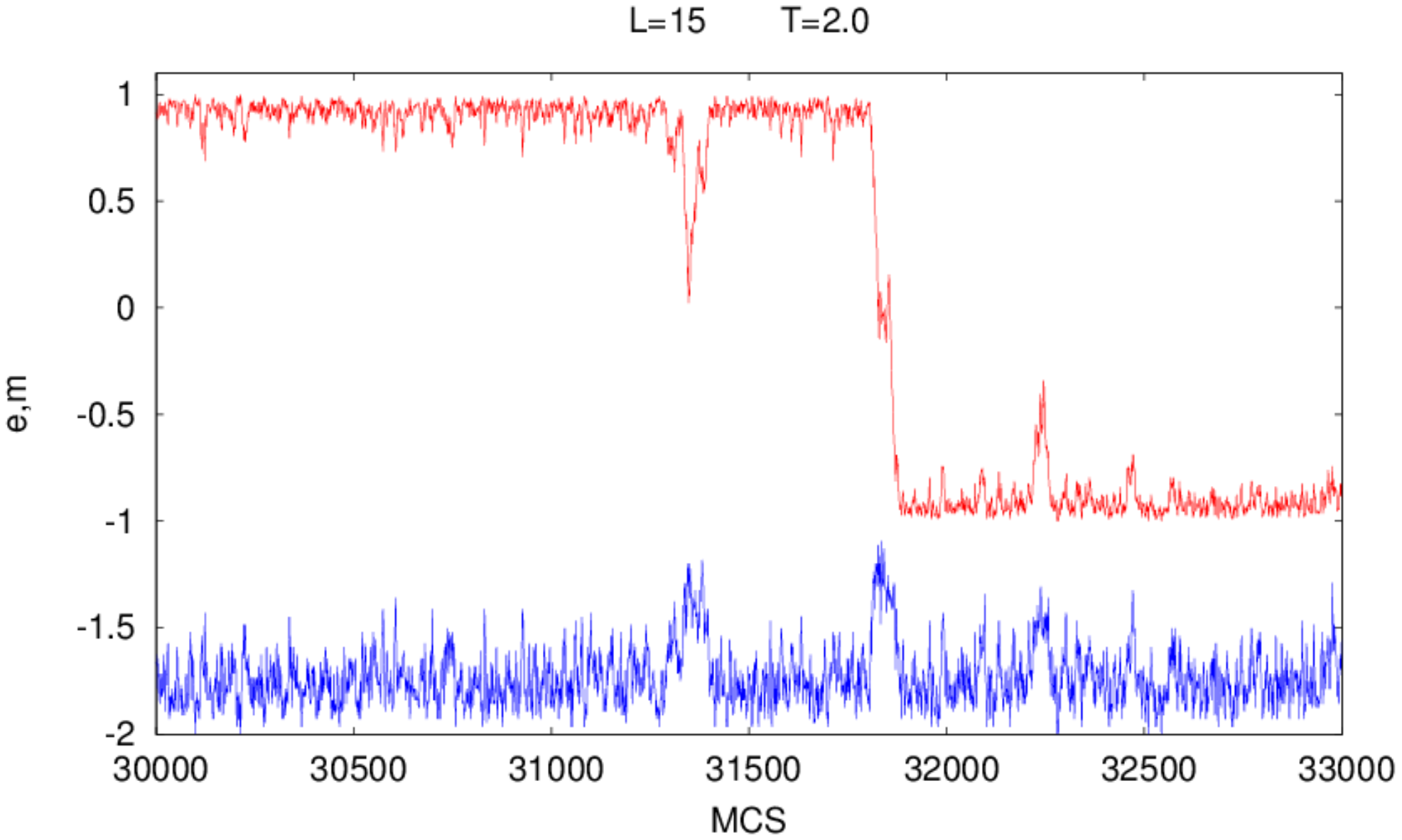
III ENCONTRO DE PÓS-GRADUAÇÃO EM FÍSICA E ASTRONOMIA DA UFSC

Métodos Computacionais em Simulações de Monte Carlo

Prof. Lucas Nicolao

PPGFSC

Barreiras de energia



Aula 2

- Implementação do algoritmo de Metropolis para o modelo de Ising e aceleração do tempo de cálculo.
- Protocolos e boas práticas em simulações. Fenomenologia.
- Alternativas.

Um passo do algoritmo de Metropolis

1) Escolhemos aleatoriamente um spin s_k para inverter → nova configuração proposta $\vec{s}'$ difere da antiga apenas por esse spin *(single spin fli*p dynamics).

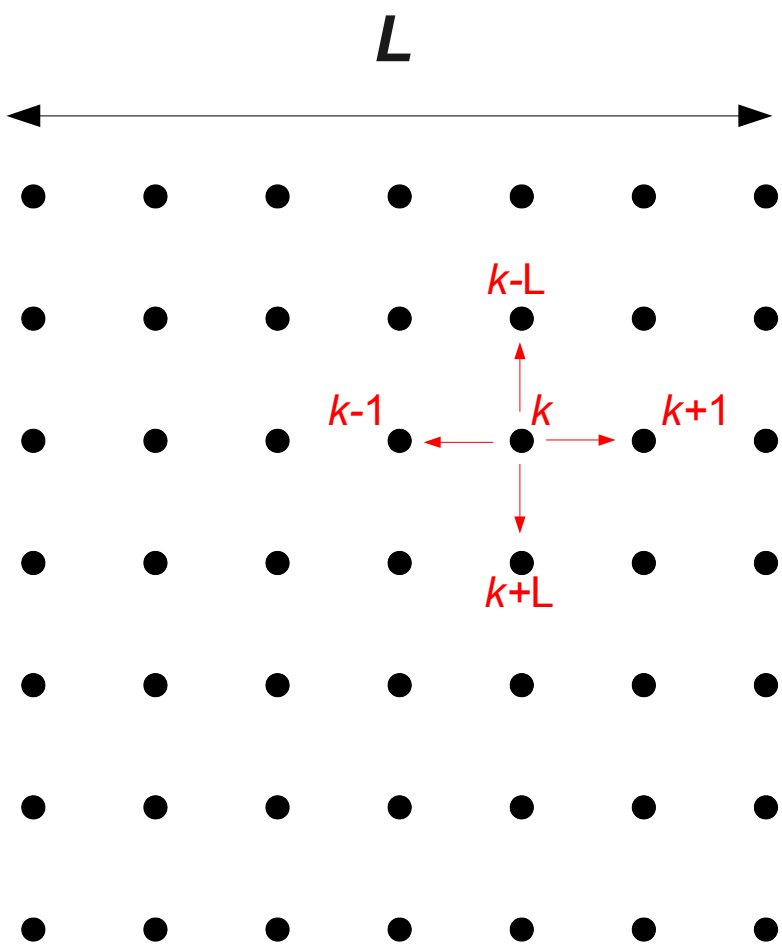
2) Calculamos a diferença de energia $\Delta E = E(\vec{s}') - E(\vec{s})$.

3) Se $\Delta E \leq 0$, a nova configuração é aceita e o passo temporal é finalizado.

4) Senão, extraímos um no. aleatório ***r*** uniforme em [0,1]:

- Se $r \leq \exp(-\Delta E / T)$ aceitamos a nova configuração $\vec{s}'$.
- Se $r > \exp(-\Delta E / T)$ rejeitamos a nova configuração.

Para o caso do modelo de Ising 2D



$$E(\vec{s}) = -\sum_{\langle i,j\rangle \neq k} s_i s_j - s_k(s_{k+1} + s_{k-1} + s_{k+L} + s_{k-L})$$

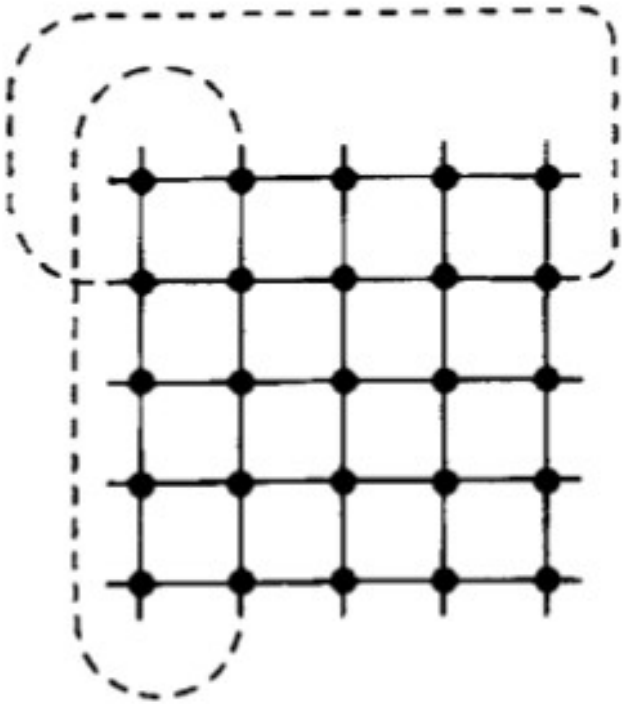
$$E(\vec{s'}) = -\sum_{\langle i,j\rangle \neq k} s_i s_j + s_k(s_{k+1} + s_{k-1} + s_{k+L} + s_{k-L})$$

$$\Delta E = E(\vec{s'}) - E(\vec{s}) = 2s_k(s_{k+1} + s_{k-1} + s_{k+L} + s_{k-L})$$

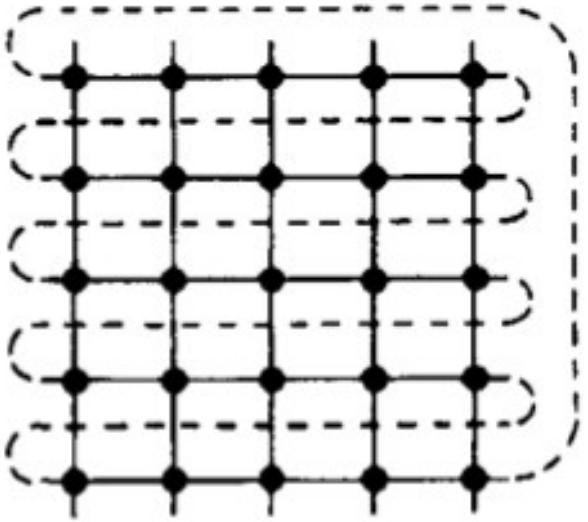
$$\Delta E = (-8, -4, \ 0, +4, +8)$$
$$\in [-2, +2] \times 4$$

$N = L^2$ spins

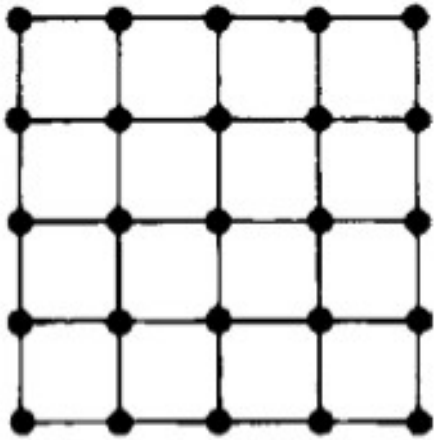
Condições de contorno



CC Periódicas



CC Helicoidais



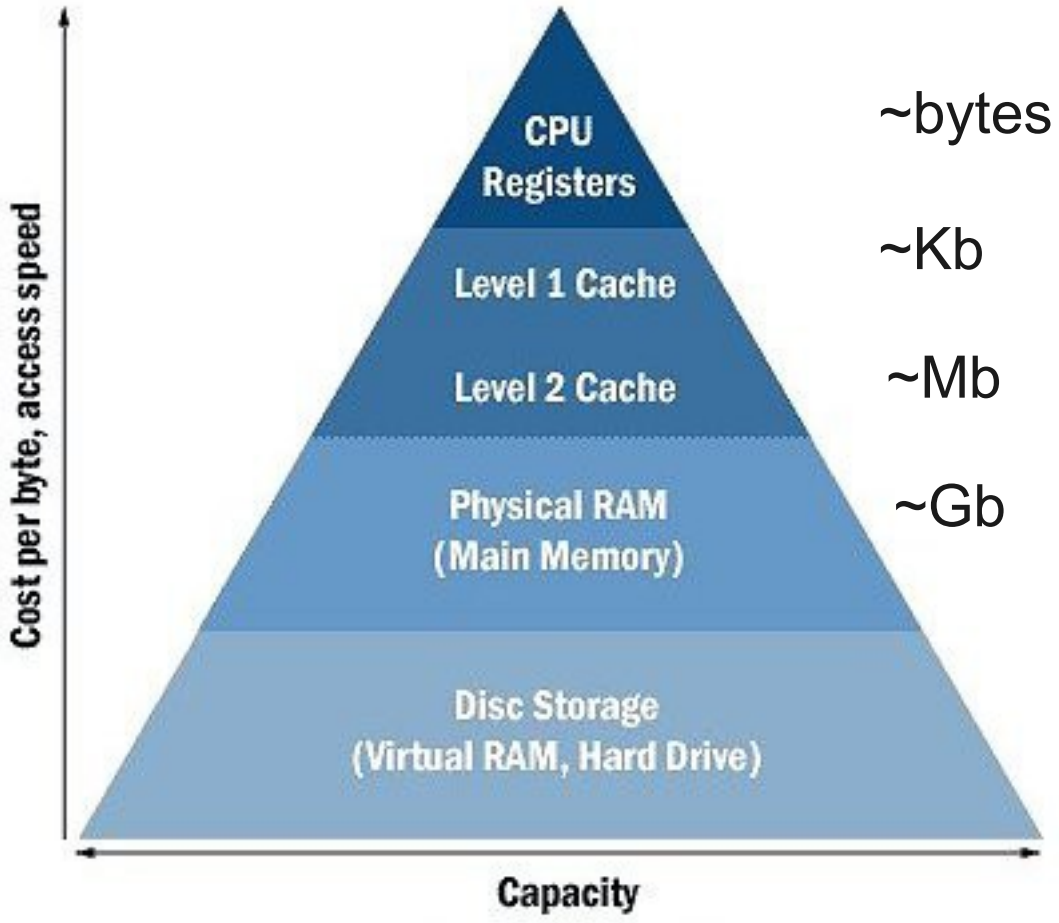
CC livres

Atualização – 1 MCS

```
...
for (iy = 1; iy < L-1; iy++) {
  for (ix = 1; ix < L-1; ix++) {

    site++;

    soma = s[site] * (s[site-L] + s[site-1] + s[site+1] + s[site+L]);

    if (soma <= 0)
      s[site] = -s[site];

    else if( FRANDOM < exp(-2.0*soma/temperatura))
      s[site] = -s[site];

  }
}
...
```

99% do tempo → otimizar

Hierarquia de memória de uma CPU



~bytes

~Kb

~Mb

~Gb

stack

Otimização do código

0 Linguagens de baixo-nível (C, C++ ou Fortran)

I. Evitar chamadas excessivas de funções ou subrotinas

Embora facilite a leitura do código, isso causa “pulos” na execução do programa

a. Toda vez que uma função é chamada, o que estava armazenado nos registers da CPU é copiado para o stack

b. Valores são copiados para as variáveis locais da função e ela é executada

c. Uma vez terminada a execução da função, o conteúdo na stack é copiado de volta para os registers

Exemplo ruim:

```
int energialocal(int s[N], int site){

  ...

  return  s[site] * (s[site-L] + s[site-1]
                   + s[site+1] + s[site+L]);
}

int atualiza(int spin, float DE){
  int newSpin = spin;

  if ( DE < 0.0 )
    newSpin = -spin;
  else if( RANDOM < exp(-2.0 * sum / temperature) )
    newSpin = -spin;

  return newSpin;
}

...

  for (i = 0; i < N; i++) {

    site = FRANDOM * N;

    DE = energialocal(s,site);
    s[site] = atualiza( s[site], DE );
```

→ Utilizar uma única função para fazer todo o passo de Monte Carlo (dos N spins)

OBS: Sem problemas para funções curtas e simples (~ 2 linhas), otimização do compilador e/ou opção *inline* copiam ela para o local em que está sendo chamada

gcc: -finline-functions (incluído na opção -O3)

Otimização do código

II. Minimizar estruturas de seleção

Estruturas *Se... Senão...* podem dificultar o planejamento e otimização da linha de execução pelo compilador (não se sabe o valor da expressão lógica), podendo causar latência do processador.

Ex: Sendo que a variável ***s*** só pode assumir valores +1 ou -1

```
if(s == 1){
  a = 2 * b;
} else {
  a = 2 * c;
}
```

Pode ser substituído por:

```
a = (b + c) + s * (b - c);
```

Otimização do código

III. Uso de lookup tables (pré-calcular exp)

O cálculo da função exponencial é uma operação que toma muitos ciclos do processador (~80). Como só precisamos de dois valores (casos ΔE= +4 e +8), mantendo T= *cte*, podemos calcular esses valores no início do programa.

A chamada de um número aleatório também é muito custosa, podendo ser da mesma ordem que a chamada de exp( ), ou em melhores casos, metade desse tempo – em geral teremos de arcar com esse custo – (algo pode ser economizado em uma atualizações sequencial)

```
for (i = 0; i <= 4; i++ )
  prob[i] = exp( - 2.0 * i / T );

...

soma = s[site] * (s[site-L] + s[site-1]
                  + s[site+1] + s[site+L]);

if (soma <= 0 || FRANDOM < prob[soma])
  s[site] = -s[site];
```

Calculado 1x no início. Sendo poucos valores utilizados toda hora → armazenados nos *registers*

soma pode valer -4,-2,0 ou 2 ou 4

a. Minimizamos a bifurcação de uma estrutura *if...else*.

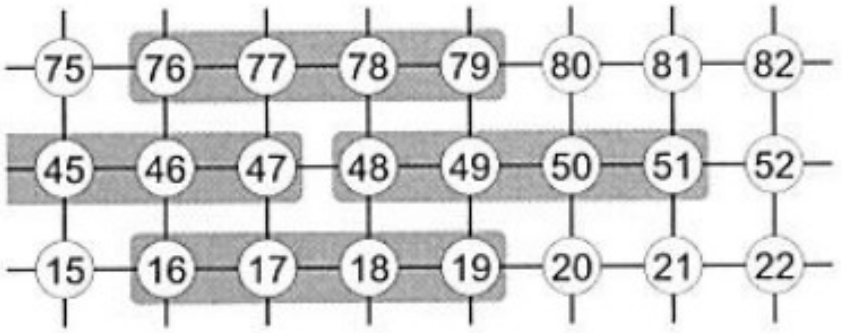
b. A ordem dentro do *if* importa, caso soma<=0, a segunda comparação lógica não é realizada, e nem chamada do número aleatório portanto.

c. Quando usada como índice do vetor, seguramente soma > 0

Otimização do código

IV. Otimizando o acesso a memória: atualização sequencial

- Em geral a transferência de dados da memória RAM/ cache para os *registers* do processador se dá em pacotes.
- Ex: assumindo pacotes de 4x32-bits, atualização s[47]



Nesse caso, os spins s[48] e s[49] e seus vizinhos já estão carregados no processador.

30-50% + rápido

Atualização em ordem sequencial ou aleatória?

Prós: além acesso à memória – menos chamadas de números aleatórios
Contras: dinâmica t~0 | quebra balanço detalhado |problemas T>>1 e T~0.

Otimização do código

V. Sequência de afirmações:

- Evitar uso de vetores para designar vizinhos:

  É comum em simulações o uso de vetores para designar os vizinhos numa rede quadrada com C.C. periódicas. Na hora da compilação o valor dessas variáveis não é conhecida.

```
for (i=0; i < L*L; i++){

  if (i % L == L-1)
    direita[i] = i-L+1;
  else
    direita[i] = i+1;

  if (i >= L*L-L)
    abaixo[i] = i % L;
  else
    abaixo[i] = i + L;
...
```

Otimização do código

V. Sequência de afirmações:

- Escrever explicitamente por extenso em geral → código mais otimizado
- Ou de maneira que compilador possa pré-calcular índices e desenrolar os laços (gcc -funroll-loops, também em -O3)

```
site = 0;
for (iy = 0; iy < L; iy++) {
  for (ix = 0; ix < L; ix++, site++) {
    soma = s[site] * (s[ix + L * ((iy+L-1)%L)] +
                      s[((ix+L-1)%L) + L * iy] +
                      s[ix + L * ((iy+1)%L)] +
                      s[((ix+1)%L) + L * iy]);
    if (sum <= 0 || FRANDOM < prob[soma]) {
      s[site] = -s[site];
    }
  }
}
```

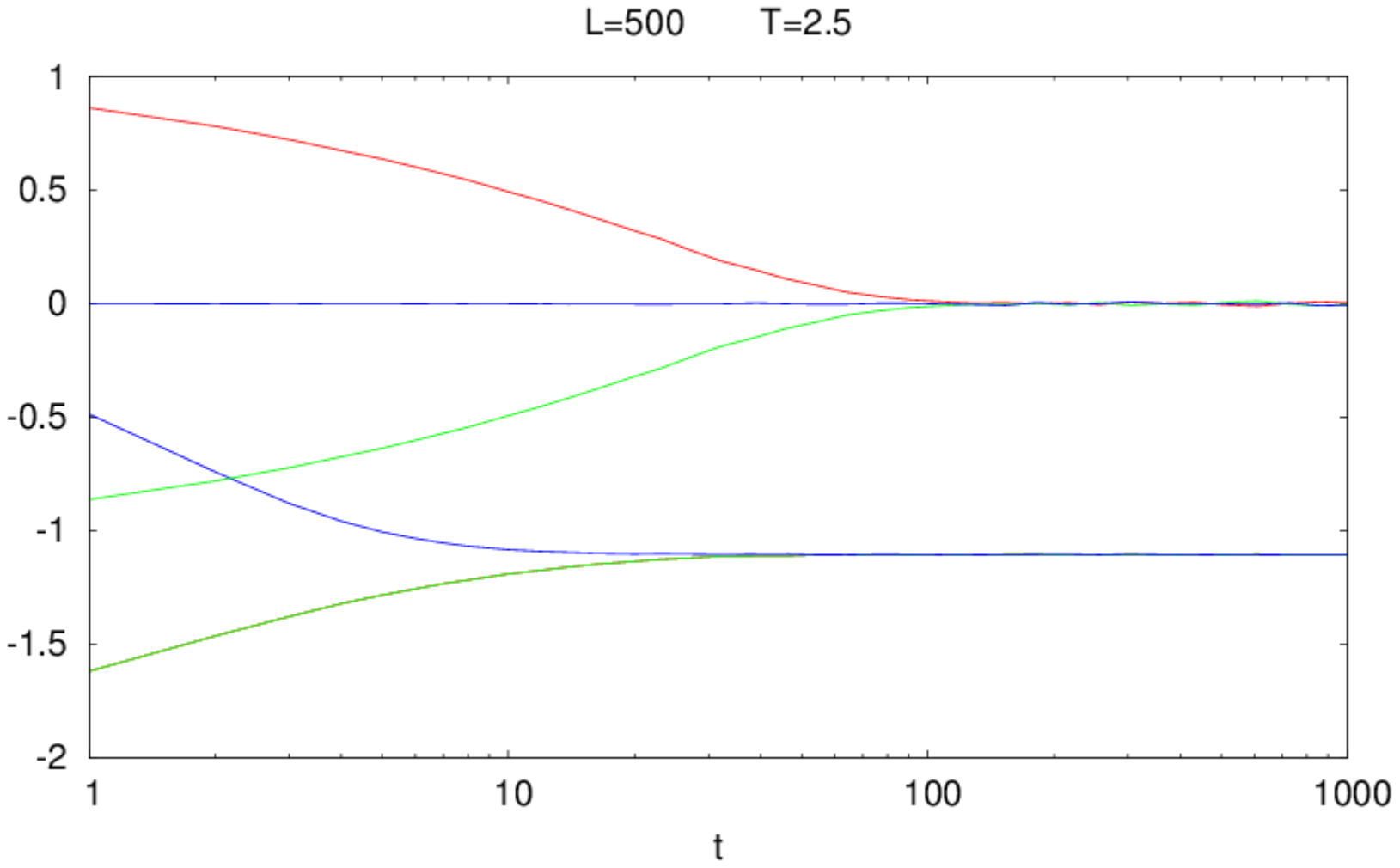
E se for medir ene e mag todo passo...

Como rodar uma simulação de MC

- Dois tipos:
  - De equilíbrio
  - Fora do equilíbrio
- Imprecisões:
  - Sempre necessitamos fazer médias sobre histórias térmicas = diferentes sequencias #'s aleatórios
  - Precisamos saber se sistema equilibrou
  - Conhecer correlações de equilíbrio (amostras independentes)
  - Identificar transições de fase

Convergência para equilíbrio

- Condições iniciais diferentes



Flutuações e respostas

$$E = -J \sum_{\langle i,j \rangle} s_i s_j - h \sum_i s_i$$

$$Z = \sum_{\vec{s}} e^{-\beta E(\vec{s})}$$

$$\beta \equiv 1/T$$

$$\langle A \rangle = \sum_{\vec{s}} A(\vec{s}) e^{-\beta E(\vec{s})}$$

$$m(\vec{s}) = \frac{M(\vec{s})}{N} = \frac{1}{N} \sum_{i=1}^{N} s_i$$

$$e(\vec{s}) = \frac{E(\vec{s})}{N} = -\frac{1}{N} \sum_{\langle i,j \rangle} s_i s_j$$

$$\chi \equiv \frac{\partial \langle M \rangle}{\partial h} = \beta N (\langle m^2 \rangle - \langle m \rangle^2)$$

$$C \equiv \frac{\partial \langle E \rangle}{\partial T}$$

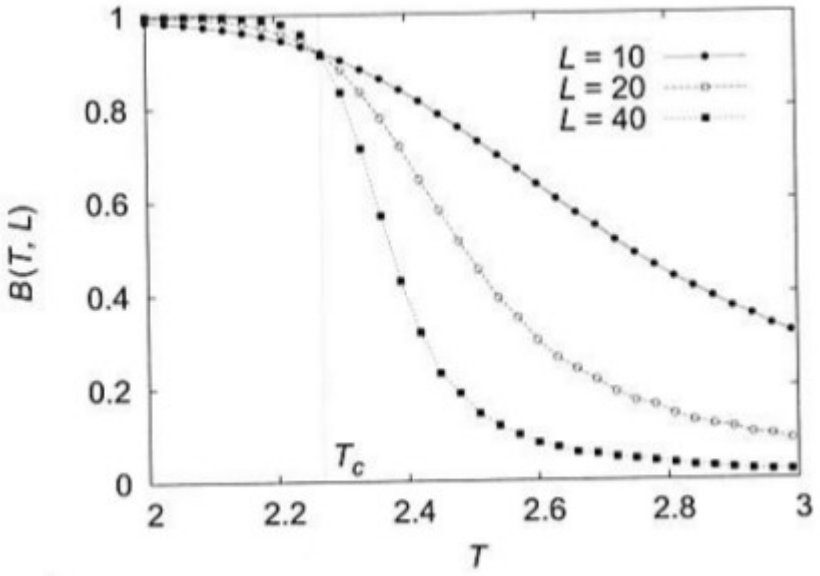
$$c = \beta^2 N (\langle e^2 \rangle - \langle e \rangle^2)$$

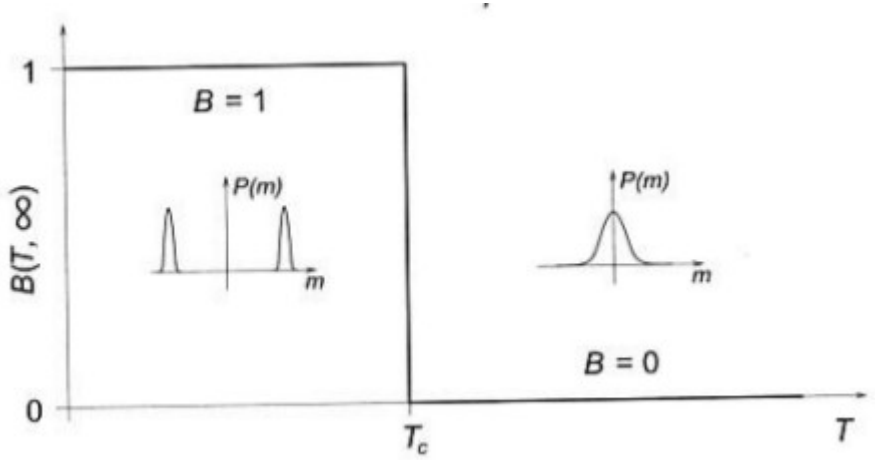
Médias sobre n histórias...
Erros: quantidades simples (m): $\sigma = \sqrt{\frac{\langle m^2 \rangle - \langle m \rangle^2}{n-1}}$

Outras F(m) viés → resampling methods (bootstrap, jacknife)...

Cumulante de Binder

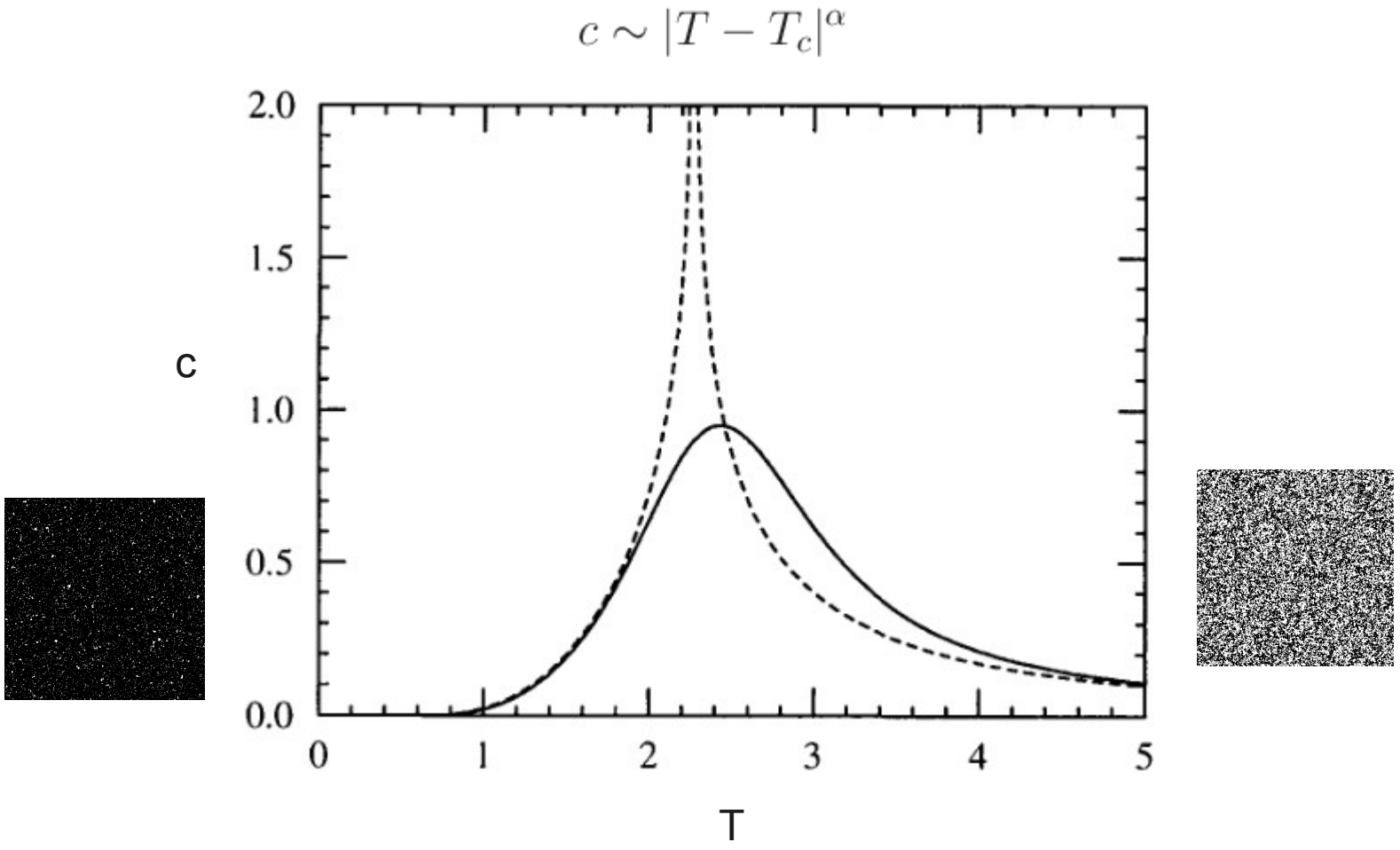
$$B(T,L) \equiv \frac{1}{2}\left(3 - \frac{\langle m^4 \rangle}{\langle m^2 \rangle^2}\right)$$





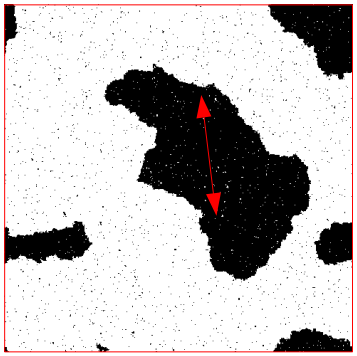
$$B(T_c, L) = cte$$

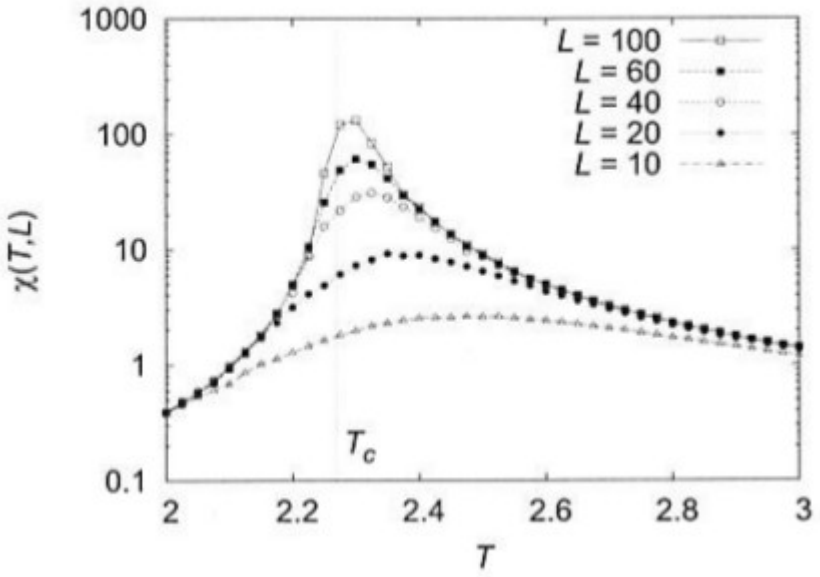
Transição de fase



Tamanho finito → ausência de transição de fase
analiticidade da função de partição... Tc(L)

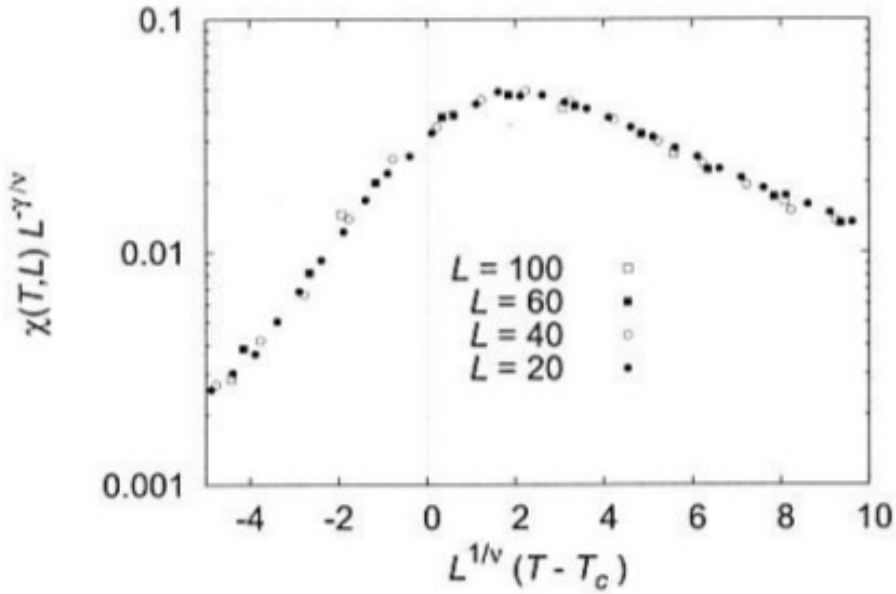
Transição de fase





$$\xi \sim |T - T_c|^{\nu}$$

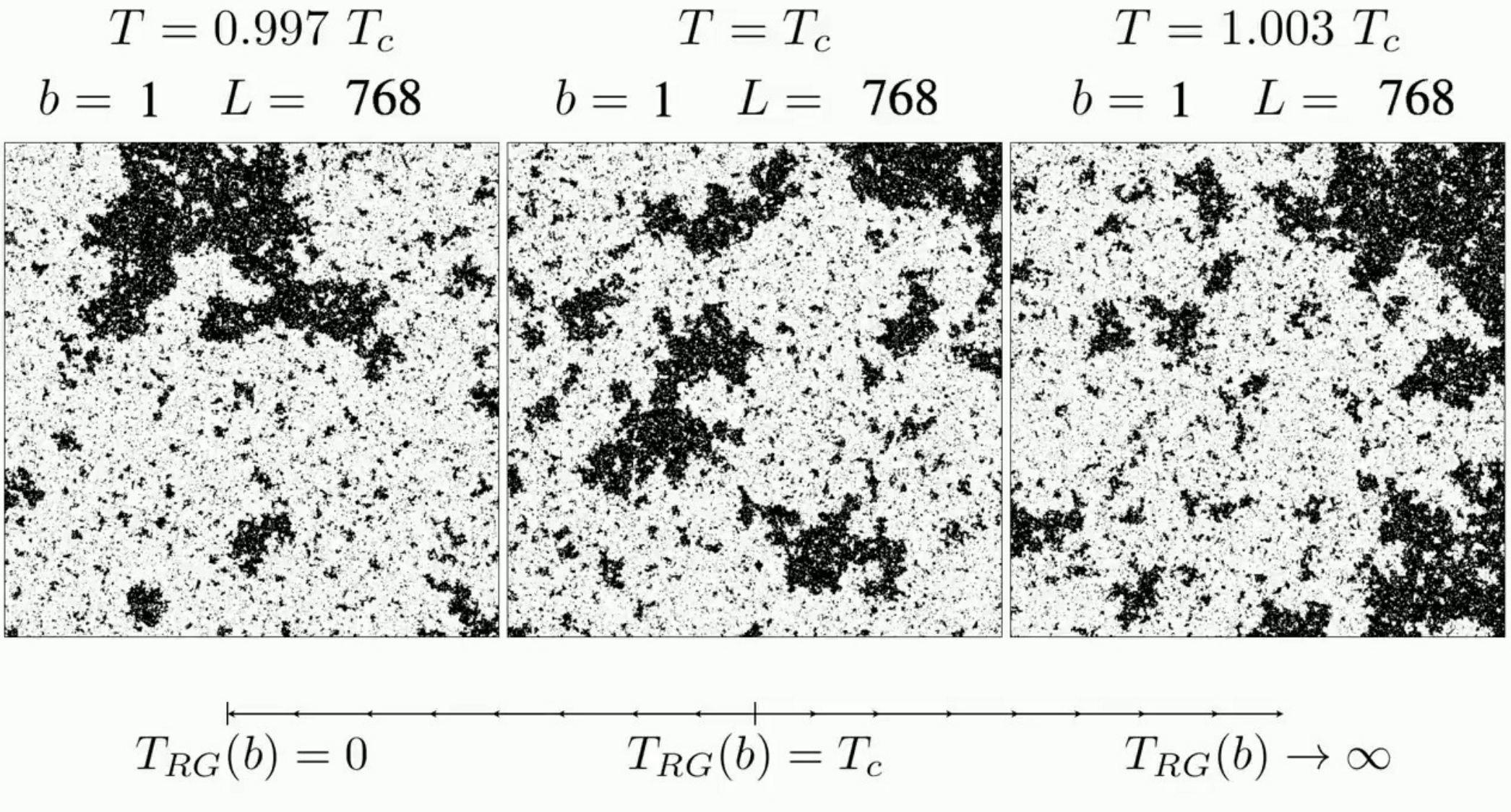
$$\chi \sim |T - T_c|^{\gamma}$$



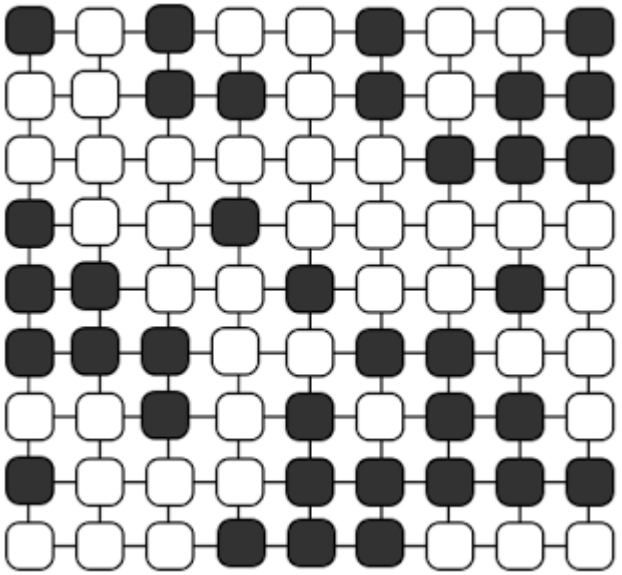
Finite size scaling
Classes de universalidade

$$C(r) \equiv \langle s_i s_{i+r} \rangle \sim \frac{1}{r^{2-d+\eta}}$$

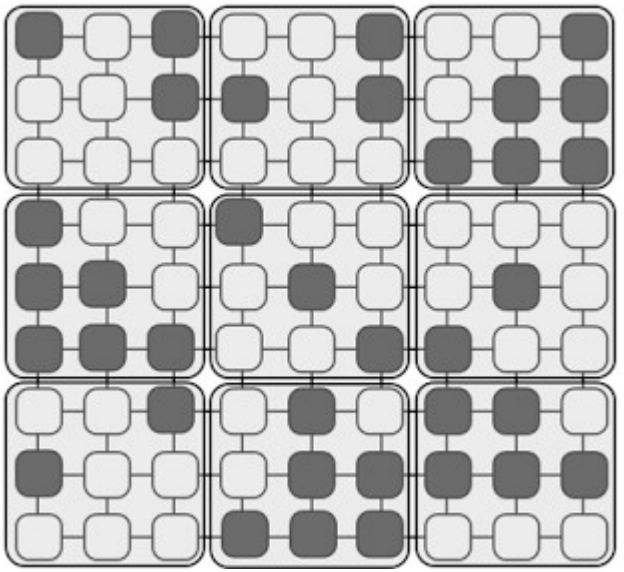
Domínios na vizinhança de Tc

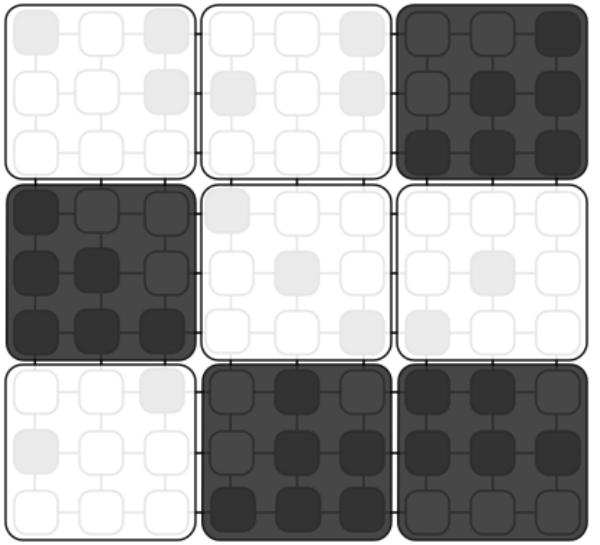


Zoom out



Decisão=maioria





(=renormalização no espaço real)

Opalescência crítica

Opalescência crítica

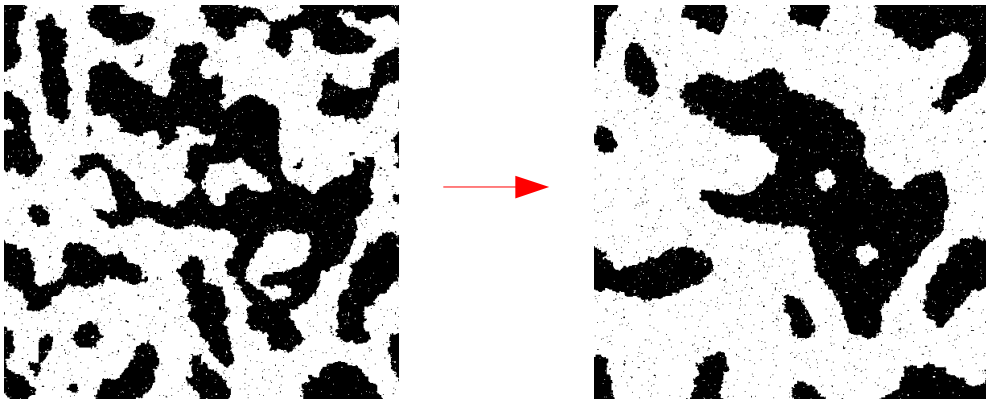
Difícil equilibrar em Tc !

- Porque o sistema está altamente correlacionado e precisa equilibrar em todas escalas espaciais

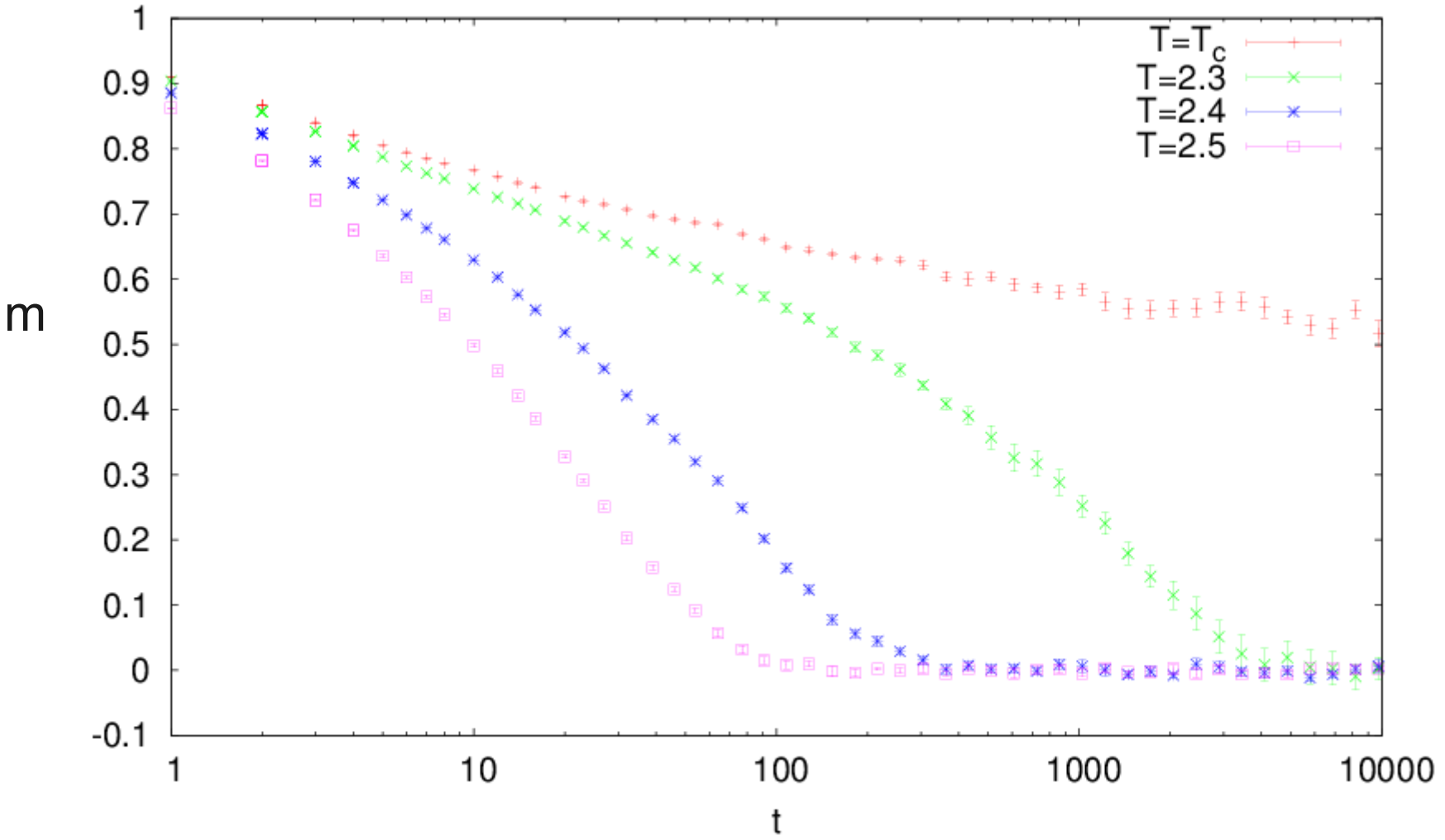
$$\xi(t) \sim t^{1/z}$$

$$\tau \sim \xi^z$$

$$\tau(T_c) \sim L^z$$



Critical slowing down

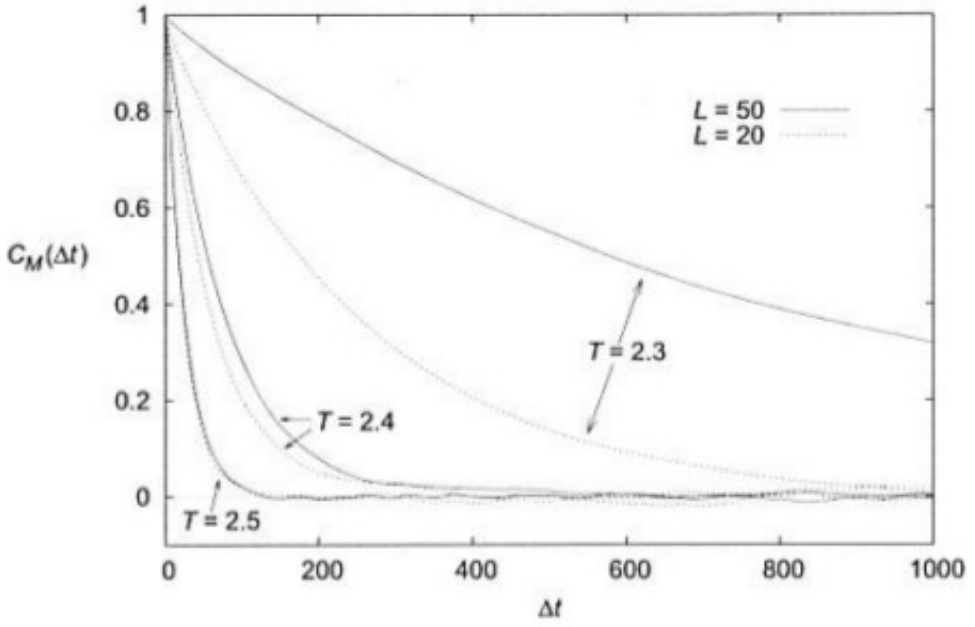


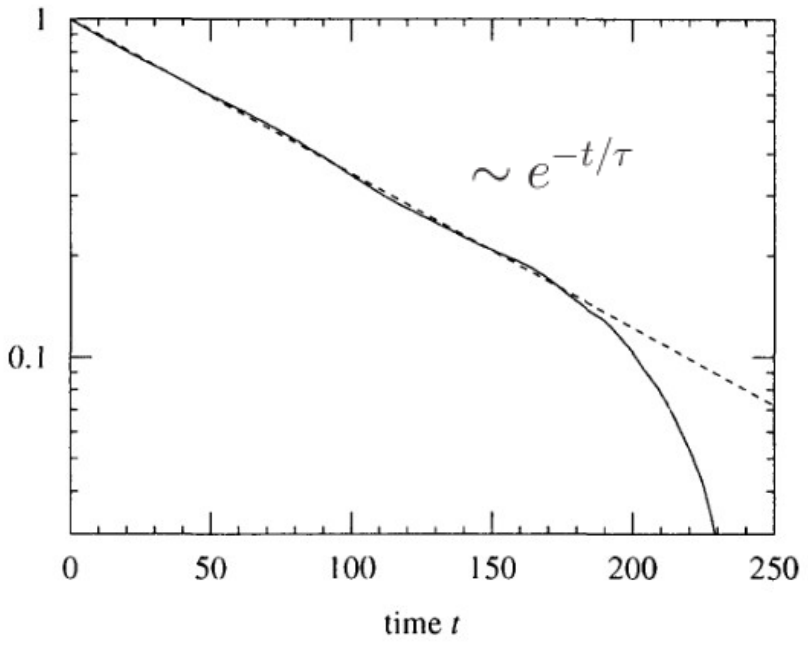
Tempo de descorrelação (eq)

$$C(t,0) = \left\langle \frac{1}{N} \sum_i s_i(t) s_i(0) \right\rangle$$

$$C_M(t) = \langle m(0) m(t) \rangle$$

Medir de τ em τ para amostras independentes





Boas práticas

- Equilíbrio
- Amostras independentes
- Atenção tratamento de erros estatísticos
- Quantidades possuem distribuição ~ $1/N^{½}$, mas custo computacional O(N) - tempo de equilíbrio

Alternativas

- Técnicas computacionais
  - Multispin coding
  - SSE
  - Paralelização (CPU's e GPU's)
- Técnicas de Monte Carlo
  - Swendsen-Wang / Wolf e MUCA / ES …
  - Parallel tempering

Parallel tempering

$$w(T_{baixa} \to T_{alta}) = e^{-(\beta_{baixa} - \beta_{alta})\Delta E} \qquad \Delta E > 0$$
$$w(T_{baixa} \to T_{alta}) = 1 \qquad \Delta E < 0$$