Plan for today

quiz

discuss quiz

dynamic programming intro

Back from quiz at

9:50

1. (3 points) Recall that we call a pair of values a_i and a_j in an array a a *significant inversion* if $i < j$ and $a_i > 2a_j$. How many significant inversions does the following array have?

12	2	9	4	3	5

12 - 2

12 - 4

12 - 3

12 - 5

9 - 4

9 - 3

6 significant inversions

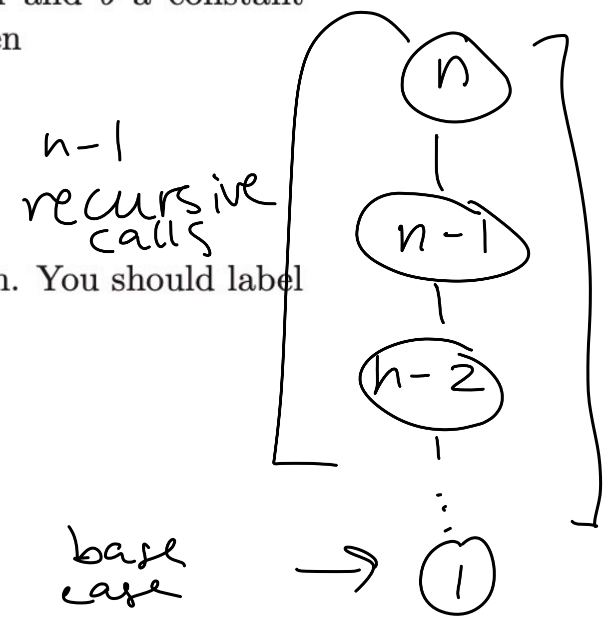
2. Recall the recursive factorial algorithm from the homework.

```
fact(n):
    if n = 1:
        Return 1
    else:
        Return n·fact(n − 1)
```

```
fact(n):
  total = 1
  for i = 1 to i = n:
    total = total · i
```

Let $T(n)$ be the runtime of fact as a function of the input, n, where c is a constant representing the number of steps taken during a recursive call and b a constant representing the number of steps during a non-recursive call. Then

$$T(n) = \begin{cases} T(n-1) + c & \text{if } n > 1 \\ b & \text{if } n = 1. \end{cases}$$



(a) (5 points) Draw the recursion tree for this recurrence relation. You should label nodes with the input size (so n should be in the first node).

$(n-1)c + b = T(n)$

(b) (1 point per blank and 3 points for the rest of the inductive case) Fill in the following parts of a proof by induction that $T(n) = c(n-1) + b$. You will need to fill in the blanks and the rest of the inductive case.

$$T(n) = \begin{cases} \boxed{T(n-1)+c} & n>1 \\ b, & n=1 \end{cases}$$

Let $n \geq 1$.

Assume that for all $m < n$, $T(m) = \boxed{c(m-1)+b}$.

There are two cases.

If $n = 1$, then $T(n) = b$. But $T(n) = c(n-1)+b = c(1-1)+b$ when $n=1$, so $T(1) = b$.

If $n > 1$, then

$T(n) = \boxed{T(n-1)}\boxed{+c}$ by definition from the recurrence

$= \boxed{c((n-1)-1)+b}\boxed{+c}$ by IH, since $n-1 < n$

$= c(n-2)+b+c$

$= cn-2c+b+c$

$= cn-c+b$

$= c(n-1)+b$

Algorithmic paradigms

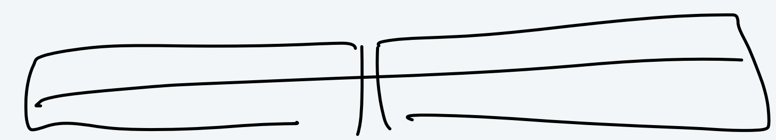
Algorithmic paradigms

Greed. Process the input in some order, myopically making irrevocable decisions.

Algorithmic paradigms

Greed. Process the input in some order, myopically making irrevocable decisions.

Divide-and-conquer. Break up a problem into independent subproblems; solve each subproblem; combine solutions to subproblems to form solution to original problem.



Algorithmic paradigms

Greed. Process the input in some order, myopically making irrevocable decisions.

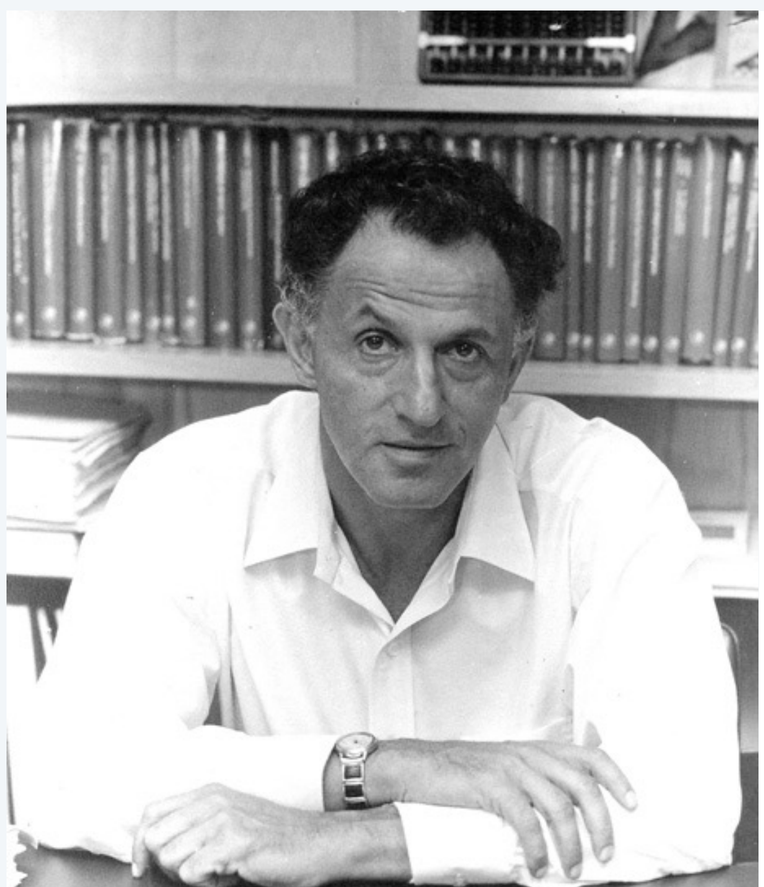
Divide-and-conquer. Break up a problem into independent subproblems; solve each subproblem; combine solutions to subproblems to form solution to original problem.

Dynamic programming. Break up a problem into a series of overlapping subproblems; combine solutions to smaller subproblems to form solution to large subproblem.

Dynamic programming history

Dynamic programming history

Bellman. Pioneered the systematic study of dynamic programming in 1950s.



THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

1. **Introduction.** Before turning to a discussion of some representative problems which will permit us to exhibit various mathematical features of the theory, let us present a brief survey of the fundamental concepts, hopes, and aspirations of dynamic programming.

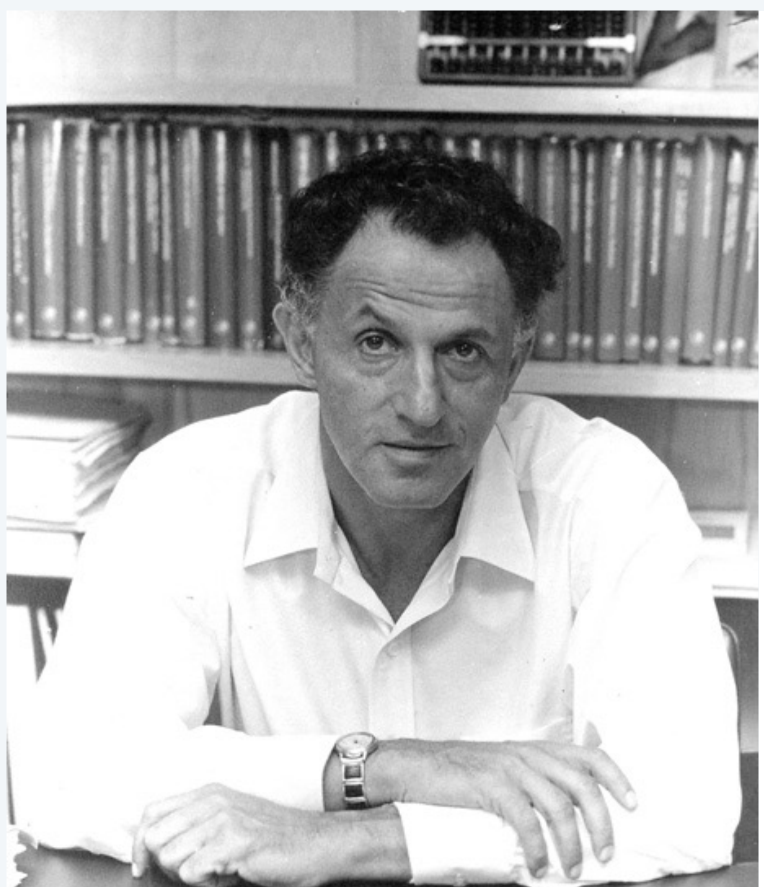
To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time t is determined by a set of quantities which we call state parameters, or state variables. At certain times, which may be prescribed in advance, or which may be determined by the process itself, we are called upon to make decisions which will affect the state of the system. These decisions are equivalent to transformations of the state variables, the choice of a decision being identical with the choice of a transformation. The outcome of the preceding decisions is to be used to guide the choice of future ones, with the purpose of the whole process that of maximizing some function of the parameters describing the final state.

Examples of processes fitting this loose description are furnished by virtually every phase of modern life, from the planning of industrial production lines to the scheduling of patients at a medical clinic; from the determination of long-term investment programs for universities to the determination of a replacement policy for machinery in factories; from the programming of training policies for skilled and unskilled labor to the choice of optimal purchasing and inventory policies for department stores and military establishments.

Dynamic programming history

Bellman. Pioneered the systematic study of dynamic programming in 1950s.

Etymology.



THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

1. **Introduction.** Before turning to a discussion of some representative problems which will permit us to exhibit various mathematical features of the theory, let us present a brief survey of the fundamental concepts, hopes, and aspirations of dynamic programming.

To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time t is determined by a set of quantities which we call state parameters, or state variables. At certain times, which may be prescribed in advance, or which may be determined by the process itself, we are called upon to make decisions which will affect the state of the system. These decisions are equivalent to transformations of the state variables, the choice of a decision being identical with the choice of a transformation. The outcome of the preceding decisions is to be used to guide the choice of future ones, with the purpose of the whole process that of maximizing some function of the parameters describing the final state.

Examples of processes fitting this loose description are furnished by virtually every phase of modern life, from the planning of industrial production lines to the scheduling of patients at a medical clinic; from the determination of long-term investment programs for universities to the determination of a replacement policy for machinery in factories; from the programming of training policies for skilled and unskilled labor to the choice of optimal purchasing and inventory policies for department stores and military establishments.

Dynamic programming history

Bellman. Pioneered the systematic study of dynamic programming in 1950s.

Etymology.

- Dynamic programming = planning over time.

THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

1. **Introduction.** Before turning to a discussion of some representative problems which will permit us to exhibit various mathematical features of the theory, let us present a brief survey of the fundamental concepts, hopes, and aspirations of dynamic programming.

To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time t is determined by a set of quantities which we call state parameters, or state variables. At certain times, which may be prescribed in advance, or which may be determined by the process itself, we are called upon to make decisions which will affect the state of the system. These decisions are equivalent to transformations of the state variables, the choice of a decision being identical with the choice of a transformation. The outcome of the preceding decisions is to be used to guide the choice of future ones, with the purpose of the whole process that of maximizing some function of the parameters describing the final state.

Examples of processes fitting this loose description are furnished by virtually every phase of modern life, from the planning of industrial production lines to the scheduling of patients at a medical clinic; from the determination of long-term investment programs for universities to the determination of a replacement policy for machinery in factories; from the programming of training policies for skilled and unskilled labor to the choice of optimal purchasing and inventory policies for department stores and military establishments.

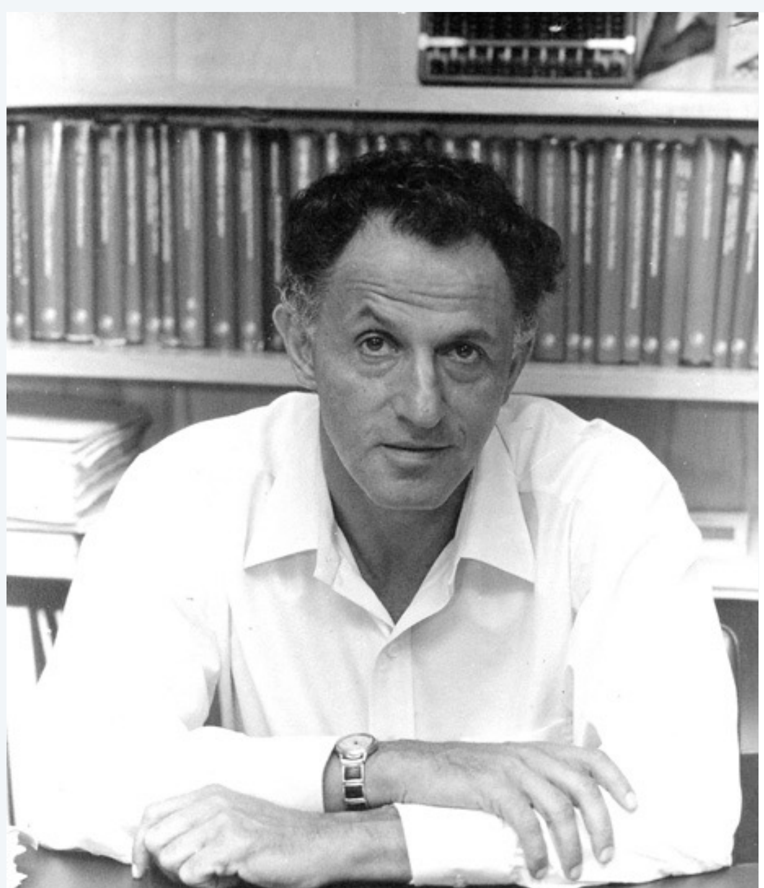
Dynamic programming history

Bellman. Pioneered the systematic study of dynamic programming in 1950s.

planning / problem solving

Etymology.

- Dynamic programming = planning over time.
- Secretary of Defense had pathological fear of mathematical research.



THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

1. **Introduction.** Before turning to a discussion of some representative problems which will permit us to exhibit various mathematical features of the theory, let us present a brief survey of the fundamental concepts, hopes, and aspirations of dynamic programming.

To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time t is determined by a set of quantities which we call state parameters, or state variables. At certain times, which may be prescribed in advance, or which may be determined by the process itself, we are called upon to make decisions which will affect the state of the system. These decisions are equivalent to transformations of the state variables, the choice of a decision being identical with the choice of a transformation. The outcome of the preceding decisions is to be used to guide the choice of future ones, with the purpose of the whole process that of maximizing some function of the parameters describing the final state.

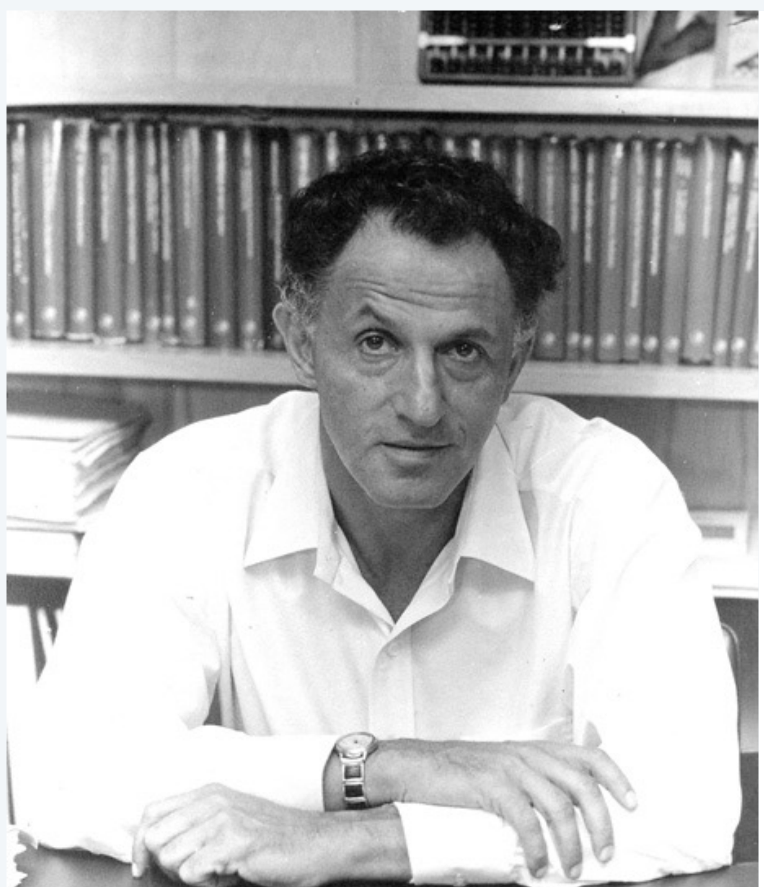
Examples of processes fitting this loose description are furnished by virtually every phase of modern life, from the planning of industrial production lines to the scheduling of patients at a medical clinic; from the determination of long-term investment programs for universities to the determination of a replacement policy for machinery in factories; from the programming of training policies for skilled and unskilled labor to the choice of optimal purchasing and inventory policies for department stores and military establishments.

Dynamic programming history

Bellman. Pioneered the systematic study of dynamic programming in 1950s.

Etymology.

- Dynamic programming = planning over time.
- Secretary of Defense had pathological fear of mathematical research.
- Bellman sought a “dynamic” adjective to avoid conflict.



THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

1. **Introduction.** Before turning to a discussion of some representative problems which will permit us to exhibit various mathematical features of the theory, let us present a brief survey of the fundamental concepts, hopes, and aspirations of dynamic programming.

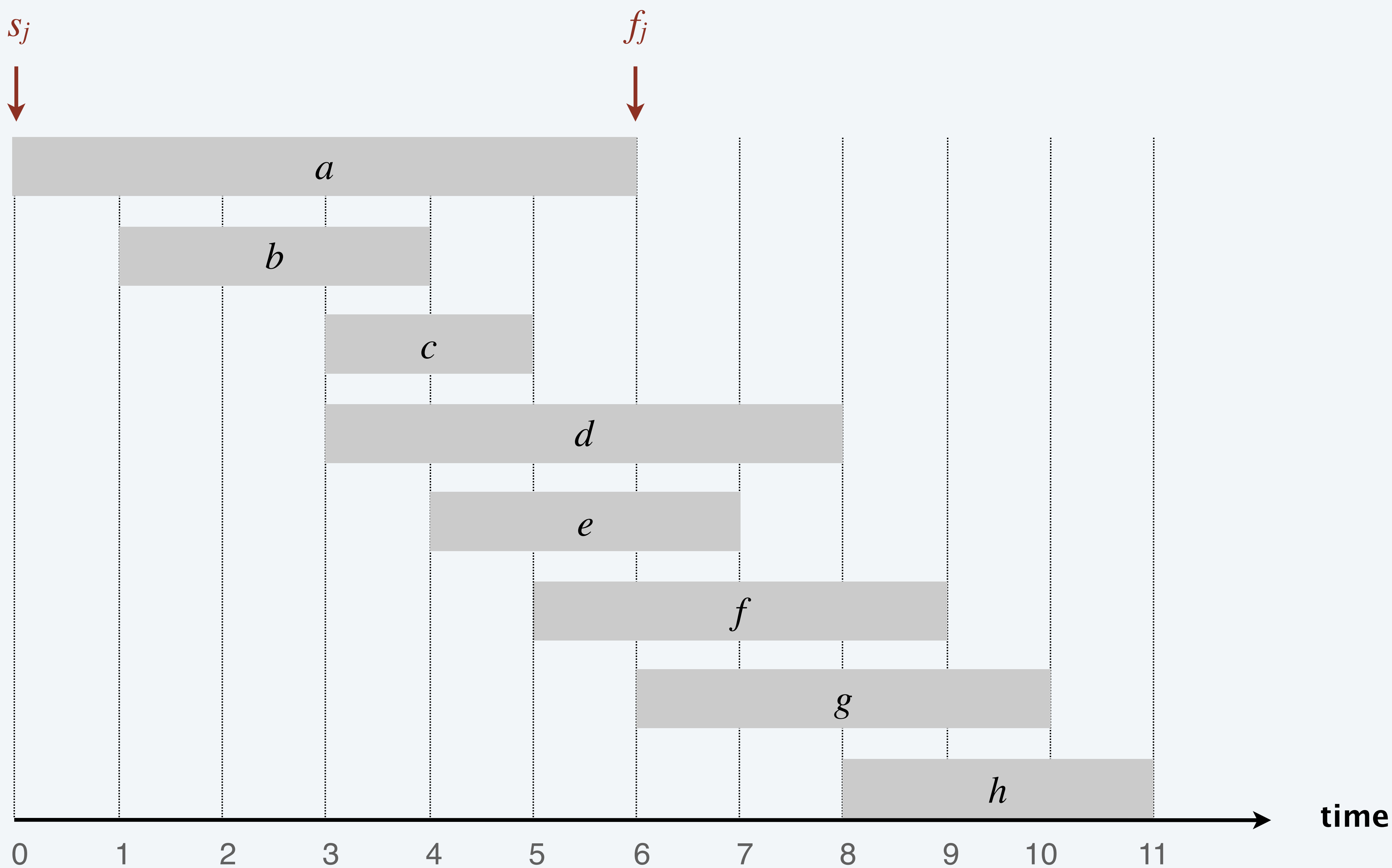
To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time t is determined by a set of quantities which we call state parameters, or state variables. At certain times, which may be prescribed in advance, or which may be determined by the process itself, we are called upon to make decisions which will affect the state of the system. These decisions are equivalent to transformations of the state variables, the choice of a decision being identical with the choice of a transformation. The outcome of the preceding decisions is to be used to guide the choice of future ones, with the purpose of the whole process that of maximizing some function of the parameters describing the final state.

Examples of processes fitting this loose description are furnished by virtually every phase of modern life, from the planning of industrial production lines to the scheduling of patients at a medical clinic; from the determination of long-term investment programs for universities to the determination of a replacement policy for machinery in factories; from the programming of training policies for skilled and unskilled labor to the choice of optimal purchasing and inventory policies for department stores and military establishments.

Weighted interval scheduling

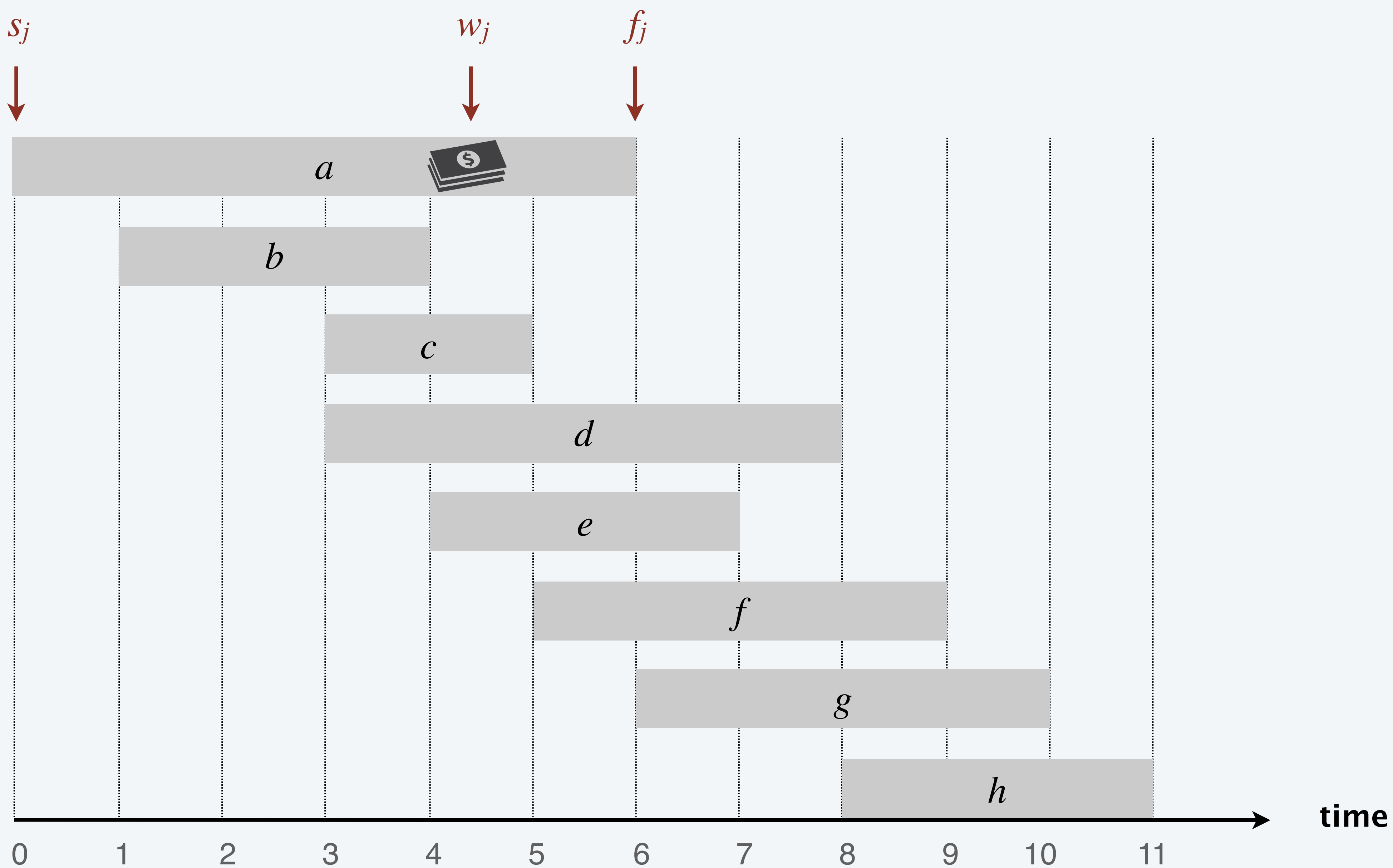
Weighted interval scheduling

- Job j starts at s_j, finishes at f_j, and has weight $w_j > 0$.



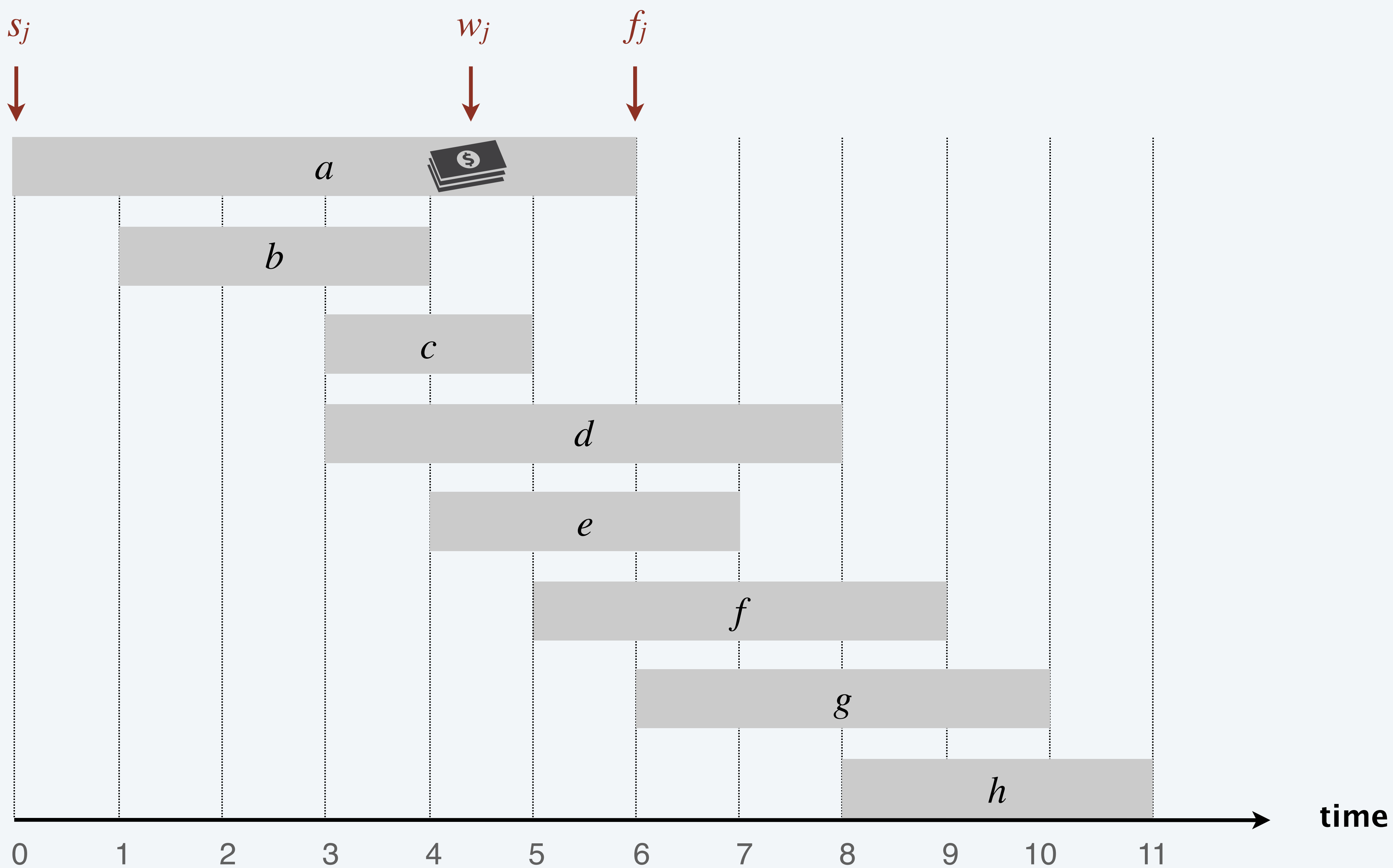
Weighted interval scheduling

- Job j starts at s_j, finishes at f_j, and has weight $w_j > 0$.



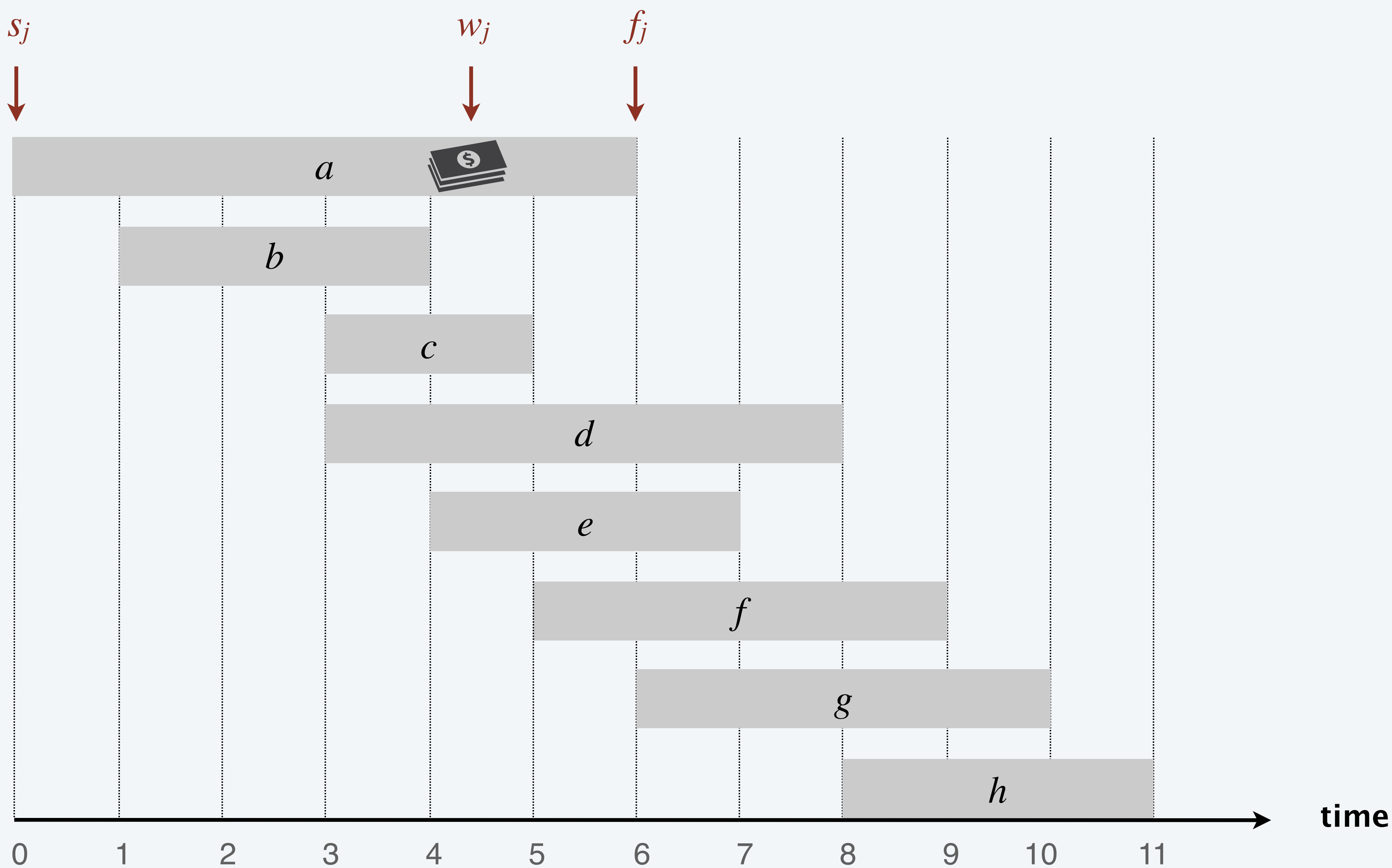
Weighted interval scheduling

- Job j starts at s_j, finishes at f_j, and has weight $w_j > 0$.
- Two jobs are compatible if they don't overlap.



Weighted interval scheduling

- Job j starts at s_j, finishes at f_j, and has weight $w_j > 0$.
- Two jobs are compatible if they don't overlap.
- Goal: find max-weight subset of mutually compatible jobs.



Earliest-finish-time first algorithm

Earliest finish-time first.

- Consider jobs in ascending order of finish time.
- Add job to subset if it is compatible with previously chosen jobs.

Earliest-finish-time first algorithm

Earliest finish-time first.

- Consider jobs in ascending order of finish time.
- Add job to subset if it is compatible with previously chosen jobs.

Recall. Greedy algorithm is correct if all weights are 1.

Earliest-finish-time first algorithm

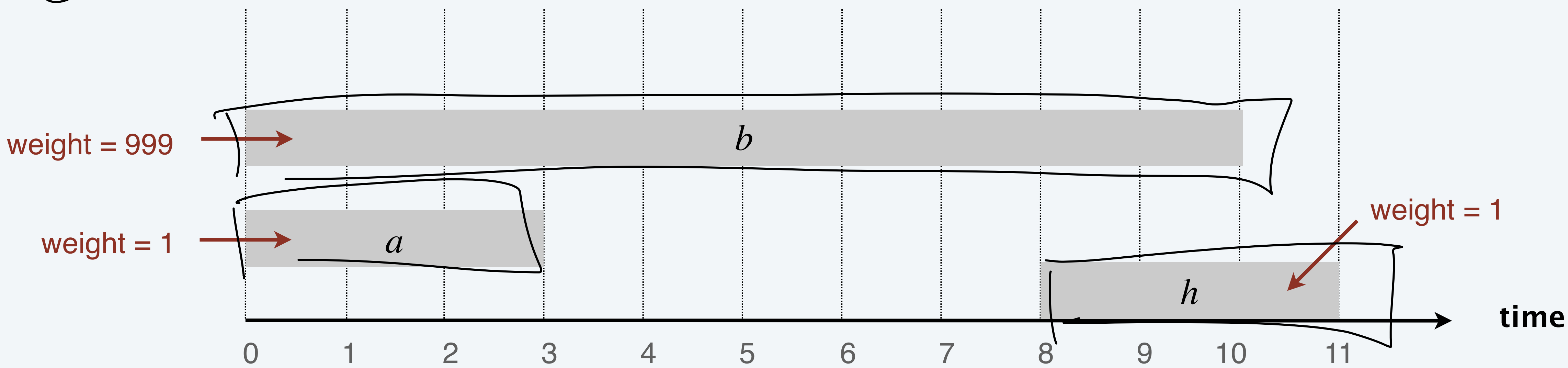
Earliest finish-time first.

- Consider jobs in ascending order of finish time.
- Add job to subset if it is compatible with previously chosen jobs.

Recall. Greedy algorithm is correct if all weights are 1.

Observation. Greedy algorithm fails spectacularly for weighted version.

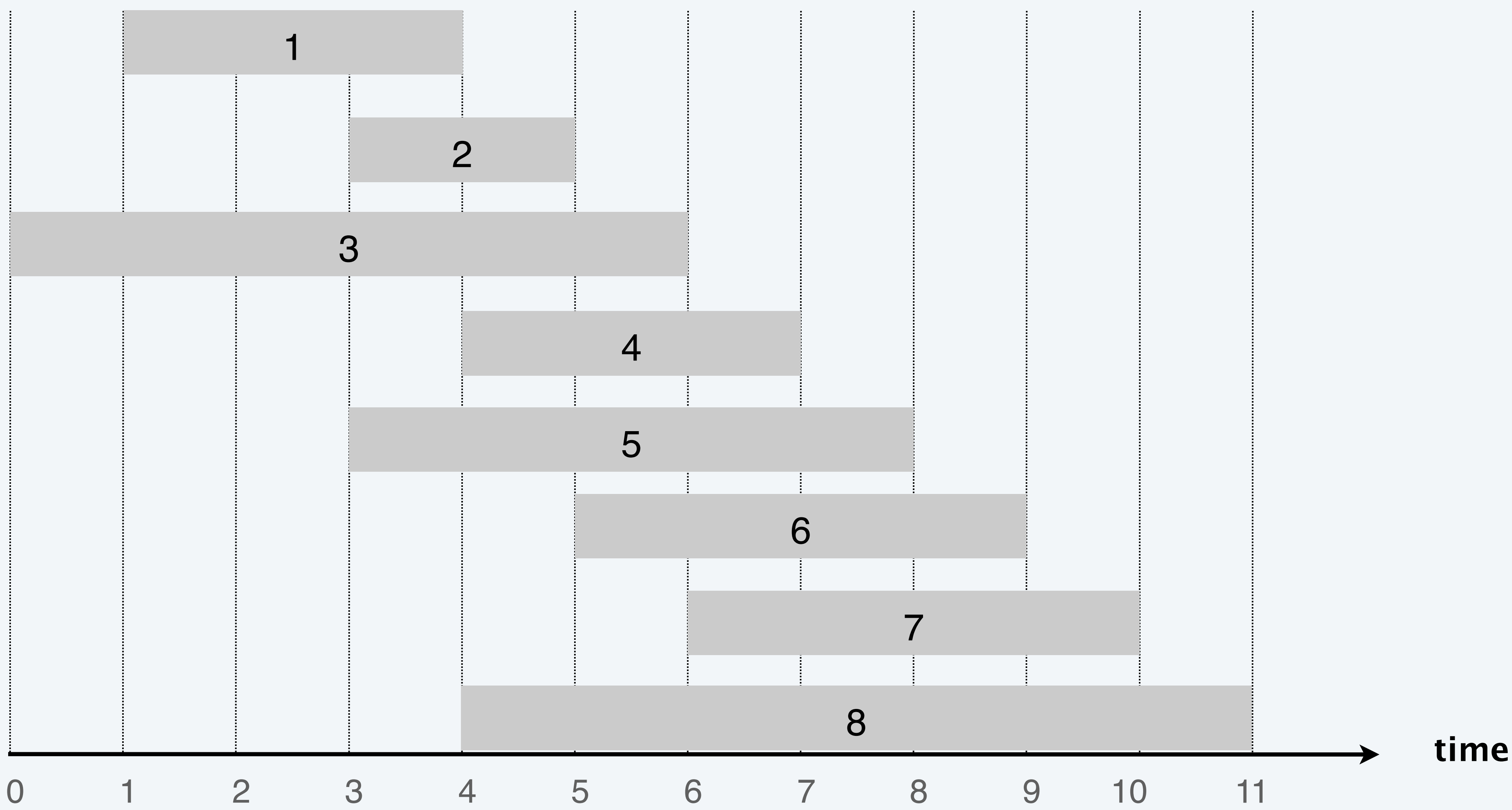
opt = 999
greedy: 2



Weighted interval scheduling

Weighted interval scheduling

Convention. Jobs are in ascending order of finish time: $f_1 \leq f_2 \leq \ldots \leq f_n$.



Weighted interval scheduling

Convention. Jobs are in ascending order of finish time: $f_1 \le f_2 \le \ldots \le f_n$.

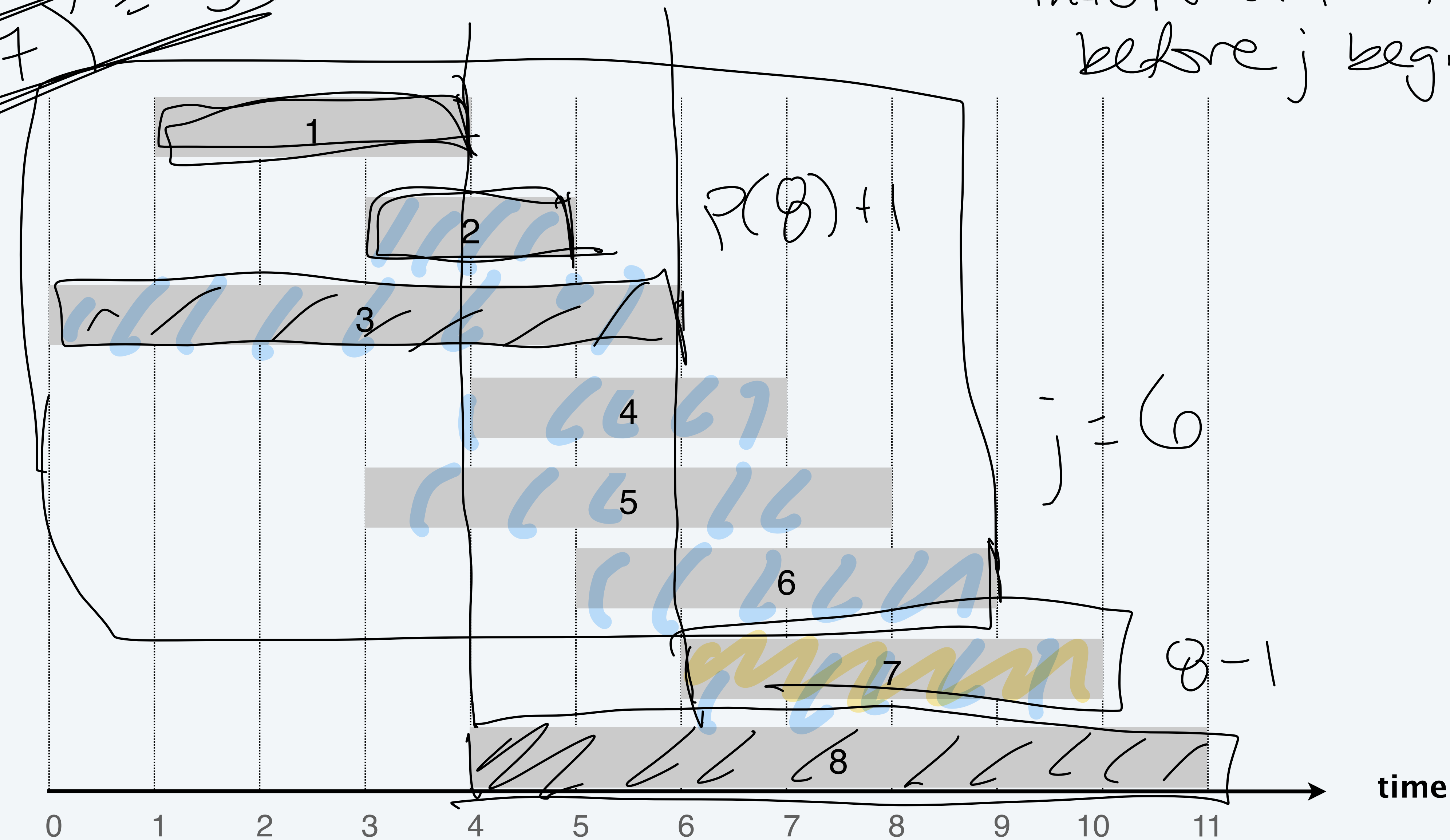
$p(3) = 0 \quad p(2) = 0 \quad p(1) = 0$

Def. $p(j)$ = largest index $i < j$ such that job i is compatible with j.

→ i is the rightmost interval that ends before j begins.

$p(8) = 1$

$\boxed{p(7) = 3}$



Dynamic programming: binary choice

Dynamic programming: binary choice

Def. $OPT(j)$ = max weight of any subset of mutually compatible jobs for subproblem consisting only of jobs $1, 2, ..., j$.

Dynamic programming: binary choice

Def. $OPT(j)$ = max weight of any subset of mutually compatible jobs for subproblem consisting only of jobs $1, 2, ..., j$.

Goal. $OPT(n)$ = max weight of any subset of mutually compatible jobs.

Case 1 Opt(j) does not select job j.

- the optimal solution must use only jobs $1, 2, ..., j-1$

Case 2 opt(j) does select job j.

- we get w_j
- can't use incompatible jobs $p(j)+1, p(j)+2, ..., j-1$
- include optimal solution using remaining compatible jobs $1, 2, ..., p(j)$.

$$OPT(j) = \begin{cases} \max\Big(\underbrace{OPT(j-1)}_{\text{do not select job } j},\ \underbrace{w_j + OPT(p(j))}_{\text{do select job } j}\Big) & j>0 \\ 0 & j=0 \end{cases}$$

Dynamic programming: binary choice

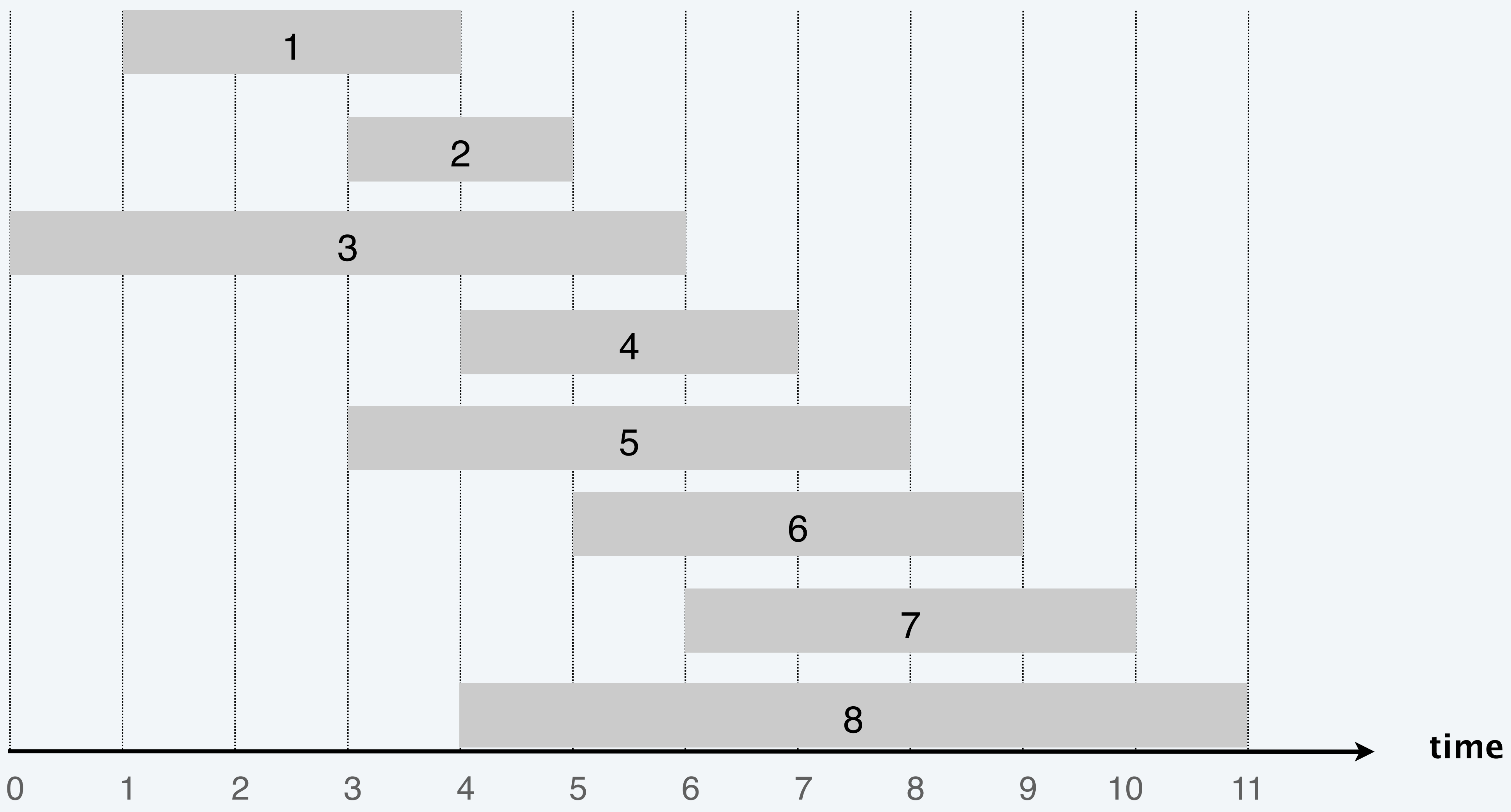
Def. $OPT(j)$ = max weight of any subset of mutually compatible jobs for subproblem consisting only of jobs $1, 2, ..., j$.

Goal. $OPT(n)$ = max weight of any subset of mutually compatible jobs.

Dynamic programming: binary choice

Dynamic programming: binary choice

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{\, OPT(j-1),\ w_j + OPT(p(j)) \,\} & \text{if } j > 0 \end{cases}$$



Weighted interval scheduling: brute force

Weighted interval scheduling: brute force

BRUTE-FORCE $(n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n)$

Sort jobs by finish time and renumber so that $f_1 \le f_2 \le \ldots \le f_n$.

Compute $p[1], p[2], \ldots, p[n]$.

RETURN COMPUTE-OPT(n).

Weighted interval scheduling: brute force

$\text{BRUTE-FORCE}\ (n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n)$

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

Compute $p[1], p[2], \ldots, p[n]$.

RETURN $\text{COMPUTE-OPT}(n)$.

$\text{COMPUTE-OPT}(j)$

IF $(j = 0)$

  RETURN 0.

ELSE

  RETURN $\max\ \{\text{COMPUTE-OPT}(j-1),\ w_j + \text{COMPUTE-OPT}(p[j])\ \}$.