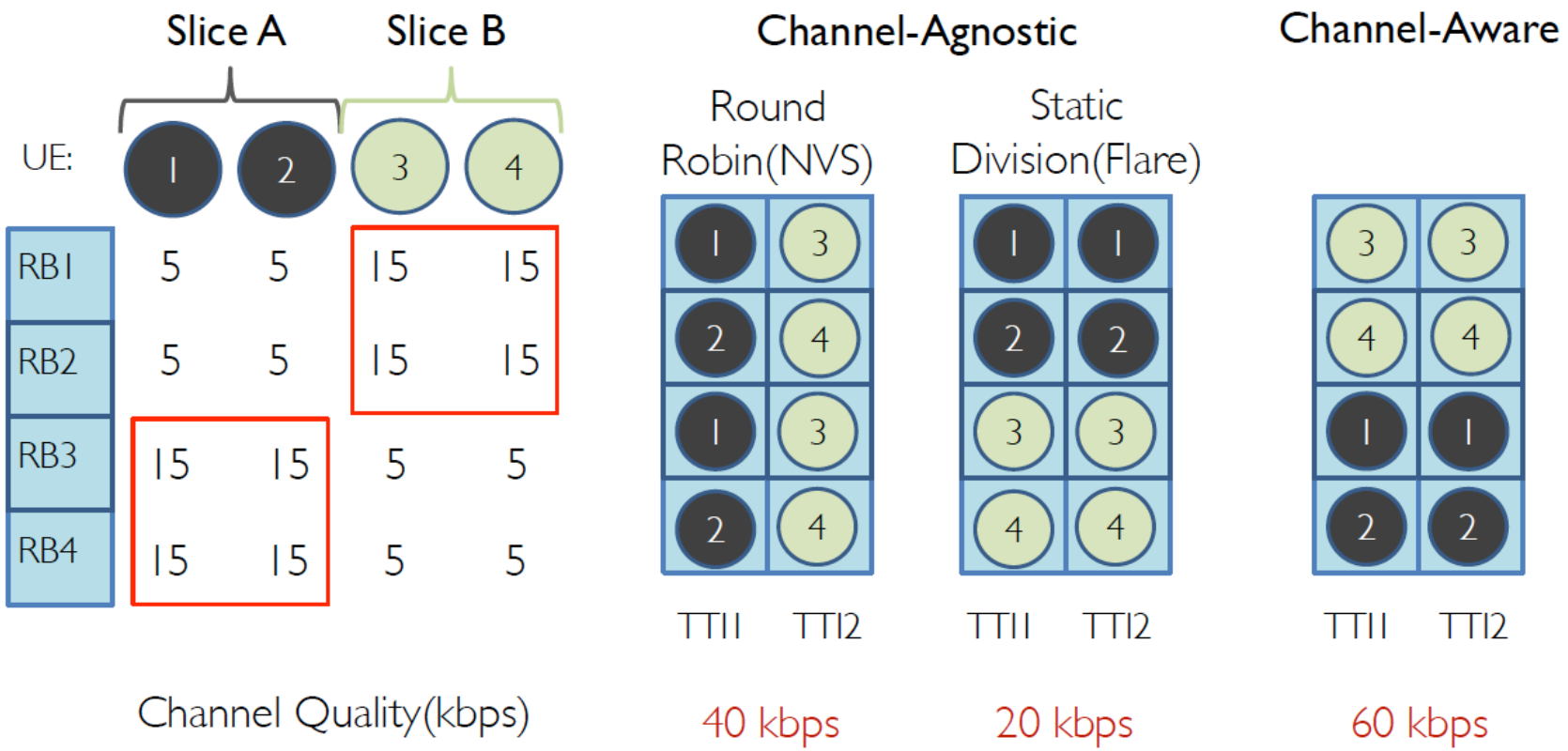
Why Do We Need Channel-Aware Inter-slice Scheduling



Why Do We Need Channel-Aware Inter-slice Scheduling

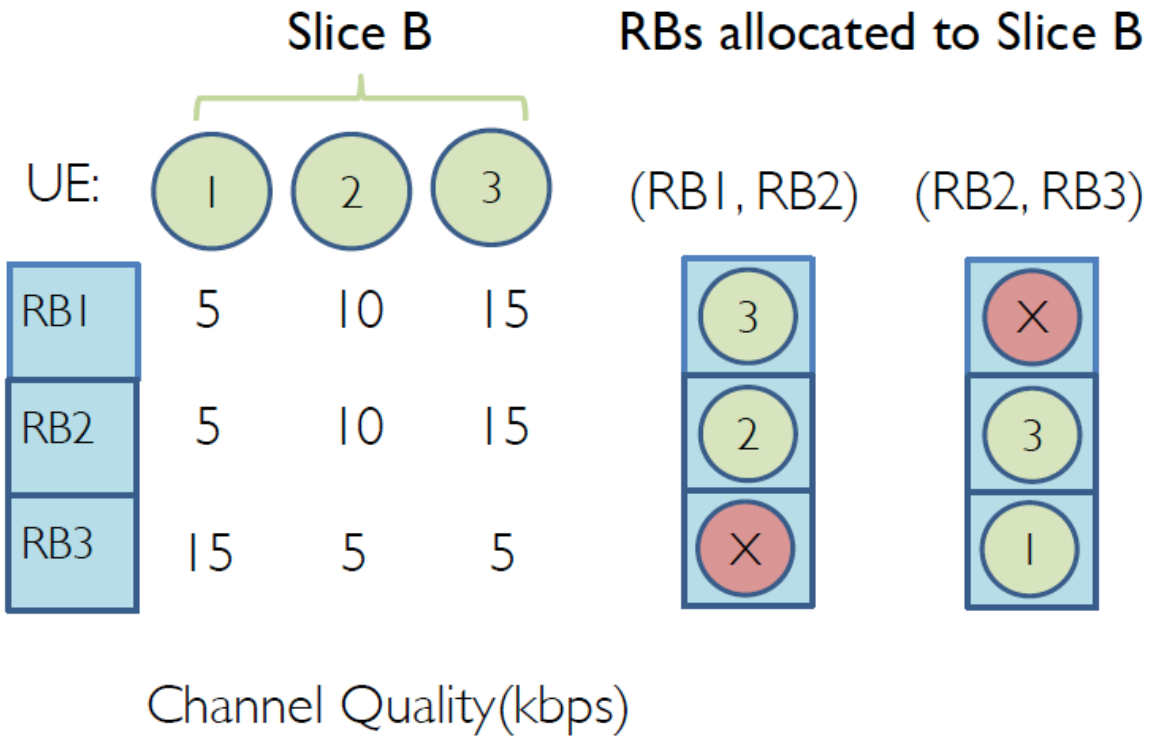


Why Do We Need Channel-Aware Inter-slice Scheduling

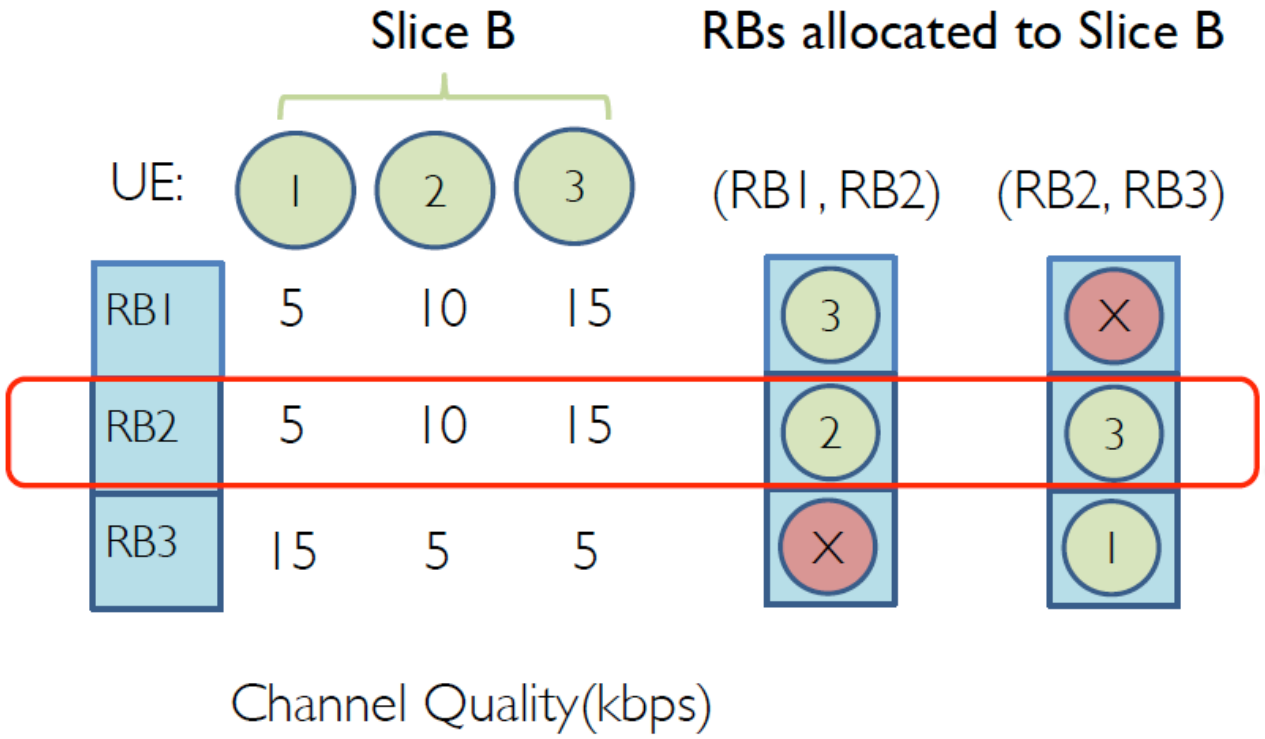


Channel-aware slicing at both inter-slice and enterprise level is challenging!

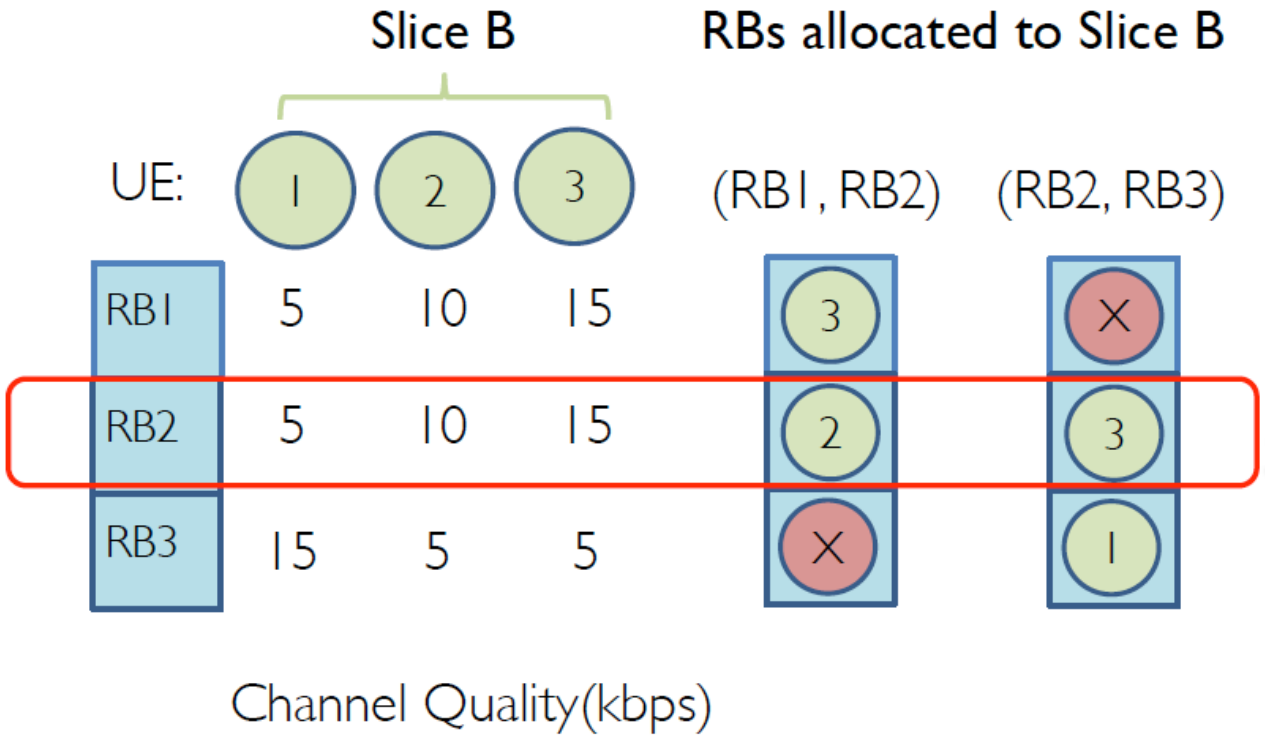
Challenge: Cyclic Dependency



Challenge: Cyclic Dependency

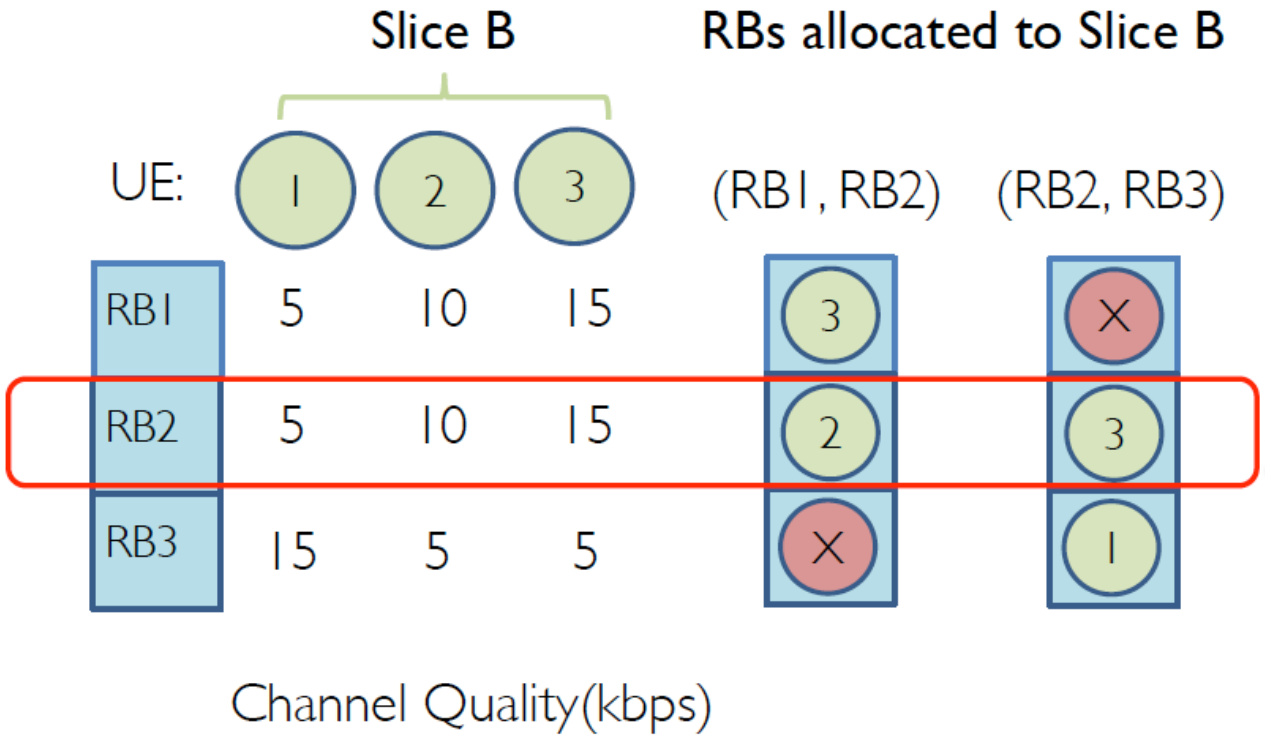


Challenge: Cyclic Dependency



- RB2 is allocated to different UEs.
- The channel quality is determined by the UE to which the RB is allocated.

Challenge: Cyclic Dependency



- RB2 is allocated to different UEs.
- The channel quality is determined by the UE to which the RB is allocated.

For an inter-slice scheduler to be channel-aware,
it must know the UE to which each RB will be allocated by the enterprise scheduler

Challenge: Cyclic Dependency

- For an inter-slice scheduler to be channel-aware,

  it must know the UE to which each RB will be allocated by the enterprise scheduler;

- For an enterprise scheduler to determine the resource allocation to its UEs,

  it must know the entire RBs allocated to this slice.

Insights to Solve the Challenge

- Both inter-slice scheduler and enterprise scheduler run in gNb.

- The inter-slice scheduler can query the enterprise scheduler:

  “If I give resource R to slice S, which UE in slice S will get resource R?”

- The enterprise scheduler is often greedy:

  It allocate RB to UE given the current RBs and historically allocated RBs,

  independent of future allocated RBs.

- The inter-slice scheduler must also be greedy.

RadioSaber’s Design

- Channel-aware inter-slice scheduler.
- Customizable enterprise scheduler.
- RadioSaber Workflow.

RadioSaber’s Design

- Channel-aware inter-slice scheduler.
- Customizable enterprise scheduler.
- RadioSaber Workflow.

Channel-Aware Inter-slice Scheduler

- Compute the quota for each slice per tti

Channel-Aware Inter-slice Scheduler

- Compute the quota for each slice per tti

For s in S do

rbgs_quota[s] = |RBG| × w_s

End for

weight for slice s

Non-integer number of quota!

Channel-Aware Inter-slice Scheduler

- Compute the quota for each slice per tti

For s in S do

rbs_share[s] = |RB|×w_s – rbs_offset[s]

rbgs_quota[s] = round_down(rbs_share[s] / k)

End for

number of RBs in a RBG

Channel-Aware Inter-slice Scheduler

- Compute the quota for each slice per tti

For s in S do

rbs_share[s] = |RB|×w_s – rbs_offset[s]

rbgs_quota[s] = round_down(rbs_share[s] / k)

End for

extra_rbgs = |RBG| - sum(rbgs_quota)

Channel-Aware Inter-slice Scheduler

- Compute the quota for each slice per tti

For s in S do

rbs_share[s] = |RB|×w_s – rbs_offset[s]

rbgs_quota[s] = round_down(rbs_share[s] / k)

End for

extra_rbgs = |RBG| - sum(rbgs_quota)

While extra_rbgs > 0 do

rbgs_quota[S.rand()] += 1

extra_rbgs -= 1

End while

Channel-Aware Inter-slice Scheduler

- Compute the quota for each slice per tti

For s in S do

rbs_share[s] = |RB|×w_s – rbs_offset[s]

rbgs_quota[s] = round_down(rbs_share[s] / k)

End for

extra_rbgs = |RBG| - sum(rbgs_quota)

While extra_rbgs > 0 do

rbgs_quota[S.rand()] += 1

extra_rbgs -= 1

End while

For s in S do

rbs_offset[s] = rbgs_quota[s] × k – rbs_share[s]

End for

Channel-Aware Inter-slice Scheduler

- Determine RB allocation to slices greedily:

Pick a RB

↓

Query every enterprise scheduler the maximum channel quality among its UEs for this RB

↓

Assign the RB to the slice which can offer the maximum channel quality

(if the quota for this slice hasn’t been satisfied)

Channel-Aware Inter-slice Scheduler

- Determine RB allocation to slices greedily:

The order to pick the RB matters!

Pick a RB

↓

Query every enterprise scheduler the maximum channel quality among its UEs for this RB

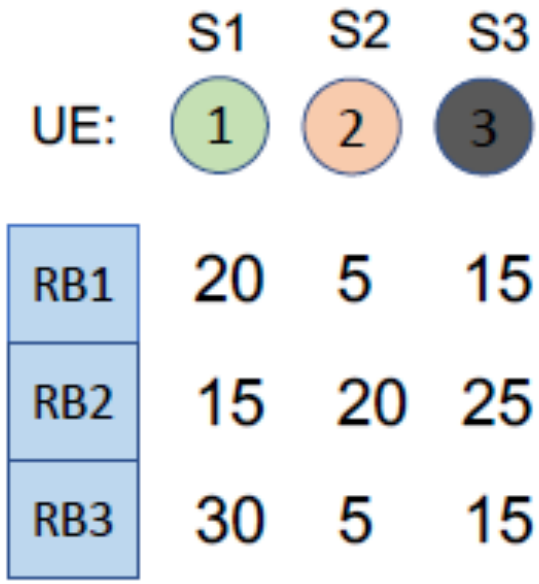
↓

Assign the RB to the slice which can offer the maximum channel quality

(if the quota for this slice hasn't been satisfied)

Channel-Aware Inter-slice Scheduler

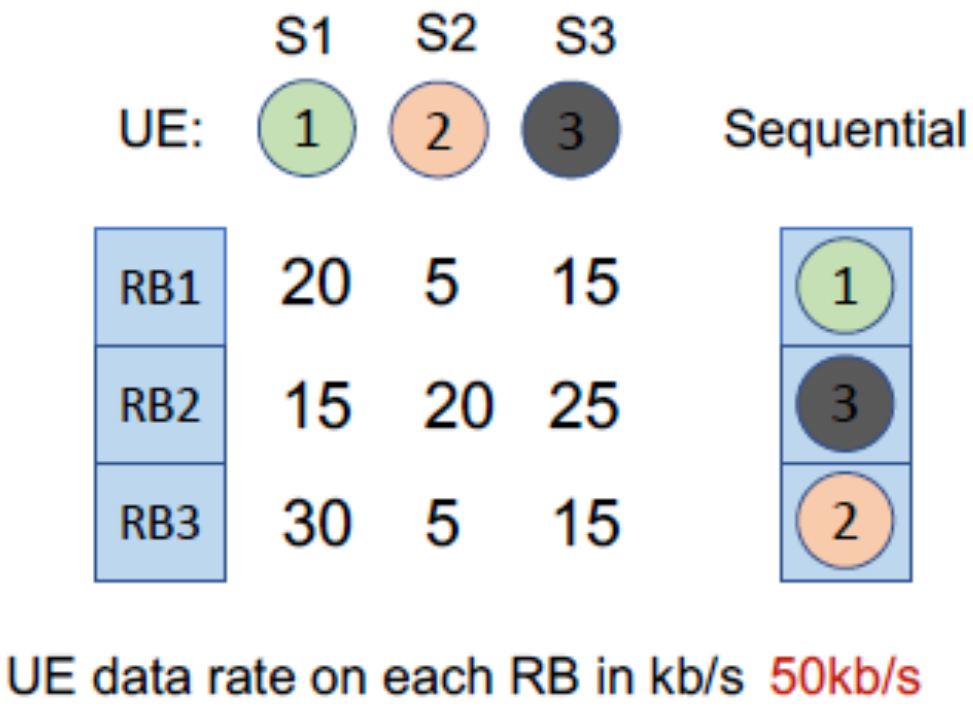
- The order to pick the RB matters.



UE data rate on each RB in kb/s

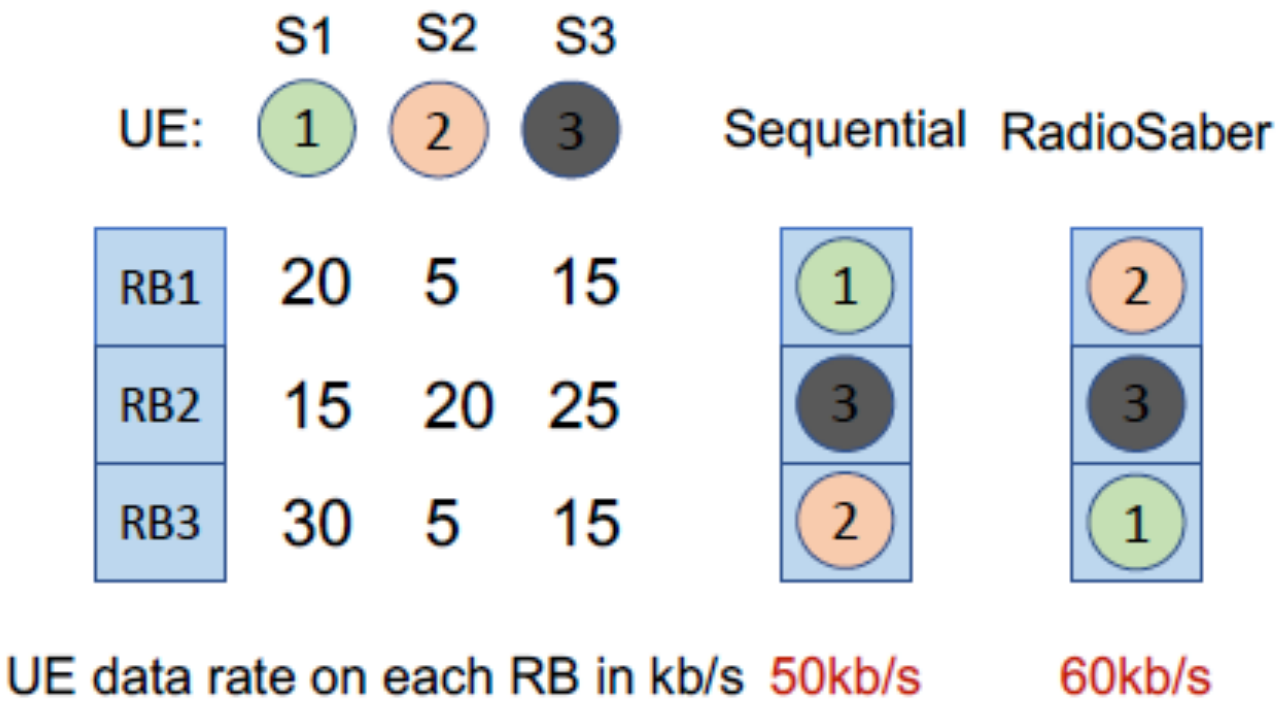
Channel-Aware Inter-slice Scheduler

- The order to pick the RB matters.



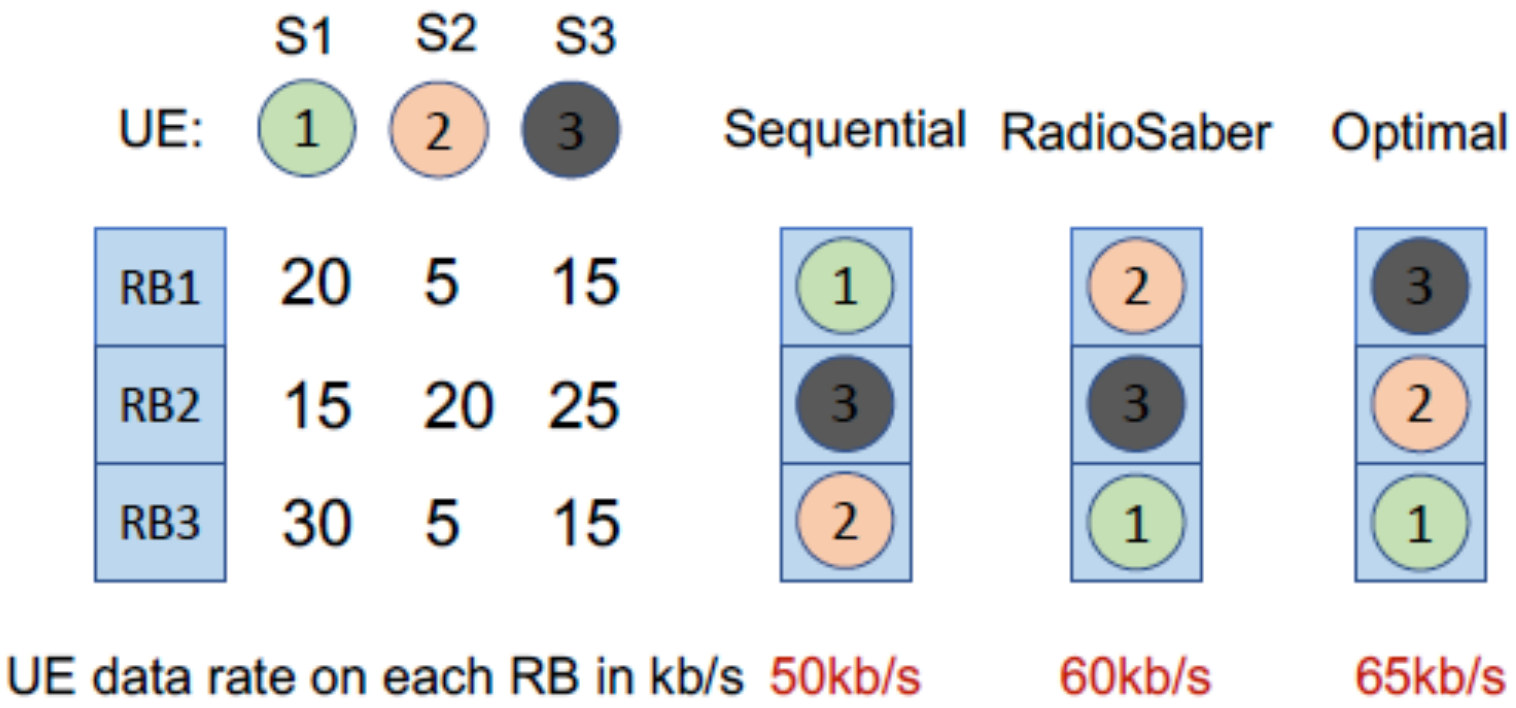
Channel-Aware Inter-slice Scheduler

- The order to pick the RB matters.



Channel-Aware Inter-slice Scheduler

- The order to pick the RB matters.



RadioSaber’s Design

- Channel-aware inter-slice scheduler.

- Customizable enterprise scheduler.

- RadioSaber Workflow.

Customizable Enterprise Scheduler

- Factors to consider:

  channel quality, fairness, flow priority, queuing delay

- Solution: parameterization.

Customizable Enterprise Scheduler

· Paradigm 1: select user first

- for UE u given RBG i:

$$metric(u,i) = d_{u,i}^{\varepsilon} / R_u^{\Psi}$$

pick the UE with highest metric and the flow with the highest priority for that UE.

- $d_{u,i}$: instantaneous data rate for UE u at RBG i

R_u : historical RBG allocation to UE u

ε, Ψ: parameters, determine the relative weightage

Customizable Enterprise Scheduler

- Paradigm 2: select highest priority first
  - for UE u given RBG i and flow priority p:

$$metric(u,p,i) = (\beta D_{u,p} + (1-\beta))(d_{u,i}^{\varepsilon}/R_u^{\Psi})$$

  - $d_{u,i}$: instantaneous data rate for UE u at RBG i

    R_u : historical RBG allocation to UE u

    $D_{u,p}$: queuing delay of UE u and priority p

    β, ε, Ψ: parameters, determine the relative weightage

RadioSaber's Design

- Channel-aware inter-slice scheduler.
- Customizable enterprise scheduler.
- RadioSaber Workflow.

RadioSaber's Workflow

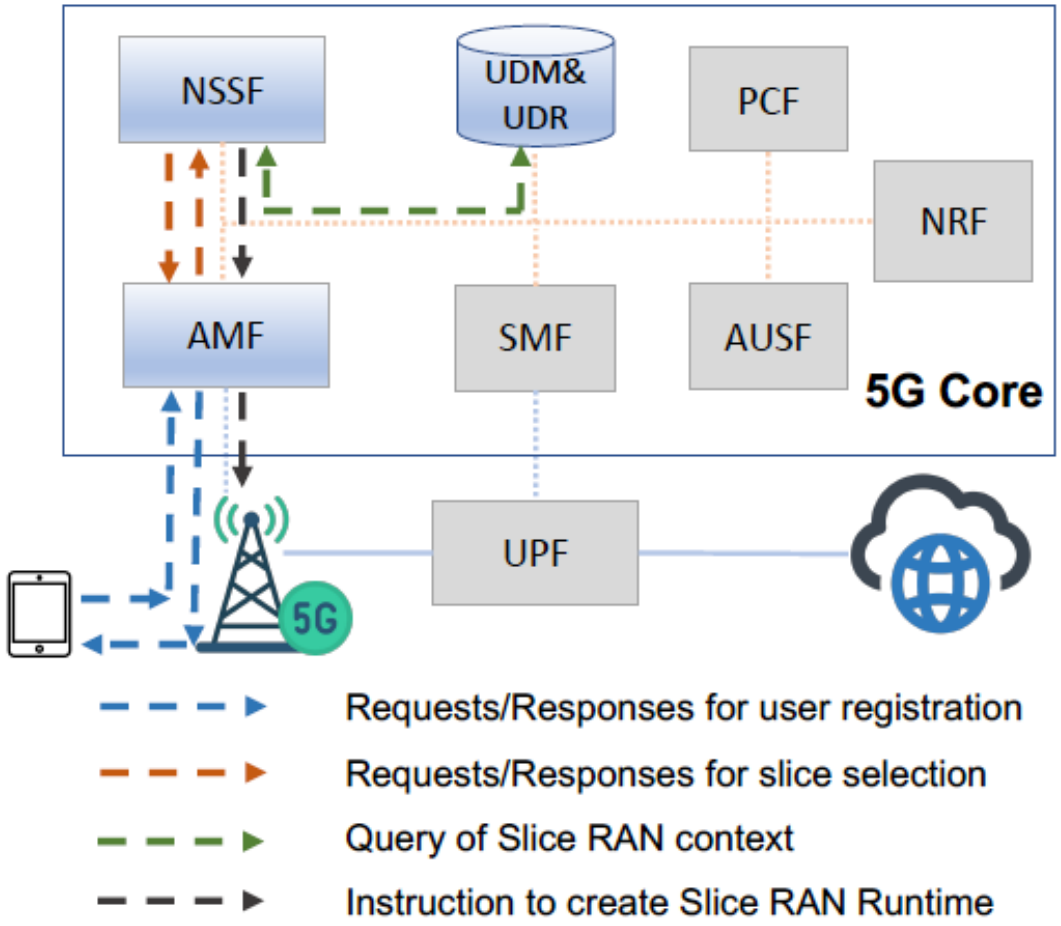


Figure 5: 5G Core network architecture, and the workflow for relaying slice context from 5G core to gNB.

RadioSaber's Workflow

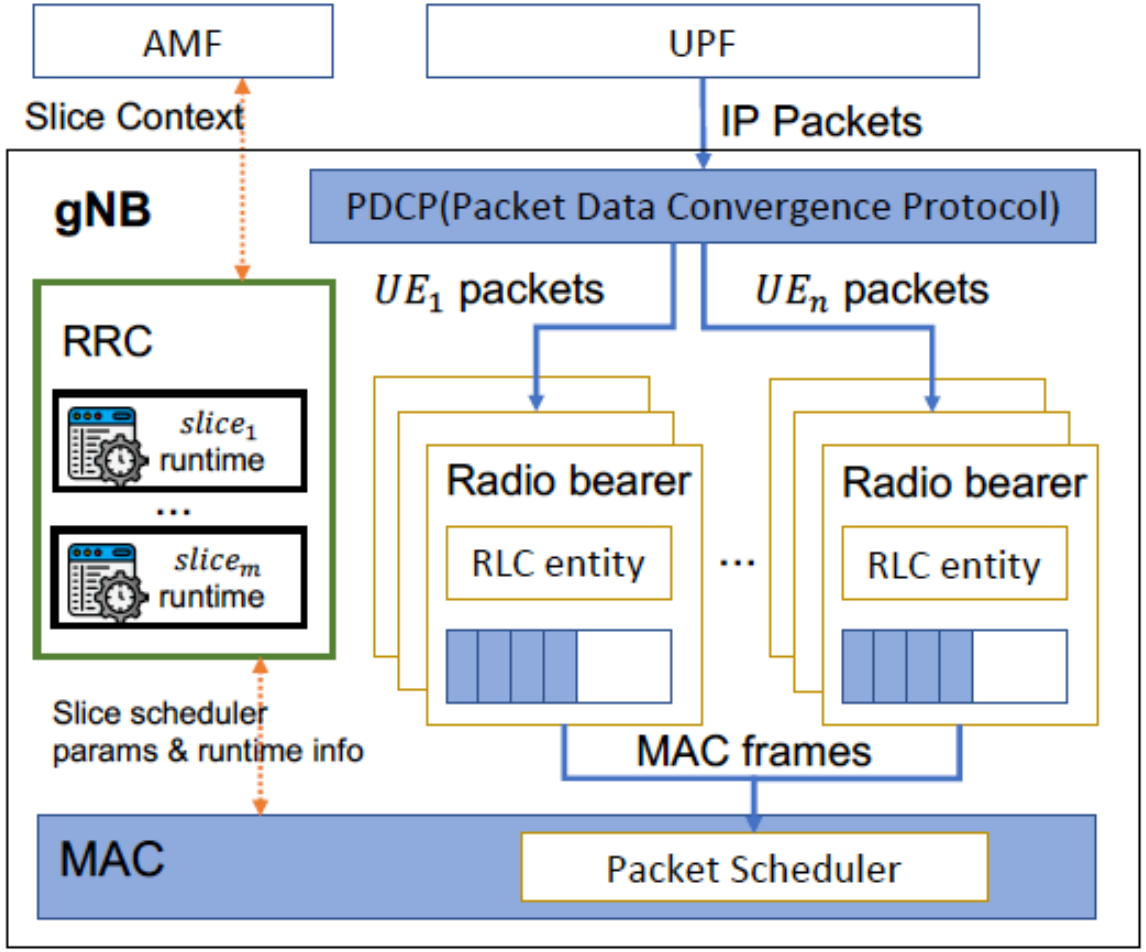


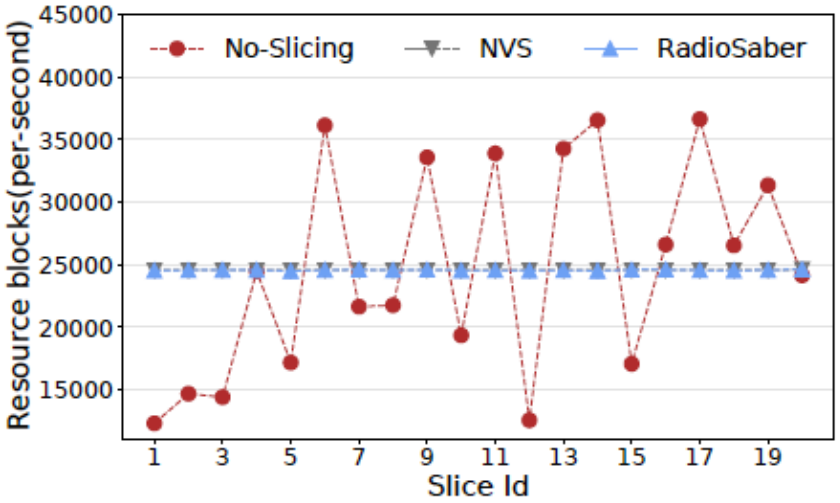
Figure 6: 5G gNB architecture: it shows how RRC maintains slice runtimes, and controls the MAC scheduler behavior.

Implementation

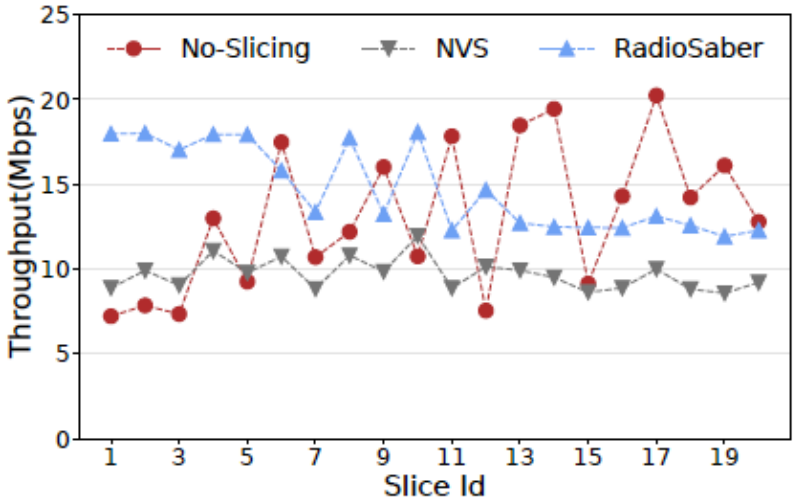
- Extend Open5GS to add support for RadioSaber control workflow
  - 530 lines code in total
- Trace-driven simulation using traces from LTScope
  - up to 800 users

Evaluation

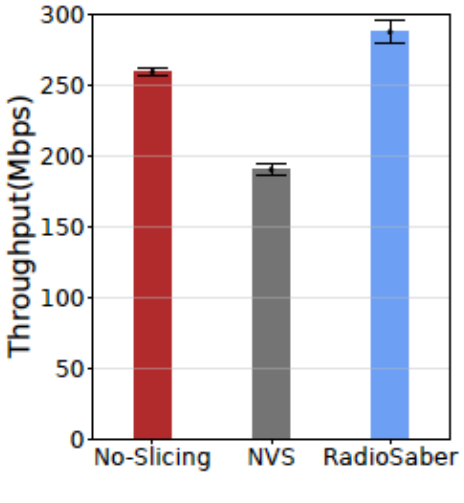
Spectrum efficiency and fairness



(a) per-slice RBs per second



(b) per-slice throughput



(c) sum total throughput

slice 1-10, maximum throughput scheduling

slice 11-20, proportional fair scheduling

Evaluation

Diverse enterprise schedulers

Slices	Scheduler $(\alpha, \beta, \varepsilon, \psi)$	Traffic generation	Metrics
1-5	PF(0,0,1,1)	a backlogged flow	average throughput
6-10	PF(1,0,1,1)	heavy-tail distributed flows	FCT(Flow Completion Time)
11-15	PF(1,0,1,1)	heavy-tail distributed flows(25% prioritized)	FCT of prioritized flows
16-20	M-LWDF (1,1,1,1)	a 1280kbps real-time video flow	average queueing delay

Table 2: Scheduling configuration, workloads per user, and metrics to evaluate in different slices.

Evaluation

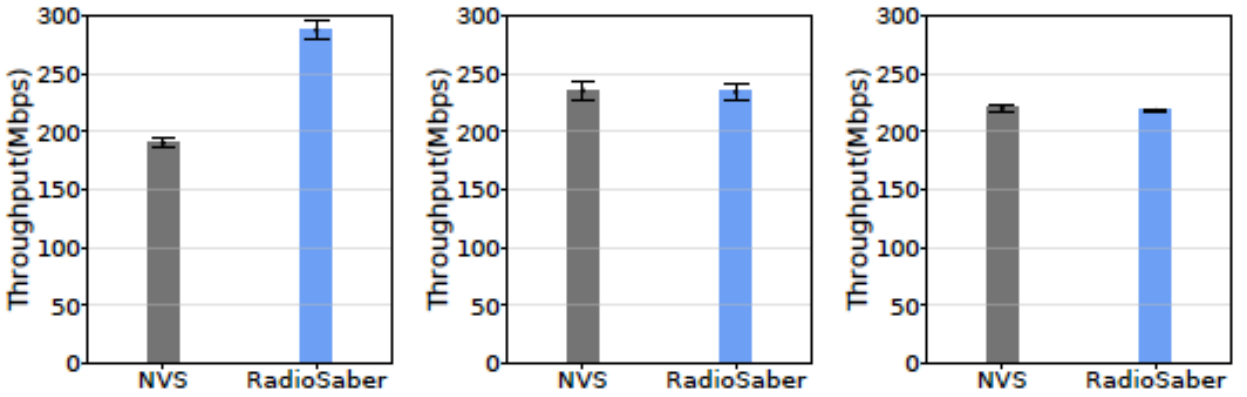
Diverse enterprise schedulers

Slices	Metrics	RadioSaber	NVS	No-Slicing
1-5	throughput (Mbps)	13.02	8.45	**17.41**
6-10	average FCT(s)	**2.606**	5.708	5.714
11-15	average FCT(s)	**0.489**	1.686	2.988
16-20	average queueing delay(s)	**0.061**	1.493	0.696

Table 3: Experiment results w.r.t different metrics in all slices of RadioSaber and baselines.

Evaluation

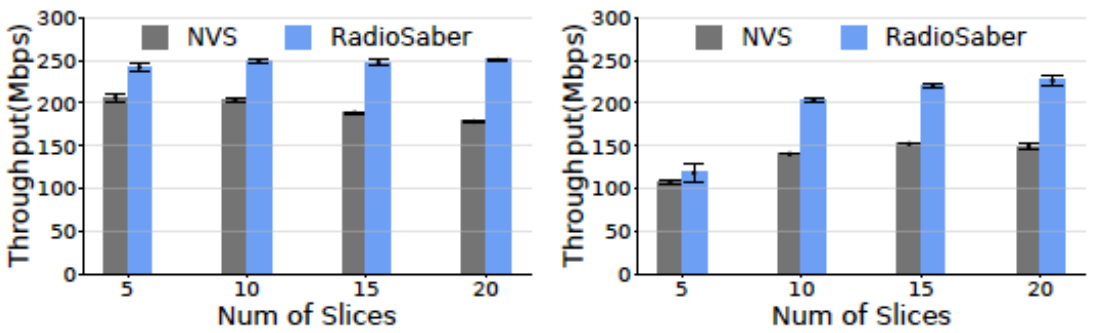
Ablation experiment



(a) Real Trace (b) Synthetic: no variation in sub-band CQI (c) Synthetic: non-complementary subband CQI

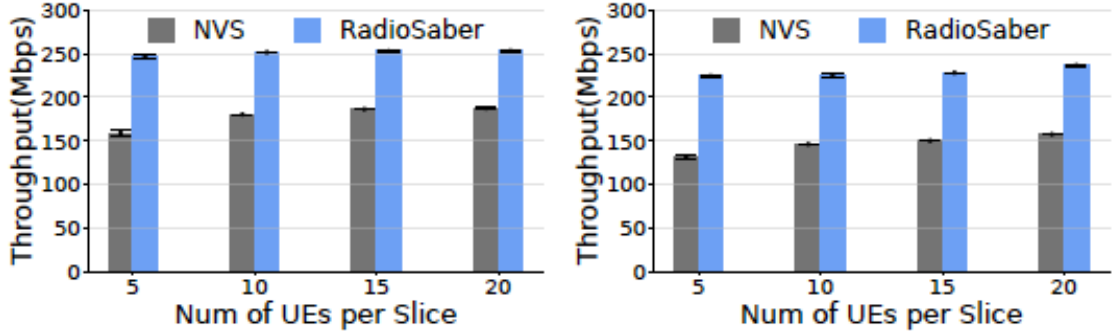
Evaluation

Varying number of slices and number of UEs per slice



(a) backlogged flows (b) heavy-tail distributed flows

Figure 9: Varying the number of slices (5-15 UEs per slice).



(a) backlogged flows (b) heavy-tail distributed flows

Figure 10: Varying the number of UEs per slice for 20 slices.

Evaluation

Non-greedy enterprise schedulers

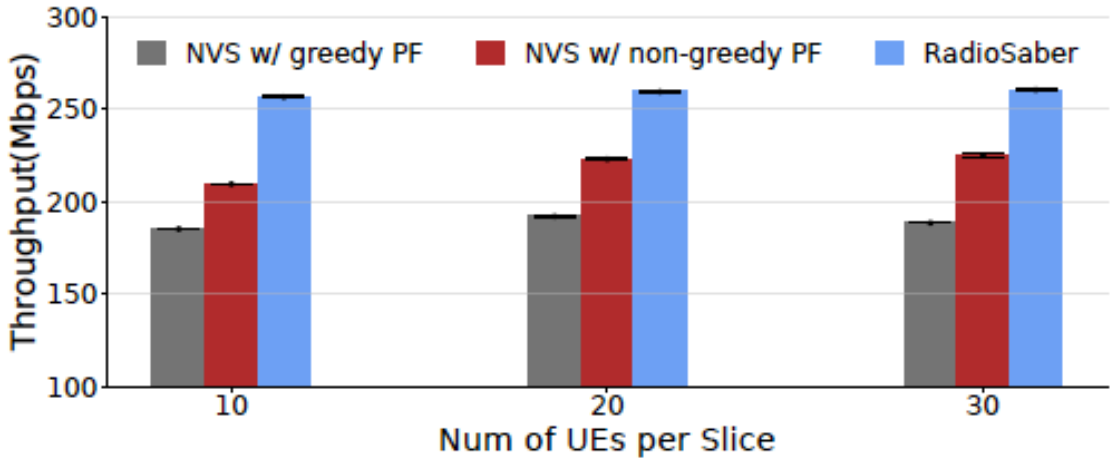


Figure 11: Comparing RadioSaber (using a greedy PF scheduler) with NVS (using a non-greedy PF scheduler). We fix number of slices to 20, and vary the number of UEs per slice.

Evaluation

Other inter-slice schedulers

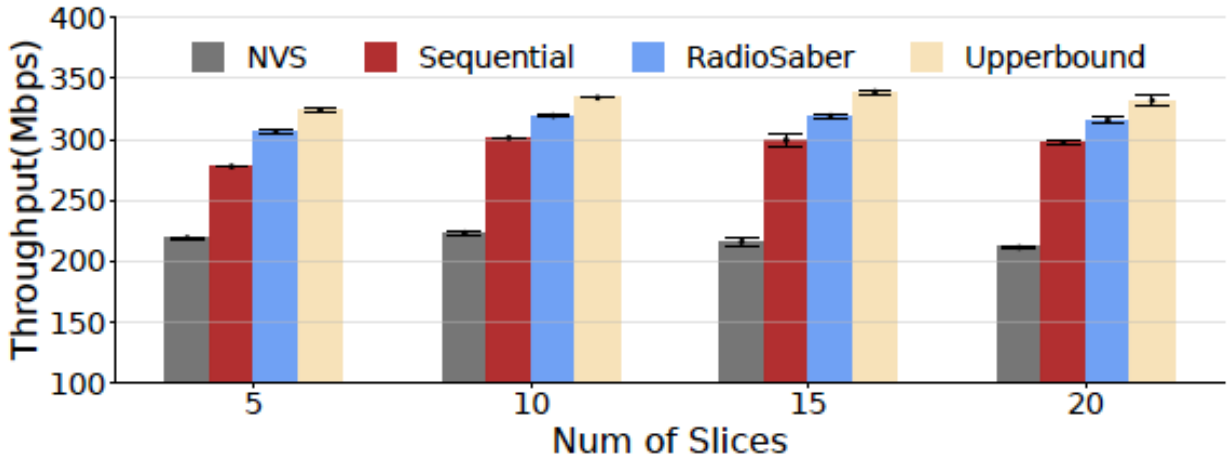


Figure 12: Comparing RadioSaber with a simple greedy inter-slice scheduler, and with a contrived upperbound. We vary the number of slices, with 5-15 random users in each slice, and use MT scheduling policy in each slice.

Evaluation

Scheduling latency

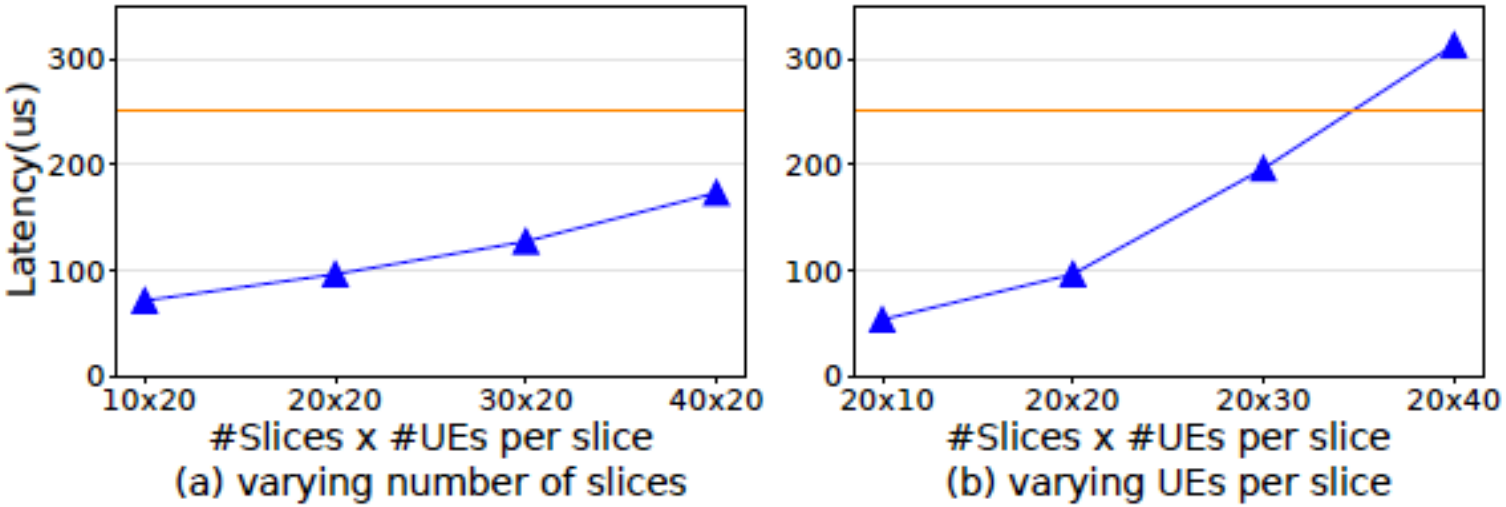


Figure 13: RadioSaber's scheduling latency

Opinion

- Low implementation
- Good background statement
- Complementary channel quality may not be very common