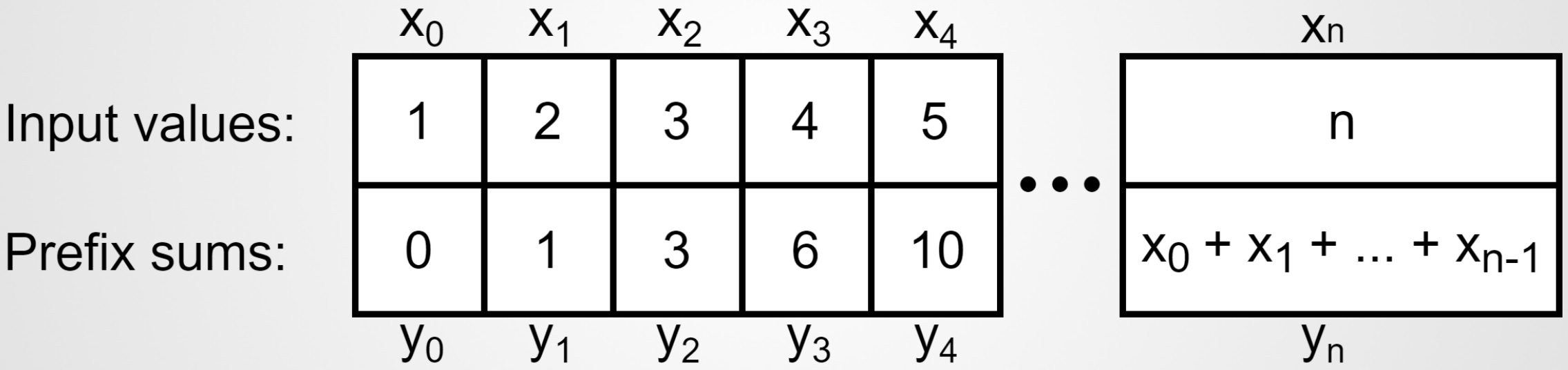
Verification of the Parallel Prefix Sum with VerCors

Thijs Wiefferink (s1366564)
Capita Selecta MTV

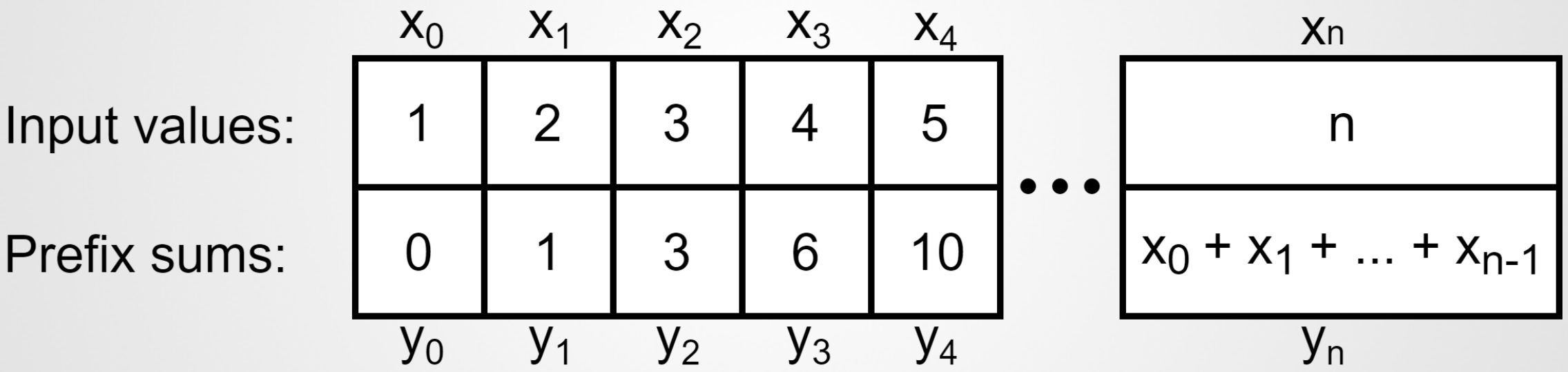
Verification

Verification

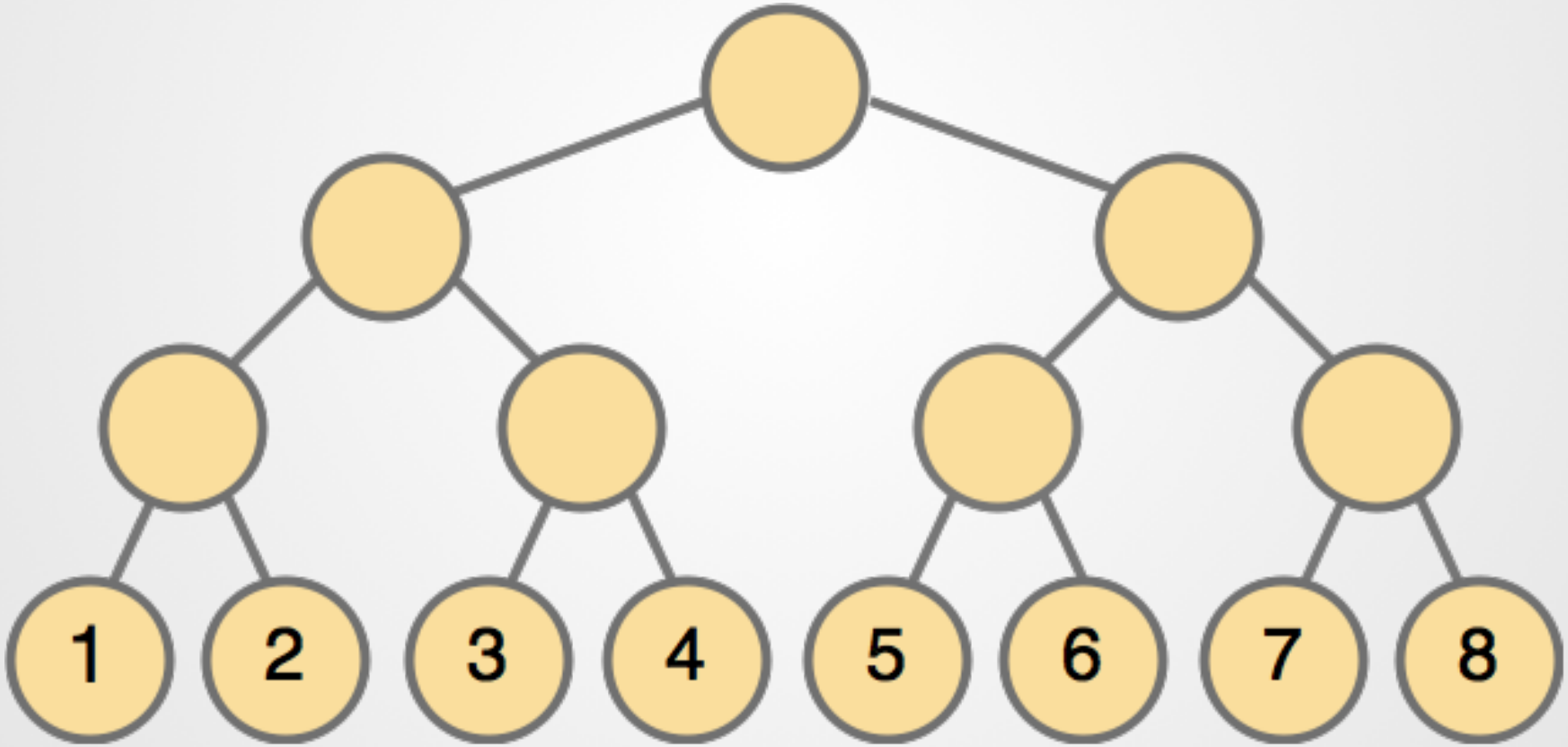
Prefix Sum


x_0 x_1 x_2 x_3 x_4 x_n
Input values:
1 2 3 4 5 n
Prefix sums:
0 1 3 6 10 $x_0 + x_1 + ... + x_{n-1}$
y_0 y_1 y_2 y_3 y_4 y_n

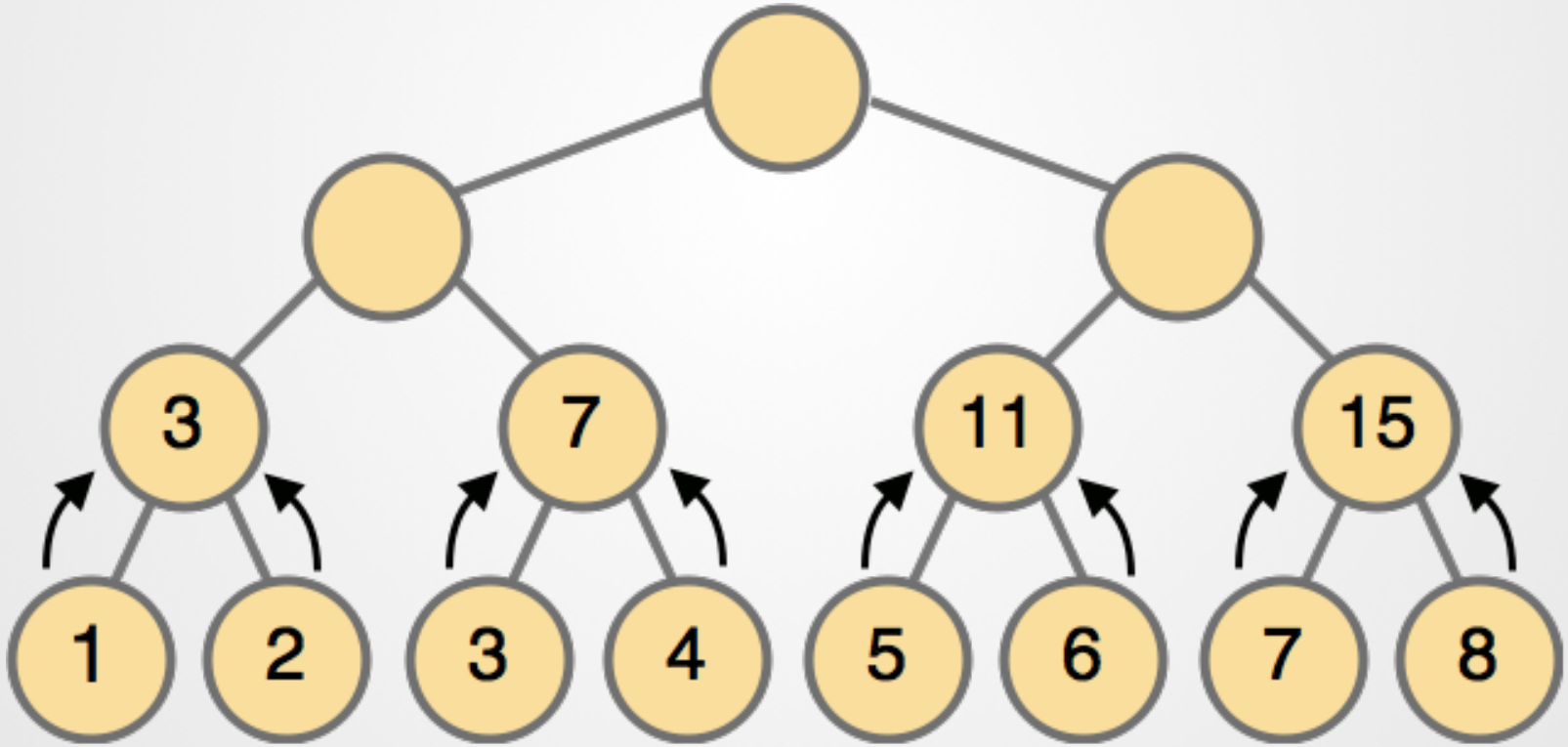
Calculate Prefix Sum on the CPU



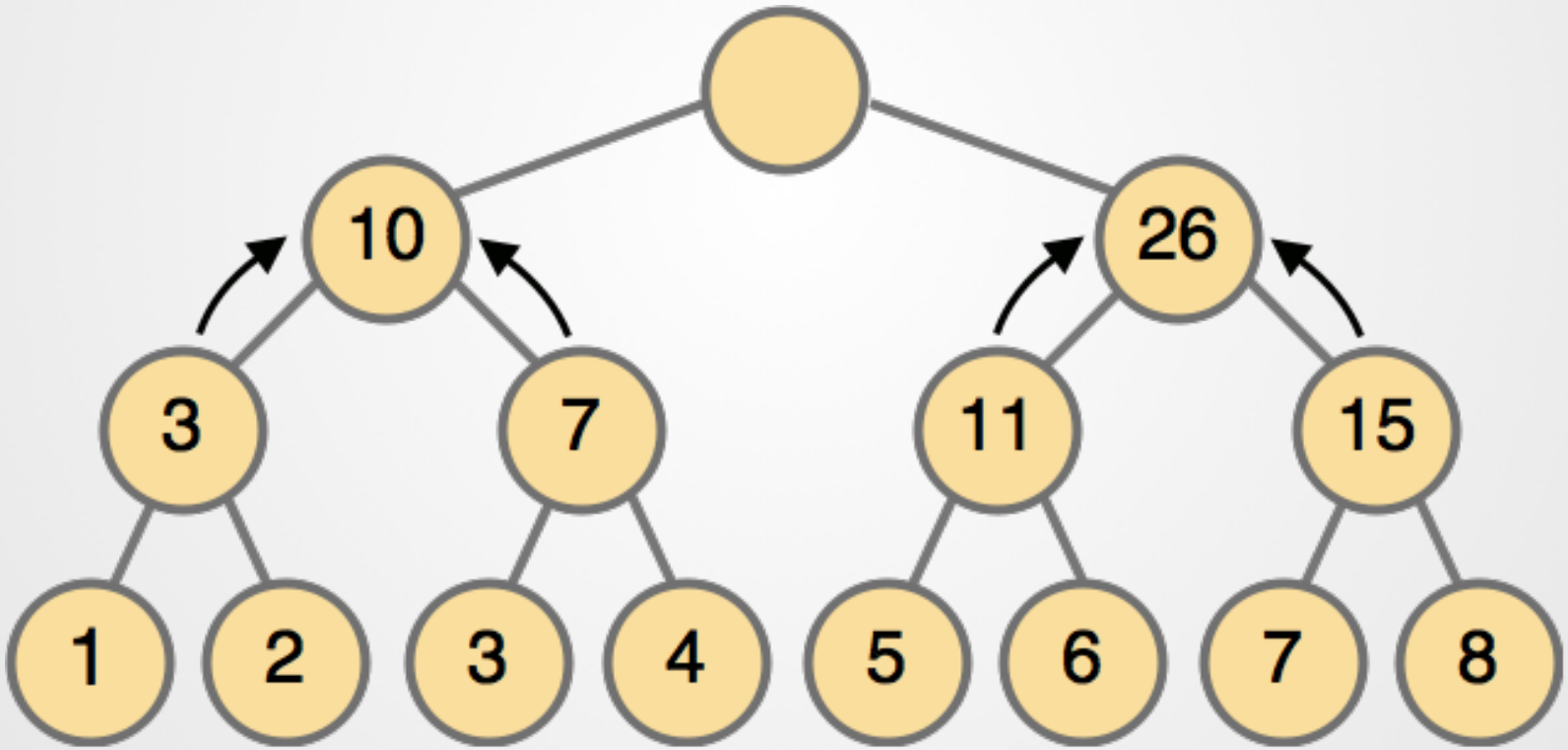
Calculate Prefix Sum on the GPU



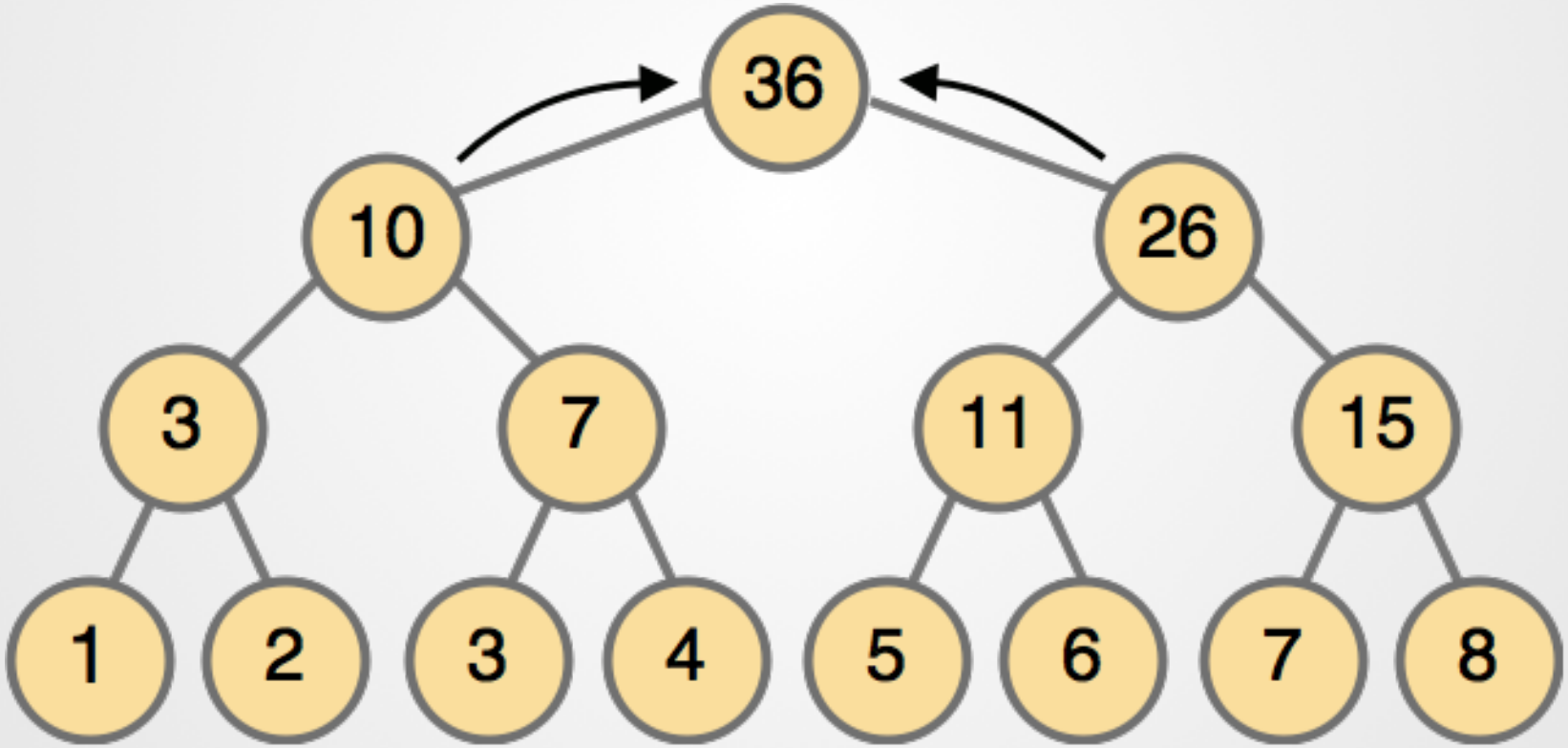
Calculate Prefix Sum on the GPU



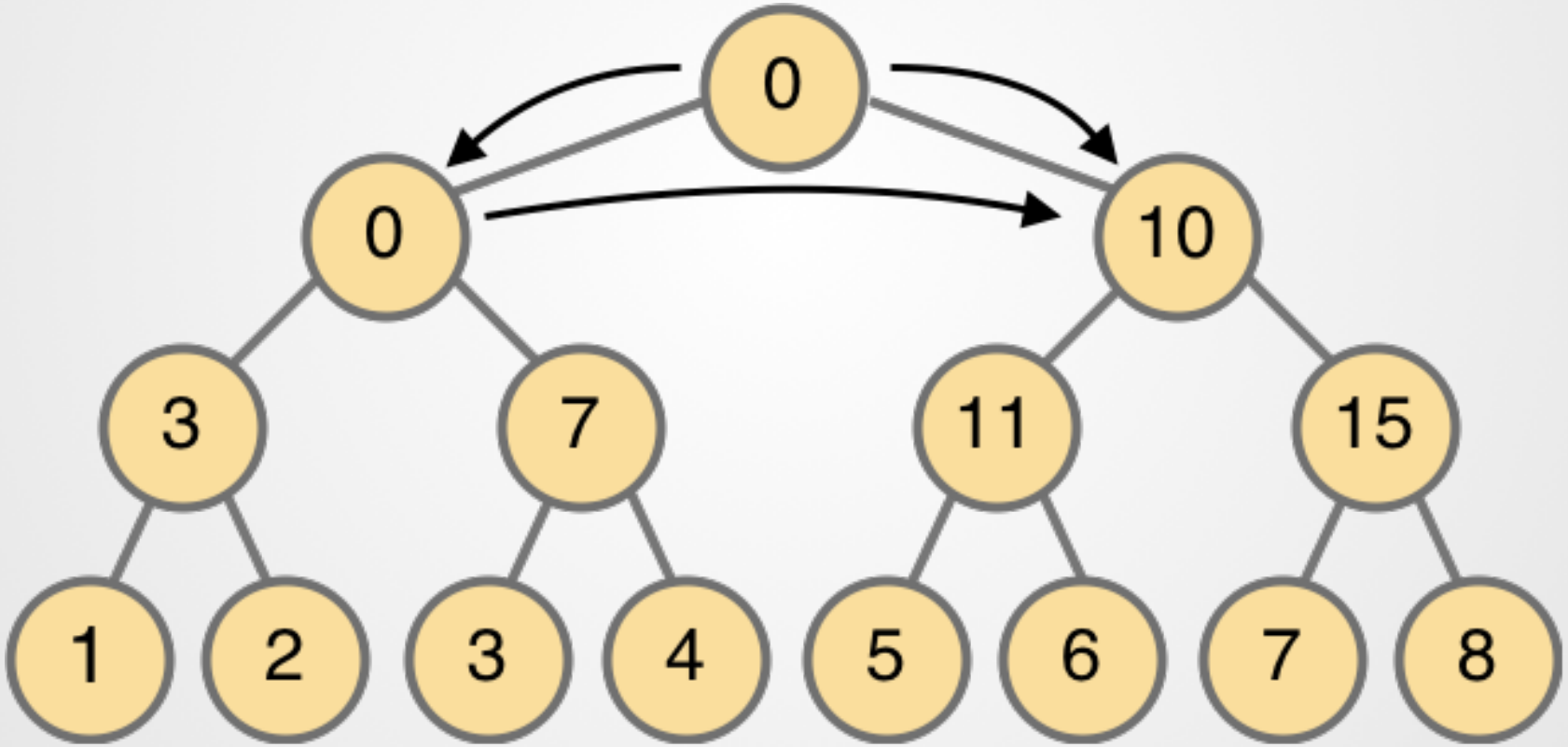
Calculate Prefix Sum on the GPU



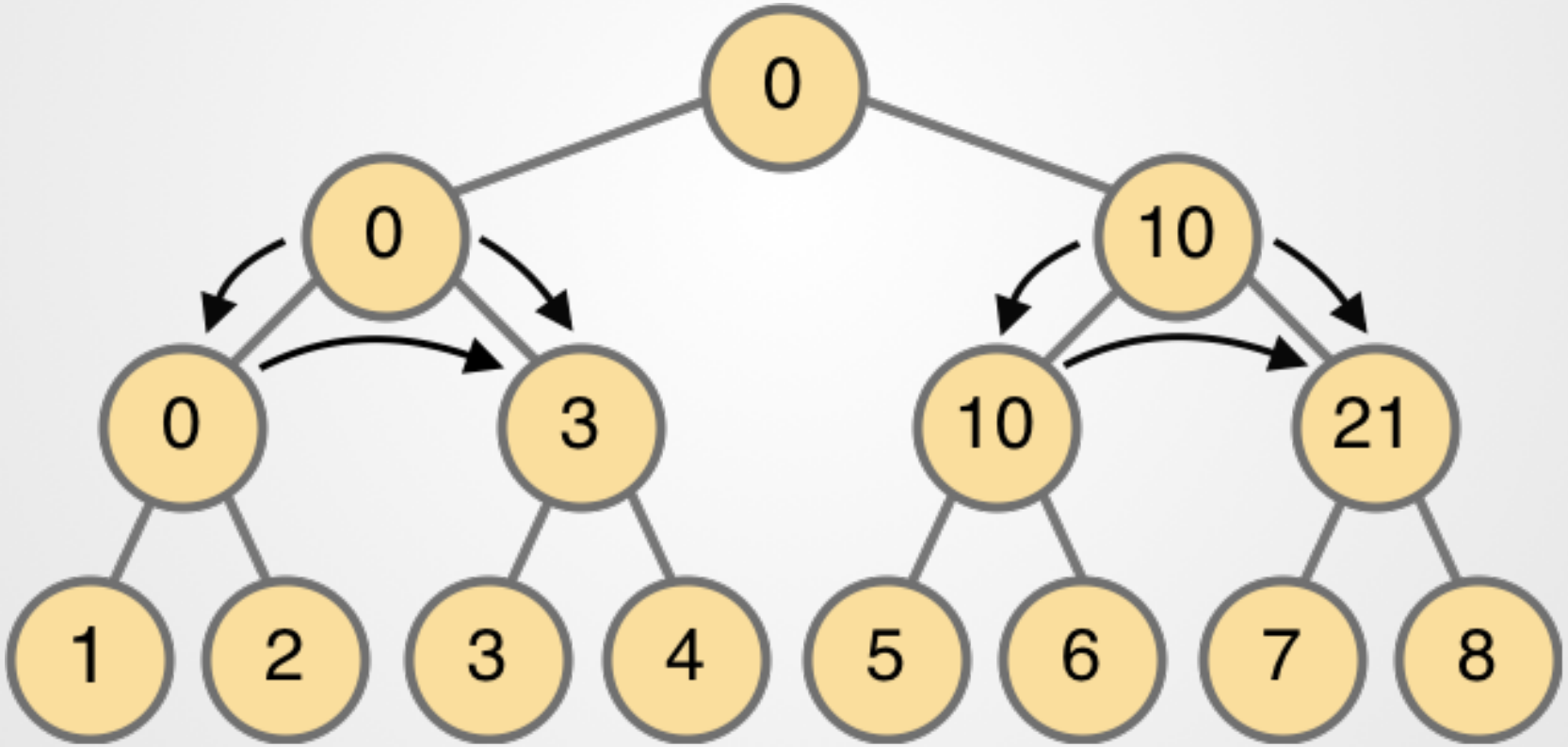
Calculate Prefix Sum on the GPU



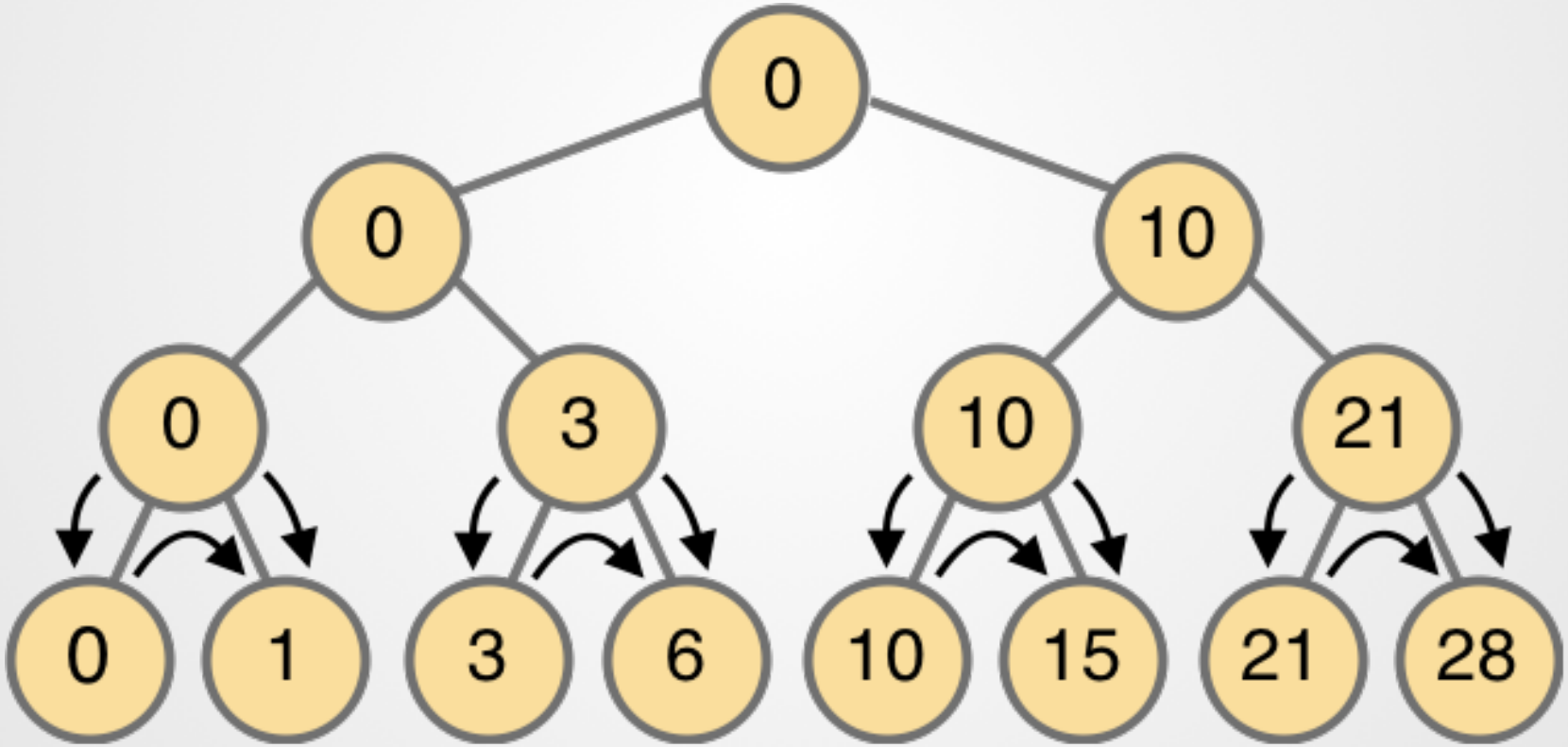
Calculate Prefix Sum on the GPU



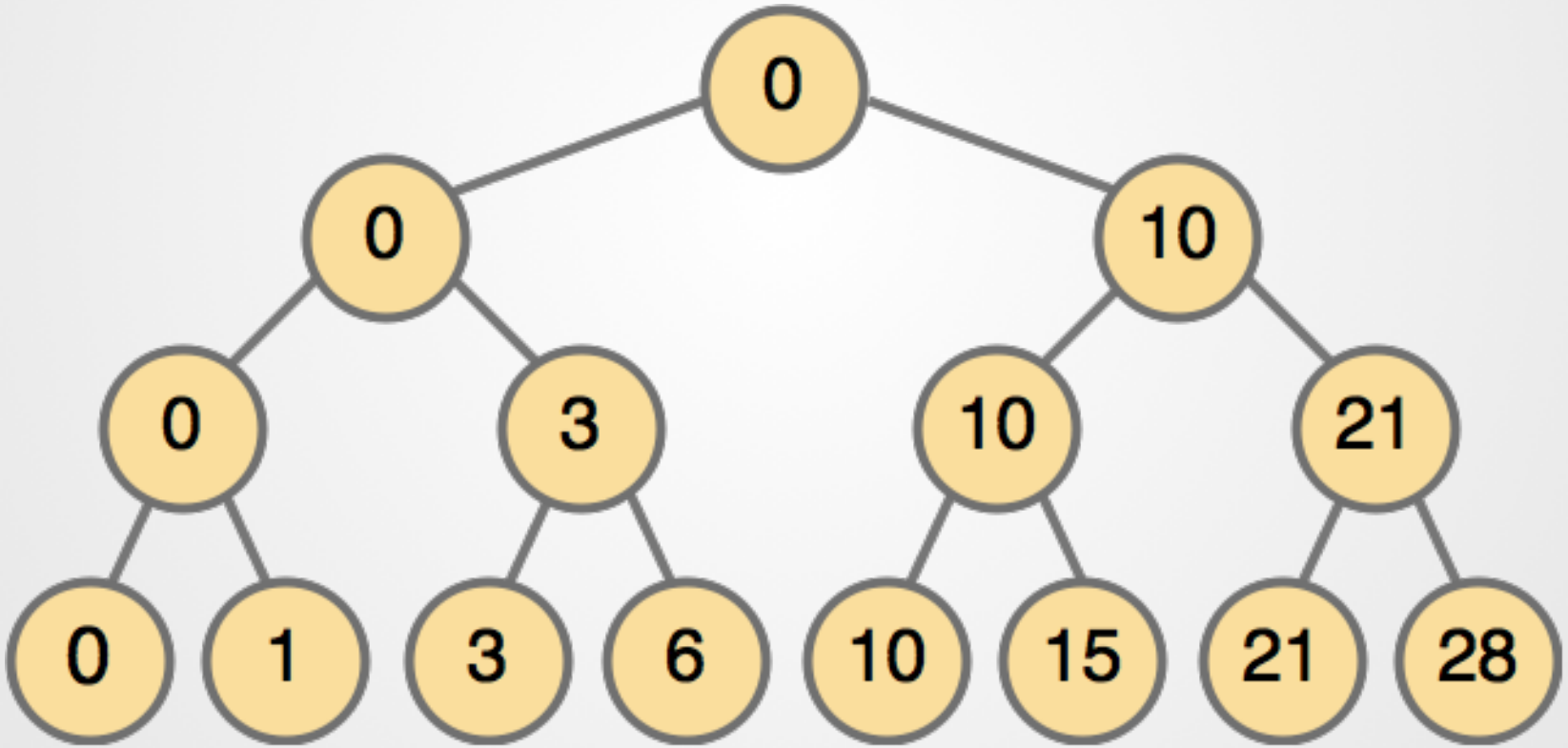
Calculate Prefix Sum on the GPU



Calculate Prefix Sum on the GPU



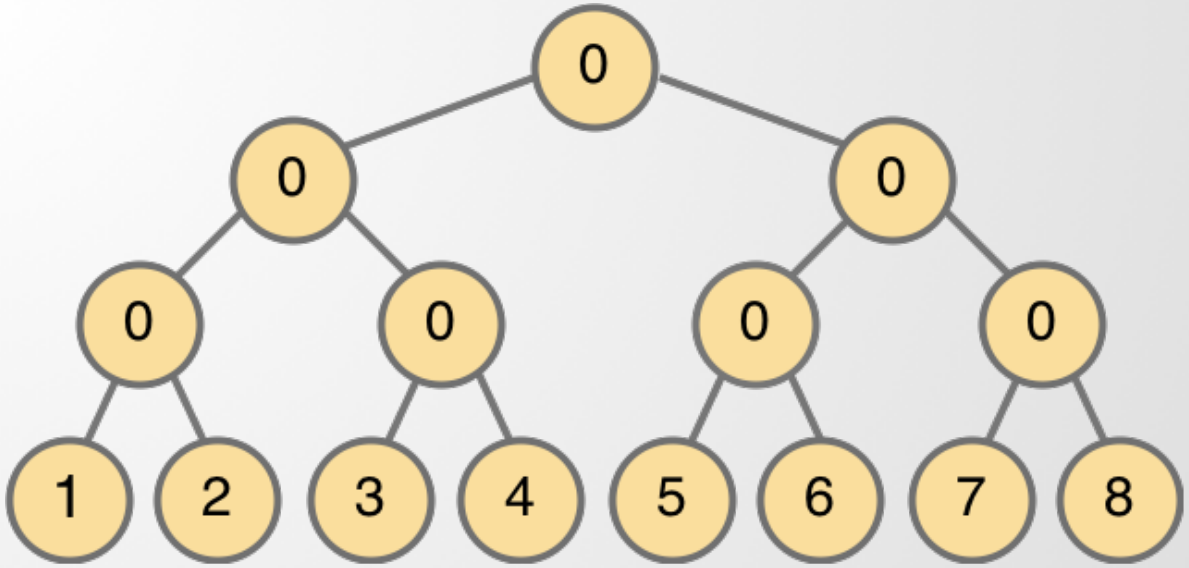
Calculate Prefix Sum on the GPU



Verification: Permissions

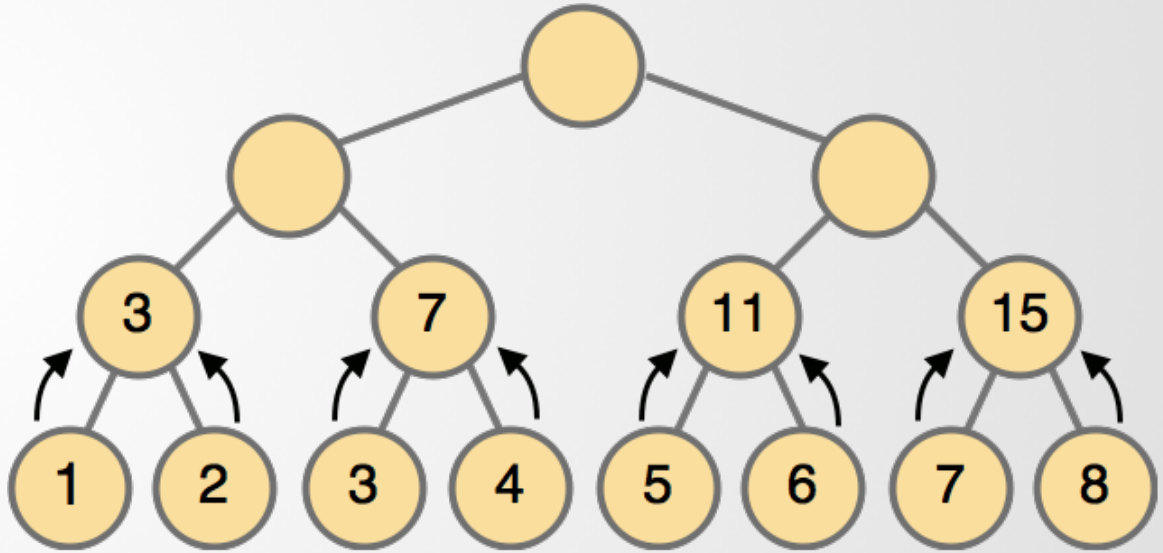
t0	t1	t2	t3	t0	t1	t2	t3
0	1	2	3	4	5	6	7

3	t0	t1	t2	t3	t0	t1	t2	t3
2	t0	t1	t2	t3	t0	t1	t2	t3
1	t0	t1	t2	t3	t0	t1	t2	t3
0	t0	t1	t2	t3	t0	t1	t2	t3
	0	1	2	3	4	5	6	7



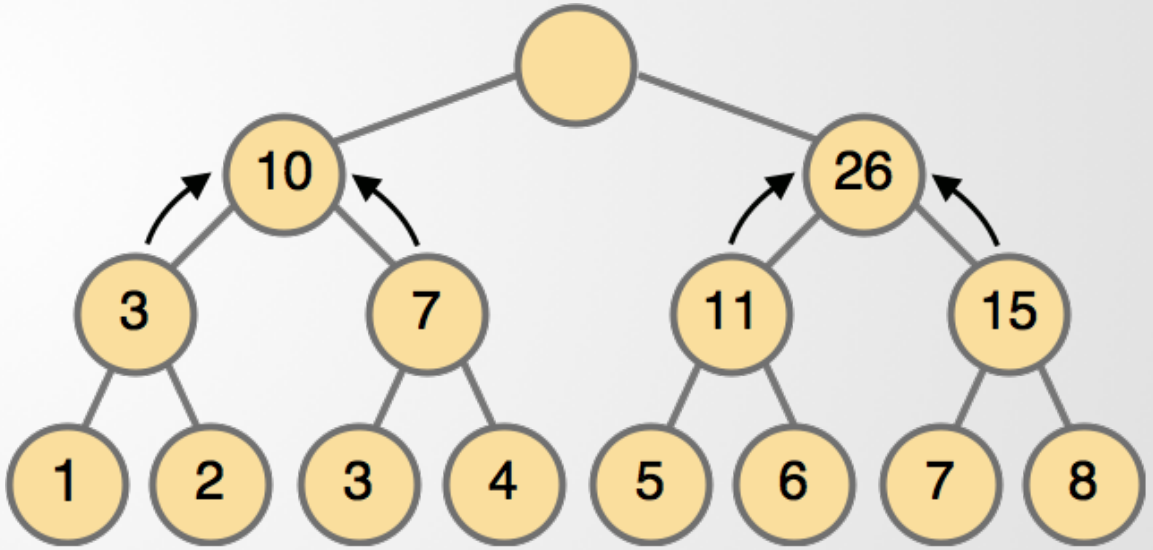
Verification: Permissions

3	t0	t1	t2	t3	t0	t1	t2	t3
2	t0	t1	t2	t3	t0	t1	t2	t3
1	t0	t1	t2	t3	t0	t1	t2	t3
0	t0	t0	t1	t1	t2	t2	t3	t3
	0	1	2	3	4	5	6	7



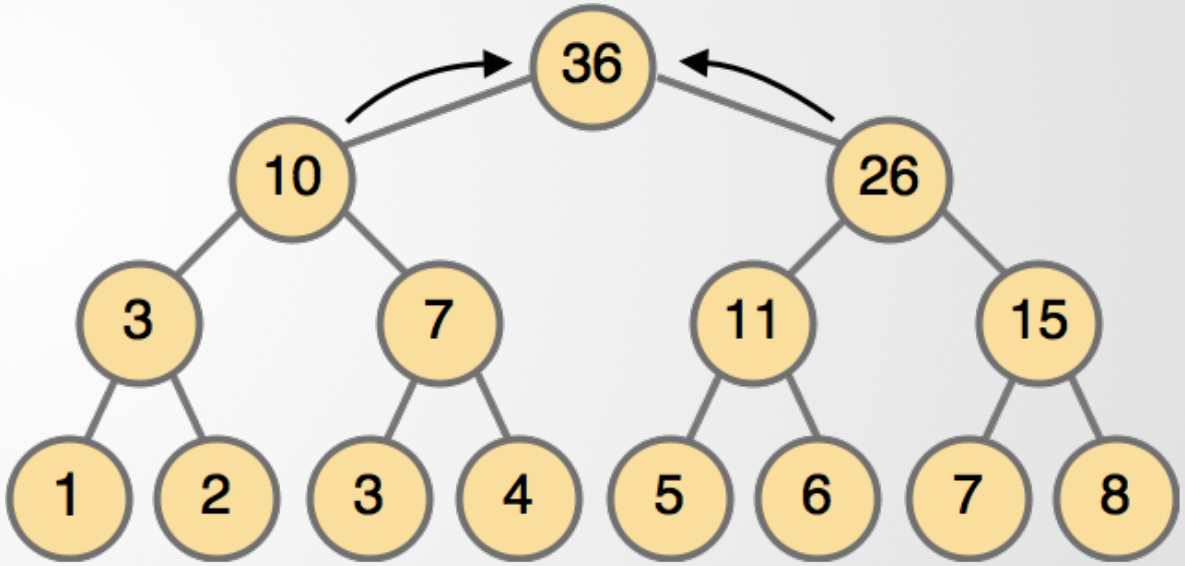
Verification: Permissions

3	t0	t1	t2	t3	t0	t1	t2	t3
2	t0	t1	t2	t3	t0	t1	t2	t3
1	t0	t0	t1	t1	t2	t2	t3	t3
0	t0	t0	t1	t1	t2	t2	t3	t3
	0	1	2	3	4	5	6	7



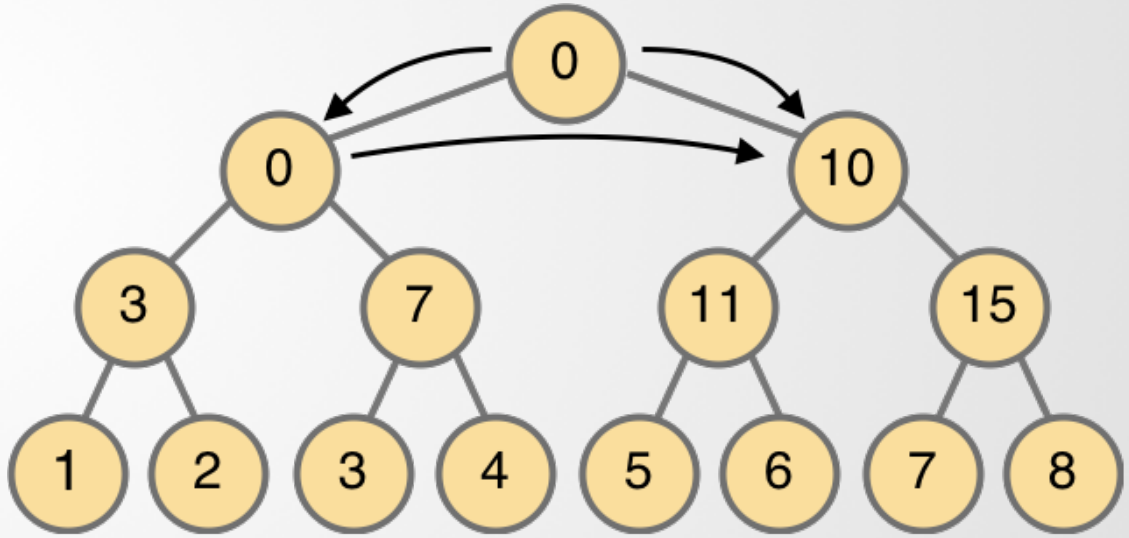
Verification: Permissions

3	t0	t1	t2	t3	t0	t1	t2	t3
2	t0	t0	t1	t1	t2	t2	t3	t3
1	t0	t0	t1	t1	t2	t2	t3	t3
0	t0	t0	t1	t1	t2	t2	t3	t3
	0	1	2	3	4	5	6	7



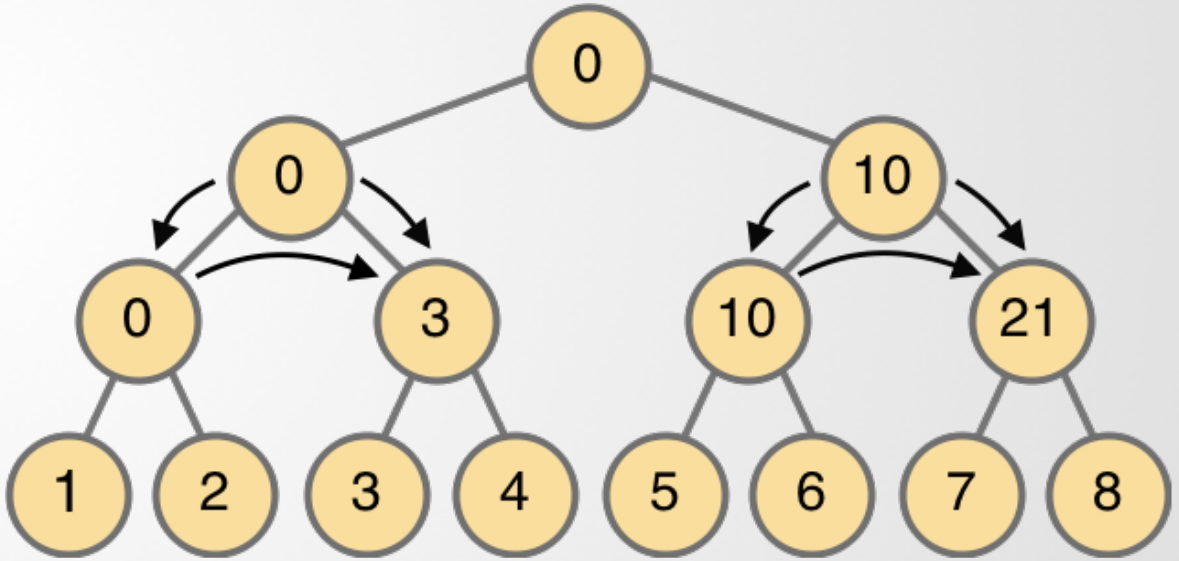
Verification: Permissions

3	t0	t1	t2	t3	t0	t1	t2	t3
2	t0	t1	t2	t3	t0	t1	t2	t3
1	t0	t0	t1	t1	t2	t2	t3	t3
0	t0	t0	t1	t1	t2	t2	t3	t3
	0	1	2	3	4	5	6	7



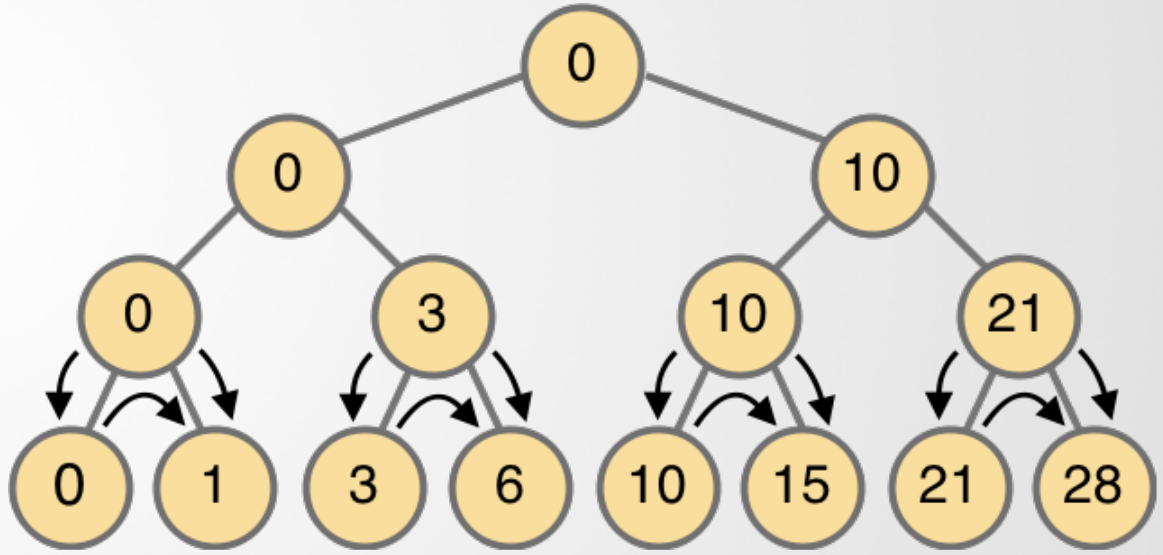
Verification: Permissions

3	t0	t1	t2	t3	t0	t1	t2	t3
2	t0	t1	t2	t3	t0	t1	t2	t3
1	t0	t1	t2	t3	t0	t1	t2	t3
0	t0	t0	t1	t1	t2	t2	t3	t3
	0	1	2	3	4	5	6	7



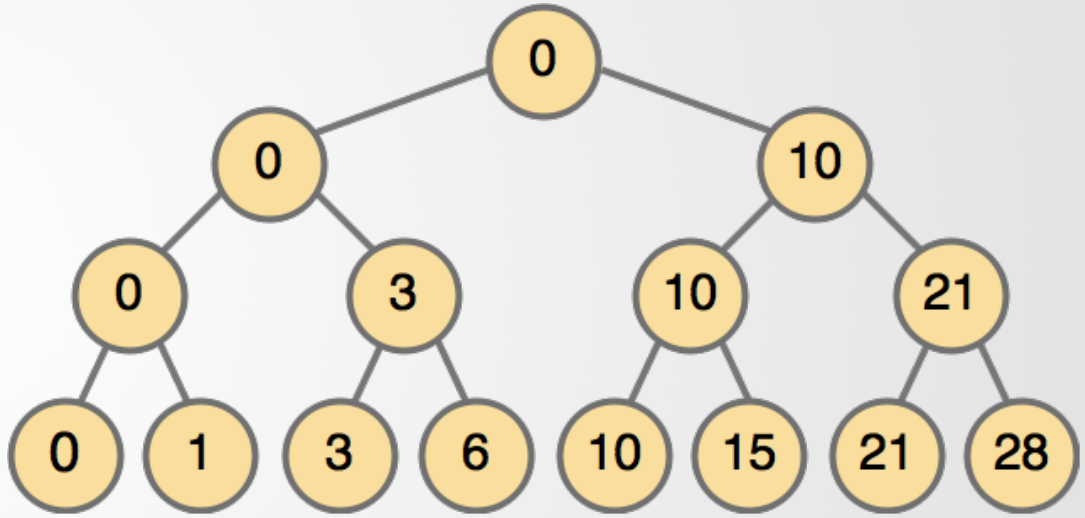
Verification: Permissions

3	t0	t1	t2	t3	t0	t1	t2	t3
2	t0	t1	t2	t3	t0	t1	t2	t3
1	t0	t1	t2	t3	t0	t1	t2	t3
0	t0	t1	t2	t3	t0	t1	t2	t3
	0	1	2	3	4	5	6	7



Verification: Permissions

3	t0	t1	t2	t3	t0	t1	t2	t3
2	t0	t1	t2	t3	t0	t1	t2	t3
1	t0	t1	t2	t3	t0	t1	t2	t3
0	t0	t1	t2	t3	t0	t1	t2	t3
	0	1	2	3	4	5	6	7



t0	t1	t2	t3	t0	t1	t2	t3
0	1	2	3	4	5	6	7

Specification: Permissions

Kernel declaration

```
class Ref {
        invariant N==16 && H==6; // H = log(N)/log(2) + 2, N should be a power of 2
        invariant \matrix(tree,H,N*2);
        invariant \array(input,N*2);
        invariant \array(output,N*2);
        context (\forall* int i; 0<=i && i<N*2; Perm(input[i], read));
        context (\forall* int i; 0<=i && i<N*2; Perm(output[i], write));
        context (\forall* int i; 0<=i && i<N*2;
                (\forall* int j; 0<=j && j<H; Perm(tree[j][i], write))
        );
        void runKernel(int N, int H, int[N*2] input, int[N*2] output, int[H][N*2] tree) {
...
```

Specification: Permissions

```
void runKernel(int N, int H, int[N*2] input, int[N*2] output, int[H][N*2] tree) {
        par runThreads (int t=0..N)
                context Perm(input[t], read);
                context Perm(input[t+N], read);

                context Perm(output[t], write);
                context Perm(output[t+N], write);

                context (\forall* int k; 0<=k && k<2;
                        (\forall* int i; 0<=i && i<H; Perm(tree[i][k*N+t], write)
                );
        {
                ...
        }
}
```

Specification: Permissions

```
tree[0][t] = input[t];
tree[0][t+N] = input[t+N];
```

Specification: Permissions

```
int level=0;
loop_invariant level>=0 && level<H;
loop_invariant t>=0 && t<N;
// Old permisison layout going away, top row excluded: (t1 | t2 | t3 | t4 | ...)
loop_invariant (\forall* int k; 0<=k && k<2;
        (\forall* int i; level<=i && i<H; Perm(tree[i][k*N+t], write))
);
// New permission layout appearing, lowest row already has it: (t1 | t1 | t2 | t2
loop_invariant (\forall* int k; 0<=k && k<2;
        (\forall* int i; 0<=i && i<level; Perm(tree[i][t*2+k], write))
);
while((level+1)<H) {
    ...
}
```

Specification: Permissions

```
while((level+1)<H) {
        level = level+1;
        barrier(runThreads) {
                context t>=0 && t<N;
                context level>=1 && level<H;

                requires Perm(tree[level-1][t], write);
                requires Perm(tree[level-1][t+N], write);

                ensures (\forall* int k; 0<=k && k<2; Perm(tree[level-1][t*2+k], w
        }
        tree[level][t] = tree[level-1][t*2] + tree[level-1][t*2+1];
}
```

Specification: Permissions

```
tree[H-1][t] = 0; // Set the root of the tree to zero

loop_invariant level>=0 && level<H;
loop_invariant t>=0 && t<N;
// Old permission layout coming back, top row already has it: (t1 | t2 | t3 | t4 |
loop_invariant (\forall* int k; 0<=k && k<2;
        (\forall* int i; level<=i && i<H; Perm(tree[i][k*N+t], write))
);
// New permission layout going away, bottom row excluded: (t1 | t1 | t2 | t2 | ...
loop_invariant (\forall* int k; 0<=k && k<2;
        (\forall* int i; 0<=i && i<level; Perm(tree[i][t*2+k], write))
);
while(level>0) {
            ...
}
```

Specification: Permissions

```
while(level>0) {
        tree[level-1][t*2+1] = tree[level-1][t*2] + tree[level][t];      // Right c
        tree[level-1][t*2] = tree[level][t];                             // Left ch

        level = level-1;

        barrier(runThreads) {
                context t>=0 && t<N;
                context level>=0 && level<H;

                requires Perm(tree[level][t*2], write);
                requires Perm(tree[level][t*2+1], write);

                ensures Perm(tree[level][t], write);
                ensures Perm(tree[level][t+N], write);
        }
}
```

Specification: Permissions

```
output[t] = tree[0][t];
output[t+N] = tree[0][t+N];
```

Specification: Permissions

Complete

```
class Ref {
        // KERNEL
        invariant N==16 && H==6; // H = log(N)/log(2) + 2, N should be a power of 2
        invariant \matrix(tree,H,N*2);
        invariant \array(input,N*2);
        invariant \array(output,N*2);
        context (\forall* int i; 0<=i && i<N*2; Perm(input[i], read));
        context (\forall* int i; 0<=i && i<N*2; Perm(output[i], write));
        context (\forall* int i; 0<=i && i<N*2;
                (\forall* int j; 0<=j && j<H; Perm(tree[j][i], write))
        );
        void runKernel(int N, int H, int[N*2] input, int[N*2] output, int[H][N*2] tree) {

                /////////// THREADS
                par runThreads (int t=0..N)
                        context Perm(input[t], read);
                        context Perm(input[t+N], read);

                        context Perm(output[t], write);
                        context Perm(output[t+N], write);

                        context (\forall* int k; 0<=k && k<2;
                                (\forall* int i; 0<=i && i<H; Perm(tree[i][k*N+t], write))
```

Specification: Functionality

```
ensures (
    \forall* int i; 0<=i && i<N*2;
        output[i] == \sum([0 ..i), inpu
);
```

Specification: Functionality

1. Bottom row contains the input
2. Updated partial sums for each upsweep step
3. The highest row is zero
4. Updated partial sums for each downsweep step
5. Output array contains prefix sums

Specification: Functionality

```
loop_invariant (\forall* int i; 0<=i && i<N*2; tree[0
```

Specification: Functionality

```
loop_invariant level>0 ==> tree[level][t] == tree[level-1][t
    tree[level-1][t*2+1];
```

```
((t+1)*2^level < N*2) => tree[level][t] ==
    \sum([t*2^level..(t+1)*2^level), tree[0]);
```

Specification: Functionality

```
loop_invariant tree[H-1][t] == 0
```

Specification: Functionality

```
loop_invariant tree[level-1][t*2+1] ==
        \old(tree[level-1][t*2]) +
        tree[level][t];
loop_invariant tree[level-1][t*2] == tree[level]
```

```
loop_invariant (t*2^(level-1) < N*2) => tree[level-1][t*2]
    \sum([0..t*2^(level-1)], tree[0]);
loop_invariant (t*2^(level-1) < N*2) => tree[level-1][t*2+1
    \sum([0..t*2^(level-1)+1], tree[0]);
```

Specification: Functionality

```
output[t] == \sum([0 ..t), input);
output[t+N] == \sum([0 ..t+N), input);
```

```
ensures (
    \forall* int i; 0<=i && i<N*2;
        output[i] == \sum([0 ..i), inpu
);
```

VerCors Performance

Until specification part	**Time**
Input copy	7882ms
Upsweep	12745ms
Downsweep	92351ms
Downsweep with thread ensures	1934471ms
Downsweep with thread and kernel ensures	2190647ms

Results

- Permissions verified
- \context
- \invariant
- VerCors performance
- VerCors consistency

Future work

- Functional specification and verification
- Investigate performance:
  - Setup some simple benchmarks
  - Track times for each commit