Parallel Computing

Announcements

- Midterm has been graded; will be distributed after class along with solutions.
- SCPD students: Midterms have been sent to the SCPD office and should be sent back to you soon.

Announcements

- Assignment 6 due right now.
- Assignment 7 (**Pathfinder**) out, due next Tuesday at 11:30AM.
  - Play around with graphs and graph algorithms!
  - Learn how to interface with library code.
- **No late submissions will be considered**. This is as late as we're allowed to have the assignment due.

Why Algorithms and Data Structures Matter

Making Things Faster

- **Choose better algorithms and data structures**.
  - Dropping from O(n^2) to O(n log n) for large data sets will make your programs faster.
- **Optimize your code**.
  - Try to reduce the constant factor in the big-O notation.
  - Not recommended unless all else fails.
- **Get a better computer**.
  - Having more memory and processing power can improve performance.
- **New option: Use parallelism.**

How Your Programs Run

Threads of Execution

- When running a program, that program gets a **thread of execution** (or **thread**).
- Each thread runs through code as normal.
- A program can have multiple threads running at the same time, each of which performs different tasks.
- A program that uses multiple threads is called **multithreaded**; writing a multithreaded program or algorithm is called **multithreading**.

Threads in C++

- The newest version of C++ (**C++11**) has libraries that support threading.
- To create a thread:
  - Write the function that you want to execute.
  - Construct an object of type **`thread`** to run that function.
    - Need header **`<thread>`** for this.
  - That function will run in parallel alongside the original program.

How Do Threads Work?

- **Preemption**: The computer runs one thread for a short time, then runs the next thread for a short time, etc.
  - Gives the illusion of everything running concurrently, though only one thread runs at a time.
  - Changing which thread runs is called a **context switch**.
- **Parallelism**: The computer has hardware that lets it run multiple threads at the same time.
  - Multiple different tasks really are running at the same time.
- Not mutually exclusive; can do both (most computers do).
- There are other options, but we'll focus on these today.

Indeterminacy

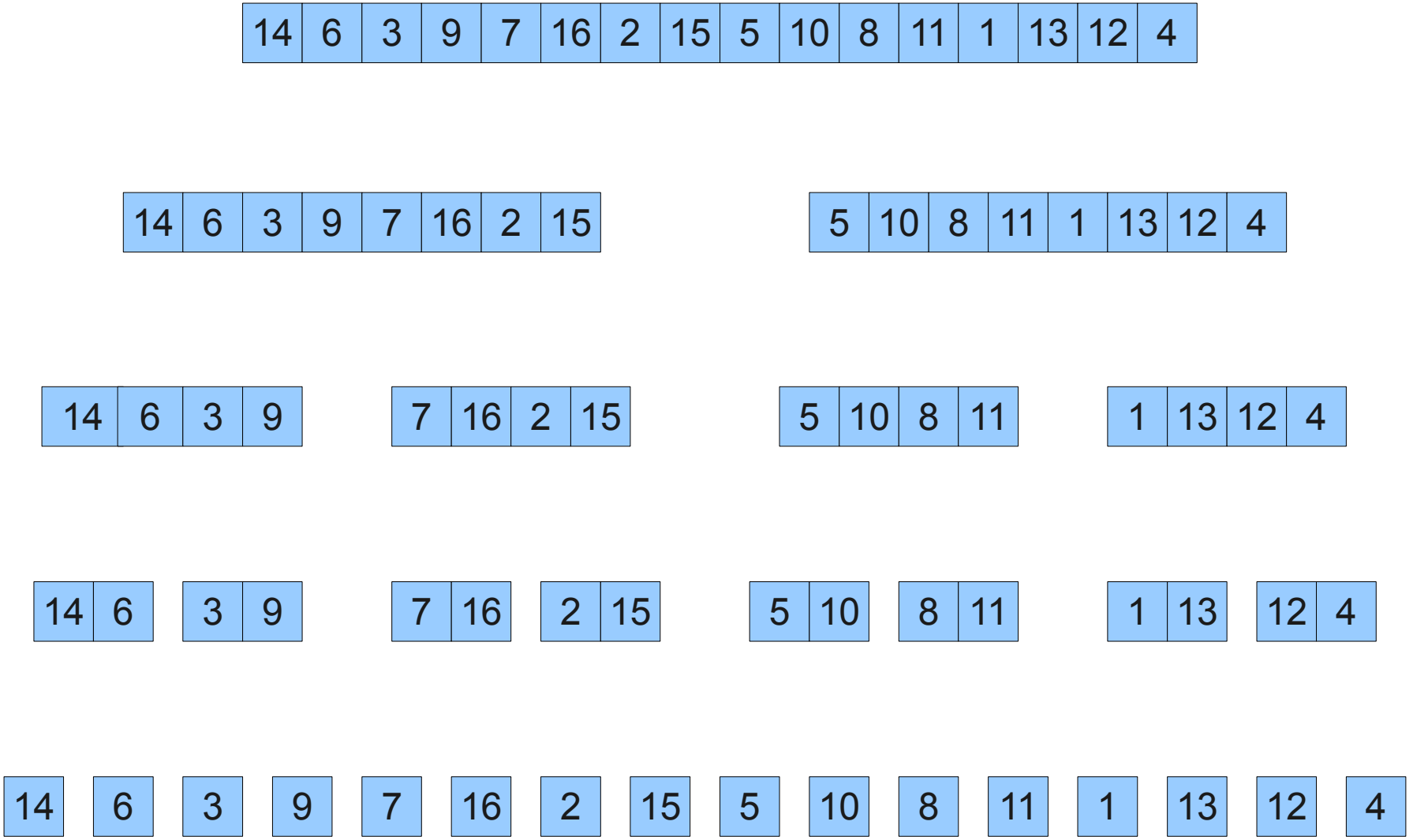
- The order in which different threads execute cannot be controlled.
- Execution order is **indeterminate**.
- Running the same multithreaded program many times can result in different outcomes each time.

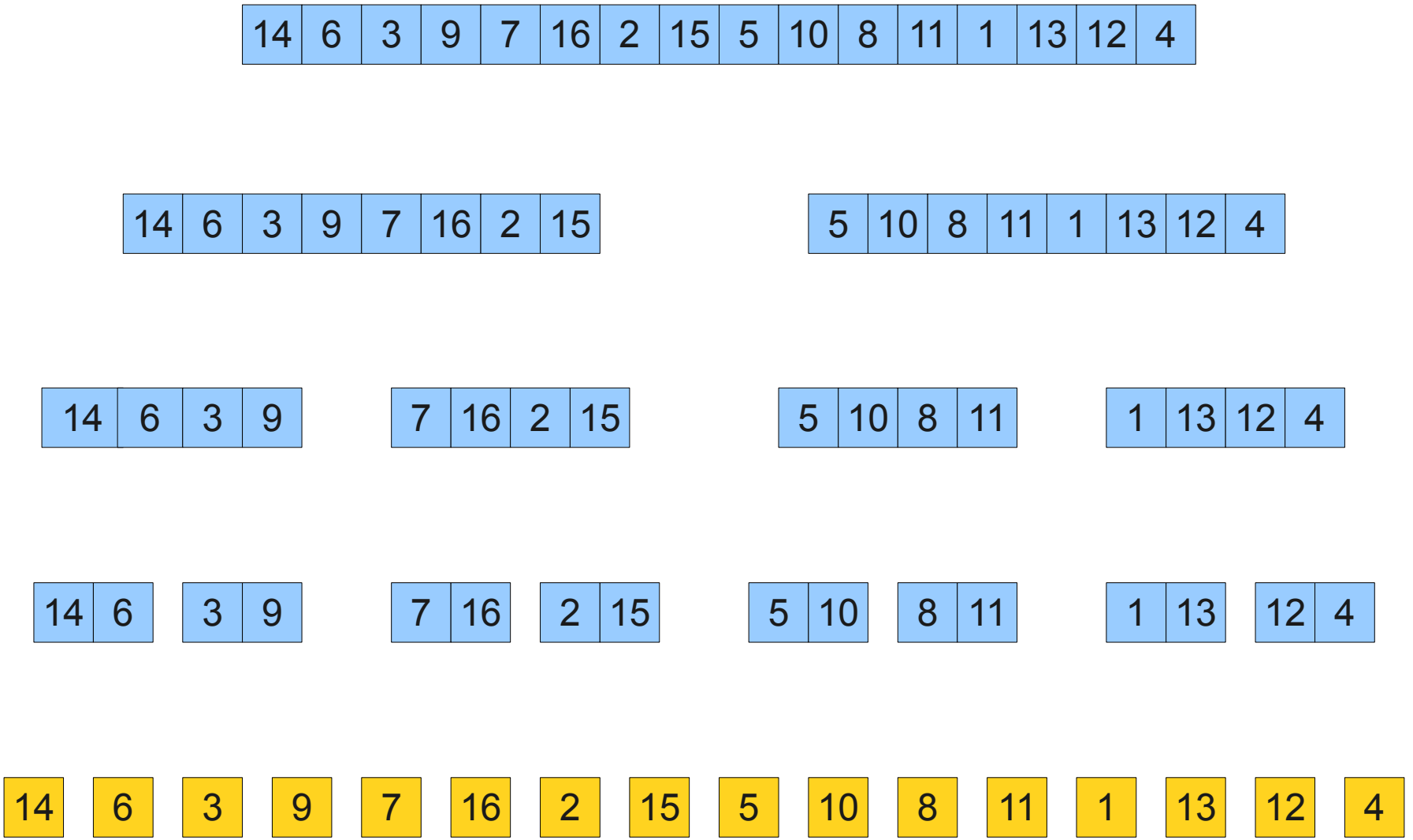
Joining a Thread

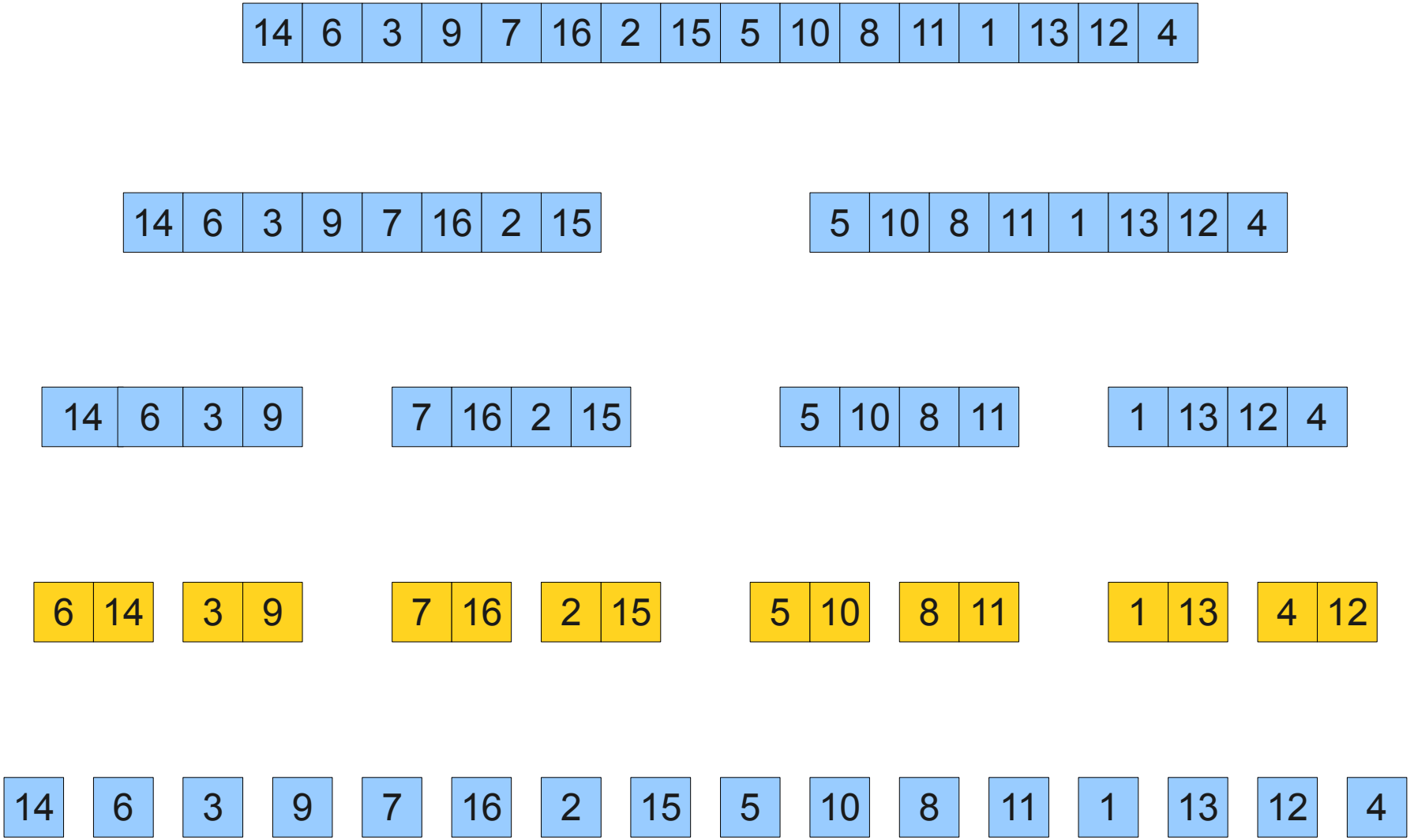
- When writing a program with multiple threads, we sometimes have to wait for a thread to finish.
- One thread **joins** a second thread if it waits for the second thread to terminate before continuing.
- In C++, you can join a thread with the `.join()` member function:

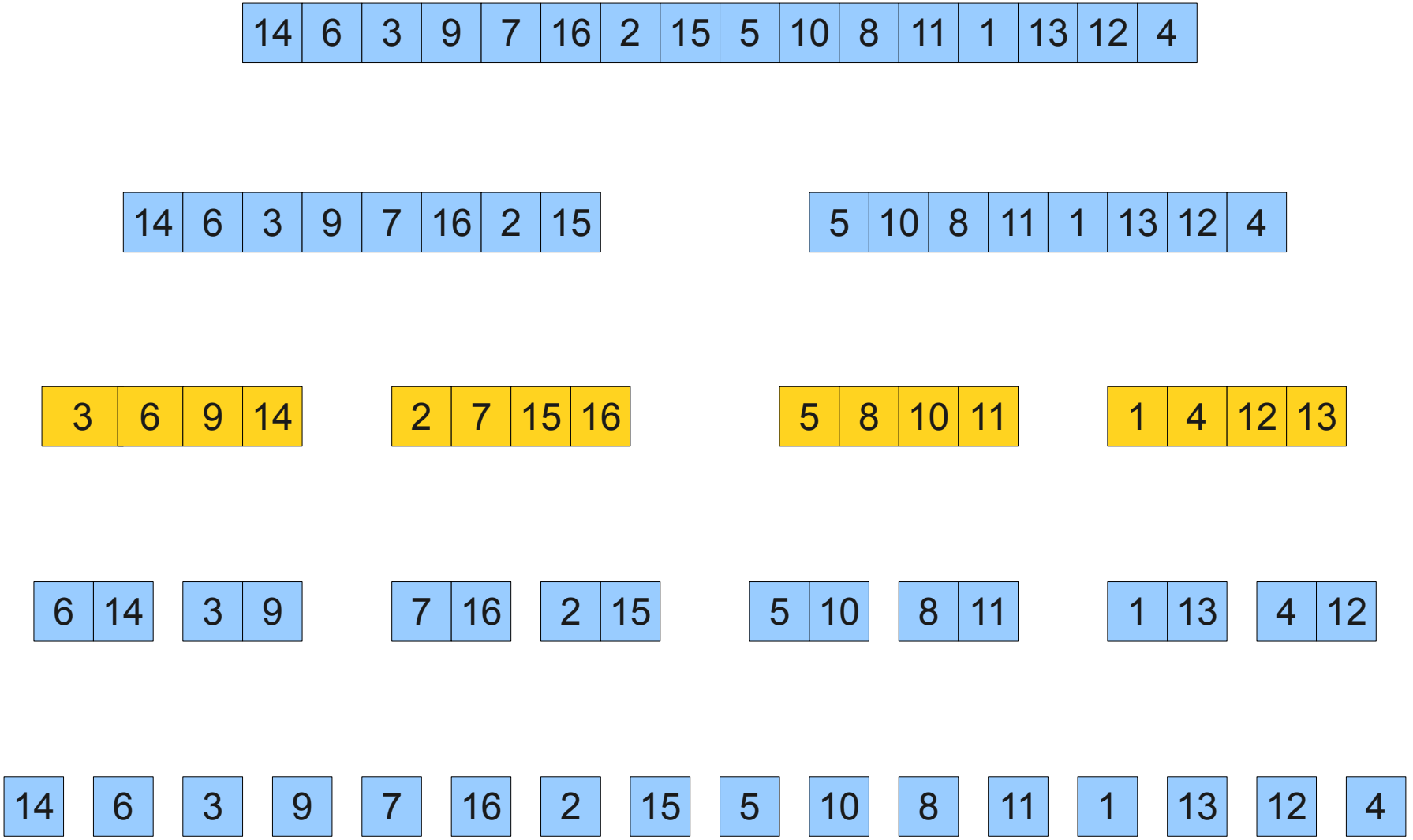
***thread*`.join();`**

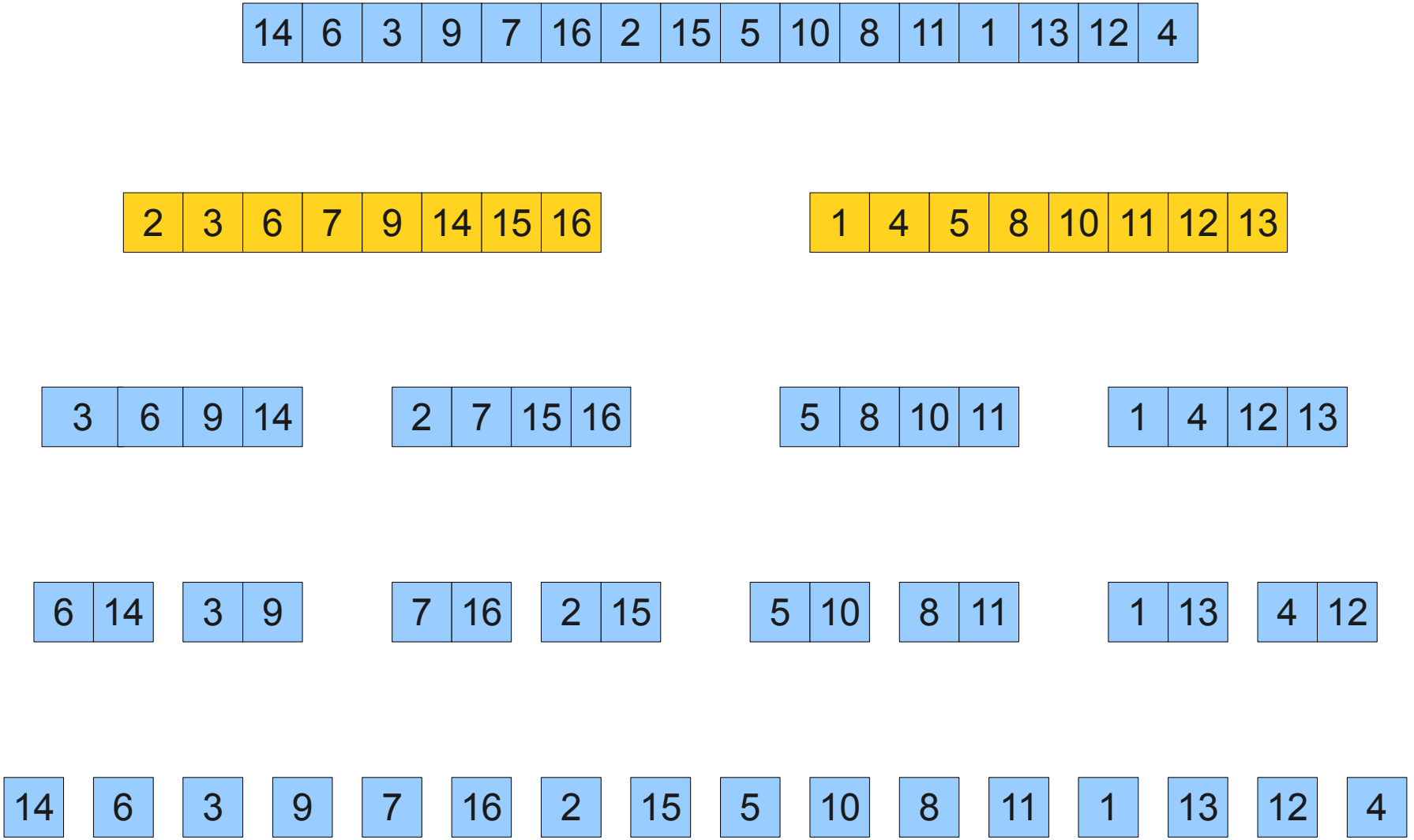
Getting Faster with Threads

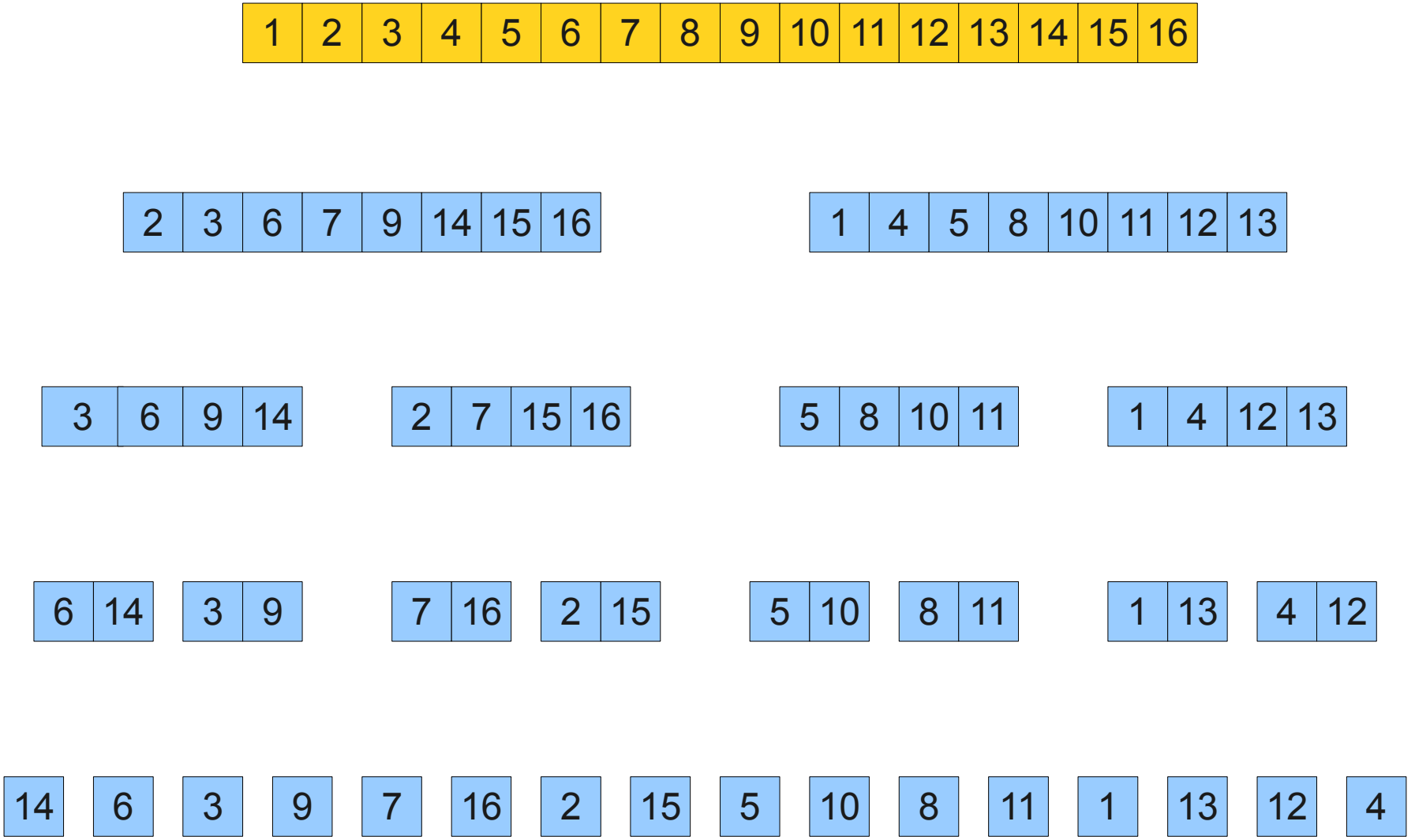

14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4
14 6 3 9 7 16 2 15
5 10 8 11 1 13 12 4
14 6 3 9
7 16 2 15
5 10 8 11
1 13 12 4
14 6
3 9
7 16
2 15
5 10
8 11
1 13
12 4
14
6
3
9
7
16
2
15
5
10
8
11
1
13
12
4


14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4
14 6 3 9 7 16 2 15
5 10 8 11 1 13 12 4
14 6 3 9
7 16 2 15
5 10 8 11
1 13 12 4
14 6
3 9
7 16
2 15
5 10
8 11
1 13
12 4
14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4


14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4
14 6 3 9 7 16 2 15
5 10 8 11 1 13 12 4
14 6 3 9
7 16 2 15
5 10 8 11
1 13 12 4
6 14
3 9
7 16
2 15
5 10
8 11
1 13
4 12
14
6
3
9
7
16
2
15
5
10
8
11
1
13
12
4


14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4
14 6 3 9 7 16 2 15
5 10 8 11 1 13 12 4
3 6 9 14
2 7 15 16
5 8 10 11
1 4 12 13
6 14 3 9
7 16 2 15
5 10 8 11
1 13 4 12
14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4


14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4
2 3 6 7 9 14 15 16
1 4 5 8 10 11 12 13
3 6 9 14
2 7 15 16
5 8 10 11
1 4 12 13
6 14
3 9
7 16
2 15
5 10
8 11
1 13
4 12
14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2 3 6 7 9 14 15 16
1 4 5 8 10 11 12 13
3 6 9 14
2 7 15 16
5 8 10 11
1 4 12 13
6 14
3 9
7 16
2 15
5 10
8 11
1 13
4 12
14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4

Parallel Merge Sort

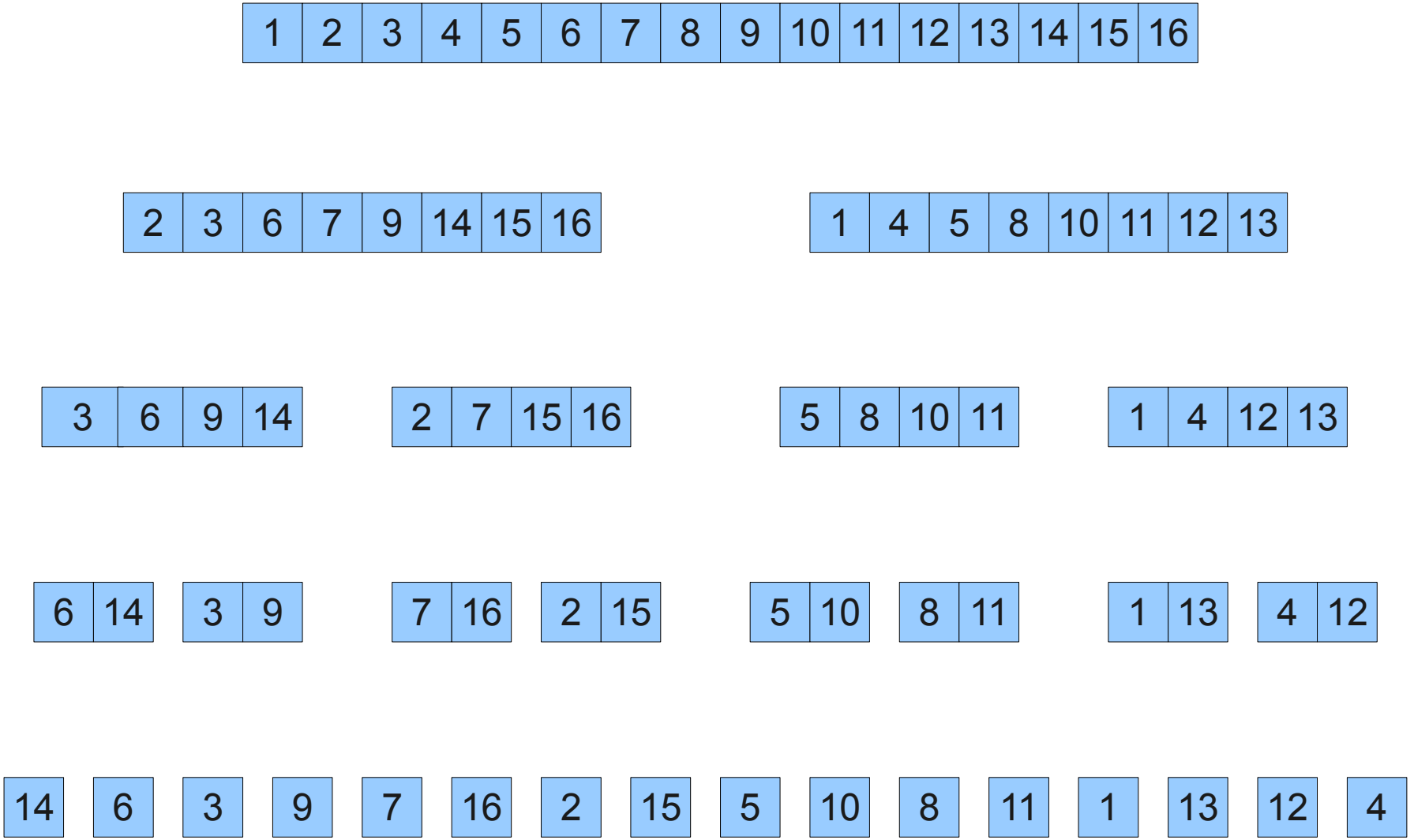
- **Idea**: Parallelize merge sort to sort numbers even faster than before!
- Same algorithm as before, but with multiple threads:
  - Create one thread for each half of the array.
  - Have each thread sort its half independently.
  - Wait for those threads to finish.
  - Merge the subarrays back together.
- Assuming we have multiple cores on the machine, this ends up being much faster.

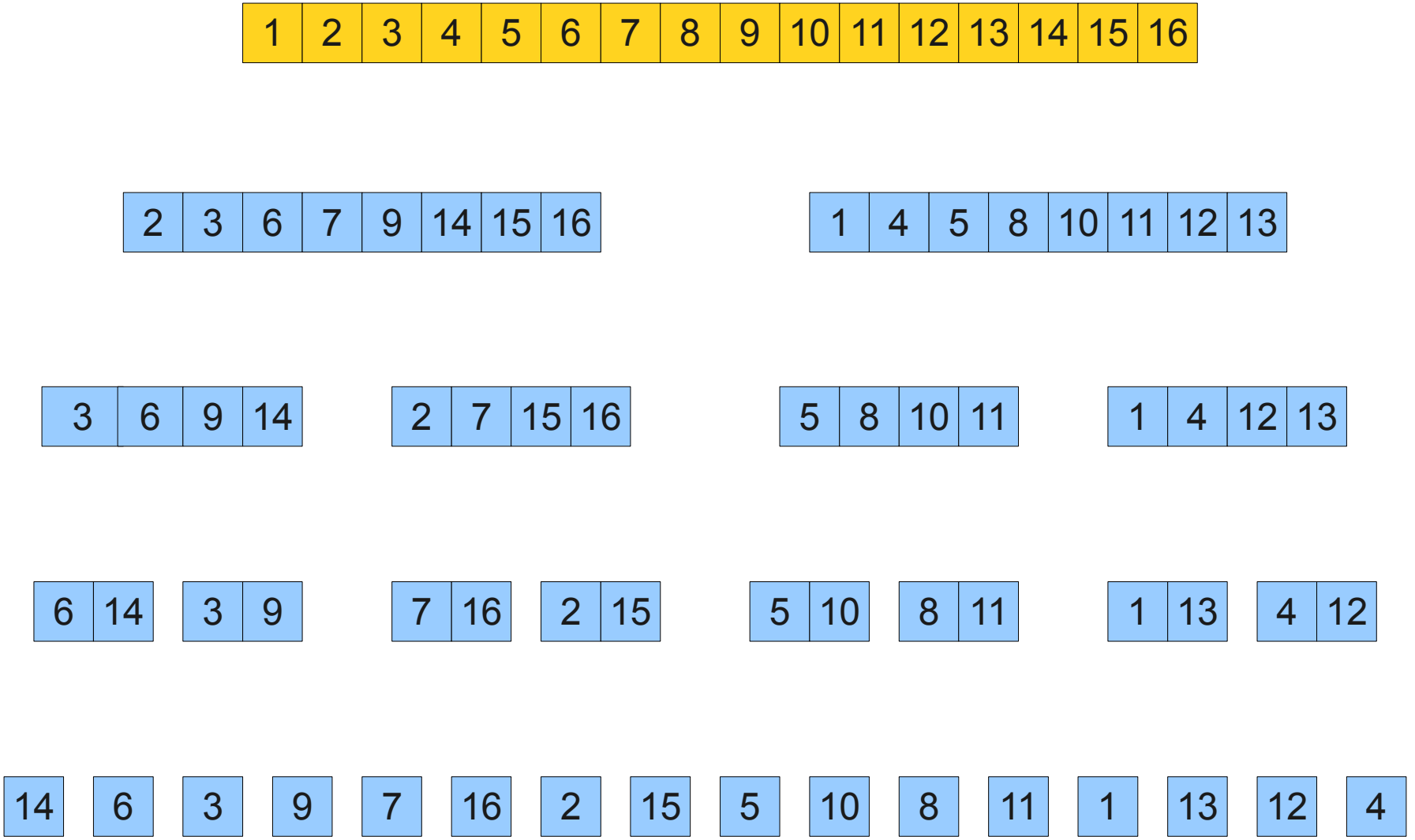
Let's Code it Up!

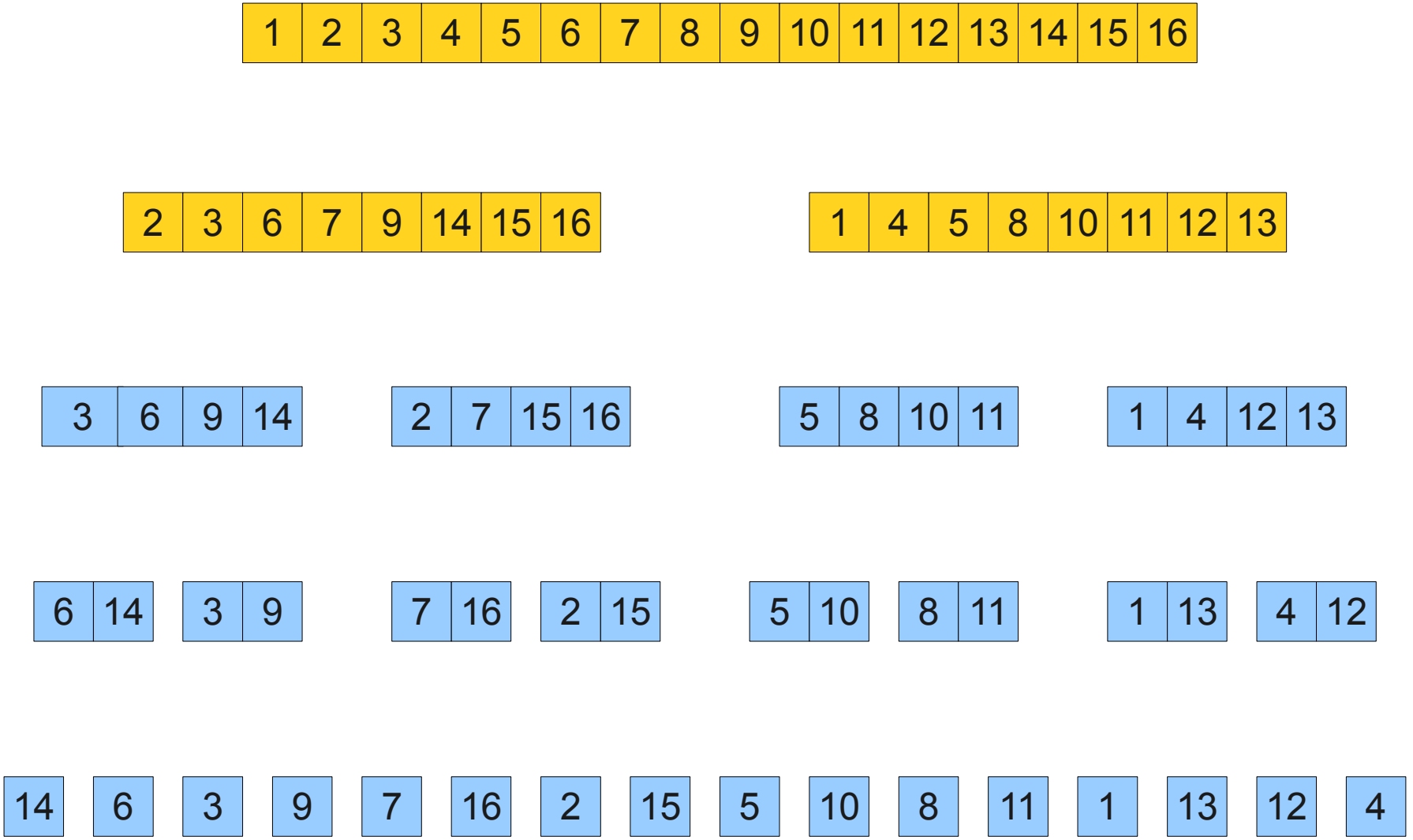
What Went Wrong?

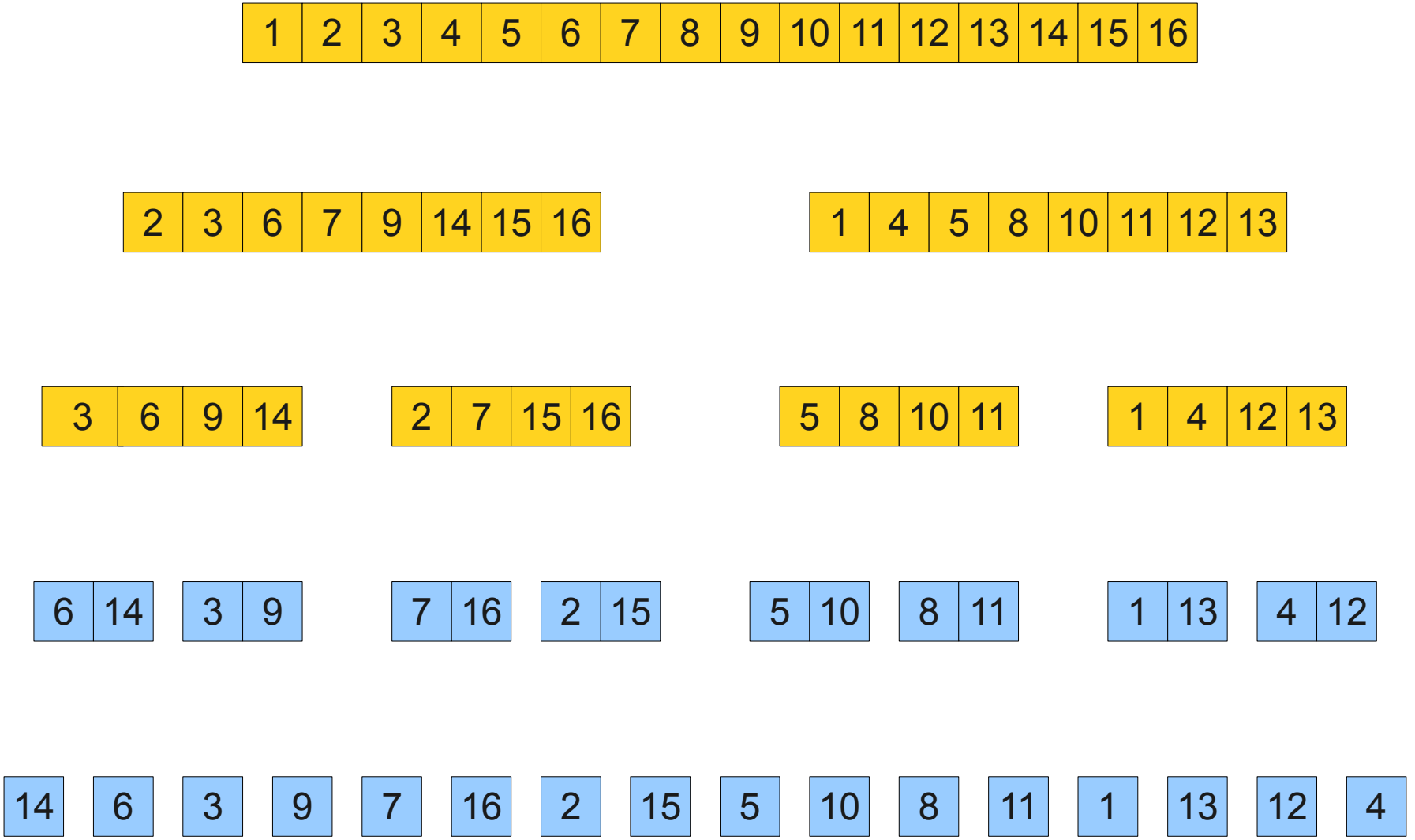
Resource Limitations

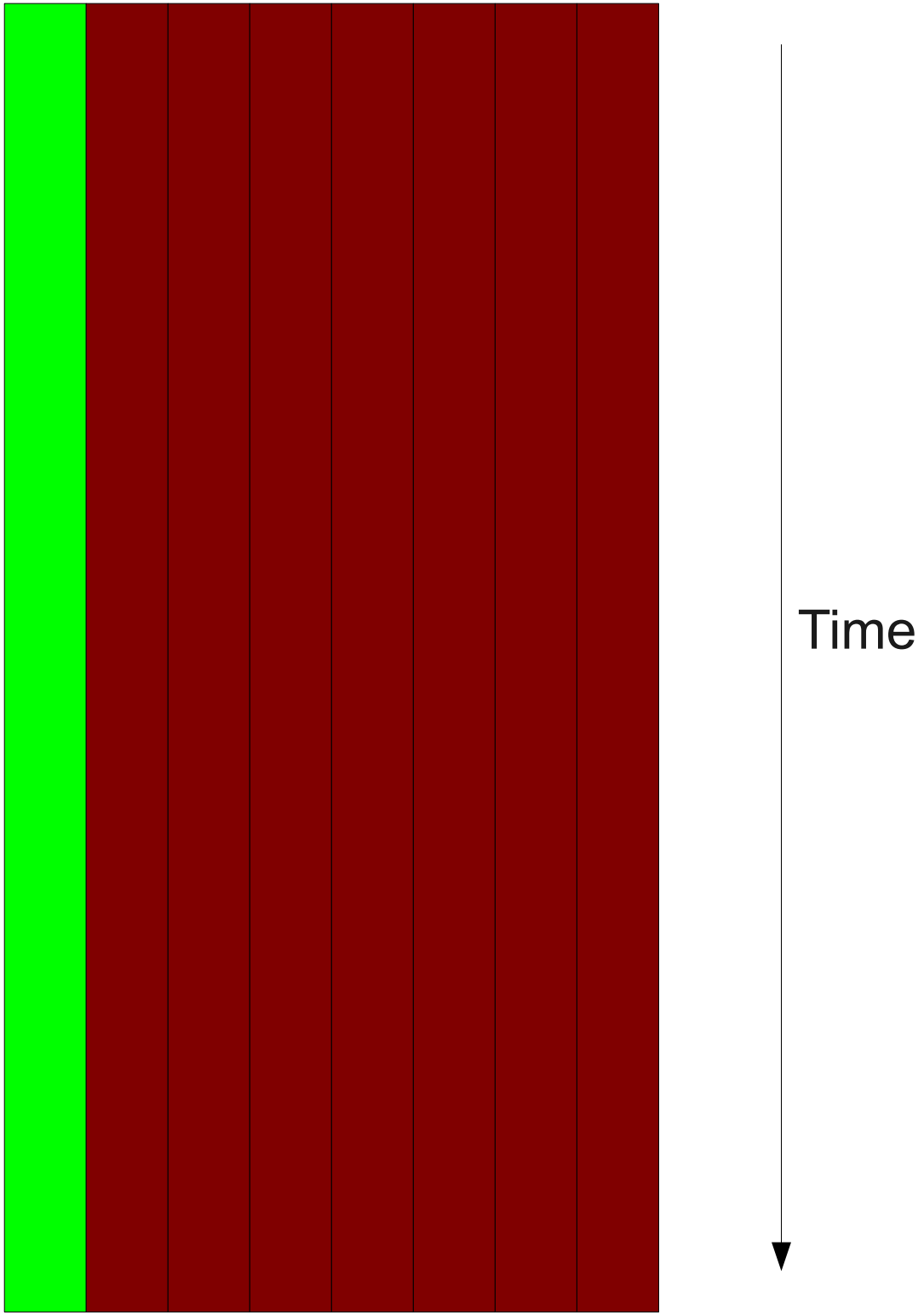
- Like all resources on the computer (memory, disk space, power, etc.), the number of threads is limited.
- We must limit how many threads we use.

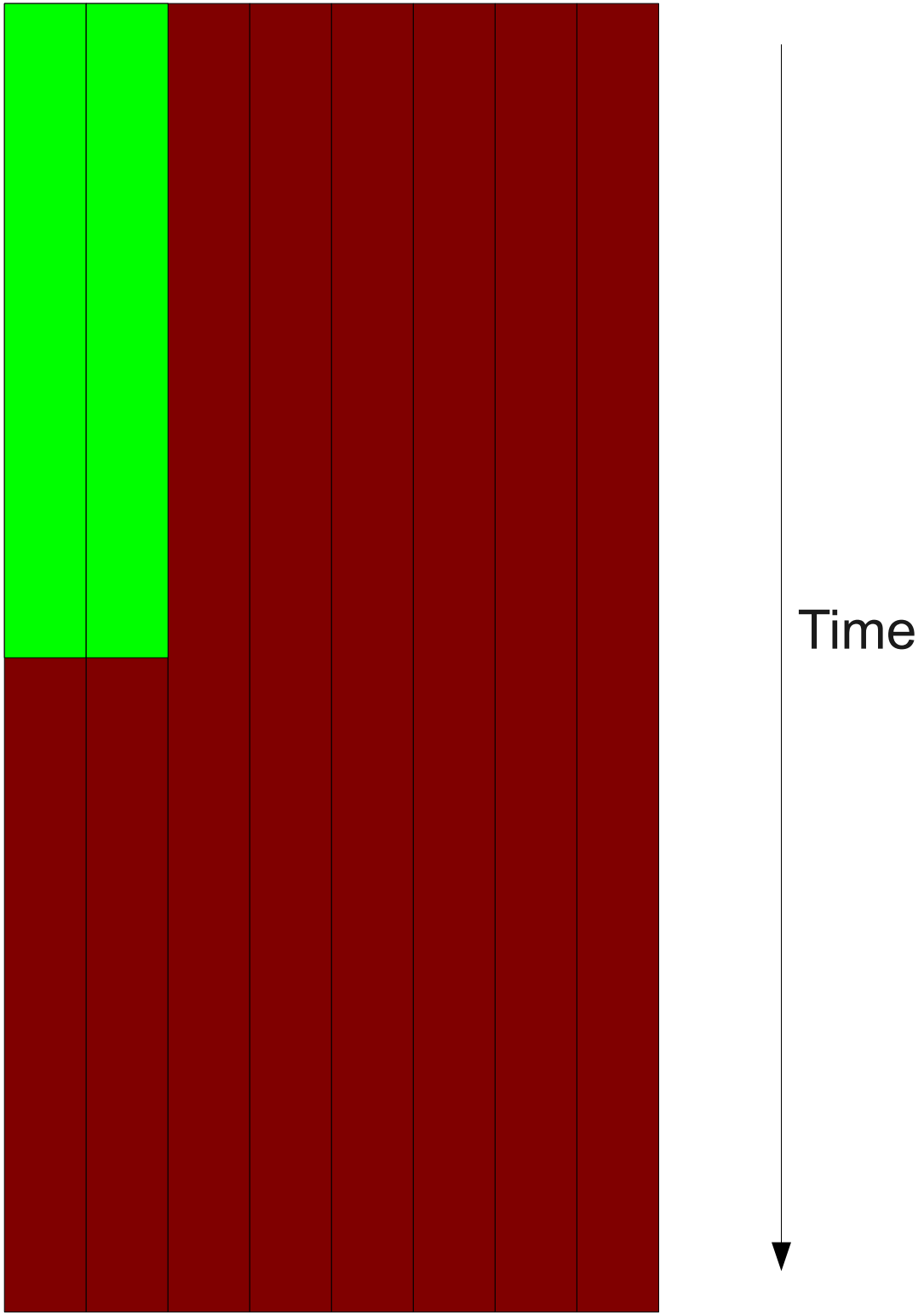

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2 3 6 7 9 14 15 16
1 4 5 8 10 11 12 13
3 6 9 14
2 7 15 16
5 8 10 11
1 4 12 13
6 14
3 9
7 16
2 15
5 10
8 11
1 13
4 12
14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4

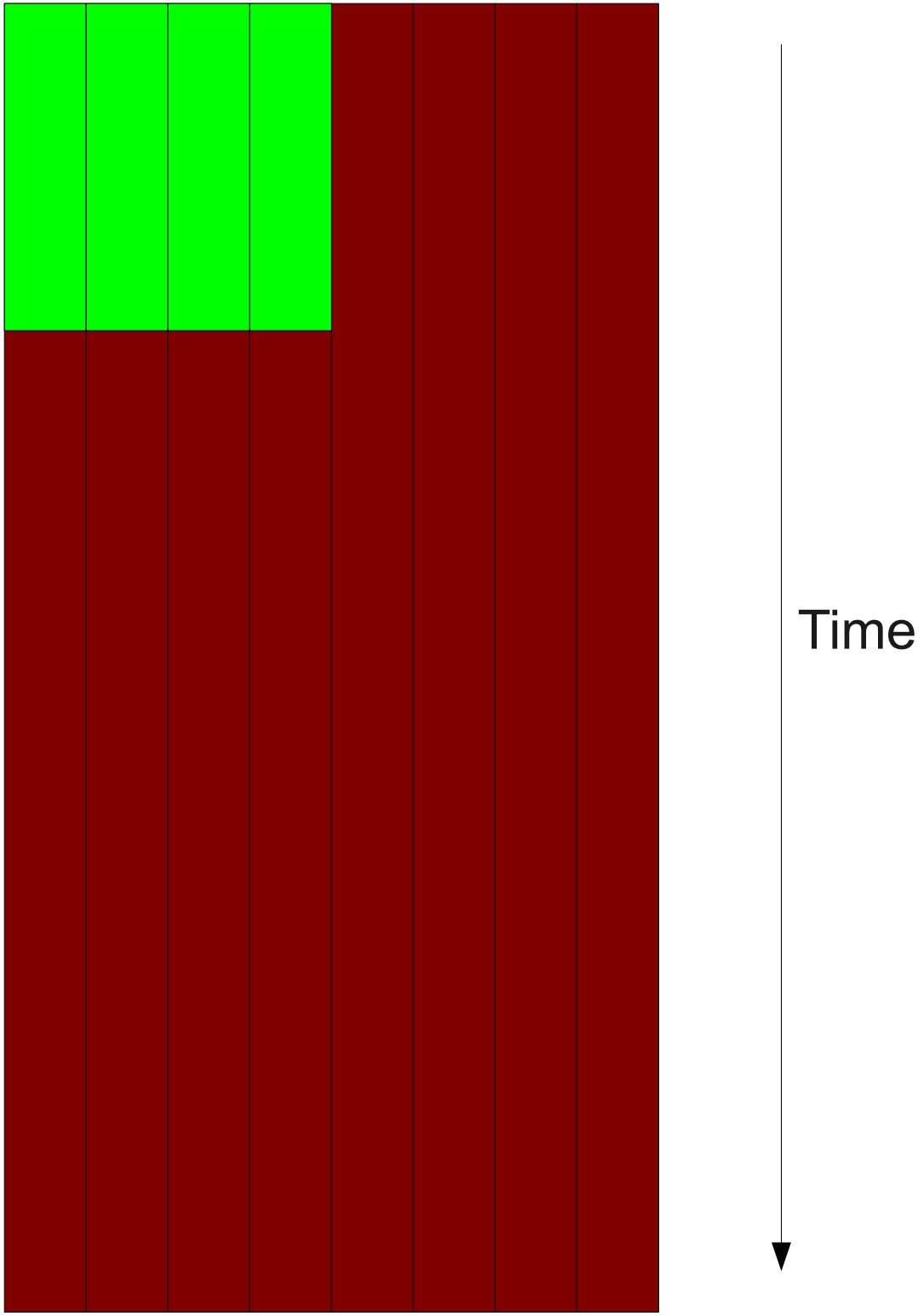

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2 3 6 7 9 14 15 16
1 4 5 8 10 11 12 13
3 6 9 14
2 7 15 16
5 8 10 11
1 4 12 13
6 14
3 9
7 16
2 15
5 10
8 11
1 13
4 12
14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4

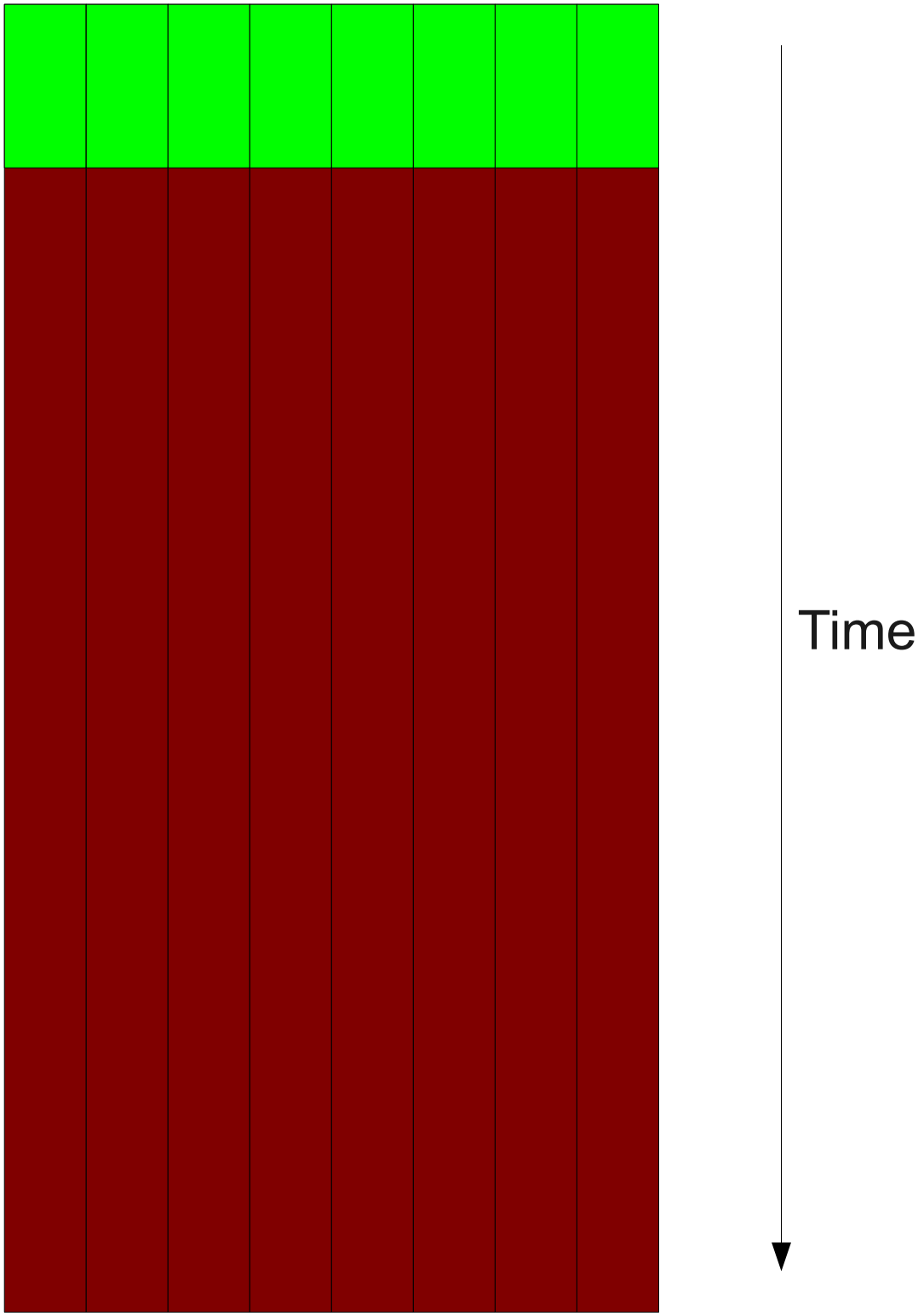

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2 3 6 7 9 14 15 16
1 4 5 8 10 11 12 13
3 6 9 14
2 7 15 16
5 8 10 11
1 4 12 13
6 14
3 9
7 16
2 15
5 10
8 11
1 13
4 12
14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4

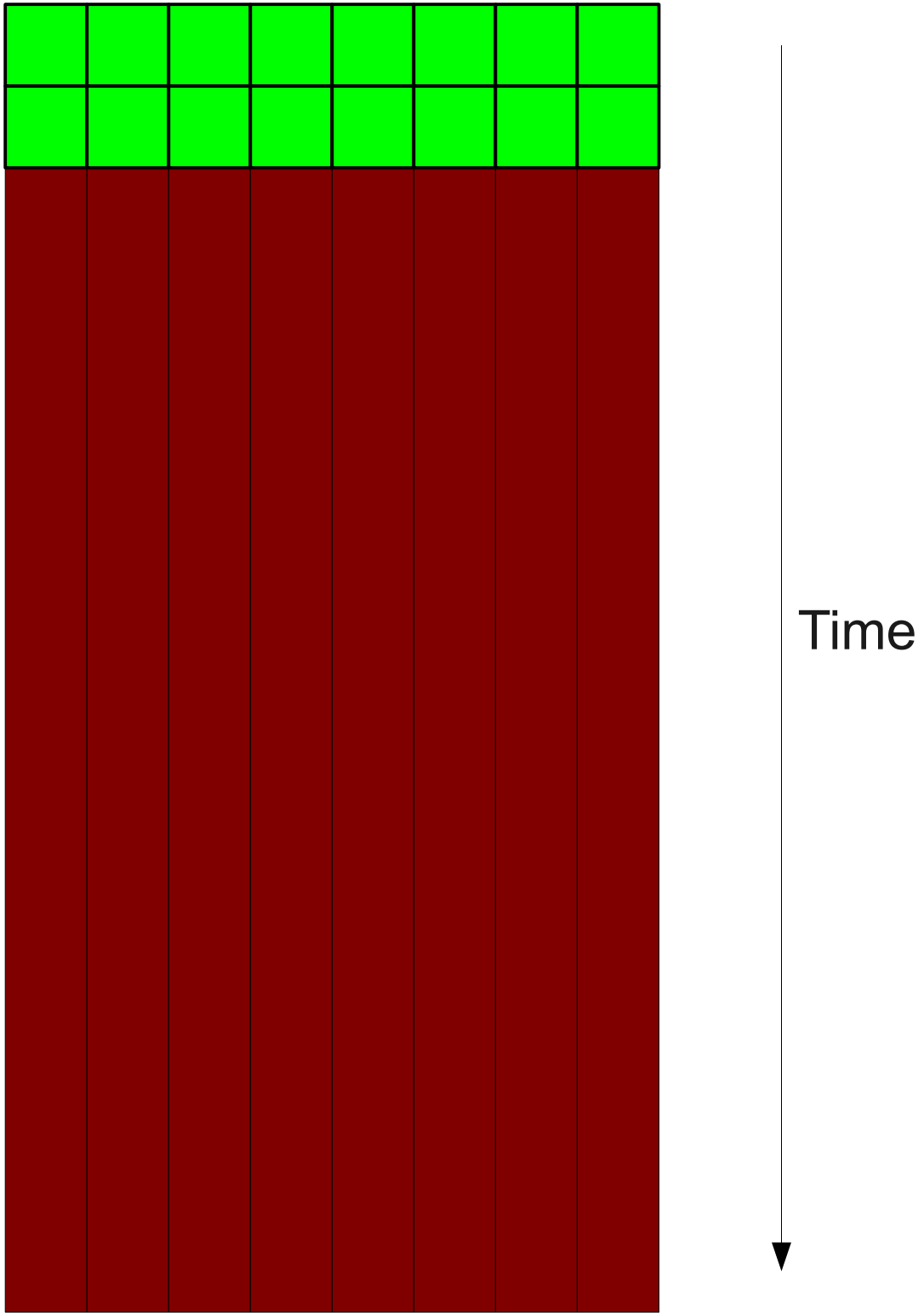

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2 3 6 7 9 14 15 16
1 4 5 8 10 11 12 13
3 6 9 14
2 7 15 16
5 8 10 11
1 4 12 13
6 14
3 9
7 16
2 15
5 10
8 11
1 13
4 12
14 6 3 9 7 16 2 15 5 10 8 11 1 13 12 4

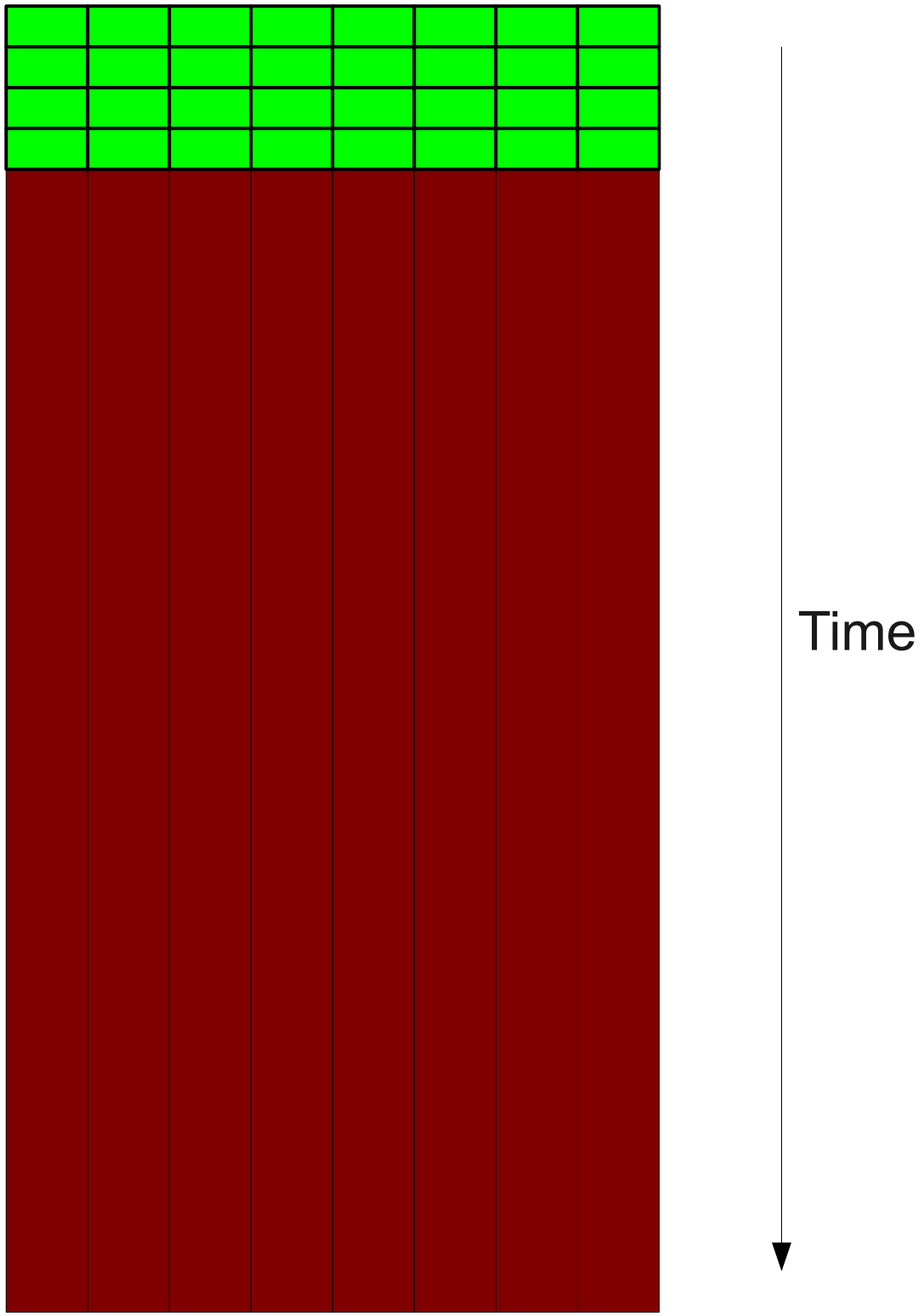

Time


Time


Time


Time


Time


Time

The Hardware Limit

- Multithreading decreases the time required to finish sorting, but doesn't decrease the total work required.
- Once all cores are working, further subdividing the task does not give any performance increase.
- Launching and joining threads adds extra work; too much subdivision can cause a *decrease* in performance.

Race Conditions

Race Condition

- A **race condition** (or **data race**) is an error in a program where the outcome of the program differs based on the timing of the threads.
- In our code, if multiple threads all read **`isSpartacus`** as **`true`** before the first thread to read it sets it to **`false`**, every thread thinks it's Spartacus.

Having Threads Coordinate

Mutexes

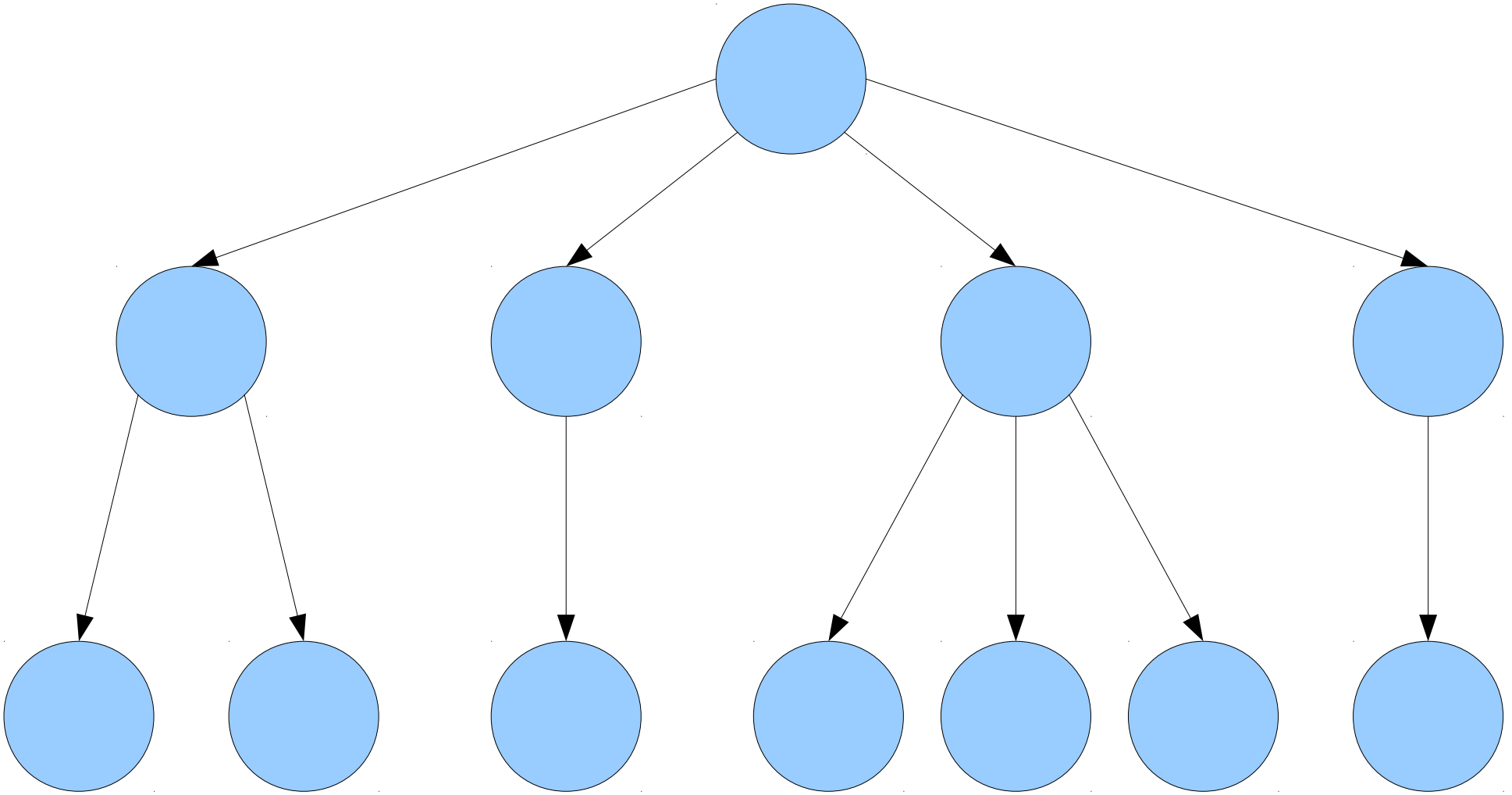
- A **mutex** (from **mut**ual **ex**clusion) is a variable designed to let threads coordinate with one another.
- A thread can try to lock the mutex.
  - If the mutex is unlocked, then the thread locks the mutex and continues as usual until it unlocks it.
  - If the mutex is already locked, the thread pauses until the mutex is unlocked, then locks the mutex.

Threads are Hard

- Coordinating multiple threads can be extremely difficult.
- Take CS110, CS140, or CS149 on details on how to do this.

Getting Faster with Threads

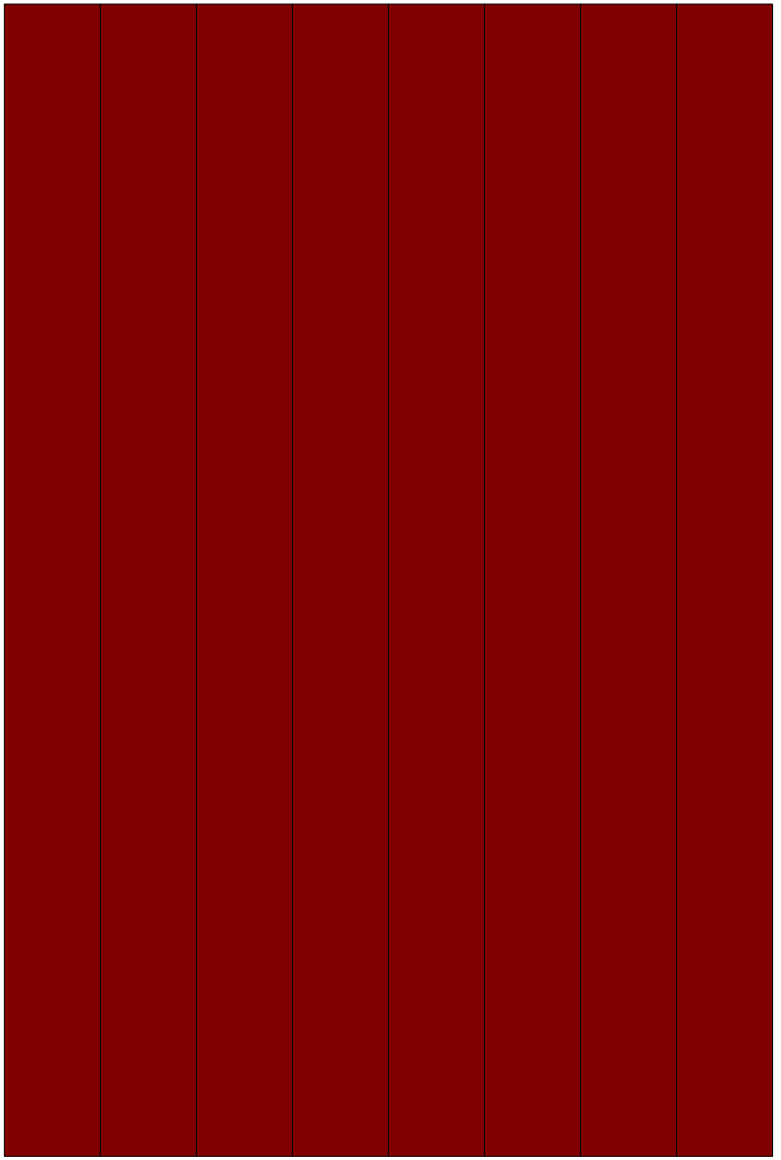
Downloading Wikipedia



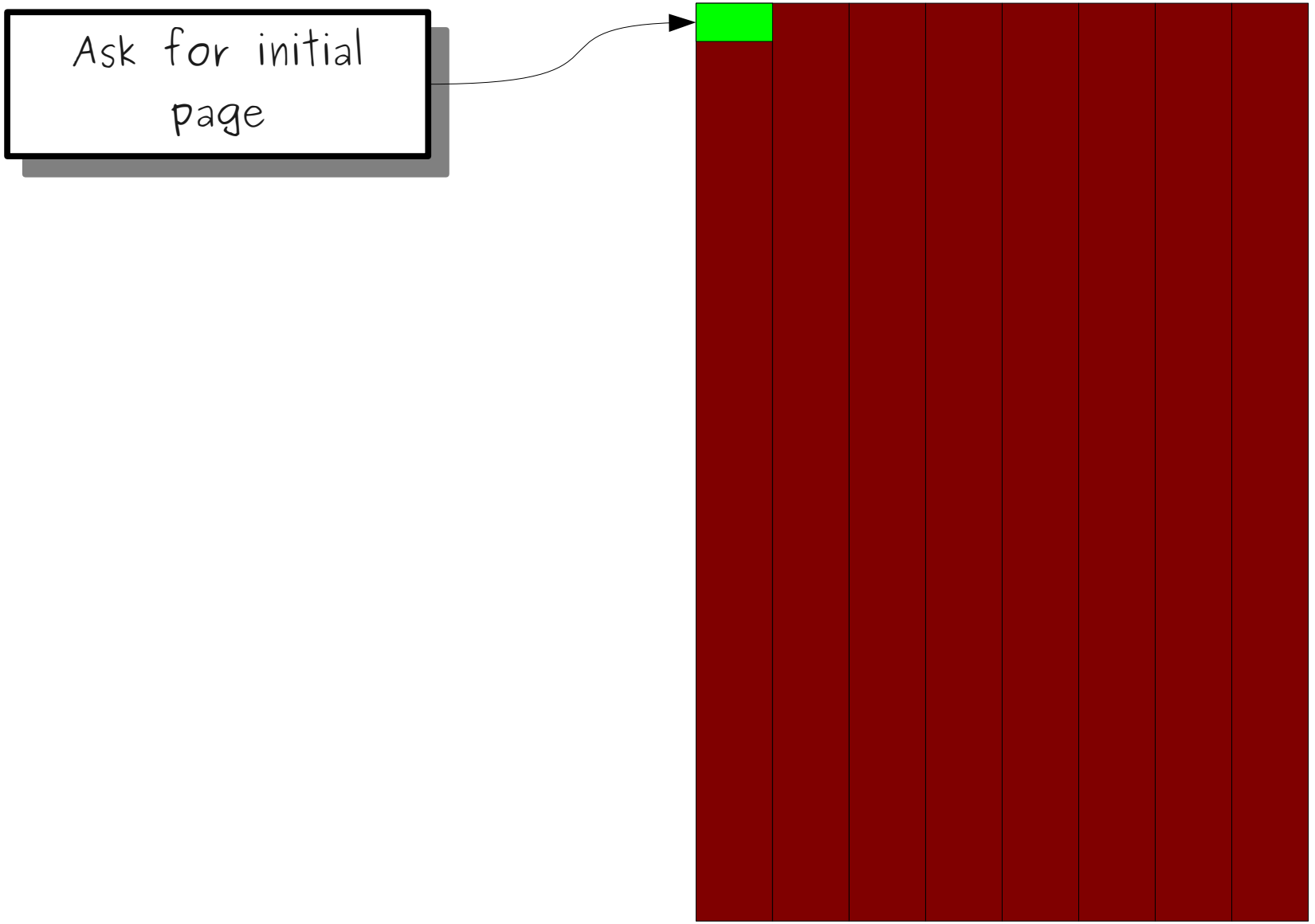
Downloading Wikipedia

- You can think of downloading all of Wikipedia as a graph search problem.
- Given some set of starting articles, do a DFS or BFS of the Wikipedia graph one page at a time.
- How fast is this?

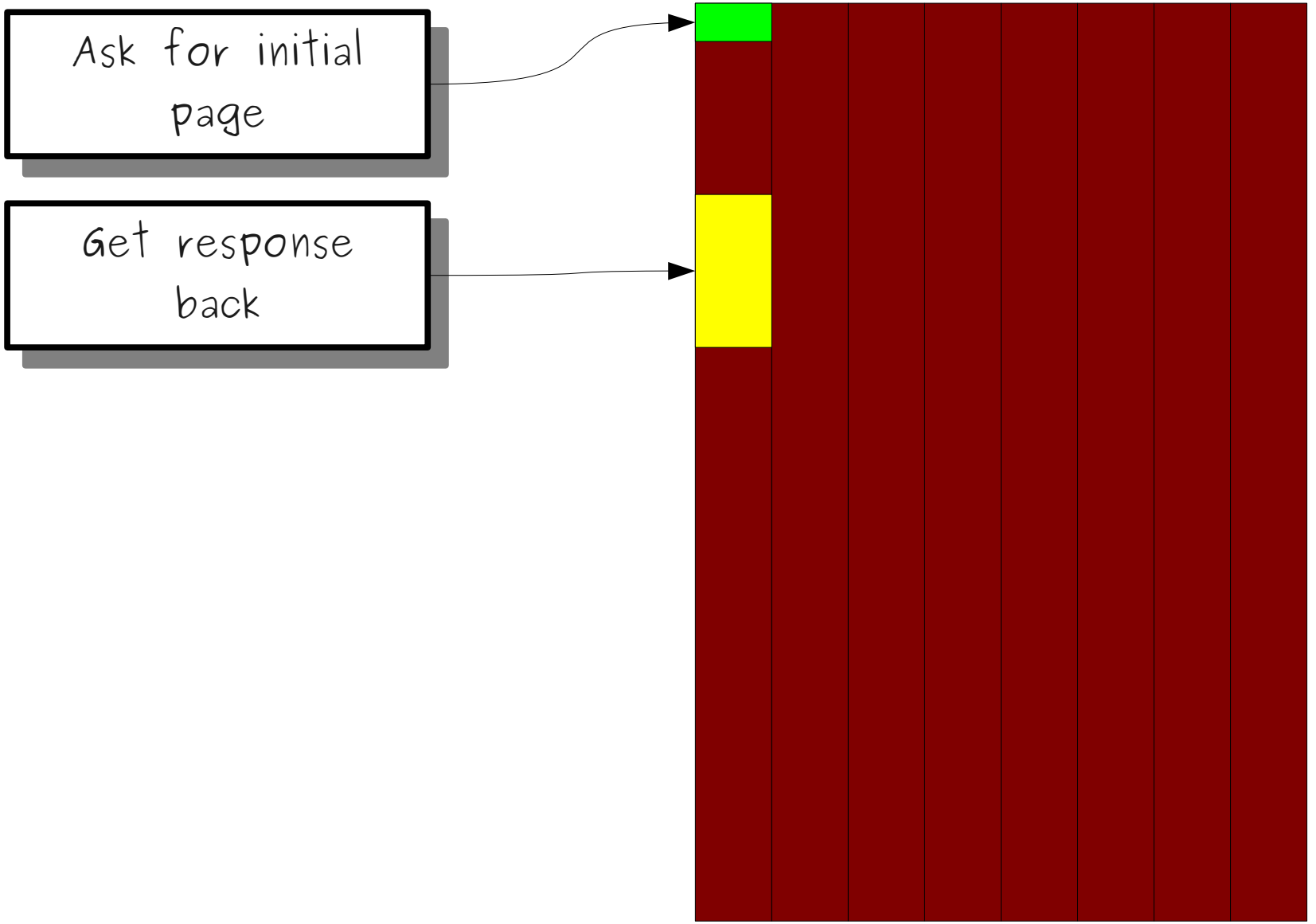
Network Latency



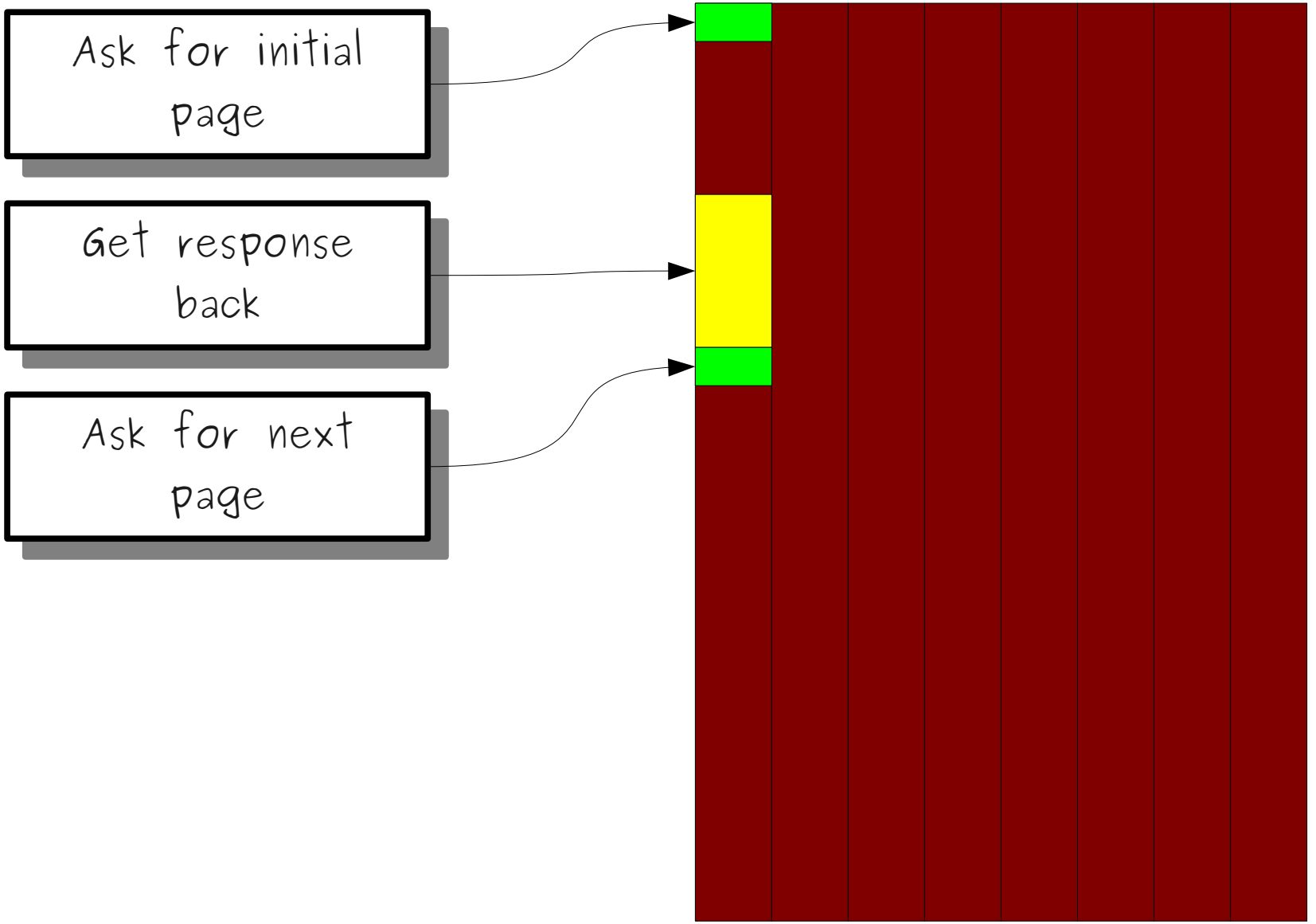
Network Latency



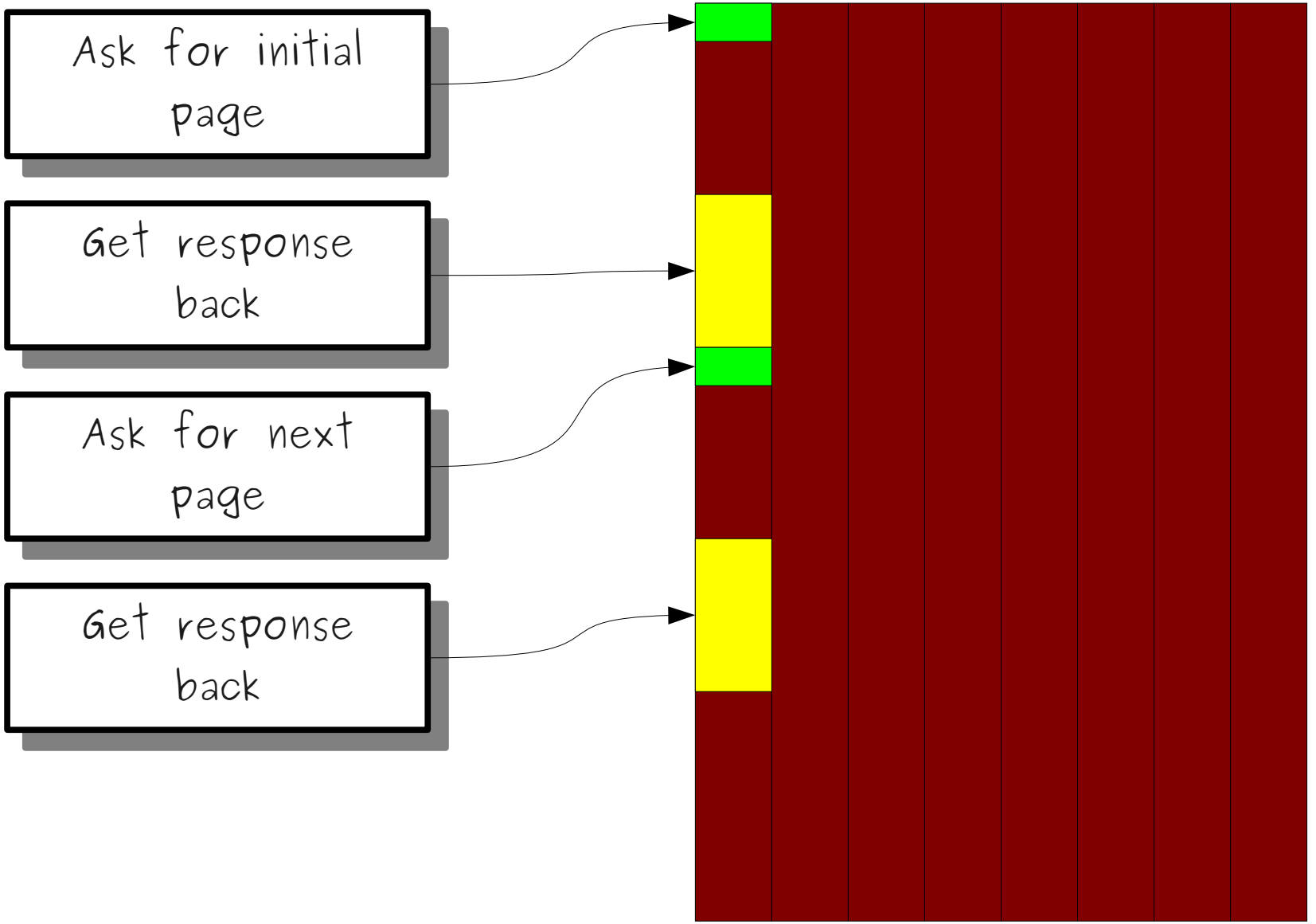
Network Latency



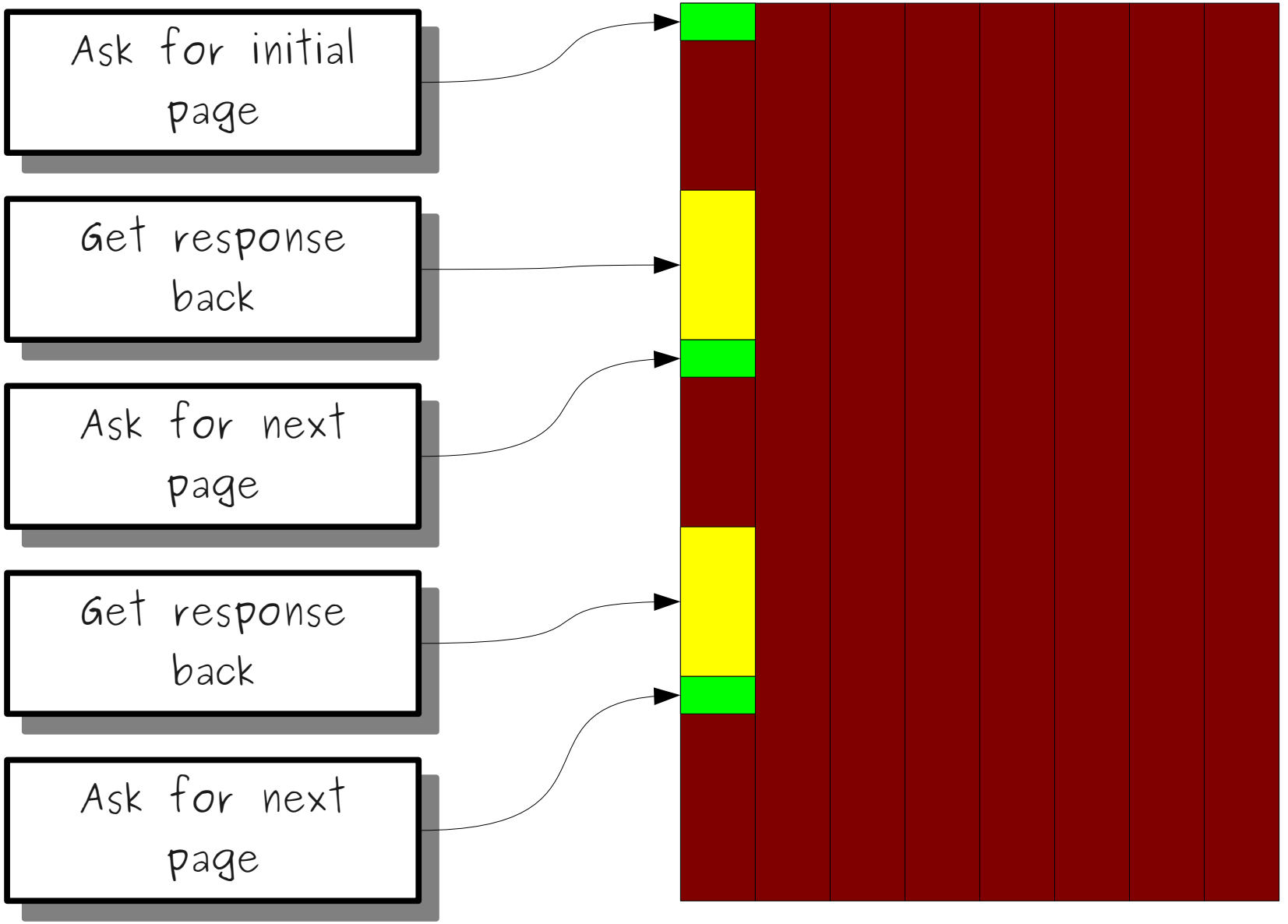
Network Latency



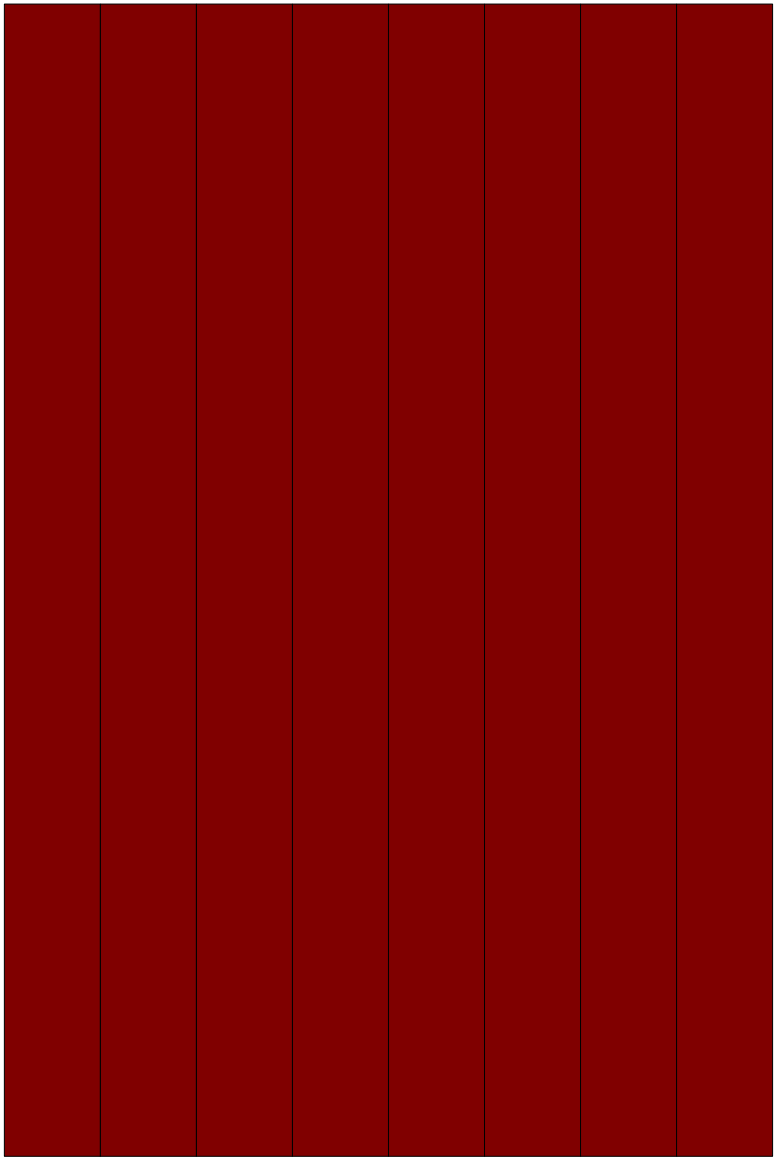
Network Latency



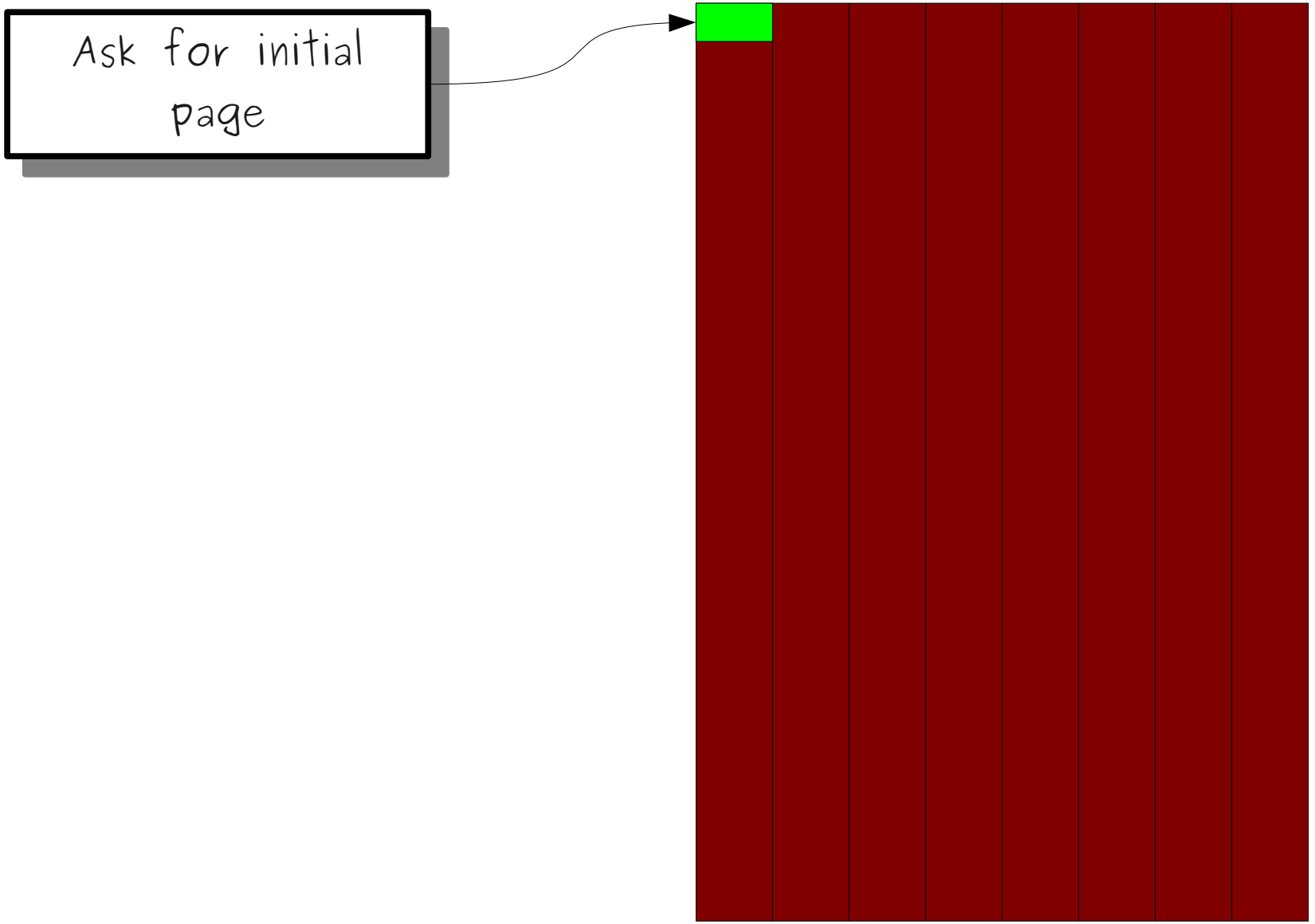
Network Latency



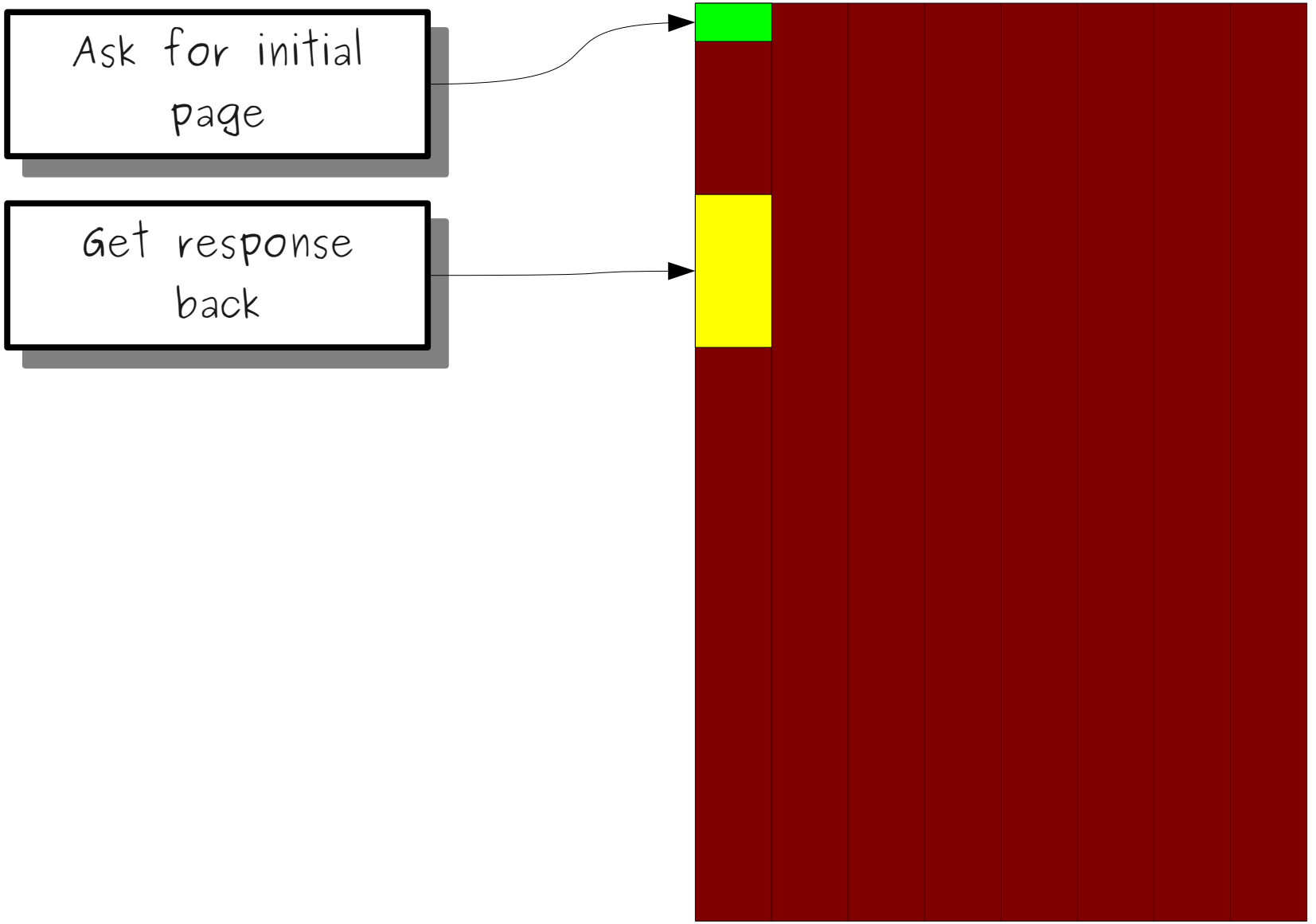
Network Latency



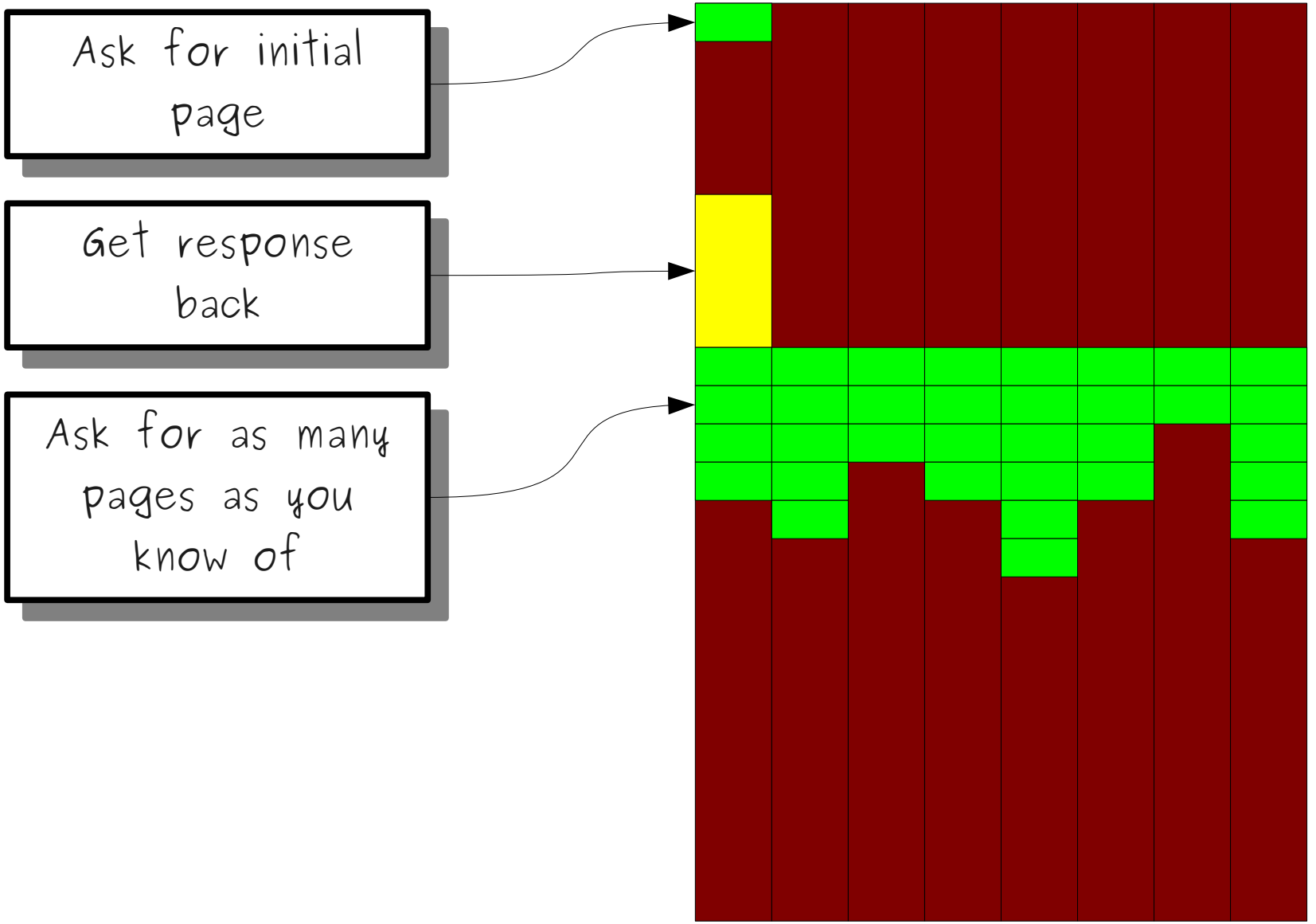
Network Latency



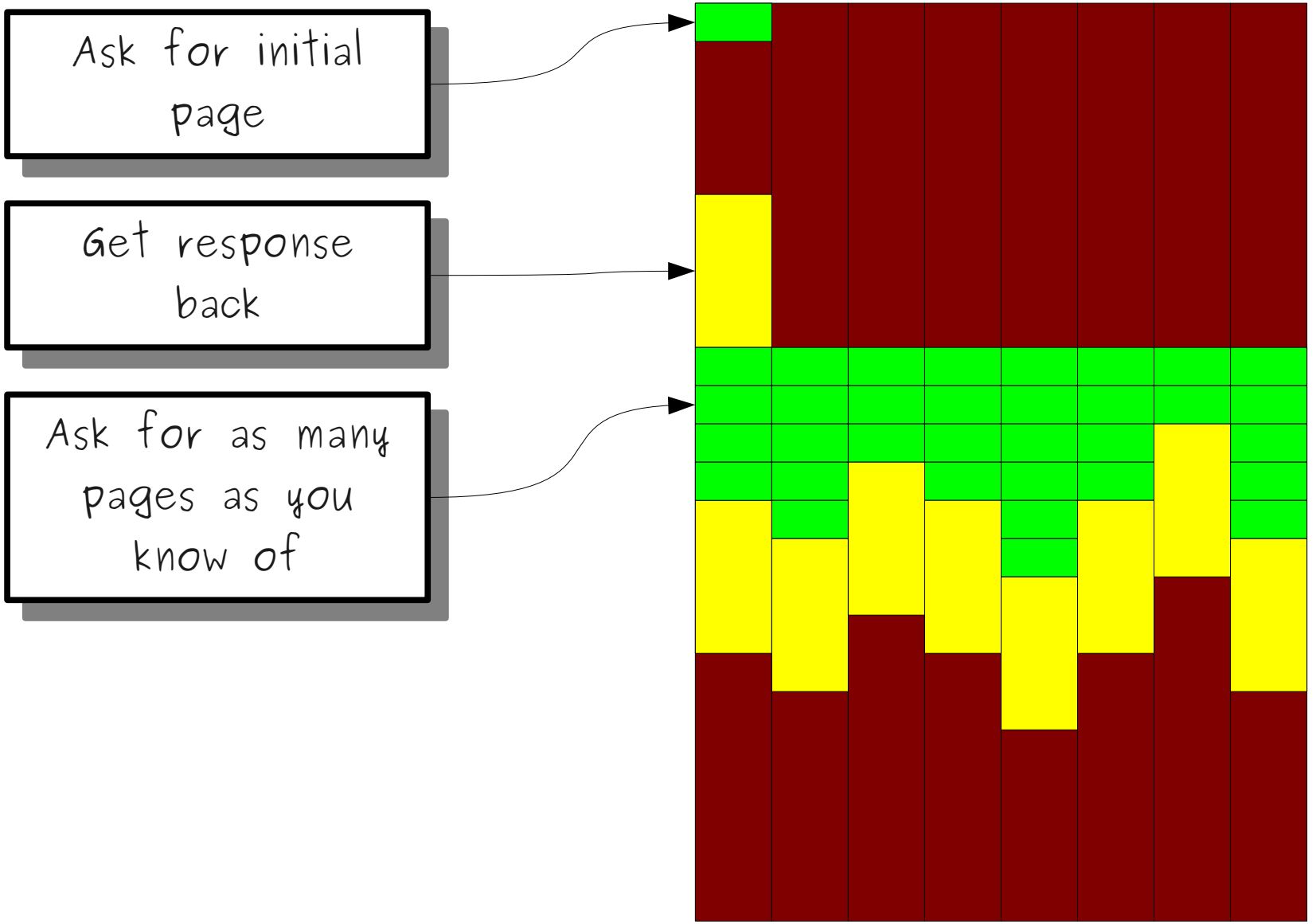
Network Latency



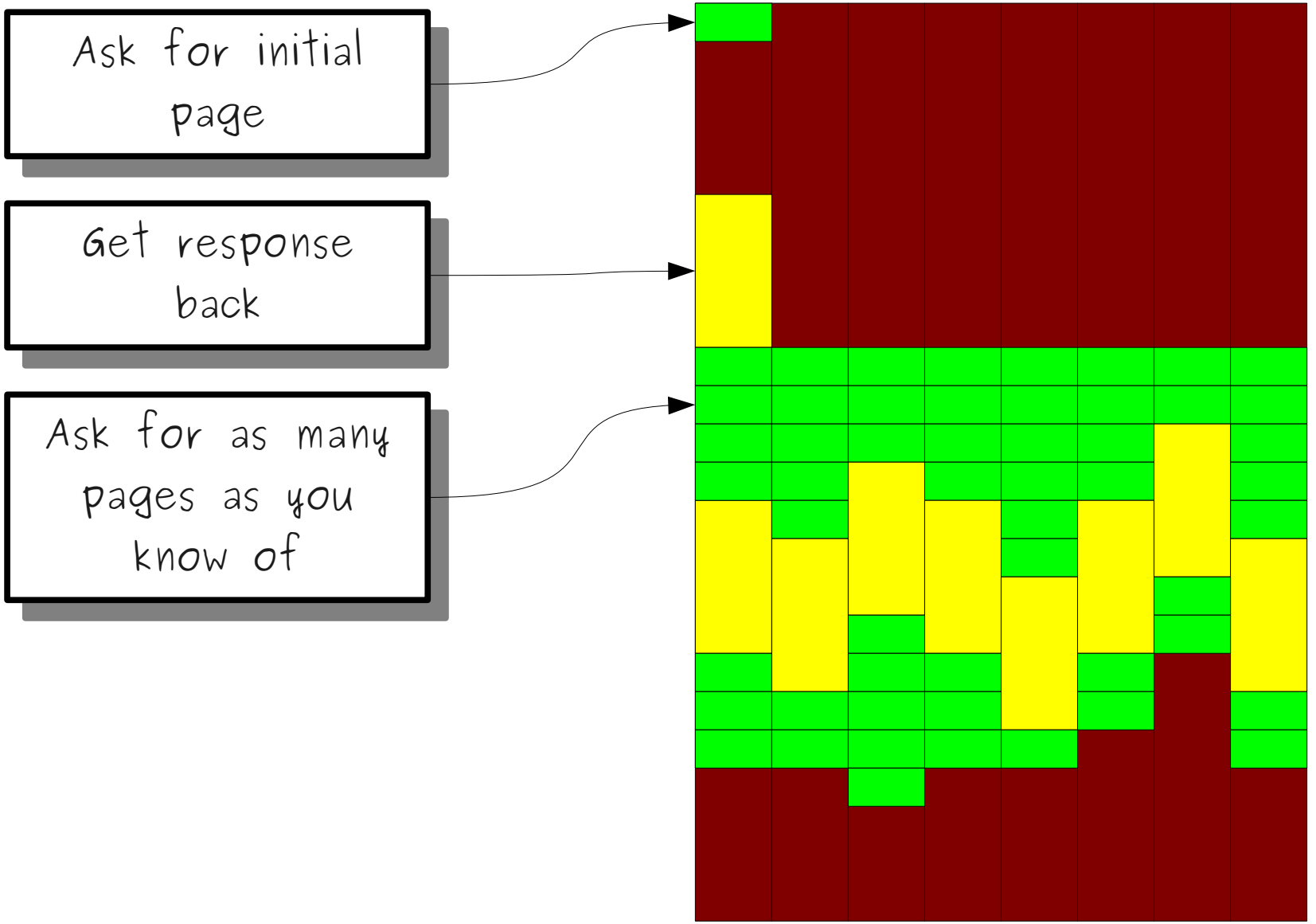
Network Latency



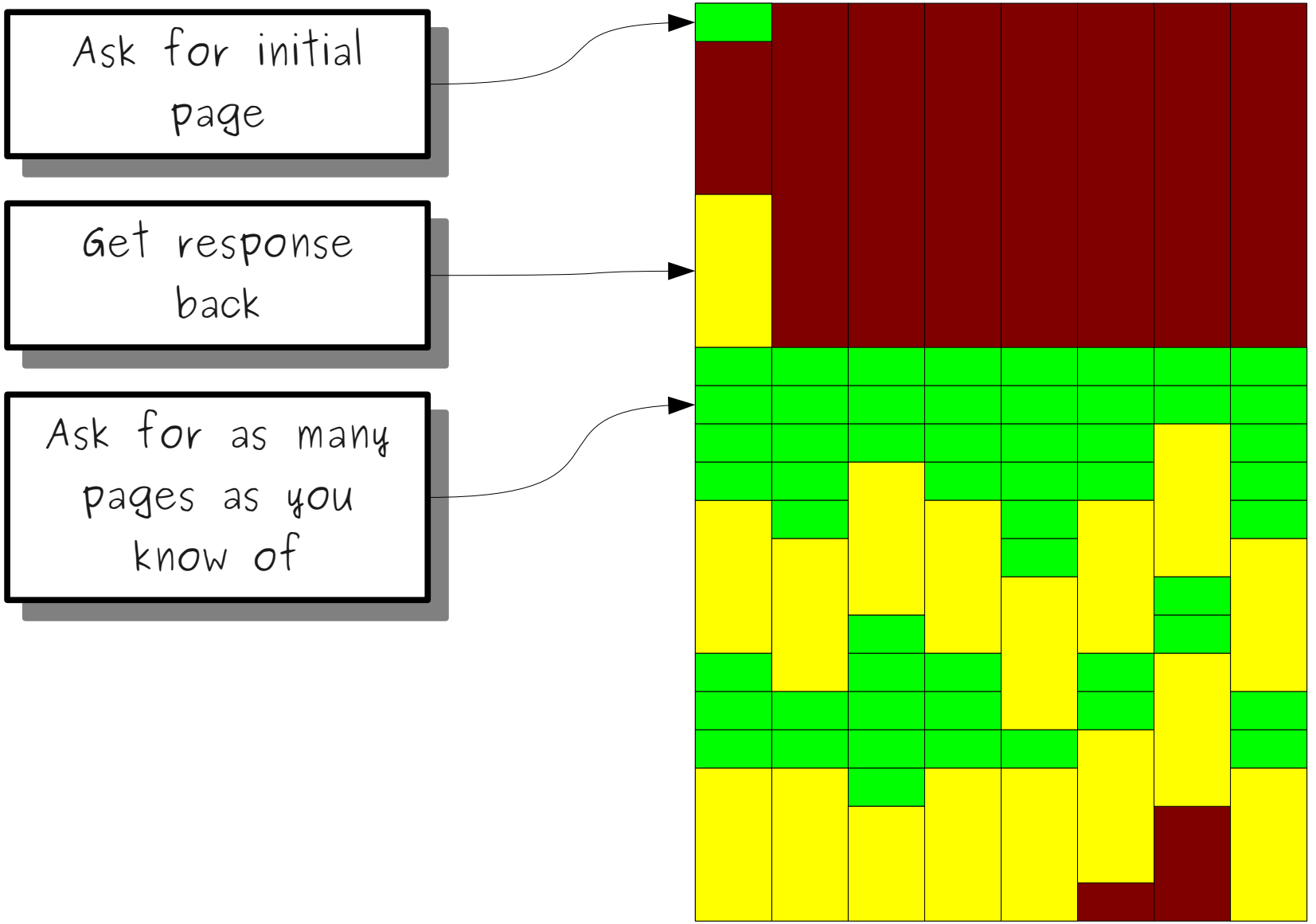
Network Latency



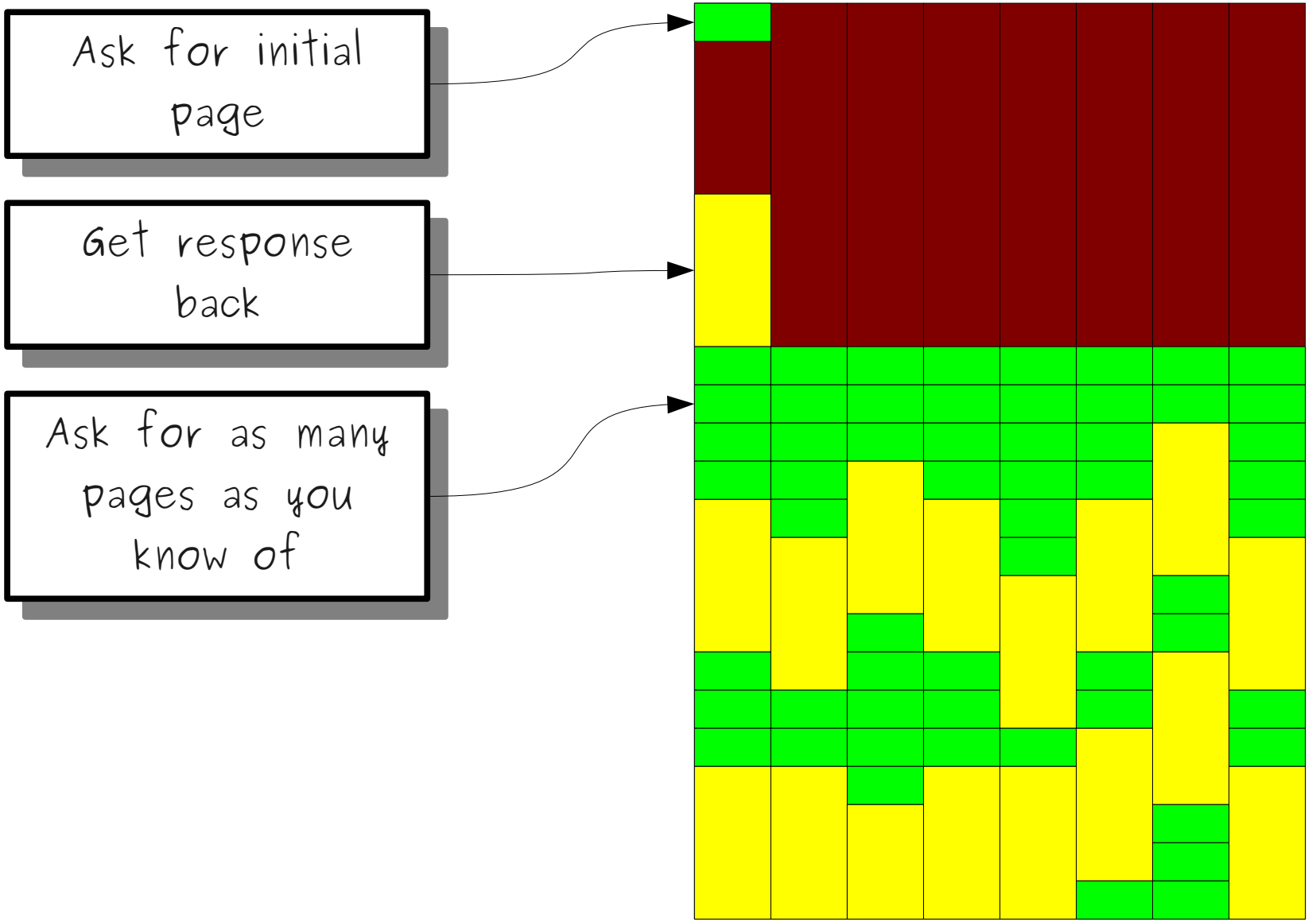
Network Latency



Network Latency



Network Latency



What's Happening?

- Network operations are called **I/O-bound** operations because most of the work done is waiting for I/O operations, not computation.
  - Those tasks are called **CPU-bound**.
- Having a huge number of threads improves efficiency because the computer is always working while waiting for the network.

A Parallelism Sampler

GPU Processing

CPUs and GPUs

- A CPU (**central processing unit**) is the actual hardware that runs your programs.
- A GPU (**graphics processing unit**) is a separate piece of hardware for displaying images on the screen.

CPUs versus GPUs

- A typical CPU has hardware to run between 1 – 8 threads at a time.
  - Each thread can do whatever it wants independently of the others.
- A typical GPU can run hundreds or thousands of threads at a time.
  - Each thread executes the same code as all the others, but processes different data.

GPU Parallelism

- GPUs can be used to parallelize mathematically intense tasks.
- Leads to enormous speedups.

Taking It Further...

Distributing Computing

- A **distributed system** is a system of computers that all work together to solve some large problem.
- Similar to threads – each computer works in parallel with the rest.
- Different from threads – each computer can only access its own memory.

Folding@home DISTRIBUTED COMPUTING

Client statistics by OS

OS Type	Native TFLOPS*	x86 TFLOPS*	Active CPUs	Total CPUs
Windows	218	218	210192	4231570
Mac OS X/PowerPC	2	2	3075	151743
Mac OS X/Intel	79	79	19219	171988
Linux	138	138	51043	706959
ATI GPU	2325	2453	16372	308636
NVIDIA GPU	1072	2262	6745	275700
PLAYSTATION®3	676	1426	23973	[illegible]
Total	4520	6588	337652	8214054

Looking Further

- Interested in parallelism?
- **Writing Parallel Code**:
  - CS110
  - CS149
- **Writing Distributed Systems**:
  - CS244B
- **Implementing Threads**:
  - CS140
  - CS240

Next Time

Where to Go from Here