FIFO WITH OFFSETS

HIGH SCHEDULABILITY WITH LOW OVERHEADS

RTAS'18
April 18, 2018

Mitra Nasri
MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

Rob Davis
THE UNIVERSITY of York
Inria INVENTEURS DU MONDE NUMÉRIQUE

Björn Brandenburg
MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

MAX-PLANCK-GESELLSCHAFT

FIFO SCHEDULING

First-In-First-Out (**FIFO**) scheduling

- extremely **simple**
- very **low overheads**

ideal for:
- *IoT-class devices*
- *deeply embedded systems*
- *hardware implementations*

*very **low schedulability*** } meeting **deadlines**?

FIFO SCHEDULING

First-In-First-Out (**FIFO**) scheduling

- extremely **simple**
- very **low overheads**

ideal for:
- *IoT-class devices*
- *deeply embedded systems*
- *hardware implementations*

very ~~low~~ HIGH! *schedulability* } meeting **deadlines**?

THIS PAPER

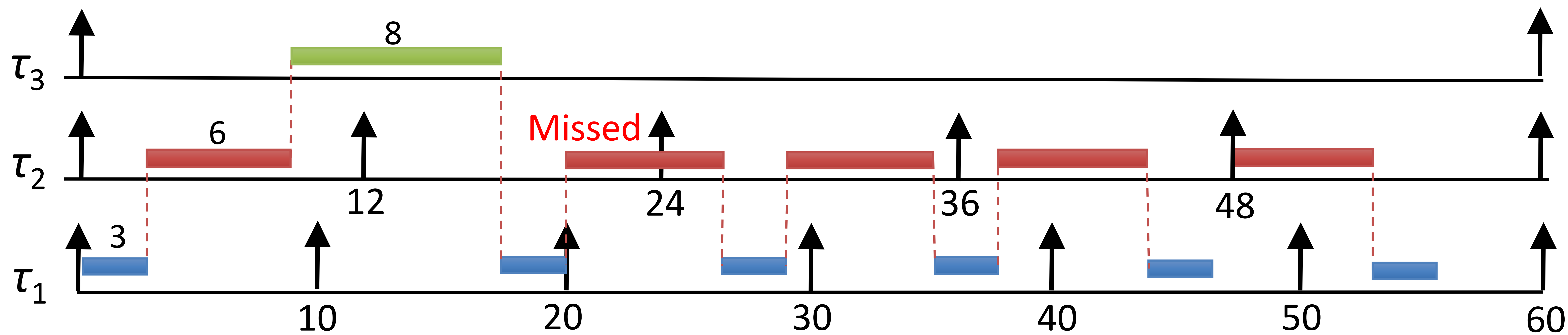
FIFO can actually achieve excellent schedulability!

[**periodic non-preemptive tasks** on a **uniprocessor**]

INTUITION

THE PROBLEM WITH PLAIN FIFO SCHEDULING

FIFO schedule of 3 periodic tasks:



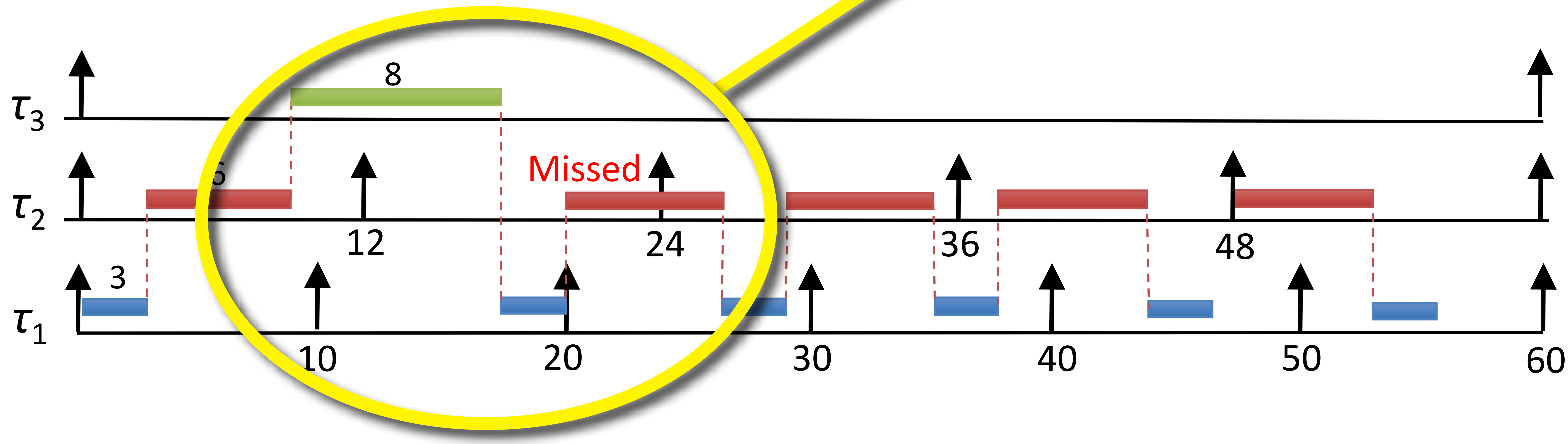
Task	WCET	Period
τ_3	8	60
τ_2	6	12
τ_1	3	10

THE PROBLEM

Plain FIFO is oblivious to deadlines and priorities

τ_3 comes first → deadline miss

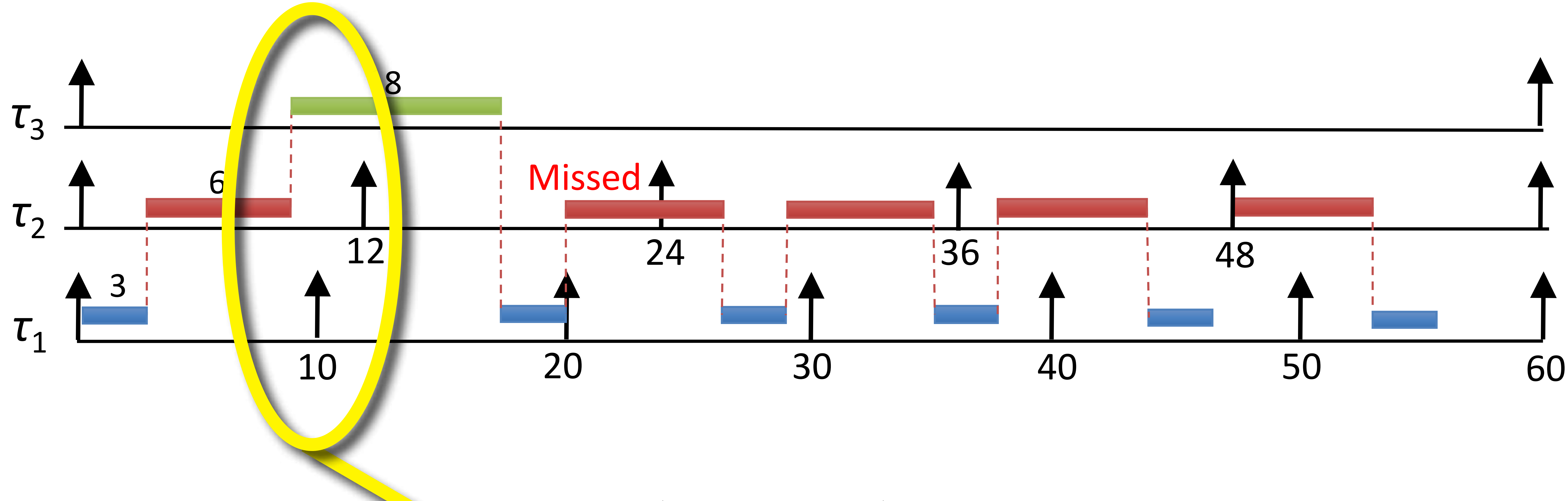
FIFO schedule of 3 periodic tasks:



Task	WCET	Period
τ_3	8	60
τ_2	6	12
τ_1	3	10

THE PROBLEM WITH PLAIN FIFO SCHEDULING

FIFO schedule of 3 periodic tasks:



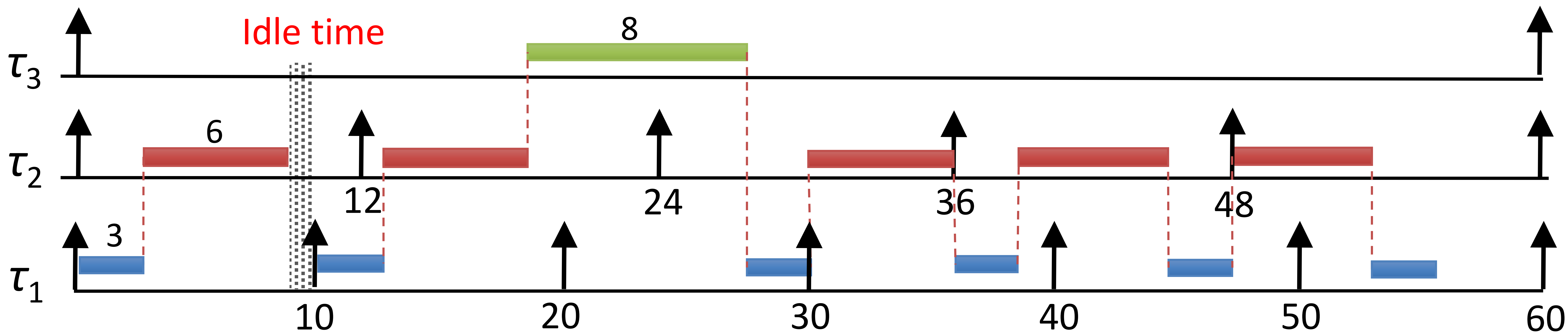
In fact, *any work-conserving policy* (EDF, RM, …)

must schedule τ_3 here → deadline miss.

NON-WORK-CONSERVING SCHEDULING

[*critical-window EDF: Nasri & Fohler, 2016*]

CW-EDF schedule of the same 3 periodic tasks:



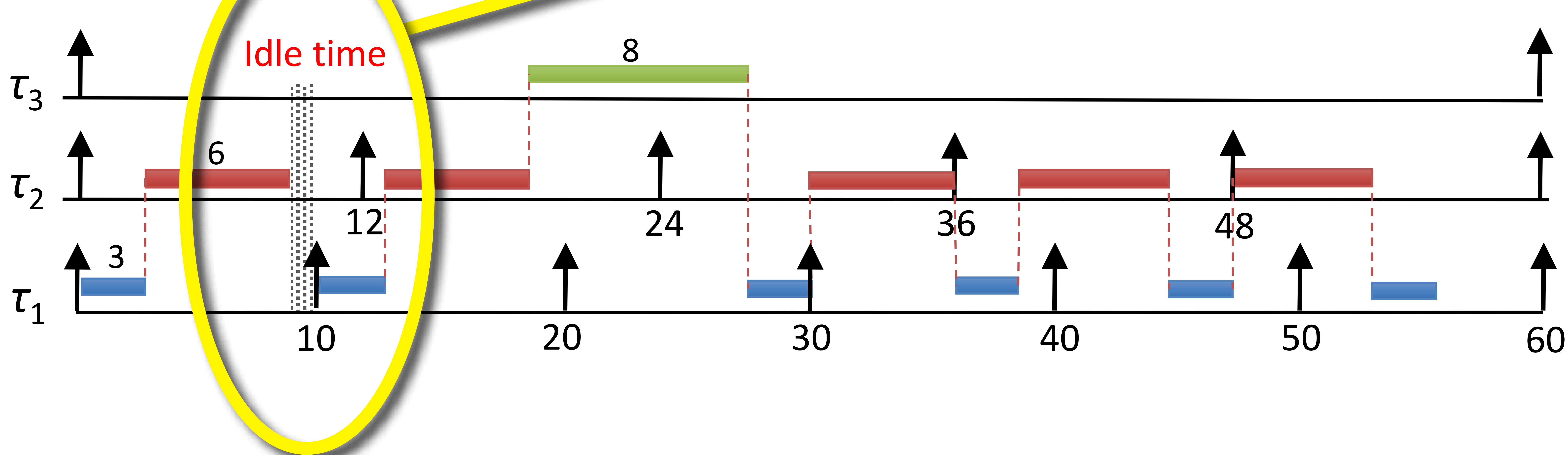
Task	WCET	Period
τ_3	8	60
τ_2	6	12
τ_1	3	10

NON-WORI

[critical-window EDF: Nasri &

CW-EDF considers *future job arrivals* in the "critical window" and postpones τ_3 until later.

CW-EDF schedule of the same 3 periodic tasks.

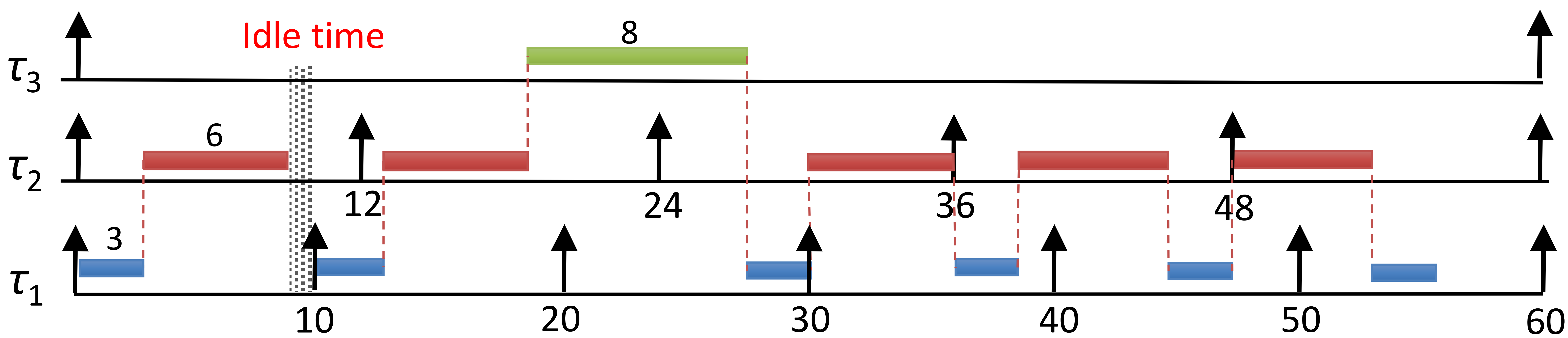


Task	WCET	Period
τ_3	8	60
τ_2	6	12
τ_1	3	10

NON-WORK-CONSERVING SCHEDULING

[*critical-window EDF: Nasri & Fohler, 2016*]

CW-EDF schedule of the same 3 periodic tasks:



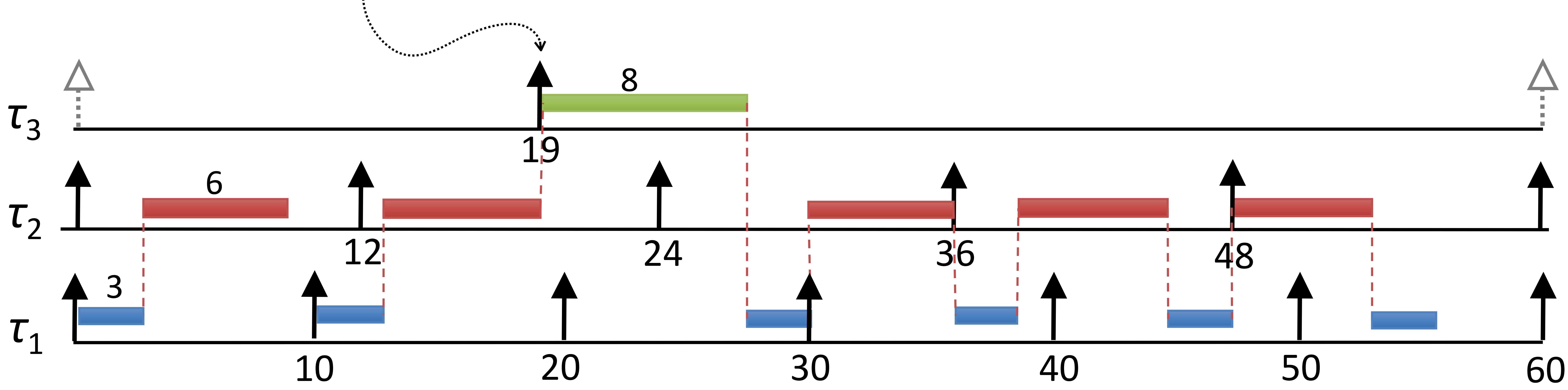
LIMITATION

CW-EDF incurs ***much higher runtime overheads*** than simple work-conserving policies.

ATMega2560 @ 16 MHz: ***9.2× higher than RM!***

INTUITION: FIFO + "JUST THE RIGHT" OFFSETS

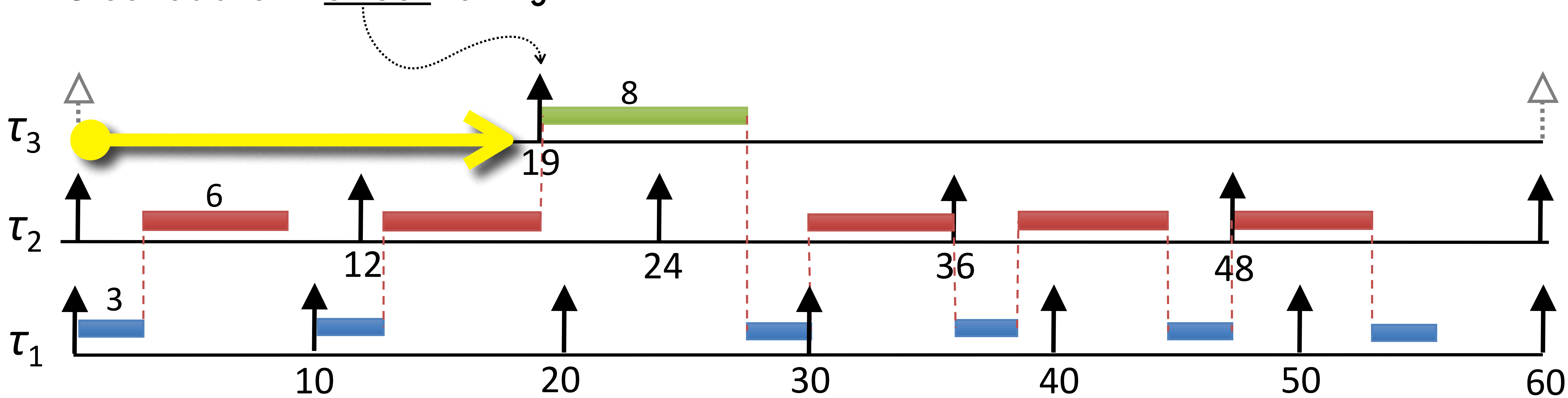
FIFO schedule + *offset for* τ_3 :



Task	WCET	Period
τ_3	8	60
τ_2	6	12
τ_1	3	10

INTUITION: FIFO + "JUST THE RIGHT" OFFSETS

FIFO schedule + *offset for τ_3* :



Move τ_3 "out of the way" by *introducing* (or *adjusting*) a *release offset*.

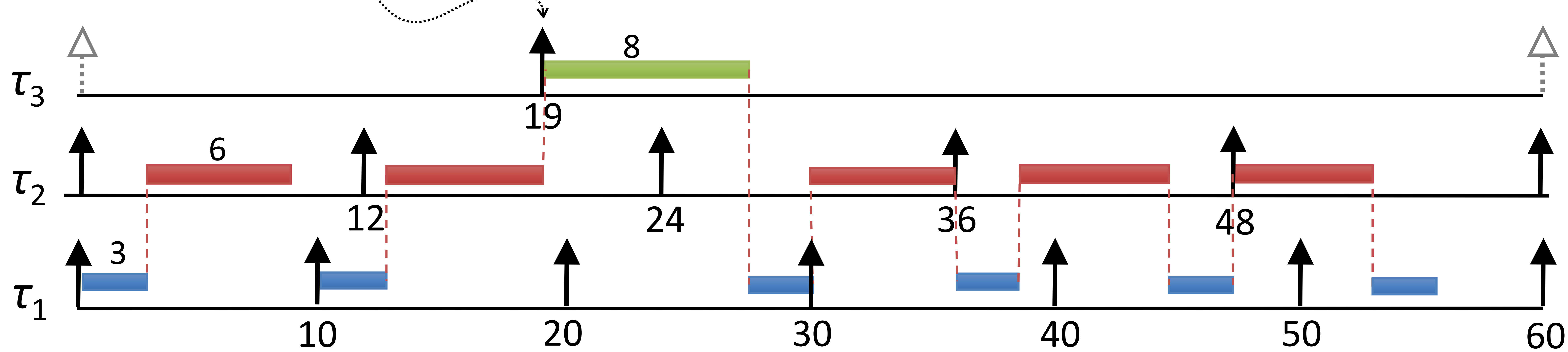
FIFO schedule becomes identical to CW-EDF schedule!

INTUITION: FIFO + “JUST THE RIGHT” OFFSETS

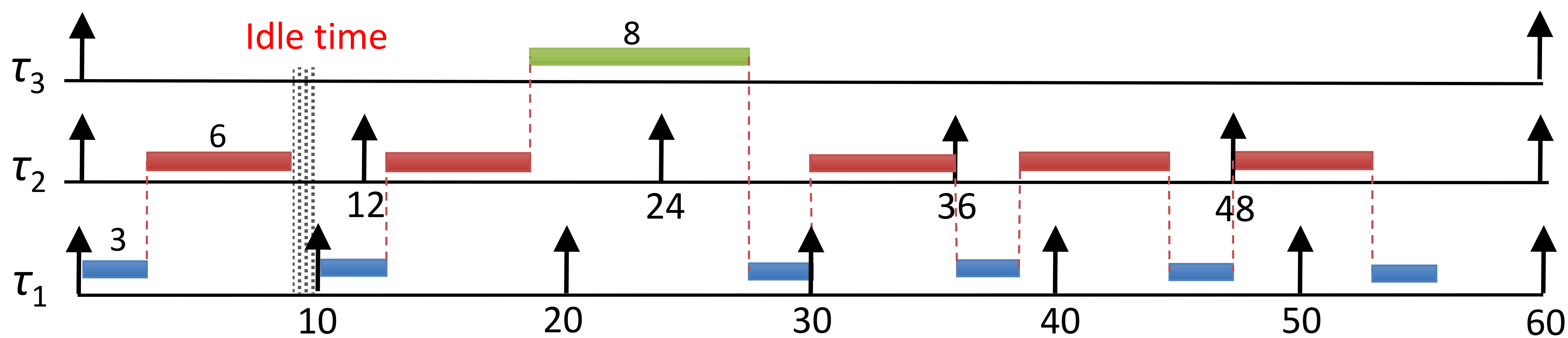
???

[Altmeyer, Sundharam, & Navet, 2016]

FIFO schedule + *offset for* τ_3 :



CW-EDF schedule is identical:



<u>THIS PAPER</u>

OFFSET TUNING ALGORITHM

PROBLEM STATEMENT

Given a set of ***n*** *periodic non-preemptive tasks*,
find, for each job of each task, a ***release offset*** such that

(A) the resulting ***FIFO schedule is feasible***, and

(B) the ***number of offsets*** per task is ***minimized***.

Challenges

- ➡ space of possible offsets is large and unstructured
- ➡ even ignoring (B), solving "just" (A) is *very* difficult

Altmeyer et al.

- ➡ randomize offsets + test
- ➡ not systematic
- ➡ scalability limitations

[*S. Altmeyer, S. Sundharam, and N. Navet, "The case for FIFO real-time scheduling," University of Luxembourg, Tech. Rep., 2016*]

KEY INSIGHT

Given a set of ***n*** *periodic non-preemptive tasks*,
find, for each job of each task, a ***release offset*** such that

(A) the resulting ***FIFO schedule is feasible***, and

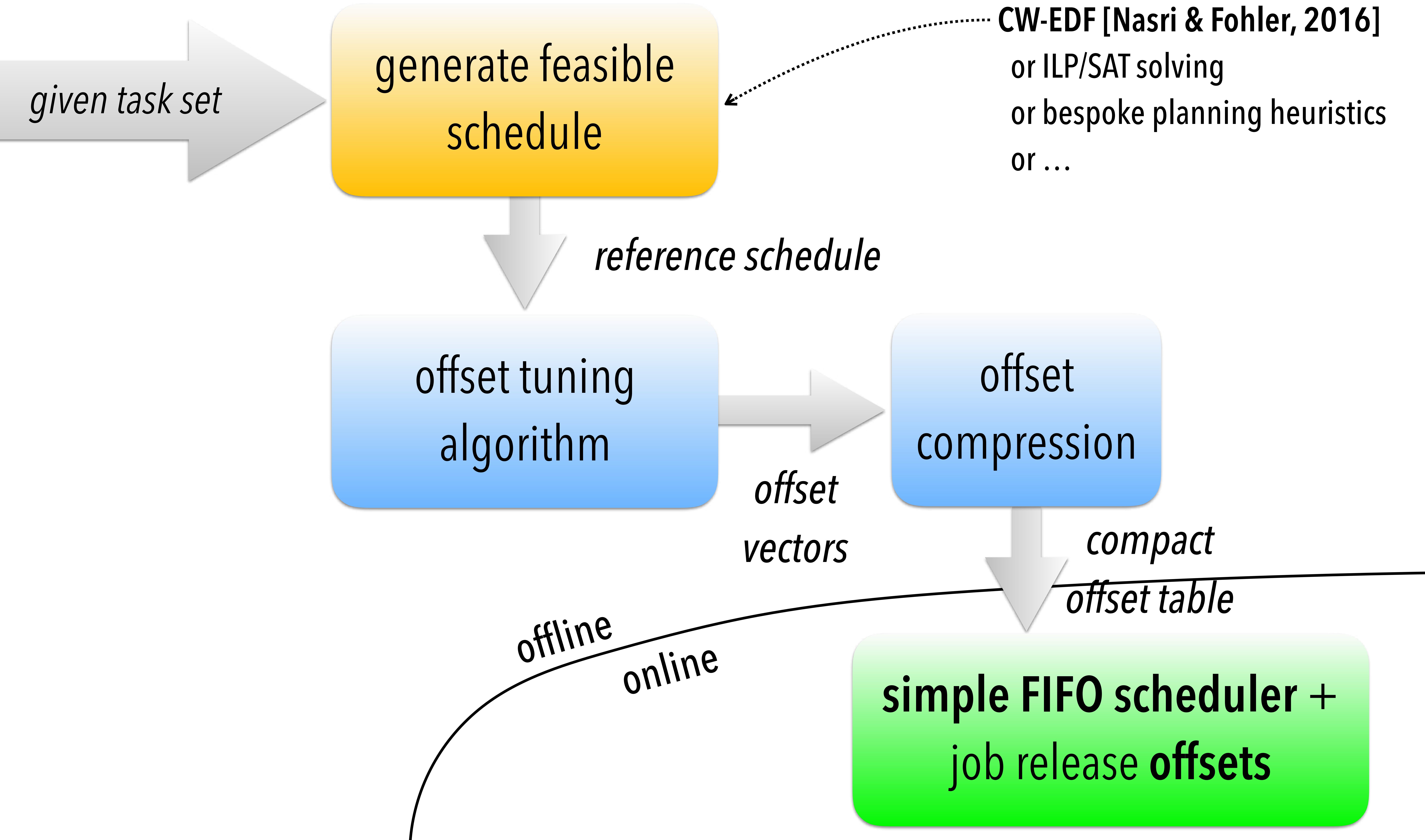
(B) the ***number of offsets*** per task is ***minimized***.

Solving (A) is very difficult… so we don't!

OFFSET TUNING

Infer offsets *from a given* ***feasible reference schedule****, while greedily working towards (B).*

OFFSET TUNING – OVERVIEW



SCHEDULE EQUIVALENCY

A schedule S_1 is equivalent to S_2 if

(i) they schedule the ***same jobs***,

(ii) in the ***same order***, and

(iii) ***jobs start no later*** in S_1 than in S_2.

Non-preemptive execution

➡ jobs also complete no later in S_1 than in S_2

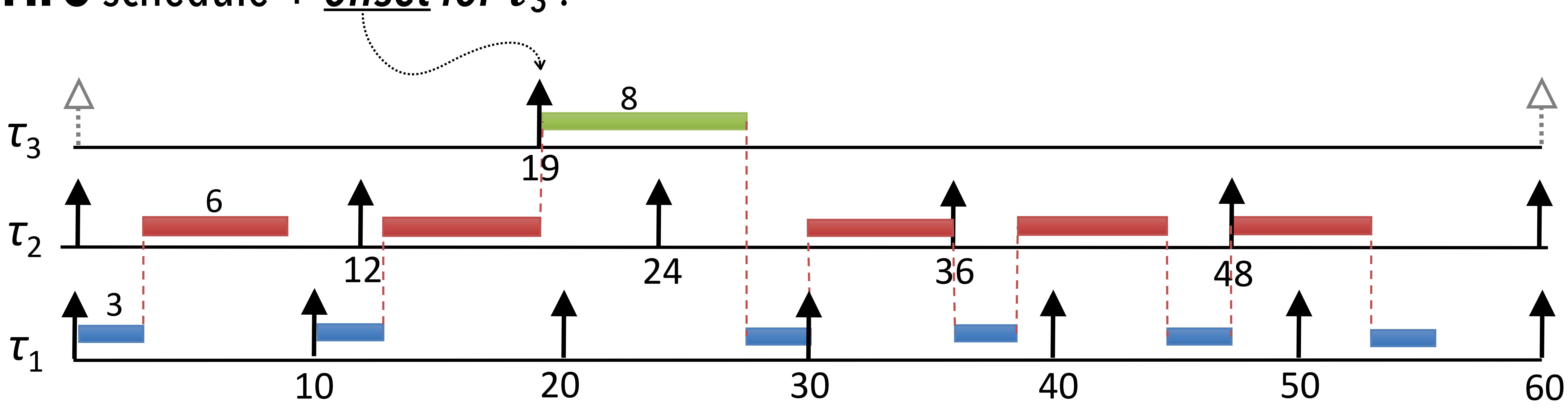
Offset Tuning

➡ ensures FIFO schedule is equivalent to reference schedule

POI: POTENTIAL OFFSETS INTERVAL

POI of a job: range of release offsets that ***guarantee schedule equivalency***.

FIFO schedule + *offset for τ_3* :

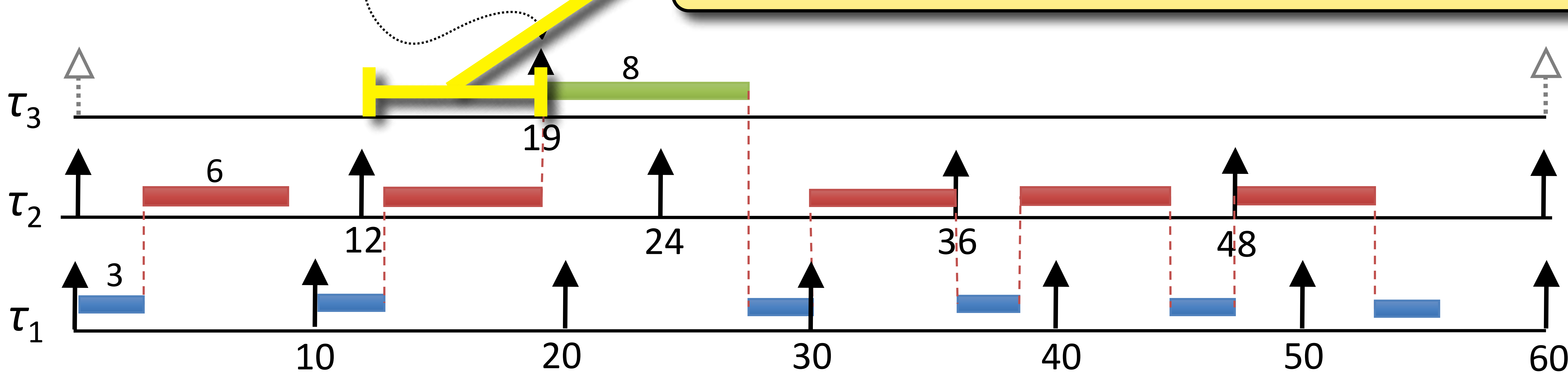


POI: POTENTIAL OFFSETS INTERVAL

POI of a job: range of release offsets that ***guarantee schedule equivalency***.

POI: any release time of τ_3 in (12, 19] will yield an equivalent schedule.

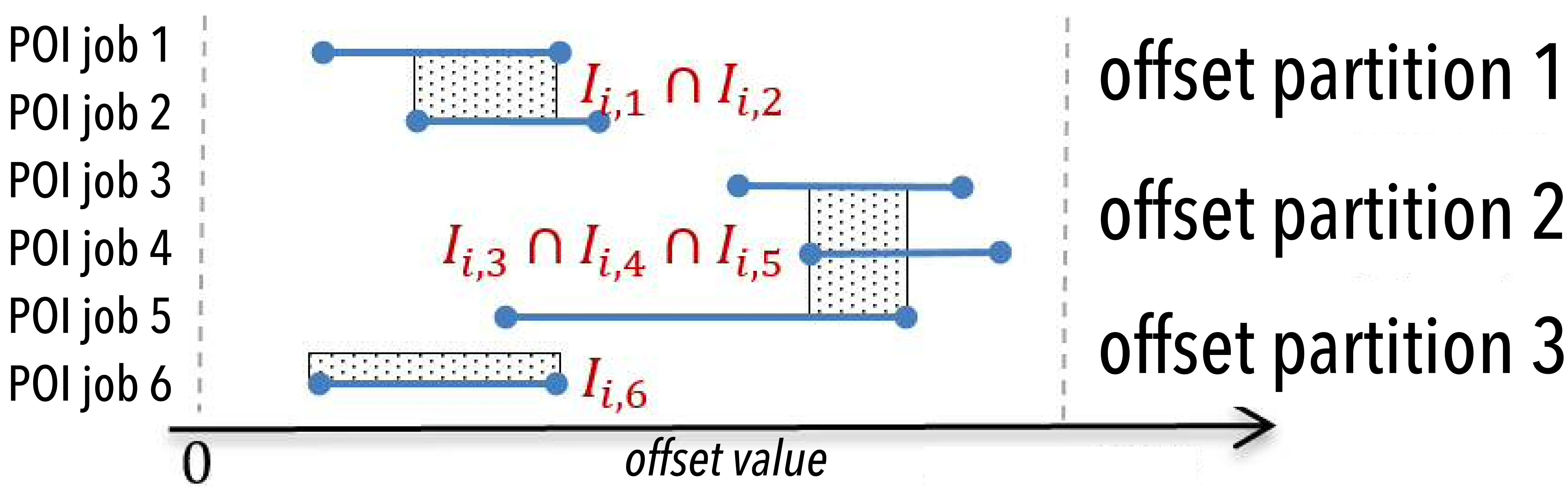
FIFO schedule + *offset for τ_3* :



OFFSET PARTITION

Consecutive jobs of a task form an **offset partition** if they have ***mutually intersecting POIs***.

➔ *can be assigned a single offset*



➔ *offset partitioning not necessarily unique*

OFFSET TUNING ALGORITHM (*SIMPLIFIED*)

for each task τ_i in deadline-monotonic order:

greedily create ***offset partitions*** for τ_i

assuming ***jobs of larger-deadline tasks***

are released as in reference schedule

Need to start somewhere…

shorter relative deadline = fewer options

…THM (*SIMPLIFIED*)

for each task τ_i in deadline-monotonic order:

greedily create ***offset partitions*** for τ_i

assuming ***jobs of larger-deadline tasks*** *are released as in reference schedule*

Release times of not-yet-processed jobs still unknown → speculate.

Mis-speculation increases the number of offset partitions,

*but **does not** cause the algorithm to fail.*

PROPERTIES OF OFFSET TUNING

REFERENCE SCHEDULE EQUIVALENCY

In the resulting FIFO schedule, ***no job completes later*** *than in the original reference schedule.*

PER-TASK MINIMAL OFFSET PARTITIONS

The greedy offset partitioning strategy yields a minimal number of offset partitions (for a given task).

NON-MINIMAL OFFSET PARTITIONS FOR ENTIRE TASK SET

Deadline-monotonic processing order does not guarantee overall minimal number of offset partitions (but ***works well empirically****).*

SINGLE-OFFSET HEURISTICS

What if we want just a *single offset* per task?

➡ no extra memory required

➡ compatibility with existing systems

FST: First-Start-Time Heuristic

➡ pick start time of first job in reference schedule

FOP: First-Offset-Partition Heuristic

➡ pick offset from first offset partition of the task

EVALUATION

EVALUATION QUESTIONS

Q1: Does FIFO + Offset Tuning still have low runtime overheads?

Q2: Does FIFO + Offset Tuning (FIFO-OT) significantly improve schedulability relative to EDF/RM?

Q3: How many offsets are assigned?

Q4: How much memory is needed?

PROTOTYPE PLATFORM

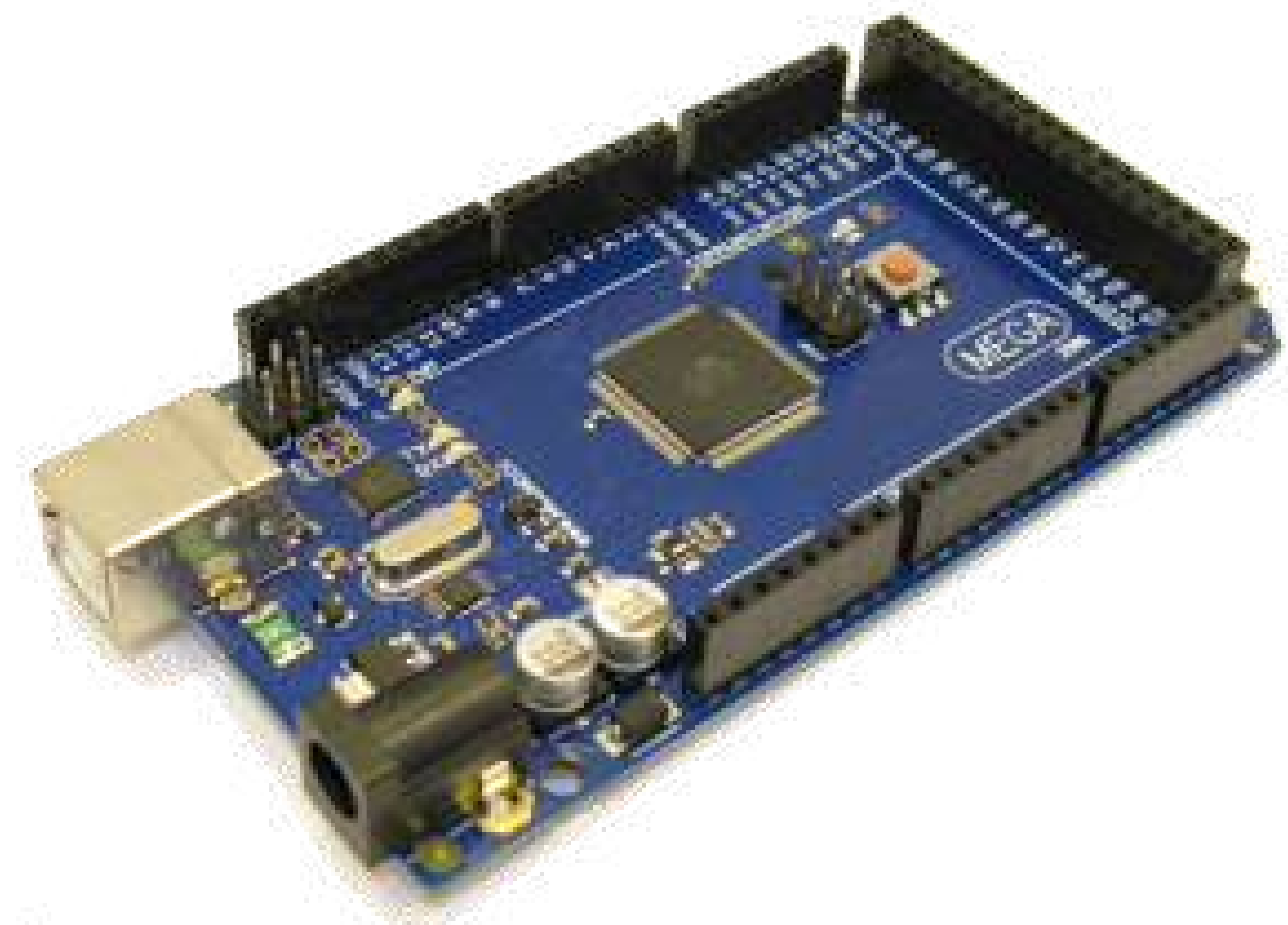
Arduino Mega 2560

ATMega2560 microcontroller

16 MHz CPU

256 KiB Flash

8 KiB SRAM (no cache)



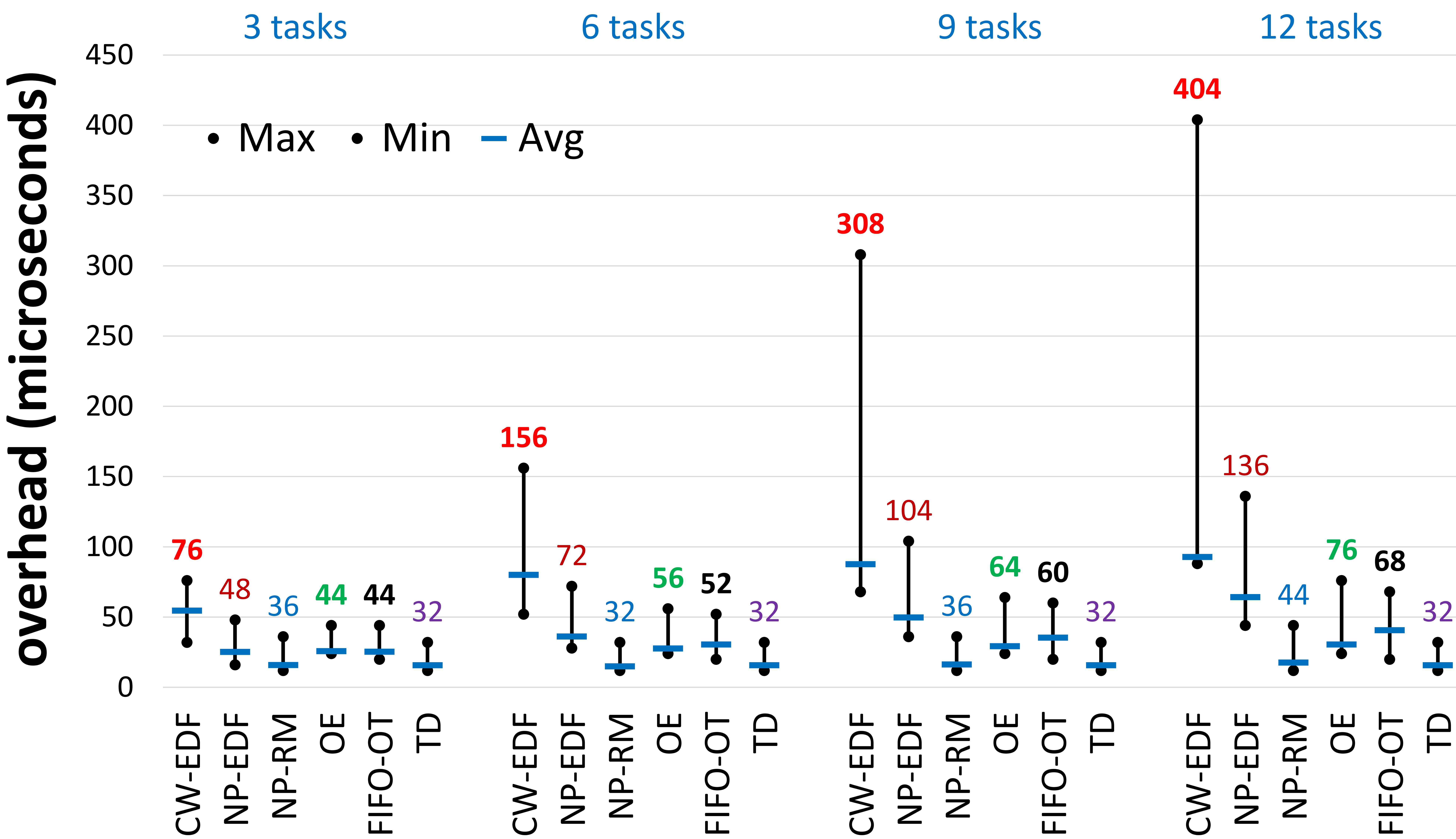
gcc: -Os

http://people.mpi-sws.org/~bbb/papers/details/rtas18

EVALUATED SCHEDULERS

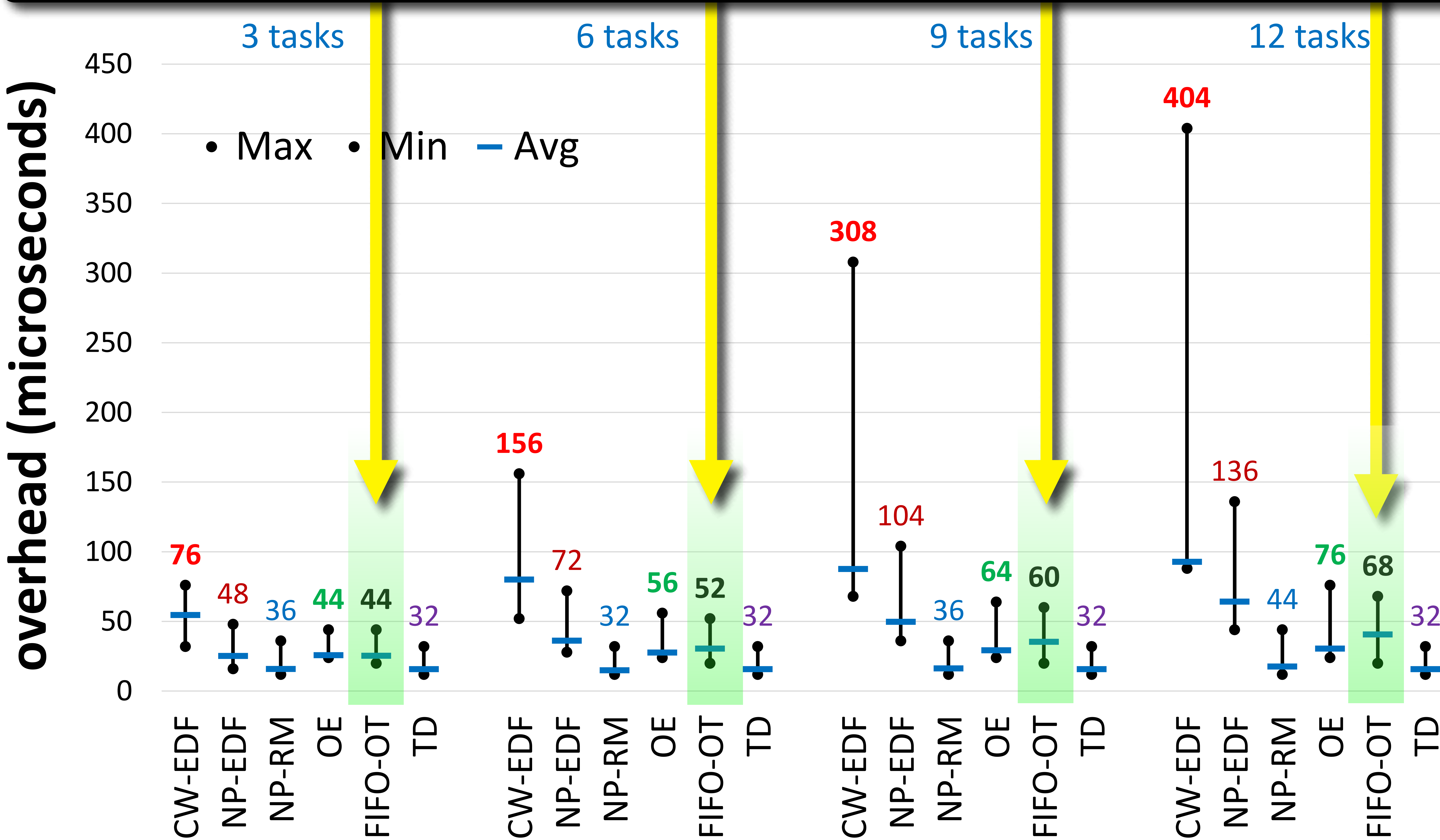
NP-RM	plain non-preemptive rate-monotonic scheduling
NP-EDF	plain non-preemptive EDF
CW-EDF	Critical Window EDF [*Nasri & Fohler, 2016*]
TD	Table-driven (a.k.a. static or time-triggered) scheduling
OE	Offline Equivalence [*Nasri & Brandenburg, 2017*]
FIFO-OT	FIFO + Offset Tuning [*this paper*]

Q1: RUNTIME OVERHEADS



LOW RUNTIME OVERHEADS

FIFO-OT is much cheaper than CW-EDF and roughly similar to NP-RM and OE.



WORKLOADS

based on

Kramer, Ziegenbein, and Hamann, "*Real world automotive benchmark for free,*" WATERS 2015

BOSCH

Periods

➡ non-uniformly in {1, 2, 5, 10, 20, 50, 100, 200, 1000} milliseconds

Runnable BCETs and WCETs

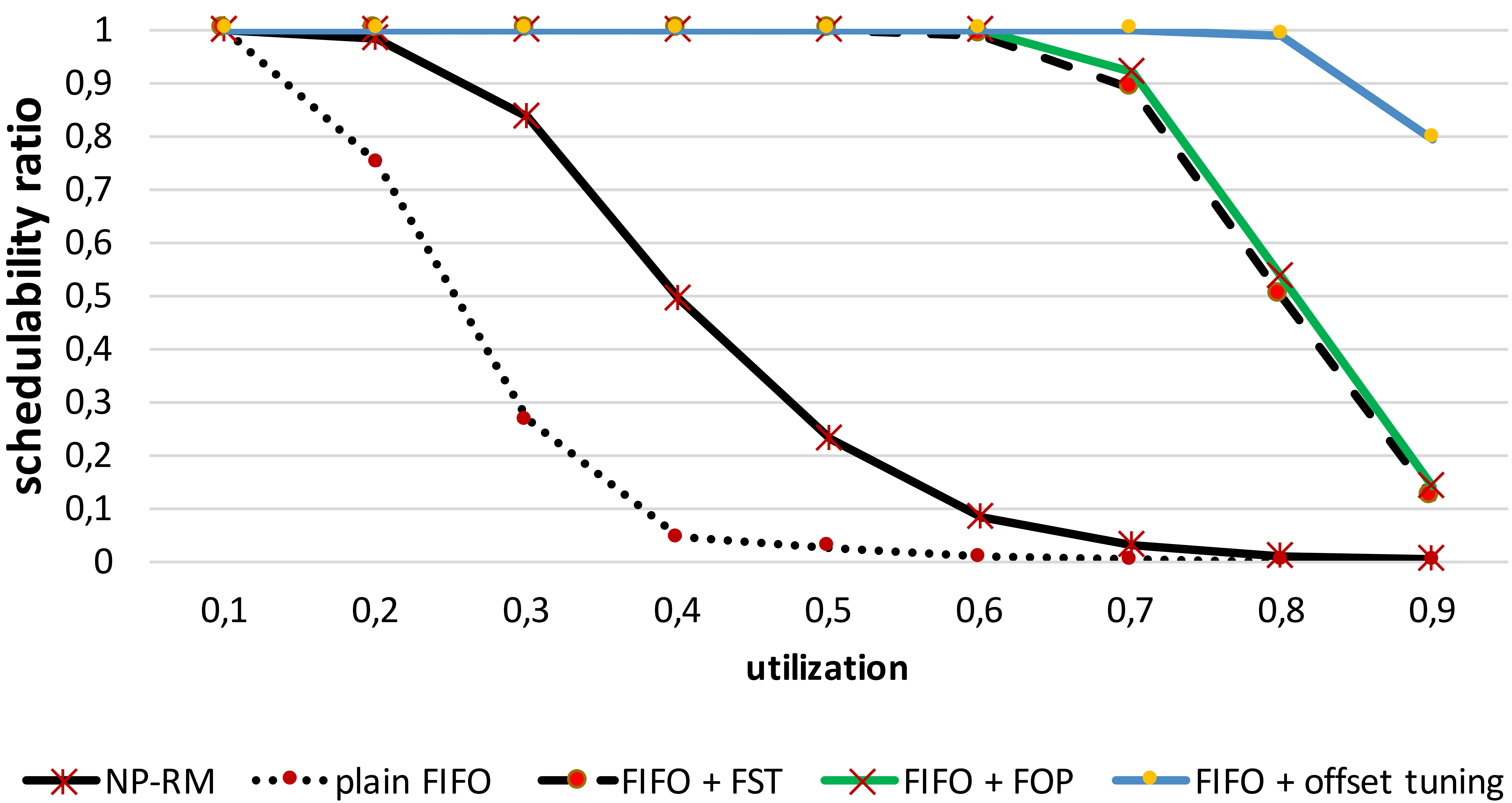
➡ randomly generated based on statistics provided by Kramer et al.

Runnable Packing

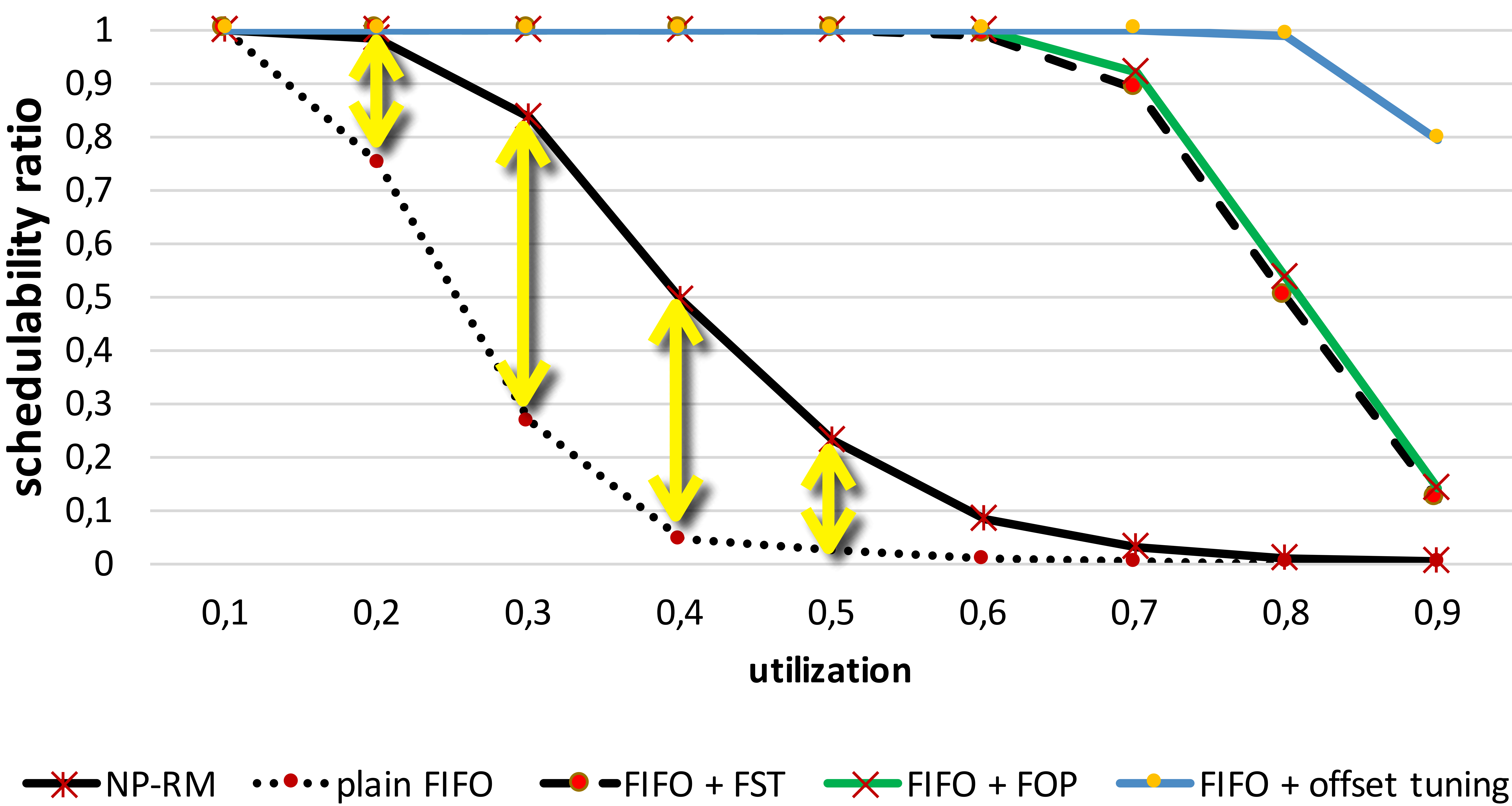
➡ Runnables aggregated into tasks until random utilization threshold reached

➡ *utilization threshold ensures feasibility under non-preemptive scheduling*

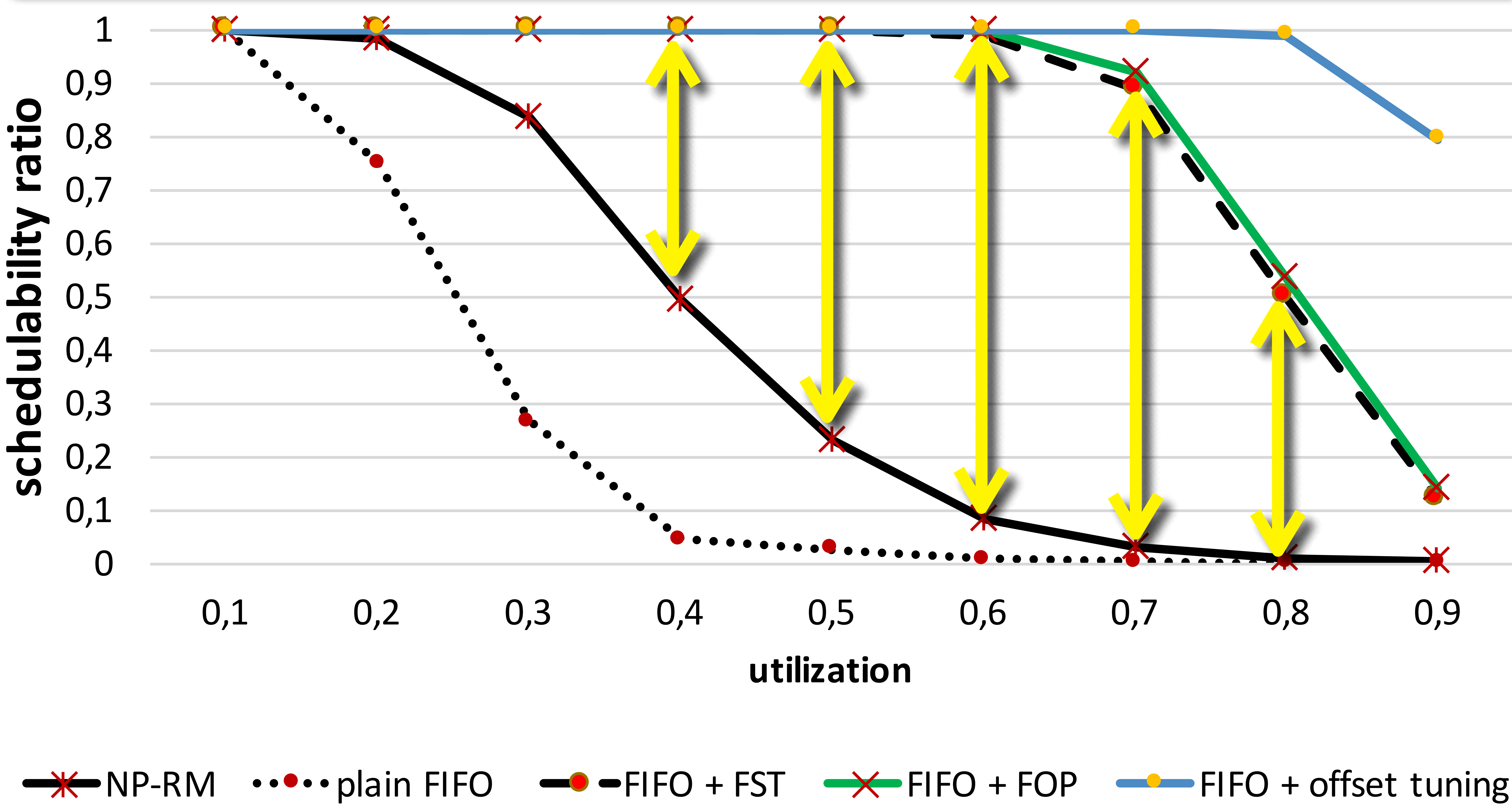
Q2: SCHEDULABILITY GAINS



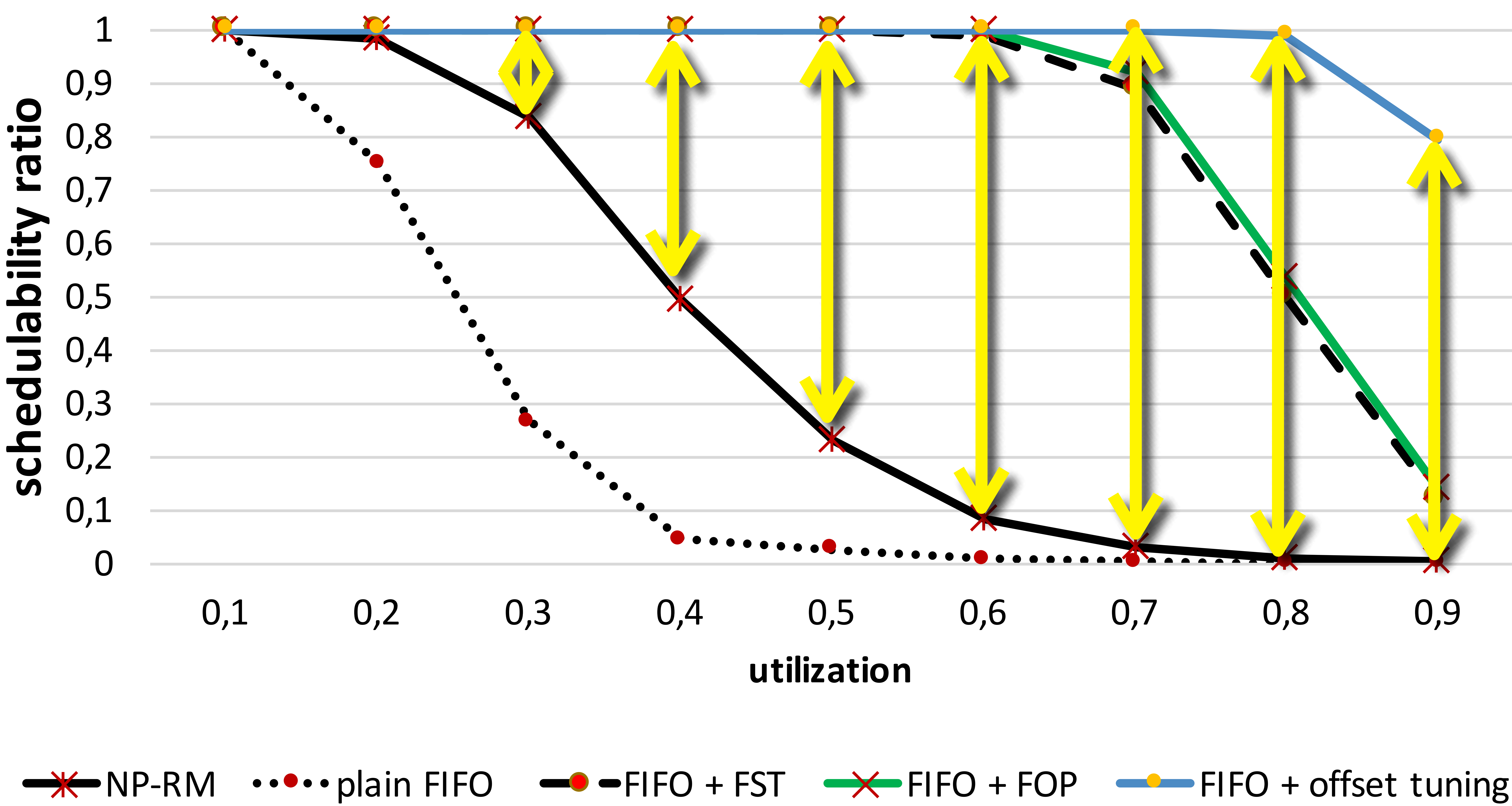
As expected, plain FIFO exhibits very low schedulability.



Assigning even a *single offset* per task can substantially increase schedulability!

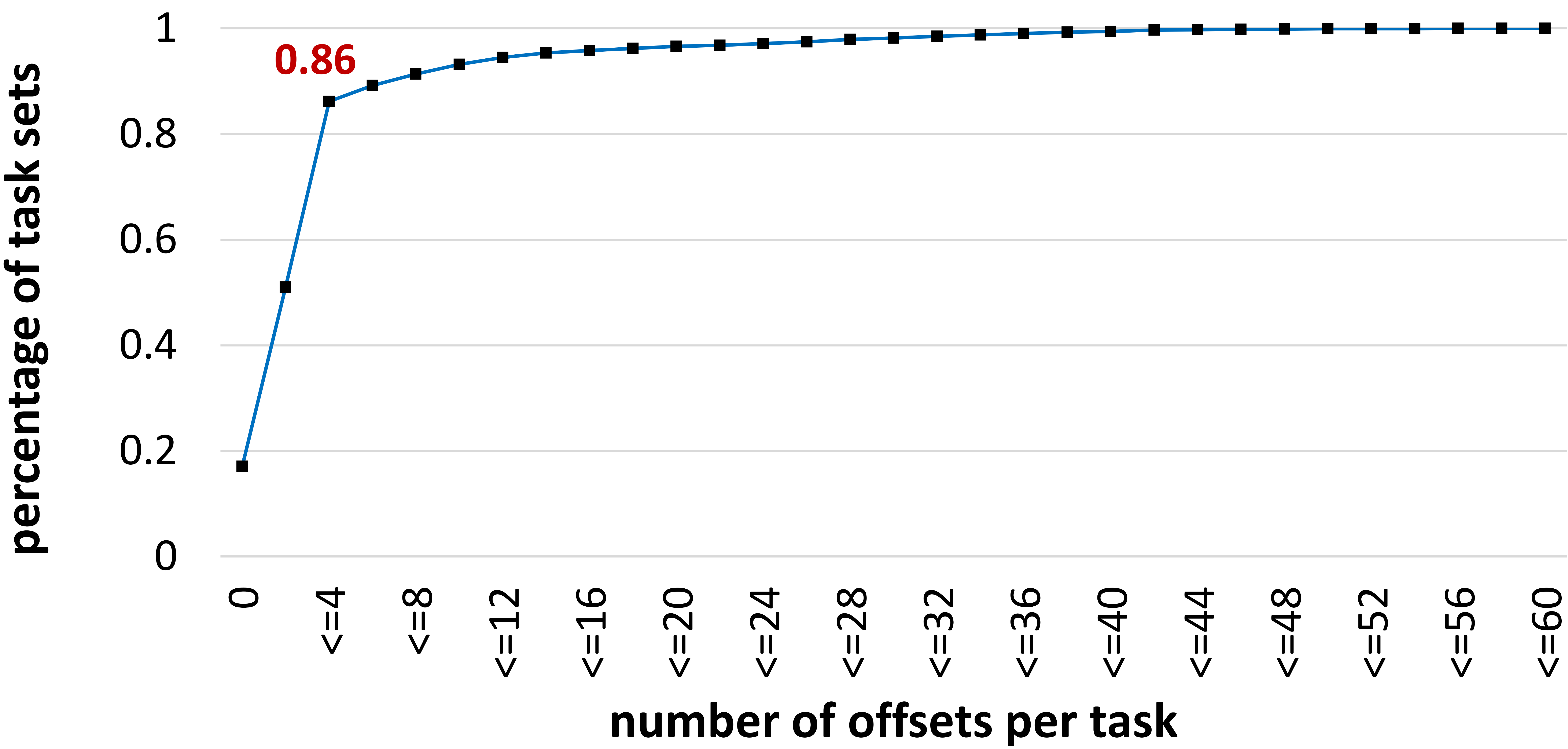


Q2: SCHEDULABILITY GAINS



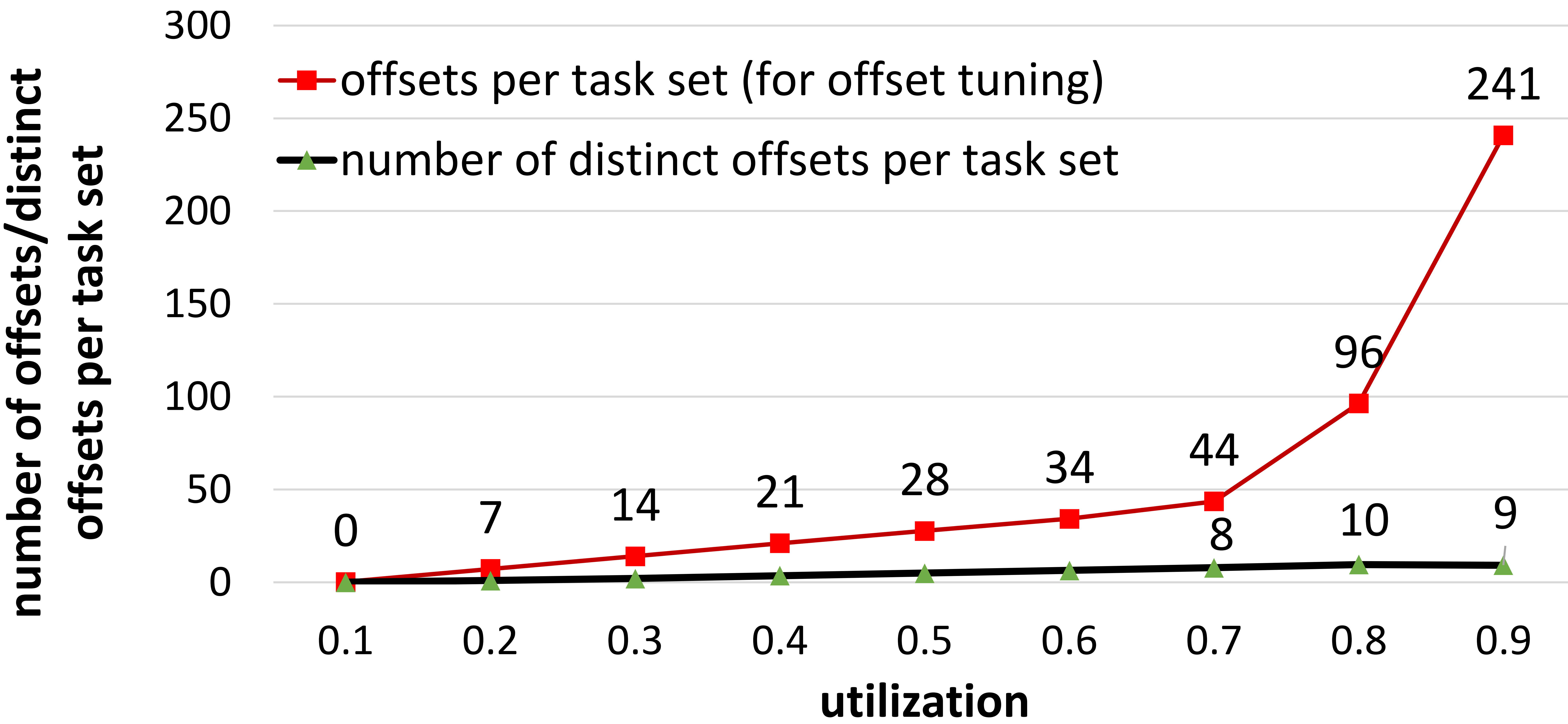
FIFO-OT achieves *much higher schedulability*, thanks to CW-EDF reference schedule.

Q3: NUMBERS OF OFFSETS PER TASK



➔ *Most tasks require only few offset partitions.*

NUMBERS OF UNIQUE OFFSETS PER TASK SET

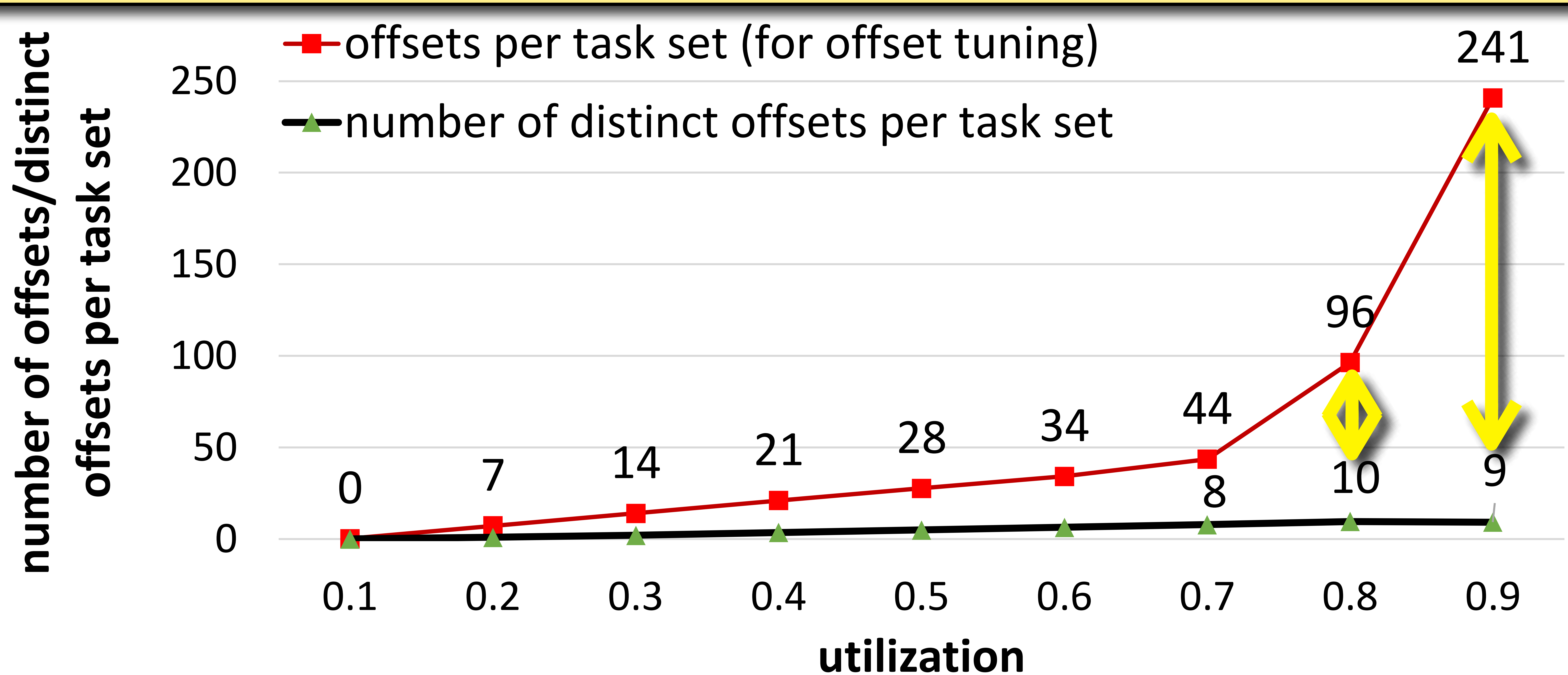


*Across the hyper-period, **offsets values repeat cyclicly**.*

➔ *Opportunity to store offsets efficiently (compression).*

NUMBERS OF UNIQUE OFFSETS PER TASK SET

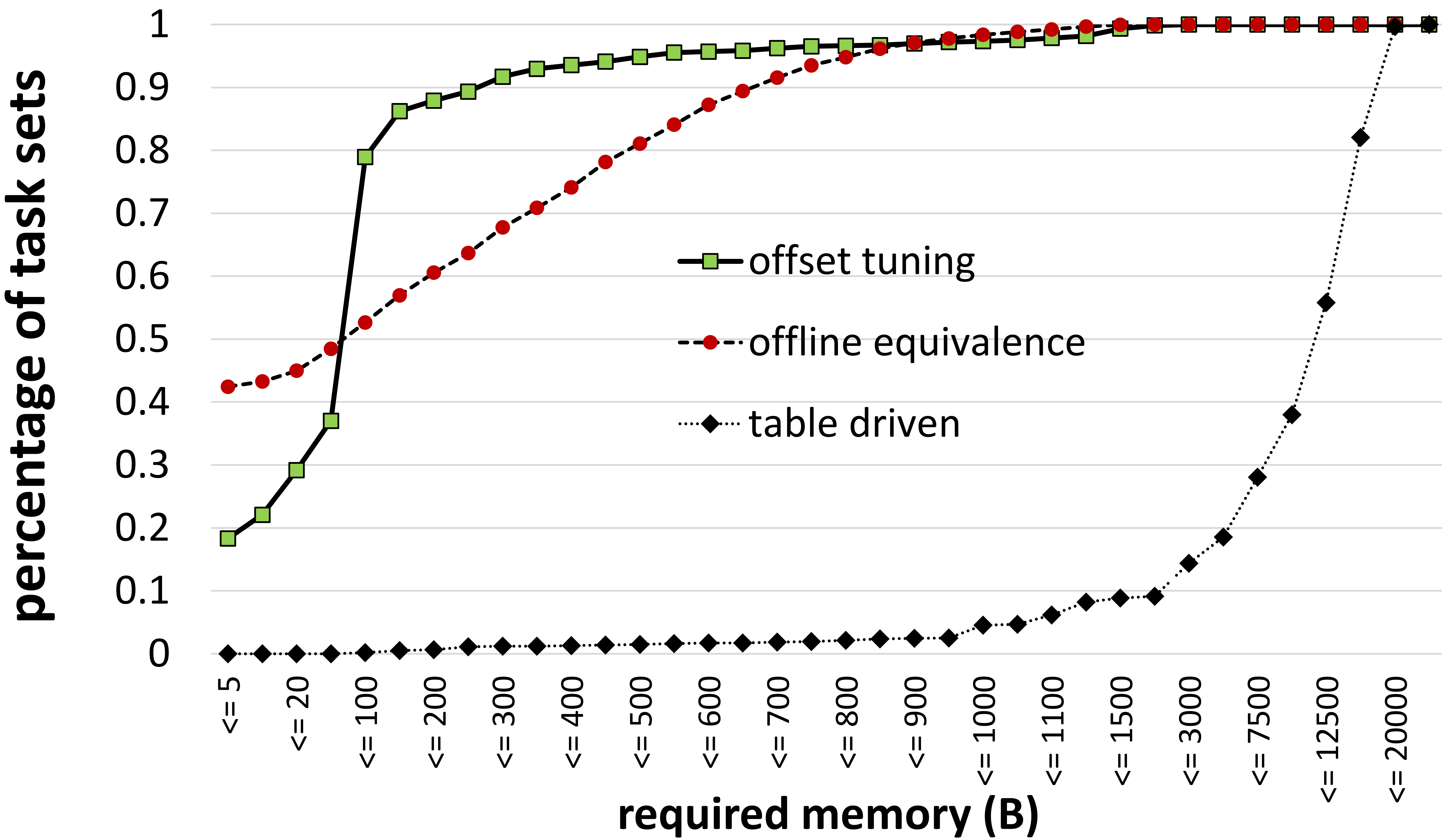
Up to 25× reduction in the number of offset values that must be stored.



Across the hyper-period, ***offsets values repeat cyclicly****.*

➔ *Opportunity to store offsets efficiently (compression).*

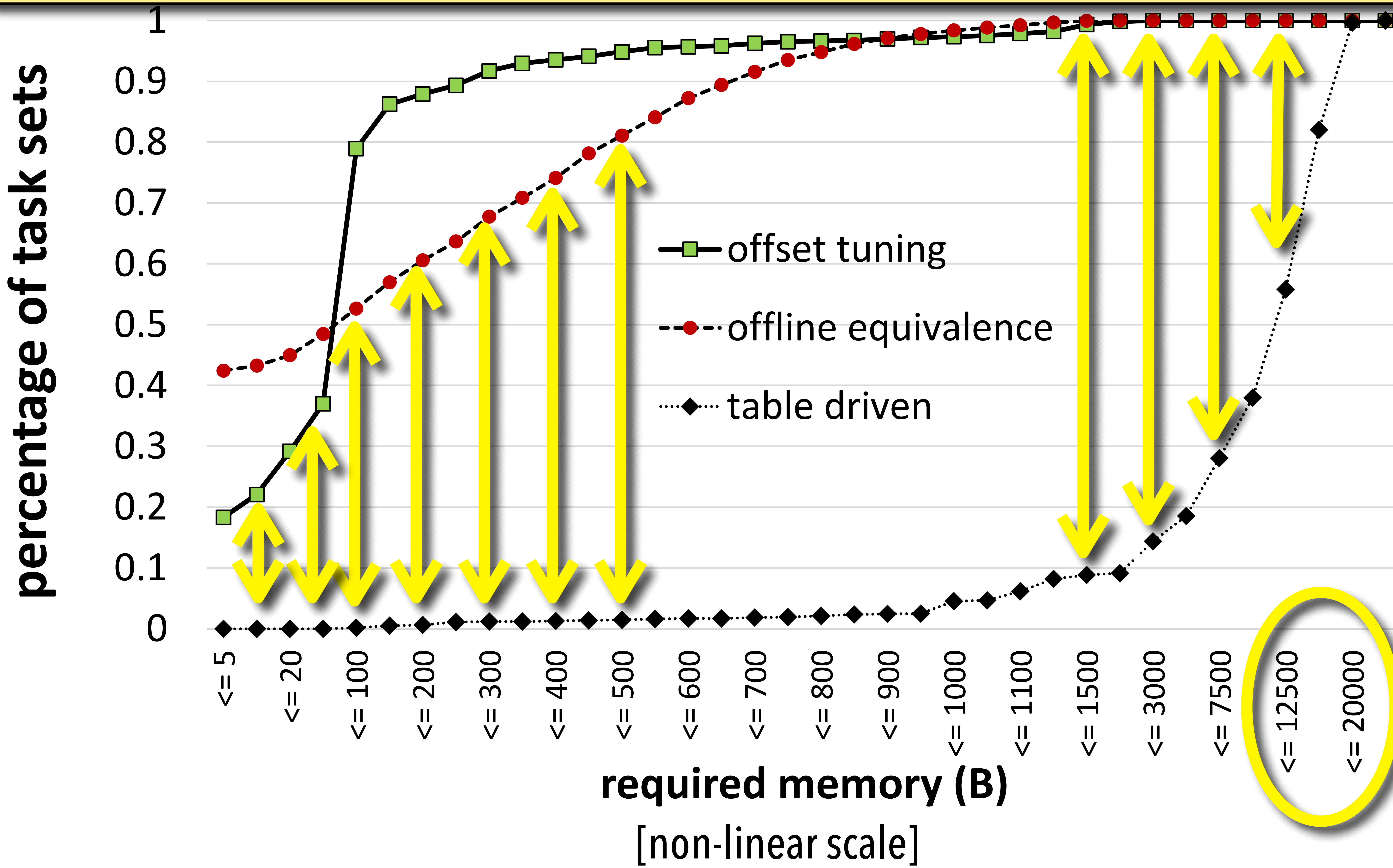
MEMORY USAGE



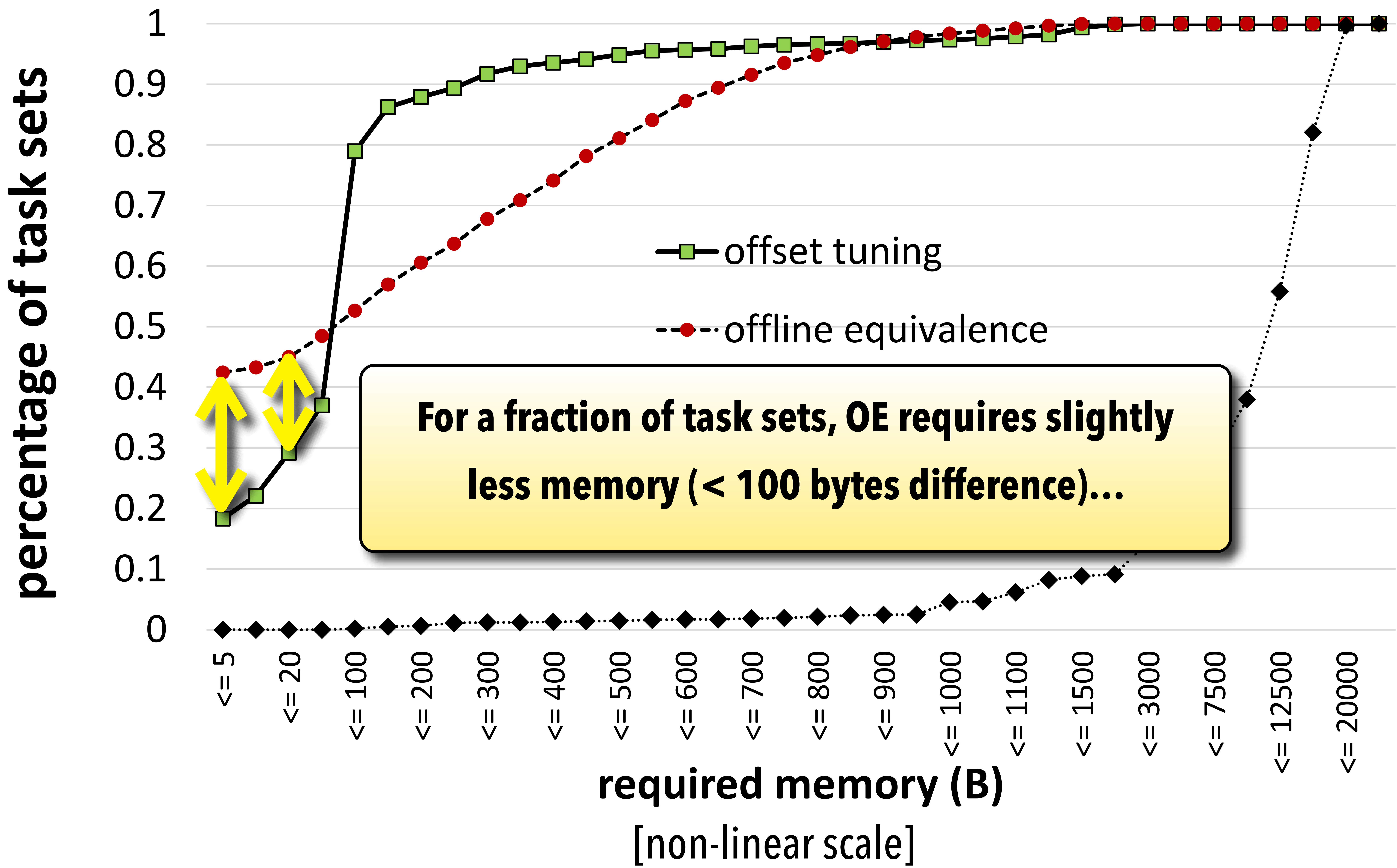
[non-linear scale]

Both *OE and FIFO-OT require much less memory* than table-driven scheduling.

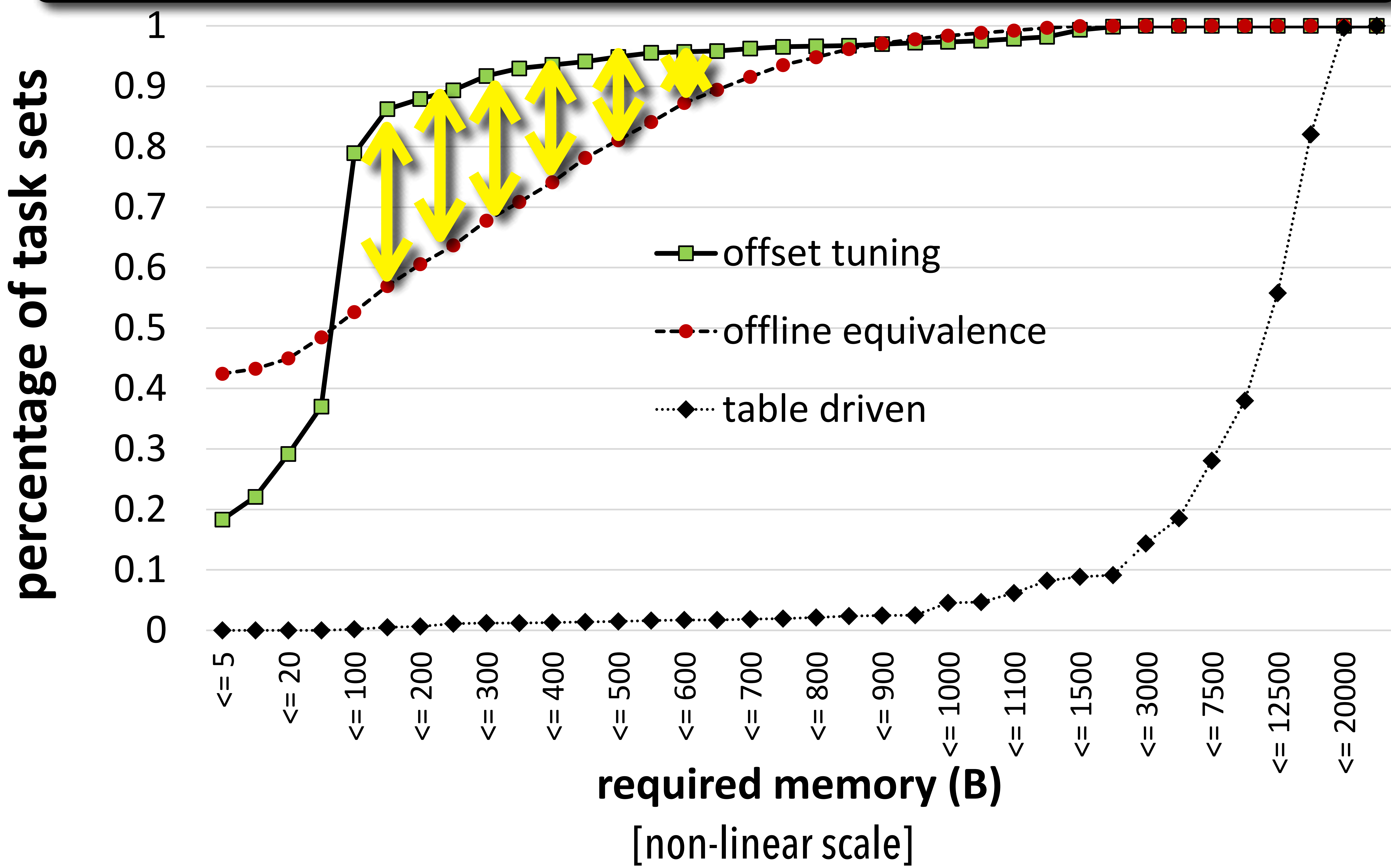
dozens to hundreds of bytes vs. 10KiB-20KiB



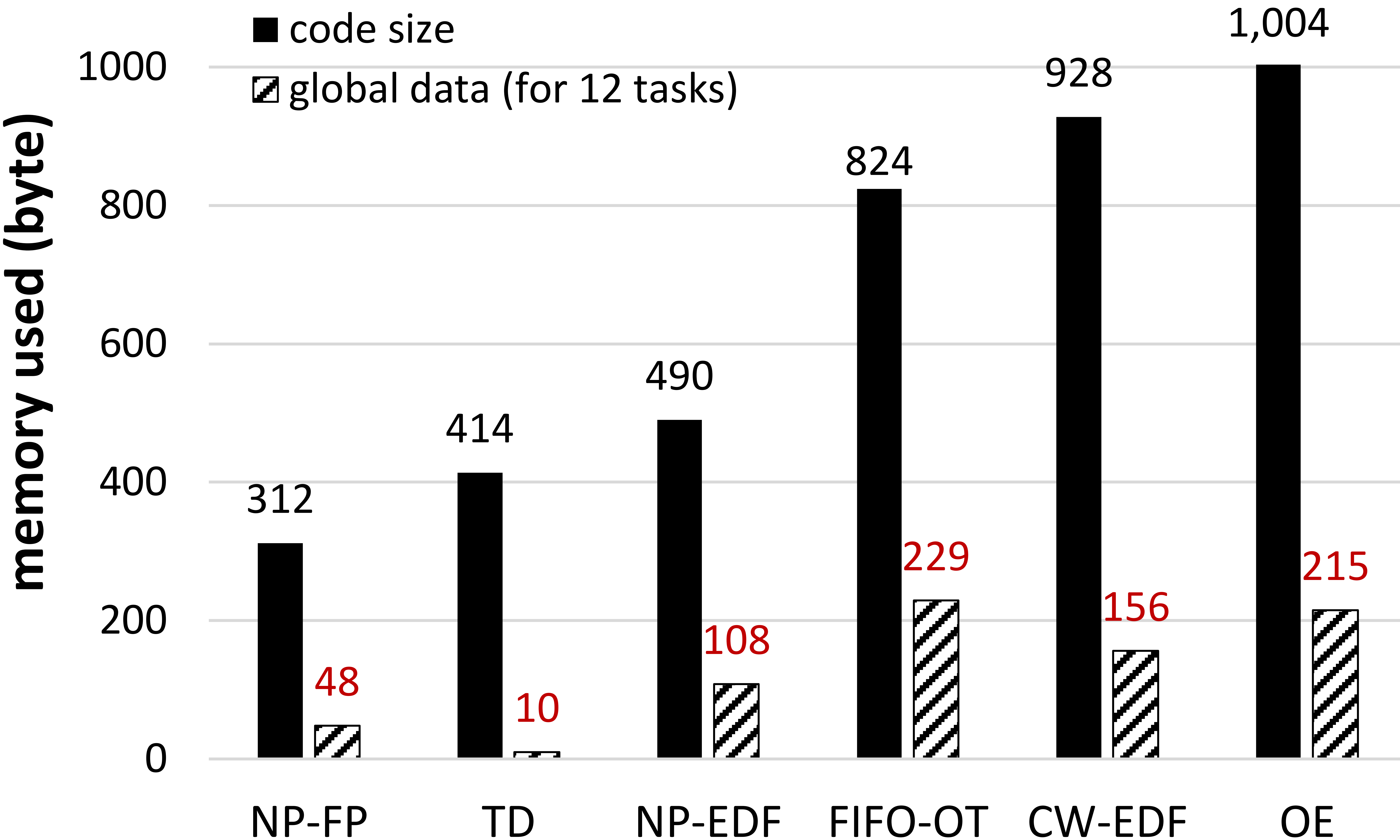
MEMORY USAGE



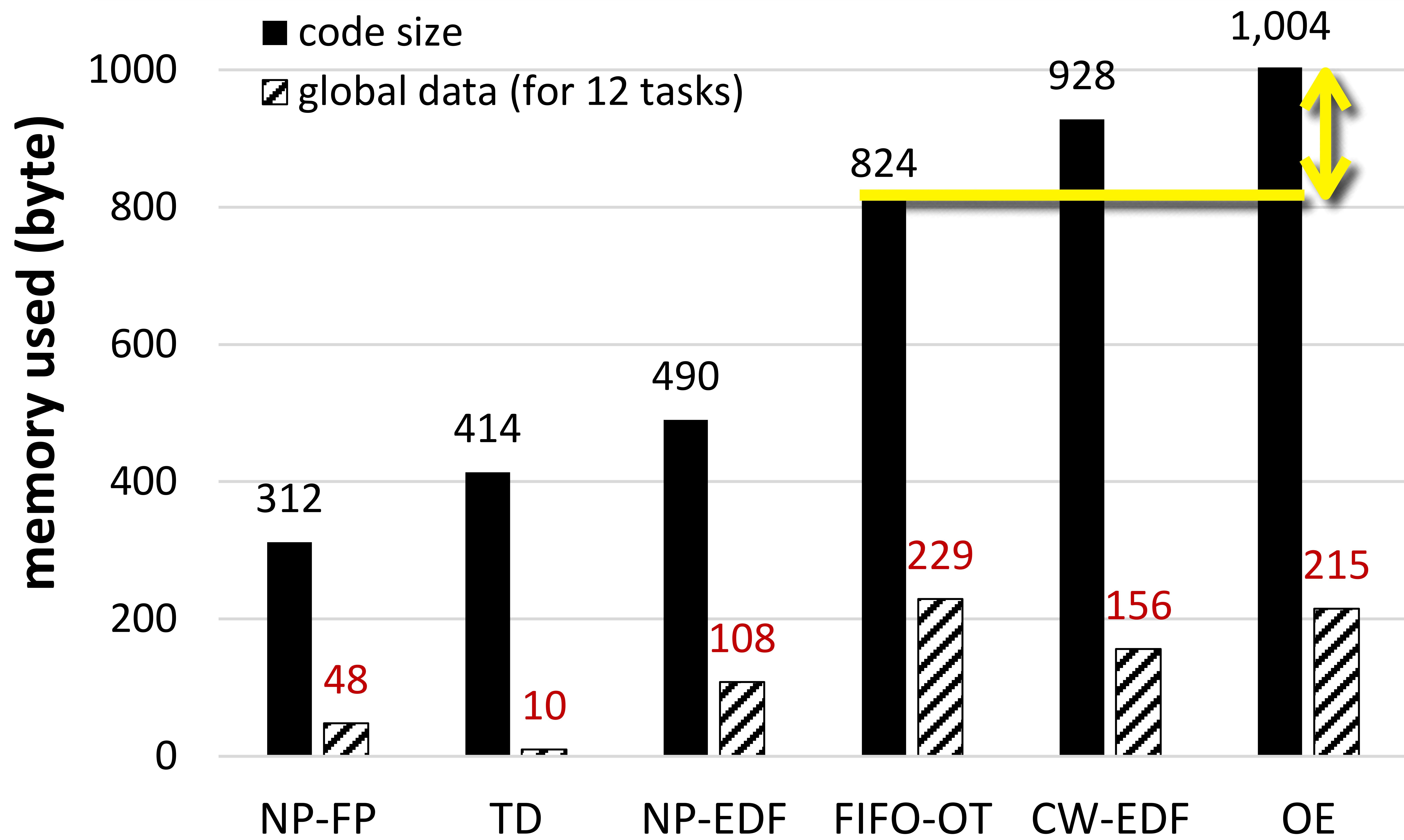
...but FIFO-OT can support over *90% of task sets with ≤ 250 bytes* of offset data.



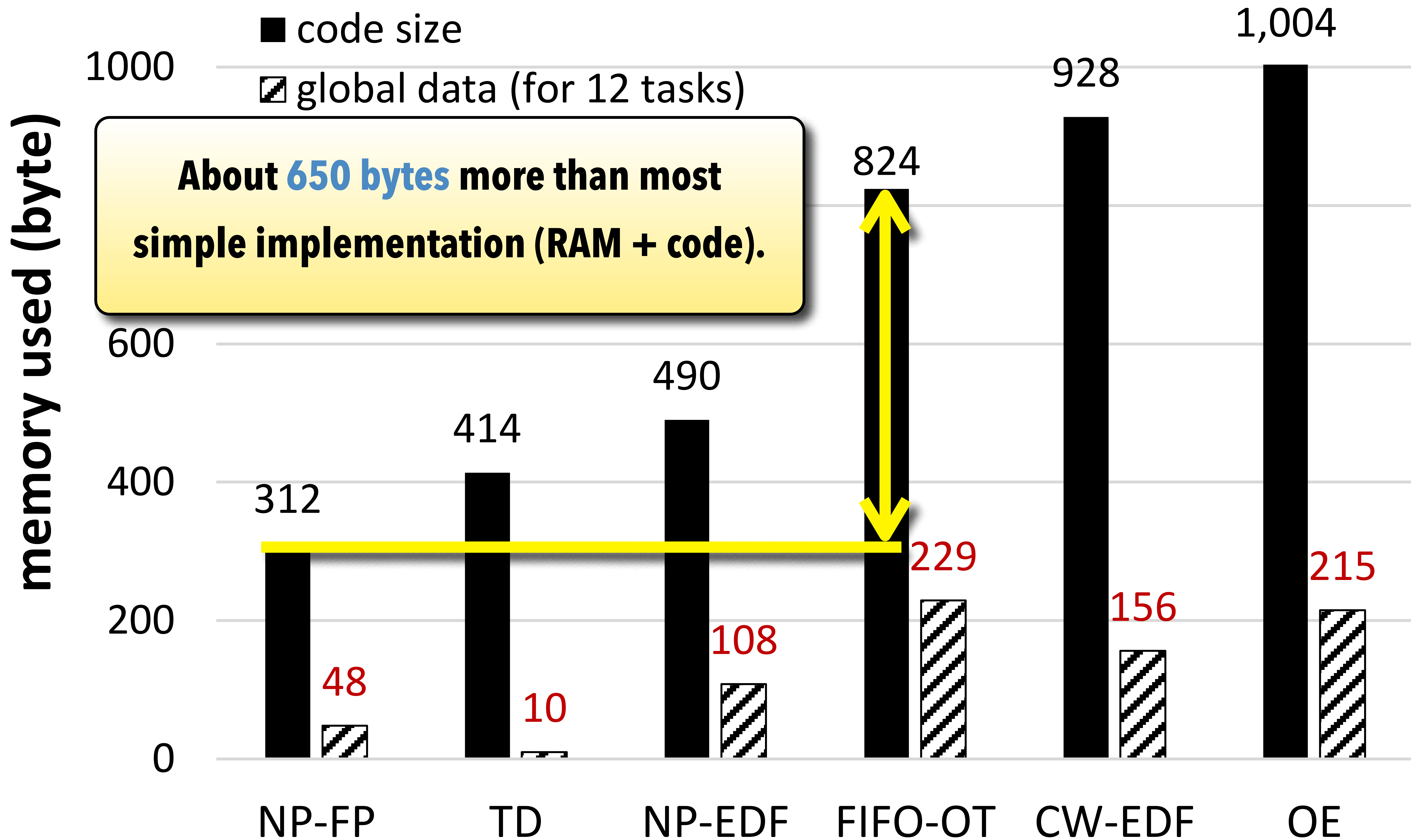
IMPLEMENTATION FOOTPRINT



About 150 bytes *smaller footprint* than OE (RAM + code).

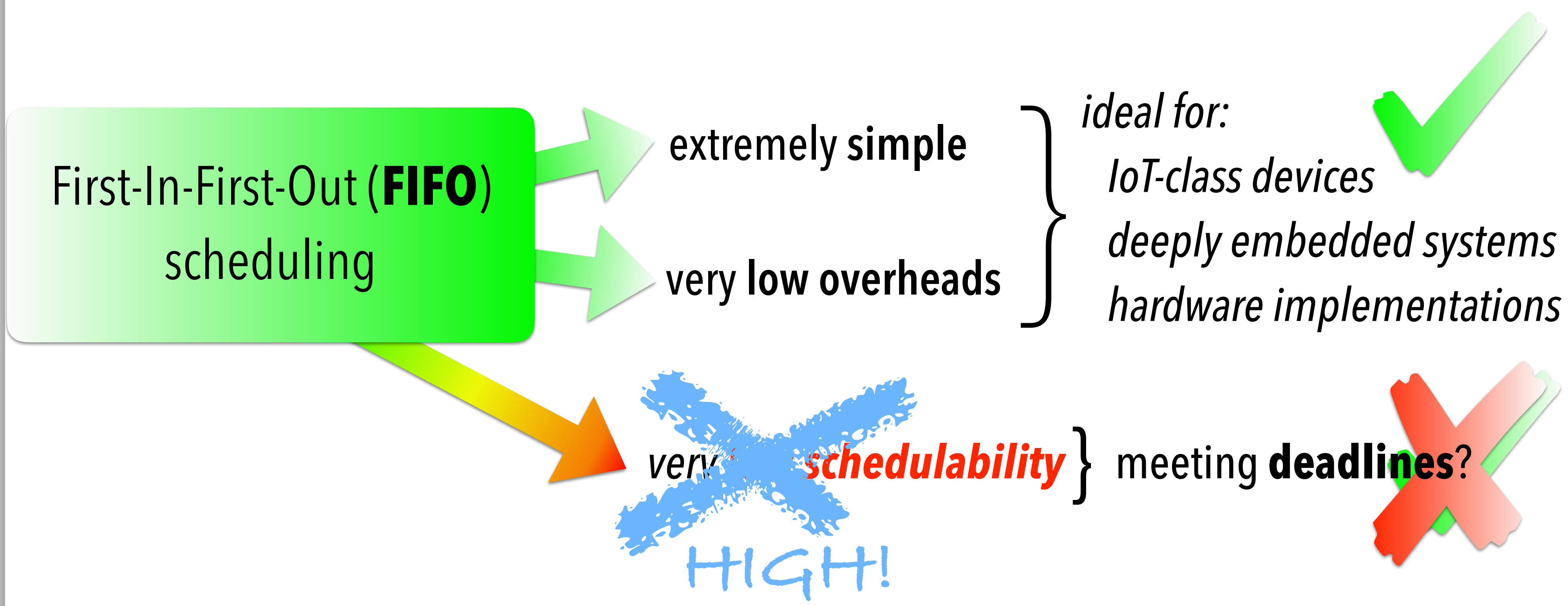


IMPLEMENTATION FOOTPRINT



CONCLUSION

FIFO SCHEDULING

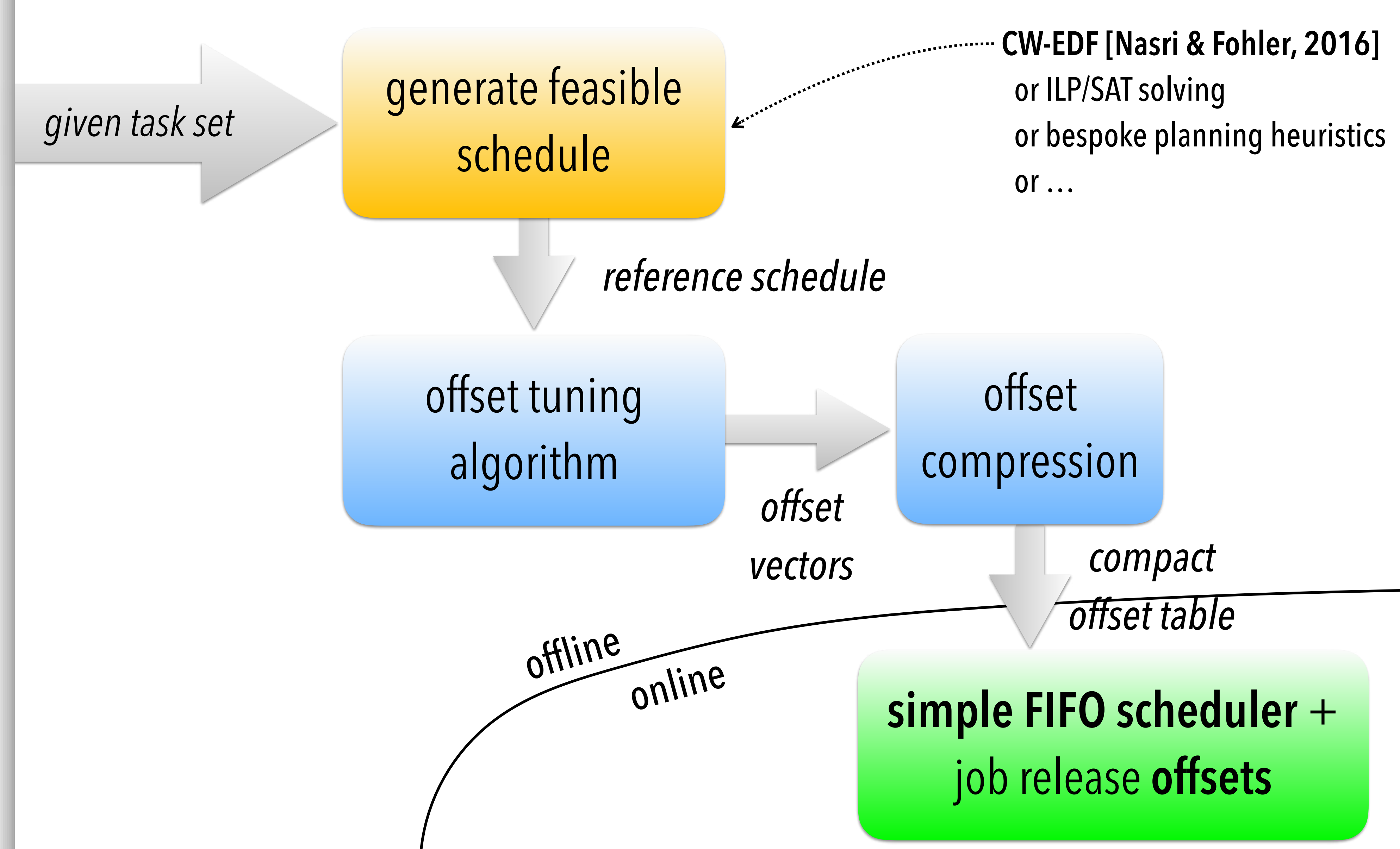


THIS PAPER

FIFO can actually achieve excellent schedulability!

[**periodic non-preemptive tasks** on a **uniprocessor**]

OFFSET TUNING – OVERVIEW



PROPERTIES OF OFFSET TUNING

REFERENCE SCHEDULE EQUIVALENCY

In the resulting FIFO schedule, ***no job completes later*** *than in the original reference schedule.*

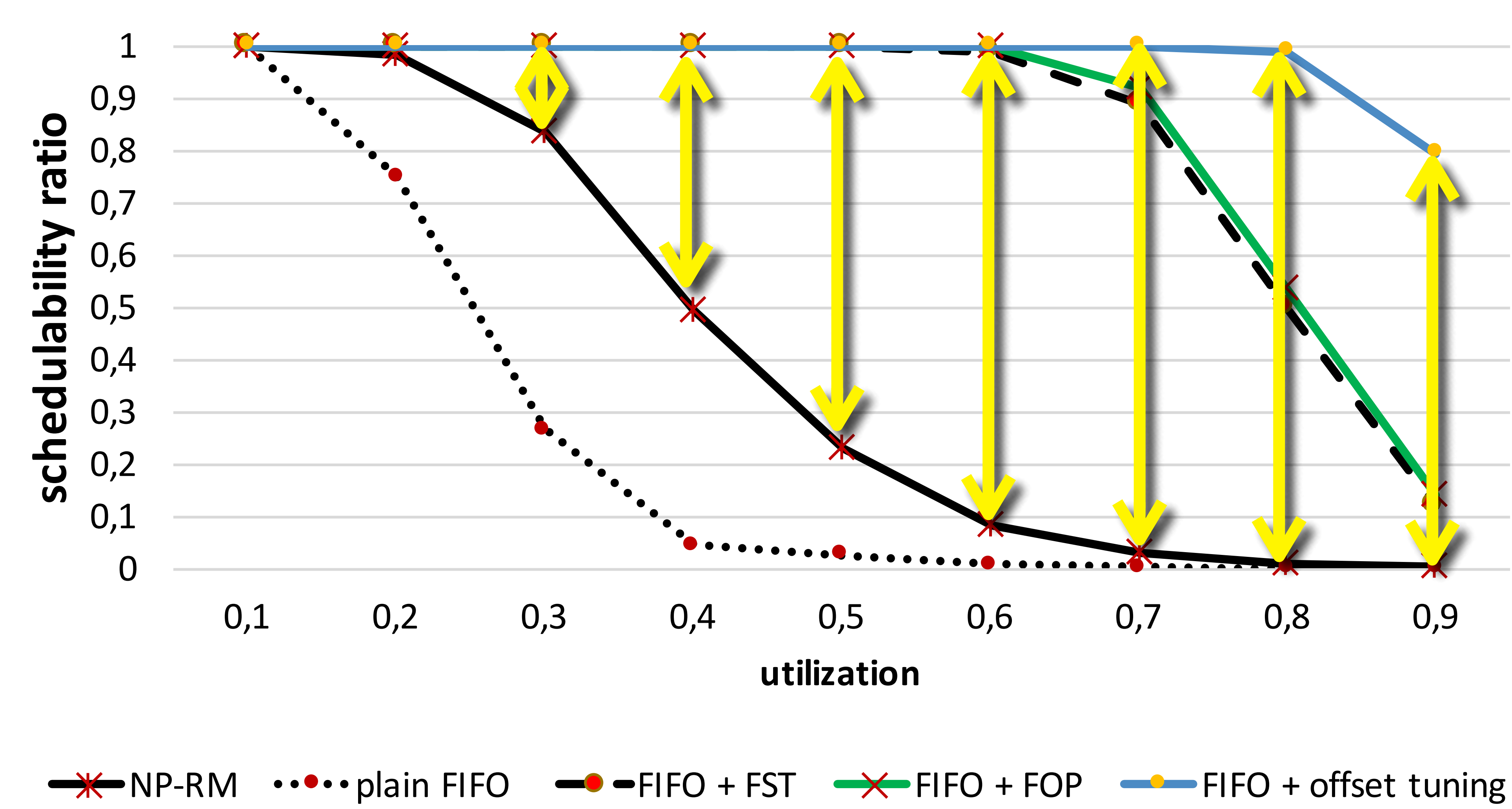
PER-TASK MINIMAL OFFSET PARTITIONS

The greedy offset partitioning strategy yields a minimal number of offset partitions (for a given task).

NON-MINIMAL OFFSET PARTITIONS FOR ENTIRE TASK SET

Deadline-monotonic processing order does not guarantee overall minimal number of offset partitions (but ***works well empirically****).*

Q2: SCHEDULABILITY GAINS



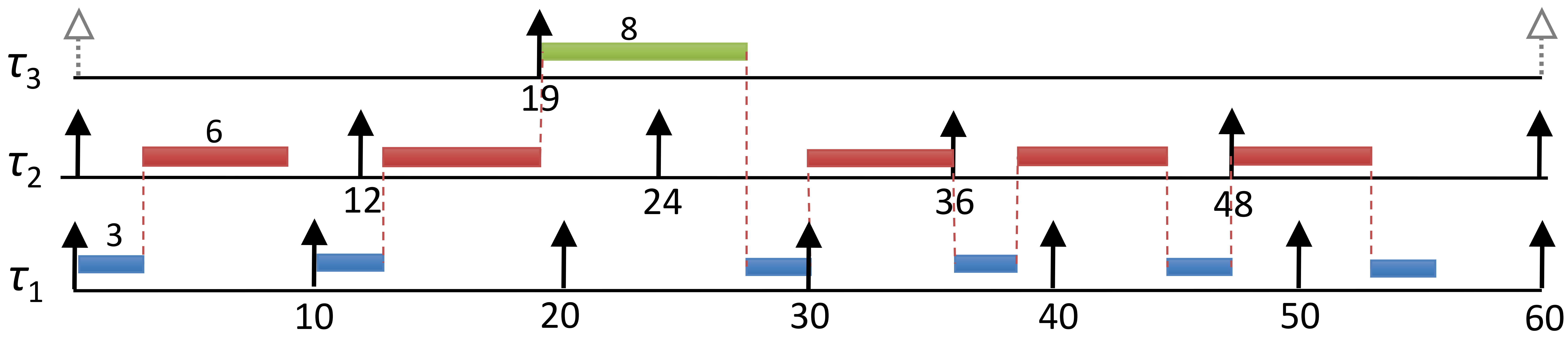
FIFO-OT achieves ***much higher schedulability*****, thanks to CW-EDF reference schedule.**

APPENDIX

CAN OFFSET TUNING BE APPLIED TO EDF OR FIXED-PRIORITY SCHEDULING?

➔ *yes in principle, but no equivalence guarantee*

FIFO schedule + ***offset*** ***for*** τ_3 :



RM schedule + ***offset*** ***for*** τ_3 :