Introduction to High-performance Neurocomputing

Tadashi Yamazaki (UEC)
Jun Igarashi (RIKEN)

CNS*2019 Barcelona Tutorial T7 2019/07/13

Account setup

1. Get a paper showing your account and machine names

2. Your account name is **`guestXX`**

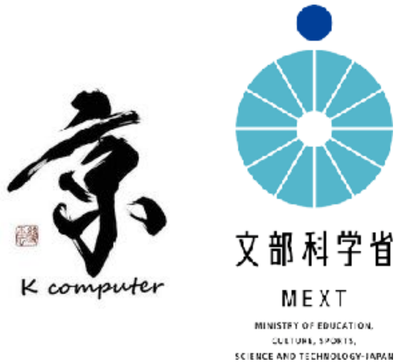
3. Give us your SSH public key with the name **`guestXX.pub`**

4. We will circulate a USB memory stick to store the keys

Welcome!

Acknowledgements

MEXT Post-K Exploratory Challenge #4



MEXT Grand-in-Aid for High-Performance Computing

NEDO Next Generation AI and Robot Core Technology

Timetable

09:30 - 10:00	Opening, Account setup
10:00 - 11:00	ODE, Single neuron simulation Network simulation
11:00 - 12:30	OpenMP and MPI
12:30 - 14:30	Lunch
14:30 - 16:00	GPU with CUDA
16:30 - 17:20	PEZY-SC2 with OpenCL
17:20 - 17:30	Closing

Lecture #1

ODE

Single neuron simulation

Network simulation

Integrate-and-fire neuron

$$\tau \frac{dv}{dt} = -(v(t) - V_{\text{leak}}) + RI_{\text{ext}}(t),$$

$$v(t) > \theta \Rightarrow \text{Spike (t)} = 1, v(t) \leftarrow V_{\text{reset}},$$

$$v(0) = V_{\text{init}}.$$

Integrate-and-fire neuron

τ : Membrane time constant V_{leak}: Leak potential

$$\tau \frac{dv}{dt} = -(v(t) - V_{leak}) + RI_{ext}(t),$$

$$\frac{v(t + \Delta t) - v(t)}{\Delta t}$$

R: Membrane resistance

I_{ext} (t): External current

Δ t : Temporal resolution

$$v(t + \Delta t) = v(t) - \frac{\Delta t}{\tau}(v(t) - V_{leak}) + \frac{\Delta t}{\tau} RI_{ext}(t)$$

By setting v (0) and I_{ext} (t), we can calculate v(Δ t), v(2 Δ t), ...

Explicit Euler method

Integrate-and-fire neuron

$$v(t) > \theta \Rightarrow \text{Spike (t)} = 1\,,\, v(t) \leftarrow V_{\text{reset}}\,,$$

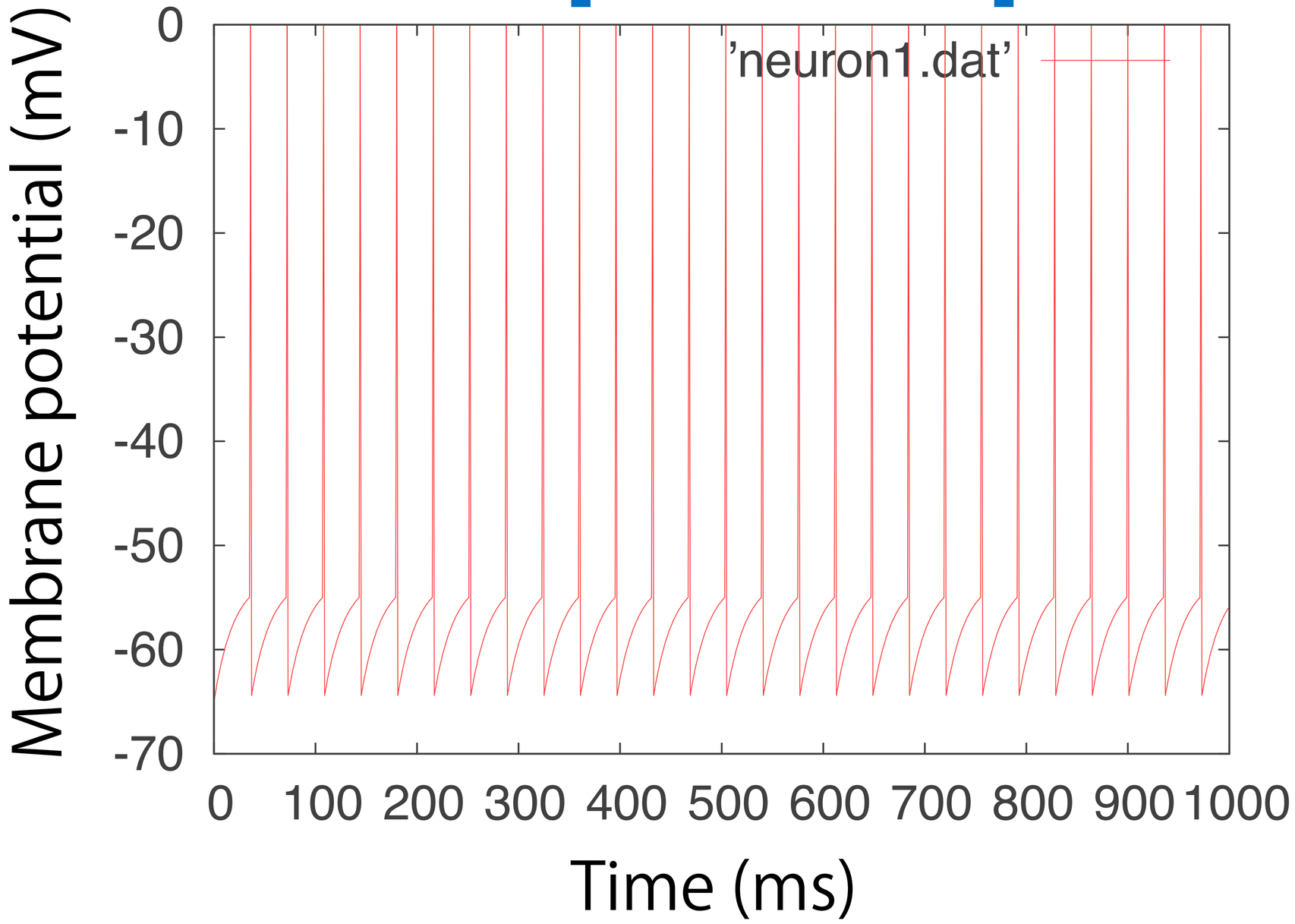
θ : Threshold

V_{reset}: Reset potential

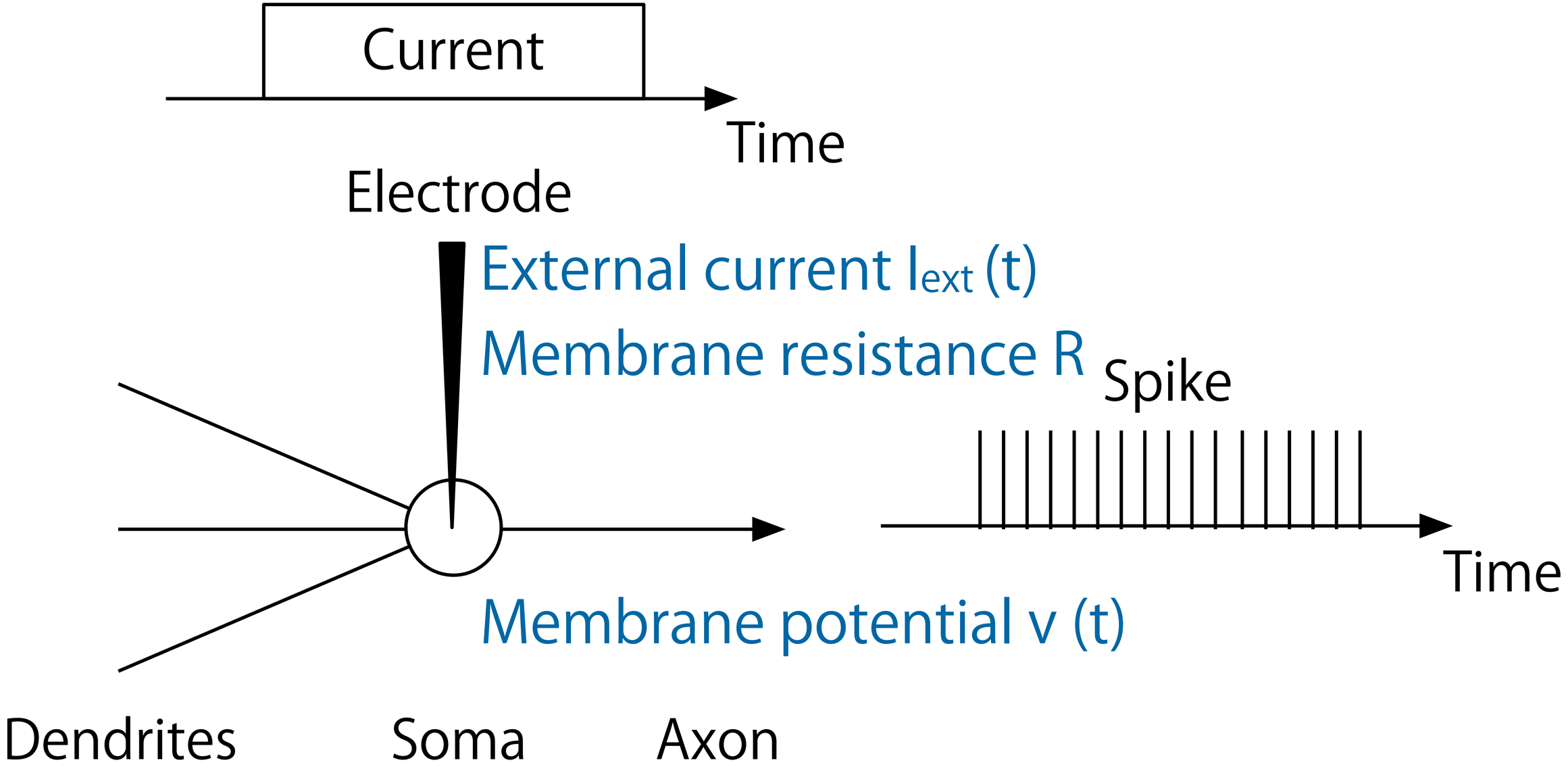
Integrate-and-fire neuron

$$v(0) = V_{\text{init}}.$$ Initial value

Example output



Case #1: 1 neuron



neuron1.c

```
1. #include <stdio.h>
2.
3. #define TAU 20.0
4. #define V_LEAK -65.0
5. #define V_INIT (V_LEAK)
6. #define V_RESET (V_LEAK)
7. #define THETA -55.0
8. #define R 1.0
9. #define DT 1.0
10.#define T 1000.0
11.#define I_EXT 12.0
12.
13.void loop ( void )
14.{
15.  double t = 0;
16.  double v = V_INIT;
17.  int spike = 0;
18.   while ( t < T ) {
19.    printf ( "%f %f\n", t, (spike ? 0.0 : v ) );
20.    double dv = ( DT / TAU ) * ( - ( v - V_LEAK ) + R * I_EXT );
21.    spike = ( v > THETA ) ? 1 : 0;
22.    v = ( v > THETA ) ? V_RESET : v + dv;
23.    t = t + DT;
24.  }
25.}
26.
27.int main ( void )
28.{
29.  loop ( );
30.  return 0;
31.}
```

Explain

Now, it's time to login

Follow the instruction in *ODE, Single neuron, Network* on the paper

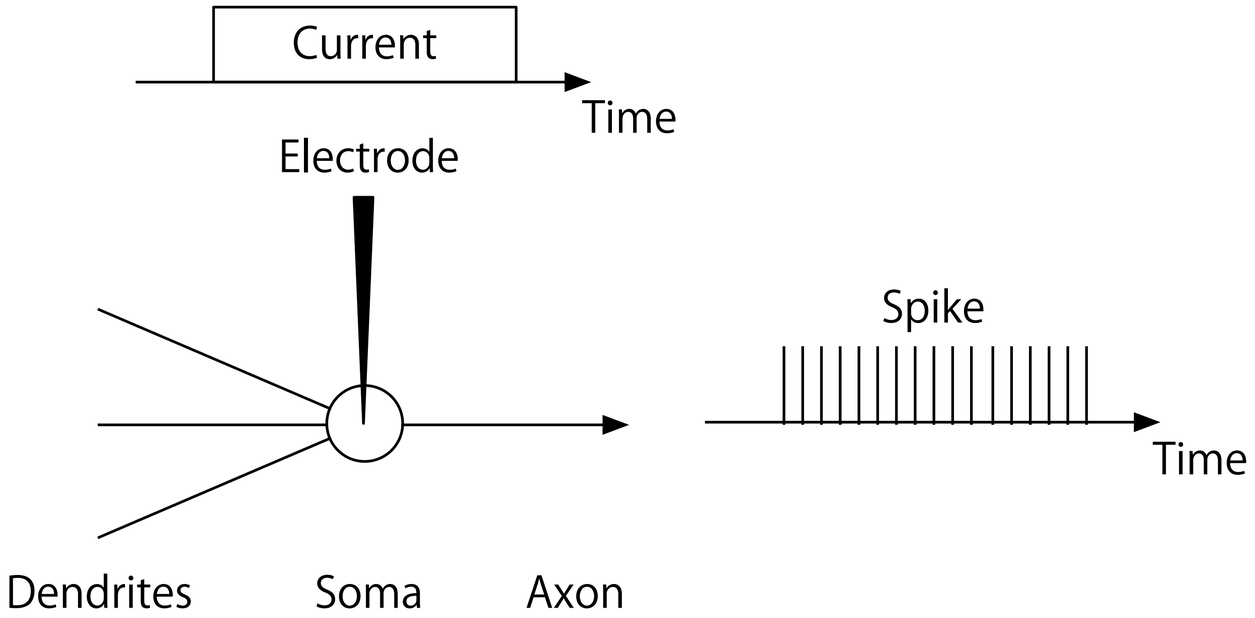
Note:
Need to ssh twice: one to the frontend,
and the other to the compute node

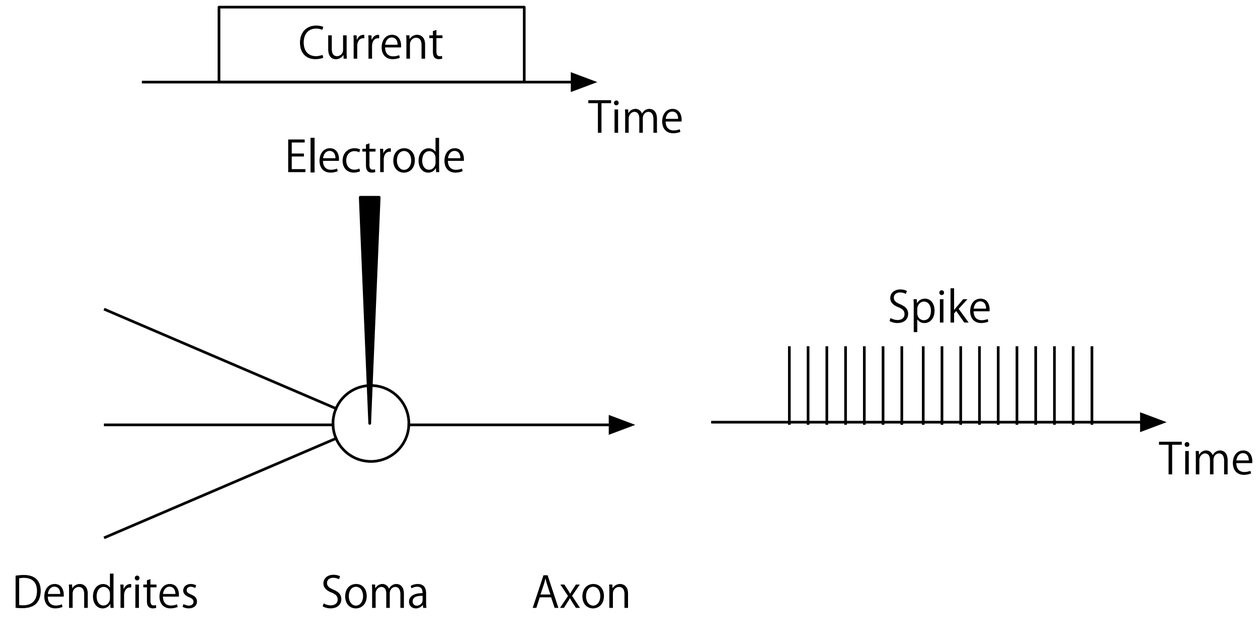
Let's try the code!

- `gcc -std=c99 -O3 -o neuron1 neuron1.c`
- `./neuron1`

  Confirm numbers are printed on screen
- `./neuron1 > neuron1.dat`
- `gnuplot`
- `plot 'neuron1.dat' with line`
- [Exercise] Change the amplitude of the external current to 10 nA or 15 nA, and observe the change of the number of emitted spikes.

Case #2: 2 neurons





neuron2.c (Excerpt)

```
1.void loop ( void )
2.{
3.  double t = 0;
4.  double v [ 2 ] = { V_INIT, V_INIT - 10.0 };
5.  int spike [ 2 ] = { 0, 0 };
6.   while ( t < T ) {
7.    printf ( "%f %f %f\n", t, ( spike [ 0 ] ? 0.0 : v [ 0 ] ),
8.                             ( spike [ 1 ] ? 0.0 : v [ 1 ] ) );
9.    double dv [ 2 ] = { 0.0, 0.0 };
10.    for ( int i = 0; i < 2; i++ ) {
11.      dv [ i ] = ( DT / TAU ) * ( - ( v [ i ] - V_LEAK ) + R * I_EXT );
12.    }
13.    for ( int i = 0; i < 2; i++ ) {
14.      spike [ i ] = ( v [ i ] > THETA ) ? 1 : 0;
15.      v [ i ] = ( v [ i ] > THETA ) ? V_RESET : v [ i ] + dv [ i ];
16.    }
17.    t = t + DT;
18.  }
19.}
```

Explain

Let's try the code!

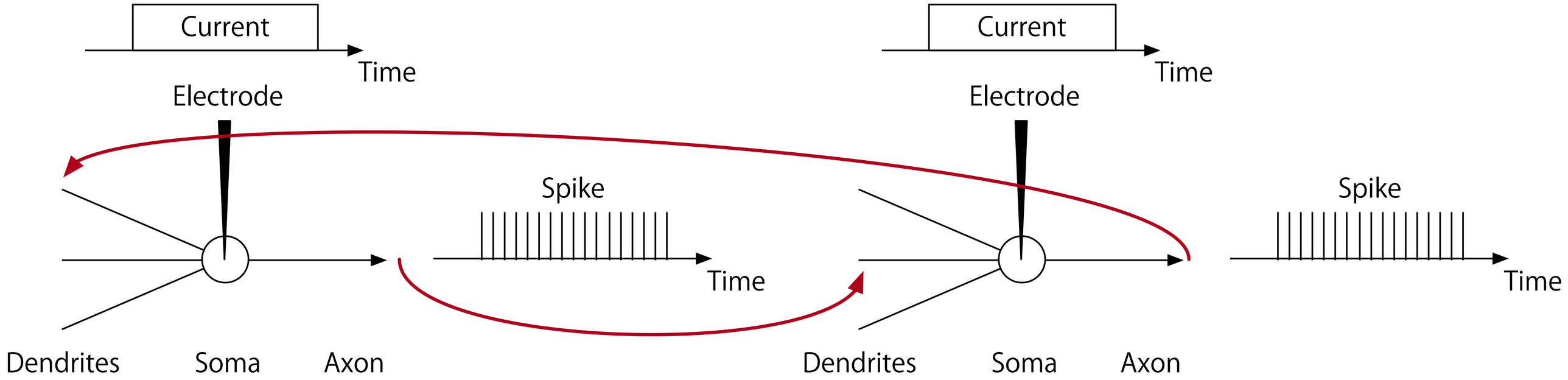
- `gcc -std=c99 -O3 -o neuron2 neuron2.c`
- `./neuron2`

  Confirm numbers are printed on screen

- `./neuron2 > neuron2.dat`
- `gnuplot`
- `plot 'neuron2.dat' using 1:2 with line,`

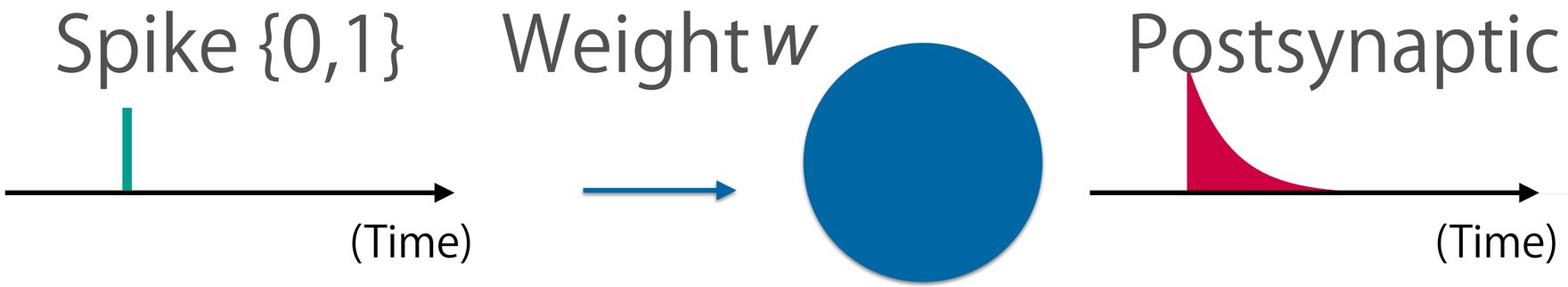
  `'neuron2.dat' using 1:3 with line`

Case #3: A network of 2 neurons



2 neurons are connected mutually

Exponential synapse



$$\tau_{\text{syn}} \frac{dg}{dt} = -g(t) + w \cdot \text{Spike}(t)$$

network2.c (Excerpt)

```
1.void loop ( void )
2.{
3.   double t = 0;
4.   double g [ 2 ] = { 0.0, 0.0 };
5.   double v [ 2 ] = { V_INIT, V_INIT - 10.0 };
6.   int spike [ 2 ] = { 0, 0 };
7.
8.   while ( t < T ) {
9.     printf ( "%f %f %f\n", t, ( spike [ 0 ] ? 0.0 : v [ 0 ] ),
10.                             ( spike [ 1 ] ? 0.0 : v [ 1 ] ) );
11.    double dg [ 2 ] = { 0.0, 0.0 };
12.    double dv [ 2 ] = { 0.0, 0.0 };
13.    for ( int i = 0; i < 2; i++ ) {
14.      dg [ i ] = ( DT / TAU_SYN ) * ( - g [ i ] + W * spike [ ( i + 1 ) % 2 ] );
15.      dv [ i ] = ( DT / TAU ) * ( - ( v [ i ] - V_LEAK ) + g [ i ] + R * I_EXT );
16.    }
17.    for ( int i = 0; i < 2; i++ ) {
18.      spike [ i ] = ( v [ i ] > THETA ) ? 1 : 0;
19.      g [ i ] = g [ i ] + dg [ i ];
20.      v [ i ] = ( v [ i ] > THETA ) ? V_RESET : v [ i ] + dv [ i ];
21.    }
22.    t = t + DT;
23.  }
24.}
```

Explain

Let's try the code!

- `gcc -std=c99 -O3 -o network2 network2.c`
- `./network2`

  Confirm numbers are printed on screen

- `./network2 > network2.dat`
- `gnuplot`
- `plot 'network2.dat' using 1:2 with line,`

  `'network2.dat' using 1:3 with line`

- **[Exercise]** Change the sign of `w` negative and try again

Wrap-up

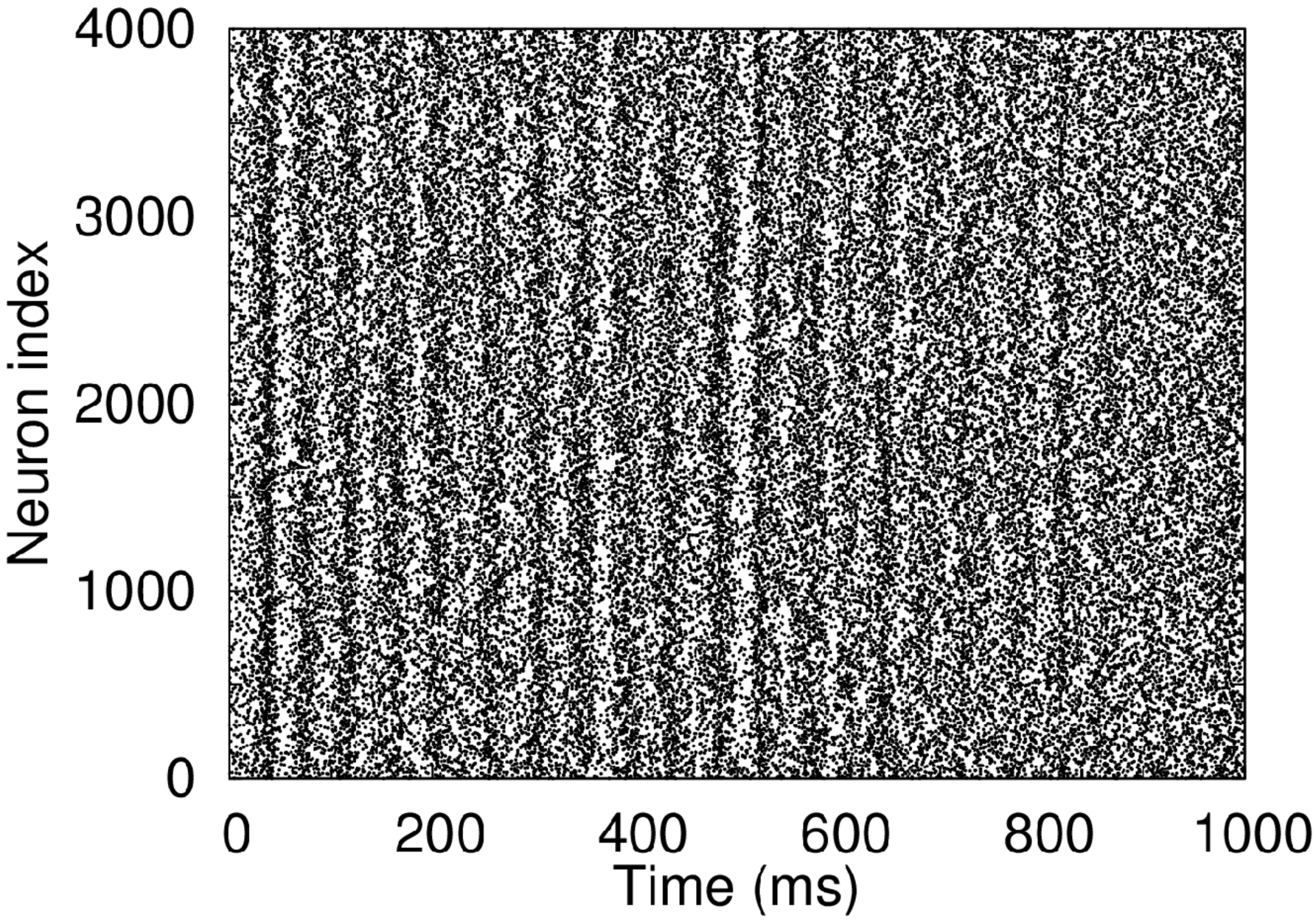
- Learned how to code 1 neuron, 2 neurons, and a network of 2 neurons
- Observed a change of the network dynamics by changing the sign of connection weights
- Connectivity matters in the brain

Break 5 min

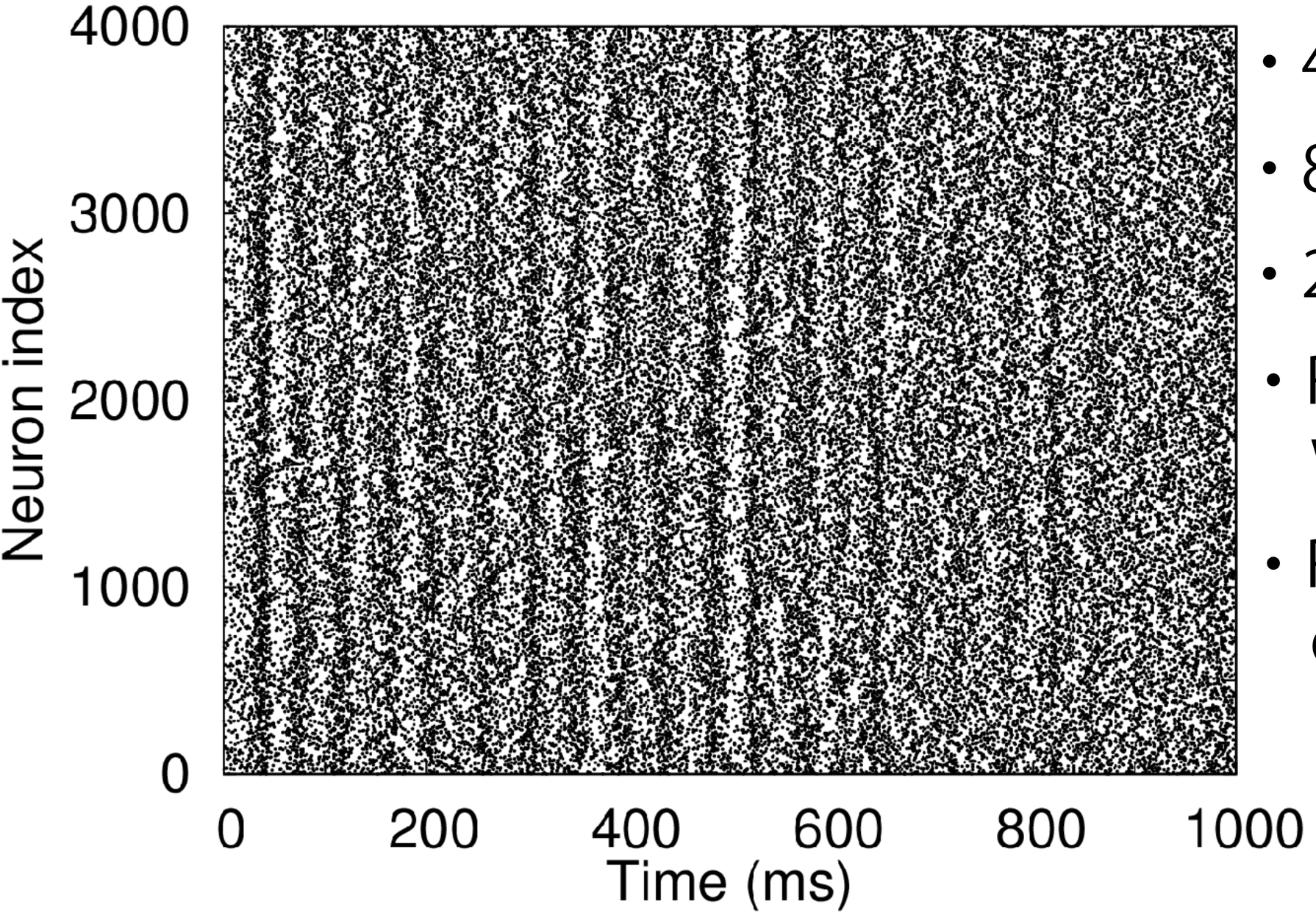
Lecture #2

OpenMP and MPI

Balanced network model



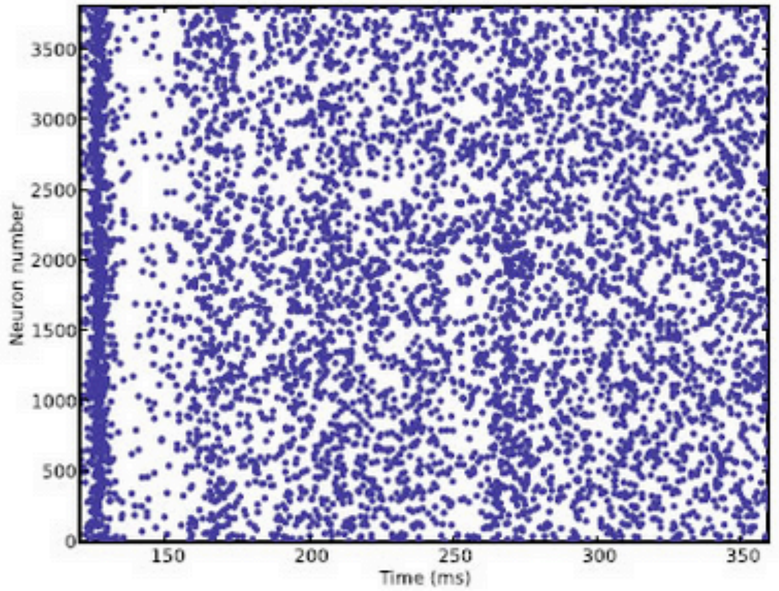
Balanced network model



- 4000 neurons
- 80% Excitatory
- 20% Inhibitory
- Random connection with $p = 0.02$
- Random initial value of v

In the case of BRIAN simulator

```
1  from brian2 import *
2  eqs = '''
3  dv/dt  = (ge+gi-(v+49*mV))/(20*ms) : volt
4  dge/dt = -ge/(5*ms)                : volt
5  dgi/dt = -gi/(10*ms)               : volt
6  '''
7  P = NeuronGroup(4000, eqs, threshold='v>-50*mV', reset='v=-60*mV')
8  P.v = -60*mV
9  Pe = P[:3200]
10 Pi = P[3200:]
11 Ce = Synapses(Pe, P, on_pre='ge+=1.62*mV')
12 Ce.connect(p=0.02)
13 Ci = Synapses(Pi, P, on_pre='gi-=9*mV')
14 Ci.connect(p=0.02)
15 M = SpikeMonitor(P)
16 run(1*second)
17 plot(M.t/ms, M.i, '.')
18 show()
```



Compact and efficient!

random.c (Excerpt)

```
1.void loop ( void )
2.{
3.   double t = 0.;
4.   timer_start ();
5.   while ( t < T ) {
6.     for ( int i = 0; i < N; i++ ) {
7.       calculateSynapse ( i );
8.       updateMembranePotential ( i );
9.     }
10.      outputSpike ( t );
11.      t = t + DT;
12.   }
13.   double elapsedTime = timer_elapsed ();
14.   printf ( "Elapsed time = %f sec.\n", elapsedTime);
15.}
16.
```

# of neurons (i.e., 4000)

Explain

Now, it's time to login

Follow the instruction in *OpenMP* section on the paper

Note:
Need to ssh twice: one to the frontend,
and the other to the compute node

Let's try the code!

- **`make random`**

  The code is compiled automatically

- **`./random`**

- **`spike.dat`** is generated

- **`gnuplot`**

- **`plot 'spike.dat' with dots`**

  Measure the computational time

How slow the simulation is!

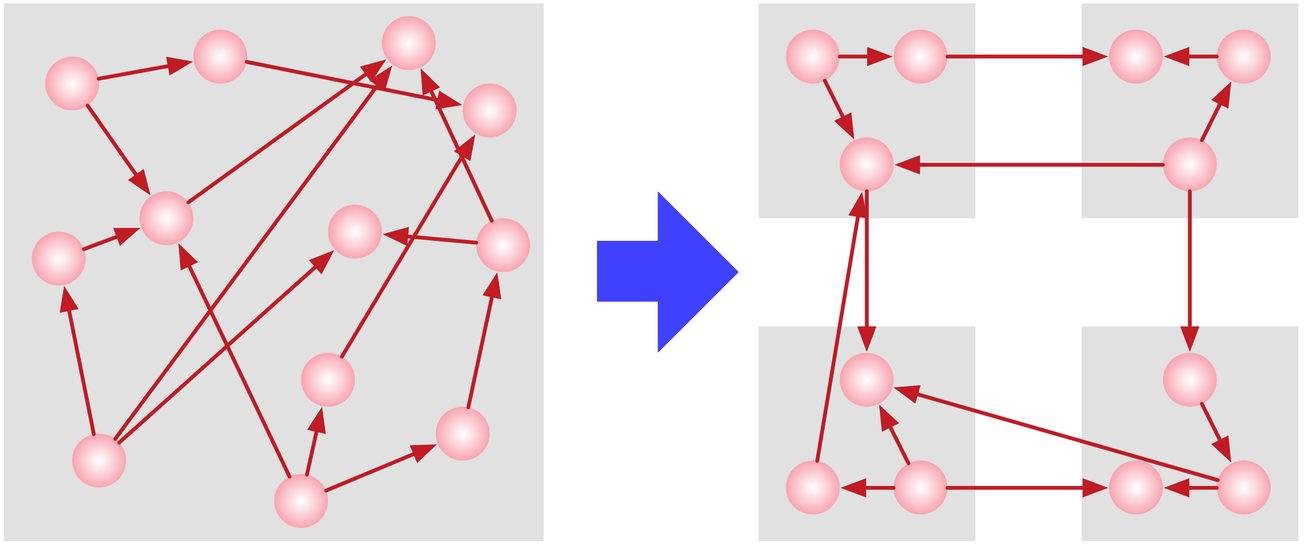
How to speed up?

- So far, we have used only 1 CPU core
- Actually, 1 CPU in our cluster consists of 12 cores
- The cluster machine has 16 CPUs (= 192 cores)
- Let's use them all ! (→ Parallel computing)

Two parallelization strategies

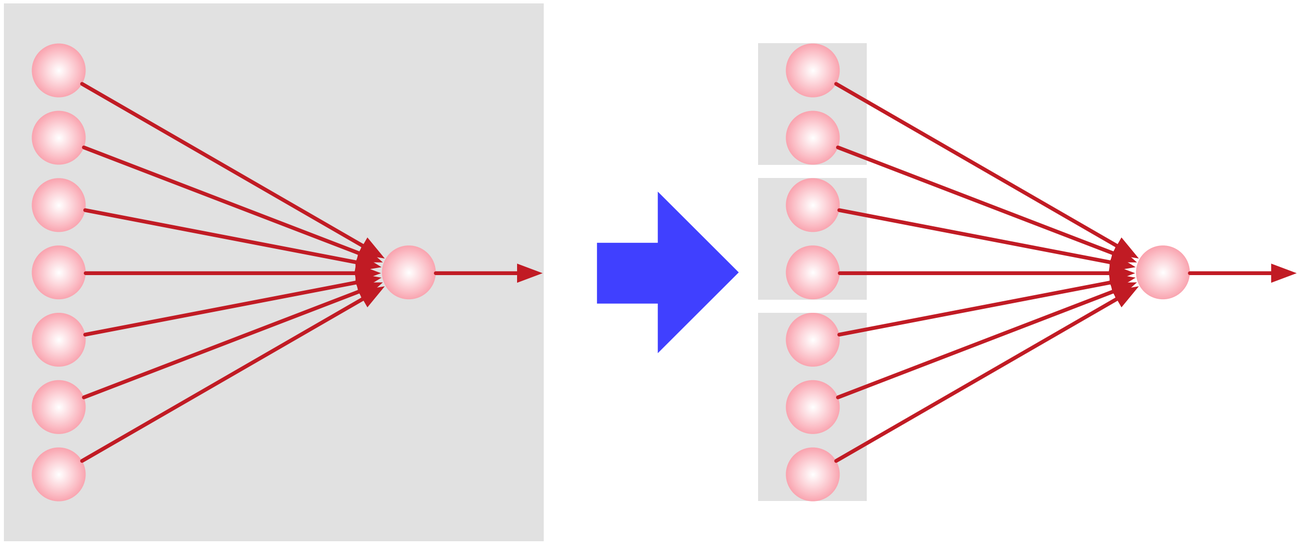
Calculate neurons in parallel

(We do this today)



Calculate synaptic inputs in parallel

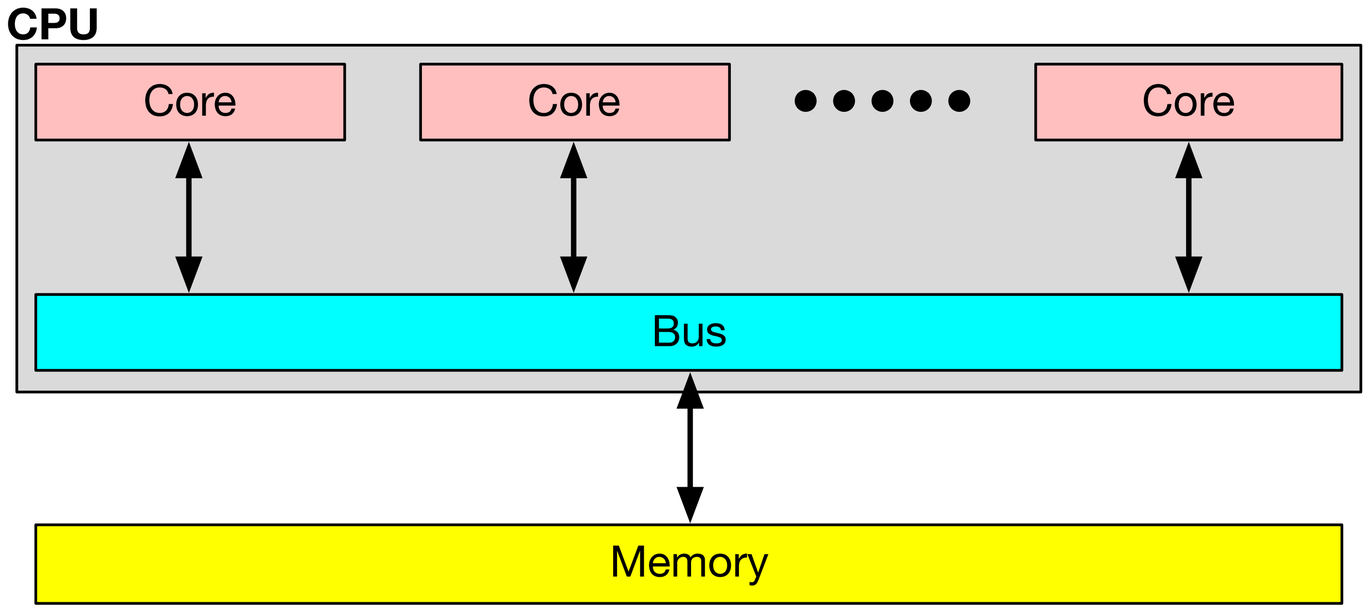
(We don't do this today)



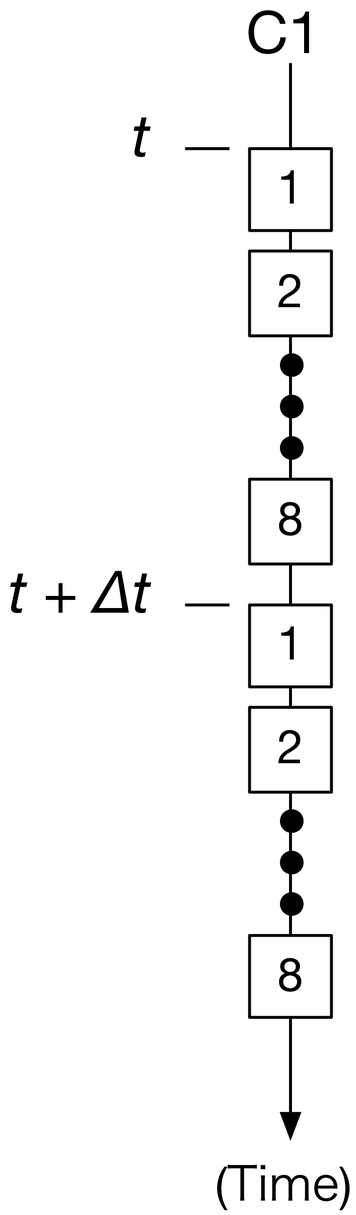
Method #1: OpenMP

(Application Programming Interface)

- OpenMP = APIs to use multicores on a single CPU
- The cores share the same memory address



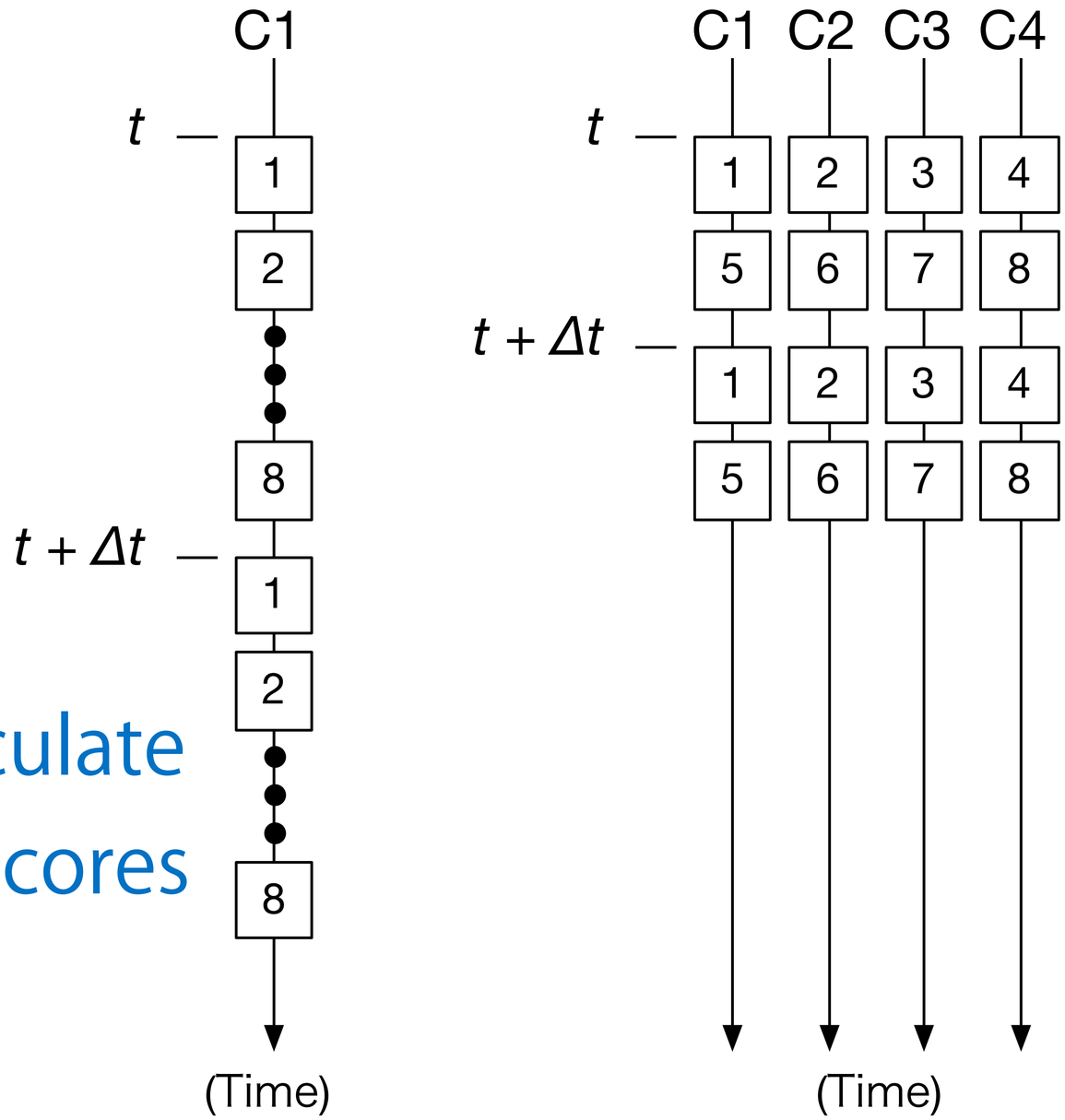
Example



Schematic to calculate

8 neurons with 1 cores

Example



Schematic to calculate

8 neurons with 4 cores

random_omp.c
(Excerpt)

```
1.void loop ( void )
2.{
3.   double t = 0.;
4.   timer_start ();
5.   while ( t < T ) {
6.#pragma omp parallel for
7.      for ( int i = 0; i < N; i++ ) {
8.        calculateSynapse ( i );
9.        updateMembranePotential ( i );
10.     }
11.     outputSpike ( t );
12.     t = t + DT;
13.  }
14.  double elapsedTime = timer_elapsed ();
15.  printf ( "Elapsed time = %f sec.\n", elapsedTime);
16.}
17.
```

Explain

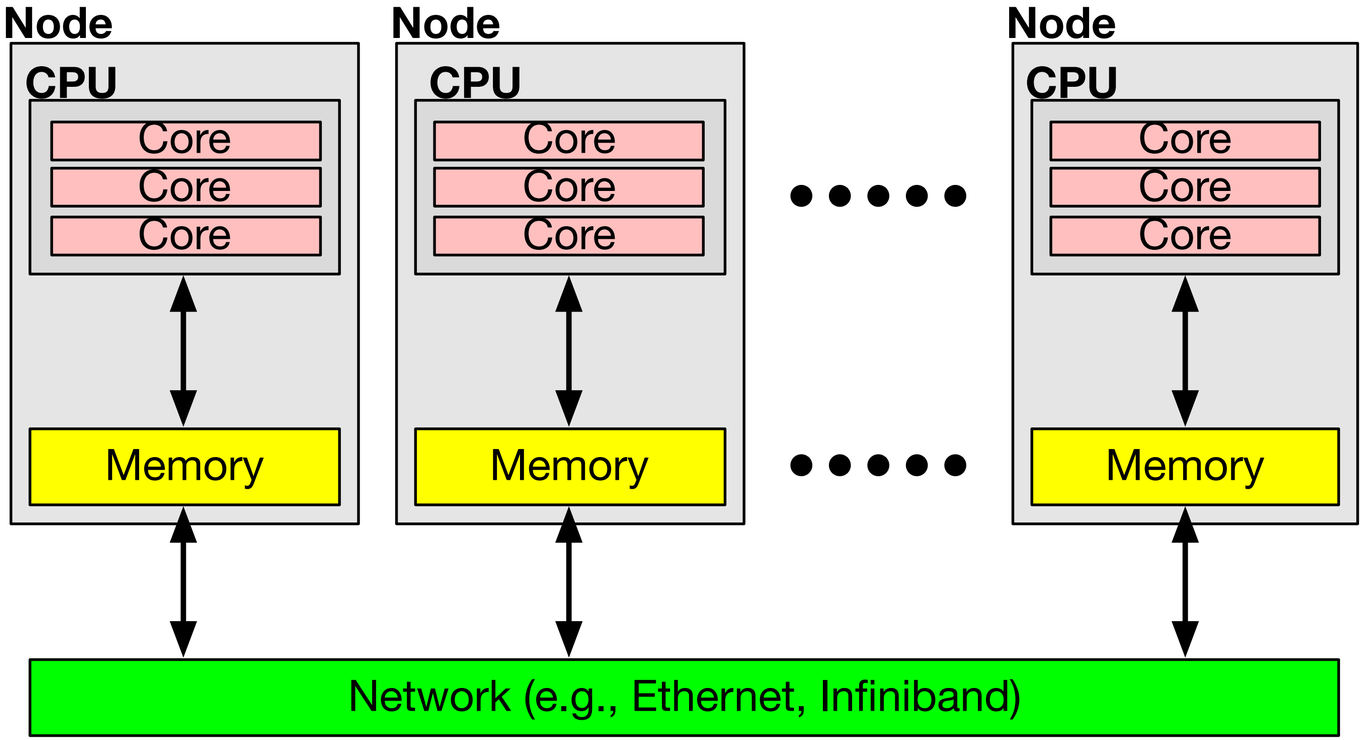
Let's try the code!

- `make random_omp`
- `./omp`
- Measure the computational time

Now, please logout from the compute node

Method #2: MPI

- MPI (Message Passing Interface)
- Multiple compute nodes with individual memory
- Need to write a code to send / receive data across nodes explicitly

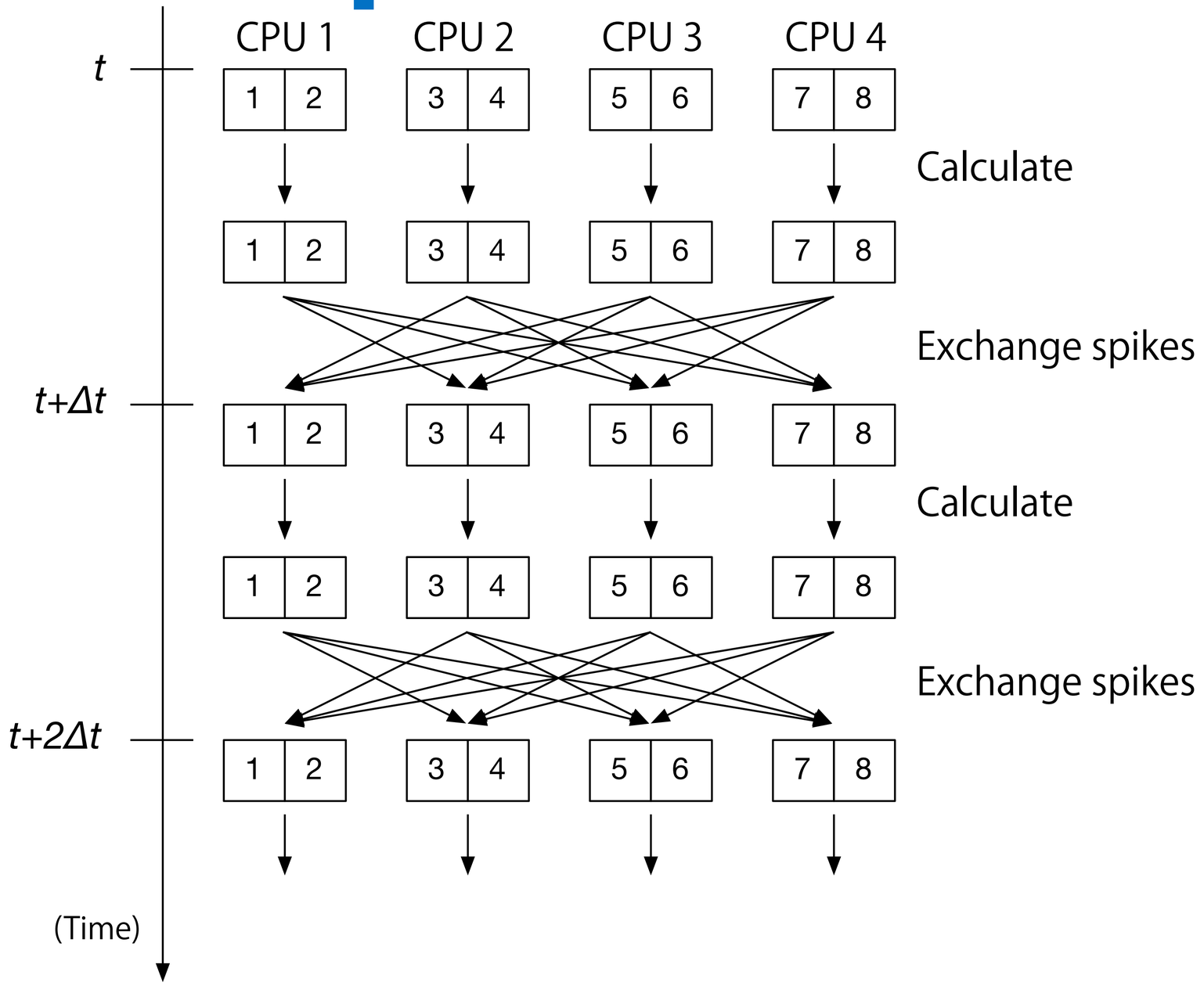


Example

1	2	3	4	5	6	7	8

Calculate 8 neurons with 4 CPUs

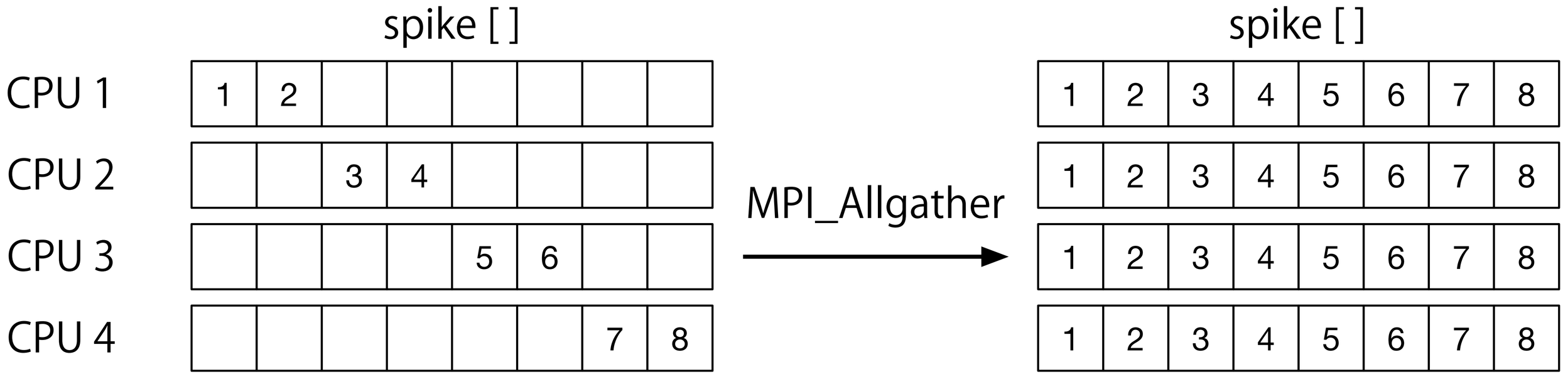
Schematic to calculate 8 neurons with 4 CPUs



Exchanging spikes via MPI

(i.e., spikes)

- Transferring data across CPUs via the network
- MPI (Message Passing Interface) is a library
- Today, we will use **`MPI_Allgather`**



random_mpi.c

(Excerpt)

```
1.void loop ( const int mpi_size, const int mpi_rank )
2.{
3.   const int n_each = ( N + mpi_size - 1 ) / mpi_size;
4.   int spike_local [ n_each ];
5.   double t = 0.;
6.   timer_start ();
7.   while ( t < T ) {
8.     for ( int n = 0; n < n_each; n++ ) {
9.        calculateSynapse ( n, n_each * mpi_rank );
10.       updateMembranePotential ( n, n_each * mpi_rank, spike_local );
11.    }
12.    MPI_Allgather ( spike_local, n_each, MPI_INT, spike, n_each,
13.                    MPI_INT, MPI_COMM_WORLD );
14.    if ( mpi_rank == 0 ) { outputSpike ( t ); }
15.    t = t + DT;
16.  }
17.  double elapsedTime = timer_elapsed ();
18.  if ( mpi_rank == 0 ) { printf ( "Elapsed time = %f sec.\n", elapsedTime); }
19.}
```

4000 / # cores

(if we have 100 cores, then n_each = 40)

(if we have 100 cores, then n_each = 40)

(0 ... n_each)

random_mpi.c

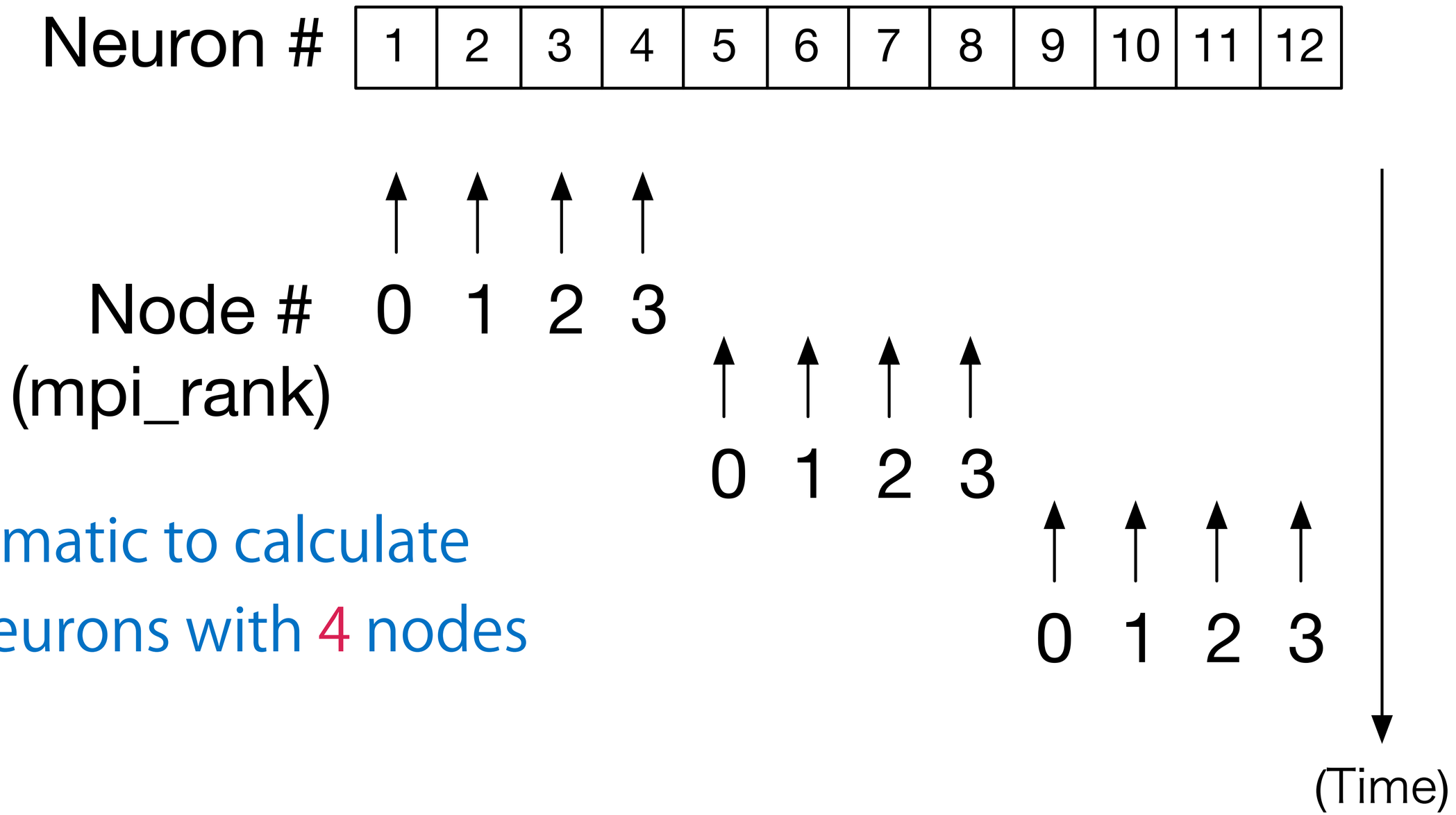
(Excerpt)

```
1. void updateMembranePotential ( const int n, const int offset, int spike_local
   [] )
2. {
3.   int i = n + offset;

5.   if ( i < N ) {
6.     double dv = DT * ( - ( v [ i ] - V_LEAK ) + ge [ i ] + gi [ i ] ) / TAU_M;
7.     spike_local [ n ] = ( v [ i ] > THETA ) ? 1 : 0;
8.     v [ i ] = ( v [ i ] > THETA ) ? V_RESET : v [ i ] + dv;
9.   }
10.}
11.
```

n_each * mpi_rank

How the computation goes



Schematic to calculate

12 neurons with 4 nodes

Now, it's time to login

Follow the instruction in *MPI* section on the paper

Note:
If you are already in the frontend, stay there

Let's try the code!

- `make random_mpi`
- `mpirun -hostfile hostfile -np 16 ./random_mpi`
- Measure the computational time
- Repeat while changing the number after `-np`

  (1, 2, 4, 8, 16, 32, 64, 128, 160, 192 should be tested)

How did the computational time change?

- If the time became half by doubling the # of CPUs, the parallelization is ideal (→ strong scaling)

- Performance increase by parallelization is constrained by a block in a code which can be executed only serially (→ Amdahl's law)

Wrap-up

- Simulated balanced network with 4000 neurons and sparse random connections
- Accelerated the simulation by OpenMP and MPI
- Good strong scaling was observed in MPI

Now, please logout from the cluster

Lunch time

Please come back by 14:30!

Lecture #3

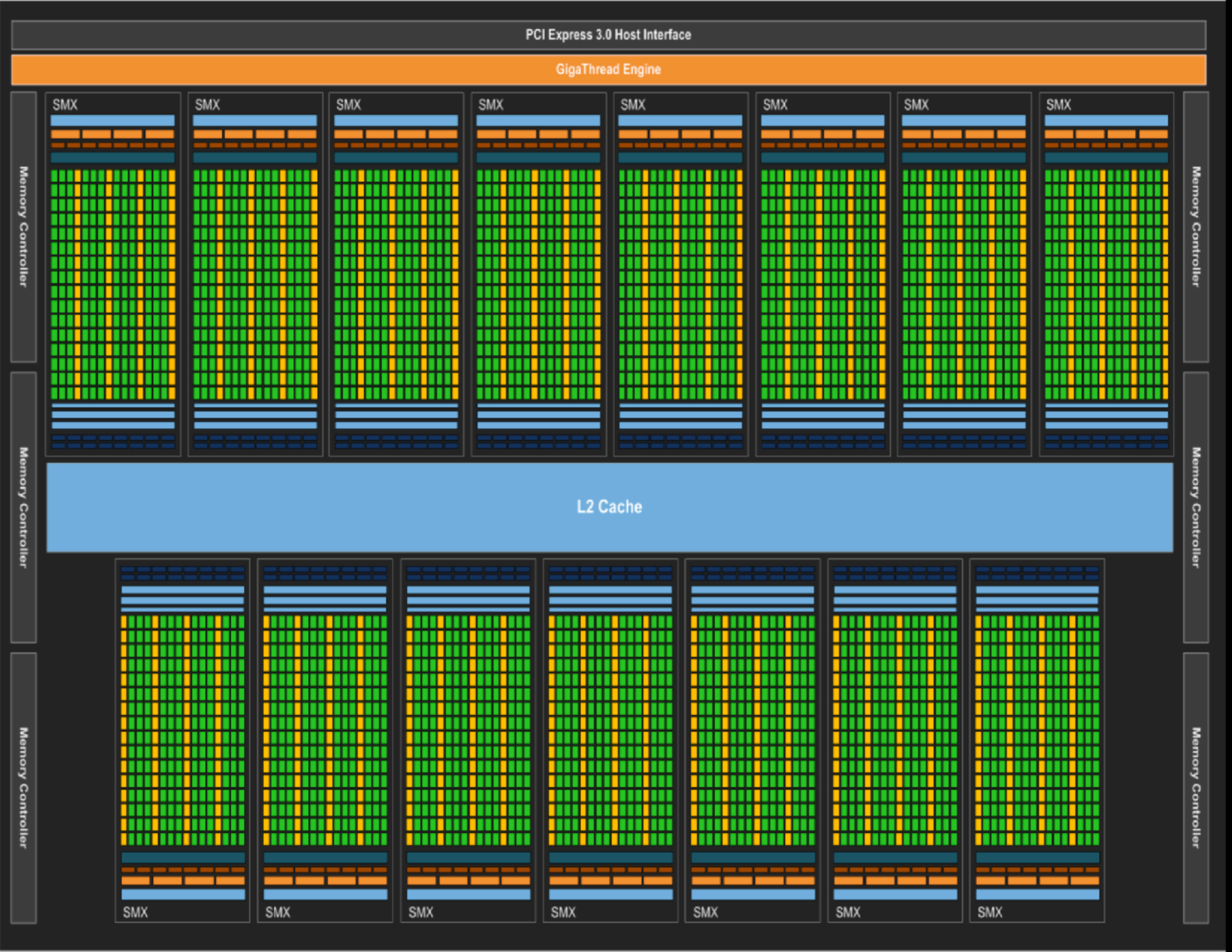
GPU with CUDA

GPU (Graphics Processing Unit)

- Special-purpose hardware for computer graphics
- Nevertheless, we can use it for computing

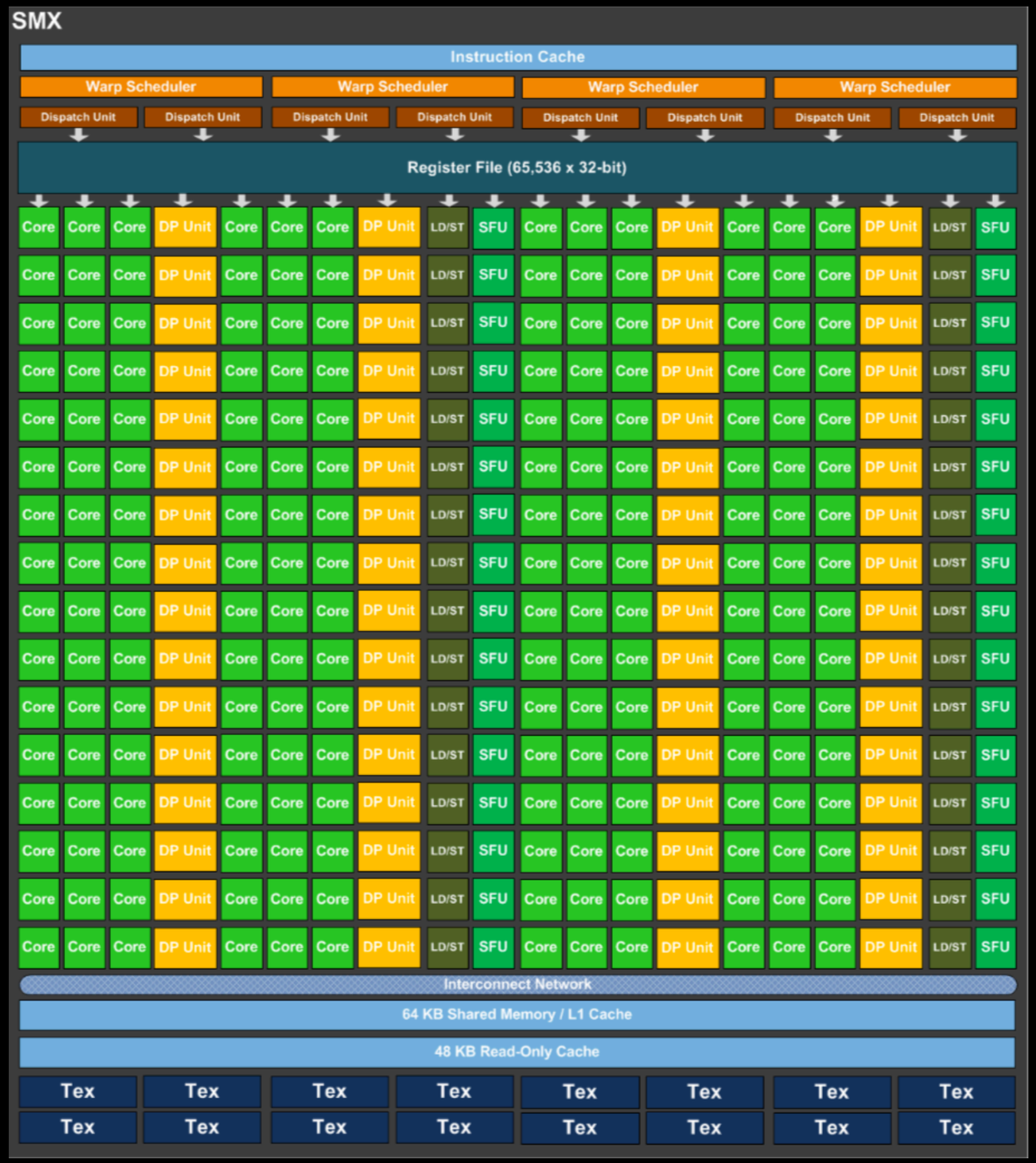
NVIDIA Tesla K40

- CUDA (Compute Unified Device Architecture) is a library for NVIDIA GPUs



K40 (GK110)

From GK110 White Pater



15 SMX (Streaming Multiprocessor)

192 CUDA Cores / SMX

Total 2880 CUDA Cores

From GK110 White Pater

How to write a code with CUDA

```
1.void loop ( void )
2.{
3.   double t = 0.;
4.   while ( t < T ) {
5.     for ( int i = 0; i < N; i++ ) {
6.       calculateSynapse ( i );
7.       updateMembranePotential ( i );
8.     }
9.     outputSpike ( nt );
10.    t += DT;
11.  }
12.}
```

```
1.void loop ( void )
2.{
3.   double t = 0.;
4.   while ( t < T ) {
5.     kernel <<< GS, BS >>> ( );
6.     outputSpike ( nt );
7.    t += DT;
8.   }
9.}
```

```
1.__global__ void kernel ( void )
2.{
3.   int i = threadIdx.x + BlockIdx.x * BlockDim.x;
4.   if ( i < N ) {
5.     calculateSynapse ( i );
6.     updateMembranePotential ( i );
7.   }
8.}
```

Thread, block, and grid (1/2)

Grid size

Block size

```
kernel <<< GS, BS >>> ( );
```

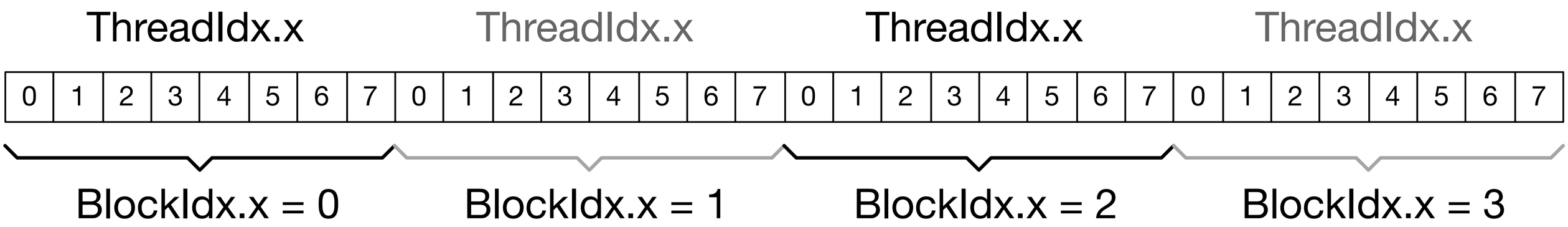
CUDA notation

Thread = A compute entity

Block = A set of threads

Grid = A set of blocks

Example: (GS, BS) = (4, 8)



4 x 8 = 32 threads will be issued

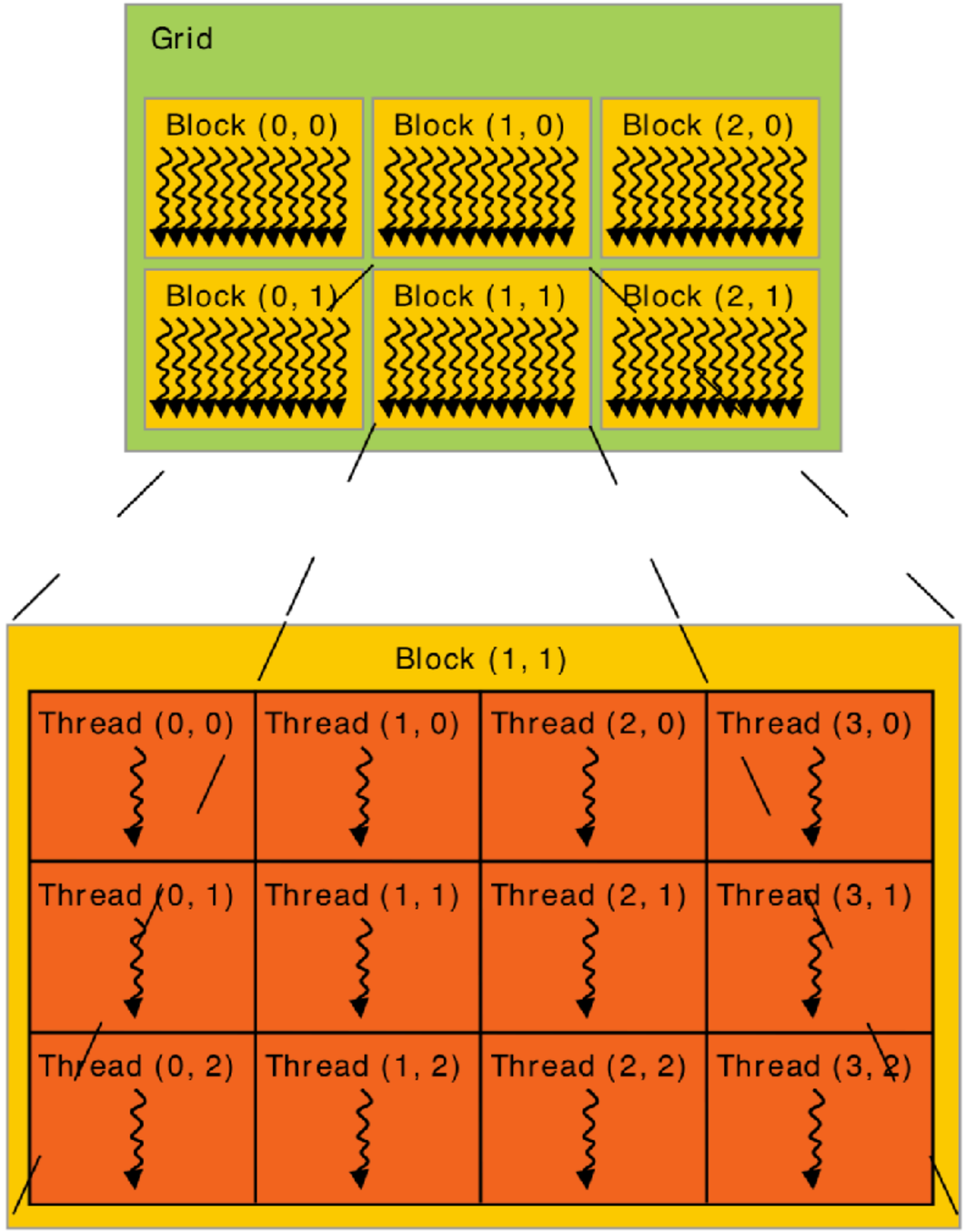


Figure 6 Grid of Thread Blocks

From CUDA C Programming Guide

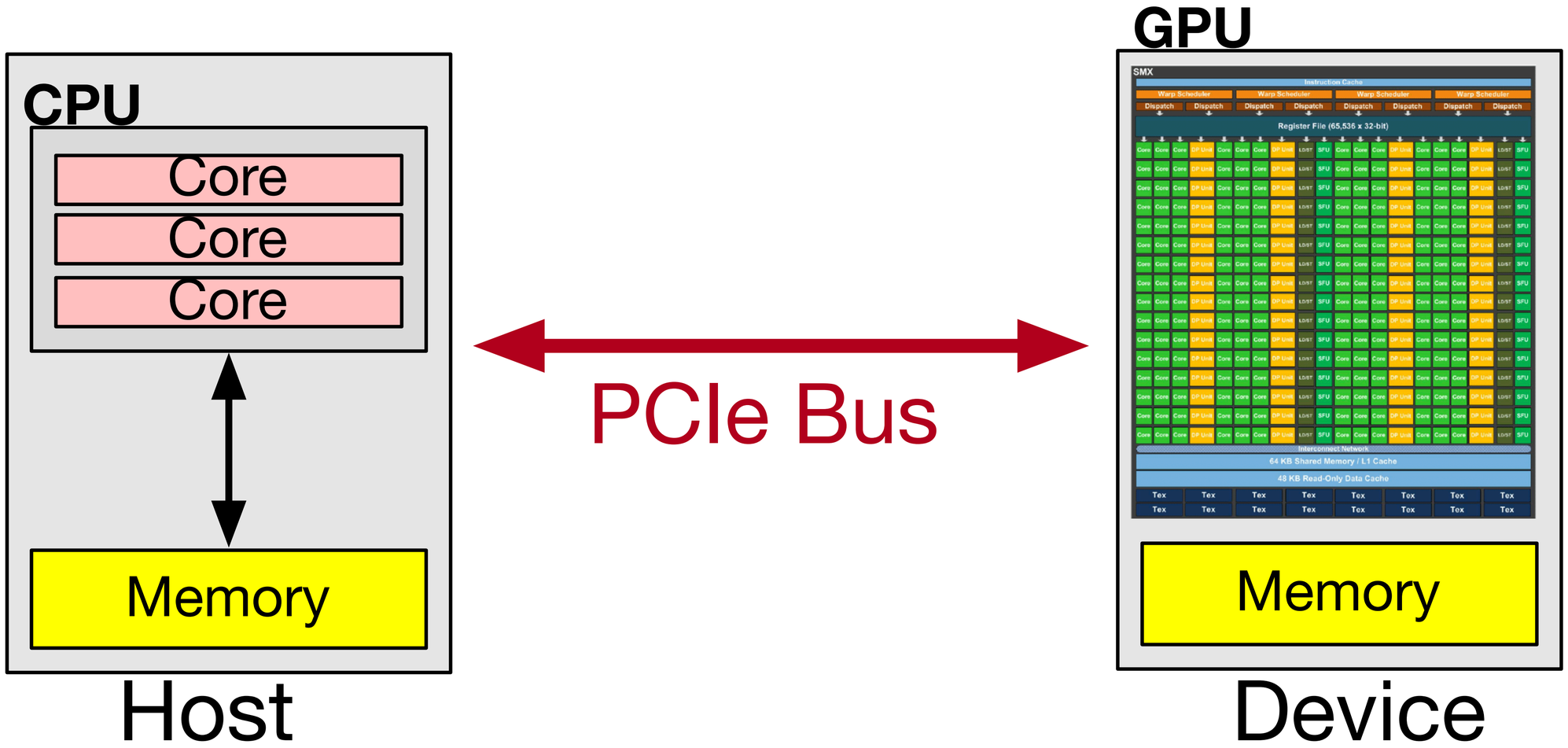
Thread, block, and grid (2/2)

So, to compute neurons in parallel, it should be

$$GS * BS \geqq \#\ \text{of neurons}\ (= N)$$

- BS must be a power of 2 (i.e., 2, 4, 8, ...) and < 1,024 (in the case of K40)
- Grids and blocks can be aligned in 2D or 3D

Heterogeneous architecture



We must manage data transfer between CPU and GPU explicitly

(Note: Unified memory automatizes this process)

Now, it's time to login

Follow the instruction in *GPU* on the paper

Note:
Need to ssh twice: one to the frontend,
and the other to the compute node

Let's take a look at the code (1/8)

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <math.h>
4. #include <SFMT.h>

6. #define N 4000
7. #define N_EXC 3200
8. #define N_INH ( ( N ) - ( N_EXC ) )

10.#define T 1000.
11.#define DT 1.

13.#define TAU_M 20.
14.#define TAU_GE 5.
15.#define TAU_GI 10.

17.#define V_LEAK -49.
18.#define V_INIT -60.
19.#define V_RESET -60.
20.#define THETA -50.

22.#define G_EXC 1.62
23.#define G_INH -9.
24.#define P 0.02
```

```
26.#define BLOCK_SIZE 32

28.extern "C" { void sfmt_init_gen_rand ( sfmt_t * sfmt, u
29.extern "C" { double sfmt_genrand_real2 ( sfmt_t * sfmt
30.extern "C" { void timer_start ( void ); }
31.extern "C" { double timer_elapsed ( void ); }

33.static double v [ N ], ge [ N ], gi [ N ];
34.static int *w_exc, *w_inh, spike [ N ];
35.static double *d_v, *d_ge, *d_gi;
36.static int *d_w_exc, *d_w_inh, *d_spike;

38.static FILE *file_spike;
```

Let's take a look at the code (2/8)

```
40.void initialize ( void )
41.{
42.  // PRNG
43.  sfmt_t rng;
44.  sfmt_init_gen_rand ( &rng,  );

46.  // Output file
47.  file_spike = fopen ( "spike.dat", "w" );

49.  // Cell parameters
50.  for ( int i = 0; i < N; i++ ) {
51.    v [ i ] = V_INIT + 10. * sfmt_genrand_real2 ( &rng );
52.    ge [ i ] = 0.;
53.    gi [ i ] = 0.;
54.    spike [ i ] = 0;
55.  }

57.  cudaMalloc ( &d_v, N * sizeof ( double ) );
58.  cudaMalloc ( &d_ge, N * sizeof ( double ) );
59.  cudaMalloc ( &d_gi, N * sizeof ( double ) );
60.  cudaMalloc ( &d_spike, N * sizeof ( int ) );

62.  cudaMemcpy ( d_v, v, N * sizeof ( double ), cudaMemcpyHostToDevice );
63.  cudaMemcpy ( d_ge, ge, N * sizeof ( double ), cudaMemcpyHostToDevice );
64.  cudaMemcpy ( d_gi, gi, N * sizeof ( double ), cudaMemcpyHostToDevice );
65.  cudaMemcpy ( d_spike, spike, N * sizeof ( int ), cudaMemcpyHostToDevice );
```

Allocate device memory

Copy host memory to device memory

Let's take a look at the code (3/8)

```
67.   // Synaptic weights
68.   w_exc = (int *) malloc ( N * N * sizeof ( int ) );
69.   w_inh = (int *) malloc ( N * N * sizeof ( int ) );

71.   for ( int i = 0; i < N; i++ ) {
72.     // From excitatory neurons to other neurons
73.     for ( int j = 0; j < N_EXC; j++ ) {
74.       w_exc [ j + N * i ] = ( sfmt_genrand_real2 ( &rng ) < P ) ? 1 : 0;
75.     }
76.     // From inhibitory neurons to other neurons
77.     for ( int j = N_EXC; j < N_EXC + N_INH; j++ ) {
78.       w_inh [ j + N * i ] = ( smft_genrand_real2 ( &rng ) < P ) ? 1 : 0;
79.     }
80.   }
81.   cudaMalloc ( &d_w_exc, N * N * sizeof ( int ) );
82.   cudaMalloc ( &d_w_inh, N * N * sizeof ( int ) );
83.   cudaMemcpy ( d_w_exc, w_exc, N * N * sizeof ( int ), cudaMemcpyHostToDevice );
84.   cudaMemcpy ( d_w_inh, w_inh, N * N * sizeof ( int ), cudaMemcpyHostToDevice );
85.}
```

Allocate device memory

Copy host memory to device memory

Let's take a look at the code (4/8)

```
87.void finalize ( void )
88.{
89.   fclose ( file_spike );
90.   free ( w_exc );
91.   free ( w_inh );
92.   cudaFree ( d_v );
93.   cudaFree ( d_ge );
94.   cudaFree ( d_gi );
95.   cudaFree ( d_spike );
96.   cudaFree ( d_w_exc );
97.   cudaFree ( d_w_inh );
98.}
```

Let's take a look at the code (5/8)

Declare the function is called from device (i.e., GPU)

```
100.__device__ void calculateSynapse ( int i, double *ge, double *gi,
101.                                   int *w_exc, int *w_inh, int *spike )
102.{
103.  if ( i < N ) // this condition is unnecessary, but just in case.
104.  {
105.    double r = 0.;
106.    for ( int j = 0; j < N_EXC; j++ ){
107.      r += w_exc [ j + N * i ] * spike [ j ];
108.    }
109.    ge [ i ] += DT * ( G_EXC * r - ge [ i ] ) / TAU_GE;

111.    r = 0.;
112.    for ( int j = N_EXC; j < N_EXC + N_INH; j++ ){
113.      r += w_inh [ j + N * i ] * spike [ j ];
114.    }
115.    gi [ i ] += DT * ( G_INH * r - gi [ i ] ) / TAU_GI;
116.  }
117.}
118.
```

Let's take a look at the code (6/8)

Declare the function is called from device (i.e., GPU)

```
118.__device__ void updateMembranePotential ( int i, double *v, double *ge, double *gi, int *spike )
119.{
120.  if ( i < N ) // this condition is unnecessary, but just in case.
121.  {
122.    double dv = DT * ( - ( v [ i ] - V_LEAK ) + ge [ i ] + gi [ i ] ) / TAU_M;
123.    spike [ i ] = ( v [ i ] > THETA ) ? 1 : 0;
124.    v [ i ] = ( v [ i ] > THETA ) ? V_RESET : v [ i ] + dv;
125.  }
126.}
```

Declare the function is called from both host and device

```
128.__global__ void kernel ( double *v, double *ge, double *gi, int *spike, int *w_exc, int *w_inh )
129.{
130.  int i = threadIdx.x + blockIdx.x * blockDim.x;

132.  if ( i < N ) {
133.    calculateSynapse ( i, ge, gi, w_exc, w_inh, spike );
134.    updateMembranePotential ( i, v, ge, gi, spike );
135.  }
136.}
137.
```

Let's take a look at the code (7/8)

```
138.void outputSpike ( const double t )
139.{
140.  for ( int i = 0; i < N; i++ ) {
141.    if ( spike [ i ] ) { fprintf ( file_spike, "%f %d\n", t, i ); }
142.  }
143.}

145.void loop ( void )
146.{
147.  double t = 0.;
148.  timer_start ();

150.  int gridsize = ( N + BLOCK_SIZE - N % BLOCK_SIZE ) / BLOCK_SIZE;
151.  while ( t < T ) {
152.    kernel <<< gridsize, BLOCK_SIZE >>> ( d_v, d_ge, d_gi, d_spike, d_w_exc, d_w_inh );
153.    cudaMemcpy ( spike, d_spike, N * sizeof ( int ), cudaMemcpyDeviceToHost );
154.    outputSpike ( t );
155.    t = t + DT;
156.  }
157.  double elapsedTime = timer_elapsed ();
158.  printf ( "Elapsed time = %f sec.\n", elapsedTime);
159.}
160.
```

Compute grid size

Kernel launch

Transfer spike data from device to host

Let's take a look at the code (8/8)

```
161.int main ( void )
162.{
163.  initialize ();
164.  loop ();
165.  finalize ();

167.  return 0;
168.}
```

Let's try the code!

or "2" or "3" or "4"
Specifies 1 of 4 GPUs

- `make random_gpu`
- `CUDA_VISIBLE_DEVICES="0" ./random_gpu`
- Measure the computational time

Wrap-up

- Heterogeneous architecture

- Memcpy CPU ↔ GPU limits the performance

```
while ( t < T ) {
  kernel <<< N/BLOCK_SIZE, BLOCK_SIZE >>> ( d_v, d_ge, d_gi, d_spike, d_w_exc, d_w_inh );
  cudaMemcpy ( spike, d_spike, N * sizeof ( int ), cudaMemcpyDeviceToHost );
  outputSpike ( t );
  t = t + DT;
}
```

Memcpy for each iteration is problematic

Please logout from the cluster

Break 30 min

Please come back by 16:30

Lecture #4

PEZY-SC2 with OpenCL

What is PEZY-SC2?

- Manycore processor developed by PEZY Computing
- 15,874 threads can be issued in parallel (1,984 x 8)
- 4 TFlops in double, 8 TFlops in float
  (Comparable to NVIDIA P100)
- See Wikichip for further information
  https://en.wikichip.org/wiki/pezy/pezy-scx/pezy-sc2

https://pc.watch.impress.co.jp/docs/news/1091458.html

The PEZY-SC2 cluster we use today

- 256 PEZY-SC2 processors
- 0.5 PFlops in double, 1 PFlops in float
- Liquid immersion cooling system

A larger system Shoubu is #1 in The GREEN 500

Green500 List for June 2019

Listed below are the June 2019 The Green500's energy-efficient supercomputers ranked from 1 to 10.

Note: Shaded entries in the table below mean the power data is derived and not meassured.

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Power Efficiency (GFlops/watts)
1	472	**Shoubu system B** - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. Advanced Center for Computing and Communication, RIKEN Japan	953,280	1,063.3	60	17.604

https://www.top500.org/green500/lists/2019/06/

Now, it's time to login

Follow the instruction in *PEZY-SC2* section on the paper

Note:
Need to ssh twice: one to the frontend,
and the other to the compute node

How to program PEZY-SC2

- PZCL, a variant of OpenCL
- OpenCL is a library for heterogeneous processors such as GPU, FPGA, and DSP

Let's take a look at the code (1/6)

```
82.int main(void)
83.{
84.  pzcl_platform_id platform;
85.  pzclGetPlatformIDs ( 1, &platform, NULL );

87.  pzcl_device_id device;
88.  pzcl_int err;
89.  pzclGetDeviceIDs ( platform, PZCL_DEVICE_TYPE_DEFAULT, 1, &device, NULL );
90.  pzcl_context context = pzclCreateContext ( NULL, 1, &device, NULL, NULL, &err );

92.  pzcl_program program; {
93.    FILE *program_handle = fopen ( KERNEL_FILE, "rb" );
94.    char *program_buffer = ( char * ) malloc ( MAX_BINARY_SIZE );
95.    size_t program_size = fread ( program_buffer, 1, MAX_BINARY_SIZE, program_handle );
96.    pzcl_int program_status;
97.    program =
98.      pzclCreateProgramWithBinary ( context, 1, &device,
99.              ( const size_t * ) &program_size,
100.             ( const unsigned char ** ) &program_buffer,
101.             &program_status, &err );
102.    free ( program_buffer );
103.    fclose ( program_handle );
104.  }
105.
```

Create "platform"

Get devices

Create "context"

Create "program"

Let's take a look at the code (2/6)

Create "kernel"

```
106.  pzcl_kernel kernel = pzclCreateKernel ( program, KERNEL_FUNC, &err );
107.  pzcl_command_queue queue = pzclCreateCommandQueue ( context, device, 0, &err );

109.  float *v  = ( float * ) malloc ( N * sizeof ( float ) );
110.  float *e = ( float * ) malloc ( N * sizeof ( float ) );
111.  float *i = ( float * ) malloc ( N * sizeof ( float ) );
112.  int   *w  = (int *) malloc ( N * N * sizeof ( int ) );
113.  int   *s  = (int *) malloc ( N * sizeof ( int ) );
114.
115.  initialize ( v, e, i, w, s );

117.  pzcl_mem v_dev = pzclCreateBuffer ( context, PZCL_MEM_READ_WRITE, N * sizeof ( float ), NULL, &err);
118.  pzcl_mem e_dev = pzclCreateBuffer ( context, PZCL_MEM_READ_WRITE, N * sizeof ( float ), NULL, &err);
119.  pzcl_mem i_dev = pzclCreateBuffer ( context, PZCL_MEM_READ_WRITE, N * sizeof ( float ), NULL, &err);
120.  pzcl_mem w_dev = pzclCreateBuffer ( context, PZCL_MEM_READ_WRITE, N * N * sizeof ( int ), NULL, &err);
121.  pzcl_mem s_dev = pzclCreateBuffer ( context, PZCL_MEM_READ_WRITE, N * sizeof ( int ), NULL, &err);
122.  pzcl_mem tmp_dev = pzclCreateBuffer ( context, PZCL_MEM_READ_WRITE, 16000 * sizeof ( float ), NULL, &err);
123.
```

Create "queue"

Similar to cudaMalloc

Let's take a look at the code (3/6)

```
82.int main(void)
83.{
84.  pzcl_platform_id platform;
85.  pzclGetPlatformIDs ( 1, &platform, NULL );

87.  pzcl_device_id device;
88.  pzcl_int err;
89.  pzclGetDeviceIDs ( platform, PZCL_DEVICE_TYPE_DEFAULT, 1, &device, NULL );
90.  pzcl_context context = pzclCreateContext ( NULL, 1, &device, NULL, NULL, &err );

92.  pzcl_program program; {
93.    FILE *program_handle = fopen ( KERNEL_FILE, "rb" );
94.    char *program_buffer = ( char * ) malloc ( MAX_BINARY_SIZE );
95.    size_t program_size = fread ( program_buffer, 1, MAX_BINARY_SIZE, program_handle );
96.    pzcl_int program_status;
97.    program =
98.      pzclCreateProgramWithBinary ( context, 1, &device,
99.              ( const size_t * ) &program_size,
100.             ( const unsigned char ** ) &program_buffer,
101.             &program_status, &err );
102.    free ( program_buffer );
103.    fclose ( program_handle );
104.  }
105.
```

Create "platform"

Get devices

Create "context"

Create "program"

Let's take a look at the code (4/6)

```
124.  pzclEnqueueWriteBuffer ( queue, v_dev, PZCL_TRUE, 0, N * sizeof ( float ), v, 0, NULL, NULL );
125.  pzclEnqueueWriteBuffer ( queue, e_dev, PZCL_TRUE, 0, N * sizeof ( float ), e, 0, NULL, NULL );
126.  pzclEnqueueWriteBuffer ( queue, i_dev, PZCL_TRUE, 0, N * sizeof ( float ), i, 0, NULL, NULL );
127.  pzclEnqueueWriteBuffer ( queue, w_dev, PZCL_TRUE, 0, N * N * sizeof ( int ), w, 0, NULL, NULL );
128.  pzclEnqueueWriteBuffer ( queue, s_dev, PZCL_TRUE, 0, N * sizeof ( int ), s, 0, NULL, NULL );

130.  pzclSetKernelArg(kernel, 0, sizeof ( pzcl_mem ), &v_dev);
131.  pzclSetKernelArg(kernel, 1, sizeof ( pzcl_mem ), &e_dev);
132.  pzclSetKernelArg(kernel, 2, sizeof ( pzcl_mem ), &i_dev);
133.  pzclSetKernelArg(kernel, 3, sizeof ( pzcl_mem ), &w_dev);
134.  pzclSetKernelArg(kernel, 4, sizeof ( pzcl_mem ), &s_dev);
135.  pzclSetKernelArg(kernel, 6, sizeof ( pzcl_mem ), &tmp_dev);

137.  size_t global_work_size = GLOBAL_WORK_SIZE;

139.  timer_start ( );
140.  for ( int t = 0; t < T; t++ ) {
141.    float i_cur = 1.0*(1+sin(t/10.0));
142.    pzclSetKernelArg(kernel, 5, sizeof ( float ), &i_cur);
143.    pzclEnqueueNDRangeKernel ( queue, kernel, 1, NULL, &global_work_size, NULL, 0, NULL, NULL );
144.    pzclEnqueueReadBuffer ( queue, s_dev, PZCL_TRUE, 0, N * sizeof(int), s, 0, NULL, NULL);

146.    outputSpike ( DT * t, s );
147.  }
148.
```

Similar to cudaMemcpy

Set kernel arguments

Kernel launch

Transfer spike data from device to host

Let's take a look at the code (5/6)

```
149.  double elapsedTime = timer_elapsed ( );
150.  printf ( "Elapsed time = %f sec.\n", elapsedTime);

152.  finalize ( v, e, i, w, s );

154.  pzclReleaseMemObject ( v_dev );
155.  pzclReleaseMemObject ( e_dev );
156.  pzclReleaseMemObject ( i_dev );
157.  pzclReleaseMemObject ( w_dev );
158.  pzclReleaseMemObject ( s_dev );
159.  pzclReleaseMemObject ( tmp_dev );

161.  pzclReleaseKernel ( kernel );
162.  pzclReleaseCommandQueue ( queue );
163.  pzclReleaseProgram ( program );
164.  pzclReleaseContext ( context );

166.  return 0;
167.}
168.
```

Similar to cudaFree

Similar to cudaFree

Let's take a look at the code (6/6)

```
116. void pzc_kernel ( float *v, float *e, float *i, int *w, int * s, float i_cur, float *tmp )
117.{
118.   int tid = get_tid ( );
119.   int pid = get_pid ( );
120.   int id = pid * get_maxtid ( ) + tid;

121.   if ( id < N ) { calculateSynapse ( id, e, i, w, s, tmp ); }
122.   flush ( );

124.   if ( id < N ) { updateMembranePotential ( id, v, e, i, s, i_cur ); }
125.   flush ( );
126.}
127.
```

Similar to `int i = threadIdx.x + blockIdx.x * blockDim.x;`

Let's try the code!

- `make`
- `PZCLReserveDevices=1 ./main`
- Measure the computational time

Wrap-up

- Writing OpenCL is tedious, but translation is straightforward
- In many cases, simple translation from CUDA is enough
- If you are interested in using this machine for your project, please contact us

Please logout from the supercomputer

Final remarks

- We learned how to use OpenMP, MPI, GPU, and PEZY-SC2 independently
- For OpenMP, GPU, and PEZY-SC2, we just used 1 single device
- Multiple devices can be used in parallel via MPI, such as OpenMP + MPI, GPU + MPI, and PEZY-SC2 + MPI
- Hybrid parallelization is necessary for large-scale simulation

Closing