FRAMEWORK-AGNOSTIC MODEL INFERENCE FOR INTRA-THREAD REAL-TIME TASKS

32nd IEEE Real-Time and Embedded Technology
and Applications Symposium
May 12, 2026

Bite Ye[1]
Filip Marković[2]
Björn Brandenburg[1]
[1]Max Planck Institute for Software Systems, Germany
[2]University of Southampton, United Kingdom

LiME IT

FRAMEWORK-AGNOSTIC MODEL INFERENCE FOR IN RA-THREAD REAL-TIME TASKS

C, C++, Rust, ROS ...

eal-Time and Embedded Technology
d Applications Symposium
May 12, 2026

Bite Ye[1]
Filip Marković[2]
Björn Brandenburg[1]
[1]Max Planck Institute for Software Systems, Germany
[2]University of Southampton, United Kingdom

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

University of Southampton

LiME^IT

FRAMEWORK-AGNOSTIC MODEL INFERENCE FOR INTRA-THREAD REAL-TIME TASKS

Sporadic, Arrival Curves, Periodic ...

32nd IEEE Real-Time and Emb
and Applications Sy
May 12, 202

Bite Ye[1]
Filip Marković[2]
Björn Brandenburg[1]
[1]Max Planck Institute for Software Systems, Germany
[2]University of Southampton, United Kingdom

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

University of Southampton

LiME IT

FRAMEWORK-AGNOSTIC MODEL INFERENCE FOR INTRA-THREAD REAL-TIME TASKS

32nd IEEE Real-Time and Embedded Technology and Applications Symposium

Async futures, callbacks, closures ...

Bite Ye[1]

Filip Marković[2]

Björn Brandenburg[1]

[1]Max Planck Institute for Software Systems, Germany

[2]University of Southampton, United Kingdom

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

University of Southampton

FRAMEWORK-AGNOSTIC MODEL INFERENCE FOR INTRA-THREAD REAL-TIME TASKS

We extract timing models for intra-thread tasks in a blackbox framework-agnostic setting

Bite Ye[1]
Filip Marković[2]
Björn Brandenburg[1]
[1]Max Planck Institute for Software Systems, Germany
[2]University of Southampton, United Kingdom

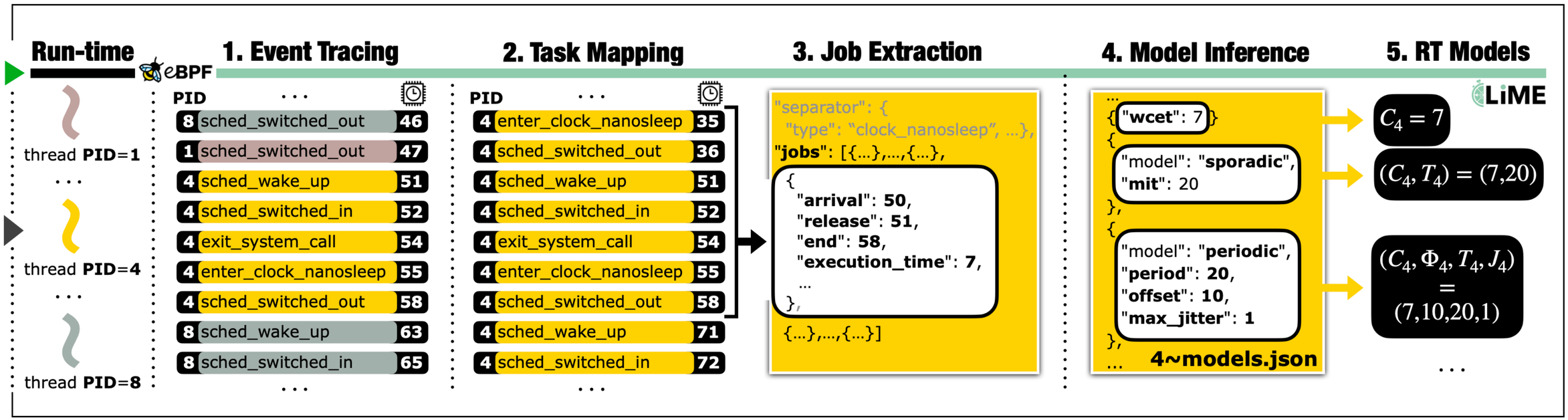
MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

University of Southampton

What's LiME?

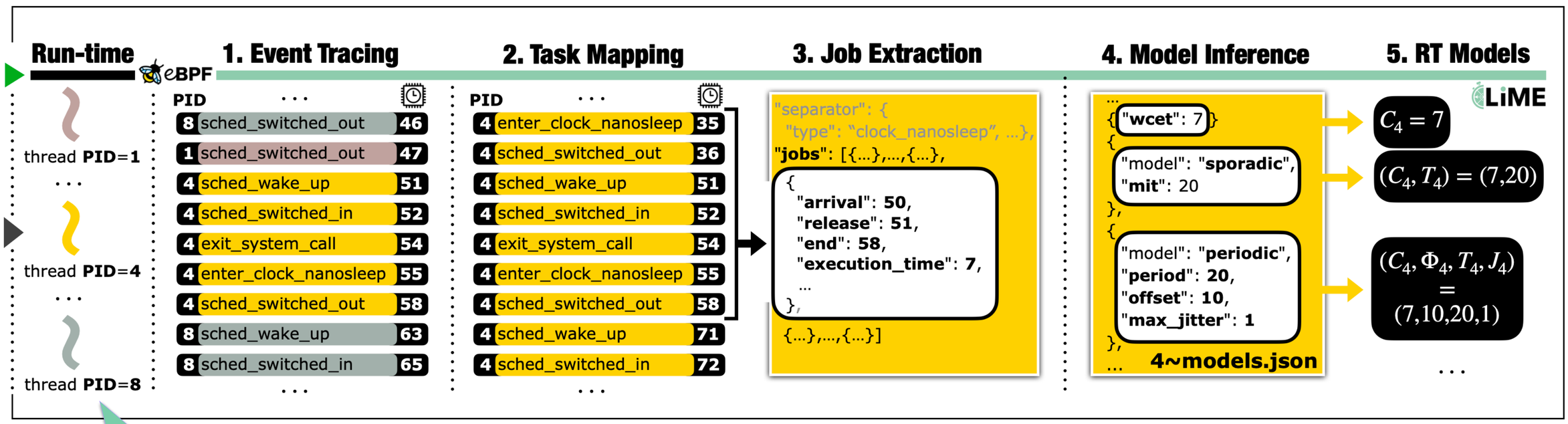
LIME IS A DYNAMIC MODLE EXTRACTOR FOR LINUX

LiME: The Linux Real-Time Task Model Extractor.
RTAS 2025, pp. 255–269.



LIME IS A DYNAMIC MODLE EXTRACTOR FOR LINUX

LiME: The Linux Real-Time Task Model Extractor.
RTAS 2025, pp. 255–269.

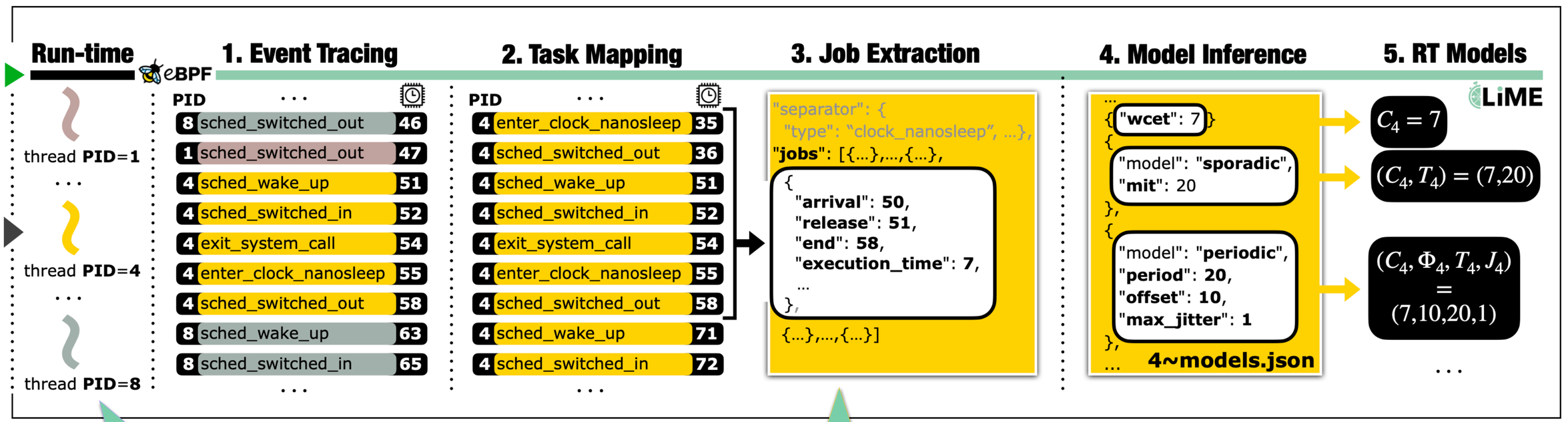


Blackbox Threads

without access to source code or static analysis of binary

LIME IS A DYNAMIC MODLE EXTRACTOR FOR LINUX

LiME: The Linux Real-Time Task Model Extractor.
RTAS 2025, pp. 255–269.



Blackbox Threads

without access to source code or static analysis of binary

Identify Individual Task Activations

RT task models assume repeatedly activated tasks:

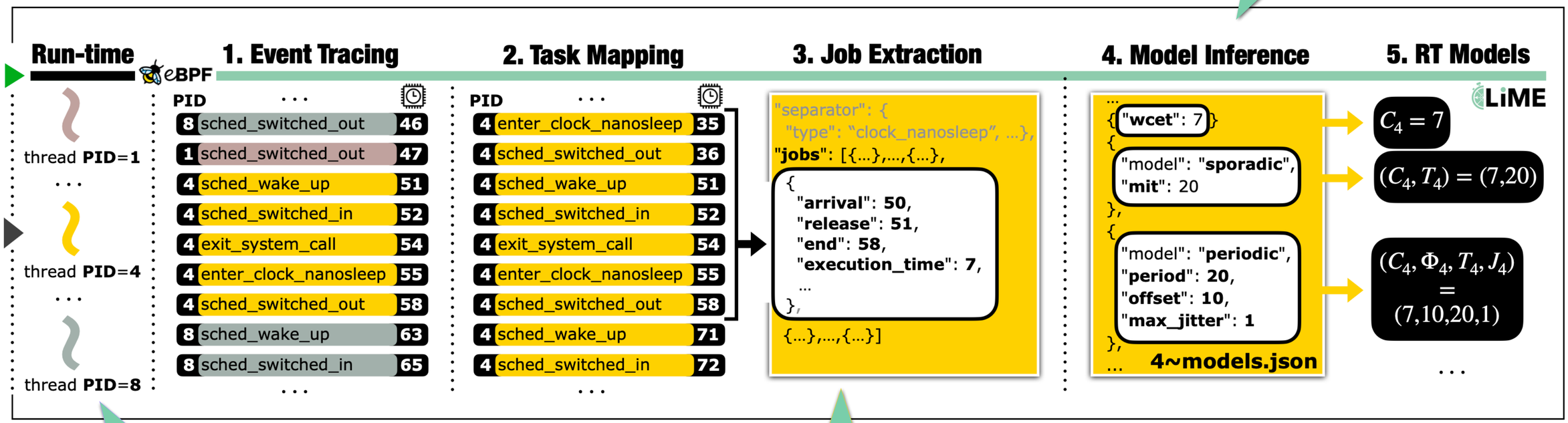
task = stream of jobs

LIME IS A DYNAMIC MODLE EXTRACTOR FOR LINUX

LiME: The Linux Real-Time Task Model Extractor.
RTAS 2025, pp. 255–269.

Online Model Inference

Infer task parameters



Blackbox Threads

without access to source code or static analysis of binary

Identify Individual Task Activations

RT task models assume repeatedly activated tasks:

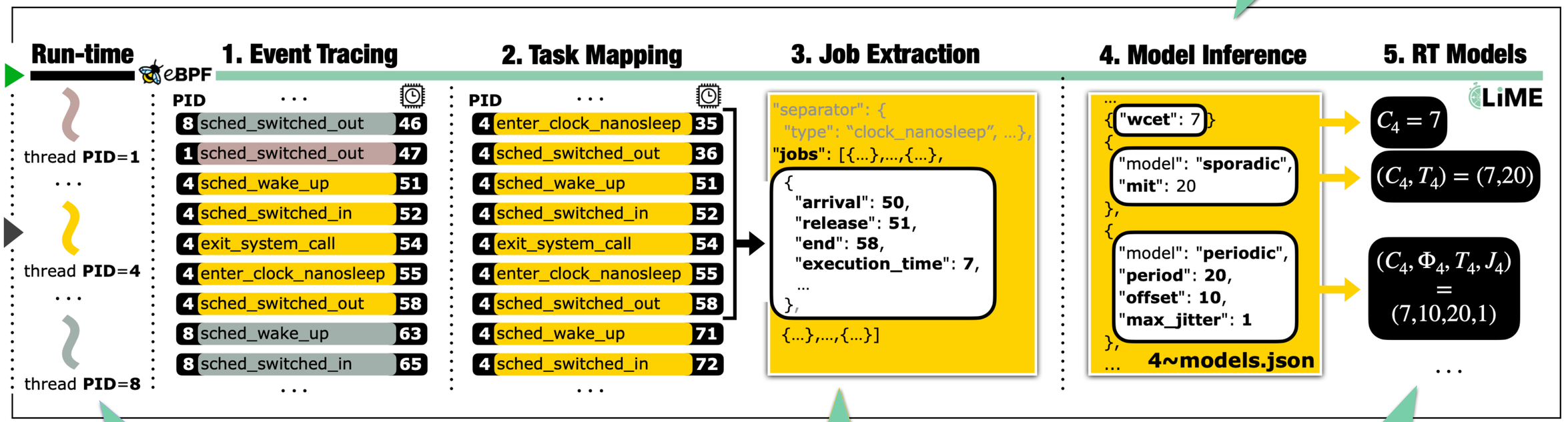
task = stream of jobs

LIME IS A DYNAMIC MODLE EXTRACTOR FOR LINUX

LiME: The Linux Real-Time Task Model Extractor.
RTAS 2025, pp. 255–269.

Online Model Inference

Infer task parameters



Blackbox Threads

without access to source code or static analysis of binary

Identify Individual Task Activations

RT task models assume repeatedly activated tasks:

task = stream of jobs

Output

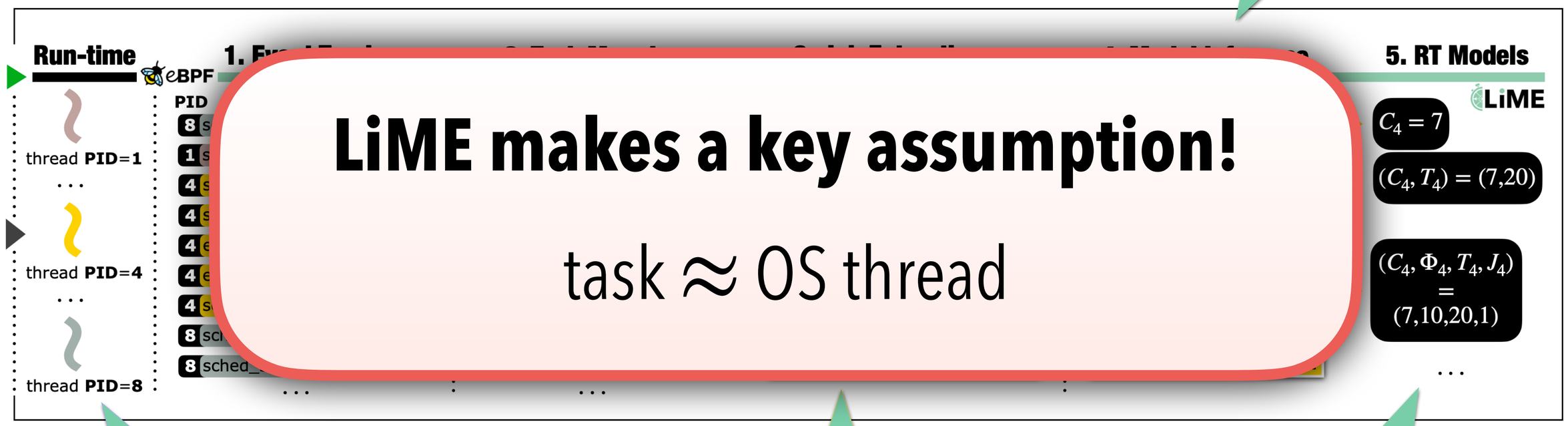
Sporadic, Arrival Curves, Periodic ...

LIME IS A DYNAMIC MODLE EXTRACTOR FOR LINUX

LiME: The Linux Real-Time Task Model Extractor.
RTAS 2025, pp. 255–269.

Online Model Inference

Infer task parameters



Blackbox Threads

without access to source code or static analysis of binary

Identify Individual Task Activations

RT task models assume repeatedly activated tasks:

task = stream of jobs

Output

Sporadic, Arrival Curves, Periodic ...

LET'S GO BEYOND

THREAD LEVEL

THREADS CONTAIN INTRA-THREAD TASKS

Modern systems use **frameworks** to map many **logical tasks** onto **fewer OS threads**

THREADS CONTAIN INTRA-THREAD TASKS

Modern systems use **frameworks** to map many **logical tasks** onto **fewer OS threads**

We call these intra-thread tasks

e.g: callbacks, futures, closures

THREADS CONTAIN INTRA-THREAD TASKS

Modern systems use **frameworks** to map many **logical tasks** onto **fewer OS threads**

THREADS CONTAIN INTRA-THREAD TASKS

Modern systems use **frameworks** to map many **logical tasks** onto **fewer OS threads**

Example Code Snippet

Rust futures

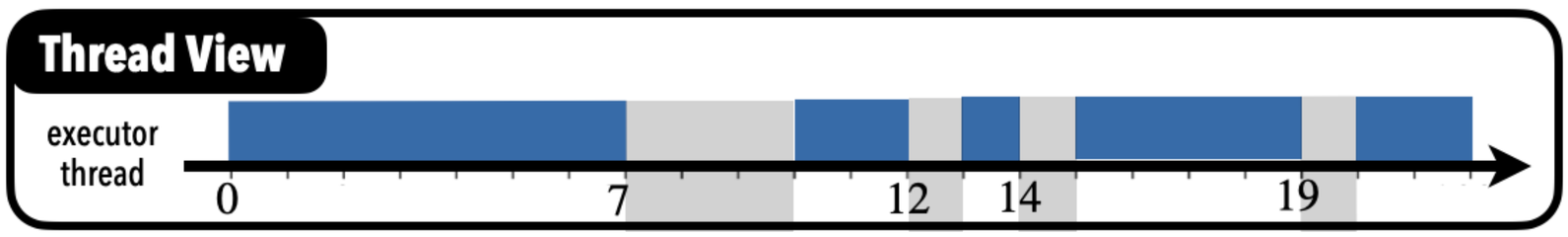
```
let task = async move {
  timer.tick().await;
  control_step().await;
};
executor.spawn(task);
```

ROS 2 callback (C++)

```
auto cb = [](auto msg) {
  update_model(msg);
};
node.create_subscription("scan", qos, cb);
executor.spin();
```

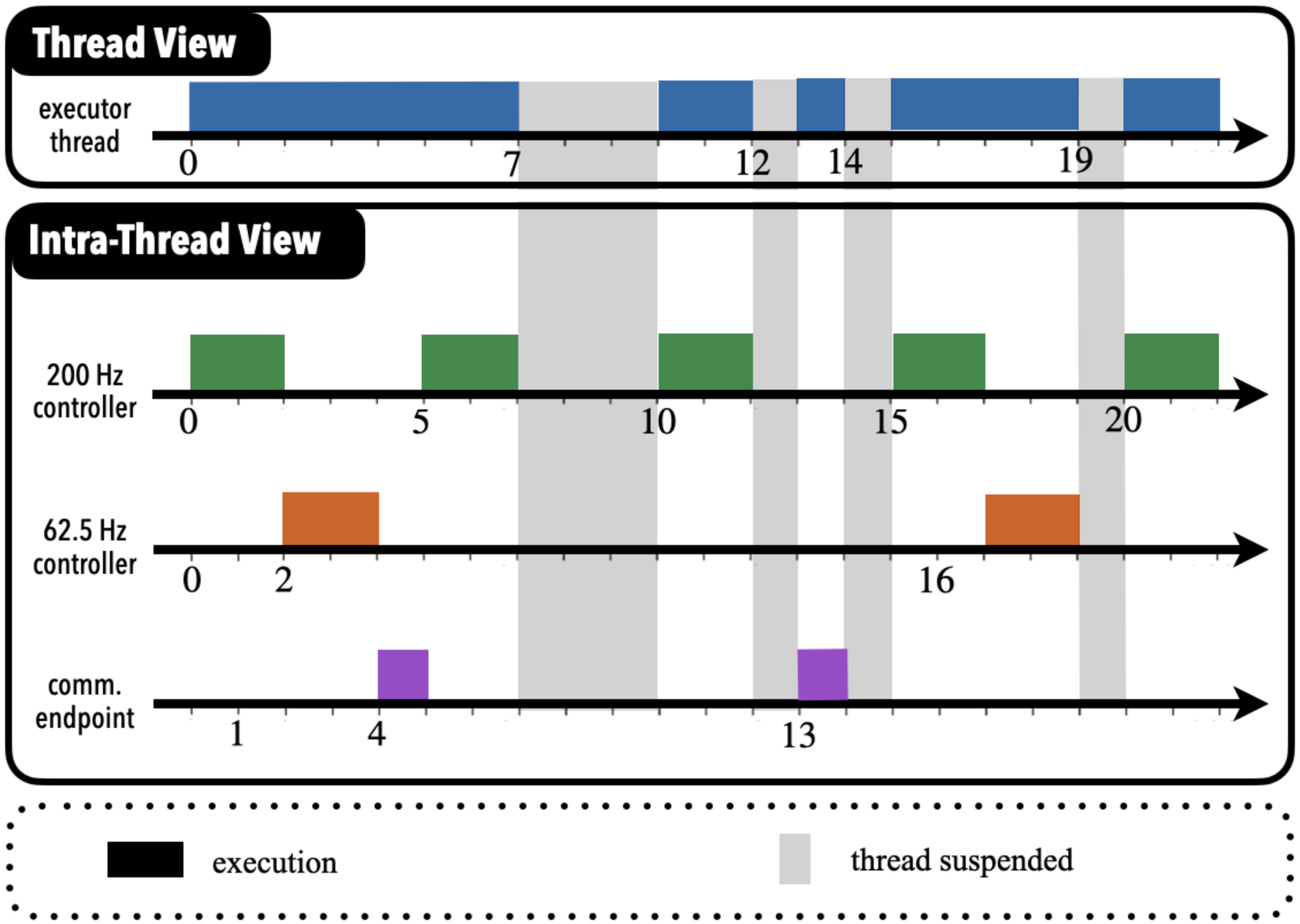
THREADS CONTAIN INTRA-THREAD TASKS

Modern systems use **frameworks** to map many **logical tasks** onto **fewer OS threads**



THREADS CONTAIN INTRA-THREAD TASKS

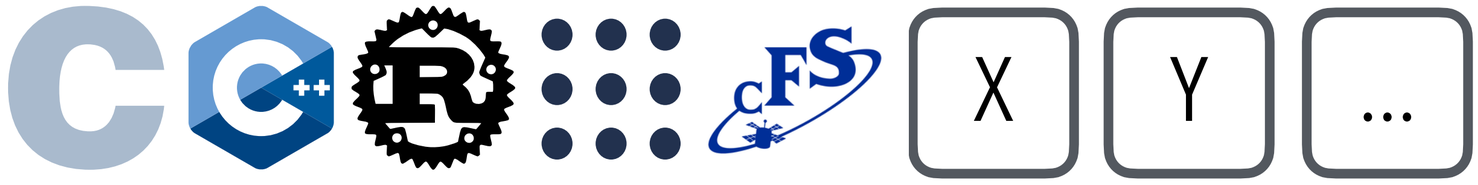
Modern systems use **frameworks** to map many **logical tasks** onto **fewer OS threads**



WHY FRAMEWORK-AGNOSTIC?

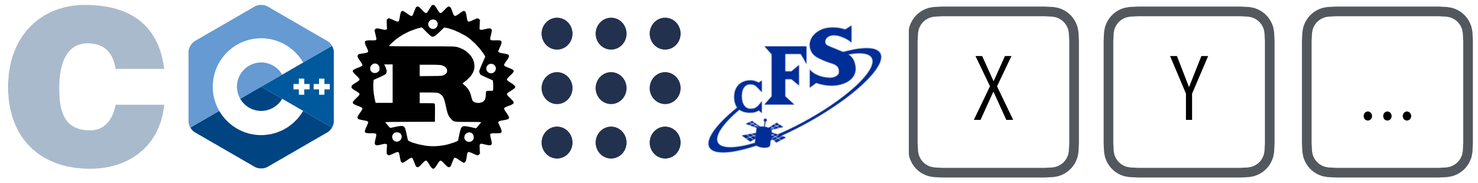
WHY FRAMEWORK-AGNOSTIC?

There are **too many** frameworks to support individually



WHY FRAMEWORK-AGNOSTIC?

There are **too many** frameworks to support individually



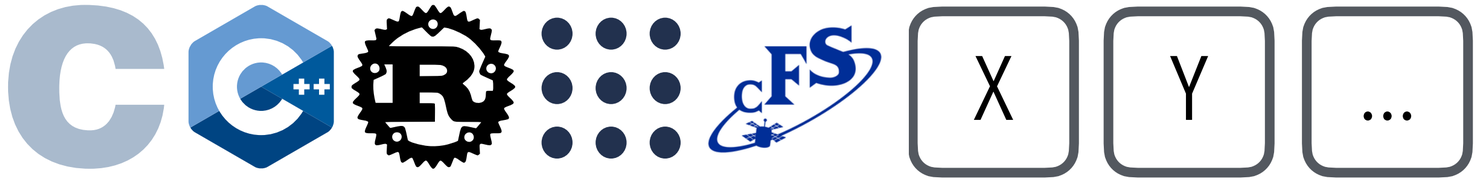
Frameworks **evolve and change** over time

ROS 2 Versions

Ardent Apalone (2017) | Bouncy Bolson (2018) | Eloquent Elusor (2019) | Humble Hawksbill (2022) | Jazzy Jalisco (2024)

WHY FRAMEWORK-AGNOSTIC?

There are **too many** frameworks to support individually

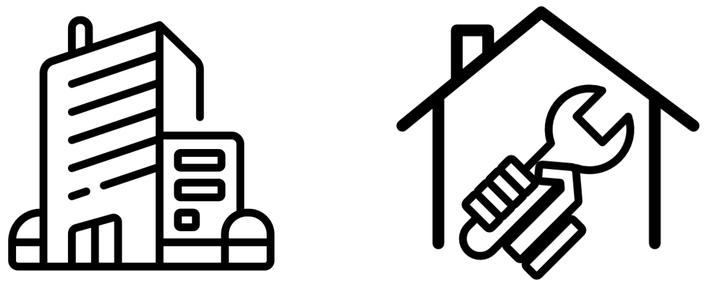


Frameworks **evolve and change** over time

ROS 2 Versions

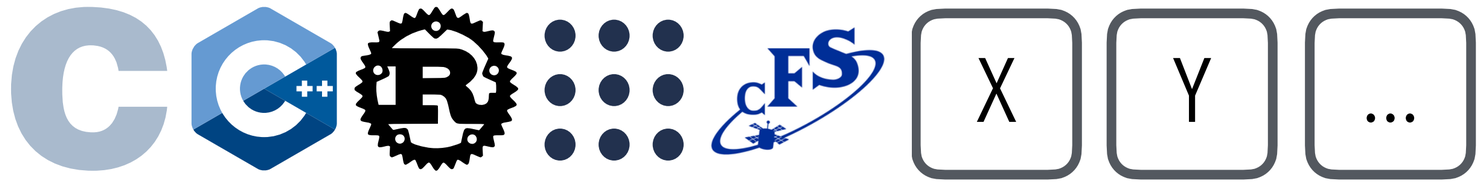
Ardent Apalone (2017) | Bouncy Bolson (2018) | Eloquent Elusor (2019) | Humble Hawksbill (2022) | Jazzy Jalisco (2024)

Companies implement **in-house closed-source** frameworks



WHY FRAMEWORK-AGNOSTIC?

There are **too many** frameworks to support individually



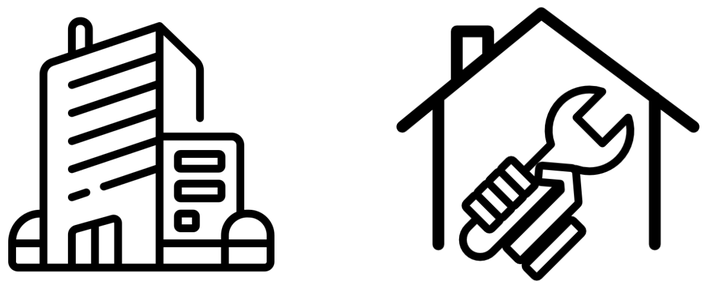
Fra

Framework-agnostic solution

⇒ **Wide** and **diverse** support "for free"

(2017) (2018) (2019) (2022) (2024)

Companies implement **in-house closed-source** frameworks

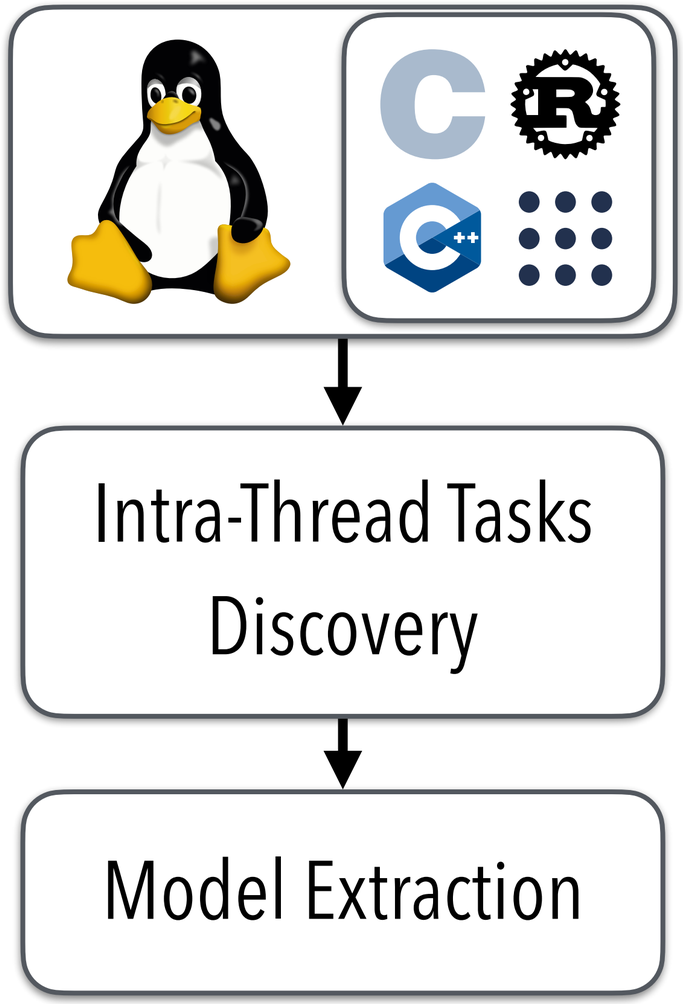


IN THIS PAPER

Framework-agnostic discovery of

+

Timing models extraction for intra-thread tasks



IN THIS PAPER

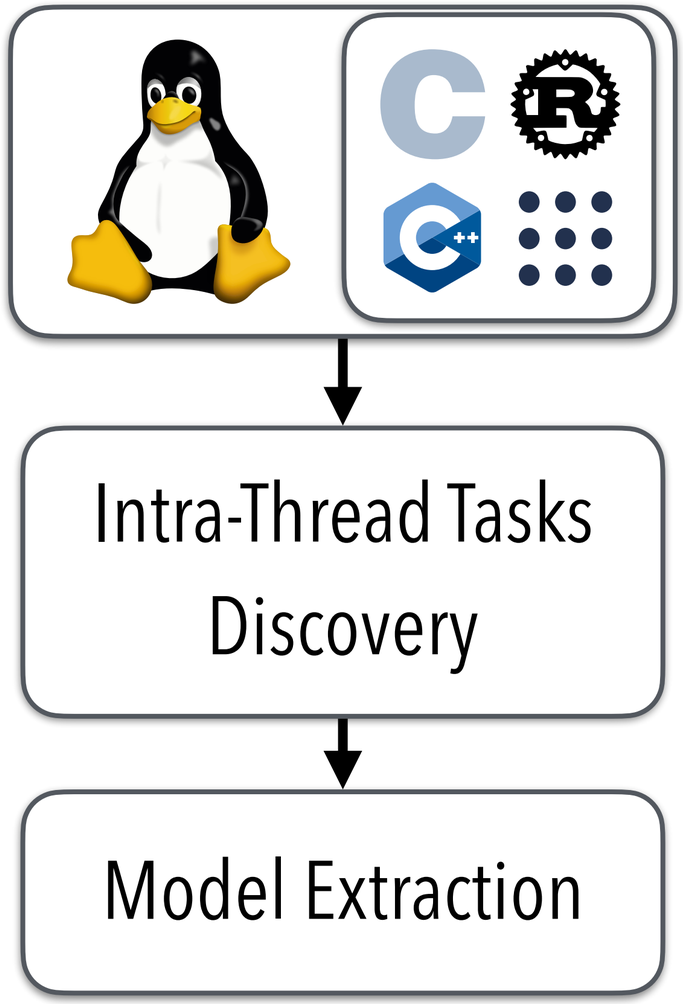
Framework-agnostic discovery of

+

Timing models extraction for

intra-thread tasks

Two fundamental challenges lie ahead



FRAMEWORK-AGNOSTIC APPROACH INTRODUCES CHALLENGES

FRAMEWORK-AGNOSTIC APPROACH INTRODUCES CHALLENGES

...

How to discover intra-thread tasks in a framework-agnostic setting?

FRAMEWORK-AGNOSTIC APPROACH INTRODUCES CHALLENGES

Intra-Thread View

Framework controls job release times
Unknown in a framework-agnostic setting

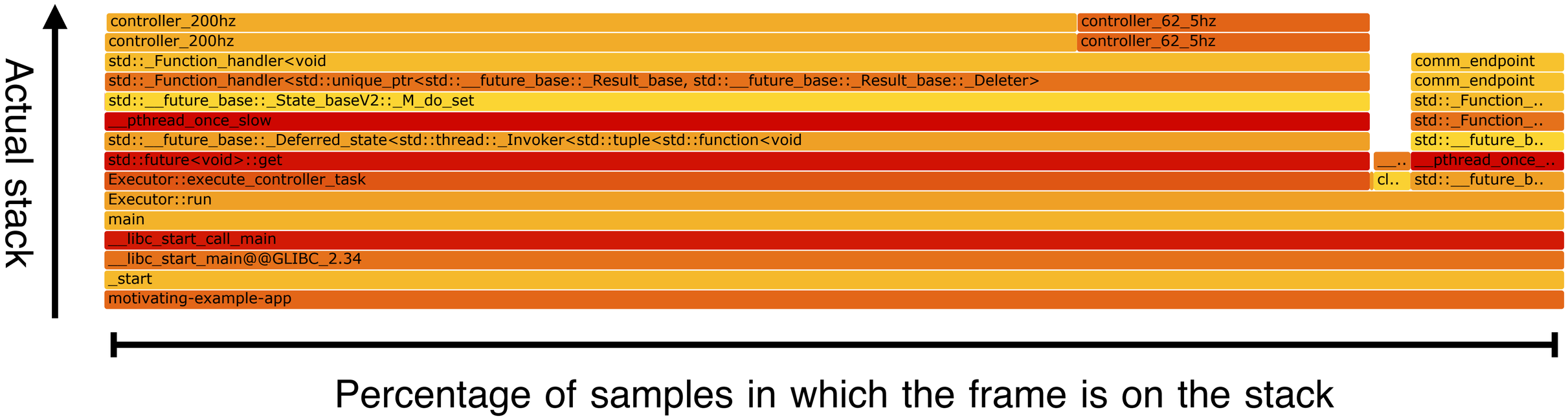
62.5 Hz
controller

How to discover intra-thread tasks in a framework-agnostic setting?

How to infer timing models without observing the exact release times?

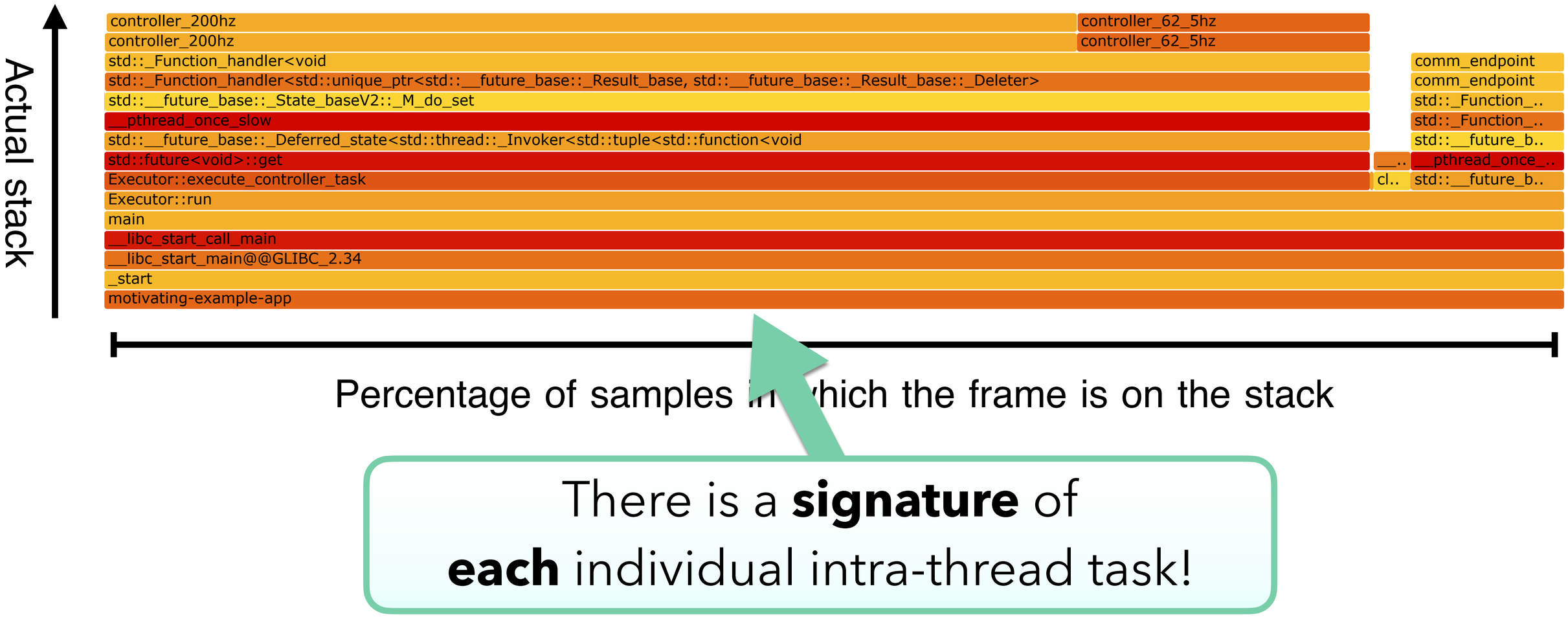
SOLUTION 1: DISCOVER INTRA-THREAD TASKS ON THE STACK

We can discover intra-thread tasks from stack profiles by exploiting the signatures they leave on the stack!



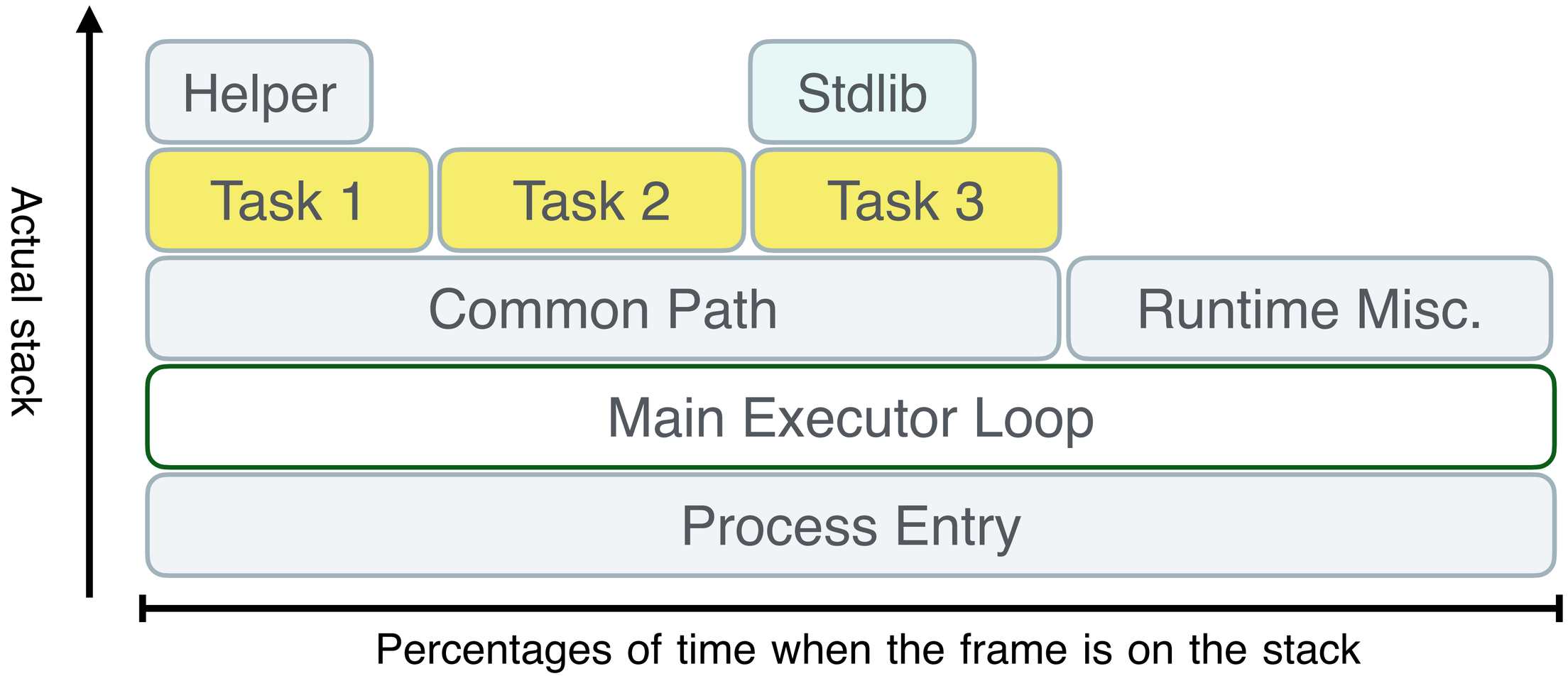
SOLUTION 1: DISCOVER INTRA-THREAD TASKS ON THE STACK

We can discover intra-thread tasks from stack profiles by exploiting the signatures they leave on the stack!

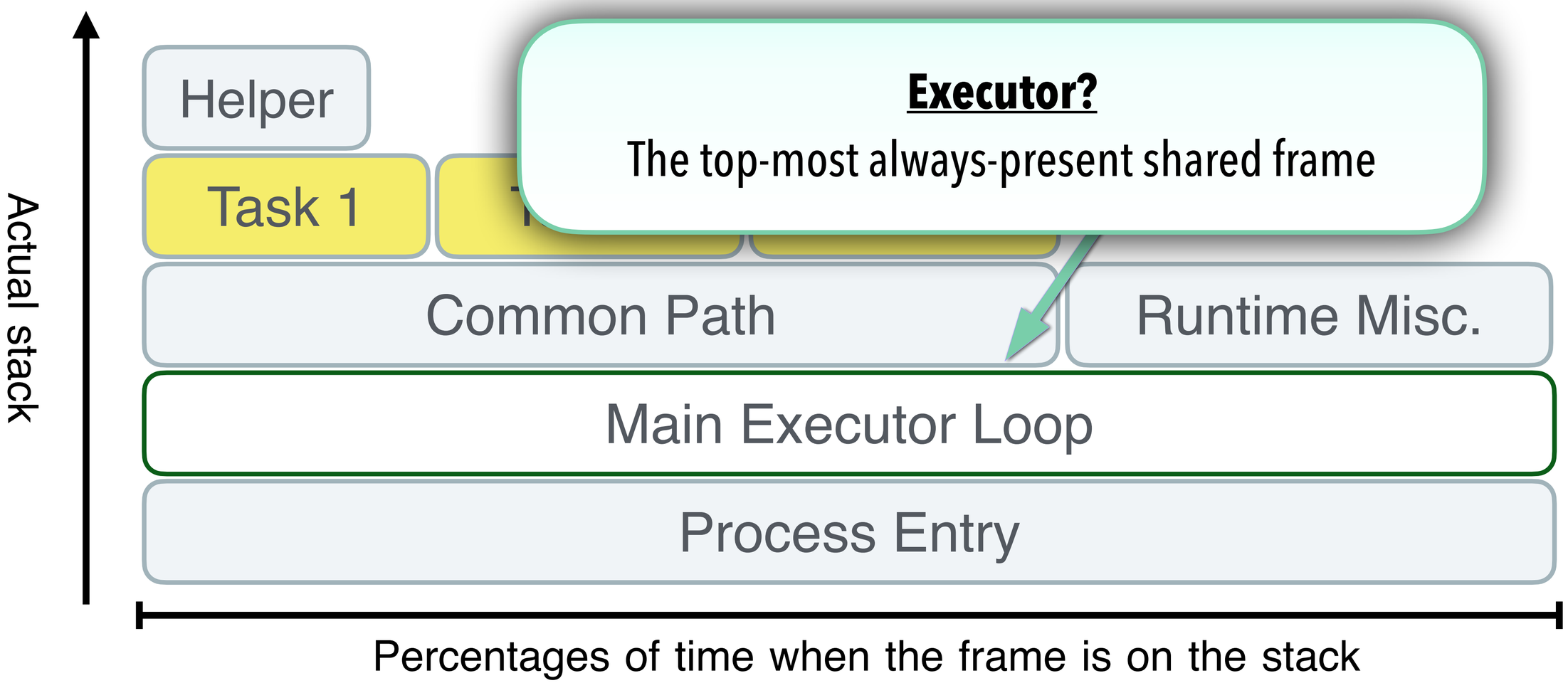


There is a **signature** of **each** individual intra-thread task!

SPOTTING SIGNATURES ON THE STACK



SPOTTING SIGNATURES ON THE STACK

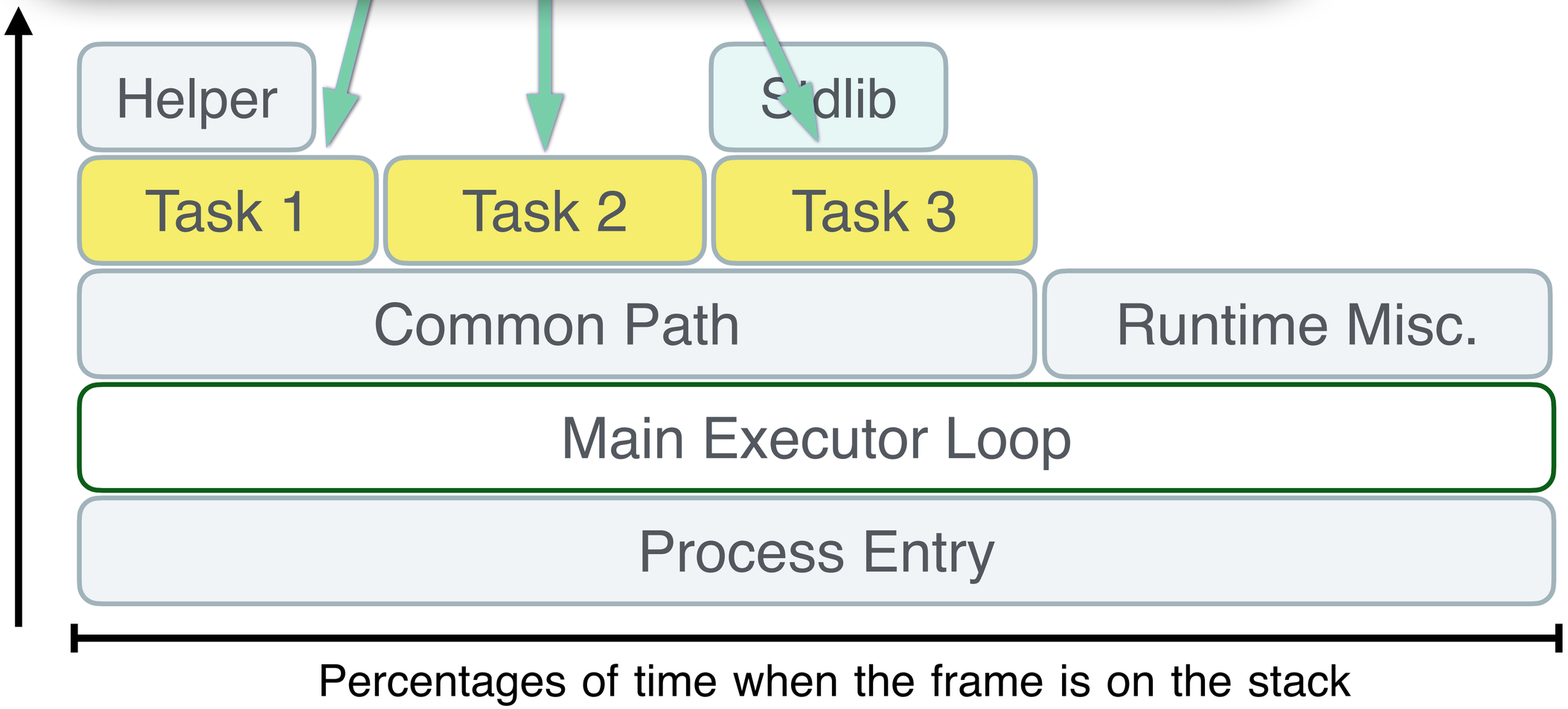


SPOTTING SIGNATURES ON THE STACK

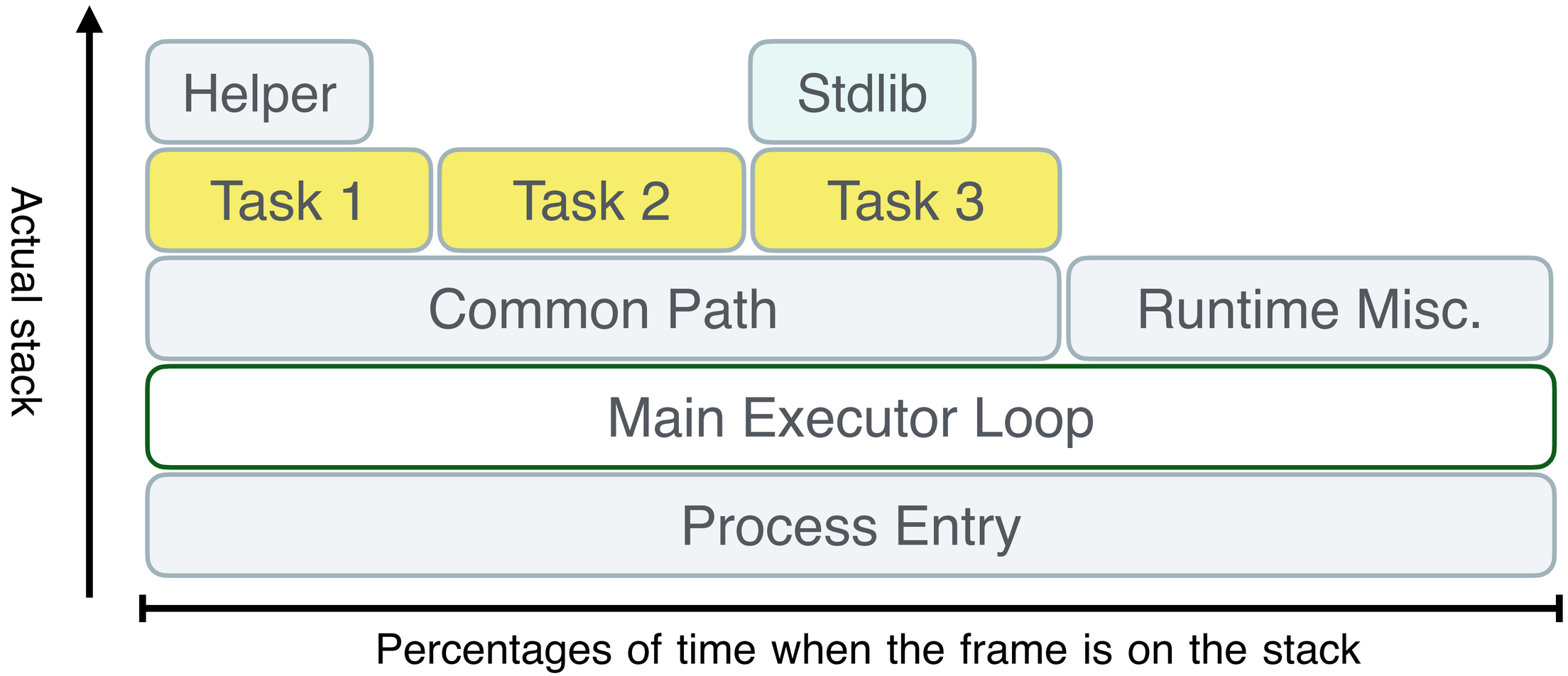
Actual intra-thread tasks?

5 heuristics to filter out non-relevant frames and discover the actual tasks

(Details in the paper)

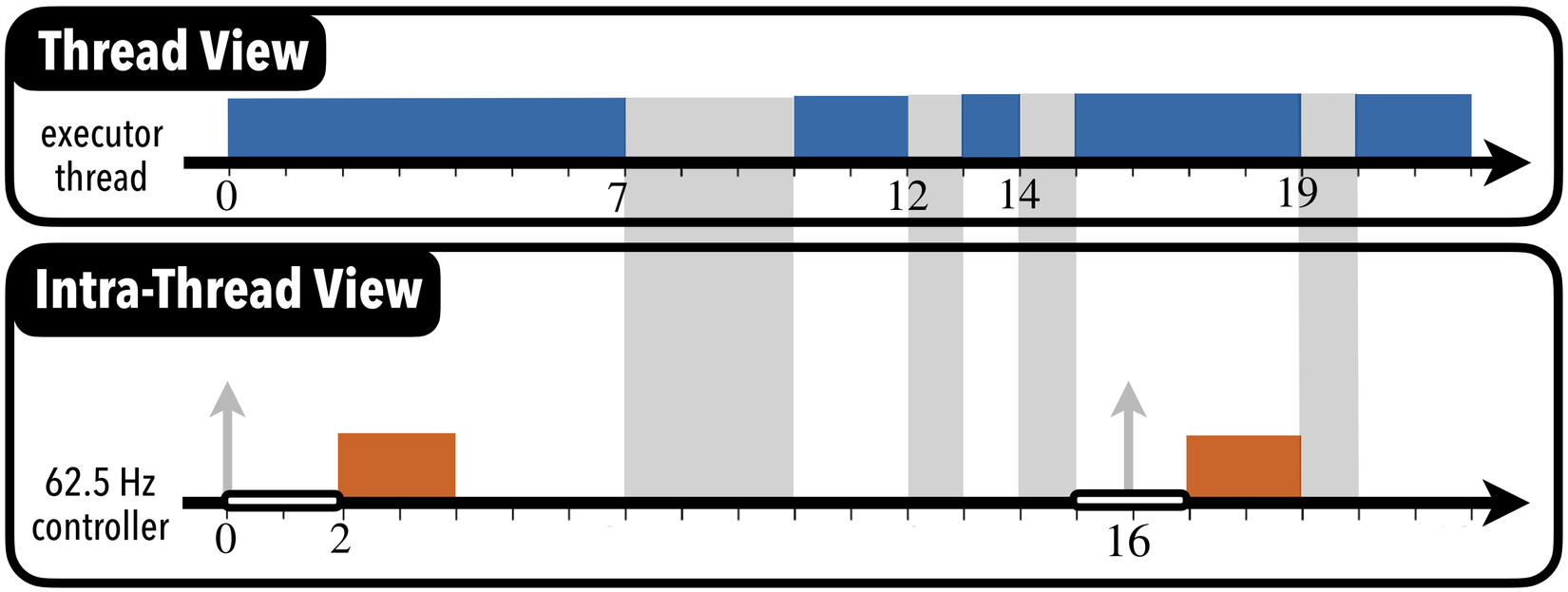


SPOTTING SIGNATURES ON THE STACK



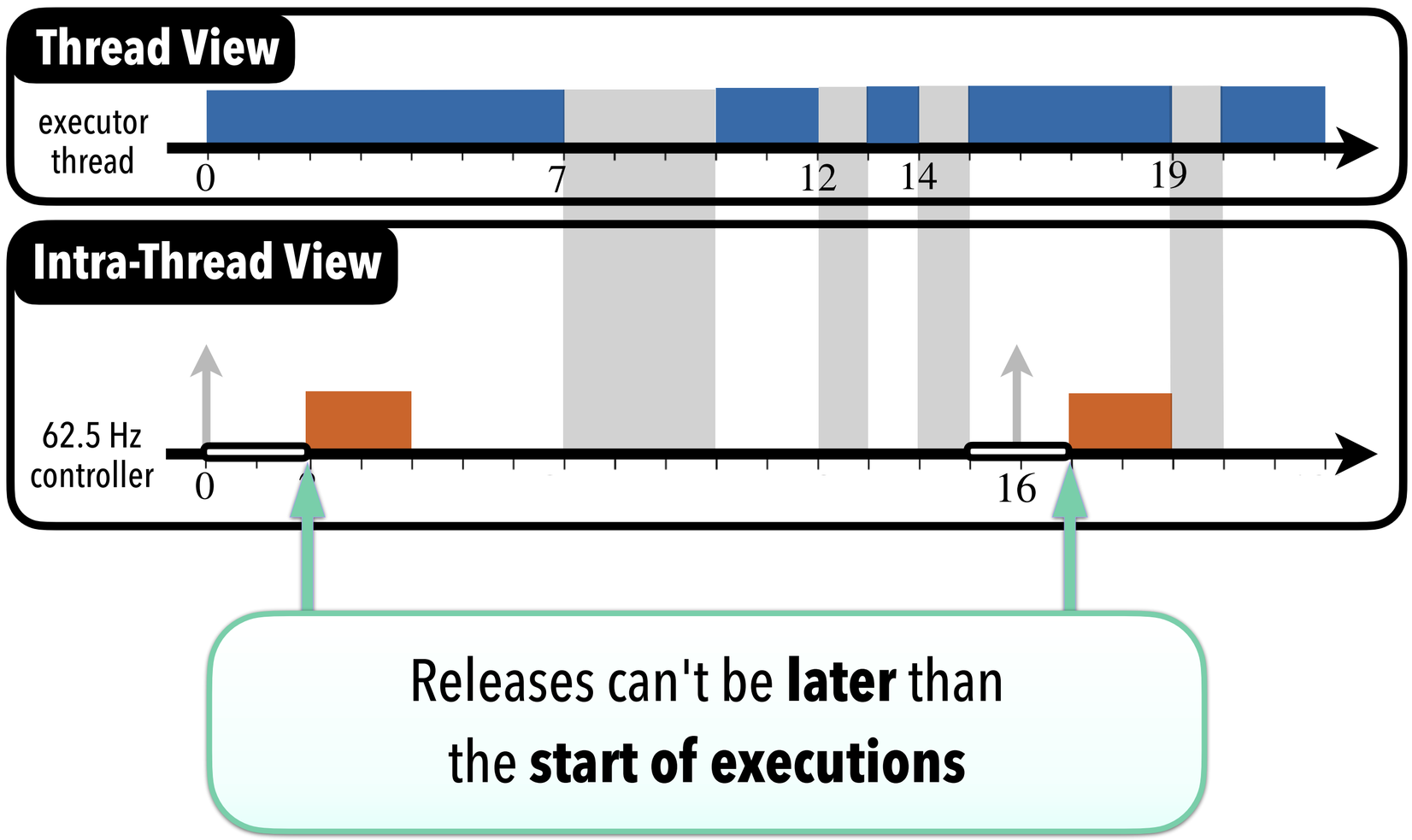
SOLUTION 2: BOUND RELEASE TIME WITHIN WINDOW

Exact release times are **uncertain**;
however, **release windows bound** them.



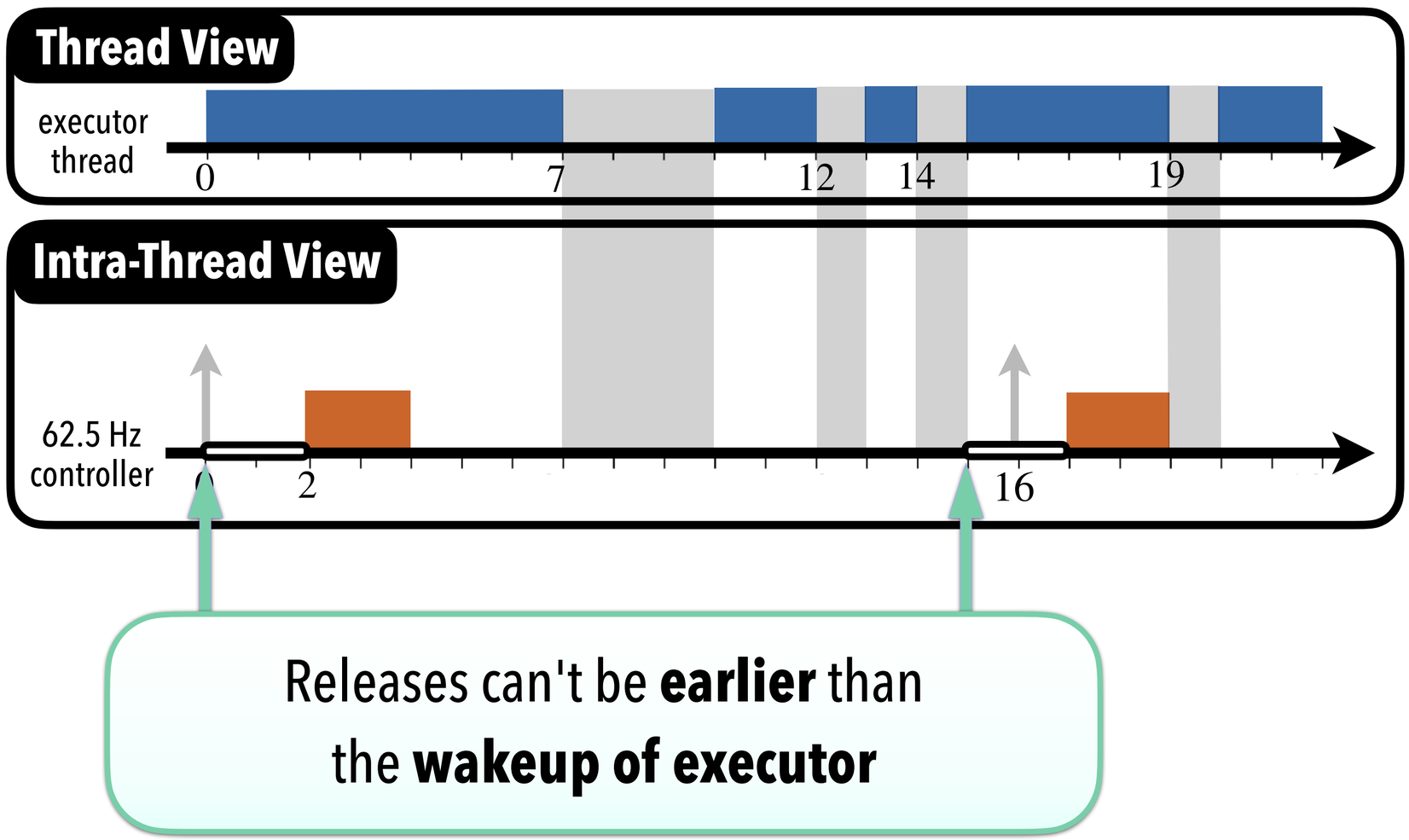
SOLUTION 2: BOUND RELEASE TIME WITHIN WINDOW

Exact release times are **uncertain**;
however, **release windows bound** them.



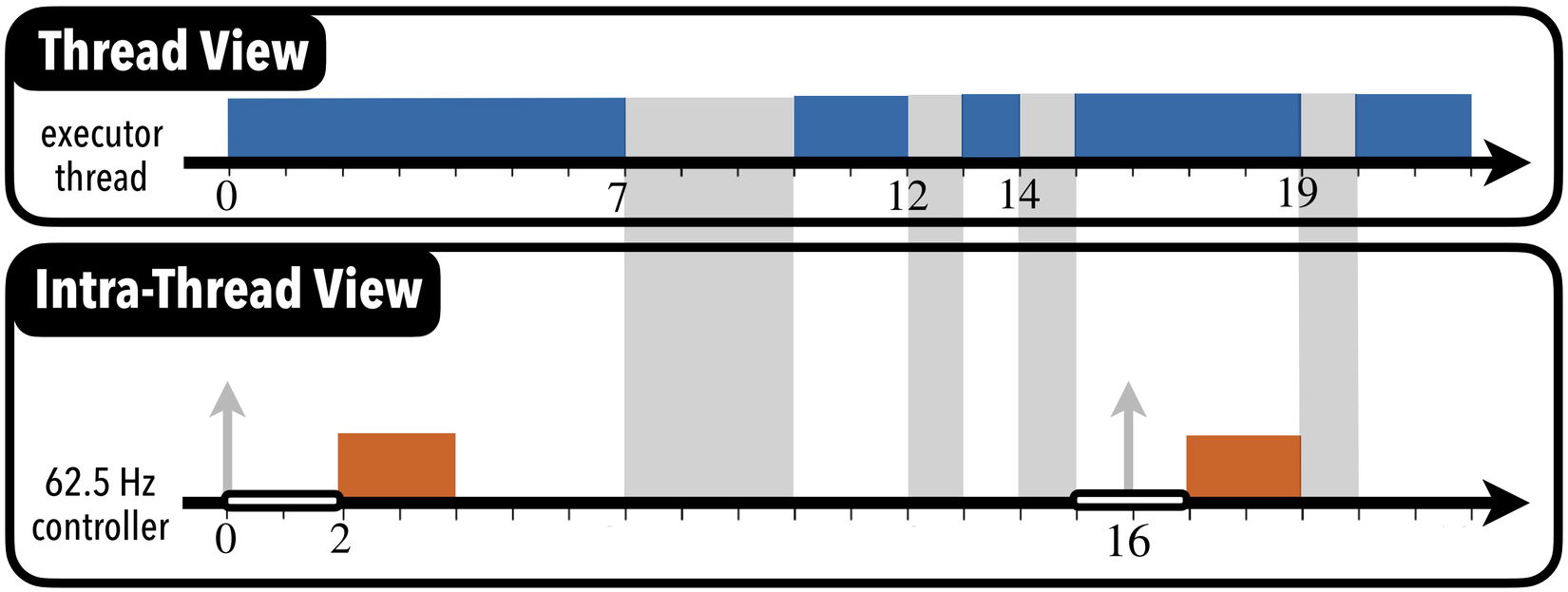
SOLUTION 2: BOUND RELEASE TIME WITHIN WINDOW

Exact release times are **uncertain**;
however, **release windows bound** them.



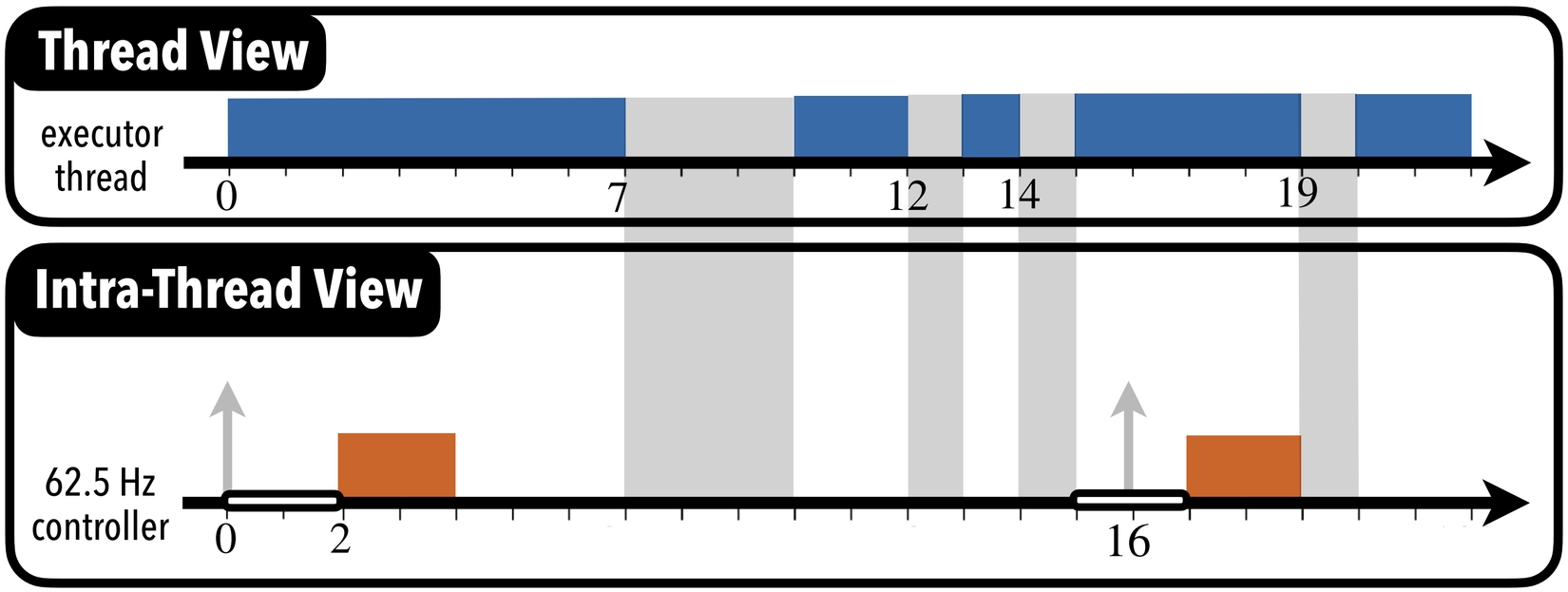
SOLUTION 2: BOUND RELEASE TIME WITHIN WINDOW

Exact release times are **uncertain**;
however, **release windows bound** them.



SOLUTION 2: BOUND RELEASE TIME WITHIN WINDOW

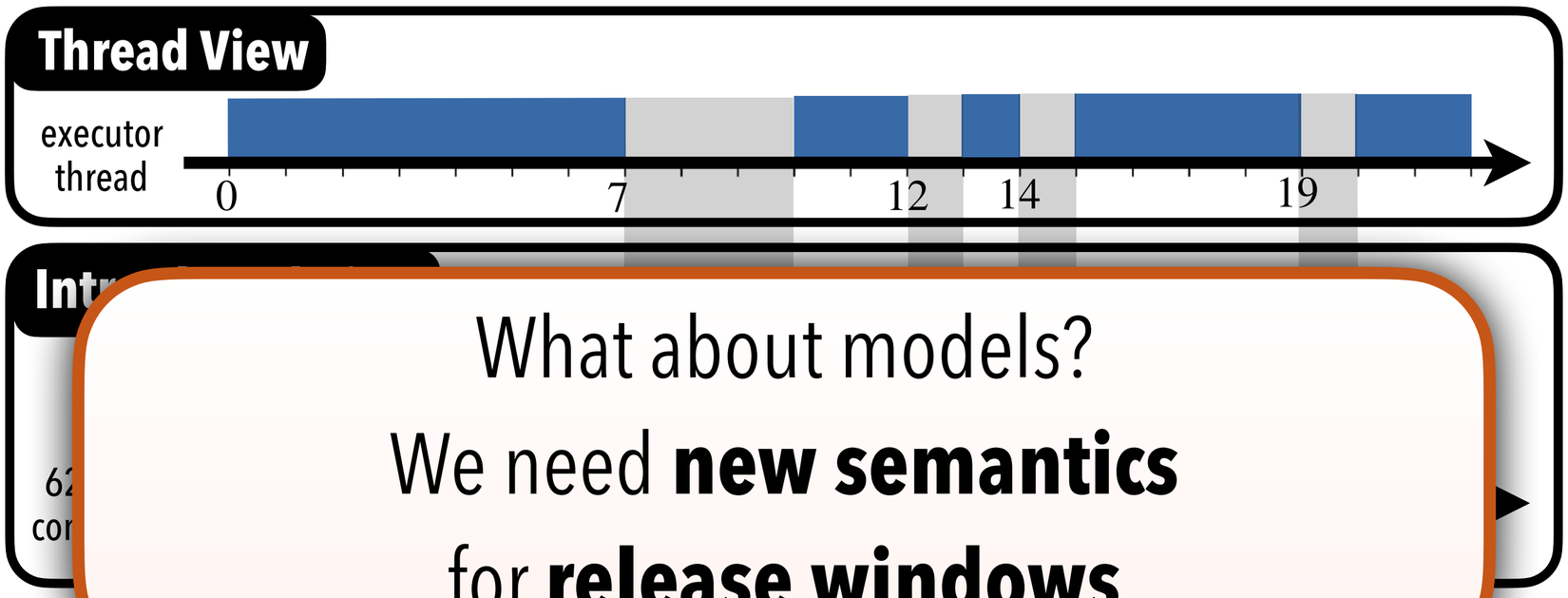
Exact release times are **uncertain**;
however, **release windows bound** them.



Define $R_i = [r_i^-, r_i^+]$ to
bound the uncertain exact release time
Where r_i^- is the **last executor wakeup time**
and r_i^+ is the **start of execution**

SOLUTION 2: BOUND RELEASE TIME WITHIN WINDOW

Exact release times are **uncertain**;
however, **release windows bound** them.



Define $R_i = [r_i^-, r_i^+]$ to
bound the uncertain exact release time
Where r_i^- is the **last executor wakeup time**
and r_i^+ is the **start of execution**

CLASSIC PERIODIC MODEL: POINT RELEASES

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

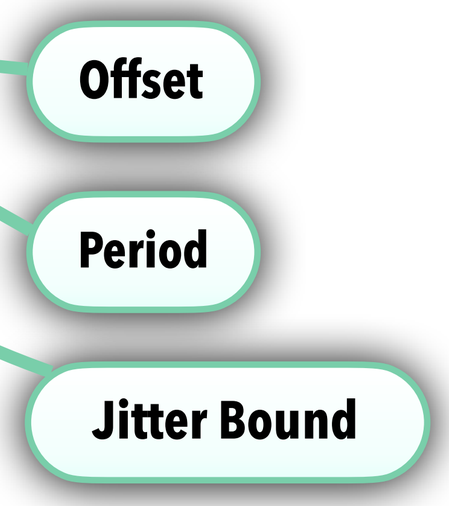
Releases: r_i

CLASSIC PERIODIC MODEL: POINT RELEASES

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

Releases: r_i

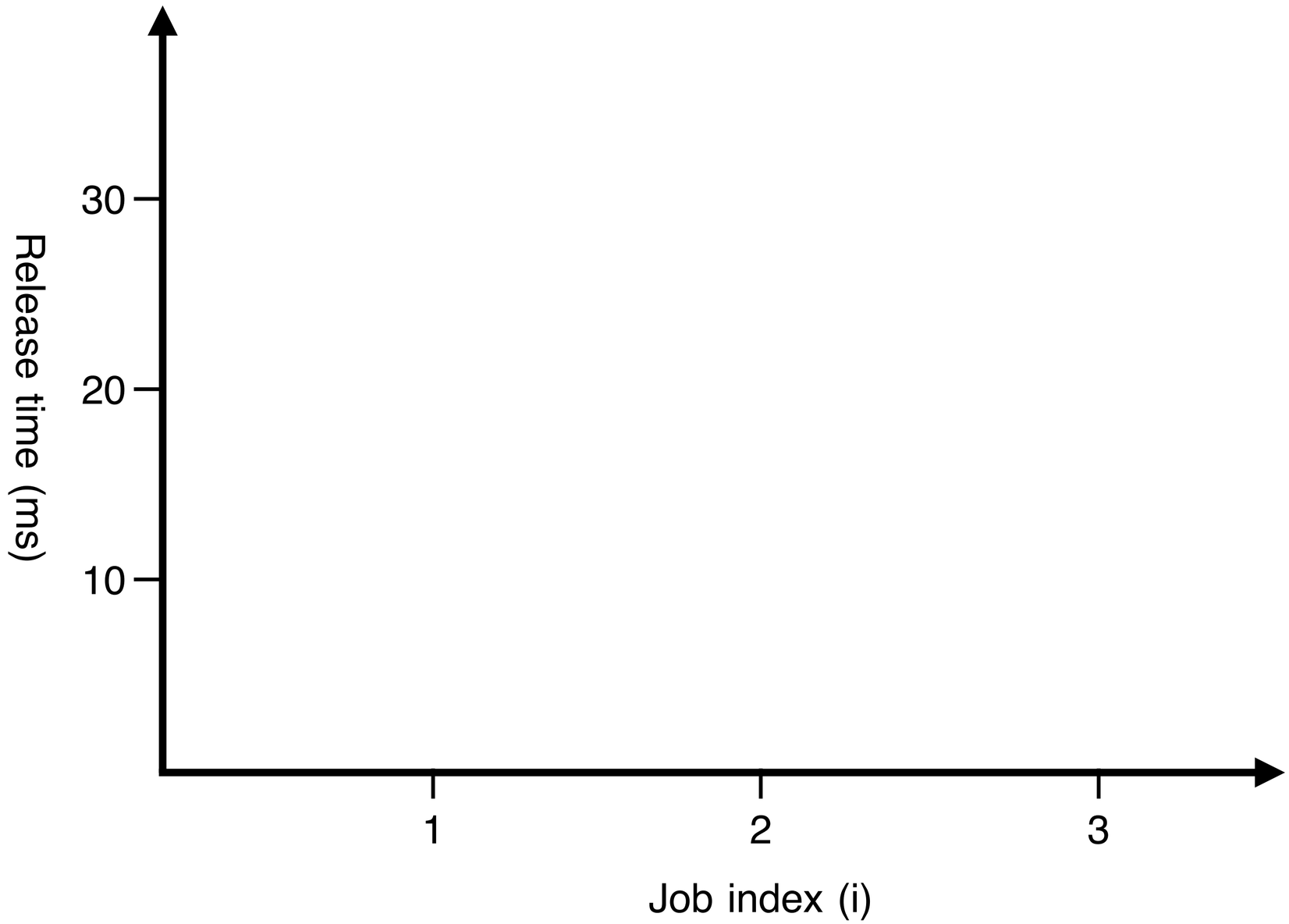


CLASSIC PERIODIC MODEL: POINT RELEASES

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

Releases: r_i

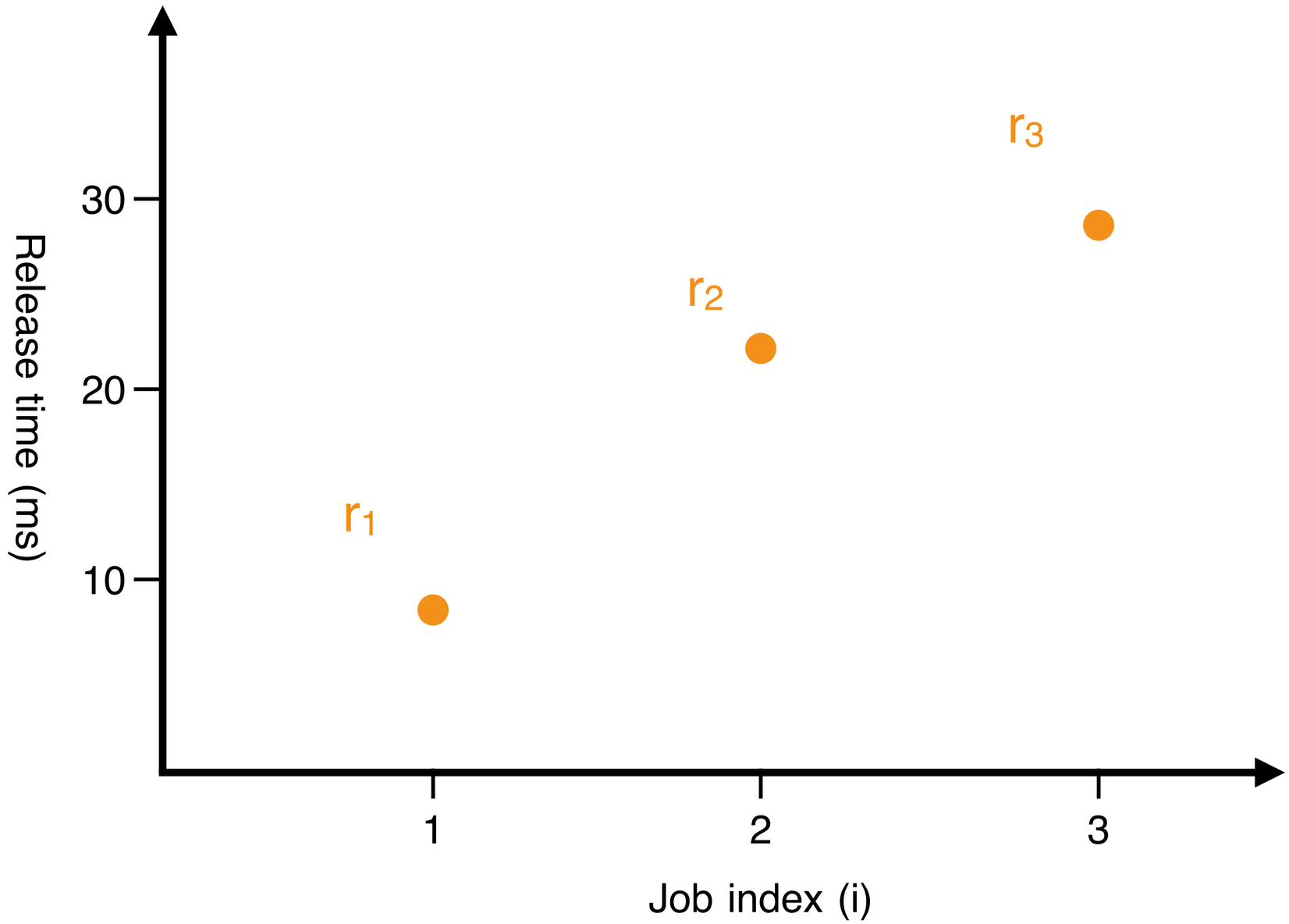


CLASSIC PERIODIC MODEL: POINT RELEASES

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

Releases: r_i

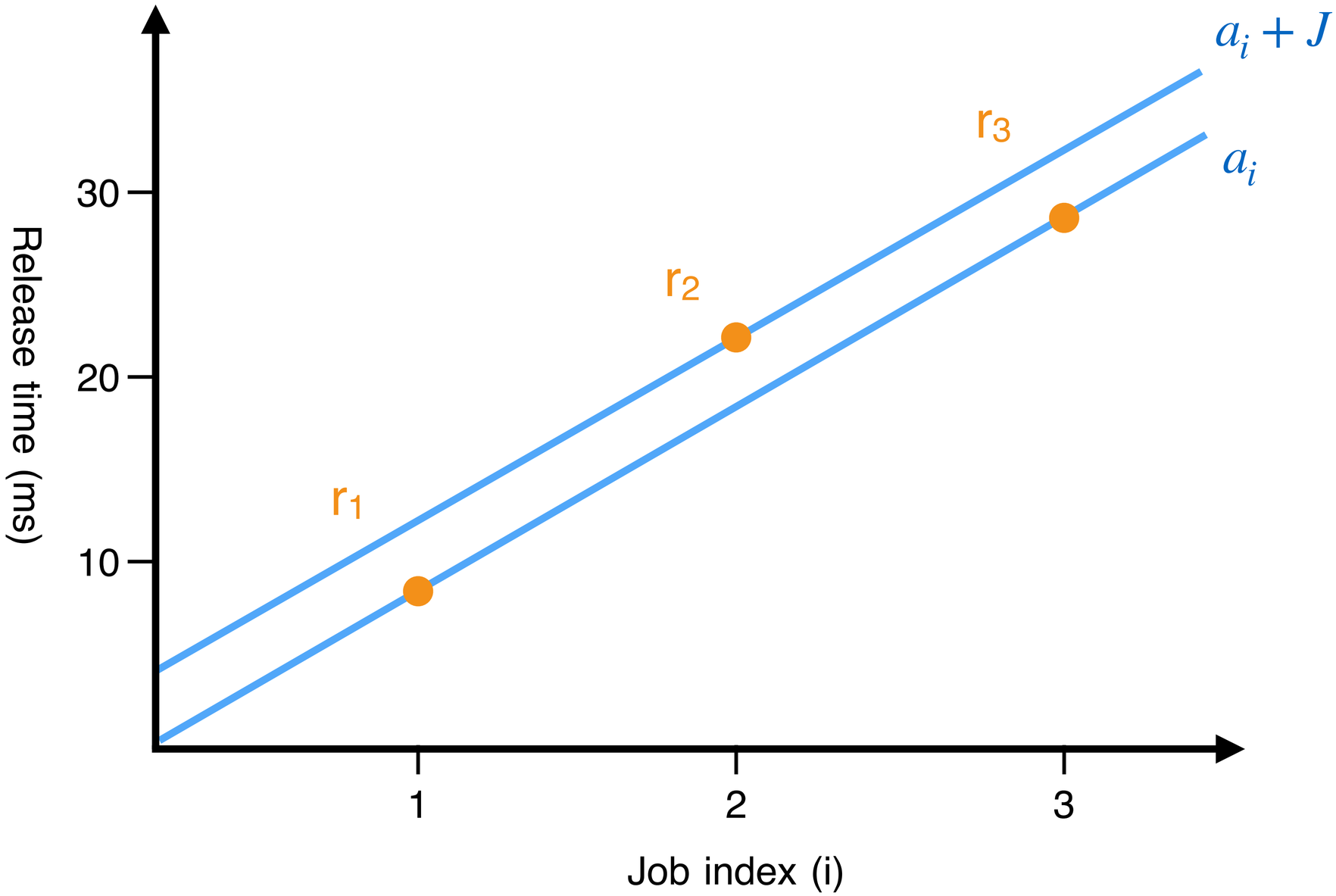


CLASSIC PERIODIC MODEL: POINT RELEASES

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

Releases: r_i

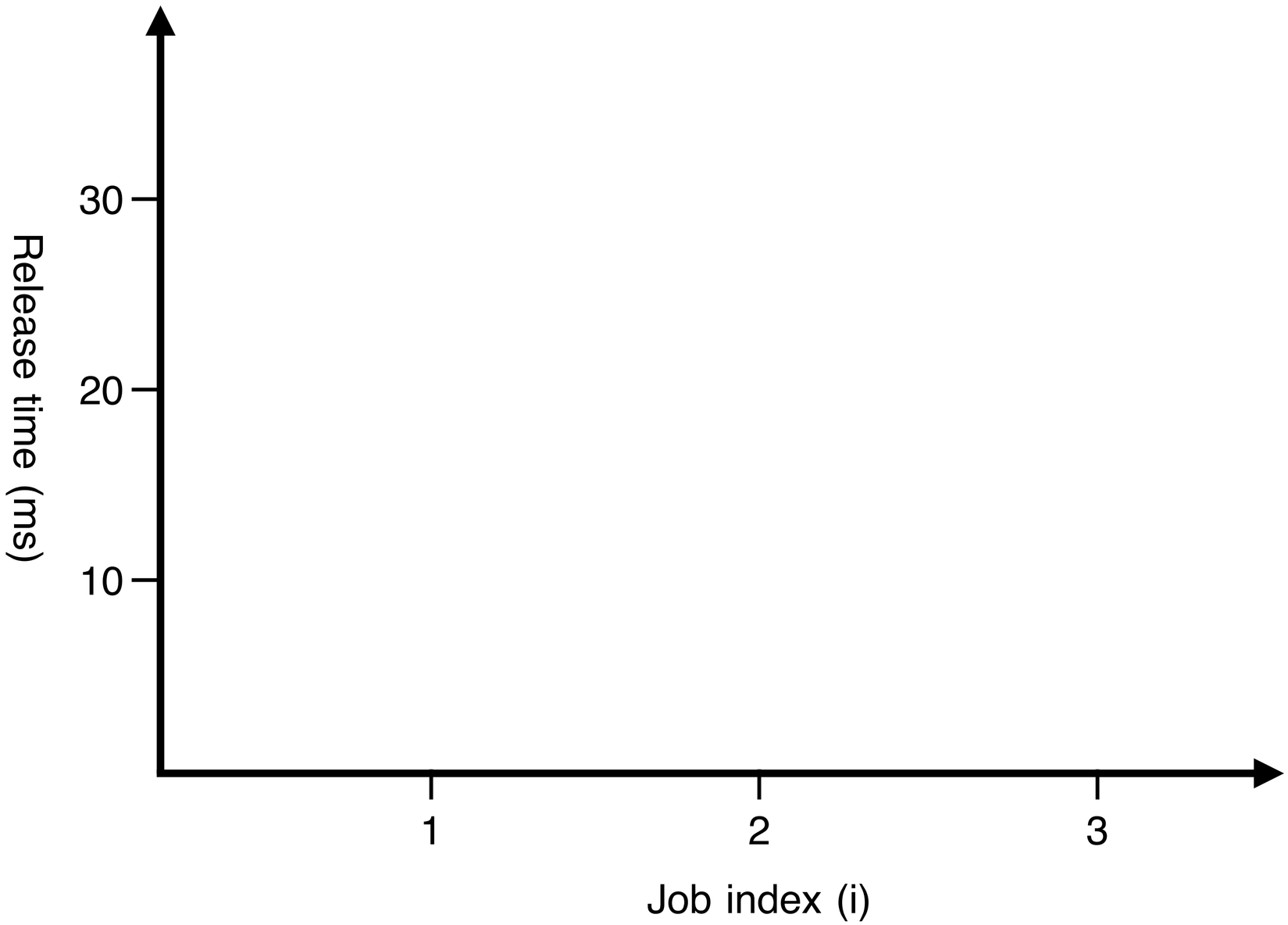


THIS PAPER: POSSIBLE-FIT PERIODIC MODEL

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

Release windows: R_i

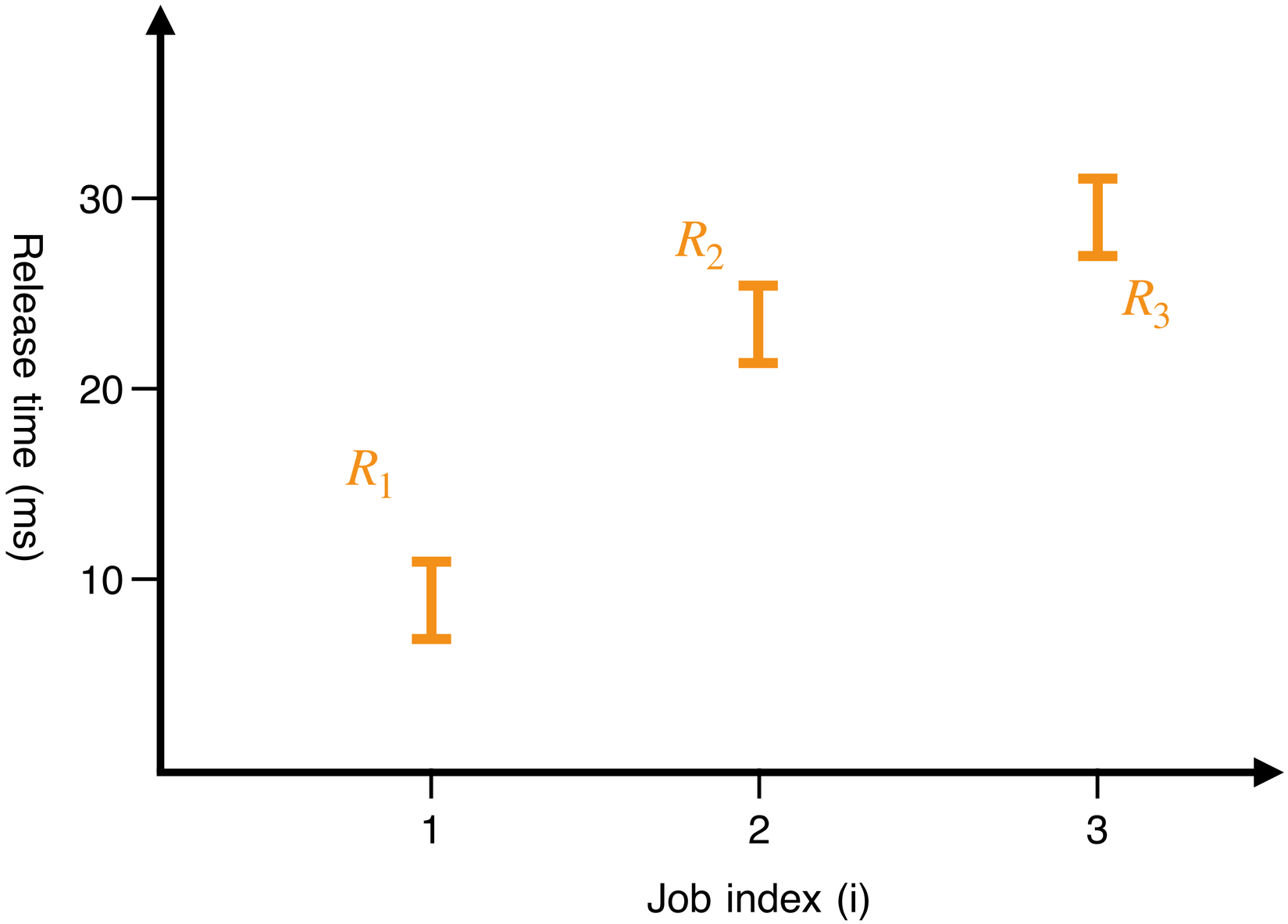


THIS PAPER: POSSIBLE-FIT PERIODIC MODEL

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

Release windows: R_i

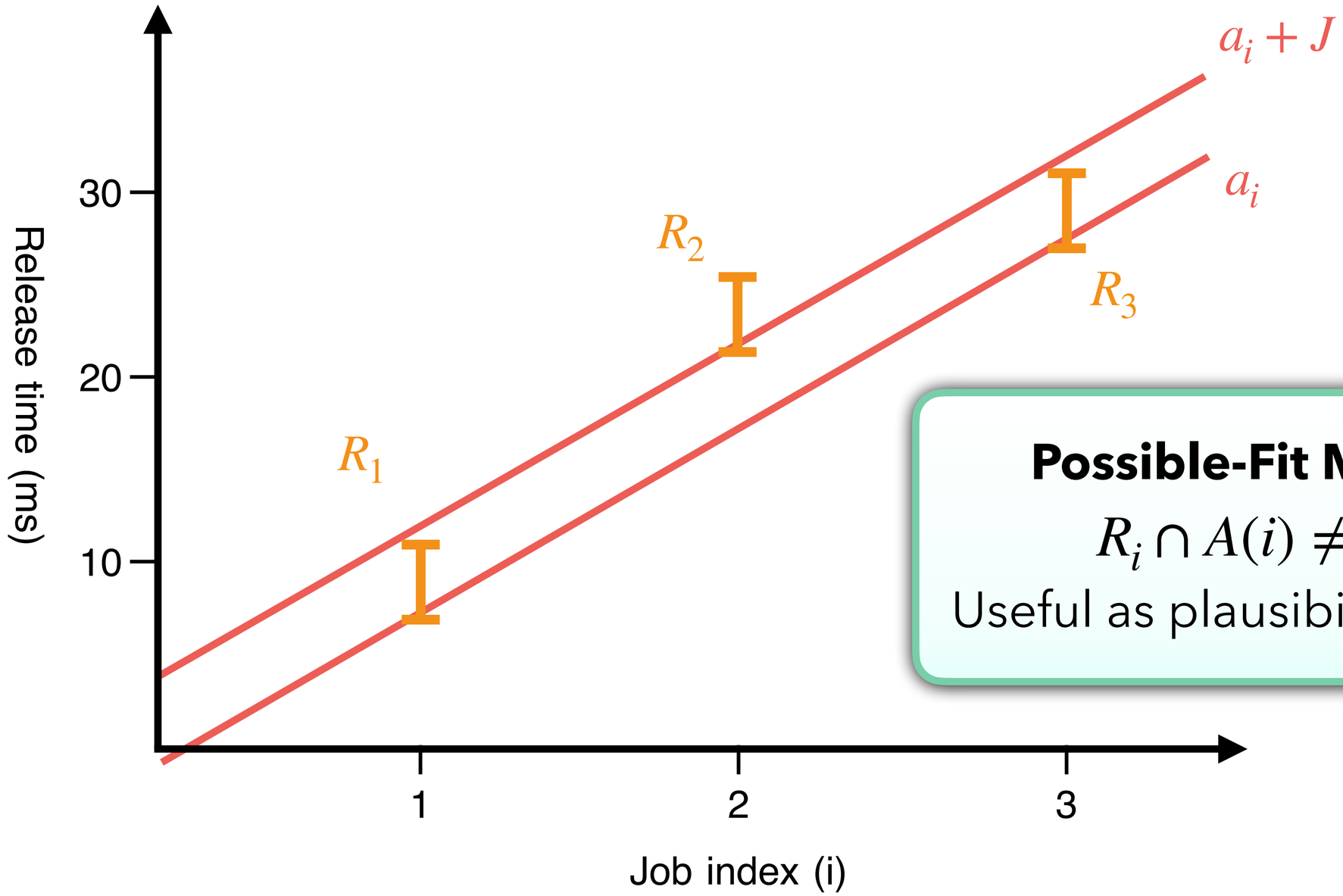


THIS PAPER: POSSIBLE-FIT PERIODIC MODEL

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

Release windows: R_i



Possible-Fit Model

$R_i \cap A(i) \neq \emptyset$

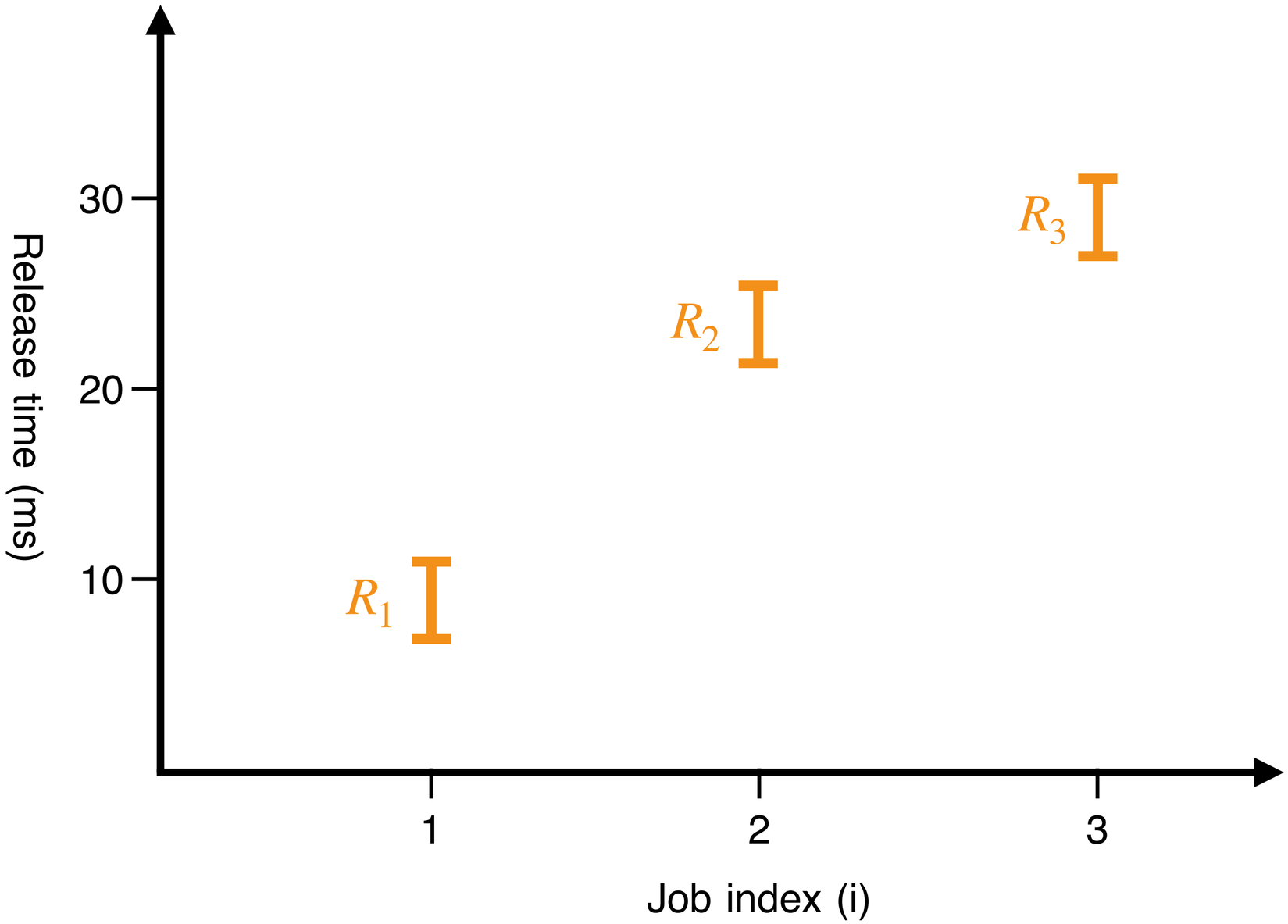
Useful as plausibility check

THIS PAPER: CERTAIN-FIT PERIODIC MODEL

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

Release windows: R_i

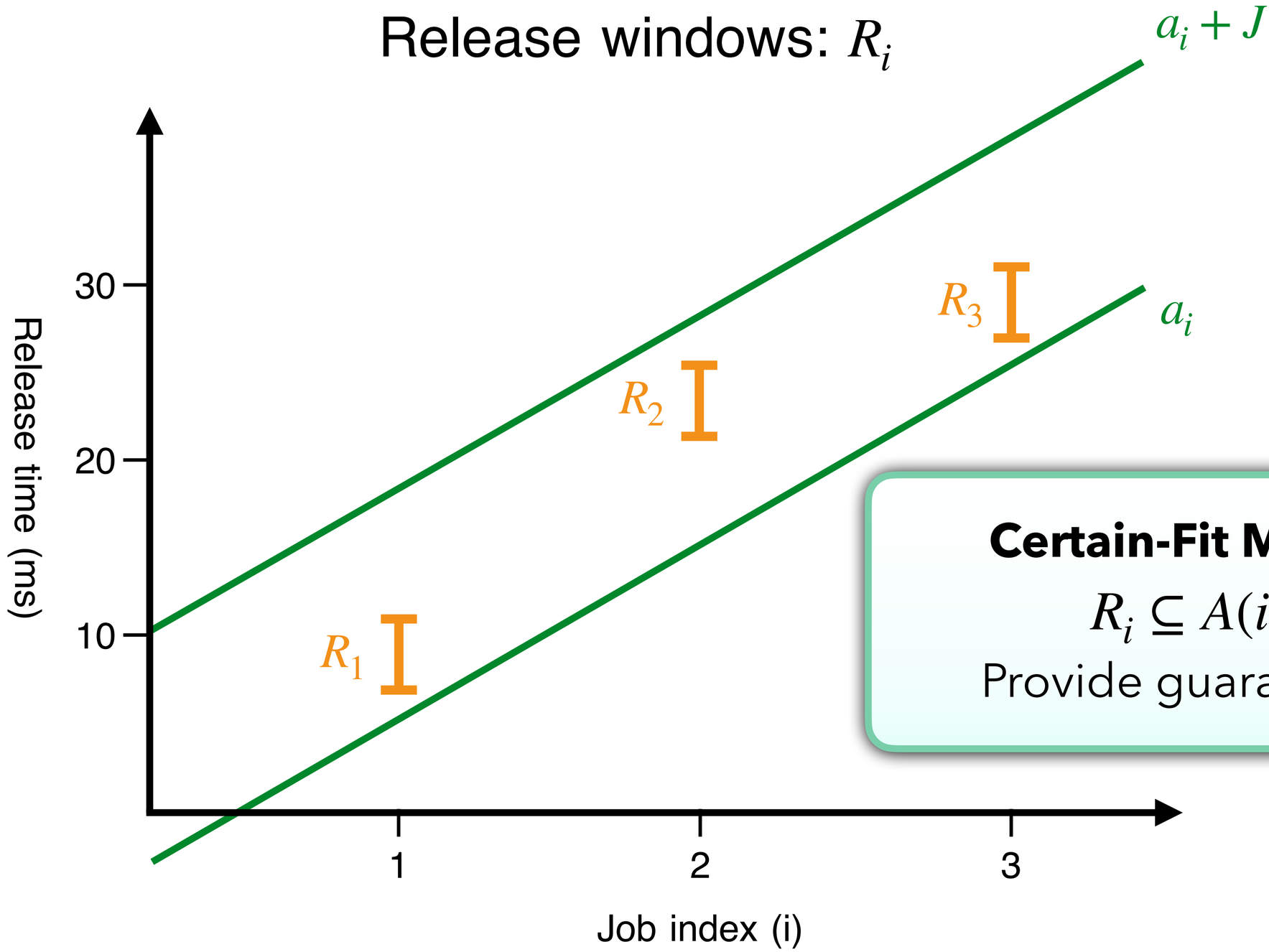


THIS PAPER: CERTAIN-FIT PERIODIC MODEL

Arrival time: $a_i = \Phi + (i-1) \times T$

Arrival window: $A(i) \triangleq [a_i, a_i + J]$

Release windows: R_i



Certain-Fit Model

$R_i \subseteq A(i)$

Provide guarantees

CLASSIC ARRIVAL CURVES: POINT RELEASES

Point release arrival curves: $a^{-}(\Delta) \leq |\{i \mid r_i \in I_t^{\Delta}\}| \leq a^{+}(\Delta)$

CLASSIC ARRIVAL CURVES: POINT RELEASES

Point release arrival curves: $a^{-}(\Delta) \leq |\{i \mid r_i \in I_t^{\Delta}\}| \leq a^{+}(\Delta)$

Lower bound on arrivals

Any **time interval** with length Δ

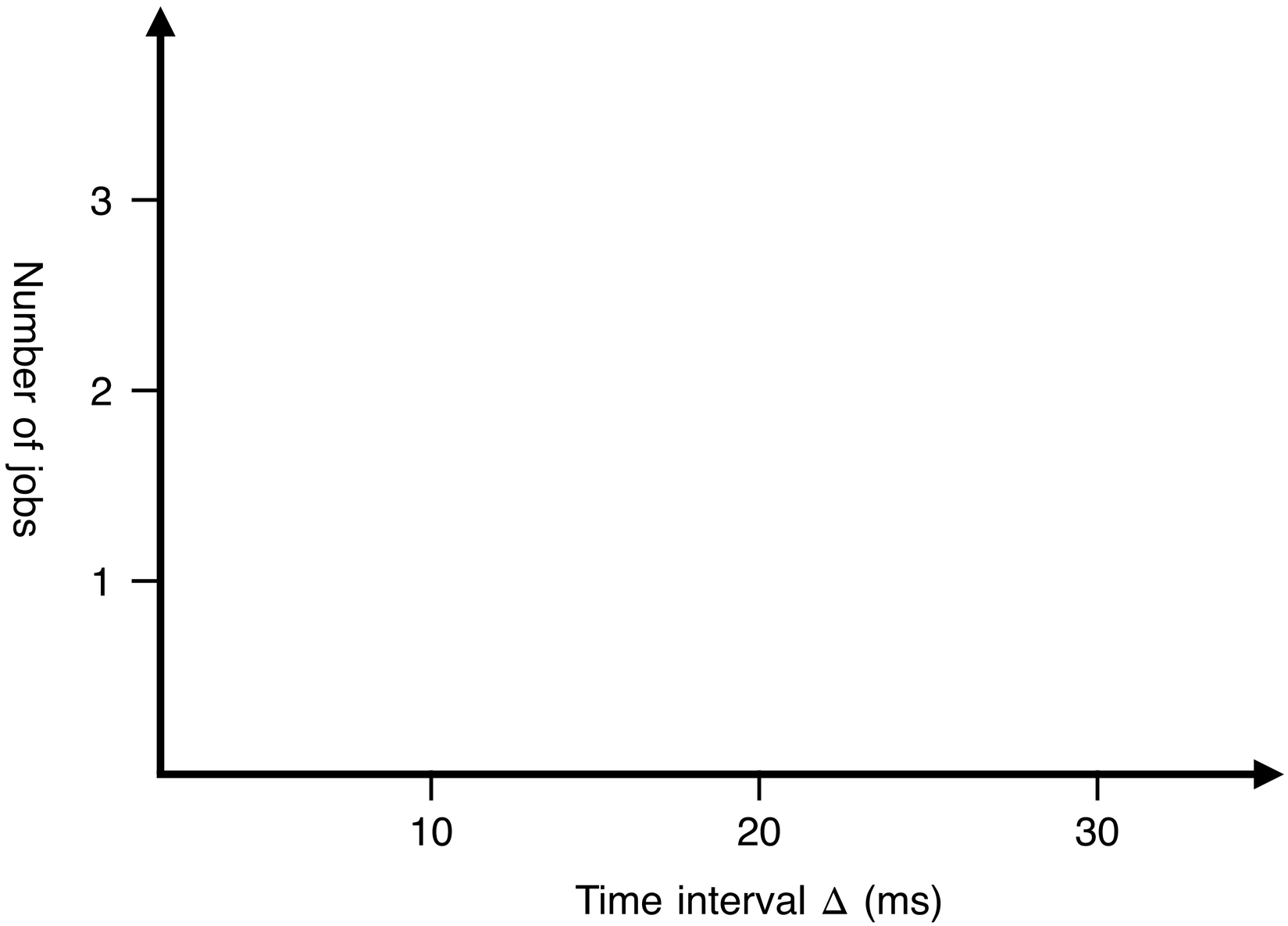
Upper bound on arrivals

CLASSIC ARRIVAL CURVES: POINT RELEASES

Point release arrival curves: $a^{-}(\Delta) \leq |\{i \mid r_i \in I_t^{\Delta}\}| \leq a^{+}(\Delta)$

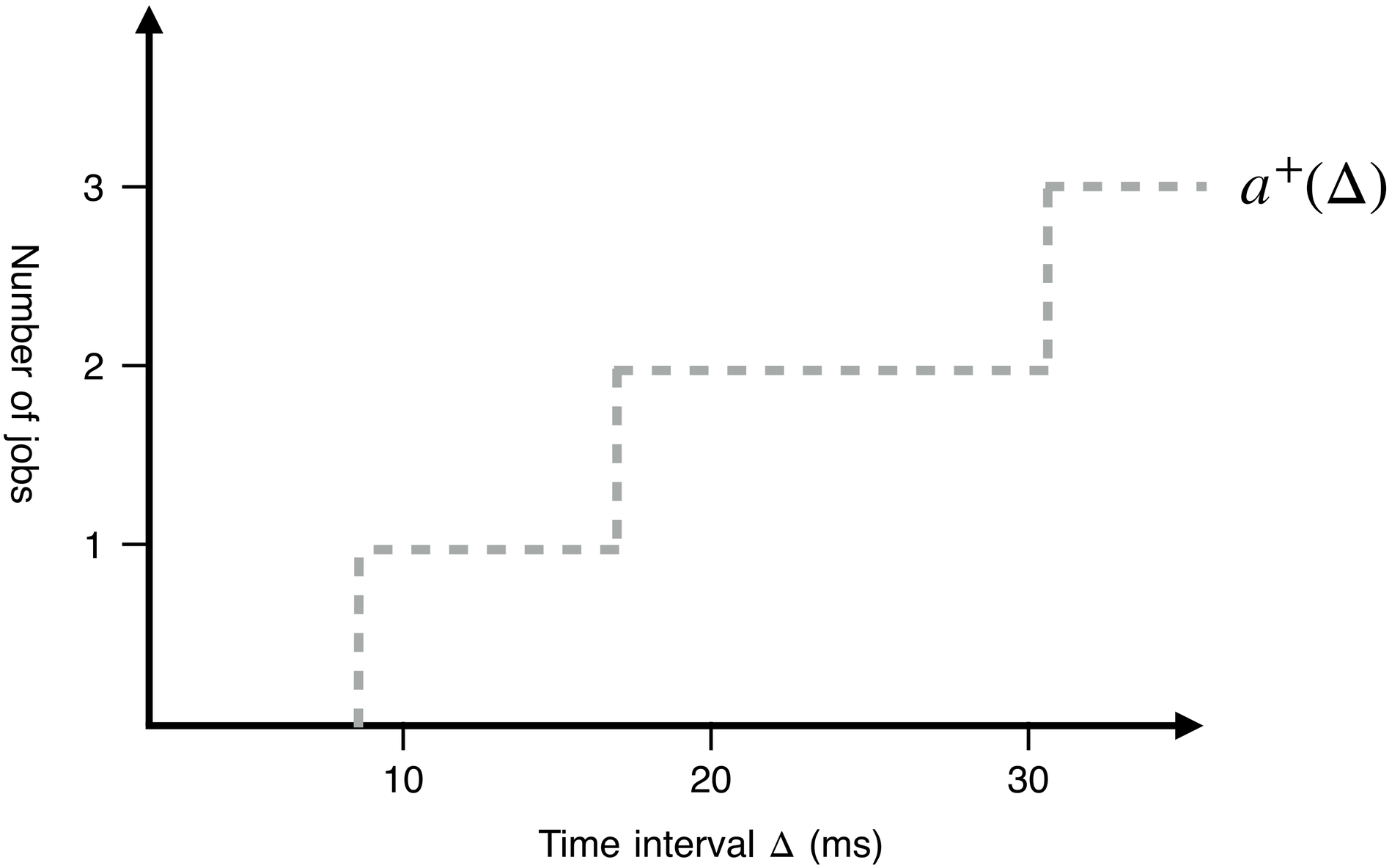
CLASSIC ARRIVAL CURVES: POINT RELEASES

Point release arrival curves: $a^{-}(\Delta) \leq |\{i \,|\, r_i \in I_t^{\Delta}\}| \leq a^{+}(\Delta)$



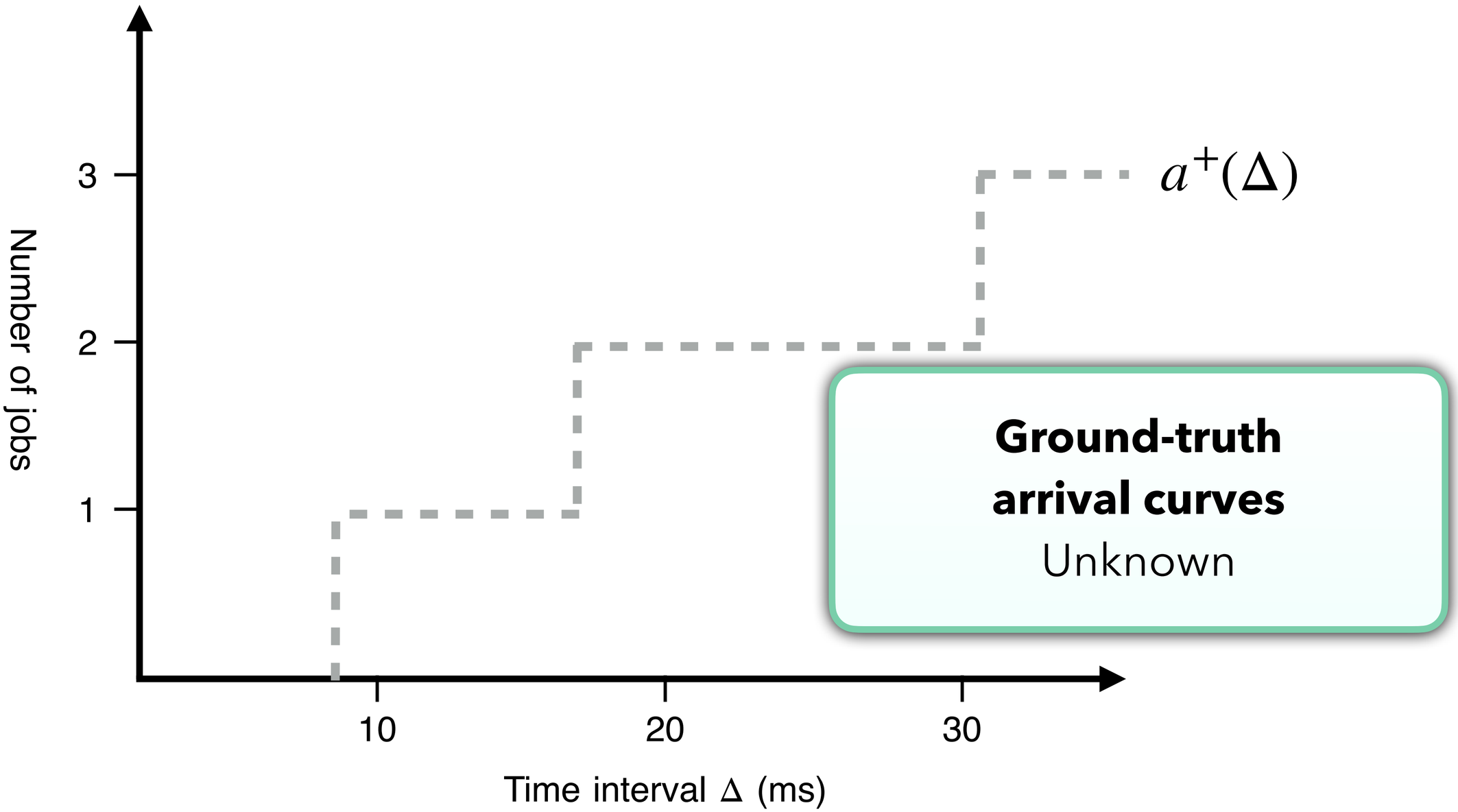
CLASSIC ARRIVAL CURVES: POINT RELEASES

Point release arrival curves: $a^{-}(\Delta) \leq |\{i \mid r_i \in I_t^{\Delta}\}| \leq a^{+}(\Delta)$



CLASSIC ARRIVAL CURVES: POINT RELEASES

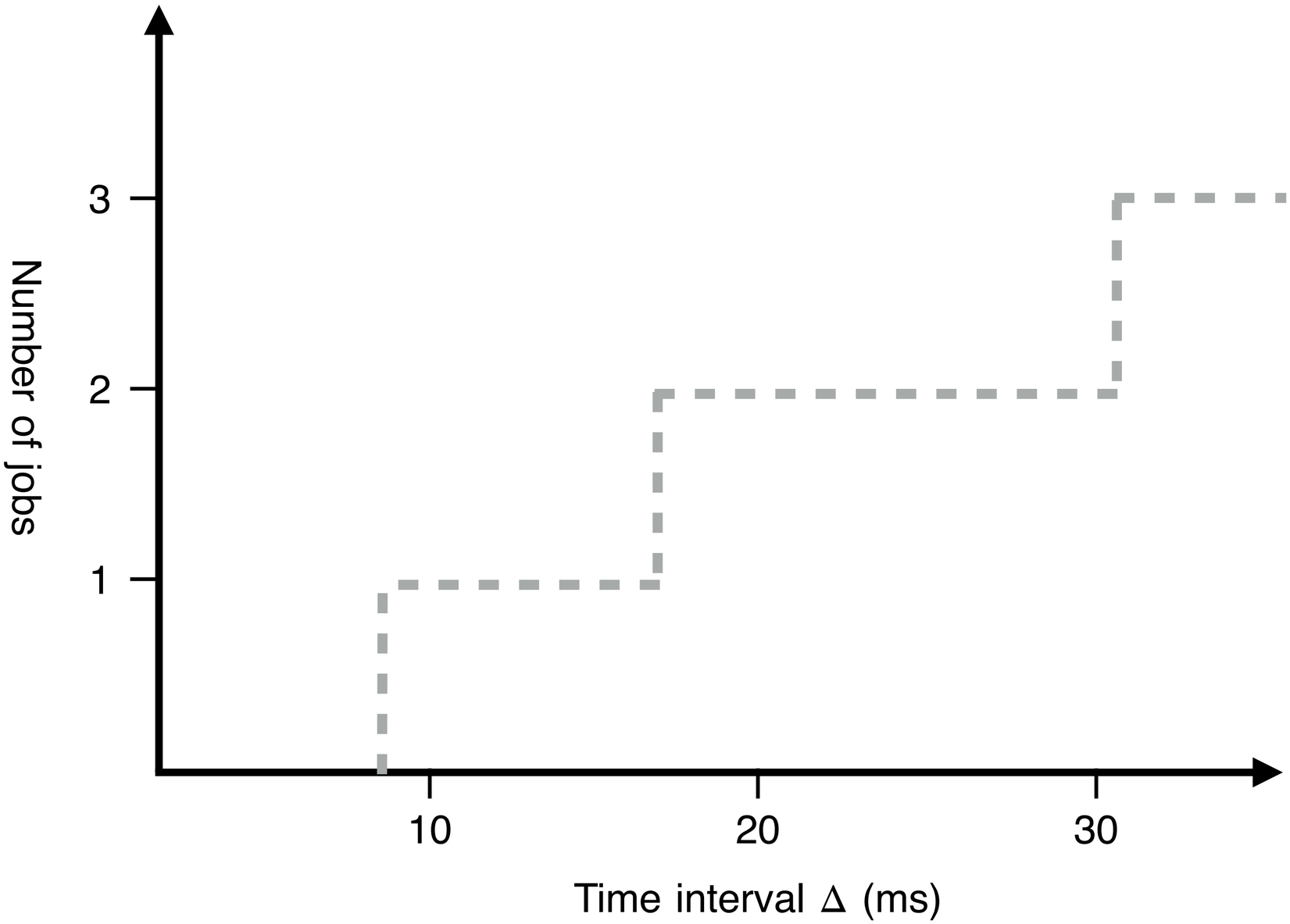
Point release arrival curves: $a^{-}(\Delta) \leq |\{i \mid r_i \in I_t^{\Delta}\}| \leq a^{+}(\Delta)$



THIS PAPER: APPROXIMATE ARRIVAL CURVES

Under-approx. Lower arrivals $\Rightarrow a_{lo}^{-}(\Delta) \leq |\{i \,|\, R_i \subseteq I_t^{\Delta}\}| \leq a_{lo}^{+}(\Delta) \Leftarrow$ **Under-approx.** Upper Arrivals

Over-approx. Lower arrivals $\Rightarrow a_{hi}^{-}(\Delta) \leq |\{i \,|\, R_i \cap I_t^{\Delta} \neq \emptyset\}| \leq a_{hi}^{+}(\Delta) \Leftarrow$ **Over-approx.** Upper arrivals

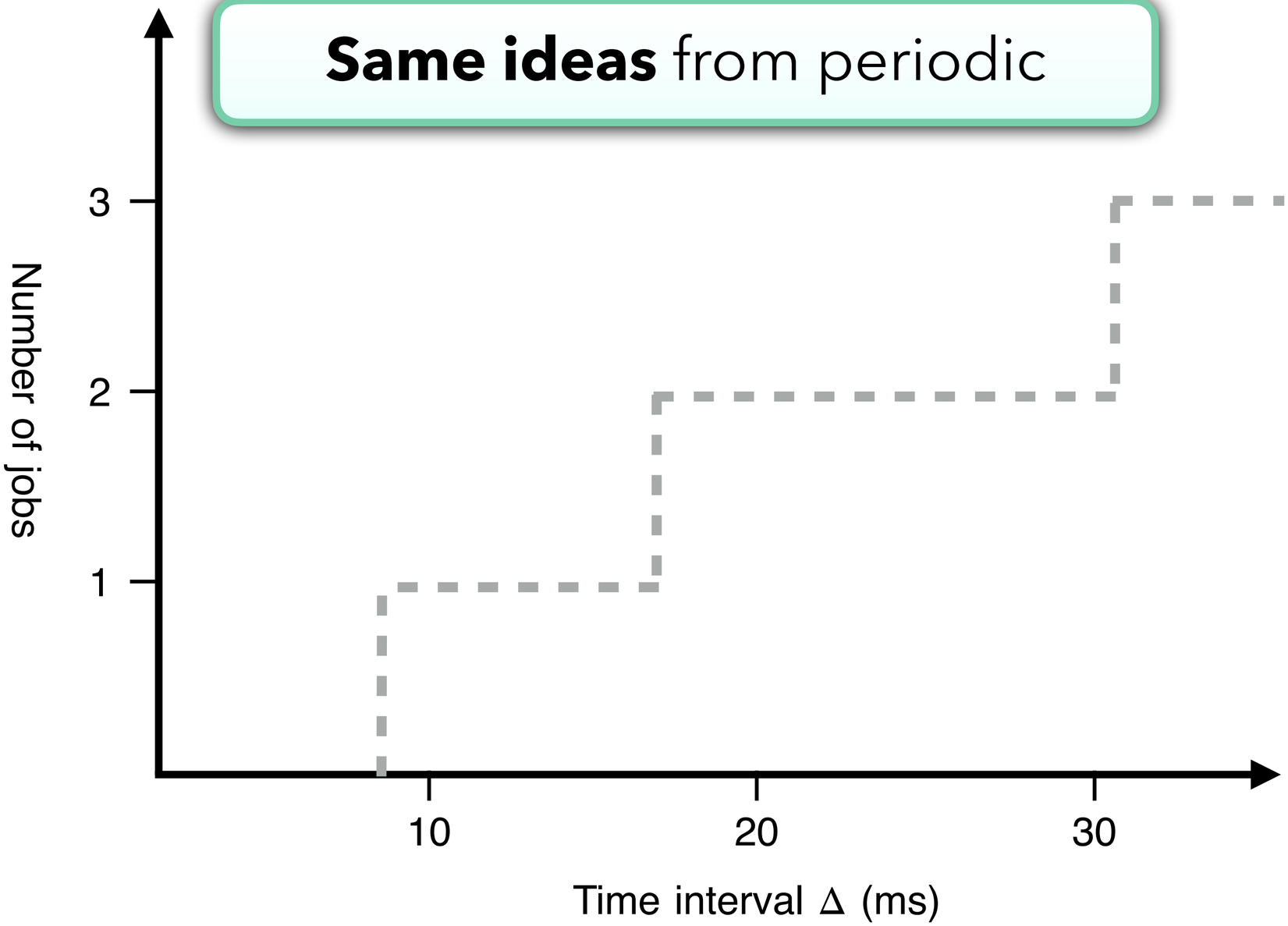


THIS PAPER: APPROXIMATE ARRIVAL CURVES

Under-approx. Lower arrivals $\Rightarrow a^-_{lo}(\Delta) \leq |\{i \,|\, R_i \subseteq I_t^\Delta\}| \leq a^+_{lo}(\Delta) \Leftarrow$ **Under-approx.** Upper Arrivals

Over-approx. Lower arrivals $\Rightarrow a^-_{hi}(\Delta) \leq |\{i \,|\, R_i \cap I_t^\Delta \neq \emptyset\}| \leq a^+_{hi}(\Delta) \Leftarrow$ **Over-approx.** Upper arrivals

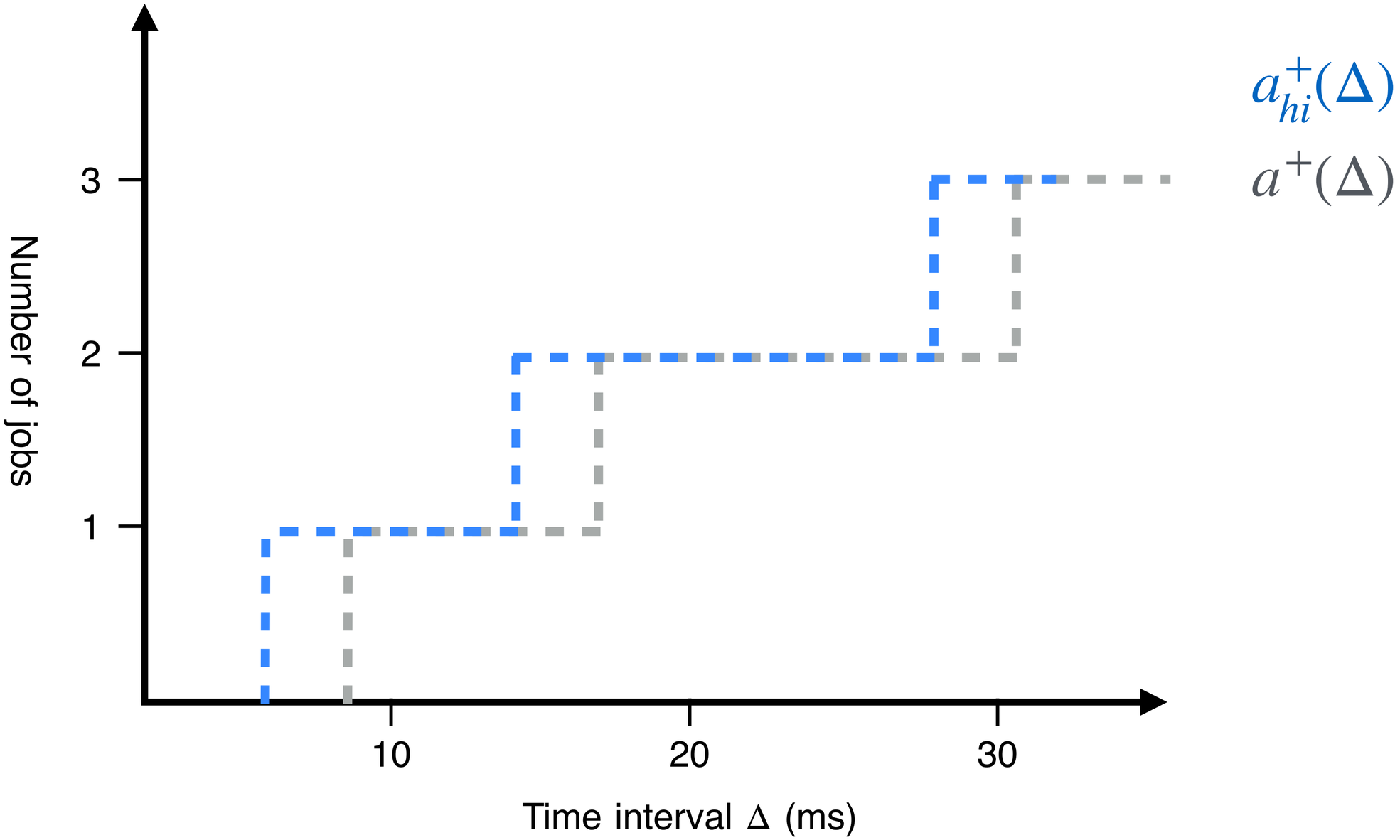
Same ideas from periodic



THIS PAPER: APPROXIMATE ARRIVAL CURVES

Under-approx. Lower arrivals $\Rightarrow a_{lo}^{-}(\Delta) \leq |\{i \,|\, R_i \subseteq I_t^{\Delta}\}| \leq a_{lo}^{+}(\Delta) \Leftarrow$ **Under-approx.** Upper Arrivals

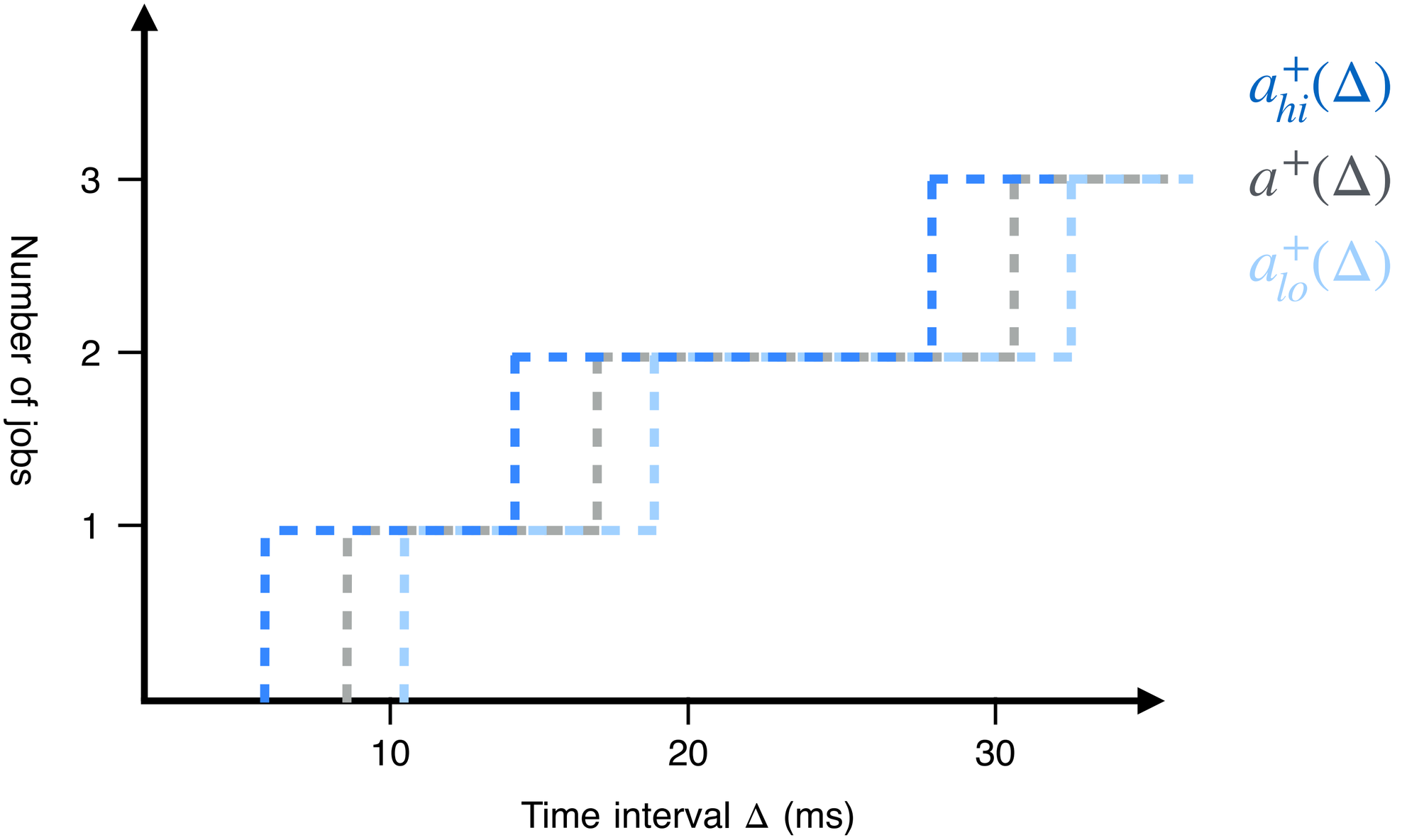
Over-approx. Lower arrivals $\Rightarrow a_{hi}^{-}(\Delta) \leq |\{i \,|\, R_i \cap I_t^{\Delta} \neq \emptyset\}| \leq a_{hi}^{+}(\Delta) \Leftarrow$ **Over-approx.** Upper arrivals



THIS PAPER: APPROXIMATE ARRIVAL CURVES

Under-approx. Lower arrivals $\Rightarrow a^-_{lo}(\Delta) \leq |\{i \,|\, R_i \subseteq I_t^\Delta\}| \leq a^+_{lo}(\Delta) \Leftarrow$ **Under-approx.** Upper Arrivals

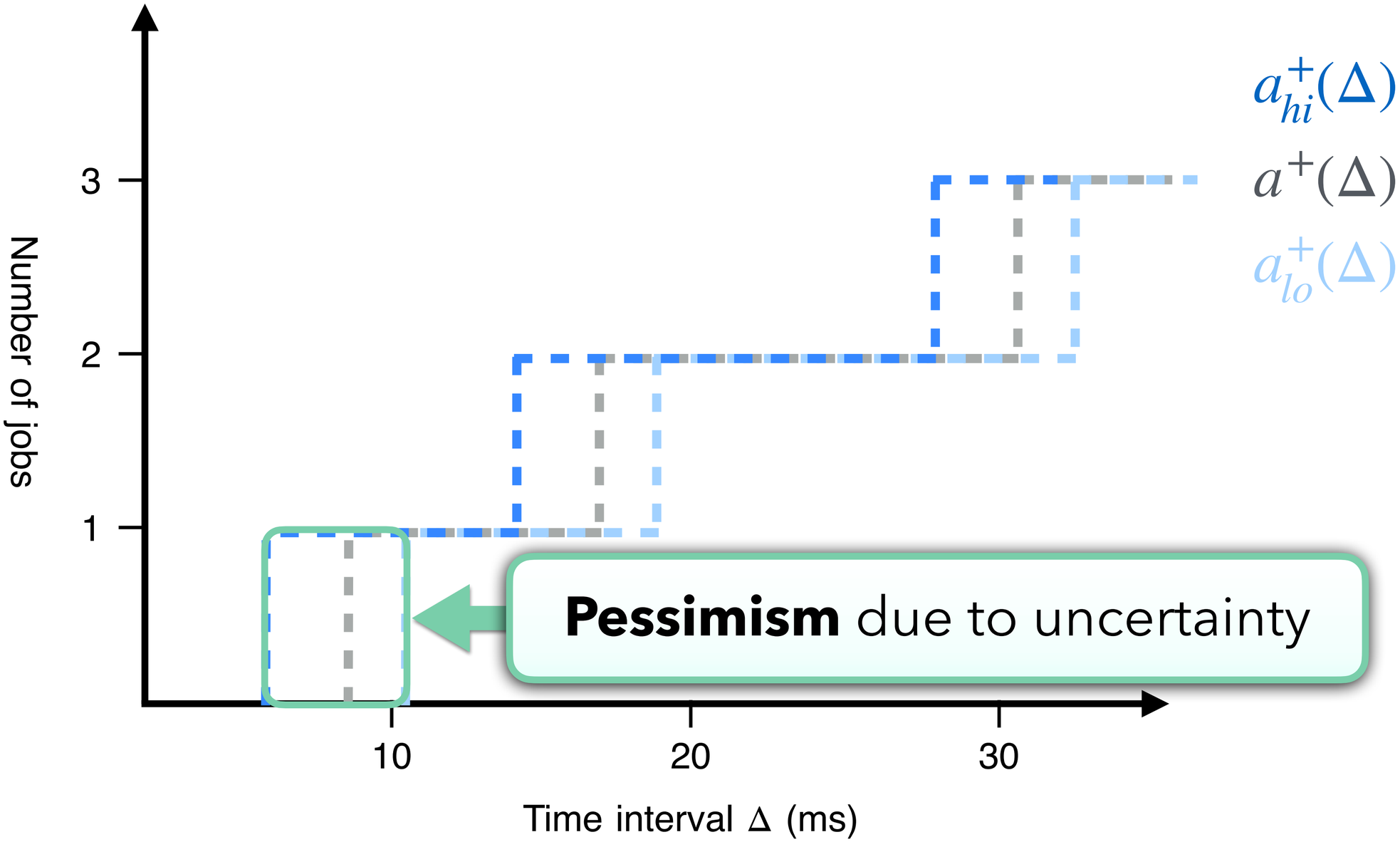
Over-approx. Lower arrivals $\Rightarrow a^-_{hi}(\Delta) \leq |\{i \,|\, R_i \cap I_t^\Delta \neq \emptyset\}| \leq a^+_{hi}(\Delta) \Leftarrow$ **Over-approx.** Upper arrivals



THIS PAPER: APPROXIMATE ARRIVAL CURVES

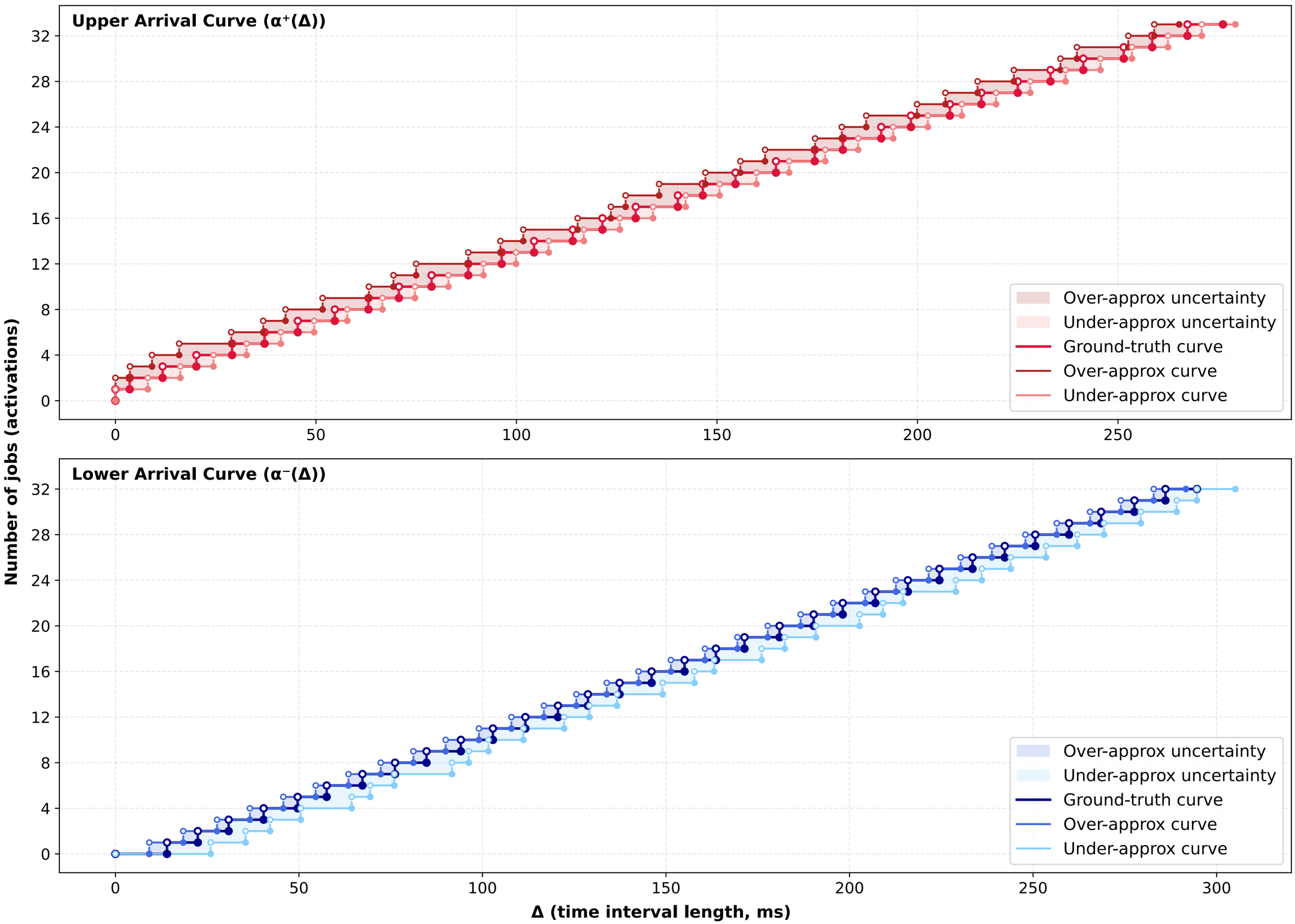
Under-approx. Lower arrivals $\Rightarrow a_{lo}^{-}(\Delta) \leq |\{i \,|\, R_i \subseteq I_t^{\Delta}\}| \leq a_{lo}^{+}(\Delta) \Leftarrow$ **Under-approx.** Upper Arrivals

Over-approx. Lower arrivals $\Rightarrow a_{hi}^{-}(\Delta) \leq |\{i \,|\, R_i \cap I_t^{\Delta} \neq \emptyset\}| \leq a_{hi}^{+}(\Delta) \Leftarrow$ **Over-approx.** Upper arrivals

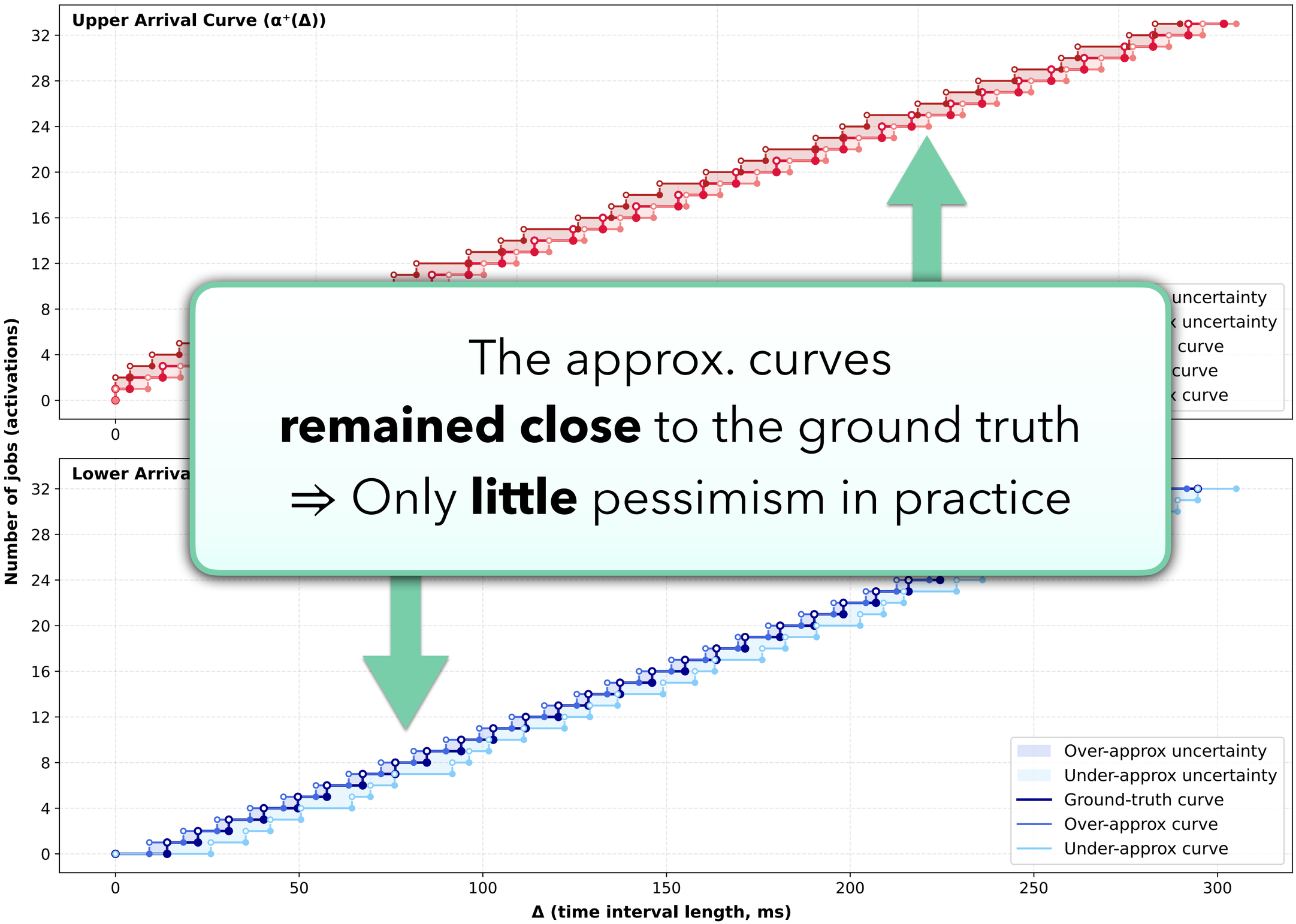


ARRIVAL CURVES FROM ACTUAL WORKLOAD

ARRIVAL CURVES FROM ACTUAL WORKLOAD



ARRIVAL CURVES FROM ACTUAL WORKLOAD

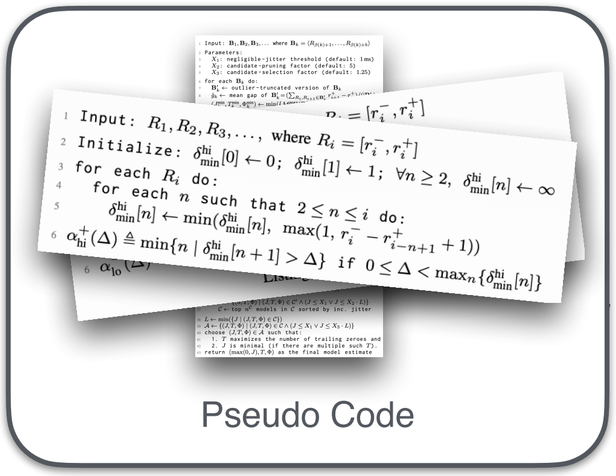


The approx. curves
remained close to the ground truth
⇒ Only **little** pessimism in practice

LiME IT

IMPLEMENTATION

OUR IMPLEMENTATION IN A NUTSHELL



Pseudo Code

Clean & Straightforward Implementation

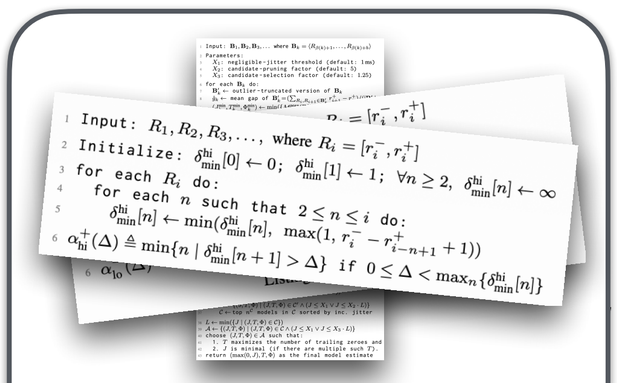
Python: Executable Specification

Performance-Focused Implementation

Automated Equivalence Tests

Rust: Model Extractor

OUR IMPLEMENTATION IN A NUTSHELL



Pseudo Code

Clean & Straightforward Implementation

Python: Executable Specification

Performance-Focused Implementation

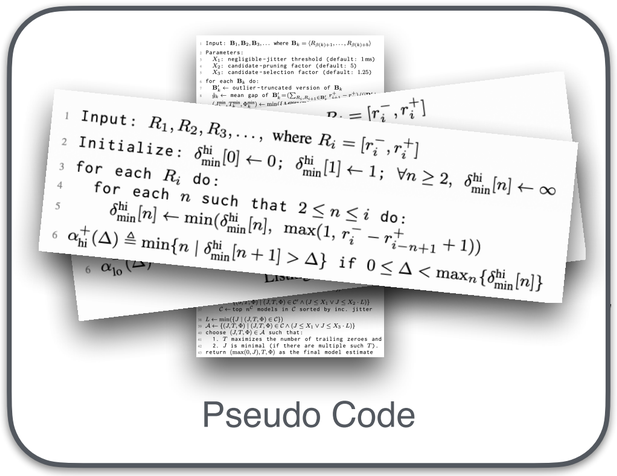
Automated Equivalence Tests

Rust: Model Extractor

Available as standalone libraries

Useful for general model inference

OUR IMPLEMENTATION IN A NUTSHELL



Pseudo Code

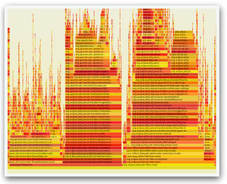
Clean & Straightforward Implementation

Python: Executable Specification

Performance-Focused Implementation

Automated Equivalence Tests

Rust: Model Extractor

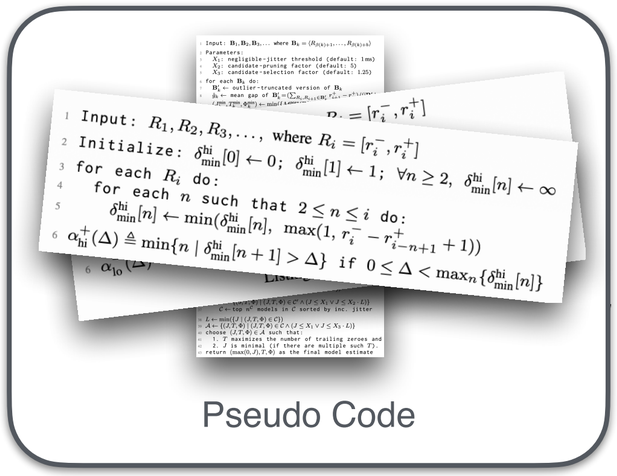


Linux Workloads

Profiling with *perf*

Rust: Intra-Thread Task Finder

OUR IMPLEMENTATION IN A NUTSHELL



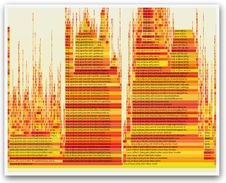
Pseudo Code

Clean & Straightforward Implementation

Python: Executable Specification

Performance-Focused Implementation

Automated Equivalence Tests



Linux Workloads

Profiling with *perf*

Rust: Intra-Thread Task Finder

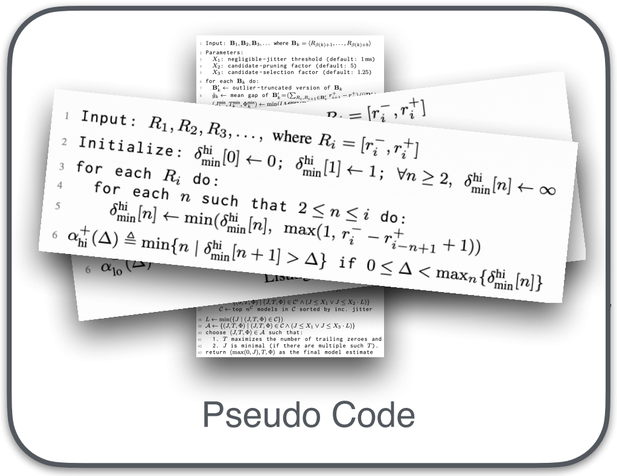
JSON

Configure LiME^{IT} with discovered intra-thread task

Rust: Model Extractor

LiME^{IT}

OUR IMPLEMENTATION IN A NUTSHELL



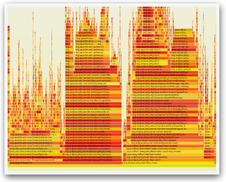
Pseudo Code

Clean & Straightforward Implementation

Python: Executable Specification

Performance-Focused Implementation

Automated Equivalence Tests



Linux Workloads

Profiling with *perf*

Rust: Intra-Thread Task Finder

JSON

Configure LiME^{IT} with discovered intra-thread task

Rust: Model Extractor

LiME^{IT}

eBPF

Tracing of raw events at both thread and intra-thread level

Output Models:

Arrival curve, periodic, etc...

A ROS 2 CASE STUDY

ROS 2 CASE STUDY

We used the reference system made by the ROS 2 Real-Time Working Group

➔ https://github.com/ros-realtime/reference-system

ROS 2 Real-Time Working Group

ROS 2 CASE STUDY

We used the reference system made by the ROS 2 Real-Time Working Group

➔ https://github.com/ros-realtime/reference-system

The system provides different configurations, and we target a

"Prioritized Executor" setup

➔ 5 executors at different priorities

➔ 24 nodes that contain 29 callbacks (intra-thread tasks)

➔ 7 timer-driven callbacks and 22 message-driven callbacks

ROS 2 Real-Time Working Group

OBSERVATIONS IN ROS 2 CASE STUDY

OBSERVATIONS IN ROS 2 CASE STUDY

Identified all callbacks

Timer Callbacks

Callback	Executor	Prio	Period [ms] Configured	Period [ms] Inferred	Jitter [ms] PF	Jitter [ms] CF
BP	planner	RR 30	100	100	0.06	0.11
FLidar	front	RR 1	100	100	0	0.08
RLidar	rear	RR 1	100	100	0	0.09
PCMap	other	EEVDF	120	120	0.12	36.88
Vis	other	EEVDF	60	60	0.35	43.73
L2Map	other	EEVDF	100	100	0.50	40.62
ECSet	other	EEVDF	25	25	0.73	44.29

Message Callbacks

Callback	Trigger(s)	Executor	Prio	Inferred Period [ms]	Jitter [ms] PF	Jitter [ms] CF
PTF	FLidar	front	RR 1	100	0	0.34
PTR	RLidar	rear	RR 1	100	0	0.42
RGF	PCF	fusion	RR 1	100	0	41.26
OCE	ECD	fusion	RR 1	100	0	11.24
PCF_1	PTR	fusion	RR 1	100	0.17	0.37
PCF_2	PTR	fusion	RR 1	100	0.52	0.79
ECD	RGF, ECSet	fusion	RR 1	20	15.62	29.08
VGD	PCF	other	EEVDF	100	9.93	47.71
PCML	PCMap	other	EEVDF	120	1.10	44.72
MPC	BP	other	EEVDF	100	0.68	43.38
Park	L2ML	other	EEVDF	120	0	69.59
LPlan	L2ML	other	EEVDF	120	0	66.16
ITO	ECD	other	EEVDF	25	7.36	56.26
NDT_1	VGD	other	EEVDF	100	0	53.28
NDT_2	PCML	other	EEVDF	120	0	48.57
VI_1	MPC	other	EEVDF	100	0	47.60
VI_2	BP	other	EEVDF	100	0	32.60
$L2GP_1$	Vis	other	EEVDF	60	0	72.03
$L2GP_2$	NDT	other	EEVDF	120	0	55.40
$L2ML_1$	L2Map	other	EEVDF	100	4.85	61.93
$L2ML_2$	L2GP	other	EEVDF	120	0	54.94
VDBW	VI	other	EEVDF	100	0	62.30

OBSERVATIONS IN ROS 2 CASE STUDY

Identified all callbacks

Extracted periods for timer callbacks matches the configuration

Timer Callbacks

Callback	Executor	Prio	Period [ms] Configured	Period [ms] Inferred	Jitter [ms] PF	Jitter [ms] CF
BP	planner	RR 30	100	100	0.06	0.11
FLidar	front	RR 1	100	100	0	0.08
RLidar	rear	RR 1	100	100	0	0.09
PCMap	other	EEVDF	120	120	0.12	36.88
Vis	other	EEVDF	60	60	0.35	43.73
L2Map	other	EEVDF	100	100	0.50	40.62
ECSet	other	EEVDF	25	25	0.73	44.29

Message Callbacks

Callback	Trigger(s)	Executor	Prio	Inferred Period [ms]	Jitter [ms] PF	Jitter [ms] CF
PTF	FLidar	front	RR 1	100	0	0.34
PTR	RLidar	rear	RR 1	100	0	0.42
RGF	PCF	fusion	RR 1	100	0	41.26
OCE	ECD	fusion	RR 1	100	0	11.24
PCF_1	PTR	fusion	RR 1	100	0.17	0.37
PCF_2	PTR	fusion	RR 1	100	0.52	0.79
ECD	RGF, ECSet	fusion	RR 1	20	15.62	29.08
VGD	PCF	other	EEVDF	100	9.93	47.71
PCML	PCMap	other	EEVDF	120	1.10	44.72
MPC	BP	other	EEVDF	100	0.68	43.38
Park	L2ML	other	EEVDF	120	0	69.59
LPlan	L2ML	other	EEVDF	120	0	66.16
ITO	ECD	other	EEVDF	25	7.36	56.26
NDT_1	VGD	other	EEVDF	100	0	53.28
NDT_2	PCML	other	EEVDF	120	0	48.57
VI_1	MPC	other	EEVDF	100	0	47.60
VI_2	BP	other	EEVDF	100	0	32.60
$L2GP_1$	Vis	other	EEVDF	60	0	72.03
$L2GP_2$	NDT	other	EEVDF	120	0	55.40
$L2ML_1$	L2Map	other	EEVDF	100	4.85	61.93
$L2ML_2$	L2GP	other	EEVDF	120	0	54.94
VDBW	VI	other	EEVDF	100	0	62.30

OBSERVATIONS IN ROS 2 CASE STUDY

Identified all callbacks

Extracted periods for timer callbacks matches the configuration

Timer Callbacks

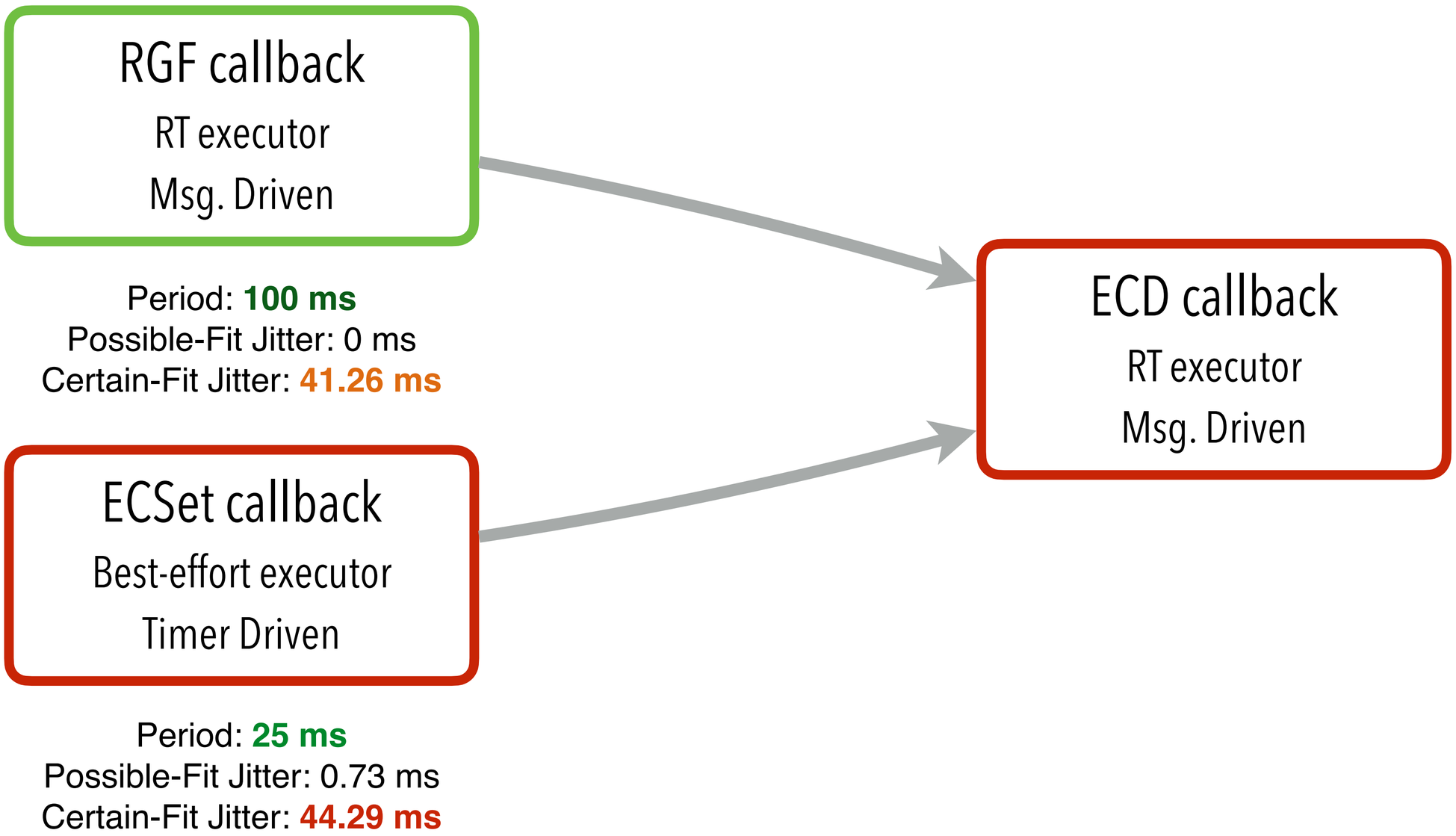
Callback	Executor	Prio	Period [ms] Configured	Period [ms] Inferred		
BP	planner	RR 30	100	100	0	
FLidar	front	RR 1	100	100		
RLidar	rear	RR 1	100	100		
PCMap	other	EEVDF	120	120	0.12	
Vis	other	EEVDF	60	60	0.35	43.73
L2Map	other	EEVDF	100	100	0.50	40.62
ECSet	other	EEVDF	25	25	0.73	44.29

Message Callbacks

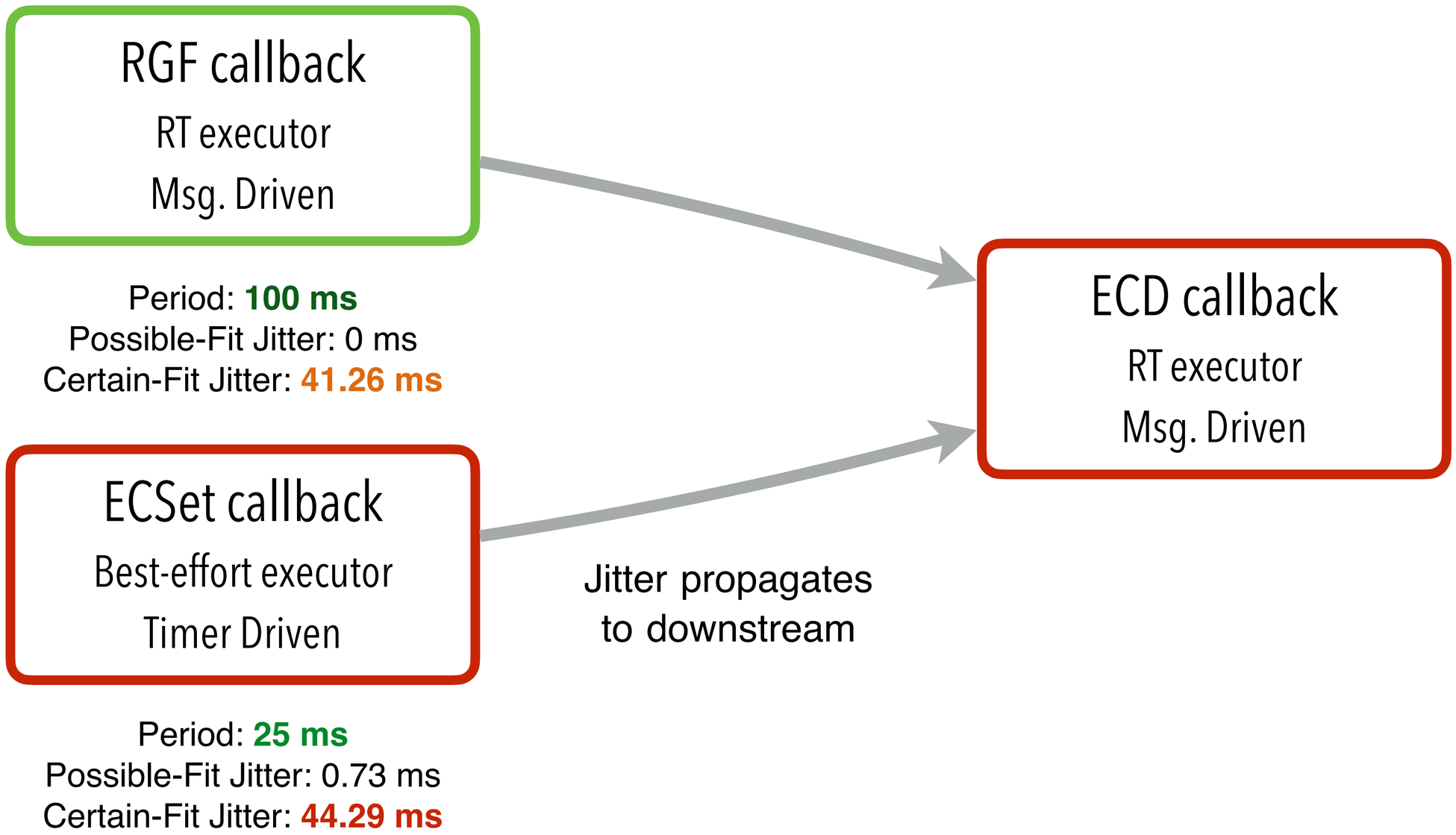
Callback	Trigger(s)	Executor	Prio	Inferred Period [ms]	Jitter [ms] PF	Jitter [ms] CF
PTF	FLidar	front	RR 1	100	0	0.34
PTR	RLidar	rear	RR 1	100	0	0.42
RGF	PCF	fusion	RR 1	100	0	41.26
OCE	ECD	fusion	RR 1	100	0	11.24
PCF_1	PTR	fusion	RR 1	100	0.17	0.37
PCF_2	PTR	fusion	RR 1	100	0.52	0.79
ECD	RGF, ECSet	fusion	RR 1	20	15.62	29.08
VGD	PCF	other	EEVDF	100	9.93	47.71
PCML	PCMap	other	EEVDF	120	1.10	44.72
MPC	BP	other	EEVDF	100	0.68	43.38
						69.59
						.6
						.40
$L2ML_1$	L2Map	other	EEVDF	100	4.85	61.93
$L2ML_2$	L2GP	other	EEVDF	120	0	54.94
VDBW	VI	other	EEVDF	100	0	62.30

One callback's **possible-fit** periodic model has a **large jitter bound**, why?

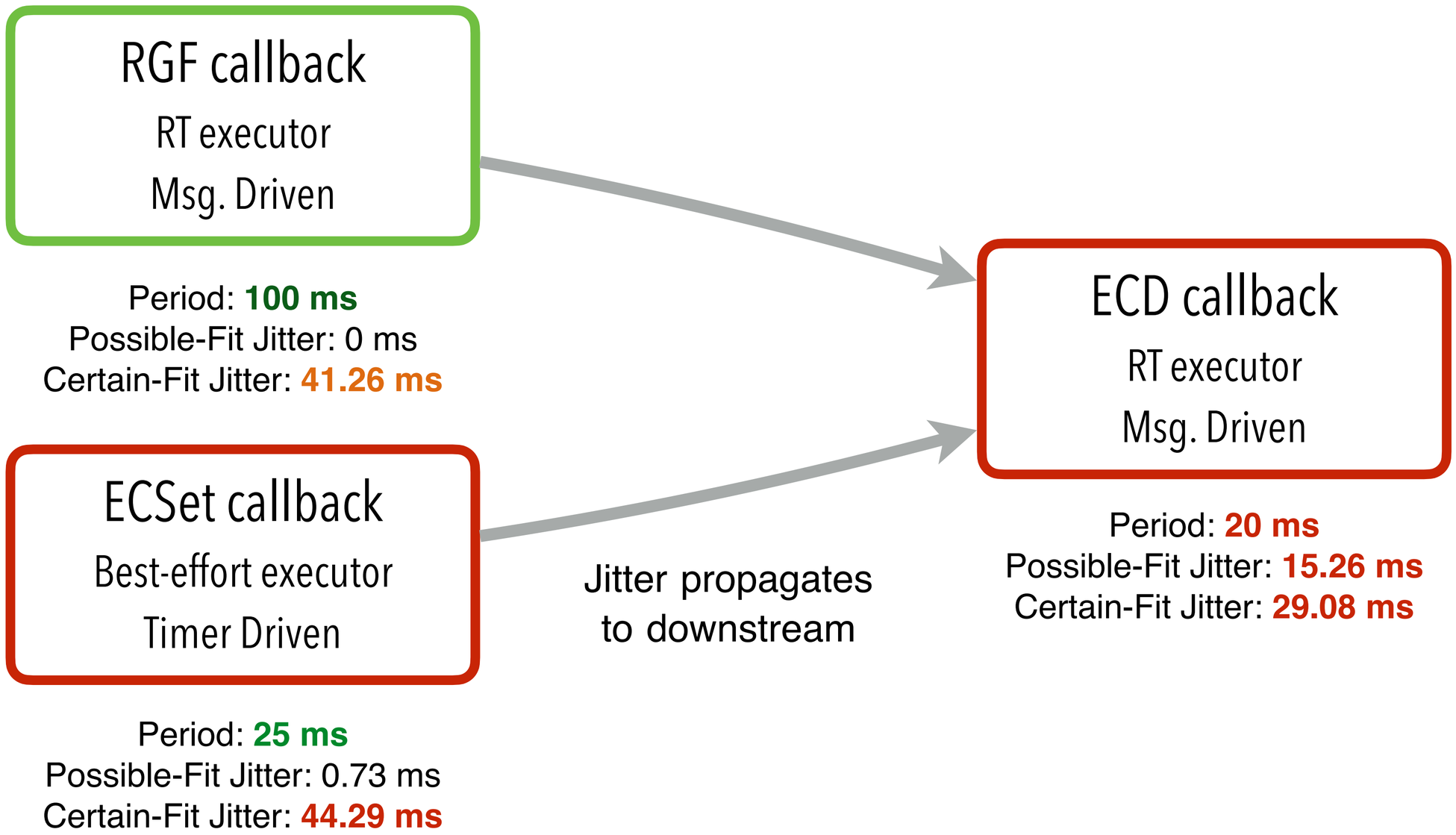
LiME^{IT} REVEALS SUB-OPTIMAL CONFIGURATION



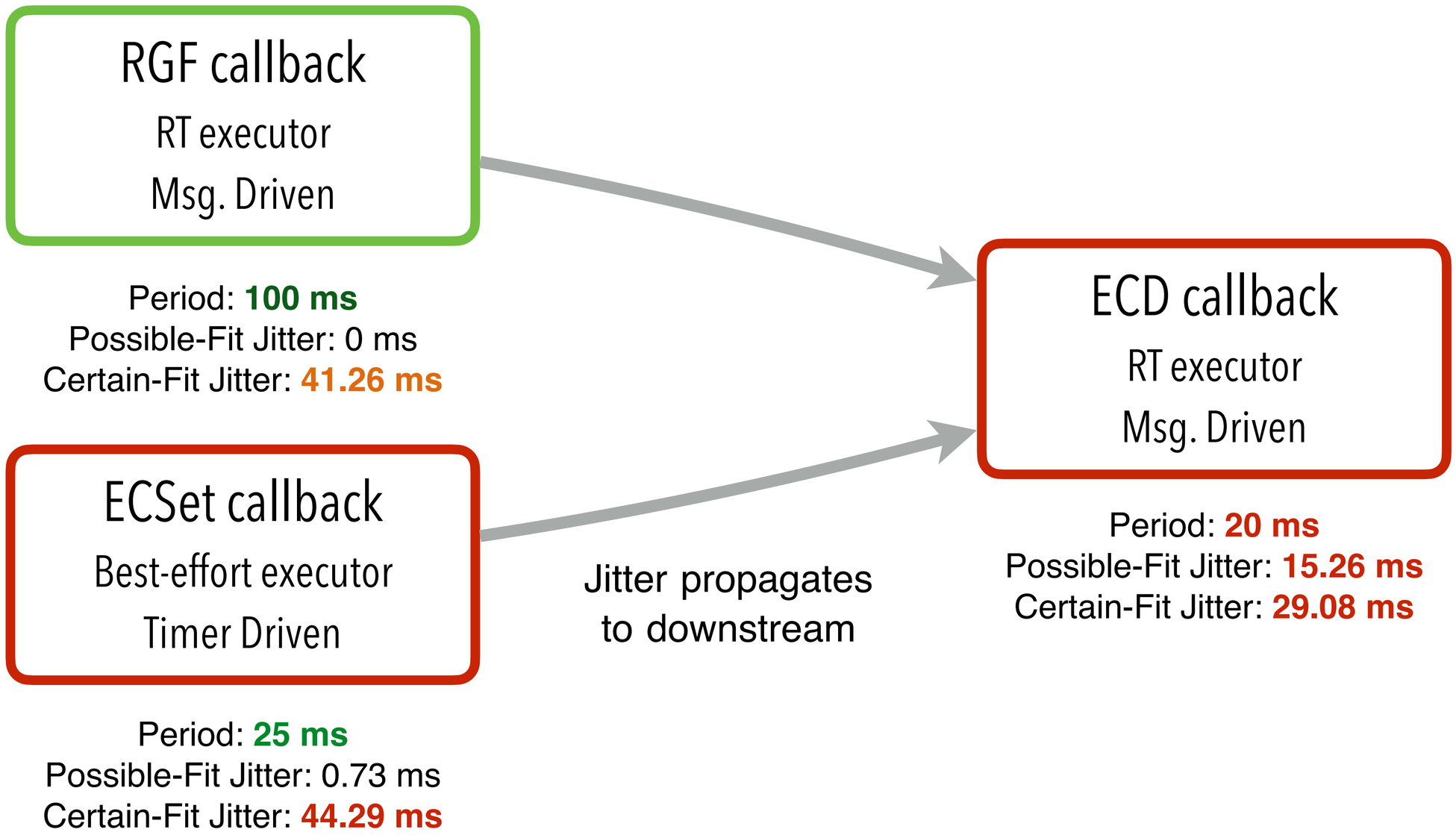
LiME^{IT} REVEALS SUB-OPTIMAL CONFIGURATION



LiME^{IT} REVEALS SUB-OPTIMAL CONFIGURATION



LiME^{IT} REVEALS SUB-OPTIMAL CONFIGURATION



Sub-optimal configuration causes weak periodicity of real-time callback

LiME^{IT} REVEALS SUB-OPTIMAL CONFIGURATION

RGF callback

RT executor

Msg. Driven

Po

Cert

LiME^{IT} can reveal

thread-internal timing issues

that can be difficult to observe otherwise

back

tor

ven

ms

Possible-Fit Jitter: **15.26 ms**

Certain-Fit Jitter: **29.08 ms**

Best-effort executor

Timer Driven

Jitter propagates

to downstream

Period: **25 ms**

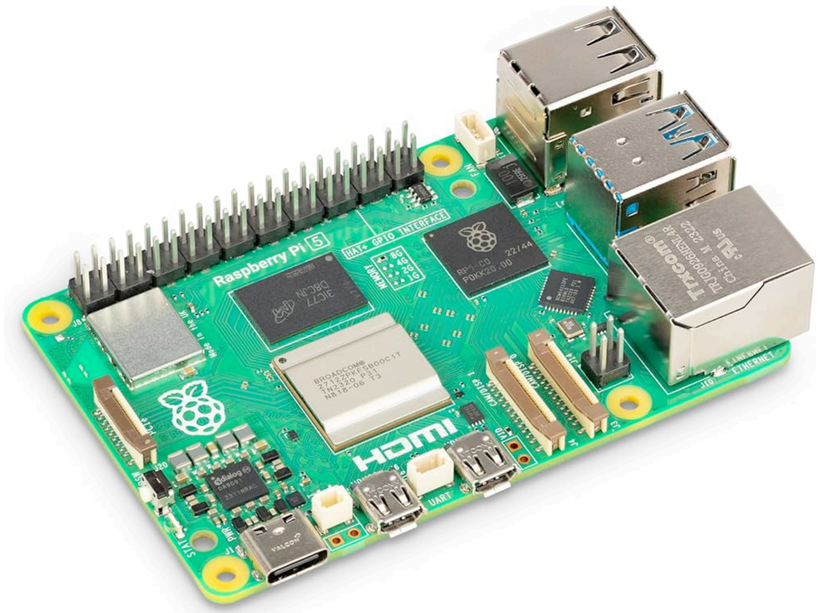
Possible-Fit Jitter: 0.73 ms

Certain-Fit Jitter: **44.29 ms**

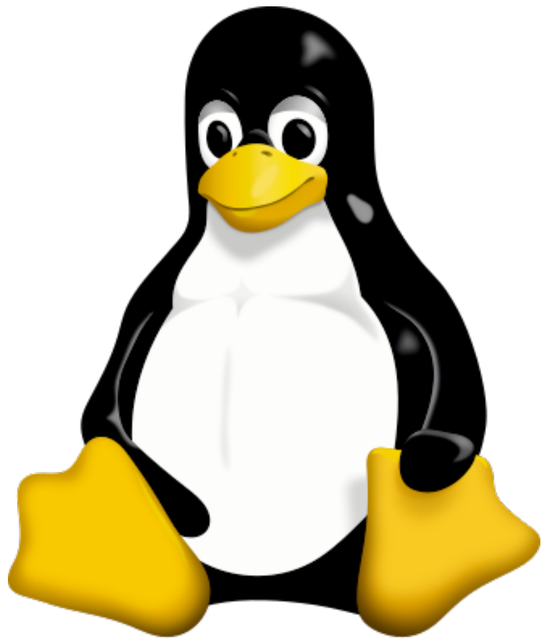
Sub-optimal configuration causes weak periodicity of real-time callback

QUANTITATIVE EVALUATION: HIGHLIGHTS

COMMODITY PLATFORM



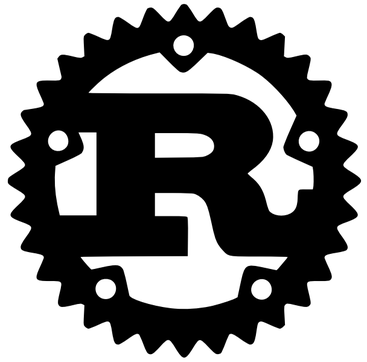
Raspberry **Pi 5**



Linux **6.12**
PREEMPT_RT

DEMONSTRATING LANGUAGE DIVERSITY

Arrival curves evaluation using C++
Covered **304,800** generated tasks



Periodic models evaluation using Rust
Covered **76,200** generated tasks

C

Overhead evaluation using Rössl* (C)
Sweeping from **10-100Hz** with **1-10** tasks

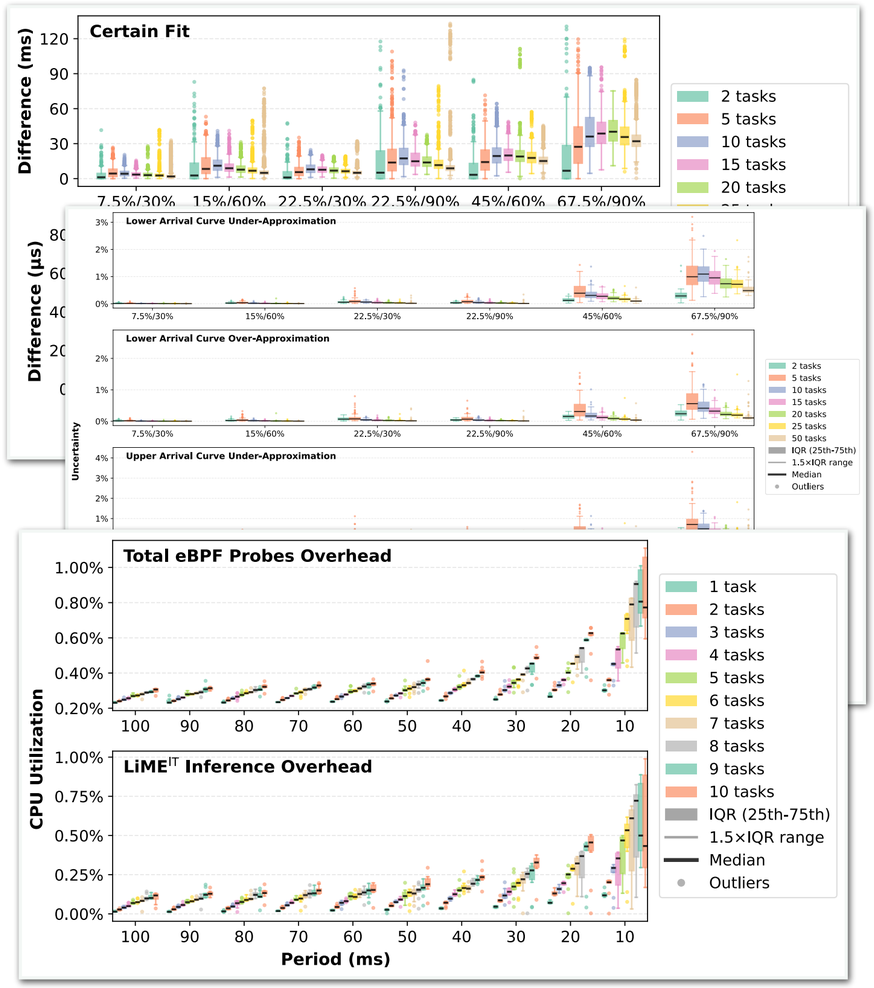
*K. Bedarkar et al., PLDI 2025

ACCURATE AND FAST

ACCURATE AND FAST

LiME^{IT} is accurate

➔ Possible-fit models recovered **100%** of ground-truth periods, while certain-fit recovered **99.73%** of them.

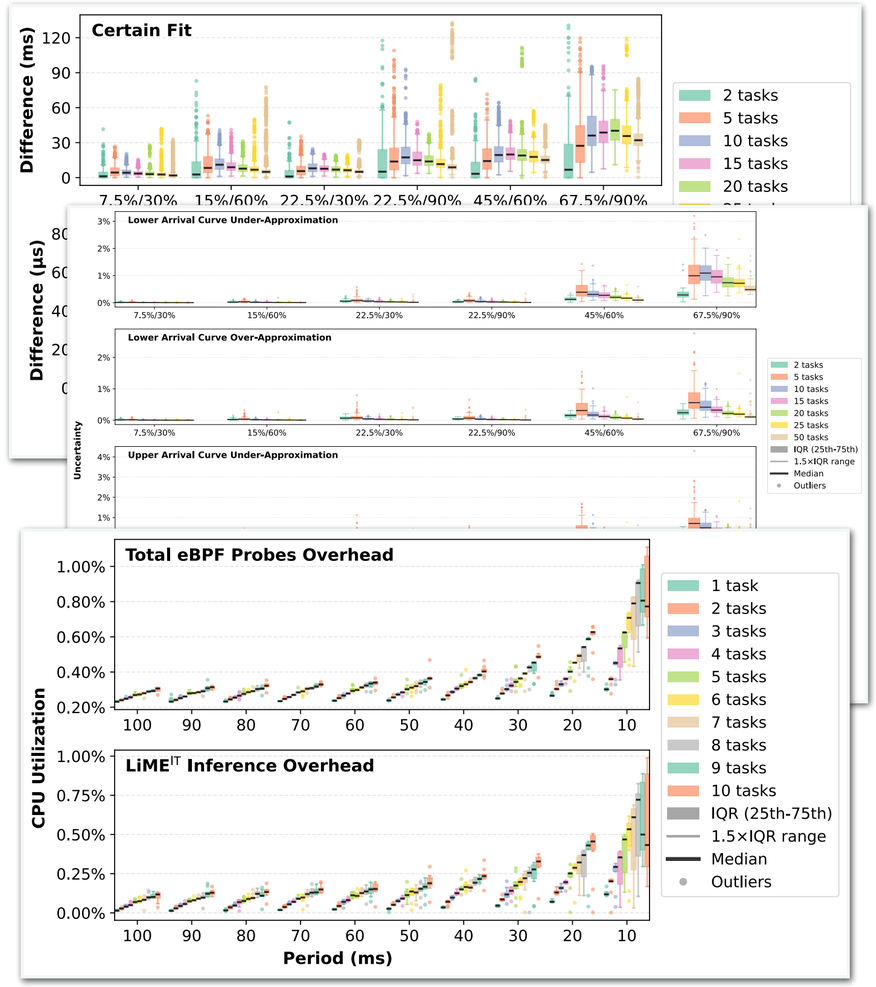


ACCURATE AND FAST

LiME^{IT} is accurate

➔ Possible-fit models recovered **100%** of ground-truth periods, while certain-fit recovered **99.73%** of them.

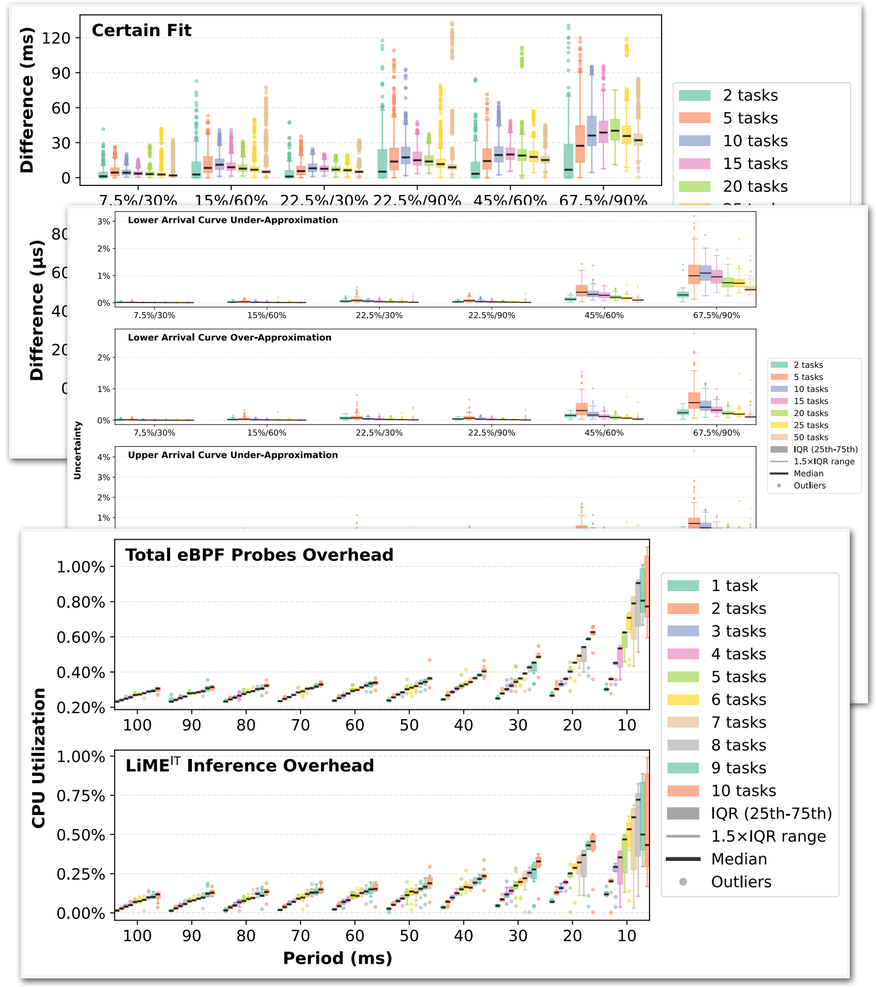
Can achieve **100%** with inference parameters tweaking



ACCURATE AND FAST

LiME^{IT} is accurate

- ➔ Possible-fit models recovered **100%** of ground-truth periods, while certain-fit recovered **99.73%** of them.
- ➔ All ground-truth arrival curves are bounded with less than **4%** pessimism.



ACCURATE AND FAST

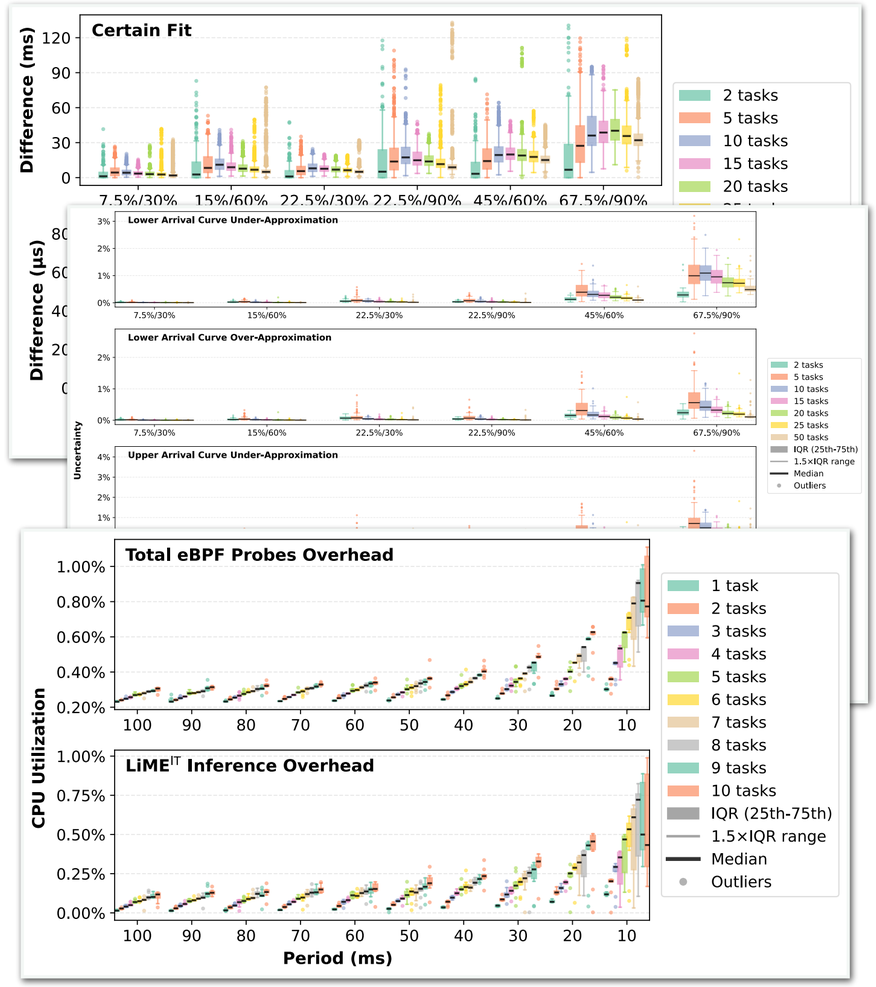
LiME^{IT} is accurate

➔ Possible-fit models recovered **100%** of ground-truth periods, while certain-fit recovered **99.73%** of them.

➔ All ground-truth arrival curves are bounded with less than **4%** pessimism.

Cost is acceptably low

➔ Runtime overhead was below **2.5%** on a single core.



ACCURATE AND FAST

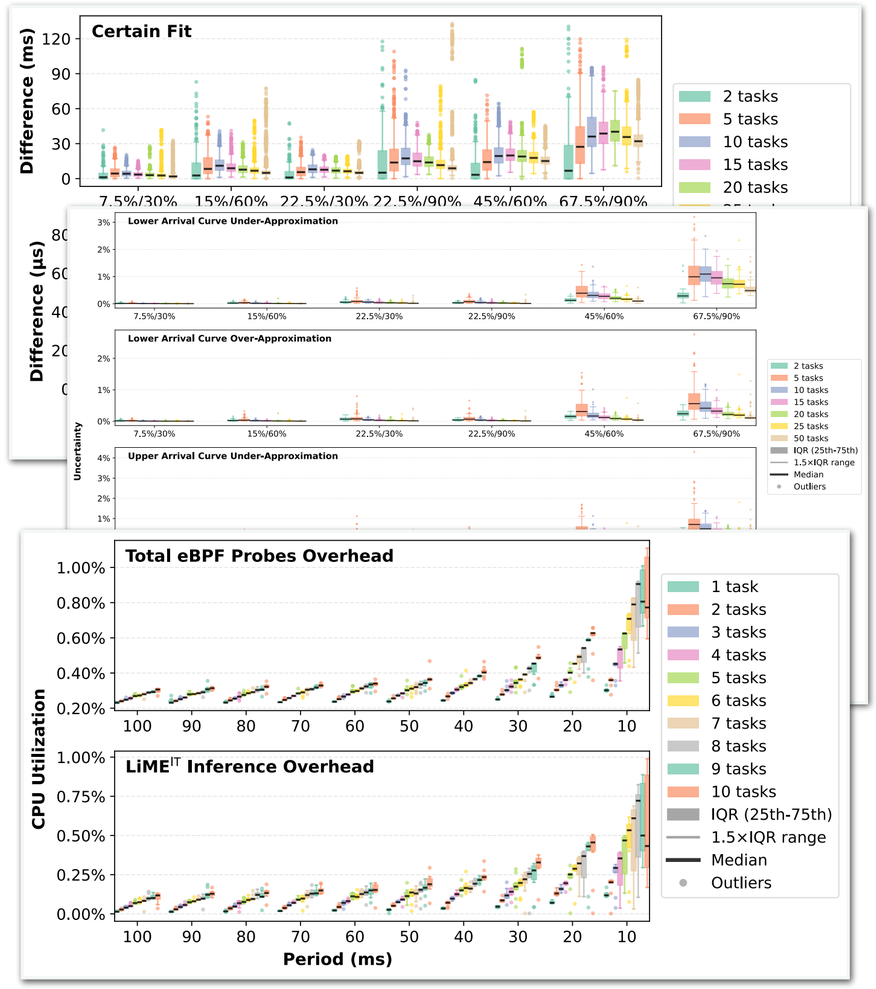
LiME^{IT} is accurate

➔ Possible-fit models recovered **100%** of ground-truth periods, while certain-fit recovered **99.73%** of them.

➔ All ground-truth arrival curves are bounded with less than **4%** pessimism.

Cost is acceptably low

➔ Runtime overhead was below **2.5%** on a single core.



More detail in the paper

CONCLUSION

LiME^{IT}

The first **framework-agnostic** dynamic **model** extractor for **intra-thread tasks** on Linux

➔ Highly Accurate & Low Overheads

Standalone libraries for model inference in **Python** & **Rust**

➔ For Broad Scenarios

LiME^IT

The first **framework-agnostic** dynamic **model** extractor for **intra-thread tasks** on Linux

➔ Highly Accurate & Low Overheads

Standalone libraries for model inference in **Python** & **Rust**

➔ For Broad Scenarios

Project Homepage

https://lime.mpi-sws.org/

More information

LiME^IT

The first **framework-agnostic** dynamic **model** extractor for **intra-thread tasks** on Linux

➔ Highly Accurate & Low Overheads

Standalone libraries for model inference in **Python** & **Rust**

➔ For Broad Scenarios

Project Homepage

https://lime.mpi-sws.org/

More information

Try It Now

GitHub Organization

https://github.com/LiME-org/