Inside Visual C++'s Parallel Algorithms

What happens when you throw std::execution::par…

https://github.com/BillyONeal/InsideParallelAlgorithms

Billy O'Neal

Sr. SDE / Standard Library Maintainer @ Microsoft

bion@Microsoft.com @MalwareMinigun

Hello Parallel World – Not This Talk

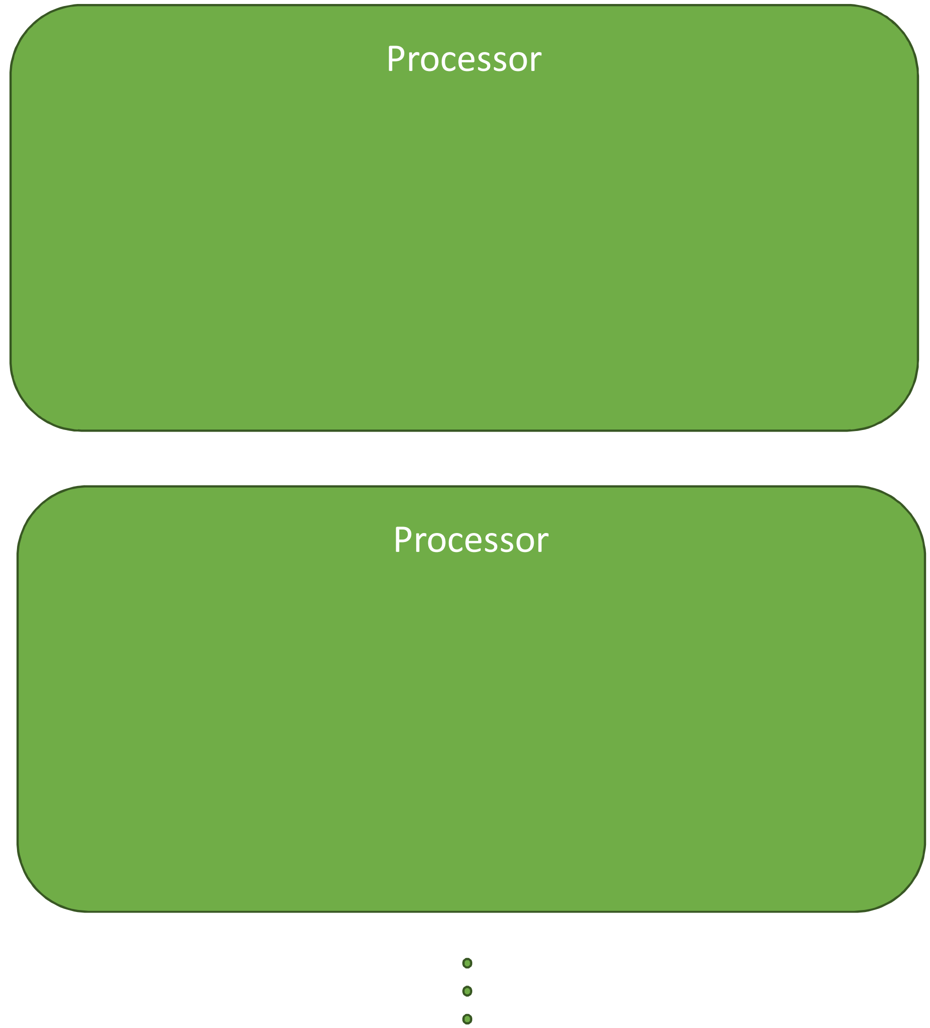
- See blog post https://blogs.msdn.microsoft.com/vcblog/2018/09/11/using-c17-parallel-algorithms-for-better-performance/
- std::sort(a, b) -> std::sort(std::execution::par, a, b)
- 75ms -> 20ms (1000000 doubles on 7980XE)

So you want to parallelize an algorithm….

- Partitioning (not std::partition!)
- Schedule on compute resources
- Merge results

Let’s look at accumulate…

X = X + *It++
X = X + *It++
X = X + *It++
X = X + *It++
X = X + *It++
X = X + *It++
X = X + *It++
X = X + *It++
X = X + *It++
X = X + *It++
X = X + *It++



Partitioning…

X = X + *It++
X = X + *It++
X = X + *It++

X = X + *It++
X = X + *It++
X = X + *It++

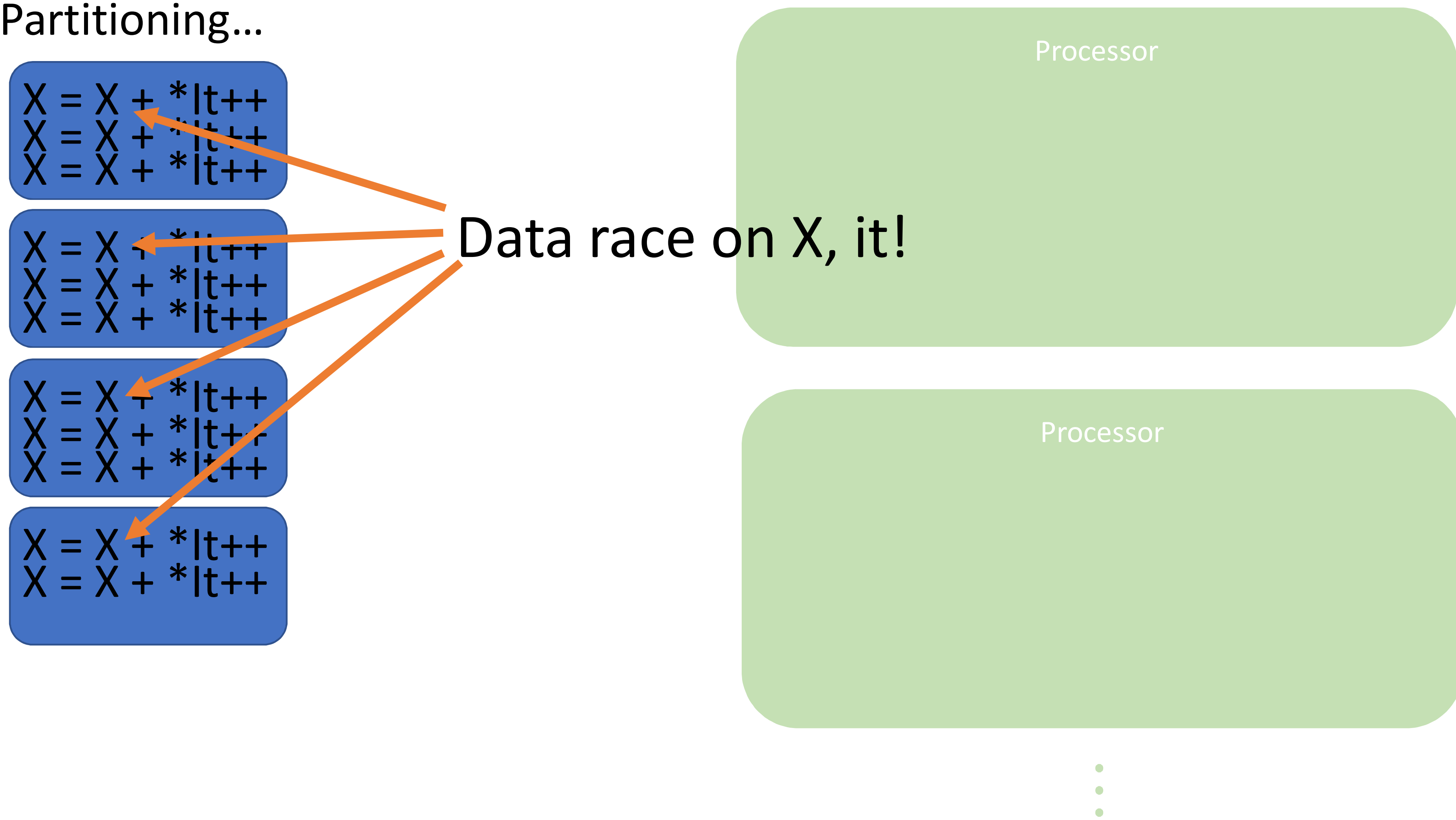
X = X + *It++
X = X + *It++
X = X + *It++

X = X + *It++
X = X + *It++

Processor

Processor

Partitioning…



Partitioning…

```
Ix = It
X = X + *Ix++
X = X + *Ix++
X = X + *Ix++
```

```
Ia = next(It, 3)
A = *Ia++ + *Ia++
A = A + *Ia++
```

```
Ib = next(It, 6)
B = *Ib++ + *Ib++
B = B + *Ib++
```

```
Ic = next(It, 9)
C = *Ic++ + *Ic++
```

Processor

Processor

Merge Results…

```
Ix = It
X = X + *Ix++
X = X + *Ix++
X = X + *Ix++
```

```
Ia = next(It, 3)
A = *Ia++ + *Ia++
A = A + *Ia++
```

```
Ib = next(It, 6)
B = *Ib++ + *Ib++
B = B + *Ib++
```

```
Ic = next(It, 9)
C = *Ic++ + *Ic++
```

```
(wait)
X = X + A
X = X + B
X = X + C
```

```
Ix = It
X = X + *Ix++
X = X + *Ix++
X = X + *Ix++
```

```
Ia = next(It, 3)
A = *Ia++ + *Ia++
A = A + *Ia++
```

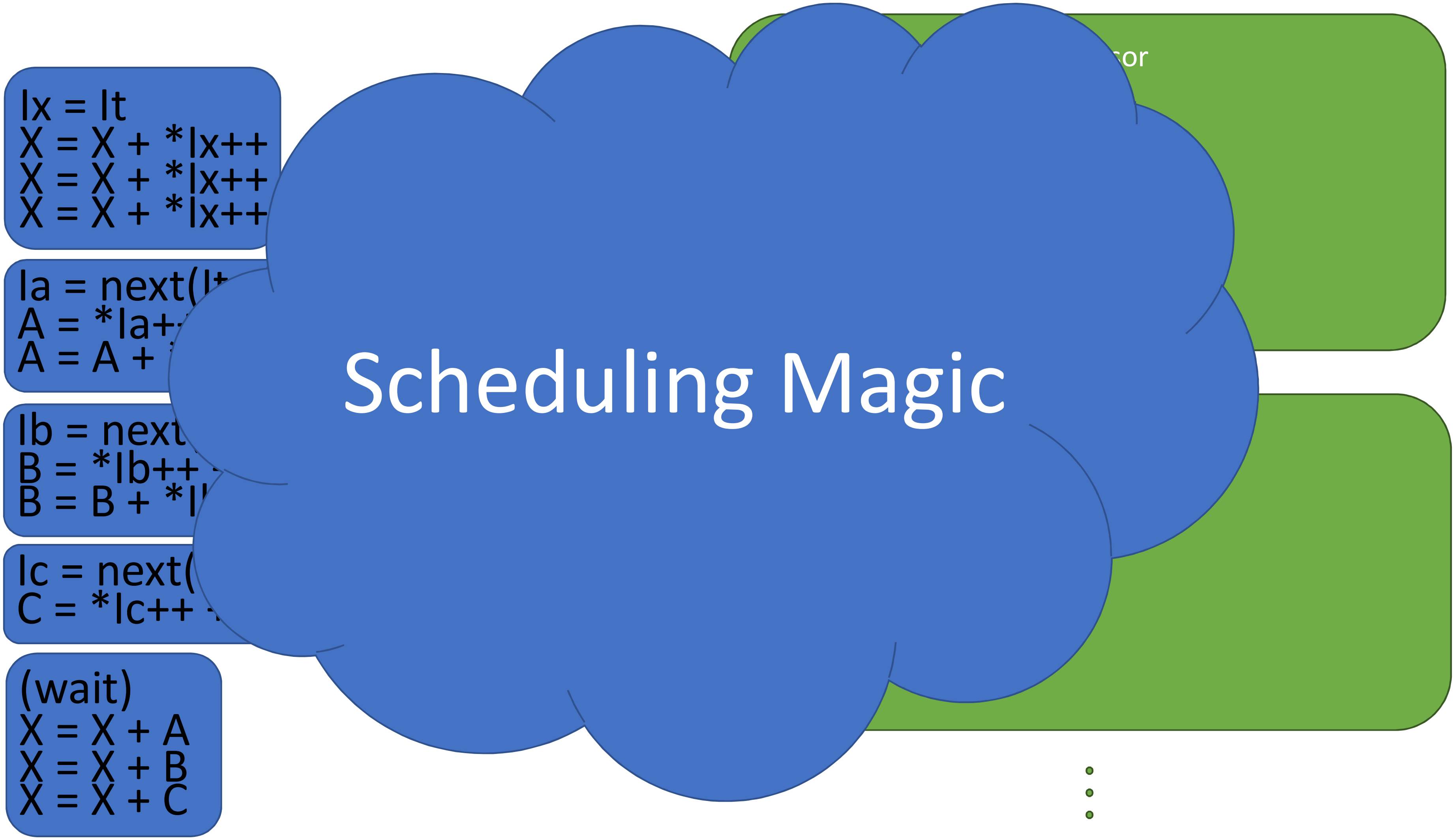
```
Ib = next(It, 6)
B = *Ib++ + *Ib++
B = B + *Ib++
```

```
Ic = next(It, 9)
C = *Ic++ + *Ic++
```

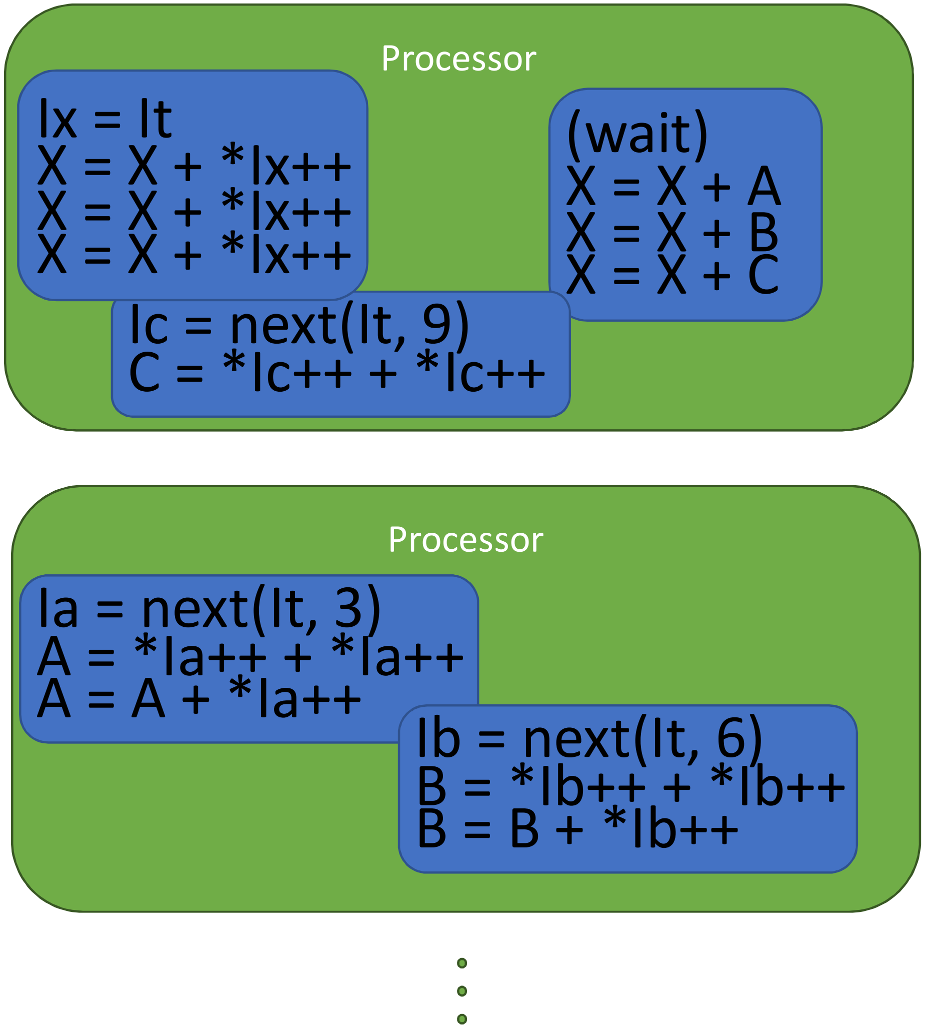
```
(wait)
X = X + A
X = X + C
X = X + B
```

Let’s look at ~~accumulate~~ reduce…

What happened to scheduling?



Ix = It
X = X + *Ix++
X = X + *Ix++
X = X + *Ix++
Ia = next(
A = *Ia+
A = A +
Ib = next
B = *Ib++
B = B + *I
Ic = next(
C = *Ic++
(wait)
X = X + A
X = X + B
X = X + C
Scheduling Magic



Processor
Ix = It
X = X + *Ix++
X = X + *Ix++
X = X + *Ix++
(wait)
X = X + A
X = X + B
X = X + C
Ic = next(It, 9)
C = *Ic++ + *Ic++
Processor
Ia = next(It, 3)
A = *Ia++ + *Ia++
A = A + *Ia++
Ib = next(It, 6)
B = *Ib++ + *Ib++
B = B + *Ib++

Scheduling, really – Windows' Thread Pool

- CreateThreadpoolWork
- SubmitThreadpoolWork
- WaitForThreadpoolWorkCallbacks
- CloseThreadpoolWork

- Pedro Teixeira's talk (Pedro works on the kernel) discusses threadpool internals

Questions on the mental model of reduce?

Demo – Debugging into std::reduce

Benchmark benchmark benchmark!

- The hardware and the input you care about are important
- Parallel algorithms generally do more work, even for for_each(random-access), to acquire threads, wait for background threads to complete, etc.
- Following numbers are from this ThinkPad X1 Carbon, i7-8650U 4c8t, 2133MHz DDR3L

Benchmark benchmark benchmark - Debug

- .\DemoReduce.exe 100000000 (762mb) – Parallel 4.9 times faster
- .\DemoReduce.exe 10000000 (76mb) – Parallel 4.6 times faster
- .\DemoReduce.exe 1000000 (7.6mb) – Parallel 3.8 times faster

(fits in cache on this chip below here)

- .\DemoReduce.exe 100000 (.76mb) – Parallel 1.5 times faster
- .\DemoReduce.exe 1000 – Parallel 13 times slower

Benchmark benchmark benchmark - Release

- .\DemoReduce.exe 100000000 (762mb) – Parallel 1.3 times faster
- .\DemoReduce.exe 10000000 (76mb) – Parallel 1.3 times faster
- .\DemoReduce.exe 1000000 (7.6mb) – Parallel 1.6 times faster

(fits in cache on this chip below here)

- .\DemoReduce.exe 100000 (.76mb) – Parallel 2 times slower
- .\DemoReduce.exe 1000 – Parallel 96 times slower

Demo - Why isn't it any faster?

Hardware matters!

- Next from the 7980XE 18c36t; 3200MHz DDR4

Benchmark benchmark benchmark - Debug

- .\DemoReduce.exe 100000000 (762mb) – Parallel 16.6 times faster
- .\DemoReduce.exe 10000000 (76mb) – Parallel 8.8 times faster
- (fits in cache on this chip below here)
- .\DemoReduce.exe 1000000 (7.6mb) – Parallel 3.8 times faster
- .\DemoReduce.exe 100000 (.76mb) – Parallel 2.8 times faster
- .\DemoReduce.exe 1000 – Parallel 15 times slower

Benchmark benchmark benchmark - Release

- .\DemoReduce.exe 100000000 (762mb) – Parallel 4.7 times faster
- .\DemoReduce.exe 10000000 (76mb) – Parallel 5.3 times faster
- (fits in cache on this chip below here)
- .\DemoReduce.exe 1000000 (7.6mb) – Parallel 2.1 times faster
- .\DemoReduce.exe 100000 (.76mb) – Parallel 3.9 times slower
- .\DemoReduce.exe 1000 – Parallel 157 times slower

Questions about Benchmarks?

A more “interesting” algorithm – stable_sort

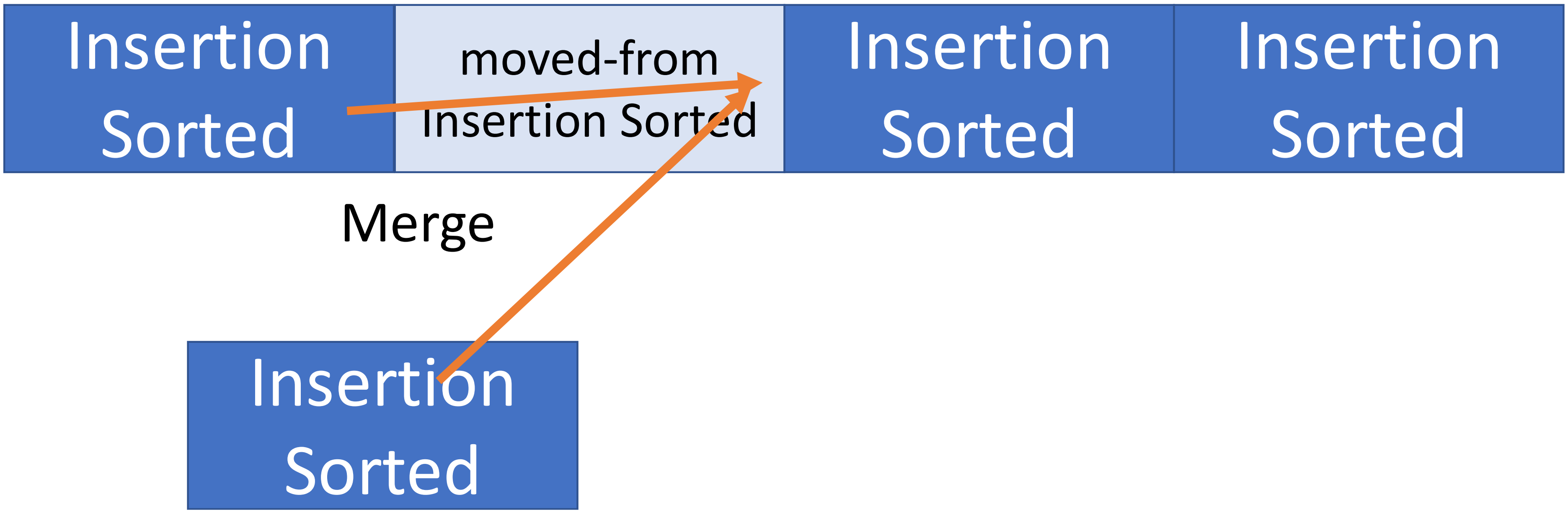
stable_sort

Unsorted

Insertion sort element chunks

Insertion Sorted	Insertion Sorted	Insertion Sorted	Insertion Sorted

inplace_merge elementwise



inplace_merge elementwise

Merged	Insertion Sorted	Insertion Sorted

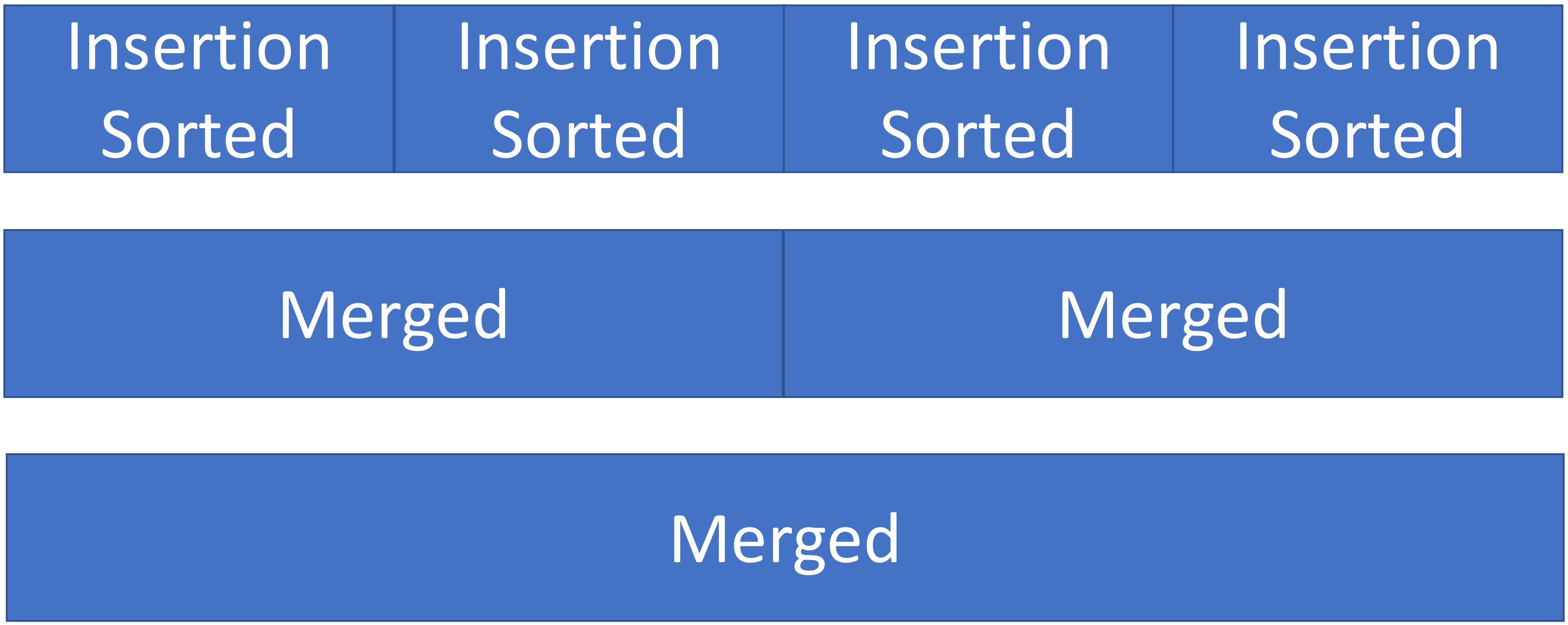
inplace_merge elementwise

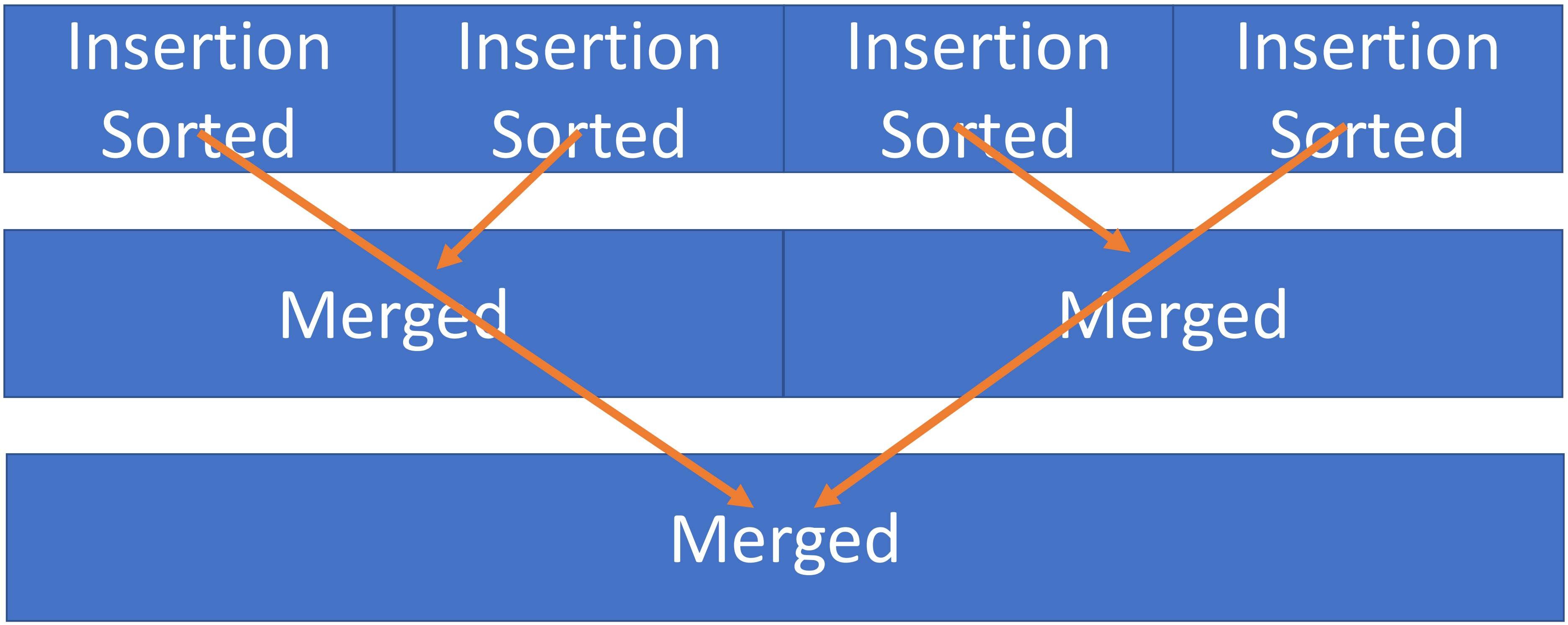
Merged	Merged

Worst case space consumption, n/2

Sorted

Merged


Insertion Sorted
Insertion Sorted
Insertion Sorted
Insertion Sorted
Merged
Merged
Merged


Insertion Sorted
Insertion Sorted
Insertion Sorted
Insertion Sorted
Merged
Merged
Merged

Get temporary space

Unsorted

Temp Space

Partition input + temp space together

Partition 0	Partition 1	Partition 2	Partition 3

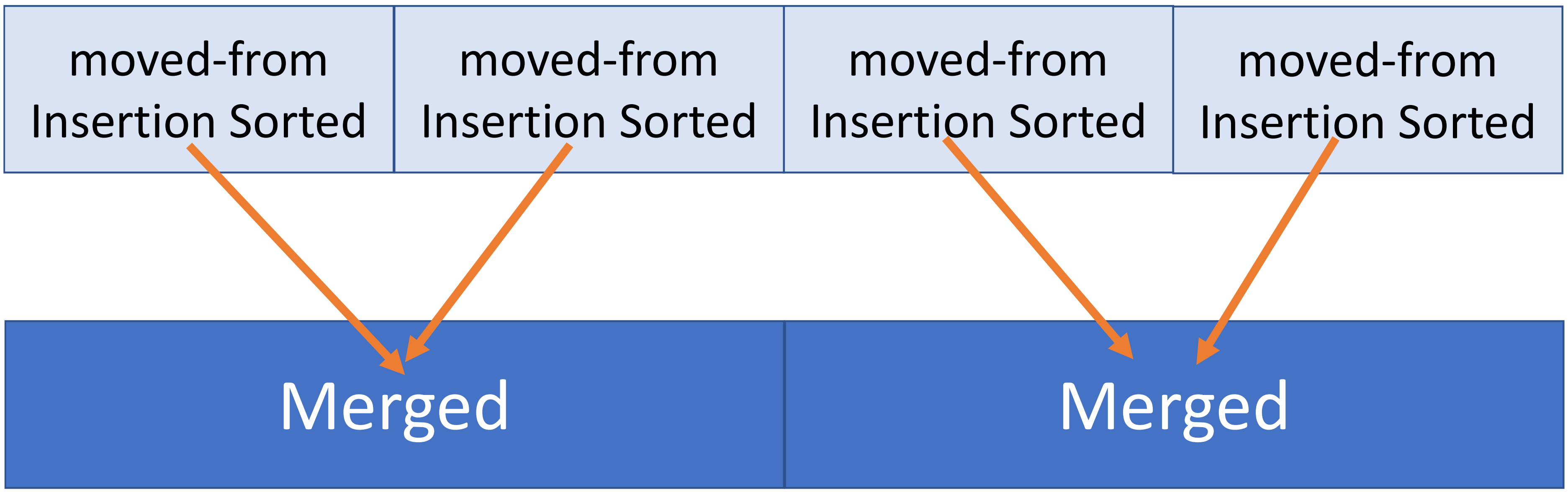
Temp Space	Temp Space	Temp Space	Temp Space

Insertion Sort – 4 Partitions

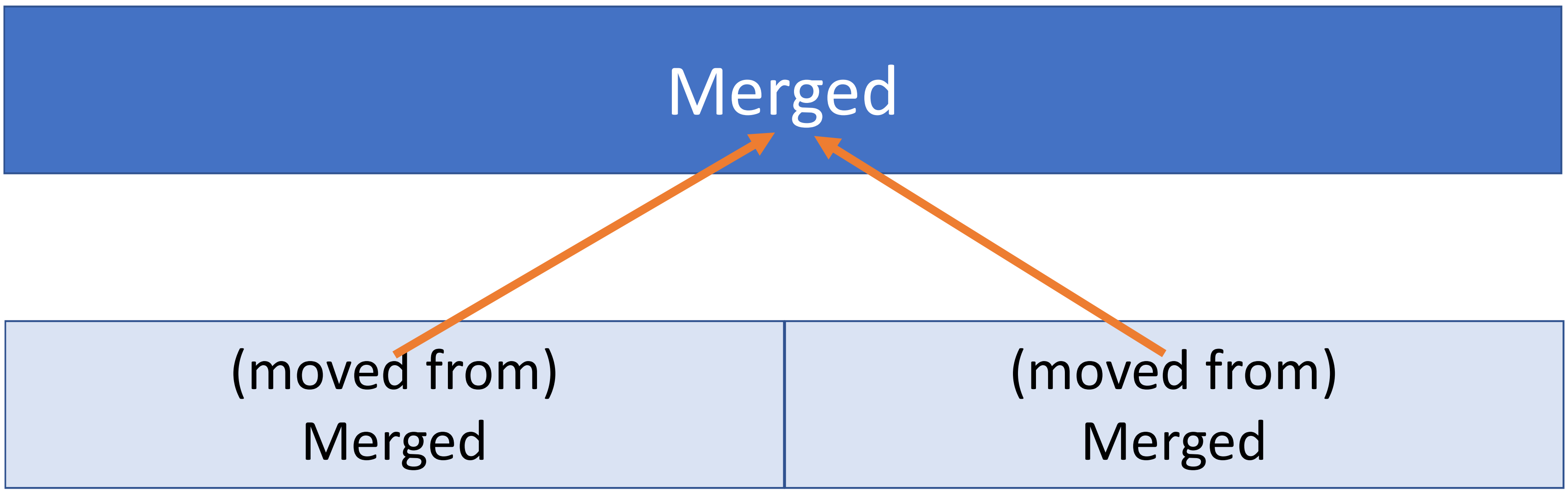
Insertion Sorted	Insertion Sorted	Insertion Sorted	Insertion Sorted

Temp Space

merge – 2 partitions

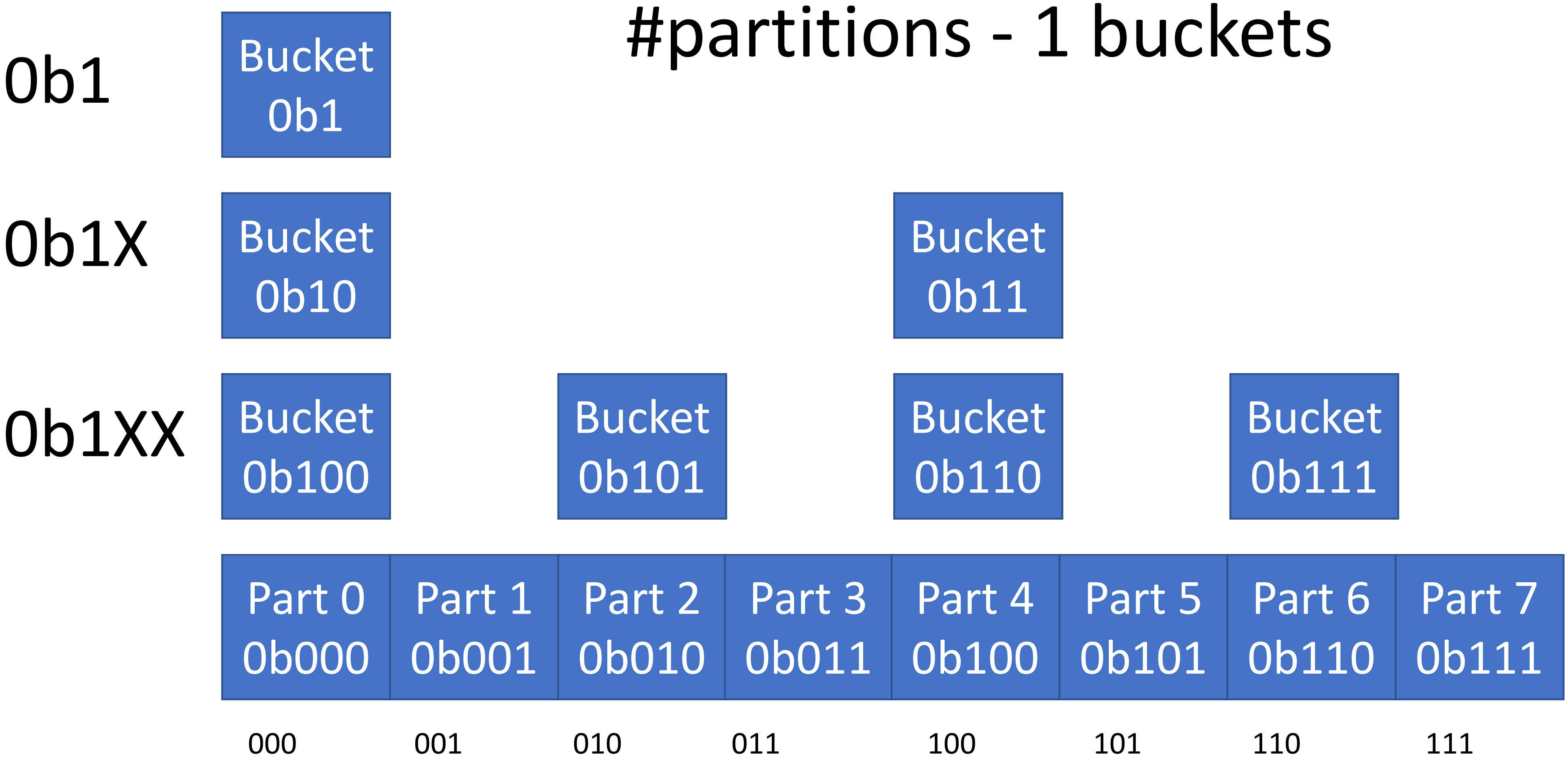


merge – 1 partition



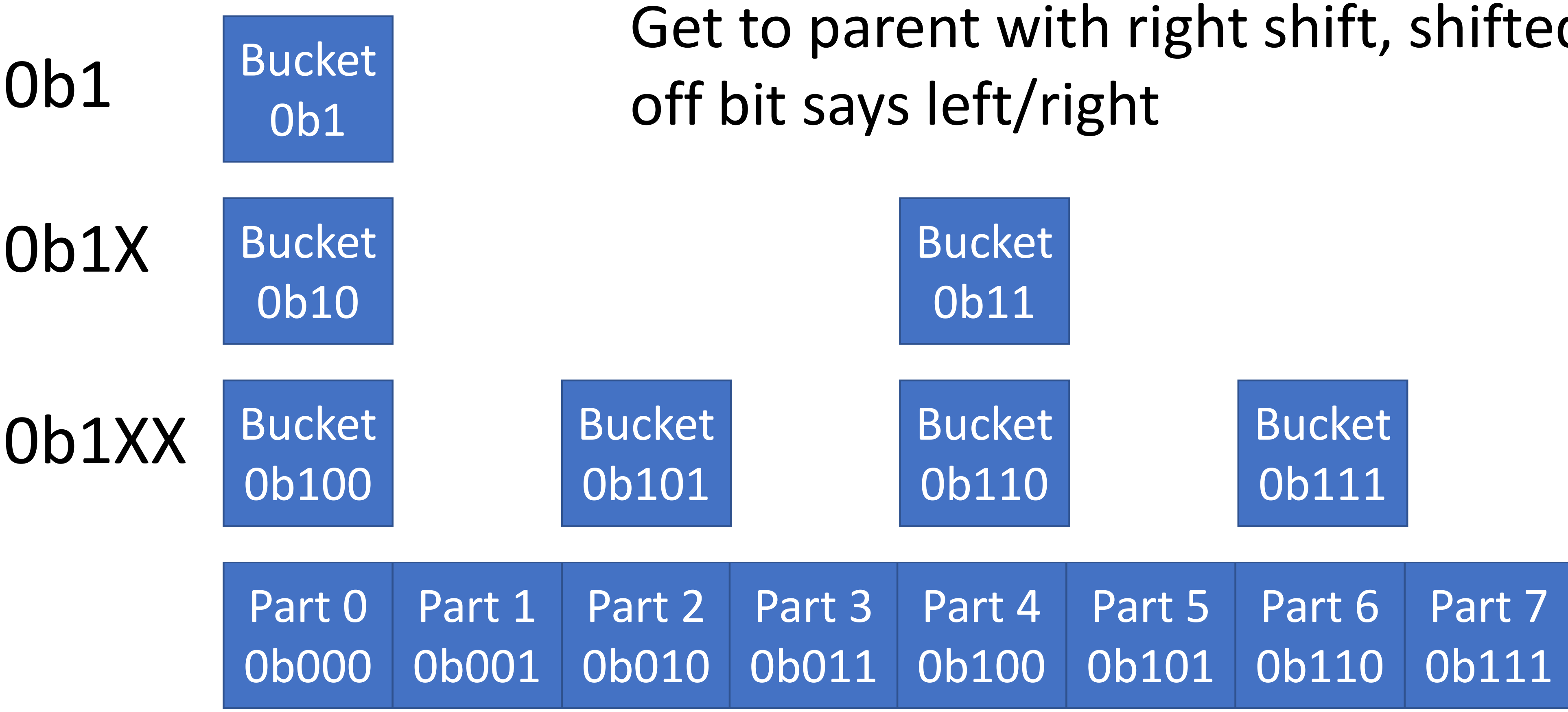
Bottom-up merge tree

#partitions - 1 buckets



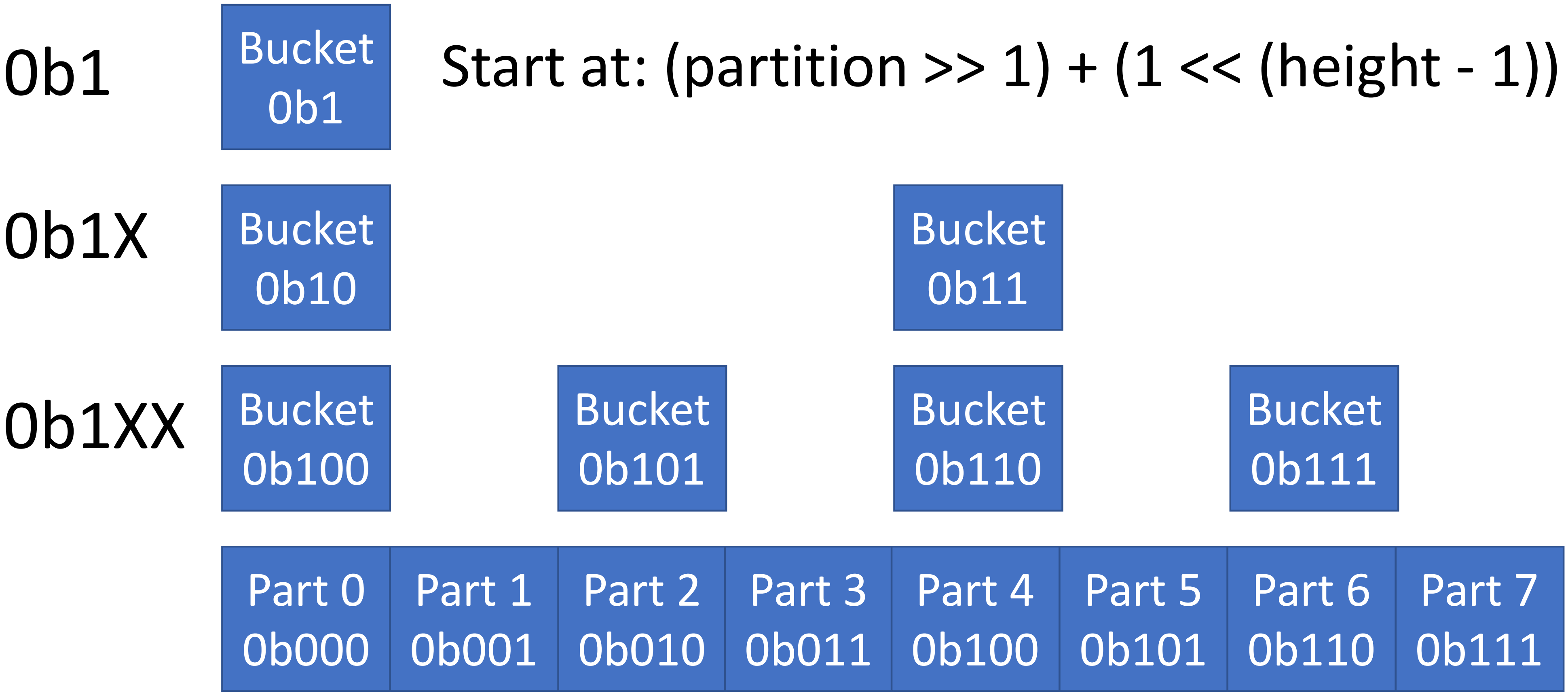
Bottom-up merge tree

Get to parent with right shift, shifted off bit says left/right



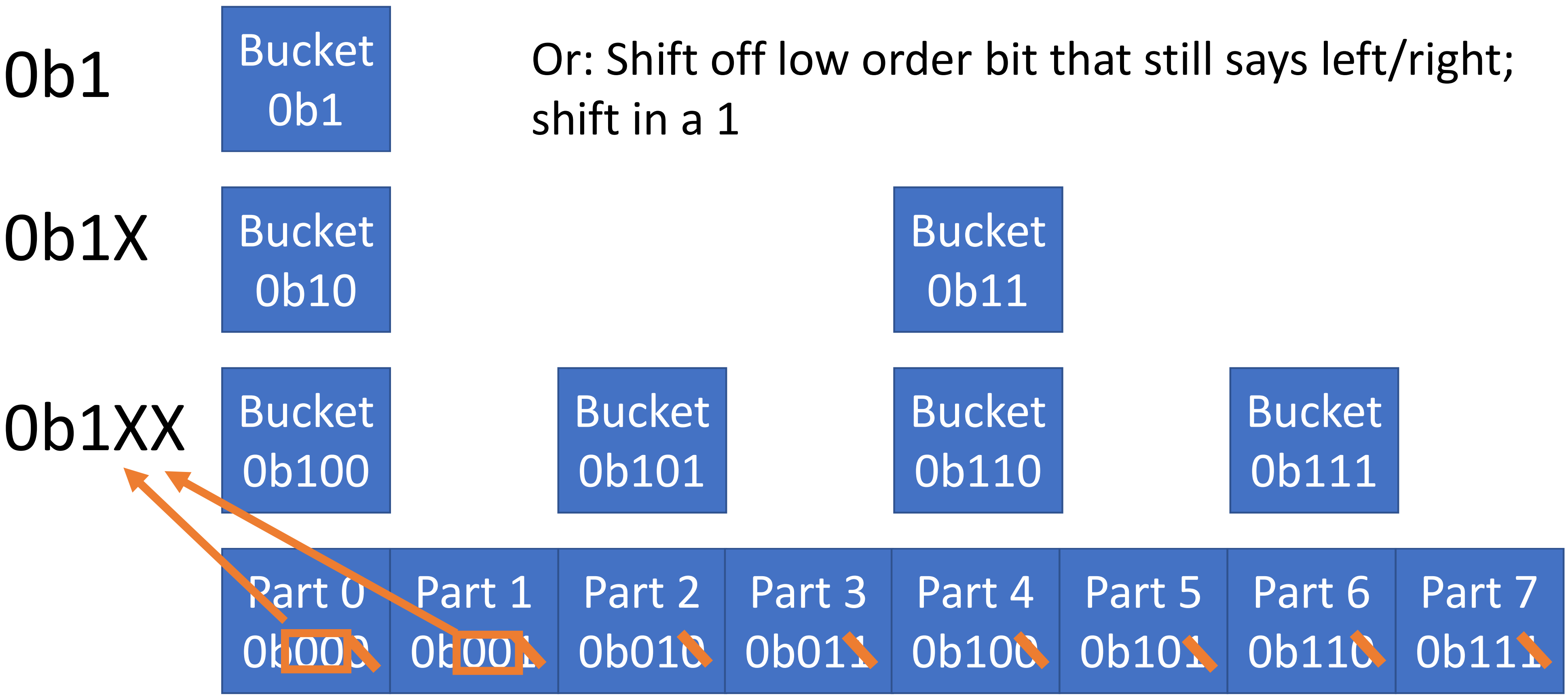
Bottom-up merge tree

Start at: (partition >> 1) + (1 << (height - 1))



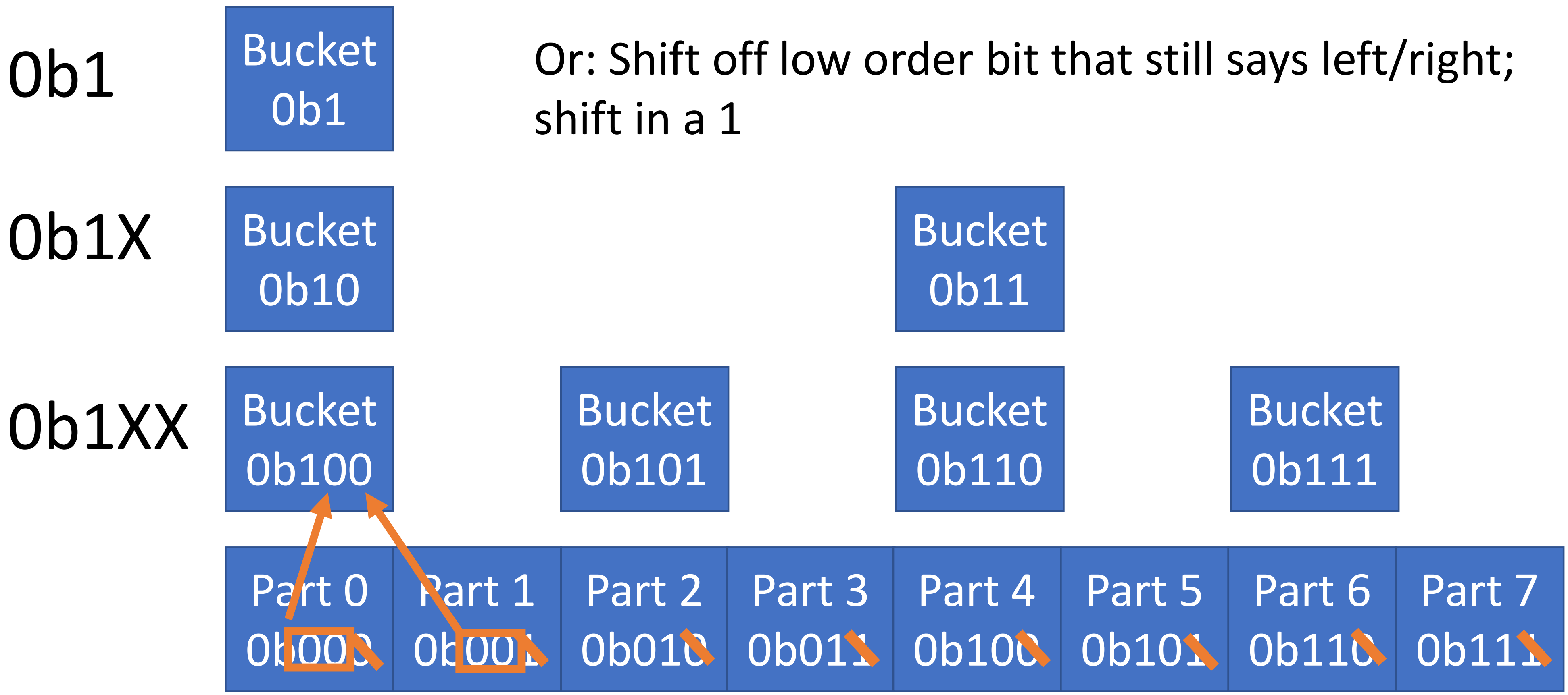
Bottom-up merge tree

Or: Shift off low order bit that still says left/right; shift in a 1



Bottom-up merge tree

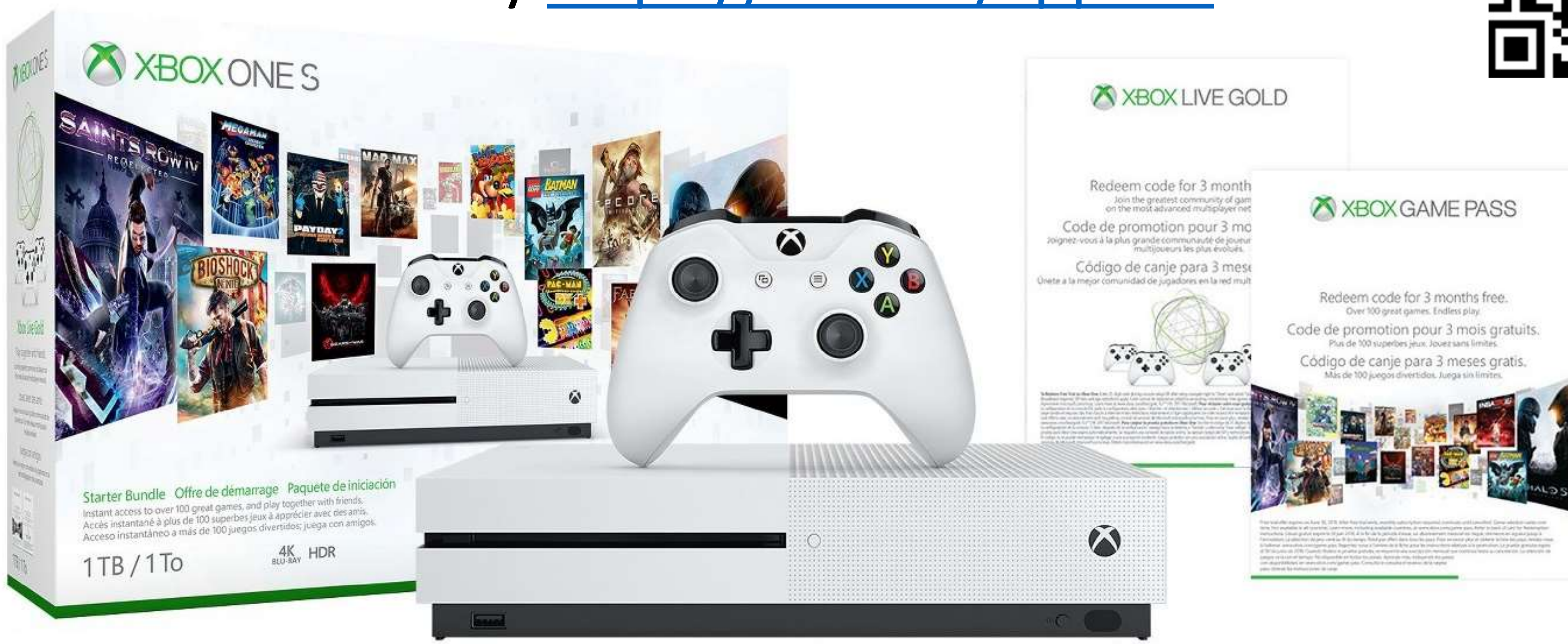
Or: Shift off low order bit that still says left/right; shift in a 1



stable_sort laptop results

- .\stable_sort_release.exe 100000000 - Parallel was 3.40229 times faster
- .\stable_sort_debug.exe 1000000 - Parallel was 3.68103 times faster
- .\stable_sort_release.exe 1000000 - Parallel was 2.96105 times faster
- .\stable_sort_debug.exe 10000 - Parallel was 1.26545 times faster
- .\stable_sort_release.exe 10000 - Parallel was 1.28868 times slower
- .\stable_sort_debug.exe 100 - Parallel was 2.55325 times slower
- .\stable_sort_release.exe 100 - Parallel was 10.2388 times slower

Take our survey https://aka.ms/cppcon



You can win an Xbox One S - Starter Bundle

9/27 10:30 – 12:00 // *Breckenridge Hall*
Thoughts on a More Powerful and Simpler C++ (5 of N), *Herb Sutter*

Other sessions

Monday, September 24th

- 14:00 – 15:00
  How to Write Well-Behaved Value Wrappers
  - by Simon Brand
- 15:15 – 16:15
  How C++ Debuggers Work
  - by Simon Brand

Tuesday, September 25th

- 14:00 – 15:00
  What Could Possibly Go Wrong?: A Tale of Expectations and Exceptions
  - by Simon Brand and Phil Nash
- 15:15 – 15:45
  Overloading: The Bane of All Higher-Order Functions
  - by Simon Brand

Wednesday, September 26th

- 12:30 – 13:30
  C++ Community Building Birds of a Feather
  - with Stephan T. Lavavej and others
- 14:00 – 15:00
  Latest and Greatest in the Visual Studio Family for C++ Developers 2018
  - by Marian Luparu and Steve Carroll
- 15:15 – 15:45
  Don't Package Your Libraries, Write Packagable Libraries!
  - by Robert Schumacher

Wednesday, September 26th

- 15:15 – 15:45
  What's new in Visual Studio Code for C++ Development
  - by Rong Lu
- 15:50 – 16:20
  Value Semantics: Fast, Safe, and Correct by Default
  - by Nicole Mazzuca
- 16:45 – 17:45
  Memory Latency Troubles You? Nano-coroutines to the Rescue! (Using Coroutines TS, of Course)
  - by Gor Nishanov
- 18:45 – 20:00
  Cross-Platform C++ Development is Challenging – let Tools Help!
  - by Marc Goodner and Will Buik

Thursday, September 27th

- 9:00 – 10:00
  Inside Visual C++'s Parallel Algorithms
  - by Billy O'Neal
- 15:15 – 15:45
  ConcurrencyCheck – Static Analyzer for Concurrency Issues in Modern C++
  - by Anna Gringauze
- 16:45 – 17:45
  Class Template Argument Deduction for Everyone
  - by Stephan T. Lavavej

Questions?

References for those looking at doing their own implementations:

A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations
  Peter M. Kogge and Harold S. Stone
Single-pass Parallel Prefix Scan with Decoupled Look-back
  Duane Merrill and Michael Garland
Thread Scheduling for Multiprogrammed Multiprocessors
  Nimar S. Arora, Robert D. Blumofe, and C. Greg Plaxton
Dynamic Circular Work-Stealing Deque
  David Chase and Yossi Lev
CppCon 2015 "Work Stealing" https://youtu.be/iLHNF7SgVN4
  Pablo Halpern
Inside Windows 8: Pedro Teixeira - Thread pools https://channel9.msdn.com/Shows/Going+Deep/Inside-Windows-8-Pedro-Teixeira-Thread-pool
  Pedro Teixeira