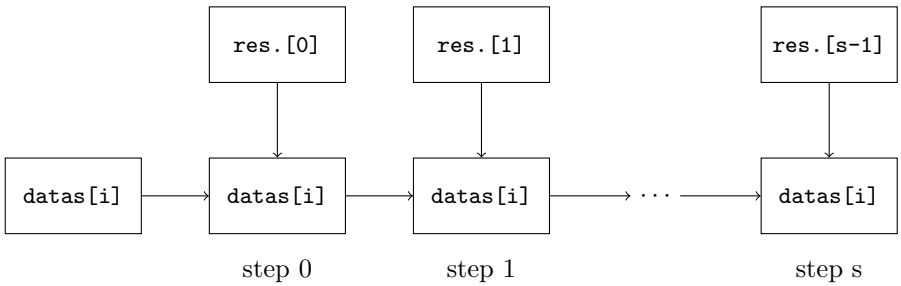
Pipelining

1 Description of the problem

Let's assume the case where we have a certain number `ndatas` of data to process. Processing of one data can be split in an arbitrary number of steps `nsteps`, which have to be performed in order. In order to execute one step, we need to use a dedicated resource; therefore, if we decide to split the processing of one data into `nsteps` steps, we will also have `nsteps` resources and to execute step `s` we need to use resource `s`:



The time for executing one step is $t_s = t/s$ where t is the time when the processing is done in only one step and s is equal to the number of steps `nsteps`.

This exercise is about parallelizing the processing of all the `ndatas` data sharing the available `nsteps` resources.

2 Package content

In the `pipelining` directory you will find the following files:

- `main.c`: this file contains the main program which first calls the `init_data` routine which initializes the `datas` array of size `ndatas`; each entry of this array contains a data that has to be processed. The main program then calls the `pipelining` routine that processes all the entries of the `datas` array by applying the defined number of steps and using the associated resources. Finally the correctness of the result is check.**Only this file has to be modified for this exercise**.

- `aux.c, aux.h`: these two files contain auxiliary routines and **must not be modified**.

The code can be compiled with the `make` command: just type `make` inside the `pipelining` directory; this will generate a `main` program that can be run like this:

```
$ ./main ndatas nsteps
```

where `ndatas` is the number of data and `nsteps` the number of steps to process each data.

3 Assignment

- ⌨ At the beginning, the `pipelining` is sequential and simply goes through all the data in natural order and applies to each of them the required steps. Modify this routine in order to parallelize it.
- ✎ Report the execution times for the implemented parallel version with 1, 2 and 4 threads. What speedup could you achieve? analyze and comment on your results. Report your answer in the form of comments at the bottom of the `main.c` file.

Advice

- Nothe that the resources have to be shared among all the working threads and therefore you have to implement a stretegy to prevent multiple processes to use the same resource at the same time.