Mathematical Foundations of Neuroscience - Lecture 15. Building large models step by step

Filip Piękniewski

Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Toruń, Poland

Winter 2009/2010

- Let us now quickly summarize the knowledge we've obtained
- We will briefly sketch the details of a fairly accurate large scale neuronal simulation (though many other frameworks are possible)
- We will see some tips on implementation with the current technology
- We shall sneak into the future...

Neurons

- We need to first establish the level of spatial accuracy of our neuron. We can choose from:
  - A mesh based on morphology, with cable-like equation (high accuracy, great computational demand)
  - Multi compartment model, with compartments connected via conductances (quite good accuracy, but much faster)
  - Single compartment model with dendritic delays (very efficient, yet still catches a lot of neuronal dynamics)
  - Single compartment model, no delays (very fast, rather inaccurate)
  - Rate based models - no spatial dynamics, just rates - typical artificial neural networks (efficient and useful in engineering, but useless in the brain simulation)

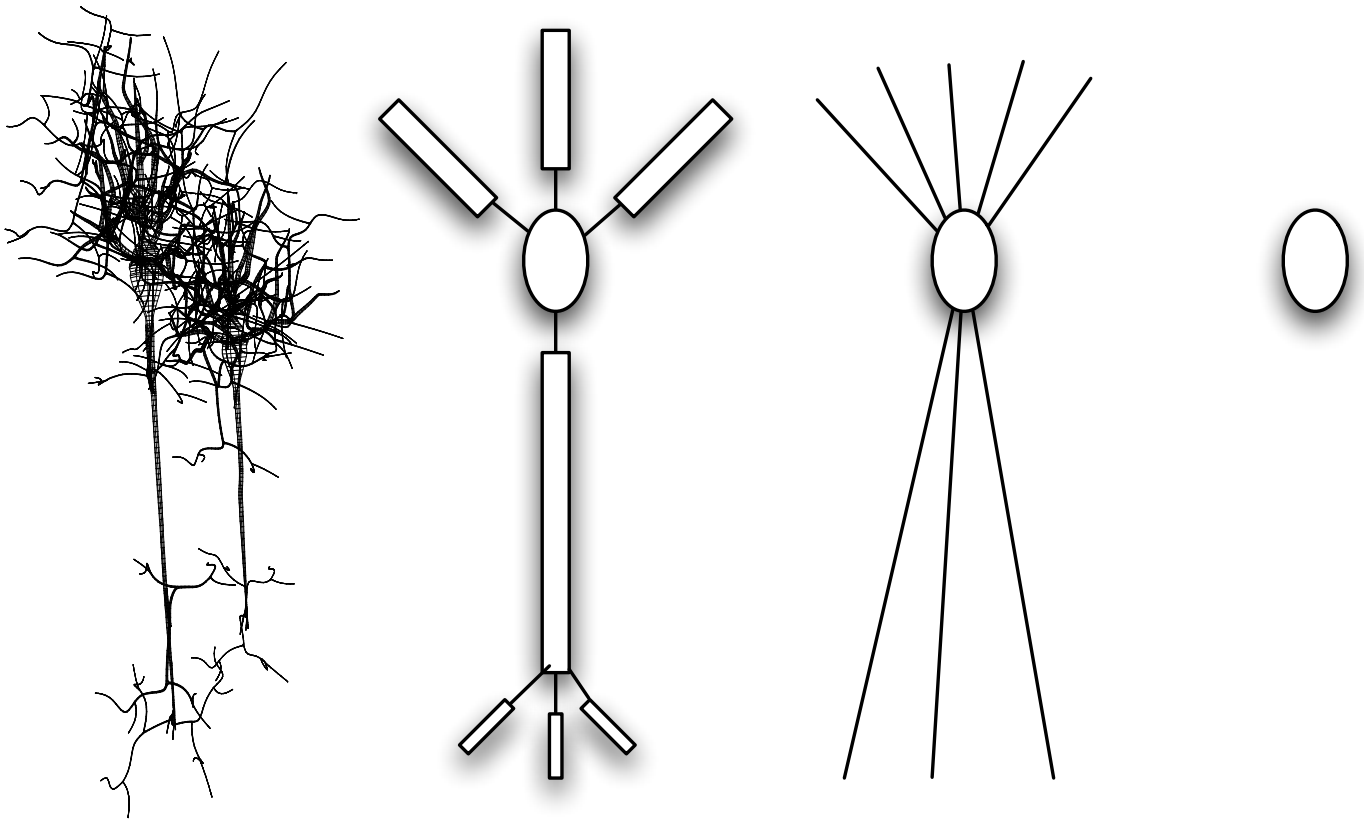

Figure: Different levels of accuracy in neuron modeling.

Neural model

Next we have to choose the spiking model for the simulated membrane:

- Hodgkin-Huxley type model - accurate but inefficient
- Simplified conductance based model (like $I_{Na,p} + I_K$ model) - fairly accurate, only a little faster than HH.
- Fithugh-Nagumo - quite accurate, quite efficient (no exponentials, but requires small integration steps).
- Izhikevich simple model - quite accurate, very efficient (no exponentials, runs with large integration steps)
- Neuromime - rather inaccurate, very efficient
- Leaky integrate and fire - inaccurate, very efficient

Depression-Facilitation

- There are synapses which depress or facilitate in reaction to spiking.
- Henry Markram and his collaborators introduced in 1998 a phenomenological model:

$$\frac{dR}{dt} = \frac{1-R}{D} \qquad \frac{dw}{dt} = \frac{U-w}{F}$$

where U, D, F are parameters. Whenever a spike is propagated through the synapse $R := R - Rw$ and $w := w + U(1-w)$. R is the depression variable and w is the facilitation variable. The total synaptic strength at time t is equal $S = Rw$

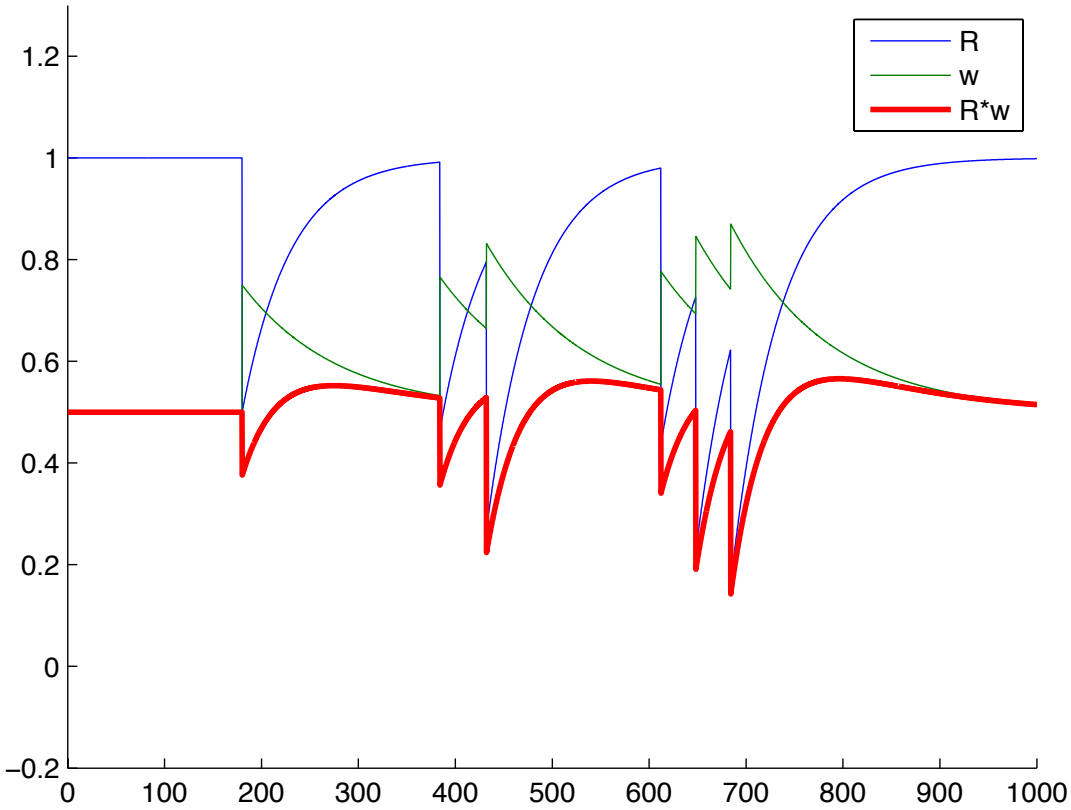

Figure: A synapse modeled by the phenomenological model of H. Markram exhibiting depression (1) $F = 50, D = 100, U = 0.5$ and facilitation (2) $D = 100, F = 50, U = 0.2$. By adjusting the parameters the model can reproduce conductances of various synapses.

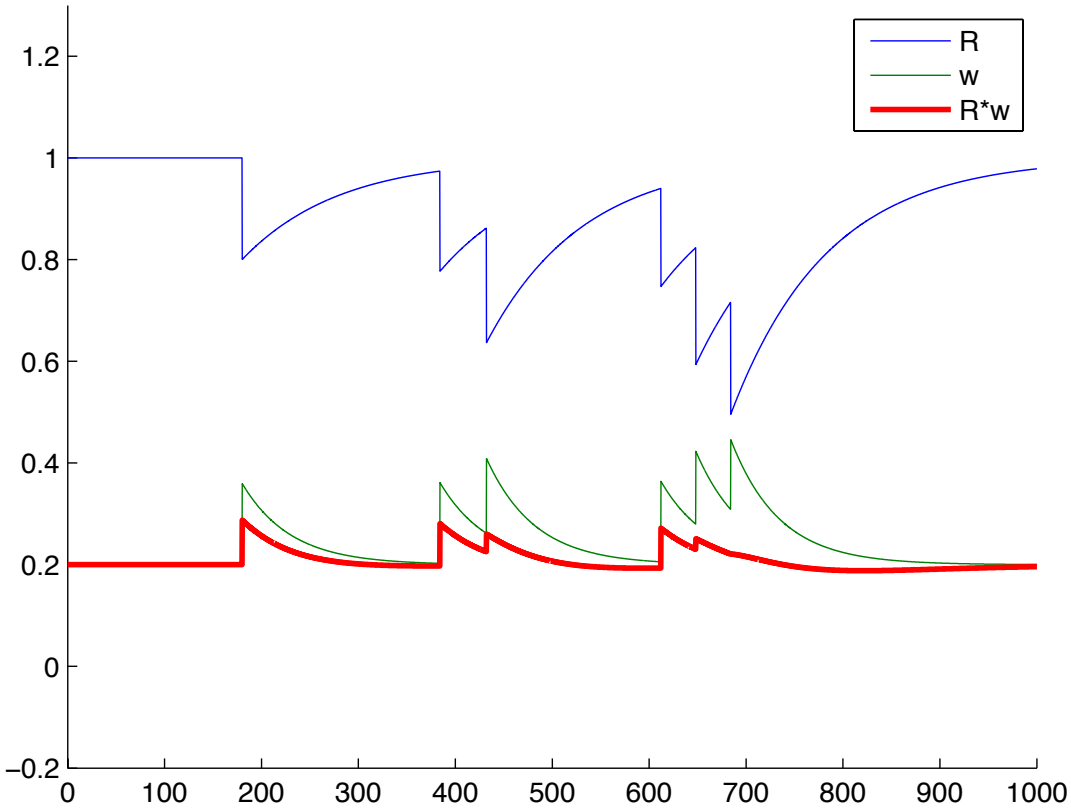

Figure: A synapse modeled by the phenomenological model of H. Markram exhibiting depression (1) $F = 50, D = 100, U = 0.5$ and facilitation (2) $D = 100, F = 50, U = 0.2$. By adjusting the parameters the model can reproduce conductances of various synapses.

Short term plasticity

- Markram model quite well resembles experimental data, but can still be too complex for very large simulations. In such case one can downgrade into yet simpler solution, like the one below.
- The simplest formulation is:

$$\frac{dS}{dt} = (1 - S)/\tau_s$$

and $S := pS$ whenever an action potential is transferred. The synapse gets depressed for $p < 0$ and facilitated for $p > 0$.

- The value of the short term depression-facilitation influences the resulting receptor conductance. In the present scope we assume there are four conductances g_{AMPA}, g_{NMDA}, g_{GABA_A} and g_{GABA_B}. Each time a spike is propagated the appropriate conductances are increased by $w_{i\to j}S_i = w_{i\to j}R_i w_i$ (before depressing or facilitating) where $w_{i\to j}$ is the strength of the synapse from neuron i to neuron j.
- The conductances have their own kinetics, that is they diminish exponentially as

$$\frac{dg}{dt} = -g/\tau$$

where $\tau_{\mathrm{AMPA}} = 5ms$, $\tau_{\mathrm{NMDA}} = 150ms$, $\tau_{\mathrm{GABA}_A} = 6ms$ and $\tau_{\mathrm{GABA}_B} = 150ms$

STDP

- To make a model interesting, it should be able to learn (modify synapses)
- Consequently one of the learning rules mentioned last week should be implemented.
- For spiking neurons it is best to use Spike Timing Dependent Plasticity, as this rule seems to be most accurate, is stable and possible to implement online via Long term depression/potentiation (LTP, LTD) traces.

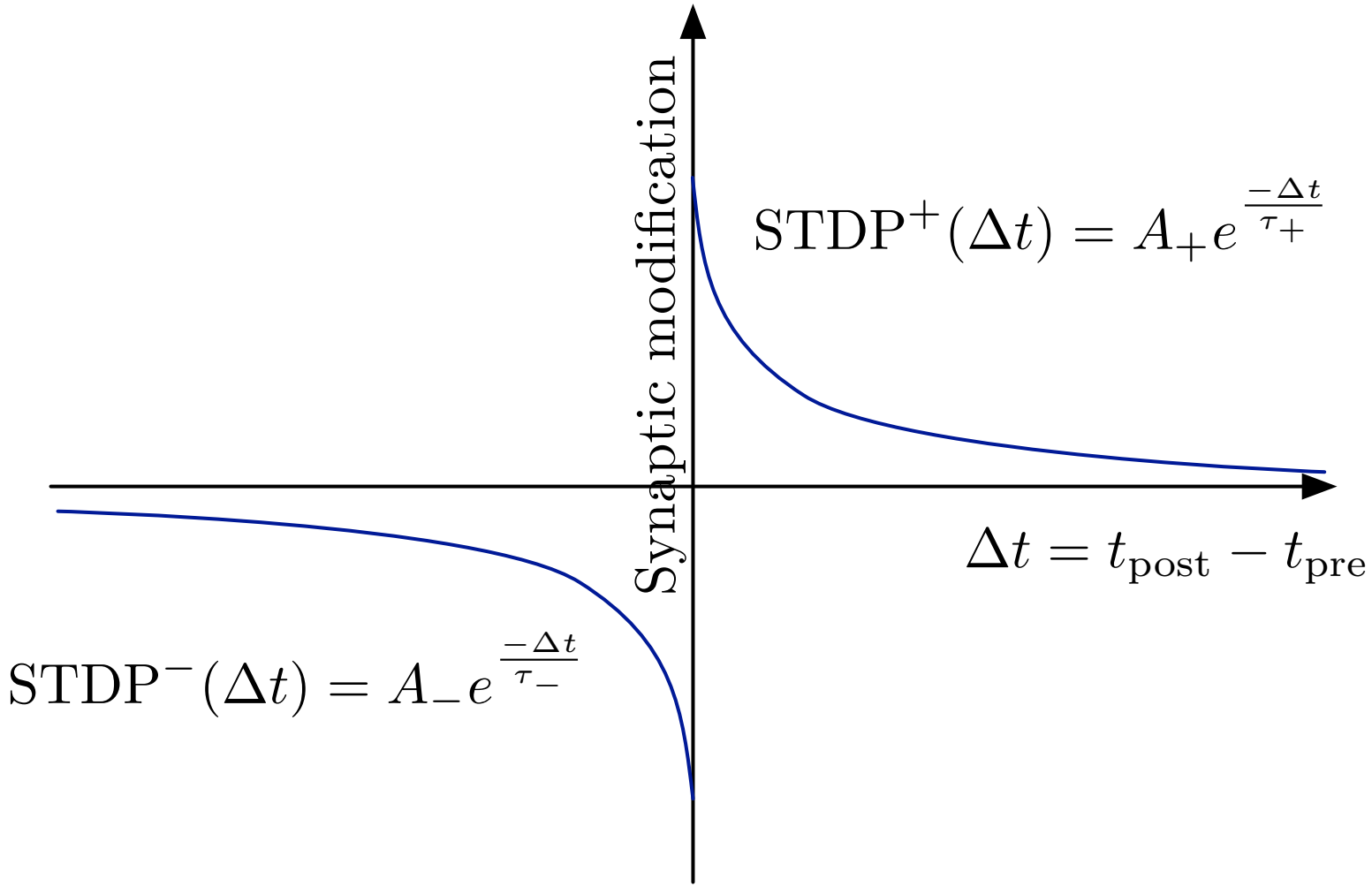

Figure: Spike timing dependent plasticity - diagram.

- Assume that each presynaptic spike leaves an exponentially decaying trace:

$$\tau_+ \frac{dP}{dt} = -P + \sum_{k=1}^{n} \delta(t - t_k^{(i)})$$

- Similarly each postsynaptic spike leaves a trace

$$\tau_- \frac{dD}{dt} = -D + \sum_{s=1}^{n} \delta(t - t_s^{(j)})$$

- The weight change is then:

$$\frac{dw_{ij}}{dt} = \mathrm{STDP}(t) = A_+ \cdot P \sum_{s=1}^{n} \delta(t - t_s^{(j)}) + A_- \cdot D \sum_{k=1}^{n} \delta(t - t_k^{(i)})$$

- The result of STDP should be slow (order of seconds/minutes). This can be achieved by taking very small A_+ and A_-. For certain reasons it is not the best approach.
- A better idea is to model the synaptic plasticity indirectly with the help of a *synaptic tag*, an additional variable c (which might be interpreted as the concentration of an enzyme necessary for synaptic modification):

$$\frac{dc}{dt} = \frac{-c}{\tau_c} + \mathrm{STDP}(t)$$

and then:

$$\frac{dw_{ij}}{dt} = cd$$

where d is the plasticity parameter (which could be time dependent, e.g. concentration of dopamine which enhances synaptic plasticity)

How to obtain connectivity data?

- For simple simulations, one can use any reasonable topology with some fraction of random long range connections. Most probably the unneeded connections will be wiped out by synaptic plasticity. However such a setup could lack important pathways that are essential for vital brain functions.
- A more detailed large scale structure can be obtained from Diffusion Tensor Imaging MRI (DTI). Note that slicer3d, a free program that allows fiber tracking can save fibers into a file! A ready source of brain-like connectivity!
- Small scale structure of the cortex and other brain areas has been studied for many years now, and much statistical data on which neurons wire to what neurons is available (see e.g. Izhikevich and Edelman 2007 PNAS paper).

Difusion tensor imaging (DTI)

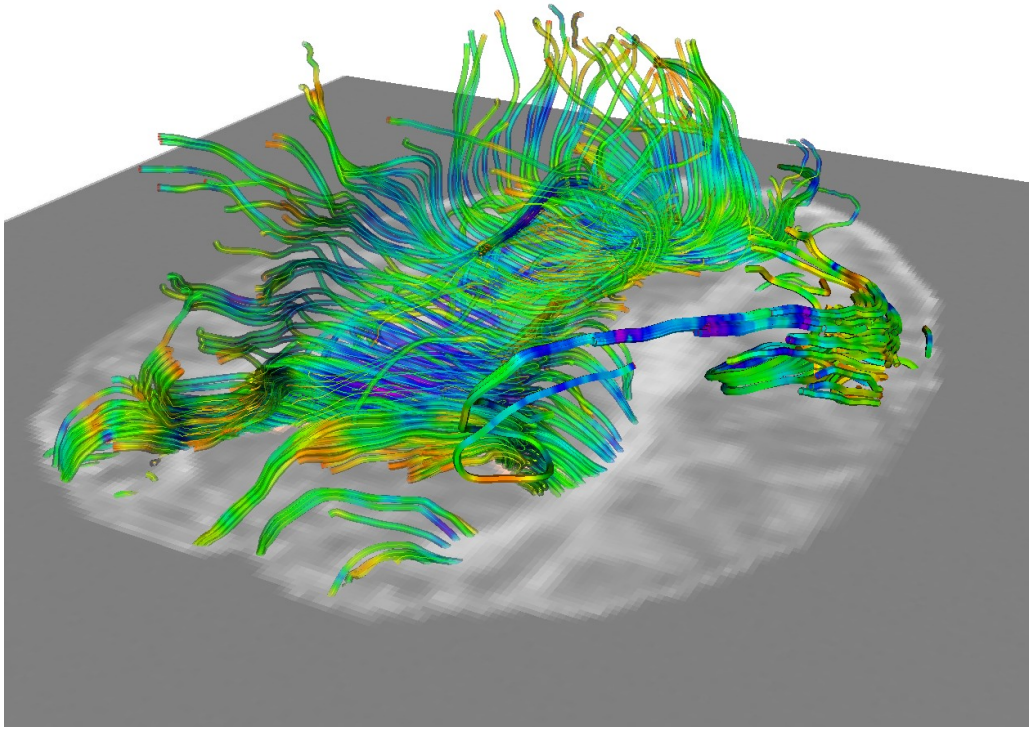

Figure: MRI fiber tracking using water diffusion tensor.

Dendritic and axonal delays

- In a coarse grained setup we may as well assume, that each neuron is connected to some others with certain (order of ms) delays. Experiments show that the signal proceeds with the speed of about 0.15m/s. Signal are much faster in long myelinated fibers (axons), in which the speed o propagation reaches 1m/s
- Propagation delays can therefore reach 10ms for non-myelinated local axonal collaterals (assuming the reach at most 1.5mm) and on the order of 100ms for myelinated connections assuming they reach 10cm within the brain (they may be longer in the spinal cord, spike the propagation velocities are somewhat larger as well).

How to implement?

Each neurosimulation can be divided into parts:

- Neural dynamics (the decision whether to spike or not)
- Spike propagation (forwarding spikes to their targets, possibly including delays)
- Synaptic plasticity (modulation of synapses in response to their activity):
  - Simulation of binding kinetics of the neurotransmitter (GABA, AMPA, NMDA etc)
  - Simulating the overall amount of neurotransmitter (short term depression/facilitation)
  - Simulating slow changes to synapse strength (STDP, BCM or other learning rule)

- Once we have the synaptic conductances we have to look what currents they may cause through the membrane, depending on voltage V. We generally have that:

$$
\begin{aligned}
I_{\text{syn}} = {} & g_{\text{AMPA}}(E_{\text{AMPA}} - V) + \\
& + g_{\text{NMDA}} \frac{\left(\frac{V+80}{60}\right)^2}{1 + \left(\frac{V+80}{60}\right)^2}(E_{\text{NMDA}} - V) + \\
& + g_{\text{GABA}_A}(E_{\text{GABA}_A} - V) + \\
& + g_{\text{GABA}_B}(E_{\text{GABA}_B} - V)
\end{aligned}
$$

with $E_{\text{AMPA}} = 0$, $E_{\text{NMDA}} = 0$, $E_{\text{GABA}_A} = -70$, $E_{\text{GABA}_B} = -90$. NMDA current looks strange but the formula is a fit to empirical data.

Eventually, using say the Izhikevich Simple Neuron (with reset) we have (for a single compartment model):

$$\begin{aligned}\frac{dV_i}{dt} =& 0.04V_i^2 + 5V_i + 140 - U_i + \sum_{j\to i} g_{\text{j,AMPA}}(0 - V_i) + \\ &+ \sum_{j\to i} g_{\text{j,NMDA}} \frac{\left(\frac{V_i+80}{60}\right)^2}{1+\left(\frac{V_i+80}{60}\right)^2}(0 - V_i) + \sum_{j\to i} g_{\text{j, GABA}_A}(-70 - V_i) + \\ &+ \sum_{j\to i} g_{\text{j,GABA}_B}(-90 - V_i) + \sum_{j\in \text{gap}(i)} g_{\text{gap}j\to i}(V_j - V_i)\end{aligned}$$

$$\frac{dU_i}{dt} = a(bV_i - U_i)$$

Consequently by solving with respect to $V_i(t+1)$

$$V_i(t+1) =$$

$$= \frac{V_i(t) + \Delta t \left(0.04 V_i(t)^2 + 5 V_i(t) + 140 - U_i(t) + \right.}{1 + \Delta t \left(\sum_{j \to i} g_{\text{j,AMPA}} + \sum_{j \to i} g_{\text{j,NMDA}} \frac{\left(\frac{V_i(t)+80}{60}\right)^2}{1+\left(\frac{V_i(t)+80}{60}\right)^2} + \right.}$$

$$\left. \frac{\left. -70 \sum_{j \to i} g_{\text{j, GABA}_A} - 90 \sum_{j \to i} g_{\text{j,GABA}_B} + \sum_{j \in \text{gap}(i)} g_{\text{gap}j \to i} (V_j - V_i(t)) \right)}{\left. \sum_{j \to i} g_{\text{j, GABA}_A} + \sum_{j \to i} g_{\text{j, GABA}_B} \right)}$$

$$U_i(t+1) = U_i(t) + \Delta t \left(a(bV_i(t) - U_i(t))\right)$$

W get a stable scheme, ready to use for large scale simulations.

- So assume that we are simulating single compartment neurons with axonal delays.
- If the integration step is of order 1ms then it makes sense to consider only discrete delays (multiples of 1ms). Say the maximum conduction delay is 50ms.
- In such case we can keep the spikes on a set of queues. At each step we process one queue (the spikes that arrive now at their targets), possibly igniting their targets. The new spikes are placed on the appropriate queues corresponding to the delays of each axon. Since the step is 1ms and the maximal delay is 50ms, we need only 50 queues for every delay, iterated by time step (modulo 50).

- Note that the short term synaptic depression/facilitation depends only on timing of the presynaptic spikes. Therefore if we assume that each neuron has identical synapses (in terms of short term plasticity), we can actually compute synaptic depression/potentiation per neuron.
- Secondly each neuron keeps its postsynaptic depression trace (LTD). Now each synapse should have presynaptic potentiation trace (LTP), but again the value of LTP depends only on timing of presynaptic spikes and is identical (modulo conductance delay) in every synapse of a single neuron. Since maximal delay is 50ms, we keep all 50 past values of LTP trace (each new one is obtained from previous via multiplication by some factor < 1). If a postsynaptic neuron spikes, it can potentiate the synapse tag by finding the appropriate trace depending on the relative delay.

Summarizing:

- Each neuron keeps its state, its synaptic conductances, short term depression/facilitation, one LTD trace and a number (depending on the maximal conductance delay) of past LTP traces.
- Each neuronal spike evokes depression/facilitation update, LTP and LTD increase and insertion of spikes into appropriate queues. Also each tag of an input synapse is updated using the appropriate value of delay dependent presynaptic LTP trace.
- Each arrival of a spike at a neuron (once the queue is processed) evokes depression of a synaptic tag by postsynaptic LTD trace and an increase in the corresponding synaptic conductance.

- Since the conductance kinetics is linear:

$$\frac{dg}{dt} = -g/\tau$$

and what comes in into neuronal equation is the sum:

$$\sum_{j \to i} g$$

we may simulate the global synaptic conductance:

$$g_{\text{global}} = \sum_{j \to i} g, \qquad \frac{dg_{\text{global}}}{dt} = -g_{\text{global}}/\tau$$

- Consequently we have only one conductance trace per neurotransmitter per neuron incremented by incoming spikes.

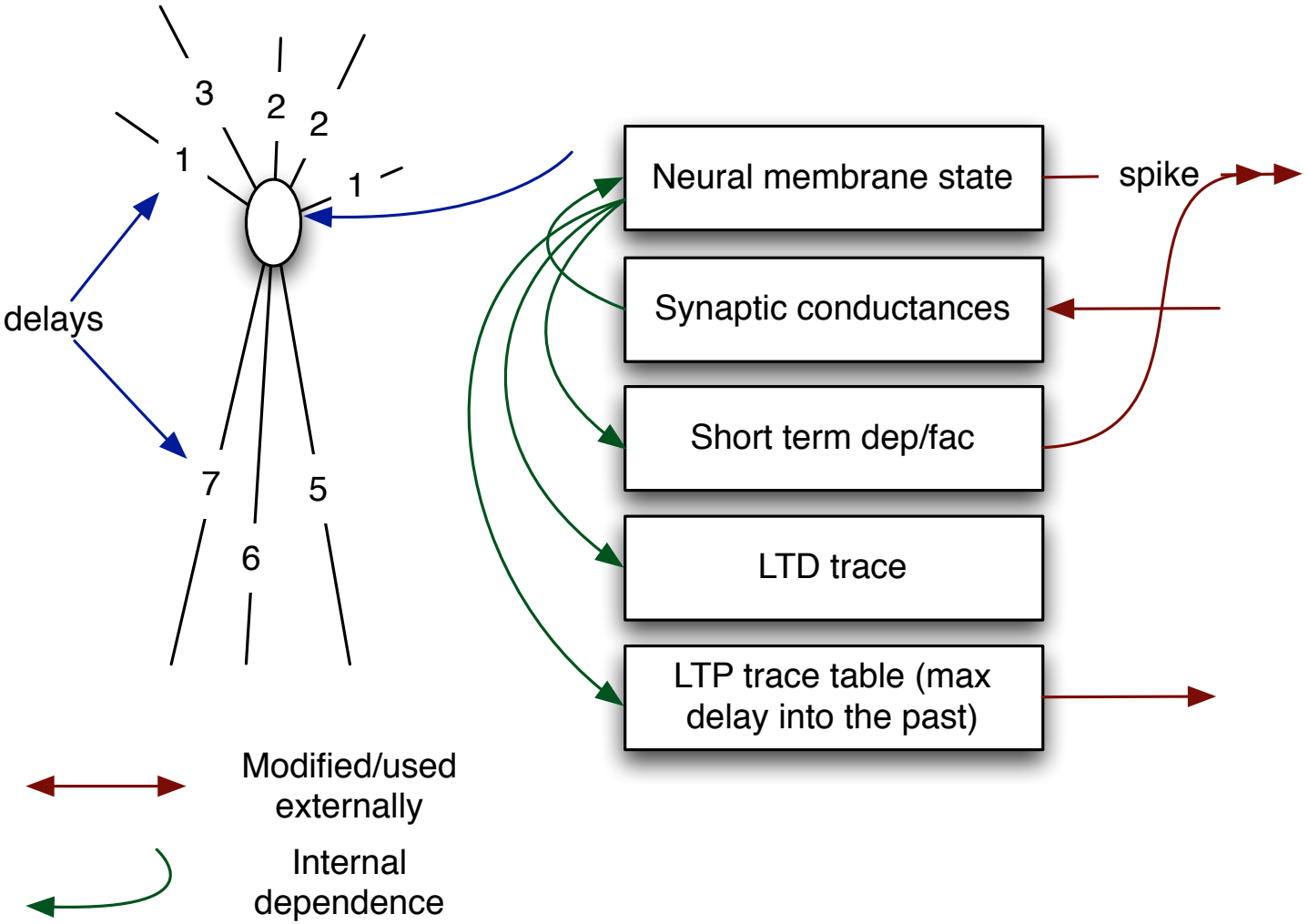

Figure: A sample model structure.

Finally note that the Markram synapse model:

$$\frac{dR}{dt} = \frac{1-R}{D} - Rw\delta(t - t_{spike})$$
$$\frac{dw}{dt} = \frac{U-w}{F} + U(1-w)\delta(t - t_{spike})$$

by a substitution of variables:

$$R_{fast} = 1 - R \qquad w_{fast} = w - U = -(U - w)$$

can be transformed into a more efficient form:

$$\frac{dR_{fast}}{dt} = -\frac{R_{fast}}{D}$$
$$\frac{dw_{fast}}{dt} = -\frac{w_{fast}}{F}$$

then

$$Rw = (1 - R_{fast}) \cdot (U + w_{fast})$$

Parallelization

- One obvious necessity in any simulation is parallelism.
- For neural simulation most computations are done locally, independently of the rest.
- The crucial part that requires synchronization is spike propagation.
- Recall Amdahl's law: if a portion P of your code can receive speedup S (say $S = N$, where N is the number of parallel processors) then the overall speedup of your program is:

$$\frac{1}{(1-P) + \frac{P}{S}}$$

- This means that a decent overall speedup with many processors can only be achieved if the non parallelizing part of the code is negligibly small!

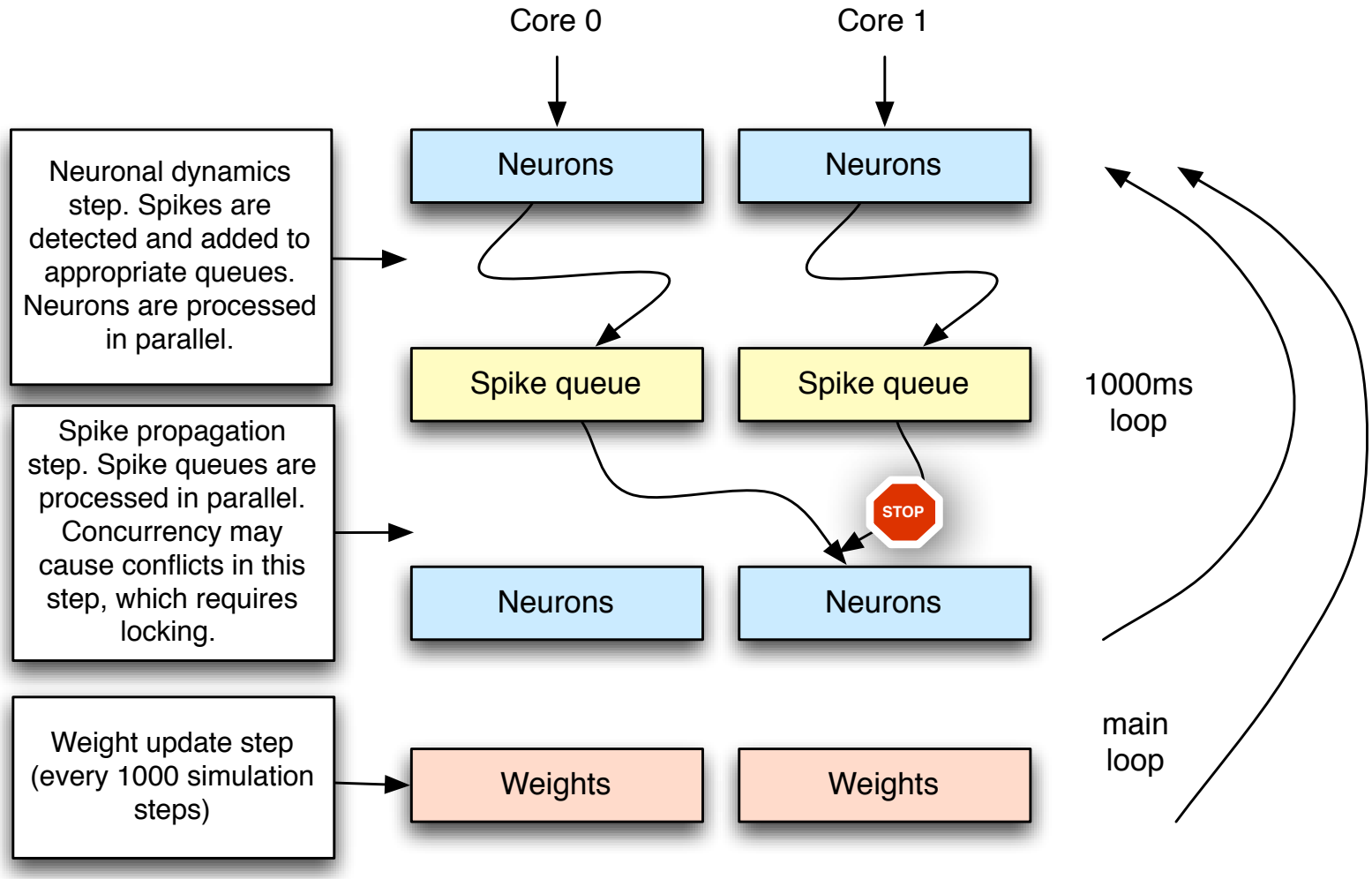

Figure: Structural presentation of the code used in simulation.

- In a straightforward implementation, target neurons have to be protected by locks.
- But any critical section in the code is a portion of non parallelizing code, that diminishes overall speedup.
- The locks can be avoided if no two spikes from two separate queues can arrive in a common target in the same time. It might not be easy, but it is possible to create a code that would satisfy such conditions.
- Technically the parallelization can be easily done using **openmp** language extensions (for single machine SMP), or using MPI etc (for clusters and computing farms).

GPU

- I recent years efficient parallel (vector like) processors have been developed to support graphics accelerators used mainly for gaming
- Currently a powerful graphics card can easily exceed the computing power of the CPU
- GPU's however need to be programmed differently than CPU's - usually they offer multiple cores (128, 512 etc) and a fast, simple memory access. All cores run a single program (called a kernel), but perform their operations on different data. Threads can branch, but branching decreases efficiency.
- The model of computing is sometimes called SIMT - Single Instruction Multiple Thread

```
 1 void step_cpu()
 2 {
 3       int i;
 4       for(i=0;i<N;i++)
 5          {
 6               // Random number generator
 7                   rnd[i] = rnd[i]* 1103515245 + 12345;
 8               float frand = (float)((rnd[i]/65536) % 32768)/32768;//
''random'' float
 9                   rnd[i] = rnd[i]* 1103515245 + 12345;
10               float frand1 = (float)((rnd[i]/65536) % 32768)/32768;
11               if (frand>0)
12                  frand=sqrt(-2*logf(frand))*cosf(2*3.14159265*frand1);
13               if (i%5!=0)  frand*=thalamic_excitatory;
14                       else frand*=thalamic_inhibitory;
15               I[i]+=frand;
16               v[i]=v[i]+0.25*((0.04*v[i]+5)*v[i]+140-u[i]+I[i]);
17               v[i]=v[i]+0.25*((0.04*v[i]+5)*v[i]+140-u[i]+I[i]);
18               v[i]=v[i]+0.25*((0.04*v[i]+5)*v[i]+140-u[i]+I[i]);
19               v[i]=v[i]+0.25*((0.04*v[i]+5)*v[i]+140-u[i]+I[i]);
20               u[i]=u[i]+0.5*a[i]*(b[i]*v[i]-u[i]);
21               u[i]=u[i]+0.5*a[i]*(b[i]*v[i]-u[i]);
22               I[i]=0;
23          }
24 }
25
```

```
1  void process_spikes()
2  {
3          int i;
4             for(i=0;i<N;i++)
5             {
6                if (v[i]>30)
7                  {
8                        v[i]=c[i];
9                        u[i]=u[i]+d[i];
10                       for(int k=0;k<nei;k++) I[(i+k)%N]+=S[i*nei+k];
11                       spikes++;
12                 }
13            }
14 }
15
```

```
__global__void step_gpu(float *v_gpu, float* u_gpu, float* a_gpu, float* b_gpu, float*
I_gpu, unsigned int *rnd_gpu)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int step = gridDim.x * blockDim.x;
  for (; i < N_gpu; i += step)
   {
          // Random number generator
          rnd_gpu[i] = rnd_gpu[i]* 1103515245 + 12345;
          float frand = (float)((rnd_gpu[i]/65536) % 32768)/32768;  //''random''
float
          rnd_gpu[i] = rnd_gpu[i]* 1103515245 + 12345;
          float frand1 = (float)((rnd_gpu[i]/65536) % 32768)/32768; //''random''
float
          if (frand>0)
             frand=sqrt(-2*logf(frand))*cosf(2*3.14159265*frand1);
          if (i%5!=0)  frand*=thalamic_excitatory;
                 else frand*=thalamic_inhibitory;
          I_gpu[i]+=frand;
          v_gpu[i]=v_gpu[i]+0.25*((0.04*v_gpu[i]+5)*v_gpu[i]+140-u_gpu[i]+I_gpu[i]);
          v_gpu[i]=v_gpu[i]+0.25*((0.04*v_gpu[i]+5)*v_gpu[i]+140-u_gpu[i]+I_gpu[i]);
          v_gpu[i]=v_gpu[i]+0.25*((0.04*v_gpu[i]+5)*v_gpu[i]+140-u_gpu[i]+I_gpu[i]);
          v_gpu[i]=v_gpu[i]+0.25*((0.04*v_gpu[i]+5)*v_gpu[i]+140-u_gpu[i]+I_gpu[i]);
          u_gpu[i]=u_gpu[i]+0.5*(a_gpu[i]*(b_gpu[i]*v_gpu[i]-u_gpu[i]));
          u_gpu[i]=u_gpu[i]+0.5*(a_gpu[i]*(b_gpu[i]*v_gpu[i]-u_gpu[i]));
          I_gpu[i]=0;
   }
}
```

```
 1 __global__void process_spikes_gpu(float *v_gpu, float* u_gpu, float* c_gpu,
float *d_gpu, float* I_gpu, float* S_gpu)
 2 {
 3   int i = blockIdx.x * blockDim.x + threadIdx.x;
 4   int step = gridDim.x * blockDim.x;
 5   int k;
 6   for (; i < N_gpu; i += step)
 7     {
 8     if (v_gpu[i]>30)
 9        {
10            v_gpu[i]=c_gpu[i];
11            u_gpu[i]=u_gpu[i]+d_gpu[i];
12            for(k=0;k<nei_gpu;k++) I_gpu[(i+k)%N_gpu]+=S_gpu[i*nei_gpu+k];
13        }
14     }
15 }
16
```

The main program

```
1   struct timeval t1, t2, t3;
2   set_up();
3   cout << "Running on CPU"<< flush << endl;
4   gettimeofday(&t1, 0);
5   for(int t=0;t<10000;t++)
6    {
7          process_spikes();
8          step_cpu();
9          if (t%1000==0) cout << '.'<< flush;
10   }
11  gettimeofday(&t2, 0);
12  double time1 = (1000000.0*(t2.tv_sec-t1.tv_sec) + t2.tv_usec-t1.tv_usec)/1000000.0;
13  cout << endl;
14  cout << time1 << "s - time CPU"<< endl;
15  cout << "Running on GPU"<< endl;
16  for(int t=0;t<10000;t++) {
17        process_spikes_gpu<<<32, 128, 0>>>(v_gpu,u_gpu,c_gpu,d_gpu,I_gpu,S_gpu);
18        cudaThreadSynchronize();
19        step_gpu<<<32, 128, 0>>>(v_gpu,u_gpu,a_gpu,b_gpu,I_gpu,rnd_gpu);
20        if (t%1000==0) cout << '.'<< flush;
21  }
22  cout << endl;
23  cout << "Waiting for GPU to finish"<< flush << endl;
24
```

Neuromorphic hardware

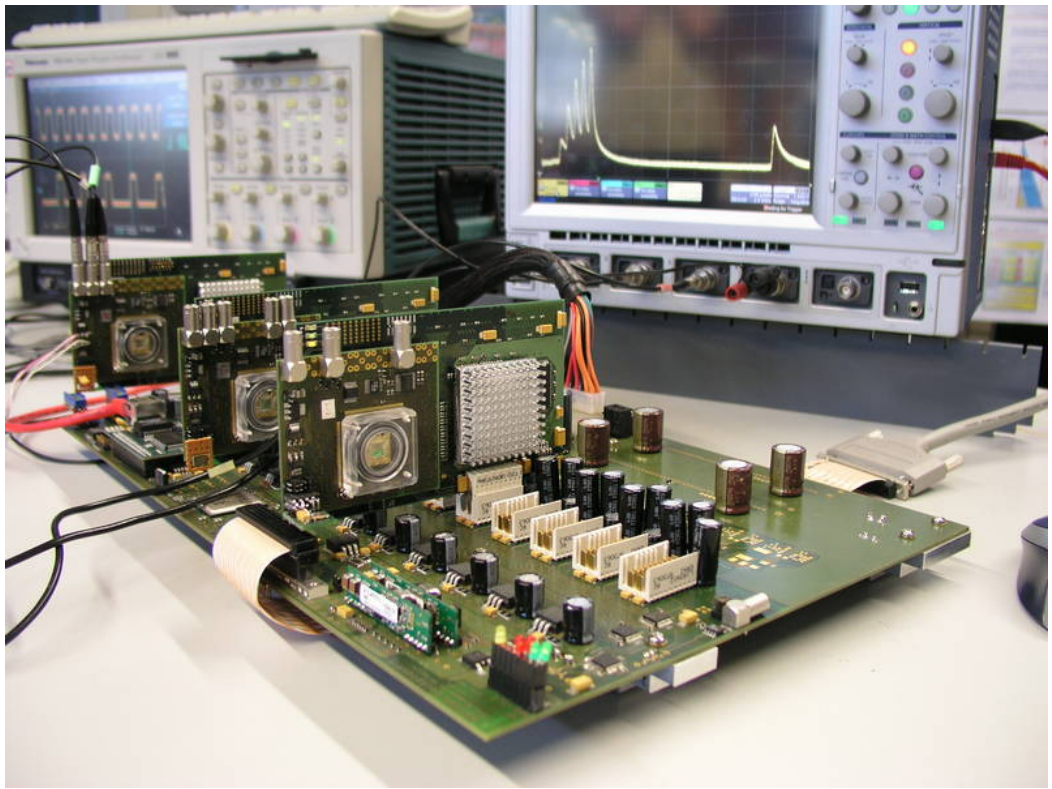

Figure: The real future is in the neuromorphic hardware. The figure shows neuromorphic chips developed at the University of Heidelberg http://www.kip.uni-heidelberg.de/cms/groups/vision/.

Future

- There is a great interest in spiking neural networks, both in the military and consumer applications
- U.S. governments Defense Advanced Research Projects Agency (DARPA) is running the SyNAPSE program (Systems of Neuromorphic Adaptive Plastic Scalable Electronics) which "seeks to develop a brain-inspired electronic 'chip' that mimics that function, size, and power consumption of a biological cortex".
- There are many corporations and startups that seek more or less similar goals:
  - Evolved machines `http://www.evolvedmachines.com`
  - Brain Corporation `http://braincorporation.com`
- Studying neurocomputing is a good choice for the future!

Future

Oh, and a one more thing:

- If the Moore's law persists we should be able to reach the computational power of a cerebral cortex by 2020, that is within 10 years from now.
- Raymond Kurzweil's technological singularity may be nearer than we think!