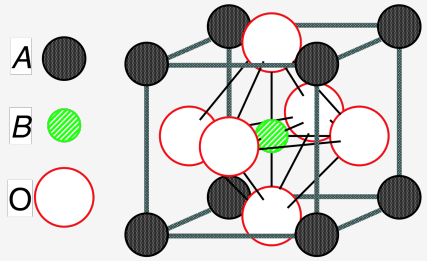
【配信講義】CMSI計算科学技術特論C (2015) 第10回
2015年12月03日

feramと強誘電体②

東北大学金属材料研究所 西松毅
t-nissie@imr.tohoku.ac.jp

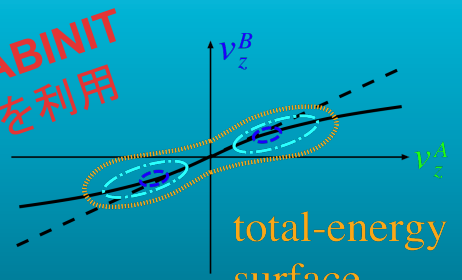
feramを1枚のスライドで紹介(再掲):
誘電体のマルチスケールシミュレーション
http://loto.sourceforge.net/feram/

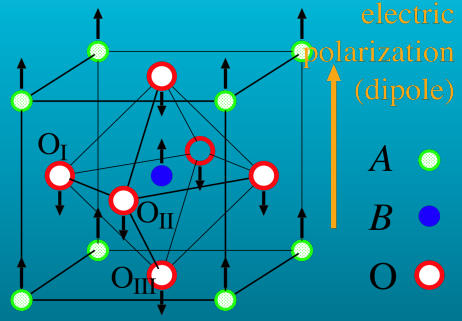


ペロブスカイト型
酸化合物ABO_3

第一原理計算による有効ハミルトニアンH_{eff}の構築

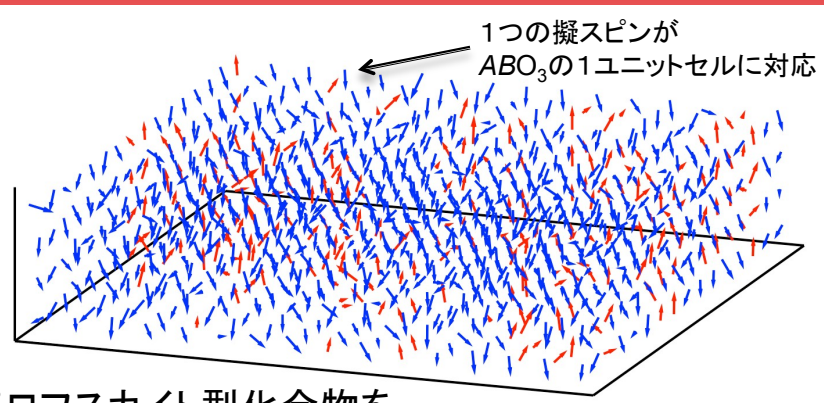
ABINIT を利用





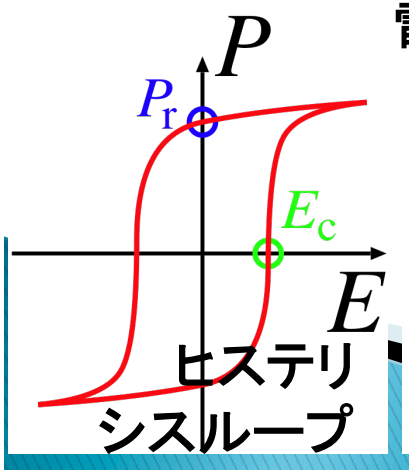
[T. Hashimoto, T. Nishimatsu *et al.*: Jpn. J. Appl. Phys. **43**, 6785 (2004)]

feramによる分子動力学計算



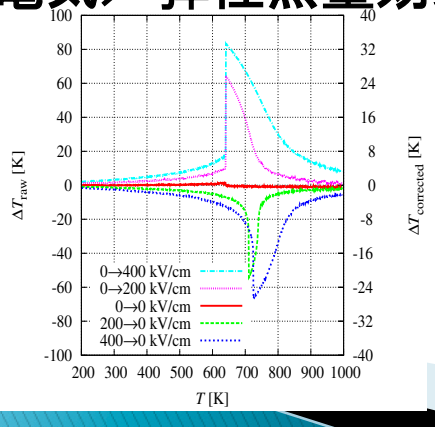
ペロフスカイト型化合物を
双極子相互作用のある擬スピン系としてH_{eff}を
時間発展させ，その有限温度の性質をシミュレート

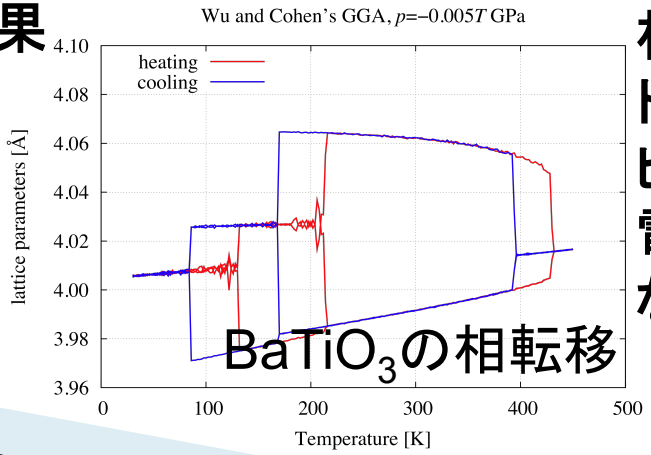
大規模(> 100 nm), 長時間 (> 100 ns) の計算が可能



ヒステリシスループ

電気／弾性熱量効果



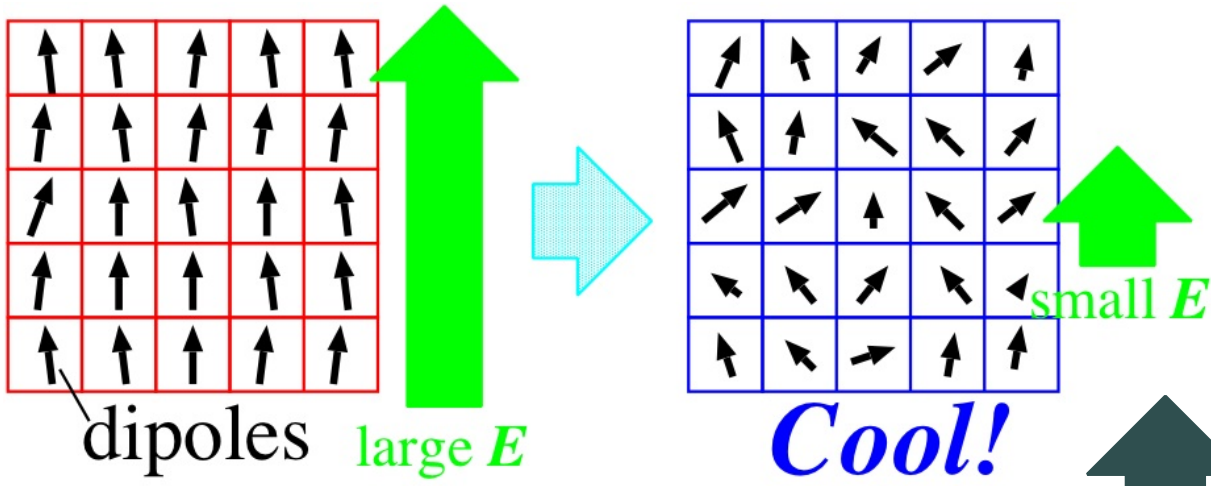


$BaTiO_3$の相転移

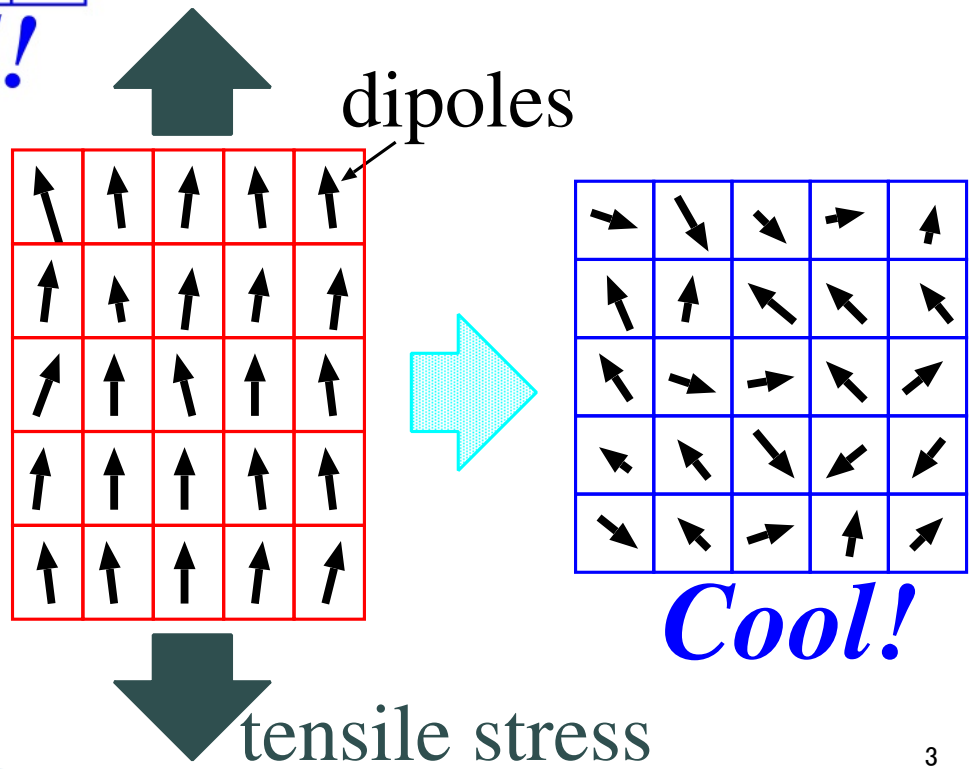
**相転移や
ドメイン構造，
ヒステリシスループ，
電気／弾性熱量効果
などのシミュレーション**

PRL **99**, 077601 (2007),
PRB **78**, 104104 (2008),
PRB **82**, 134106 (2010),
JPSJ **84**, 024716 (2015), …

電気熱量効果 vs 弾性熱量効果



固体冷却素子としての
応用が期待されている
feramでシミュレートできないかと
•米国アイオワ州立大学 (DoE?)
•ドイツ http://www.ferroiccooling.de
から問い合わせ（4～5年前）

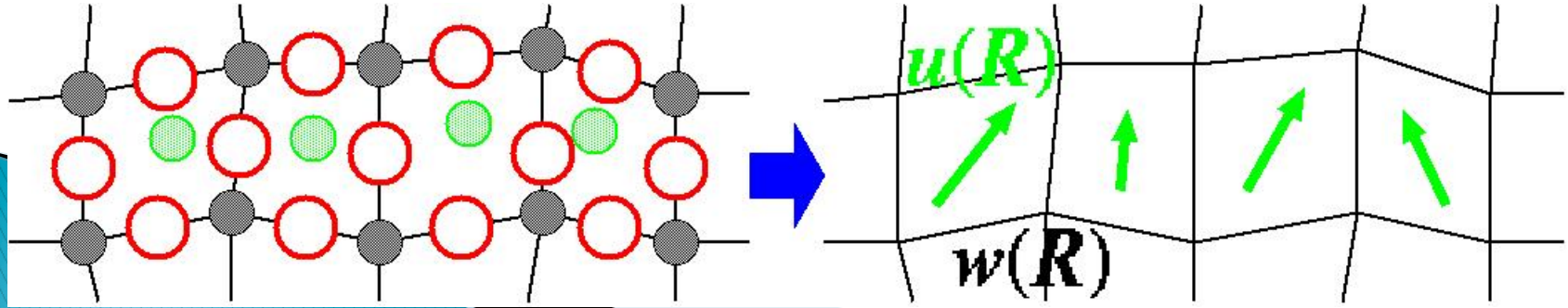


強誘電体の第一原理有効ハミルトニアンにもとづいた分子動力学シミュレーションの手順

- 強誘電体$BaTiO_3$や$PbTiO_3$を**第一原理計算**で調べて有効ハミルトニアン（25個のパラメータを持つ）を構築
  - **ABINIT**を改造して利用 http://www.abinit.org/
    - 平面波展開: E_{cut}=60 Hartree, on 8x8x8 *k*-points
    - Rappeのノルム保存擬ポテンシャル http://opium.sf.net/
    - GGA (Wu and Cohen), LDAやGGA (PBE) ではダメ
  - 絶対0度の物性しかわからない
- その有効ハミルトニアンを**分子動力学法**を使って
  いろいろな条件下で時間発展して物性を予測
  - 独自開発した**feram**を利用 http://loto.sf.net/feram/
  - 大規模（32x32x512ユニット・セル、～100nm）な系の長時間（～1000ns）のシミュレーションが可能
  - 温度、圧力、ひずみ、バルクか薄膜か、外部電場

強誘電体の分子動力学計算のための粗視化: 系の自由度 (DoF) を簡単化

- 実際のペロブスカイト型ABO_3: $15N+6$ 自由度
  - 単位胞中5個の原子
  - 各原子はx, y, zの3方向に動く
  - スーパーセル中N個の単位胞
  - 歪みの6成分
- 粗視化したモデル: $6N+6$ 自由度
  - 単位胞に1つの双極子ベクトル$Z^* u(R)$
  - 単位胞に1つの「音響変位」ベクトル$w(R)$
  - 計 6 自由度/単位胞



有効ハミルトニアン（25個のパラメータは第一原理計算により決める）

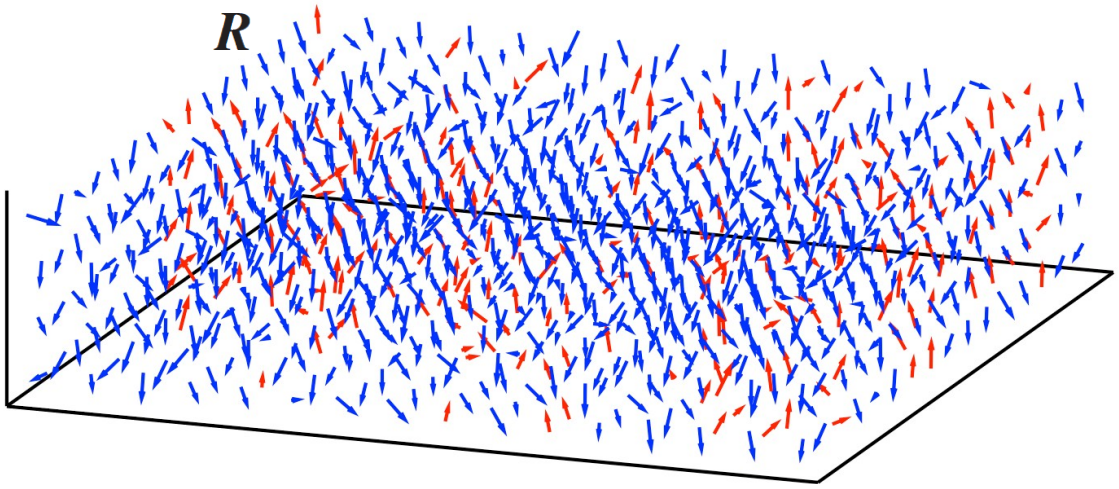
$$H^{\mathrm{eff}} = \frac{M^*_{\mathrm{dipole}}}{2}\sum_{\boldsymbol{R},\alpha}\dot{u}^2_\alpha(\boldsymbol{R}) + \frac{M^*_{\mathrm{acoustic}}}{2}\sum_{\boldsymbol{R},\alpha}\dot{w}^2_\alpha(\boldsymbol{R}) + V^{\mathrm{self}}(\{\boldsymbol{u}\})$$

$$+ V^{\mathrm{dpl}}(\{\boldsymbol{u}\}) + V^{\mathrm{short}}(\{\boldsymbol{u}\}) + V^{\mathrm{elas,homo}}(\eta_1, \cdots, \eta_6)$$

$$+ V^{\mathrm{elas,inho}}(\{\boldsymbol{w}\}) + V^{\mathrm{coup,homo}}(\{\boldsymbol{u}\}, \eta_1, \cdots, \eta_6)$$

$$+ V^{\mathrm{coup,inho}}(\{\boldsymbol{u}\},\{\boldsymbol{w}\}) - Z^*\sum_{\boldsymbol{R}} \mathcal{E}\cdot\boldsymbol{u}(\boldsymbol{R}),$$

すなわち、双極子をたくさんならべたスーパーセル（周期的境界条件）で計算



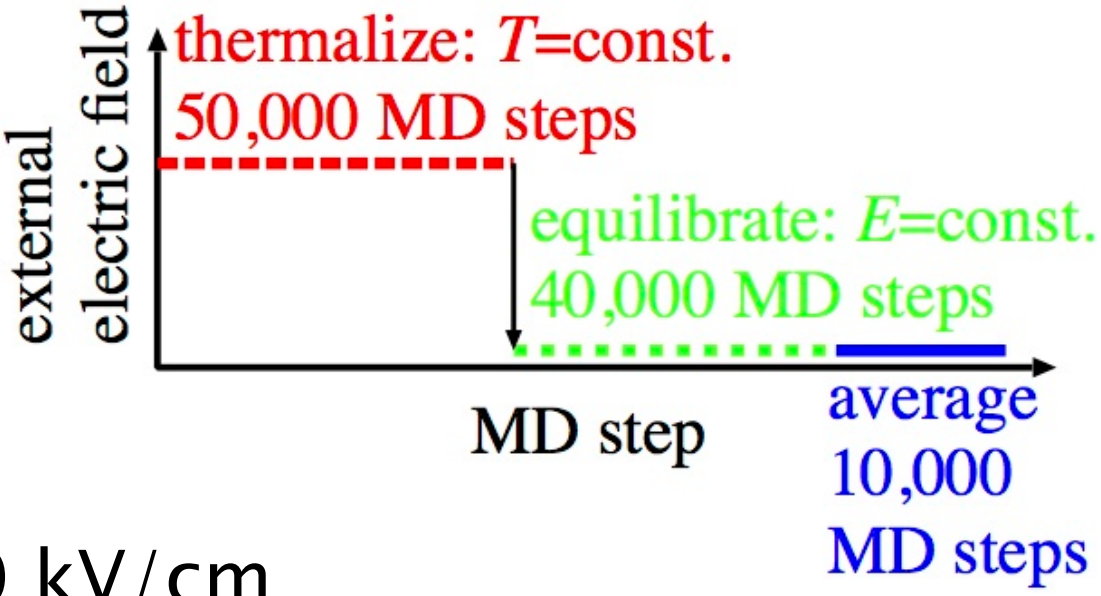
BaTiO$_3$用パラメータ [Takeshi Nishimatsu *et al.*: PRB **82**, 134106 (2010)]

さらなる自由度 (DoF) の減量:「音響変位」ベクトル$w(R)$ は$u(R)$ に対して「最適化」

- イオンの描像: 5x3=15/unit cell
  ↓ 粗視化
- ベクトルの描像: $\boldsymbol{u}(\boldsymbol{R})$ と $\boldsymbol{w}(\boldsymbol{R})$: 2x3=6/unit cell
  ↓
  ↓ $\boldsymbol{w}(\boldsymbol{R})$ を $V^{\mathrm{elas,inho}}(\{\boldsymbol{u}\})+V^{\mathrm{coup,inho}}(\{\boldsymbol{u}\},\{\boldsymbol{v}\})$
  ↓ を最小化するように「最適化」で決定
  ↓
- $\boldsymbol{u}(\boldsymbol{R})$ のみMD、$\boldsymbol{w}(\boldsymbol{R})$はMDしない:
  1x3=3/unit cell

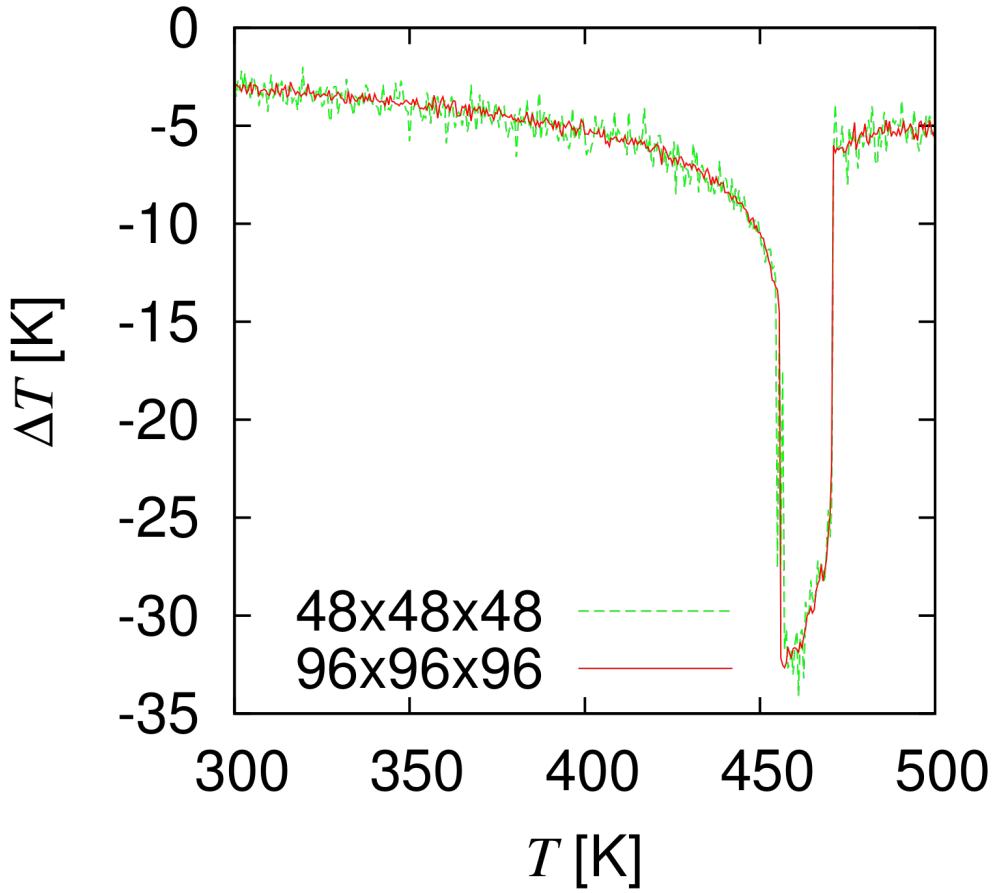
直接的な電気熱量効果の分子動力学シミュレーションの計算条件

- $BaTiO_3$, 96×96×96ユニットセルのスーパーセル
- 有限の外部電場E_z下でカノニカル・アンサンブル計算（定温の熱浴と接触）
- その後E_z=0としてミクロ・カノニカル・アンサンブル計算（系外とは断熱，全エネルギは一定）

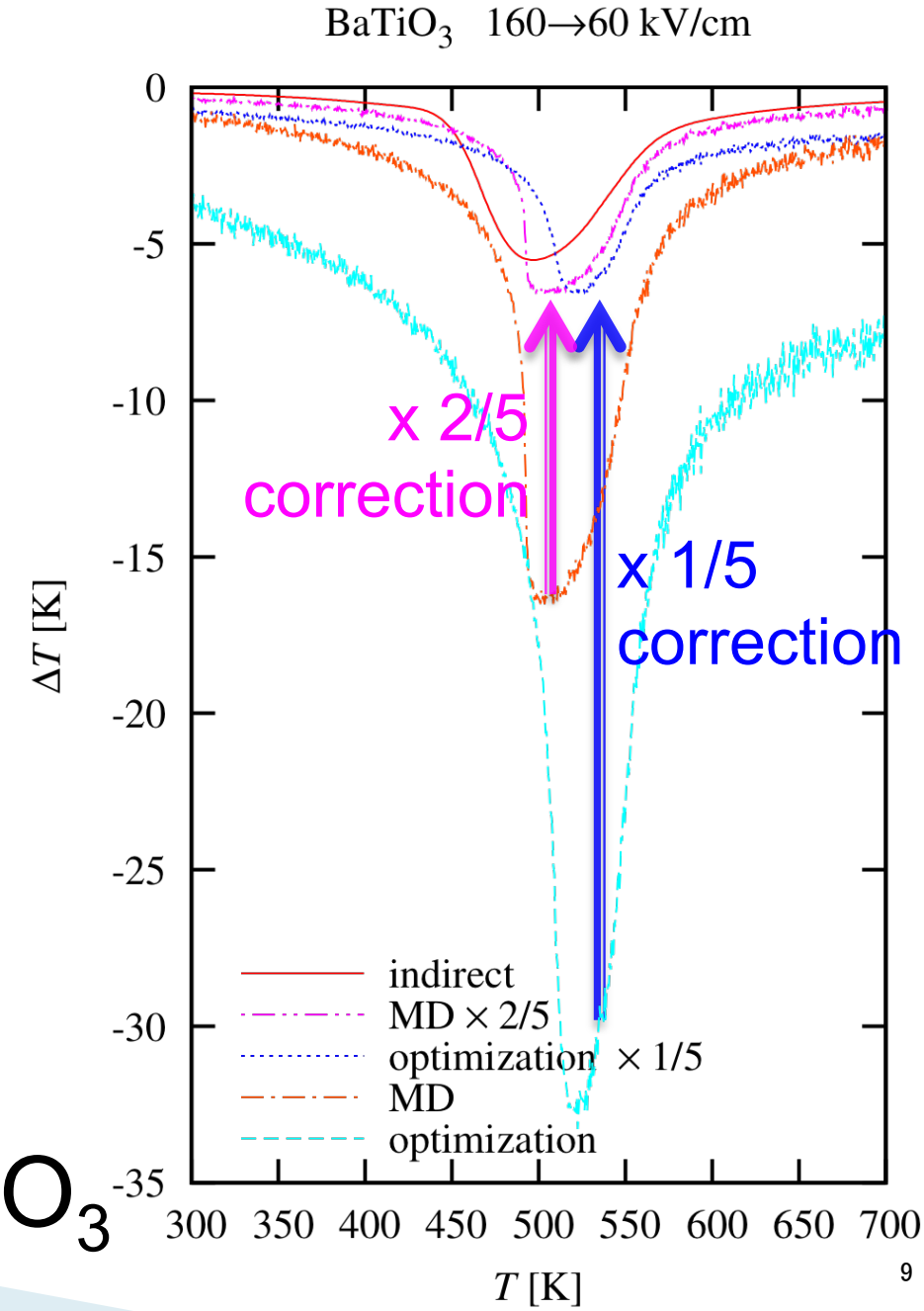


- 外部電場E_z=0～500 kV/cm
- 実験値の比熱 C_vは使わない→比熱を過小評価
  →$|\Delta T|$を過大評価→補正

【直接的】電気熱量効果の計算結果その1

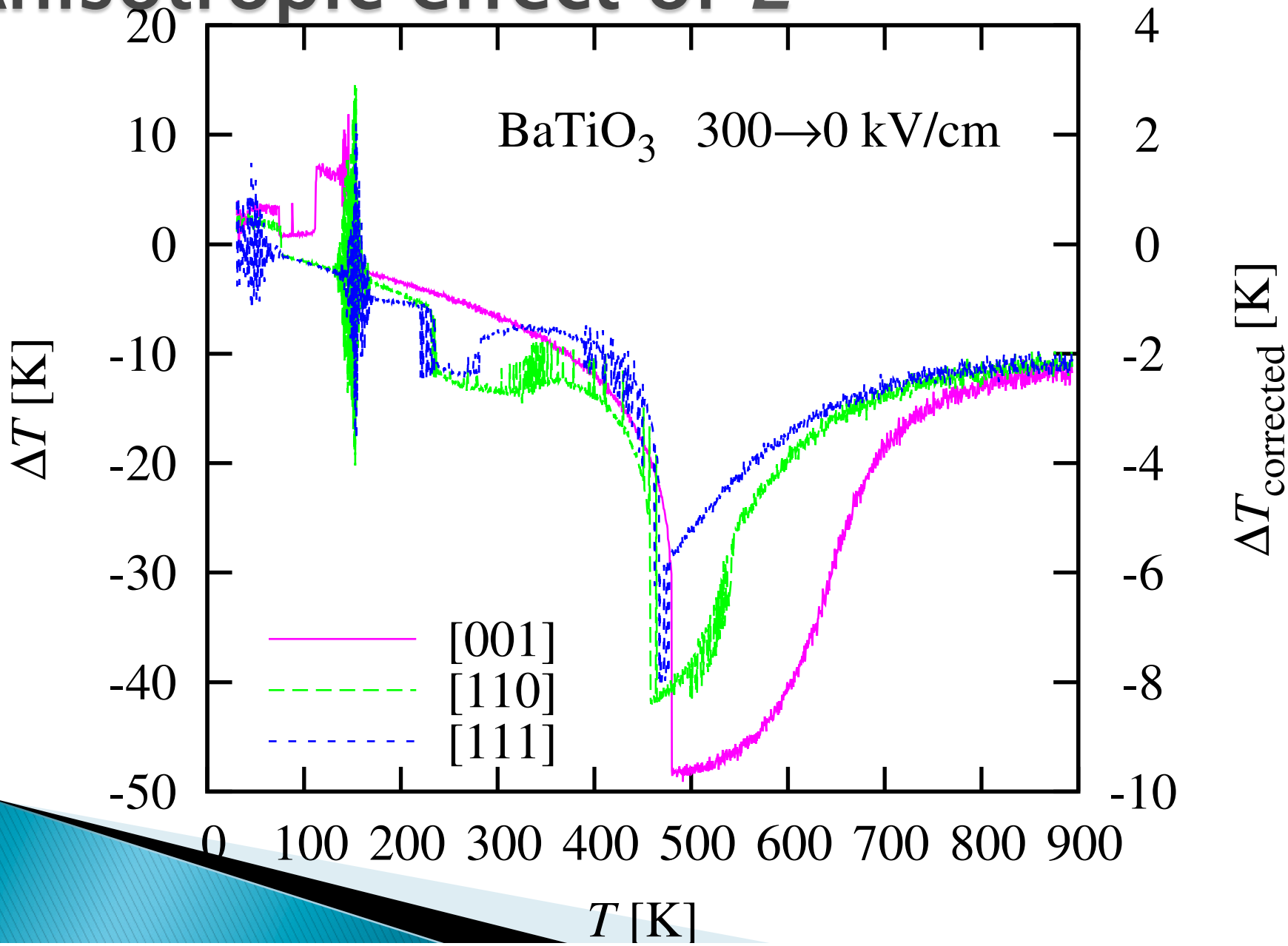


スーパーセルのサイズが
小さいと，ΔTのゆらぎが
大きくなってしまう．



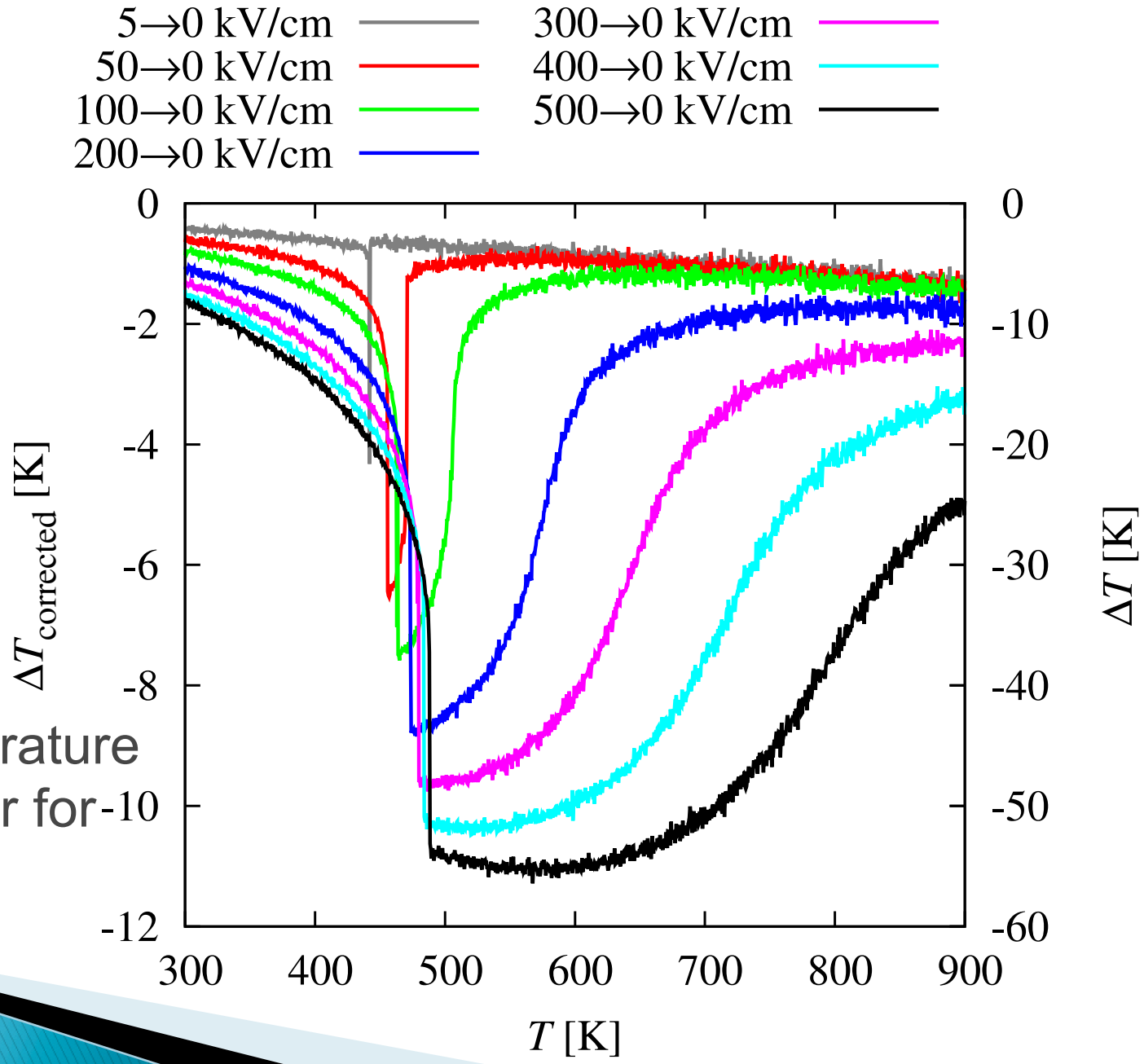
$BaTiO_3$

【直接的】電気熱量効果の計算結果その2
Anisotropic effect of *E*



【直接的】電気熱量効果の計算結果その3

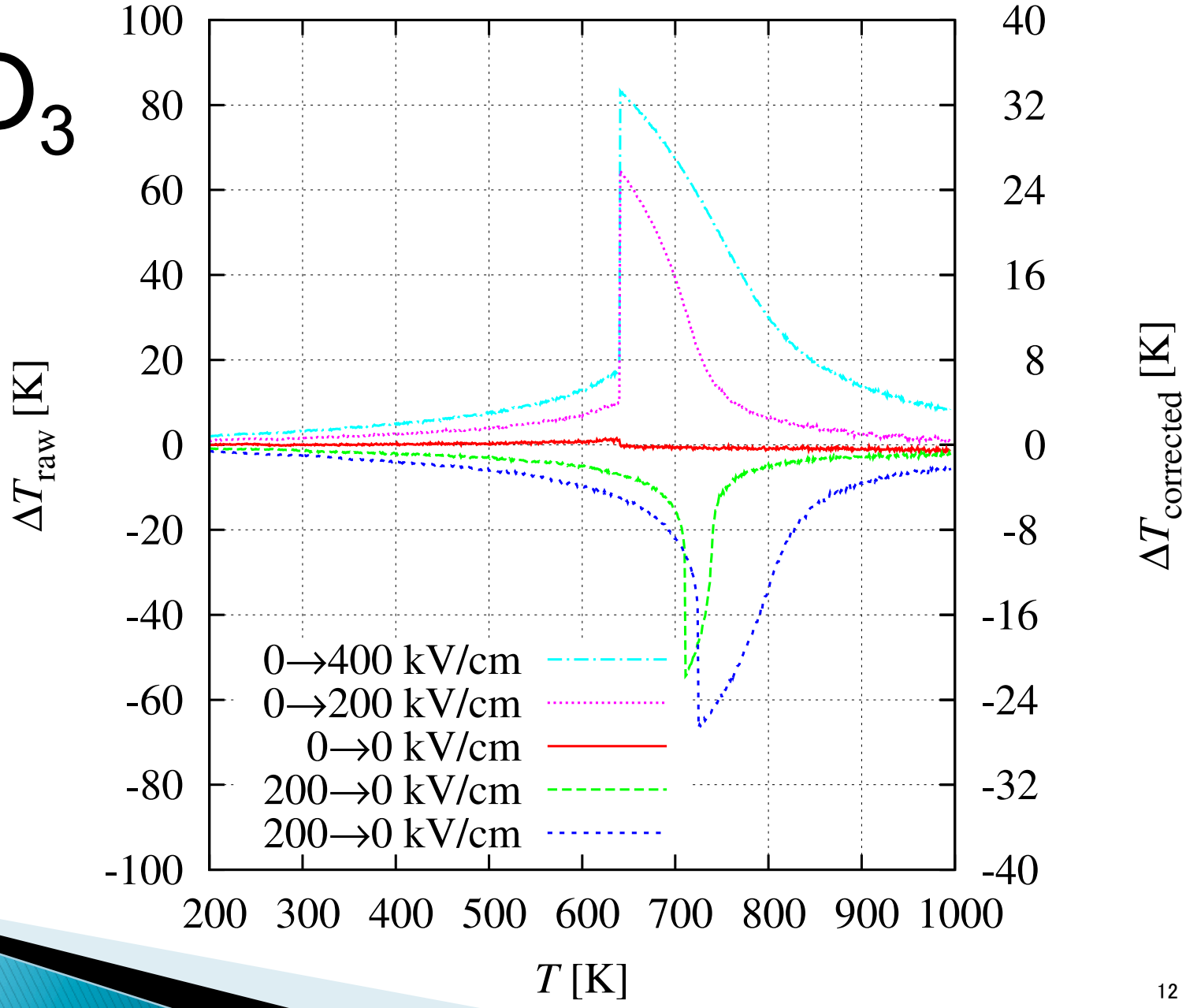
$BaTiO_3$



Effective temperature rang is narrower for smaller *E* field.

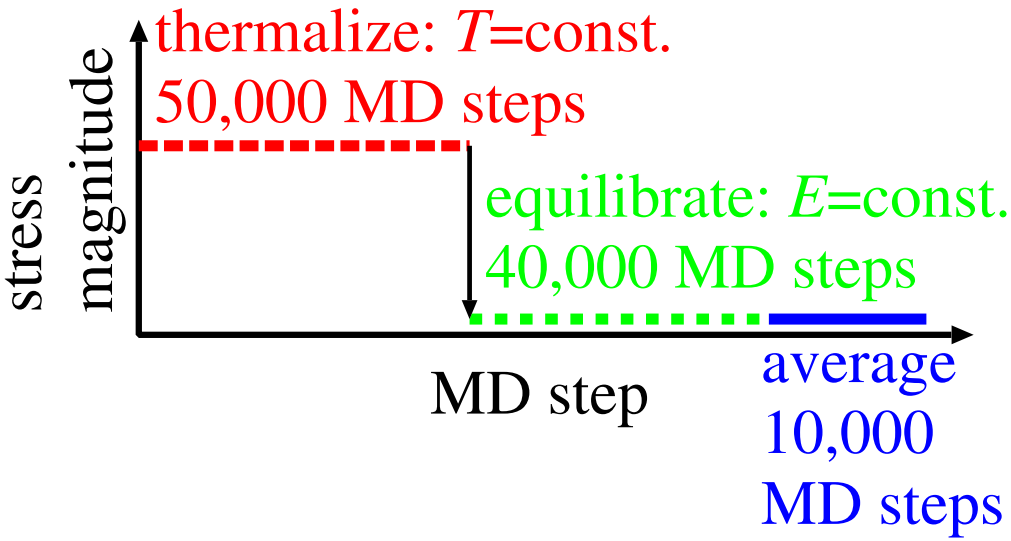
Direct MD simulations of electrocaloric effects

$PbTiO_3$



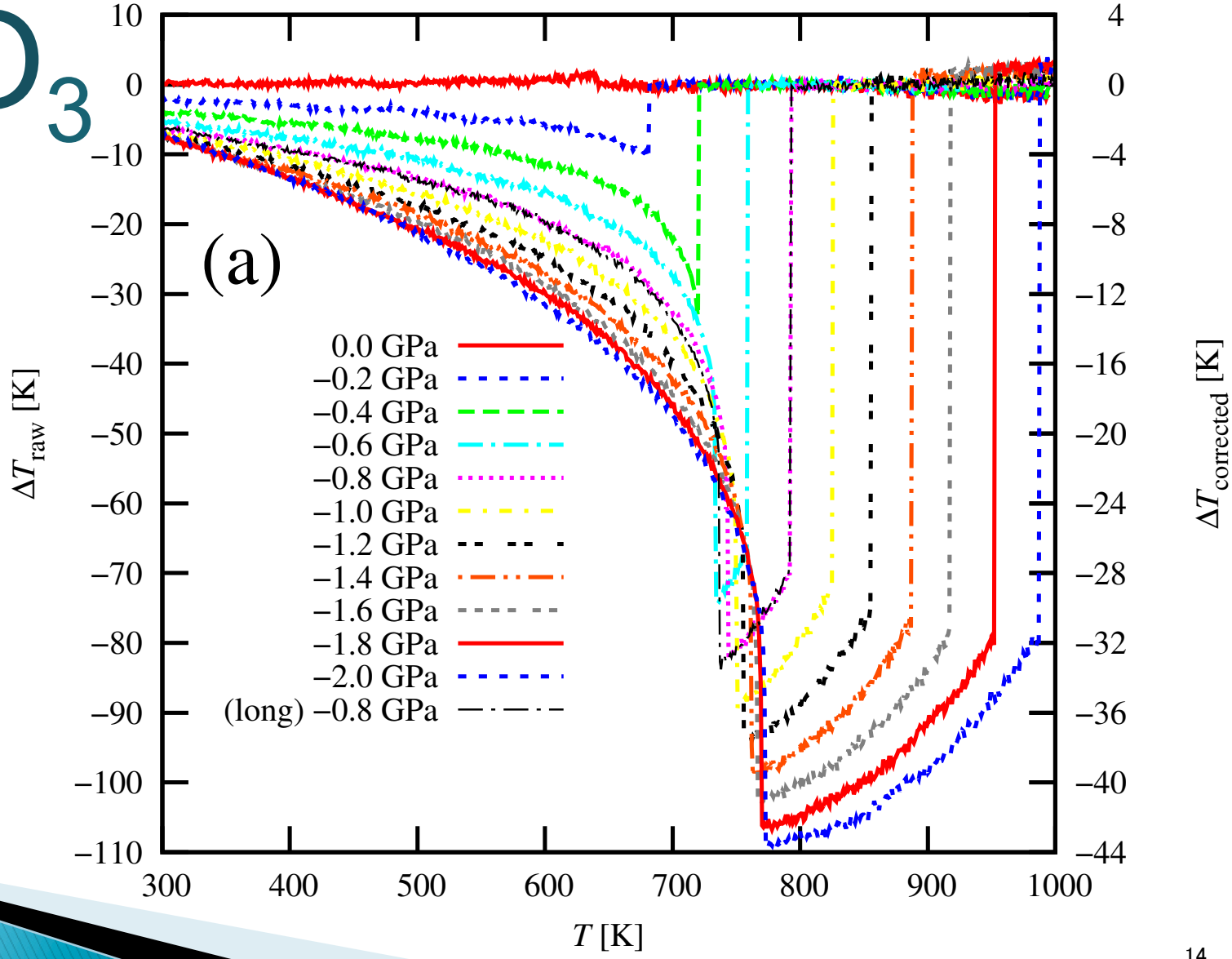
弾性熱量効果の直接的MDシミュレーションの計算条件

- $PbTiO_3$ 64×64×64ユニットセルのスーパーセル
- 一軸応力下で*T*一定のカノニカルアンサンブルでMD
- その後、外部応力を切って、エネルギー一定のミクロカノニカルアンサンブルでMD、温度変化を見積る
- 外部応力:
  0.0～−2.0 GPa
- 粗視化 →
  比熱 C_v の過小評価
  → 温度変化Δ*T*の過大評価 → 補正

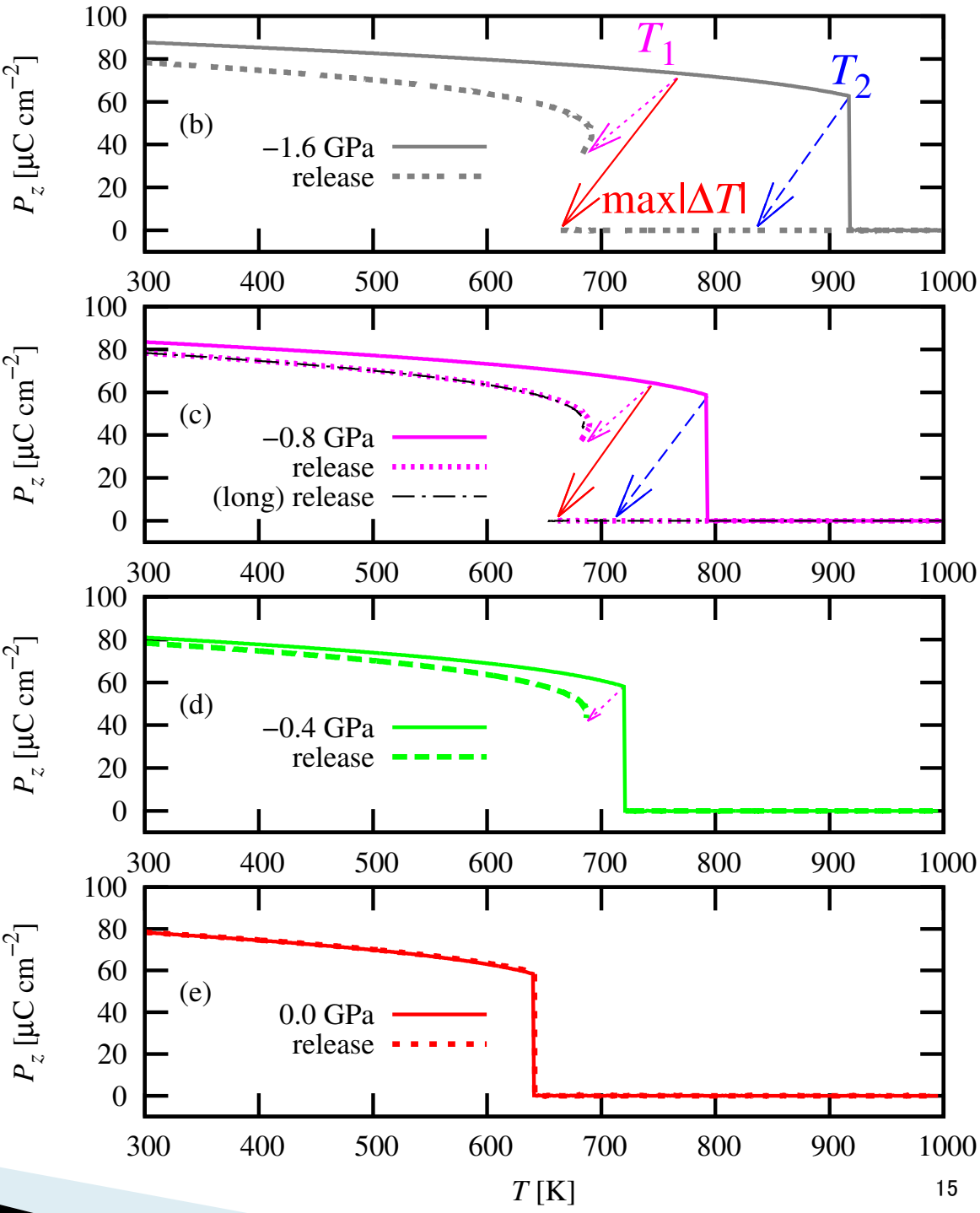


弾性熱量効果による冷却

$PbTiO_3$

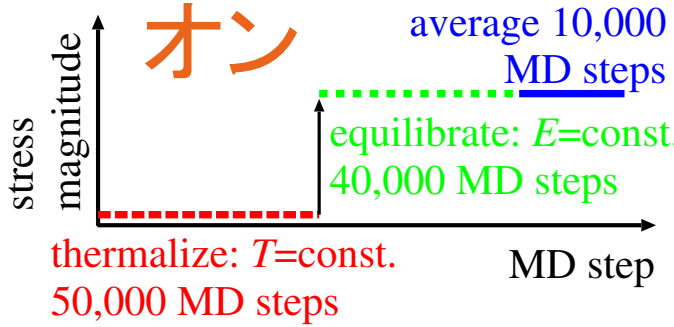


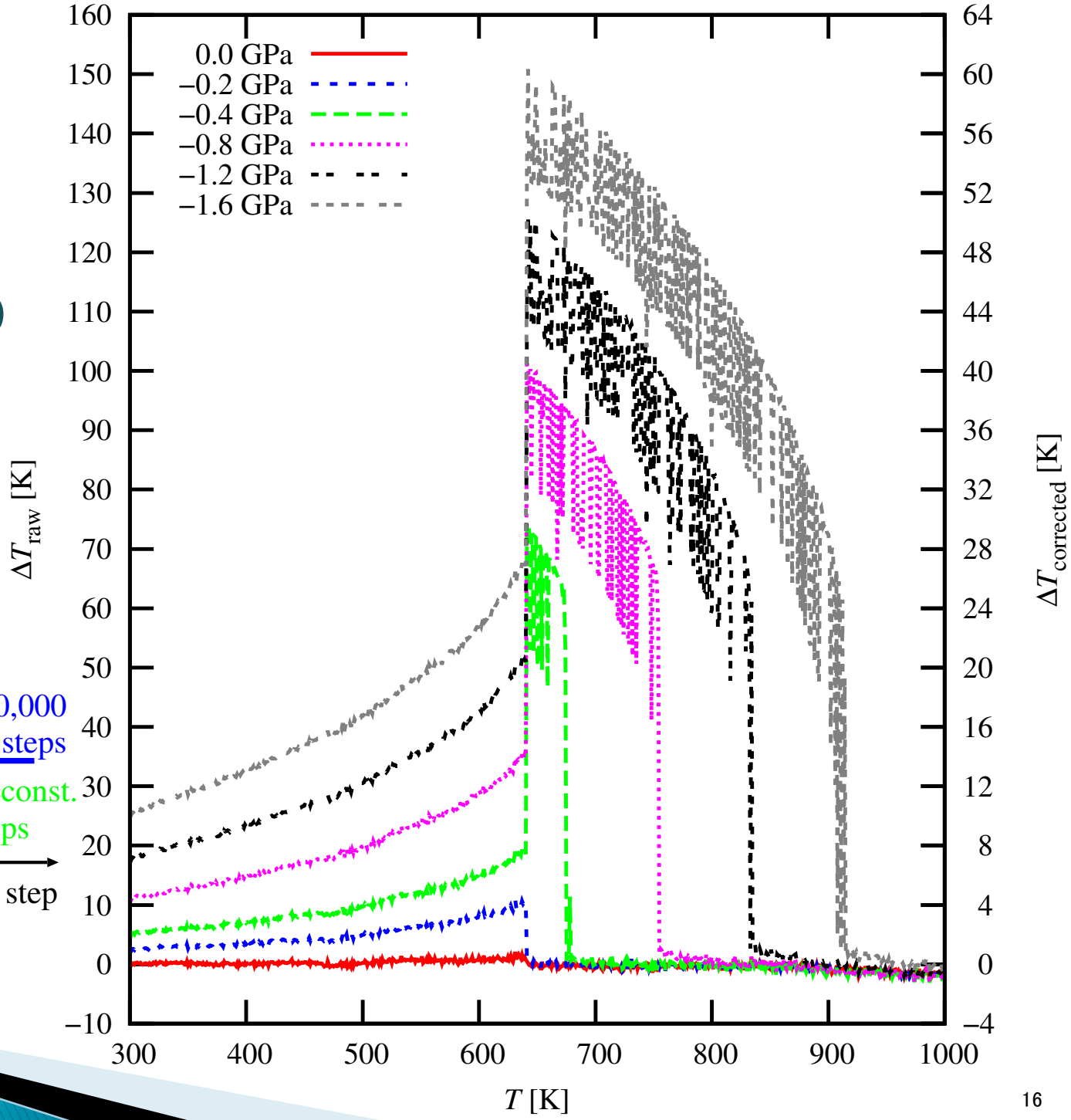
引っぱり応力をスイッチオフするときの$PbTiO_3$の状態の遷移



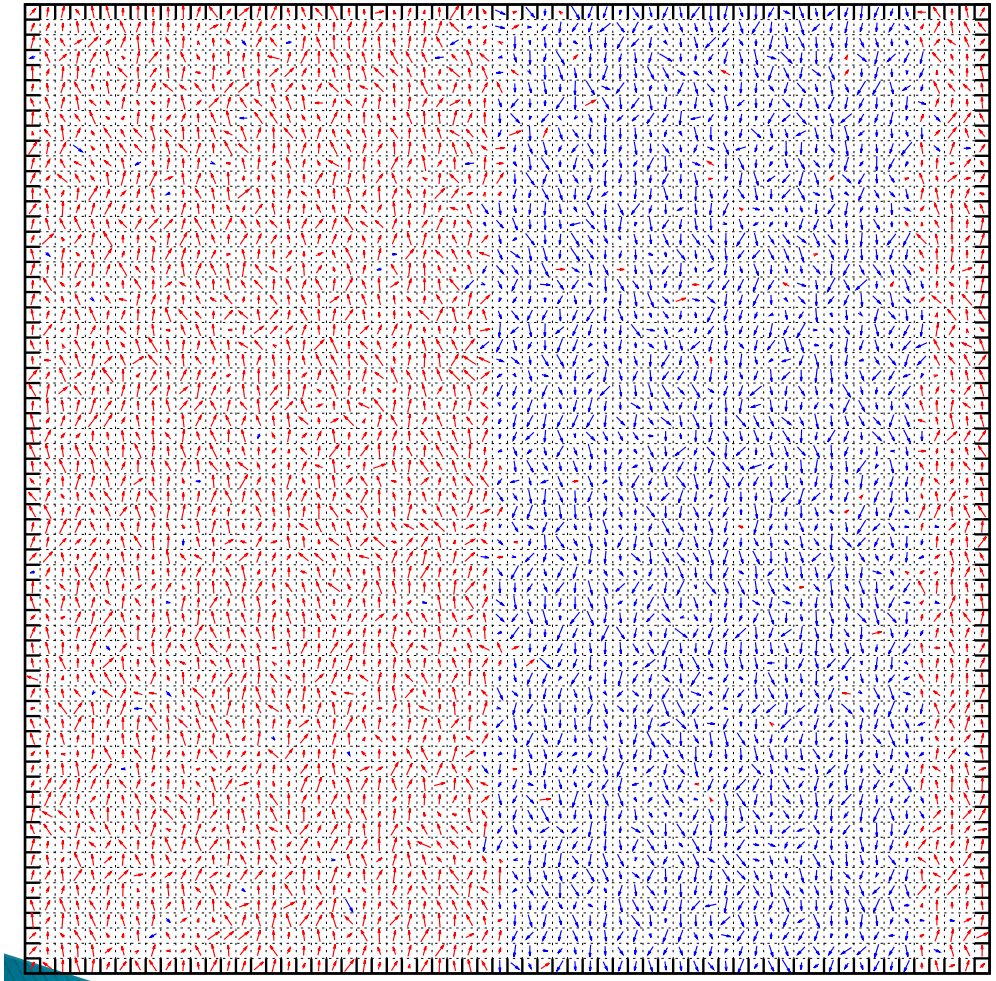
$PbTiO_3$: 弾性熱量効果による加熱

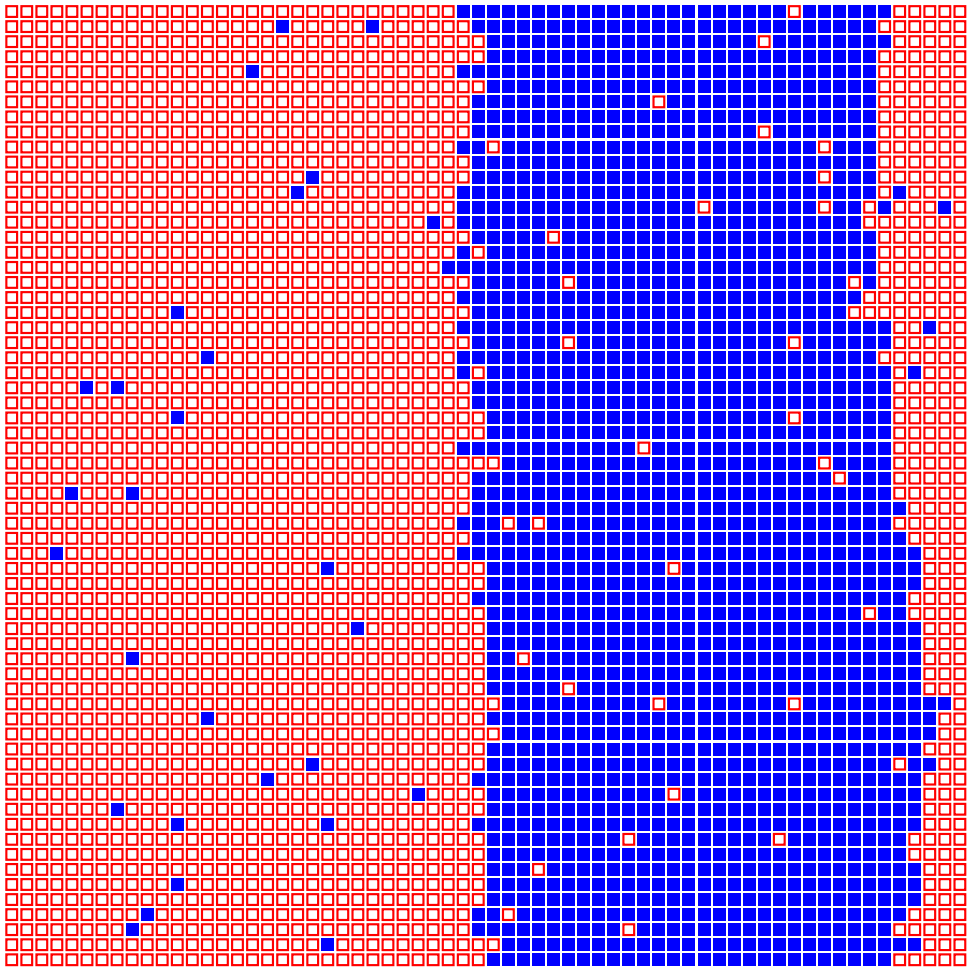
外部応力のスイッチオン





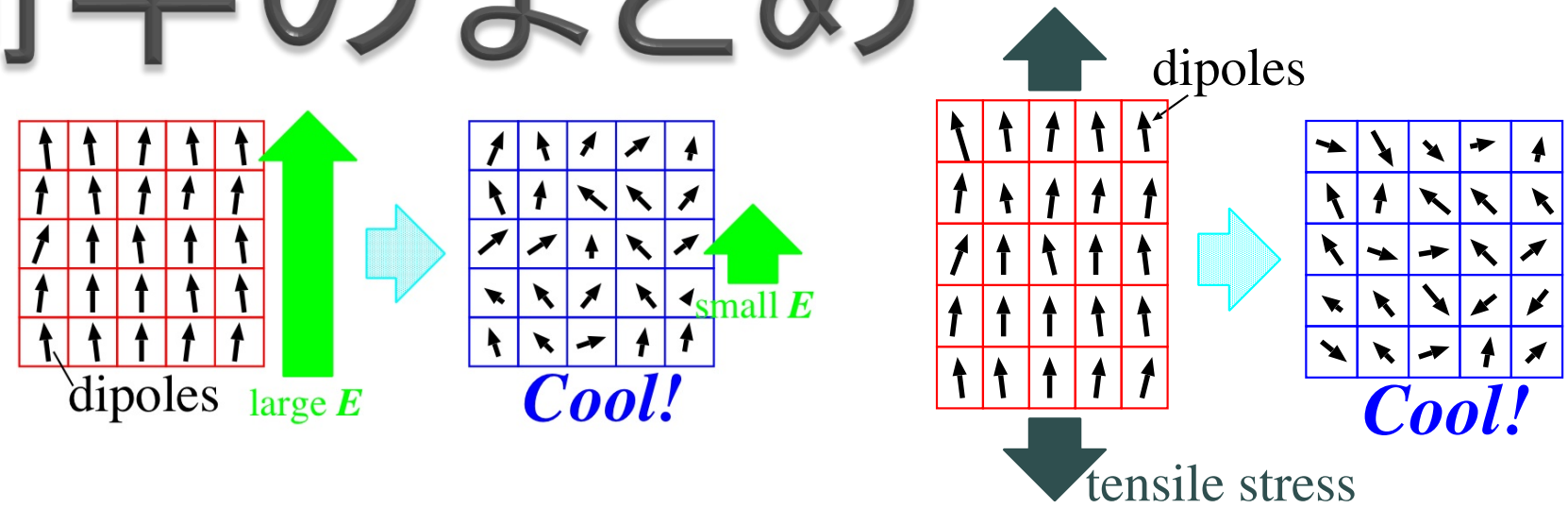
$T>T_C$で常誘電状態から引っぱり応力を突然かけると180°ドメイン構造ができる





cross section and slice

前半のまとめ



- ペロブスカイト型強誘電体ABO_3のための第一原理有効ハミルトニアンに基づく高速な分子動力学計算コードferam

- 電気熱量効果と弾性熱量効果とを
  velocity scaling → leapfrog
  で直接MDシミュレーション

弾性熱量効果のMDの実行方法

```
$ feram elasto770K.1vs.feram elasto770K.2lf.feram
```

- 2つ目の入力ファイル elasto770K.2lf.feram には continue = .true. と書いてあり、1つ前の入力ファイルの終状態からMDが継続する。
- feramの実行方法と入出力ファイルの特徴
  - 独自入力ファイル
  - feram 392K.feram 393K.feram 394K.feram
  - ログは 392K.log 393K.log 394K.log へ
  - その他の出力ファイルも「basename+拡張子」
  - 特に注意を要するエラーや警告は標準エラー出力へ
  - 標準出力は使わない。標準出力をバッファをしないシステムで遅くなるので。

入力ファイルをどうする？

- 実行時引数で指定（poptを使うと簡単に割とうまくいく）
```
./xtalgrowth -c 1 --diameter=0.03
```
- ファイルで順番に指定（論外: 機能を追加すると破綻）
```
393  # Temperature in Kelvin
'md' # Method
```
- ファイルで名前と値を指定（feramは独自パーサで）
```
Kelvin = 393
method = 'md'
```
- FortranのNamelistの利用
- XMLの利用
- DSL（Domain Specific Language）の利用

→次ページからそれぞれの利点欠点を解説

入力ファイルをどうする？
実行時引数で指定

- 実行時引数の順番に意味を持たせるのは論外
  `./rongai 1 0.03`
- 超便利なコマンド・ライン・オプション・パース・ライブラリ popt http://rpm5.org/files/popt/ を使うと簡単に割とうまくいく。2次元結晶成長デモxtalgrowthで採用。

```
$ ./xtalgrowth --help  # 自動生成されるHELP
Usage: xtalgrowth [OPTION...]
  -d, --diameter=0.nnn       Diameter of a ball, d/width
  -h, --height=1.nnn         Height of the system, h/width
  -c, --criterion=n          Criterion, 1=<n=<3
  -v, --velocity=0.000n      Falling velocity par step
  -g, --guest='String'       Name of the guest
  -H, --help                 Print Usage
$ ./xtalgrowth -c 1 --diameter=0.03
   # コマンドラインオプションが簡単に実装できる
```

入力ファイルをどうする？
独自ファイル+独自パーサ

- 利点
  - 可読性が高い
  - 分数 1/3 や 単位(例: Bohr or Angstrom)を
    指定できる(ABINITが採用)
  - コメントを #(ハッシュ)から書くことにしておけば
    1行目にシバン「#!./feram」を書くことで
    「入力ファイル」を**実行属性をもつスクリプト**にできる
- 欠点
  - 独自パーサを書くのが面倒
  - パラメータを振るためのループの実装とか
    凝りだすとキリがない

入力ファイルをどうする？ XMLの利用

- 利点
  - 機械可読性が高い
  - 要素と属性を使い分けることができる
  - 入れ子構造にできる
  - Fortran用のFoXライブラリというのもある
- 欠点
  - 人間可読性が低い
  - スペルミスを検出しにくい
    （カウンタを導入して参照されなかったタグを見つける？）
  - 何を要素にして何を属性にするか迷う

入力ファイルをどうする？ FortranのNamelistの利用

- 利点
  - どんなFortranコンパイラにも実装されている
- 欠点
  - 古い
  - 可読性が低い
  - エラーメッセージが不親切

入力ファイルをどうする？
DSL (Domain Specific Language) の利用

- DSLの定義は曖昧かもしれないが、例えば
  - Ruby版のmake(1)であるrakeを使えば、入力ファイル間の依存関係を記述できるかも、並列処理もできるかも (pwrake)
  - PythonのAtomistic Simulation Environment (ASE) なら多彩な機能がすぐ使える
- 利点
  - スクリプト言語のループ機能をそのまま使ってパラメータを振る
  - 「入力ファイル」に実行属性を与えられる
- 欠点
  - CやFortranとのインターフェイスを書くのが面倒

“Programming is a process of designing your own DSL” --Dave Thomas

GUI (Graphical User Interface) は使えるか？

- 欠点: ①独自パーサ、②マニュアル（英語・日本語）、に加えて③GUI までバージョンアップ（パラメーターの増加や変更）に対応させるのは大変
- この3つを自動生成できるような仕組みが必要か？
  - GUI以外はpoptが参考になるかもしれない

出力ログファイルのくふう

- 入力が 393K.feram なら 393K.log が出力ファイル
- verbose値（饒舌値）による出力量の制御が可能
- ファイル名と行番号が書き出されるので、デバッグ時などに、どこで何をやっているかが一目瞭然。Emacs エディタ上で実行すれば、クリックや next-error でソースコードの該当部分に飛んで行ける。__FILE__, __LINE__を利用。

```
make && OMP_NUM_THREADS=2 ./feram 393K.feram && cat 393K.log
make: Nothing to be done for 'all'.
feram_common.F:45: BEGIN: feram by Takeshi NISHIMATSU 0.25.01unstable
  feram_common.F:47: HOSTNAME: portlandite.imr.tohoku.ac.jp
  feram_common.F:51: DATE_AND_TIME: 2015-10-31T21:34:12.732+0900
  feram_common.F:54: N_THREADS:     2
  feram_common.F:59: FFTW_WISDOM: Successfully imported FFTW wisdom in current dire
  param_module.F:124: BEGIN: read_Param().
    param_module.F:126: INPUT_FILENAME: 393K.feram
    verbose = 2
    #--- Method, Temperature, and mass ---------------
    method = 'md'
    kelvin = 393
    mass_amu = 38.24
    Q_Nose = 0.05
          :
          :
```

Emacsエディタで393K.feramを実行してログファイル393K.logの冒頭部分をハイライト表示している様子

feramの可視化機能

- feram 393K.feram と実行されたとき、スナップショットが 393K.0000040000.coord に記録される
- それをEPSに可視化するツールをパッケージに同梱
  - feram_cross_section_q.sh（断面図。分極を矢印で。）
  - feram_slicer.rb（スライス。分極を□か■で。）, etc...
- アニメーションはそれをImageMagick(1)のconvert(1)でGIFに変換して、gifsicle(1)でパラパラマンガに（ファイルサイズが大きくなるのが欠点）
- gifアニメーションの作成はshellでくふう↓

```
$ for y in `seq 0 1 31`
do feram_cross_section_q.sh 393K.0000060000.coord 4.0 $y y 0.73;\
mv 393K.0000060000-q-y.eps 393K.0000060000-q-y-`printf "%0.2d" $y`.eps; done
$ for e in 393K.0000060000-q-y-*.eps
do convert -flatten -density 400 $e -resize 25% `basename $e .eps`.gif; done
$ gifsicle --colors=256 --delay=40 --loop 393K.0000060000-q-y-??.gif \
        > 393K.0000060000-q-y-animation.gif
$ gifview 393K.0000060000-q-y-animation.gif
```

feramのコンパイルの手順

```
$ tar xf feram-X.YY.ZZ.tar.xz
$ mkdir feram-X.YY.ZZ/build-with-gfortran
$ cd $_
$ ../configure --help
$ ../configure
$ make -j4
$ make check   # テストの実行
$ sudo make install
```

- このように ./configure && make && make install でコンパイル／インストールができるソースパッケージの構築を自動化してくれるのがAutotoolsです
- feramはautoconfとautomakeとを使っています
  今回の後半はこのAutotoolsを紹介します

Autotoolsとは

- テストとパッケージングの自動化をしてくれるフリーソフトウェア
- autoconf+automake
  - m4で書かれているが知識は不要
  - コメント行はdnlで始める
- 基本的に3つのファイルを用意
  - configure.ac
  - Makefile.am
  - src/Makefile.am
- autoreconfだけ☆簡単☆
- さらに詳しい日本語の解説: http://loto.sourceforge.net/feram/Autotools-memo.ja.html

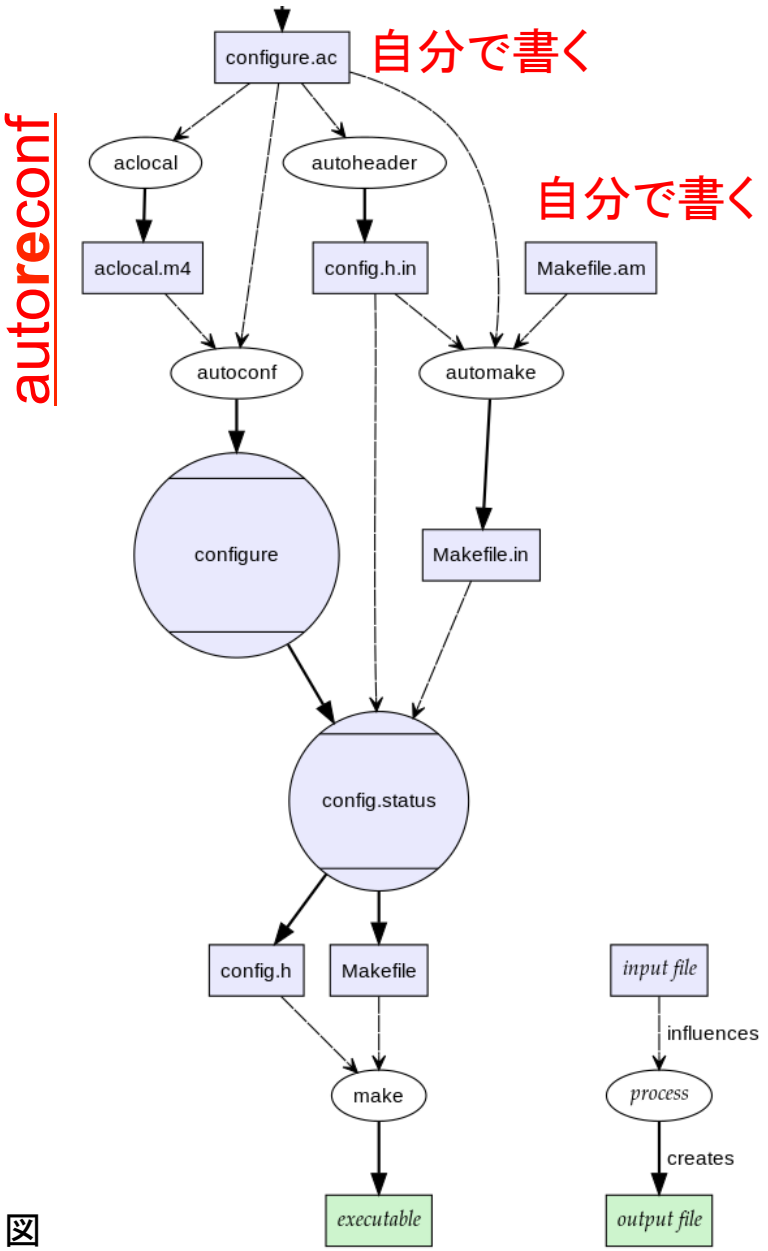


図
Autotools（autoconf と automake）による作業の流れ。実際はautoreconfが自動でやってくれる。
https://commons.wikimedia.org/wiki/File:Autoconf-automake-process.svg
より CC-BY-SA-3.0 で頒布されているもの。

Autotoolsについての注意

- Autotoolsは開発者用のツール。パッケージのユーザは そのマシンにAutotoolsをインストールする必要はないし、Autotoolsについて知っている必要もない
- ソースファイルは伝統的にsrc/以下に置く
- AutotoolsはFortranのプロジェクトにも使える
  - AC_LANG(Fortran), AC_PROG_FC(gfortran ifort)
  - 関係する変数はFC, FCFLAGS, FCLIBS, CPPFLAGS
  - Fortranでもcpp（Cプリプロセッサ）を使うと便利
  - AC_FC_FREEFORM()で自由形式のオプションがわかる
  - C言語のプロジェクトの場合はファイル間の依存関係を自動的に調べてくれるが、Fortranの場合はMakefile.amに依存関係を書いておくことがまだ必要

Autotoolsを使ってみる

```
$ svn checkout https://svn.code.sf.net/p/loto/code/feram/trunk ft
$ cd ft/
$ autoreconf -v
$ automake --add-missing    # 初回にautoreconfで実行を要求されたら
$ autoreconf -v             # ☆
$ ./configure --help        # ヘルプメッセージも自動で作ってくれる
$ ./configure
$ make                      # ソースの編集など開発を進め、最後にmake
$ make check                # 自動テスト
$ make clean                # コンパイルでできるファイルを消去
$ make distclean            # Makefileなどをきれいさっぱり消す
$ ./configure
$ make dist                 # 配布用パッケージを作成(ulmul, netpbm, libjpegも必要)
$ make distcheck            # 配布用パッケージを厳格にテストしてから作成
                  # (オプションは DISTCHECK_CONFIGURE_FLAGS で指定できる)
```

- 新たにconfigure.ac, Makefile.am, src/Makefile.amを書き変えたら☆からやり直す。

Fortranのススメ(1)

- implicit none を使いましょう
- module を common の代わりに使うのはやめましょう。そもそもいいかげん大域変数（的なもの）を使うのはやめましょう
- 自由形式 (free form) で書きましょう
- 構造体 (type) を使いましょう
- 配列の範囲が自由です a(1:N), a(0:N−1), a(−N+1:N), a(3,0:N−1,0:N−1) とか
- 配列やベクトルの演算が楽→matmul()とか
- module, interface を使えばsubroutoneの引数の不整合によるバグが避けられる

Fortranのススメ (2)

- オブジェクト指向プログラミングしましょう
- 複素数がはじめから使える
- 数学関数が豊富
- 拡張子を .F にしておけばCプリプロセッサが使える
- make(1)とAutotoolsにはFCとかFCFLAGSとかFortran用の機能があります
- LAPACKとかFFTWとかライブラリが豊富
- ポインタも使えるが積極的に使う理由はとくにない。基本、参照渡しだから。

Fortranのススメ (3)

- 動的にallocateした配列はsubroutineの最後で解放される
- 解放されたくないものは構造体の中に置く
- read/write/他 でテキスト処理も
- OpenMPで並列化できる
- command_argument_count() と get_command_argument() とでコマンドライン引数も扱える
- 浮動小数点数の精度はどう指定する？
  1.2d0と書くのはヤダ→コンパイルオプションを使う？

GNU Fortran (gfortran) のMATMUL()内部関数の最新動向

- GCC 6のgfortranからMATMUL()内部関数をインライン展開する最適化が利用可能に
  `-finline-matmul-limit=`*n*
- `-fexternal-blas` というのもある
- 積極的にMATMUL() を使って美しいコードを書きましょう

FortranでもCプリプロセッサを使う

- 技巧に走って使いすぎないこと！デバッグが困難に
- 定数を1つのファイルにまとめる
  - 物理定数
  - ファイルのユニット番号
  - *別解としては Fortran の module を使う手もある*
- Autotools (autoconf+automake) との連携
- __FILE__, __LINE__の利用
- 規格外のサブルーチン名の差異を吸収（次頁に例）
- コンパイラの差異を吸収
- ライブラリの差異を吸収
- Cプリプロセッサで処理される *.F と
  処理されない *.f とを使い分ける

例: 新バージョンでログファイルに実行したホスト名が記録されるようにした

- Fortranでホスト名の文字列を入手するための内部サブルーチンはまだ規格化されていない
  - GNU Fortran (gfortran): hostnm(*str*)
  - Intel Fortran (ifort): hostnm(*str*)
  - IBM XLF Fortran (xlf): hostnm_(*str*)
- autoconf と Cプリプロセッサ でこの違いを吸収:
  configure.ac で xlfが選択されたとき
  AC_DEFINE([hostnm], [hostnm_], [説明文])
- feram_common.F:47: HOSTNAME: app26
  とログファイルに記録されるようにした

ドキュメントとWebページの共通化

- とりあえずREADMEさえ読めばferamは使える
  - README.en（英語版）
  - README.ja（日本語版）
  - INSTALL(コンパイル方法)
  - parameter/parameter.txt(パラメータの決定方法)
- 数式の書ける独自のマークアップ言語ULMUL
- HTMLに変換してそのままWebページ
  - make distで自動変換されてパッケージにも同梱
  - 手間暇の削減
  - ドキュメントの散逸の防止

マークアップ言語のススメ

- ドキュメントはHTMLに変換しやすいマークアップ言語で書くのが吉
- ULMUL
  - 西松独自: ulmul2html5, ulmul2mathjax
  - 数式が書けて、MathMLかMathJaxに変換できる
  - GFMに比べて余計な空白行が入らない。
- GFM (GitHub Flavored Markdown)
  - GitHub標準。GitHubではREADME.mdがHTMLに変換されて表示される。
  - 美しい表が書ける。
  - gfm2html.rbとかで手元でもHTMLに変換できる

配布用パッケージのmake

- Autotools (autoconf+automake) の強力なパッケージング機能
- make distcheckでビルドディレクトリでチェックが進み、問題がなければ自動的にパッケージが完成
- オプションは configure.ac の中に指定

```
AM_INIT_AUTOMAKE([1.14.1 no-dist-gzip dist-xz tar-pax])
```

  - automakeのバージョン1.14.1以上を使用
  - パッケージをtar.gzでなくtar.xzで作る

```
$ make distcheck
   :
make  check-TESTS
   :
==========================================================
feram-0.25.02unstable archives ready for distribution:
feram-0.25.02unstable.tar.xz
==========================================================
$
```

Autotoolsの代わり

改良版のAutotoolsを目指していくつかのパッケージングツール／ビルドツールが開発されている。CMakeとSConsではFortranのプロジェクトもサポートされている。

- CMake http://www.cmake.org/
  - ALPSが採用
- SCons http://www.scons.org/
- Ninja https://github.com/martine/ninja
  - スピードに重点を置いて開発されているらしい

アプリケーション普及のために

- GPLv3のフリーソフトウェアとしてSourceForge.net から手軽にダウンロードできるようにしている
- **feramを使ってよい論文を書いて出版する**
  - 論文の出版直後にダウンロード数は急増
- ドキュメントの充実
  - チュートリアル
- メーリングリストの質問には質問者のプロファイリング をしてから答える（Debianのムトウ神の教え）
- 利用者講習会
  - なかなか本当のユーザになってもらうのは難しい
  - follow up supportが必要？
- Search Engine Optimization (SEO)

これからやりたいこと

- Nightly buildができるようにする
  - 自動 svn checkout
  - 自動 configure && make && make check
  - 自動ベンチマーク
  - 以上の結果のメールによるレポート
- ToDoリストを（SourceForge.netの）チケットシステムで管理
- Debianの公式パッケージにする
- **物理**: リラクサーのシミュレーション
  →来春の仙台の物理学会で発表予定

「こうしておけばよかった」と思う事

- データ構造をはじめにしっかり設計する（後から変えるのは大変）
- ファイル名、サブルーチン名、変数名をはじめにしっかり設計する（後から変えるのは大変）
- ぐずぐず言ってないでプログラムを書き始める。書き換え始める。公開する。（問題が出てきたら後から変えればよい）

まとめ

- ペロブスカイト型強誘電体ABO_3のための第一原理有効ハミルトニアンに基づく高速な分子動力学計算コード**feram**を開発。フリーソフトウェアとして公開。
- 電気熱量効果と弾性熱量効果のシミュレーションが可能 `feram elasto770K.1vs.feram elasto770K.2lf.feram`
- 入出力ファイルのくふう: 393K.feram→393K.log
- ビルドツールAutotoolsでパッケージングの自動化
  →リリースの作業がとても楽！
- Fortranを使う上での注意点
- ドキュメンテーションのくふう