Simple Dispatch Rules

- We will first look at some simple dispatch rules: algorithms for which the decision about which job to run next is made based on the jobs and the time (but not on the history of jobs running, or by computing tentative schedules).
- These are also called greedy algorithms.

Goals:

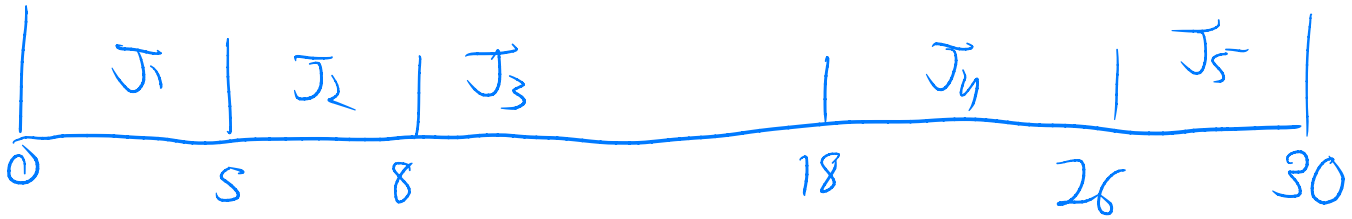
- To recognize when simple dispatch rules apply.
- To prove that they are the correct algorithm.
- To analyze the running time of the algorithm.

$1||\sum C_j$

Alg: schedule jobs in order given

Example:

j	p_j
1	**5**
2	**3**
3	**10**
4	**8**
5	**4**



$\sum C_j = 5+8+18+26+30 = 87$

1 J4 16 J5 20 J3 30 = 79

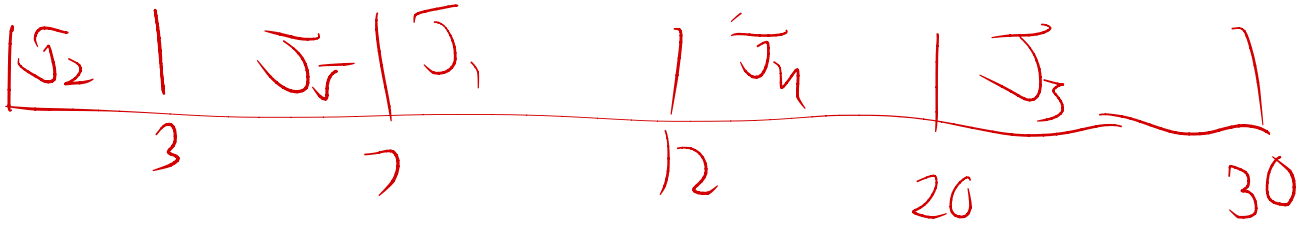
Questions:

- **What is the right algorithm?**
- **What is its running time?**
- **How do we prove it?**

$1||\Sigma C_j$

Example:

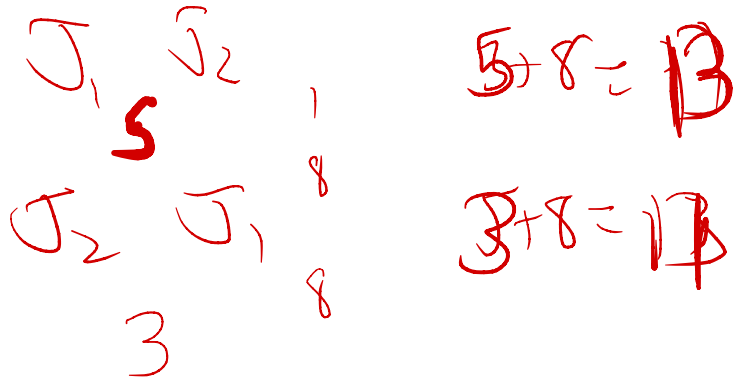
j	p_j
1	5
2	3
3	10
4	8
5	4



3+7+12+20+30 = 72

Questions:

- **What is the right algorithm? – SPT**
- **What is its running time?**
- **How do we prove it?**



$1||\Sigma C_j$

Example:

j	p_j
1	**5**
2	**3**
3	**10**
4	**8**
5	**4**

Questions:

- **What is the right algorithm?– SPT**
- **What is its running time? –** $O(n \log n)$
- **How do we prove it?**

$1||\Sigma C_j$

Example:

j	p_j
1	**5**
2	**3**
3	**10**
4	**8**
5	**4**

Questions:

- **What is the right algorithm?– SPT**
- **What is its running time? –** $O(n \log n)$
- **How do we prove it? – Interchange Argument**

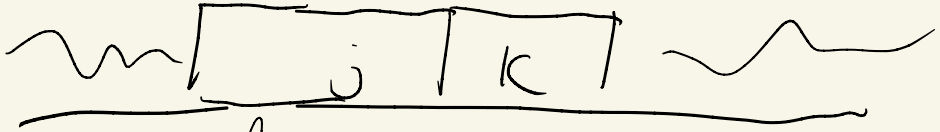
Basic format of an interchange argument

- **Specify the simple dispatch rule X.**
- **Assume, for the purposed of contradiction, that you have an optimal schedule that does not obey the rule X.**
- **Find two specific jobs j and k that violate rule X.**
- **Show that if you interchange jobs j and k, then**
  - **The resulting schedule is still feasible.**
  - **The objective function value does not increase (for a minimization problem).**
- **You can then conclude that, via repeated swaps, there must exist an optimal schedule that satisfied rule X.**

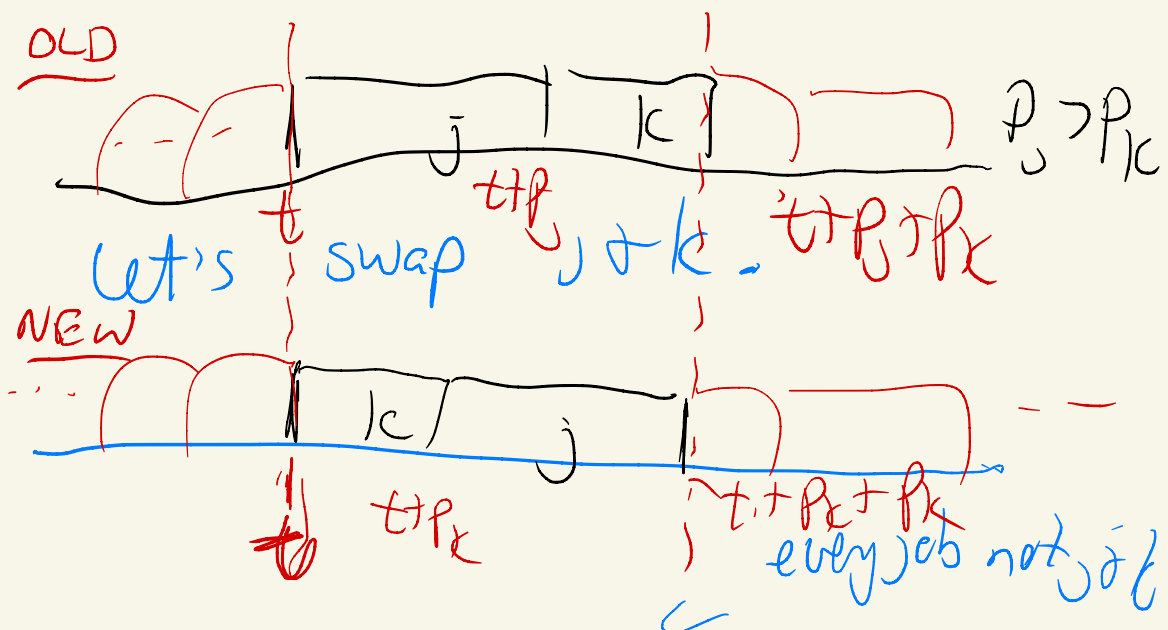
Comment: **Even though the proof is boilerplate, you must provide enough mathematical detail that shows that your proof applies to the particular problem and the particular rule!**

Claim: SPT is optimal for $1||\sum C_j$

Pf: Suppose not. Then there must be an optimal schedule that is not in SPT order. (OPT)



somewhere there are two consecutive jobs, j & k, w/ $p_j > p_k$.



OLD $\sum C_i = Q + C_j + C_k$

$= Q + (t+P_j) + (t+P_j+P_k)$

NEW $\sum C_i' = Q + C_k' + C_j'$

$= Q + (t+P_k) + (t+P_k+P_j)$

OLD

$$Q + (t+p_j) + (t+p_j+p_k)$$

NEW

$$Q + (t+p_k) + (t+p_k+p_j)$$

OLD − NEW

$$p_j - p_k \qquad (p_j > p_k)$$

$$> 0.$$

$\Rightarrow$ OLD > NEW

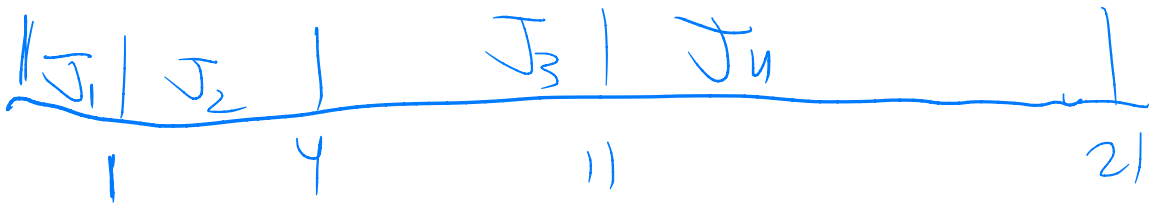
New schedule has a smaller $\sum C_j$

$\Rightarrow$ old schedule is not optimal.

Contradiction. ⊠

$1||\sum w_j C_j$

	j	p_j	w_j
Example:	**1**	**1**	**1**
	2	**3**	**2**
	3	**7**	**1**
	4	**10**	**20**

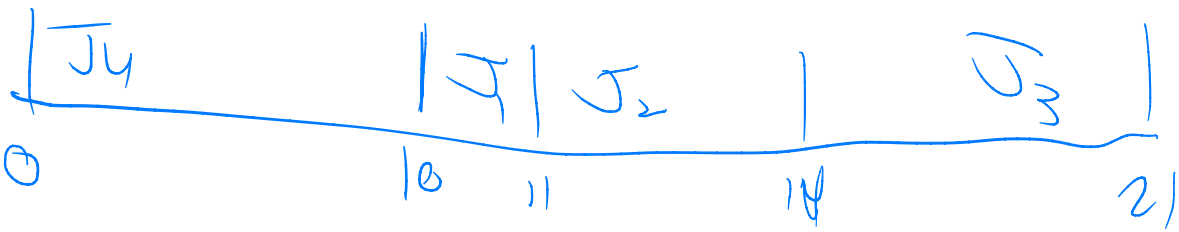


$\sum C_j = 1(1) + 2(4) + 1(11) + 20(21)$
$= 1 + 8 + 11 + 420 = 440.$

Questions:

- **How can we figure out a simple dispatch rule?**
- **How do we prove it is correct – Interchange Argument**

idea:
- good to have high weight early
- good to have short jobs early



$\sum C_j = 20(10) + 1(11) + 2(14) + 1(21)$
$= 200 + 11 + 28 + 21 = 260$

high wt jobs earlier

short jobs earlier

Maybe

$w_j - p_j$ (largest first)

$w_j^2 - p_j$

$w_j - 2p_j$

w_j / p_j

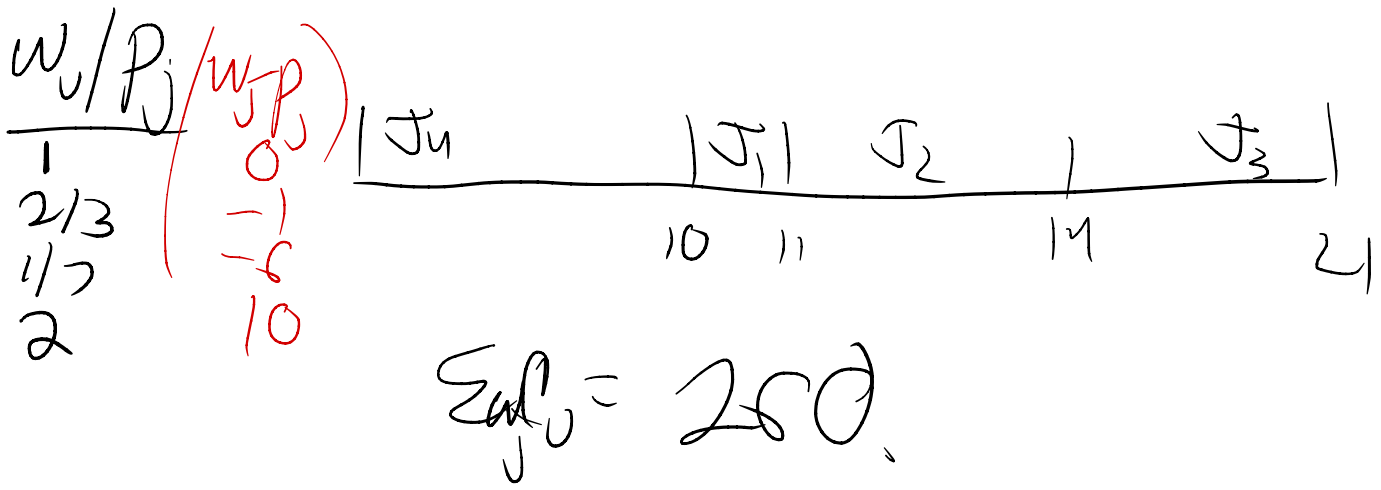
$w_j / \sqrt{p_j}$

what is the right function?

$1||\sum w_j C_j$

Example:

j	p_j	w_j
1	**1**	**1**
2	**3**	**2**
3	**7**	**1**
4	**10**	**20**

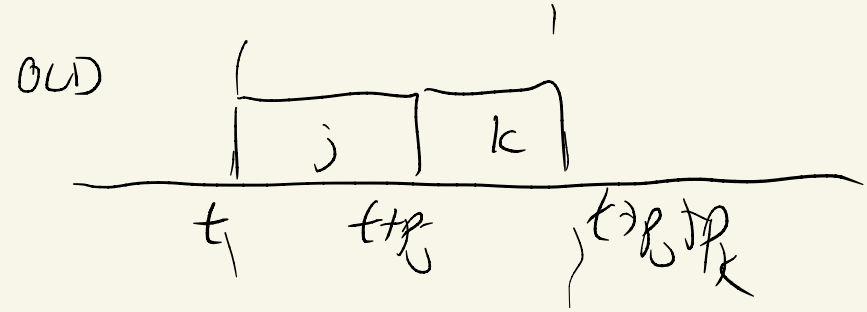


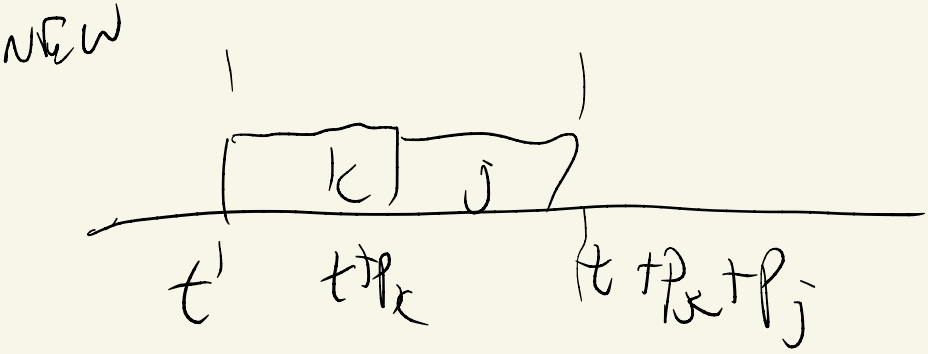
Questions:

- **How can we figure out a simple dispatch rule?**
- **How do we prove it is correct – Interchange Argument**

Two answers to first question;

- **Experiment with small examples and develop a plausible rule.**
- **Start an exchange argument and see what you need to make it work.**





OLD

$$\sum w_i C_i = Q + w_j C_j + w_k C_k$$
$$= Q + w_j(t+p_j) + w_k(t+p_j+p_k)$$

NEW

$$\sum w_i C'_i = Q + w_k C'_k + w_j C'_j$$
$$= Q + w_k(t+p_j) + w_j(t+p_j+p_k)$$

OLD - NEW

$= \cancel{Q} + w_j(t + \cancel{p_j}) + w_k(t + p_j + \cancel{p_k}) - (\cancel{Q} + w_k(t + \cancel{p_k}) + w_j(t + p_k + \cancel{p_j}))$

$= w_k p_j - w_j p_k$

When is $w_k p_j - w_j p_k > 0$ (NEW SCH. BETTER)

$w_k p_j > w_j p_k$

$\Leftrightarrow \frac{w_k}{p_k} > \frac{w_j}{p_j}$

k before j when $\frac{w_k}{p_k} > \frac{w_j}{p_j}$

WSPT, Smith's rule

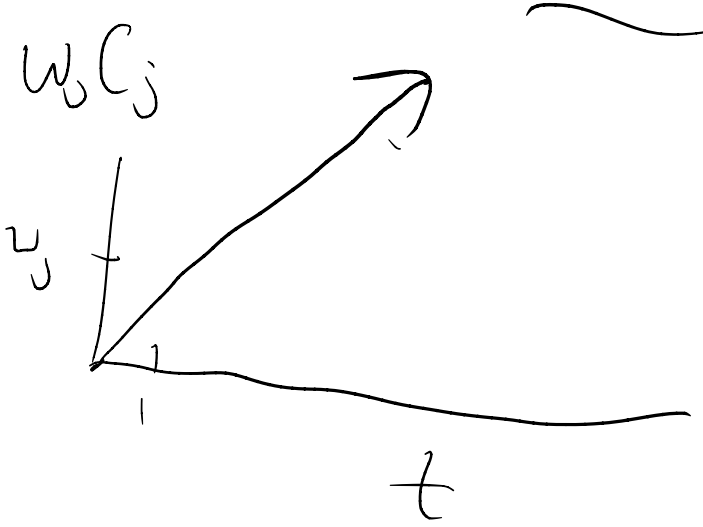
- **Want to have large weight, small processing time jobs early**
- **Schedule jobs in decreasing order of** w_j/p_j **.**

$1||\sum_j w_j(1-e^{-rC_j})$

Weighted Discounted Completion Time

- r **is a constant.**
- **For intuition:**

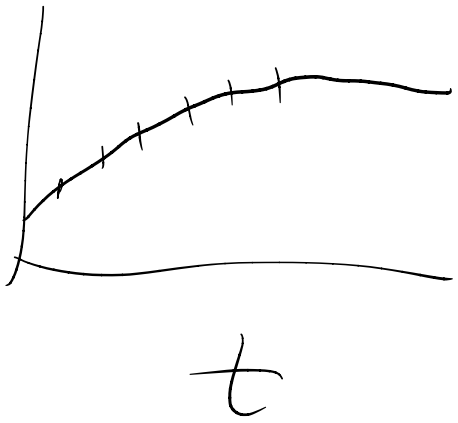
C_j	$(1-e^{-rC_j}$	$r=.1$
0	**0**	**0**
1	$1-e^{-r}$	**.10**
2	$1-e^{-2r}$	**.18**
3	$1-e^{-3r}$	**.25**
10	$1-e^{-10r}$	**.63**
100	$1-e^{-100r}$	**.99**



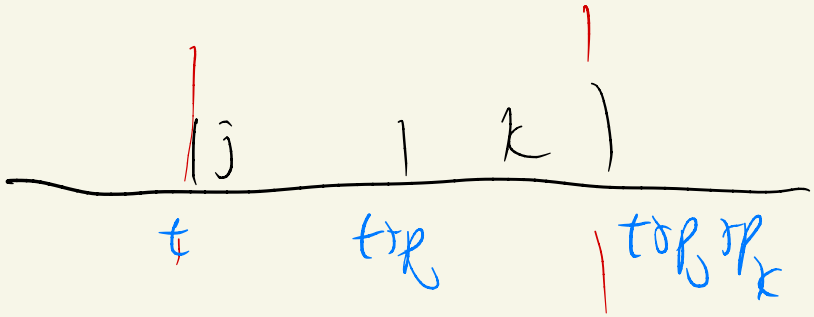
The optimal rule is to order by

$$\frac{w_j e^{-rp_j}}{1-e^{-rp_j}}$$

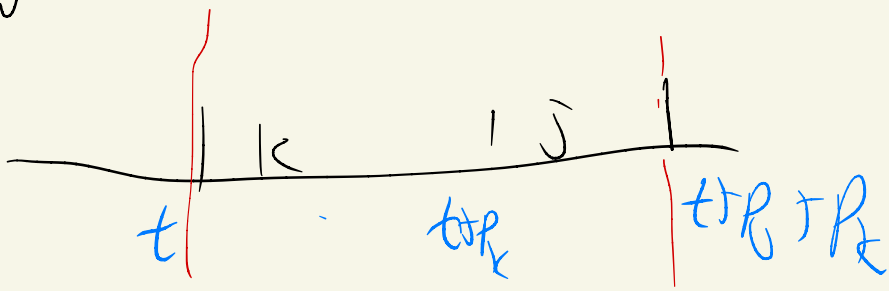
- larger w_j earlier
- smaller p_j earlier



OLD



NEW



OLD

$$Q + w_j\left(1 - e^{-r(t+p_j)}\right) + w_k\left(1 - e^{-r(t+p_j+p_k)}\right)$$

NEW

$$Q + w_k\left(1 - e^{-r(t+p_k)}\right) + w_j\left(1 - e^{-r(t+p_j+p_k)}\right)$$

$$-w_j e^{-r(t+p_j)} - w_k e^{-r(t+p_j+p_k)}$$
$$+w_k e^{-r(t+p_k)} + w_j e^{-r(t+p_j+p_k)}$$
$$= e^{-rt}\left(-w_j e^{-rp_j} - w_k e^{-r(p_j+p_k)}\right.$$
$$\left. + w_j e^{-r(p_k+p_k)} + w_k e^{-rp_k}\right)$$
$$= w_j\left(e^{-rp_j}\right)\left(e^{-rp_k}-1\right)$$
$$+ w_k e^{-rp_k}\left(1-e^{-rp_j}\right) \geq 0$$
$$\iff w_j e^{-rp_j}\left(e^{-rp_k}-1\right) \geq w_k e^{-rp_k}\left(e^{-rp_j}-1\right)$$
$$\iff \frac{w_j e^{-rp_j}}{e^{-rp_j}-1} \geq \frac{w_k e^{-rp_k}}{e^{-rp_k}-1}$$

k before j

why

$$\frac{w_k e^{-p_k}}{e^{-rp_k}-1} < \frac{w_j e^{-p_j}}{e^{-rp_j}-1}$$

$$\frac{w_k e^{-p_k}}{1-e^{-rp_k}}$$

largest first

If you got

$$\left(w_j w_k^2 - \frac{w_j p_k}{w_j}\right)^3 + p_j p_k > 0$$

$j \neq k$ cannot be separated

then you can't conclude that there is a greedy cycle.

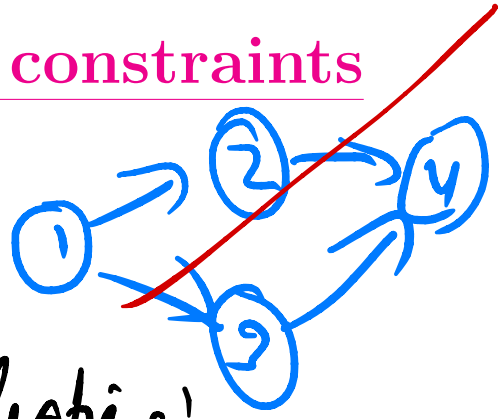
Problems with chain precedence constraints

j	p_j	w_j
1	3	6
2	6	18
3	6	12
4	5	8
5	4	8
6	8	17
7	10	18

Version 1:

1 2 3 4 5 6 7

5 6 7 1 2 3 4



Considerations:

Σp_j for a chain (smaller earlier)

w_j higher first

Chains: $1 \to 2 \to 3 \to 4$, $5 \to 6 \to 7$

- **Version 1:** Once you start a chain, the whole chain must be completed.
- **Version 2:** Chains represent normal precedence constraints.

What are the right algorithms?

Version 2: or 1, 2, 5, 3, 6, 4, 7

1, 2 3 4 5 6 7

5 6 1 2 3 7 4

⋮

high $\frac{w_j}{p_j}$ first.

Suppose we had 2 chains

$1 \to 2$ $\quad 3 \to 4$ $\quad 1,2,3,4$ $\quad 3,4,1,2$

which first?

$1 \to 2$ $\quad 3 \to 4$

$w_1(P_1) + w_2(P_1+P_2) + w_3(P_1+P_2+P_3) + w_4(P_1+P_2+P_3+P_4)$

$w_3 P_3 + w_4(P_3+P_4) + w_1(P_3+P_4+P_1) + w_2(P_3+P_4+P_1+P_2)$

$= w_3(P_1+P_2) + w_4(P_1+P_2) - w_1(P_3+P_4) - w_2(P_3+P_4)$

$(w_3+w_4)(p_1+p_2) - (w_1+w_2)(p_3+p_4) > 0$

$$\frac{w_3+w_4}{p_3+p_4} > \frac{w_1+w_2}{p_1+p_2}$$

3 -> 4 first when

compute $\frac{\sum w_i}{\sum p_i}$ for each

chain, & select in that order

Version 1: Whole Chain

j	p_j	w_j
1	**3**	**6**
2	**6**	**18**
3	**6**	**12**
4	**5**	**8**
5	**4**	**8**
6	**8**	**17**
7	**10**	**18**

44/20

43/22

$\frac{44}{20} > \frac{43}{22}$ so

out sched. is

1, 2, 3, 4, 5, 6,)

Chains: $1 \to 2 \to 3 \to 4$, $5 \to 6 \to 7$

Choose the chain with the maximum

$$\frac{\Sigma\, w_j}{\Sigma\, p_j}$$

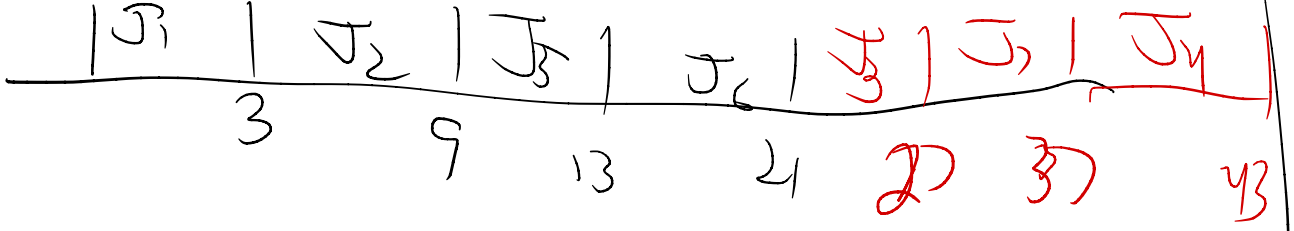
Version 2: Normal Chain Precedence Constraints

j	p_j	w_j	chain ratio	second round	third round
1	3	6	2	x	x
2	6	18	$\frac{6+18}{3+6} = 8/3$	x	x
3	6	12	12/5	2	2
4	5	8	11/5	20/11	20/11
5	4	8	2	2	x
6	8	17	25/12	25/12	x
7	10	18	43/22	43/22	9/5

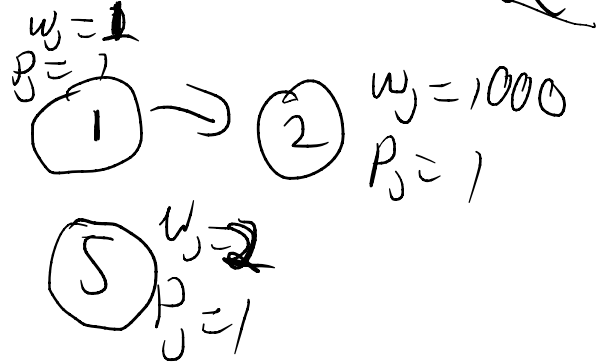
Chains: $1 \to 2 \to 3 \to 4$, $5 \to 6 \to 7$

Choose the chain prefix with the maximum

$$\frac{\Sigma w_j}{\Sigma p_j}$$



not enough to only look at 1 & 5 and close



1, 2, 5

5, 1, 2

get to job 2 as early as possible

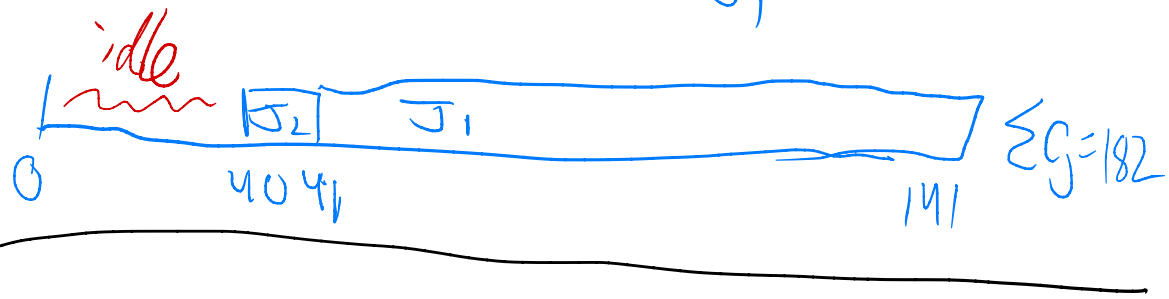
Adding release dates to a completion time problem.

NO SIMPLE DISPATCH RULE!

$1|r_j|\sum C_j$

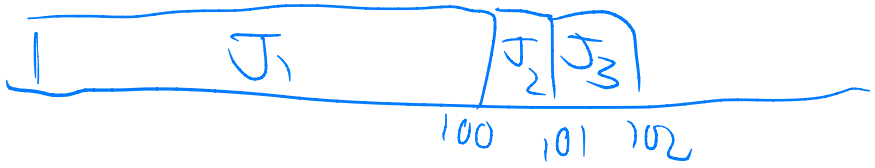
Example 1

j	r_j	p_j
1	**0**	**100**
2	**40**	**1**



Example 2

j	r_j	p_j
1	**0**	**100**
2	**40**	**1**
3	**101**	**1**



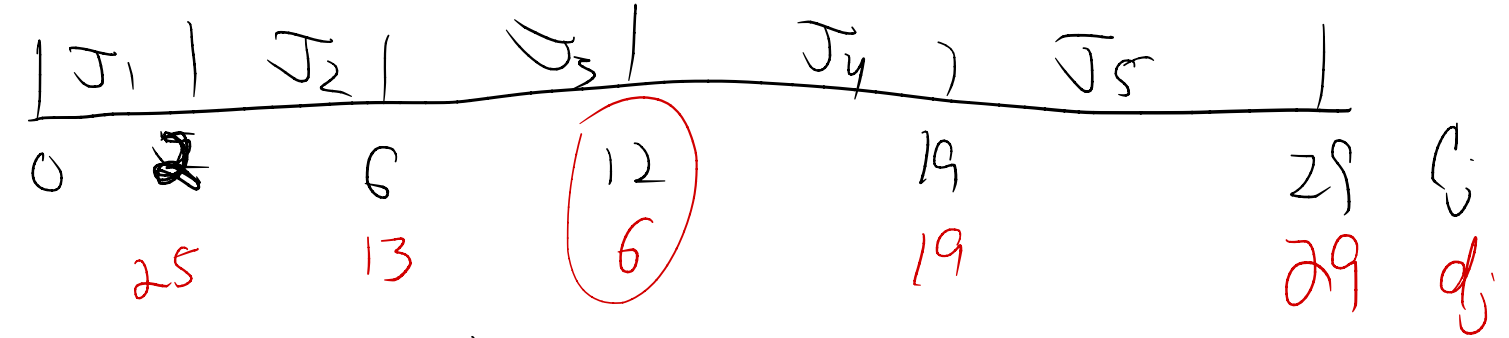
Problems with Deadlines

Deadlines

- Can be hard (deadlines) or soft (due dates).
- Model Real Time scheduling.

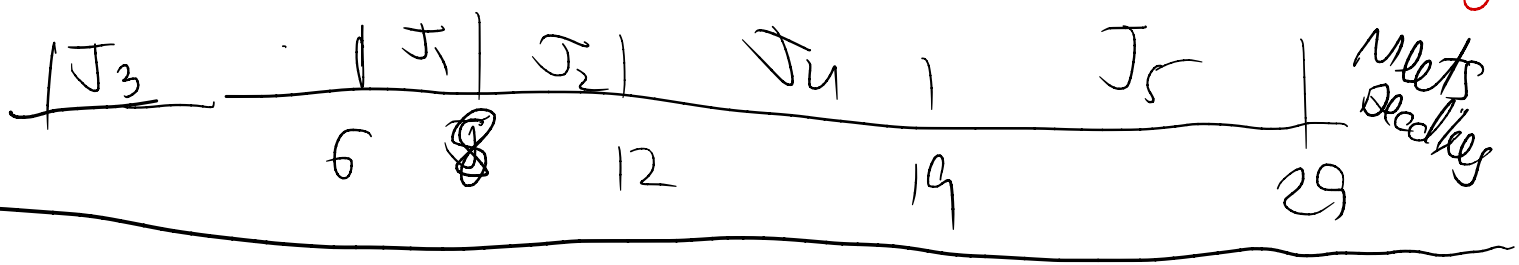
Example 1

j	p_j	d_j
1	2	25
2	4	13
3	6	6
4	7	19
5	10	29

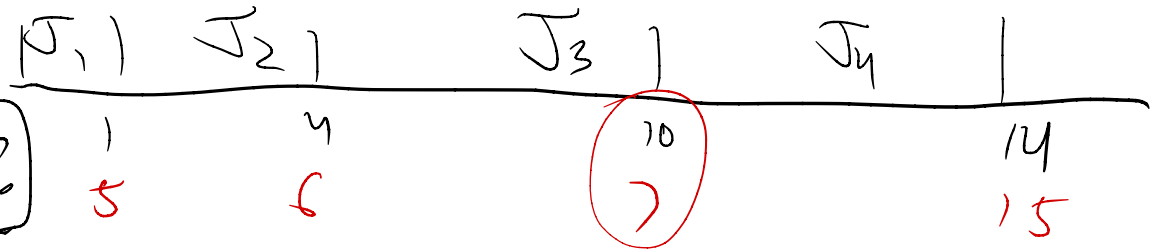


Example 2

j	p_j	d_j
1	1	5
2	3	6
3	6	7
4	4	15



NO FEASIBLE SCHEDULE



idea: Earliest due date first

idea shortest jobs earlier

Does not meet deadlines

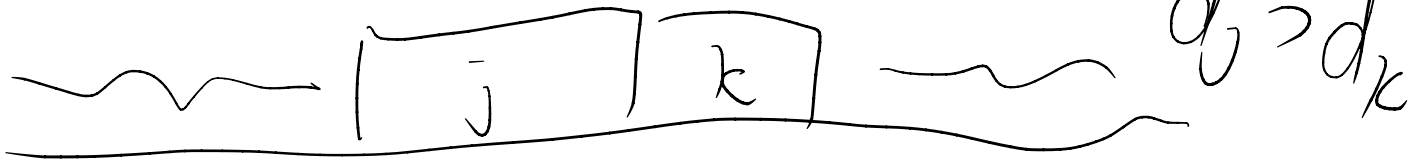
EDD - Earliest Due date

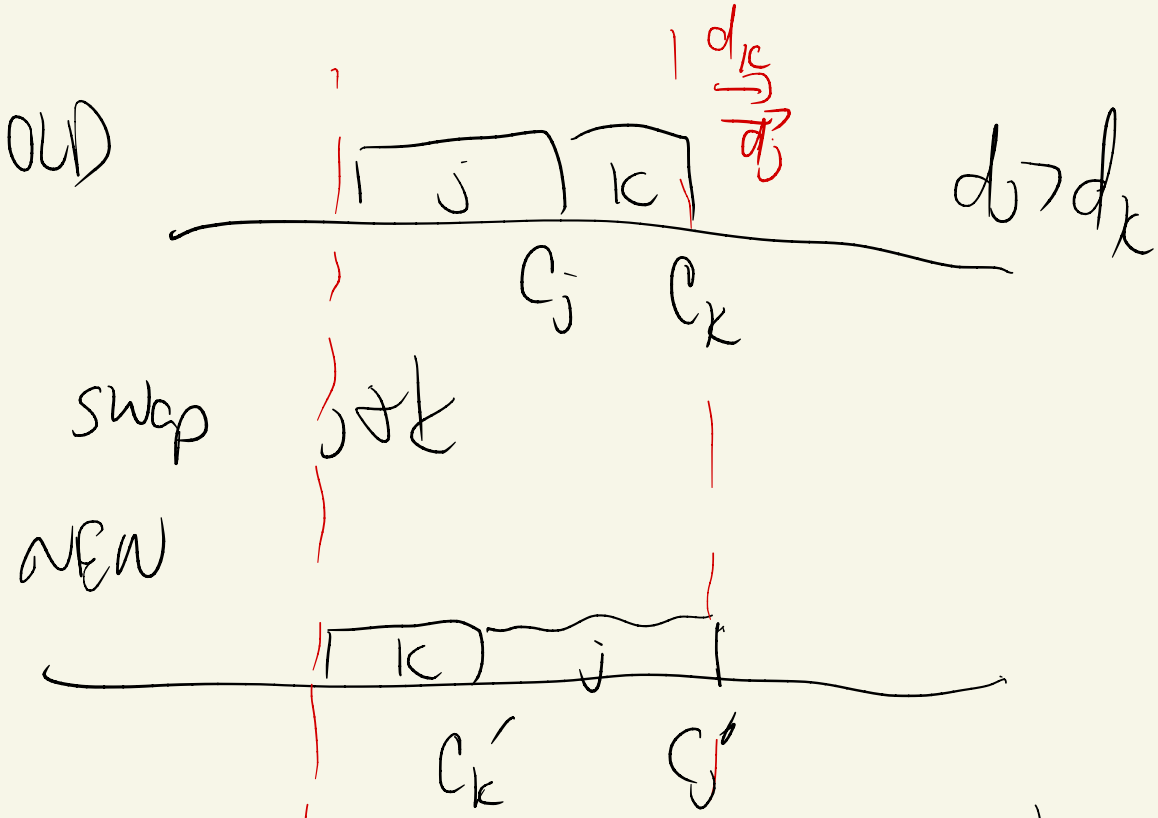
a.k.a. EDF - Earliest deadline first.

Theorem **If, for an instance, on one machine with processing times and deadlines, there is a schedule meeting all deadlines, then EDF meets all deadlines.**

P_j	d_j
1	1000
1	1001
1	1002

Pf Assume that there is a schedule meeting all deadlines, but it is not EDD. This means that there are 2 adj. jobs $j \neq k$ w/ $d_j > d_k$





Show that is NEW j & k finish before deadlines

k: $C_k' < C_k \leq d_k$

j: $C_j' = C_k \leq d_k < d_j$ ☒

What if you can't meet all deadlines:

$L_j = 0$ at deadline
$L_j > 0$ after
$L_j < 0$ before

- **One metric: lateness** $L_j = C_j - d_j$
- $1||L_{\max}$
- **Interpretation: if** $L_{\max}$ **is** 4**, then there is a schedule in which no job misses its deadline by more than** 4 **time units.**

$\sum L_j$ not used that much

Question: **Does EDD minimize** $L_{\max}$ **?**

Answer:

one sched. $L_1 = 0$ $L_2 = 0$ L_{max} 0

another sched $L_1 = 10$ $L_2 = -10$ 10

What if you can't meet all deadlines:

- **One metric: lateness** $L_j = C_j - d_j$
- $1||L_{\max}$
- **Interpretation: if** $L_{\max}$ **is** 4**, then there is a schedule in which no job misses its deadline by more than** 4 **time units.**

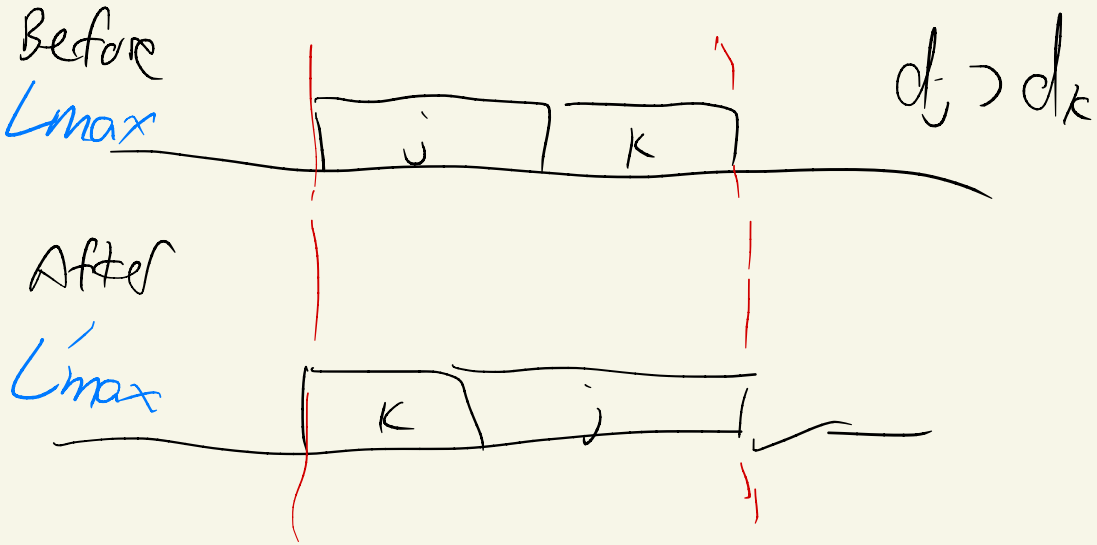
Question: **Does EDD minimize** $L_{\max}$ **?**

Answer: **YES**

- **Reason 1. Think about** $L_{\max} = t$ **as extending all deadlines by** t **.**
- **Reason 2. Interchange argument.**

Claim EDD minimizes $1||L_{max}$

Pf Assume not, TOE is a non-EDD schedule where j is before k and $d_j > d_k$. We will swap j & k and show that L_{max} does not increase.



$L'_{max} \leq L_{max}$

we will show

$$\max(L'_j, L'_k) \leq \max(L_j, L_k)$$

$L''_k \leq L_k$

$L_j < L_k$

$C_j - d_j \quad C_k - d_k$

smaller d_j

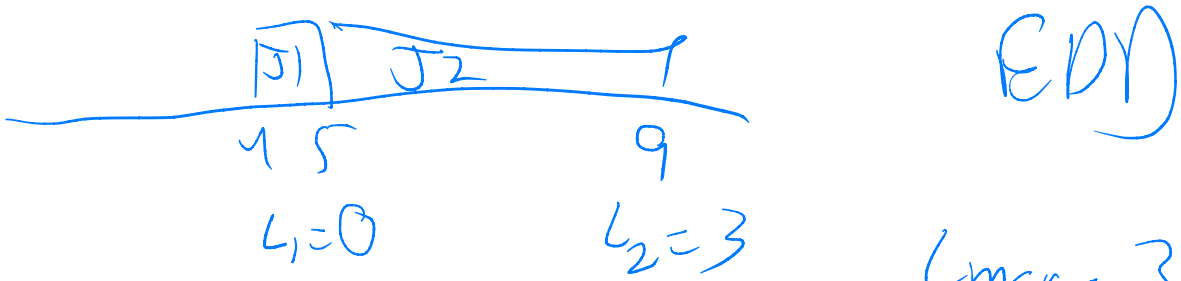
$L'_j \quad L_k$

$L'_j = C'_j - d_j = C_k - d_j < C_k - d_k = L_k$

Adding Release Dates. $1|r_j|L_{\max}$

Question: **Does EDD still produce an optimal schedule?**

j	r_j	p_j	d_j
1	**4**	**1**	**5**
2	**0**	**4**	**6**



EDD

$L_{max}=3$

$L_{max}=0$

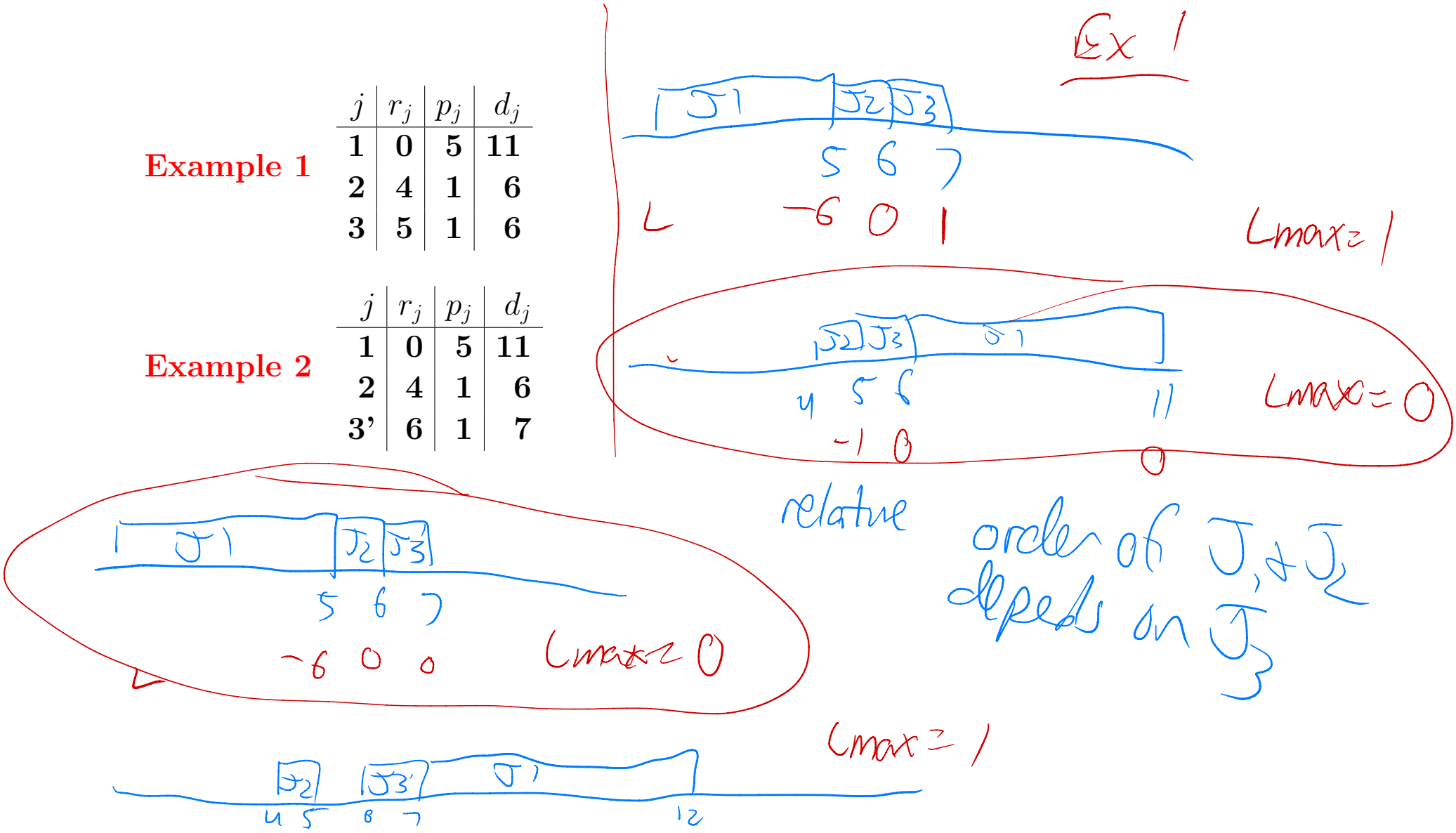
Maybe some other dispatch rule still works?

Example 1

j	r_j	p_j	d_j
1	**0**	**5**	**11**
2	**4**	**1**	**6**
3	**5**	**1**	**6**

Example 2

j	r_j	p_j	d_j
1	**0**	**5**	**11**
2	**4**	**1**	**6**
3'	**6**	**1**	**7**



Maybe some other dispatch rule still works?

Example 1

j	r_j	p_j	d_j
1	**0**	**5**	**11**
2	**4**	**1**	**6**
3	**5**	**1**	**6**

Example 2

j	r_j	p_j	d_j
1	**0**	**5**	**11**
2	**4**	**1**	**6**
3'	**6**	**1**	**7**

Maybe some other dispatch rule still works?

Example 1

j	r_j	p_j	d_j
1	**0**	**5**	**11**
2	**4**	**1**	**6**
3	**5**	**1**	**6**

Example 2

j	r_j	p_j	d_j
1	**0**	**5**	**11**
2	**4**	**1**	**6**
3'	**6**	**1**	**7**

Maybe some other dispatch rule still works?

Example 1

j	r_j	p_j	d_j
1	**0**	**5**	**11**
2	**4**	**1**	**6**
3	**5**	**1**	**6**

Example 2

j	r_j	p_j	d_j
1	**0**	**5**	**11**
2	**4**	**1**	**6**
3'	**6**	**1**	**7**