Mathematical Tools for Engineering and Management

Lecture 13

25 Jan 2012

- Models, Data and Algorithms
- Linear Optimization
- Mathematical Background: Polyhedra, Simplex-Algorithm
- Sensitivity Analysis; (Mixed) Integer Programming
- MIP Modelling
- MIP Modelling: More Examples; Branch & Bound
- Cutting Planes; Combinatorial Optimization: Examples, Graphs, Algorithms
- TSP-Heuristics
- Network Flows
- Shortest Path Problem
- Complexity Theory
- Nonlinear Optimization
- Scheduling
- Lot Sizing & Intro to Multiobjective Optimization (Feb 01)
- Summary (Feb 08)
- Oral exam (Feb 15)

Printing machine

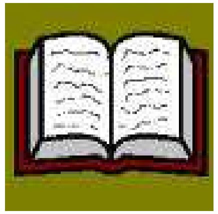
Printing machine

Jobs

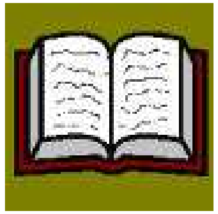
Example: printing machine

Printing machine

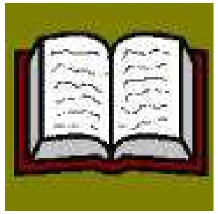
Jobs



Job 1: Book
200 pages, 500 copies
3h printing time



Job 2: Book
60 pages, 2500 copies
4h printing time

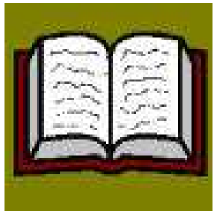


Job 3: Thesis
170 pages, 10 copies
1h printing time

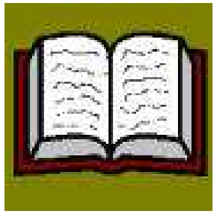
Example: printing machine

Printing machine

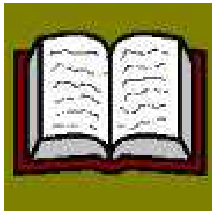
Jobs



Job 1: Book
200 pages, 500 copies
3h printing time



Job 2: Book
60 pages, 2500 copies
4h printing time

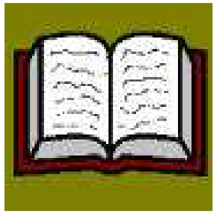


Job 3: Thesis
170 pages, 10 copies
1h printing time

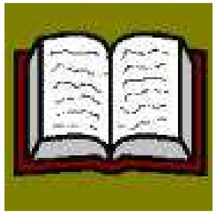
▷ Determine an optimal order for the jobs to be processed...

Printing machine

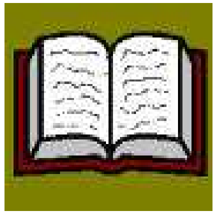
Jobs



Job 1: Book
200 pages, 500 copies
3h printing time



Job 2: Book
60 pages, 2500 copies
4h printing time



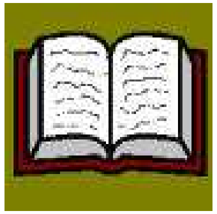
Job 3: Thesis
170 pages, 10 copies
1h printing time

- Determine an optimal order for the jobs to be processed...

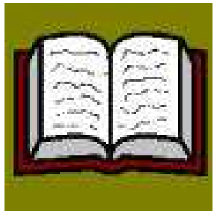
➡ ...if jobs have to be finished at a given time

Printing machine

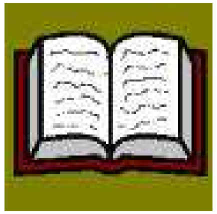
Jobs



Job 1: Book
200 pages, 500 copies
3h printing time



Job 2: Book
60 pages, 2500 copies
4h printing time



Job 3: Thesis
170 pages, 10 copies
1h printing time

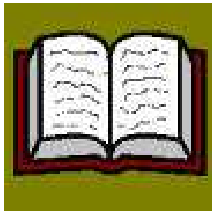
▷ Determine an optimal order for the jobs to be processed...

➡ ...if jobs have to be finished at a given time

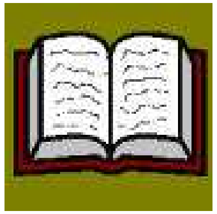
➡ ...if some jobs are more important than others

Printing machine

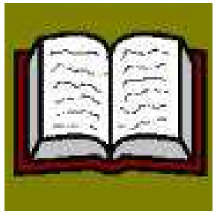
Jobs



Job 1: Book
200 pages, 500 copies
3h printing time



Job 2: Book
60 pages, 2500 copies
4h printing time

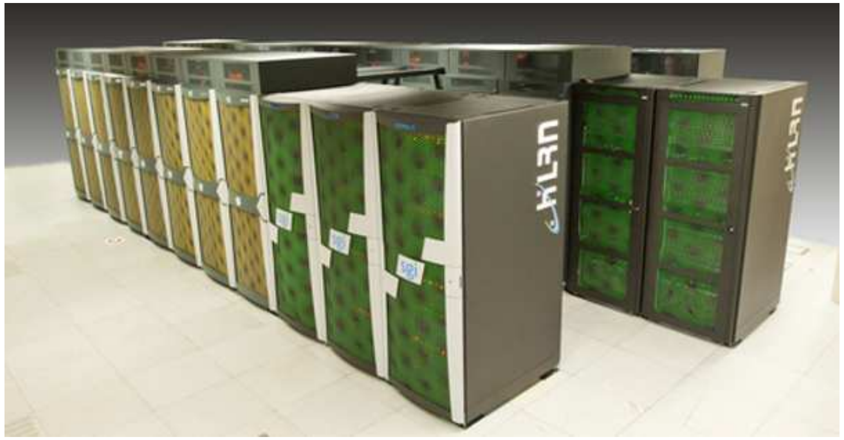


Job 3: Thesis
170 pages, 10 copies
1h printing time

▷ Determine an optimal order for the jobs to be processed...

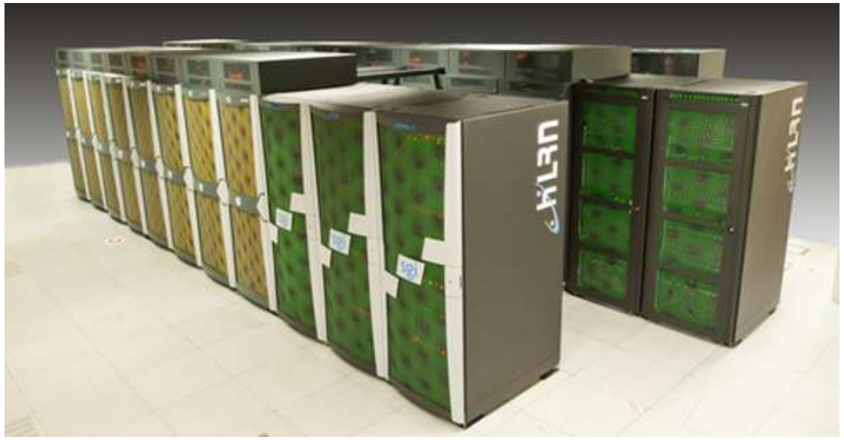
- ➡ ...if jobs have to be finished at a given time
- ➡ ...if some jobs are more important than others
- ➡ ...if there is more than one machine (identical or different machines)

- Supercomputing at ZIB



- Supercomputing at ZIB

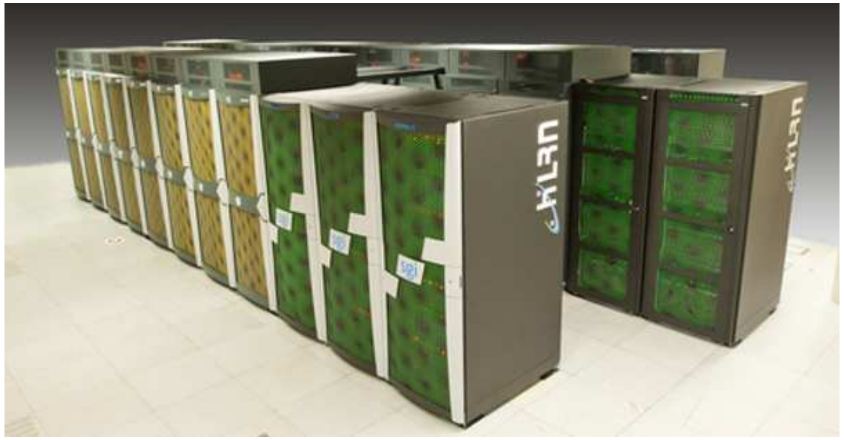
Norddeutscher Verbund für Hoch- und Höchstleistungsrechnen



- ~1500 compute nodes with ~13000 cores

- Supercomputing at ZIB

HLRN
Norddeutscher Verbund für Hoch- und Höchstleistungsrechnen



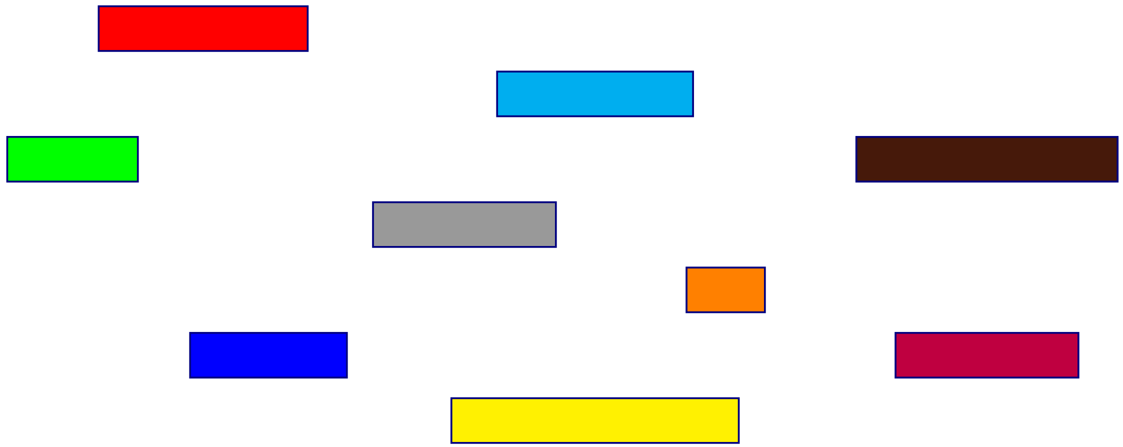
- ∼1500 compute nodes with ∼13000 cores
- Schedule computation jobs...

  ...consisting of thousands of parallel processes

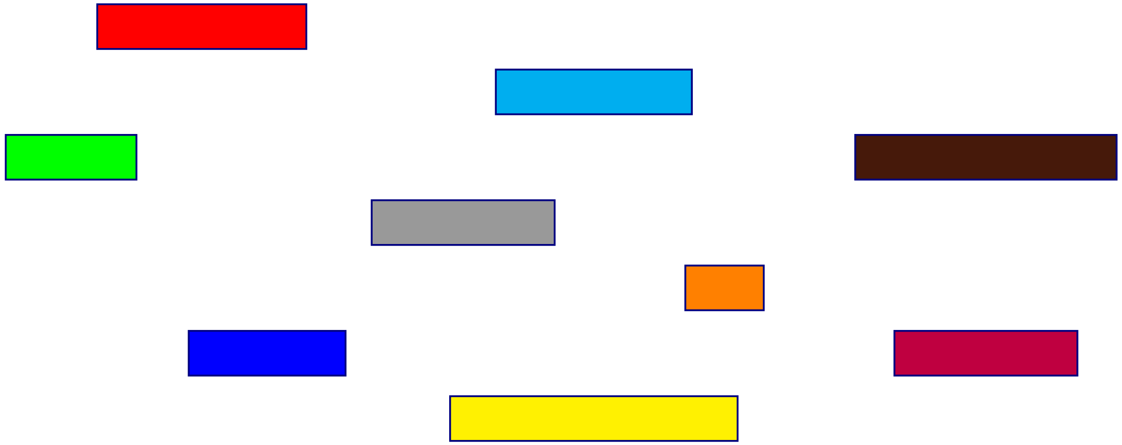
  ...according to their release times

General principle

- Jobs:

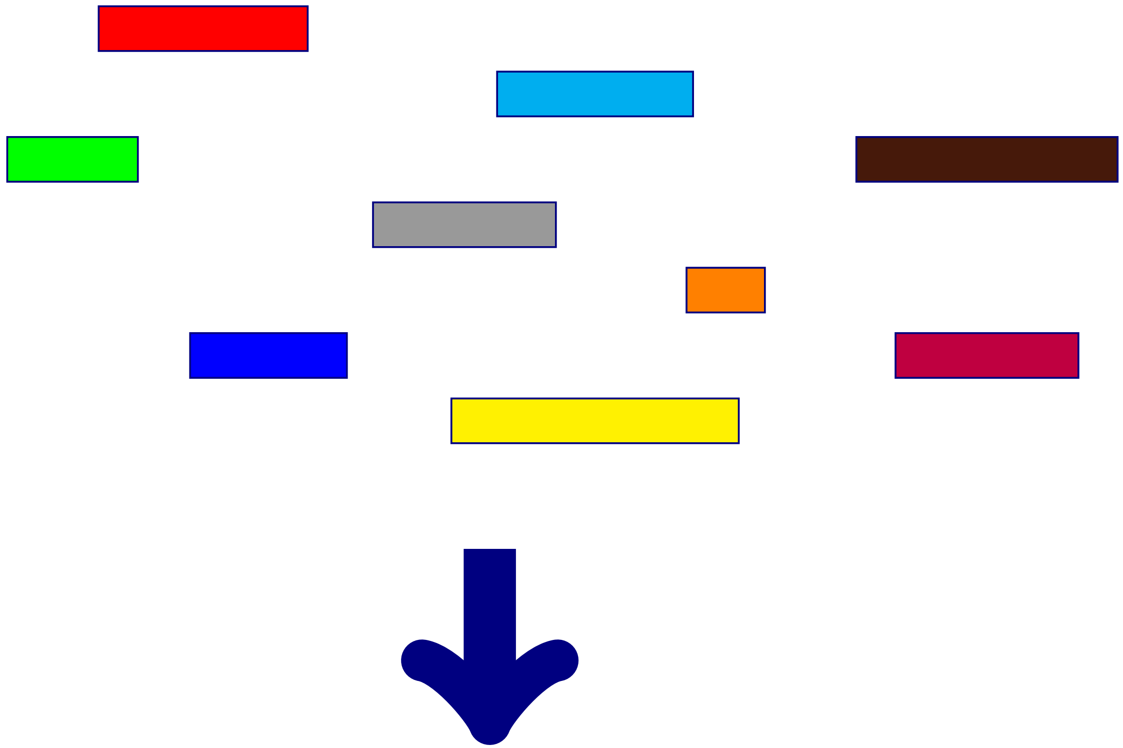


- Jobs:

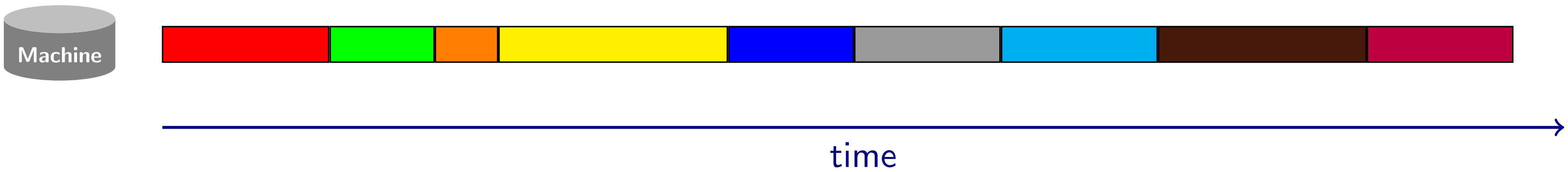


- Schedule (Gantt chart):

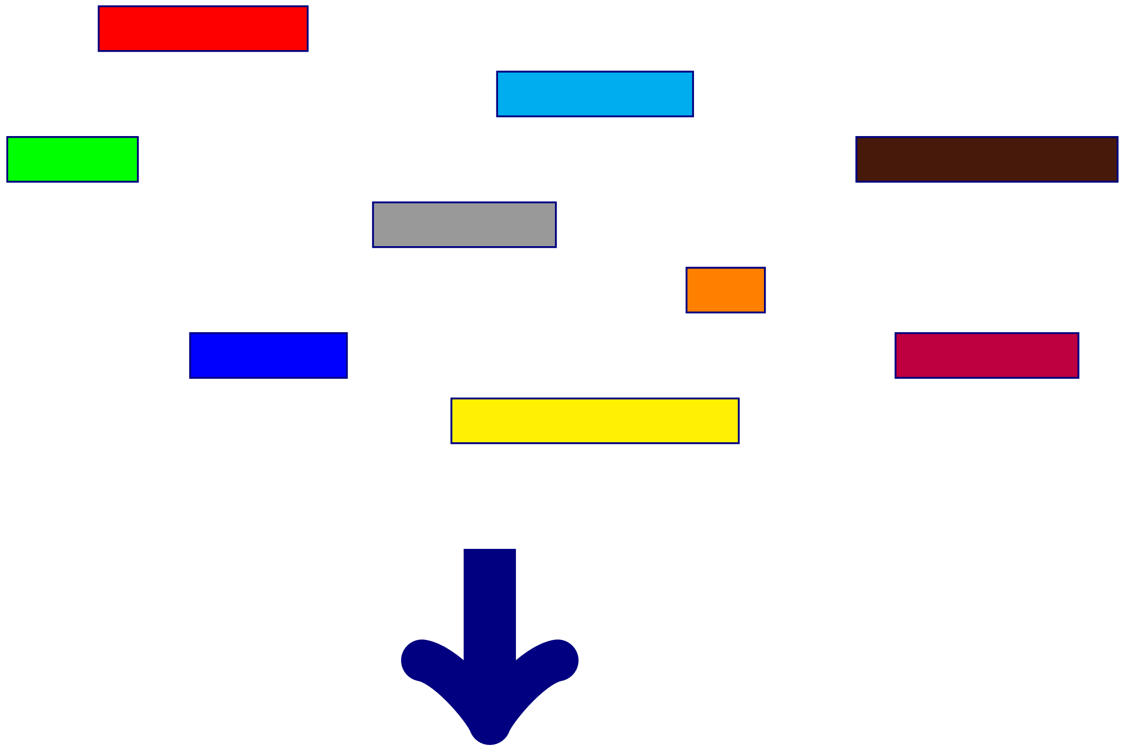
- Jobs:



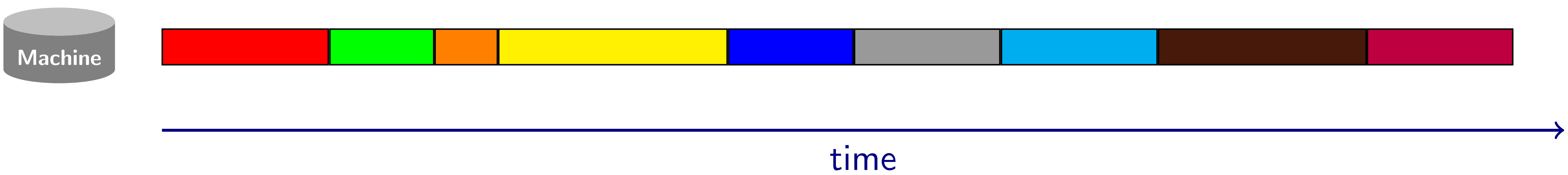
- Schedule (Gantt chart):



- ▷ Jobs:

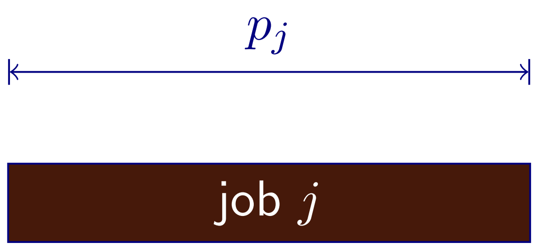


- ▷ Schedule (Gantt chart): ➡ optimal with respect to an objective to specify!

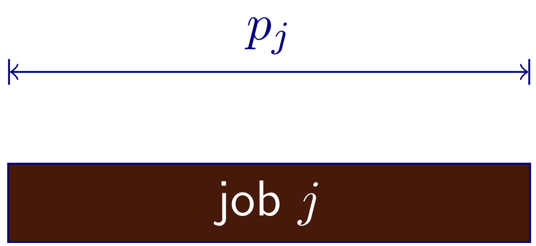


- Jobs usually have: a processing time p_j

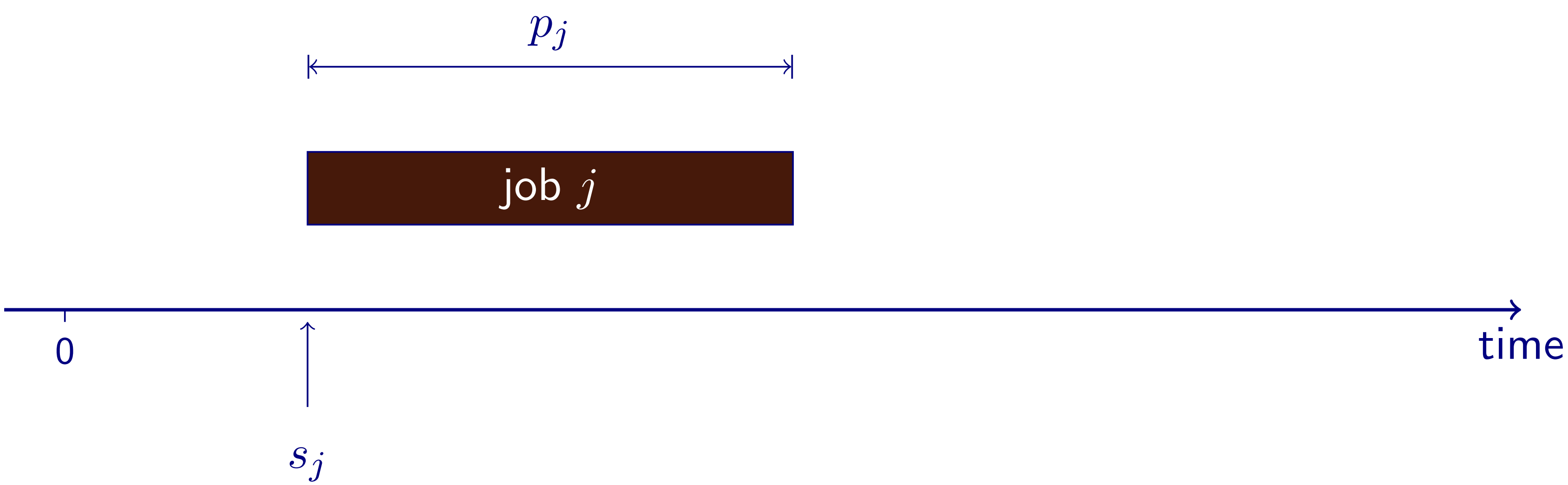
- Jobs usually have: a processing time p_j



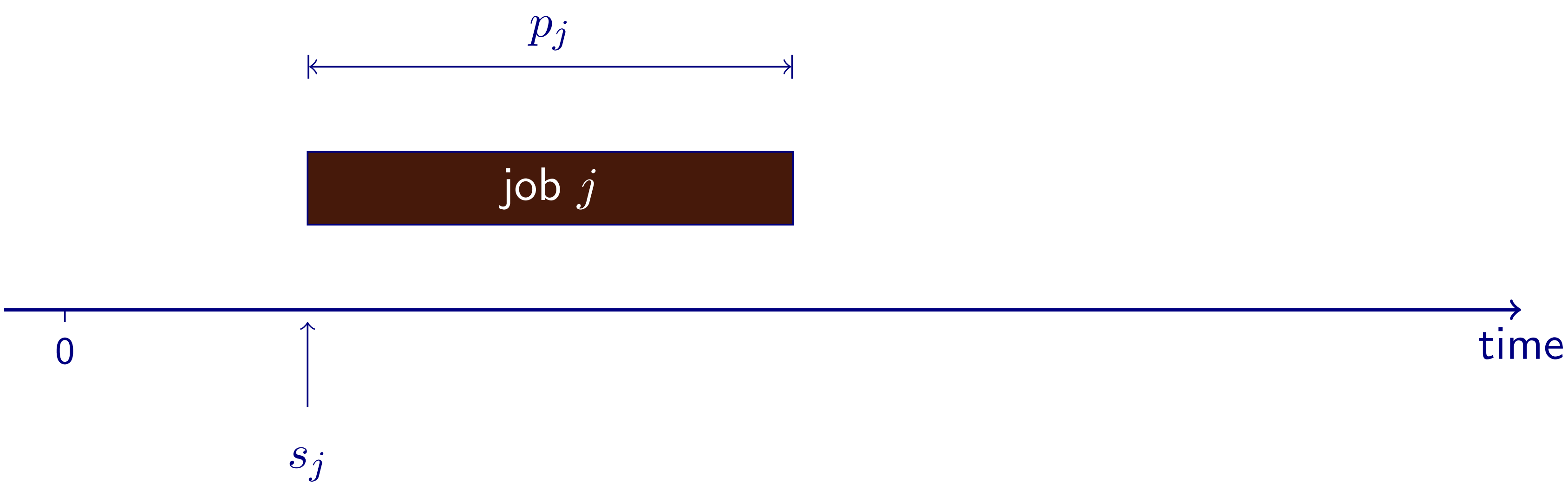
- Jobs usually have: a **processing time** p_j
- A schedule has to provide: a **start time** s_j



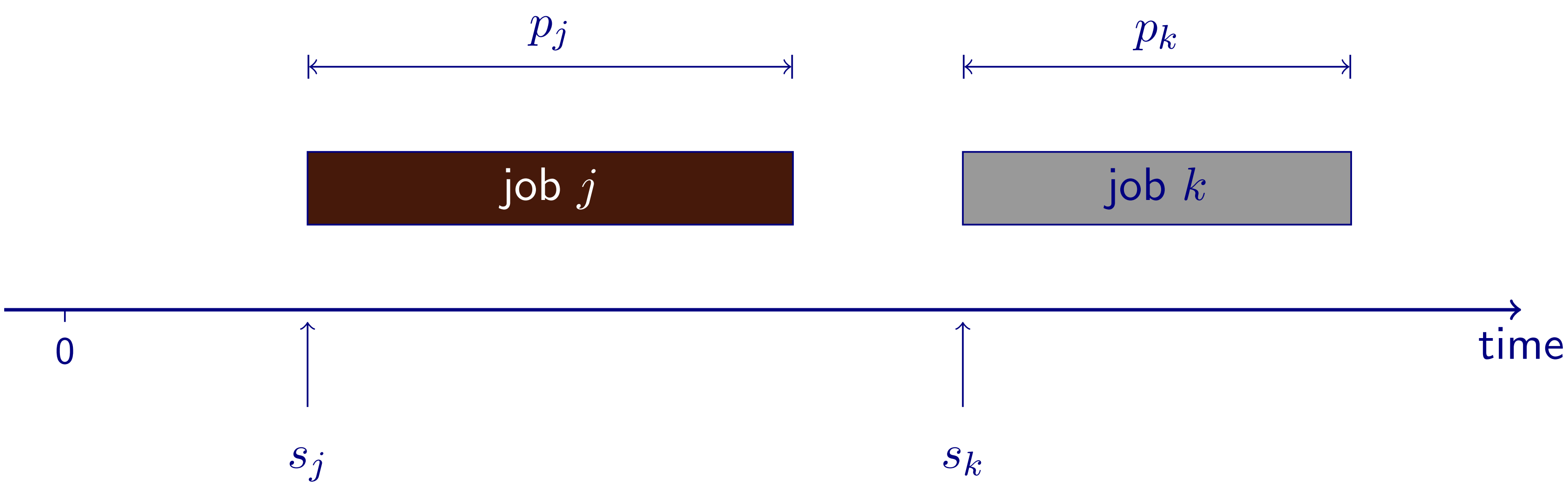
- Jobs usually have: a **processing time** p_j
- A schedule has to provide: a **start time** s_j



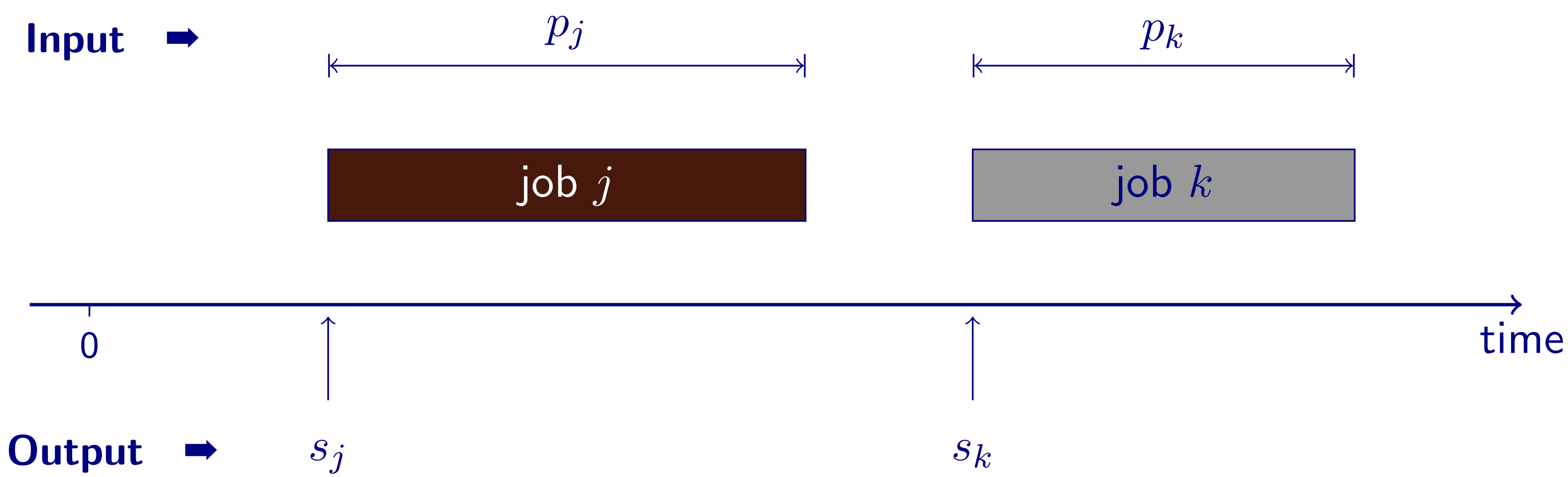
- Jobs usually have: a **processing time** p_j
- A schedule has to provide: a **start time** s_j, such that different jobs do not overlap



- Jobs usually have: a **processing time** p_j
- A schedule has to provide: a **start time** s_j, such that different jobs do not overlap

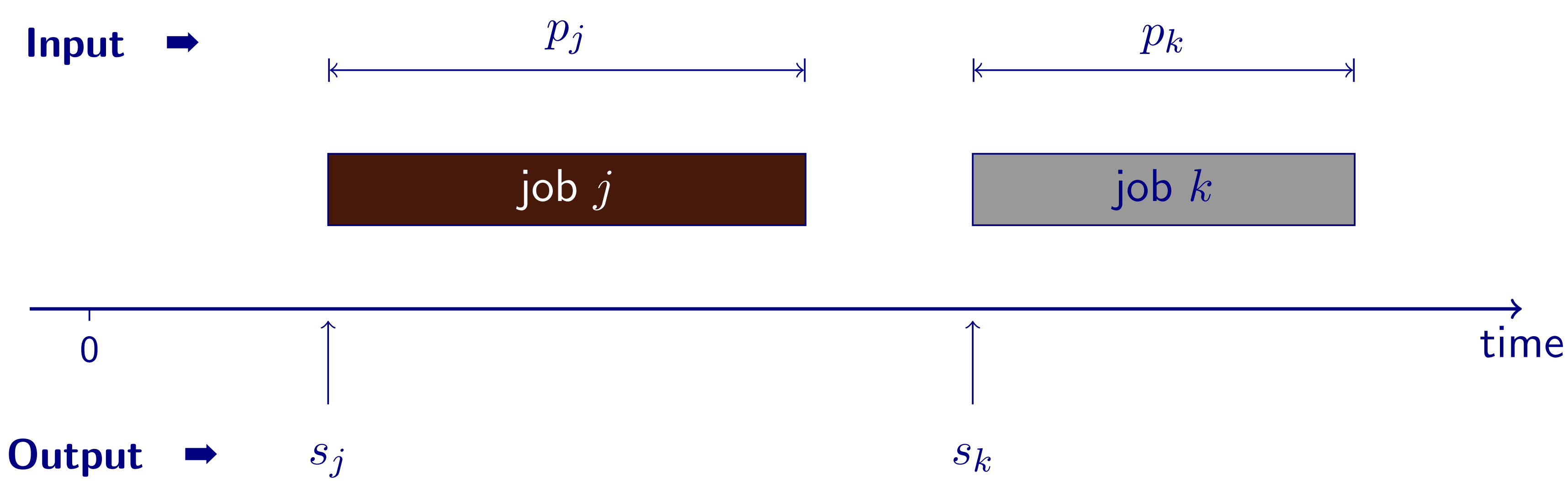


- Jobs usually have: a **processing time** p_j
- A schedule has to provide: a **start time** s_j, such that different jobs do not overlap



- ▷ Jobs usually have: a **processing time** p_j
- ▷ A schedule has to provide: a **start time** s_j, such that different jobs do not overlap

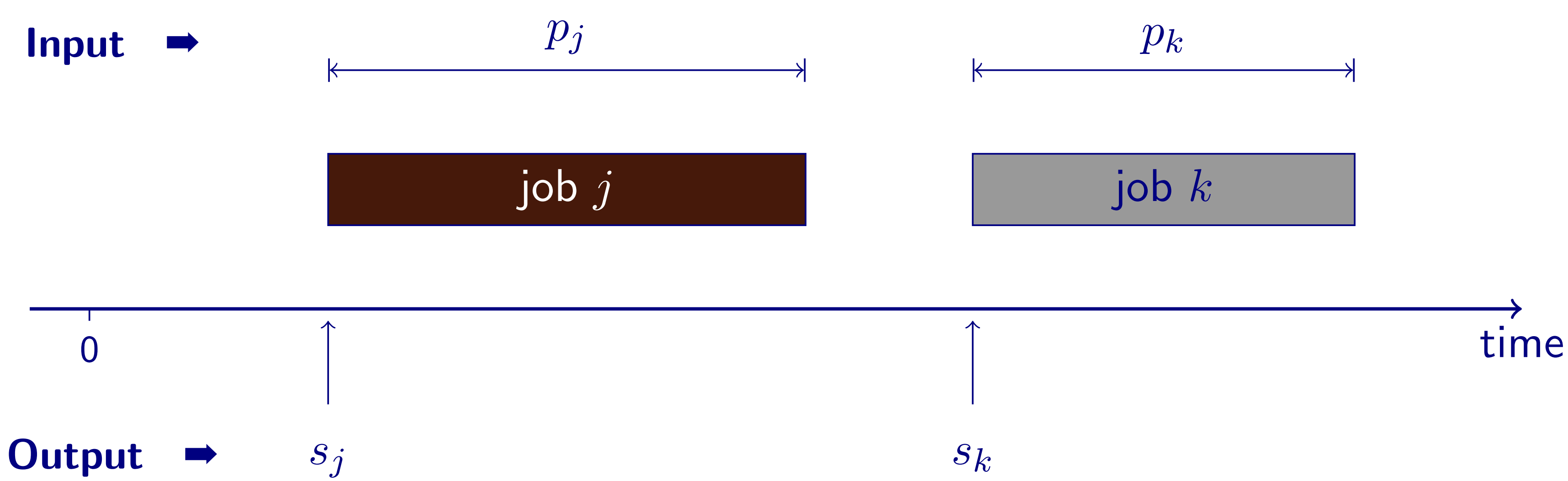
➡ Combinatorial optimization problem



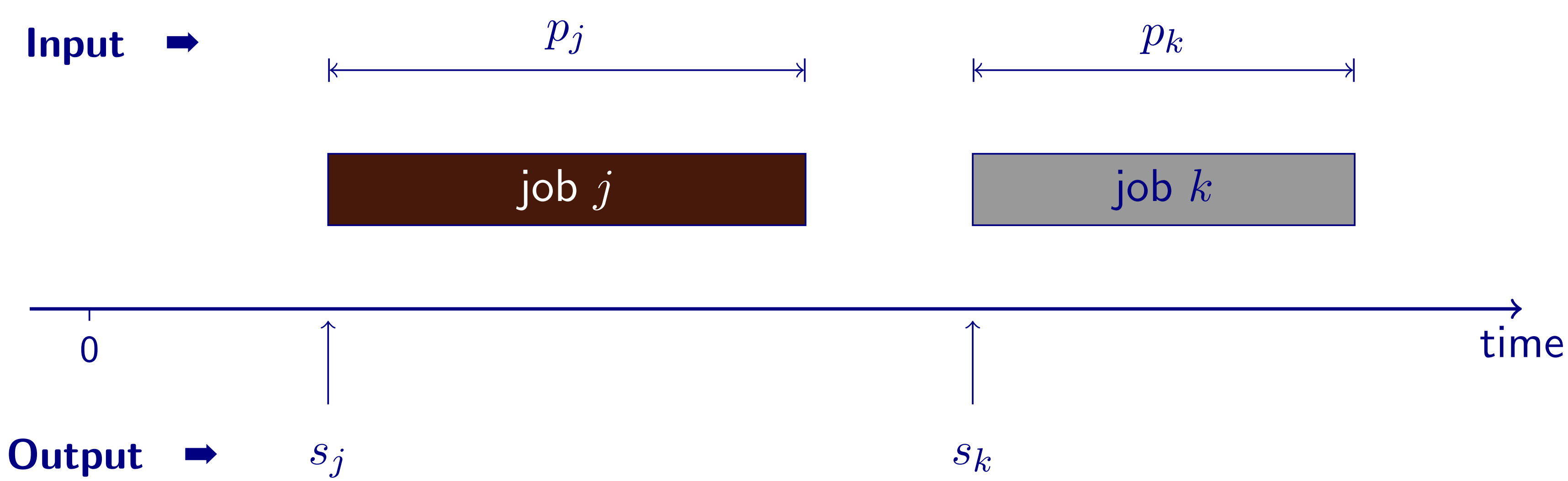
▷ Jobs usually have: a processing time p_j

▷ A schedule has to provide: a start time s_j, such that different jobs do not overlap

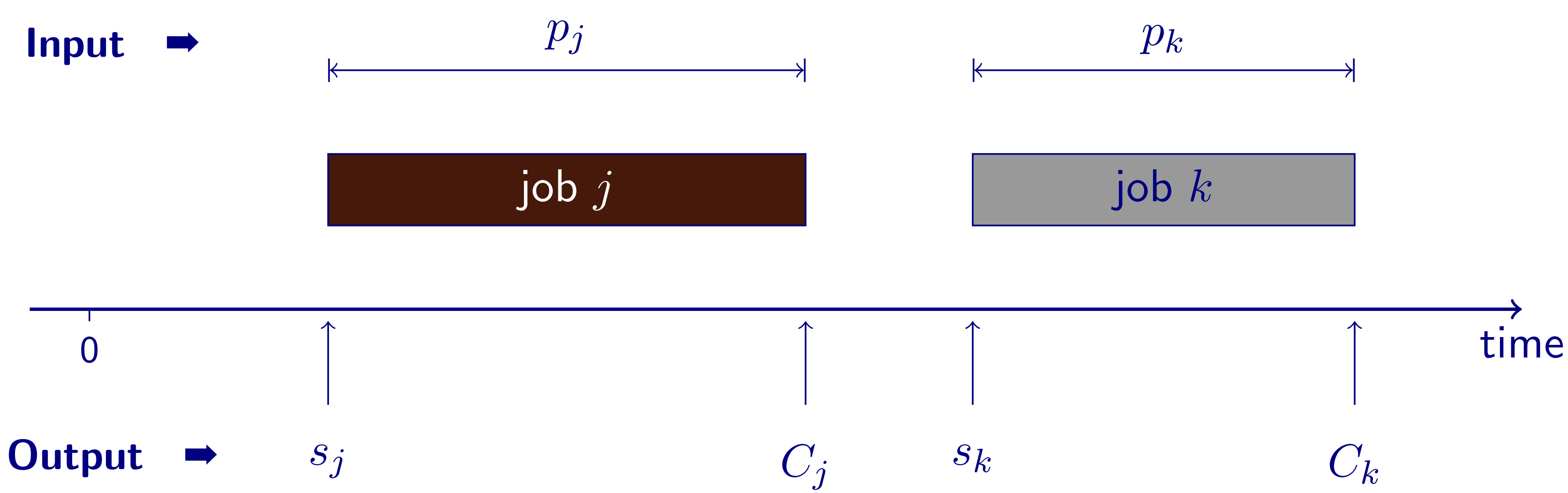
➡ Combinatorial optimization problem ➡ IP formulations for most scheduling problems



- ▷ Jobs usually have: a **processing time** p_j
- ▷ A schedule has to provide: a **start time** s_j, such that different jobs do not overlap
- ➡ Combinatorial optimization problem ➡ IP formulations for most scheduling problems
- ➡ **Completion time** $C_j := s_j + p_j$



- ▷ Jobs usually have: a **processing time** p_j
- ▷ A schedule has to provide: a **start time** s_j, such that different jobs do not overlap
- ➡ Combinatorial optimization problem ➡ IP formulations for most scheduling problems
- ➡ **Completion time** $C_j := s_j + p_j$



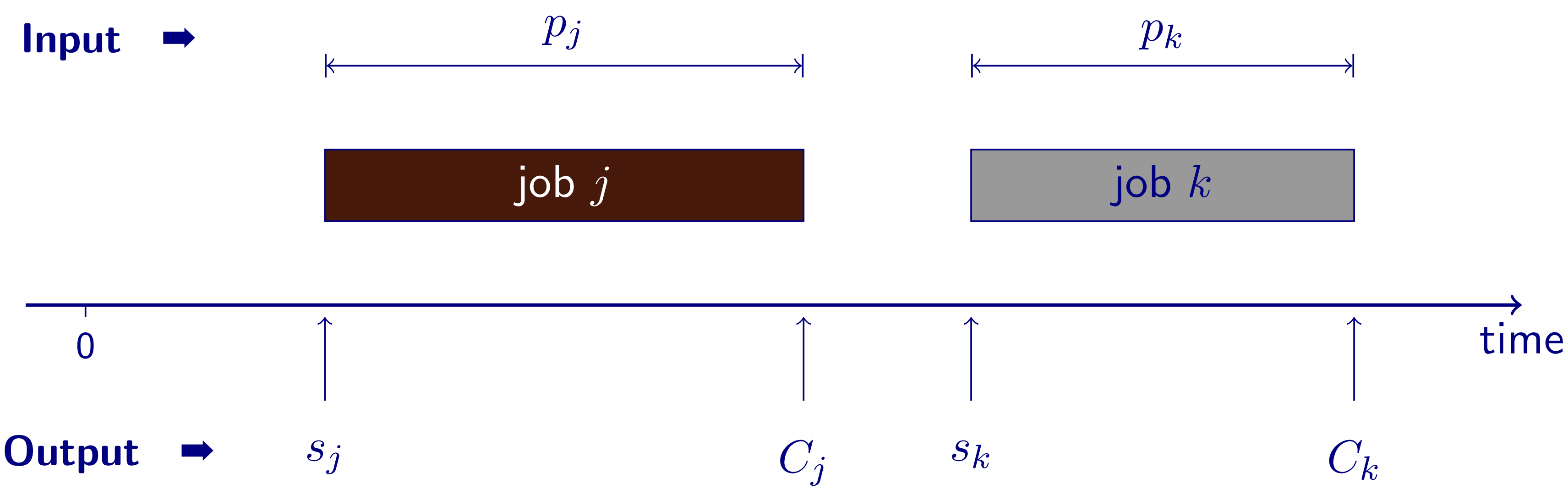
Notation

- Jobs usually have: a processing time p_j
- A schedule has to provide: a start time s_j, such that different jobs do not overlap

➡ Combinatorial optimization problem ➡ IP formulations for most scheduling problems

➡ Completion time $C_j := s_j + p_j$

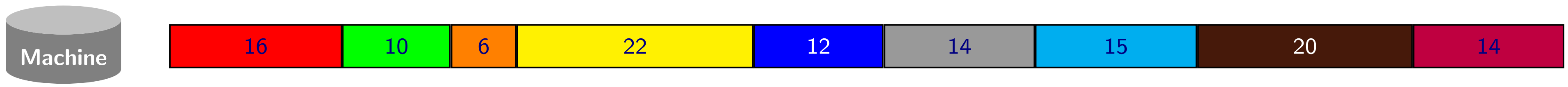
➡ Average completion time for n jobs: $\frac{1}{n}\sum_{j=1}^{n} C_j$



- For fixed number n of jobs: minimize sum of completion times $\sum_{j=1}^{n} C_j$

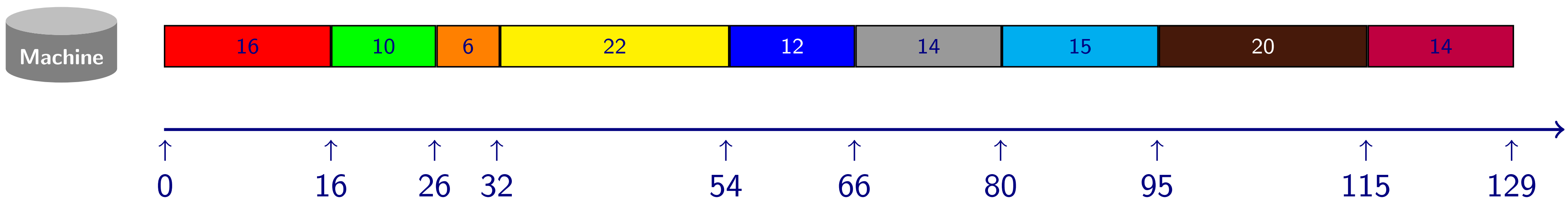
Minimize sum of completion times

- For fixed number n of jobs: minimize sum of completion times $\sum_{j=1}^{n} C_j$
- Example schedule:

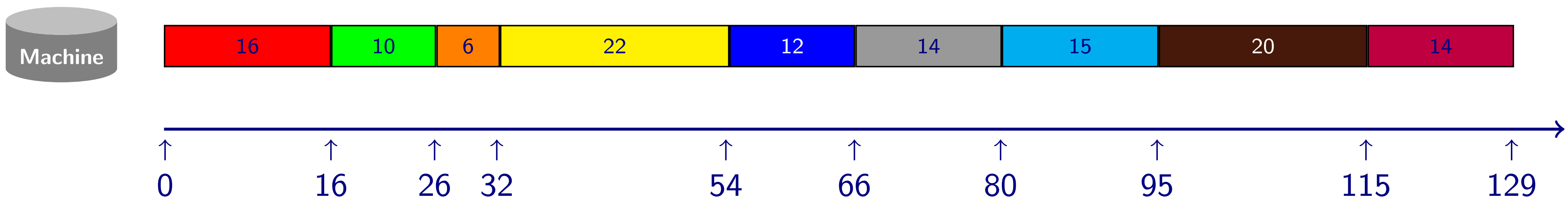


Minimize sum of completion times

- For fixed number n of jobs: minimize **sum of completion times** $\sum_{j=1}^{n} C_j$
- Example schedule:

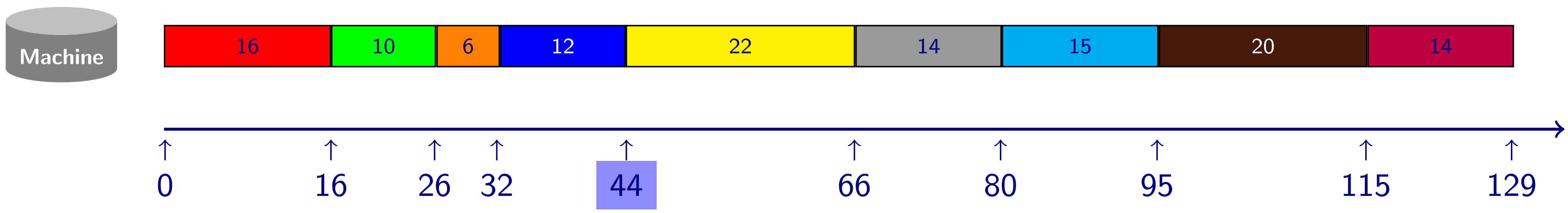


- For fixed number n of jobs: minimize **sum of completion times** $\sum_{j=1}^{n} C_j$
- Example schedule:



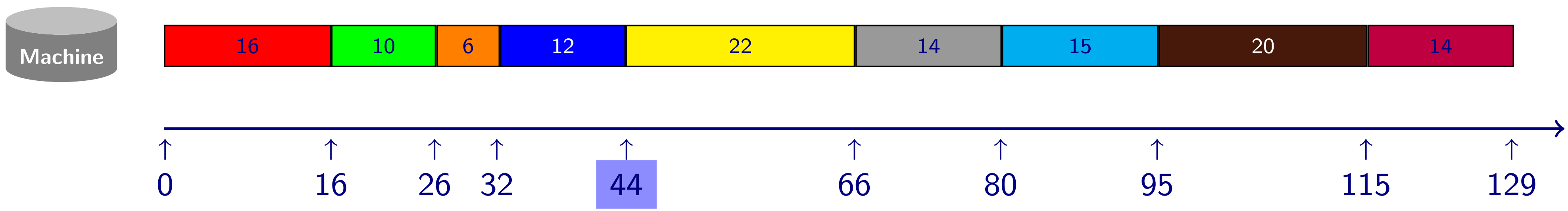
$$\Rightarrow \sum_{j=1}^{n} C_j = 16 + 26 + 32 + 54 + 66 + 80 + 95 + 115 + 129 = 613$$

- For fixed number n of jobs: minimize **sum of completion times** $\sum_{j=1}^{n} C_j$
- Example schedule:



$$\Rightarrow \sum_{j=1}^{n} C_j = 16 + 26 + 32 + 44 + 66 + 80 + 95 + 115 + 129 = 603$$

- For fixed number n of jobs: minimize **sum of completion times** $\sum_{j=1}^{n} C_j$
- Example schedule:

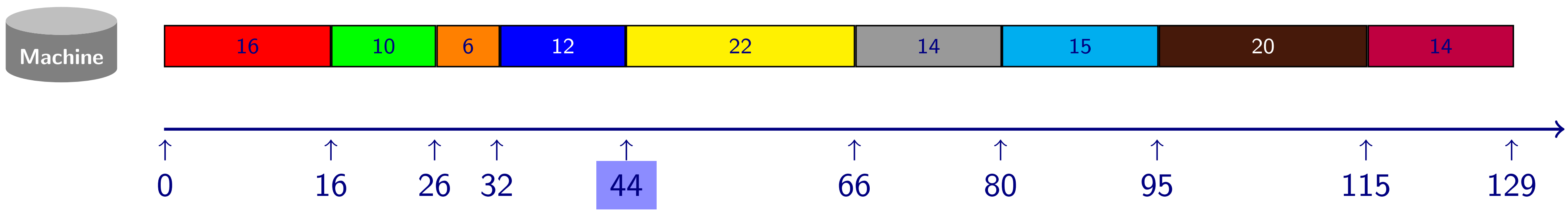


➡ $$\sum_{j=1}^{n} C_j = 16 + 26 + 32 + \boxed{44} + 66 + 80 + 95 + 115 + 129 = \boxed{603}$$

➡ Idea: Schedule jobs in order of non-decreasing processing time!

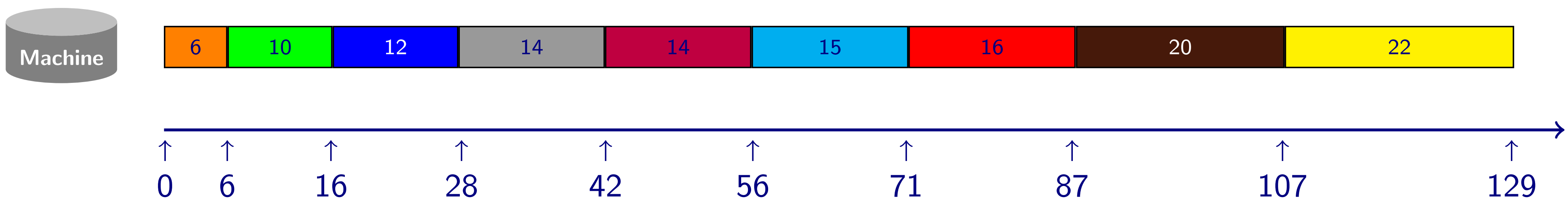
Minimize sum of completion times

- For fixed number n of jobs: minimize **sum of completion times** $\sum_{j=1}^{n} C_j$
- Example schedule:



$$\Rightarrow \sum_{j=1}^{n} C_j = 16 + 26 + 32 + 44 + 66 + 80 + 95 + 115 + 129 = 603$$

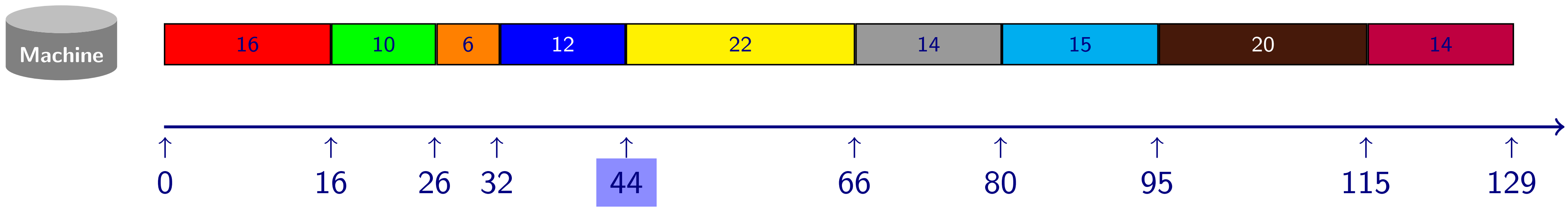
➡ Idea: Schedule jobs in order of non-decreasing processing time!



$$\Rightarrow \sum_{j=1}^{n} C_j = 6 + 16 + 28 + 42 + 56 + 71 + 87 + 107 + 129 = 542$$

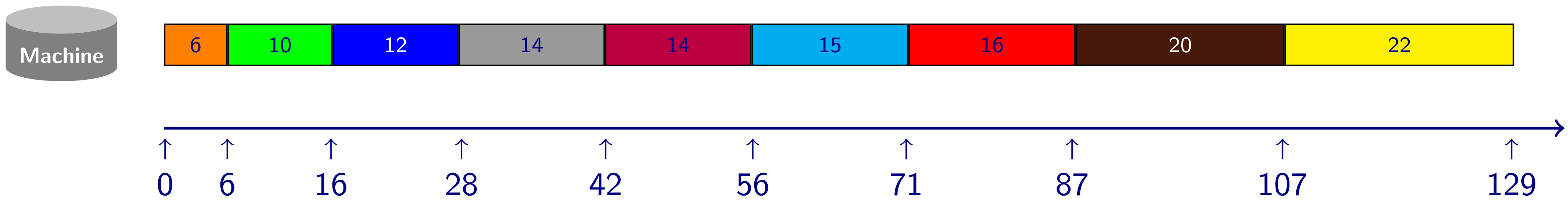
Minimize sum of completion times

- For fixed number n of jobs: minimize **sum of completion times** $\sum_{j=1}^{n} C_j$
- Example schedule:



➡ $$\sum_{j=1}^{n} C_j = 16 + 26 + 32 + 44 + 66 + 80 + 95 + 115 + 129 = 603$$

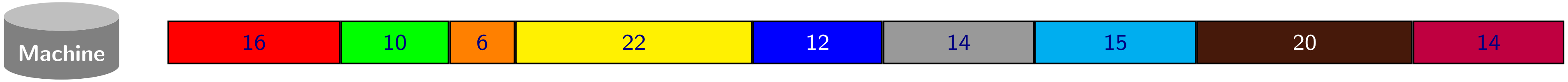
➡ Idea: Schedule jobs in order of non-decreasing processing time! ➡ **provably optimal!**



➡ $$\sum_{j=1}^{n} C_j = 6 + 16 + 28 + 42 + 56 + 71 + 87 + 107 + 129 = 542$$

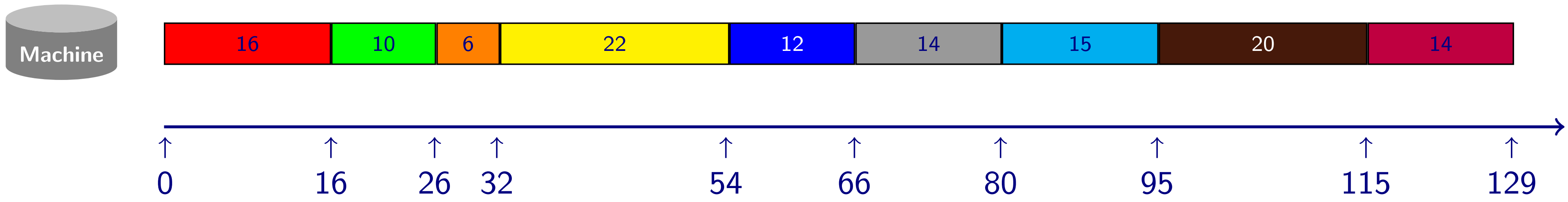
- Latest completion time ➡ minimize makespan $\max_{j=1,\ldots,n} C_j$

- Latest completion time ➡ minimize makespan $\max_{j=1,\ldots,n} C_j$
- Example schedule:

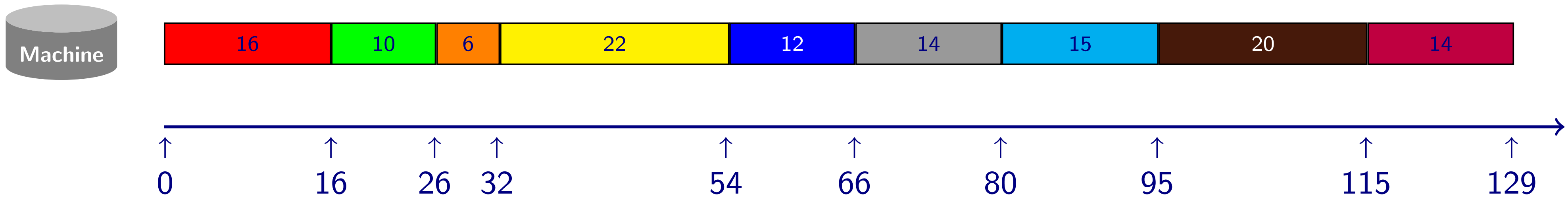


Minimize makespan

- Latest completion time ➡ minimize makespan $\max_{j=1,\dots,n} C_j$
- Example schedule:

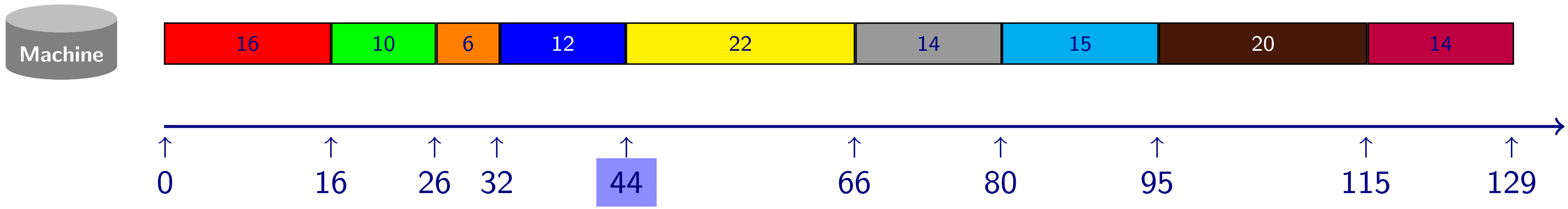


- ▷ Latest completion time ➡ minimize **makespan** $\max_{j=1,\ldots,n} C_j$
- ▷ Example schedule:



➡ makespan: 129

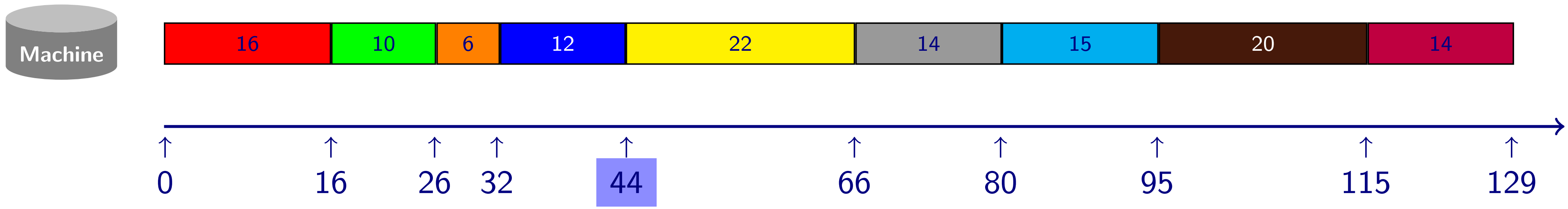
- ▷ Latest completion time ➡ minimize makespan $\max_{j=1,\dots,n} C_j$
- ▷ Example schedule:



➡ makespan: 129

Minimize makespan

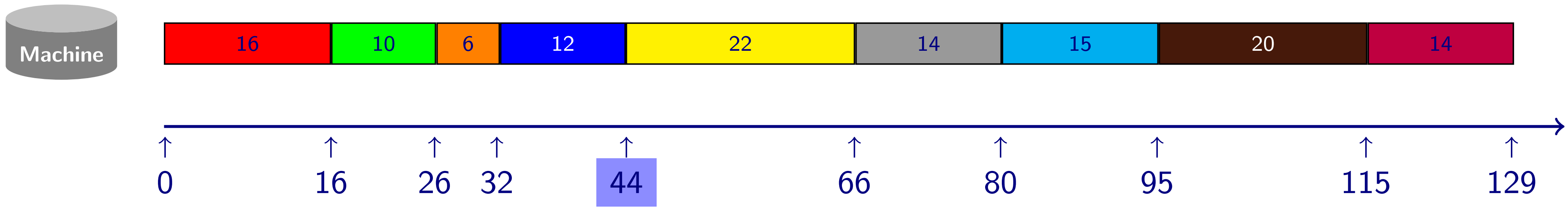
- Latest completion time ➡ minimize makespan $\max_{j=1,\ldots,n} C_j$
- Example schedule:



➡ makespan: 129

➡ Any schedule (without idle times) gives the same makespan!

- Latest completion time ➡ minimize makespan $\max_{j=1,\ldots,n} C_j$
- Example schedule:



➡ makespan: 129

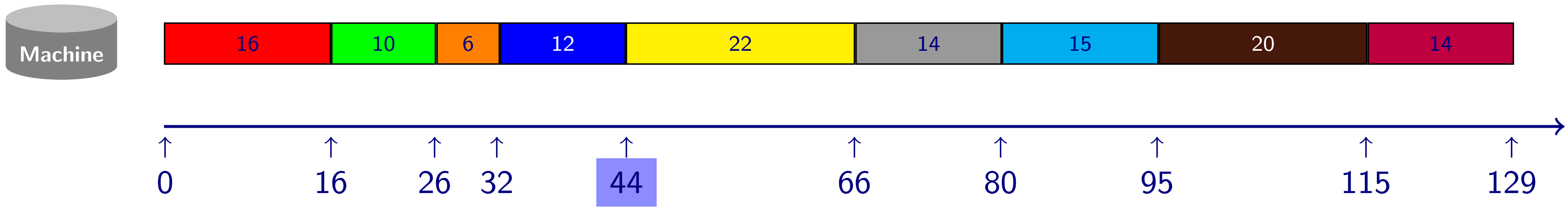
➡ Any schedule (without idle times) gives the same makespan!

Obvious, since the makespan

$$\max_{j=1,\ldots,n} C_j = \sum_{j=1}^{n} p_j$$

Minimize makespan

- Latest completion time ➡ minimize makespan $\max_{j=1,\dots,n} C_j$
- Example schedule:



➡ makespan: 129

➡ Any schedule (without idle times) gives the same makespan!

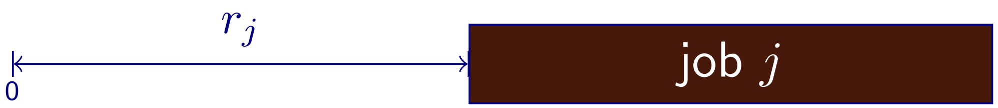
Obvious, since the makespan

$$\max_{j=1,\dots,n} C_j = \sum_{j=1}^{n} p_j$$

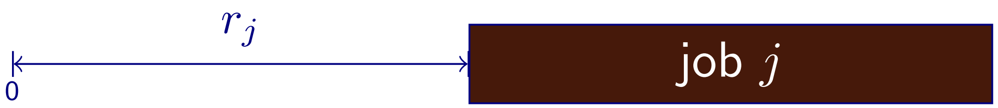
depends only on the input (processing times), not on the schedule itself

- Jobs can have: a release date r_j

- Jobs can have: a release date r_j

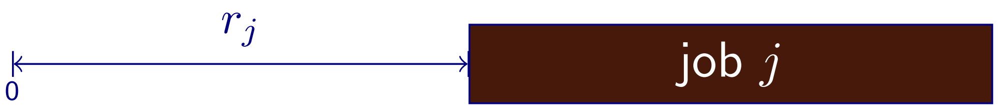


▷ Jobs can have: a release date r_j

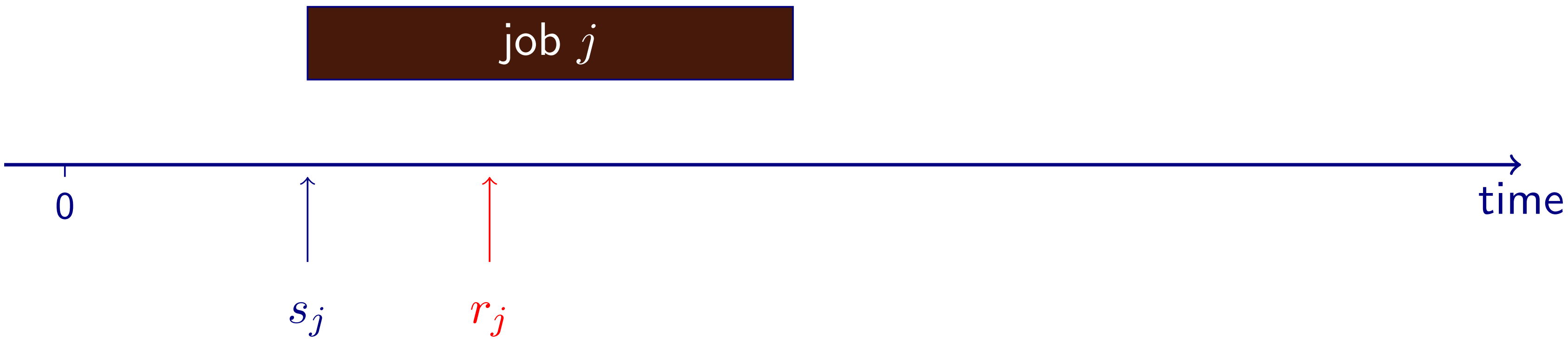


➡ Start time of job j cannot be before its release date

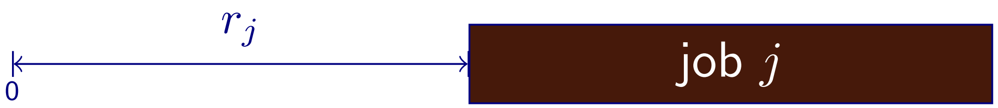
- Jobs can have: a release date r_j



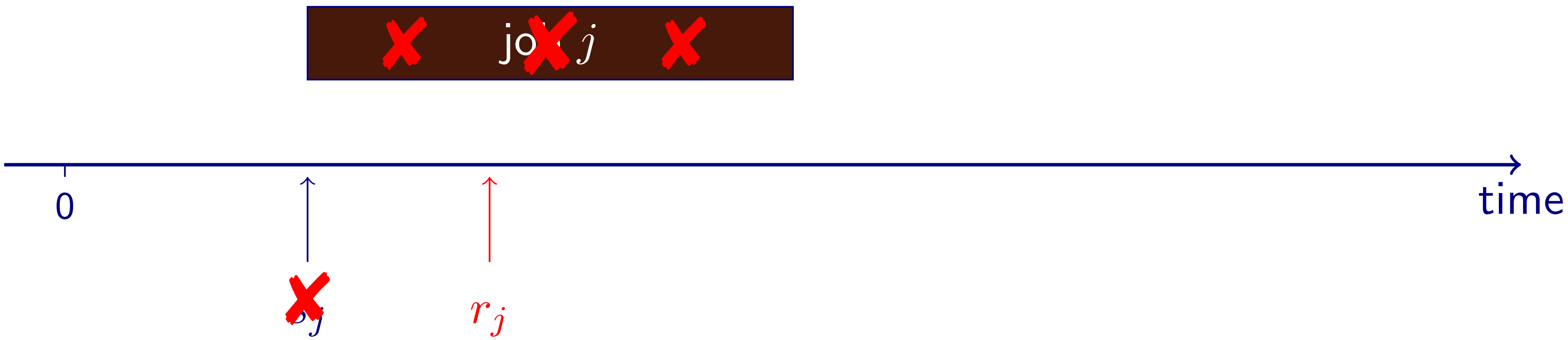
➡ Start time of job j cannot be before its release date



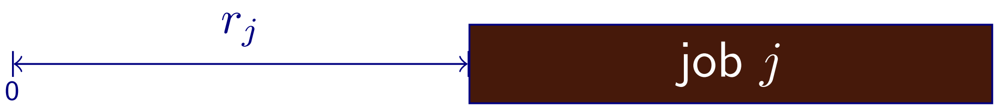
▷ Jobs can have: a release date r_j



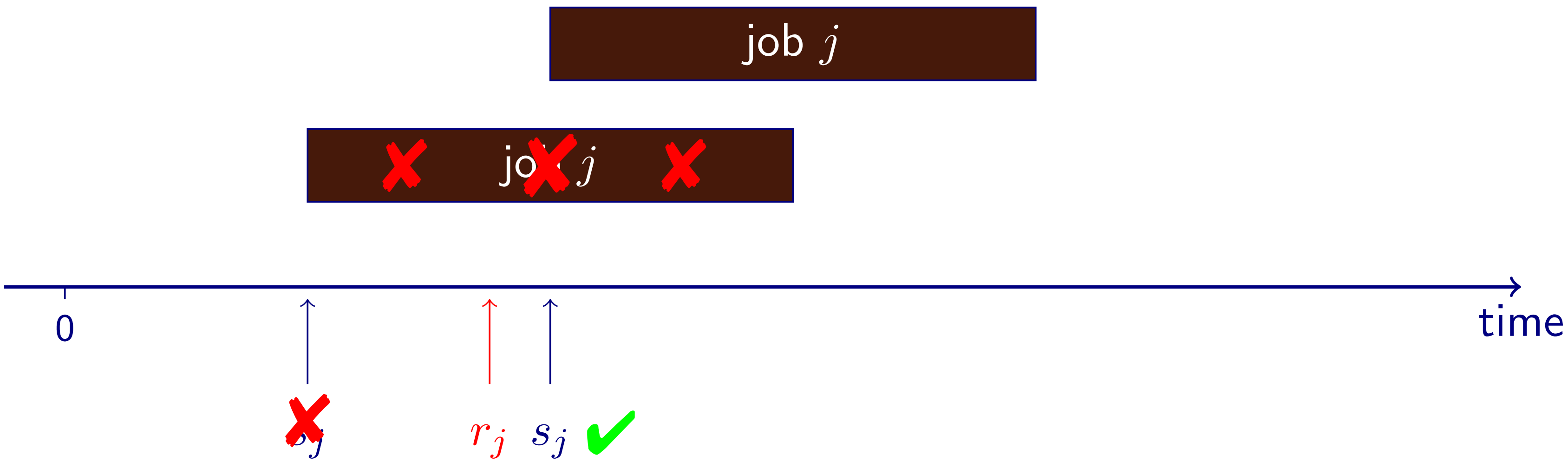
➡ Start time of job j cannot be before its release date



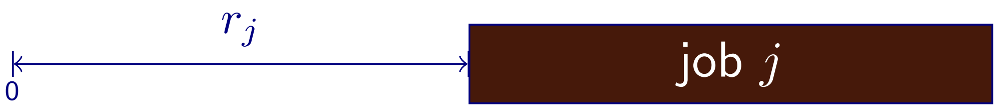
- Jobs can have: a release date r_j



➡ Start time of job j cannot be before its release date

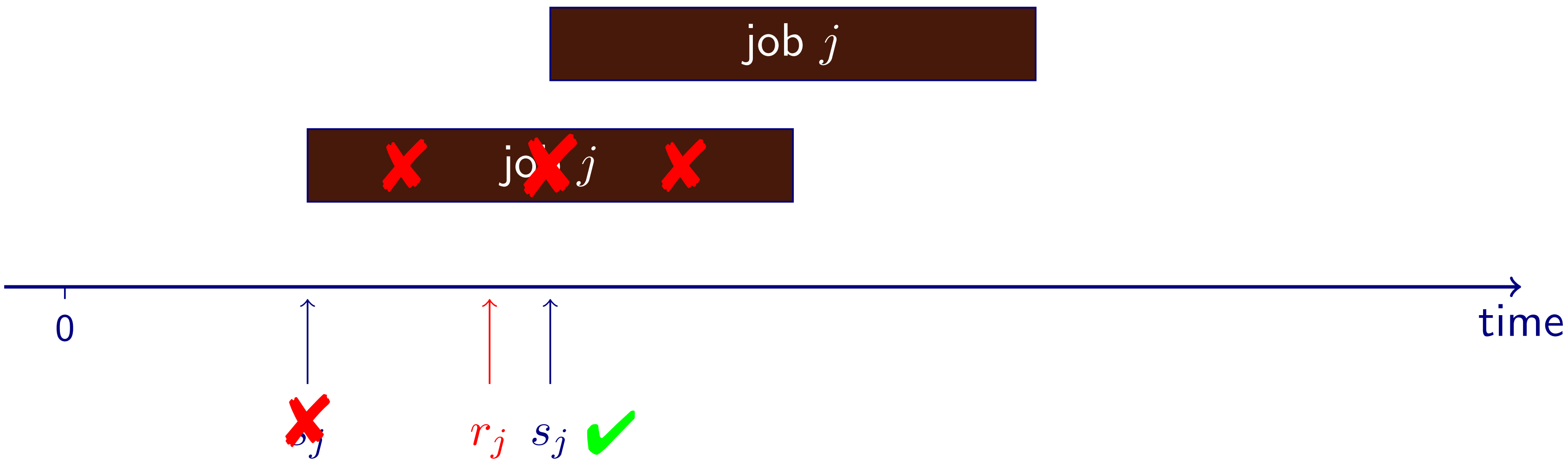


▷ Jobs can have: a release date r_j

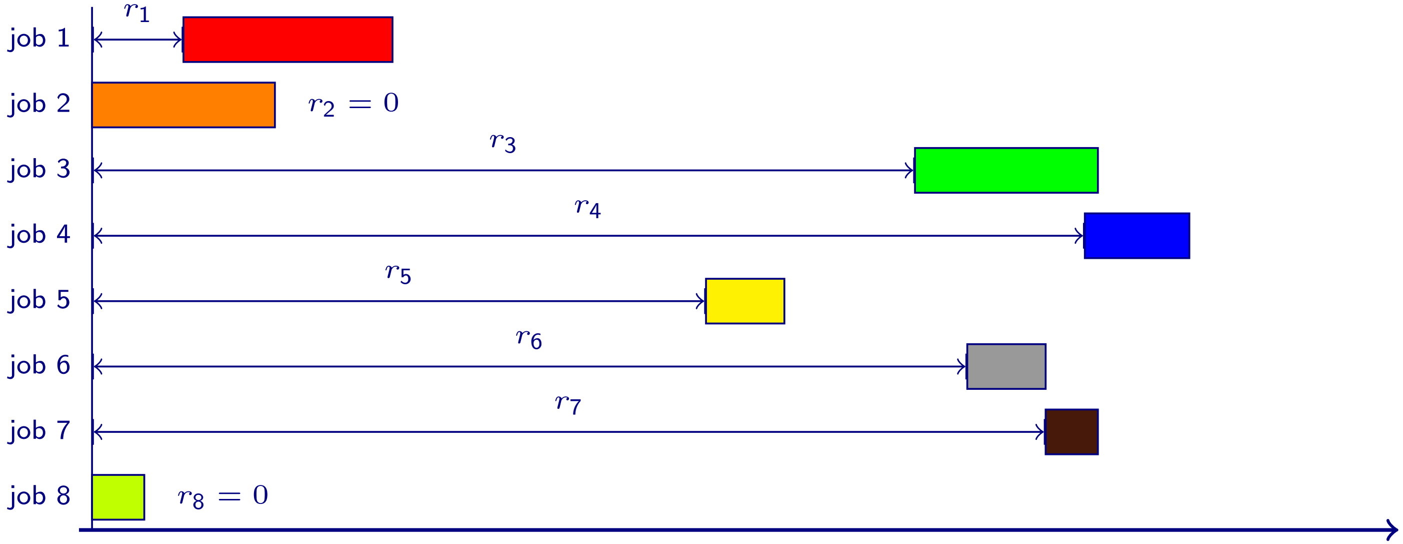


➡ Start time of job j cannot be before its release date

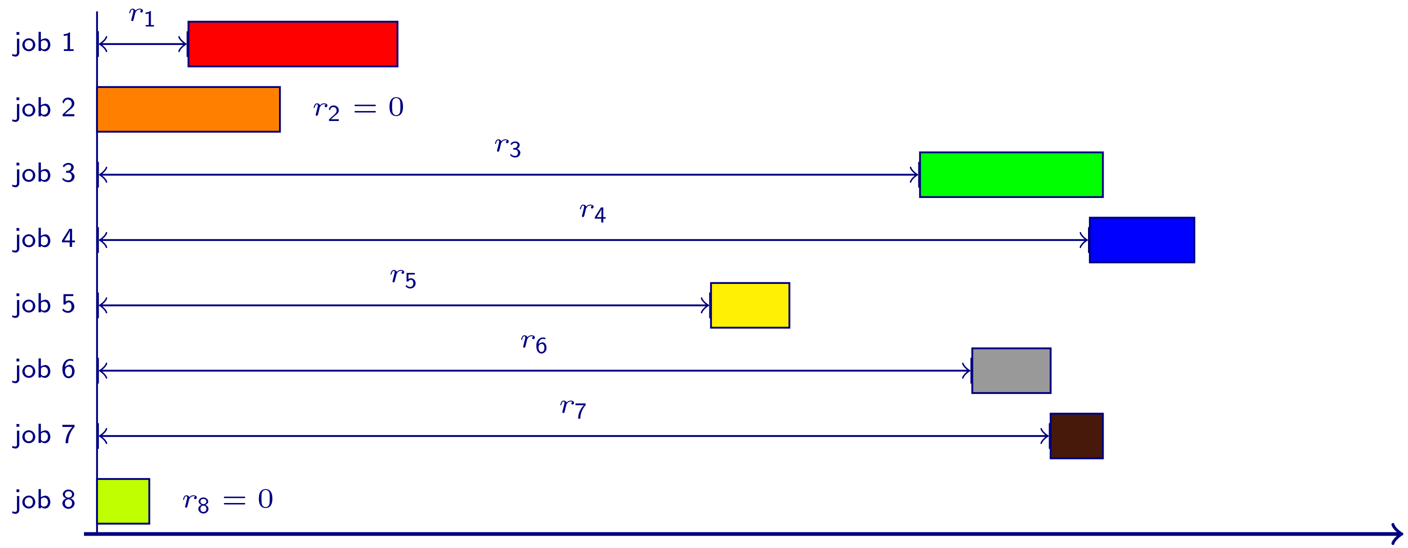
➡ Constraint: $s_j \geq r_j$



- Input now: jobs with release dates

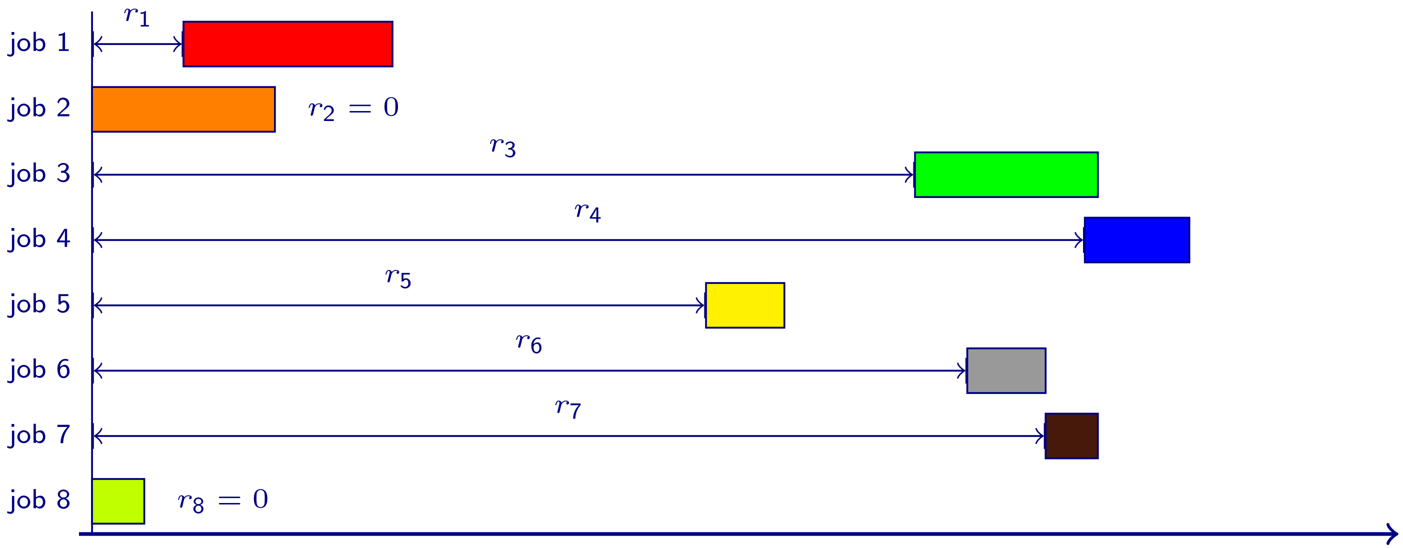


- Input now: jobs with release dates



- Minimize makespan

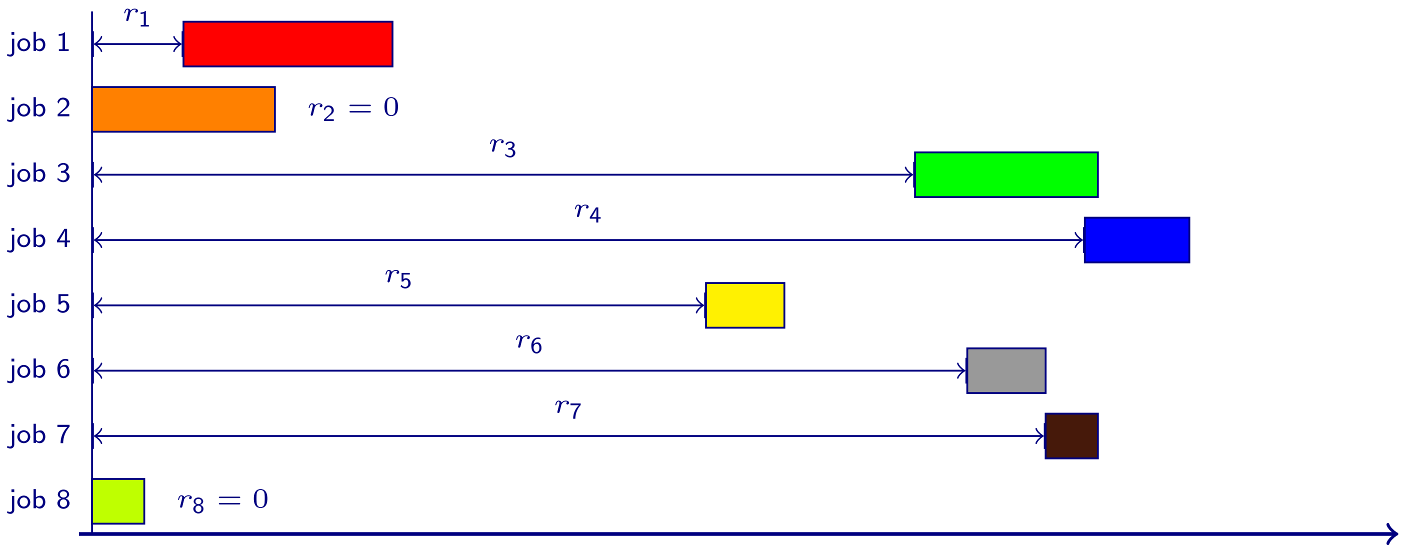
▷ Input now: jobs with release dates



▷ Minimize makespan

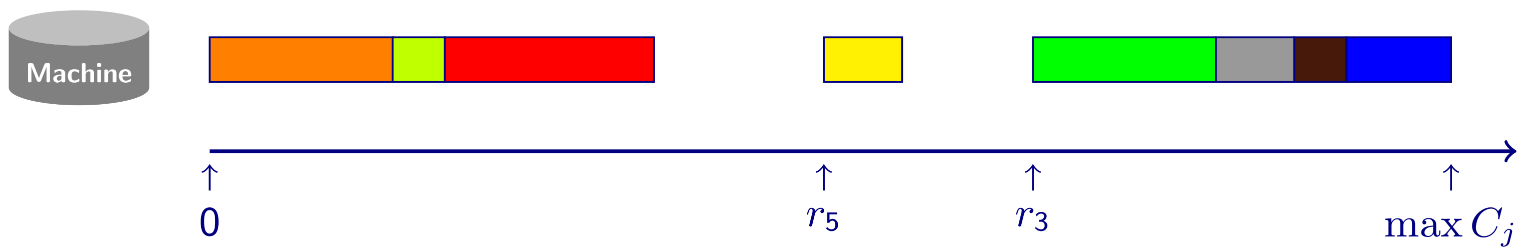
➡ Optimal algorithm: schedule jobs in the order of non-decreasing release dates

- ▷ Input now: jobs with release dates



- ▷ Minimize makespan

➡ Optimal algorithm: schedule jobs in the order of non-decreasing release dates



- ▷ Jobs can...
  - ...have weights (priorities)

- ▷ Jobs can...
  - ...have weights (priorities)
    - ➡ minimize weighted sum of completion times: same as in the unweighted case

▷ Jobs can...

- ...have weights (priorities)
  - ➡ minimize weighted sum of completion times: same as in the unweighted case
- ...have due dates (preferred latest completion time)

▷ Jobs can...

- ...have weights (priorities)
  - ➡ minimize weighted sum of completion times: same as in the unweighted case
- ...have due dates (preferred latest completion time)
  - ➡ minimize maximum lateness: schedule jobs in order of non-decreasing due dates

▷ Jobs can...

- ...have weights (priorities)
  - ➡ minimize weighted sum of completion times: same as in the unweighted case
- ...have due dates (preferred latest completion time)
  - ➡ minimize maximum lateness: schedule jobs in order of non-decreasing due dates
- ...have both due dates and release dates

▷ Jobs can...

- ...have weights (priorities)
  - ➡ minimize weighted sum of completion times: same as in the unweighted case
- ...have due dates (preferred latest completion time)
  - ➡ minimize maximum lateness: schedule jobs in order of non-decreasing due dates
- ...have both due dates and release dates
  - ➡ more complicated ($\mathcal{NP}$-hard)

▷ Jobs can...

- ...have weights (priorities)
  - ➡ minimize weighted sum of completion times: same as in the unweighted case
- ...have due dates (preferred latest completion time)
  - ➡ minimize maximum lateness: schedule jobs in order of non-decreasing due dates
- ...have both due dates and release dates
  - ➡ more complicated ($\mathcal{NP}$-hard)
- ...be allowed to be interrupted (possibly at additional cost/time)

▷ Jobs can...

- ...have weights (priorities)
  - ➡ minimize weighted sum of completion times: same as in the unweighted case
- ...have due dates (preferred latest completion time)
  - ➡ minimize maximum lateness: schedule jobs in order of non-decreasing due dates
- ...have both due dates and release dates
  - ➡ more complicated ($\mathcal{NP}$-hard)
- ...be allowed to be interrupted (possibly at additional cost/time)
  - ➡ easier if no cost/time involved, harder otherwise

▷ Jobs can...

- ...have weights (priorities)
  - ➡ minimize weighted sum of completion times: same as in the unweighted case
- ...have due dates (preferred latest completion time)
  - ➡ minimize maximum lateness: schedule jobs in order of non-decreasing due dates
- ...have both due dates and release dates
  - ➡ more complicated ($\mathcal{NP}$-hard)
- ...be allowed to be interrupted (possibly at additional cost/time)
  - ➡ easier if no cost/time involved, harder otherwise
- ...consume resources

More variants

▷ Jobs can...

- ...have weights (priorities)
  - ➡ minimize weighted sum of completion times: same as in the unweighted case
- ...have due dates (preferred latest completion time)
  - ➡ minimize maximum lateness: schedule jobs in order of non-decreasing due dates
- ...have both due dates and release dates
  - ➡ more complicated ($\mathcal{NP}$-hard)
- ...be allowed to be interrupted (possibly at additional cost/time)
  - ➡ easier if no cost/time involved, harder otherwise
- ...consume resources
  - ➡ Resource-constrained scheduling

- Single Machine Scheduling: only one machine available

- ▷ Single Machine Scheduling: only one machine available
- ➡ Minimal latest completion time is constant

- ▷ Single Machine Scheduling: only one machine available
- ➡ Minimal latest completion time is constant
- ➡ With no release dates, greedy strategy gives an optimal solution

- ▷ Single Machine Scheduling: only one machine available
- ➡ Minimal latest completion time is constant
- ➡ With no release dates, greedy strategy gives an optimal solution
- ➡ With release dates, greedy strategy only works for makespan minimization

- ▷ Single Machine Scheduling: only one machine available
- ➡ Minimal latest completion time is constant
- ➡ With no release dates, greedy strategy gives an optimal solution
- ➡ With release dates, greedy strategy only works for makespan minimization

- ▷ Summary if no interruptions and resources are involved:

	Objective		
	$\sum C_j$	$\max C_j$	lateness
no release dates	non-decreasing process times	trivial	non-decreasing due dates
with release dates	$\mathcal{NP}$-hard	non-decreasing release dates	$\mathcal{NP}$-hard

▷ Example:

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

▷ Example:

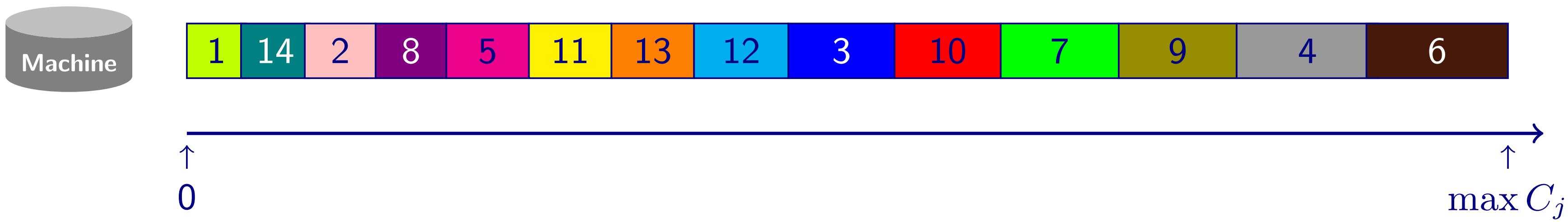
job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

➡ Order by non-decreasing process times:

▷ Example:

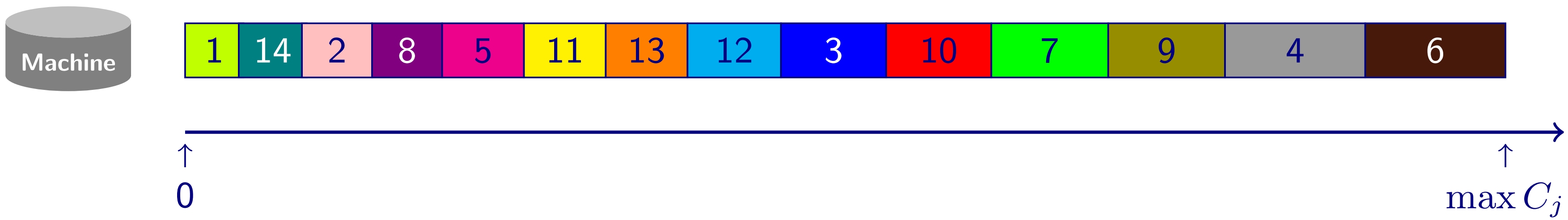
job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

➡ Order by non-decreasing process times:



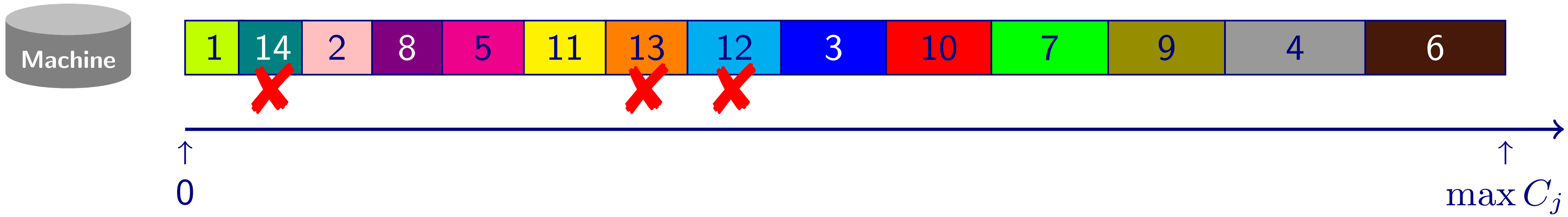
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



▷ Example: with precedence constraints ➡ Project scheduling

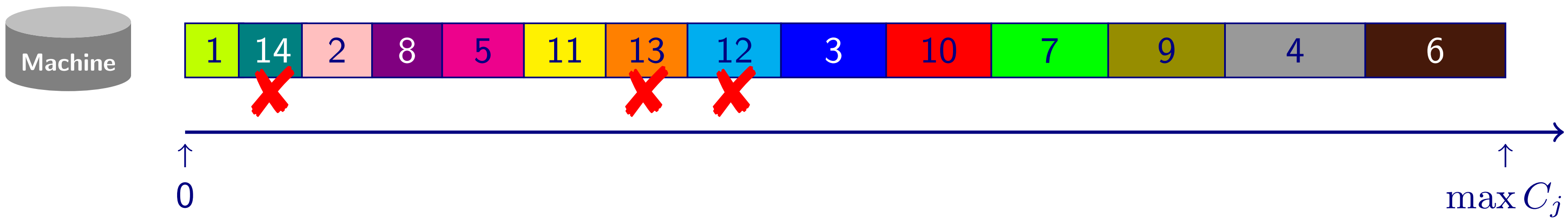
job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



Jobs with precedence constraints

▷ Example: with precedence constraints ➡ Project scheduling

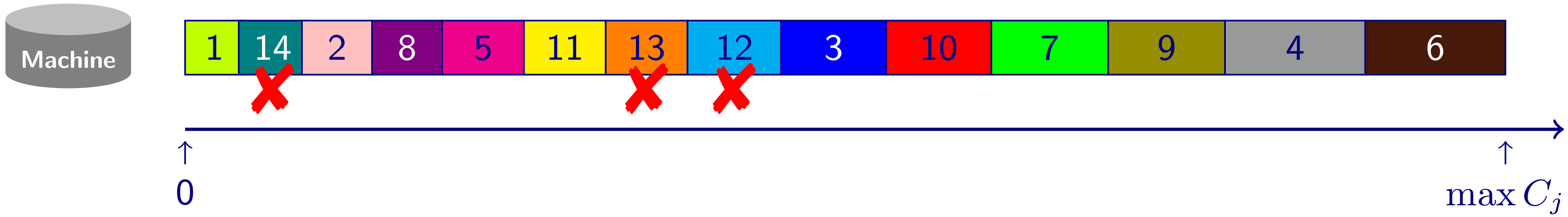
job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



➡ Schedule is infeasible!

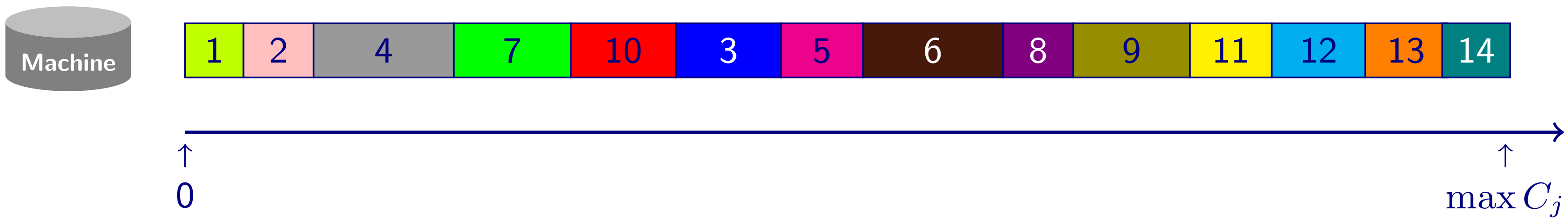
▷ Example: with **precedence constraints** ➡ **Project scheduling**

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



➡ Schedule is infeasible!

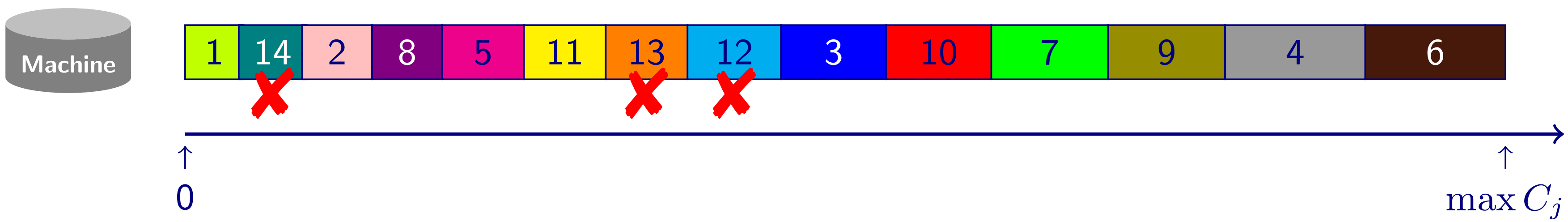
▷ Feasible schedule:



Jobs with precedence constraints

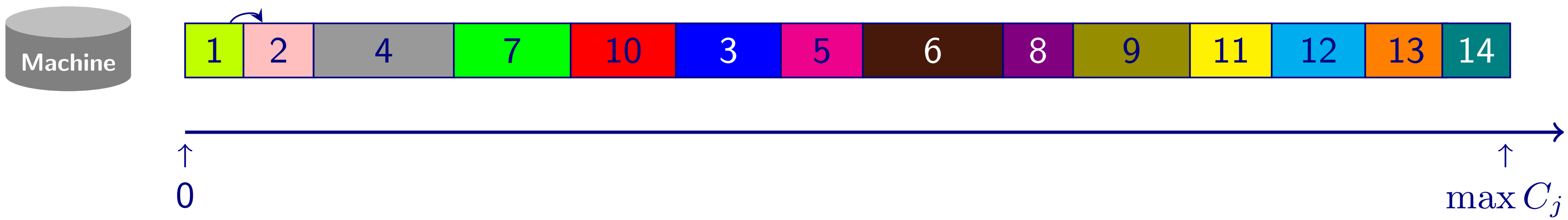
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



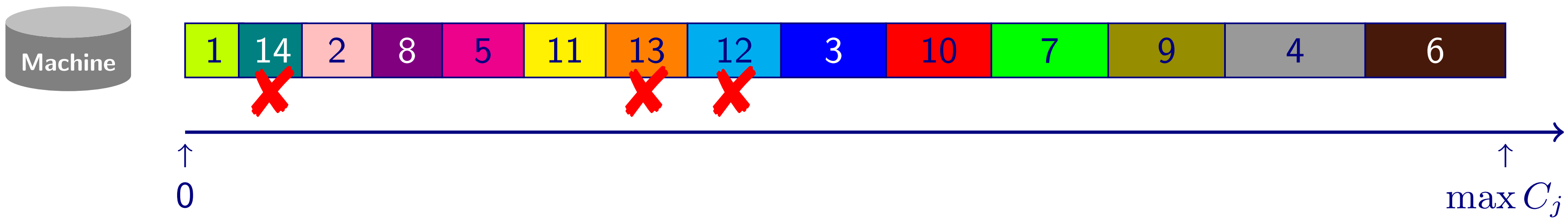
➡ Schedule is infeasible!

▷ Feasible schedule:



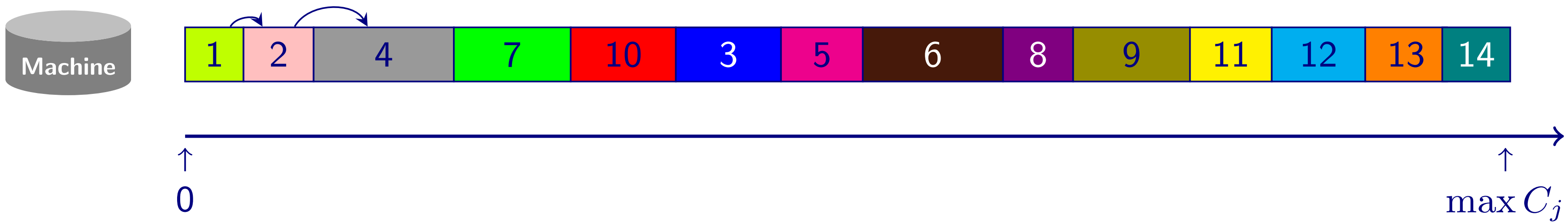
- Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



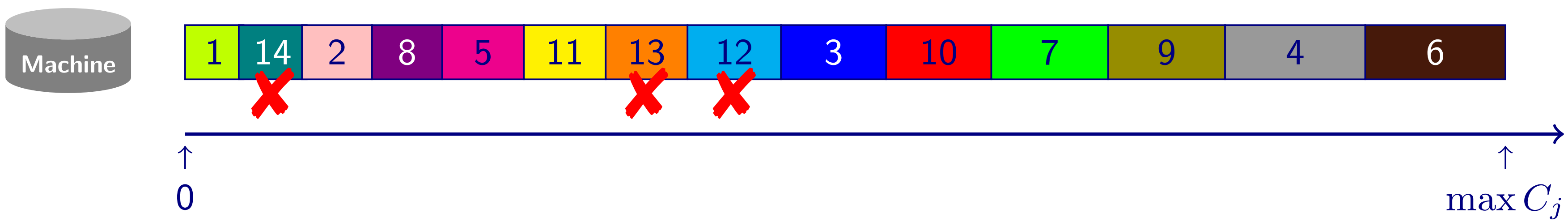
➡ Schedule is infeasible!

- Feasible schedule:



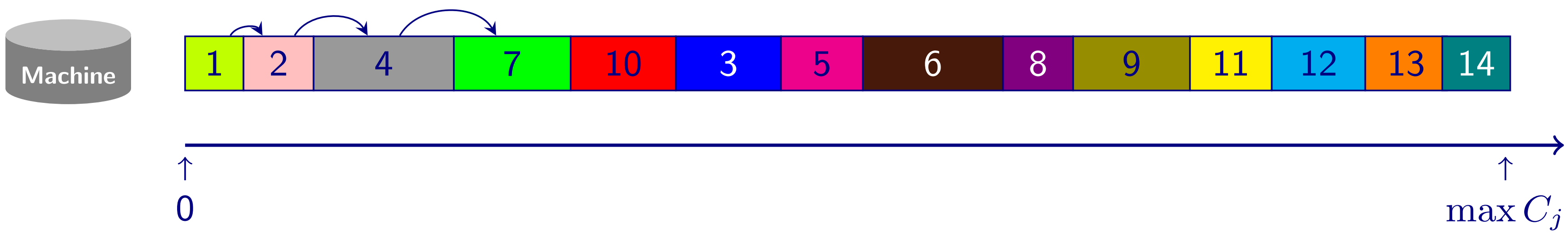
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



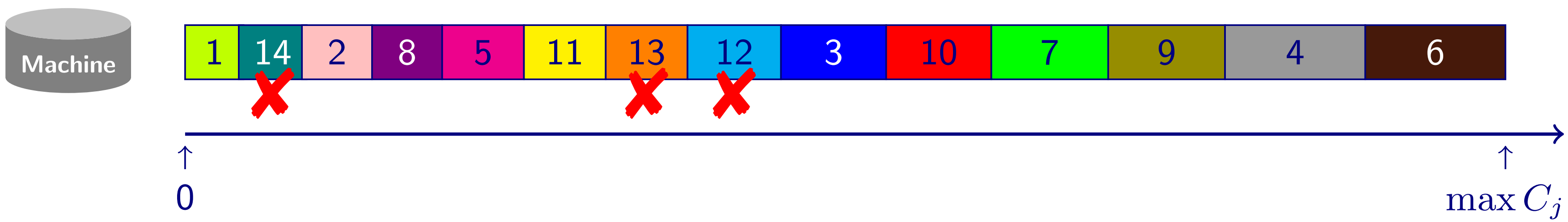
➡ Schedule is infeasible!

▷ Feasible schedule:



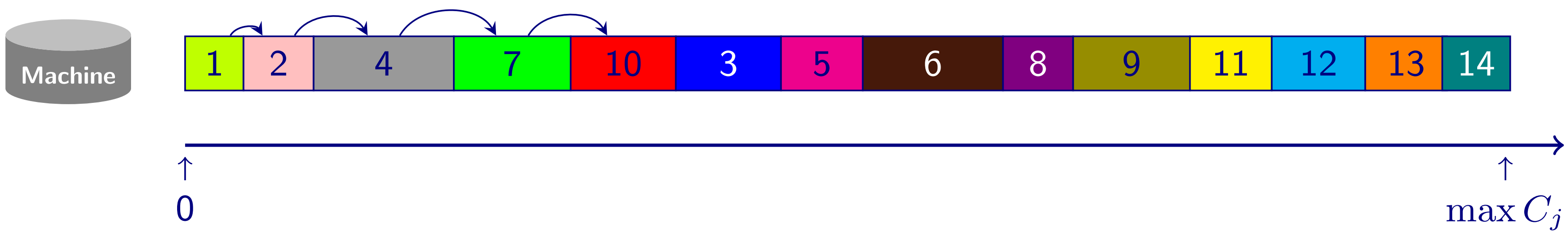
▷ Example: with **precedence constraints** ➡ **Project scheduling**

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



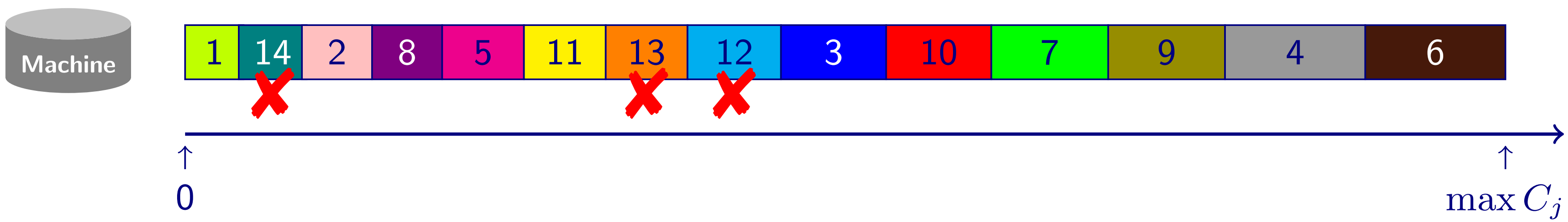
➡ Schedule is infeasible!

▷ Feasible schedule:



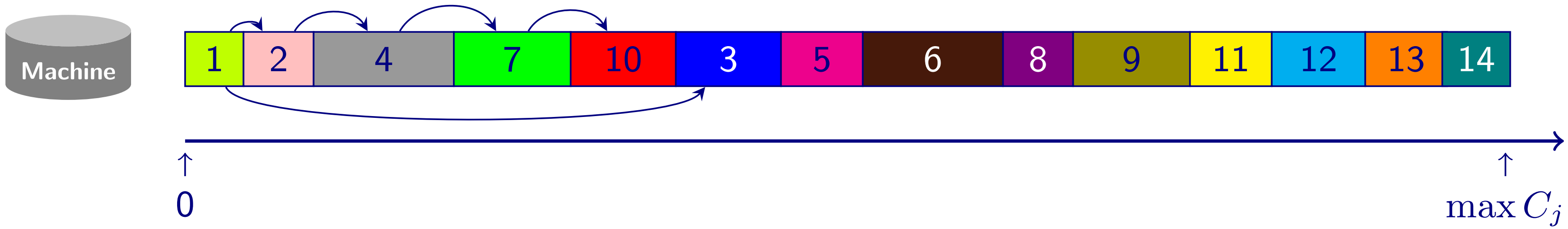
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



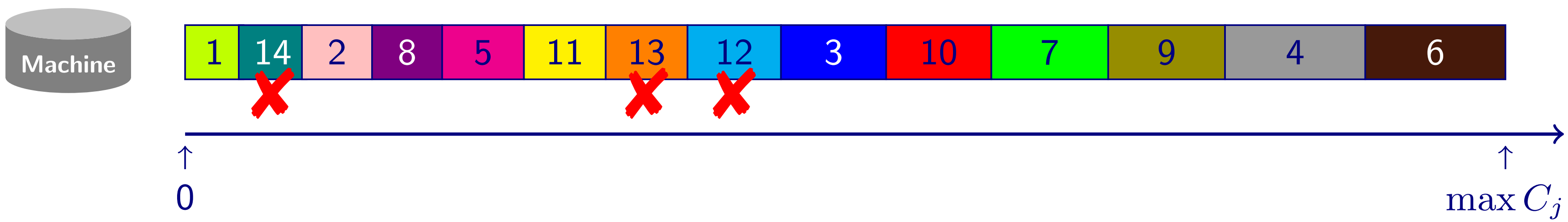
➡ Schedule is infeasible!

▷ Feasible schedule:



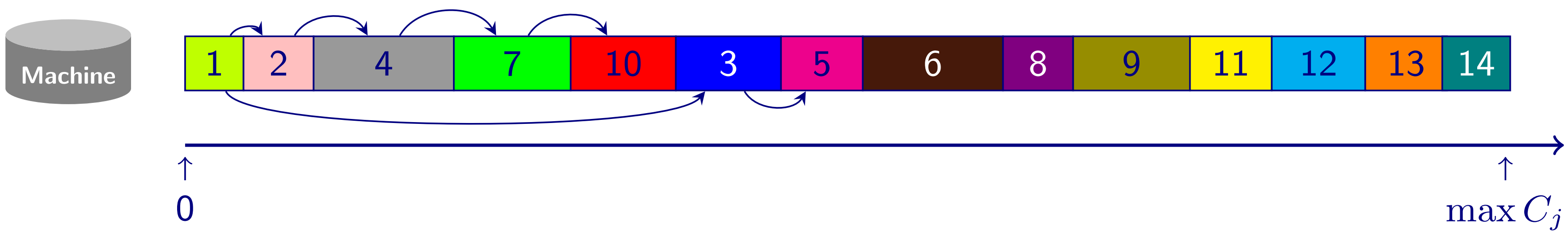
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



➡ Schedule is infeasible!

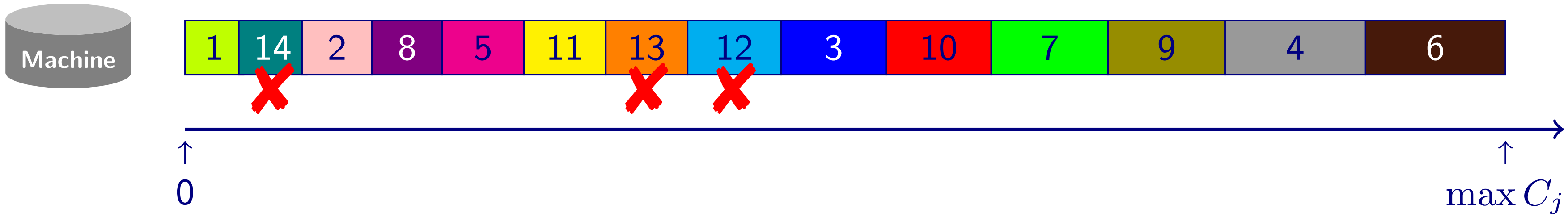
▷ Feasible schedule:



Jobs with precedence constraints

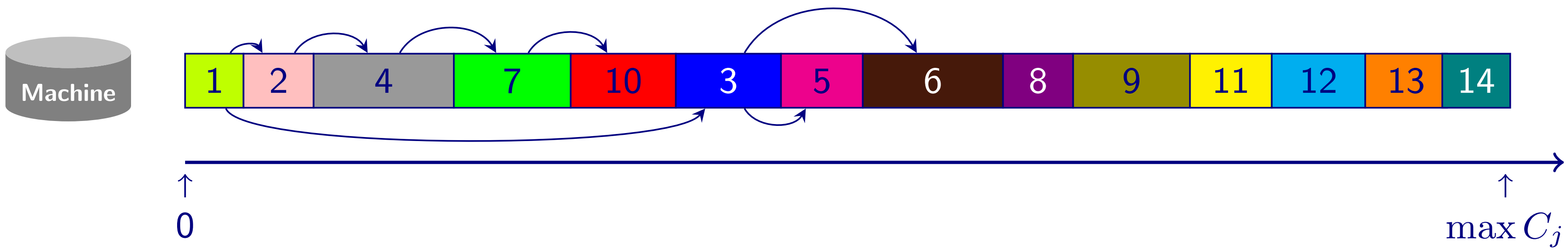
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



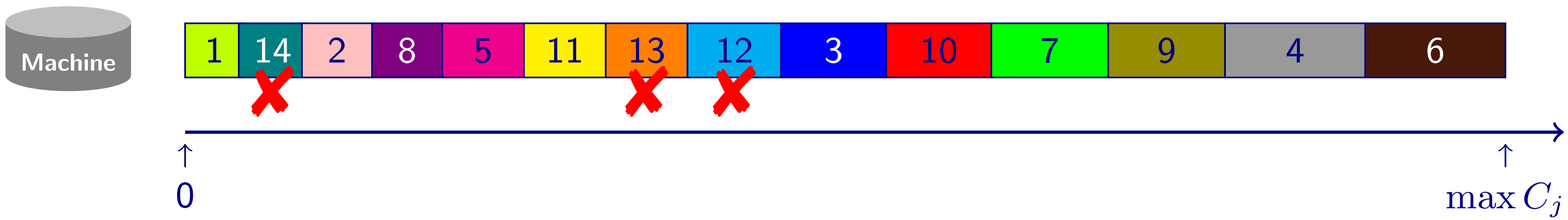
➡ Schedule is infeasible!

▷ Feasible schedule:



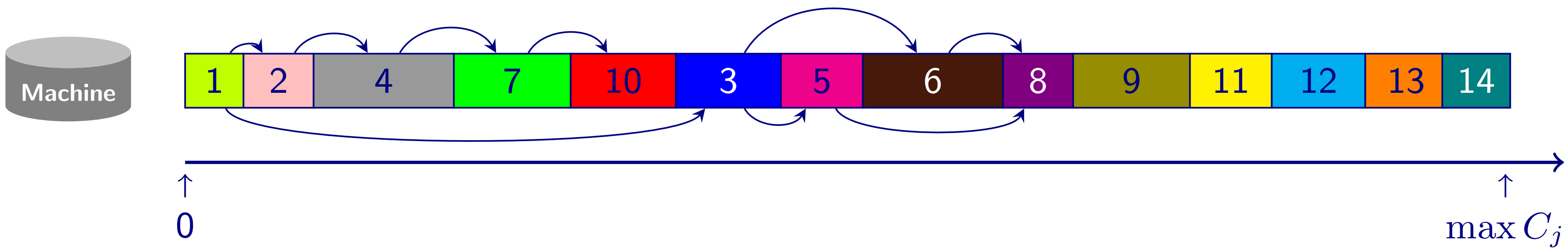
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



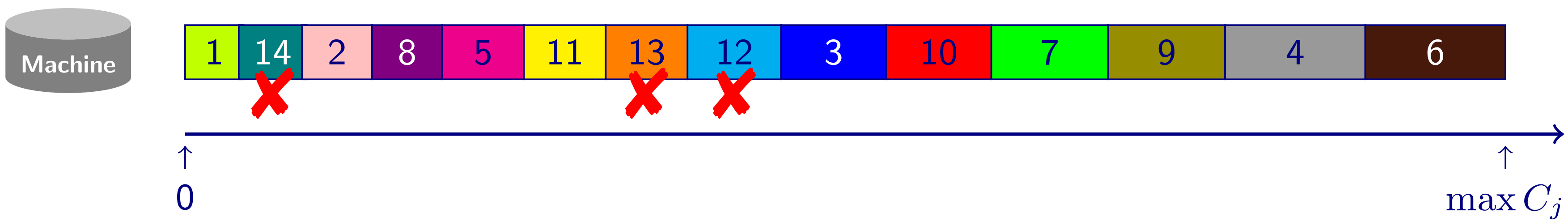
➡ Schedule is infeasible!

▷ Feasible schedule:



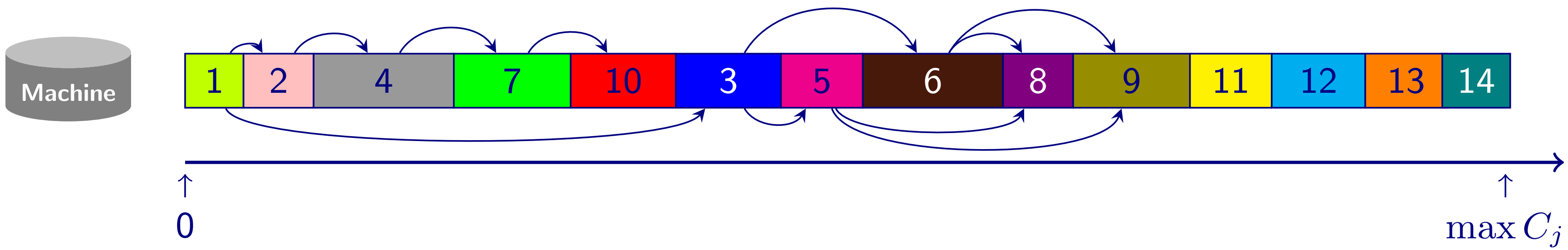
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



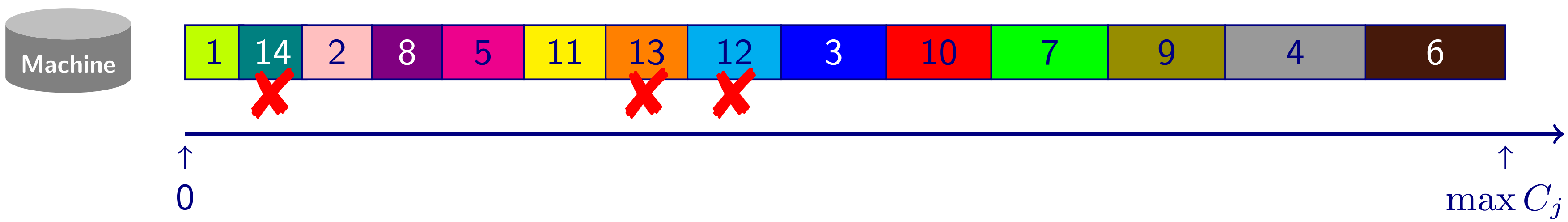
➡ Schedule is infeasible!

▷ Feasible schedule:



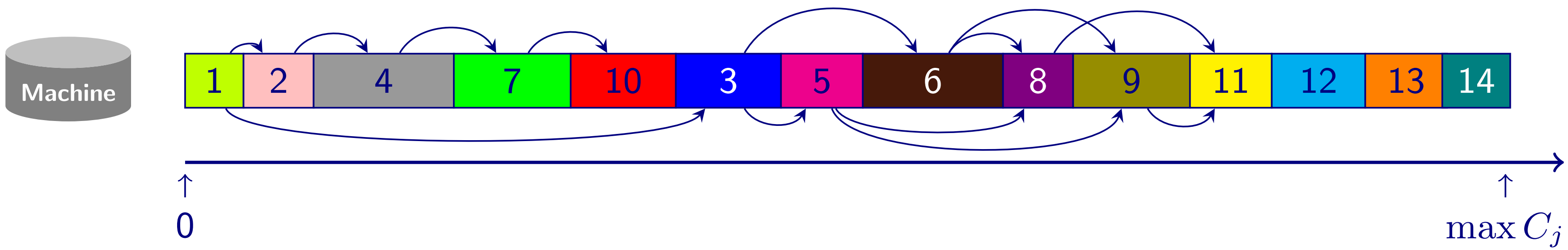
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



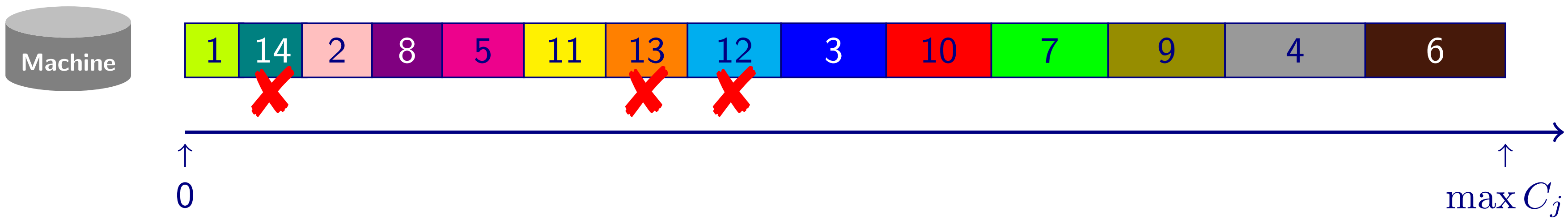
➡ Schedule is infeasible!

▷ Feasible schedule:



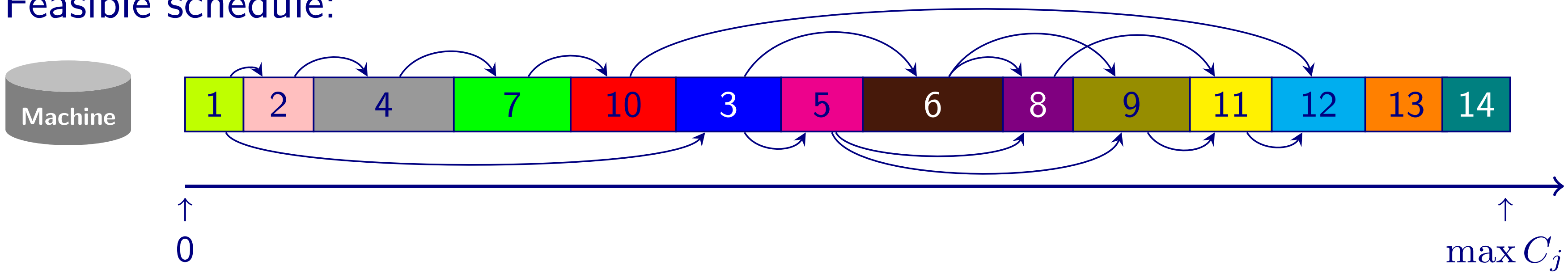
- Example: with **precedence constraints** ➡ **Project scheduling**

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



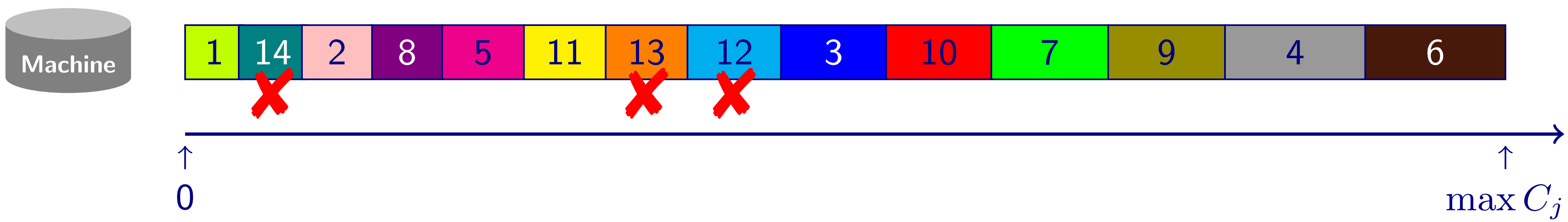
➡ Schedule is infeasible!

- Feasible schedule:



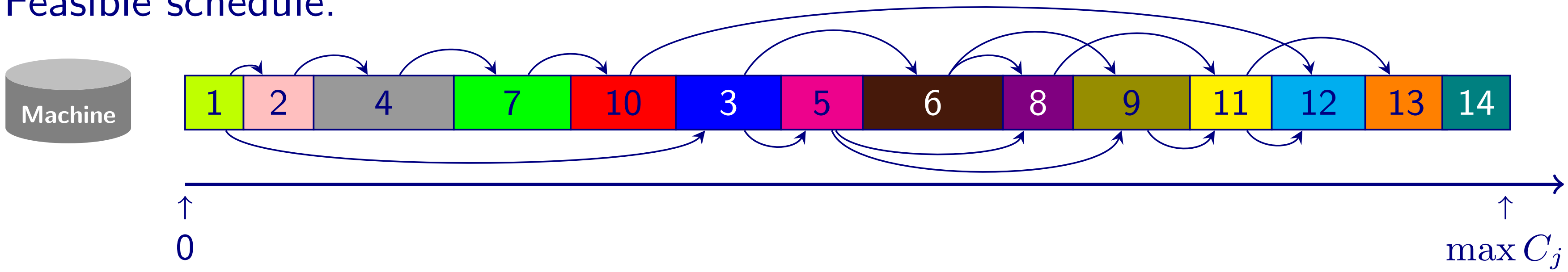
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



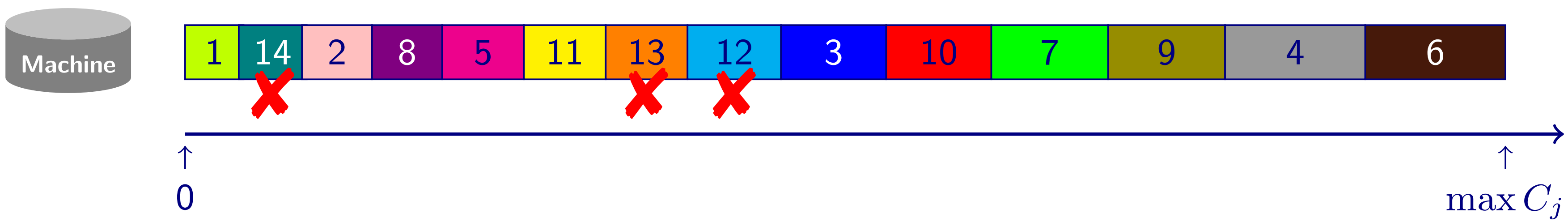
➡ Schedule is infeasible!

▷ Feasible schedule:



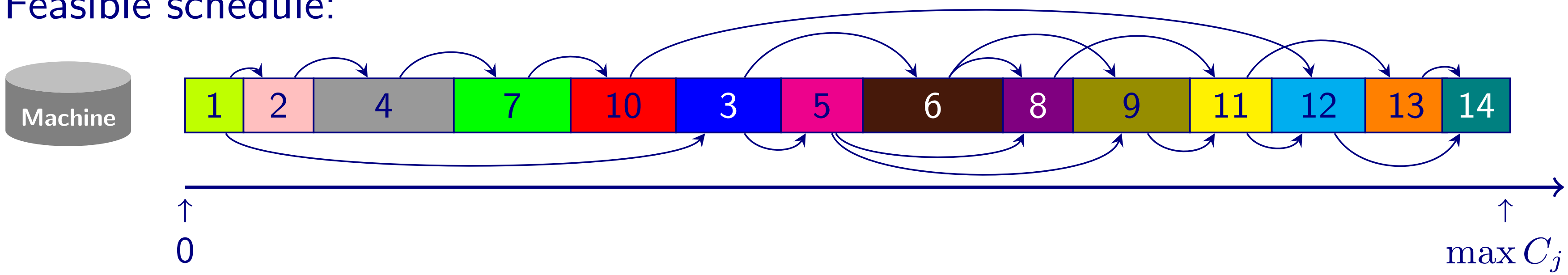
▷ Example: with precedence constraints ➡ Project scheduling

job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



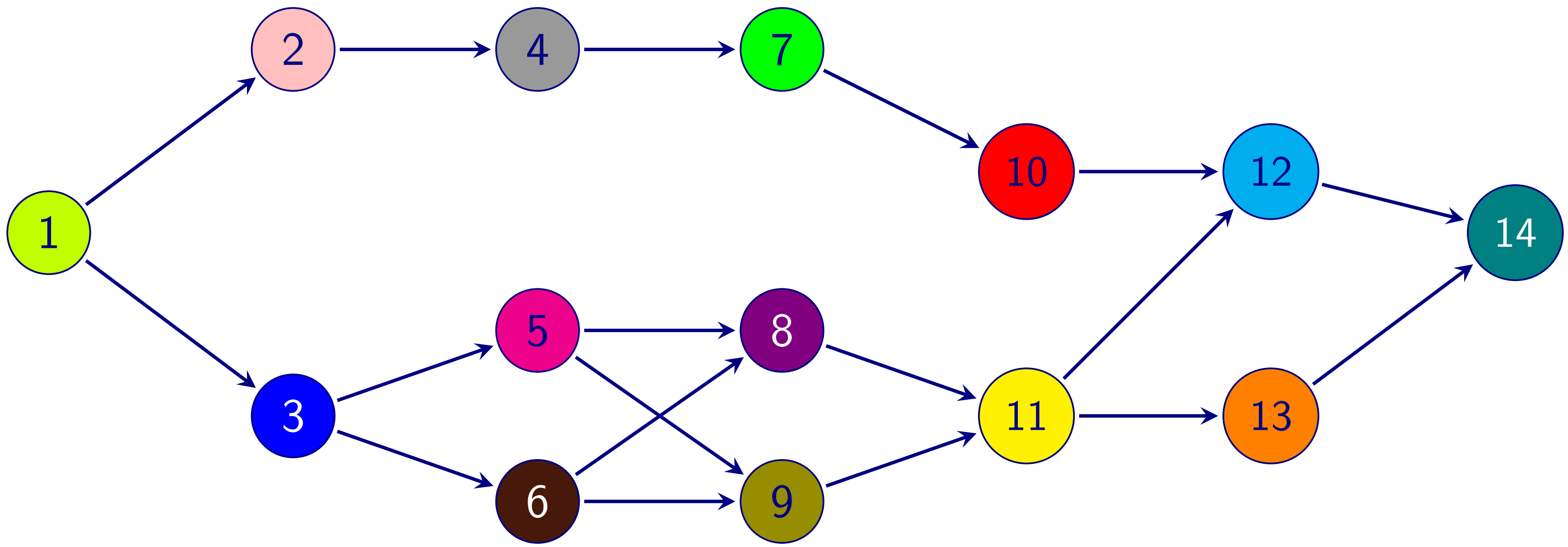
➡ Schedule is infeasible!

▷ Feasible schedule:

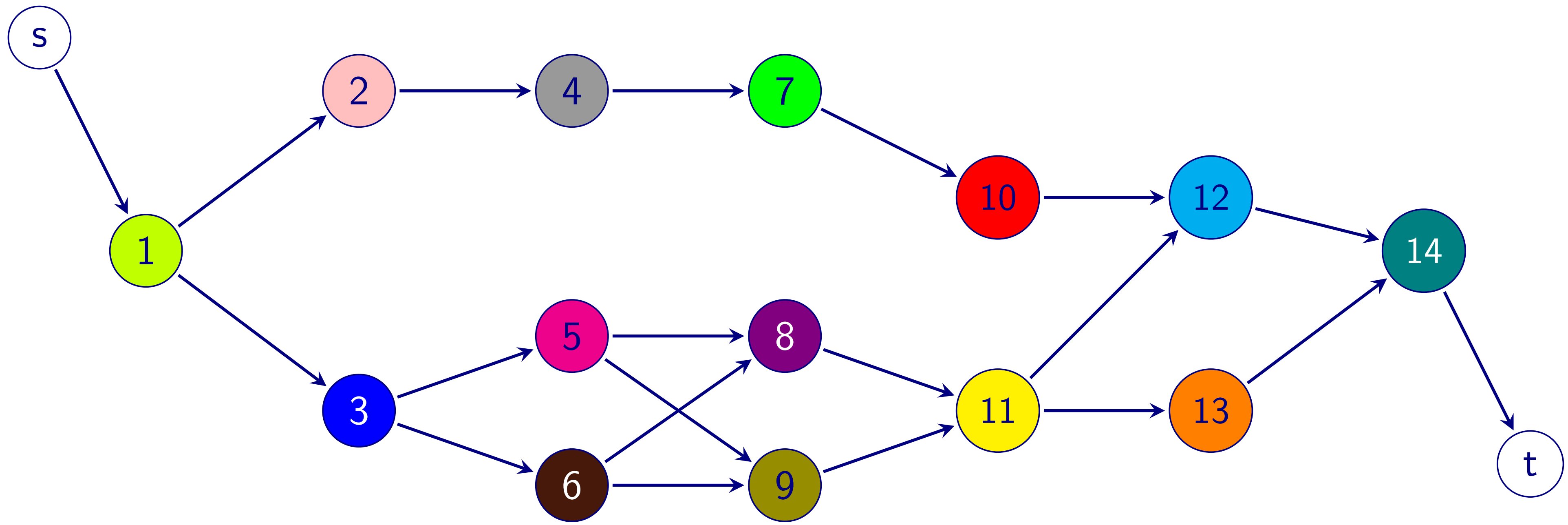


job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13

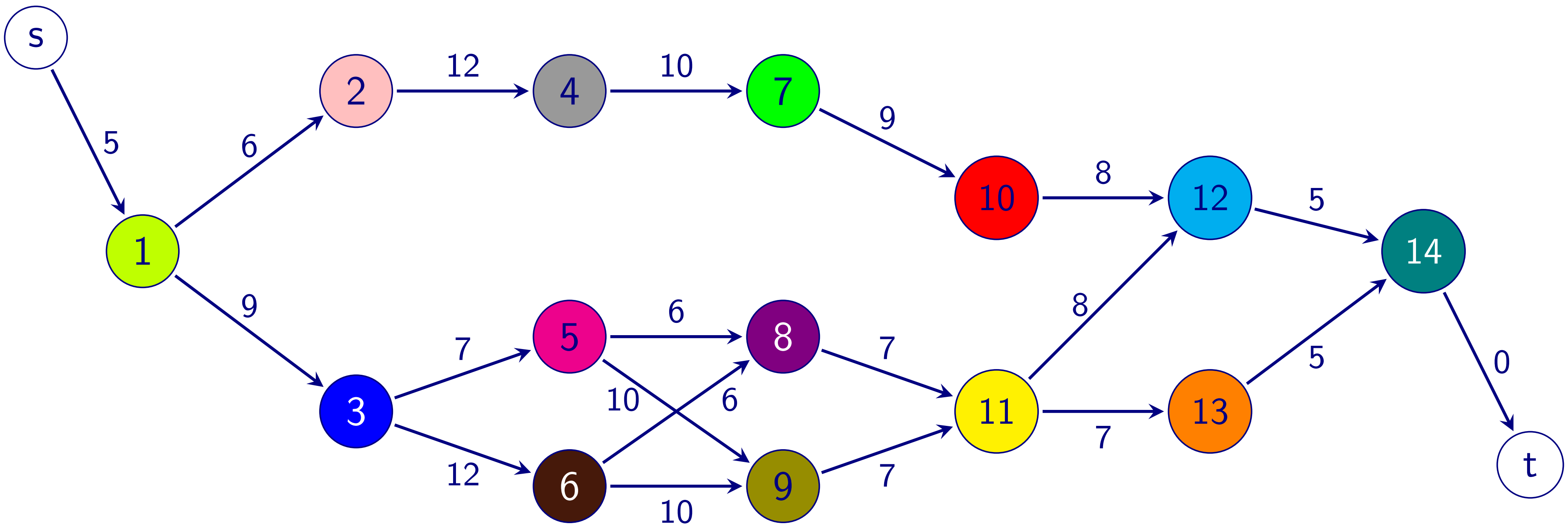
job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



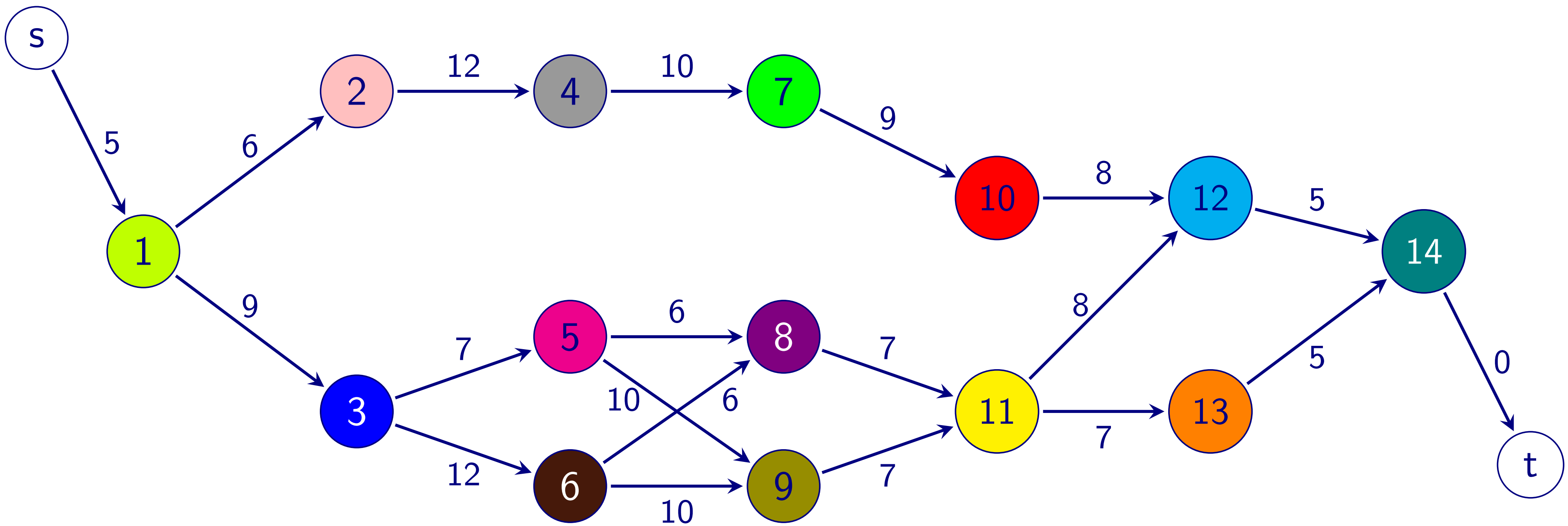
job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



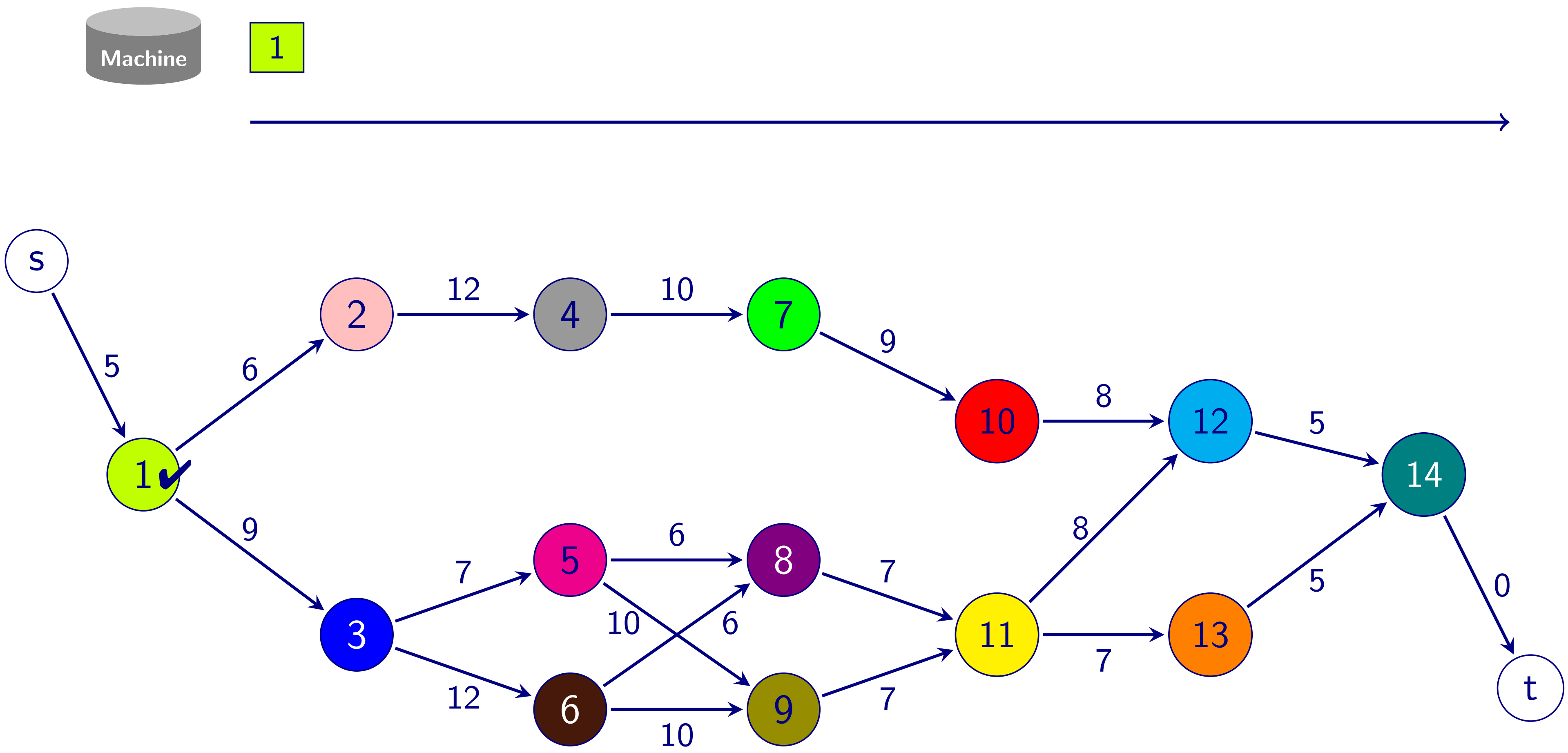
job j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
process time p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5
preceded by	–	1	1	2	3	3	4	5,6	5,6	7	8,9	10,11	11	12,13



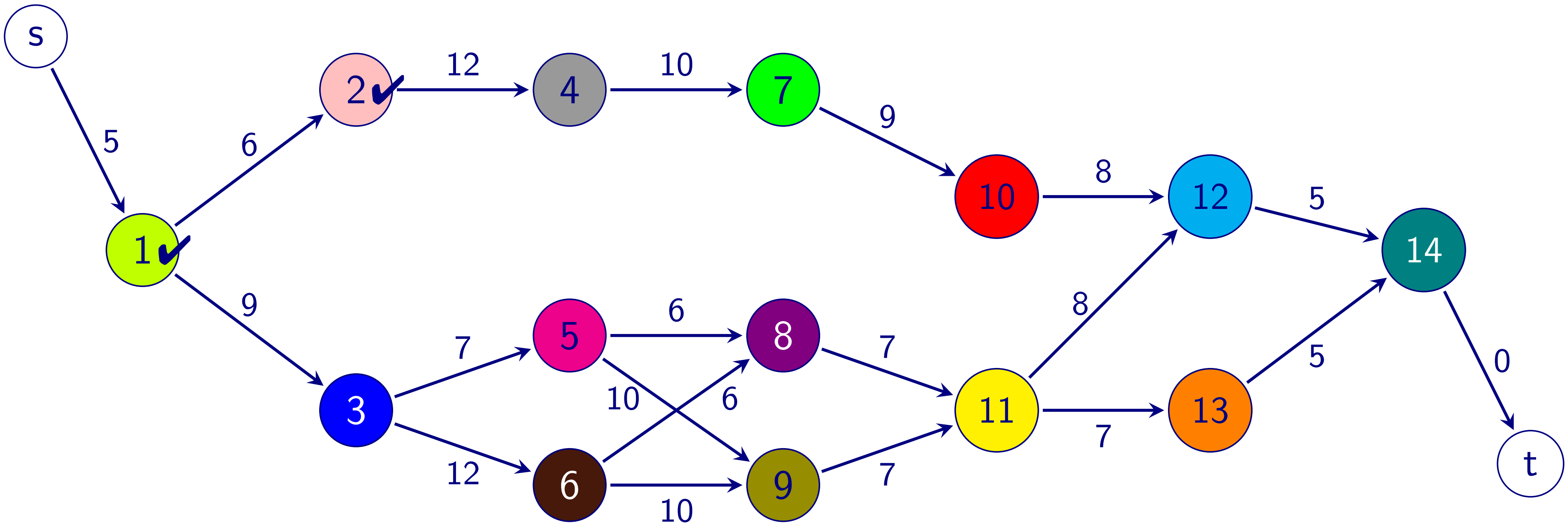
- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences



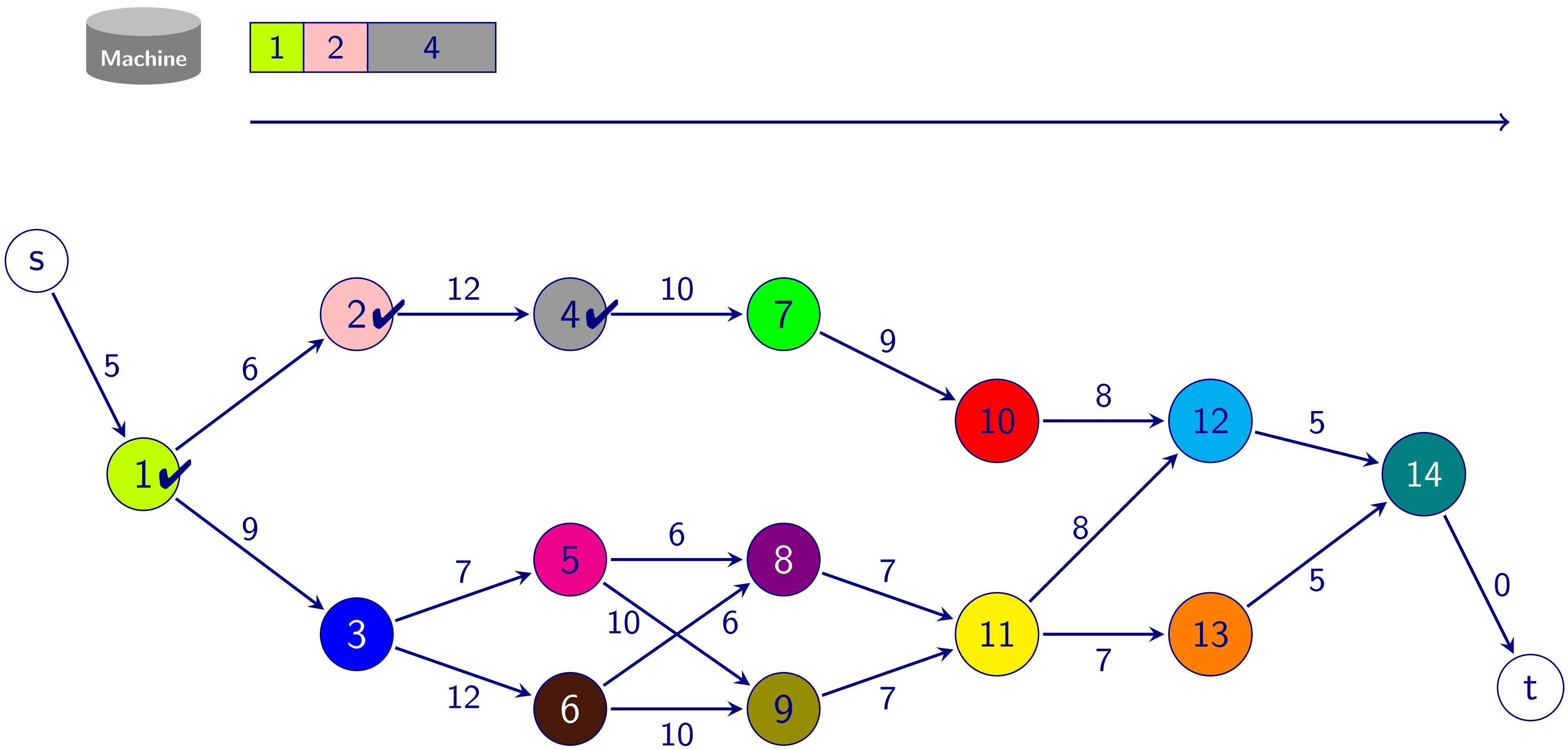
- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences



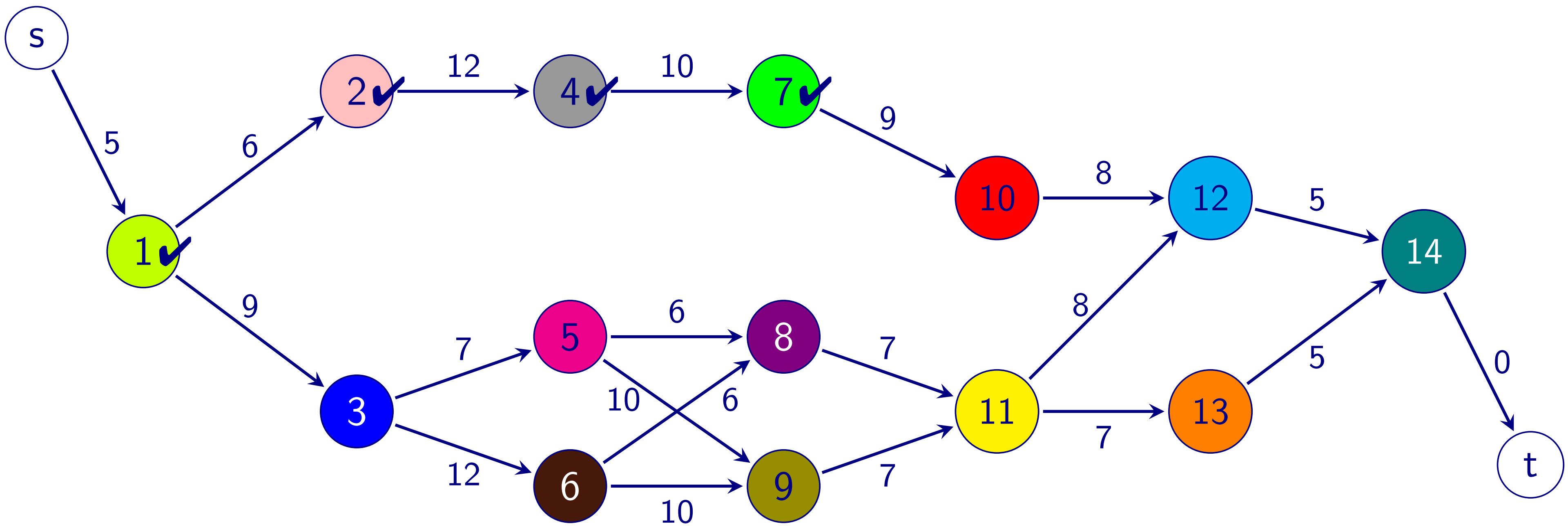
- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences



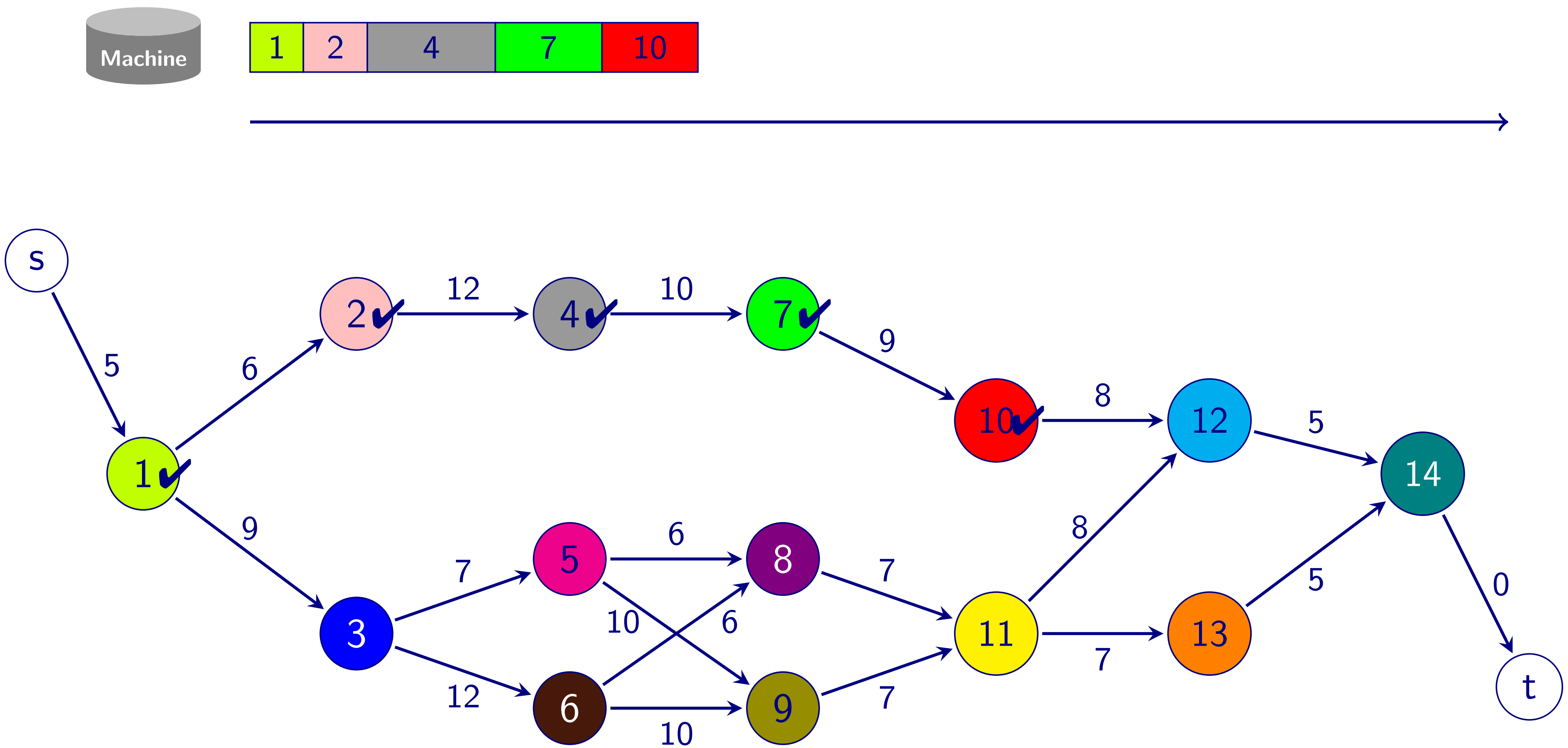
- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences



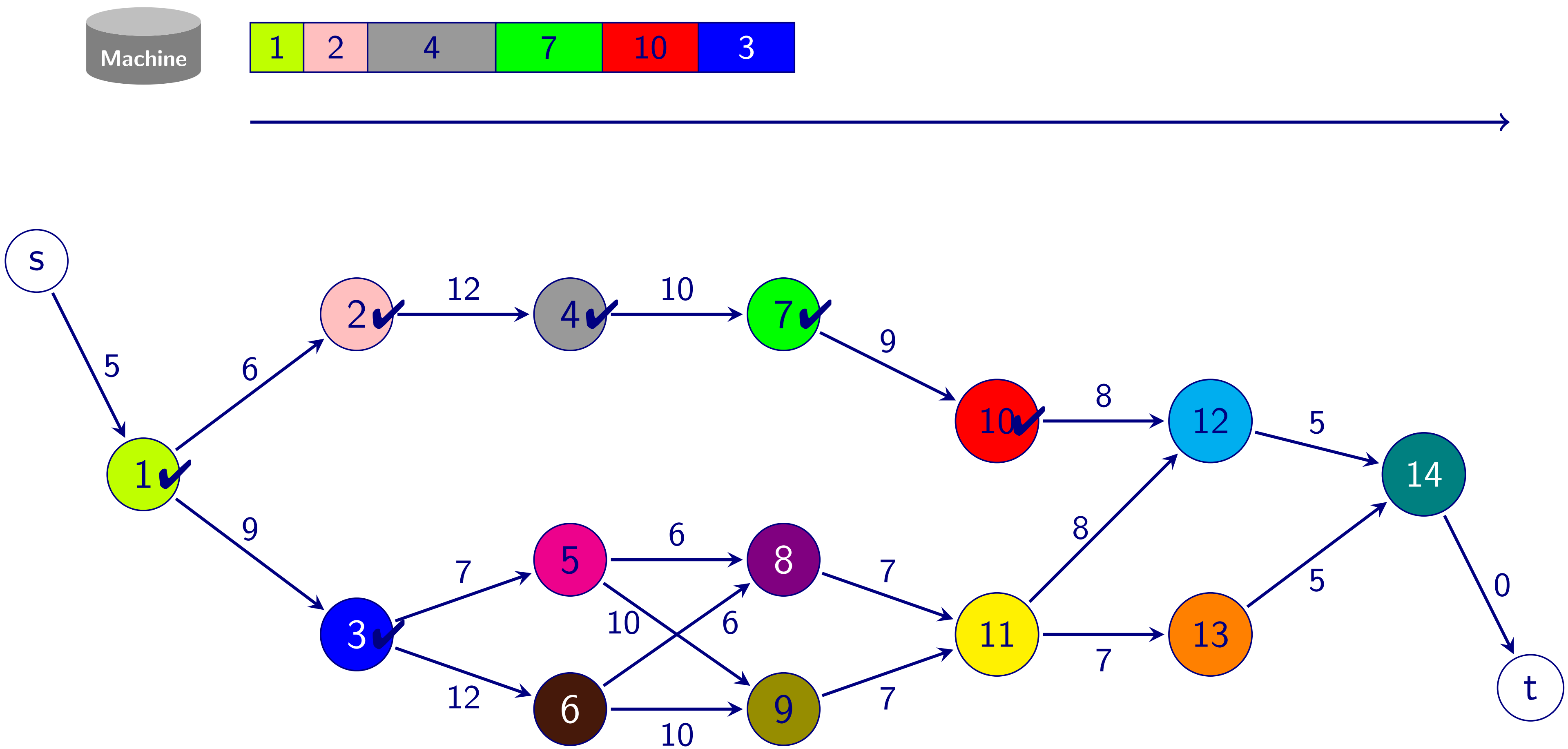
- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences



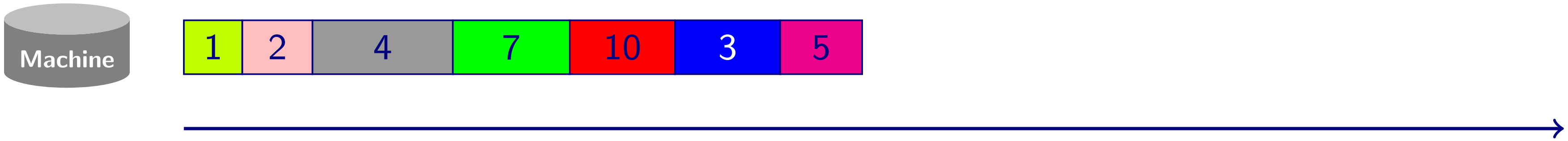
- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences

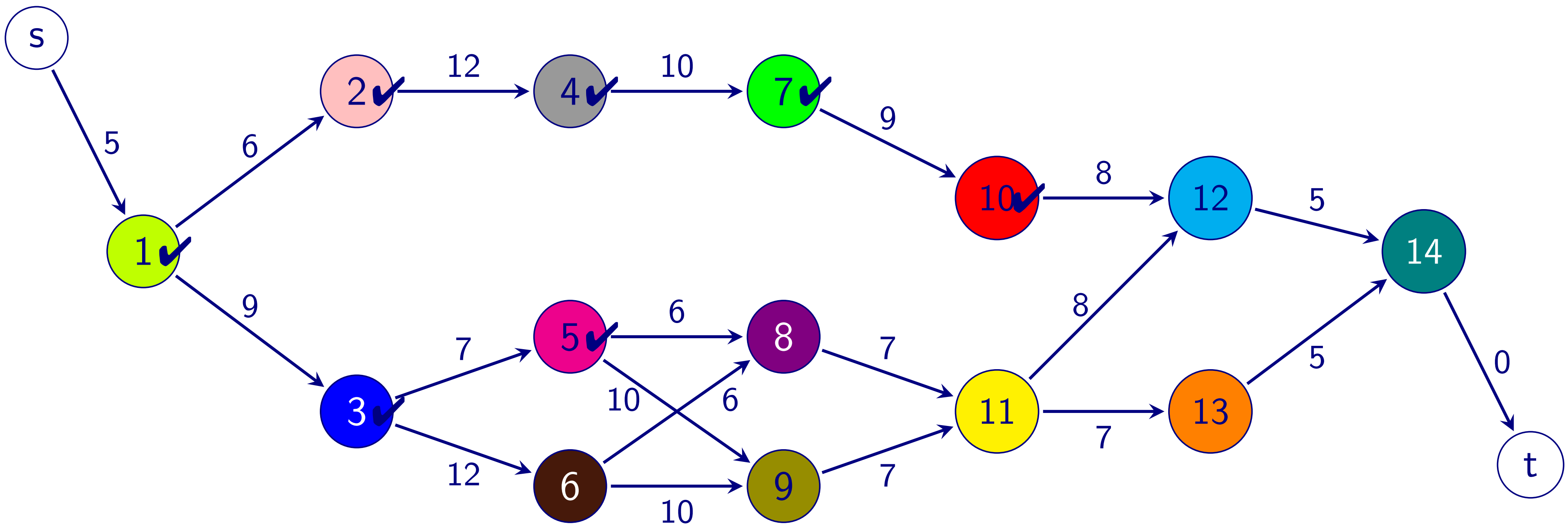


▷ Greedy strategy: schedule an arbitrary job next with already fulfilled precedences

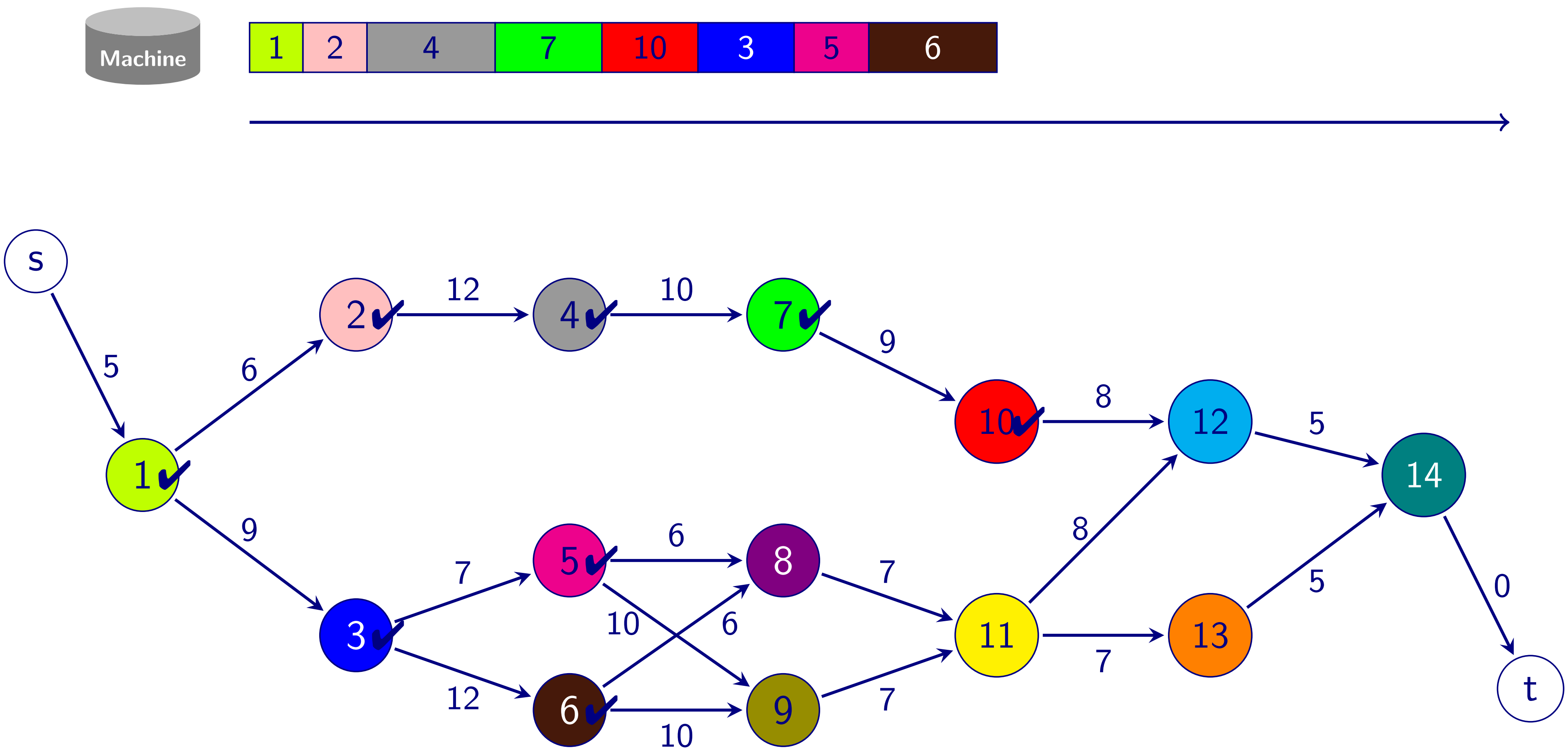


- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences

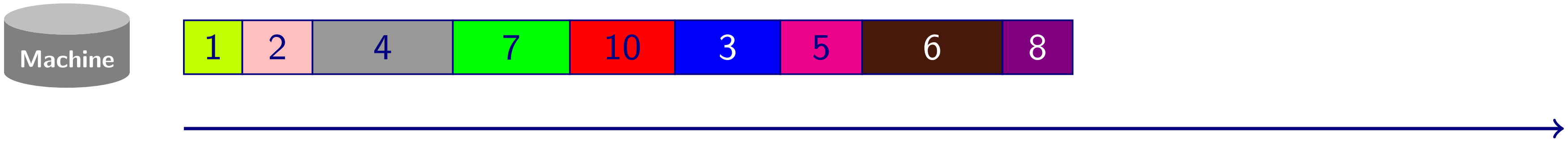


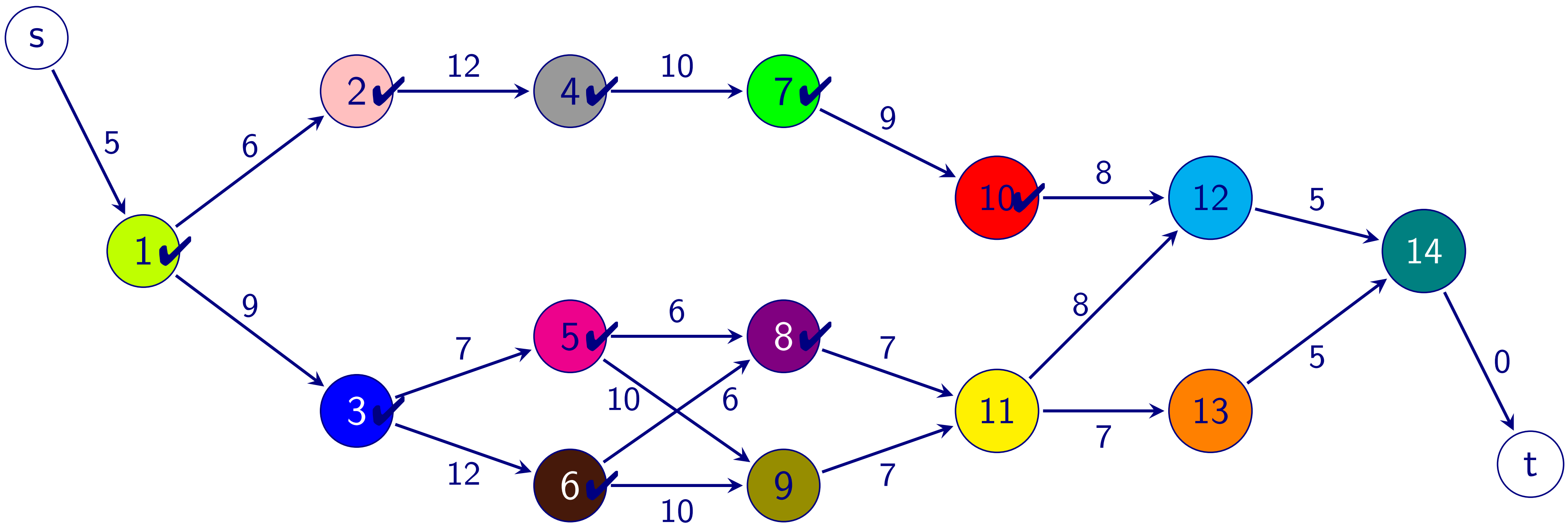


- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences

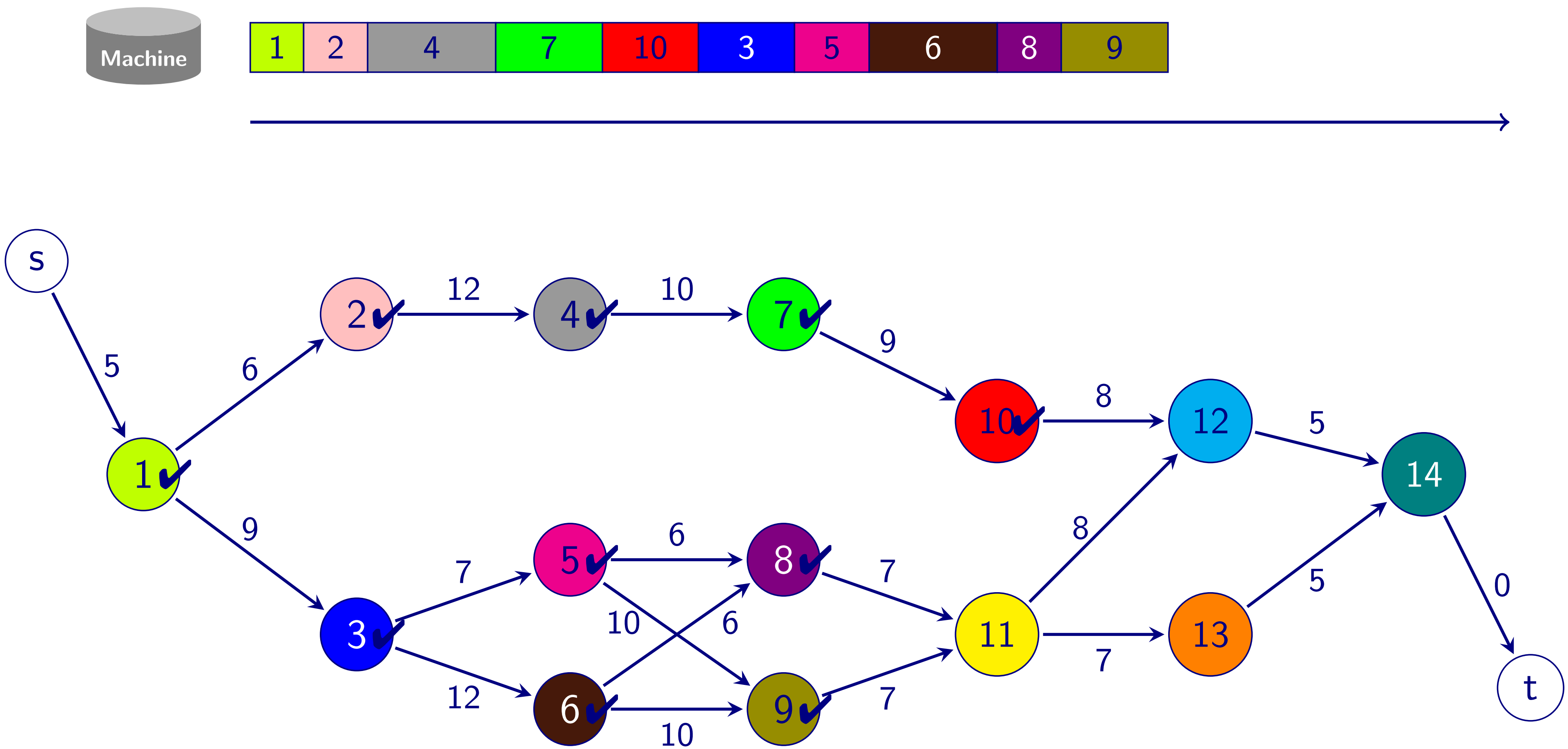


▷ Greedy strategy: schedule an arbitrary job next with already fulfilled precedences

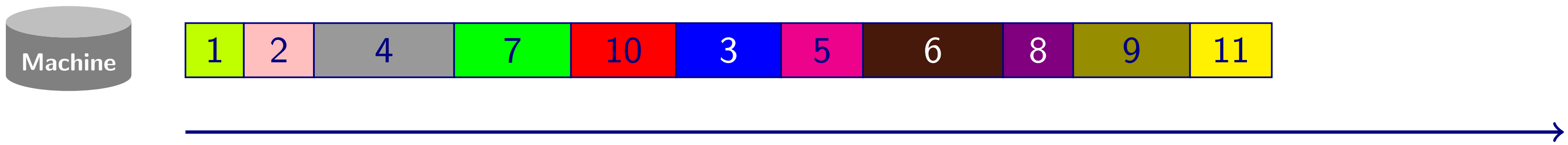


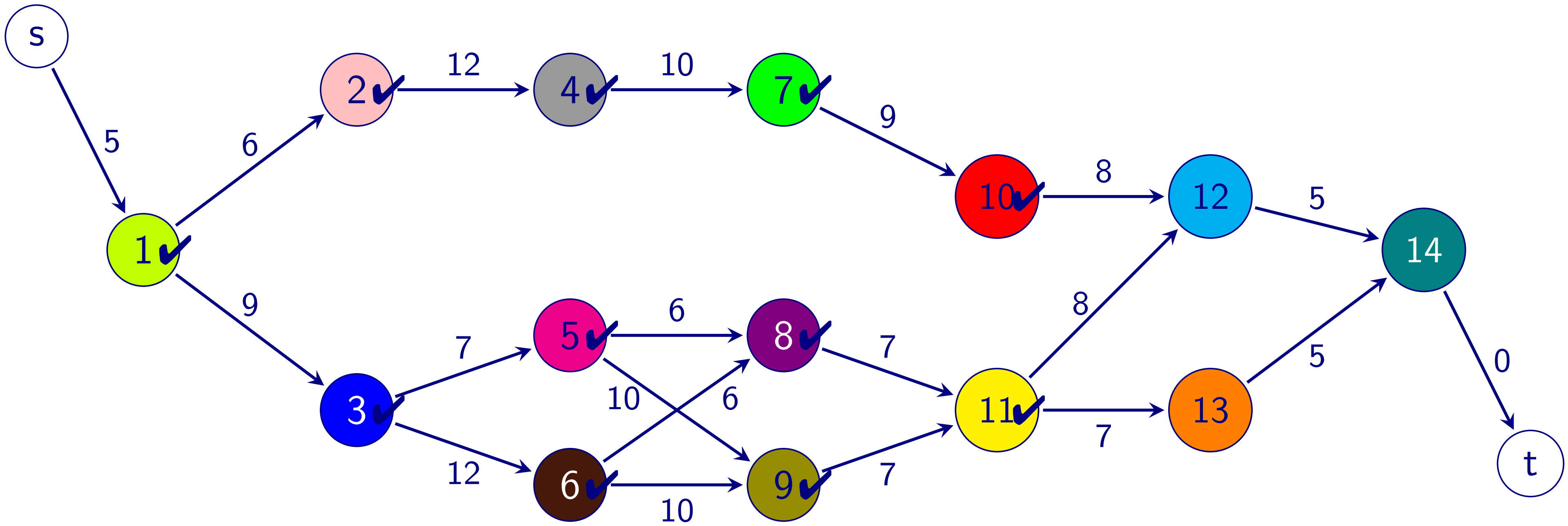


- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences

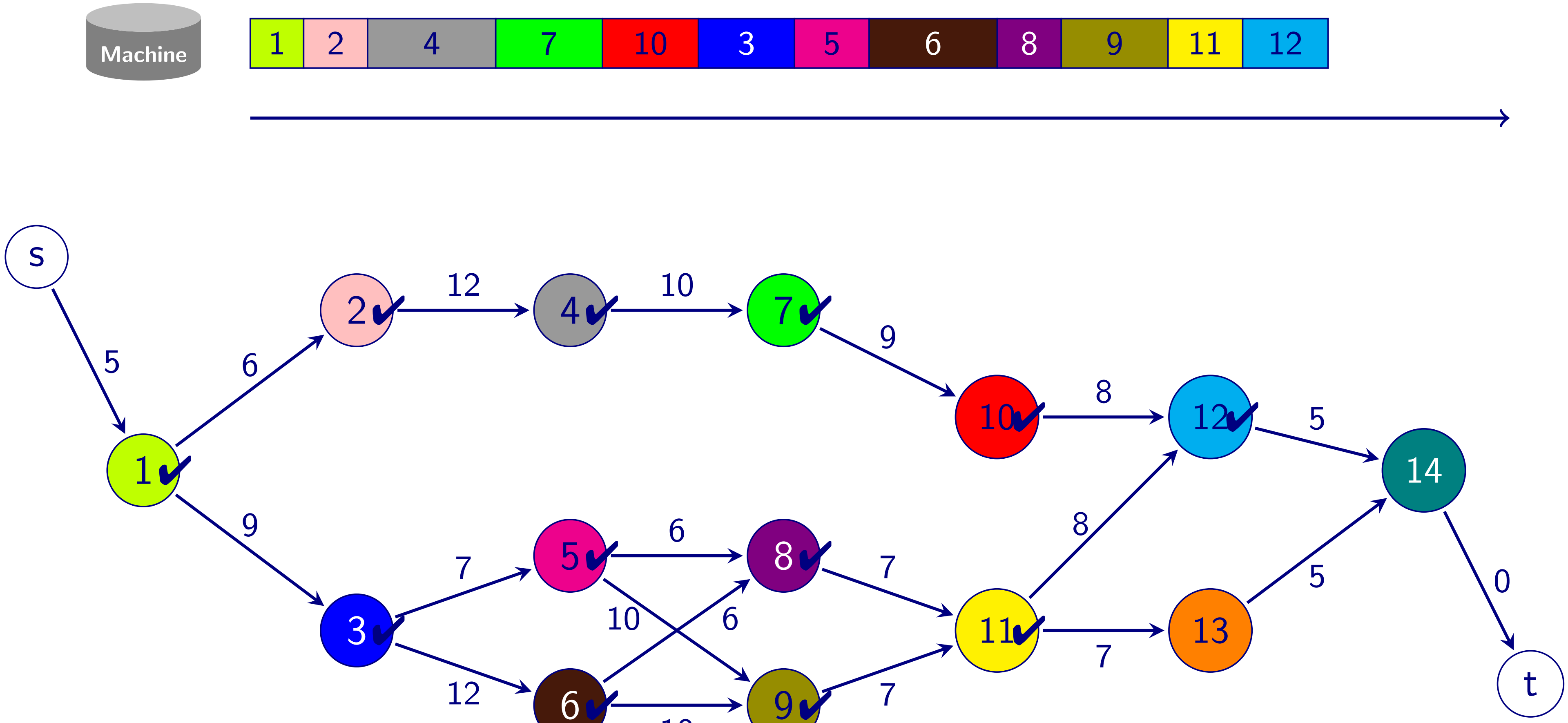


- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences

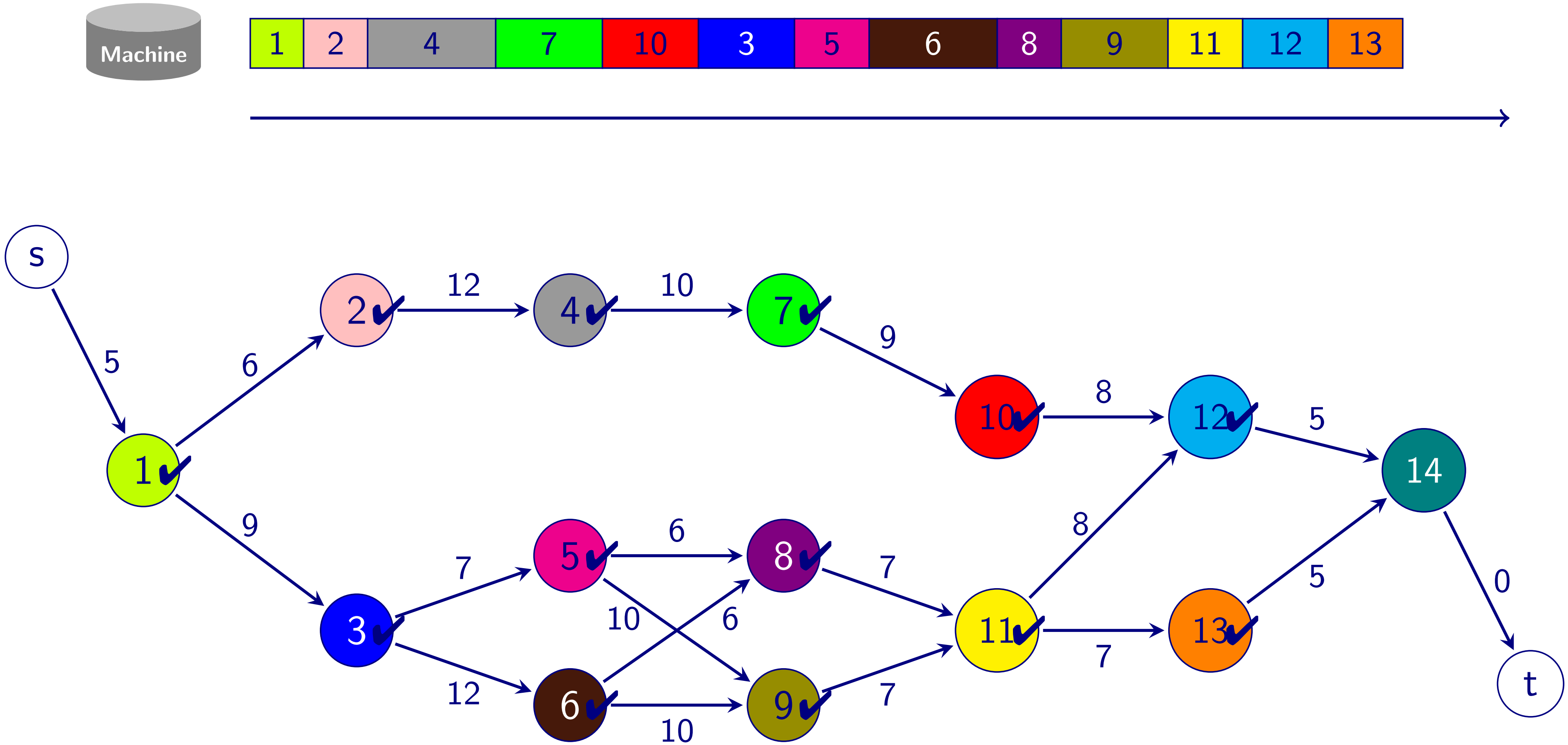




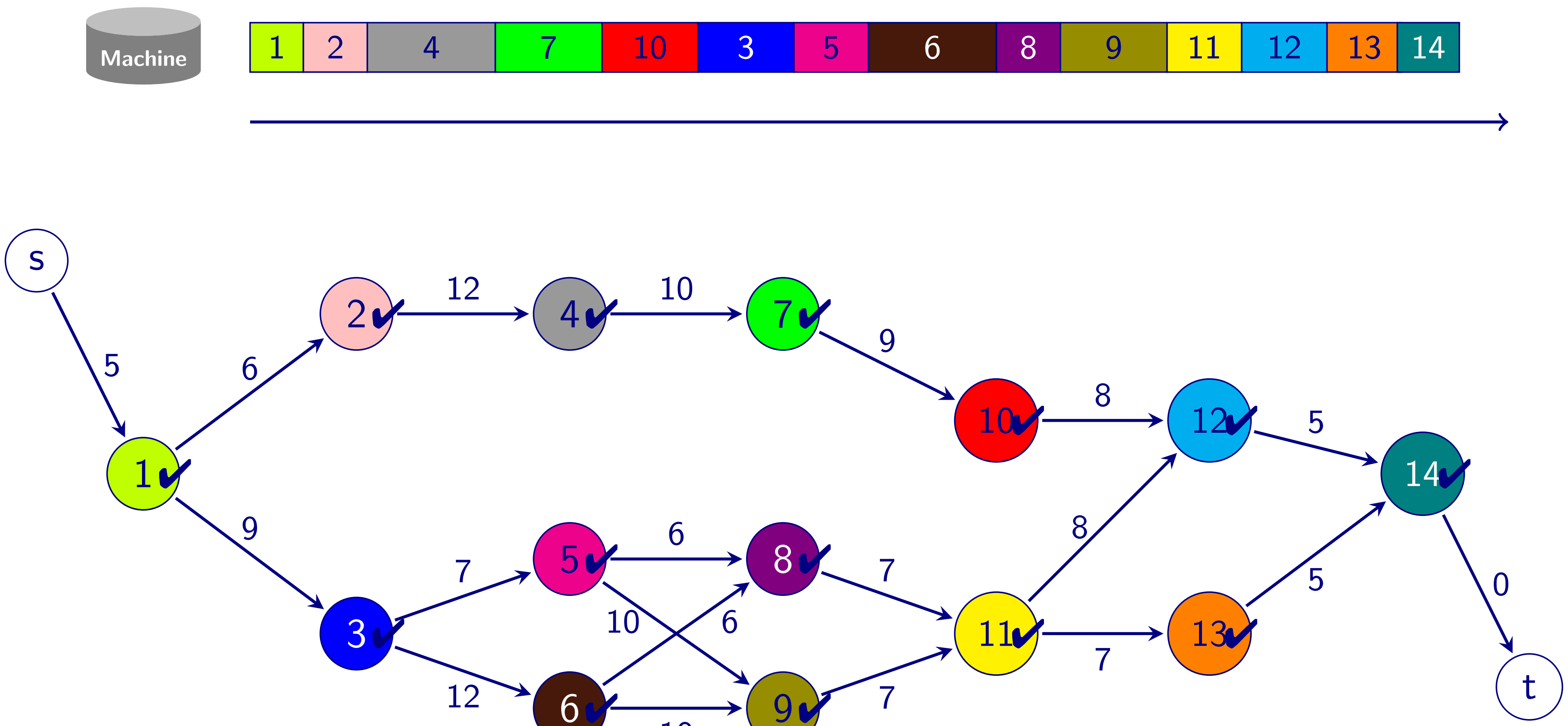
- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences



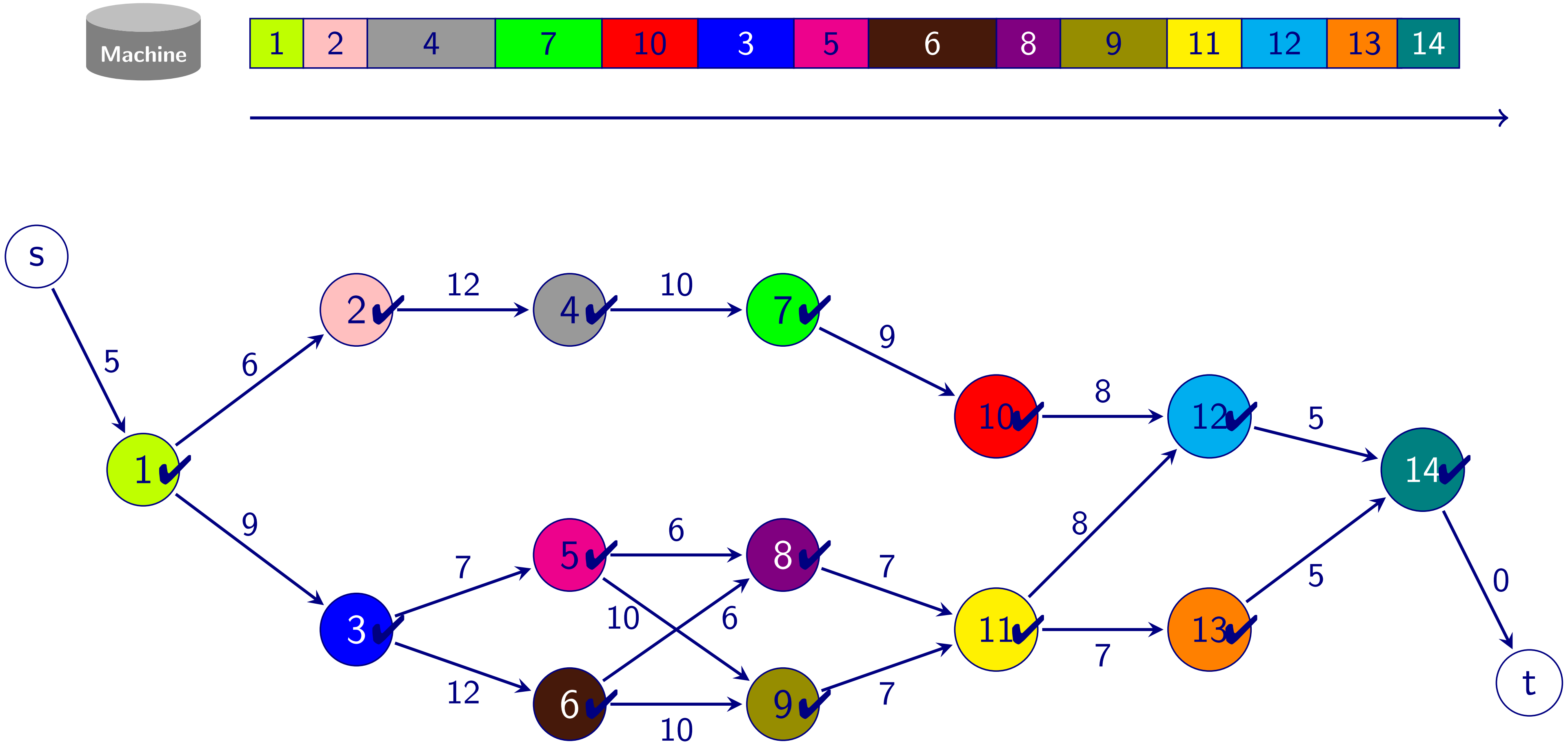
- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences



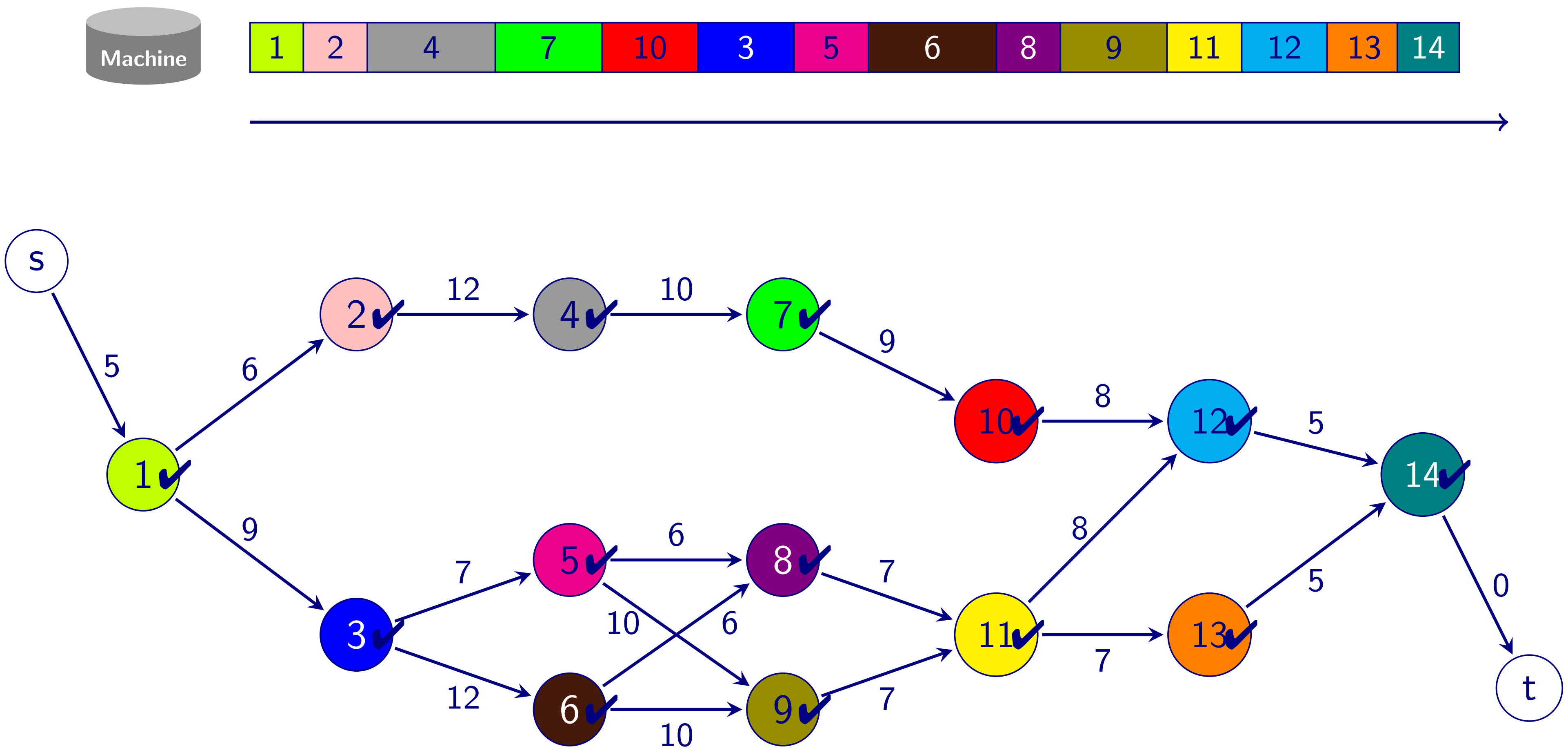
- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences



- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences
  - ➡ Polynomial runtime



- Greedy strategy: schedule an arbitrary job next with already fulfilled precedences
  - ➡ Polynomial runtime ➡ Efficient algorithm!

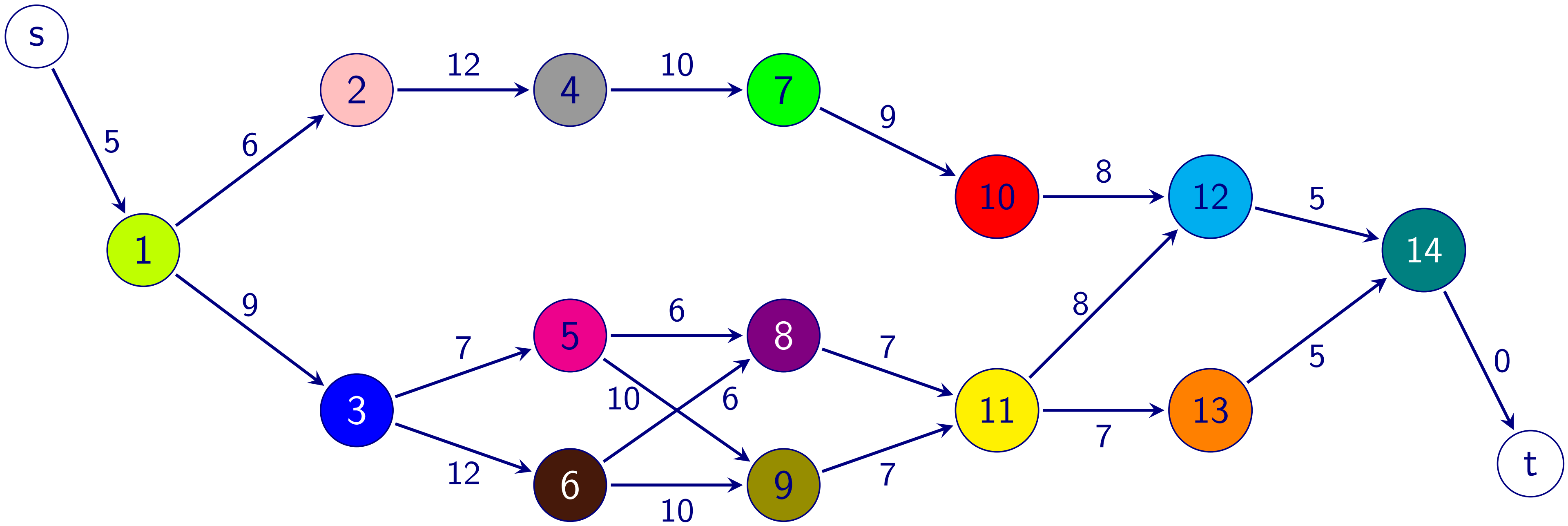


- Suppose there are arbitrarily many machines available

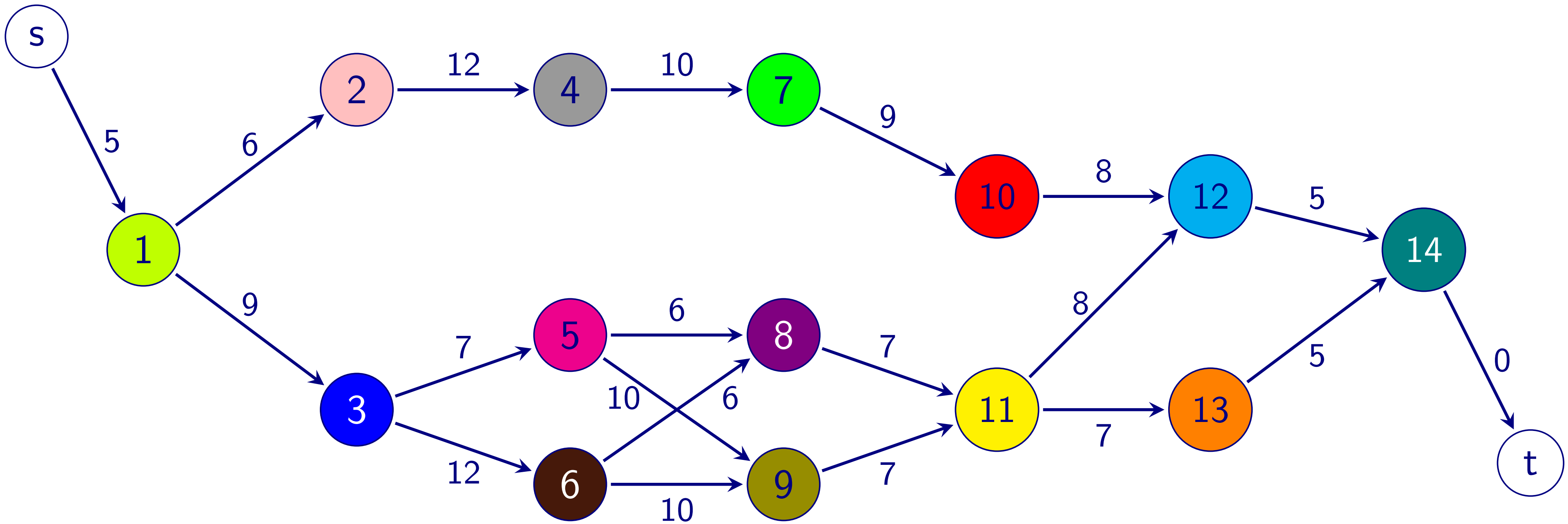
- ▷ Suppose there are arbitrarily many machines available
  - ➡ All jobs with fulfilled precedences can be carried out immediately and parallely

- Suppose there are arbitrarily many machines available
  - ➡ All jobs with fulfilled precedences can be carried out immediately and parallely
- Example: Project scheduling on construction site

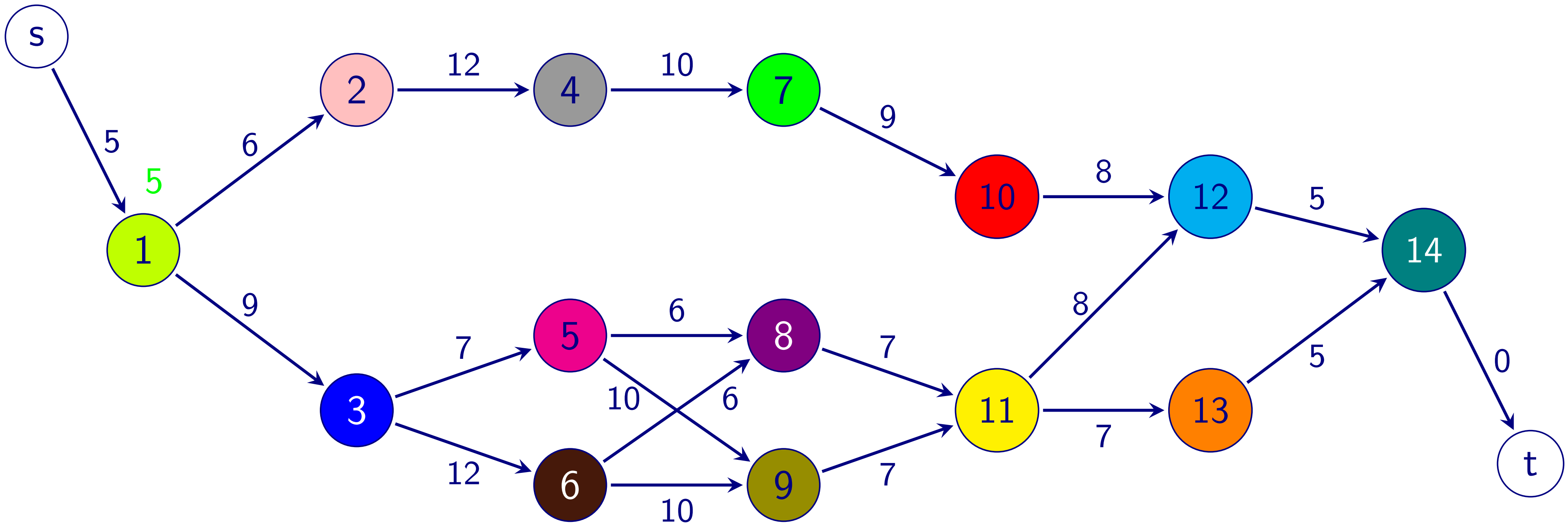
- ▷ Suppose there are arbitrarily many machines available
  - ➡ All jobs with fulfilled precedences can be carried out immediately and parallely
- ▷ Example: Project scheduling on construction site
  - ➡ Different tasks done by different contractors: Concrete builder, stonemasonry, house painter, glazier, ...
  - ➡ can provide as many workers as necessary to carry out each task


s
5
1
6
9
2
12
4
10
7
9
3
7
12
5
6
10
6
6
10
8
9
7
7
10
8
11
8
7
12
13
5
5
14
0
t

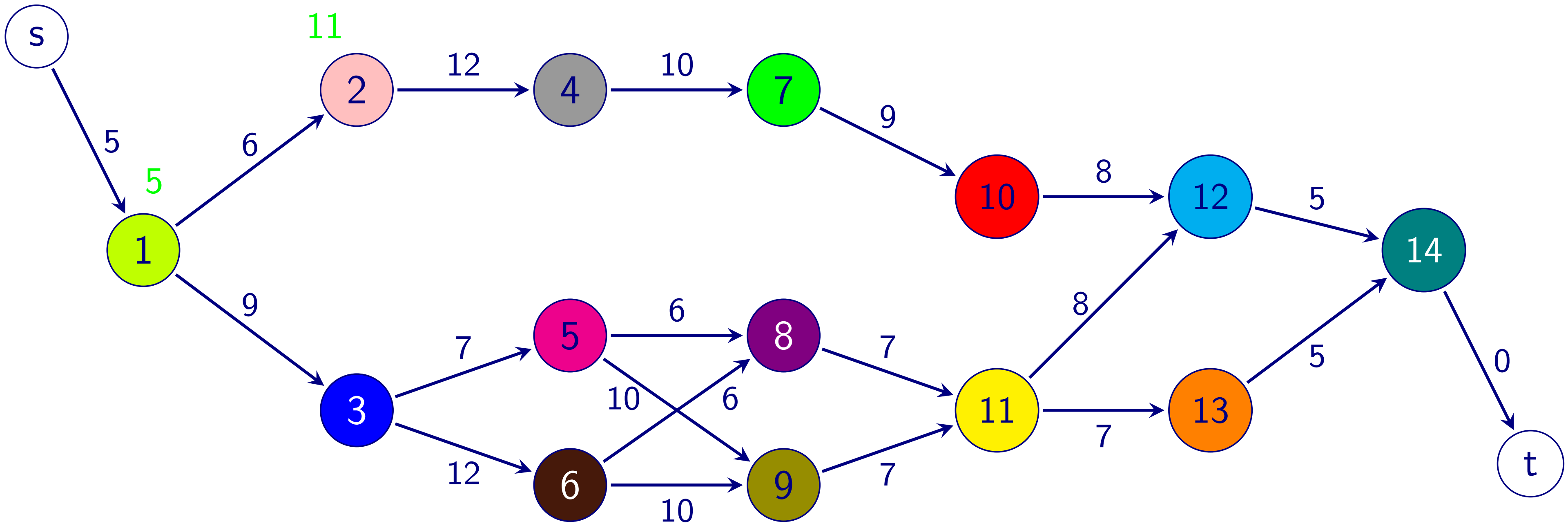
▷ Forward procedure: compute earliest possible completion times for all jobs



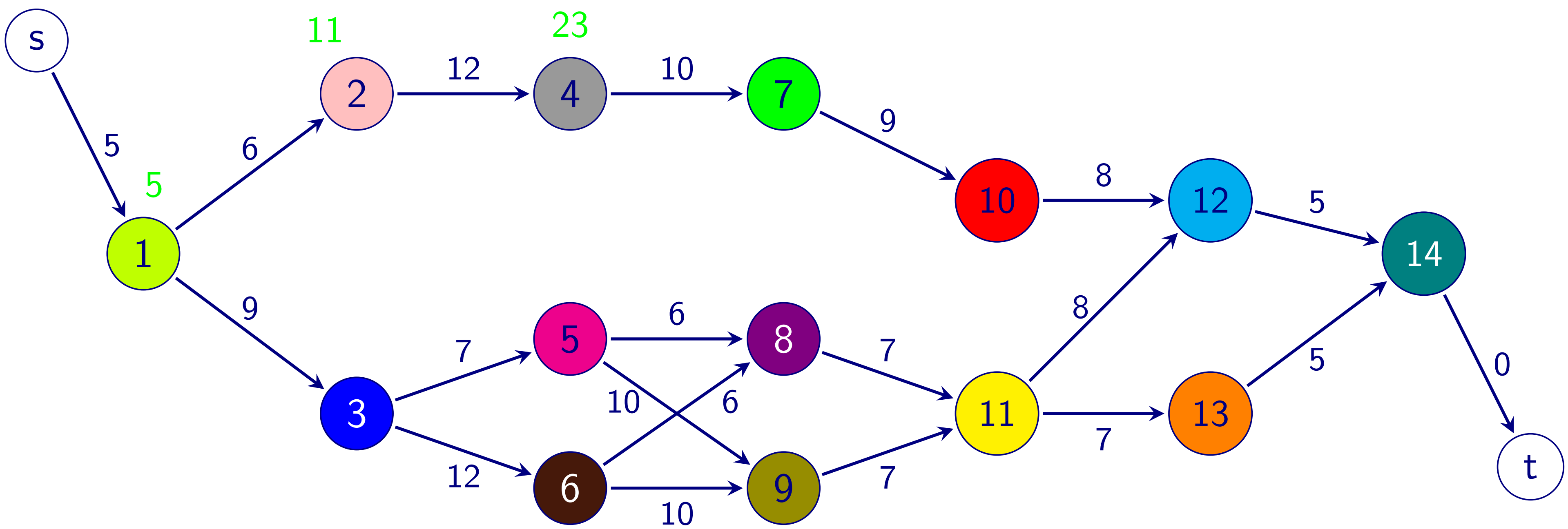
- Forward procedure: compute earliest possible completion times for all jobs



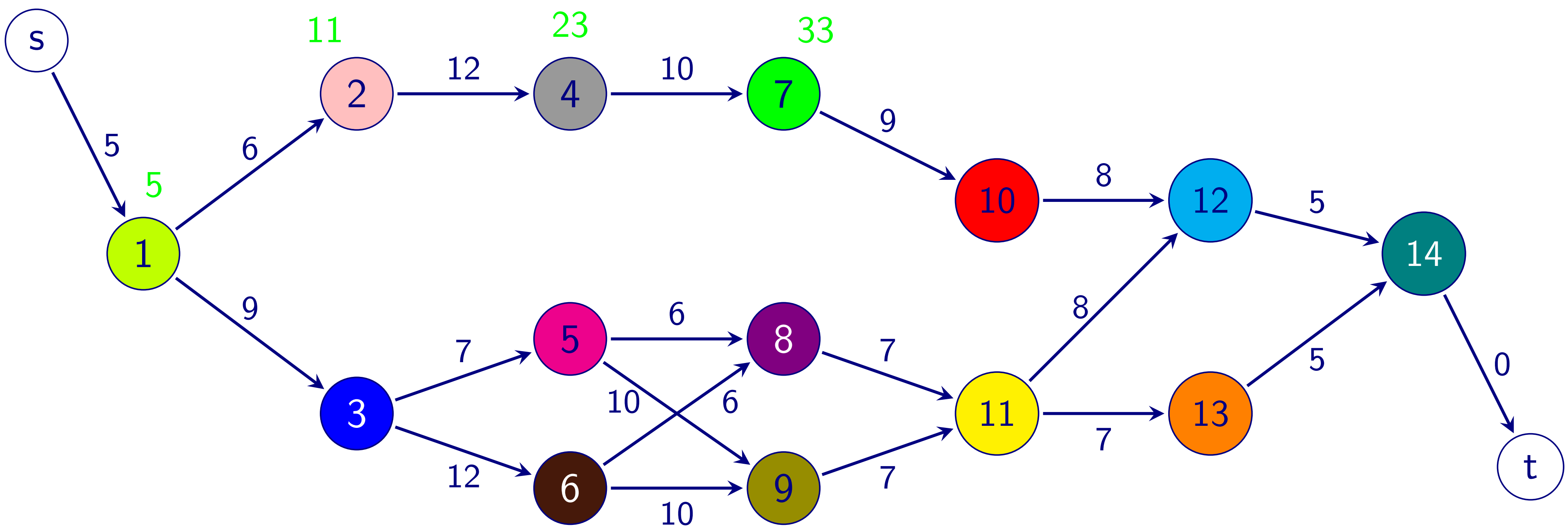
- Forward procedure: compute earliest possible completion times for all jobs



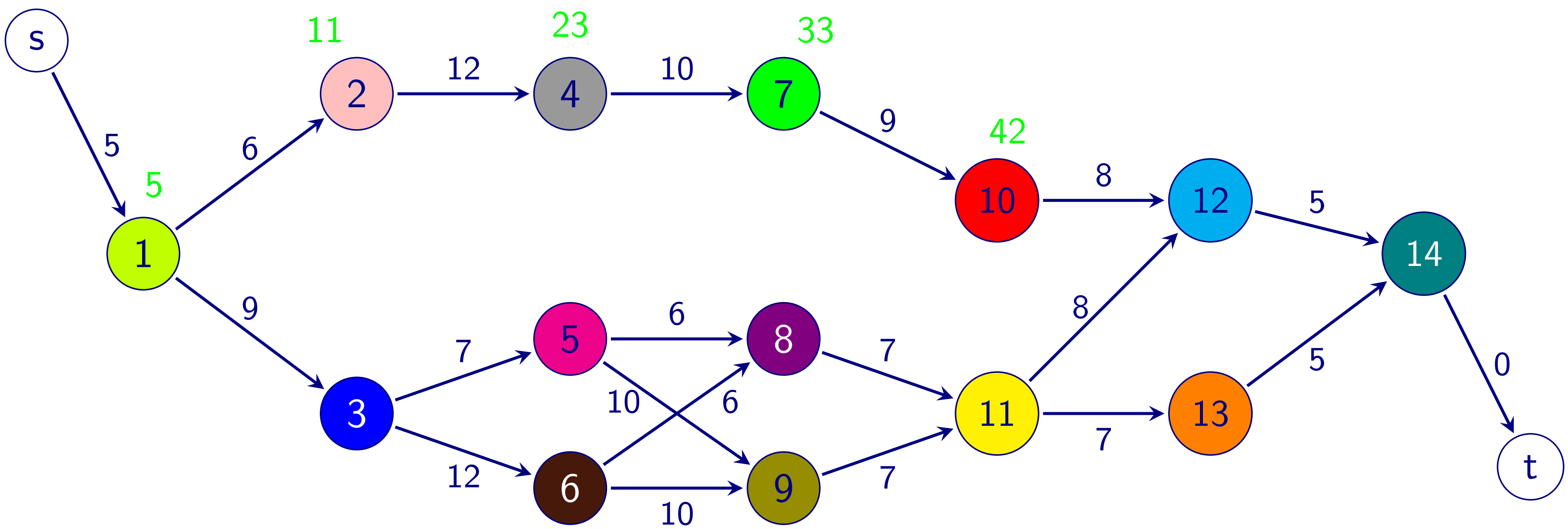
▷ Forward procedure: compute earliest possible completion times for all jobs



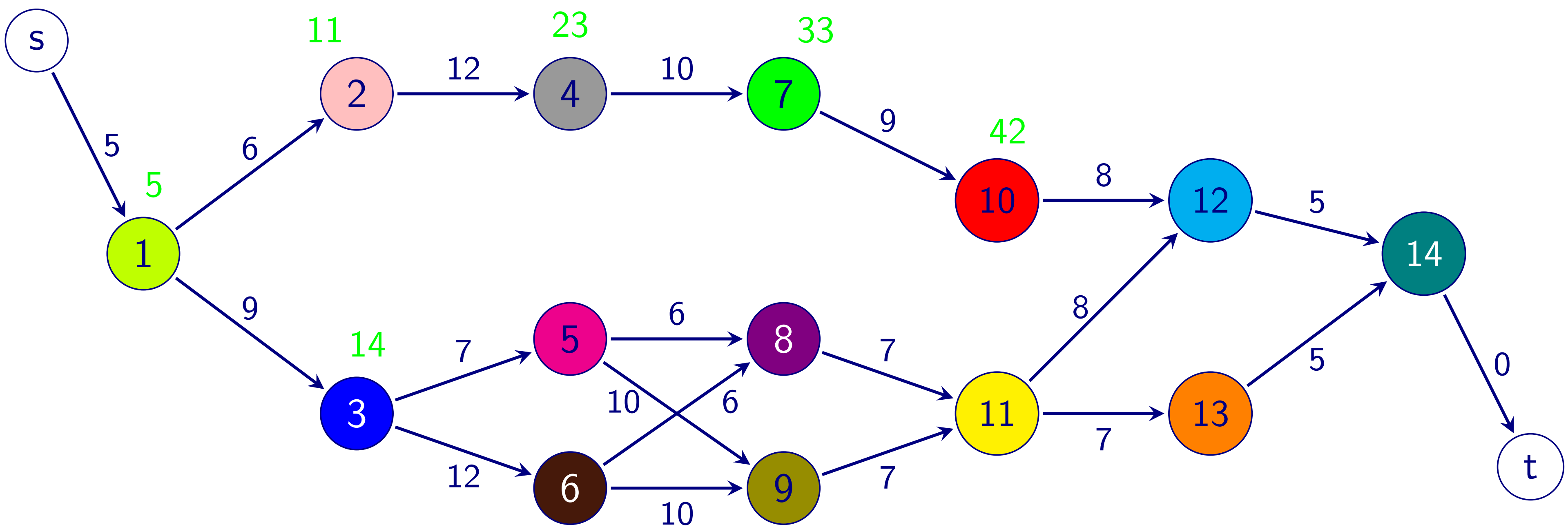
▷ Forward procedure: compute earliest possible completion times for all jobs



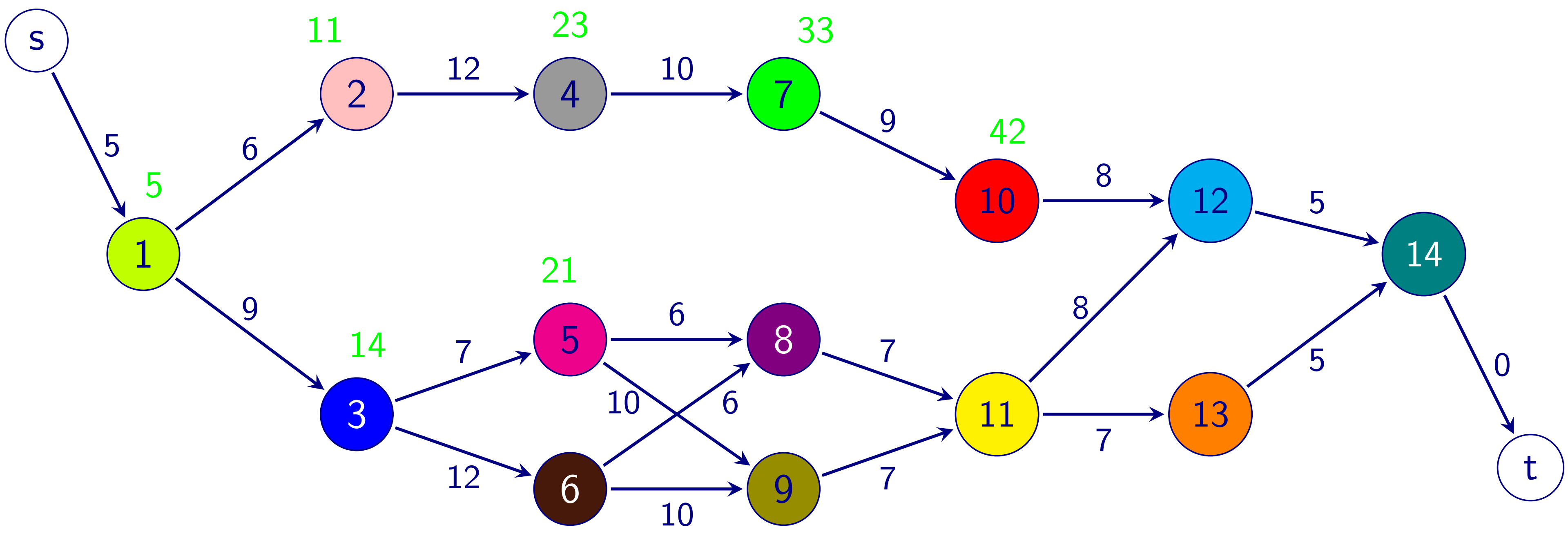
▷ Forward procedure: compute earliest possible completion times for all jobs



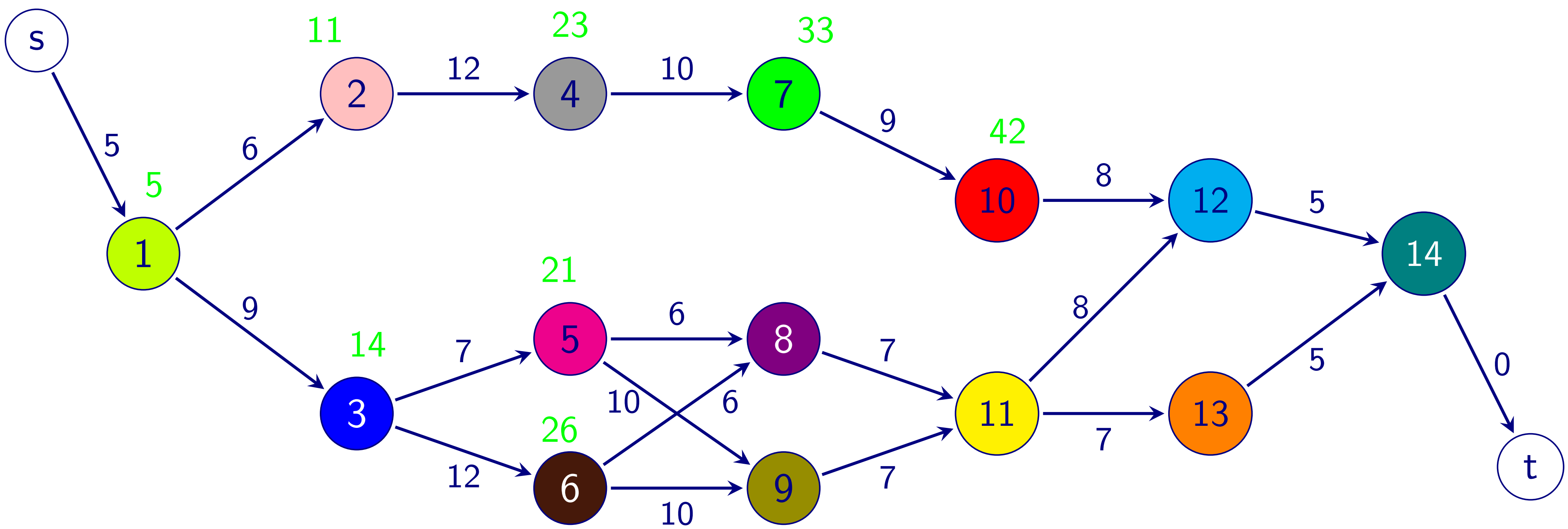
▷ Forward procedure: compute earliest possible completion times for all jobs



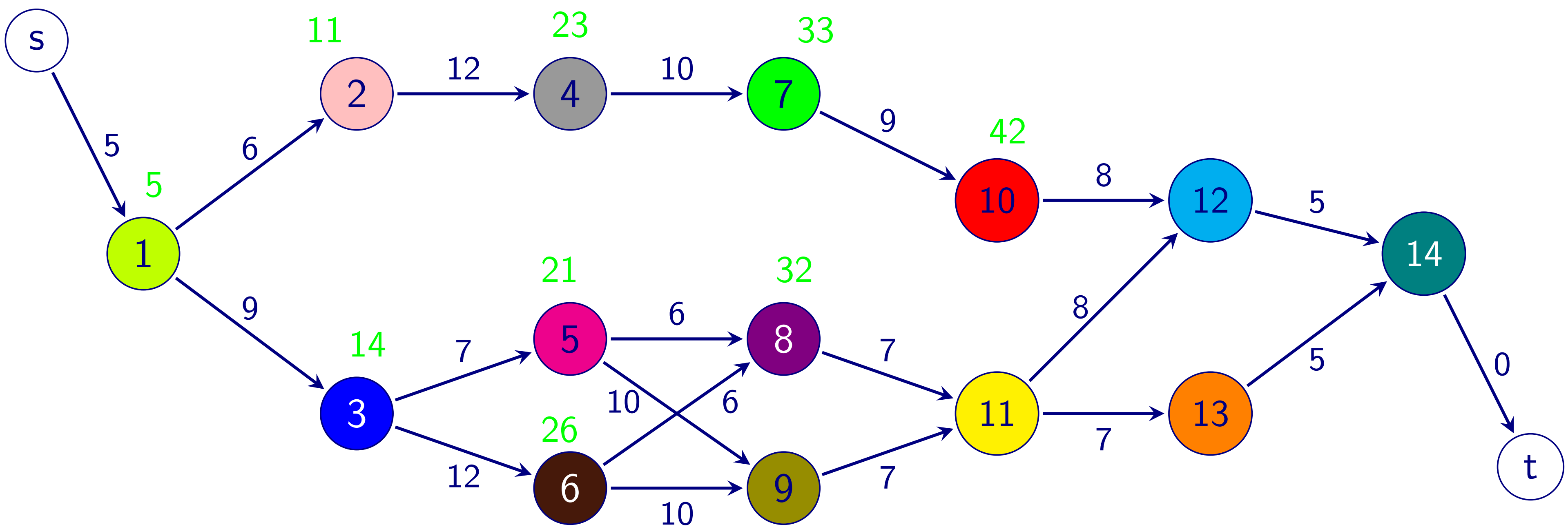
- Forward procedure: compute earliest possible completion times for all jobs



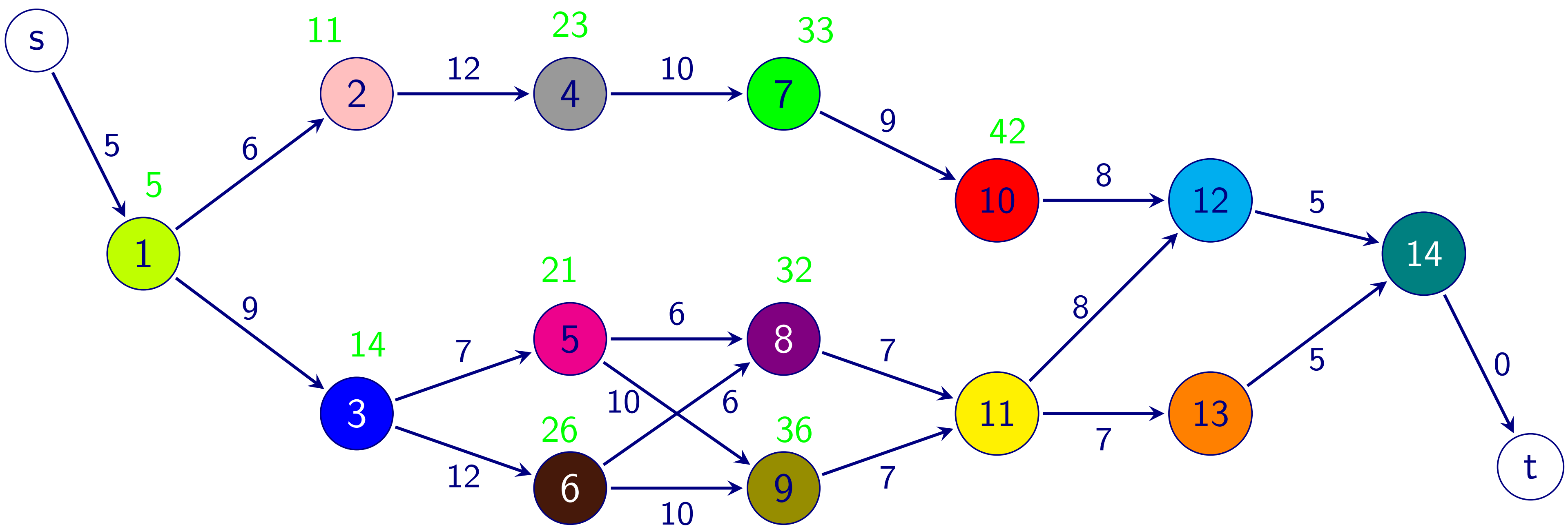
- Forward procedure: compute earliest possible completion times for all jobs



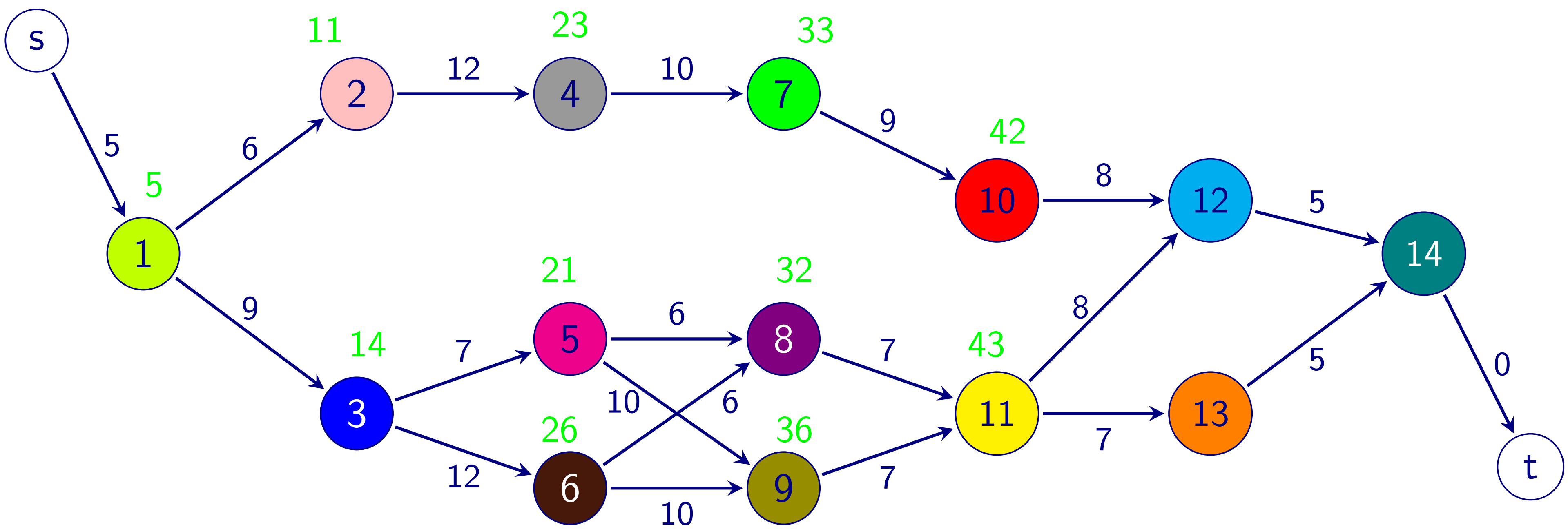
▷ Forward procedure: compute earliest possible completion times for all jobs



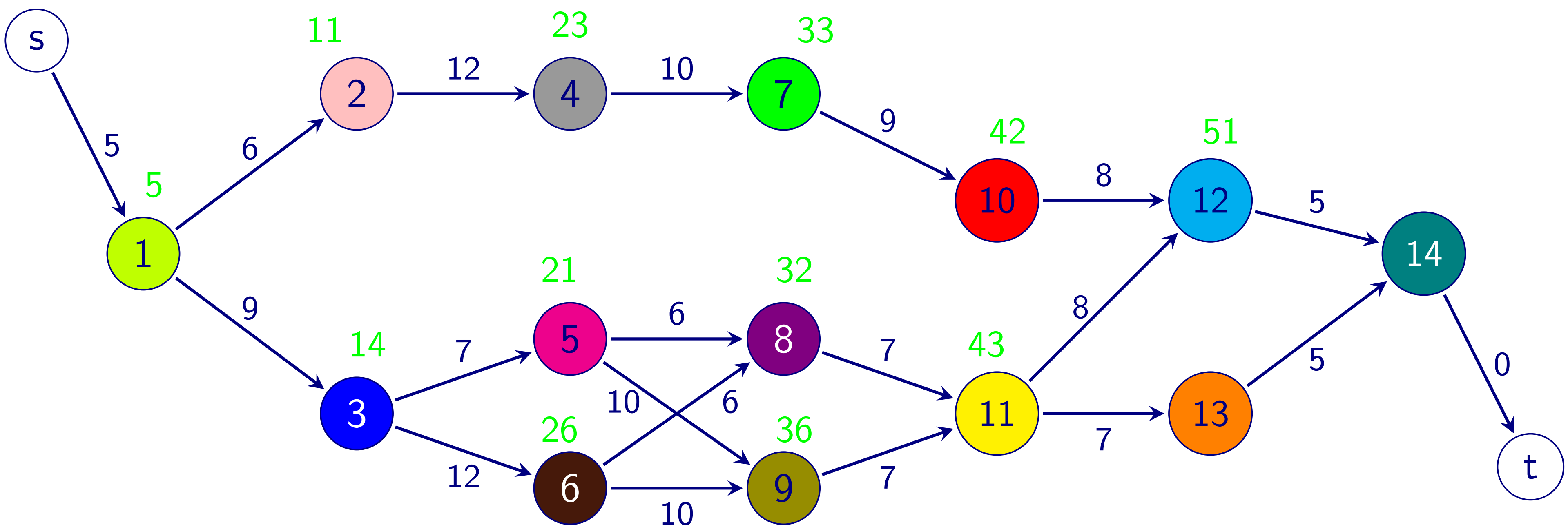
▷ Forward procedure: compute earliest possible completion times for all jobs



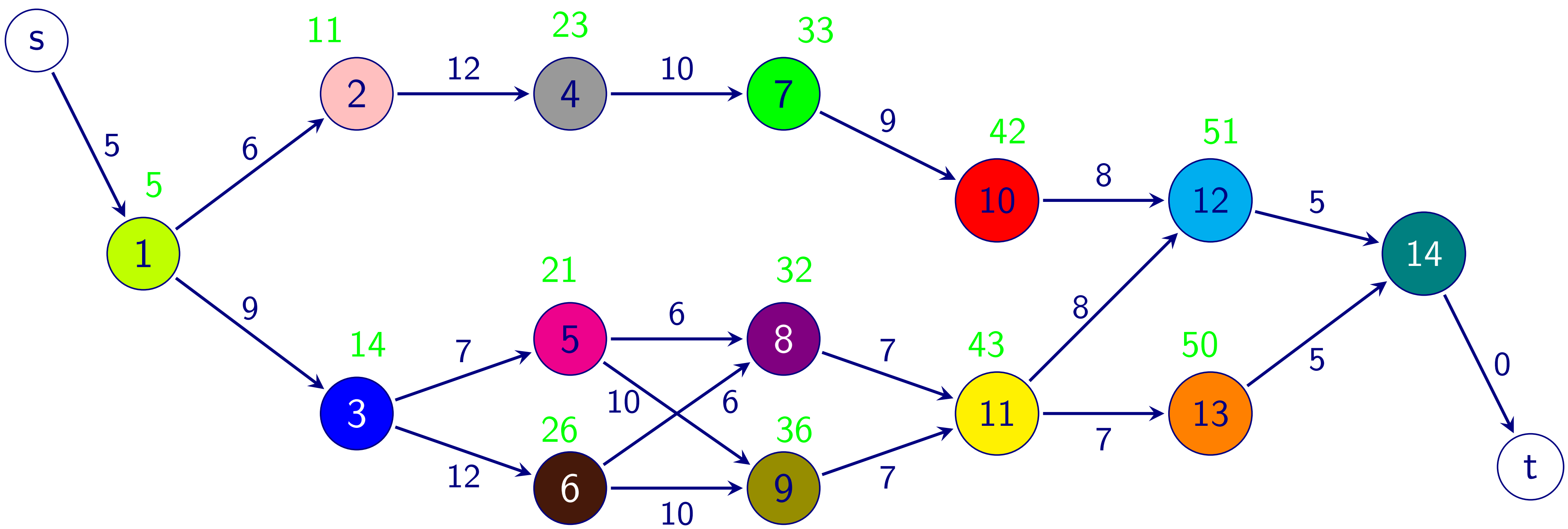
- Forward procedure: compute earliest possible completion times for all jobs



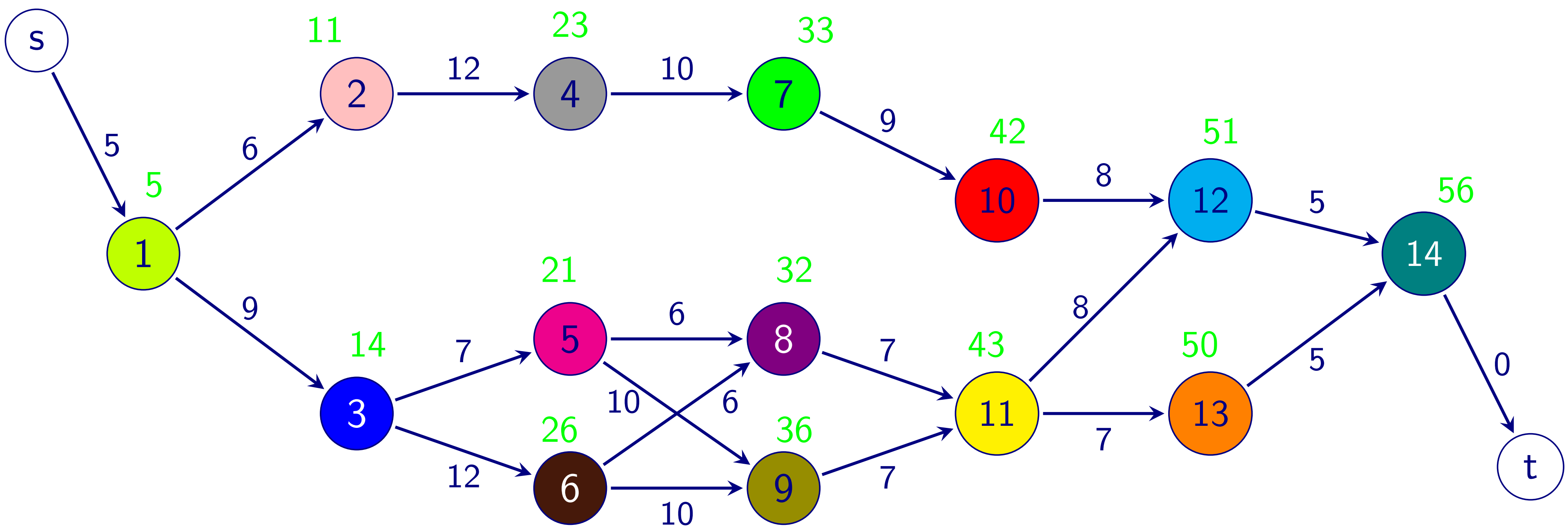
- Forward procedure: compute earliest possible completion times for all jobs



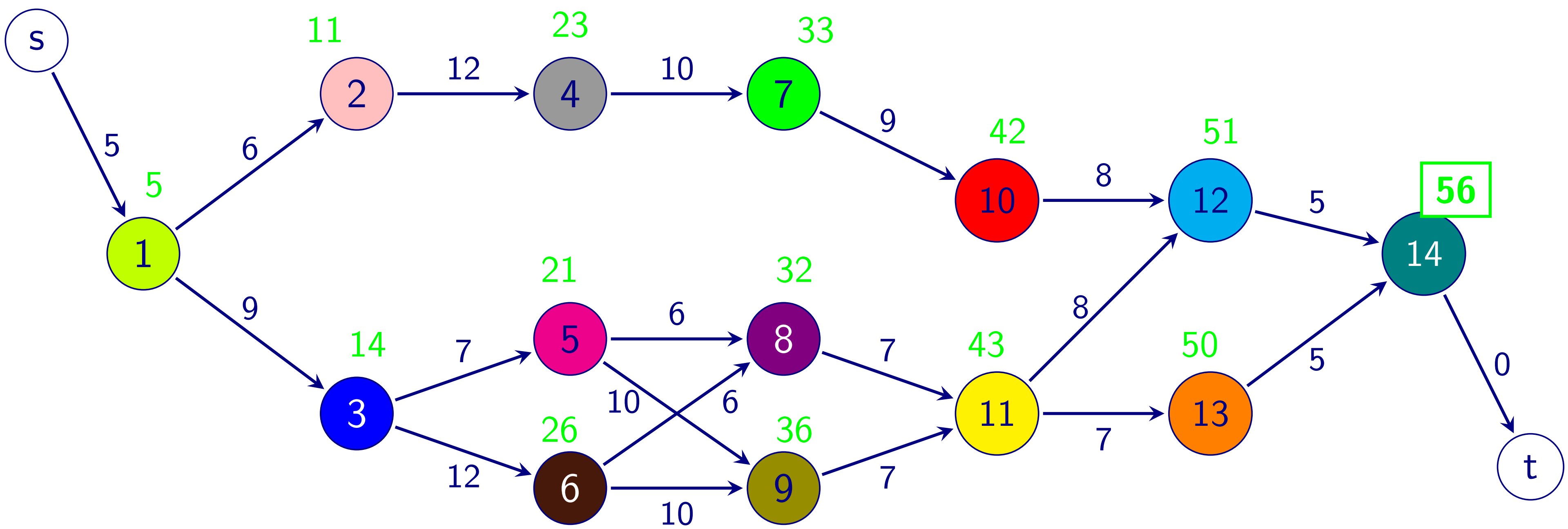
- Forward procedure: compute earliest possible completion times for all jobs



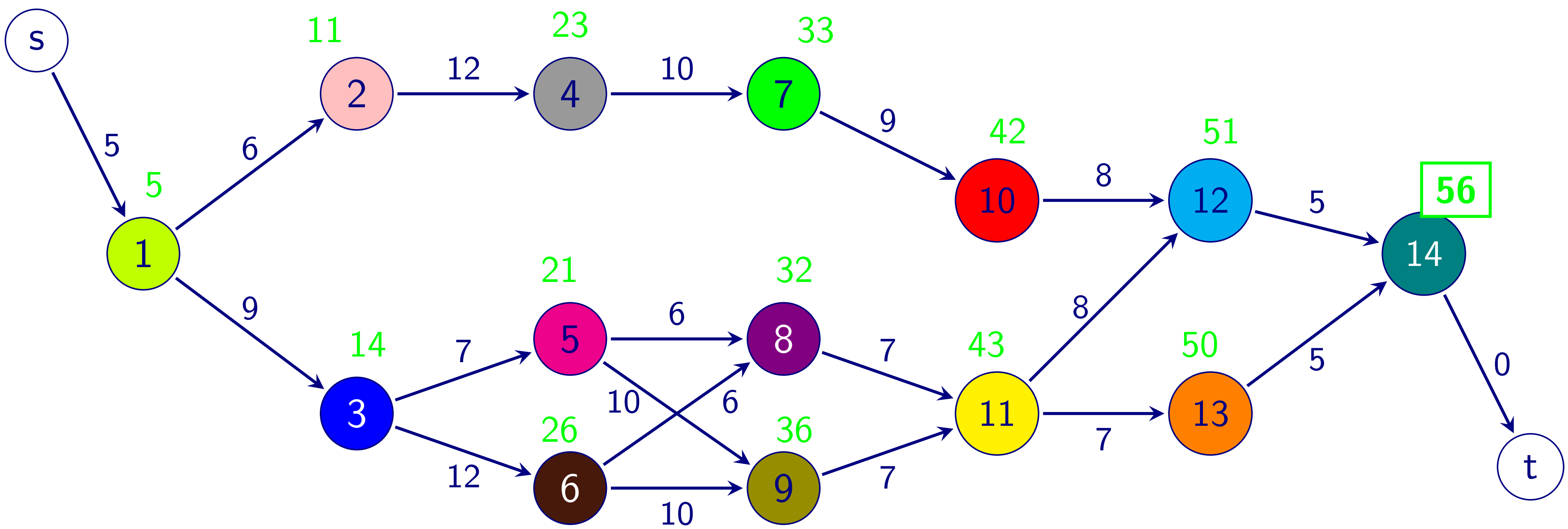
- Forward procedure: compute earliest possible completion times for all jobs



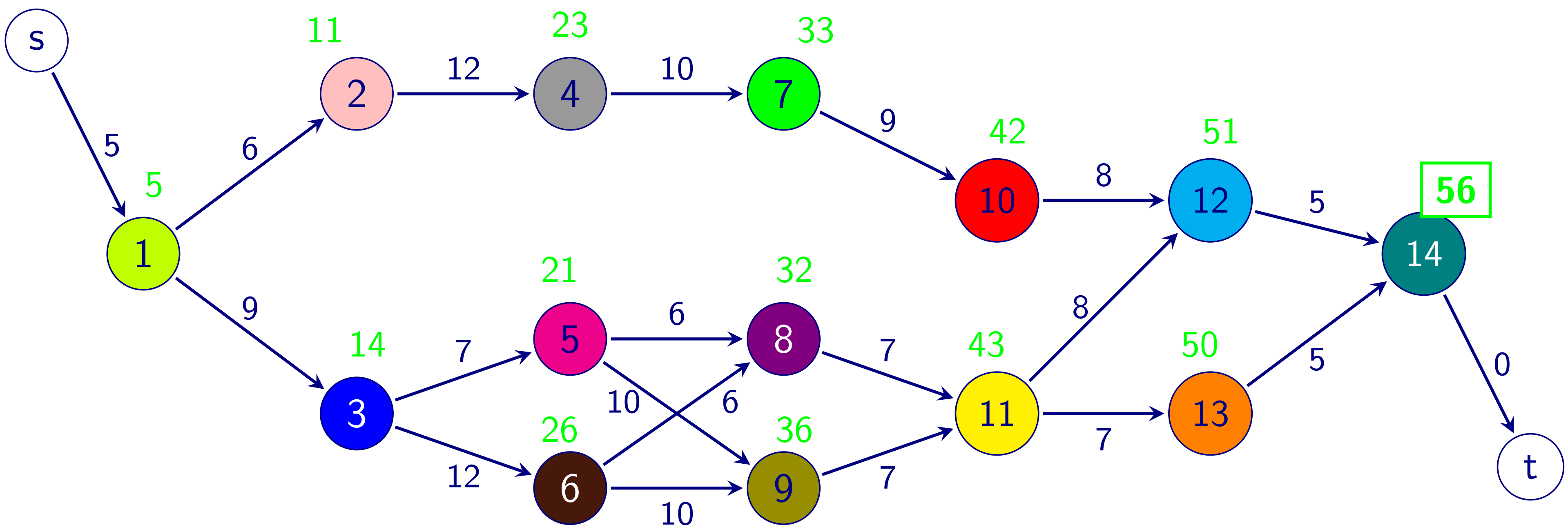
▷ Forward procedure: compute earliest possible completion times for all jobs



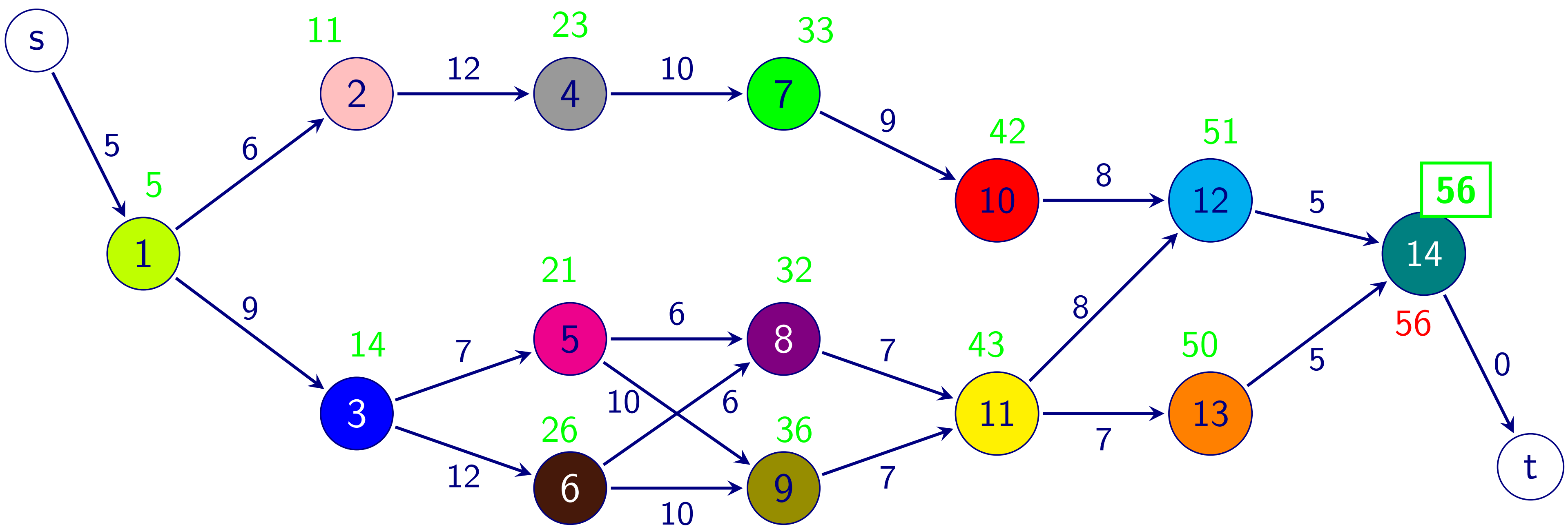
- Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed



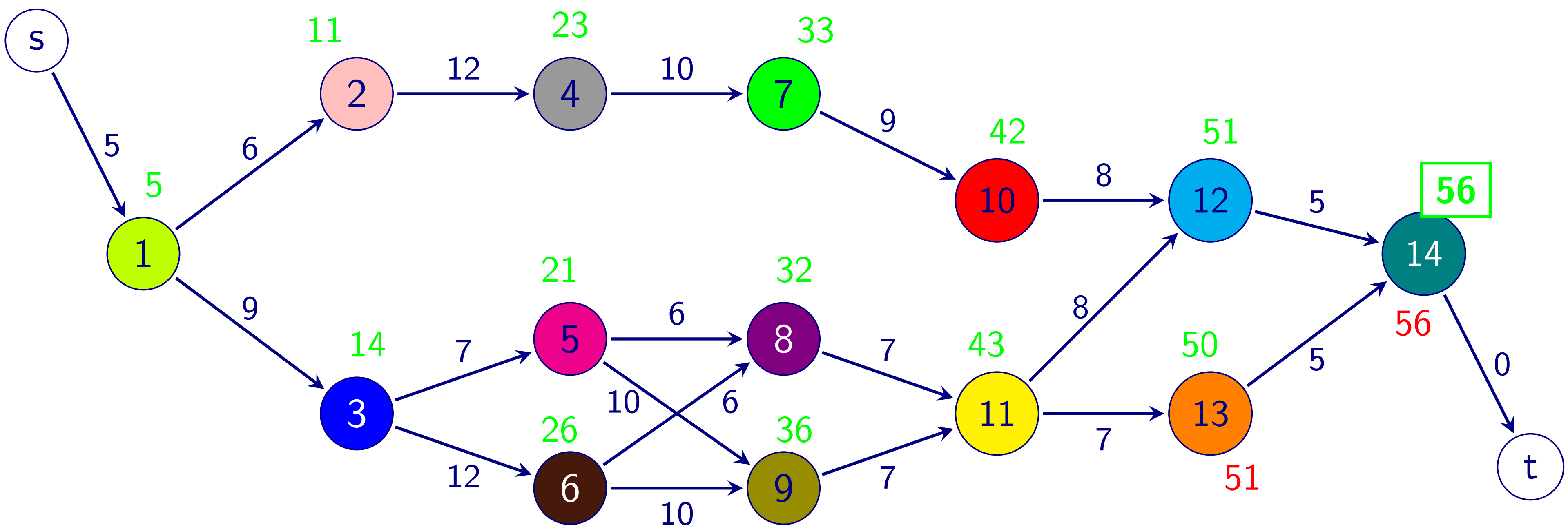
- Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- Backward procedure: compute latest possible completion times for all jobs



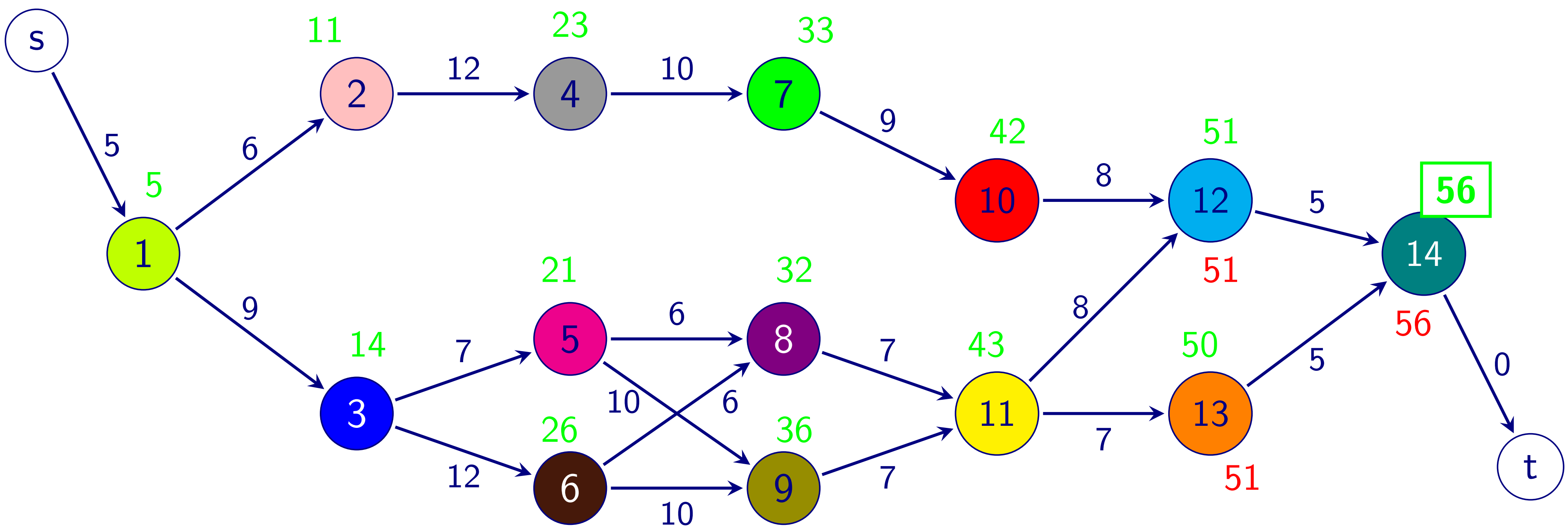
- Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- Backward procedure: compute latest possible completion times for all jobs



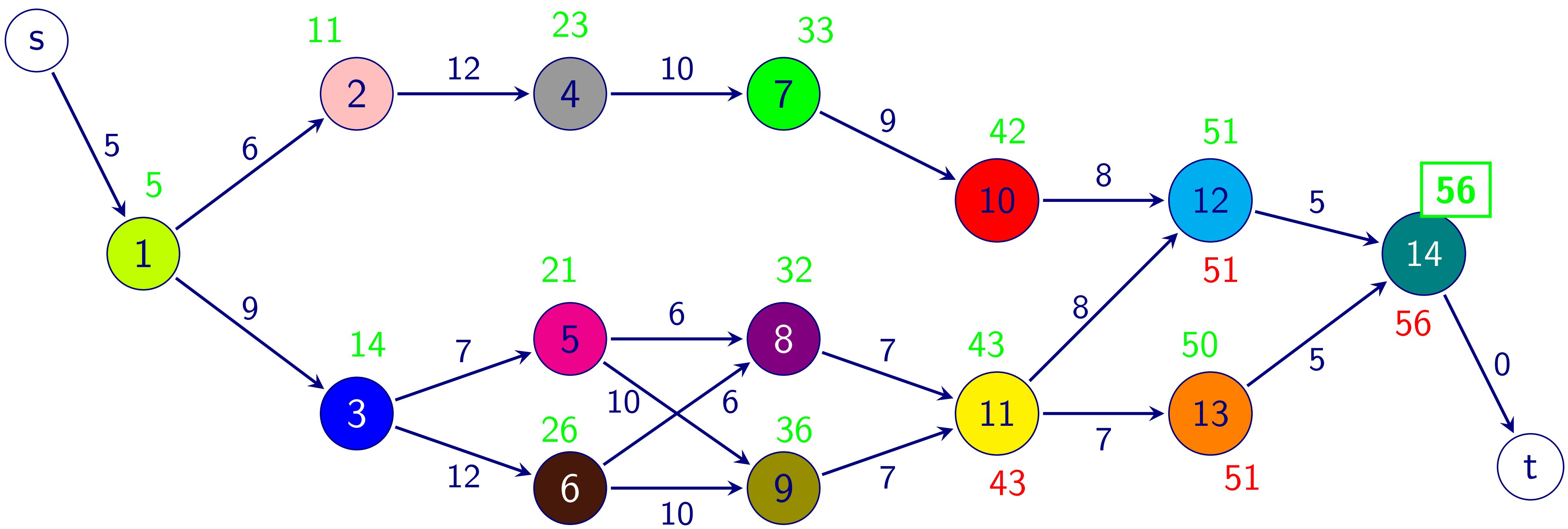
- Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- Backward procedure: compute latest possible completion times for all jobs



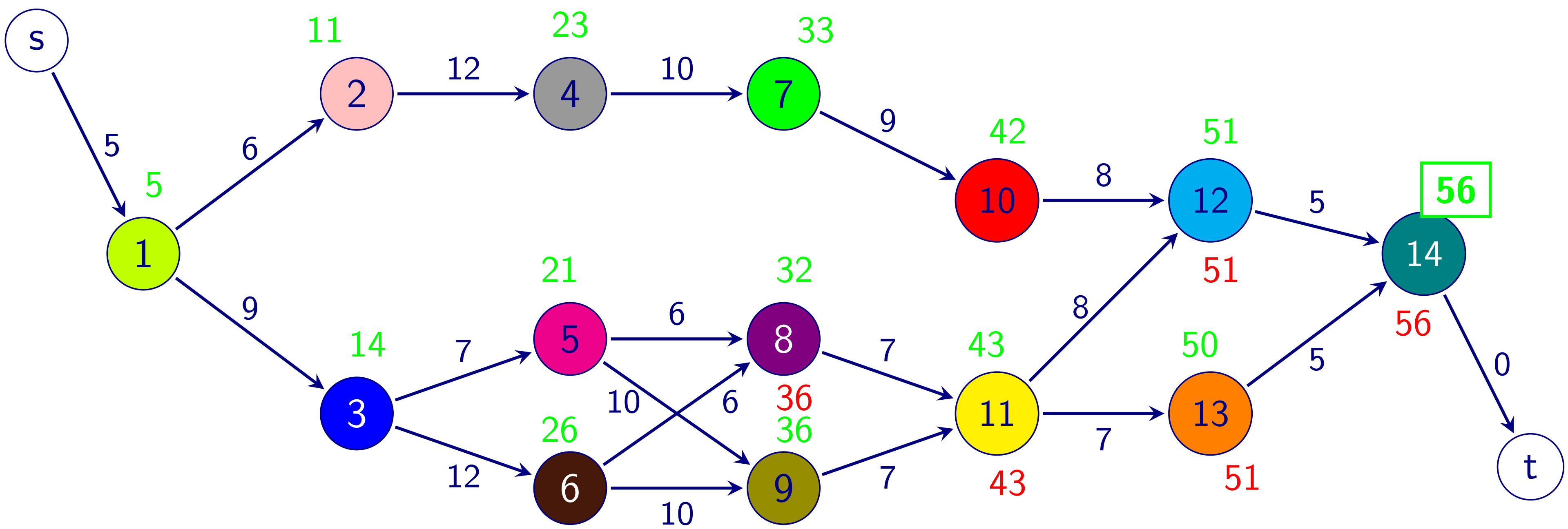
- Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- Backward procedure: compute latest possible completion times for all jobs



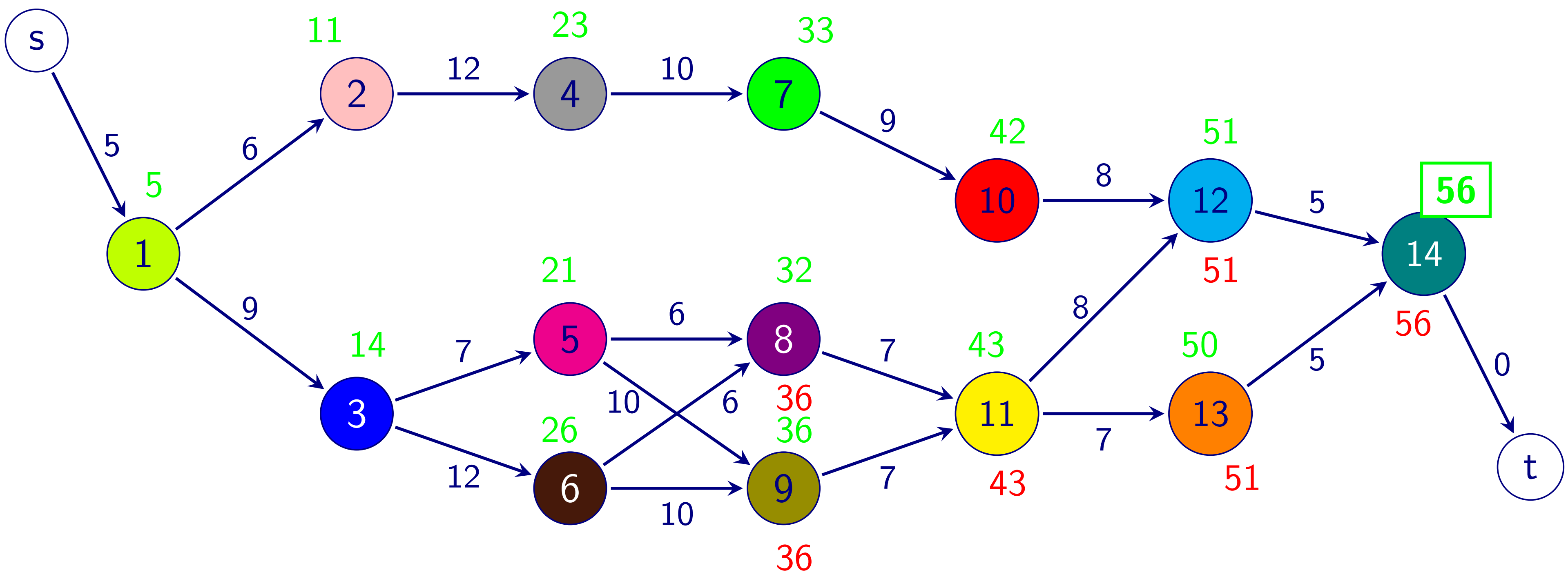
- ▷ Forward procedure: compute **earliest possible completion times** for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute **latest possible completion times** for all jobs



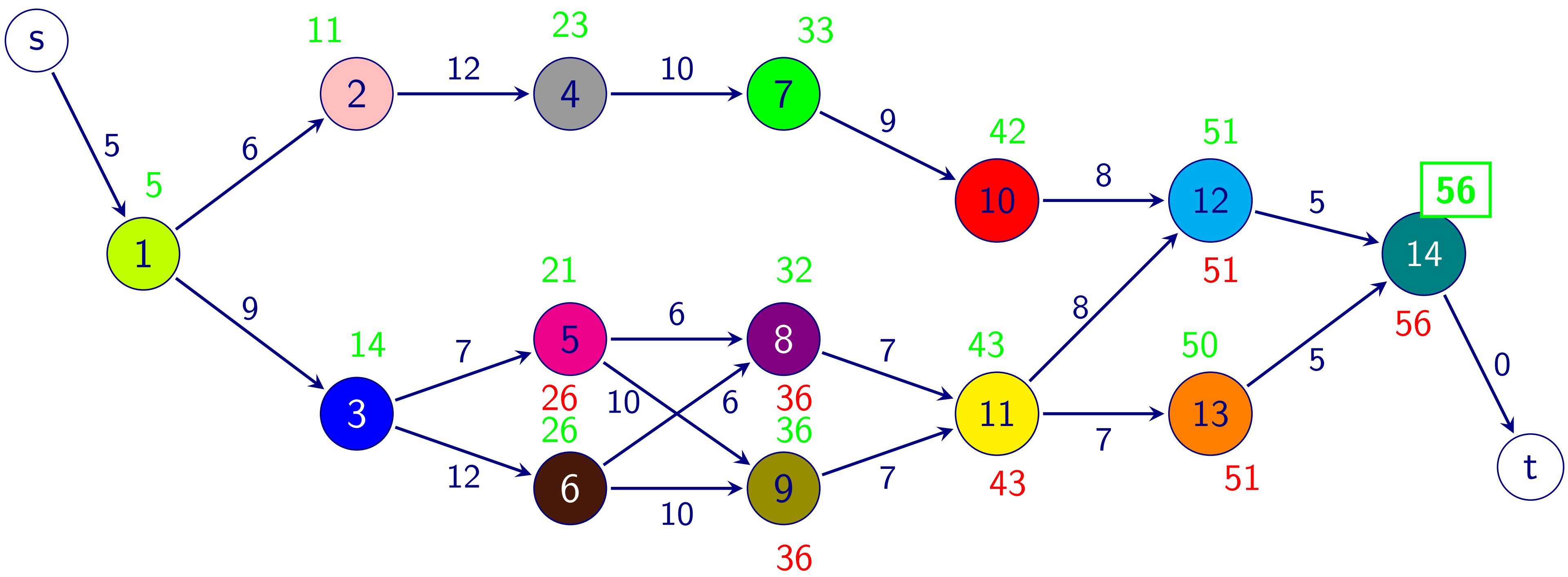
- Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- Backward procedure: compute latest possible completion times for all jobs



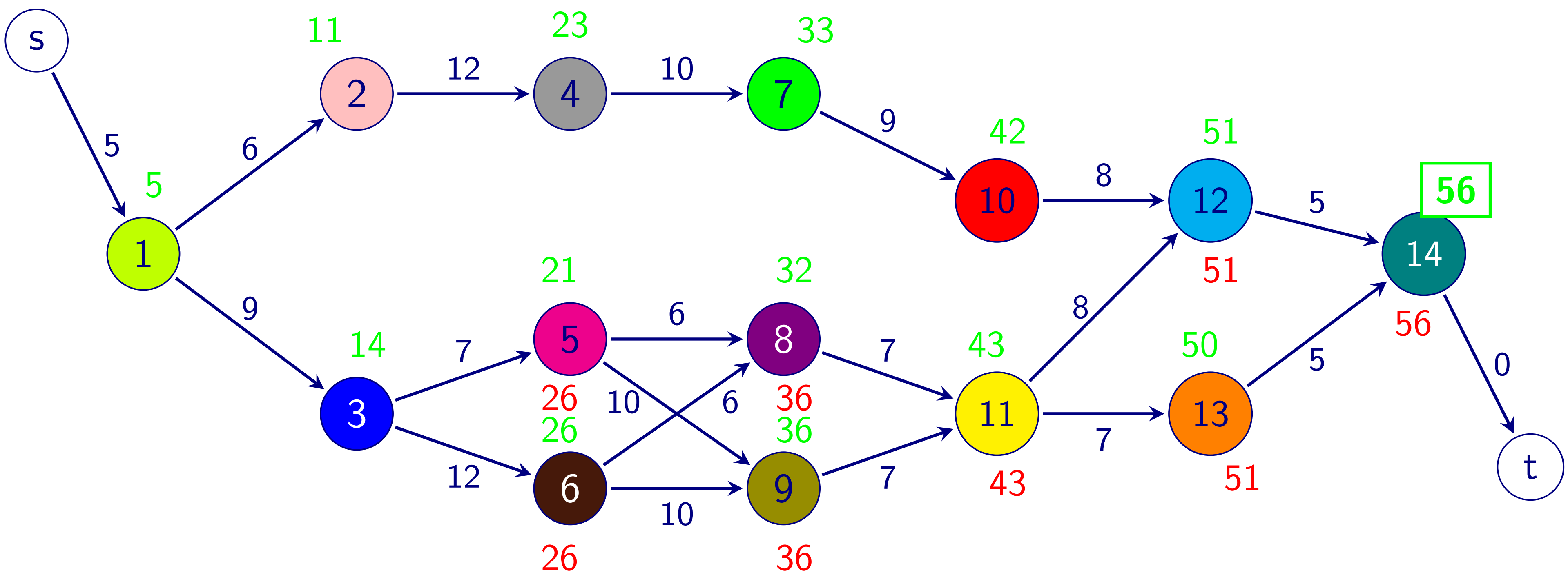
- ▷ Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute latest possible completion times for all jobs



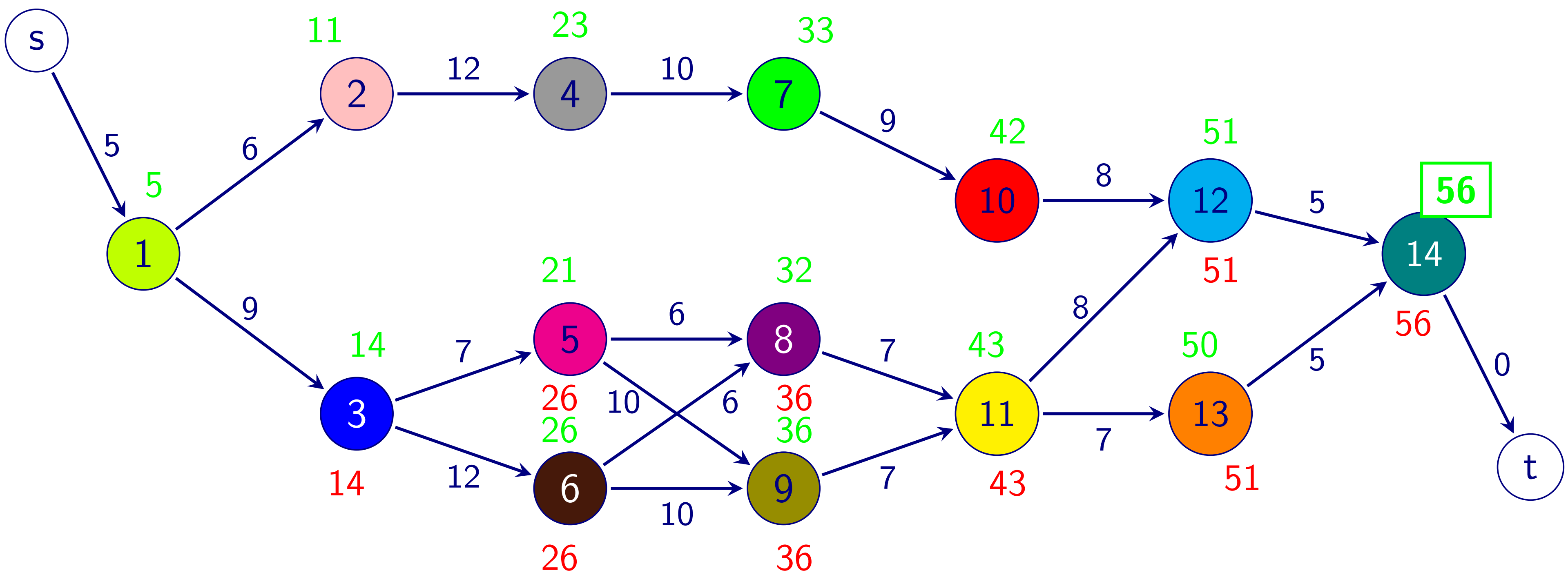
- ▷ Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute latest possible completion times for all jobs



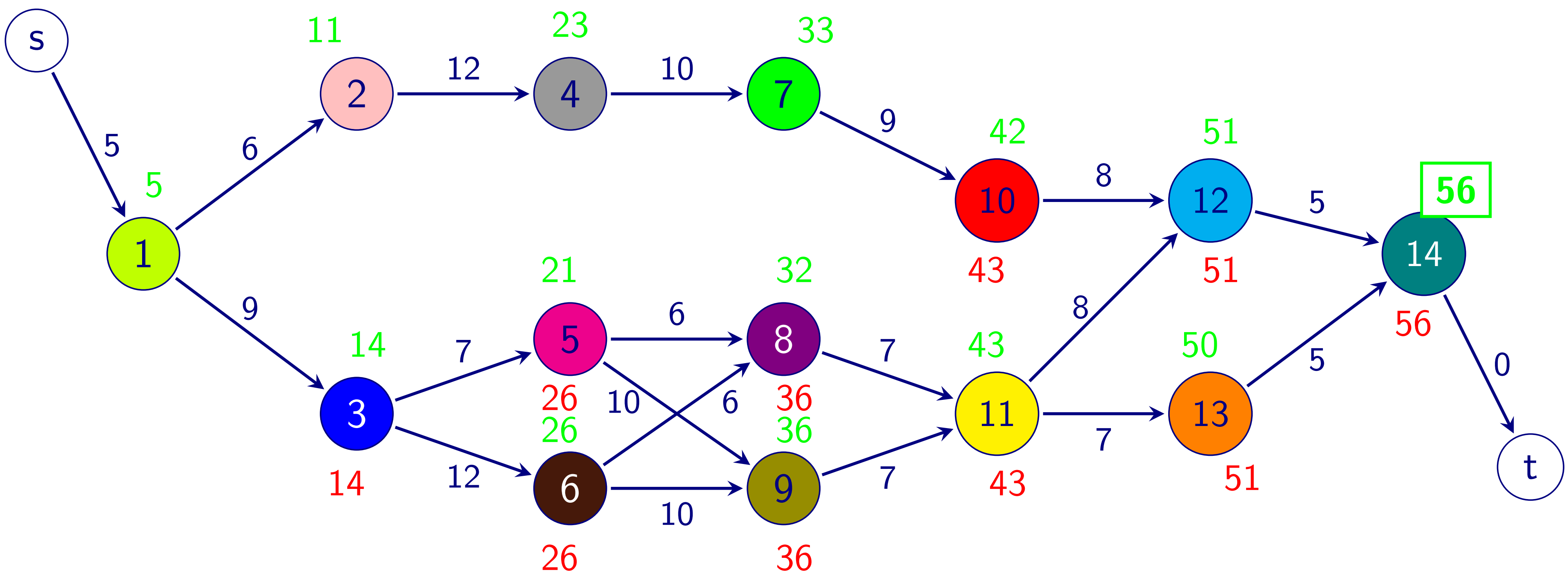
- Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- Backward procedure: compute latest possible completion times for all jobs



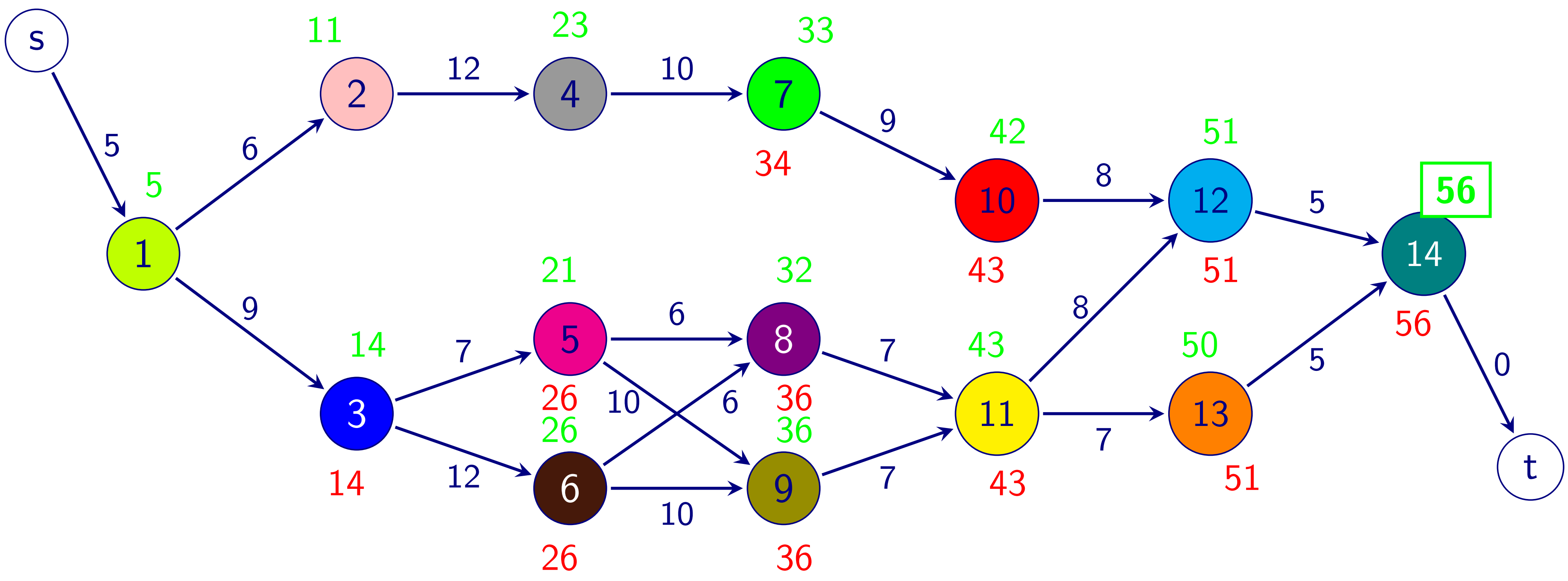
- ▷ Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute latest possible completion times for all jobs



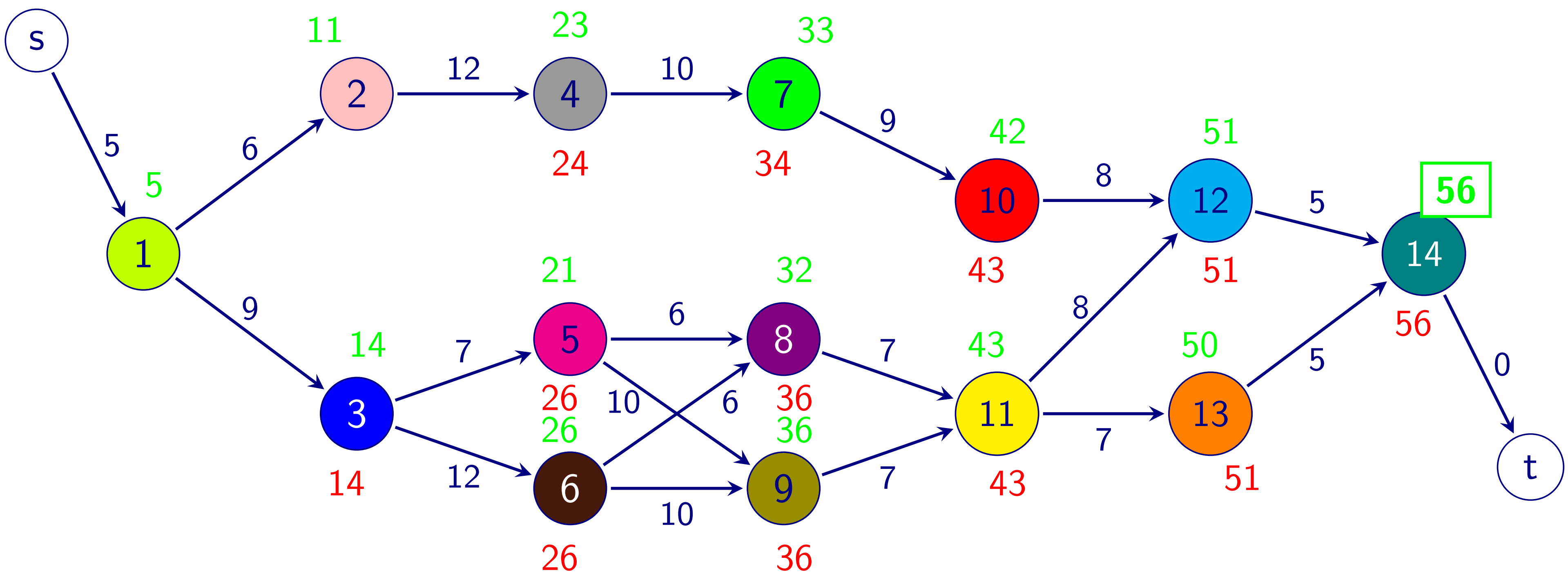
- ▷ Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute latest possible completion times for all jobs



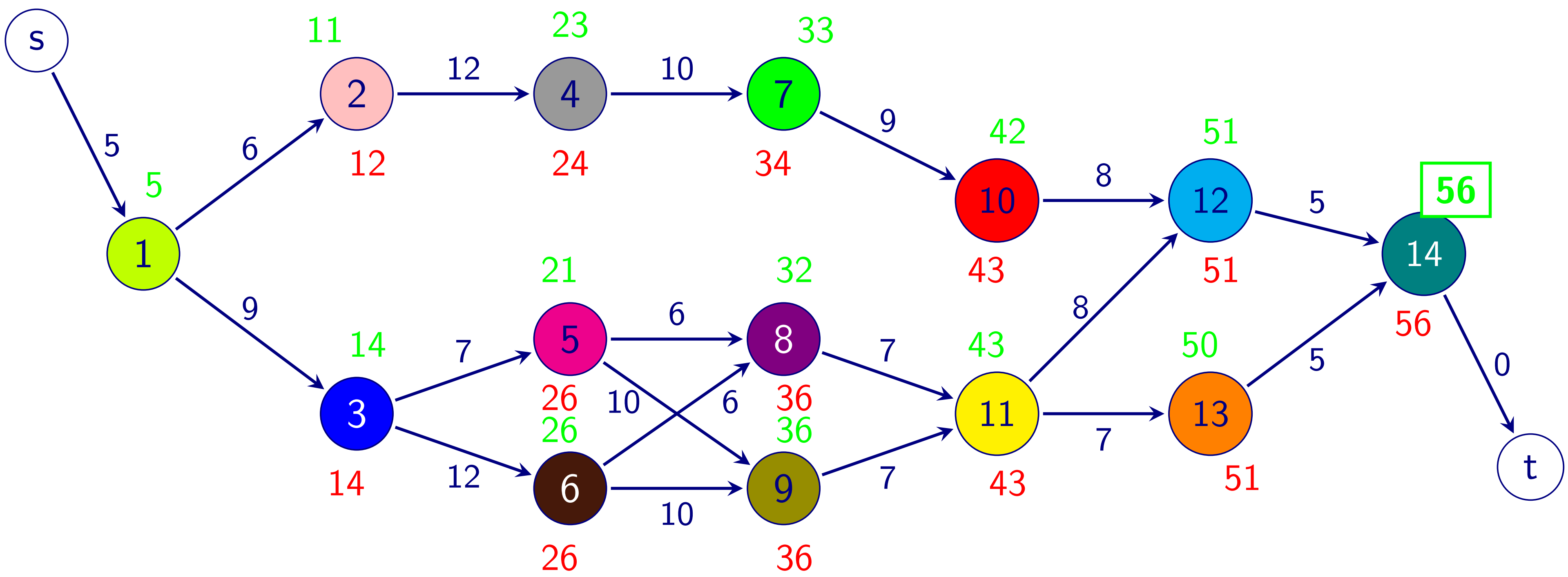
- ▷ Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute latest possible completion times for all jobs



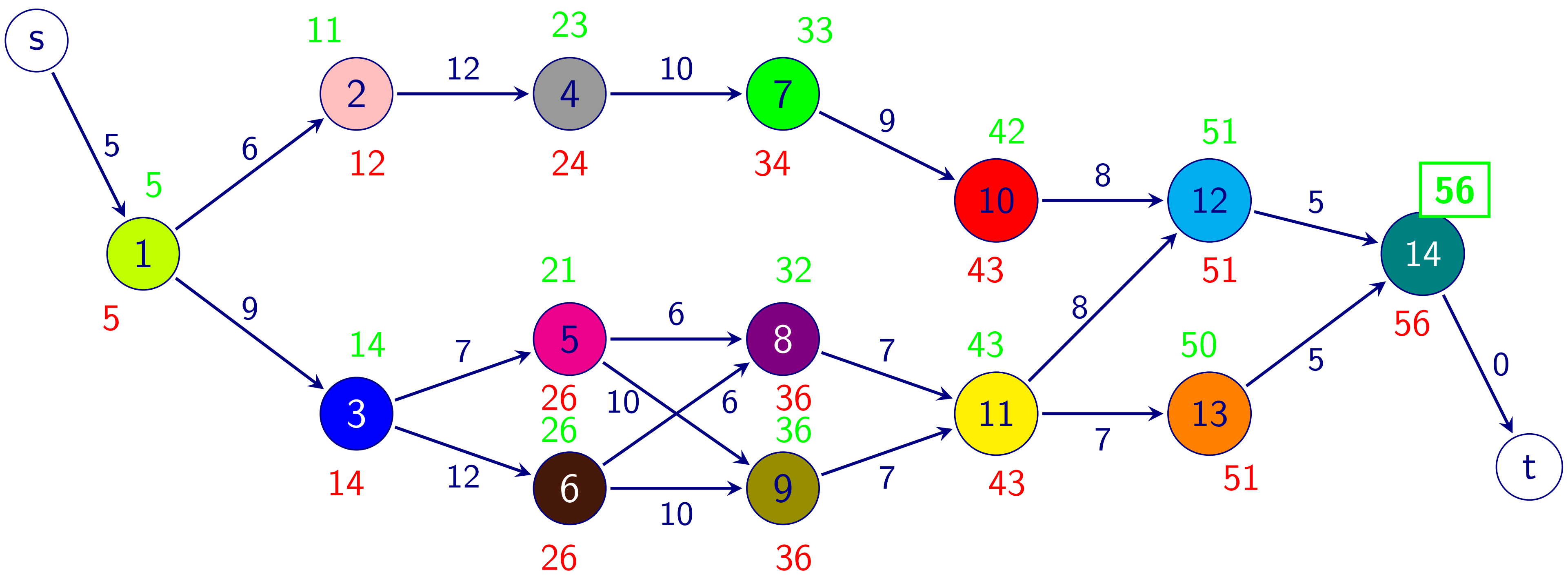
- Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- Backward procedure: compute latest possible completion times for all jobs



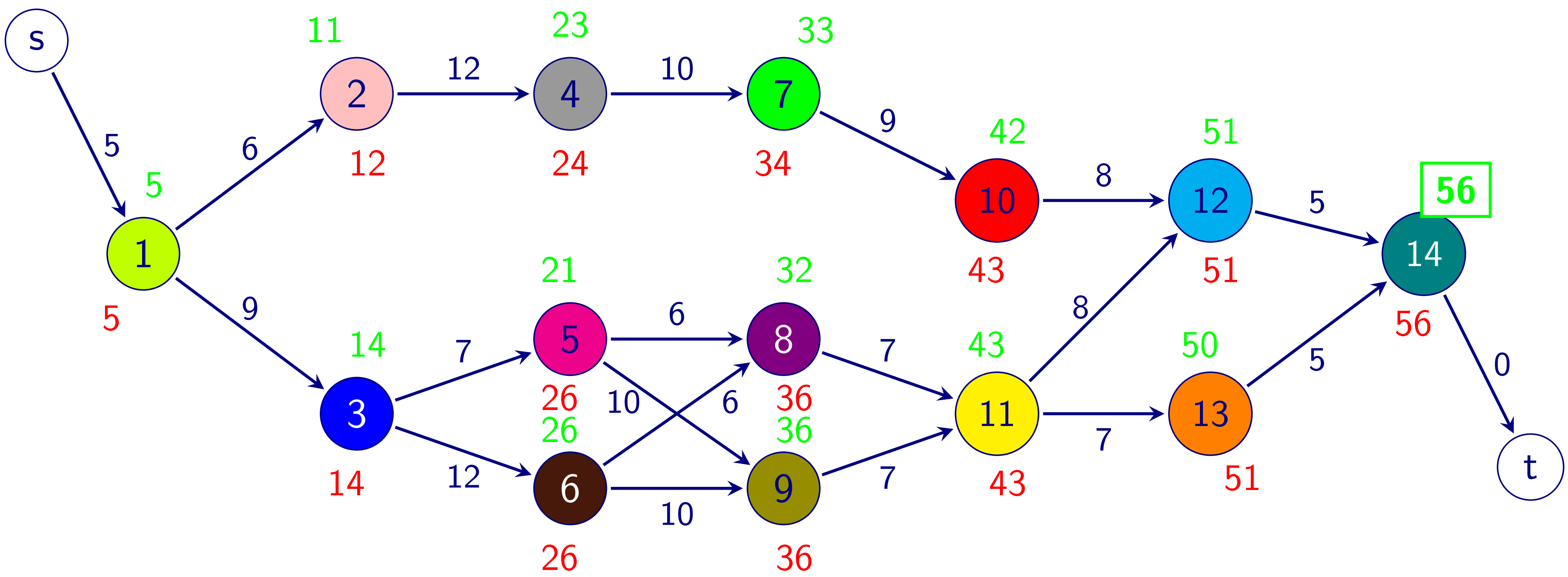
- ▷ Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute latest possible completion times for all jobs



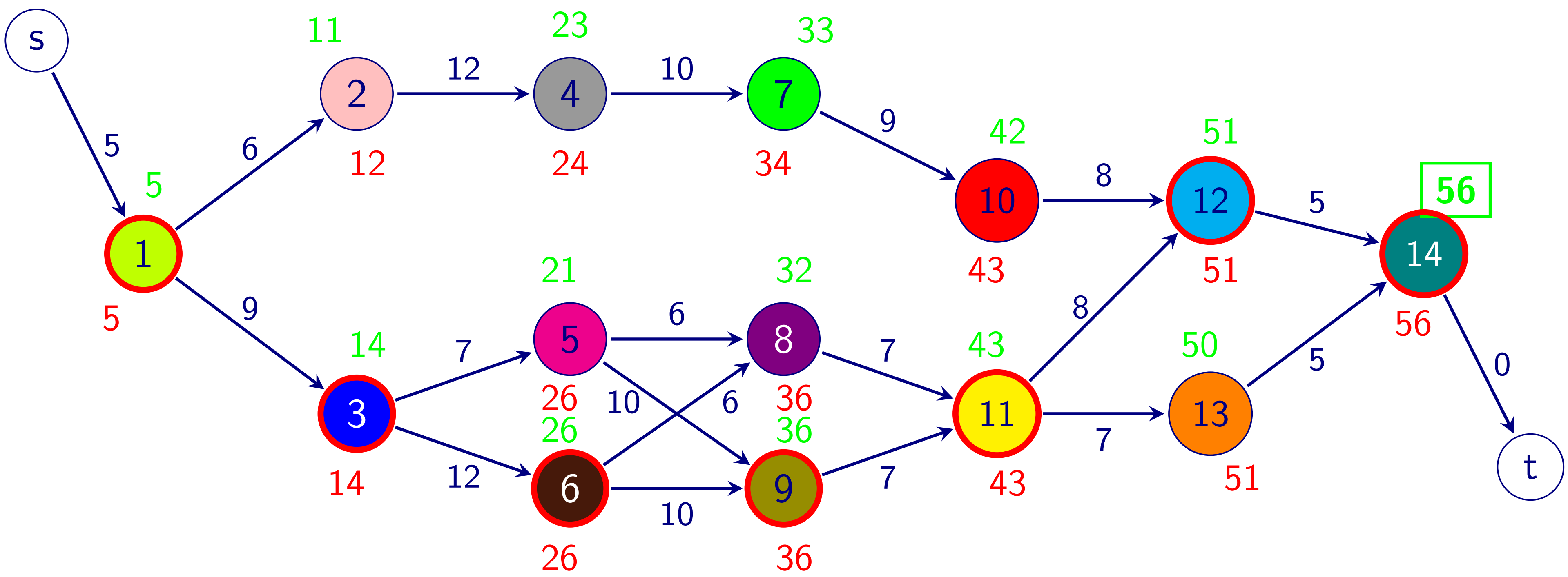
- ▷ Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute latest possible completion times for all jobs



- ▷ Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute latest possible completion times for all jobs
  - ➡ Jobs with earliest = latest possible completion time are the critical jobs

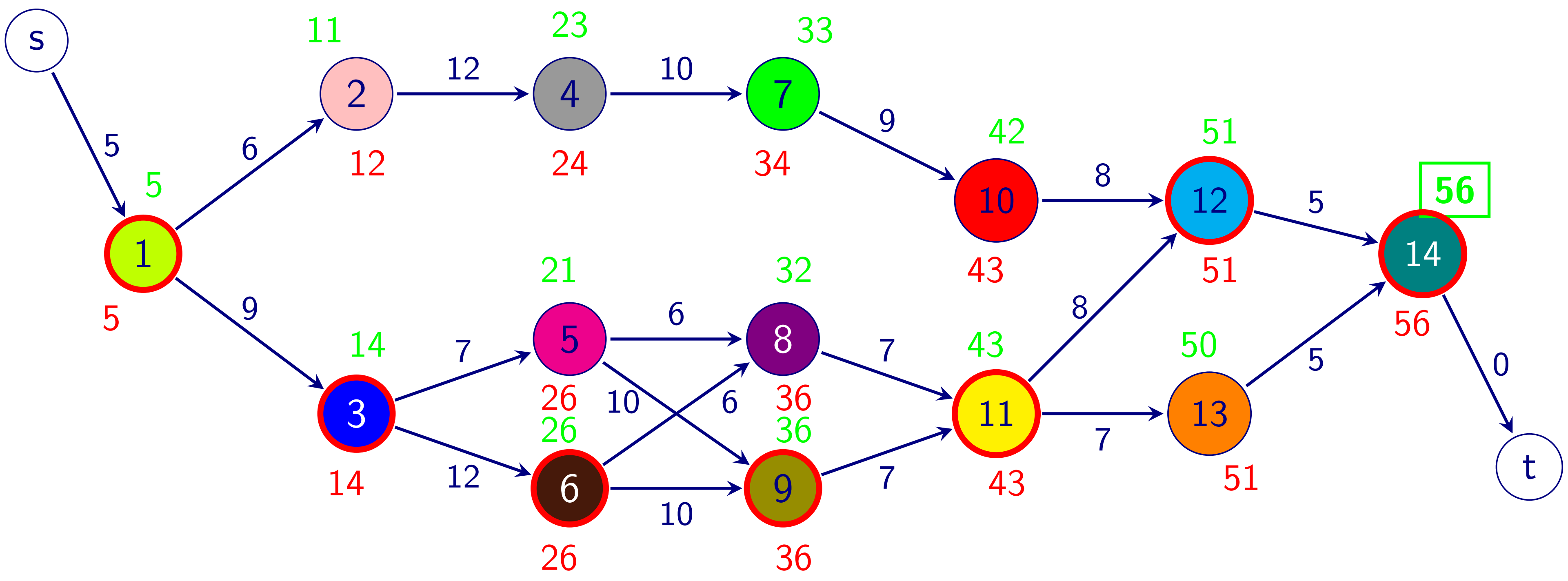


- Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- Backward procedure: compute latest possible completion times for all jobs
  - ➡ Jobs with earliest = latest possible completion time are the critical jobs

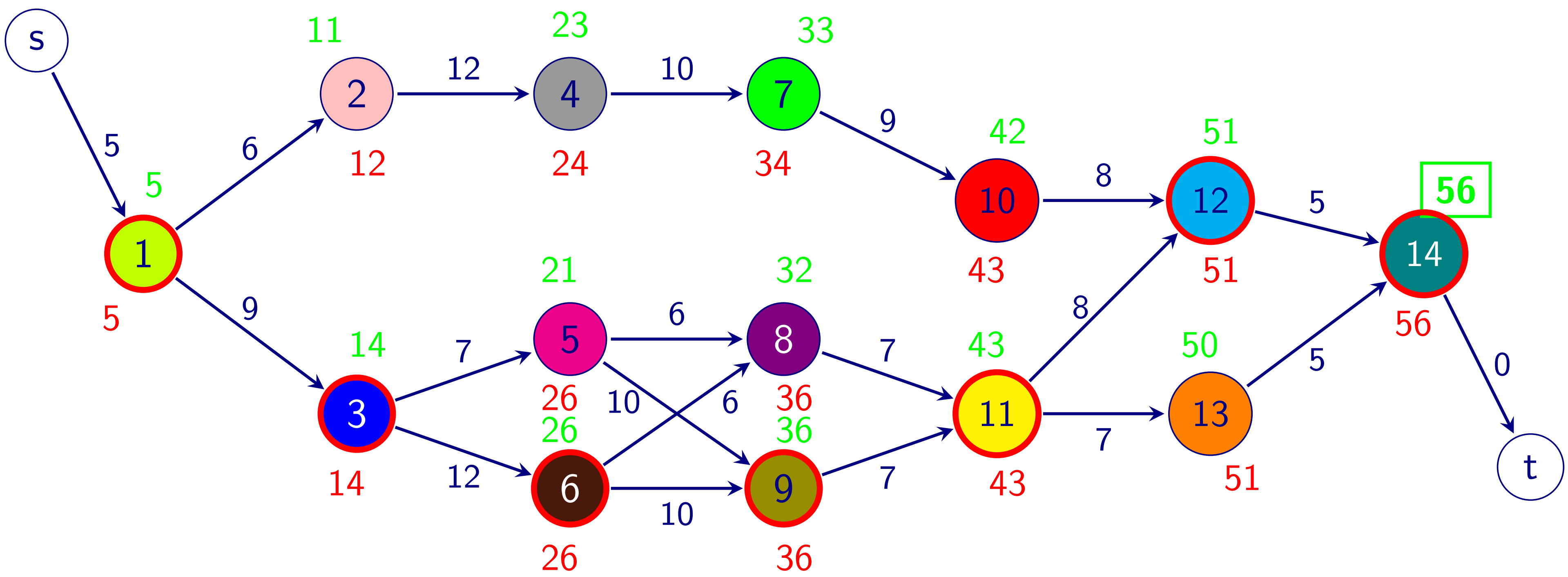


- ▷ Forward procedure: compute earliest possible completion times for all jobs
  - ➡ Makespan is the maximal earliest possible completion time computed
- ▷ Backward procedure: compute latest possible completion times for all jobs
  - ➡ Jobs with earliest = latest possible completion time are the critical jobs

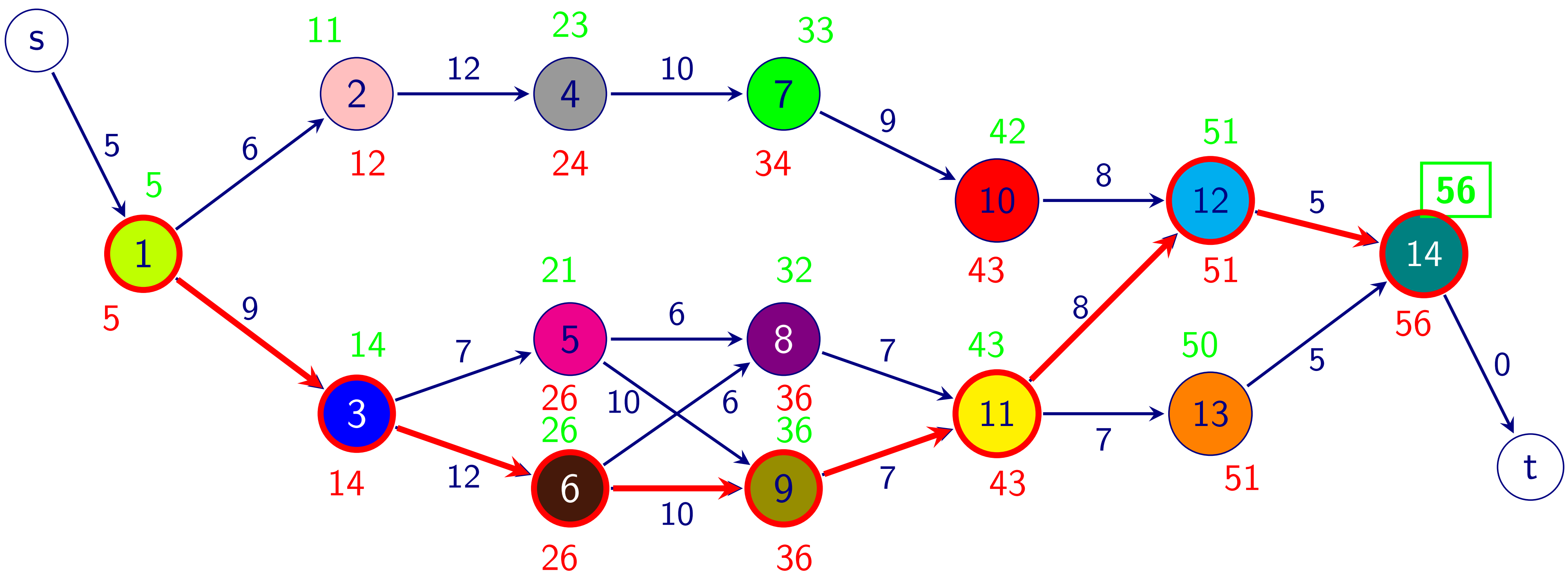
polynomial!



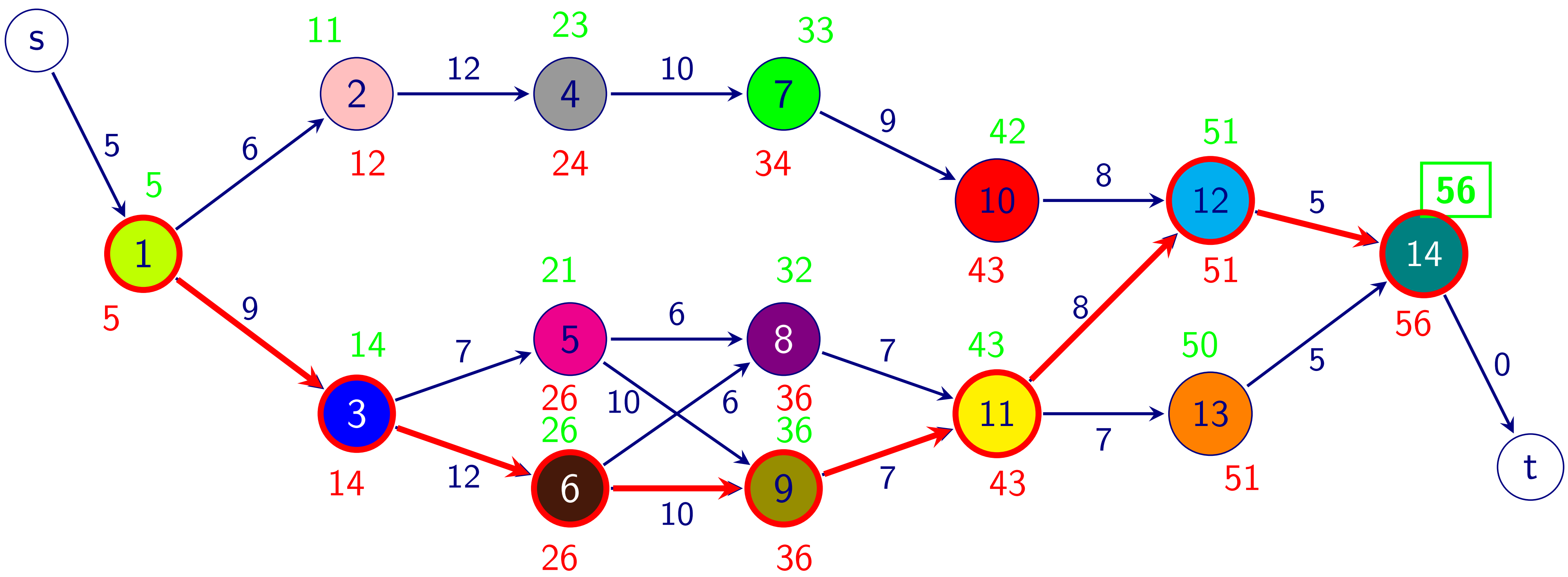
▷ A critical path is a chain of critical jobs, starting at time 0 and ending at the makespan



- A critical path is a chain of critical jobs, starting at time 0 and ending at the makespan



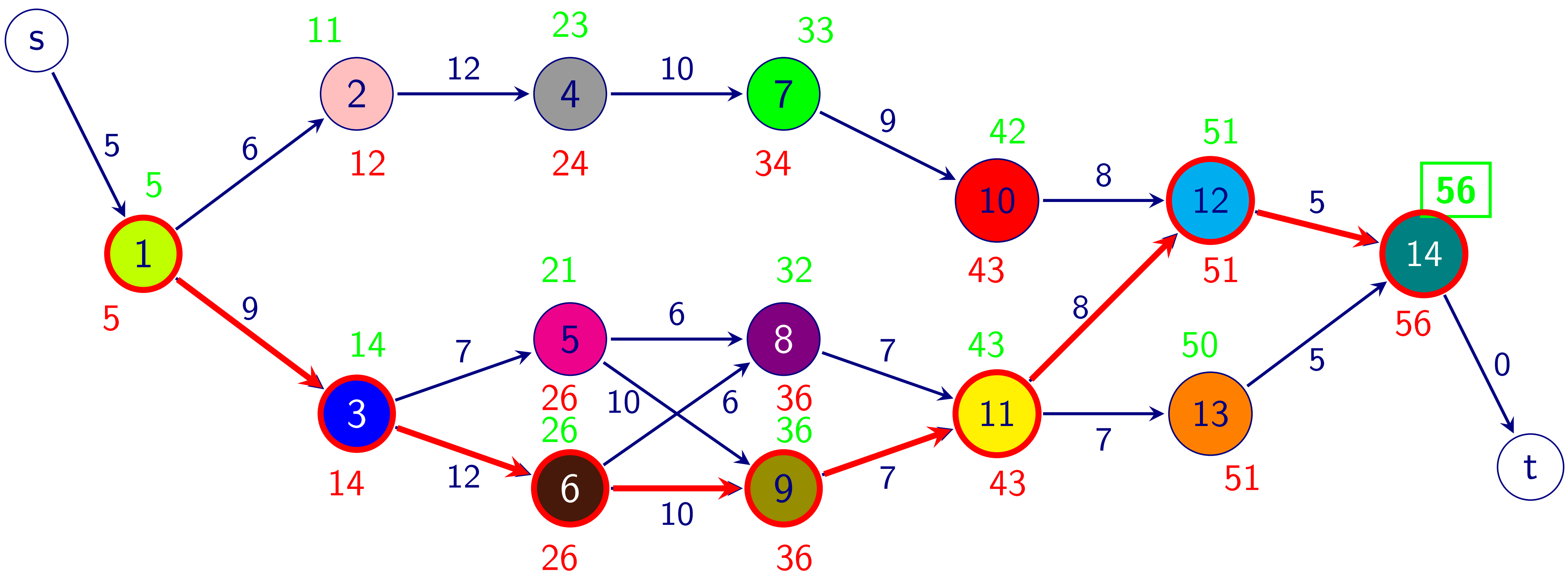
- A critical path is a chain of critical jobs, starting at time 0 and ending at the makespan
- Critical path is a longest path in the precedence graph from s to t



- A critical path is a chain of critical jobs, starting at time 0 and ending at the makespan
- Critical path is a longest path in the precedence graph from s to t

➡ Can be computed by a shortest path algorithm!

(using negative arc lengths – allowed since there are no cycles in the precedence graph)

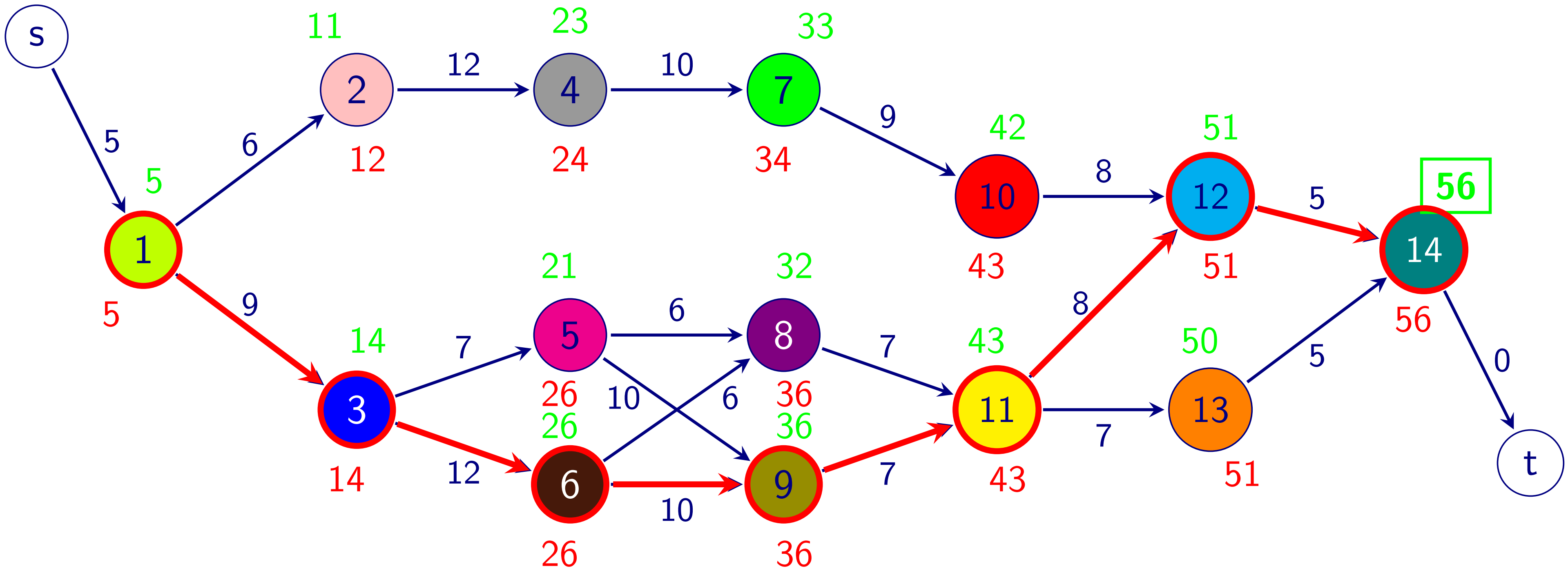


Critical Path

- ▷ A critical path is a chain of critical jobs, starting at time 0 and ending at the makespan
- ▷ Critical path is a longest path in the precedence graph from s to t

➡ Can be computed by a shortest path algorithm!

(using negative arc lengths – allowed since there are no cycles in the precedence graph)



- Project scheduling: minimize makespan with single machine

- ▷ Project scheduling: minimize makespan with single machine
  - ➡ Efficiently solvable (by greedy algorithm)

- Project scheduling: minimize makespan with single machine
  - ➡ Efficiently solvable (by greedy algorithm)
- Minimize sum of completion times

- ▷ Project scheduling: minimize makespan with single machine
  - ➡ Efficiently solvable (by greedy algorithm)
- ▷ Minimize sum of completion times is more difficult...

- Project scheduling: minimize makespan with single machine
  - ➡ Efficiently solvable (by greedy algorithm)
- Minimize sum of completion times is more difficult...
- More than one parallel machines

- ▷ Project scheduling: minimize makespan with single machine
  - ➡ Efficiently solvable (by greedy algorithm)
- ▷ Minimize sum of completion times is more difficult...
- ▷ More than one parallel machines is also difficult...

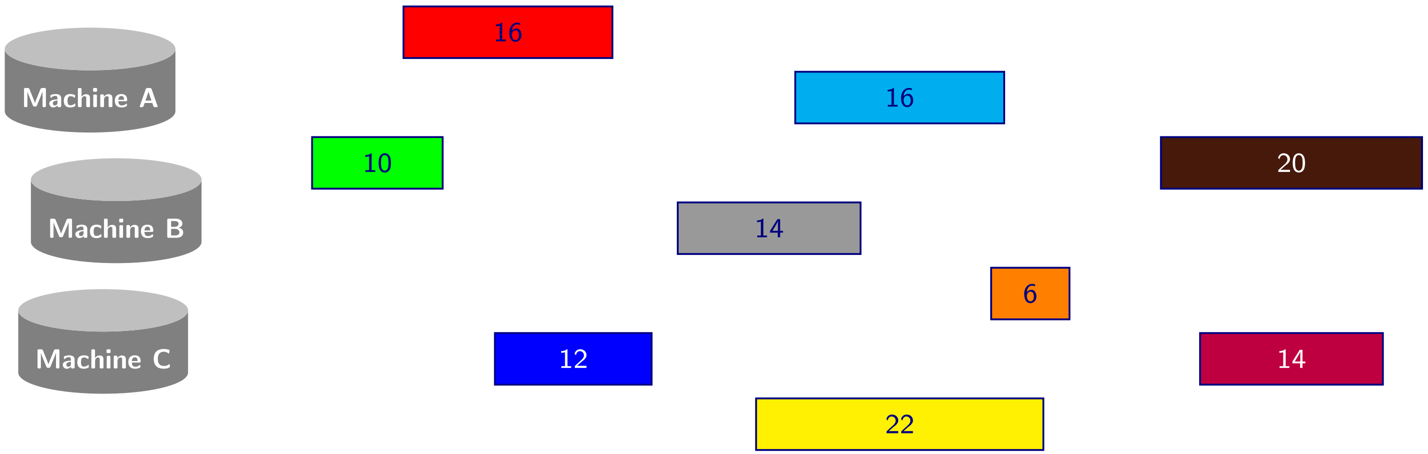
- Project scheduling: minimize makespan with single machine
  - ➡ Efficiently solvable (by greedy algorithm)
- Minimize sum of completion times is more difficult...
- More than one parallel machines is also difficult...
- Except: unlimited number of machines

- Project scheduling: minimize makespan with single machine
  - ➡ Efficiently solvable (by greedy algorithm)
- Minimize sum of completion times is more difficult...
- More than one parallel machines is also difficult...
- Except: unlimited number of machines
  - ➡ Critical Path Method

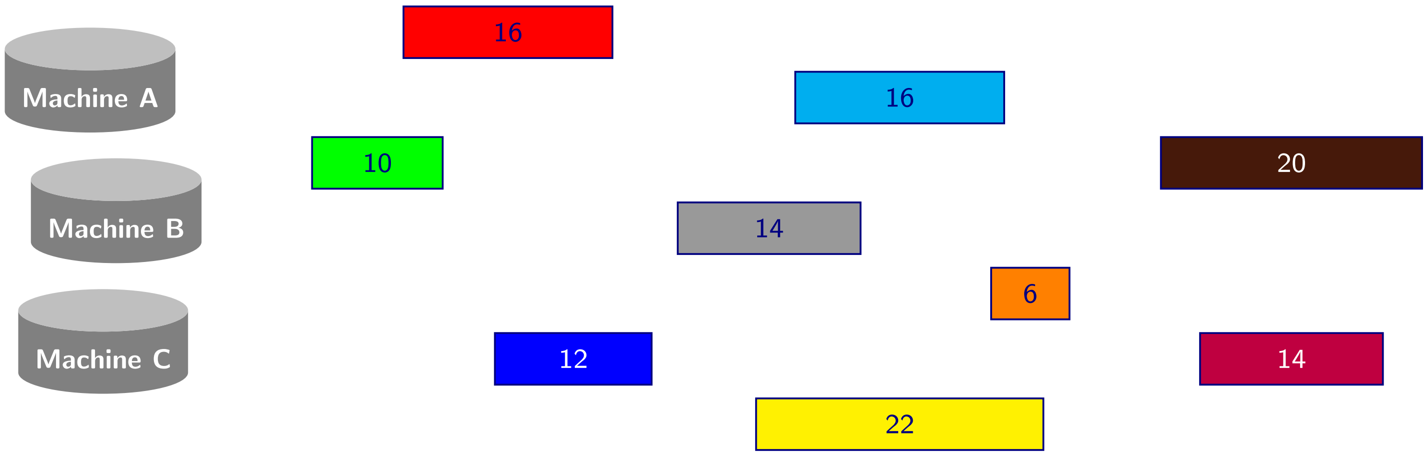
- Project scheduling: minimize makespan with single machine
  - ➡ Efficiently solvable (by greedy algorithm)
- Minimize sum of completion times is more difficult...
- More than one parallel machines is also difficult...
- Except: unlimited number of machines
  - ➡ Critical Path Method
- Summary:

	Objective	
	$\sum C_j$	$\max C_j$
single machine	$\mathcal{NP}$-hard	polynomial (greedy algorithm)
$\geq$ 2 machines	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard
unlimited machines	...?	polynomial (critical path method)

- Jobs can be carried out on one of 3 identical machines

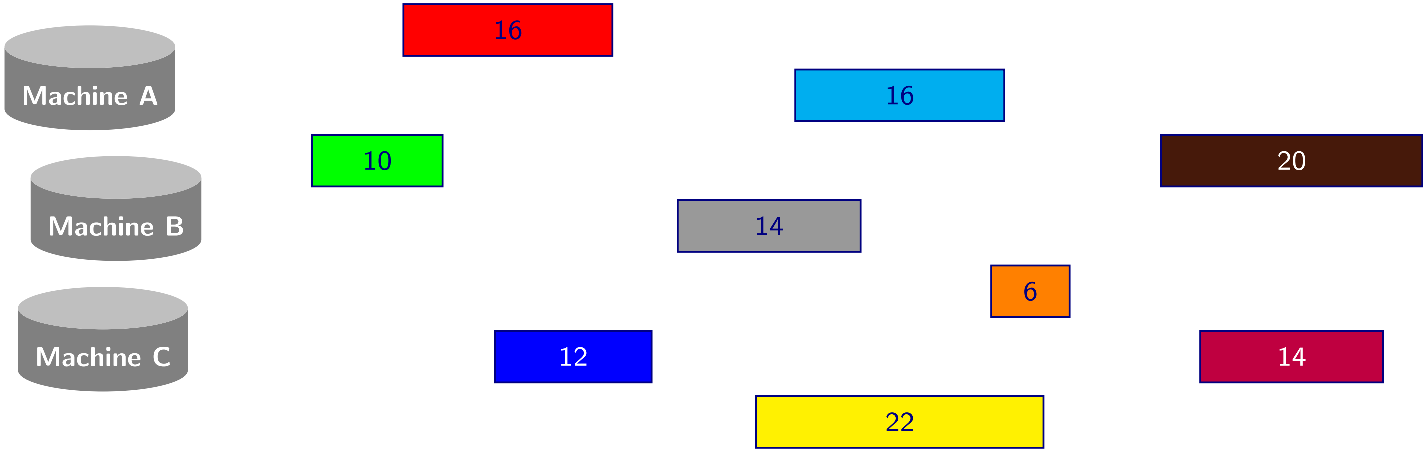


- Jobs can be carried out on one of 3 identical machines

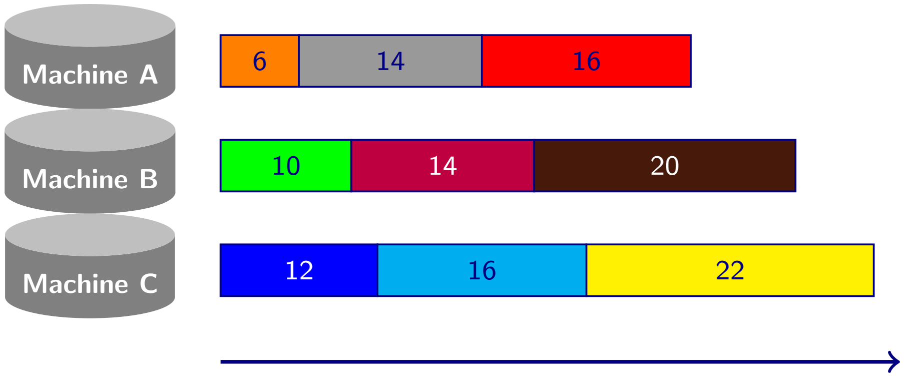


- Minimize sum of completion times

▷ Jobs can be carried out on one of 3 identical machines

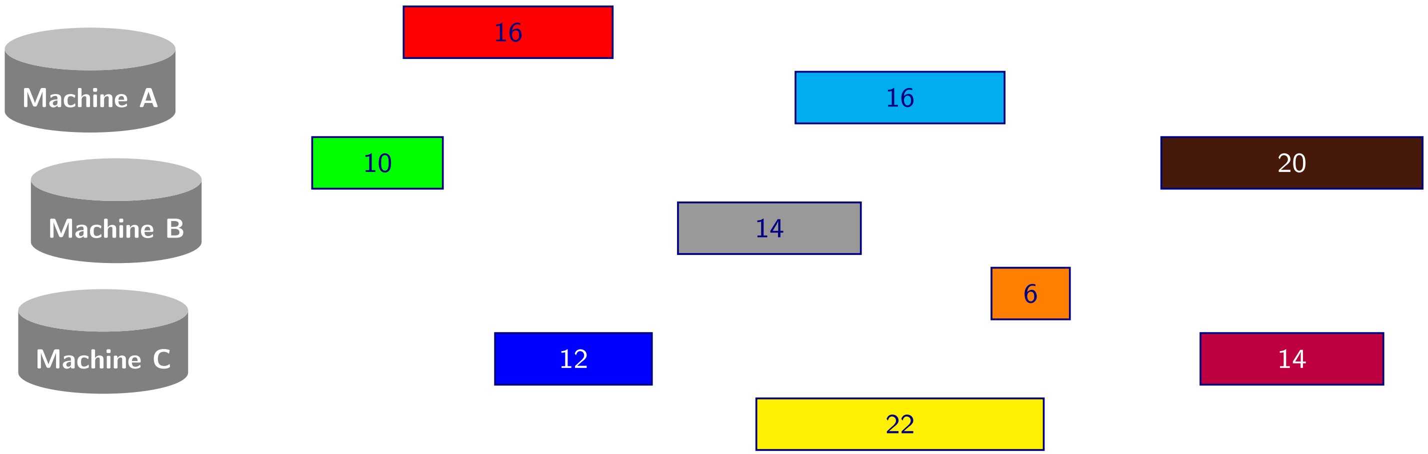


▷ Minimize sum of completion times

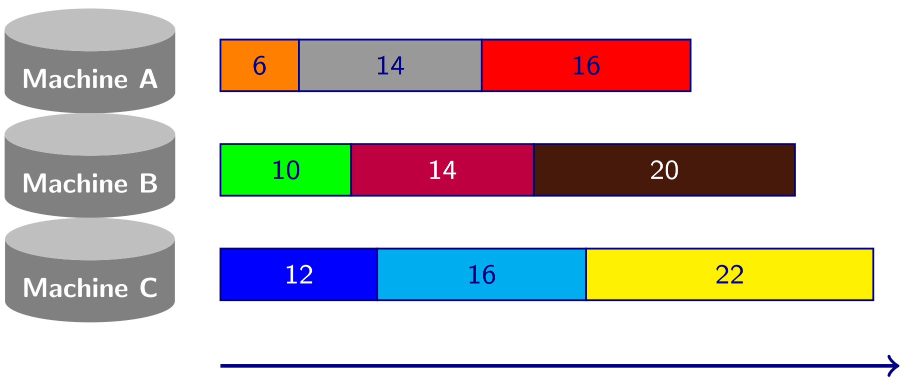


Scheduling on 3 identical machines

- Jobs can be carried out on one of 3 identical machines



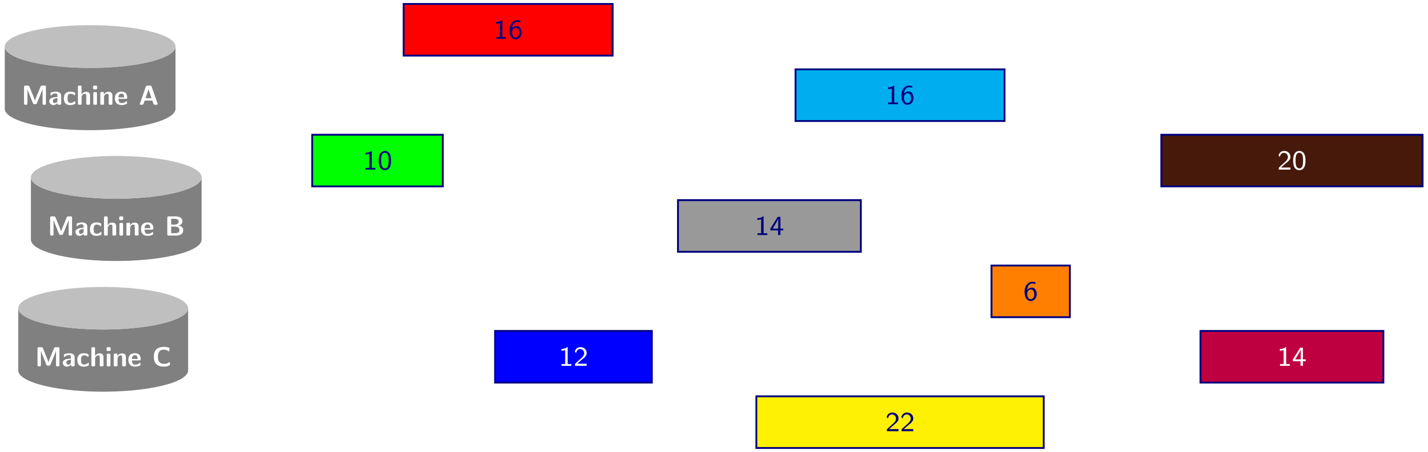
- Minimize sum of completion times



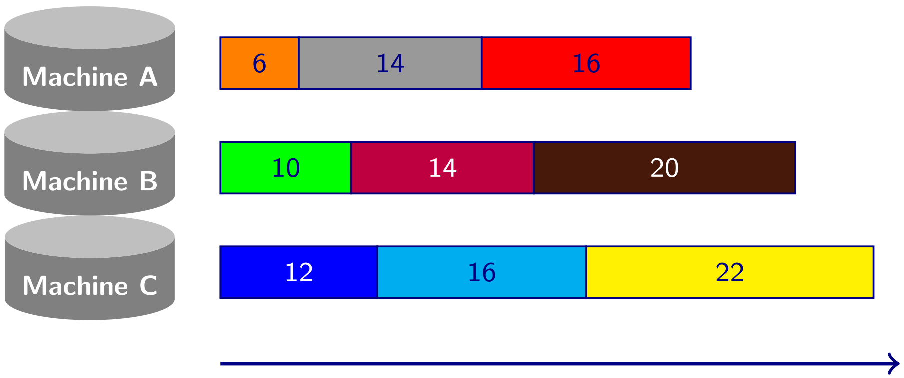
$$
\begin{aligned}
\sum_{j=1}^{n} C_j &= 6 + 20 + 36 \\
&= 10 + 24 + 44 \\
&= 12 + 28 + 50 \\
&= 230
\end{aligned}
$$

Scheduling on 3 identical machines

▷ Jobs can be carried out on one of 3 identical machines



▷ Minimize sum of completion times



$$\sum_{j=1}^{n} C_j = 6 + 20 + 36 = 10 + 24 + 44 = 12 + 28 + 50 = 230$$

➡ Optimal: schedule by non-decreasing processing times, on earliest available machine

- Minimize sum of completion times: polynomial (greedy algorithm)

- Minimize sum of completion times: polynomial (greedy algorithm)
- Minimize makespan: $\mathcal{NP}$-hard

- ▷ Minimize sum of completion times: polynomial (greedy algorithm)
- ▷ Minimize makespan: $\mathcal{NP}$-hard
- ▷ Variants: types of machines
  - Identical machines

- ▷ Minimize sum of completion times: polynomial (greedy algorithm)
- ▷ Minimize makespan: $\mathcal{NP}$-hard
- ▷ Variants: types of machines
  - Identical machines
  - Uniform machines: machines differ by a fixed speed factor

- ▷ Minimize sum of completion times: polynomial (greedy algorithm)
- ▷ Minimize makespan: $\mathcal{NP}$-hard
- ▷ Variants: types of machines
  - Identical machines
  - Uniform machines: machines differ by a fixed speed factor
  - Unrelated machines: processing times differ for every job on each machine

- Minimize sum of completion times: polynomial (greedy algorithm)
- Minimize makespan: $\mathcal{NP}$-hard
- Variants: types of machines
  - Identical machines
  - Uniform machines: machines differ by a fixed speed factor
  - Unrelated machines: processing times differ for every job on each machine
- All the other additional features: weights, release dates, precedence constraints, ...

- ▷ Minimize sum of completion times: polynomial (greedy algorithm)
- ▷ Minimize makespan: $\mathcal{NP}$-hard
- ▷ Variants: types of machines
  - Identical machines
  - Uniform machines: machines differ by a fixed speed factor
  - Unrelated machines: processing times differ for every job on each machine
- ▷ All the other additional features: weights, release dates, precedence constraints, ...
- ▷ Multi-operation models: job has to be processed sequentially on multiple machines

- Minimize sum of completion times: polynomial (greedy algorithm)
- Minimize makespan: $\mathcal{NP}$-hard
- Variants: types of machines
  - Identical machines
  - Uniform machines: machines differ by a fixed speed factor
  - Unrelated machines: processing times differ for every job on each machine
- All the other additional features: weights, release dates, precedence constraints, ...
- Multi-operation models: job has to be processed sequentially on multiple machines
  - Open shop: order in which jobs pass through machines is unimportant

- Minimize sum of completion times: polynomial (greedy algorithm)
- Minimize makespan: $\mathcal{NP}$-hard
- Variants: types of machines
  - Identical machines
  - Uniform machines: machines differ by a fixed speed factor
  - Unrelated machines: processing times differ for every job on each machine
- All the other additional features: weights, release dates, precedence constraints, ...
- Multi-operation models: job has to be processed sequentially on multiple machines
  - Open shop: order in which jobs pass through machines is unimportant
  - Flow shop: each job has the same machine order (A, B, ...)

- ▷ Minimize sum of completion times: polynomial (greedy algorithm)
- ▷ Minimize makespan: $\mathcal{NP}$-hard
- ▷ Variants: types of machines
  - Identical machines
  - Uniform machines: machines differ by a fixed speed factor
  - Unrelated machines: processing times differ for every job on each machine
- ▷ All the other additional features: weights, release dates, precedence constraints, ...
- ▷ Multi-operation models: job has to be processed sequentially on multiple machines
  - Open shop: order in which jobs pass through machines is unimportant
  - Flow shop: each job has the same machine order (A, B, ...)
  - Job shop: each job can have a different machine order

- Minimize sum of completion times: polynomial (greedy algorithm)
- Minimize makespan: $\mathcal{NP}$-hard
- Variants: types of machines
  - Identical machines
  - Uniform machines: machines differ by a fixed speed factor
  - Unrelated machines: processing times differ for every job on each machine
- All the other additional features: weights, release dates, precedence constraints, ...
- Multi-operation models: job has to be processed sequentially on multiple machines
  - Open shop: order in which jobs pass through machines is unimportant
  - Flow shop: each job has the same machine order (A, B, ...)
  - Job shop: each job can have a different machine order
    - ➡ Makespan minimization for job shop scheduling can also be solved using networks

- Models, Data and Algorithms
- Linear Optimization
- Mathematical Background: Polyhedra, Simplex-Algorithm
- Sensitivity Analysis; (Mixed) Integer Programming
- MIP Modelling
- MIP Modelling: More Examples; Branch & Bound
- Cutting Planes; Combinatorial Optimization: Examples, Graphs, Algorithms
- TSP-Heuristics
- Network Flows
- Shortest Path Problem
- Complexity Theory
- Nonlinear Optimization
- Scheduling
- Lot Sizing & Intro to Multiobjective Optimization (Feb 01)
- Summary (Feb 08)
- Oral exam (Feb 15)