In the paper “Approximate Computing and the Quest for Computing Efficiency” the authors look to explain what designing an approximate computer architecture would entail and the principles they should strive to adhere to. The author's proposal is through the use of approximate computing, by following key principles we can design computers in such a way that we receive a good enough quality output given an approximate input. In the current age of computing we want a precise answer, however, sometimes this answer does not exist, or it is computationally expensive to the point that a good enough answer can be better. Furthermore there are a few principles which the authors provide, which should guide approximate computer architecture.

1. Measurable notion of quality, we want to know what a “good enough” answer is, does that mean we are within a 5% error of margin of the precise answer or is a 30% error of margin enough? This matters as it is important to maintain a good end of product quality, we can’t have a 20% margin of error in health applications but that may be fine for identifying birds.
2. We cannot simplify and approximate everything. Some calculations are far more tolerant to being approximated. For instance adding two 16 bit numbers losing the first 4 bits may not affect the outcome drastically, while approximating a memory address can be catastrophic accessing memory which should not be.
3. Approximations give large advantages compared to the loss of quality for the final output. Meaning if we lose a 5% accuracy in quality we must receive a 40% decrease in power for instance.
4. We focus on achieving the best performance vs efficiency tradeoff rather than aiming for a specific performance.

This can be applied to all levels of computing, circuitry, architecture, and software. The examples they give for each of these is reducing the number of transistors in an adder by being okay with the output table being slightly different in output for a few values. With Architecture they mention using less of each of these cells. Going back to the adder example, only representing the 6 MSB of a 8 bit int can save 2 full adders, saving space and energy while receiving a good enough answer. The example they give for software is DNN in which certain neurons are updated less and use simpler functions since they are not as essential to the final output (pruning).

Ultimately I think that this is a good paper, it shows that in certain situations approximation is just as good as having an exact answer and may even be better, however we must design for these systems to be intrinsically resilient to approximations by following a few design principles. While the authors do provide some examples in each of these domains it is a high level explanation of them. If I were to improve anything in this paper it would be to dive a little deeper into these examples.