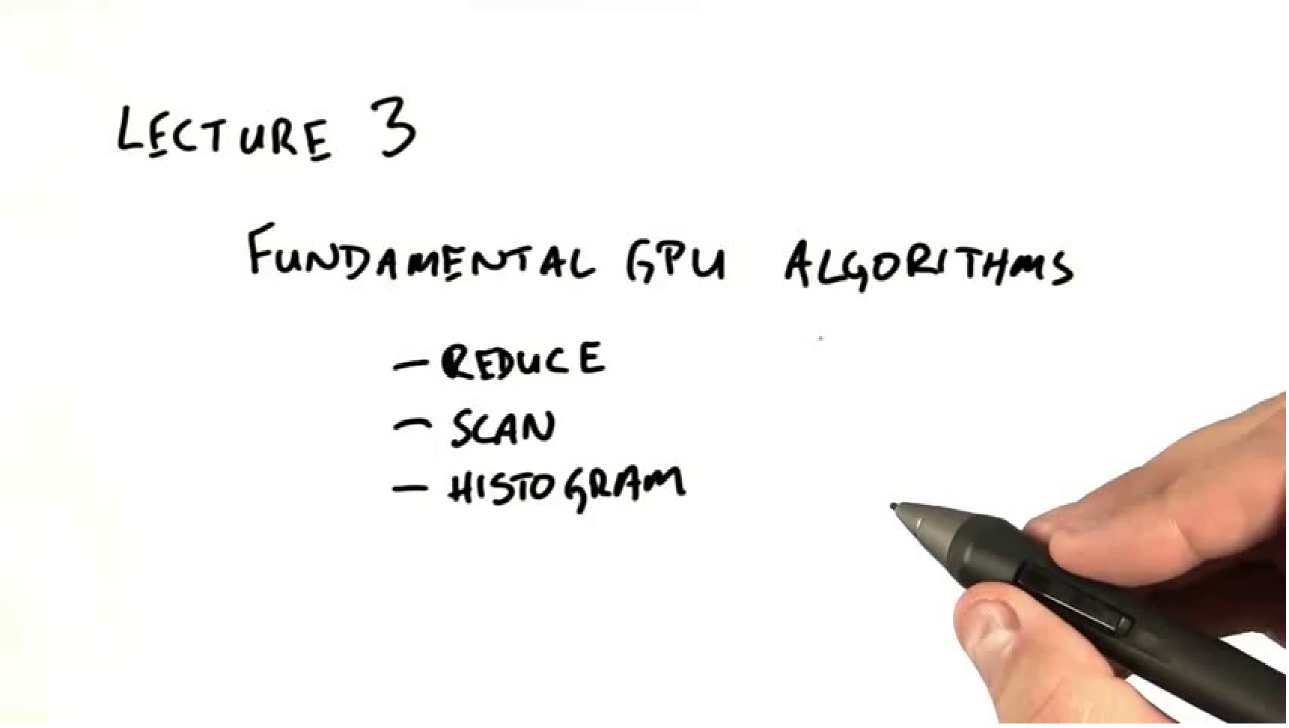
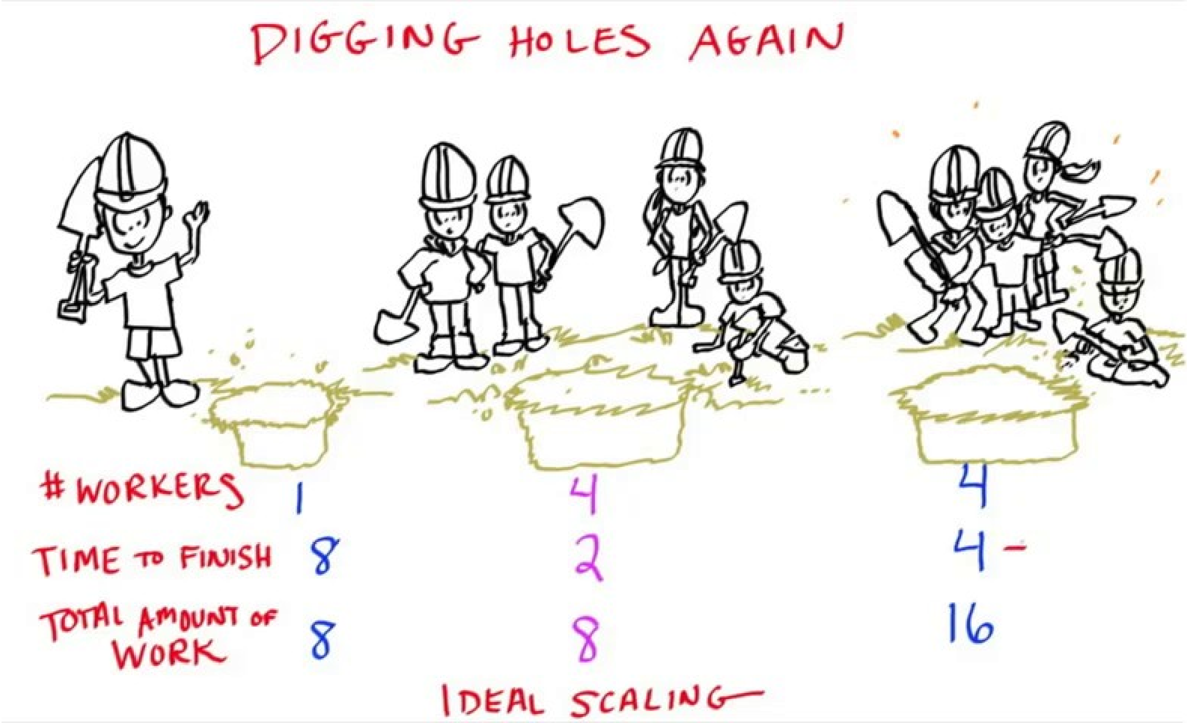
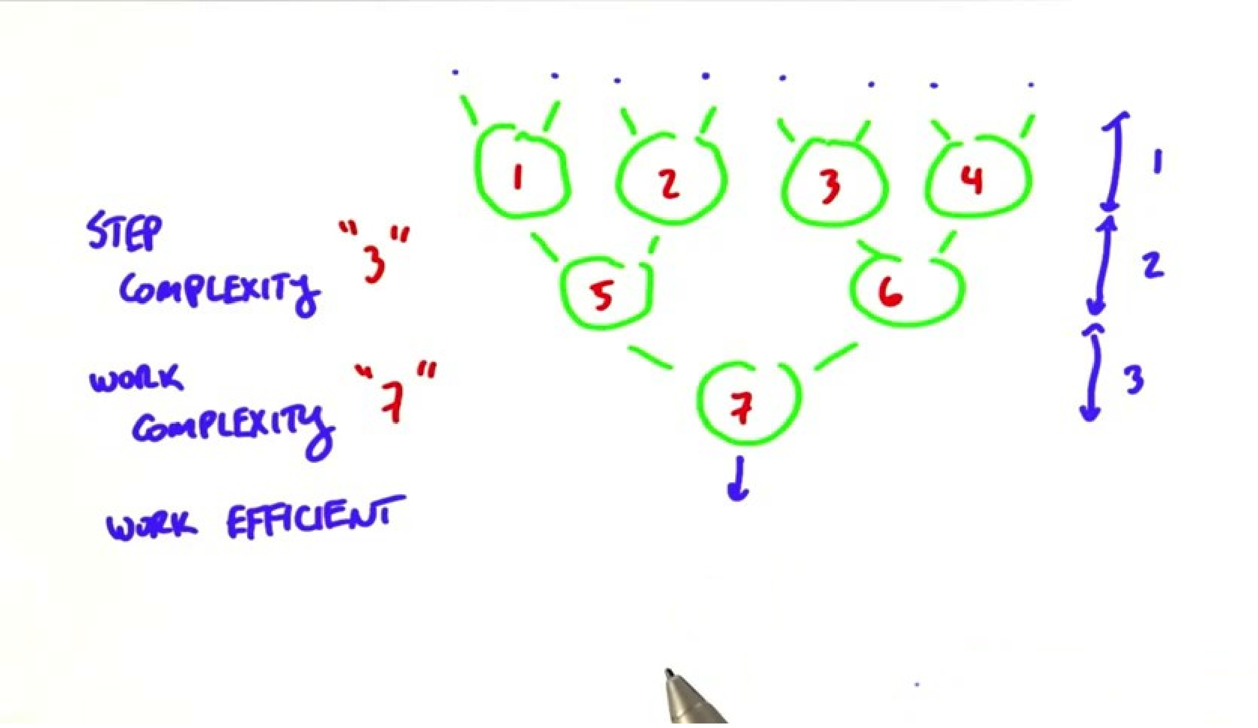
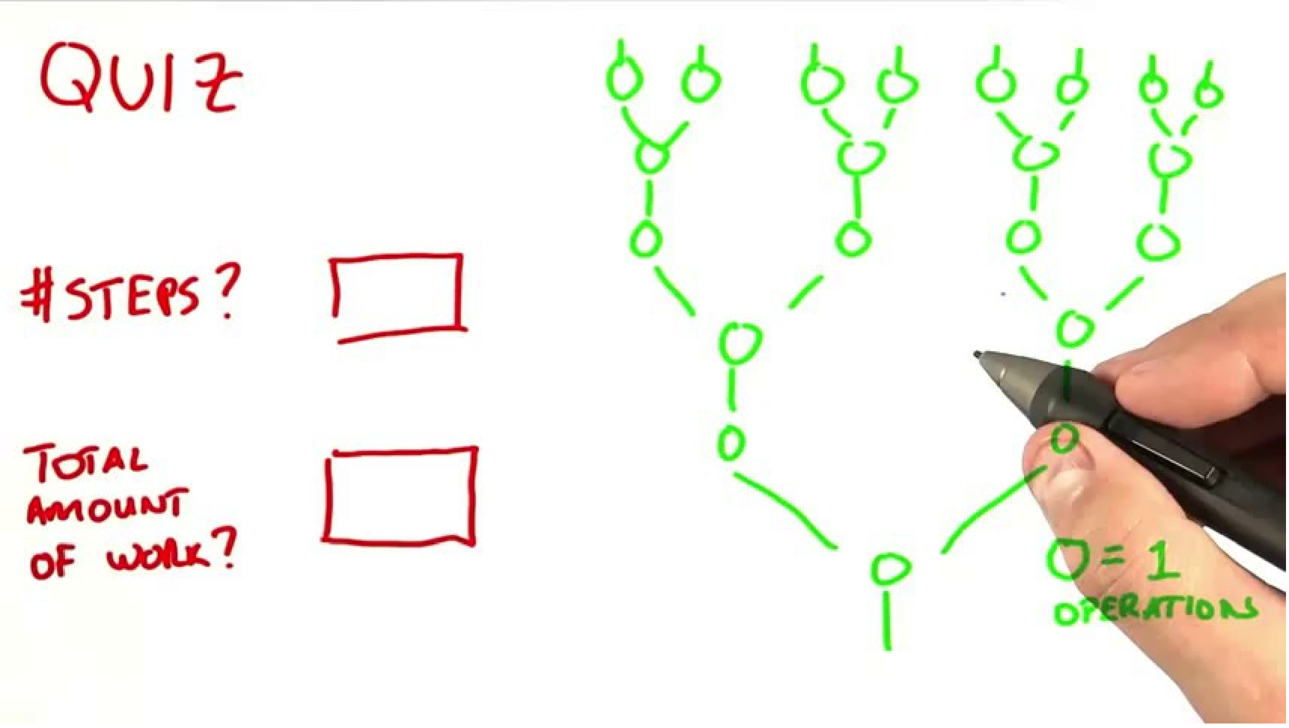

LECTURE 3
FUNDAMENTAL GPU ALGORITHMS
– REDUCE
– SCAN
– HISTOGRAM

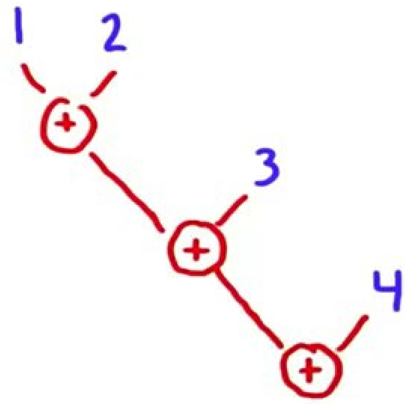

DIGGING HOLES AGAIN
WORKERS 1 4 4
TIME TO FINISH 8 2 4-
TOTAL AMOUNT OF WORK 8 8 16
IDEAL SCALING

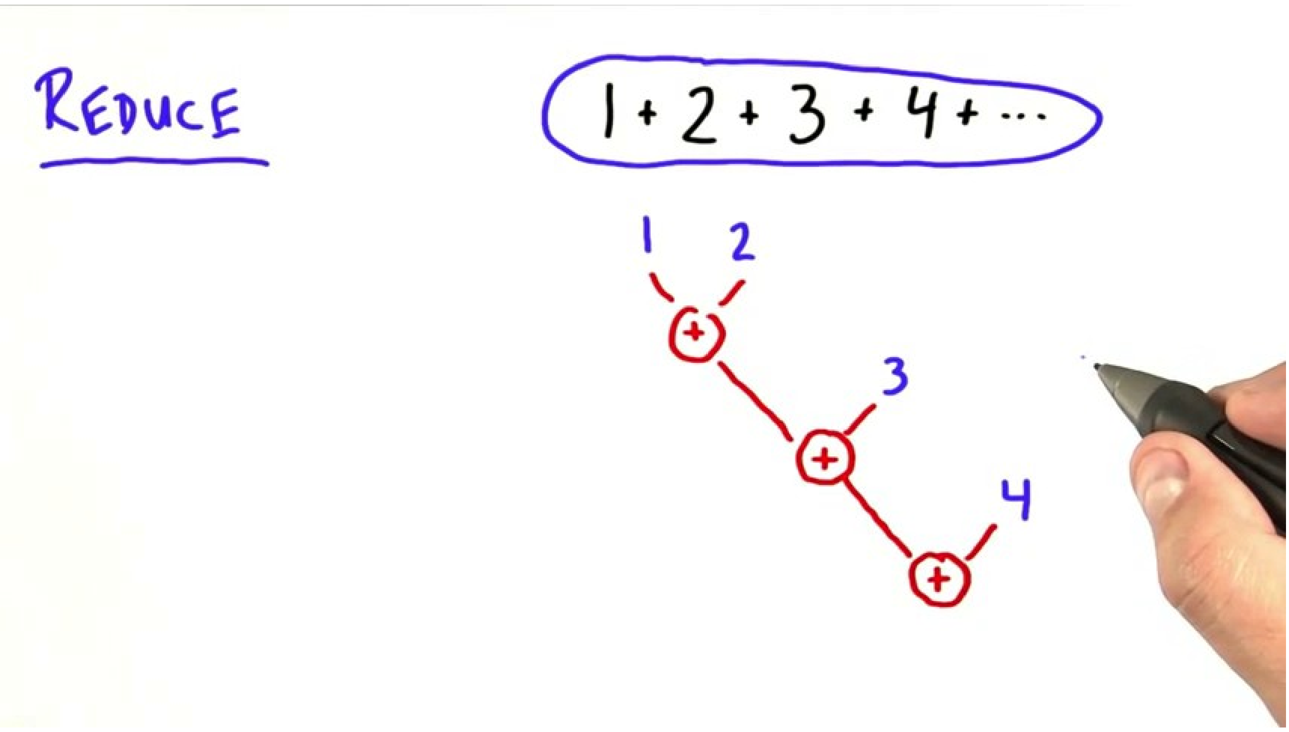

STEP COMPLEXITY "3"
WORK COMPLEXITY "7"
WORK EFFICIENT
1
2
3
4
5
6
7
1
2
3

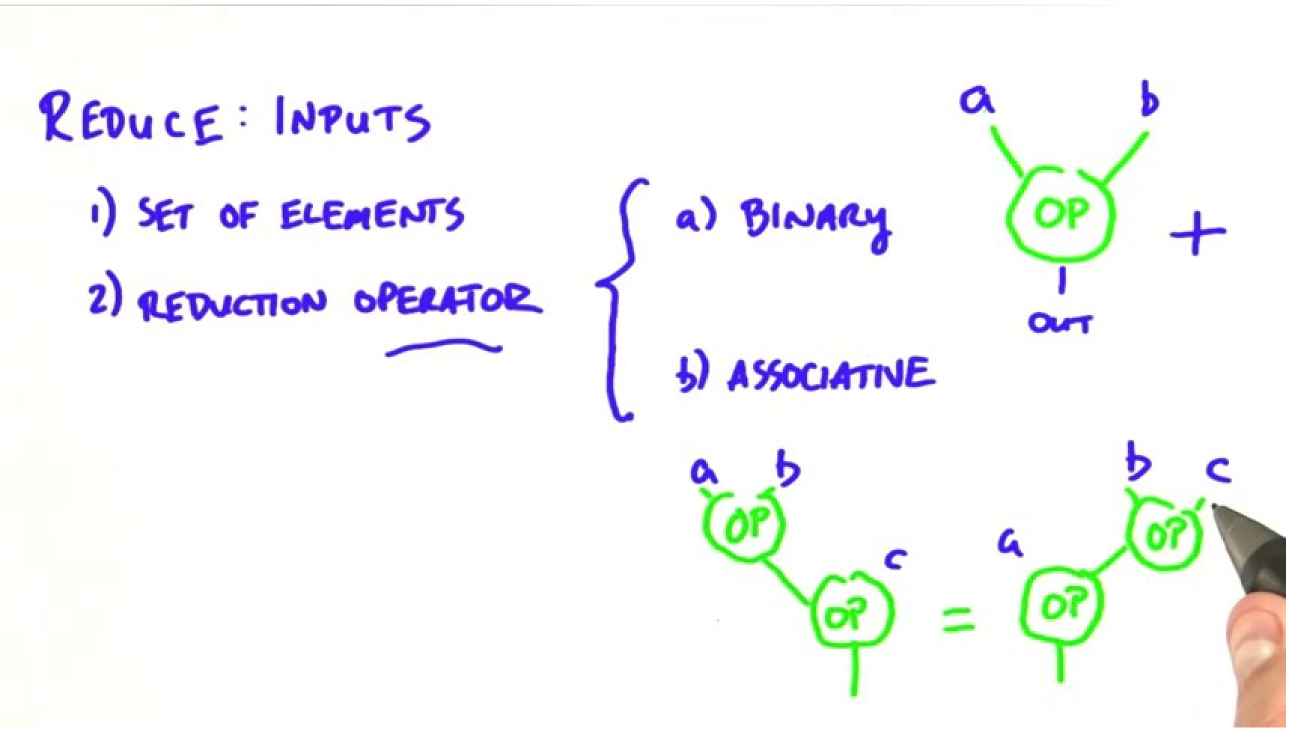

QUIZ
#STEPS?
TOTAL AMOUNT OF WORK?
O = 1 OPERATION

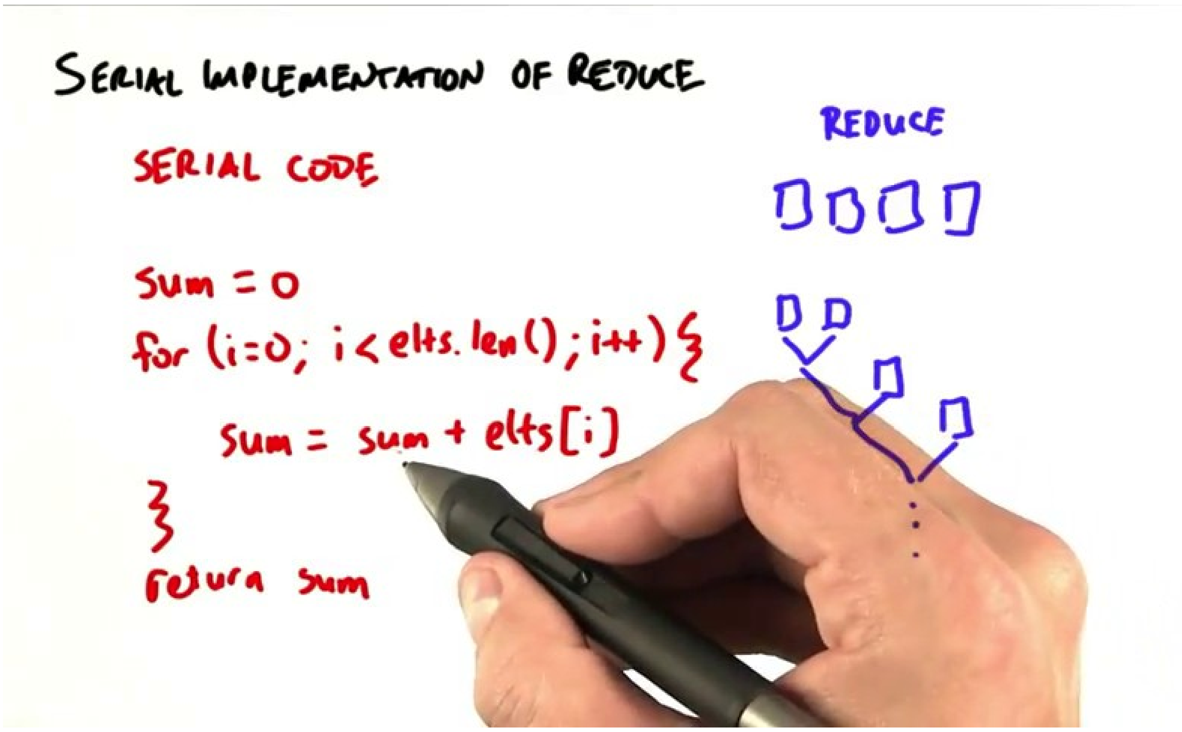
REDUCE

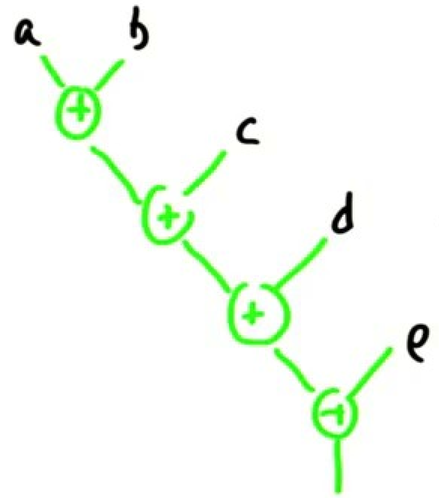
$1 + 2 + 3 + 4 + \cdots$

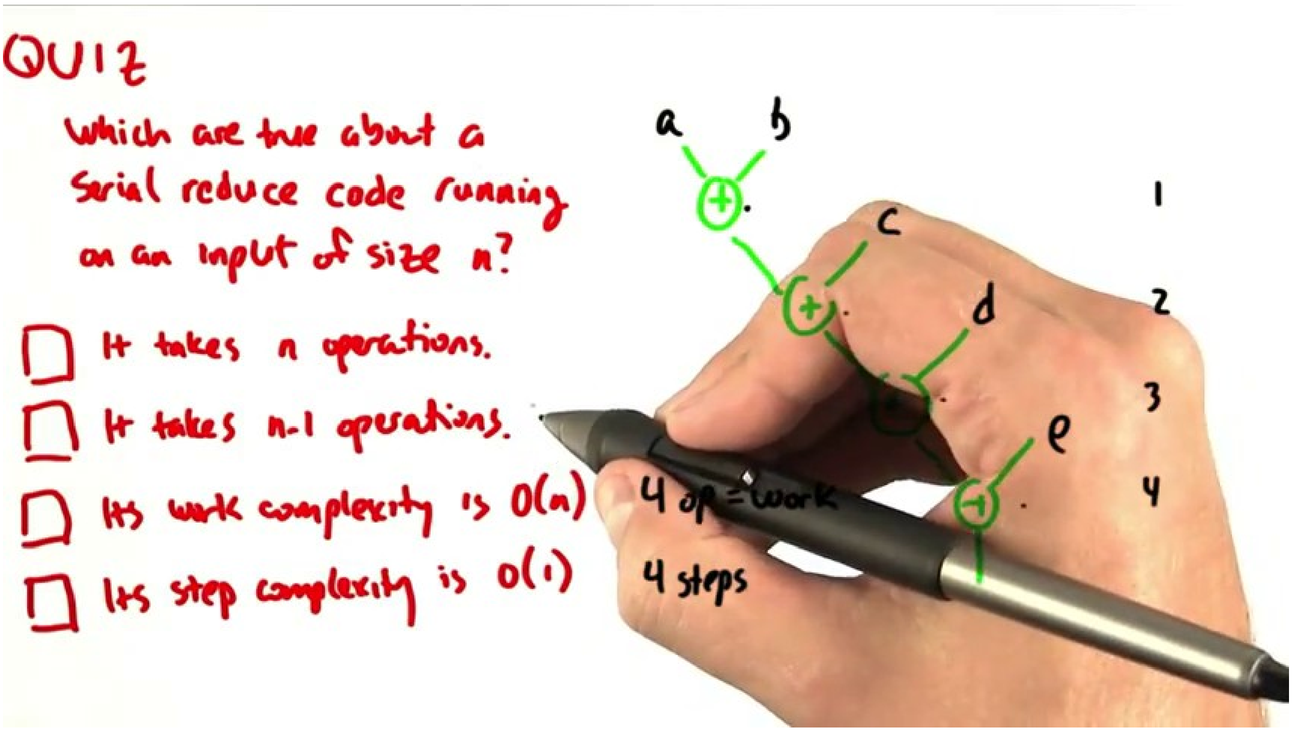



REDUCE
1 + 2 + 3 + 4 + ...
1
2
+
3
+
4
+

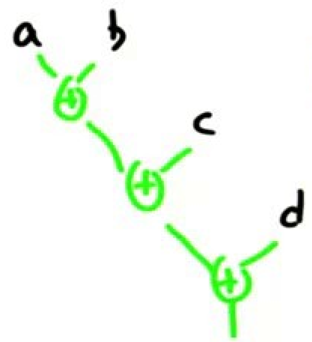

REDUCE: INPUTS
1) SET OF ELEMENTS
2) REDUCTION OPERATOR
a) BINARY
a
b
OP
OUT
+
b) ASSOCIATIVE
a
b
OP
c
OP
=
a
b
c
OP
OP

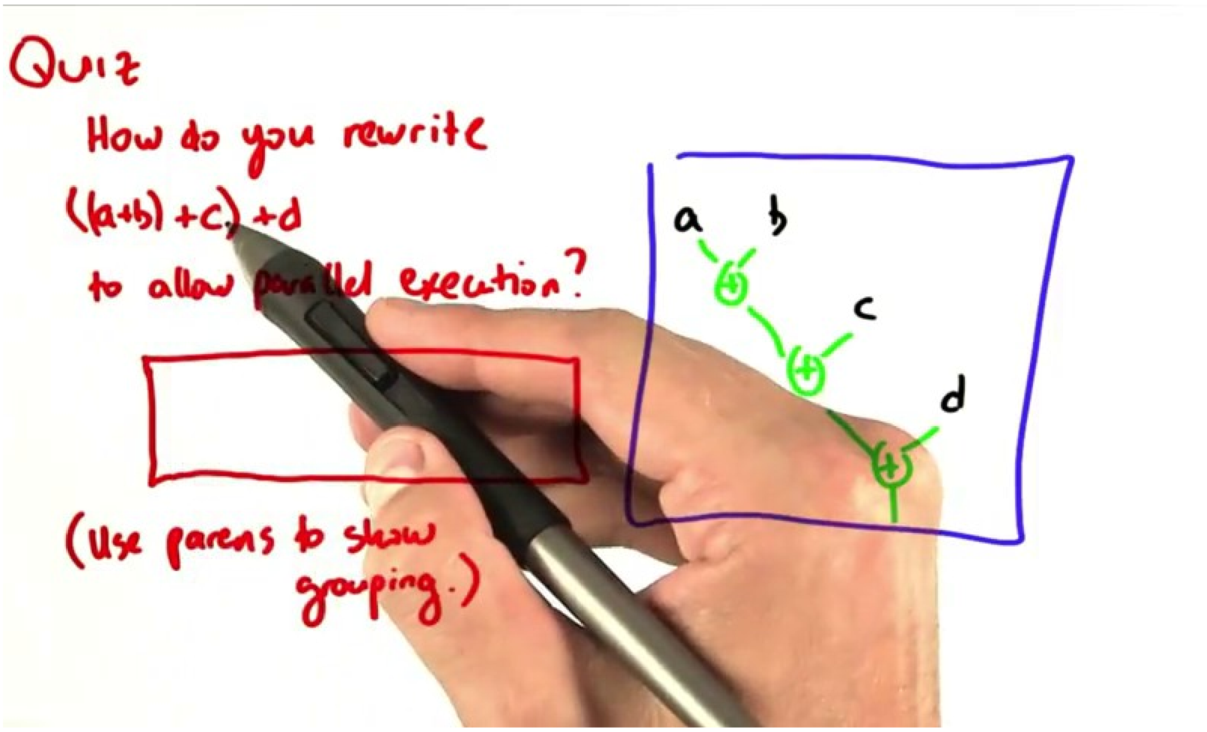

SERIAL IMPLEMENTATION OF REDUCE
SERIAL CODE
sum = 0
for (i=0; i< elts.len(); i++) {
sum = sum + elts[i]
}
return sum
REDUCE

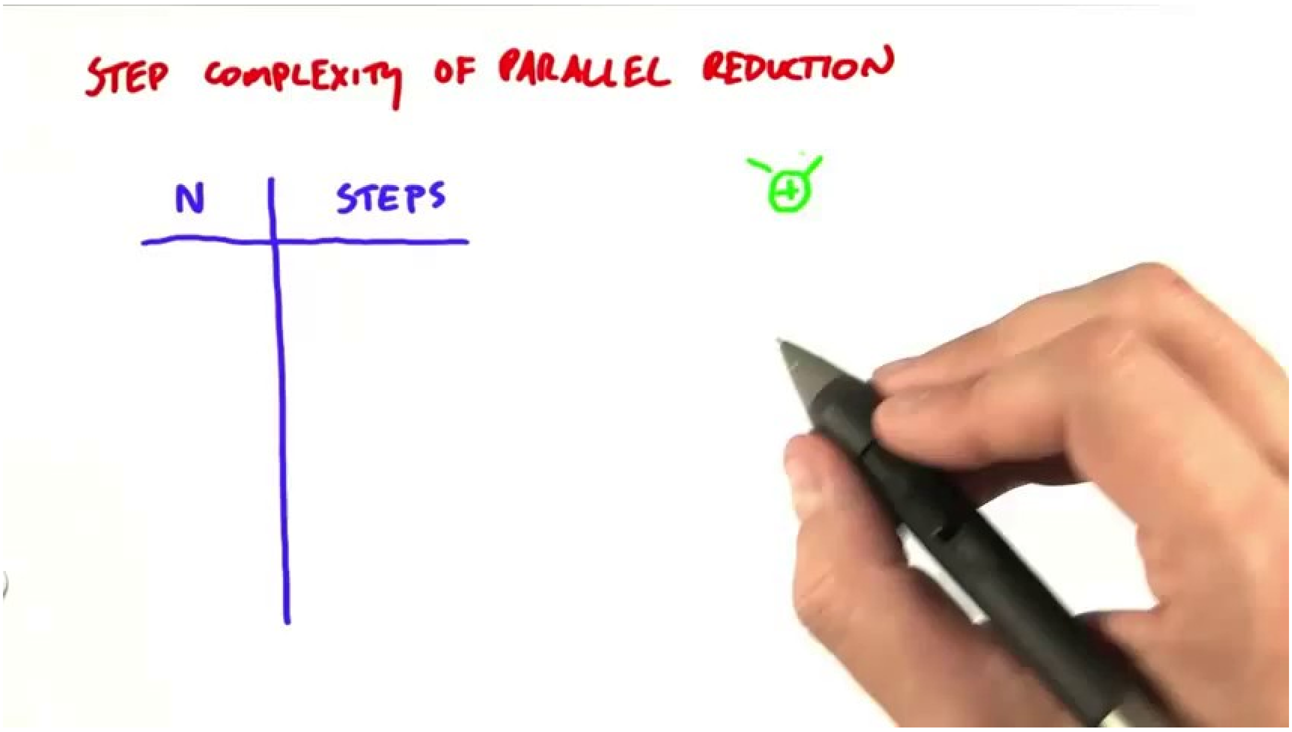

a
b
+
c
+
d
+
e
-

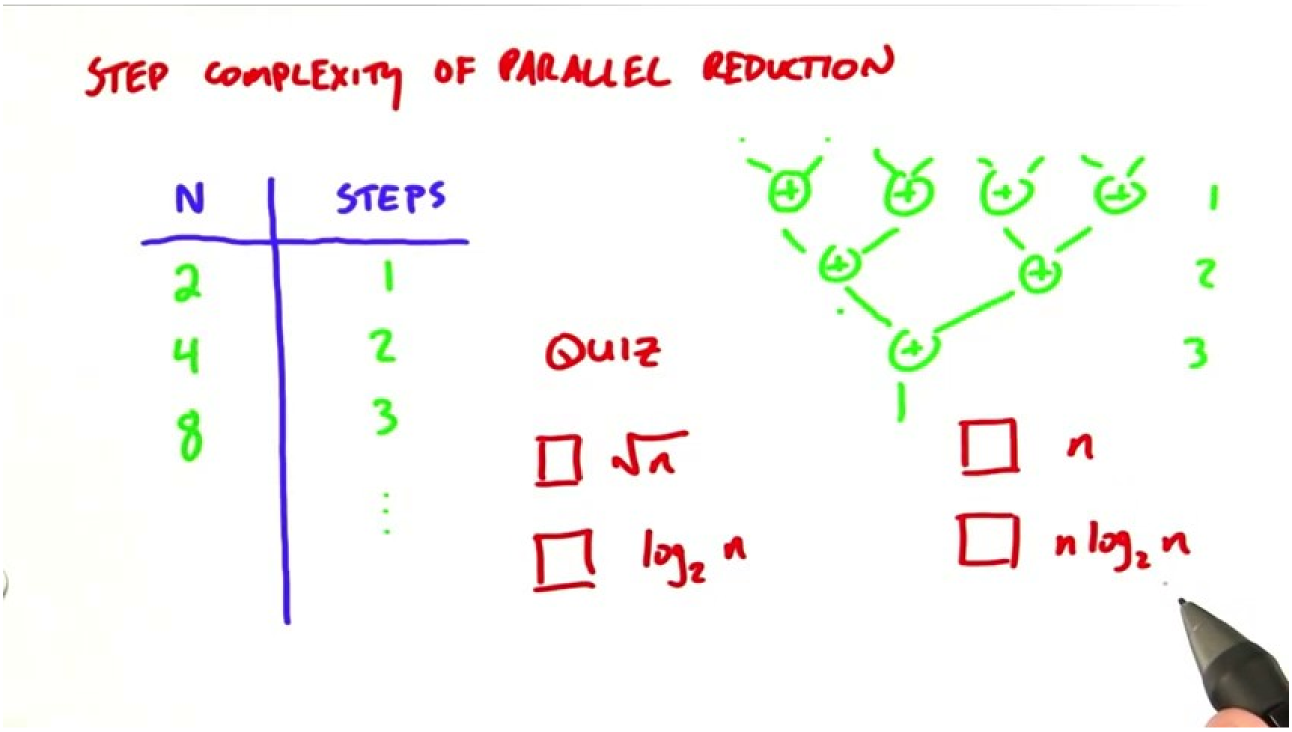

QUIZ
Which are true about a serial reduce code running on an input of size n?
It takes n operations.
It takes n-1 operations.
Its work complexity is O(n)
Its step complexity is O(1)
a
b
c
d
e
1
2
3
4
4 op = work
4 steps

PARALLEL REDUCE




Quiz
How do you rewrite
((a+b)+c)+d
to allow parallel execution?
(Use parens to show grouping.)
a
b
c
d
+
+
+


STEP COMPLEXITY OF PARALLEL REDUCTION
N
STEPS

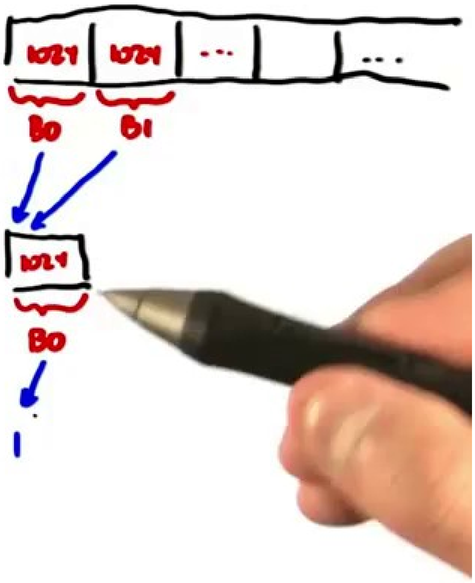


STEP COMPLEXITY OF PARALLEL REDUCTION
N
STEPS
2 1
4 2
8 3
⋮
1
2
3
QUIZ
√n
n
log2 n
n log2 n

REDUCING 1M ELEMENTS

(1) 1024 BLOCKS x 1024 THREADS

(2) 1 BLOCK x 1024 THREADS



```
__global__ void global_reduce_kernel(float * d_out, float * d_in)
{
    int myId = threadIdx.x + blockDim.x * blockIdx.x;
    int tid  = threadIdx.x;

    // do reduction in global mem
    for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1)
    {
        if (tid < s)
        {
            d_in[myId] += d_in[myId + s];
        }
        __syncthreads();        // make sure all adds at one stage are done!
    }

    // only thread 0 writes result for this block back to global mem
    if (tid == 0)
    {
        d_out[blockIdx.x] = d_in[myId];
    }
}
```

U:--- reduce.cu   2% L16   (C++/l A

```
__global__ void shmem_reduce_kernel(float * d_out, const float * d_in)
{
    // sdata is allocated in the kernel call: 3rd arg to <<<b, t, shmem>>>
    extern __shared__ float sdata[];

    int myId = threadIdx.x + blockDim.x * blockIdx.x;
    int tid  = threadIdx.x;

    // load shared mem from global mem
    sdata[tid] = d_in[myId];
    __syncthreads();            // make sure entire block is loaded!

    // do reduction in shared mem
    for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1)
    {
        if (tid < s)
        {
            sdata[tid] += sdata[tid + s];
        }
        __syncthreads();        // make sure all adds at one stage are done!
    }
```

U:--- reduce.cu 13% L38 (C++/l Abbrev)

```
    int tid  = threadIdx.x;

    // load shared mem from global mem
    sdata[tid] = d_in[myId];
    __syncthreads();            // make sure entire block is loaded!

    // do reduction in shared mem
    for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1)
    {
        if (tid < s)
        {
            sdata[tid] += sdata[tid + s];
        }
        __syncthreads();        // make sure all adds at one stage are done!
    }

    // only thread 0 writes result for this block back to global mem
    if (tid == 0)
    {
        d_out[blockIdx.x] = sdata[0];
    }
}
U:---  reduce.cu     17% L44    (C++/l Abbrev)
```

SHARED VS GLOBAL MEMORY BANDWIDTH

THE GLOBAL MEMORY VERSION USES

$\boxed{}$

TIMES AS MUCH GLOBAL MEM BW AS THE SHARED MEM VERSION?

SCAN

— EXAMPLE

INPUT: 1 2 3 4

OPERATION: ADD

OUTPUT: 1 3 6 10

SCAN

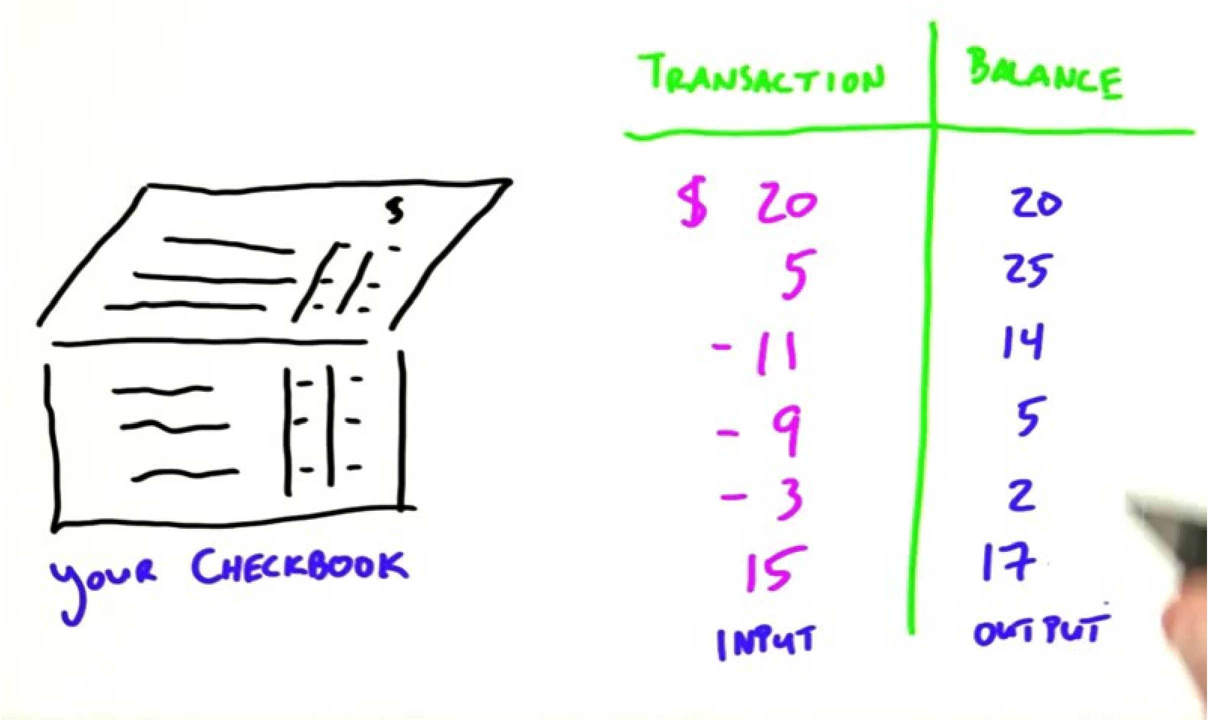
- EXAMPLE

  INPUT: 1 2 3 4

  OPERATION: ADD

  OUTPUT: 1 3 6 10

- ADDRESSES SET OF PROBLEMS OTHERWISE DIFFICULT TO PARALLELIZE
- NOT USEFUL IN SERIAL WORLD BUT VERY USEFUL IN PARALLEL
- TODAY: EXPLAINING WHAT + HOW
  BUT NOT WHY (NEXT LECTURE)


$
YOUR CHECKBOOK
TRANSACTION
BALANCE
$ 20
20
5
25
- 11
14
- 9
5
- 3
2
15
17
INPUT
OUTPUT

INPUTS TO SCAN

- INPUT ARRAY
- BINARY ASSOCIATIVE OPERATOR

$\Big\}$ LIKE REDUCE

- IDENTITY ELEMENT $[I \text{ op } a = a]$

OP	I	BECAUSE
+	Ø	Ø + a = a
min (on unsigned chars)	ØxFF	min(ØxFF, a) = a

QUIZ

WHAT IS THE IDENTITY FOR ...

Multiply

Logical or

Logical and

What Scan Does

Input: array A, operator $\oplus$, identity I

$$\left[\, a_0 \quad a_1 \quad a_2 \quad a_3 \quad \cdots \quad a_{n-1} \,\right]$$

INPUT

$$\left[\, I \quad a_0 \quad a_0 \oplus a_1 \quad a_0 \oplus a_1 \oplus a_2 \quad \cdots \quad a_0 \oplus a_1 \oplus a_2 \oplus \cdots a_{n-2} \,\right]$$

OUTPUT

What Scan Does

Input: array A, operator $\oplus$, identity I

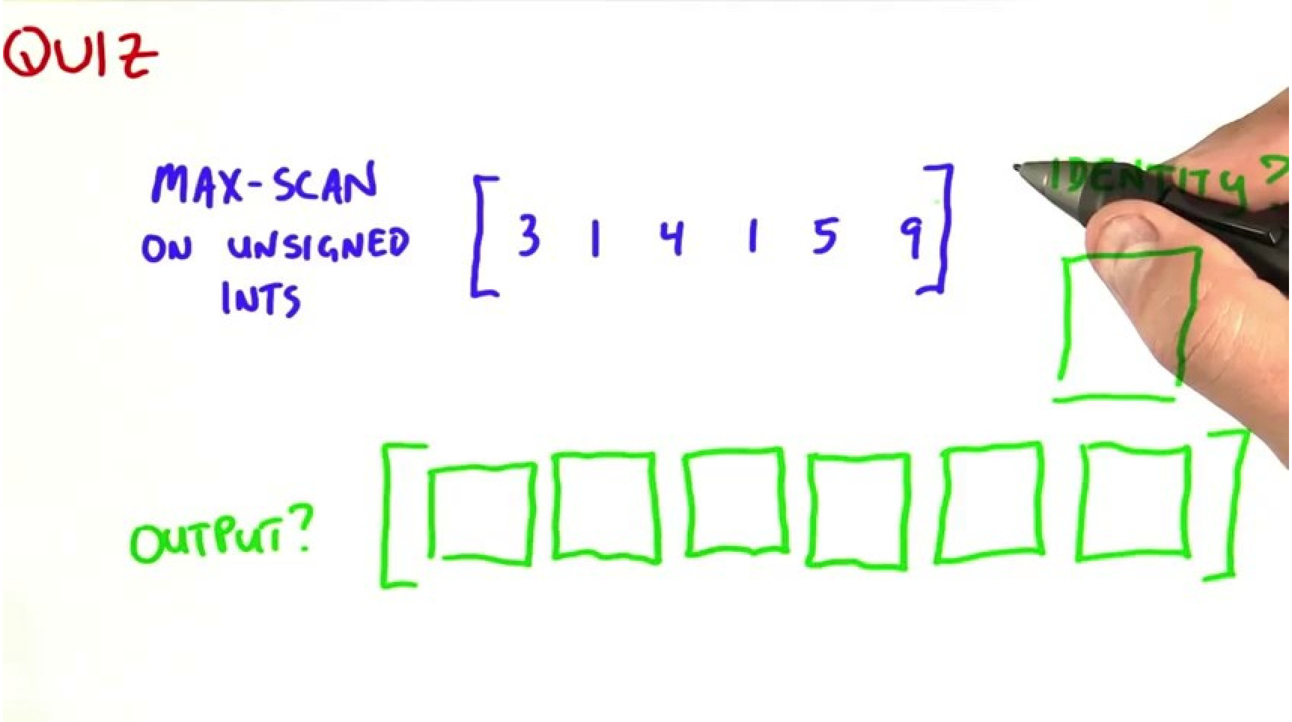
$\oplus$: PLUS, I: $\emptyset$

$$[\, a_0 \quad a_1 \quad a_2 \quad a_3 \quad \cdots \quad a_{n-1} \,] \qquad \text{INPUT}$$

$$[\, I \quad a_0 \quad a_0 \oplus a_1 \quad a_0 \oplus a_1 \oplus a_2 \quad \cdots \quad a_0 \oplus a_1 \oplus a_2 \oplus \cdots a_{n-2} \,] \qquad \text{OUTPUT}$$

IN + $[\,3 \quad 1 \quad 4 \quad 1 \quad 5 \quad 9\,]$

OUT $[\,0 \quad 3 \quad 4 \quad 8 \quad 9 \quad 14\,]$


QUIZ
MAX-SCAN ON UNSIGNED INTS
[3 1 4 1 5 9]
IDENTITY?
OUTPUT?

SERIAL IMPLEMENTATION OF SCAN

```
int acc = identity;
for (i=0 ; i< elements.length(); i++) {
    acc = acc op element[i];
    out[i]= acc;
}
```

SERIAL IMPLEMENTATION OF INCLUSIVE SCAN

```
int acc = identity;
for (i=0 ; i< elements.length(); i++) {
    acc = acc op element[i];
    out[i]= acc;
}
```

QUIZ: CONVERT TO EXCLUSIVE SCAN.

INCLUSIVE VS EXCLUSIVE SCAN

INPUT: [13 7 16 21 8 20 13 12]

EXCLUSIVE SCAN OUTPUT : [ ]

INCLUSIVE SCAN OUTPUT

INCLUSIVE VS EXCLUSIVE SCAN

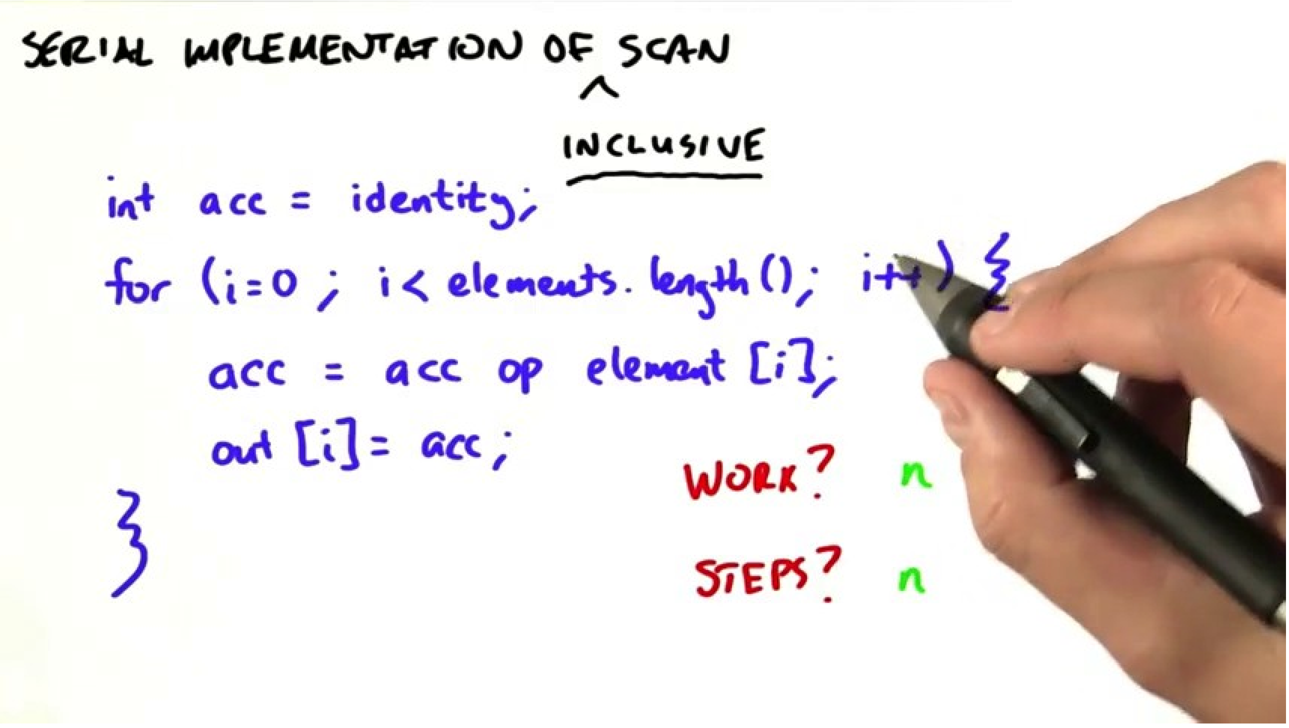
INPUT: [13 7 16 21 8 20 13 12]

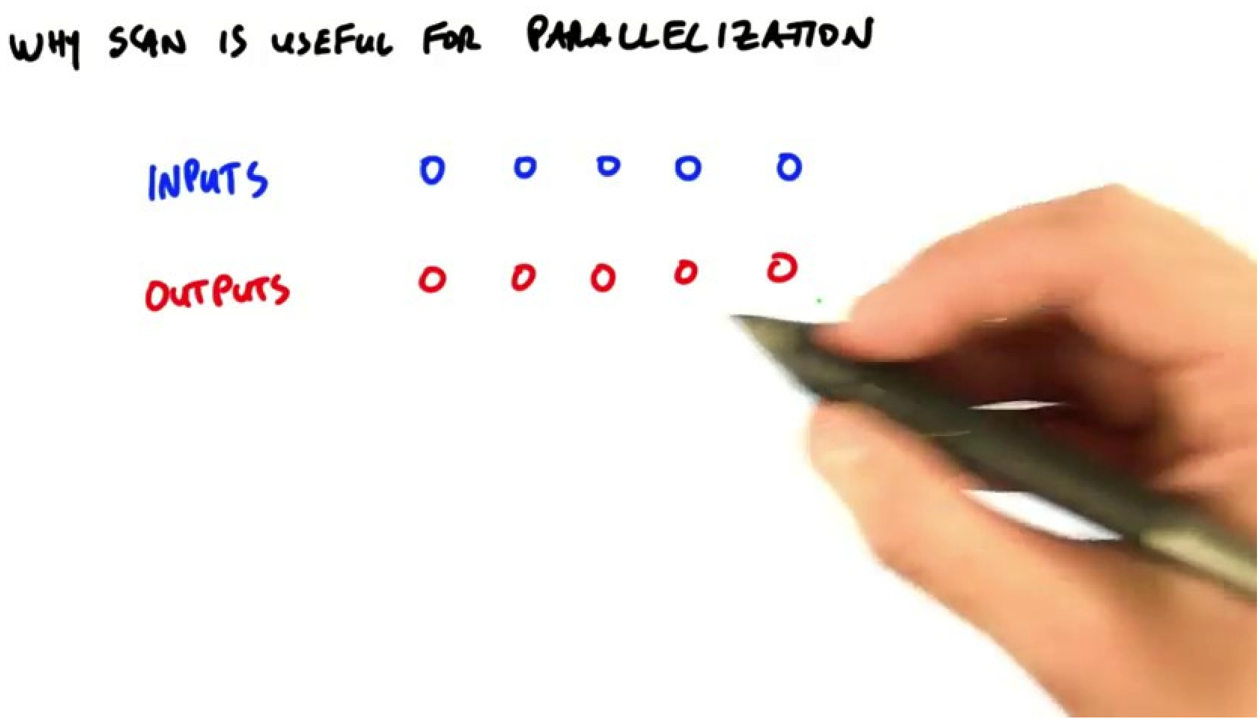
EXCLUSIVE SCAN OUTPUT: [0 13 20 36 57 65 85 98]

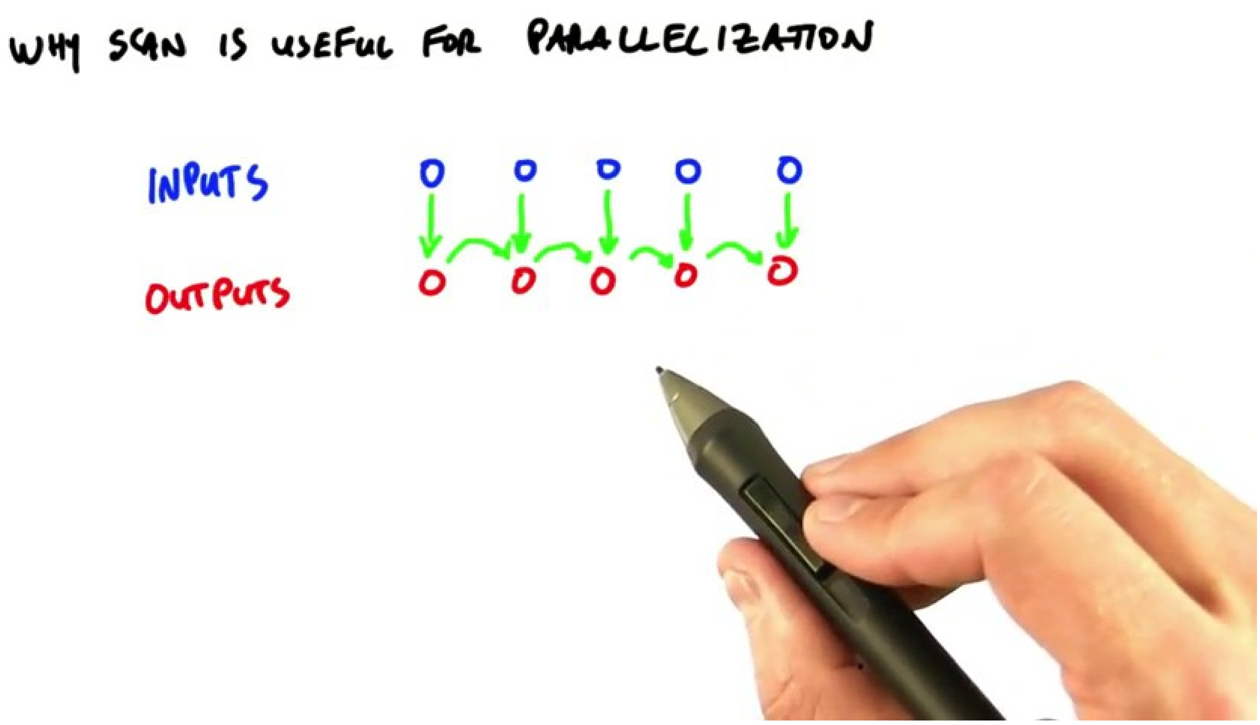
OUTPUT: ALL ELEMENTS BEFORE, NOT CURRENT ELT.

INCLUSIVE SCAN OUTPUT [13 20 36 57 65 85 98 110]

OUTPUT: ALL ELEMENTS BEFORE AND CURRENT ELT.


SERIAL IMPLEMENTATION OF SCAN
INCLUSIVE
int acc = identity;
for (i=0 ; i< elements.length(); i++) {
acc = acc op element [i];
out [i]= acc;
}
WORK? n
STEPS? n

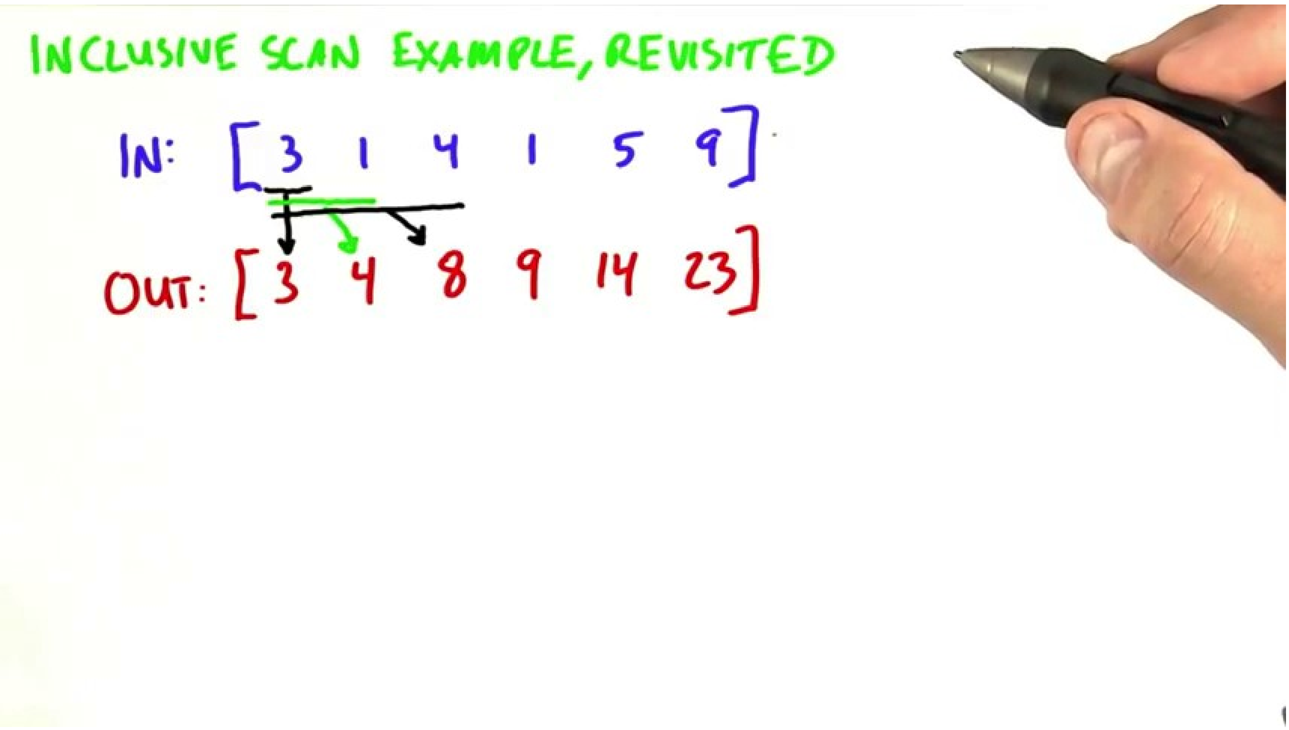

WHY SCAN IS USEFUL FOR PARALLELIZATION
INPUTS
OUTPUTS

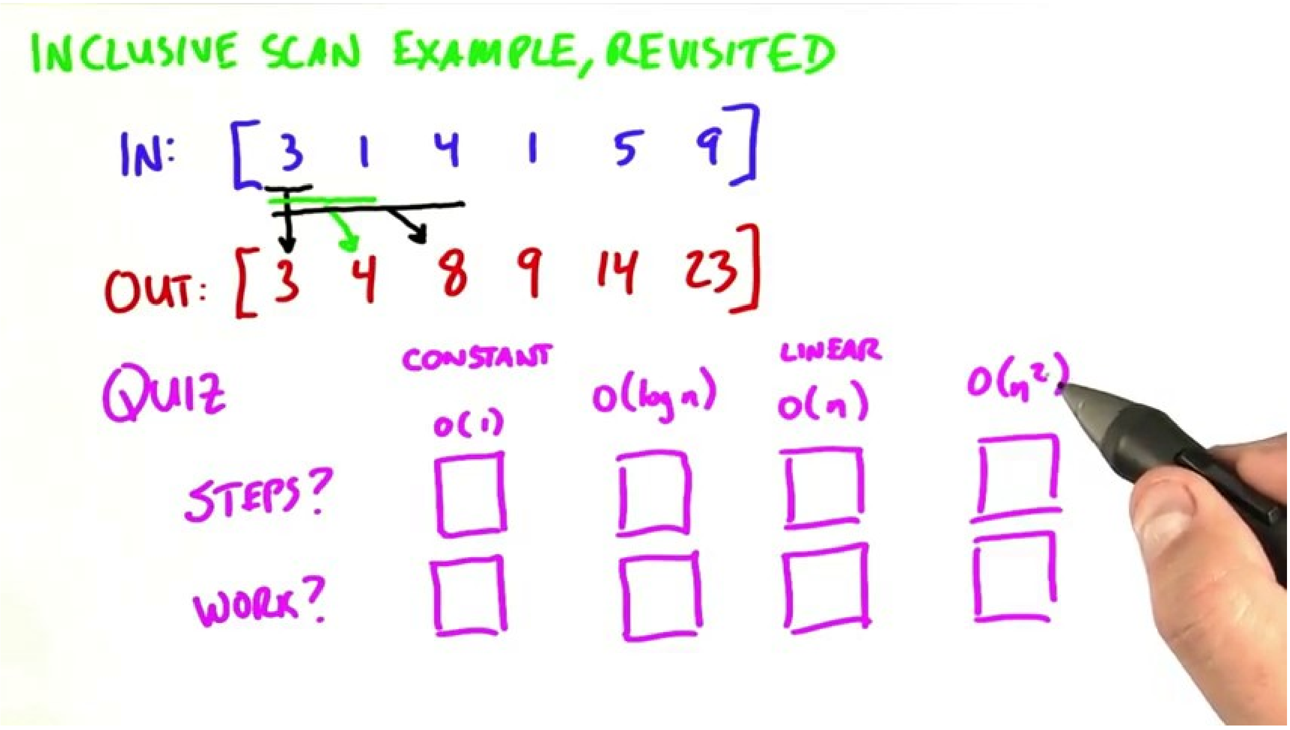

WHY SCAN IS USEFUL FOR PARALLELIZATION
INPUTS
OUTPUTS

INCLUSIVE SCAN EXAMPLE, REVISITED

IN: [3 1 4 1 5 9]

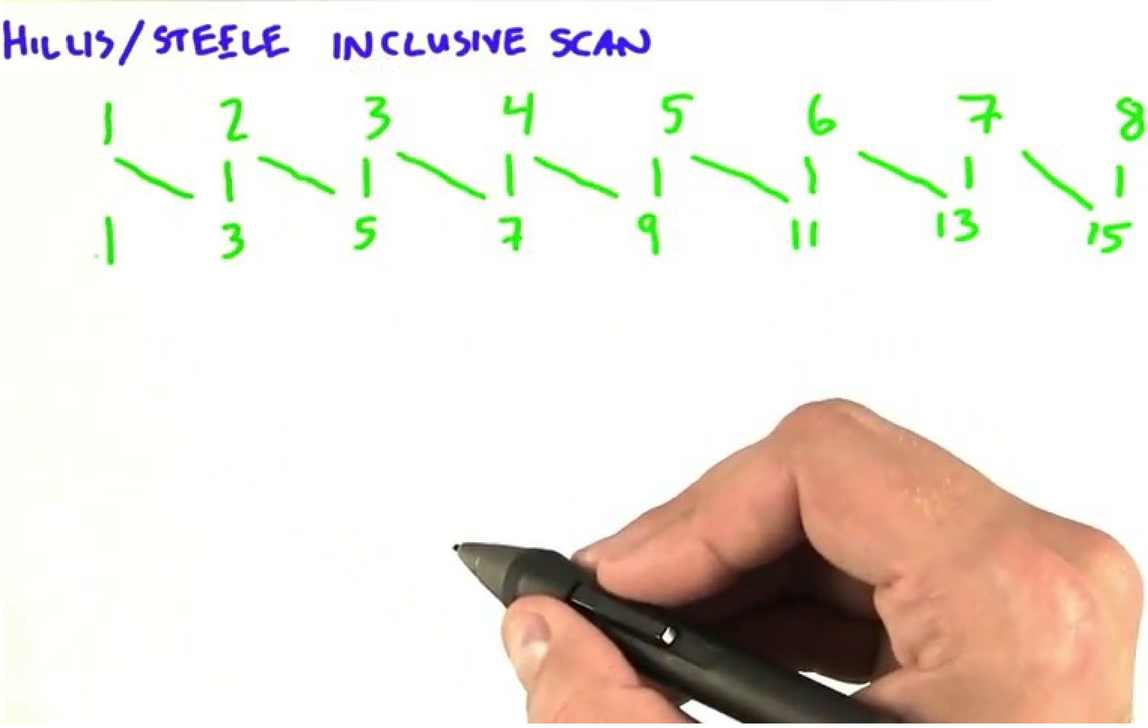
OUT: [3 4 8 9 14 23]

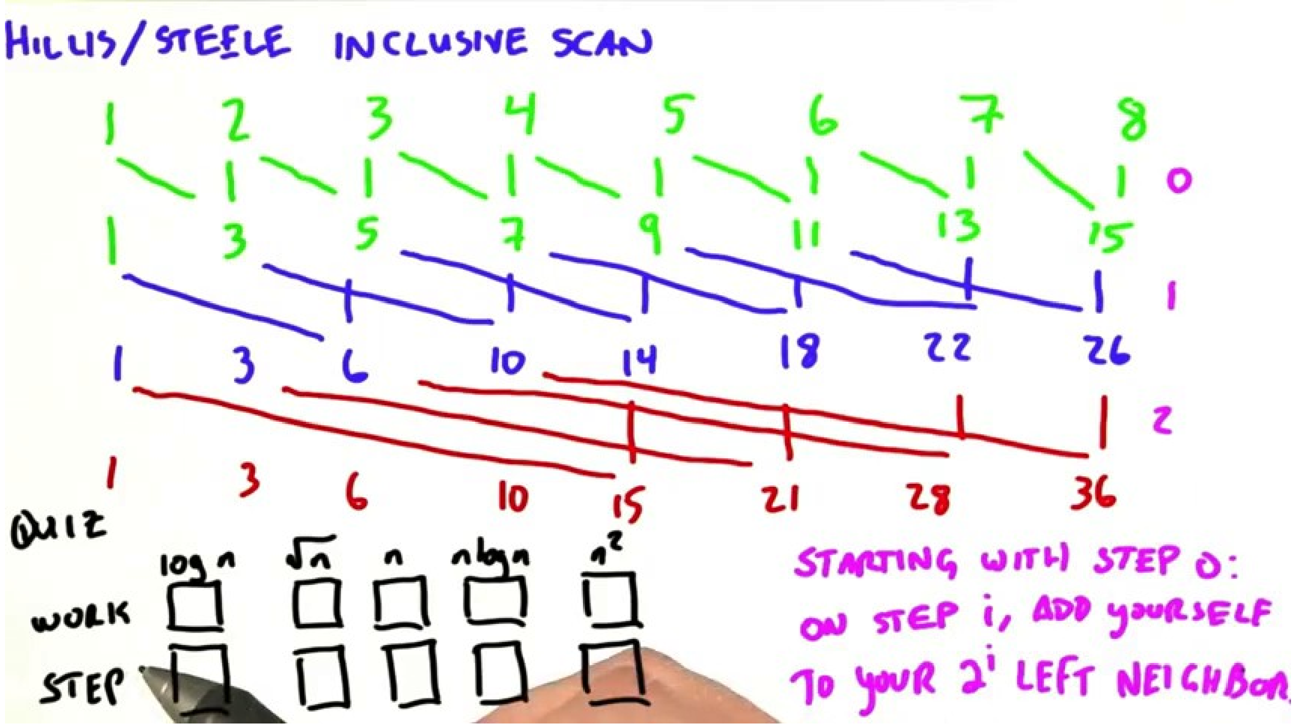

INCLUSIVE SCAN EXAMPLE, REVISITED
IN: [3 1 4 1 5 9]
OUT: [3 4 8 9 14 23]

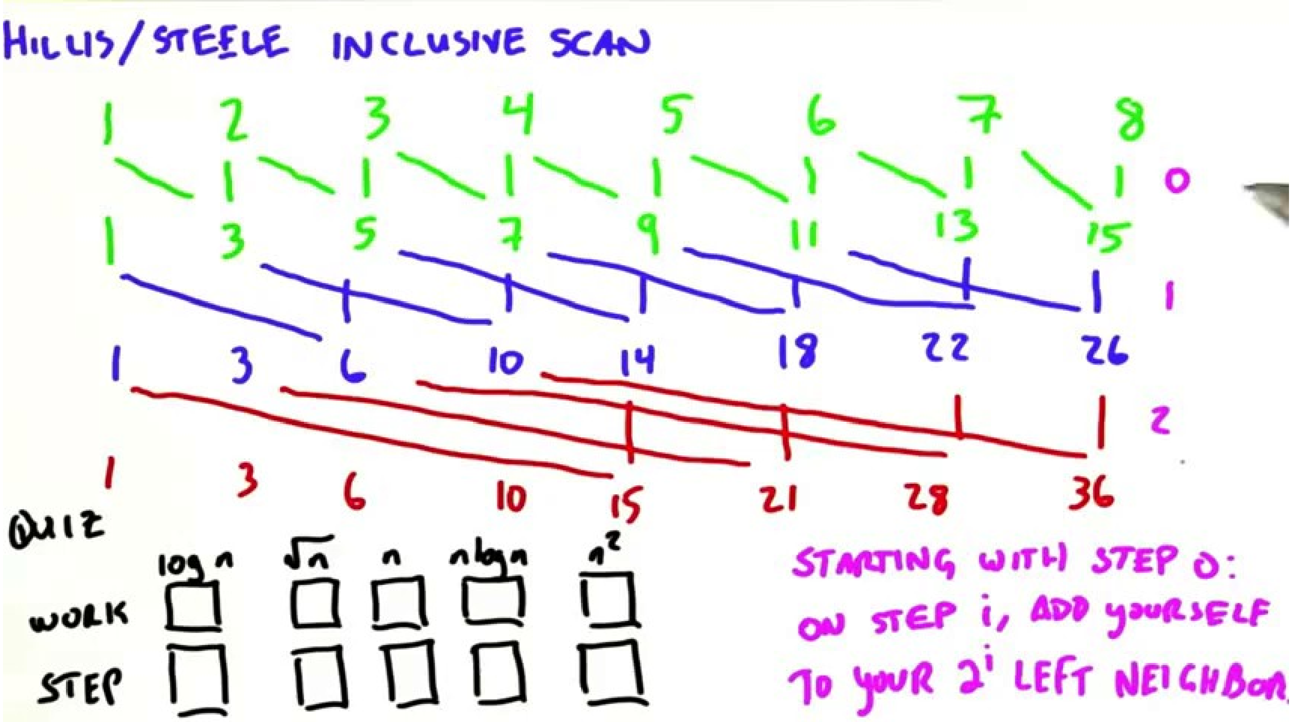

INCLUSIVE SCAN EXAMPLE, REVISITED
IN: [3 1 4 1 5 9]
OUT: [3 4 8 9 14 23]
QUIZ
CONSTANT
O(1)
O(log n)
LINEAR
O(n)
O(n²)
STEPS?
WORK?

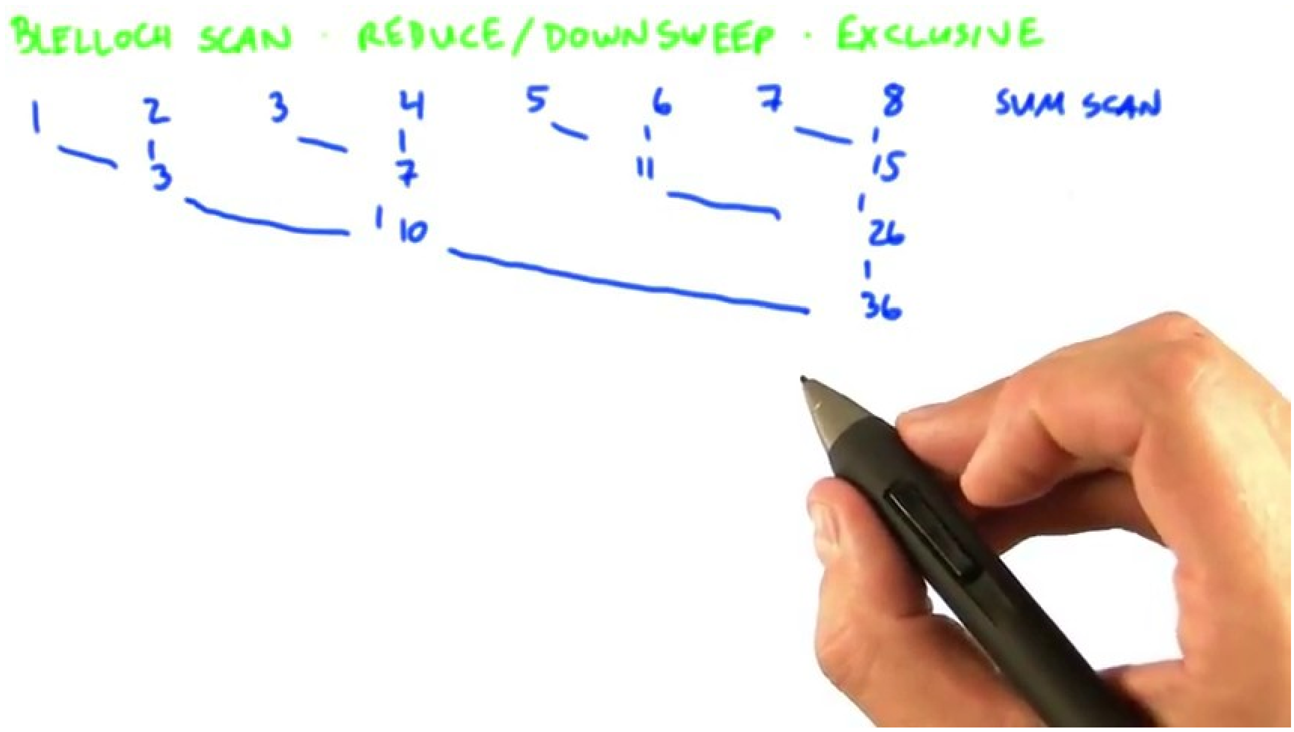
TWO PARALLEL SCAN ALGORITHMS

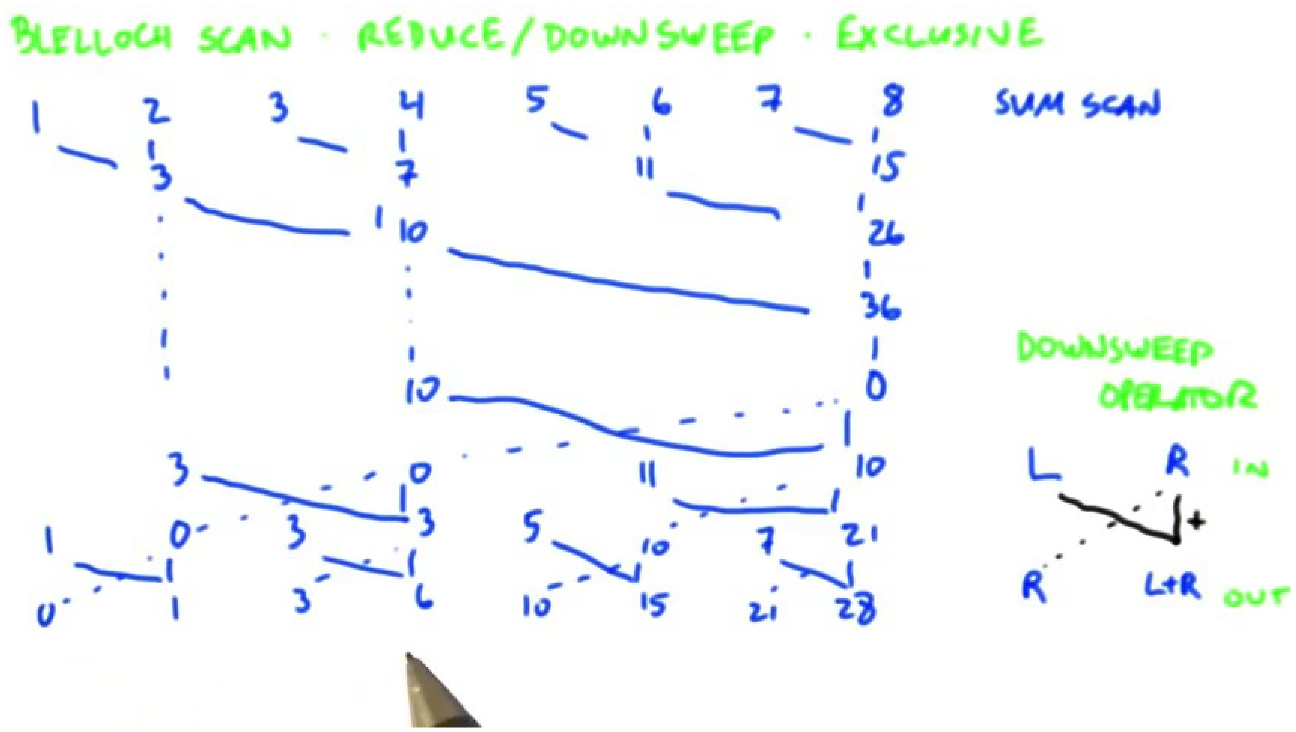
	MORE STEP-EFFICIENT	MORE WORK-EFFICIENT
HILLIS + STEELE	X	
BLELLOCH		X


HILLIS/STEELE INCLUSIVE SCAN
1 2 3 4 5 6 7 8
1 1 1 1 1 1 1
1 3 5 7 9 11 13 15


HILLIS/STEELE INCLUSIVE SCAN
1 2 3 4 5 6 7 8
0
1 3 5 7 9 11 13 15
1
1 3 6 10 14 18 22 26
2
1 3 6 10 15 21 28 36
QUIZ
log n
√n
n
n log n
n²
WORK
STEP
STARTING WITH STEP 0:
ON STEP i, ADD YOURSELF
TO YOUR 2^i LEFT NEIGHBOR.

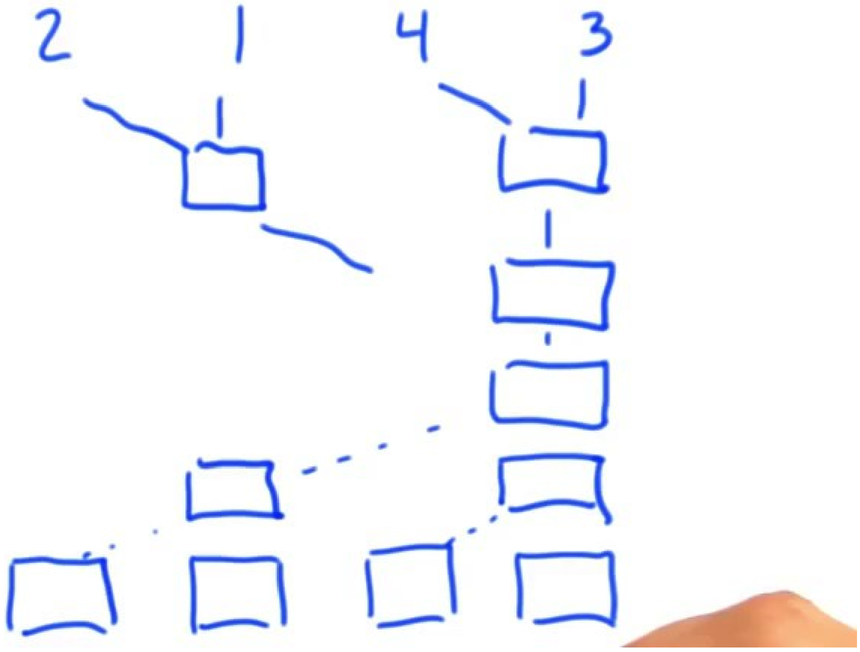

HILLIS/STEELE INCLUSIVE SCAN
1 2 3 4 5 6 7 8
0
1 3 5 7 9 11 13 15
1
1 3 6 10 14 18 22 26
2
1 3 6 10 15 21 28 36
QUIZ
log n
√n
n
n log n
n²
WORK
STEP
STARTING WITH STEP 0:
ON STEP i, ADD YOURSELF
TO YOUR 2^i LEFT NEIGHBOR

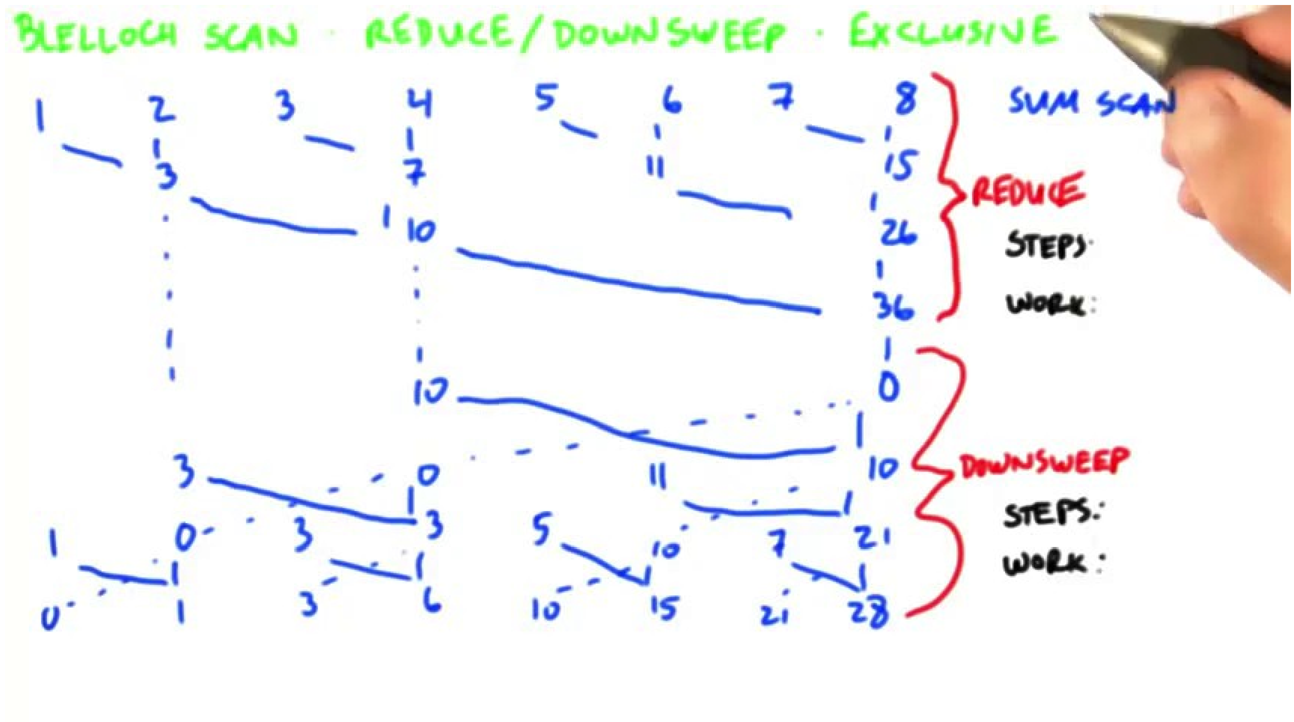

BLELLOCH SCAN · REDUCE/DOWNSWEEP · EXCLUSIVE
1 2 3 4 5 6 7 8
SUM SCAN
3 7 11 15
10 26
36

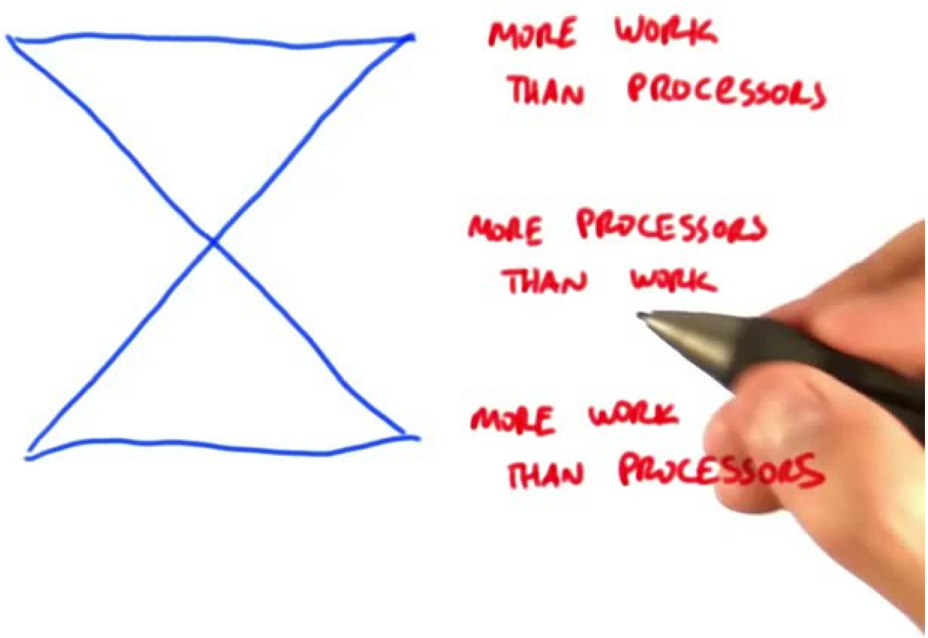

BLELLOCH SCAN · REDUCE / DOWNSWEEP · EXCLUSIVE
SUM SCAN
1 2 3 4 5 6 7 8
3 7 11 15
10 26
36
0
10 0
3 0 11 10
1 0 3 3 5 10 7 21
0 1 3 6 10 15 21 28
DOWNSWEEP OPERATOR
L R IN
R L+R OUT

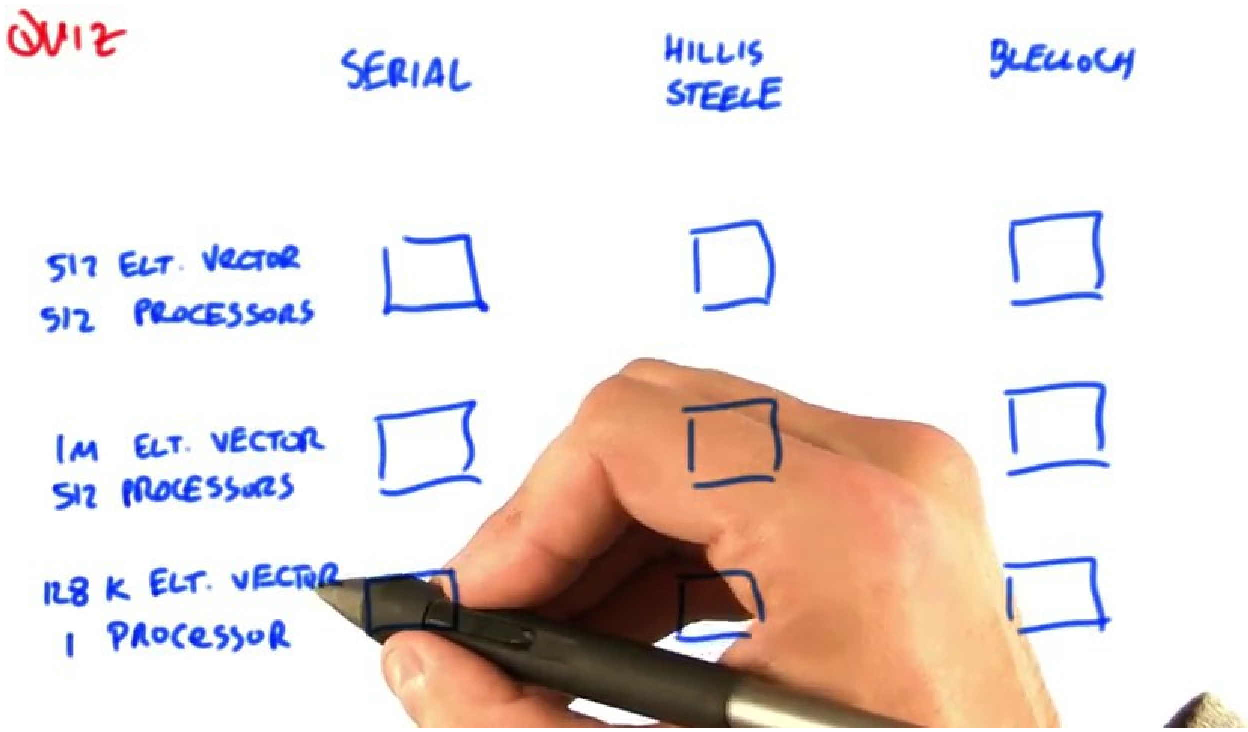
QUIZ

MAX SCAN USING REDUCE/DOWNSWEEP

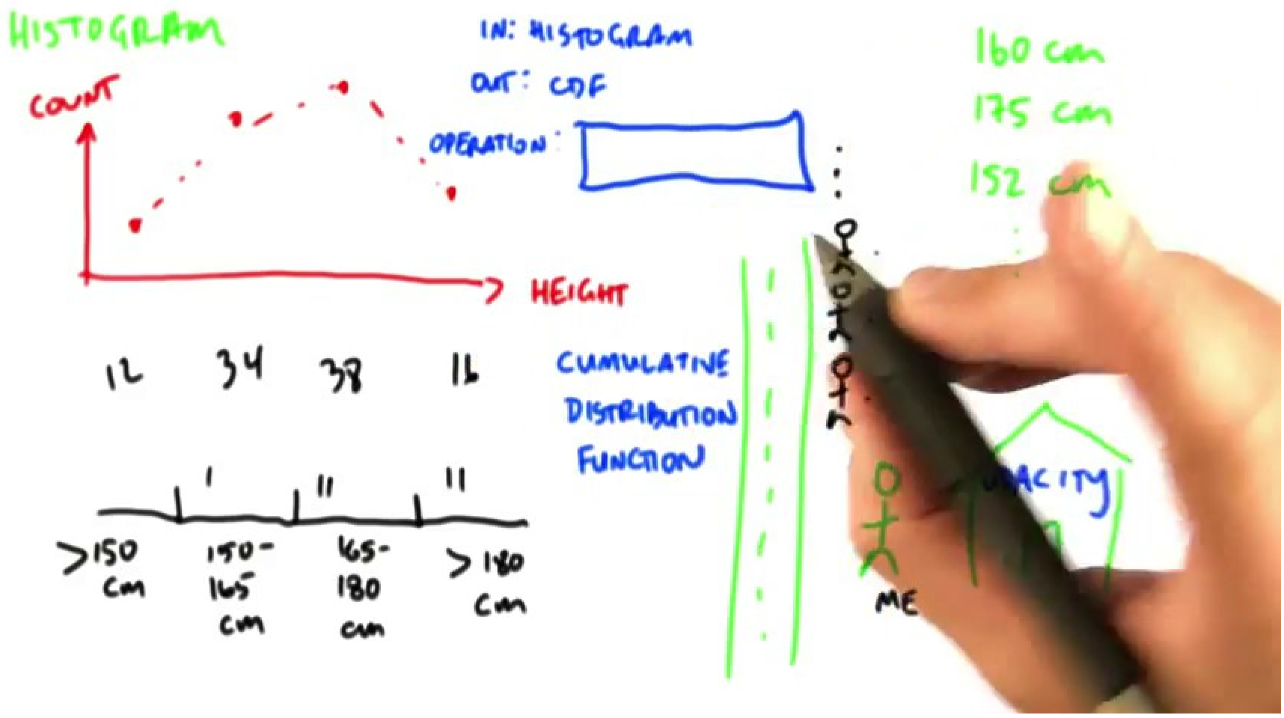


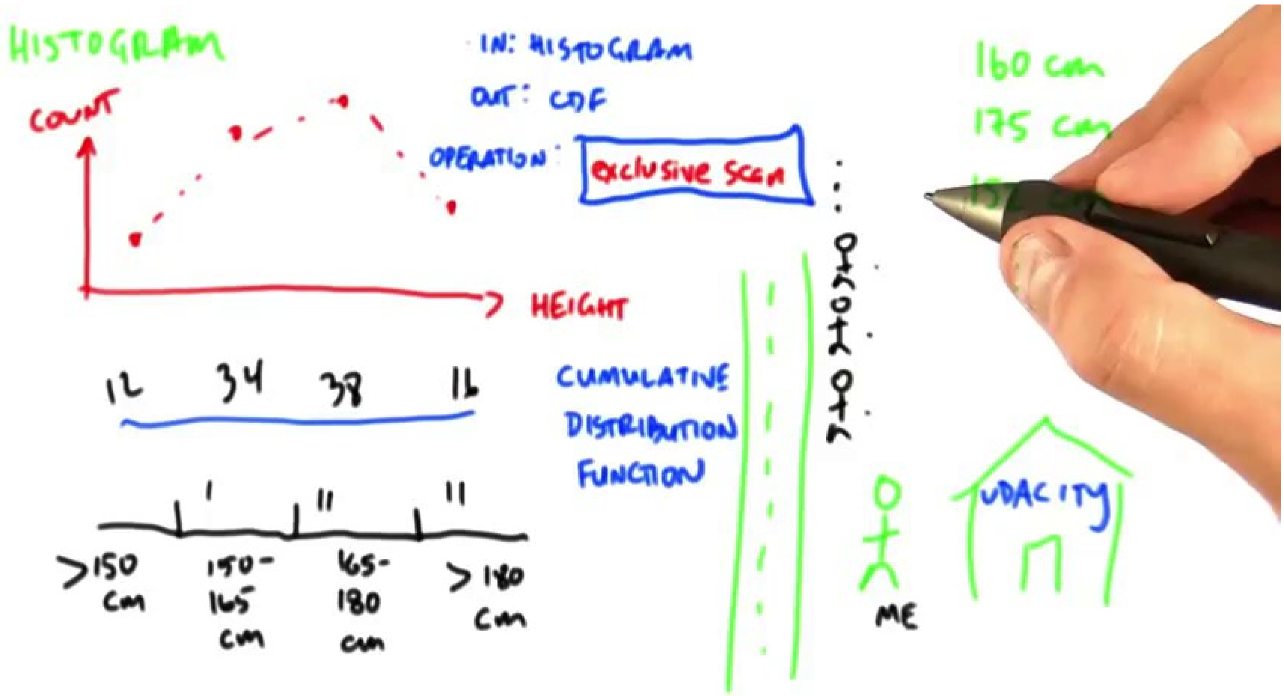

BLELLOCH SCAN · REDUCE/DOWNSWEEP · EXCLUSIVE
1 2 3 4 5 6 7 8
3 7 11 15
10 26
36
SUM SCAN
REDUCE
STEPS:
WORK:
10 0
3 0 11 10
1 0 3 3 5 10 7 21
0 1 3 6 10 15 21 28
DOWNSWEEP
STEPS:
WORK:


MORE WORK
THAN PROCESSORS
MORE PROCESSORS
THAN WORK
MORE WORK
THAN PROCESSORS


QUIZ
SERIAL
HILLIS STEELE
BLELLOCH
512 ELT. VECTOR
512 PROCESSORS
1M ELT. VECTOR
512 PROCESSORS
128 K ELT. VECTOR
1 PROCESSOR

HISTOGRAM


HISTOGRAM
COUNT
HEIGHT
12
34
38
16
>150 cm
150-165 cm
165-180 cm
>180 cm
IN: HISTOGRAM
OUT: CDF
OPERATION:
CUMULATIVE DISTRIBUTION FUNCTION
160 cm
175 cm
152 cm
ME


HISTOGRAM
COUNT
HEIGHT
12 34 38 16
>150 cm
150-165 cm
165-180 cm
>180 cm
IN: HISTOGRAM
OUT: CDF
OPERATION:
exclusive scan
CUMULATIVE DISTRIBUTION FUNCTION
160 cm
175 cm
ME
UDACITY

SERIAL ALGORITHM : HISTOGRAM

```
for (i=0; i<BIN_COUNT; i++)
    result[i]=0;
for (i=0; i<BIN_COUNT; i++)
    result[computeBin(measurements[i])]++;
```

↑ TO WHICH BIN DOES THIS MEASUREMENT BELONG?

INPUT:
155
150
175
170

0	<150
0	150-165
0	165-180
0	>180

SERIAL ALGORITHM : HISTOGRAM

```
for (i=0; i<BIN_COUNT; i++)
    result[i] = 0;
for (i=0; i<BIN_COUNT; i++)
    result[computeBin(measurements[i])]++;
```

TO WHICH BIN DOES THIS MEASUREMENT BELONG?

INPUT:
155
150
175
170

0	<150
2	150-165
2	165-180
0	>180

QUIZ

n measurements

b bins

MAXIMUM # OF MEASUREMENTS/BIN $\boxed{}$

AVERAGE # OF MEASUREMENTS/BIN $\boxed{}$

SERIAL ALGORITHM: HISTOGRAM

```
for (i=0; i< BIN_COUNT; i++)
    result[i] = 0;
for (i=0; i< measurements.size(); i++)
    result[computeBin(measurements[i])]++;
```

```
{
    int r = 0;
    for (int i = 0; i < bits; i++)
    {
        int bit = (w & (1 << i)) >> i;
        r |= bit << (bits - i - 1);
    }
    return r;
}

__global__ void naive_histo(int *d_bins, const int *d_in, const int BIN_COUNT)
{
    int myId = threadIdx.x + blockDim.x * blockIdx.x;
    int myItem = d_in[myId];
    int myBin = myItem % BIN_COUNT;
    d_bins[myBin]++;
}

__global__ void simple_histo(int *d_bins, const int *d_in, const int BIN_COUNT)
{
    int myId = threadIdx.x + blockDim.x * blockIdx.x;
    int myItem = d_in[myId];
```

-:--- histo.cu   6% L29   (C++/l Abbrev)

WHY THE OBVIOUS METHOD DOESN'T WORK

$\boxed{5}$ BIN

THREAD 1 2

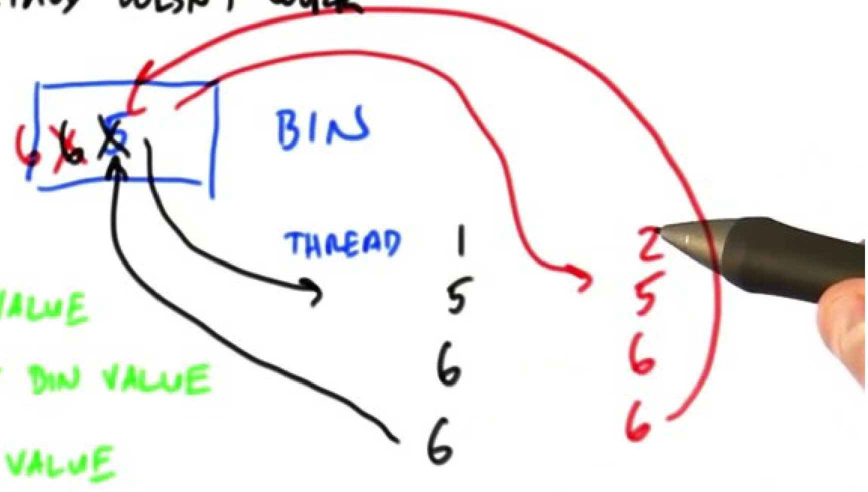
THREAD:

1. READ BIN VALUE
2. INCREMENT BIN VALUE
3. WRITE BIN VALUE

WHY THE OBVIOUS METHOD DOESN'T WORK

THREAD:

1. READ BIN VALUE
2. INCREMENT BIN VALUE
3. WRITE BIN VALUE



RACE CONDITION

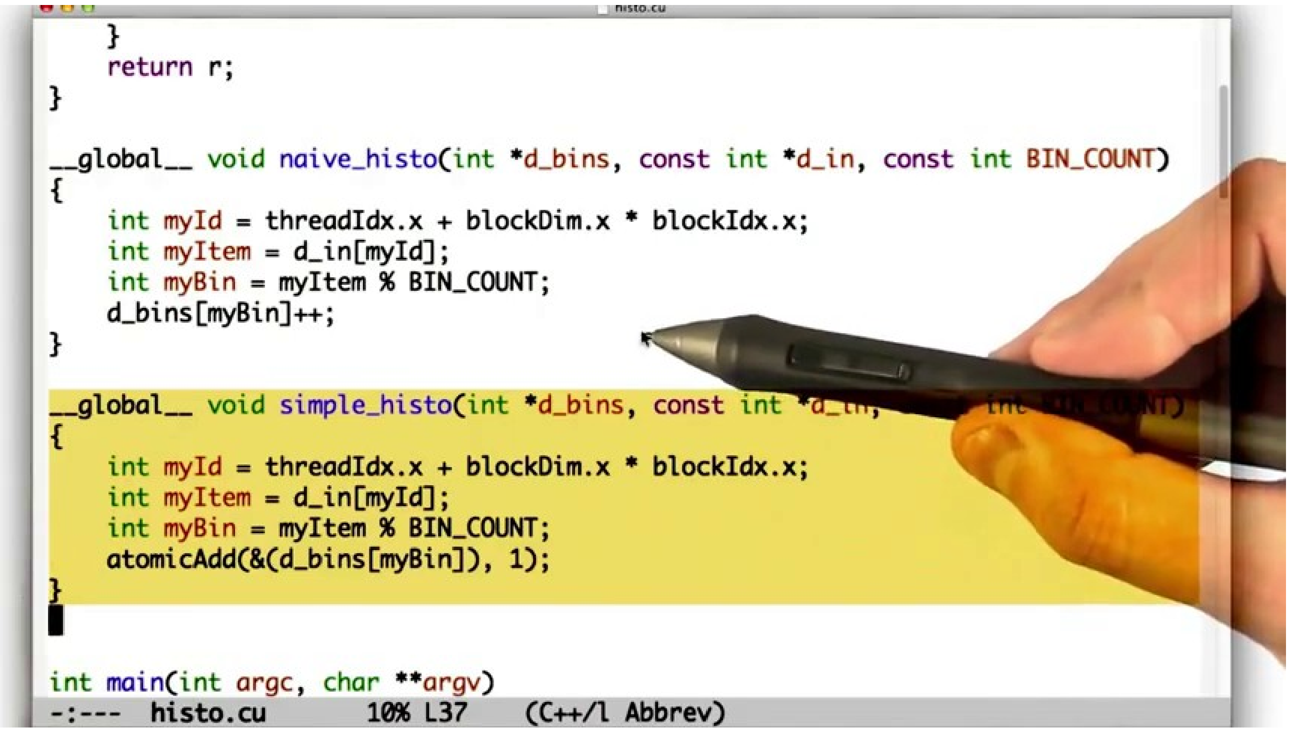
METHOD 1: ACCUMULATE USING ATOMICS

$\boxed{5}$ BIN

THREAD 1 2

THREAD:

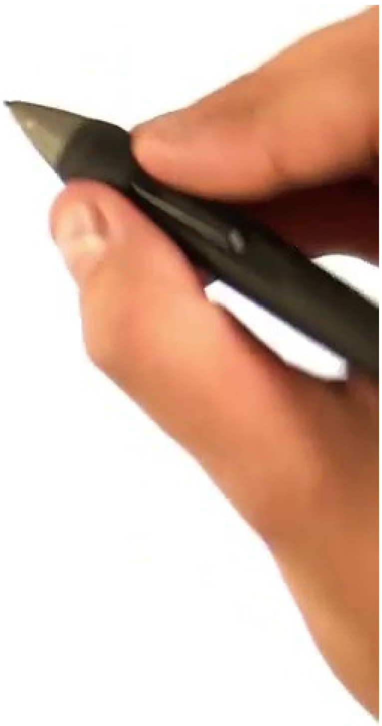
$$\left.\begin{array}{l}\text{(1) READ BIN VALUE}\\ \text{(2) INCREMENT BIN VALUE}\\ \text{(3) WRITE BIN VALUE}\end{array}\right\}\text{RMW}$$


histo.cu
}
return r;
}
__global__ void naive_histo(int *d_bins, const int *d_in, const int BIN_COUNT)
{
int myId = threadIdx.x + blockDim.x * blockIdx.x;
int myItem = d_in[myId];
int myBin = myItem % BIN_COUNT;
d_bins[myBin]++;
}
__global__ void simple_histo(int *d_bins, const int *d_in, const int BIN_COUNT)
{
int myId = threadIdx.x + blockDim.x * blockIdx.x;
int myItem = d_in[myId];
int myBin = myItem % BIN_COUNT;
atomicAdd(&(d_bins[myBin]), 1);
}
int main(int argc, char **argv)
-:--- histo.cu 10% L37 (C++/l Abbrev)

QUIZ

- Histogram with 1M elements
- You can choose # of bins:

10 ☐ 100 ☐ 1000 ☐



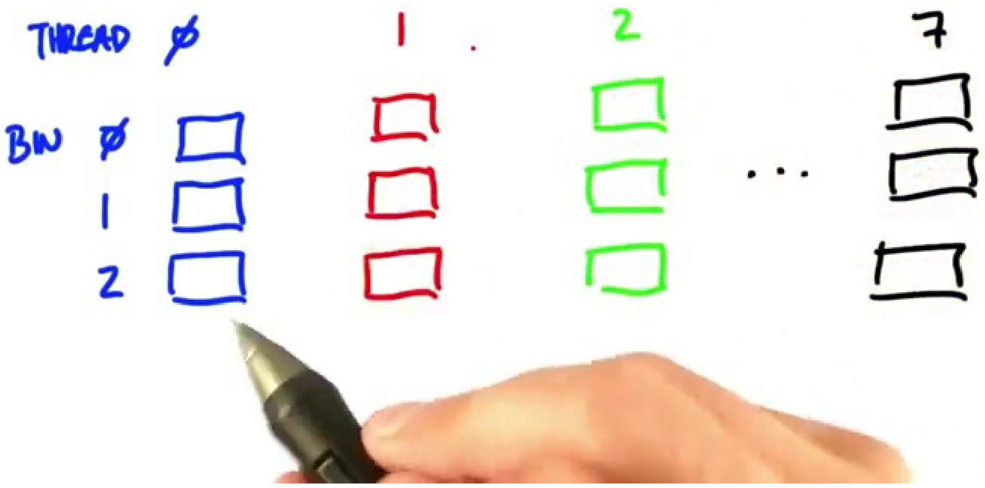
PER-THREAD PRIVATIZED (LOCAL) HISTOGRAMS, THEN REDUCE

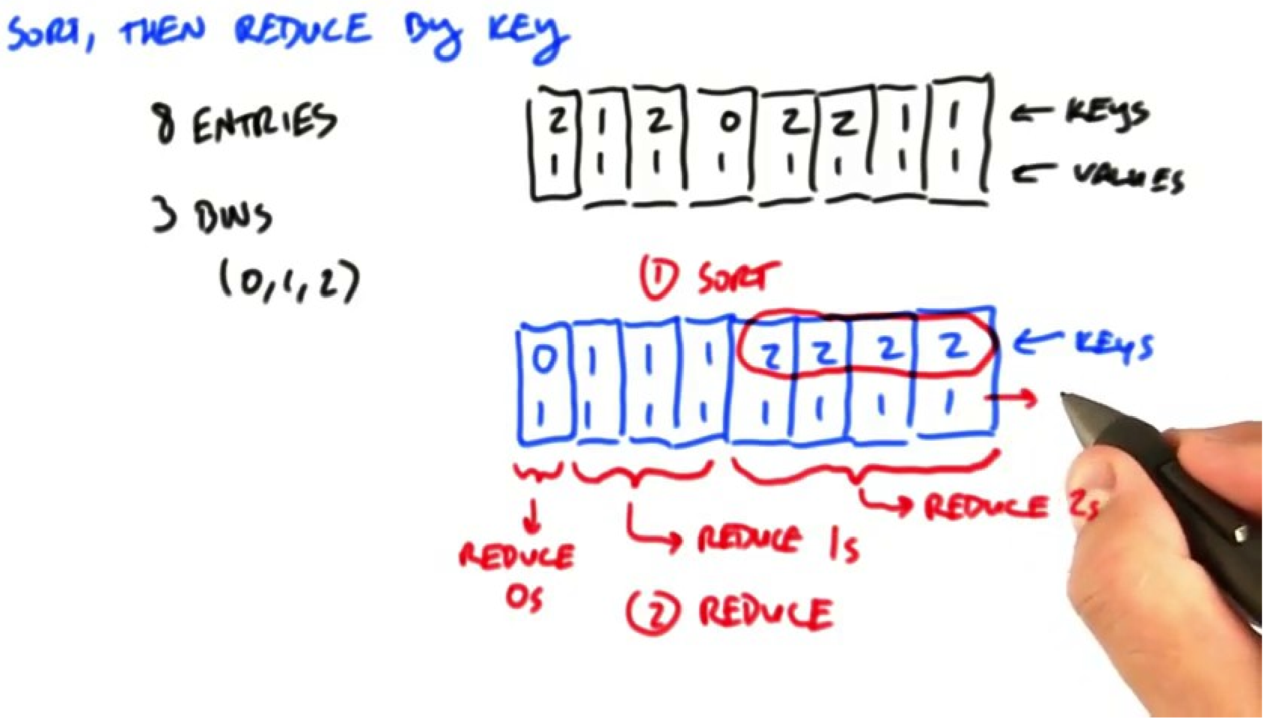
128 ITEMS · 8 THREADS · 3 BINS

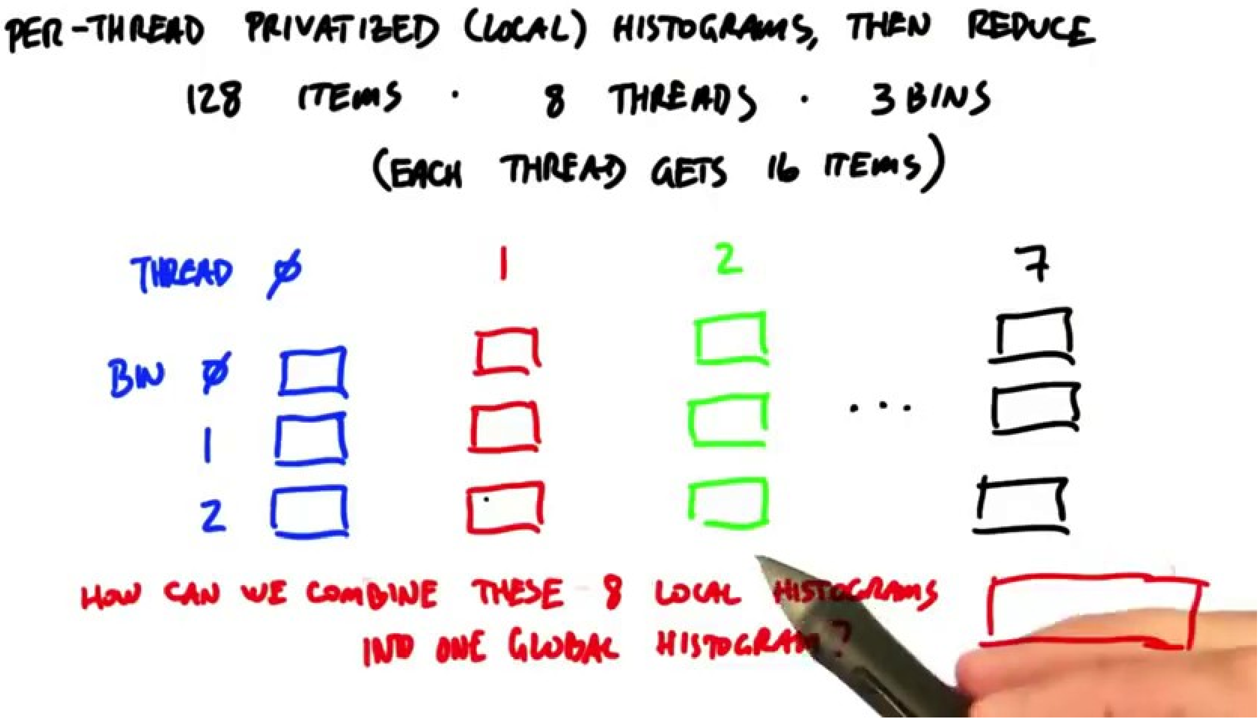
PER-THREAD PRIVATIZED (LOCAL) HISTOGRAMS, THEN REDUCE

128 ITEMS · 8 THREADS · 3 BINS

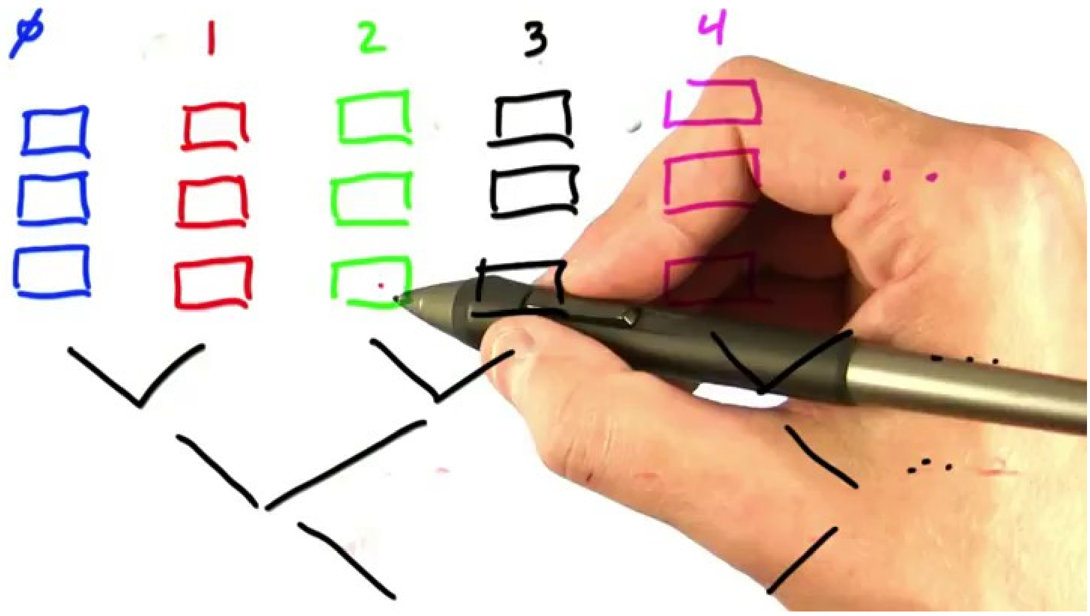
(EACH THREAD GETS 16 ITEMS)




SORT, THEN REDUCE BY KEY
8 ENTRIES
3 BINS
(0,1,2)
2 1 2 0 2 2 1 1
1 1 1 1 1 1 1 1
← KEYS
← VALUES
① SORT
0 1 1 1 2 2 2 2
1 1 1 1 1 1 1 1
← KEYS
REDUCE 0s
REDUCE 1s
REDUCE 2s
② REDUCE


PER-THREAD PRIVATIZED (LOCAL) HISTOGRAMS, THEN REDUCE
128 ITEMS · 8 THREADS · 3 BINS
(EACH THREAD GETS 16 ITEMS)
THREAD ∅
1
2
7
BIN ∅
1
2
. . .
HOW CAN WE COMBINE THESE 8 LOCAL HISTOGRAMS
INTO ONE GLOBAL HISTOGRAM?

REDUCING 8 LOCAL HISTOGRAMS



FINAL THOUGHTS ON HISTOGRAM

- ATOMICS (2)
- PER-THREAD HISTOGRAMS, THEN REDUCE (1)
- SORT, THEN REDUCE BY KEY

HOW MANY ATOMIC ADDS?

256 THREADS, 8 BINS:

ATOMIC TECHNIQUE: $\boxed{}$

REDUCE TO 8-ELEMENT HISTOGRAM THEN ATOMICS $\boxed{}$