Scheduling theory, part 1

Aperiodic jobs

Pontus Ekberg

Uppsala University

2018-09-14

Real-time systems?

Not necessarily fast…

…but always predictable

Real-time systems: theory & practice

Practice

Building systems

Software, middleware, RTOS…

Hardware design

⋮

Real-time systems: theory & practice

Theory

Verification

Scheduling theory

Control theory

⋮

Practice

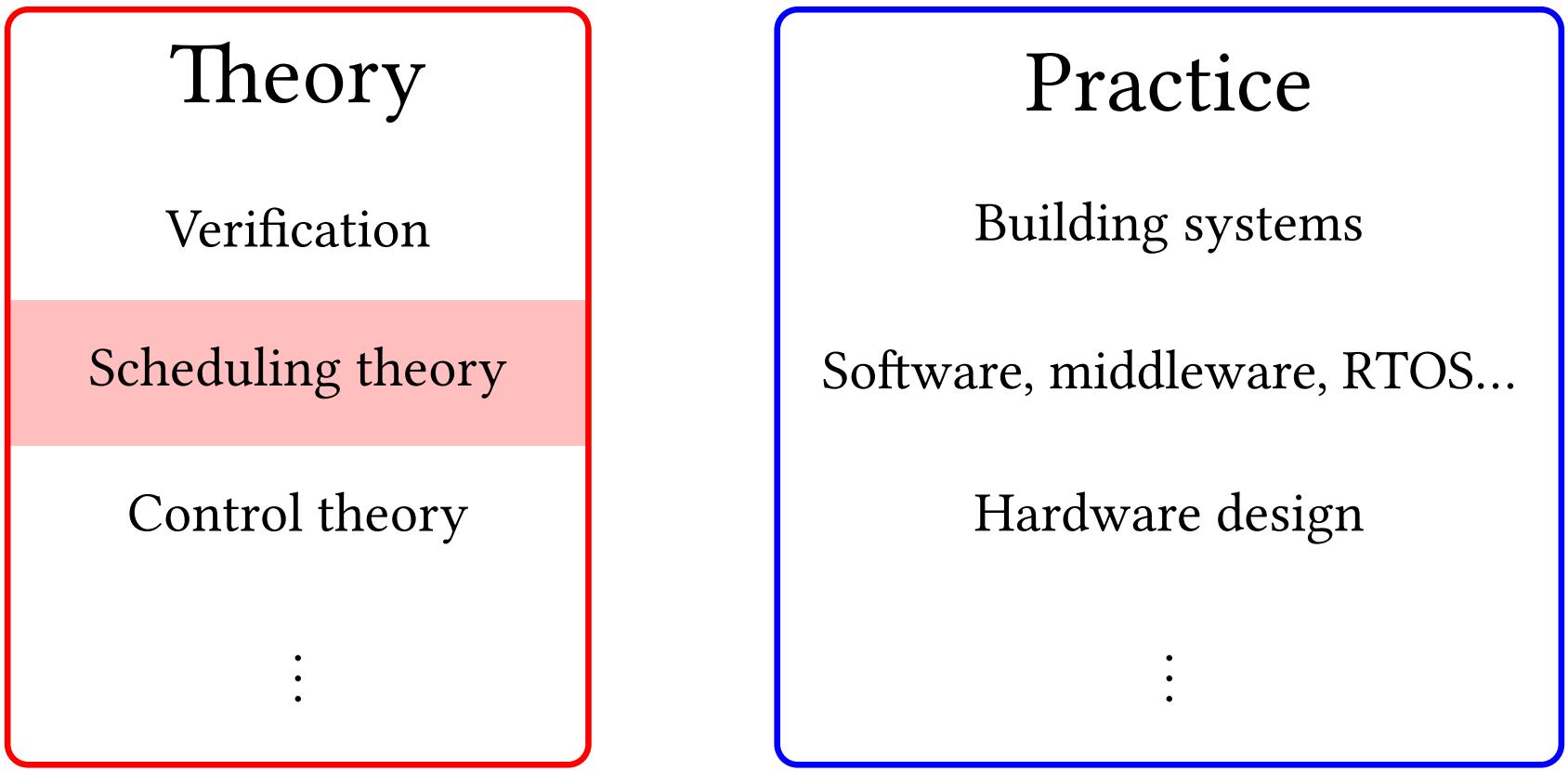
Building systems

Software, middleware, RTOS…

Hardware design

⋮

Real-time systems: theory & practice



Scheduling theory

Classical scheduling theory (e.g., in operations research)

generally deals with *finite* processes (job-shop, flow-shop &c.)

to *optimize* some metric.

Real-time scheduling theory

generally deals with *infinite* processes (control loops &c.)

to *guarantee* a safety specification.

The components of real-time scheduling theory

Task models:

Formalisms to specify workload and timing constraints

Scheduling algorithms:

Run-time strategies for scheduling workload

Analysis:

Offline methods for proving timing safety

The components of real-time scheduling theory

Task models:

Formalisms to specify workload and timing constraints

Scheduling algorithms:

Run-time strategies for scheduling workload

Analysis:

Offline methods for proving timing safety

Modeling: the art of abstraction

Modeling: the art of abstraction

Norbert Wiener, 1945

"*The best material model of a cat is another, or preferably the same, cat.*"

Modeling: the art of abstraction

Norbert Wiener, 1945

"*The best material model of a cat is another, or preferably the same, cat.*"

Edsger W. Dijkstra, 1972

"*[...] the purpose of abstracting is* **not** *to be vague, but to create a new semantic level in which one can be absolutely precise.*"

The job: a unit of work

A job j_i is given by a triple $(A_i, C_i, D_i) \in \mathbb{N}^3$, where

- A_i is the arrival time (or release time).
- C_i is the worst-case execution time (WCET), and
- D_i is the deadline.

The job: a unit of work

A job j_i is given by a triple $(A_i, C_i, D_i) \in \mathbb{N}^3$, where

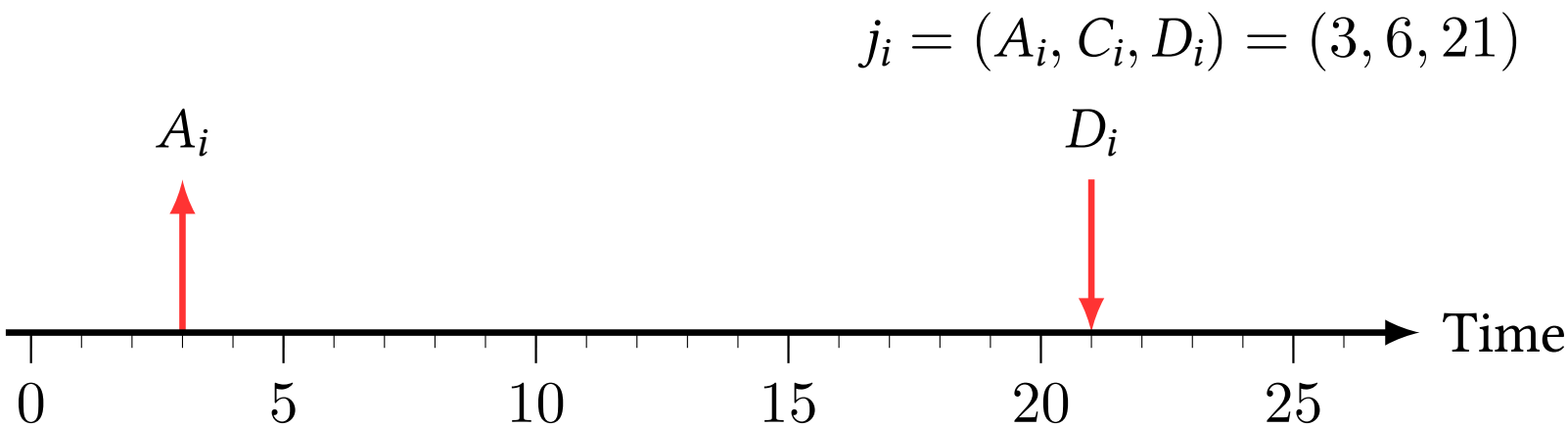
- A_i is the arrival time (or release time).
- C_i is the worst-case execution time (WCET), and
- D_i is the deadline.

$$j_i = (A_i, C_i, D_i) = (3, 6, 21)$$

The job: a unit of work

A job j_i is given by a triple $(A_i, C_i, D_i) \in \mathbb{N}^3$, where

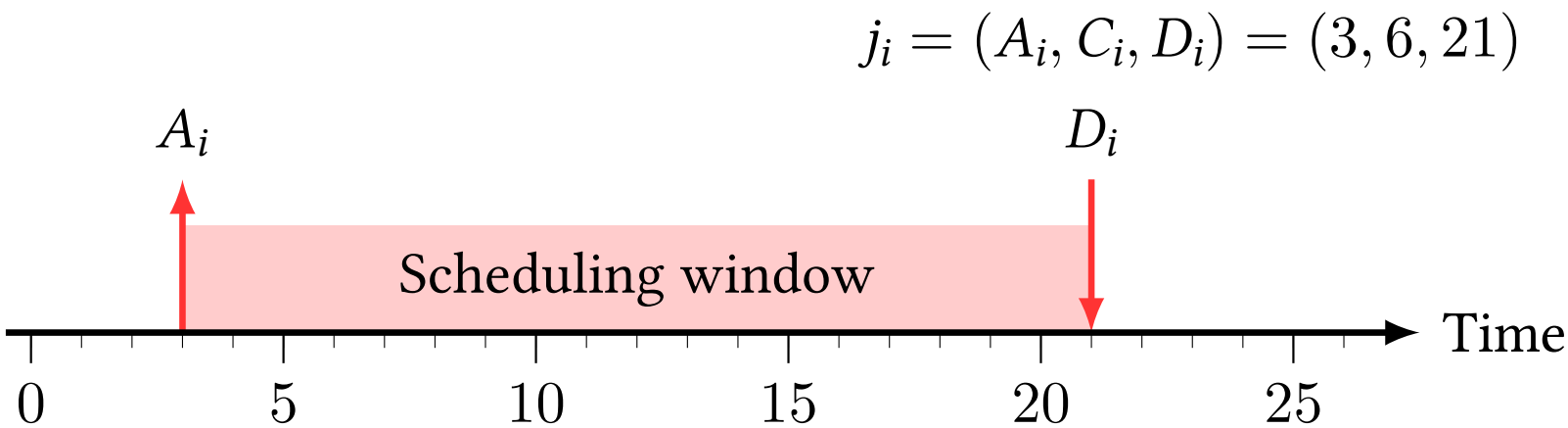
- A_i is the arrival time (or release time).
- C_i is the worst-case execution time (WCET), and
- D_i is the deadline.



The job: a unit of work

A job j_i is given by a triple $(A_i, C_i, D_i) \in \mathbb{N}^3$, where

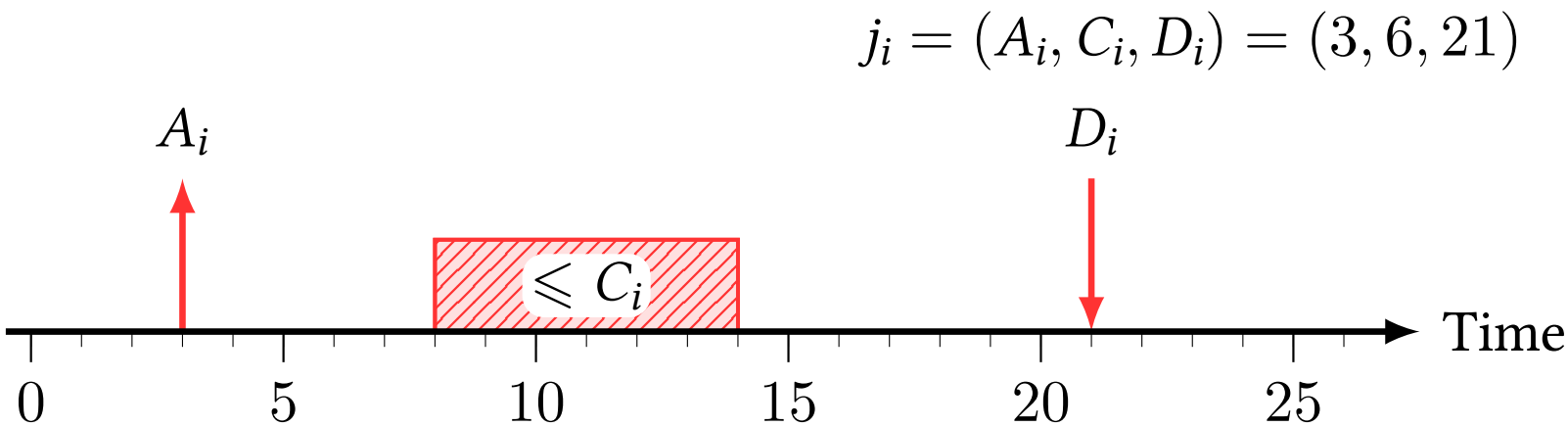
- A_i is the arrival time (or release time).
- C_i is the worst-case execution time (WCET), and
- D_i is the deadline.



The job: a unit of work

A job j_i is given by a triple $(A_i, C_i, D_i) \in \mathbb{N}^3$, where

- A_i is the arrival time (or release time).
- C_i is the worst-case execution time (WCET), and
- D_i is the deadline.



Scheduling a collection of independent jobs

The problem

Given a (multi-)set $\mathcal{J} = \{j_1, \ldots, j_n\}$ of n jobs, find a schedule where all jobs meet their deadlines.

Scheduling a collection of independent jobs

The problem

Given a (multi-)set $\mathcal{J} = \{j_1, \ldots, j_n\}$ of n jobs, find a schedule where all jobs meet their deadlines.

Assumptions

- All jobs are independent
- A single processor
- Fully preemptive *-or-* non-preemptive scheduling

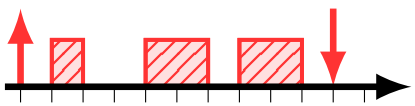
Scheduling a collection of independent jobs

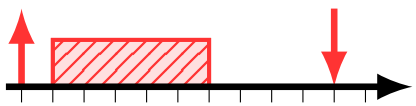
The problem

Given a (multi-)set $\mathcal{J} = \{j_1, \ldots, j_n\}$ of n jobs, find a schedule where all jobs meet their deadlines.

Assumptions

- All jobs are independent
- A single processor
- Fully preemptive *-or-* non-preemptive scheduling

Preemptive: 

Non-preemptive: 

The components of real-time scheduling theory

Task models:

Formalisms to specify workload and timing constraints

Scheduling algorithms:

Run-time strategies for scheduling workload

Analysis:

Offline methods for proving timing safety

The components of real-time scheduling theory

Task models:

Formalisms to specify workload and timing constraints

Scheduling algorithms:

Run-time strategies for scheduling workload

Analysis:

Offline methods for proving timing safety

Challenge

Schedule these jobs

$$\mathcal{J} = \left\{ \begin{array}{lll} (0,2,6), & (0,2,14), & (0,2,3), \\ (0,7,13), & (0,1,15), & (0,1,2) \end{array} \right\}$$

Note: All jobs in $\mathcal{J}$ have the same arrival time. Such jobs are called *synchronous*.

A solution

Earliest Deadline First (EDF)

Scheduling rule: *Choose among the ready jobs to execute the job with the earliest deadline (ties broken arbitrarily).*

A solution

Earliest Deadline First (EDF)

Scheduling rule: *Choose among the ready jobs to execute the job with the earliest deadline (ties broken arbitrarily).*

Note 1: We didn't need to use preemption!

A solution

Earliest Deadline First (EDF)

Scheduling rule: *Choose among the ready jobs to execute the job with the earliest deadline (ties broken arbitrarily).*

Note 1: We didn't need to use preemption!

Note 2: In this setting, EDF is also called *Earliest Due Date (EDD)* or *Jackson's algorithm*.

Is this a good general strategy?

Question

Is EDF a good strategy for all sets of synchronous jobs?

Is this a good general strategy?

Question

Is EDF a good strategy for all sets of synchronous jobs?

Theorem (Jackson, 1955)

If it is possible to schedule a set $\mathcal{J}$ of synchronous jobs, then $\mathcal{J}$ can also be scheduled by EDF.

Proof on black board!

Some important definitions

Schedulability

$\mathcal{J}$ is *$\mathcal{A}$-schedulable* iff scheduling algorithm $\mathcal{A}$ always generates a schedule without deadline misses for $\mathcal{J}$.

Feasibility

$\mathcal{J}$ is *feasible* iff there exists a scheduling algorithm $\mathcal{A}$ such that $\mathcal{J}$ is $\mathcal{A}$-schedulable.

Optimality

$\mathcal{A}$ is *optimal* iff all feasible $\mathcal{J}$ are also $\mathcal{A}$-schedulable.

The components of real-time scheduling theory

Task models:

Formalisms to specify workload and timing constraints

Scheduling algorithms:

Run-time strategies for scheduling workload

Analysis:

Offline methods for proving timing safety

The components of real-time scheduling theory

Task models:

Formalisms to specify workload and timing constraints

Scheduling algorithms:

Run-time strategies for scheduling workload

Analysis:

Offline methods for proving timing safety

Some simple analysis

How do we know if a set $\mathcal{J}$ of synchronous jobs is EDF-schedulable?

Schedulability test (Jackson, 1955)

Without loss of generality, let the indices of the jobs in $\mathcal{J} = \{j_1, \ldots, j_n\}$ be ordered by non-decreasing deadlines, and let all the arrival times be zero. Then, $\mathcal{J}$ is EDF-schedulable iff

$$\forall i, 1 \leqslant i \leqslant n : \quad \sum_{k=1}^{i} C_k \leqslant D_i.$$

Some simple analysis

How do we know if a set $\mathcal{J}$ of synchronous jobs is EDF-schedulable?

Schedulability test (Jackson, 1955)

Without loss of generality, let the indices of the jobs in $\mathcal{J} = \{j_1, \ldots, j_n\}$ be ordered by non-decreasing deadlines, and let all the arrival times be zero. Then, $\mathcal{J}$ is EDF-schedulable iff

$$\forall i, 1 \leqslant i \leqslant n : \quad \sum_{k=1}^{i} C_k \leqslant D_i.$$

Proof?

Now let's do arbitrary arrival times

Question

For a set $\mathcal{J}$ of jobs with *asynchronous* arrival times, does it matter if we allow preemptions?

Now let's do arbitrary arrival times

Question

For a set $\mathcal{J}$ of jobs with *asynchronous* arrival times, does it matter if we allow preemptions?

YES!

The preemptive case

Question

Is EDF still optimal for preemptive scheduling of job sets with asynchronous arrival times?

The preemptive case

Question

Is EDF still optimal for preemptive scheduling of job sets with asynchronous arrival times?

Theorem (Dertouzos, 1973)

EDF is optimal for scheduling *any* set of independent jobs on a single preemptive processor.

Proof of the optimality of EDF

First, let's define an important function.

The demand bound function

For a job j_i and time instants t_1 and t_2, where $0 \leqslant t_1 \leqslant t_2$, let the *demand bound function* $\mathrm{dbf}(j_i, t_1, t_2)$ be defined as

$$\mathrm{dbf}(j_i, t_1, t_2) = \begin{cases} C_i, & \text{if } t_1 \leqslant A_i \text{ and } D_i \leqslant t_2 \\ 0, & \text{otherwise.} \end{cases}$$

For a job set $\mathcal{J}$, let $\mathrm{dbf}(\mathcal{J}, t_1, t_2)$ be defined as

$$\mathrm{dbf}(\mathcal{J}, t_1, t_2) = \sum_{j_i \in \mathcal{J}} \mathrm{dbf}(j_i, t_1, t_2).$$

Proof of the optimality of EDF

Feasibility test / EDF-schedulability test

A job set $\mathcal{J}$ is feasible on a single preemptive processor iff

$$\forall t_1, t_2 \text{ such that } 0 \leqslant t_1 \leqslant t_2 : \quad \text{dbf}(\mathcal{J}, t_1, t_2) \leqslant t_2 - t_1.$$

Proof of the optimality of EDF

Feasibility test / EDF-schedulability test

A job set $\mathcal{J}$ is feasible on a single preemptive processor iff

$$\forall t_1, t_2 \text{ such that } 0 \leqslant t_1 \leqslant t_2 : \quad \text{dbf}(\mathcal{J}, t_1, t_2) \leqslant t_2 - t_1.$$

Let's prove the validity of this test and the optimality of EDF in one go!

Proof of the optimality of EDF

Feasibility test / EDF-schedulability test

A job set $\mathcal{J}$ is feasible on a single preemptive processor iff

$$\forall t_1, t_2 \text{ such that } 0 \leqslant t_1 \leqslant t_2 : \quad \text{dbf}(\mathcal{J}, t_1, t_2) \leqslant t_2 - t_1.$$

Let's prove the validity of this test and the optimality of EDF in one go!

Step 1: Prove that the condition is *necessary*.

Proof of the optimality of EDF

Feasibility test / EDF-schedulability test

A job set $\mathcal{J}$ is feasible on a single preemptive processor iff

$$\forall t_1, t_2 \text{ such that } 0 \leqslant t_1 \leqslant t_2 : \quad \text{dbf}(\mathcal{J}, t_1, t_2) \leqslant t_2 - t_1.$$

Let's prove the validity of this test and the optimality of EDF in one go!

Step 1: Prove that the condition is *necessary*.
Step 2: Prove that it is *sufficient* for EDF.

Proof of the optimality of EDF

Feasibility test / EDF-schedulability test

A job set $\mathcal{J}$ is feasible on a single preemptive processor iff

$$\forall t_1, t_2 \text{ such that } 0 \leqslant t_1 \leqslant t_2 : \quad \text{dbf}(\mathcal{J}, t_1, t_2) \leqslant t_2 - t_1.$$

Let's prove the validity of this test and the optimality of EDF in one go!

Step 1: Prove that the condition is *necessary*.
Step 2: Prove that it is *sufficient* for EDF.

Conclusion: The test is valid *and* EDF is optimal. (Why?)

Proof of the optimality of EDF

Feasibility test / EDF-schedulability test

A job set $\mathcal{J}$ is feasible on a single preemptive processor iff

$$\forall t_1, t_2 \text{ such that } 0 \leqslant t_1 \leqslant t_2 : \quad \text{dbf}(\mathcal{J}, t_1, t_2) \leqslant t_2 - t_1.$$

Let's prove the validity of this test and the optimality of EDF in one go!

Step 1: Prove that the condition is *necessary*.
Step 2: Prove that it is *sufficient* for EDF.

Conclusion: The test is valid *and* EDF is optimal. (Why?)

Proofs on the blackboard!

The preemptive case: conclusions

Theorem (Dertouzos, 1973)

EDF is optimal for scheduling *any* set of independent jobs on a single preemptive processor.

Feasibility test / EDF-schedulability test

A job set $\mathcal{J}$ is feasible on a single preemptive processor iff

$$\forall t_1, t_2 \text{ such that } 0 \leqslant t_1 \leqslant t_2 : \quad \text{dbf}(\mathcal{J}, t_1, t_2) \leqslant t_2 - t_1.$$

(It is enough to consider values of t_1 picked from the arrival times and values of t_2 picked from the deadlines.)

The non-preemptive case

Question

Is EDF still optimal for non-preemptive scheduling of job sets with asynchronous arrival times?

The non-preemptive case

Question

Is EDF still optimal for non-preemptive scheduling of job sets with asynchronous arrival times?

NO!

The non-preemptive case

Question

Is EDF still optimal for non-preemptive scheduling of job sets with asynchronous arrival times?

NO!

(But it is still optimal if idling is forbidden! Proof omitted.)

The non-preemptive case

Question

How can we then find the best schedule for a job set $\mathcal{J}$ of non-preemptive jobs?

The non-preemptive case

Question

How can we then find the best schedule for a job set $\mathcal{J}$ of non-preemptive jobs?

One possible approach

Step 1: Assume the execution time of all jobs is the WCET.
Step 2: Try all possible orderings of executing the jobs.

The non-preemptive case

Question

How can we then find the best schedule for a job set $\mathcal{J}$ of non-preemptive jobs?

One possible approach

Step 1: Assume the execution time of all jobs is the WCET.
Step 2: Try all possible orderings of executing the jobs.

Good news: This works!

The non-preemptive case

Question

How can we then find the best schedule for a job set $\mathcal{J}$ of non-preemptive jobs?

One possible approach

Step 1: Assume the execution time of all jobs is the WCET.
Step 2: Try all possible orderings of executing the jobs.

Good news: This works!

Bad news: There are $n!$ orderings of n jobs.

The non-preemptive case

Question

Is there an *efficient* way to find a valid schedule for a set of non-preemptive jobs?

The non-preemptive case

Question

Is there an *efficient* way to find a valid schedule for a set of non-preemptive jobs?

Probably not: This problem is strongly NP-hard.
(There is a simple reduction from 3-PARTITION.)

The non-preemptive case

Question

Is there an *efficient* way to find a valid schedule for a set of non-preemptive jobs?

Probably not: This problem is strongly NP-hard.
(There is a simple reduction from 3-PARTITION.)

In practice, various heuristic search techniques could work well.