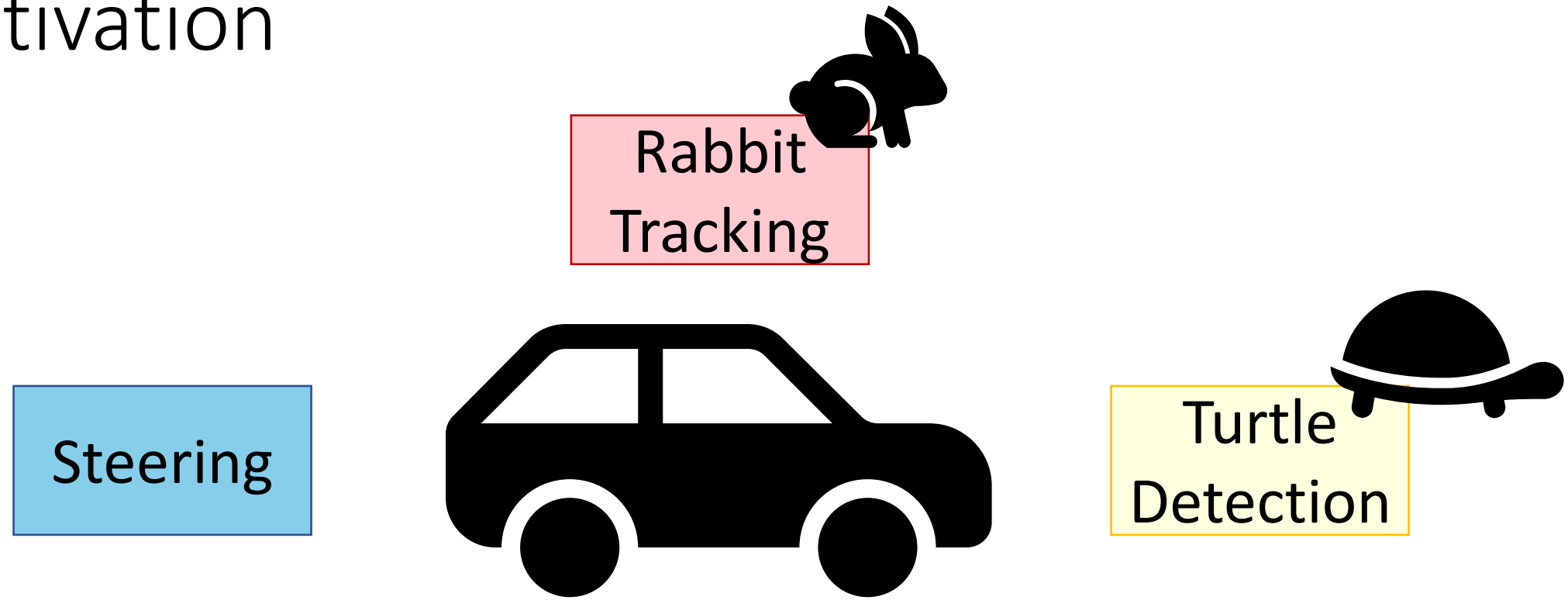
OpenVX and Real-Time Certification: The Troublesome History

Tanya Amert, Sergey Voronov, and James H. Anderson
University of North Carolina at Chapel Hill

Motivation



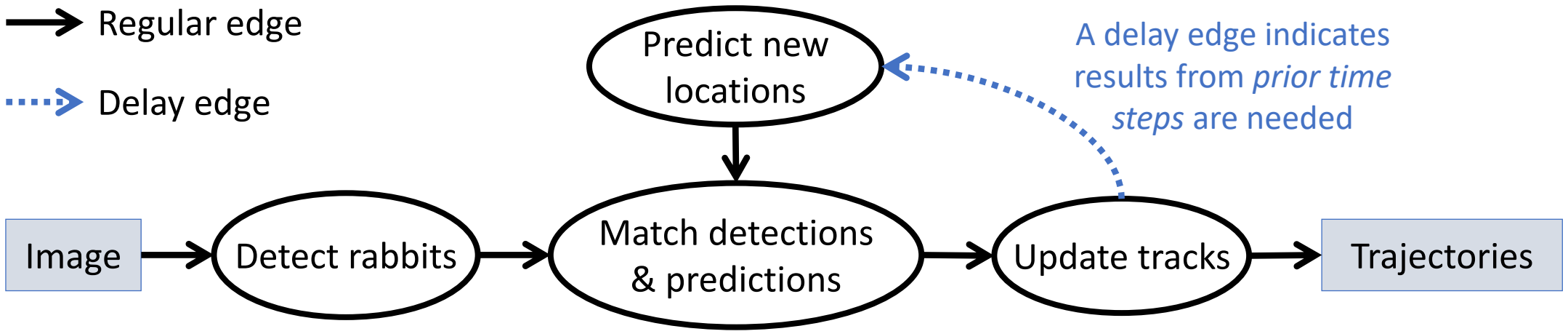
Many of these processes can be represented as graphs.

Can we guarantee response times bounds for them all?

The Challenge

Tracking requires predicting the next position from a series of past positions. This results in a cycle in the graph.

Rabbit Tracking



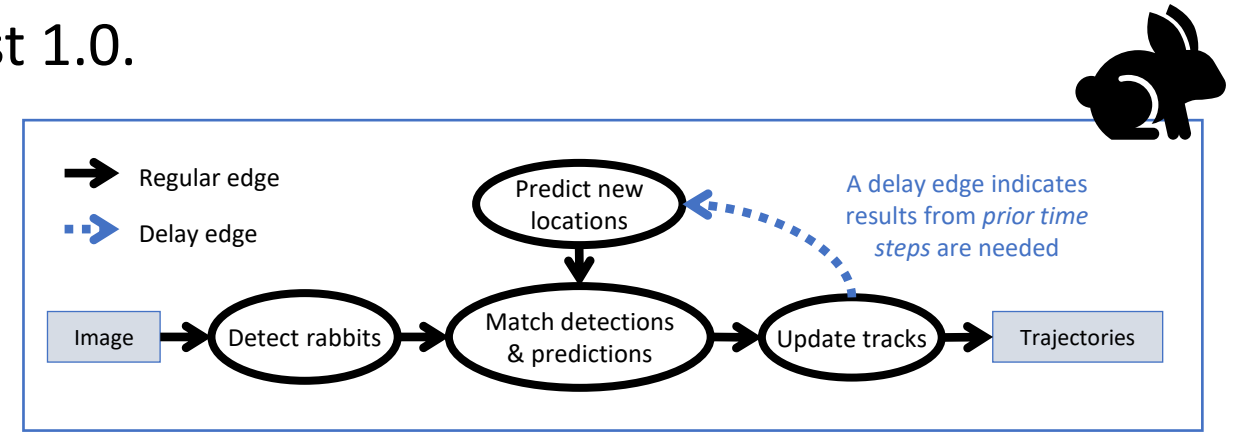
The Challenge

Tracking requires predicting the next position from a series of past positions. This results in a cycle in the graph.

<u>Existing schedulability analysis precludes cycles</u>:

- requires data to be *very* old, or
- requires cycle utilization to be at most 1.0.

GPU sharing can result in cycles with utilization exceeding 1.0.



The Challenge

Tracking requires predicting the next position from a series of past positions. This results in a cycle in the graph.

Existing schedulability analysis precludes cycles:

- requires data to be *very* old, or
- requires cycle utilization to be at most 1.0.

GPU sharing can result in cycles with utilization exceeding 1.0.

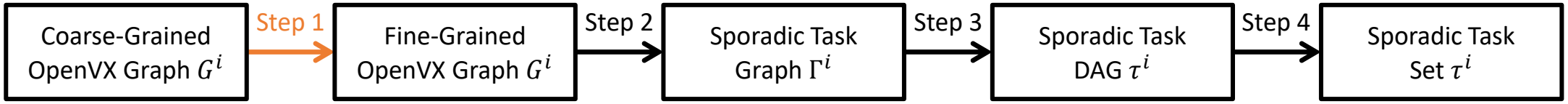
We need response-time analysis that allows for arbitrary cycles in graph-based task systems.

Outline

- Motivation
- Transforming an OpenVX graph to independent sporadic tasks
- Bounding response-times for rp-sporadic tasks
- Evaluation
- Future work

Transformation from graph to tasks

Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.



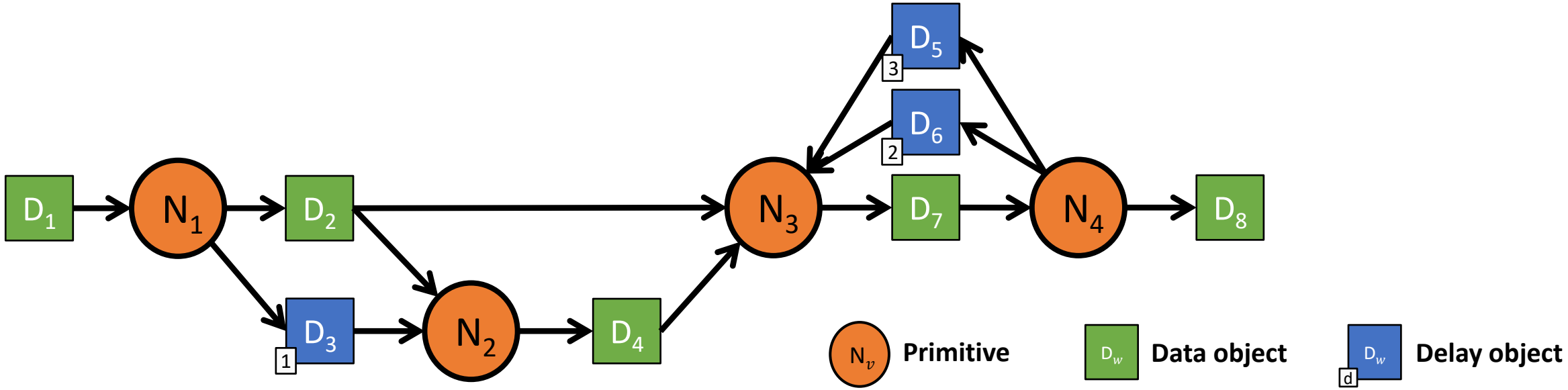
M. Yang et al.
RTSS '18

Before describing how we modify this transformation, we will overview the existing steps.

Starting point: OpenVX

OpenVX is a standard for specifying computer vision applications

- graph-based: primitives act on data objects (regular or delay)
- delay objects represent historical dependencies

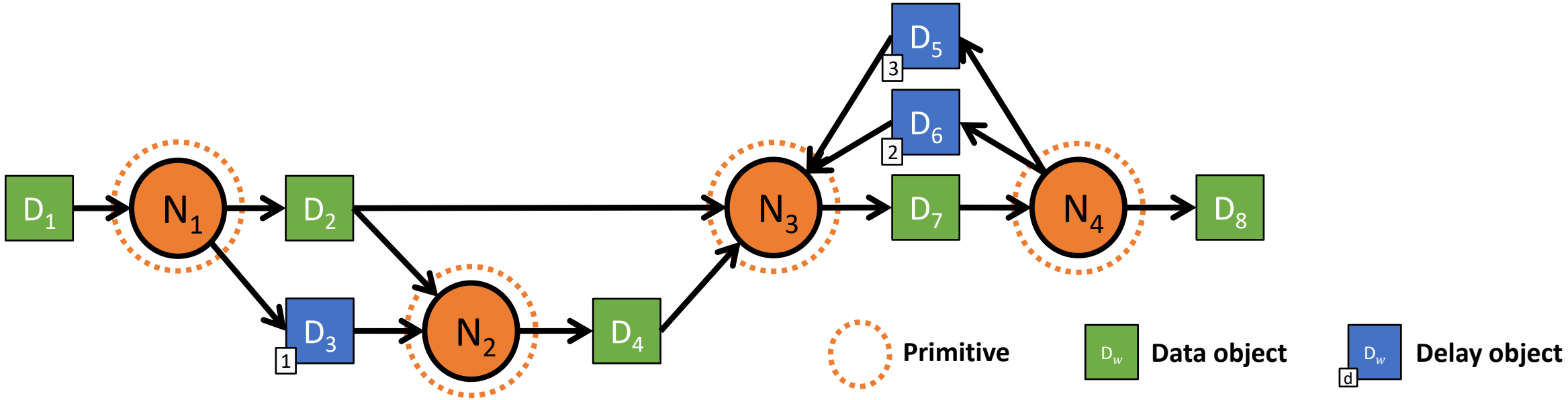


Step 1: Coarse- to Fine-Grained OpenVX

[M. Yang et al., RTSS '18]

OpenVX primitives can be too coarsely grained

– idea: break primitives into CPU and GPU nodes

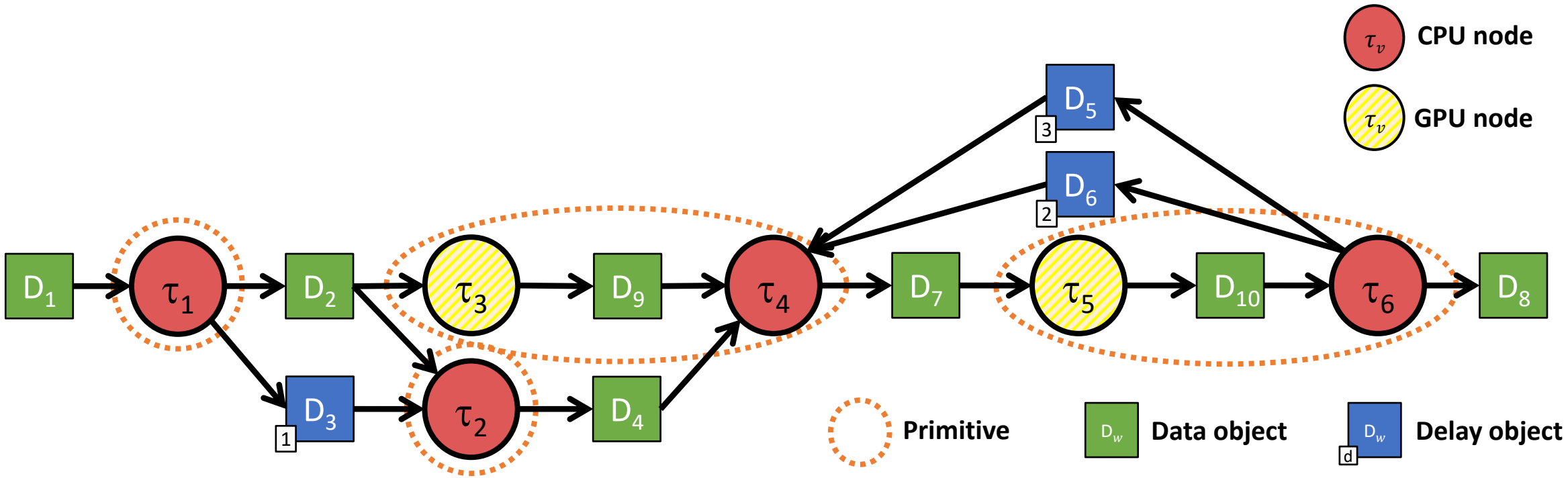


Step 1: Coarse- to Fine-Grained OpenVX

[M. Yang et al., RTSS '18]

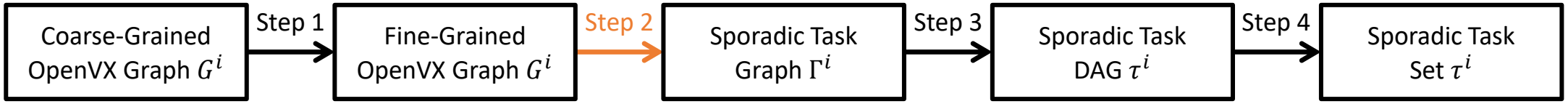
OpenVX primitives can be too coarsely grained

– idea: break primitives into CPU and GPU nodes



Transformation from graph to tasks

Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.



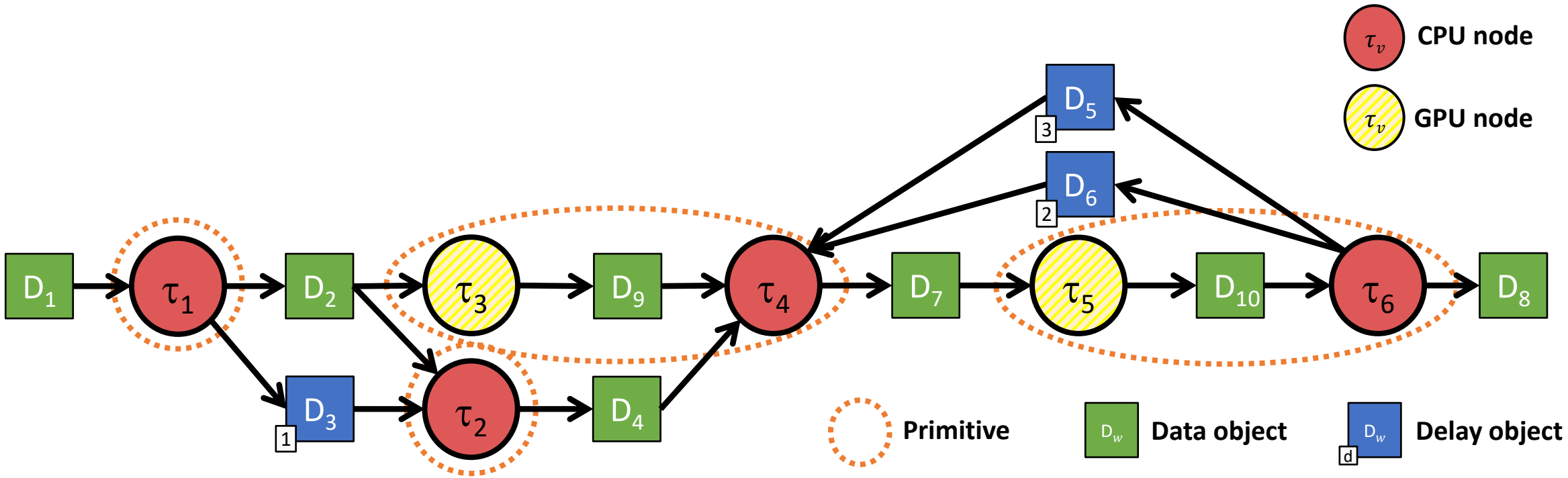
M. Yang et al.
RTSS '18

K. Yang et al.
RTNS '15

Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph Γ

[K. Yang et al., RTNS '15]

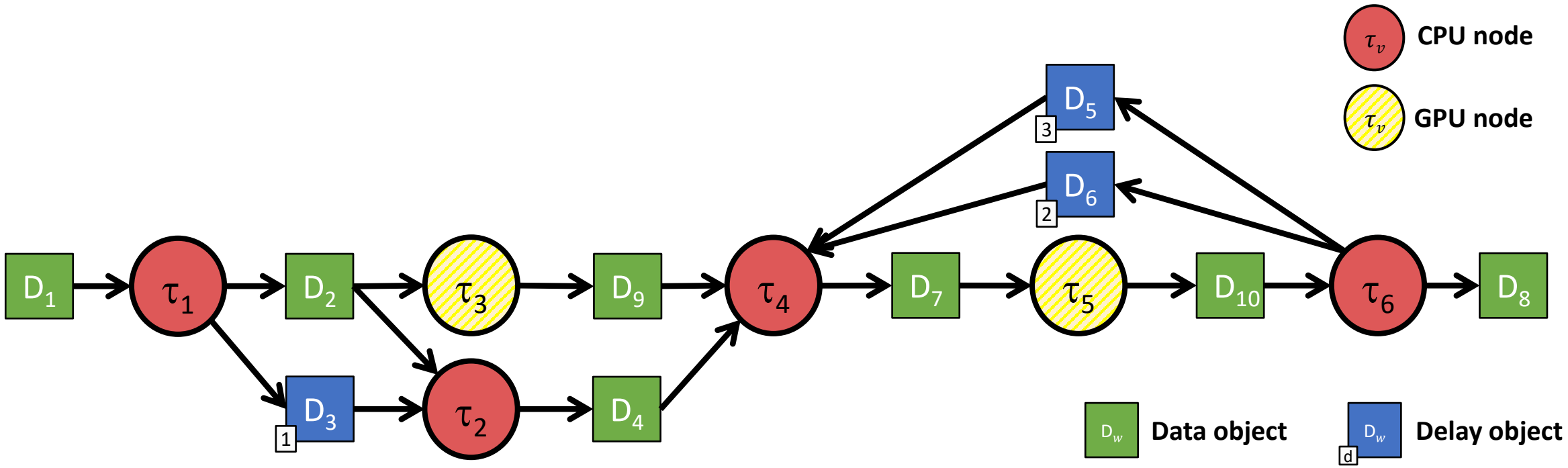
- One-to-one mapping of nodes



Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph Γ

[K. Yang et al., RTNS '15]

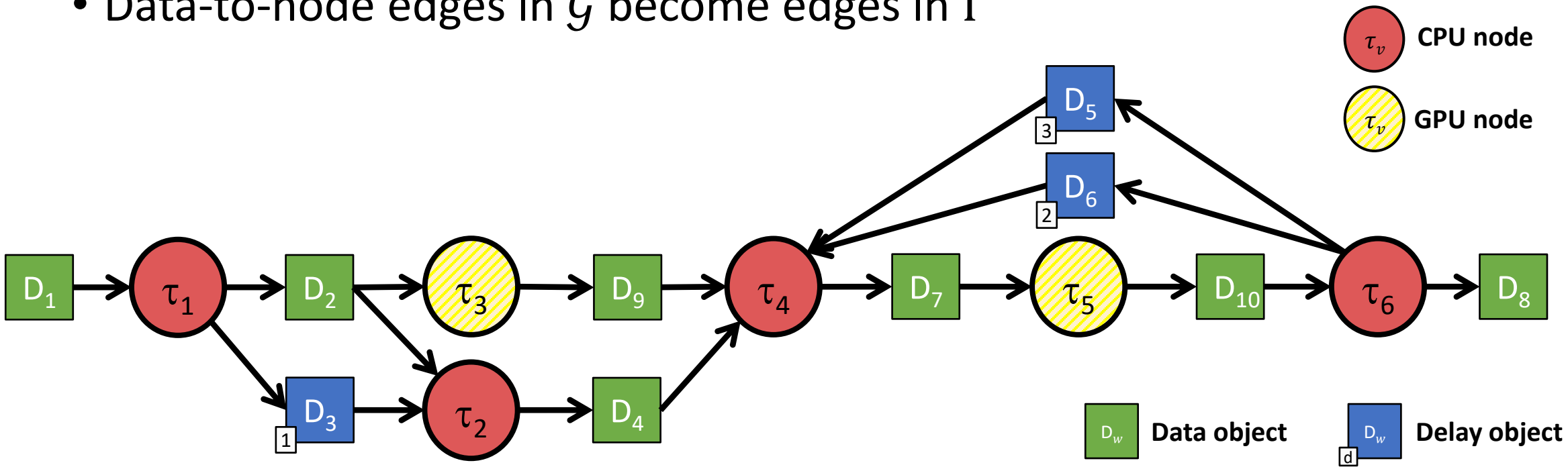
- One-to-one mapping of nodes



Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph Γ

[K. Yang et al., RTNS '15]

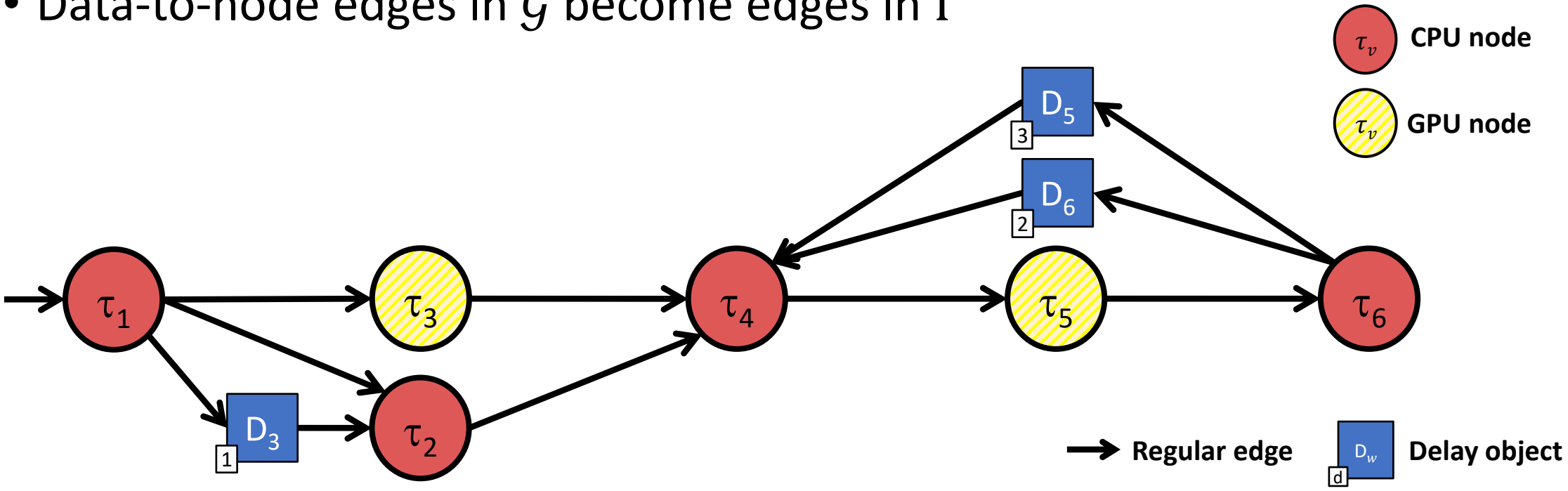
- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in Γ



Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph Γ

[K. Yang et al., RTNS '15]

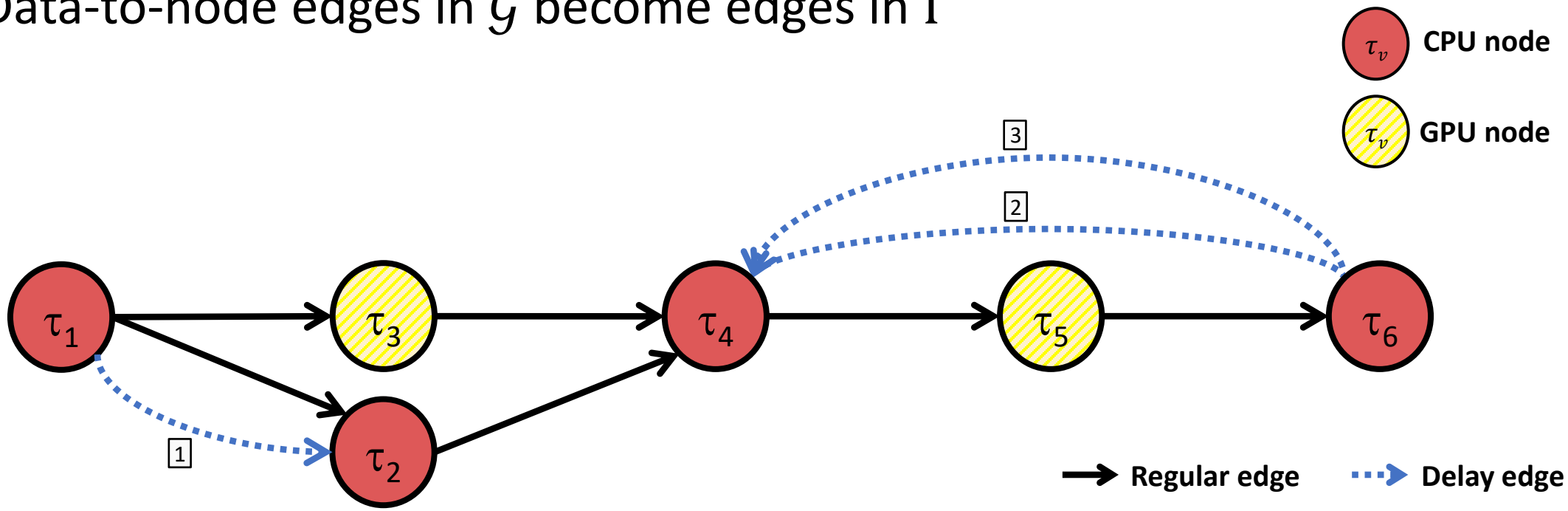
- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in Γ



Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph Γ

[K. Yang et al., RTNS '15]

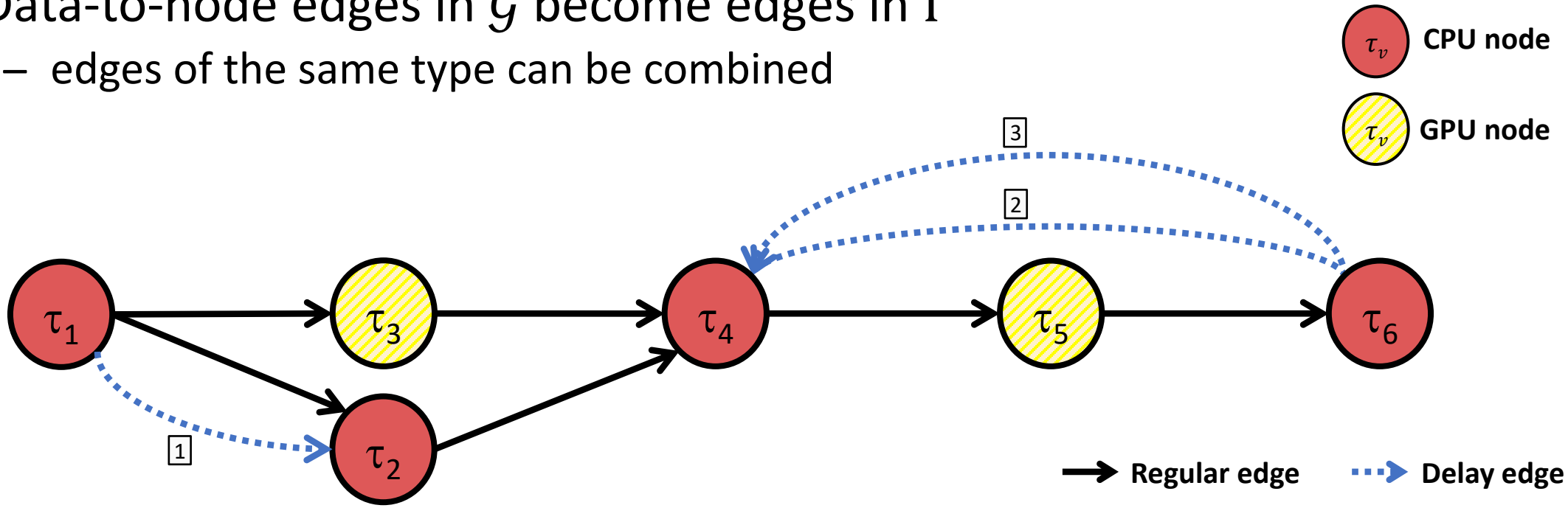
- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in Γ



Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph Γ

[K. Yang et al., RTNS '15]

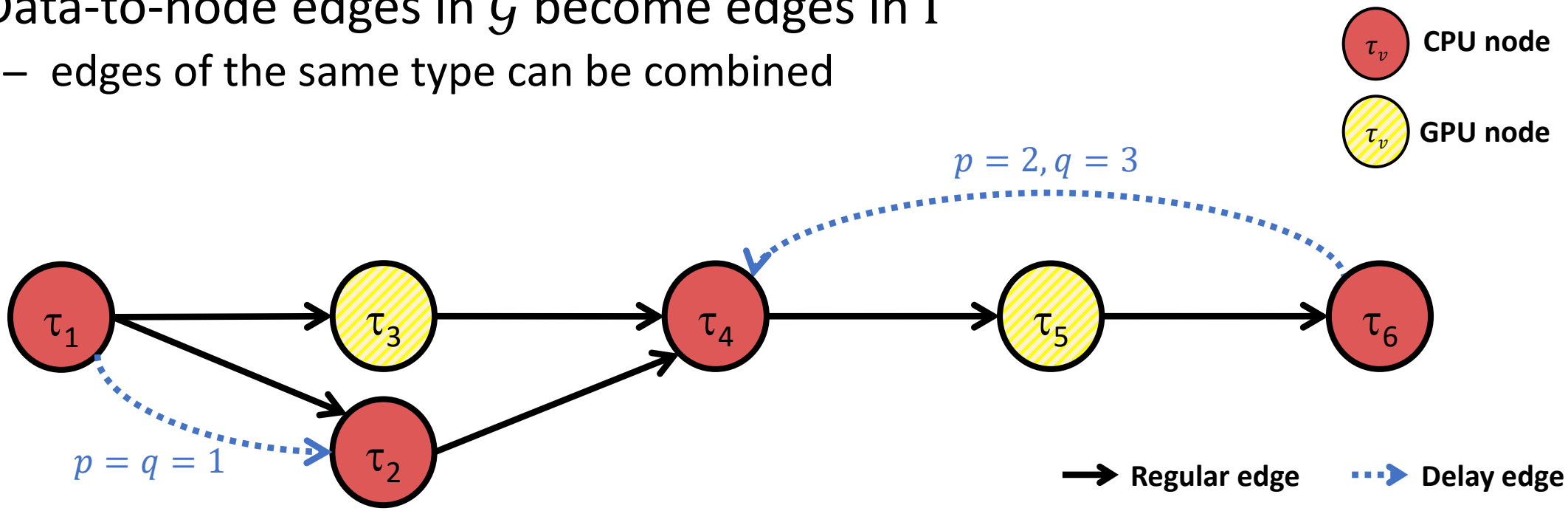
- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in Γ
  - edges of the same type can be combined



Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph Γ

[K. Yang et al., RTNS '15]

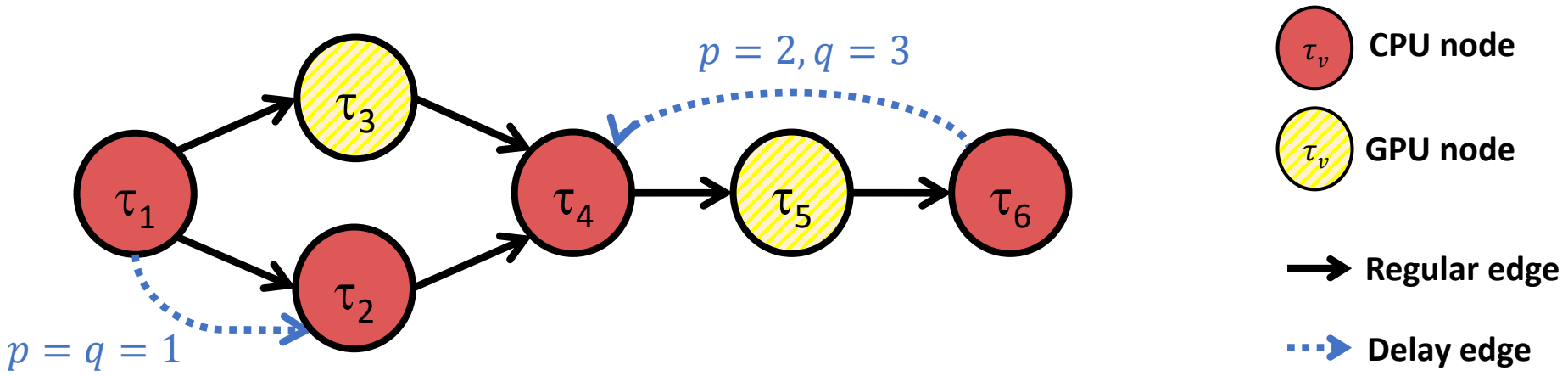
- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in Γ
  - edges of the same type can be combined



Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph Γ

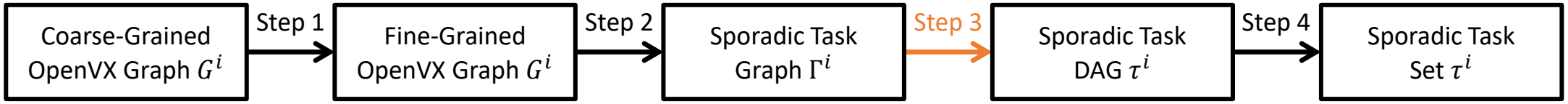
[K. Yang et al., RTNS '15]

- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in Γ
  - edges of the same type can be combined



Transformation from graph to tasks

Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.



M. Yang et al. RTSS '18

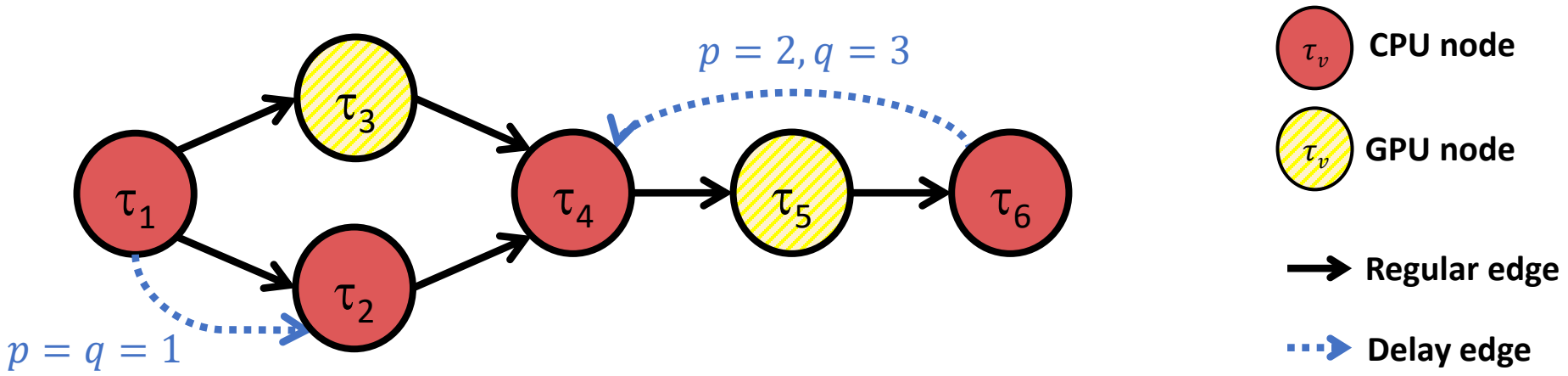
K. Yang et al. RTNS '15

K. Yang et al. RTNS '15

Step 3: Sporadic task graph Γ to sporadic task DAG τ

[K. Yang et al., RTNS '15]

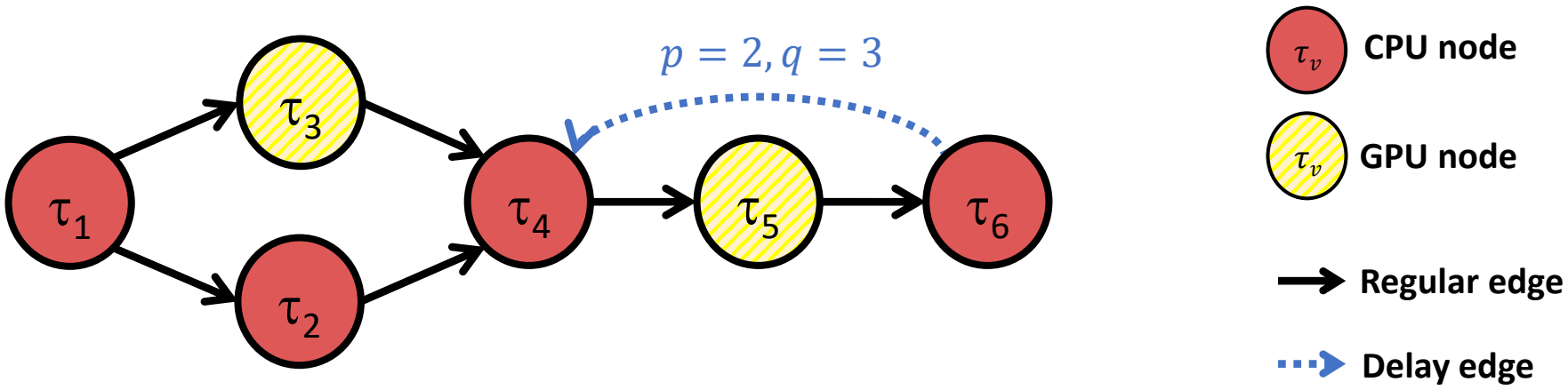
- Forward delay edges can be dropped



Step 3: Sporadic task graph Γ to sporadic task DAG τ

[K. Yang et al., RTNS '15]

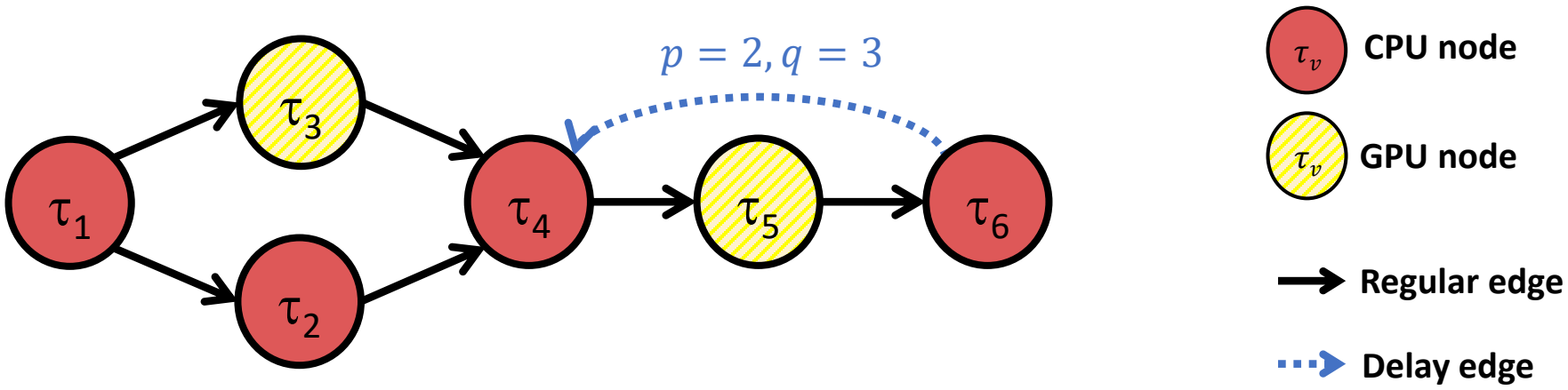
- Forward delay edges can be dropped



Step 3: Sporadic task graph Γ to sporadic task DAG τ

[K. Yang et al., RTNS '15]

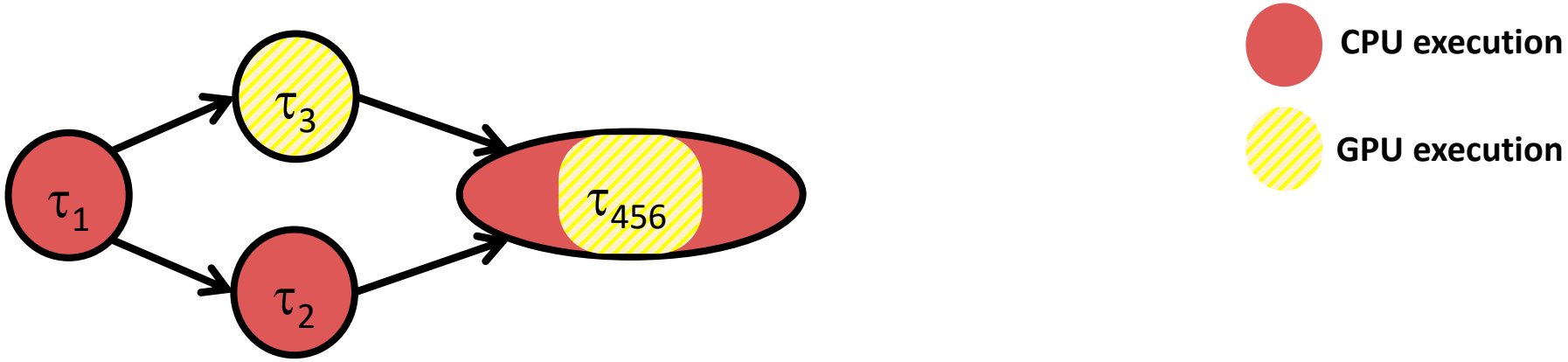
- Forward delay edges can be dropped
- Cycles (due to backward delay edges) are combined into supernodes



Step 3: Sporadic task graph Γ to sporadic task DAG τ

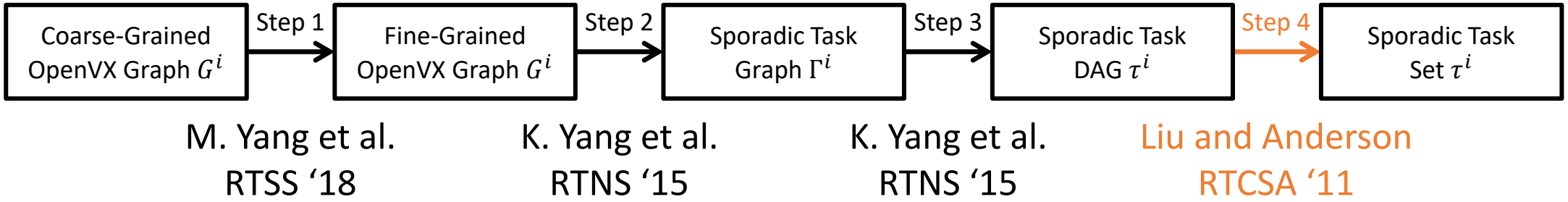
[K. Yang et al., RTNS '15]

- Forward delay edges can be dropped
- Cycles (due to backward delay edges) are combined into supernodes



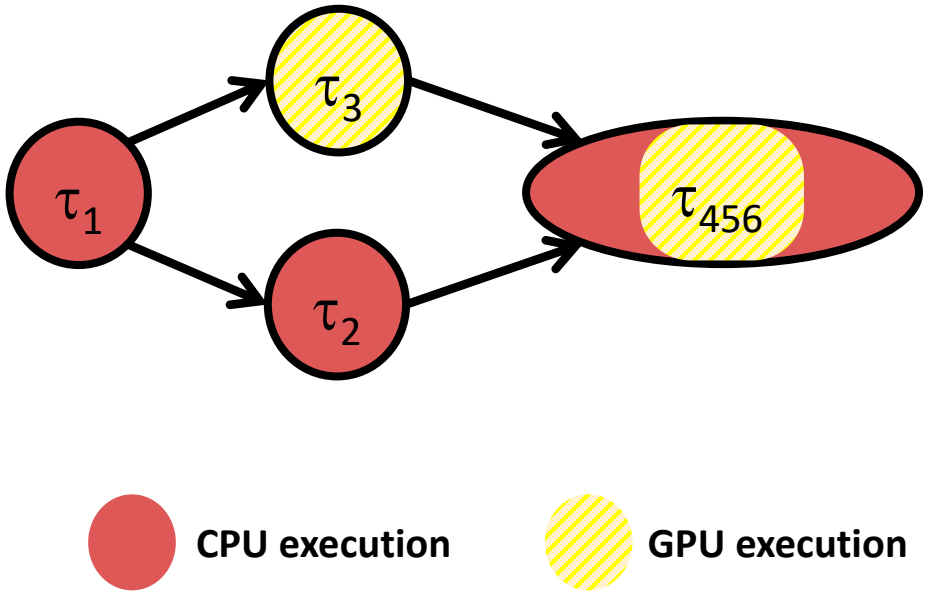
Transformation from graph to tasks

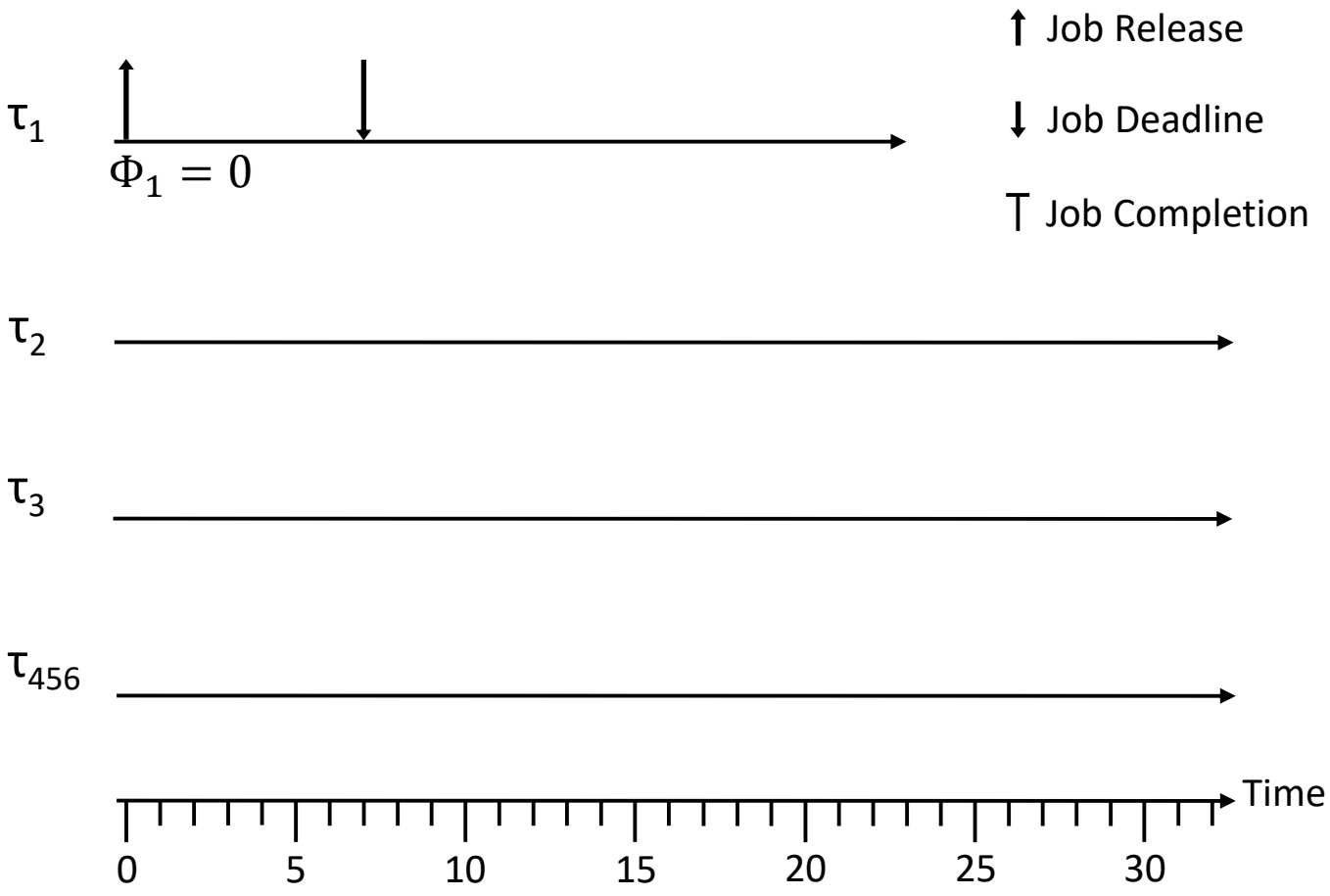
Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.



Step 4: Sporadic task DAG τ to independent sporadic tasks

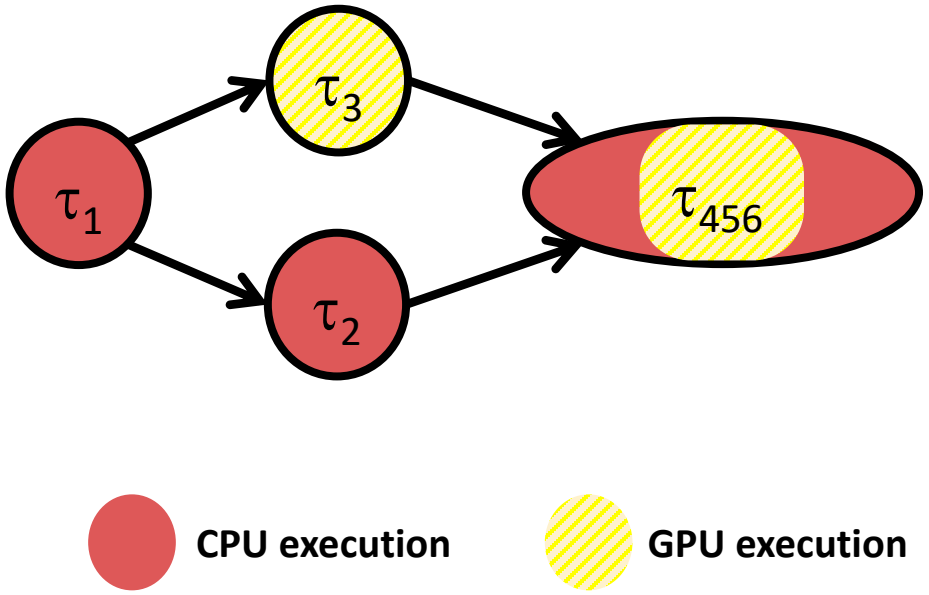
[Liu and Anderson, RTCSA 2011]

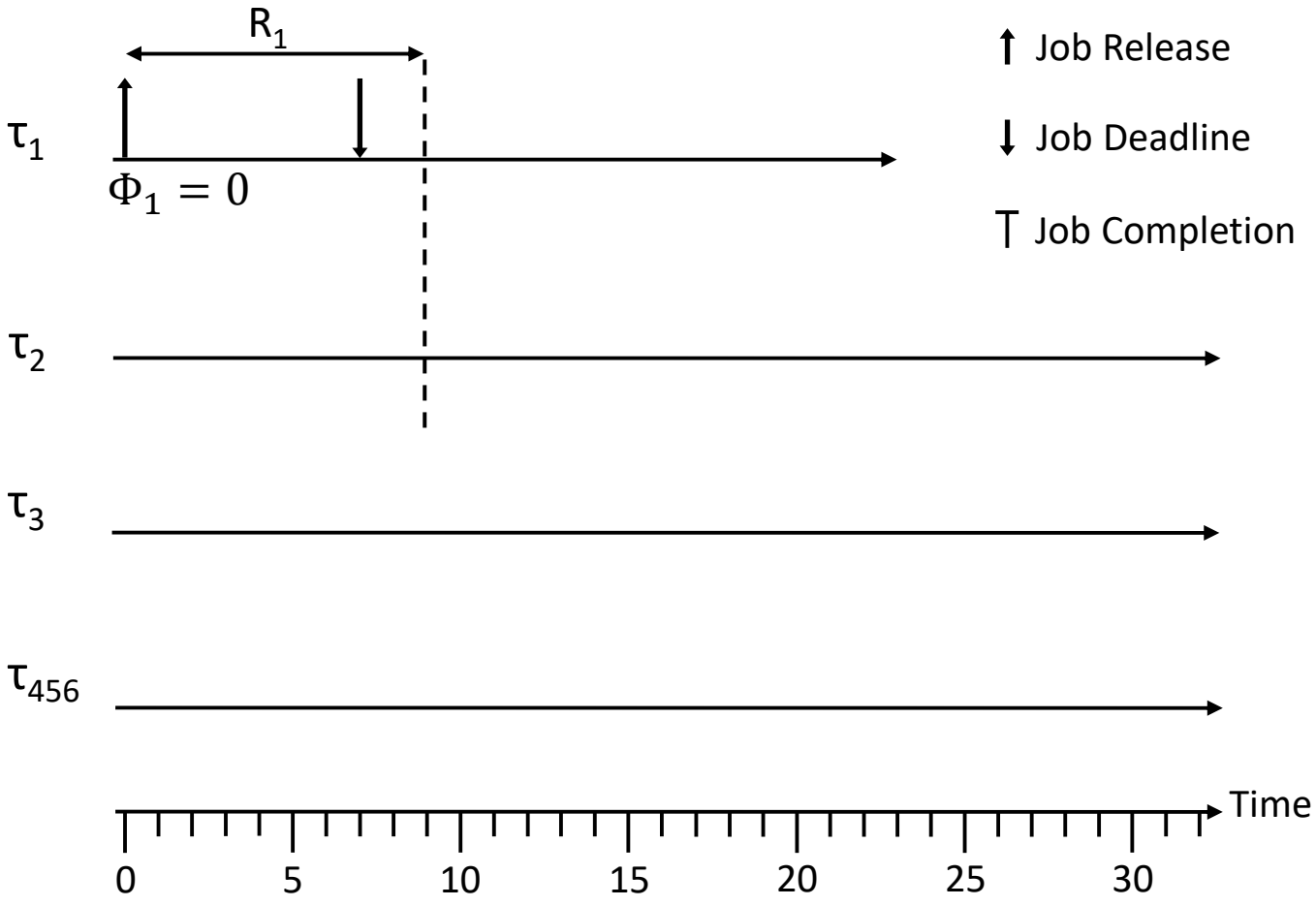




Step 4: Sporadic task DAG τ to independent sporadic tasks

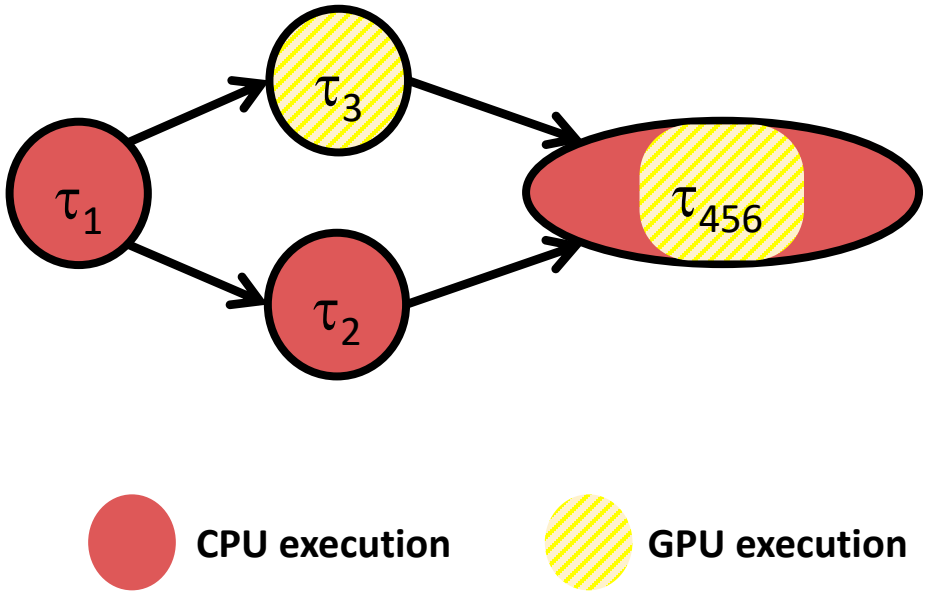
[Liu and Anderson, RTCSA 2011]

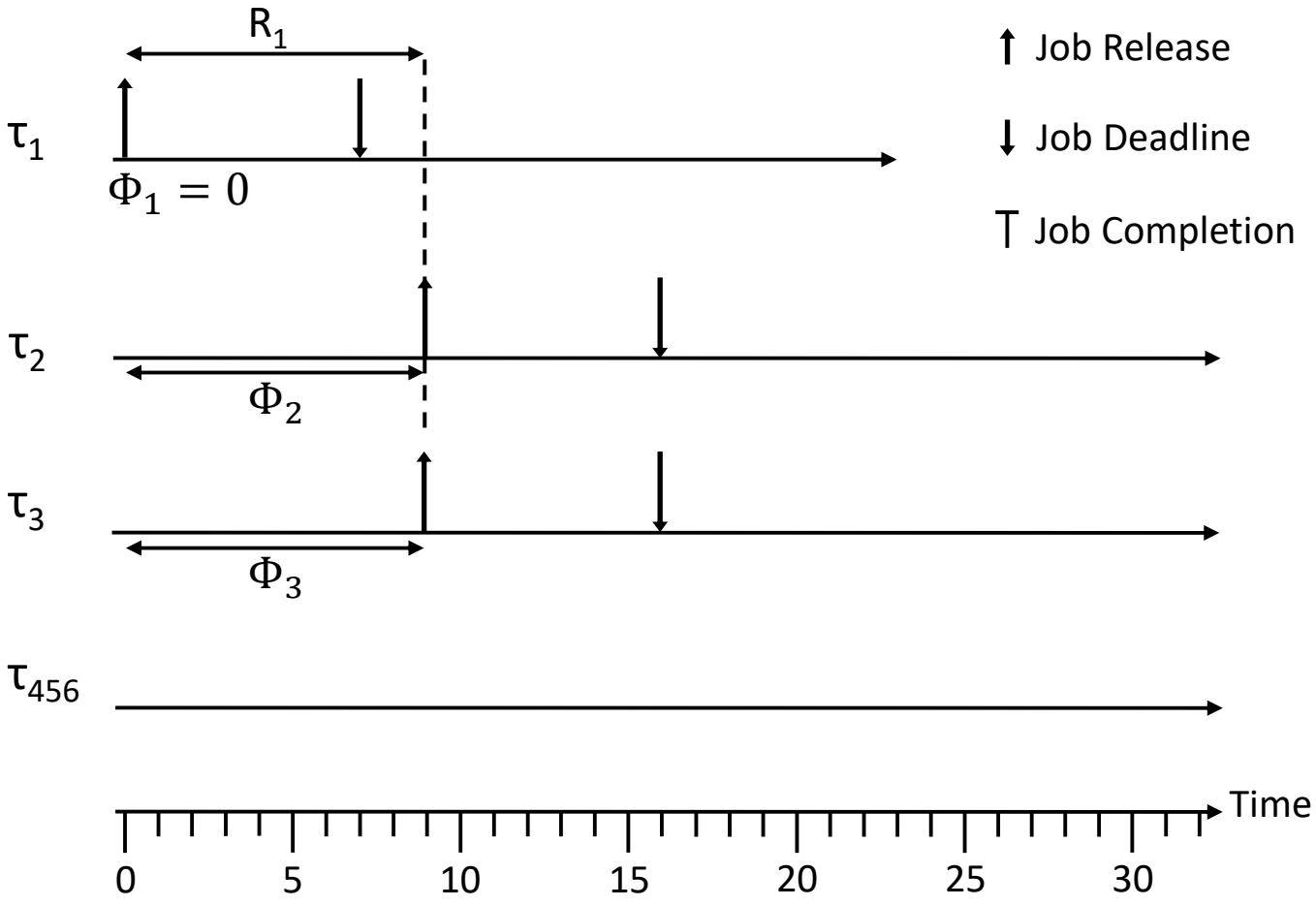




Step 4: Sporadic task DAG τ to independent sporadic tasks

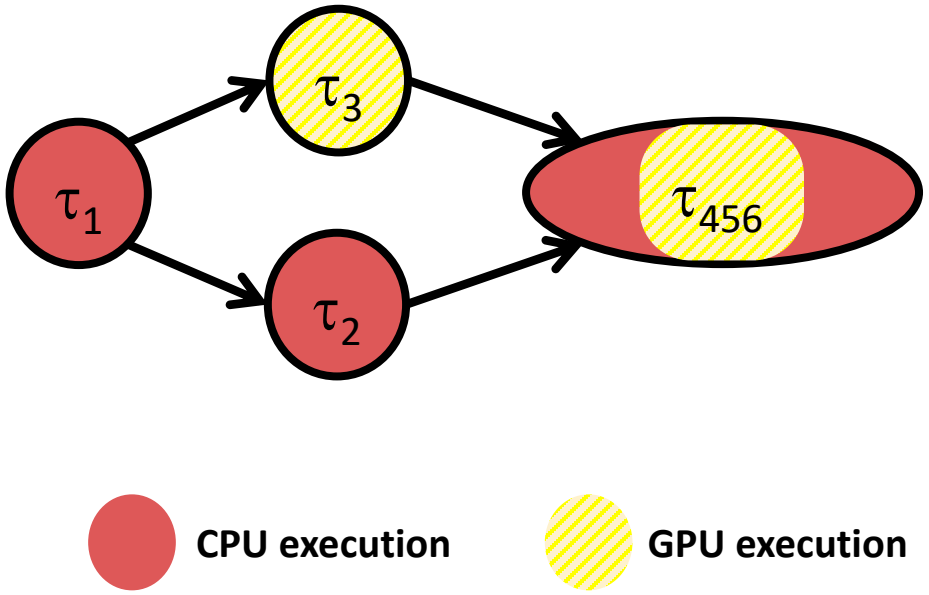
[Liu and Anderson, RTCSA 2011]

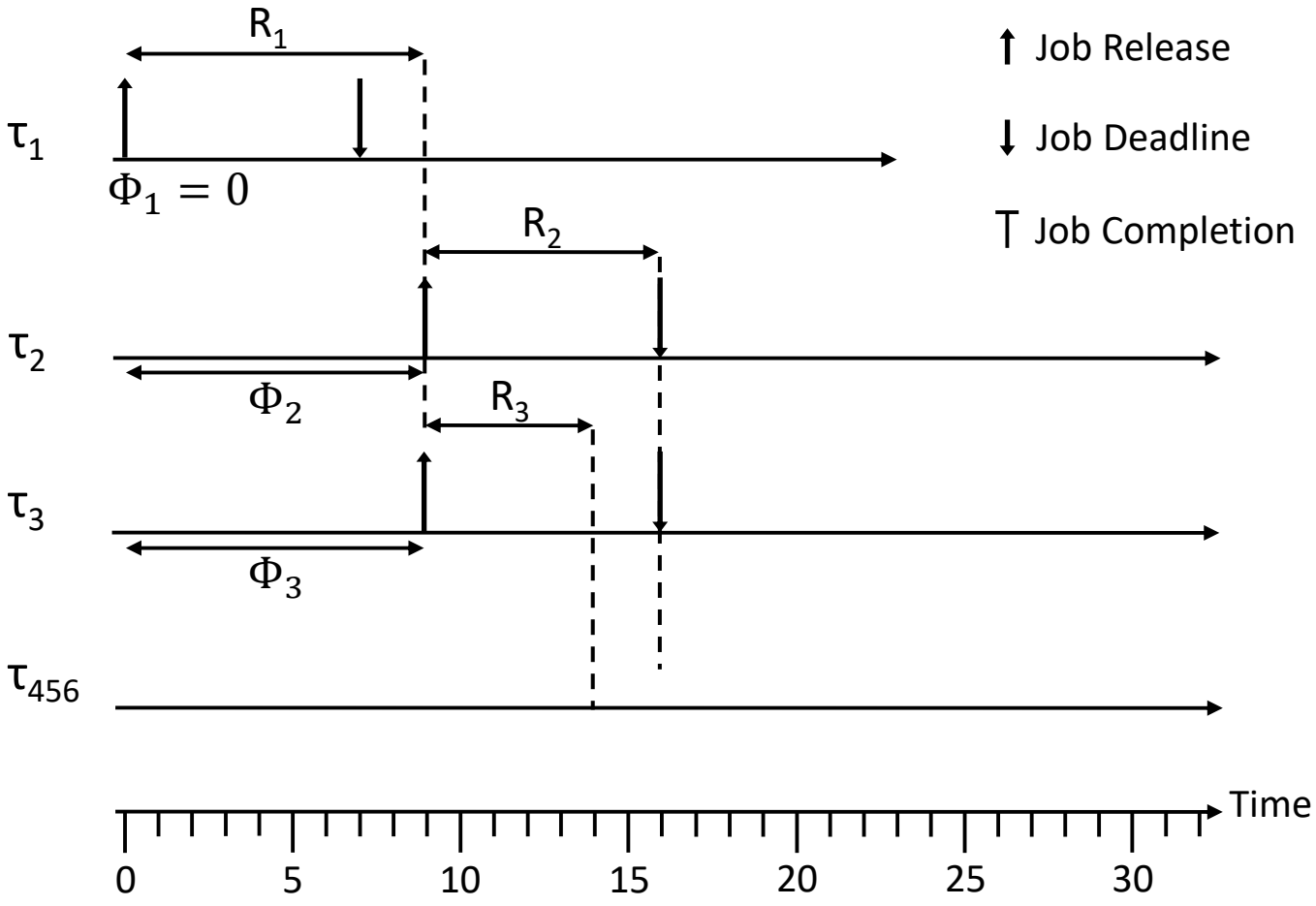




Step 4: Sporadic task DAG τ to independent sporadic tasks

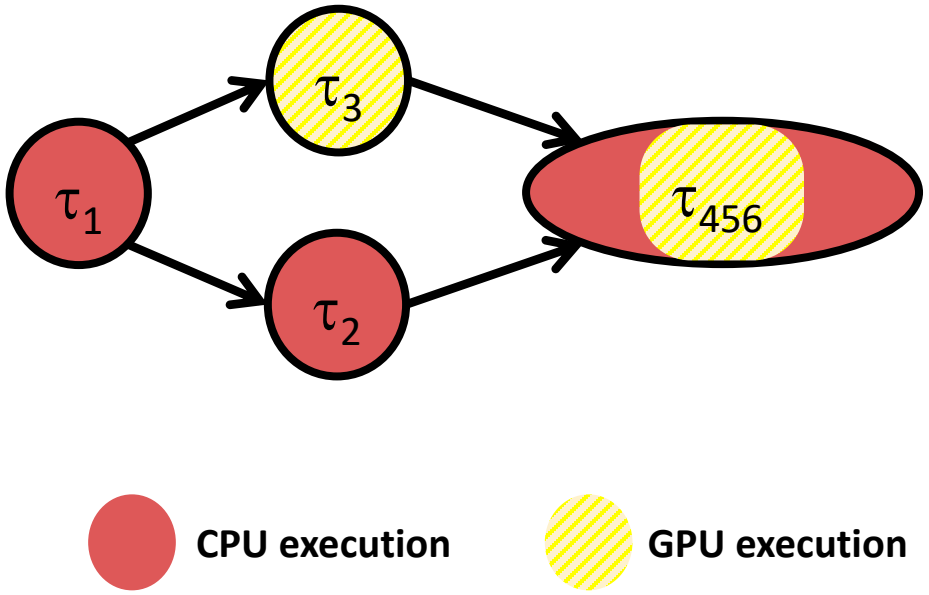
[Liu and Anderson, RTCSA 2011]

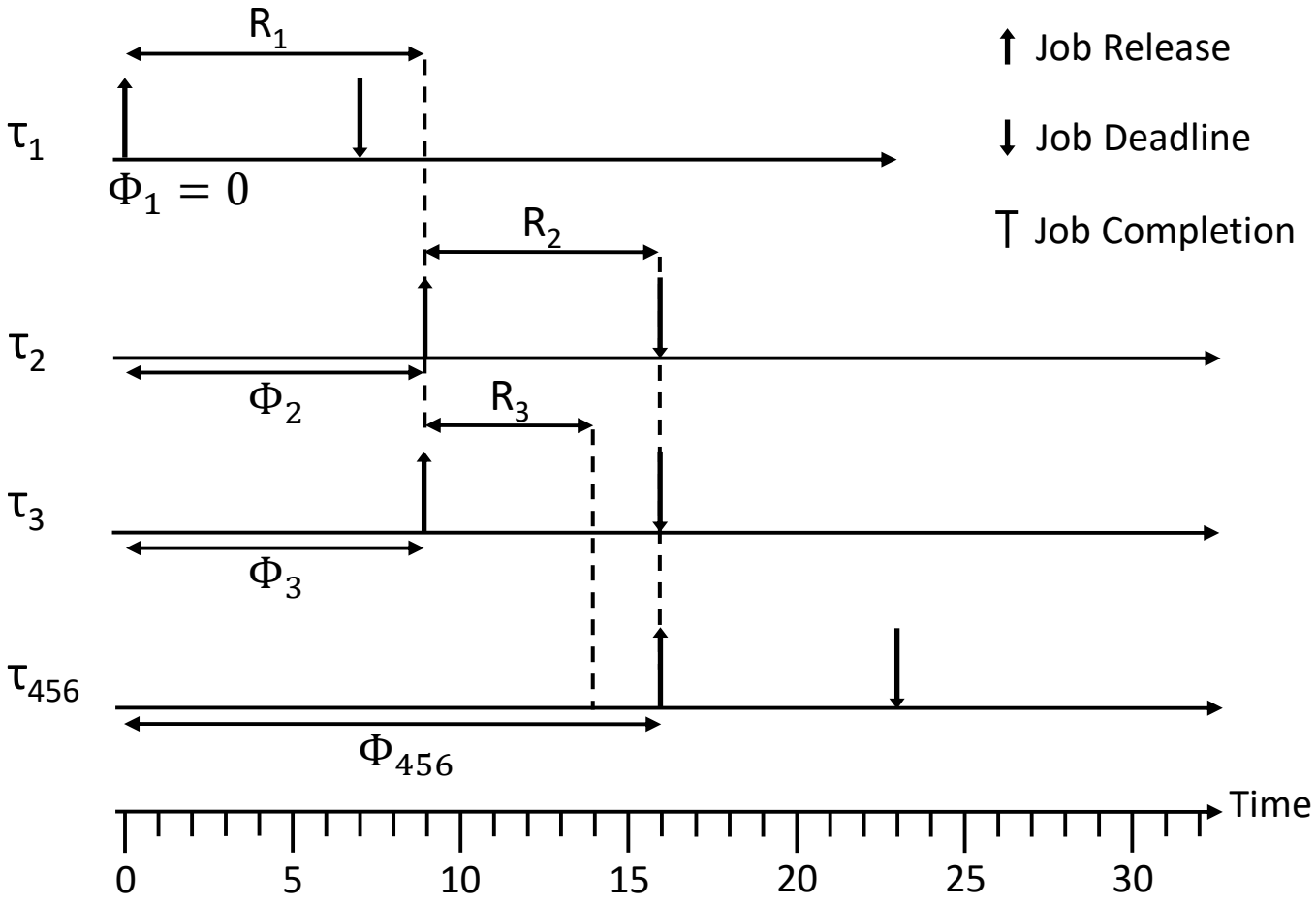




Step 4: Sporadic task DAG τ to independent sporadic tasks

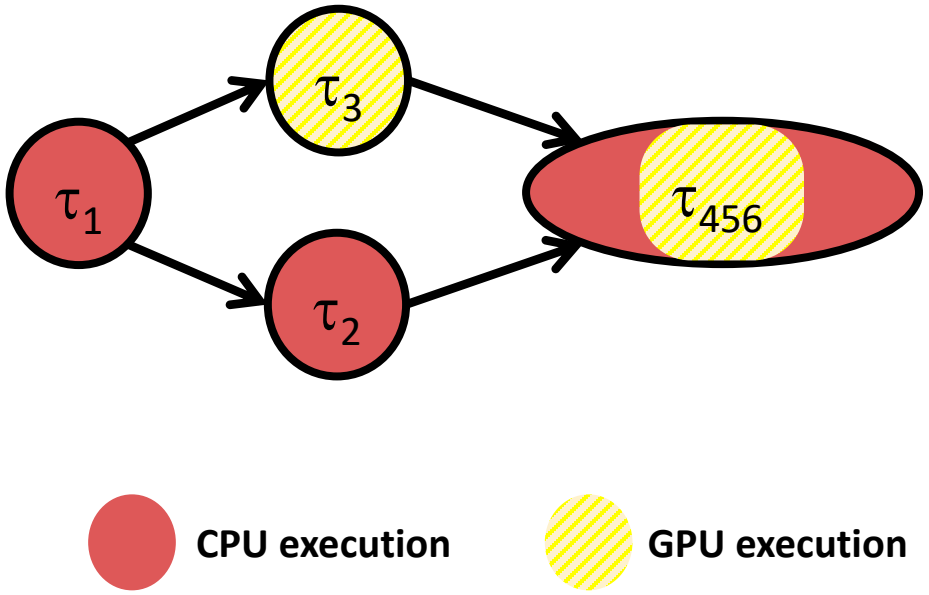
[Liu and Anderson, RTCSA 2011]

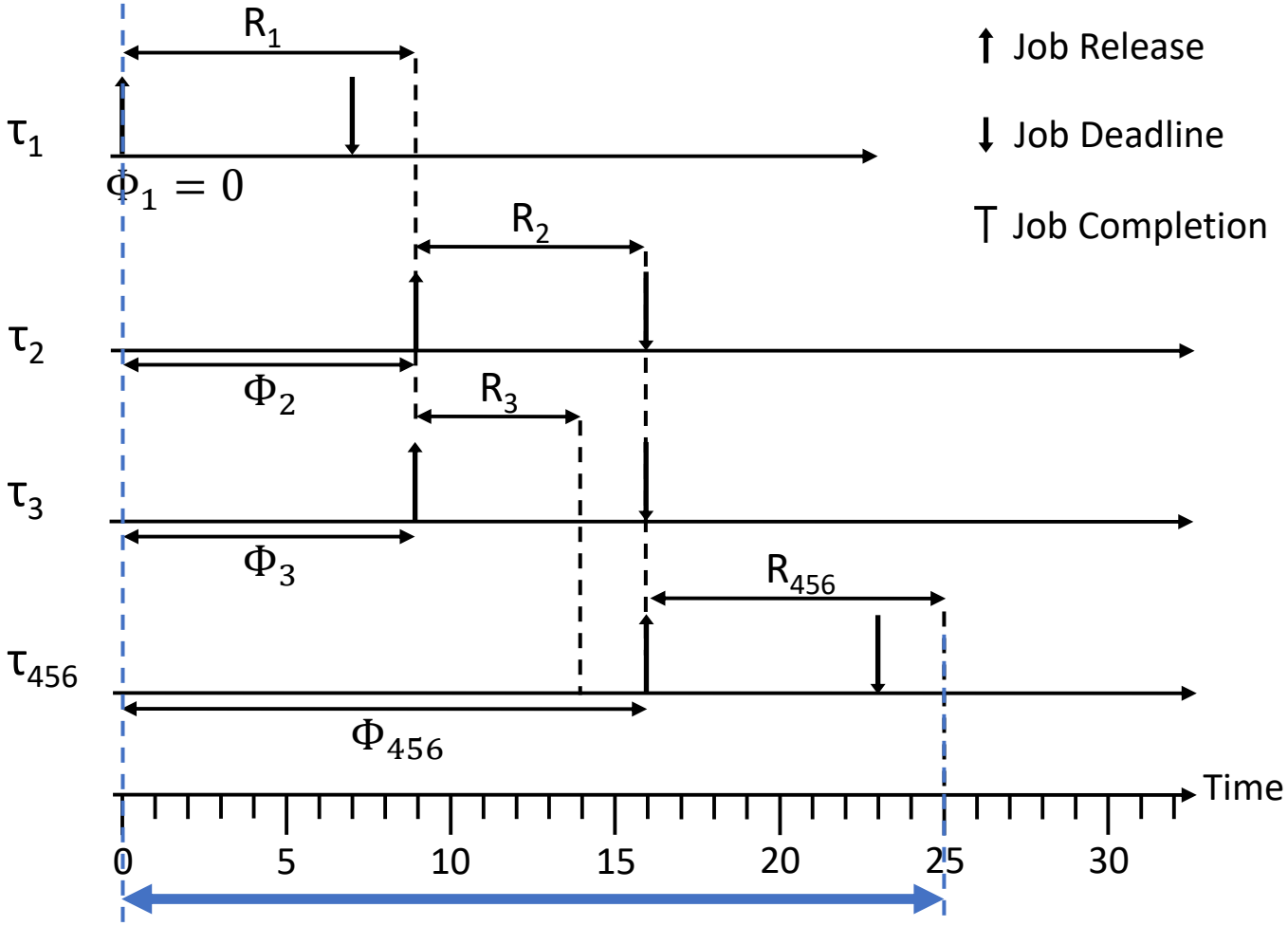




Step 4: Sporadic task DAG τ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]

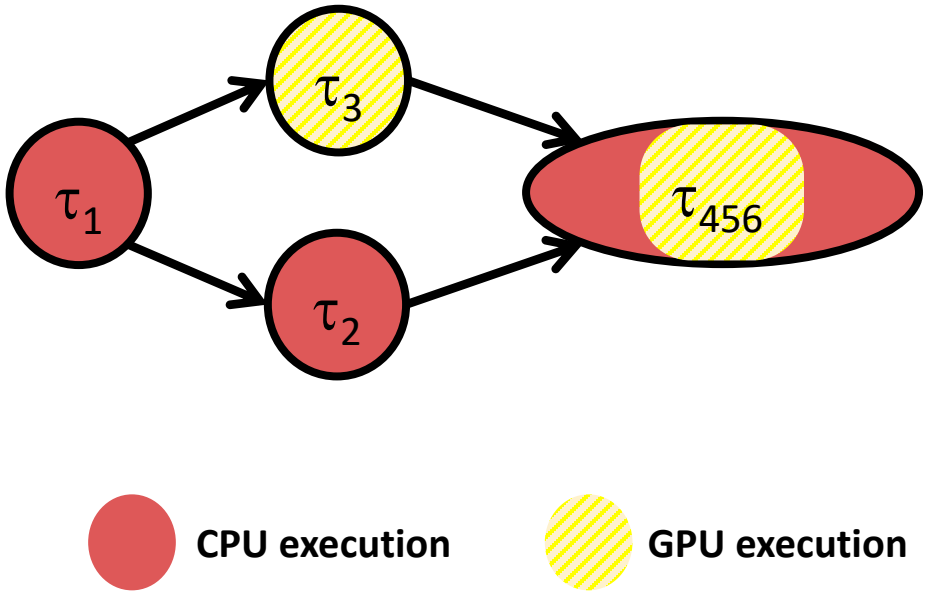


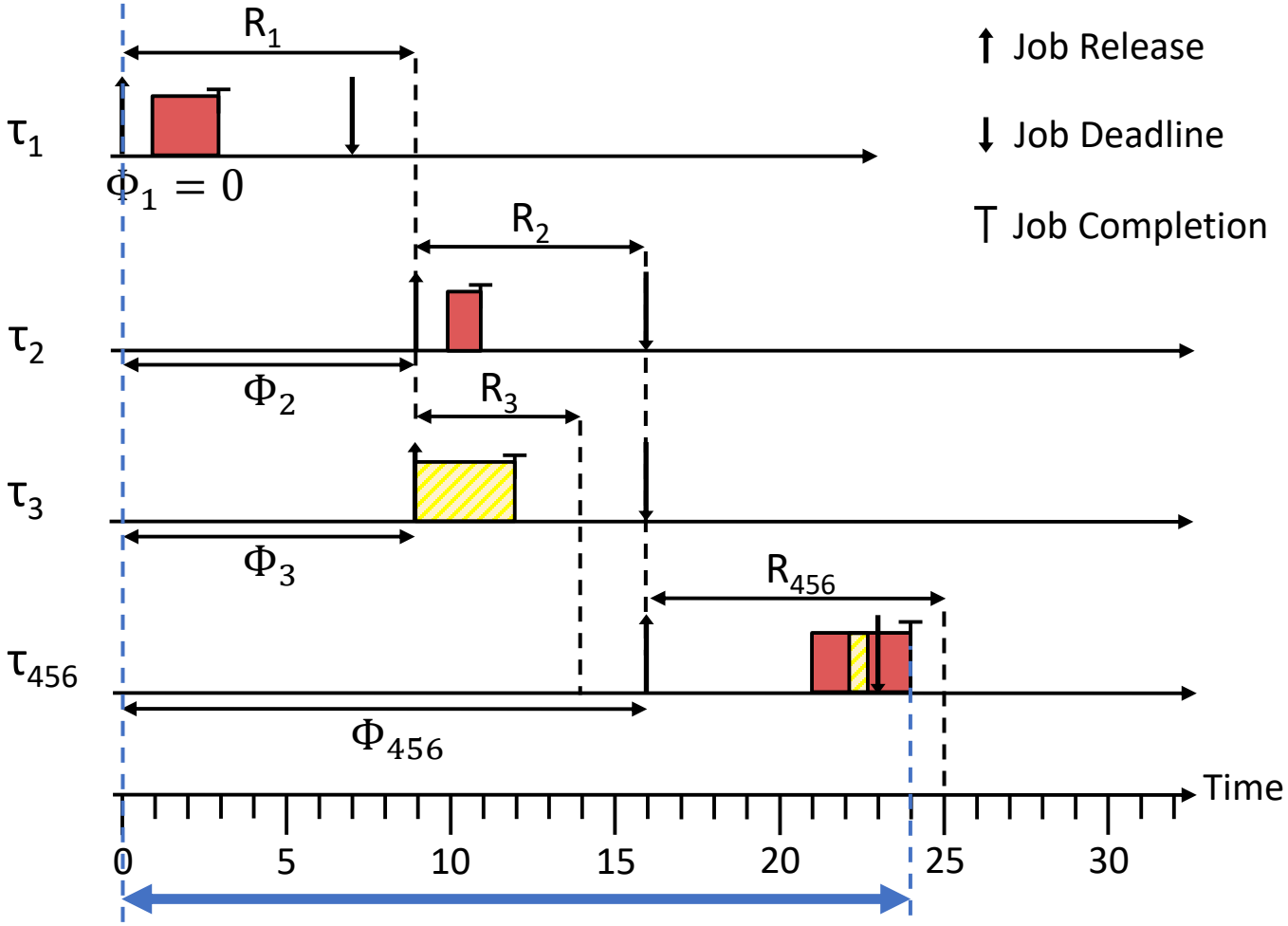


graph end-to-end response time bound: 25

Step 4: Sporadic task DAG τ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]

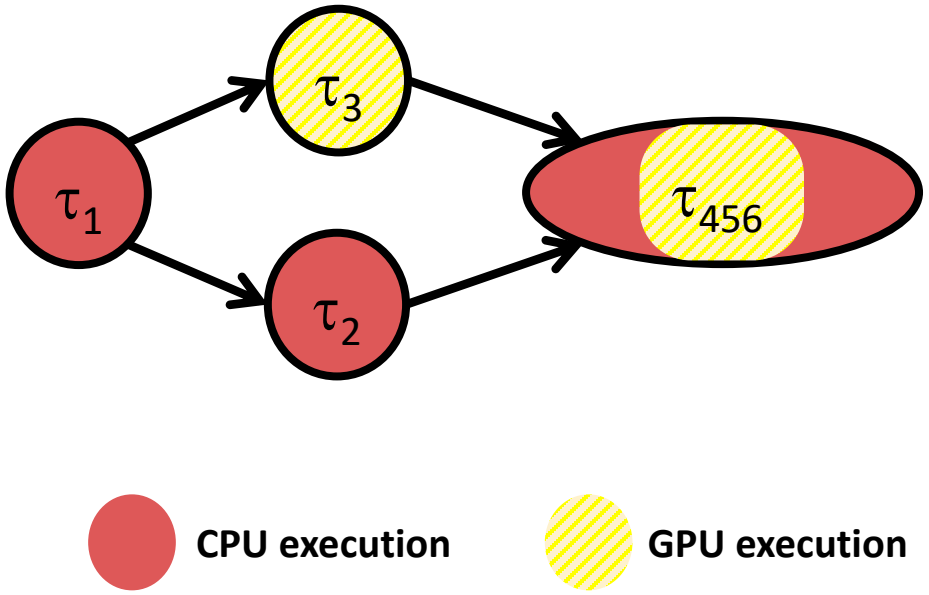


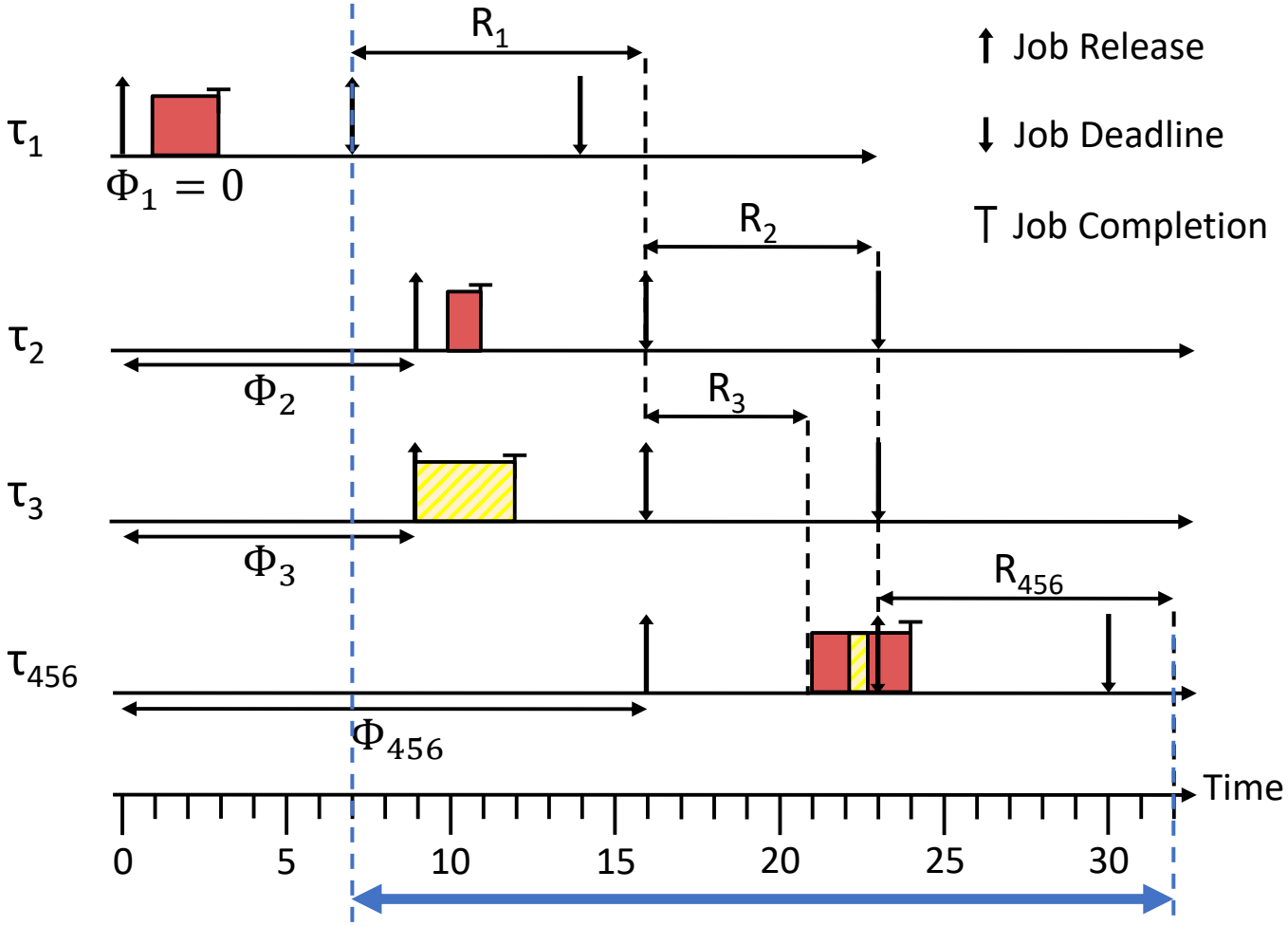


end-to-end response time of first invocation: 24

Step 4: Sporadic task DAG τ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]

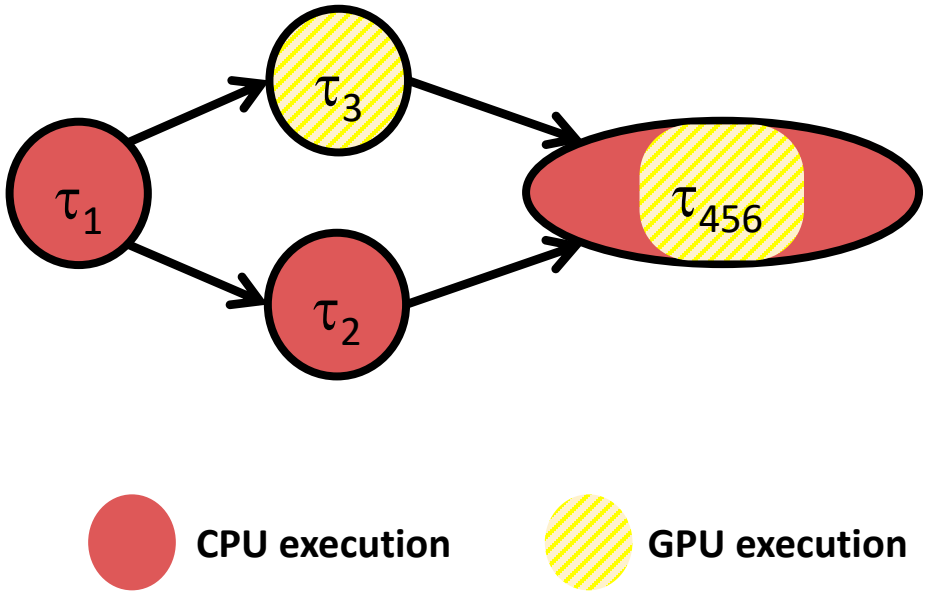


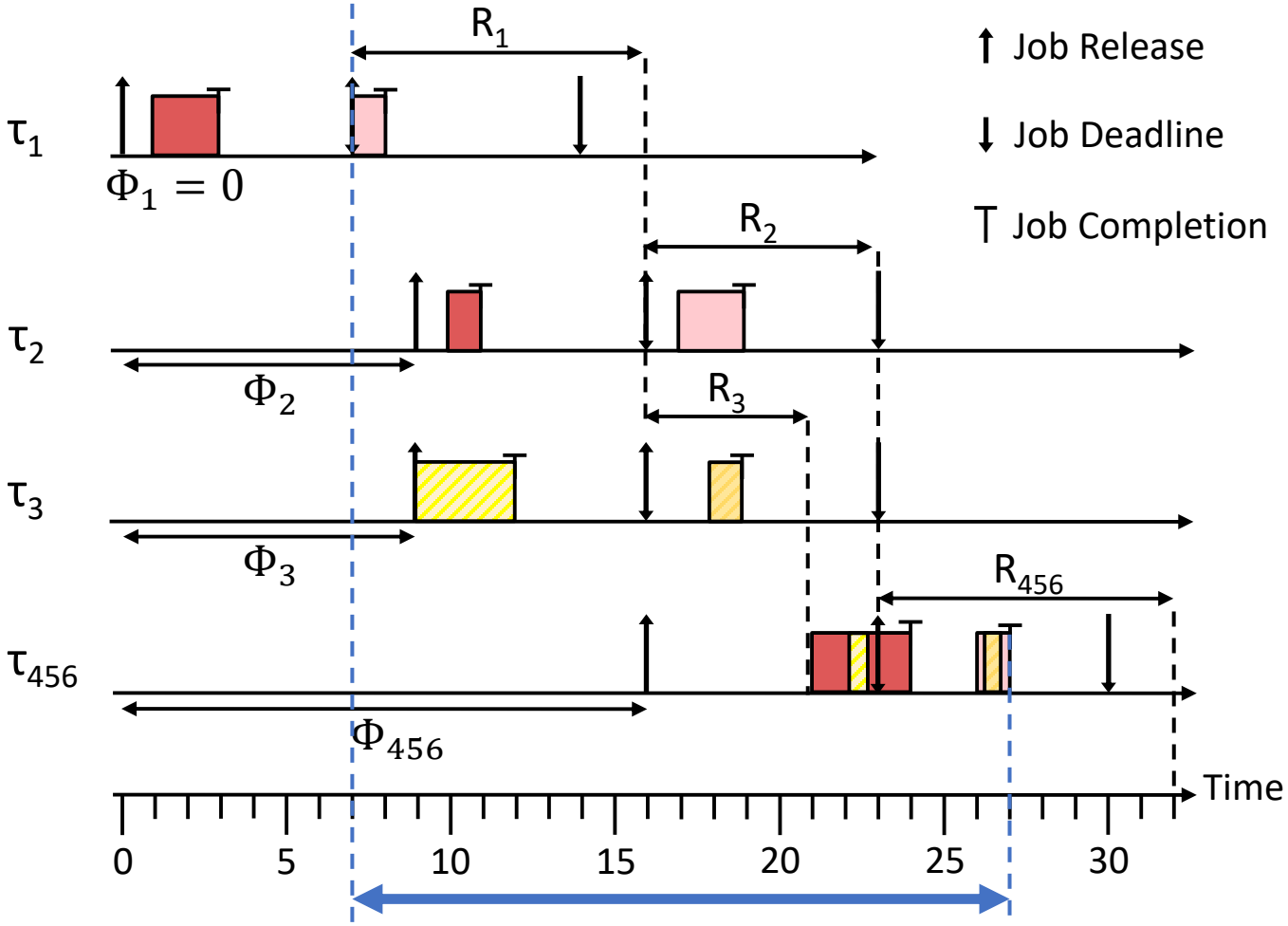


graph end-to-end response time bound: 25

Step 4: Sporadic task DAG τ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]



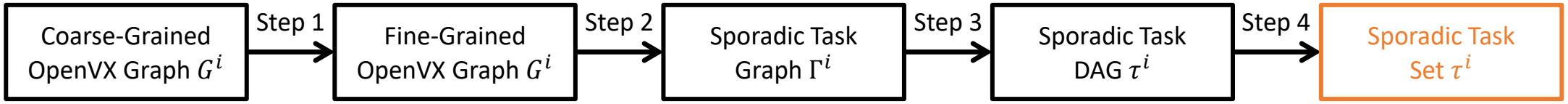


end-to-end response time of second invocation: 20

Transformation from graph to tasks

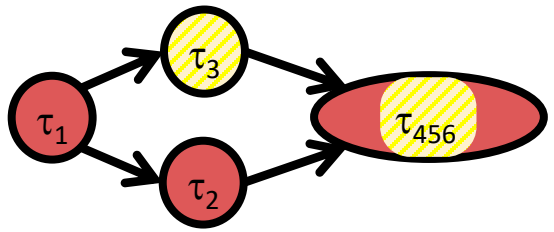
Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.

Choice of intra-task parallelism



M. Yang et al. RTSS '18 | K. Yang et al. RTNS '15 | K. Yang et al. RTNS '15 | Liu and Anderson RTCSA '11

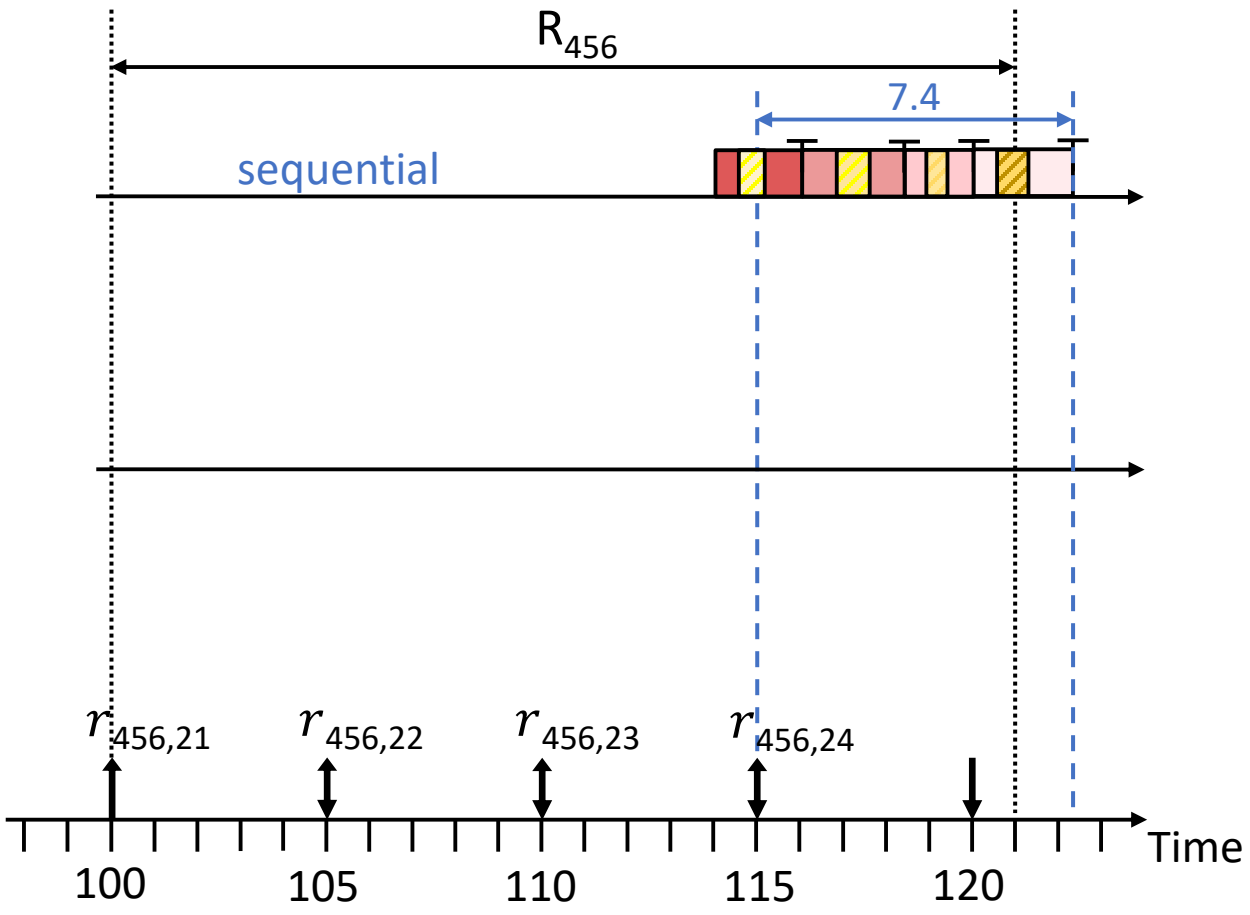
Choices of intra-task parallelism



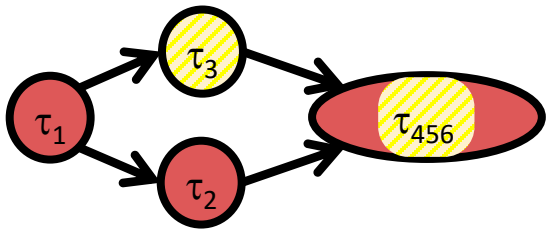
Let P_i be the intra-task parallelism allowed for task τ_i.

Sequential sporadic tasks assume no intra-task parallelism:

$$P_i = 1.$$



Choices of intra-task parallelism



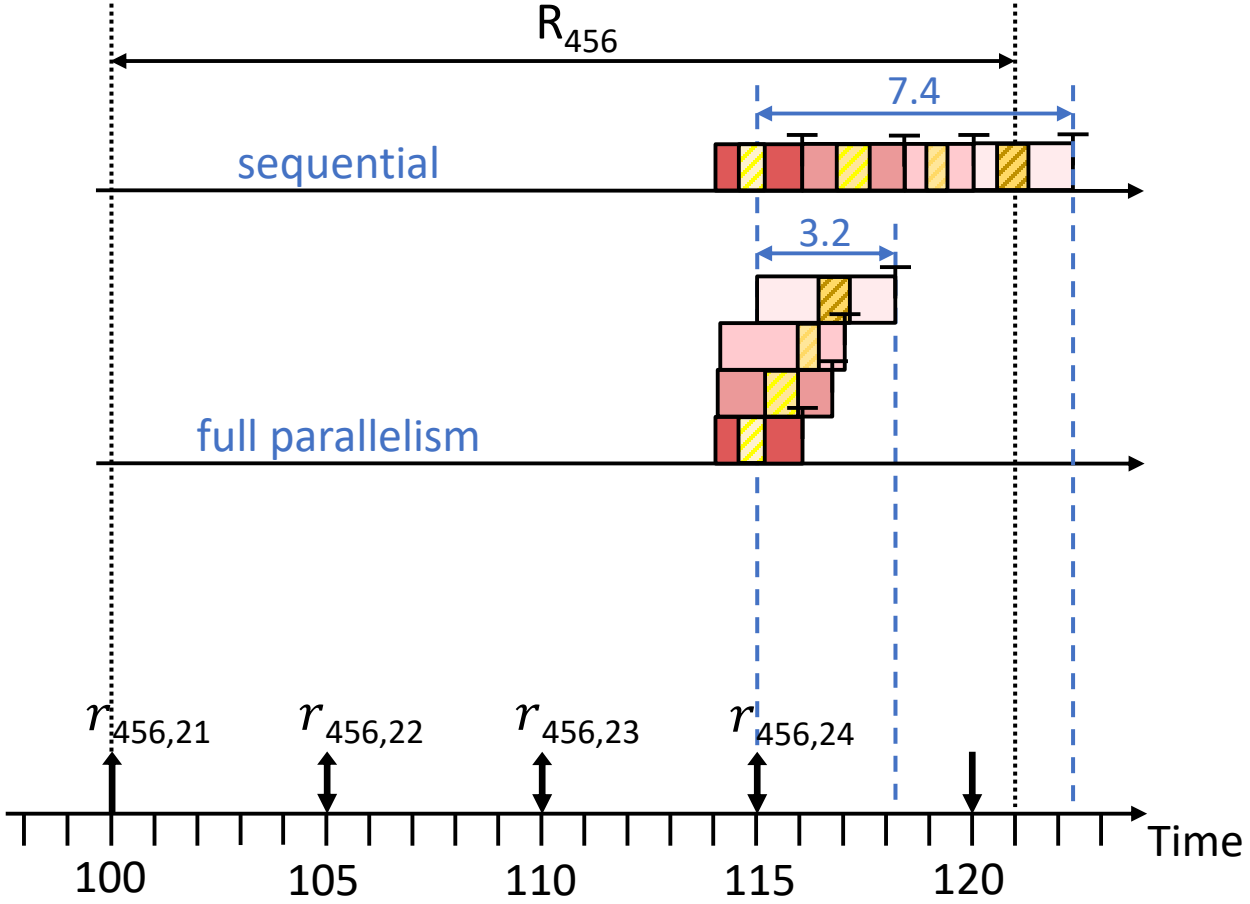
Let P_i be the intra-task parallelism allowed for task τ_i.

Sequential sporadic tasks assume no intra-task parallelism:

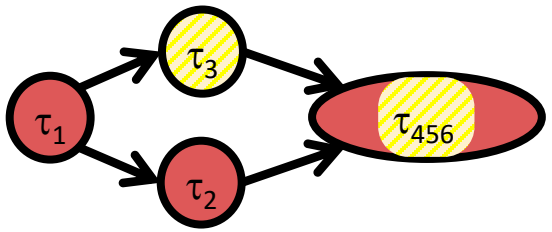
$$P_i = 1.$$

Parallel sporadic tasks assume full intra-task parallelism:

$$P_i = \#CPUs = m.$$



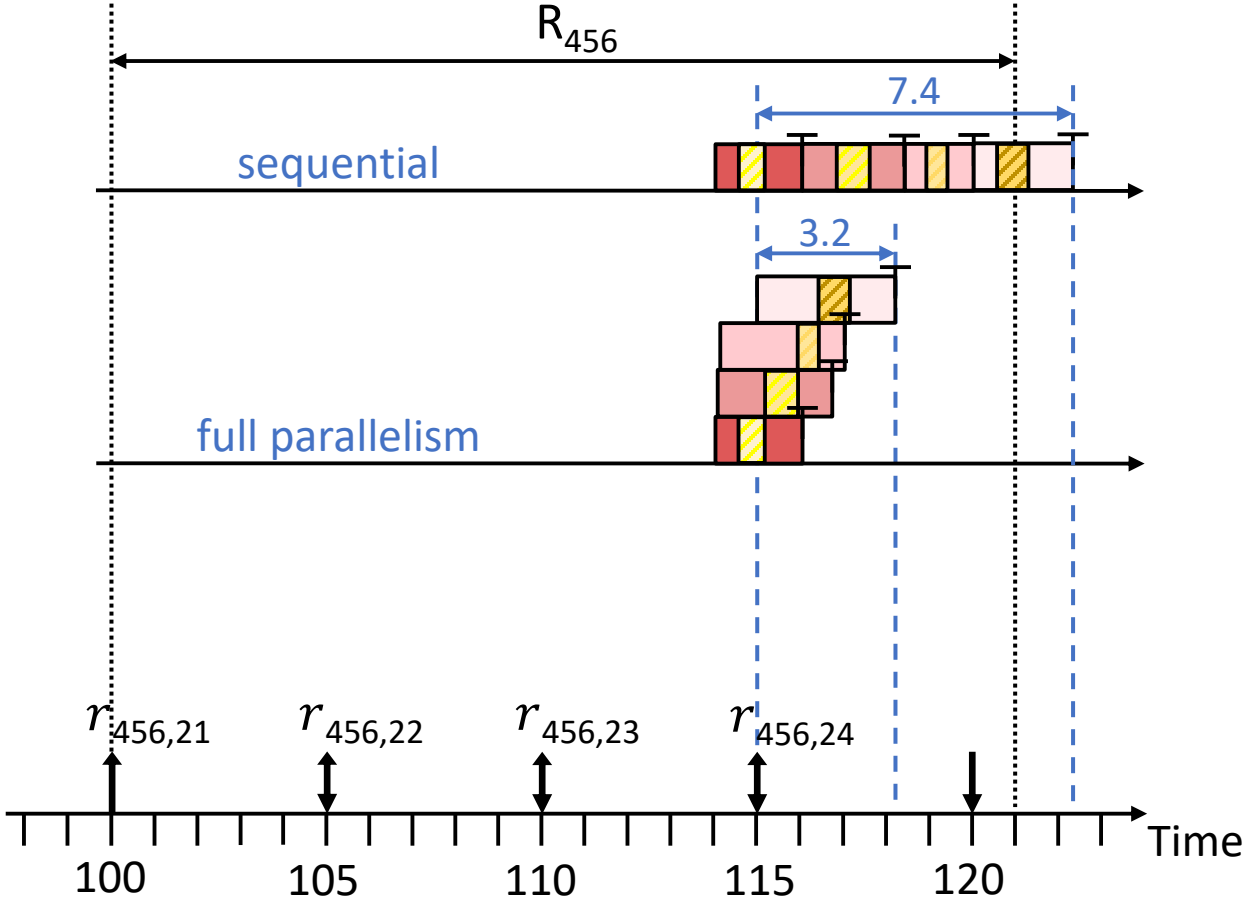
The problem



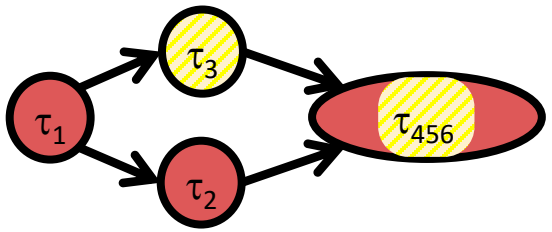
Let P_i be the intra-task parallelism allowed for task τ_i.

<u>Sequential:</u> if the cycle utilization is greater than 1.0, response times can be unbounded.

<u>Full parallelism:</u> precedence constraints (p of delay edge) may not be respected.



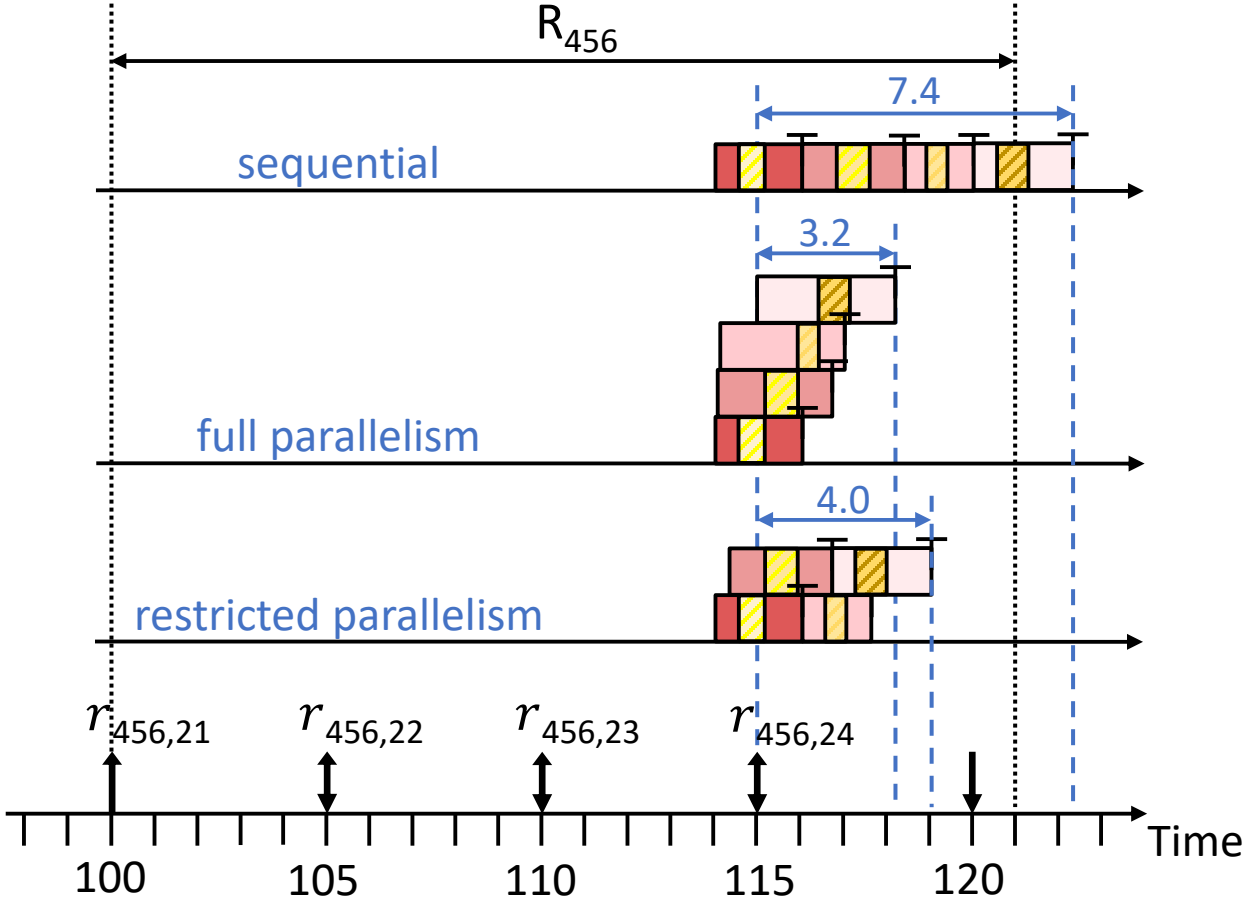
Contribution: restricted parallelism



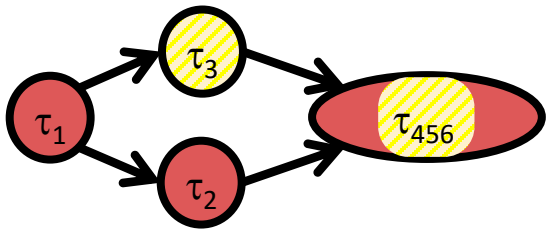
Let P_i be the intra-task parallelism allowed for task τ_i.

<u>Restricted parallelism</u>: allow supernodes to execute with restricted intra-task parallelism.

$$P_i = p$$



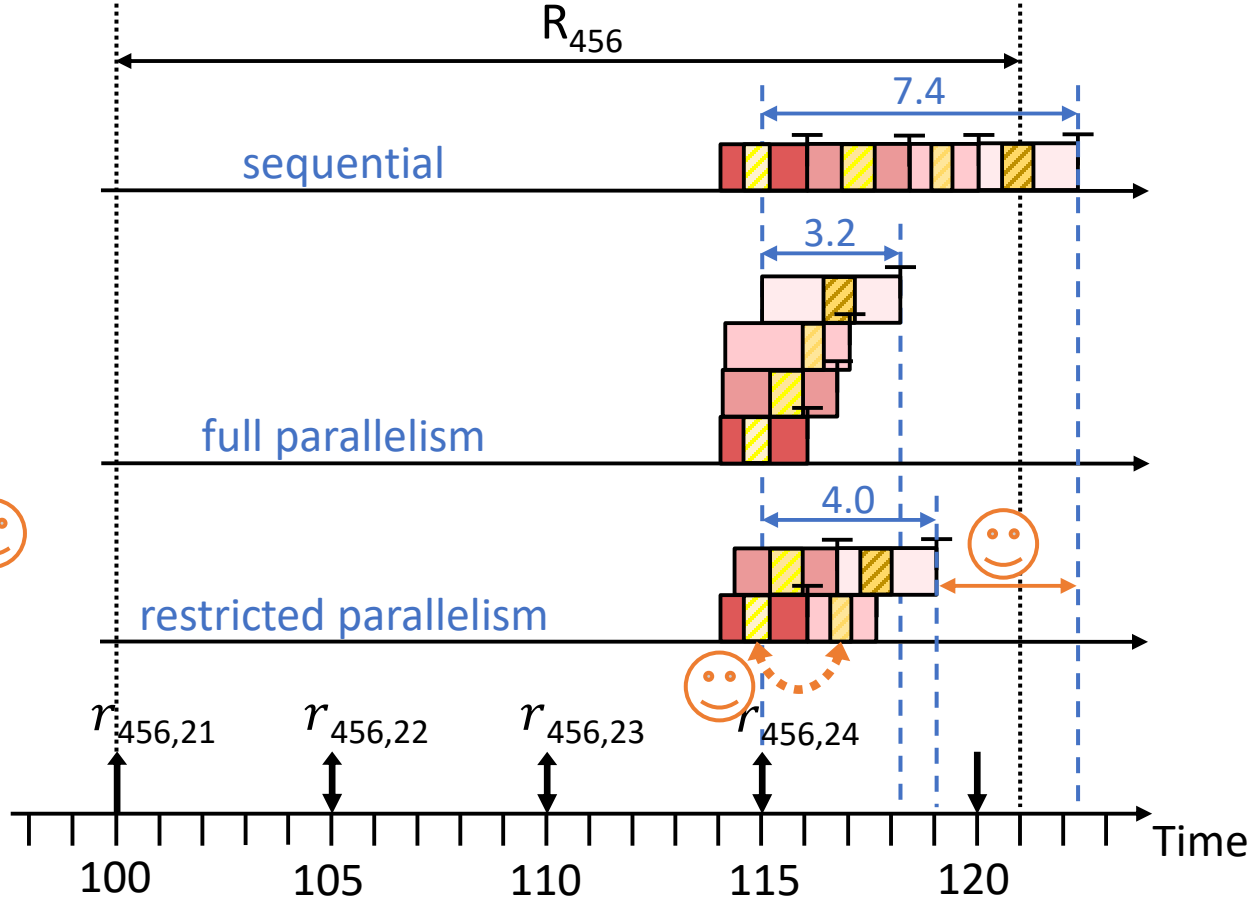
Contribution: restricted parallelism



Let P_i be the intra-task parallelism allowed for task τ_i.

<u>Restricted parallelism</u>: allow supernodes to execute with restricted intra-task parallelism.

- Respects precedence constraints ☺
- Better response times than sequential ☺
- Impacts accuracy ☹



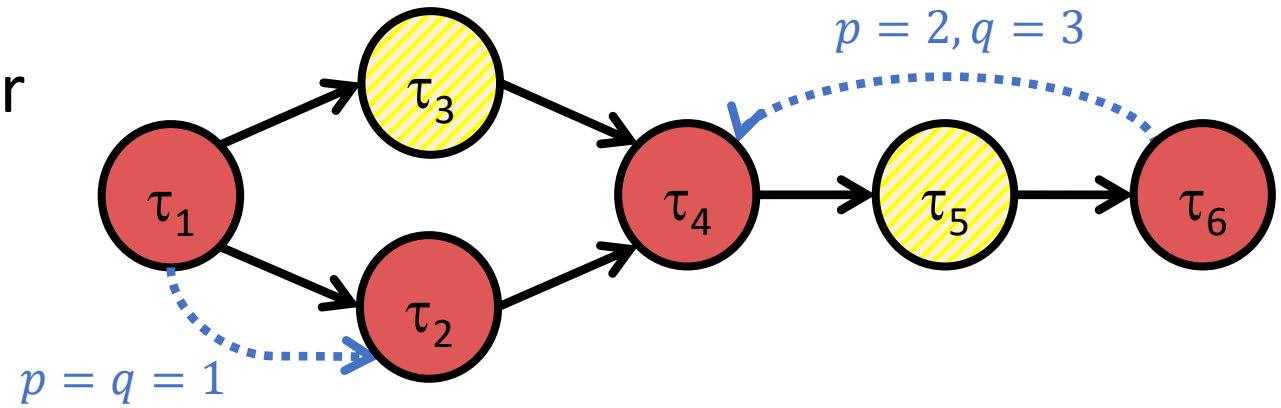
Outline

- Motivation
- Transforming an OpenVX graph to independent sporadic tasks
- Bounding response-times for rp-sporadic tasks
- Evaluation
- Future work

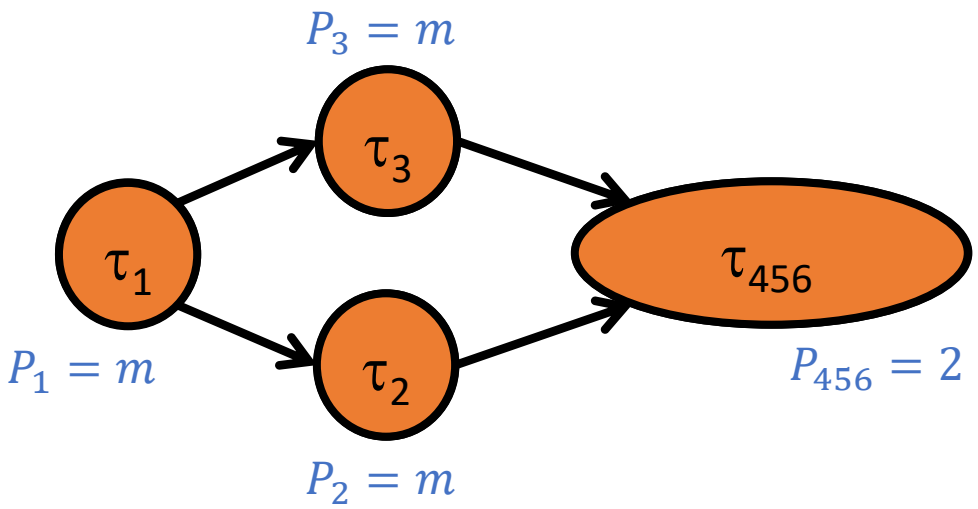
The rp-sporadic task model

We add an additional parameter to the sporadic task model:

$$\tau_i = (\Phi_i, T_i, C_i, P_i)$$



P_i corresponds to the p value from OpenVX.



Note: consider all nodes as CPU.

(we'll come back to this)

The rp-sporadic task model

We add an additional parameter to the sporadic task model:

$$\tau_i = (\Phi_i, T_i, C_i, P_i)$$

P_i corresponds to the p value from OpenVX.

Note: consider all nodes as CPU.

(we'll come back to this)

Feasibility conditions:

- $\sum_i u_i \leq m$
  - same as in either sequential or fully parallel task models
- $\forall i: \quad u_i \leq P_i$
  - generalizes to sequential execution if $p = 1$, or full parallelism if $p = m$

Bounding response times

Definitions:

Let J be the job of interest.

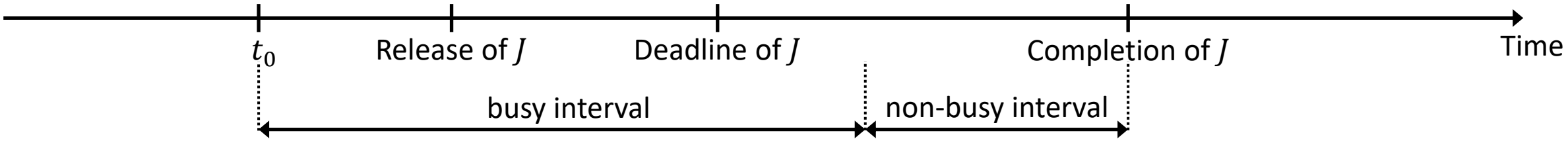
Assume EDF scheduling:

- Let HP be the set of jobs with deadlines at or before J
- Let LP be the set of jobs with deadlines after J

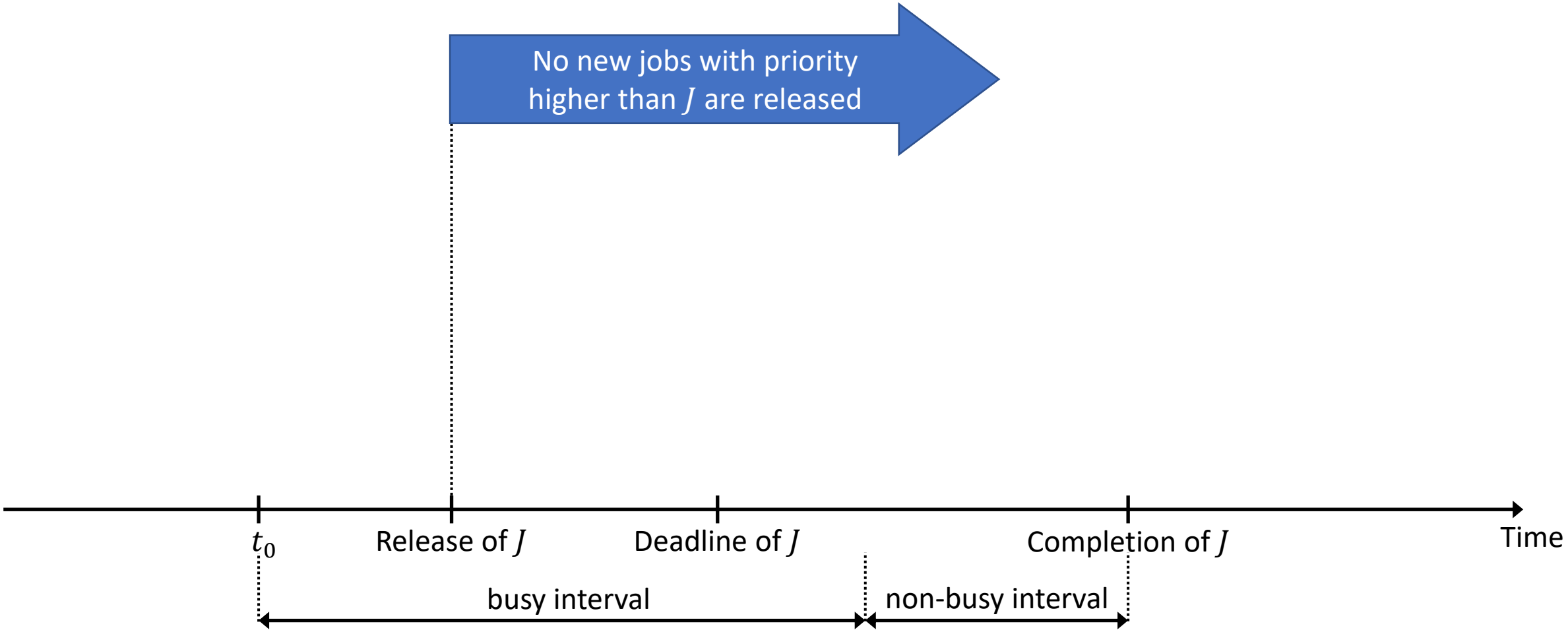
Assumptions:

- GPU use is arbitrated by a mutex lock
  - GPU execution time is included in CPU worst-case execution time
- A task τ_i can have up to P_i jobs scheduled concurrently
- A job in LP can have a non-preemptive section (e.g., GPU use)
  - We'll call these *np-sections*

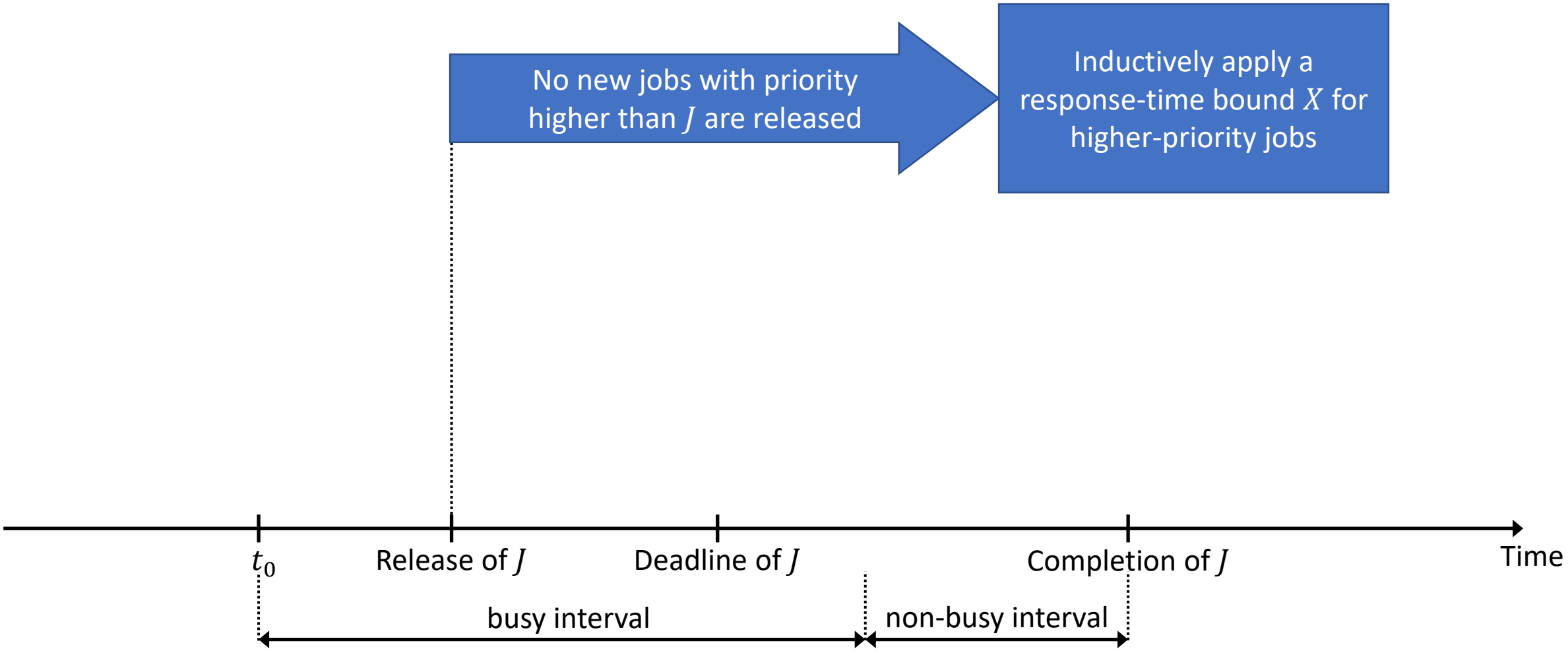
Bounding response times – proof overview



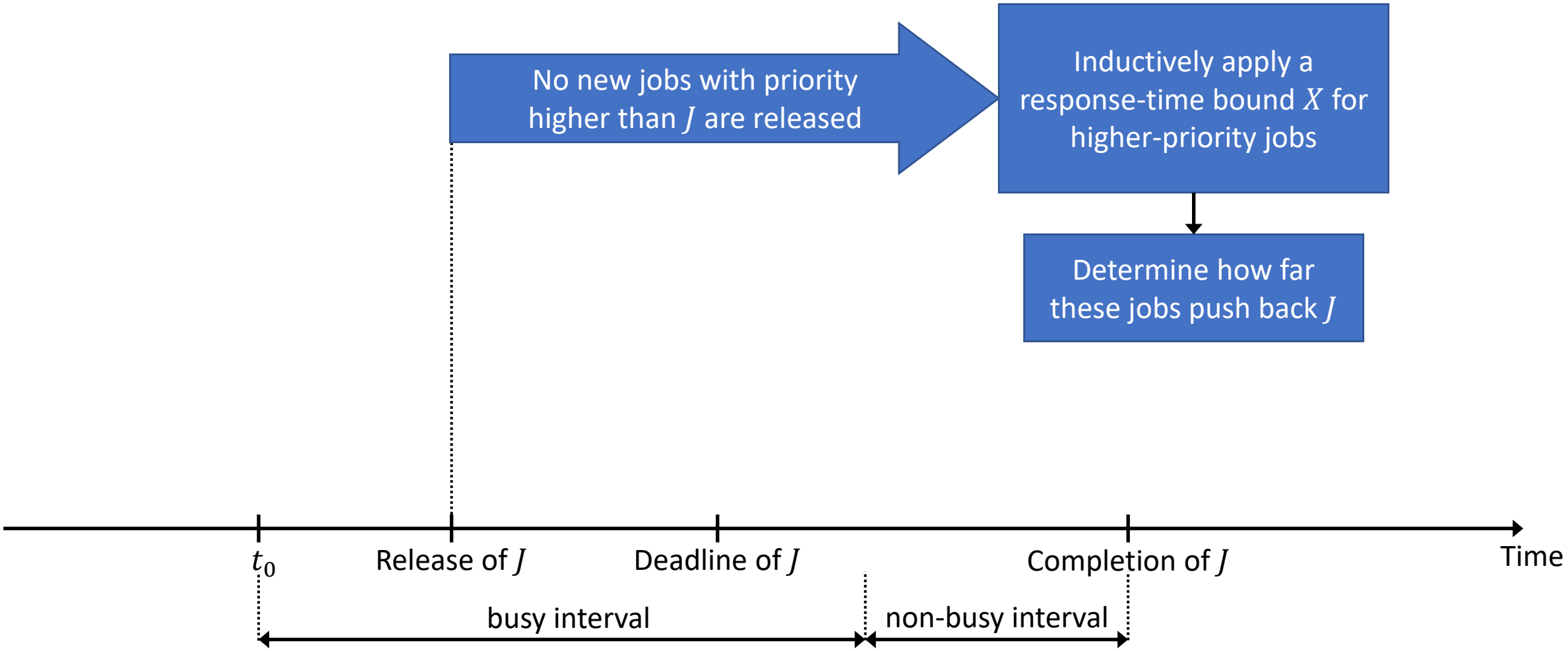
Bounding response times – proof overview



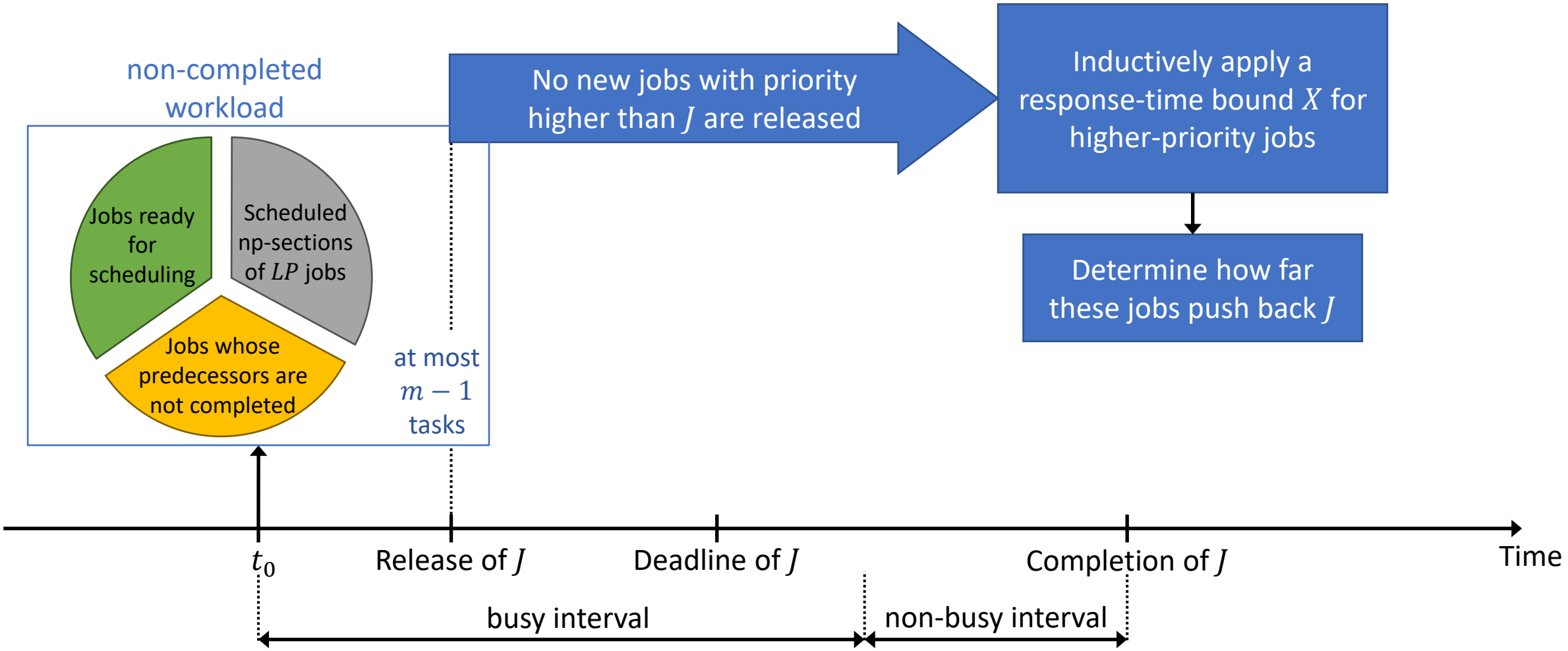
Bounding response times – proof overview



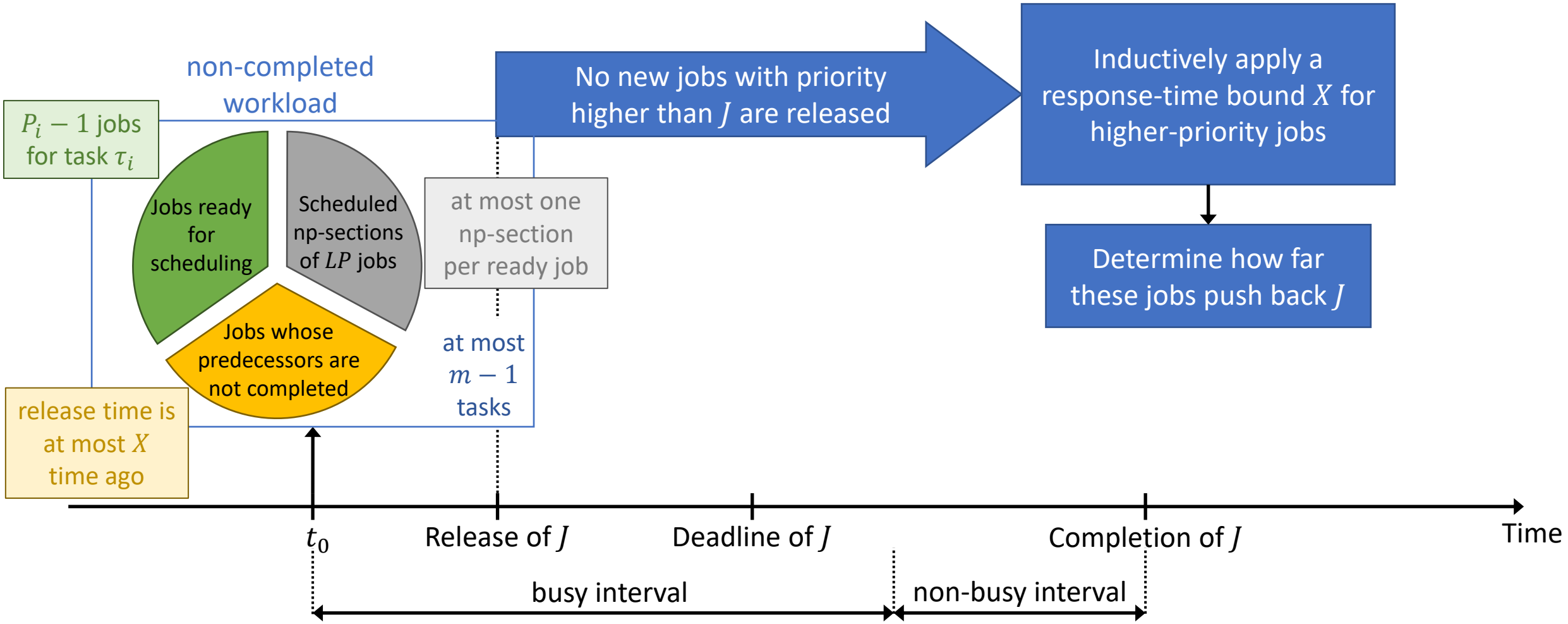
Bounding response times – proof overview



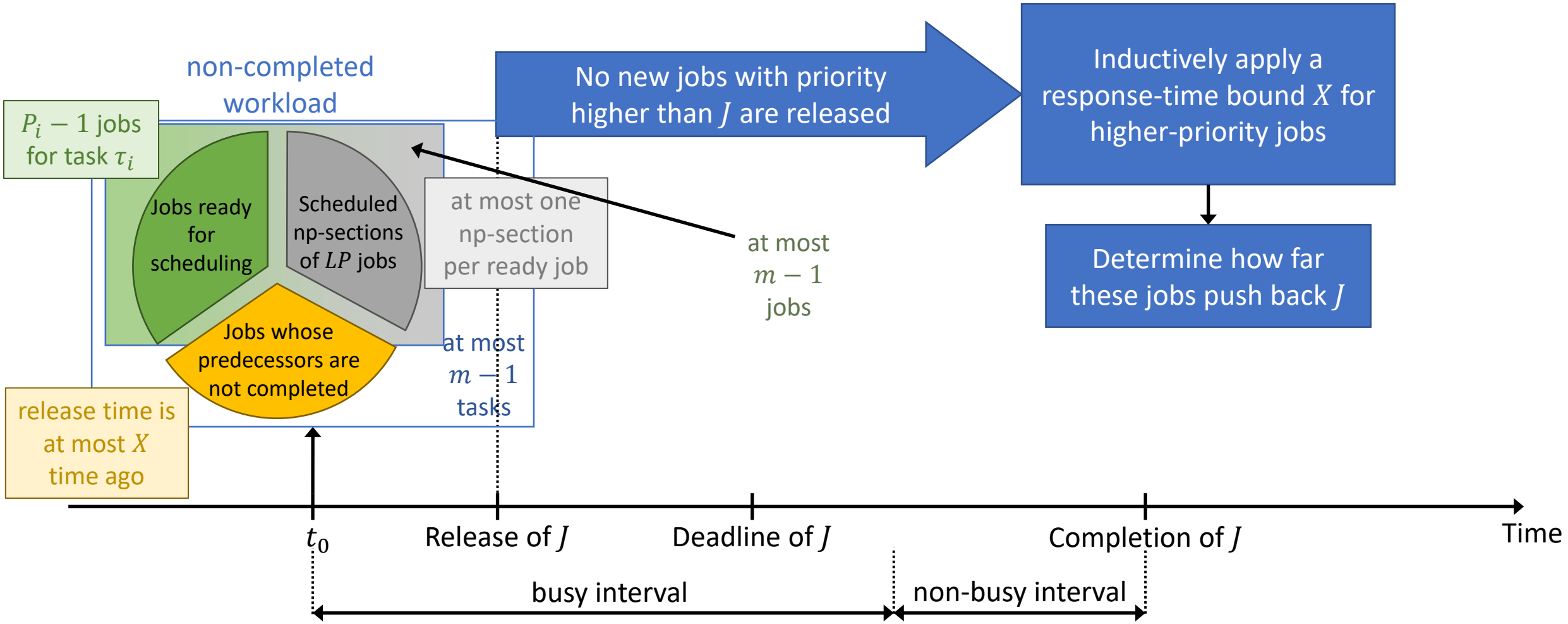
Bounding response times – proof overview



Bounding response times – proof overview

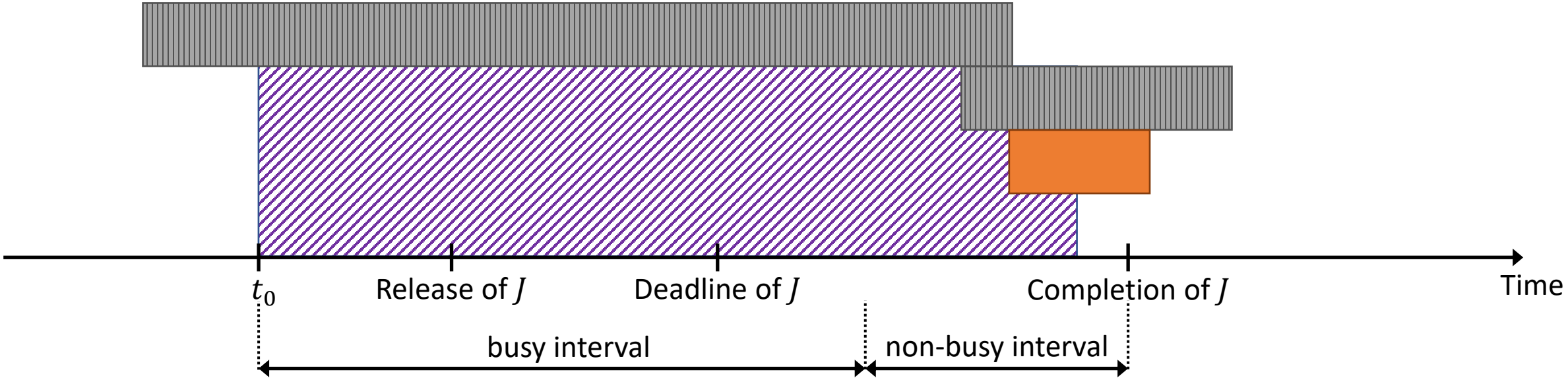


Bounding response times – proof overview



Bounding response times – example schedule

- Scheduled HP jobs
- Scheduled np-sections of LP jobs
- J

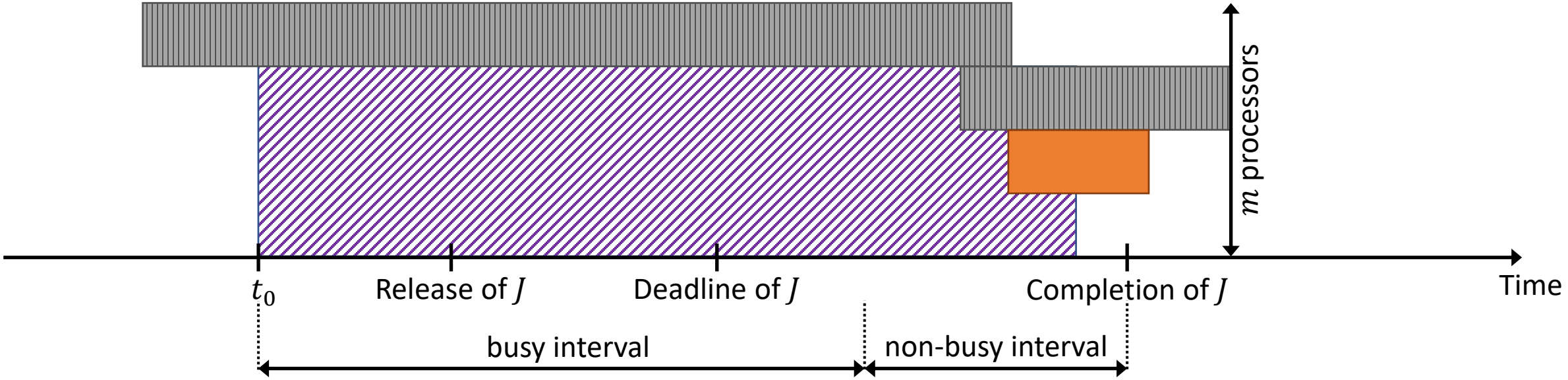


Bounding response times – example schedule

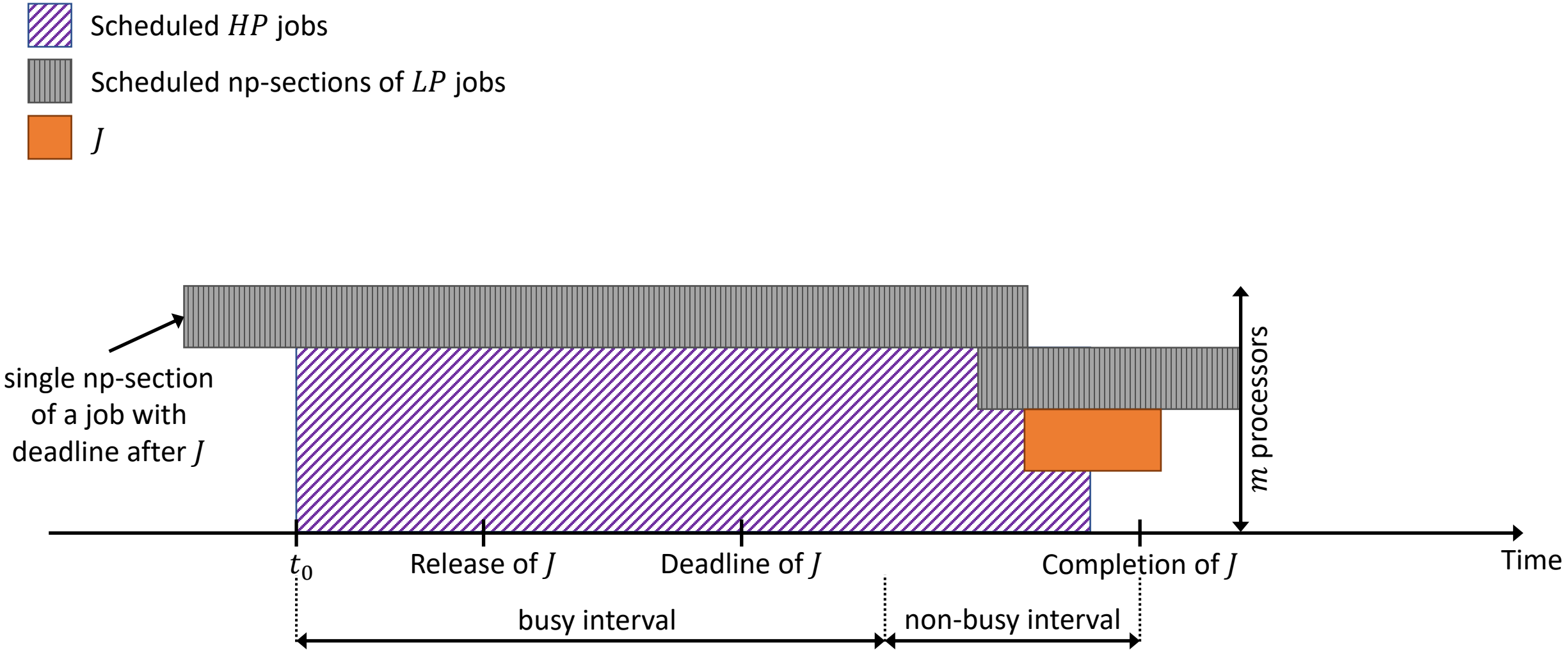
Scheduled HP jobs

Scheduled np-sections of LP jobs

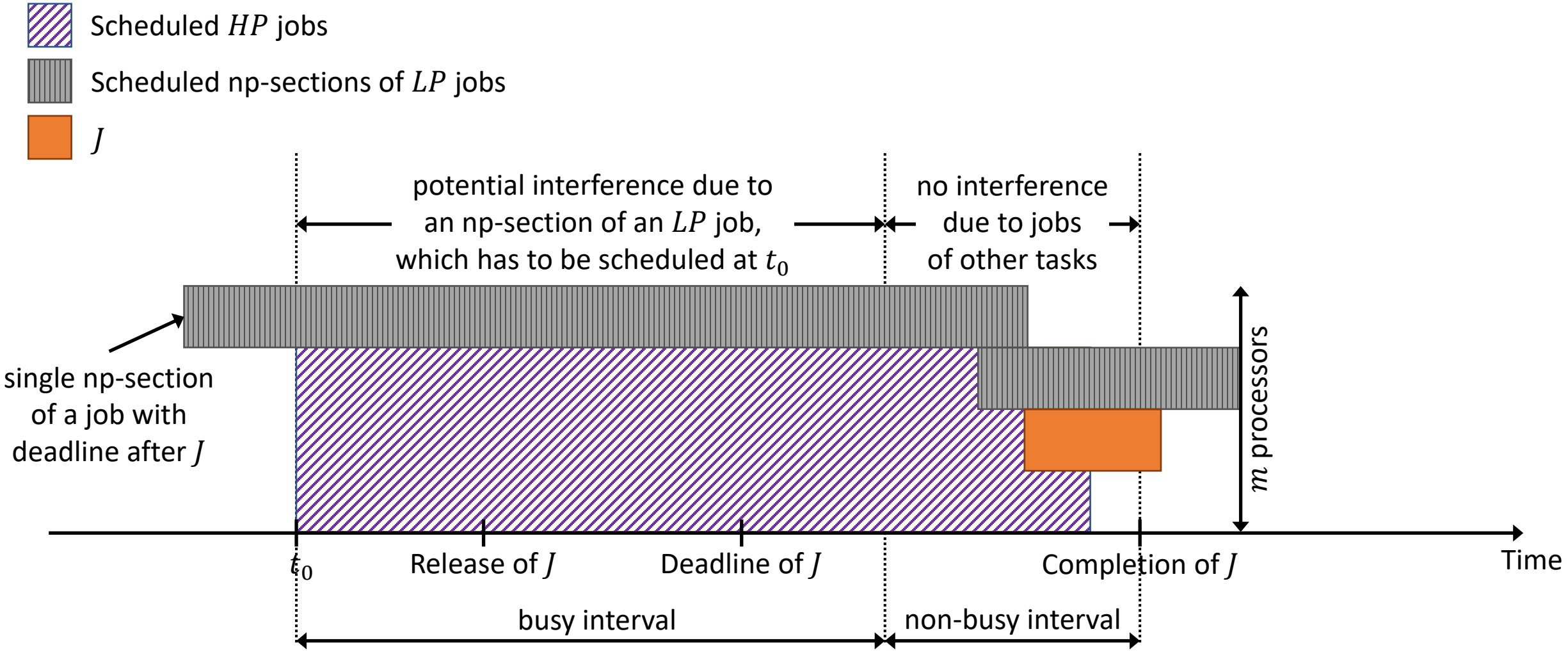
J



Bounding response times – example schedule



Bounding response times – example schedule

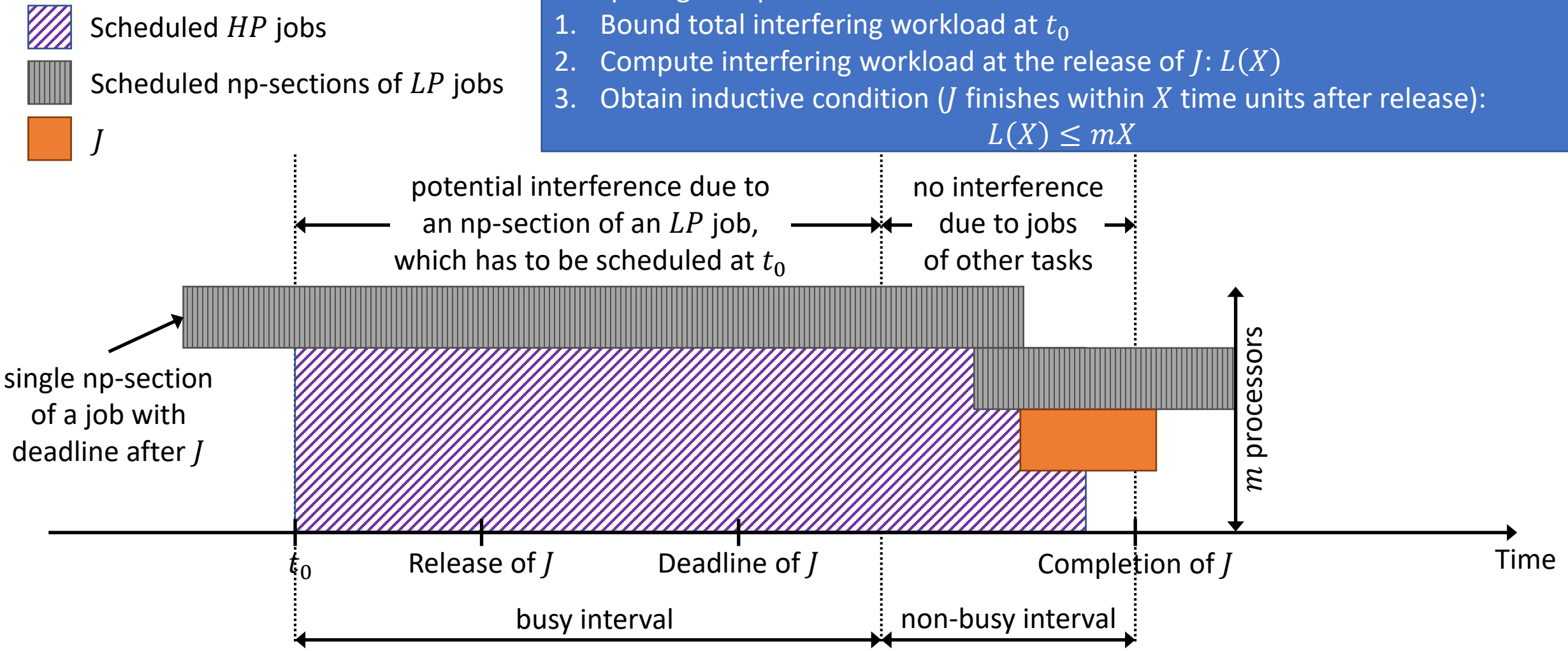


Bounding response times – example schedule

Computing a response-time bound:

1. Bound total interfering workload at t_0
2. Compute interfering workload at the release of J: $L(X)$
3. Obtain inductive condition (J finishes within X time units after release):

$$L(X) \leq mX$$

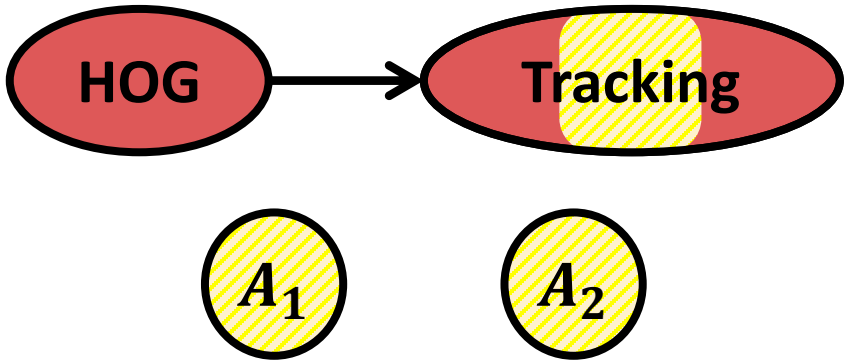


Outline

- Motivation
- Transforming an OpenVX graph to independent sporadic tasks
- Bounding response-times for rp-sporadic tasks
- Evaluation
- Future work

Response-times versus history

- Case study: pedestrian tracking
  - Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0



	Sequential	$p = 1$	$p = 2$	$p = 3$
Analytical Bound (ms)	N/A	N/A	927.27	928.37
Observed Max. Response Time (ms)	25250.67	572.81	713.53	537.60
Observed Avg. Response Time (ms)	11765.23	293.63	280.86	293.07

Response-times versus history

- Case study: pedestrian tracking
  - Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0

Observation: The system is unschedulable if the supernode is not replicated ($p = 1$) or if the graph is scheduled sequentially.

	Sequential	$p = 1$	$p = 2$	$p = 3$
Analytical Bound (ms)	N/A	N/A	927.27	928.37
Observed Max. Response Time (ms)	25250.67	572.81	713.53	537.60
Observed Avg. Response Time (ms)	11765.23	293.63	280.86	293.07

Response-times versus history

- Case study: pedestrian tracking
  - Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0

Observation: The analytical response-time bounds upper-bounded the observed response times for $p \geq 2$.

	Sequential	$p = 1$	$p = 2$	$p = 3$
Analytical Bound (ms)	N/A	N/A	927.27	928.37
Observed Max. Response Time (ms)	25250.67	572.81	713.53	537.60
Observed Avg. Response Time (ms)	11765.23	293.63	280.86	293.07

Response-times versus history

- Case study: pedestrian tracking
  - Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0

Observation: For $p \geq 2$, as p increases, the observed maximum response times decrease.

	Sequential	$p = 1$	$p = 2$	$p = 3$
Analytical Bound (ms)	N/A	N/A	927.27	928.37
Observed Max. Response Time (ms)	25250.67	572.81	713.53	537.60
Observed Avg. Response Time (ms)	11765.23	293.63	280.86	293.07

Response-times versus history

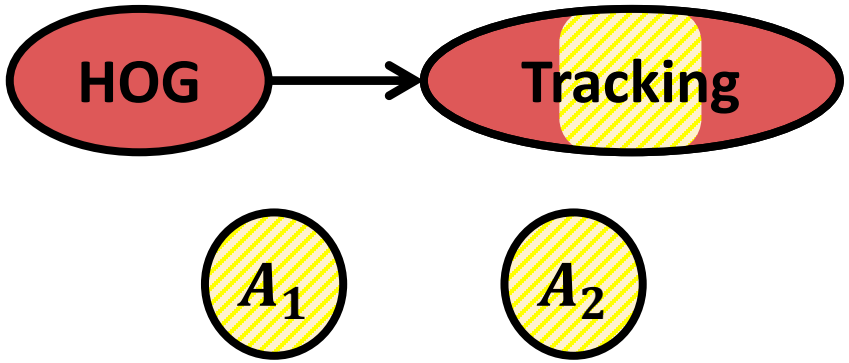
- Case study: pedestrian tracking
  - Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0

Observation: The analytical response-time bounds are almost identical for $p \geq 2$, as the number of tasks with $P_i < m$ is unchanged.

	Sequential	$p = 1$	$p = 2$	$p = 3$
Analytical Bound (ms)	N/A	N/A	927.27	928.37
Observed Max. Response Time (ms)	25250.67	572.81	713.53	537.60
Observed Avg. Response Time (ms)	11765.23	293.63	280.86	293.07

Accuracy versus history

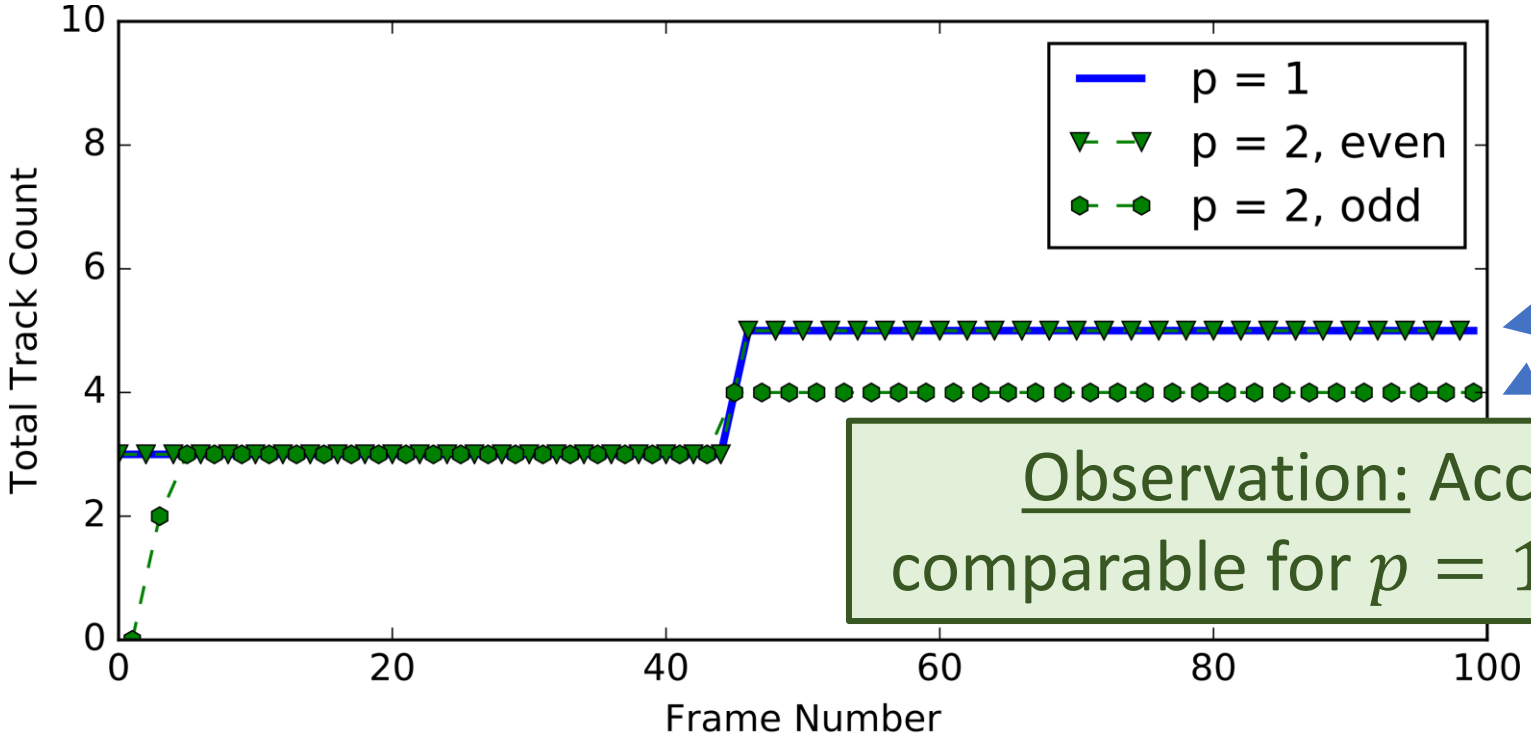
- We expect the accuracy to decrease as p increases
  - The distance between the "current" position of a pedestrian and their last-tracked position increases



- In our experiments, p represents the actual age of history, rather than the maximum age
  - E.g., for $p = 2$, data produced by frames 0,2,4,6,… is never available to frames 1,3,5,7,…

Accuracy versus history: $p = 2$

- Accuracy metric: total number of tracks throughout the video
  - Success: track count is similar to that of $p = 1$

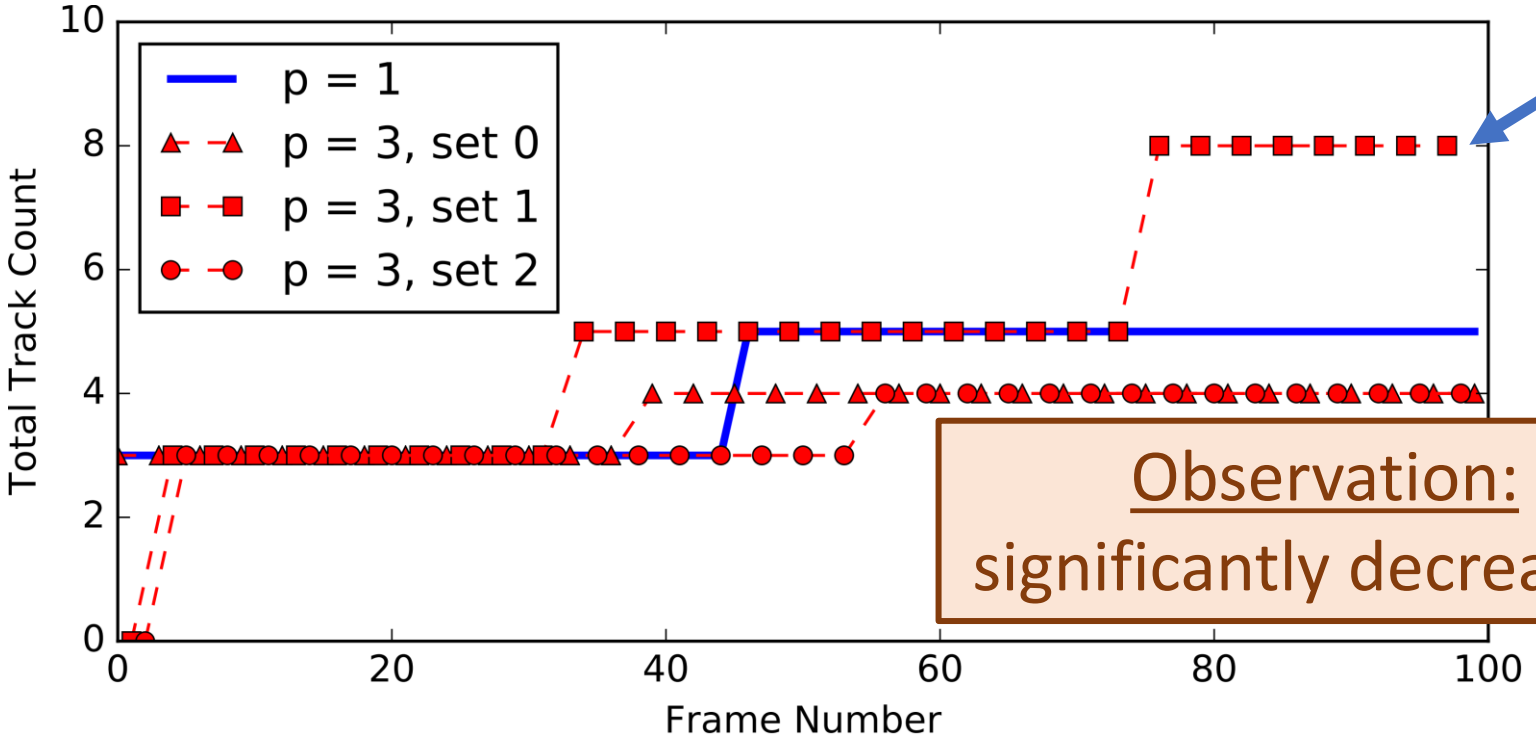


Both sets of frames result in similar track counts

Observation: Accuracy is comparable for $p = 1$ and $p = 2$.

Accuracy versus history: $p = 3$

- Accuracy metric: total number of tracks throughout the video
  - Success: track count is similar to that of $p = 1$



One of the three sets of frames has many more tracks (e.g., might have lost and then re-tracked pedestrians)

Observation: Accuracy significantly decreases for $p = 3$.

Outline

- Motivation
- Transforming an OpenVX graph to independent sporadic tasks
- Bounding response-times for rp-sporadic tasks
- Evaluation
- Future work

Future Work

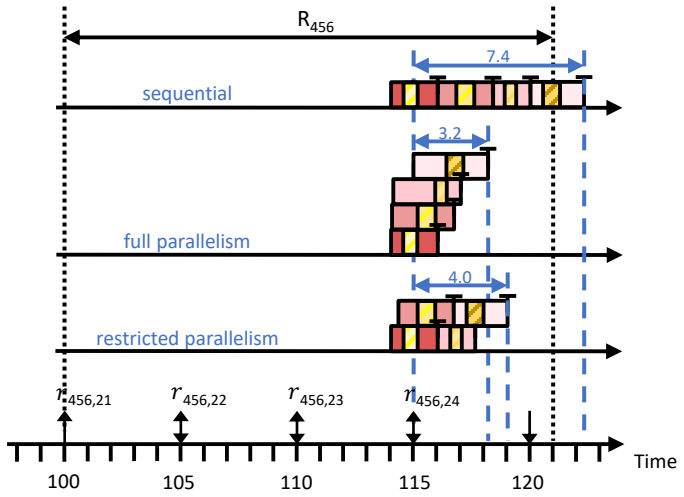
- Remove assumptions in analysis
- Extend experiments to consider higher-level notions of accuracy (e.g., number of obstacles not detected)
- Perform a large-scale study of the trade-off between response times, history, and accuracy
- Develop a tool to enable computer-vision programmers to graphically specify OpenVX programs, which are then automatically transformed to fine-grained implementations with response-time analysis

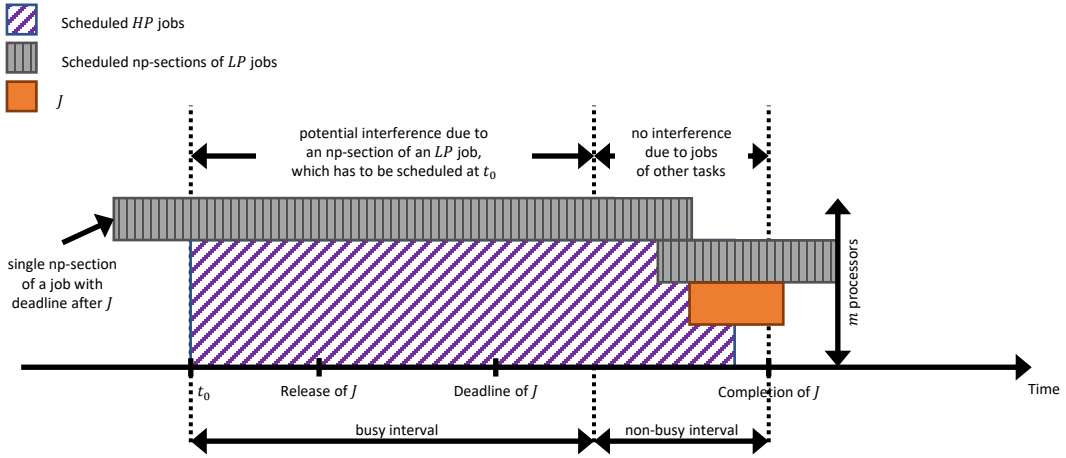
Summary

- Introduced rp-sporadic task model
  - Provided response-time bound derivation
  - First work to bound response-times for graphs with arbitrary cycles
- Preliminary evaluation of history/accuracy/response-time trade-off

Questions? Please let us know!

{amert,rdkl,anderson}@cs.unc.edu





	Sequential	$p = 1$	$p = 2$	$p = 3$
Analytical Bound (ms)	N/A	N/A	927.27	928.37
Observed Max. Response Time (ms)	25250.67	572.81	713.53	537.60
Observed Avg. Response Time (ms)	11765.23	293.63	280.86	293.07