Incremental BPE Tokenization

by Shenghu Jiang (姜圣虎) and Ruihao Gong (龚睿昊)

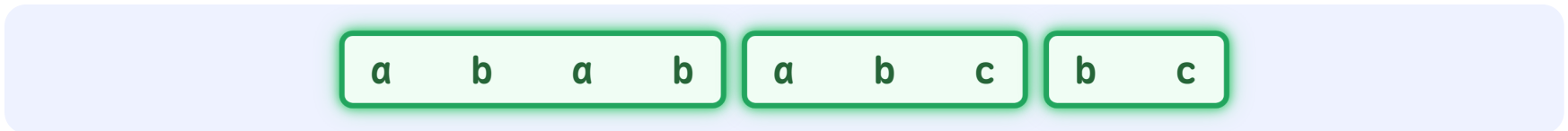
BUAA & SenseTime

Byte-Pair Encoding (BPE)

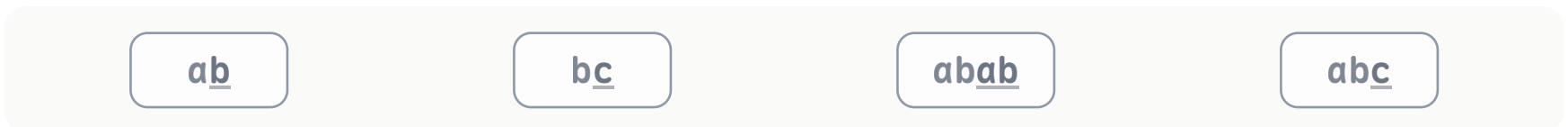
1. By Priority

2. Left-to-Right

3. Non-Overlap



Current Token Sequence

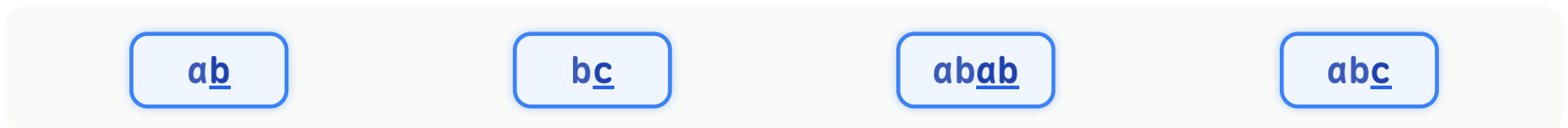


Merge Rules: `["a b", "b c", "ab ab", "ab c"]`

Token-wise Truncation

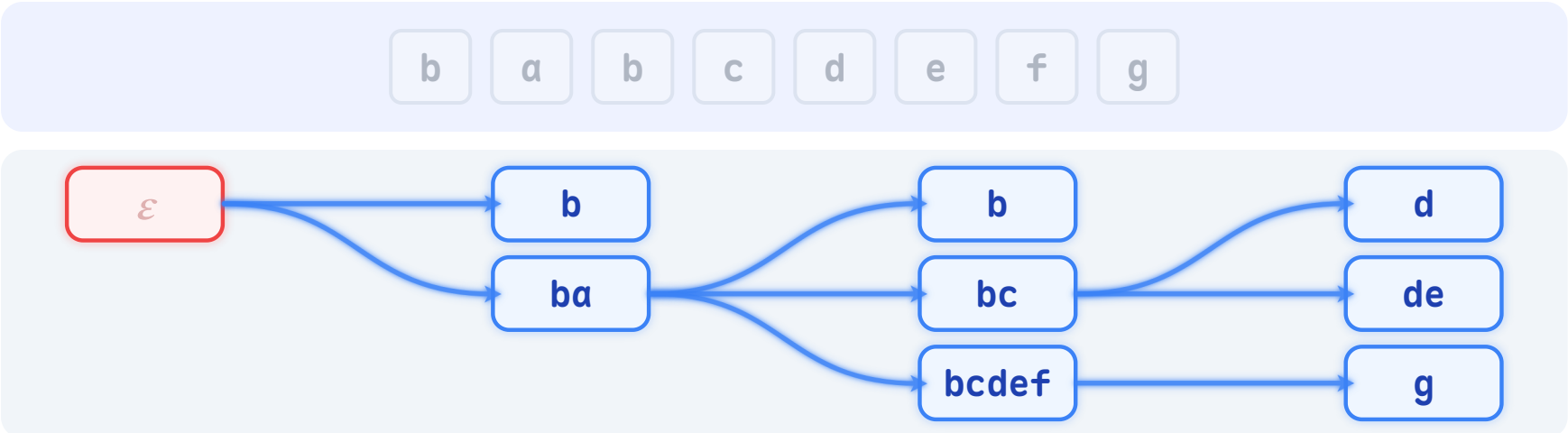
Token-wise truncation $\Longleftrightarrow$ Tokenization of the corresponding string

Current Token Sequence



Merge Rules: `["a b", "b c", "ab ab", "ab c"]`

Prefix Tree of Tokens



Prefix Tree of Tokens for `babcdefg`

Merge Rules

Incremental Objective

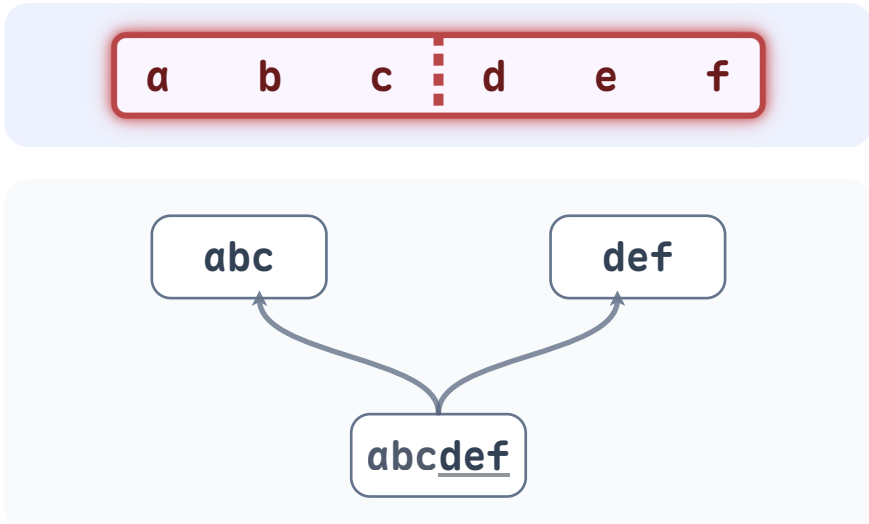
The "Last Token"

Incrementally maintain the "**last token**" $\theta(p)$ for *each prefix* p of the input string s.

> **Note**
> Here we can naturally derive the "eager output" mechanism. Check out the "Eager Output" section in the paper for details!

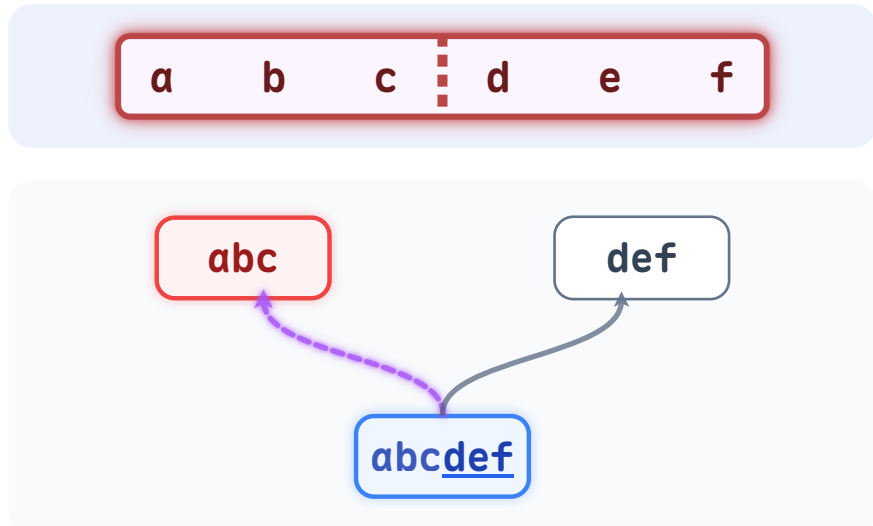
Predecessors & Successors

- Predecessor: $\text{pre}(t)$
- Successor: $\text{suc}(t)$
- $t = \text{concatenate}(\text{pre}(t), \text{suc}(t))$
- Deterministic & token-wise truncation ->
  - The relationship is **unique**!



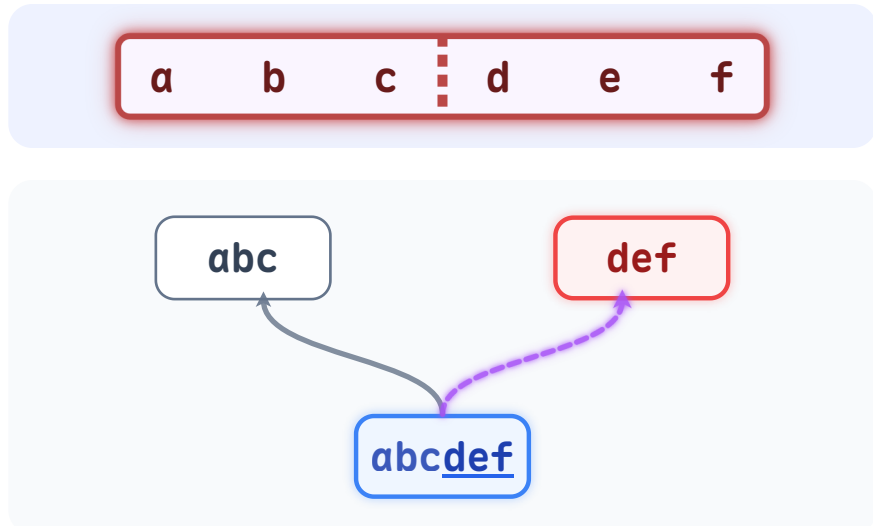
Predecessors & Successors

- Predecessor: $\text{pre}(t)$
- Successor: $\text{suc}(t)$
- $t = \text{concatenate}(\text{pre}(t), \text{suc}(t))$
- Deterministic & token-wise truncation ->
  - The relationship is **unique**!

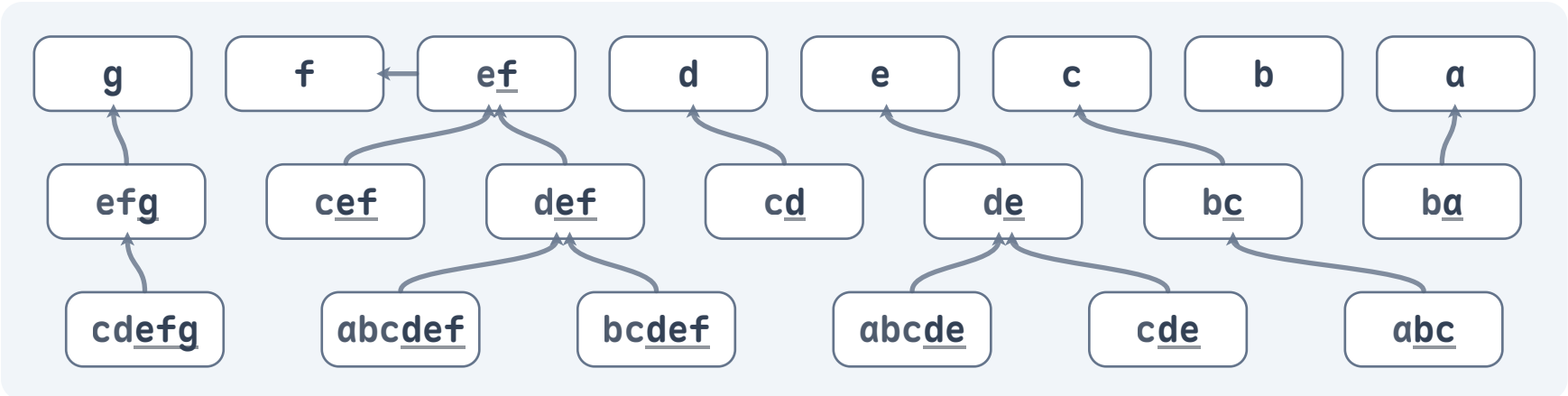


Predecessors & Successors

- Predecessor: $\text{pre}(t)$
- Successor: $\text{suc}(t)$
- $t = \text{concatenate}(\text{pre}(t), \text{suc}(t))$
- Deterministic & token-wise truncation ->
  - The relationship is **unique**!



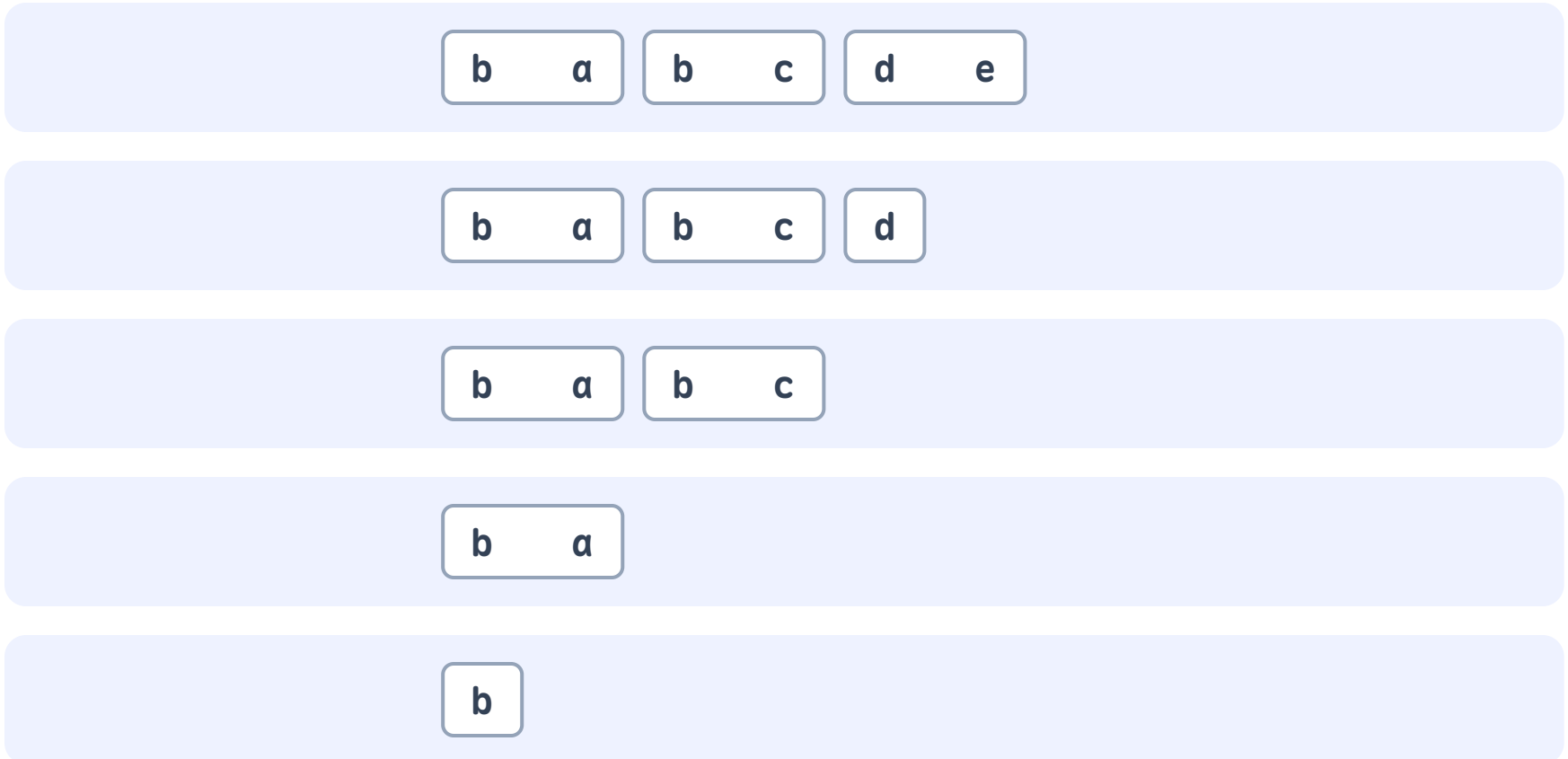
Successor Forest



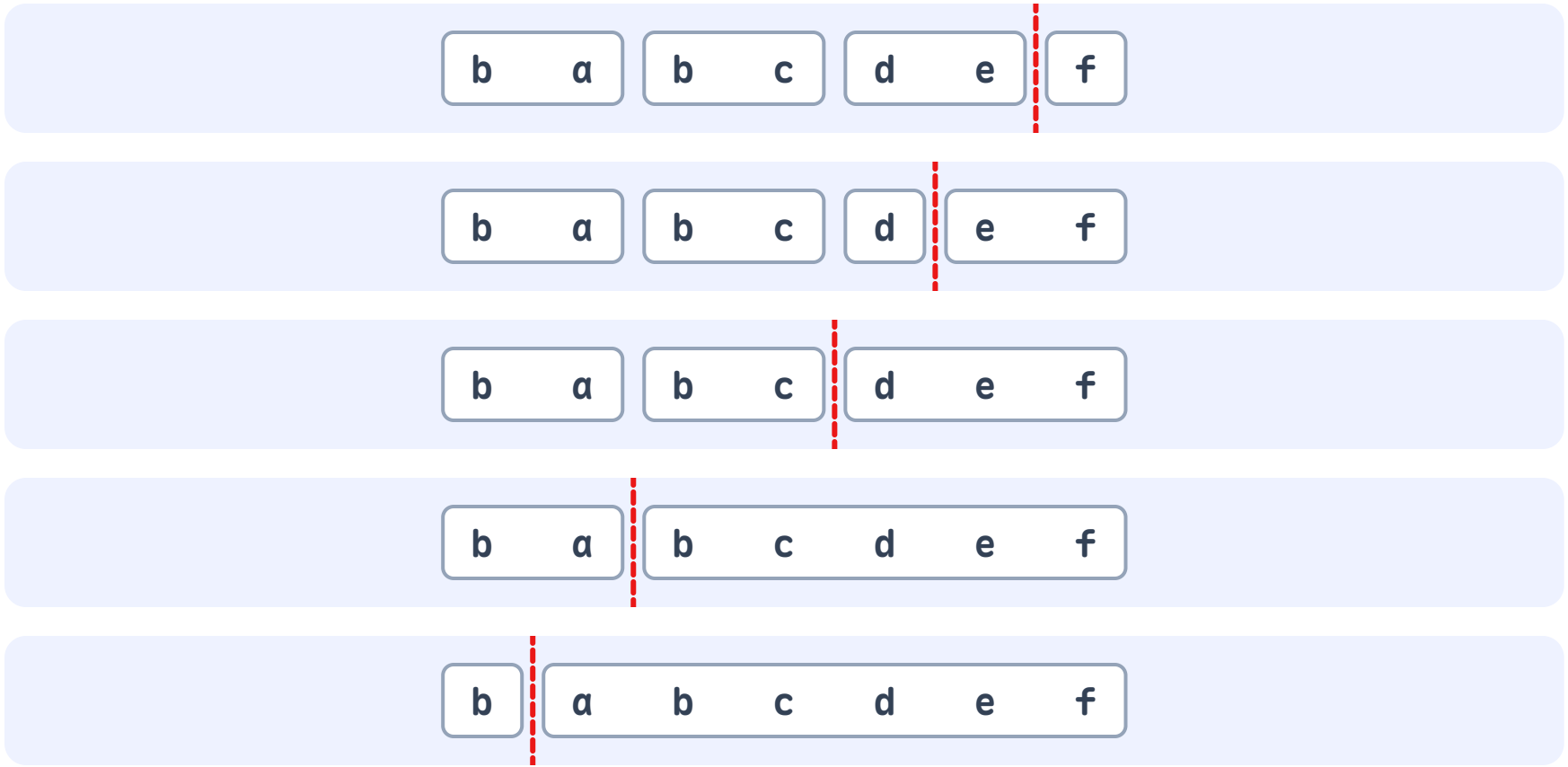
Successor Forest

Merge Rules

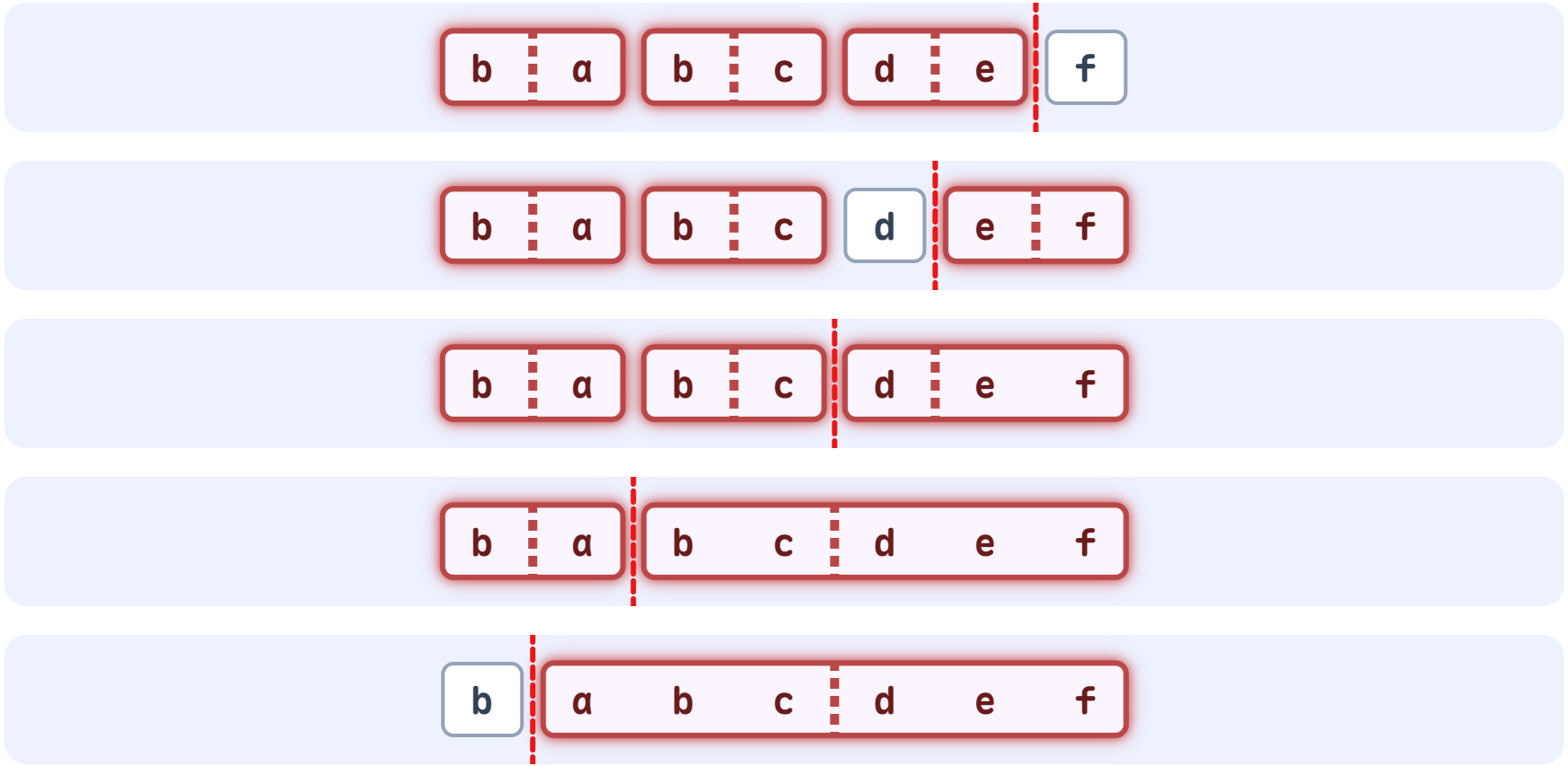
Which One is the Answer?



Which One is the Answer?



Which One is the Answer?



Which One is the Answer?

b a b c d e f

b a b c d e f

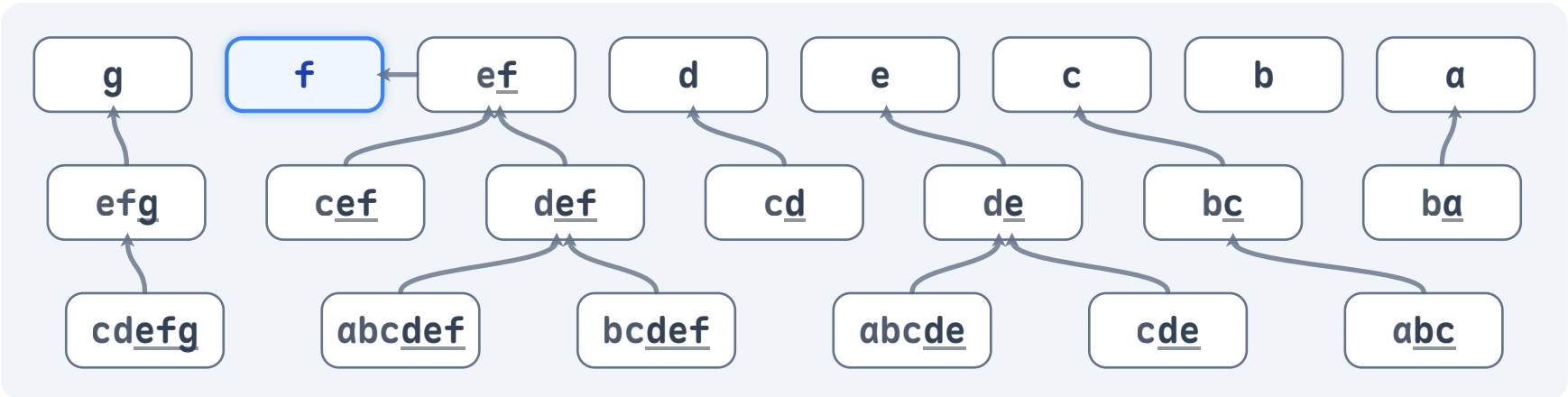
b a b c d e f

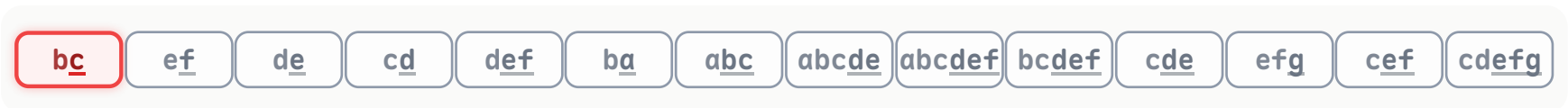
b a b c d e f

b a b c d e f

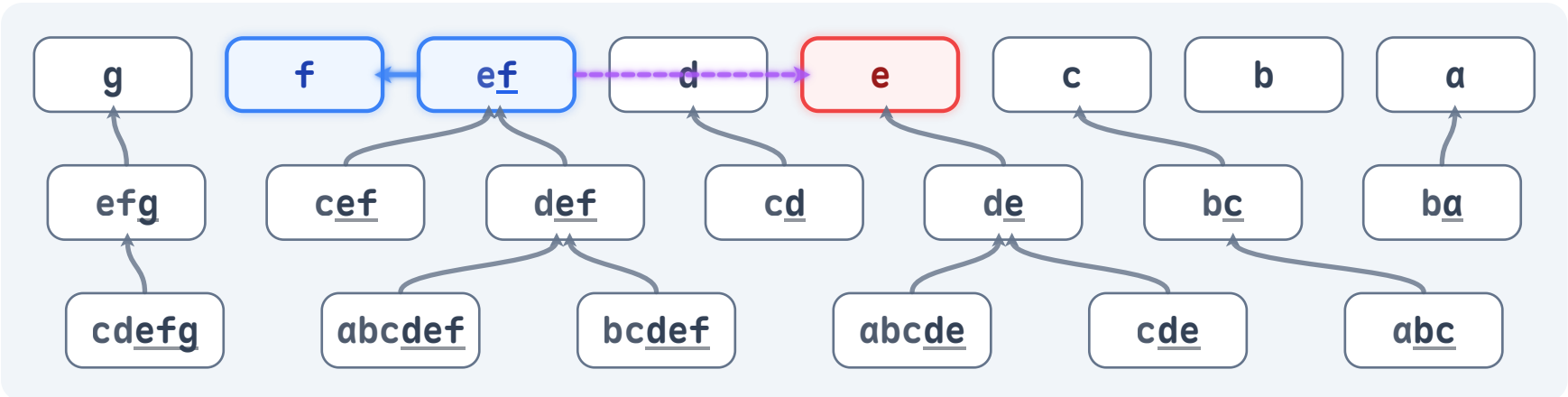
Monotonic Path: Chain to the "Last Token"

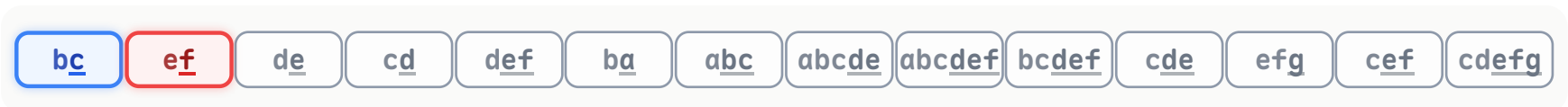
Monotonic Path: Chain to the "Last Token"



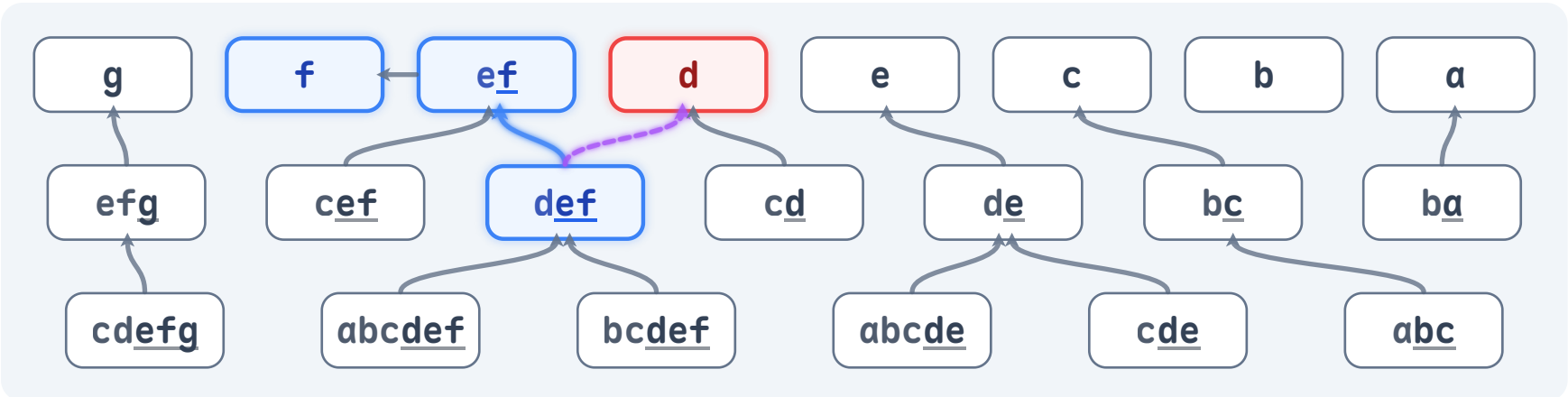


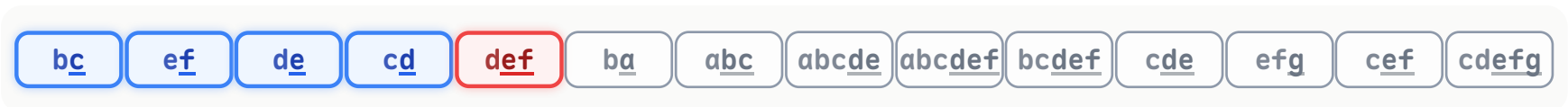
Monotonic Path: Chain to the "Last Token"



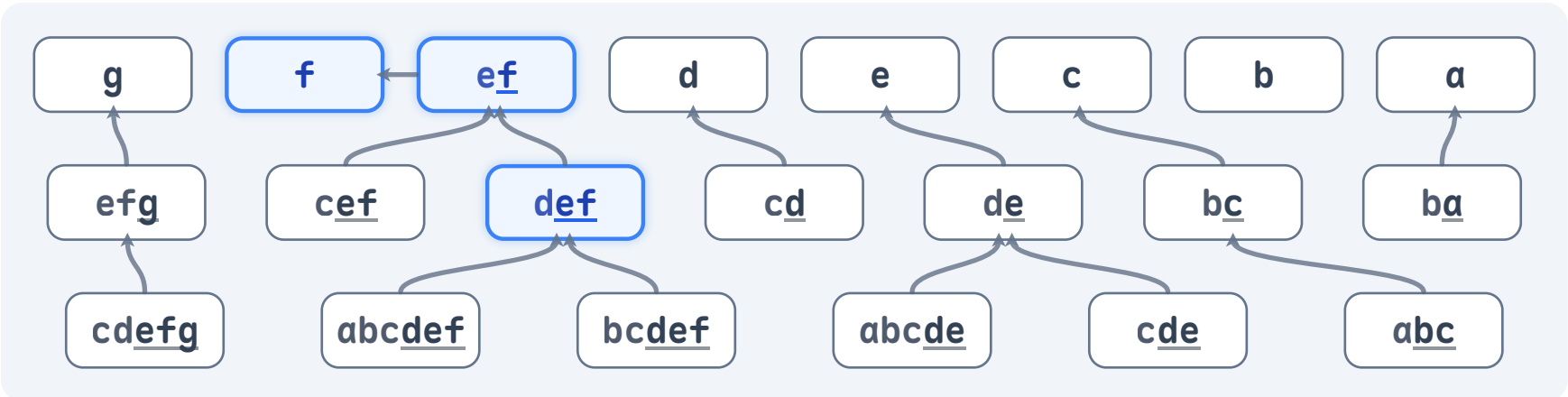


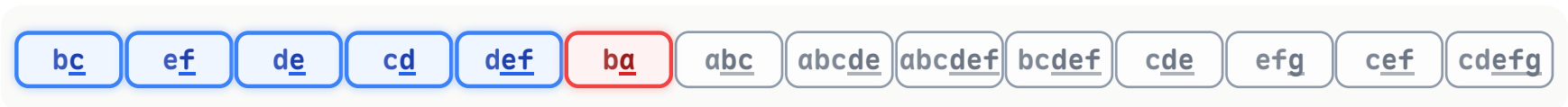
Monotonic Path: Chain to the "Last Token"



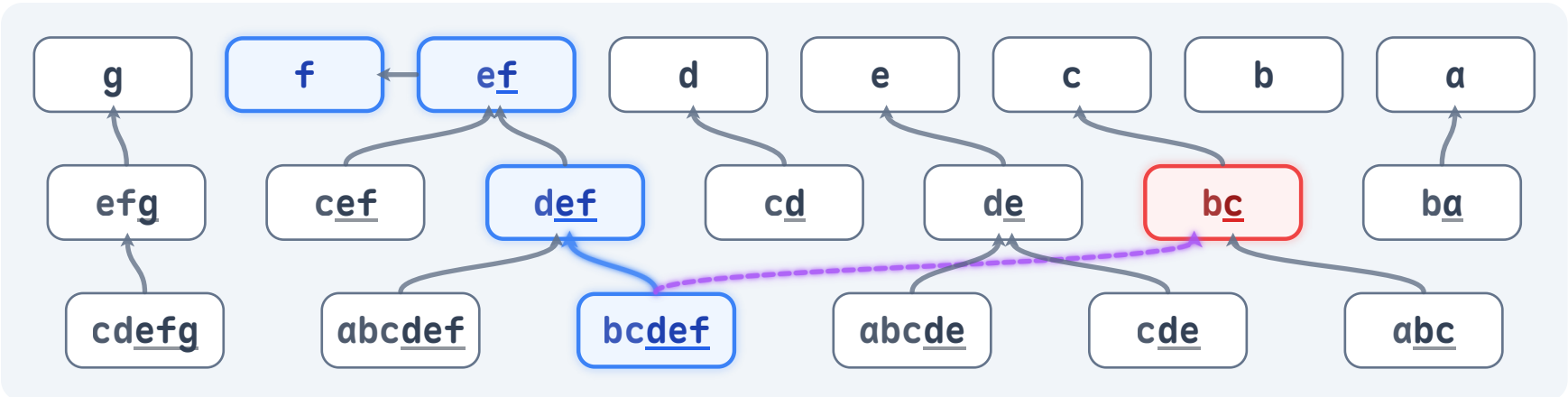


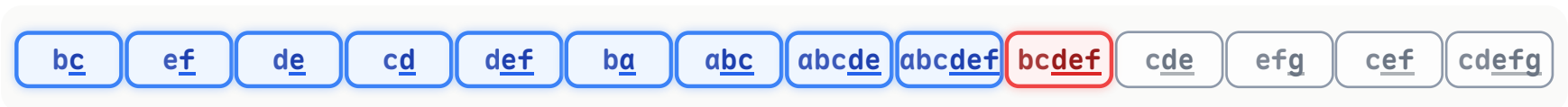
Monotonic Path: Chain to the "Last Token"



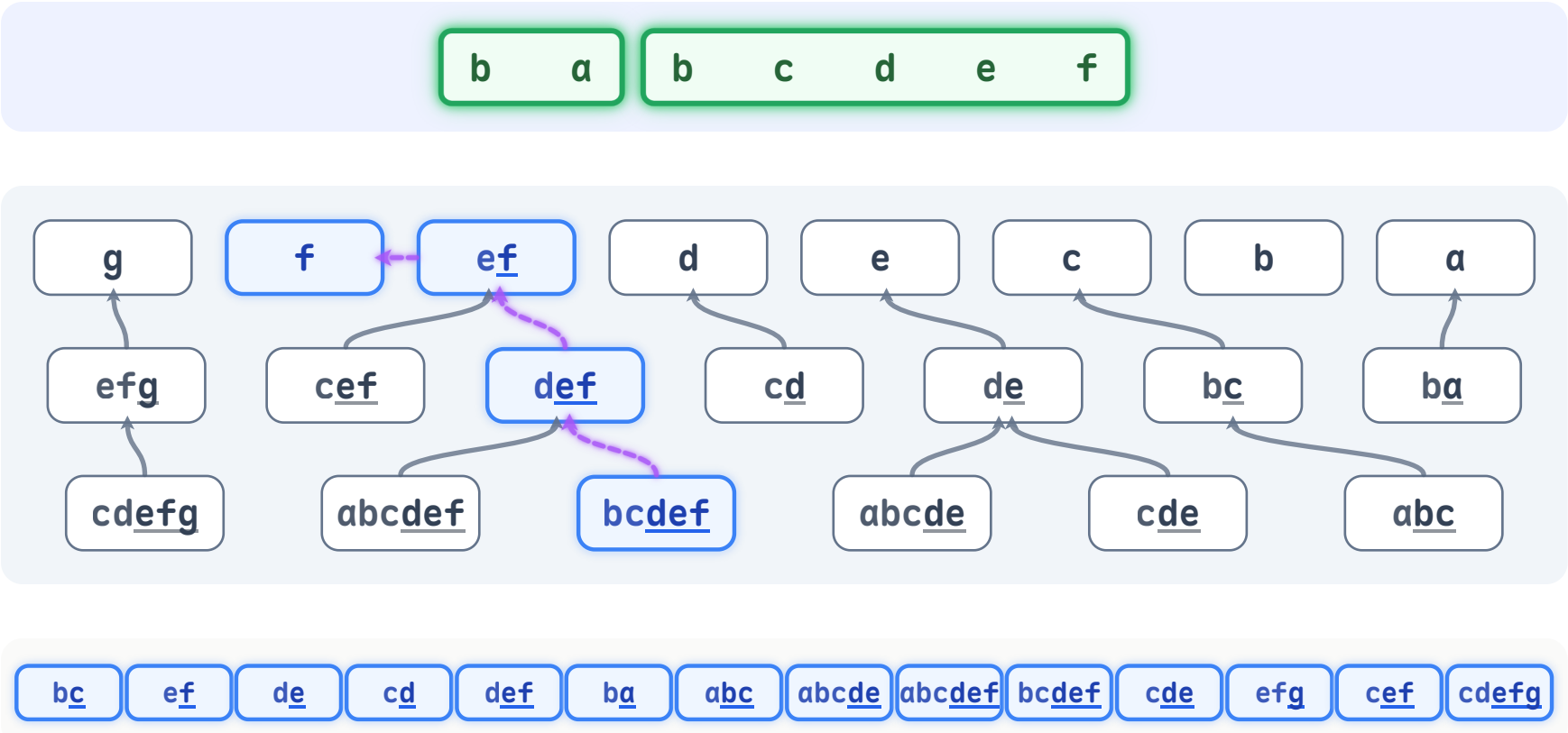


Monotonic Path: Chain to the "Last Token"

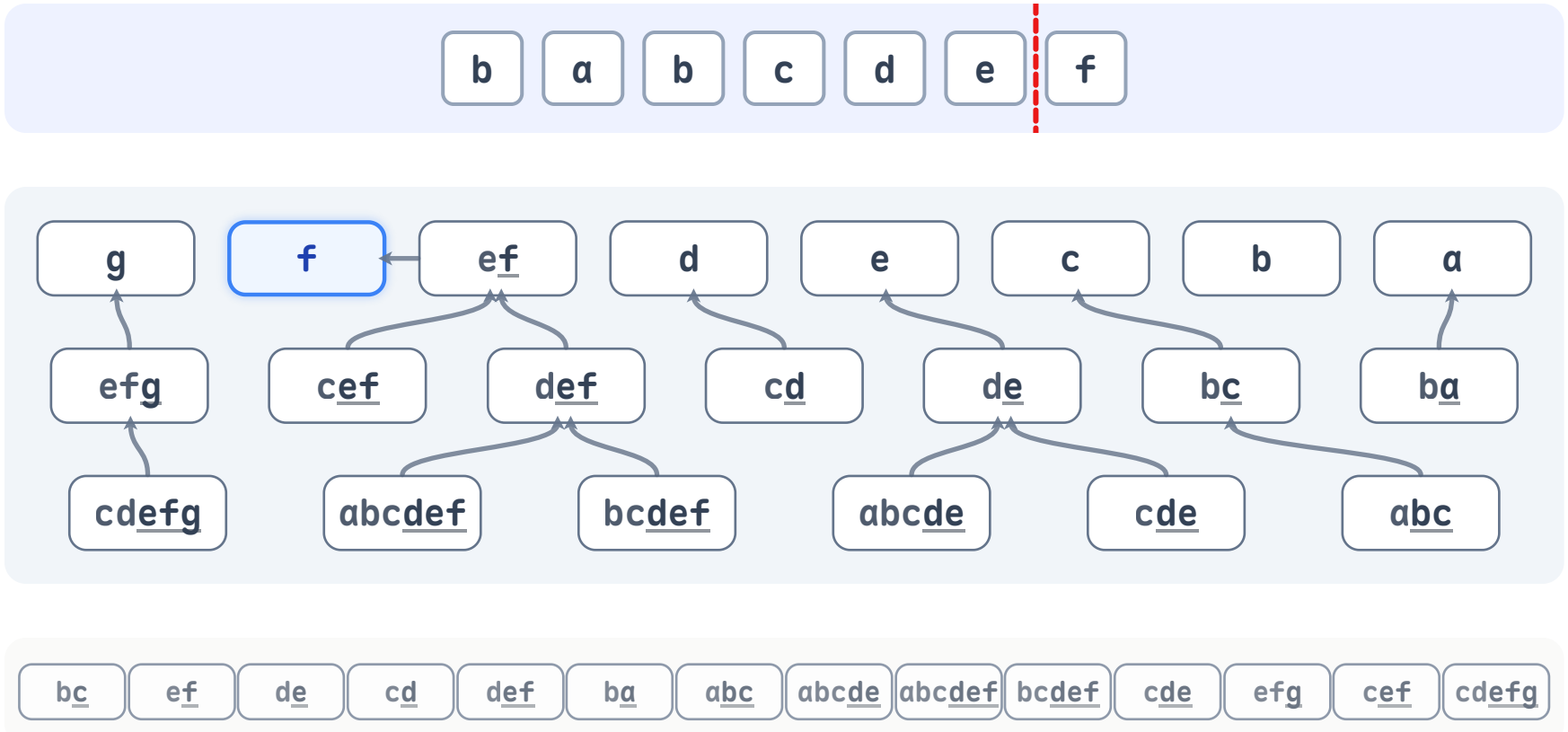




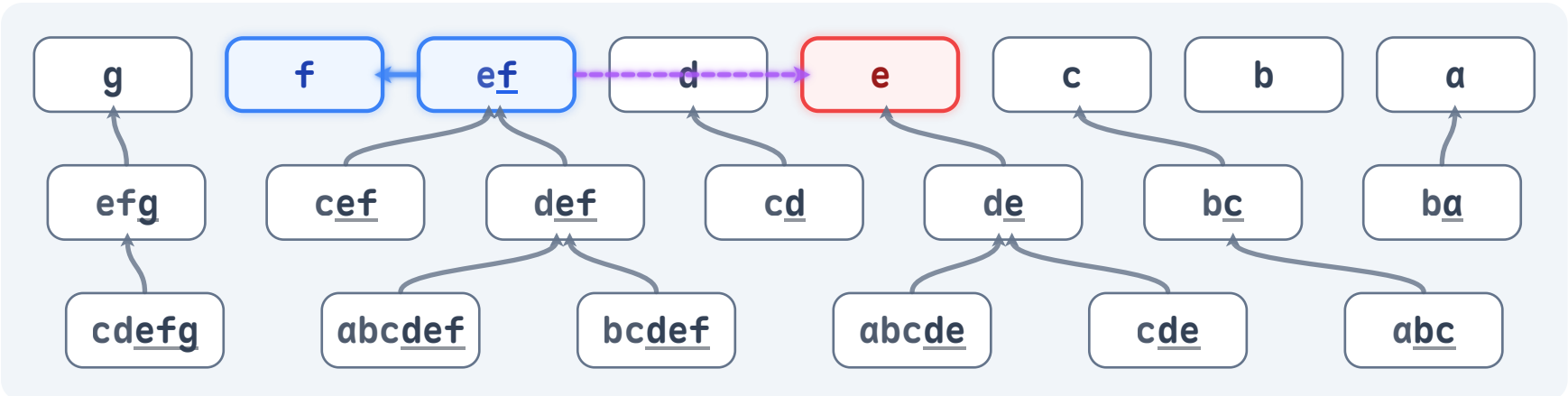
Monotonic Path: Chain to the "Last Token"

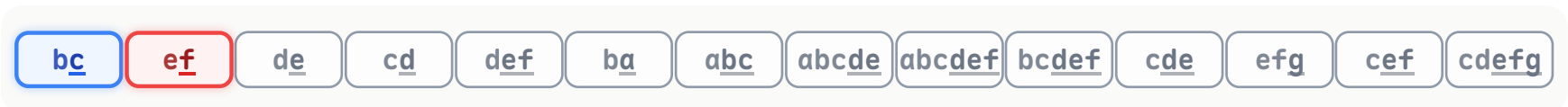


"Steal" the Tokens from the Previous States

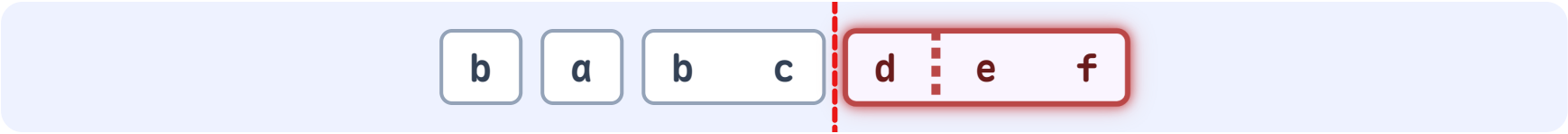


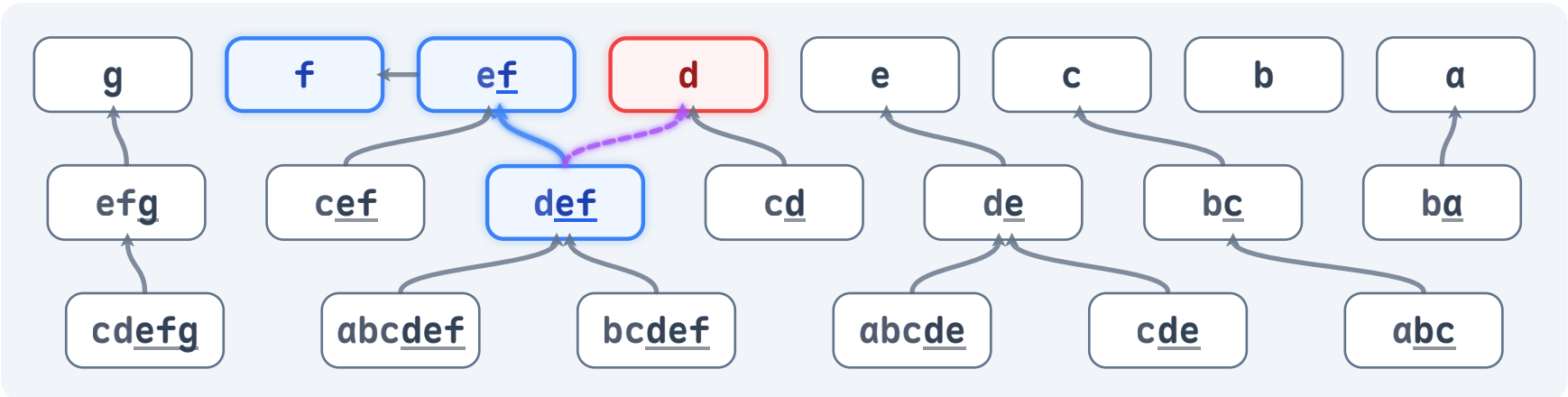
"Steal" the Tokens from the Previous States

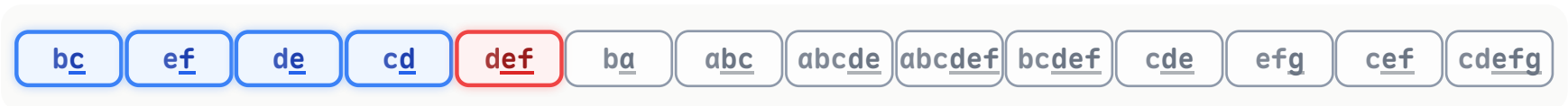




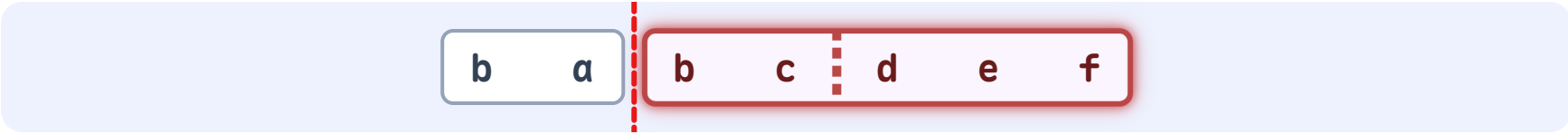
"Steal" the Tokens from the Previous States

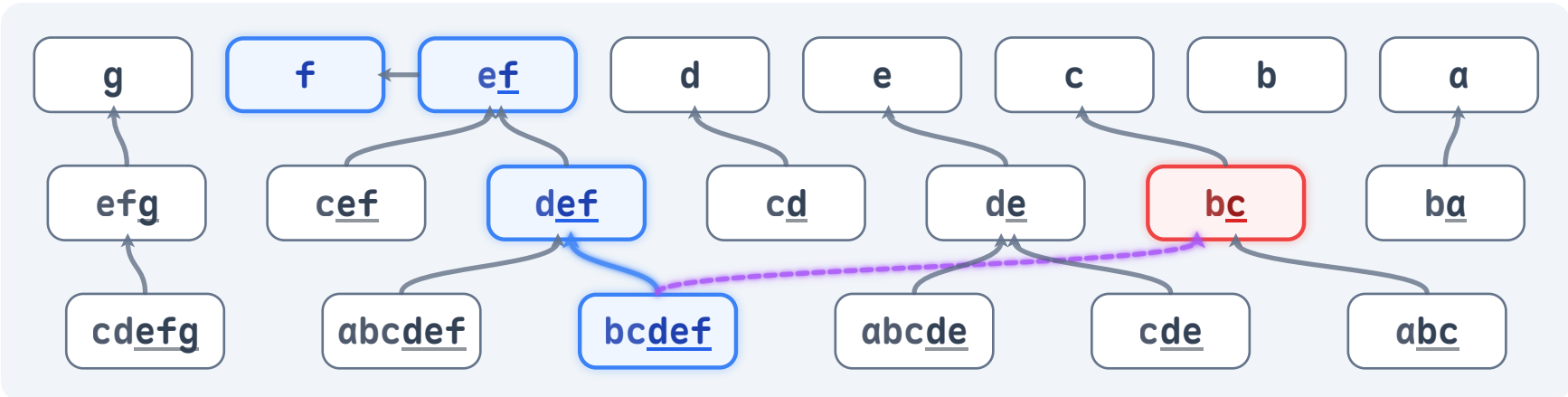






"Steal" the Tokens from the Previous States





Which One is the Answer?

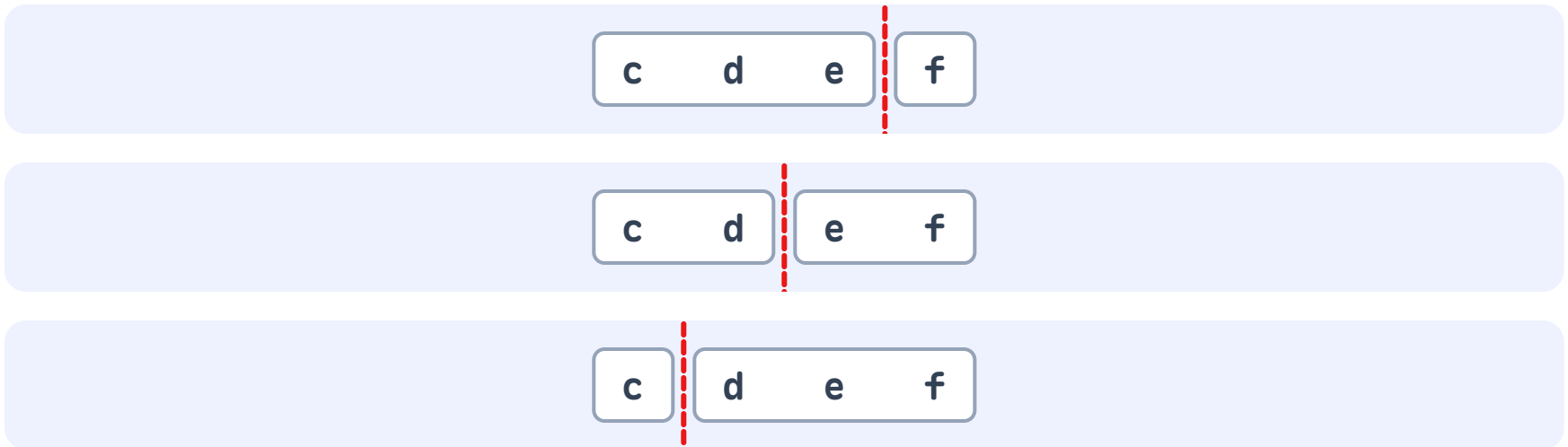
c d e

c d

c

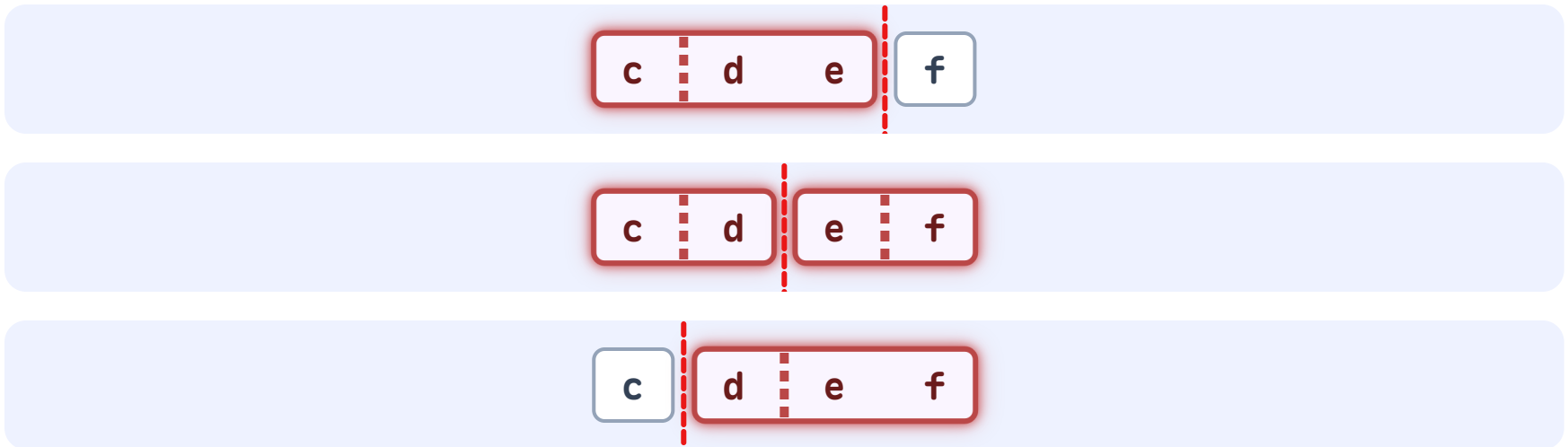
bc ef de cd def ba abc abcde abcdef bcdef cde efg cef cdefg

Which One is the Answer?



bc ef de cd def ba abc abcde abcdef bcdef cde efg cef cdefg

Which One is the Answer?



bc ef de cd def ba abc abcde abcdef bcdef cde efg cef cdefg

Which One is the Answer?

c d e f

c d e f

c d e f

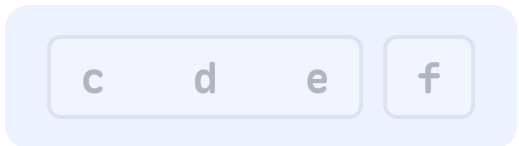
bc ef de cd def ba abc abcde abcdef bcdef cde efg cef cdefg

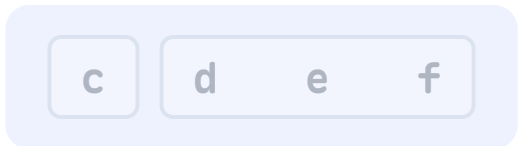
Priorities Rule the Game

Priorities Rule the Game

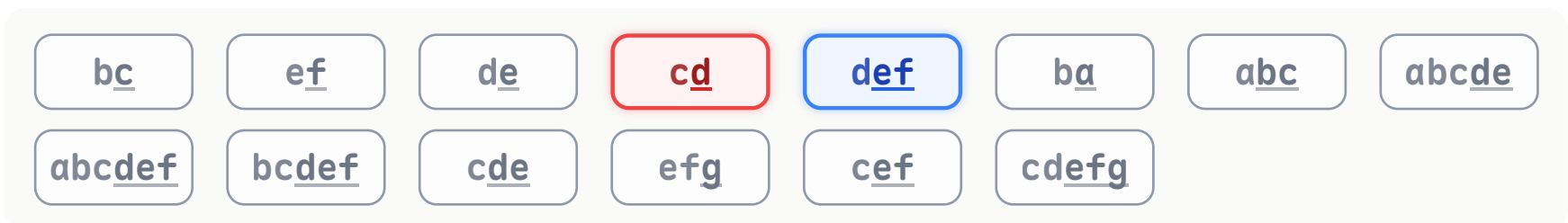
- `ef` is capable of "stealing" `e` from `cde`, where `de` has a *lower* priority than `ef`

Priorities Rule the Game



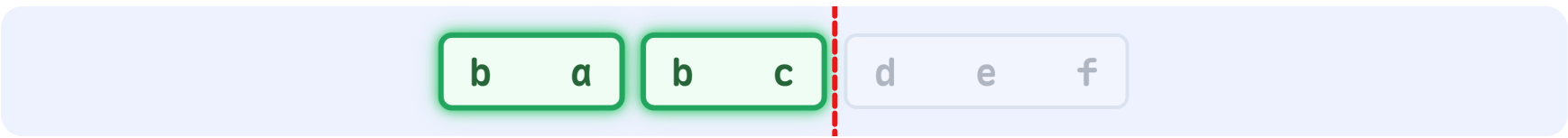


- `ef` is capable of "stealing" `e` from `cde`, where `de` has a *lower* priority than `ef`
- `def` is **not** capable of "stealing" `d` from `cd`, where `cd` has a *higher* priority than `def`



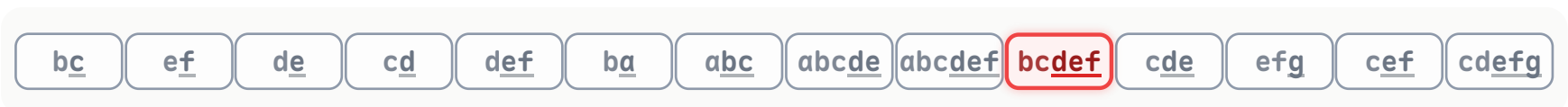
Prefix Last-Token Condition

Inspecting the token t = `bcdef`



The previous state k = the tokenization of `babc`

(Removing the successor of t from the suffix)

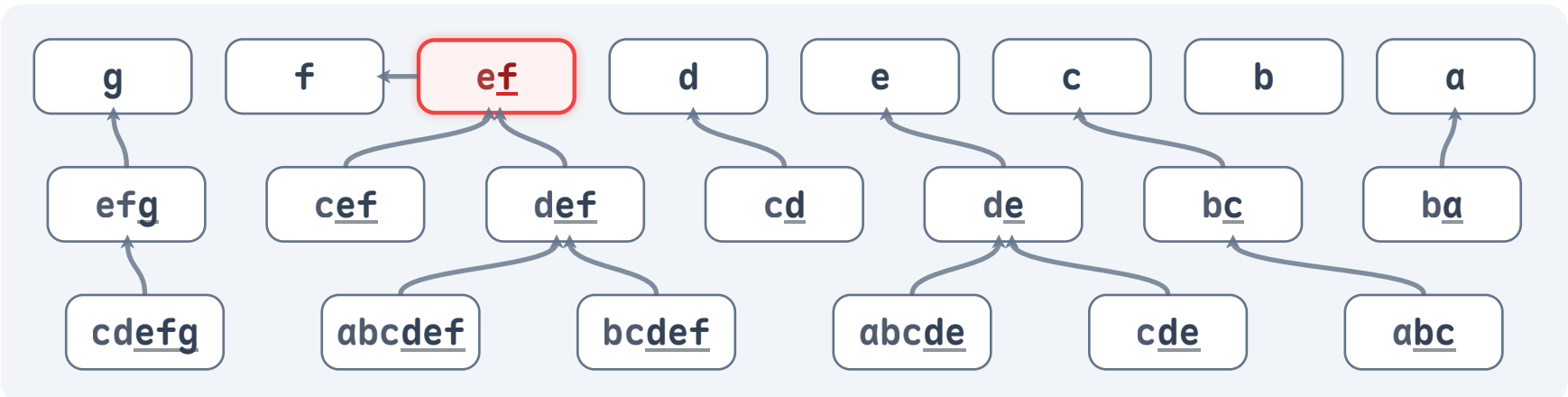


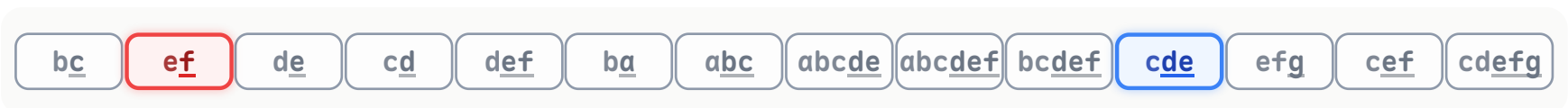
Prefix Last-Token Condition $\iff$ Monotonic Path

Theorem 4.2: Monotonic Path Property

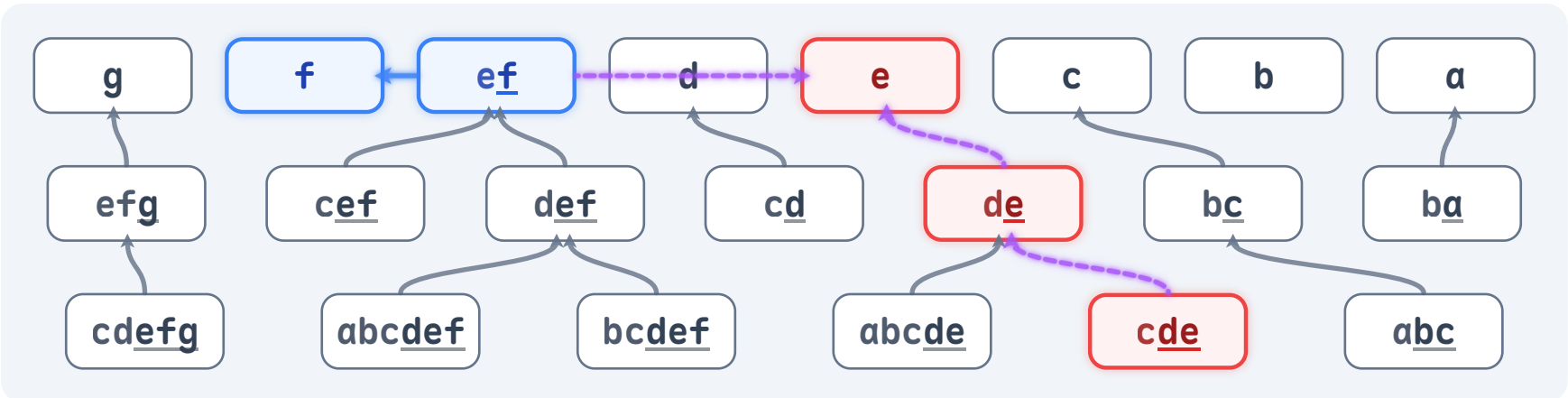
With respect to the input string s, a token t is on the *Monotonic Path*, **if and only if** it satisfies the *Prefix Last-Token Condition*.

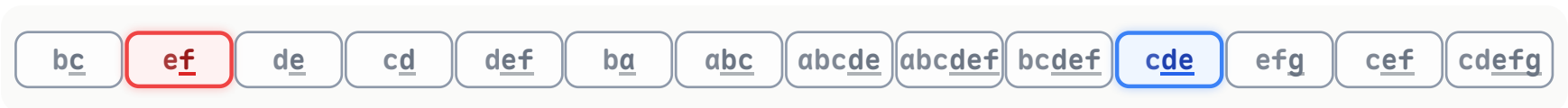
Prefix Last-Token Condition: t = ef



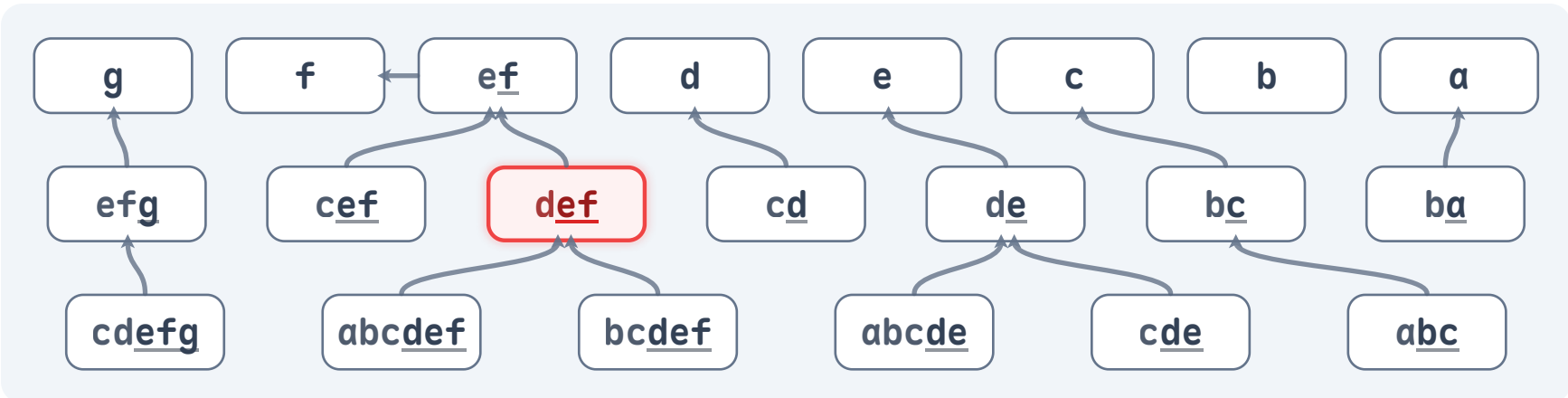


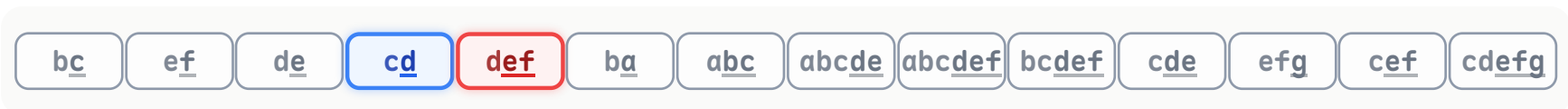
Prefix Last-Token Condition: t = ef



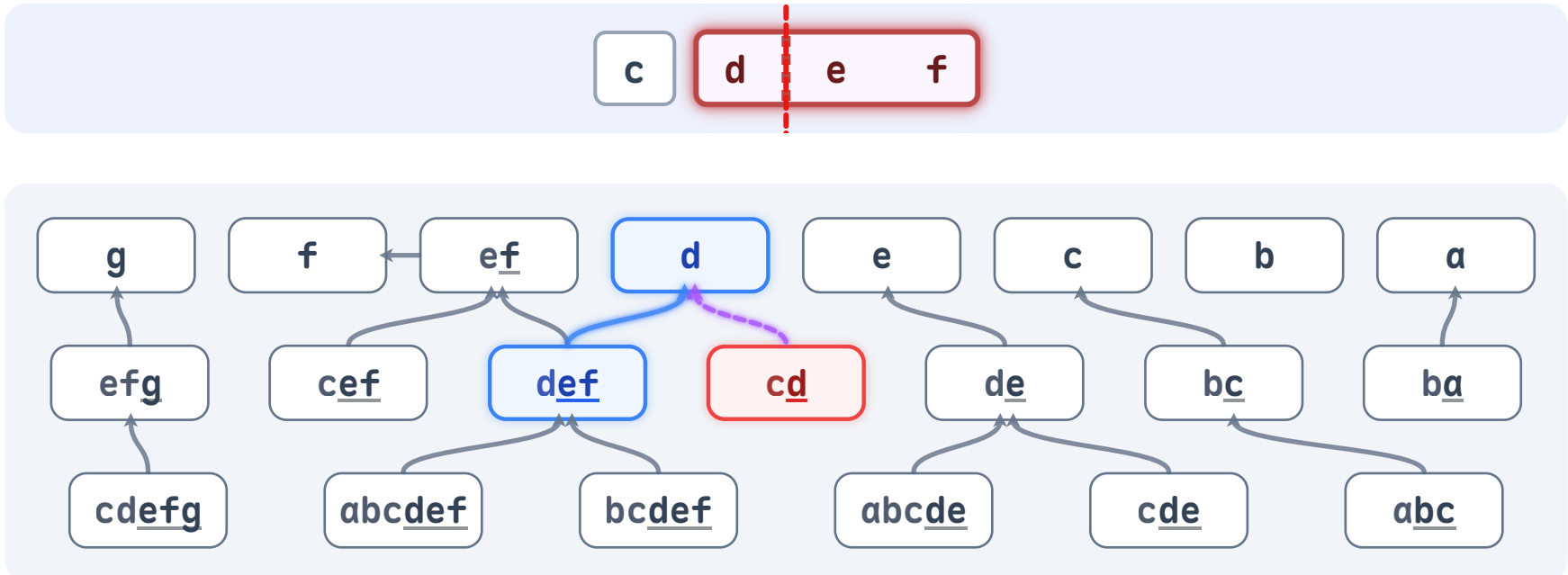


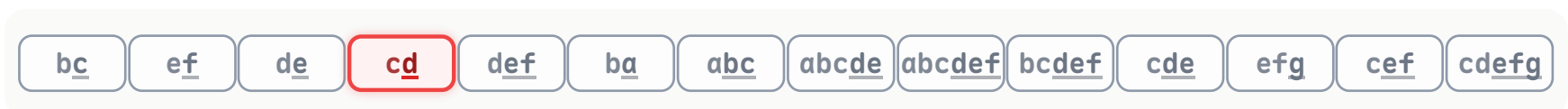
Prefix Last-Token Condition: t = def



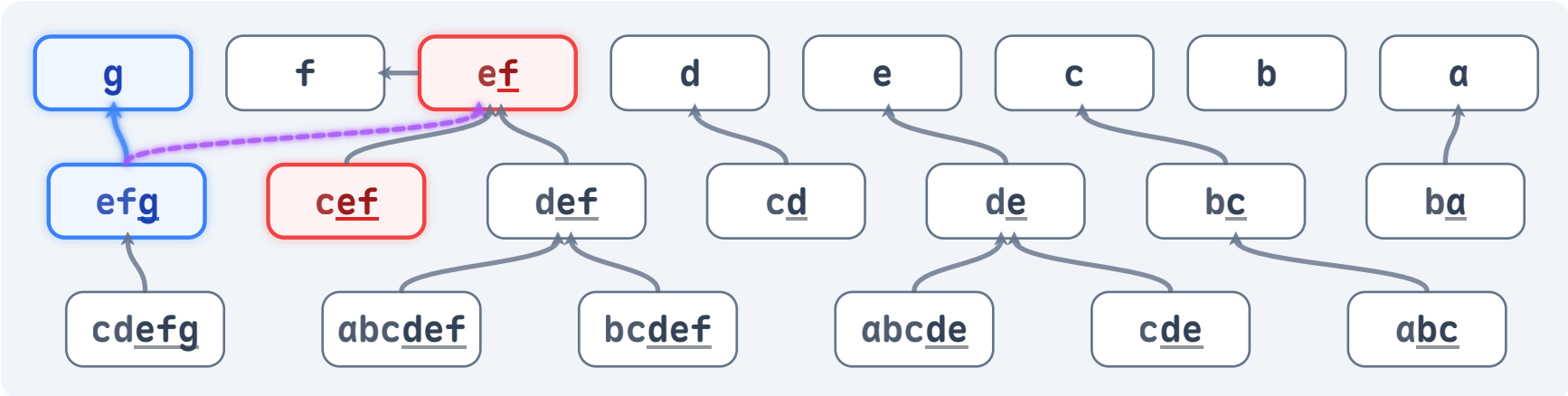


Prefix Last-Token Condition: t = def

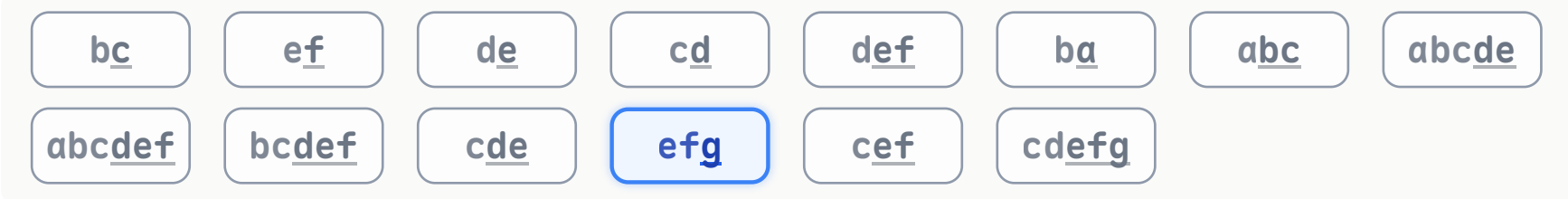




Valid "Last Tokens of Previous States": $\mathcal{C}_t$

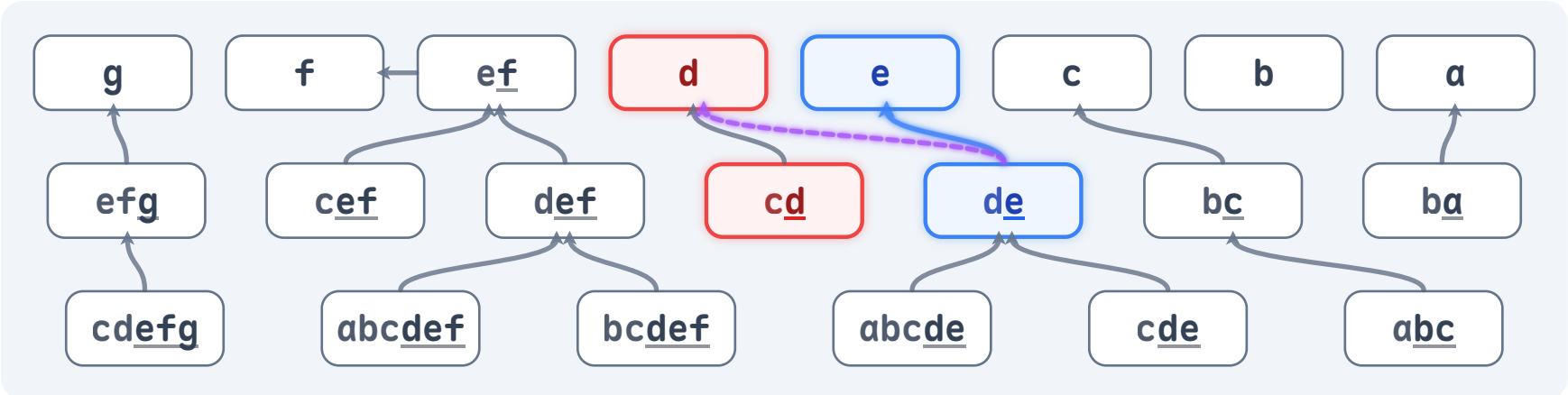


$\mathcal{C}_t$: Candidates (in **red**) as the "last tokens" of the previous states for the token t



Merge Rules

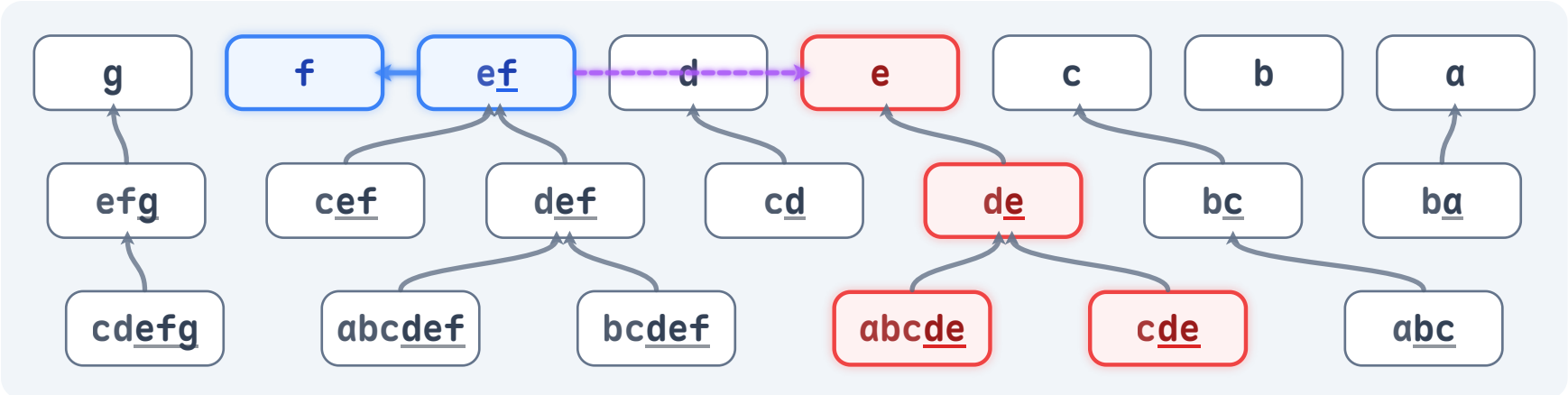
Valid "Last Tokens of Previous States": $\mathcal{C}_t$



$\mathcal{C}_t$: Candidates (in **red**) as the "last tokens" of the previous states for the token t

Merge Rules

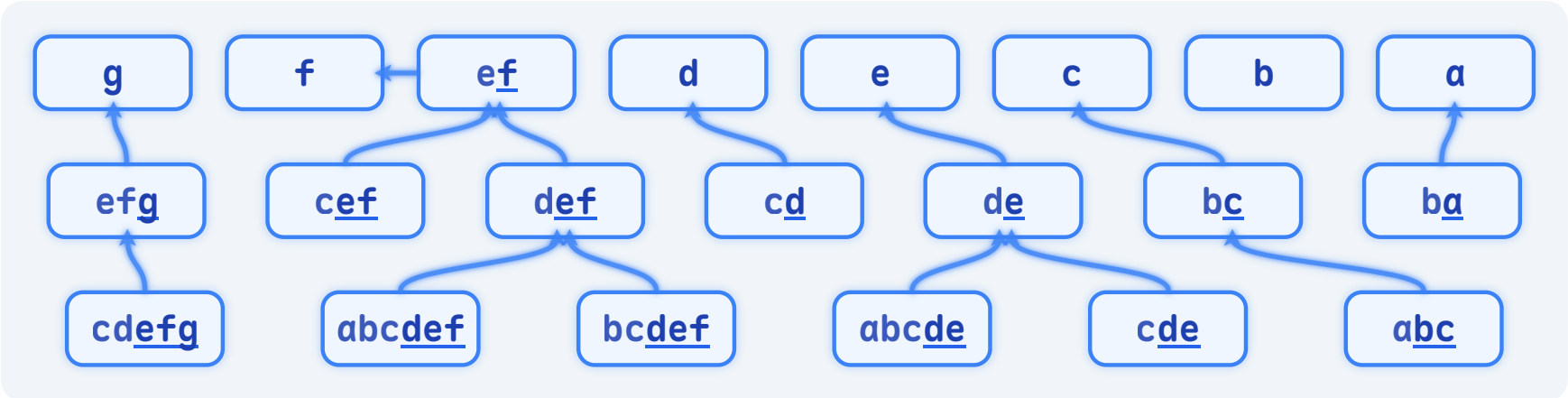
Valid "Last Tokens of Previous States": $\mathcal{C}_t$



$\mathcal{C}_t$: Candidates (in **red**) as the "last tokens" of the previous states for the token t

Merge Rules

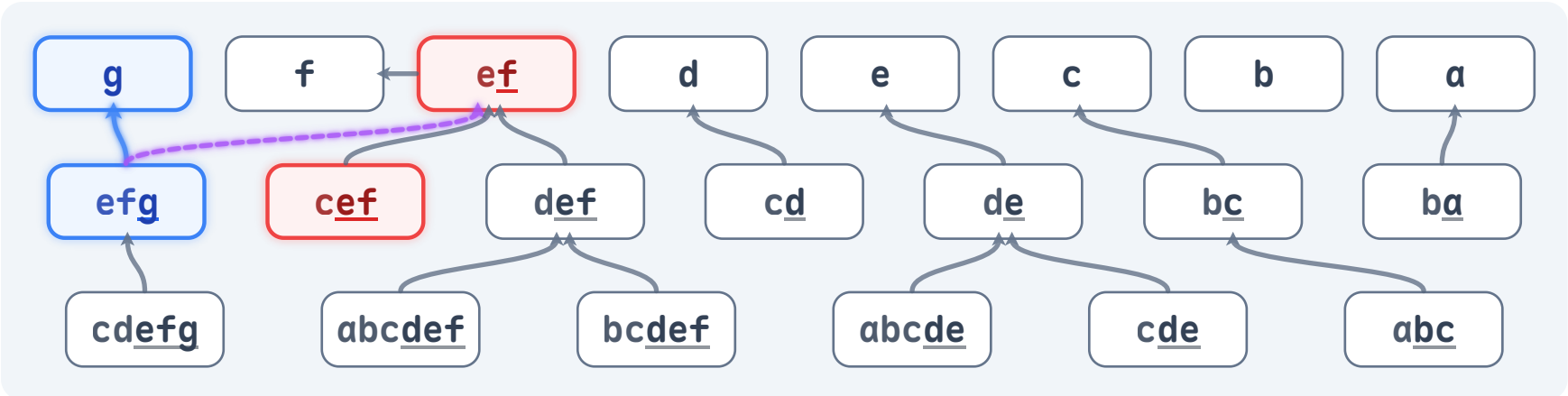
Linearization with Pre-order DFS



DFS the Successor Forest, visiting lower-priority children first

Pre-order DFS Linearization (DFS Numbering)

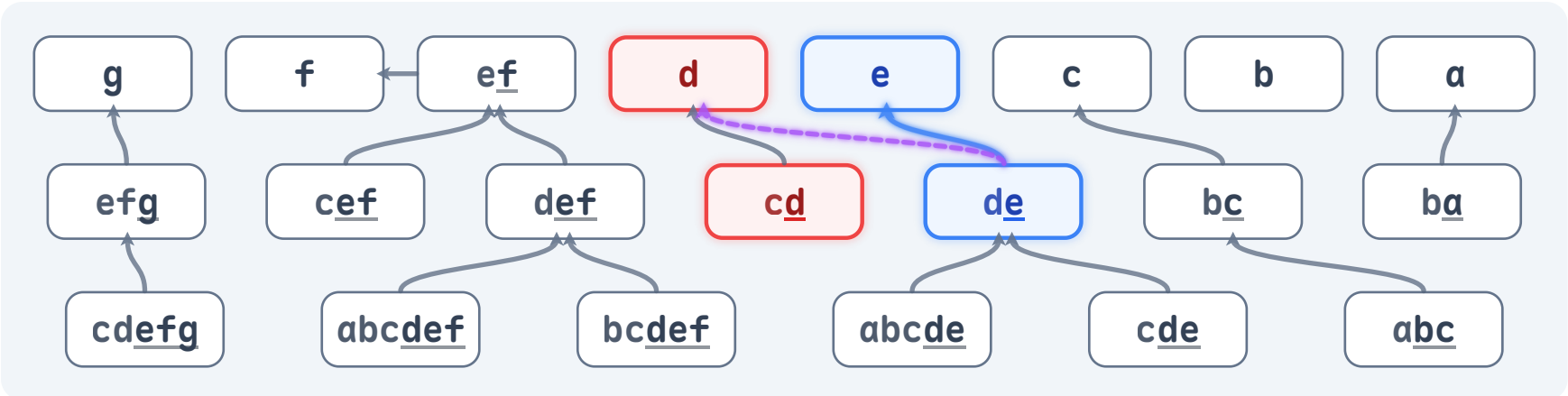
$\mathcal{C}_t \iff$ Intervals of Linearization



$\mathcal{C}_t$: Candidates (in **red**) as the "last tokens" of the previous states for the token t

Pre-order DFS Linearization (DFS Numbering)

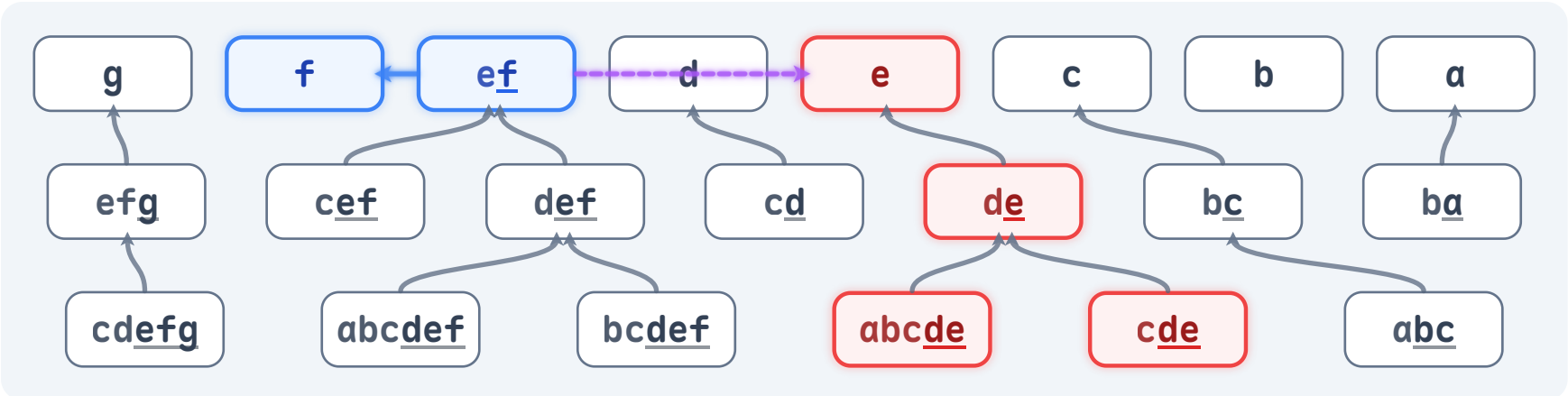
$\mathcal{C}_t \Longleftrightarrow$ Intervals of Linearization



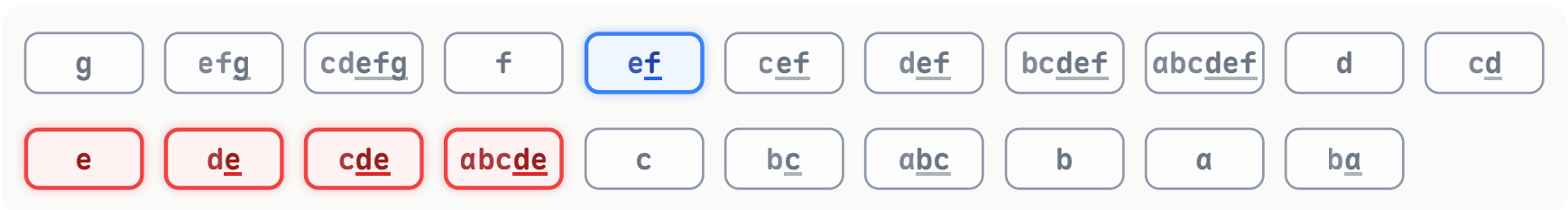
$\mathcal{C}_t$: Candidates (in **red**) as the "last tokens" of the previous states for the token t

Pre-order DFS Linearization (DFS Numbering)

$\mathcal{C}_t \Longleftrightarrow$ Intervals of Linearization

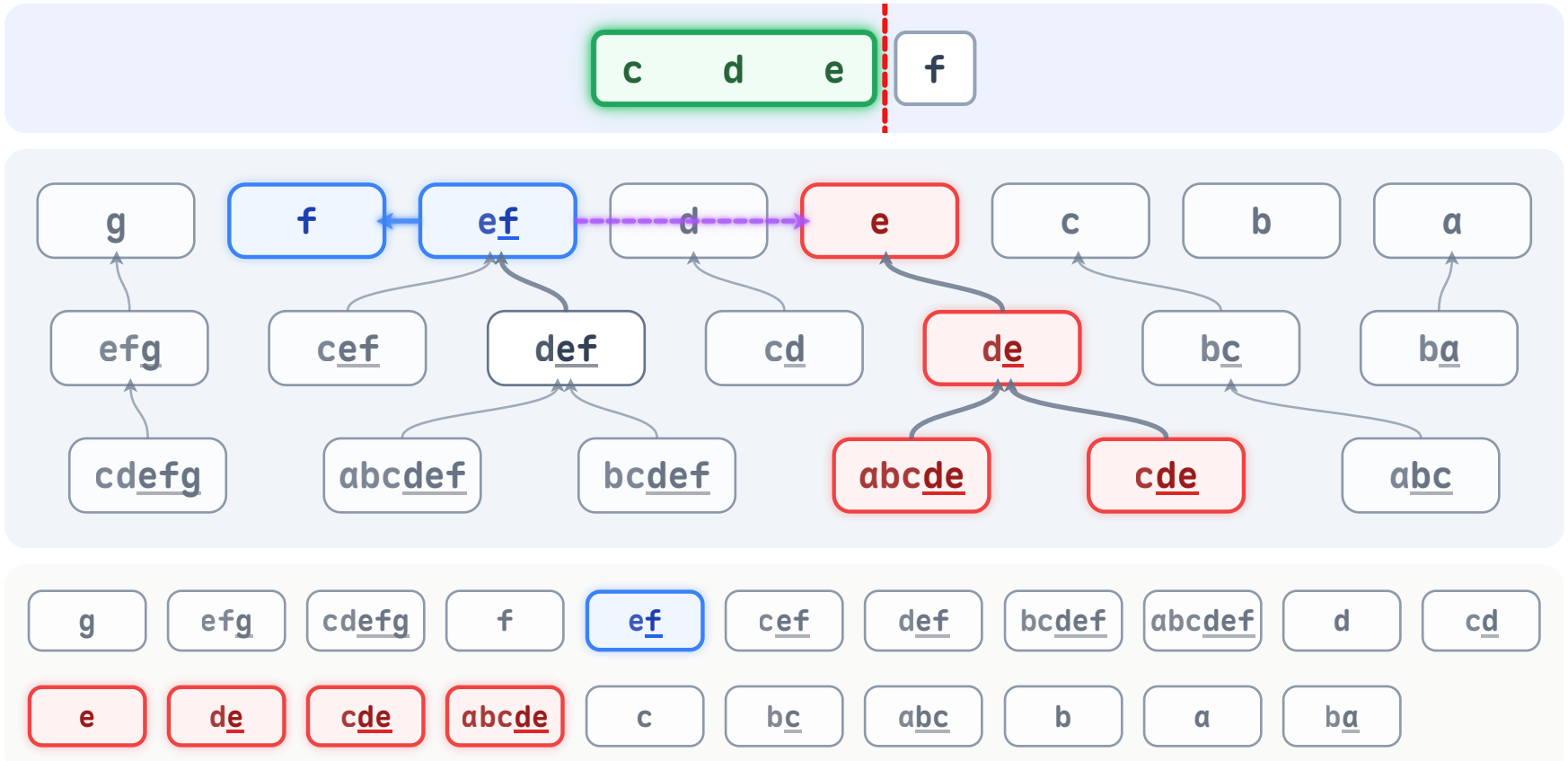


$\mathcal{C}_t$: Candidates (in **red**) as the "last tokens" of the previous states for the token t

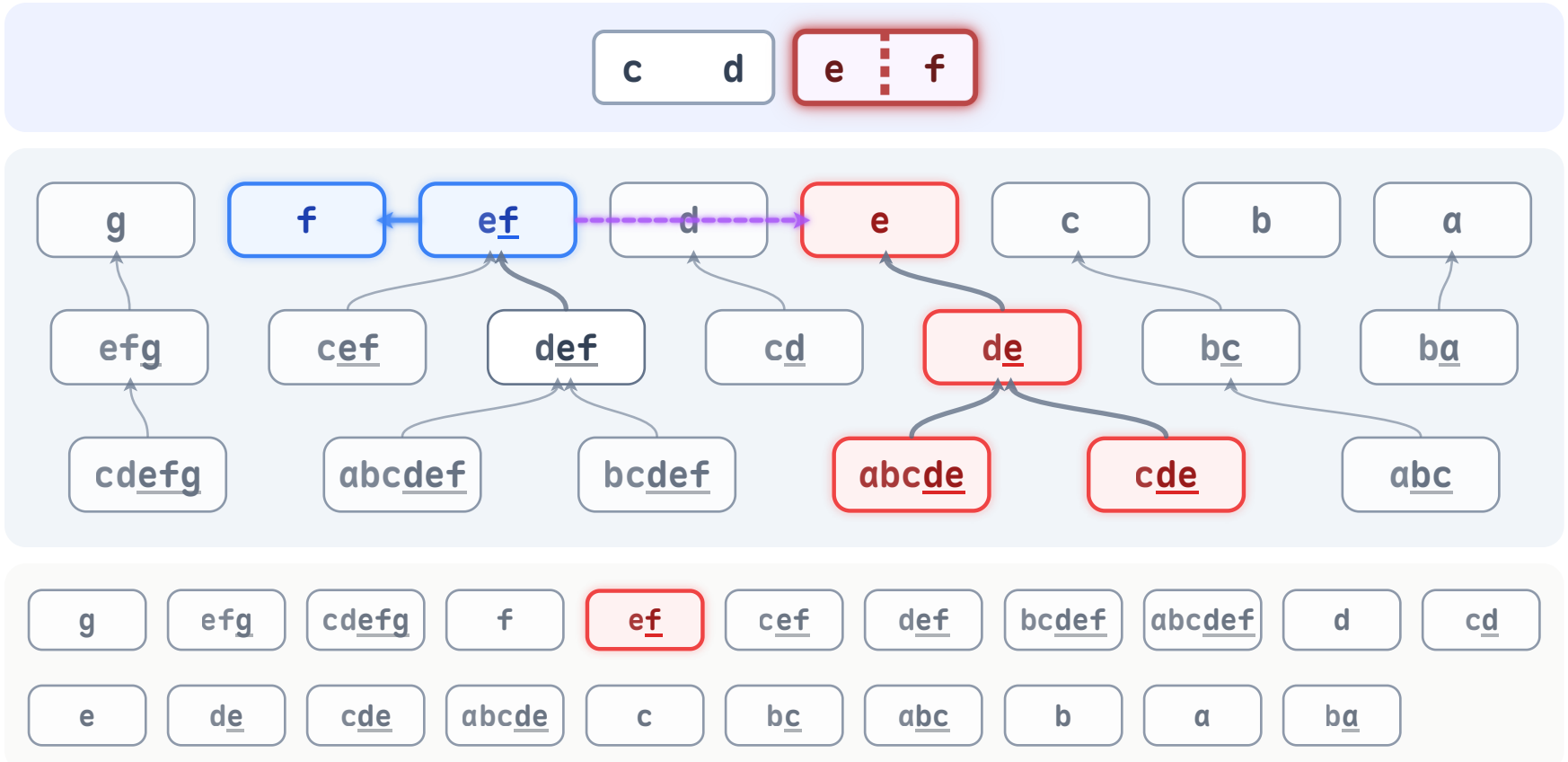


Pre-order DFS Linearization (DFS Numbering)

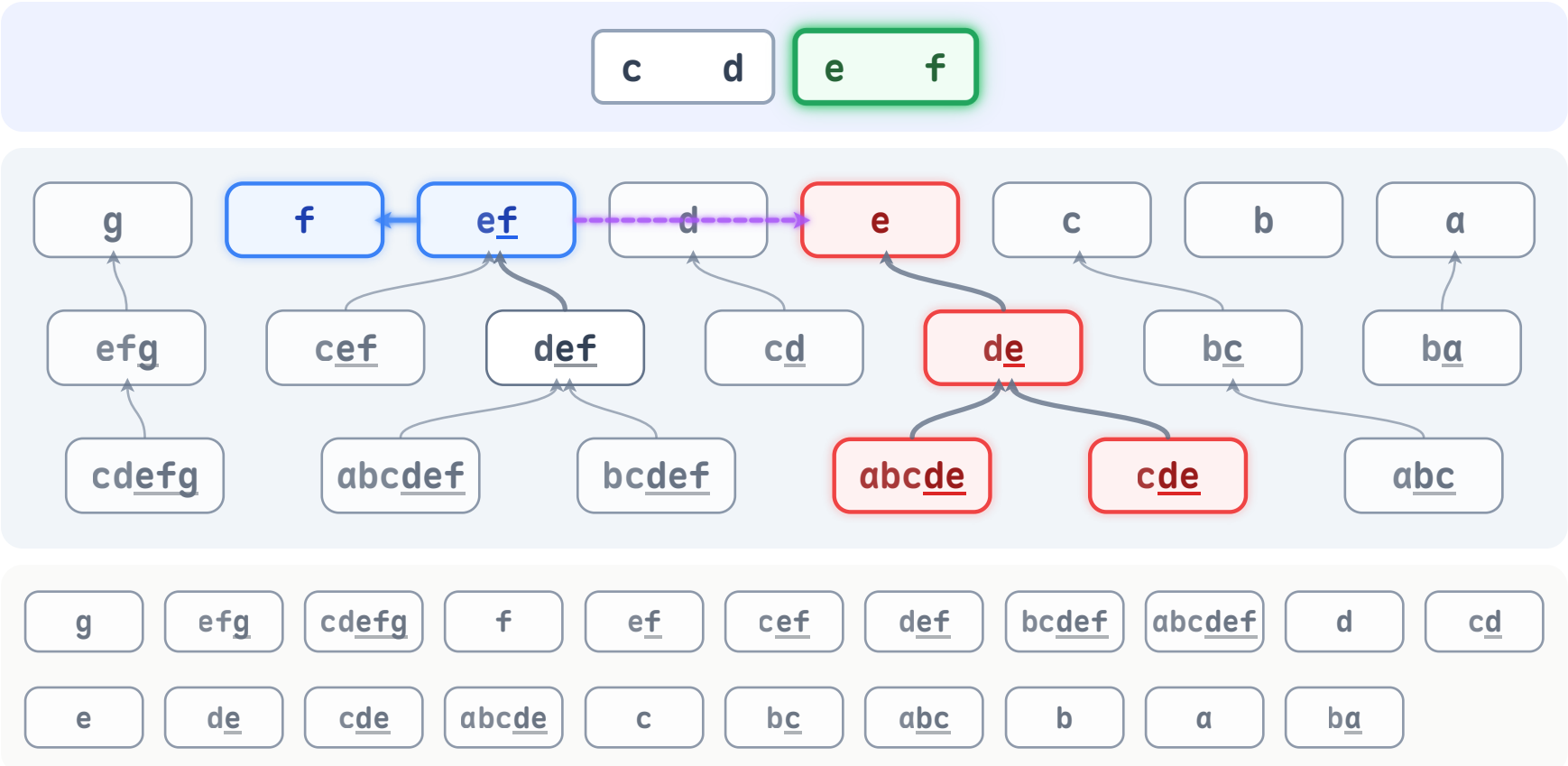
$\mathcal{C}_t \iff$ Intervals of Linearization: t = ef



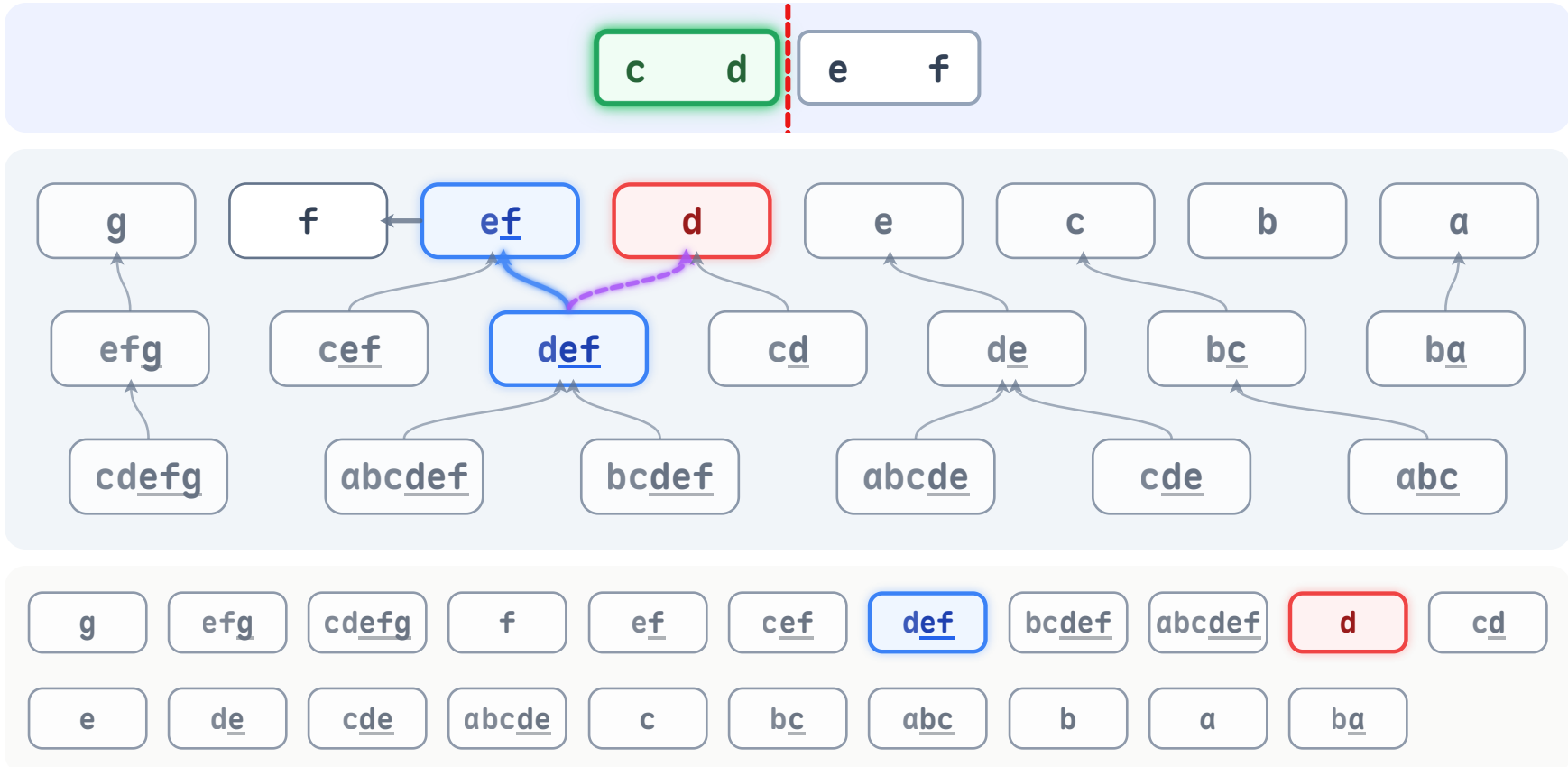
$\mathcal{C}_t \iff$ Intervals of Linearization: t = ef



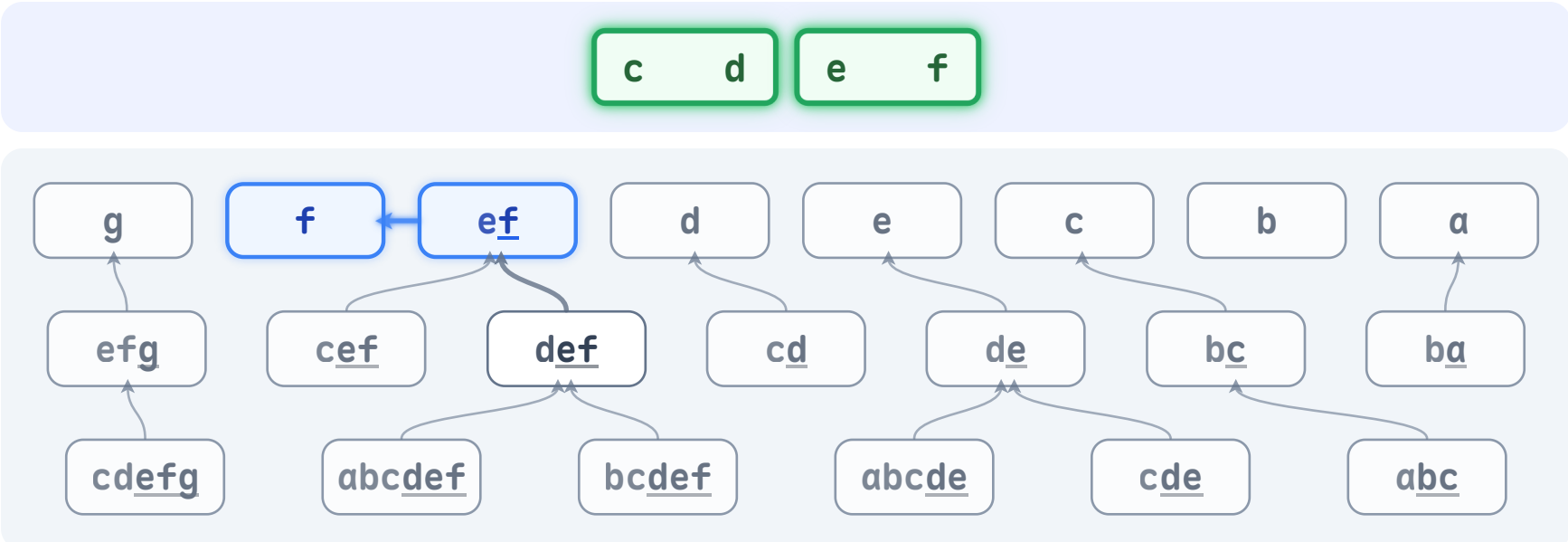
$\mathcal{C}_t \iff$ Intervals of Linearization: t = ef



$\mathcal{C}_t \iff$ Intervals of Linearization: t = def



$\mathcal{C}_t \iff$ Intervals of Linearization: t = def

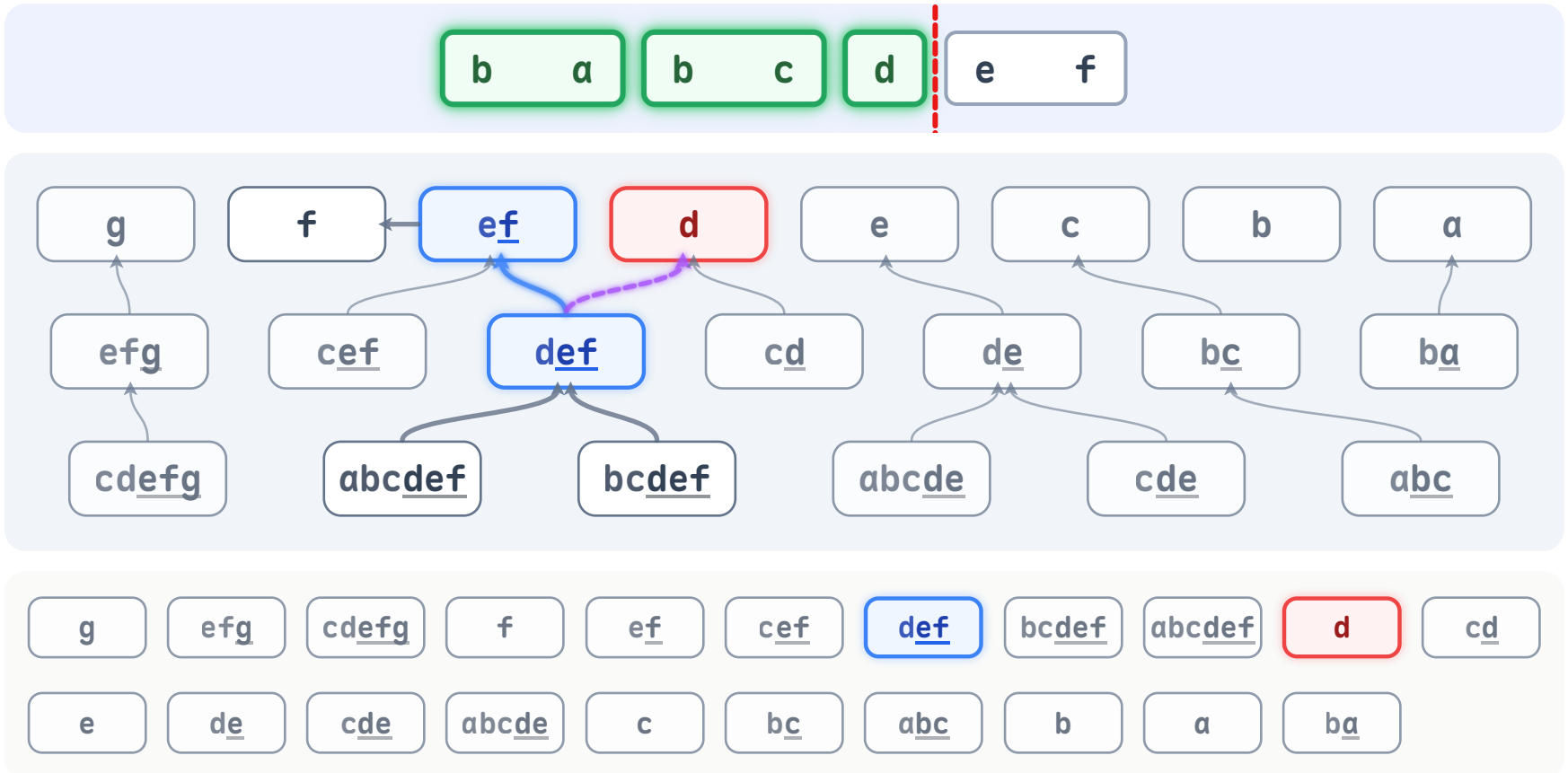


g efg cdefg f ef cef def bcdef abcdef d cd

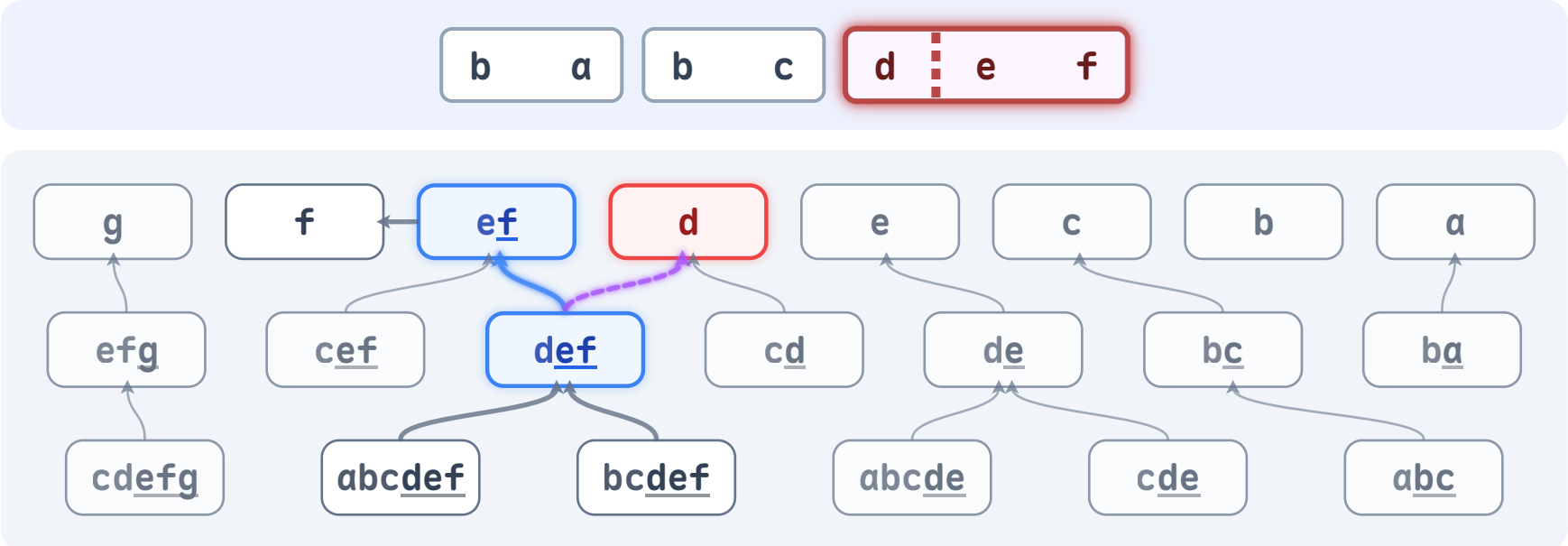
e de cde abcde c bc abc b a ba

Centroid Decomposition

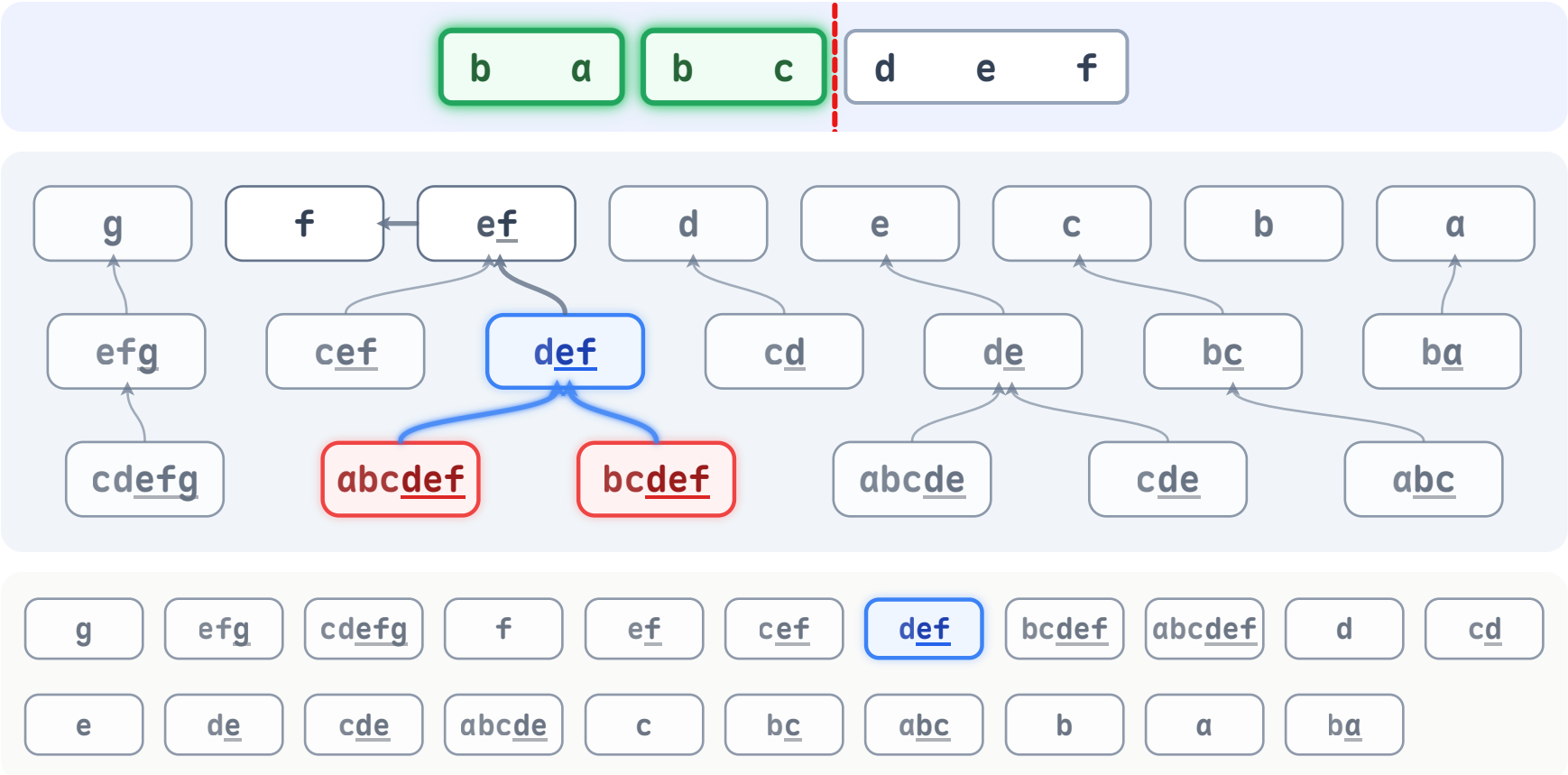
Centroid Decomposition



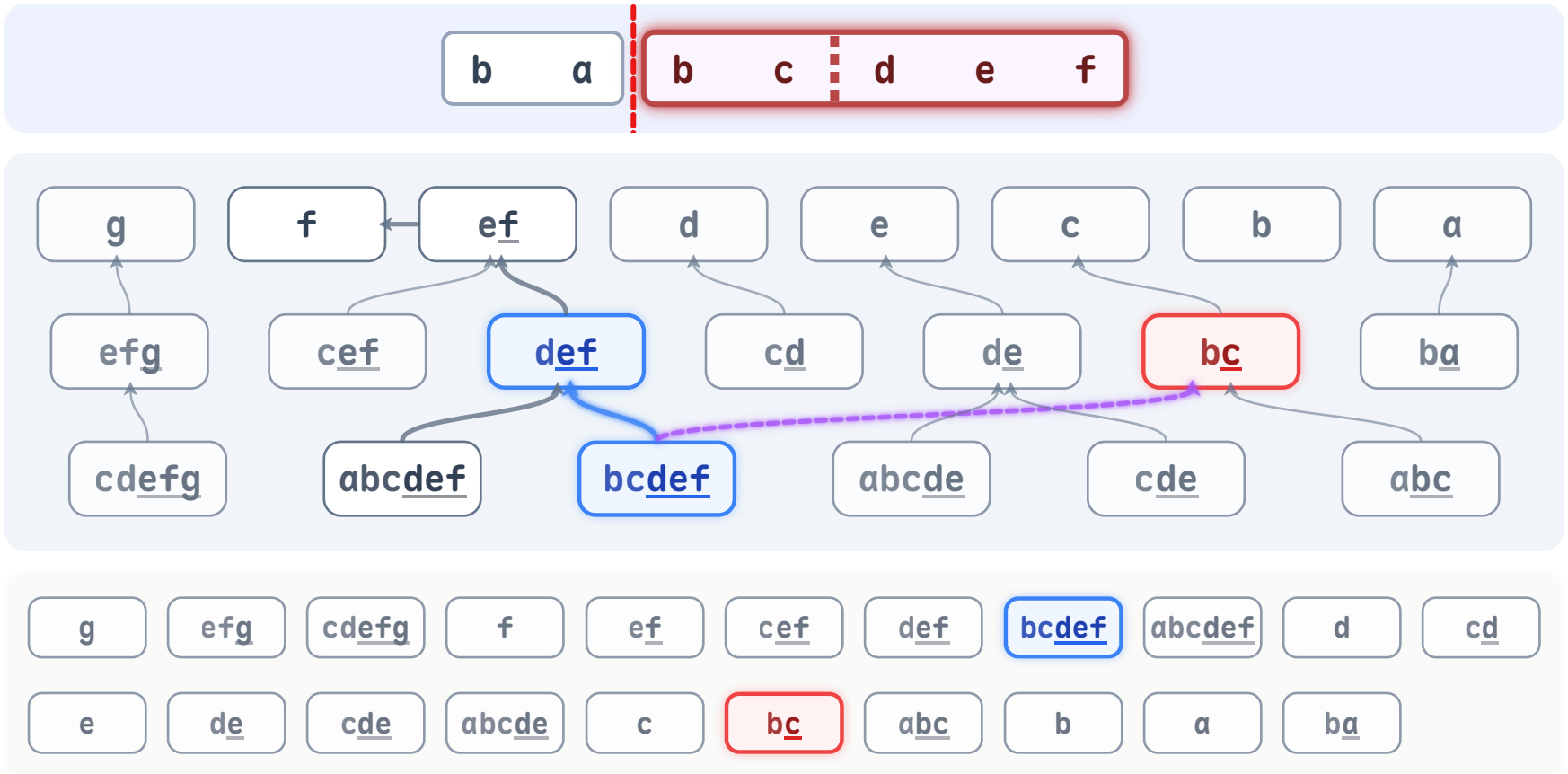
Centroid Decomposition



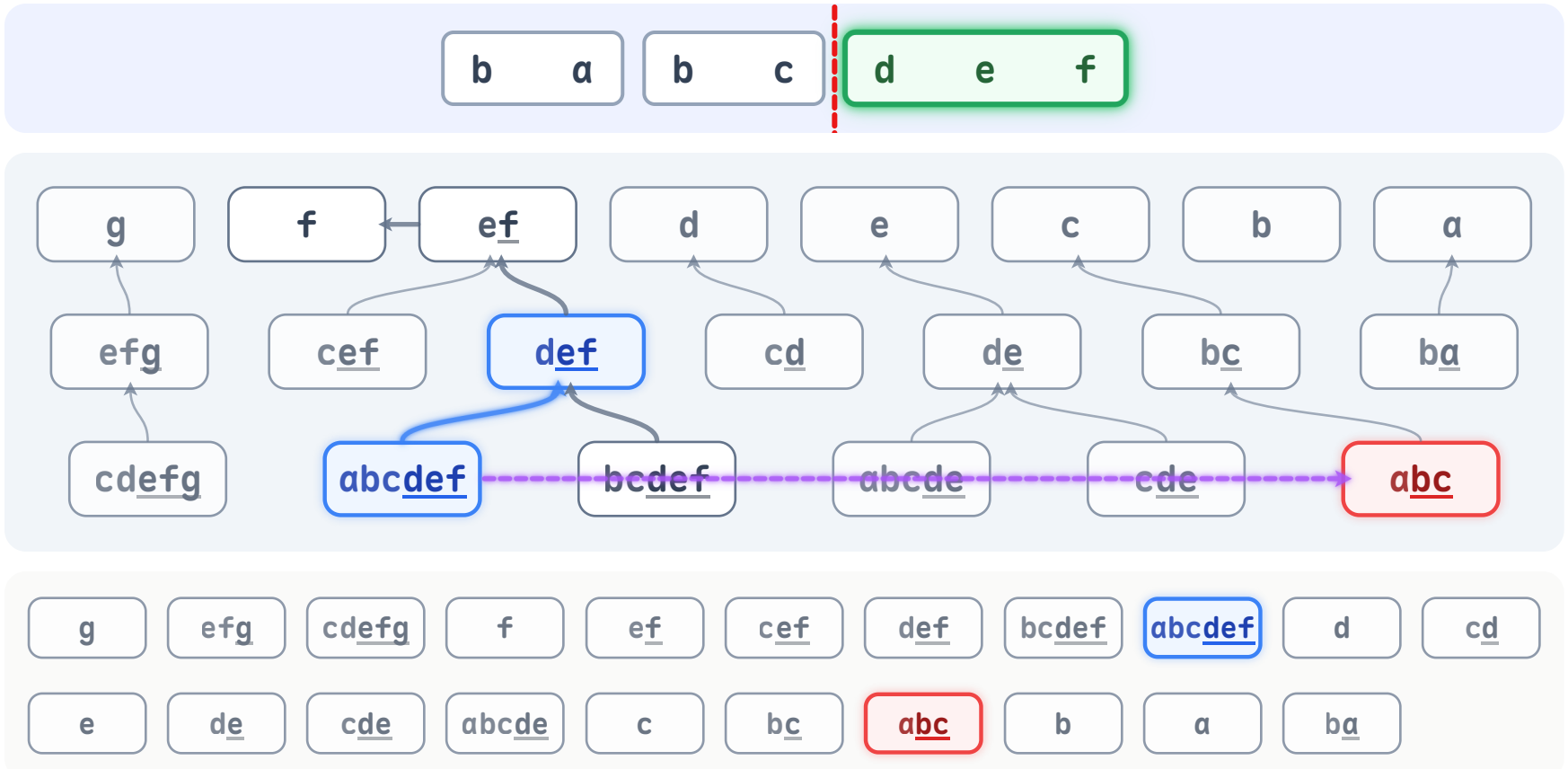
Centroid Decomposition



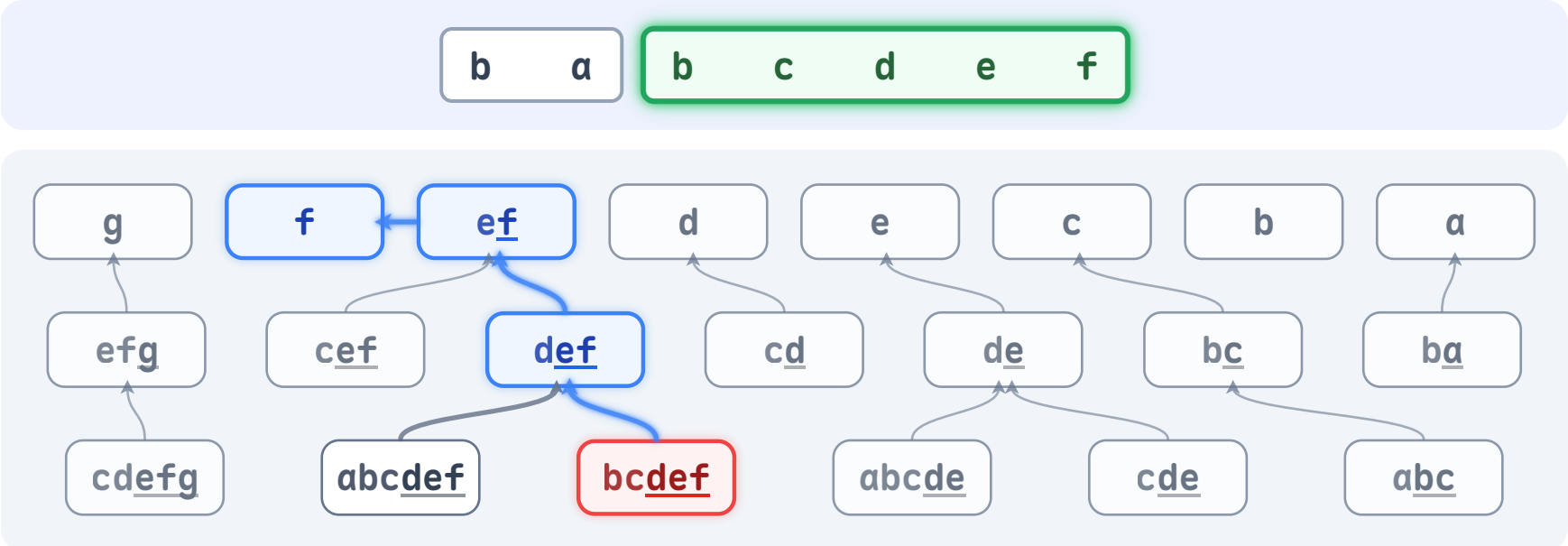
Centroid Decomposition



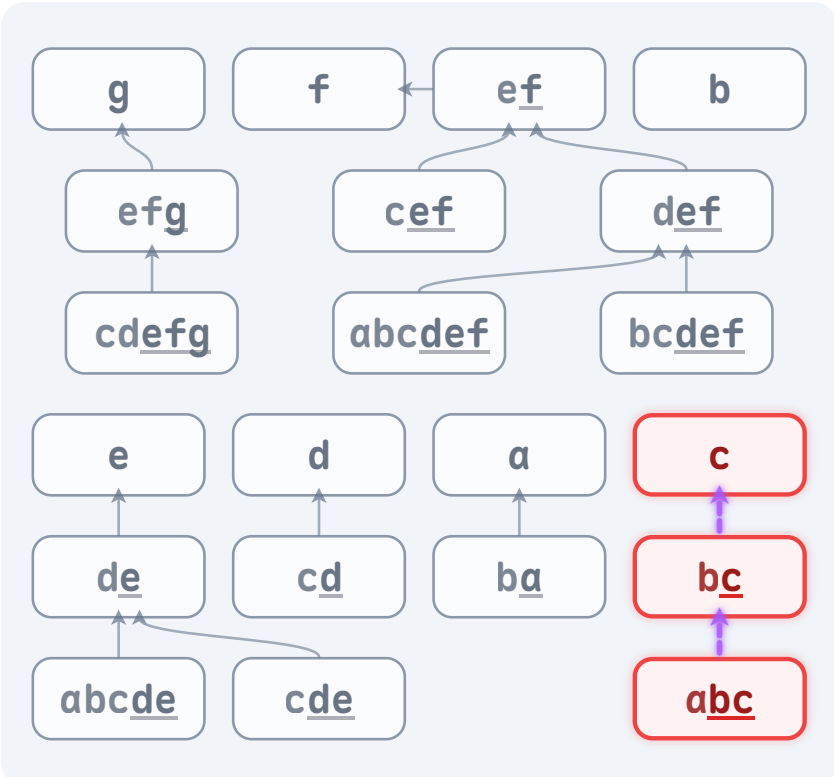
Centroid Decomposition



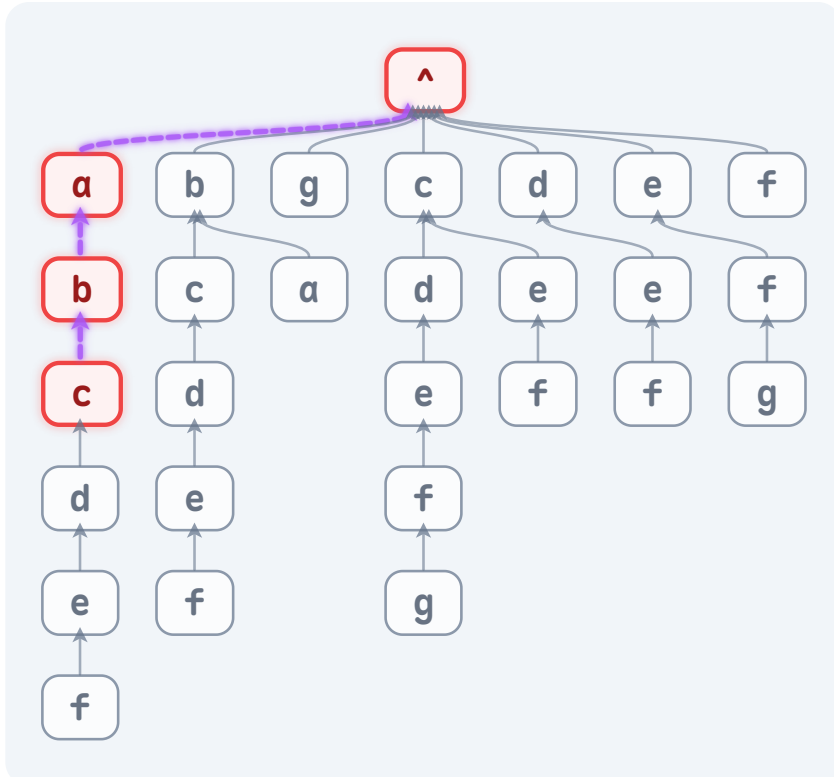
Centroid Decomposition



Aho–Corasick Automaton

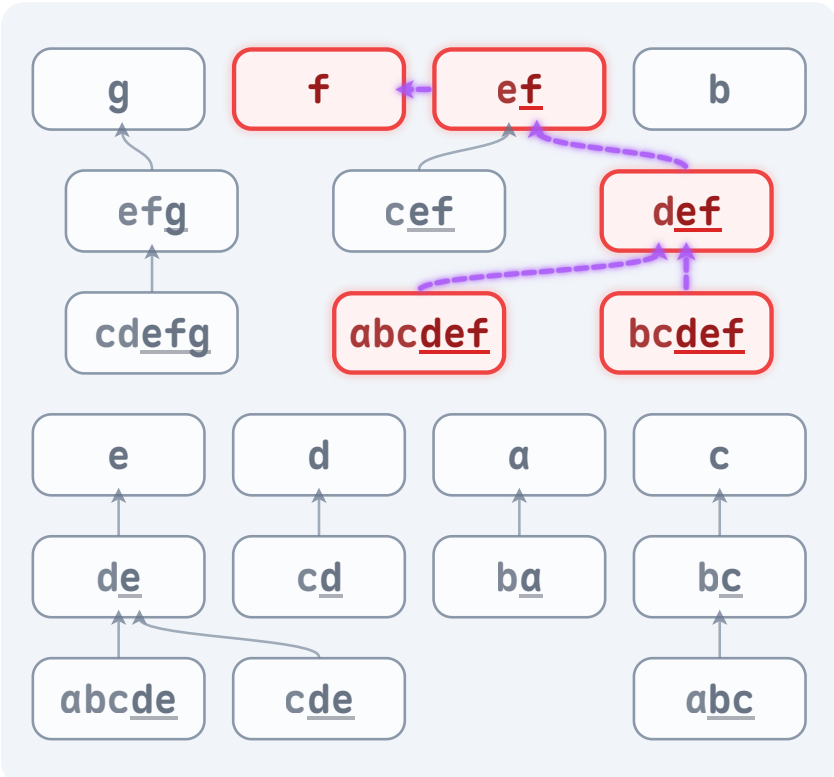


Successor Forest

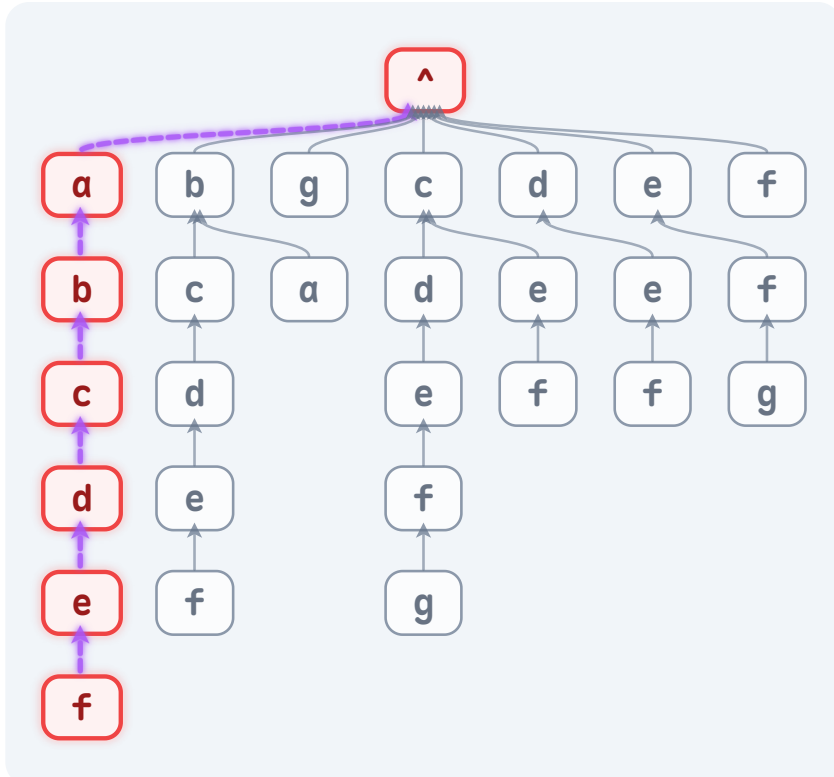


Aho–Corasick Automaton

Aho–Corasick Automaton

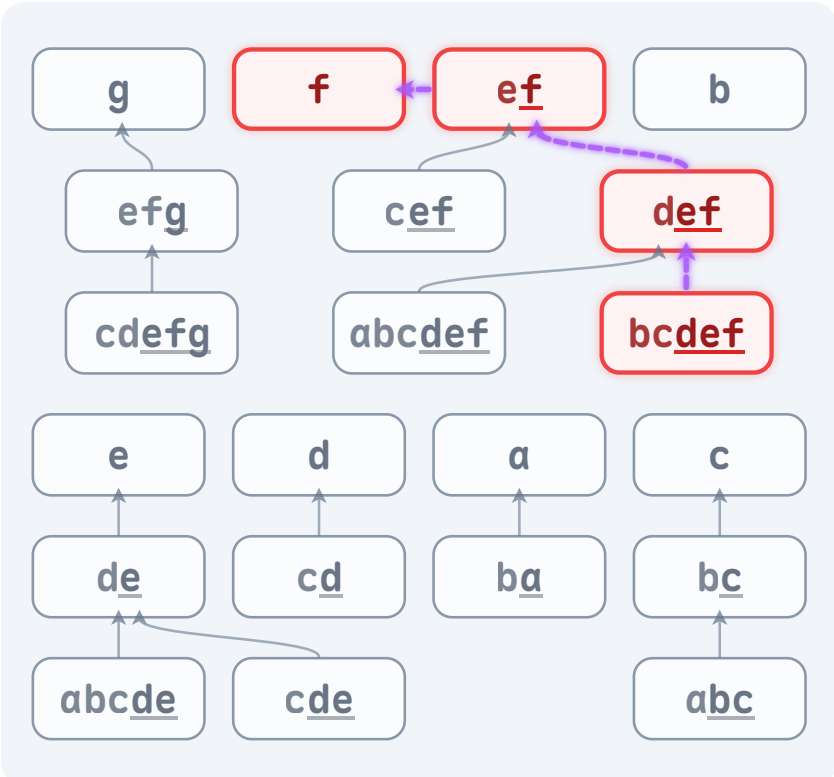


Successor Forest

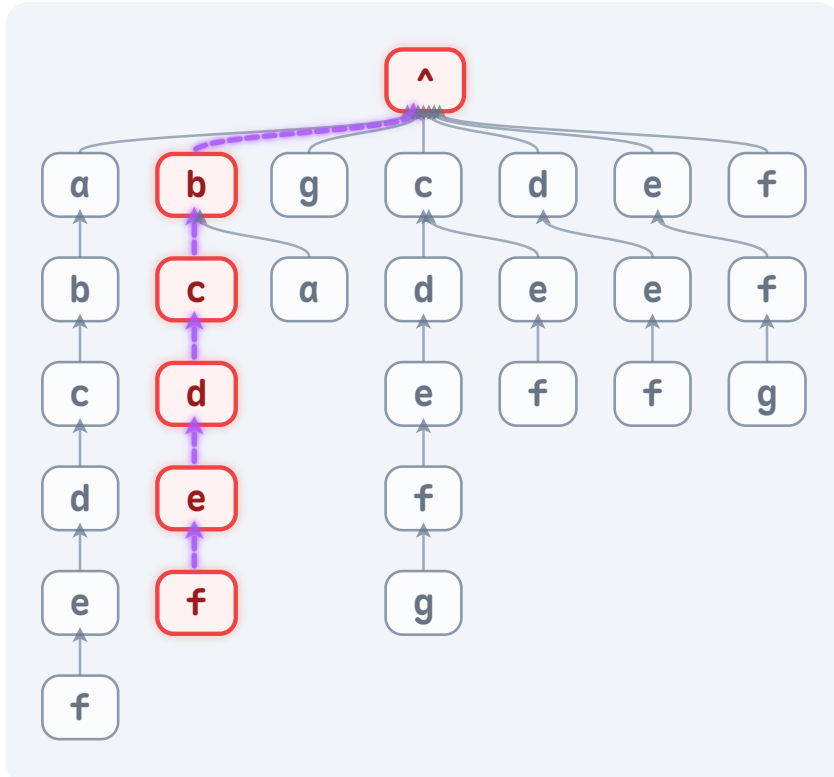


Aho–Corasick Automaton

Aho–Corasick Automaton

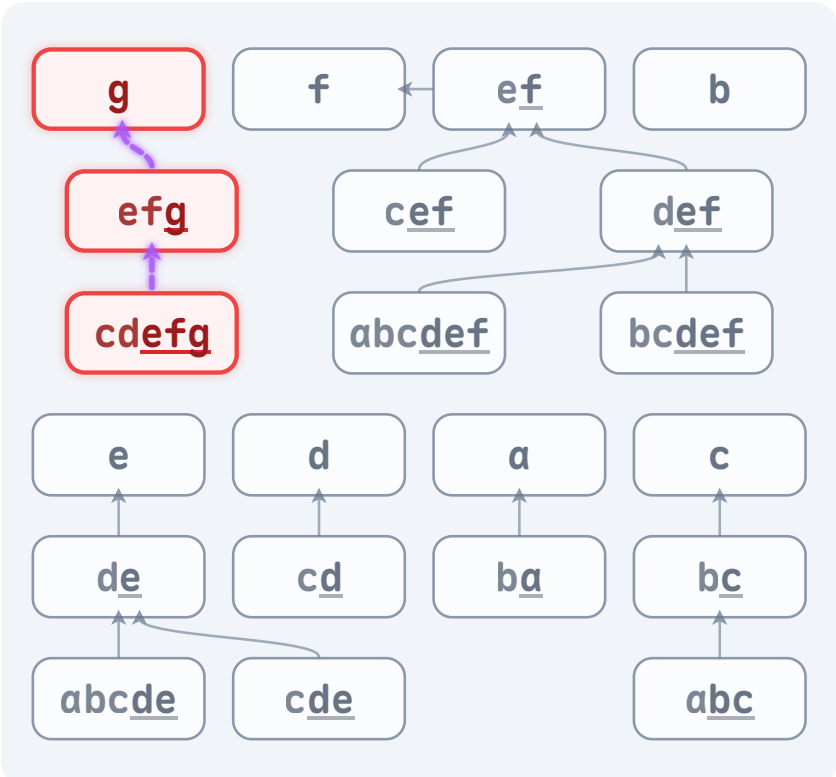


Successor Forest

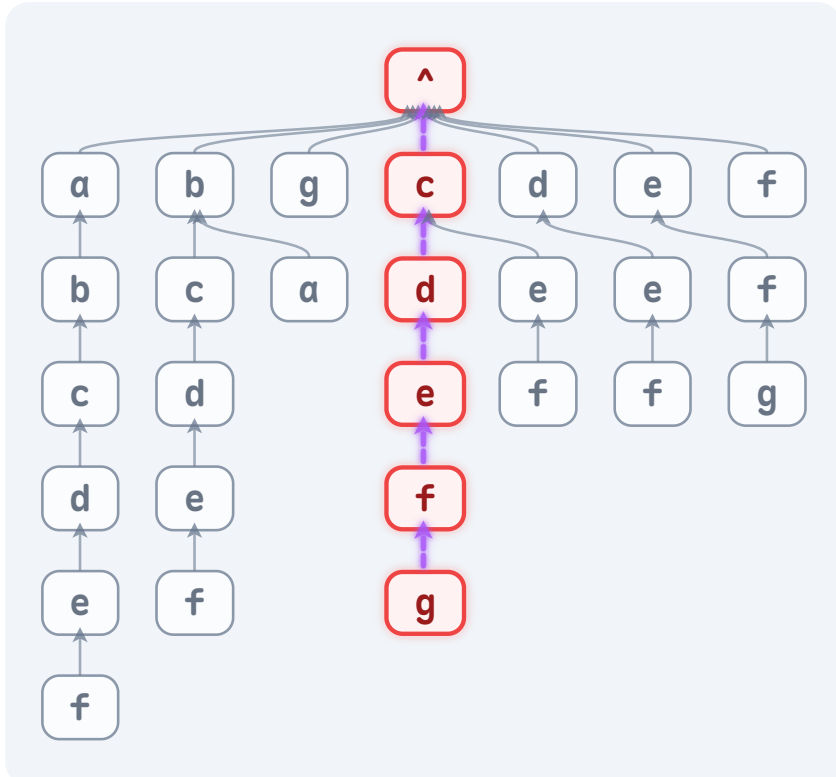


Aho–Corasick Automaton

Aho–Corasick Automaton

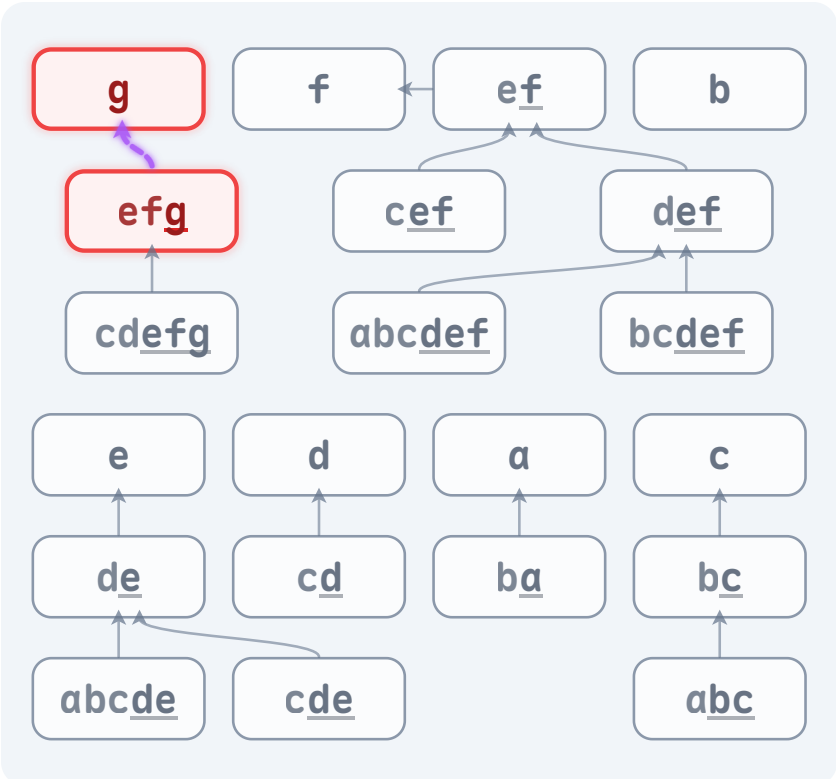


Successor Forest

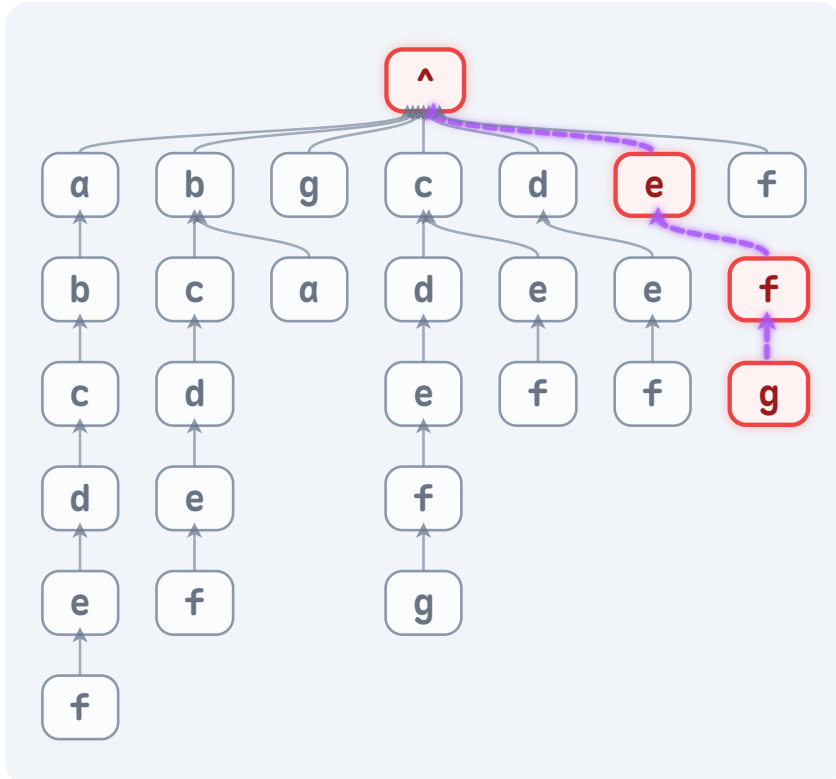


Aho–Corasick Automaton

Aho–Corasick Automaton



Successor Forest



Aho–Corasick Automaton

Incremental BPE Algorithm

- **For each byte:**
  - **Transition on the Aho–Corasick automaton:** $\mathcal{O}(1)$
  - **Centroid Decomposition:** $\mathcal{O}(\log t)$
    - **Verification for the centroid:** $\mathcal{O}(1)$
    - **Binary search for its children:** $\mathcal{O}(\log t)$
- $\mathcal{O}(\log^2 t)$ **in total!** (t **is the length of the longest token)**
- **Check out the paper for more details!**

Thank you!

Incremental BPE Tokenization