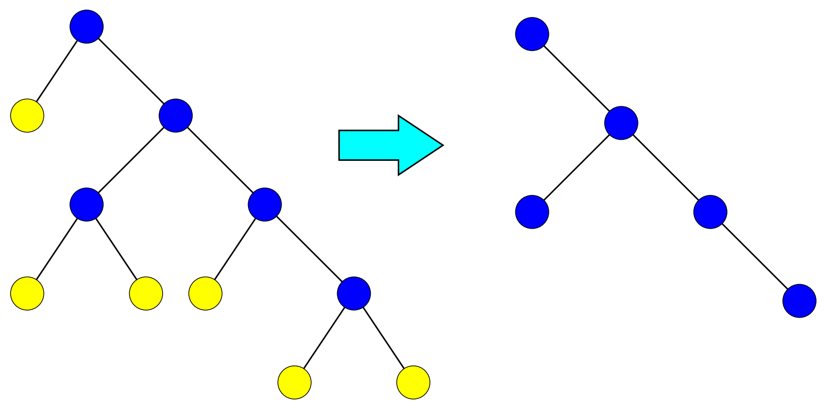
Section #26: **Parallel tree contraction**

Tree contraction

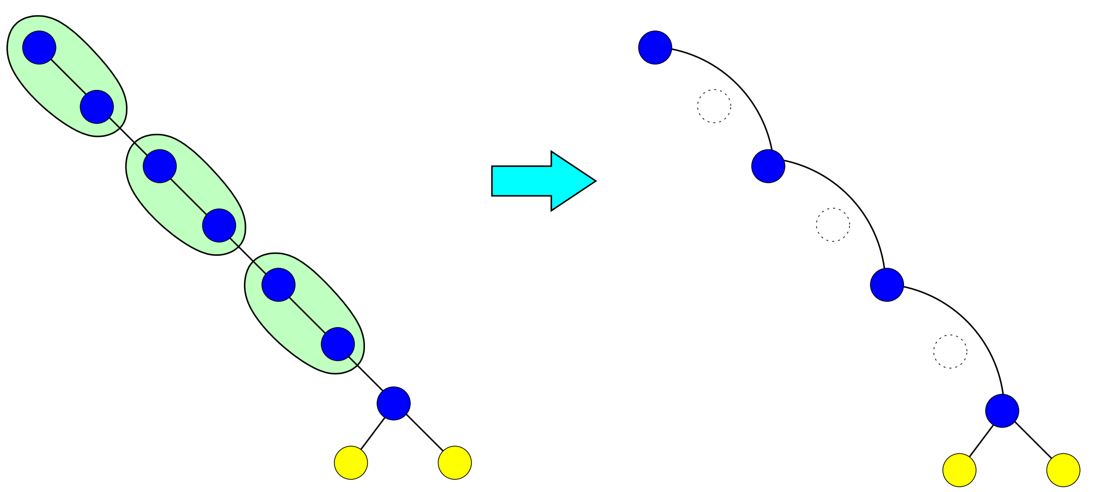
- ▷ ***Divide&Conquer***: parallel splitting of trees into approx. equal subtrees difficult
- ▷ ***bottom-up***: local modifications of the tree by removing leaves

Rake and Compress

- ▷ **Rake**: removing leaves. But: tends to linearize trees $\Longrightarrow$ linear parallel time
- ▷ **Compress**: reduces chains using ***pointer jumping***
- ▷ **Compress** and **Rake** can be applied in parallel to disjoint parts of a tree.
- ▷ **Compress** produces leaves for **Rake** and **Rake** produces linear lists for **Compress**.



Rake



Compress

Basic Contract: CREW PRAM algorithm for generic parallel tree contraction

$T = (V, E)$, $|V| = n$

Input:	$P[1, \ldots, n]$; /* $P[x]$ is a pointer to the parent of x */
	$children[1, \ldots, n]$; /* $children[v] = \{v_1, \ldots, v_k\}$ – pointers to all children */
	$index[1, \ldots, n]$; /* $index[v_i] = i$ – each child v knows its index in $children[P[v]]$ */
Auxil:	$label[1, \ldots, n]$; /* $label[v] = \{f_1, \ldots, f_k\}$, where $f_i \in \{U, M\}$ */
	/* $f_i = M$ = marked iff a child supplied its value to its parent */
	$UnMarkChil(x)$ **returns** `int`; /* function returning the # of unmarked children */
Output:	the value accumulated in the root

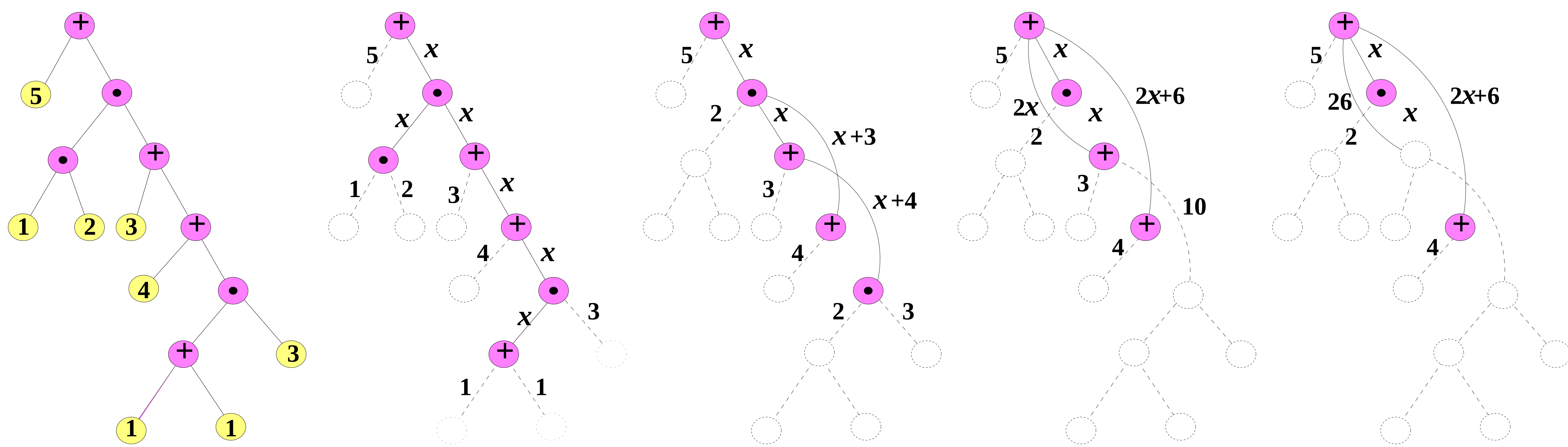
```
/* initialize the data structures */
for all nodes v ∈ T do_in_parallel initialize(v);
while UnMarkChil(root) > 0 do
   { for all nodes v ∈ T do_in_parallel
      if P[v] ≠ nil then
        { case UnMarkChil[v] of
          0: { Rake(v); label[P[v]][index[v]] := M; P[v] := nil; }
          1: if UnMarkChil[P[v]] = 1 then { Compress(v); P[v] := P[P[v]]; }
        endcase }};
Rake(root);
```

Complexity

Theorem 1 *After* $O(\log_{4/3} n)$ *applications of* **Basic Contract** *to* n*-node tree* T*, it is reduced to a root. If* **Rake** *and* **Compress** *take* $O(1)$ *time, then the parallel time with* $p = \Theta(n)$ *is* $O(\log n)$*.*

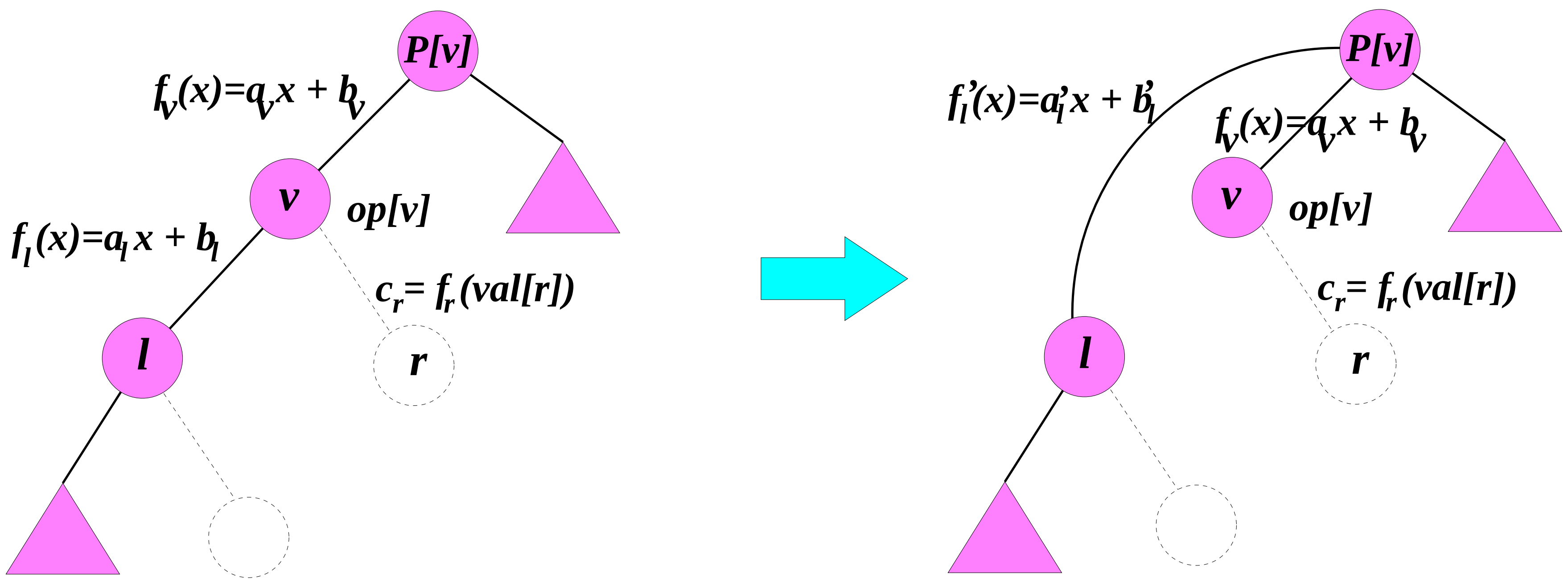
Binary expression tree evaluation

- ▷ internal nodes represent binary operators $+$ and $\times$
- ▷ leaves contain constant input integer values.



(a) (b) (c) (d) (e)

Compress



The algorithm

Input:	$P[1,\ldots,n]$; /* $P[x]$ is a pointer to the parent of x */
	$val[1,\ldots,n]$; /* $val[v]$ – value in v after its subtree is evaluated */
	$op[1,\ldots,n]$; /* $op[v]$ is the operator of an internal node v */
	$side[1,\ldots,n]$; /* $side[v] \in \{L,R\}$ */
Auxil:	$(a,b)[1,\ldots,n]$; /* $a[v]$ and $b[v]$ are labels of edge $\langle v, P[v]\rangle$ */
	$contr[1,\ldots,n][L,R]$ /* auxiliary array to store contributions from children */
	$UnMarkChil(x)$ **returns** `int`; /* function returning the # of unmarked children */
Output:	the value of the expression tree stored in the root

```
/* initialize the data structures */
for all nodes v ∈ T do_in_parallel /* initialize(v) */
   if UnMarkChil(v) = 0 /* leaves */
     then {contr[P[v]][side[v]] := val[v]; P[v] := nil; }
     else (a, b)[v] := (1, 0); /* internal nodes */
while UnMarkChil(root) > 0 do
  { for all nodes v ∈ T do_in_parallel
     if P[v] ≠ nil then
       { case UnMarkChil[v] of
         0: { val[v] := eval(op[v], contr[v][L], contr[v][R]); /* Rake(v) */
            contr[P[v]][side[v]] := a[v]val[v] + b[v]; P[v] := nil; } /* Mark */
         1: if UnMarkChil[P[v]] = 1 then /* Compress(v) */
           { (a, b)[v] := simplify((a, b)[v], (a, b)[P[v]], op[P[v]], contr[P[v]][side_sibl[v]]);
            P[v] := P[P[v]]; }
       endcase }};
val[root] := eval(op[root], contr[root][L], contr[root][R]);
```

Drawbacks

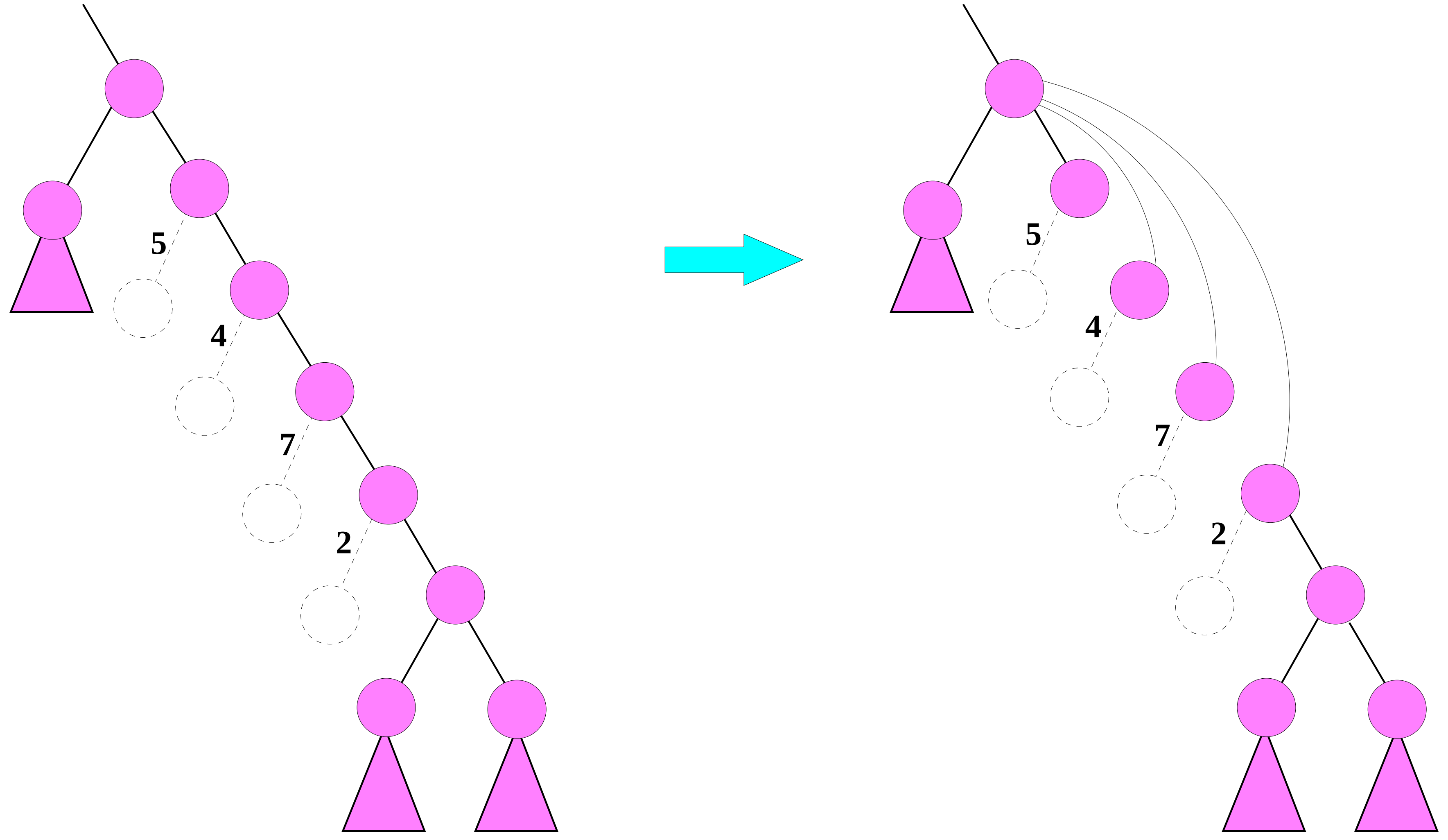
▷ **Basic Contract *is not work-optimal*.**

- $W(n,p) = n \log n$ (in contrast to $SU(n) = O(n)$)
- Reason: except ***essential*** chains, needed for the result

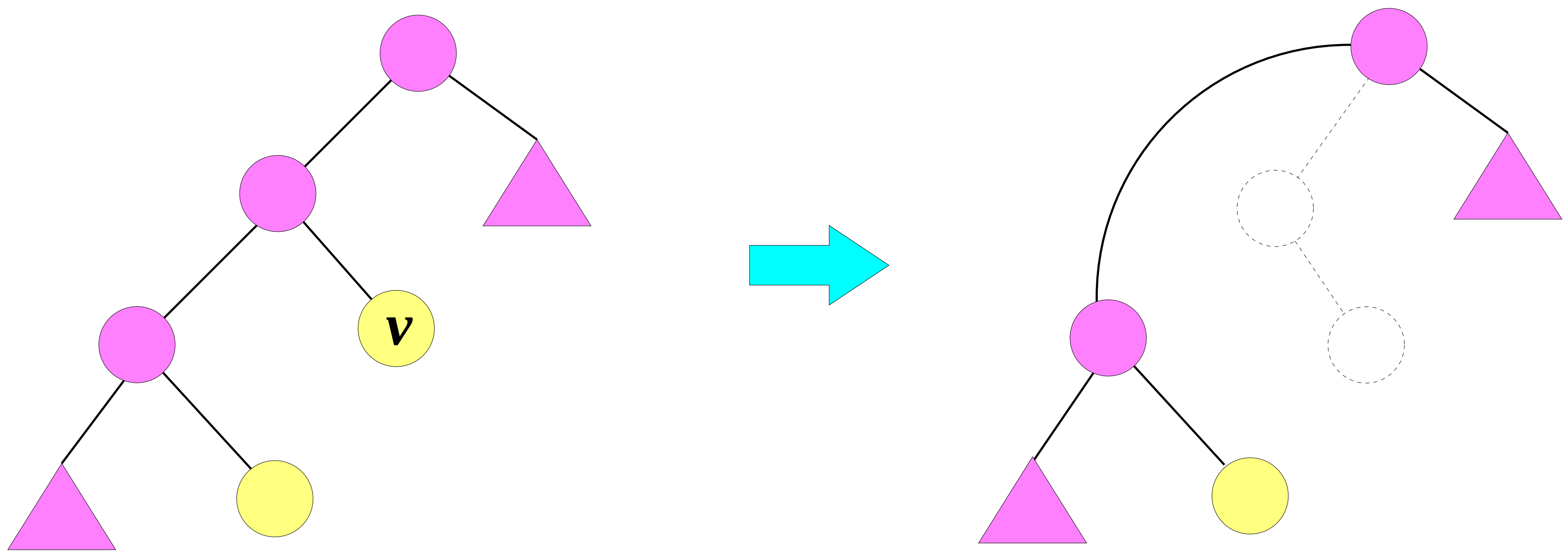
Compress produces also ***nonessential*** chains

- tree = linked list $\Longrightarrow$ the same problem as list ranking using pointer jumping

▷ ***It requires CREW PRAM***



Solution = Shunt operation



$$\textbf{Shunt}(v) = \textbf{Rake}(v) + \textbf{Compress}(sibling[v]).$$

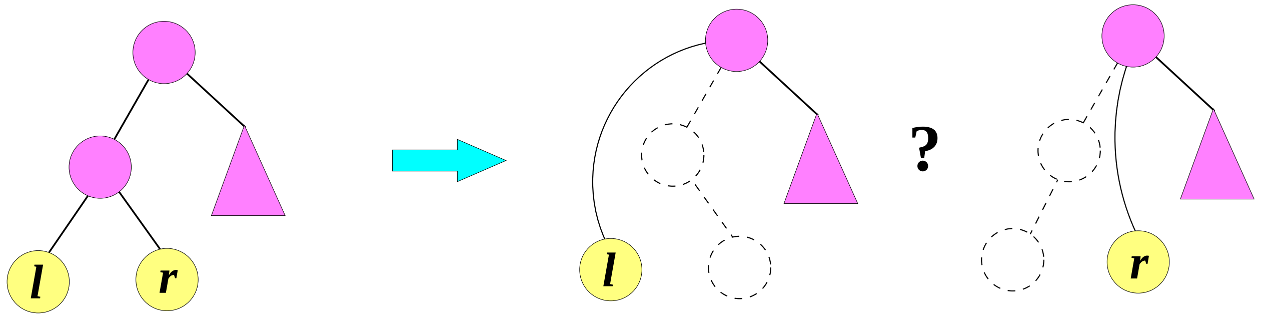
Parallel Shunt constraints

▷ **Shunt *is not defined for children of the root***

Compress cannot be applied to the root.

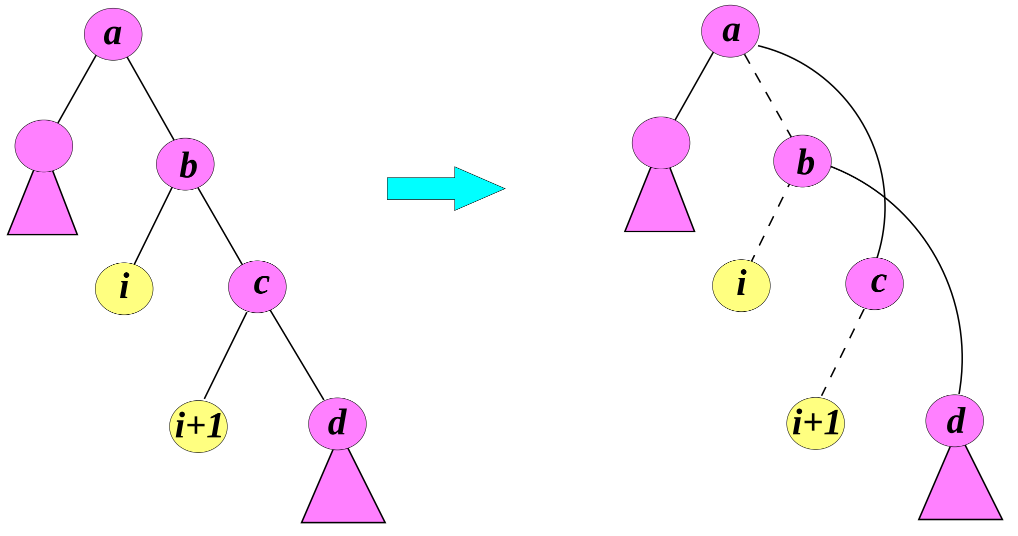
▷ **Shunt *cannot be performed on two siblings simultaneously***.

Concurrent-Write PRAM and nondeterminism



▷ **Shunt *cannot be applied in parallel to 2 adjacent leaves in left-to-right ordering.***

disconnected tree

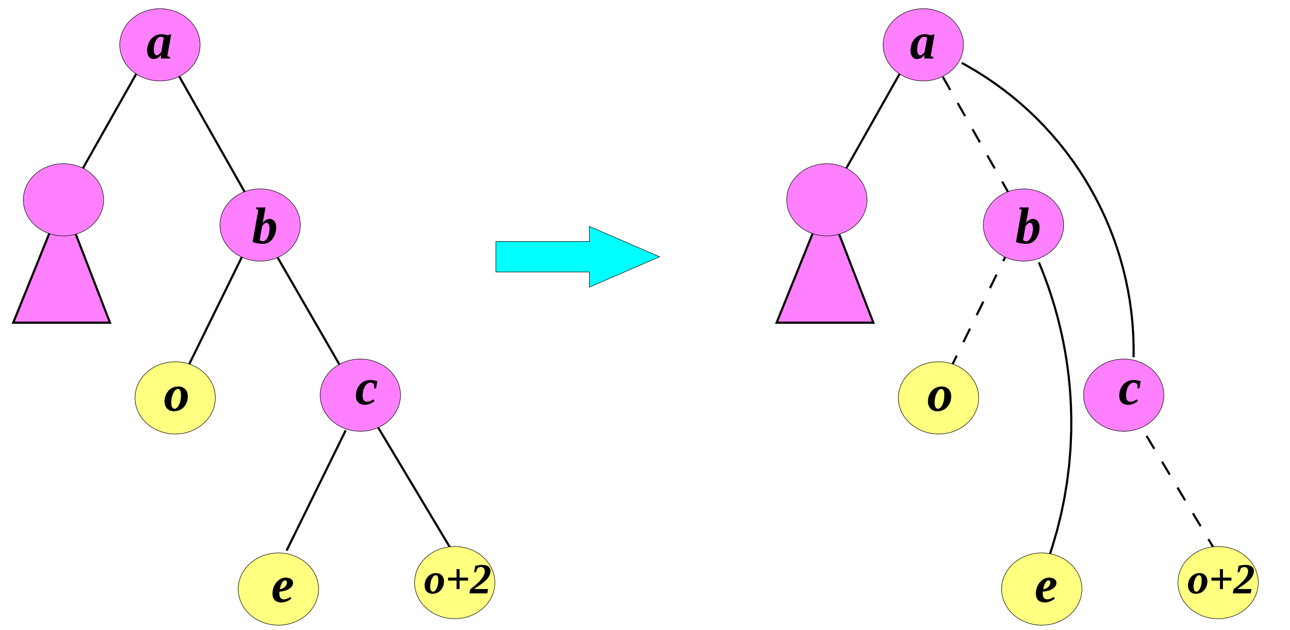


Solution?

apply **Shunt** to ***odd-numbered leaves*** first and to ***even-numbered leaves*** then.

▷ **Shunt *cannot be applied to consecutive left and right odd-numbered leaves.***

disconnected tree + nondeterminism + Concurrent-Write



Left-Right numbering of leaves

Input: $EA'[1,\ldots,m]$;
Auxiliary: $IsLeaf[1,\ldots,n]$; /* flags identifying leaves */
Output: $LR\_Numbering[1,\ldots,n]$;

for all nodes $v \in T$ **do_in_parallel**
  $IsLeaf[v] := 0$;
for all arcs $xy \in EA'$ **do_in_parallel**
  if $rank[xy] = rank[yx] + 1$
    then $\{Weight[xy] := 1;\ IsLeaf[y] := 1\}$ **else** $Weight[xy] := 0$;
apply Parallel Scan on $Weight[1,\ldots,m]$;
for all arcs $xy \in EA'$ **do_in_parallel**
  if $rank[xy] = rank[yx] + 1$
    then $LR\_Numbering[y] := Weight[xy] - 1$ **else** $LR\_Numbering[y] := 0$

Generic Shunt Contract algorithm for binary trees

Input:	$EA'[1,\ldots,m]$; /* Euler array */
	$P[1,\ldots,n]$; /* $P[x]$ is a pointer to the parent of x */
	$side[1,\ldots,n]$; /* $side[v] \in \{L,R\}$ */
	$sibling[1,\ldots,n]$; /* $sibling[v]$ points to the other child of $P[v]$ */
Auxil:	$IsLeaf[1,\ldots,n]$; /* flags identifying leaves */
	$active[1,\ldots,n]$; /* flags keeping track of nodes still in game */
	$LR\_numbering[1,\ldots,n]$; /* Left-to-right numbering of leaves +/
Output:	the value of the reduced tree stored in the root

Procedure Shunt$(v : node)$;
{ **Rake**(v); $active[v] := 0$; $active[P[v]] := 0$;
Compress$(sibling[v])$; $P[sibling[v]] := P[P[v]]$; }

```
/* initialize the data structures */
for all nodes v ∈ T do_in_parallel /* initialize(v); */
call LR_numbering(T);
for all nodes v ∈ T do_in_parallel
 if IsLeaf[v] then active[v] := 1 else active[v] := 0;
repeat log n times
 for all nodes v ∈ T do_in_parallel
   if (v ≠ root and active[v])
     if (Is_Odd(LR_Numbering[v]) and P[v] ≠ root)
        then { if (side[v] = L) then Shunt(v);
             if (side[v] = R) then Shunt(v) };
        else LR_Numbering[v] := LR_Numbering[v]/2;
Rake(root);
```

Performance

Theorem 2 **Shunt Contract** *runs correctly on EREW PRAM.*

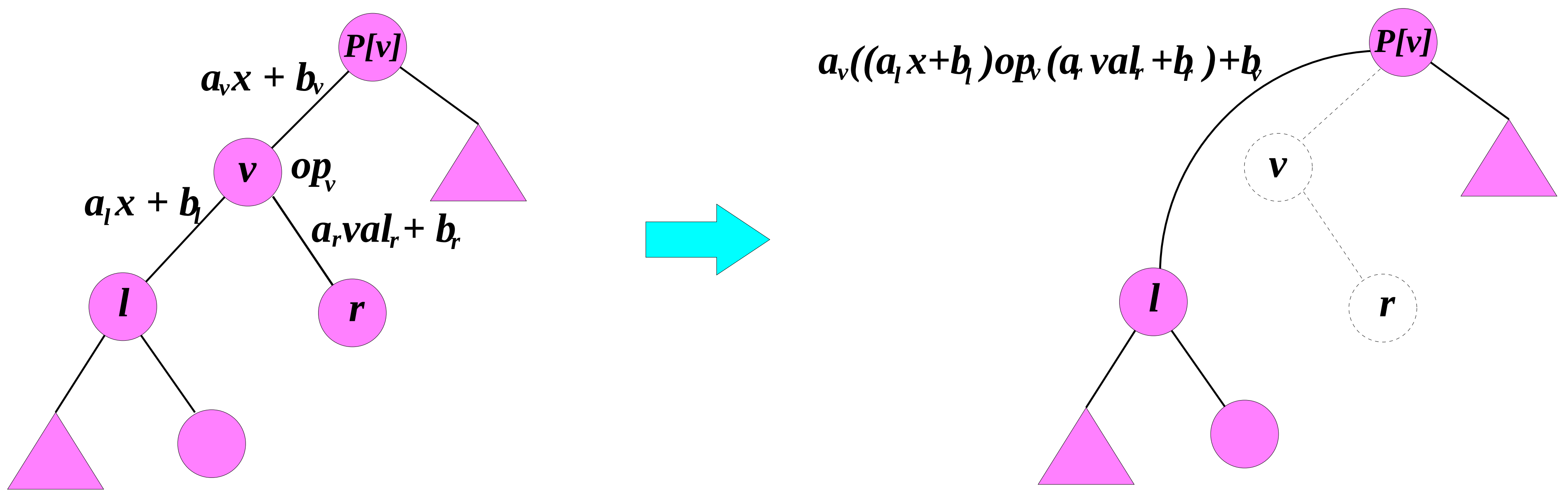
Proof. Let v_1 and v_2 be two nonconsecutive ***left*** leaves of T. Then $P[v_1] \neq P[v_2]$ and $P[v_1] \neq P[P[v_2]]$. It follows from the definition of **Shunt** that no collision can appear. ■

Theorem 3 *If* $p = \Theta(n/\log n)$, *then* $T(n,p) = O(\tau_{\mathsf{Shunt}} \log n)$.

Proof. Assign $\log n/2$ leaves to each processor. One application of **Shunt Contract** eliminates one half of current leaves, so that the total number of parallel **Shunt** operations is at most

$$\log n/2 + \log n/4 + \cdots + \log n/(2^{\log\log n}) + 1 + \cdots + 1 \leq 2\log n.$$

Shunt in an expression tree.



Binary expression tree evaluation using Shunt