Scheduling Algorithms

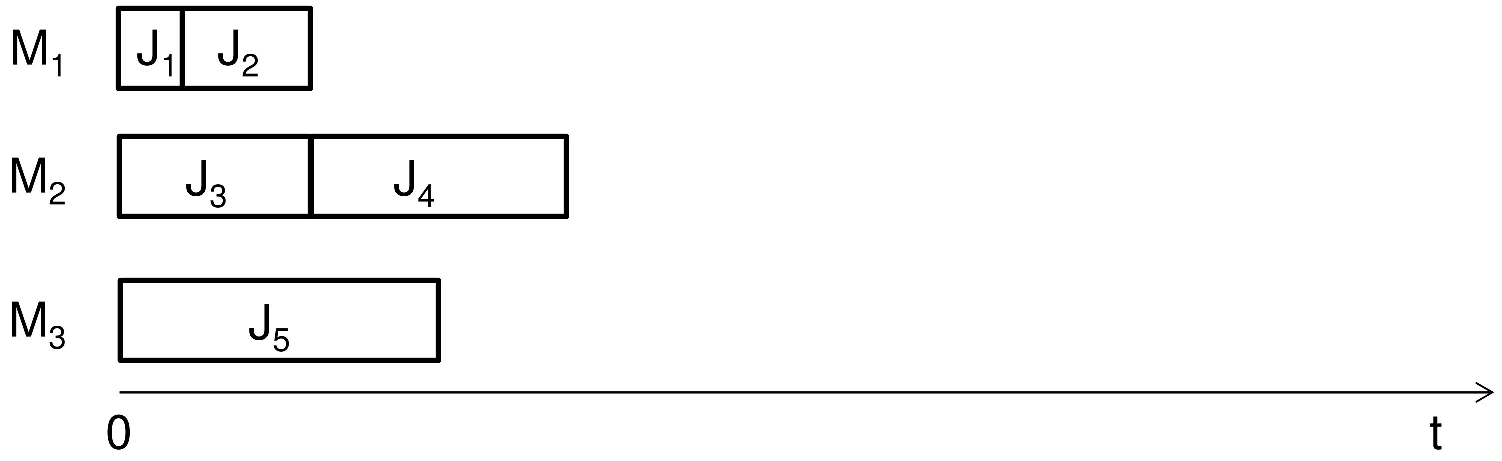
(Part III)

Scheduling Identical Job SUM (minimization) problem: an example. (3)

- INPUT: 5 jobs, 3 machines.

p_i=

J_1	J_2	J_3	J_4	J_5
1	2	3	4	5



$JC_1(S)=1$; $JC_2(S)=1+2=3$; $JC_3(S)=3$; $JC_4(S)=3+4=7$; $JC_5(S)=5$.

$$\sum_{i=1}^{5} JC_i(S) = 1+3+3+7+5 = 19$$

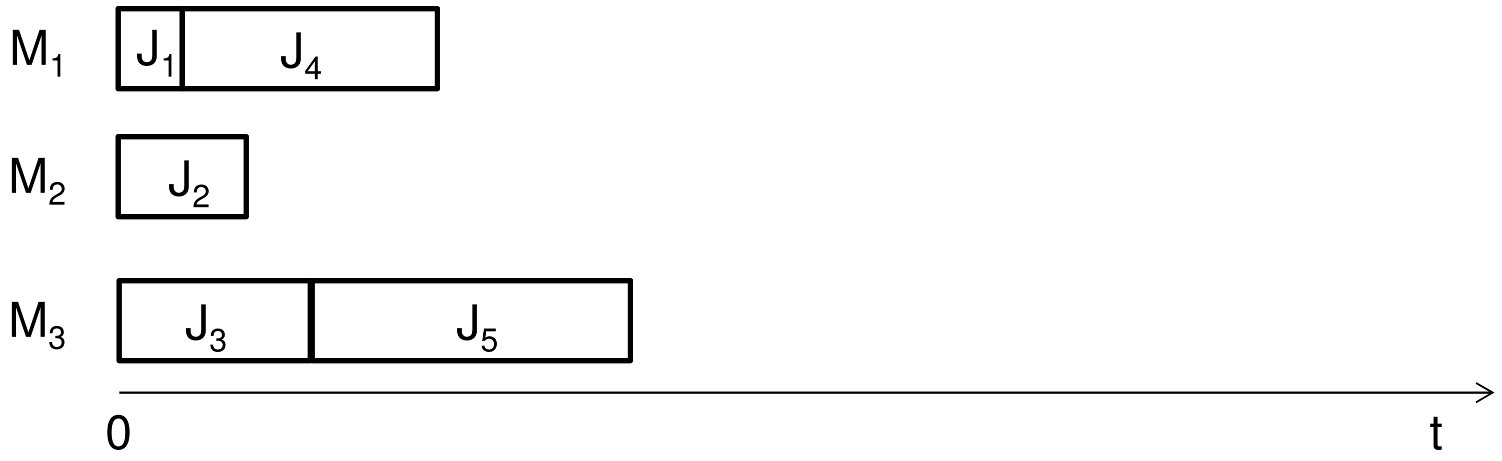
Is it optimal?

Scheduling Identical Job SUM (minimization) problem: an example. (4)

- INPUT: 5 jobs, 3 machines.

p_i=

J_1	J_2	J_3	J_4	J_5
1	2	3	4	5



$JC_1(S)=1$; $JC_4(S)=1+4=5$; $JC_2(S)=2$; $JC_3(S)=3$; $JC_5(S)=3+5=8$;

$$\sum_{i=1}^{5} JC_i(S) = 1+2+3+5+8=19$$

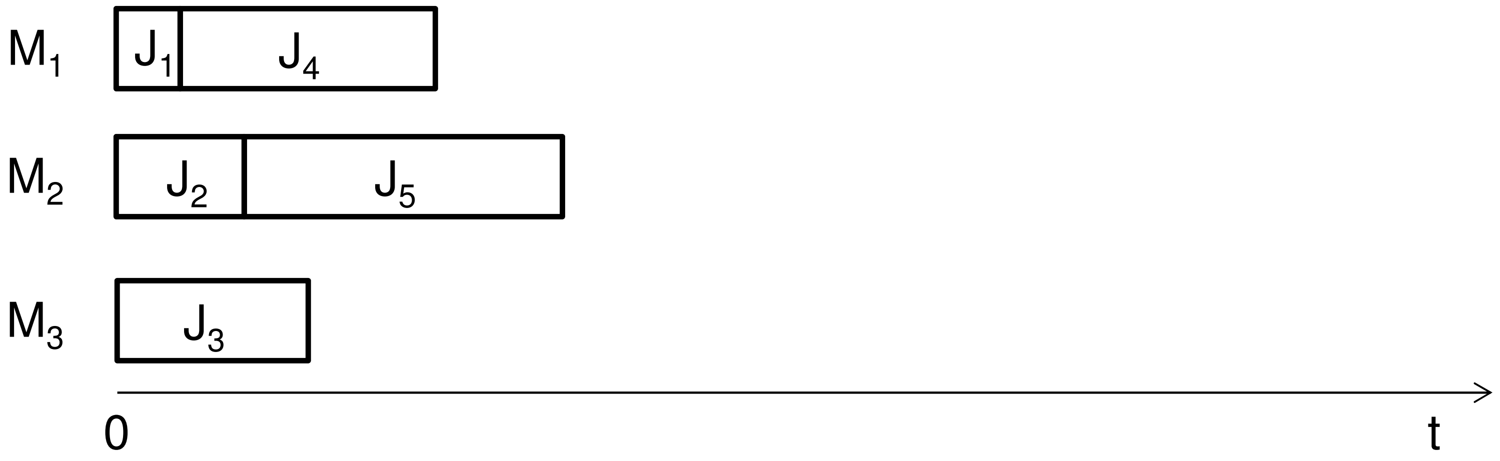
Is it optimal?

Scheduling Identical Job SUM (minimization) problem: an example. (5)

- INPUT: 5 jobs, 3 machines.

p_i=

J_1	J_2	J_3	J_4	J_5
1	2	3	4	5



$JC_1(S)=1$; $JC_4(S)=1+4=5$; $JC_2(S)=2$; $JC_5(S)=2+5=7$; $JC_3(S)=3$.

$$\sum_{i=1}^{5} JC_i(S) = 1+2+3+5+7=18$$

Is it optimal?

Scheduling Identical Job SUM (minimization) problem: a simple algorithm.

- INPUT: *m identical* machine (*h = 1, . . .,m*), *n* jobs (*i = 1, . . . , n*), $p_i > 0$.
- OUTPUT: a schedule ***S***=($S_{1,\dots,}S_m$) *where* $S_h=(S_{h,1},\dots,S_{h,k_h})$.
- GOAL*:* Minimizing the Job SUM, that is minimizing

$$\sum_{i=1}^{n} JC_i(S)$$

ALGORITHM JSUM:

```
1.  Arrange jobs in non-decreasing order of $p_i$; such that $p_1 \le p_2 \le p_3 \le \cdots \le p_n$. That is rename jobs such
    that job 1 has the smallest processing time, job 2 has the second smallest processing time and so on.
    Moreover consider machines in any order (in fact remind that we are dealing with identical machines
    and therefore the order does not matter).
2.  For i=1 to n
        Assign job i to machine i mod m.   (When i mod m=0, then assign job i to machine m)
    End for
3.  Return the obtained schedule S.
```

COMPLEXITY: *O(n∗log(n))* (A simple sort can be done in O(*n∗log(n))* time, and assigning jobs to machines can be done in time *n*).

Theorem 4: *Algorithm JSUM finds an optimal solution for the Scheduling Identical Job SUM (minimization) problem.*

Proof of Theorem 4.

Proof:

- In the proof of Theorem 2 (where *m=1*) we have seen the following:

Given a schedule *S*, let $p_{(j)}(S)$ be the processing time of the job in the *j*th position in the schedule *S* (i.e., $p_{(j)}(S)$ is the processing time of the job $S_{1,j}$). By the definition of job completion time, it follows that:

$$\sum_{i=1}^{n} JC_i(S) = n * p_{(1)}(S) + (n-1) * p_{(2)}(S) + \ldots + 2 * p_{(n-1)}(S) + p_{(n)}(S)$$

- In the case of *m* parallel machines there are *n*m* possible coefficients to which processing times can be assigned. Specifically these coefficients are *m n*'s, *m (n−1)*'s, …….., *m* ones (see next slides).
- The processing times have to be assigned to a subset of these coefficients in order to minimize the sum of the products. We assume that *n* is a multiple of *m* (i.e., *n/m=q* and *n mod m=0*). If it is not an integer add a number of dummy jobs with zero processing times so that *n* is a multiple of *m* (adding jobs with zero processing times does not change the problem; these jobs would be instantaneously processed at time zero and would not contribute to the objective function).

Proof of Theorem 4. (2)

$$\left.\begin{array}{ccccccc} \text{m n's} & & \text{m (n–1)'s} & & \text{m 2's} & & \text{m ones} \\ n * p_{(1)}(S) & + & (n-1) * p_{(2)}(S) & + \ldots + & 2 * p_{(n-1)}(S) & + & p_{(n)}(S) \\ n * p_{(1)}(S) & + & (n-1) * p_{(2)}(S) & + \ldots + & 2 * p_{(n-1)}(S) & + & p_{(n)}(S) \\ n * p_{(1)}(S) & + & (n-1) * p_{(2)}(S) & + \ldots + & 2 * p_{(n-1)}(S) & + & p_{(n)}(S) \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \\ n * p_{(1)}(S) & + & (n-1) * p_{(2)}(S) & + \ldots + & 2 * p_{(n-1)}(S) & + & p_{(n)}(S) \end{array}\right\} m$$

The processing times have to be assigned to a subset of these coefficients (n overall since we have to assign n jobs) in order to minimize the sum of the products.

Proof of Theorem 4. (3)

- The sum of the completion time in the scheduling returned by ALGORITHM JSUM has n additive terms with one coefficient each:

m coefficients with value q
m coefficients with value q–1
:
:
m coefficients with value 1

- Suppose, by contradiction, that there exists an optimal scheduling *S'* where there is at least one coefficient with value $x>q$ (that is one machine gets more than q jobs).
- It implies that must exist another machine *h* with maximum coefficient $y<q$ (the maximum coefficient is associated with the first job assigned to machine *h*).
- However, if we replace *x* with $y+1$ (by assigning the job with coefficient *x* as first job to the machine *h*) we obtain another scheduling with strictly smaller Job SUM (notice that $y+1<x$).
- Therefore *S'* is not optimal.

Proof of Theorem 4. (4)

- Therefore, in a similar manner for the case m=1, the set of *m* shortest processing times have to be assigned to the *q* ones, the set of second *m* shortest processing times have to be assigned to the *q–1 ones*, and so on.

- Thus, if we assign the smallest job to machine *1* at time zero, the second smallest one to machine *2*, and so on; The (m+1)th smallest job follows the smallest job on machine 1, the (m+2)th smallest job follows the second smallest on machine 2, and so on, we get it.

- The algorithm JSUM uses such order to assign jobs to the machine. Thus the returned schedule is optimal. □

Considerations.

The Scheduling <u>Unrelated</u> Job SUM (minimization) problem also admits a polynomial time algorithm that finds a schedule that minimizes the Job SUM.

However it is a bit more complicated and we are not going to study it in this course.

Considerations. (2)

- In literature, the Scheduling Identical Job SUM (minimization) problem with weights and m>1 has been proved to be an NP-Hard problem.

- It means that, unless P=NP, the Scheduling Identical Job SUM (minimization) problem with weights and m>1 does not admit a polynomial time algorithm finding a schedule that minimizes the (weighted) Job SUM.

Scheduling Identical Machine MAKESPAN (minimization) problem.

- INPUT: *m identical* machine (*h = 1, . . .,m*), *n* jobs (*i = 1, . . . , n*), $p_i > 0$.
- OUTPUT: a schedule ***S=****(S*$_{1,\dots,}$*S*$_m$*).*
- GOAL*:* Minimizing the Machine MAKESPAN, that is minimizing $Max_{h=1,\dots,m}\{MC_h(S)\}$

Does such problem admit a polynomial time algorithm that finds a schedule that minimizes the Machine MAKESPAN?

This problem is trivial when *m=1*, in fact remind that the Machine Completion time does not depend on the order by which jobs are assigned to the machine. Thus for *m=1* any schedule is an optimal schedule.

If *m≥2* the problem has been shown to be NP-Hard. It means that (unless P=NP) the problem does not admit any polynomial time algorithm that finds a schedule that minimizes the Machine MAKESPAN.

Approximation algorithms.

- Therefore, we are going to design a polynomial time algorithm that finds good (or approximated) solutions for our problem.

We now roughly define what is an approximation algorithm for our problem:

- Let us suppose we have designed a polynomial time* algorithm that finds a schedule. Let *SOL* be the solution (schedule) returned by our algorithm and let *OPT* be an optimal solution to the problem.
- Moreover let C_{max}(*SOL)* (respectively C_{max}(*OPT)*) be the Machine MAKESPAN of the schedule *SOL* (respectively of the optimal schedule *OPT*).
- That is $C_{\max}(SOL) = Max_{h=1,\dots,m}\{MC_h(SOL)\}$ and $C_{\max}(OPT) = Max_{h=1,\dots,m}\{MC_h(OPT)\}$
- We say that our algorithm is an *r-approximation* algorithm (where *r>1*) for the problem, if for any instance of the problem it holds that:

$$\frac{C_{\max}(SOL)}{C_{\max}(OPT)} \le r$$

* Notice that the definition of approximation algorithms is not restricted to polynomial time algorithms. However we are only interested in polynomial time algorithms.

Scheduling Identical Machine MAKESPAN (minimization) problem: a polynomial time approximation algorithm.

- INPUT: *m identical* machine ($h = 1, \dots, m$), n jobs ($i = 1, \dots, n$), $p_i > 0$.
- OUTPUT: a schedule ***S*=*(S*_1,…,_S_m)*.** $S=(S_1,\dots,S_m)$.
- GOAL*:* Minimizing the Machine MAKESPAN, that is minimizing $Max_{h=1,\dots,m}\{MC_h(S)\} = C_{\max}(S)$

DEFINITION: let $T_h(j)$ be the machine completion time of the machine *h* at the time where *j* jobs have been allocated to machines by the algorithm.

ALGORITHM MAKESPAN:

1. Arrange jobs in non-increasing order of p_i; such that $p_1 \geq p_2 \geq p_3 \geq \cdots \geq p_n$. That is rename jobs such that job *1* has the greatest processing time, job *2* has the second greatest processing time and so on. (Notice that at this step the algorithm does not assign any job to machines).
2. For j=1 to n do

   Assigns job *j* to a machine with minimum $T_h(j\text{-}1)$; *(if there are more than one machines with minimum* $T_h(j\text{-}1)$ *choose any one of them).*
3. Return the obtained schedule *S*.

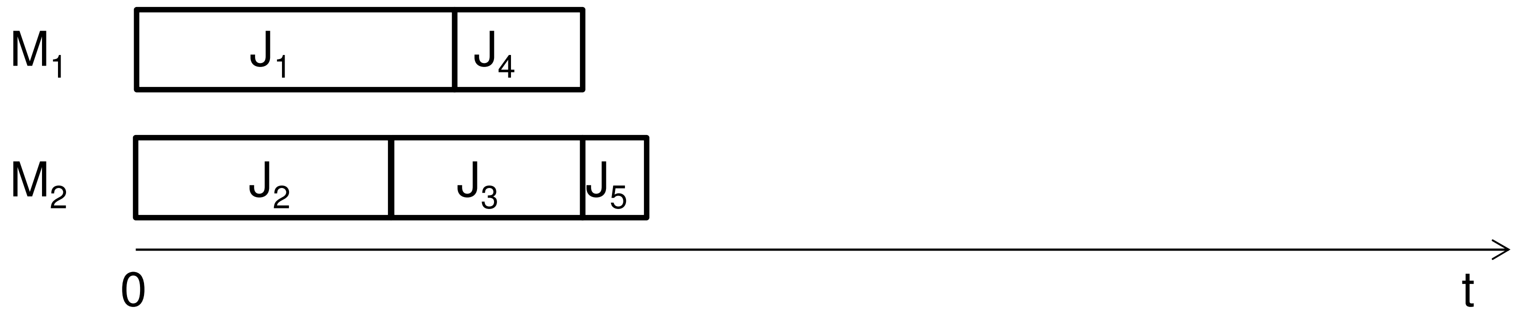
COMPLEXITY: $O(n*\log(n))$ (A simple sort can be done in $O(n*\log(n))$ time, and assigning jobs to machines can be done in time *n*).

Execution of the algorithm MAKESPAN: an example.

- INPUT: 2 identical machines, 5 jobs.

p_i=

J_1	J_2	J_3	J_4	J_5
5	4	3	2	1



1) *$T_1(0)=0$; $T_2(0)=0$; → job 1 is assigned to machine M_1*
2) *$T_1(1)=5$; $T_2(1)=0$; → job 2 is assigned to machine M_2*
3) *$T_1(2)=5$; $T_2(2)=4$; → job 3 is assigned to machine M_2*
4) *$T_1(3)=5$; $T_2(3)=7$; → job 4 is assigned to machine M_1*
5) *$T_1(4)=7$; $T_2(4)=7$; → job 5 is assigned to machine M_2*

Scheduling Identical Machine MAKESPAN (minimization) problem: a polynomial time approximation algorithm. (2)

- INPUT: *m identical* machine (*h = 1, . . .,m*), *n* jobs (*i = 1, . . . , n*), $p_i > 0$.
- OUTPUT: a schedule ***S*=(*S*$_{1,\dots,}$*S*$_m$).**
- GOAL*:* Minimizing the Machine MAKESPAN, that is minimizing

$$Max_{h=1,\dots,m}\{MC_h(S)\} = C_{\max}(S)$$

Theorem 5: *Algorithm MAKESPAN is an* $\frac{4}{3} - \frac{1}{3m}$ *approximation algorithm for the Scheduling Identical Machine MAKESPAN (minimization) problem.*

Proof of Theorem 5.

Proof:

- Let *SOL* be the solution (schedule) returned by our algorithm and let *OPT* be an optimal solution to the problem.
- By contradiction. Assume that there exists one or more counterexamples with the ratio strictly larger than $4/3 - 1/3m$.
- If more than one such counterexample exist, there must exist an example with the smallest number of jobs.
- Consider this "smallest" counterexample and assume it has *n* jobs.
- This smallest counterexample has useful properties:
  - By the definition of the algorithm MAKESPAN, the shortest job is the last to start its processing in the solution returned by algorithm MAKESPAN.
  - The shortest job is the last to complete its processing, in fact otherwise the deletion of this shortest job will result in a counterexample with fewer jobs (in fact the $C_{max}(SOL)$ remains the same while the $C_{max}(OPT)$ may remain the same or may decrease).

Proof of Theorem 5. (2)

- Therefore for the smallest counterexample the starting time of the shortest job under algorithm MAKESPAN is $C_{max}(SOL)-p_n$.
- Since at this point in time all other machines are still busy it follows that

$$C_{\max}(SOL) - p_n \le \frac{\sum_{j=1}^{n-1} p_j}{m}$$

- The right hand side is an upper bound on the starting time of the shortest job. This upper bound is achieved when scheduling the first $n-1$ jobs according to algorithm MAKESPAN results in each machine having exactly the same amount of processing to do.

Proof of Theorem 5. (3)

- Now we make some calculations.
- From the previous slide we know that: $C_{\max}(SOL) - p_n \le \dfrac{\sum_{j=1}^{n-1} p_j}{m}$

$$C_{\max}(SOL) \le p_n + \frac{\sum_{j=1}^{n-1} p_j}{m} = p_n\left(1 - \frac{1}{m}\right) + \frac{\sum_{j=1}^{n} p_j}{m}$$

Proof of Theorem 5. (4)

- Moreover, we have the following trivial lower bound of an optimal solution:

$$C_{\max}(OPT) \geq \frac{\sum_{j=1}^{n} p_j}{m}$$

- In fact the following general property holds:

Claim: Given *s≥0* and *m* numbers $a_1,\ldots,a_m$ such that $a_1+ a_2+ a_3+\ldots+ a_m= s$, <u>there exists</u> j, 1≤j≤m, such that: $a_j \geq \frac{s}{m}$

Otherwise, $a_1 + a_2 + a_3 + \ldots + a_m < m \cdot \frac{s}{m} = s$ $\Longrightarrow$ Absurd! □

Proof of Theorem 5. (5)

- Therefore the following series of inequalities holds for the counterexample:

$$\frac{4}{3}-\frac{1}{3m} < \frac{C_{\max}(SOL)}{C_{\max}(OPT)} \le \frac{p_n(1-1/m)+\sum_{j=1}^{n} p_j/m}{C_{\max}(OPT)} =$$

$$= \frac{p_n(1-1/m)}{C_{\max}(OPT)} + \frac{\sum_{j=1}^{n} p_j/m}{C_{\max}(OPT)} \le \frac{p_n(1-1/m)}{C_{\max}(OPT)} + 1.$$

By contradiction Hypothesis

By results of the previous slide 19

By the lower bound to the optimum of the previous slide

Proof of Theorem 5. (6)

- Thus

$$\begin{tikzcd}
 & \frac{4}{3}-\frac{1}{3m} < \frac{p_n(1-1/m)}{C_{\max}(OPT)}+1 \arrow[dl] & \\
C_{\max}(OPT)(\frac{1}{3}-\frac{1}{3m}) < p_n(1-\frac{1}{m}) \arrow[rr] & & C_{\max}(OPT)(\frac{m-1}{3m}) < p_n(1-\frac{1}{m}) \arrow[dl, "m>1"'] \\
 & C_{\max}(OPT) < p_n(1-\frac{1}{m})(\frac{3m}{m-1}) \arrow[dr] & \\
 & & C_{\max}(OPT) < 3p_n
\end{tikzcd}$$

Proof of Theorem 5. (7)

- Thus we have got that in the (smallest) counterexample it holds:

$$C_{\max}(OPT) < 3p_n$$

- Notice that the inequality is a strict inequality! Moreover remind that p_n is the shortest processing time. This implies that for the smallest counterexample the optimal schedule may result in at most two jobs on each machine.

- It can be shown (see the following Lemma 1) that if $p_n > C_{max}(OPT)/3$ then the schedule returned by the algorithm MAKESPAN is optimal and therefore the ratio of the two Machine MAKESPANS (i.e. the Machine MAKESPAN of the solution returned by the algorithm MAKESPAN and the Machine MAKESPAN of the optimal solution) is equal to one.
- This contradiction completes the proof of the theorem. □

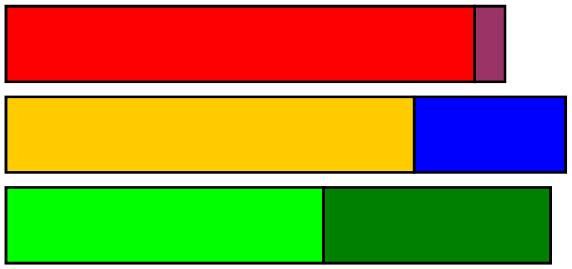
…

Lemma 1: *If, for a given instance* of *the Scheduling Identical Machine MAKESPAN (minimization) problem we have that $C_{max}(OPT)<3p_n$ then Algorithm MAKESPAN finds an optimal solution (for such an instance).*

Proof of Lemma 1.

Proof:

- First of all notice that if $3p_n > C_{max}(OPT)$ then an optimal schedule *OPT* results in at most two jobs on any machine.
- It follows that $n \le 2m$. In fact suppose (by contradiction) that $n > 2m$, then in any schedule (and then also in the optimal one) there exists at least one machine processing at least *3* jobs. A contradiction!
- In this proof we suppose that $n = 2m$, and therefore we suppose that each machine of the optimum gets exactly *2* jobs.
- (For the case $n < 2m$, by adding jobs with zero processing times we can make $n = 2m$, however the proof in this case is a bit tedious and we skip this case).
- We are going to show that there exists an *OPT* where the largest job is with the smallest job in a machine, the second largest job is with the second smallest job in another machine, etc etc.



Proof of Lemma 1. (2)

- Let $p_1 \geq p_2 \geq \ldots\ldots \geq p_n$.
- If in *OPT*, the largest job matches with the smallest job, the second largest job matches with the second smallest job etc etc, then: $C_{max}(OPT)=Max\{(p_1+p_n), (p_2+p_{n-1}), (p_3+p_{n-2}), \ldots\ldots,(p_{n/2}+p_{n/2+1})\}$

- Let $S_1=\{1,2,\ldots,n/2\}$ and $S_2=\{n/2+1,n/2+2,\ldots,n\}$. That is S_1 (resp. S_2) contains the first *n/2* jobs (resp. the last *n/2* jobs) where $p_1 \geq p_2 \geq \ldots\ldots \geq p_n$.

- First we show that there exists an *OPT* where each machine gets one job from S_1 and one from S_2. Consider any schedule (then also an optimal one) with exactly 2 jobs in each machine.
- Suppose one machine has *2* jobs *{a,b}* from S_1 and another machine has two jobs *{c,d}* from S_2 such that $p_a \geq p_b \geq p_c \geq p_d$. The completion times for these two machines would be (p_a+p_b) and (p_c+p_d).
- But $Max\{(p_a+p_b), (p_c+p_d)\} \geq Max\{(p_a+p_d), (p_b+p_c)\}$, so by interchanging jobs *b* and *d*, we can possibly improve the cost (for sure we do not increase it). Thus, there exists an optimal schedule where each machine has one job from S_1 and the other one from S_2.

Proof of Lemma 1. (3)

- Now consider any two machines in a schedule with each machine having one job from S_1 and one from S_2.
- Suppose *a* and *c* are processed on one machine and *b* and *d* are on the other, where $\{a,b\}$ are from S_1 and $\{c,d\}$ from S_2 such that $p_a \geq p_b \geq p_c \geq p_d$.
- The completion times for these two machines would be $(p_a + p_c)$ and $(p_b + p_d)$.
- But $Max\{(p_a + p_c), (p_b + p_d)\} \geq Max\{(p_a + p_d), (p_b + p_c)\}$, so by interchanging jobs *c* and *d*, we can possibly improve the cost.

- That is to say that there exists an *OPT* where the largest job matches with the smallest job, the second largest job matches with the second smallest job etc etc. (that is what we wanted to prove at the beginning).

Proof of Lemma 1. (4)

- Now we differentiate between 2 cases:

- **Case 1)** If the schedule returned by Algorithm MAKESPAN assigns at most *2* jobs to each machine, then it is easy to see that in such a schedule the largest job matches with the smallest job, the second largest job matches with the second smallest job etc etc, thus providing an optimal schedule.

Proof of Lemma 1. (5)

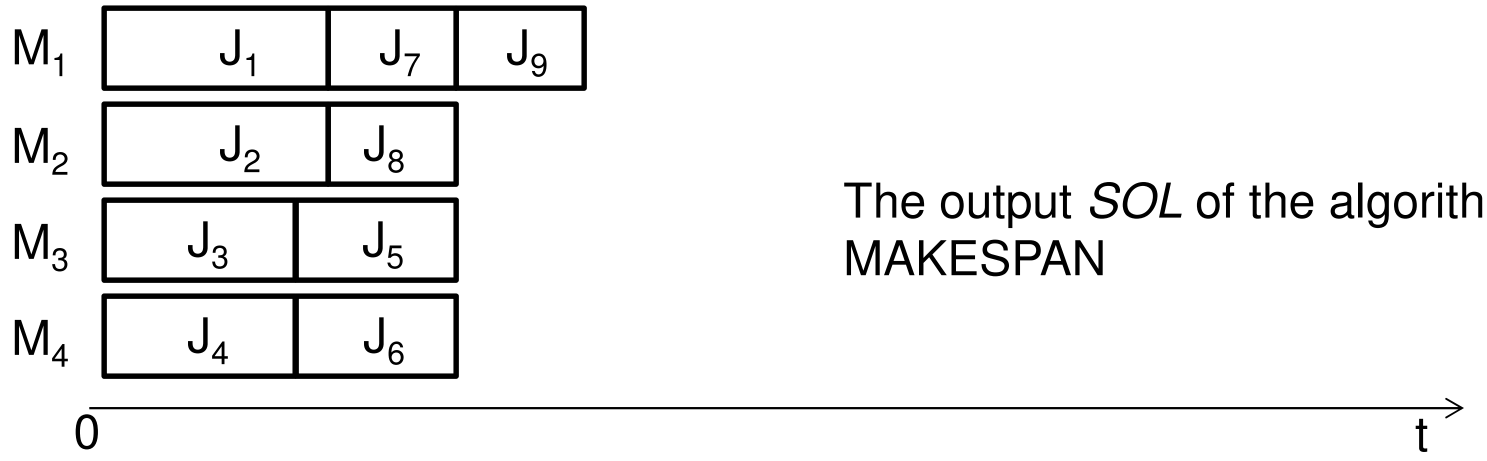
- **Case 2)** Suppose the schedule returned by Algorithm MAKESPAN processes at least *3* jobs on a machine.
- Let *f* be the first job which is put on a machine with two jobs already.
- Call a job a *loner* if it is the only job processed by a machine.
- Note that if Algorithm MAKESPAN processes at least three jobs on a machine there must be a loner job in the schedule returned by Algorithm MAKESPAN *(*remind that $n \leq 2m$*)* which is not a loner in the OPT schedule.
- Furthermore, when the job *f* is being processed by Algorithm MAKESPAN, a loner must have already been processed (otherwise it can't be a loner).
- Therefore, the loner has processing time at least $2{*}p_n > \underbrace{2{*}C_{max}(OPT)/3}_{\text{By Lemma's assumptions}}$ (otherwise *f* would be processed with the loner).
- Finally, consider the loner of the schedule returned by Algorithm MAKESPAN that is not a loner in *OPT*. The job which goes with it in the optimum schedule must have processing time smaller than $C_{max}(OPT)/3$ contradicting the premise of the Lemma. □

A worst case example of Algorithm MAKESPAN.

- Could we prove in general a strictly better approximation ratio (i.e., smaller than *4/3 - 1/3m*) for the algorithm MAKESPAN when applied to the Scheduling Identical Machine MAKESPAN (minimization) problem?
- Unfortunately not! As proved by the following example.
- INPUT: 4 identical machines, 9 jobs.

p_i=

J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
7	7	6	6	5	5	4	4	4



The output *SOL* of the algorithm MAKESPAN

$$C_{\max}(SOL) = p_1 + p_7 + p_9 = 7 + 4 + 4 = 15$$

A lower bound of OPT

Claim: $C_{\max}(OPT) \geq \frac{\sum_{j=1}^{n} p_j}{m}$

Proof:

Suppose that it is not true, that is: $C_{\max}(OPT) < \frac{\sum_{j=1}^{n} p_j}{m}$

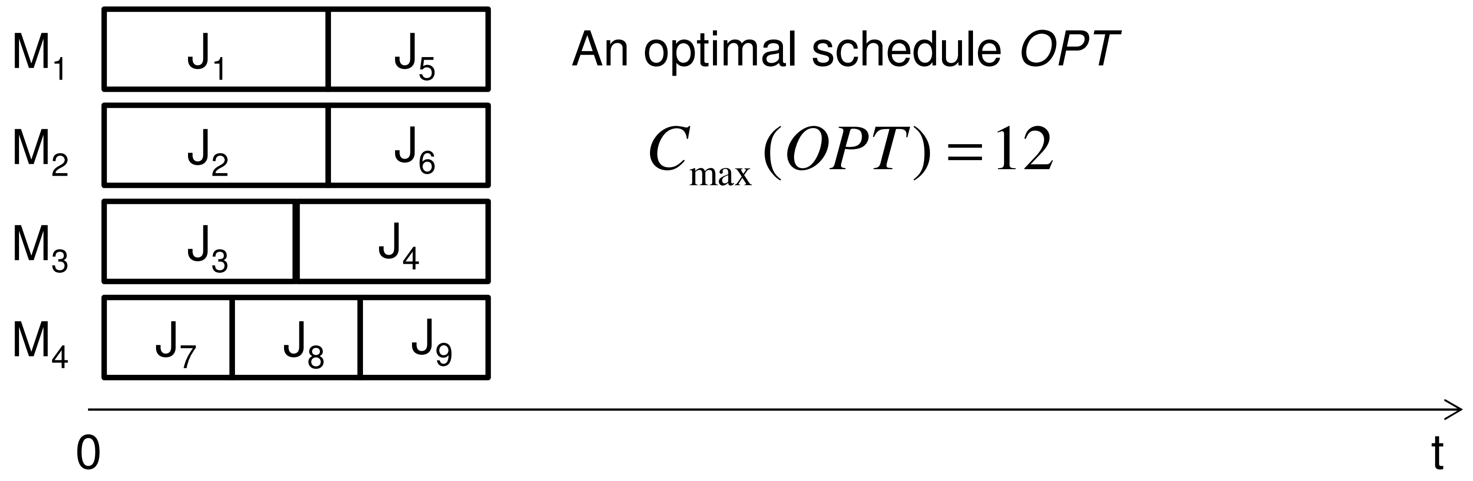
It means that if we sum over all the machines completion time we get that

$$\sum_{h=1}^{m} MC_h(OPT) < \sum_{h=1}^{m} \frac{\sum_{j=1}^{n} p_j}{m} = m \frac{\sum_{j=1}^{n} p_j}{m} = \sum_{j=1}^{n} p_i$$

That is a contradiction. □

A worst case example of Algorithm MAKESPAN. (2)

p_i=	J1	J2	J3	J4	J5	J6	J7	J8	J9
	7	7	6	6	5	5	4	4	4



An optimal schedule *OPT*

$$C_{\max}(OPT) = 12$$

We have shown an example with m=4, where the Algorithm MAKESPAN finds a schedule whose Machine MAKESPAN is exactly *4/3 - 1/3m* times the Machine MAKESPAN of the optimal solution (in fact *15/12= 4/3 - 1/3m* when *m=4*) and therefore in general we can't prove a strictly better approximation ratio for the algorithm MAKESPAN.