Greedy Algorithms

An activity-selection problem

Introduction

- Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step.
- For many optimization problems, using dynamic programming to determine the best choices is overkill; simpler, more efficient algorithms will do.
- A ***greedy algorithm*** always makes the choice that looks best at the moment.
- That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \ldots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \ldots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}

Each activity a_i has a ***start time***, s_i

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \ldots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	s_1 = 1	s_2 = 3	s_3 = 0	s_4 = 5	s_5 = 3	s_6 = 5	s_7 = 6	s_8 = 8	s_9 = 8	s_{10} = 2	s_{11} = 12

Each activity a_i has a ***start time***, s_i

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \dots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$

Each activity a_i has a ***start time***, s_i , $\mathbf{0 \leq s_i}$

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \ldots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$

Each activity a_i has a ***start time***, s_i , $\mathbf{0 \leq s_i}$

Each activity a_i has a ***finish time***, f_i

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \dots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Each activity a_i has a ***start time***, s_i , $\mathbf{0} \leq \boldsymbol{s_i}$

Each activity a_i has a ***finish time***, f_i

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \dots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Each activity a_i has a ***start time***, s_i , $\mathbf{0 \le s_i}$

Each activity a_i has a ***finish time***, f_i , $\boldsymbol{s_i < f_i}$

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \dots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Each activity a_i has a ***start time***, s_i , $\mathbf{0} \leq \boldsymbol{s_i}$

Each activity a_i has a ***finish time***, f_i , $\boldsymbol{s_i} < \boldsymbol{f_i}$

$\mathbf{0} \leq \boldsymbol{s_i} < \boldsymbol{f_i} < \infty$

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \ldots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Each activity a_i has a ***start time***, s_i , $\mathbf{0} \leq \boldsymbol{s_i}$

Each activity a_i has a ***finish time***, f_i , $\boldsymbol{s_i} < \boldsymbol{f_i}$

$\mathbf{0} \leq \boldsymbol{s_i} < \boldsymbol{f_i} < \infty$

If selected activity $\boldsymbol{a_i}$ takes place during the half-open interval $[\, \boldsymbol{s_i}, \boldsymbol{f_i} \,)$

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \ldots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Each activity a_i has a ***start time***, s_i , $\mathbf{0} \leq \boldsymbol{s_i}$

Each activity a_i has a ***finish time***, f_i , $\boldsymbol{s_i} < \boldsymbol{f_i}$

$\mathbf{0} \leq \boldsymbol{s_i} < \boldsymbol{f_i} < \infty$

If selected activity $\boldsymbol{a_i}$ takes place during the half-open interval $[\,\boldsymbol{s_i}, \boldsymbol{f_i}\,)$

closed-end

open-end

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \ldots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Each activity a_i has a ***start time***, s_i , $\mathbf{0} \leq \boldsymbol{s_i}$

Each activity a_i has a ***finish time***, f_i , $\boldsymbol{s_i} < \boldsymbol{f_i}$

$\mathbf{0} \leq \boldsymbol{s_i} < \boldsymbol{f_i} < \infty$

If selected activity $\boldsymbol{a_i}$ takes place during the half-open interval $[\, \boldsymbol{s_i}, \boldsymbol{f_i} \,)$

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \ldots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Each activity a_i has a ***start time***, s_i , $\mathbf{0} \le \boldsymbol{s_i}$

Each activity a_i has a ***finish time***, f_i , $\boldsymbol{s_i} < \boldsymbol{f_i}$

$\mathbf{0} \le \boldsymbol{s_i} < \boldsymbol{f_i} < \infty$

If selected activity $\boldsymbol{a_i}$ takes place during the half-open interval $[\, \boldsymbol{s_i}, \boldsymbol{f_i} \,)$

An activity-selection problem

- **Problem Statement:** Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Example:** Consider the following set $S = \{a_1, a_2, \dots, a_{11}\}$ of eleven activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Activities are in monotonically increasing order of finish time, $\boldsymbol{f_1 \le f_2 \le \dots \le f_{11}}$

Each activity a_i has a ***start time***, s_i , $\mathbf{0} \le \boldsymbol{s_i}$

Each activity a_i has a ***finish time***, f_i , $\boldsymbol{s_i < f_i}$

$\mathbf{0} \le \boldsymbol{s_i < f_i} < \infty$

If selected activity $\boldsymbol{a_i}$ takes place during the half-open interval $\boldsymbol{[\, s_i , f_i \,)}$

Compatibility of competing activities

- Activities $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***compatible*** if the **intervals** $[\,\boldsymbol{s_i}, \boldsymbol{f_i}\,)$ **and** $[\,\boldsymbol{s_j}, \boldsymbol{f_j}\,)$ **do not overlap**.
- That is, $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***compatible*** if $\boldsymbol{s_i} \geq \boldsymbol{f_j}$ **or** $\boldsymbol{s_j} \geq \boldsymbol{f_i}$.
- Hence, $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***not compatible*** if $\boldsymbol{s_i} < \boldsymbol{f_j}$ **and** $\boldsymbol{s_j} < \boldsymbol{f_i}$

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Compatibility of competing activities

- Activities $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***compatible*** if the **intervals [$\boldsymbol{s_i}$, $\boldsymbol{f_i}$) and [$\boldsymbol{s_j}$, $\boldsymbol{f_j}$)** **do not overlap**.
- That is, $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***compatible*** if $\boldsymbol{s_i \geq f_j}$ **or** $\boldsymbol{s_j \geq f_i}$.
- Hence, $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***not compatible*** if $\boldsymbol{s_i < f_j}$ **and** $\boldsymbol{s_j < f_i}$
- Thus, activities $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ are ***not compatible***
  - since, $\boldsymbol{s_1\ (= 1) < f_2\ (= 5)}$ **and** $\boldsymbol{s_2\ (= 3) < f_1 (= 4)}$

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

Compatibility of competing activities

- Activities $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***compatible*** if the **intervals [$\boldsymbol{s_i}$, $\boldsymbol{f_i}$) and [$\boldsymbol{s_j}$, $\boldsymbol{f_j}$) do not overlap**.
- That is, $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***compatible*** if $\boldsymbol{s_i \geq f_j}$ **or** $\boldsymbol{s_j \geq f_i}$.
- Hence, $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***not compatible*** if $\boldsymbol{s_i < f_j}$ **and** $\boldsymbol{s_j < f_i}$
- Thus, activities $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ are ***not compatible***
  - since, $\boldsymbol{s_1} (= \mathbf{1}) < \boldsymbol{f_2} (= \mathbf{5})$ **and** $\boldsymbol{s_2} (= \mathbf{3}) < \boldsymbol{f_1} (= \mathbf{4})$
- Similarly, activities $\boldsymbol{a_1}$ and $\boldsymbol{a_3}$ are ***not compatible***
  - since, $\boldsymbol{s_1} (= \mathbf{1}) < \boldsymbol{f_3} (= \mathbf{6})$ **and** $\boldsymbol{s_3} (= \mathbf{0}) < \boldsymbol{f_1} (= \mathbf{4})$

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

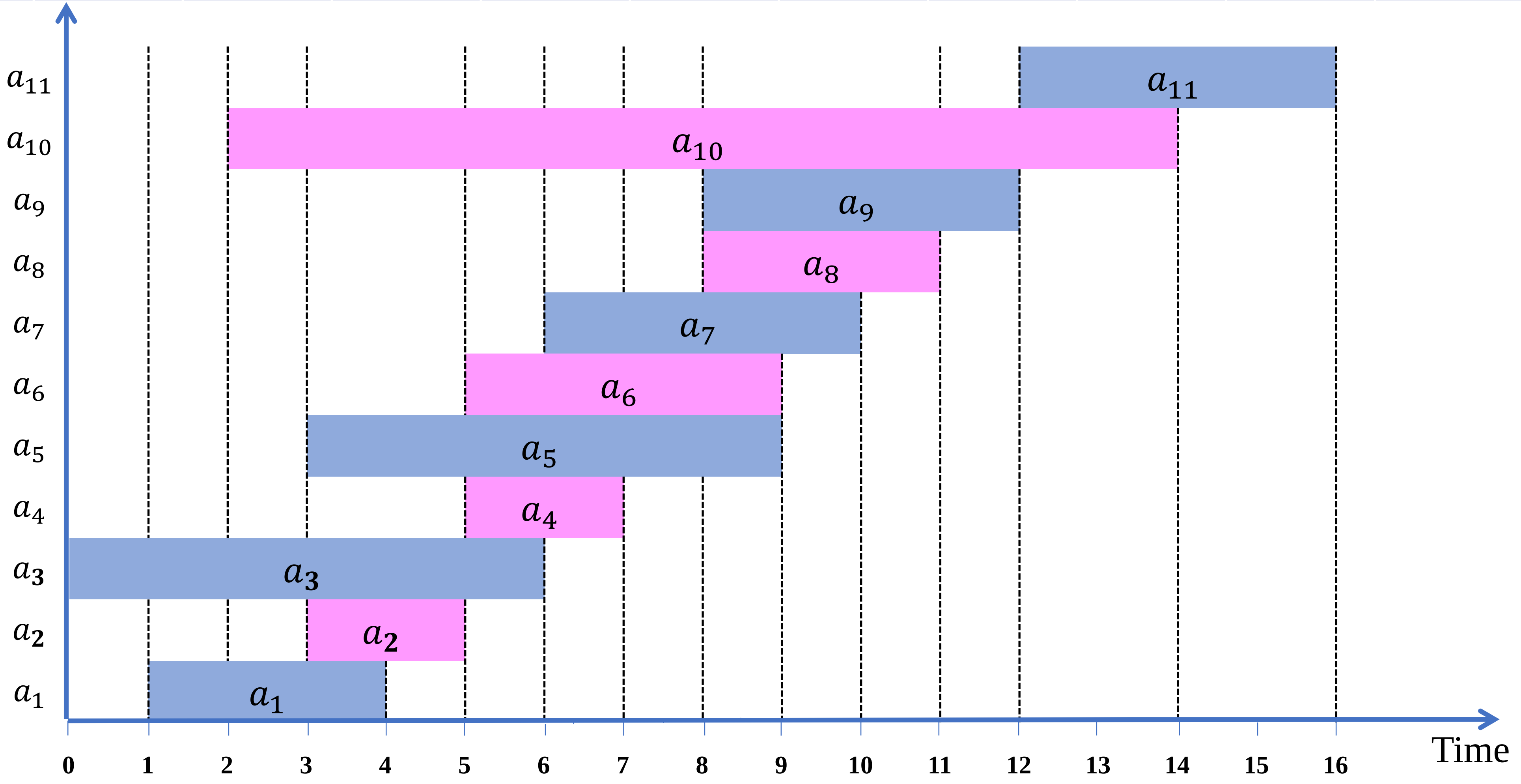
Compatibility of competing activities

- Activities $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***compatible*** if the **intervals [** $\boldsymbol{s_i}$ **,** $\boldsymbol{f_i}$ **) and [** $\boldsymbol{s_j}$ **,** $\boldsymbol{f_j}$ **)** **do not overlap**.
- That is, $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***compatible*** if $\boldsymbol{s_i \geq f_j}$ **or** $\boldsymbol{s_j \geq f_i}$.
- Hence, $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***not compatible*** if $\boldsymbol{s_i < f_j}$ **and** $\boldsymbol{s_j < f_i}$
- Thus, activities $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ are ***not compatible***
    - since, $\boldsymbol{s_1\ (= 1) < f_2\ (= 5)}$ **and** $\boldsymbol{s_2\ (= 3) < f_1 (= 4)}$
- Similarly, activities $\boldsymbol{a_1}$ and $\boldsymbol{a_3}$ are ***not compatible***
    - since, $\boldsymbol{s_1\ (= 1) < f_3\ (= 6)}$ **and** $\boldsymbol{s_3\ (= 0) < f_1 (= 4)}$
- But, activities $\boldsymbol{a_1}$ and $\boldsymbol{a_4}$ are ***compatible***
    - since, $\boldsymbol{s_4\ (= 5) \geq f_1 (= 4)}$

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

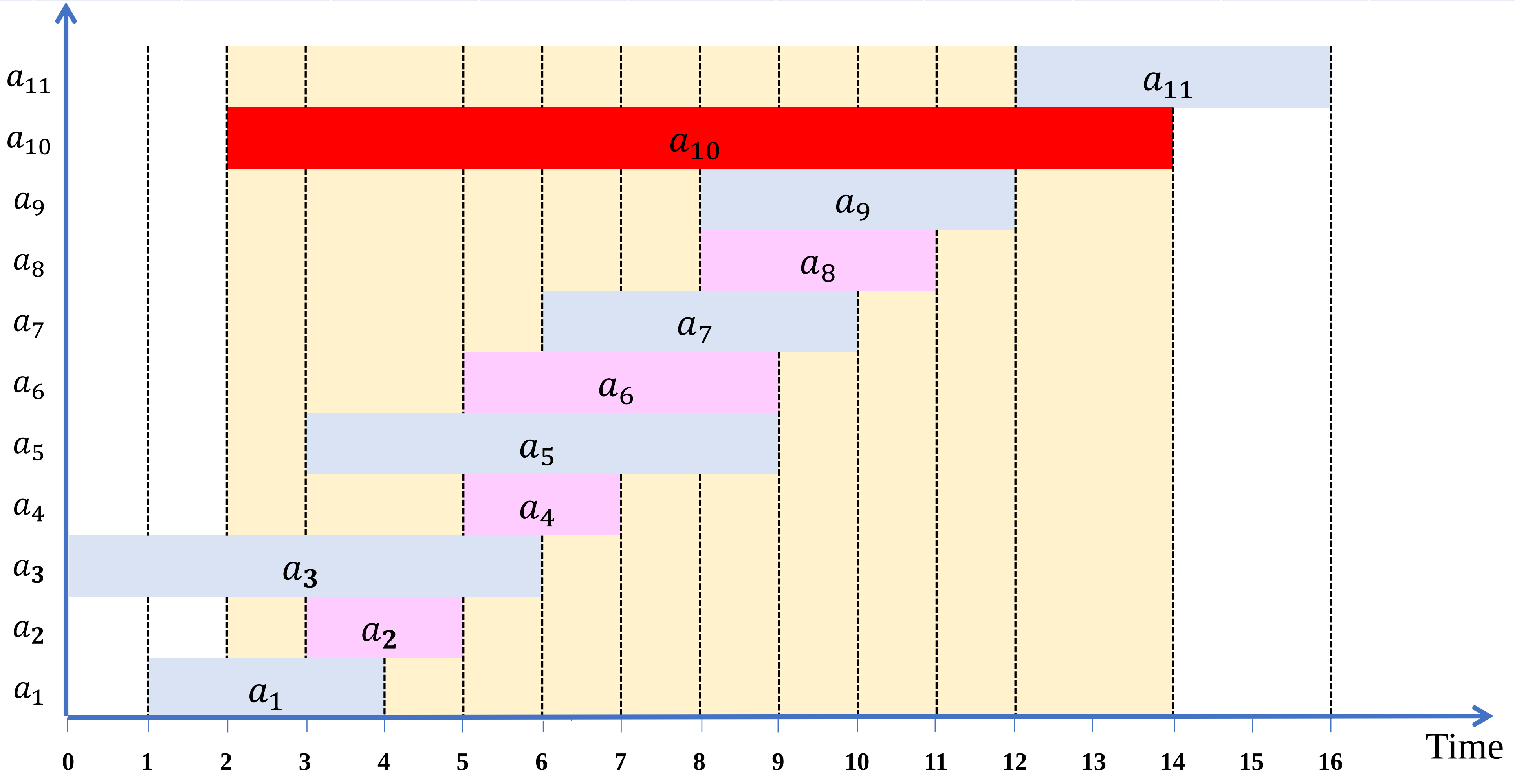
Timeline representation of all competing activities

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$



Timeline representation of all competing activities

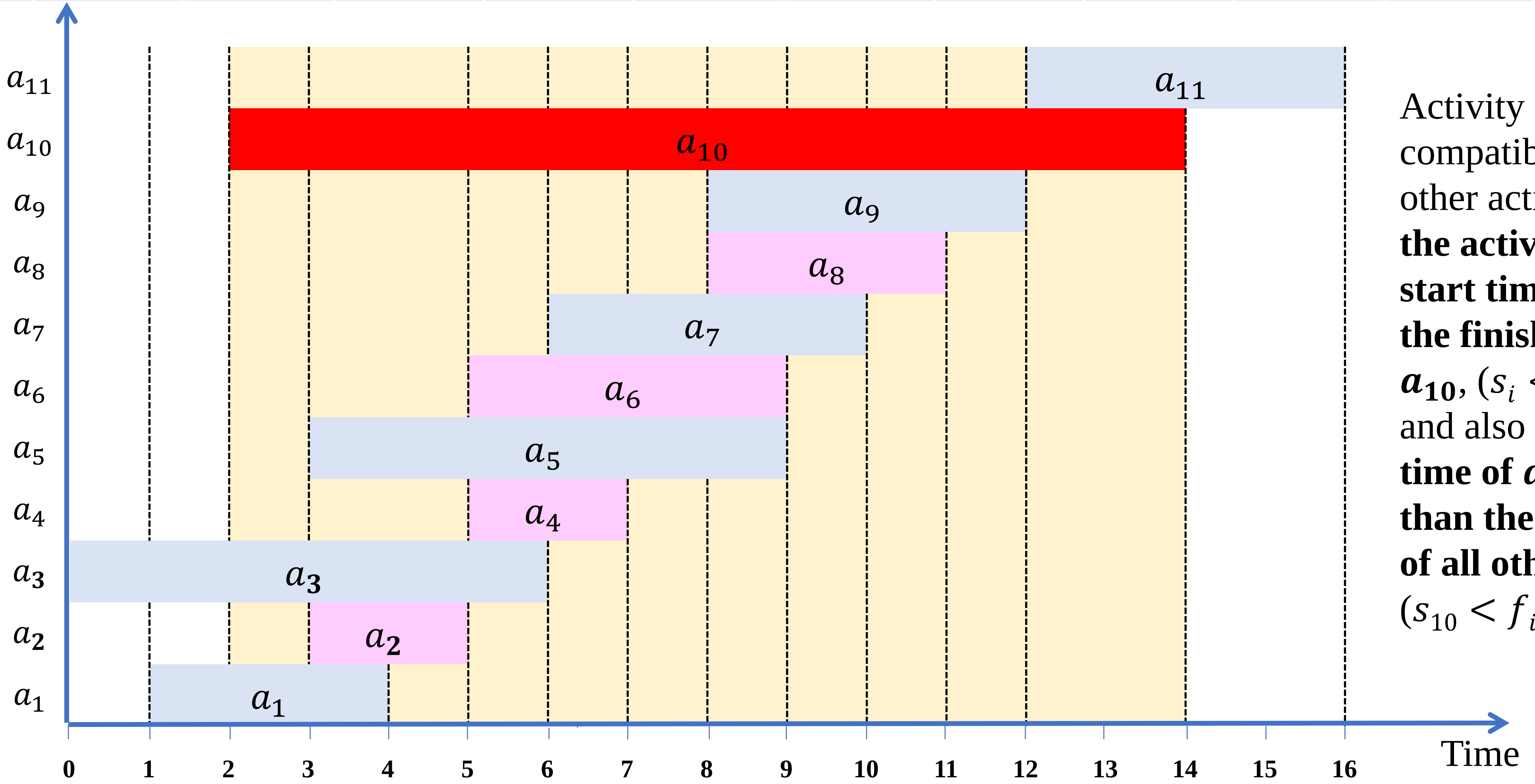
Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$



Compatible set with one activity $\{\boldsymbol{a_{10}}\}$

Timeline representation of all competing activities

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

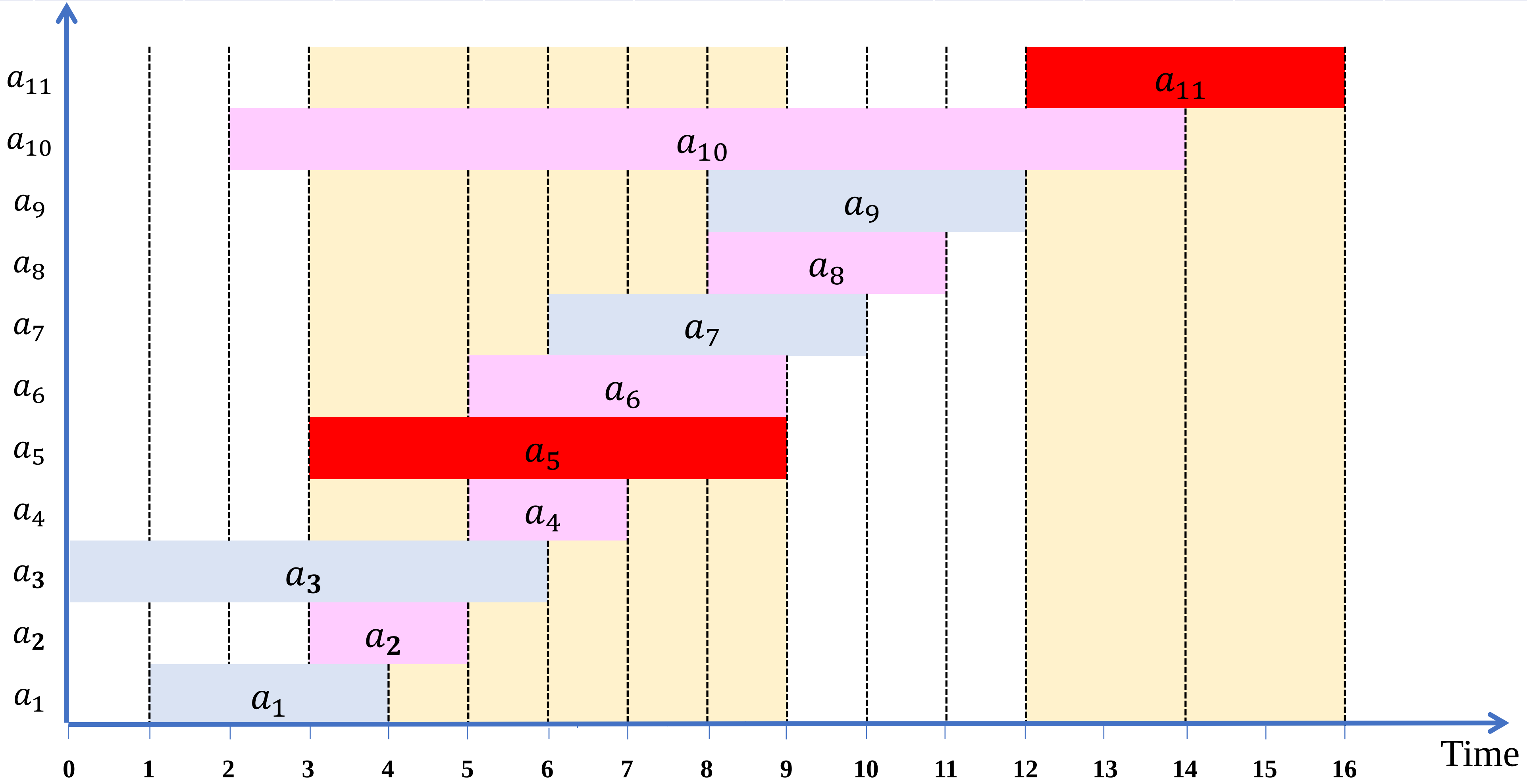


Activity a_{10} is not compatible with any other activity as **all the activities have a start time less than the finish time of $\boldsymbol{a_{10}}$**, $(s_i < f_{10}, \forall i)$ and also **the start time of $\boldsymbol{a_{10}}$, is less than the finish time of all other activities** $(s_{10} < f_i, \forall i)$

Compatible set with one activity $\{\boldsymbol{a_{10}}\}$

Timeline representation of all competing activities

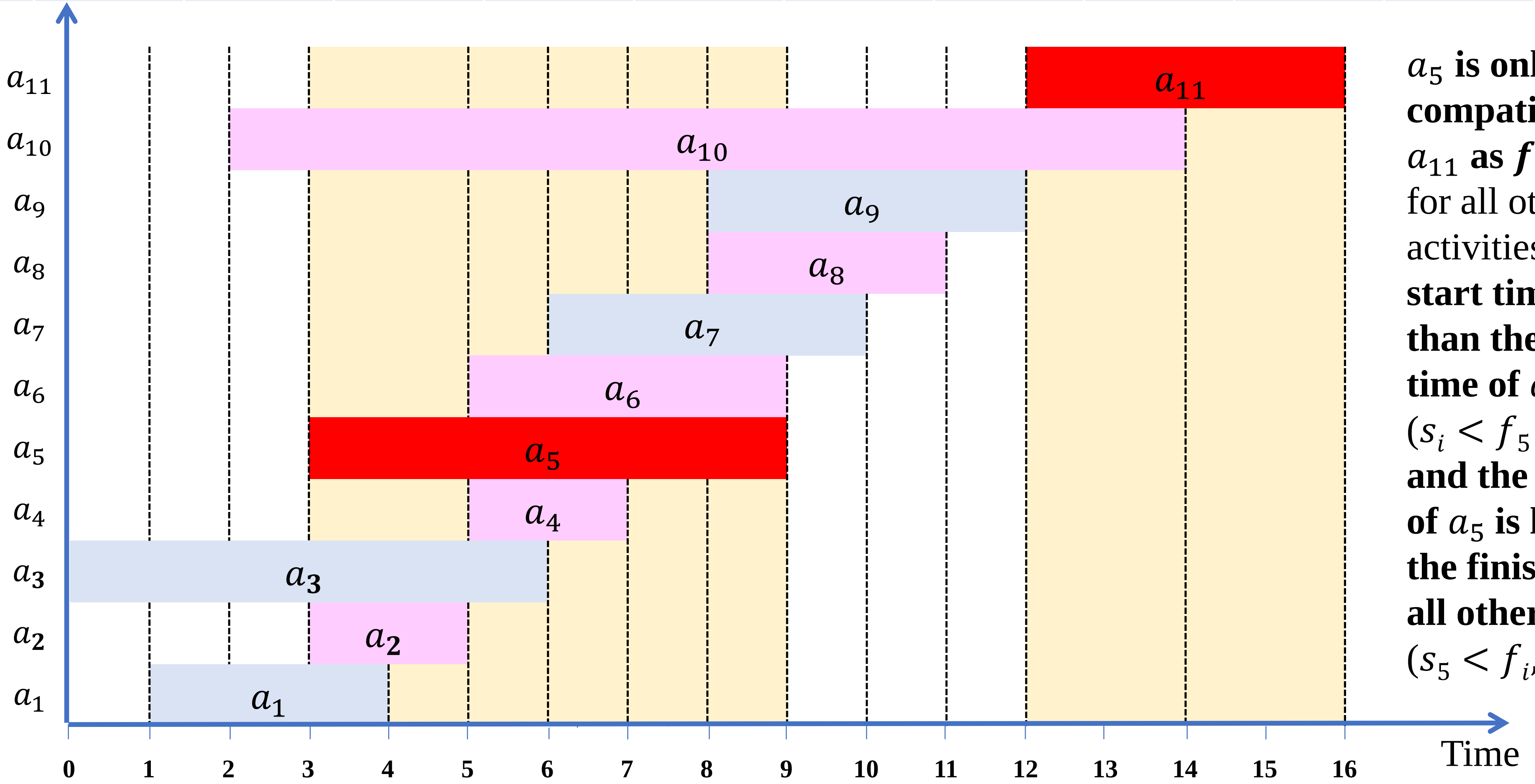
Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$



Compatible set with two activities $\{\boldsymbol{a_5}, \boldsymbol{a_{11}}\}$

Timeline representation of all competing activities

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

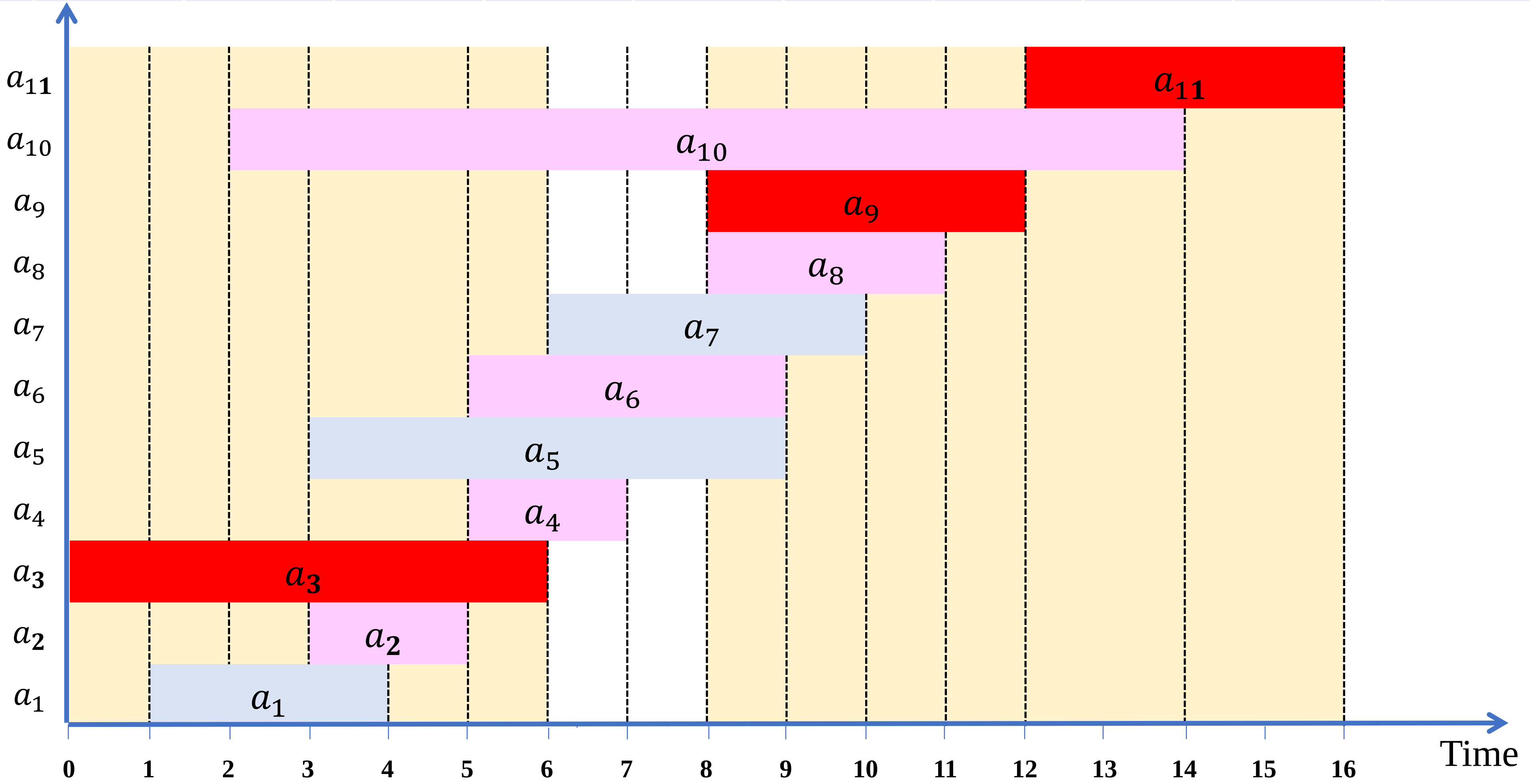


a_5 **is only compatible with** a_{11} **as** $\boldsymbol{f_5 < s_{11}}$**,** for all other activities **their start time is less than the finish time of** a_5 $(s_i < f_5, \forall i \leq 10)$ **and the start time of** a_5 **is less than the finish time of all other activities** $(s_5 < f_i, \forall i)$

Compatible set with two activities $\{\boldsymbol{a_5}, \boldsymbol{a_{11}}\}$

Timeline representation of all competing activities

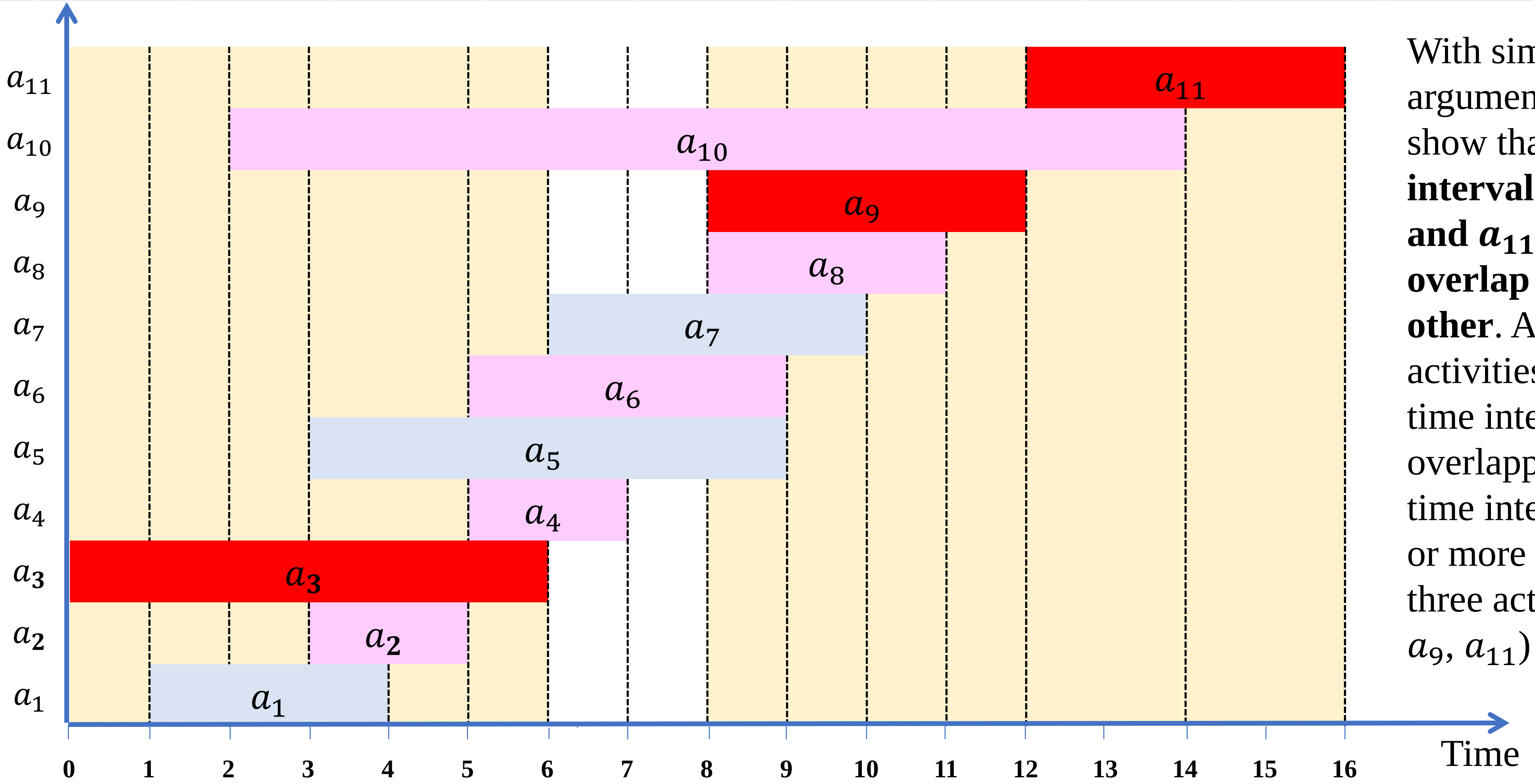
Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$



Compatible set with three activities $\{\boldsymbol{a_3}, \boldsymbol{a_9}, \boldsymbol{a_{11}}\}$

Timeline representation of all competing activities

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$

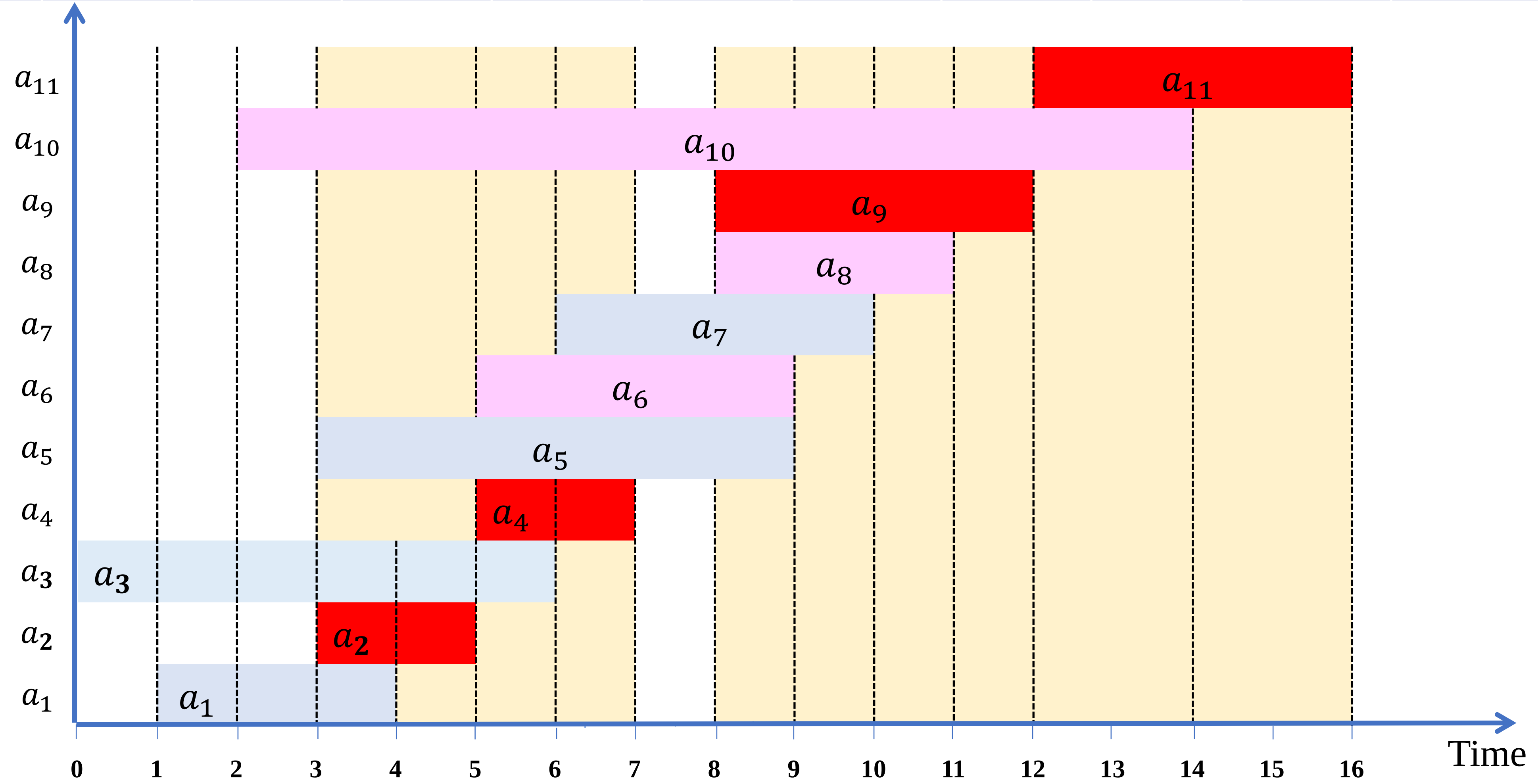


With similar arguments we can show that the **time intervals of a_3, a_9, and a_{11} do not overlap with each other**. All other activities have their time intervals overlapping with the time intervals of one or more of these three activities (a_3, a_9, a_{11})

Compatible set with three activities $\{\boldsymbol{a_3}, \boldsymbol{a_9}, \boldsymbol{a_{11}}\}$

Timeline representation of all competing activities

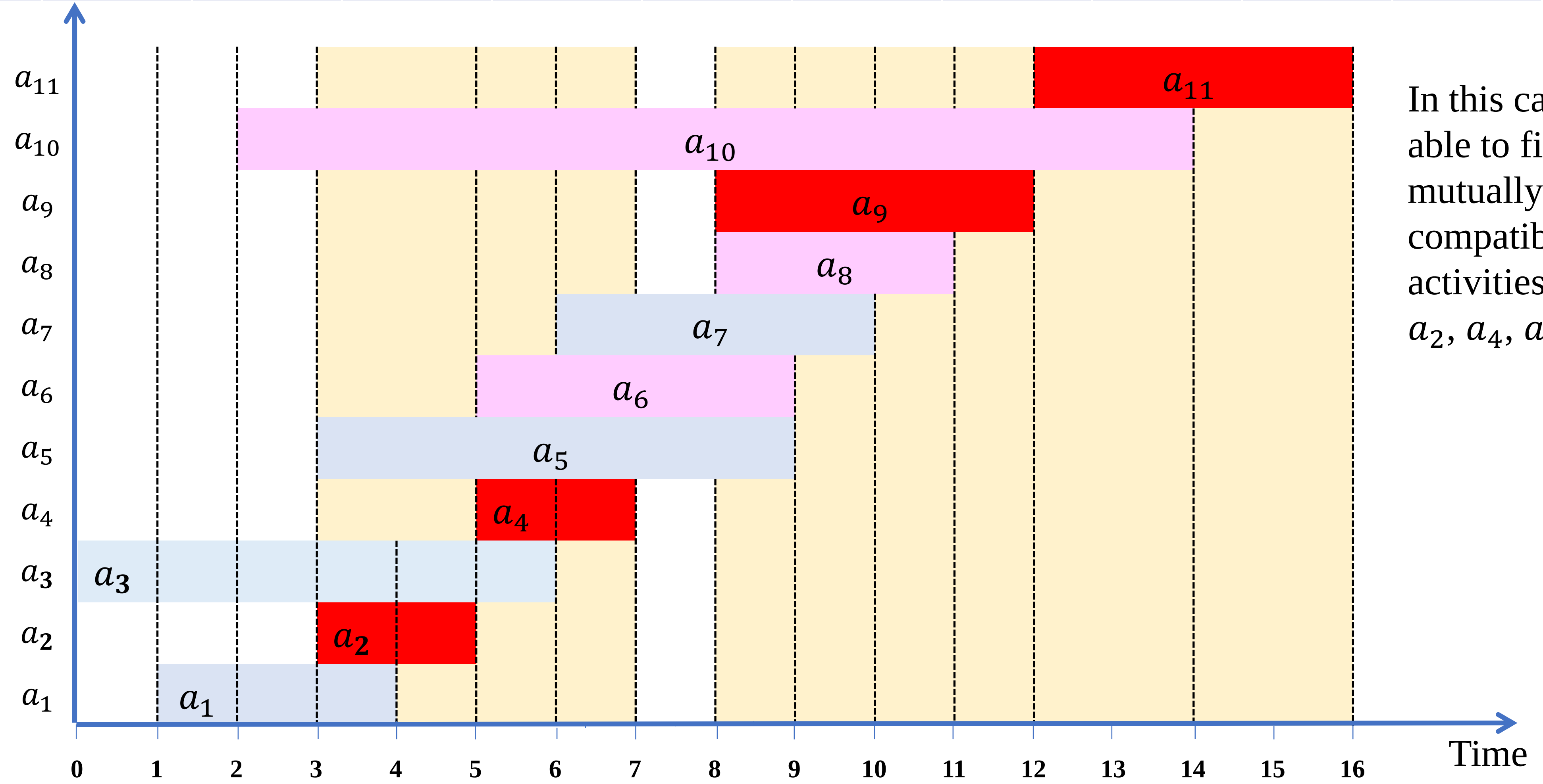
Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$



Compatible set with four activities $\{\boldsymbol{a_2}, \boldsymbol{a_4}, \boldsymbol{a_9}, \boldsymbol{a_{11}}\}$

Timeline representation of all competing activities

Activity, a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time, s_i	$s_1 = 1$	$s_2 = 3$	$s_3 = 0$	$s_4 = 5$	$s_5 = 3$	$s_6 = 5$	$s_7 = 6$	$s_8 = 8$	$s_9 = 8$	$s_{10} = 2$	$s_{11} = 12$
Finish time, f_i	$f_1 = 4$	$f_2 = 5$	$f_3 = 6$	$f_4 = 7$	$f_5 = 9$	$f_6 = 9$	$f_7 = 10$	$f_8 = 11$	$f_9 = 12$	$f_{10} = 14$	$f_{11} = 16$



In this case we are able to find four mutually compatible activities, namely, a_2, a_4, a_8, and a_{11}.

Compatible set with four activities $\{\boldsymbol{a_2}, \boldsymbol{a_4}, \boldsymbol{a_9}, \boldsymbol{a_{11}}\}$

An activity-selection problem

- In the preceding slides we see that we can have different sets of mutually compatible activities.
- So, what is it that we want to achieve?
- We want to obtain the ***maximum-size subset of mutually compatible activities***.
- Now, let us check the details of the activity-selection problem once again.

An activity-selection problem

- **Problem Statement:**
  - Scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- **Input:**
  - A set $\boldsymbol{S} = \{\boldsymbol{a_1}, \boldsymbol{a_2}, \dots, \boldsymbol{a_n}\}$ of $\boldsymbol{n}$ proposed ***activities*** that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.
  - Each activity a_i has a ***start time***, $\boldsymbol{s_i}$ and a **finish time**, $\boldsymbol{f_i}$, $\boldsymbol{0 \leq s_i < f_i < \infty}$.
- **Output:**
  - A ***maximum-size subset of mutually compatible activities***.
- **Assumption:**
  - The activities are sorted in monotonically increasing order of finish time $\boldsymbol{f_1 \leq f_2 \leq \dots \leq f_n}$

To find the solution to the activity-selection problem

- We first look into the dynamic programming solution and provide the ***optimal substructure***.
- Then we will delve into the possibility of making a ***greedy choice*** from the optimal substructure and check its correctness.
- Next, provide a recursive greedy algorithm.
- Finally, give an iterative greedy algorithm for the activity-selection problem.

The Optimal Substructure of the activity-selection problem

Let

- S_{ij}: Set of activities that **start after activity a_i finishes** and **finish before the activity a_j starts**, such that

$\boldsymbol{S_{ij} = \{a_{i+1}, a_{i+2}, \dots, a_{j-2}, a_{j-1}\}}$

- A_{ij}: **Maximum-size set of mutually compatible activities in S_{ij}** , which is an optimal solution of S_{ij}.

Let a_k be some activity in the set S_{ij}, such that, a_k starts after activity a_i finishes and finishes before activity a_j starts.

$$\boldsymbol{S_{ij} = \{a_{i+1}, a_{i+2}, \dots, a_{k-2}, a_{k-1}, a_k, a_{k+1}, a_{k+2}, \dots, a_{j-2}, a_{j-1}\}.}$$

If now, a_k **is included in the optimal solution, A_{ij}** , then we are left with two subproblems: finding mutually compatible activities in the set, $\boldsymbol{S_{ik}}$ and finding mutually compatible activities in the set $\boldsymbol{S_{kj}}$, where,

- **Subproblem, $S_{ik} = \{a_{i+1}, a_{i+2}, \dots, a_{k-2}, a_{k-1}\}$**, set of activities that **start after activity a_i finishes** and **finish before the activity a_k starts.**
- **Subproblem, $S_{kj} = \{a_{k+1}, a_{k+2}, \dots, a_{j-2}, a_{j-1}\}$**, set of activities that **start after activity a_k finishes** and **finish before the activity a_j starts.**

The Optimal Substructure of the activity-selection problem

Now let,

- $\boldsymbol{A_{ik}}$: **Maximum-size set of mutually compatible activities in $\boldsymbol{S_{ik}}$** , which is an optimal solution to the subproblem, S_{ik} .
- $\boldsymbol{A_{ik} = A_{ij} \cap S_{ik}}$, so that A_{ik} contains the activities in A_{ij} that finish before a_k starts.

and

- $\boldsymbol{A_{kj}}$: **Maximum-size set of mutually compatible activities in $\boldsymbol{S_{kj}}$** , which is an optimal solution for the subproblem, S_{kj}.
- $\boldsymbol{A_{kj} = A_{ij} \cap S_{kj}}$, so that A_{kj} contains the activities in A_{ij} that start after a_k finishes.

Thus,

$$
\begin{aligned}
A_{ij} &= \{\ldots, a_k, \ldots\} \\
&= A_{ik} \cup \{a_k\} \cup A_{kj} \\
&= \{\text{Optimal solution to subproblem } S_{ik}\} \cup \{a_k\} \cup \{\text{Optimal solution to subproblem } S_{kj}\} \\
&= (A_{ij} \cap S_{ik}) \cup \{a_k\} \cup (A_{ij} \cap S_{kj})
\end{aligned}
$$

So the maximum-size set A_{ij} of mutually compatible activities in S_{ij} consists of

$$|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$$

The Optimal Substructure of the activity-selection problem

The optimal solution $\boldsymbol{A_{ij}}$ must also include optimal solution to the two subproblems for $\boldsymbol{S_{ik}}$ and $\boldsymbol{S_{kj}}$.

Proof:

Let us assume that there is a set $\boldsymbol{A_{kj}}'$ of mutually compatible activities such that $|A_{kj}'| > |A_{kj}|$, then A_{kj}' would be the optimal solution to the subproblem for S_{kj} rather than A_{kj}.

Therefore, the solution to the subproblem S_{ij} can be constructed as $|A_{ik}| + |A_{kj}'| + 1$ of mutually compatible activities in S_{ij}.

But, $|A_{ik}| + |A_{kj}'| + 1 > |A_{ik}| + |A_{kj}| + 1 = |A_{ij}|$ (since, $|A_{kj}'| > |A_{kj}|$)

This contradicts the assumption that A_{ij} is an optimal solution for S_{ij}.

Therefore, A_{kj} is the optimal solution for S_{kj}.

A symmetric argument applies to the activities in S_{ik}.

The Optimal Substructure of the activity-selection problem

This way of characterizing optimal substructure suggests that we might solve the activity-selection problem by dynamic programming.

Let $c[i, j]$ denote the size of an optimal solution for the set S_{ij},

hence the recurrence,

$$\boldsymbol{c[i, j] = c[i, k] + c[k, j] + 1}$$

But, if we did not know that an optimal solution for the set S_{ij}, includes activity a_k, we would have to examine all activities in S_{ij} to find which one to choose, so that

$$c[i, j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset, \\ \max\limits_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{if } S_{ij} \neq \emptyset. \end{cases}$$

We could then develop a recursive algorithm and memoize it to obtain a dynamic programming solution.

But can we do better?

Making the greedy choice

- What if we could choose an activity to add to our optimal solution without having to first solve all the subproblems?
- It will save us from having to consider all the choices inherent in the recurrence given in the previous slide.
- In fact, for the activity-selection problem, we need consider only one choice: ***the greedy choice***.

Making the greedy choice

- What do we mean by a ***greedy choice*** for the activity-selection problem?
- Intuition says that we should choose an activity that leaves the resource available for as many other activities as possible.
- Hence, if we ***choose the activity having the earliest finish time*** we are leaving the common resource available for the maximum possible time for other activities that follow the selected activity.
- Since the activities are sorted in monotonically increasing order of finish time, ***the first greedy choice is activity*** $\boldsymbol{a_1}$.

Making the greedy choice

- Making the greedy choice, $\boldsymbol{a_1}$, we have only one remaining subproblem to solve: *finding activities that start after* $\boldsymbol{a_1}$ *finishes*.
- We do not have to consider the activities that finish before $\boldsymbol{a_1}$ starts because
    - $s_1 < f_1$ i.e., for any activity start time is less than its finish time.
    - f_1 is the earliest finish time of any activity, $f_1 \leq f_2 \leq \ \dots \leq f_{11}$ since activities are sorted in the order of their finish time.
    - Therefore, $s_1 < f_1 \leq f_2 \leq \ \dots \leq f_{11}$ i.e., no activity can have a finish time less than or equal to s_1.
    - Thus, all activities that are compatible with activity a_1 must start after a_1 finishes.

Making the greedy choice

Let $S_k = \{a_i \in S : s_i \geq f_k\}$ be the set of activities that start after activity a_k finishes.

As a_1 has the earliest finish time we make a greedy choice of activity a_1, then S_1 (containing all activities that start after a_1 finishes) remains as the only subproblem to solve.

Now, activity-selection problem exhibits optimal substructure.

Hence, if a_1 is in the optimal solution, then an optimal solution to the original problem consists of activity a_1 and all the activities in an optimal solution to the subproblem S_1.

But, is the greedy choice always part of some optimal solution?

The following theorem shows that it is.

Correctness of making the greedy choice

***Theorem*:**

Consider any nonempty subproblem S_k and let a_m be an activity in S_k with the earliest finish time. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k.

Proof: Let A_k be a maximum-size subset of mutually compatible activities of S_k, and a_j be the activity in A_k with the earliest finish time.

If $a_j = a_m$, then we have shown that the activity a_m, which has the earliest finish time in S_k, is included in some maximum-size subset of mutually compatible activities of S_k.

If $a_j \neq a_m$, then, $f_m \leq f_j$, given a_m has the earliest finish time in S_k.

Let the set $A_k' = A_k - \{a_j\} \cup \{a_m\}$ (obtained by substituting a_m for a_j in A_k).

Activities in A_k are disjoint since it contains mutually compatible activities. (Activities $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are ***compatible*** if the intervals $\boldsymbol{[\, s_i\,,\, f_i\,)}$ and $\boldsymbol{[\, s_j\,,\, f_j\,)}$ do not overlap.) Hence it follows that the activities in A_k' are also disjoint ($s_m < fm \leq f_i, \forall a_i \in A_k'$, but $f_m \leq s_i, \forall a_i \in A_k'$).

a_j is the first activity to finish in A_k, and $f_m \leq f_j$. Thus a_m is the first activity to finish in A_k'.

Since $|A_k'| = |A_k|$ (from construction), we conclude that A_k' is a maximum-size subset of mutually compatible activities of S_k, and it includes a_m.

Making the greedy choice

Thus,

- We do not need a dynamic programming approach.
- We can repeatedly choose the activity that finishes first.
- Select only the activities compatible with the chosen activity.
- And repeat until no activities remain.
- Because we always choose the activity with the earliest finish time, the finish time of the activities that we choose must strictly increase.
- Each activity can be considered just once overall, in monotonically increasing order of finish time.
- Greedy algorithms typically have this top-down design: **make a choice and then solve a subproblem**, rather than the bottom-up technique of solving subproblems before making a choice.

A Recursive Greedy Algorithm to solve the Activity-selection problem

Input:

1. Set, $S = \{a_1, a_2, \ldots, a_n\}$ of n activities that wish to use a common resource, which can serve only one activity at a time.
2. Array ***s*** that contains the start time of the activities.
3. Array ***f*** that contains the finish time of the activities.
4. Index ***k*** that defines the subproblem $\boldsymbol{S_k}$ it is to solve.
5. The size of the original subproblem, ***n***.

Output:

Maximum-size subset of compatible activities of $S = \{a_1, a_2, \ldots, a_n\}$.

Assumption:

The ***n*** input activities are sorted by monotonically increasing finish time , $\boldsymbol{f_1 \leq f_2 \leq \ldots \leq f_n}$. If not sorted we can sort them in this order in $O(n\ lg\ n)$ time, breaking ties arbitrarily.

Initially:

We add the fictitious activity a_0 with $\boldsymbol{f_0} = 0$, so that subproblem $\boldsymbol{S_0}$ is the entire set of activities in S.

A Recursive Greedy Algorithm to solve the Activity-selection problem

Initial call: RECURSIVE-ACTIVITY-SELECTOR $(s, f, 0, n)$

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)

$m = k + 1$
while $m \leq n$ **and** $s[m] < f[k]$
    $m = m + 1$
if $m \leq n$
    return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)
else return $\emptyset$

A Recursive Greedy Algorithm to solve the Activity-selection problem

Initial call: RECURSIVE-ACTIVITY-SELECTOR $(s, f, 0, n)$

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)

$m = k + 1$

Sets m to the index of the activity that comes after a_k in the monotonically increasing order of finish time.

while $m \leq n$ **and** $s[m] < f[k]$

  $m = m + 1$

if $m \leq n$

  return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)

else return $\emptyset$

A Recursive Greedy Algorithm to solve the Activity-selection problem

Initial call: RECURSIVE-ACTIVITY-SELECTOR $(s, f, 0, n)$

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)

1 $m = k + 1$

2 **while** $m \leq n$ **and** $s[m] < f[k]$

3 $\quad\quad m = m + 1$

4 **if** $m \leq n$

5 $\quad\quad$ **return** $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)

6 **else return** $\emptyset$

Sets m to the index of the activity that comes after a_k in the monotonically increasing order of finish time.

The while loop of lines 2-3 looks for the first activity in S_k to finish by examining $a_{k+1}, a_{k+2}, \ldots a_n$, until it finds the first activity a_m that is compatible with a_k, i.e., $s_m \geq f_k$.

A Recursive Greedy Algorithm to solve the Activity-selection problem

Initial call: RECURSIVE-ACTIVITY-SELECTOR $(s, f, 0, n)$

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)

1 $m = k + 1$

2 **while** $m \leq n$ **and** $s[m] < f[k]$

3 $m = m + 1$

4 **if** $m \leq n$

5 **return** $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)

6 **else return** $\emptyset$

Sets m to the index of the activity that comes after a_k in the monotonically increasing order of finish time.

The while loop of lines 2-3 looks for the first activity in S_k to finish by examining $a_{k+1}, a_{k+2}, \ldots a_n$, until it finds the first activity a_m that is compatible with a_k, i.e., $s_m \geq f_k$.

If the loop terminates because it finds an activity, a_m which is compatible with a_k, i.e., $s_m \geq f_k$, line 5 returns the union of $\{a_m\}$ and the maximum-size subset of compatible activities in S_m returned by the recursive call RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n).

A Recursive Greedy Algorithm to solve the Activity-selection problem

Initial call: RECURSIVE-ACTIVITY-SELECTOR $(s, f, 0, n)$

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)

```
1   m = k + 1
2   while m ≤ n and s[m] < f[k]
3       m = m + 1
4   if m ≤ n
5       return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)
6   else return ∅
```

Sets m to the index of the activity that comes after a_k in the monotonically increasing order of finish time.

The while loop of lines 2-3 looks for the first activity in S_k to finish by examining a_{k+1}, a_{k+2}, … a_n, until it finds the first activity a_m that is compatible with a_k, i.e., $s_m \geq f_k$.

The loop may terminate because $m > n$, which means all activities in S_k have been examined without finding one that is compatible with a_k. Hence, $S_k = \emptyset$.
Procedure returns ∅ in line 6.

Time complexity Analysis of the Recursive Greedy Algorithm

Initial call: RECURSIVE-ACTIVITY-SELECTOR ($s, f, 0, n$)

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)	Times executed
$m = k + 1$	1
while $m \leq n$ **and** $s[m] < f[k]$	$m - k + 1$
$m = m + 1$	$m - k$
if $m \leq n$	1
return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)	≤ 1
else return $\emptyset$	≤ 1

Time complexity Analysis of the Recursive Greedy Algorithm

Initial call: RECURSIVE-ACTIVITY-SELECTOR ($s, f, 0, n$)

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)	Times executed
$m = k + 1$	1
while $m \leq n$ **and** $s[m] < f[k]$	$m - k + 1$
$m = m + 1$	$m - k$
if $m \leq n$	1
return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)	≤ 1
else return $\emptyset$	≤ 1

Recursive call to subproblem S_m which consists of only activities that come after a_m in the order of monotonically increasing finish time. Size of the subproblem $|S_m| = n - m$

Time complexity Analysis of the Recursive Greedy Algorithm

Initial call: RECURSIVE-ACTIVITY-SELECTOR (s, f , 0, n)

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f , k, n)	Times executed
$m = k + 1$	1
while $m \leq n$ **and** $s[m] < f[k]$	$m - k + 1$
$m = m + 1$	$m - k$
if $m \leq n$	1
return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f , m, n)	≤ 1
else return $\emptyset$	≤ 1

Recursive call to subproblem S_m which consists of only activities that come after a_m in the order of monotonically increasing finish time. Size of the subproblem $|S_m| = n - m$

In particular, only $m - k$ activities, that finish after a_k finishes (starting from activity a_{k+1} to activity a_m) are examined in this call in the order of increasing finish time. These activities are not examined in any other recursive call.

Time complexity Analysis of the Recursive Greedy Algorithm

Initial call: RECURSIVE-ACTIVITY-SELECTOR ($s, f, 0, n$)

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)	Times executed
1 $m = k + 1$	1
2 **while** $m \leq n$ **and** $s[m] < f[k]$	$m - k + 1$
3 $\quad m = m + 1$	$m - k$
4 **if** $m \leq n$	1
5 $\quad$ **return** $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)	≤ 1
6 **else return** $\emptyset$	≤ 1

Over all recursive calls, each activity is examined exactly once in the **while** loop test of line 2.

In particular, only $m - k$ activities, that finish after a_k finishes (starting from activity a_{k+1} to activity a_m) are examined in this call in the order of increasing finish time. These activities are not examined in any other recursive call.

Recursive call to subproblem S_m which consists of only activities that come after a_m in the order of monotonically increasing finish time. Size of the subproblem $|S_m| = n - m$

Time complexity Analysis of the Recursive Greedy Algorithm

Initial call: RECURSIVE-ACTIVITY-SELECTOR (s, f, 0, n)

Algorithm:

RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)	Times executed
1 $m = k + 1$	1
2 **while** $m \leq n$ **and** $s[m] < f[k]$	$m - k + 1$
3 $\quad m = m + 1$	$m - k$
4 **if** $m \leq n$	1
5 $\quad$ **return** $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)	≤ 1
6 **else return** $\emptyset$	≤ 1

Over all recursive calls, each activity is examined exactly once in the **while** loop test of line 2.

Recursive call to subproblem S_m which consists of only activities that come after a_m in the order of monotonically increasing finish time. Size of the subproblem $|S_m| = n - m$

In particular, only $m - k$ activities, that finish after a_k finishes (starting from activity a_{k+1} to activity a_m) are examined in this call in the order of increasing finish time. These activities are not examined in any other recursive call.

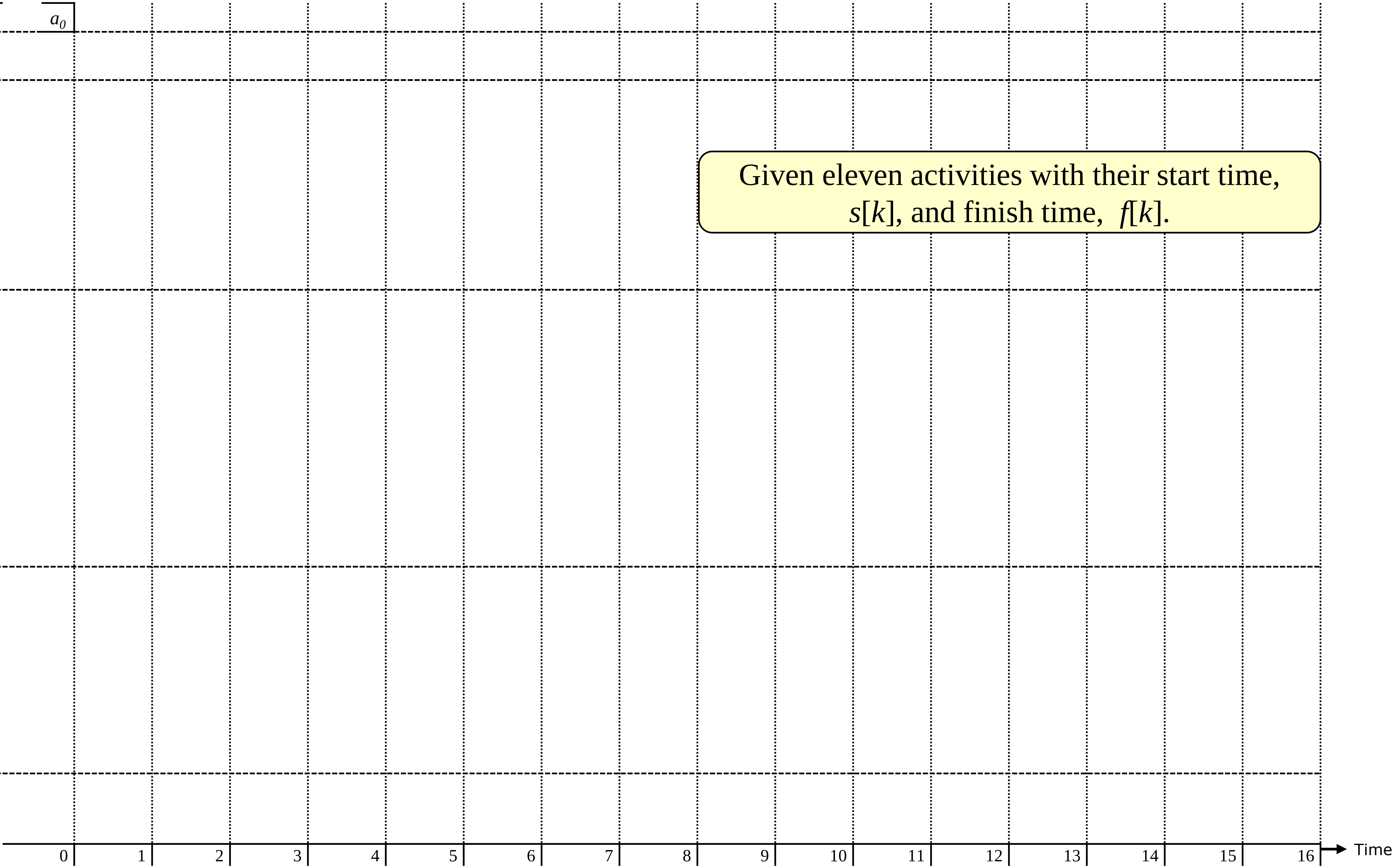
Hence, over all recursive calls the number of times lines 2-3 are executed is bounded by $\theta(n)$.

Time complexity Analysis of the Recursive Greedy Algorithm

Assuming that the activities are sorted in the order of increasing finish time, the running time of the call RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, n) is $\boldsymbol{\theta(n)}$.

Demonstration of the Recursive Greedy Algorithm to solve the Activity-selection problem

k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



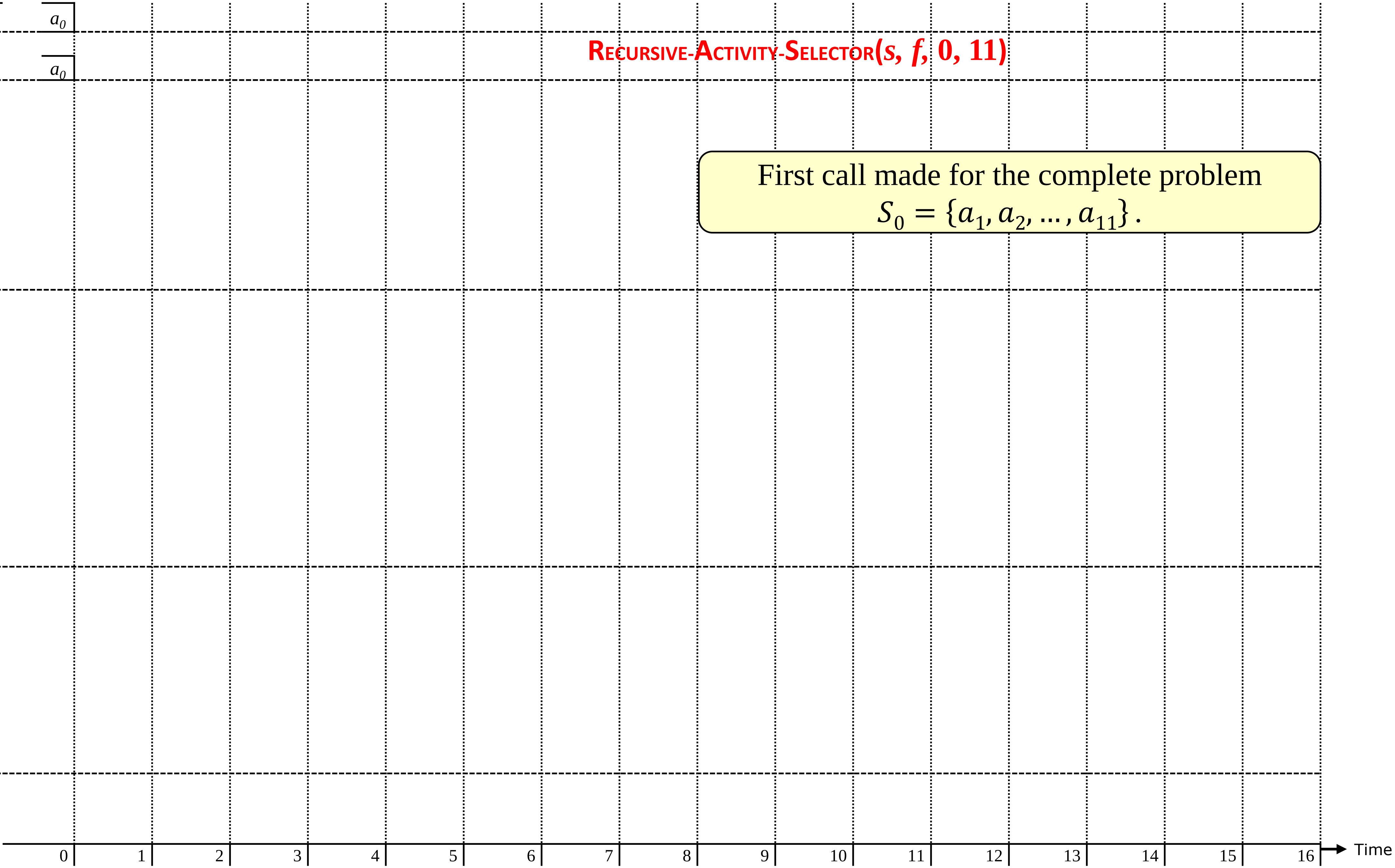
Given eleven activities with their start time, $s[k]$, and finish time, $f[k]$.

k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16

RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, 11)

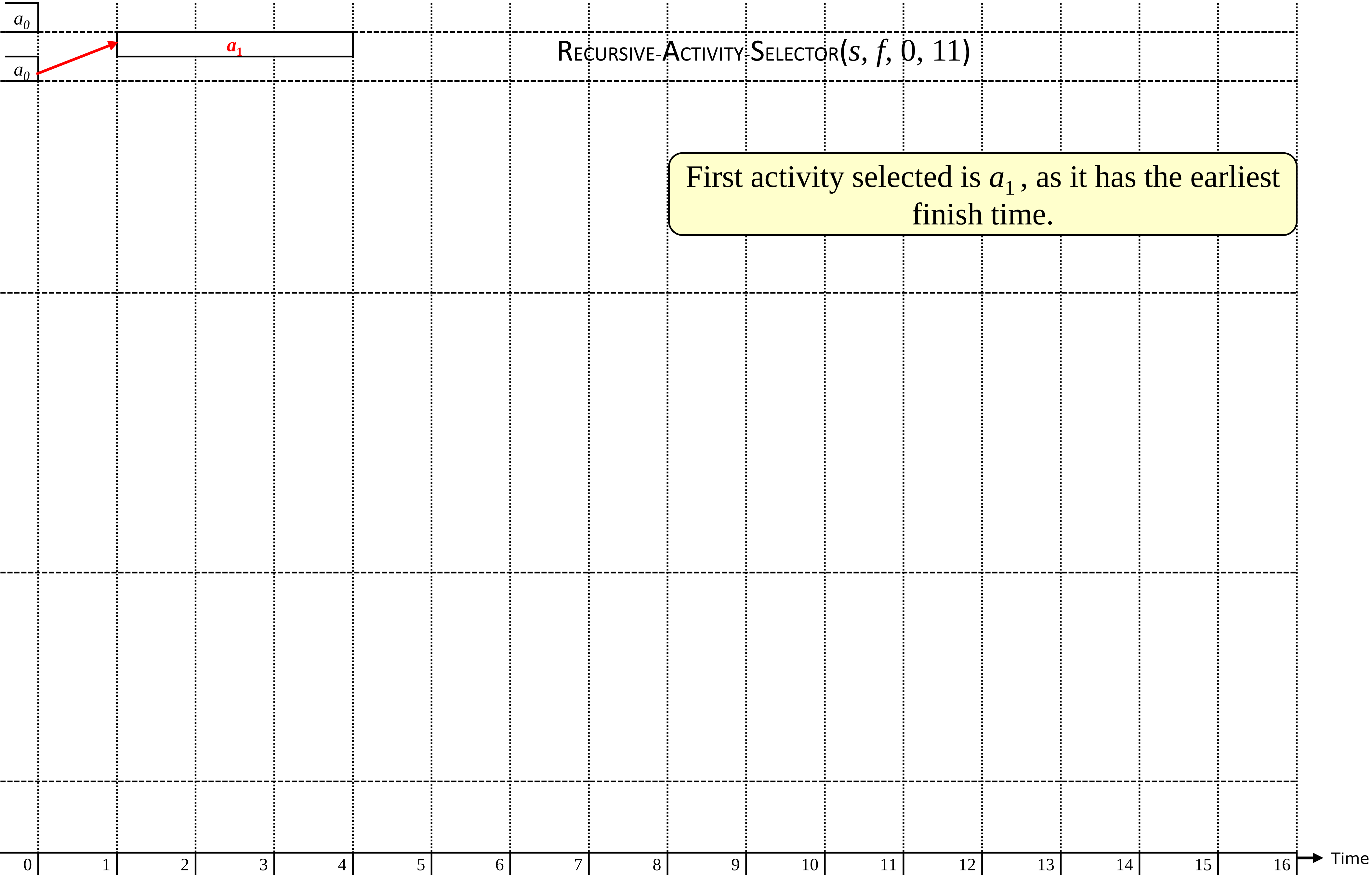
First call made for the complete problem

$S_0 = \{a_1, a_2, \dots, a_{11}\}$.



k	$s[k]$	$f[k]$
0	-	0
1	**1**	**4**
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16

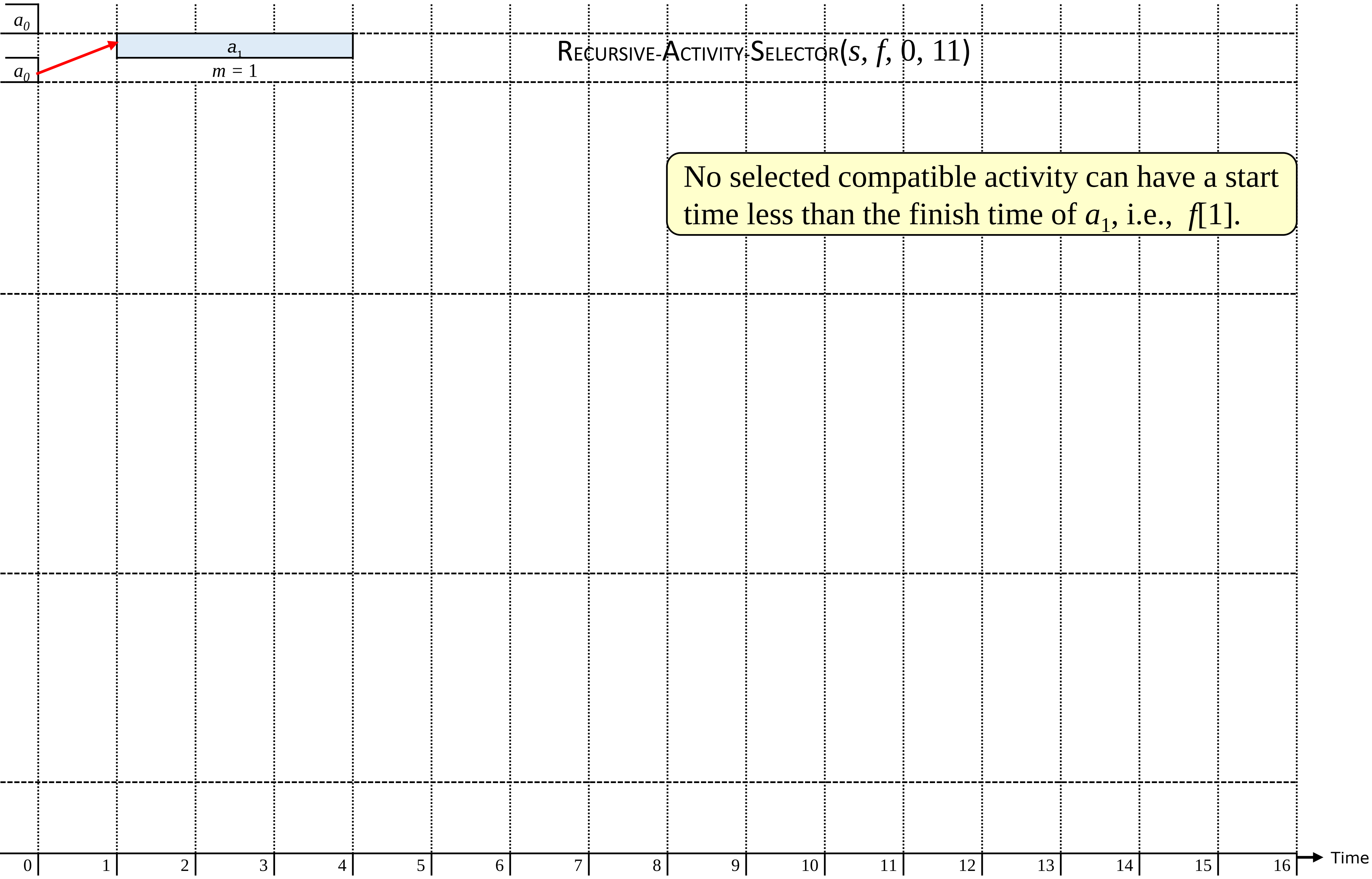
RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, 11)$



First activity selected is a_1, as it has the earliest finish time.

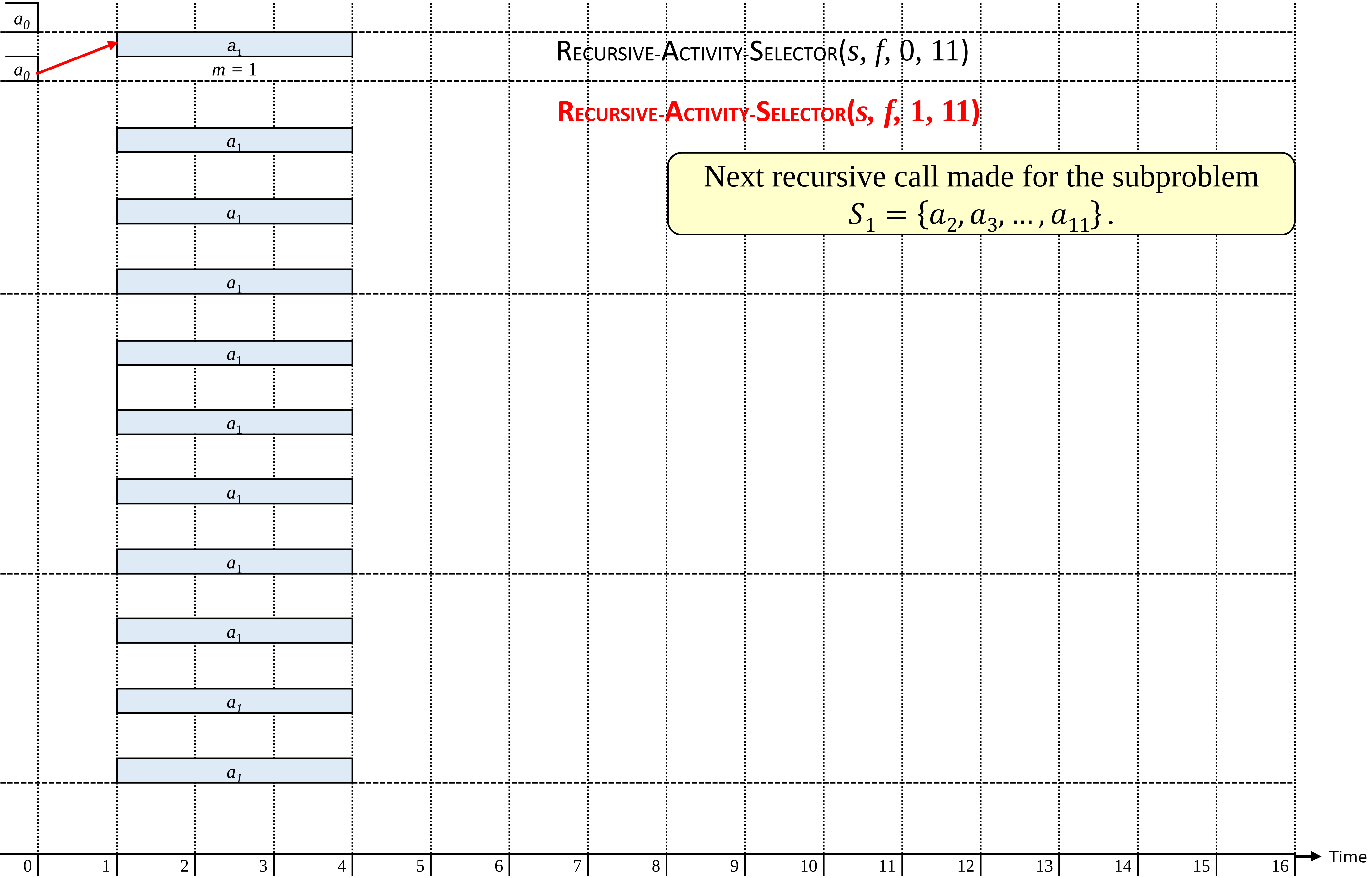
RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, 11)

k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16

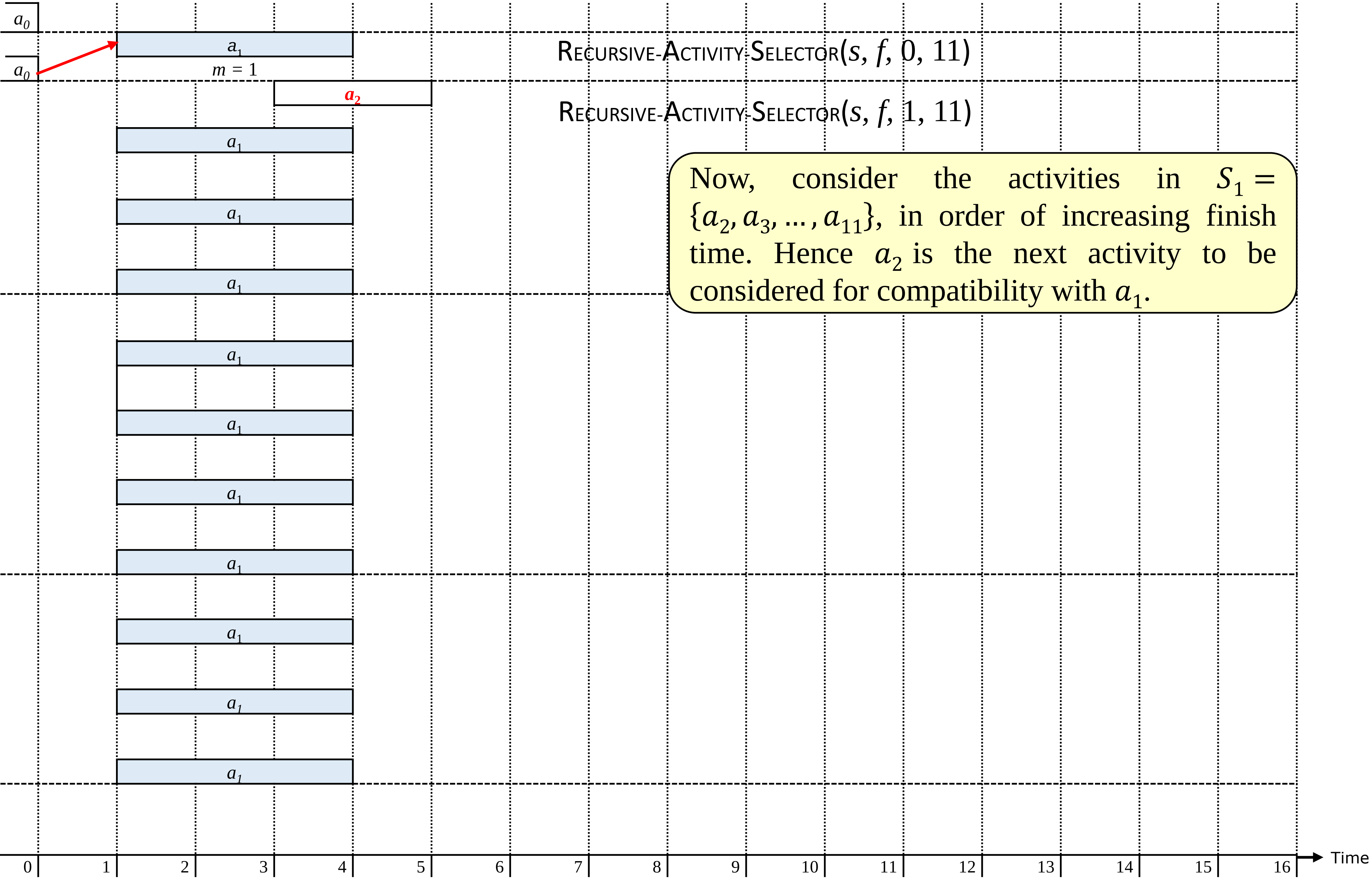


No selected compatible activity can have a start time less than the finish time of a_1, i.e., $f[1]$.

k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	**3**	**5**
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16

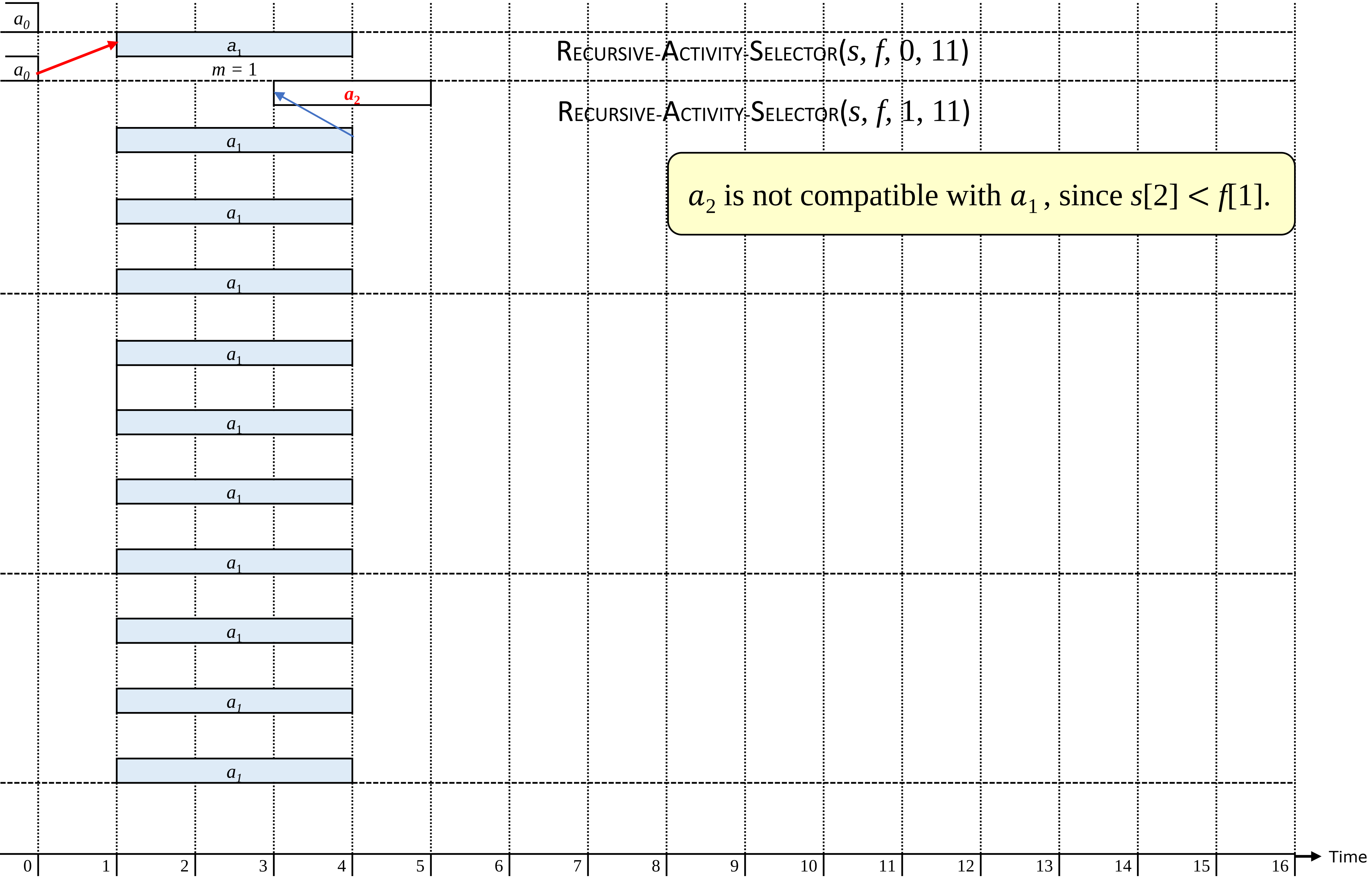


RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, 11)$

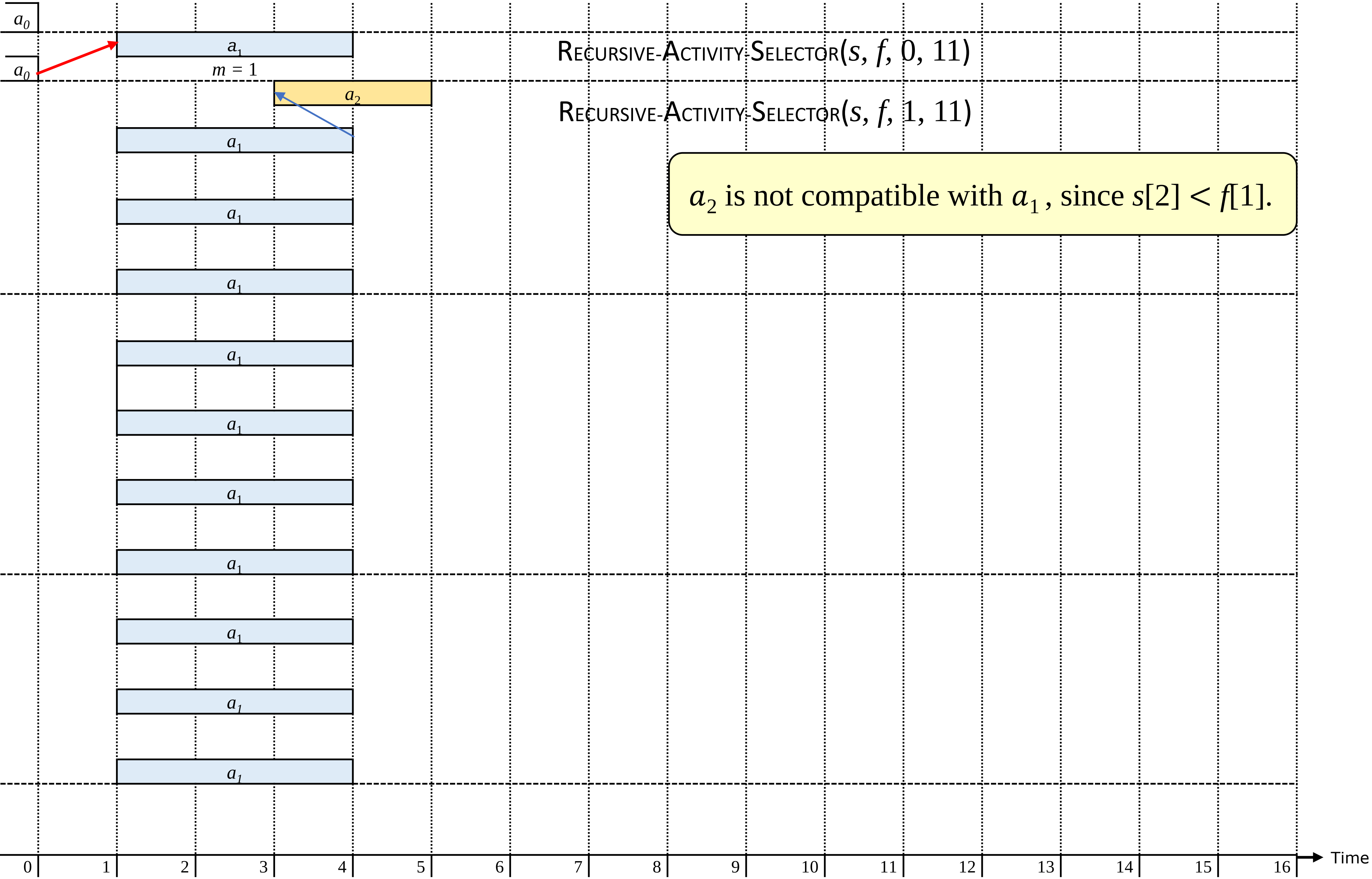
RECURSIVE-ACTIVITY-SELECTOR$(s, f, 1, 11)$

Now, consider the activities in $S_1 = \{a_2, a_3, \dots, a_{11}\}$, in order of increasing finish time. Hence a_2 is the next activity to be considered for compatibility with a_1.

k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	**3**	**5**
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16

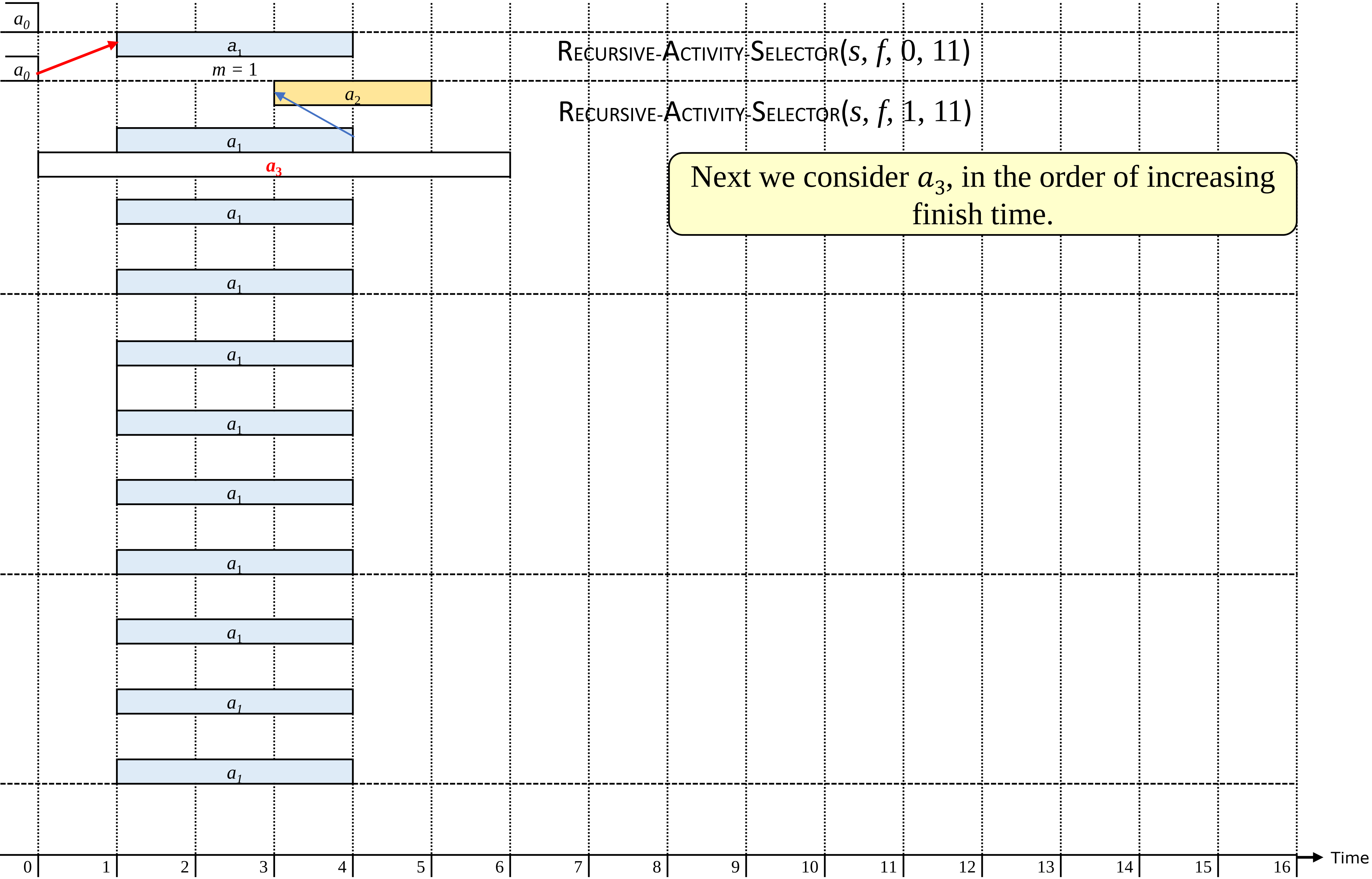


k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



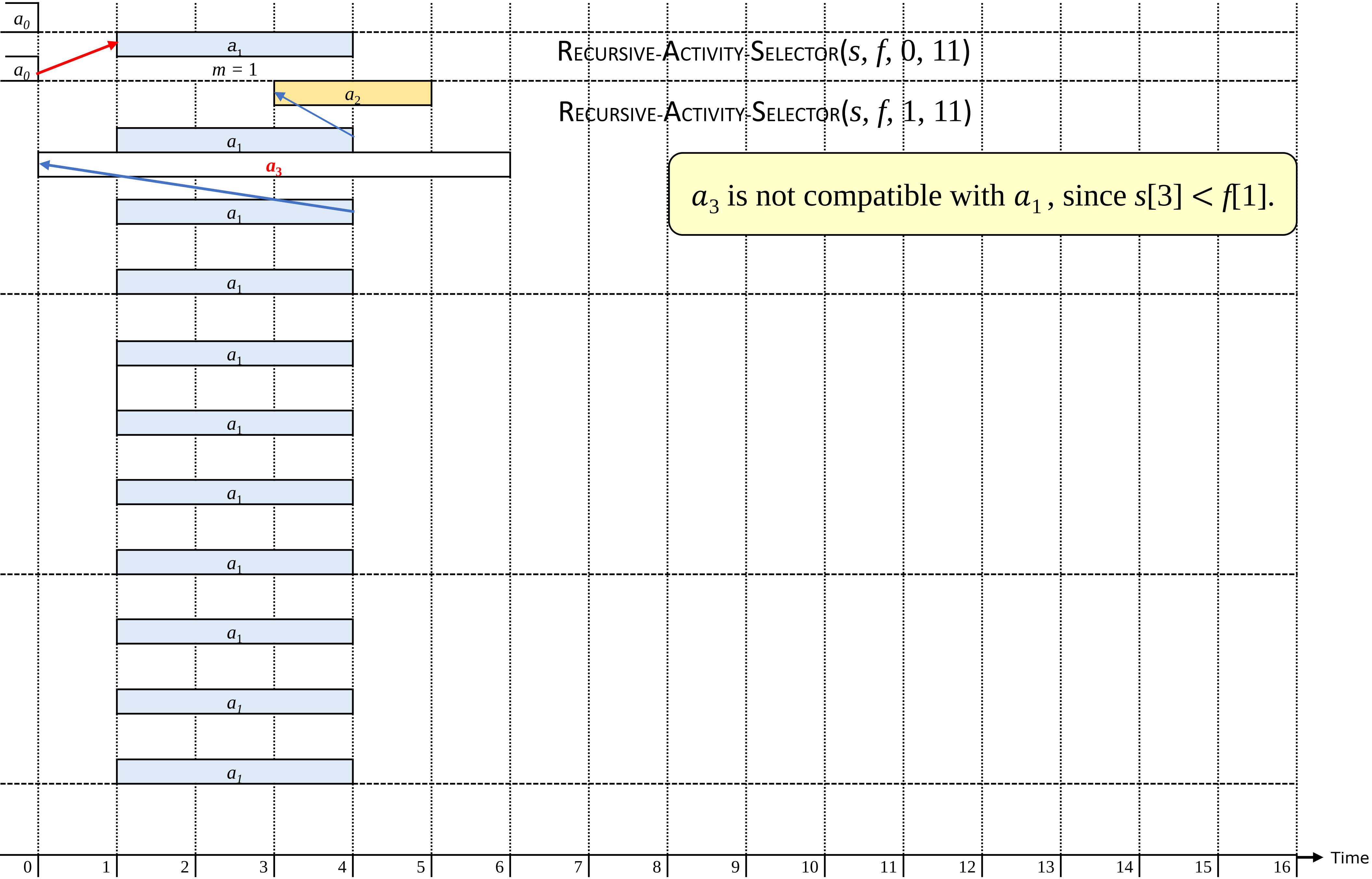
a_2 is not compatible with a_1, since $s[2] < f[1]$.

k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	**0**	**6**
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



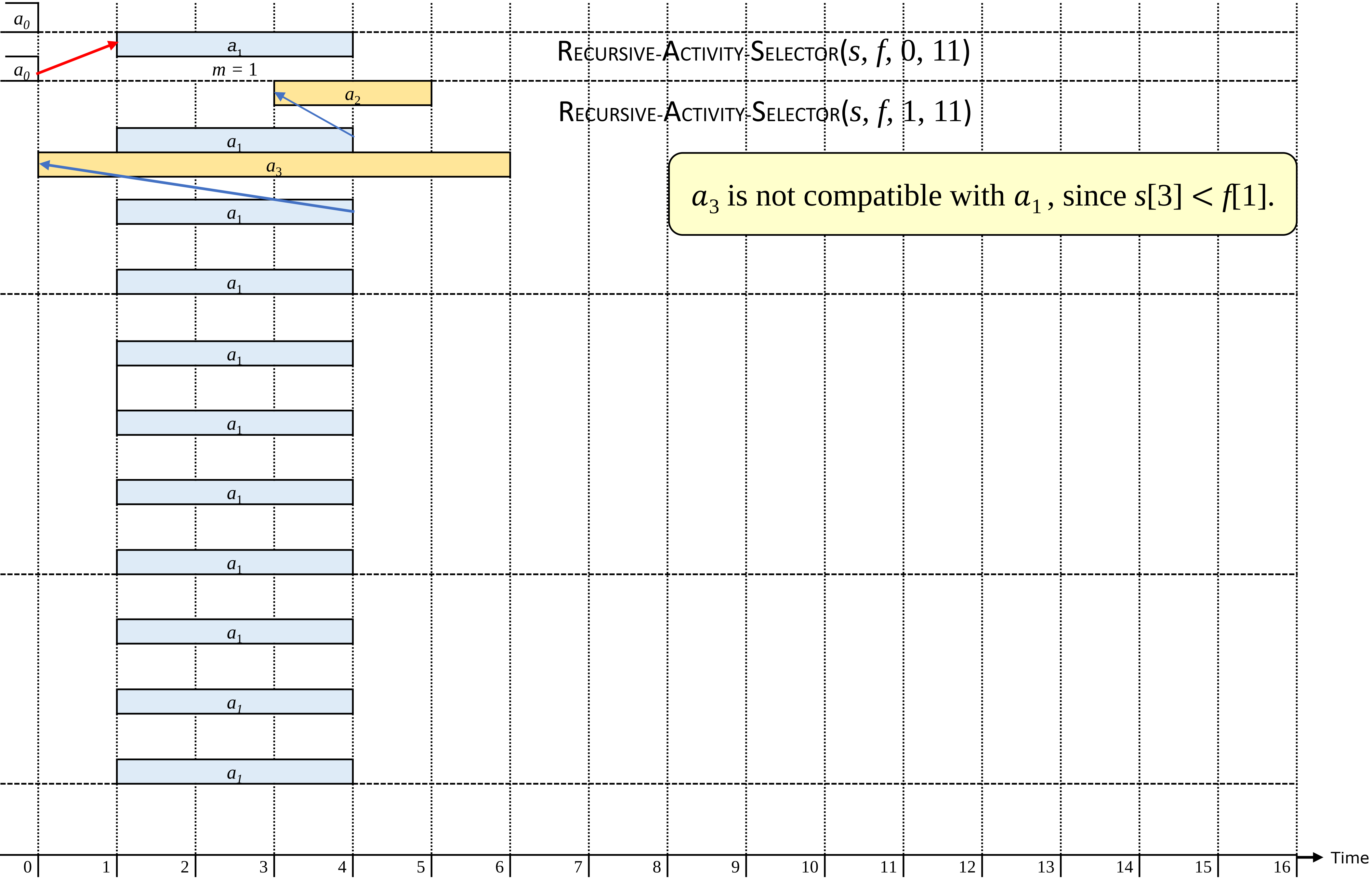
Next we consider a_3, in the order of increasing finish time.

k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	**0**	**6**
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16

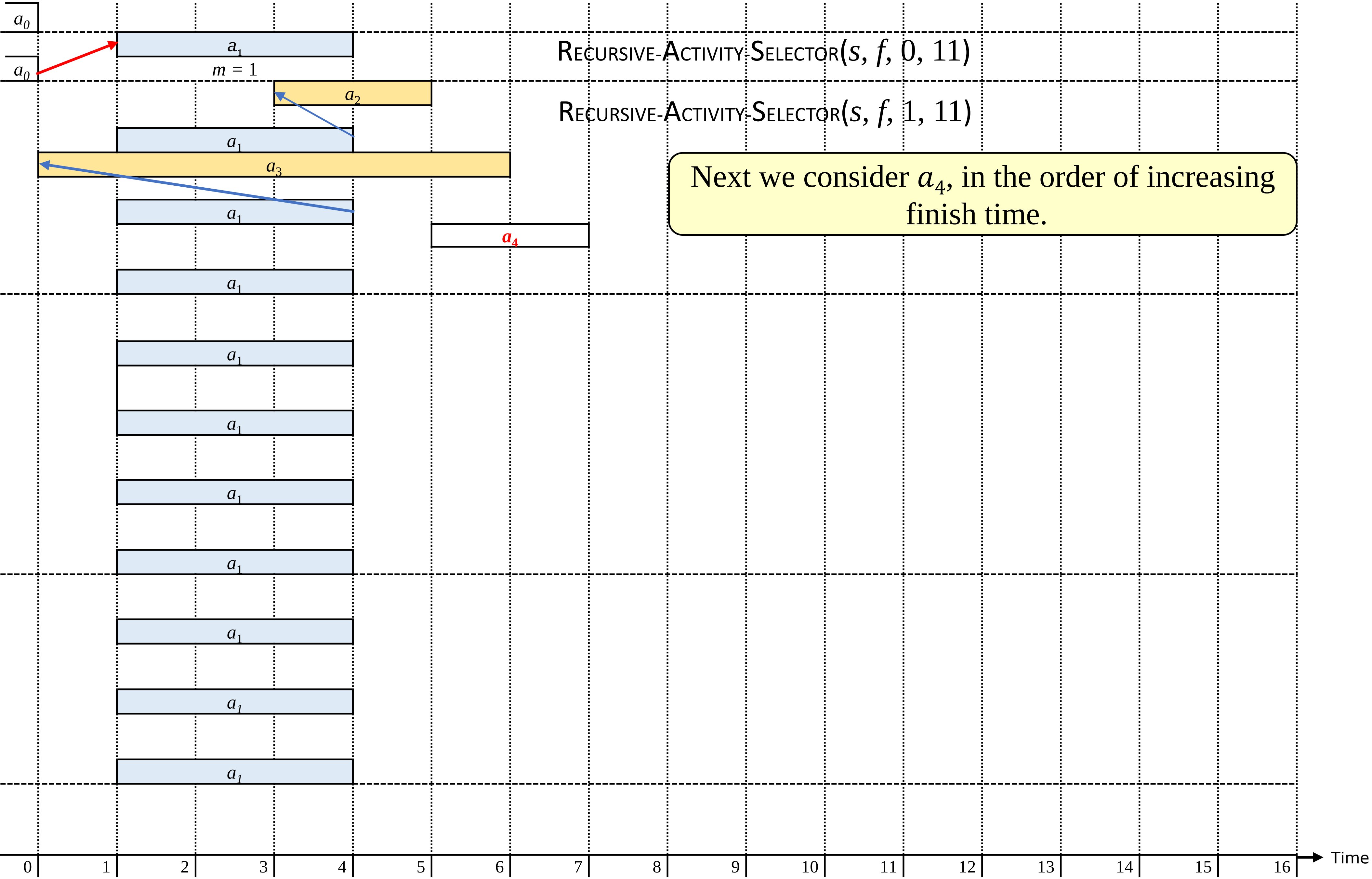


a_3 is not compatible with a_1, since $s[3] < f[1]$.

k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16

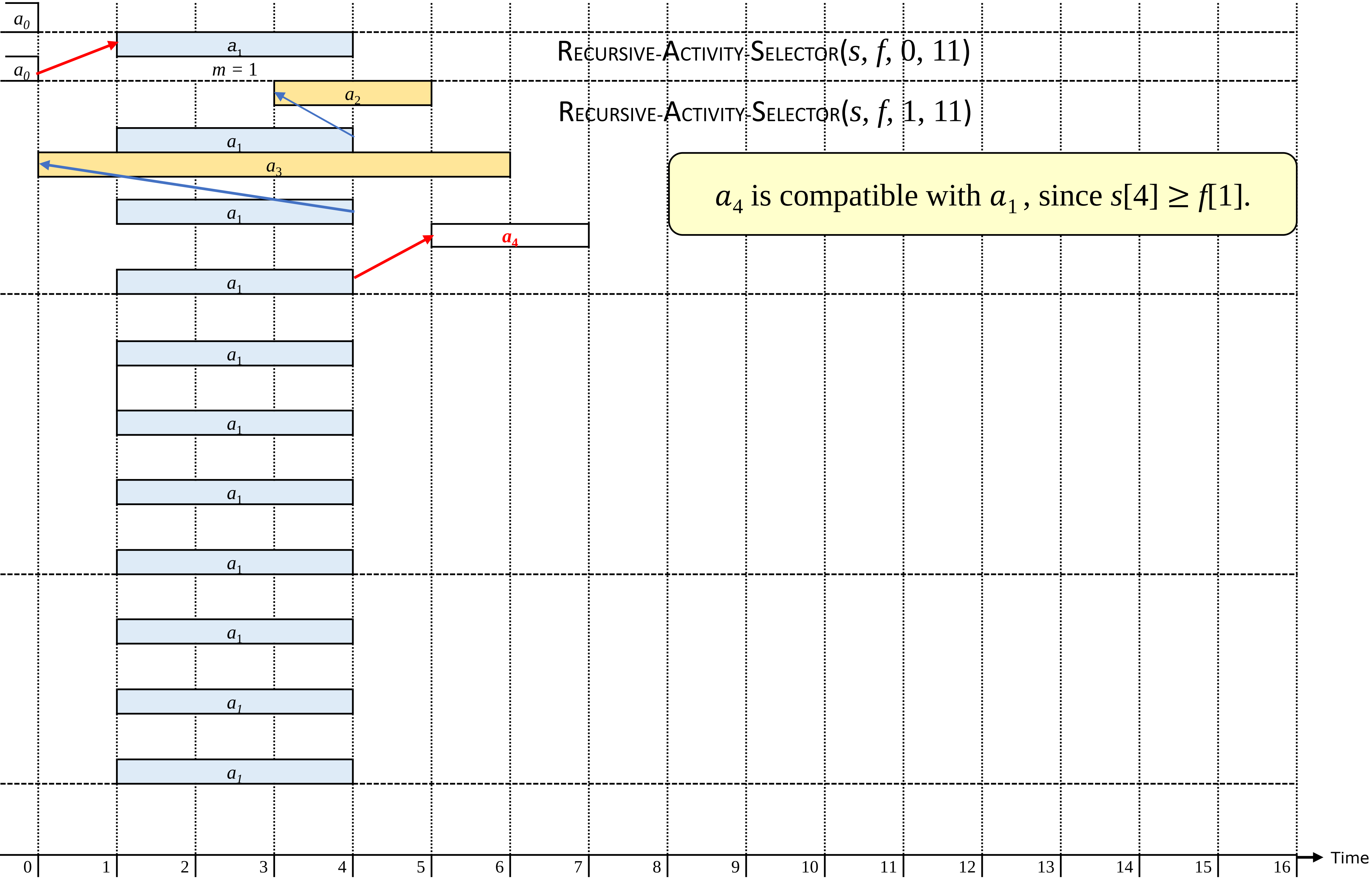


k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	**5**	**7**
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16

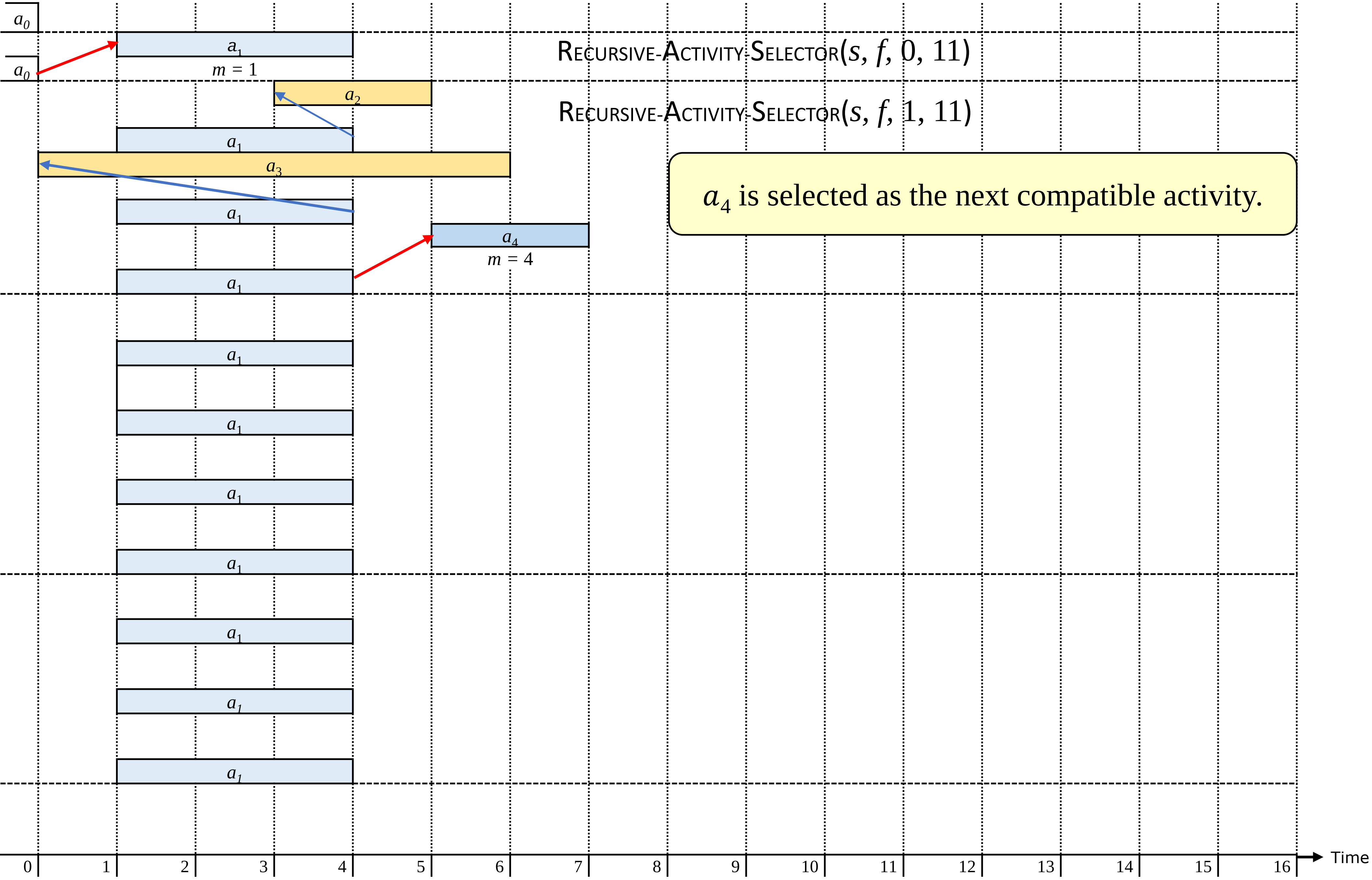


Next we consider a_4, in the order of increasing finish time.

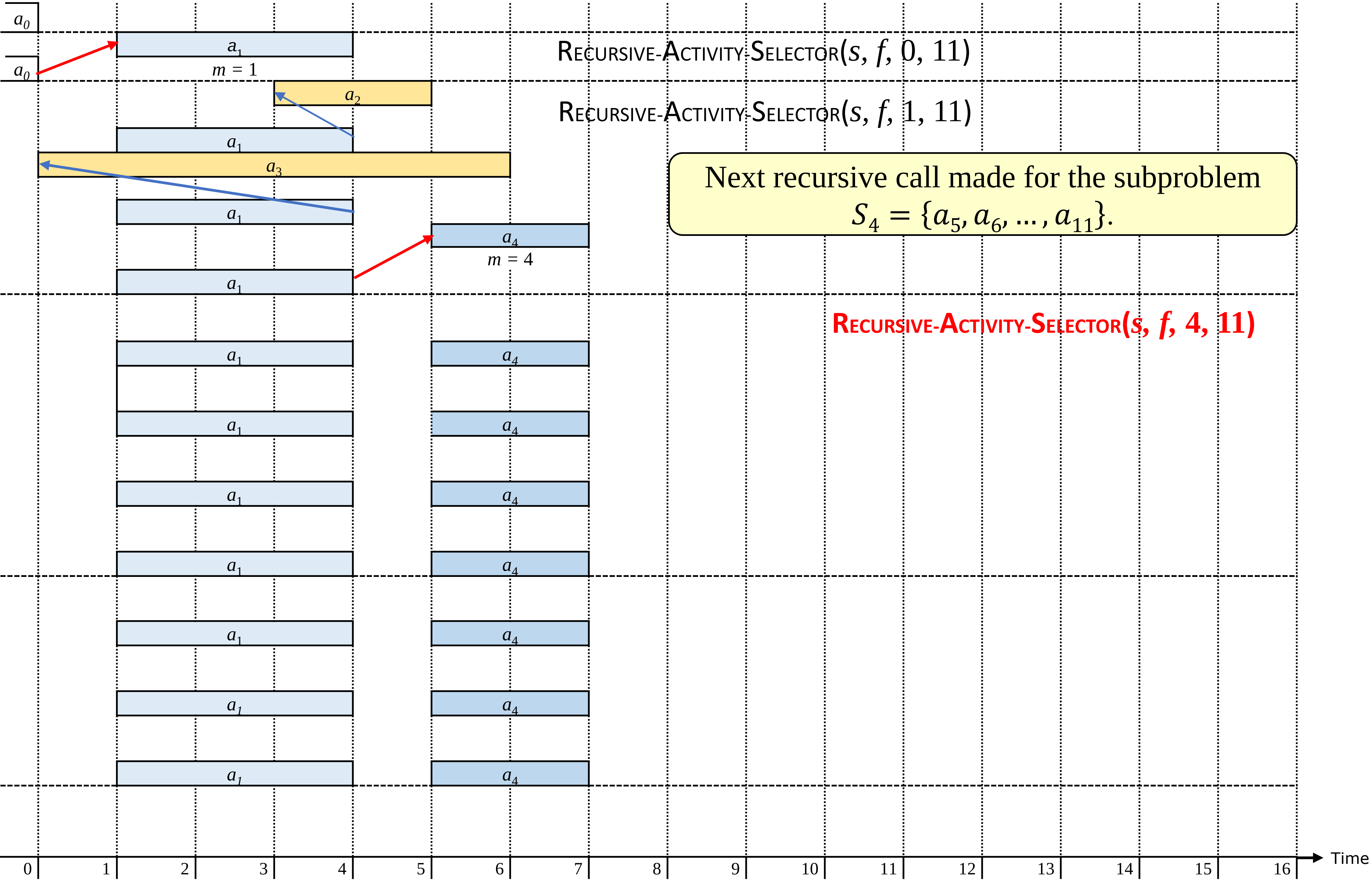
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	**5**	**7**
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



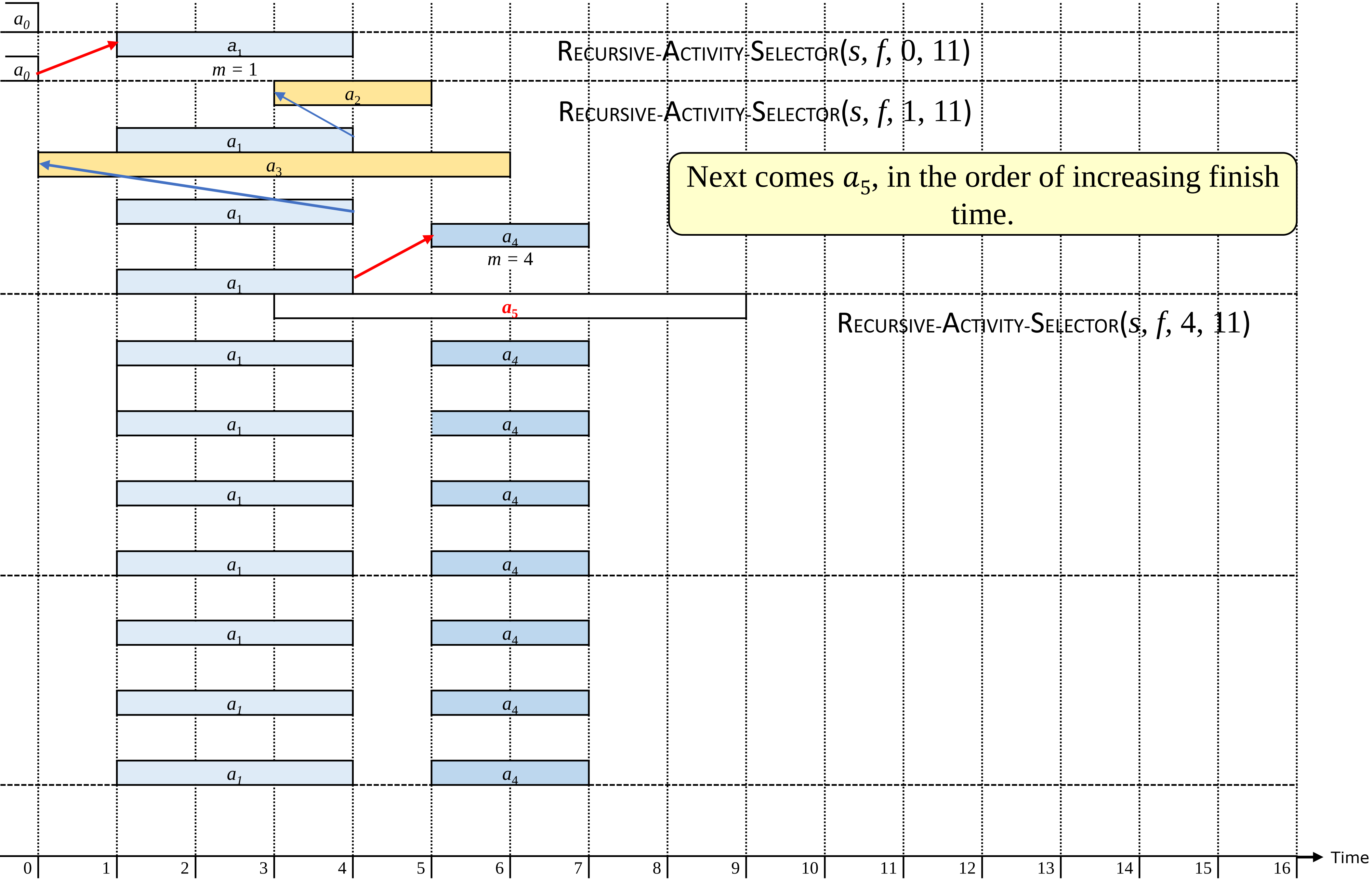
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



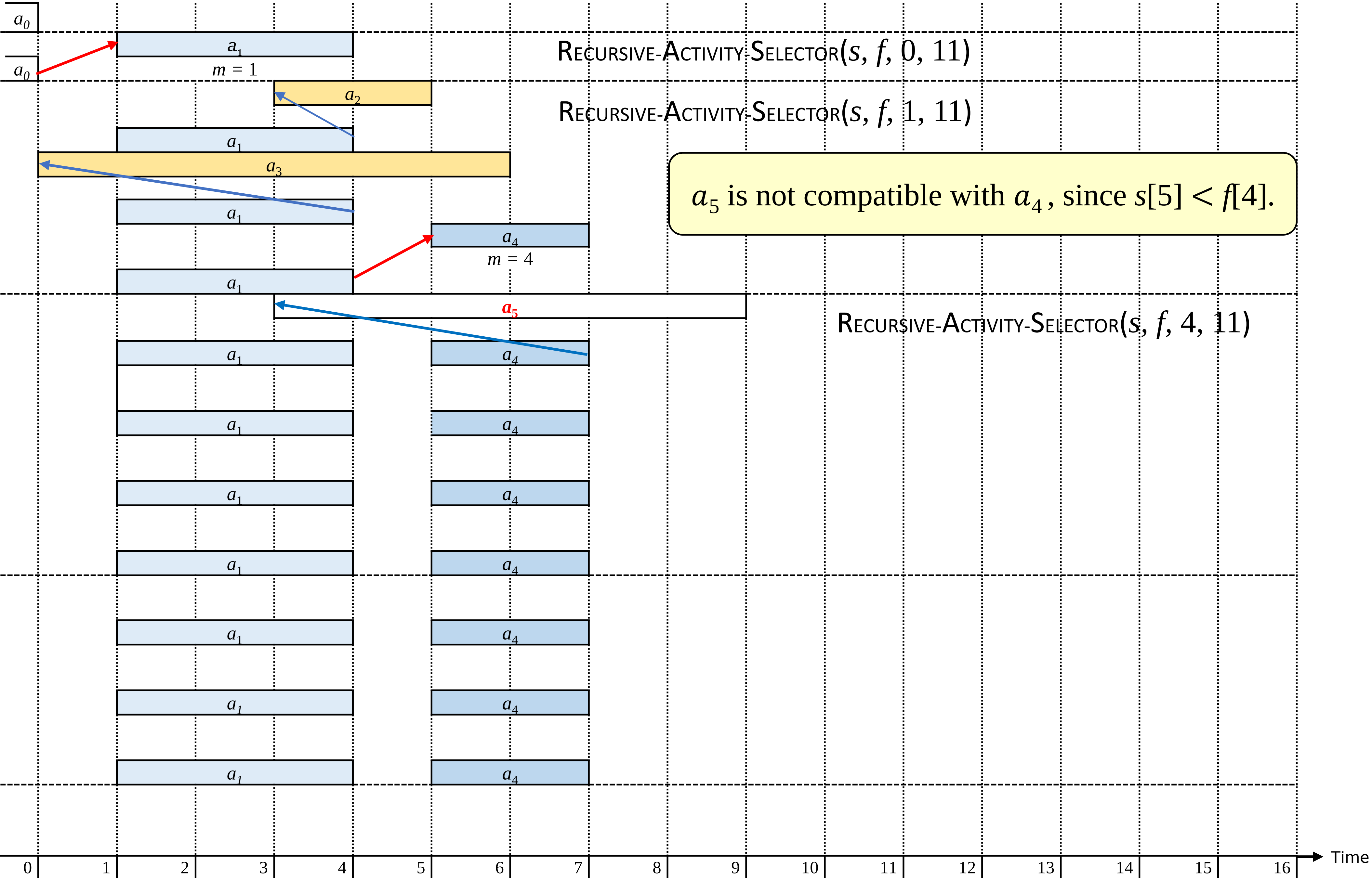
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



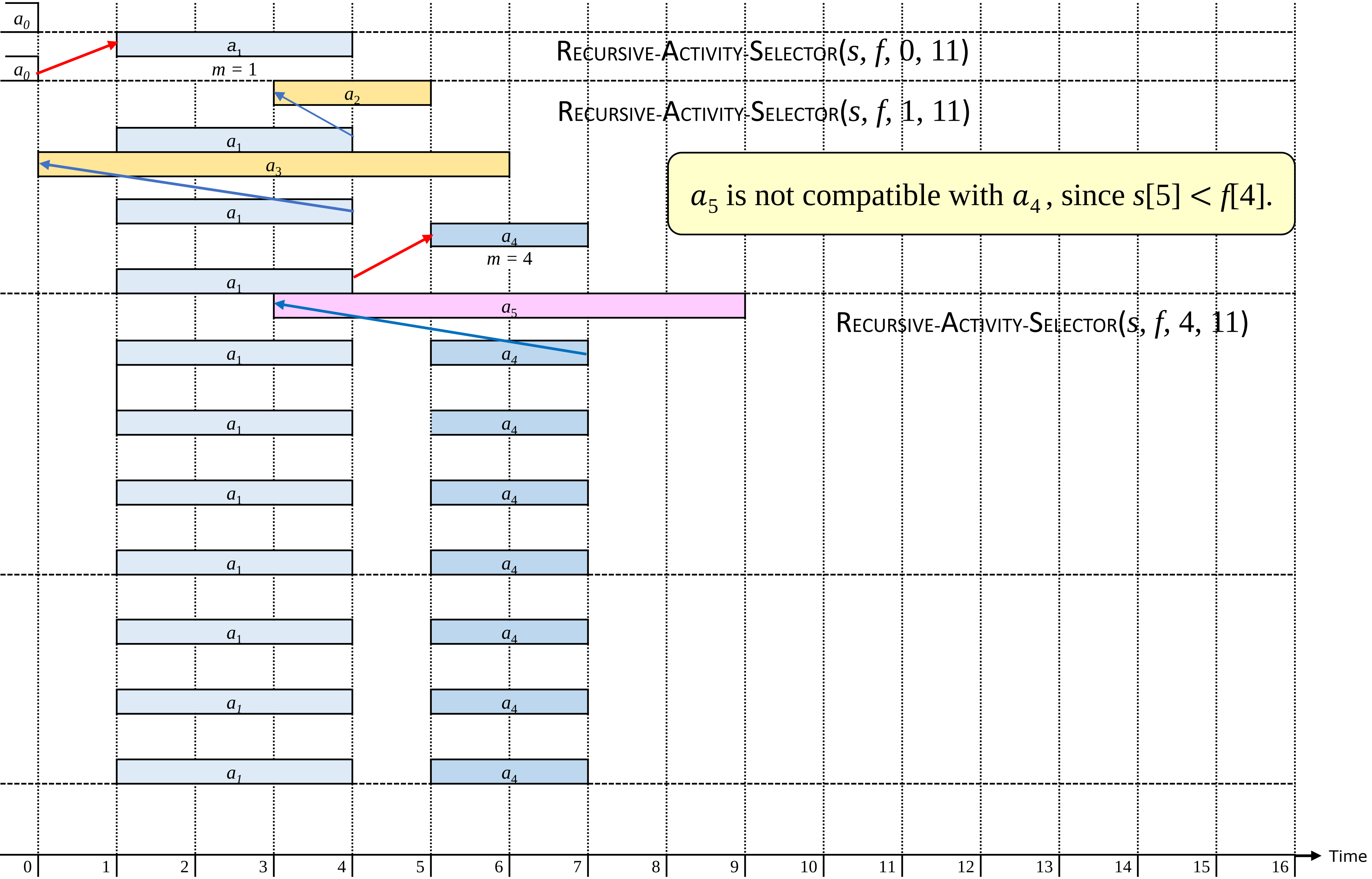
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	**3**	**9**
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



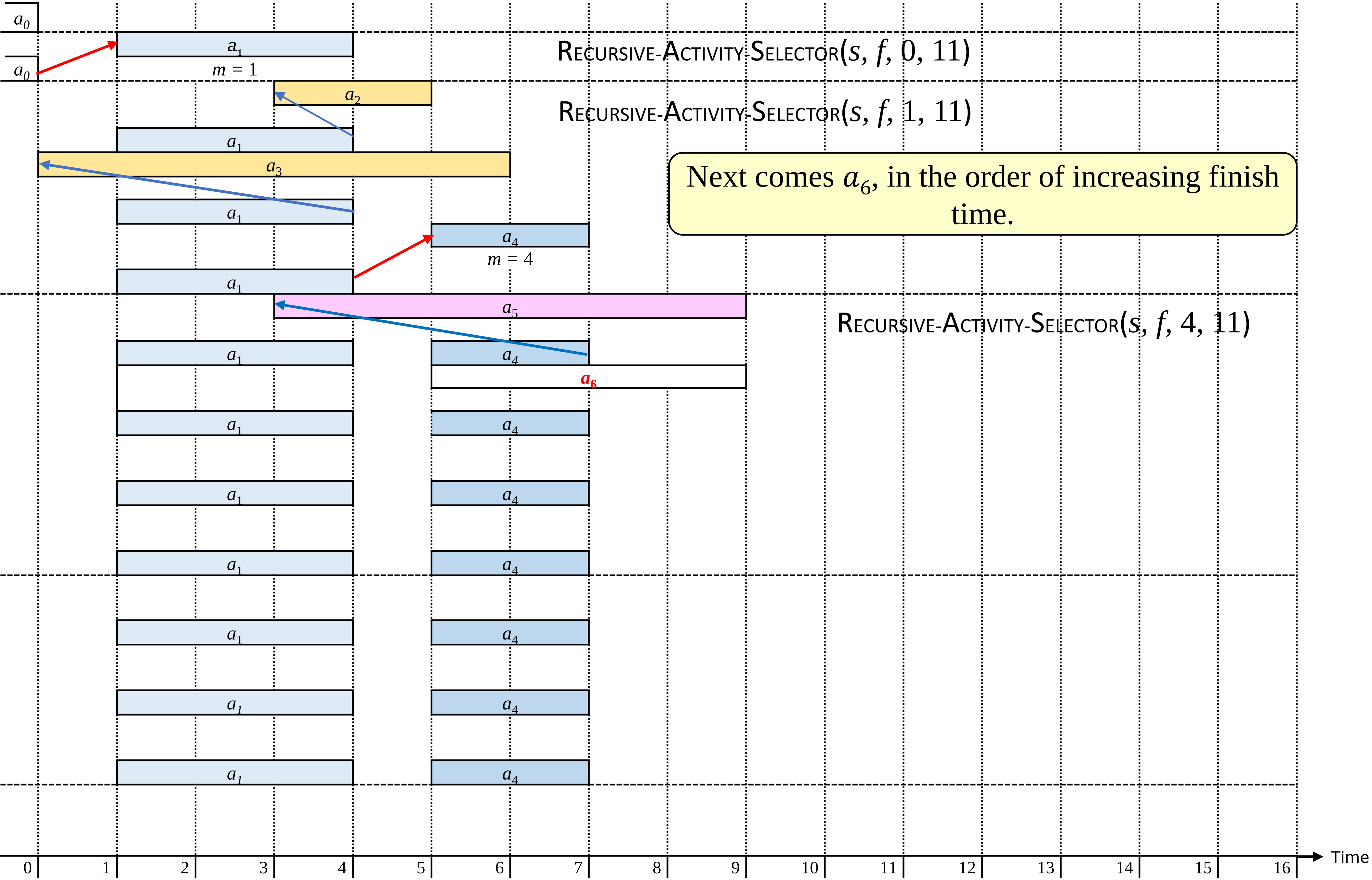
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	**3**	**9**
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



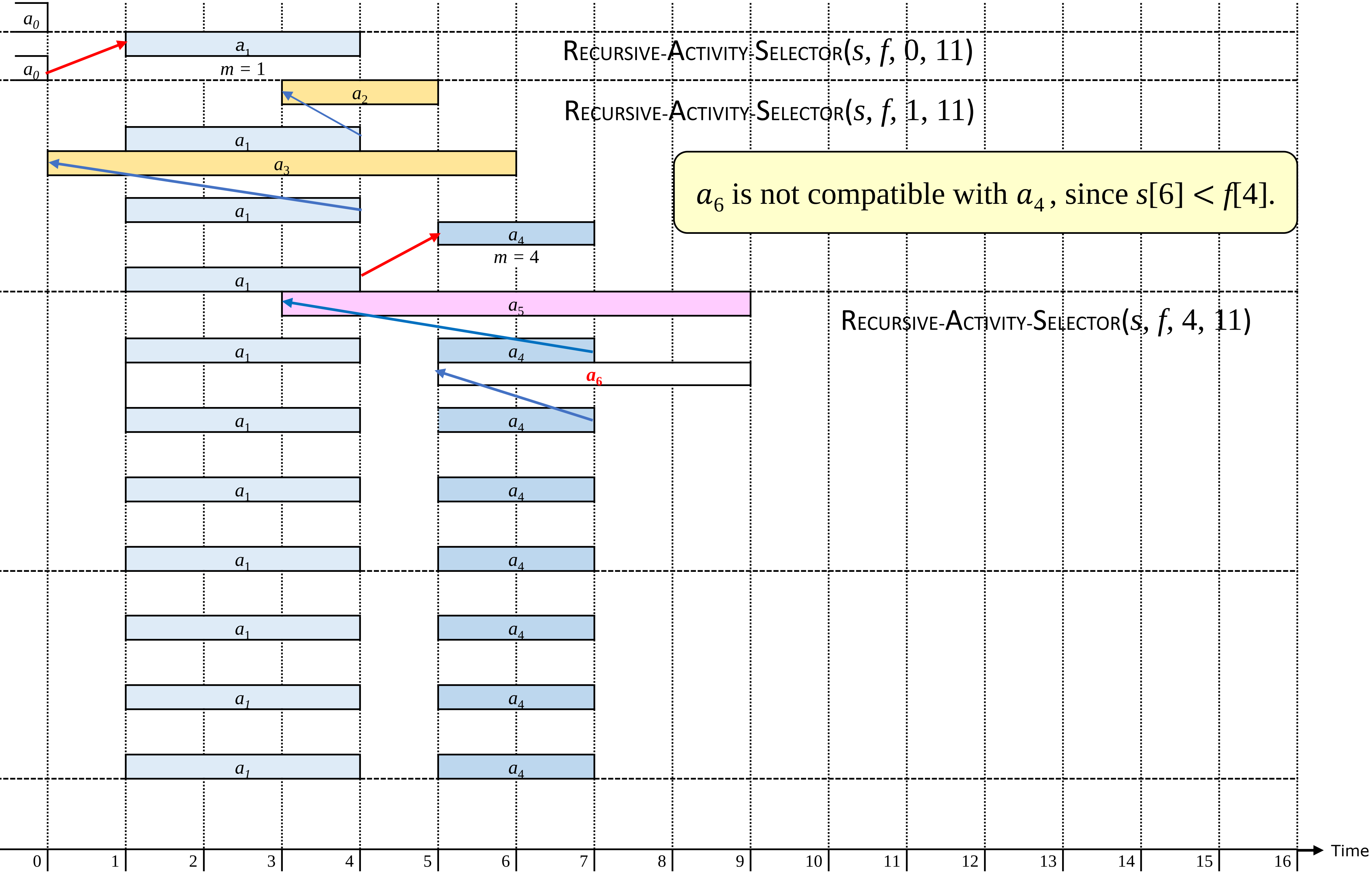
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



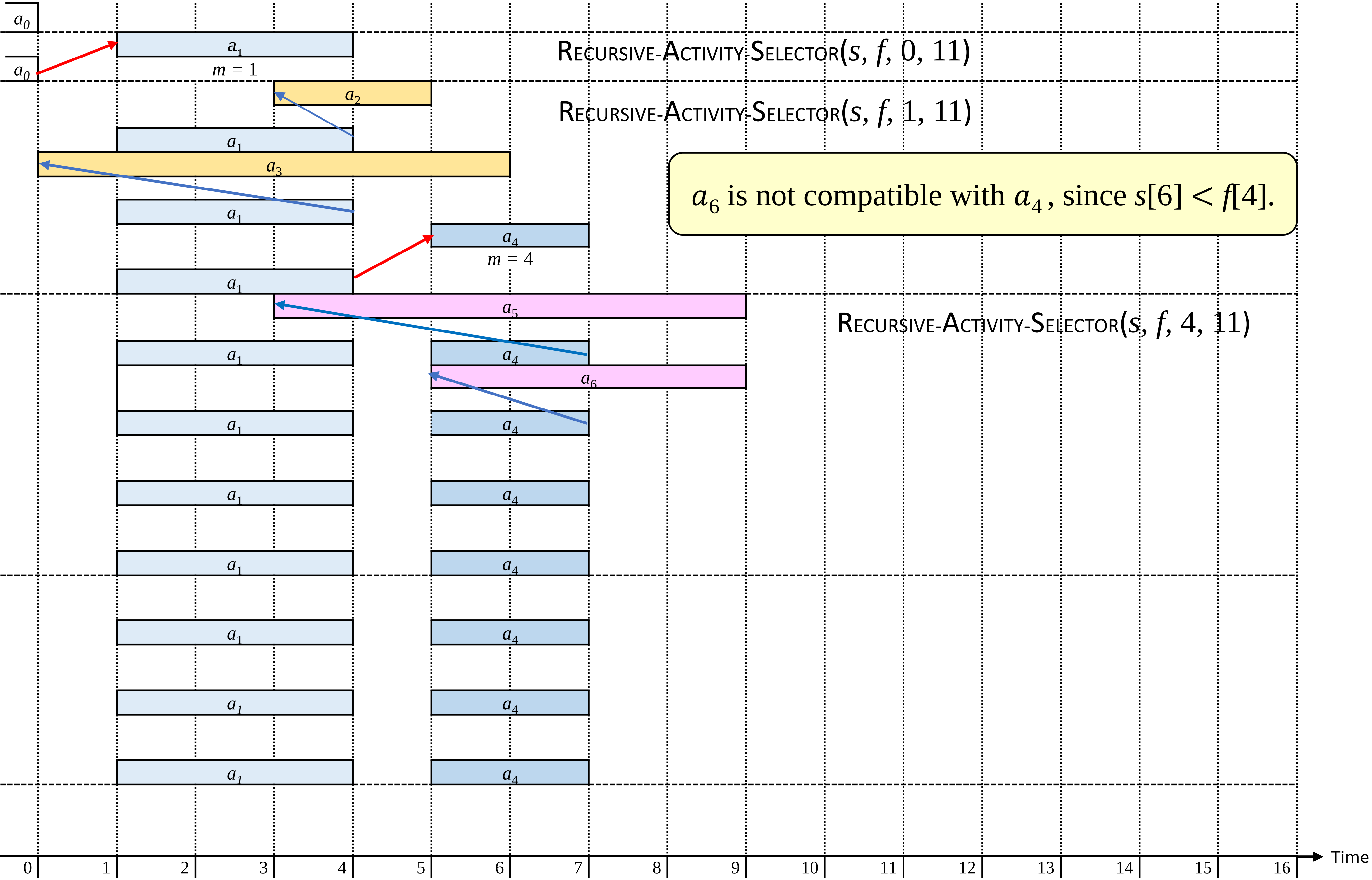
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	**5**	**9**
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



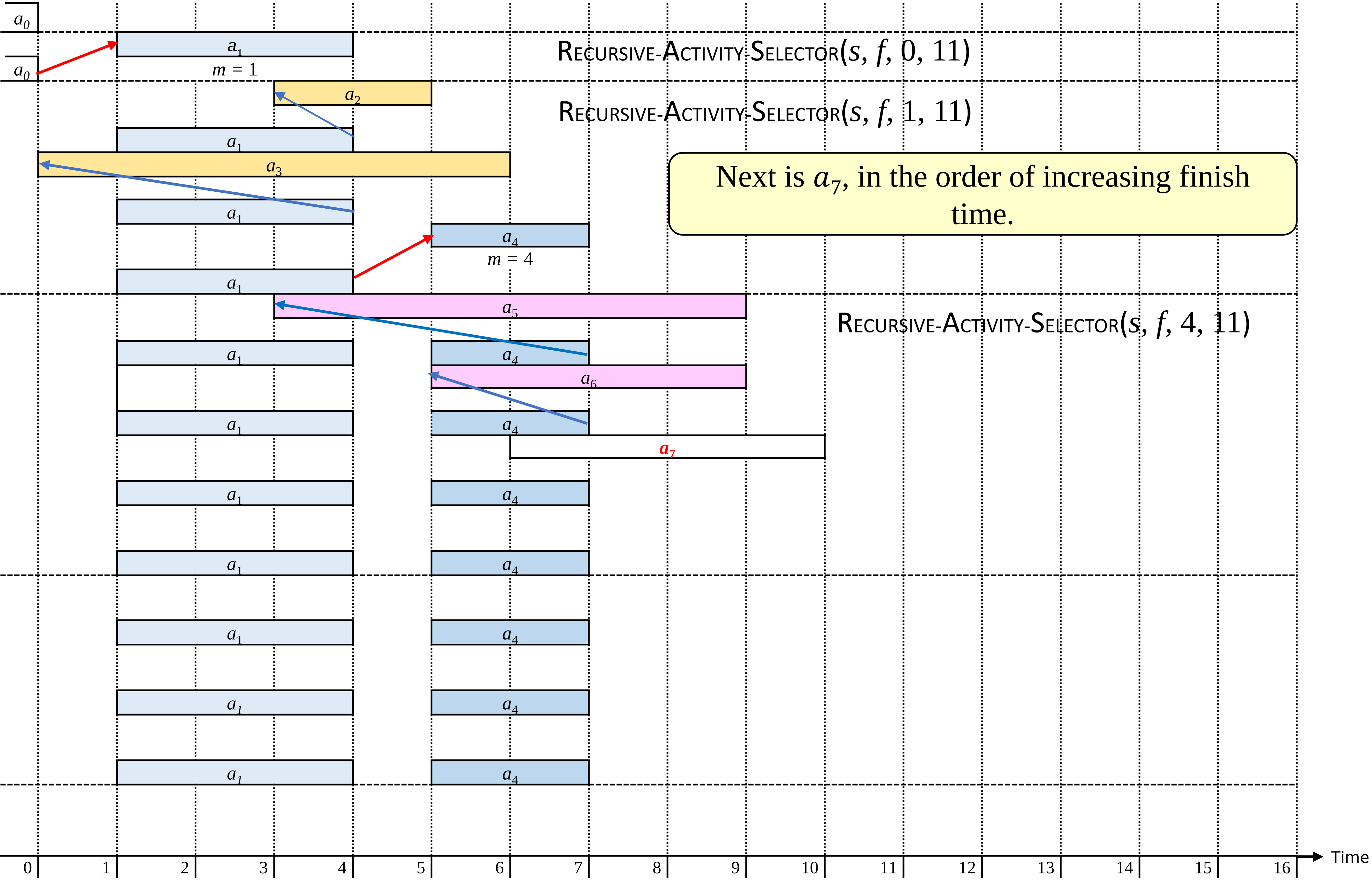
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	**5**	**9**
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



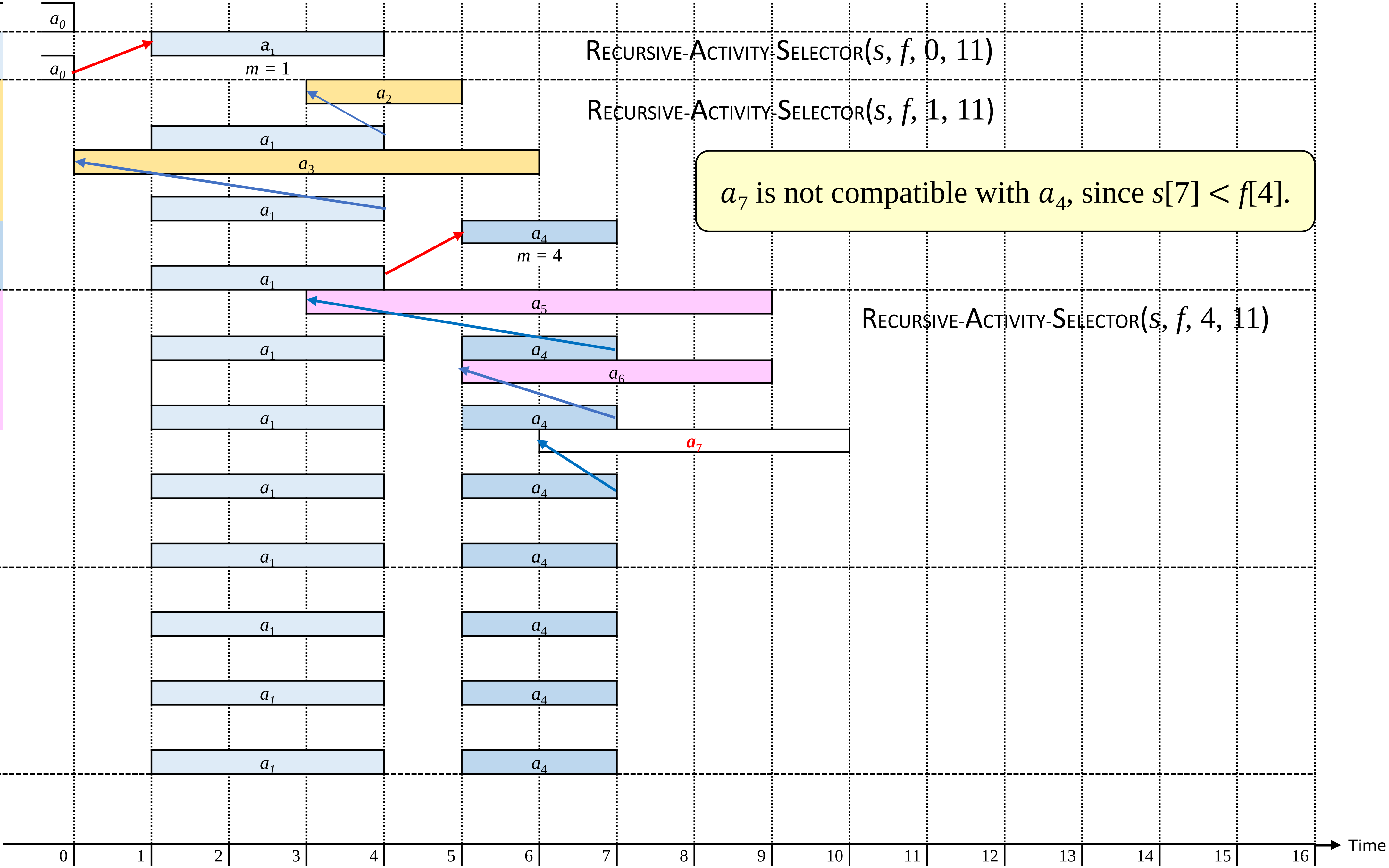
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



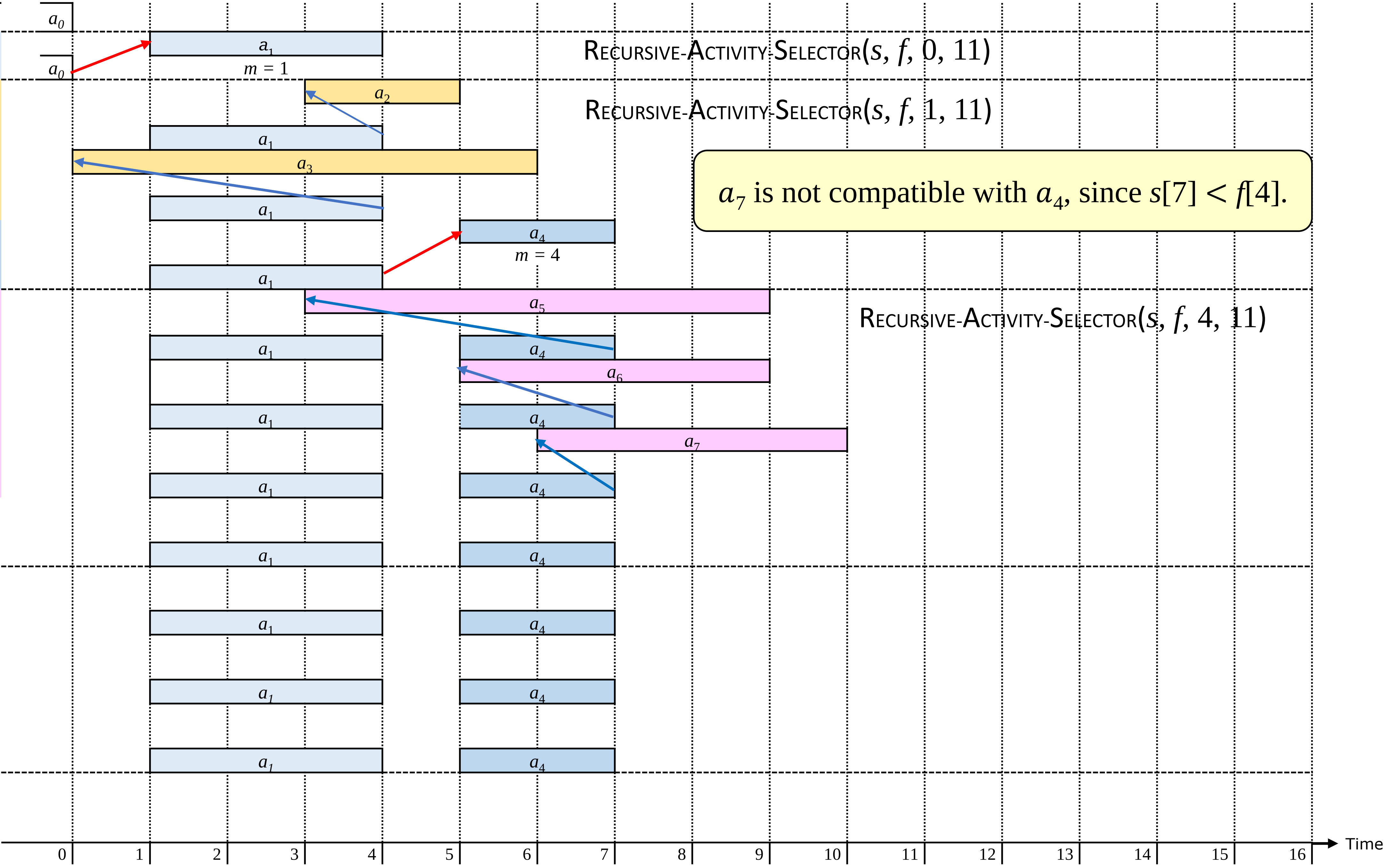
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	**6**	**10**
8	8	11
9	8	12
10	2	14
11	12	16



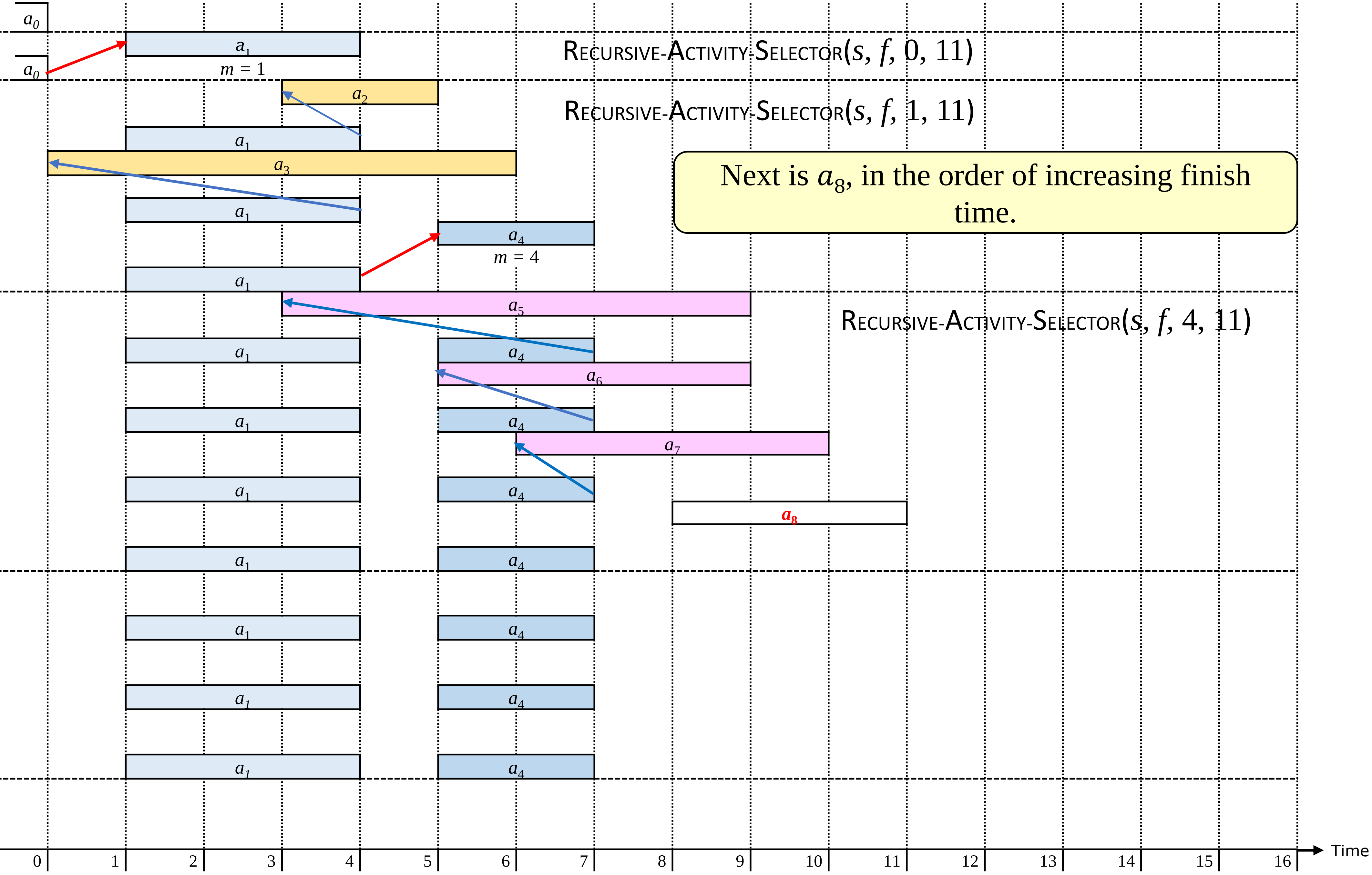
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	**6**	**10**
8	8	11
9	8	12
10	2	14
11	12	16



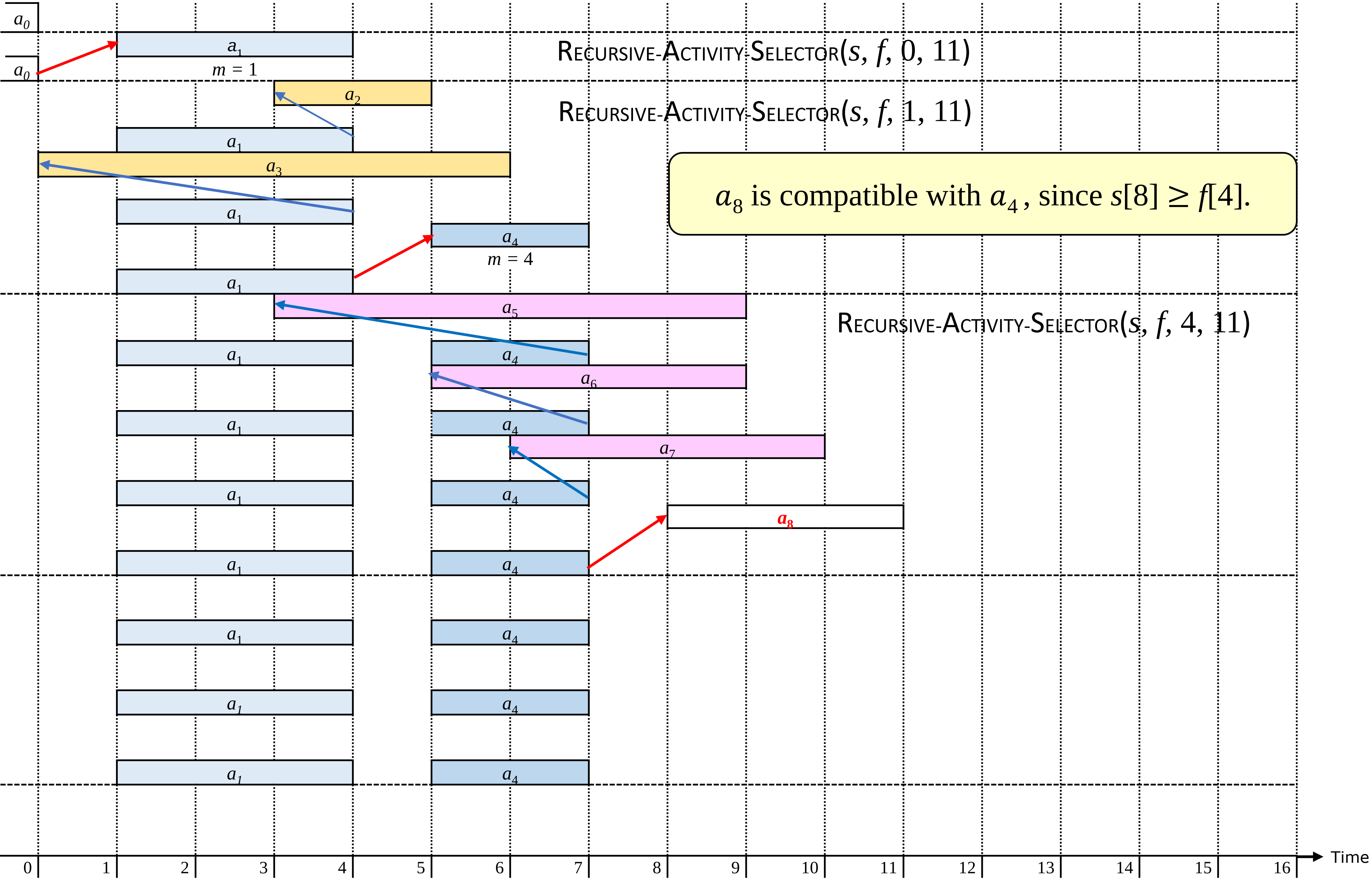
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	**8**	**11**
9	8	12
10	2	14
11	12	16

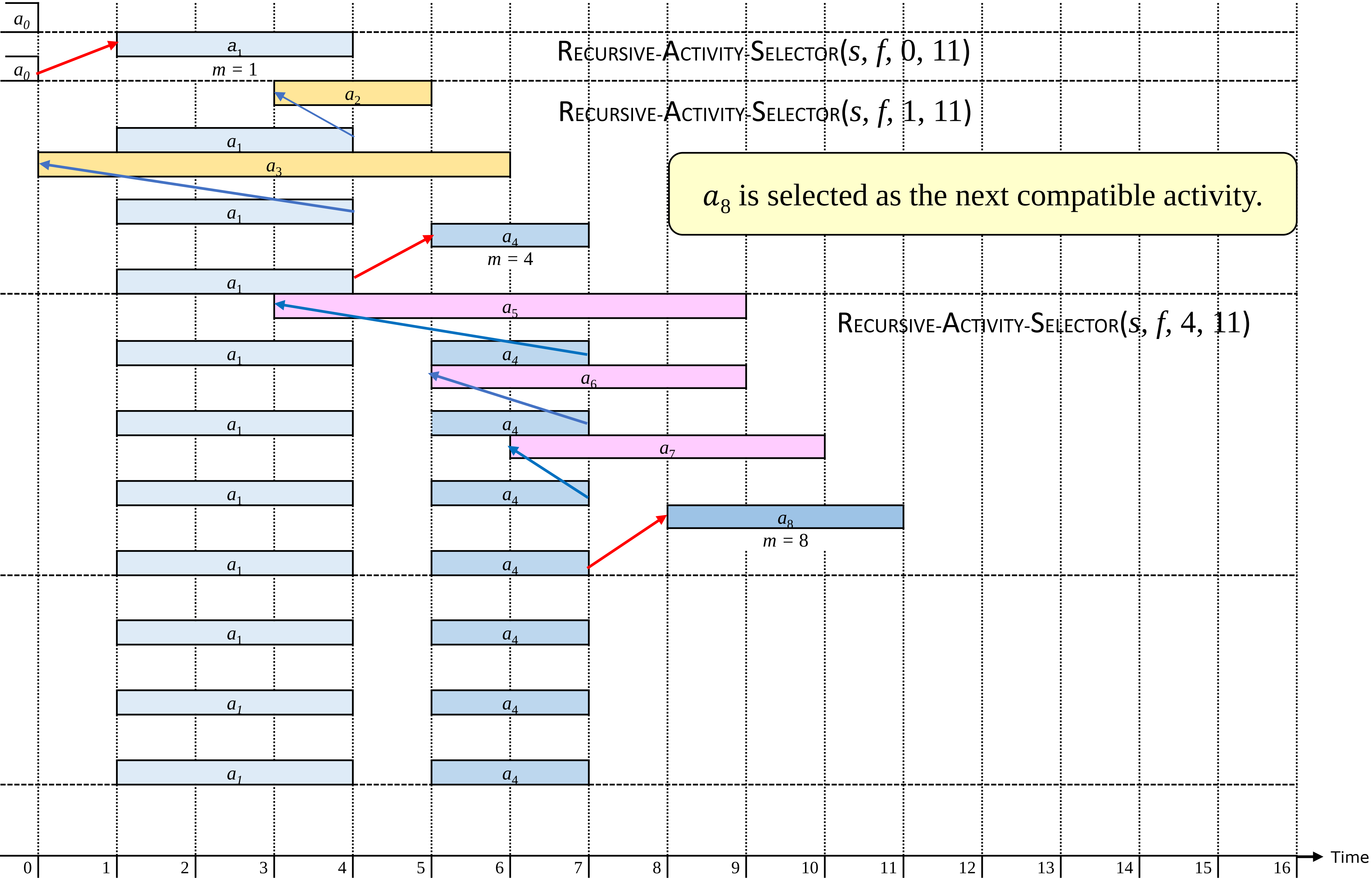


k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	**8**	**11**
9	8	12
10	2	14
11	12	16

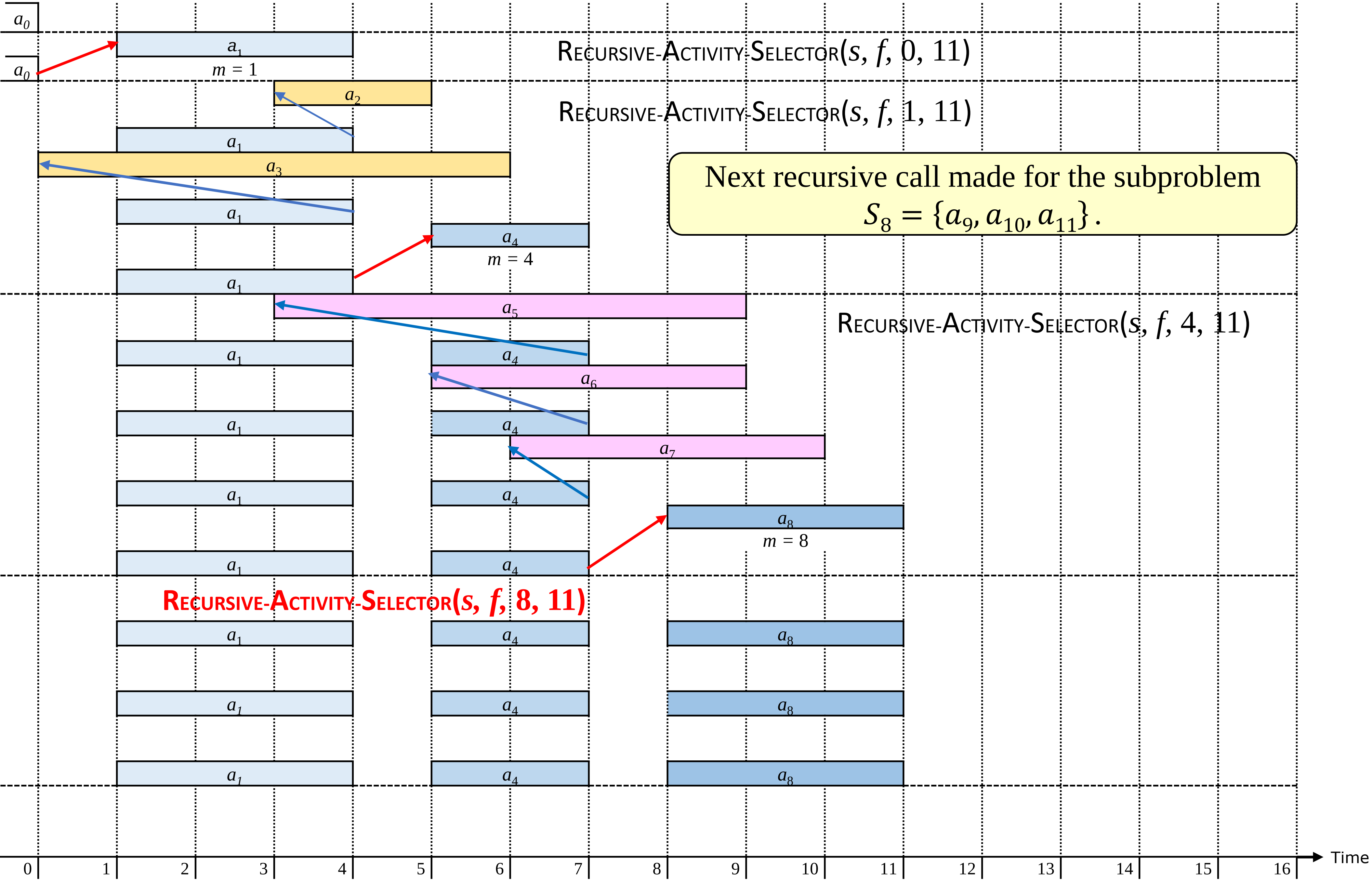


a_8 is compatible with a_4, since $s[8] \geq f[4]$.

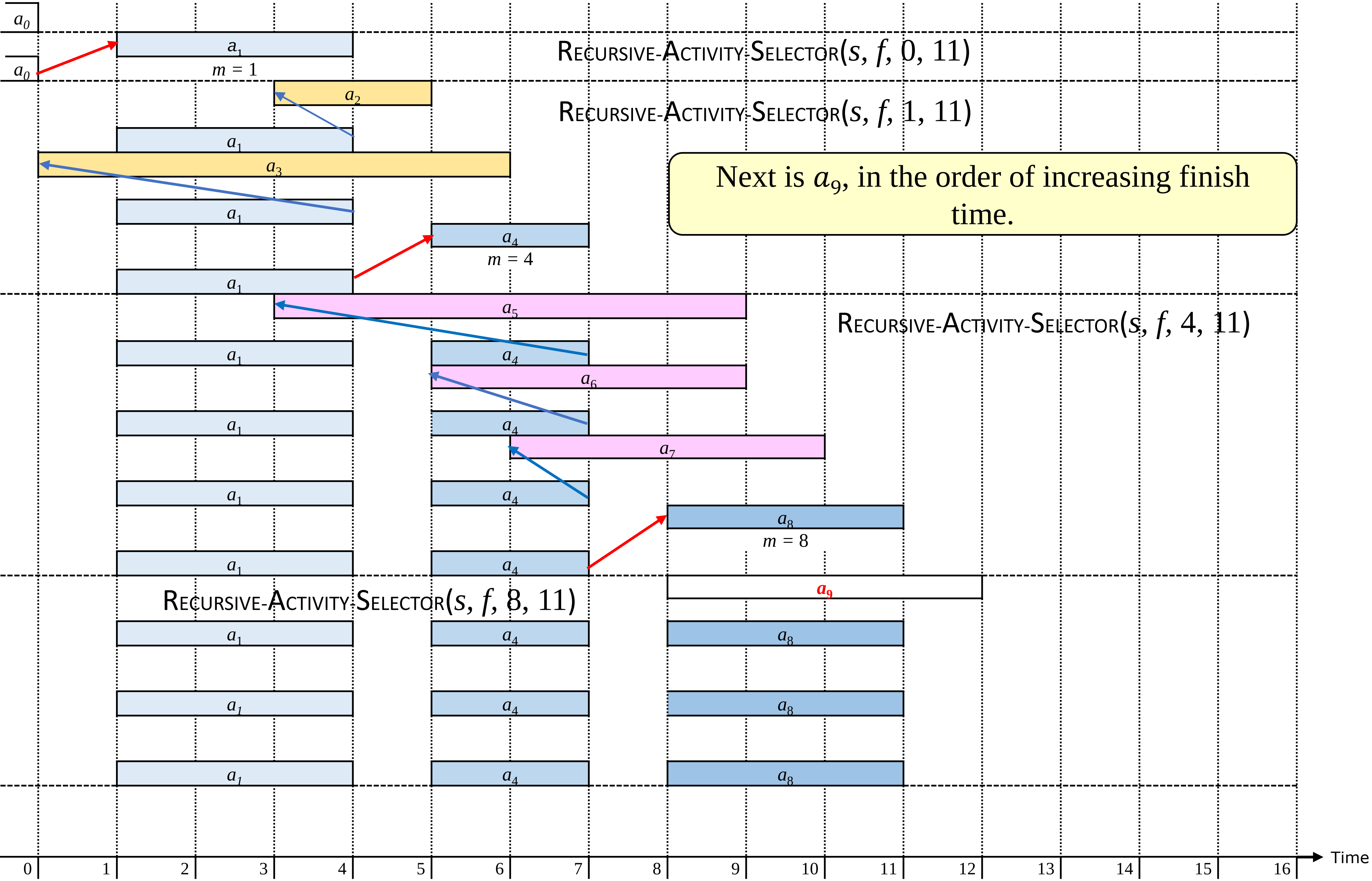
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



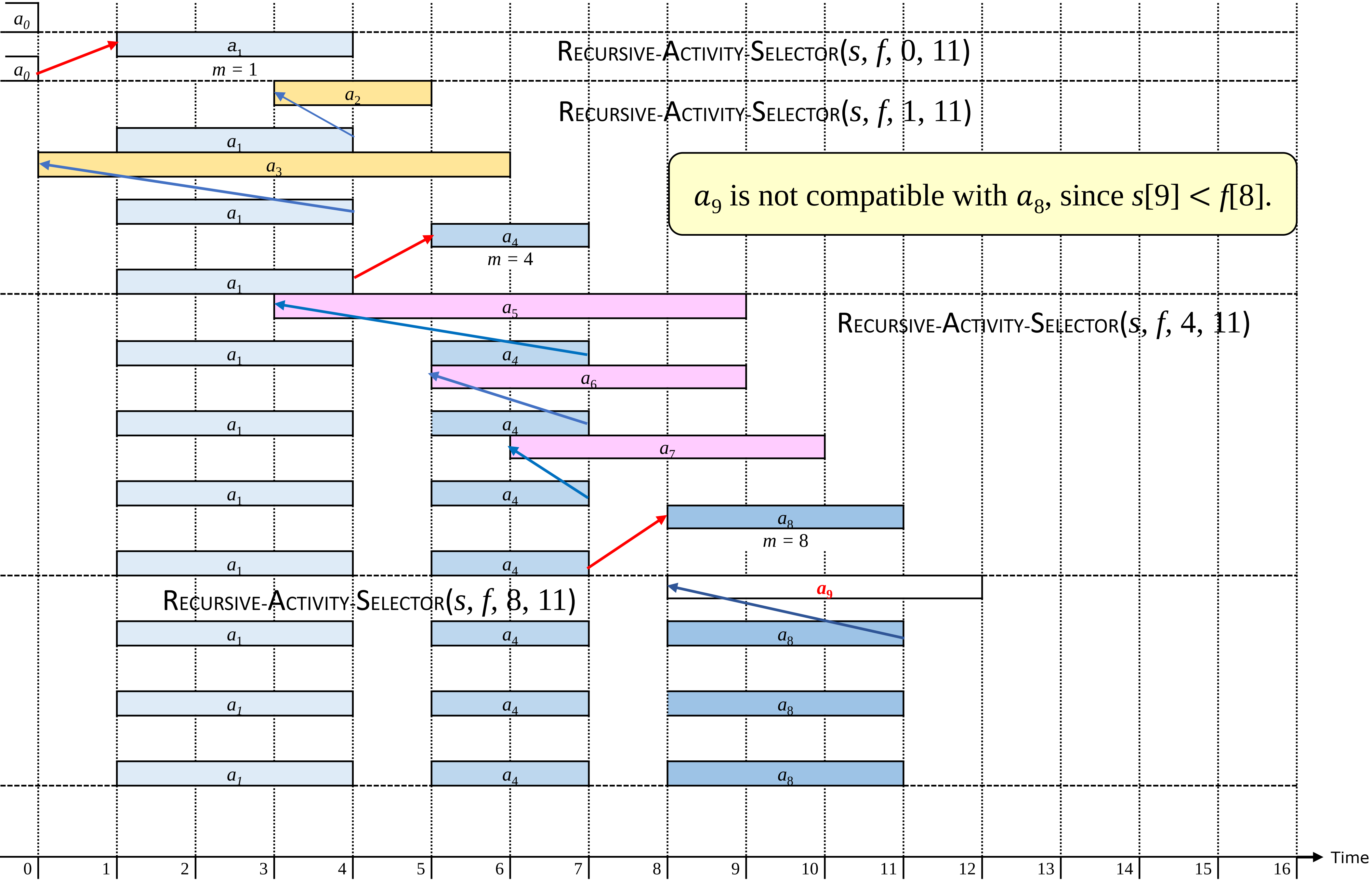
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



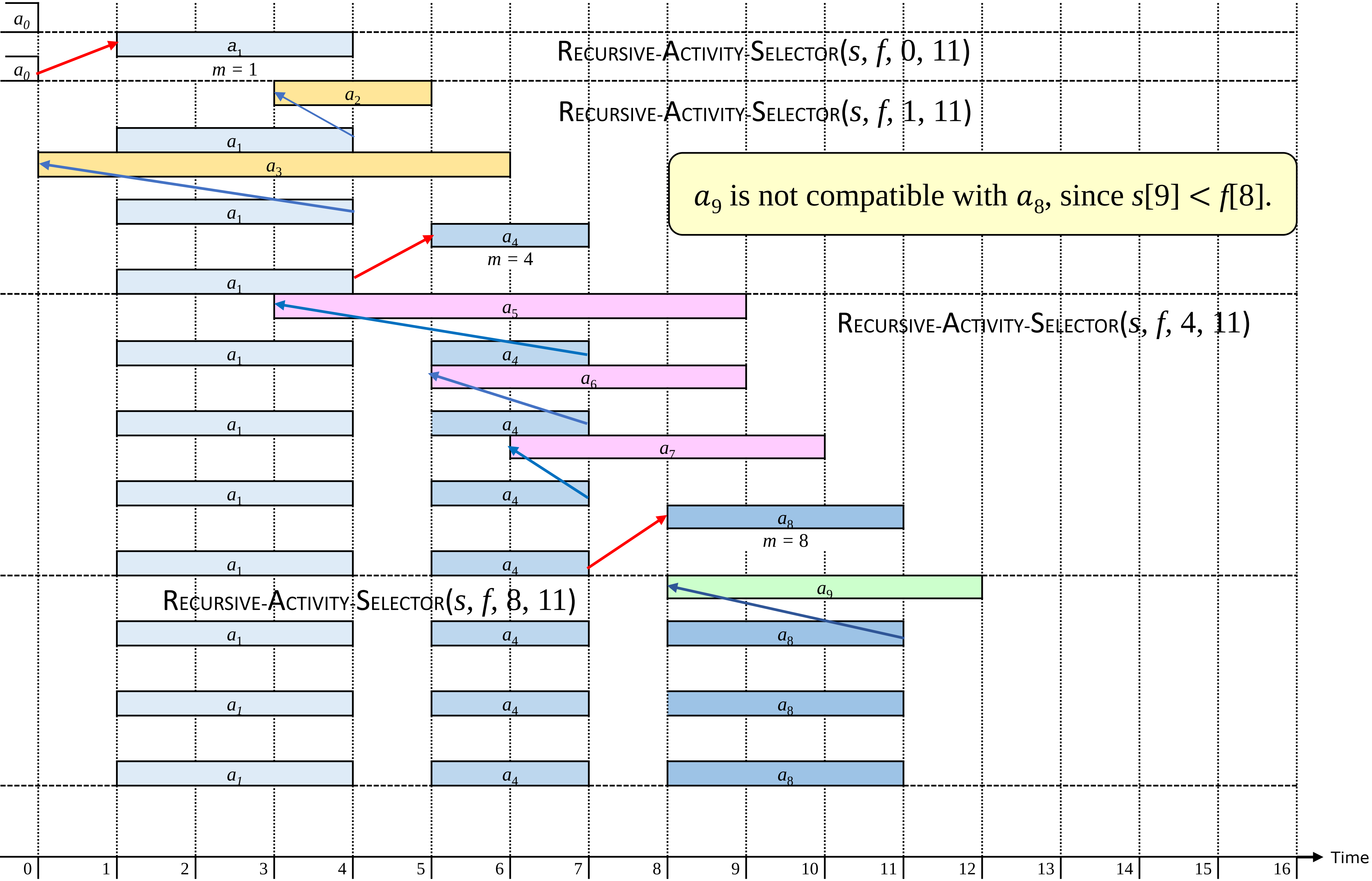
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	**8**	**12**
10	2	14
11	12	16



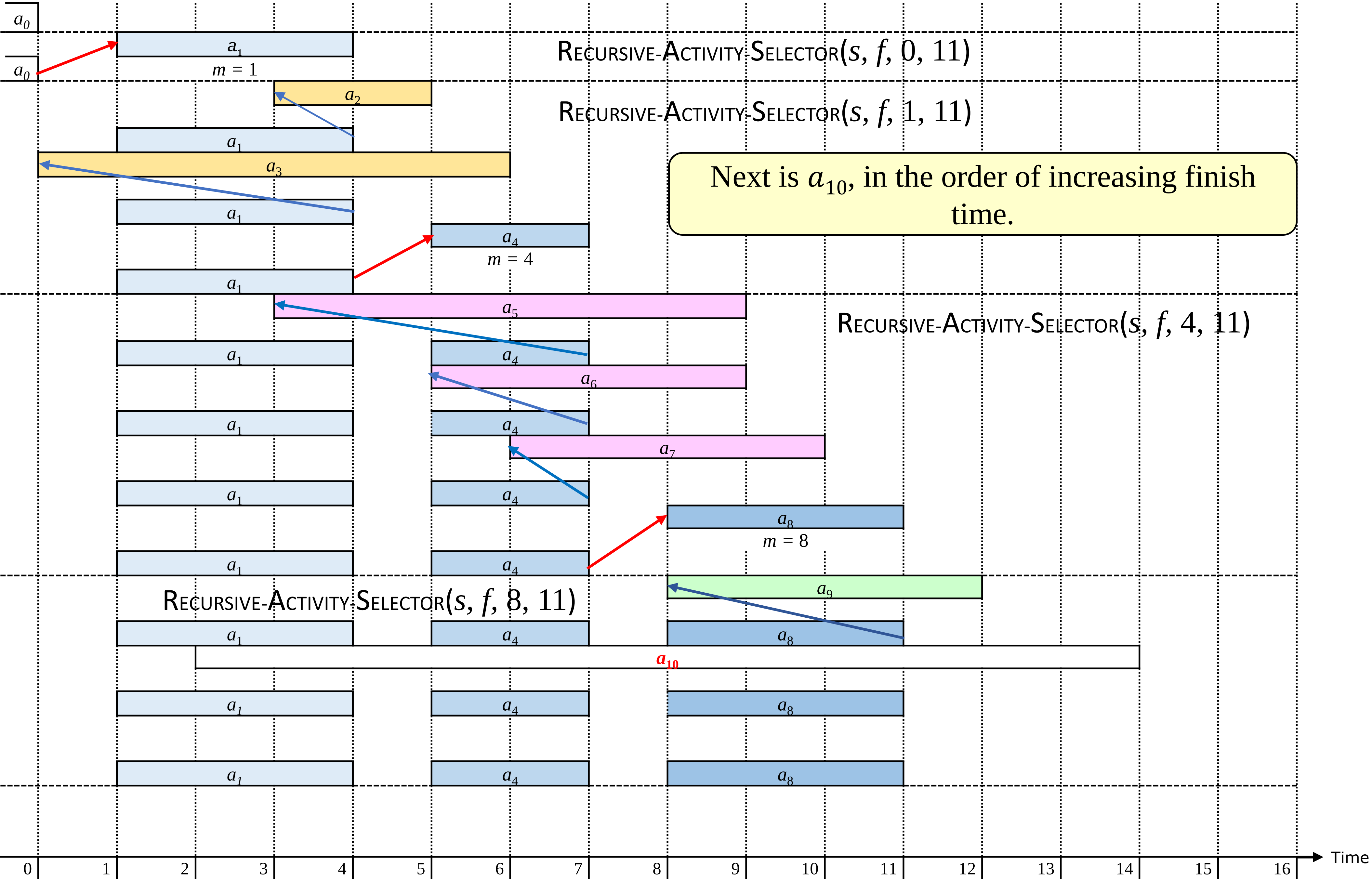
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	**8**	**12**
10	2	14
11	12	16



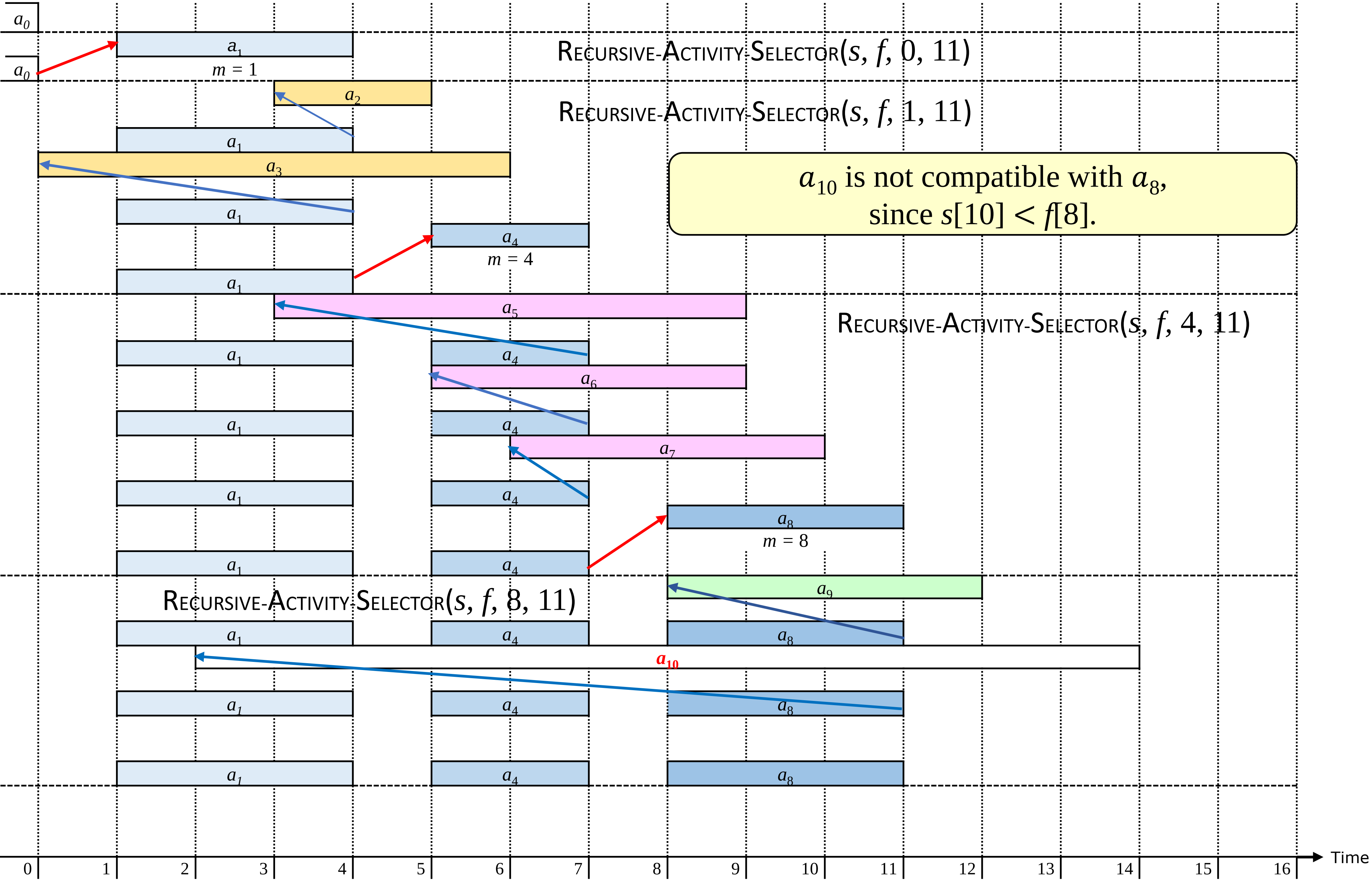
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



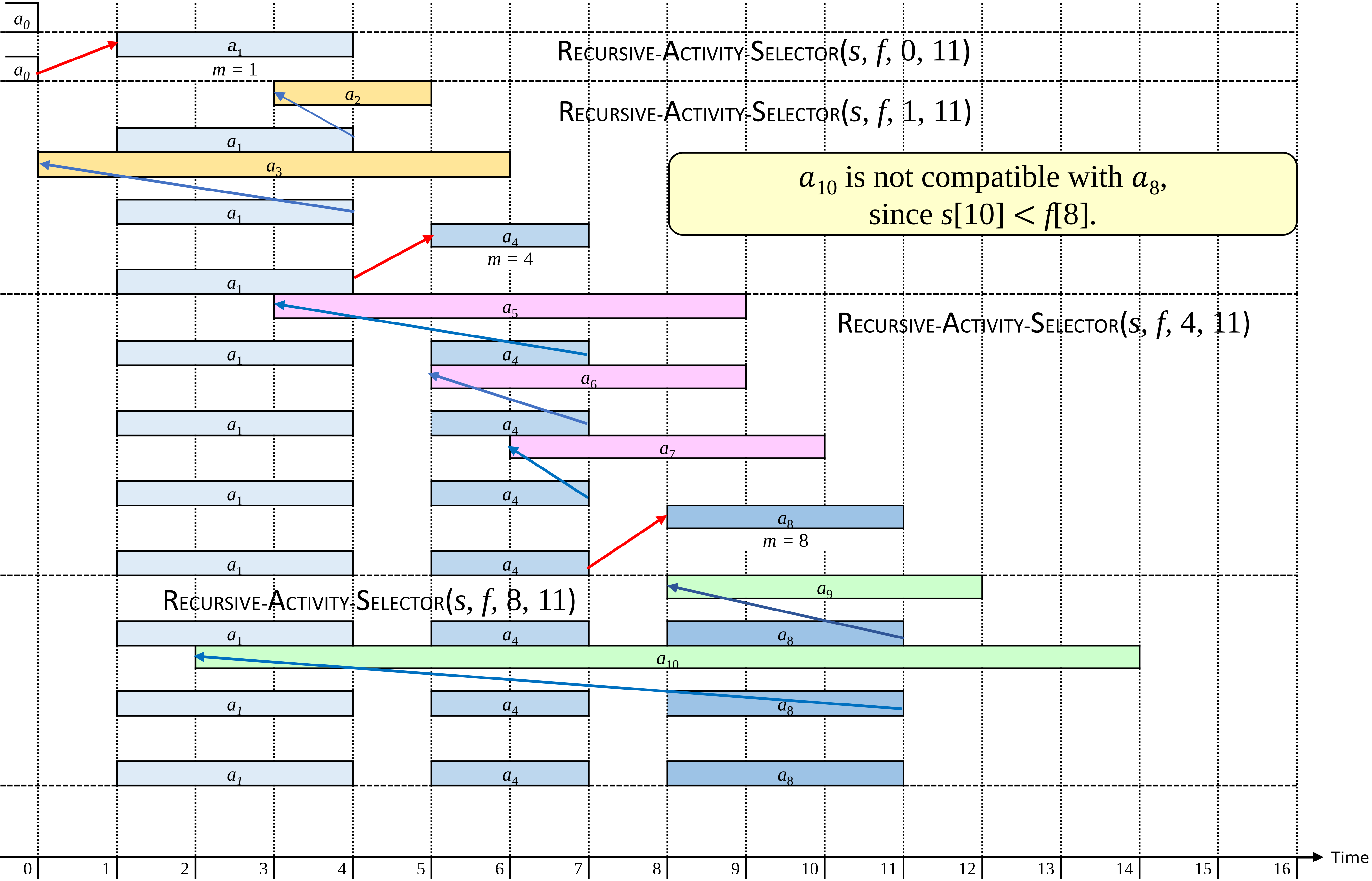
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	**2**	**14**
11	12	16



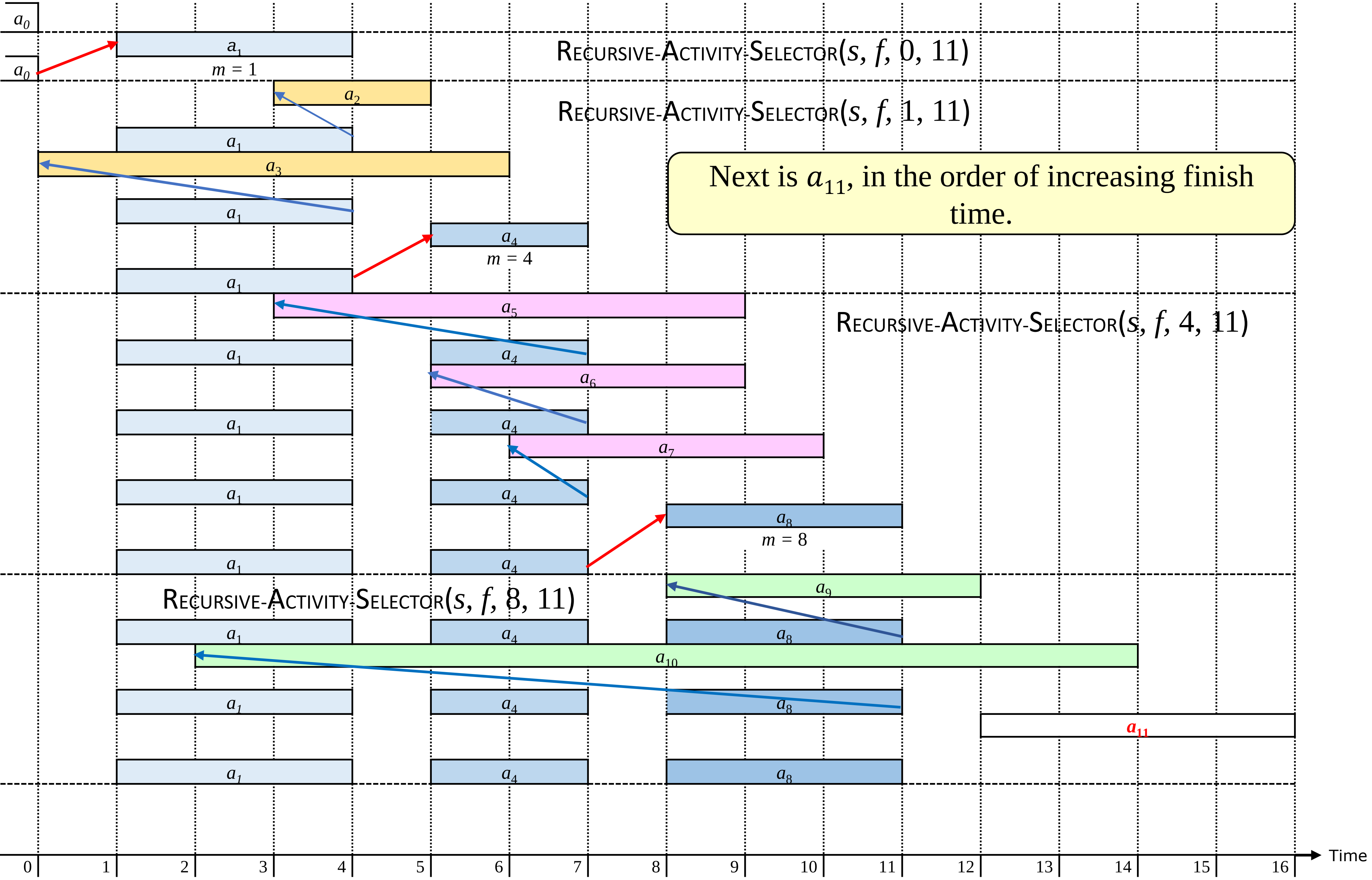
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	**2**	**14**
11	12	16



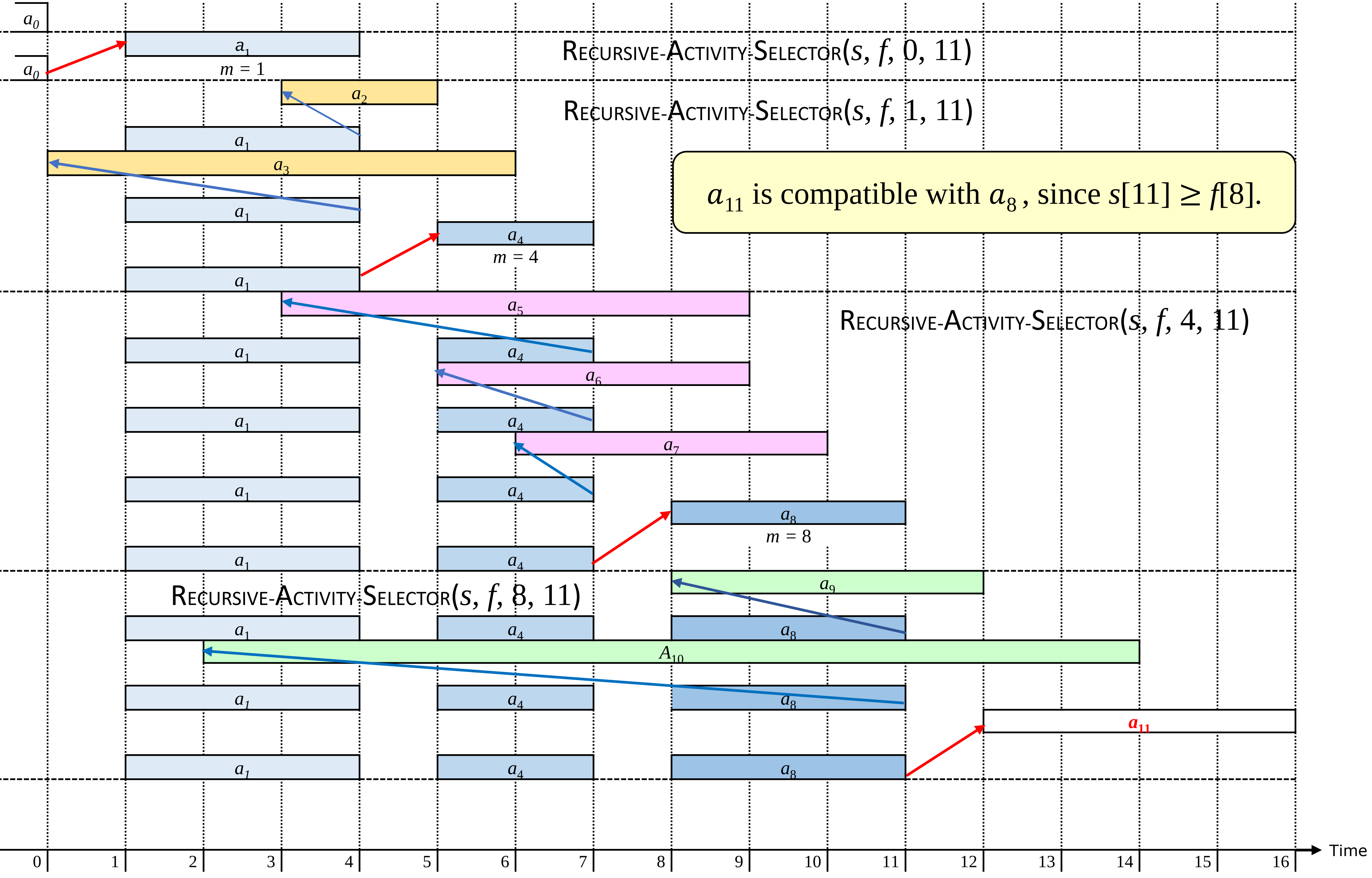
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



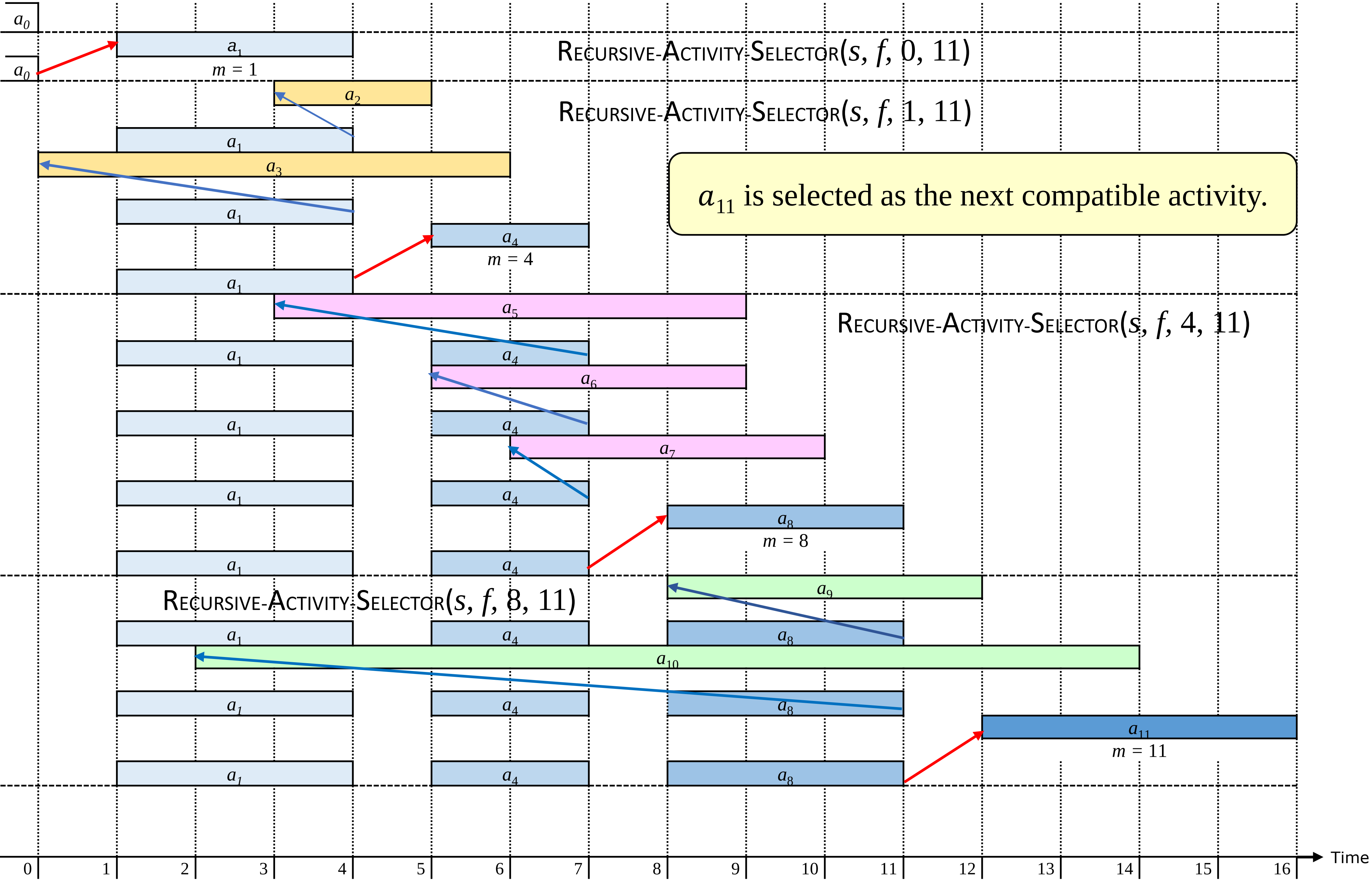
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	**12**	**16**



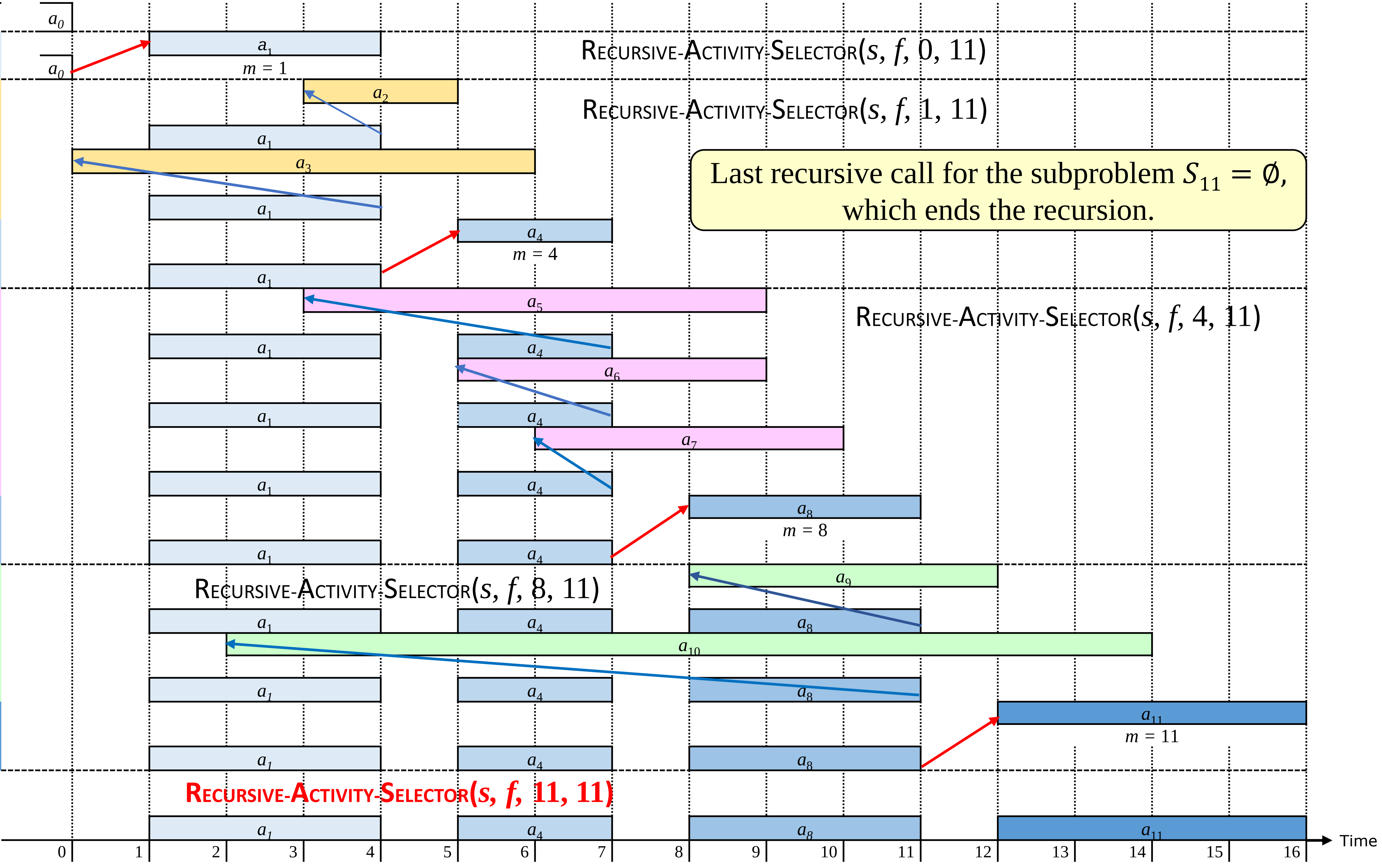
k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	**12**	**16**



k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



k	$s[k]$	$f[k]$
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

- The procedure RECURSIVE-ACTIVITY-SELECTOR can be converted to an iterative form, namely, GREEDY-ACTIVITY-SELECTOR in a straightforward manner.

Input:

1. Set, $S = \{a_1, a_2, \dots, a_n\}$ of n activities that wish to use a common resource, which can serve only one activity at a time.
2. Array ***s*** that contains the start time of the activities.
3. Array ***f*** that contains the finish time of the activities.

Output:

- Maximum-size subset of compatible activities in S.

Assumption:

- The ***n*** input activities are sorted by monotonically increasing finish time , $\boldsymbol{f_1 \leq f_2 \leq \dots \leq f_n}$. If not sorted we can sort them in this order in $O(n\ lg\ n)$ time, breaking ties arbitrarily.

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

GREEDY-ACTIVITY-SELECTOR (s, f)

```
n = s.length
A = {a_1}
k = 1
for m = 2 to n
    if s[m] ≥ f[k]
        A = A ∪ {a_m}
        k = m
return A
```

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

GREEDY-ACTIVITY-SELECTOR (s, f)

$n = s.length$

$A = \{a_1\}$

$k = 1$

for $m = 2$ **to** n

  if $s[m] \geq f[k]$

    $A = A \cup \{a_m\}$

    $k = m$

return A

n is assigned the total no. of activities in the input which are ordered by monotonically increasing finish time.

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

GREEDY-ACTIVITY-SELECTOR (s, f)

$n = s.length$

$A = \{a_1\}$

$k = 1$

for $m = 2$ **to** n

  if $s[m] \geq f[k]$

    $A = A \cup \{a_m\}$

    $k = m$

return A

n is assigned the total no. of activities in the input which are ordered by monotonically increasing finish time.

Initialize A to contain activity a_1, because initially a_1 is the first activity to finish in S.

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

GREEDY-ACTIVITY-SELECTOR (s, f)

$n = s.length$

n is assigned the total no. of activities in the input which are ordered by monotonically increasing finish time.

$A = \{a_1\}$

Initialize A to contain activity a_1, because initially a_1 is the first activity to finish in S.

$k = 1$

Initialize k to 1 the index of activity a_1. The variable k indexes the most recent addition to A.

for $m = 2$ **to** n

  if $s[m] \geq f[k]$

    $A = A \cup \{a_m\}$

    $k = m$

return A

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

```
GREEDY-ACTIVITY-SELECTOR (s, f)
    n = s.length
    A = {a_1}
    k = 1
    for m = 2 to n
        if s[m] ≥ f[k]
            A = A ∪ {a_m}
            k = m
    return A
```

n is assigned the total no. of activities in the input which are ordered by monotonically increasing finish time.

Initialize A to contain activity a_1, because initially a_1 is the first activity to finish in S.

Initialize k to 1 the index of activity a_1. The variable k indexes the most recent addition to A.

Since we consider the activities in order of monotonically increasing finish time, f_k is always the maximum finish time of any activity in A. That is, $f_k = \max\{f_i : a_i \in A\}$.

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

GREEDY-ACTIVITY-SELECTOR (s, f)

```
1   n = s.length
2   A = {a_1}
3   k = 1
4   for m = 2 to n
5       if s[m] ≥ f[k]
6           A = A ∪ {a_m}
7           k = m
8   return A
```

n is assigned the total no. of activities in the input which are ordered by monotonically increasing finish time.

Initialize A to contain activity a_1, because initially a_1 is the first activity to finish in S.

Initialize k to 1 the index of activity a_1. The variable k indexes the most recent addition to A.

The **for** loop of lines 4-7 finds the earliest activity in S_k to finish.

Since we consider the activities in order of monotonically increasing finish time, f_k is always the maximum finish time of any activity in A. That is, $f_k = \max\{f_i : a_i \in A\}$.

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

GREEDY-ACTIVITY-SELECTOR (s, f)

1 $n = s.length$

2 $A = \{a_1\}$

3 $k = 1$

4 **for** $m = 2$ **to** n

5   **if** $s[m] \geq f[k]$

6         $A = A \cup \{a_m\}$

7         $k = m$

8 **return** A

n is assigned the total no. of activities in the input which are ordered by monotonically increasing finish time.

Initialize A to contain activity a_1, because initially a_1 is the first activity to finish in S.

Initialize k to 1 the index of activity a_1. The variable k indexes the most recent addition to A.

The **for** loop of lines 4-7 finds the earliest activity in S_k to finish.

Since we consider the activities in order of monotonically increasing finish time, f_k is always the maximum finish time of any activity in A. That is, $f_k = \max\{f_i : a_i \in A\}$.

The loop considers each activity a_m in turn and adds a_m to A if it is compatible with all previously selected activities; such an activity is earliest in S_k to finish. To see whether activity a_m is compatible with every activity currently in A, it suffices by $f_k = \max\{f_i : a_i \in A\}$ to check (in line 5) that its start time s_m is not earlier than the finish time f_k of the activity most recently added to A.

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

GREEDY-ACTIVITY-SELECTOR (s, f)

```
1   n = s.length
2   A = {a_1}
3   k = 1
4   for m = 2 to n
5           if s[m] ≥ f[k]
6                   A = A ∪ {a_m}
7                   k = m
8   return A
```

n is assigned the total no. of activities in the input which are ordered by monotonically increasing finish time.

Initialize A to contain activity a_1, because initially a_1 is the first activity to finish in S.

Initialize k to 1 the index of activity a_1. The variable k indexes the most recent addition to A.

The **for** loop of lines 4-7 finds the earliest activity in S_k to finish.

Since we consider the activities in order of monotonically increasing finish time, f_k is always the maximum finish time of any activity in A. That is, $f_k = \max\{f_i : a_i \in A\}$.

The loop considers each activity a_m in turn and adds a_m to A if it is compatible with all previously selected activities; such an activity is earliest in S_k to finish. To see whether activity a_m is compatible with every activity currently in A, it suffices by $f_k = \max\{f_i : a_i \in A\}$ to check (in line 5) that its start time s_m is not earlier than the finish time f_k of the activity most recently added to A.

If the activity a_m is compatible, then lines 6-7 add activity a_m to A and set k to m.

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

GREEDY-ACTIVITY-SELECTOR (s, f)

1 $n = s.length$

2 $A = \{a_1\}$

3 $k = 1$

4 **for** $m = 2$ **to** n

5 **if** $s[m] \geq f[k]$

6 $A = A \cup \{a_m\}$

7 $k = m$

8 **return** A

n is assigned the total no. of activities in the input which are ordered by monotonically increasing finish time.

Initialize A to contain activity a_1, because initially a_1 is the first activity to finish in S.

Initialize k to 1 the index of activity a_1. The variable k indexes the most recent addition to A.

Since we consider the activities in order of monotonically increasing finish time, f_k is always the maximum finish time of any activity in A. That is, $f_k = \max\{f_i : a_i \in A\}$.

The **for** loop of lines 4-7 finds the earliest activity in S_k to finish.

The **for** loop of lines 4-7 runs until $m>n$.

The loop considers each activity a_m in turn and adds a_m to A if it is compatible with all previously selected activities; such an activity is earliest in S_k to finish. To see whether activity a_m is compatible with every activity currently in A, it suffices by $f_k = \max\{f_i : a_i \in A\}$ to check (in line 5) that its start time s_m is not earlier than the finish time f_k of the activity most recently added to A.

If the activity a_m is compatible, then lines 6-7 add activity a_m to A and set k to m.

An iterative greedy algorithm, GREEDY-ACTIVITY-SELECTOR

GREEDY-ACTIVITY-SELECTOR (s, f)

1 $n = s.length$

2 $A = \{a_1\}$

3 $k = 1$

4 **for** $m = 2$ **to** n

5 **if** $s[m] \geq f[k]$

6 $A = A \cup \{a_m\}$

7 $k = m$

8 **return** A

n is assigned the total no. of activities in the input which are ordered by monotonically increasing finish time.

Initialize A to contain activity a_1, because initially a_1 is the first activity to finish in S.

Initialize k to 1 the index of activity a_1. The variable k indexes the most recent addition to A.

Since we consider the activities in order of monotonically increasing finish time, f_k is always the maximum finish time of any activity in A. That is, $f_k = \max\{f_i : a_i \in A\}$.

The **for** loop of lines 4-7 finds the earliest activity in S_k to finish.

The **for** loop of lines 4-7 runs until $m>n$.

The loop considers each activity a_m in turn and adds a_m to A if it is compatible with all previously selected activities; such an activity is earliest in S_k to finish. To see whether activity a_m is compatible with every activity currently in A, it suffices by $f_k = \max\{f_i : a_i \in A\}$ to check (in line 5) that its start time s_m is not earlier than the finish time f_k of the activity most recently added to A.

Set A returns all the selected compatible activities in S.

If the activity a_m is compatible, then lines 6-7 add activity a_m to A and set k to m.

Time complexity Analysis of Greedy-Activity-Selector

Initial call: Greedy-Activity-Selector (s, f)

Algorithm:

Greedy-Activity-Selector (s, f)	Times executed
$n = s.length$	1
$A = \{a_1\}$	1
$k = 1$	1
for $m = 2$ **to** n	n
if $s[m] \geq f[k]$	$n-1$
$A = A \cup \{a_m\}$	$\leq (n-1)$
$k = m$	$\leq (n-1)$
return A	1

Hence, the running time of the iterative Greedy-Activity-Selector(s, f) is $\boldsymbol{\theta(n)}$.