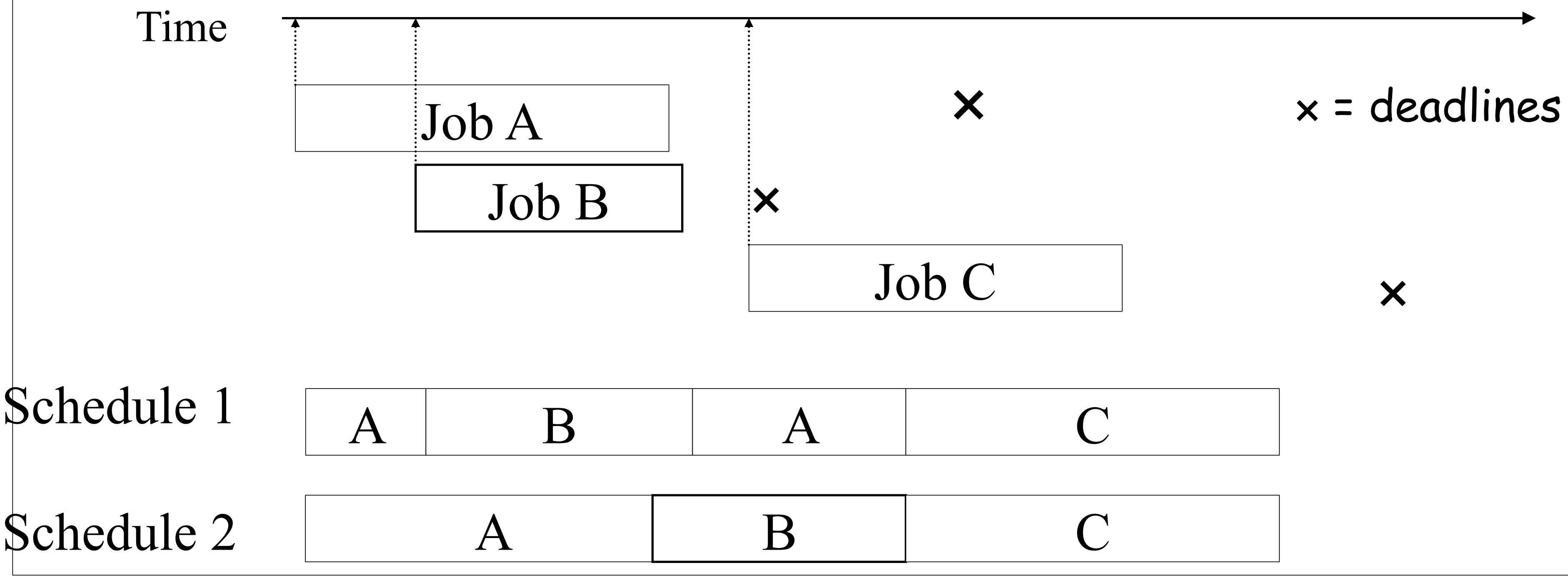
Job scheduling

- Given a set of jobs, find a schedule to run the jobs on one or more processors.
- Jobs have different release times, sizes (amount of work in terms of processing time on a processor), and perhaps deadlines.
- A processor can only process a job at a time. Preemption is allowed.



Scheduling objectives

- Offline scheduling: all jobs are known in advance.

  Online scheduling: <u>jobs are each known only at their release time</u>
  - requires continuous adjustment of the schedule as more jobs are known.
- For jobs without deadlines:
  - Minimize average/total response time (flow time)
  - Minimize average slowdown (stretch).
- For jobs with deadlines:
  - Hard deadlines: all jobs must be competed before the deadlines (e.g., nuclear reactor)
  - Soft (firm) deadlines: some jobs can be missed; maximize the total work of the jobs that can be completed by their deadlines.

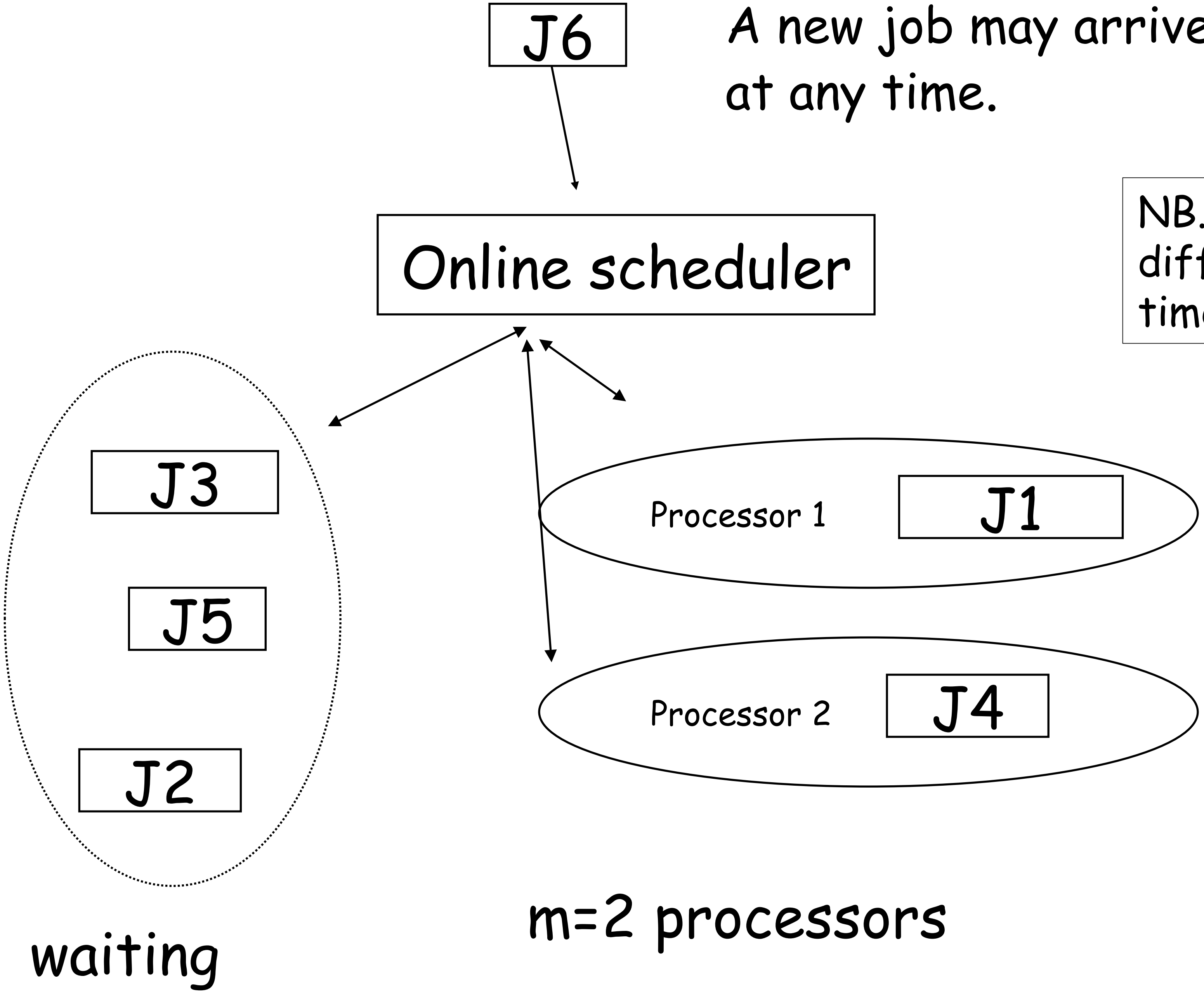
Online flow (response) time scheduling

- There is a pool of $m \geq 1$ identical processors.
- Jobs are released at arbitrary times.
- Each job can be executed on only one processor at a time.
- The work (processing time) of a job is known at its release time.; no deadline.

- An online scheduler, based on the jobs released so far, adjusts the (future) ordering of job processing.
- Flow time of a job = completion time – release time.
- Objective: Complete all jobs with total flow time minimized.

A new job may arrive at any time.

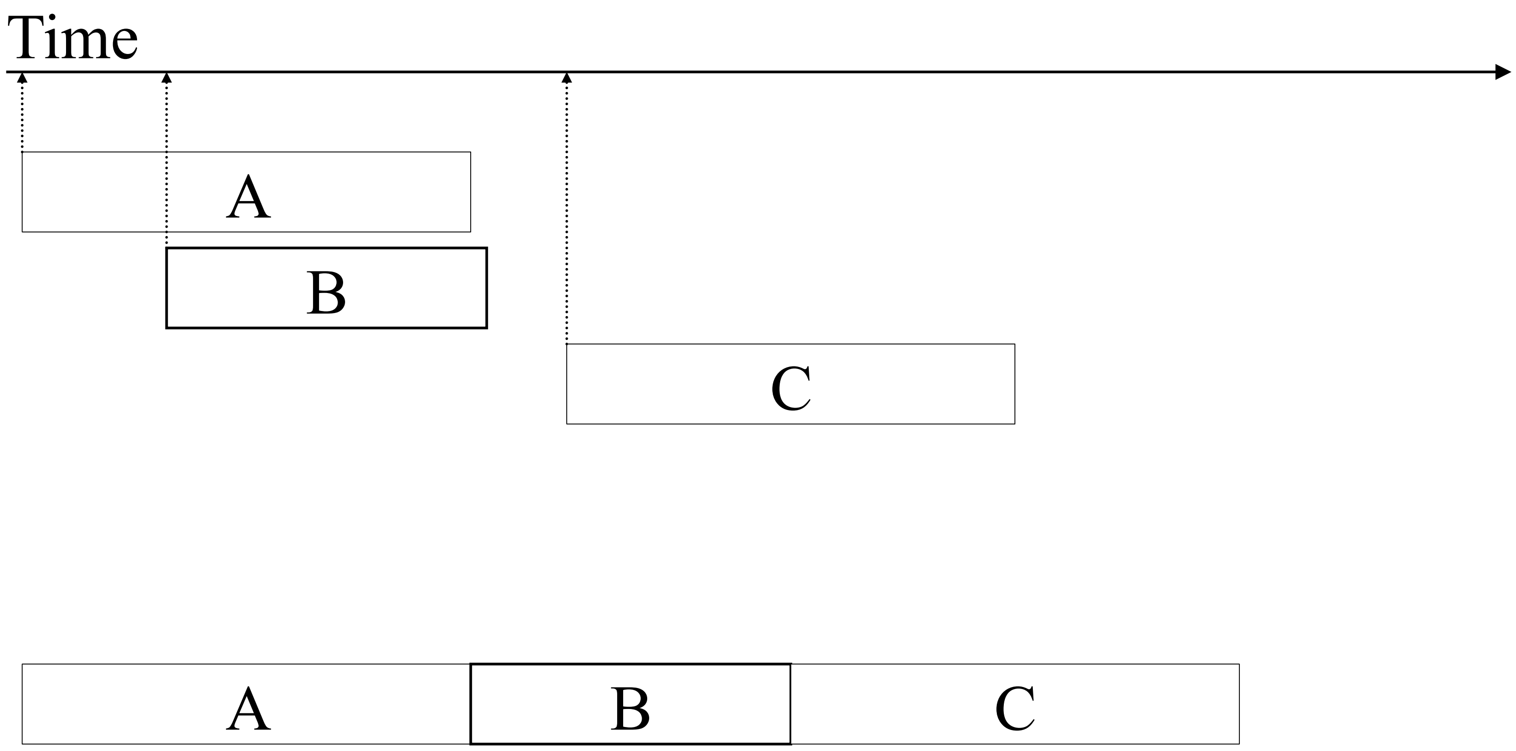
NB. Jobs may have different processing times.



Example: m = 1 processors

A sequence of 3 jobs

Shortest Remaining Processing time



One processor: SRPT minimizes flow time

SRPT: At any time, schedule the job with the shortest remaining work (processing time).

Theorem. For one-processor scheduling, SRPT is 1-competitive for flow time.

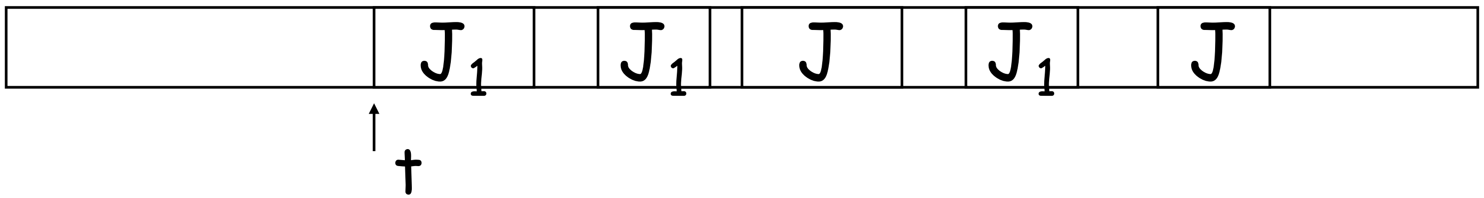
Proof : a "Transformation" argument

Given a job sequence I, let **S** be a schedule that achieves the minimum total flow time, and that doesn't follow SRPT.

Let t be the first time S schedules a job J_1 that is not the job with the shortest remaining work.

Let J be the job with the shortest remaining work at t.

Consider the schedule of J_1 & J in S.



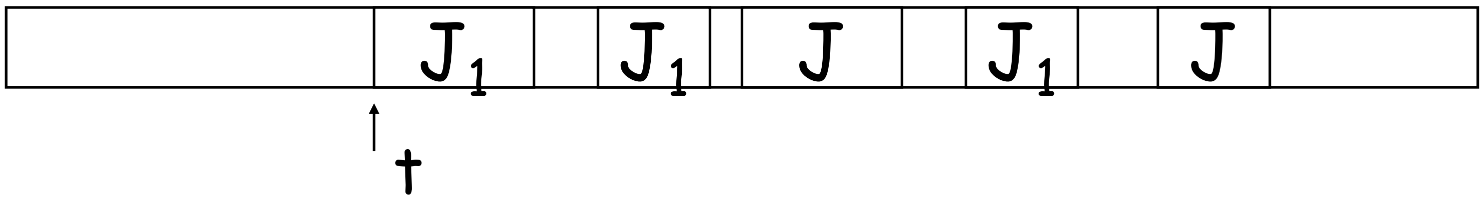
Proof : a "transformation" argument

Given a job sequence I, let **S** be a schedule that achieves the minimum total flow time.

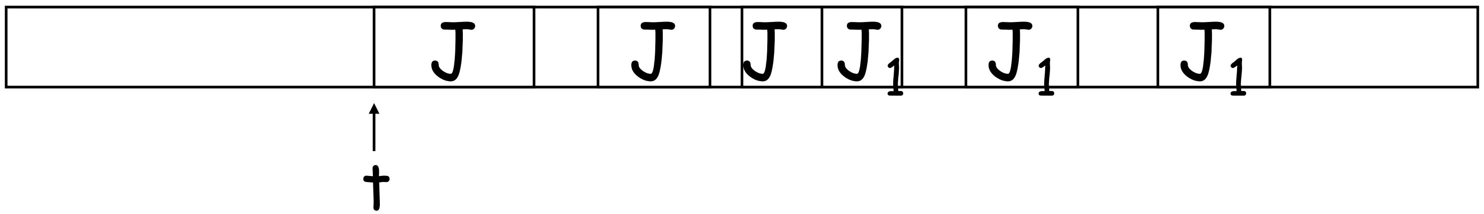
Let t be the first time S schedules a job J_1 that is not the job with the shortest remaining work.

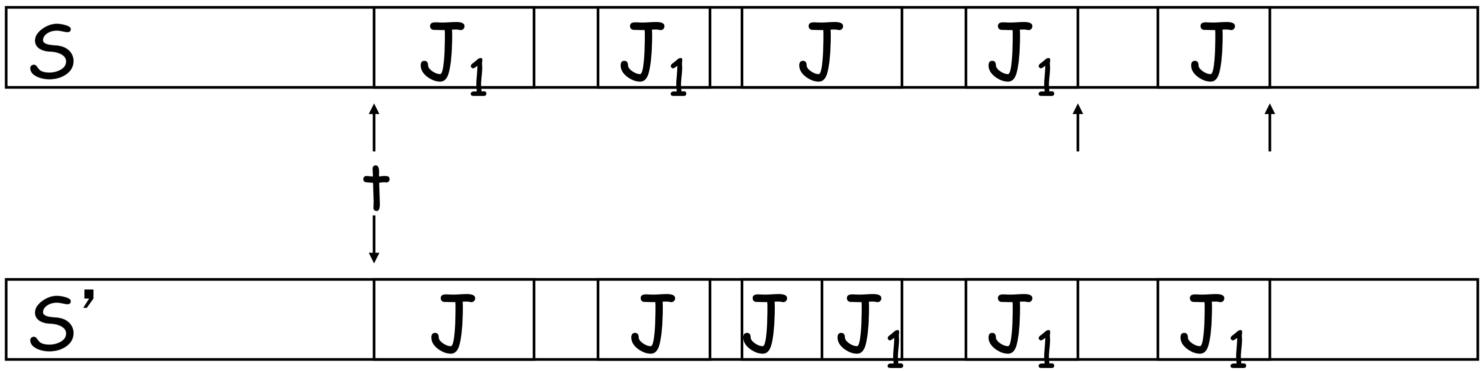
Let J be the job with the shortest remaining work at t.

Consider the schedule of J_1 & J in S.



Rearrange S so that J is processed before J_1 after time t.





In S, let x be the time S finishes one of J and J_1. Let y be the time S finishes both.

- (J_1 & J)'s flow time = total completion time – total release time
  = x + y – release(J_1) – release(J)

In S', J will be completed before J_1.

- J must be completed strictly before x.
- J_1 will be completed at y.
- (J_1 & J)'s flow time < x + y – release(J_1) – release(J)

This contradicts that S minimizes the total flow time.

Multi-processor Flow Time scheduling (m > 1)

- SRPT is no longer optimal (1-competitive).

- Example: 2004 jobs, m = 2 processors
  - At time 0 : four jobs with 10, 10, 20, 20 units of work, resp
  - At time 20 (and 21, 22, …, 1019) : two 1-unit jobs

SRPT: 10 + 10 + 2000 + 1030 + 1030

Better schedule: 10 + 20 + 20 + 2000 + 1040

Multi-processor Flow Time scheduling (m > 1)

- SRPT (using m speed-1 processors):
  - O(log P)-competitive, where P is the ratio of the maximum job size and the minimum job size.

    [Leonardi & Raz 96]
  - **Lower bound**: No algorithm can be better than log P-competitive.

- Can we do better ?

Resource Augmentation

For many scheduling problems, an online scheduler has no way to compete with an offline scheduler (which has knowledge of the future).

A naïve approach to obtaining optimality is to allow the online scheduler to have **more resources**, such as using faster processors or more processors.

(notation speed-x processors: x times faster)

Can extra resources compensate the lack of future information? How much?

Multi-processor Flow Time Scheduling (m > 1)

- SRPT (using m speed-1 processors): $O(\log P)$-competitive, where P is the ratio of the maximum job size and the minimum job size.
  [Leonardi & Raz 96]]
- SRPT (using m speed-2 processors): **1**-competitive
  [Phillips et al. stoc 97]
- SRPT (using m speed-**s** processors, where s ≥ 2): **1/s**-competitive
  - e.g., s = 4, SRPT is four times better than optimal offline algorithm.
    [McCullough & Torng soda 04]
- SRPT variant (using O(m) speed-1 processors): 1-competitive.
  [Chan, Lam & Liu soda 06]
- SRPT (using m speed-($1+\varepsilon$) processors, for any $\varepsilon>0$): $4/\varepsilon$-competitive
  [Fox & Mosely soda 2011]

Non-migratory online algorithm:

- IMD (using m speed-1 processors): $O(\log P)$-competitive [Awerbuch SPAA 02]
- IMD (using m speed-($1+\varepsilon$) processors): $O(1+1/\varepsilon)$-competitive [Chekuri, Khana & Zhu stoc 04]

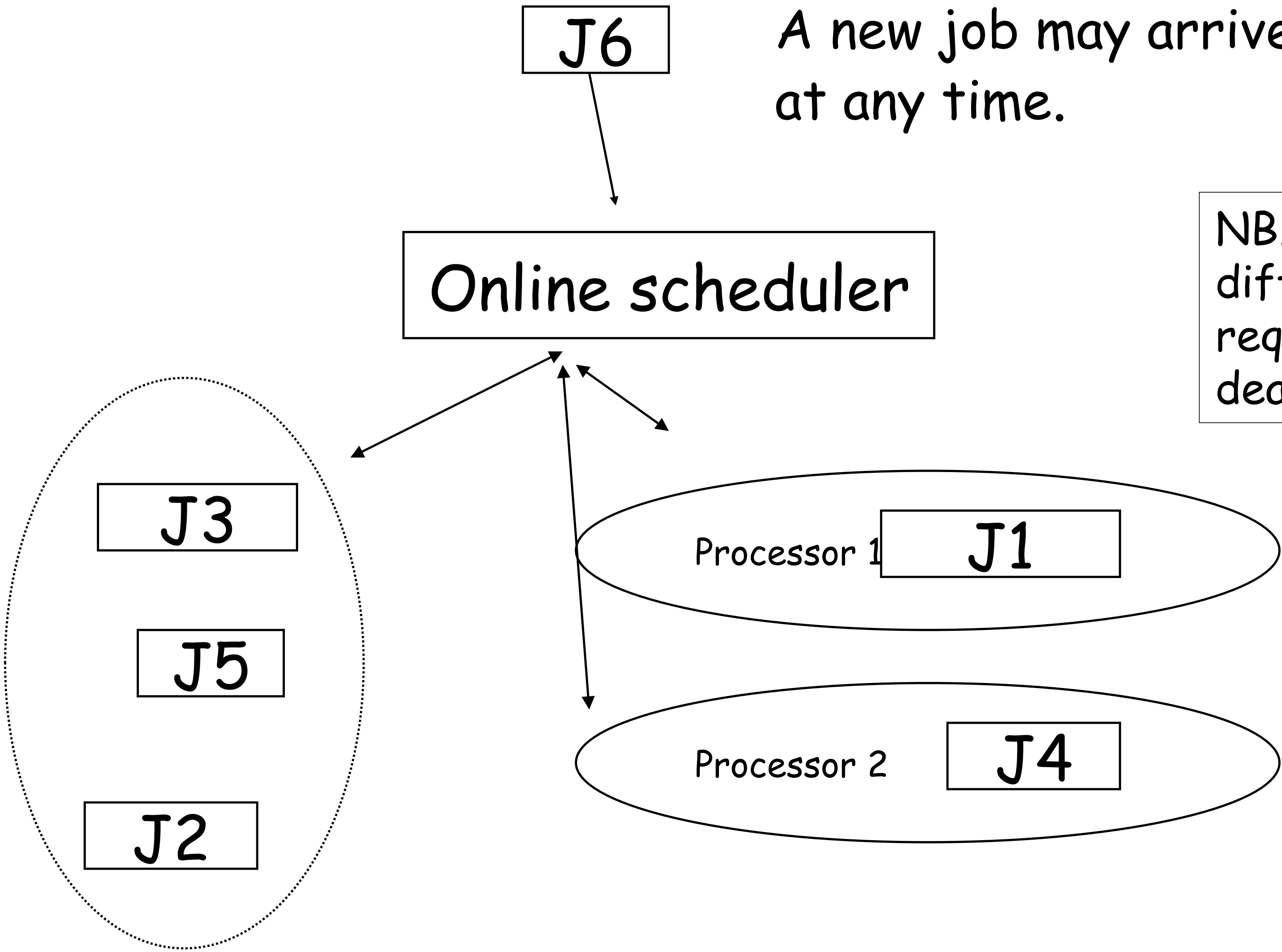
Online deadline scheduling

Next, we consider scheduling jobs with **deadlines**.

- There is a pool of $m \geq 1$ identical processors.
- Jobs are released at arbitrary times.
- The work (processing time) & deadline of a job are known at its release time.

- An online scheduler, based on the jobs released so far, adjusts the (future) ordering of job processing.
- The aim is to complete the jobs on or before their deadlines.

A new job may arrive at any time.



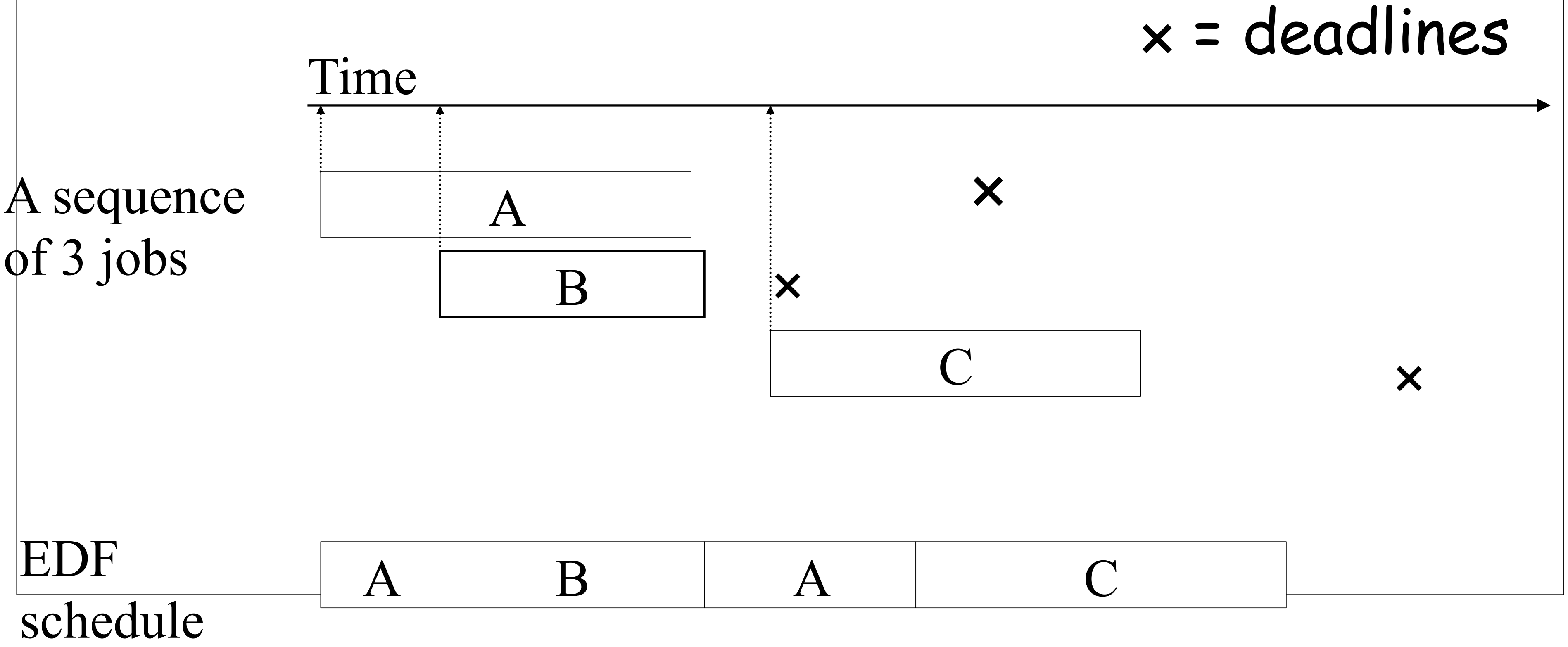
NB. Jobs have different work requirements & deadlines.

waiting

m=2 processors

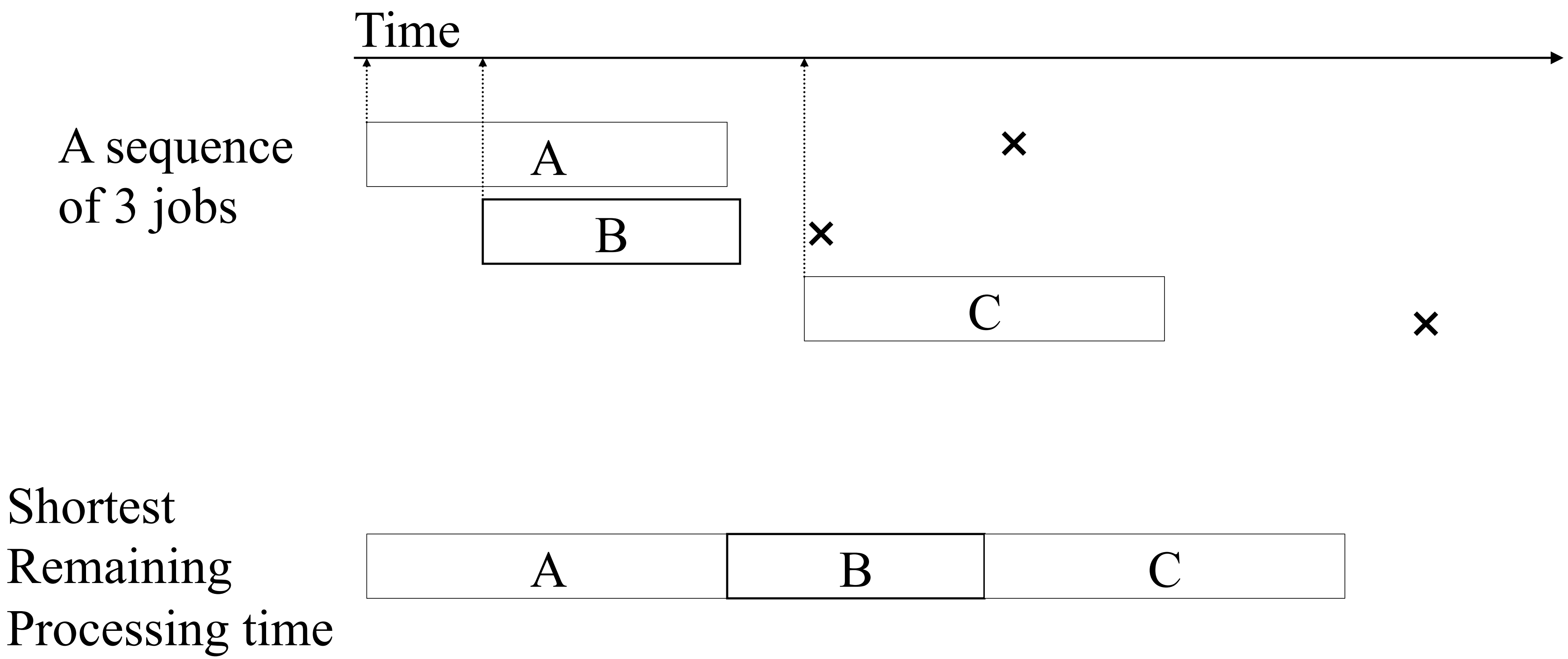
Example: m = 1 processor

EDF (Earliest Deadline First) is a widely used algorithm in many real-time systems. (Ref: Deadline scheduling for real-time systems: EDF and related algorithms, Stankovic, et al., Kluwer Academic Publishers, 1988)



NB. Preemption is free.

Example



Job B fails to meet the deadline!

Hard deadline systems

- Missing the deadline of any job can't be tolerated.

  E.g., the controller of a nuclear reactor.

- An online algorithm is said to be ***optimal*** for ***hard deadline*** systems if it can schedule all jobs to meet the deadlines whenever some offline algorithm can do so.

- In other words, we are only interested in scheduling for the ***underloaded*** setting, i.e., when it is possible to meet the deadlines of all jobs.

- E.g., When m=1, EDF is ***optimal.***

Precisely, EDF works as follows.

- There are $m \geq 1$ identical processors.
- Whenever a job is released or completed, check how many jobs haven't been completed.
- At most m such jobs: assign each of them to a processor.
- Otherwise, pick the m jobs with the earliest deadlines and assign each of them to a processor.
  - We assume that a processor, whenever given a job, will process it even if the deadline has already passed.
- Assumption: Every job has a unique deadline (break ties using job Ids).

Theorem. EDF is optimal when m = 1

Observation

EDF is a busy (or greedy) algorithm. I.e., EDF never lets a processor idle if there is a job not yet completed.

Notation

Consider any input job sequence L. Suppose that an offline algorithm Opt can meet all deadlines of L.

At any time t, let

- $E(L,t)$ = the amount of work EDF has scheduled for jobs in L up to time t.
- $O(L,t)$ = the total amount of work Opt has scheduled for jobs in L up to time t.

Lemma 1: For any time t, $E(L,t) \geq O(L,t)$

Prove by contradiction.

Suppose that t is the first time when $E(L, t) < O(L, t)$.

Is it possible that at time t, EDF has processed more work than Opt on one particular job?

Is it possible that at time t, EDF has processed more work than Opt on all jobs?

Lemma 1: For any time t, E(L,t) ≥ O(L,t)

Prove by contradiction.

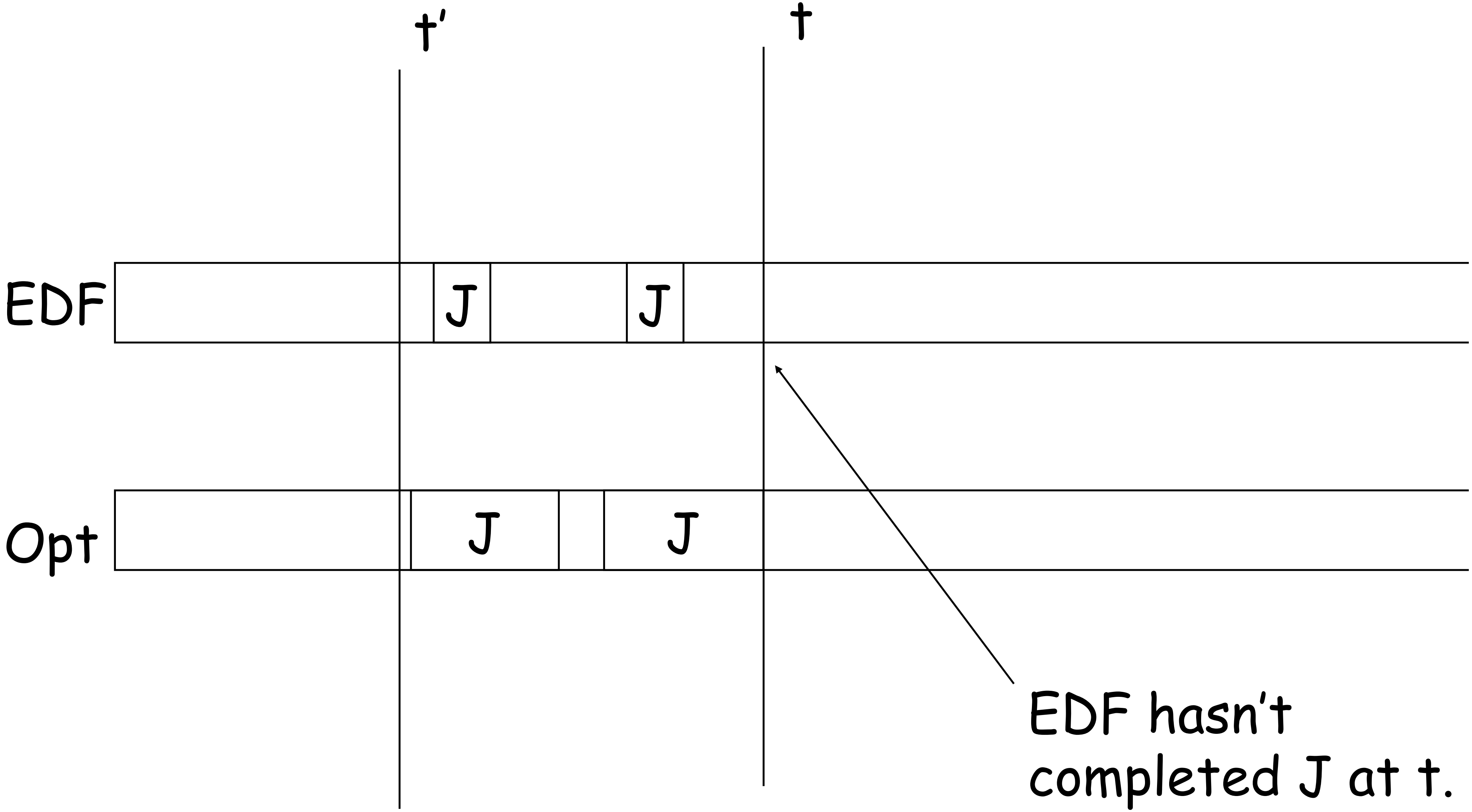
Suppose that t is the first time when $E(L, t) < O(L, t)$.

Then there exists at least one job, say, J, for which EDF schedules less work than Opt up to time t.

Note that at t, J is not yet completed by EDF.

Let t' be the release time of J. Note that $t' < t$.

Since EDF is a busy algorithm, EDF doesn't have any idling time during the interval $[t', t]$.


t'
t
EDF
J
J
Opt
J
J
EDF hasn't completed J at t.

On the other hand, at time t', E(L, t') ≥ O(L, t').

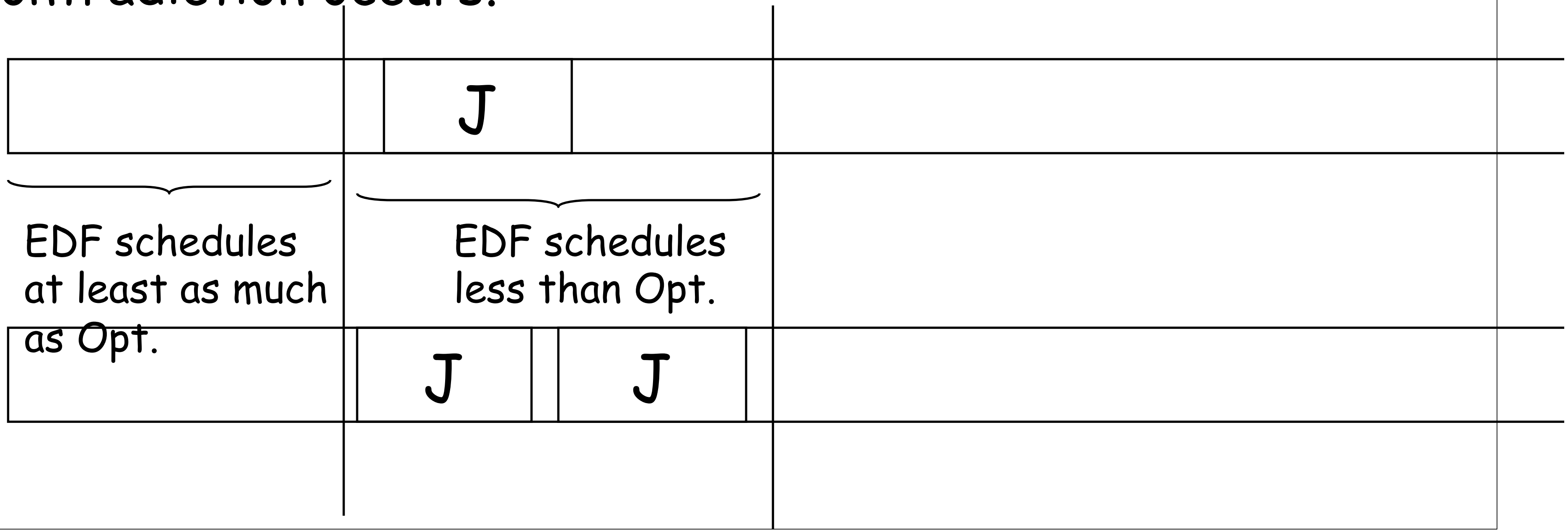
And E(L, t) < O(L, t).

During [t',t], EDF schedules less work than Opt does.

If EDF is not idle throughout [t', t], then EDF has scheduled the maximum possible work and can't be outperformed by Opt.

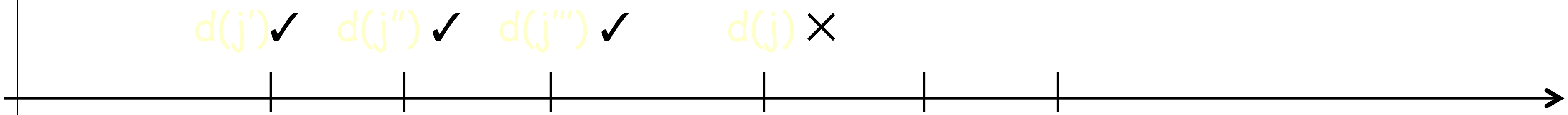
Thus, EDF must be idle at some time during [t', t].

A contradiction occurs.



Lemma 1 E(L,t) ≥ O(L, t) ➔ EDF is optimal

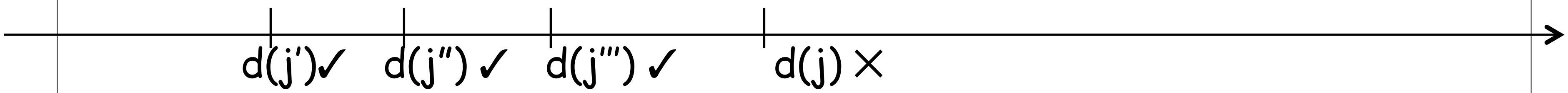
- Suppose, for the sake of contradiction, that there exists a sequence I of jobs that
  - can be completed by their deadlines using the optimal offline algorithm Opt
  - but can't be completed … using EDF.
- Among all jobs in I that EDF fails to meet the deadline, we let J denote the one with the <u>earliest</u> deadline.



- Let I' = { Jobs in I with deadlines ≤ deadline of J }.

  (NB. Jobs have distinct deadlines; use job id to break ties)
  - With respect to I', J has the latest deadline.
  - Opt can also meet all the deadlines of I'.

- With respect to EDF, the schedule of a particular job is not affected by the presence of jobs with <u>later</u> deadlines.
- The EDF schedule of I' is identical to that of the corresponding jobs in the schedule of I.
- I.e., EDF when *given* I' will complete all jobs except *J* by their deadlines.



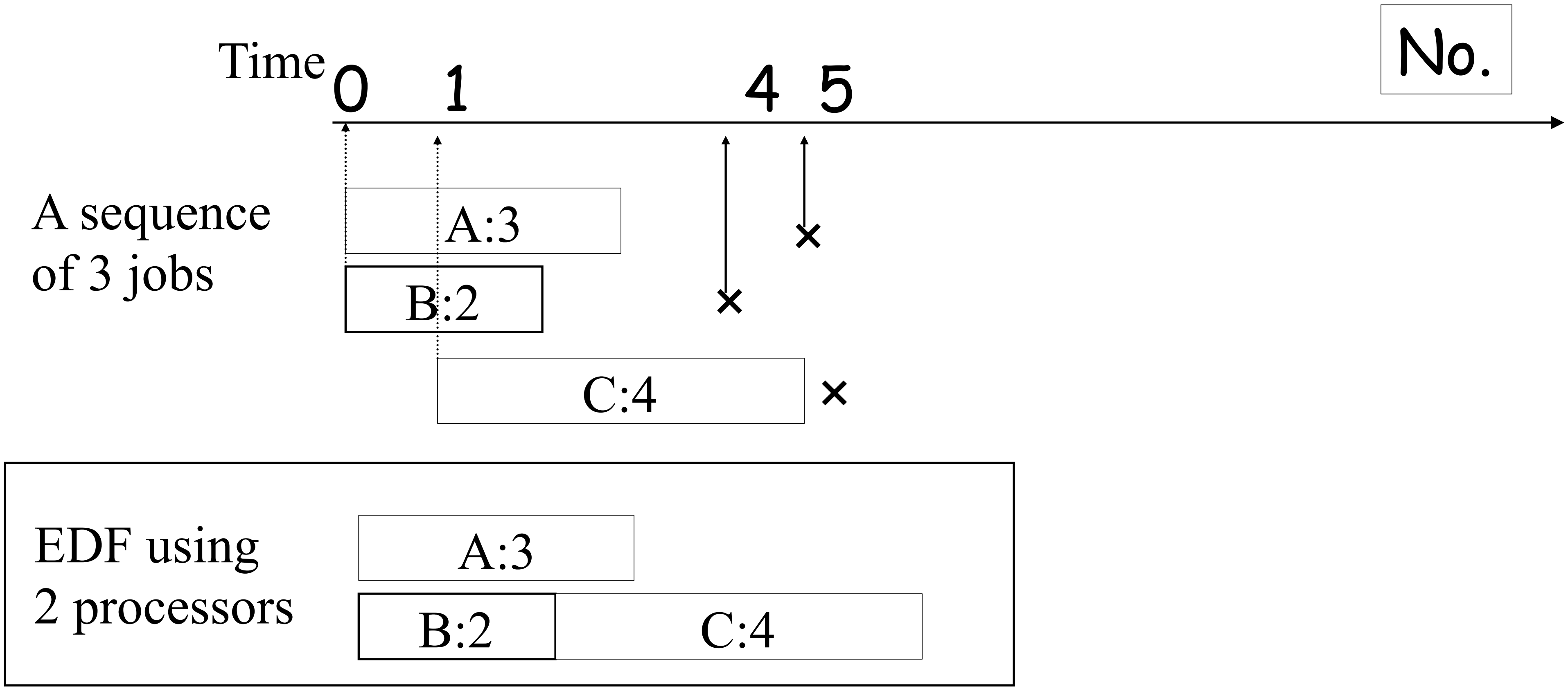
- Let d(J) be the deadline of J. As Opt can meet the deadlines of I', *O*(I', d(J)) = the total work of I'.
- By Lemma 1, E(I', d(J)) ≥ *O*(I', d(J)). Thus, EDF at time d(J) must have completed all jobs in I', including J.
- A contradiction occurs.

Lemma 1 $E(L,t) \geq O(L, t)$ ➔ EDF is optimal

- Use induction instead of proof by contradiction.
- Induction on job deadlines.
- Let J_i denote the one with the i-th earliest deadline.

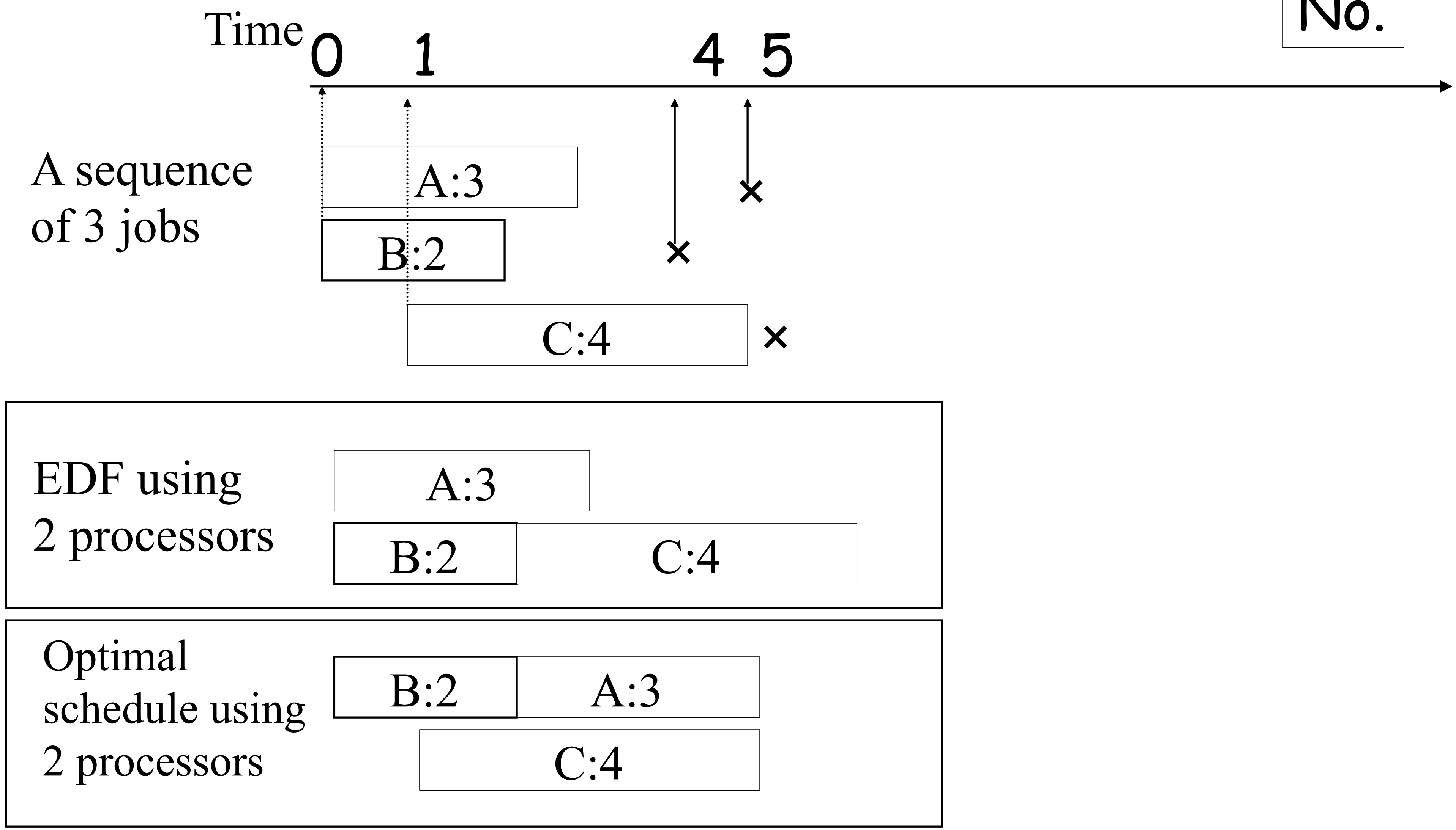
  Let $I' = \{J_1, J_2, \ldots, J_i\}$. (NB. jobs are assumed to have distinct deadlines)
  - With respect to I', J has the latest deadline.
  - Opt can also meet all the deadlines of I'.
  - By induction hypothesis, EDF on $I' - \{J_i\}$ can meet all the deadlines.
  - EDF on I' and EDF on $I' - \{J_i\}$ have identical schedule for all jobs in $I' - \{J_i\}$.
  - EDF on I' can meet all the deadlines of $I' - \{J_i\}$.
  - At time t = deadline(J_1), $E(I',t) \geq O(I', t)$, and Opt meets all the deadlines of I'.
  - Thus, EDF on I' must have finished J1 at time t.

- Question: When m = 2, is EDF optimal?



- Question: When m = 2, is EDF optimal?

No.



NB. No online algorithm is optimal [Dertouzos & Mok 1989].

NB. No online algorithm is optimal [Dertouzos & Mok 1989].

Resource Augmentation

EDF using speed-2 processors.

- I.e., online: m speed-2 processors; offline: m speed-1 processors.

- A trivial result: When $m = 2$, EDF using speed-2 processors is optimal for **hard deadline** scheduling.

- A surprising result: For all $m > 2$, EDF using speed-2 processors is optimal for **hard deadline** scheduling.

[Phillips et al. STOC 97]

Theorem 2: For m ≥ 2, EDF is speed-2 optimal.

- Precisely, for any job sequence I, if any offline algorithm using ***m*** speed-1 processors can meet all deadlines of I, then EDF using ***m*** speed-s processors, where $s = (\mathbf{1} + {}^{m-1}/_{m})$, can always do so.

- When m = 2, $(\mathbf{1} + {}^{m-1}/_{m}) = 1.5$
  When m = 3, $(\mathbf{1} + {}^{m-1}/_{m}) = 1.66\ldots$
  When m = 4, $(\mathbf{1} + {}^{m-1}/_{m}) = 1.75$
  For all m, $(\mathbf{1} + {}^{m-1}/_{m}) < 2$.

Observation

EDF is a busy (or greedy) algorithm. I.e., EDF never lets a processor idle if there are m or more jobs not yet completed.

In other words, whenever a processor is idle, there are at most m -1 jobs not yet completed and each of them is currently processed by a processor.

Notation

Consider any job set L. Suppose that an offline algorithm Opt using m speed-1 processors can meet all deadlines of L.

At any time t, let $E_s(L,t)$ and $O(L,t)$ denote respectively the total amount of **work** EDF (using m speed-s processors) and Opt (using m speed-1 processors) have scheduled for jobs in L so far.

Lemma 2: For any time t, E_s(L,t) ≥ O(L,t)

Prove by contradiction.

Suppose that t is the first time when $E_s(L, t) < O(L, t)$.

There exists a job J for which EDF, up to t, has scheduled less work than Opt has.

Note that at t, J is not yet completed by EDF.

Let $t' < t$ be the release time of J.

Consider the interval $[t', t]$.

- The presence of J guarantees that EDF would keep at least one processor busy at any time.
- Let x be the amount of time when EDF schedules all m processors, and let y be the remaining time. I.e., $t - t' = x + y$.

Within the interval [t′, t]

Consider job J.

- How much work EDF has scheduled J?

  $\geq s\,y$. Reason: Whenever EDF lets a processor idle, J must be scheduled on a processor.
- Work scheduled by Opt for J is $\leq x + y$
- Recall that EDF schedules less work for J than Opt does. Thus, $x + y > s\,y$.

- Intuitively, as $s > 1$, "$x + y > s\,y$" favors a big x, or equivalently, a small y.

Within the interval [t′, t]

By definition of t, $E_s(L, t) < O(L, t)$, and $E_s(L, t') \geq O(L, t')$.

Thus, the total amount of work EDF has scheduled during $[t', t]$, denoted α, is strictly **less** than

the total amount of work Opt has scheduled during $[t', t]$, denoted λ.

- $\alpha \geq s\ (m\ \mathrm{x} + \mathrm{y})$
- $\lambda \leq m\ (\mathrm{x} + \mathrm{y})$
- Therefore, $m\ (\mathrm{x} + \mathrm{y}) > \mathrm{s}\ (m\ \mathrm{x} + \mathrm{y})$

Intuitively, the above inequality favors a small x.

Contradiction

I. $\quad x + y > s\,y$
II. $\quad m\,(x + y) > s\,(m\,x + y)$

$(m\text{-}1)$ I : $(m\text{-}1)\,(x + y) > s\,(m\text{-}1)y$

$(m\text{-}1)$ I + II: $(m + m\ \text{-}1)\,(x + y) > s\,m\,(\,x + y\,)$

Recall that $s \geq 1 + {}^{m-1}/_{m}$.
Thus, $s\,m \geq m + m - 1$.

A contradiction occurs!

Lemma 2 $\Rightarrow$ EDF is speed-(**1+** $^{m-1}/_{m}$) optimal.

- Suppose, for the sake of contradiction, that EDF is not speed-(**1+** $^{m-1}/_{m}$) optimal.
- I.e., there exists a sequence I of jobs that can be completed by their deadlines using Opt but not using EDF.
- Let J $\in$ I be the job with the earliest deadline that EDF fails to meet its deadline.

- Let I′ = { Jobs in I with deadlines $\leq$ J's deadline }.

  (NB. Recall that jobs are assumed to have unique deadlines)
  - With respect to I′, J′ has the latest deadline.
  - (I - I′) contains jobs with deadlines > J's deadline.
  - Opt can also meet the deadlines of I′.

- With respect to EDF, the schedule of a particular job is not affected by the presence of jobs with later deadlines.
- Thus, the schedule of I' is identical to the corresponding jobs in the **schedule** of I.
- I.e., EDF when *given* I' will again fail to complete J by its deadline.

- Let *t* be the deadline of J. As Opt can meet the deadlines of I', $O(I', t)$ = the total work of I'.
- By Lemma 2, $E(I', t) \geq O(I', t)$. Thus, EDF at time *t* must have completed all jobs in I', including J.
- A contradiction occurs.

Trade-offs between processors & speed

- Can we reduce the speed requirement for optimality by allowing the online scheduler to use more than *m* processors? (technology versus money)

Trade-offs between processors & speed

- Can we reduce the speed requirement for optimality by allowing the online scheduler to use more than *m* processors? (technology versus money)
- Yes.
- Precisely, for any job sequence I, if any offline scheduling algorithm using *m* speed-1 processors can meet all deadlines of I, then EDF using $m + \varepsilon$ speed-s processors, where $s = (1 + {}^{m-1}/_{m+\varepsilon})$, can always do so.
- Examples:

When m = 2, $\varepsilon = 1$ $\quad (1 + {}^{m-1}/_{m+\varepsilon}) = 1.3333$

When m = 4, $\varepsilon = 2$ $\quad (1 + {}^{m-1}/_{m+\varepsilon}) = 1.555$

When $\varepsilon = 9$ m, $\quad (1 + {}^{m-1}/_{m+\varepsilon}) \approx 1.1$

- By choosing sufficiently large **ε,** we can make the speed requirement arbitrary close to 1.

Proof Sketch

I. $x + y > s\,y$

II. $m\,(x + y) > s\,[(m + \varepsilon)\,x + y]$

$(m + \varepsilon - 1)$ I : $(m + \varepsilon - 1)\,(x + y) > s\,(m + \varepsilon - 1)y$

$(m-1)$ I + II: $(m + m + \varepsilon - 1)\,(x + y) > s\,(m + \varepsilon)\,(x + y)$

Thus, $s < (m + m + \varepsilon - 1) / (m + \varepsilon) = \mathbf{1} + {}^{m-1}/_{m+\varepsilon}$

Open problem

- Does there exist an online algorithm that uses $m + O(m)$ speed-1 processors and can meet the deadlines of all jobs whenever it is feasible in the offline sense?

No job migration among the processors

- EDF requires migration, i.e., a preempted job may resume execution in another processor with any cost.
- In reality, migration requires overhead and it is not realistic to expect that migration costs nothing.
- Let L be a job sequence that can be completed by an offline algorithm using m speed-1 processors
- The L can also be completed by
  - migratory online algorithms (EDF and FR) using speed-2 processors
  - non-migratory online algorithm PARK [Chan, Lam & To, SODA 2004] using speed-5.828 processors

Overloaded systems

- ***Under loaded*** systems (hard deadline): there always exists a schedule that can meet the deadlines of all jobs.
- ***Overloaded*** systems (firm/soft deadline): there might be too many jobs and no (online/offline) algorithms can meet all the deadlines.
- In the overloaded case, an online algorithm A is said to be
  - **optimal** (work-optimal) if A always matches the optimal offline algorithm regarding the total work of jobs that are completed by the deadlines.
  - ***c*-competitive** if the total work completed by A is always at least a fraction of $1/c$ of that of the optimal offline algorithm.

(NB. We do not require the same set of jobs to be completed.)

Deadline scheduling under overload

Question: In the overloaded setting, is EDF optimal for scheduling a processor ($m=1$)?

No.

In fact, no algorithm is optimal.

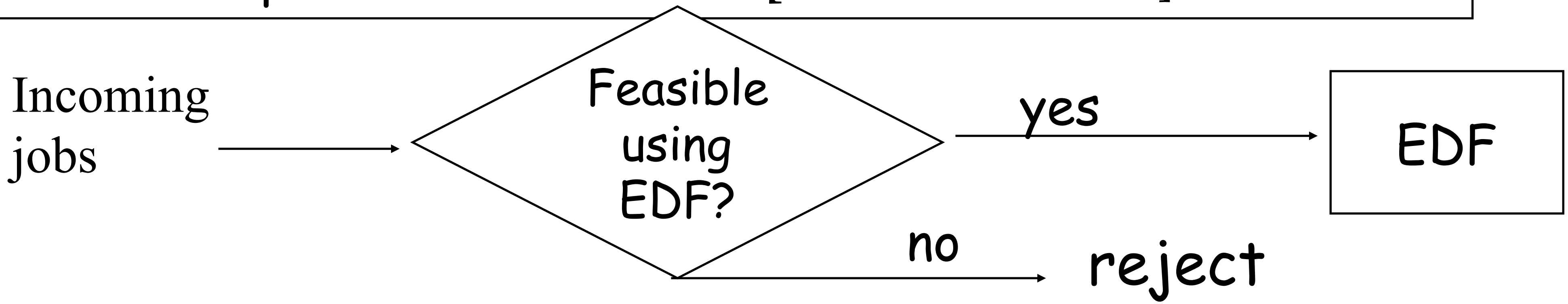
Resource augmentation: Is EDF, using a speed-2 processor, optimal for scheduling on a processor?

No.

Online algorithms for overloaded scheduling

For scheduling one processor (m = 1):

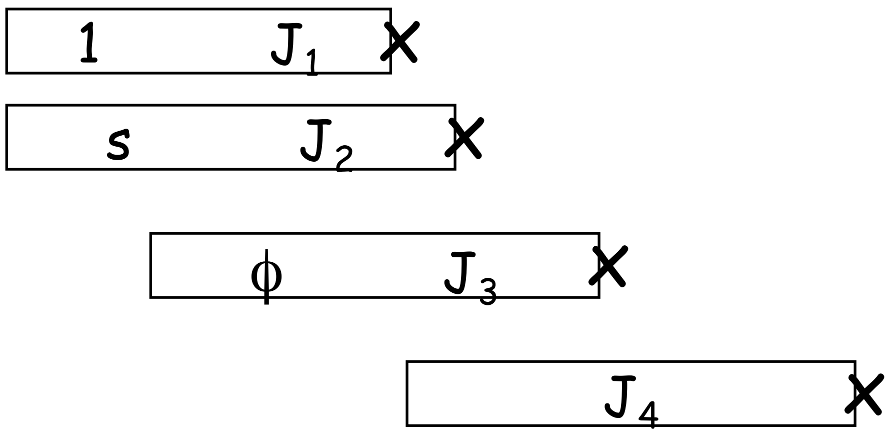
- D^{over}: (using a speed-1 processor) is 4-competitive; there is also a lower result that no algorithm can be better than 4 competitive. [Koren & Shasha sicomp 96]
- EDF with admission control: (using a speed-2 processor) is 1-competitive. [Lam & To SODA'01]



Lower bound: No algorithm is speed-s optimal if $s <$ 1.618 (i.e., the golden ratio.)

Suppose, for the sake of contradiction that, there is an online algorithm **A**, using a speed-s processor, is optimal, where $1 < s < \phi = (1+\sqrt{5})/2 \approx 1.618$.

Then we can obtain a contradiction with a job sequence of 4 jobs.



Offline algorithm (using a speed-1 processor) can complete both J_1 and J_4.

One can prove that A will complete either J_3 or J_4.