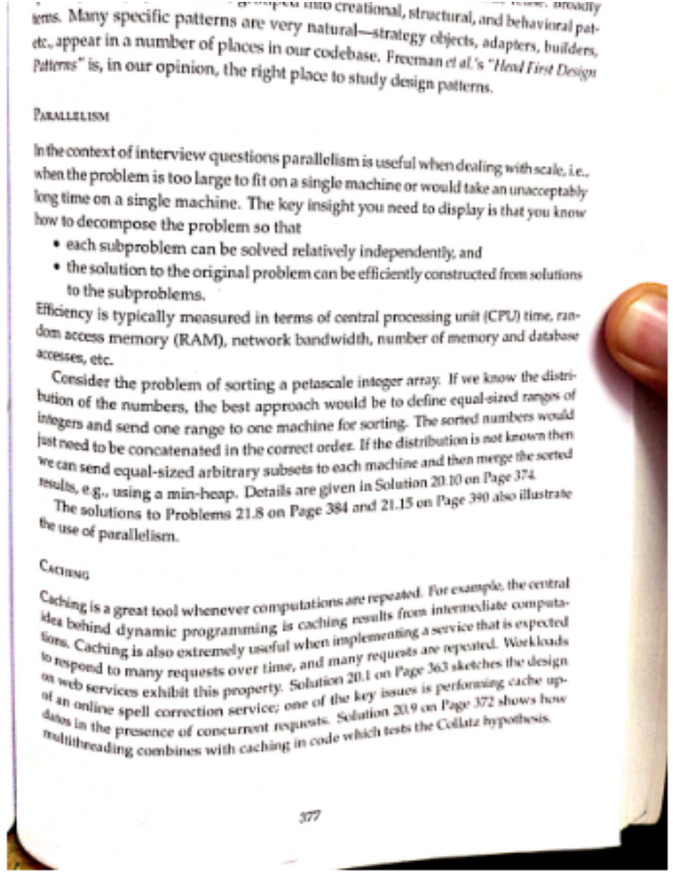


tems. Many specific patterns are very natural—strategy objects, adapters, builders, etc., appear in a number of places in our codebase. Freeman *et al.*'s "*Head First Design Patterns*" is, in our opinion, the right place to study design patterns.

Parallelism

In the context of interview questions parallelism is useful when dealing with scale, i.e., when the problem is too large to fit on a single machine or would take an unacceptably long time on a single machine. The key insight you need to display is that you know how to decompose the problem so that

- each subproblem can be solved relatively independently, and
- the solution to the original problem can be efficiently constructed from solutions to the subproblems.

Efficiency is typically measured in terms of central processing unit (CPU) time, random access memory (RAM), network bandwidth, number of memory and database accesses, etc.

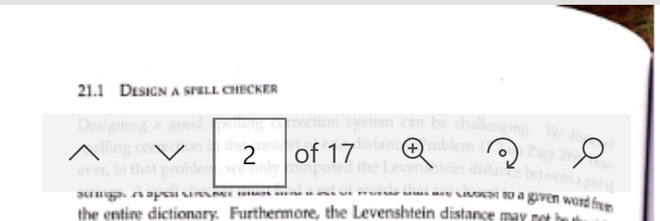
Consider the problem of sorting a petascale integer array. If we know the distribution of the numbers, the best approach would be to define equal-sized ranges of integers and send one range to one machine for sorting. The sorted numbers would just need to be concatenated in the correct order. If the distribution is not known then we can send equal-sized arbitrary subsets to each machine and then merge the sorted results, e.g., using a min-heap. Details are given in Solution 20.10 on Page 374.

The solutions to Problems 21.8 on Page 384 and 21.15 on Page 390 also illustrate the use of parallelism.

Caching

Caching is a great tool whenever computations are repeated. For example, the central idea behind dynamic programming is caching results from intermediate computations. Caching is also extremely useful when implementing a service that is expected to respond to many requests over time, and many requests are repeated. Workloads on web services exhibit this property. Solution 20.1 on Page 363 sketches the design of an online spell correction service; one of the key issues is performing cache updates in the presence of concurrent requests. Solution 20.9 on Page 372 shows how multithreading combines with caching in code which tests the Collatz hypothesis.

377



21.1 Design a spell checker