Lecture: Workload Models (Advanced Topic)

Real-Time Systems, HT15

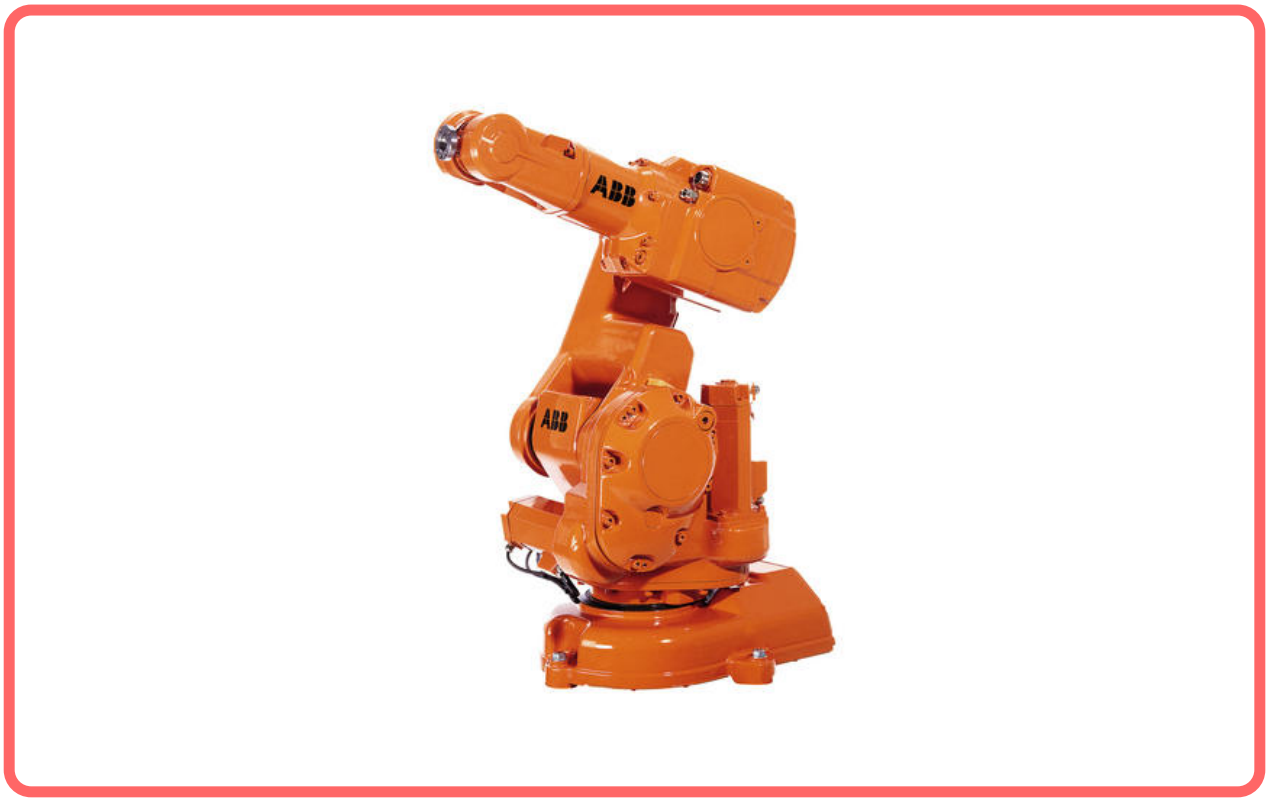
Philipp Rümmer

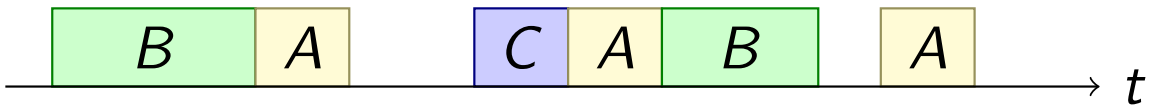
Slides mainly by: Martin Stigge

12. October 2015

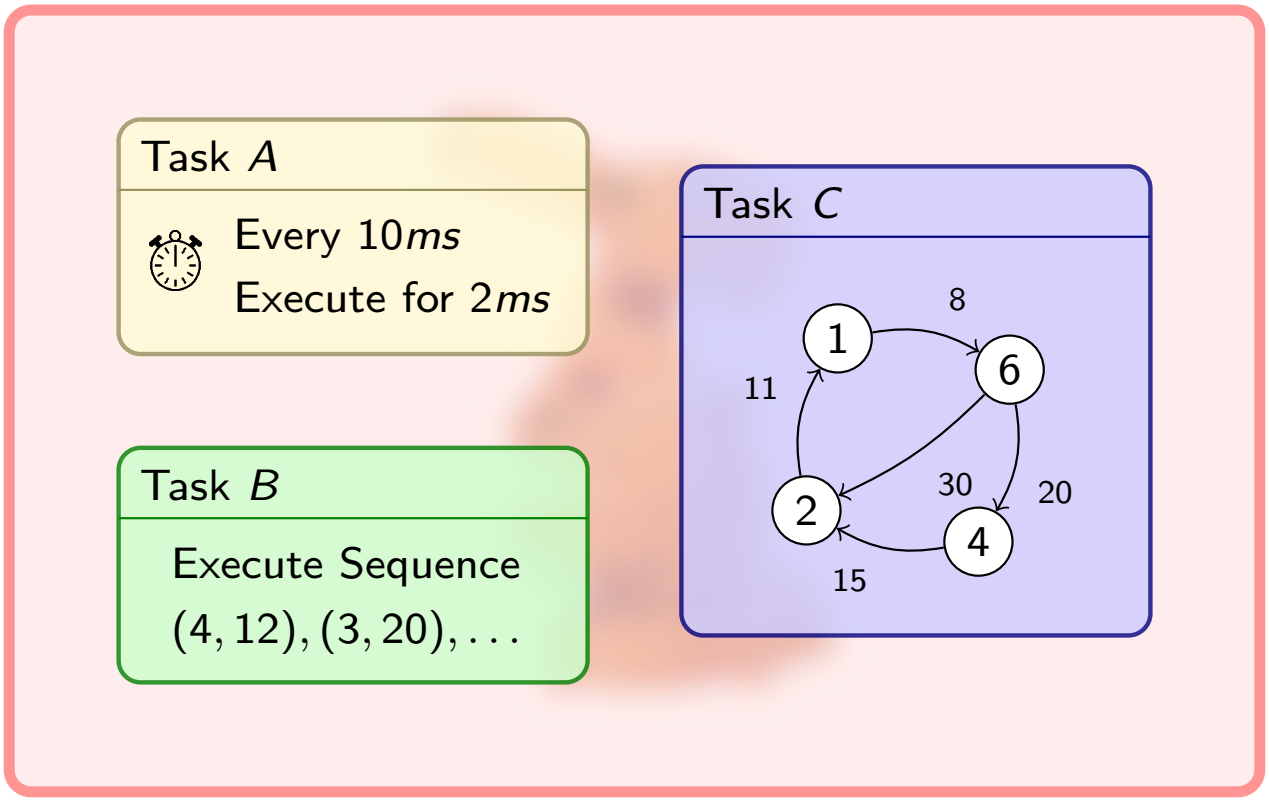
System Abstraction for Analysis

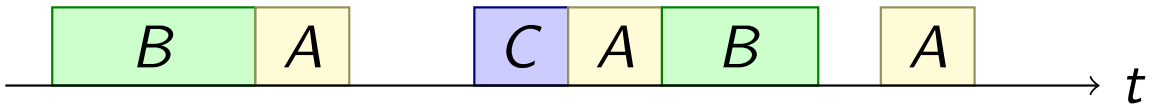
System Abstraction for Analysis





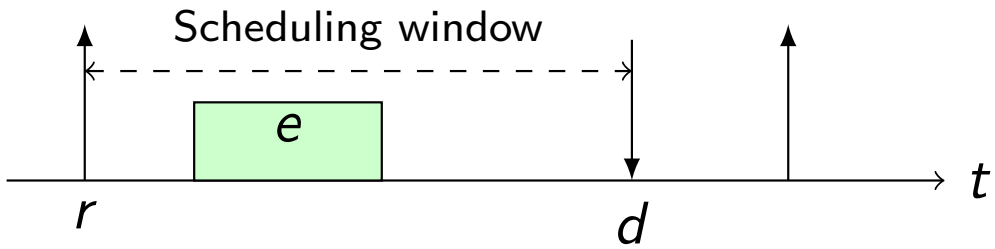
System Abstraction for Analysis





Abstracting Code: Jobs

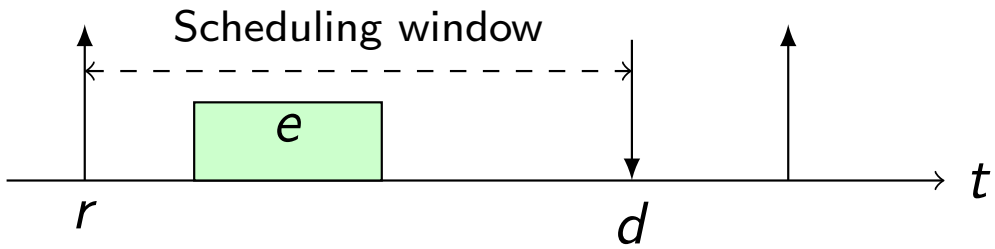
- A *job* J is a piece of code
- Abstracted as three numbers, $J = (r, e, d)$
  - Release time r
  - Worst-case execution time (WCET) e
  - Deadline d



- Real-time scheduling theory:
  1. How are jobs released?
  2. How are they scheduled?
  3. How is schedulability analyzed?

Abstracting Code: Jobs

- A *job* J is a piece of code
- Abstracted as three numbers, $J = (r, e, d)$
  - Release time r
  - Worst-case execution time (WCET) e
  - Deadline d



- Real-time scheduling theory:
  1. How are jobs released?
  2. How are they scheduled?
  3. How is schedulability analyzed?

Abstracting Code: Tasks

- Typical task code structure:

```
...
loop

    // Execute some function for, e.g.,
    //  up to 11ms
    //  (obtained via WCET analysis)

    delay until Previous_Period + 50ms;
end loop;
...
```

- *Periodic Task*
  - Execution time $e = 11ms$ of each *job*
  - Period $p = 50ms$
  - Implicit deadline $d = p$ for each job

Abstracting Code: Tasks

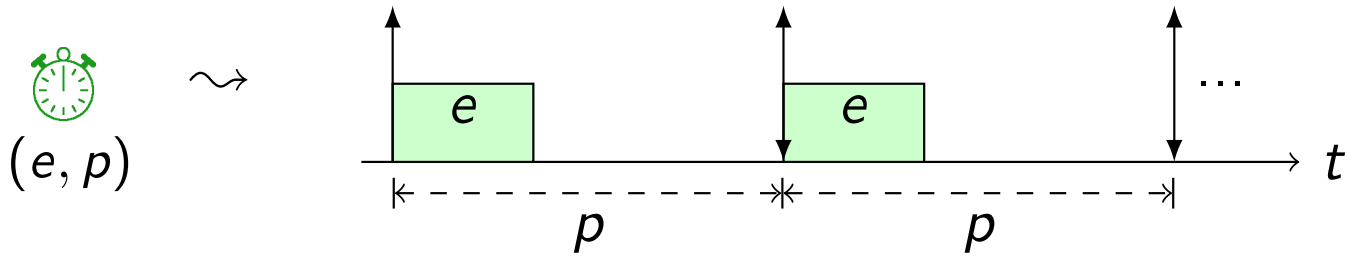
- Typical task code structure:

```
...
loop

    // Execute some function for, e.g.,
    //  up to 11ms
    //  (obtained via WCET analysis)

    delay until Previous_Period + 50ms;
end loop;
...
```

- *Periodic Task*
  - Execution time $e = 11ms$ of each *job*
  - Period $p = 50ms$
  - Implicit deadline $d = p$ for each job

The Liu and Layland Task Model

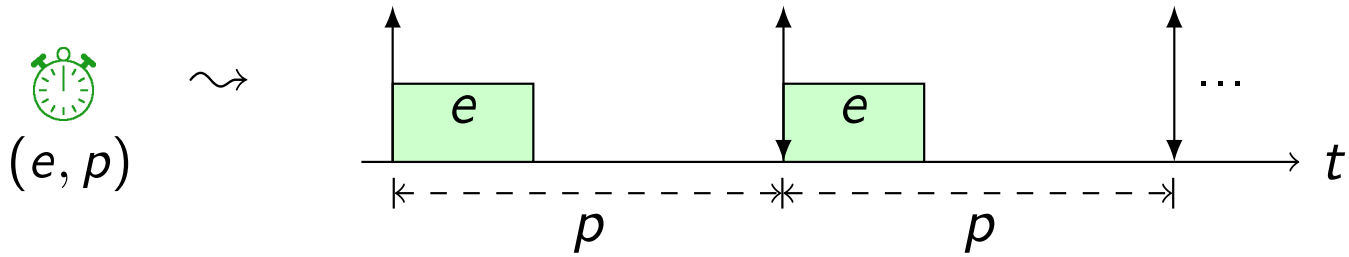
- Each *periodic* task defined via 2 numbers:
  - Job WCET e
  - Minimum inter-release delay p (implicit deadline)



- Task set $\tau = \{\tau_1, \ldots, \tau_n\}$ with $\tau_i = (e_i, p_i)$
- How to *schedule* these tasks?
  - Static/fixed priority scheduling (e.g. RM)
  - Dynamic priority scheduling (e.g. EDF)
- How to *analyze* schedulability?

The Liu and Layland Task Model

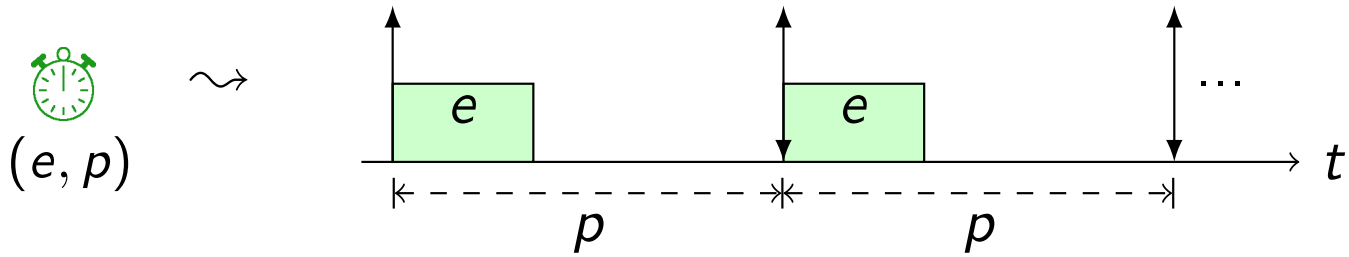
- Each *periodic* task defined via 2 numbers:
  - Job WCET e
  - Minimum inter-release delay p (implicit deadline)



- Task set $\tau = \{\tau_1, \ldots, \tau_n\}$ with $\tau_i = (e_i, p_i)$
- How to *schedule* these tasks?
  - Static/fixed priority scheduling (e.g. RM)
  - Dynamic priority scheduling (e.g. EDF)
- How to *analyze* schedulability?

The Liu and Layland Task Model

- Each *periodic* task defined via 2 numbers:
  - Job WCET e
  - Minimum inter-release delay p (implicit deadline)



- Task set $\tau = \{\tau_1, \ldots, \tau_n\}$ with $\tau_i = (e_i, p_i)$
- How to *schedule* these tasks?
  - Static/fixed priority scheduling (e.g. RM)
  - Dynamic priority scheduling (e.g. EDF)
- How to *analyze* schedulability?

The Liu and Layland Task Model: Schedulability

- *Static priority scheduling*
  - Response time analysis for all tasks

$$R_i = e_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil \cdot e_j$$

    - ★ Compute iteratively; compare with p_i
    - ★ *Precise* test (sufficient and necessary)
  - Utilization test
    - ★ Define $U_i := e_i/p_i$ as *utilization* of task τ_i
    - ★ τ schedulable if $\sum_i U_i \leqslant n(2^{1/n} - 1)$
    - ★ Only sufficient test
- *Dynamic priority scheduling*
  - Just focus on EDF, it's optimal
  - Simple and precise test: $\sum_i U_i \leqslant 1$

The Liu and Layland Task Model: Schedulability

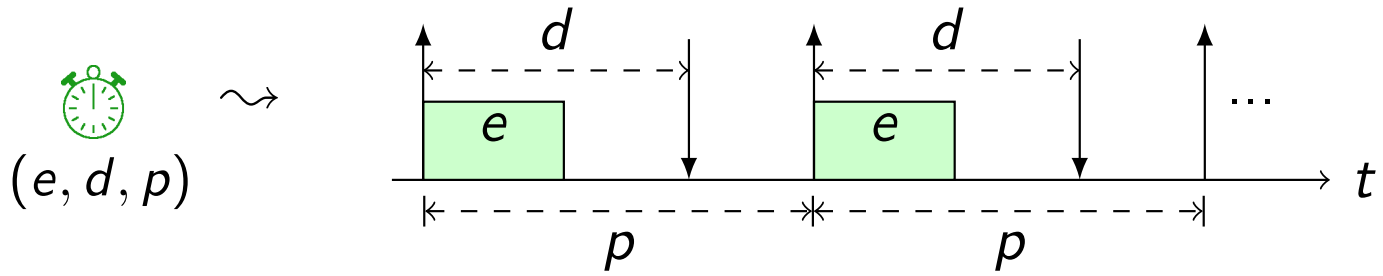
- *Static priority scheduling*
  - Response time analysis for all tasks

$$R_i = e_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil \cdot e_j$$

    - ★ Compute iteratively; compare with p_i
    - ★ *Precise* test (sufficient and necessary)
  - Utilization test
    - ★ Define $U_i := e_i/p_i$ as *utilization* of task τ_i
    - ★ τ schedulable if $\sum_i U_i \leqslant n(2^{1/n} - 1)$
    - ★ Only sufficient test
- *Dynamic priority scheduling*
  - Just focus on EDF, it's optimal
  - Simple and precise test: $\sum_i U_i \leqslant 1$

The Sporadic Task Model

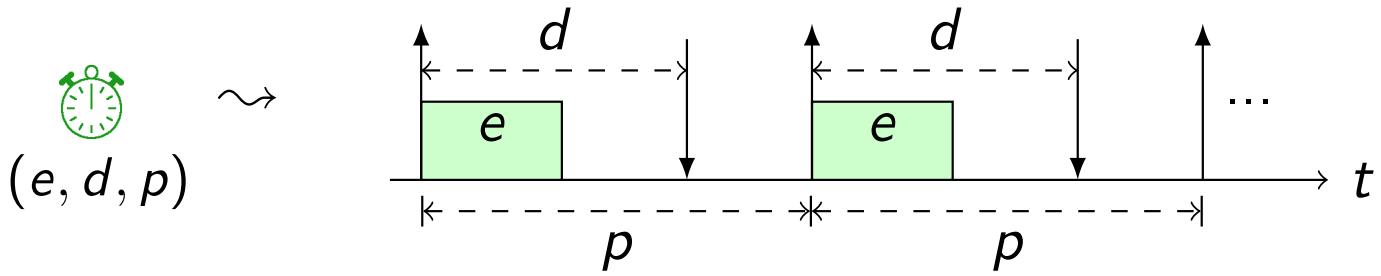
- What if deadlines $\neq$ periods?
- Each *sporadic* task defined via 3 numbers:
  - Job WCET e
  - Relative deadline d
  - Minimum inter-release delay p



- Task set $\tau = \{\tau_1, \ldots, \tau_n\}$ with $\tau_i = (e_i, d_i, p_i)$
- Scheduling?
  - Static/fixed priority scheduling: DM *optimal*
  - Dynamic priority scheduling: EDF *optimal*
  - But how to *analyze* schedulability?

The Sporadic Task Model

- What if deadlines $\neq$ periods?
- Each *sporadic* task defined via 3 numbers:
  - Job WCET e
  - Relative deadline d
  - Minimum inter-release delay p



- Task set $\tau = \{\tau_1, \ldots, \tau_n\}$ with $\tau_i = (e_i, d_i, p_i)$
- Scheduling?
  - Static/fixed priority scheduling: DM *optimal*
  - Dynamic priority scheduling: EDF *optimal*
  - But how to *analyze* schedulability?

The Sporadic Task Model: Schedulability

- *Static priority scheduling*
  - As before, compute response times
  - (They are independent of deadlines!)
- *Dynamic priority scheduling*, i.e., EDF
  - Simple test $\sum_i U_i \leqslant 1$ doesn't work anymore (why?)
  - Introduce *demand bound functions*

The Sporadic Task Model: Schedulability

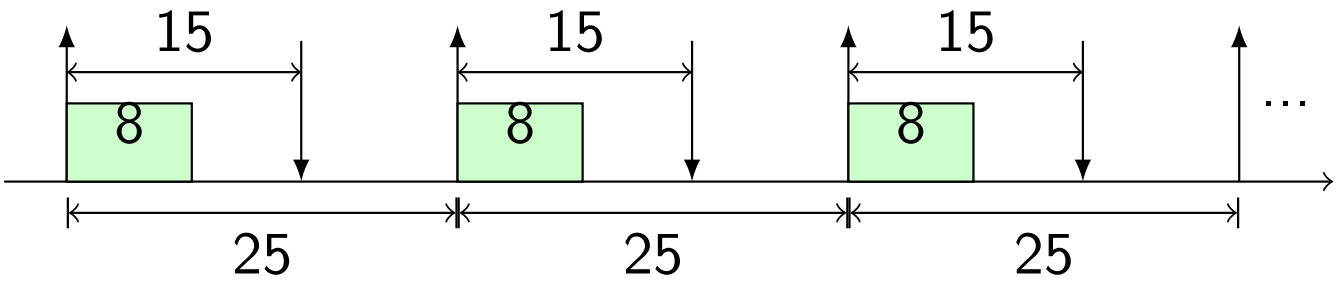
- *Static priority scheduling*
  - As before, compute response times
  - (They are independent of deadlines!)
- *Dynamic priority scheduling*, i.e., EDF
  - Simple test $\sum_i U_i \leqslant 1$ doesn't work anymore (why?)
  - Introduce *demand bound functions*

The Demand Bound Function

- General tool/technique for EDF schedulability analysis: dbf(t)
- Intuition:
  - Given a time interval length t
  - dbf(t) bounds the *demand* for processor time within *any* t interval

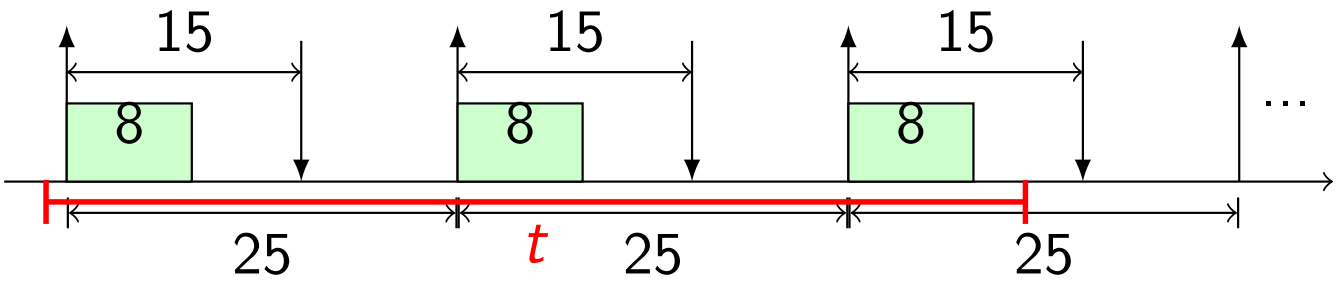
The Demand Bound Function

- General tool/technique for EDF schedulability analysis: dbf(t)
- Intuition:
  - Given a time interval length t
  - dbf(t) bounds the *demand* for processor time within *any* t interval



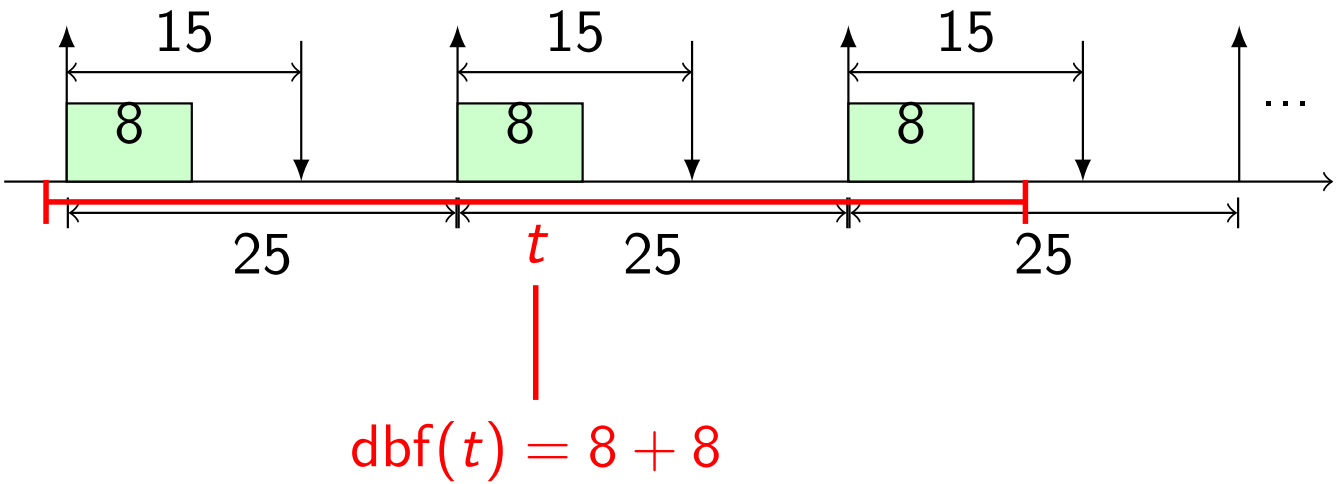
The Demand Bound Function

- General tool/technique for EDF schedulability analysis: dbf(t)
- Intuition:
  - Given a time interval length t
  - dbf(t) bounds the *demand* for processor time within *any* t interval



The Demand Bound Function

- General tool/technique for EDF schedulability analysis: $\mathrm{dbf}(t)$
- Intuition:
  - Given a time interval length t
  - $\mathrm{dbf}(t)$ bounds the *demand* for processor time within *any* t interval



The Demand Bound Function (cont.)

Definition (Demand Bound Function)

For any t, $\mathrm{dbf}(t)$ is the *maximal* WCET requirement of jobs with

- release time
- and deadline

within any interval of length t.

- For a single task τ_i, we write $\mathrm{dbf}_{\tau_i}(t)$. Clearly: $\mathrm{dbf}(t) = \sum_i \mathrm{dbf}_{\tau_i}(t)$
- Typical shape for sporadic tasks:

The Demand Bound Function (cont.)

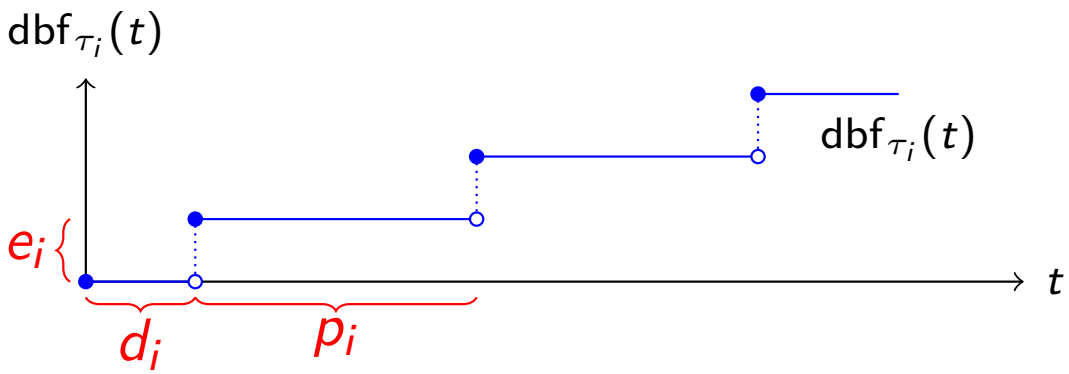
Definition (Demand Bound Function)

For any t, $\mathrm{dbf}(t)$ is the *maximal* WCET requirement of jobs with

- release time
- and deadline

within any interval of length t.

- For a single task τ_i, we write $\mathrm{dbf}_{\tau_i}(t)$. Clearly: $\mathrm{dbf}(t) = \sum_i \mathrm{dbf}_{\tau_i}(t)$
- Typical shape for sporadic tasks:



The Demand Bound Function (cont. 2)

So, ...

1. How to use it for schedulability analysis?
2. How to compute $\mathrm{dbf}(t)$?
3. For which values of t?

Schedulability Analysis with dbf(t)

Theorem

A sporadic task system $\tau = \{\tau_1, \ldots, \tau_n\}$ *is EDF schedulable iff:*

$$\forall t \geqslant 0 : \sum_i \mathrm{dbf}_{\tau_i}(t) \leqslant t$$

- Computing the *step function*:

- *Which values t need to be checked?*

Schedulability Analysis with dbf(t)

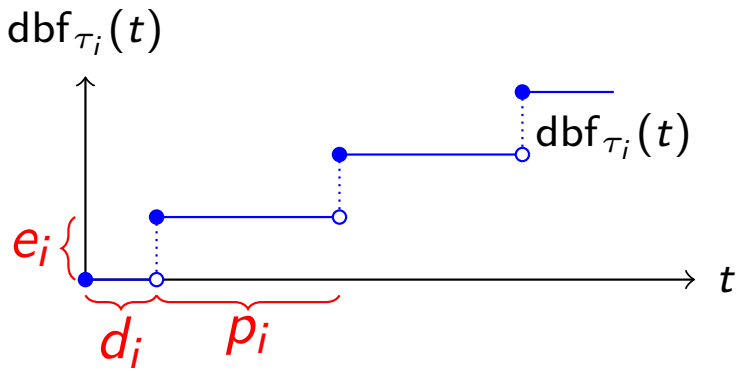
Theorem

A sporadic task system $\tau = \{\tau_1, \ldots, \tau_n\}$ *is EDF schedulable iff:*

$$\forall t \geqslant 0 : \sum_i \mathrm{dbf}_{\tau_i}(t) \leqslant t$$

- Computing the *step function*:

$$\mathrm{dbf}_{\tau_i}(t) = \max\left\{0, \left\lfloor \frac{t - d_i}{p_i} + 1 \right\rfloor \cdot e_i\right\}$$



- *Which values t need to be checked?*

Schedulability Analysis with dbf(t)

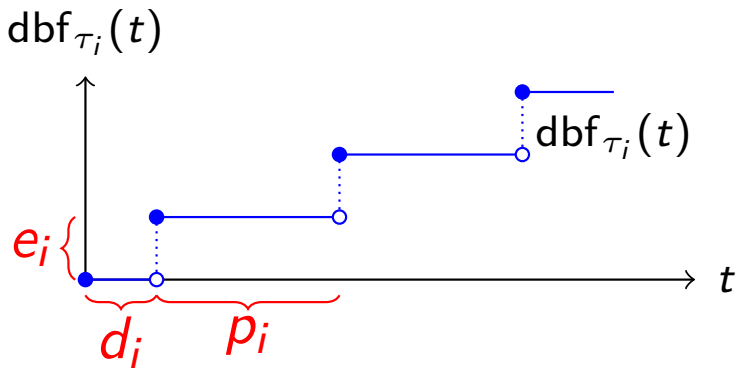
Theorem

A sporadic task system $\tau = \{\tau_1, \ldots, \tau_n\}$ *is EDF schedulable iff:*

$$\forall t \geqslant 0 : \sum_i \mathrm{dbf}_{\tau_i}(t) \leqslant t$$

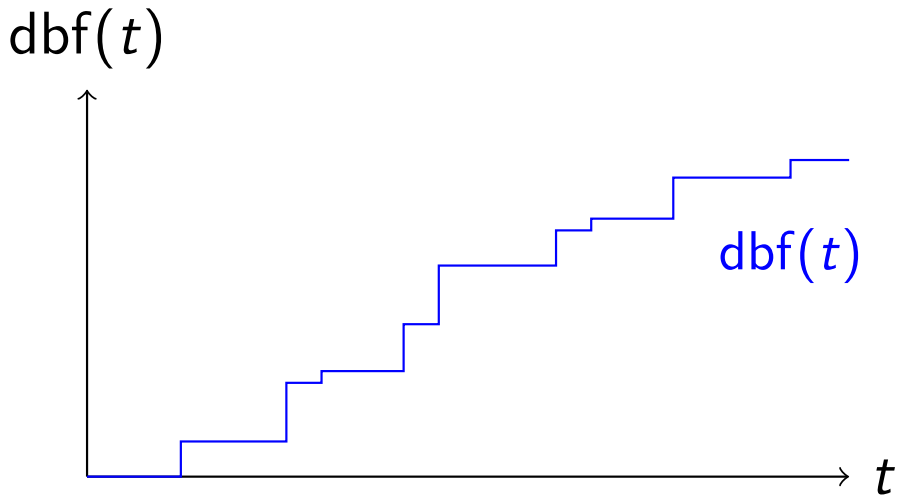
- Computing the *step function*:

$$\mathrm{dbf}_{\tau_i}(t) = \max\left\{0, \left\lfloor \frac{t - d_i}{p_i} + 1 \right\rfloor \cdot e_i\right\}$$



- *Which values t need to be checked?*

Which t to Check?



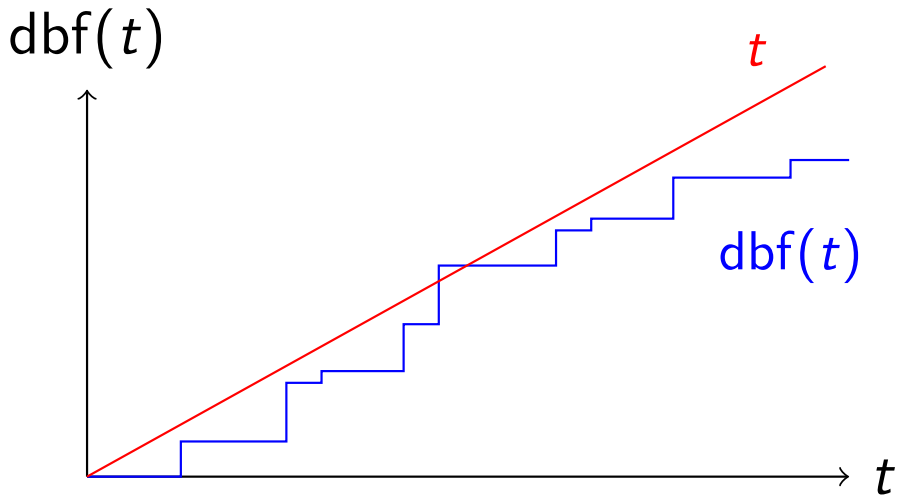
$$\forall t : \mathrm{dbf}(t) \leqslant t$$

- Test for intersection with diagonal t
- Trick: Derive linear bound for $\mathrm{dbf}(t)$
  - Intersection with t gives bound D
  - Check only up to D, no intersection beyond that
  - Can be derived using: $\mathrm{dbf}(t) \leqslant \sum_i e_i + t \cdot \sum_i U_i$
  - Thus: Check $\mathrm{dbf}(t)$ only for

$$t < \frac{\sum_i e_i}{1 - \sum_i U_i}.$$

- Note: $\sum_i U_i \leqslant 1$, otherwise system is overloaded

Which t to Check?



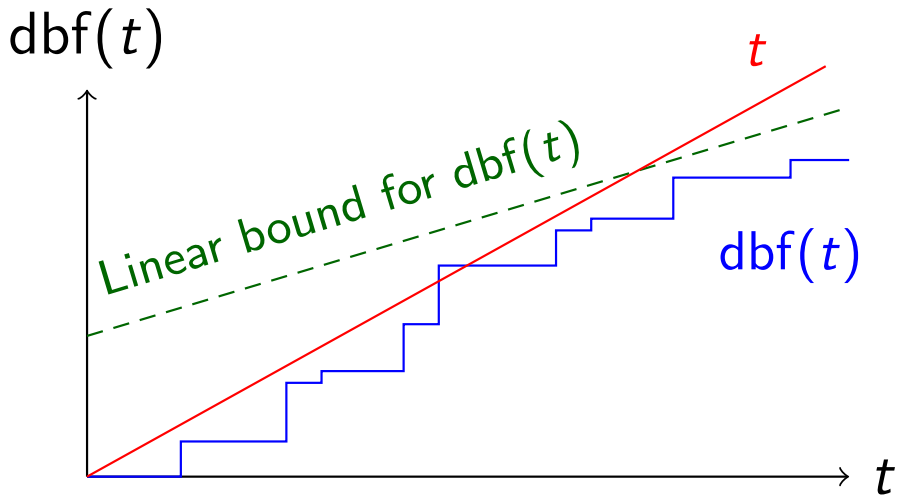
$$\forall t : \mathrm{dbf}(t) \leqslant t$$

- Test for intersection with diagonal t
- Trick: Derive linear bound for $\mathrm{dbf}(t)$
  - Intersection with t gives bound D
  - Check only up to D, no intersection beyond that
  - Can be derived using: $\mathrm{dbf}(t) \leqslant \sum_i e_i + t \cdot \sum_i U_i$
  - Thus: Check $\mathrm{dbf}(t)$ only for

$$t < \frac{\sum_i e_i}{1 - \sum_i U_i}.$$

- Note: $\sum_i U_i \leqslant 1$, otherwise system is overloaded

Which t to Check?



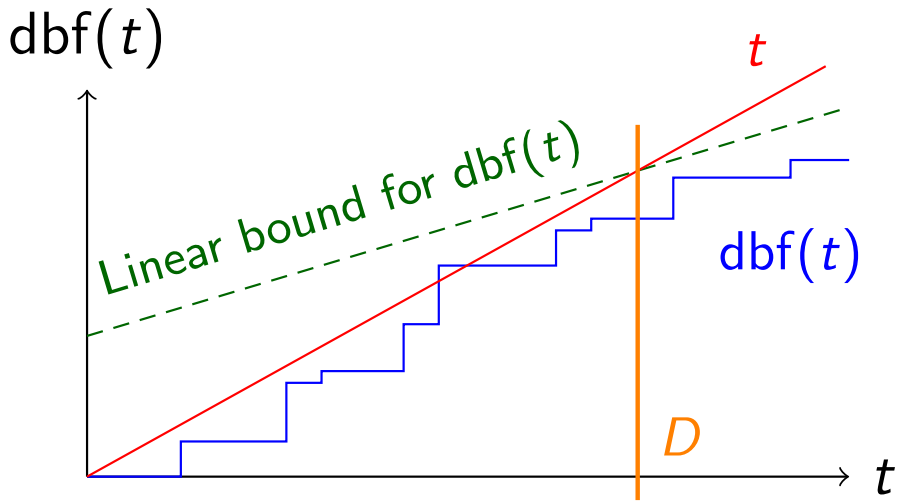
$$\forall t : \mathrm{dbf}(t) \leqslant t$$

- Test for intersection with diagonal t
- Trick: Derive linear bound for $\mathrm{dbf}(t)$
  - Intersection with t gives bound D
  - Check only up to D, no intersection beyond that
  - Can be derived using: $\mathrm{dbf}(t) \leqslant \sum_i e_i + t \cdot \sum_i U_i$
  - Thus: Check $\mathrm{dbf}(t)$ only for

$$t < \frac{\sum_i e_i}{1 - \sum_i U_i}.$$

- Note: $\sum_i U_i \leqslant 1$, otherwise system is overloaded

Which t to Check?



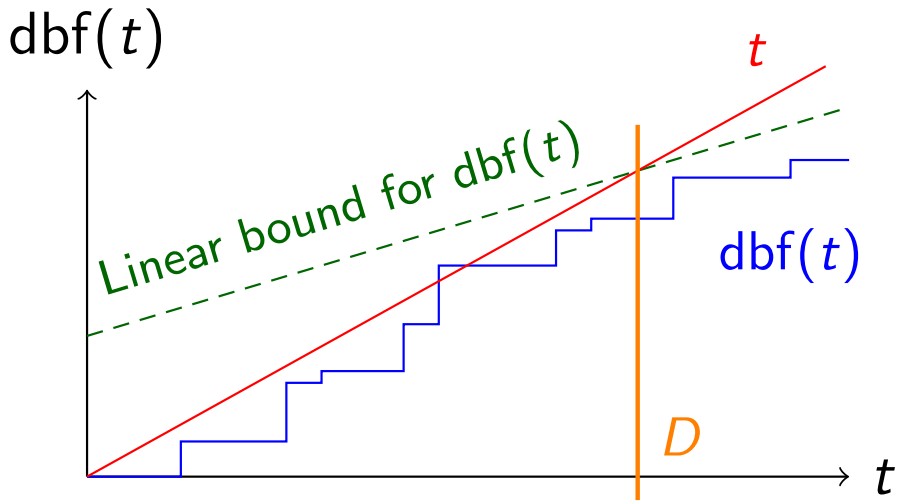
$$\forall t : \mathrm{dbf}(t) \leqslant t$$

- Test for intersection with diagonal t
- Trick: Derive linear bound for $\mathrm{dbf}(t)$
  - Intersection with t gives bound D
  - Check only up to D, no intersection beyond that
  - Can be derived using: $\mathrm{dbf}(t) \leqslant \sum_i e_i + t \cdot \sum_i U_i$
  - Thus: Check $\mathrm{dbf}(t)$ only for

$$t < \frac{\sum_i e_i}{1 - \sum_i U_i}.$$

- Note: $\sum_i U_i \leqslant 1$, otherwise system is overloaded

Which t to Check?



$$\forall t : \mathrm{dbf}(t) \leqslant t$$

- Test for intersection with diagonal t
- Trick: Derive linear bound for $\mathrm{dbf}(t)$
  - Intersection with t gives bound D
  - Check only up to D, no intersection beyond that
  - Can be derived using: $\mathrm{dbf}(t) \leqslant \sum_i e_i + t \cdot \sum_i U_i$
  - Thus: Check $\mathrm{dbf}(t)$ only for

$$t < \frac{\sum_i e_i}{1 - \sum_i U_i}.$$

- Note: $\sum_i U_i \leqslant 1$, otherwise system is overloaded

Schedulability for Sporadic Tasks: Summary

- Given a sporadic task set $\tau = \{\tau_1, \ldots, \tau_n\}$
- *Compute* $\mathrm{dbf}(t)$ for increasing $t = 1, 2, \ldots$ using

$$\mathrm{dbf}(t) = \sum_i \mathrm{dbf}_{\tau_i}(t) = \sum_i \max\left\{0, \left\lfloor \frac{t - d_i}{p_i} + 1 \right\rfloor \cdot e_i\right\}$$

- For each t, *check* whether $\mathrm{dbf}(t) \leqslant t$
  - If violated: *unschedulable*!
  - (Optimizations: Only compute at "steps")
- *Terminate* procedure as soon as

$$t = \frac{\sum_i e_i}{1 - \sum_i U_i}$$

  - If no violation found, τ is *schedulable*!
  - Optimization: Terminate already at lcm. (Exercise: Why?)

The General Multiframe (GMF) Task Model

- Code is not always periodic:

```
loop

    // Execute some function
    delay until Period_Start + 50ms;
    // Execute another function
    delay until Prev_Function + 30ms;
    // Execute yet another function
    delay until Prev_Function + 70ms;

end loop;
```

- Task is split in *frames*, each with own
  - Execution time $e^{(j)}$
  - Inter-release separation $p^{(j)}$
  - Deadline $d^{(j)}$ for the job

The General Multiframe (GMF) Task Model

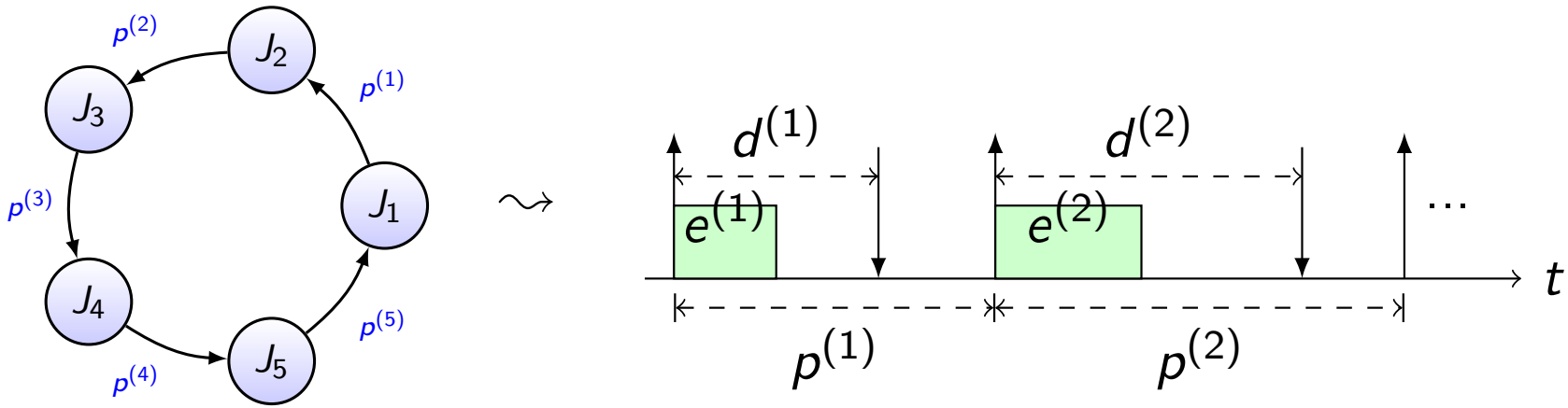
- Code is not always periodic:

```
loop

    // Execute some function
    delay until Period_Start + 50ms;
    // Execute another function
    delay until Prev_Function + 30ms;
    // Execute yet another function
    delay until Prev_Function + 70ms;

end loop;
```

- Task is split in *frames*, each with own
  - Execution time $e^{(j)}$
  - Inter-release separation $p^{(j)}$
  - Deadline $d^{(j)}$ for the job

The General Multiframe (GMF) Task Model (cont.)

- Tasks *cycle* through job types
  - Vector for WCET $(e^{(1)}, \ldots, e^{(n)})$
  - Vector for deadlines $(d^{(1)}, \ldots, e^{(n)})$
  - Vector for minimum inter-release delays $(p^{(1)}, \ldots, p^{(n)})$



- Schedulability with EDF?
  - Also use dbf like for sporadic tasks
  - How to calculate dbf? What about the bound?
  - Exercise for the interested!

The Recurring Branching (RB) Task Model

- What about *branches*?

```
loop
    // Execute some function
    delay until Period_Start + 50ms;
    if (condition) then {
        // Execute another function
        delay until Prev_Function + 30ms;
    } else {
        // Execute yet another function
        delay until Prev_Function + 70ms;
    }
end loop;
```

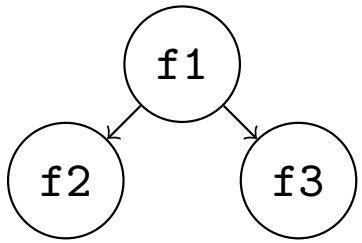
- Task is a tree
  - Vertices represent jobs
  - Edges represent control flow and delays
  - Restarted once a leaf is reached

The Recurring Branching (RB) Task Model

- What about *branches*?

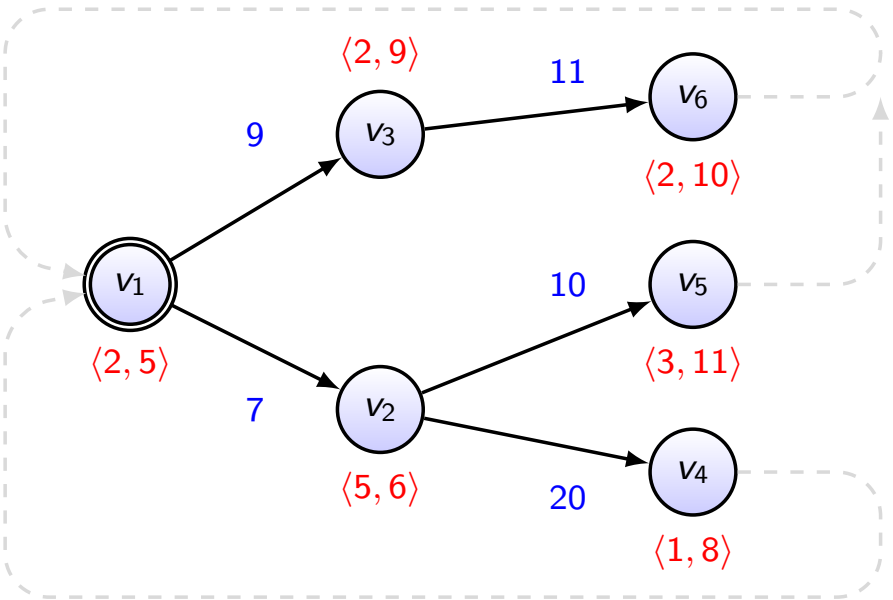
```
loop
    // Execute some function
    delay until Period_Start + 50ms;
    if (condition) then {
        // Execute another function
        delay until Prev_Function + 30ms;
    } else {
        // Execute yet another function
        delay until Prev_Function + 70ms;
    }
end loop;
```

- Task is a tree
  - Vertices represent jobs
  - Edges represent control flow and delays
  - Restarted once a leaf is reached



The Recurring Branching (RB) Task Model (cont.)

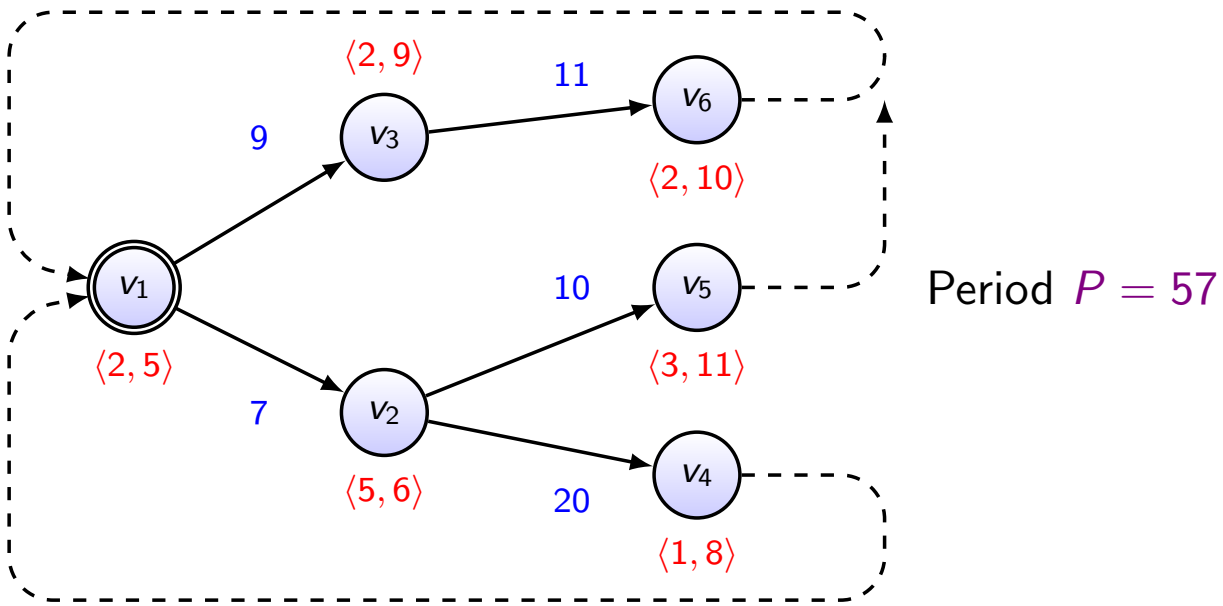
- Introduces *branching* structures
- A *tree* for each task
  - Vertices v: job types with WCET and deadline $\langle e(v), d(v) \rangle$
  - Edges (u, v): minimum inter-release delays $p(u, v)$
  - General period parameter P



- Schedulability analysis dbf-based, rather involved

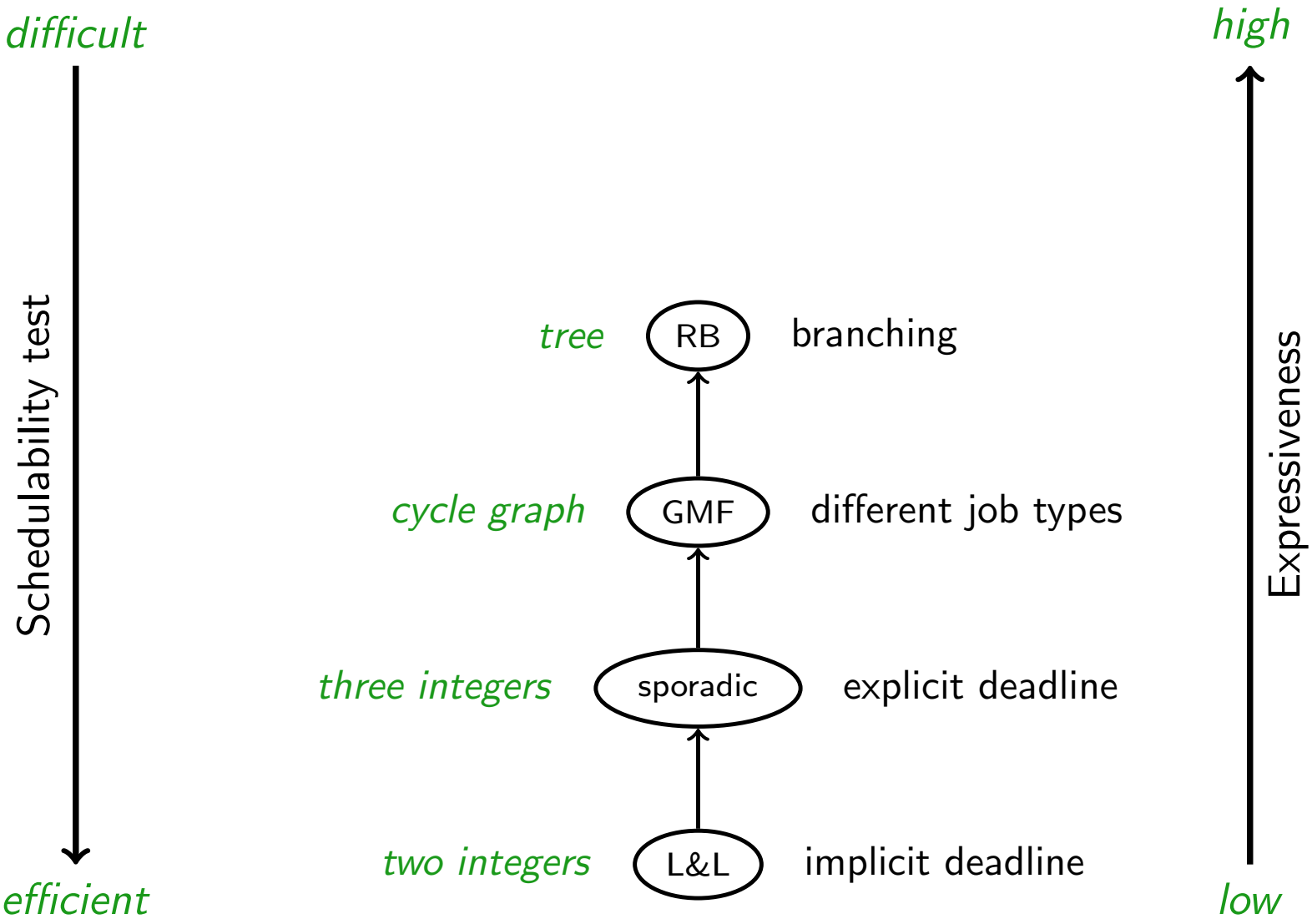
The Recurring Branching (RB) Task Model (cont.)

- Introduces *branching* structures
- A *tree* for each task
  - Vertices v: job types with WCET and deadline $\langle e(v), d(v) \rangle$
  - Edges (u, v): minimum inter-release delays $p(u, v)$
  - General period parameter P



- Schedulability analysis dbf-based, rather involved

Hierarchy of Models



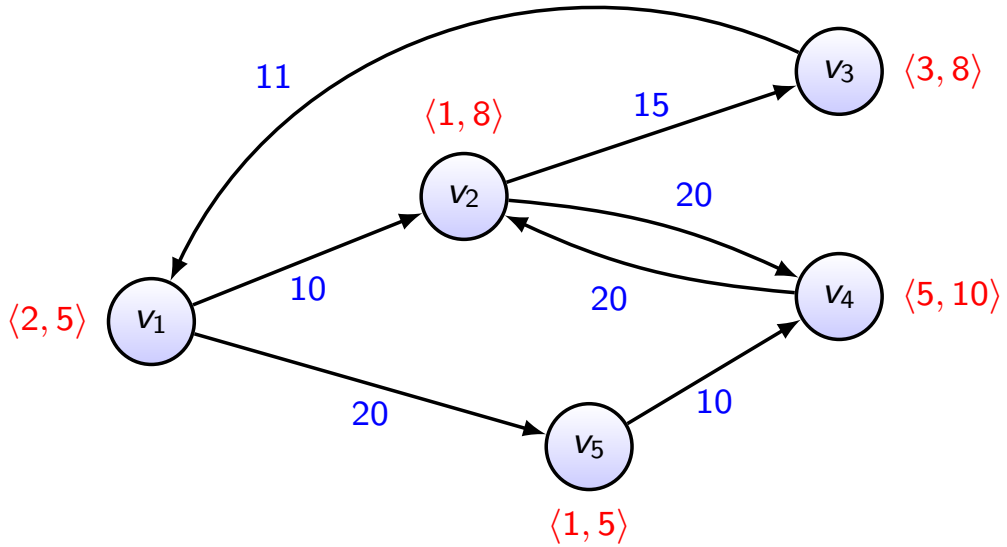
Generalize Further?

- What we can model now:
  - Periodic behavior
  - Arbitrary deadlines
  - Multiple frames
  - Branching behavior
- Still not possible:
  - Local loops
  - Local modes
  - ...

```
loop
    f1();
    delay ...
    if (cond) then {
        while (cond2) {
            f2();
            delay ...
        }
    } else {
        f3();
        delay ...
    }
end loop;
```

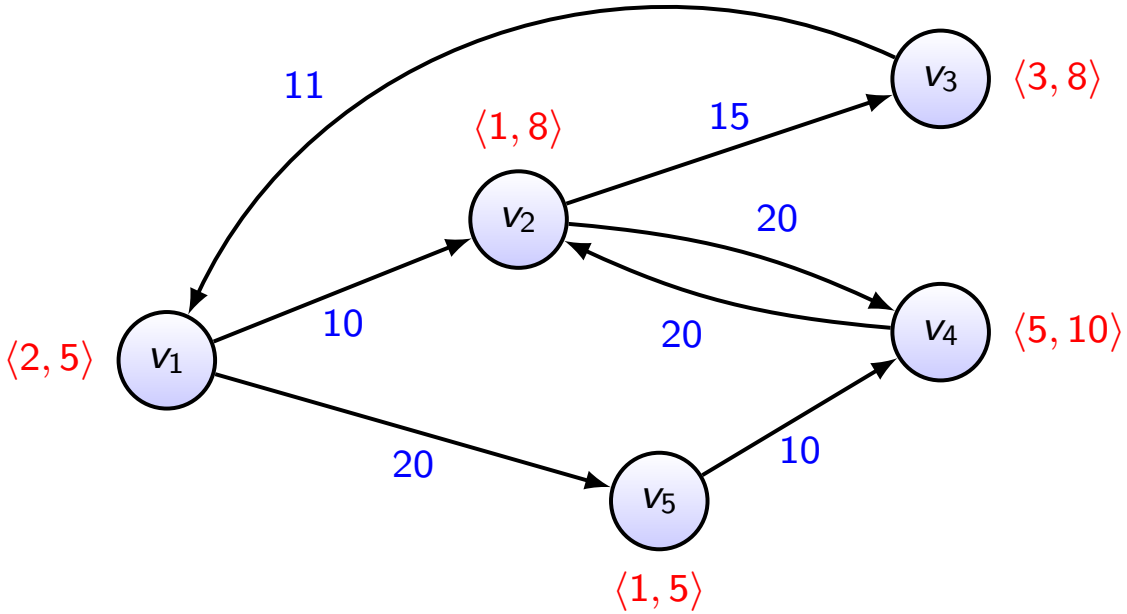
The Digraph Real-Time (DRT) Task Model

- Branching, cycles (loops), ...
- *Directed graph* for each task
  - Vertices v: job types with WCET and deadline $\langle e(v), d(v) \rangle$
  - Edges (u, v): minimum inter-release delays $p(u, v)$

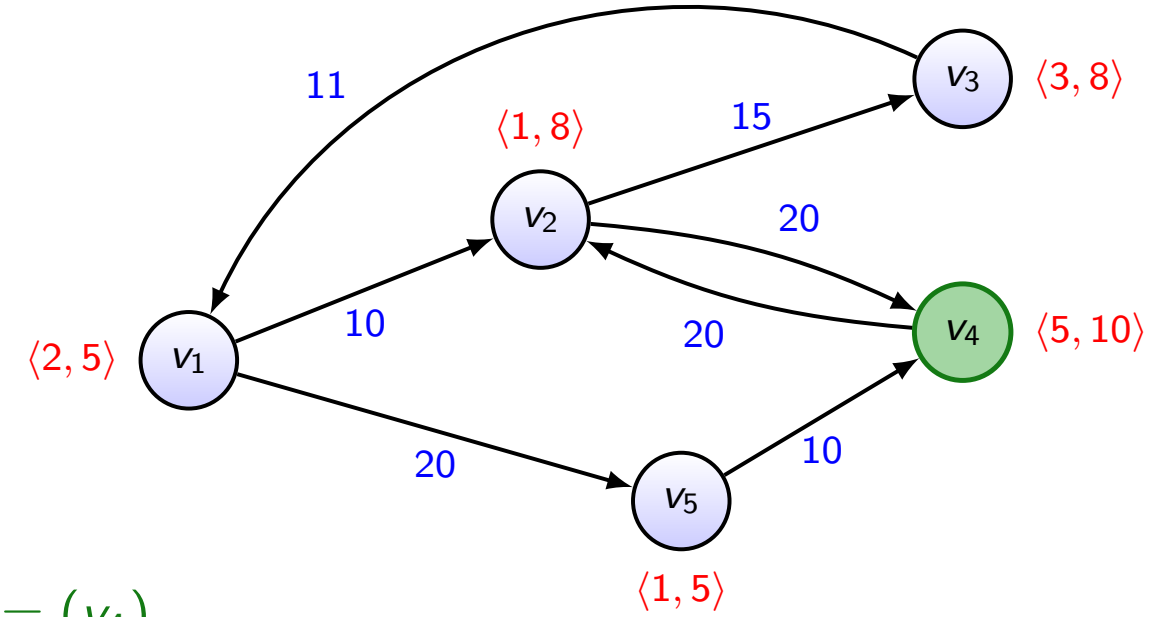


- Rest of lecture: Analyze schedulability?

DRT: Semantics

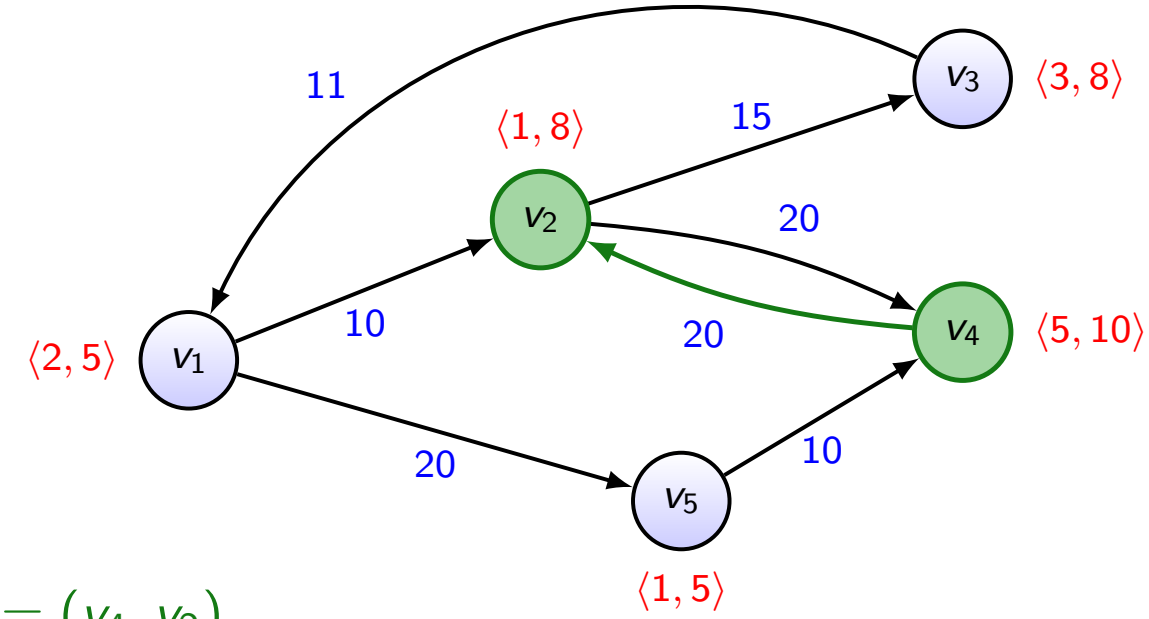


DRT: Semantics

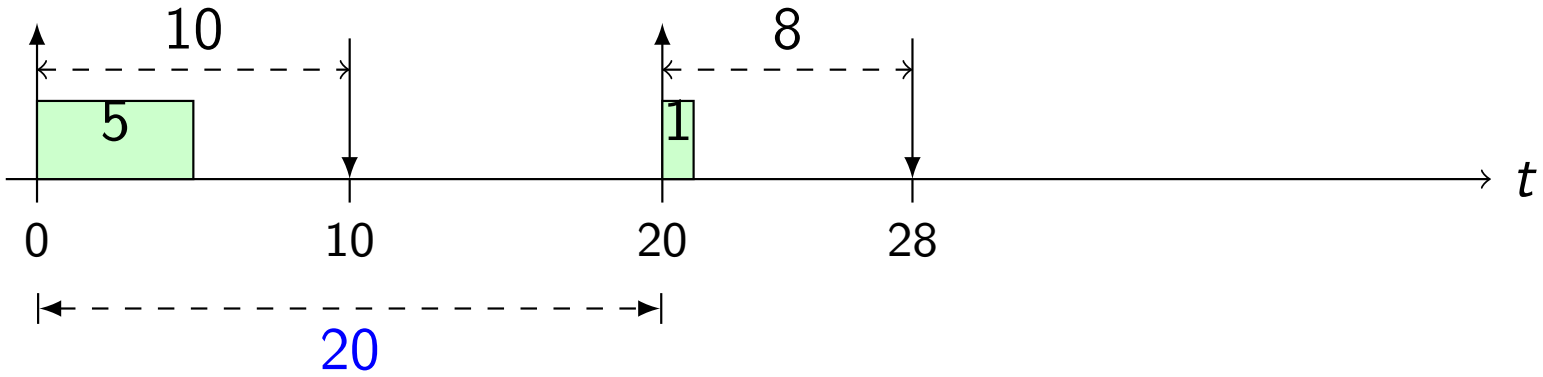


Path $\pi = (v_4)$

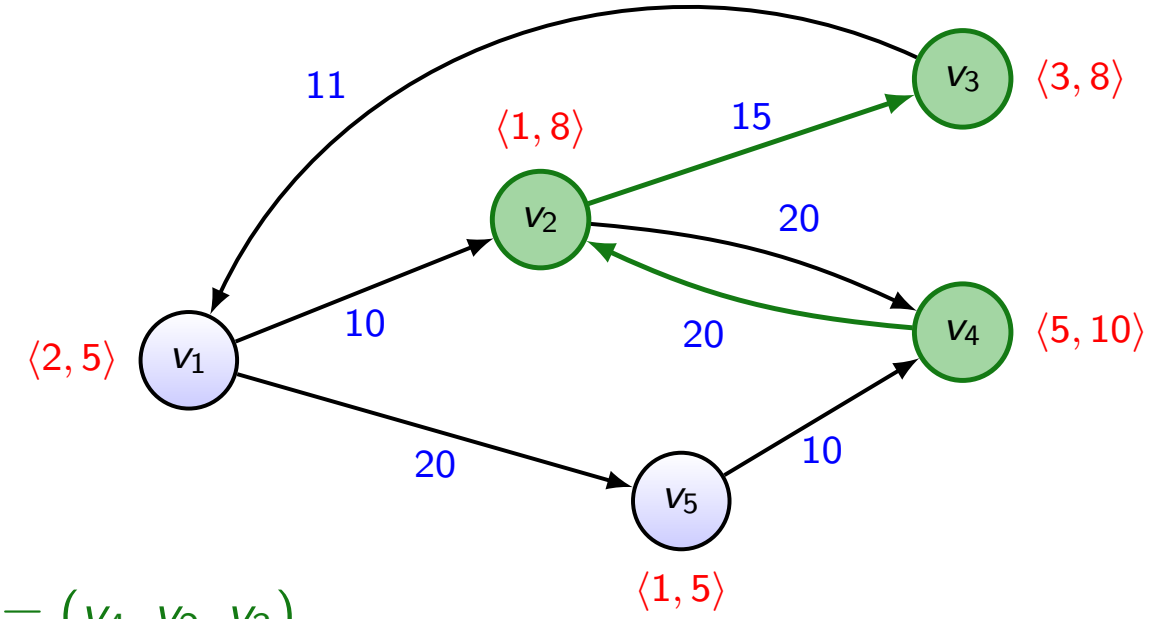
DRT: Semantics



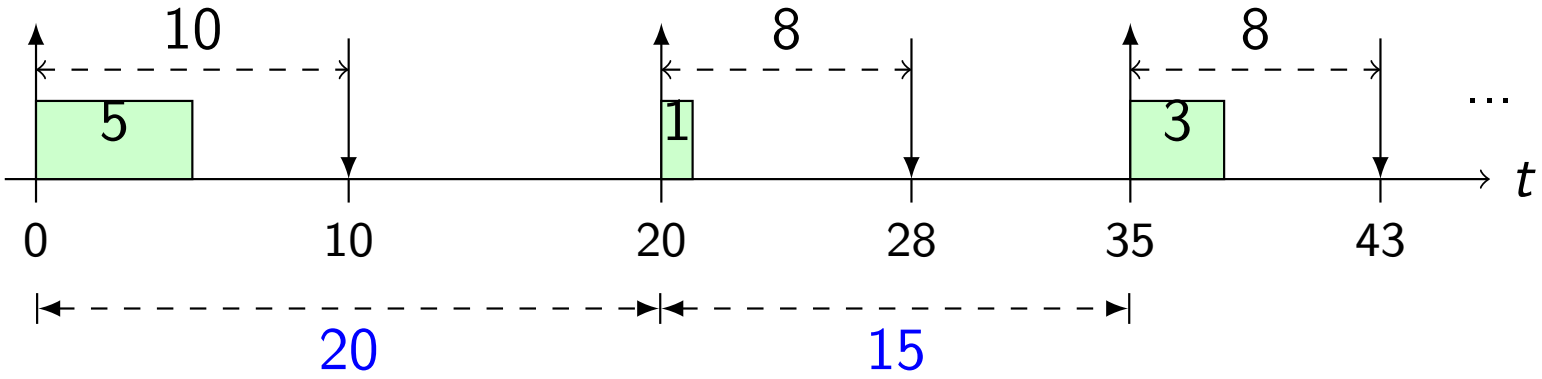
Path $\pi = (v_4, v_2)$



DRT: Semantics



Path $\pi = (v_4, v_2, v_3)$



DRT: Schedulability Analysis

- Theorem still holds:

Theorem

A DRT task system $\tau = \{\tau_1, \ldots, \tau_n\}$ *is EDF schedulable iff:*

$$\forall t \geqslant 0 : \sum_i \mathrm{dbf}_{\tau_i}(t) \leqslant t$$

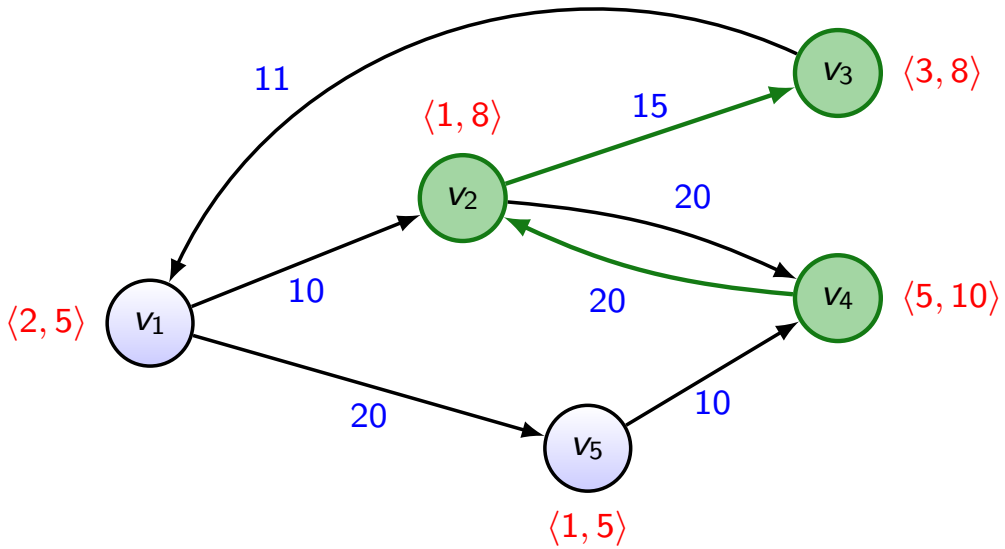
- Thus, do as before:
  1. Compute demand bound function $\mathrm{dbf}_{\tau_i}(t)$
     - ⋆ How to do that for given t?
  2. Test $\mathrm{dbf}(t)$ for all $t \leqslant D$ for some bound D
     - ⋆ How to derive the bound?

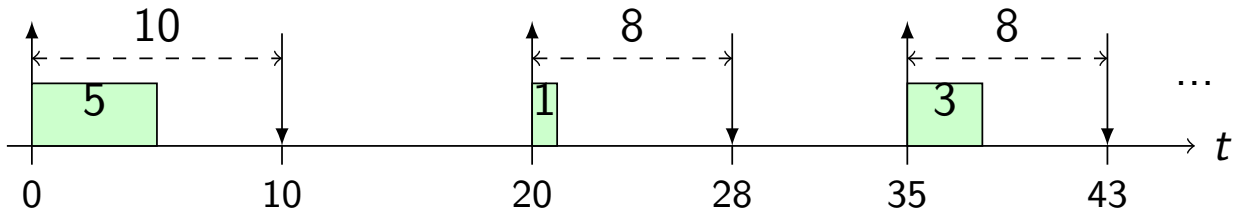
Demand Pairs

- Recall demand bound function:
  - Interval length t, sum all demand ...
  - ... of jobs *released* and with *deadline* inside
- Consider a path in a task's graph:

Demand Pairs

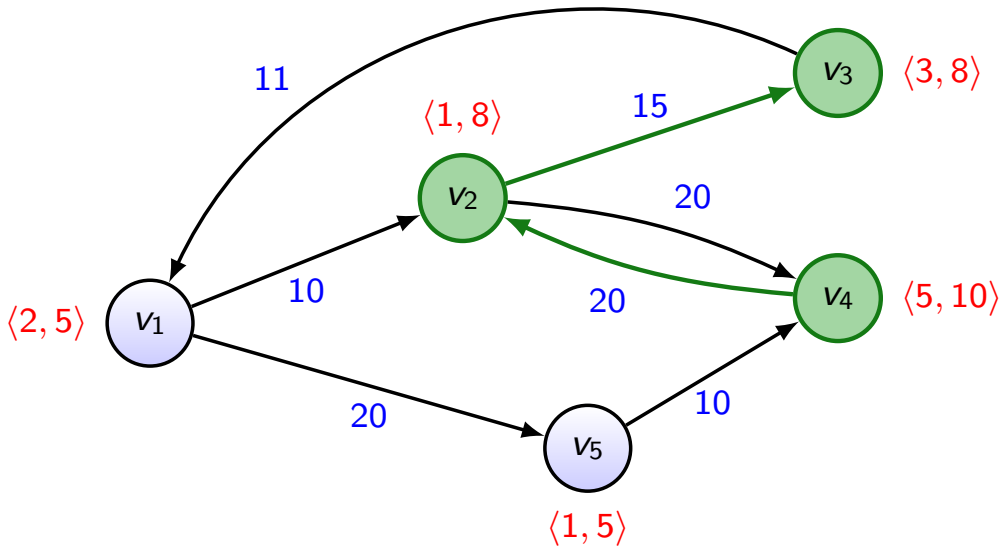
- Recall demand bound function:
  - Interval length t, sum all demand ...
  - ... of jobs *released* and with *deadline* inside
- Consider a path in a task's graph:





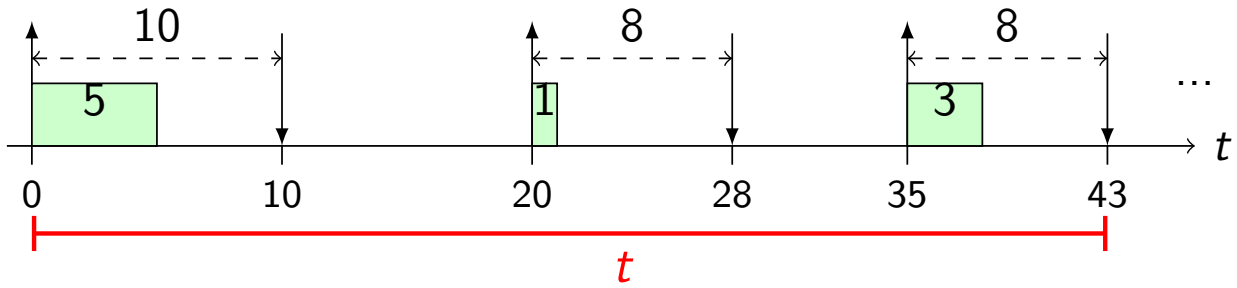
Demand Pairs

- Recall demand bound function:
  - Interval length t, sum all demand ...
  - ... of jobs *released* and with *deadline* inside
- Consider a path in a task's graph:



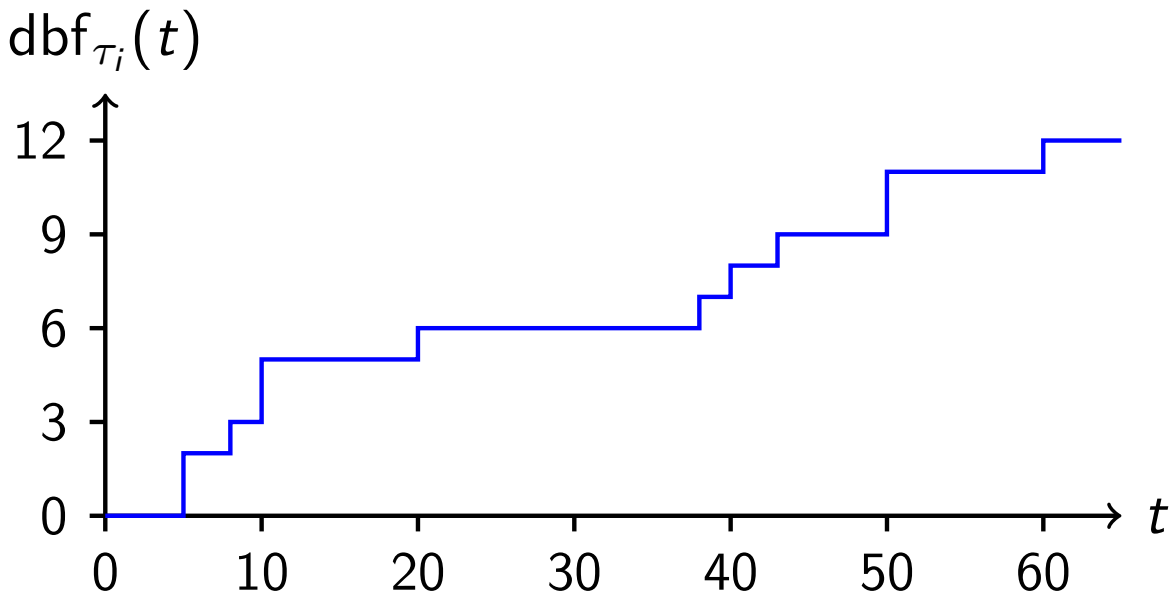
Execution demand:
$5 + 1 + 3 = 9$
Interval size:
$20 + 15 + 8 = 43$

Demand pair $\langle 9, 43 \rangle$



Demand Pairs (cont.)

- From demand pair $\langle e, d \rangle$ we learn:
  - Task can create e units of exec. demand ...
  - ... during interval of size d
- Useful for demand bound function!

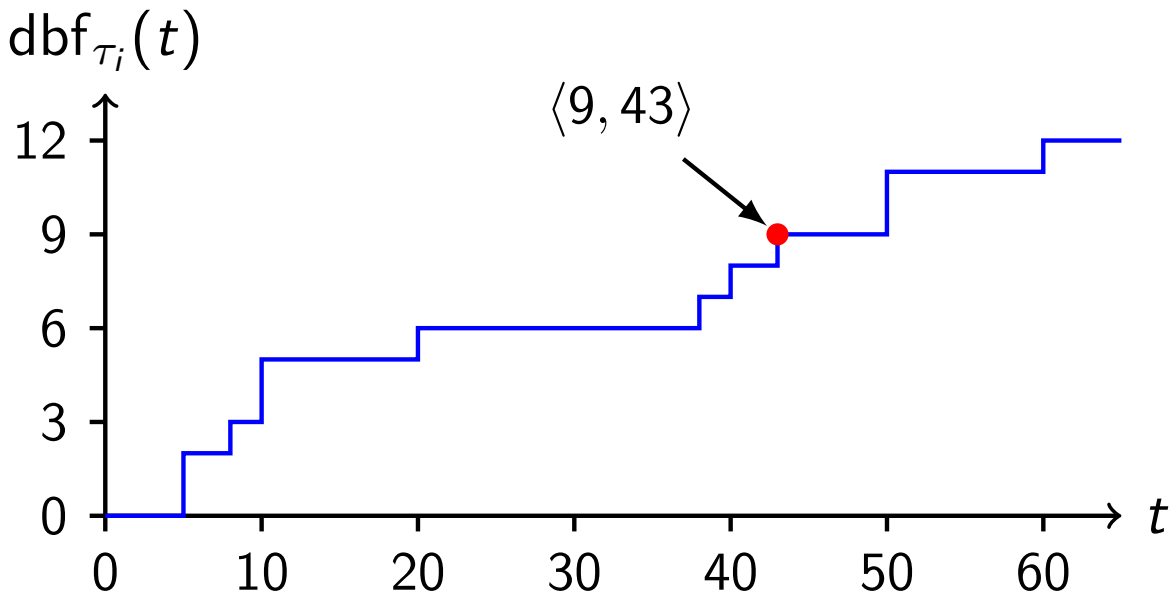


- Thus: Compute *all demand pairs*; take "maximum"

$$\text{dbf}_{\tau_i}(t) = \max\{e \mid \langle e, d \rangle \text{ demand pair with } d \leqslant t\}$$

Demand Pairs (cont.)

- From demand pair $\langle e, d \rangle$ we learn:
  - Task can create e units of exec. demand ...
  - ... during interval of size d
- Useful for demand bound function!

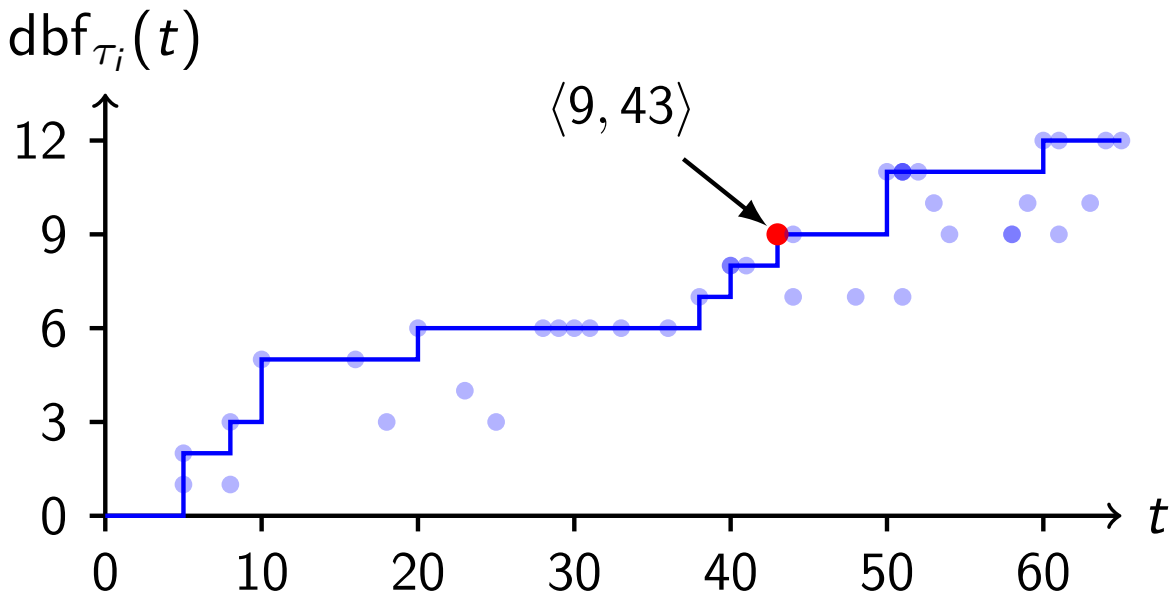


- Thus: Compute *all demand pairs*; take "maximum"

$$\text{dbf}_{\tau_i}(t) = \max\{e \mid \langle e, d \rangle \text{ demand pair with } d \leqslant t\}$$

Demand Pairs (cont.)

- From demand pair $\langle e, d \rangle$ we learn:
  - Task can create e units of exec. demand ...
  - ... during interval of size d
- Useful for demand bound function!

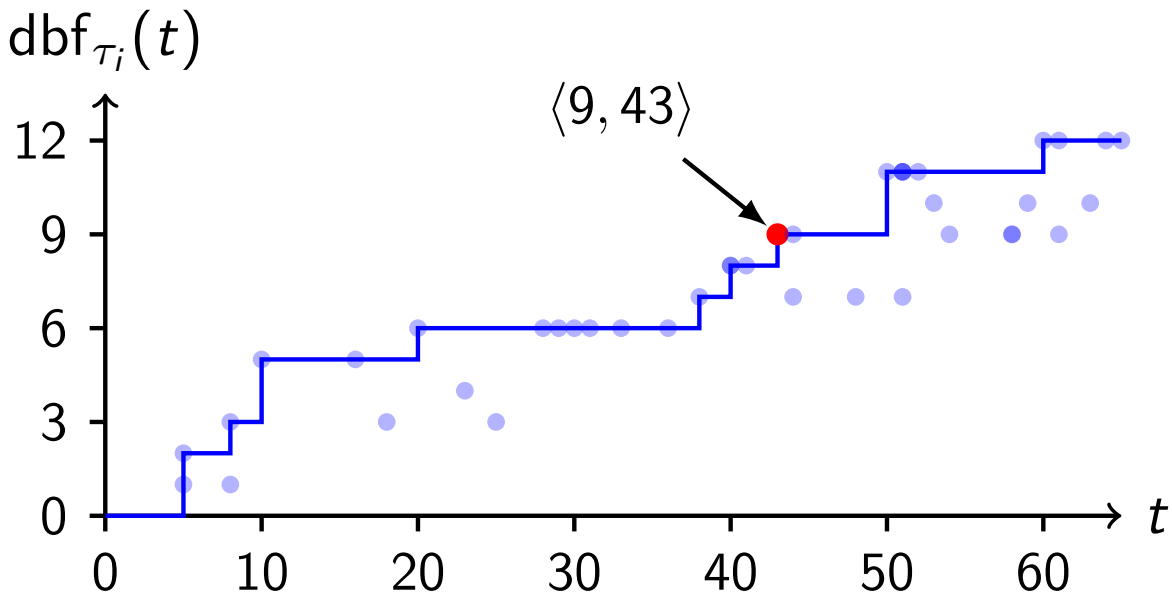


- Thus: Compute *all demand pairs*; take "maximum"

$$\text{dbf}_{\tau_i}(t) = \max\{e \mid \langle e, d \rangle \text{ demand pair with } d \leqslant t\}$$

Demand Pairs (cont.)

- From demand pair $\langle e, d \rangle$ we learn:
  - Task can create e units of exec. demand ...
  - ... during interval of size d
- Useful for demand bound function!



- Thus: Compute *all demand pairs*; take "maximum"

$$\text{dbf}_{\tau_i}(t) = \max\{e \mid \langle e, d \rangle \text{ demand pair with } d \leqslant t\}$$

Demand Pairs (cont. 2)

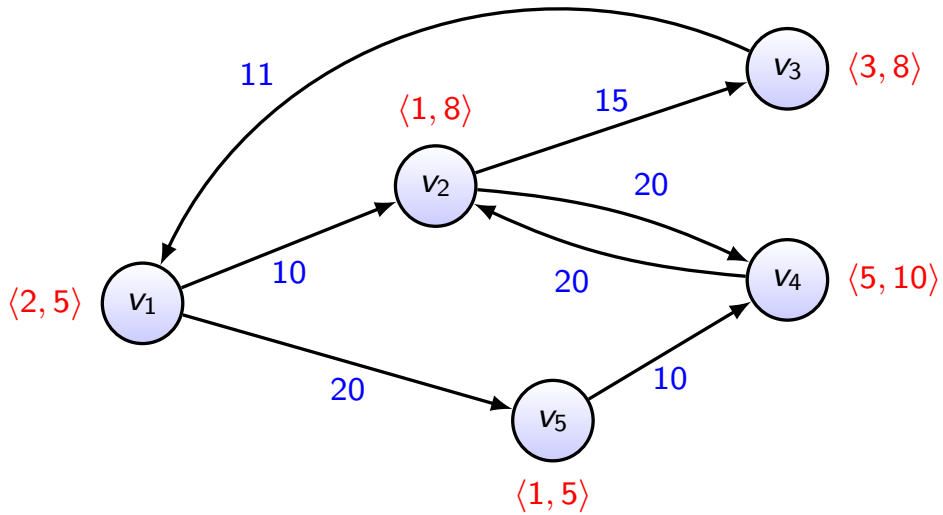
- Formally:
  - Given path $\pi = (\pi_0, \ldots, \pi_l)$
  - *Execution demand:* $e(\pi) := \sum_{i=0}^{l} e(\pi_i)$
  - *Deadline:* $d(\pi) := \sum_{i=0}^{l-1} p(\pi_i, \pi_{i+1}) + d(\pi_l)$
  - $\langle e(\pi), d(\pi) \rangle$ is a *demand pair* for π
- How to compute all demand pairs?
  - Enumerate them: Too expensive! (Exponential..)
  - Better: Iteration using *abstraction*
  - (Remark: Demand pairs are abstractions of paths)

Demand Pairs (cont. 2)

- Formally:
  - Given path $\pi = (\pi_0, \ldots, \pi_l)$
  - *Execution demand:* $e(\pi) := \sum_{i=0}^{l} e(\pi_i)$
  - *Deadline:* $d(\pi) := \sum_{i=0}^{l-1} p(\pi_i, \pi_{i+1}) + d(\pi_l)$
  - $\langle e(\pi), d(\pi) \rangle$ is a *demand pair* for π
- How to compute all demand pairs?
  - Enumerate them: Too expensive! (Exponential..)
  - Better: Iteration using *abstraction*
  - (Remark: Demand pairs are abstractions of paths)

Demand Triples

- Idea: Start with 0-paths (one vertex), extend stepwise
- We need: Abstraction which
  1. allows to *extend* paths,
  2. contains demand pair information,
  3. without visiting/storing all paths
- Idea: *Demand triples*
  - Execution demand $e(\pi)$
  - Deadline $d(\pi)$
  - Last vertex π_l
- Demand triple $\langle e(\pi), d(\pi), \pi_l \rangle$ is another abstraction!



Path (v_4)
$\rightsquigarrow \langle 5, 10, v_4 \rangle$

Path (v_4, v_2)
$\rightsquigarrow \langle 6, 28, v_2 \rangle$

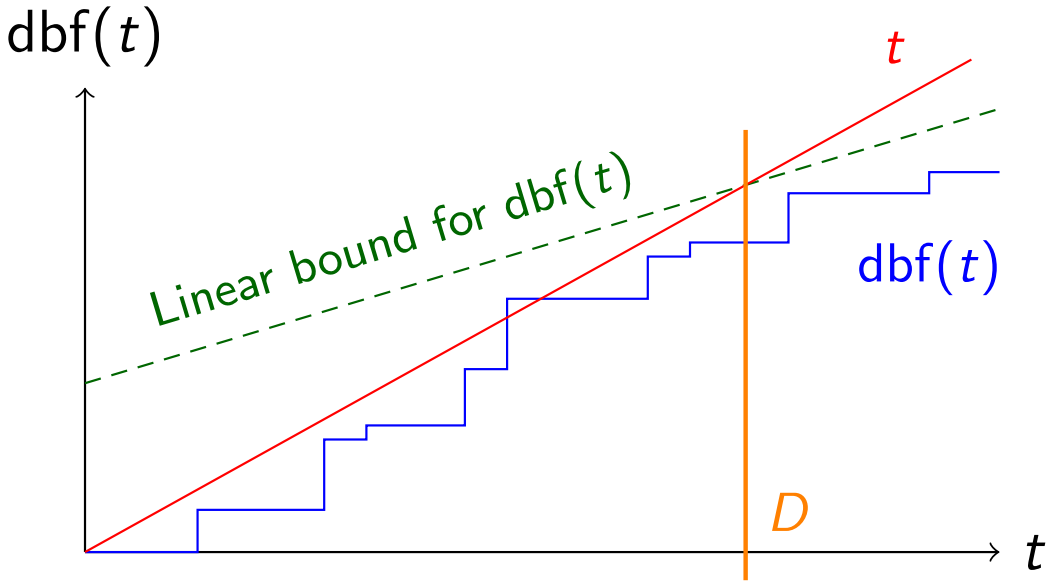
Path (v_4, v_2, v_3)
$\rightsquigarrow \langle 9, 43, v_3 \rangle$

Iterative Procedure

- Create all demand triples up to D:
  1. Start with all *0-paths*, i.e., $\langle e(v), d(v), v \rangle$ for all vertices v
  2. Pick some stored demand triple $\langle e, d, u \rangle$
  3. *Create new demand triple*:
     - ★ Choose successor vertex v of u
     - ★ $e' = e + e(v)$
     - ★ $d' = d - d(u) + p(u, v) + d(v)$
     - ★ $\langle e', d', v \rangle$ is new demand triple!
  4. Store $\langle e', d', v \rangle$ if
     - ★ not stored yet, and
     - ★ $d' \leqslant D$
  5. Repeat from 2 until no change
- *Efficient* procedure!
  - Note: Actual paths never stored
  - Optimizations: Discard non-critical triples along the way
- Exercise: What's $\mathrm{dbf}_{\tau_i}(26)$ for graph on previous slide?

Which t to check?

- Recall: Want to check $\text{dbf}(t) \leqslant t$ for all t
- Find a bound for t just like for sporadic tasks!

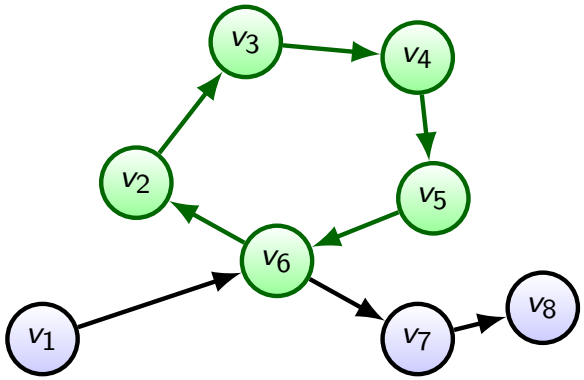


$$\forall t : \text{dbf}(t) \leqslant t$$

- Derive linear bound for $\text{dbf}(t)$
  - Intersection with t gives bound D
  - *How to find the linear bound?*

Linear Bound for dbf

- Bound is based on *utilization*
  - Long-term demand "rate" (asymptotic)
  - For DRT: "most dense" cycle
  - Highest ratio execution demand (vertices) vs. duration (edges)
    - ★ How to find this value? (Exercise!)

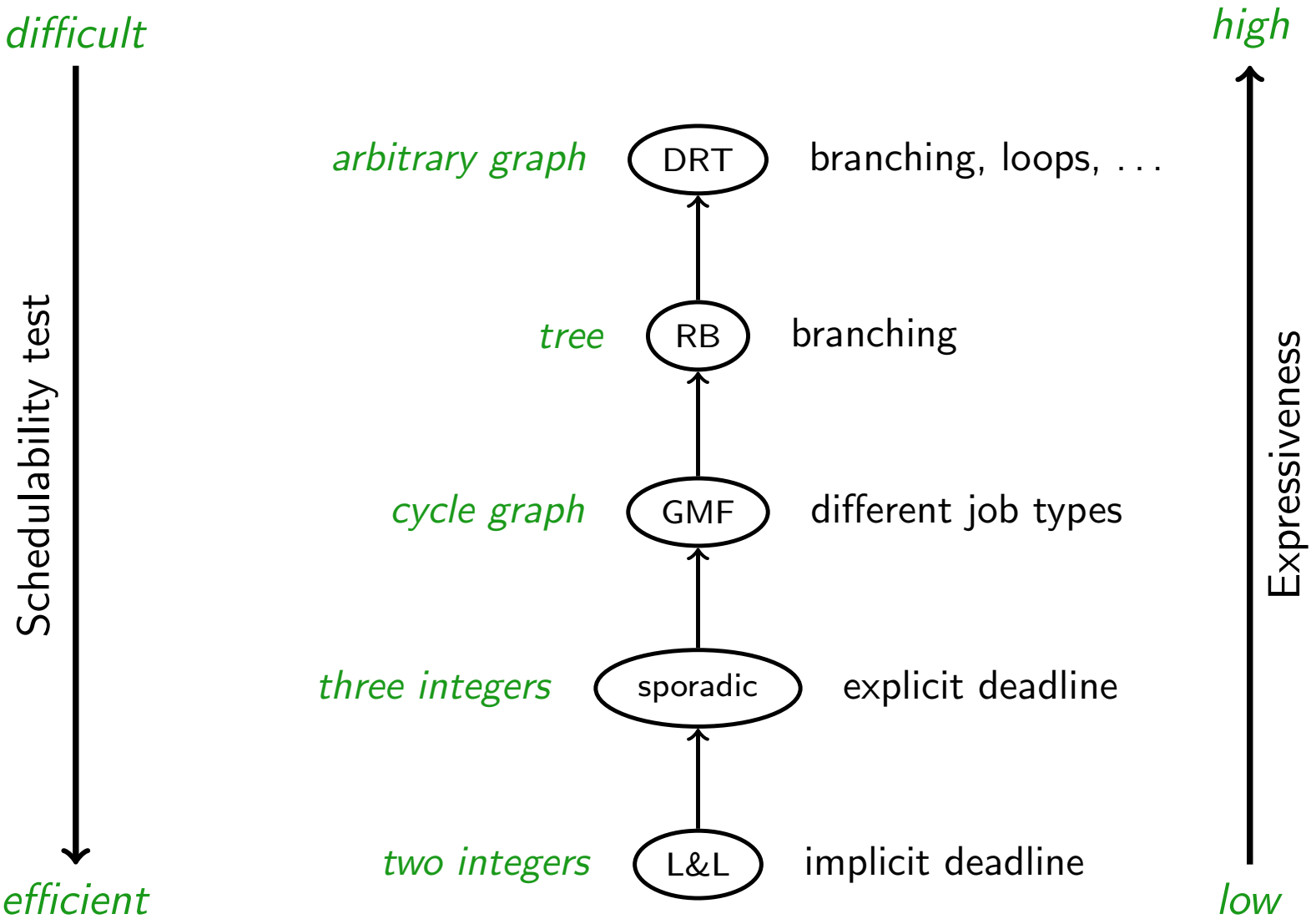


- Worst behavior: Prefix + Cycle + Suffix
- Leads to $\mathrm{dbf}_{\tau_i}(t) \leqslant t \cdot U(\tau_i) + \sum_j e(v_j)$
  - So, check $\mathrm{dbf}(t)$ for which t? (Exercise!)

DRT Schedulability: Summary

- Schedulability test for *EDF*, based on dbf
- First, compute *utilization* for all tasks
  - Based on most dense cycles in graphs
- Derive *bound D*
- Compute $\mathrm{dbf}(t)$ for all $t \leqslant D$
  - Uses iterative procedure with *demand triples*
  - Path abstraction to reduce complexity
- If $t \leqslant D$ with $\mathrm{dbf}(t) > t$ found: τ unschedulable
- Else: τ schedulable

Hierarchy of Models



References


M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The Digraph Real-Time Task Model," in *Proc. of RTAS 2011*, pp. 71–80.


M. Stigge, "Real-time workload models : Expressiveness vs. analysis efficiency," Ph.D. dissertation, Uppsala University, 2014, http://uu.diva-portal.org/smash/record.jsf?pid=diva2%3A699155&dswid=-2116.