高能所计算中心 IHEP Computing Center

中国科学院高能物理研究所
Institute of High Energy Physics, Chinese Academy of Sciences

高能物理计算暑期学校 IHEP School of Computing

高能物理中的高性能计算技术

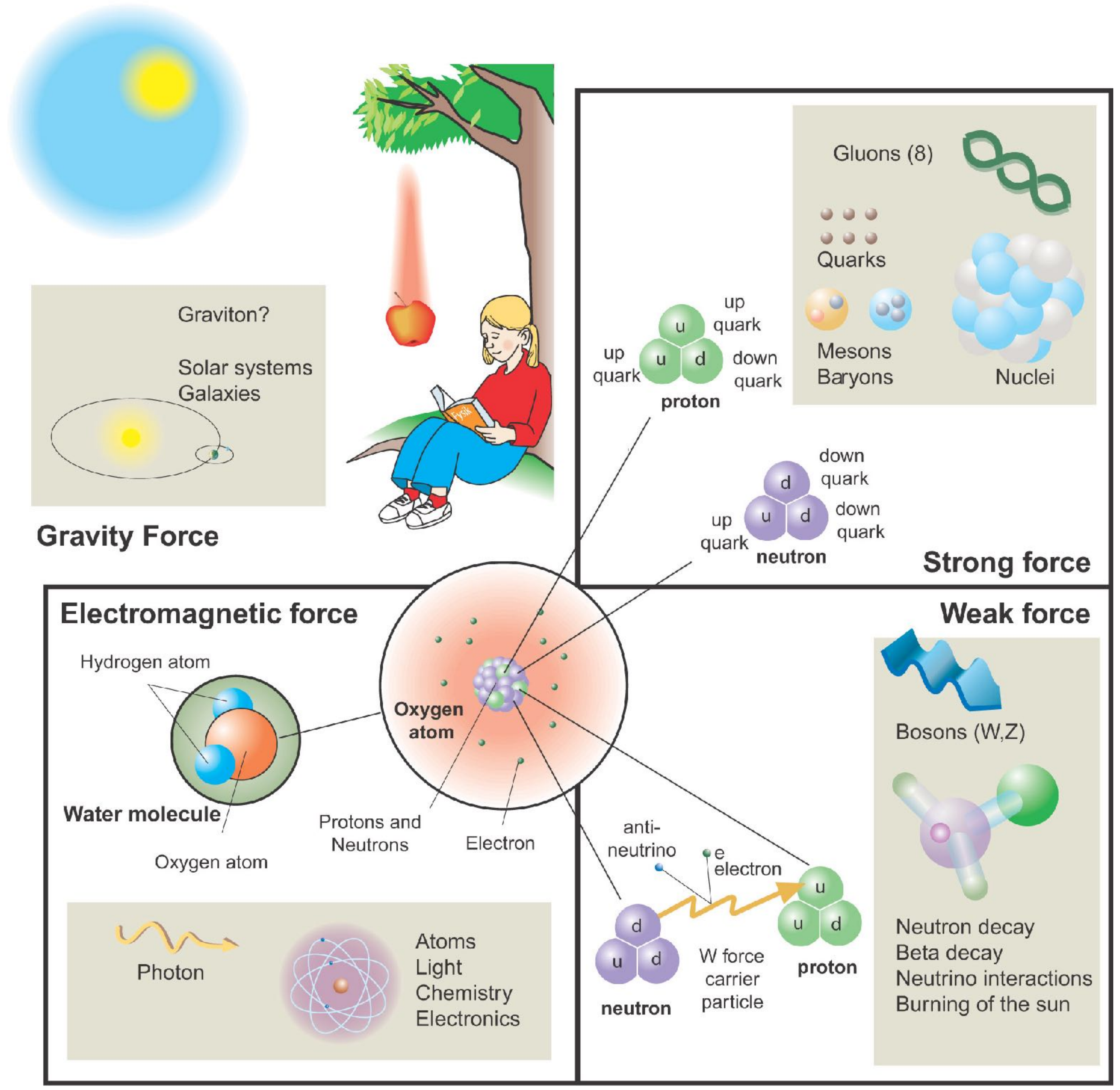
Wei Sun (孙玮)

Computing Center, Institute of High Energy Physics

IHEP School of Computing 2023, 2023.8.16-18

Contents

Introduction



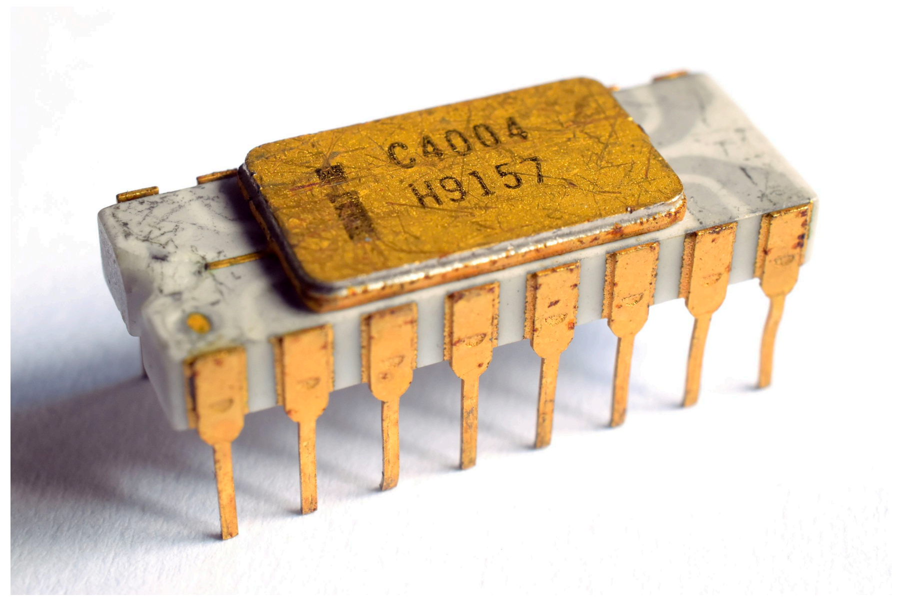
Standard Model

Weinberg–Salam theory

Electromagnetic interaction
Weak interaction

Quantum Chromodynamics

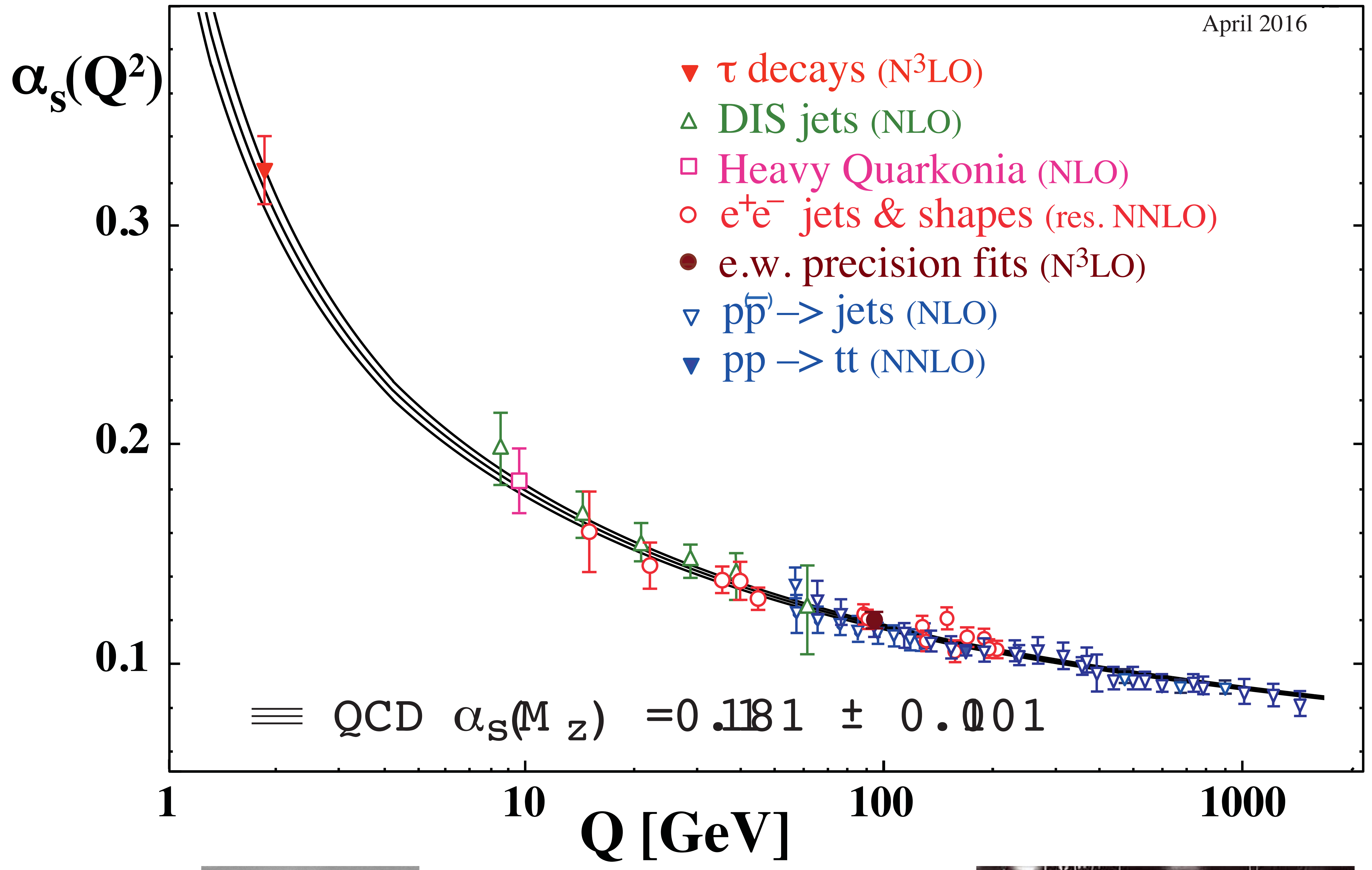
Strong interaction



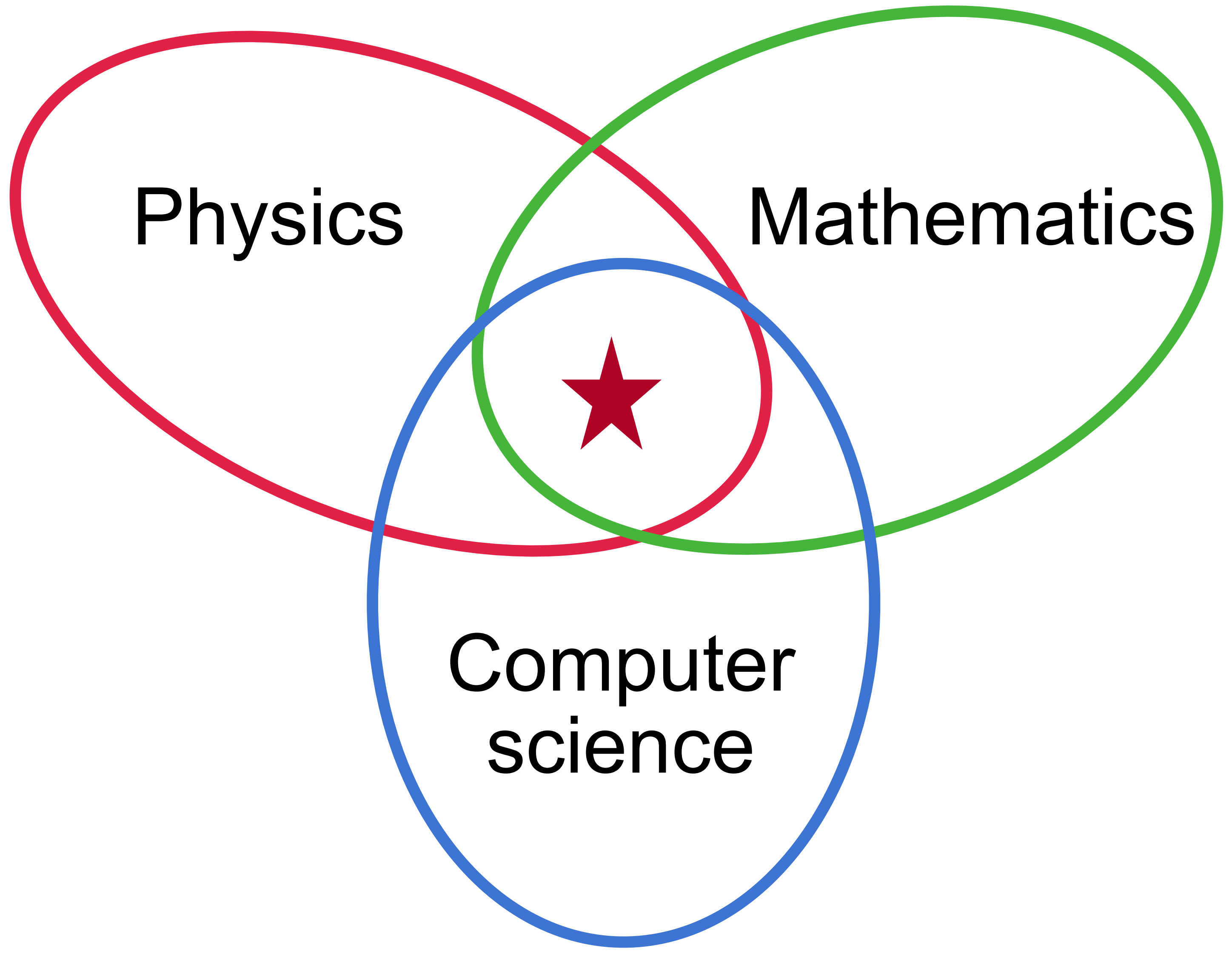
classical computer

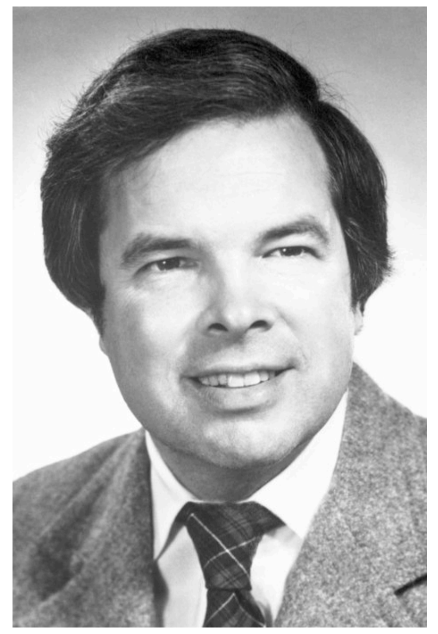
quantum computer

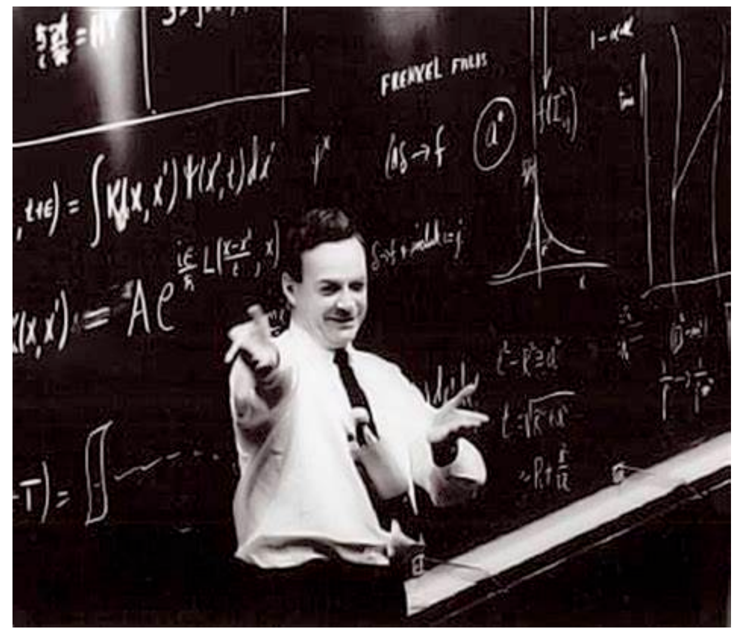
Introduction



- 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	$\sim$10PB	>100PB	300Gb/s (burst)
Lattice QCD	$\sim$1G core-hrs/yr	$\sim$ 100G – 1000G core-hrs/yr	$\sim$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	$\sim$1PB	10 – 100PB	
Energy Frontier Experiments	$\sim$ 100M core-hrs/yr	$\sim$ 10G – 100G core-hrs/yr	$\sim$1PB	>100PB	300Gb/s
Intensity Frontier Experiments	$\sim$ 10M core-hrs/yr	$\sim$ 100M – 1G core-hrs/yr	$\sim$1PB	10 – 100PB	300Gb/s

ASCR/HEP Exascale Report [arXiv:1603.09303]

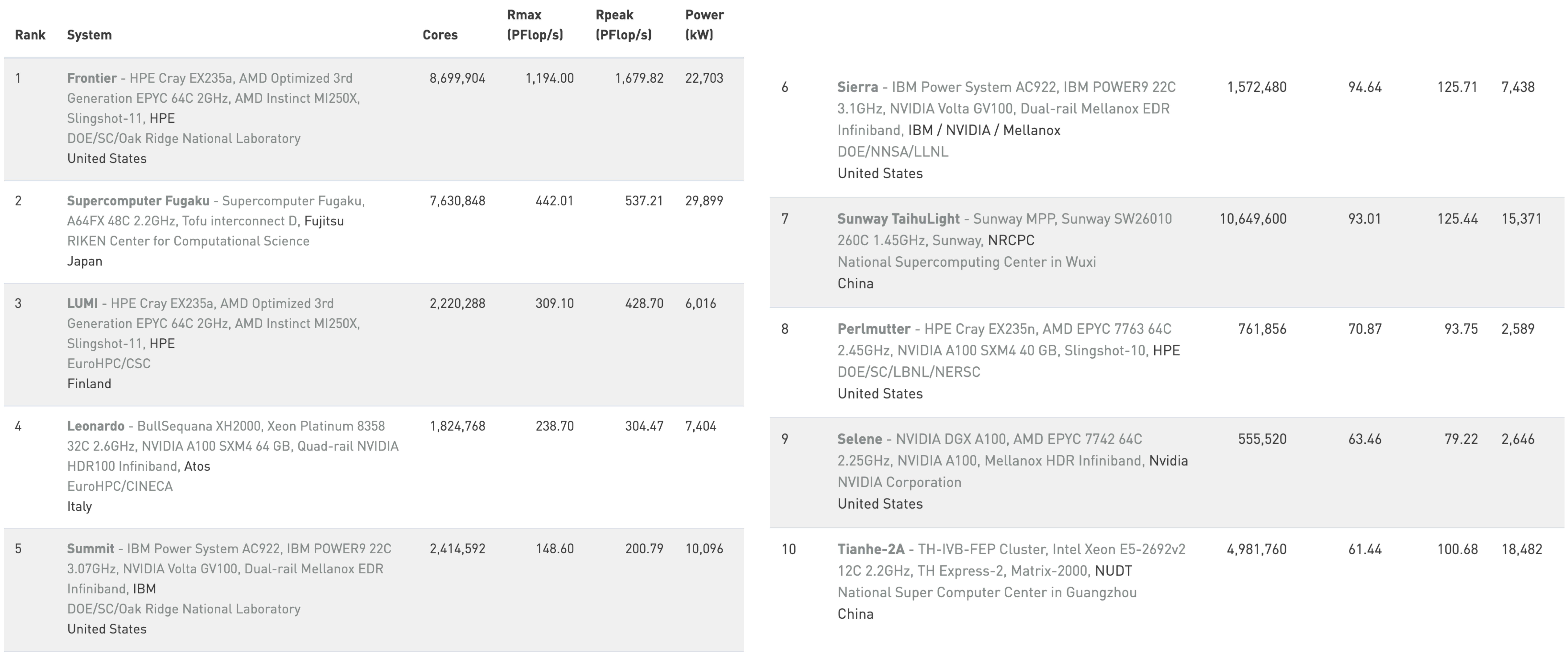
Introduction

my definition

High Performance Computing $\approx$ Numerical Linear Algebra on Supercomputers

High performance computers and supercomputers

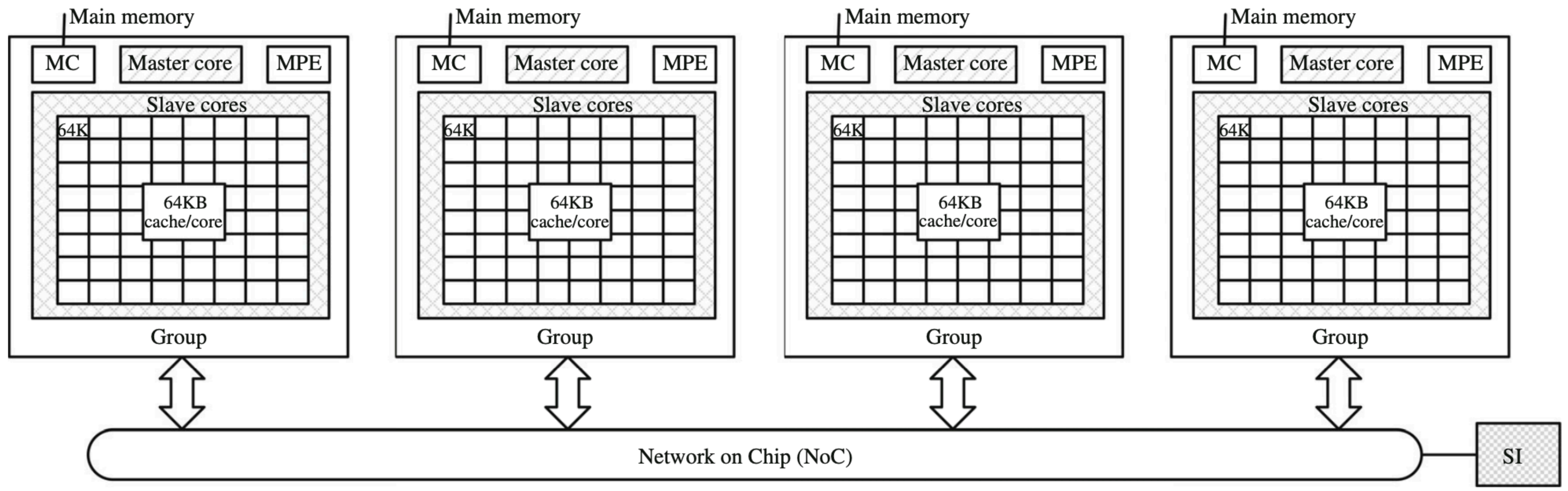
https://www.top500.org/lists/top500/2023/06/



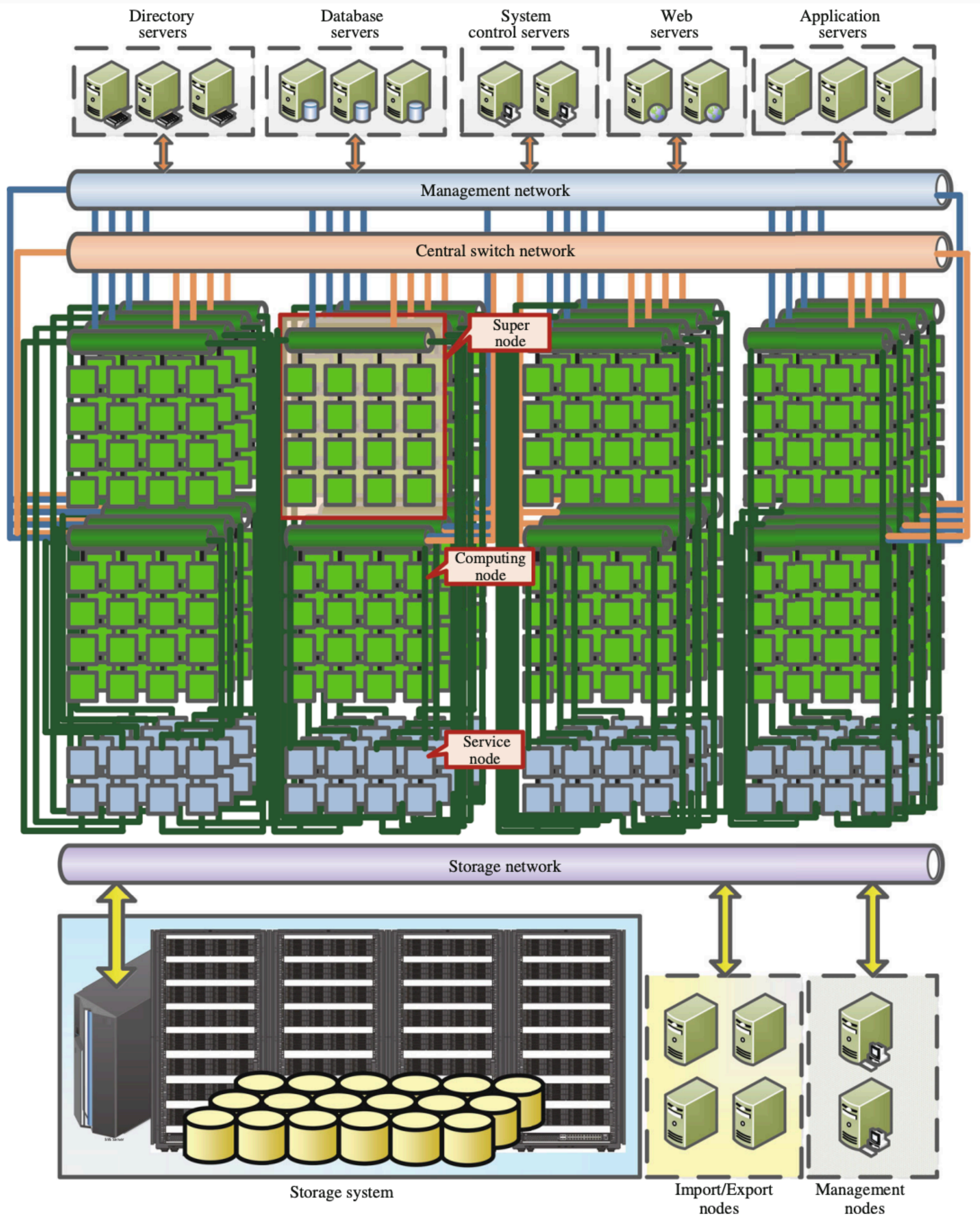
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
6	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	125.71	7,438
7	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93.01	125.44	15,371
8	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70.87	93.75	2,589
9	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63.46	79.22	2,646
10	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61.44	100.68	18,482

High performance computers and supercomputers

Supercomputer example: Sunway TaihuLight

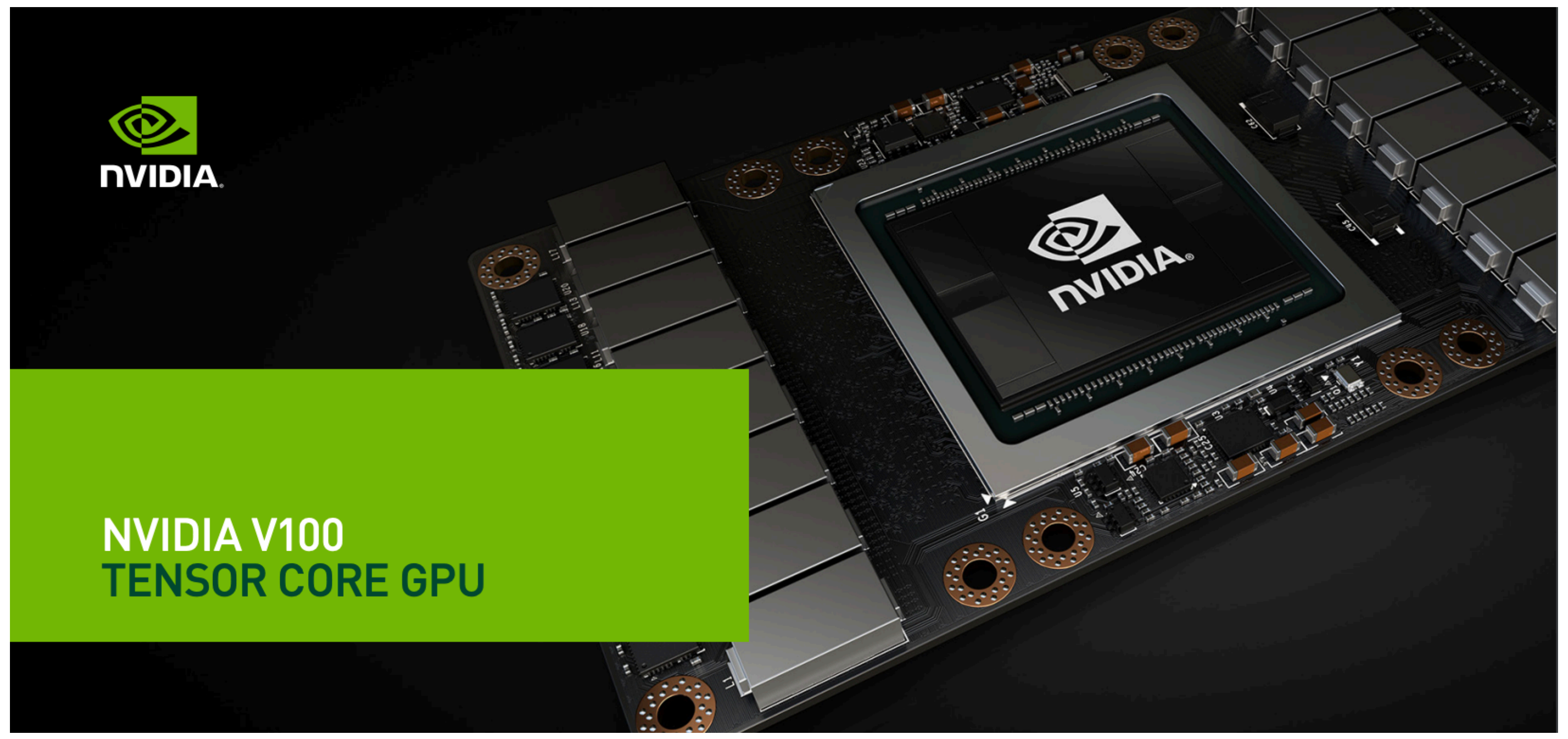


Fu, H., Liao, J., Yang, J. *et al.,* Sci. China Inf. Sci. **59,** 072001 (2016).

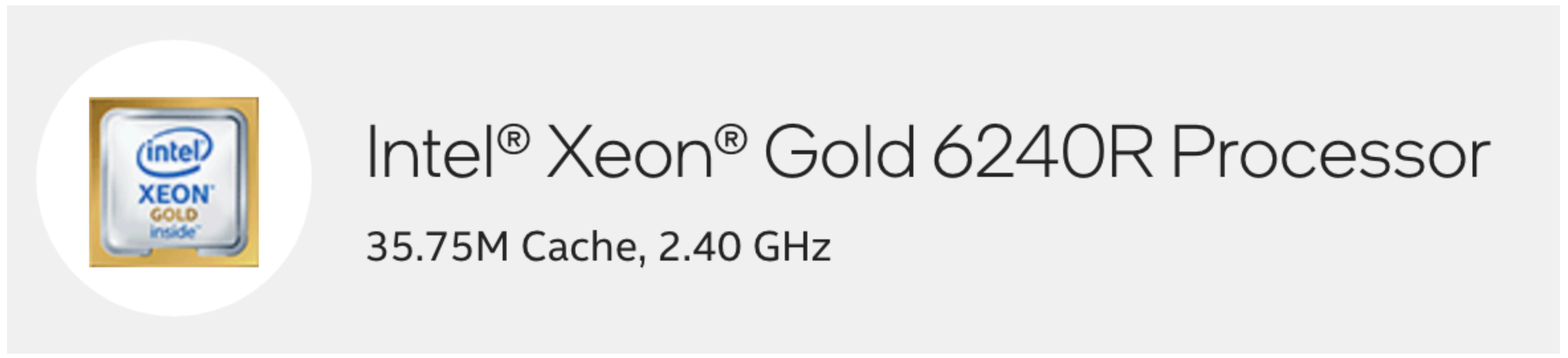


High performance computers and supercomputers

High performance clusters at IHEP



Nvidia V100 GPU: $\approx$ 300 cards



x86 CPU: $\approx O(10000)$ cores

ARM CPU: $\approx O(10000)$ cores

High performance computers and supercomputers

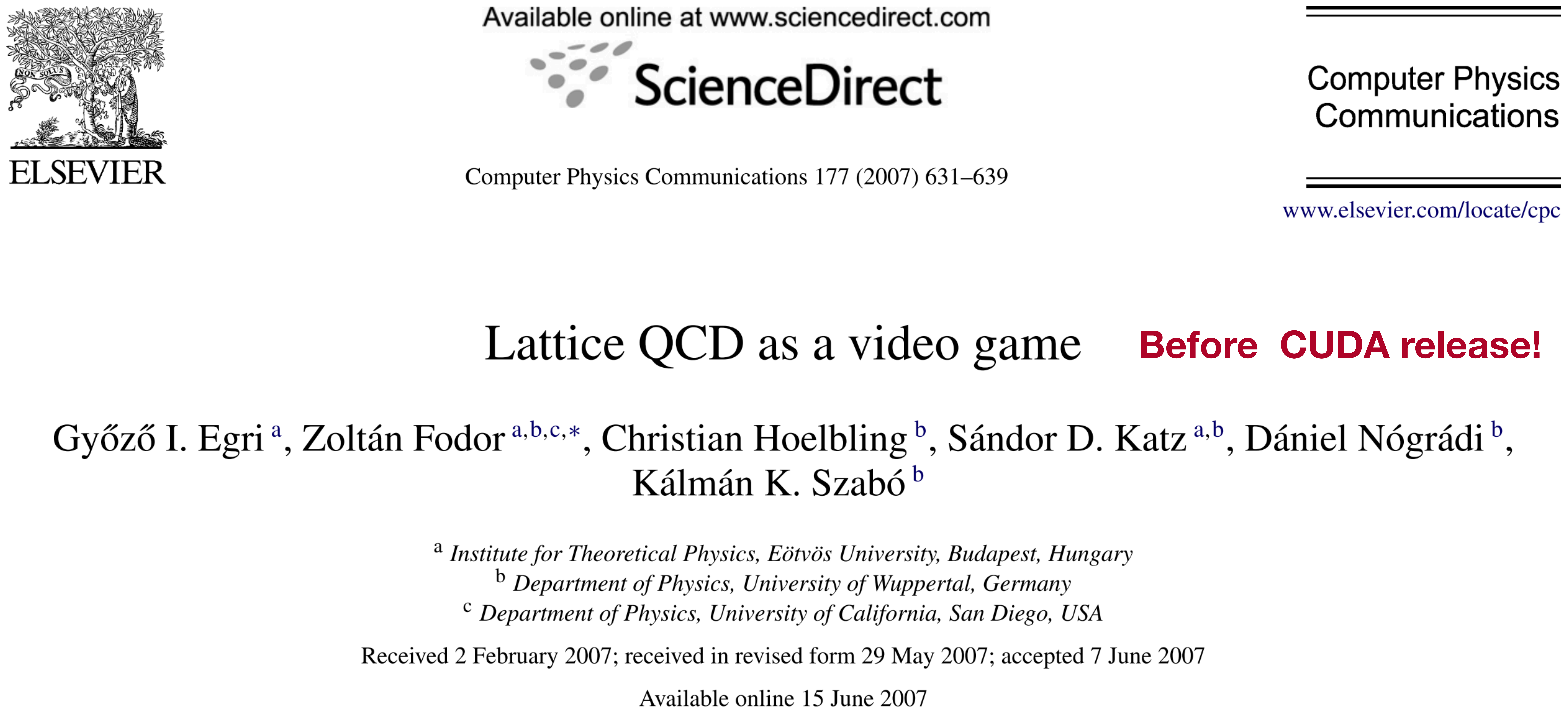
Example in HEP: LQCD with HPC

- 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

ELSEVIER

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)
```

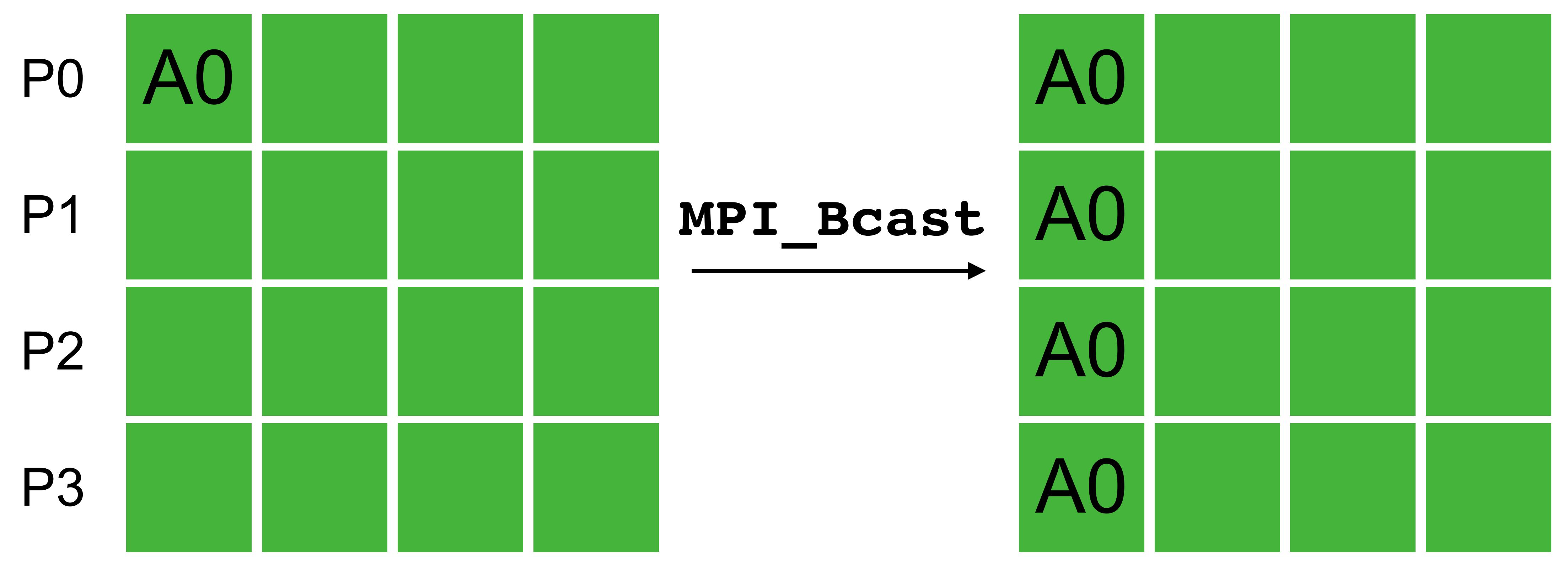
```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

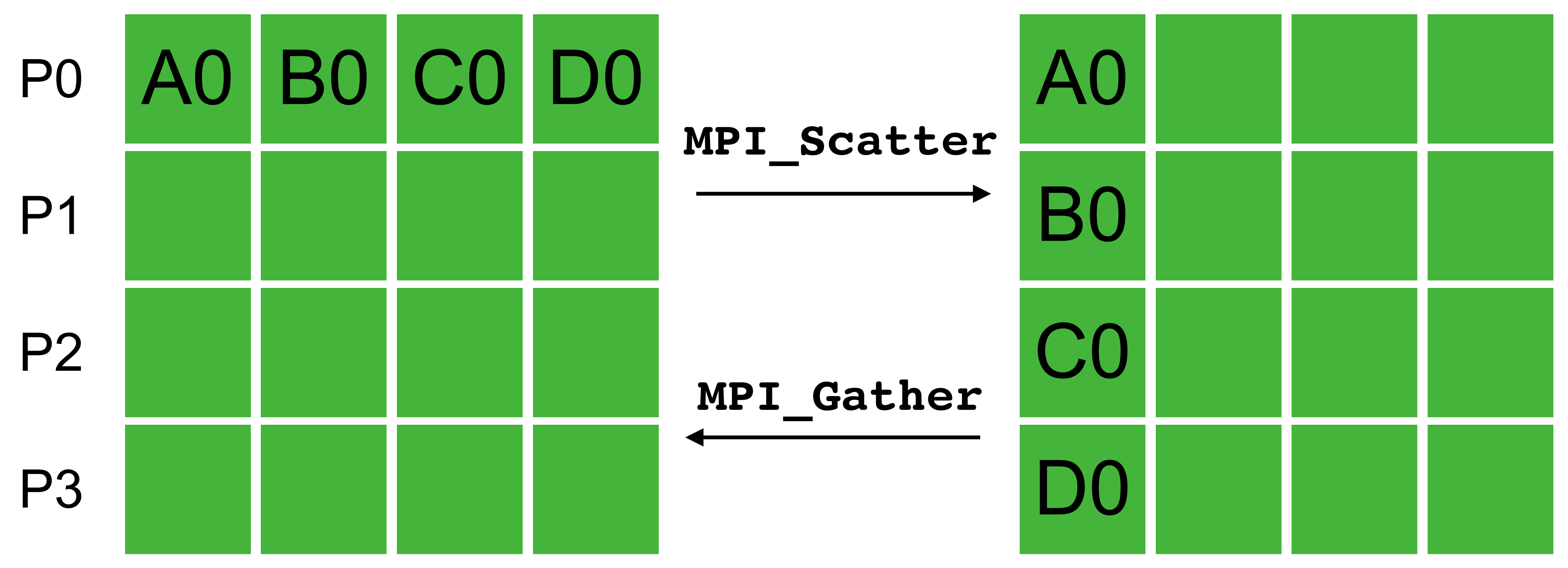
```
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

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);
}
```

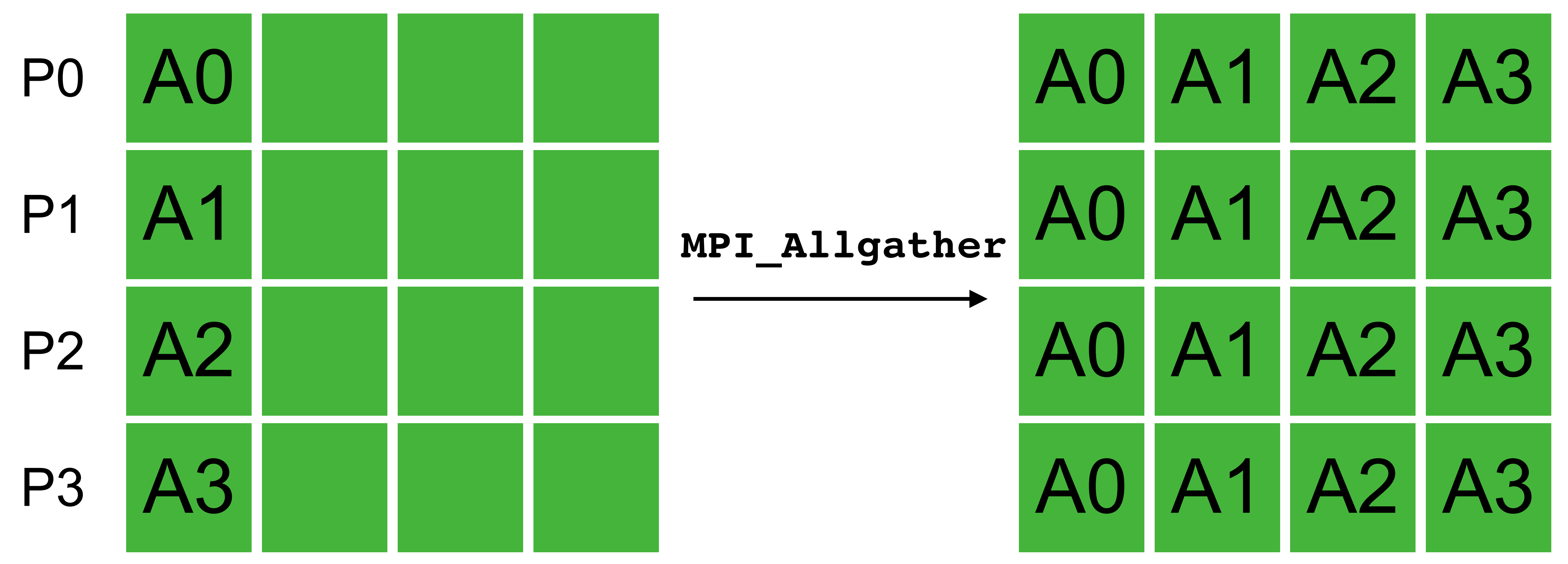
Parallel programming models

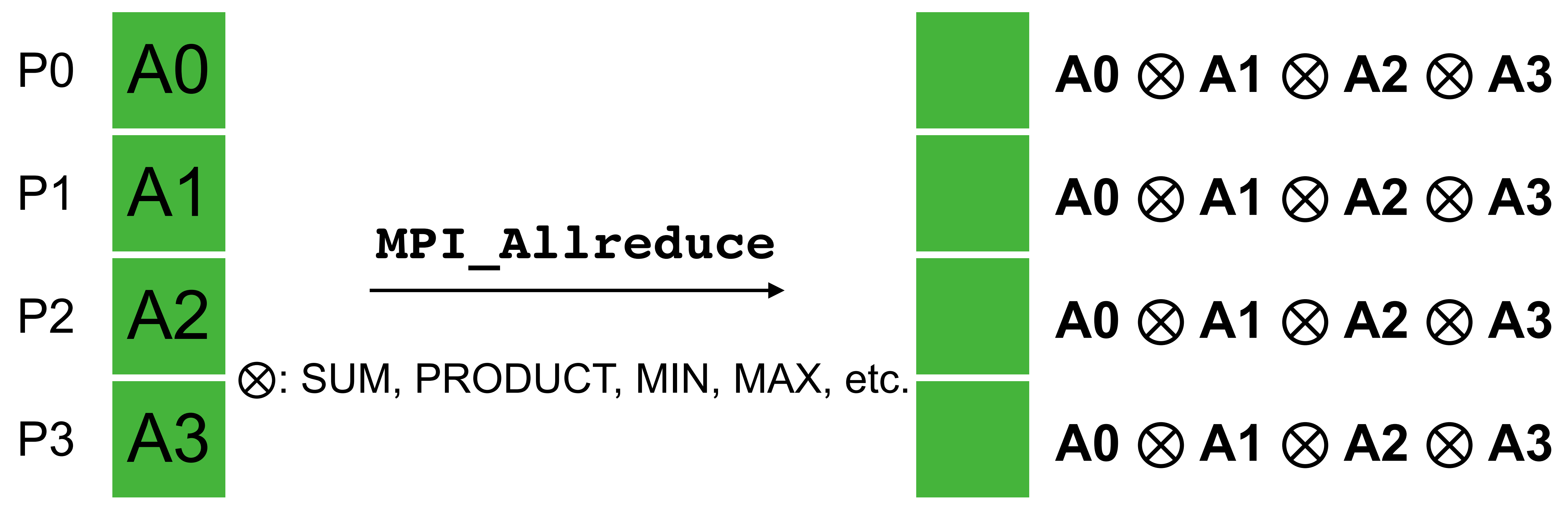
MPI Basics (collective communication)

- Total 400+ APIs





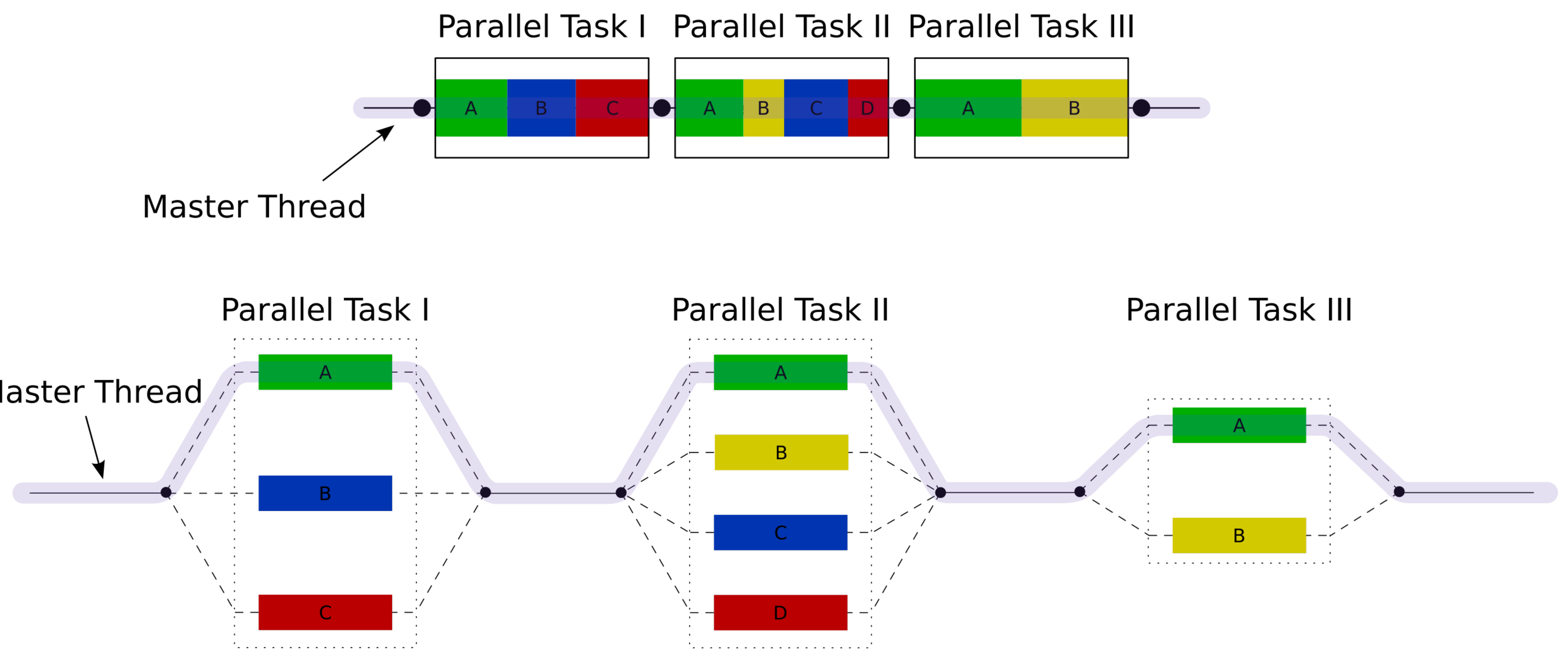




Parallel programming models

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;
}
```

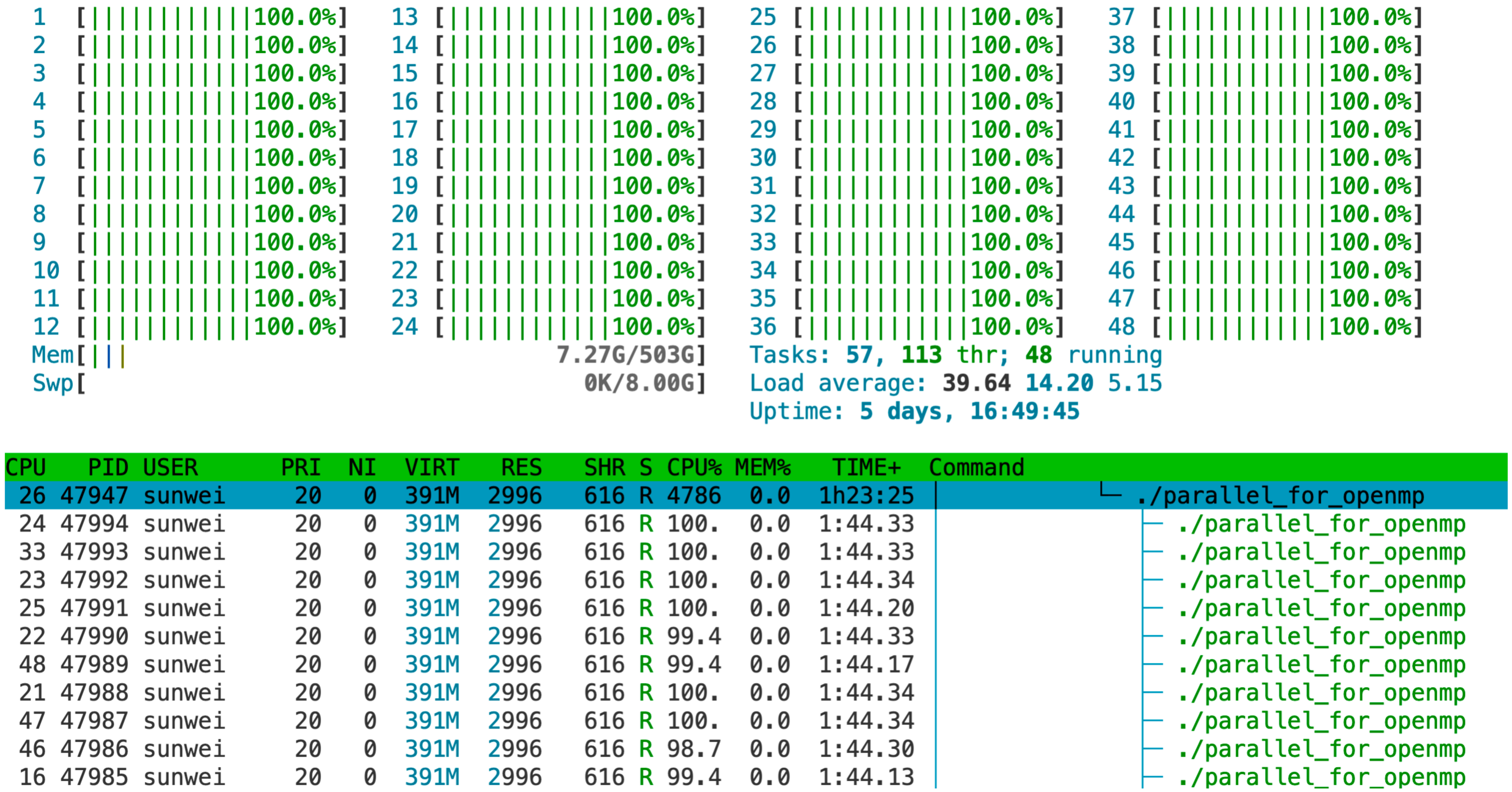
Compile: `gcc -fopenmp hello_world.c -o hello_world`

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



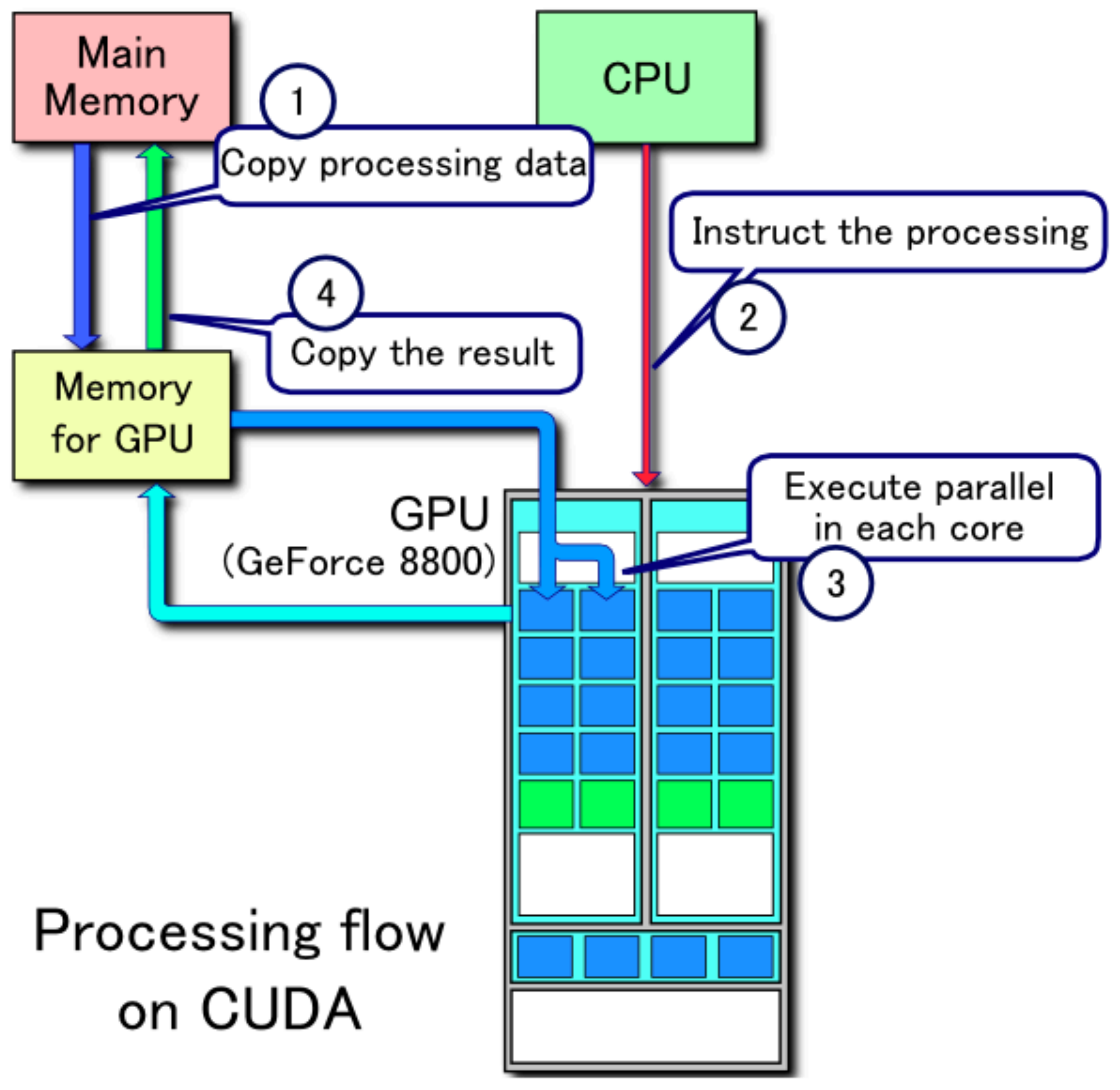
```
  1  [||||||||||||100.0%]   13 [||||||||||||100.0%]   25 [||||||||||||100.0%]   37 [||||||||||||100.0%]
  2  [||||||||||||100.0%]   14 [||||||||||||100.0%]   26 [||||||||||||100.0%]   38 [||||||||||||100.0%]
  3  [||||||||||||100.0%]   15 [||||||||||||100.0%]   27 [||||||||||||100.0%]   39 [||||||||||||100.0%]
  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

Details in Friday's lectures

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

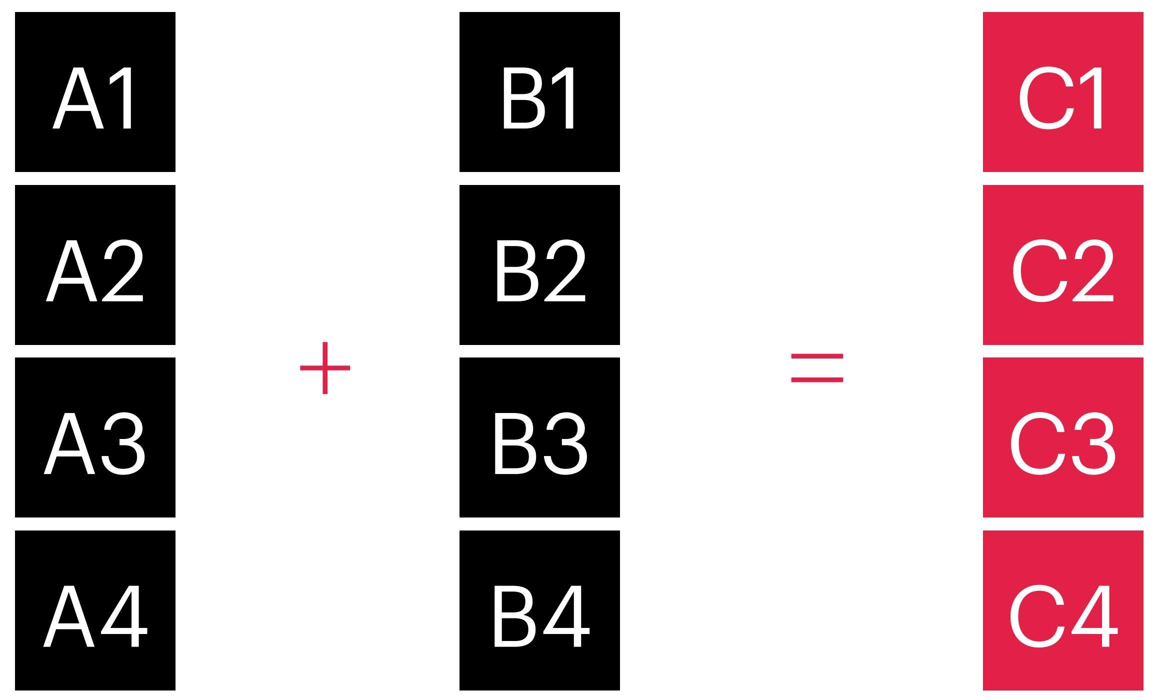


source: wikipedia

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

```
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

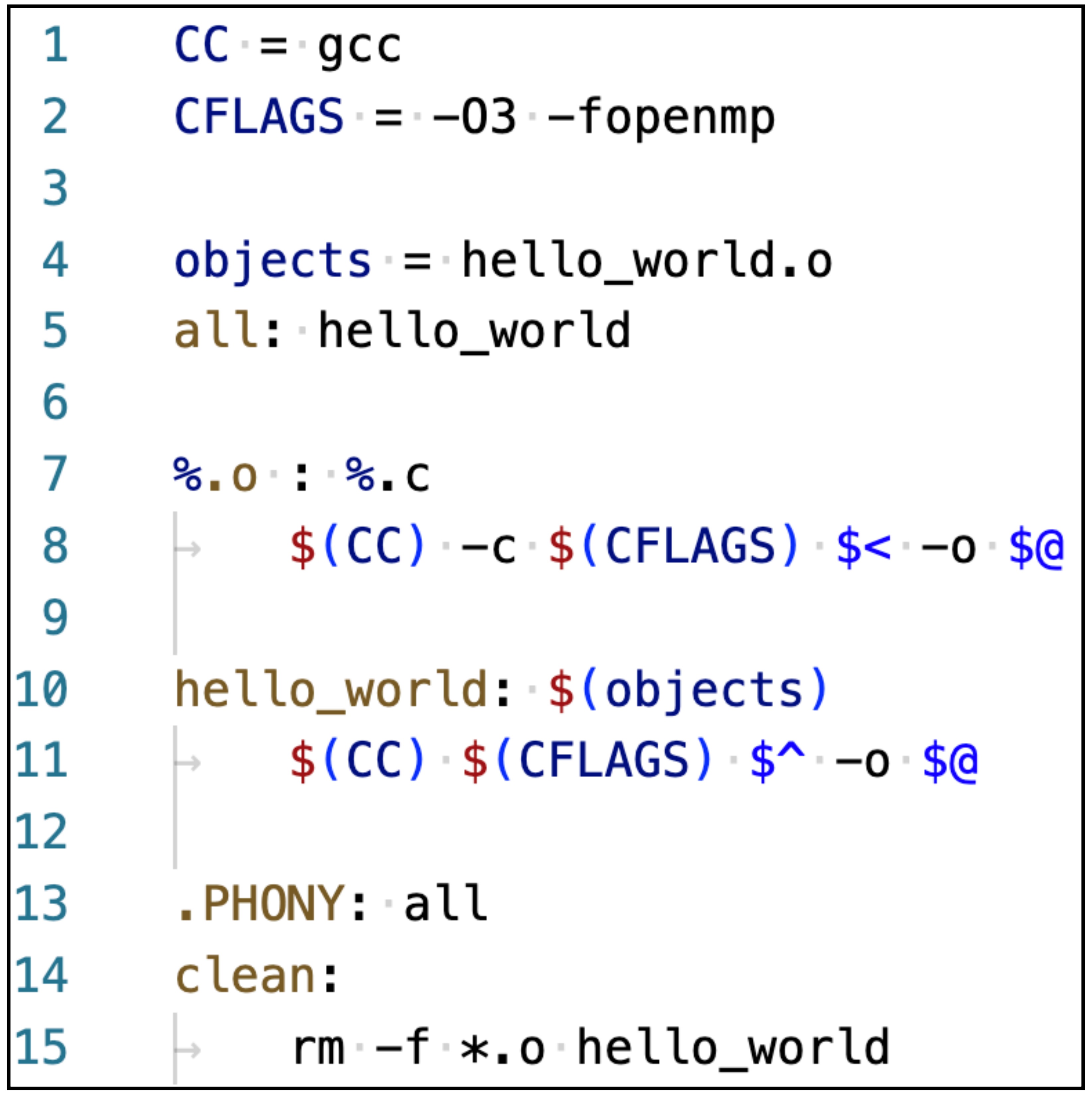
```
#include<immintrin.h>
//compile: g++ -O3 -mavx -o exe src.c

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

```
#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

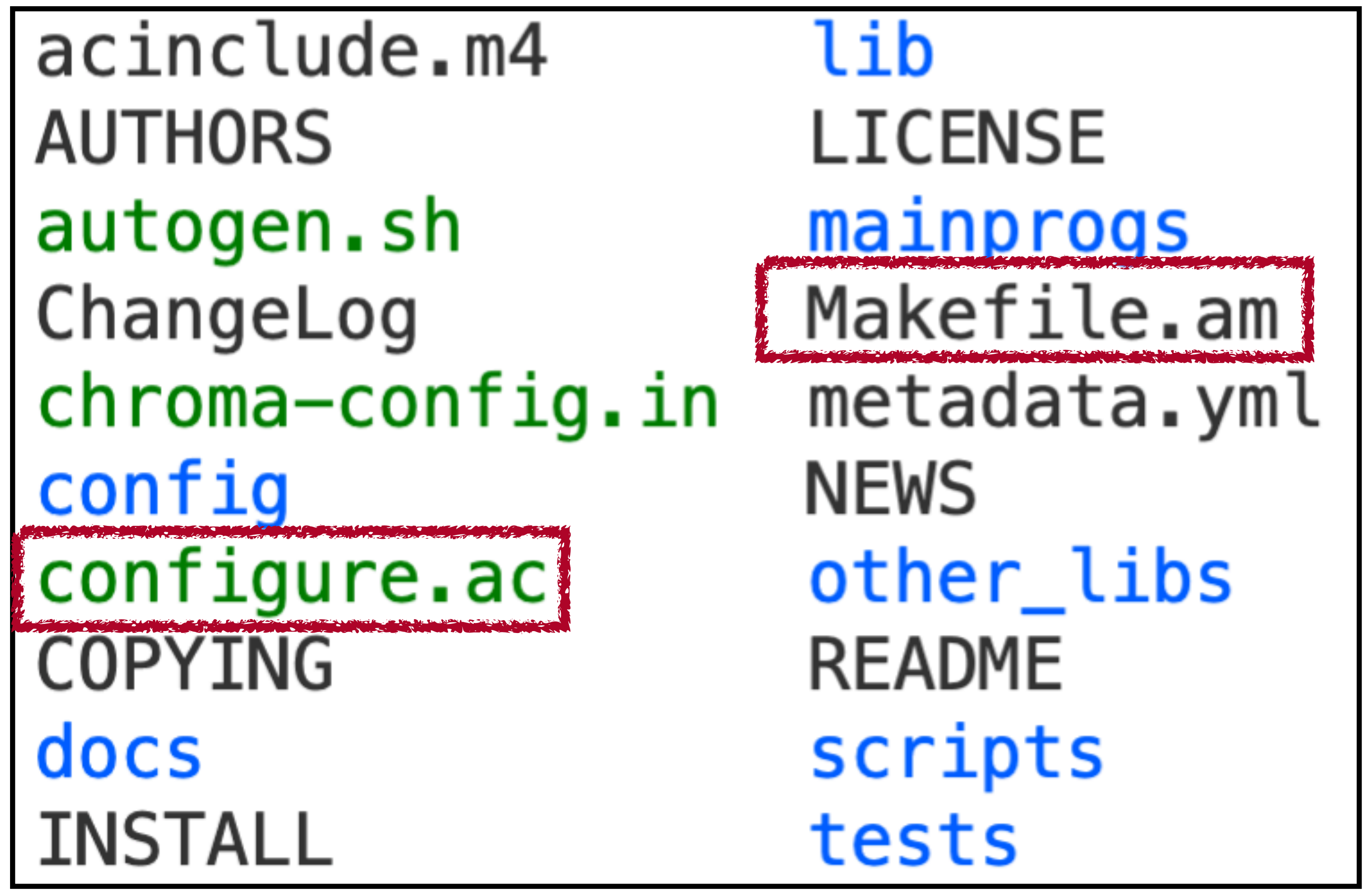
Software build tools



```
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`**



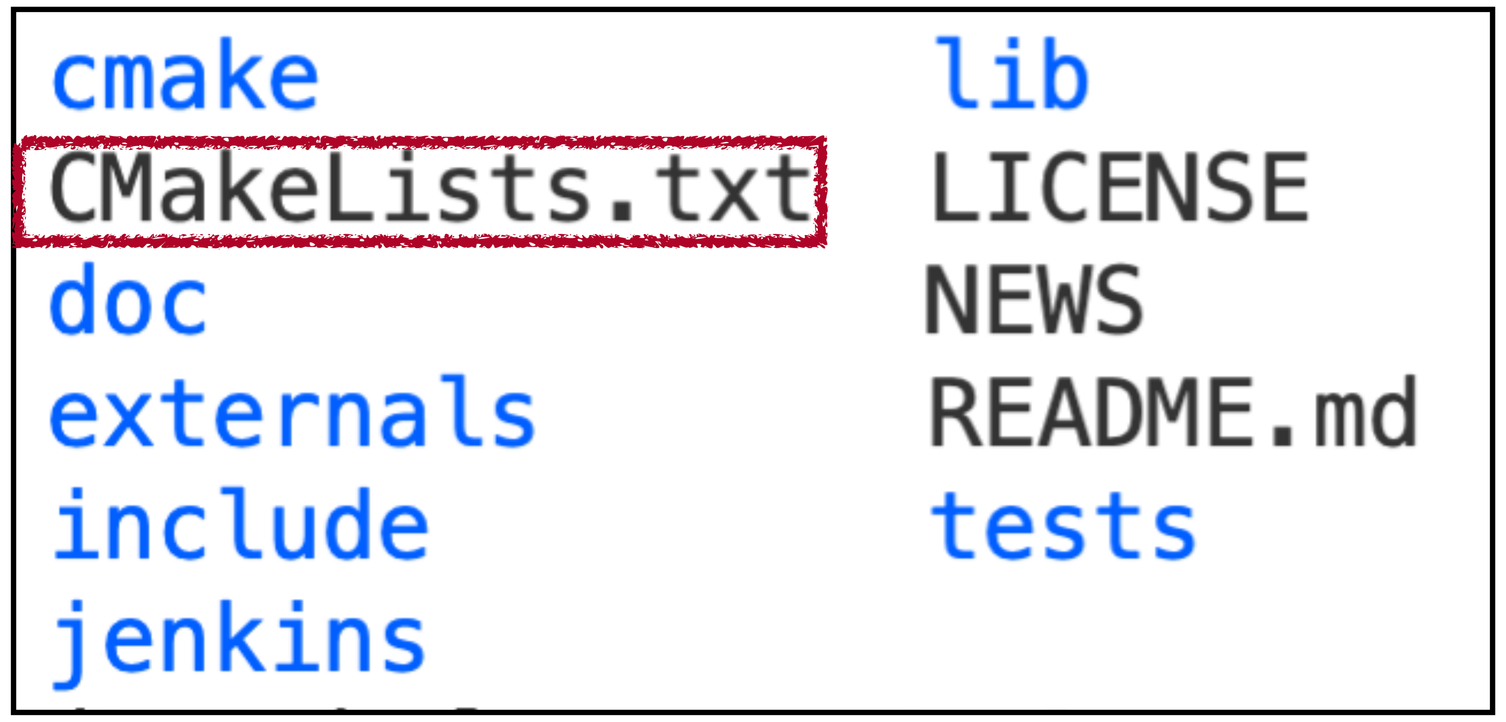
```
acinclude.m4       lib
AUTHORS            LICENSE
autogen.sh         mainprogs
ChangeLog          Makefile.am
chroma-config.in   metadata.yml
config             NEWS
configure.ac       other_libs
COPYING            README
docs               scripts
INSTALL            tests
```

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 tips

- Covered basics of high performance computing programming model 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 and 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**

Hands-on exercises

- MPI
  1. Hello world
  2. Compute π with formula $\frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots$
- OpenMP
  1. Compute π