Order Statistic Problems on Suffixes

Gianni Franceschini

University of Pisa

francesc@di.unipi.it

Order Statistic Problems

Order Statistic Problems

Generic Order Statistic Problem:

Order Statistic Problems

Generic Order Statistic Problem:

- Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),

Order Statistic Problems

Generic Order Statistic Problem:

- Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),
- and a *rank set* $\mathcal{R}$ (i.e. $k \in \{1, 2, \ldots, n\}$, for any $k \in \mathcal{R}$)

Order Statistic Problems

Generic Order Statistic Problem:

- Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),
- and a *rank set* $\mathcal{R}$ (i.e. $k \in \{1, 2, \ldots, n\}$, for any $k \in \mathcal{R}$)
- *find* the k*-th smallest element* in S, *for any* $k \in \mathcal{R}$.

Order Statistic Problems

Generic Order Statistic Problem:

- Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),
- and a *rank set* $\mathcal{R}$ (i.e. $k \in \{1, 2, \ldots, n\}$, for any $k \in \mathcal{R}$)
- *find* the k*-th smallest element* in S, *for any* $k \in \mathcal{R}$.

Classical order statistic problems:

Order Statistic Problems

Generic Order Statistic Problem:

- Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),
- and a *rank set* $\mathcal{R}$ (i.e. $k \in \{1, 2, \ldots, n\}$, for any $k \in \mathcal{R}$)
- *find* the k*-th smallest element* in S, *for any* $k \in \mathcal{R}$.

Classical order statistic problems:

- *Sorting*: extreme case where $\mathcal{R} = \{1, 2, \ldots n\}$.

Order Statistic Problems

Generic Order Statistic Problem:

- Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),
- and a *rank set* $\mathcal{R}$ (i.e. $k \in \{1, 2, \ldots, n\}$, for any $k \in \mathcal{R}$)
- *find* the k*-th smallest element* in S, *for any* $k \in \mathcal{R}$.

Classical order statistic problems:

- *Sorting*: extreme case where $\mathcal{R} = \{1, 2, \ldots n\}$.
- *Selection of the smallest* (largest) *element*: $\mathcal{R} = \{1\}$ ($\mathcal{R} = \{n\}$).

Order Statistic Problems

Generic Order Statistic Problem:

- Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),
- and a *rank set* $\mathcal{R}$ (i.e. $k \in \{1, 2, \ldots, n\}$, for any $k \in \mathcal{R}$)
- *find* the k*-th smallest element* in S, *for any* $k \in \mathcal{R}$.

Classical order statistic problems:

- *Sorting*: extreme case where $\mathcal{R} = \{1, 2, \ldots n\}$.
- *Selection of the smallest* (largest) *element*: $\mathcal{R} = \{1\}$ ($\mathcal{R} = \{n\}$).
- *Selection of the smallest AND largest elements*: $\mathcal{R} = \{1, n\}$.

Order Statistic Problems

Generic Order Statistic Problem:

- Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),
- and a *rank set* $\mathcal{R}$ (i.e. $k \in \{1, 2, \ldots, n\}$, for any $k \in \mathcal{R}$)
- *find* the k*-th smallest element* in S, *for any* $k \in \mathcal{R}$.

Classical order statistic problems:

- *Sorting*: extreme case where $\mathcal{R} = \{1, 2, \ldots n\}$.
- *Selection of the smallest* (largest) *element*: $\mathcal{R} = \{1\}$ ($\mathcal{R} = \{n\}$).
- *Selection of the smallest AND largest elements*: $\mathcal{R} = \{1, n\}$.
- *Selection of the median(s) element(s)*: $\mathcal{R} = \{\lfloor n/2 \rfloor, \lceil n/2 \rceil\}$.

Order Statistic Problems

> *Generic Order Statistic Problem*:
>
> - Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),
> - and a *rank set* $\mathcal{R}$ (i.e. $k \in \{1, 2, \ldots, n\}$, for any $k \in \mathcal{R}$)
> - *find* the k*-th smallest element* in S, *for any* $k \in \mathcal{R}$.

Classical order statistic problems:

- *Sorting*: extreme case where $\mathcal{R} = \{1, 2, \ldots n\}$.
- *Selection of the smallest* (largest) *element*: $\mathcal{R} = \{1\}$ ($\mathcal{R} = \{n\}$).
- *Selection of the smallest AND largest elements*: $\mathcal{R} = \{1, n\}$.
- *Selection of the median(s) element(s)*: $\mathcal{R} = \{\lfloor n/2 \rfloor, \lceil n/2 \rceil\}$.
- *Generic selection*: $\mathcal{R} = \{k\}$, for any k.

Order Statistic Problems

Generic Order Statistic Problem:

- Given a *set* $\mathcal{S}$ of n elements (drawn from a *total order* $(\mathcal{U}, <)$),
- and a *rank set* $\mathcal{R}$ (i.e. $k \in \{1, 2, \ldots, n\}$, for any $k \in \mathcal{R}$)
- *find* the k*-th smallest element* in S, *for any* $k \in \mathcal{R}$.

Classical order statistic problems:

- *Sorting*: extreme case where $\mathcal{R} = \{1, 2, \ldots n\}$.
- *Selection of the smallest* (largest) *element*: $\mathcal{R} = \{1\}$ ($\mathcal{R} = \{n\}$).
- *Selection of the smallest AND largest elements*: $\mathcal{R} = \{1, n\}$.
- *Selection of the median(s) element(s)*: $\mathcal{R} = \{\lfloor n/2 \rfloor, \lceil n/2 \rceil\}$.
- *Generic selection*: $\mathcal{R} = \{k\}$, for any k.
- *Generic multi-selection*: $\mathcal{R} \subset \{1, 2, \ldots, n\}$.

Order Statistic Problems

Usual settings for Order Statistic Problems:

Order Statistic Problems

Usual settings for Order Statistic Problems:

- $\mathcal{S}$ is a *set*...

Order Statistic Problems

Usual settings for Order Statistic Problems:

- $\mathcal{S}$ is a *set*. . . but it can also be a *multi-set* (i.e. multiple occurrences of an element allowed), if the *rank of an element* is well-defined (usually by considering $\mathcal{S}$ as a sequence).

Order Statistic Problems

Usual settings for Order Statistic Problems:

- $\mathcal{S}$ is a *set*... but it can also be a *multi-set* (i.e. multiple occurrences of an element allowed), if the *rank of an element* is well-defined (usually by considering $\mathcal{S}$ as a sequence).
- The *input elements* are *unidimensional objects* (comparable in $O(1)$ time)...

Order Statistic Problems

Usual settings for Order Statistic Problems:

- $\mathcal{S}$ is a *set*... but it can also be a *multi-set* (i.e. multiple occurrences of an element allowed), if the *rank of an element* is well-defined (usually by considering $\mathcal{S}$ as a sequence).
- The *input elements* are *unidimensional objects* (comparable in $O(1)$ time)...
- ... but results can be *easily extended* to *multidimensional objects*, like *strings* and *vectors*, and the *lexicographical order*.

Order Statistic Problems

Usual settings for Order Statistic Problems:

- $\mathcal{S}$ is a *set*… but it can also be a *multi-set* (i.e. multiple occurrences of an element allowed), if the *rank of an element* is well-defined (usually by considering $\mathcal{S}$ as a sequence).
- The *input elements* are *unidimensional objects* (comparable in $O(1)$ time)…
- … but results can be *easily extended* to *multidimensional objects*, like *strings* and *vectors*, and the *lexicographical order*.
- *Not so easy* when the input objects are the *suffixes of a sequence* T.

Order Statistic Problems

Usual settings for Order Statistic Problems:

- $\mathcal{S}$ is a *set*... but it can also be a *multi-set* (i.e. multiple occurrences of an element allowed), if the *rank of an element* is well-defined (usually by considering $\mathcal{S}$ as a sequence).
- The *input elements* are *unidimensional objects* (comparable in $O(1)$ time)...
- ... but results can be *easily extended* to *multidimensional objects*, like *strings* and *vectors*, and the *lexicographical order*.
- *Not so easy* when the input objects are the *suffixes of a sequence* T.
- In all cases, the *comparison model* is considered: the input objects (or for the cases of strings, vectors and suffixes, the elements they are made of) *can only be compared*.

Generic Suffix Selection

Generic Selection

Generic Selection

Given a set $\mathcal{S}$ of n elements and an integer $k \in \{1, \ldots, n\}$, find the k-th smallest element of $\mathcal{S}$.

Generic Selection

Given a set $\mathcal{S}$ of n elements and an integer $k \in \{1, \ldots, n\}$, find the k-th smallest element of $\mathcal{S}$.

- Simple solution: *sort* $\mathcal{S}$ in $\Theta(n \log n)$ time and *select* the k-th smallest element in $O(1)$ time.

Generic Selection

Given a set $\mathcal{S}$ of n elements and an integer $k \in \{1, \ldots, n\}$, find the k-th smallest element of $\mathcal{S}$.

- Simple solution: *sort* $\mathcal{S}$ in $\Theta(n \log n)$ time and *select* the k-th smallest element in $O(1)$ time.
- Is this optimal? *Are the asymptotic complexities of sorting and selection the same?*

Generic Selection

Given a set $\mathcal{S}$ of n elements and an integer $k \in \{1, \ldots, n\}$, find the k-th smallest element of $\mathcal{S}$.

- Simple solution: *sort* $\mathcal{S}$ in $\Theta(n \log n)$ time and *select* the k-th smallest element in $O(1)$ time.
- Is this optimal? *Are the asymptotic complexities of sorting and selection the same?*
- That was unknown until the *early '70s*:

Generic Selection

Given a set $\mathcal{S}$ *of* n *elements and an integer* $k \in \{1, \ldots, n\}$, *find the* k*-th smallest element of* $\mathcal{S}$.

- Simple solution: *sort* $\mathcal{S}$ in $\Theta(n \log n)$ time and *select* the k-th smallest element in $O(1)$ time.
- Is this optimal? *Are the asymptotic complexities of sorting and selection the same?*
- That was unknown until the *early '70s*:
  - famous "textbook" results [Blum, Floyd, Pratt, Rivest, Tarjan, STOC 1972, JCSS 7, 1973],

Generic Selection

Given a set $\mathcal{S}$ of n elements and an integer $k \in \{1, \ldots, n\}$, find the k-th smallest element of $\mathcal{S}$.

- Simple solution: *sort* $\mathcal{S}$ in $\Theta(n \log n)$ time and *select* the k-th smallest element in $O(1)$ time.
- Is this optimal? *Are the asymptotic complexities of sorting and selection the same?*
- That was unknown until the *early '70s*:
  - famous "textbook" results [Blum, Floyd, Pratt, Rivest, Tarjan, STOC 1972, JCSS 7, 1973],
  - *generic selection* requires $O(n)$ time in the worst case.

Generic Selection

Given a set $\mathcal{S}$ of n elements and an integer $k \in \{1, \ldots, n\}$, find the k-th smallest element of $\mathcal{S}$.

- Simple solution: *sort* $\mathcal{S}$ in $\Theta(n \log n)$ time and *select* the k-th smallest element in $O(1)$ time.
- Is this optimal? *Are the asymptotic complexities of sorting and selection the same?*
- That was unknown until the *early '70s*:
  - famous "textbook" results [Blum, Floyd, Pratt, Rivest, Tarjan, STOC 1972, JCSS 7, 1973],
  - *generic selection* requires $O(n)$ time in the worst case.
  - classic example of *divide et impera* approach.

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.
4. *Partition the input* into two subsets $\mathcal{L}$ and $\mathcal{R}$ according to x ($y < x$, for any $y \in \mathcal{L}$).

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.
4. *Partition the input* into two subsets $\mathcal{L}$ and $\mathcal{R}$ according to x ($y < x$, for any $y \in \mathcal{L}$).
5. *Recursively select*

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.
4. *Partition the input* into two subsets $\mathcal{L}$ and $\mathcal{R}$ according to x ($y < x$, for any $y \in \mathcal{L}$).
5. *Recursively select*
   - the k*-th* smallest element in $\mathcal{L}$, if $k \leq |\mathcal{L}|$,

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.
4. *Partition the input* into two subsets $\mathcal{L}$ and $\mathcal{R}$ according to x ($y < x$, for any $y \in \mathcal{L}$).
5. *Recursively select*
   - the k*-th* smallest element in $\mathcal{L}$, if $k \leq |\mathcal{L}|$,
   - or the $(k - |\mathcal{L}|)$*-th* smallest element in $\mathcal{R}$, if $k > |\mathcal{L}|$.

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.
4. *Partition the input* into two subsets $\mathcal{L}$ and $\mathcal{R}$ according to x ($y < x$, for any $y \in \mathcal{L}$).
5. *Recursively select*
   - the k-*th* smallest element in $\mathcal{L}$, if $k \leq |\mathcal{L}|$,
   - or the $(k - |\mathcal{L}|)$-*th* smallest element in $\mathcal{R}$, if $k > |\mathcal{L}|$.

Simple analysis:

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.
4. *Partition the input* into two subsets $\mathcal{L}$ and $\mathcal{R}$ according to x ($y < x$, for any $y \in \mathcal{L}$).
5. *Recursively select*
   - the k*-th* smallest element in $\mathcal{L}$, if $k \leq |\mathcal{L}|$,
   - or the $(k - |\mathcal{L}|)$*-th* smallest element in $\mathcal{R}$, if $k > |\mathcal{L}|$.

Simple analysis:

- At least *half* of the $n/5$ groups have 3 elements *greater than* x...

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.
4. *Partition the input* into two subsets $\mathcal{L}$ and $\mathcal{R}$ according to x ($y < x$, for any $y \in \mathcal{L}$).
5. *Recursively select*
   - the k*-th* smallest element in $\mathcal{L}$, if $k \leq |\mathcal{L}|$,
   - or the $(k - |\mathcal{L}|)$*-th* smallest element in $\mathcal{R}$, if $k > |\mathcal{L}|$.

Simple analysis:

- At least *half* of the $n/5$ groups have 3 elements *greater than* x...
- ...we have a *lower bound on the size of* $\mathcal{R}$: $|\mathcal{R}| \geq \frac{3}{10}n$.

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.
4. *Partition the input* into two subsets $\mathcal{L}$ and $\mathcal{R}$ according to x ($y < x$, for any $y \in \mathcal{L}$).
5. *Recursively select*
   - the k*-th* smallest element in $\mathcal{L}$, if $k \leq |\mathcal{L}|$,
   - or the $(k - |\mathcal{L}|)$*-th* smallest element in $\mathcal{R}$, if $k > |\mathcal{L}|$.

Simple analysis:

- At least *half* of the $n/5$ groups have 3 elements *greater than* x...
- ...we have a *lower bound on the size of* $\mathcal{R}$: $|\mathcal{R}| \geq \frac{3}{10}n$.
- Therefore $T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + O(n)$

Selection in $O(n)$ time: [Blum, Floyd, Pratt, Rivest, Tarjan, 1973]

1. *Divide the input* into $n/5$ *groups* of 5 elements each.
2. *Find the median* of each group (e.g. by insertion sorting).
3. *Recursively find the median* x of the $n/5$ medians found in step 2.
4. *Partition the input* into two subsets $\mathcal{L}$ and $\mathcal{R}$ according to x ($y < x$, for any $y \in \mathcal{L}$).
5. *Recursively select*
   - the k*-th* smallest element in $\mathcal{L}$, if $k \leq |\mathcal{L}|$,
   - or the $(k - |\mathcal{L}|)$*-th* smallest element in $\mathcal{R}$, if $k > |\mathcal{L}|$.

Simple analysis:

- At least *half* of the $n/5$ groups have 3 elements *greater than* x...
- ...we have a *lower bound on the size of* $\mathcal{R}$: $|\mathcal{R}| \geq \frac{3}{10}n$.
- Therefore $T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + O(n) = O(n)$.

The Lexicographical Order

The Lexicographical Order

Given two sequences x, y we denote with $lcp(x, y)$ the *length of the longest common prefix* of x and y.

The Lexicographical Order

Given two sequences x, y we denote with $lcp(x, y)$ the *length of the longest common prefix* of x and y.

x is *lexicographically smaller than* y

if and only if

$x[l+1] < y[l+1]$, where $l = lcp(x, y)$

The Lexicographical Order

Given two sequences x, y we denote with $lcp(x, y)$ the *length of the longest common prefix* of x and y.

x is *lexicographically smaller than* y

if and only if

$x[l+1] < y[l+1]$, where $l = lcp(x, y)$

or x is a *proper prefix* of y

The Lexicographical Order

Given two sequences x, y we denote with $lcp(x, y)$ the *length of the longest common prefix* of x and y.

x is *lexicographically smaller than* y
if and only if
$x[l+1] < y[l+1]$, where $l = lcp(x, y)$
or x is a *proper prefix* of y

- When working with suffixes of a sequence T, it is customary to assume that
  - T *has* $n+1$ *elements* and
  - $T[n+1]$ *is smaller than any other* $T[j]$.

The Lexicographical Order

Given two sequences x, y we denote with $lcp(x, y)$ the *length of the longest common prefix* of x and y.

x is *lexicographically smaller than* y

if and only if

$x[l+1] < y[l+1]$, where $l = lcp(x, y)$

or x is a *proper prefix* of y

- When working with suffixes of a sequence T, it is customary to assume that
  - T *has* $n+1$ *elements* and
  - $T[n+1]$ *is smaller than any other* $T[j]$.
- We represent $T[n+1]$ with $\bullet$.

Generic Suffix Selection

Generic Suffix Selection

This time we deal with

- *suffixes*, i.e. $\mathcal{S} = \{T_1, T_2, \ldots T_n\}$, where $T_i = T[i \cdots n]$

Generic Suffix Selection

This time we deal with

- *suffixes*, i.e. $\mathcal{S} = \{T_1, T_2, \ldots T_n\}$, where $T_i = T[i \cdots n]$
- *the lexicographical order.*

Generic Suffix Selection

This time we deal with

- *suffixes*, i.e. $\mathcal{S} = \{T_1, T_2, \ldots T_n\}$, where $T_i = T[i \cdots n]$
- *the lexicographical order.*

> *Given a sequence of n elements T and an integer $k \in \{1, \ldots, n\}$, find the k-th lexicographically smallest suffix of T.*

Generic Suffix Selection

This time we deal with

- *suffixes*, i.e. $\mathcal{S} = \{T_1, T_2, \ldots T_n\}$, where $T_i = T[i \cdots n]$
- *the lexicographical order.*

> *Given a sequence of n elements T and an integer $k \in \{1, \ldots, n\}$, find the k-th lexicographically smallest suffix of T.*

It is well known that the *suffixes of T can be sorted* in $O(n \log n)$ *time* in the worst case:

Generic Suffix Selection

This time we deal with

- *suffixes*, i.e. $\mathcal{S} = \{T_1, T_2, \ldots T_n\}$, where $T_i = T[i \cdots n]$
- *the lexicographical order.*

> *Given a sequence of n elements T and an integer $k \in \{1, \ldots, n\}$, find the k-th lexicographically smallest suffix of T.*

It is well known that the *suffixes of T can be sorted* in $O(n \log n)$ *time* in the worst case:

- Directly, by building the *Suffix Array* [Manber, Myers, SICOMP 22, 1993].

Generic Suffix Selection

This time we deal with

- *suffixes*, i.e. $\mathcal{S} = \{T_1, T_2, \ldots T_n\}$, where $T_i = T[i \cdots n]$
- *the lexicographical order.*

> *Given a sequence of n elements T and an integer $k \in \{1, \ldots, n\}$, find the k-th lexicographically smallest suffix of T.*

It is well known that the *suffixes of T can be sorted* in $O(n \log n)$ *time* in the worst case:

- Directly, by building the *Suffix Array* [Manber, Myers, SICOMP 22, 1993].
- Indirectly, through the *Suffix Tree* [Farach, FOCS 1997].

Generic Suffix Selection

This time we deal with

- *suffixes*, i.e. $\mathcal{S} = \{T_1, T_2, \ldots T_n\}$, where $T_i = T[i \cdots n]$
- *the lexicographical order.*

> *Given a sequence of n elements T and an integer $k \in \{1, \ldots, n\}$, find the k-th lexicographically smallest suffix of T.*

It is well known that the *suffixes of T can be sorted* in $O(n \log n)$ *time* in the worst case:

- Directly, by building the *Suffix Array* [Manber, Myers, SICOMP 22, 1993].
- Indirectly, through the *Suffix Tree* [Farach, FOCS 1997].

Natural question:

Are the complexities of Suffix Sorting and Suffix Selection the same?

Generic Suffix Selection

Generic Suffix Selection

- Practical motivations: a fast suffix selection has potential applications in *bioinformatics*, *information retrieval*. . .

Generic Suffix Selection

- Practical motivations: a fast suffix selection has potential applications in *bioinformatics*, *information retrieval*...
- ...but the problem has mainly a *theoretical appealing*.

Generic Suffix Selection

- Practical motivations: a fast suffix selection has potential applications in *bioinformatics*, *information retrieval*. . .
- . . . but the problem has mainly a *theoretical appealing*.

Complexity established in [Franceschini, Muthukrishnan, STOC 2007]:

Generic Suffix Selection

- Practical motivations: a fast suffix selection has potential applications in *bioinformatics*, *information retrieval*...
- ...but the problem has mainly a *theoretical appealing*.

Complexity established in [Franceschini, Muthukrishnan, STOC 2007]:

Suffix Selection requires $O(n)$ *time in the worst case*

Generic Suffix Selection

- Practical motivations: a fast suffix selection has potential applications in *bioinformatics*, *information retrieval*...
- ...but the problem has mainly a *theoretical appealing*.

Complexity established in [Franceschini, Muthukrishnan, STOC 2007]:

Suffix Selection requires $O(n)$ time in the worst case

- The divide and conquer approach used in [Blum et al, JCSS 7, 1973] *is not viable for suffix selection*.

Generic Suffix Selection

- Practical motivations: a fast suffix selection has potential applications in *bioinformatics*, *information retrieval*...
- ...but the problem has mainly a *theoretical appealing*.

Complexity established in [Franceschini, Muthukrishnan, STOC 2007]:

Suffix Selection requires $O(n)$ *time in the worst case*

- The divide and conquer approach used in [Blum et al, JCSS 7, 1973] *is not viable for suffix selection*.
  - If the approach was applied to suffixes, the two *recursive subproblems* (the finding of the median of medians and the recursive application on $\mathcal{L}$ or $\mathcal{R}$) *would not be instances of the Suffix Selection problem* anymore...

Generic Suffix Selection

- Practical motivations: a fast suffix selection has potential applications in *bioinformatics*, *information retrieval*...
- ...but the problem has mainly a *theoretical appealing*.

Complexity established in [Franceschini, Muthukrishnan, STOC 2007]:

Suffix Selection requires $O(n)$ time in the worst case

- The divide and conquer approach used in [Blum et al, JCSS 7, 1973] *is not viable for suffix selection*.
  - If the approach was applied to suffixes, the two *recursive subproblems* (the finding of the median of medians and the recursive application on $\mathcal{L}$ or $\mathcal{R}$) *would not be instances of the Suffix Selection problem* anymore...
  - ...same sequence T but only *a fraction of the n suffixes* would be considered in the sub-problems.

Generic Suffix Selection

- Practical motivations: a fast suffix selection has potential applications in *bioinformatics*, *information retrieval*...
- ...but the problem has mainly a *theoretical appealing*.

Complexity established in [Franceschini, Muthukrishnan, STOC 2007]:

Suffix Selection requires $O(n)$ time in the worst case

- The divide and conquer approach used in [Blum et al, JCSS 7, 1973] *is not viable for suffix selection*.
  - If the approach was applied to suffixes, the two *recursive subproblems* (the finding of the median of medians and the recursive application on $\mathcal{L}$ or $\mathcal{R}$) *would not be instances of the Suffix Selection problem* anymore...
  - ...same sequence T but only *a fraction of the n suffixes* would be considered in the sub-problems.
- However, we will use the selection algorithm in [Blum et al, JCSS 7, 1973] as a *basic tool for suffix selection*.

Suffix selection, first attempt

Suffix selection, first attempt

Phase-based approach.

Suffix selection, first attempt

Phase-based approach.

For each phase t we have the following.

Suffix selection, first attempt

Phase-based approach.

For each phase t we have the following.

- A prefix σ_t of the k*-th smallest suffix*. This represents our current *knowledge* about the wanted suffix.

Suffix selection, first attempt

Phase-based approach.

For each phase t we have the following.

- A prefix σ_t of the k*-th smallest suffix*. This represents our current *knowledge* about the wanted suffix.
- A set of *active suffixes* $\mathcal{A}_t$. It contains all the suffixes *with* σ_t *as a prefix* (that is all the suffixes that could still be the k-th smallest suffix at that point)

Suffix selection, first attempt

Phase-based approach.

For each phase t we have the following.

- A prefix σ_t of the k*-th smallest suffix*. This represents our current *knowledge* about the wanted suffix.
- A set of *active suffixes* $\mathcal{A}_t$. It contains all the suffixes *with* σ_t *as a prefix* (that is all the suffixes that could still be the k-th smallest suffix at that point)
- A set of *inactive suffixes* $\mathcal{I}_t$ with the suffixes that do not have σ_t as a prefix.

Suffix selection, first attempt

Phase-based approach.

For each phase t we have the following.

- A prefix σ_t of the k*-th smallest suffix*. This represents our current *knowledge* about the wanted suffix.
- A set of *active suffixes* $\mathcal{A}_t$. It contains all the suffixes *with* σ_t *as a prefix* (that is all the suffixes that could still be the k-th smallest suffix at that point)
- A set of *inactive suffixes* $\mathcal{I}_t$ with the suffixes that do not have σ_t as a prefix.
- The number l_t of the suffixes *lexicographically less than any of the active suffixes* of phase t.

Suffix selection, first attempt

Phase-based approach.

For each phase t we have the following.

- A prefix σ_t of the *k-th smallest suffix*. This represents our current *knowledge* about the wanted suffix.
- A set of *active suffixes* $\mathcal{A}_t$. It contains all the suffixes *with σ_t as a prefix* (that is all the suffixes that could still be the k-th smallest suffix at that point)
- A set of *inactive suffixes* $\mathcal{I}_t$ with the suffixes that do not have σ_t as a prefix.
- The number l_t of the suffixes *lexicographically less than any of the active suffixes* of phase t.

Our knowledge about the k-th smallest suffix is increased during *Phase Transitions*.

Suffix selection, first attempt

Suffix selection, first attempt

1-*st Phase Transition:* from phase 0 to phase 1

Suffix selection, first attempt

1-*st Phase Transition:* from phase 0 to phase 1

- *Phase* 0: σ_0 is void and all the suffixes are active.

Suffix selection, first attempt

1*-st Phase Transition:* from phase 0 to phase 1

- *Phase* 0: σ_0 is void and all the suffixes are active.
- *Transition*: we apply the selection algorithm in [Blum et al., JCSS 7, 1973] and find *the* k*-th smallest element* α_1 *of* T.

Suffix selection, first attempt

1*-st Phase Transition:* from phase 0 to phase 1

- *Phase* 0: σ_0 is void and all the suffixes are active.
- *Transition*: we apply the selection algorithm in [Blum et al., JCSS 7, 1973] and find *the k-th smallest element α_1 of T*.
- *Phase* 1: $\sigma_1 = \alpha_1$ and the active suffixes are the ones starting with α_1.

Suffix selection, first attempt

1-*st Phase Transition:* from phase 0 to phase 1

- *Phase* 0: σ_0 is void and all the suffixes are active.
- *Transition*: we apply the selection algorithm in [Blum et al., JCSS 7, 1973] and find *the k-th smallest element α_1 of T*.
- *Phase* 1: $\sigma_1 = \alpha_1$ and the active suffixes are the ones starting with α_1.

$(t+1)$-*th Phase Transition:* from phase t to phase $t+1$

Suffix selection, first attempt

1-*st Phase Transition:* from phase 0 to phase 1

- *Phase* 0: σ_0 is void and all the suffixes are active.
- *Transition*: we apply the selection algorithm in [Blum et al., JCSS 7, 1973] and find *the k-th smallest element α_1 of T*.
- *Phase* 1: $\sigma_1 = \alpha_1$ and the active suffixes are the ones starting with α_1.

$(t+1)$-*th Phase Transition:* from phase t to phase $t+1$

- Let's consider the *multiset* $\mathcal{D}_t = \{T_i[t+1] | T_i \in \mathcal{A}_t\}$.

Suffix selection, first attempt

1*-st Phase Transition:* from phase 0 to phase 1

- *Phase* 0: σ_0 is void and all the suffixes are active.
- *Transition*: we apply the selection algorithm in [Blum et al., JCSS 7, 1973] and find *the k-th smallest element α_1 of T*.
- *Phase* 1: $\sigma_1 = \alpha_1$ and the active suffixes are the ones starting with α_1.

$(t+1)$*-th Phase Transition:* from phase t to phase $t+1$

- Let's consider the *multiset* $\mathcal{D}_t = \{T_i[t+1] | T_i \in \mathcal{A}_t\}$.
- Using [Blum et al. 1973], we select from $\mathcal{D}_t$ the *$(k - l_t)$-th smallest element* α_{t+1}.

Suffix selection, first attempt

1-st Phase Transition: from phase 0 to phase 1

- *Phase* 0: σ_0 is void and all the suffixes are active.
- *Transition*: we apply the selection algorithm in [Blum et al., JCSS 7, 1973] and find *the k-th smallest element α_1 of T*.
- *Phase* 1: $\sigma_1 = \alpha_1$ and the active suffixes are the ones starting with α_1.

$(t+1)$-th Phase Transition: from phase t to phase $t+1$

- Let's consider the *multiset* $\mathcal{D}_t = \{T_i[t+1] | T_i \in \mathcal{A}_t\}$.
- Using [Blum et al. 1973], we select from $\mathcal{D}_t$ the $(k - l_t)$*-th smallest element* α_{t+1}.
- We set $\sigma_{t+1} = \sigma_t \alpha_{t+1}$.

Suffix selection, first attempt

1*-st Phase Transition:* from phase 0 to phase 1

- *Phase* 0: σ_0 is void and all the suffixes are active.
- *Transition*: we apply the selection algorithm in [Blum et al., JCSS 7, 1973] and find *the k-th smallest element α_1 of T*.
- *Phase* 1: $\sigma_1 = \alpha_1$ and the active suffixes are the ones starting with α_1.

$(t+1)$*-th Phase Transition:* from phase t to phase $t+1$

- Let's consider the *multiset* $\mathcal{D}_t = \{T_i[t+1] | T_i \in \mathcal{A}_t\}$.
- Using [Blum et al. 1973], we select from $\mathcal{D}_t$ the $(k - l_t)$*-th smallest element* α_{t+1}.
- We set $\sigma_{t+1} = \sigma_t \alpha_{t+1}$.
- $\mathcal{A}_{t+1}$ contains all the suffixes in $\mathcal{A}_t$ having α_{t+1} *as their* $(t+1)$*-th element*.

Suffix selection, first attempt

1*-st Phase Transition:* from phase 0 to phase 1

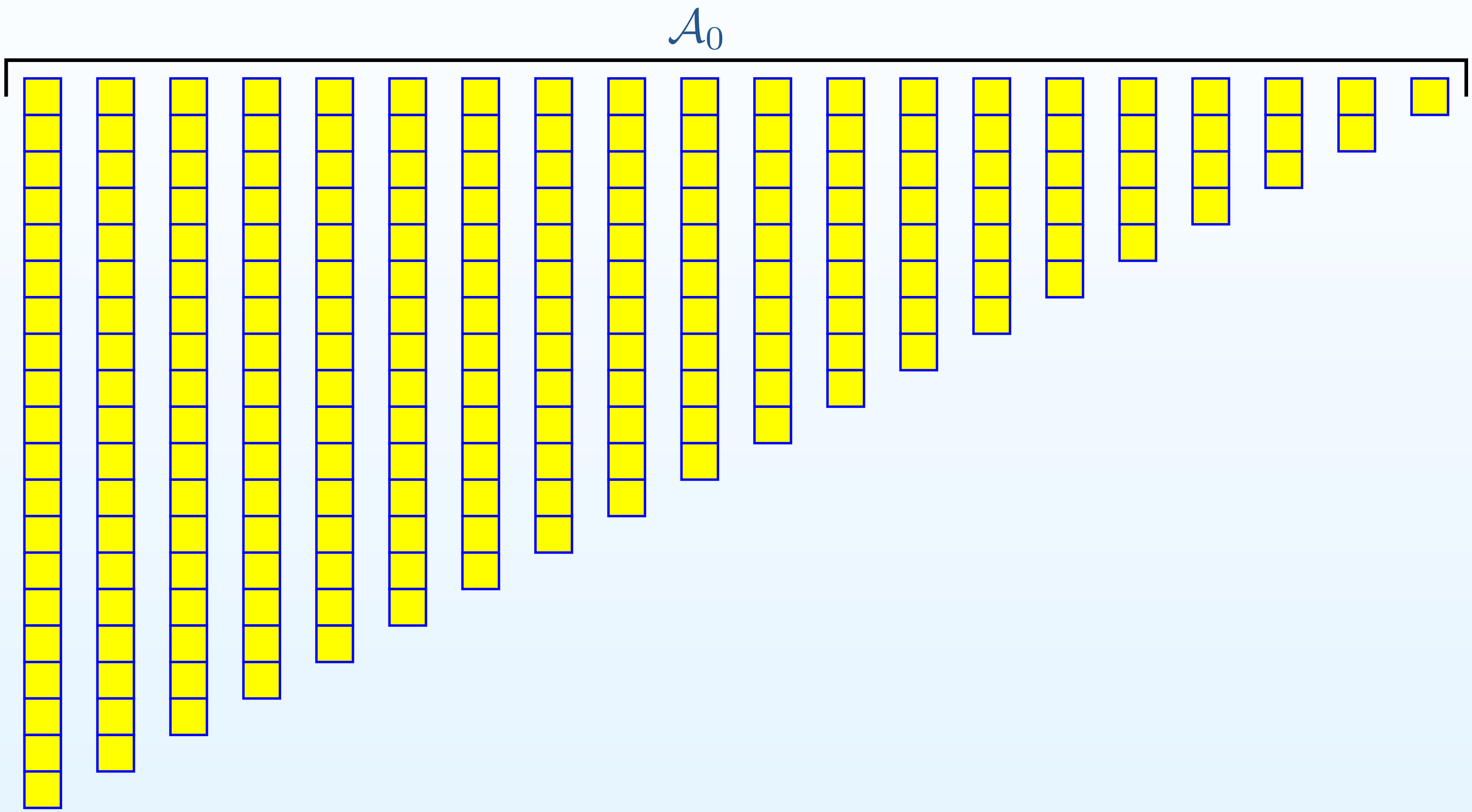
- *Phase* 0: σ_0 is void and all the suffixes are active.
- *Transition*: we apply the selection algorithm in [Blum et al., JCSS 7, 1973] and find *the k-th smallest element α_1 of T*.
- *Phase* 1: $\sigma_1 = \alpha_1$ and the active suffixes are the ones starting with α_1.

$(t+1)$*-th Phase Transition:* from phase t to phase $t+1$

- Let's consider the *multiset* $\mathcal{D}_t = \{T_i[t+1] | T_i \in \mathcal{A}_t\}$.
- Using [Blum et al. 1973], we select from $\mathcal{D}_t$ the $(k - l_t)$*-th smallest element* α_{t+1}.
- We set $\sigma_{t+1} = \sigma_t \alpha_{t+1}$.
- $\mathcal{A}_{t+1}$ contains all the suffixes in $\mathcal{A}_t$ having α_{t+1} *as their* $(t+1)$*-th element*.

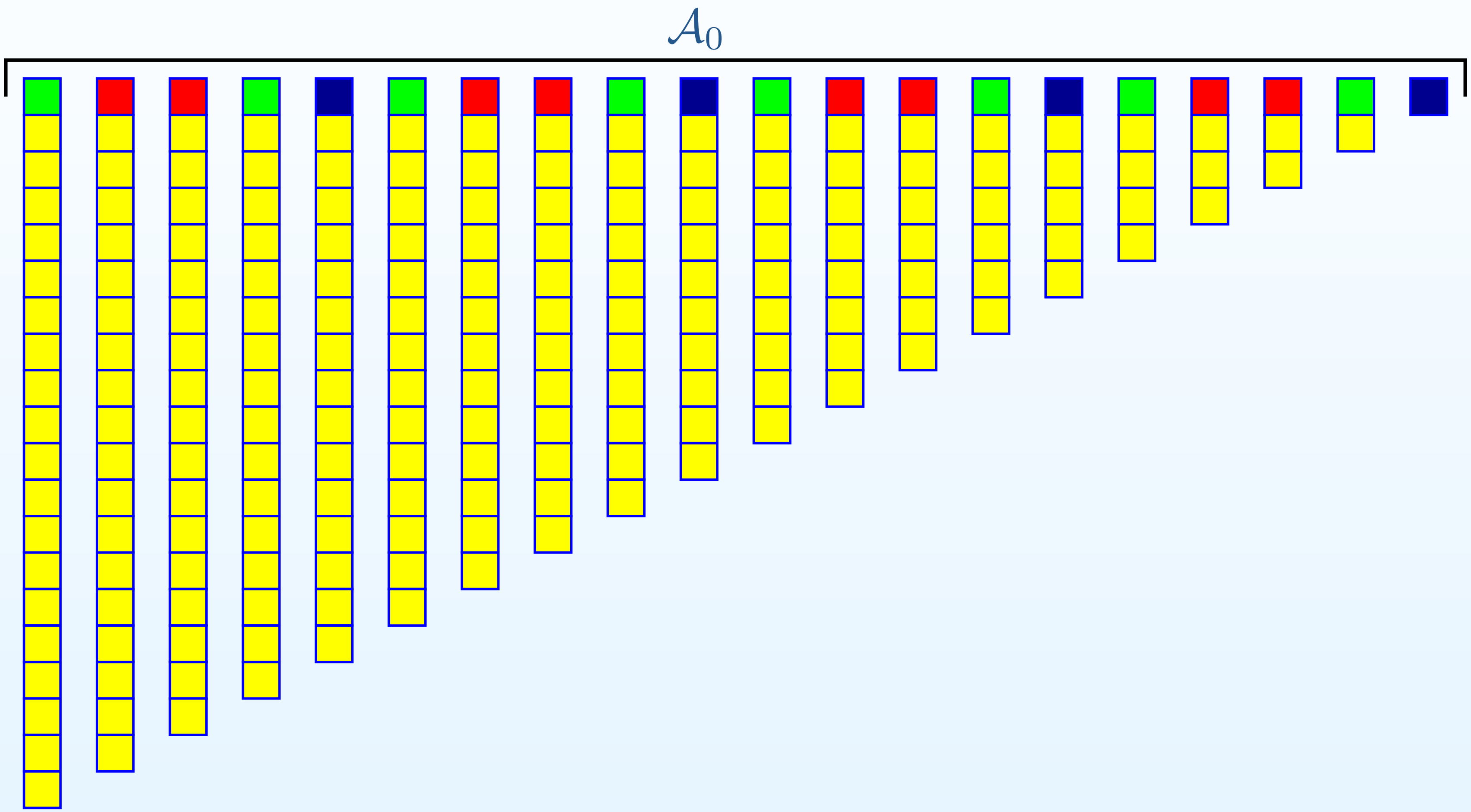
The computation ends when a phase transition leaves us *with only one active suffix*.

Suffix selection, first attempt



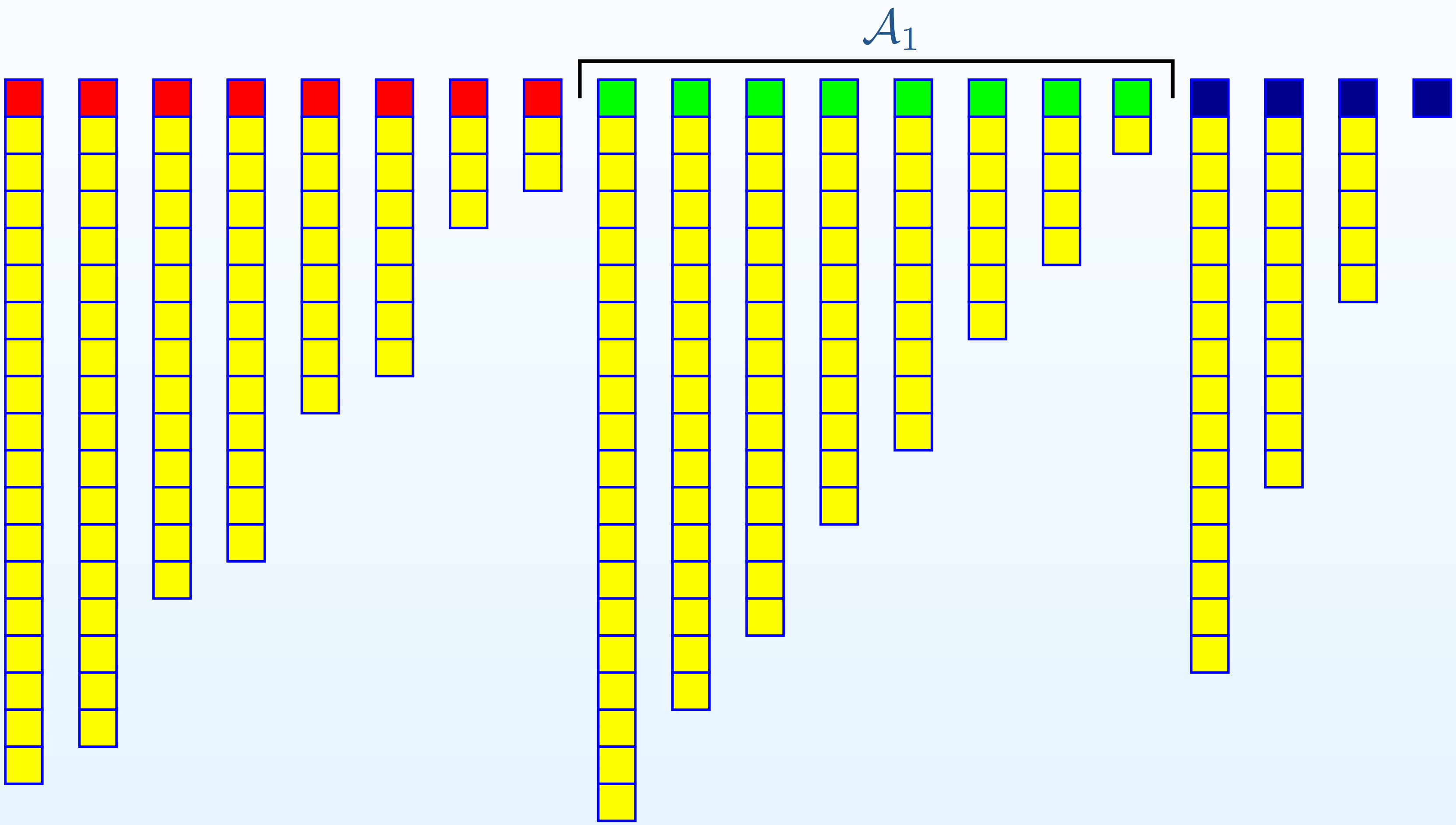
$k = 10$, $l_0 = 0$, Phase 0

Suffix selection, first attempt



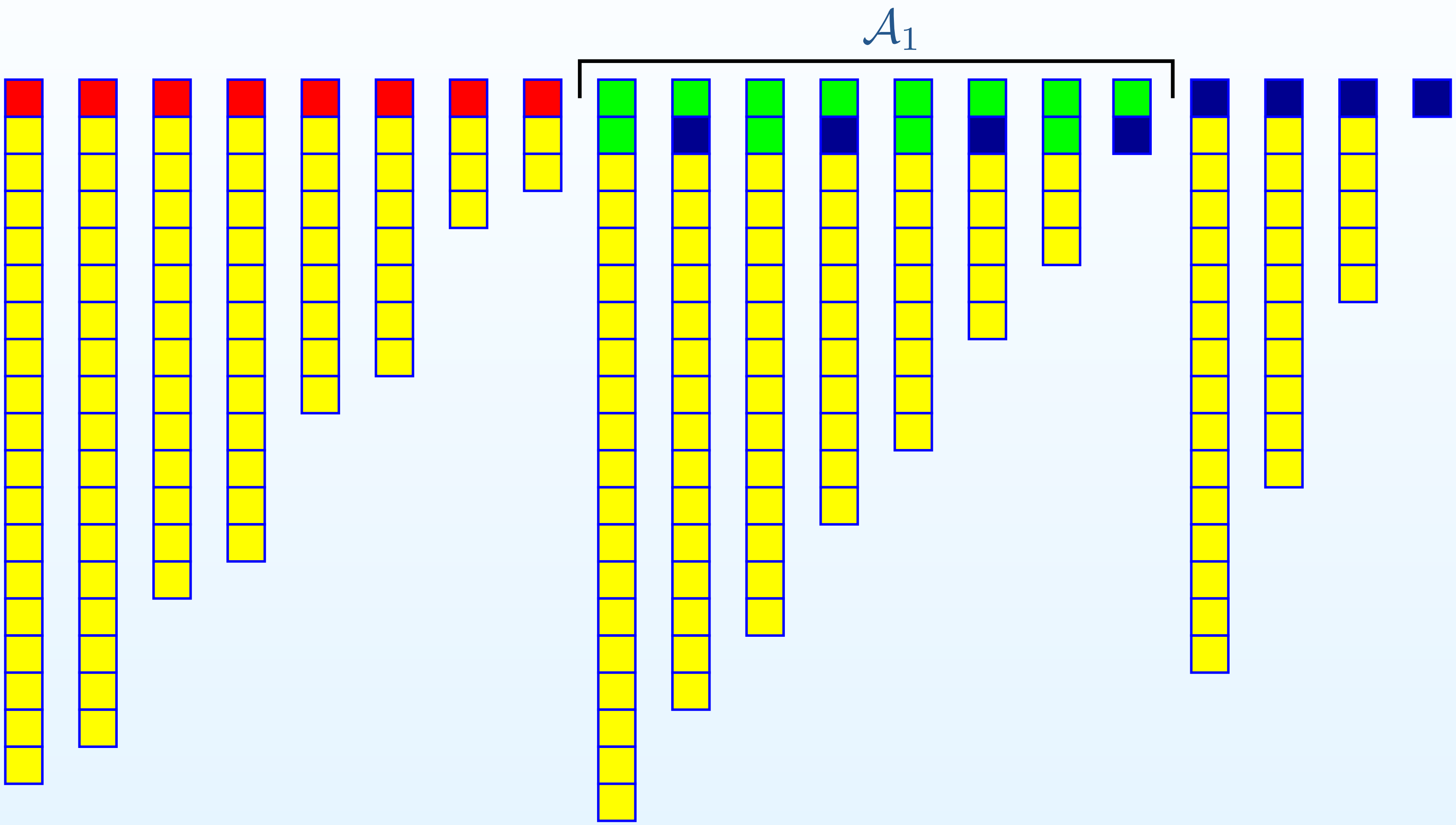
$k = 10$, $l_0 = 0$, Phase 0

Suffix selection, first attempt



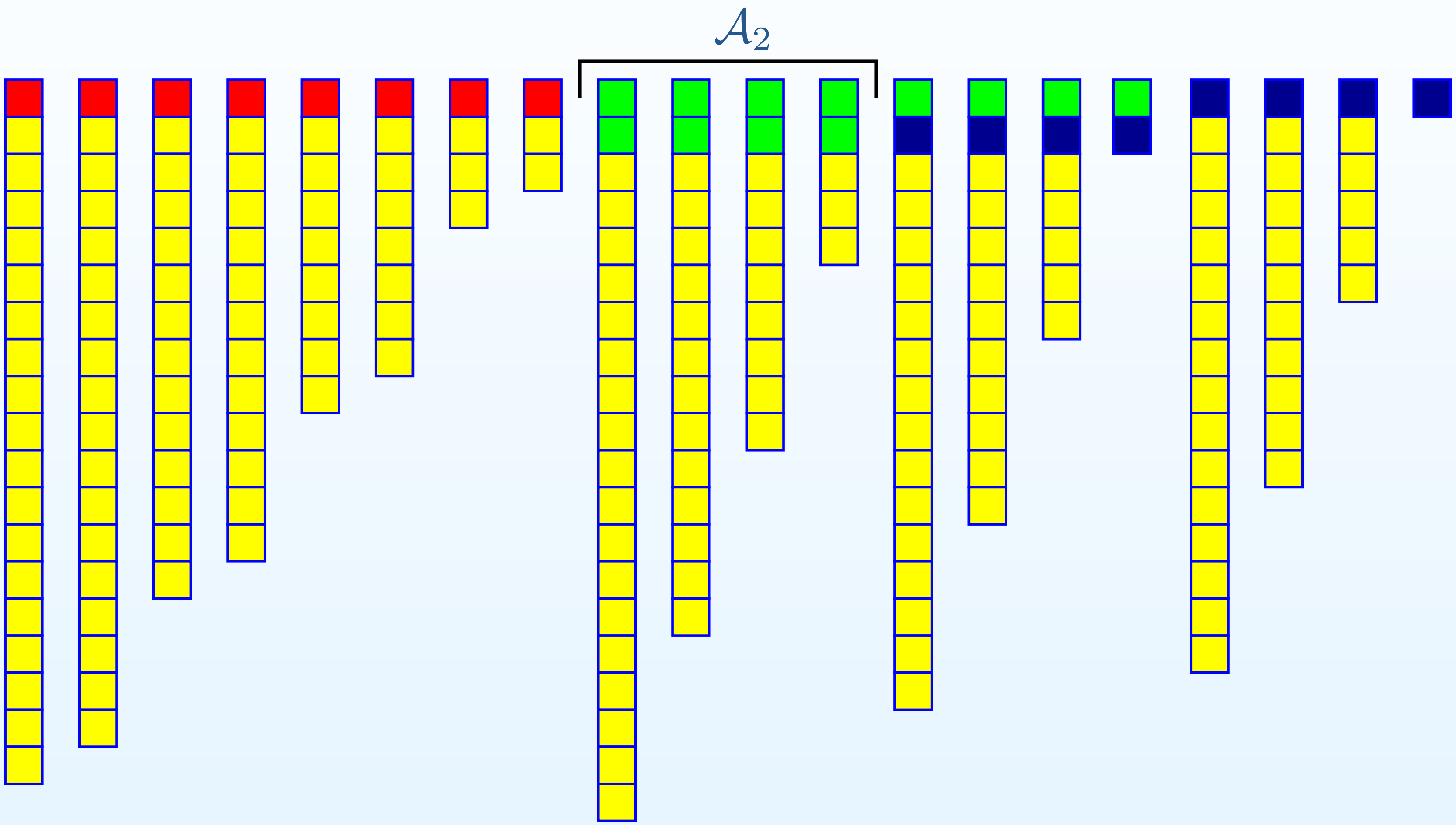
$k = 10$, $l_1 = 8$, Phase 1

Suffix selection, first attempt



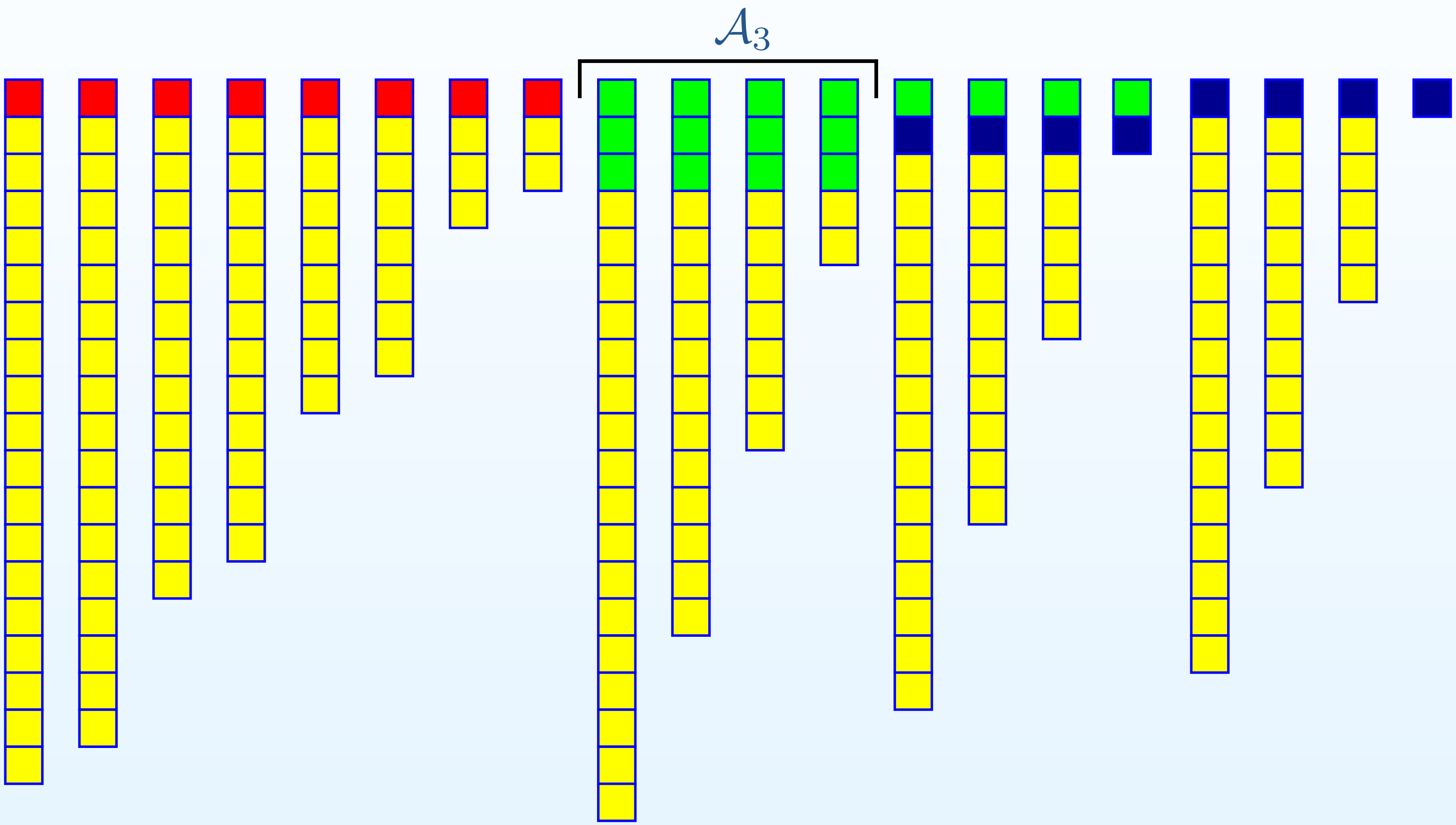
$k = 10$, $l_1 = 8$, Phase 1

Suffix selection, first attempt



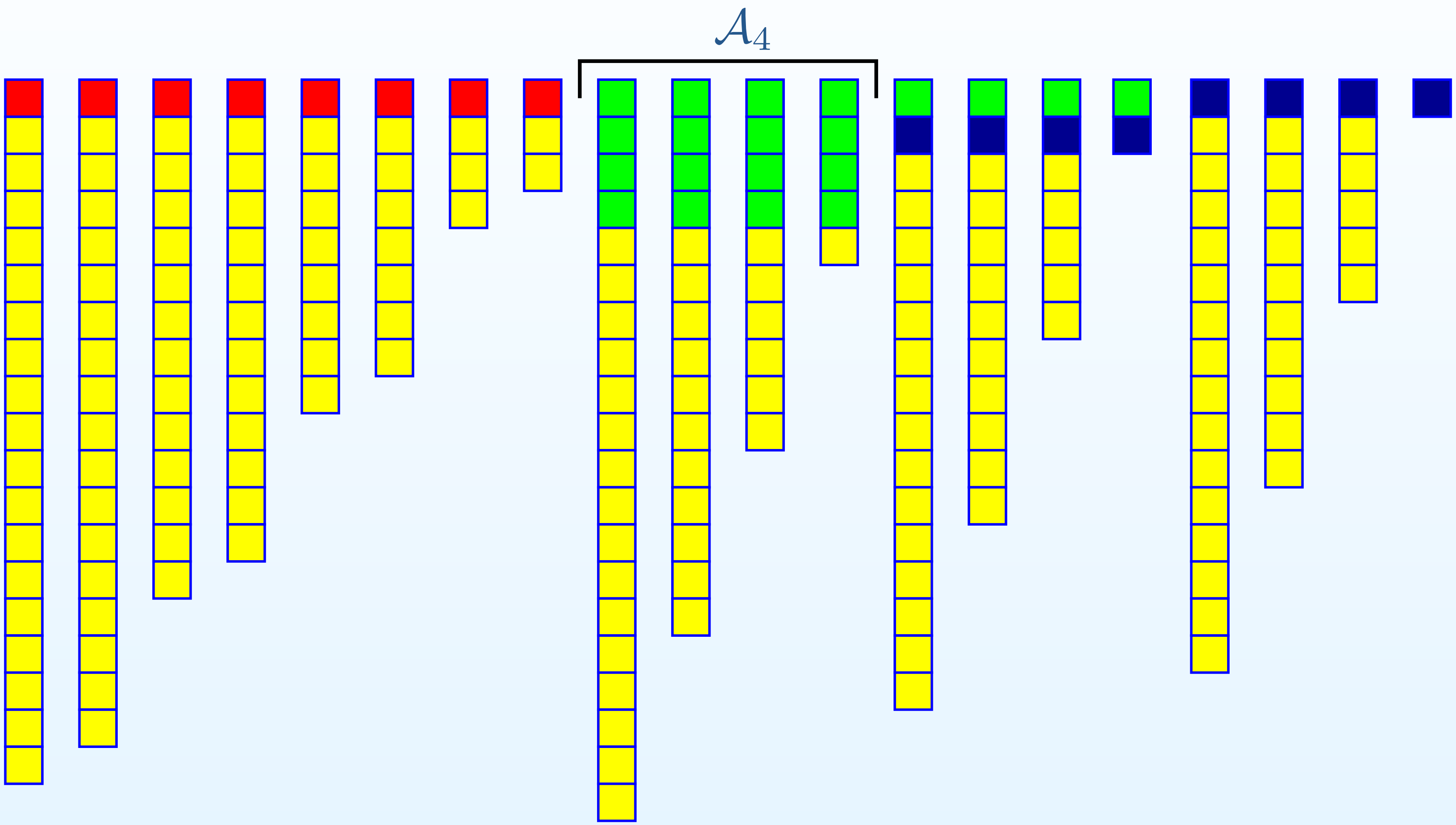
$k = 10$, $l_2 = 8$, Phase 2

Suffix selection, first attempt



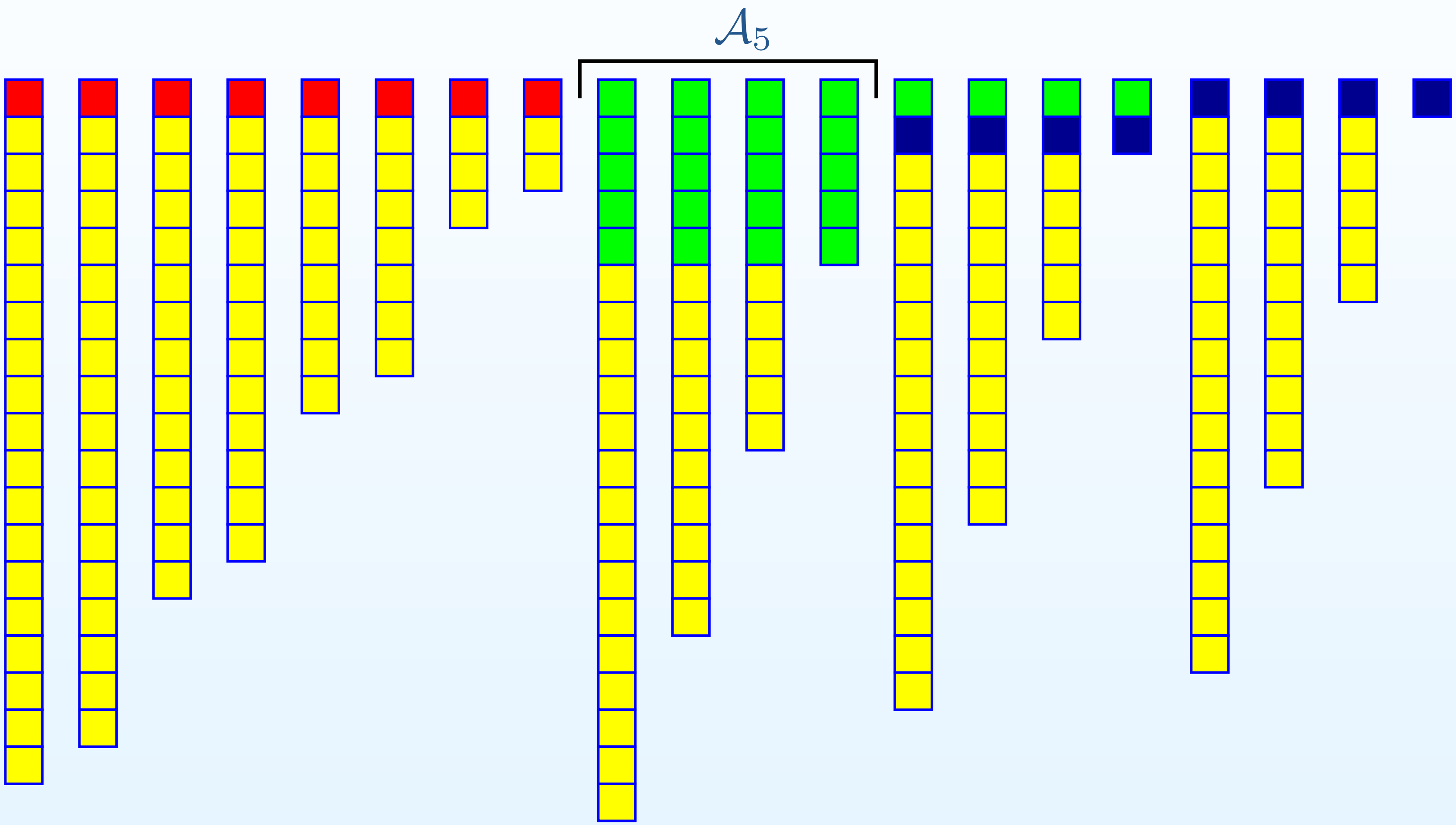
$k = 10$, $l_3 = 8$, Phase 3

Suffix selection, first attempt



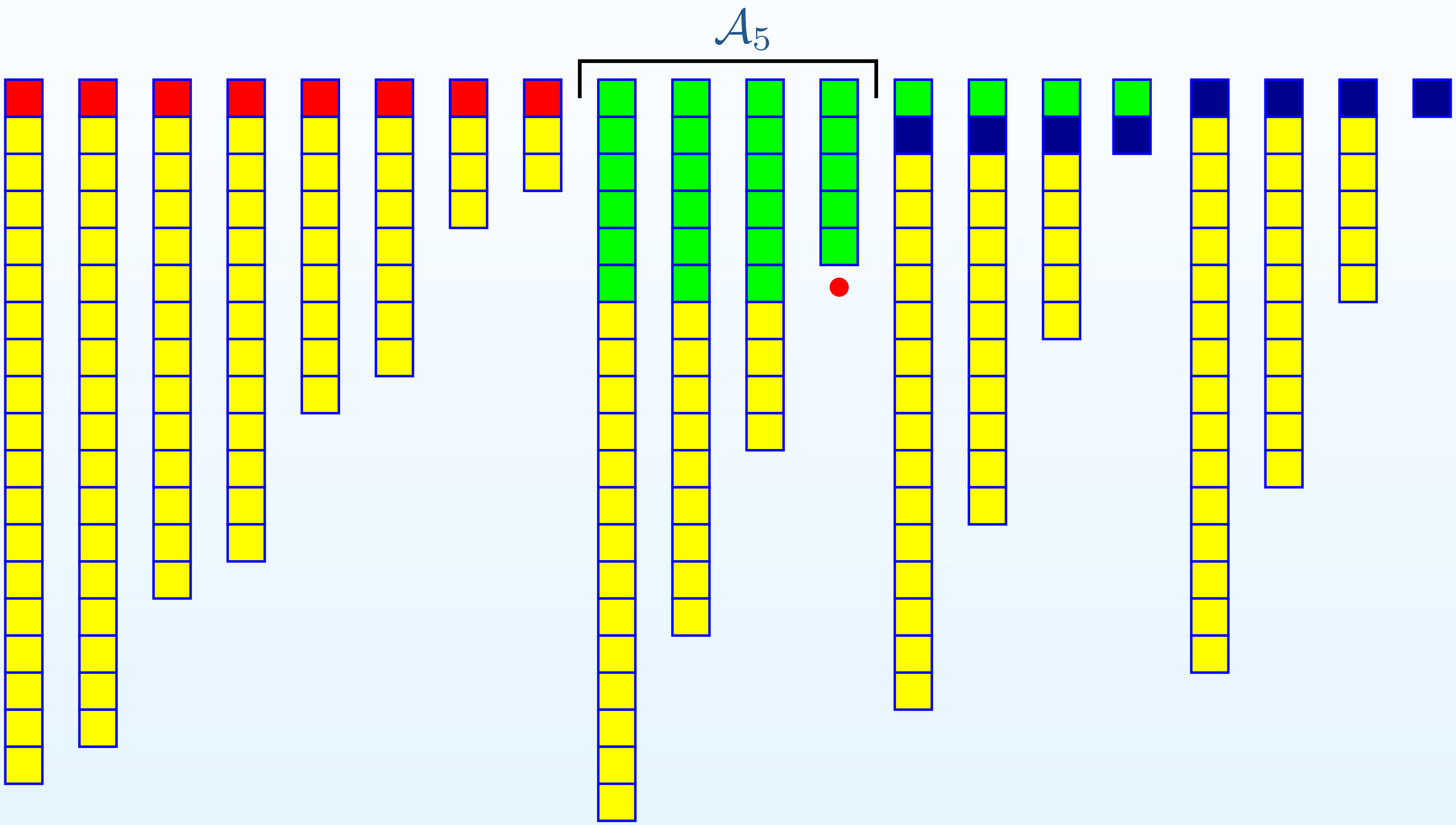
$k = 10$, $l_4 = 8$, Phase 4

Suffix selection, first attempt



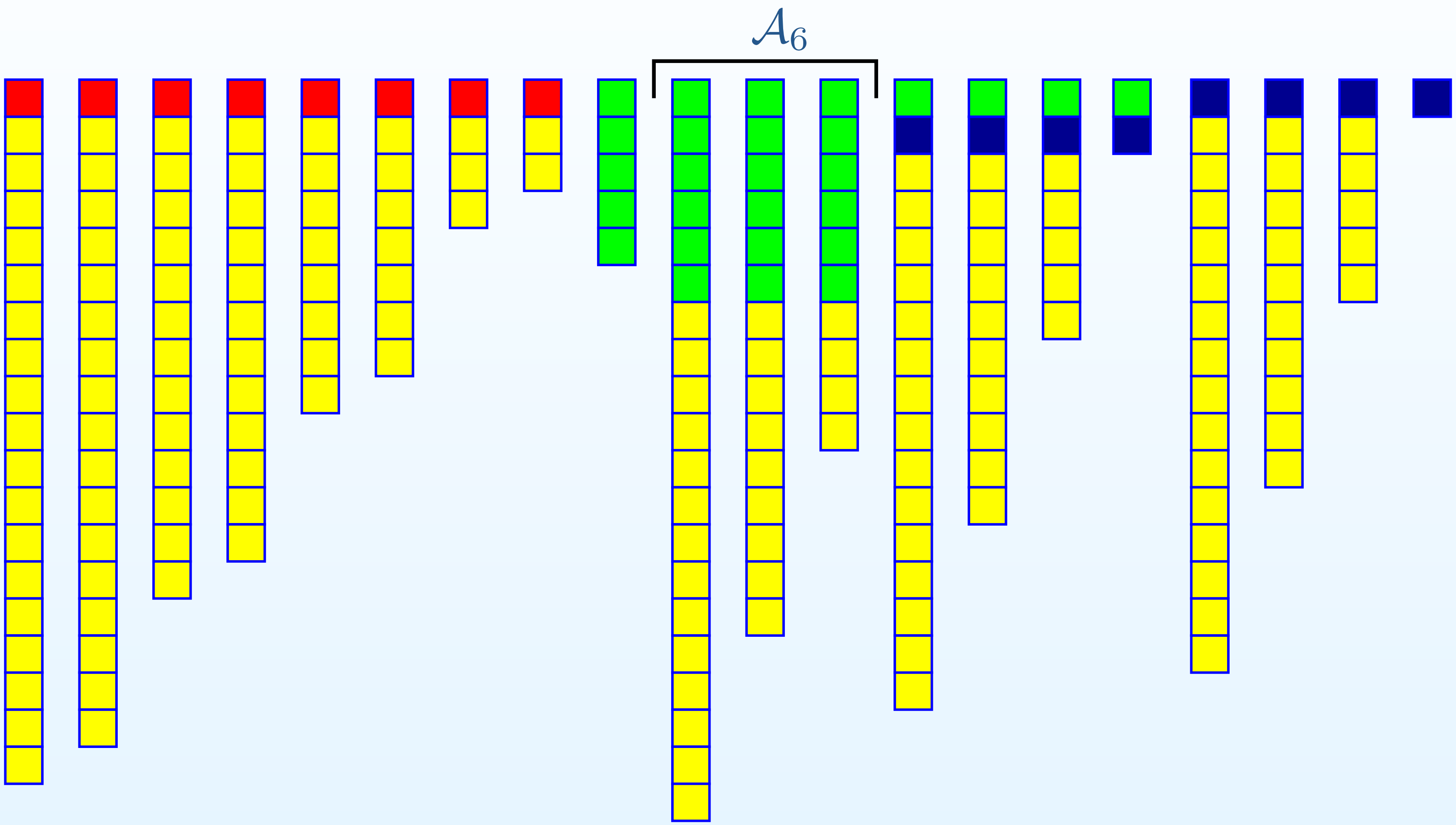
$k = 10$, $l_5 = 8$, Phase 5

Suffix selection, first attempt



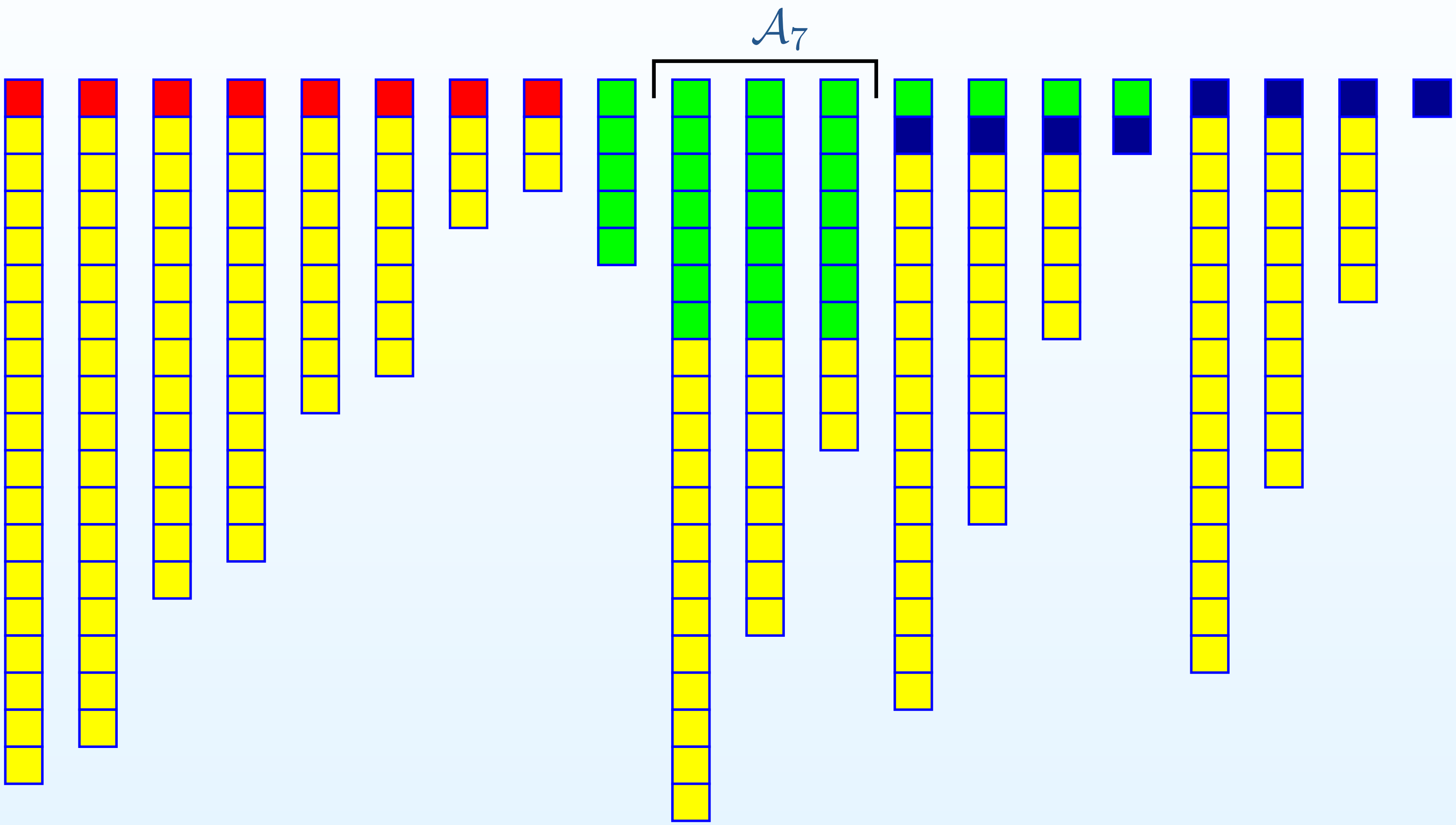
$k = 10$, $l_5 = 8$, Phase 5

Suffix selection, first attempt



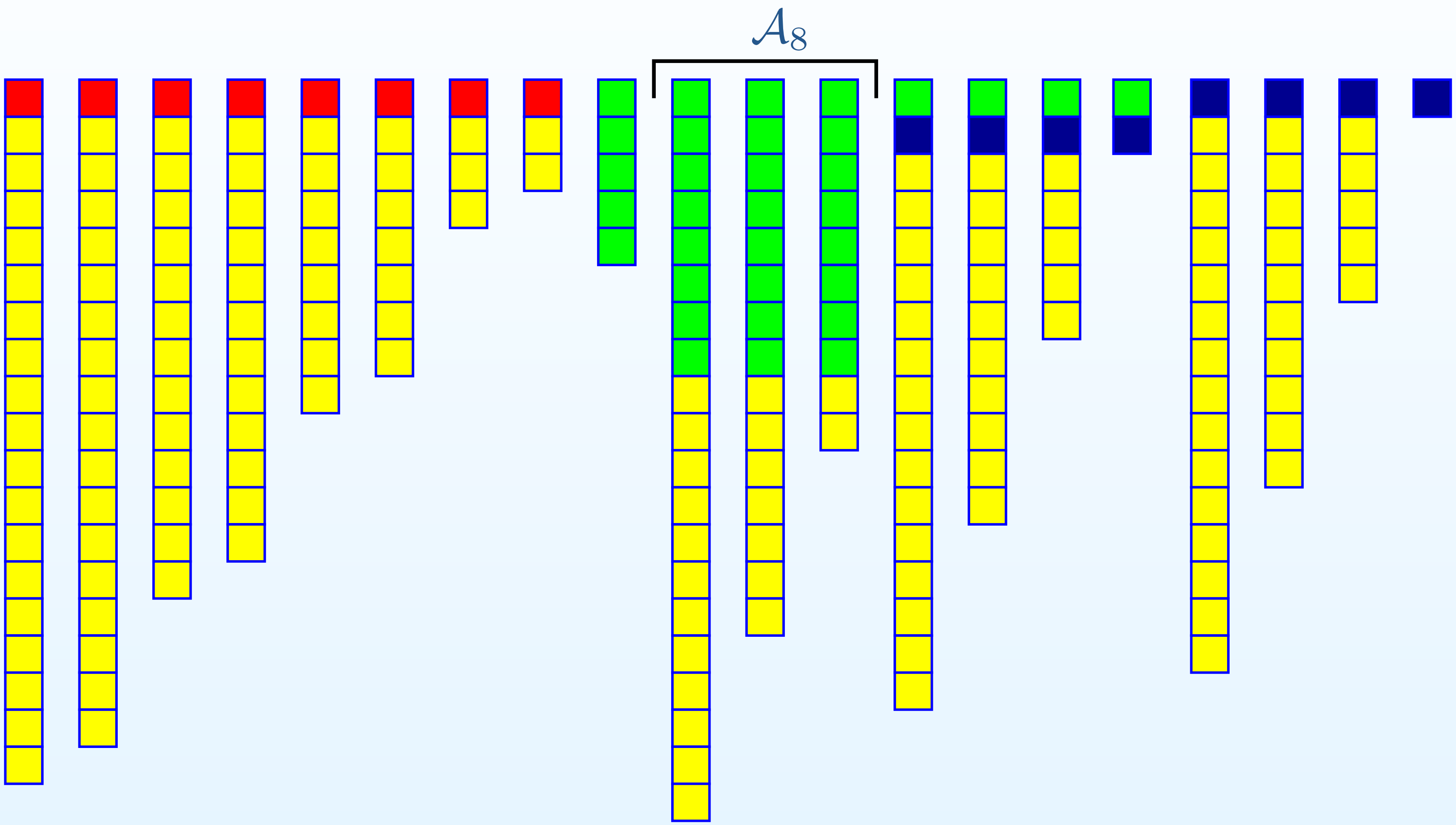
$k = 10$, $l_6 = 9$, Phase 6

Suffix selection, first attempt



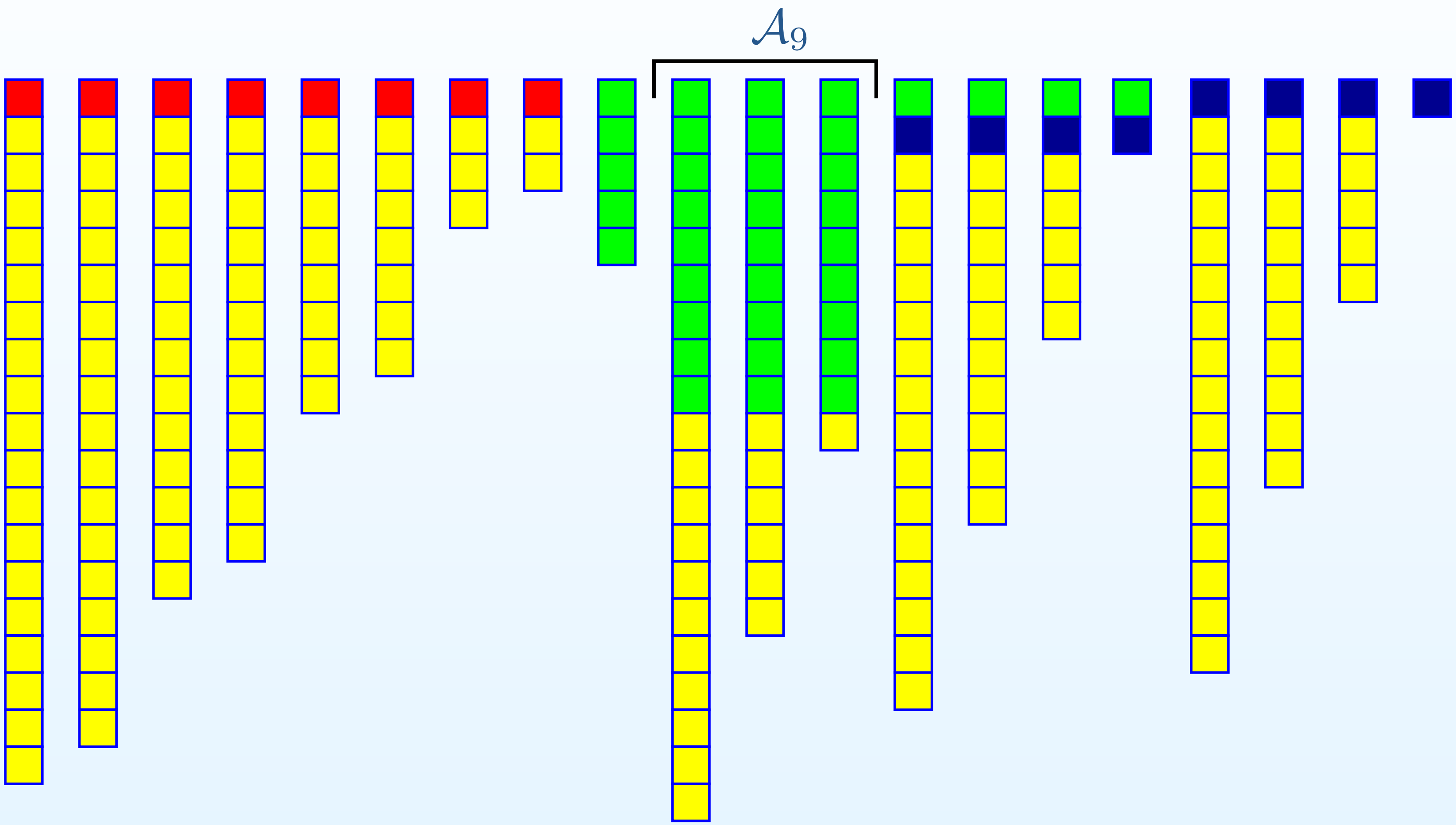
$k = 10$, $l_7 = 9$, Phase 7

Suffix selection, first attempt



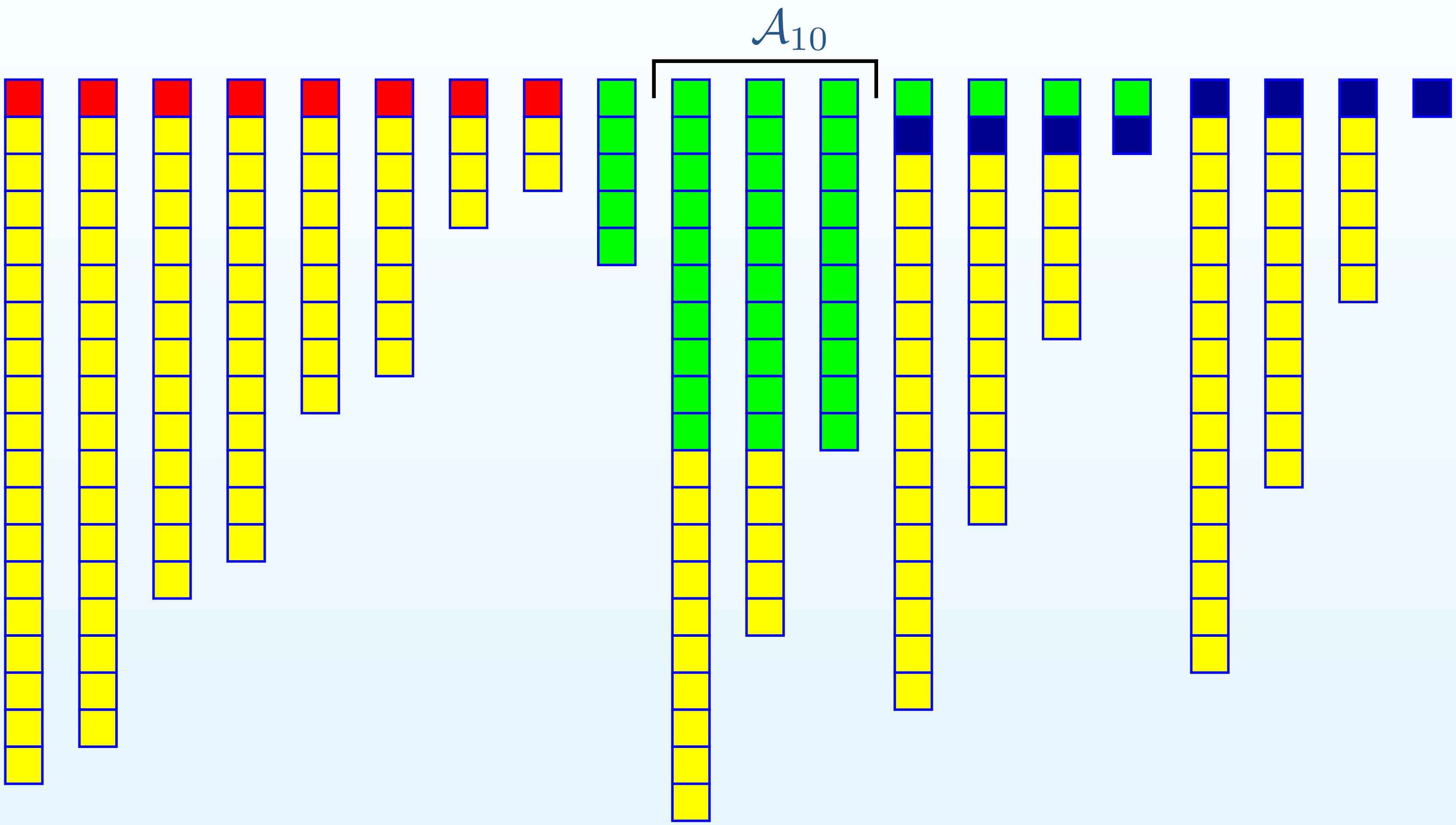
$k = 10$, $l_8 = 9$, Phase 8

Suffix selection, first attempt



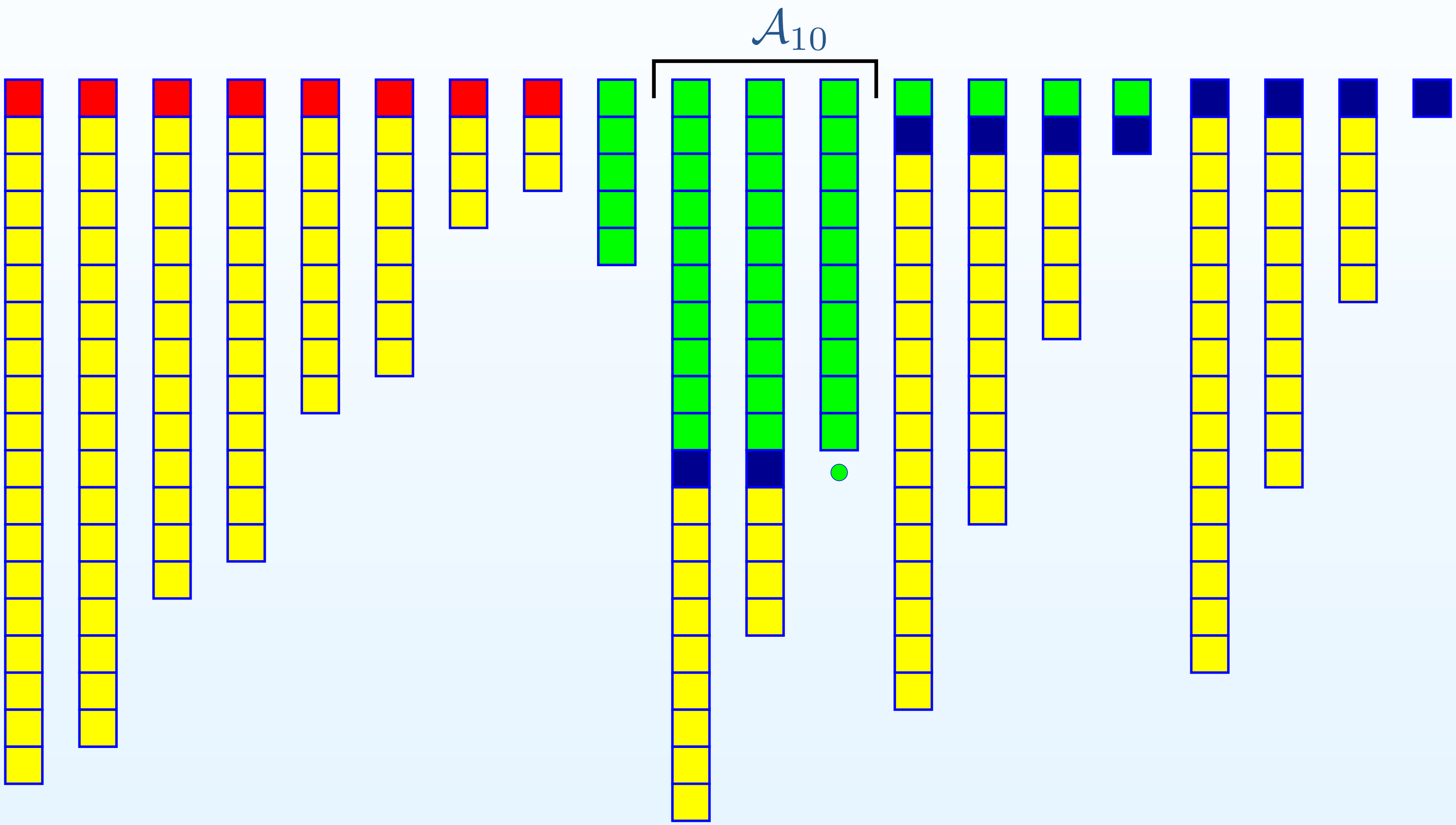
$k = 10$, $l_9 = 9$, Phase 9

Suffix selection, first attempt



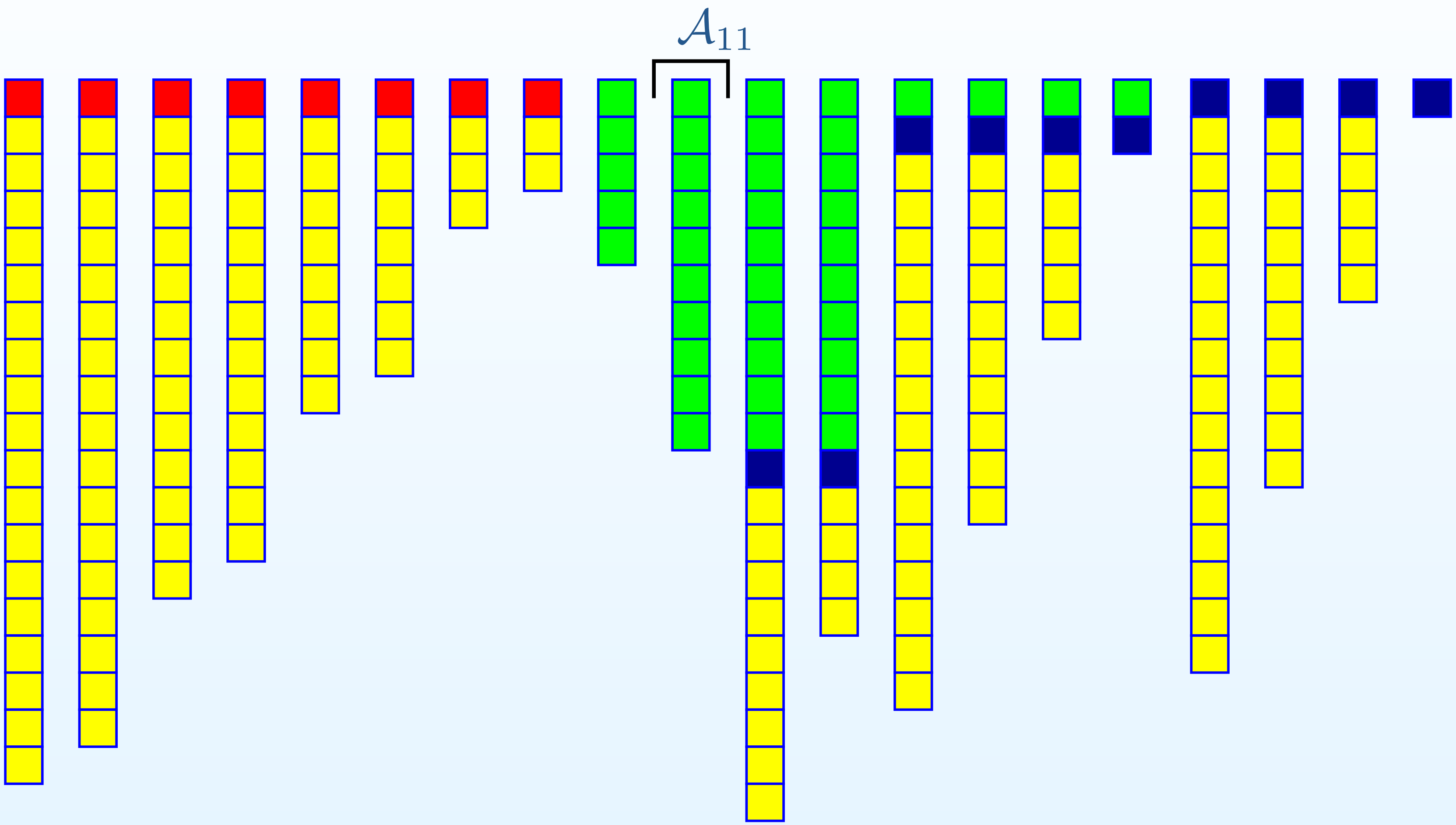
$k = 10$, $l_{10} = 9$, Phase 10

Suffix selection, first attempt



$k = 10$, $l_{10} = 9$, Phase 10

Suffix selection, first attempt



$k = 10$, $l_{11} = 9$, Phase 11

Suffix selection, first attempt

Suffix selection, first attempt

Clearly, this simple approach is not optimal:

Suffix selection, first attempt

Clearly, this simple approach is not optimal:

$\boxed{\textit{it takes } O\left(n^2\right) \textit{ time in the worst case.}}$

Suffix selection, first attempt

Clearly, this simple approach is not optimal:

> *it takes* $O\left(n^2\right)$ *time in the worst case.*

- We have not exploited the basic fact that *suffixes overlaps*.

Suffix selection, first attempt

Clearly, this simple approach is not optimal:

it takes $O(n^2)$ *time in the worst case.*

- We have not exploited the basic fact that *suffixes overlaps*.
- The elements of T are unnecessarily accessed *multiple times*.

Suffix selection, first attempt

Clearly, this simple approach is not optimal:

$\boxed{\textit{it takes } O\left(n^2\right) \textit{ time in the worst case.}}$

- We have not exploited the basic fact that *suffixes overlaps*.
- The elements of T are unnecessarily accessed *multiple times*.
- The phase-based approach can be improved in two ways:

Suffix selection, first attempt

Clearly, this simple approach is not optimal:

$$\boxed{\textit{it takes } O\left(n^2\right) \textit{ time in the worst case.}}$$

- We have not exploited the basic fact that *suffixes overlaps*.
- The elements of T are unnecessarily accessed *multiple times*.
- The phase-based approach can be improved in two ways:
  - *By exploiting collisions of active suffixes.*

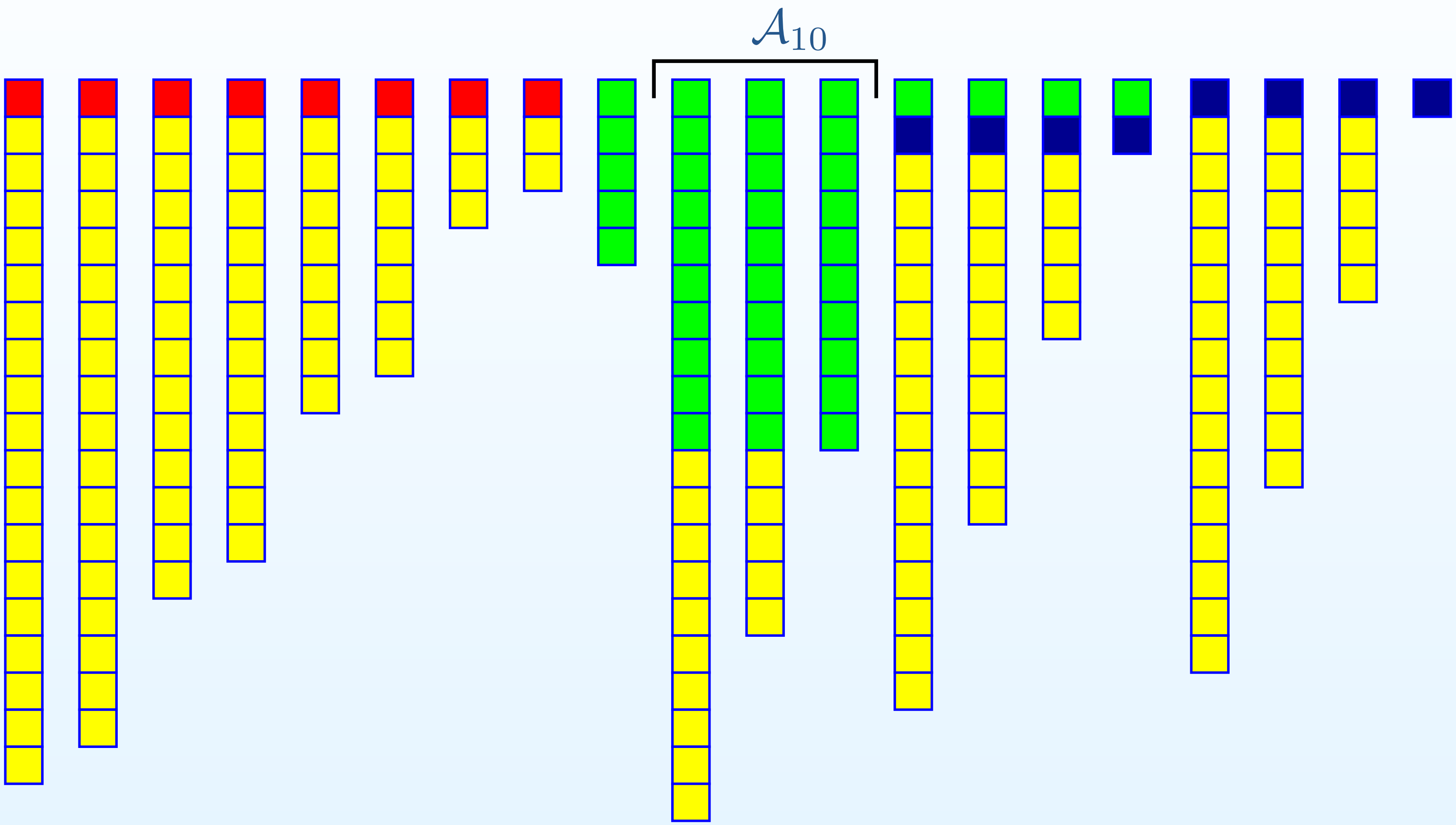
Suffix selection, first attempt

Clearly, this simple approach is not optimal:

$$\boxed{\textit{it takes } O\left(n^2\right) \textit{ time in the worst case.}}$$

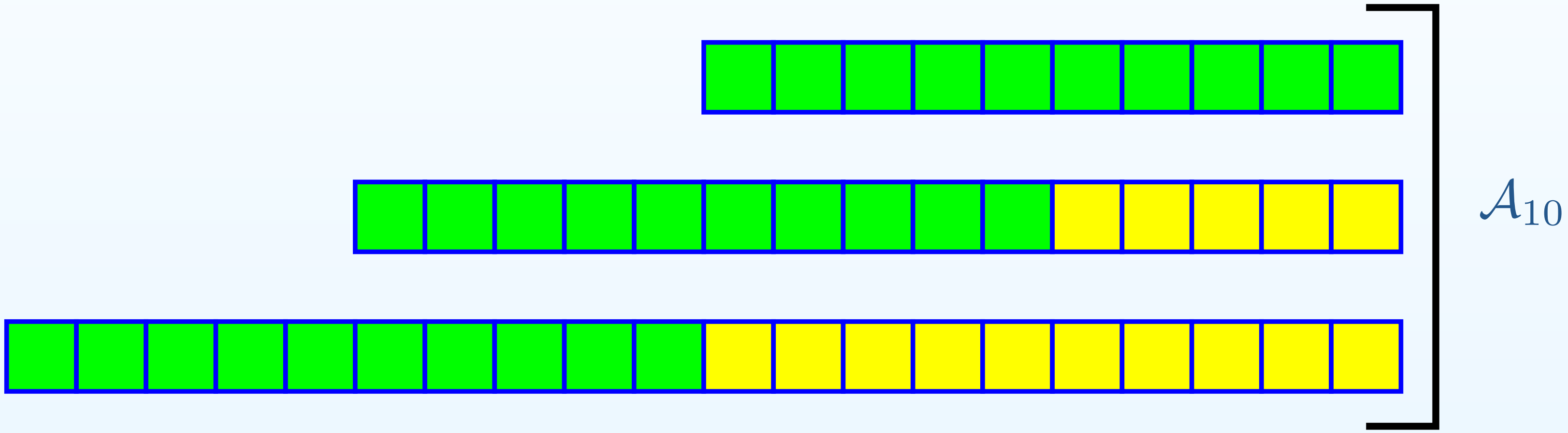
- We have not exploited the basic fact that *suffixes overlaps*.
- The elements of T are unnecessarily accessed *multiple times*.
- The phase-based approach can be improved in two ways:
  - *By exploiting collisions of active suffixes.*
  - *By reusing the work done on inactive suffixes.*

Suffix selection: collisions of active suffixes

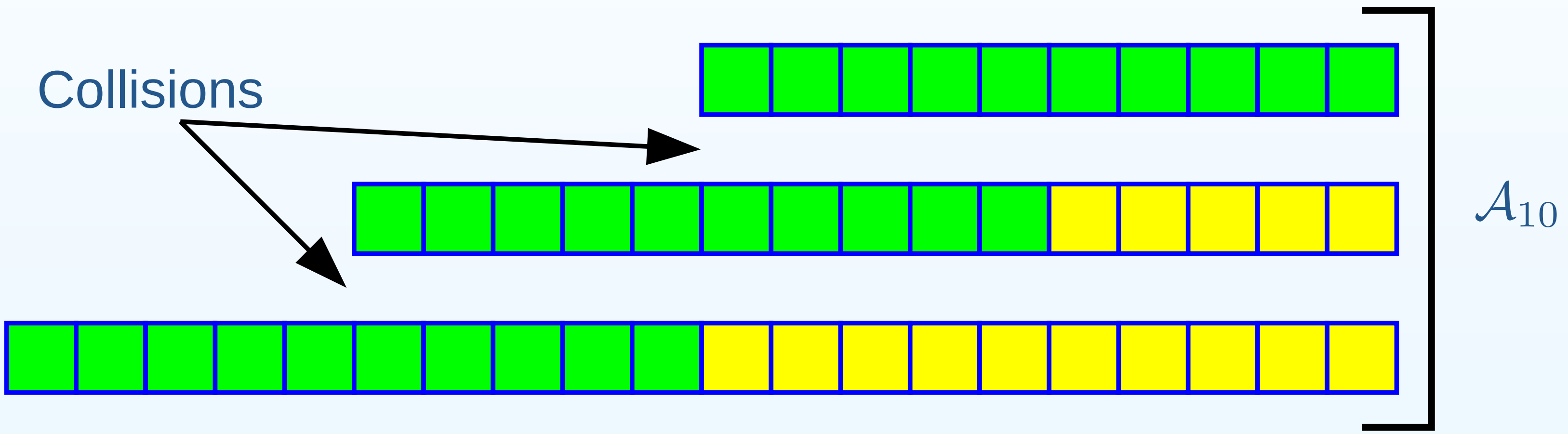


$k = 10$, $l_{10} = 9$, Phase 10

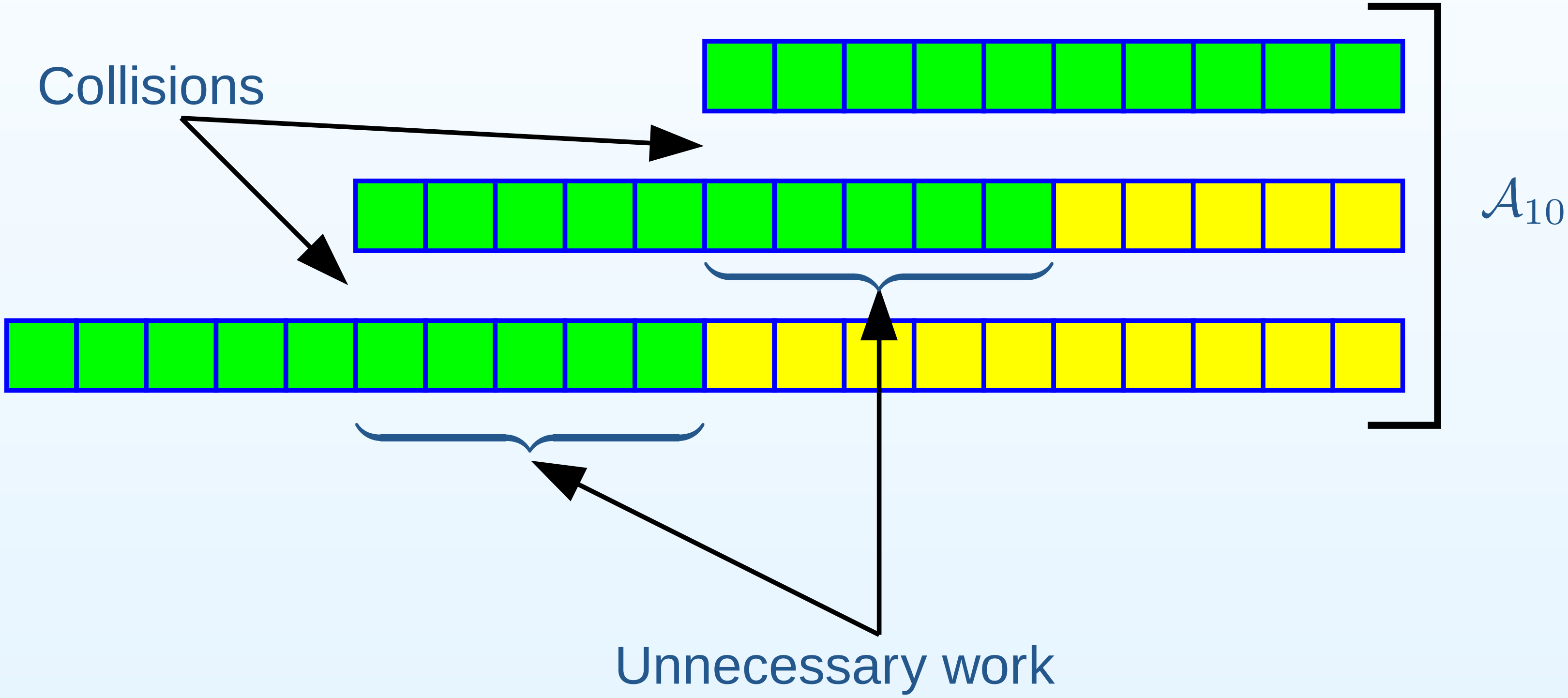
Suffix selection: collisions of active suffixes



Suffix selection: collisions of active suffixes



Suffix selection: collisions of active suffixes



Suffix selection, second attempt: exploiting collisions

Some terminology:

Suffix selection, second attempt: exploiting collisions

Some terminology:

For any phase t,

Suffix selection, second attempt: exploiting collisions

Some terminology:

For any phase t,

- The *extent* of a suffix T_i (active or inactive) is the *longest common prefix with* σ_t.

Suffix selection, second attempt: exploiting collisions

Some terminology:

For any phase t,

- The *extent* of a suffix T_i (active or inactive) is the *longest common prefix with* σ_t.
- Two suffixes T_i, T_j *collide* when their *extents are either adjacent* (i.e. the last element of the extent of T_i is adjacent to the first element of the extent of T_j or vice versa) *or overlapping*.

Suffix selection, second attempt: exploiting collisions

Some terminology:

For any phase t,

- The *extent* of a suffix T_i (active or inactive) is the *longest common prefix with* σ_t.
- Two suffixes T_i, T_j *collide* when their *extents are either adjacent* (i.e. the last element of the extent of T_i is adjacent to the first element of the extent of T_j or vice versa) *or overlapping*.

In the first attempt, with any phase transition

Suffix selection, second attempt: exploiting collisions

Some terminology:

For any phase t,

- The *extent* of a suffix T_i (active or inactive) is the *longest common prefix with* σ_t.
- Two suffixes T_i, T_j *collide* when their *extents are either adjacent* (i.e. the last element of the extent of T_i is adjacent to the first element of the extent of T_j or vice versa) *or overlapping*.

In the first attempt, with any phase transition

- we tried to enlarge *by just one element* the extent of each active suffix

Suffix selection, second attempt: exploiting collisions

Some terminology:

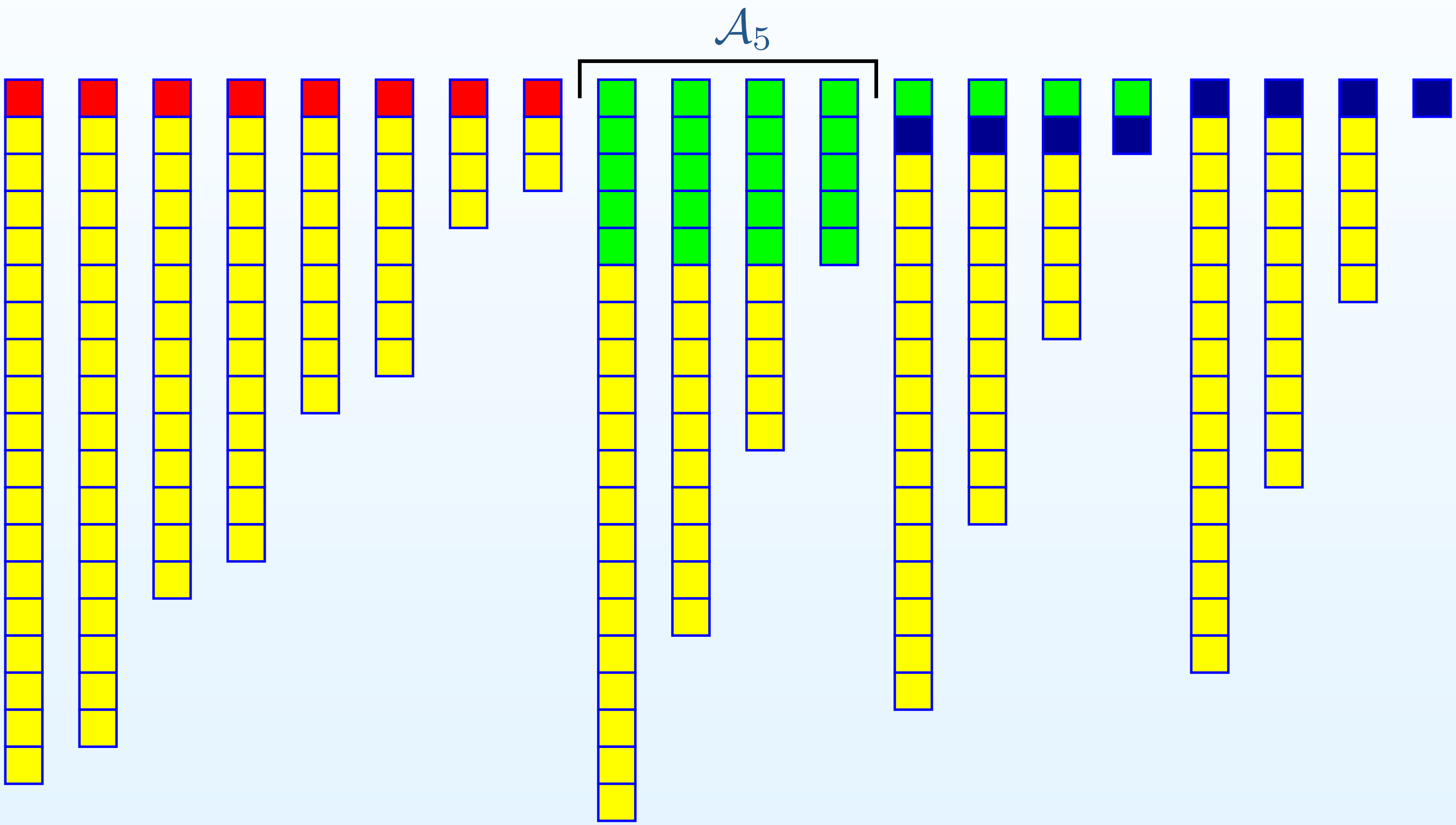
For any phase t,

- The *extent* of a suffix T_i (active or inactive) is the *longest common prefix with* σ_t.
- Two suffixes T_i, T_j *collide* when their *extents are either adjacent* (i.e. the last element of the extent of T_i is adjacent to the first element of the extent of T_j or vice versa) *or overlapping*.

In the first attempt, with any phase transition

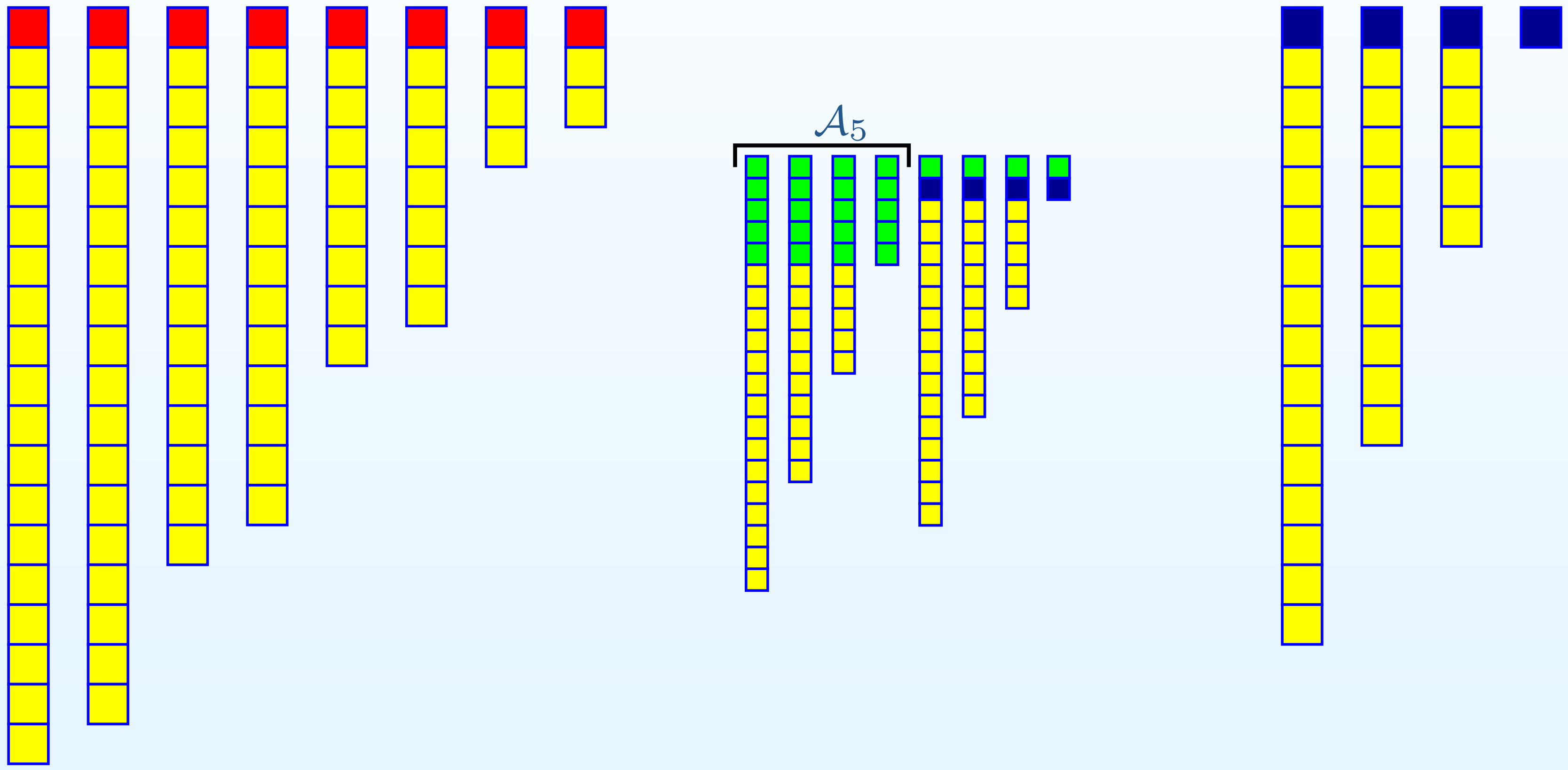
- we tried to enlarge *by just one element* the extent of each active suffix
- while completely *ignoring the emerging of collisions*.

Suffix selection, second attempt: exploiting collisions



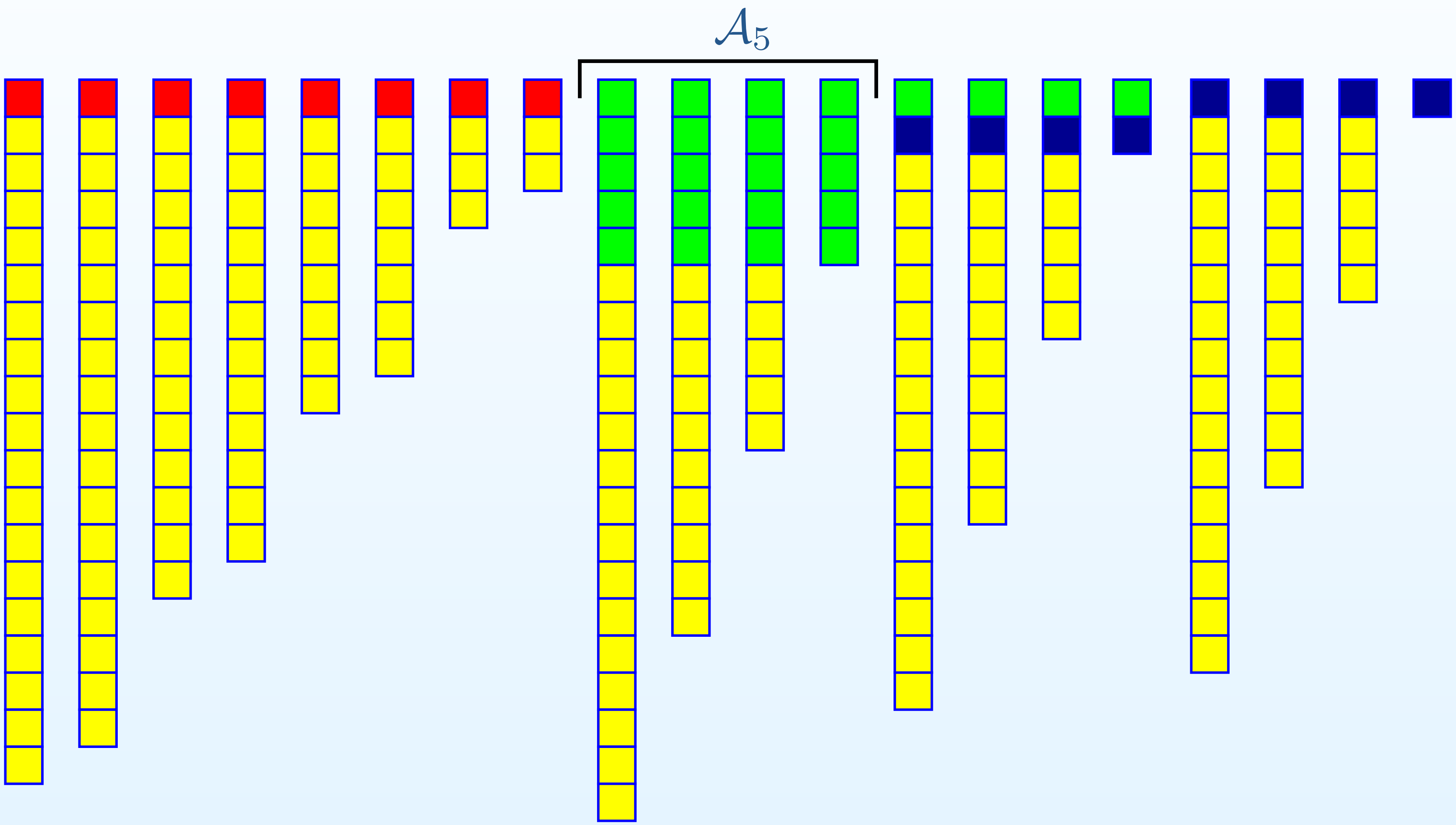
$k = 10$, $l_5 = 8$, Phase 5

Suffix selection, second attempt: exploiting collisions



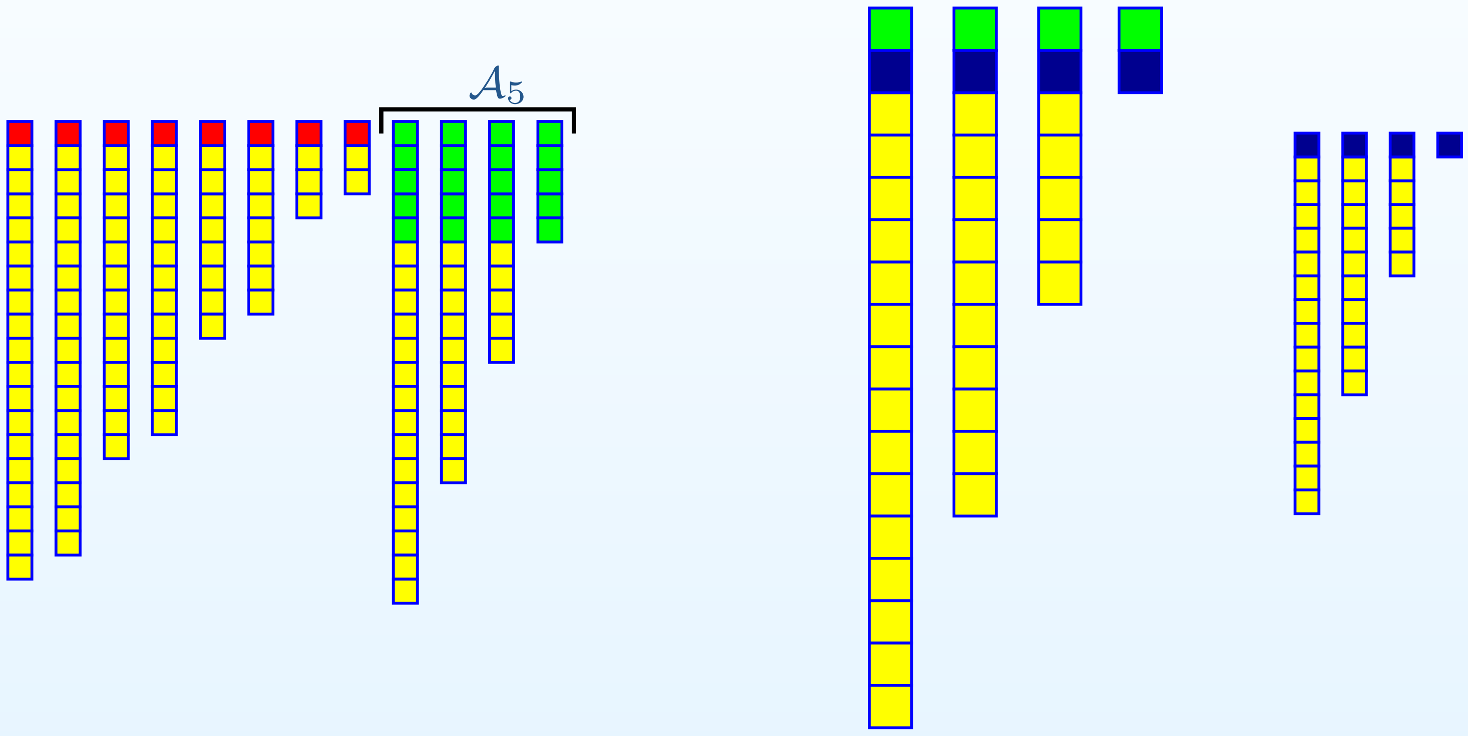
$k = 10$, $l_5 = 8$, Phase 5

Suffix selection, second attempt: exploiting collisions



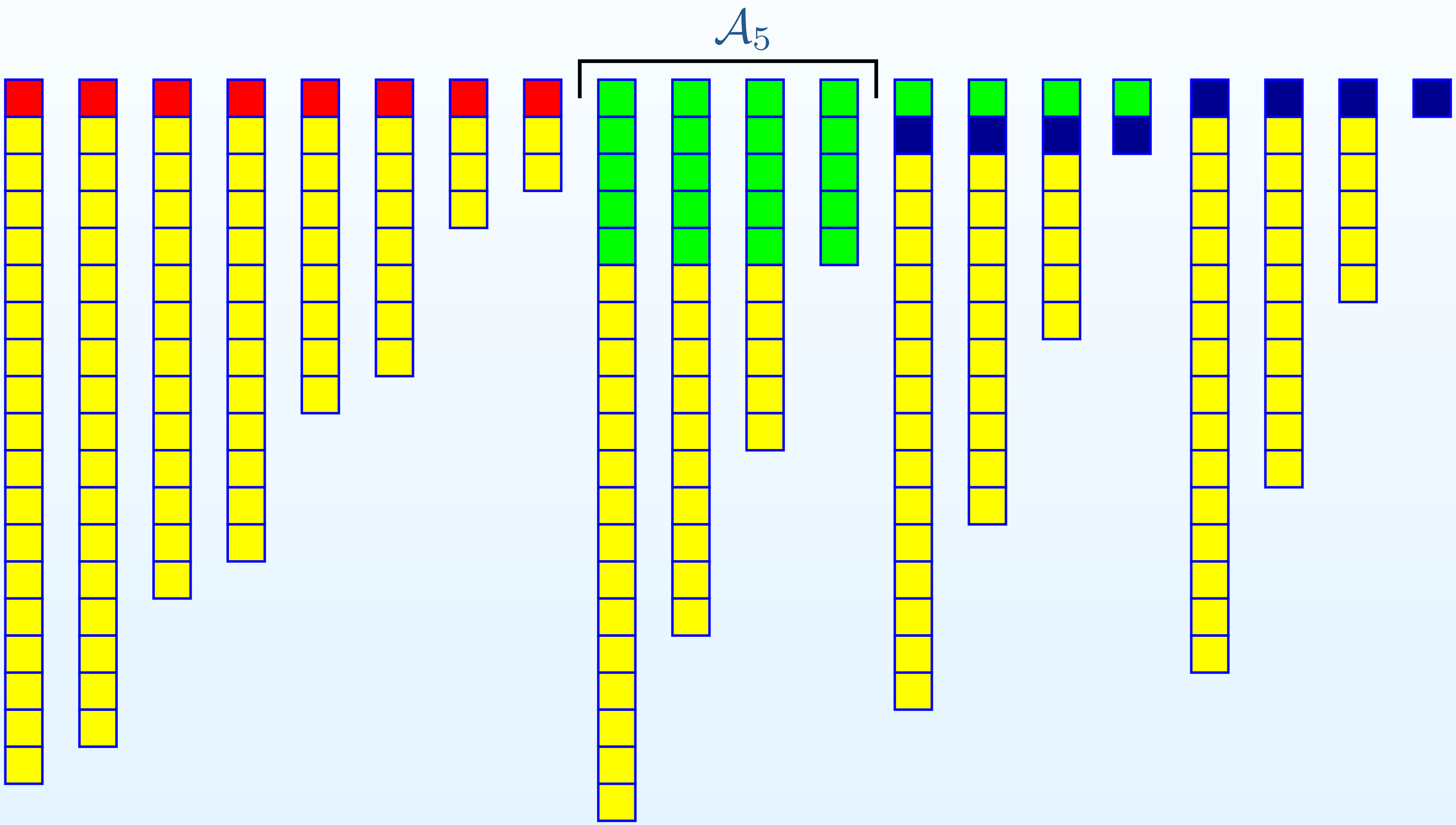
$k = 10$, $l_5 = 8$, Phase 5

Suffix selection, second attempt: exploiting collisions



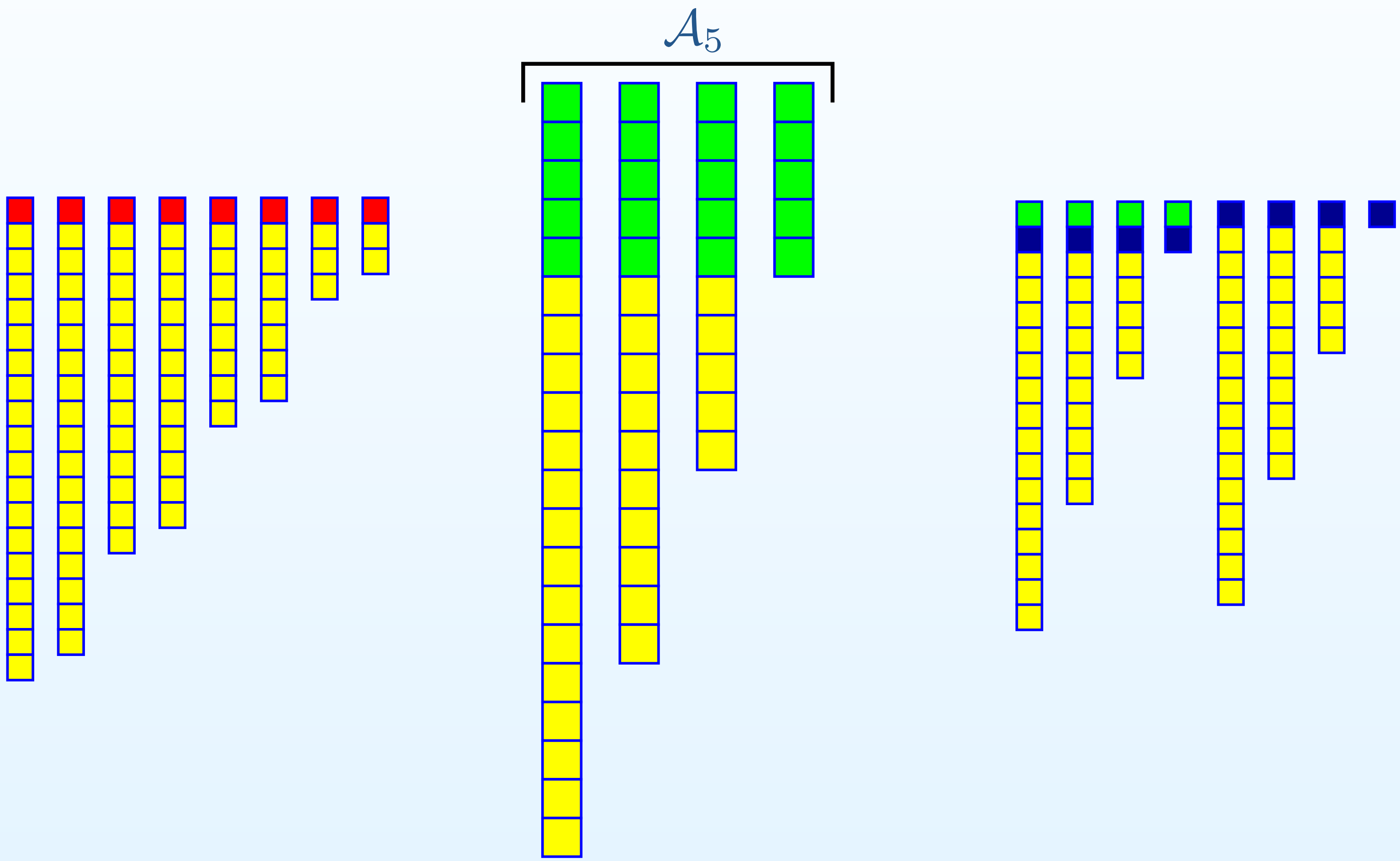
$k = 10$, $l_5 = 8$, Phase 5

Suffix selection, second attempt: exploiting collisions



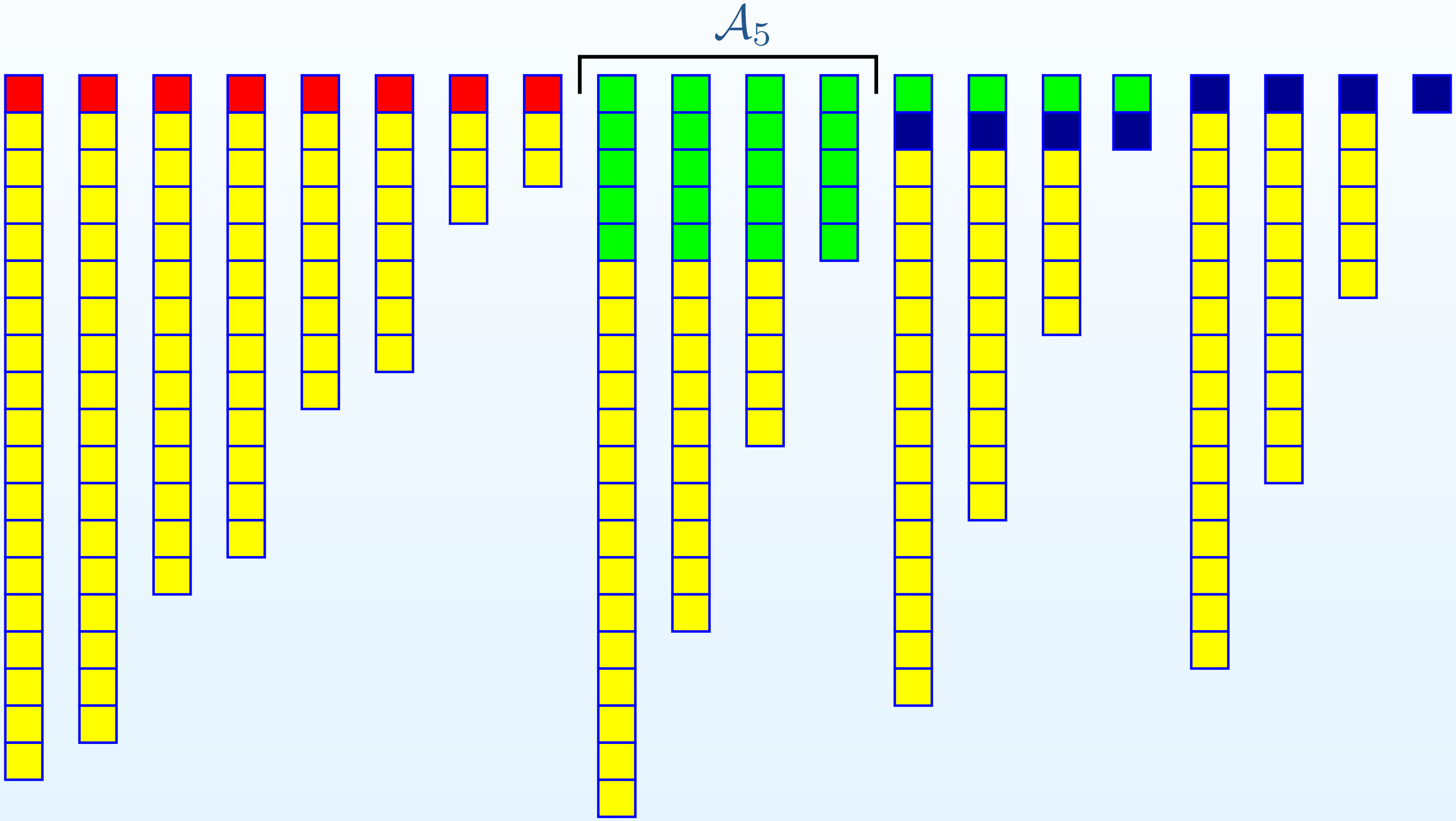
$k = 10$, $l_5 = 8$, Phase 5

Suffix selection, second attempt: exploiting collisions



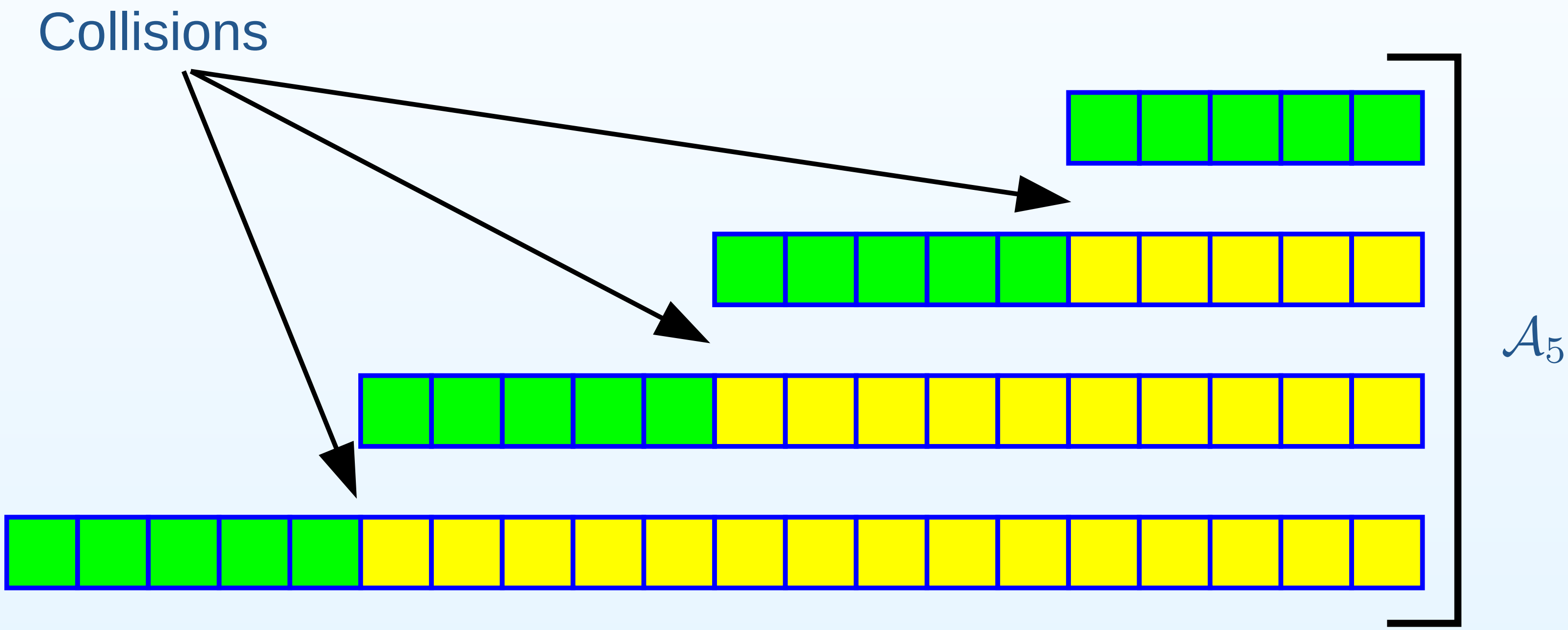
$k = 10$, $l_5 = 8$, Phase 5

Suffix selection, second attempt: exploiting collisions



$k = 10$, $l_5 = 8$, Phase 5

Suffix selection, second attempt: exploiting collisions



Suffix selection, second attempt: exploiting collisions

Let's consider a *Phase Transition* from phase t to $t+1$.

Suffix selection, second attempt: exploiting collisions

Let's consider a *Phase Transition* from phase t to $t+1$.

- If there are *no collisions of active suffixes* in phase t, the transition proceeds as before (we try to enlarge *by just one element* the extents).

Suffix selection, second attempt: exploiting collisions

Let's consider a *Phase Transition* from phase t to $t+1$.

- If there are *no collisions of active suffixes* in phase t, the transition proceeds as before (we try to enlarge *by just one element* the extents).
- Otherwise, it can be proven that the *extents of the colliding active* suffixes *are simply adjacent and do not overlap*.

Suffix selection, second attempt: exploiting collisions

Let's consider a *Phase Transition* from phase t to $t+1$.

- If there are *no collisions of active suffixes* in phase t, the transition proceeds as before (we try to enlarge *by just one element* the extents).
- Otherwise, it can be proven that the *extents of the colliding active* suffixes *are simply adjacent and do not overlap*.

  Let the *prospective extent* of an active suffix T_i be composed by the following:
  - *The subsequence of extents following it* (just T_i's extent, in case T_i does not collide).
  - *The element c_i next to the extent of the rightmost suffix in the collision*.

Suffix selection, second attempt: exploiting collisions

Let's consider a *Phase Transition* from phase t to $t+1$.

- If there are *no collisions of active suffixes* in phase t, the transition proceeds as before (we try to enlarge *by just one element* the extents).
- Otherwise, it can be proven that the *extents of the colliding active* suffixes *are simply adjacent and do not overlap*.

> Let the *prospective extent* of an active suffix T_i be composed by the following:
>
> - *The subsequence of extents following it* (just T_i's extent, in case T_i does not collide).
> - *The element c_i next to the extent of the rightmost suffix in the collision*.

- Since the extent of an active suffix T_i is σ_t, *the prospective extent of* T_i is *the periodic sequence*

$$\overbrace{\sigma_t\sigma_t\sigma_t\cdots\sigma_t}^{r_i}\, c_i$$

  for an integer r_i.

Suffix selection, second attempt: exploiting collisions

Let's consider a *Phase Transition* from phase t to $t+1$.

- If there are *no collisions of active suffixes* in phase t, the transition proceeds as before (we try to enlarge *by just one element* the extents).
- Otherwise, it can be proven that the *extents of the colliding active* suffixes *are simply adjacent and do not overlap*.

Let the *prospective extent* of an active suffix T_i be composed by the following:

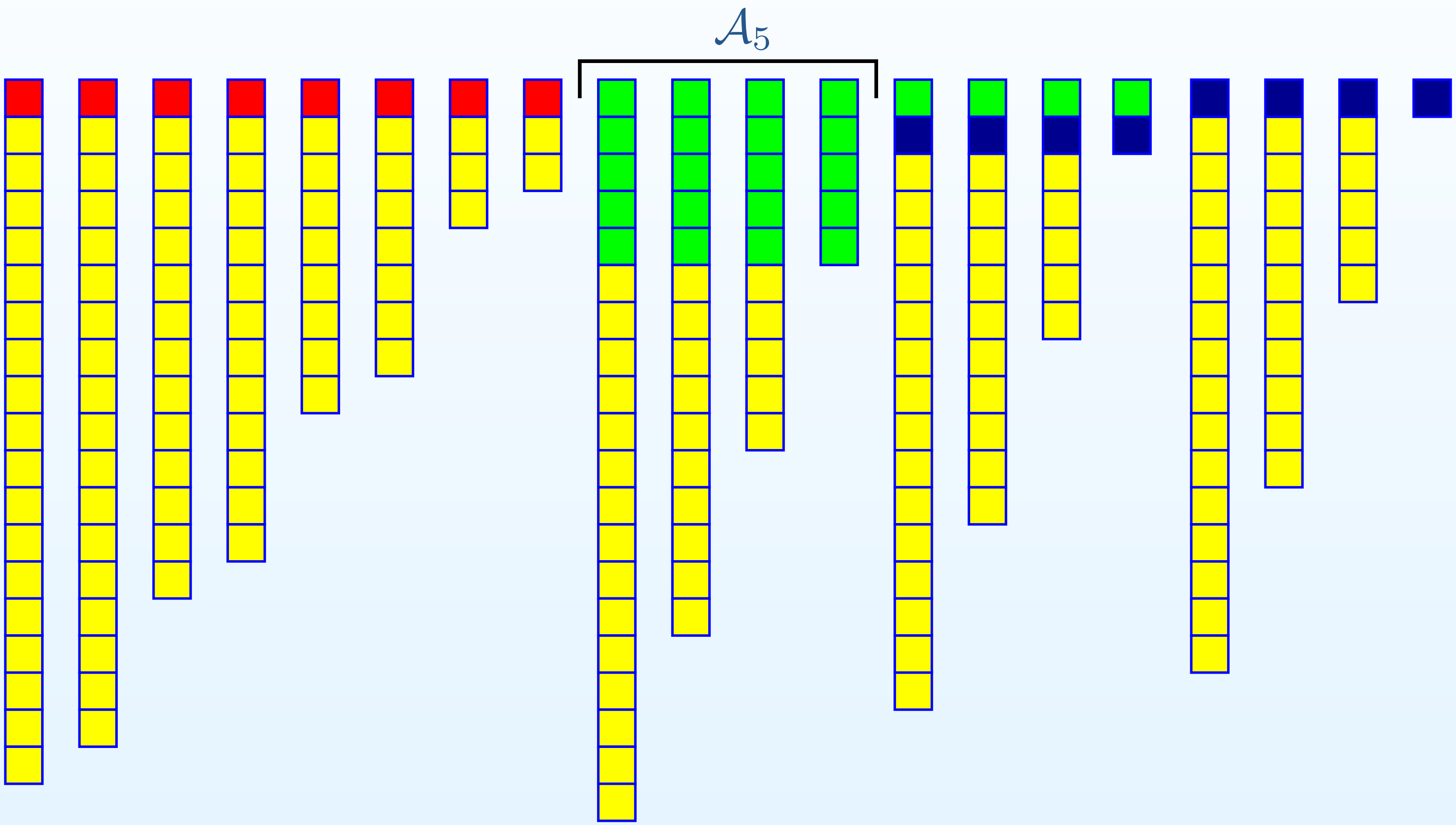
- *The subsequence of extents following it* (just T_i's extent, in case T_i does not collide).
- *The element c_i next to the extent of the rightmost suffix in the collision*.

- Since the extent of an active suffix T_i is σ_t, *the prospective extent of T_i* is *the periodic sequence*

$$\overbrace{\sigma_t \sigma_t \sigma_t \cdots \sigma_t}^{r_i} c_i$$

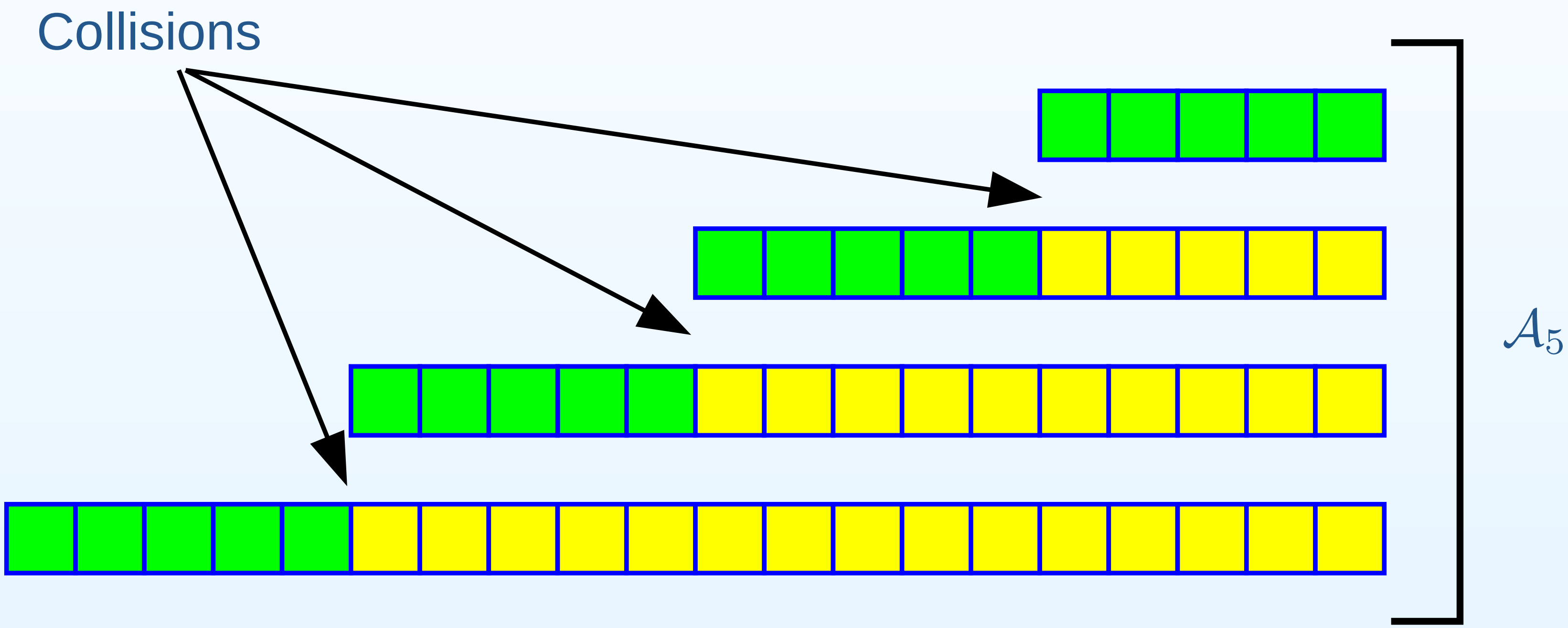
  for an integer r_i.
- Therefore, *the prospective extents* of any two active suffixes *can be compared in $O(1)$ time*.

Suffix selection, second attempt: exploiting collisions

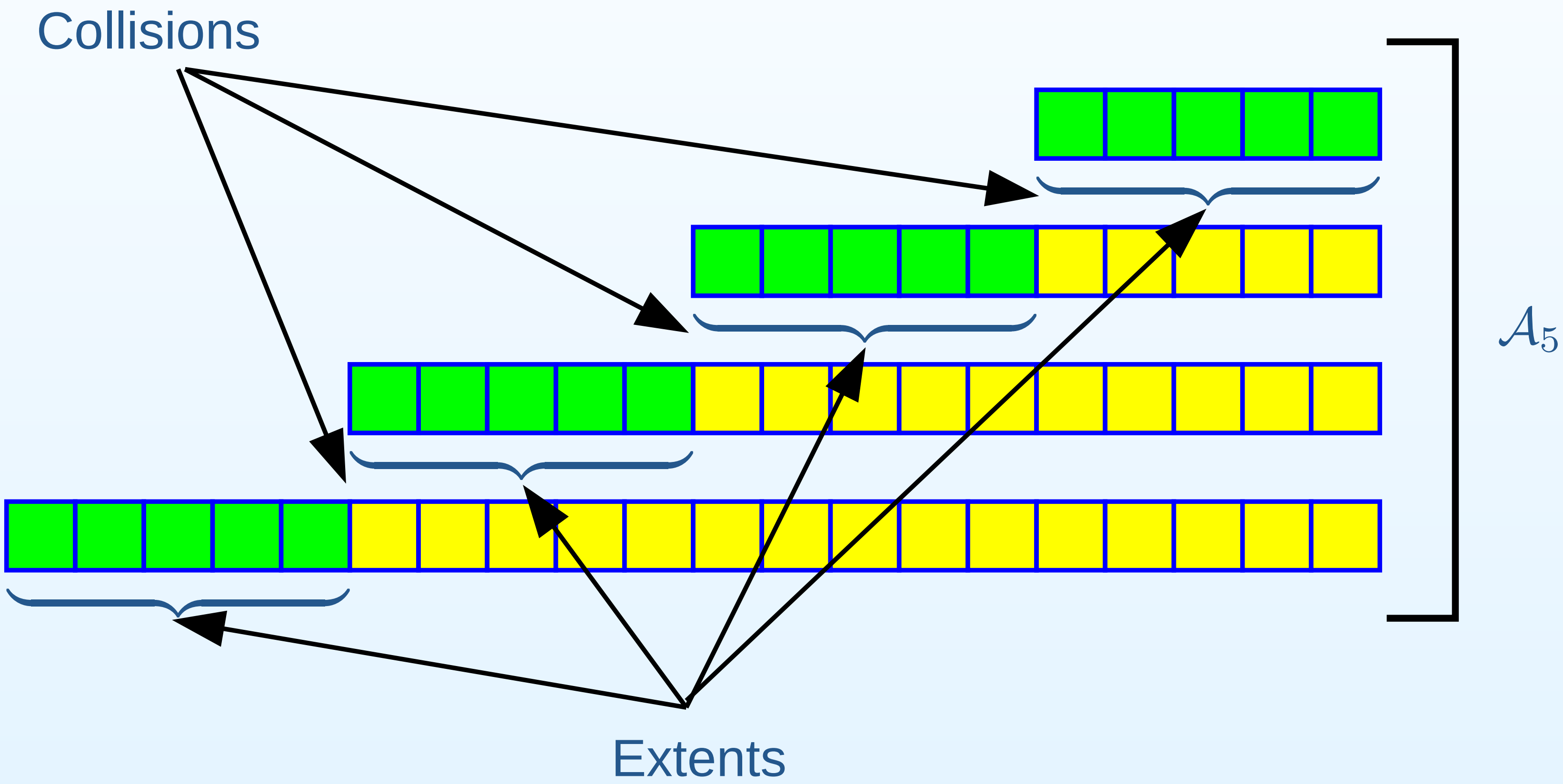


$k = 10$, $l_5 = 8$, Phase 5

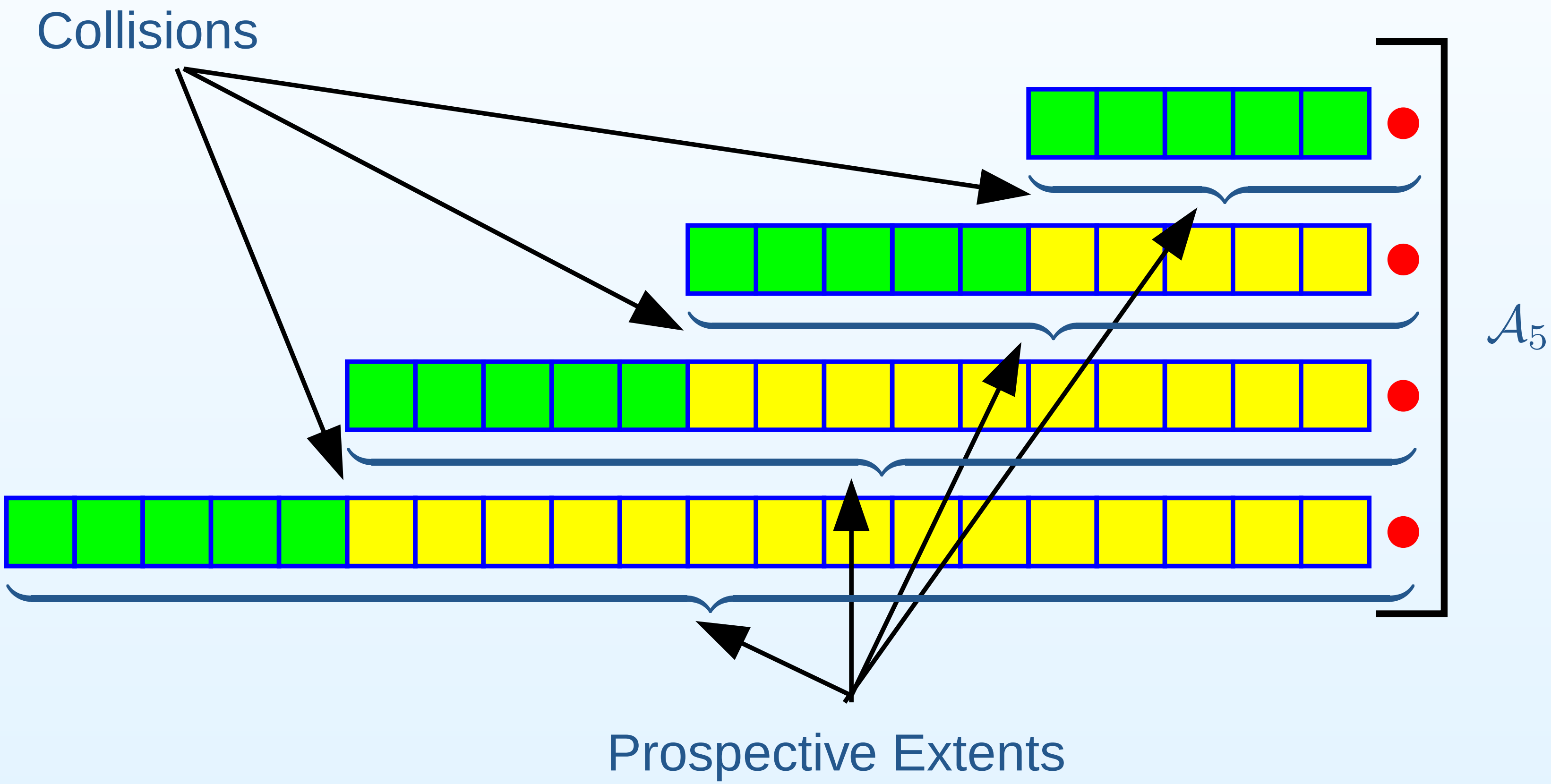
Suffix selection, second attempt: exploiting collisions



Suffix selection, second attempt: exploiting collisions



Suffix selection, second attempt: exploiting collisions



Suffix selection, second attempt: exploiting collisions

Let's go back to the *Phase Transition* from phase t to $t+1$.

Suffix selection, second attempt: exploiting collisions

Let's go back to the *Phase Transition* from phase t to $t + 1$.

- If there are no collisions of active suffixes in Phase t,

Suffix selection, second attempt: exploiting collisions

Let's go back to the *Phase Transition* from phase t to $t+1$.

- If there are no collisions of active suffixes in Phase t,
  - We select from the *multiset* $\mathcal{D}_t = \{T_i[t+1] \mid T_i \in \mathcal{A}_t\}$ the $(k - l_t)$*-th smallest element* α_{t+1}, using [Blum et al. 1973].

Suffix selection, second attempt: exploiting collisions

Let's go back to the *Phase Transition* from phase t to $t+1$.

- If there are no collisions of active suffixes in Phase t,
  - We select from the *multiset* $\mathcal{D}_t = \{T_i[t+1] \mid T_i \in \mathcal{A}_t\}$ the $(k-l_t)$*-th smallest element* α_{t+1}, using [Blum et al. 1973].
  - We set $\sigma_{t+1} = \sigma_t \alpha_{t+1}$.

Suffix selection, second attempt: exploiting collisions

Let's go back to the *Phase Transition* from phase t to $t+1$.

- If there are no collisions of active suffixes in Phase t,
  - We select from the *multiset* $\mathcal{D}_t = \{T_i[t+1] \mid T_i \in \mathcal{A}_t\}$ the $(k - l_t)$*-th smallest element* α_{t+1}, using [Blum et al. 1973].
  - We set $\sigma_{t+1} = \sigma_t \alpha_{t+1}$.
  - $\mathcal{A}_{t+1}$ contains all the suffixes in $\mathcal{A}_t$ having α_{t+1} *as their* $(t+1)$*-th element*.

Suffix selection, second attempt: exploiting collisions

Let's go back to the *Phase Transition* from phase t to $t+1$.

- If there are no collisions of active suffixes in Phase t,
  - We select from the *multiset* $\mathcal{D}_t = \{T_i[t+1] \mid T_i \in \mathcal{A}_t\}$ the $(k - l_t)$*-th smallest element* α_{t+1}, using [Blum et al. 1973].
  - We set $\sigma_{t+1} = \sigma_t \alpha_{t+1}$.
  - $\mathcal{A}_{t+1}$ contains all the suffixes in $\mathcal{A}_t$ having α_{t+1} *as their* $(t+1)$*-th element*.
- Otherwise,

Suffix selection, second attempt: exploiting collisions

Let's go back to the *Phase Transition* from phase t to $t + 1$.

- If there are no collisions of active suffixes in Phase t,
  - We select from the *multiset* $\mathcal{D}_t = \{T_i[t+1] \mid T_i \in \mathcal{A}_t\}$ the $(k - l_t)$*-th smallest element* α_{t+1}, using [Blum et al. 1973].
  - We set $\sigma_{t+1} = \sigma_t \alpha_{t+1}$.
  - $\mathcal{A}_{t+1}$ contains all the suffixes in $\mathcal{A}_t$ having α_{t+1} *as their* $(t+1)$*-th element*.
- Otherwise,
  - We select the $(k - l_t)$*-th smallest subsequence* π_{t+1}. from the *multiset*

    $$\mathcal{F}_t = \Big\{ (\sigma_t)^{r_i} c_i \Big| \begin{array}{ll} & T_i \in \mathcal{A}_t \\ \textit{and} & (\sigma_t)^{r_i} c_i \textit{ is the prosp. ext. of } T_i \end{array} \Big\}$$

    using [Blum et al. 1973] (two subsequences in $\mathcal{F}_t$ can be compared in $O(1)$).

Suffix selection, second attempt: exploiting collisions

Let's go back to the *Phase Transition* from phase t to $t+1$.

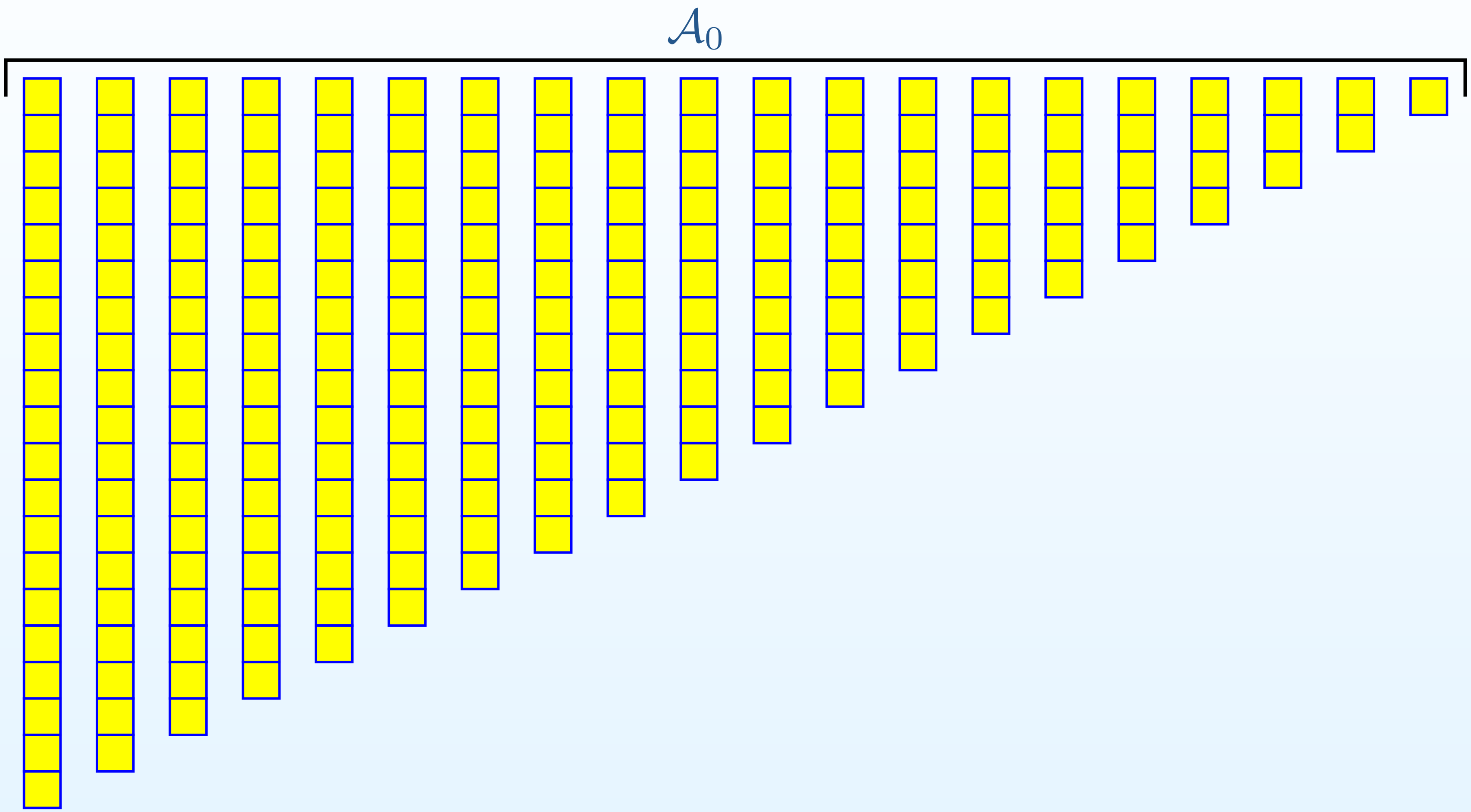
- If there are no collisions of active suffixes in Phase t,
  - We select from the *multiset* $\mathcal{D}_t = \{T_i[t+1] \mid T_i \in \mathcal{A}_t\}$ the $(k - l_t)$*-th smallest element* α_{t+1}, using [Blum et al. 1973].
  - We set $\sigma_{t+1} = \sigma_t \alpha_{t+1}$.
  - $\mathcal{A}_{t+1}$ contains all the suffixes in $\mathcal{A}_t$ having α_{t+1} *as their* $(t+1)$*-th element*.
- Otherwise,
  - We select the $(k - l_t)$*-th smallest subsequence* π_{t+1}. from the *multiset*
    $$\mathcal{F}_t = \Big\{ (\sigma_t)^{r_i} c_i \Big| \quad T_i \in \mathcal{A}_t \text{ and } (\sigma_t)^{r_i} c_i \text{ is the prosp. ext. of } T_i \Big\}$$
    using [Blum et al. 1973] (two subsequences in $\mathcal{F}_t$ can be compared in $O(1)$).
  - We set $\sigma_{t+1} = \pi_{t+1}$.

Suffix selection, second attempt: exploiting collisions

Let's go back to the *Phase Transition* from phase t to $t+1$.

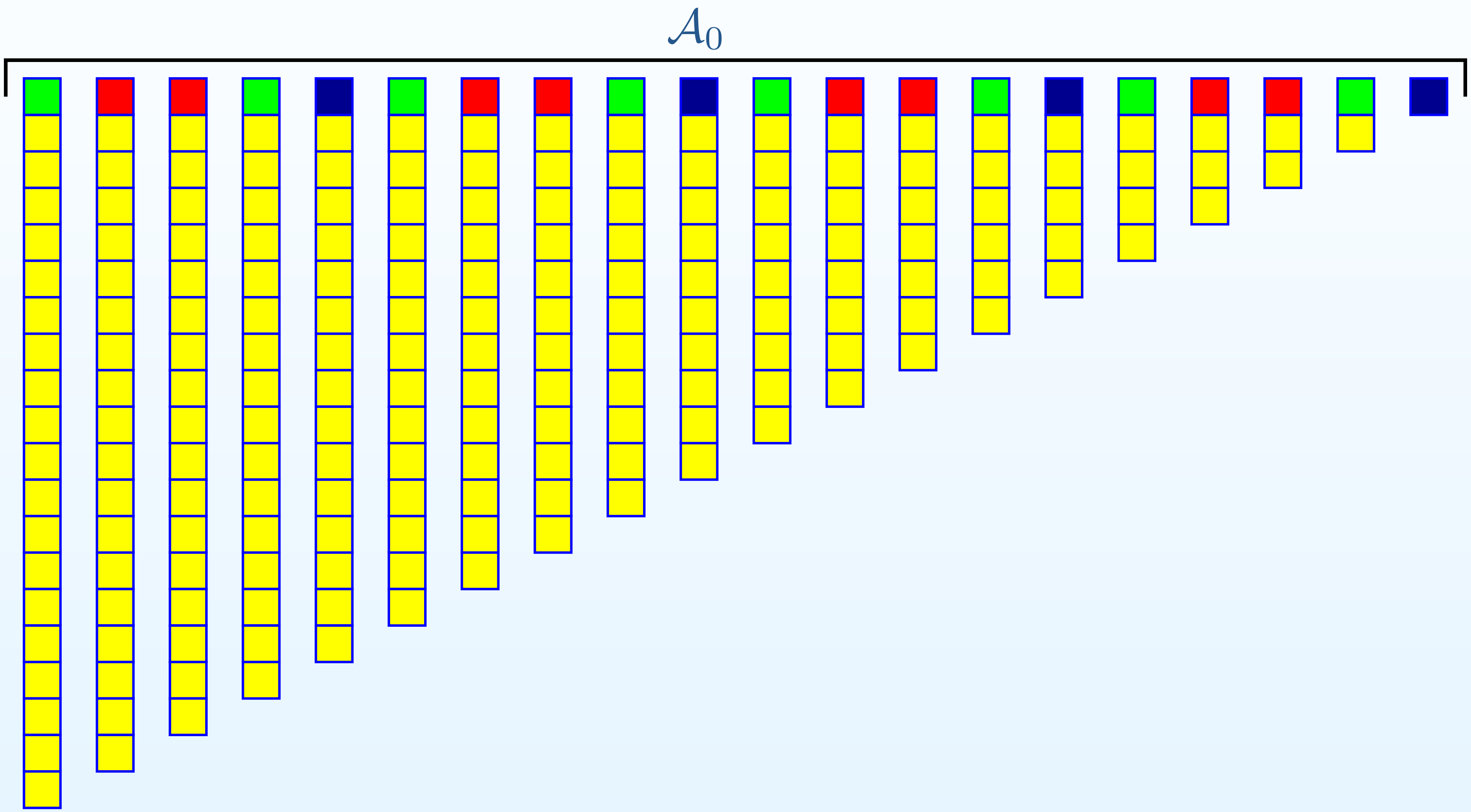
- If there are no collisions of active suffixes in Phase t,
  - We select from the *multiset* $\mathcal{D}_t = \{T_i[t+1] \mid T_i \in \mathcal{A}_t\}$ the $(k - l_t)$*-th smallest element* α_{t+1}, using [Blum et al. 1973].
  - We set $\sigma_{t+1} = \sigma_t \alpha_{t+1}$.
  - $\mathcal{A}_{t+1}$ contains all the suffixes in $\mathcal{A}_t$ having α_{t+1} *as their* $(t+1)$*-th element*.
- Otherwise,
  - We select the $(k - l_t)$*-th smallest subsequence* π_{t+1}. from the *multiset*
    $$\mathcal{F}_t = \Big\{ (\sigma_t)^{r_i} c_i \Big| \begin{array}{ll} & T_i \in \mathcal{A}_t \\ \textit{and} & (\sigma_t)^{r_i} c_i \textit{ is the prosp. ext. of } T_i \end{array} \Big\}$$
    using [Blum et al. 1973] (two subsequences in $\mathcal{F}_t$ can be compared in $O(1)$).
  - We set $\sigma_{t+1} = \pi_{t+1}$.
  - $\mathcal{A}_{t+1}$ contains all the suffixes in $\mathcal{A}_t$ having π_{t+1} *as their extent*.

Suffix selection, second attempt: exploiting collisions



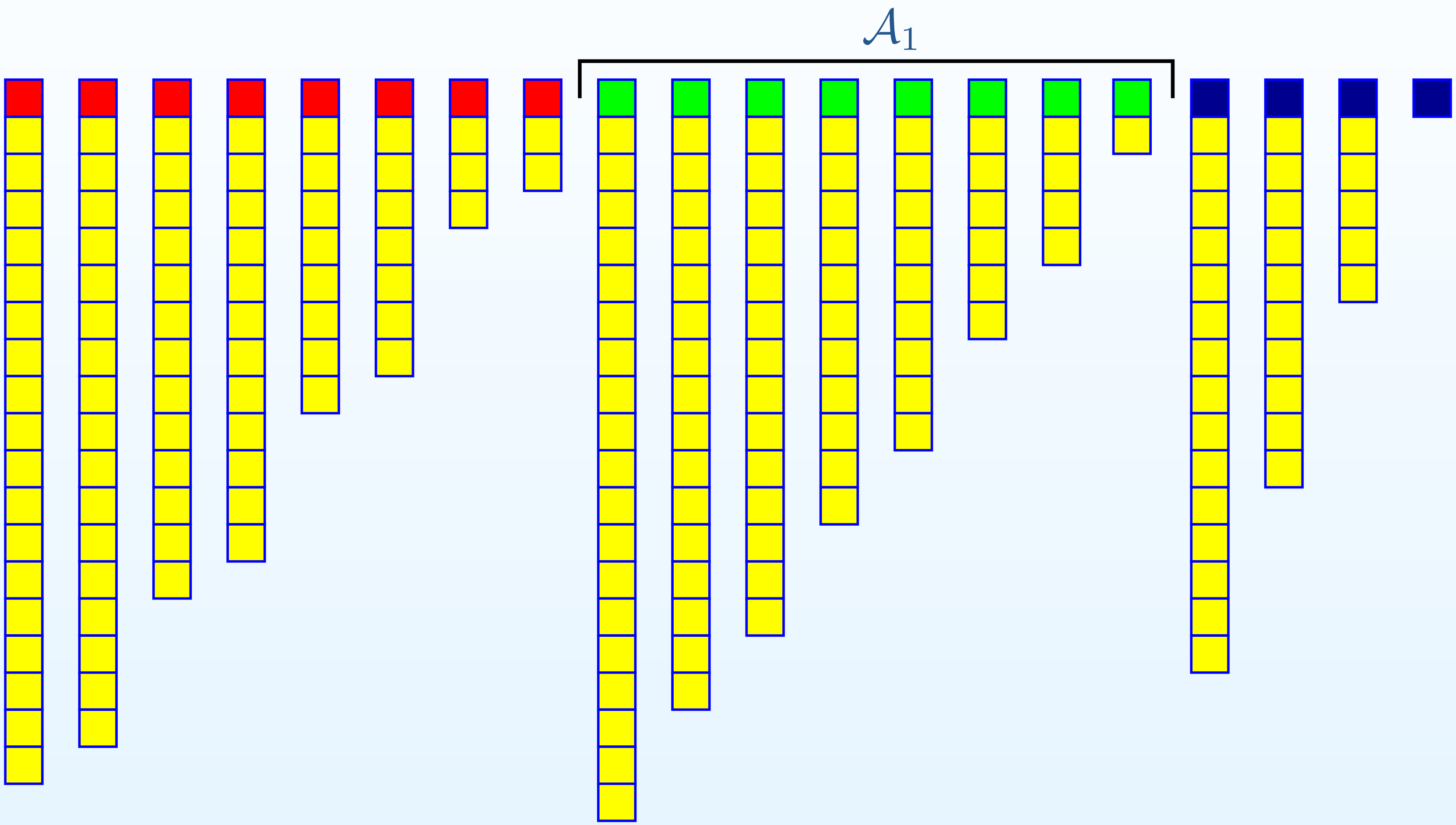
$k = 10$, $l_0 = 0$, Phase 0

Suffix selection, second attempt: exploiting collisions



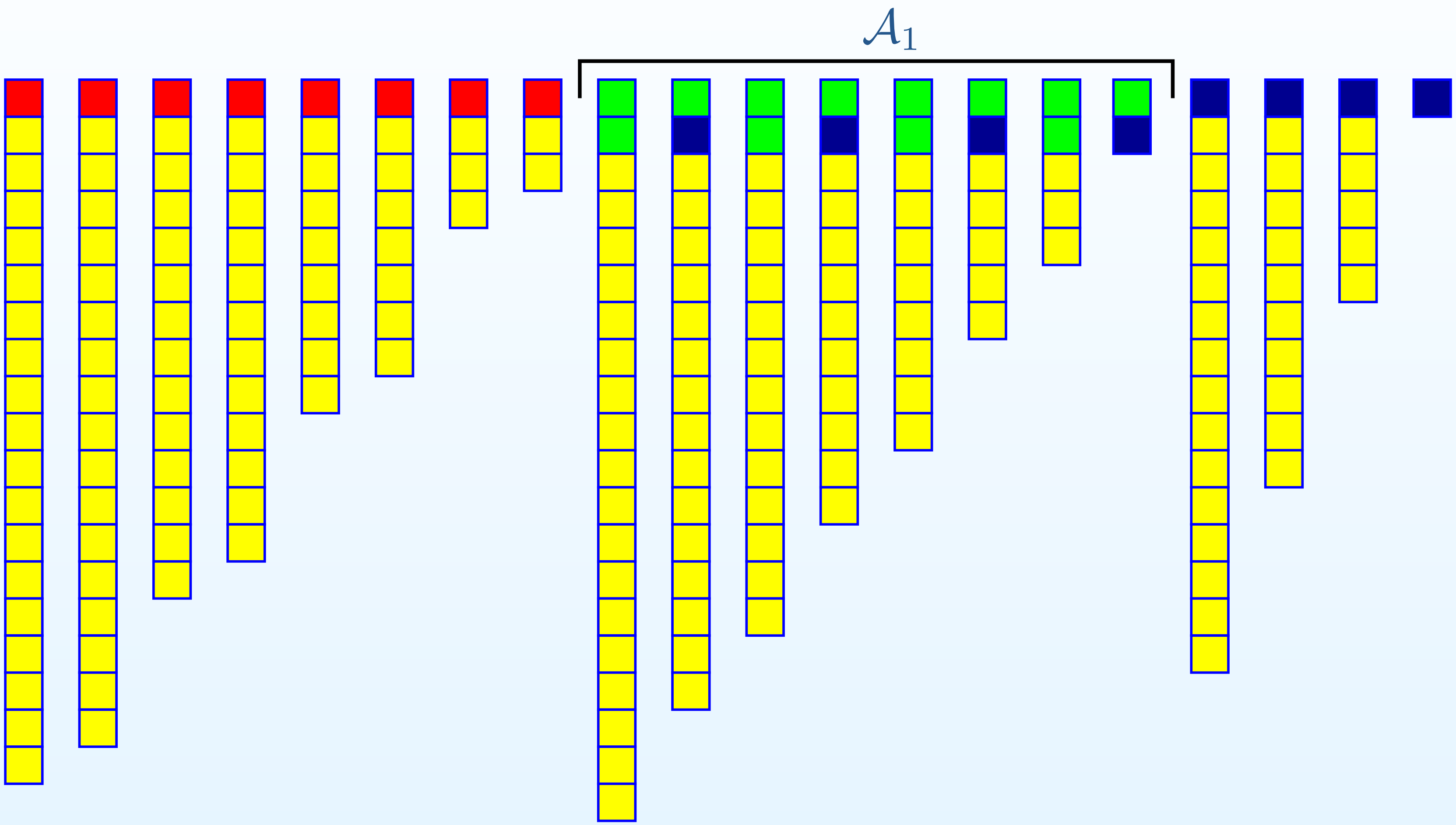
$k = 10$, $l_0 = 0$, Phase 0

Suffix selection, second attempt: exploiting collisions



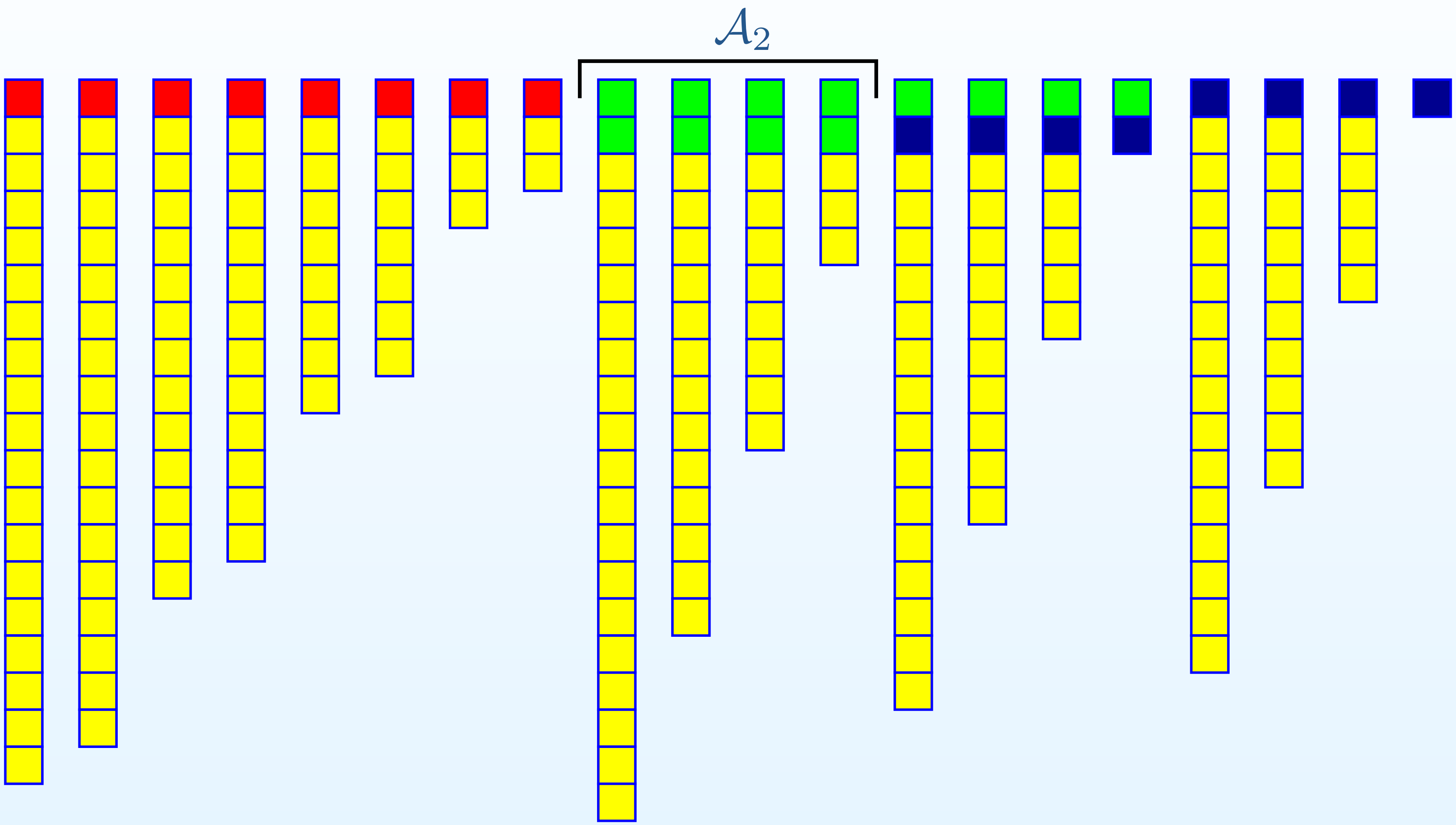
$k = 10$, $l_1 = 8$, Phase 1

Suffix selection, second attempt: exploiting collisions



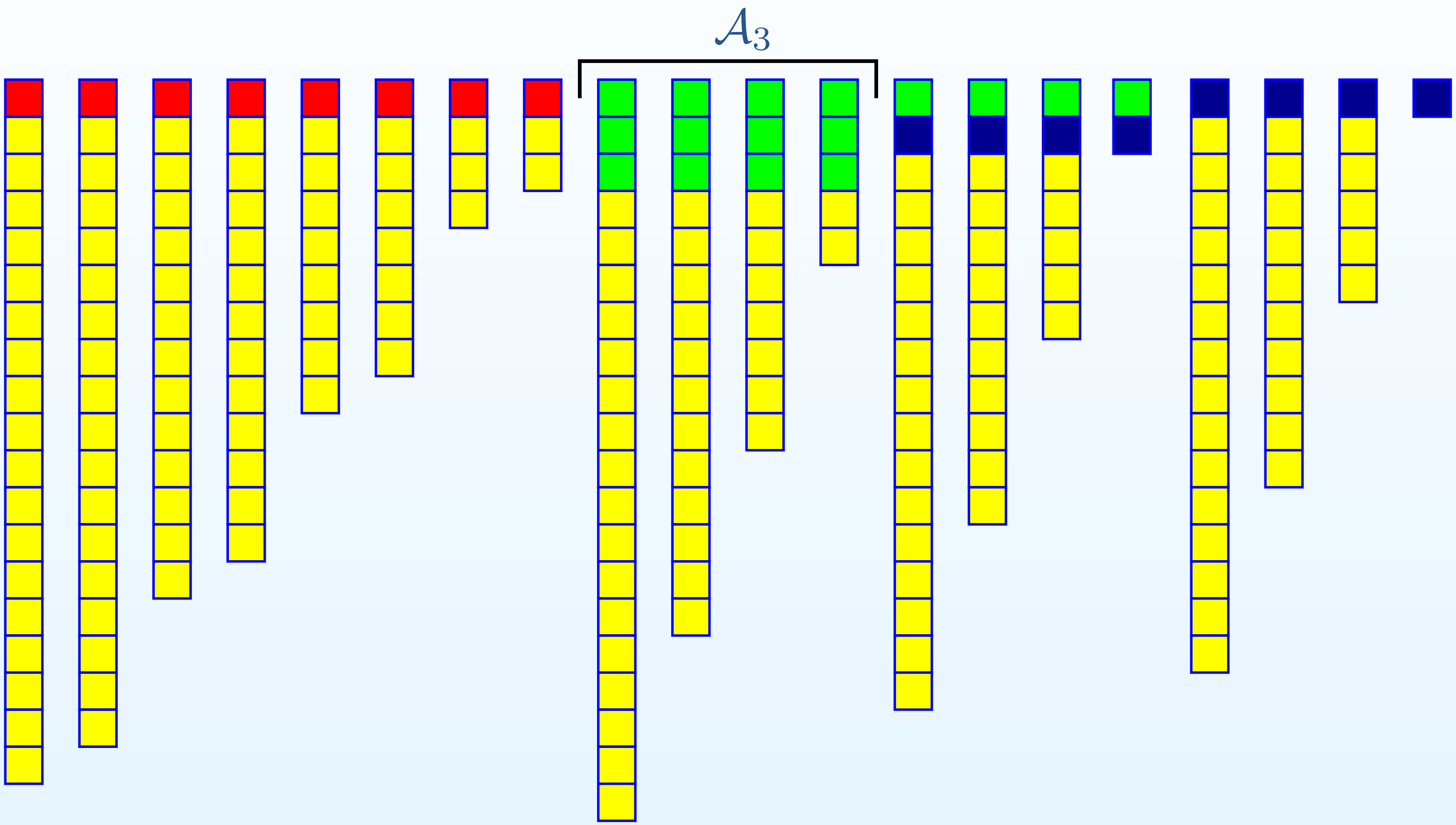
$k = 10$, $l_1 = 8$, Phase 1

Suffix selection, second attempt: exploiting collisions



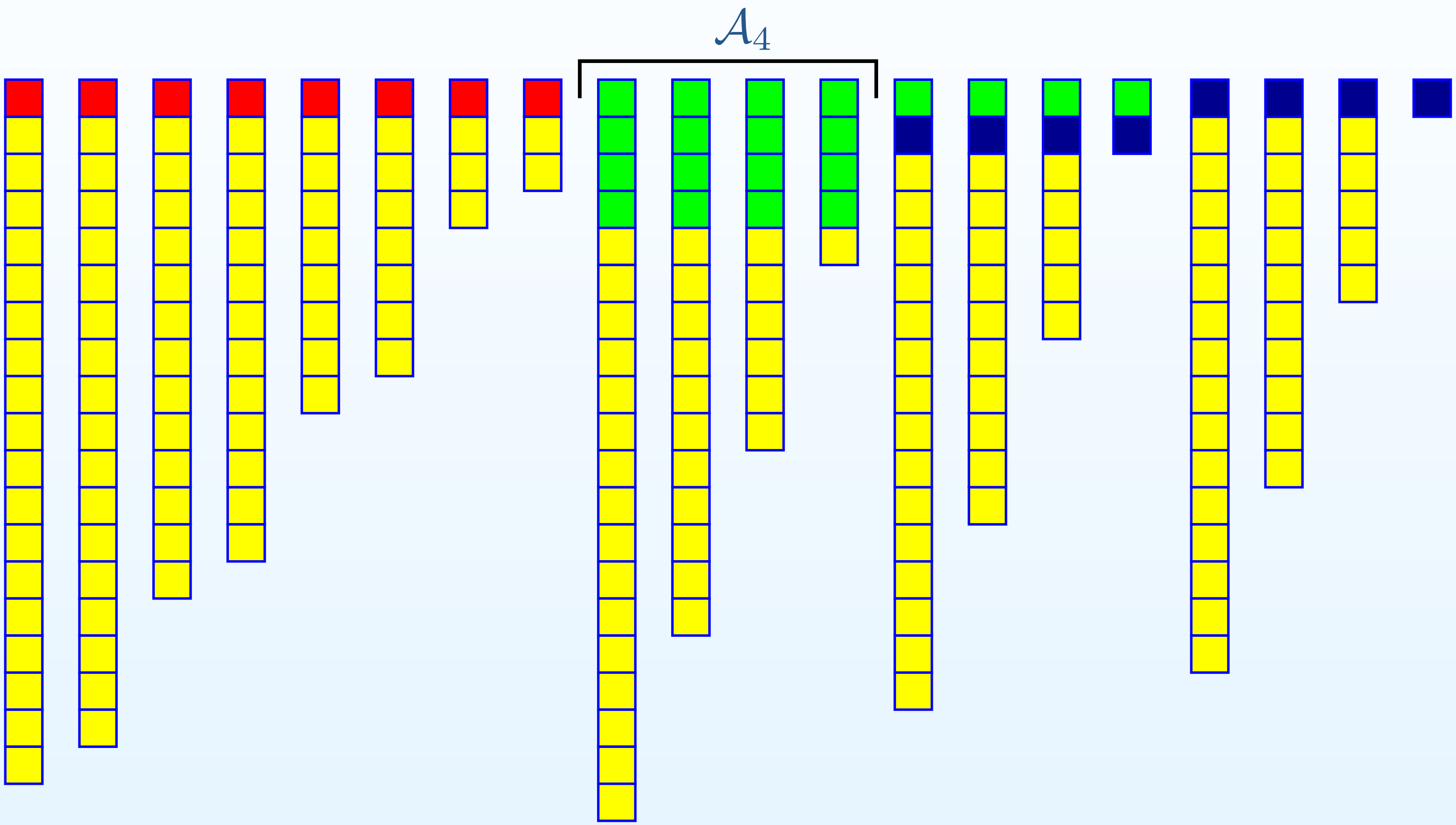
$k = 10$, $l_2 = 8$, Phase 2

Suffix selection, second attempt: exploiting collisions



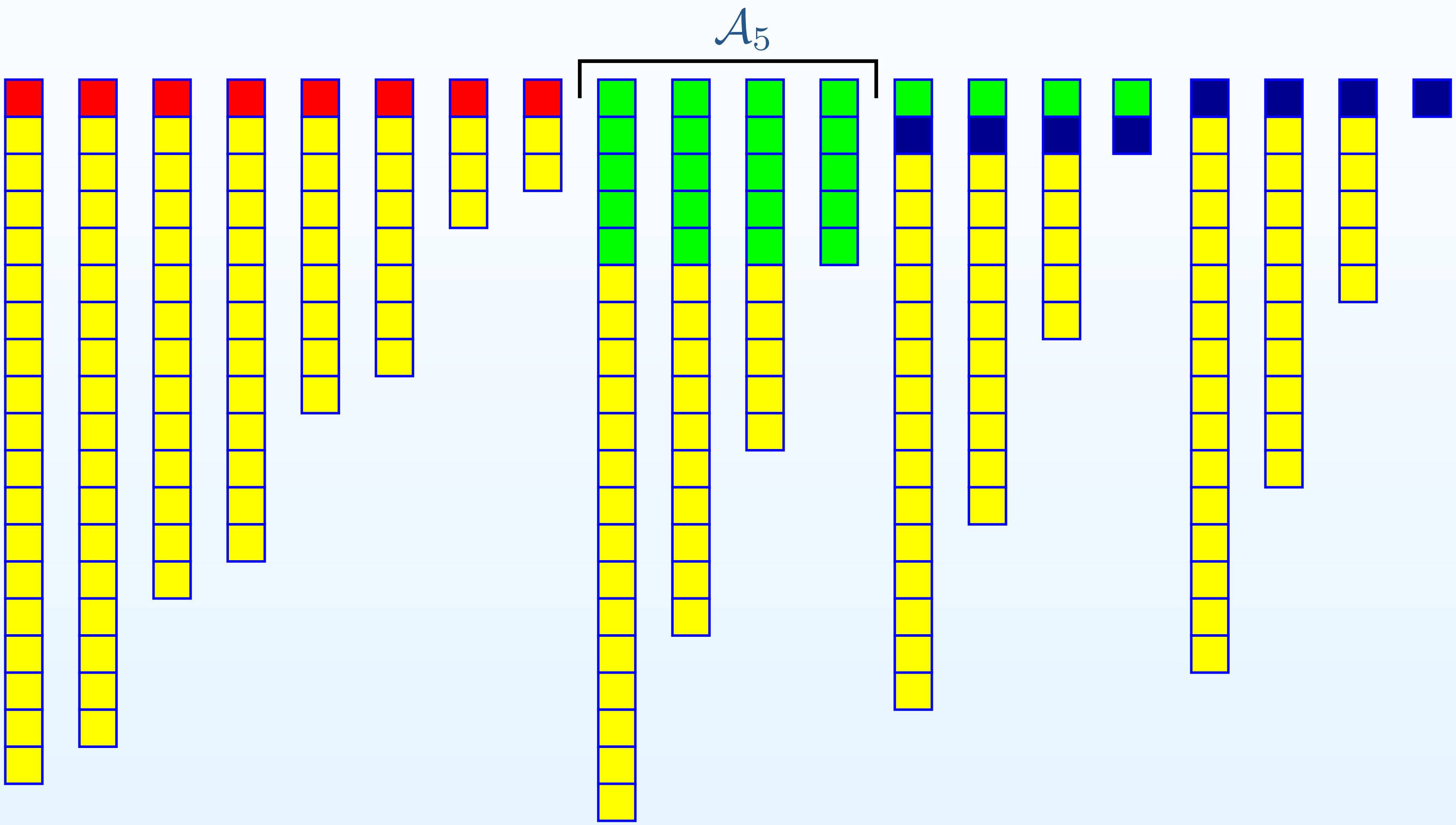
$k = 10$, $l_3 = 8$, Phase 3

Suffix selection, second attempt: exploiting collisions



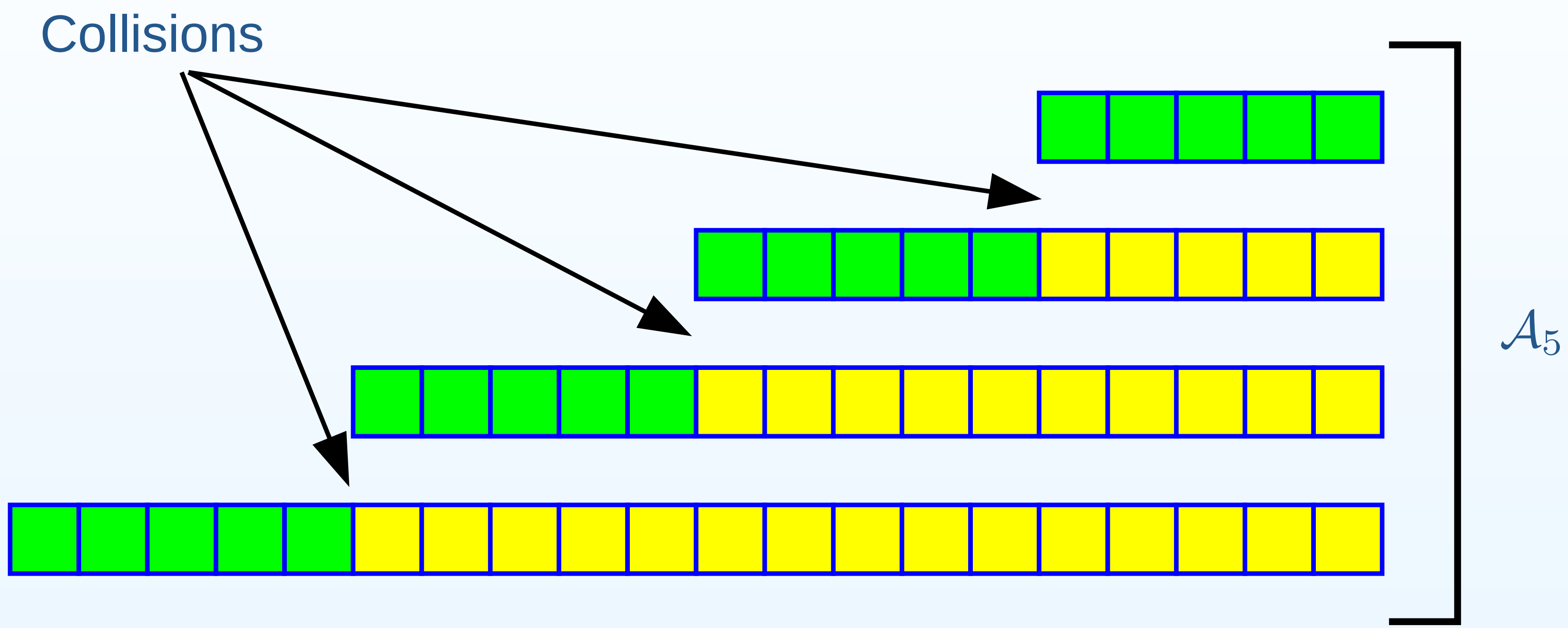
$k = 10$, $l_4 = 8$, Phase 4

Suffix selection, second attempt: exploiting collisions

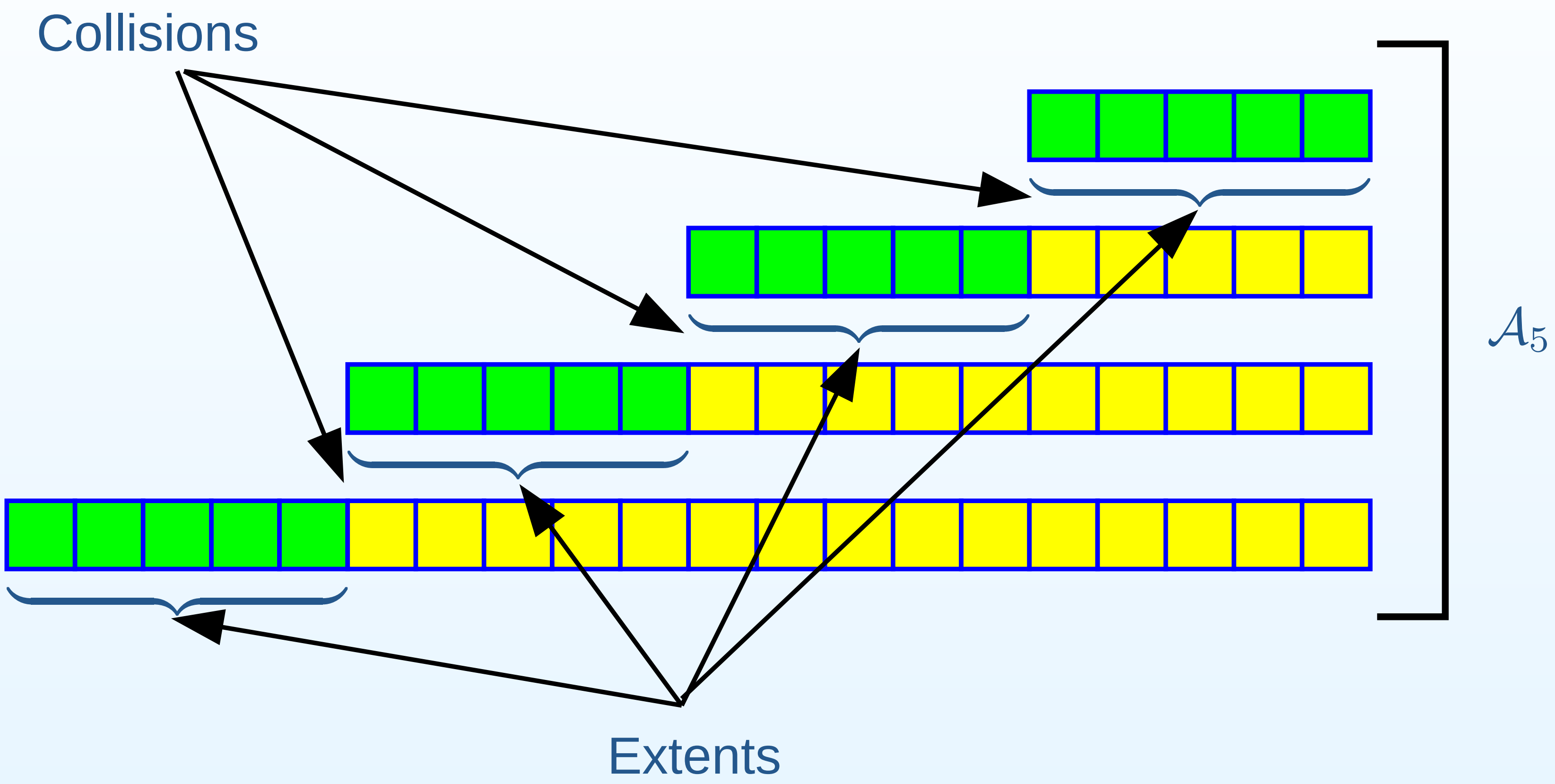


$k = 10$, $l_5 = 8$, Phase 5

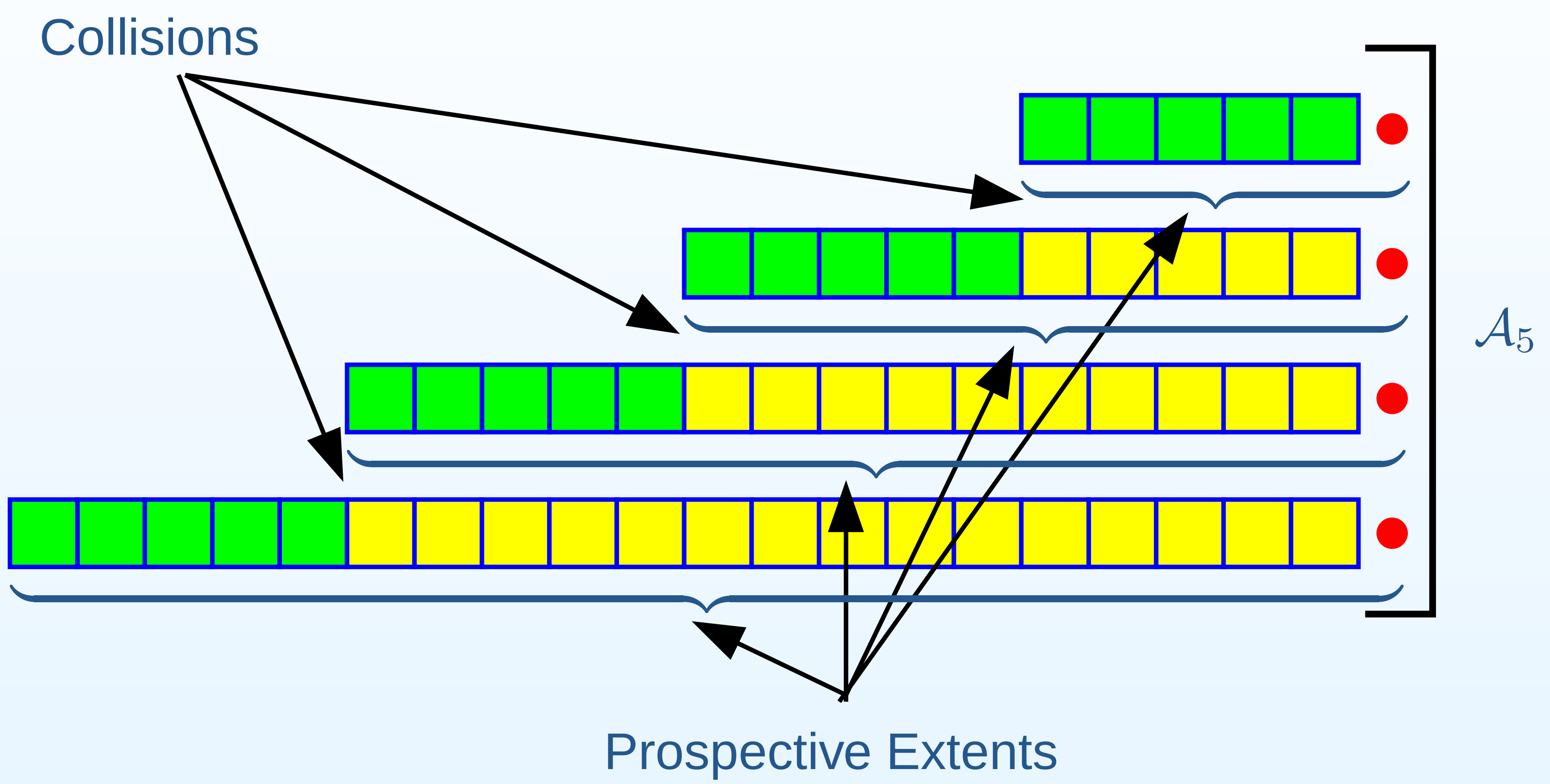
Suffix selection, second attempt: exploiting collisions



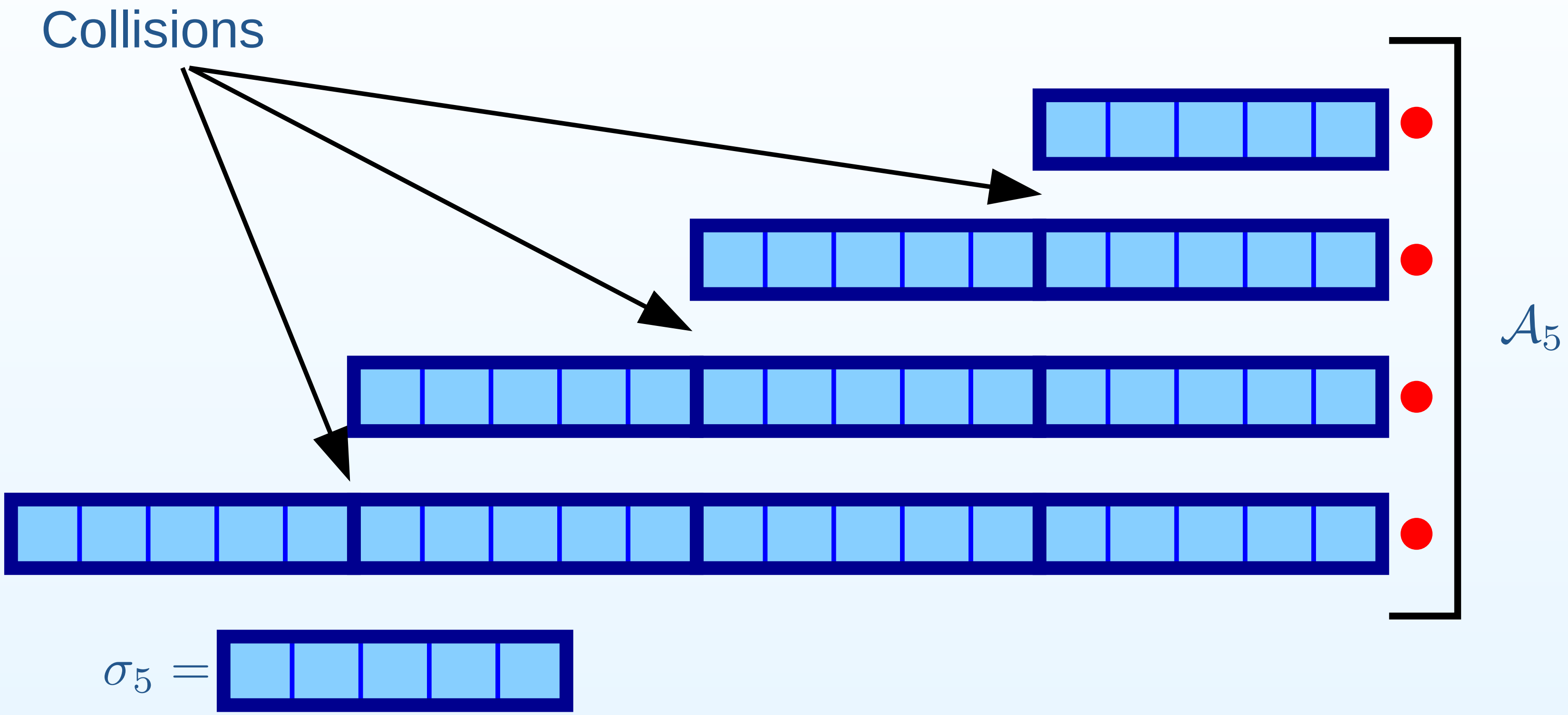
Suffix selection, second attempt: exploiting collisions



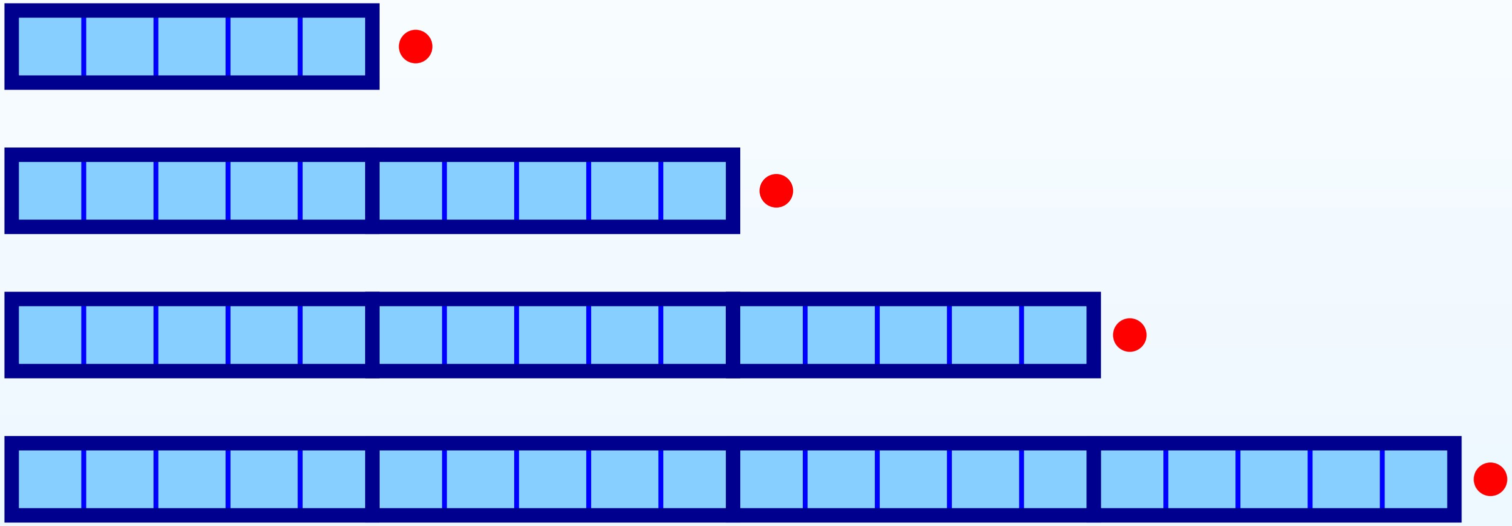
Suffix selection, second attempt: exploiting collisions



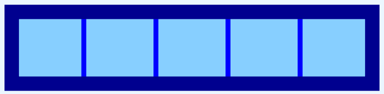
Suffix selection, second attempt: exploiting collisions



Suffix selection, second attempt: exploiting collisions

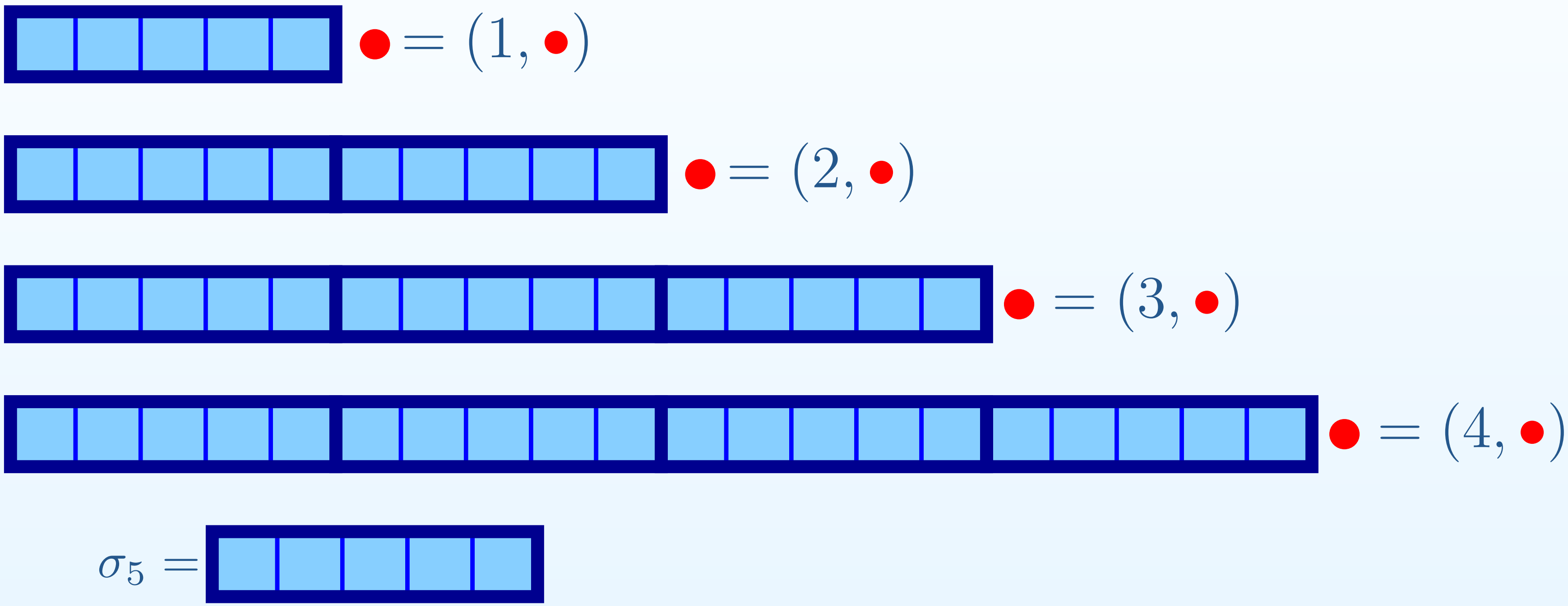


$\sigma_5 =$



How do we compare the prospective extents in multiset $\mathcal{F}_t$?

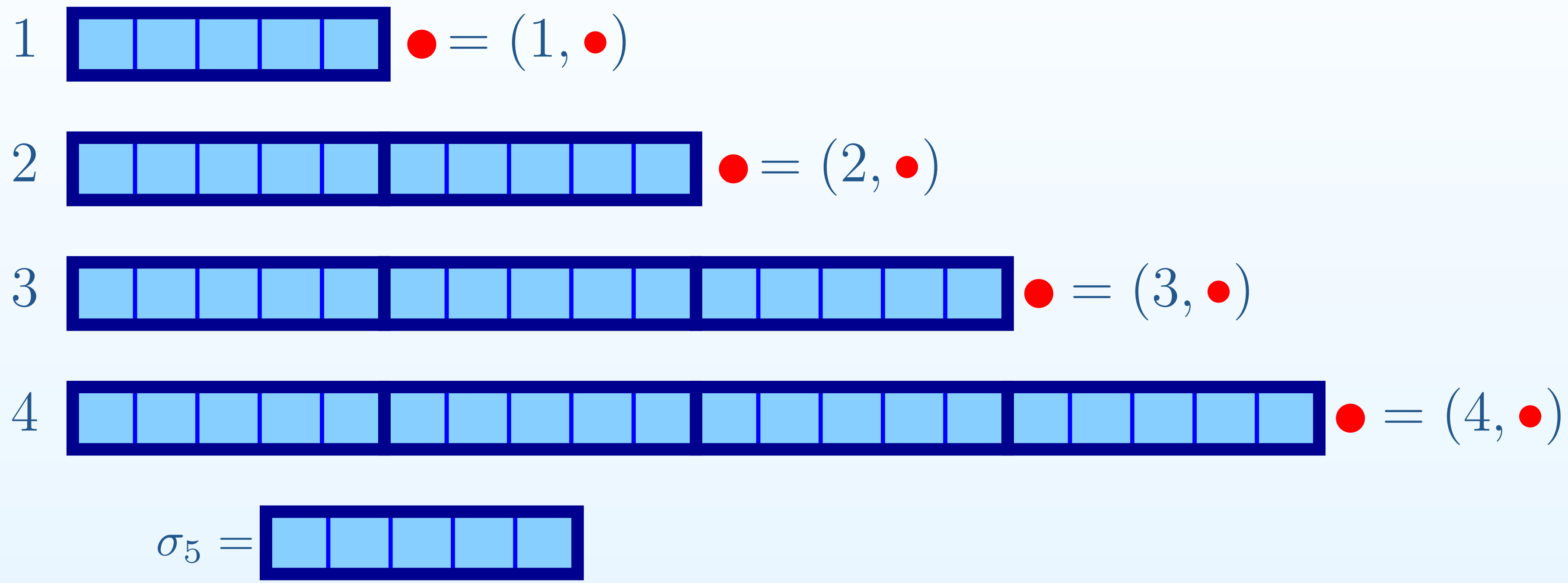
Suffix selection, second attempt: exploiting collisions



How do we compare the prospective extents in multiset $\mathcal{F}_t$?

- Each subsequence $(\sigma_t)^{r_i} c_i$ can be represented *by the pair* (r_i, c_i) (integer/element pair).

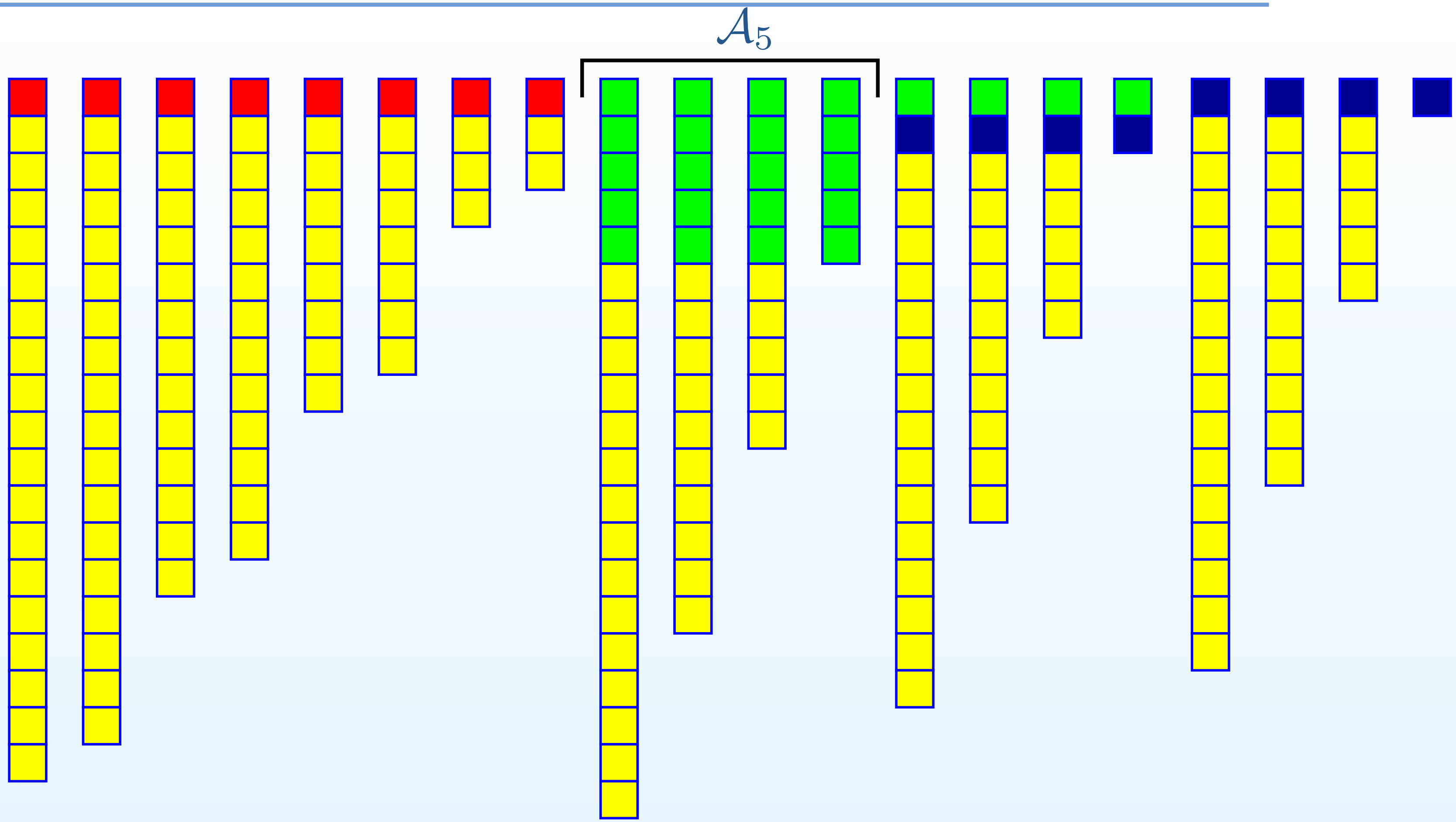
Suffix selection, second attempt: exploiting collisions



How do we compare the prospective extents in multiset $\mathcal{F}_t$?

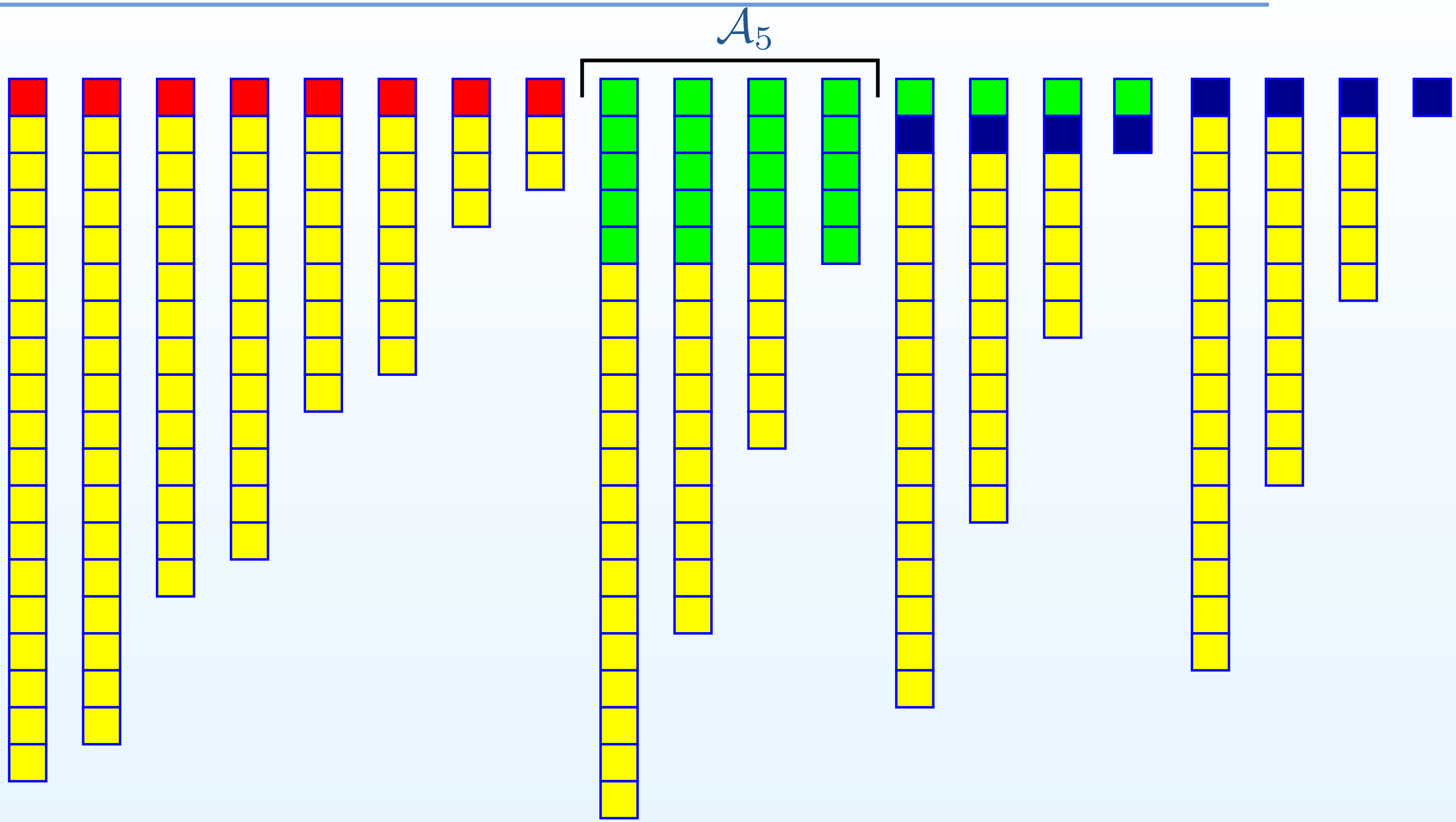
- Each subsequence $(\sigma_t)^{r_i} c_i$ can be represented *by the pair* (r_i, c_i) (integer/element pair).
- To compare two subsequences in $\mathcal{F}_t$ *we can just use their pairs*.

Suffix selection, second attempt: exploiting collisions



$k = 10$, $l_5 = 8$, Phase 5

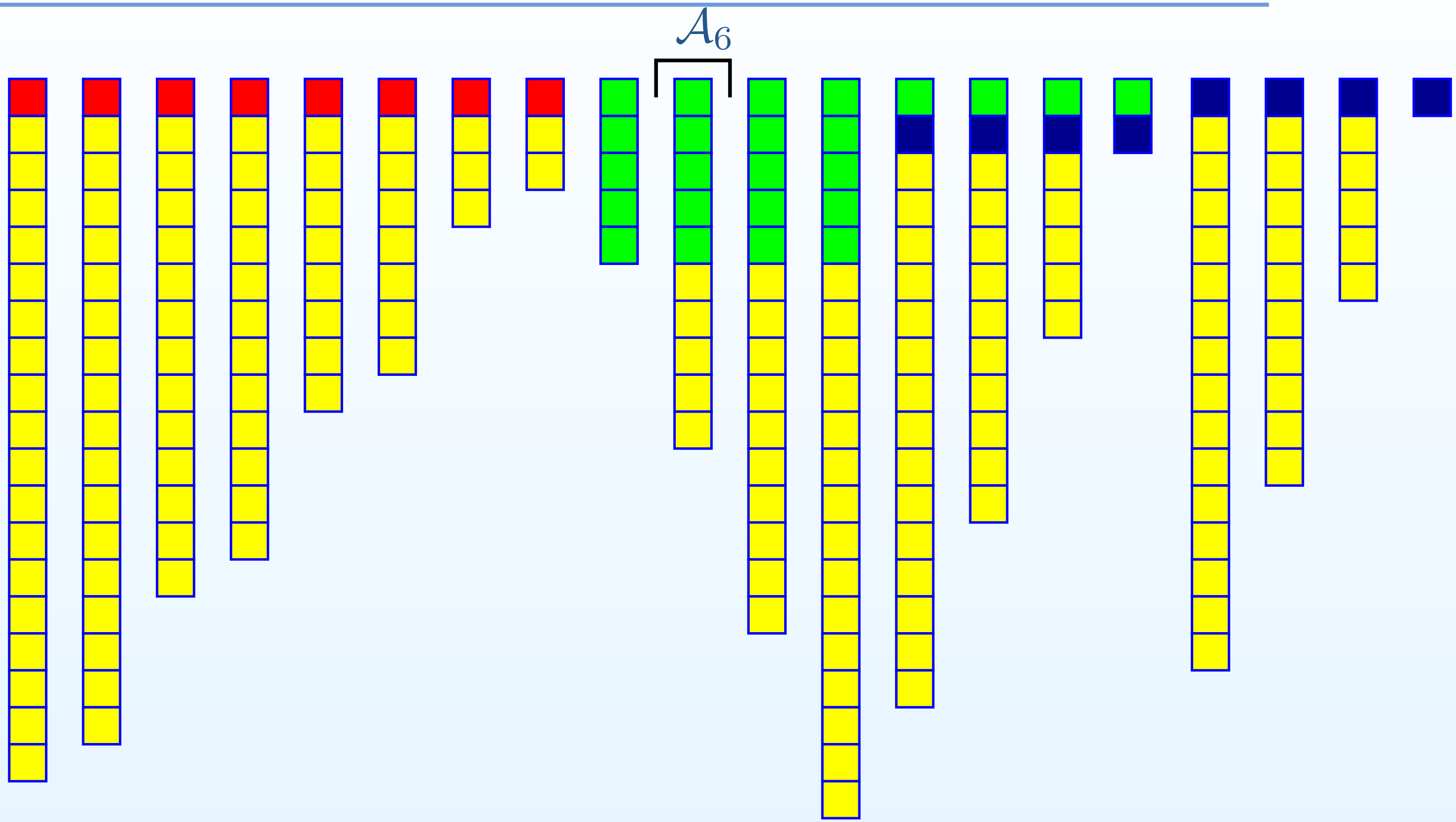
Suffix selection, second attempt: exploiting collisions



$k = 10$, $l_5 = 8$, Phase 5

Now we want the active suffix *with the prosp ext. of rank* $(k - l_5) = 2\ldots$

Suffix selection, second attempt: exploiting collisions



$k = 10$, $l_6 = 9$, Phase 6

Now we want the active suffix *with the prosp ext. of rank* $(k - l_5) = 2$. . .

. . . exploiting the collisions in $\mathcal{A}_5$, *we find it in just one Phase Transition*.

Suffix selection, second attempt: exploiting collisions

How well did we do?

Suffix selection, second attempt: exploiting collisions

How well did we do?

- In the example we went *from the* 11 *phases of the first attempt to just* 6.

Suffix selection, second attempt: exploiting collisions

How well did we do?

- In the example we went *from the* 11 *phases of the first attempt to just* 6.
- *The complexity of the second algorithm is* $O(n \log n)$ *in the worst case.*

Suffix selection, second attempt: exploiting collisions

How well did we do?

- In the example we went *from the* 11 *phases of the first attempt to just* 6.
- *The complexity of the second algorithm is* $O(n \log n)$ *in the worst case.*
  - Let us *group the phases into macro-phases* $m_0, m_1, \ldots, m_w$ such that, *for any* m_i *and any phase* $t \in m_i$, we have that
$$2^i \leq |\sigma_t| < 2^{i+1}.$$

Suffix selection, second attempt: exploiting collisions

How well did we do?

- In the example we went *from the* 11 *phases of the first attempt to just* 6.
- *The complexity of the second algorithm is* $O(n \log n)$ *in the worst case.*
  - Let us *group the phases into macro-phases* $m_0, m_1, \ldots, m_w$ such that, *for any* m_i *and any phase* $t \in m_i$, we have that
    $$2^i \leq |\sigma_t| < 2^{i+1}.$$
  - For any phase t, the extents of the active suffixes *do not overlap*...

Suffix selection, second attempt: exploiting collisions

How well did we do?

- In the example we went *from the* 11 *phases of the first attempt to just* 6.
- *The complexity of the second algorithm is* $O(n \log n)$ *in the worst case.*
  - Let us *group the phases into macro-phases* $m_0, m_1, \ldots, m_w$ such that, *for any* m_i *and any phase* $t \in m_i$, we have that
  $$2^i \leq |\sigma_t| < 2^{i+1}.$$
  - For any phase t, the extents of the active suffixes *do not overlap*...
  - ...and phase t has at most n/σ_t *active suffixes*.

Suffix selection, second attempt: exploiting collisions

How well did we do?

- In the example we went *from the* 11 *phases of the first attempt to just* 6.
- *The complexity of the second algorithm is* $O(n \log n)$ *in the worst case.*
  - Let us *group the phases into macro-phases* $m_0, m_1, \ldots, m_w$ such that, *for any* m_i *and any phase* $t \in m_i$, we have that
  $$2^i \leq |\sigma_t| < 2^{i+1}.$$
  - For any phase t, the extents of the active suffixes *do not overlap*...
  - ...and phase t has at most n/σ_t *active suffixes*.
  - Therefore, *the cost of each macro-phase is* $O(n)$ and the final $O(n \log n)$ bound follows immediately.

Suffix selection, second attempt: exploiting collisions

Suffix selection, second attempt: exploiting collisions

- So, while we improved the $O\left(n^2\right)$ of the first, simple algorithm...

Suffix selection, second attempt: exploiting collisions

- So, while we improved the $O\left(n^2\right)$ of the first, simple algorithm...
- ...*we have not yet answered our original question*

$$Comp(\text{Suffix Sorting}) \neq Comp(\text{Suffix Selection})?,$$

since *Suffix Sorting can be done in* $O\left(n \log n\right)$ *time* as well.

Suffix selection, second attempt: exploiting collisions

- So, while we improved the $O\left(n^2\right)$ of the first, simple algorithm...
- ...*we have not yet answered our original question*

$$Comp(\text{Suffix Sorting}) \neq Comp(\text{Suffix Selection})?,$$

since *Suffix Sorting can be done in* $O\left(n \log n\right)$ *time* as well.

We need one last step:

> *We have to be able to reuse the work done on inactive suffixes.*

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

The first two solutions have one aspect in common:

Suffix selection, third attempt: reusing inactive suffixes

The first two solutions have one aspect in common:

> *They do not fully exploit the available information about inactive suffixes (i.e. their extents).*

Suffix selection, third attempt: reusing inactive suffixes

The first two solutions have one aspect in common:

> *They do not fully exploit the available information about inactive suffixes (i.e. their extents).*

- The central issue is *how much the extent of an active suffix T_i in phase $t+1$ is enlarged during the transition from phase t*.

Suffix selection, third attempt: reusing inactive suffixes

The first two solutions have one aspect in common:

They do not fully exploit the available information about inactive suffixes (i.e. their extents).

- The central issue is *how much the extent of an active suffix T_i in phase $t+1$ is enlarged during the transition from phase t*.
- What do we add to the extent of T_i in the second solution?

Suffix selection, third attempt: reusing inactive suffixes

The first two solutions have one aspect in common:

> *They do not fully exploit the available information about inactive suffixes (i.e. their extents).*

- The central issue is *how much the extent of an active suffix T_i in phase $t+1$ is enlarged during the transition from phase t*.
- What do we add to the extent of T_i in the second solution?
  (a) *All the extents of the active suffixes that follow T_i and collide with it.*

Suffix selection, third attempt: reusing inactive suffixes

The first two solutions have one aspect in common:

They do not fully exploit the available information about inactive suffixes (i.e. their extents).

- The central issue is *how much the extent of an active suffix T_i in phase $t+1$ is enlarged during the transition from phase t*.
- What do we add to the extent of T_i in the second solution?
  - (a) *All the extents of the active suffixes that follow T_i and collide with it.*
  - (b) *The element c next to the extent of the rightmost suffix in the collision.*

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

This “limited” way to enlarge the extents *implies that* c *may be accessed again* $\omega(1)$ *times* in the subsequent phase transitions:

Suffix selection, third attempt: reusing inactive suffixes

This “limited” way to enlarge the extents *implies that c may be accessed again $\omega(1)$ times* in the subsequent phase transitions:

(1) T_i can later *become inactive*, c can then be *accessed again* and added to the extent of *another active suffix $T_{i'}$*.

Suffix selection, third attempt: reusing inactive suffixes

This “limited” way to enlarge the extents *implies that c may be accessed again $\omega(1)$ times* in the subsequent phase transitions:

(1) T_i can later *become inactive*, c can then be *accessed again* and added to the extent of *another active suffix* $T_{i'}$.

(2) $T_{i'}$ can then *become inactive in its turn*, c can later be *accessed once again* and added to $T_{i''}$.

Suffix selection, third attempt: reusing inactive suffixes

This “limited” way to enlarge the extents *implies that c may be accessed again $\omega(1)$ times* in the subsequent phase transitions:

(1) T_i can later *become inactive*, c can then be *accessed again* and added to the extent of *another active suffix* $T_{i'}$.

(2) $T_{i'}$ can then *become inactive in its turn*, c can later be *accessed once again* and added to $T_{i''}$.

. . . .

Suffix selection, third attempt: reusing inactive suffixes

This “limited” way to enlarge the extents *implies that* c *may be accessed again* $\omega(1)$ *times* in the subsequent phase transitions:

(1) T_i can later *become inactive*, c can then be *accessed again* and added to the extent of *another active suffix* $T_{i'}$.

(2) $T_{i'}$ can then *become inactive in its turn*, c can later be *accessed once again* and added to $T_{i''}$.

. . . .

(?) Over time, *this causes the extra* $\log n$ *factor* in the complexity bound of the second solution.

Suffix selection, third attempt: reusing inactive suffixes

This “limited” way to enlarge the extents *implies that c may be accessed again $\omega(1)$ times* in the subsequent phase transitions:

(1) T_i can later *become inactive*, c can then be *accessed again* and added to the extent of *another active suffix* $T_{i'}$.

(2) $T_{i'}$ can then *become inactive in its turn*, c can later be *accessed once again* and added to $T_{i''}$.

. . . .

(?) Over time, *this causes the extra $\log n$ factor* in the complexity bound of the second solution.

The challenge now is to avoid these multiple accesses.

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

Let's consider an *active suffix* T_i *in phase* t that remains active after the *transition to phase* $t+1$.

Suffix selection, third attempt: reusing inactive suffixes

Let's consider an *active suffix* T_i *in phase* t that remains active after the *transition to phase* $t+1$.

> *The forward suffix of a suffix* T_j
> is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

Suffix selection, third attempt: reusing inactive suffixes

Let's consider an *active suffix* T_i *in phase* t that remains active after the *transition to phase* $t+1$.

> *The forward suffix of a suffix* T_j
> is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

In the third solution the *prospective extent* of T_i is composed by the following:

Suffix selection, third attempt: reusing inactive suffixes

Let's consider an *active suffix* T_i *in phase* t that remains active after the *transition to phase* $t + 1$.

> *The forward suffix of a suffix* T_j
> is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

In the third solution the *prospective extent* of T_i is composed by the following:

(a) *All the extents of the active suffixes following* T_i *and colliding with it*

Suffix selection, third attempt: reusing inactive suffixes

Let's consider an *active suffix* T_i *in phase* t that remains active after the *transition to phase* $t+1$.

> *The forward suffix of a suffix* T_j
> is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

In the third solution the *prospective extent* of T_i is composed by the following:

(a) *All the extents of the active suffixes following* T_i *and colliding with it* (although they don't collide as nicely as in the second solution, as we will see).

Suffix selection, third attempt: reusing inactive suffixes

Let's consider an *active suffix* T_i *in phase* t that remains active after the *transition to phase* $t + 1$.

> *The forward suffix of a suffix* T_j
> is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

In the third solution the *prospective extent* of T_i is composed by the following:

(a) *All the extents of the active suffixes following* T_i *and colliding with it* (although they don't collide as nicely as in the second solution, as we will see).

(b) *The extent of the forward suffix* f *of the rightmost suffix* r *in collision with* T_i (r is T_i itself if T_i is not in a collision).

Suffix selection, third attempt: reusing inactive suffixes

Let's consider an *active suffix* T_i *in phase* t that remains active after the *transition to phase* $t+1$.

> *The forward suffix of a suffix* T_j
>
> is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

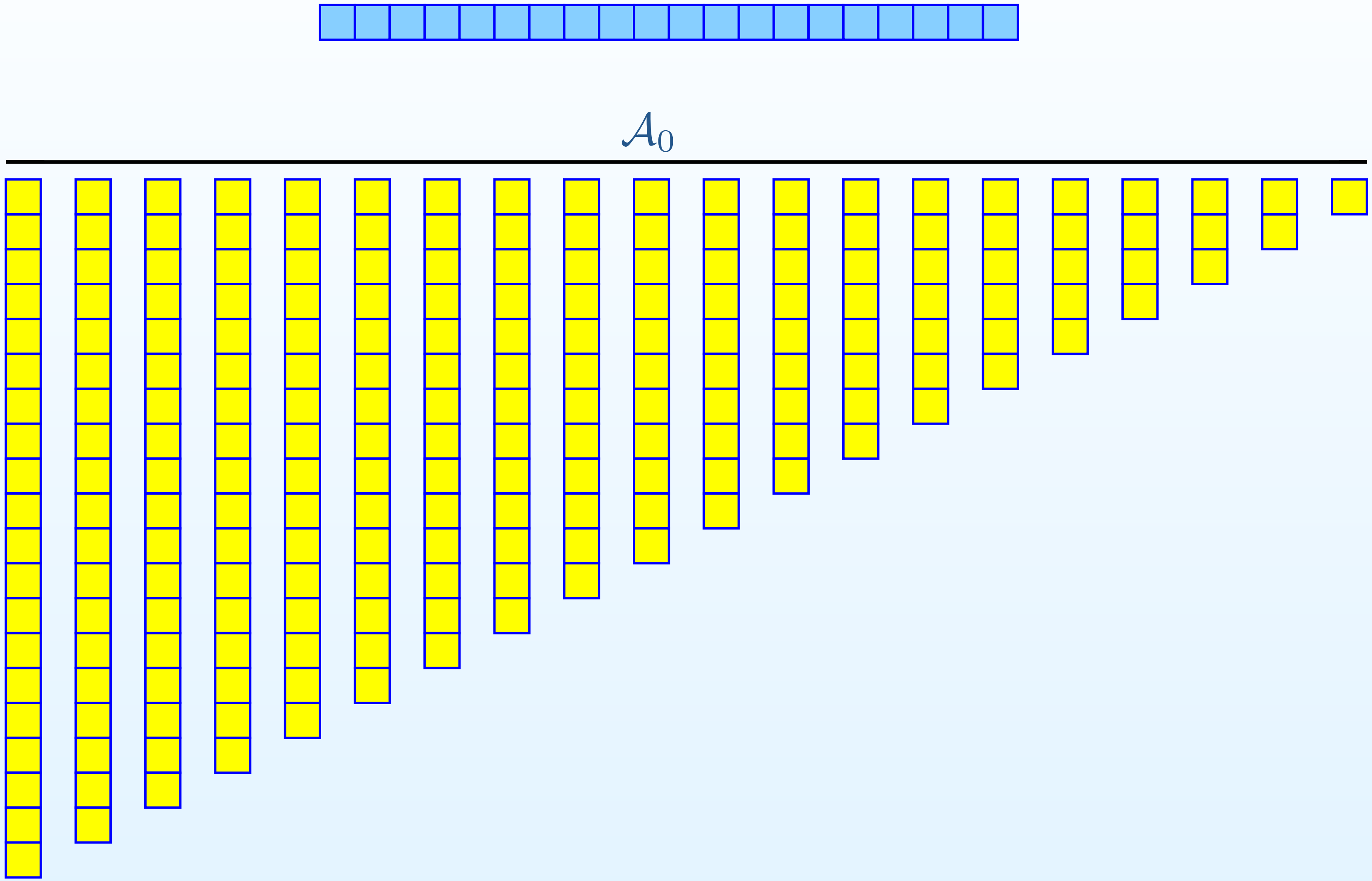
In the third solution the *prospective extent* of T_i is composed by the following:

(a) *All the extents of the active suffixes following* T_i *and colliding with it* (although they don't collide as nicely as in the second solution, as we will see).

(b) *The extent of the forward suffix* f *of the rightmost suffix* r *in collision with* T_i (r is T_i itself if T_i is not in a collision).

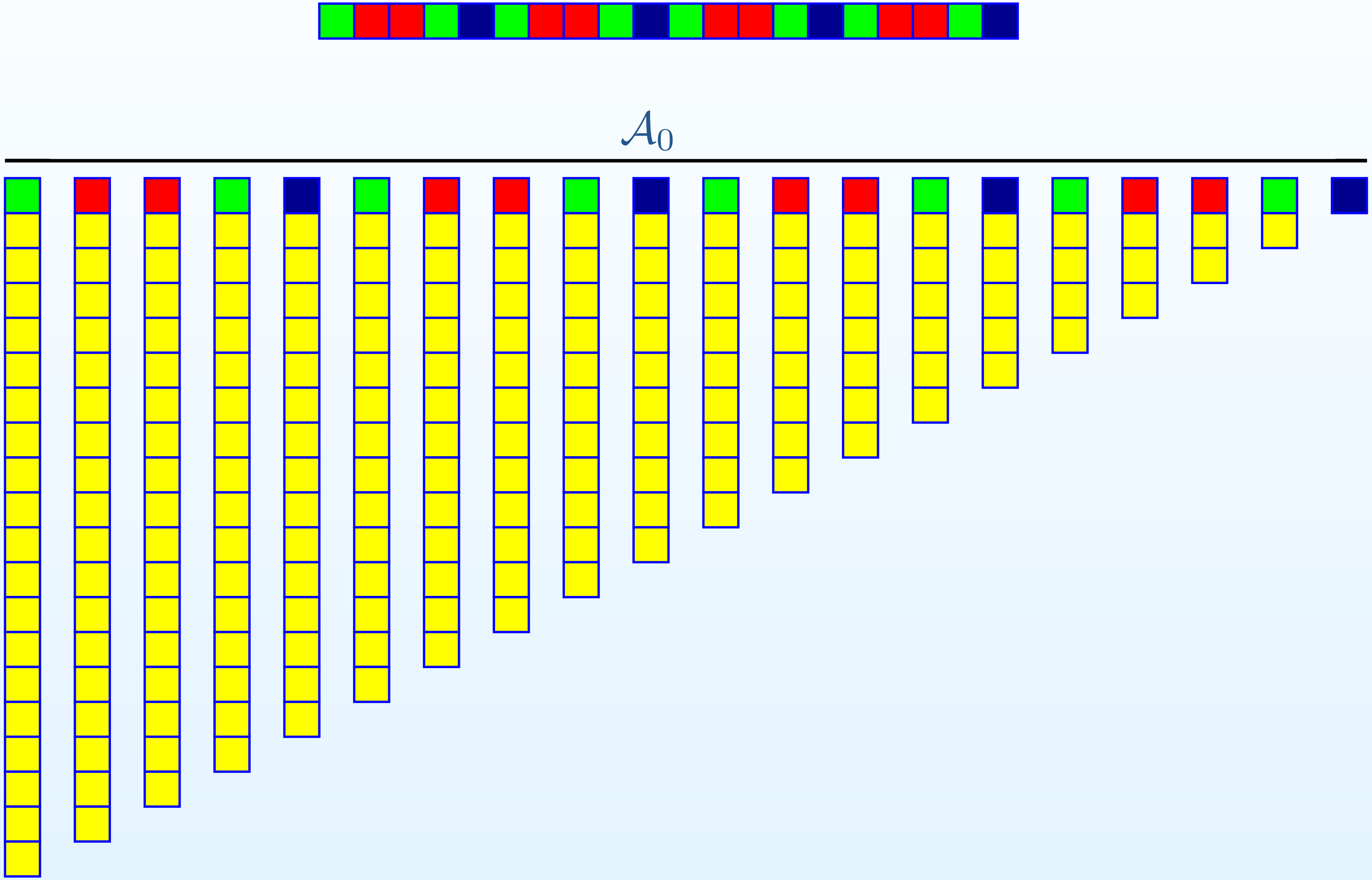
(c) *The element* c *next to the extent of* f

Suffix selection, third attempt: reusing inactive suffixes



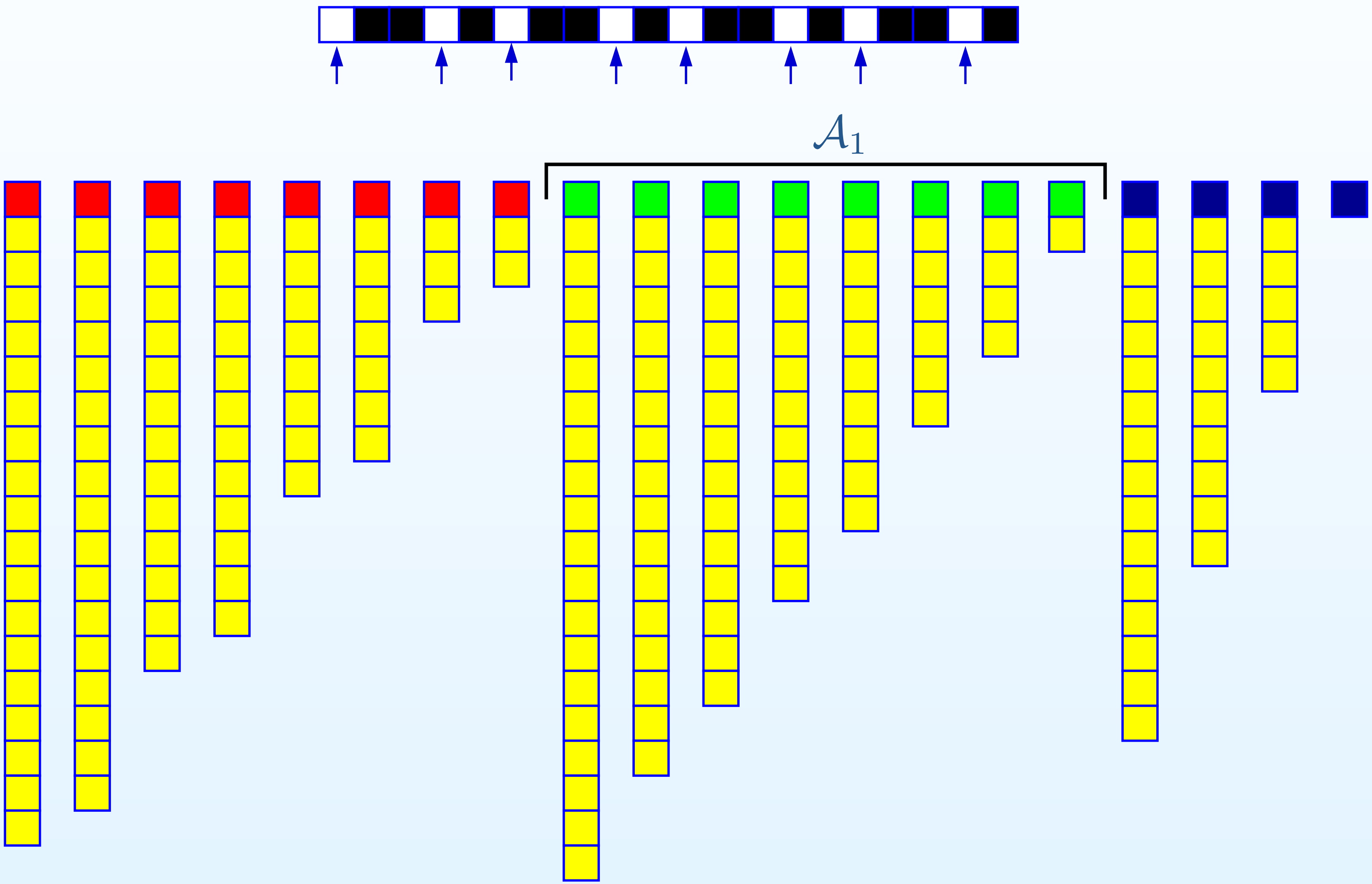
$k = 10$, $l_0 = 0$, Phase 0

Suffix selection, third attempt: reusing inactive suffixes



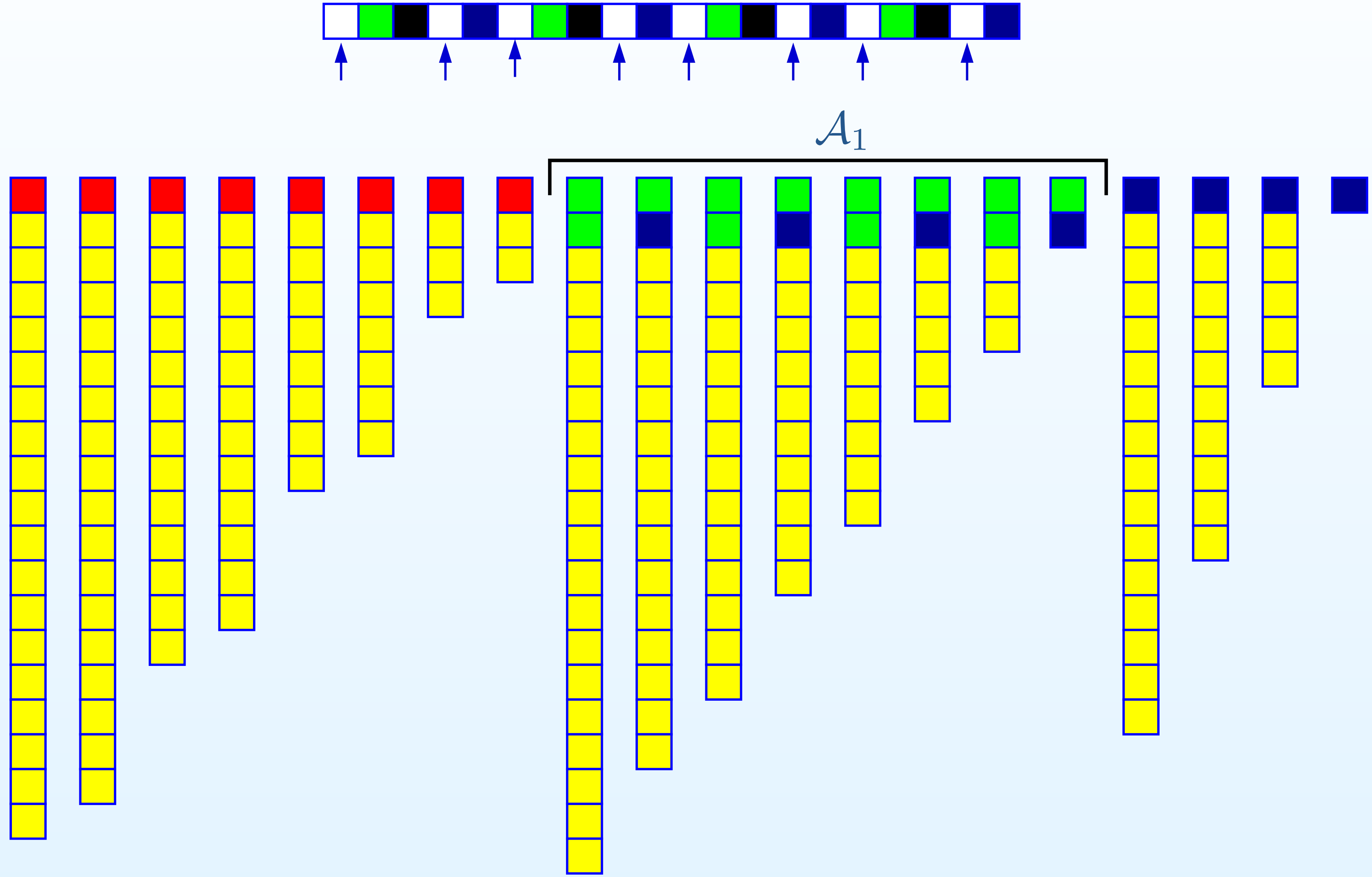
$k = 10$, $l_0 = 0$, Phase 0

Suffix selection, third attempt: reusing inactive suffixes



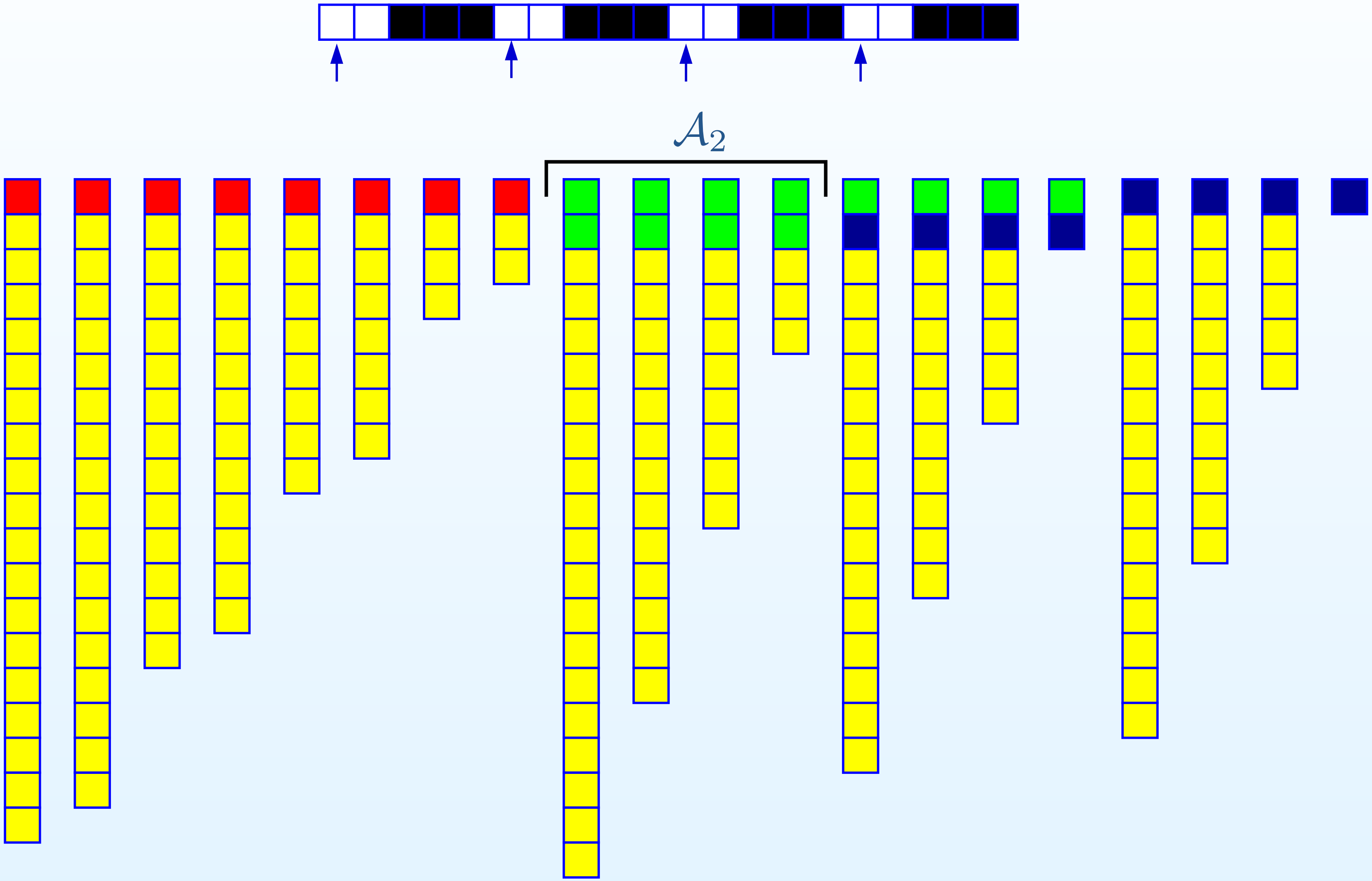
$k = 10$, $l_1 = 8$, Phase 1

Suffix selection, third attempt: reusing inactive suffixes



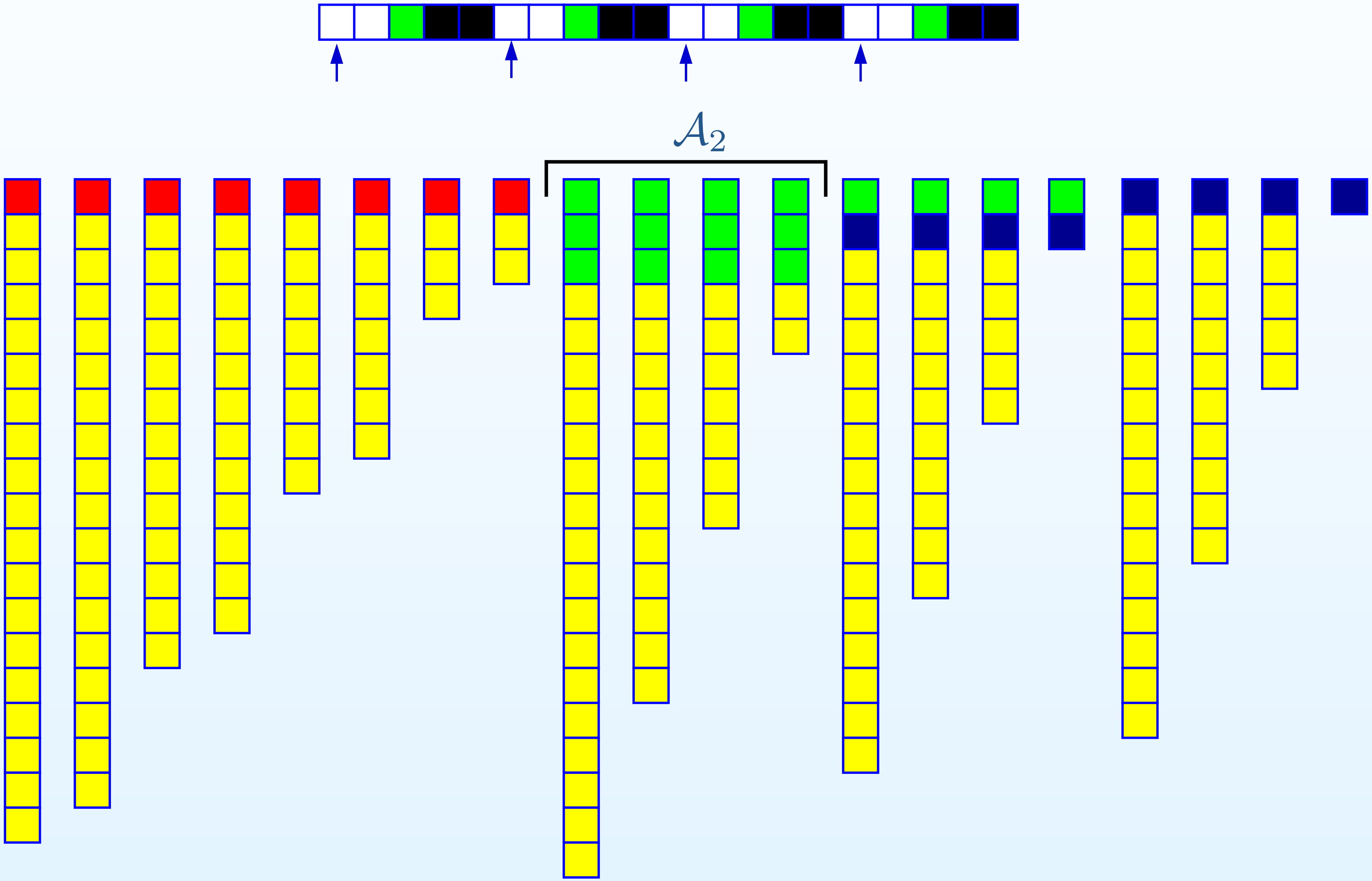
$k = 10$, $l_1 = 8$, Phase 1

Suffix selection, third attempt: reusing inactive suffixes



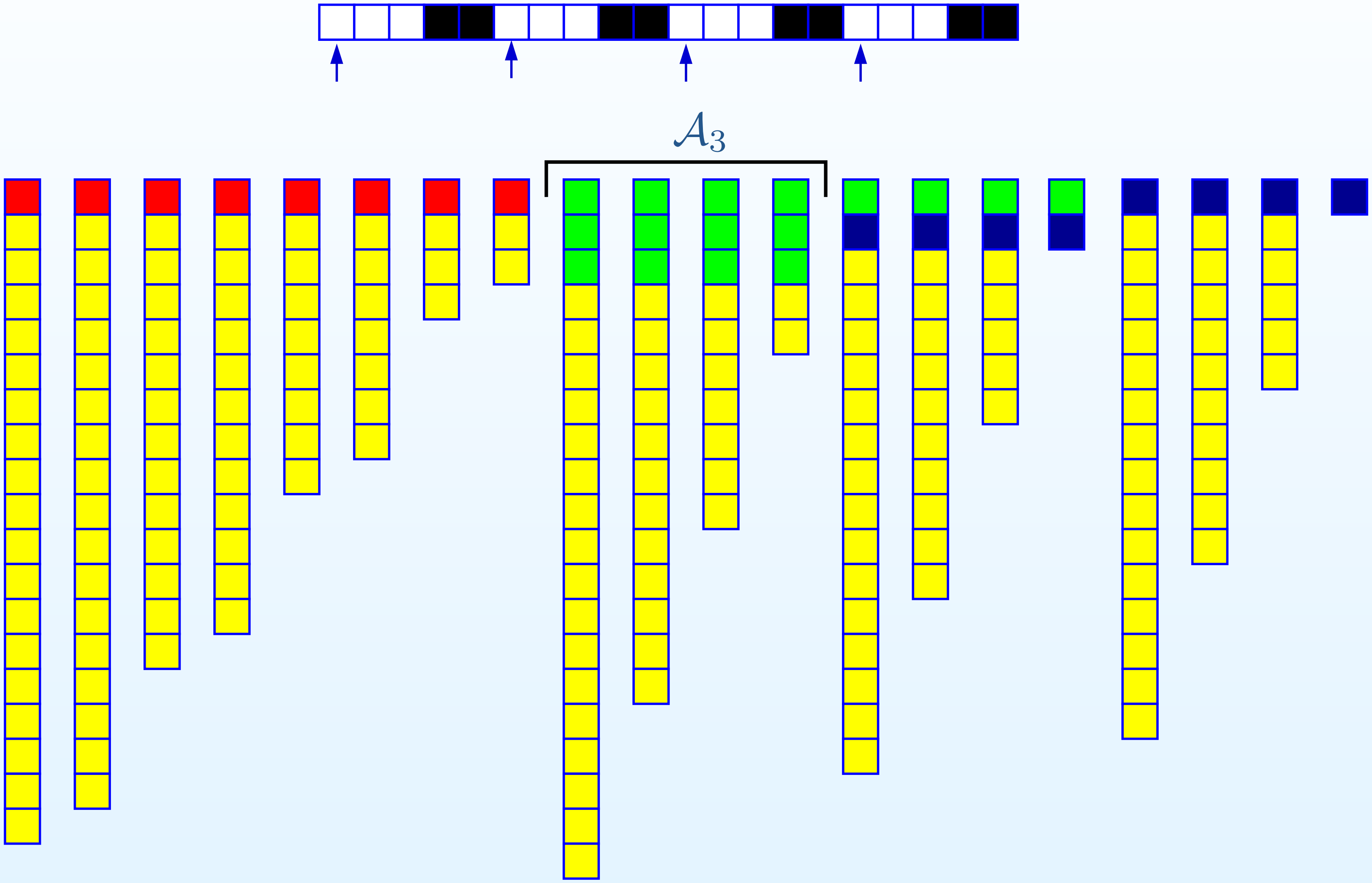
$k = 10$, $l_2 = 8$, Phase 2

Suffix selection, third attempt: reusing inactive suffixes



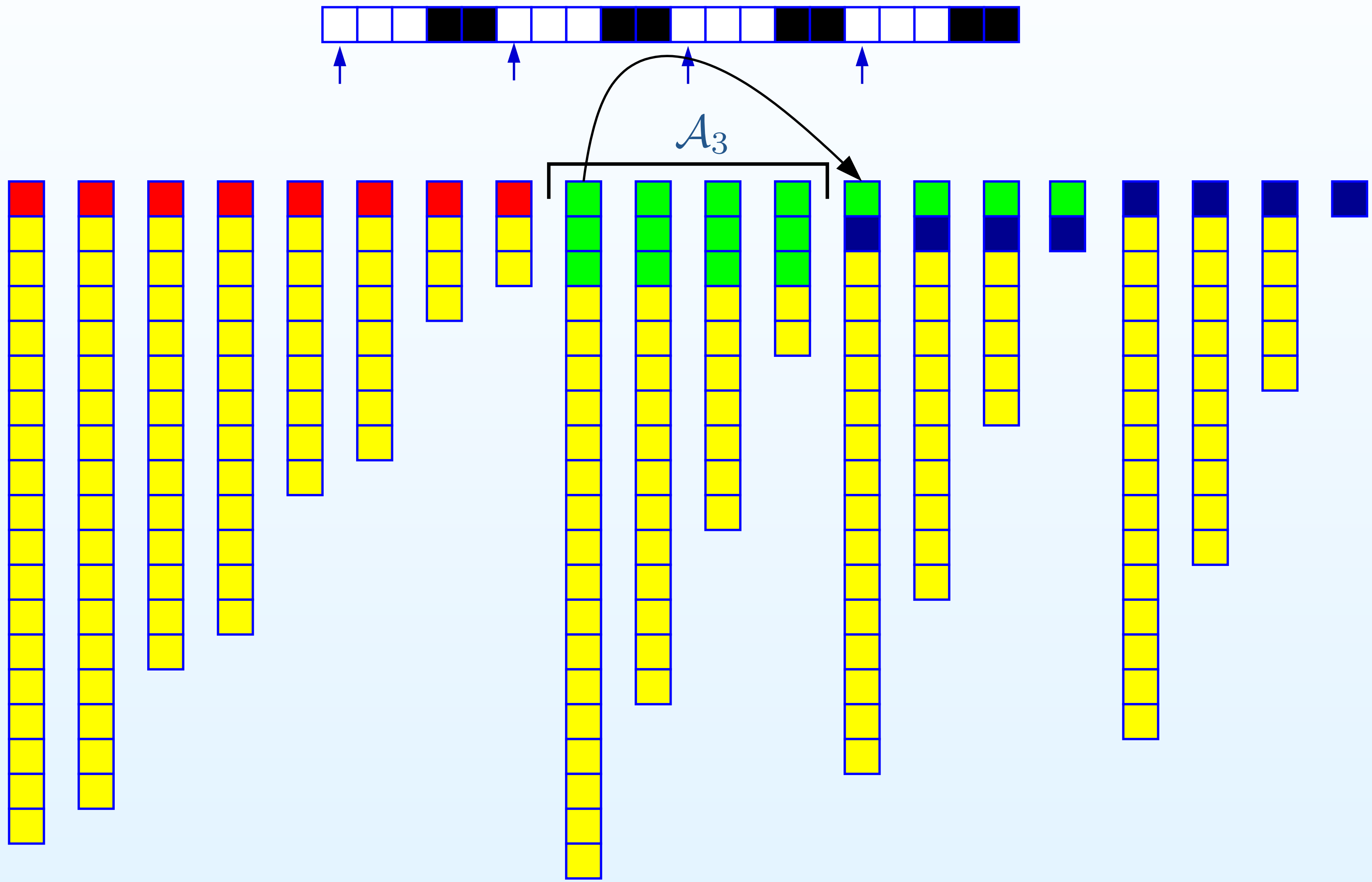
$k = 10$, $l_2 = 8$, Phase 2

Suffix selection, third attempt: reusing inactive suffixes



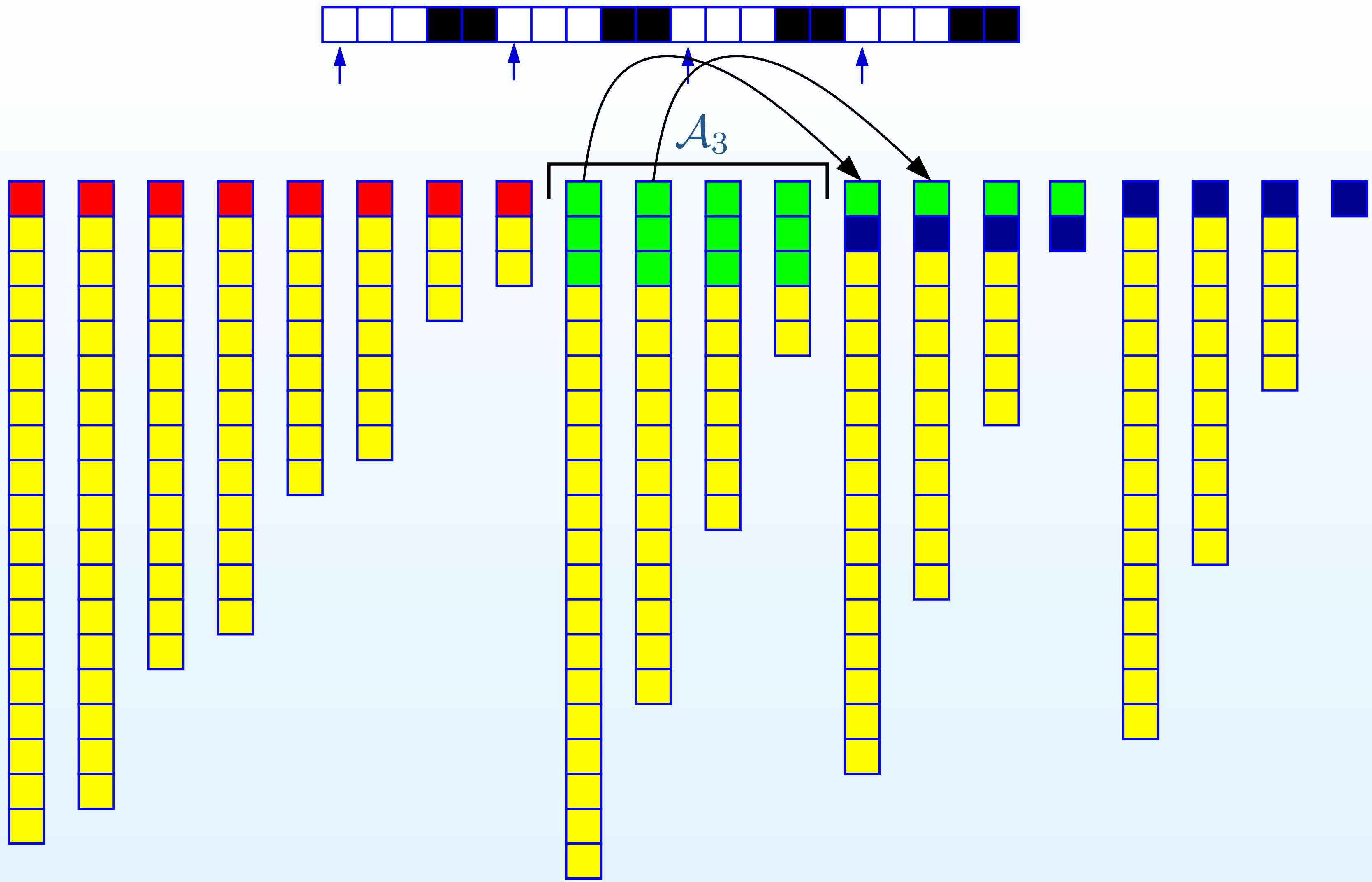
$k = 10$, $l_3 = 8$, Phase 3

Suffix selection, third attempt: reusing inactive suffixes



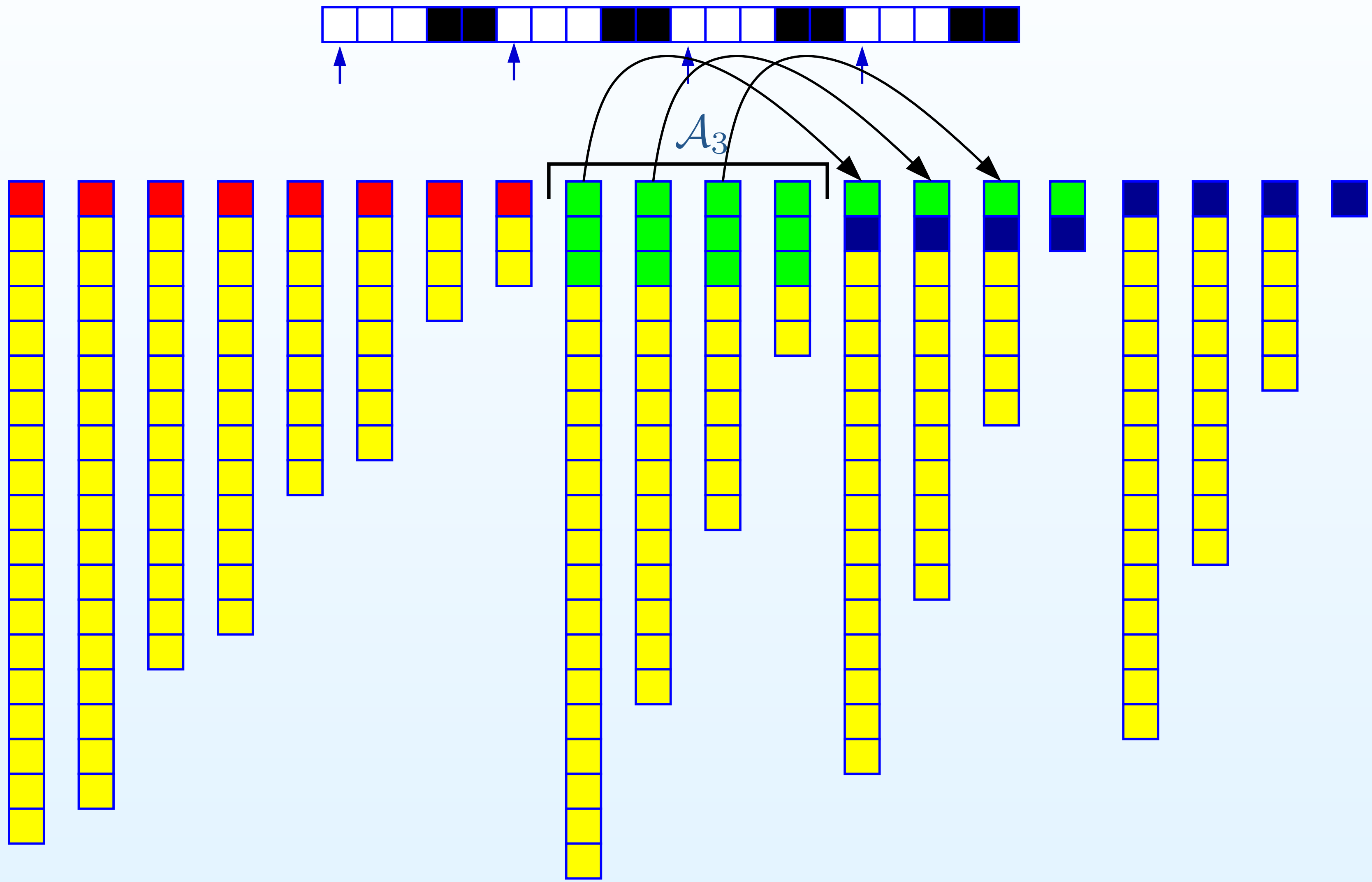
$k = 10$, $l_3 = 8$, Phase 3

Suffix selection, third attempt: reusing inactive suffixes



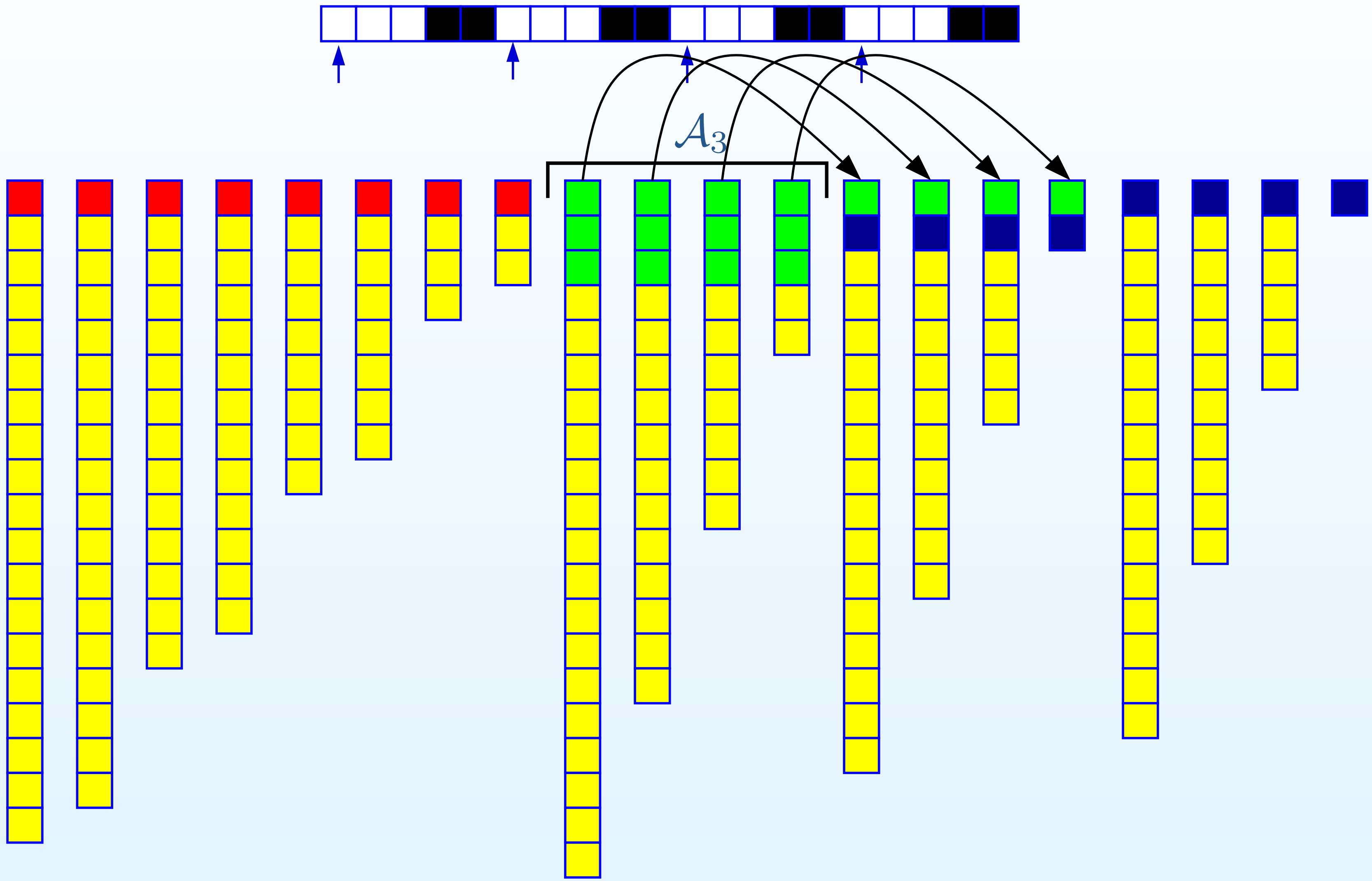
$k = 10$, $l_3 = 8$, Phase 3

Suffix selection, third attempt: reusing inactive suffixes



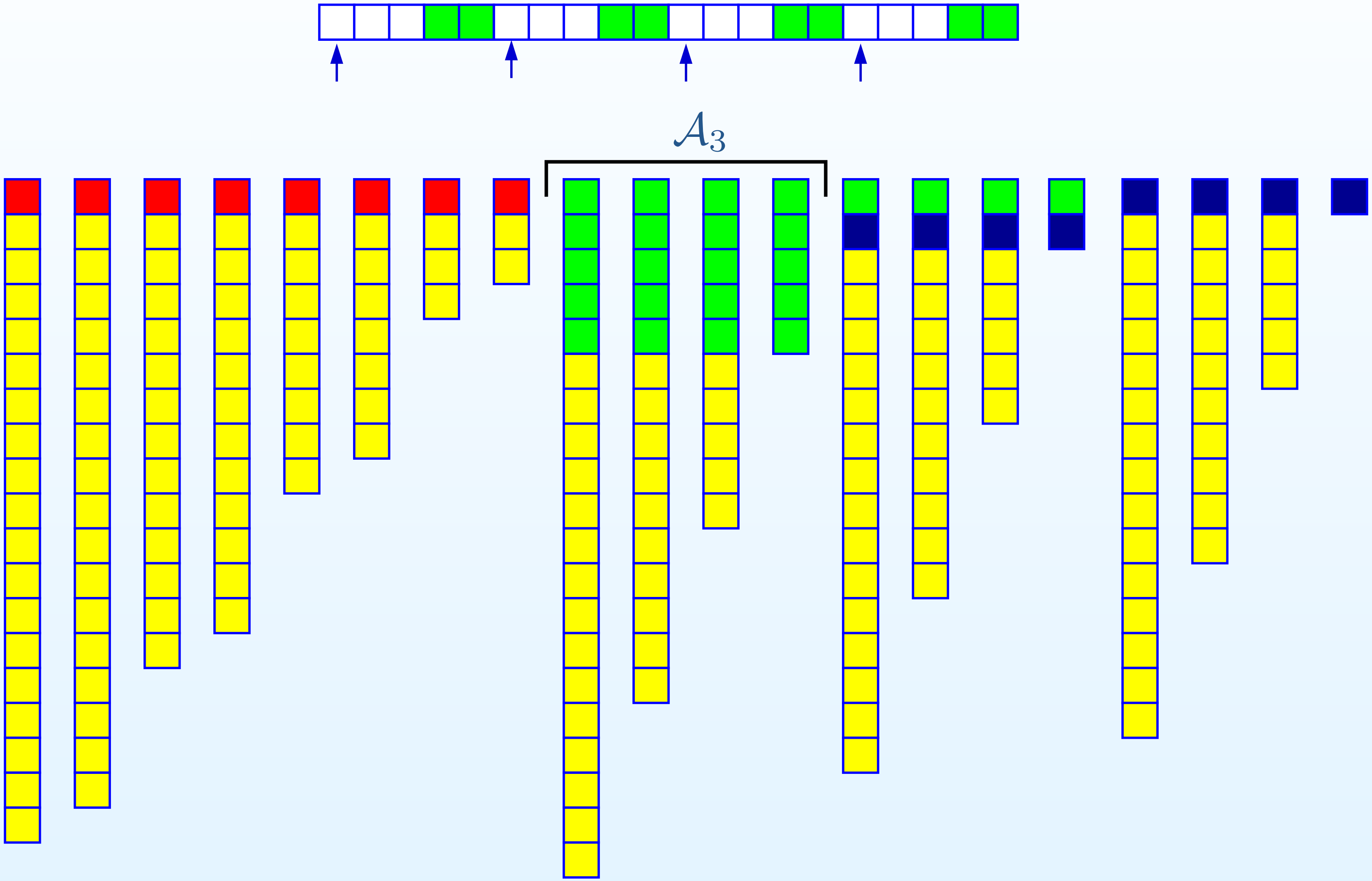
$k = 10$, $l_3 = 8$, Phase 3

Suffix selection, third attempt: reusing inactive suffixes



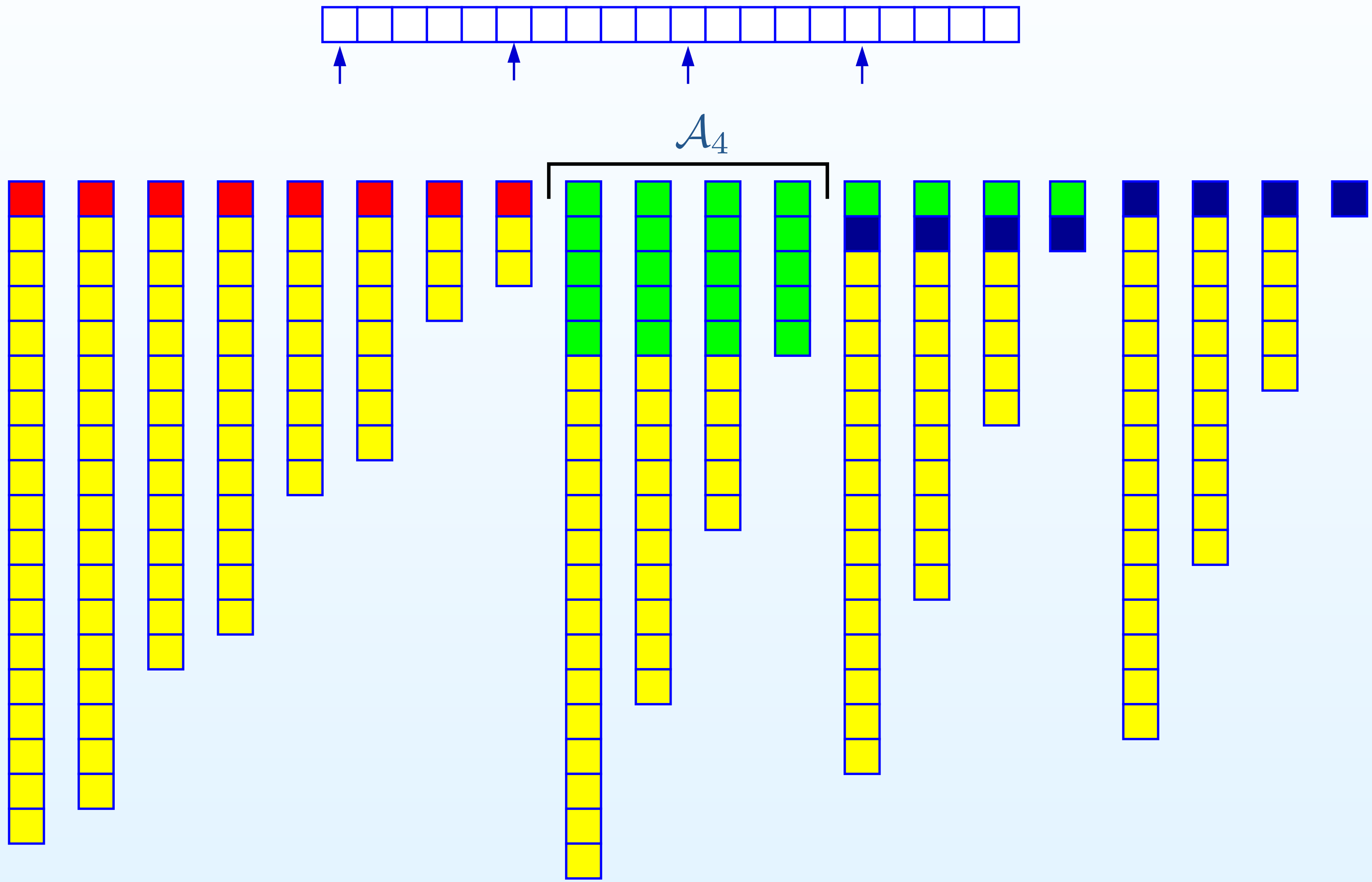
$k = 10$, $l_3 = 8$, Phase 3

Suffix selection, third attempt: reusing inactive suffixes



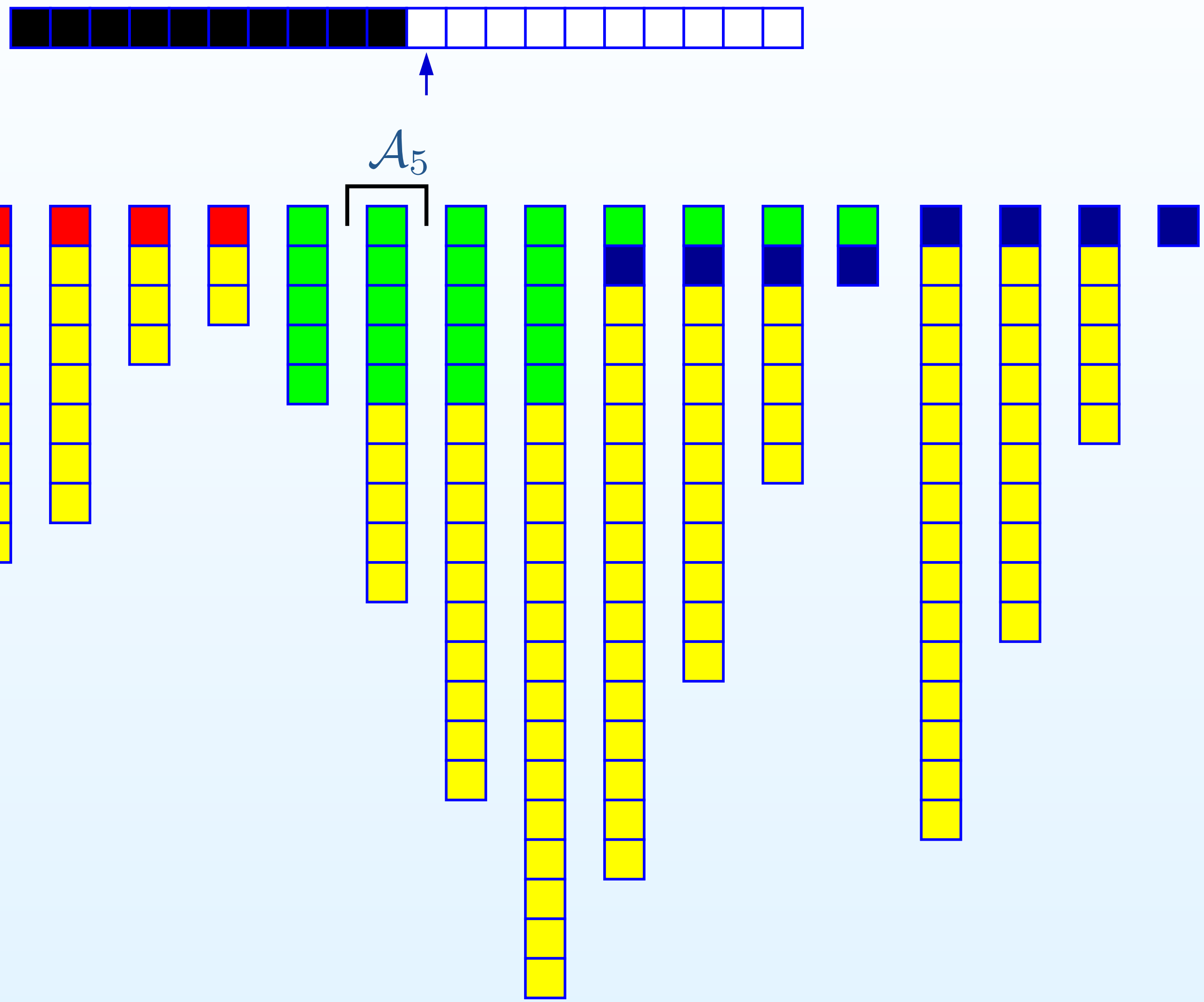
$k = 10$, $l_3 = 8$, Phase 3

Suffix selection, third attempt: reusing inactive suffixes



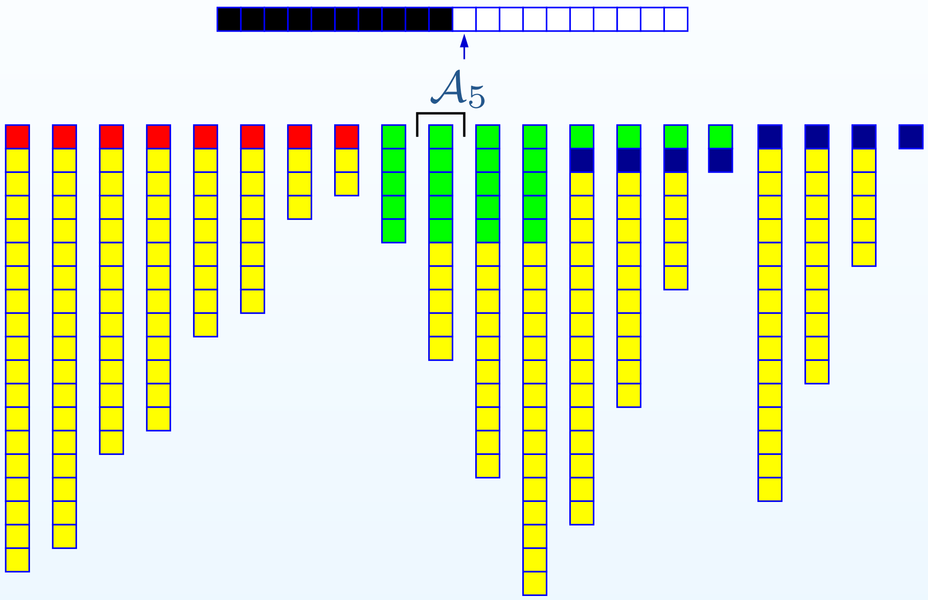
$k = 10$, $l_4 = 8$, Phase 4

Suffix selection, third attempt: reusing inactive suffixes



$k = 10$, $l_5 = 9$, Phase 5

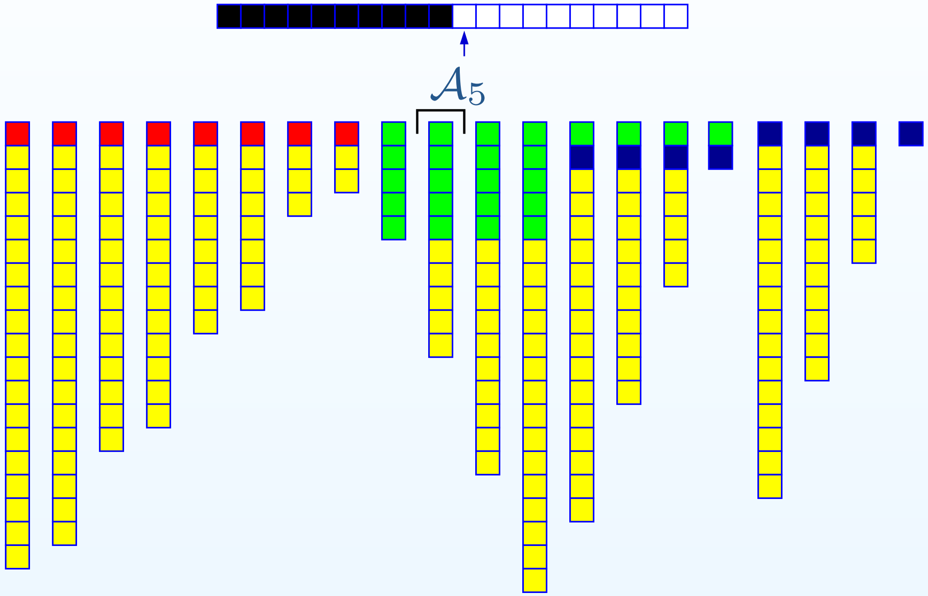
Suffix selection, third attempt: reusing inactive suffixes



$k = 10$, $l_5 = 9$, Phase 5

- By reusing the work done on inactive suffixes, *the computation ended in* 5 *phases*...

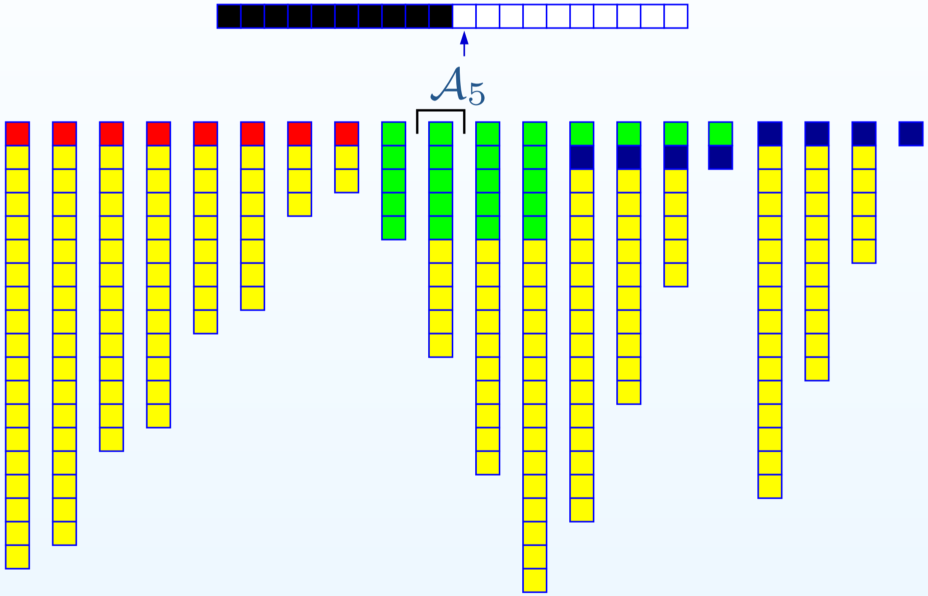
Suffix selection, third attempt: reusing inactive suffixes



$k = 10$, $l_5 = 9$, Phase 5

- By reusing the work done on inactive suffixes, *the computation ended in 5 phases*. . . one phase less than the second attempt.

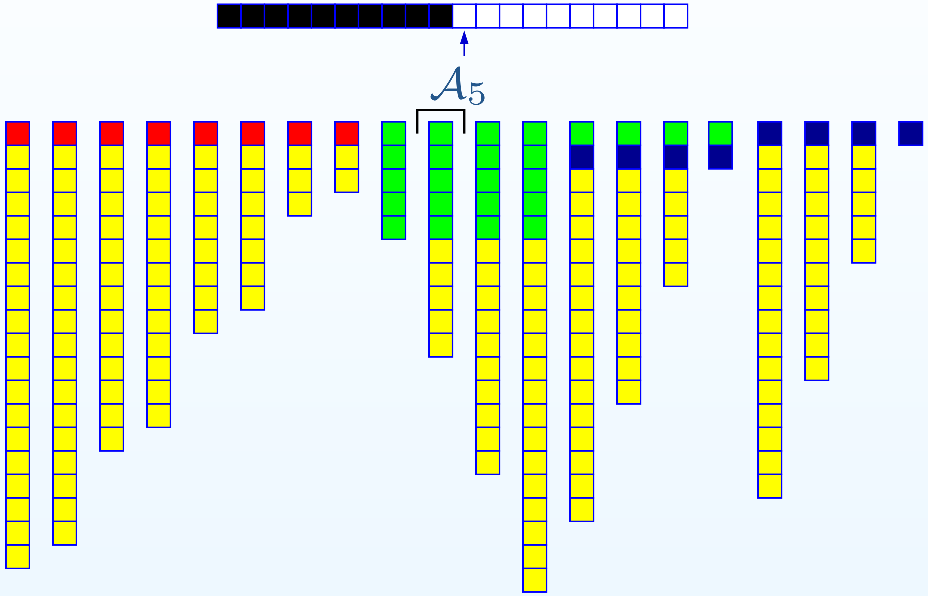
Suffix selection, third attempt: reusing inactive suffixes



$k = 10$, $l_5 = 9$, Phase 5

- By reusing the work done on inactive suffixes, *the computation ended in* 5 *phases*... one phase less than the second attempt.
- This is a particularly *lucky example*...

Suffix selection, third attempt: reusing inactive suffixes



$k = 10$, $l_5 = 9$, Phase 5

- By reusing the work done on inactive suffixes, *the computation ended in 5 phases*… one phase less than the second attempt.
- This is a particularly *lucky example*… in the general case the *exploiting of collisions of active suffixes* and the *reuse of the extents of inactive suffixes* do not play along so nicely, as we will see.

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

(i) Assuming that we are able to *compare prospective extents efficiently* (i.e. in $O(1)$ time).

Suffix selection, third attempt: reusing inactive suffixes

(i) Assuming that we are able to *compare prospective extents efficiently* (i.e. in $O(1)$ time).

(ii) Assuming that we can *find the forward suffixes efficiently* (that is with a total cost $O(n)$ for the entire computation).

Suffix selection, third attempt: reusing inactive suffixes

(i) Assuming that we are able to *compare prospective extents efficiently* (i.e. in $O(1)$ time).

(ii) Assuming that we can *find the forward suffixes efficiently* (that is with a total cost $O(n)$ for the entire computation).

(iii) Assuming that all the above “querying machineries” can be *maintained efficiently* (again, with a total cost $O(n)$).

Suffix selection, third attempt: reusing inactive suffixes

(i) Assuming that we are able to *compare prospective extents efficiently* (i.e. in $O(1)$ time).

(ii) Assuming that we can *find the forward suffixes efficiently* (that is with a total cost $O(n)$ for the entire computation).

(iii) Assuming that all the above "querying machineries" can be *maintained efficiently* (again, with a total cost $O(n)$).

The new way to enlarge extents guarantees that $O(n)$ comparisons are made during the computation.

Suffix selection, third attempt: reusing inactive suffixes

(i) Assuming that we are able to *compare prospective extents efficiently* (i.e. in $O(1)$ time).

(ii) Assuming that we can *find the forward suffixes efficiently* (that is with a total cost $O(n)$ for the entire computation).

(iii) Assuming that all the above “querying machineries” can be *maintained efficiently* (again, with a total cost $O(n)$).

> *The new way to enlarge extents guarantees that $O(n)$ comparisons are made during the computation.*

- An element c of T *will not be accessed again once it is inside an extent* (i.e. c is not in the rightmost position of the extent).

Suffix selection, third attempt: reusing inactive suffixes

(i) Assuming that we are able to *compare prospective extents efficiently* (i.e. in $O(1)$ time).

(ii) Assuming that we can *find the forward suffixes efficiently* (that is with a total cost $O(n)$ for the entire computation).

(iii) Assuming that all the above "querying machineries" can be *maintained efficiently* (again, with a total cost $O(n)$).

> *The new way to enlarge extents guarantees that $O(n)$ comparisons are made during the computation.*

- An element c of T *will not be accessed again once it is inside an extent* (i.e. c is not in the rightmost position of the extent).
- As long as an element c of T is in the rightmost position of an extent, *there can be multiple accesses to it*...

Suffix selection, third attempt: reusing inactive suffixes

(i) Assuming that we are able to *compare prospective extents efficiently* (i.e. in $O(1)$ time).

(ii) Assuming that we can *find the forward suffixes efficiently* (that is with a total cost $O(n)$ for the entire computation).

(iii) Assuming that all the above “querying machineries” can be *maintained efficiently* (again, with a total cost $O(n)$).

The new way to enlarge extents guarantees that $O(n)$ comparisons are made during the computation.

- An element c of T *will not be accessed again once it is inside an extent* (i.e. c is not in the rightmost position of the extent).
- As long as an element c of T is in the rightmost position of an extent, *there can be multiple accesses to it*...
- ...but any of those accesses to c *can be charged on an active suffix becoming inactive* during the current phase transition.

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

Let’s deal with the prospective extents first.

Suffix selection, third attempt: reusing inactive suffixes

Let's deal with the prospective extents first.

During any phase t, two suffixes T_i, T_j *collide* when their *extents are either adjacent or overlapping*.

Suffix selection, third attempt: reusing inactive suffixes

Let's deal with the prospective extents first.

> During any phase t, two suffixes T_i, T_j *collide* when their *extents are either adjacent or overlapping*.

In the second solution the extents of colliding suffixes are *always adjacent*.

Suffix selection, third attempt: reusing inactive suffixes

Let's deal with the prospective extents first.

> During any phase t, two suffixes T_i, T_j *collide* when their *extents are either adjacent or overlapping*.

In the second solution the extents of colliding suffixes are *always adjacent*.

- The prospective extent of T_i has a simple periodic form: $(\sigma_t)^{r_i} c$.

Suffix selection, third attempt: reusing inactive suffixes

Let's deal with the prospective extents first.

> During any phase t, two suffixes T_i, T_j *collide* when their *extents are either adjacent or overlapping*.

In the second solution the extents of colliding suffixes are *always adjacent*.

- The prospective extent of T_i has a simple periodic form: $(\sigma_t)^{r_i} c$.
- Thus, it can be represented by the pair integer/element (r_i, c), *no matter how many suffixes are in collision with* T_i.

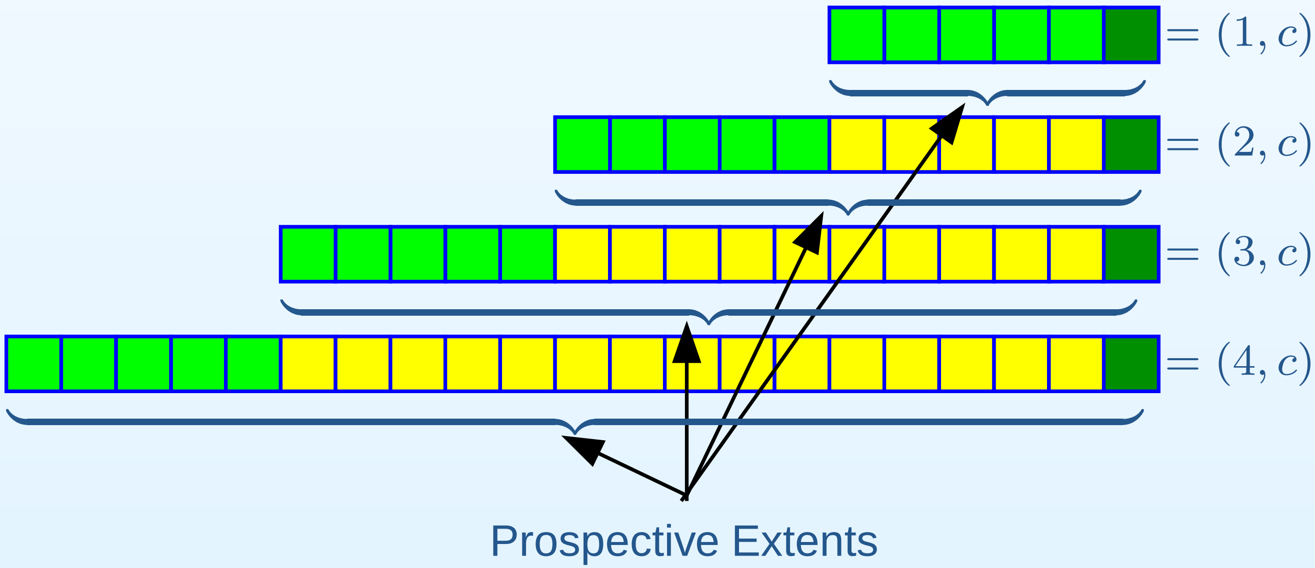
Suffix selection, third attempt: reusing inactive suffixes

Let's deal with the prospective extents first.

> During any phase t, two suffixes T_i, T_j *collide* when their *extents are either adjacent or overlapping*.

In the second solution the extents of colliding suffixes are *always adjacent*.

- The prospective extent of T_i has a simple periodic form: $(\sigma_t)^{r_i}c$.
- Thus, it can be represented by the pair integer/element (r_i, c), *no matter how many suffixes are in collision with T_i*.



Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

In the third solution the extents of colliding suffixes *overlap in generic ways*.

Suffix selection, third attempt: reusing inactive suffixes

In the third solution the extents of colliding suffixes *overlap in generic ways*.

- The prospective extent of T_i is a sequence $\sigma_t w_{d_1} w_{d_2} \ldots w_{d_{l-1}} w u_{d_l} c$

Suffix selection, third attempt: reusing inactive suffixes

In the third solution the extents of colliding suffixes *overlap in generic ways*.

- The prospective extent of T_i is a sequence $\sigma_t w_{d_1} w_{d_2} \ldots w_{d_{l-1}} w u_{d_l} c$
  - w_{d_j} is the d_j*-th suffix of* σ_t,
  - u_{d_l} is the d_l*-th suffix of the extent of the forward suffix*,
  - c is the *element following* u_{d_l} in T.

Suffix selection, third attempt: reusing inactive suffixes

In the third solution the extents of colliding suffixes *overlap in generic ways*.

- The prospective extent of T_i is a sequence $\sigma_t w_{d_1} w_{d_2} \dots w_{d_{l-1}} w u_{d_l} c$
  - w_{d_j} is the d_j*-th suffix of* σ_t,
  - u_{d_l} is the d_l*-th suffix of the extent of the forward suffix*,
  - c is the *element following* u_{d_l} in T.
- The overlapping is *limited*: $1 \leq w_{d_j} \leq |\sigma_t| / 2$.

Suffix selection, third attempt: reusing inactive suffixes

In the third solution the extents of colliding suffixes *overlap in generic ways*.

- The prospective extent of T_i is a sequence $\sigma_t w_{d_1} w_{d_2} \ldots w_{d_{l-1}} w u_{d_l} c$
  - w_{d_j} is the d_j*-th suffix of* σ_t,
  - u_{d_l} is the d_l*-th suffix of the extent of the forward suffix*,
  - c is the *element following* u_{d_l} in T.
- The overlapping is *limited*: $1 \leq w_{d_j} \leq |\sigma_t| / 2$.
- The pros. ext. must be represented by l *integers* and *one element*: $(d_1, \ldots, d_l, c)$.

Suffix selection, third attempt: reusing inactive suffixes

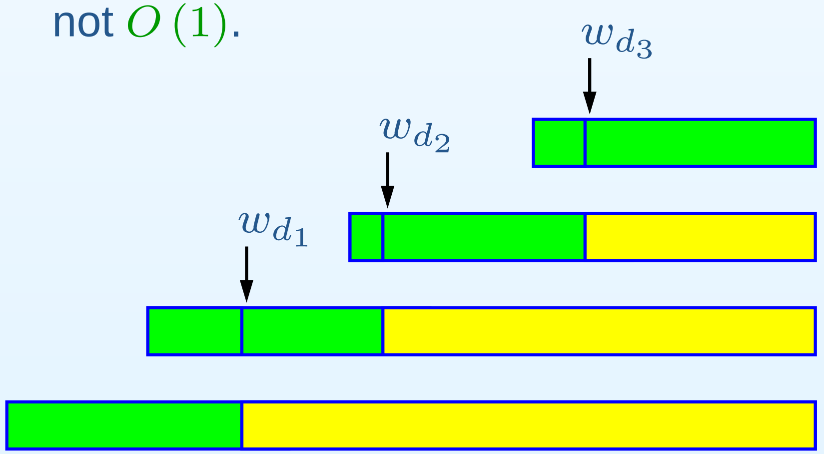
In the third solution the extents of colliding suffixes *overlap in generic ways*.

- The prospective extent of T_i is a sequence $\sigma_t w_{d_1} w_{d_2} \ldots w_{d_{l-1}} w u_{d_l} c$
  - w_{d_j} is the d_j*-th suffix of* σ_t,
  - u_{d_l} is the d_l*-th suffix of the extent of the forward suffix*,
  - c is the *element following* u_{d_l} in T.
- The overlapping is *limited*: $1 \leq w_{d_j} \leq |\sigma_t| /2$.
- The pros. ext. must be represented by l *integers* and *one element*: $(d_1, \ldots, d_l, c)$.
- l *is the number of suffixes following* T_i *in the collision plus the forward suffix* and is not $O(1)$.

Suffix selection, third attempt: reusing inactive suffixes

In the third solution the extents of colliding suffixes *overlap in generic ways*.

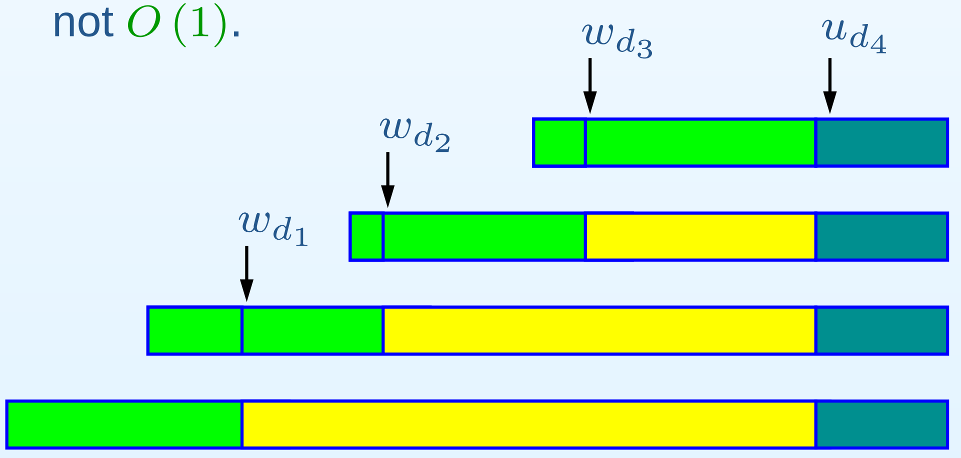
- The prospective extent of T_i is a sequence $\sigma_t w_{d_1} w_{d_2} \ldots w_{d_{l-1}} w u_{d_l} c$
  - w_{d_j} is the d_j*-th suffix of* σ_t,
  - u_{d_l} is the d_l*-th suffix of the extent of the forward suffix*,
  - c is the *element following* u_{d_l} in T.
- The overlapping is *limited*: $1 \leq w_{d_j} \leq |\sigma_t| / 2$.
- The pros. ext. must be represented by l *integers* and *one element*: $(d_1, \ldots, d_l, c)$.
- l *is the number of suffixes following* T_i *in the collision plus the forward suffix* and is not $O(1)$.



Suffix selection, third attempt: reusing inactive suffixes

In the third solution the extents of colliding suffixes *overlap in generic ways*.

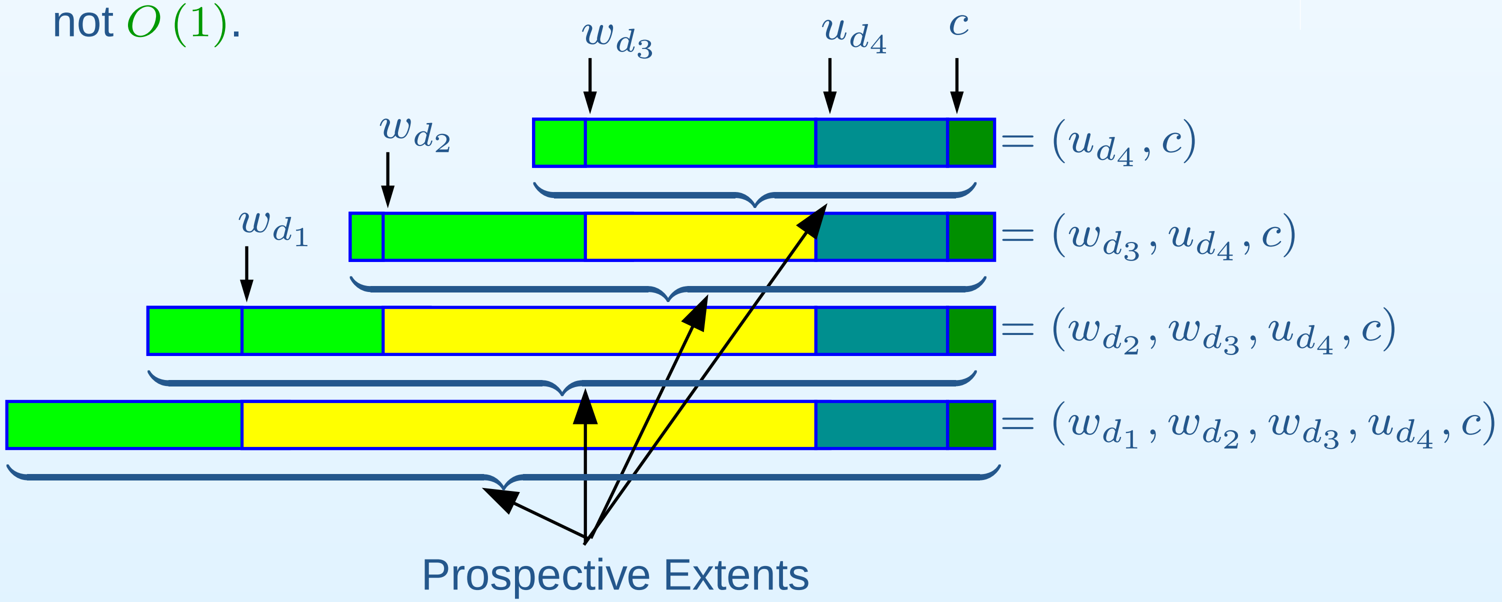
- The prospective extent of T_i is a sequence $\sigma_t w_{d_1} w_{d_2} \ldots w_{d_{l-1}} w u_{d_l} c$
  - w_{d_j} is the d_j*-th suffix of* σ_t,
  - u_{d_l} is the d_l*-th suffix of the extent of the forward suffix*,
  - c is the *element following* u_{d_l} in T.
- The overlapping is *limited*: $1 \leq w_{d_j} \leq |\sigma_t| / 2$.
- The pros. ext. must be represented by l *integers* and *one element*: $(d_1, \ldots, d_l, c)$.
- *l is the number of suffixes following T_i in the collision plus the forward suffix* and is not $O(1)$.



Suffix selection, third attempt: reusing inactive suffixes

In the third solution the extents of colliding suffixes *overlap in generic ways*.

- The prospective extent of T_i is a sequence $\sigma_t w_{d_1} w_{d_2} \ldots w_{d_{l-1}} w u_{d_l} c$
  - w_{d_j} is the d_j*-th suffix of* σ_t,
  - u_{d_l} is the d_l*-th suffix of the extent of the forward suffix*,
  - c is the *element following* u_{d_l} in T.
- The overlapping is *limited*: $1 \leq w_{d_j} \leq |\sigma_t| / 2$.
- The pros. ext. must be represented by l *integers* and *one element*: $(d_1, \ldots, d_l, c)$.
- l *is the number of suffixes following* T_i *in the collision plus the forward suffix* and is not $O(1)$.



Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

We have the following problem to solve *for any phase* t:

Suffix selection, third attempt: reusing inactive suffixes

We have the following problem to solve *for any phase* t:

- We have q_t sequences of *integers* $G_1, \ldots, G_{q_t}$:

Suffix selection, third attempt: reusing inactive suffixes

We have the following problem to solve *for any phase* t:

- We have q_t sequences of *integers* $G_1, \ldots, G_{q_t}$:
  - *One sequence for each collision* of active suffixes of phase t.

Suffix selection, third attempt: reusing inactive suffixes

We have the following problem to solve *for any phase* t:

- We have q_t sequences of *integers* $G_1, \ldots, G_{q_t}$:
  - *One sequence for each collision* of active suffixes of phase t.
  - Each sequence represents *the overlapping pattern of its collision*.

Suffix selection, third attempt: reusing inactive suffixes

We have the following problem to solve *for any phase* t:

- We have q_t sequences of *integers* $G_1, \ldots, G_{q_t}$:
  - *One sequence for each collision* of active suffixes of phase t.
  - Each sequence represents *the overlapping pattern of its collision*.
- For any two suffixes h_i of G_i and h_j of G_j, *we want to be able to retrieve* $lcp(h_i, h_j)$ *in* $O(1)$ *time*.

Suffix selection, third attempt: reusing inactive suffixes

We have the following problem to solve *for any phase* t:

- We have q_t sequences of *integers* $G_1, \ldots, G_{q_t}$:
  - *One sequence for each collision* of active suffixes of phase t.
  - Each sequence represents *the overlapping pattern of its collision*.
- For any two suffixes h_i of G_i and h_j of G_j, *we want to be able to retrieve* $lcp(h_i, h_j)$ *in* $O(1)$ *time*.

If we can solve this problem then

The comparison of any two prospective extents of phase t is reduced to one lcp query and one element comparison.

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

A *partial solution* to the problem.

Before the phase transition from t to $t+1$ we do the following:

Suffix selection, third attempt: reusing inactive suffixes

A *partial solution* to the problem.

Before the phase transition from t to $t+1$ we do the following:

(1) We *concatenate* the G_p's into a *single sequence*

$$G = G_1 0 G_2 0 \ldots 0 G_{q_t}$$

of $O\left(|\mathcal{A}_t|\right)$ integers.

Suffix selection, third attempt: reusing inactive suffixes

A *partial solution* to the problem.

Before the phase transition from t to $t+1$ we do the following:

(1) We *concatenate* the G_p's into a *single sequence*

$$G = G_1 0 G_2 0 \ldots 0 G_{q_t}$$

of $O\left(|\mathcal{A}_t|\right)$ integers.

(2) We *sort* the suffixes of G.

Suffix selection, third attempt: reusing inactive suffixes

A *partial solution* to the problem.

Before the phase transition from t to $t+1$ we do the following:

(1) We *concatenate* the G_p's into a *single sequence*

$$G = G_1 0 G_2 0 \ldots 0 G_{q_t}$$

of $O\left(|\mathcal{A}_t|\right)$ integers.

(2) We *sort* the suffixes of G.

Since we are dealing with a sequence of integers, we can use a *linear-time integer suffix sorting* algorithm (e.g. [Karkkainen and Sanders, ICALP 2003]).

Suffix selection, third attempt: reusing inactive suffixes

A *partial solution* to the problem.

Before the phase transition from t to $t+1$ we do the following:

(1) We *concatenate* the G_p's into a *single sequence*

$$G = G_1 0 G_2 0 \ldots 0 G_{q_t}$$

of $O\left(|\mathcal{A}_t|\right)$ integers.

(2) We *sort* the suffixes of G.

Since we are dealing with a sequence of integers, we can use a *linear-time integer suffix sorting* algorithm (e.g. [Karkkainen and Sanders, ICALP 2003]).

(3) We process the suffix array of G so that lcp queries *on the suffixes of* G can be answered in $O(1)$ time.

Suffix selection, third attempt: reusing inactive suffixes

A *partial solution* to the problem.

Before the phase transition from t to $t+1$ we do the following:

(1) We *concatenate* the G_p's into a *single sequence*

$$G = G_1 0 G_2 0 \ldots 0 G_{q_t}$$

of $O\left(|\mathcal{A}_t|\right)$ integers.

(2) We *sort* the suffixes of G.

Since we are dealing with a sequence of integers, we can use a *linear-time integer suffix sorting* algorithm (e.g. [Karkkainen and Sanders, ICALP 2003]).

(3) We process the suffix array of G so that lcp queries *on the suffixes of* G can be answered in $O(1)$ time.

We can use [Kasai, et al, CPM 2001] and [Harel, Tarjan, SICOMP 13, 1984].

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

Why is it a *partial solution*?

Suffix selection, third attempt: reusing inactive suffixes

Why is it a *partial solution*?

- In order to have a total cost $O(n)$, *the cost of the preprocessing for phase* t has to be $O(|\mathcal{A}_t|)$.

Suffix selection, third attempt: reusing inactive suffixes

Why is it a *partial solution*?

- In order to have a total cost $O(n)$, *the cost of the preprocessing for phase* t has to be $O(|\mathcal{A}_t|)$.
- Any *linear-time integer suffix sorting* algorithm (e.g. [Karkkainen and Sanders, ICALP 2003]) requires *the size of the alphabet of* G *to be linear in the length of* G.

Suffix selection, third attempt: reusing inactive suffixes

Why is it a *partial solution*?

- In order to have a total cost $O(n)$, *the cost of the preprocessing for phase* t has to be $O(|\mathcal{A}_t|)$.
- Any *linear-time integer suffix sorting* algorithm (e.g. [Karkkainen and Sanders, ICALP 2003]) requires *the size of the alphabet of* G *to be linear in the length of* G.
- Unfortunately, the integers in G are *suffix indexes of* σ_t (requiring $\log n$ bits to be represented) while $|G| = O(|\mathcal{A}_t|)$ and tends to 1 over time.

Suffix selection, third attempt: reusing inactive suffixes

Why is it a *partial solution*?

- In order to have a total cost $O(n)$, *the cost of the preprocessing for phase* t has to be $O(|\mathcal{A}_t|)$.
- Any *linear-time integer suffix sorting* algorithm (e.g. [Karkkainen and Sanders, ICALP 2003]) requires *the size of the alphabet of* G *to be linear in the length of* G.
- Unfortunately, the integers in G are *suffix indexes of* σ_t (requiring $\log n$ bits to be represented) while $|G| = O(|\mathcal{A}_t|)$ and tends to 1 over time.

The complete solution

Suffix selection, third attempt: reusing inactive suffixes

Why is it a *partial solution*?

- In order to have a total cost $O(n)$, *the cost of the preprocessing for phase* t has to be $O(|\mathcal{A}_t|)$.
- Any *linear-time integer suffix sorting* algorithm (e.g. [Karkkainen and Sanders, ICALP 2003]) requires *the size of the alphabet of* G *to be linear in the length of* G.
- Unfortunately, the integers in G are *suffix indexes of* σ_t (requiring $\log n$ bits to be represented) while $|G| = O(|\mathcal{A}_t|)$ and tends to 1 over time.

The complete solution

- Before we proceed with the second and third step, *we change the range* of the integers in G from $[1 \ldots n]$ to $[1 \ldots |\mathcal{A}_t|]$.

Suffix selection, third attempt: reusing inactive suffixes

Why is it a *partial solution*?

- In order to have a total cost $O(n)$, *the cost of the preprocessing for phase* t has to be $O(|\mathcal{A}_t|)$.
- Any *linear-time integer suffix sorting* algorithm (e.g. [Karkkainen and Sanders, ICALP 2003]) requires *the size of the alphabet of* G *to be linear in the length of* G.
- Unfortunately, the integers in G are *suffix indexes of* σ_t (requiring $\log n$ bits to be represented) while $|G| = O(|\mathcal{A}_t|)$ and tends to 1 over time.

The complete solution

- Before we proceed with the second and third step, *we change the range* of the integers in G from $[1 \dots n]$ to $[1 \dots |\mathcal{A}_t|]$.
- This is possible because the range change *does not need to maintain the lexicographical order* of the suffixes of G.

Suffix selection, third attempt: reusing inactive suffixes

Why is it a *partial solution*?

- In order to have a total cost $O(n)$, *the cost of the preprocessing for phase* t has to be $O(|\mathcal{A}_t|)$.
- Any *linear-time integer suffix sorting* algorithm (e.g. [Karkkainen and Sanders, ICALP 2003]) requires *the size of the alphabet of* G *to be linear in the length of* G.
- Unfortunately, the integers in G are *suffix indexes of* σ_t (requiring $\log n$ bits to be represented) while $|G| = O(|\mathcal{A}_t|)$ and tends to 1 over time.

The complete solution

- Before we proceed with the second and third step, *we change the range* of the integers in G from $[1 \dots n]$ to $[1 \dots |\mathcal{A}_t|]$.
- This is possible because the range change *does not need to maintain the lexicographical order* of the suffixes of G.
- We only need to preserve *the length of the longest common prefix* of any two suffixes of G.

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

Finally, let's deal with the *forward suffixes*.

> *The forward suffix of a suffix* T_j is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

Suffix selection, third attempt: reusing inactive suffixes

Finally, let's deal with the *forward suffixes*.

> *The forward suffix of a suffix* T_j is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

We want to maintain the following invariant:

> During any phase t, for any suffix T_i, active or inactive, *the forward suffix of* T_i *is known* (i.e. its index is explicitly stored and accessible in $O(1)$ time).

Suffix selection, third attempt: reusing inactive suffixes

Finally, let's deal with the *forward suffixes*.

> *The forward suffix of a suffix* T_j is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

We want to maintain the following invariant:

> During any phase t, for any suffix T_i, active or inactive, *the forward suffix of* T_i *is known* (i.e. its index is explicitly stored and accessible in $O(1)$ time).

For any phase t, during the *phase transition from* t *to* $t+1$ we have the following:

Suffix selection, third attempt: reusing inactive suffixes

Finally, let's deal with the *forward suffixes*.

> *The forward suffix of a suffix* T_j is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

We want to maintain the following invariant:

> During any phase t, for any suffix T_i, active or inactive, *the forward suffix of* T_i *is known* (i.e. its index is explicitly stored and accessible in $O(1)$ time).

For any phase t, during the *phase transition from* t *to* $t+1$ we have the following:

- The forward suffix of any $T_i \in \mathcal{I}_t$ *does not change*.

Suffix selection, third attempt: reusing inactive suffixes

Finally, let's deal with the *forward suffixes*.

> *The forward suffix of a suffix* T_j is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

We want to maintain the following invariant:

> During any phase t, for any suffix T_i, active or inactive, *the forward suffix of* T_i *is known* (i.e. its index is explicitly stored and accessible in $O(1)$ time).

For any phase t, during the *phase transition from* t *to* $t+1$ we have the following:

- The forward suffix of any $T_i \in \mathcal{I}_t$ *does not change*.
- The extent of any suffix $T_i \in \mathcal{A}_t$ *is enlarged* and so *the forward suffix of* T_i *must be updated*.

Suffix selection, third attempt: reusing inactive suffixes

Finally, let's deal with the *forward suffixes*.

> *The forward suffix of a suffix* T_j is the *inactive suffix* starting within the extent of T_j or right after it *whose extent goes the farthest* from the right end of T_j's extent.

We want to maintain the following invariant:

> During any phase t, for any suffix T_i, active or inactive, *the forward suffix of* T_i *is known* (i.e. its index is explicitly stored and accessible in $O(1)$ time).

For any phase t, during the *phase transition from* t *to* $t+1$ we have the following:

- The forward suffix of any $T_i \in \mathcal{I}_t$ *does not change*.
- The extent of any suffix $T_i \in \mathcal{A}_t$ *is enlarged* and so *the forward suffix of* T_i *must be updated*.

Therefore:

To maintain the forward suffixes, we have to solve a Dynamic Range Maximum Query problem

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

According to the length of σ_t, the computation is divided into *two epochs*:

Suffix selection, third attempt: reusing inactive suffixes

According to the length of σ_t, the computation is divided into *two epochs*:

***Early Phases*,** where $|\sigma_t| = O\left(\log^2 n\right)$.

Suffix selection, third attempt: reusing inactive suffixes

According to the length of σ_t, the computation is divided into *two epochs*:

***Early Phases*,** where $|\sigma_t| = O\left(\log^2 n\right)$.

- For the early phases *we develop a Dynamic Range Maximum Query structure* that can be
  - *built in linear time*
  - *queried in* $O(1)$ *time*
  - *updated in* $O(1)$ *time*.

Suffix selection, third attempt: reusing inactive suffixes

According to the length of σ_t, the computation is divided into *two epochs*:

Early Phases, where $|\sigma_t| = O\left(\log^2 n\right)$.

- For the early phases *we develop a Dynamic Range Maximum Query structure* that can be
  - *built in linear time*
  - *queried in* $O(1)$ *time*
  - *updated in* $O(1)$ *time*.
- The structure exploits the following crucial fact:

> *Both the integer values stored in the structure and the length of the query intervals* are $O\left(\log^2 n\right)$.

Suffix selection, third attempt: reusing inactive suffixes

Suffix selection, third attempt: reusing inactive suffixes

***Late Phases*,** where $|\sigma_t| = \Omega\left(\log^2 n\right)$.

Suffix selection, third attempt: reusing inactive suffixes

Late Phases, where $|\sigma_t| = \Omega\left(\log^2 n\right)$.

- For the late phases *we use a much simpler Dynamic Range Maximum Query structure* that can be
  - *built in linear time*,
  - *queried in* $O(\log^2 n)$ *time*
  - *updated in* $O(\log n)$ *time*.

Suffix selection, third attempt: reusing inactive suffixes

Late Phases, where $|\sigma_t| = \Omega\left(\log^2 n\right)$.

- For the late phases *we use a much simpler Dynamic Range Maximum Query structure* that can be
  - *built in linear time*,
  - *queried in* $O(\log^2 n)$ *time*
  - *updated in* $O(\log n)$ *time*.
- In the late phases *we cannot exploit the hypothesis on the length of* σ_t...

Suffix selection, third attempt: reusing inactive suffixes

Late Phases, where $|\sigma_t| = \Omega\left(\log^2 n\right)$.

- For the late phases *we use a much simpler Dynamic Range Maximum Query structure* that can be
  - *built in linear time*,
  - *queried in* $O(\log^2 n)$ *time*
  - *updated in* $O(\log n)$ *time*.
- In the late phases *we cannot exploit the hypothesis on the length of* σ_t...
- ...but we know that *from the first late phase* t' *to the last one* there will be $O\left(n/\left|\sigma_{t'}\right|\right) = O\left(n/\log^2 n\right)$ *active suffixes*.

Suffix selection, third attempt: reusing inactive suffixes

Late Phases, where $|\sigma_t| = \Omega\left(\log^2 n\right)$.

- For the late phases *we use a much simpler Dynamic Range Maximum Query structure* that can be
  - *built in linear time*,
  - *queried in* $O(\log^2 n)$ *time*
  - *updated in* $O(\log n)$ *time*.
- In the late phases *we cannot exploit the hypothesis on the length of* $\sigma_t \ldots$
- $\ldots$ but we know that *from the first late phase* t' *to the last one* there will be $O\left(n/\left|\sigma_{t'}\right|\right) = O\left(n/\log^2 n\right)$ *active suffixes*.

Therefore,

The total cost for maintaining the forward suffixes during both early and late phases is $O\left(n\right)$.

Selection of Extreme Suffixes

Selection of Extreme Suffixes

Selection of Extreme Suffixes

- *Same settings seen in the Suffix Selection problem* (sequence T, each $T[i]$ drawn from $(\mathcal{U}, <)$, comparison model, lexicographical order$\ldots$)

Selection of Extreme Suffixes

- *Same settings seen in the Suffix Selection problem* (sequence T, each $T[i]$ drawn from $(\mathcal{U}, <)$, comparison model, lexicographical order...)
- But this time we want to find...
  - *maximum suffix*
  - *minimum suffix*
  - *maximum suffix AND minimum suffix* (i.e. simultaneously).

Selection of Extreme Suffixes

- *Same settings seen in the Suffix Selection problem* (sequence T, each $T[i]$ drawn from $(\mathcal{U}, <)$, comparison model, lexicographical order...)
- But this time we want to find...
  - *maximum suffix*
  - *minimum suffix*
  - *maximum suffix AND minimum suffix* (i.e. simultaneously).
- ...and we want the *exact complexities* (i.e. including the constant factors).

Selection of Extreme Suffixes

- *Same settings seen in the Suffix Selection problem* (sequence T, each $T[i]$ drawn from $(\mathcal{U}, <)$, comparison model, lexicographical order...)
- But this time we want to find...
  - *maximum suffix*
  - *minimum suffix*
  - *maximum suffix AND minimum suffix* (i.e. simultaneously).
- ...and we want the *exact complexities* (i.e. including the constant factors).
- Surprisingly, the exact complexities of such basic problems *were not known*...

Selection of Extreme Suffixes

- *Same settings seen in the Suffix Selection problem* (sequence T, each $T[i]$ drawn from $(\mathcal{U}, <)$, comparison model, lexicographical order...)
- But this time we want to find...
  - *maximum suffix*
  - *minimum suffix*
  - *maximum suffix AND minimum suffix* (i.e. simultaneously).
- ...and we want the *exact complexities* (i.e. including the constant factors).
- Surprisingly, the exact complexities of such basic problems *were not known*...
- ...*and still aren't*, since *we don't have matching lower bounds* for the new upper bounds.

Selection of Extreme Suffixes

Selection of Extreme Suffixes

Previous best upper bounds:

Selection of Extreme Suffixes

Previous best upper bounds:

- For finding the maximum suffix or the minimum suffix

$$\leq \frac{3}{2}n \text{ comparisons}$$

Selection of Extreme Suffixes

Previous best upper bounds:

- For finding the maximum suffix or the minimum suffix

$$\leq \frac{3}{2}n \text{ comparisons}$$

[Shiloach, J. Algorithms 2, 1981] or [Duval, J. Algorithms 4, 1983]

Selection of Extreme Suffixes

Previous best upper bounds:

- For finding the maximum suffix or the minimum suffix

$$\leq \frac{3}{2}n \text{ comparisons}$$

[Shiloach, J. Algorithms 2, 1981] or [Duval, J. Algorithms 4, 1983]

- Maximum AND minimum: $\leq 3n$ (just apply two times).

Selection of Extreme Suffixes

Previous best upper bounds:

- For finding the maximum suffix or the minimum suffix

$$\leq \frac{3}{2}n \text{ comparisons}$$

[Shiloach, J. Algorithms 2, 1981] or [Duval, J. Algorithms 4, 1983]

- Maximum AND minimum: $\leq 3n$ (just apply two times).

New upper bounds:

Selection of Extreme Suffixes

Previous best upper bounds:

- For finding the maximum suffix or the minimum suffix

$$\leq \frac{3}{2}n \text{ comparisons}$$

[Shiloach, J. Algorithms 2, 1981] or [Duval, J. Algorithms 4, 1983]

- Maximum AND minimum: $\leq 3n$ (just apply two times).

New upper bounds:

- Maximum or minimum:

$$\leq \frac{4}{3}n \text{ comparisons}$$

Selection of Extreme Suffixes

Previous best upper bounds:

- For finding the maximum suffix or the minimum suffix

$$\leq \frac{3}{2}n \text{ comparisons}$$

[Shiloach, J. Algorithms 2, 1981] or [Duval, J. Algorithms 4, 1983]

- Maximum AND minimum: $\leq 3n$ (just apply two times).

New upper bounds:

- Maximum or minimum:

$$\leq \frac{4}{3}n \text{ comparisons}$$

[Franceschini, Hagerup, 2007]

Selection of Extreme Suffixes

Previous best upper bounds:

- For finding the maximum suffix or the minimum suffix

$$\leq \frac{3}{2}n \text{ comparisons}$$

[Shiloach, J. Algorithms 2, 1981] or [Duval, J. Algorithms 4, 1983]

- Maximum AND minimum: $\leq 3n$ (just apply two times).

New upper bounds:

- Maximum or minimum:

$$\leq \frac{4}{3}n \text{ comparisons}$$

[Franceschini, Hagerup, 2007]

- Maximum AND minimum: $\leq \frac{5}{2}n$.

Selection of the Maximum Suffix

Selection of the Maximum Suffix

Let's focus on finding the *maximum suffix* and let's consider *Duval's algorithm:*

Selection of the Maximum Suffix

Let's focus on finding the *maximum suffix* and let's consider *Duval's algorithm:*

- The algorithm does *one pass over* T from left to right, going through *phases* and *transitions* where the knowledge about the maximum suffix is *increased/changed*.

Selection of the Maximum Suffix

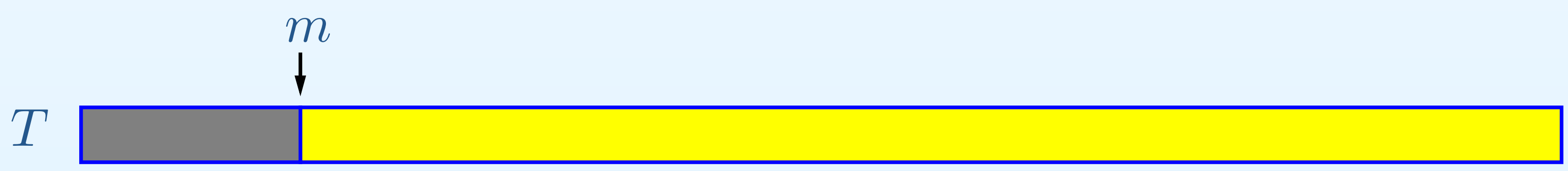
Let's focus on finding the *maximum suffix* and let's consider *Duval's algorithm:*

- The algorithm does *one pass over* T from left to right, going through *phases* and *transitions* where the knowledge about the maximum suffix is *increased/changed*.
- At any phase we have the following:

Selection of the Maximum Suffix

Let's focus on finding the *maximum suffix* and let's consider *Duval's algorithm:*

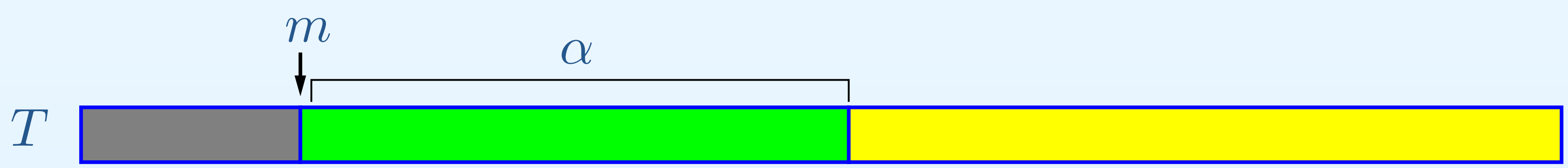
- The algorithm does *one pass over* T from left to right, going through *phases* and *transitions* where the knowledge about the maximum suffix is *increased/changed*.
- At any phase we have the following:
  - The *candidate suffix* m.



Selection of the Maximum Suffix

Let's focus on finding the *maximum suffix* and let's consider *Duval's algorithm:*

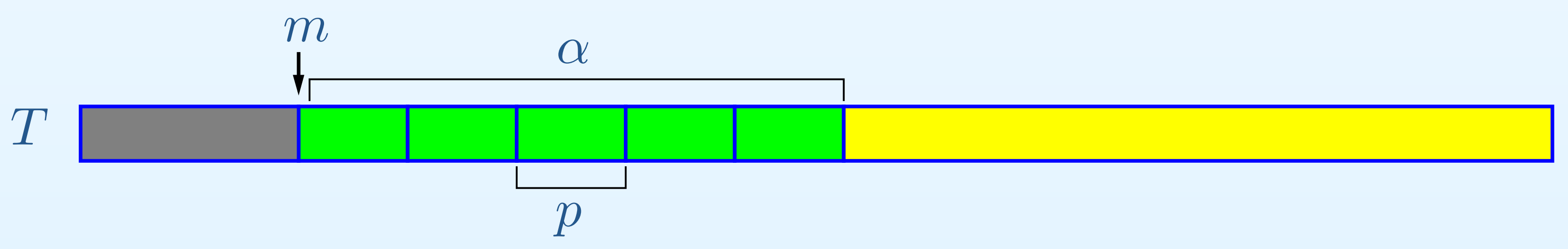
- The algorithm does *one pass over* T from left to right, going through *phases* and *transitions* where the knowledge about the maximum suffix is *increased/changed*.
- At any phase we have the following:
  - The *candidate suffix* m.
  - A prefix α of m, *the known zone*.



Selection of the Maximum Suffix

Let's focus on finding the *maximum suffix* and let's consider *Duval's algorithm:*

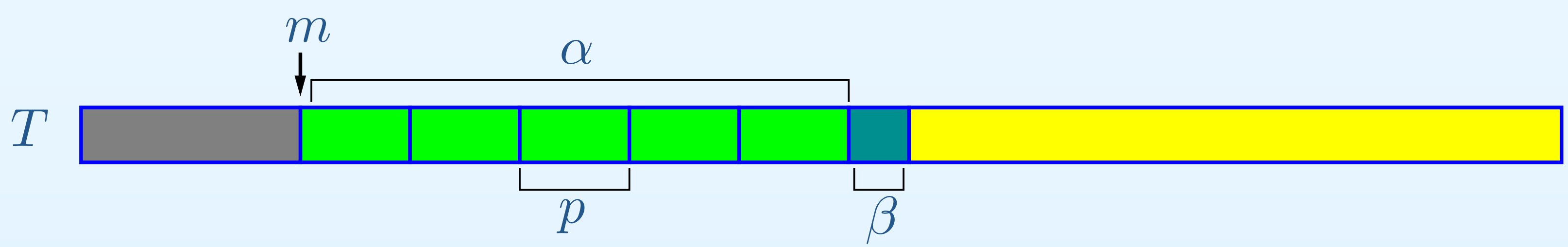
- The algorithm does *one pass over* T from left to right, going through *phases* and *transitions* where the knowledge about the maximum suffix is *increased/changed*.
- At any phase we have the following:
  - The *candidate suffix* m.
  - A prefix α of m, *the known zone*.
  - The *period* p of α (i.e. $\alpha = p^l$ for an integer l).



Selection of the Maximum Suffix

Let's focus on finding the *maximum suffix* and let's consider *Duval's algorithm:*

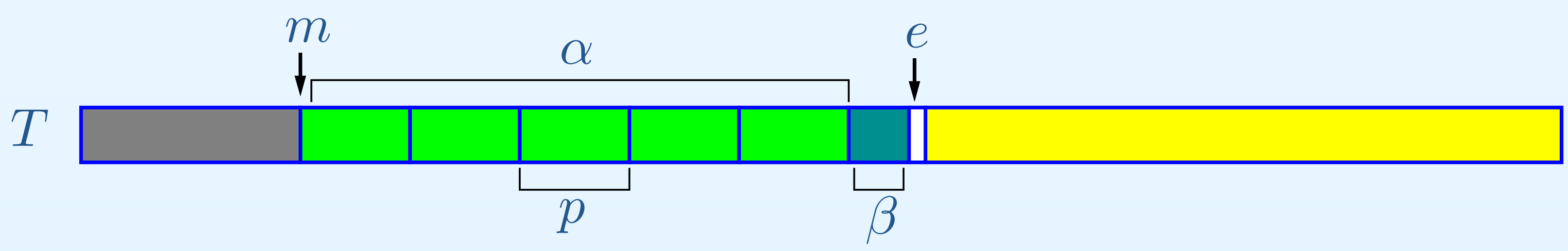
- The algorithm does *one pass over* T from left to right, going through *phases* and *transitions* where the knowledge about the maximum suffix is *increased/changed*.
- At any phase we have the following:
  - The *candidate suffix* m.
  - A prefix α of m, *the known zone*.
  - The *period* p of α (i.e. $\alpha = p^l$ for an integer l).
  - A prefix β of p, *the expansion zone*.



Selection of the Maximum Suffix

Let's focus on finding the *maximum suffix* and let's consider *Duval's algorithm:*

- The algorithm does *one pass over* T from left to right, going through *phases* and *transitions* where the knowledge about the maximum suffix is *increased/changed*.
- At any phase we have the following:
  - The *candidate suffix* m.
  - A prefix α of m, *the known zone*.
  - The *period* p of α (i.e. $\alpha = p^l$ for an integer l).
  - A prefix β of p, *the expansion zone*.
  - The *currently examined element* e.



Selection of the Maximum Suffix

Selection of the Maximum Suffix

Then, e is compared to the *corresponding element* e' in p.

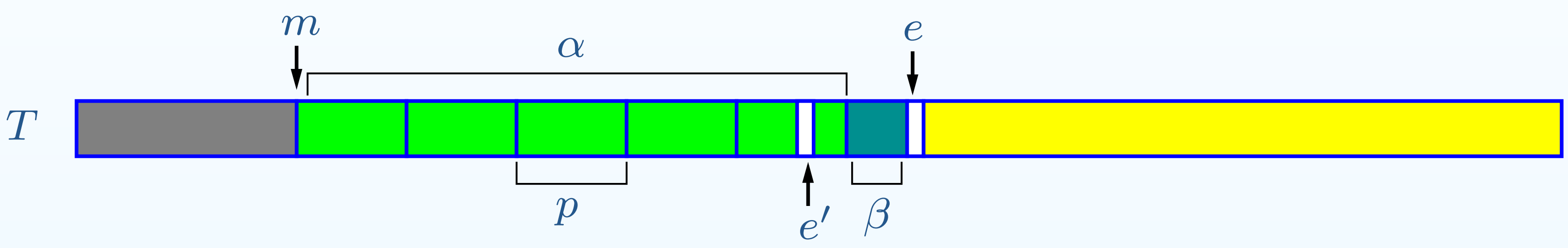
We have three types of transitions:

Selection of the Maximum Suffix

Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

(1) $e = e'$

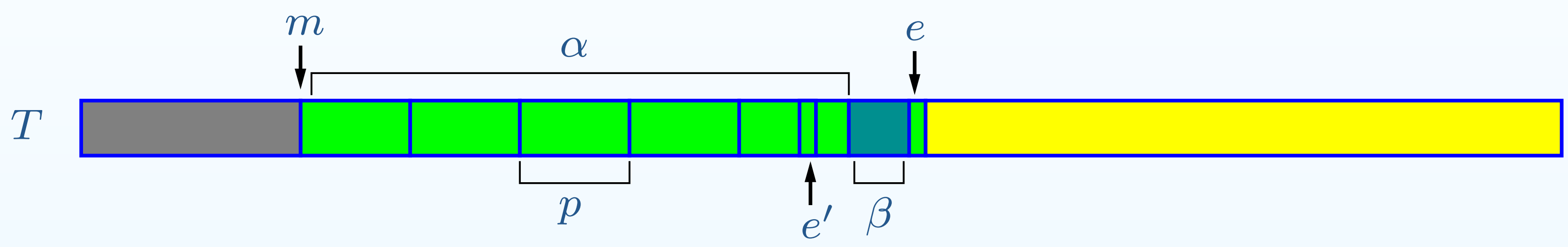


Selection of the Maximum Suffix

Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

(1) $e = e'$

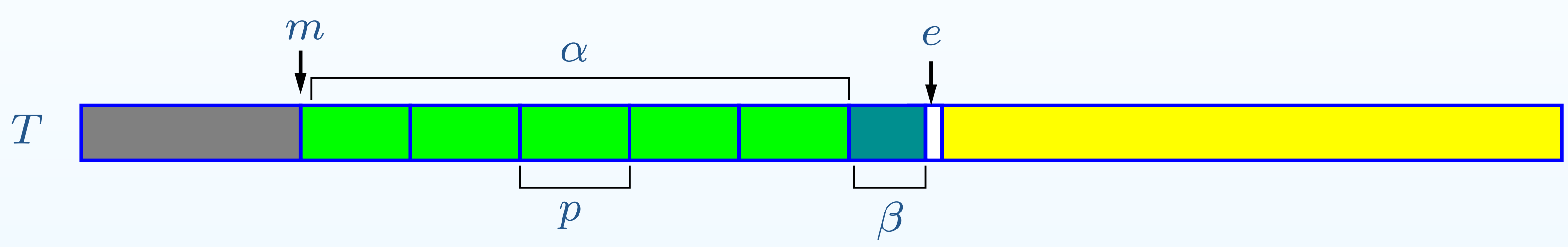


Selection of the Maximum Suffix

Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

(1) $e = e'$

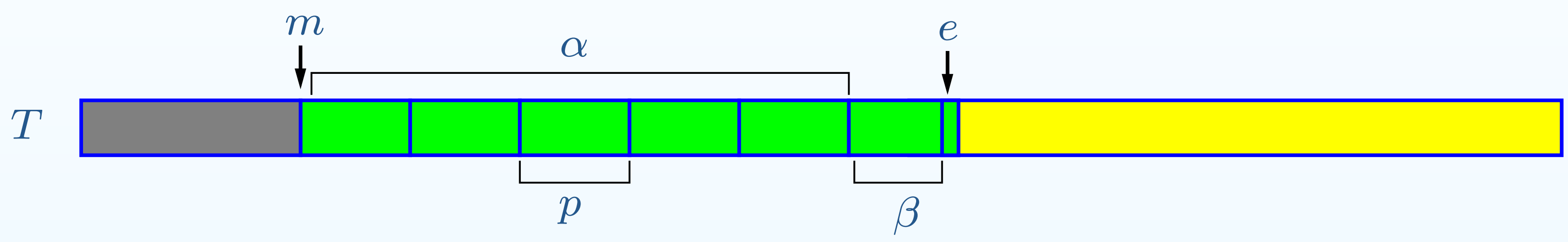


Selection of the Maximum Suffix

Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

(1) $e = e'$

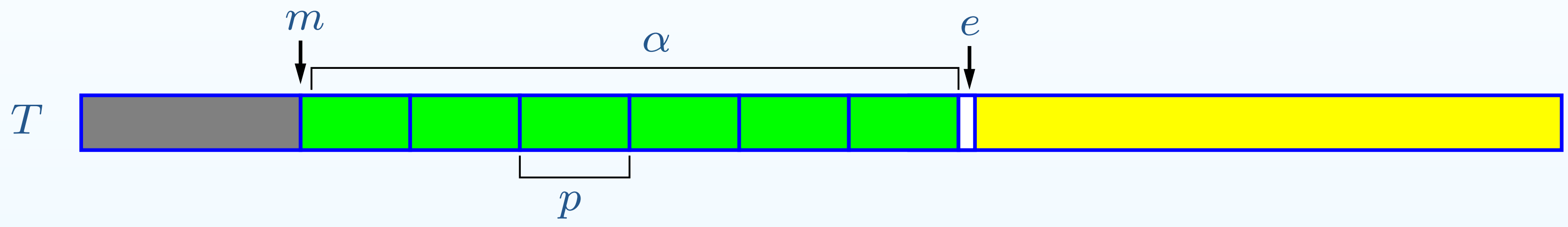


Selection of the Maximum Suffix

Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

(1) $e = e'$

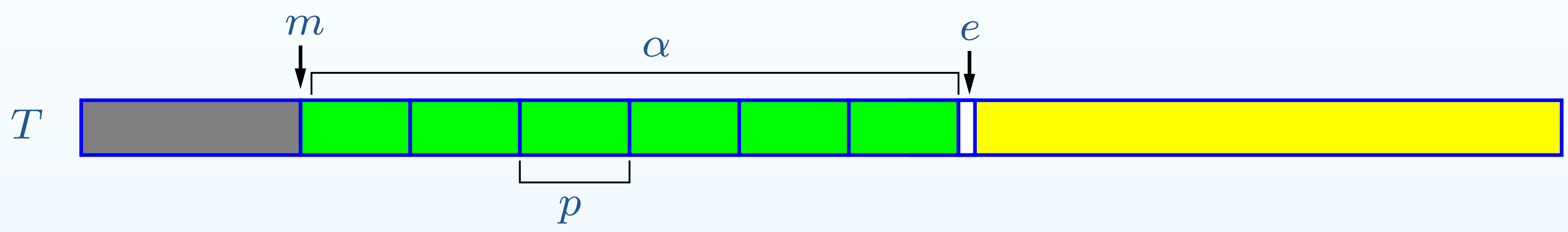


Selection of the Maximum Suffix

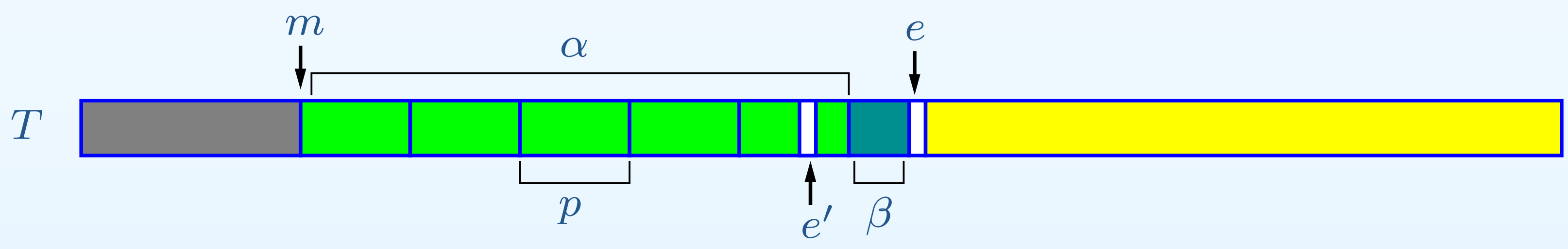
Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

(1) $e = e'$



(2) $e < e'$

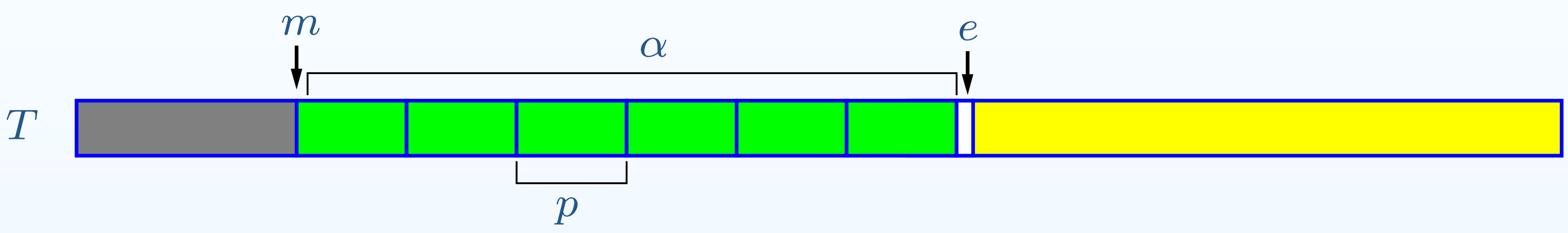


Selection of the Maximum Suffix

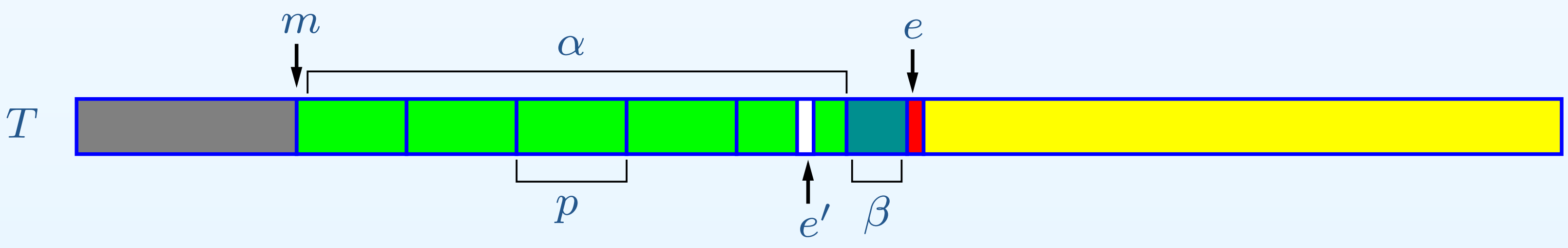
Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

(1) $e = e'$



(2) $e < e'$

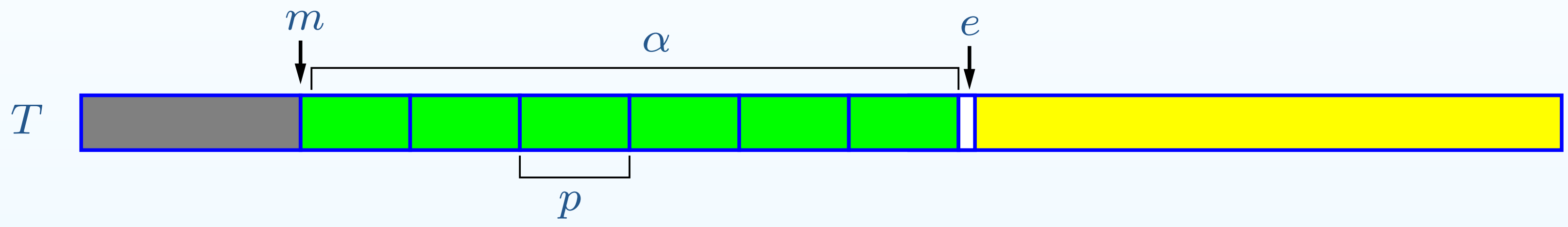


Selection of the Maximum Suffix

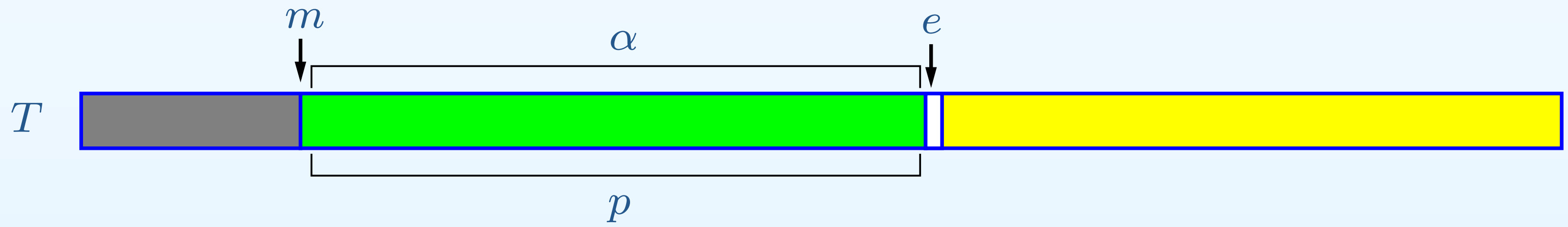
Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

(1) $e = e'$



(2) $e < e'$

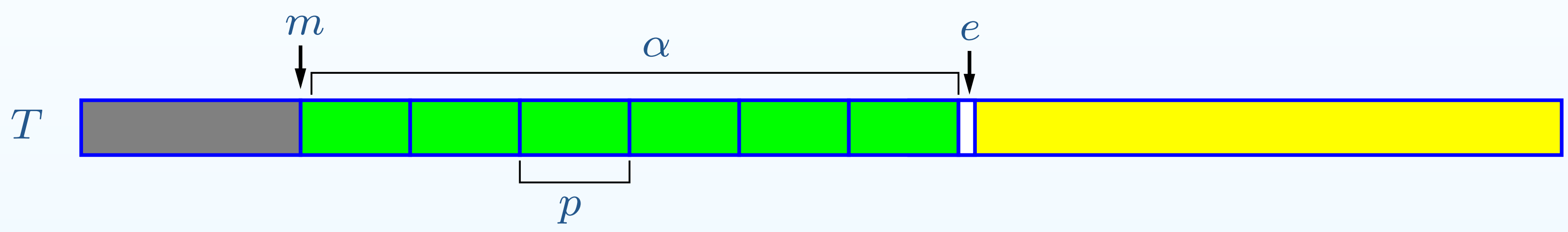


Selection of the Maximum Suffix

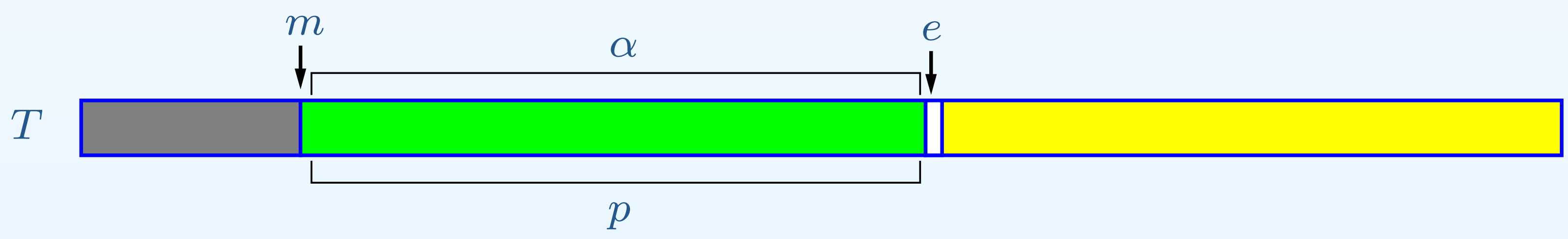
Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

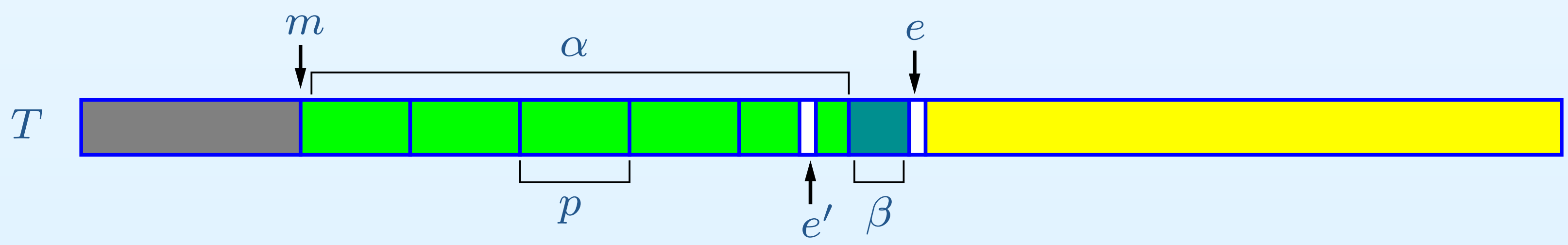
(1) $e = e'$



(2) $e < e'$



(3) $e > e'$

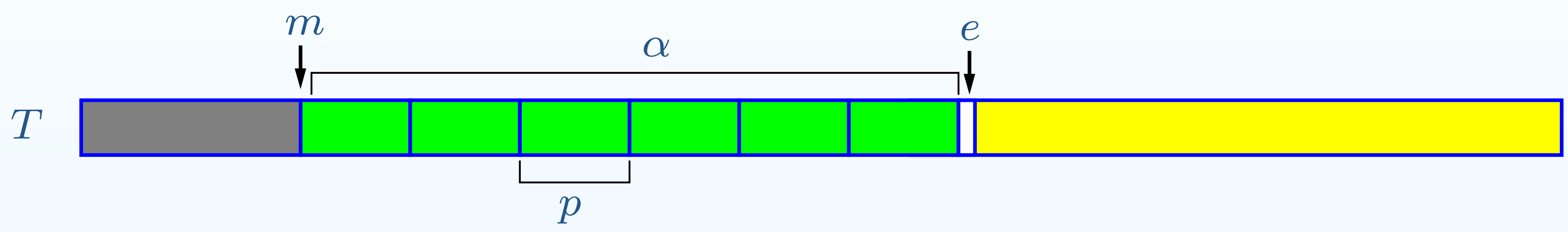


Selection of the Maximum Suffix

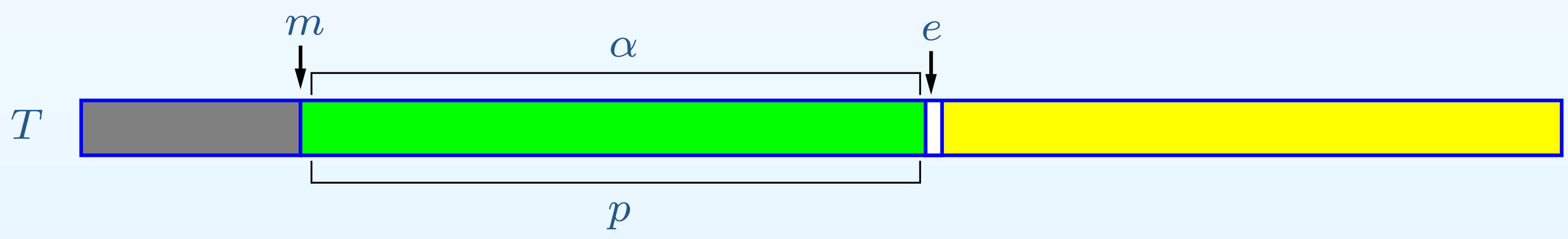
Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

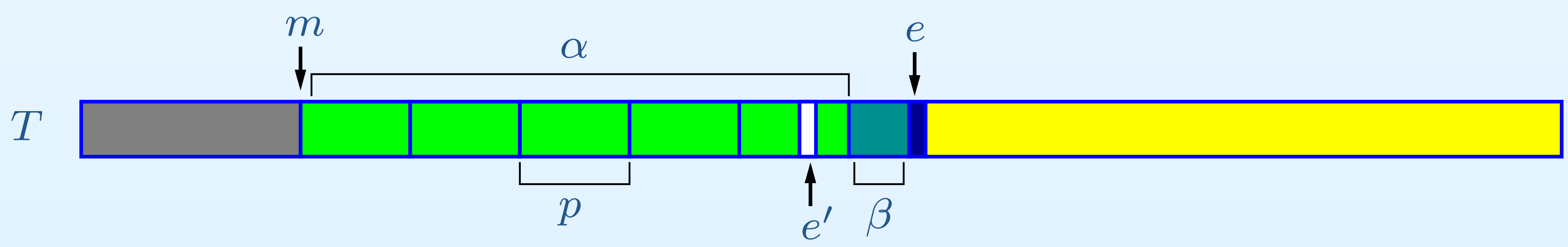
(1) $e = e'$



(2) $e < e'$



(3) $e > e'$

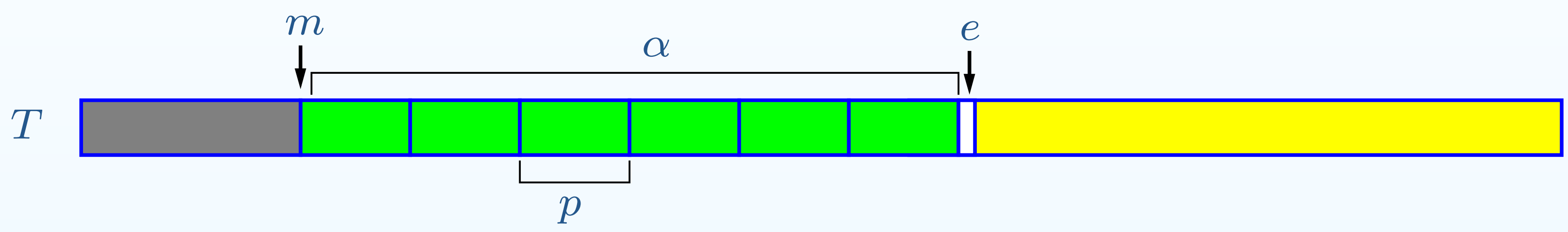


Selection of the Maximum Suffix

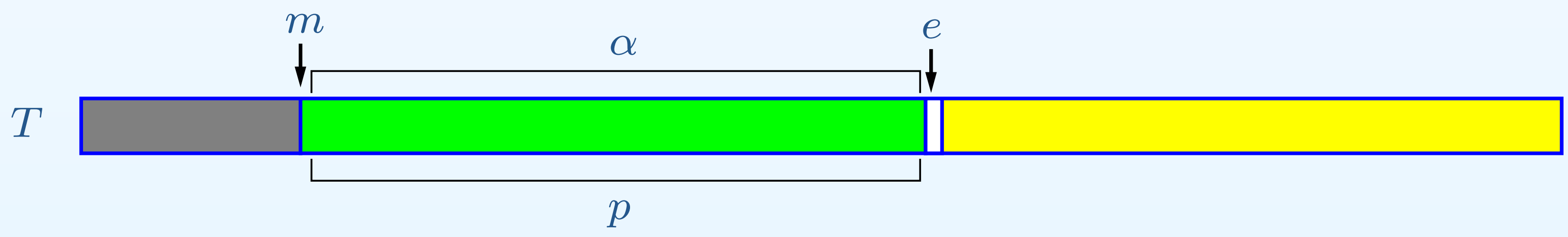
Then, e is compared to the *corresponding element* e' in p.

We have three types of transitions:

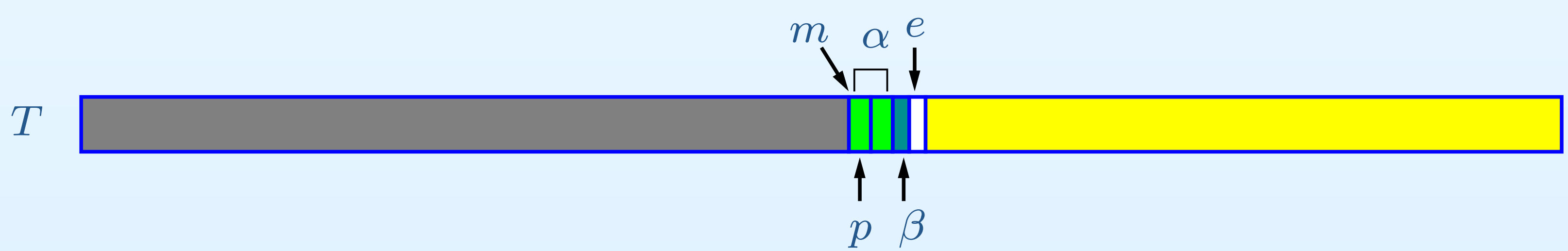
(1) $e = e'$



(2) $e < e'$



(3) $e > e'$



Selection of the Maximum Suffix

Selection of the Maximum Suffix

Duval's algorithm finds the maximum suffix with at most $\frac{3}{2}n$ comparisons

Selection of the Maximum Suffix

> *Duval's algorithm finds the maximum suffix with at most $\frac{3}{2}n$ comparisons*

Why?

- During any transition element e *is compared one time*.

Selection of the Maximum Suffix

> *Duval's algorithm finds the maximum suffix with at most $\frac{3}{2}n$ comparisons*

Why?

- During any transition element e *is compared one time*.
- During transitions of *type 1* and *2* we move to the *next unseen element*...

Selection of the Maximum Suffix

Duval's algorithm finds the maximum suffix with at most $\frac{3}{2}n$ comparisons

Why?

- During any transition element e *is compared one time*.
- During transitions of *type 1* and *2* we move to the *next unseen element*...
- ...but that *does not happen with type 3 transitions* in which we stay on the current e.

Selection of the Maximum Suffix

Duval's algorithm finds the maximum suffix with at most $\frac{3}{2}n$ comparisons

Why?

- During any transition element e *is compared one time*.
- During transitions of *type 1* and *2* we move to the *next unseen element*...
- ...but that *does not happen with type 3 transitions* in which we stay on the current e.
- However, *there cannot be two consecutive type 3 transitions*...

Selection of the Maximum Suffix

Duval's algorithm finds the maximum suffix with at most $\frac{3}{2}n$ comparisons

Why?

- During any transition element e *is compared one time*.
- During transitions of *type 1* and *2* we move to the *next unseen element*...
- ...but that *does not happen with type 3 transitions* in which we stay on the current e.
- However, *there cannot be two consecutive type 3 transitions*...
- ...*unless* e *has been compared to the first element of a the period* p but this is a particular case that *does not need the extra comparison*.

Selection of the Maximum Suffix

> *Duval's algorithm finds the maximum suffix with at most* $\frac{3}{2}n$ *comparisons*

Why?

- During any transition element e *is compared one time*.
- During transitions of *type 1* and *2* we move to the *next unseen element*...
- ...but that *does not happen with type 3 transitions* in which we stay on the current e.
- However, *there cannot be two consecutive type 3 transitions*...
- ...*unless* e *has been compared to the first element of a the period* p but this is a particular case that *does not need the extra comparison*.

Worst case scenario for Duval's algorithm:

9	1	9	2	9	3	9	4	9	5	9	6	9	7	9	8	9	9

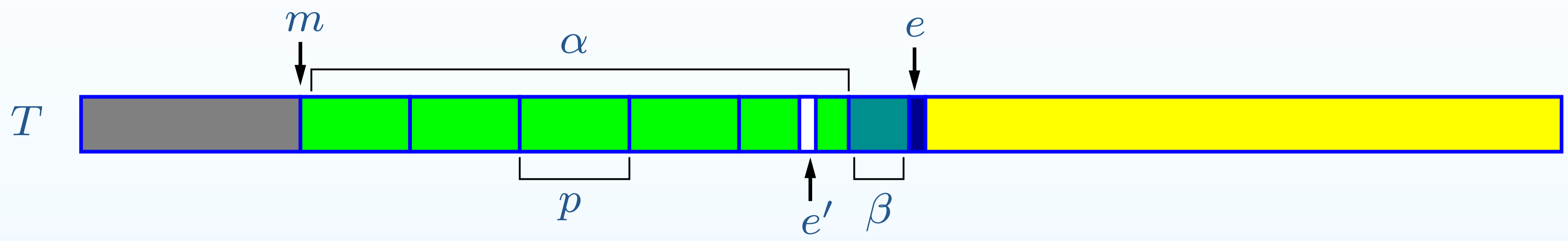
Maximum Suffix Selection: Uncertainty Approach

Maximum Suffix Selection: Uncertainty Approach

The reasons for remaining on e after a type 3 transition:

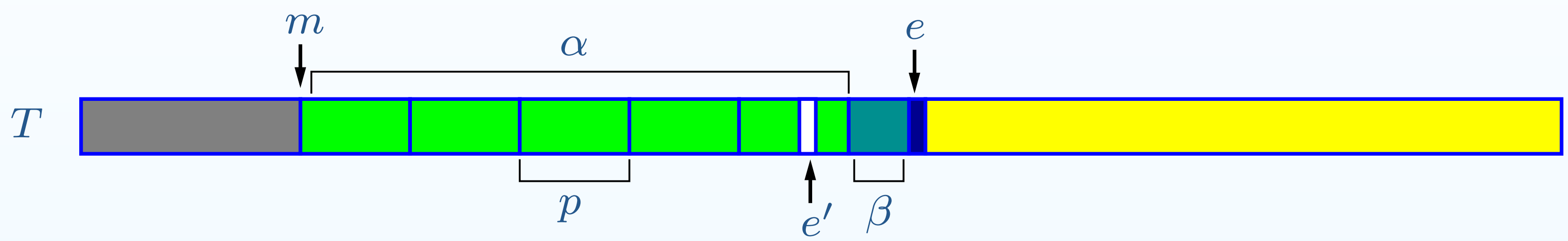
Maximum Suffix Selection: Uncertainty Approach

The reasons for remaining on e after a type 3 transition:



Maximum Suffix Selection: Uncertainty Approach

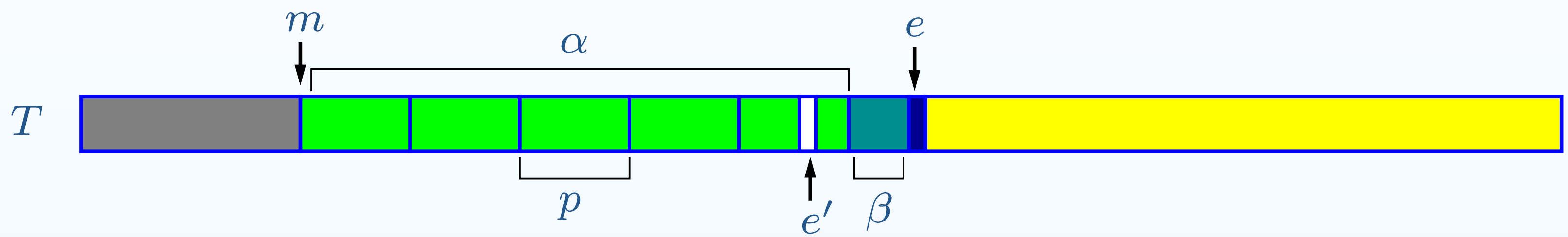
The reasons for remaining on e after a type 3 transition:



- e *could be the start of the actual maximum suffix.*

Maximum Suffix Selection: Uncertainty Approach

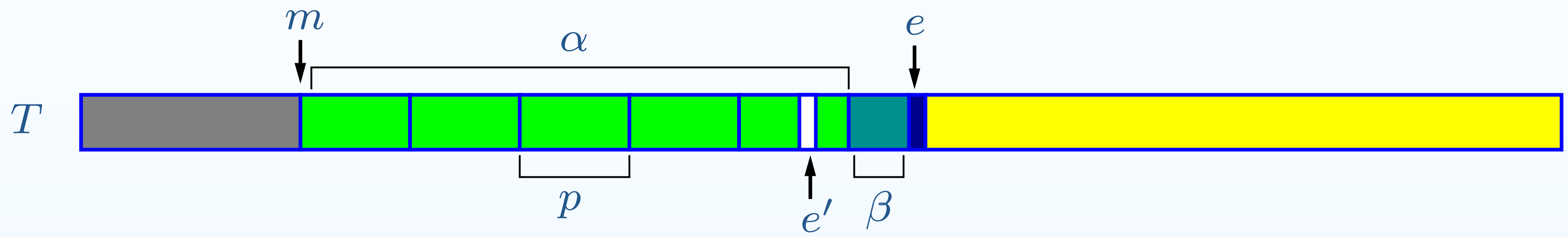
The reasons for remaining on e after a type 3 transition:



- *e could be the start of the actual maximum suffix.*
- *the actual maximum suffix could start somewhere within* β (thanks to e being greater than e').

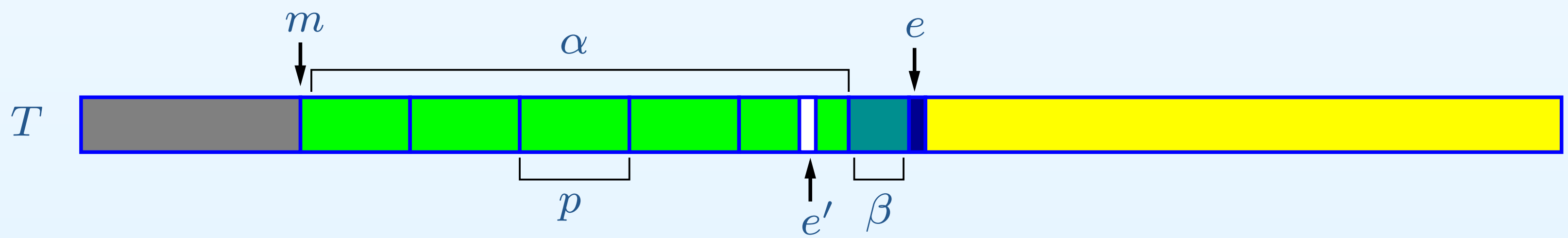
Maximum Suffix Selection: Uncertainty Approach

The reasons for remaining on e after a type 3 transition:



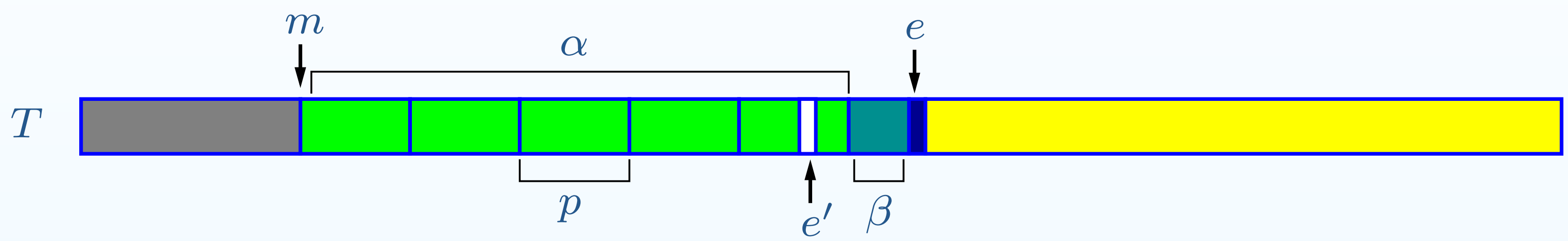
- *e could be the start of the actual maximum suffix.*
- *the actual maximum suffix could start somewhere within β* (thanks to e being greater than e').

The uncertainty approach:



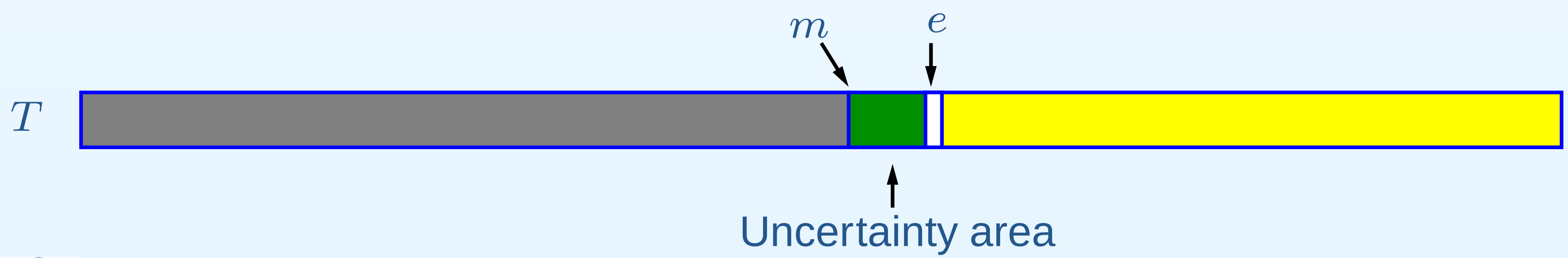
Maximum Suffix Selection: Uncertainty Approach

The reasons for remaining on e after a type 3 transition:



- *e could be the start of the actual maximum suffix.*
- *the actual maximum suffix could start somewhere within β* (thanks to e being greater than e').

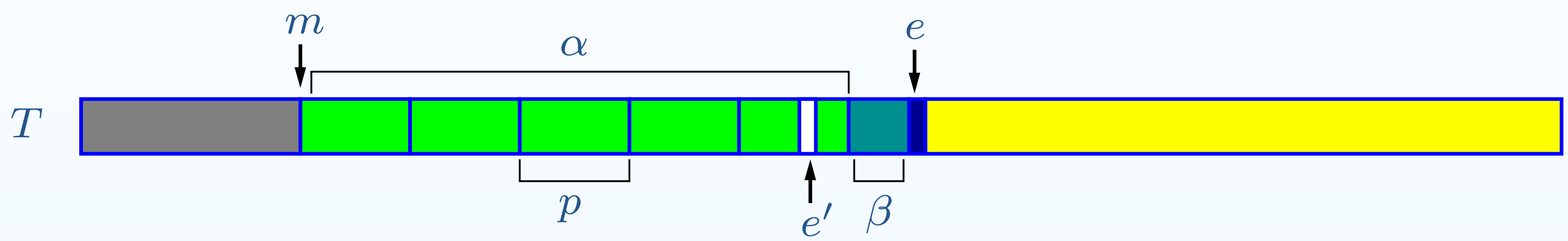
The uncertainty approach:



- Obviously, *we still move m* (the current m cannot be the maximum suffix)...

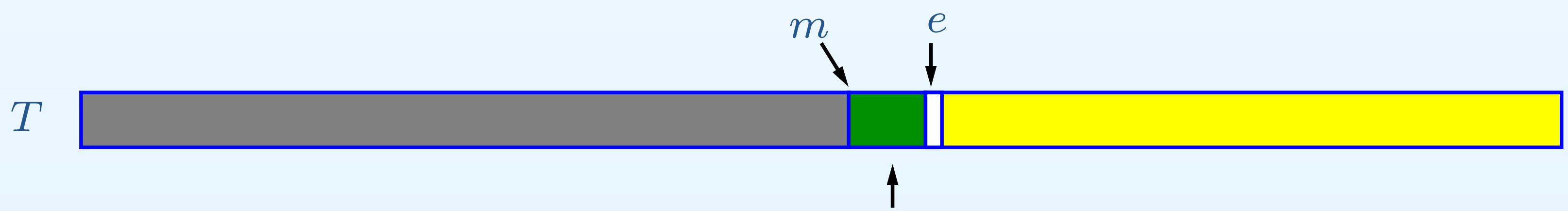
Maximum Suffix Selection: Uncertainty Approach

The reasons for remaining on e after a type 3 transition:



- *e could be the start of the actual maximum suffix.*
- *the actual maximum suffix could start somewhere within β* (thanks to e being greater than e').

The uncertainty approach:



- Obviously, *we still move m* (the current m cannot be the maximum suffix)...
- ...but *we move e too* and we keep *an uncertainty area within which the current maximum suffix starts* (but we don't know where exactly).

Maximum Suffix Selection: Uncertainty Approach

Maximum Suffix Selection: Uncertainty Approach

- The uncertainty area has a *fixed size*.

Maximum Suffix Selection: Uncertainty Approach

- The uncertainty area has a *fixed size*.
- When, during the computation, *new uncertainties appear outside the uncertainty area* we need
  - *to break the uncertainty*
  - and find *where the current maximum suffix actually starts*.

Maximum Suffix Selection: Uncertainty Approach

- The uncertainty area has a *fixed size*.
- When, during the computation, *new uncertainties appear outside the uncertainty area* we need
  - *to break the uncertainty*
  - and find *where the current maximum suffix actually starts*.
- But the time we waited in uncertainty *allows us to save comparisons* in the final count.

Maximum Suffix Selection: Uncertainty Approach

- The uncertainty area has a *fixed size*.
- When, during the computation, *new uncertainties appear outside the uncertainty area* we need
  - *to break the uncertainty*
  - and find *where the current maximum suffix actually starts*.
- But the time we waited in uncertainty *allows us to save comparisons* in the final count.

Unfortunately,

this approach does not seem to work with uncertainty areas larger than two positions

Maximum Suffix Selection: Uncertainty Approach

- The uncertainty area has a *fixed size*.
- When, during the computation, *new uncertainties appear outside the uncertainty area* we need
  - *to break the uncertainty*
  - and find *where the current maximum suffix actually starts*.
- But the time we waited in uncertainty *allows us to save comparisons* in the final count.

Unfortunately,

this approach does not seem to work with uncertainty areas larger than two positions

But this is enough to deal with Duval's worst case scenarios

9	1	9	2	9	3	9	4	9	5	9	6	9	7	9	8	9	9

with less than $\frac{4}{3}n$ comparisons.