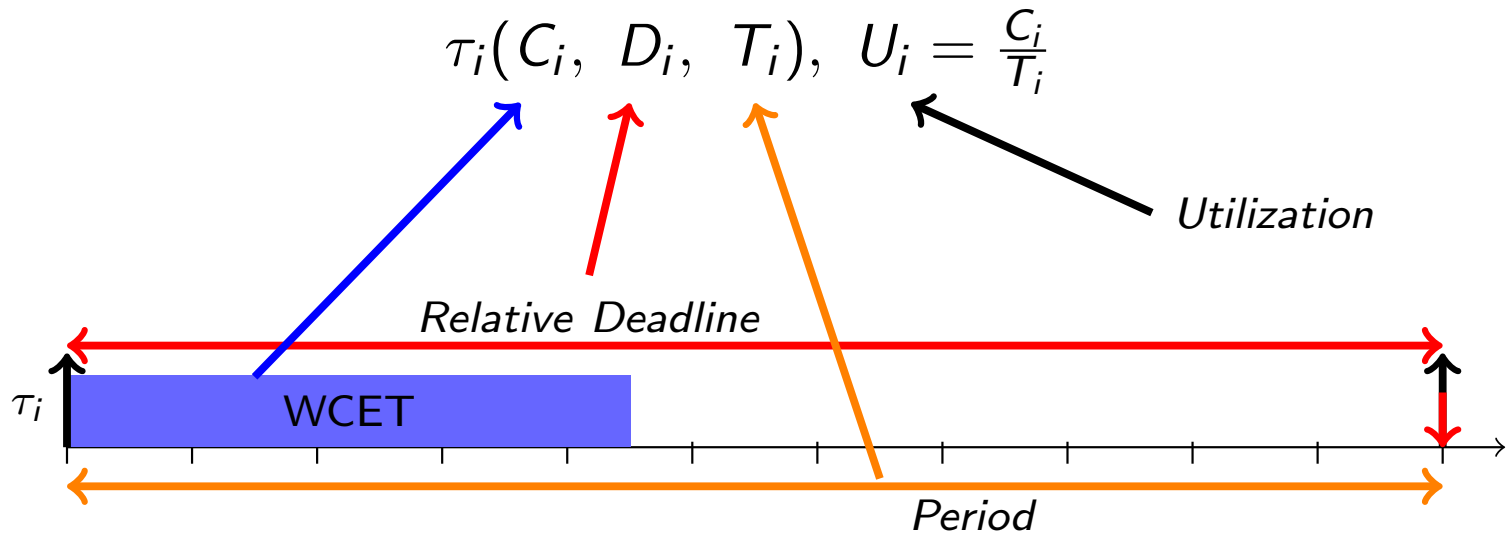
Scheduling Self-Suspending Tasks: New and Old Results

Jian-Jia Chen[1], Tobias Hahn[2], Ruben Hoeksma[2], Nicole Megow[2], and Georg von der Brüggen[1]

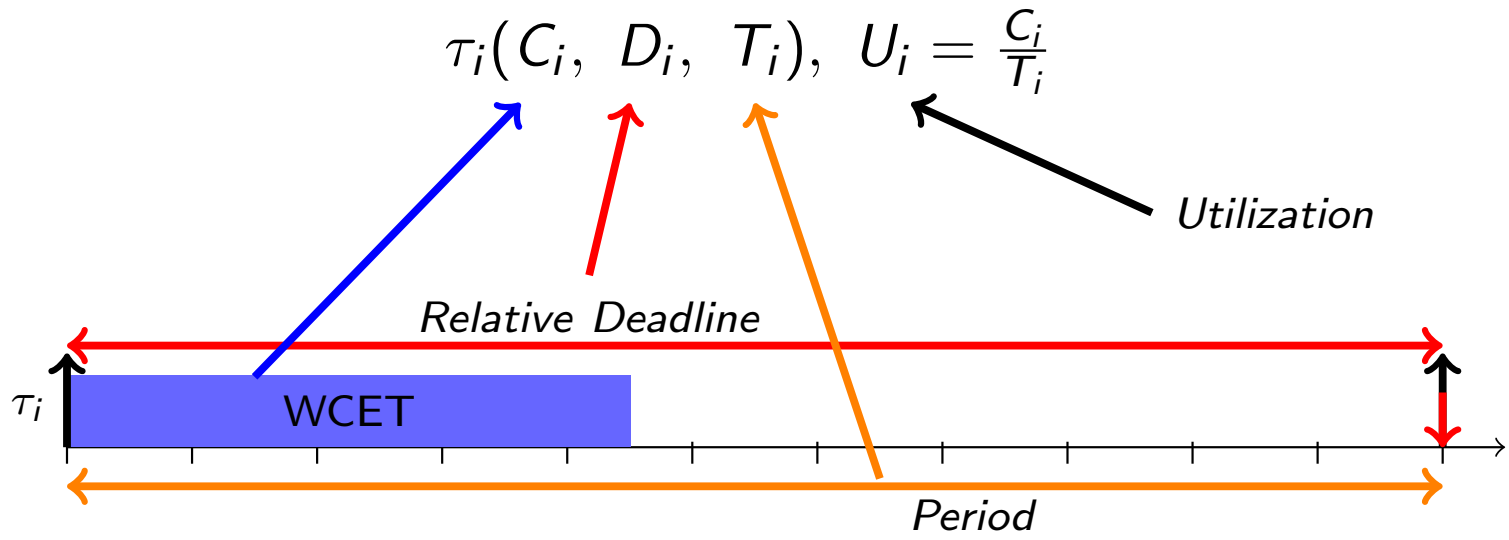
[1]**TU Dortmund University, Germany**
[2]**University of Bremen, Germany**

11 July 2019

Sporadic Task Model



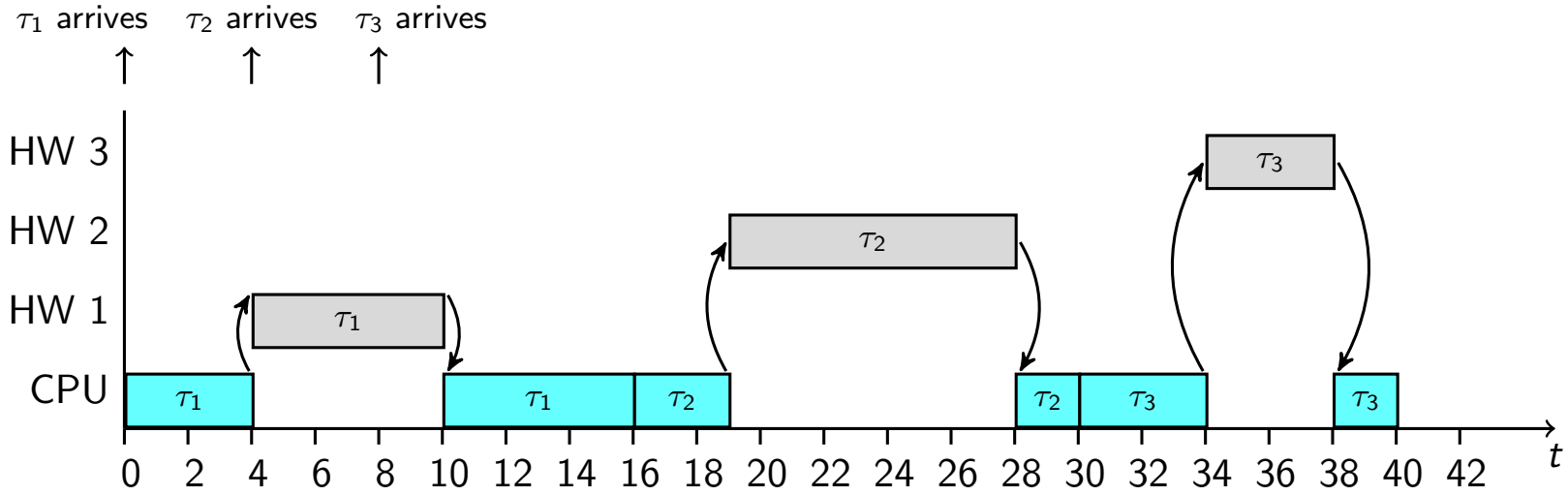
Sporadic Task Model



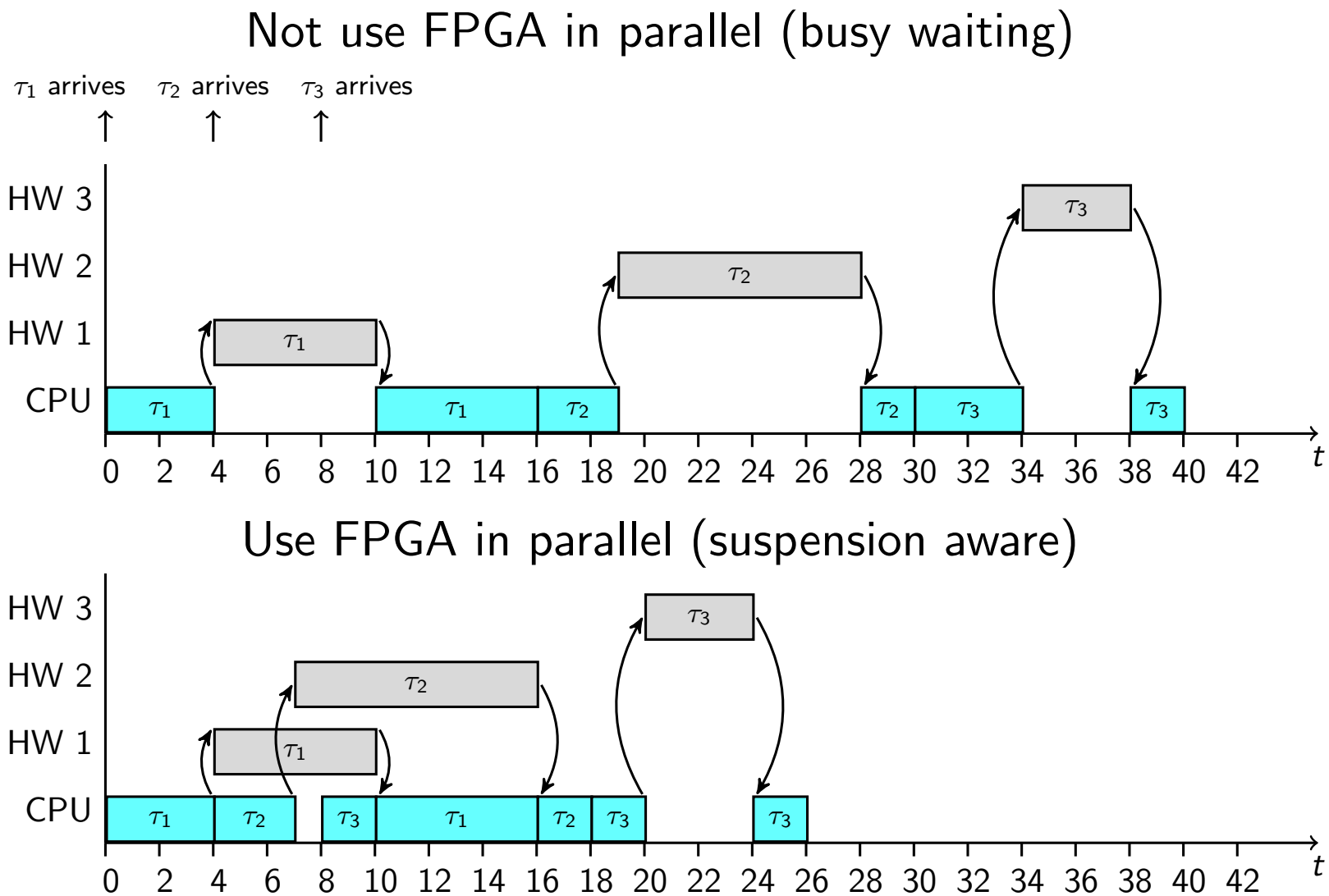
- Assumption: tasks do not voluntarily suspend themselves

Reasons for Suspension: Hardware Acceleration

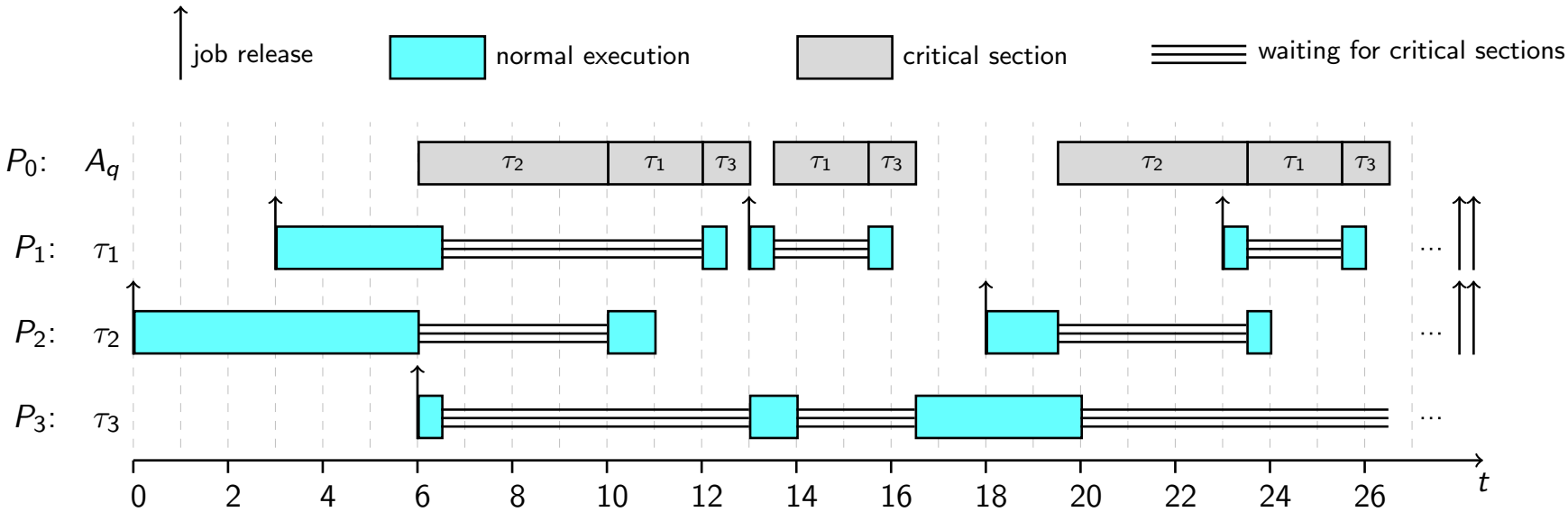
Not use FPGA in parallel (busy waiting)



Reasons for Suspension: Hardware Acceleration

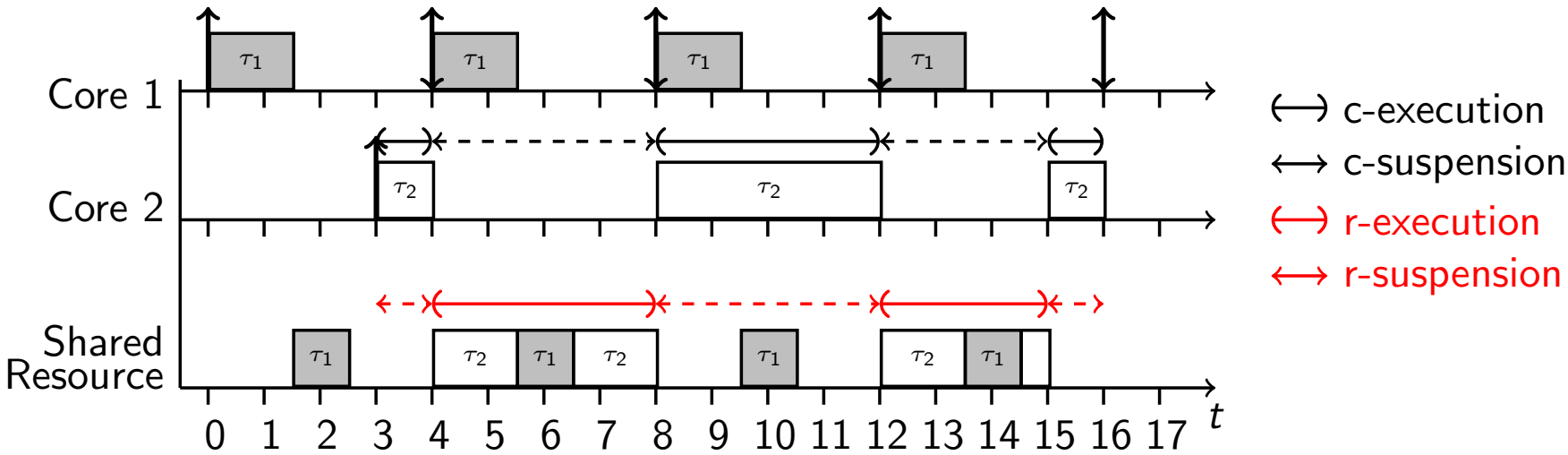


Reasons for Self-Suspension: Locking Protocols



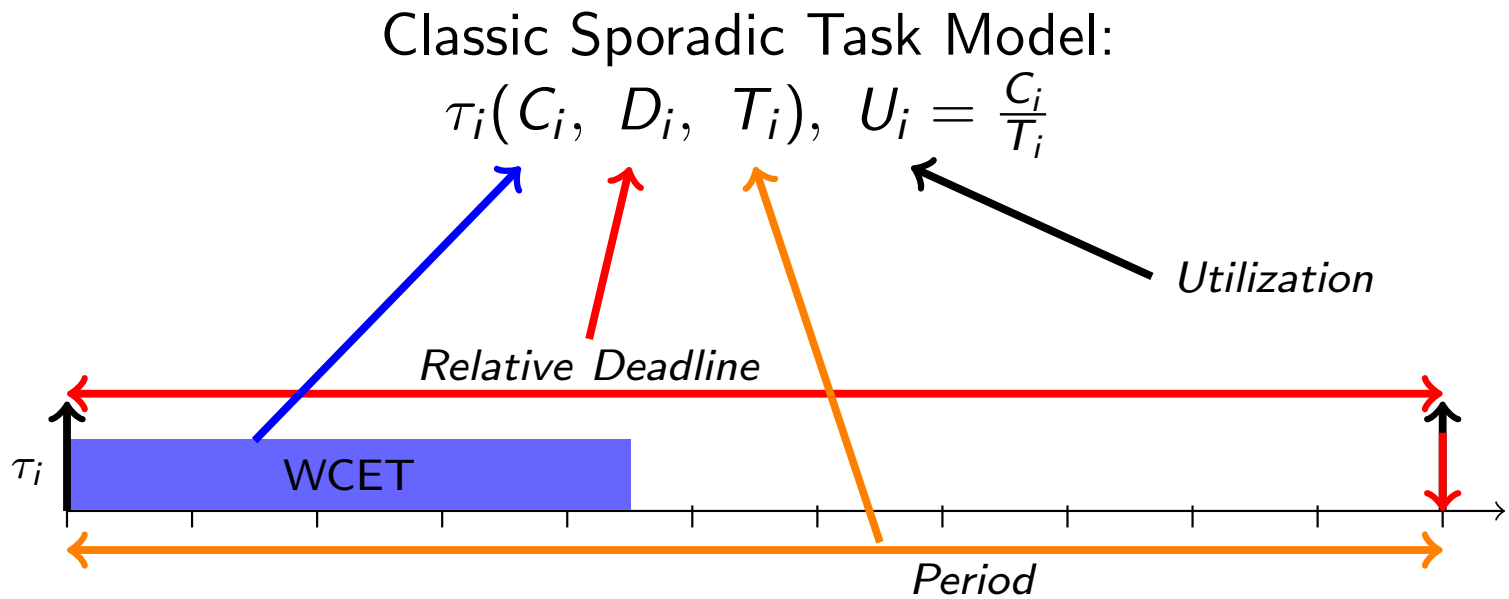
- Semaphores in multiprocessor systems: remote blocking due to mutual exclusion

Reasons for Self-Suspension: Physical Resource Sharing



- Multiple cores may share a bus
- Memory centric scheduling

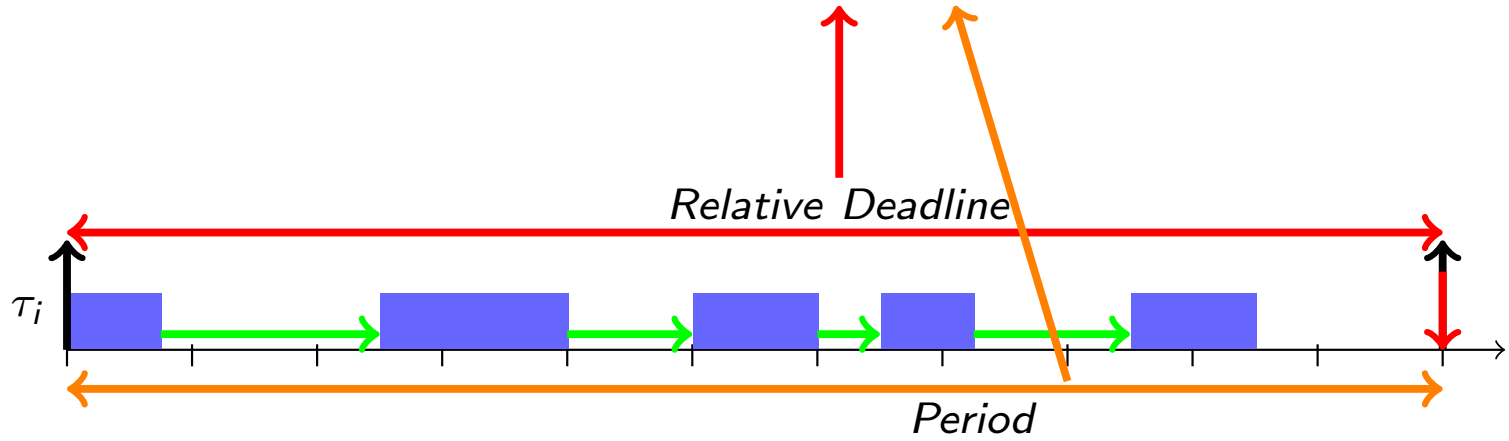
Self-Suspension Task Models



Self-Suspension Task Models

Dynamic Self-Suspending Sporadic Task Model:

$\tau_i((C_i, S_i),\ D_i,\ T_i)$

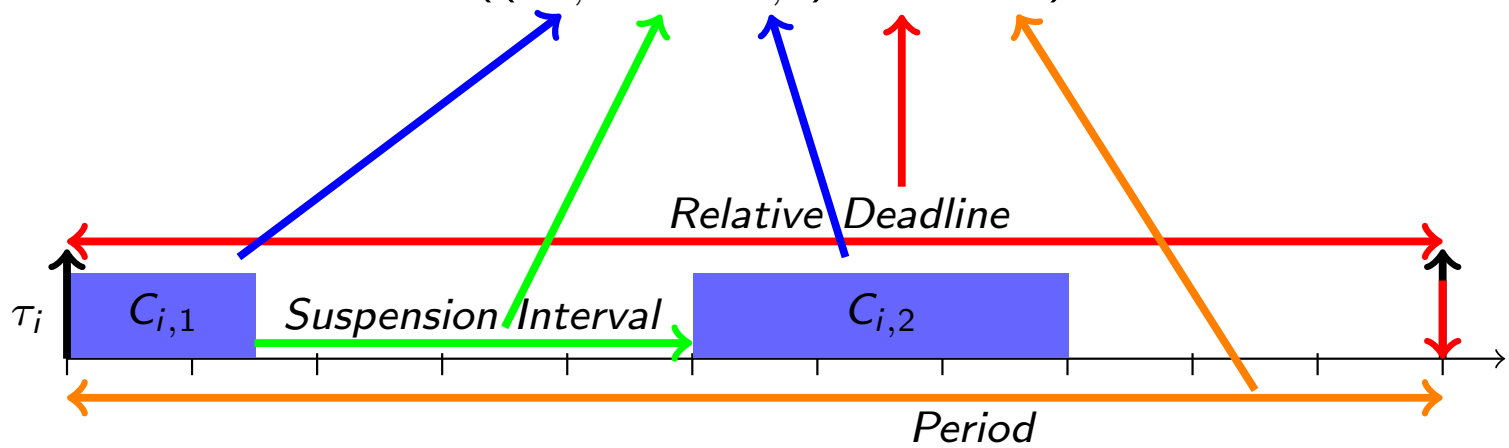


- S_i = maximum total suspension time
- C_i = sum of segment WCETs
- No information about the execution / suspension pattern
- Flexible and inaccurate

Self-Suspension Task Models

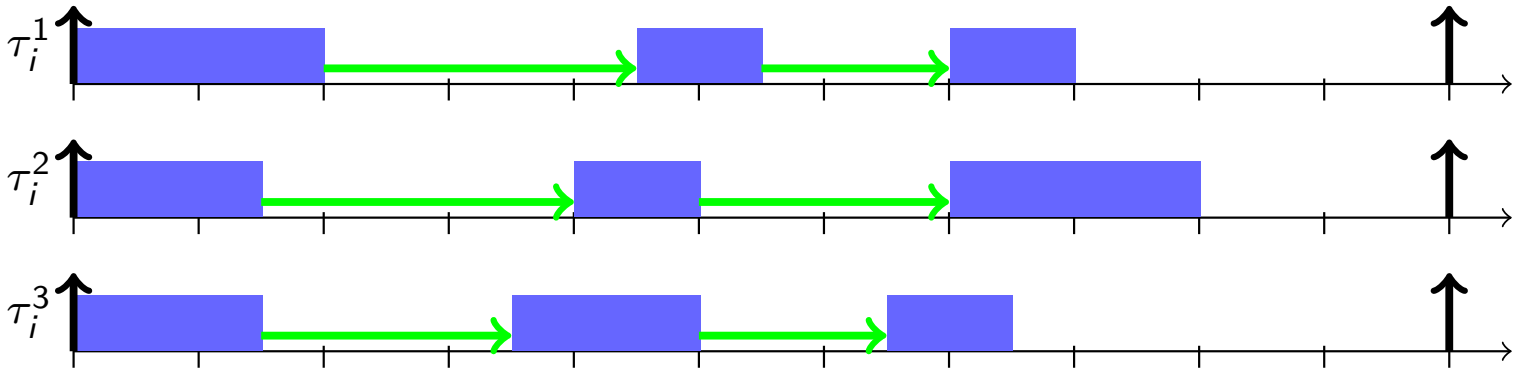
Segmented Self-Suspending Sporadic Task Model:

$\tau_i((C_{i,1}, S_i, C_{i,2}),\ D_i,\ T_i)$



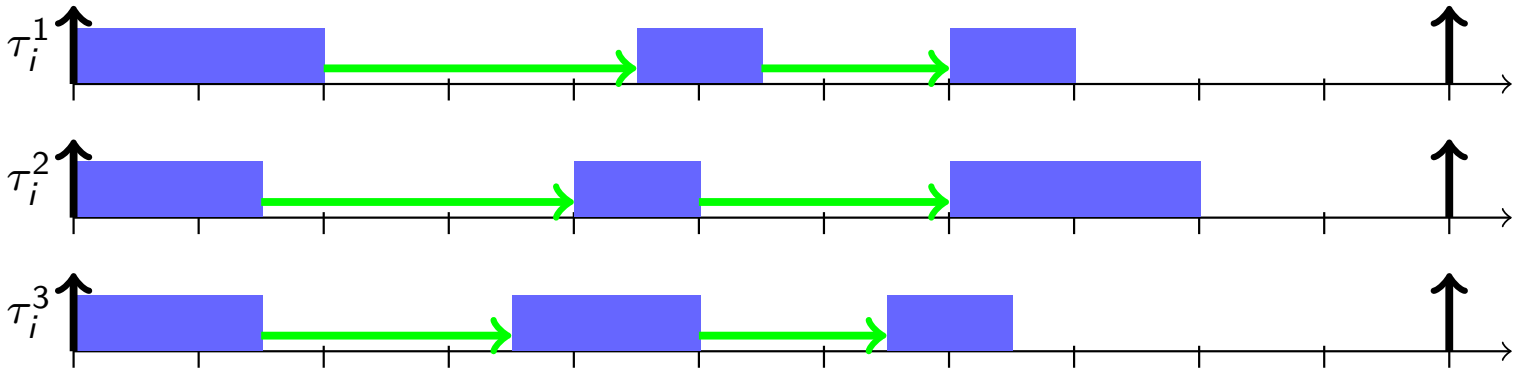
- Fixed interleaved execution / suspension pattern
- Accurate and restrictive

Hybrid Self-Suspension Models



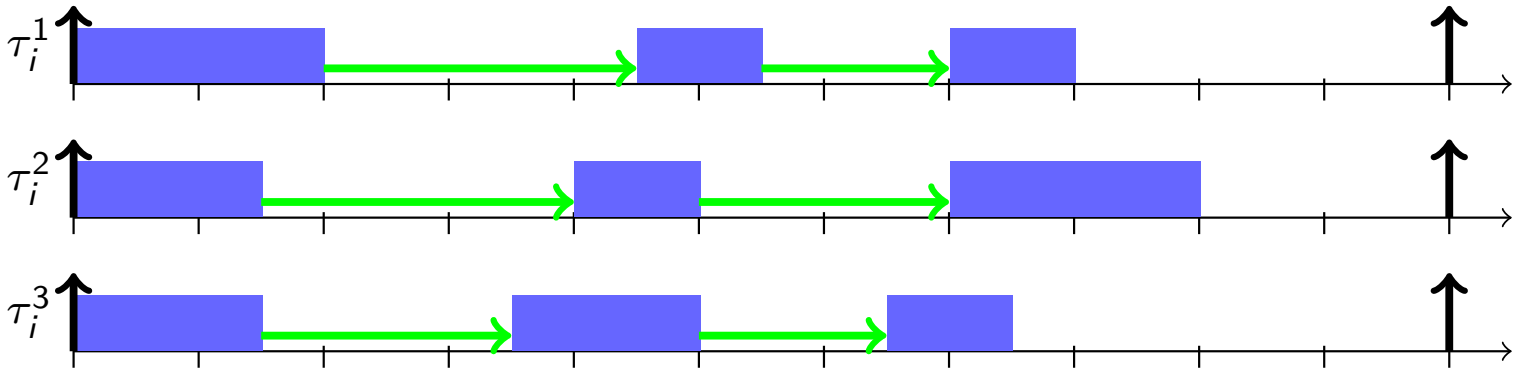
- Set of execution / suspension patterns

Hybrid Self-Suspension Models



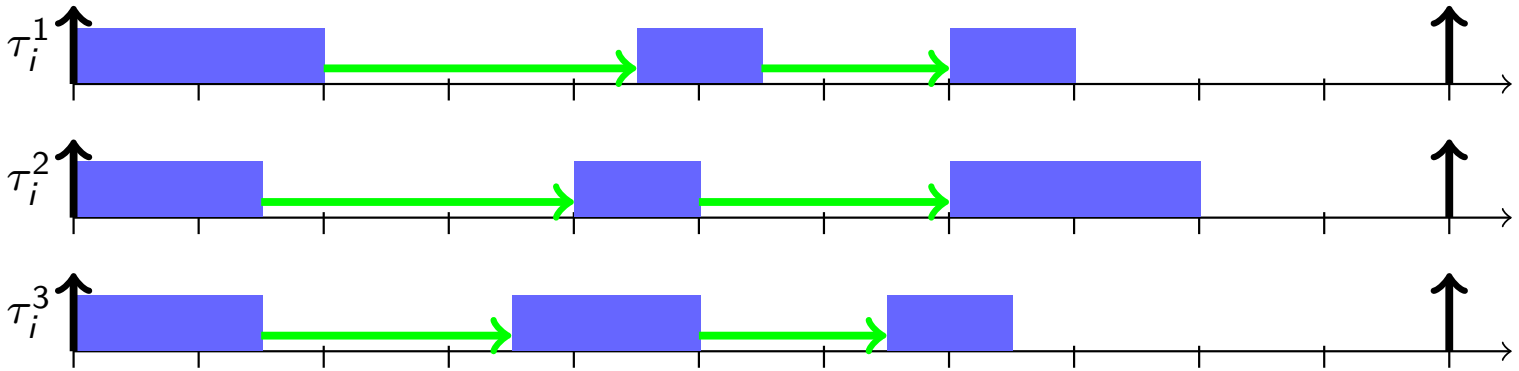
- Set of execution / suspension patterns
- Assumes number of suspension intervals to be given

Hybrid Self-Suspension Models



- Set of execution / suspension patterns
- Assumes number of suspension intervals to be given
- Different tradeoffs regarding flexibility and accuracy

Hybrid Self-Suspension Models



- Set of execution / suspension patterns
- Assumes number of suspension intervals to be given
- Different tradeoffs regarding flexibility and accuracy
- Depending on
  - Additional information
  - Time information is available

A Brief Overview

- Self-suspension models with different application scenarios
- Scheduling algorithms and schedulability tests

A Brief Overview

- Self-suspension models with different application scenarios
- Scheduling algorithms and schedulability tests
- Multiple flaws in the literature

A Brief Overview

- Self-suspension models with different application scenarios
- Scheduling algorithms and schedulability tests
- Multiple flaws in the literature
- Theoretical results (segmented):
  - Ridouard et al. 2004: scheduler design for the segmented model is $\mathcal{NP}$-hard in the strong sense

A Brief Overview

- Self-suspension models with different application scenarios
- Scheduling algorithms and schedulability tests
- Multiple flaws in the literature
- Theoretical results (segmented):
  - Ridouard et al. 2004: scheduler design for the segmented model is $\mathcal{NP}$-hard in the strong sense
  - Speedup factors for fixed relative deadline scheduling: Chen and Liu 2014, von der Brüggen et al. 2016

A Brief Overview

- Self-suspension models with different application scenarios
- Scheduling algorithms and schedulability tests
- Multiple flaws in the literature
- Theoretical results (segmented):
  - Ridouard et al. 2004: scheduler design for the segmented model is $\mathcal{NP}$-hard in the strong sense
  - Speedup factors for fixed relative deadline scheduling: Chen and Liu 2014, von der Brüggen et al. 2016
- Theoretical results (dynamic):
  - Huang et al. 2014: priority assignment with resource augmentation factor 2 compared to optimal fixed-priority
  - Chen 2016: unbounded speedup factors for FP, EDF, LLF, and EDZL if only the execution time is sped up

A Brief Overview

- Self-suspension models with different application scenarios
- Scheduling algorithms and schedulability tests
- Multiple flaws in the literature
- Theoretical results (segmented):
  - Ridouard et al. 2004: scheduler design for the segmented model is $\mathcal{NP}$-hard in the strong sense
  - Speedup factors for fixed relative deadline scheduling: Chen and Liu 2014, von der Brüggen et al. 2016
- Theoretical results (dynamic):
  - Huang et al. 2014: priority assignment with resource augmentation factor 2 compared to optimal fixed-priority
  - Chen 2016: unbounded speedup factors for FP, EDF, LLF, and EDZL if only the execution time is sped up
- Here: fundamental theoretical analysis of the most basic recurrent setting, i.e., frame based tasks

Results from Operations Research - Uniprocessor

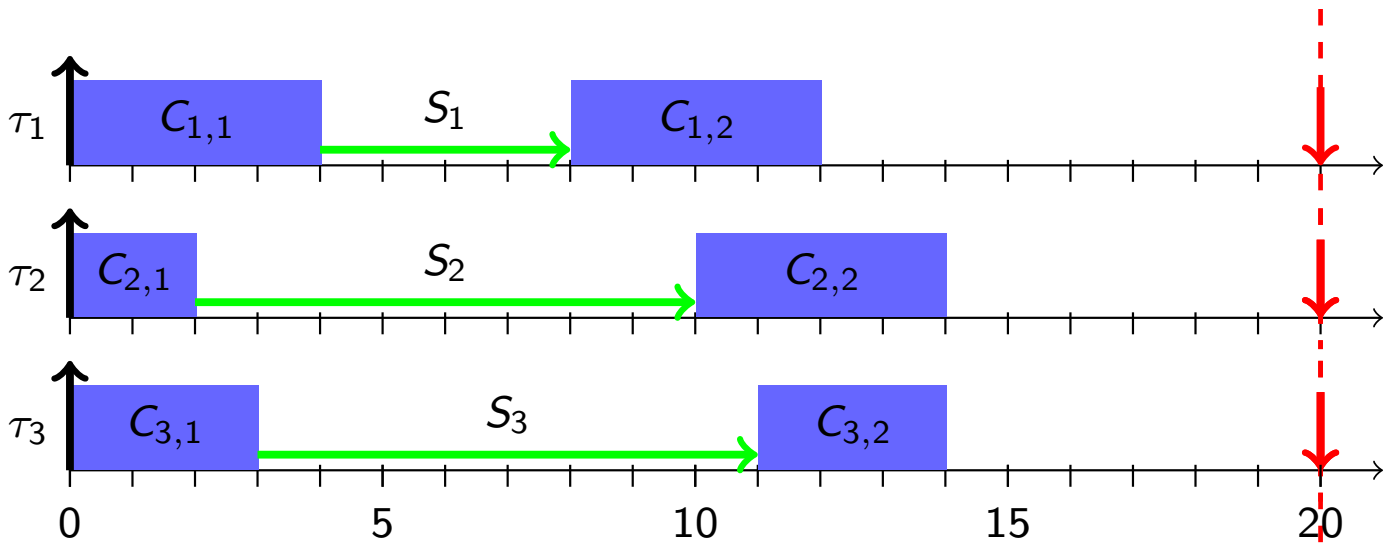
- Kern and Nawijn 1991: multi-operation jobs with time lags

Multi-Operation Jobs with Time Lags on a Single Machine

There are jobs to be processed on a single machine. Each job requires two operations to be processed in a given order. The time between the start of the second operation and the completion of the first operation cannot be less than a pre-specified time constant, i.e., there is a minimal time lag between the two operations of a job. Our aim is to minimize the makespan, i.e., the completion time of the second operation of the last job in the schedule.

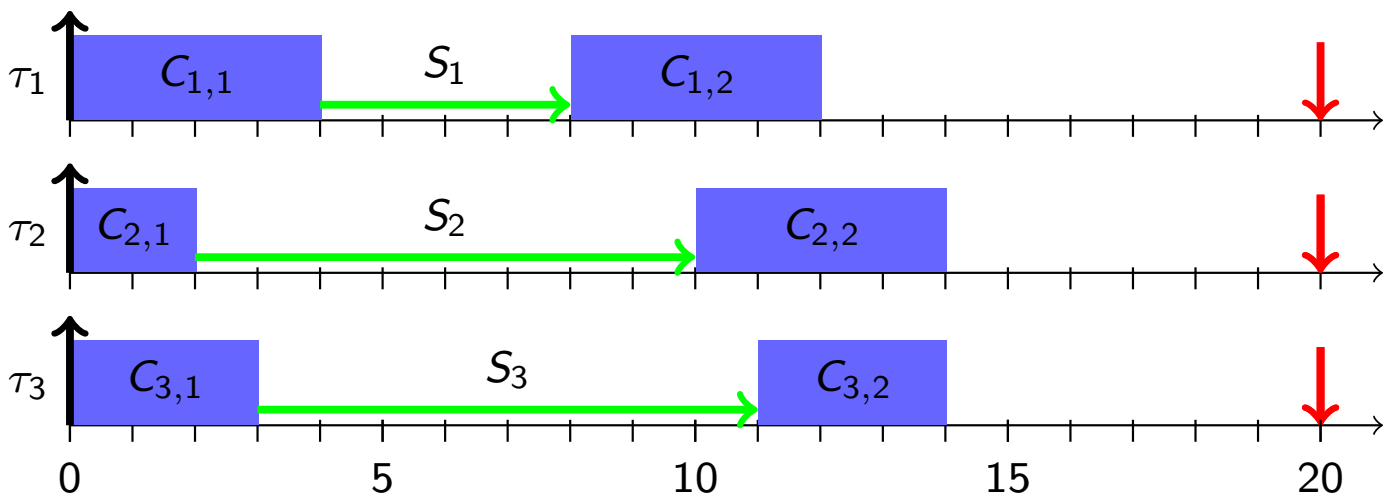
Results from Operations Research - Uniprocessor

- Kern and Nawijn 1991: multi-operation jobs with time lags
- Identical to frame-based one-segmented self-suspension



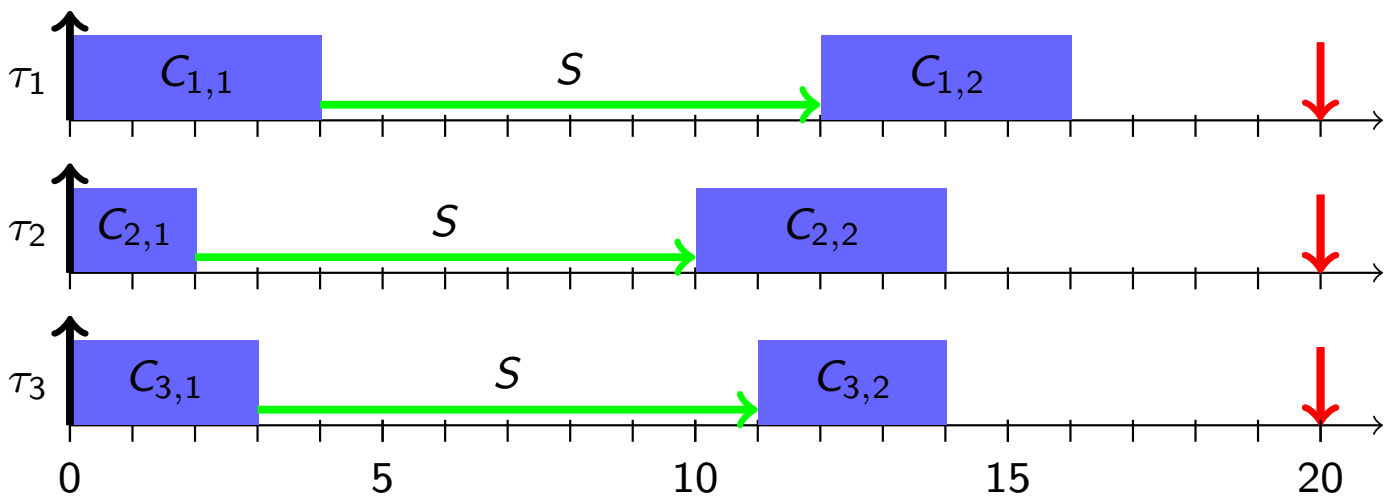
Results from Operations Research - Uniprocessor

- Kern and Nawijn 1991: multi-operation jobs with time lags
- Identical to frame-based one-segmented self-suspension
- Decision version: schedule to meet uniform deadline D $\mathcal{NP}$-complete in the weak sense



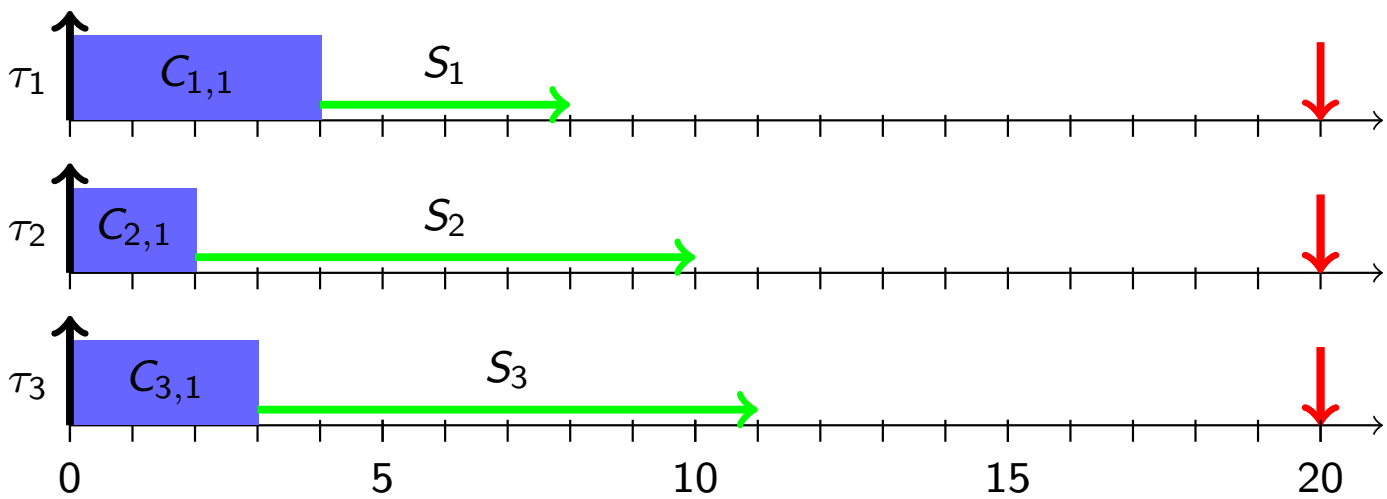
Results from Operations Research - Uniprocessor

- Kern and Nawijn 1991: multi-operation jobs with time lags
- Identical to frame-based one-segmented self-suspension
- Decision version: schedule to meet uniform deadline D $\mathcal{NP}$-complete in the weak sense
- Special cases in polynomial time:
  - All jobs have the same lag $\Rightarrow$ uniform suspension time



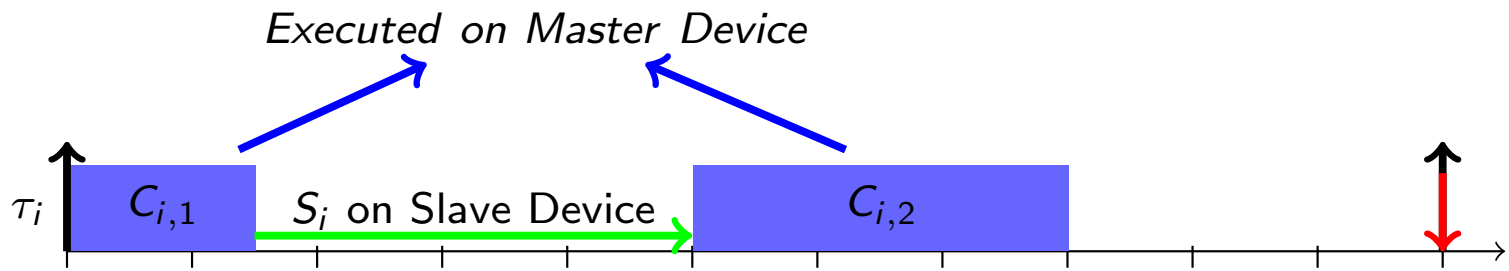
Results from Operations Research - Uniprocessor

- Kern and Nawijn 1991: multi-operation jobs with time lags
- Identical to frame-based one-segmented self-suspension
- Decision version: schedule to meet uniform deadline D $\mathcal{NP}$-complete in the weak sense
- Special cases in polynomial time:
  - All jobs have the same lag $\Rightarrow$ uniform suspension time
  - All jobs have only the first operation $\Rightarrow$ $C_{i,2} = 0$



Results from Operations Research - Master-Slave Model

- Kern and Nawijn 1991: multi-operation jobs with time lags
- Sahni 1995: master-slave scheduling model



Results from Operations Research - Master-Slave Model

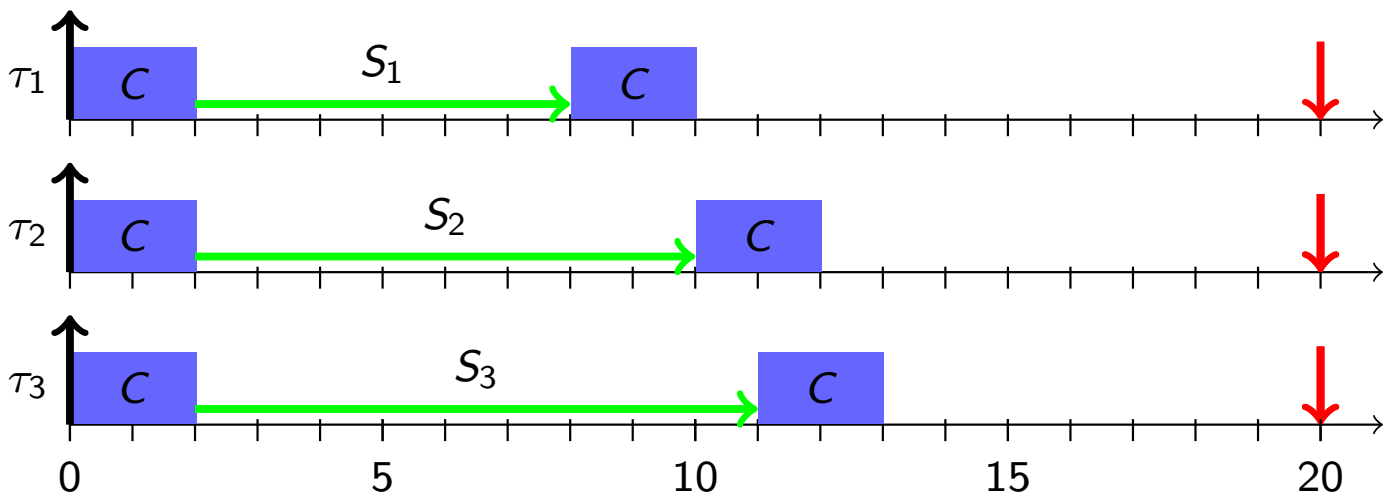
- Kern and Nawijn 1991: multi-operation jobs with time lags
- Sahni 1995: master-slave scheduling model
- Special case of two-stage flow shop with transfer lags
- $\mathcal{NP}$-hard in the strong sense

Results from Operations Research - Master-Slave Model

- Kern and Nawijn 1991: multi-operation jobs with time lags
- Sahni 1995: master-slave scheduling model
- Special case of two-stage flow shop with transfer lags
- $\mathcal{NP}$-hard in the strong sense
- Dell'Amico 1996: minimize makespan $\mathcal{NP}$-hard in the strong sense for both preemptive and non-preemptive scheduling

Results from Operations Research - Master-Slave Model

- Kern and Nawijn 1991: multi-operation jobs with time lags
- Sahni 1995: master-slave scheduling model
- Special case of two-stage flow shop with transfer lags
- $\mathcal{NP}$-hard in the strong sense
- Dell'Amico 1996: minimize makespan $\mathcal{NP}$-hard in the strong sense for both preemptive and non-preemptive scheduling
- Special case: Yu et al. 2004: $\mathcal{NP}$-hard in the strong sense even for unit time, i.e., $C_{i,1} = C_{i,2} = 1$



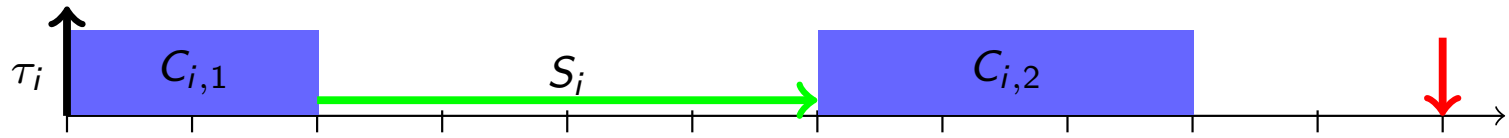
Results from Operations Research - Master-Slave Model

- Kern and Nawijn 1991: multi-operation jobs with time lags
- Sahni 1995: master-slave scheduling model
- Special case of two-stage flow shop with transfer lags
- $\mathcal{NP}$-hard in the strong sense
- Dell'Amico 1996: minimize makespan $\mathcal{NP}$-hard in the strong sense for both preemptive and non-preemptive scheduling
- Special case: Yu et al. 2004: $\mathcal{NP}$-hard in the strong sense even for unit time, i.e., $C_{i,1} = C_{i,2} = 1$

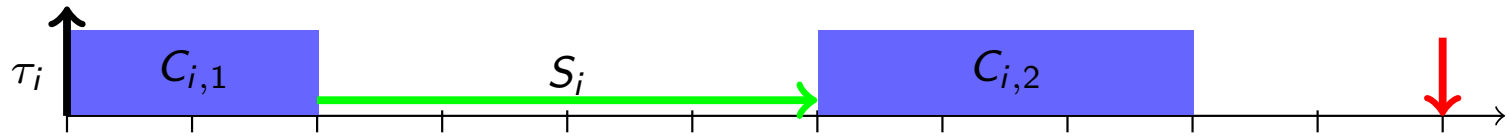
Conclusion

The computational complexity of the scheduler design problem of segmented self-suspension task systems is mainly due to the non-uniform *self-suspension time*.
Removing the periodicity and non-uniform execution times of the computation segments does not make the problem easier with respect to the computational complexity.

Speedup Factors - Coherent and Only Processor

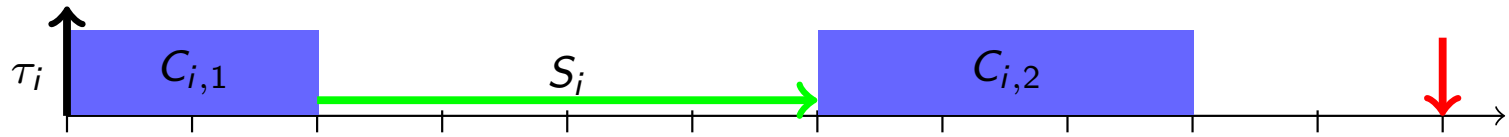


Speedup Factors - Coherent and Only Processor



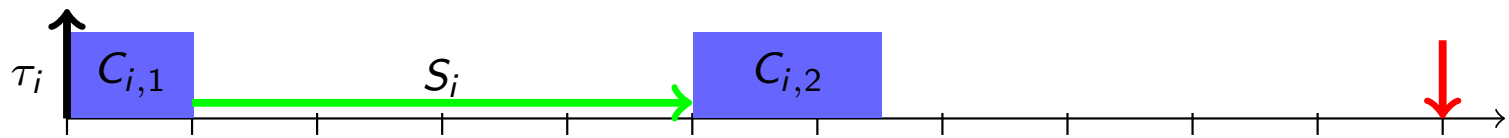
Suspension-coherent speedup (by 2)

Speedup Factors - Coherent and Only Processor



Suspension-coherent speedup (by 2)

Speedup only the processor (by 2)



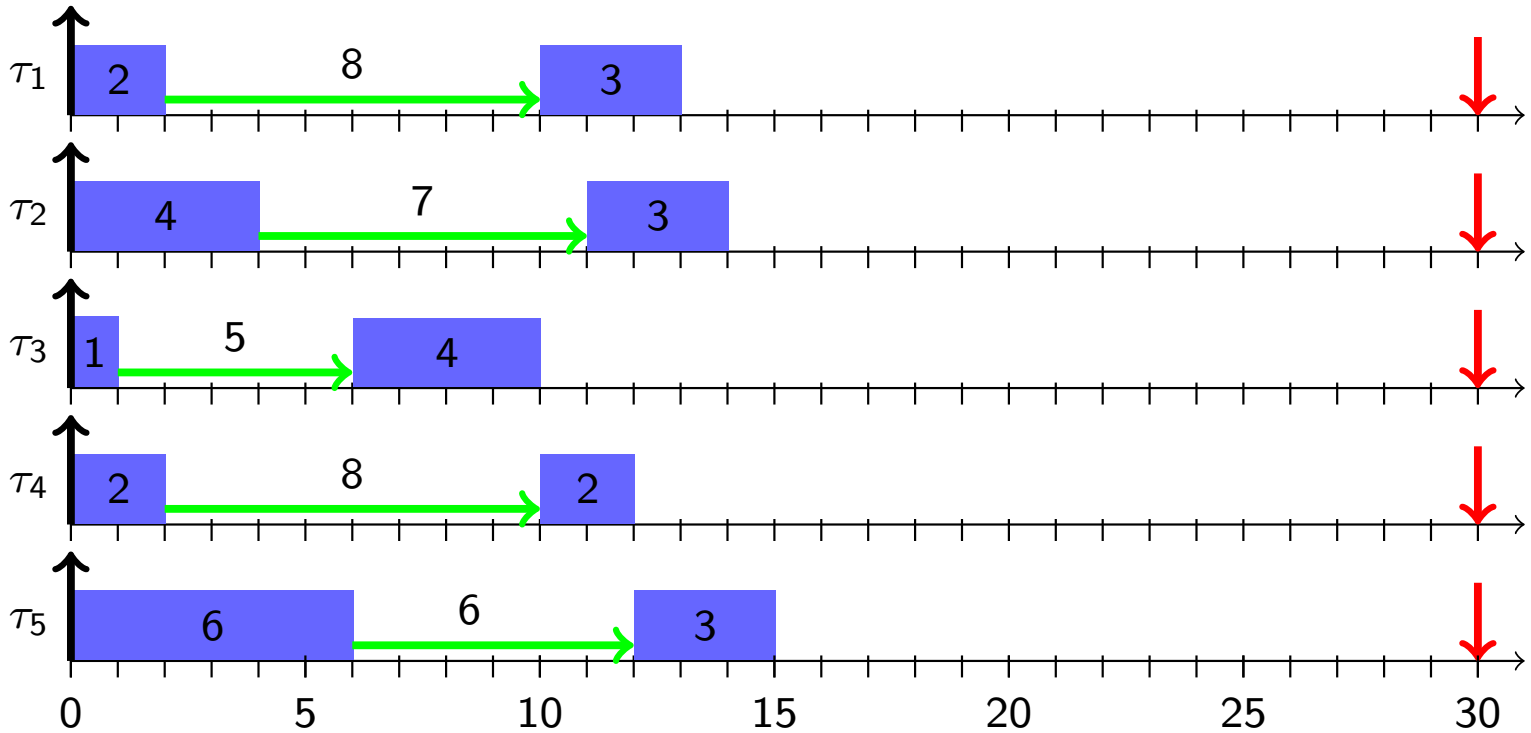
Speedup Factors - Current Status

Uniprocessor	segmented (one suspe.)	hybrid (one suspe.)	dynamic (multiple suspe.)
coherent	1.5 [42]	-	-
only processor	-	-	-

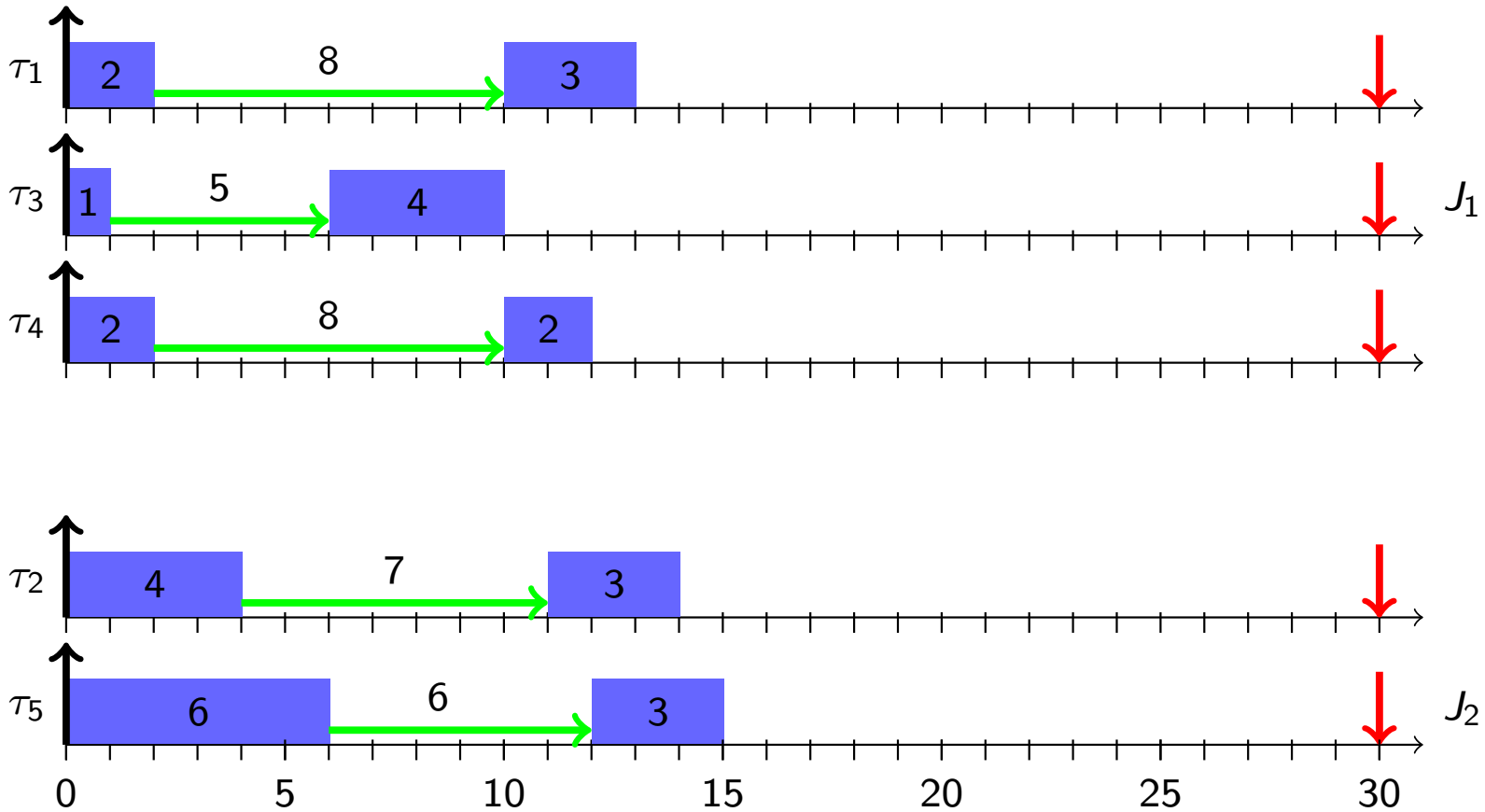
Multiprocessor *m* identical	segmented (one suspe.)	hybrid (one suspe.)	dynamic (multiple suspe.)
coherent	2 [42]	2 [42]	-
only processors	-	-	-

[42] Sahni and Vairaktarakis 1996

Algorithm by Sahni and Vairaktarakis

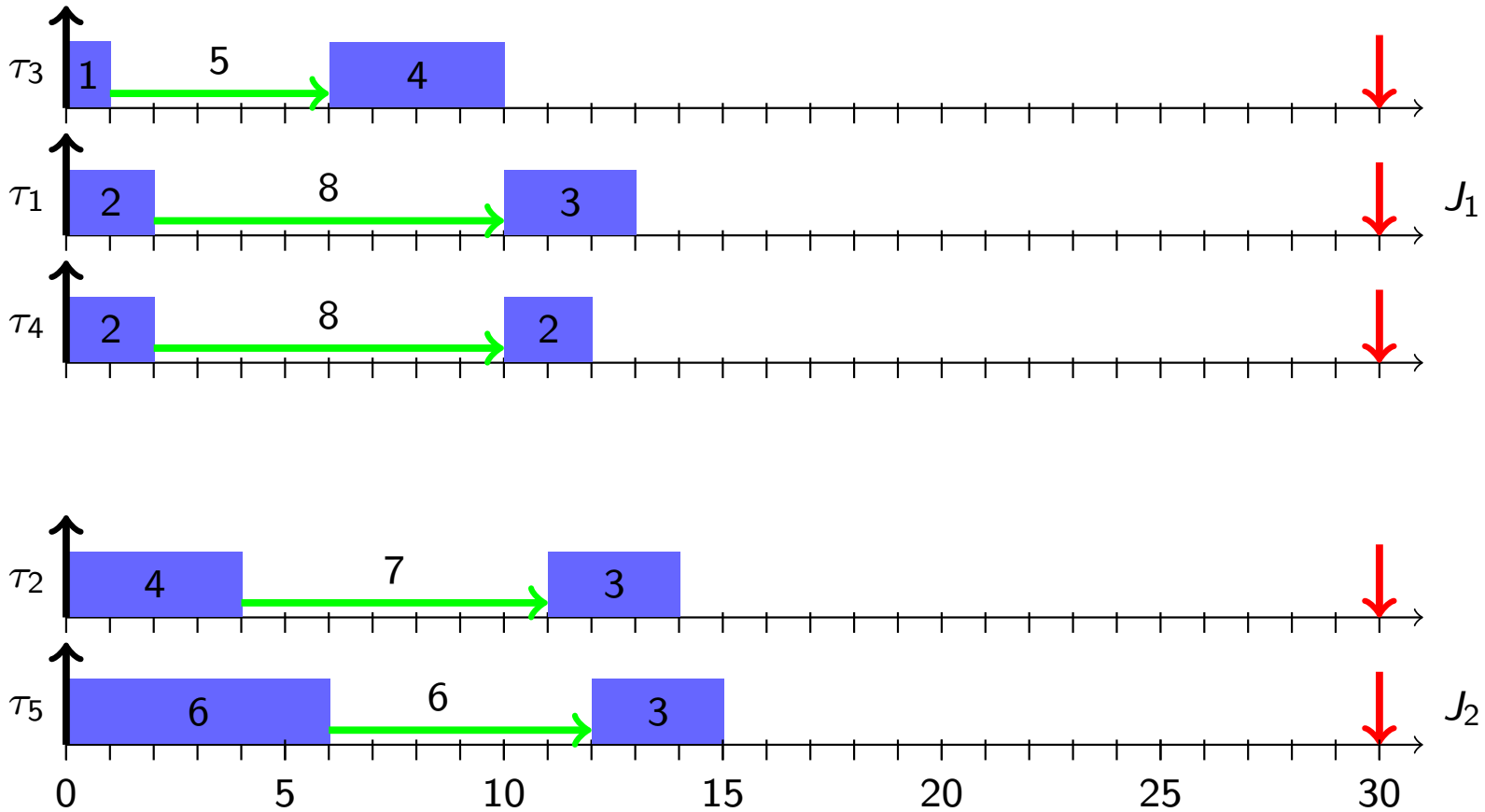


Algorithm by Sahni and Vairaktarakis



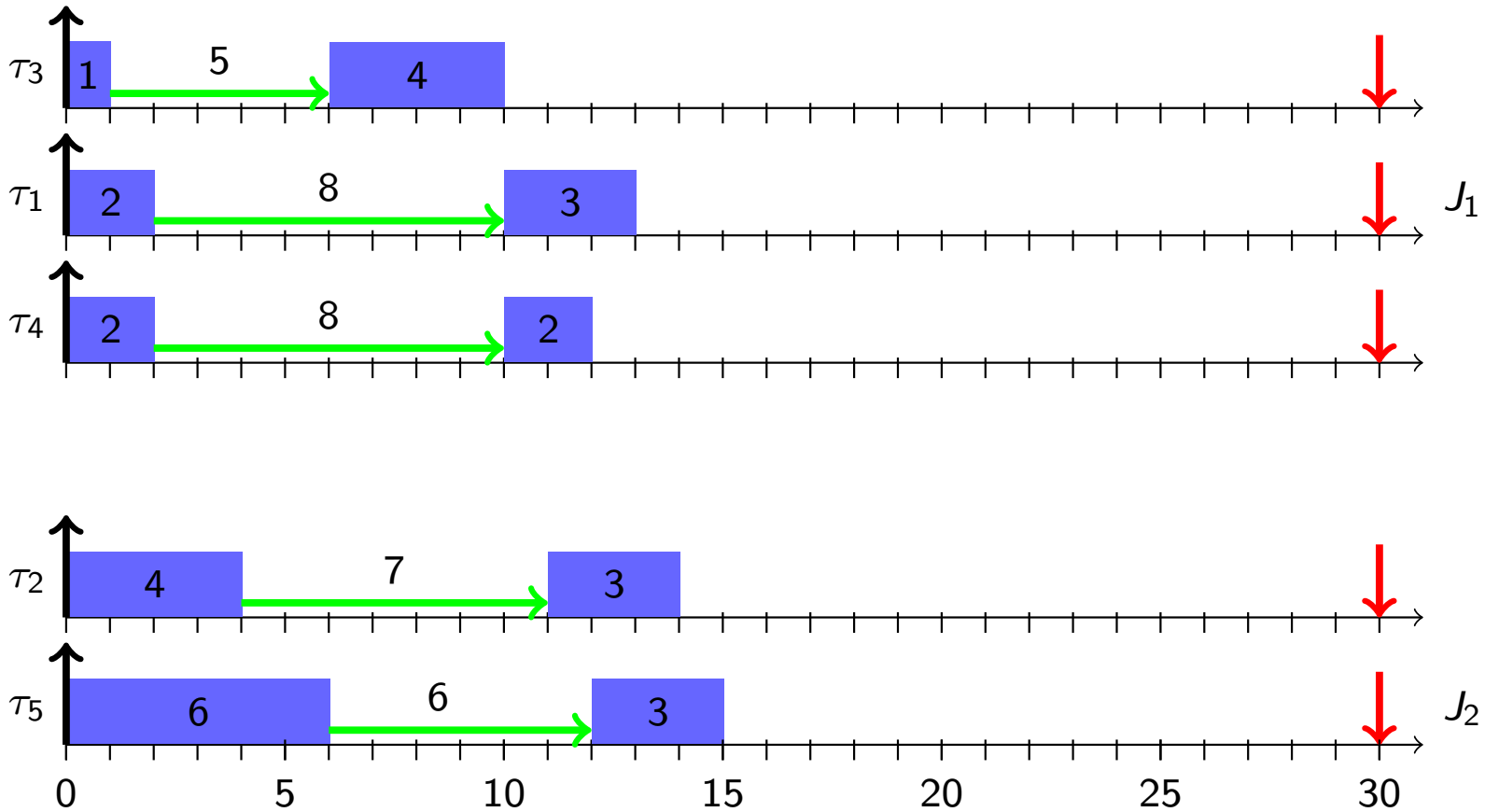
$J_1 : \ C_{i,1} \leq C_{i,2} \qquad J_2 : \ C_{i,1} > C_{i,2}$

Algorithm by Sahni and Vairaktarakis



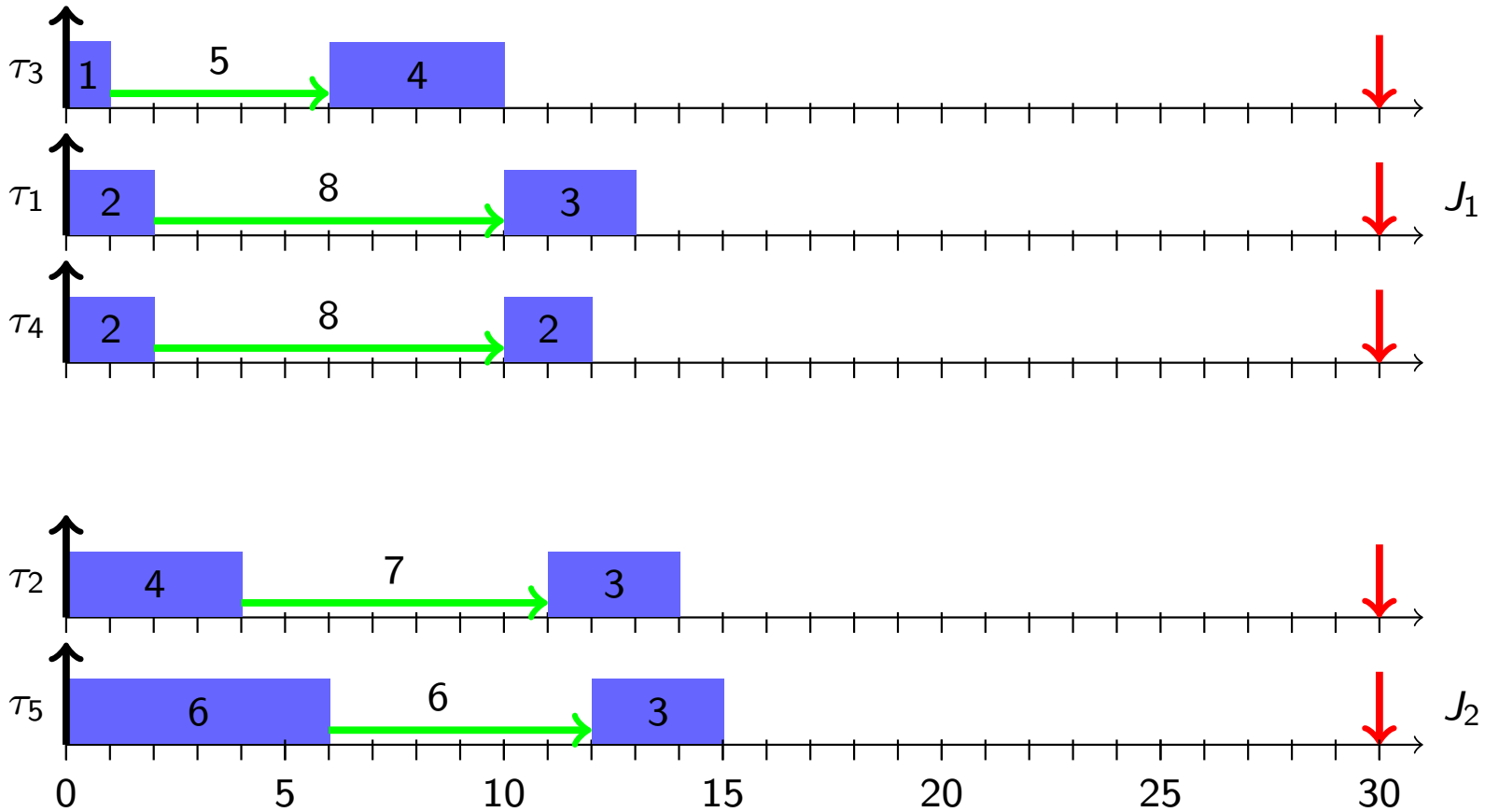
J_1: order non-decreasing to S_i - shortest suspension first

Algorithm by Sahni and Vairaktarakis



J_2: order non-increasing to S_i - longest suspension first

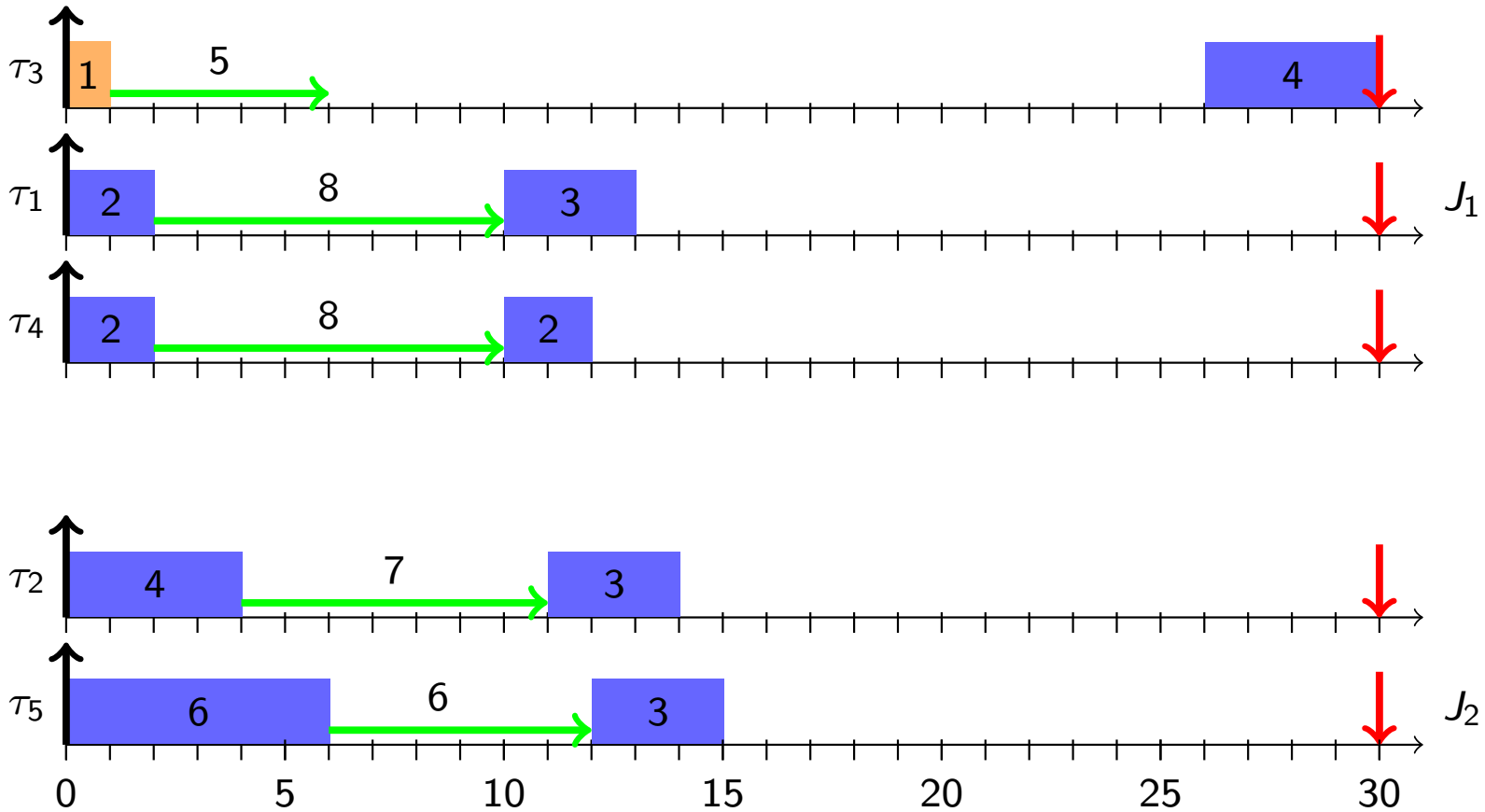
Algorithm by Sahni and Vairaktarakis



Schedule J_1, then J_2 according to order

Prioritize first computation segments

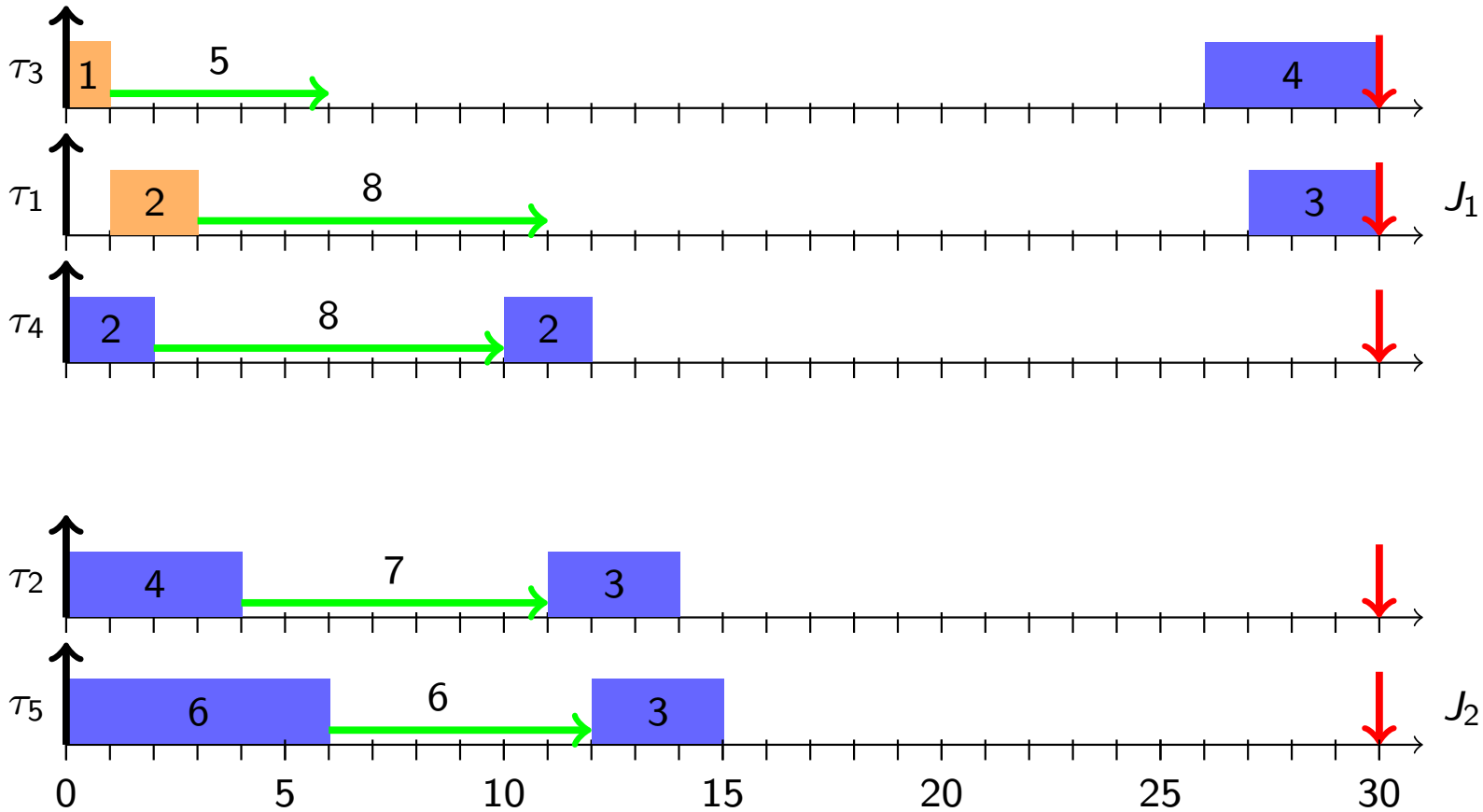
Algorithm by Sahni and Vairaktarakis



Schedule J_1, then J_2 according to order

Prioritize first computation segments

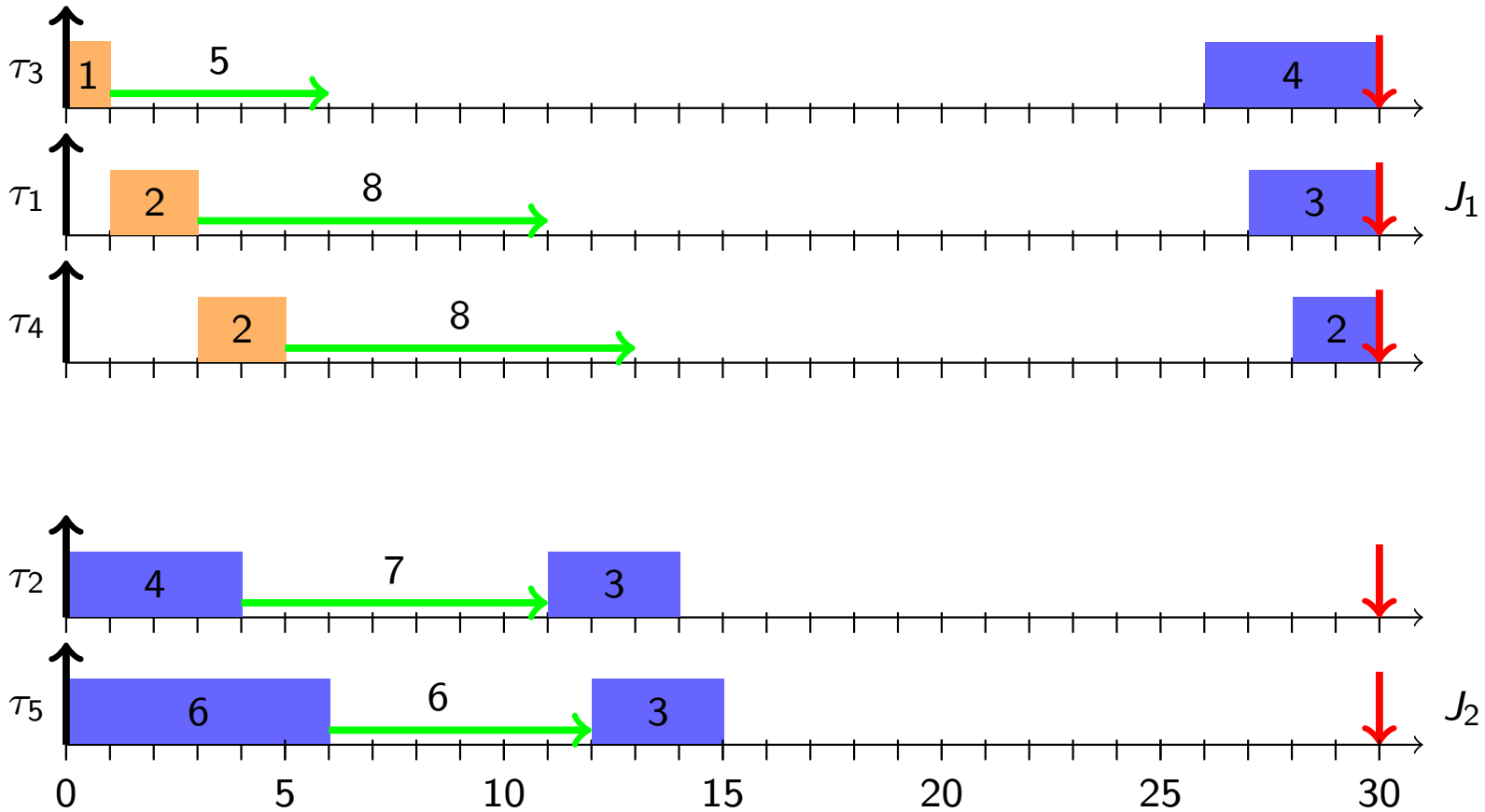
Algorithm by Sahni and Vairaktarakis



Schedule J_1, then J_2 according to order

Prioritize first computation segments

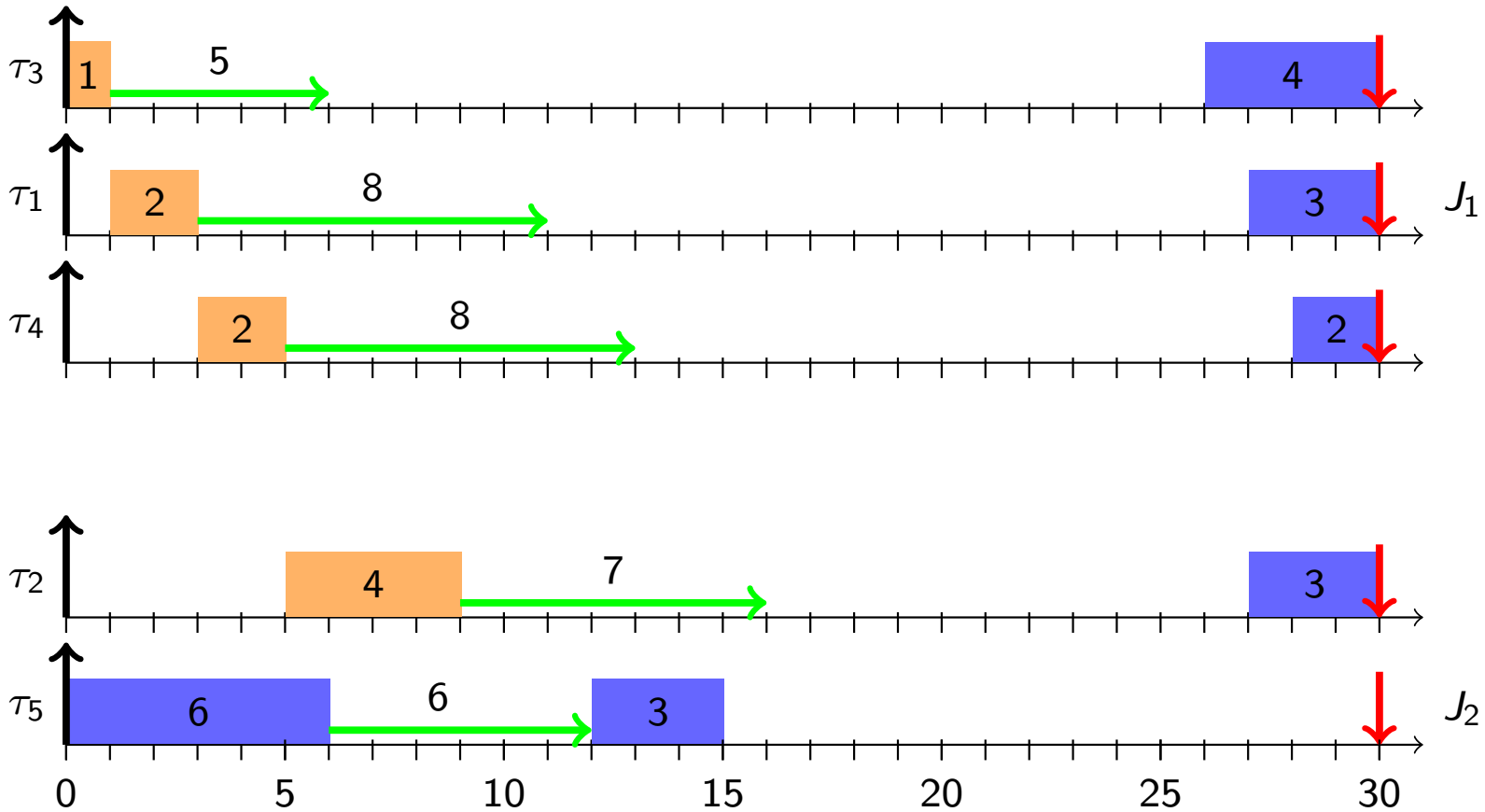
Algorithm by Sahni and Vairaktarakis



Schedule J_1, then J_2 according to order

Prioritize first computation segments

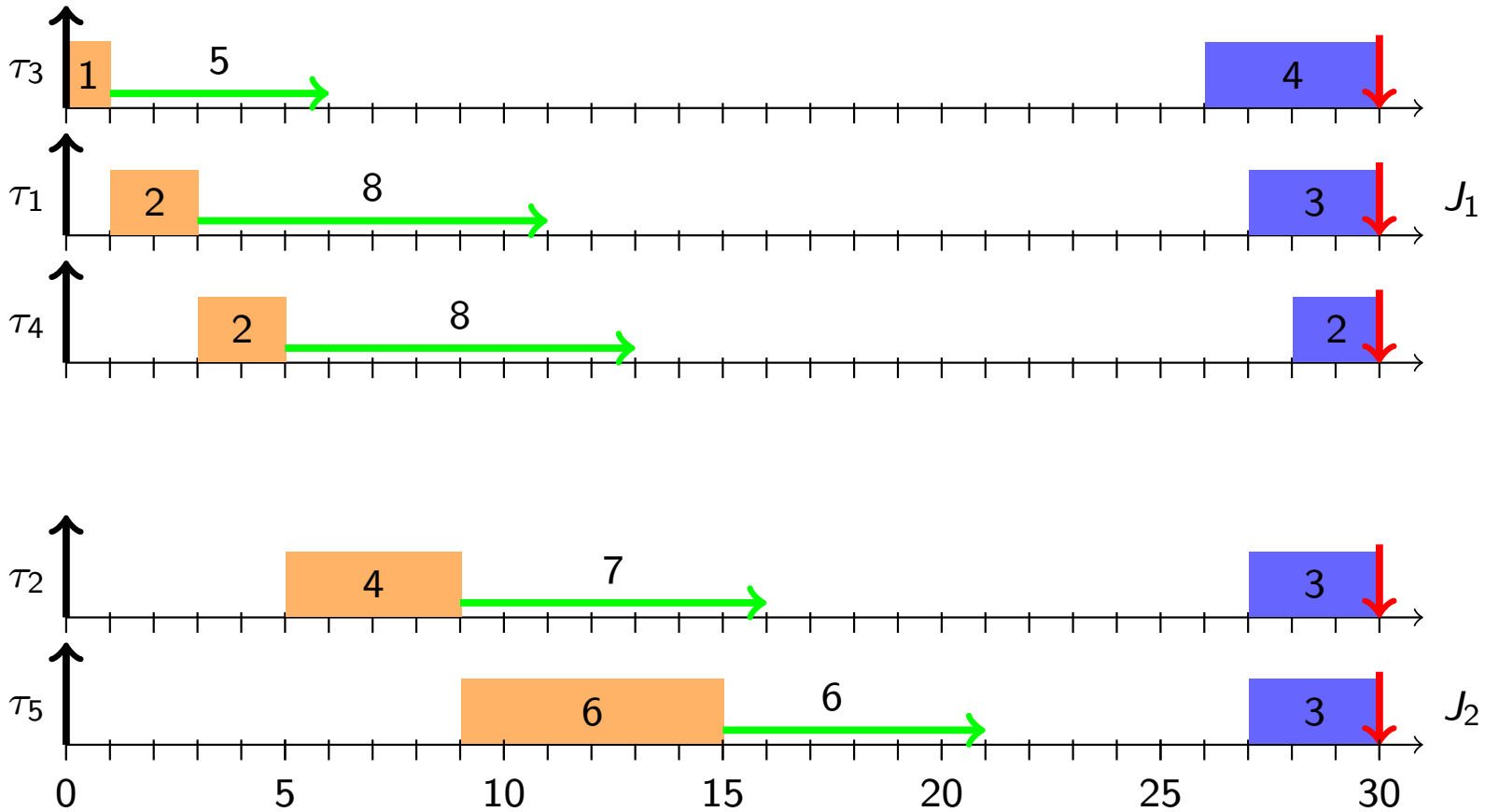
Algorithm by Sahni and Vairaktarakis



Schedule J_1, then J_2 according to order

Prioritize first computation segments

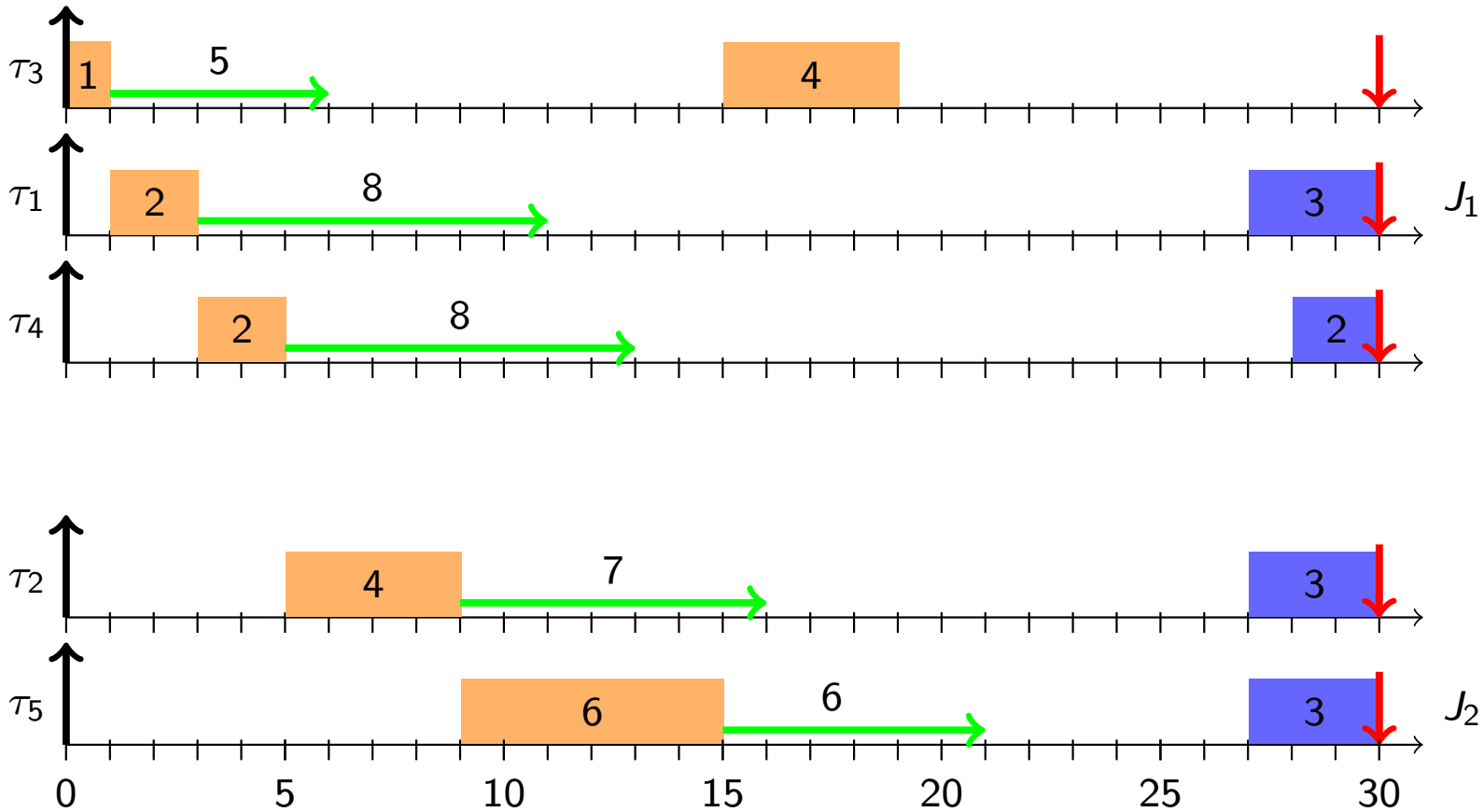
Algorithm by Sahni and Vairaktarakis



Schedule J_1, then J_2 according to order

Prioritize first computation segments

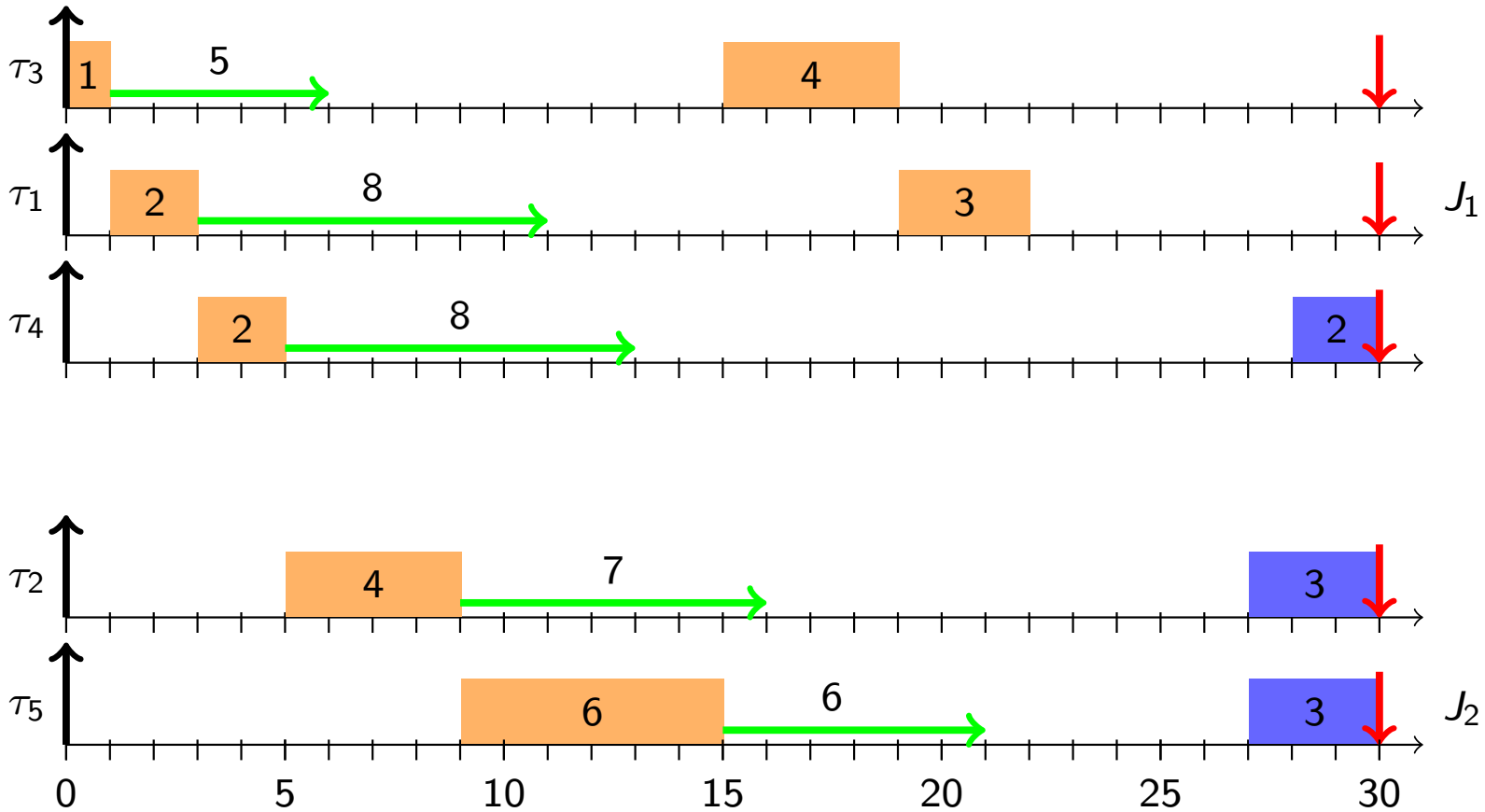
Algorithm by Sahni and Vairaktarakis



Schedule J_1, then J_2 according to order

Prioritize first computation segments

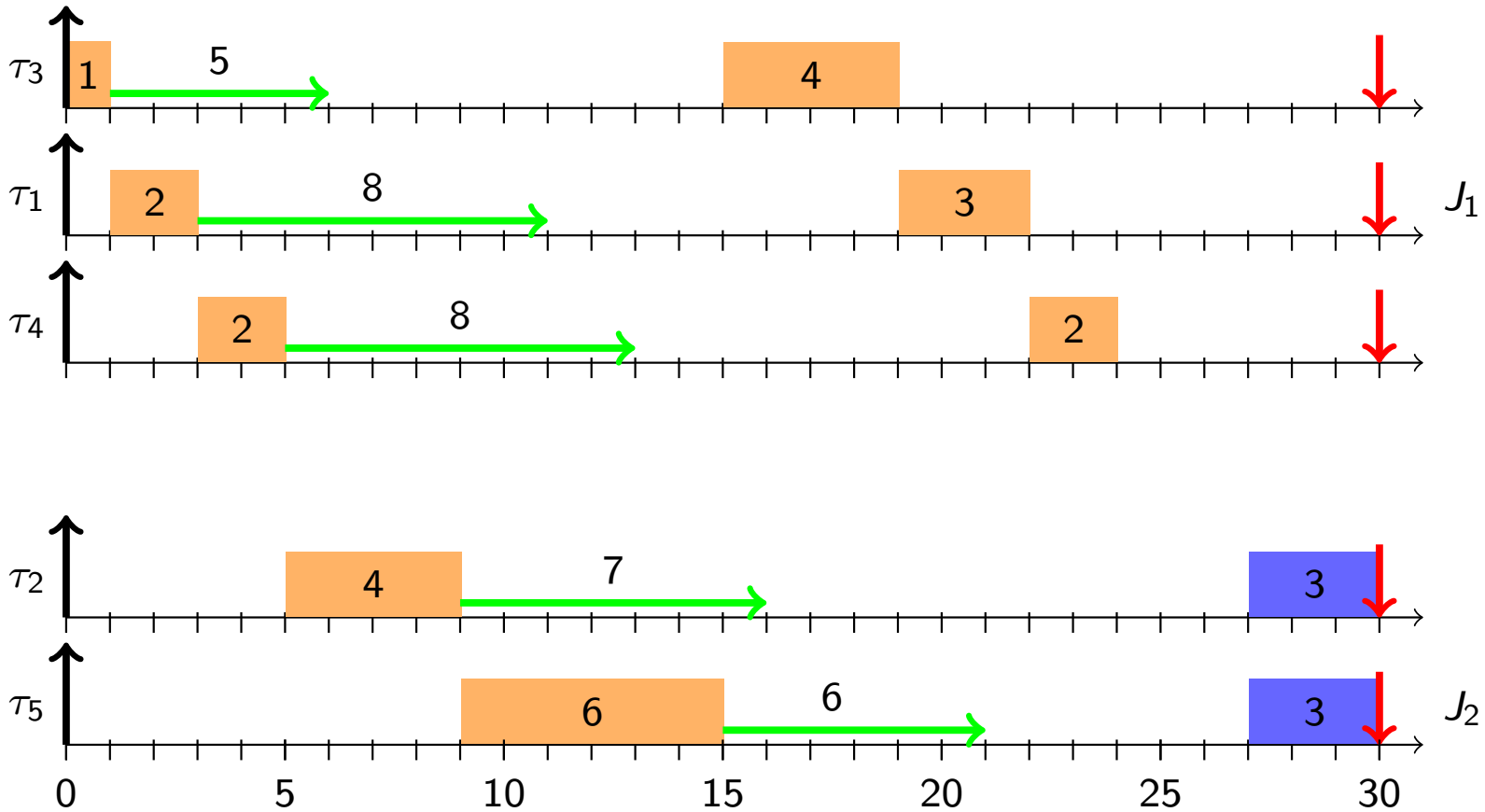
Algorithm by Sahni and Vairaktarakis



Schedule J_1, then J_2 according to order

Prioritize first computation segments

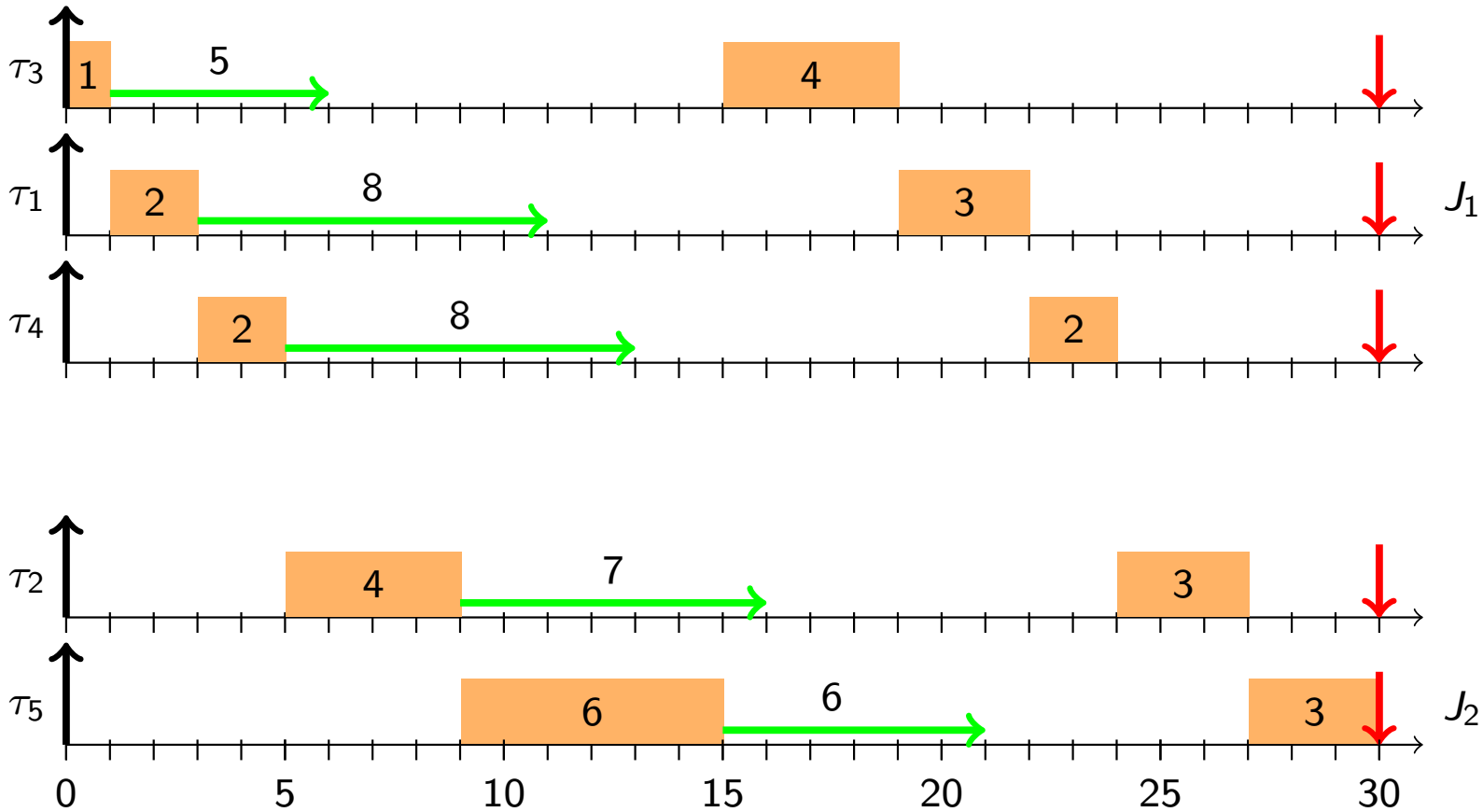
Algorithm by Sahni and Vairaktarakis



Schedule J_1, then J_2 according to order

Prioritize first computation segments

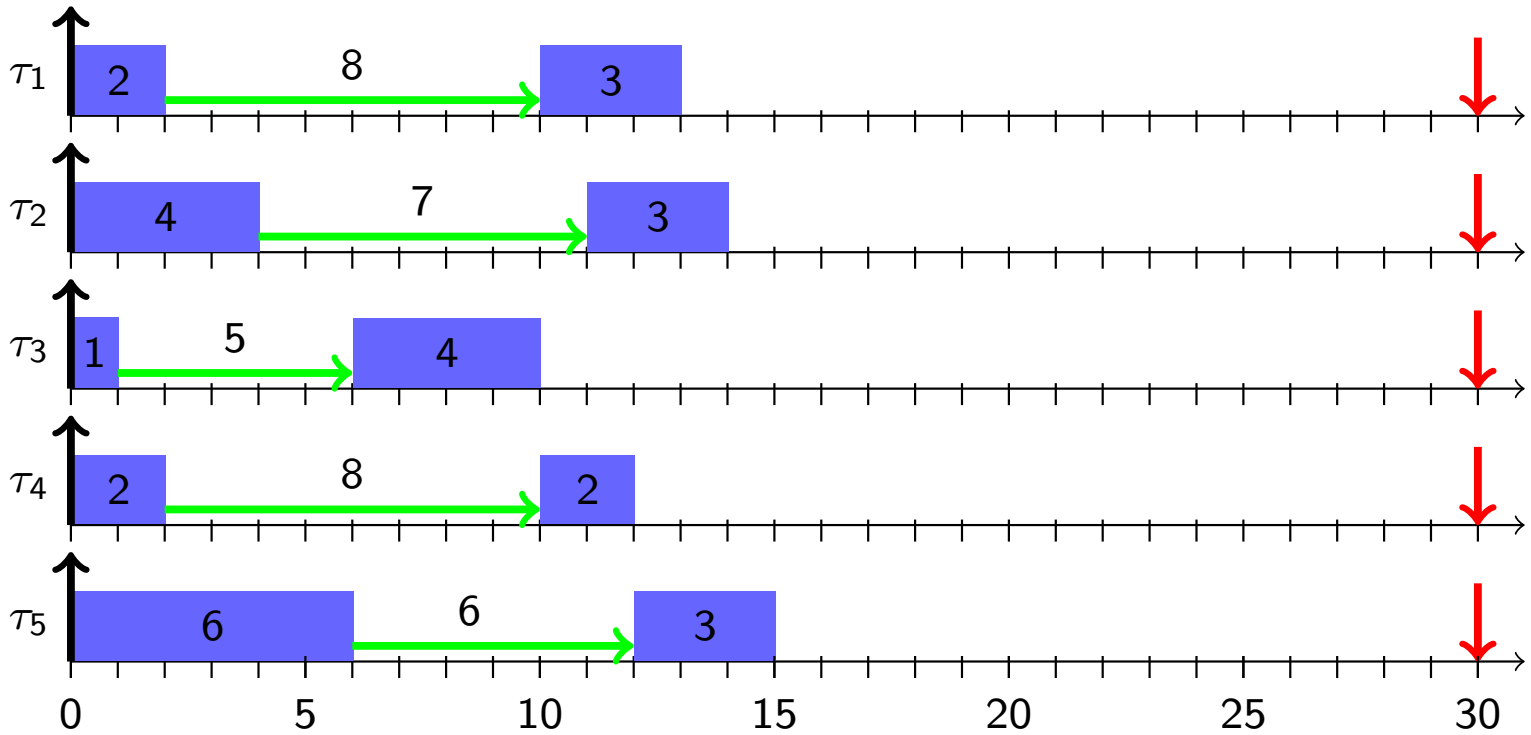
Algorithm by Sahni and Vairaktarakis



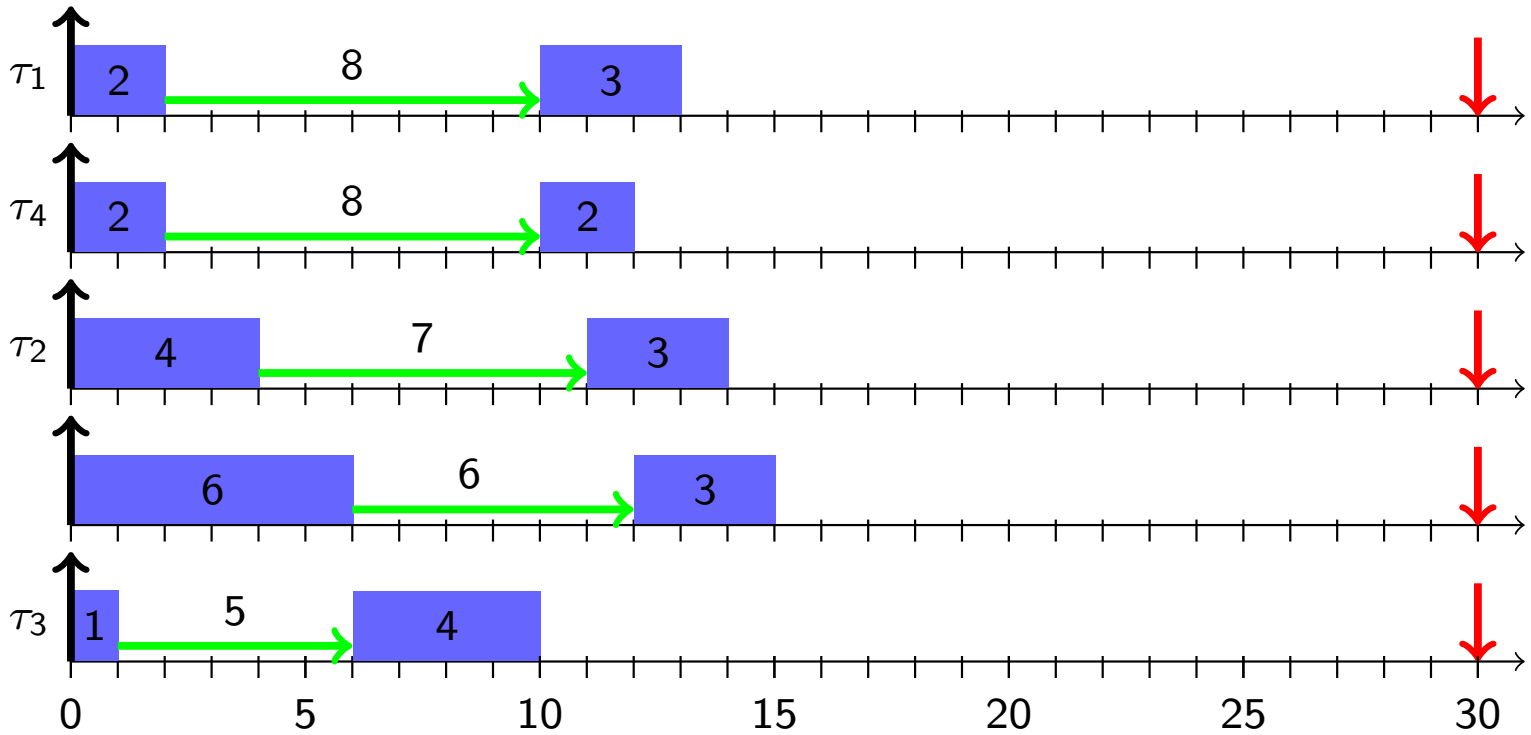
Schedule J_1, then J_2 according to order

Prioritize first computation segments

Longest Suspension First Algorithm

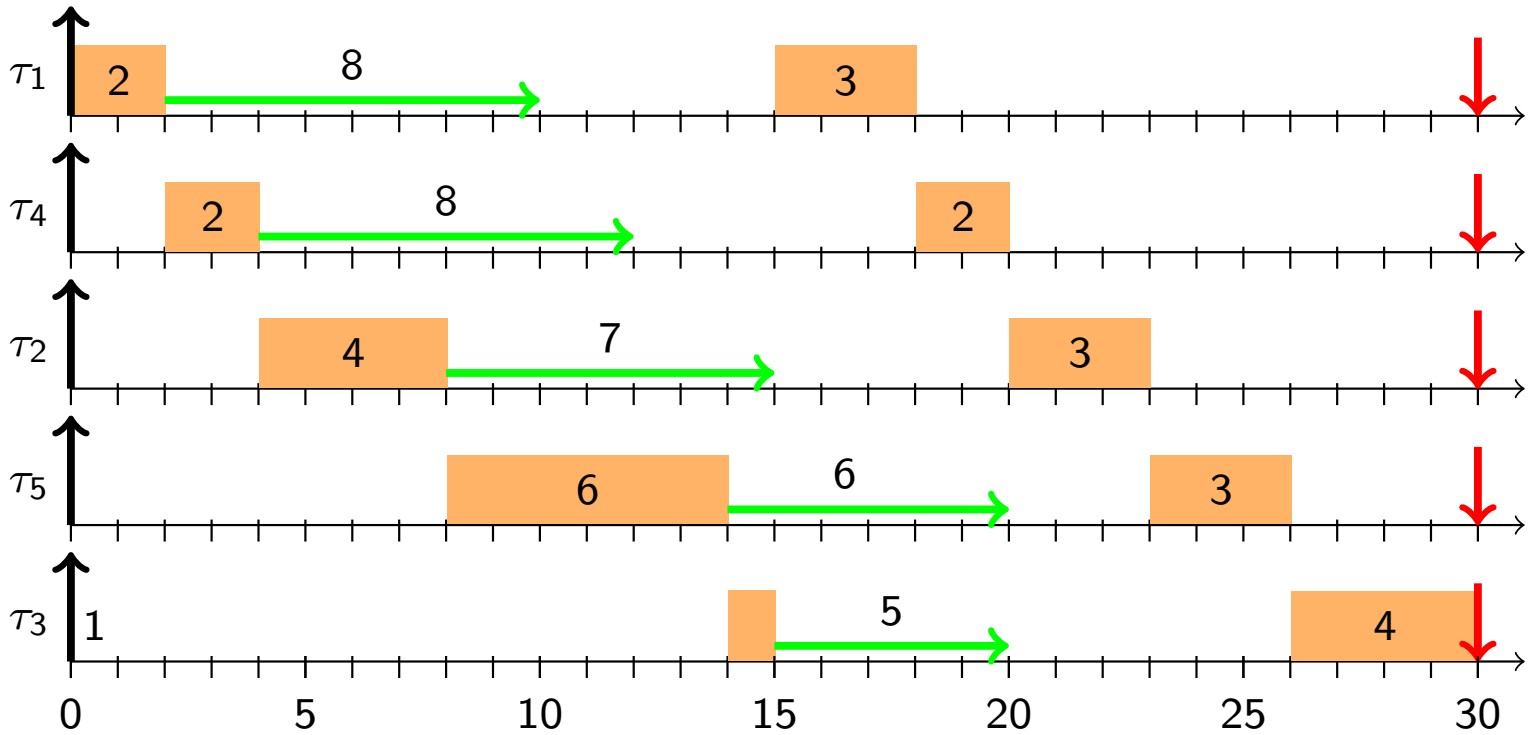


Longest Suspension First Algorithm



Order non-increasingly according to S_i

Longest Suspension First Algorithm



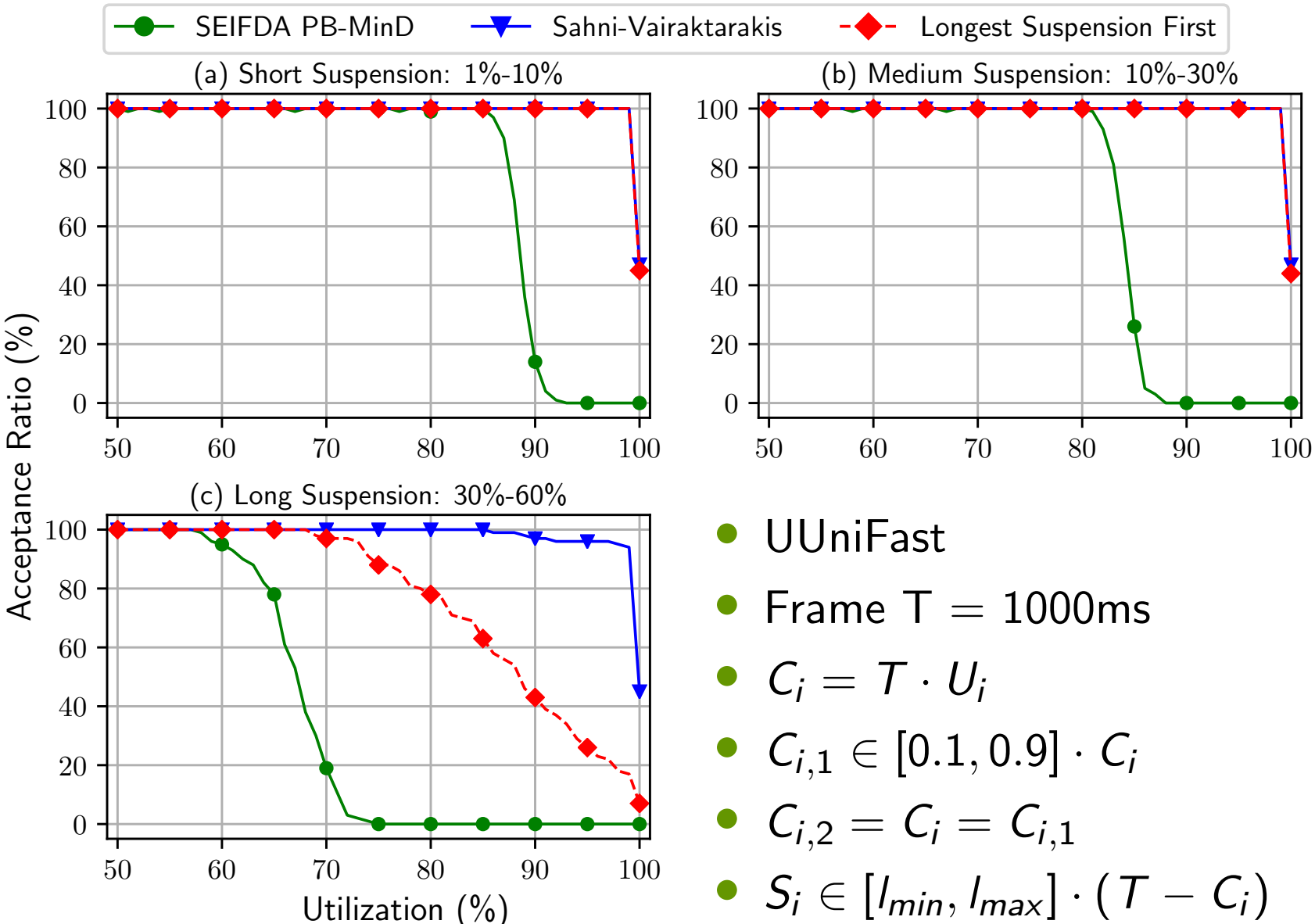
Schedule first computation segments according to order
Schedule second computation segments according to FCFS

Speedup Factors - New Results

Uniprocessor	segmented (one suspe.)	hybrid (one suspe.)	dynamic (multi sus.)
coherent	1.5 [42]	1.5 (Cor. 4.5)	2 (Thm. 4.7)
only processor	2 (Thm. 4.12)	2 (Thm. 4.13)	-
Multiprocessor m identical	segmented (one suspe.)	hybrid (one suspe.)	dynamic (multi sus.)
coherent	2 [42]	2 [42]	2 (Thm. 5.5)
only processors	$3 - \frac{1}{m}$ (T. 5.9)	$3 - \frac{1}{m}$ (T. 5.10)	-

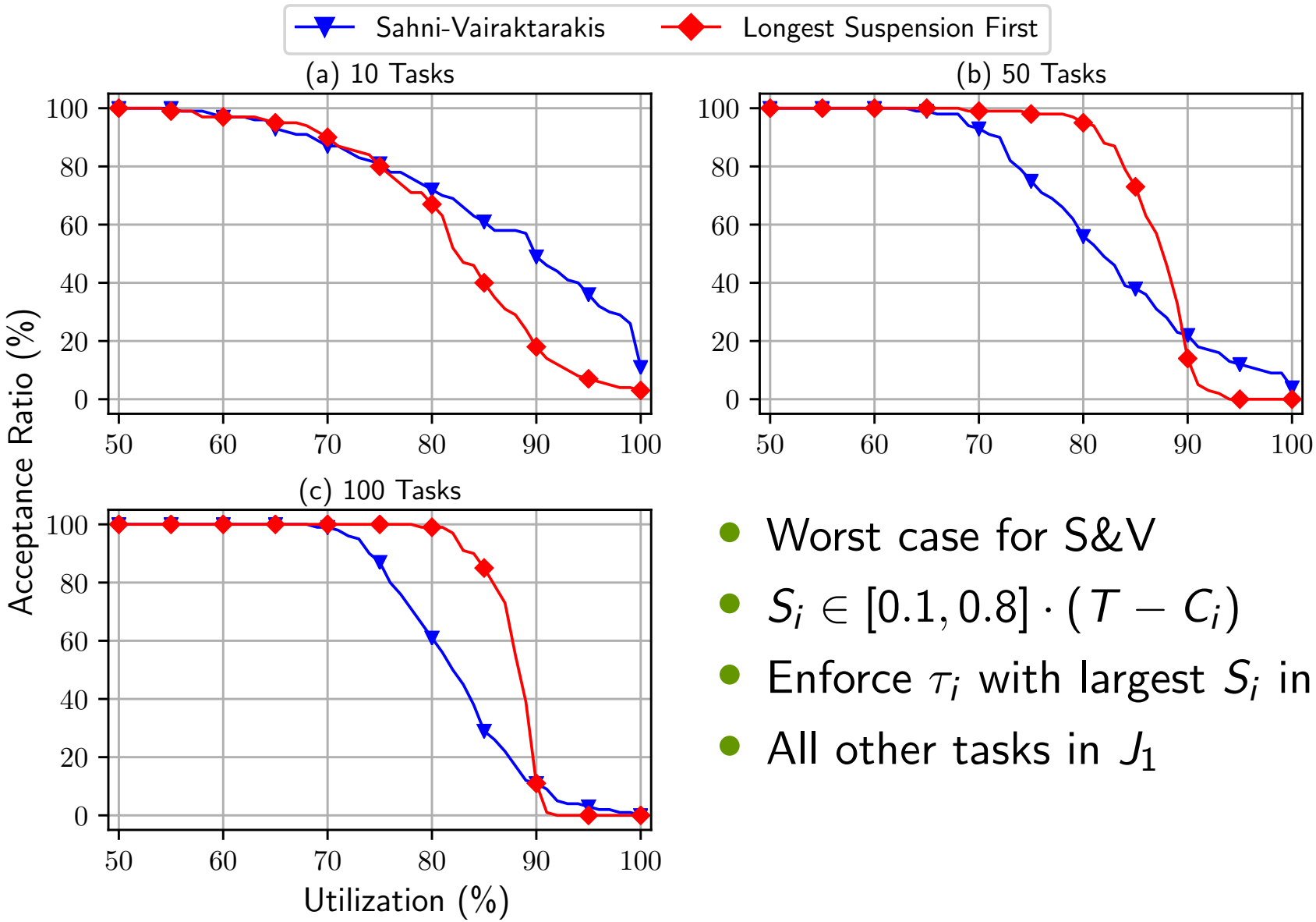
[42] Sahni and Vairaktarakis 1996

Evaluation



- UUniFast
- Frame T = 1000ms
- $C_i = T \cdot U_i$
- $C_{i,1} \in [0.1, 0.9] \cdot C_i$
- $C_{i,2} = C_i = C_{i,1}$
- $S_i \in [I_{min}, I_{max}] \cdot (T - C_i)$

Evaluation - Enforced Worst Case for S&V



- Worst case for S&V
- $S_i \in [0.1, 0.8] \cdot (T - C_i)$
- Enforce τ_i with largest S_i in J_2
- All other tasks in J_1

Conclusion

- Fundamental theoretical analysis

Conclusion

- Fundamental theoretical analysis
- Interesting results in the operations research community

Conclusion

- Fundamental theoretical analysis
- Interesting results in the operations research community
- Complexity for frame-based self-suspending task systems $\mathcal{NP}$-hard in the strong sense
  - Direct result of suspension behaviour
  - Removing periodicity or having unit time computation segments does not make the problem easier
  - Polynomial time algorithms for some special cases

Conclusion

- Fundamental theoretical analysis
- Interesting results in the operations research community
- Complexity for frame-based self-suspending task systems $\mathcal{NP}$-hard in the strong sense
  - Direct result of suspension behaviour
  - Removing periodicity or having unit time computation segments does not make the problem easier
  - Polynomial time algorithms for some special cases
- Coherent speedup and processor only speedup

Conclusion

- Fundamental theoretical analysis
- Interesting results in the operations research community
- Complexity for frame-based self-suspending task systems $\mathcal{NP}$-hard in the strong sense
  - Direct result of suspension behaviour
  - Removing periodicity or having unit time computation segments does not make the problem easier
  - Polynomial time algorithms for some special cases
- Coherent speedup and processor only speedup
- Tight speedup factors for simple algorithms

Conclusion

- Fundamental theoretical analysis
- Interesting results in the operations research community
- Complexity for frame-based self-suspending task systems $\mathcal{NP}$-hard in the strong sense
  - Direct result of suspension behaviour
  - Removing periodicity or having unit time computation segments does not make the problem easier
  - Polynomial time algorithms for some special cases
- Coherent speedup and processor only speedup
- Tight speedup factors for simple algorithms

Thank You!