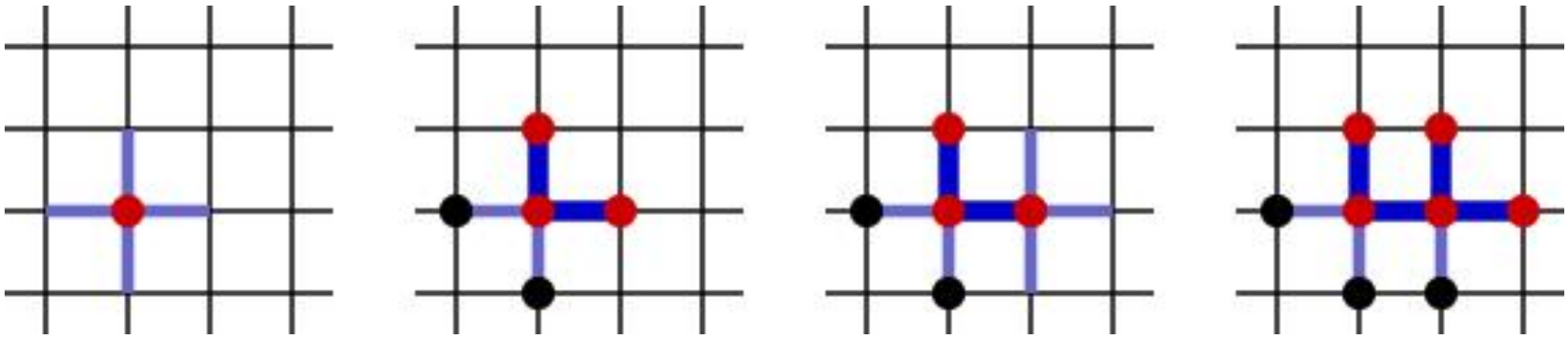
Cluster finding/flipping

Clusters can be constructed and flipped in the same process

➢ Decide whether or not to flip (50% probability) before starting

Store array with flags for spins visited

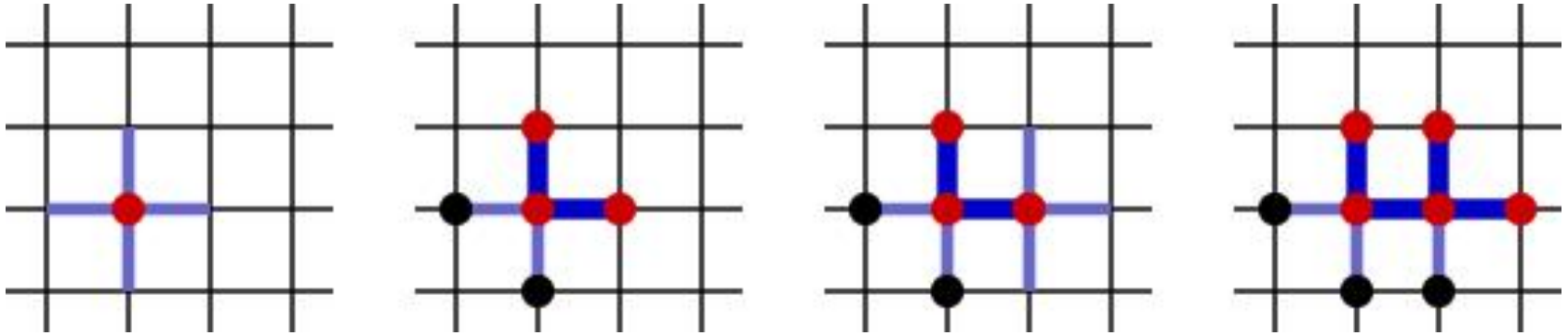
- Start with spin that has not been visited; seed of cluster
- Add connected (by filled bonds) neighbors to cluster
- Examine the non-visited neighbors of the new spins added
- Add connected neighbors to cluster
- Until no more spins in the cluster with non-visited neighbors



Use stack to store spins with neighbors to be examined

Wolff cluster algorithm

Construct a single cluster, starting from randomly chosen "seed" spin. Flip the cluster with probabiolity 1.



Cluster construction same as in Swendsen-Wang, but generate filled/non-filled bonds at the same time as the neighbors of added spins are examined.

The Wolff clusters are the same as in Swendsen-Wang, but the probability is higher to flip large clusters

Leads to slightly smaller dynamic exponents

The Wolff method is easier to generalize to other models

Generality of Metropolis and cluster methods

The Metropolis method can be used for any system

➢ Critical slowing down can be serious

➢ The dynamics can be slow also in non-critical systems,
e.g., in systems with "glassy" behavior

Cluster algorithms have been developed for many systems,
but there are still no working cluster methods for

➢ Magnets with frustration (competing interactions)
- the clusters span the whole system (percolate)
before the critical point is reached ($T > T_c$)

➢ Magnets in external magnetic fields

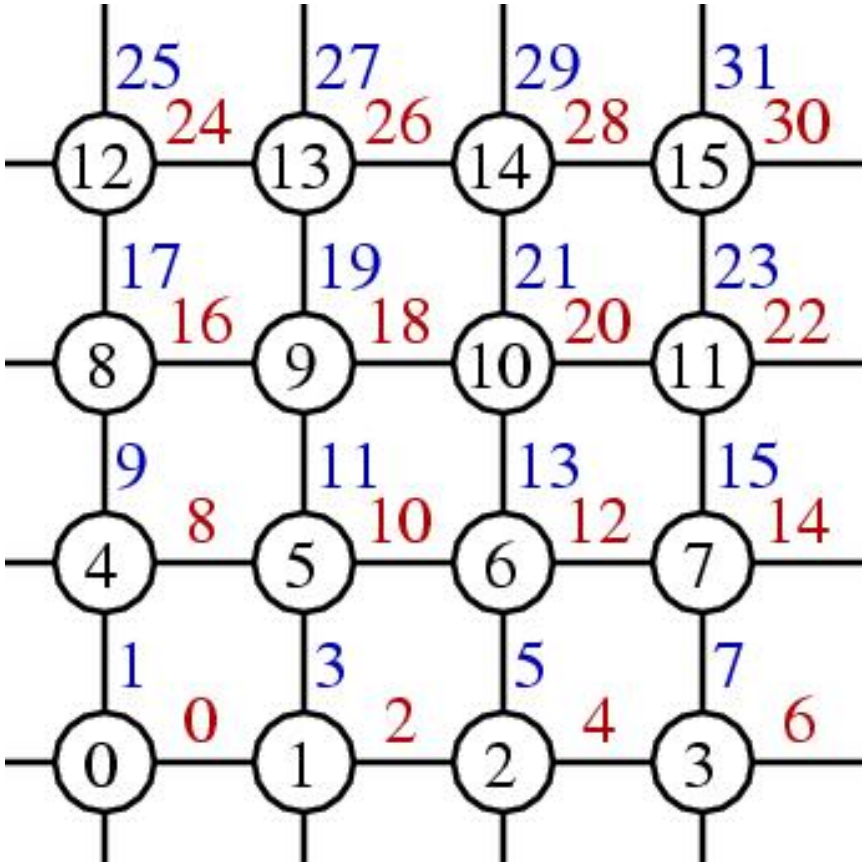
➢ Most systems of particles in continuous space
- Not known how to construct clusters in general

Programming the Swendsen-Wang algorithm

To construct clusters, we need arrays containing

- Neighbors of given site s: `neighbor[i,s]`
- Two spins connected by given bond b: `bondspin[i,b]`
- Bonds connected to given spin s: `spinbond[i,s]`

Labeling of spins and bonds; example in 2D



Note: in Julia the labels have to start from 1, adjust accordingly

Storing spin and bond variables in one-dimensional vectors

```
spin[1:n], bond[1:d*n]
```

Construction of lattice arrays in 2D

```
subroutine lattice

for s0=1:n
   x0=mod(s0-1,ll)
   y0=div(s0-1,ll
   x1=mod(x0+1,ll)
   x2=mod(x0-1,ll)
   y1=mod(y0+1,ll)
   y2=mod(y0-1,ll)
   s1=1+x1+y0*ll
   s2=1+x0+y1*ll
   s3=1+x2+y0*ll
   s4=1+x0+y2*ll
   neighbor[1,s0]=s1
   neighbor[2,s0]=s2
   neighbor[3,s0]=s3
   neighbor[4,s0]=s4
   bondspin[1,2*s0]=s0
   bondspin[2,2*s0]=s1
   bondspin[1,2*s0+1]=s0
   bondspin[2,2*s0+1]=s2
   spinbond[1,s0]=2*s0
   spinbond[2,s0]=2*s0+1
   spinbond[3,s1]=2*s0
   spinbond[4,s2]=2*s0+1
end do
```

Main program

```
bprob=1.d0-exp(-2.d0/temp)
 for i=1:div(steps,4)
    castbonds()
    flipclusters()
 end
 for j=1:bins
    resetbindata()
    for i=1:steps
       castbonds()
       flipclusters()
       measure()
    end
    writebindata(n,steps)
 end
```

Generating bond configuration

```
function castbonds()

for b=1:2^n
   if spin[bondspin[1,b]]==spin[bondspin[2,b]]
      if ran()<=bprob
         bond[b]=true
      else
         bond[b]=false
      end
   else
      bond[b]=F
   end
end
```

For cluster finding/flipping, see program sw.jl

Construct/flip clusters

`notvisited[s] = T` for sites not yet visited

`notvisited[s] = F` for sites that have been visited

```
function flipclusters()
Notvisited[:] .= T
cseed=1

 1 if (ran()<0.5d0) then
    flipclus=.true.
 else
    flipclus=.false.
 endif

 notvisited(cseed)=.false.
 if (flipclus) spin(cseed)=-spin(cseed)
 nstack=1
 stack(1)=cseed
    ......
```

```
do
    if (nstack==0) exit
    s0=stack(nstack)
    nstack=nstack-1
    do i=1,nbors
       s1=neighbor(i,s0)
       if (bond(spinbond(i,s0)).and.notvisited(s1)) then
          notvisited(s1)=.false.
          if (flipclus) spin(s1)=-spin(s1)
          nstack=nstack+1
          stack(nstack)=s1
       endif
    enddo
enddo

do i=cseed+1,n-1                ! find starting spin
    if (notvisited(i)) then     ! for the next cluster
       cseed=i
       goto 1
    endif
enddo
```