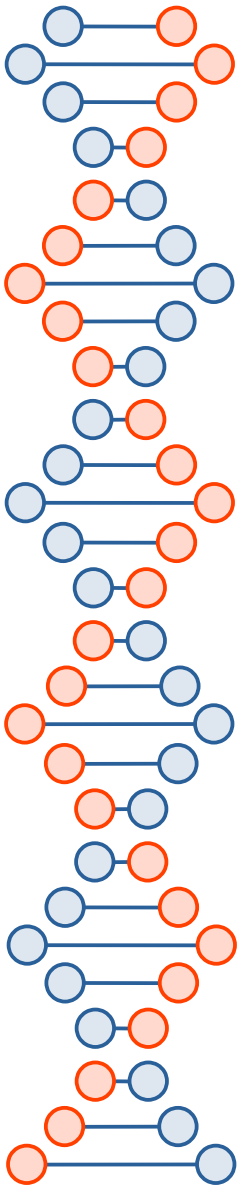
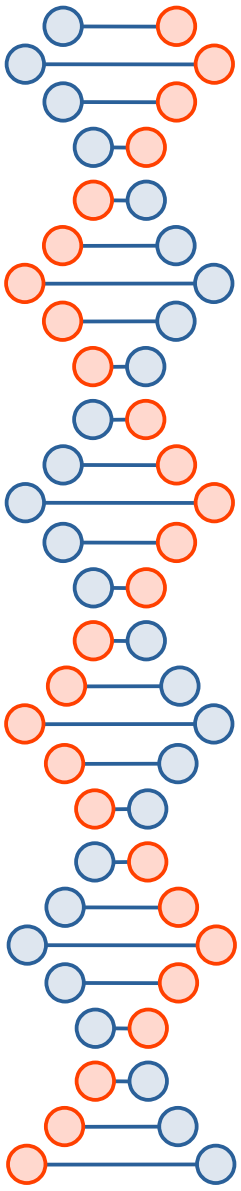


High Performance Computing

Ch. 5 Basics of Parallelization

Computer Eng., KMITL
Assoc. Prof. Dr. Surin. K.



Ch.5 Basics of Parallelization

5.1 Why parallelize?
5.2 Parallelism

- 5.2.1 Data parallelism
- 5.2.2 Functional parallelism

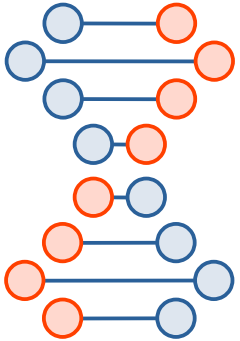
5.3 Parallel scalability

- 5.3.1 Factors that limit parallel execution
- 5.3.2 Scalability metrics
- 5.3.3 Simple scalability laws
- 5.3.4 Parallel efficiency
- 5.3.5 Serial performance versus strong scalability
- 5.3.9 Load imbalance

5.1 Why parallelize?

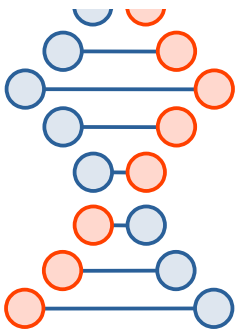
- A single core may be too slow to perform the required task(s) in a “tolerable” amount of time. The definition of “tolerable” certainly varies, but “overnight” is often a reasonable estimate. Depending on the requirements, “over lunch” or “duration of a PhD thesis” may also be valid.

- The memory requirements cannot be met by the amount of main memory which is available on a single system, because larger problems (with higher resolution, more physics, more particles, etc.) need to be solved.



5.2.1 Data parallelism

Many problems in scientific computing involve processing of large quantities of data stored on a computer. If this manipulation can be performed in parallel, i.e., by multiple processors working on different parts of the data, we speak of *data parallelism*. As a matter of fact, this is the dominant parallelization concept in scientific computing on MIMD-type computers. It also goes under the name of *SPMD* (Single Program Multiple Data), as usually the same code is executed on all processors, with independent instruction pointers. It is thus not to be confused with SIMD parallelism.



Example: Medium-grained loop parallelism Number of systems versus core count

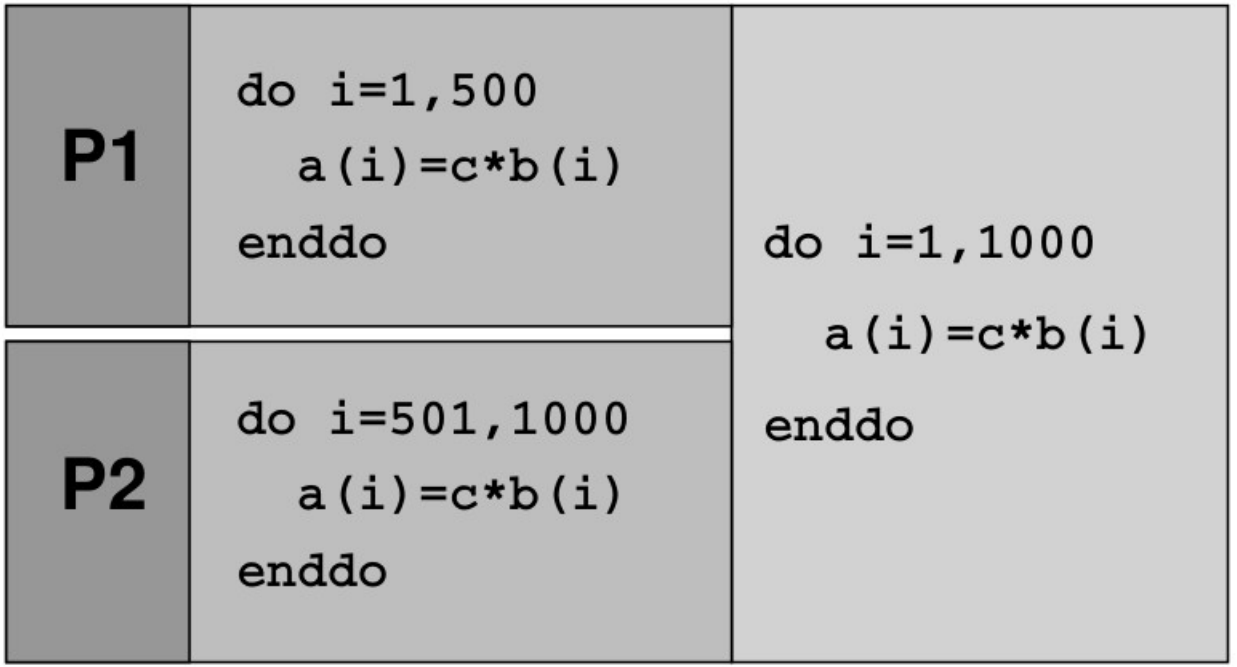
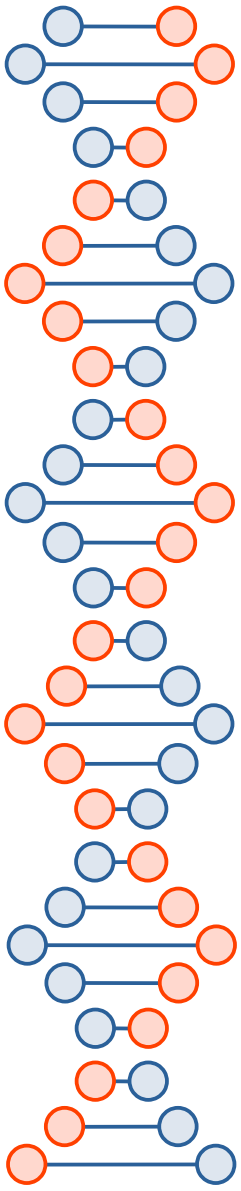


Figure 5.1: An example for medium-grained parallelism: The iterations of a loop are distributed to two processors P1 and P2 (in shared memory) for concurrent execution.



5.2.2 Functional parallelism

- Example: Master-worker scheme
- Example: Functional decomposition

5.2.2 Functional parallelism Example: Master-worker scheme

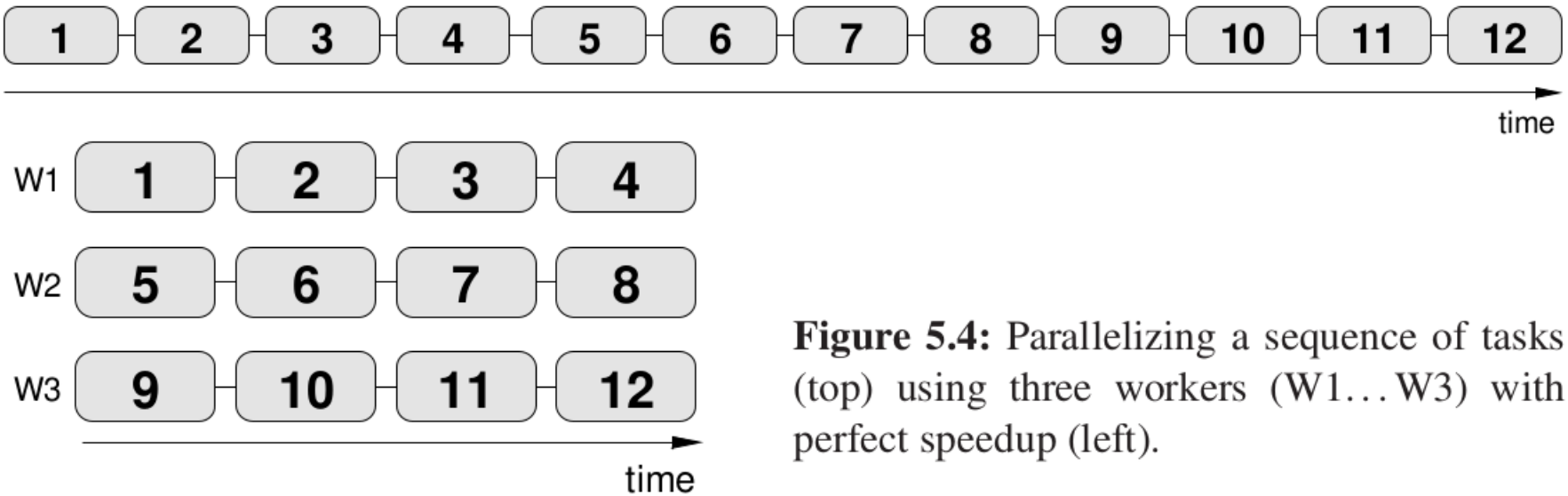


Figure 5.4: Parallelizing a sequence of tasks (top) using three workers (W1...W3) with perfect speedup (left).

5.3 Parallel Scalability
5.3.1 Factors that limit parallel execution

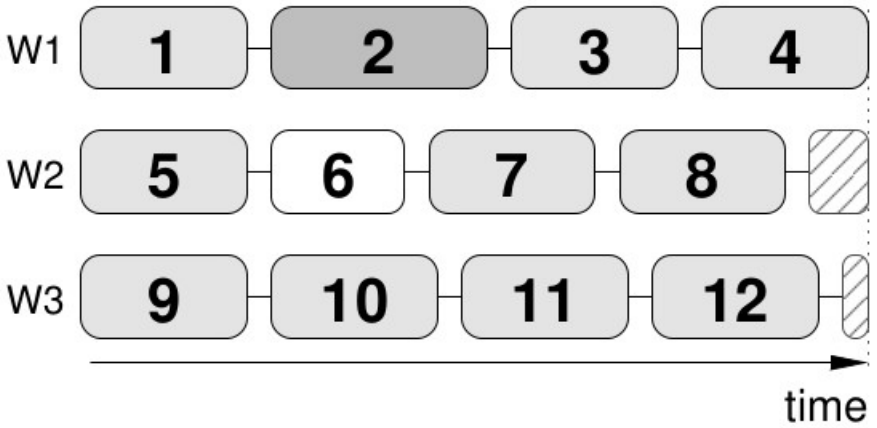


Figure 5.5: Some tasks executed by different workers at different speeds lead to *load imbalance*. Hatched regions indicate unused resources.

5.3 Parallel Scalability
5.3.1 Factors that limit parallel execution

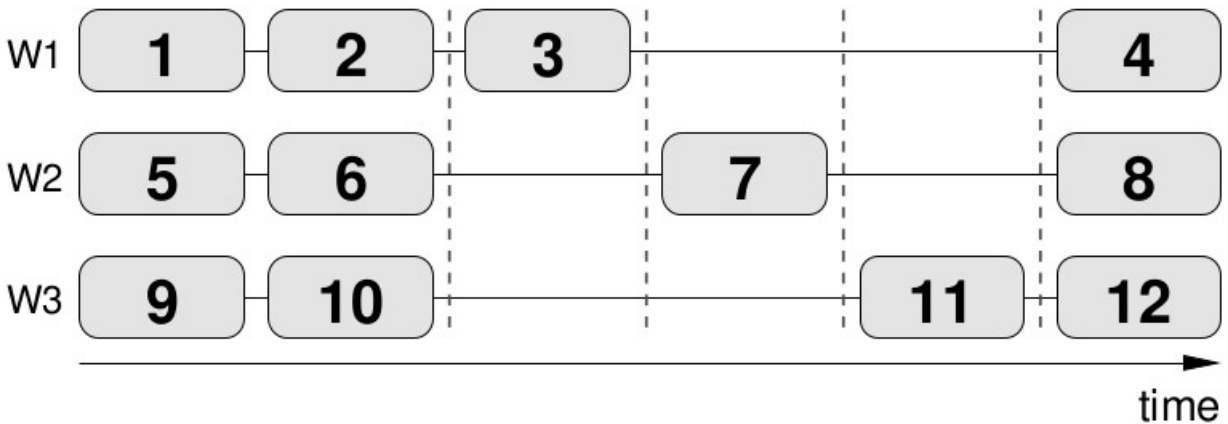


Figure 5.6: Parallelization with a bottleneck. Tasks 3, 7 and 11 cannot overlap with anything else across the dashed "barriers."

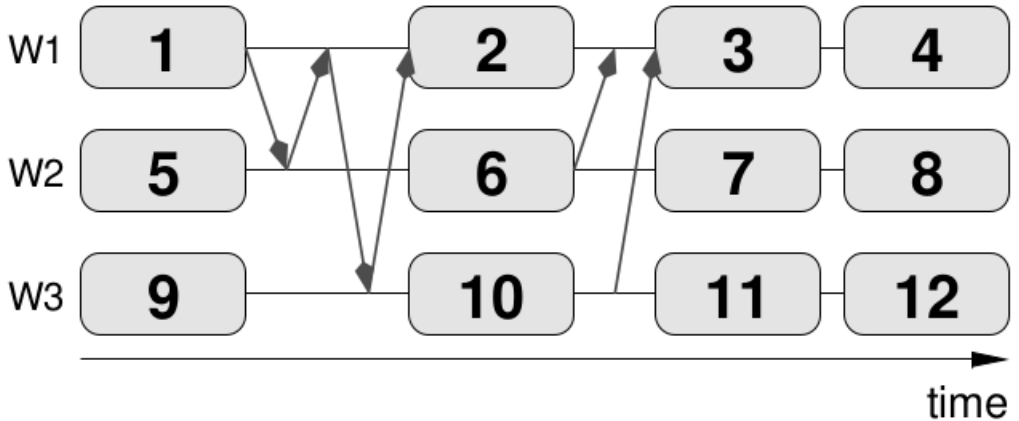
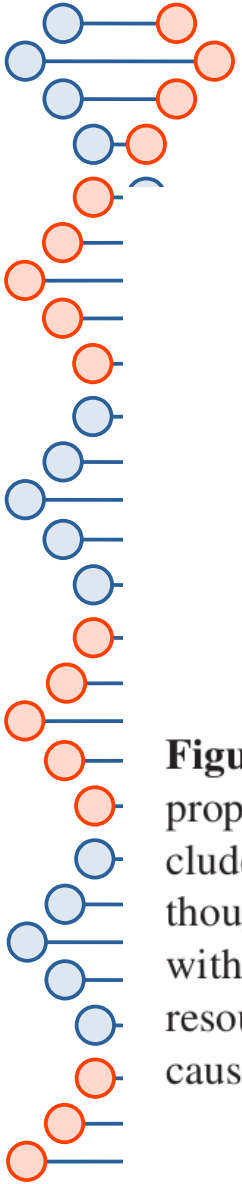


Figure 5.7: Communication processes (arrows represent messages) limit scalability if they cannot be overlapped with each other or with calculation.



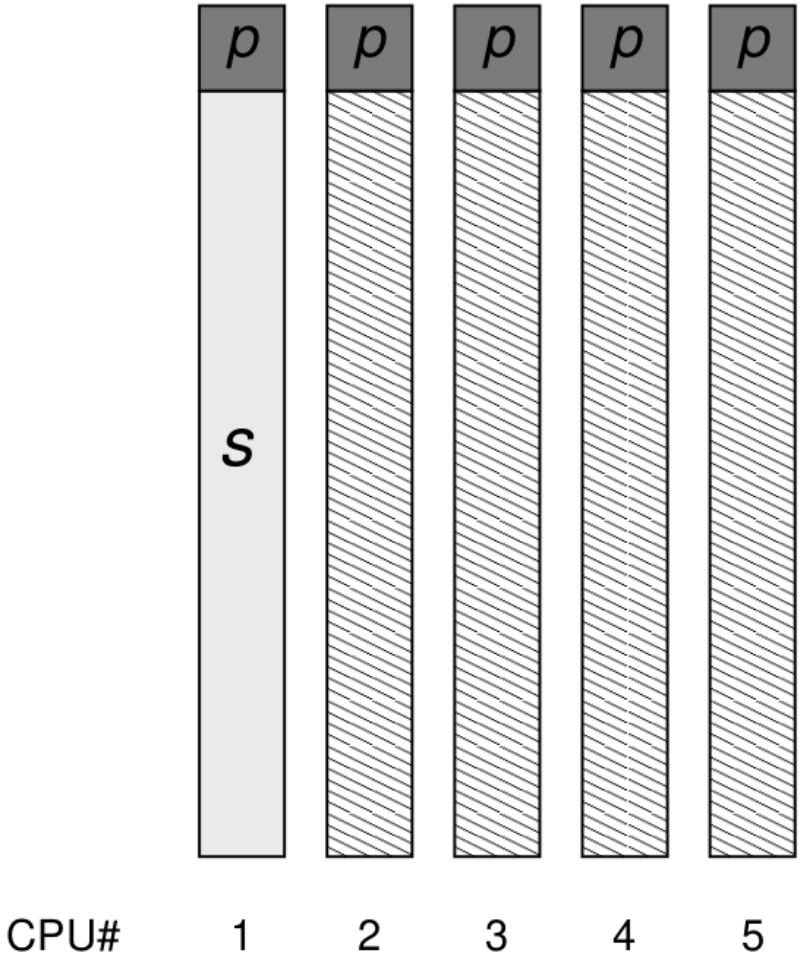


Figure 5.8: Weak scaling with an inappropriate definition of "work" that includes only the parallelizable part. Although "work over time" scales perfectly with CPU count, i.e., $\varepsilon_p = 1$, most of the resources (hatched boxes) are unused because $s \gg p$.

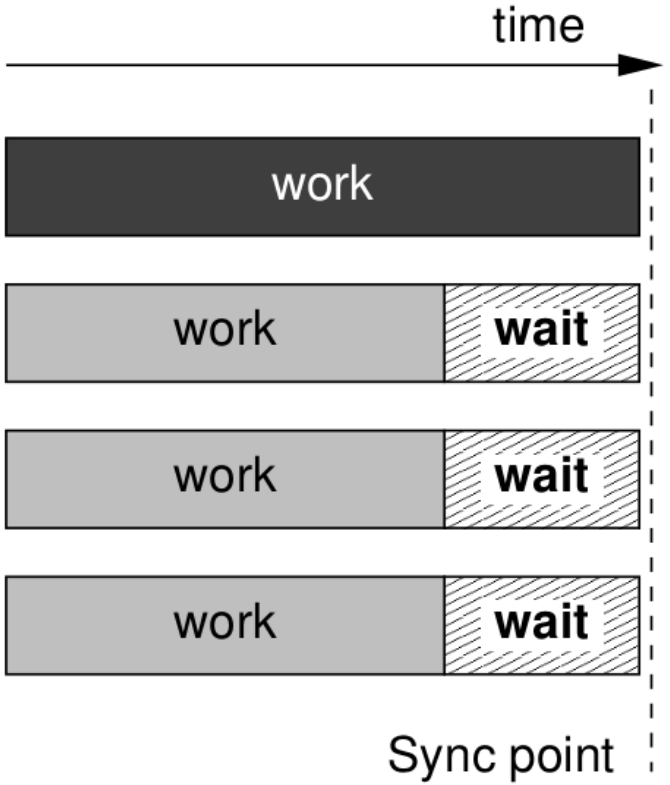


Figure 5.13: Load imbalance with few (one in this case) "laggers": A lot of resources are underutilized (hatched areas).

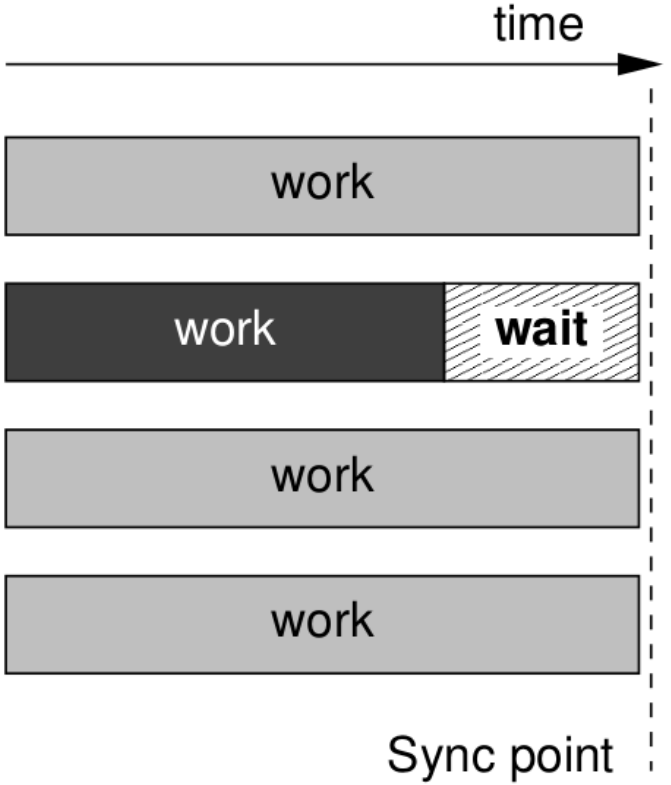
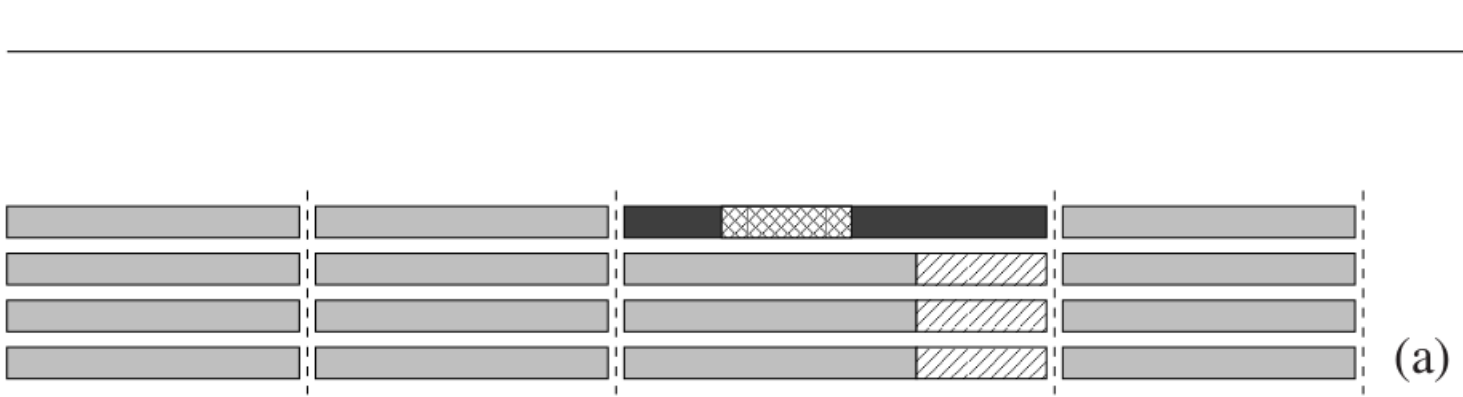
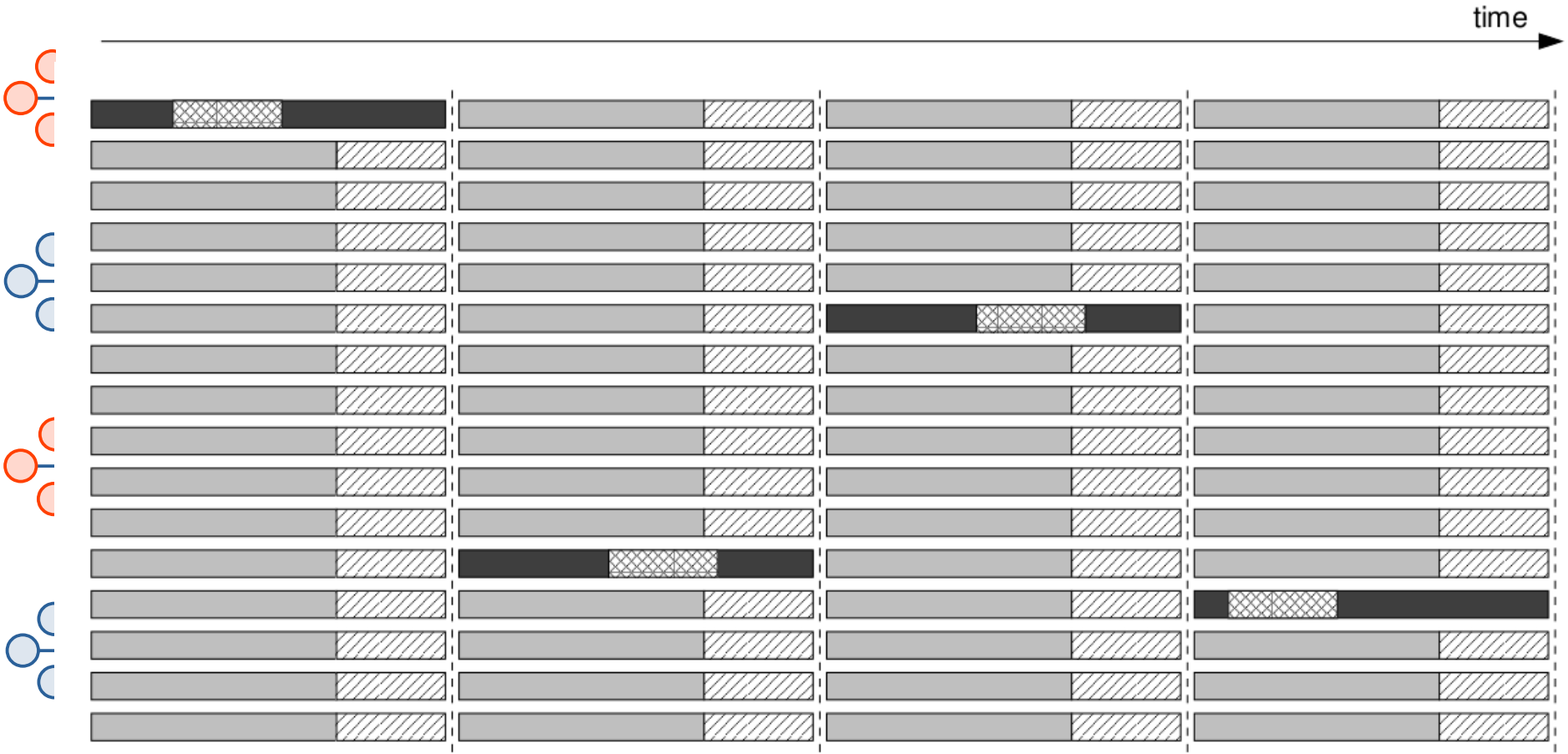
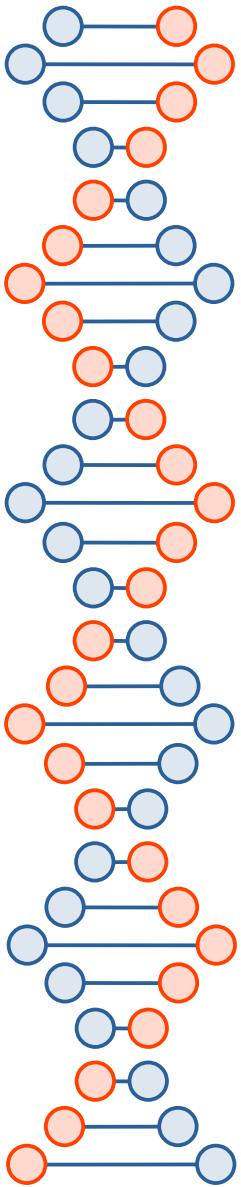


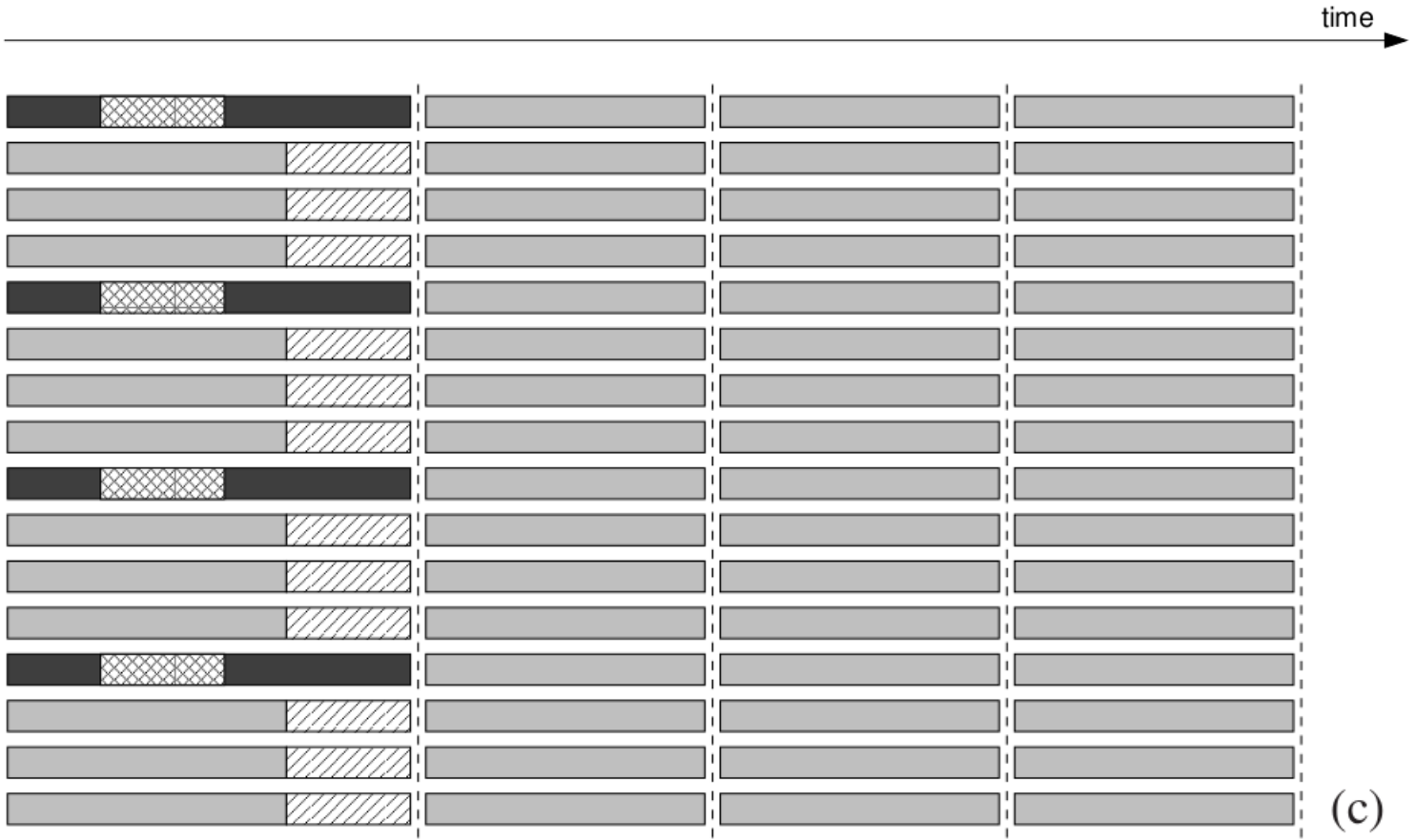
Figure 5.14: Load imbalance with few (one in this case) "speeders": Underutilization may be acceptable.


time
(a)



(b)




time
(c)