# Introduction to Parallel and Quantum Computing

**Wei Sun (孙玮, sunwei@ihep.ac.cn)**

**Computing Center, Institute of High Energy Physics**

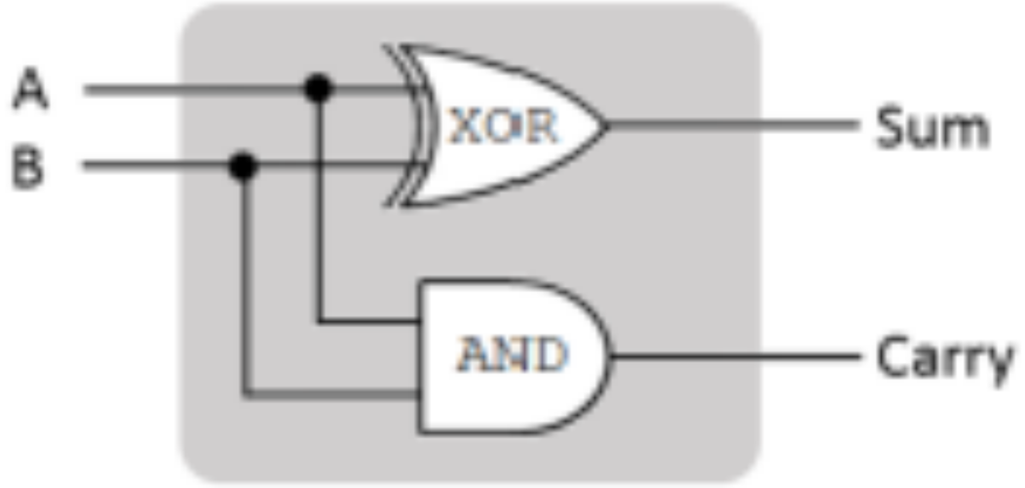
IHEP School of Computing 2024, 2024.8.21-23

Physics problem

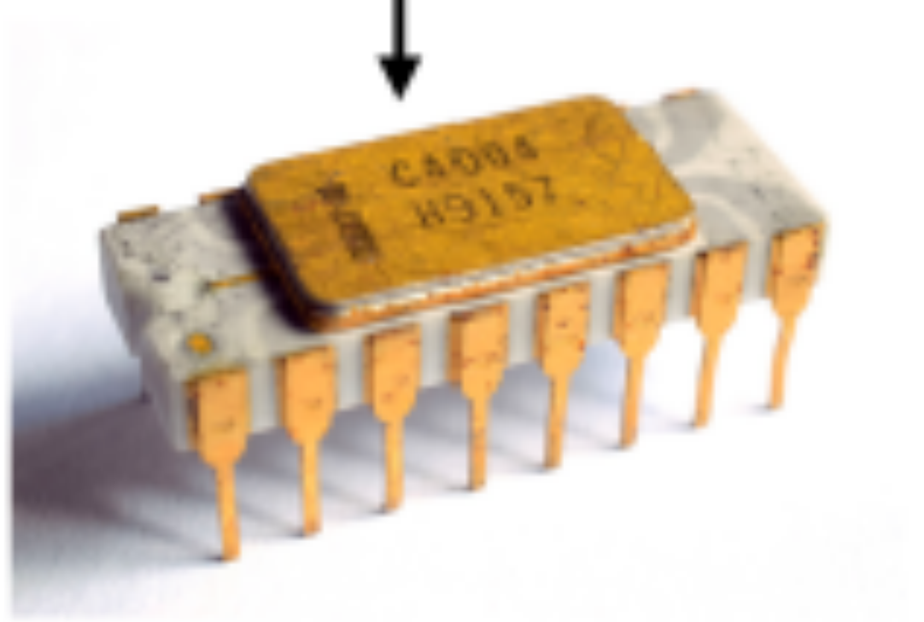
↓

Boolean algebra

↓

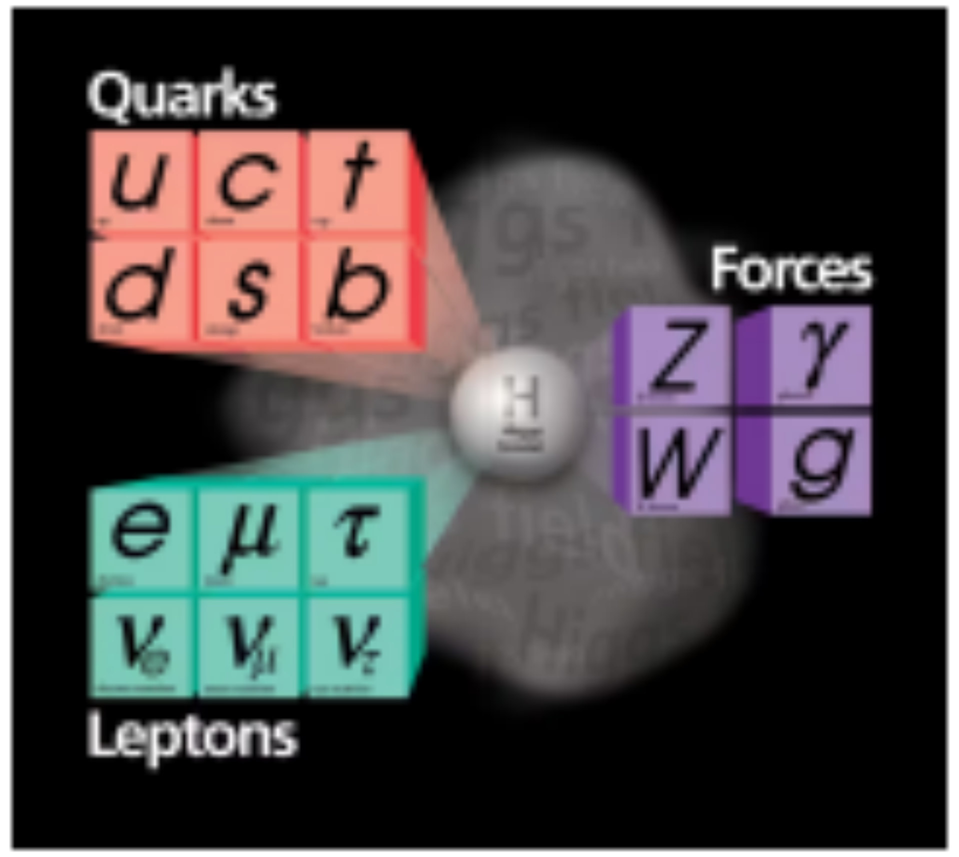



↓

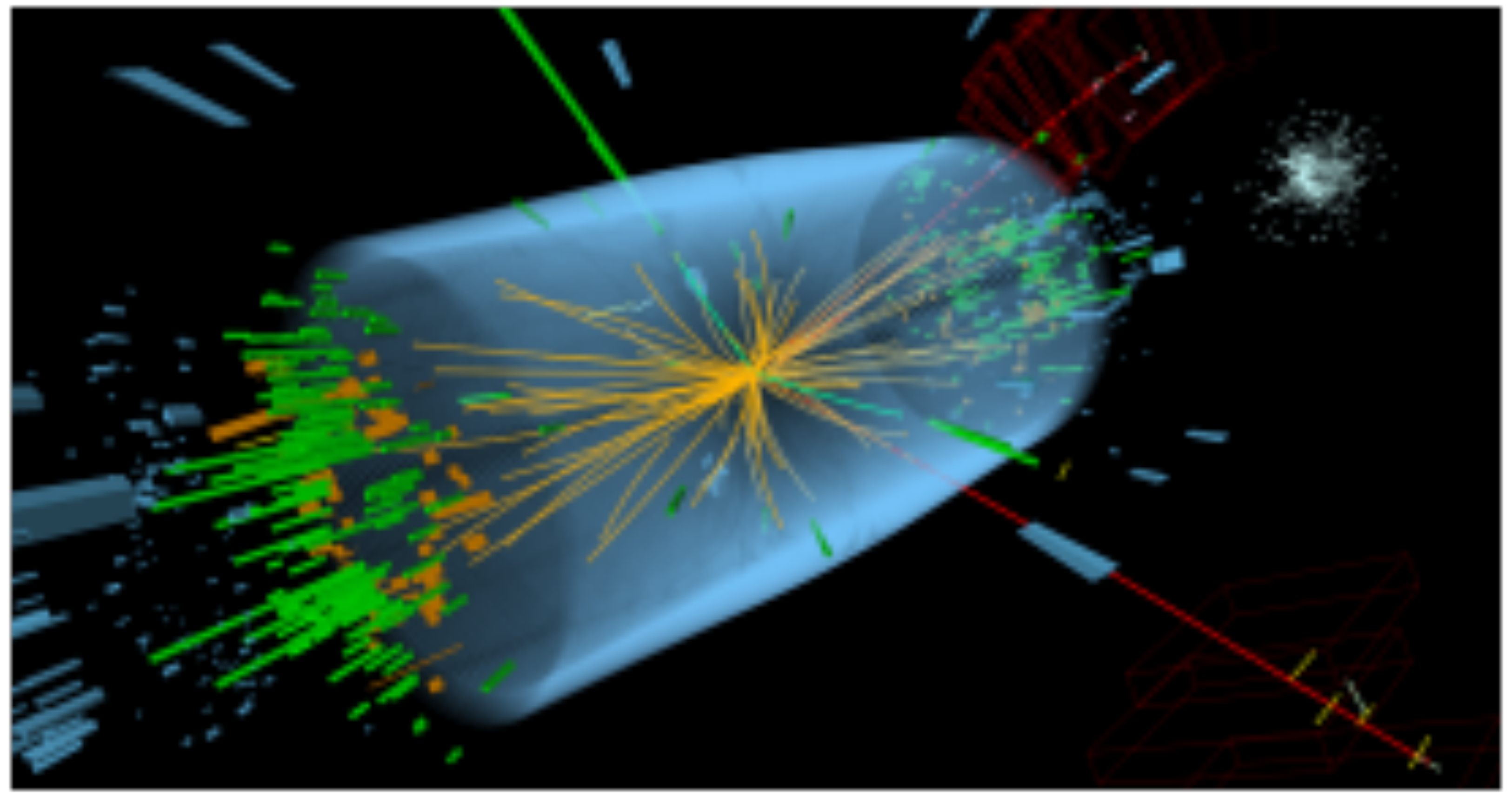


classical

Theory




Experiment

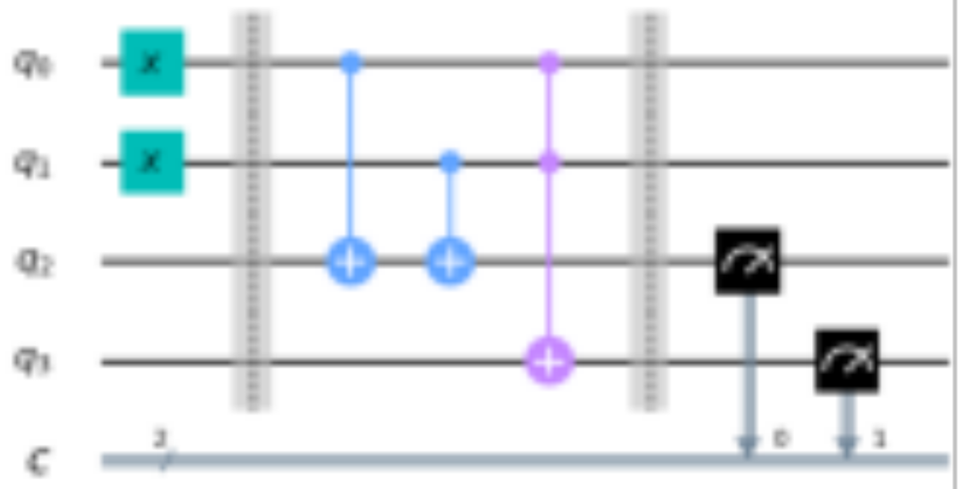


Physics problem

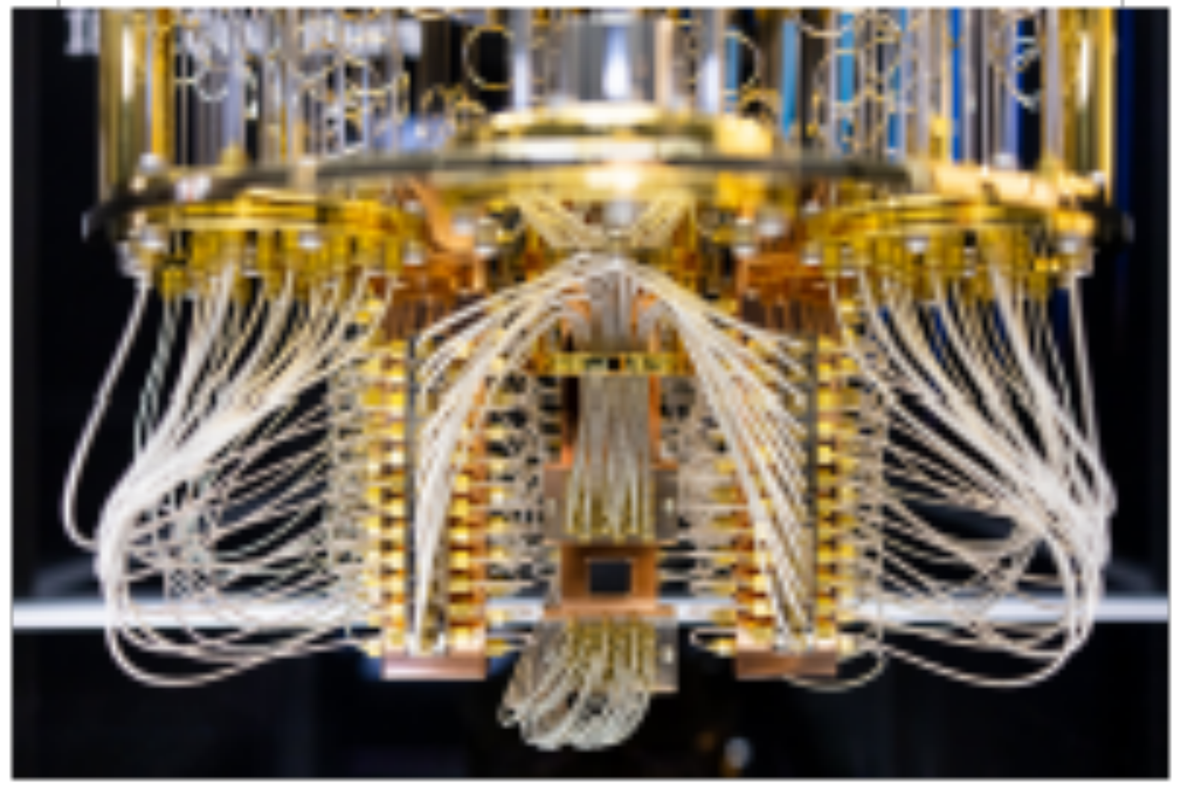
↓

Linear algebra

↓



↓

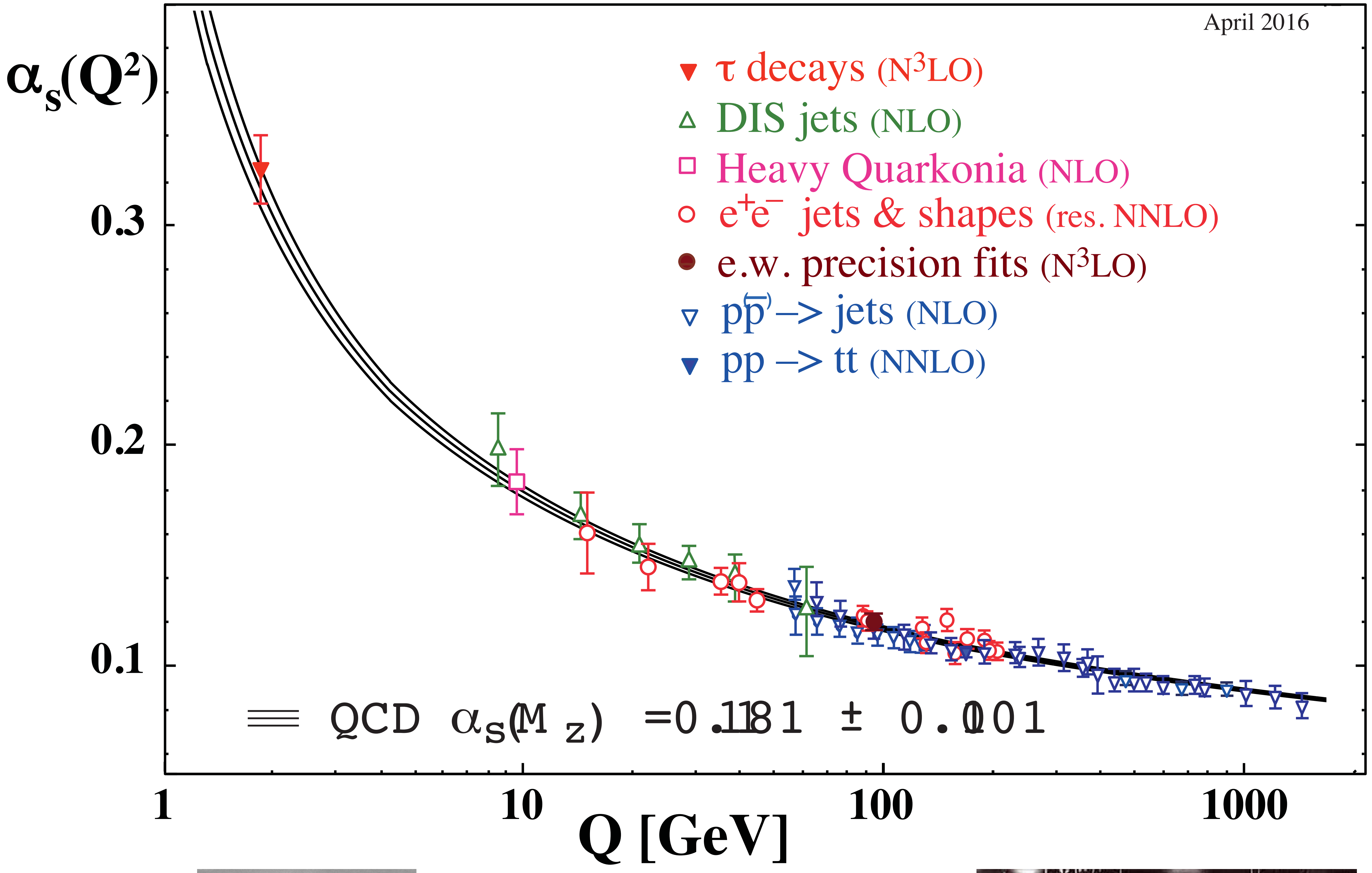


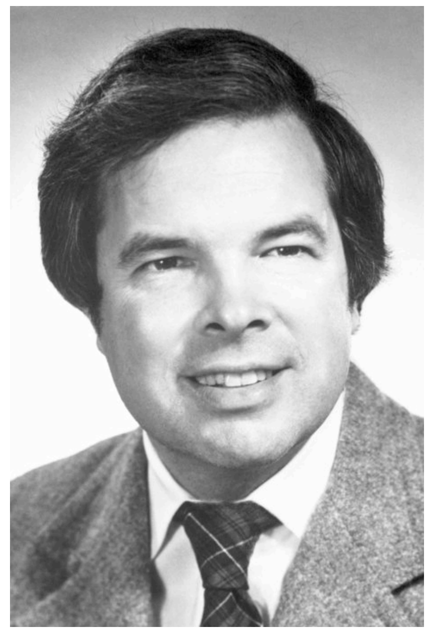
quantum

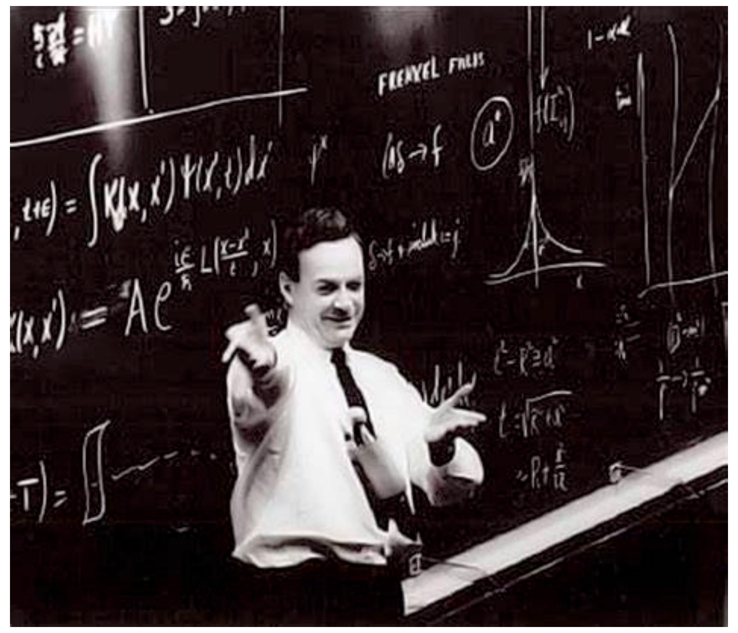
# Parallel Computing Outline

- Introduction
- High performance computers and supercomputers
- Parallel programming models
- Summary and further reading

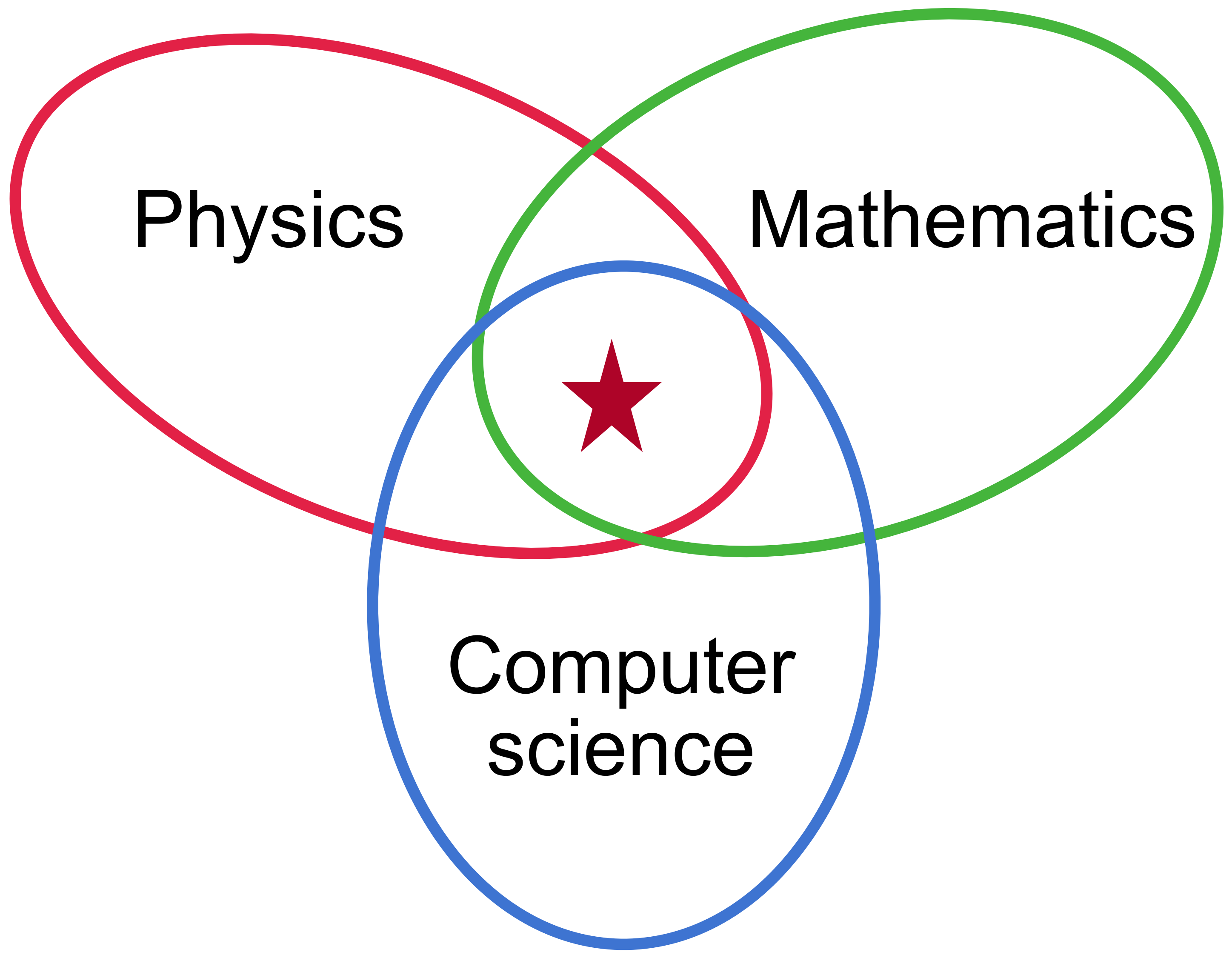
# Introduction





- analytical method at high energy
- numerical **Monte Carlo** method at low energy



# Introduction

| Computational Task | Current Usage | 2025 Usage | Current Storage (Disk) | 2025 Storage (Disk) | 2025 Network Requirements (WAN) |
|---|---|---|---|---|---|
| Accelerator Modeling | $\sim$ 10M – 100M core-hrs/yr | $\sim$ 10G – 100G core-hrs/yr | | | |
| Computational Cosmology | $\sim$ 100M – 1G core-hrs/yr | $\sim$ 100G – 1000G core-hrs/yr | ~10PB | >100PB | 300Gb/s (burst) |
| Lattice QCD | ~1G core-hrs/yr | $\sim$ 100G – 1000G core-hrs/yr | ~1PB | >10PB | |
| Theory | $\sim$ 1M – 10M core-hrs/yr | $\sim$ 100M – 1G core-hrs/yr | | | |
| Cosmic Frontier Experiments | $\sim$ 10M – 100M core-hrs/yr | $\sim$ 1G – 10G core-hrs/yr | ~1PB | 10 – 100PB | |
| Energy Frontier Experiments | $\sim$ 100M core-hrs/yr | $\sim$ 10G – 100G core-hrs/yr | ~1PB | >100PB | 300Gb/s |
| Intensity Frontier Experiments | $\sim$ 10M core-hrs/yr | $\sim$ 100M – 1G core-hrs/yr | ~1PB | 10 – 100PB | 300Gb/s |

ASCR/HEP Exascale Report [arXiv:1603.09303]

# Introduction

my definition

**Parallel (High Performance) Computing $\approx$ Numerical Linear Algebra on Supercomputers**

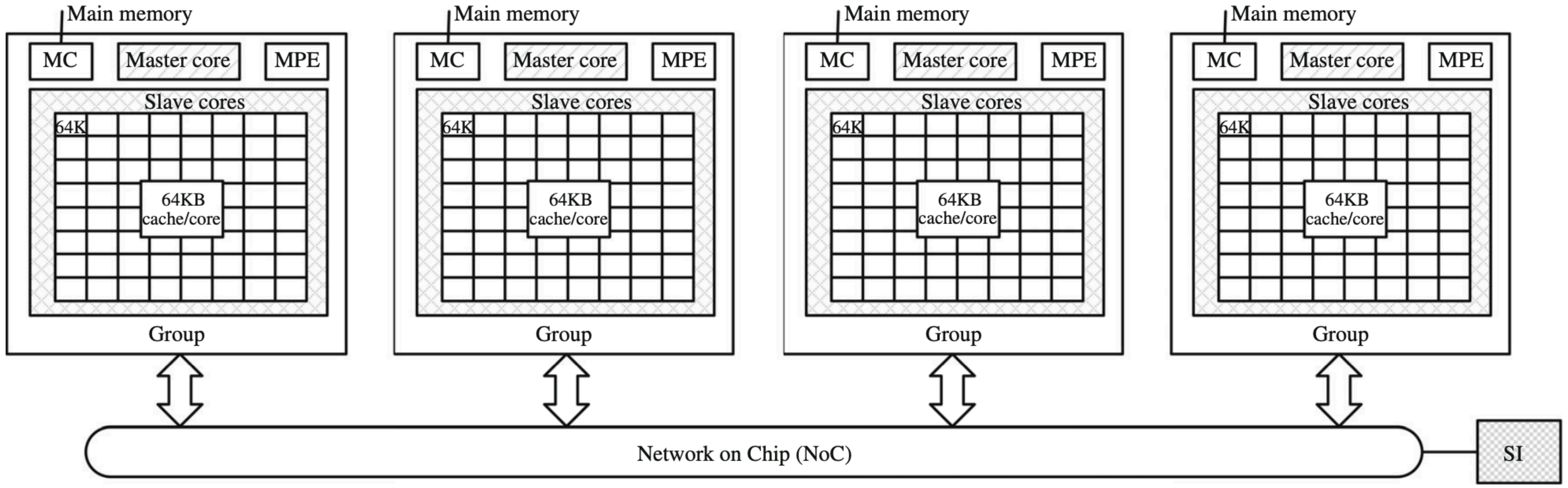
# High Performance Computers and Supercomputers

https://top500.org/lists/top500/2024/06/

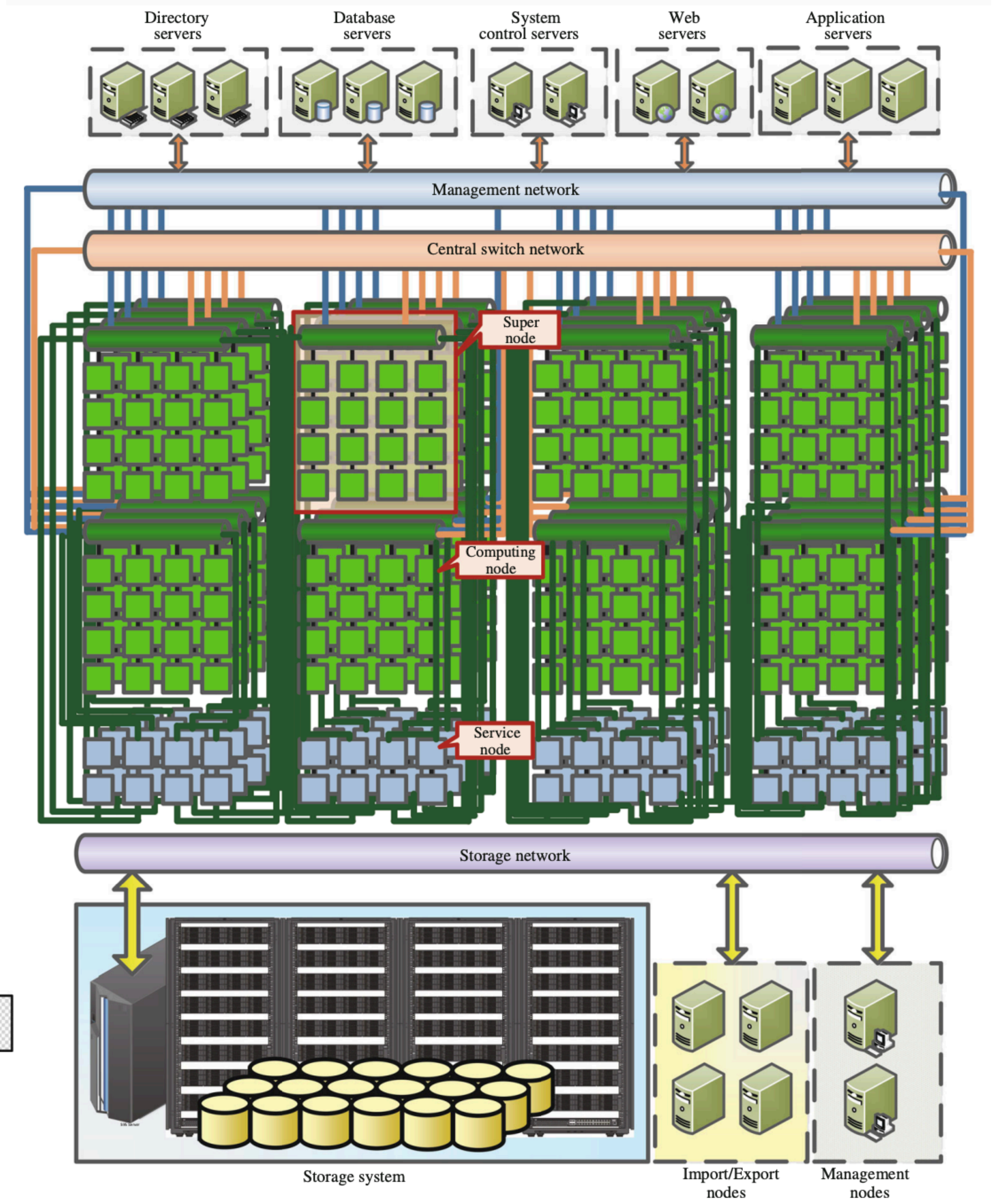
| Rank | System | Cores | Rmax (PFlop/s) | Rpeak (PFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 8,699,904 | 1,206.00 | 1,714.81 | 22,786 |
| 2 | **Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel<br>DOE/SC/Argonne National Laboratory<br>United States | 9,264,128 | 1,012.00 | 1,980.01 | 38,698 |
| 3 | **Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure<br>Microsoft Azure<br>United States | 2,073,600 | 561.20 | 846.84 | |
| 4 | **Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu<br>RIKEN Center for Computational Science<br>Japan | 7,630,848 | 442.01 | 537.21 | 29,899 |
| 5 | **LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>EuroHPC/CSC<br>Finland | 2,752,704 | 379.70 | 531.51 | 7,107 |
| 6 | **Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE<br>Swiss National Supercomputing Centre (CSCS)<br>Switzerland | 1,305,600 | 270.00 | 353.75 | 5,194 |
| 7 | **Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN<br>EuroHPC/CINECA<br>Italy | 1,824,768 | 241.20 | 306.31 | 7,494 |
| 8 | **MareNostrum 5 ACC** - BullSequana XH3000, Xeon Platinum 8460Y+ 32C 2.3GHz, NVIDIA H100 64GB, Infiniband NDR, EVIDEN<br>EuroHPC/BSC<br>Spain | 663,040 | 175.30 | 249.44 | 4,159 |
| 9 | **Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 2,414,592 | 148.60 | 200.79 | 10,096 |
| 10 | **Eos NVIDIA DGX SuperPOD** - NVIDIA DGX H100, Xeon Platinum 8480C 56C 3.8GHz, NVIDIA H100, Infiniband NDR400, Nvidia<br>NVIDIA Corporation<br>United States | 485,888 | 121.40 | 188.65 | |

# High Performance Computers and Supercomputers

Supercomputer example: Sunway TaihuLight




Fu, H., Liao, J., Yang, J. *et al.,* Sci. China Inf. Sci. **59,** 072001 (2016).



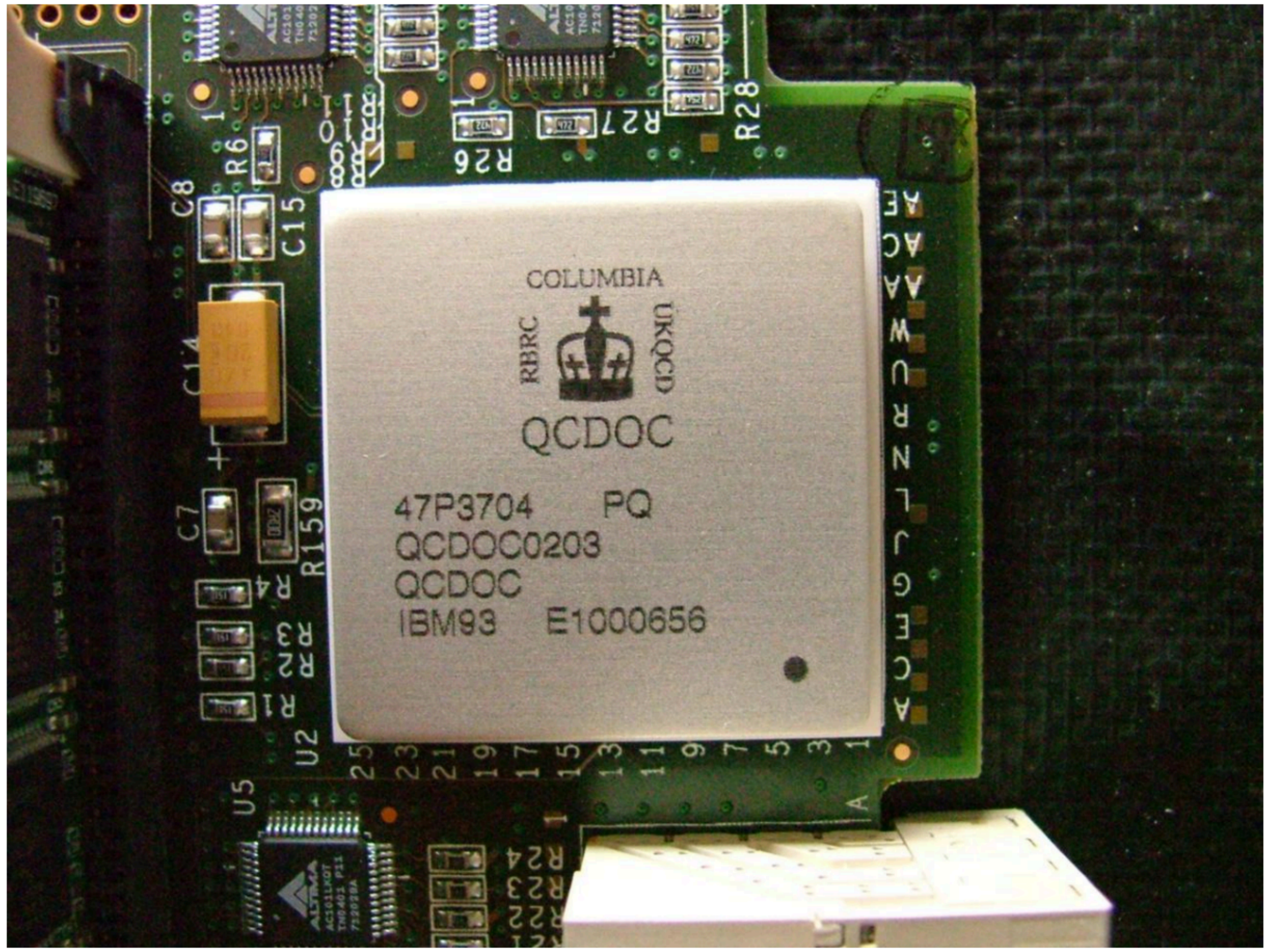
# High Performance Computers and Supercomputers

Example in HEP: LQCD with HPC

- Decades ago - customized processors
- QCDOC (QCD On a Chip)

- Nowadays - supercomputers / clusters
- TOP 500

QCDOC Asic, 1 Gflop/s




- LQCD awarded 1995,1998,2006 Goldon Bell Prize and 2018 finalist

Available online at www.sciencedirect.com
ScienceDirect
Computer Physics Communications 177 (2007) 631–639
Computer Physics Communications
www.elsevier.com/locate/cpc

# Lattice QCD as a video game

**Before CUDA release!**

Győző I. Egri[a], Zoltán Fodor[a,b,c,*], Christian Hoelbling[b], Sándor D. Katz[a,b], Dániel Nógrádi[b], Kálmán K. Szabó[b]

[a] *Institute for Theoretical Physics, Eötvös University, Budapest, Hungary*
[b] *Department of Physics, University of Wuppertal, Germany*
[c] *Department of Physics, University of California, San Diego, USA*

Received 2 February 2007; received in revised form 29 May 2007; accepted 7 June 2007

Available online 15 June 2007

# Parallel Programming Models

## Common programming language in HPC

- Fortran (Formula Translation)
  - Oldest high level programming language, first compiler released in 1957
  - Designed for numerical and scientific computing
  - Highly efficient, still widely used in high performance computing today
- C
  - Flexible, efficient, …
- C++
  - Efficient, abstract, multi-paradigm (procedural, object oriented, functional)
- Assembly
  - Highly efficient but not portable across different processor architecture
- Python
  - Slow in python itself, but with great library such as Scipy, very suitable for data processing, analysis and visualization

# Parallel Programming Models

## MPI + X model (**cluster level + node level + processor level + instruction level**)

- MPI (Message Passing Interface)
  - MPI is a communication protocol for programming parallel computers
  - The  dominant programming model in high performance computing today
  - Support point-to-point and collective communication
  - MPI version 1.0 standard released in 1994
  - Directly callable from C, C++, Fortran
  - Very suitable for **distributed memory system**, therefore supported by all kinds of supercomputers
- Major implementation
  - MPICH (https://www.mpich.org/)
  - Open MPI (https://www.open-mpi.org/)
  - Many others derived from MPICH and Open MPI, such as Intel MPI, Cray MPI, IBM Spectrum MPI

# Parallel Programming Models

## MPI Basics

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, rank, size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

- Compile: `mpicc hello_world.c -o hello_world`
- Run: `mpirun -np 4 hello_world`
- NOTE: MPI is a library and mpicc is not a compiler, it is a wrapper over regular C compiler
- Use `mpicc -show` to see the compile and link flags
- **`gcc -I /path to MPI/include -L /path to MPI/lib -lmpi`**

output

```
Hello world from processor ui03.hep.ustc.edu.cn, rank 1 out of 4 processors
Hello world from processor ui03.hep.ustc.edu.cn, rank 2 out of 4 processors
Hello world from processor ui03.hep.ustc.edu.cn, rank 3 out of 4 processors
Hello world from processor ui03.hep.ustc.edu.cn, rank 0 out of 4 processors
```

# Parallel Programming Models

## MPI Basics (point-to-point communication)

- Total 400+ APIs

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

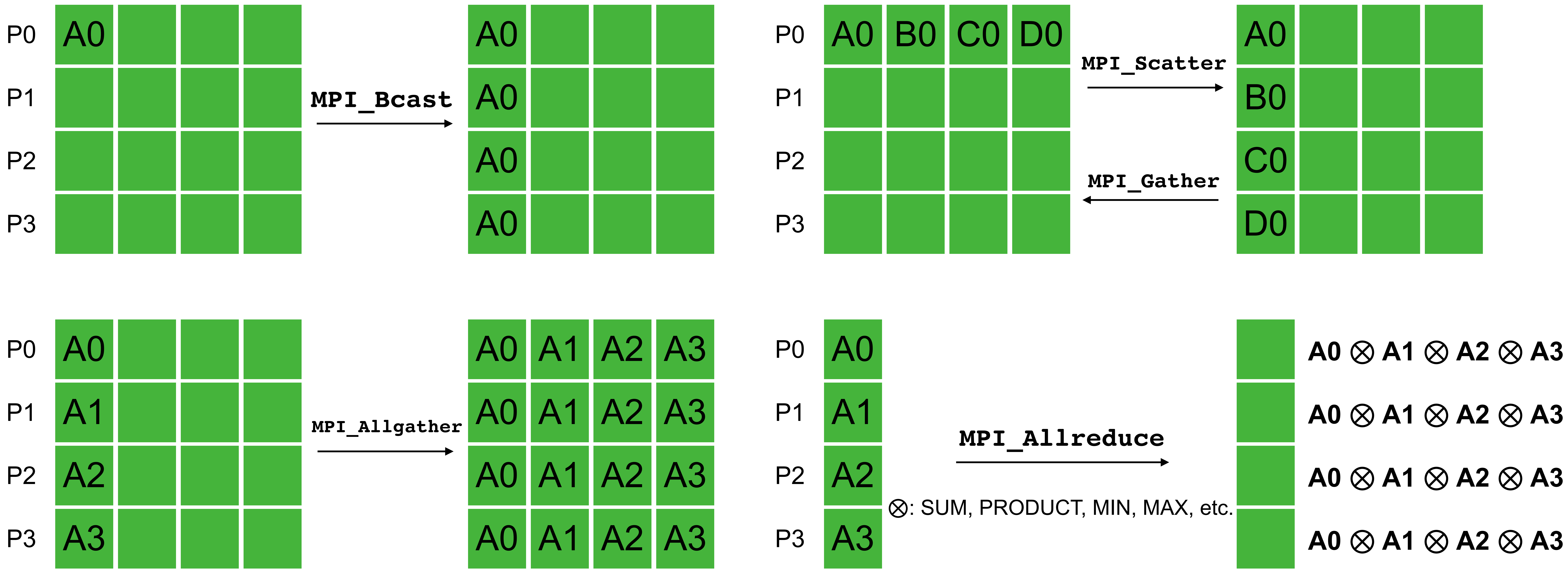
```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

```
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int number;
if (world_rank == 0) {
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (world_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n",
           number);
}
```

# Parallel Programming Models

## MPI Basics (collective communication)

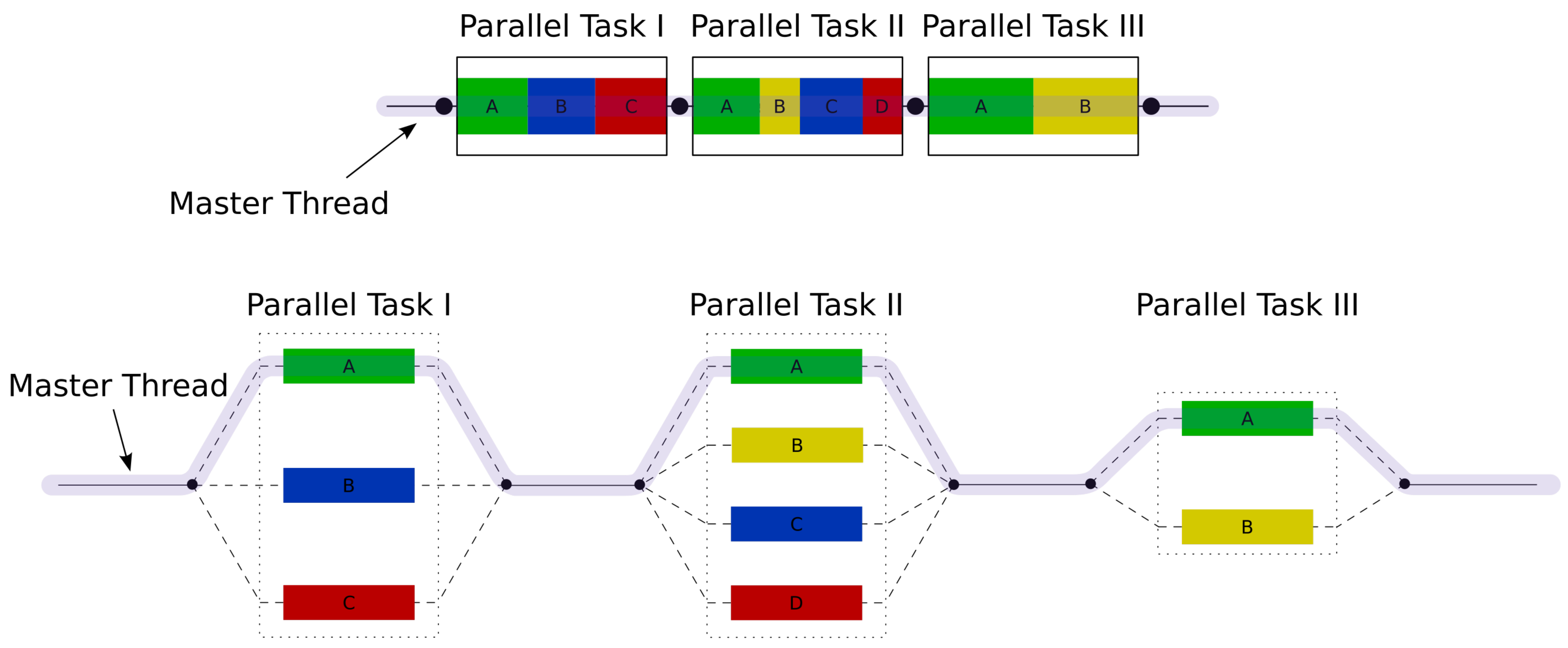
- Total 400+ APIs



# Parallel Programming Models

## OpenMP (Open Multi-Processing)

- Pros
  - API that supports various instruction set architectures, operating system, and C, C++, Fortran
  - First standard released in 1997
  - Compiler directive based
  - Simple, flexible, portable, scalable
  - Easy to modify existing serial code into parallel
  - OpenMP 4.0 and later version support GPUs
- Cons
  - Multi-threading programming is easy to implement but hard to debug in general
  - Need to deal with **race condition** very carefully
  - Only used for parallelism **within a node**
- Major implementation
  - GCC, Intel, Clang




source: wikipedia

# Parallel Programming Models

## OpenMP hello world example

```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv) {
#pragma omp parallel
  {
    int threads_total = omp_get_num_threads();
    int thread_id = omp_get_thread_num();
    printf("Hello, world from thread %d,"
           "out of %d threads.\n ",
           thread_id,
           threads_total);
  }
  return 0;
}
```

```c
#include <omp.h>
#include <math.h>

int main(int argc, char **argv) {
  const int N = 1000000;
  int a[N];

#pragma omp parallel for
  for (int i = 0; i < N; i++) {
    a[i] = sin(i);
  }

  return 0;
}

```

Compile: `gcc -fopenmp hello_world.c -o hello_world`

Run: `./hello_world` # use all cores / hardware threads available on single node

`OMP_NUM_THREADS=4 ./hello_world` # use 4 cores / hardware threads

# Parallel Programming Models

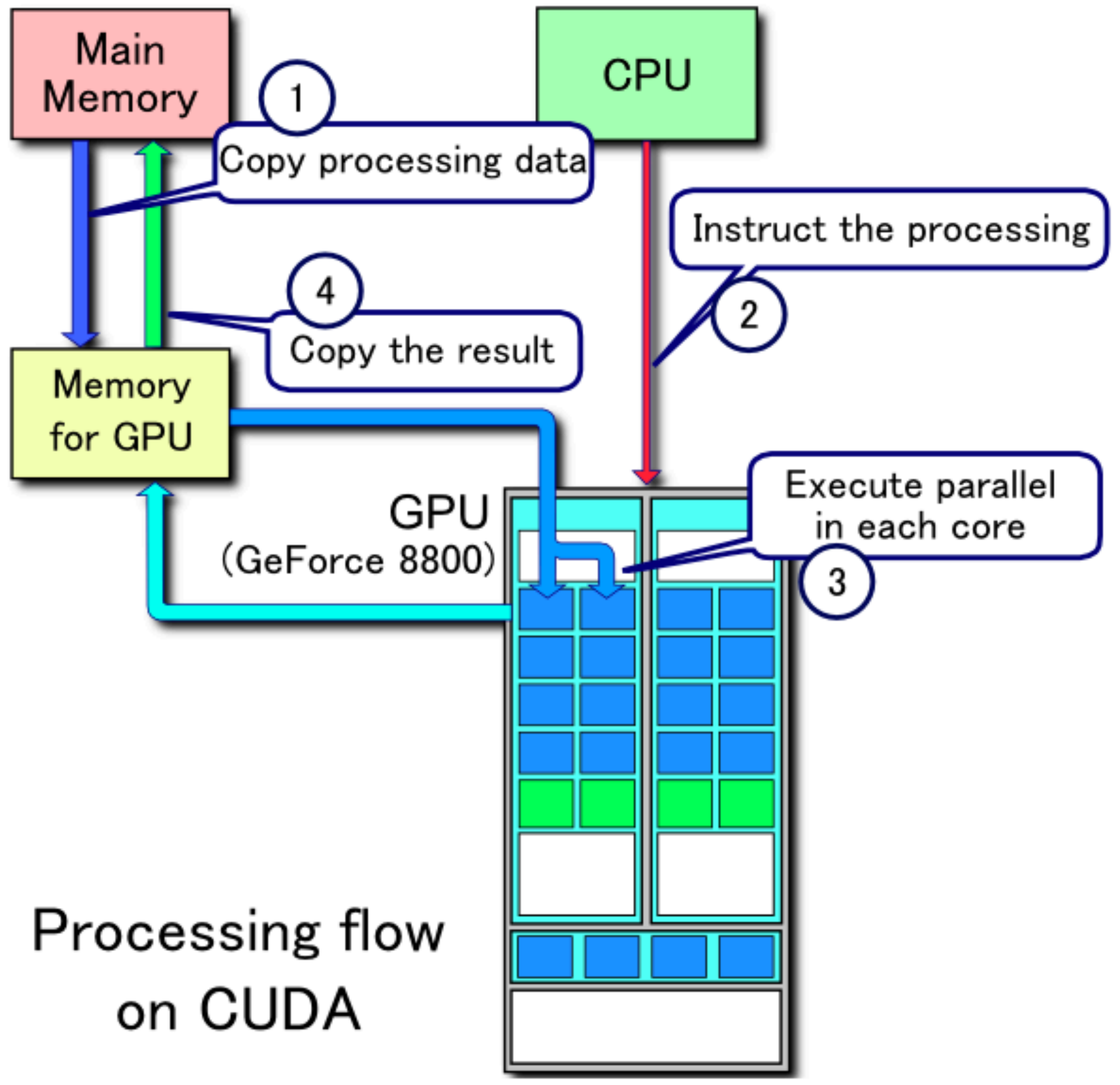
OpenMP program monitored with htop

```
  1  [||||||||||||100.0%]   13 [||||||||||||100.0%]   25 [||||||||||||100.0%]   37 [||||||||||||100.0%]
  2  [||||||||||||100.0%]   14 [||||||||||||100.0%]   26 [||||||||||||100.0%]   38 [||||||||||||100.0%]
  3  [||||||||||||100.0%]   15 [||||||||||||100.0%]   27 [||||||||||||100.0%]   39 [||||||||||||100.0%]
  4  [||||||||||||100.0%]   16 [||||||||||||100.0%]   28 [||||||||||||100.0%]   40 [||||||||||||100.0%]
  5  [||||||||||||100.0%]   17 [||||||||||||100.0%]   29 [||||||||||||100.0%]   41 [||||||||||||100.0%]
  6  [||||||||||||100.0%]   18 [||||||||||||100.0%]   30 [||||||||||||100.0%]   42 [||||||||||||100.0%]
  7  [||||||||||||100.0%]   19 [||||||||||||100.0%]   31 [||||||||||||100.0%]   43 [||||||||||||100.0%]
  8  [||||||||||||100.0%]   20 [||||||||||||100.0%]   32 [||||||||||||100.0%]   44 [||||||||||||100.0%]
  9  [||||||||||||100.0%]   21 [||||||||||||100.0%]   33 [||||||||||||100.0%]   45 [||||||||||||100.0%]
  10 [||||||||||||100.0%]   22 [||||||||||||100.0%]   34 [||||||||||||100.0%]   46 [||||||||||||100.0%]
  11 [||||||||||||100.0%]   23 [||||||||||||100.0%]   35 [||||||||||||100.0%]   47 [||||||||||||100.0%]
  12 [||||||||||||100.0%]   24 [||||||||||||100.0%]   36 [||||||||||||100.0%]   48 [||||||||||||100.0%]
  Mem[|||                              7.27G/503G]   Tasks: 57, 113 thr; 48 running
  Swp[                                  0K/8.00G]   Load average: 39.64 14.20 5.15
                                                    Uptime: 5 days, 16:49:45

CPU   PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 26 47947 sunwei     20   0  391M  2996   616 R 4786  0.0 1h23:25          └─ ./parallel_for_openmp
 24 47994 sunwei     20   0  391M  2996   616 R 100.  0.0 1:44.33             ├─ ./parallel_for_openmp
 33 47993 sunwei     20   0  391M  2996   616 R 100.  0.0 1:44.33             ├─ ./parallel_for_openmp
 23 47992 sunwei     20   0  391M  2996   616 R 100.  0.0 1:44.34             ├─ ./parallel_for_openmp
 25 47991 sunwei     20   0  391M  2996   616 R 100.  0.0 1:44.20             ├─ ./parallel_for_openmp
 22 47990 sunwei     20   0  391M  2996   616 R 99.4  0.0 1:44.33             ├─ ./parallel_for_openmp
 48 47989 sunwei     20   0  391M  2996   616 R 99.4  0.0 1:44.17             ├─ ./parallel_for_openmp
 21 47988 sunwei     20   0  391M  2996   616 R 100.  0.0 1:44.34             ├─ ./parallel_for_openmp
 47 47987 sunwei     20   0  391M  2996   616 R 100.  0.0 1:44.34             ├─ ./parallel_for_openmp
 46 47986 sunwei     20   0  391M  2996   616 R 98.7  0.0 1:44.30             ├─ ./parallel_for_openmp
 16 47985 sunwei     20   0  391M  2996   616 R 99.4  0.0 1:44.13             ├─ ./parallel_for_openmp
```

# Parallel Programming Models

## CUDA for GPU computing

- CUDA (Compute Unified Device Architecture)
  - CUDA is a parallel programming framework and API for general purpose GPU (GPGPU) computing
  - Developed by Nvidia and support Nvidia's GPUs
  - Supported Tesla -> Fermi -> Kepler -> Maxwell -> Pascal -> Volta -> Turing -> Ampere -> Hopper
  - Directly callable from C, C++, Fortran
  - Need CUDA Toolkit to compile
  - Free but not open source
  - Multi-node GPU programming with CUDA-aware MPI
  - The HIP ( Heterogeneous Interface for Portability) developed by AMD can is portable both for AMD and Nvidia's GPUs, and also free and open source

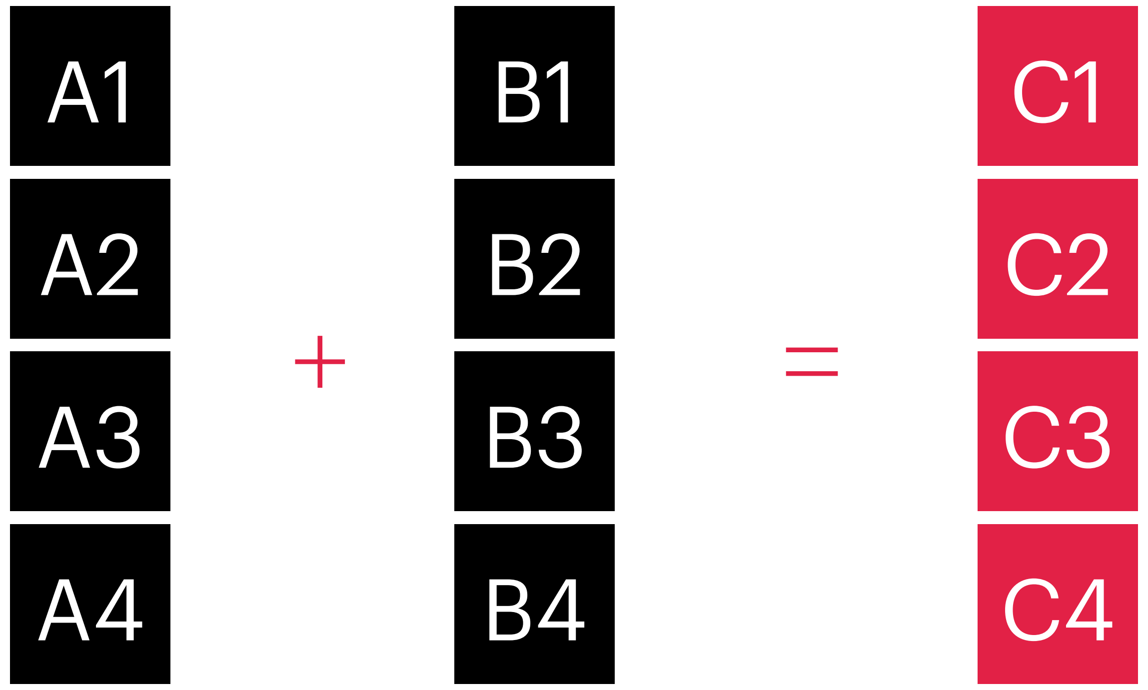



source: wikipedia

# Parallel Programming Models

SIMD (Single Instruction Multiple Data)

- Vectorization: supported by x86 (SSE, AVX, AVX2, AVX512 etc.), Arm (NEON, SVE), PowerPC (AltiVec) etc.
- Implementation: optimized math libraries (such as Intel MKL), inline assembly, intrinsic function




# Parallel Programming Models

## SIMD with intrinsic functions

**No explicit SIMD**

```c
void add(float* out, const float* input1, const float* input2, int N)
{
    for(int i=0; i<N; i++){
        out[i] = input1[i] + input2[i];
    }
}
```

**x86 AVX SIMD**

```c
#include<immintrin.h>
//compile: g++ -O3 -mavx -o exe src.c

void add_avx(float* out, const float* input1,
             const float* input2, int N)
{
    for(int i=0; i<N; i+=8){
        __m256 v1 = _mm256_load_ps(input1+i);
        __m256 v2 = _mm256_load_ps(input2+i);

        __m256 v0 = _mm256_add_ps(v1, v2);
        _mm256_store_ps(out+i, v0);
    }
}
```

**ARM NEON SIMD**

```c
#include<arm_neon.h>
//compile: g++ -O3 -march=armv8-a -o exe src.c

void add_neon(float* out, const float* input1,
             const float* input2, int N)
{
    for(int i=0; i<N; i+=4){
        float32x4_t v1 = vld1q_f32(input1+i);
        float32x4_t v2 = vld1q_f32(input2+i);

        float32x4_t v0 = vaddq_f32(v1, v2);
        vst1q_f32(out+i, v0);
    }
}
```

# Parallel Programming Models

## Software build tools

```
CC = gcc
CFLAGS = -O3 -fopenmp

objects = hello_world.o
all: hello_world

%.o : %.c
	$(CC) -c $(CFLAGS) $< -o $@

hello_world: $(objects)
	$(CC) $(CFLAGS) $^ -o $@

.PHONY: all
clean:
	rm -f *.o hello_world
```

Makefile

Build: **`make`**

```
acinclude.m4        lib
AUTHORS             LICENSE
autogen.sh          mainprogs
ChangeLog           Makefile.am
chroma-config.in    metadata.yml
config              NEWS
configure.ac        other_libs
COPYING             README
docs                scripts
INSTALL             tests
```

GNU Autotools

Build: **`autoreconf`**
**`./configure`**
**`make && make install`**

```
cmake               lib
CMakeLists.txt      LICENSE
doc                 NEWS
externals           README.md
include             tests
jenkins
```

CMake

Build: **`mkdir build && cd build`**
**`cmake ..`**
**`make && make install`**

# Summary and Further Reading

- Covered basics of high performance computing parallel programming models and tools widely used in high energy physics
- **Tips:**
  - **Select the right programming model and tools before writing the code**
  - **Correctness is the top priority, NOT performance at the beginning of the software development**
  - **Use well established, tested libraries, do NOT reinvent the wheels unless you know what you are doing**
  - **Use version control system such as git for code development, use github or gitlab for collaborative development**
- **Useful resources: there are plenty of lectures, tutorials, courses available online**
  - **Some very valuable links: https://www.nersc.gov (National Energy Research Scientific Computing Center)**
  - **https://www.olcf.ornl.gov/ (Oak Ridge National Laboratory Leadership Computing Facility)**

# Quantum Computing Outline

- What is a qubit
- How does quantum computers look like
- How to program a quantum computer
- Summary and further reading

# Simulating Physics with Computers

**Richard P. Feynman**

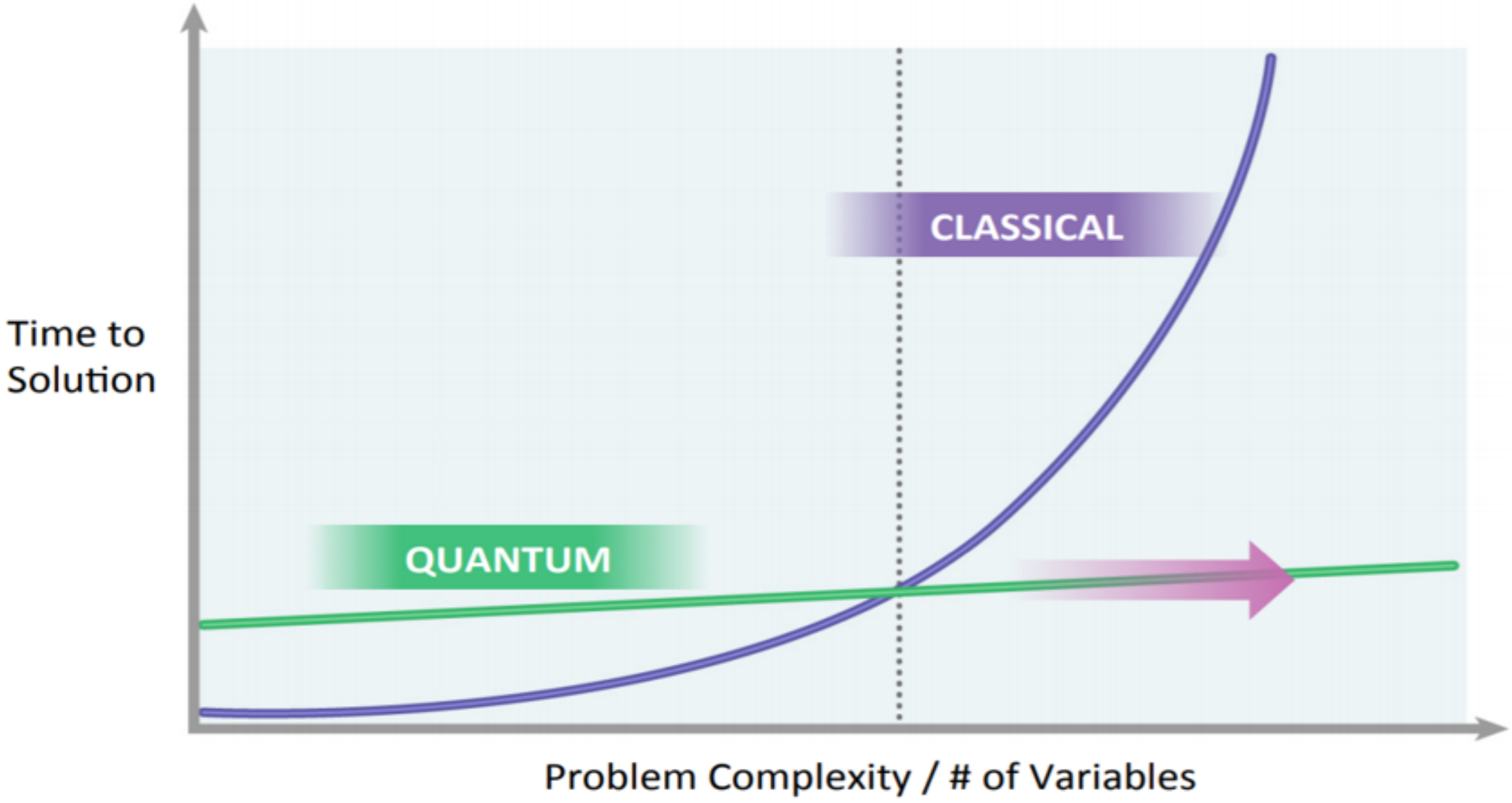
*Department of Physics, California Institute of Technology, Pasadena, California 91107*

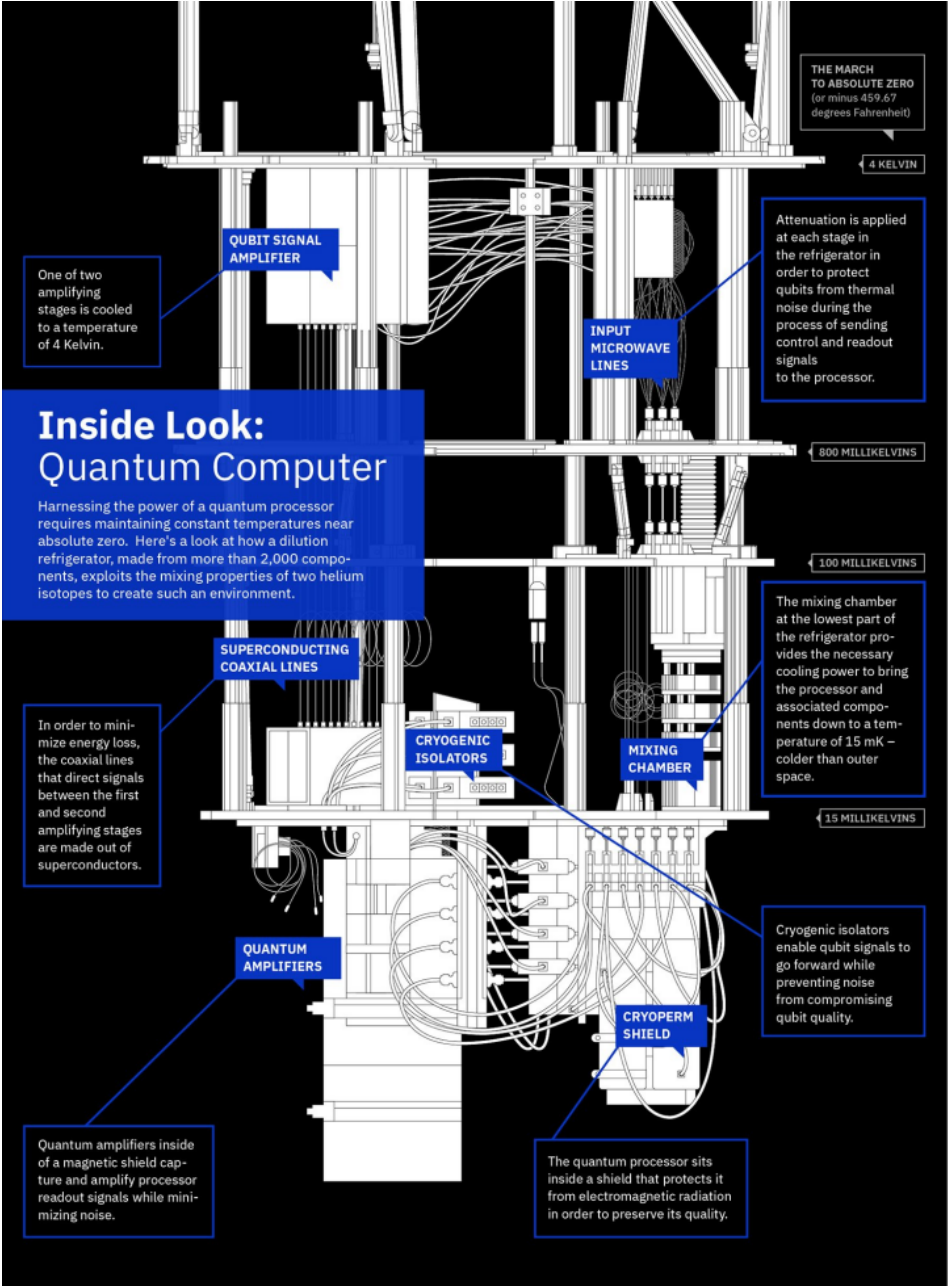
*Received May 7, 1981*

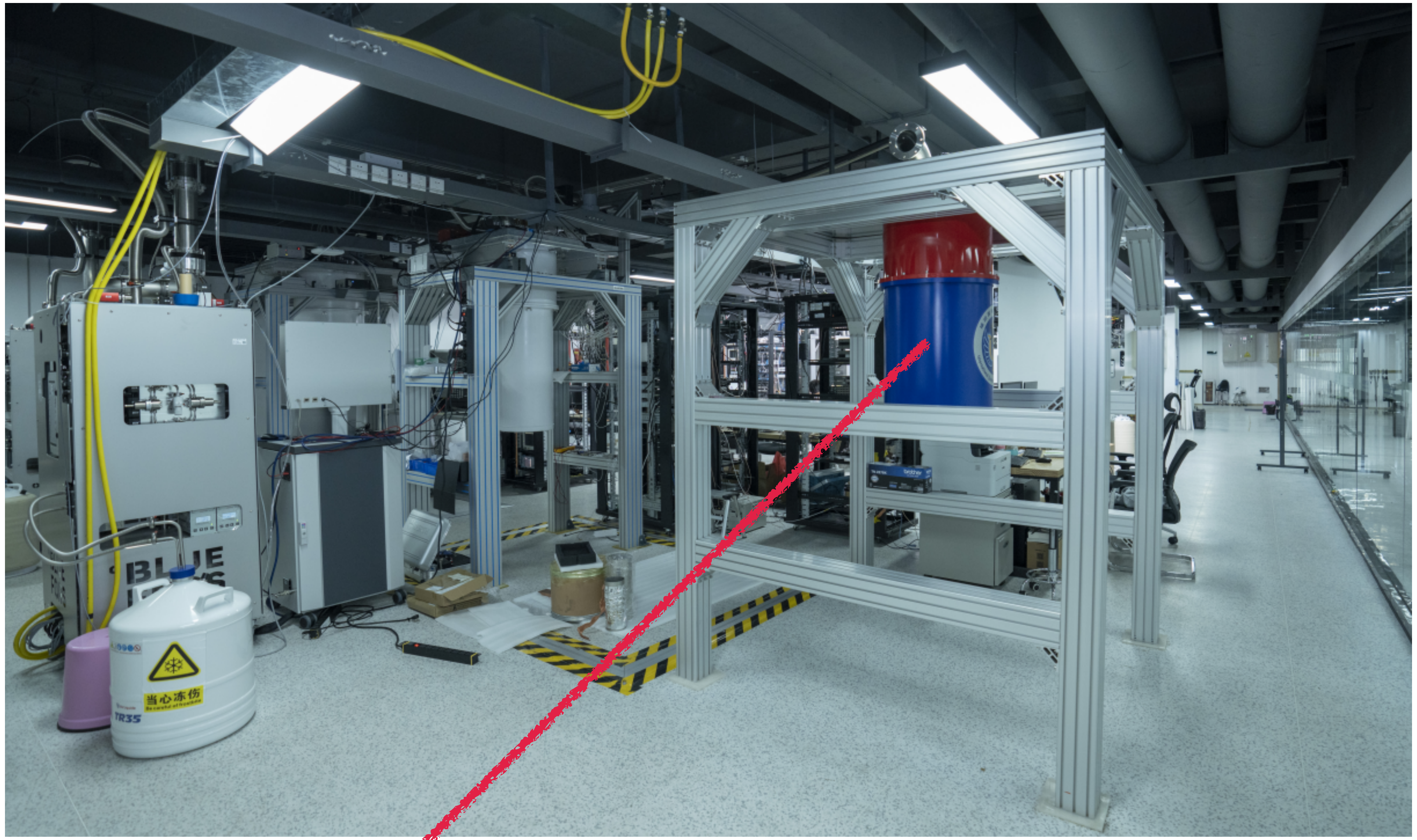
Nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical... Can you do it with a new kind of computer--a quantum computer? It's not a Turing machine, but a machine of a different kind.

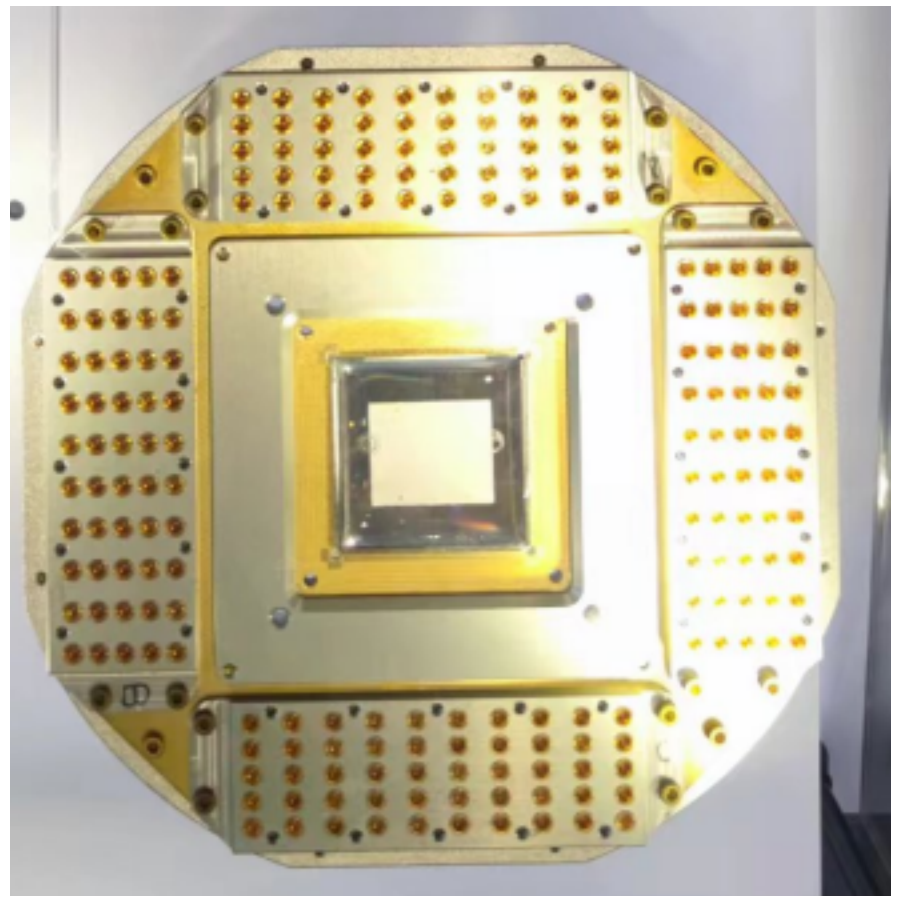
R. P. Feynman 1981

# Seeking for Quantum Advantage







- A quantum computer is a machine that performs computation based on quantum mechanics
- The data is represented by qubits, a two level system
- The operations on qubits are unitary quantum gates

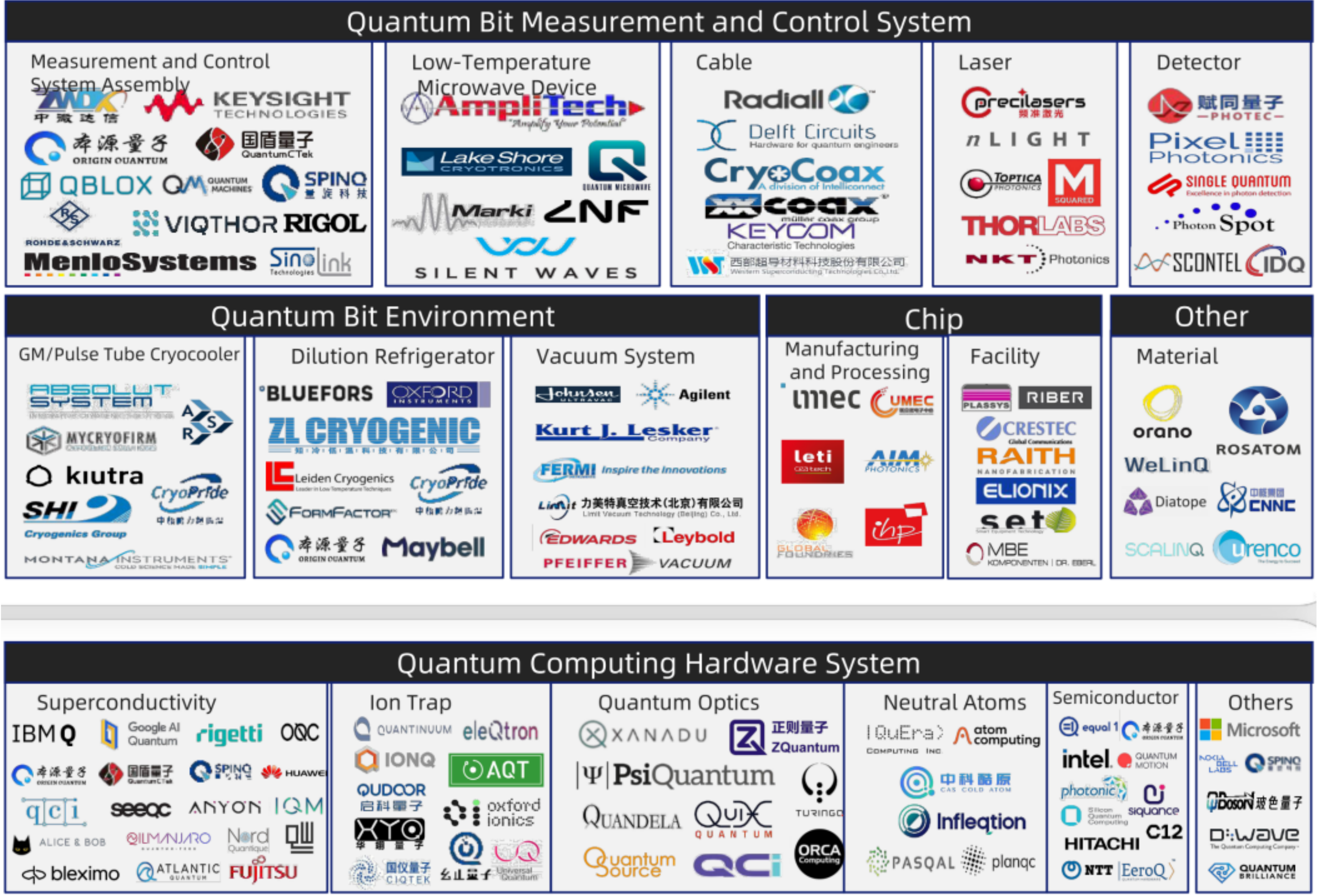
# DiVincenzo's Criteria

D. DiVincenzo, arXiv: quant-ph/0002077

1. A **scalable** physical system with well characterized qubits
2. The ability to **initialize** the state of the qubits to a simple fiducial state, such as $|000...000\rangle$
3. **Long relevant decoherence times**, much longer than the gate operation time
4. A **universal** set of quantum gates
5. A qubit-specific **measurement** capability

# Brief History of Quantum Computing

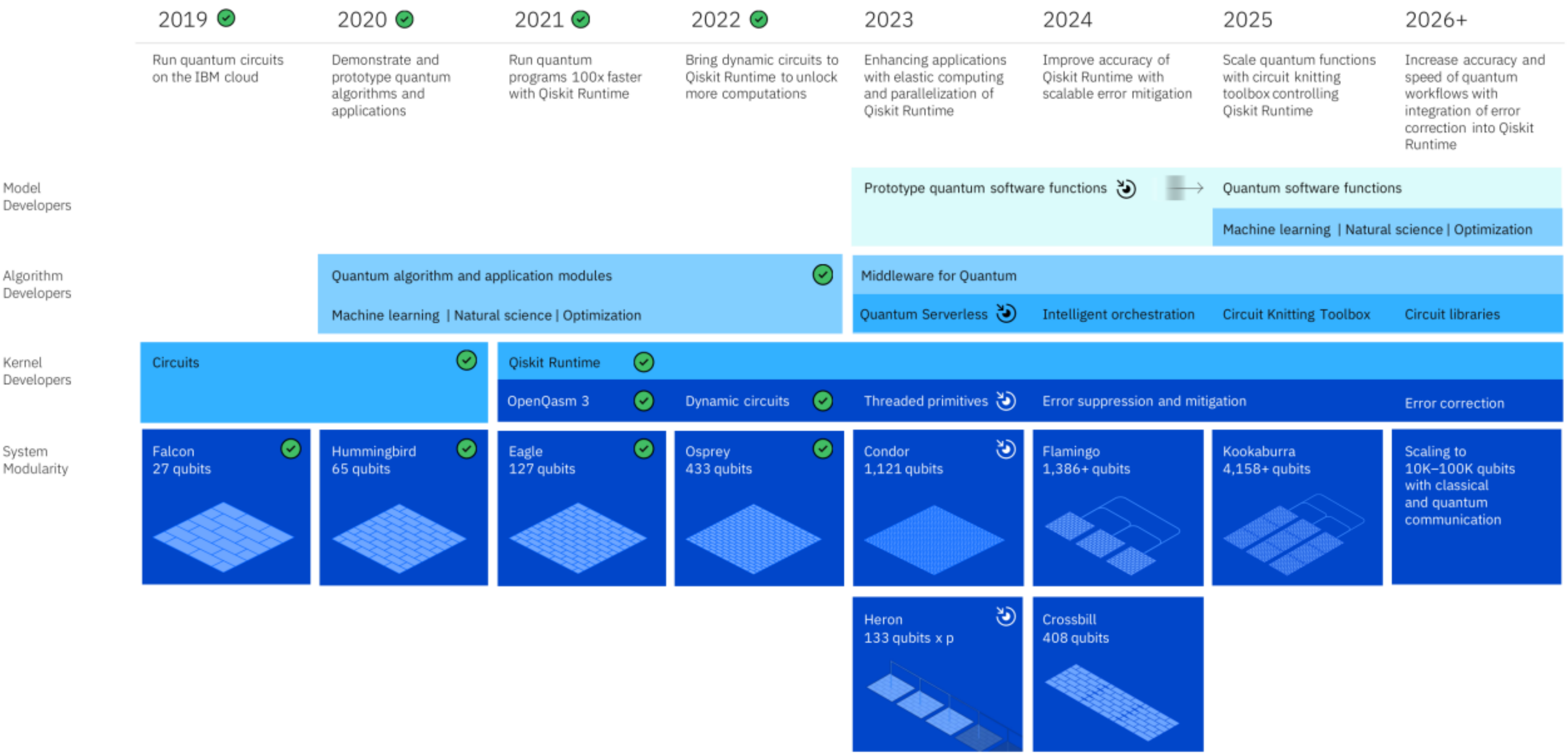

Credit: Quantumpedia



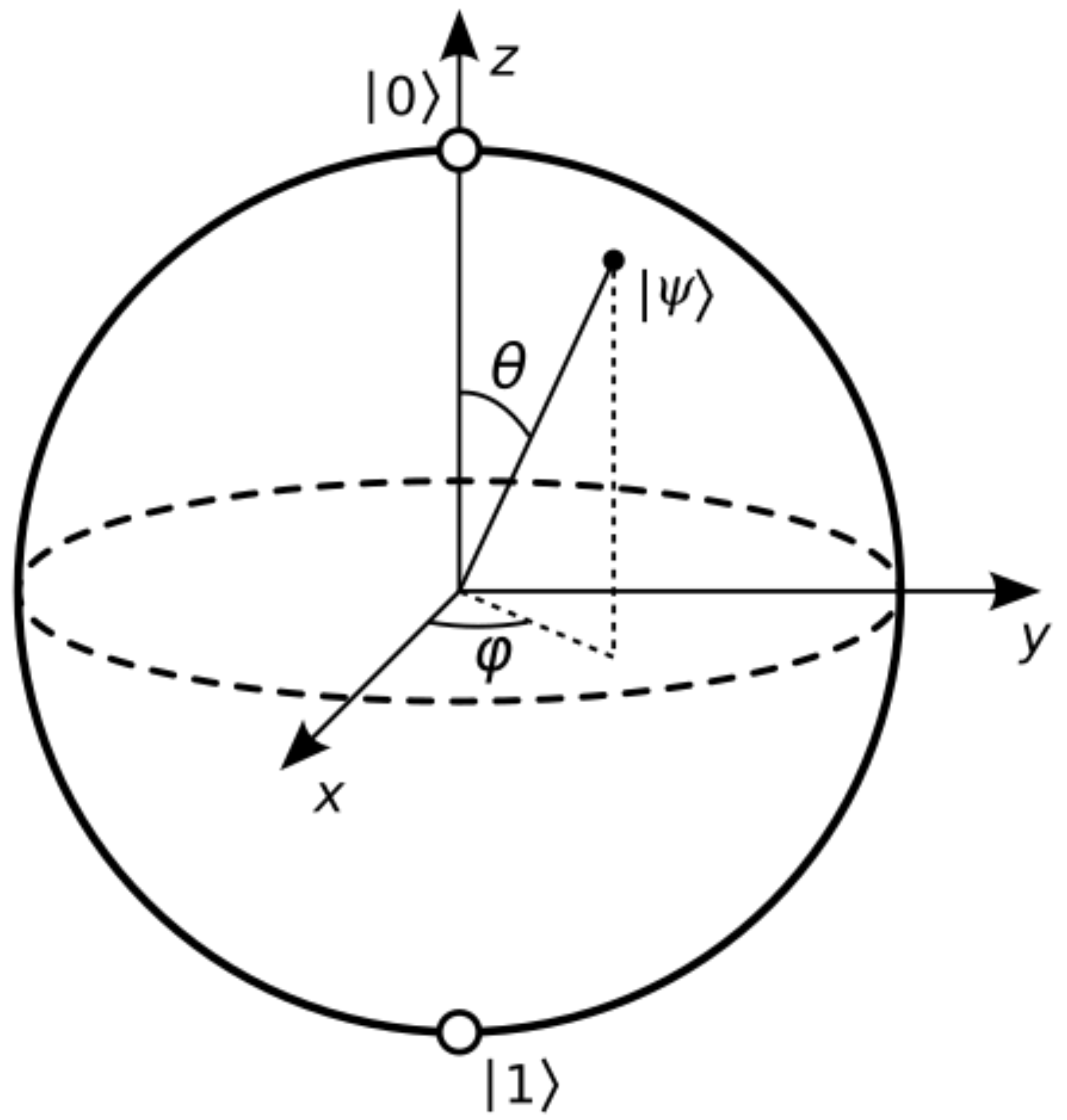

ICV TA&K and QUANTUMCHINA

# Development Roadmap

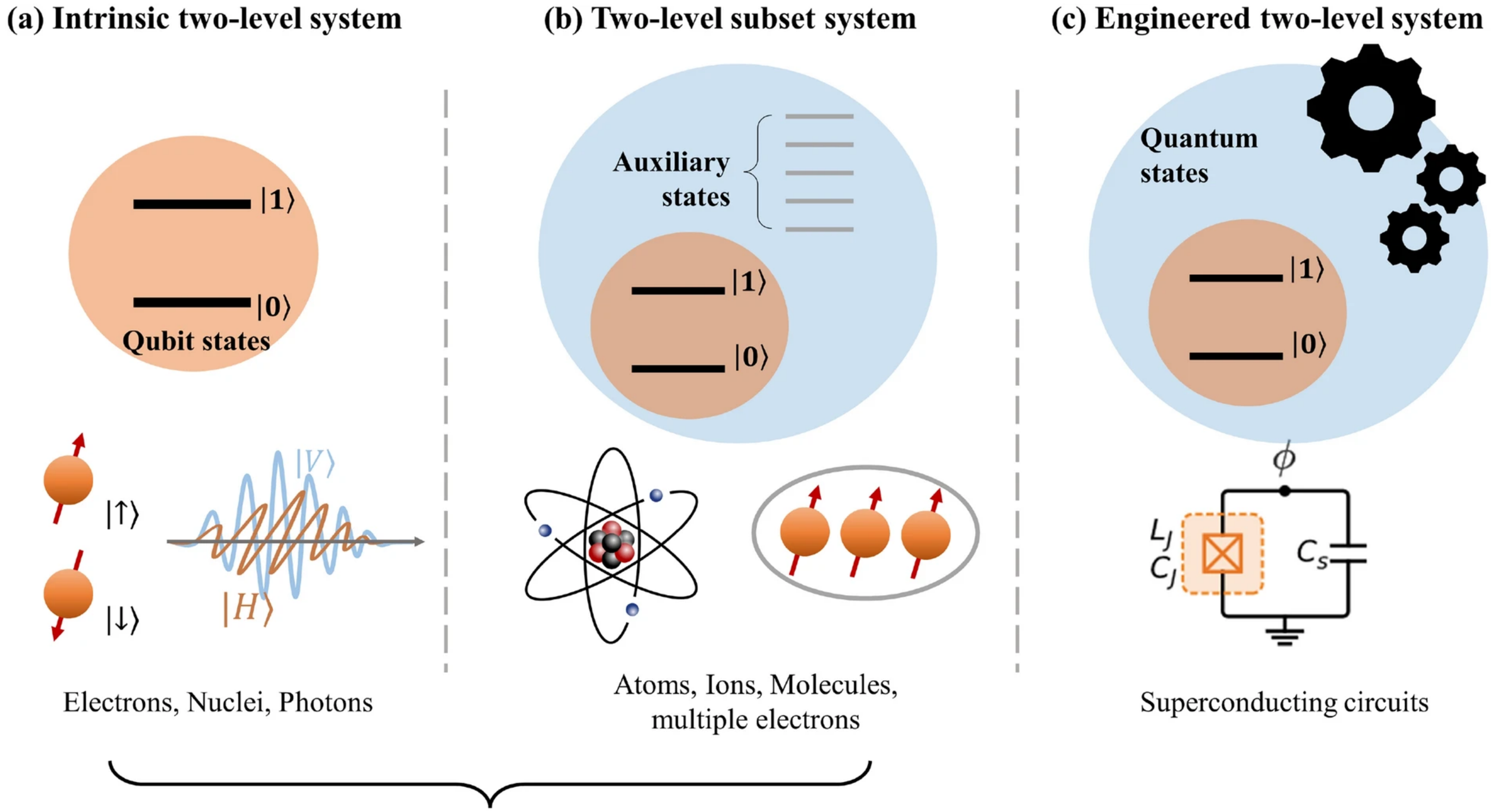
Executed by IBM ✓
On target



# What is a Qubit




- A qubit is a quantum state of a two-level quantum system
- Orthonormal basis states denoted as $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- A general qubit can be represented by a linear superposition of basis states, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \alpha, \beta \in C, |\alpha|^2 + |\beta|^2 = 1$




**Natural Qubits** [*Nano Convergence* **11**, 11 (2024)] **Synthetic Qubits**

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \qquad |\Phi^+\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

$$|\Psi^+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \qquad |\Psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

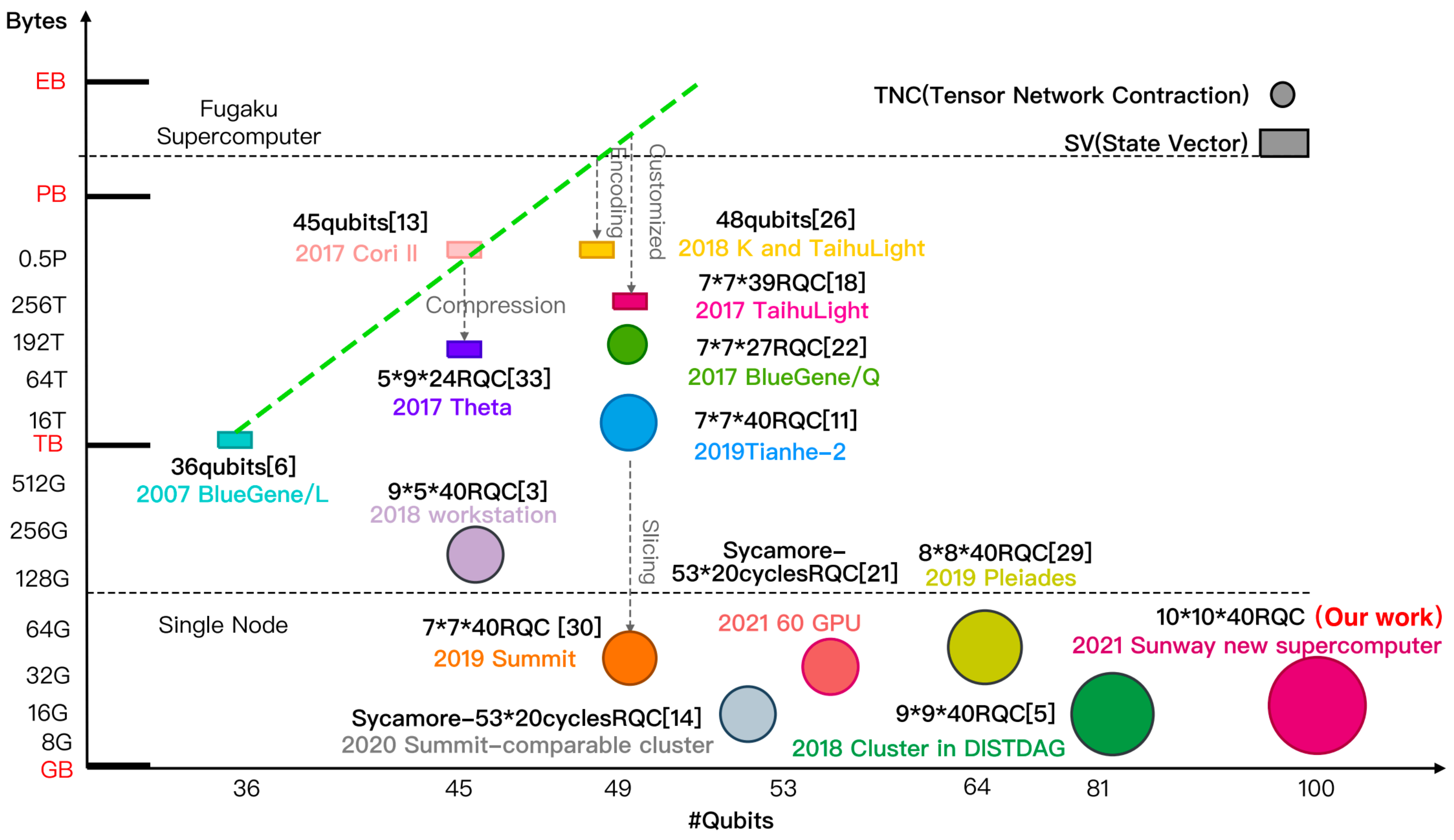
Bell states

# Quantum Gates

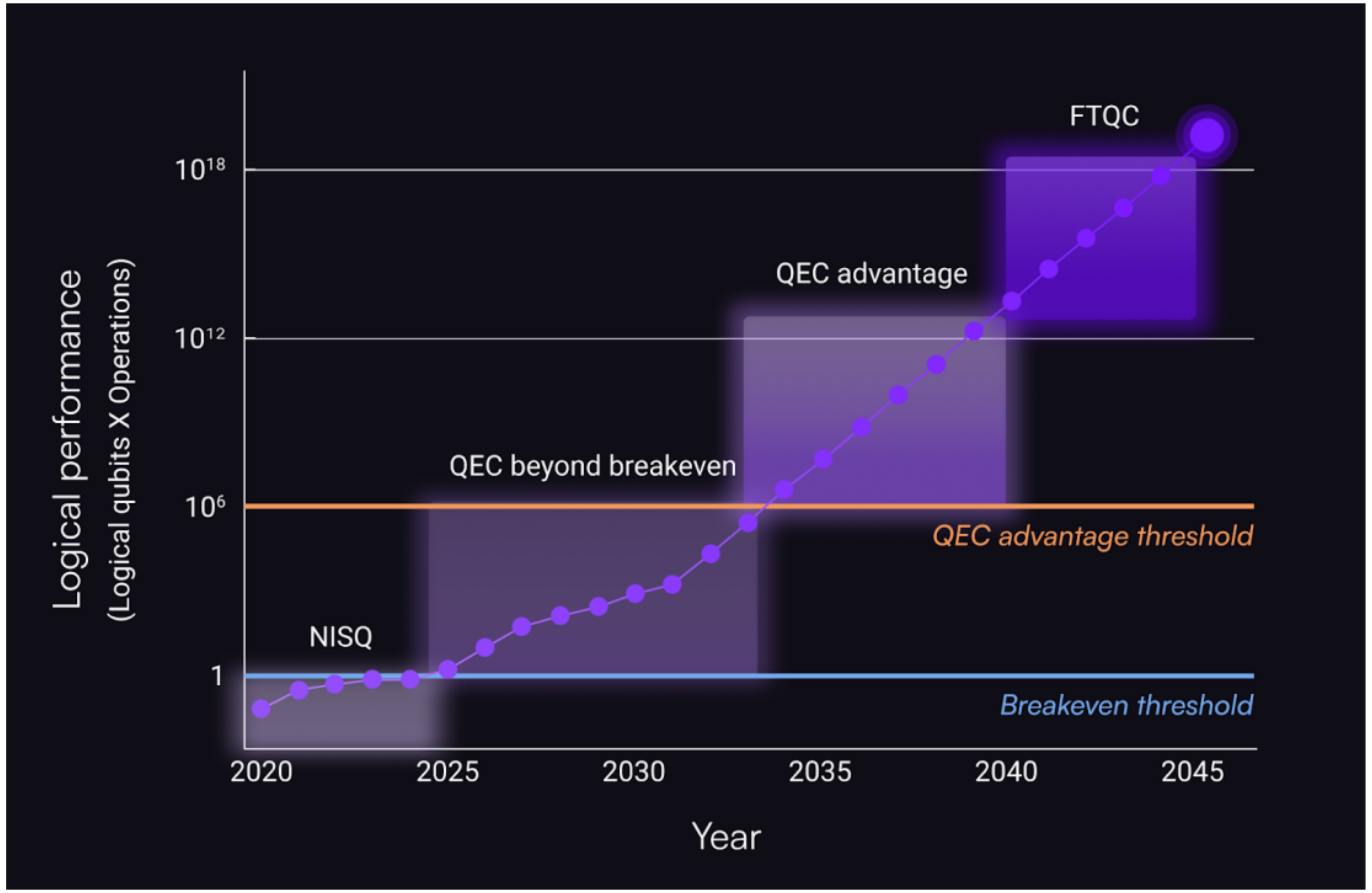
- Quantum gates are represented by unitary matrix and operated on qubits
- **Single qubit gates**: X, Y, Z, H, P, T, …
- **Two qubit gates**: CNOT, CZ, …
- Universal quantum gate sets: **approximate any unitary gate by any precision**
- Choose one of the possible universal gates set (Solovay-Kitaev theorem)
  - **{CNOT, H, T}**
  - **{CNOT, all single qubit gates}**
  - **{Toffoli, H}**
- $X|0\rangle = |1\rangle, |+\rangle = H|0\rangle$
- $CNOT|01\rangle = |01\rangle, CNOT|11\rangle = |10\rangle$

| Operator | Gate(s) | Matrix |
|---|---|---|
| Pauli-X (X) | X | $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ |
| Pauli-Y (Y) | Y | $\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ |
| Pauli-Z (Z) | Z | $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ |
| Hadamard (H) | H | $\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ |
| Phase (S, P) | S | $\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$ |
| $\pi/8$ (T) | T | $\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$ |
| Controlled Not (CNOT, CX) | | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ |
| Controlled Z (CZ) | Z | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$ |
| SWAP | | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |
| Toffoli (CCNOT, CCX, TOFF) | | $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ |

# Simulation of quantum computer on classical computer needs exponential resource




Y. Liu et.al. SC' 21




Current quantum computers are in the NISQ (Noisy Intermediate-Scale Quantum) era

- **Real hardwares are very noisy**
- **Error mitigation / correction is essential**
- **Need classical simulator to verify the quantum algorithms, while need $O(2^N)$ memory to simulate the quantum circuits**

# Quantum Programming Softwares

- Many high quality quantum computing softwares available
- Curated list of open-source quantum software projects
  - Most based on Python interfaced with C++
  - https://github.com/qosf/awesome-quantum-software
- Drag and drop playing with quantum circuits (https://qc.ihep.ac.cn)
- If you want to try the high performance GPU simulator, please contact me

# Quantum Algorithms

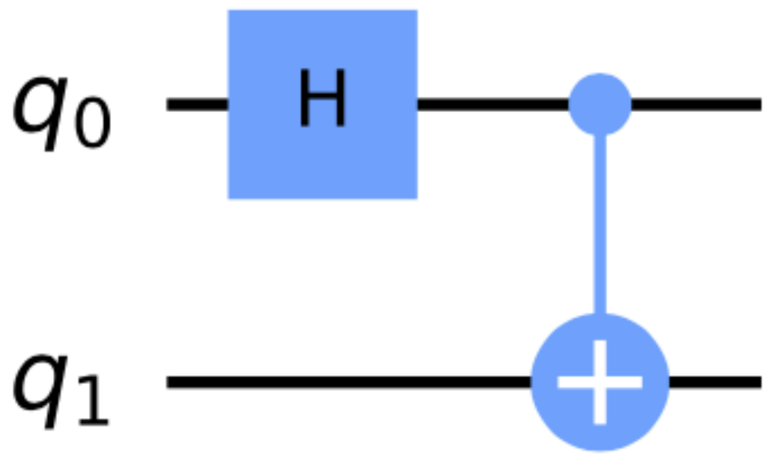
**Quantum Algorithm Zoo**

https://quantumalgorithmzoo.org/

- Compare the time complexity
- Try to implement the algorithms with popular qiskit package
  - pip install qiskit, play with jupyter notebook

| Algorithms | Classical steps | quantum logic steps |
|---|---|---|
| Fourier transform<br>e.g.:<br>- Shor's prime factorization<br>- discrete logarithm problem<br>- Deutsch Jozsa algorithm | $N\log(N) = n2^n$<br>$N = 2^n$<br>- n qubits<br>- N numbers | $\log^2(N) = n^2$<br>- hidden information!<br>- Wave function collapse prevents us from directly accessing the information |
| Search Algorithms | $N$ | $\sqrt{N}$ |
| Quantum Simulation | $c^N$ bits | kn qubits |

```python
from qiskit import QuantumCircuit

# Create a new circuit with two qubits
qc = QuantumCircuit(2)

# Add a Hadamard gate to qubit 0
qc.h(0)

# Perform a CNOT gate on qubit 1, controlled by qubit 0
qc.cx(0, 1)

# draw the circuit
qc.draw("mpl")
```

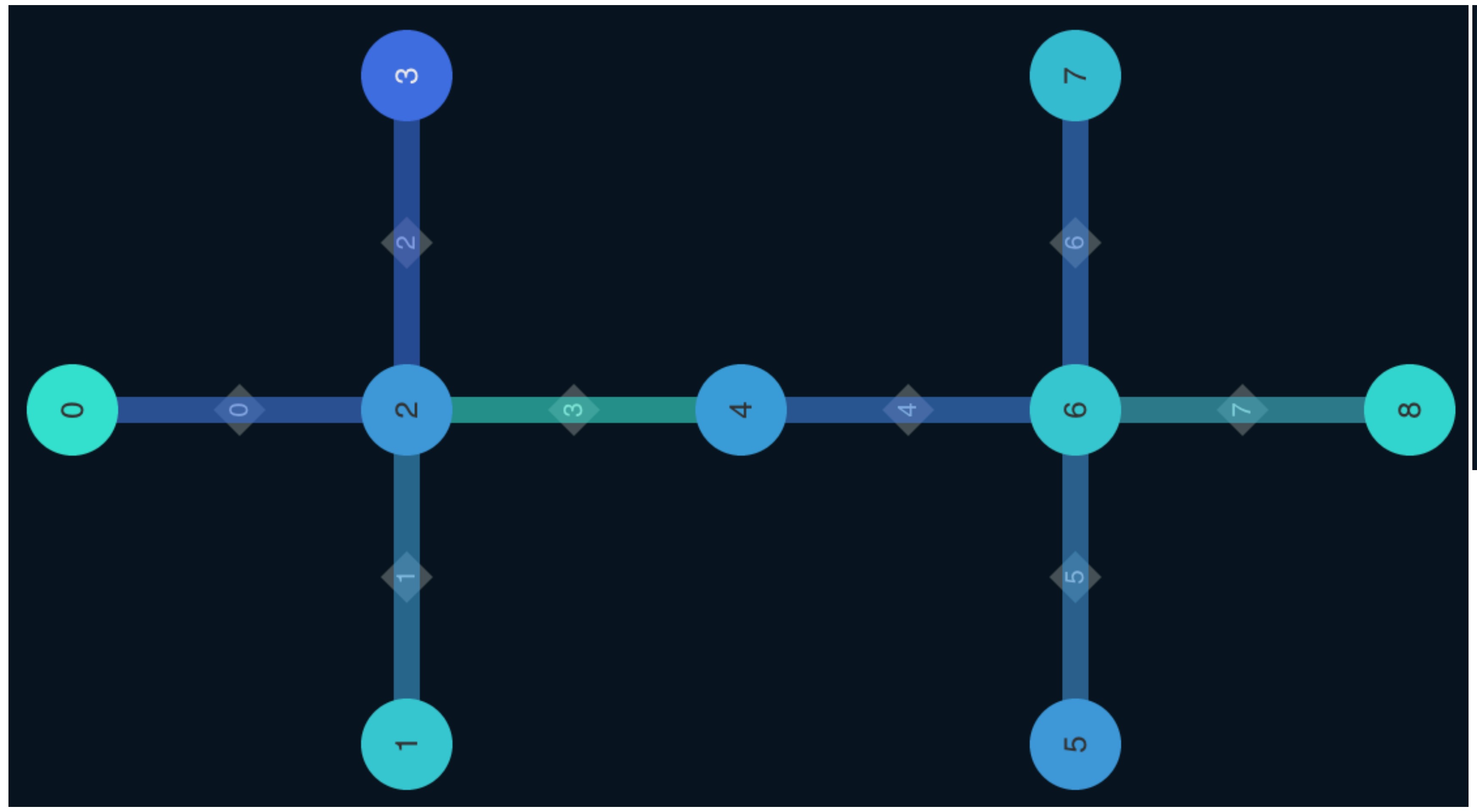


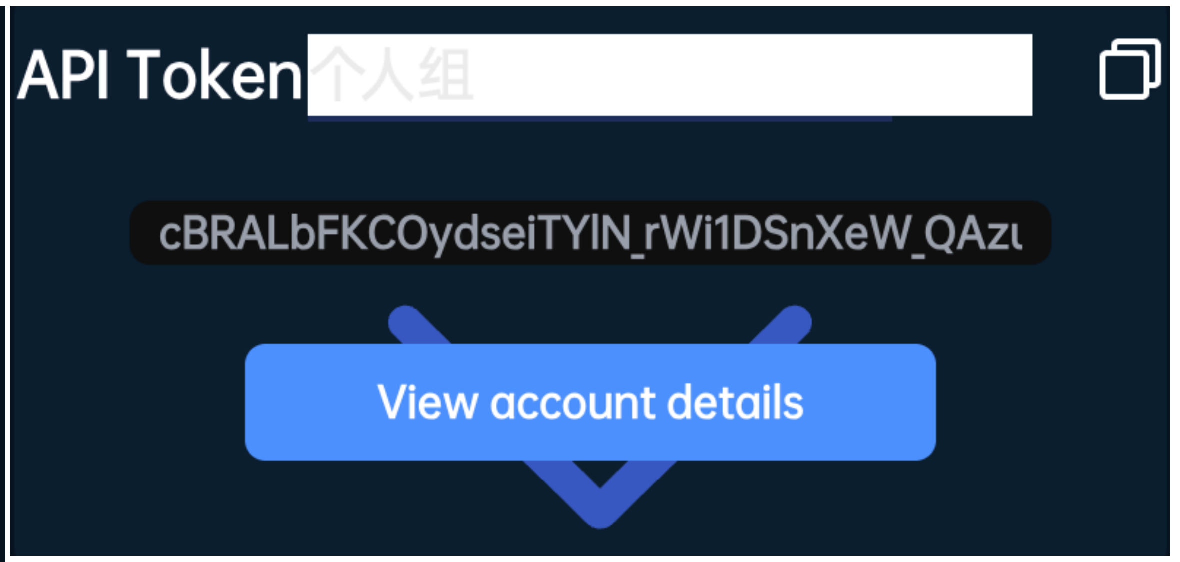

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

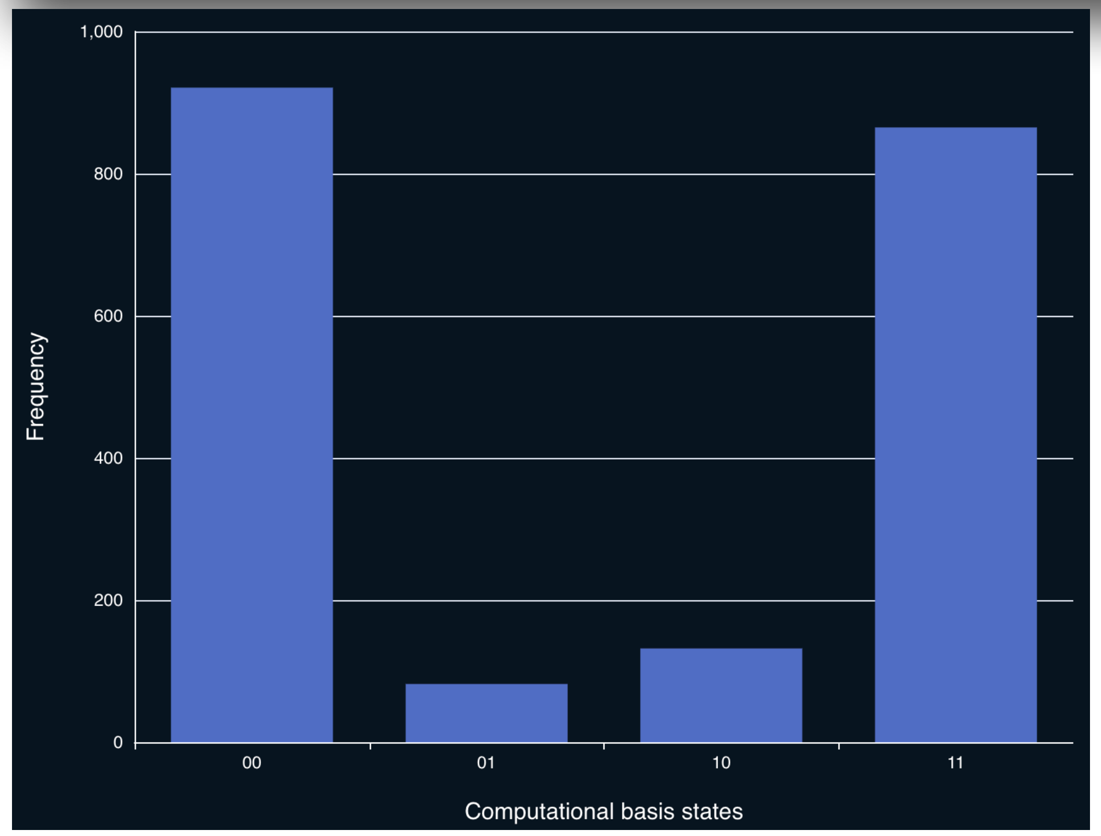
# Running on real hardwares

- Quafu from Beijing Academy of Quantum Information Sciences [https://quafu.baqis.ac.cn]
- `pip install pyquafu`    Some other quantum cloud platform: OriginQ (not free)


```python
from quafu import User
from quafu import QuantumCircuit
from quafu import Task

#user = User("<your API token>")
user = User("cBRALbFKCOydseiTYlN_rWi1DSnXeW_QAzu9-w3F9Da.
user.save_apitoken()
print(user.get_available_backends())

qc = QuantumCircuit(2)
qc.h(0)
qc.cnot(0,1)
qc.measure()

task = Task()
task.config(backend="Dongling", shots=2000, compile=True)

# submit job asynchronously
res = task.send(qc, wait=False)
# retrieve results after the job is done
#res = task.retrieve("<Your Task ID>")
print(res.counts) #counts
print(res.probabilities) #probabilities
```

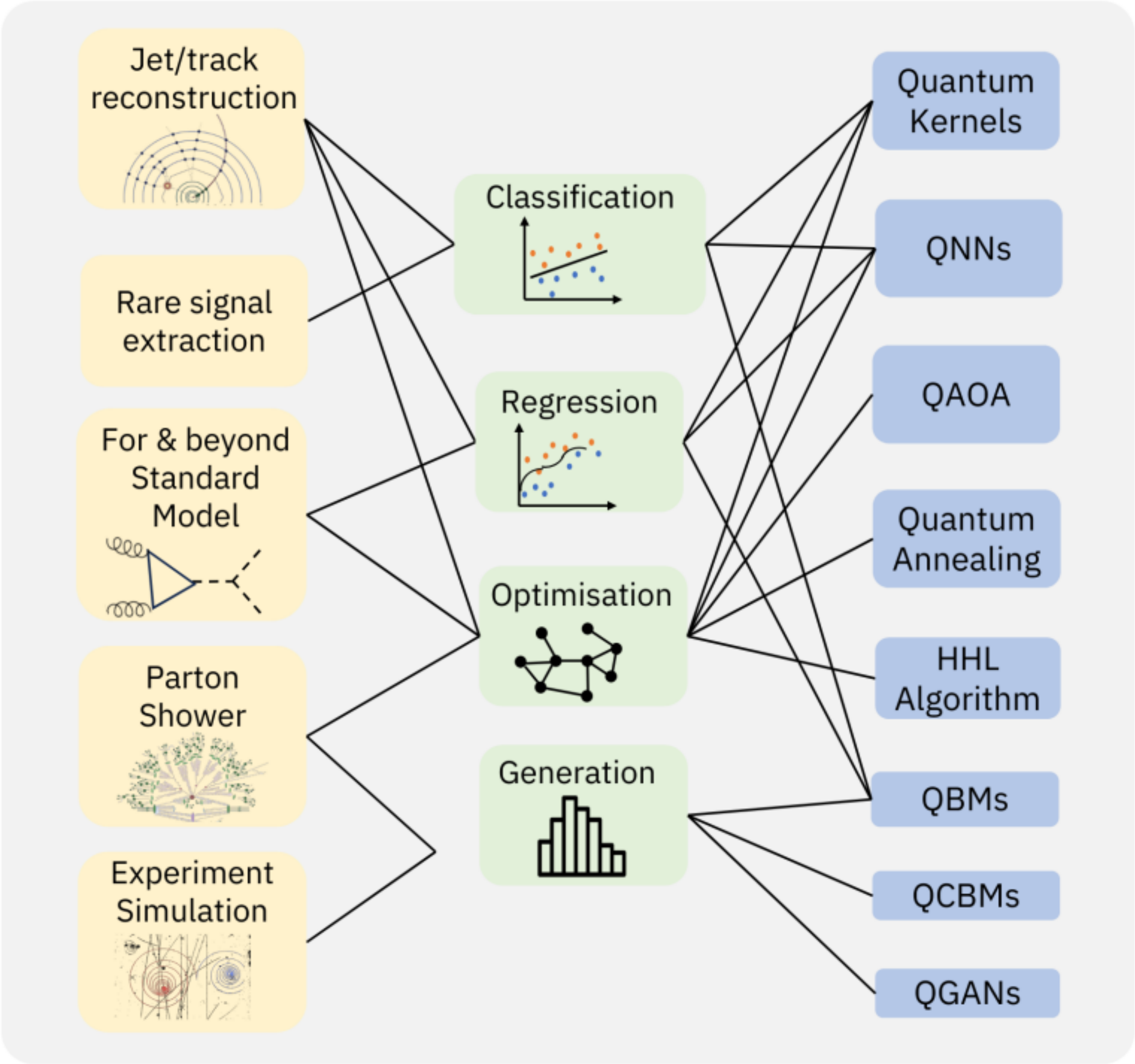


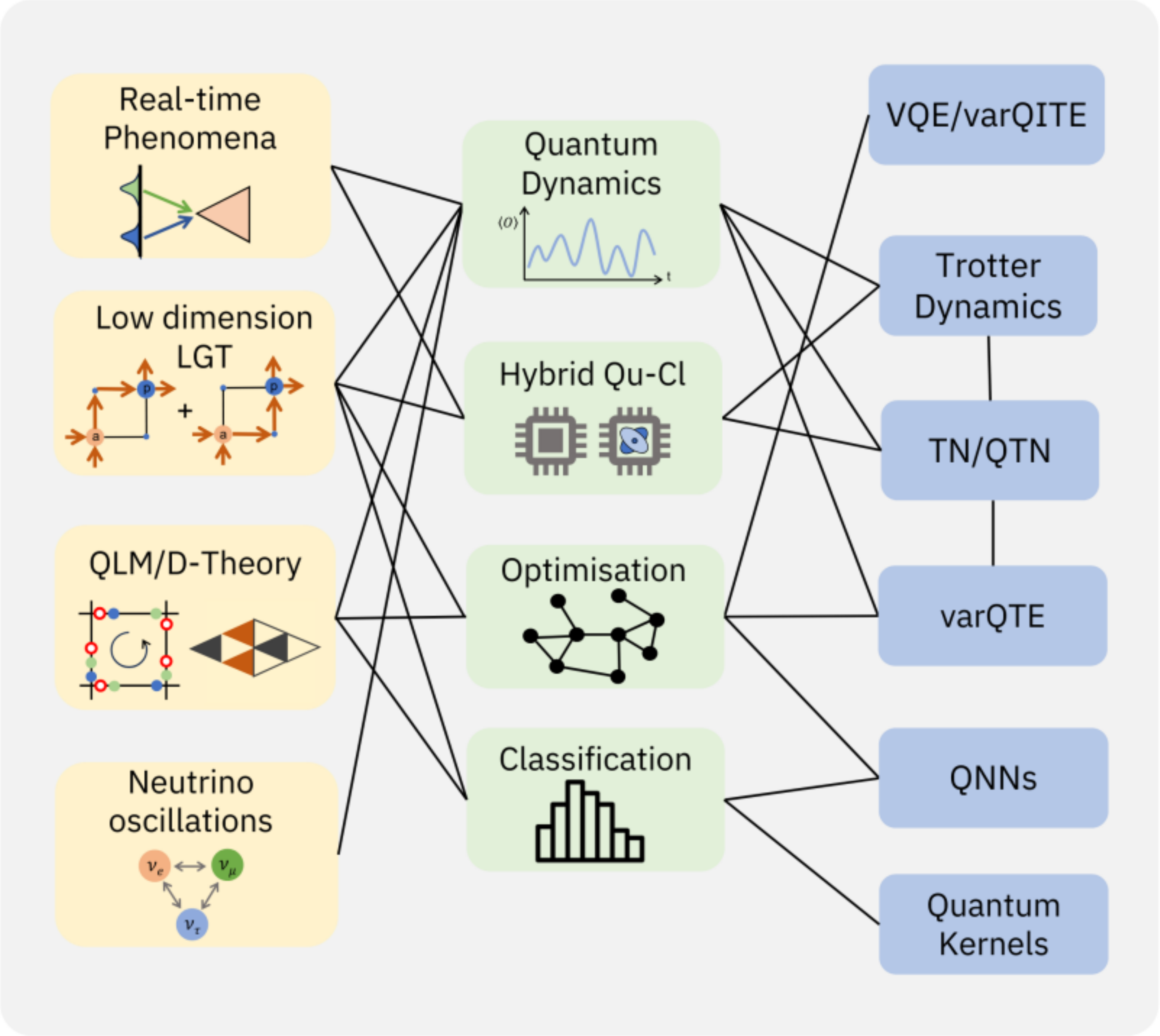


# Application of Quantum Computing in HEP

Experiment




Theory

Quantum machine learning for HEP experiments

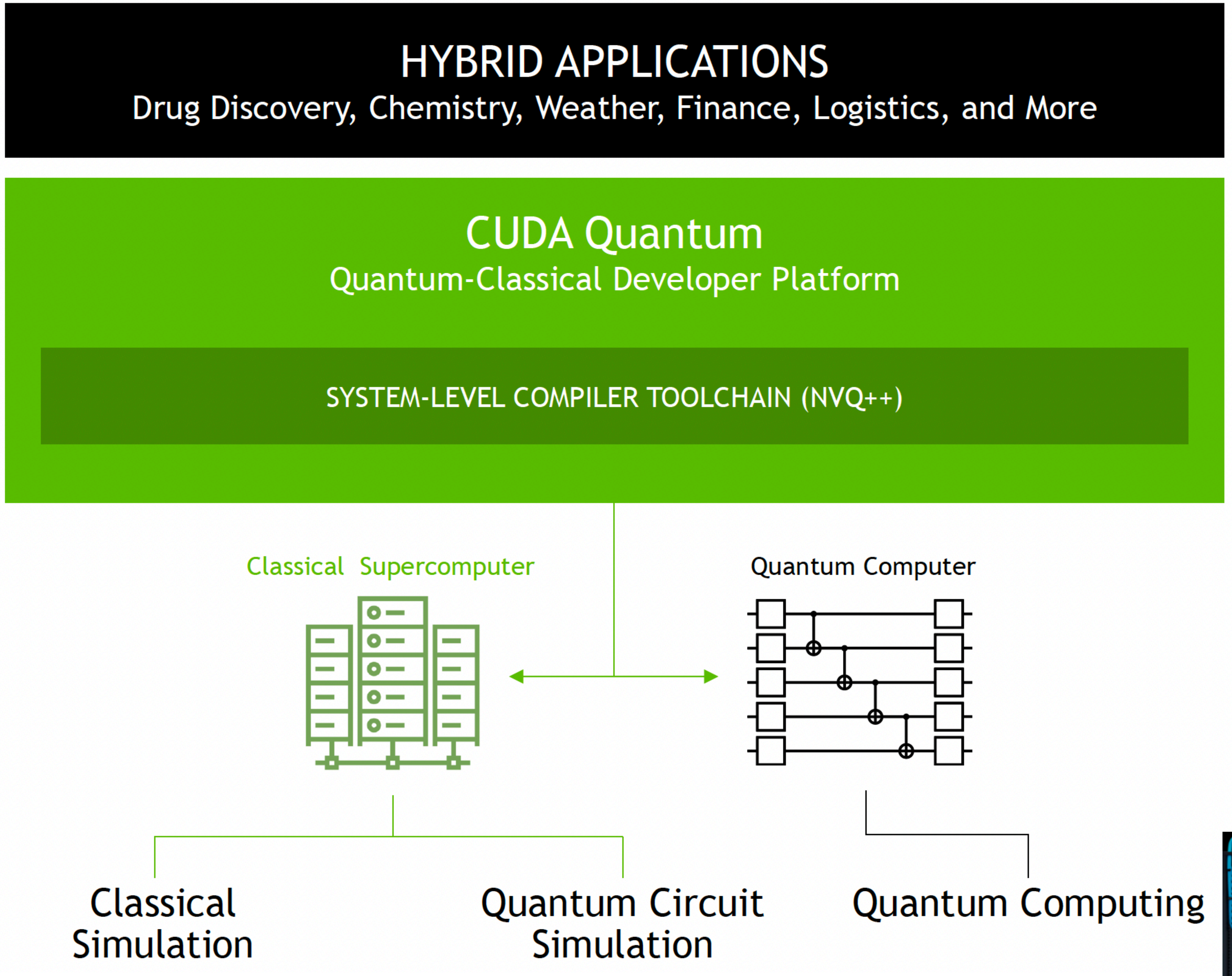
- **Classification of particle collision events**
- **Particle track reconstruction**
- W. Guan et al, Mach. Learn.: Sci. Technol. 2021

Quantum simulation of quantum field theories

- **1+1 dimensional model on atomic, optical, trapped ion, superconducting qubits**
- C. Bauer et al., PRX Quantum 4, 027001, 2023

Summary of the QC4HEP Working Group [arXiv: 2307.03236]

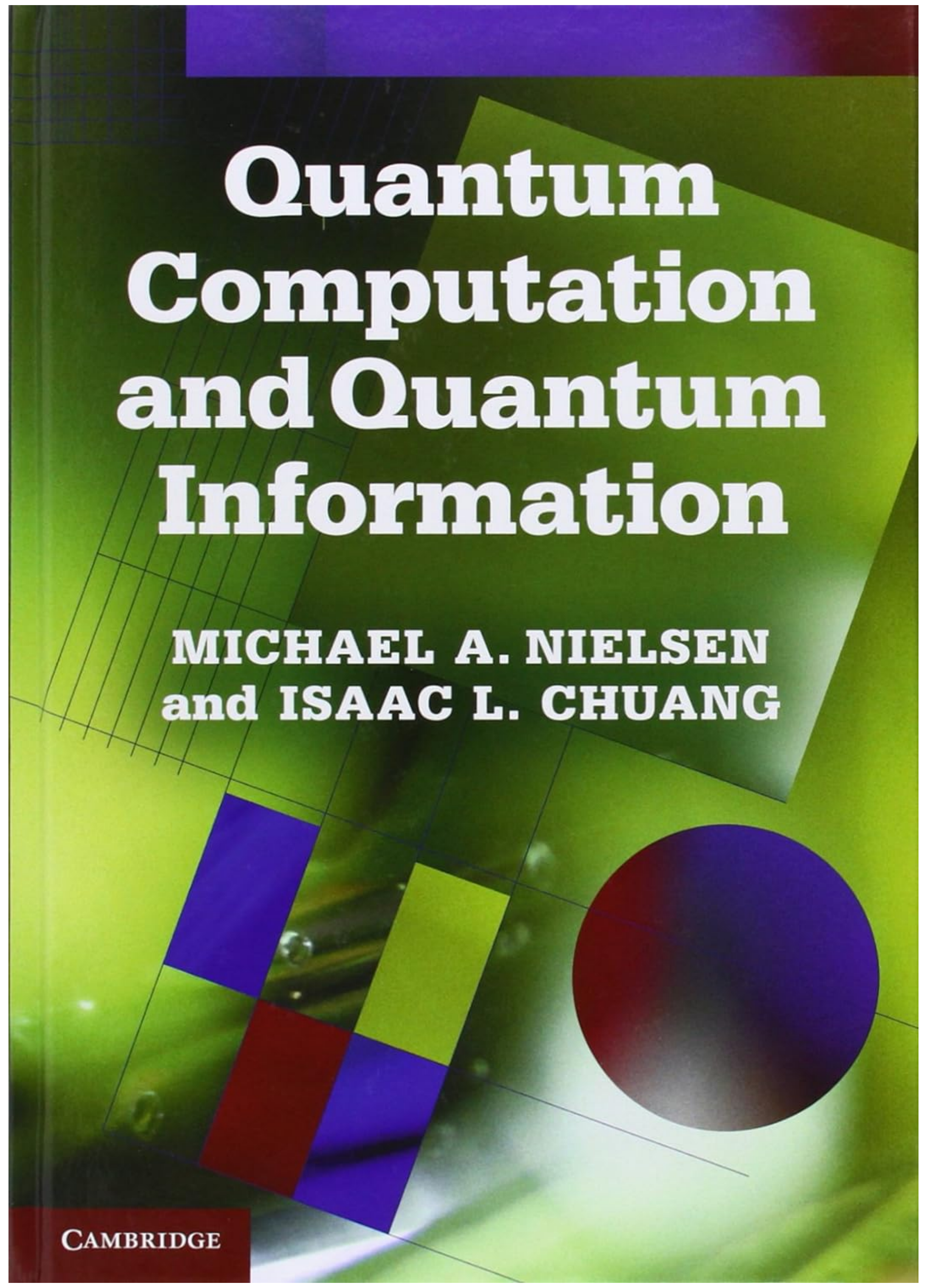
# The Future - Hybrid Quantum Classical Computing

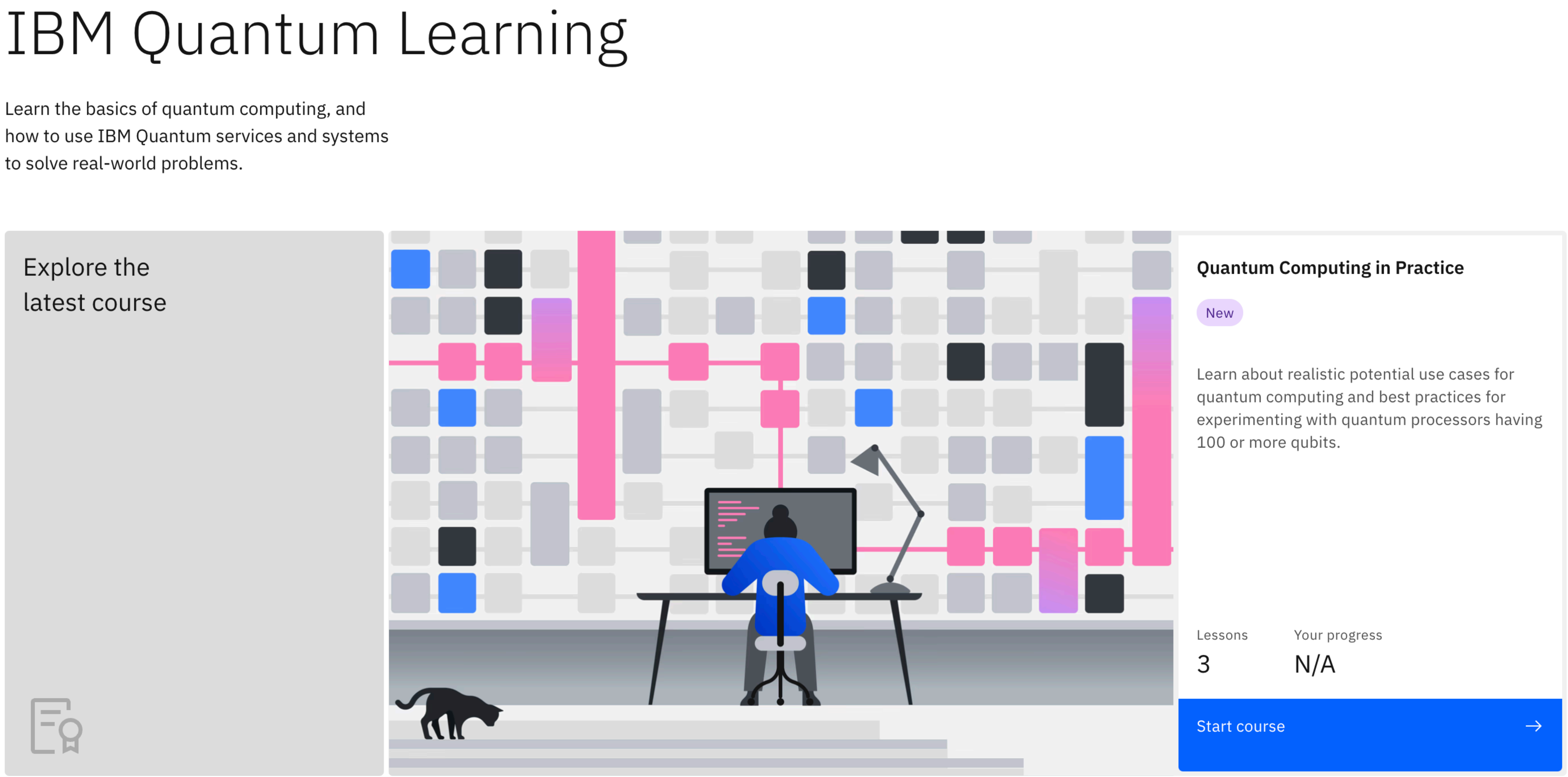



```
#include <cudaq.h>

int main() {
      // Define the CUDA Quantum kernel as a C++ lambda
        auto ghz =[](int numQubits) __qpu__ {
           // Allocate a vector of qubits
                cudaq::qvector q(numQubits);

           // Prepare the GHZ state, leverage standard
           // control flow, specify the x operation
           // is controlled.
                h(q[0]);
                for (int i = 0; i < numQubits - 1; ++i)
                        x<cudaq::ctrl>(q[i], q[i + 1]);
        };

      // Sample the final state generated by the kernel
auto results = cudaq::sample(ghz, 15);
results.dump();

return 0;
}
```

# Summary and Further Reading

- Covered the very basics of quantum computing, including qubits, quantum gates, quantum programming softwares and running jobs on real hardware.
- Further reading: plenty of useful online resources




A practical introduction to quantum computing(CERN): https://indico.cern.ch/event/970903/

**If you are interested in parallel and quantum computing, and want more in-depth discussion, please contact me (sunwei@ihep.ac.cn)**