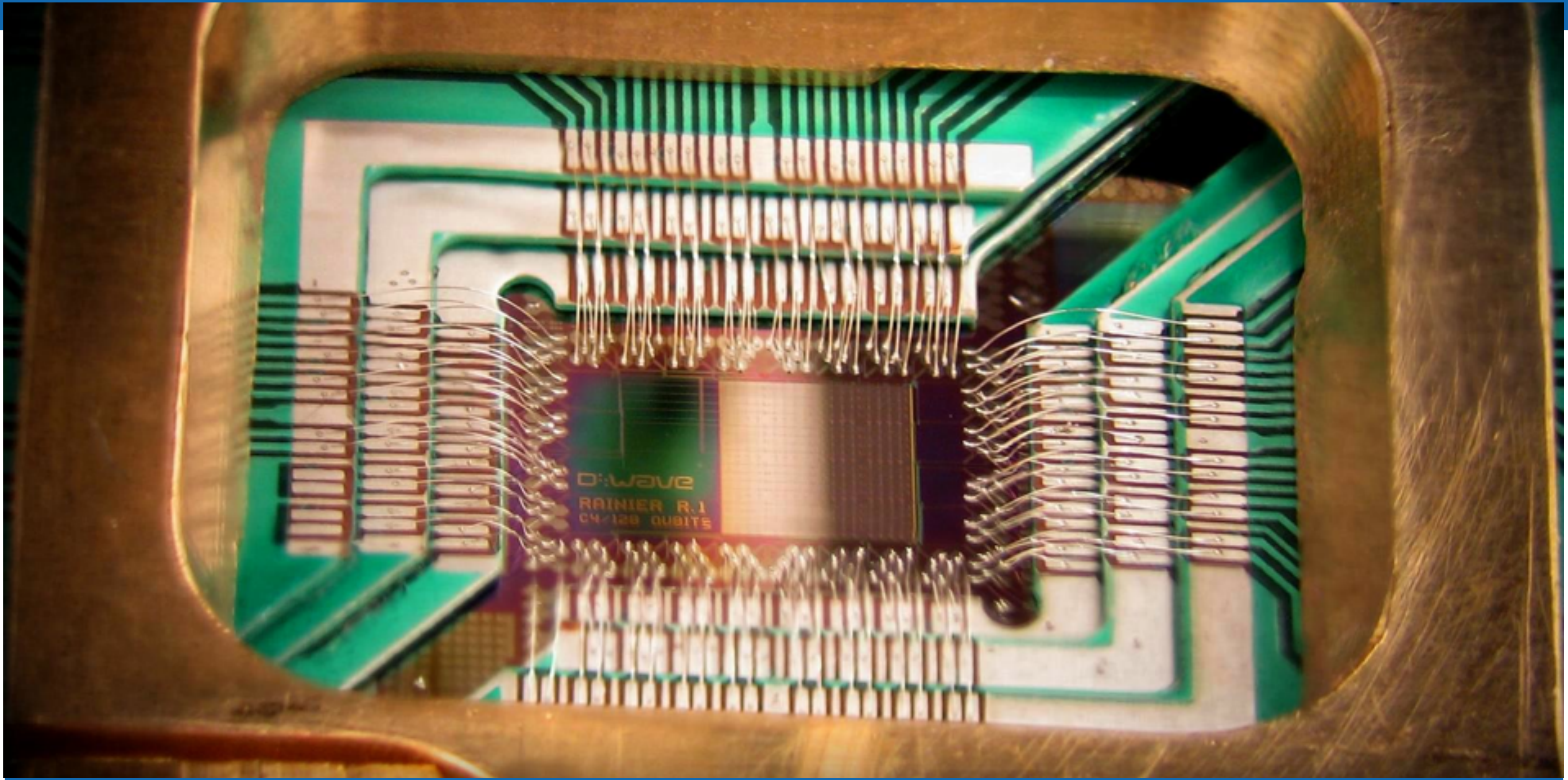
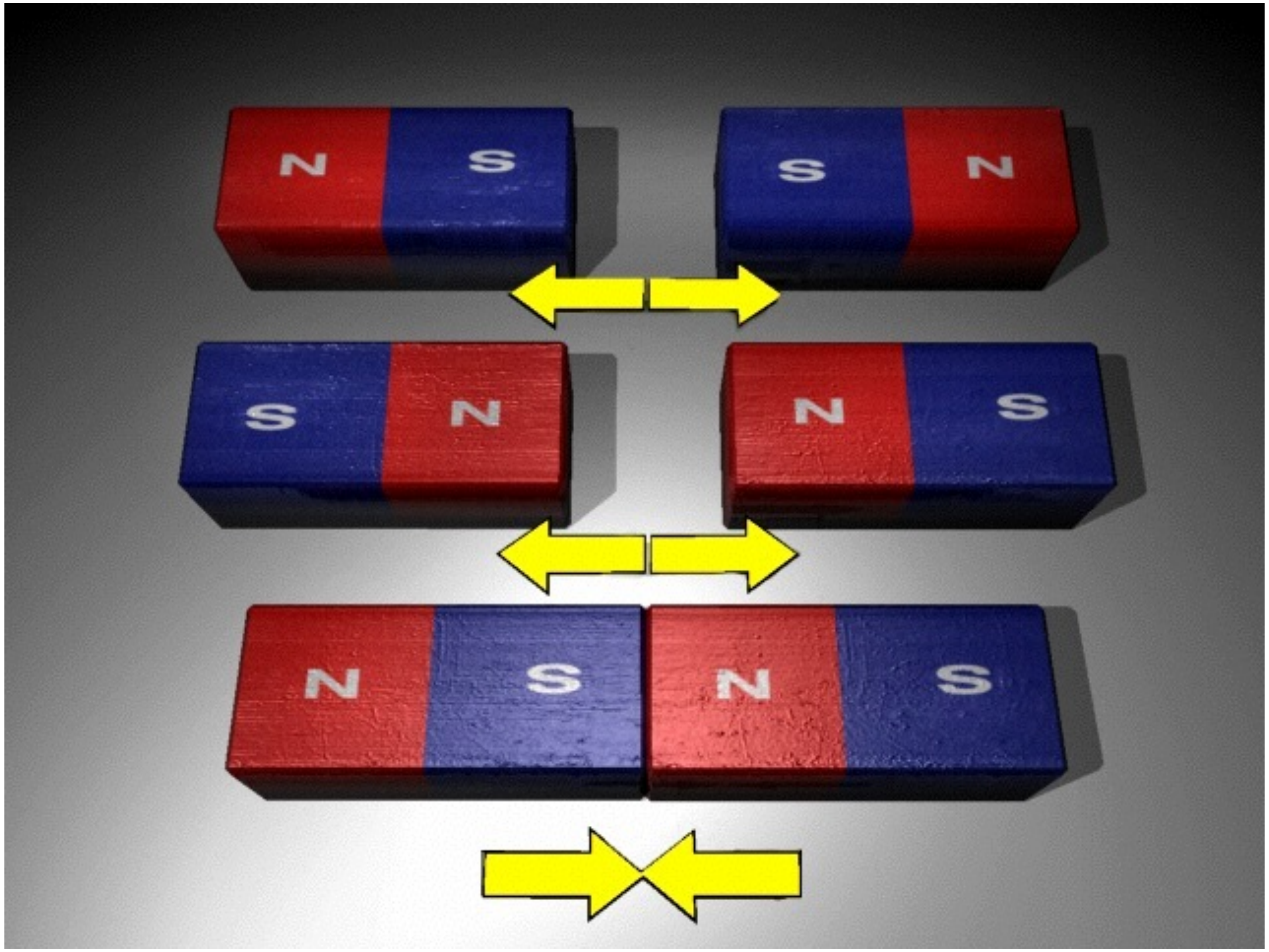
ETH zürich



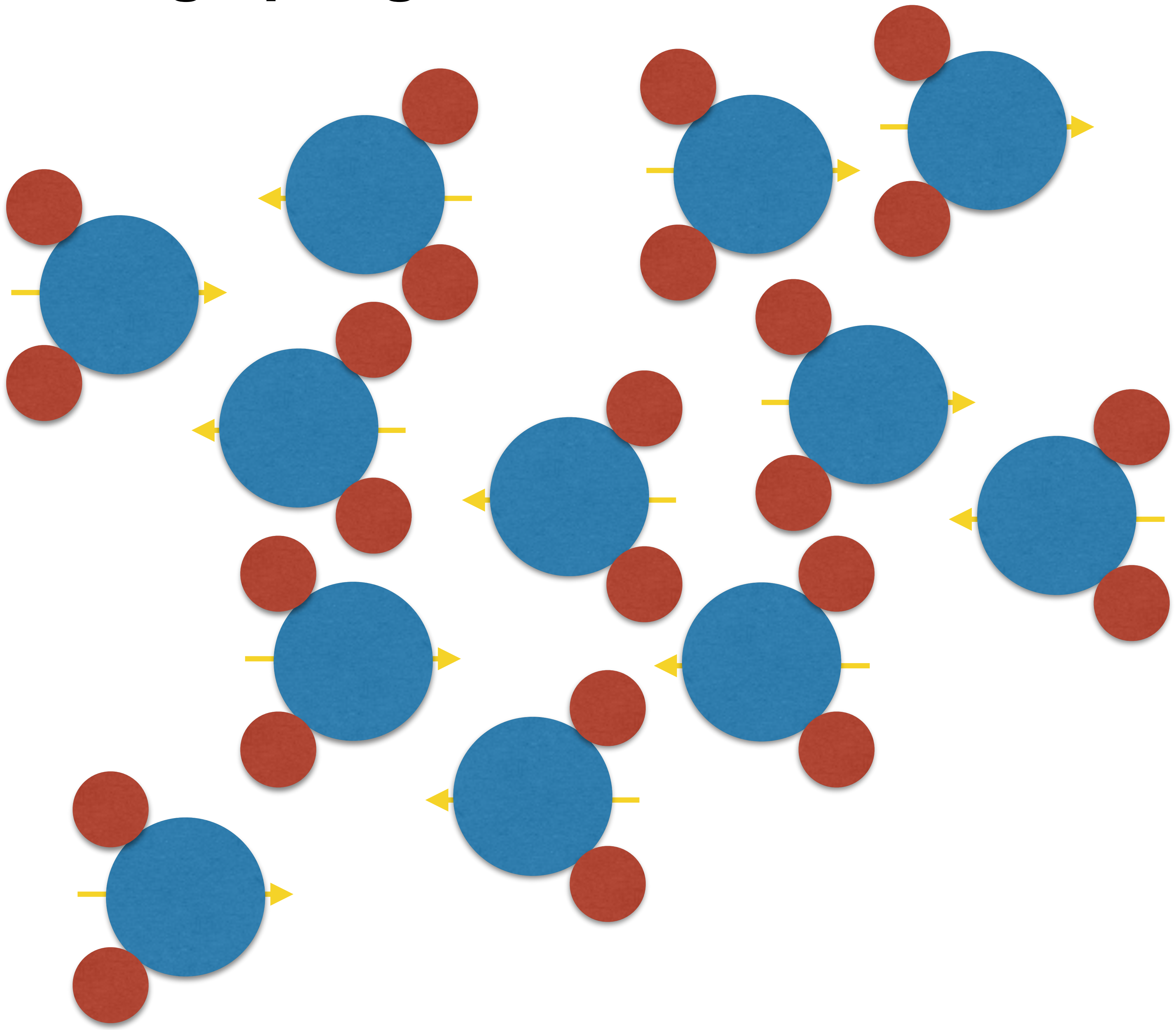
Case study: Simulated annealing

Troels F. Rønnow

What are Ising spin glasses?



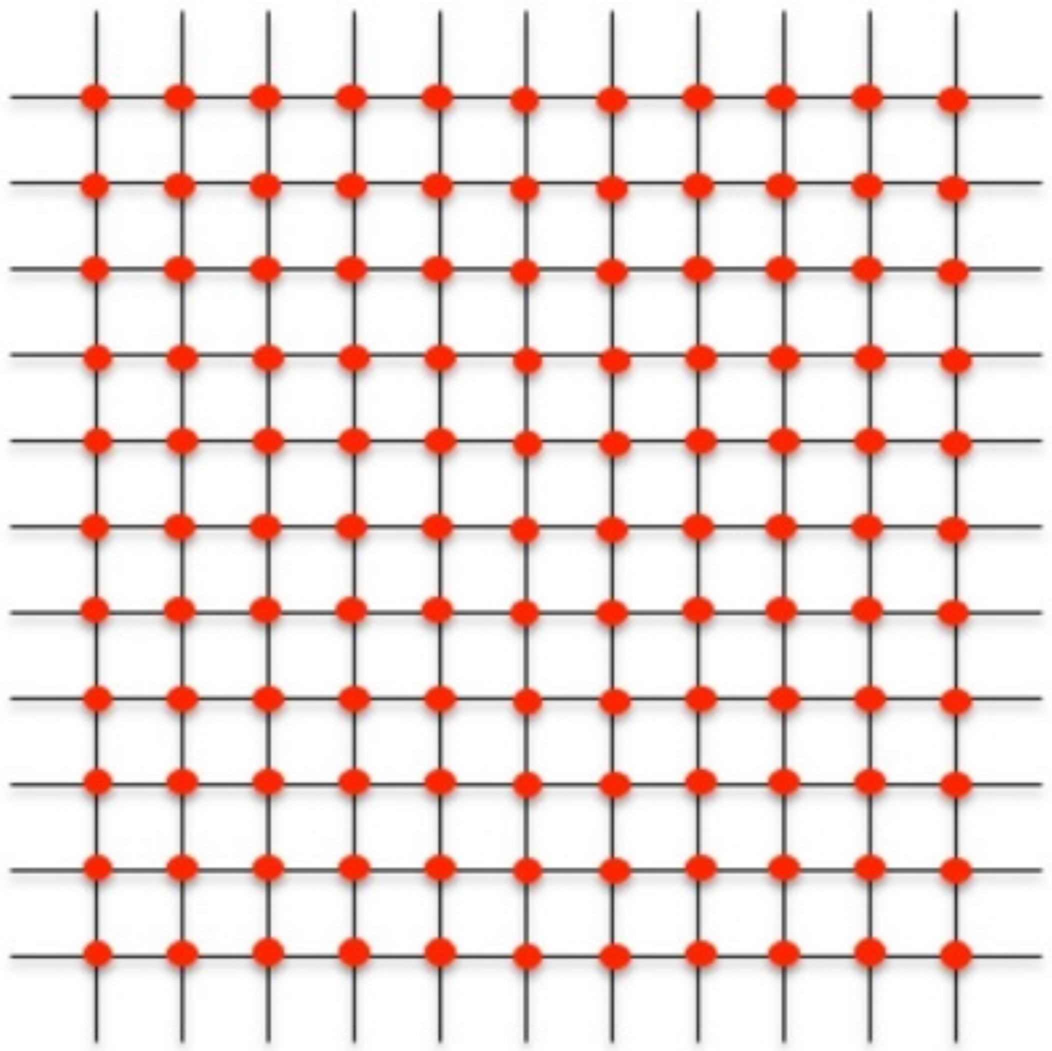
What are Ising spin glasses?



What are Ising spin glasses?

What are Ising spin glasses?

What are Ising spin glasses?



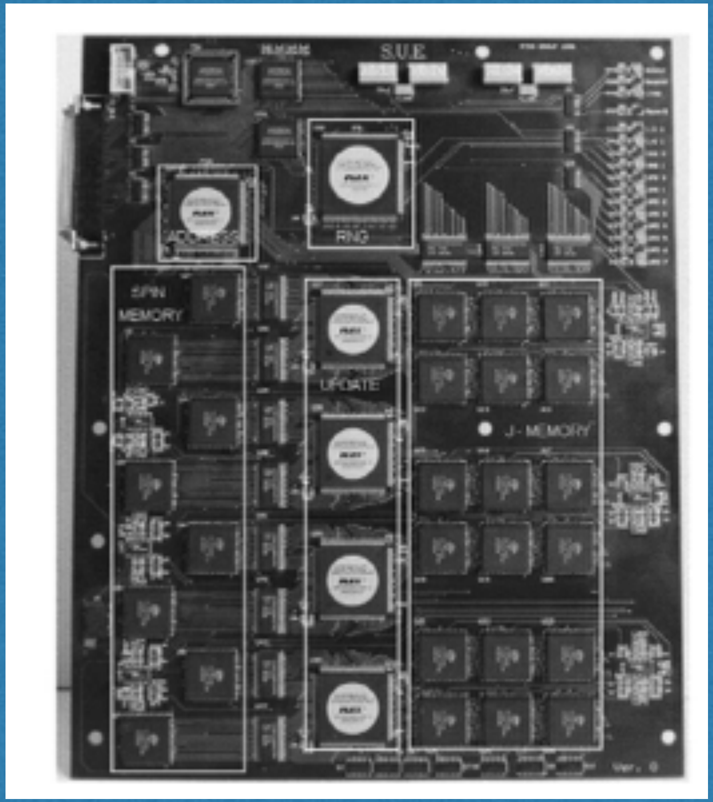
The energy functional of spin glasses are given by:

$$H = \sum_{ij} J_{ij} s_i s_j + \sum_i h_i s_i + const. \qquad \text{with} \qquad s_i = \pm 1$$

Finding the minimum of this functional is NP-hard and therefore have many potential applications including:

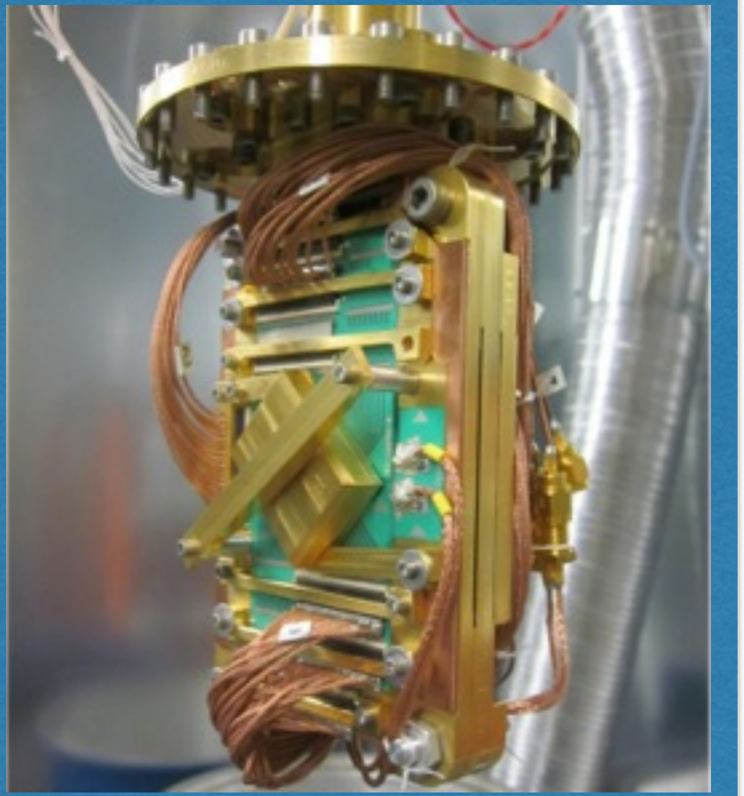
- Travelling salesman problem
- Knapsack problem
- Finishing Super Mario 3 in best possible time

Special purpose machines



SUE

Janus II



D-Wave II

These are fast special purpose machines which find the minimum of the previous functional. Moreover, they are fast and it is hard to write codes for ordinary computers which can compete.

So can classical computers compete with these machines?

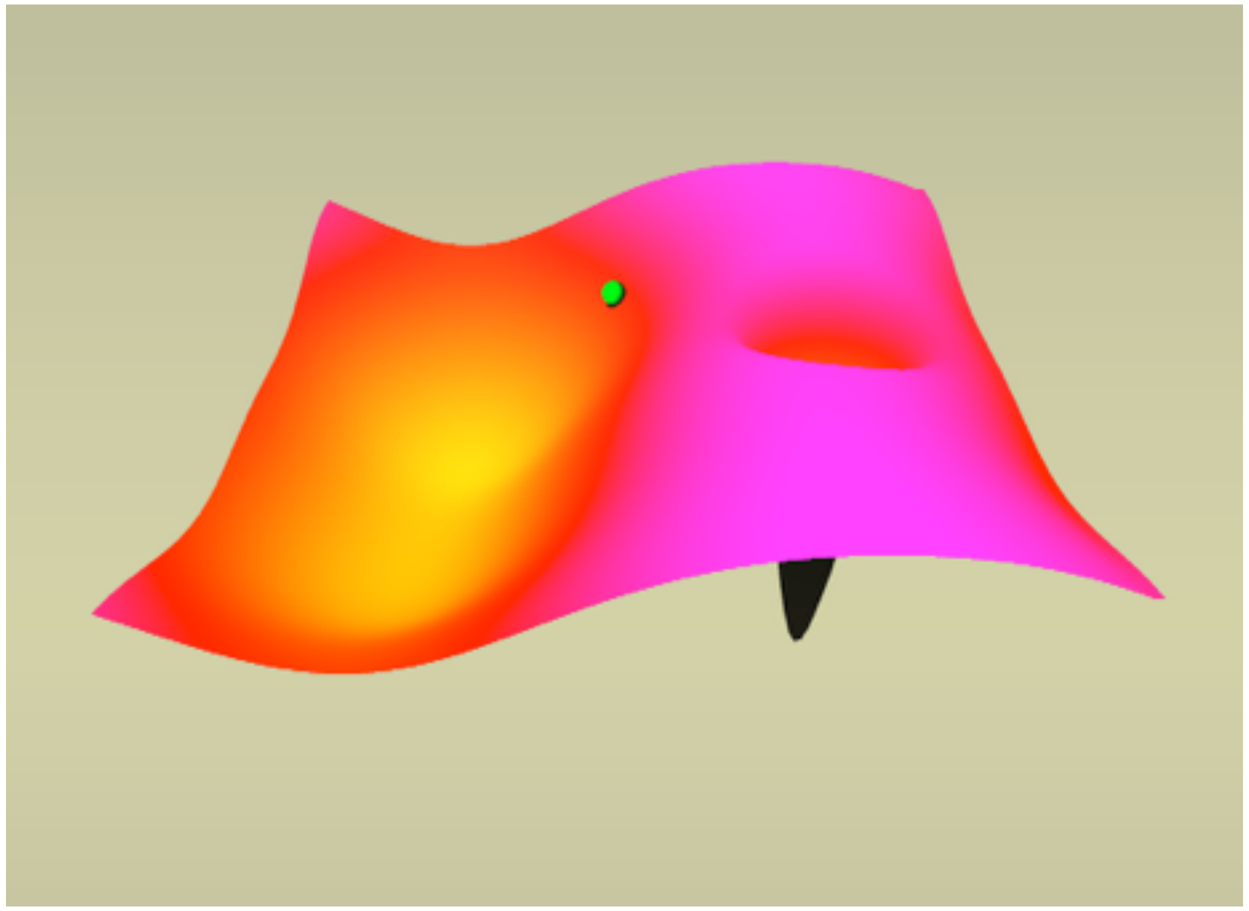
ETH zürich

Annealing and simulated annealing

Annealing

A 7000 year old neolithic technology

Slowly cool metal or glass
to improve its properties



Simulated annealing

Kirkpatrick, Gelatt and Vecchi, Science (1983)

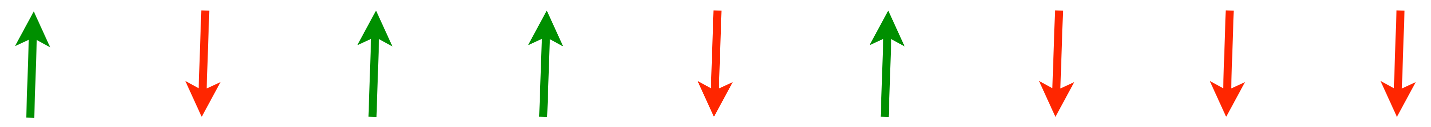
A 30 year old optimisation technique

Slowly cool a model in a Monte Carlo simulation
to find the solution to an optimisation problem

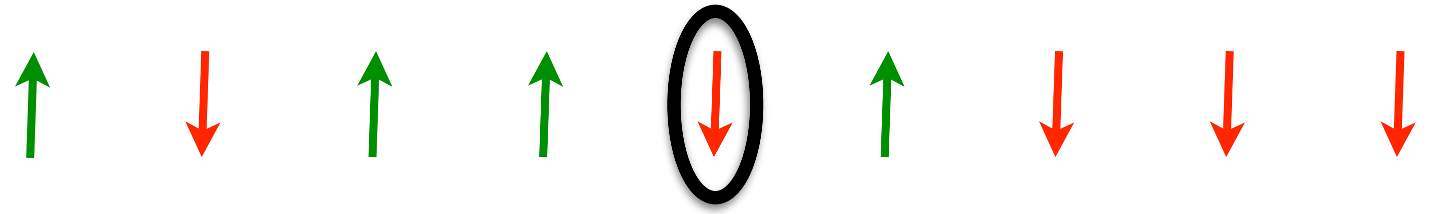
We don’t always find the global minimum and have to try many times

How does simulated annealing on spin glasses work?

1) Start at a random configuration

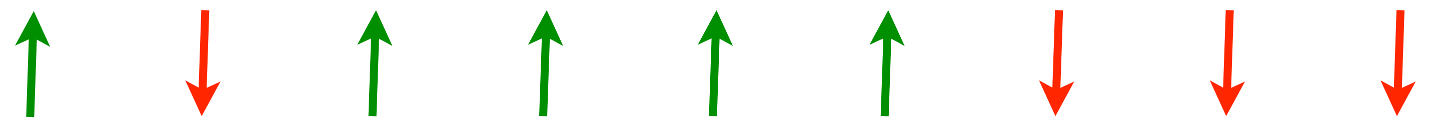


2) Pick a random spin



3) If the energy is lowered by flipping it, flip it.

Otherwise flip it with probability



4) Repeat this many times while gradually lowering the temperature.

ETH zürich

The most “naive” implementation

- Open Matlab and write 20 lines of code:

```
function ising(N, sweeps)
  M = N*N;
  odd = 1:2:M;
  even = odd + 1;
  S = 1-2*round(rand(1,M));
  Ju = (1 - 2*round(rand(M,M))) .* (diag(mod(1:(N*N-1), N) ~= 0,1) \
                                                 + diag(ones(M-N,1),N));
  J = (Ju + Ju');
  for beta= 0.01:(3.0 - 0.01)/(sweeps-1):3.0,
    r = rand(M,1);
    E = (J * S') .* S';
    U = r <= exp(- 2* beta* E);
    U(even) = 0;
    S(U ) = -S( U );

    E = (J * S') .* S';
    U = r <= exp(- 2* beta* E);
    U(odd) = 0;
    S( U ) = -S( U );
  end
end
```

Benchmarking

	Spin flips/ns	Relative speedup
Matlab, “naive” code	0.001	1

What does the code do?

$$J = \begin{pmatrix}
0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0
\end{pmatrix}$$

$$E_L = (J \cdot \vec{s})$$

The “high school” implementation

- In high-school I did not know about matrices and neither about Matlab - however, I knew C++.
- The natural approach is to implement sparse matrices:

```
inline void update_site(site &csite, schedule_step const &sched, word const
&r) {
  energy de = csite.h;
  for(std::size_t i = 0, j; i < csite.neighbour_count(); ++i) {
    j = csite.index[i];
    de -=  hudt * csite.couplings[i] * lattice.sites[j].spin ;
  }
  de *= csite.spin;

  if( de <= 0 || rnd() < word(-1) * std::exp( -2 * sched.beta * de ) ) {
    reference_energy +=  2 * de;
    csite.spin = -csite.spin;
  }
}
```

Benchmarking

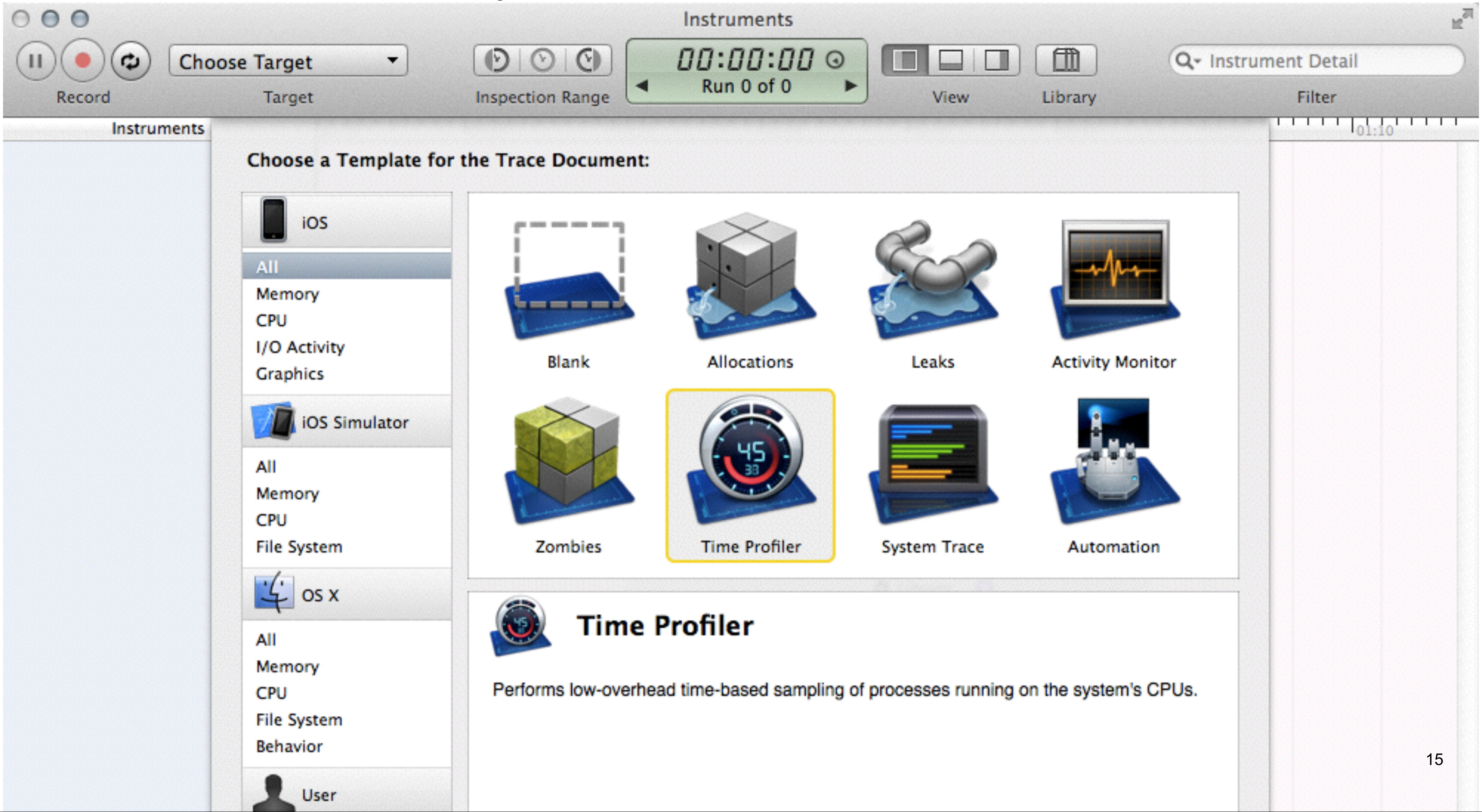
	Spin flips/ns	Relative speedup
Matlab, "naive" code	0.001	1
C++, high-school	0.04	40

Keep It Simple, Stupid: Fancy datatypes and high-level languages are likely to slow your code significantly down.

ETH zürich

Identifying bottle necks

- How do we identify bottlenecks?



Choose a Template for the Trace Document:

Time Profiler

Performs low-overhead time-based sampling of processes running on the system's CPUs.

Optimisations

```
inline void update_site(site &csite, schedule_step const &sched, word const &r) {
  energy de = csite.h;
  for(std::size_t i = 0, j; i < csite.neighbour_count(); ++i) {
    j = csite.index[i];
    de -= csite.couplings[i] * lattice.sites[j].spin ;
  }
  de *= csite.spin;

  if( de <= 0 || rnd() < word(-1) * std::exp( -2 * sched.beta * de ) ) {
    reference_energy +=  2 * de;
    csite.spin = -csite.spin;
  }
}
```

Optimisations

```
inline void update_site(site &csite, schedule_step const &sched, word const &r) {
  if( csite.de <= 0 || rnd() < word(-1) * std::exp( -2 * sched.beta * csite.de )  )
{
    reference_energy +=  ( 2 * csite.de );
    csite.spin = -csite.spin;

    csite.de = -csite.de;
    energy nudt = 2 * csite.spin;
    for(std::size_t i = 0, j; i < csite.neighbour_count(); ++i) {
      j = csite.index[i];
      lattice.sites[j].de -=  nudt * csite.couplings[i] * lattice.sites[j].spin ;
    }
  }
}
```

Optimisations

```
inline void update_site(site &csite, schedule_step const &sched, word const &r) {
  if( csite.de <= 0 || rnd() < sched.levels[csite.de] ) {
    reference_energy +=  ( 2 * csite.de );
    csite.spin = -csite.spin;

    csite.de = -csite.de;
    energy nudt = 2 * csite.spin;
    for(std::size_t i = 0, j; i < csite.neighbour_count(); ++i) {
      j = csite.index[i];
      lattice.sites[j].de -=  nudt * csite.couplings[i] * lattice.sites[j].spin ;
    }
  }
}
```

Benchmarking

	Spin flips/ns	Relative speedup
Matlab, “naive” code	0.001	1
C++, high-school	0.04	40
C++, optimised	0.5	500

Profile your code: This helps you identify where to improve your code and sometimes you get a factor of 10 with little extra effort.

Choosing the correct datatypes

- Currently we use integers to store spins. However, spins are really binary variables:

$$s_i = 1 - 2b_i$$

- In this way we can optimise memory usage by storing in spins in single bits.
- Using binary operations we can update several spins simultaneously and thereby optimise the computational effort:

$$b_i = b_i \oplus u$$

Computing update rates

- We compute the energies bitwise

s_0 0

$l_1 = d_1 \oplus d_2 \qquad l_2 = d_3 \oplus d_4$

$h_1 = d_1 \wedge d_2 \qquad h_2 = d_3 \wedge d_4$

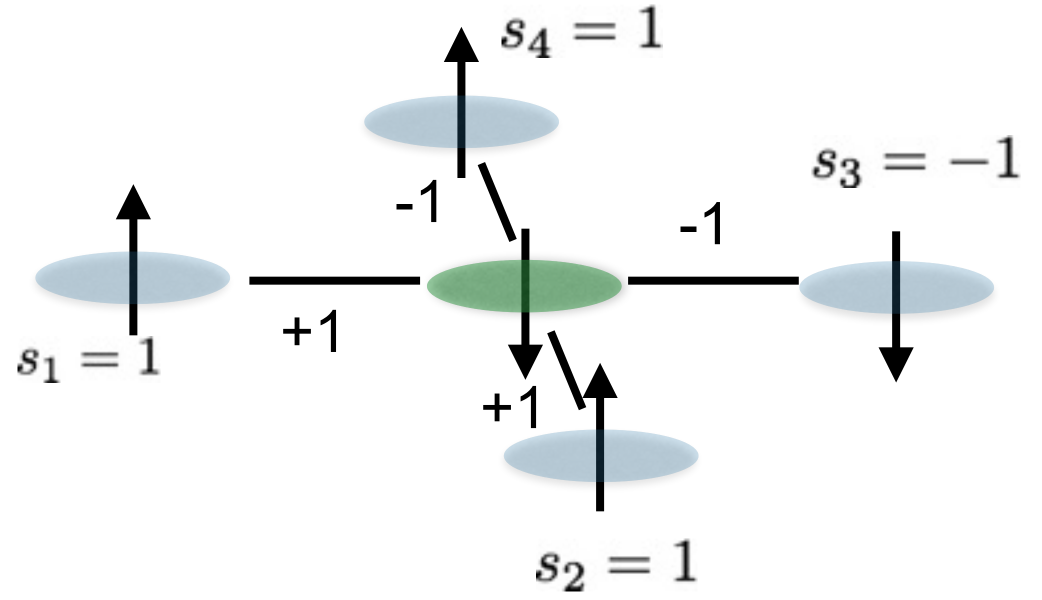
$e_1 = l_1 \oplus l_2$

$e_2 = (l_1 \wedge l_2) \oplus h_1 \oplus h_2$

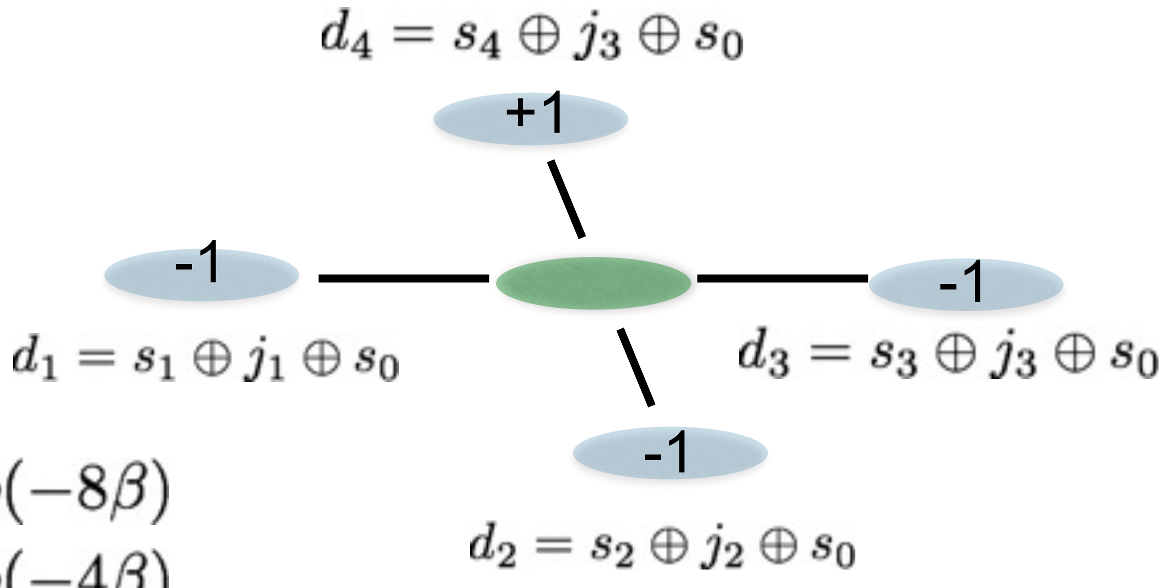
$e_3 = (l_1 \wedge l_2) \wedge (h_1 \vee h_2) \vee h_1 \wedge h_2$

$e_4 = h_1 \wedge h_2$

Current configuration



Effective energy contributions



$d_4 = s_4 \oplus j_3 \oplus s_0$
+1
-1
-1
-1
$d_1 = s_1 \oplus j_1 \oplus s_0$
$d_3 = s_3 \oplus j_3 \oplus s_0$
$d_2 = s_2 \oplus j_2 \oplus s_0$

Result:

$= -(\Delta E/2 + 2)$

e_1 1 $\leftarrow p_1 = exp(-8\beta)$

e_2 1 $\leftarrow p_2 = exp(-4\beta)$

e_3 0 $\leftarrow p_3 = 1$

e_4 0 $\swarrow$

Now all we need to do is to compute the probability of flipping each spin in the word.

You can efficiently compute bits with a given probability using the right algorithm

ETH zürich

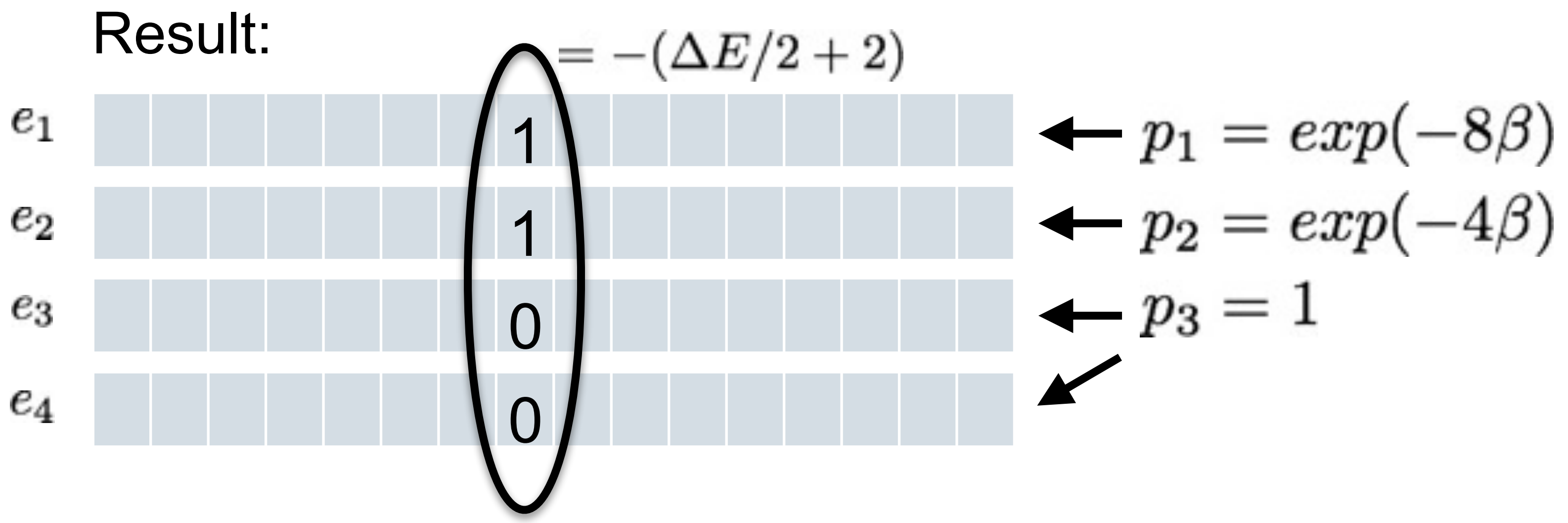
Benchmarking

	Spin flips/ns	Relative speedup
Matlab, “naive” code	0.001	1
C++, high-school	0.04	40
C++, optimised	0.5	500
C++, multispin version 1	2.3	2.300

Put thought into datatypes and the underlying algorithm: Choosing the correct datatypes with the correct algorithm, we can improve the code.

Checking the literature

- Reading the literature, we later found a more effective way to compute the probability of updating a spin.
- This is just a small modification to our previous algorithm.



This gives correlations, but it turns out that they are negligible in most cases.

Benchmarking

	Spin flips/ns	Relative speedup
Matlab, "naive" code	0.001	1
C++, high-school	0.04	40
C++, optimised	0.5	500
C++, multispin version 1	2.3	2.300
C++, multispin version 2	6.0	6.000

Always check literature:
You are not the first to consider a specific problem. With more than 50 years digital computing in academia, great ideas are around - use them!

Use OpenMP and OpenMPI

- For many codes you have one or two loops which can be made parallel in a straight-forward manner.
- The individual repetitions can be computed in parallel

```
alg = alg_type(lattice, sched);

/* ... */

for (std::size_t rep = rep0; rep < rep0+nreps; ++rep) {
  alg.reset_sites(rep);
    /* ... */
}
```

OpenMP for repetitions

It only requires few lines of code:

```
#pragma omp parallel num_threads(n) {
  unsigned m = omp_get_thread_num();
  algs[m] = alg_type(lattice, sched);
}

/* ... */

#pragma omp parallel num_threads(n) {
  unsigned n = omp_get_num_threads();
  unsigned m = omp_get_thread_num();
  std::size_t r0 = rep0 + nreps * m / n;
  std::size_t r1 = rep0 + nreps * (m + 1) / n;
  std::size_t offs = nreps * m / n * alg_type::word_size;
  for (std::size_t rep = r0; rep < r1; ++rep) {
    algs[m].reset_sites(rep);
    /* ... */
  }
}
```

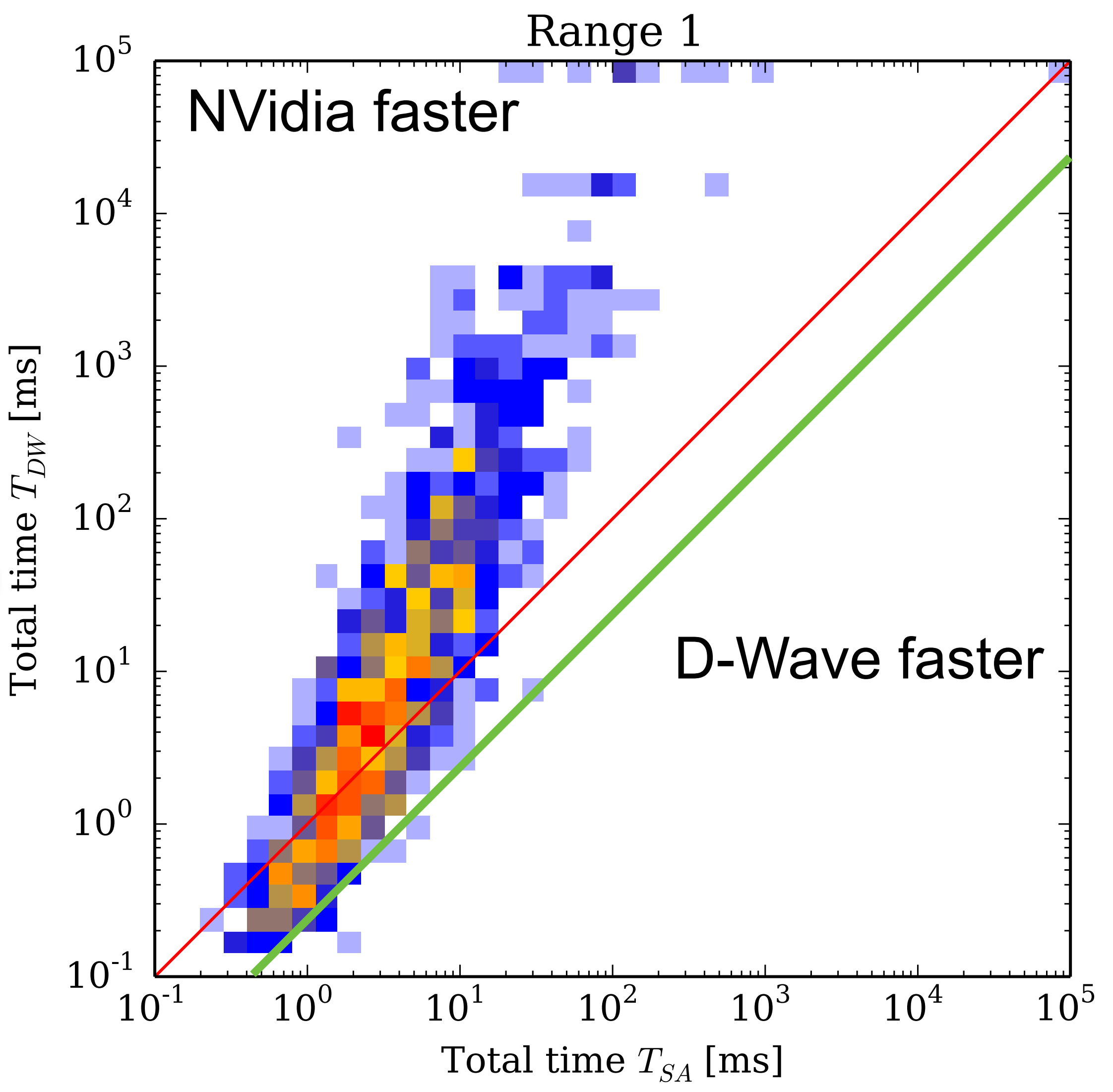
Benchmarking

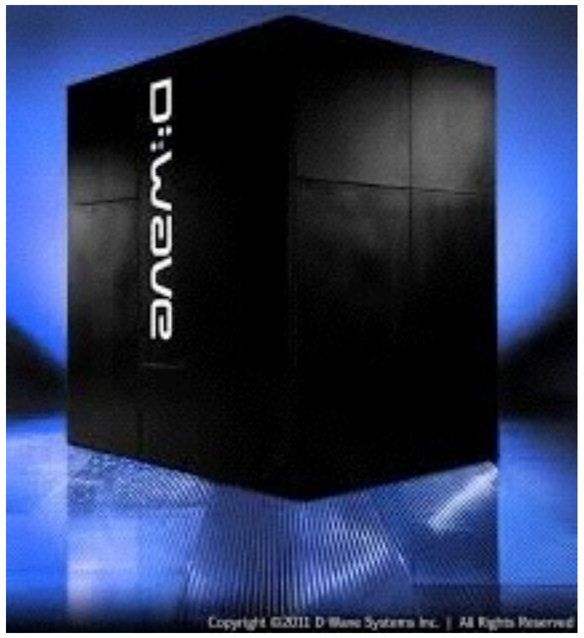
	Spin flips/ns	Relative speedup
Matlab, "naive" code	0.001	1
C++, high-school	0.04	40
C++, optim		
C++, multis version 1		
C++, multispin version 2	6.0	6.000
C++, parallel, multispin	52.0	52.000

Learn to write parallel codes: Exploit the full CPU and not just a fraction of it.

Moving to GPUs

- Rewrite the entire code for NVIDIA Kepler cards
- … and finally we are ready to compete with the special purpose machines:





Benchmarking

	Spin flips/ns	Relative speedup
Matlab, “naive” code	0.001	1
C++, high-school	0.04	40
C++, optim		
C++, multis version 1		
C++, multispin version 2	6.0	6.000
C++, parallel, multispin	52.0	52.000
CUDA, GPU version	250	250.000

Hire a programmer: If you don’t have time or skills to optimise your codes, it might be worth hiring someone to do it.

Conclusion

	Price for one billion simulations	Development cost
Matlab, “naive” code	CHF 300.000	CHF 60 (2 hours)
C++, high-school	CHF 7.500	CHF 300 (10 hours)
C++, optimised	CHF 600	CHF 600 (20 hours)
C++, multispin version 1	CHF 130	CHF 2.400 (2 weeks)
C++, multispin version 2	CHF 50	CHF 4.800 (1 month)
C++, parallel, multispin	CHF 50	CHF 4.860 (1 month, 2 hours)
CUDA, GPU version	CHF 20	CHF 9.600 (2 months)

Conclusion

- Fancy datatypes will not do you any good if there is no thought behind them.
- Use a profiler to optimise your codes.
- Check what has been done by other people.
- If you cannot write good and fast codes yourself, you might want to consider hiring someone to do that.