

Linear-Time Suffix-Sorting

Proseminar Datenkompression

bei Prof. Böttcher – WS 16/17 – Clemens Damke



Konstruktion eines **Suffix Arrays** mit
einem **rekursionsfreien Linearzeit-Algorithmus**.

Konstruktion eines **Suffix Arrays** mit
einem rekursionsfreien Linearzeit-Algorithmus.

Was ist ein Suffix Array?

P A R A L L E L

Was ist ein Suffix Array?

1	P	A	R	A	L	L	E	L
2		A	R	A	L	L	E	L
3			R	A	L	L	E	L
4				A	L	L	E	L
5					L	L	E	L
6						L	E	L
7							E	L
8								L

Was ist ein Suffix Array?

1	P	A	R	A	L	L	E	L
2	A	R	A	L	L	E	L	
3	R	A	L	L	E	L		
4	A	L	L	E	L			
5	L	L	E	L				
6	L	E	L					
7	E	L						
8	L							

Was ist ein Suffix Array?

1	A L L E L							
2	A R A L L E L							
3	E L							
4	L							
5	L E L							
6	L L E L							
7	P A R A L L E L							
8	R A L L E L							

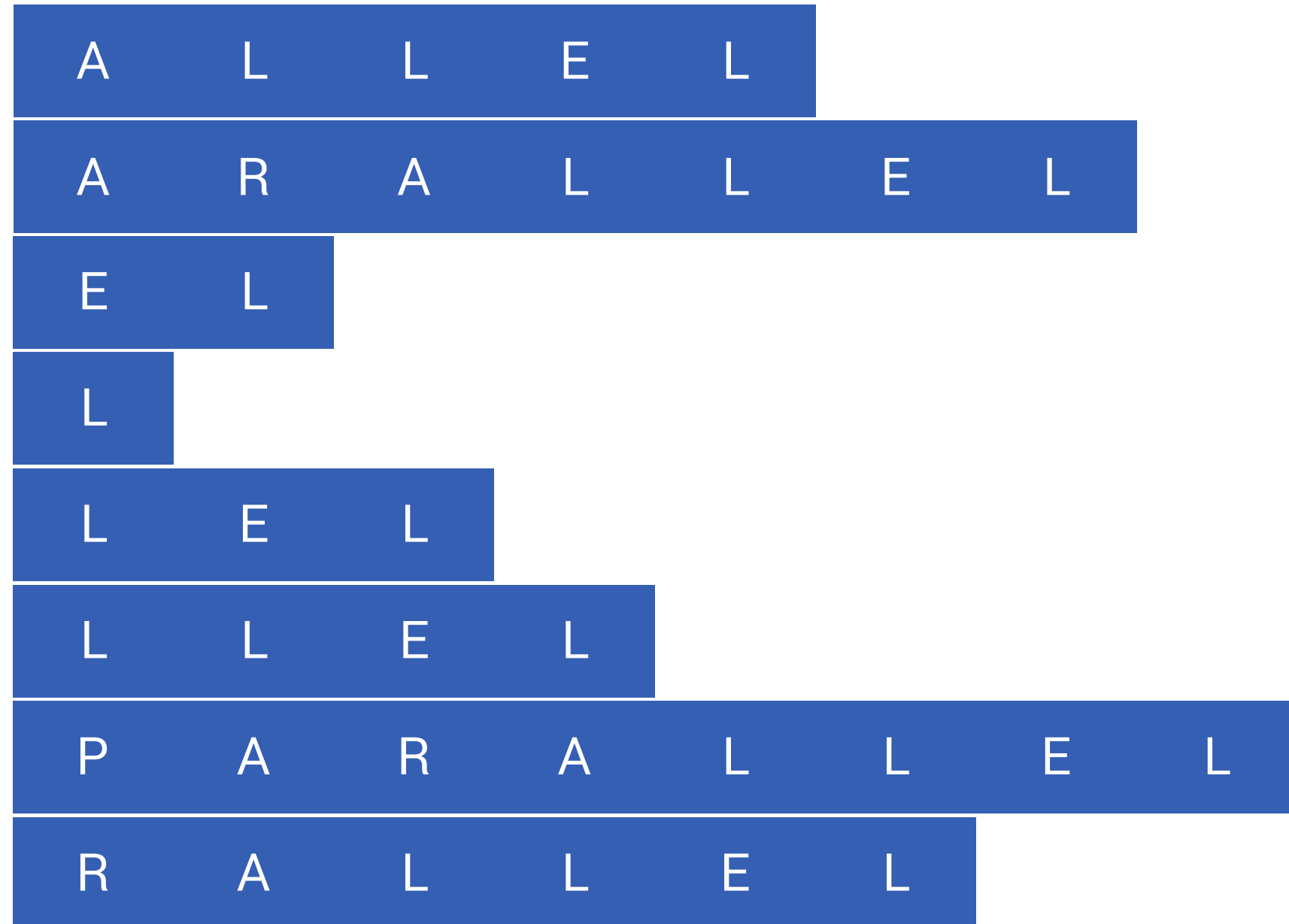
Was ist ein Suffix Array?

1	A L L E L	4
2	A R A L L E L	2
3	E L	7
4	L	8
5	L E L	6
6	L L E L	5
7	P A R A L L E L	1
8	R A L L E L	3

Einsatzgebiete

Substringsuche

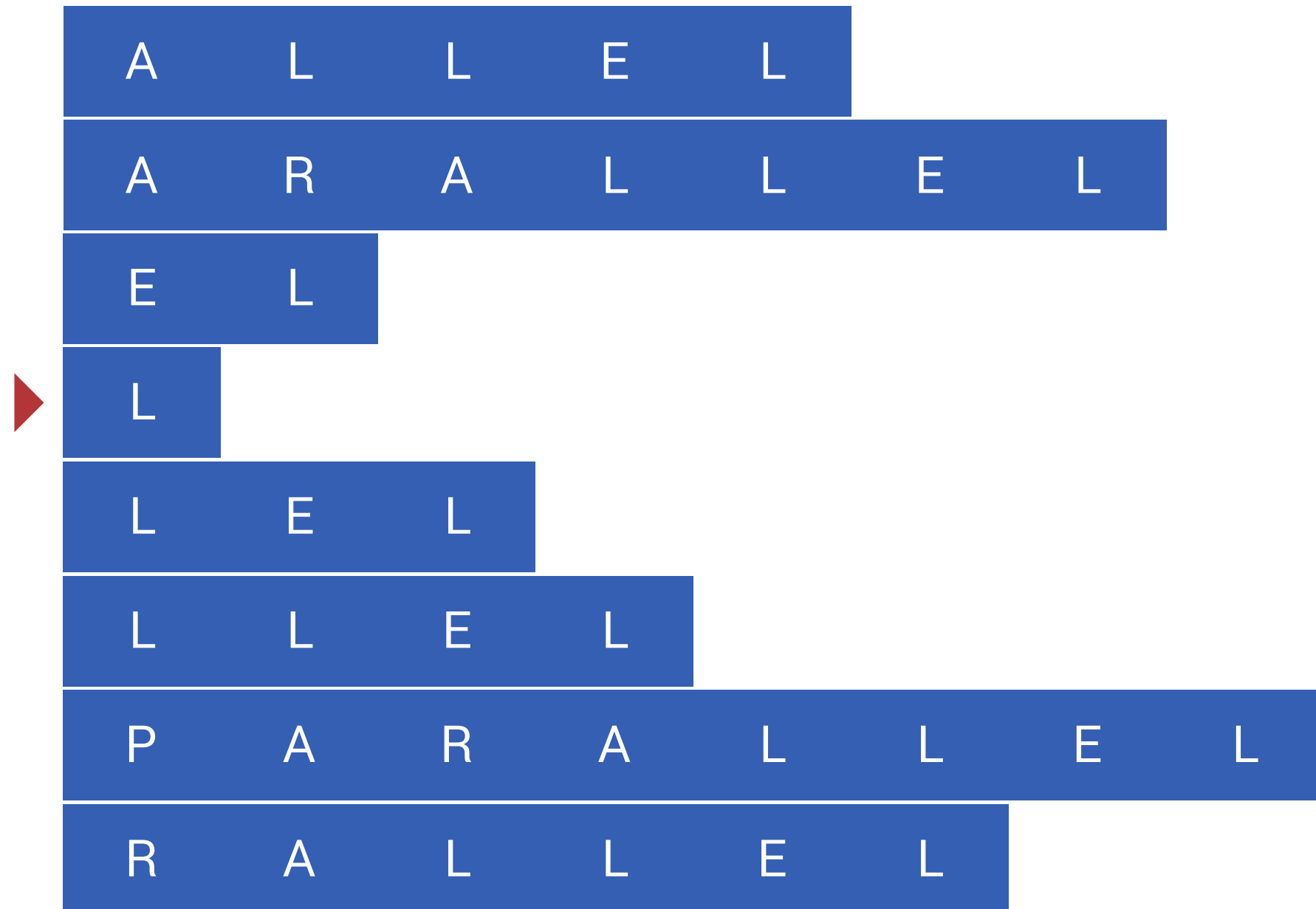
Ist *alle* in *parallel* enthalten?



Einsatzgebiete

Substringsuche

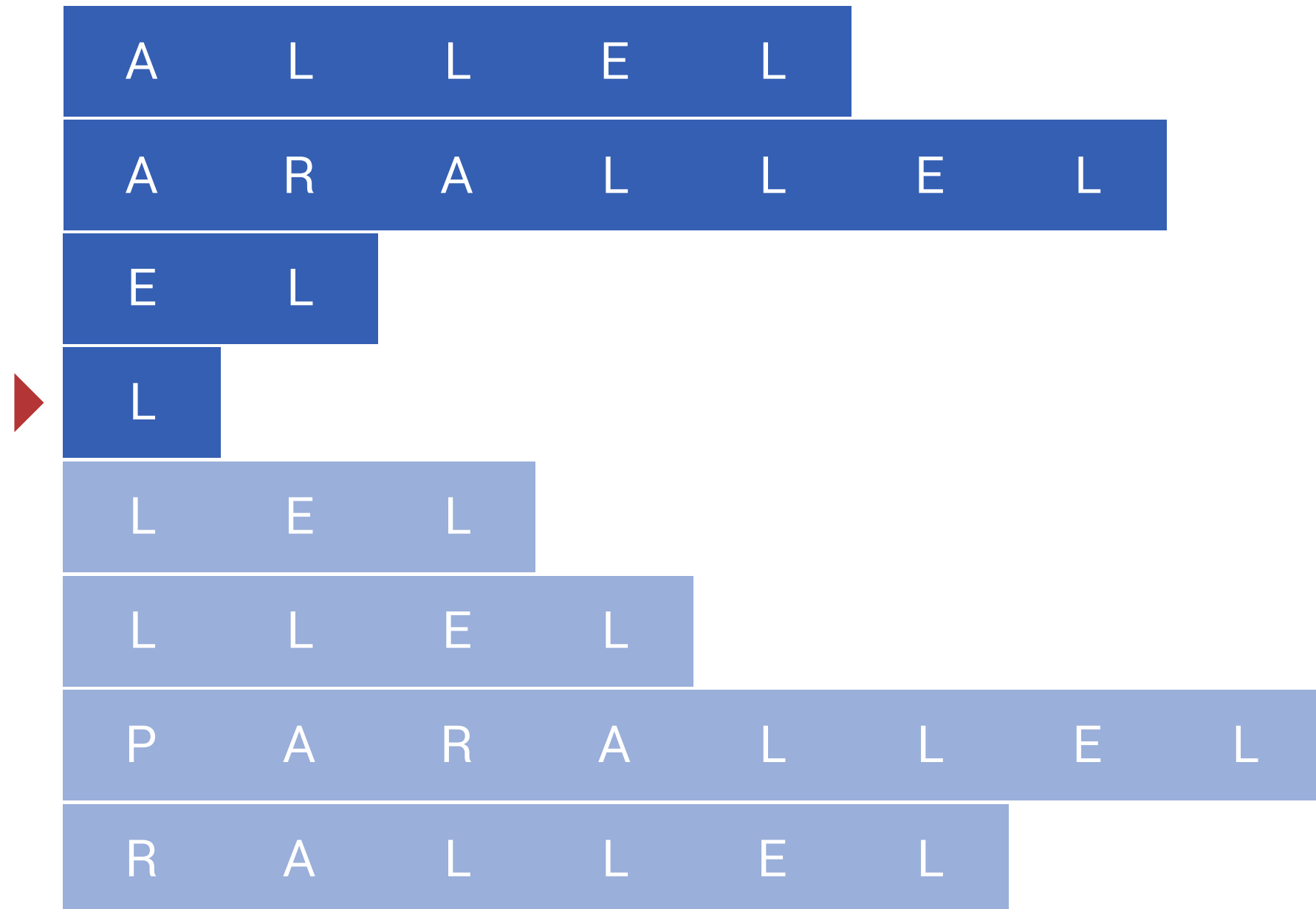
Ist *alle* in *parallel* enthalten?



Einsatzgebiete

Substringsuche

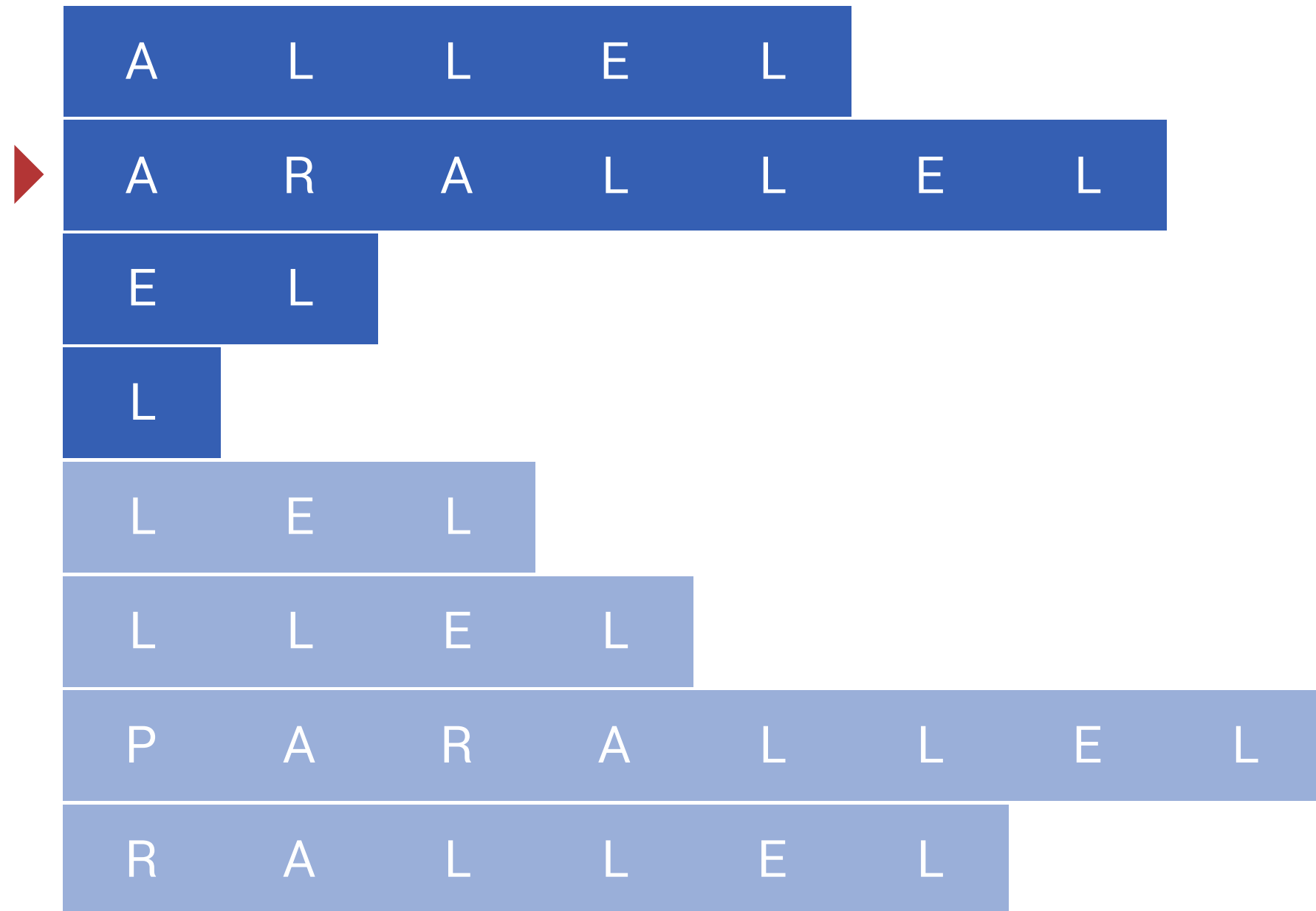
Ist *alle* in *parallel* enthalten?



Einsatzgebiete

Substringsuche

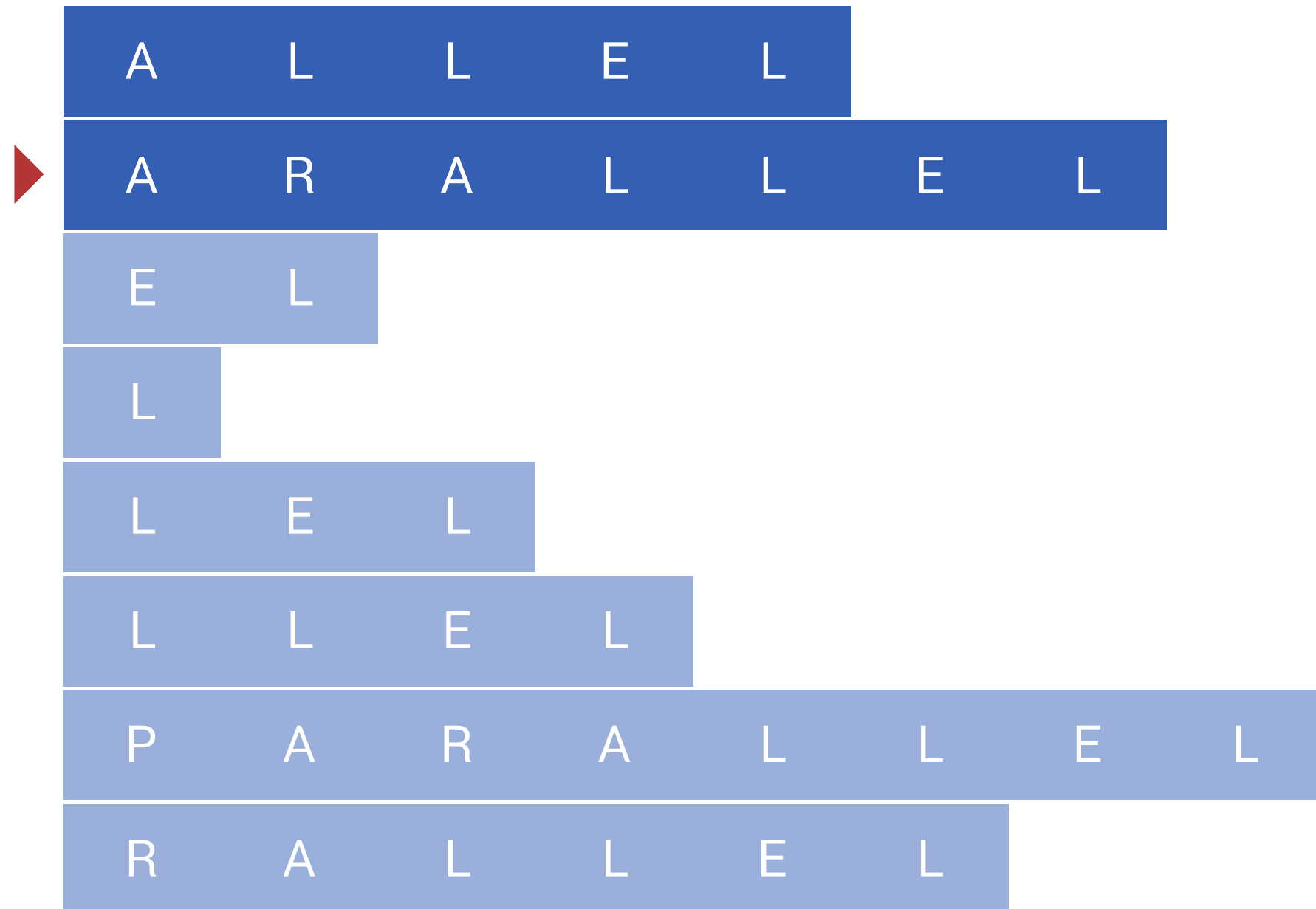
Ist *alle* in *parallel*
enthalten?



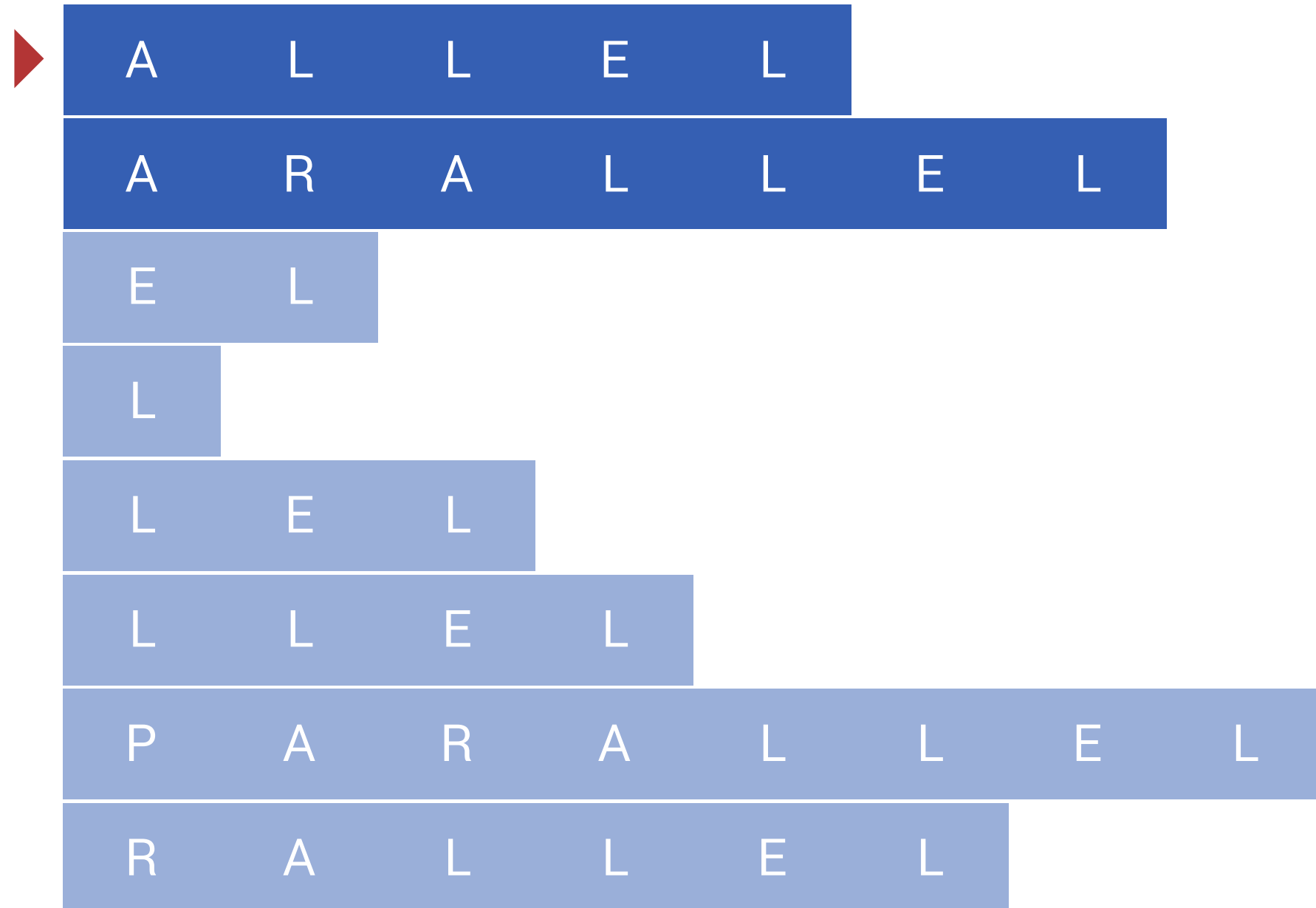
Einsatzgebiete

Substringsuche

Ist *alle* in *parallel* enthalten?



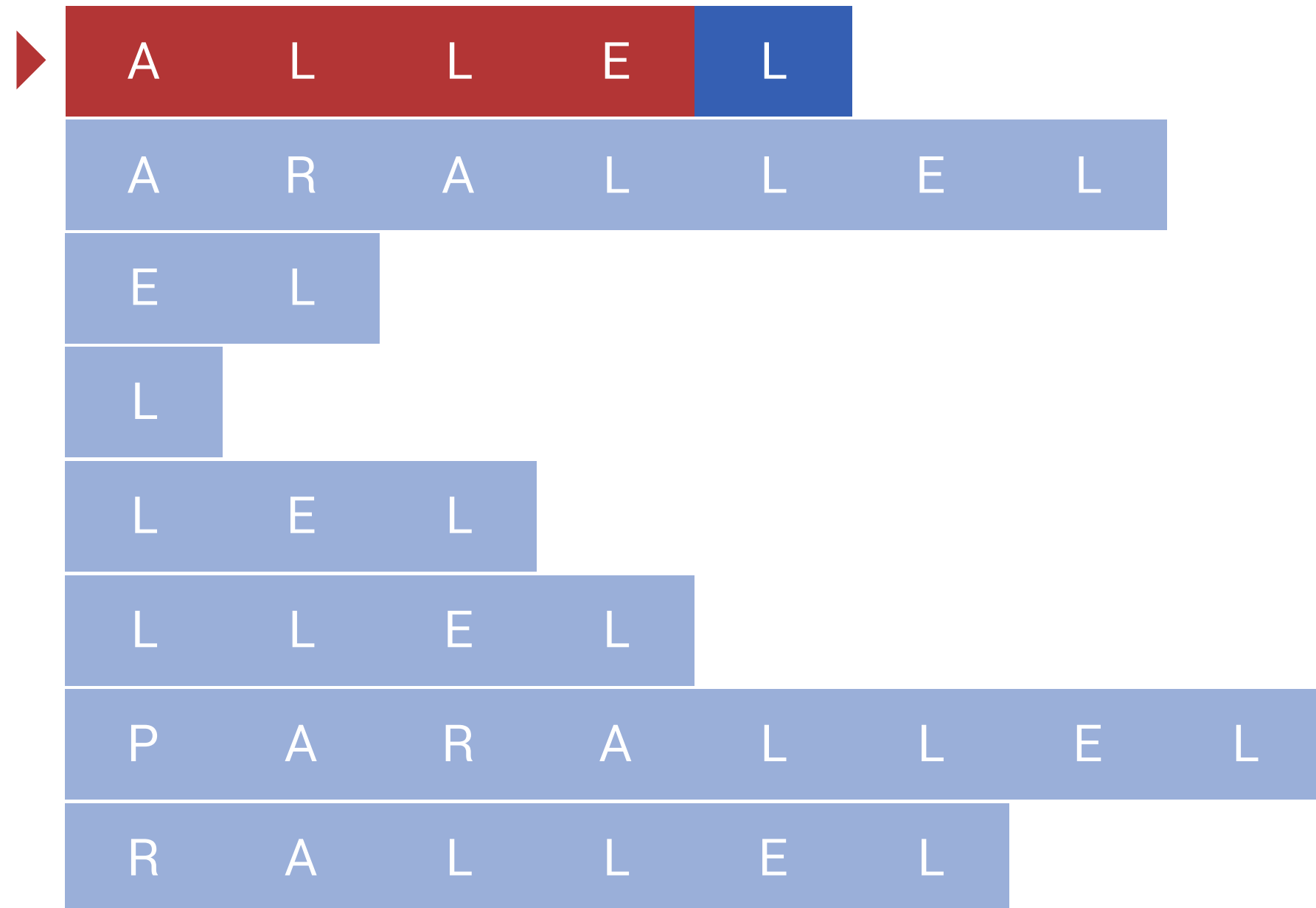
Einsatzgebiete



Substringsuche

Ist *alle* in *parallel*
enthalten?

Einsatzgebiete



Substringsuche

Ist *alle* in *parallel*
enthalten?

Ja, an Stelle 4.

Einsatzgebiete

Verwendet in Implementationen
des **LZ77**-Kompressionsalgorithmus

Konstruktion eines **Suffix Arrays** mit
einem **rekursionsfreien Linearzeit-Algorithmus**.

Konstruktion eines Suffix Arrays mit
einem rekursionsfreien Linearzeit-Algorithmus

Übersicht

Problemstellung

Lösungsansätze

GSACA

Performance

Rückblick

Übersicht

Problemstellung ✓

Lösungsansätze

GSACA

Performance

Rückblick

Lösungsansätze

Naiver Ansatz

Verwendung eines allgemeinen Sortierverfahrens
(z. B. Quicksort)

$$O(n \log n) \cdot O(n) = O(n^2 \log n)$$

Naiver Ansatz

Verwendung eines allgemeinen Sortierverfahrens
(z. B. Quicksort)

$$O(n \log n) \cdot O(n) = O(n^2 \log n) \neq O(n)$$

Linearzeit Ansätze

Skew

SA-IS

Art

rekursiv

rekursiv

Zeit

$O(n)$

$O(n)$

Speicher

$O(\log n) + \max 24n$

$O(\log n) + \max 2n$

Linearzeit Ansätze

	Skew	SA-IS	?
Art	rekursiv	rekursiv	iterativ
Zeit	$O(n)$	$O(n)$	$O(n)$
Speicher	$O(\log n) + \max 24n$	$O(\log n) + \max 2n$	$O(1) + ?$

GSACA

iterativ

$O(n)$

$O(1) + ?$

GSACA

Greedy Suffix Array Construction Algorithm

Definitionen

P	A	R	A	L	L	E	L	\$
1	2	3	4	5	6	7	8	9

Definitionen

$S =$

P	A	R	A	L	L	E	L	\$
1	2	3	4	5	6	7	8	$n = 9$

$S :=$ Eingabe, eine mit \$ terminierte Zeichenkette der Länge n

Definitionen

$S =$

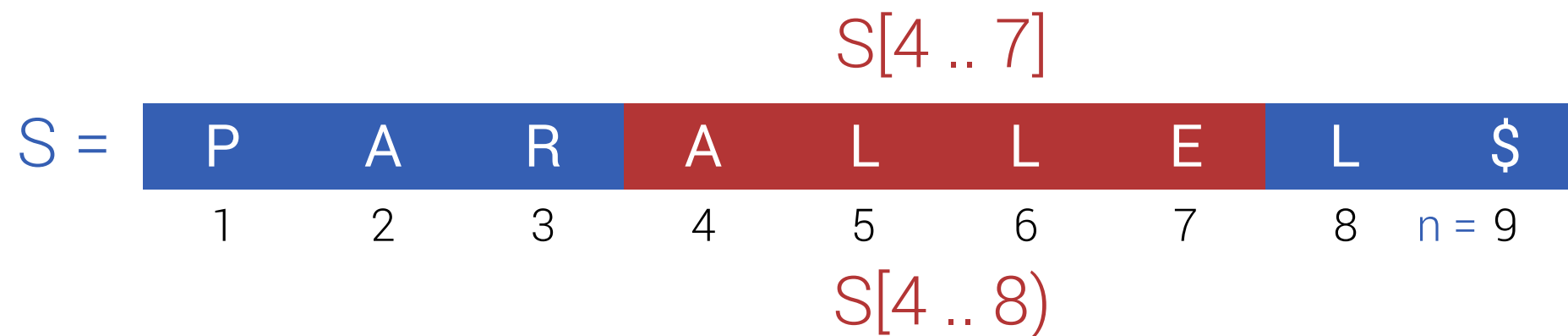
P	A	R	A	L	L	E	L	\$
1	2	3	4	5	6	7	8	$n = 9$

$S[4]$

$S :=$ Eingabe, eine mit \$ terminierte Zeichenkette der Länge n

$S[i] :=$ i-tes Zeichen von S

Definitionen



$S :=$ Eingabe, eine mit \$ terminierte Zeichenkette der Länge n

$S[i] :=$ i-tes Zeichen von S

$S[i \dots j + 1) := S[i \dots j] := S[i] \dots S[j]$

Definitionen



$S :=$ Eingabe, eine mit \$ terminierte Zeichenkette der Länge n

$S[i] :=$ i-tes Zeichen von S

$S[i .. j + 1) := S[i .. j] := S[i] \dots S[j]$

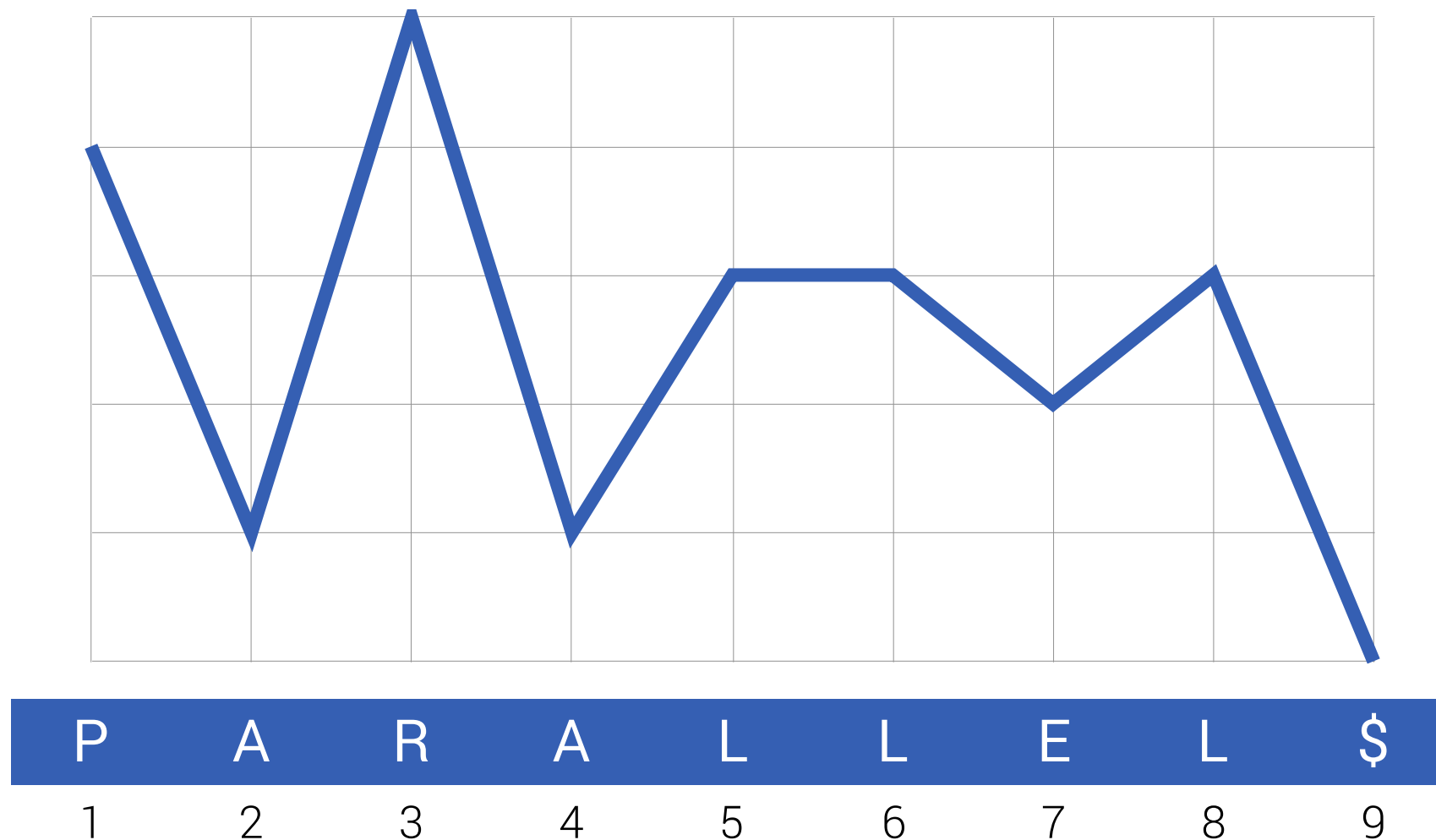
$S_i := S[i .. n]$

Definitionen

$$\hat{i} := \min \{ j \in [i .. n] : S_j <_{\text{lex}} S_i \}$$

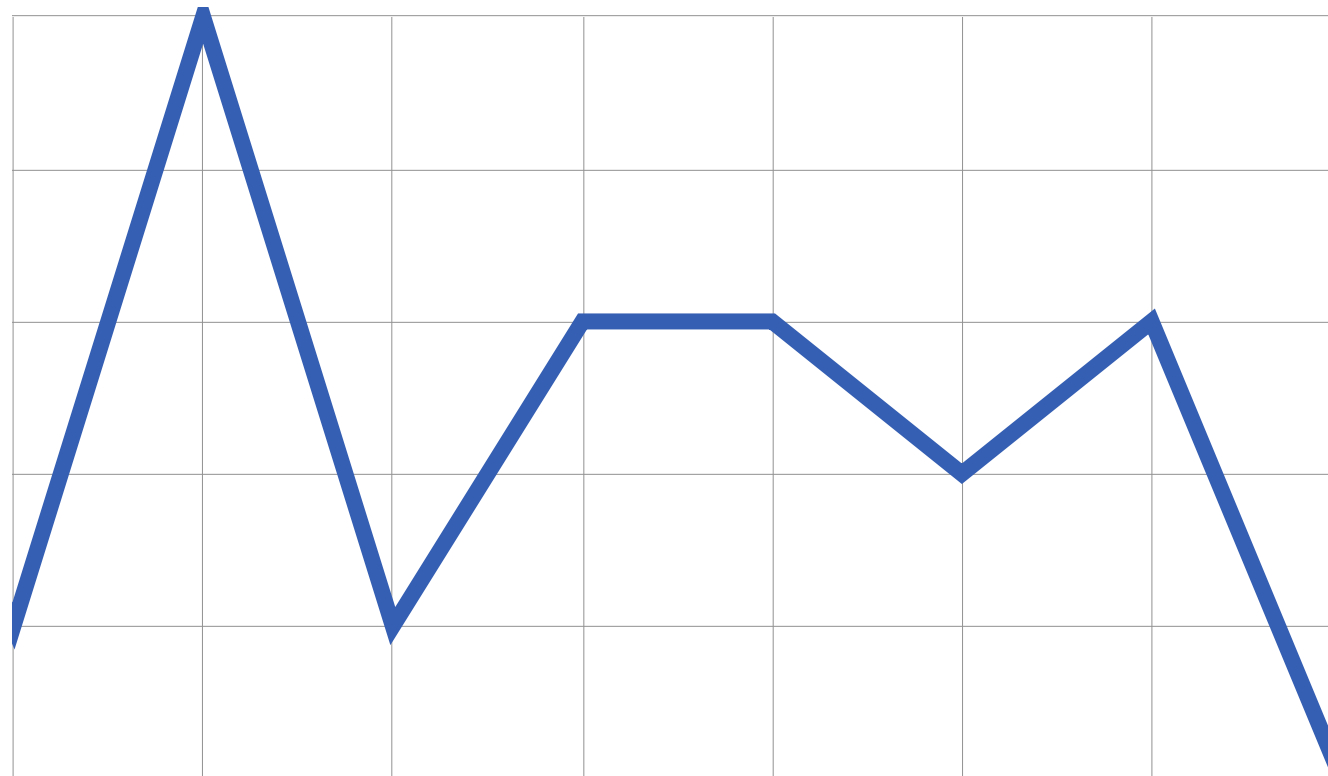
Definitionen

$$\hat{i} := \min \{ j \in [i .. n] : S_j <_{\text{lex}} S_i \}$$



Definitionen

$$\hat{i} := \min \{ j \in [i .. n] : S_j <_{\text{lex}} S_i \}$$

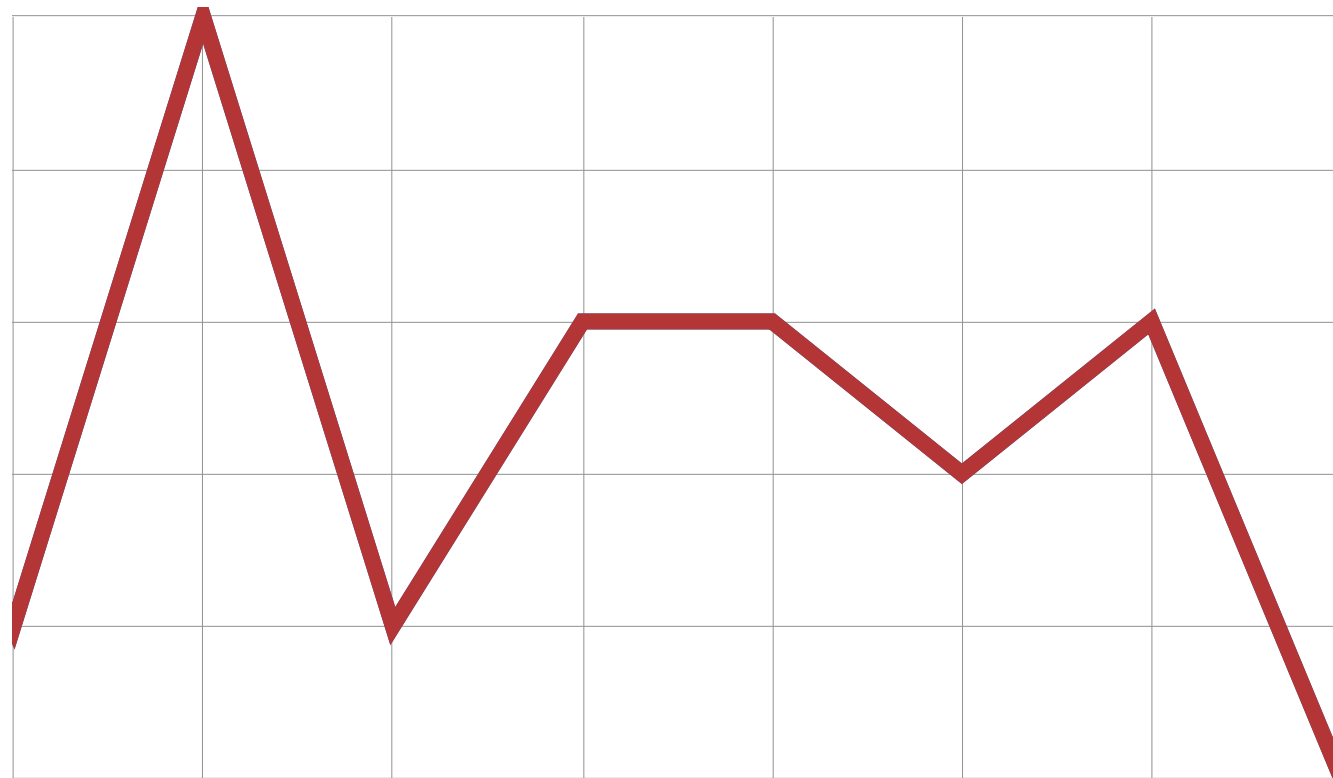


$S_i =$

A	R	A	L	L	E	L	\$
i = 2	3	4	5	6	7	8	9

Definitionen

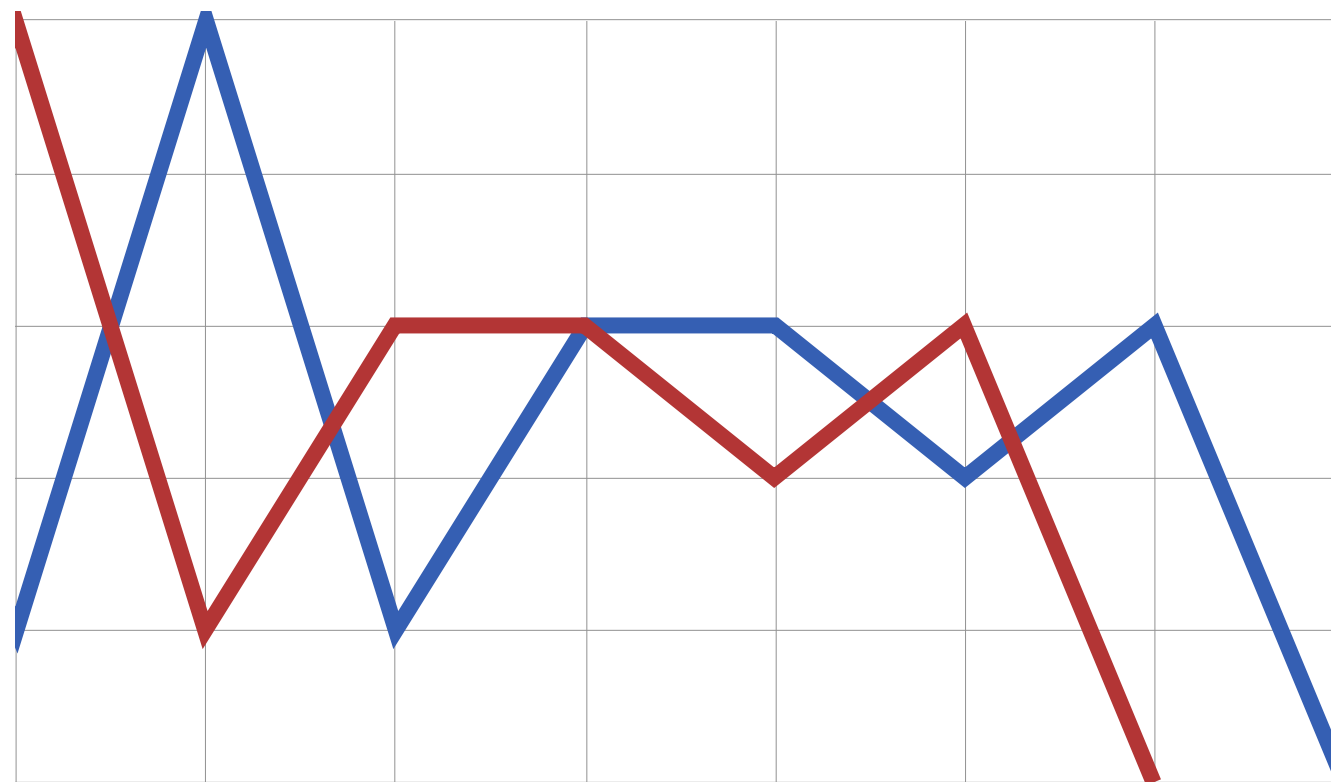
$$\hat{i} := \min \{ j \in [i .. n]: S_j <_{\text{lex}} S_i \}$$



$S_i =$	A	R	A	L	L	E	L	\$
$i =$	2	3	4	5	6	7	8	9
$S_j =$	A	R	A	L	L	E	L	\$
$j =$	2	3	4	5	6	7	8	9

Definitionen

$$\hat{i} := \min \{ j \in [i .. n] : S_j <_{\text{lex}} S_i \}$$



$S_i =$

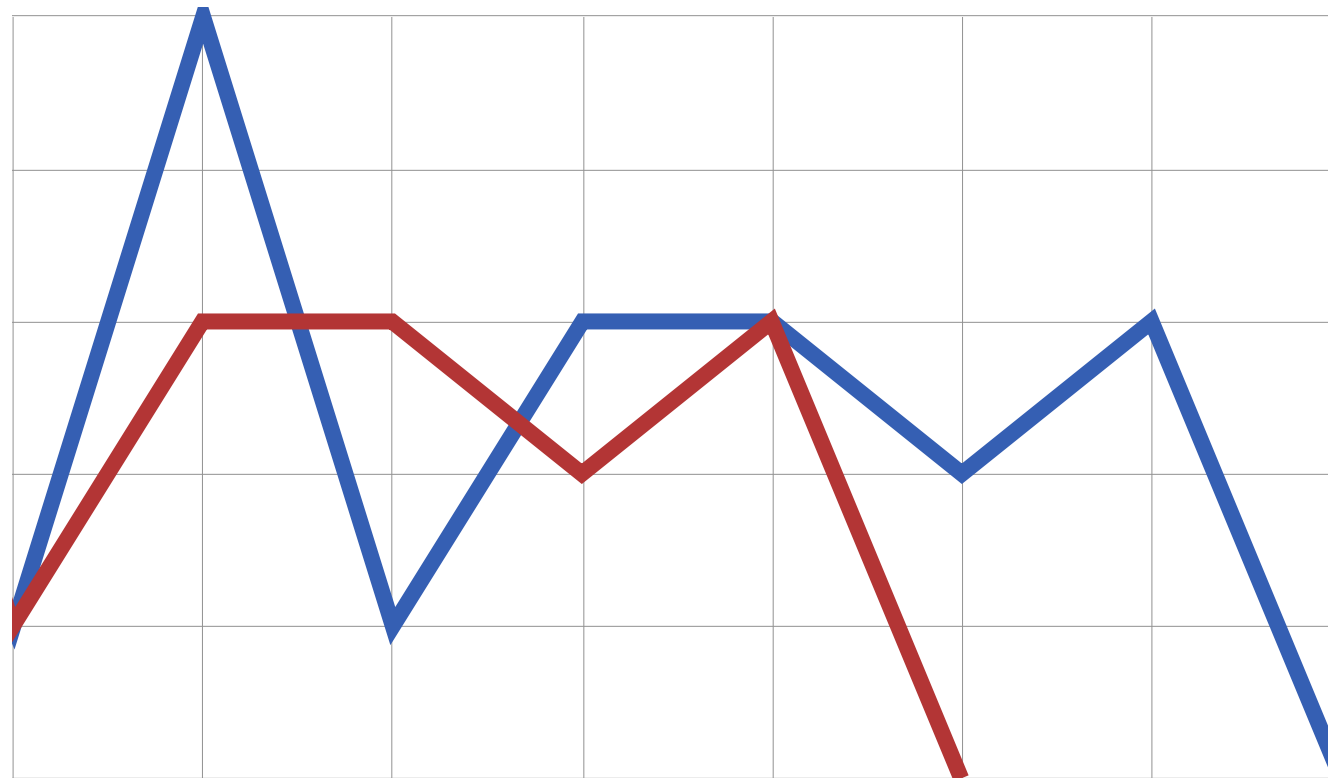
A	R	A	L	L	E	L	\$
$i = 2$	3	4	5	6	7	8	9

$S_j =$

R	A	L	L	E	L	\$
$j = 3$	4	5	6	7	8	9

Definitionen

$$\hat{i} := \min \{ j \in [i .. n]: S_j <_{\text{lex}} S_i \}$$



$S_i =$

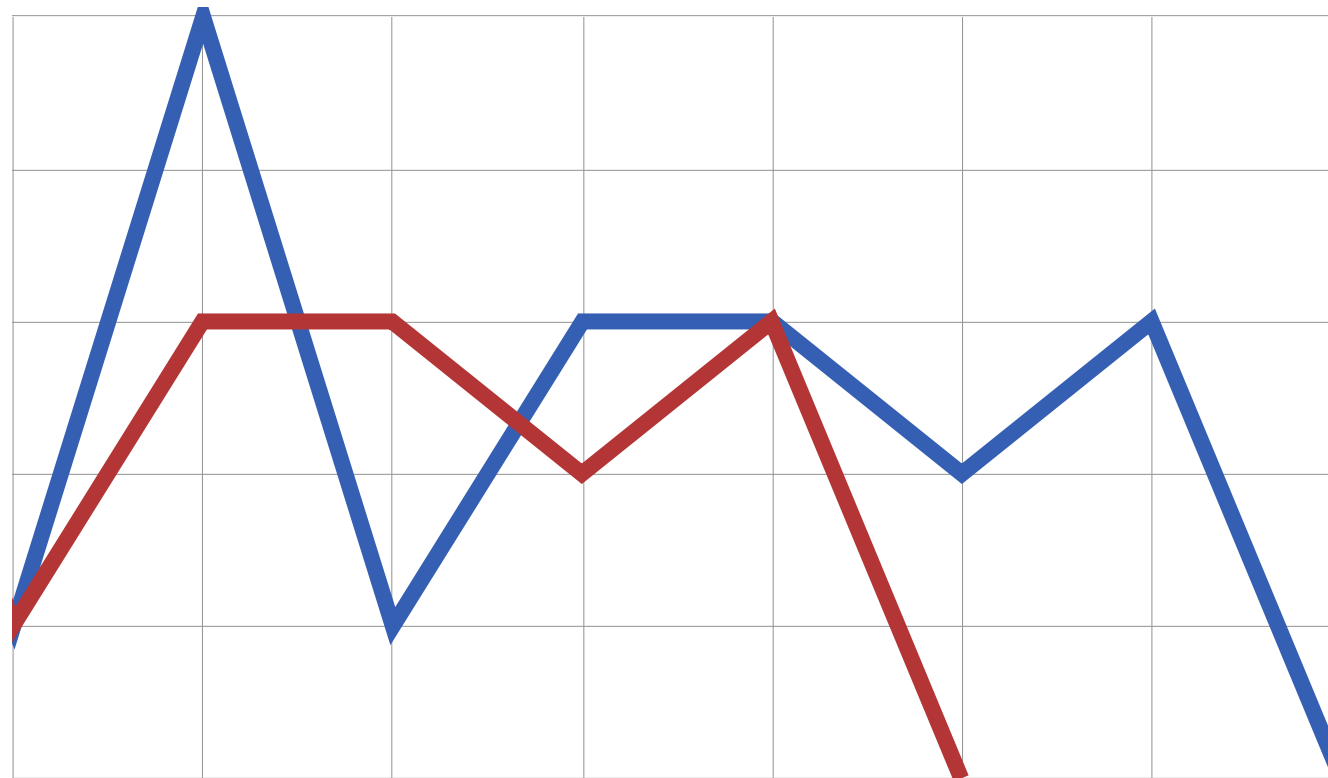
A	R	A	L	L	E	L	\$
$i = 2$	3	4	5	6	7	8	9

$S_j =$

A	L	L	E	L	\$
$j = 4$	5	6	7	8	9

Definitionen

$$\hat{i} := \min \{ j \in [i .. n] : S_j <_{\text{lex}} S_i \}$$



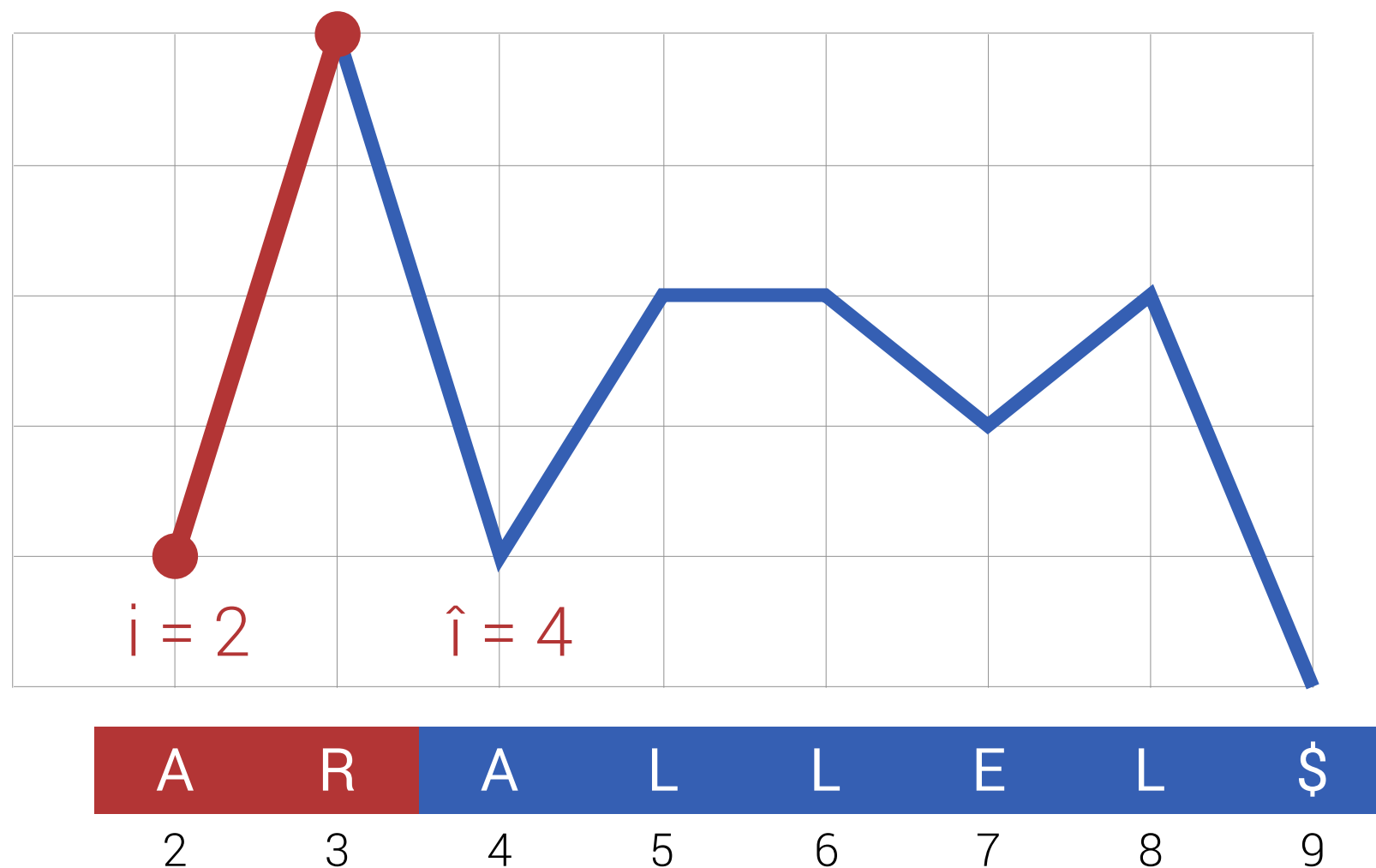
$S_i =$ A R A L L E L \$
 i = 2 3 4 5 6 7 8 9

$S_j =$ A L L E L \$
 j = 4 5 6 7 8 9

$\hat{i} = 4$

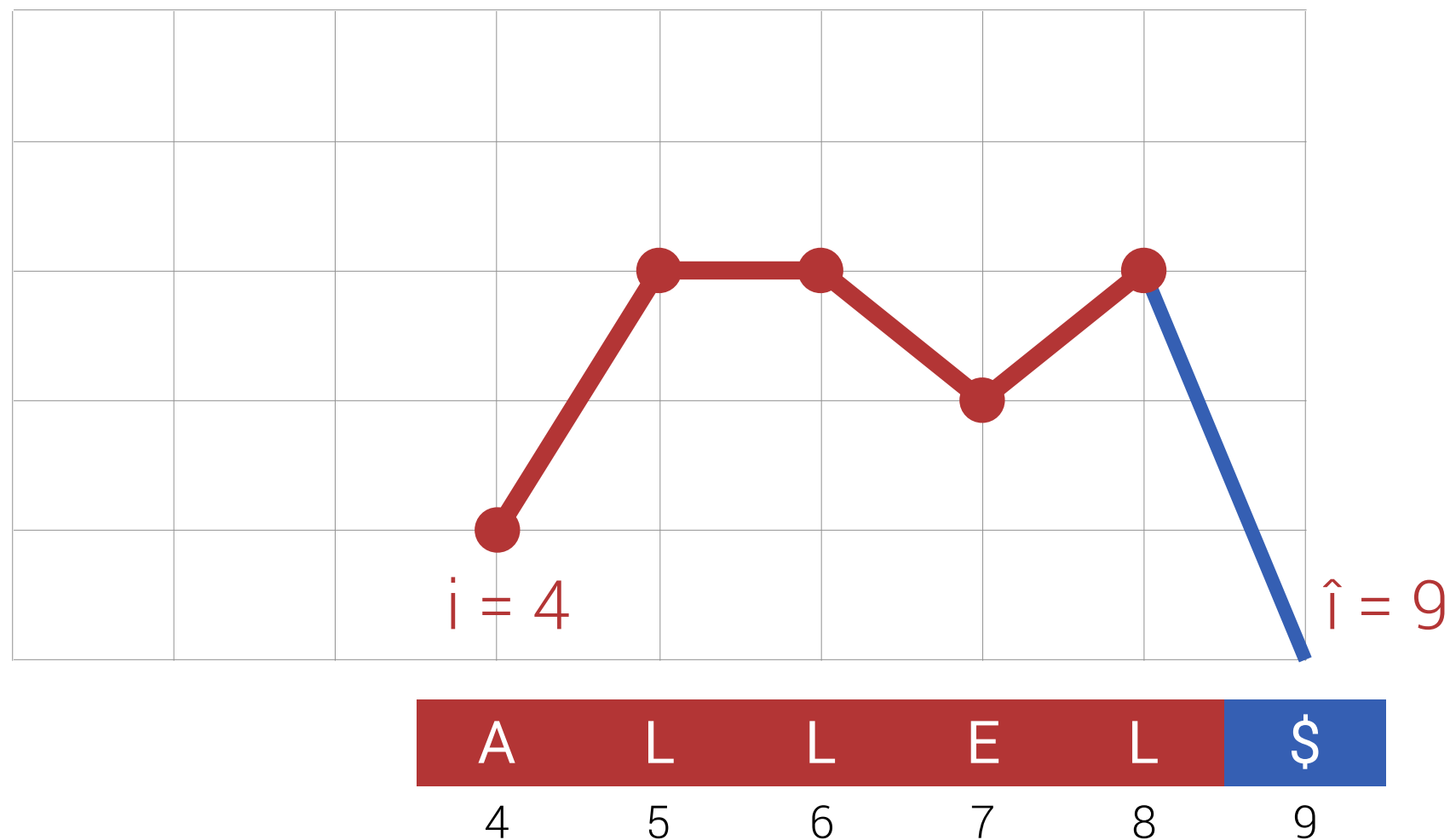
Definitionen

Gruppenkontext von $S_i := S[i .. \hat{i})$



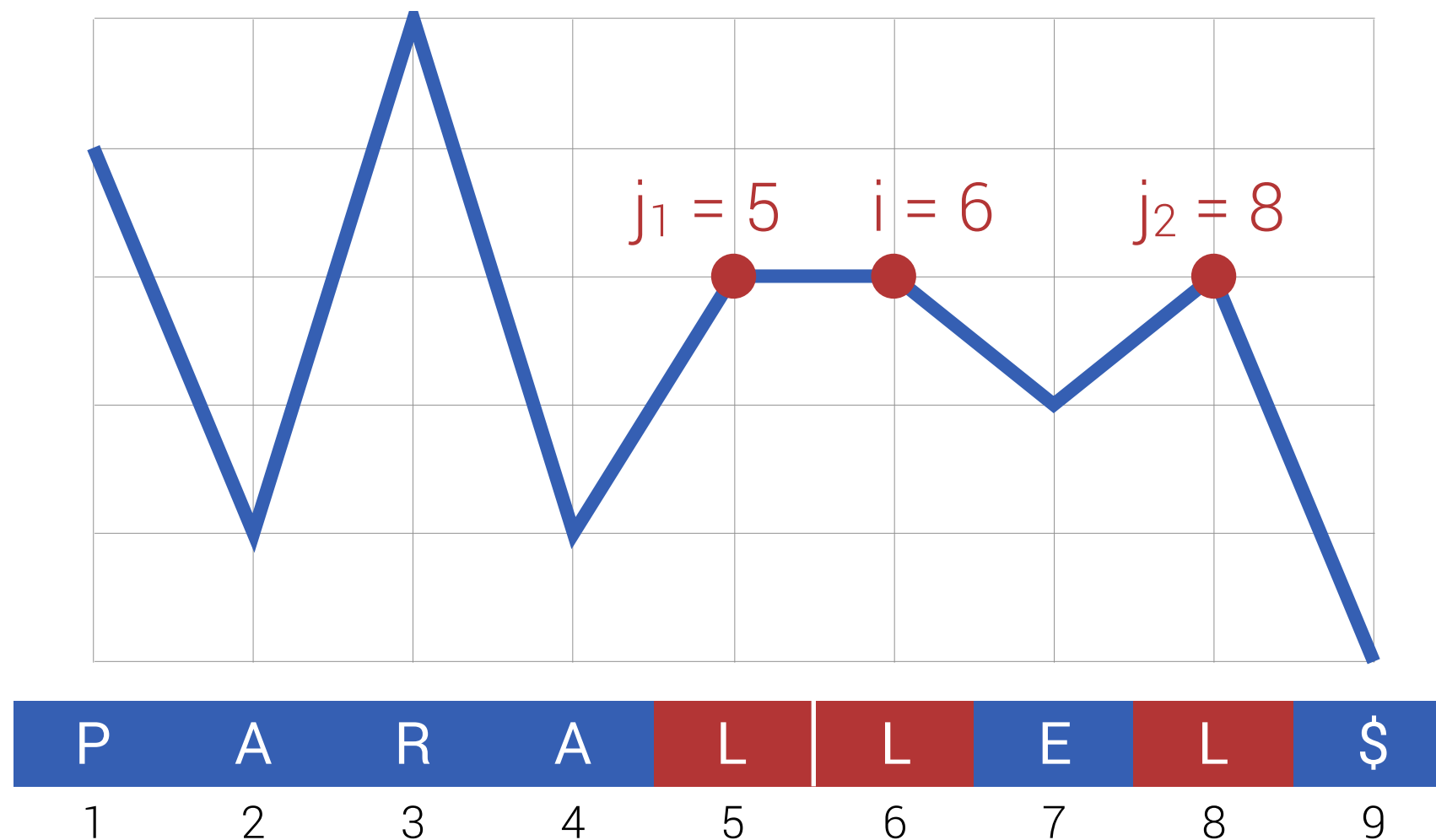
Definitionen

Gruppenkontext von $S_i := S[i .. \hat{i})$



Definitionen

Gruppe von $S_i := \{ S_j : \text{Gr.kontext } S_j = \text{Gr.kontext } S_i \}$



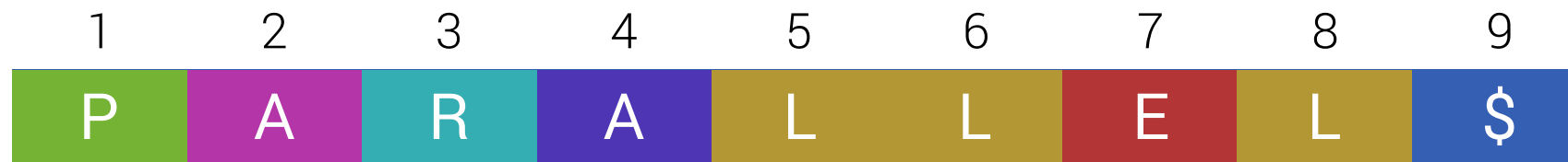
Grundprinzip

Eingabe

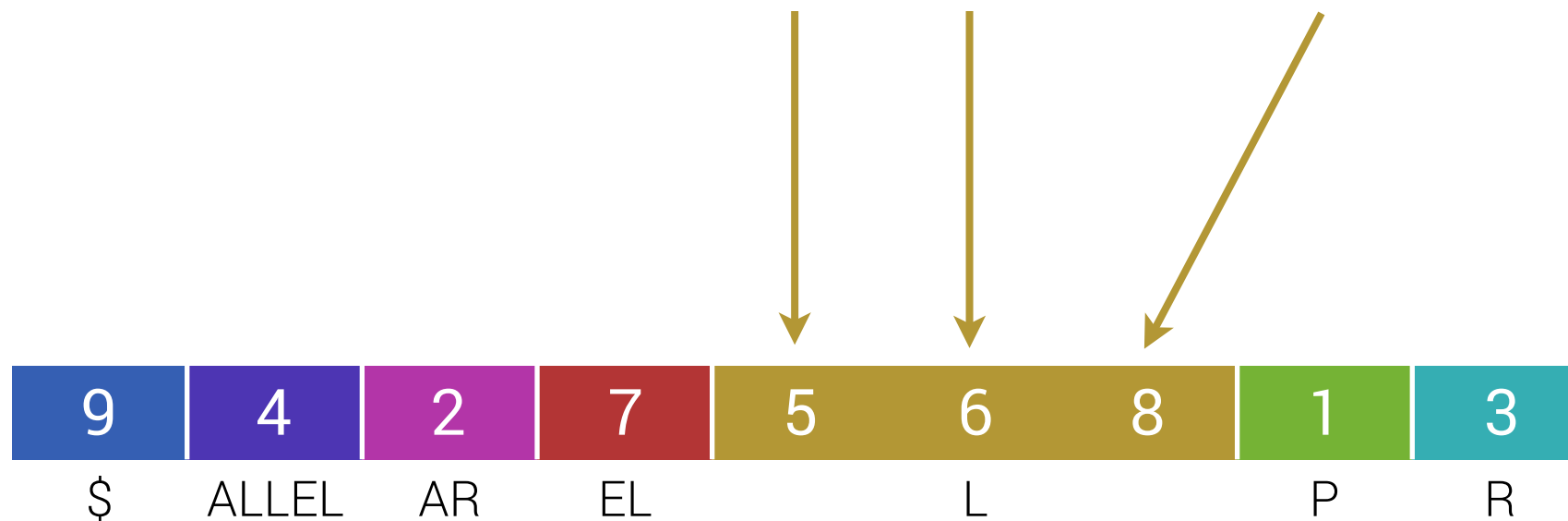
1	2	3	4	5	6	7	8	9
P	A	R	A	L	L	E	L	\$

Grundprinzip

Eingabe

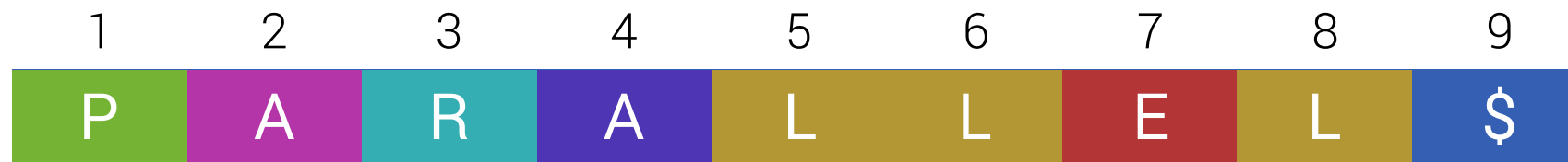


Phase 1

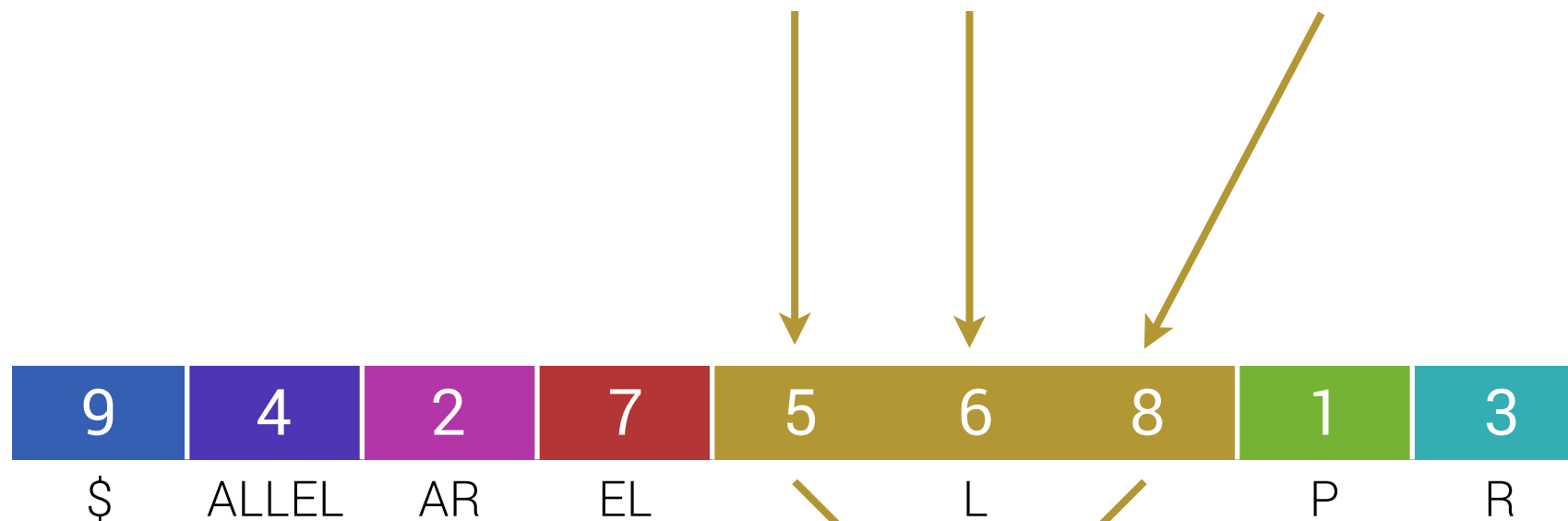


Grundprinzip

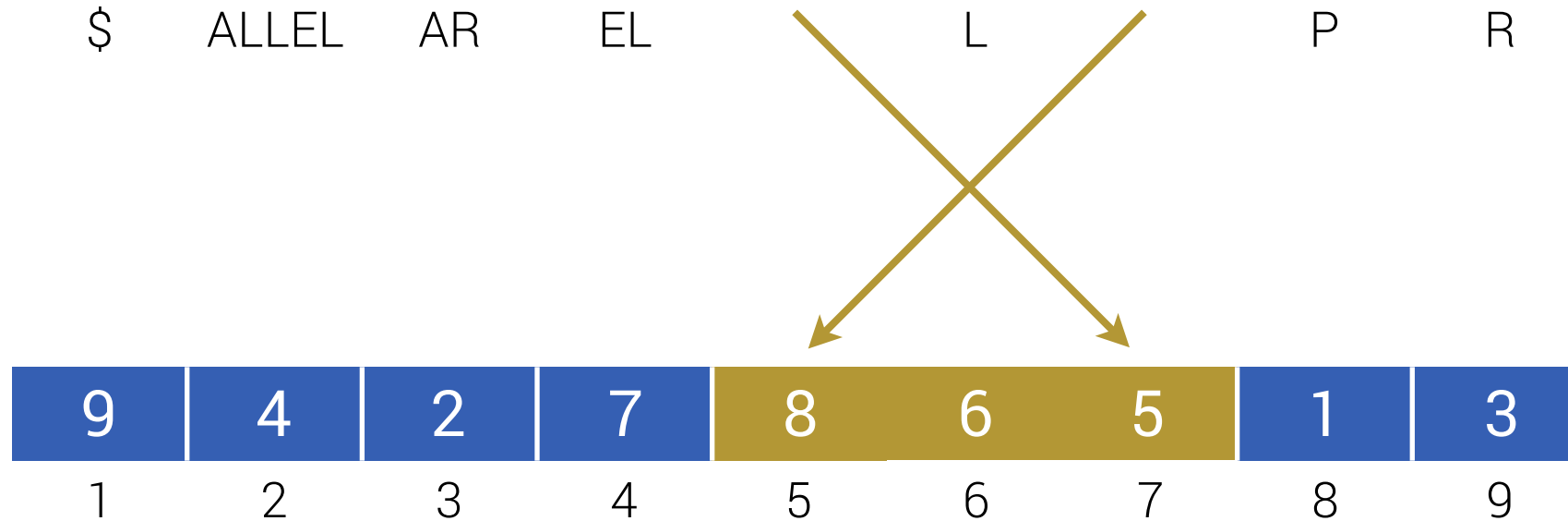
Eingabe



Phase 1

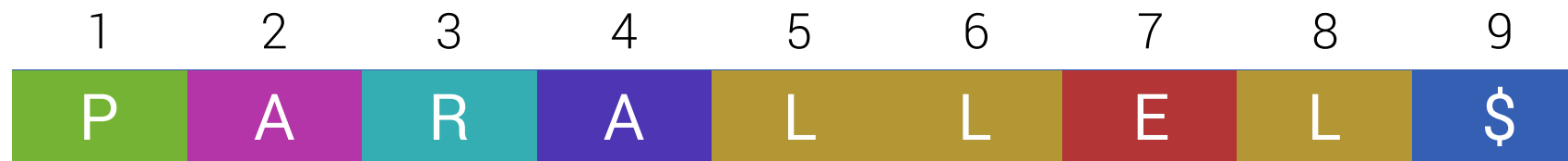


Phase 2

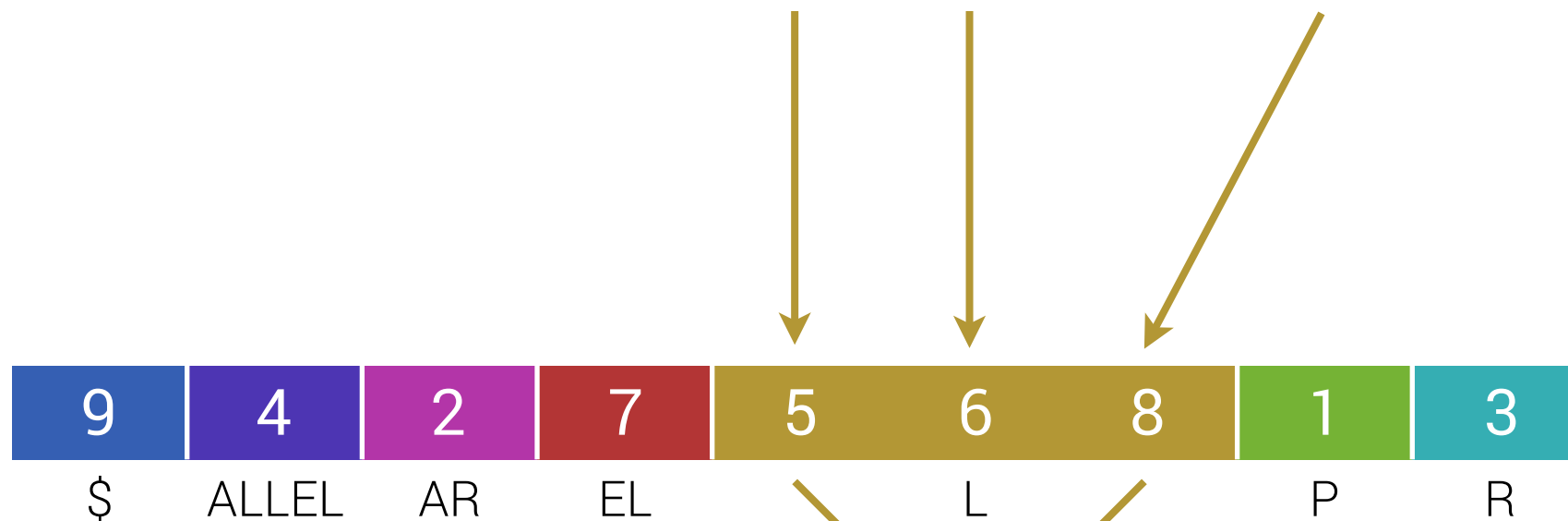


Grundprinzip

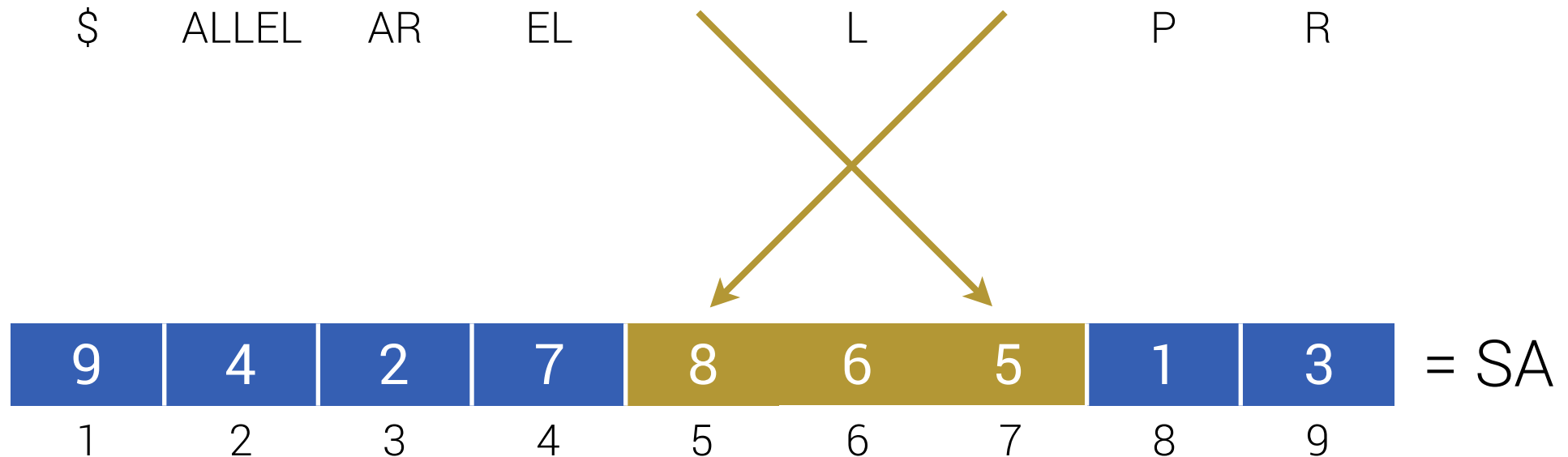
Eingabe



Phase 1



Phase 2



Grundprinzip

Eingabe



Phase 1 Sortierte Folge von Gruppen berechnen



Phase 2 Suffixe innerhalb der Gruppen sortieren

Grundprinzip

Eingabe



Phase 1 Sortierte Folge von Gruppen berechnen $O(n)$



Phase 2 Suffixe innerhalb der Gruppen sortieren $O(n)$

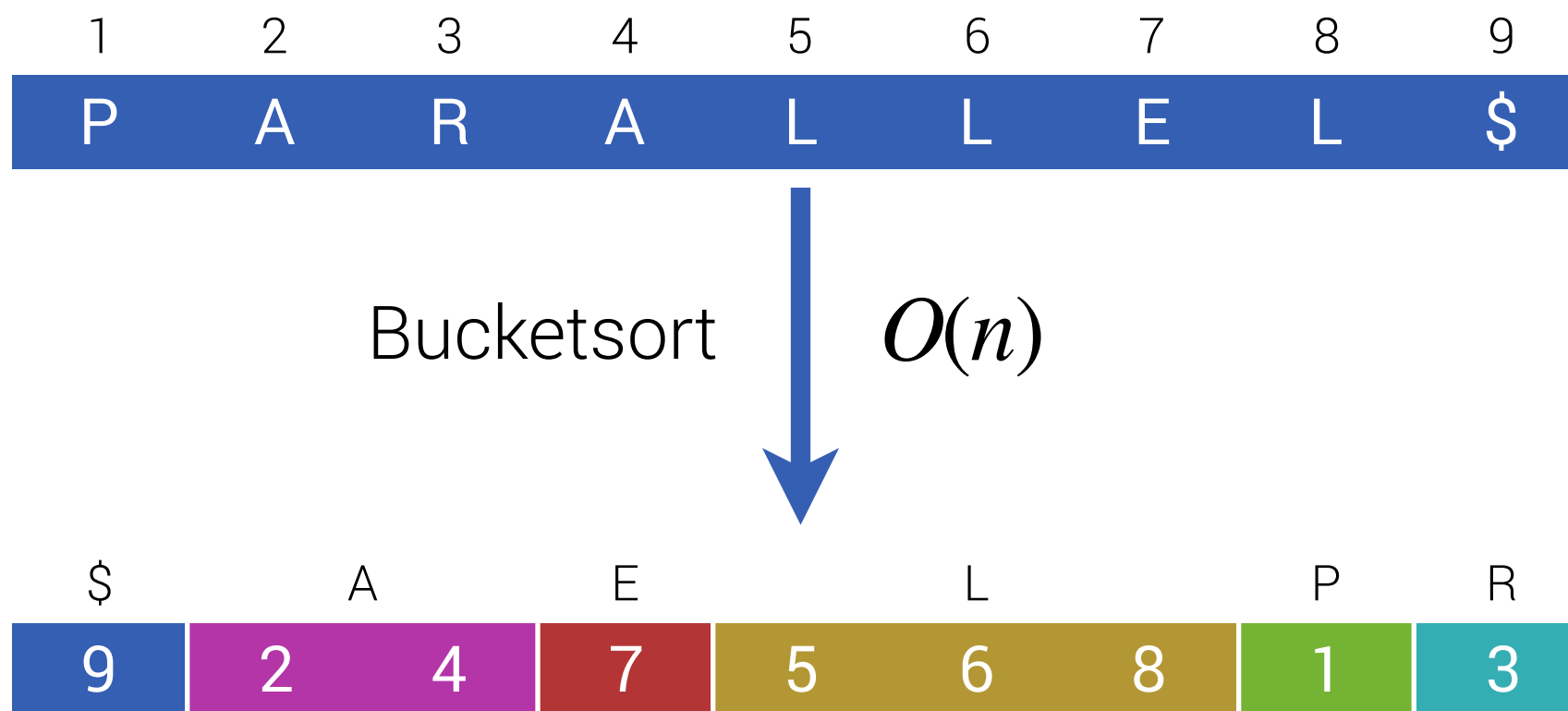
Phase 1

Sortierte Folge von Gruppen berechnen

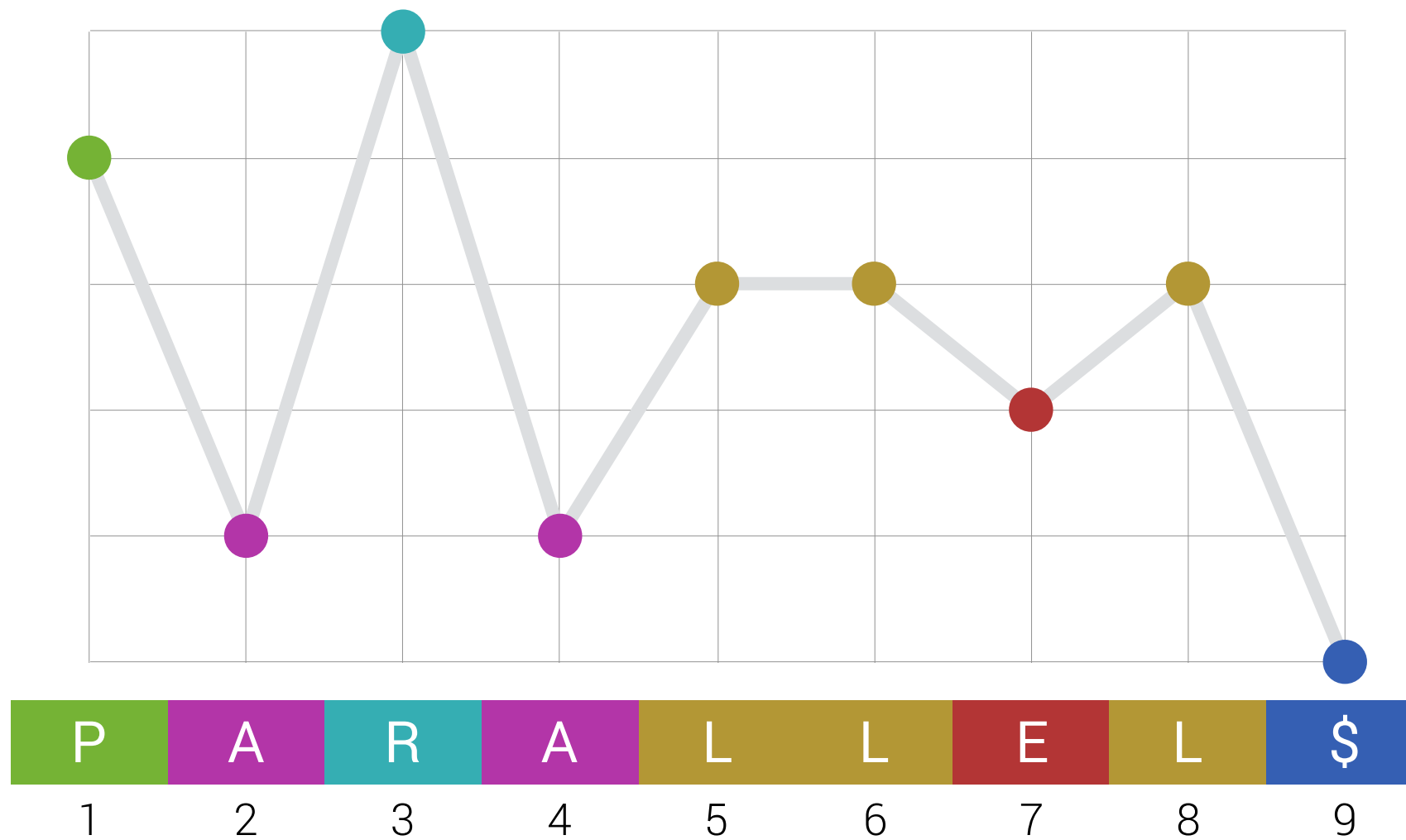
1	2	3	4	5	6	7	8	9
P	A	R	A	L	L	E	L	\$

Phase 1

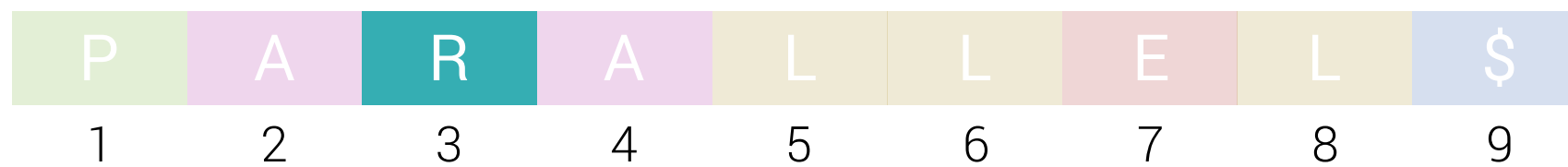
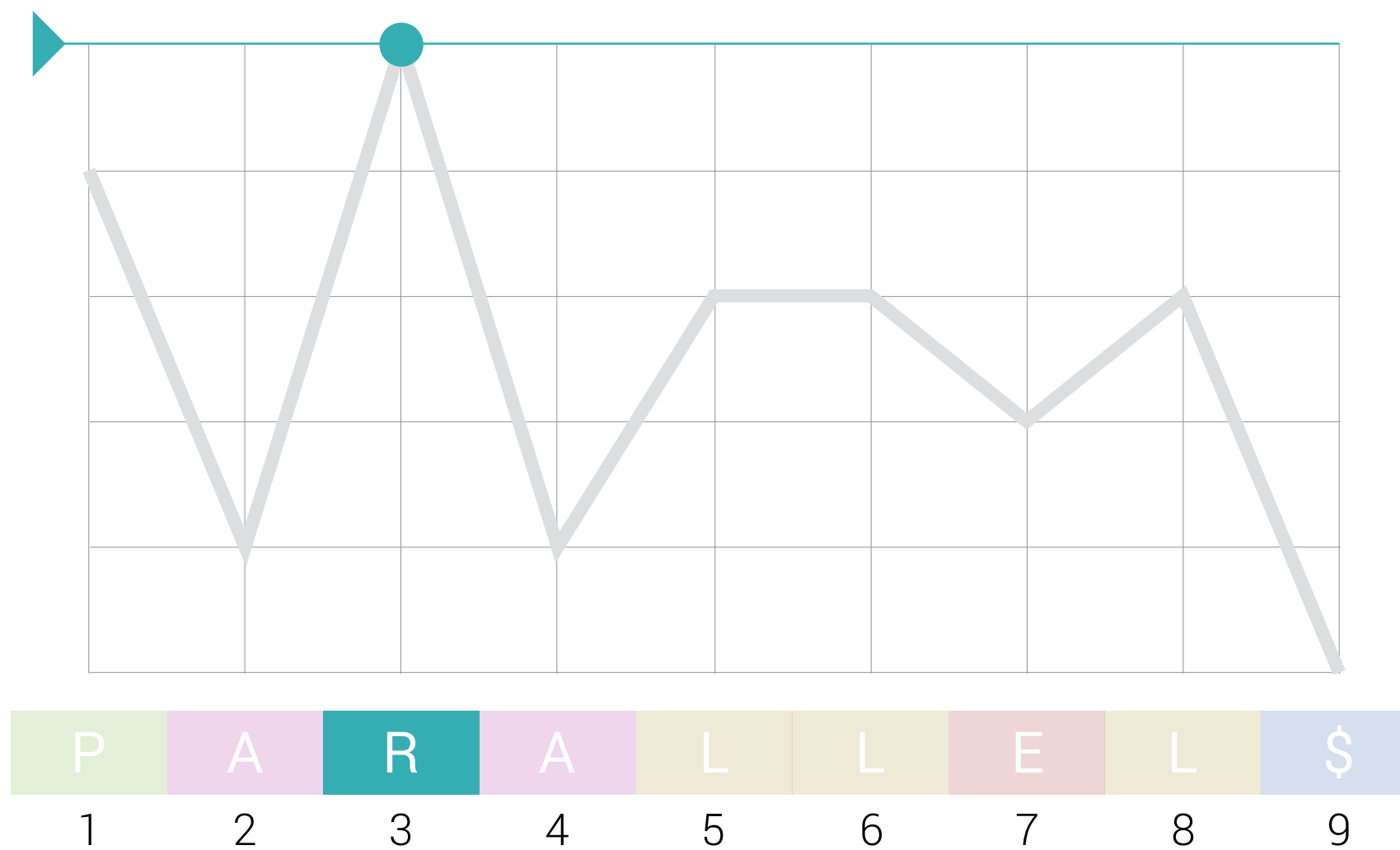
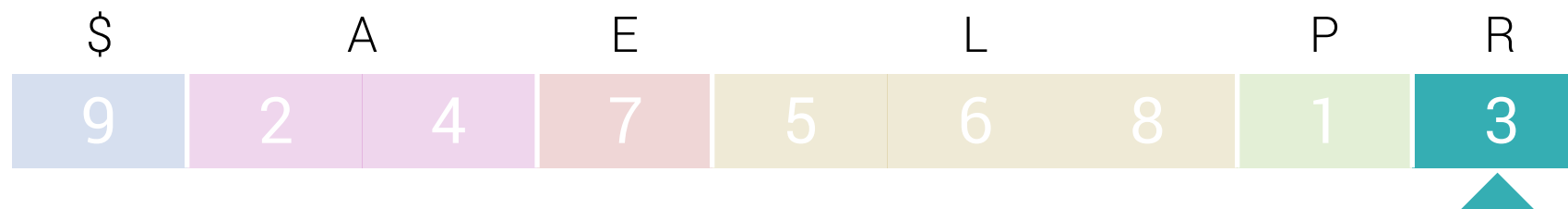
Sortierte Folge von Gruppen berechnen



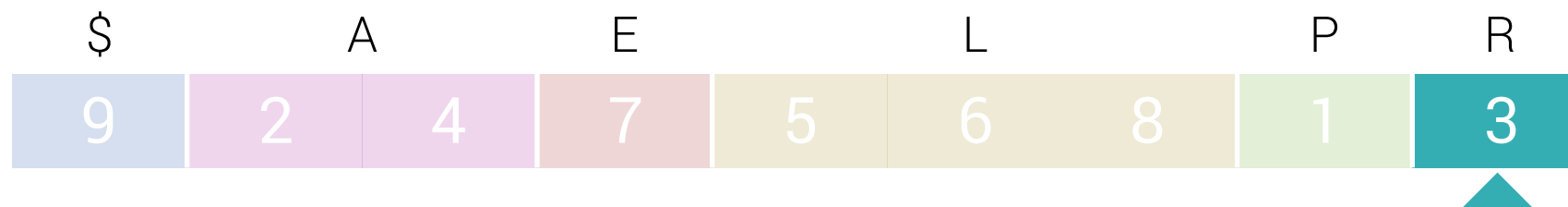
Phase 1



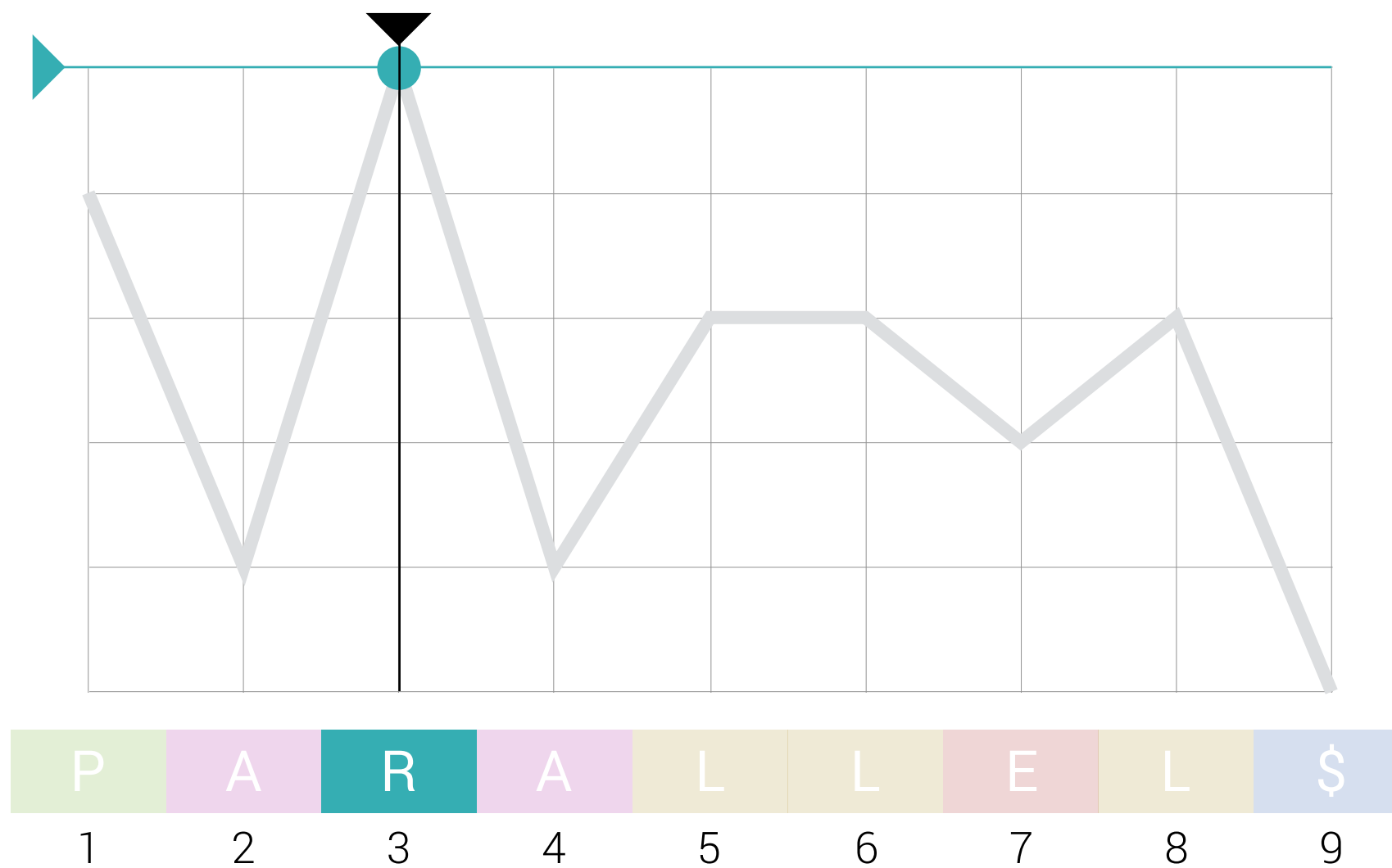
Phase 1



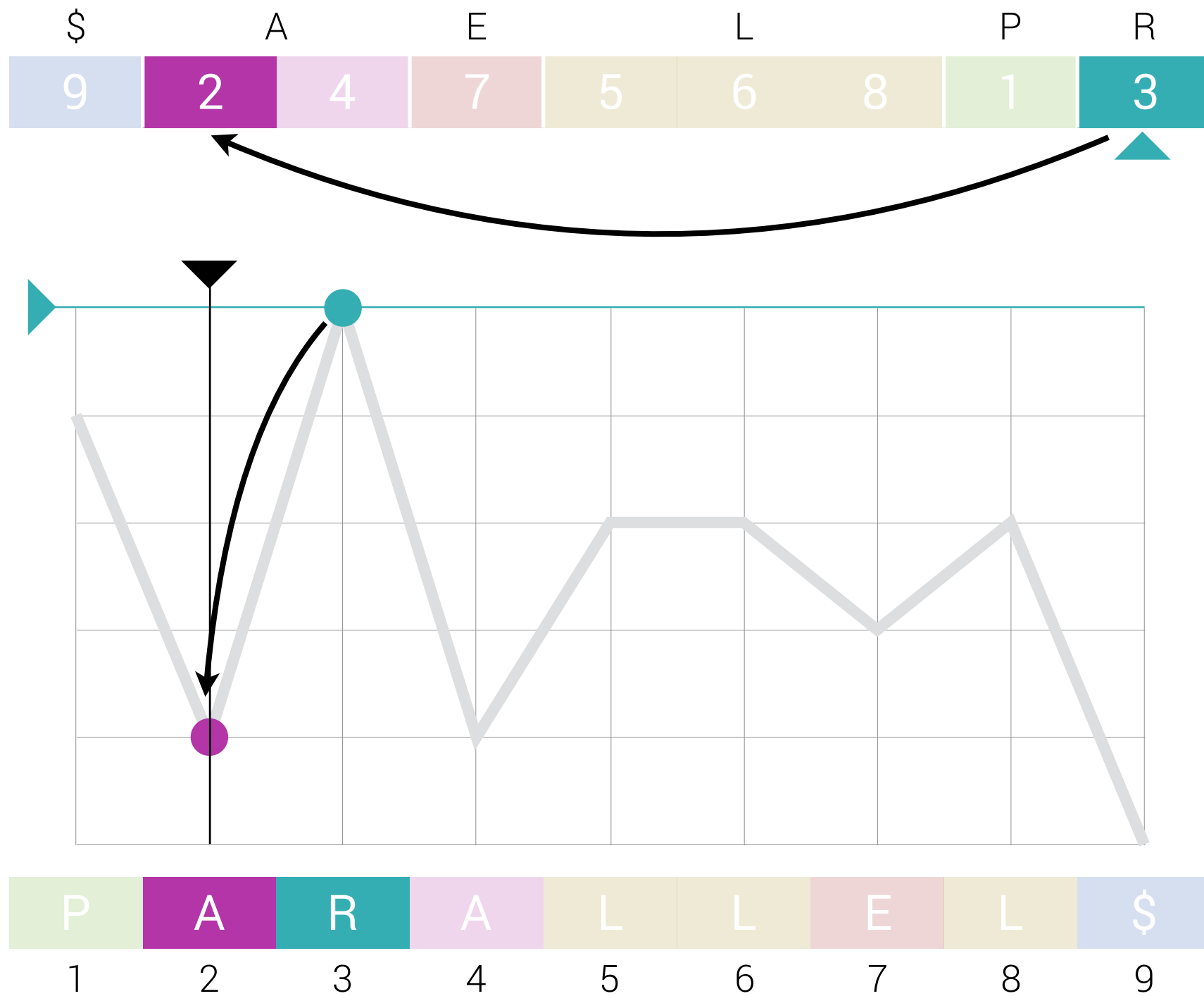
Phase 1



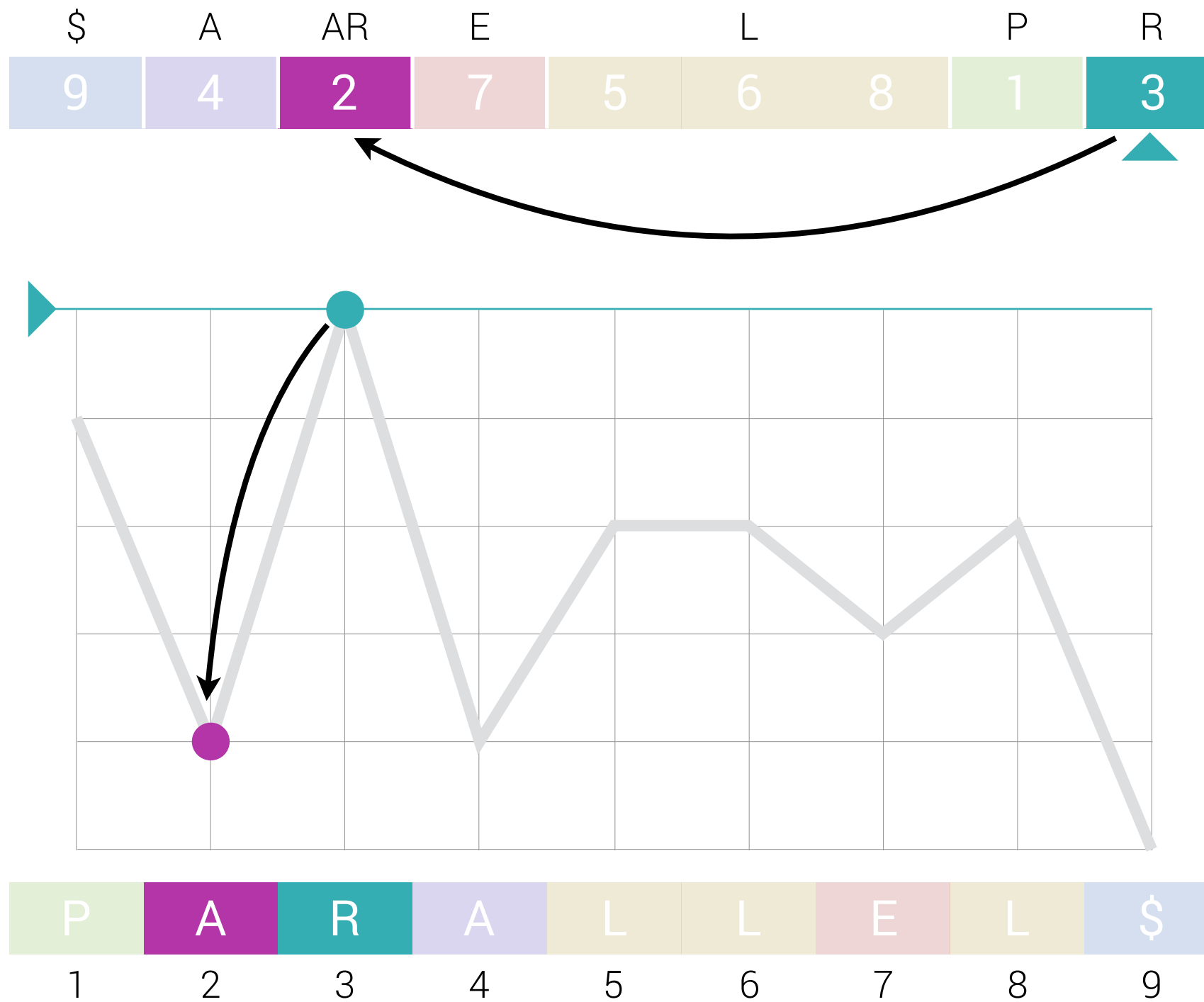
$prev(i) := \max \{ j \in [1 .. i] : \text{Gr.kontext } S_j <_{\text{lex}} \text{Gr.kontext } S_i \}$



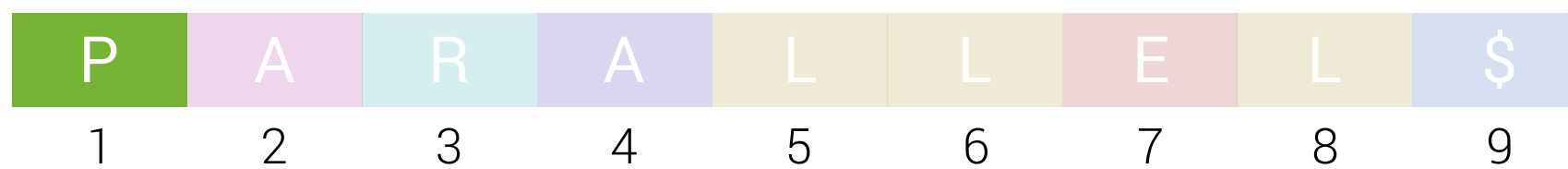
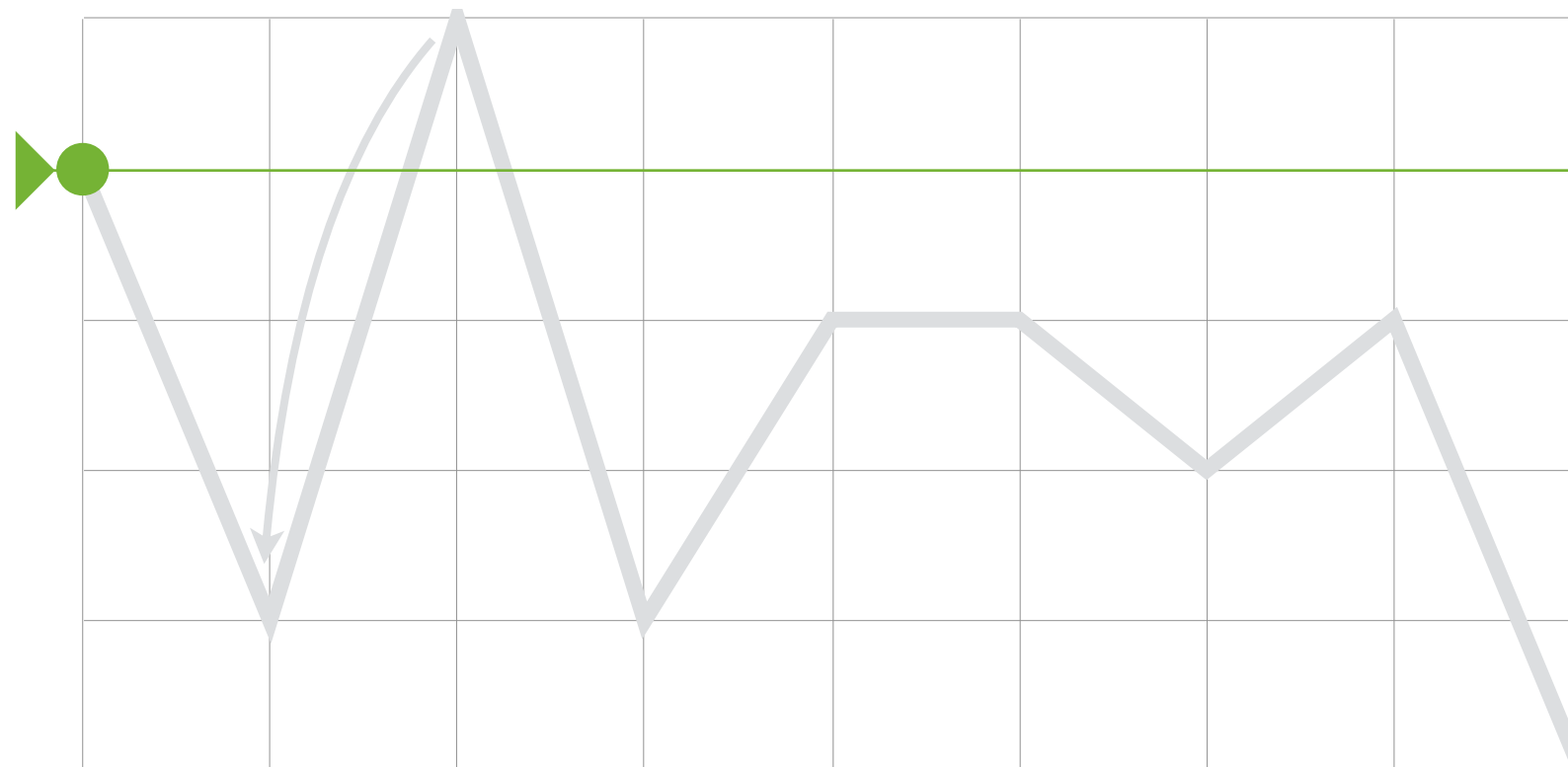
Phase 1



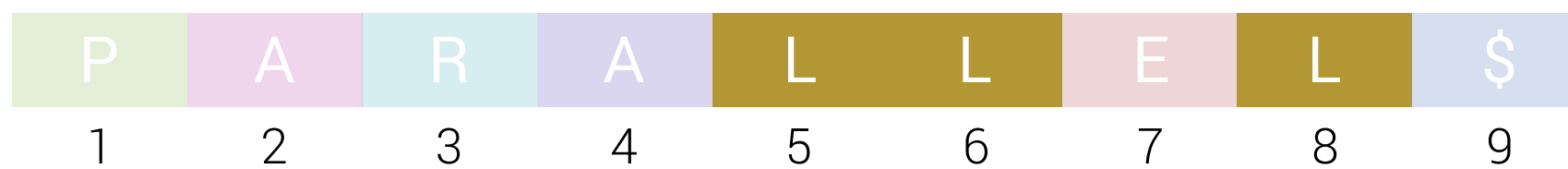
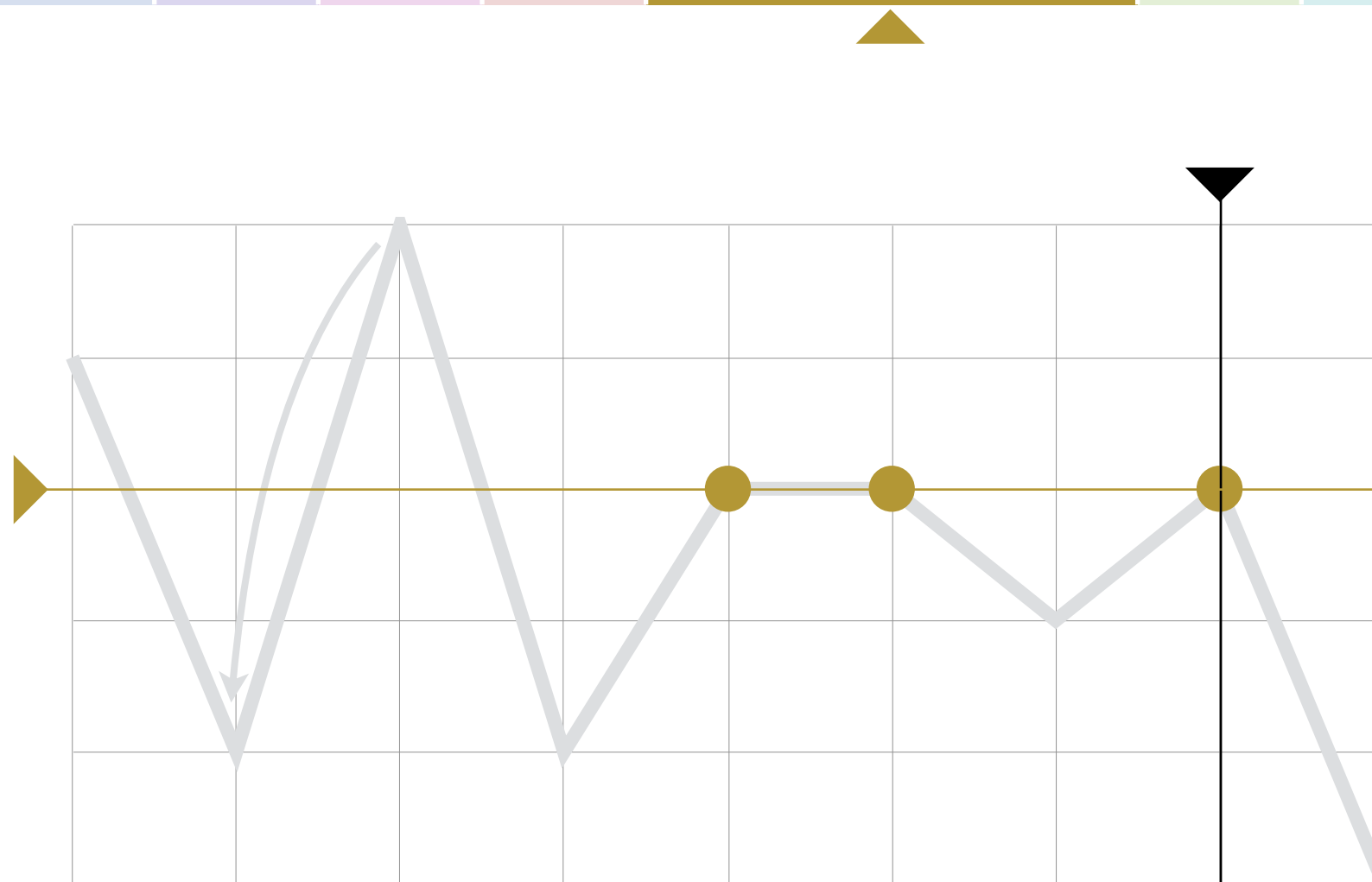
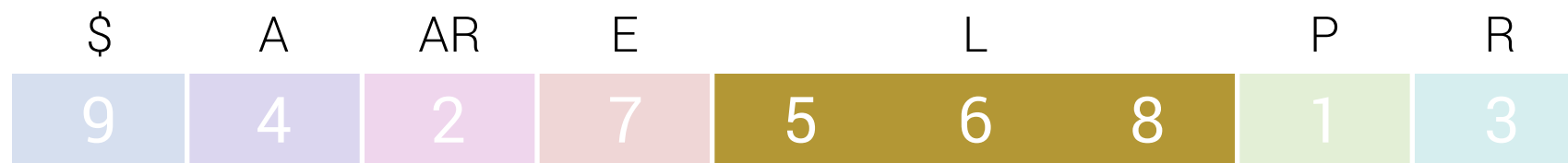
Phase 1



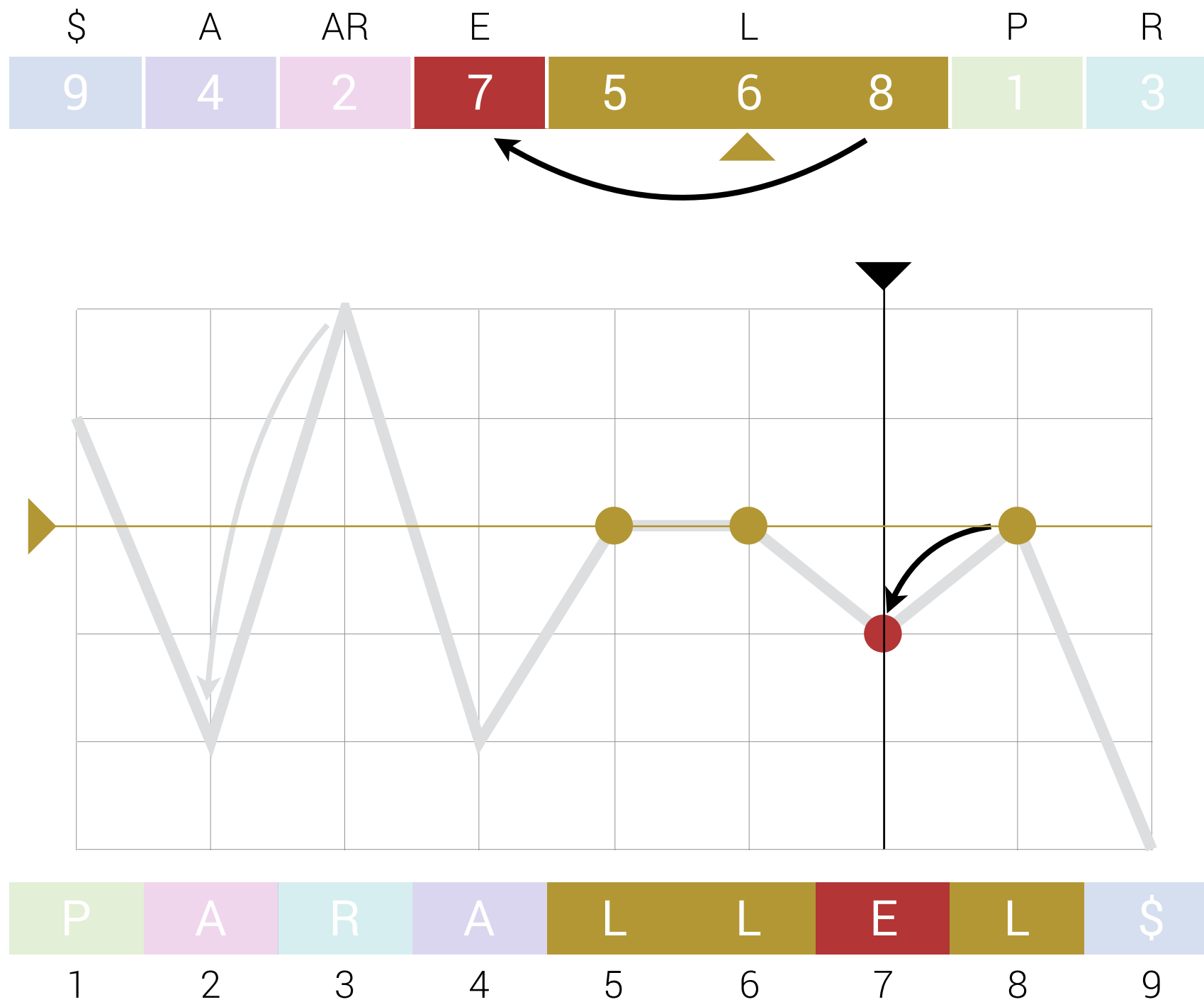
Phase 1



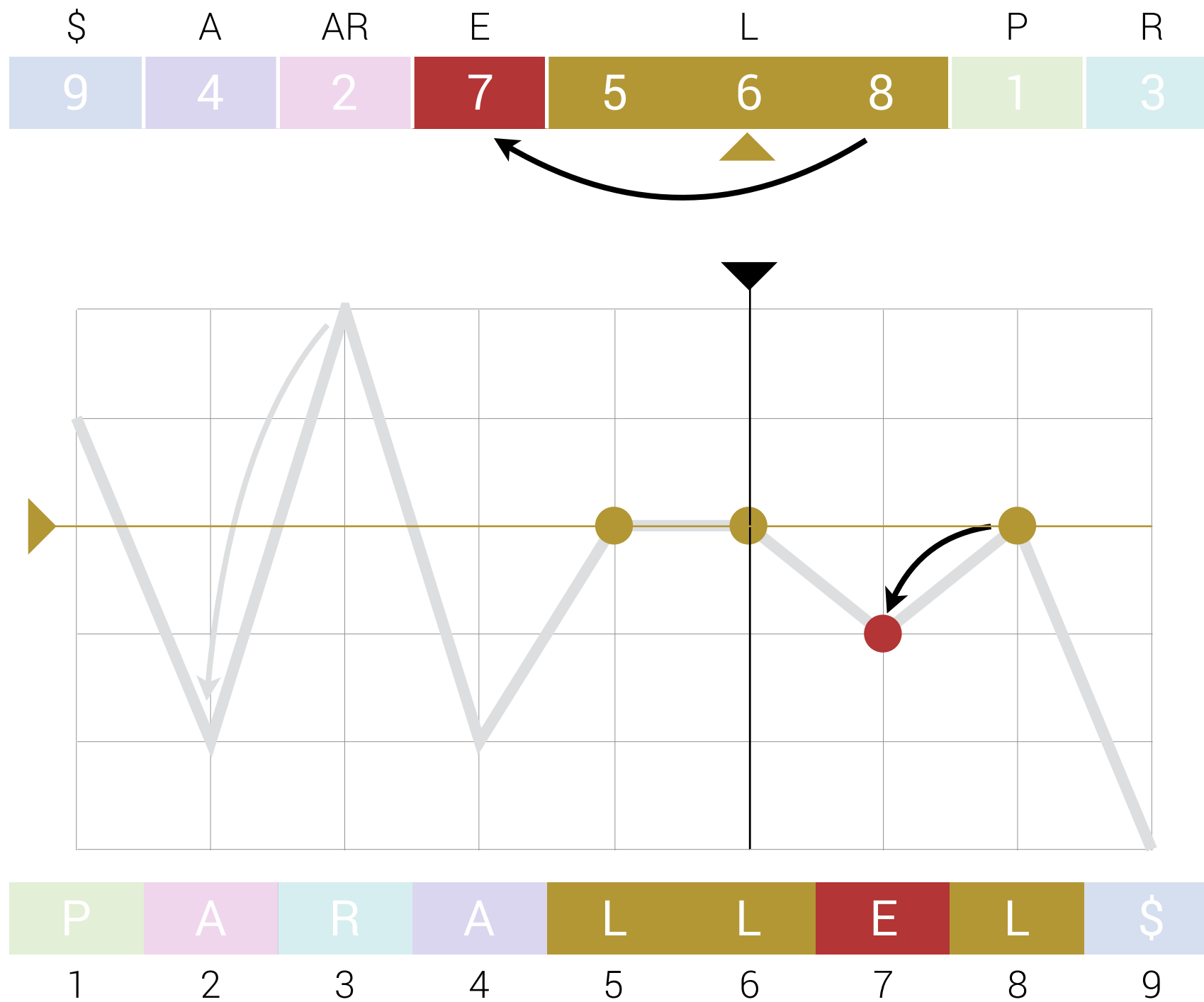
Phase 1



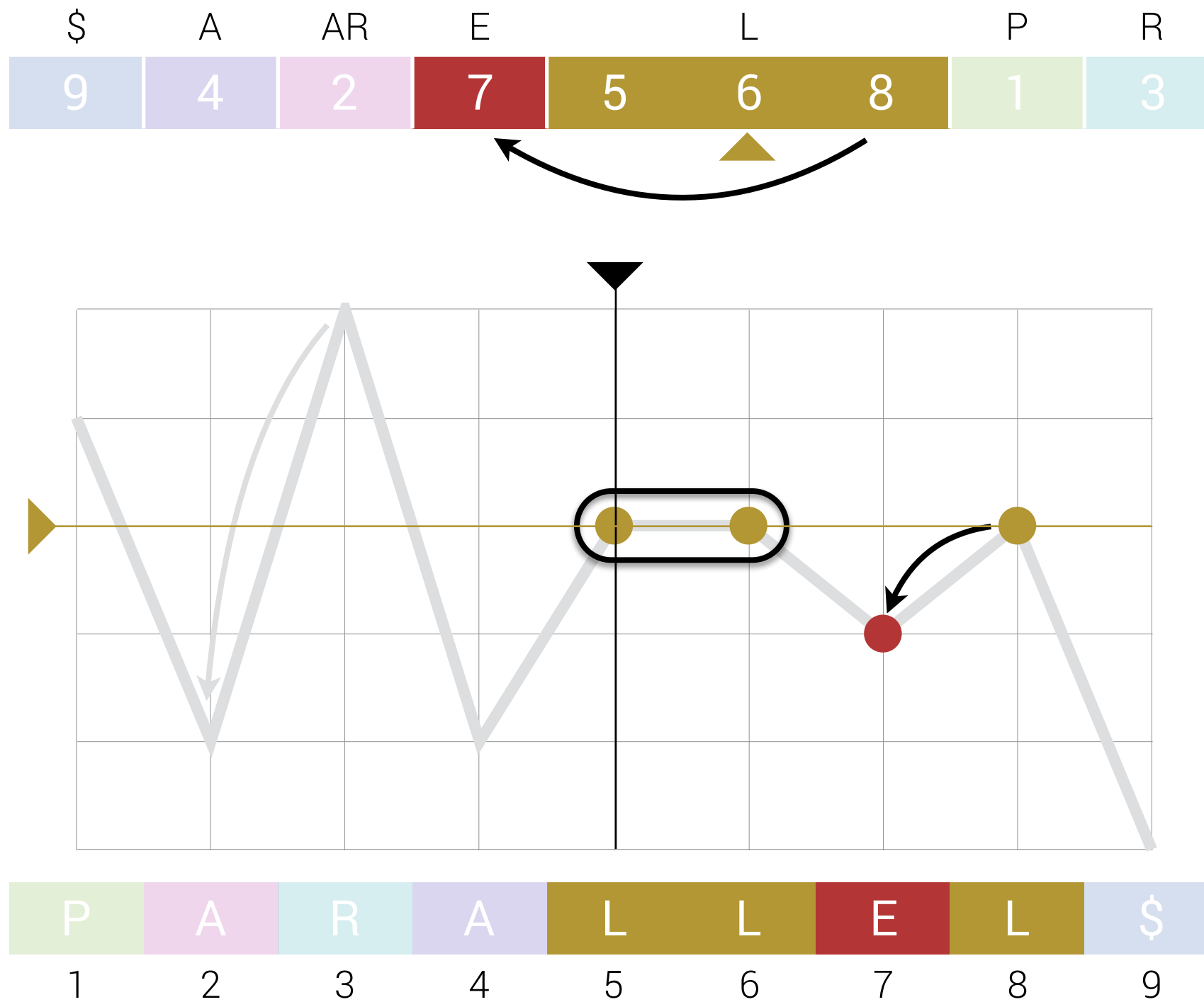
Phase 1



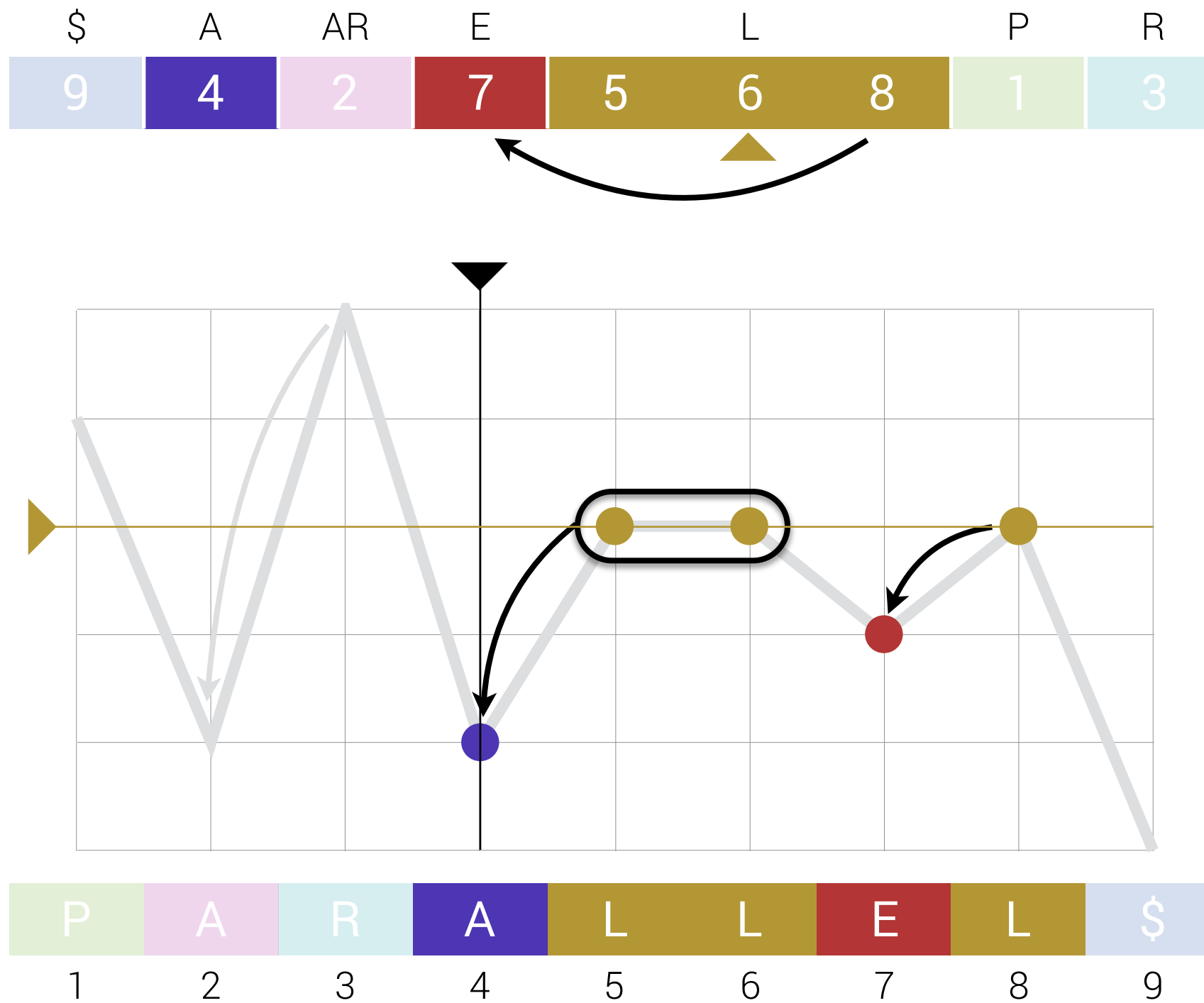
Phase 1



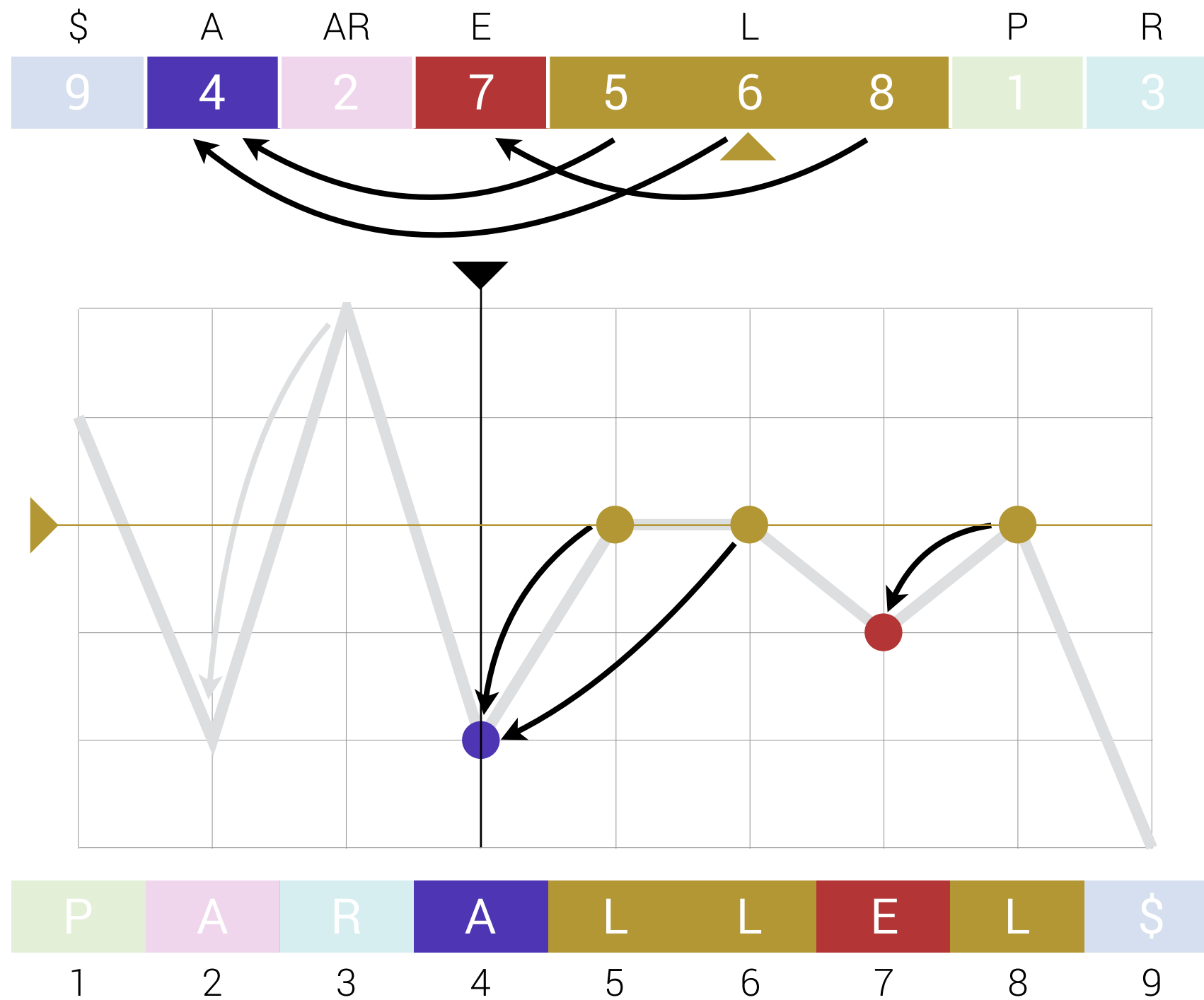
Phase 1



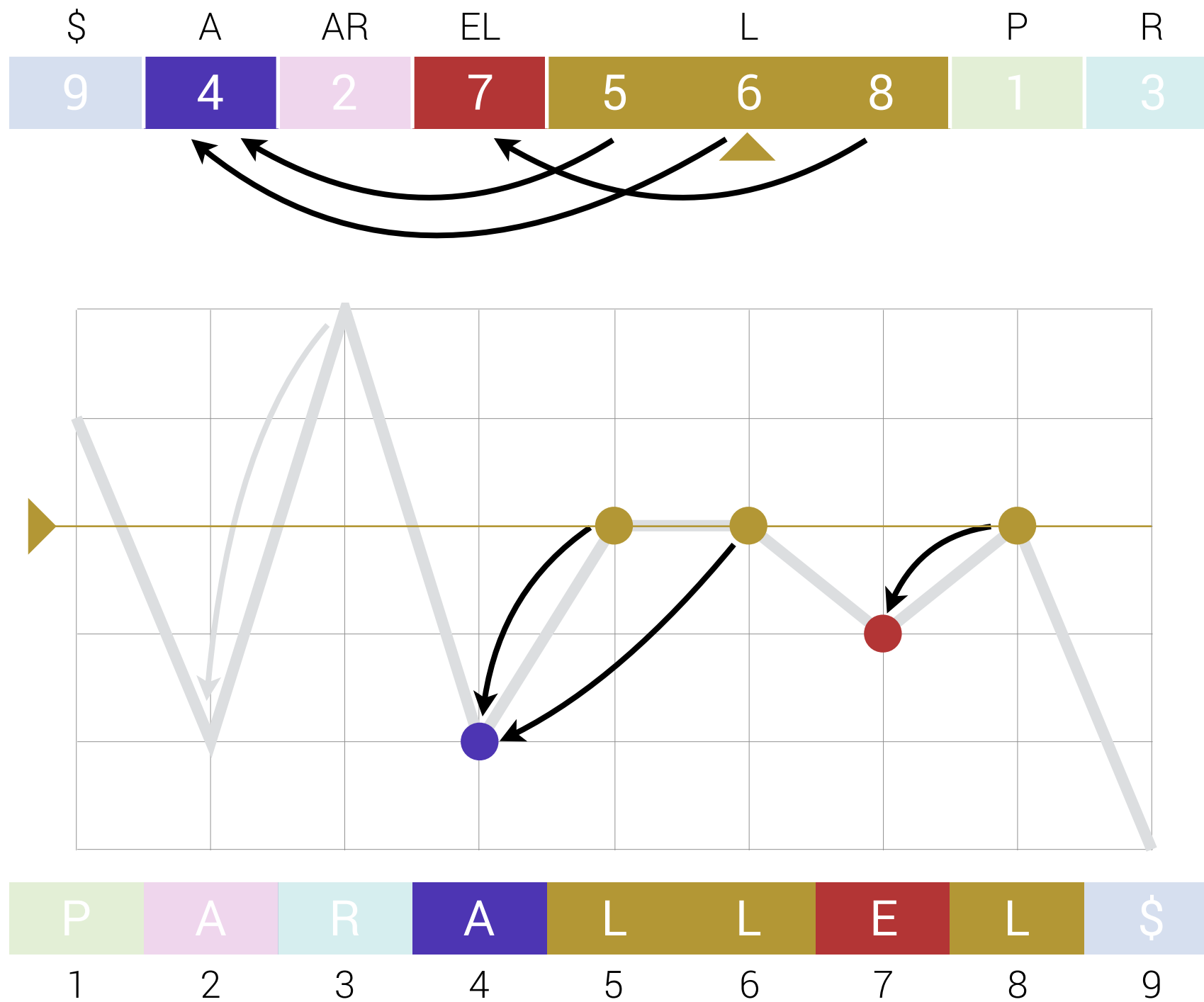
Phase 1



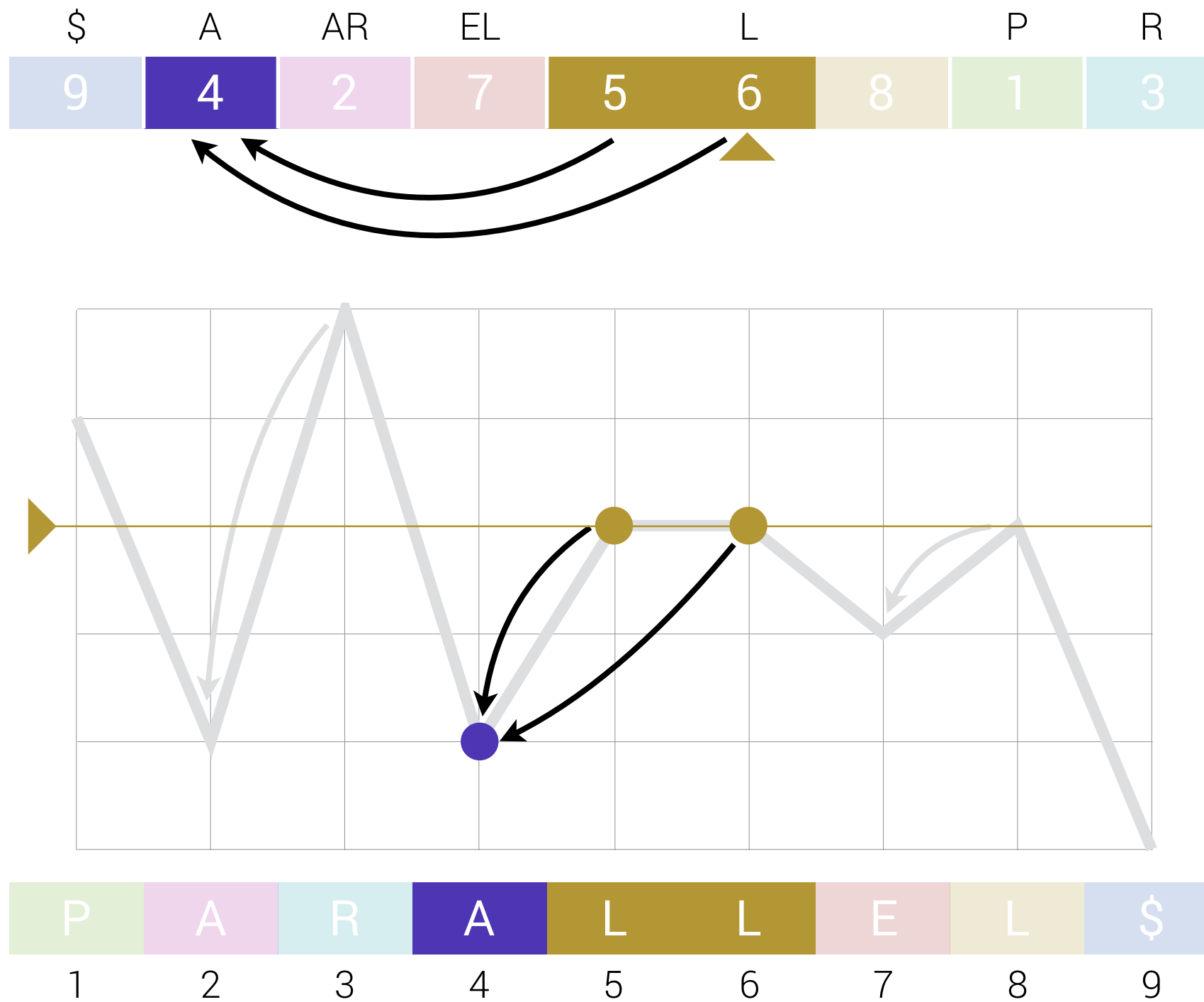
Phase 1



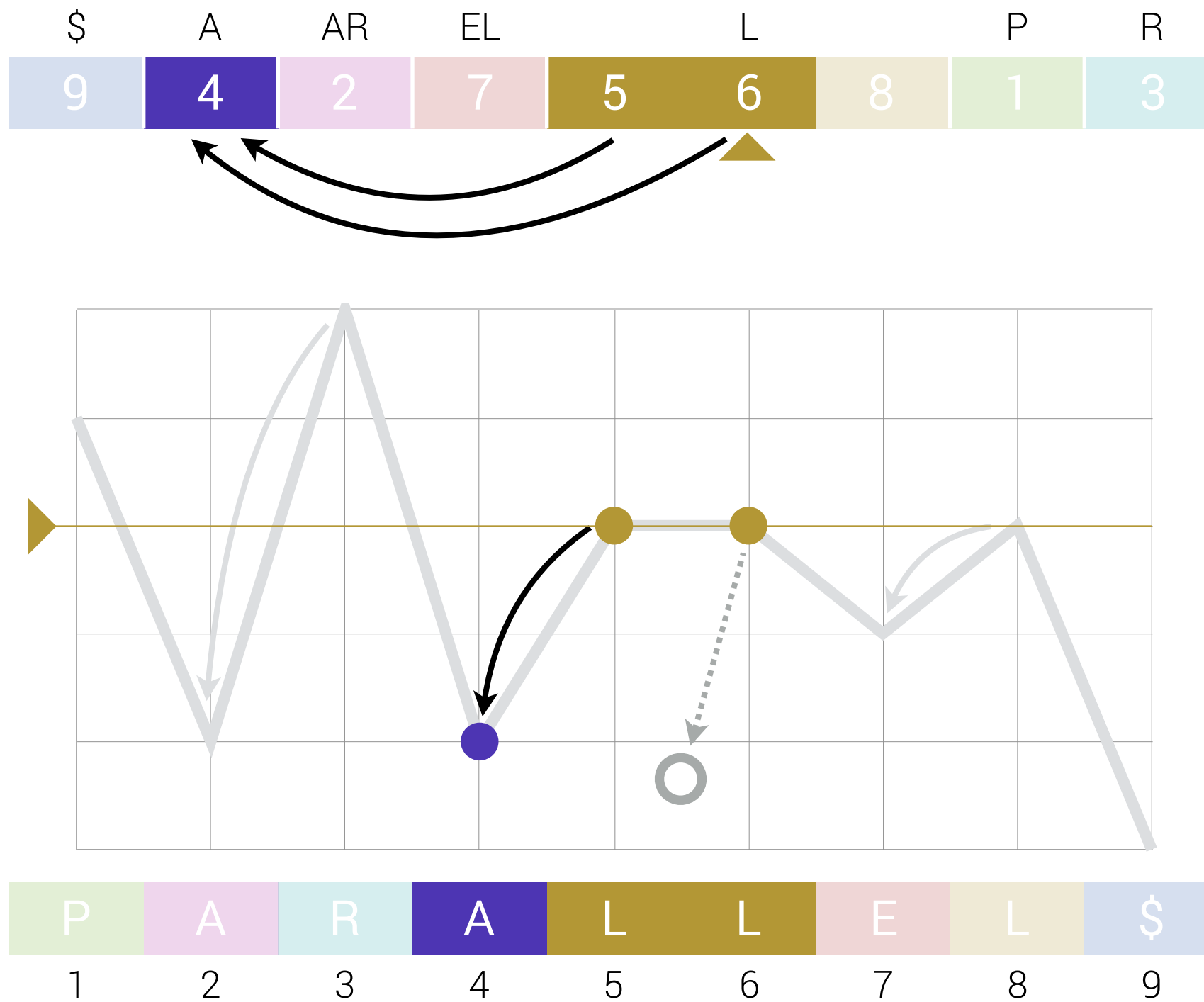
Phase 1



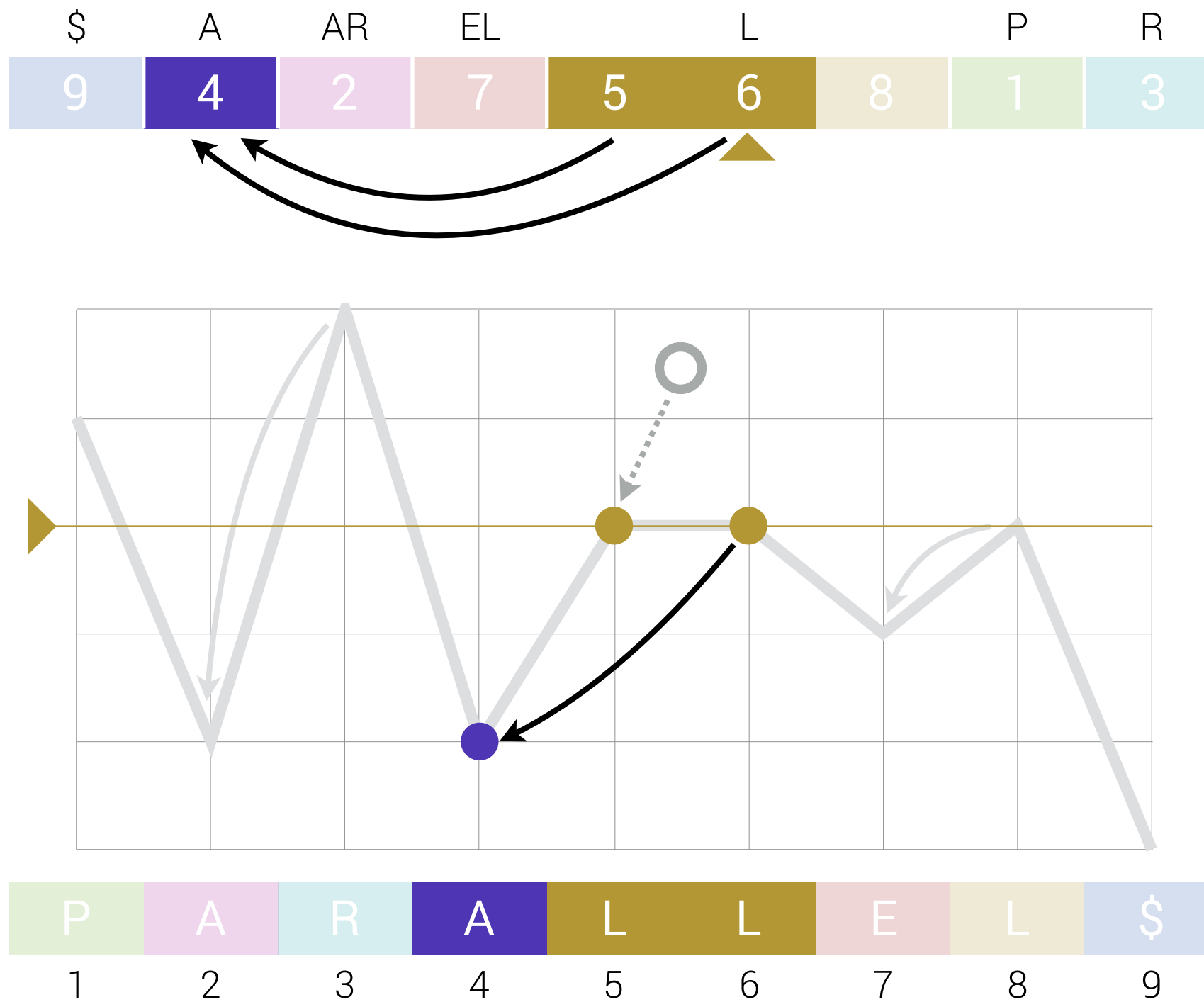
Phase 1



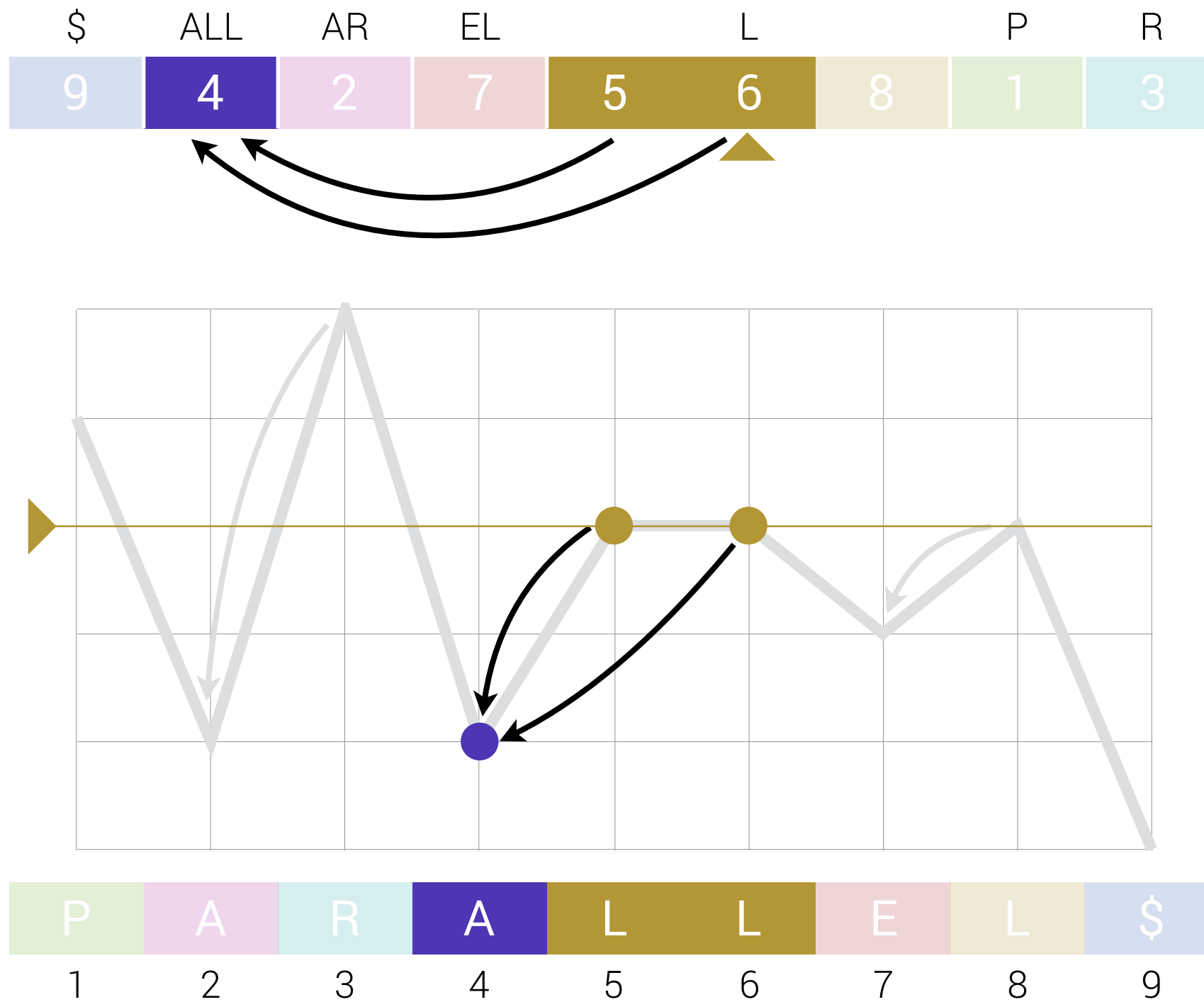
Phase 1



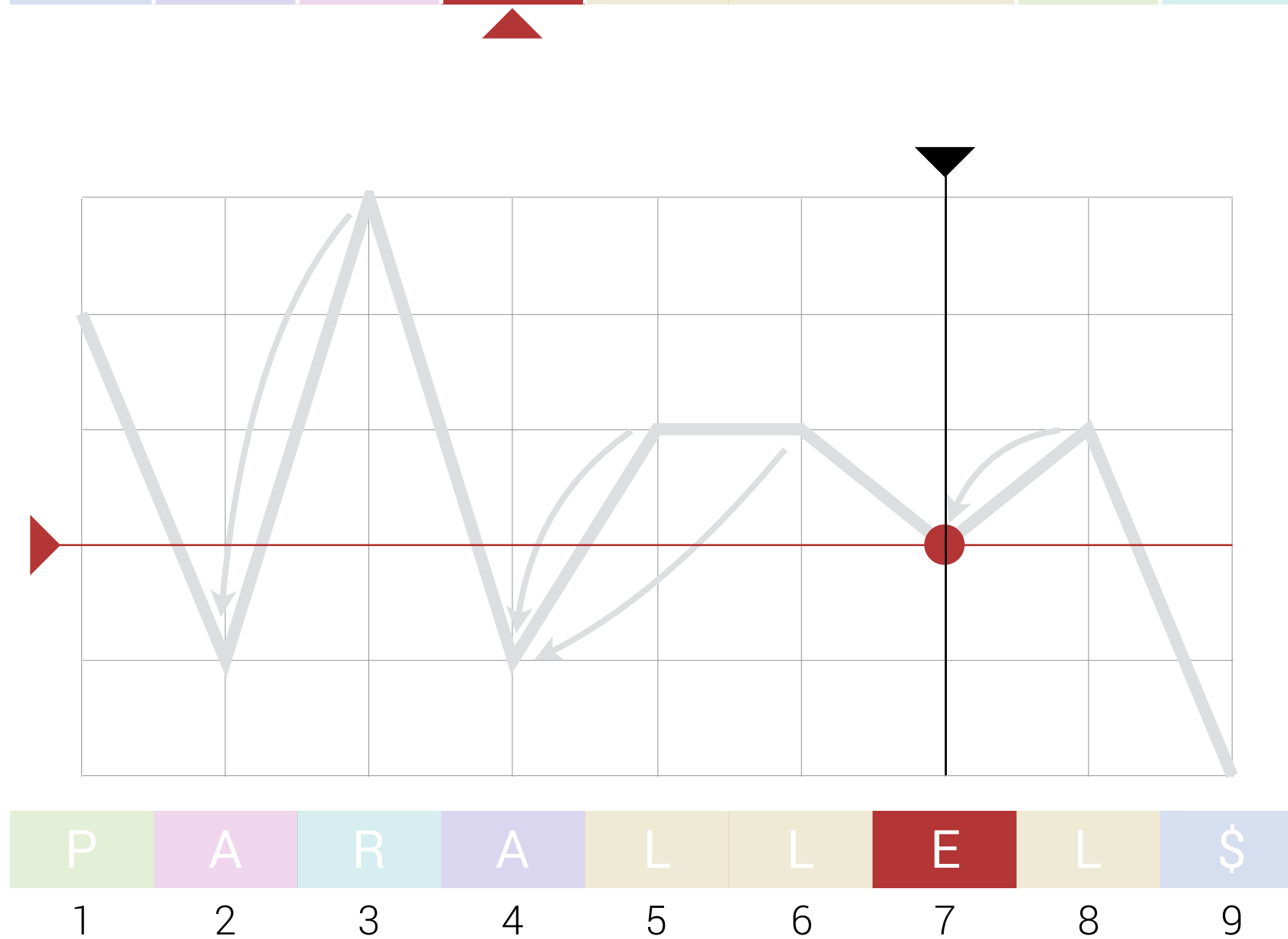
Phase 1



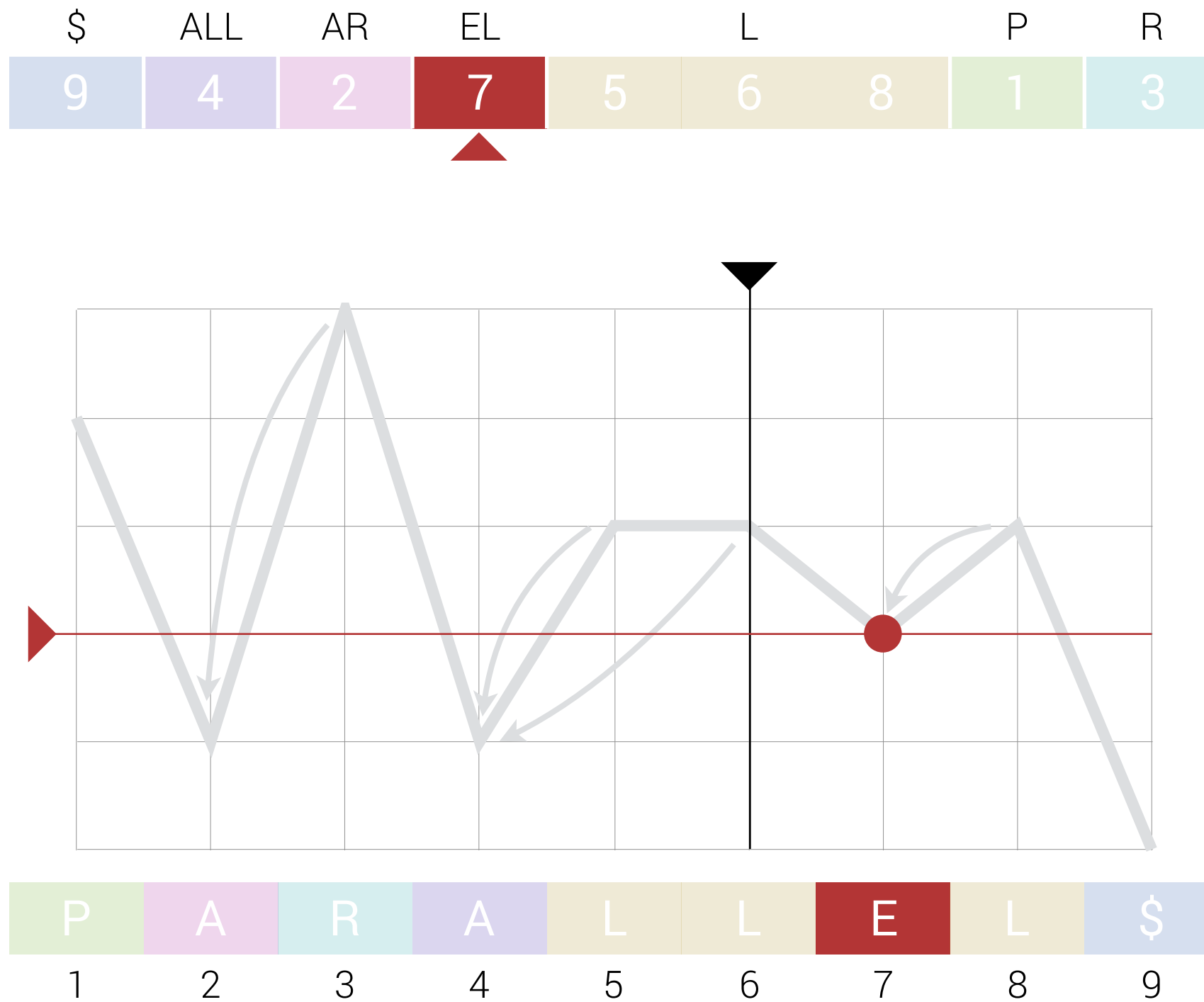
Phase 1



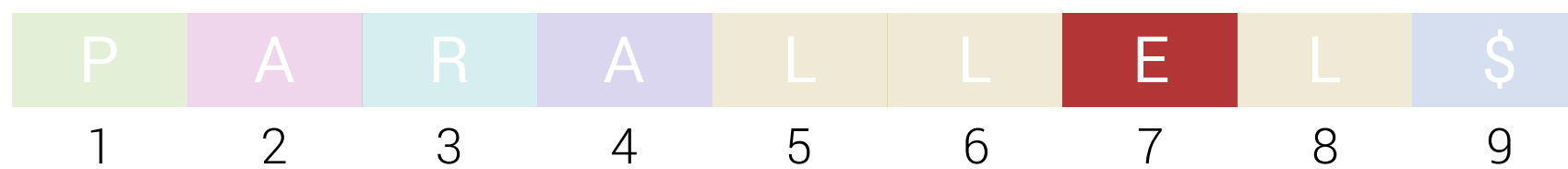
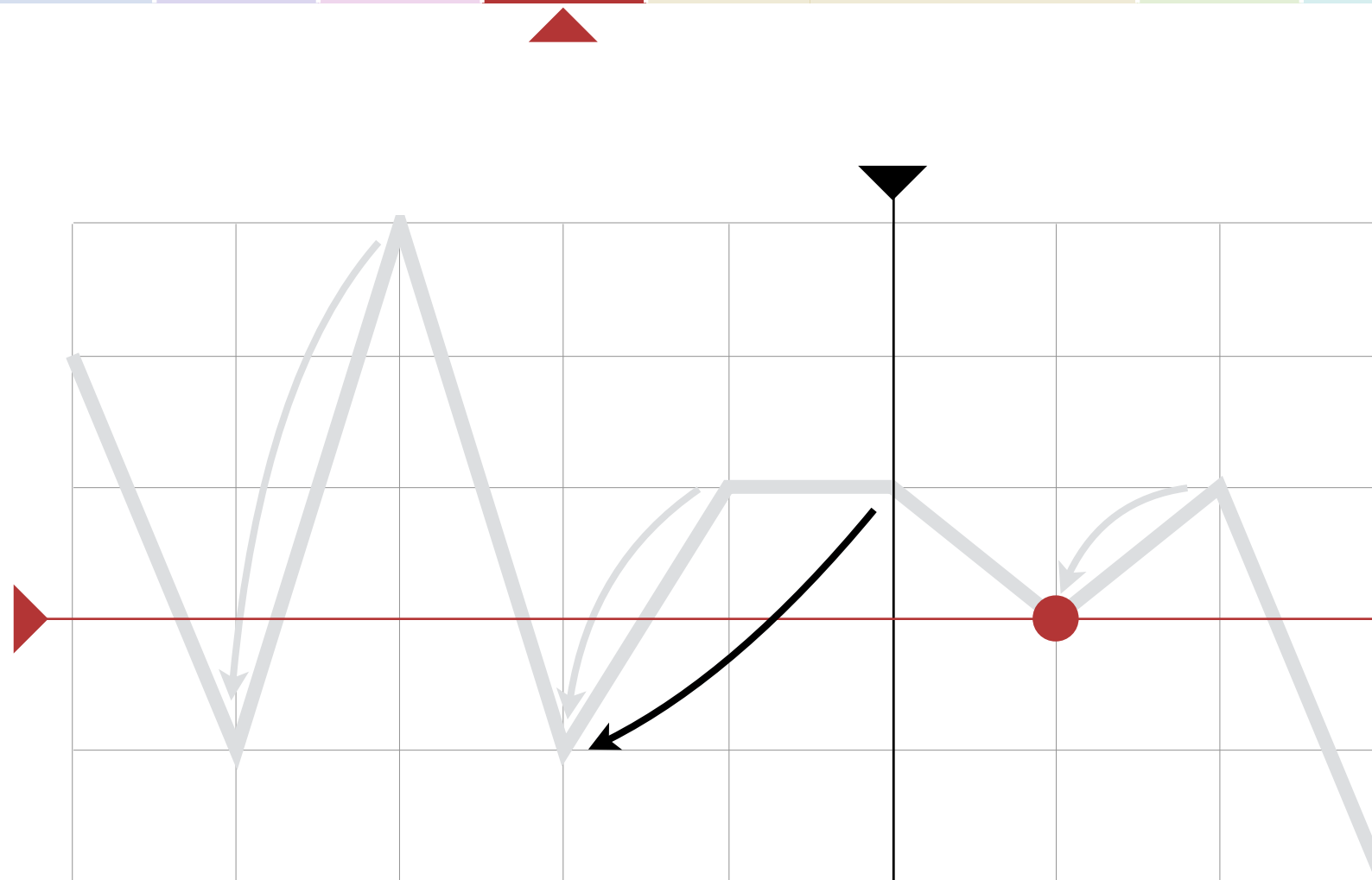
Phase 1



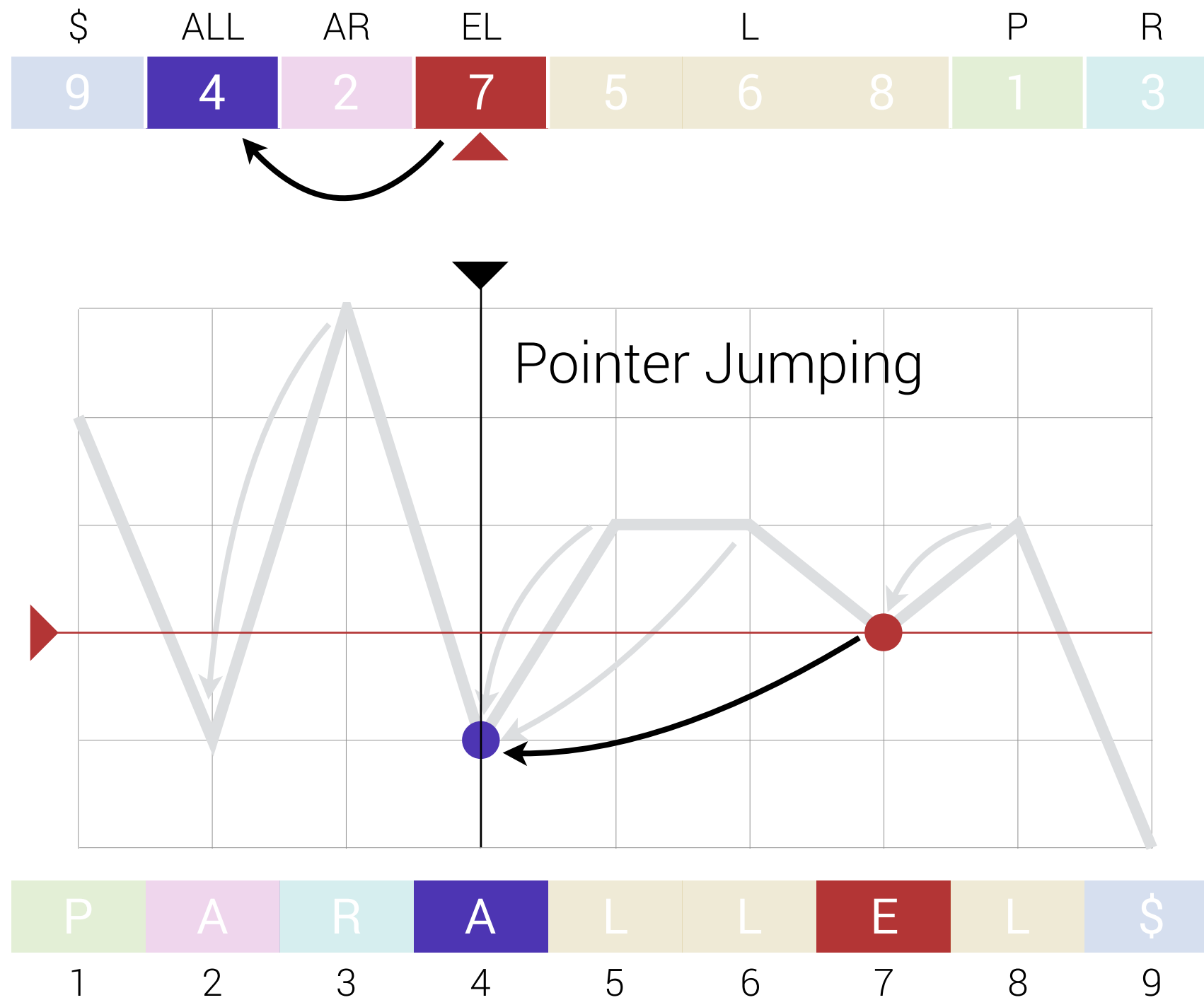
Phase 1



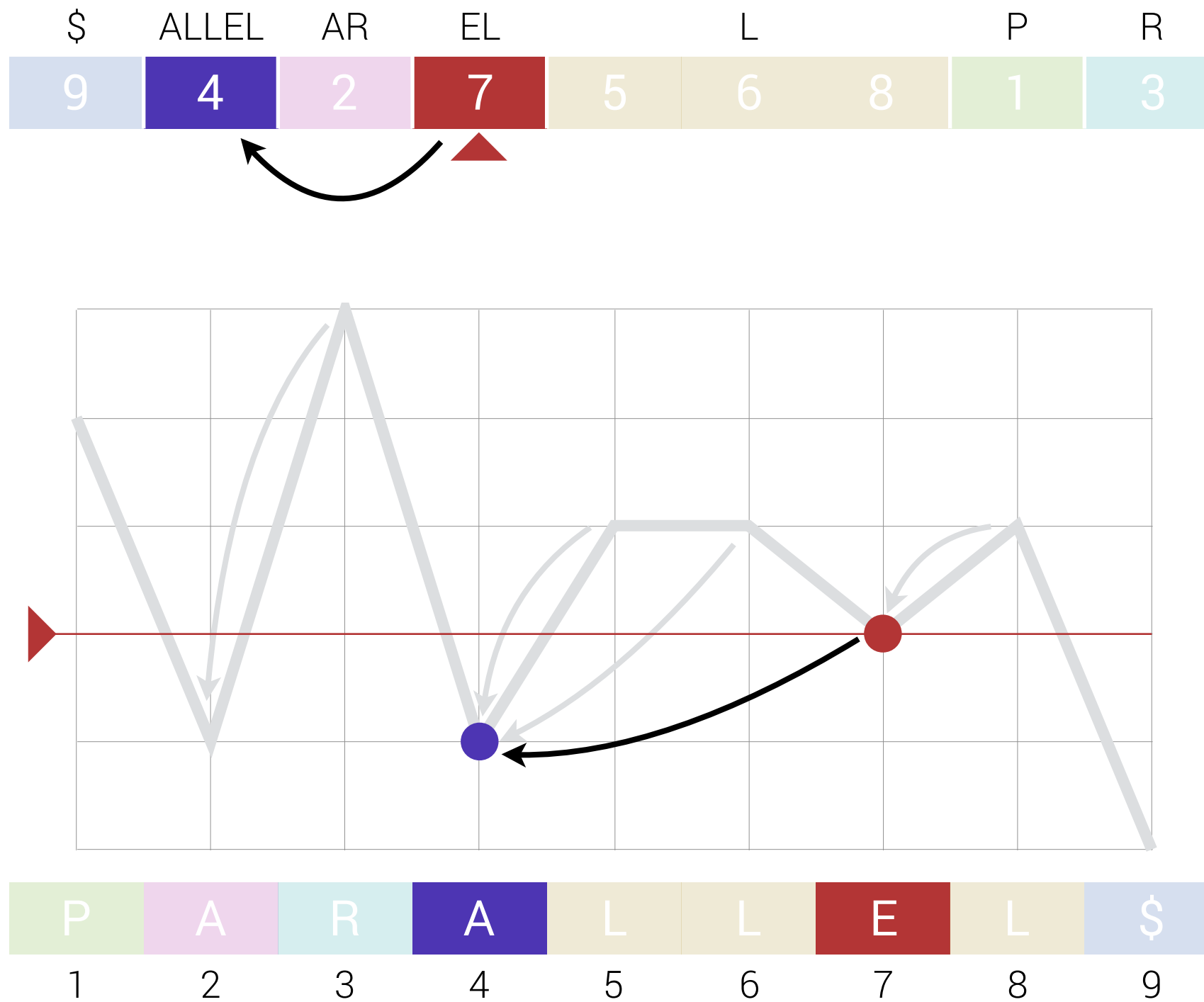
Phase 1



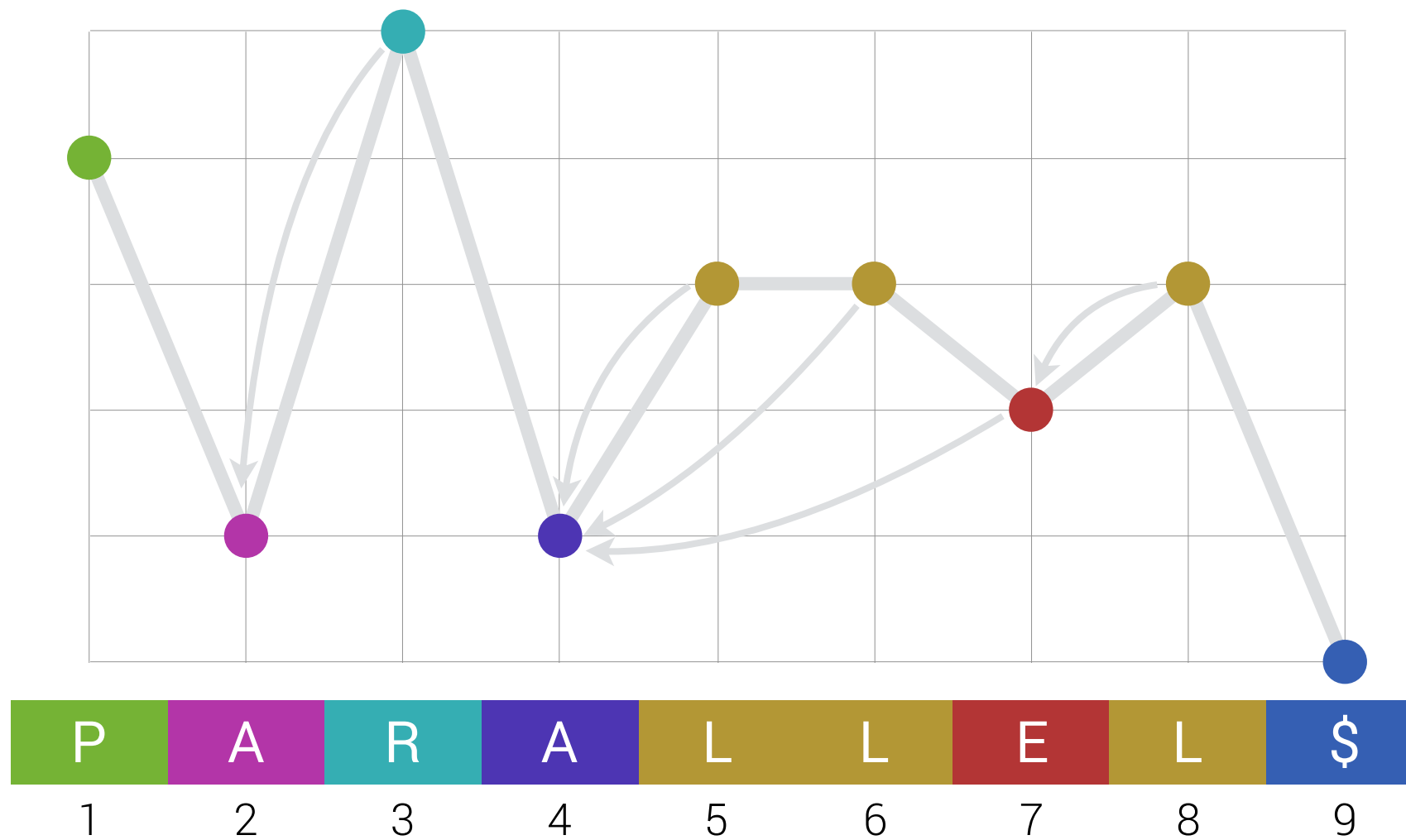
Phase 1



Phase 1

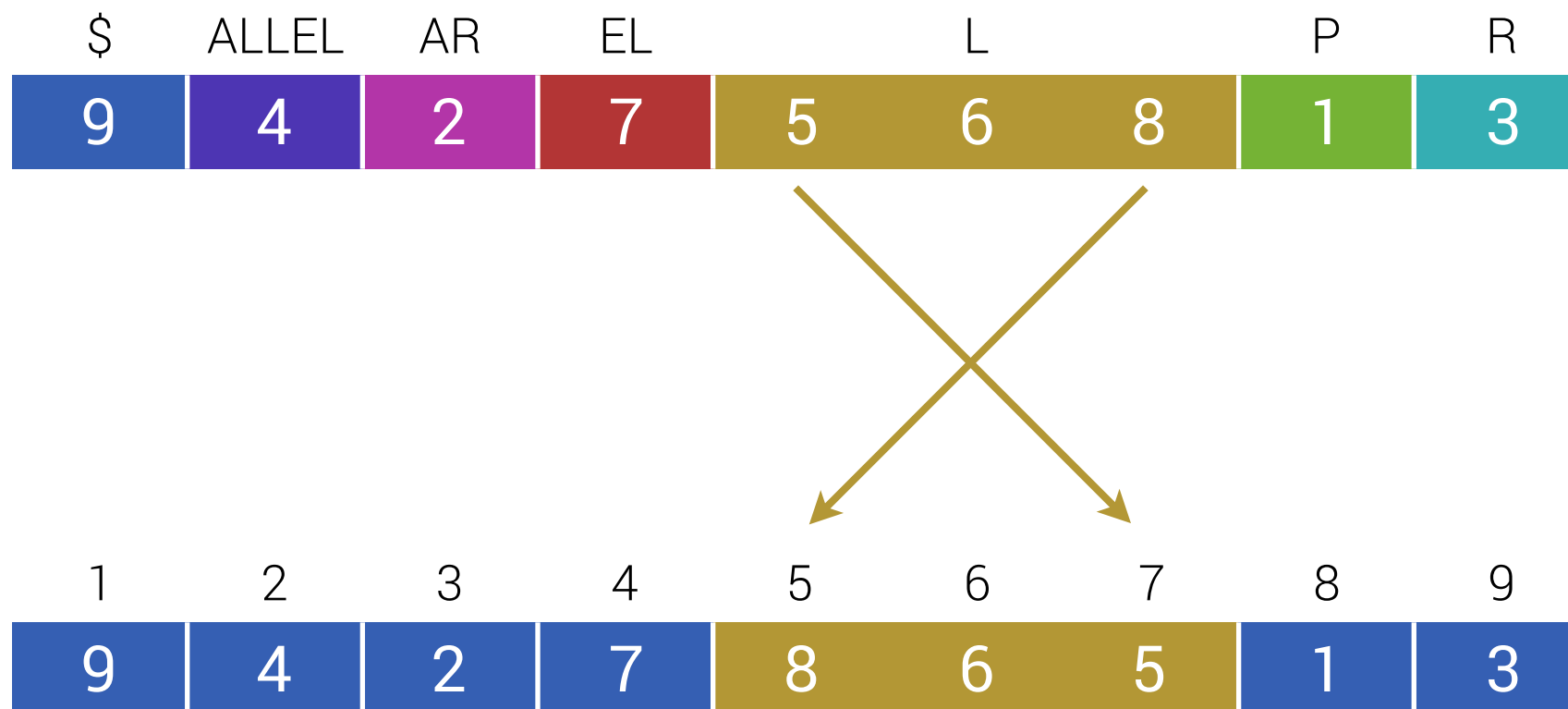


Phase 1



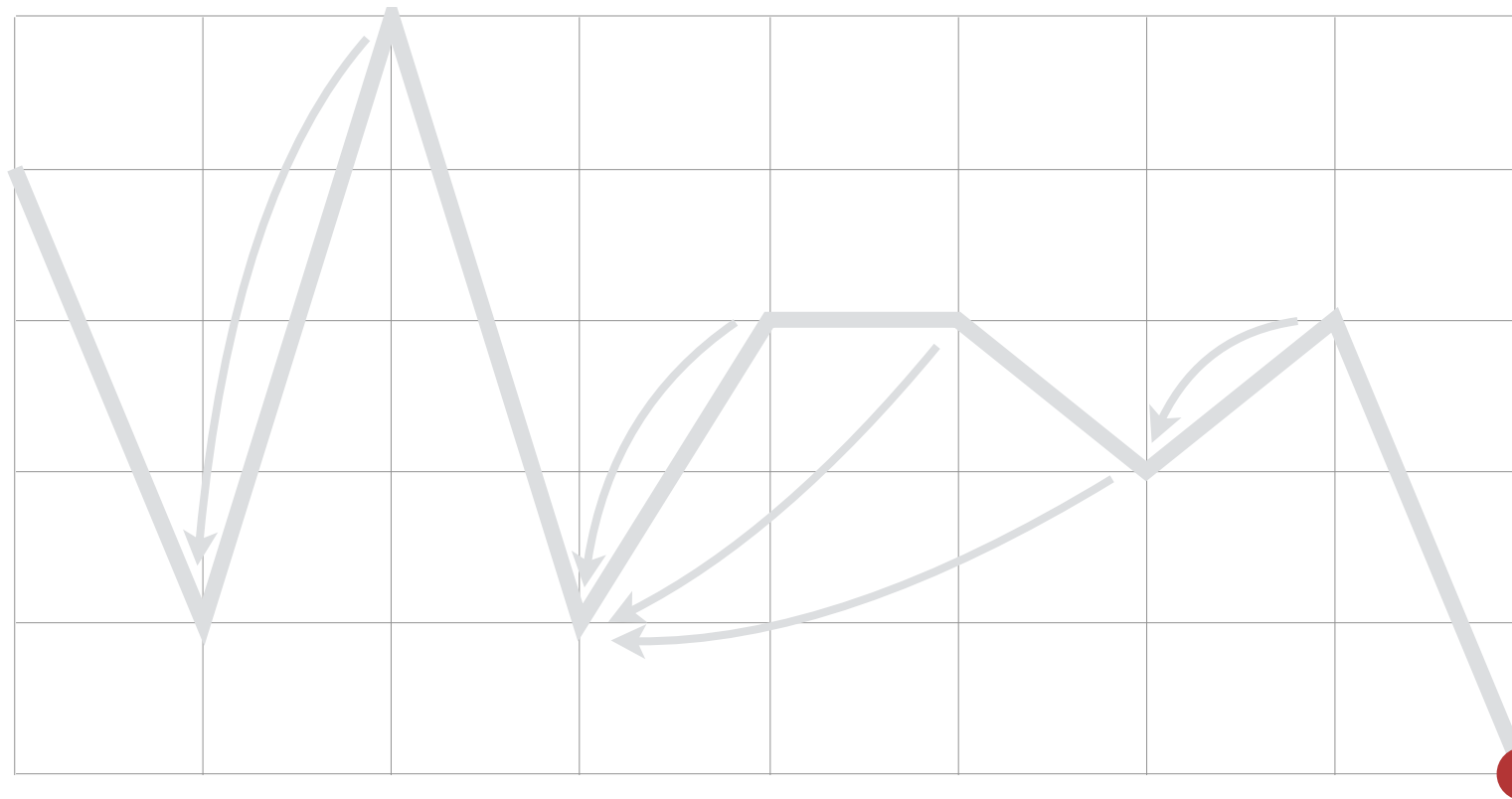
Phase 2

Suffixe innerhalb der Gruppen sortieren



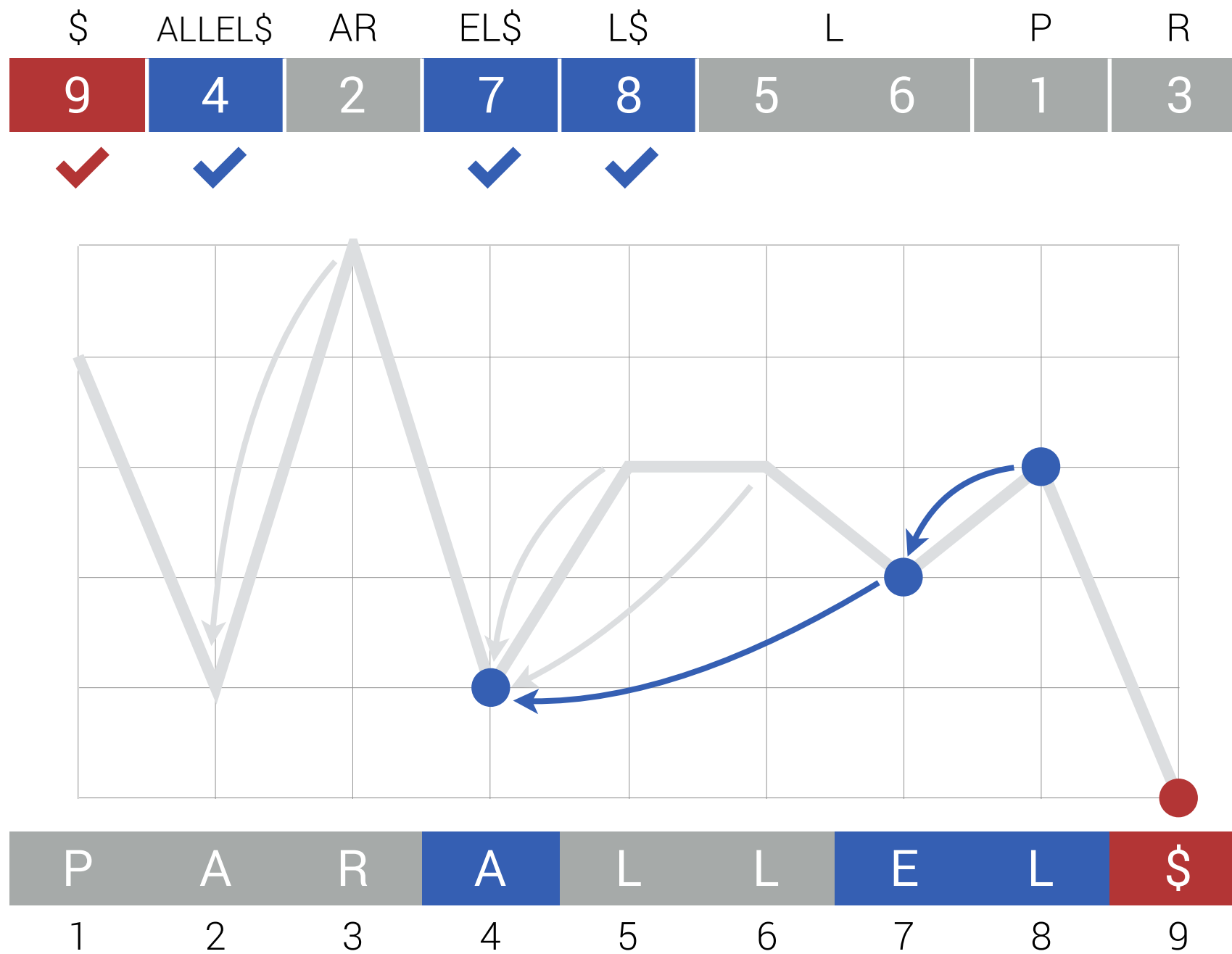
Phase 2

\$	ALLEL	AR	EL	L			P	R
9	4	2	7	5	6	8	1	3

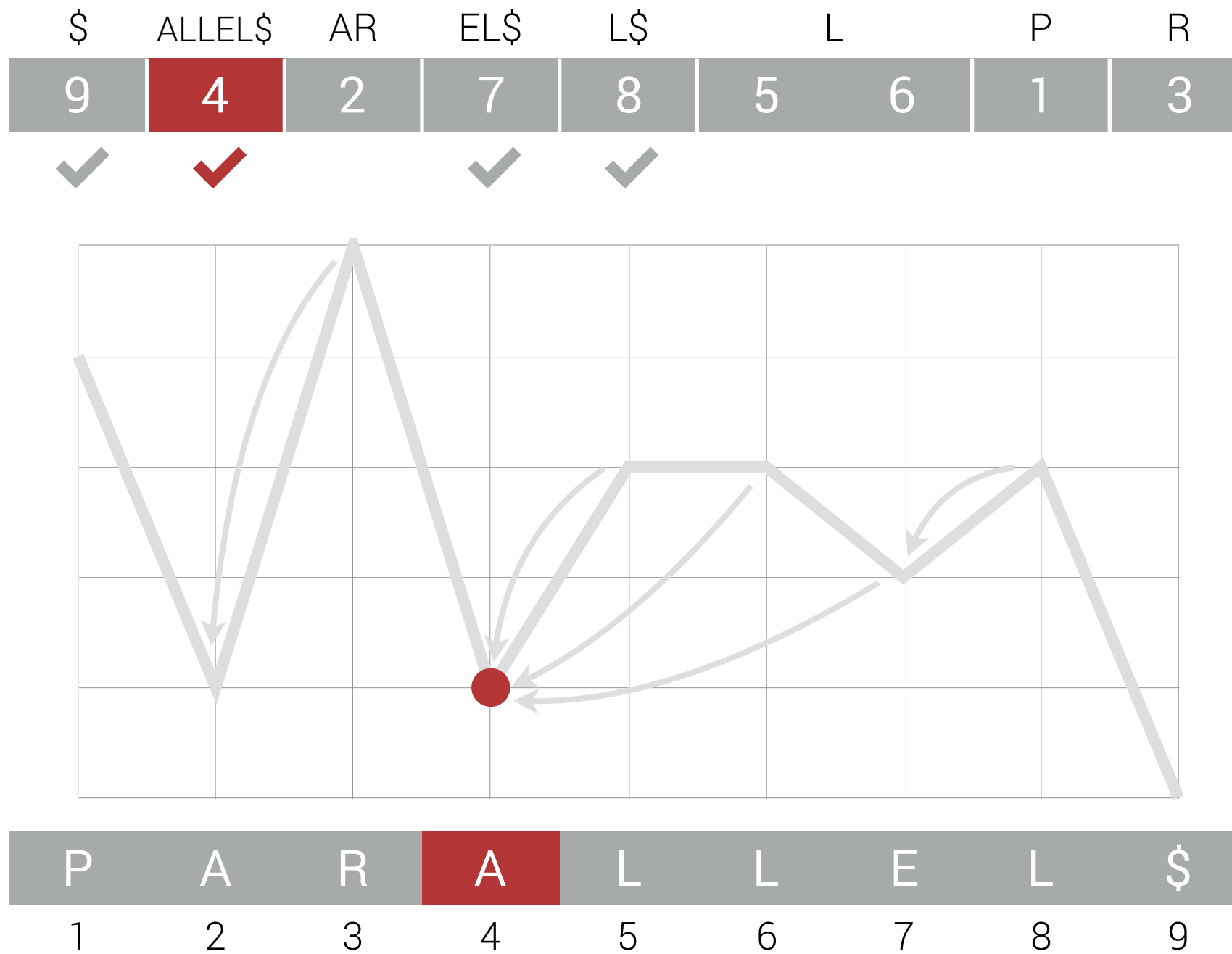


P	A	R	A	L	L	E	L	\$
1	2	3	4	5	6	7	8	9

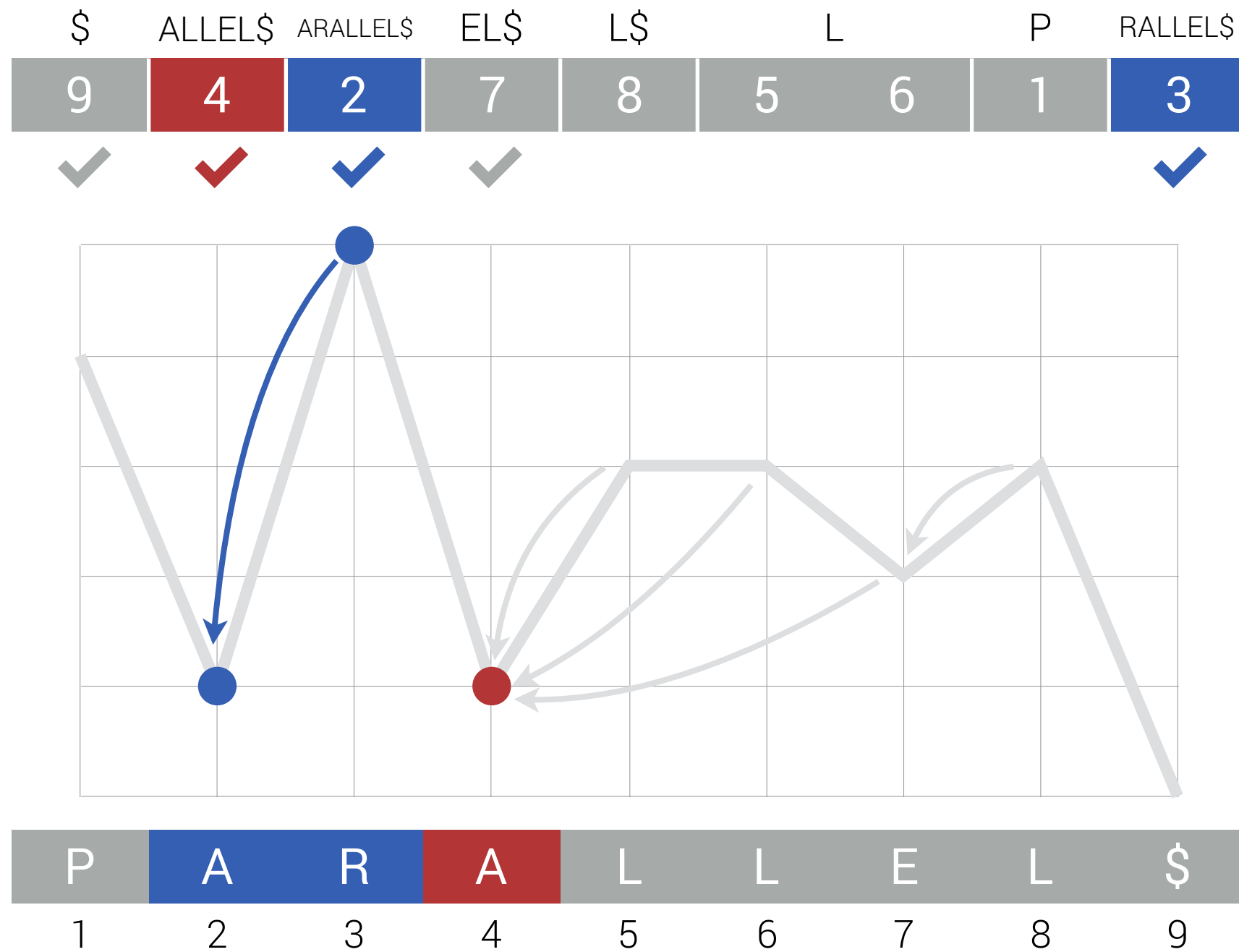
Phase 2



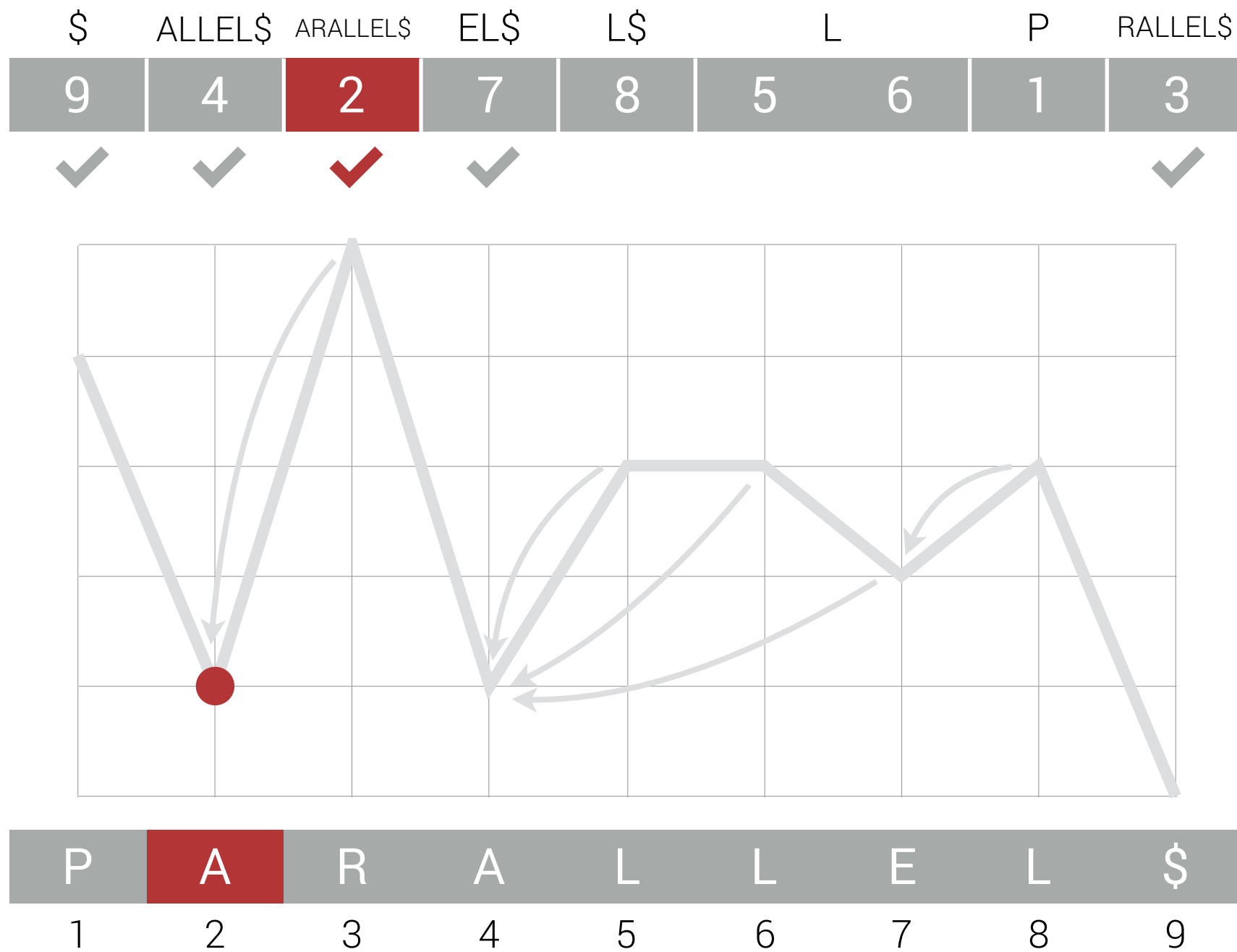
Phase 2



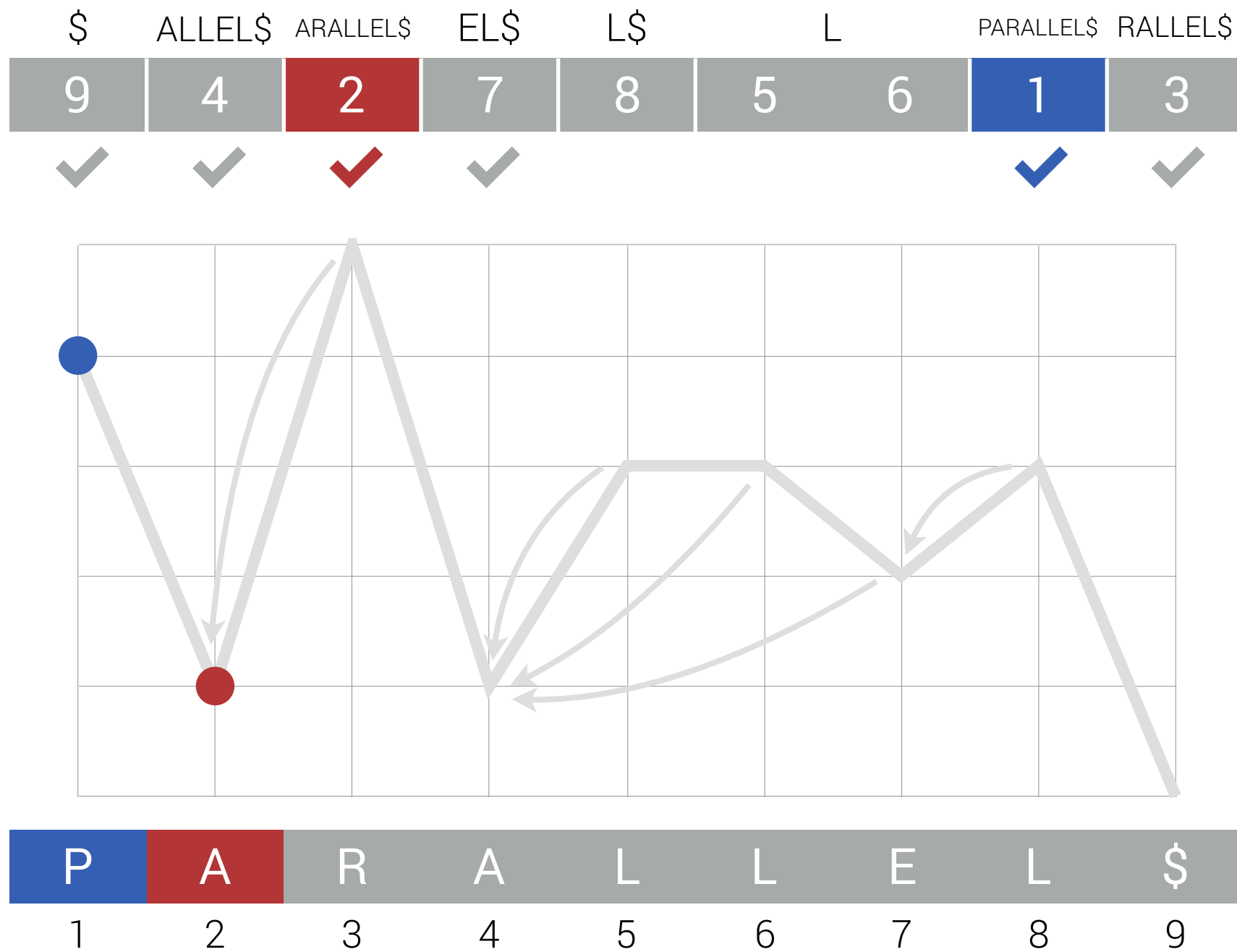
Phase 2



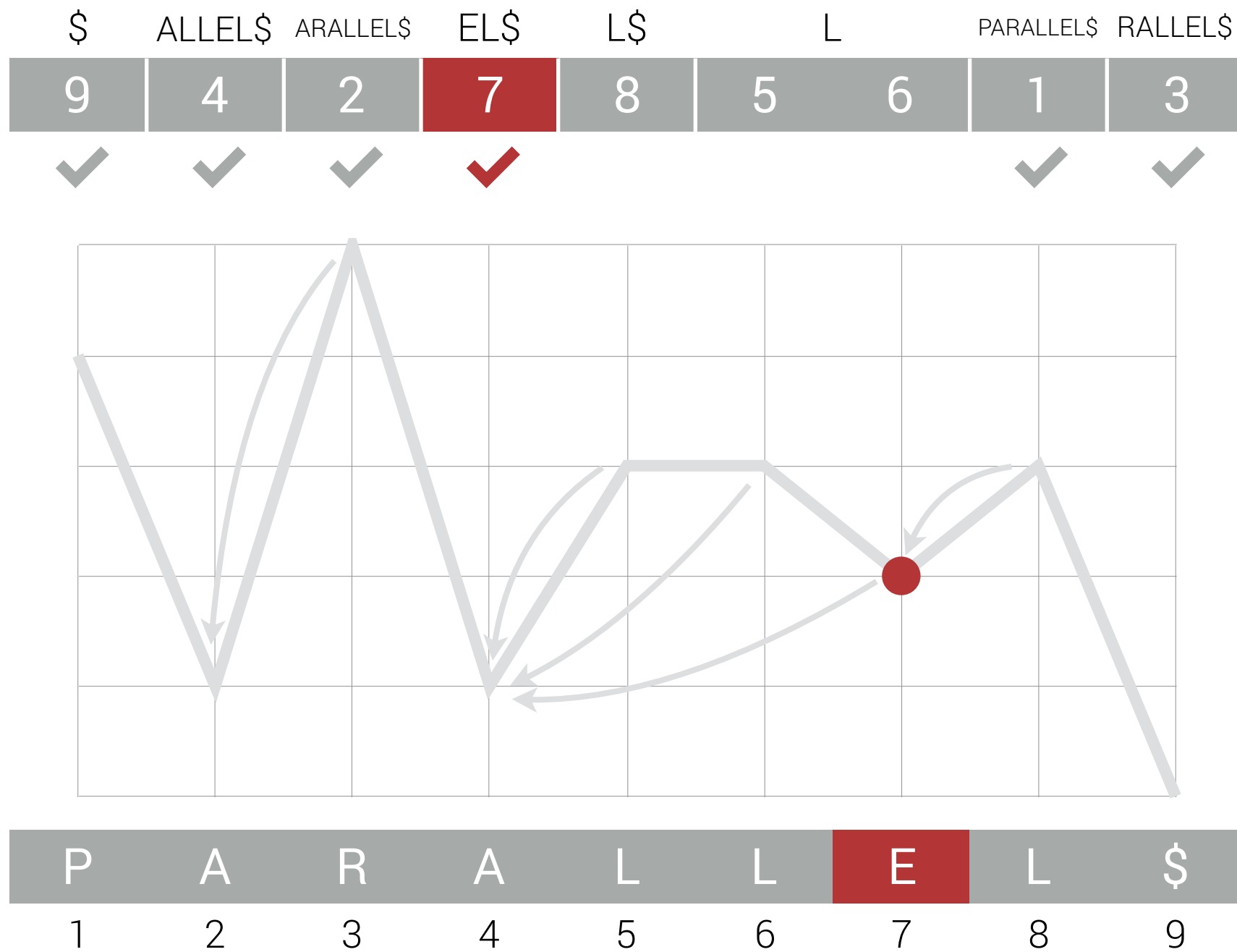
Phase 2



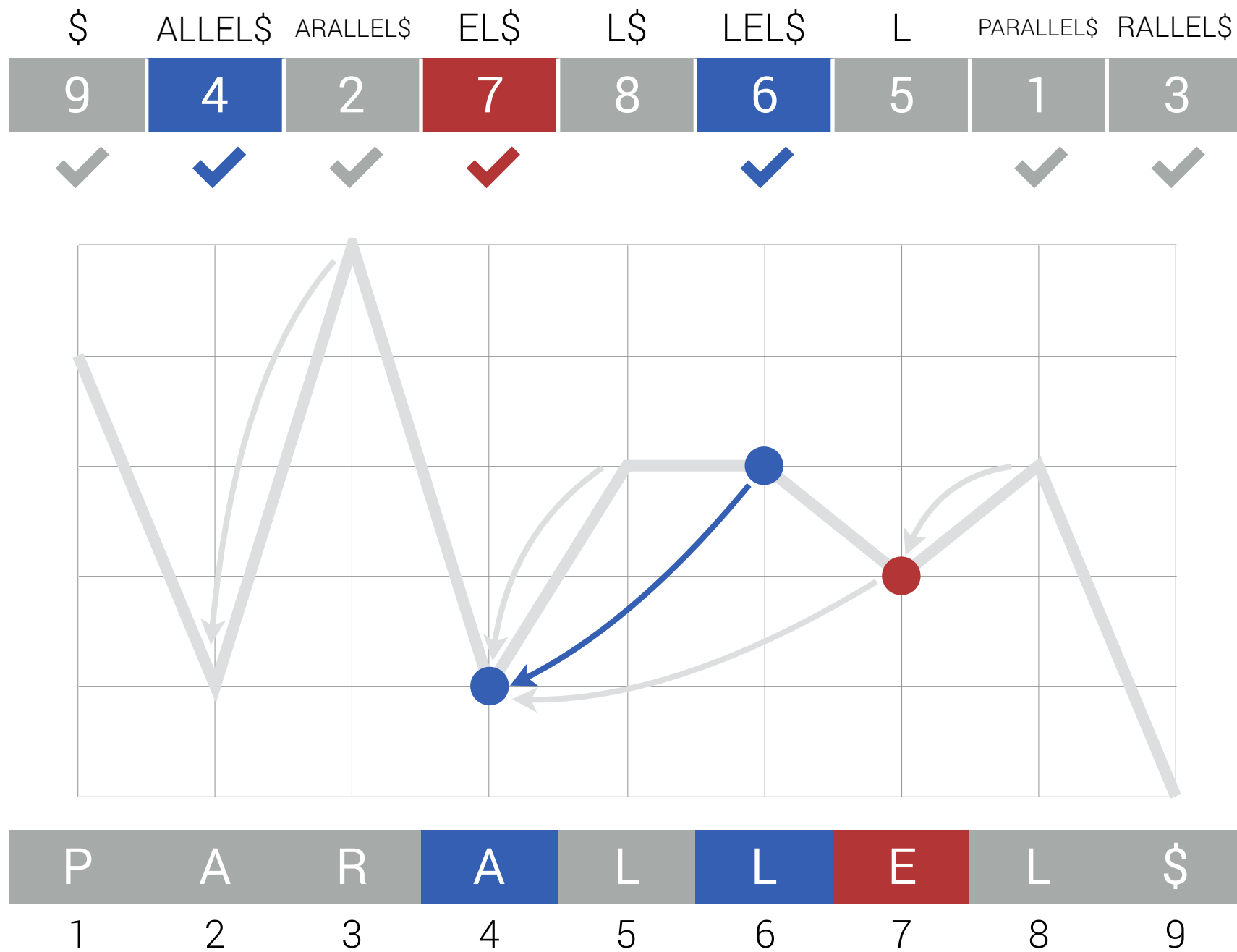
Phase 2



Phase 2



Phase 2



Phase 2

SA =

\$	ALLEL\$	ARALLEL\$	EL\$	L\$	LEL\$	LLEL\$	PARALLEL\$	RALLEL\$
9	4	2	7	8	6	5	1	3

Performance

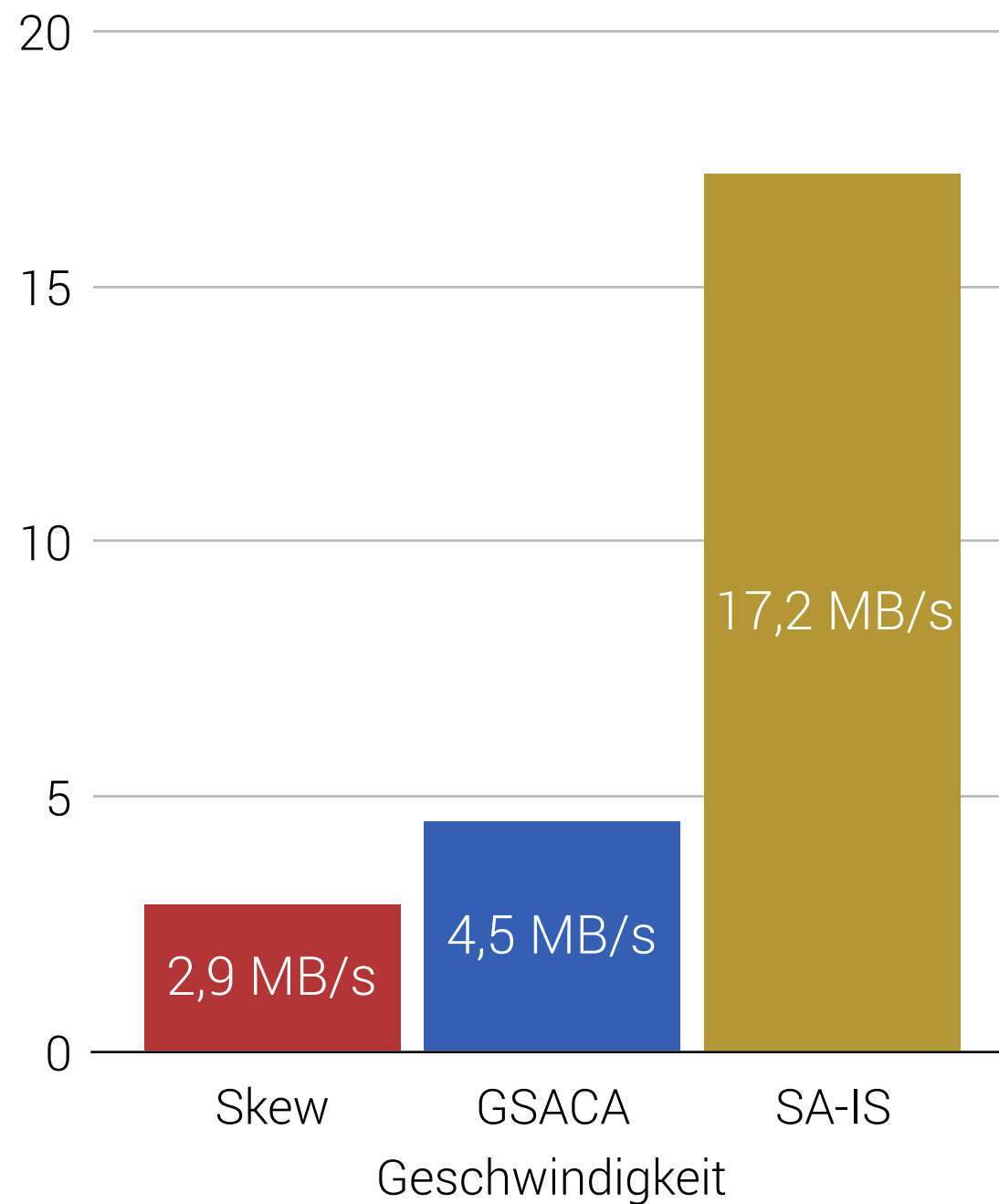
Linearzeit Ansätze

	Skew	SA-IS	GSACA
Art	rekursiv	rekursiv	iterativ
Zeit	$O(n)$	$O(n)$	$O(n)$
Speicher	$O(\log n) + \max 24n$	$O(\log n) + \max 2n$	$O(1) + ?$

Linearzeit Ansätze

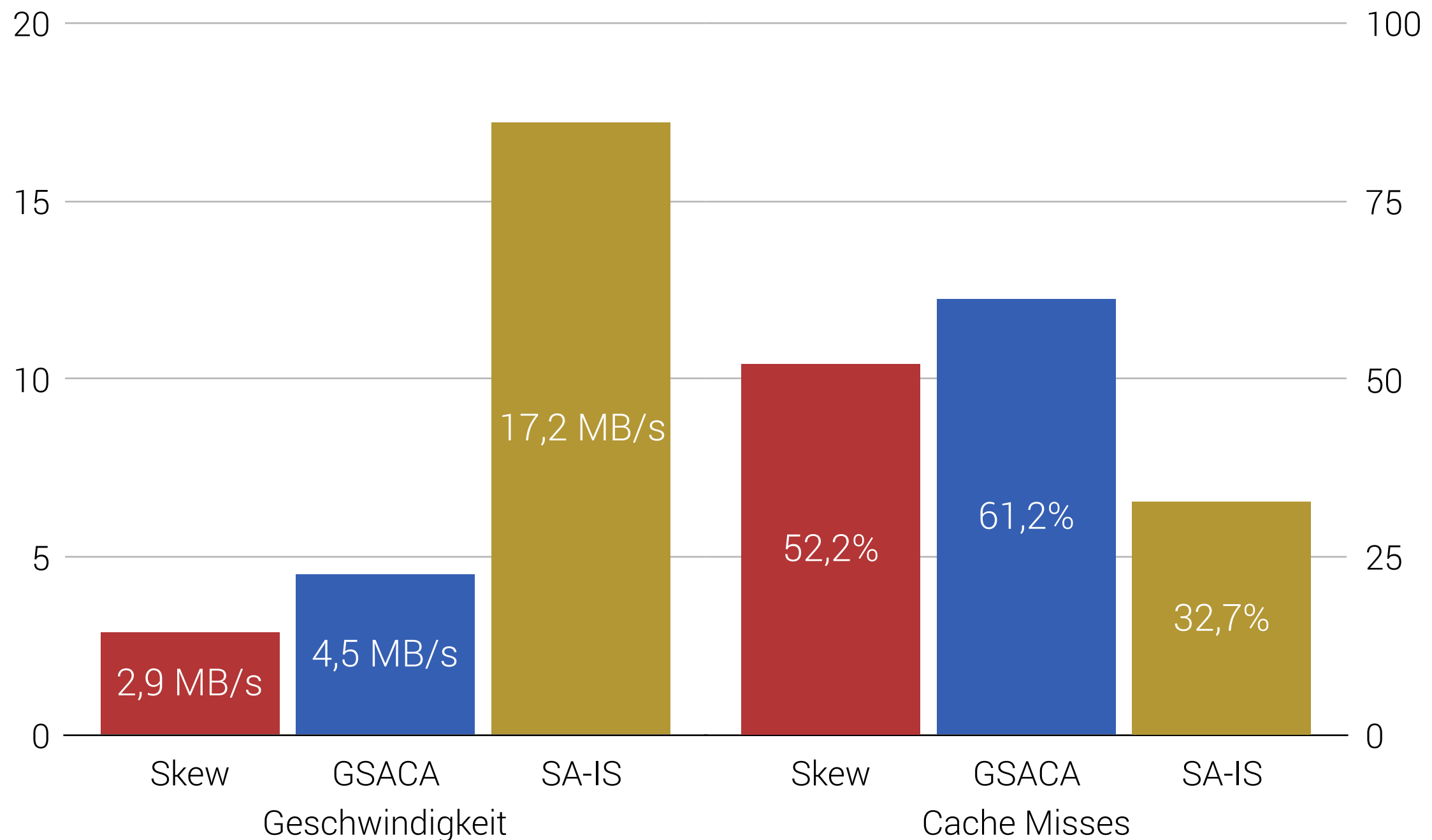
	Skew	SA-IS	GSACA
Art	rekursiv	rekursiv	iterativ
Zeit	$O(n)$	$O(n)$	$O(n)$
Speicher	$O(\log n) + \max 24n$	$O(\log n) + \max 2n$	$O(1) + 12n$

GSACA im Vergleich



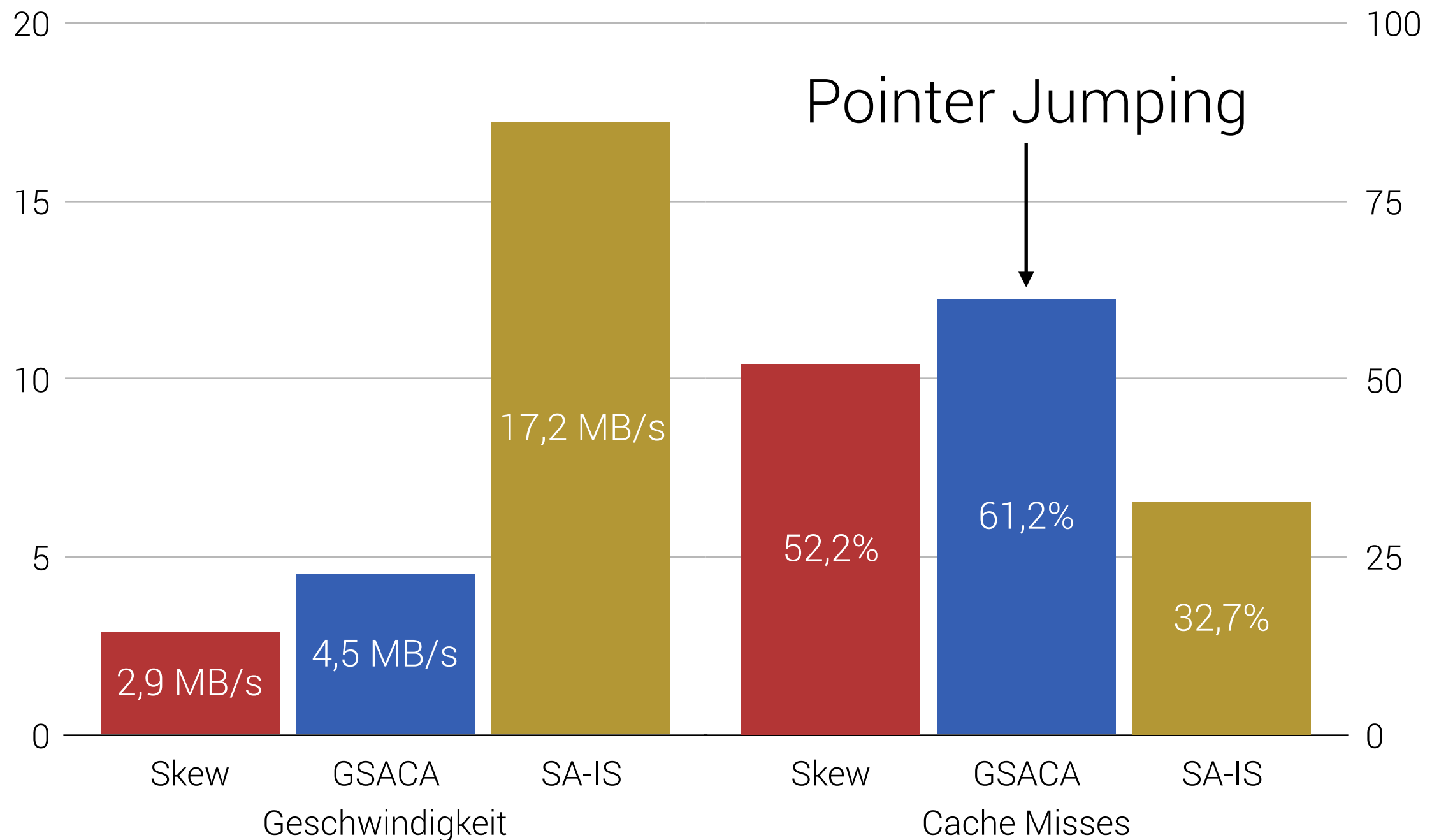
Testdaten: Silesia Corpus

GSACA im Vergleich



Testdaten: Silesia Corpus

GSACA im Vergleich

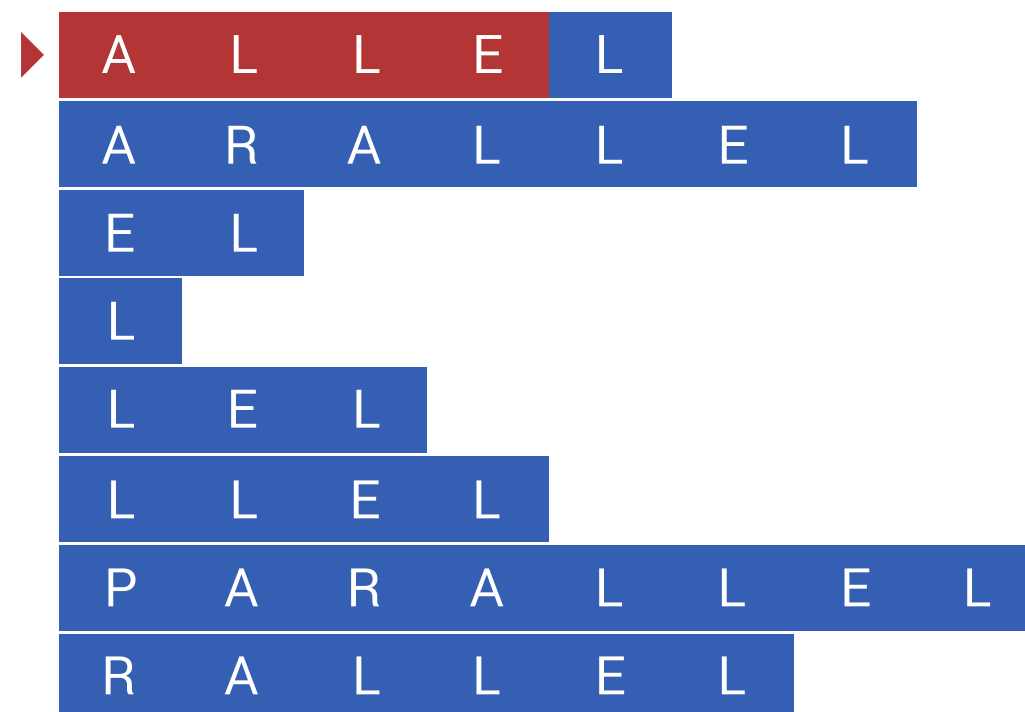


Testdaten: Silesia Corpus

Rückblick

Einsatzgebiete

Substringsuche



LZ77
Kompression

GSACA

Eingabe



Phase 1 Sortierte Folge von Gruppen berechnen $O(n)$



Phase 2 Suffixe innerhalb der Gruppen sortieren $O(n)$

Performance

Performance

Noch nicht praxistauglich.

Performance

~~Noch nicht praxistauglich.~~

Performance

~~Noch nicht praxistauglich.~~

Neuartiges Konzept mit vielen spannenden noch zu lösenden Problemen...

Danke!

Ergänzungen

Phase 1

- 1: order all suffixes of S into groups according to their first character:
 Let S_i and S_j be two suffixes. Then, $\text{group}(i) = \text{group}(j) \Leftrightarrow S[i] = S[j]$.
- 2: order the suffix groups: Let \mathcal{G}_1 be a suffix group with group context character u , \mathcal{G}_2 be a suffix group with group context character v . Then, $\mathcal{G}_1 < \mathcal{G}_2$ if $u < v$.
- 3: **for each** group \mathcal{G} in descending group order **do**
- 4: **for each** $i \in \mathcal{G}$ **do**
- 5: $\text{prev}(i) \leftarrow \max(\{ j \in [1 \dots i] \mid \text{group}(j) < \text{group}(i) \} \cup \{0\})$
- 6: let \mathcal{P} be the set of previous suffixes from \mathcal{G} ,
 $\mathcal{P} := \{ j \in [1 \dots n] \mid \text{prev}(i) = j \text{ for any } i \in \mathcal{G} \}$.
- 7: split \mathcal{P} into k subsets $\mathcal{P}_1, \dots, \mathcal{P}_k$ such that a subset \mathcal{P}_l contains
 suffixes whose number of prev pointers from \mathcal{G} pointing to them
 is equal to l , i.e. $i \in \mathcal{P}_l \Leftrightarrow |\{ j \in \mathcal{G} \mid \text{prev}(j) = i \}| = l$.
- 8: **for** $l = k$ **down to** 1 **do**
- 9: split \mathcal{P}_l into m subsets $\mathcal{P}_{l_1}, \dots, \mathcal{P}_{l_m}$ such that suffixes
 of same group are gathered in the same subset.
- 10: **for** $q = 1$ **up to** m **do**
- 11: remove suffixes of \mathcal{P}_{l_q} from their group and put them into a new
 group placed as immediate successor of their old group.

Phase 2

```

12:  $SA[1] \leftarrow n$ 
13: for  $i = 1$  up to  $n$  do
14:    $j \leftarrow SA[i] - 1$ 
15:   while  $j \neq 0$  do
16:     let  $sr$  be the number of suffixes placed in lower groups,
       i.e.  $sr := |\{ s \in [1 \dots n] \mid \text{group}(s) < \text{group}(j) \}|$ .
17:     if  $SA[sr + 1] \neq \text{nil}$  then
18:       break
19:      $SA[sr + 1] \leftarrow j$ 
20:     remove  $j$  from its current group and put it in a new group
       placed as immediate predecessor of  $j$ 's old group.
21:    $j \leftarrow \text{prev}(j)$ 

```

Performance

Text Corpus		divsufsort	SA-IS	Ko-Aluru	Skew	GSACA
Silesia Dateien < 40 MB	speed	15,9 MB/s	17,2 MB/s	8,1 MB/s	2,9 MB/s	4,5 MB/s
	cache misses	26,5 %	32,7 %	24,2 %	52,0 %	61,2 %
Pizza & Chilli Dateien ~200 MB	speed	9,2 MB/s	8,1 MB/s	3,5 MB/s	1,1 MB/s	3,0 MB/s
	cache misses	49,5 %	74,8 %	55,2 %	86,1 %	79,0 %
Repetitive Dateien > 45 MB	speed	12,5 MB/s	14,2 MB/s	5,3 MB/s	1,7 MB/s	3,5 MB/s
	cache misses	41,9 %	68,6 %	49,7 %	78,0 %	76,9 %