ALG 08

Merge sort -- řazení sléváním

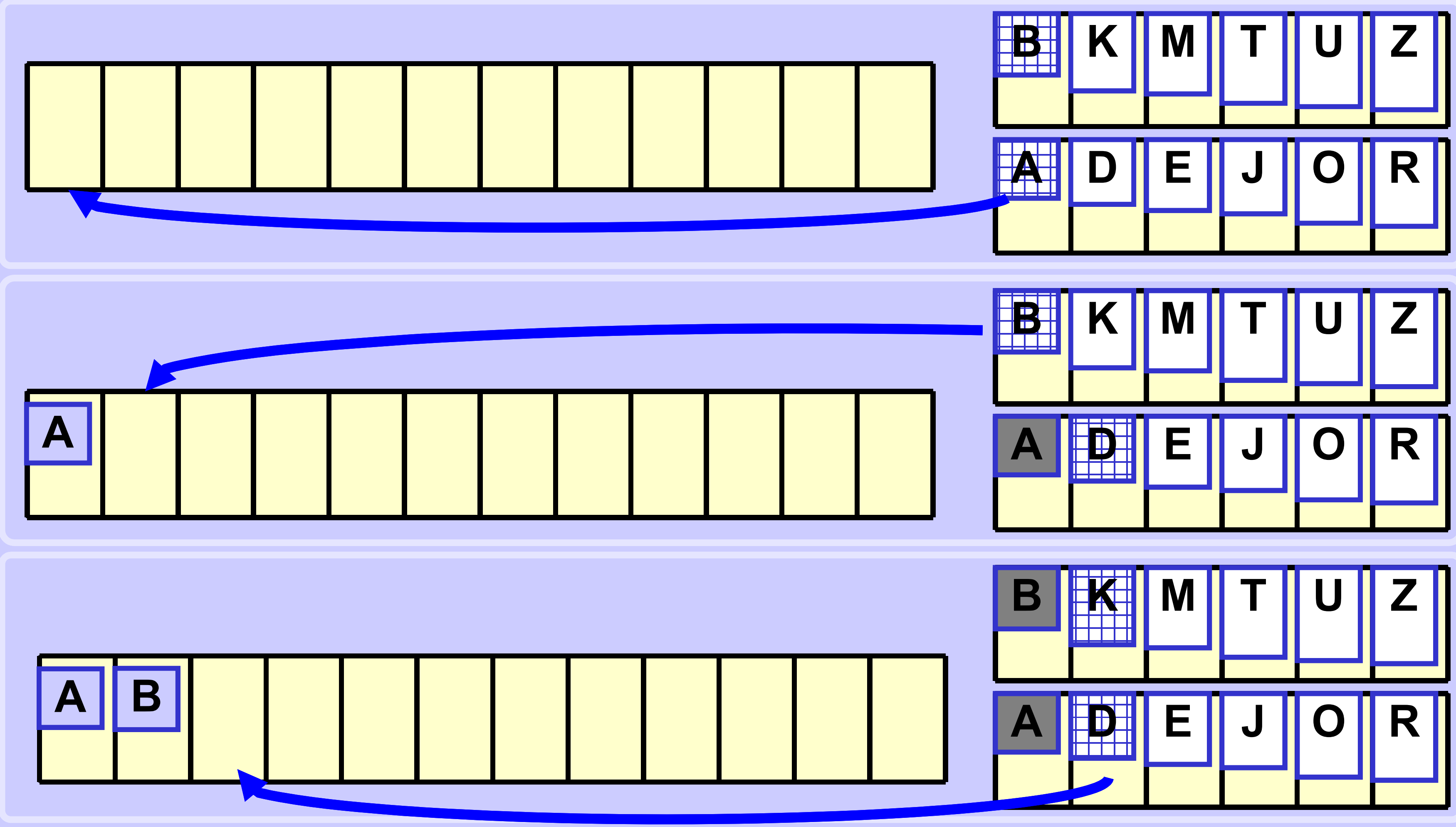
Heap Sort -- řazení binární haldou

Prioritní fronta implementovaná binární haldou

Merge sort

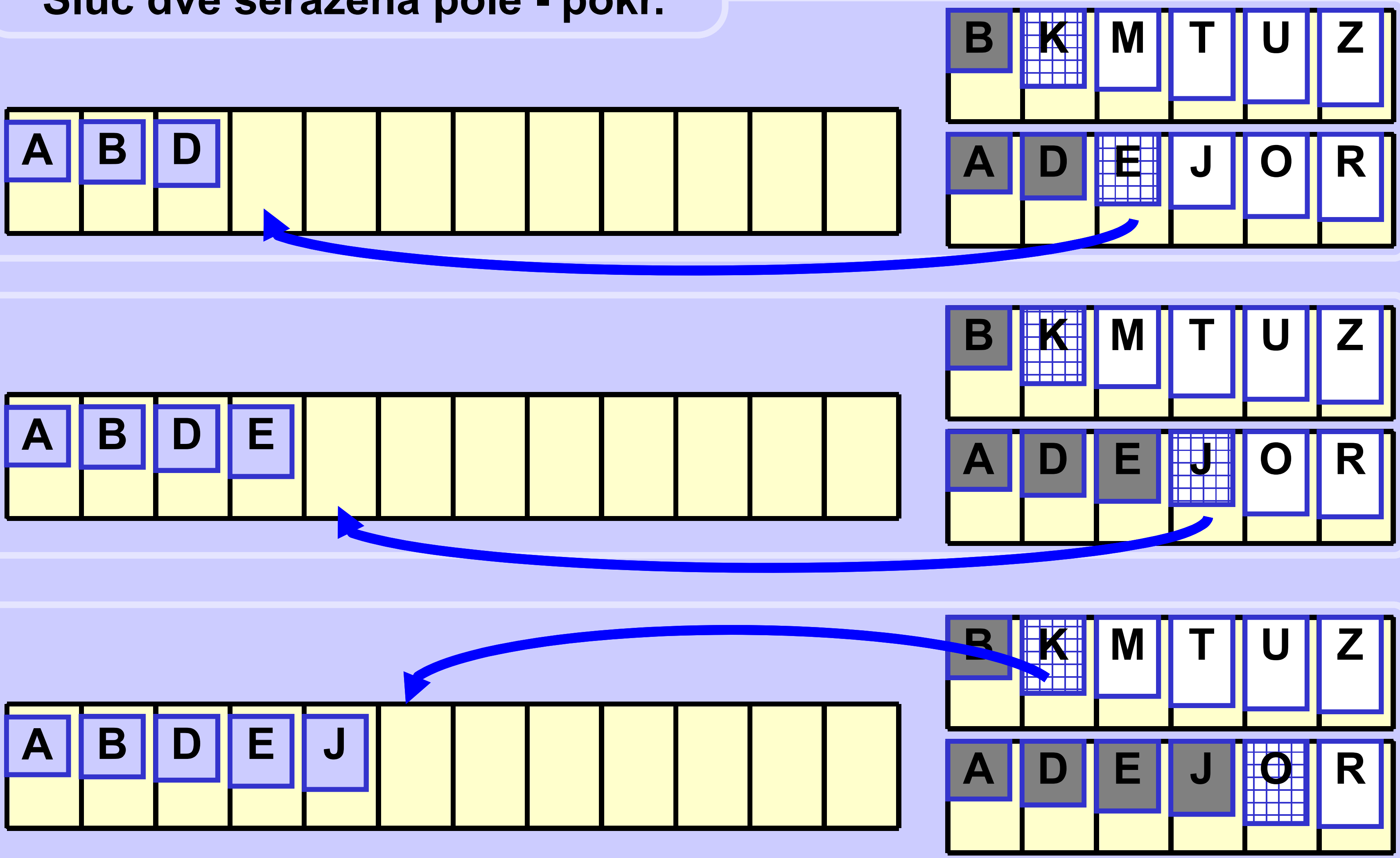
Sluč (slij?) dvě seřazená pole

Porovnávané prvky



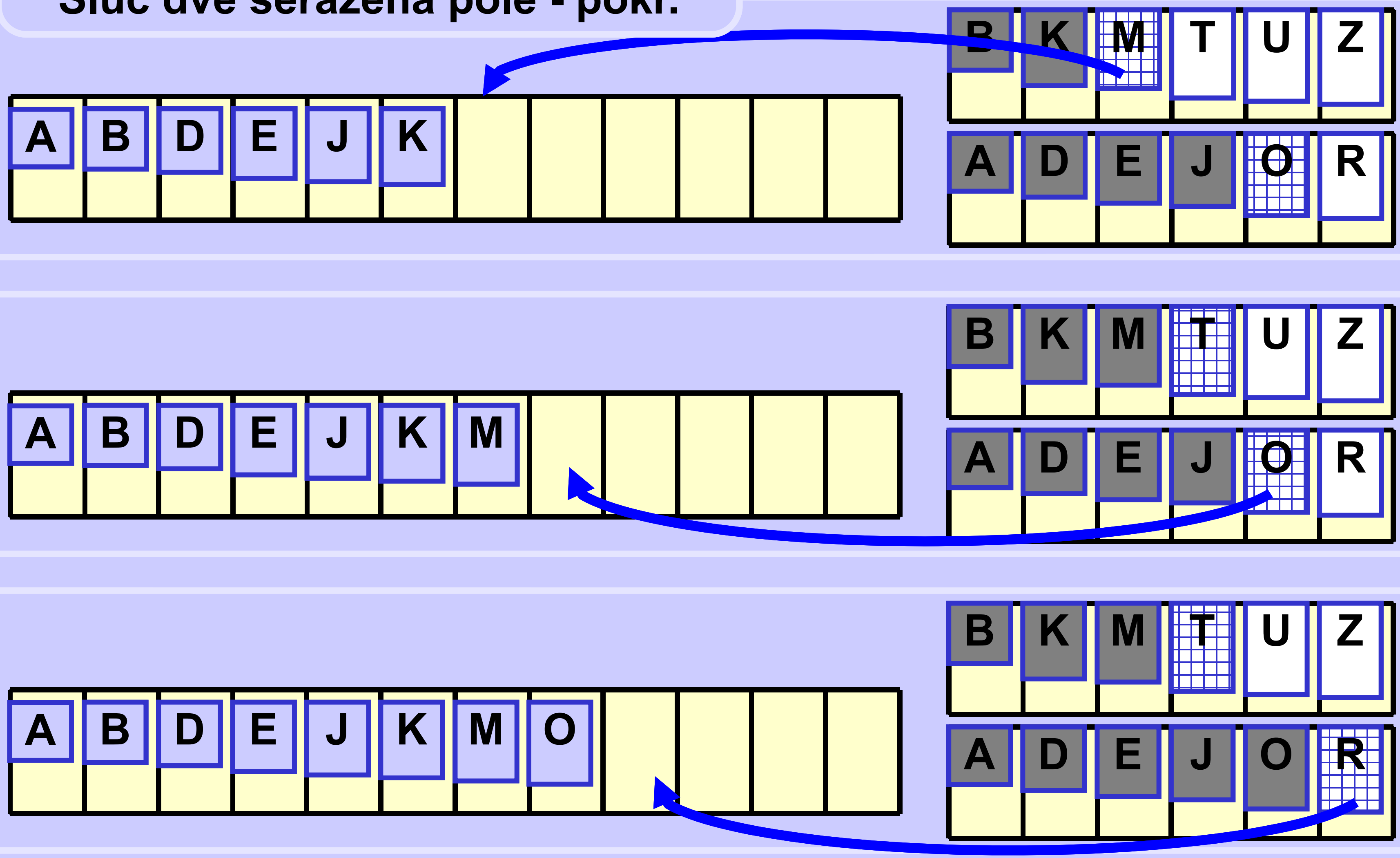
Merge sort

Sluč dvě seřazená pole - pokr.



Merge sort

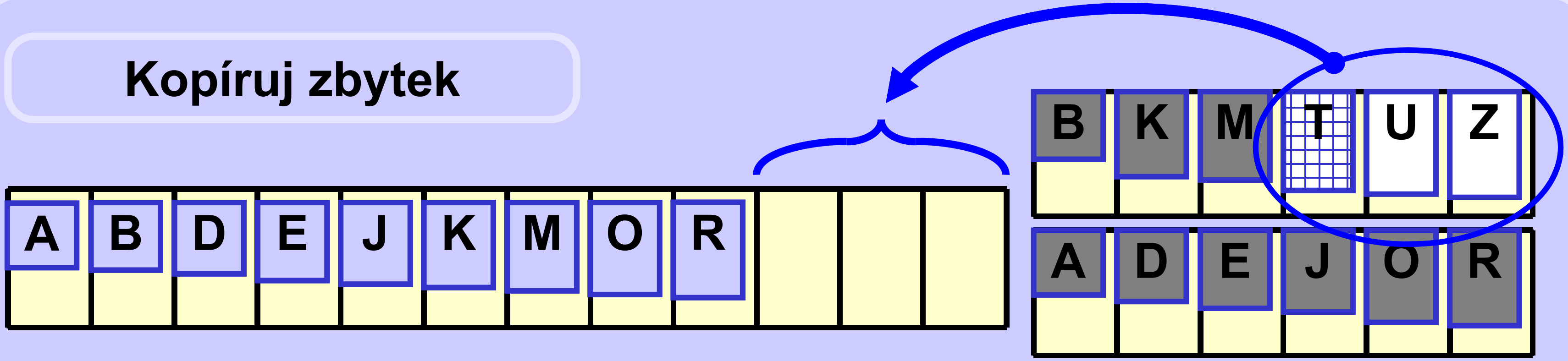
Sluč dvě seřazená pole - pokr.



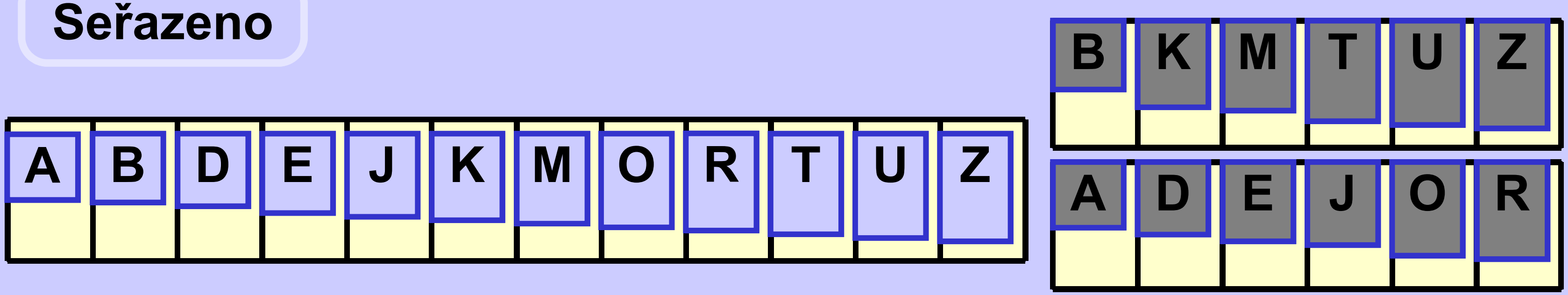
Merge sort

Sluč dvě seřazená pole - pokr.

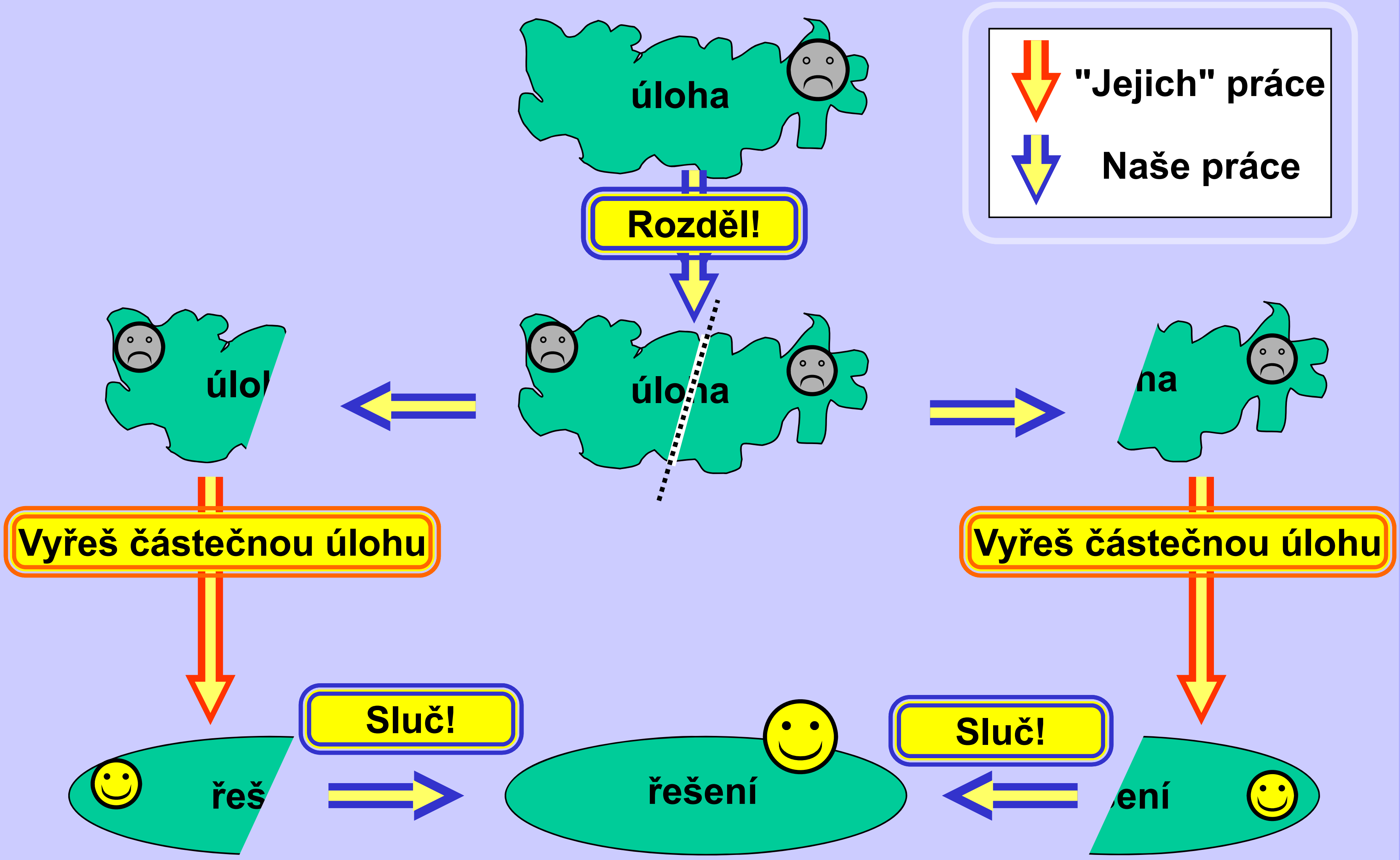
Kopíruj zbytek



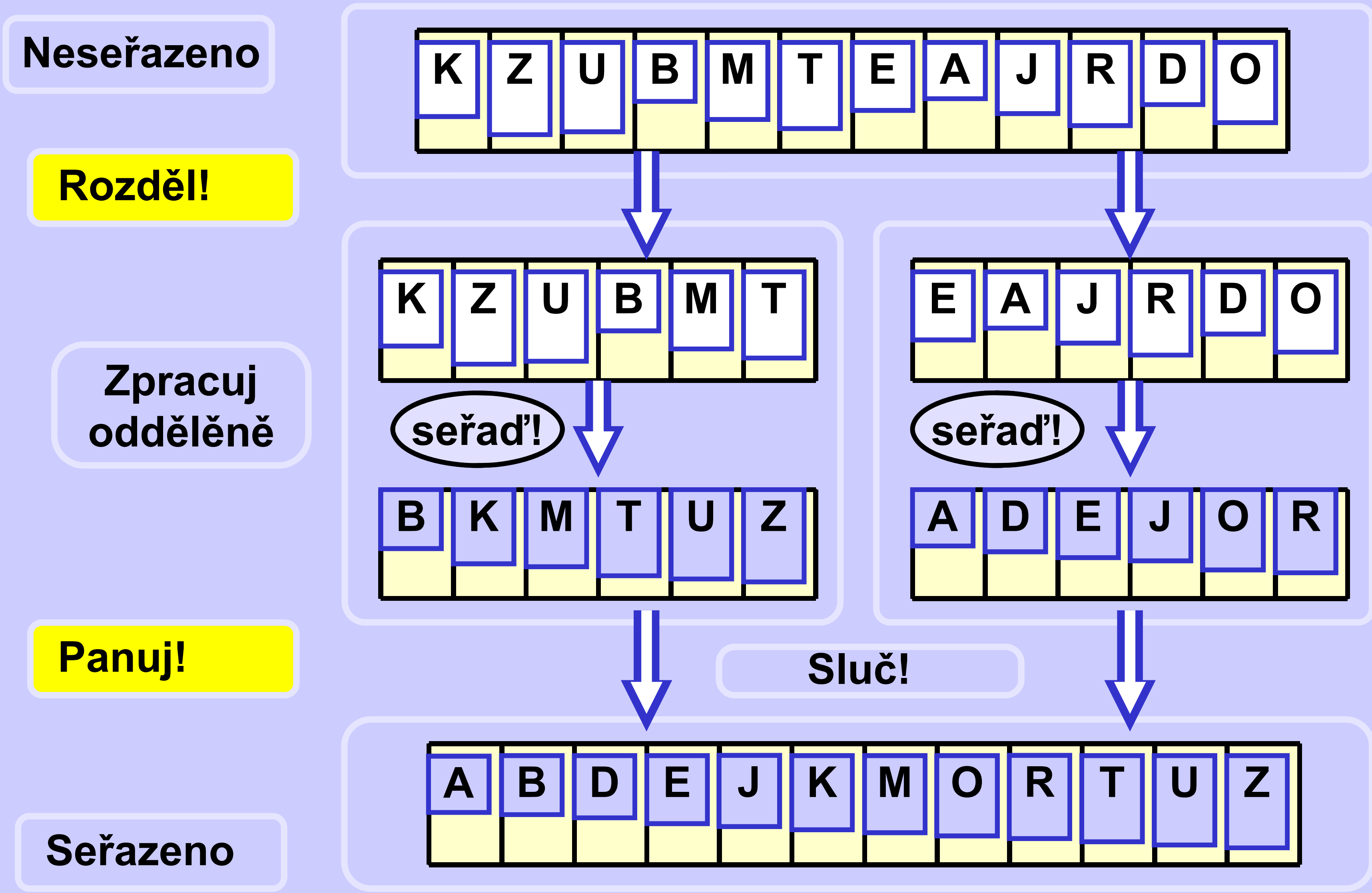
Seřazeno



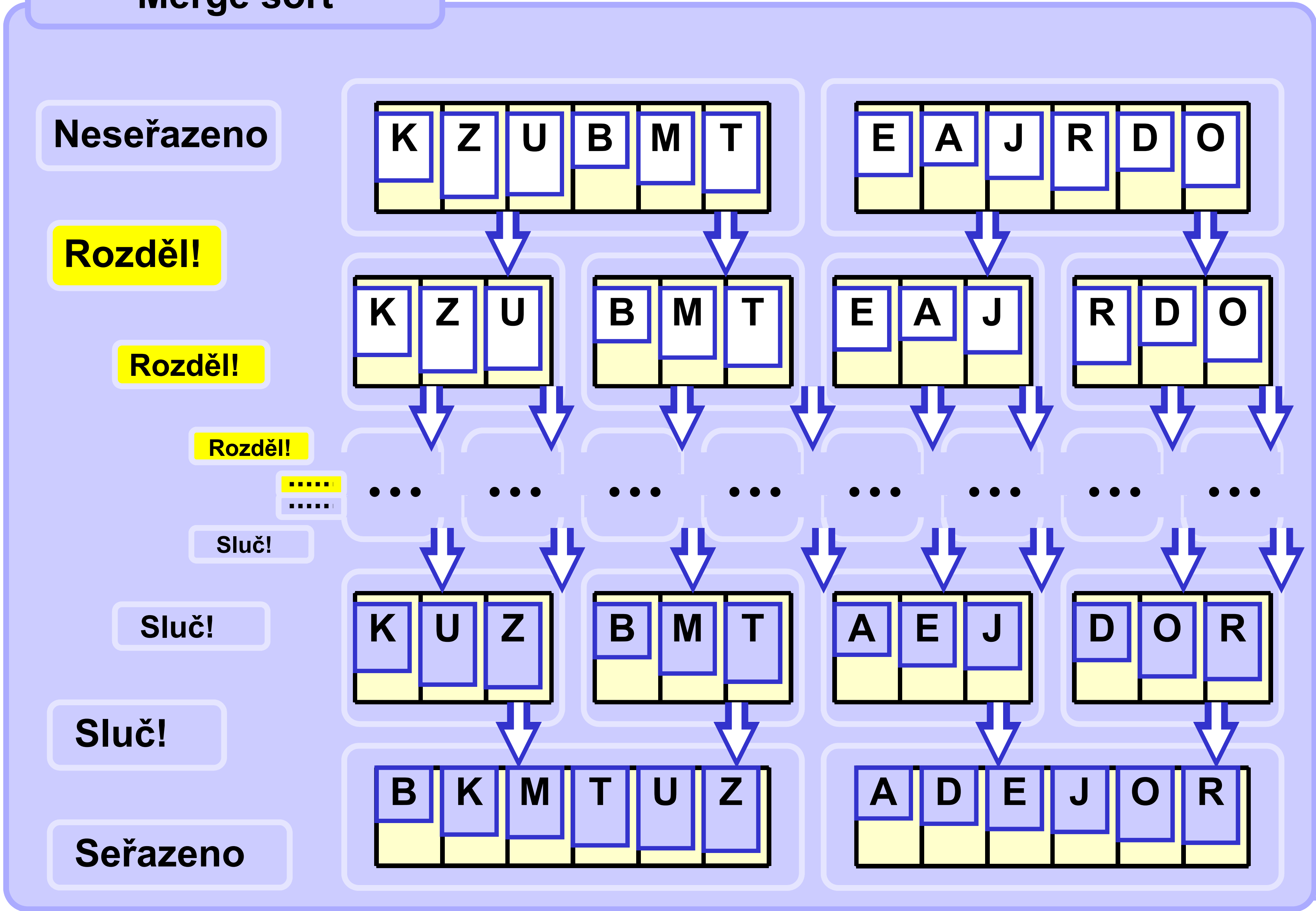
Rozděl a panuj! Divide and conquer! Divide et impera!



Merge sort


Neseřazeno
K Z U B M T E A J R D O
Rozděl!
Zpracuj odděleně
K Z U B M T
E A J R D O
seřaď!
seřaď!
B K M T U Z
A D E J O R
Panuj!
Sluč!
A B D E J K M O R T U Z
Seřazeno

Merge sort



Neseřazeno
Rozděl!
Rozděl!
Rozděl!
Sluč!
Sluč!
Sluč!
Seřazeno
K Z U B M T
E A J R D O
K Z U
B M T
E A J
R D O
K U Z
B M T
A E J
D O R
B K M T U Z
A D E J O R

Merge sort

```
void merge( int [] in, int [] out, int low, int high ){
  int half = (low+high)/2;
  int i1 = low;
  int i2 = half+1;
  int j = low;

                                    // compare and merge
  while( (i1 <= half) && (i2 <= high) ){
    if( in[i1] <= in[i2] ){ out[j] = in[i1]; i1++; }
    else { out[j] = in[i2]; i2++; }
    j++;
  }
                                   // copy the rest
  while( i1 <= half ){ out[j] = in[i1]; i1++; j++; }
  while( i2 <= high ){ out[j] = in[i2]; i2++; j++; }
}
```

Merge sort

```
void mergeSort( int [] a, int [] aux,
                                   int low, int high ){
  int half = (low+high)/2;
  if( low >= high ) return;             // too small!

                                        // sort
  mergeSort( a, aux, low, half );       // left half
  mergeSort( a, aux, half+1, high );    // right half
  merge( a, aux, low, high );           // merge halves

  // (*) put result back to a -- clumsy method!
  for( int i = low; i <= high; i++ ) a[i] = aux[i];

  // (*) better solution: swap references
  //to a and aux in even depths of recursion
}
```

Merge sort - optimalizované využití pomocného pole

```
void mergeSort( int [] a ) {
  int [] aux = Arrays.copyOf( a, a.length );
  mergeSort(a, aux, 0, a.length-1, 0 );
}

void mergeSort( int [] a, int [] aux,
                int low, int high, int depth ){
  int half = (low+high)/2;
  if( low >= high ) return;

  mergeSort( a, aux, low, half, depth+1 );
  mergeSort( a, aux, half+1, high, depth+1 );

                //  note the exchange of a and aux
  if( d % 2 == 0 ) merge( aux, a, low, high );
  else             merge( a, aux, low, high );
}
```

Merge sort

Asymptotická složitost

Rozděl! .... $\log_2(n)$ krát $\Rightarrow$

$\Rightarrow$ Sluč! .... $\log_2(n)$ krát

Rozděl! .... $\Theta(1)$ operací

Sluč! .... $\Theta(n)$ operací

Celkem ...... $\Theta(n)\cdot\Theta(\log_2(n)) = \Theta(n\cdot\log_2(n))$ operací

Asymptotická složitost Merge sortu je $\Theta(n\cdot\log_2(n))$

Merge sort

Stabilita

Rozděl! .... **Nepohybuje prvky**

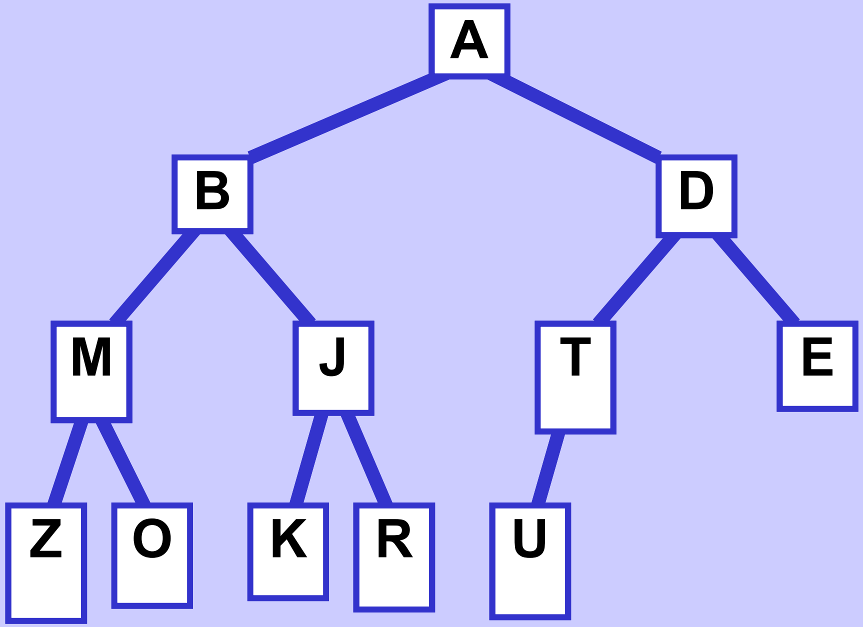
Sluč! ……. “ if (in[i1] <= in[i2]) { out[j] = in[i1]; …”

Zařaď nejprve levý prvek,
když slučuješ stejné hodnoty

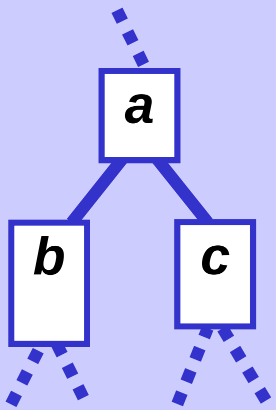
MergeSort je stabilní

Heap sort

Halda



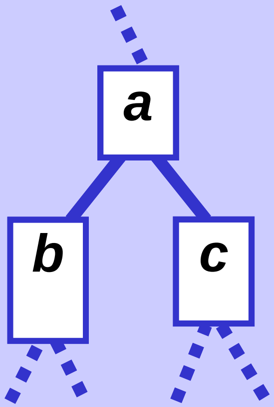
Pravidlo haldy



$$a \leq b \;\&\&\; a \leq c$$

Heap sort

Terminologie

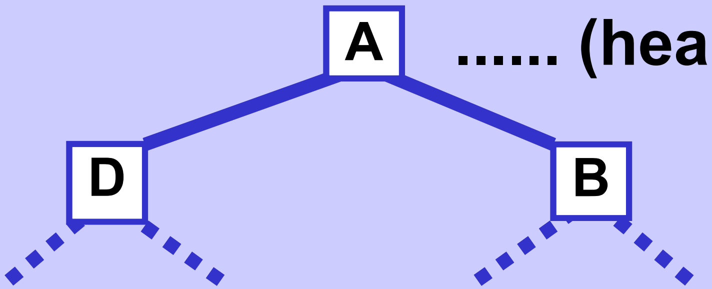


a ...... predecessor, parent of b c

...... předchůdce, rodič

b , c ...... successor, child of a

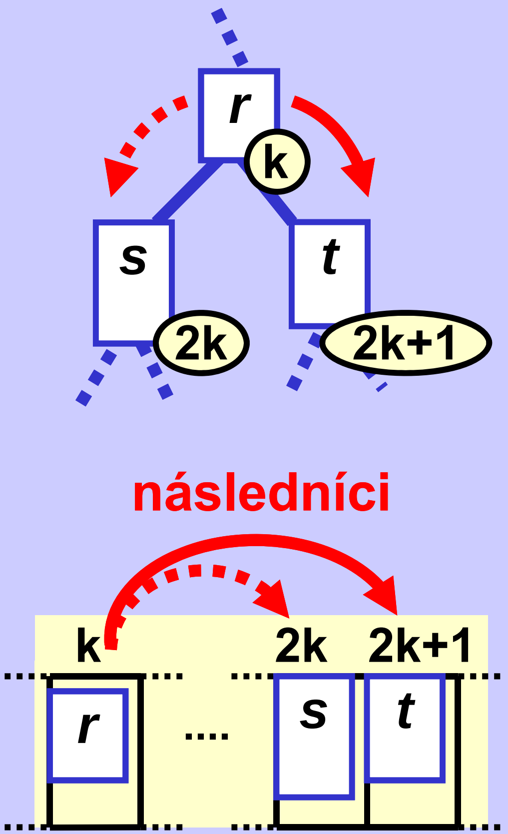
...... následník, potomek

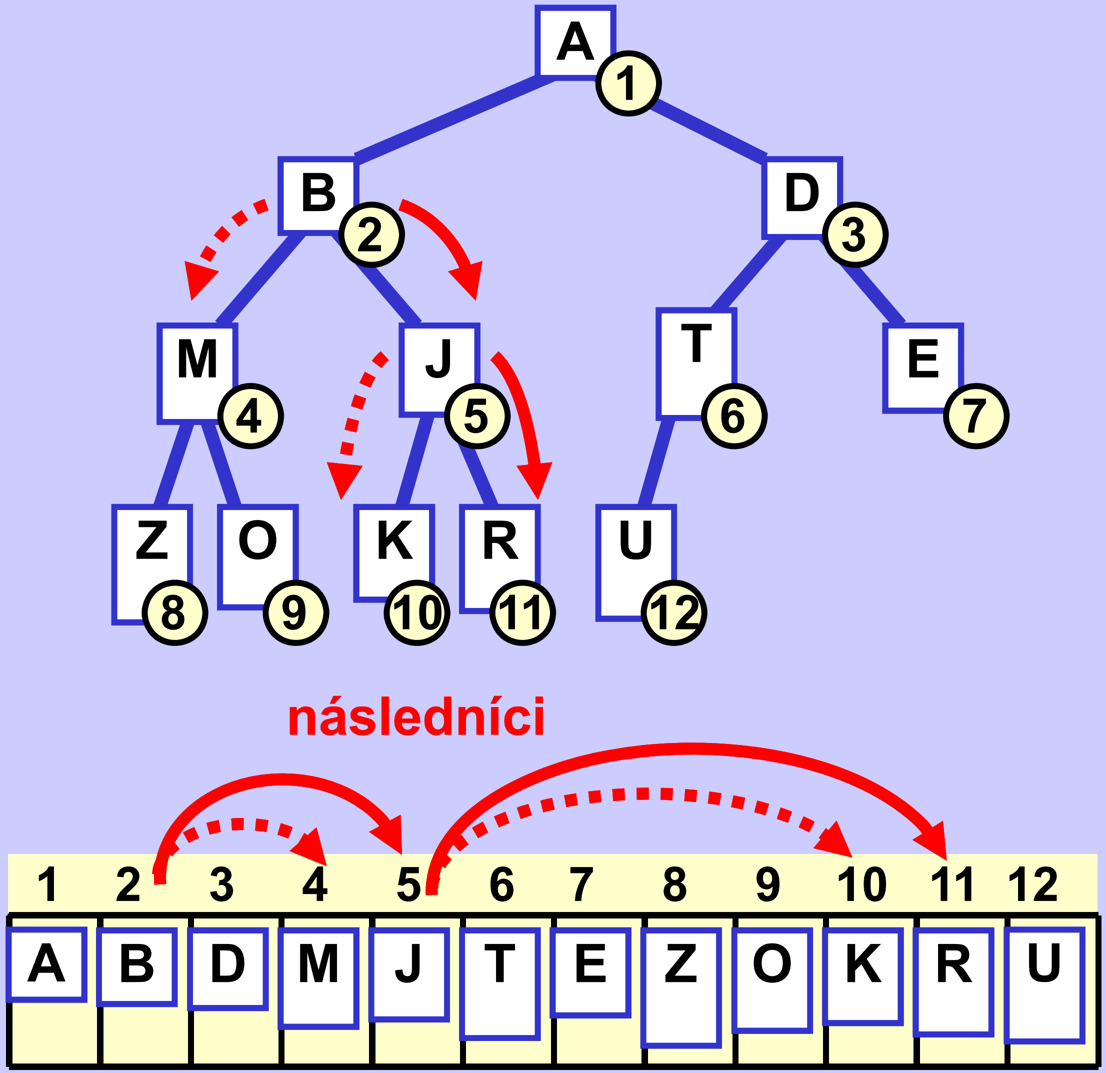


...... (heap) top ...... vrchol (haldy)

Heap sort

Halda uložená v poli

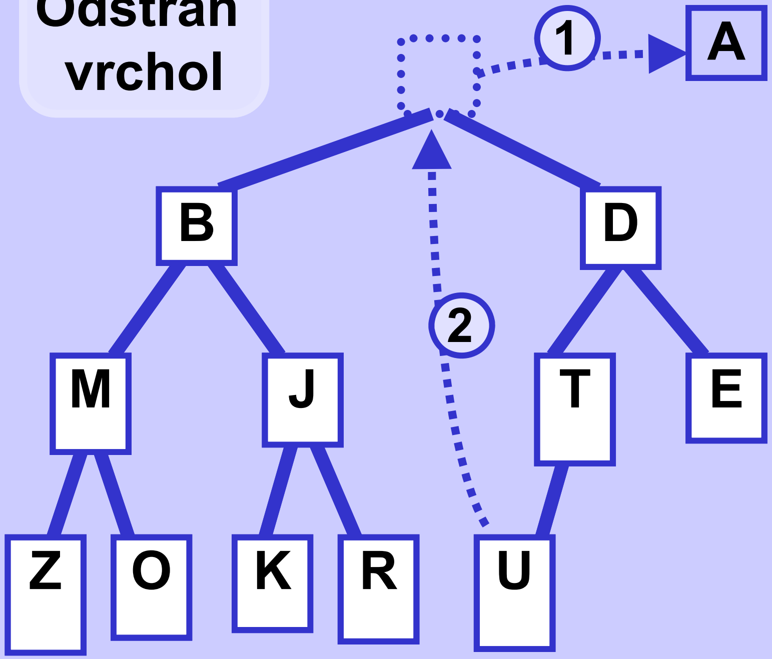




Oprava haldy

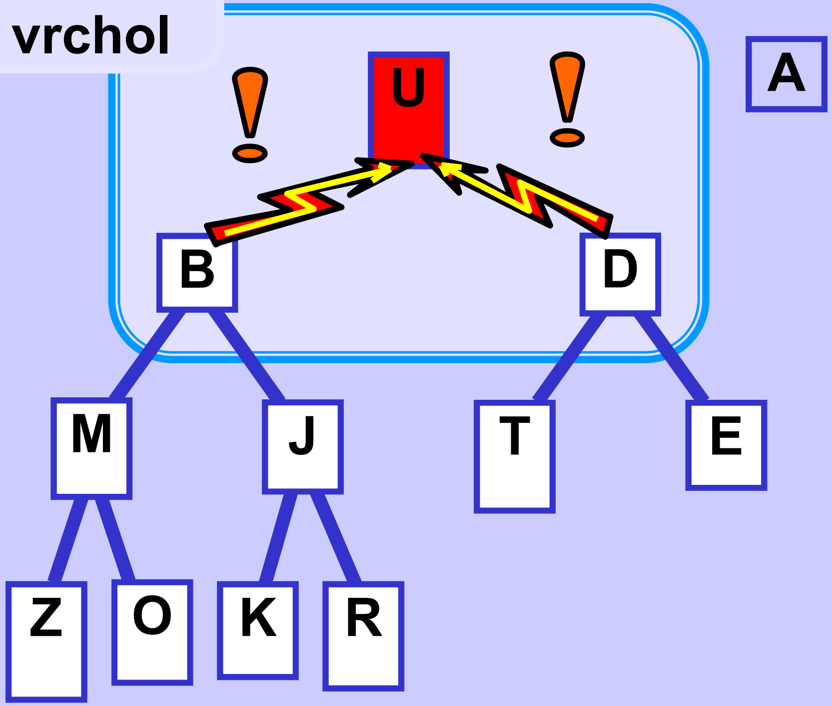
Vrchol odstraněn (1)

1 Odstraň vrchol



2 poslední ⟶ první

3 Vlož na vrchol



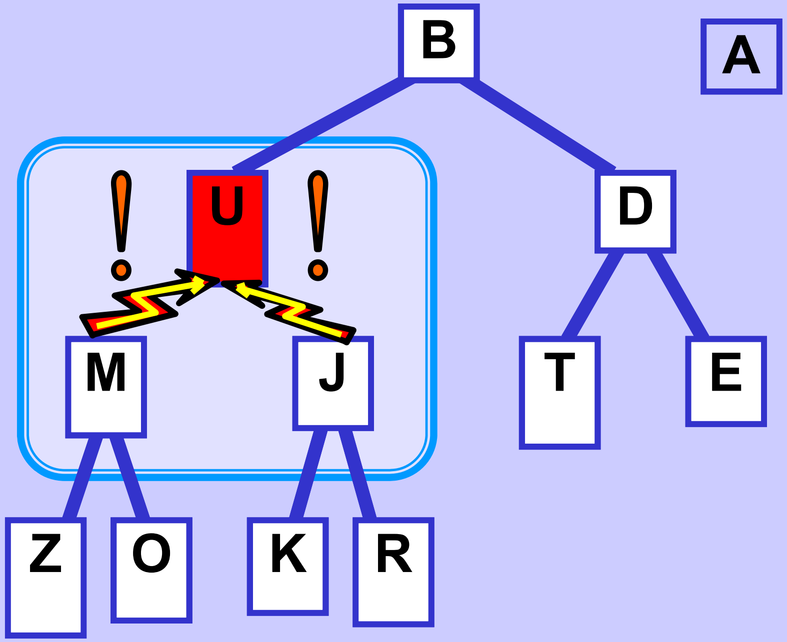
U > B, U > D, <u>B < D</u>

⇒ prohoď B ⟷ U

Oprava haldy

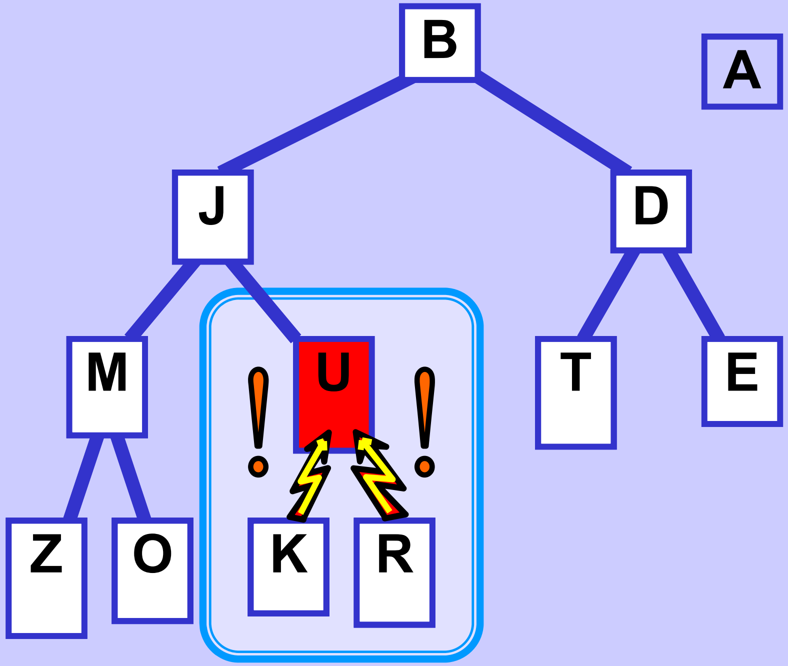
Vrchol odstraněn (2)

③ Vlož na vrchol - pokračování



U > M, U > J, J < M

⇒ prohoď J ⟷ U



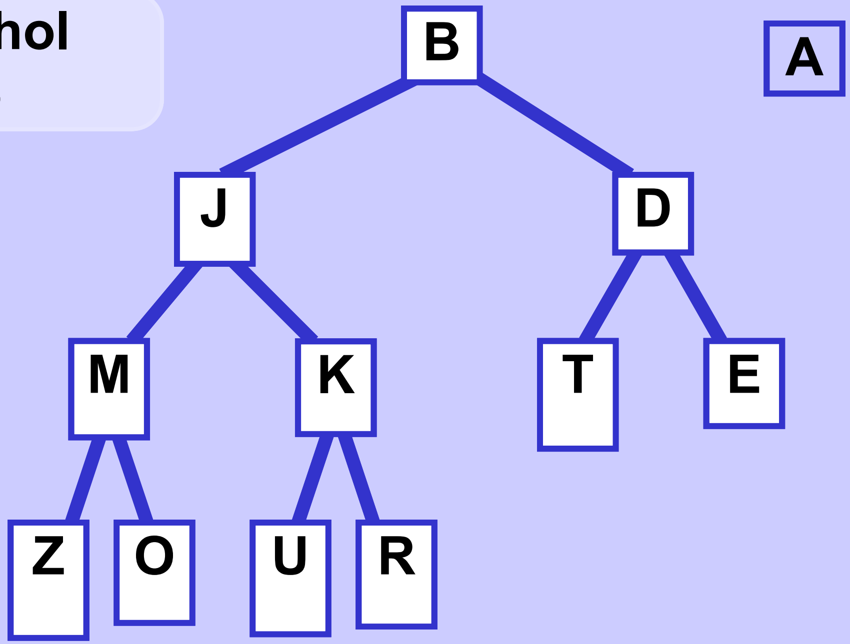
U > K, U > R, K < R

⇒ prohoď K ⟷ U

Oprava haldy

Vrchol odstraněn (3)

3 Vlož na vrchol - hotovo

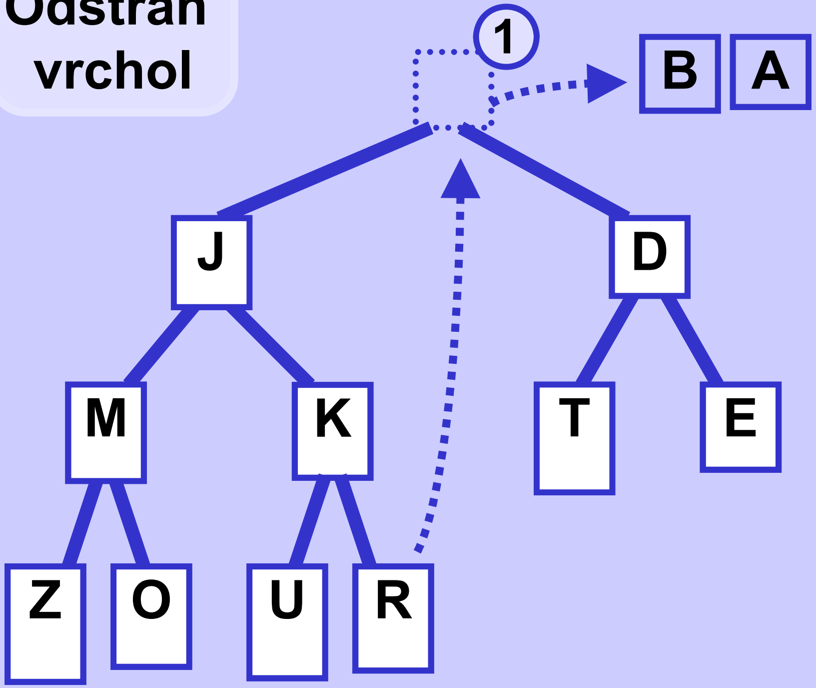


Nová halda

Oprava haldy

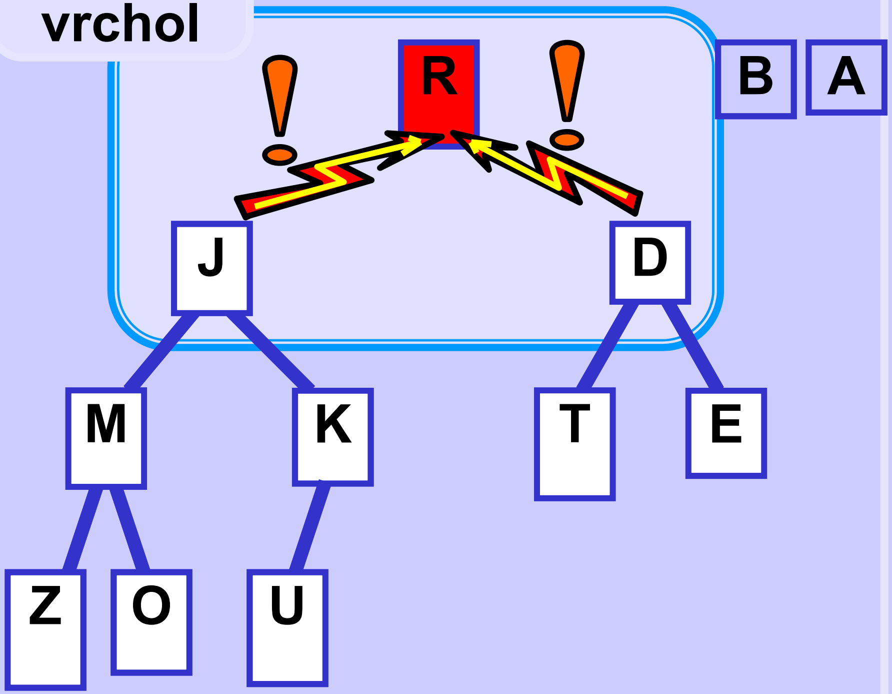
Vrchol odstraněn II (1)

1 Odstraň vrchol



2 poslední $\longrightarrow$ první

3 Vlož na vrchol



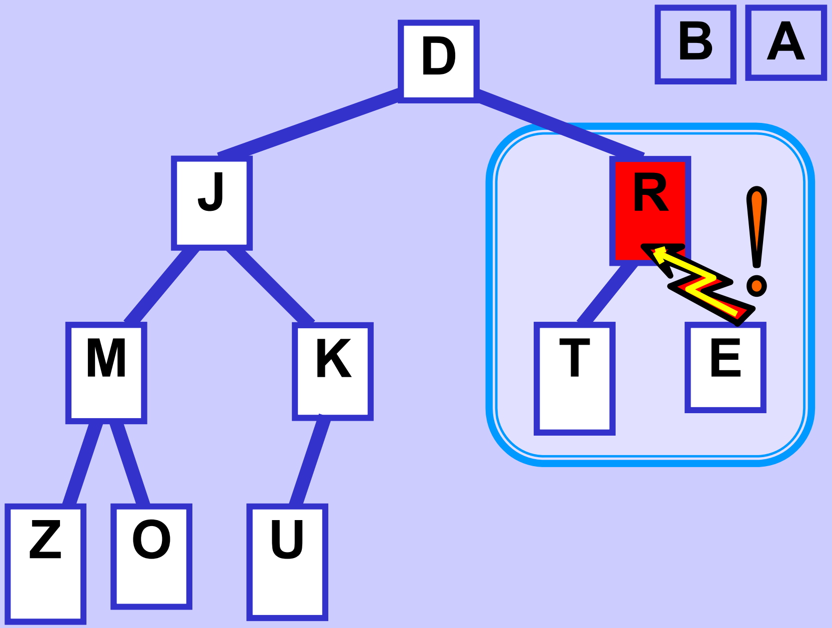
R > J, R > D, <u>D < J</u>

$\Rightarrow$ prohoď D $\longleftrightarrow$ R

Oprava haldy

Vrchol odstraněn II (2)

3 Vlož na vrchol - pokračování

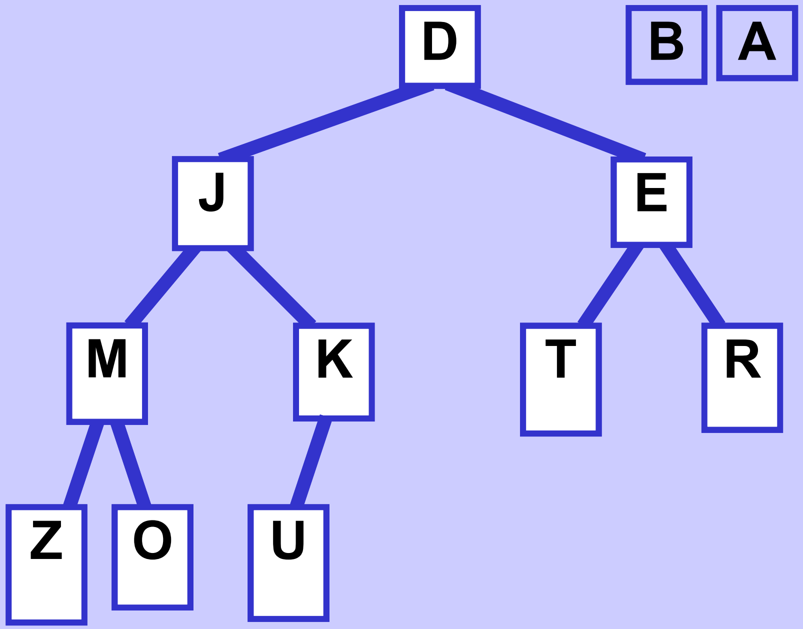


R < T, R > E

⇒ prohoď E ⟷ R

Vrchol odstraněn II (3)

3 Vlož na vrchol - hotovo



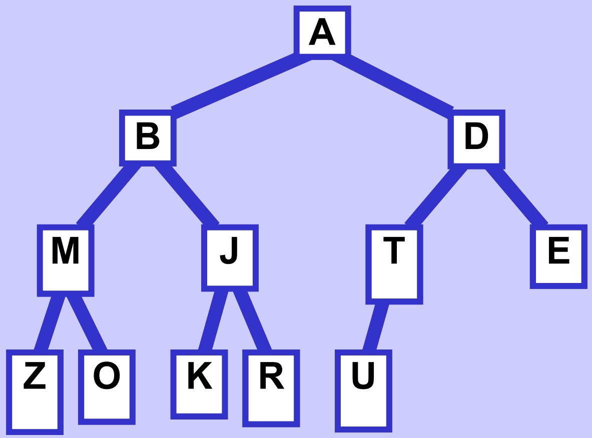
Nová halda

Heap sort

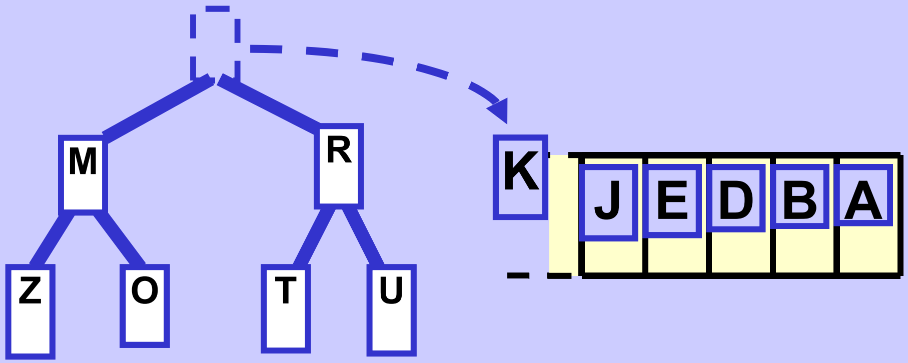
I Neseřazeno

R	J	U	Z	B	T	E	M	O	K	A	D

II Vytvoř haldu



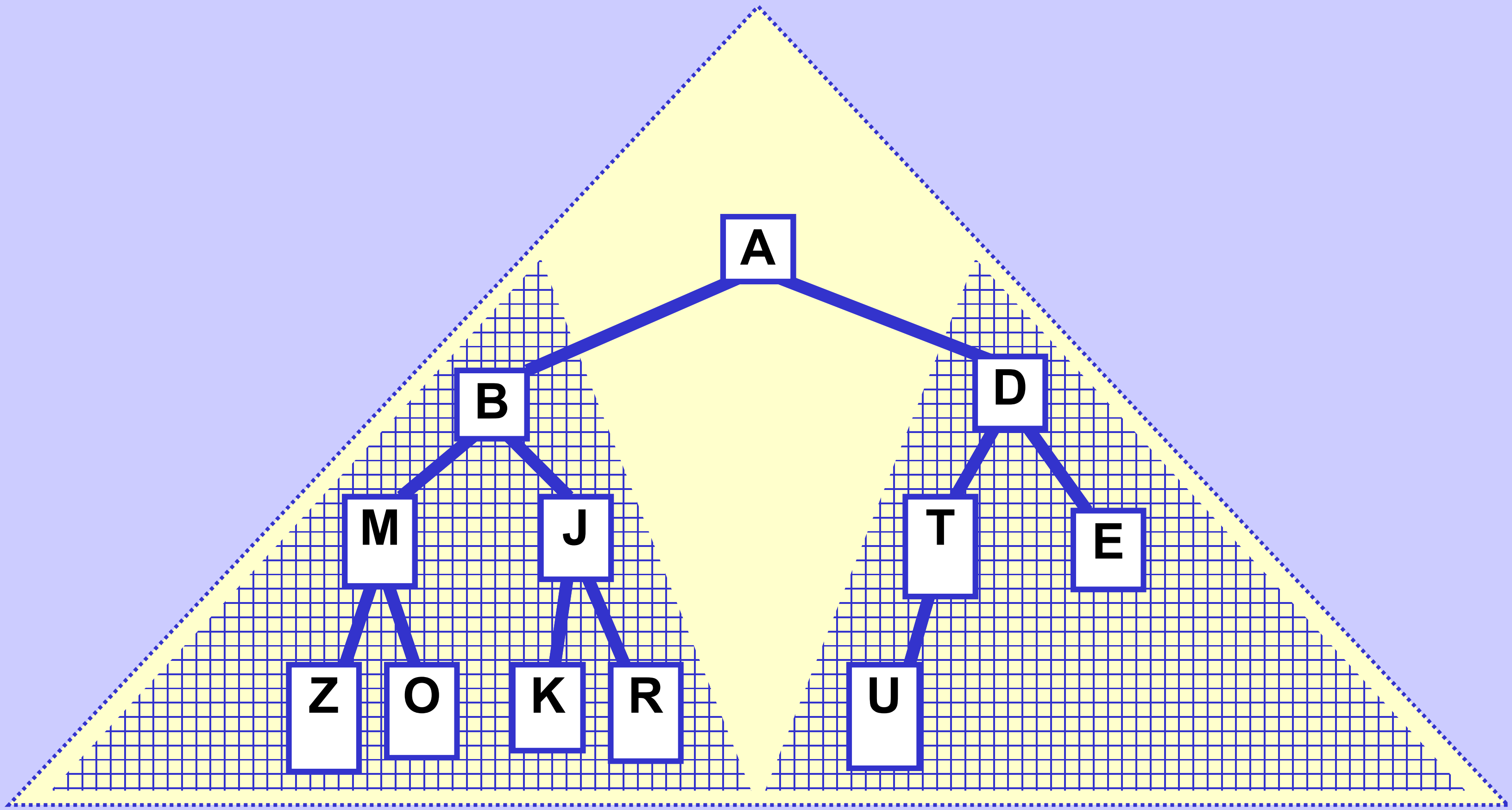
III for (i = 1; i <= n; i++)
“odstraň vrchol”;

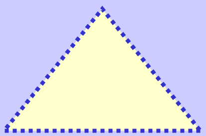
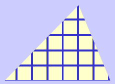
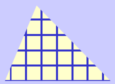


IV Seřazeno

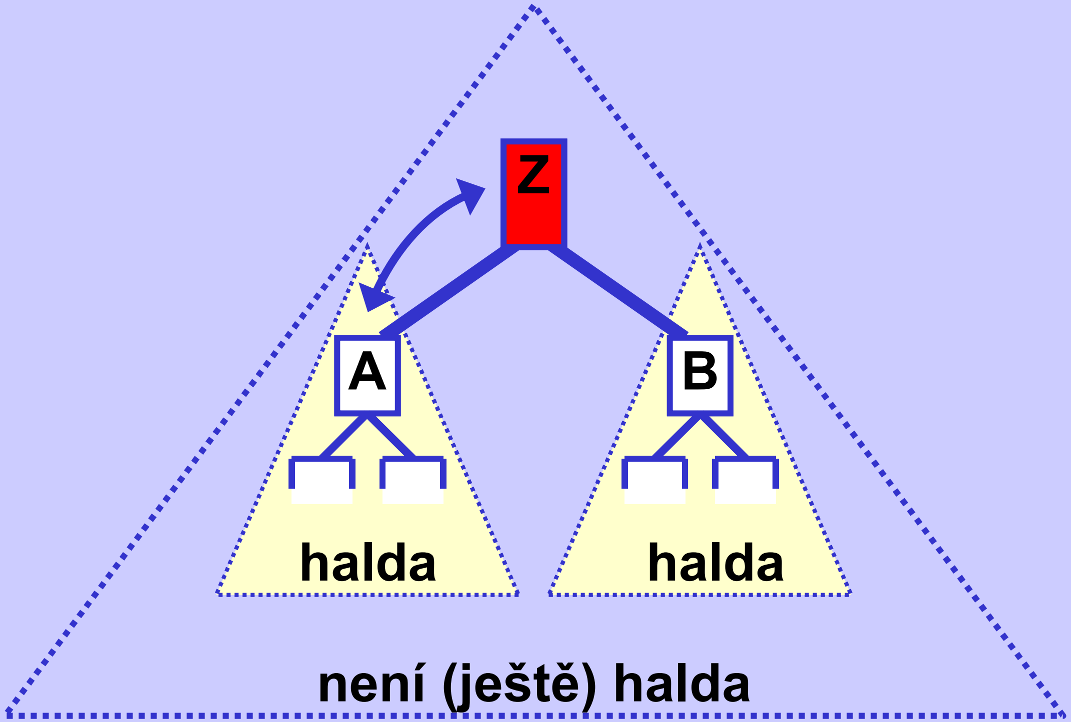
Z	U	T	R	O	M	K	J	E	D	B	A

Rekurzivní vlastnost "býti haldou"



 je halda $\Rightarrow$  je halda a  je halda

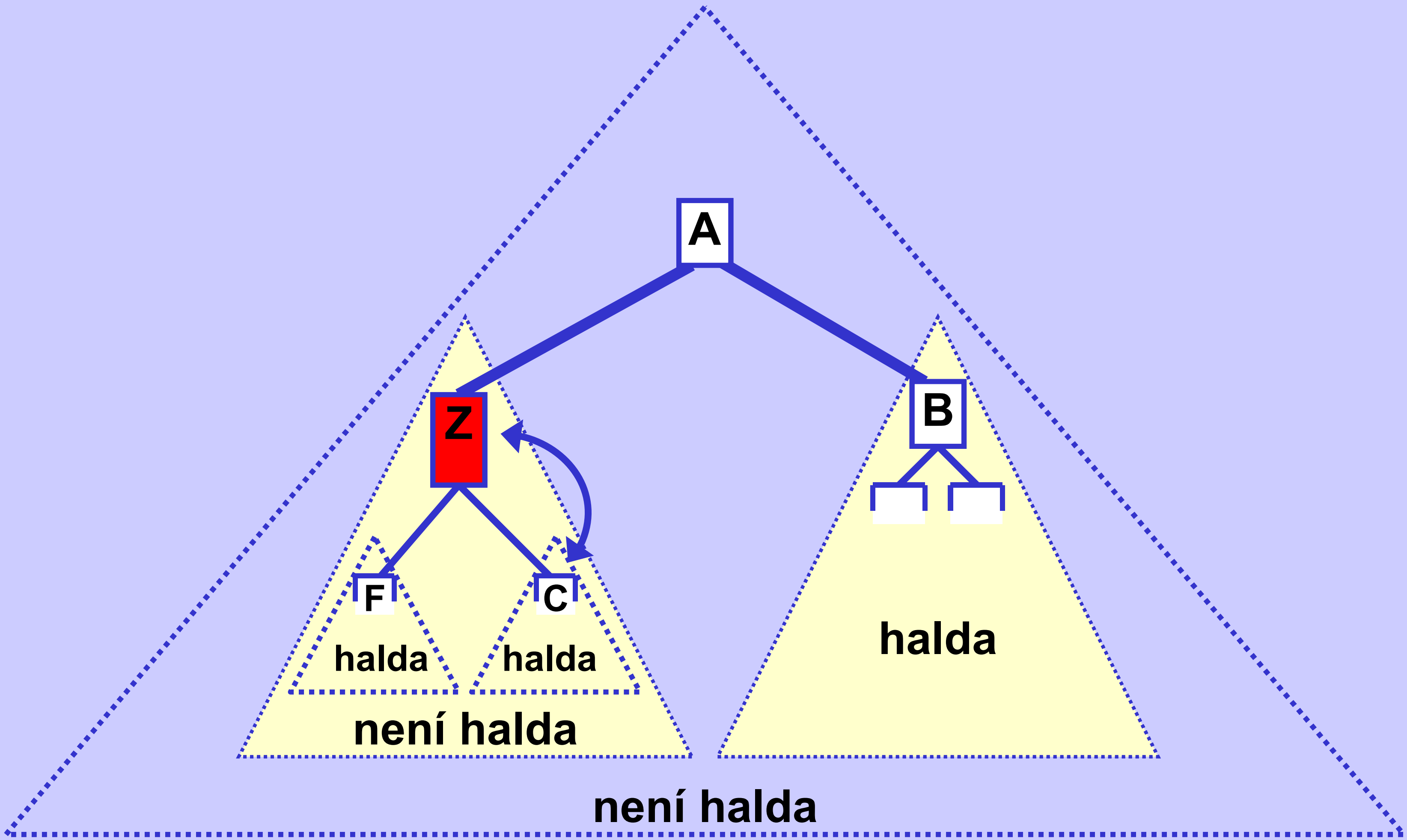
Vytvoř jednu haldu ze dvou menších



Z > A nebo Z > B

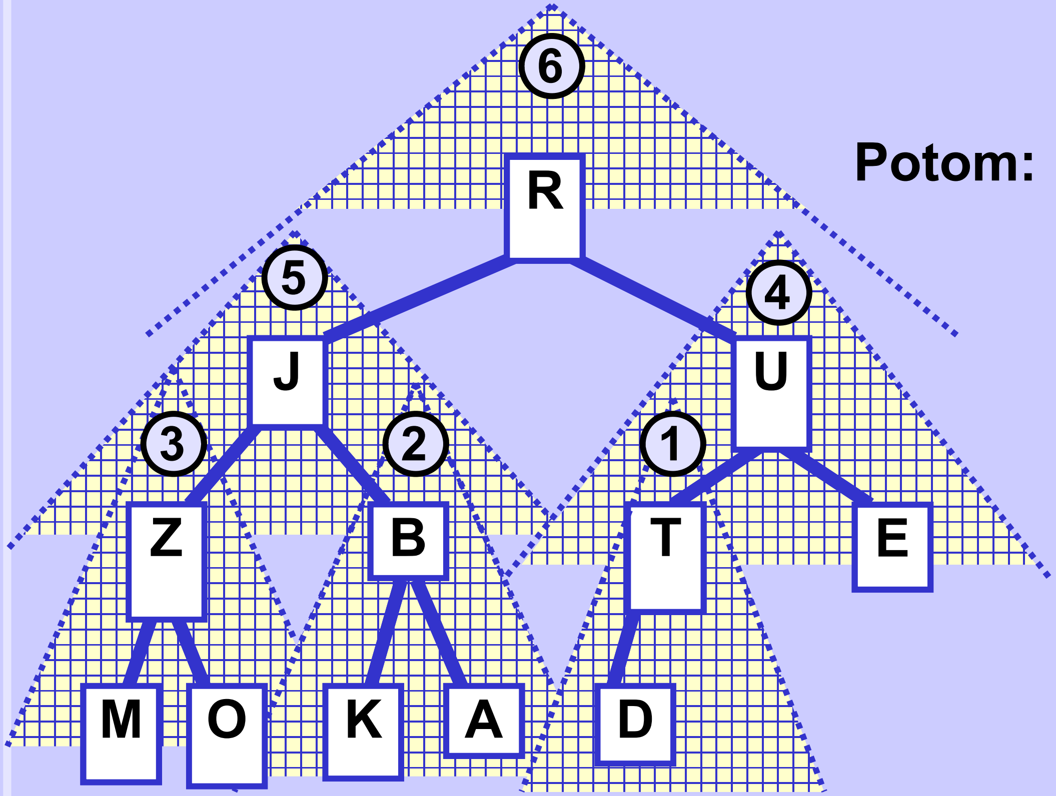
$\Rightarrow$ prohoď: Z $\longleftrightarrow$ min(A,B)

Vytvoř jednu haldu ze dvou menších



Vytvoř haldu

Stavba haldy zdola



Uvaž: Každý list je samostanou (malinkou) haldou

Potomм: Vytvoř haldu v 1 ...

... a vytvoř haldu v 2 ...

... a vytvoř haldu v 3 ...

... a vytvoř haldu v 4 ...

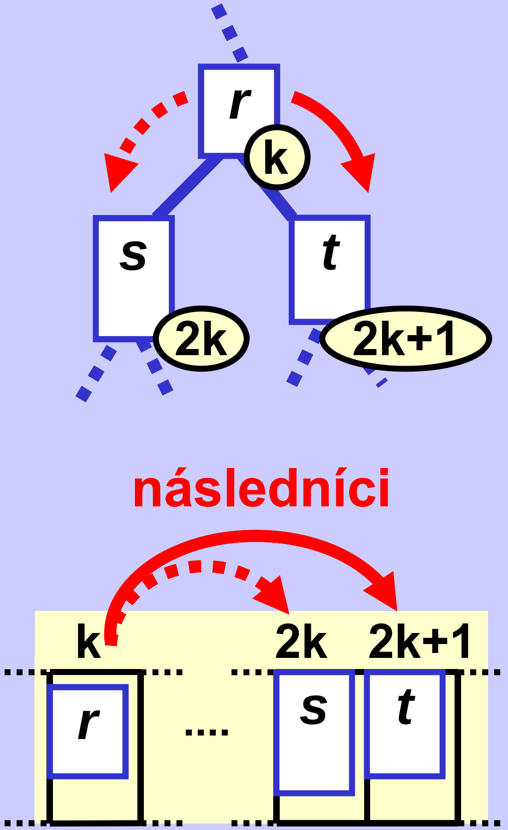
... a vytvoř haldu v 5 ...

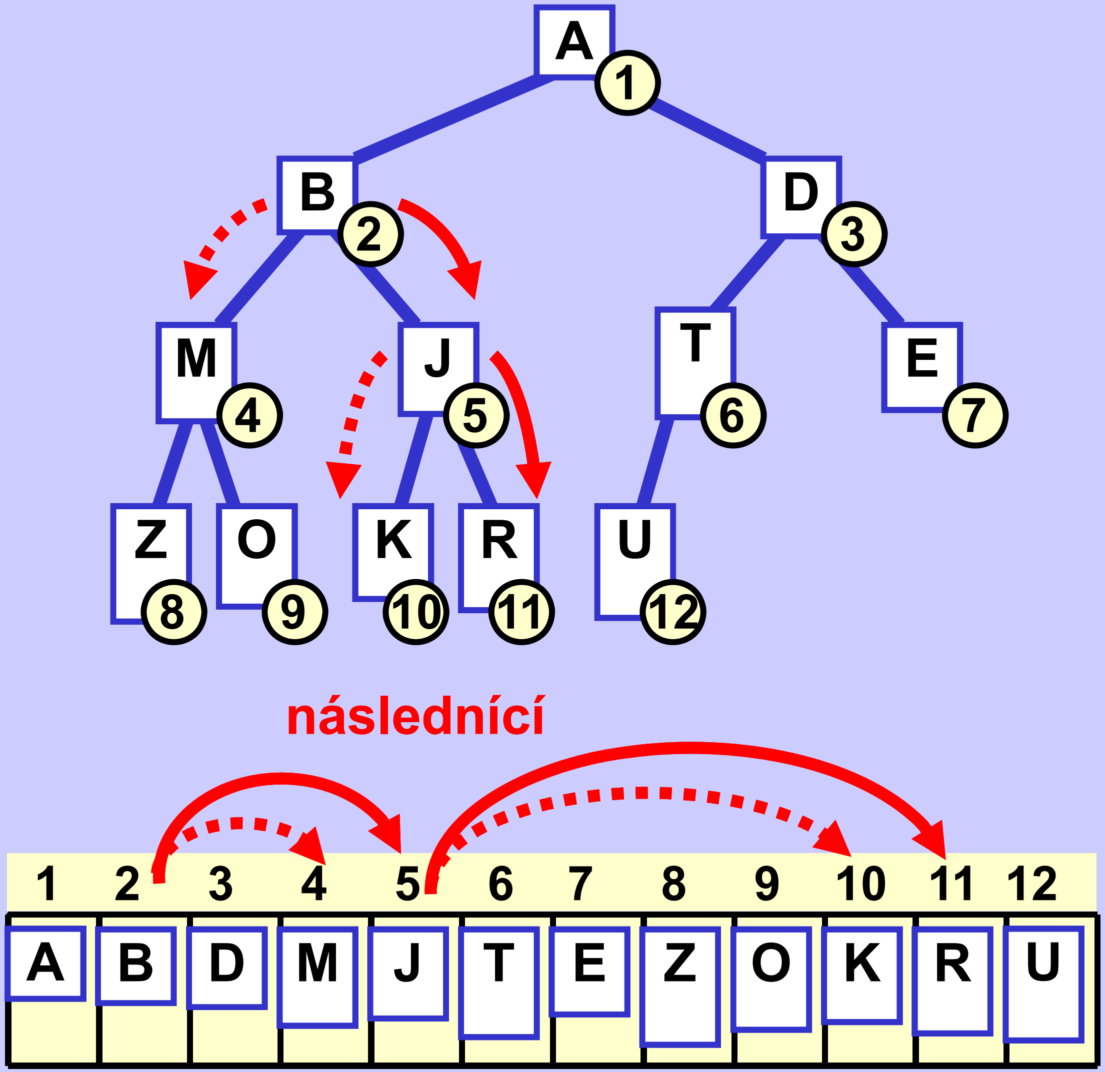
... a vytvoř haldu v 6 ...

… a celá halda je hotova.

Halda v poli

Halda uložená v poli





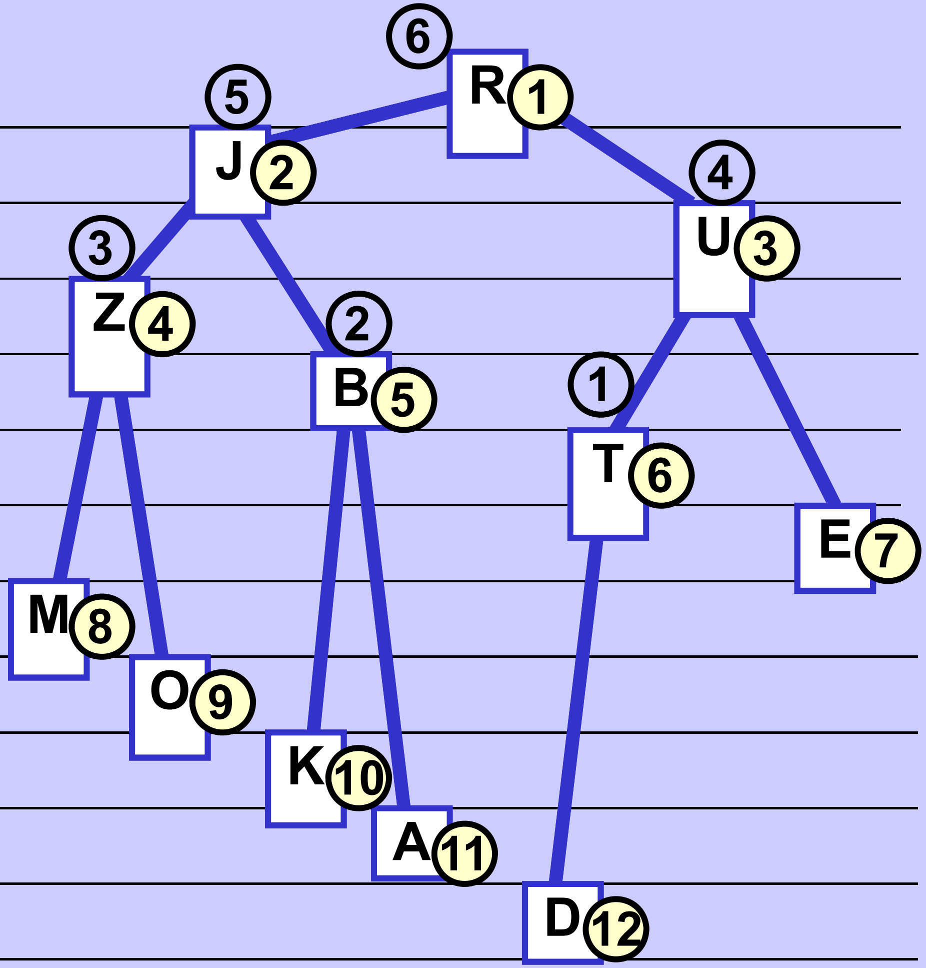
Tvorba haldy v poli

Pole

6	1	R
5	2	J
4	3	U
3	4	Z
2	5	B
1	6	T
	7	E
	8	M
	9	O
	10	K
	11	A
	12	D

Prvky náhodně uspořádány

Není halda

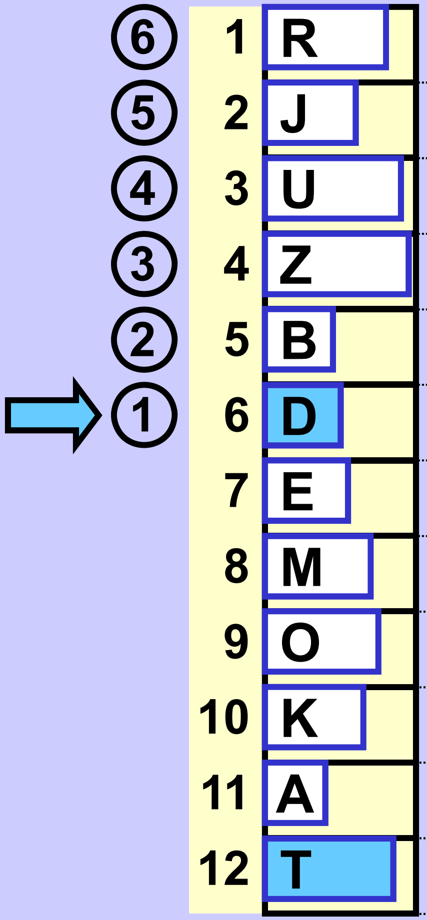


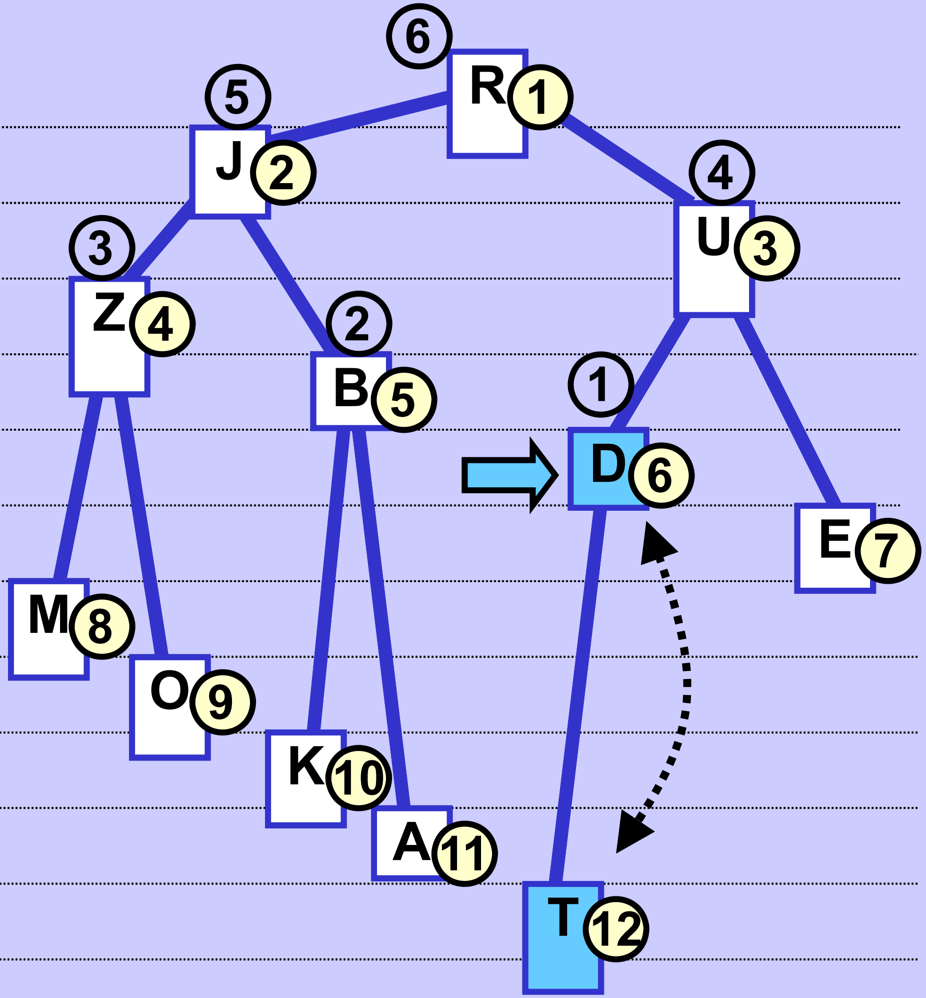
Tvorba haldy v poli

Přesuny

Aktuálně vytvořená halda

Pole





Tvorba haldy v poli

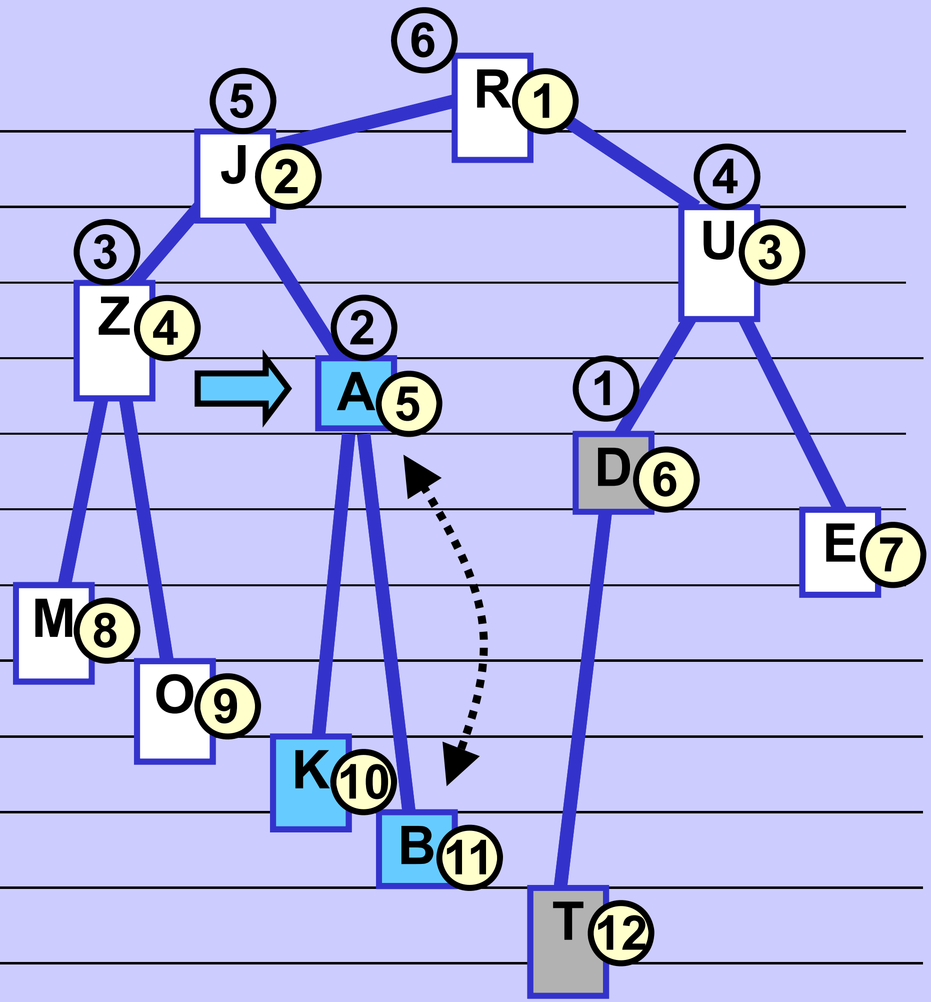
Dříve vytvořená halda

Přesuny

Aktuálně vytvořená halda

Pole

6	1	R
5	2	J
4	3	U
3	4	Z
2	5	A
1	6	D
	7	E
	8	M
	9	O
	10	K
	11	B
	12	T



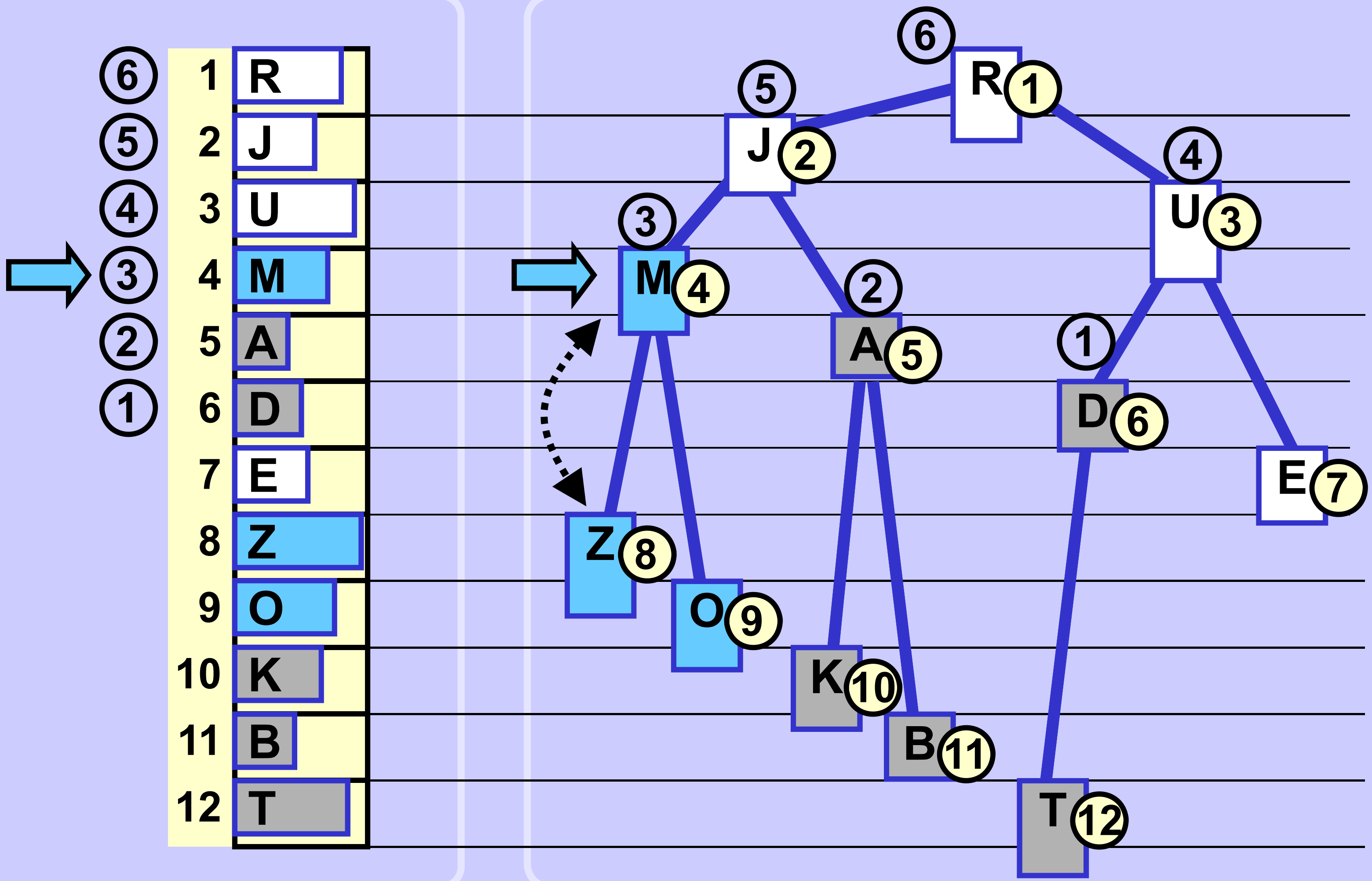
Tvorba haldy v poli

Dříve vytvořená halda

Přesuny

Aktuálně vytvořená halda

Pole



Tvorba haldy v poli

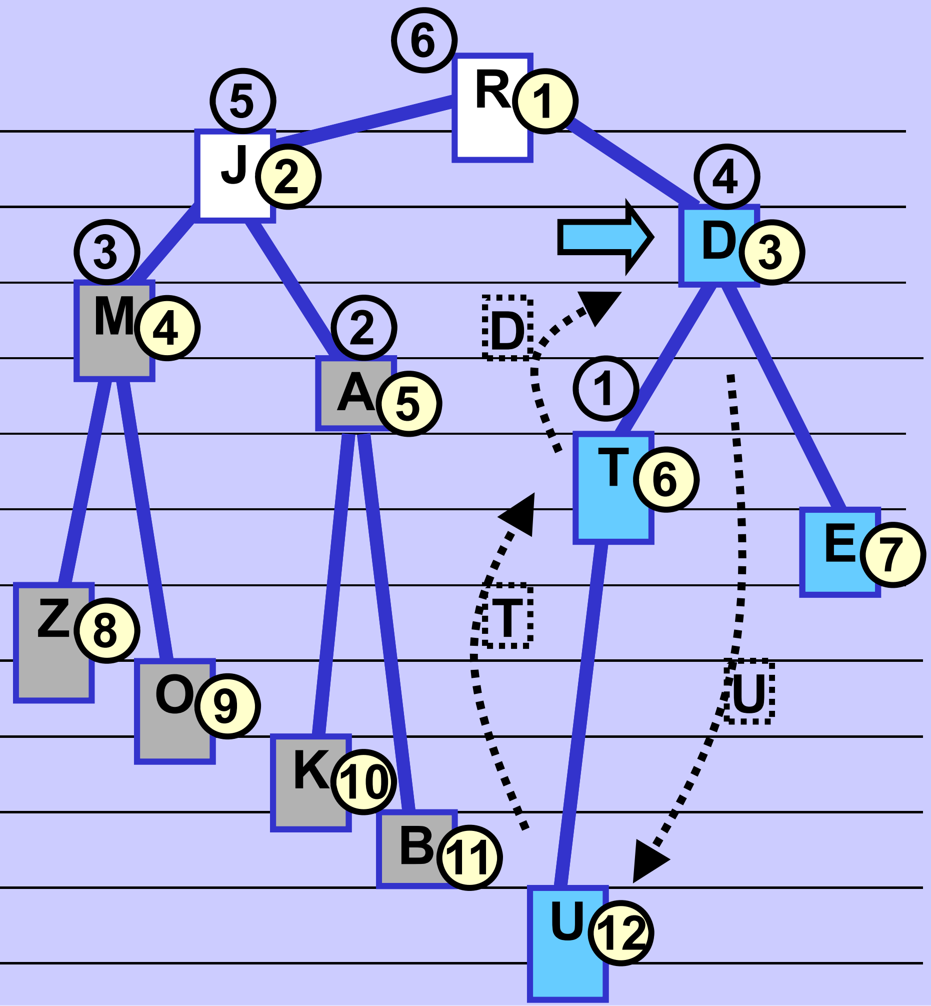
Dříve vytvořená halda

Přesuny

Aktuálně vytvořená halda

Pole

6	1	R
5	2	J
4	3	D
3	4	M
2	5	A
1	6	T
	7	E
	8	Z
	9	O
	10	K
	11	B
	12	U



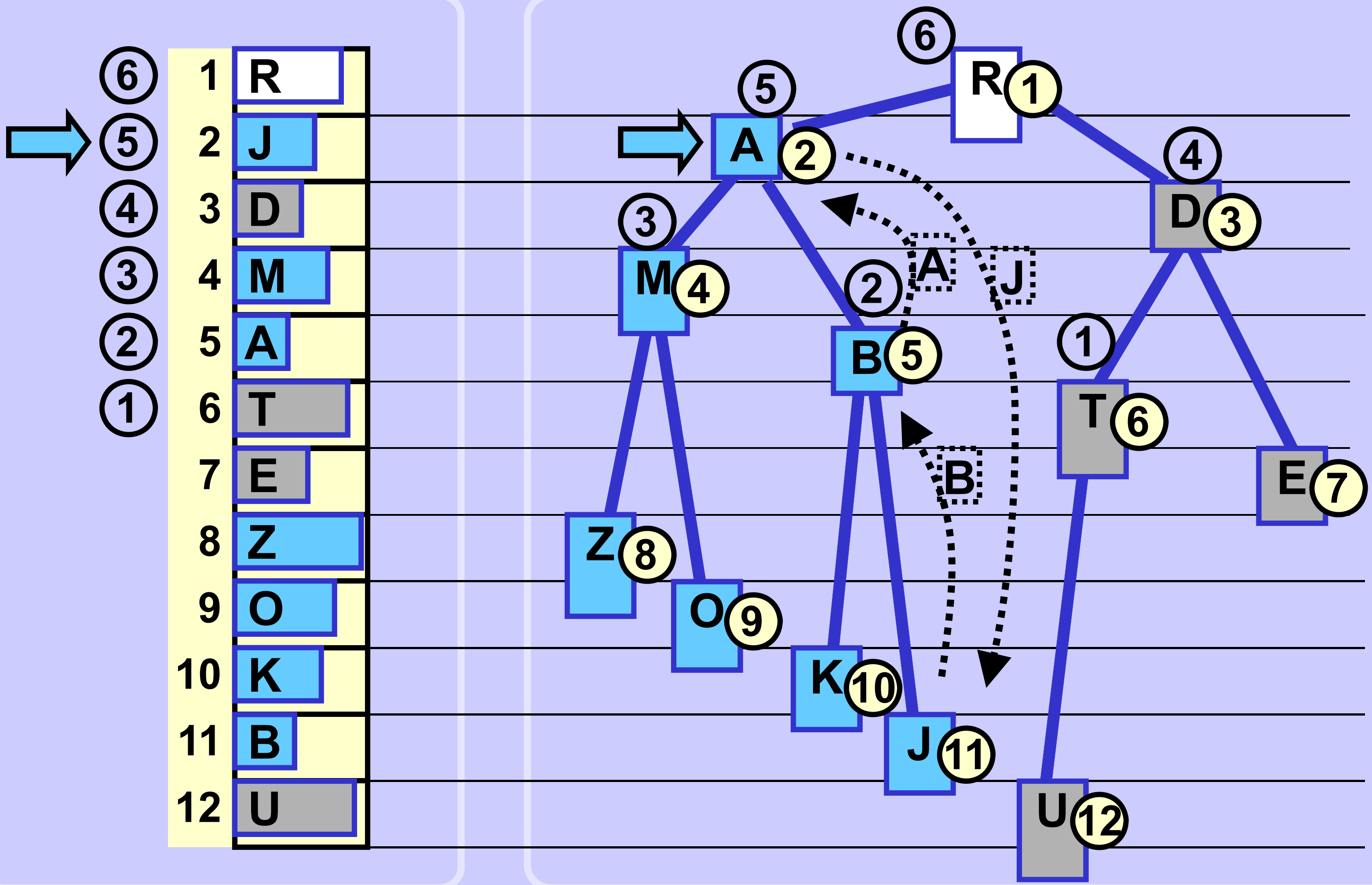
Tvorba haldy v poli

Dříve vytvořená halda

Přesuny

Aktuálně vytvořená halda

Pole

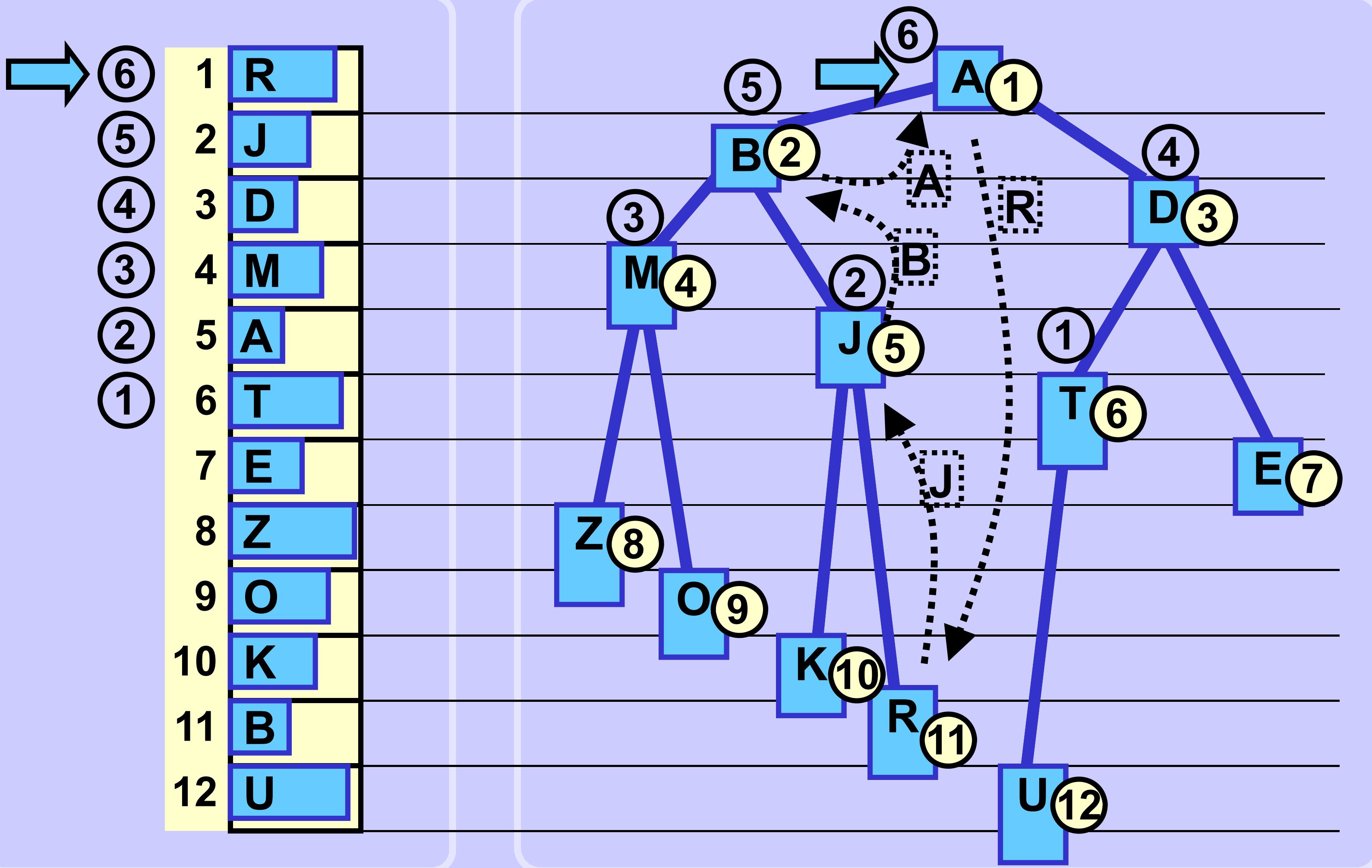


Tvorba haldy v poli

Přesuny

Aktuálně vytvořená halda

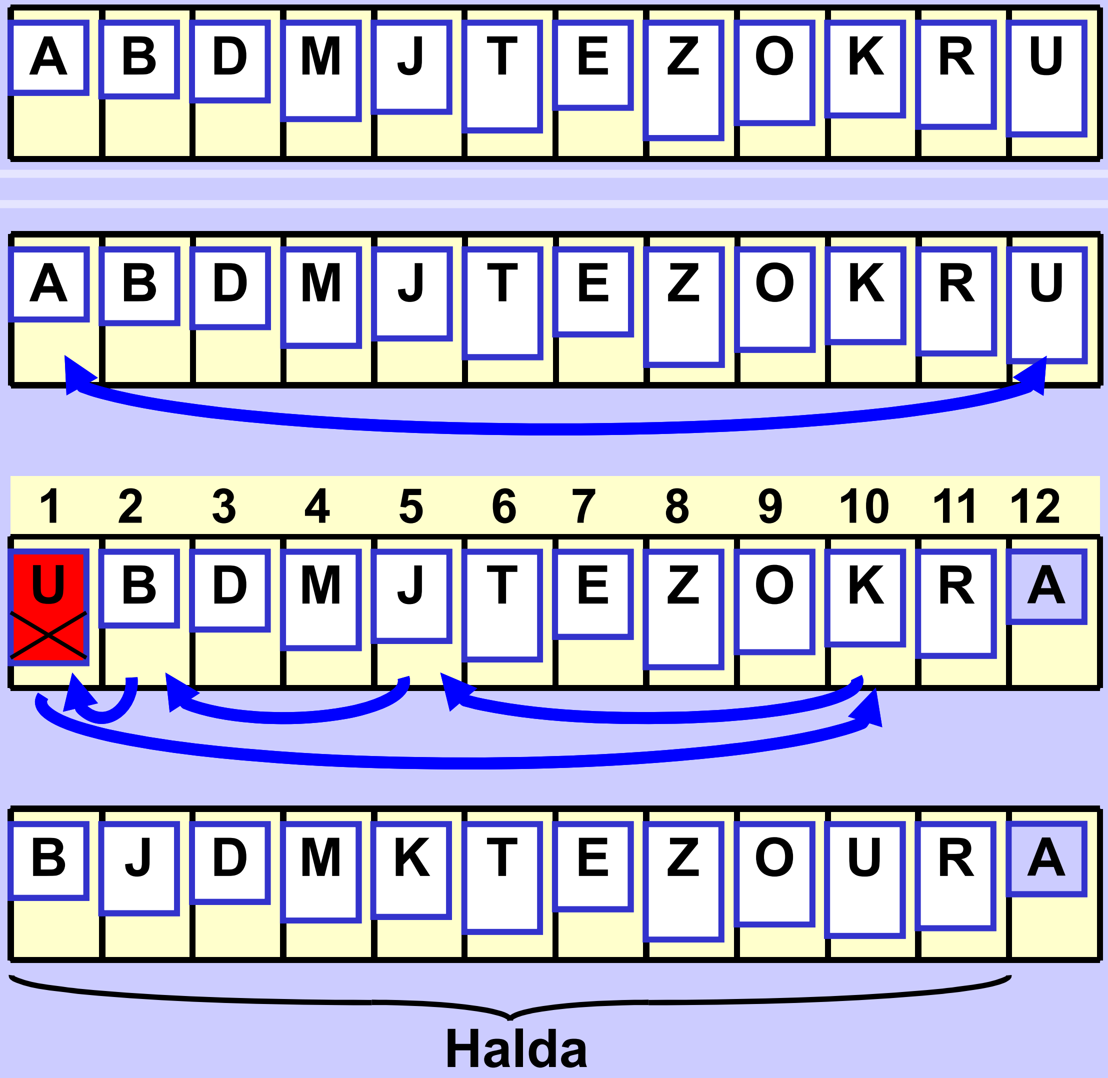
Pole



Heap sort

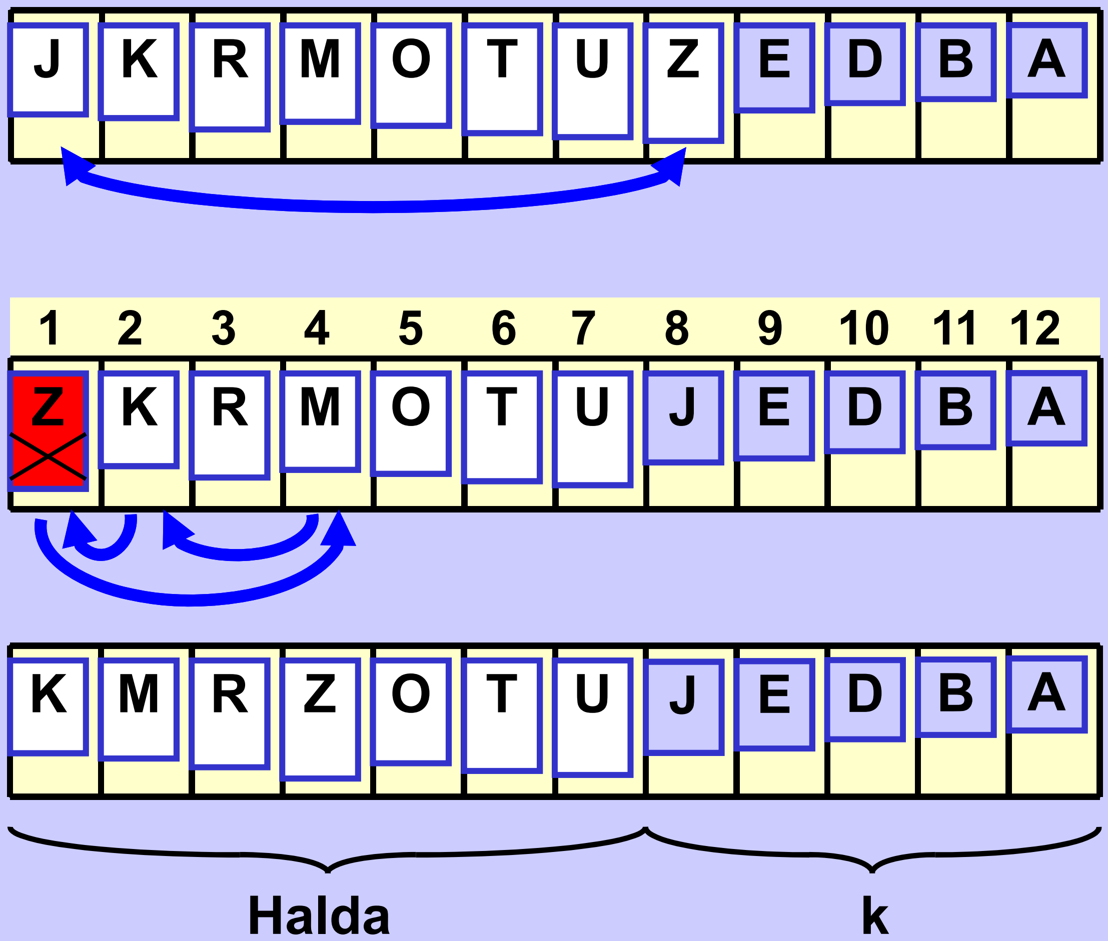
Halda

Krok 1



Heap sort

Krok k



Heap sort

```
                                 // array: a[1]...a[n]  !!!!

void heapSort( Item [] a, int n ) {

                                        // create a heap
  for( int i = n/2; i > 0; i-- )
    repairTop( a, i, n );

                                        // sort
  for( int i = n; i > 1; i-- ) {
    swap( a, 1, i );
    repairTop( a, 1, i-1 );
  }
}
```

Heap sort

```
                         // array:  a[1]...a[n]  !!!!!!
void repairTop( Item [] a, int top, int bottom ) {
  int i = top;           // a[2*i] and a[2*i+1]
  int j = i*2;           // are children of a[i]

  Item topVal = a[top];

                        // try to find a child < topVal
  if( (j < bottom) && (a[j] > a[j+1]) ) j++;

                       // while (children < topVal)
                       //       move children up
  while( (j <= bottom) && (topVal > a[j]) ){
    a[i] = a[j];
    i = j;  j = j*2; // skip to next child
    if( (j < bottom) && (a[j] > a[j+1]) ) j++;
  }
  a[i] = topVal;     // put topVal where it belongs
}
```

Heap sort

repairTop operace nejhorší případ ... $\log_2(n)$ (n=velikost haldy)

vytvoř haldu ... n/2 repairTop operací

$\log_2(n/2) + \log_2(n/2+1) + \ldots + \log_2(n) \leq (n/2)(\log_2(n)) = \underline{\underline{O(n\cdot\log(n))}}$

lze ukázat ... n/2 repairTop operací ... $\underline{\underline{\Theta(n)}}$

seřaď haldu ... n-1 repairTop operací, nejhorší případ:

$\log_2(n) + \log_2(n-1) + \ldots + 1 \leq n \cdot \log_2(n) = O(n\cdot\log(n))$

celkem ... vytvoř haldu + seřaď haldu = $\underline{\underline{O(n\cdot\log(n))}}$

Asymptotická složitost Heap sortu je $O(n\cdot\log(n))$

V praktických situacích se očekává $\Theta(n\cdot\log(n))$

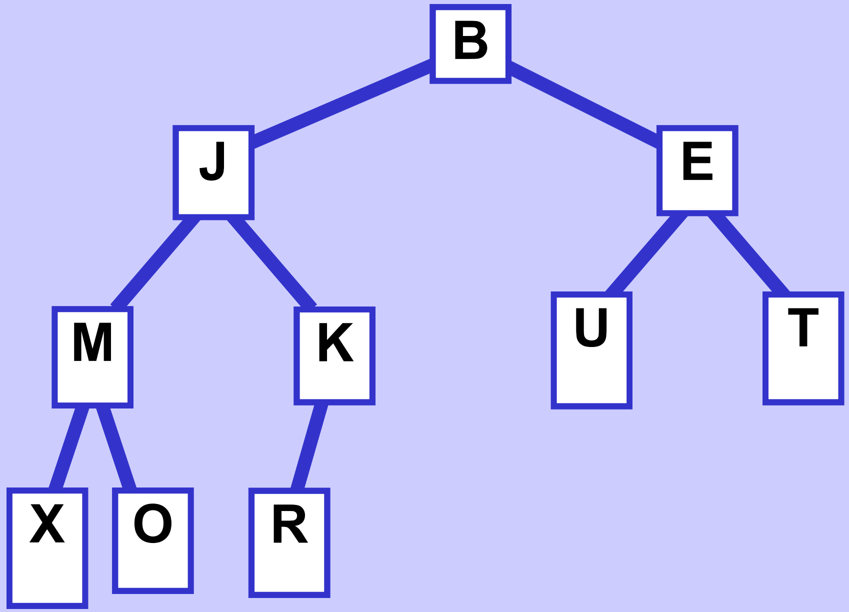
Heap sort není stabilní

Otázka: Pro jaká data nastane složitost menší než $\Theta(n\cdot\log(n))$?

Prioritní fronta

Prioritní fronta má stejné operace jako obyčejná fronta
- vlož do fronty (Insert, Enqueue, apod),
- čti první prvek (Front, Top, apod),
- smaž první prvek (Dequeue, Pop, apod).

Navíc interně zajišťuje, že na čele fronty je vždy prvek s minimální (maximální) hodnotou ze všech prvků ve frontě.



Prioritní frontu lze implementovat pomocí haldy.

Plným jménem: "Binární haldy".

Prioritní fronta pomocí binární haldy -- operace

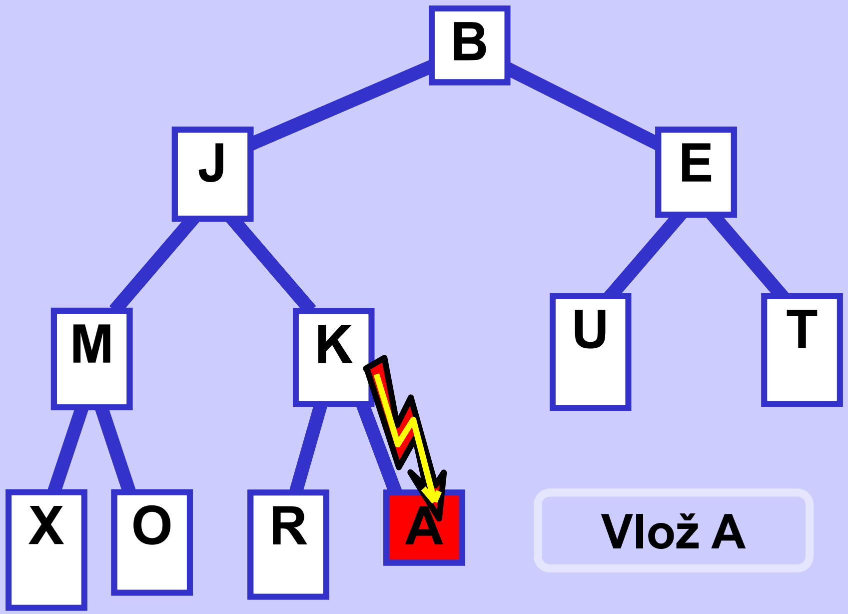
Čti první prvek (Front, Top, apod) .

Zřejmé.

Smaž první prvek (Dequeue, Pop, apod) = Odstraň vrchol a oprav haldu.

Viz výše.

Vlož do fronty (Insert, Enqueue, apod).



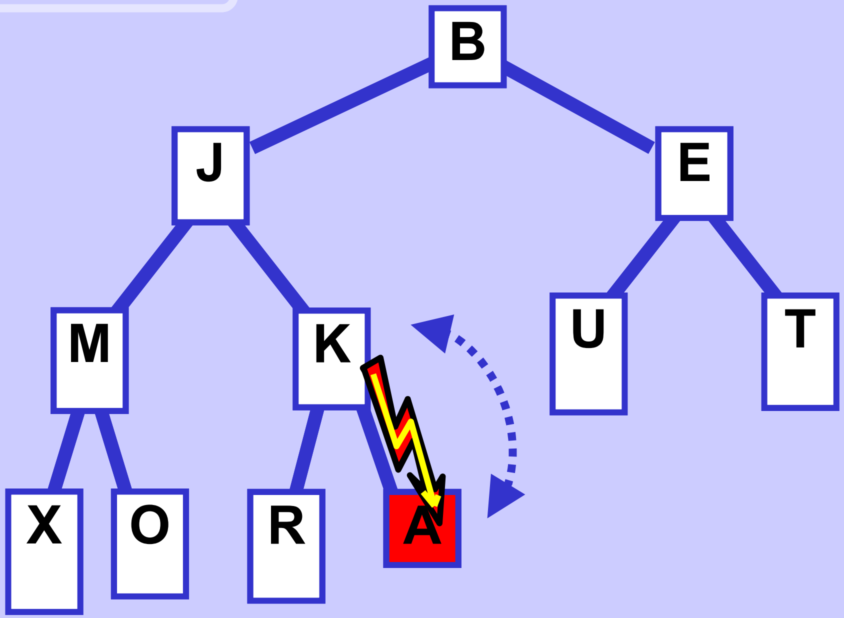
Vložíme prvek
na konec fronty (haldy).

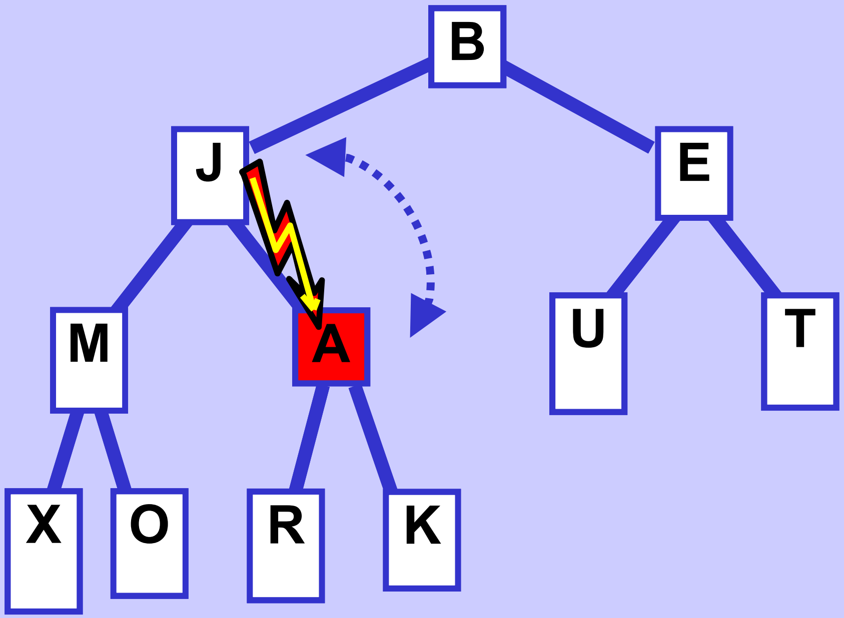
Ve většině případů
se tím poruší pravidlo haldy

a je nutno haldu opravit.

Prioritní fronta pomocí binární haldy – vlož prvek

Vlož A



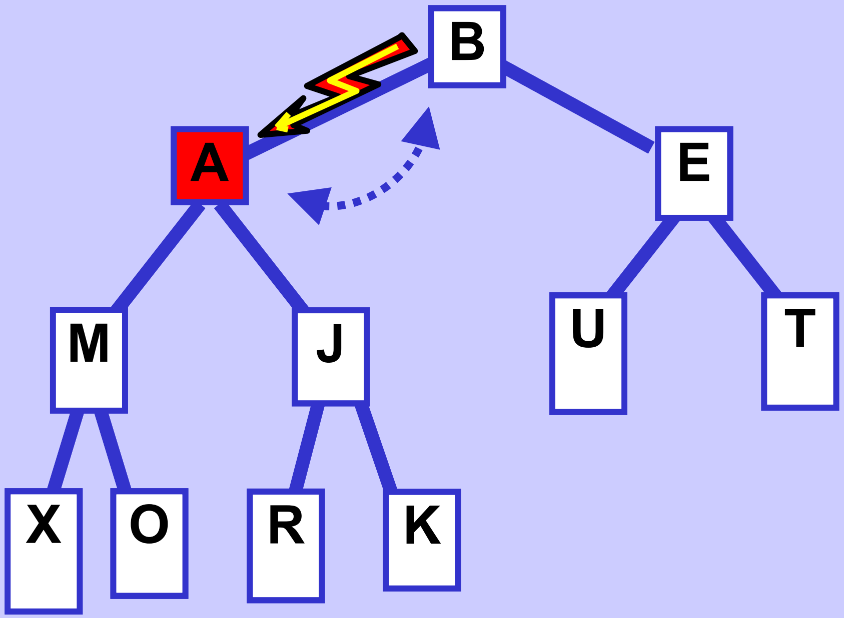


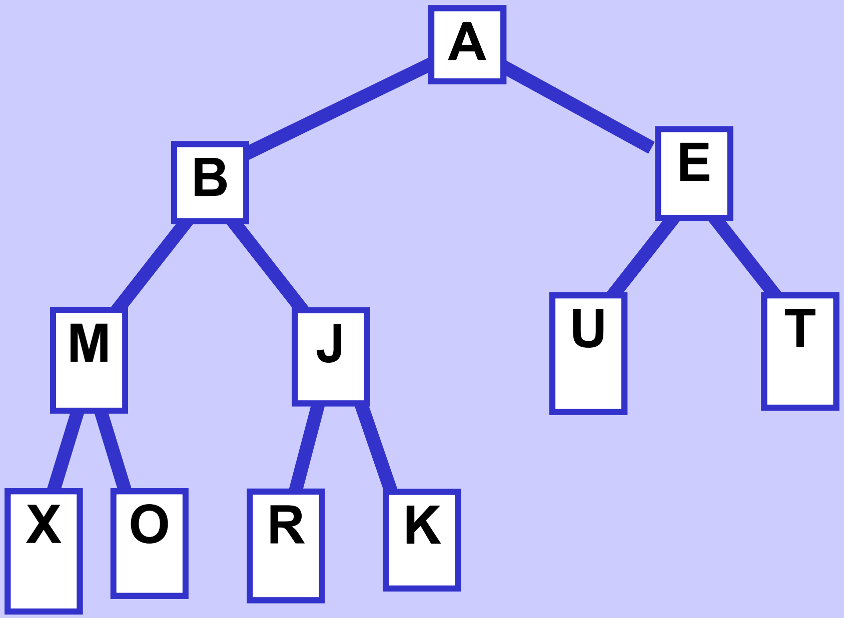
Pravidlo haldy je porušeno,
vyměň vkládaný prvek
s jeho rodičem.

Pravidlo haldy je stále porušeno,
vyměň vkládaný prvek
s jeho rodičem.

Prioritní fronta pomocí binární haldy – vlož prvek

Vkládáme A

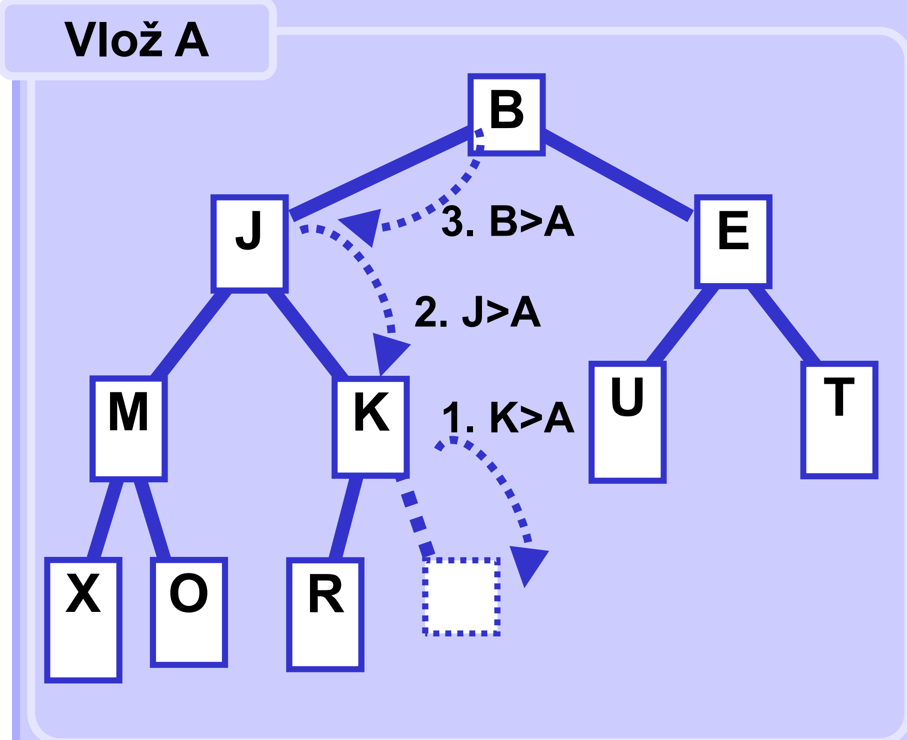


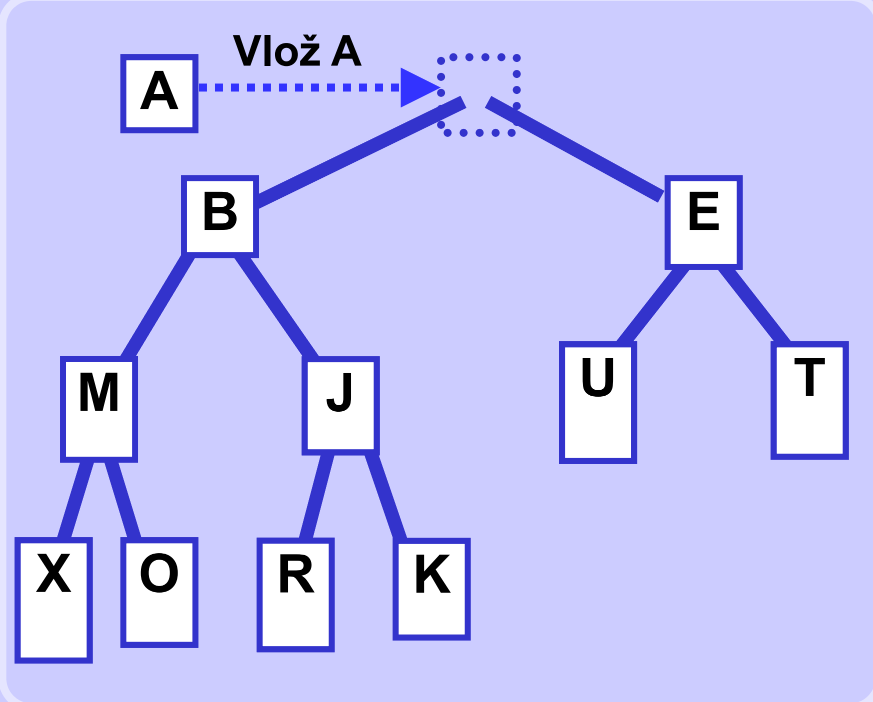


Pravidlo haldy je stále porušeno, vyměň vkládaný prvek s jeho rodičem.

Pravidlo haldy je zachováno, vkládaný prvek našel své místo v haldě.

Binární halda – vlož prvek, efektivněji





Vkládaný prvek na konec haldy nevkládej.
Napřed zjisti, kam patří, ostatní (větší) prvky posuň o patro dolů ...

... a teprve nakonec
vlož prvek na jeho místo.

Binární halda – vlož prvek

```
                                   // array:  a[1]...a[n]  !!!!!!
int insert( Item [] a, int x, int bottom ){
   int j = ++bottom;               // expand the heap
   int i = j/2;                    // parent index

   while( (i > 0) && (a[i] > x) ){
      a[j] = a[i];                 // move elem down the heap
      j = i; i /= 2;               // move indices up the heap
   }
   a[i] = x;                       // put inserted elem to its place
   return bottom;
}
```

Binární halda – Složitost operace insert

Vkládání představuje průchod vyváženým stromem s n prvky od listu nejvýše ke kořeni, složitost operace Insert je tedy $O(\log_2(n))$.