Notes on Exclusive Scan

Parallel Computer Architecture and Programming
CMU 15-418/15-618, Spring 2017

Data-parallel scan

let $a = [a_0, a_1, a_2, a_3, ..., a_{n-1}]$

let $\oplus$ be an associative binary operator with identity element I

$\texttt{scan\_inclusive}(\oplus, a) = [a_0,\ a_0 \oplus a_1,\ a_0 \oplus a_1 \oplus a_2,\ ...$

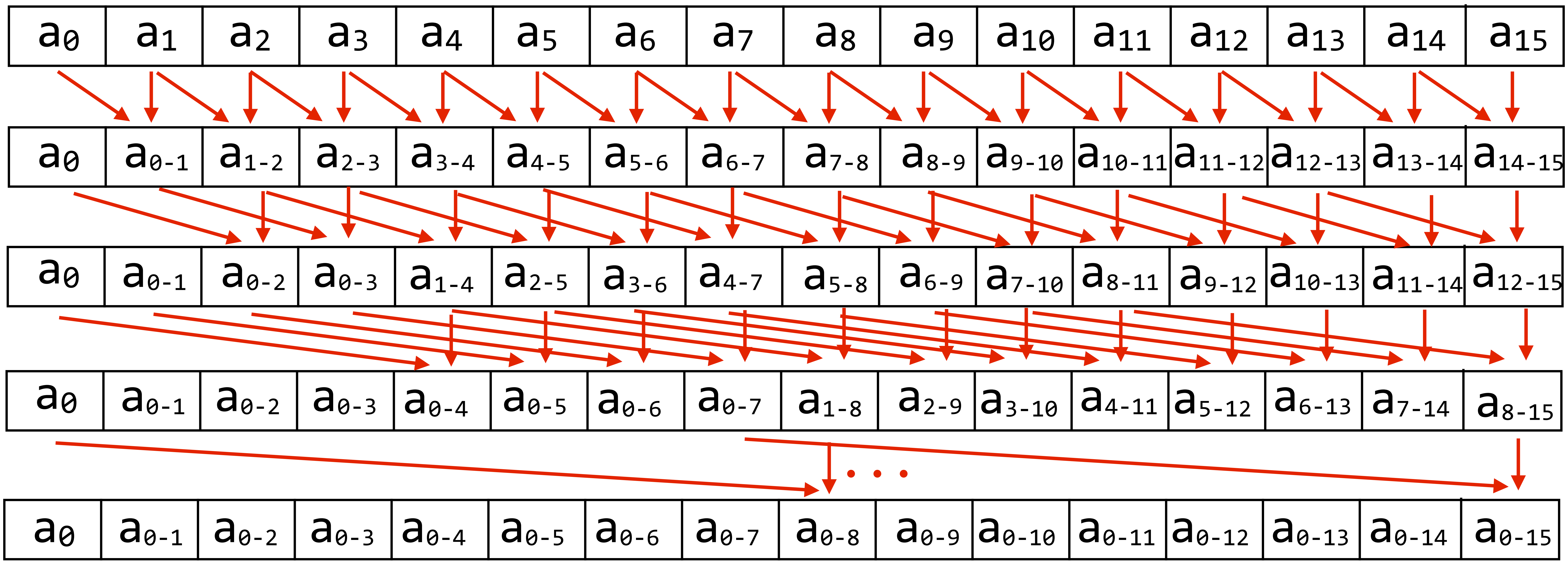
$\texttt{scan\_exclusive}(\oplus, a) = [I,\ a_0,\ a_0 \oplus a_1,\ a_0 \oplus a_1 \oplus a_2,\ ...$

If operator is +, then `scan_inclusive(+,a)` is a prefix sum

$\texttt{prefix\_sum}(a) = [a_0,\ a_0 + a_1,\ a_0 + a_1 + a_2,\ ...$

Data-parallel inclusive scan

(Just subtract original vector to get the exclusive scan result)



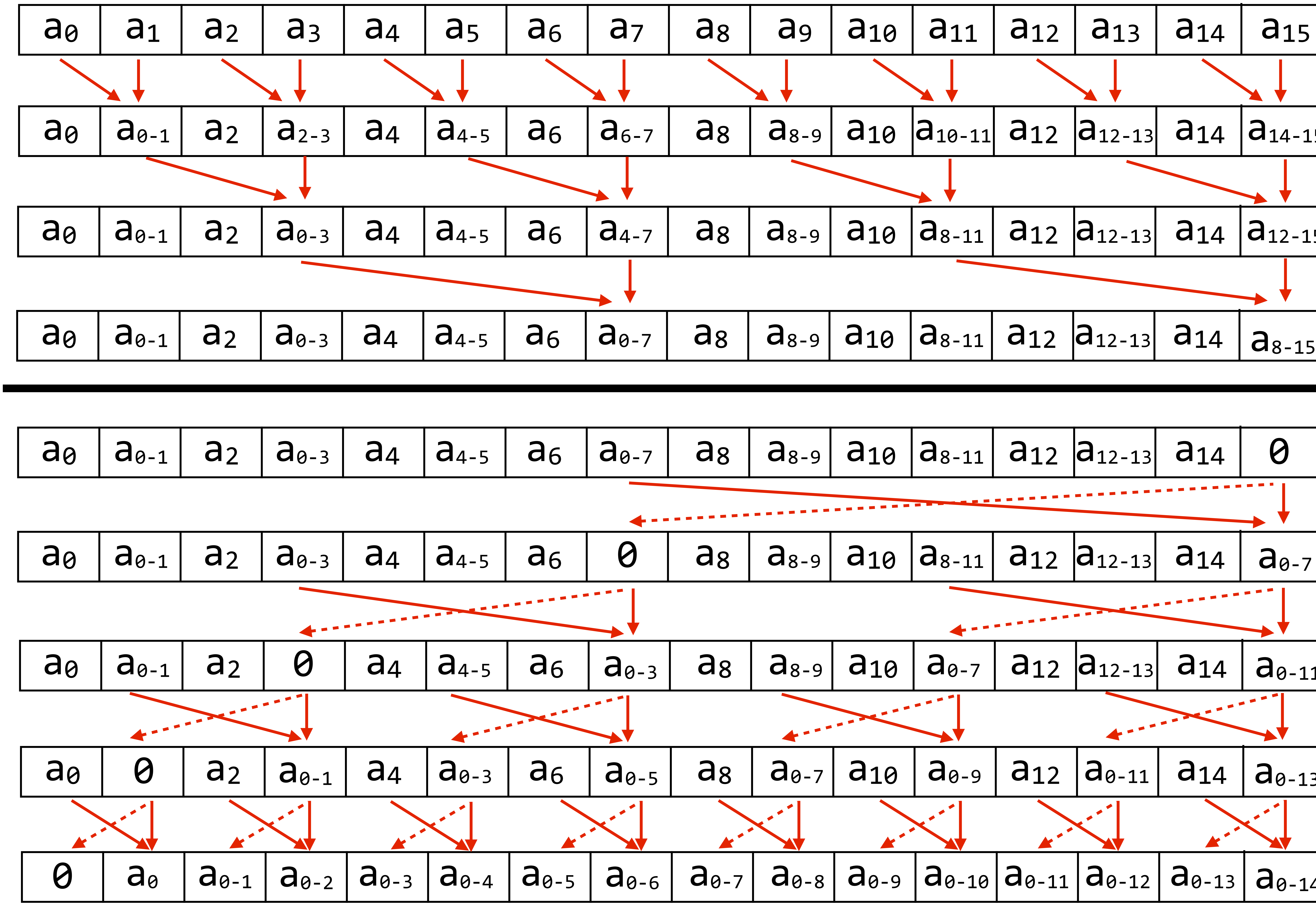
* not showing all dependencies in last step

Work: O(N lg N)

Inefficient compared to sequential algorithm!

Span: O(lg N)

Work-efficient parallel exclusive scan (O(N) work)



Work efficient exclusive scan algorithm

Up-sweep:

```
for d=0 to (log₂n - 1) do
   forall k=0 to n-1 by 2^(d+1) do
     a[k + 2^(d+1) - 1] = a[k + 2^d - 1] + a[k + 2^(d+1) - 1]
```

Down-sweep:

```
x[n-1] = 0
for d=(log₂n - 1) down to 0 do
   forall k=0 to n-1 by 2^(d+1) do
     tmp = a[k + 2^d - 1]
     a[k + 2^d - 1] = a[k + 2^(d+1) - 1]
     a[k + 2^(d+1) - 1] = tmp + a[k + 2^(d+1) - 1]
```

Work: O(N) (but what is the constant?)

Span: O(lg N) (but what is the constant?)

Locality: ??

- **The rest of these slides are not necessary for Assignment 2**
  - **But the following SIMD implementation is what we provide you in `sharedMemExclusiveScan`**

Exclusive scan: wide SIMD implementation

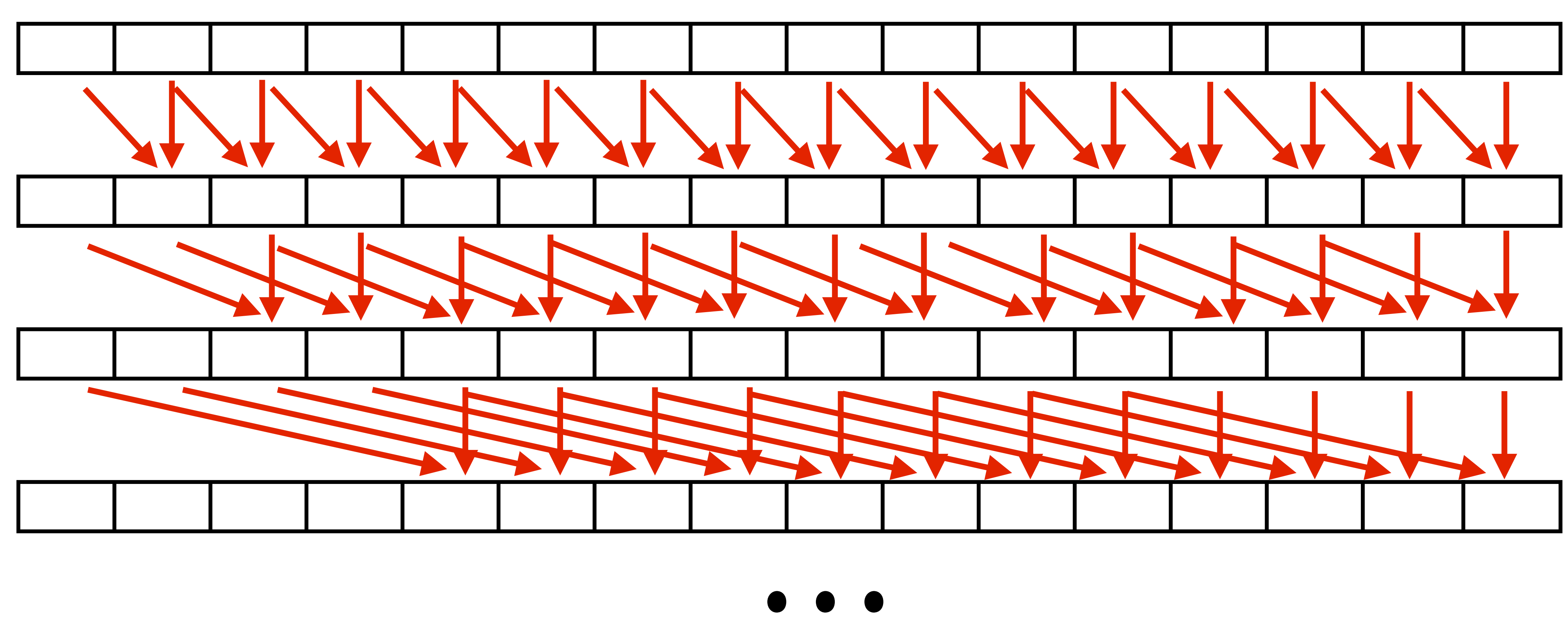
Example: perform exclusive scan on 32-element array: 32-wide GPU execution (SPMD program)

When `scan_warp` is run by a group of 32 CUDA threads, each thread returns the exclusive scan result for element 'idx' (note: upon completion ptr[] stores inclusive scan result)

CUDA thread index

```
template<class OP, class T>
__device__ T scan_warp(volatile T *ptr, const unsigned int idx)
{
   const unsigned int lane = idx & 31; // index of thread in warp (0..31)

   if (lane >= 1)  ptr[idx] = OP::apply(ptr[idx - 1],  ptr[idx]);
   if (lane >= 2)  ptr[idx] = OP::apply(ptr[idx - 2],  ptr[idx]);
   if (lane >= 4)  ptr[idx] = OP::apply(ptr[idx - 4],  ptr[idx]);
   if (lane >= 8)  ptr[idx] = OP::apply(ptr[idx - 8],  ptr[idx]);
   if (lane >= 16) ptr[idx] = OP::apply(ptr[idx - 16], ptr[idx]);

   return (lane>0) ? ptr[idx-1] : OP::identity();
}
```

Work: ??



Wide SIMD implementation

Example: exclusive scan 32-element array

32-wide GPU execution (SPMD program)

CUDA thread index

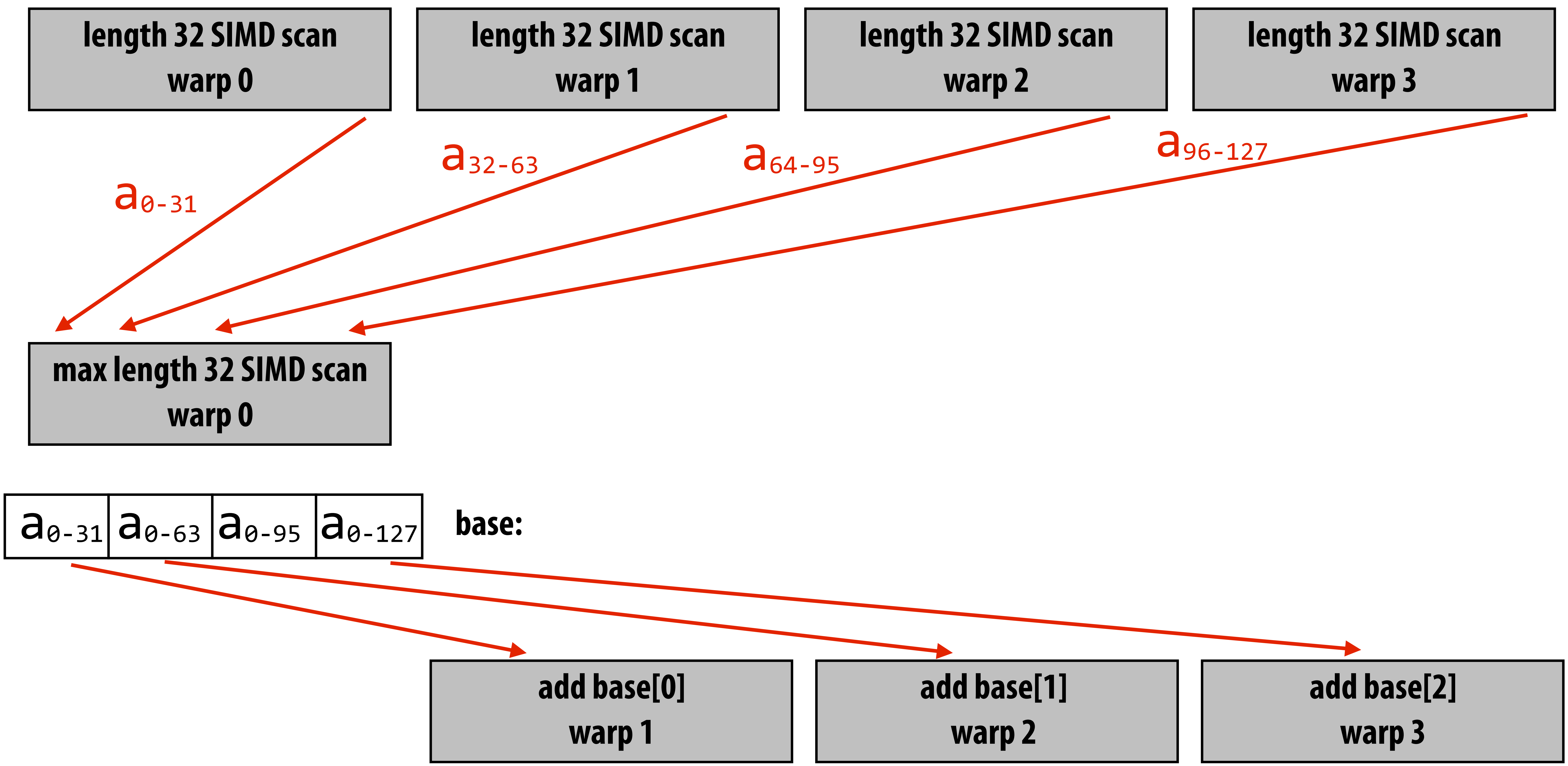
```
template<class OP, class T>
__device__ T scan_warp(volatile T *ptr, const unsigned int idx)
{
   const unsigned int lane = idx & 31; // index of thread in warp (0..31)

   if (lane >= 1)  ptr[idx] = OP::apply(ptr[idx - 1],  ptr[idx]);
   if (lane >= 2)  ptr[idx] = OP::apply(ptr[idx - 2],  ptr[idx]);
   if (lane >= 4)  ptr[idx] = OP::apply(ptr[idx - 4],  ptr[idx]);
   if (lane >= 8)  ptr[idx] = OP::apply(ptr[idx - 8],  ptr[idx]);
   if (lane >= 16) ptr[idx] = OP::apply(ptr[idx - 16], ptr[idx]);

   return (lane>0) ? ptr[idx-1] : OP::identity();
}
```

Work: N lg(N)

Work-efficient formulation of scan is not beneficial in this context because it results in low SIMD utilization. It would require more than 2x the number of instructions as the implementation above!

Building scan on larger array

Example: 128-element scan using four-warp thread block



Multi-threaded, SIMD implementation

Example: cooperating threads in a CUDA thread block

(We provided similar code in assignment 2, assumes length of array given by ptr is same as number of threads per block)

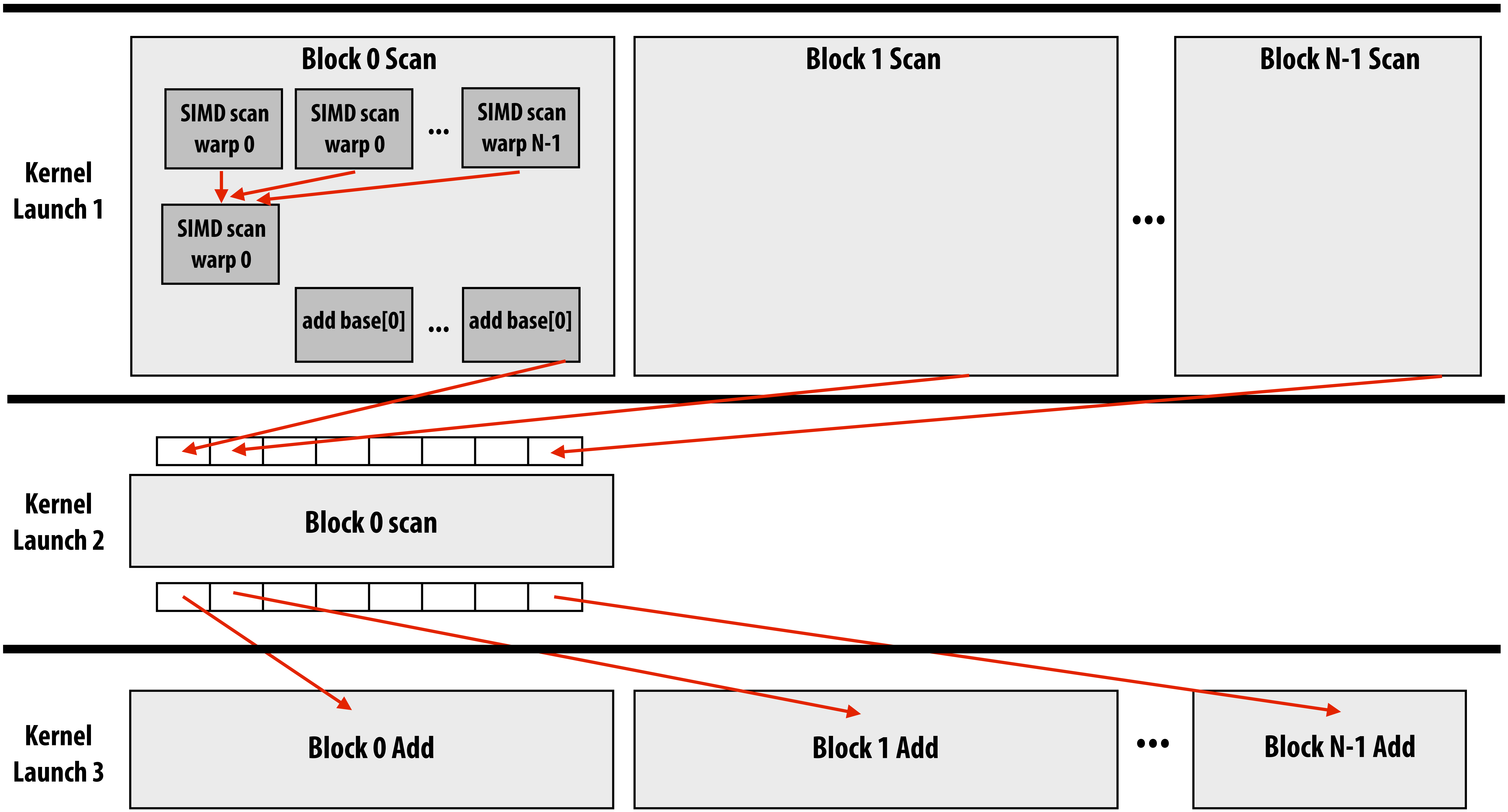
CUDA thread index

```
template<class OP, class T>
__device__ void scan_block(volatile T *ptr, const unsigned int idx)
{
   const unsigned int lane = idx & 31; // index of thread in warp (0..31)
   const unsigned int warpid = idx >> 5;

   T val = scan_warp<OP,T>(ptr, idx);            // Step 1. per-warp partial scan

   if (lane == 31)  ptr[warpid] = ptr[idx];      // Step 2. copy partial-scan bases
   __syncthreads();


   if (warpid == 0) scan_warp<OP, T>(ptr, idx); // Step 3. scan to accumulate bases
   __syncthreads();


   if (warpid > 0)                              // Step 4. apply bases to all elements
       val = OP::apply(ptr[warpid-1], val);
   __syncthreads();


   ptr[idx] = val;
}
```

- **And if you are really interested in building a fast scan for large arrays...**

Building a larger scan

Example: 1 million element scan (1024 elements per block)



Exceeding 1 million elements requires partitioning phase 2 into multiple blocks

Scan implementation

- **Parallelism**
  - Scan algorithm features O(N) parallel work
  - But efficient implementations only leverage as much parallelism as required to make good utilization of the machine
    - Reduce work and reduce communication/synchronization
- **Locality**
  - Multi-level implementation matches memory hierarchy

    (Per-block implementation carried out in local memory)
- **Heterogeneity: different strategy at different machine levels**
  - Different algorithm for intra-warp scan than inter-thread scan

Challenge

- Can you approach the performance of the Thrust library's scan?
- See function cudaScanThrust() in /scan/scan.cu
- Also see the Thrust documentation:
  - http://thrust.github.io/
  - http://thrust.github.io/doc/group__prefixsums.html