Chapter 1: Why Quantum Computing Exists

Quantum computing exists because two powerful ideas meet.

The first idea is that computation is not just a human activity. It is not only what happens when a person does arithmetic or when a programmer writes code. Computation is a physical process: information is stored in some physical system, changed by physical operations, and eventually read out as a result. A notebook stores information using ink on paper. A laptop stores information using electric charges and magnetic or electronic states. A brain stores and processes information through biological activity. As Rolf Landauer famously emphasized, information is physical: every real act of information processing must be carried by some physical system (Landauer, 1961).

The second idea is that the deepest known rules of microscopic physics are quantum rules. At the scale of atoms, electrons, photons, and molecules, nature does not behave exactly like tiny billiard balls following ordinary everyday intuition. It behaves according to quantum mechanics. If computation is physical, and if the physical world is quantum at its foundations, then it is natural to ask:

What kinds of computation become possible when we use quantum systems directly?

That question is the beginning of quantum computing.

This chapter is a first orientation. We will not yet build quantum circuits or calculate with qubits. Instead, we will understand why the subject exists at all: why classical computers are so powerful, where they struggle, and why quantum physics suggests a different model of computation.

1.1 Classical computers are already extraordinary

Before asking why we need a new kind of computer, we should appreciate the old kind.

A classical computer is a computer whose information can be described using ordinary definite states. At the simplest level, modern digital computers store information using bits. A bit is a value with two possible states, usually written as 0 and 1.

One bit is tiny. But many bits can represent rich information.

For example:

- A yes/no answer can be stored in one bit: 0 for no, 1 for yes.
- A letter can be represented by a pattern of bits.
- A photograph can be represented by many numbers, and those numbers can be represented by bits.
- A video is a sequence of images, so it can also be represented by bits.
- A program is itself stored as bits.

The power of digital computing comes from the fact that once information is represented as bits, we can transform it using simple reliable rules. A computer does not “understand” a photograph the way a person does. It follows operations on bit patterns. Yet with enough carefully designed operations, those bit patterns can become images, music, simulations, games, messages, maps, and scientific calculations.

This idea is not new. Alan Turing’s 1936 work gave a mathematical model of computation showing how a simple abstract machine could perform any calculation that can be described by a step-by-step procedure, now called an algorithm (Turing, 1936). The modern computer is vastly more complex as engineering, but conceptually it is built on this remarkable insight: complicated computation can be assembled from simple symbolic operations.

An algorithm is a precise procedure for solving a problem. For example, suppose you want to find the largest number in this list:

```
4, 9, 2, 15, 6
```

A simple algorithm is:

1. Remember the first number, 4, as the largest seen so far.
2. Compare it with 9. Since 9 is larger, remember 9.
3. Compare it with 2. Keep 9.
4. Compare it with 15. Since 15 is larger, remember 15.
5. Compare it with 6. Keep 15.
6. Output 15.

This is computation in a very basic form: store information, compare information, update information, output information.

Modern computers perform billions of such low-level operations per second. They are among the most successful technologies humans have ever built.

So why not stop there?

1.2 Power is not the same as unlimited power

Classical computers are powerful, but they are not unlimited.

The important issue is not whether a computer can solve a problem in principle. The important issue is often whether it can solve the problem using realistic amounts of time, memory, and energy.

This brings us to a central idea: computational cost.

The computational cost of an algorithm is the amount of resources it needs. The most common resources are:

- Time: how many steps the algorithm takes.
- Memory: how much information it must store while running.
- Energy: how much physical energy the computation consumes.
- Hardware size: how many physical devices are needed.

For this book, we will often focus on time and memory, because they are easiest to reason about mathematically. But it is important to remember that real computation is physical, so time, memory, and energy are connected to engineering limits.

A problem becomes difficult when the resources needed grow too quickly as the input size increases.

Suppose you have an algorithm that checks every possible password of length n, where each character can be one of 10 digits. Then the number of possibilities is:

```
10^n
```

If n = 3, there are 1,000 possibilities. That is easy.

If n = 12, there are 1,000,000,000,000 possibilities. That is much harder.

The difference is not just that the second problem is “a little bigger.” The number of possibilities grows explosively. This kind of growth is called exponential growth.

Exponential growth appears when each additional piece of input multiplies the number of possibilities. For example:

```
2^10 = 1,024
2^20 = 1,048,576
2^30 ≈ 1 billion
2^100 ≈ 1.27 × 10^30
```

The expression 2^100 means 2 multiplied by itself 100 times. That number is far beyond anything we can check one by one by ordinary methods.

This is one of the main reasons quantum computing matters. Many important problems are not hard because we lack cleverness or because our computers are slow today. They are hard because the number of possibilities grows so quickly that even enormous improvements in hardware may not be enough.

1.3 Better hardware helps, but it does not solve every scaling problem

For decades, ordinary computers improved dramatically because engineers learned how to place more and more components on integrated circuits. Gordon Moore observed in 1965 that the number of components on an integrated circuit had been increasing rapidly, a trend later called Moore's law (Moore, 1965). This trend helped make computers smaller, cheaper, and faster.

But faster hardware does not remove the difference between easy scaling and hard scaling.

Imagine two algorithms for the same problem.

Algorithm A takes about:

```
n^2 steps
```

Algorithm B takes about:

```
2^n steps
```

Here n means the size of the input.

If n = 10, then:

```
n^2 = 100
2^n = 1,024
```

Both are manageable.

If n = 100, then:

```
n^2 = 10,000
2^n ≈ 1.27 × 10^30
```

Now the difference is enormous.

An algorithm whose cost grows like n^2, n^3, or another fixed power of n is called polynomial-time. An algorithm whose cost grows like 2^n, 3^n, or similar expressions is called exponential-time.

Polynomial-time algorithms are not automatically easy in practice, and exponential-time algorithms are not automatically impossible for small inputs. But as a broad rule, polynomial growth is usually much more manageable than exponential growth.

This is why computer science cares so deeply about scaling. A slow polynomial algorithm may become useful with better engineering. An exponential brute-force algorithm may remain hopeless even on extremely fast machines.

Quantum computing is not mainly about making each individual operation faster. It is about changing the model of computation so that some problems can be solved with a better scaling pattern.

That distinction is crucial.

A quantum computer is not simply a classical computer with a faster clock. It is a computer that processes information according to different physical rules.

1.4 The first motivation: nature itself is quantum

One of the earliest and deepest motivations for quantum computing came from the problem of simulating physics.

A simulation is a computation that imitates the behavior of a system. For example:

- A weather simulation imitates the atmosphere.
- A flight simulator imitates an airplane.
- A molecular simulation imitates atoms and chemical bonds.

Classical computers can simulate many physical systems very well. But quantum systems are especially difficult to simulate in full detail.

Why?

Because the mathematical description of a general quantum system grows extremely quickly as the number of parts increases. We will study this carefully later, but the basic idea can be introduced now.

A single ordinary bit has two possible values:

```
0 or 1
```

Two ordinary bits have four possible values:

```
00, 01, 10, 11
```

Three ordinary bits have eight possible values:

```
000, 001, 010, 011, 100, 101, 110, 111
```

In general, n bits have:

```
2^n
```

possible bit strings.

A classical computer storing n bits is in one of those possible strings at a time. For example, three bits might be in the state 101.

A quantum system is different. A general quantum state of n quantum bits, or qubits, requires amplitudes associated with all 2^n possible classical bit strings. An amplitude is a number used by quantum theory to calculate probabilities. We will define amplitudes precisely in later chapters. For now, the important point is that describing a general quantum state on an ordinary computer can require an amount of information that grows exponentially with the number of qubits.

For example:

- A 1-qubit state involves amplitudes for 0 and 1.
- A 2-qubit state involves amplitudes for 00, 01, 10, and 11.
- A 100-qubit state generally involves amplitudes for 2^100 possible bit strings.

This does not mean every 100-qubit system is impossible to describe compactly. Some quantum states have special structure that allows efficient classical descriptions. But a general quantum state has an exponentially large description in the usual representation, which is one reason quantum simulation can be so difficult on classical computers (Nielsen & Chuang, 2010).

Richard Feynman argued in 1982 that if nature is quantum, then simulating quantum physics may require computers that themselves operate quantum mechanically (Feynman, 1982). This was not just a practical engineering suggestion. It was a conceptual shift:

Instead of forcing quantum nature to fit inside a classical computer, build a computer that follows quantum rules.

This is perhaps the most natural reason quantum computing exists.

If we want to understand molecules, materials, superconductors, chemical reactions, or quantum fields, a quantum computer may be the right kind of machine because it speaks the same physical language as the systems being studied.

1.5 The second motivation: quantum rules allow new information processing

Quantum computing is not only about simulating physics. Quantum rules also allow new ways to process information.

To see why that is plausible, we need a first gentle look at several quantum ideas. We will define them carefully later. For now, we only need their motivational meaning.

A qubit is the quantum version of a bit. Like a bit, it can produce outcomes 0 or 1 when measured in the simplest standard way. But before measurement, its state is described by quantum amplitudes, not by a definite classical value alone.

A superposition is a quantum state that combines multiple classical possibilities through amplitudes. This phrase is often misexplained as “the qubit is both 0 and 1 at the same time.” That slogan can be misleading. A better first statement is:

A qubit can be in a state whose measurement probabilities are determined by amplitudes for 0 and 1.

For example, a qubit might be prepared so that if we measure it many times, about half the results are 0 and half are 1. But this does not mean the qubit secretly stores a normal random coin flip. Quantum amplitudes can also have phase, a kind of directional or sign-like information that affects how amplitudes combine.

That brings us to interference.

Interference is what happens when quantum amplitudes combine and either strengthen or cancel each other. A familiar wave analogy helps. If two water waves meet crest-to-crest, they can make a larger wave. If a crest meets a trough, they can cancel. Quantum amplitudes are not water waves, but they have a similar mathematical feature: they can add constructively or destructively.

This matters because quantum algorithms often work by arranging interference so that wrong answers tend to cancel and useful answers become more likely.

A third idea is entanglement.

Entanglement is a kind of quantum relationship between systems in which the whole system has a definite quantum description, but the parts cannot be fully described independently. Entanglement is not just ordinary correlation. For example, if you and I each take one glove from a pair, and later you discover you have the left glove, you immediately know I have the right glove. That is strong correlation, but it is classical. Entanglement is deeper: the measurement statistics of entangled systems cannot always be explained as if each part carried a pre-existing local instruction list. This distinction was clarified through Bell's theorem and later experiments, though we will wait until Chapter 8 to study the idea properly.

For now, the main point is this:

Quantum information is not just classical information hidden from us.

It follows different mathematical rules. Because of that, it can support different computational strategies.

David Deutsch proposed a model of a universal quantum computer in 1985, extending the idea of universal computation into the quantum setting (Deutsch, 1985). This helped turn Feynman's question into a theory of computation: not only "Can quantum systems simulate physics?" but also "What can computation become if the computer itself is quantum?"

1.6 Quantum computers do not simply try all answers at once

At this point, it is tempting to say:

“A quantum computer is powerful because it tries every possible answer at the same time.”

This sentence is common, but it is dangerously incomplete.

Quantum computation does involve superpositions over many possible classical states. However, when we measure a quantum system, we do not get all the amplitudes as a list. Measurement gives us an ordinary classical outcome, such as a bit string. If a quantum computer merely created a superposition of many possible answers and then measured it, the result would usually be no better than a random sample.

The real art of quantum algorithm design is not “try everything and read everything.”

The real art is:

Create amplitudes, transform them, make them interfere, and measure only after the useful outcomes have become likely.

Here is a simple analogy.

Imagine a dark room with many doors, and only one door leads outside. A bad strategy is to run randomly into a door. A slightly better classical strategy is to test doors one by one. A quantum strategy, when available, is not equivalent to opening all doors and seeing everything. It is more like arranging waves in the room so that the wave patterns cancel near bad doors and build up near the good door. When you finally look, you are more likely to find the exit.

This analogy is imperfect, but it captures the key point: quantum algorithms use interference to shape probabilities.

This is why quantum computing is subtle. Superposition alone is not enough. Entanglement alone is not enough. Measurement alone is not enough. Useful quantum computation requires a carefully designed sequence of operations that turns quantum structure into a measurable advantage.

1.7 What kinds of advantages are known?

Quantum computers are not known to be faster for every problem. They are not magic accelerators for all software. They offer speedups for particular kinds of problems with particular structure.

Two famous examples show the range of possibilities.

The first is Shor's algorithm. Peter Shor discovered a quantum algorithm that can factor large integers efficiently, in the sense that its running time grows polynomially with the number of digits of the input (Shor, 1994). Factoring means taking a number such as:

```
21
```

and writing it as a product:

```
21 = 3 × 7
```

For small numbers, factoring is easy. For very large numbers, no classical polynomial-time factoring algorithm is known. This matters because widely used public-key cryptographic systems, such as RSA, rely on the practical difficulty of factoring large integers. Shor's algorithm does not mean current quantum computers can already break all such systems; large-scale fault-tolerant quantum hardware would be needed. But it showed that quantum computation can change the landscape of what is efficiently computable.

The second example is Grover's algorithm. Lov Grover discovered a quantum algorithm for unstructured search that gives a quadratic speedup (Grover, 1996). Suppose there are N possible items and one is marked as the target. A classical search that has no extra structure may need on the order of N checks in the worst case. Grover's algorithm can find the marked item using on the order of √N quantum queries.

This is a real advantage, but it is not exponential. If N = 1,000,000, then:

```
N = 1,000,000
√N = 1,000
```

That is a major improvement. But if N doubles, the quantum cost still grows. Grover's algorithm teaches an important lesson: quantum speedup comes in different strengths. Sometimes it is exponential, sometimes polynomial, sometimes quadratic, and sometimes no useful speedup is known.

These examples also show why we must be careful. Quantum computing is powerful, but its power is specific.

A quantum computer probably will not make your word processor instantly better. It probably will not speed up every database task automatically. It will not replace classical computers. Instead, it will work alongside them for tasks where quantum structure gives an advantage.

1.8 The central question: what changes when information is quantum?

We can now state the guiding question of this book more precisely:

How does computation change when the basic unit of information is a quantum state rather than a classical bit?

Classical computation is built from bits, logic gates, circuits, algorithms, and complexity analysis.

Quantum computation keeps some of this language but changes the underlying mathematics.

A classical bit has a definite value:

```
0 or 1
```

A qubit has a quantum state, which can involve amplitudes for 0 and 1.

A classical gate, such as NOT, transforms bits:

```
0 → 1
1 → 0
```

A quantum gate transforms quantum states. Before measurement, these transformations must be reversible and must preserve the total probability structure of the state. Later, we will call such transformations unitary operations.

A classical algorithm is a sequence of operations designed to produce an answer.

A quantum algorithm is also a sequence of operations, but it must account for superposition, phase, interference, entanglement, and measurement.

Measurement is especially important. In classical computing, looking at a bit usually does not change it in any meaningful idealized sense. If a classical bit stores 1, reading it gives 1, and it remains 1.

In quantum computing, measurement is an operation with consequences. It produces a classical outcome, but it also changes the quantum state according to the rules of quantum mechanics. This is not a minor detail. It is one of the central constraints that makes quantum algorithm design different from ordinary parallel computation.

So quantum computing is not just “classical computing plus randomness.” Classical randomized algorithms already exist. A randomized algorithm uses random choices during computation. For example, a program might randomly choose test cases or randomly sample a large dataset.

Quantum algorithms also produce probabilities, but those probabilities come from amplitudes that can interfere. This makes quantum probability different from ordinary probability in a computationally useful way.

1.9 A small example of why interference matters

We can understand the flavor of interference without formal quantum mathematics.

Suppose there are two possible computational paths leading to an outcome A.

In ordinary probability, if one path contributes probability 0.25 and another path contributes probability 0.25, then the total probability is:

```
0.25 + 0.25 = 0.50
```

Probabilities add as nonnegative quantities.

Quantum theory works differently at the amplitude level. Probabilities come from amplitudes, and amplitudes can be positive, negative, or even complex numbers. We will study complex numbers later. For now, think of amplitudes as quantities that can carry direction.

Suppose two paths contribute amplitudes:

```
+1/2 and +1/2
```

They add to:

```
+1
```

This is constructive interference.

But if two paths contribute:

```
+1/2 and -1/2
```

they add to:

```
0
```

This is destructive interference.

After amplitudes combine, probabilities are calculated from the resulting amplitudes. So if the amplitude becomes zero, that outcome cannot be observed.

This is one of the deepest differences between classical and quantum information processing. A quantum algorithm can be designed so that many computational paths contribute amplitudes that cancel for unwanted answers and reinforce for wanted answers.

This does not happen automatically. Poorly designed quantum computations can produce useless random results. Good quantum algorithms are carefully engineered interference patterns.

1.10 Why quantum computing is difficult to build

If quantum computing is so promising, why do we not already have large quantum computers everywhere?

The short answer is that quantum systems are fragile.

A useful quantum computer must preserve delicate quantum states long enough to perform many operations. But real physical systems interact with their environment. Heat, electromagnetic noise, imperfect control signals, material defects, and measurement errors can disturb the computation.

One important term is decoherence. Decoherence is the process by which a quantum system loses the special phase relationships that allow quantum interference. When decoherence happens, the system begins to behave more like an ordinary classical mixture, and the quantum advantage can disappear.

Another important term is noise. Noise means unwanted disturbance in the computation. In classical computers, noise also exists, but classical digital information is relatively easy to stabilize. A voltage near one range can be treated as 0; a voltage near another range can be treated as 1. Small errors can often be corrected by restoring the signal.

Quantum information is harder to protect. We cannot simply inspect an unknown qubit repeatedly to see what state it is in, because measurement itself changes the state. We also cannot copy an arbitrary unknown quantum state perfectly; this is the content of the no-cloning theorem, which we will study later.

This is why quantum error correction is essential for large-scale quantum computing. The theory of quantum error correction shows that quantum information can be protected without directly copying unknown states, but doing so requires extra qubits, careful operations, and very low error rates (Nielsen & Chuang, 2010).

So there are two grand challenges:

1. The algorithmic challenge: find problems where quantum computation gives a real advantage.

2. The engineering challenge: build physical devices that can run those computations reliably.

Both challenges are active areas of research.

1.11 What quantum computing is not

It is useful to remove several misconceptions early.

Quantum computing is not magic. It is based on precise mathematical rules and physical experiments.

Quantum computing is not simply faster computing. A quantum computer may be worse than a classical computer for many ordinary tasks.

Quantum computing is not just parallel computing. Although a quantum state can involve amplitudes for many possibilities, measurement does not reveal all of them. The advantage comes from interference, not from reading out infinitely many parallel answers.

Quantum computing does not make classical computing obsolete. Classical computers are still needed to control quantum hardware, prepare inputs, process outputs, compile circuits, run optimization loops, and handle everyday computation.

Quantum computing does not eliminate computational difficulty. Many problems remain hard even for quantum computers, and for many tasks no quantum speedup is known.

A healthy view is this:

Quantum computers are special-purpose accelerators for certain kinds of problems whose structure matches quantum information processing.

This view is less dramatic than popular hype, but it is more powerful because it is true.

1.12 The learning path from here

This chapter has introduced the motivation. The rest of the book will build the machinery step by step.

First, we will review classical bits, logic, gates, circuits, and computational cost. This gives us the baseline. We need to know what classical computing is before we can clearly see what quantum computing changes.

Next, we will learn the mathematical language of quantum states: vectors, complex numbers, matrices, inner products, and basis representations. These are not decorative topics. They are the grammar of quantum computing.

Then we will study qubits, measurement, and quantum gates. We will learn how a qubit is represented, how probabilities arise from amplitudes, and how gates transform states.

After that, we will combine qubits using tensor products, study entanglement, and learn multi-qubit circuits. This is where quantum computing begins to show its distinctive structure.

Only then will we study algorithms such as Deutsch-Jozsa, phase estimation, Shor’s algorithm, Grover’s algorithm, and quantum simulation. The goal is not to memorize famous algorithms. The goal is to understand how they work from first principles.

By the end of the book, quantum computing should no longer feel like a mysterious phrase. It should feel like a precise model of computation: strange at first, but learnable.

1.13 Chapter checkpoint

Before moving on, make sure you can answer these questions in your own words:

1. Why is computation a physical process?
2. What is a bit?
3. What is an algorithm?

4. Why does exponential growth make some problems difficult?
5. Why does faster hardware not automatically solve exponential scaling?
6. Why are quantum systems hard to simulate on classical computers?
7. What is the basic idea of superposition?
8. Why is the phrase “tries all answers at once” misleading?
9. What role does interference play in quantum algorithms?
10. Why are quantum computers difficult to build?

If these questions feel challenging, that is normal. This chapter is a map, not the full journey. The coming chapters will turn each major idea into something you can calculate with.

The key idea to carry forward is simple:

Quantum computing exists because information is physical, and the physical world is quantum.

References

Deutsch, D. (1985). Quantum theory, the Church-Turing principle and the universal quantum computer. Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences, 400(1818), 97–117.

Feynman, R. P. (1982). Simulating physics with computers. International Journal of Theoretical Physics, 21, 467–488.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (pp. 212–219).

Landauer, R. (1961). Irreversibility and heat generation in the computing process. IBM Journal of Research and Development, 5(3), 183–191.

Moore, G. E. (1965). Cramming more components onto integrated circuits. Electronics, 38(8), 114–117.

Nielsen, M. A., & Chuang, I. L. (2010). Quantum Computation and Quantum Information (10th anniversary ed.). Cambridge University Press.

Shor, P. W. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings 35th Annual Symposium on Foundations of Computer Science (pp. 124–134).

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, s2-42(1), 230–265.

Document information

Chapter 1: Why Quantum Computing Exists

Project	Quantum Computing from First Principles
Document	Document 1.5
Author	mujirin
Verifier	Not verified
Downloaded	July 04, 2026 20:15 KST
Status	Working
Document link	https://theorytrace.com/projects/quantum-computing-from-first-principles/documents/chapter-1-why-quantum-computing-exists/