ECE408 Fall 2022

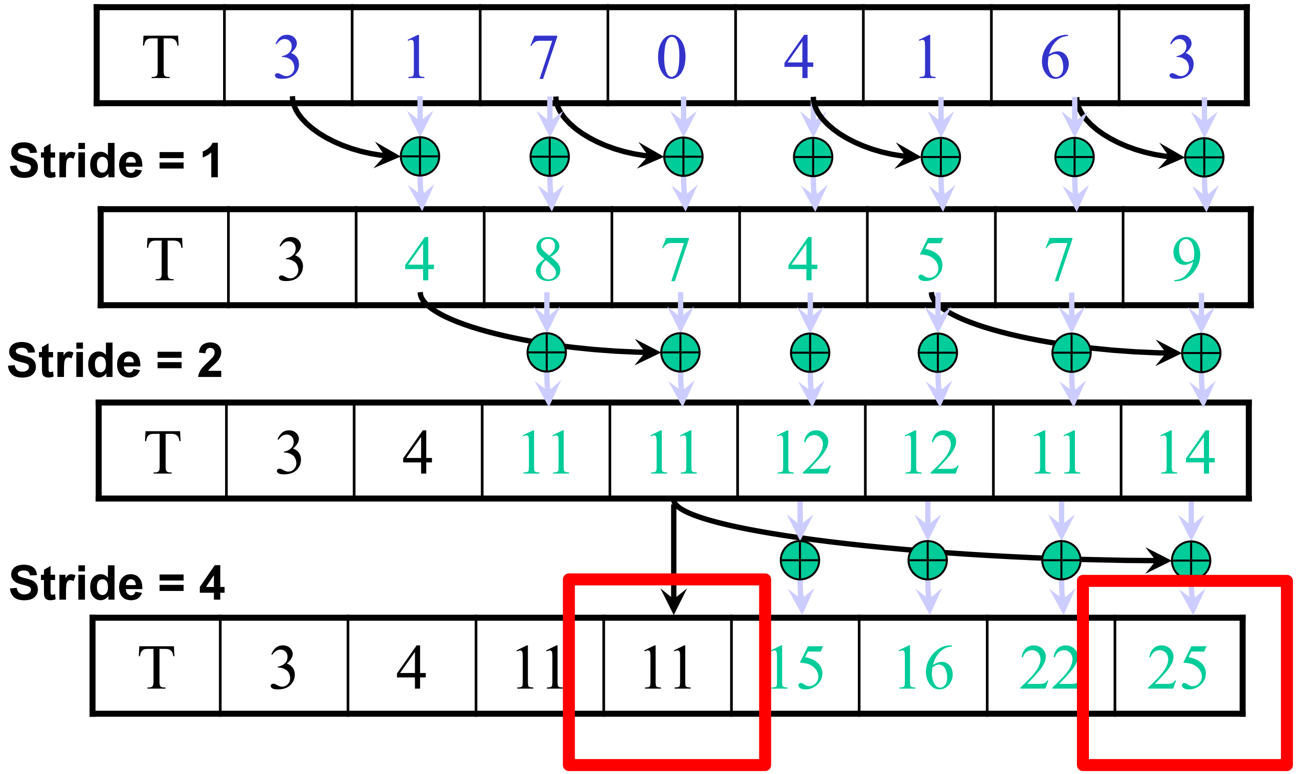
Applied Parallel Programming

Lecture 17
Parallel Scan Part-2

Objective

- To master parallel scan (prefix sum) algorithms
  - Work-efficiency vs. latency
  - Brent-Kung Tree Algorithm
  - Hierarchical algorithms

A Kogge-Stone Parallel Scan Algorithm



Improving Efficiency

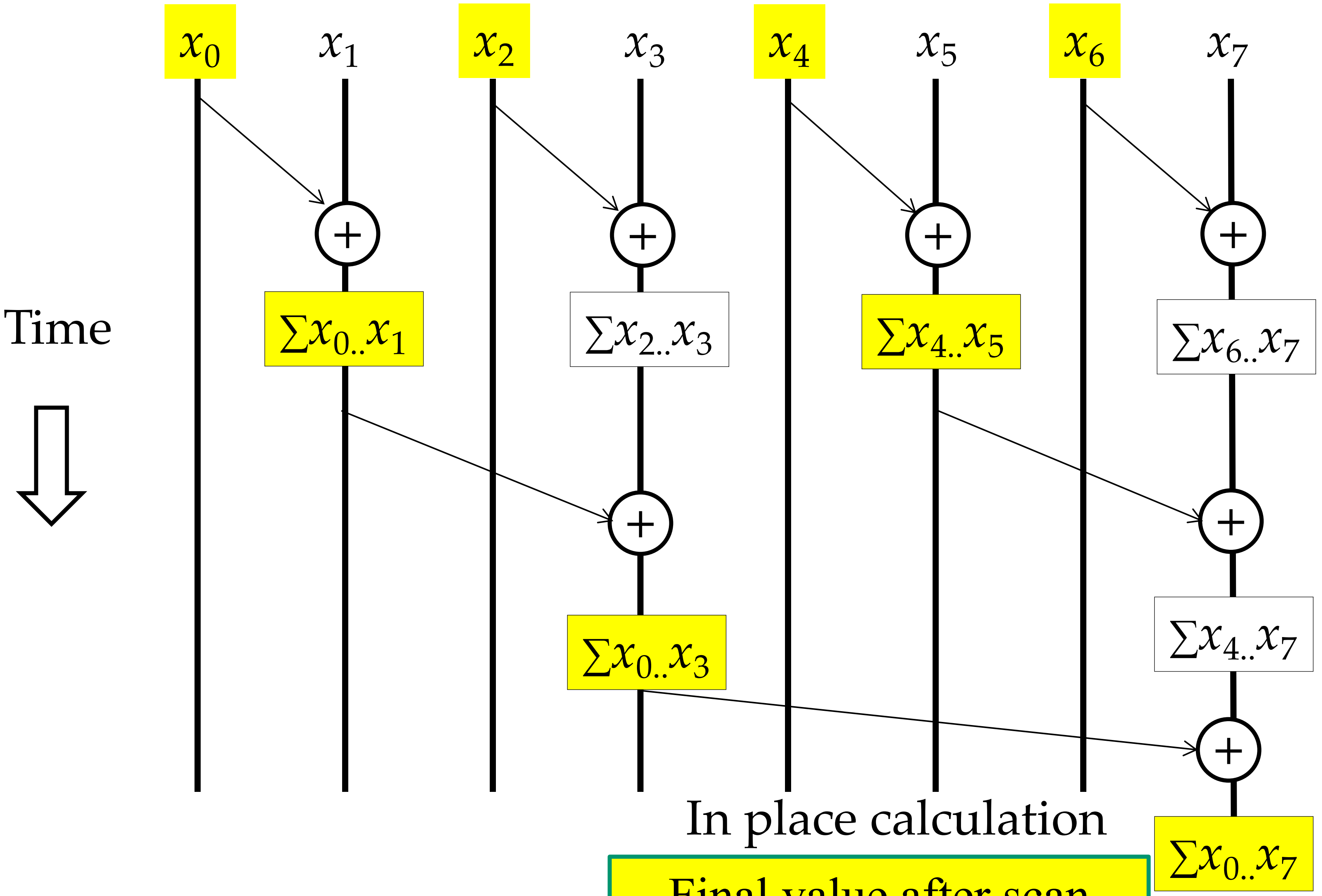
- A common parallel algorithm pattern:

  Balanced Trees

  – Build a balanced binary tree on the input data and sweep it to and from the root
  – Tree is not an actual data structure, but a conceptual pattern

- For scan:
  – Traverse down from leaves to root building partial sums at internal nodes in the tree
    - Root holds sum of all leaves
  – Traverse back up the tree building the scan from the partial sums

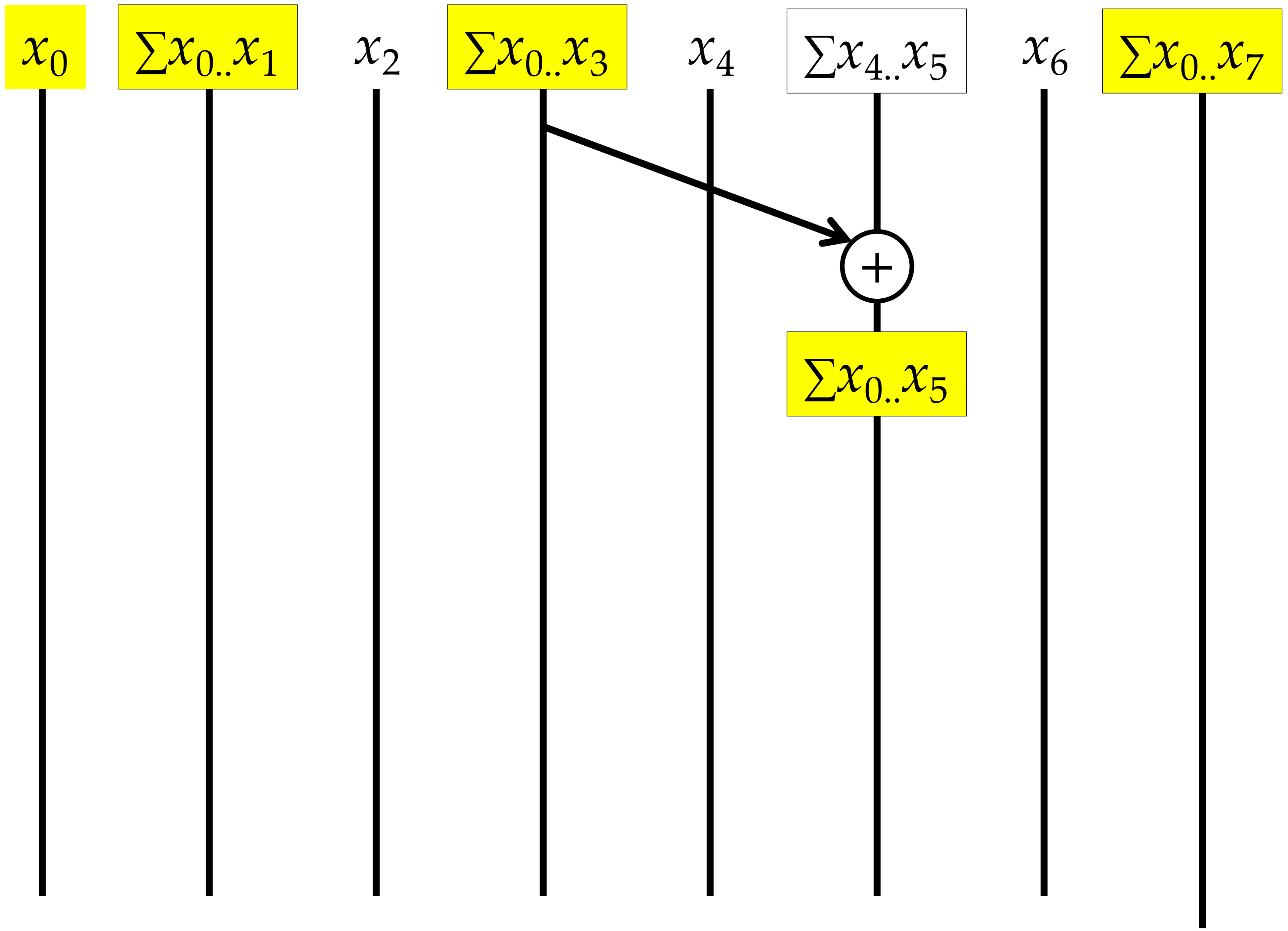
Brent-Kung Parallel Scan Step



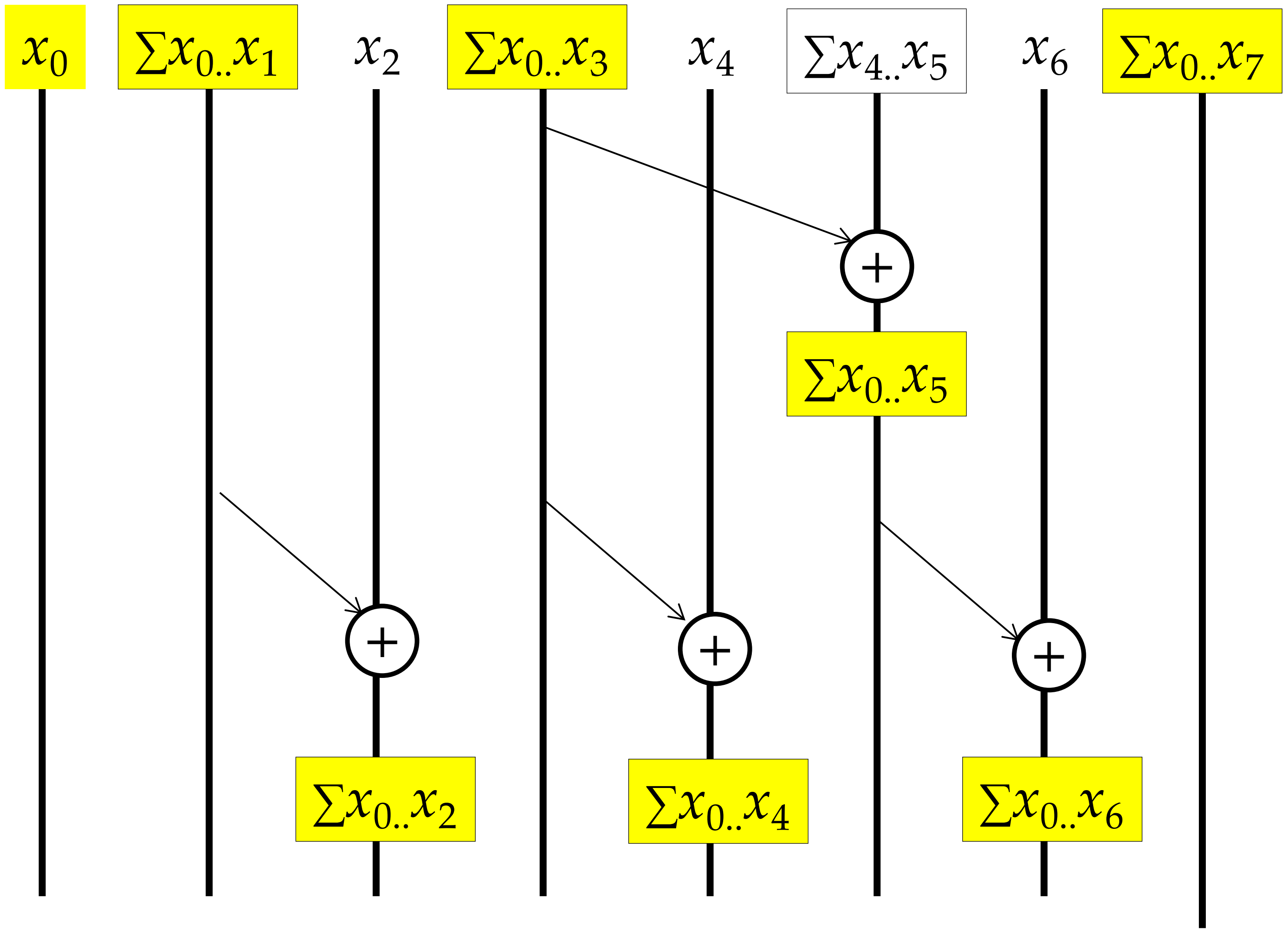
In place calculation

Final value after scan

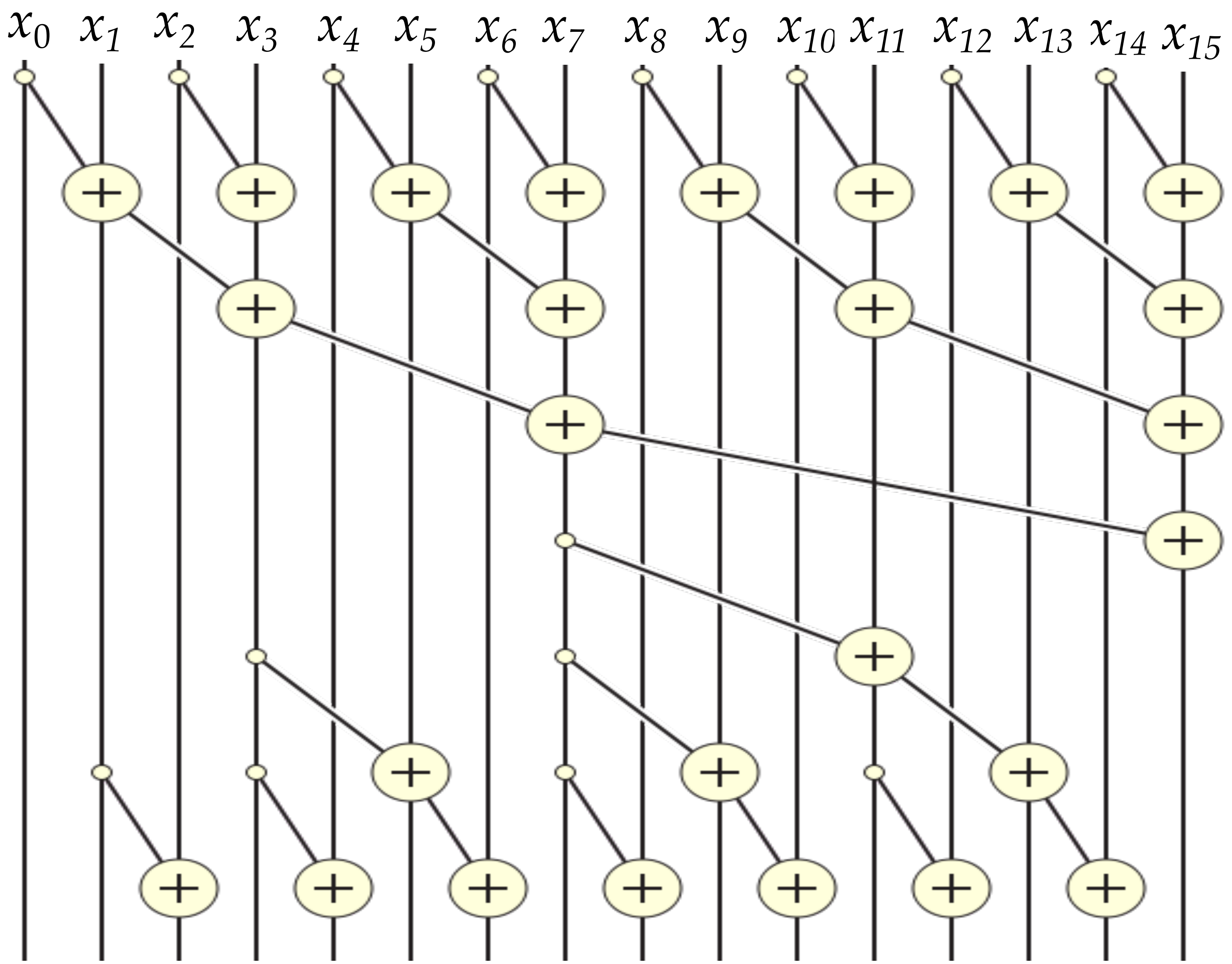
Post-Scan Step



Post Scan Step



Putting it Together (Data View)

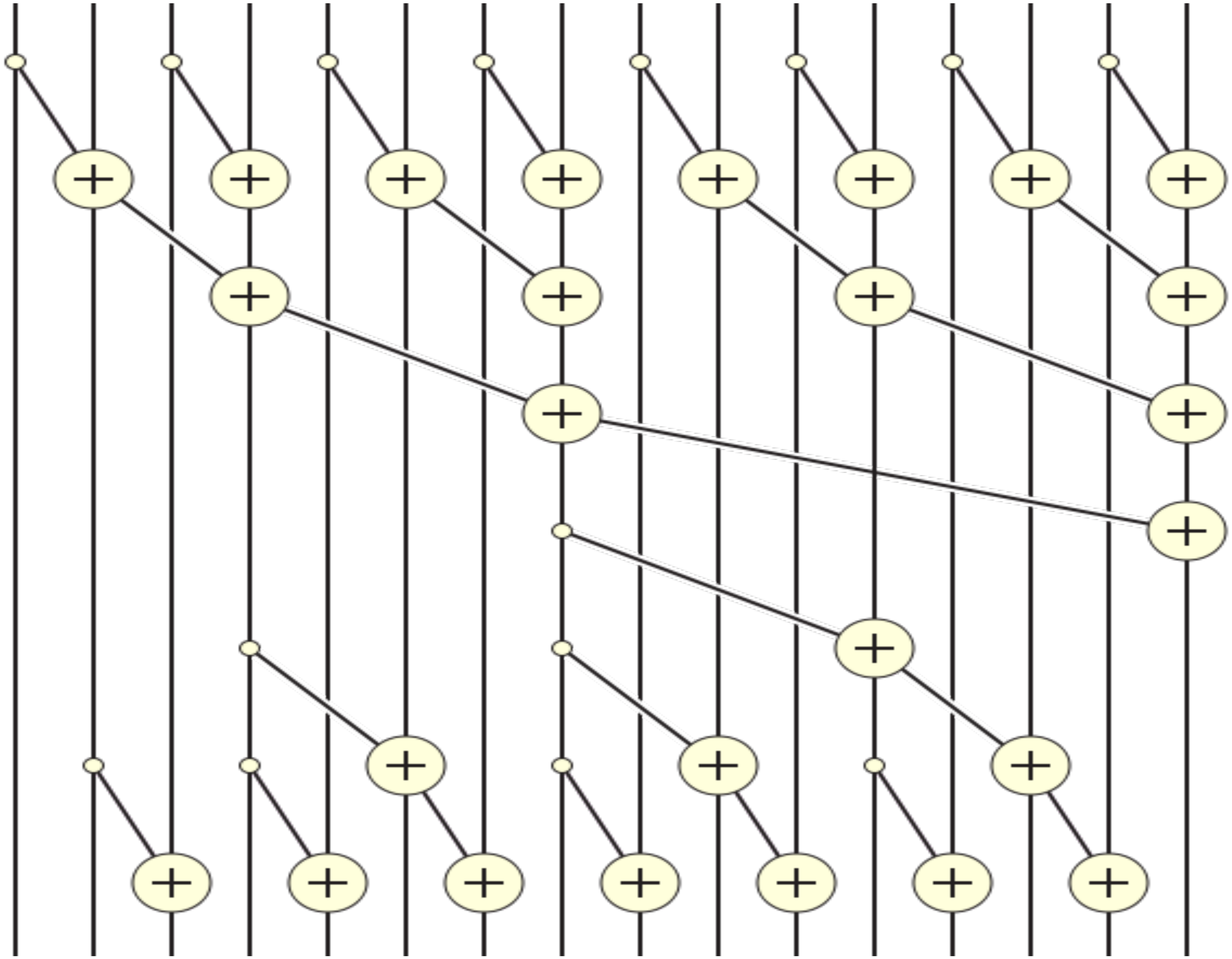


Scan Step Kernel Code

```
// float T[2*BLOCK_SIZE] is in shared memory
// for previous slide, BLOCK_SIZE is 8
int stride = 1;
while(stride < 2*BLOCK_SIZE) {
        __syncthreads();
        int index = (threadIdx.x+1)*stride*2 - 1;
        if(index < 2*BLOCK_SIZE && (index-stride) >= 0)
            T[index] += T[index-stride];
        stride = stride*2;
}
```

```
// In our example,
// threadIdx.x+1    = 1, 2, 3, 4, 5, 6, 7, 8
// stride = 1, index = 1, 3, 5, 7, 9, 11, 13, 15
```

Putting it Together



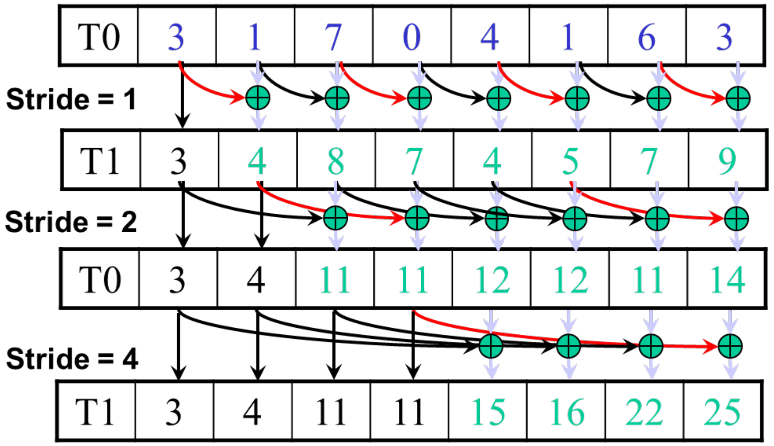
Post Scan Step

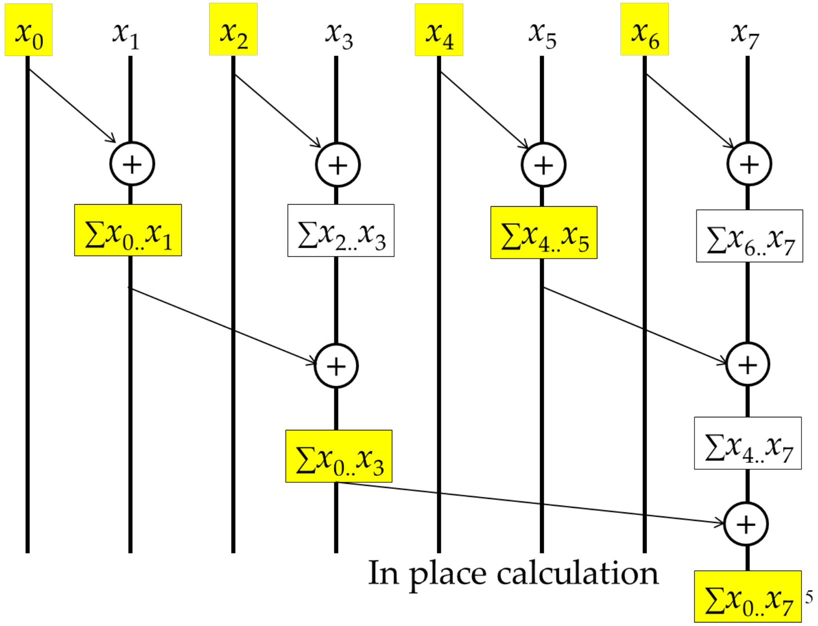
```
int stride = BLOCK_SIZE/2;
while(stride > 0) {
    __syncthreads();
    int index = (threadIdx.x+1)*stride*2 - 1;
    if ((index+stride) < 2*BLOCK_SIZE)
        T[index+stride] += T[index];
    stride = stride / 2;
}

// In our example,
// BLOCK_SIZE=8 stride=4, 2, 1
// for first iteration, active thread = 0 index = 7, +stride = 11
```

Work Analysis

- The parallel Scan executes 2* log(n) parallel iterations
  - log(n) in reduction and log(n) in post scan
  - The iterations do n/2, n/4,..1, (2-1), …., (n/4-1), (n/2-1) useful adds
  - In our example, n = 16, the number of useful adds is 16/2 + 16/4 + 16/8 + 16/16 + (16/8-1) + (16/4-1) + (16/2-1)
  - Total adds: (n-1) + (n-2) – (log(n) -1) = 2*(n-1) – log(n) → O(n) work
- The total number of adds is no more than twice of that done in the efficient sequential algorithm
  - The benefit of parallelism can easily overcome the 2X work when there is sufficient hardware

Kogge-Stone vs. Brent-Kung

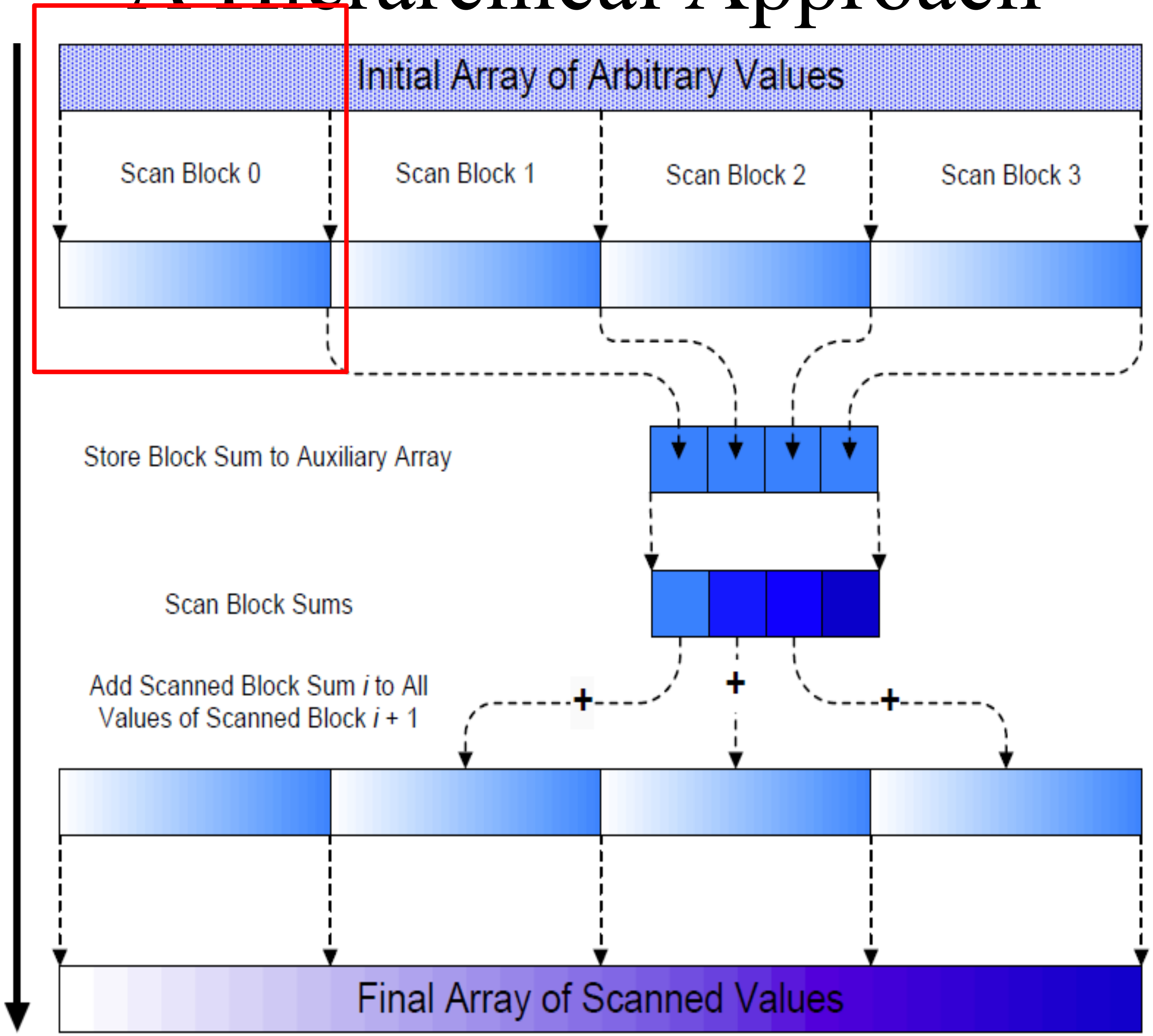




- Brent-Kung uses half the number of threads compared to Kogge-Stone
  - Each thread should load two elements into the shared memory
- Brent-Kung takes twice the number of steps compared to Kogge-Stone
  - Kogge-Stone is more popular for parallel scan with blocks in GPUs
  - Threads in the block are ”occupied” until entire block finishes

Overall Flow of Complete Scan A Hierarchical Approach

Scan kernel



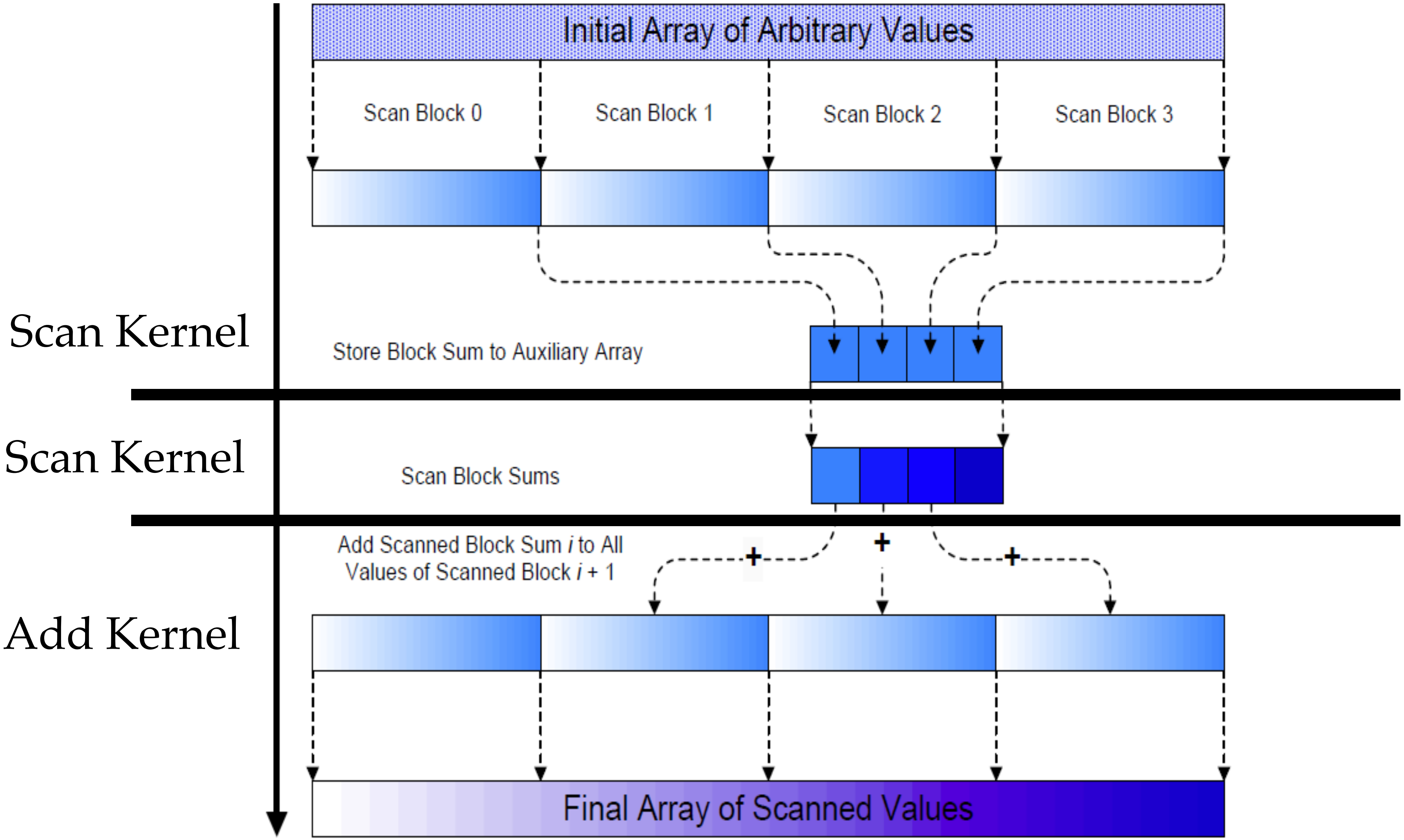
Using Global Memory Contents in CUDA

- Data in registers and shared memory of one thread block are not visible to other blocks
- To make data visible, the data has to be written into global memory
- However, any data written to the global memory are not visible until a memory fence. This is typically done by terminating the kernel execution
- Launch another kernel to continue the execution. The global memory writes done by the terminated kernels are visible to all thread blocks.

Scan of Arbitrary Length Input

- Build on the scan kernel that handles up to 2*blockDim.x elements from Brent-Kung
  - For Kogge-Stone, have each section of blockDim.x elements assigned to a block
- Have each block write the sum of its section into a Sum array using its blockIdx.x as index
- Run parallel scan on the Sum array
  - May need to break down Sum into multiple sections if it is too big for a block
- Add the scanned Sum array values to the elements of corresponding sections

Overall Flow of Complete Scan
A Hierarchical Approach



Exclusive Scan Definition

Definition: *The exclusive* scan *operation takes a binary associative operator* $\oplus$, *and an array of n elements*

$$[x_0, x_1, \ldots, x_{n-1}]$$

and returns the array

$$[0, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-2})].$$

Example: If $\oplus$ is addition, then the exclusive scan operation

on [3 1 7 0 4 1 6 3],

would return [0 3 4 11 11 15 16 22].

Why Exclusive Scan

- To find the beginning address of allocated buffers

- Inclusive and Exclusive scans can be easily derived from each other; it is a matter of convenience

	[3 1 7 0 4 1 6 3]
Exclusive	[0 3 4 11 11 15 16 22]
Inclusive	[3 4 11 11 15 16 22 25]

A simple exclusive scan kernel

- Adapt an inclusive, Kogge-Stone scan kernel
  - Block 0:
    - Thread 0 loads 0 into (shared) XY[0]
    - Other threads load (global) X[threadIdx.x-1] into XY[threadIdx.x]
  - All other blocks:
    - All thread load X[blockIdx.x*blockDim.x+threadIdx.x-1] into XY[threadIdex.x]

- Similar adaption for Brent-Kung kernel but pay attention that each thread loads two elements
  - Only one zero should be loaded
  - All elements should be shifted by only one position

ANY MORE QUESTIONS? READ CHAPTER 8