

Cache Efficient Parallel Partition Algorithms

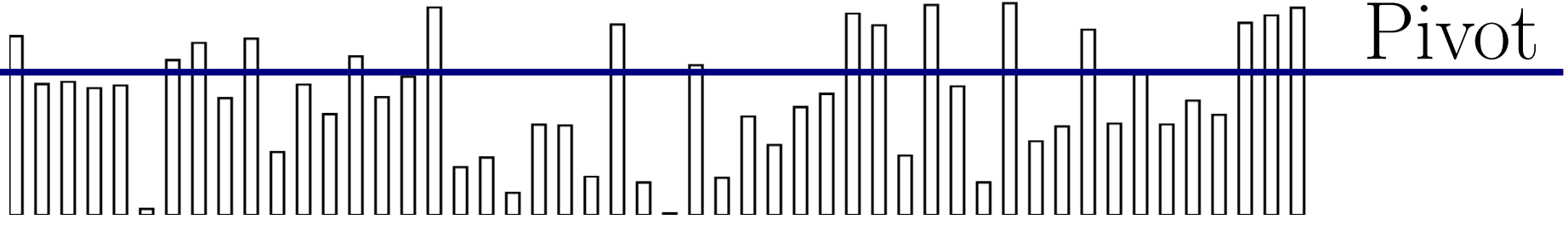
An In-Place Exclusive Read/Write Memory Algorithm

WHAT IS THE PARTITION PROBLEM?

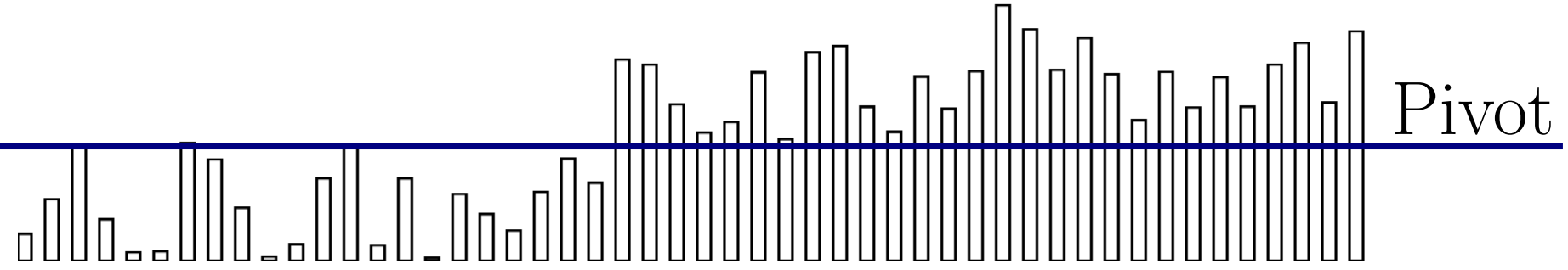
Explanation: The *Partition Problem* is to reorder the elements in a list so that elements in the same group occur in the same part of the list.

Example: A common way of grouping elements is based on whether they exceed or fall short of a certain “pivot” value.

An unpartitioned array:



An array partitioned relative to the pivot value:

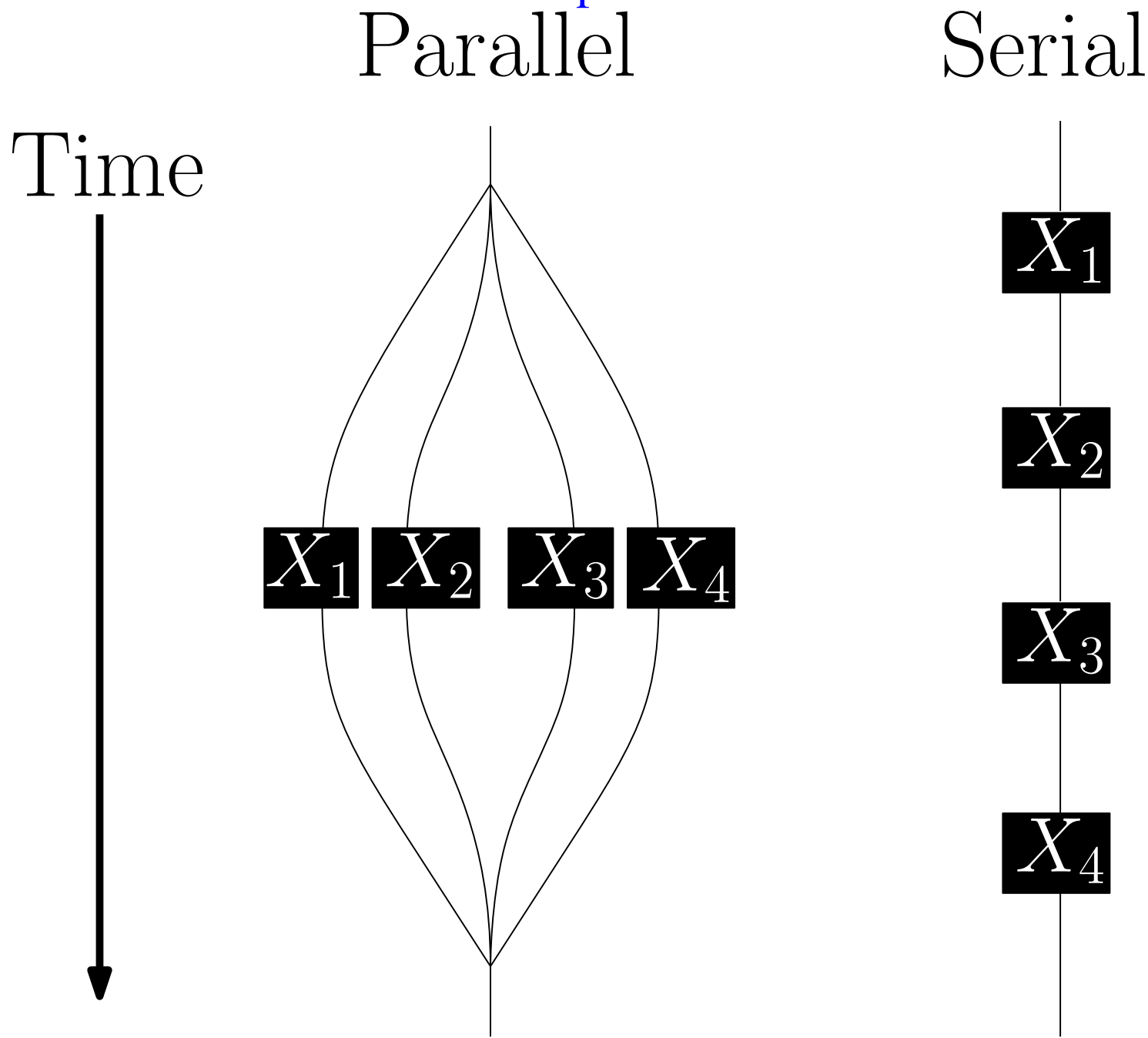


WHAT IS A PARALLEL ALGORITHM?

Explanation: Whereas a typical (i.e. serial) algorithm runs on a single processor, a *parallel algorithm* runs on $p \geq 1$ processors.

Example: Many tasks have parts that can be performed concurrently; such tasks can be performed faster with parallel computing.

Program execution in serial and in parallel



WHAT IS CACHE EFFICIENCY?

Explanation: *Cache* is a small part of memory that can be accessed much faster than ordinary RAM. When data is already loaded into Cache a program can rapidly access it; this is called a *cache hit*. When data needed by a program isn't in cache it must be loaded into cache; this is called a *cache miss*, and takes time.

Remark: An algorithm with very few cache misses is *Cache Efficient*; cache efficiency leads to faster performance in practice.

Factors in Cache-Efficiency:

- ▶ Perform low number of passes over the data
- ▶ Don't use extra memory, i.e. are *In-Place*
- ▶ Deal with elements that are close in memory together

PREVIOUS WORK ON THE PARTITION PROBLEM

The “Standard Algorithm” is **theoretically optimal with span $O(\log n)$** , but **slow in practice due to poor cache behavior**.

The **fastest algorithms in practice lack theoretical guarantees**

- ▶ Lock-based and atomic-variable based algorithms

[Michael Axtmann, Sascha Witt, Daniel Ferizovic, and Peter Sanders, 2017;

Philip Heidelberger, Alan Norton, and John T. Robinson, 1990; Philippas Tsigas and Yi Zhang, 2003]

Not Exclusive Read/Write Memory

- ▶ The Strided Algorithm

[Francis and Pannan, 92; Frias and Petit, 08]

No locks or atomic-variables, but no bound on span

OUR RESEARCH QUESTION

Can we create an algorithm with *theoretical guarantees* that is *fast in practice*?

RESULT

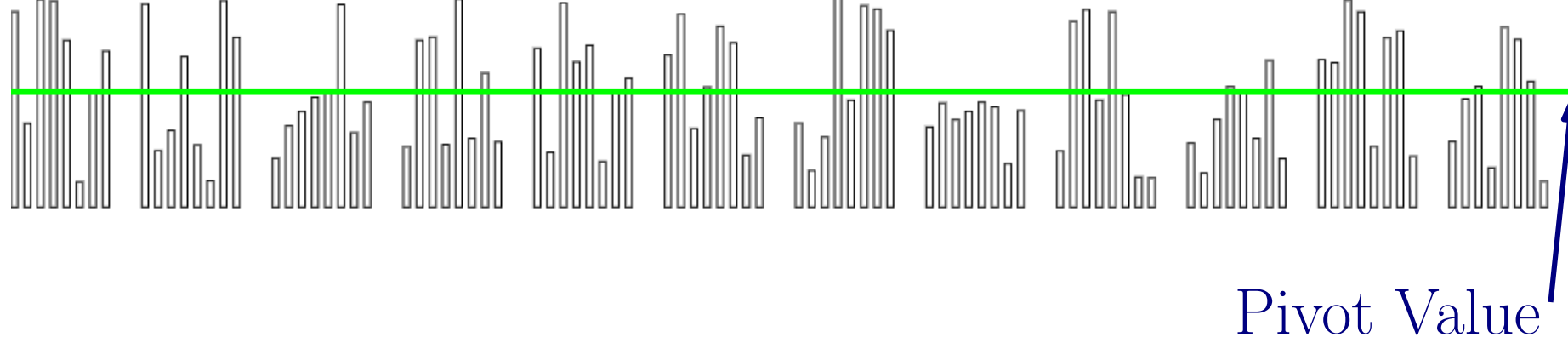
We created the *Smoothed Striding Algorithm*.

Key Features:

- ▶ linear work and polylogarithmic span
(like the Standard Algorithm)
- ▶ fast in practice
(like the Strided Algorithm)
- ▶ theoretically optimal cache behavior
(unlike any past algorithm)

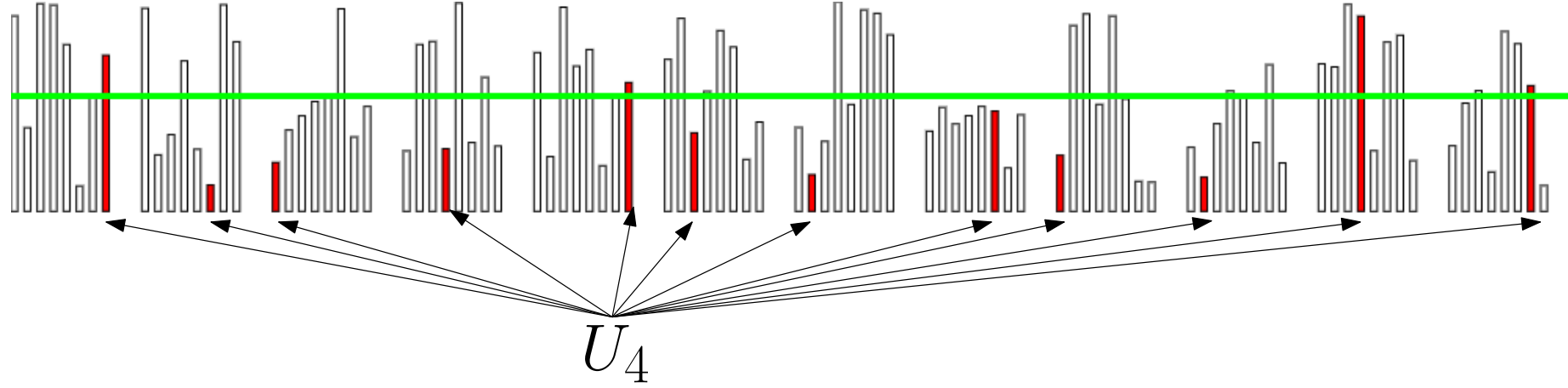
SMOOTHED STRIDING ALGORITHM

Logically partition the array into chunks of adjacent elements.



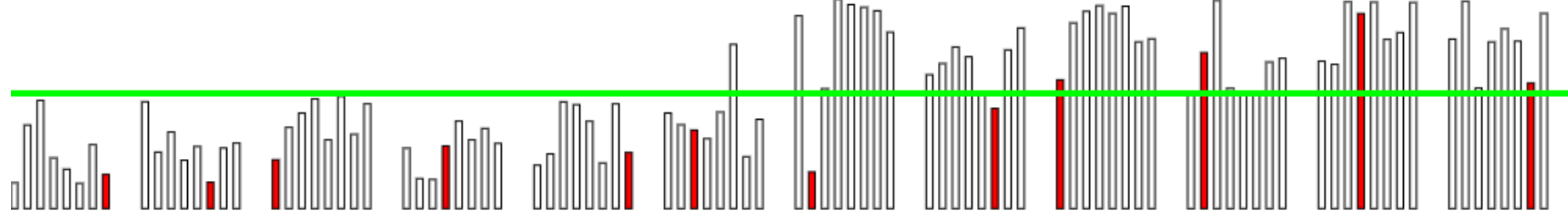
Form groups U_i that contain a random element from each chunk.

This randomization step was one of our key insights; it guarantees that the U_i 's have similar compositions regardless of the input.

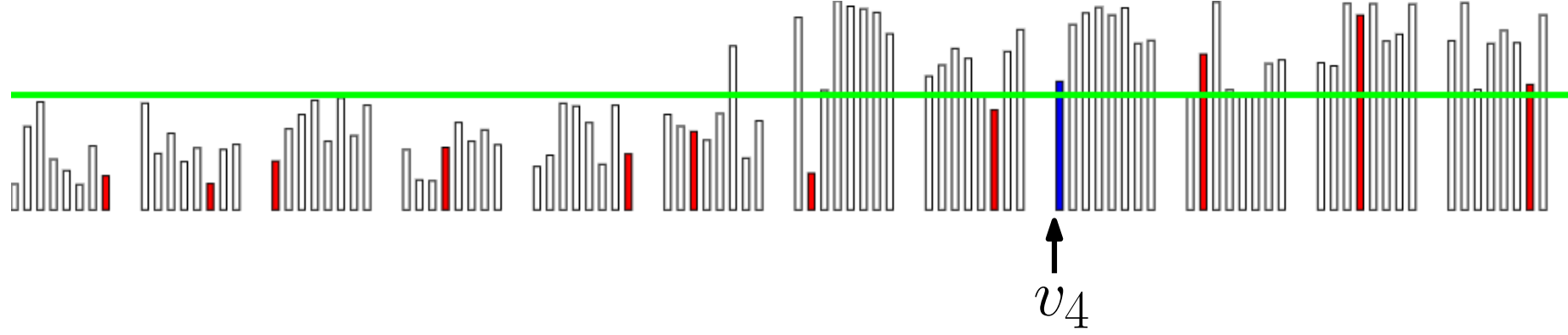


Perform serial partitions on each U_i in parallel over the U_i 's.

This step is highly parallel.

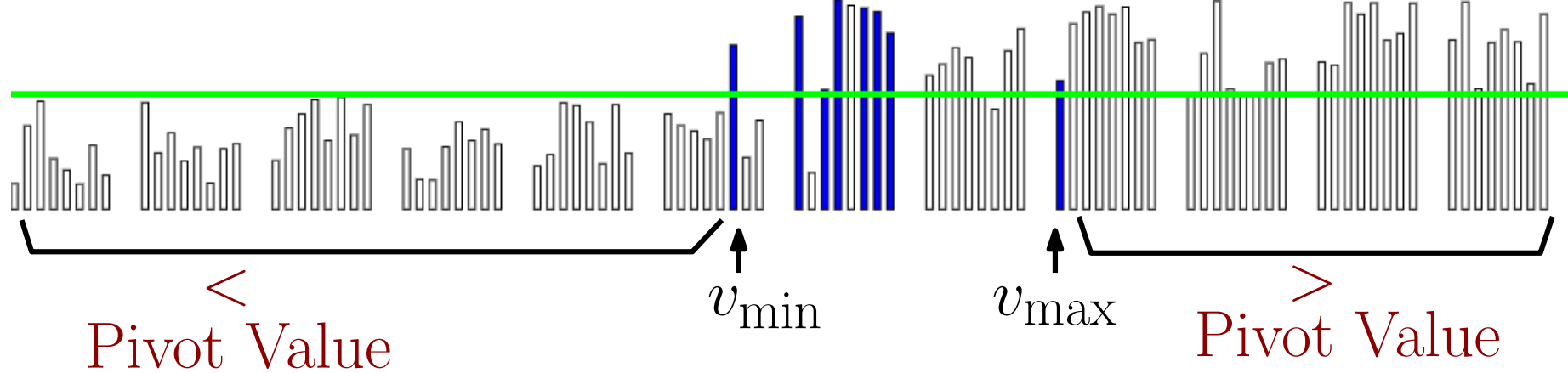


Define v_i = index of first element greater than the pivot in U_i .



Identify leftmost and rightmost v_i .

Note that $A[k] \leq \text{pivot}$ for all $k < v_{\min}$, and $A[k] > \text{pivot}$ for all $k \geq v_{\max}$.



Recursively partition the subarray.

This step was previously impossible; adding randomization enables this step, which enables our algorithm's low span.

