# The Pochoir Stencil Compiler

Status Update

Feb. 23, 2012

# Story

- Stencils are prevailing
- Conventional numerical library focus on optimizing individual computation operator
- Highly cache-efficient stencil algorithm is known yet hard to write from case to case.
- How to automate the optimization of a family of computation (such as stencil) in one framework is open question.
  - Library?
  - DSL?
  - EDSL?
  - Compiler’s pragma?
  - Autotuner?

## Roadmap

- Release 0.5 (Feb. 2011)
- Release 1.0 (Mar. 2012)
- Release 2.0 (TBD)

## Release 0.5

- Released in Feb. 2011
- Published in SPAA’11 & HotPar’11
- Simple, concise, declarative, and easily verifiable DSL embedded in C++, with Intel Cilk Plus extension.
- Arbitrary shaped, arbitrary depth stencil on arbitrary d-dimensional space-time grid, with complex boundary condition.

## Current User List

- Oscar Barenys, Univ. Politechnica of Catalonia, Spain.
- Volker Strumpen, Johannes Kepler University, Austria.
- Nicolas Pinto, MIT/Harvard
- Nicolas Vasilache, Reservoir Lab.
- Patrick S. McCormick, Los Alamos National Lab.
- Mohammed Shaheen, Max Planck Institut Informatik, Germany
- Wim Vanroose, Universiteit Antwerpen, Belgium.
- Tom Henretty, Ohio State Univ.
- Protonu Basu, Univ. of Utah.
- Shoaib Kamil, Berkeley.
- Hal Finkel, Argonne National Lab.
- Matthias Christen, Klingelbergstrass, Basel, Switzerland.
- Vinayaka Bandishti, Indian Institute of Science, Bangalore, India.
- Hans Vandierendonck, Ghent University, Belguim.

## Benchmark Suite

- Physics
  - Heat equation
  - Wave equation
  - Maxwell's equation
  - Lattice Boltzmann Method
- Computational Biology
  - RNA secondary structure prediction
  - Pairwise sequence alignment
- Computational Finance
  - American Put Stock Option Pricing
- Mechanical Engineering
  - Compressible Euler Flow
- Others
  - Conway's Game of Life
  - …

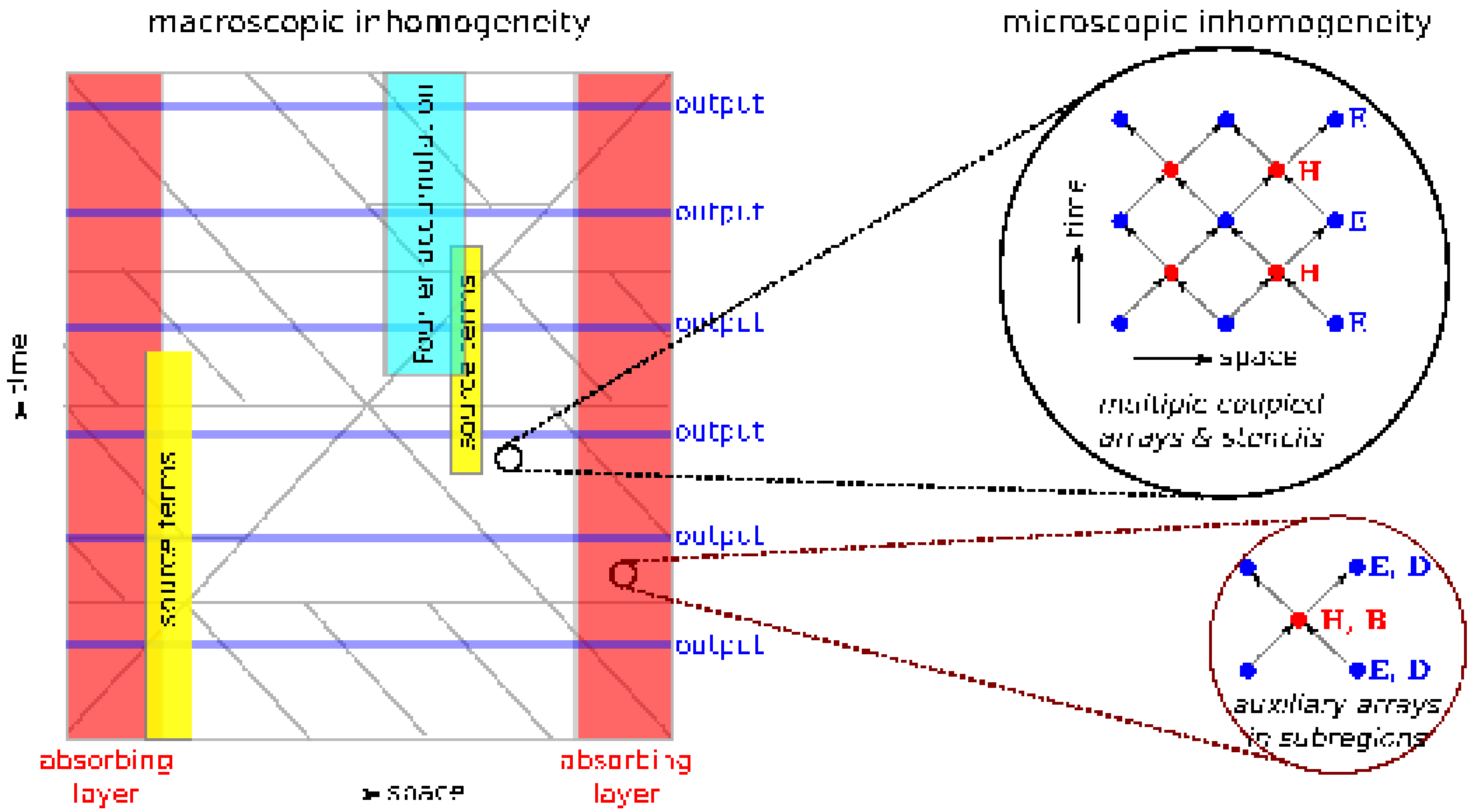
## Release 1.0

- Mar. 2012
- Bug Fixes
- User’s feedback
- Variadic Template Support
  - Even Simpler user interface

# Release 2.0

- TBD
- Inhomogeneity
  - Macroscopic Inhomogeneity
  - Microscopic Inhomogeneity
- Generalized Dependency
  - Both PUSH and PULL
  - Slope 0 cut
    - from orthogonal grid to general graph

# Inhomogeneity

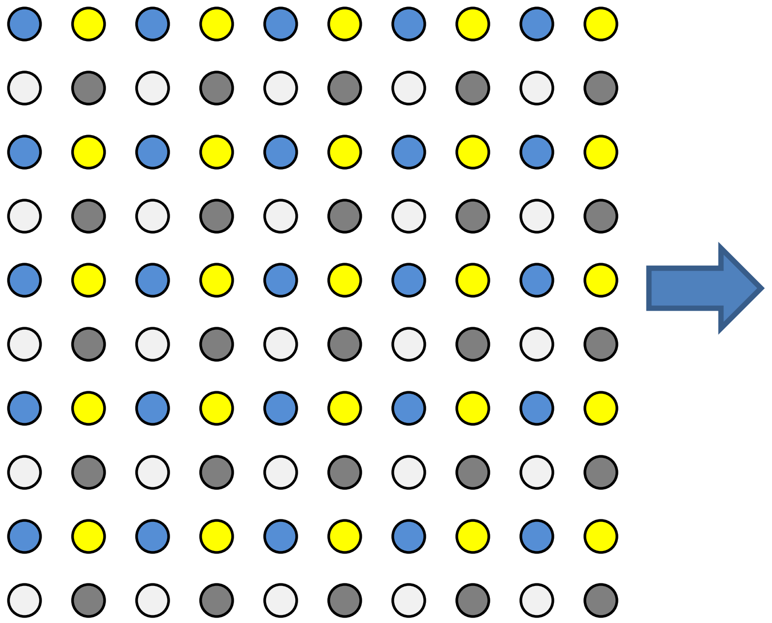


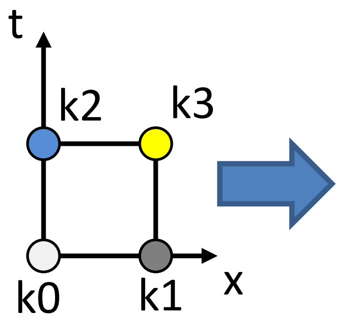
## Inhomogeneity

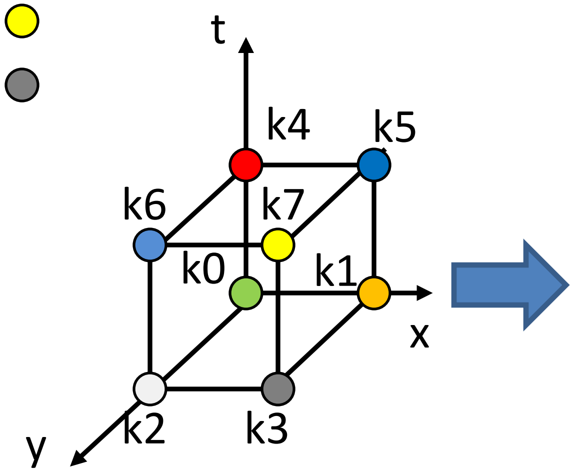
- How to specify the stencil
  - Pochoir.Register_Tile_Kernel(guard, kernel);
  - Pochoir.Register_Tile_Kernel(guard, tile);

    For macroscopic inhomogeneity
  - Tile for microscopic inhomogeneity
- How to execute the specification
  - Define our own “homogeneity” measure
    - partial order, norm, arithmetic op, etc.
  - Jitting the kernels

# What's a tile?

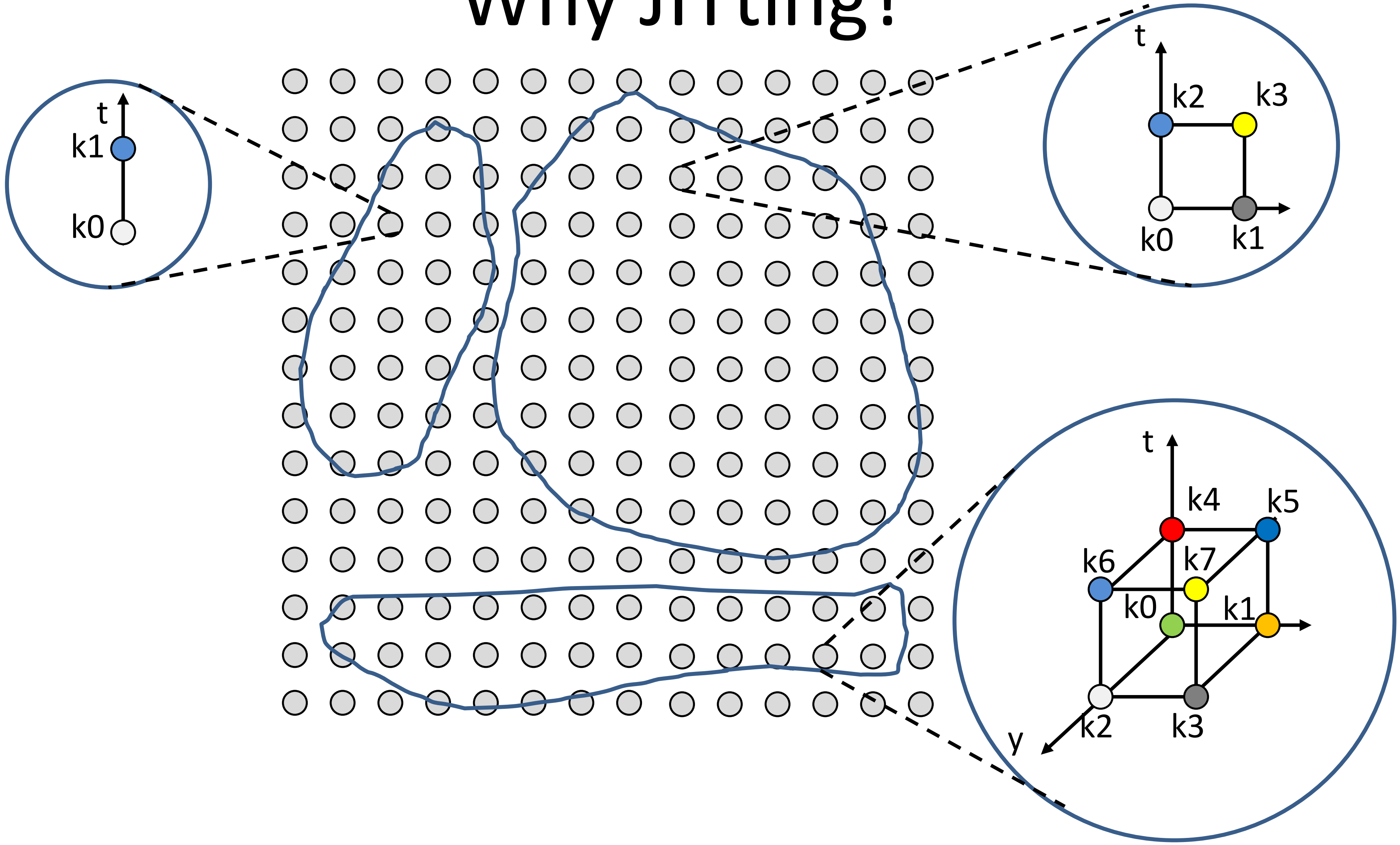




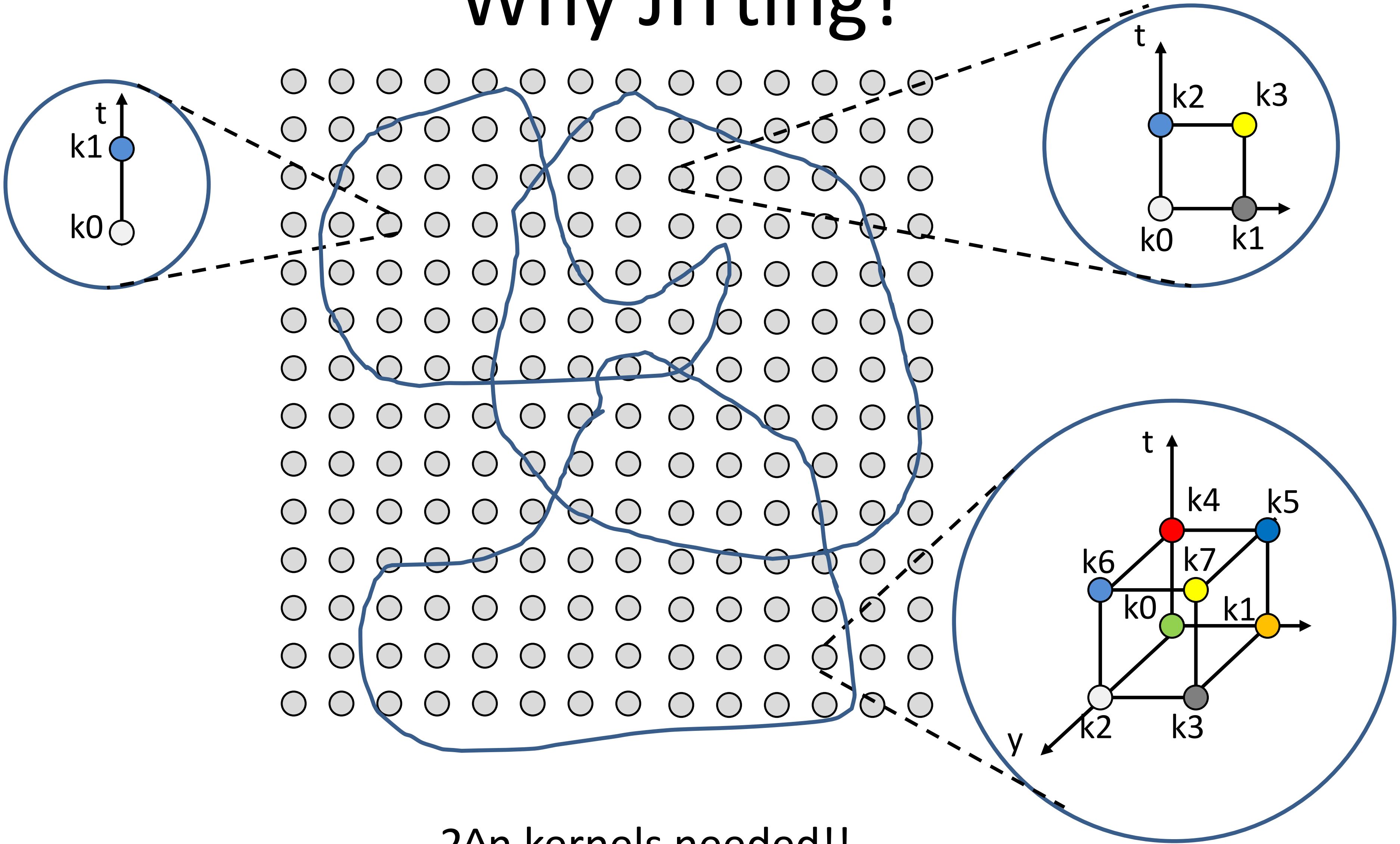


- Different color denotes different computing kernel.
- It's a checkerboard style irregular stencil computation, which tiles along both time and spatial dimension.
- Tile={{k0, k1}, {k2, k3}}:
  - Inner parenthesis denotes the tile along smaller stride dimension.
  - Outer parenthesis denotes the tile along larger stride dimension (time dimension will be the out-most loop)
  - For 2D stencil, it could be: Tile = {{{k0, k1}, {k2, k3}}, {{k4, k5}, {k6, k7}}}

# Why JITting?

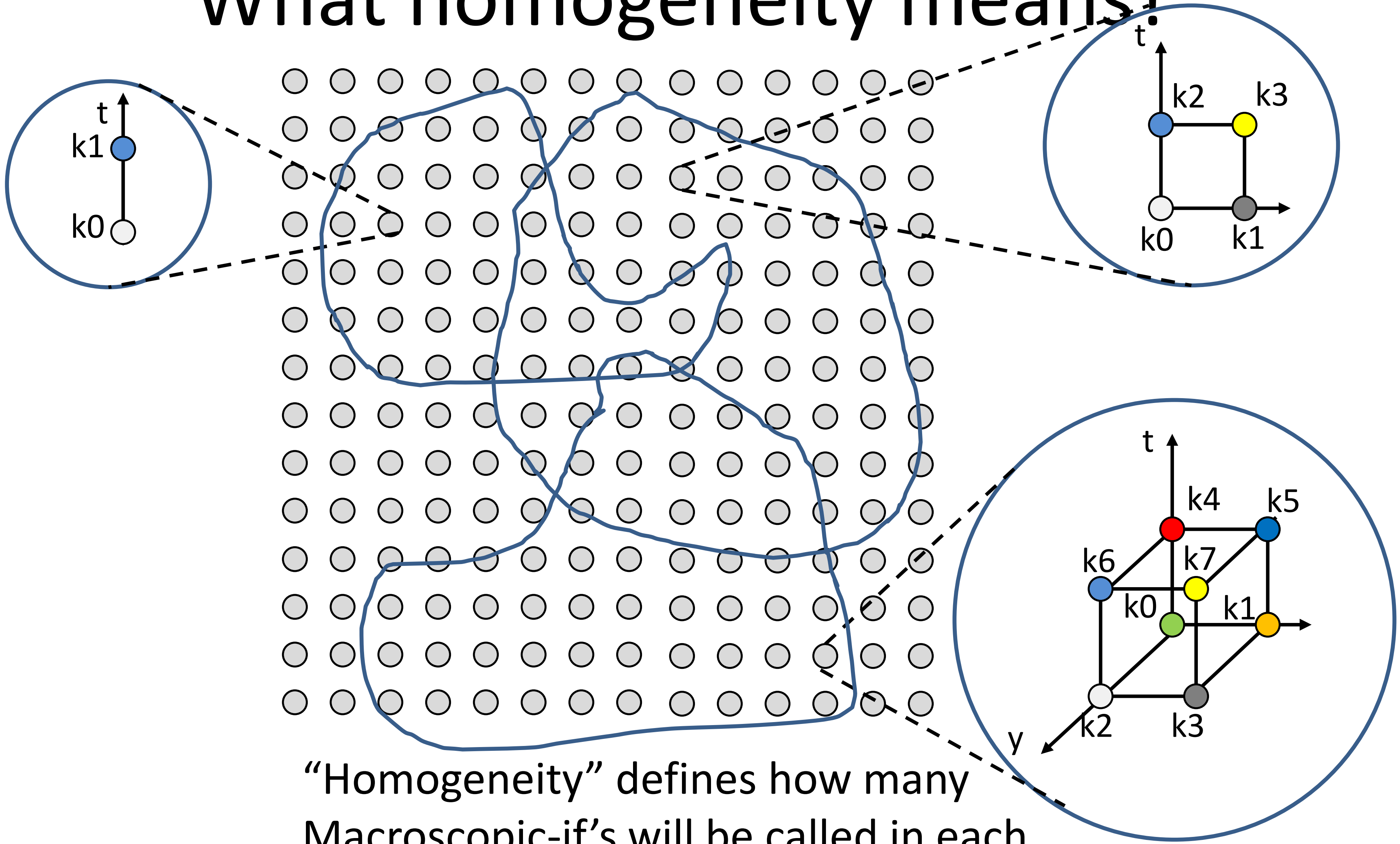


# Why JITting?



2^n kernels needed!!

# What homogeneity means?



“Homogeneity” defines how many Macroscopic-if’s will be called in each trapezoid!!

## JIT framework

- Preprocess
  - Get the homogeneity measure
  - Prepare for the later query
- Generate Only necessary (possibly pruned) kernels
  - genstencils + icpc
  - Disk as medium (what if everything in memory?)
- In the later divide-and-conquer, query and link the kernel on the fly.

## Optimization on Tile

```
if g0 {
  if (t%2==0 && i%2==0 && j%2==0) k_0_0;
  else if (t%2==0 && i%2==0 && j%2==1) k_0_1;
  else if (t%2==0 && i%2==1 && j%2==0) k_0_2;
  else if (t%2==0 && i%2==1 && j%2==1) k_0_3;
  else if (t%2==1 && i%2==0 && j%2==0) k_0_4;
  else if (t%2==1 && i%2==0 && j%2==1) k_0_5;
  else if (t%2==1 && i%2==1 && j%2==0) k_0_6;
  else if (t%2==1 && i%2==1 && j%2==1) k_0_7;}
if g1 {
  if (t%2==0 && i%2==0) k_1_0;
  else if (t%2==0 && i%2==1) k_1_1;
  else if (t%2==1 && i%2==0) k_1_2;
  else if (t%2==1 && i%2==1) k_1_3;}
if g2 {
  if (t%2==0) k_2_0;
  else if (t%2==1) k_2_1;}
if g3 {
  if (t%2==0 && i%2==0 && j%2==0) k_3_0;
  else if (t%2==0 && i%2==0 && j%2==1) k_3_1;
  else if (t%2==0 && i%2==1 && j%2==0) k_3_2;
  else if (t%2==0 && i%2==1 && j%2==1) k_3_3;
  else if (t%2==1 && i%2==0 && j%2==0) k_3_4;
  else if (t%2==1 && i%2==0 && j%2==1) k_3_5;
  else if (t%2==1 && i%2==1 && j%2==0) k_3_6;
  else if (t%2==1 && i%2==1 && j%2==1) k_3_7;}
if g4 {
  if (t%2==0 && i%2==0) k_4_0;
  else if (t%2==0 && i%2==1) k_4_1;
  else if (t%2==1 && i%2==0) k_4_2;
  else if (t%2==1 && i%2==1) k_4_3;}
if g5 {
  if (t%2==0) k_5_0;
  else if (t%2==1) k_5_1;}
```

15.9 conditional to check / kernel

# Optimization on Tile

```
if g0 {
  if (t%2==0 && i%2==0 && j%2==0) k_0_0;
  else if (t%2==0 && i%2==0 && j%2==1) k_0_1;
  else if (t%2==0 && i%2==1 && j%2==0) k_0_2;
  else if (t%2==0 && i%2==1 && j%2==1) k_0_3;
  else if (t%2==1 && i%2==0 && j%2==0) k_0_4;
  else if (t%2==1 && i%2==0 && j%2==1) k_0_5;
  else if (t%2==1 && i%2==1 && j%2==0) k_0_6;
  else if (t%2==1 && i%2==1 && j%2==1) k_0_7;}
if g1 {
  if (t%2==0 && i%2==0) k_1_0;
  else if (t%2==0 && i%2==1) k_1_1;
  else if (t%2==1 && i%2==0) k_1_2;
  else if (t%2==1 && i%2==1) k_1_3;}
if g2 {
  if (t%2==0) k_2_0;
  else if (t%2==1) k_2_1;}
if g3 {
  if (t%2==0 && i%2==0 && j%2==0) k_3_0;
  else if (t%2==0 && i%2==0 && j%2==1) k_3_1;
  else if (t%2==0 && i%2==1 && j%2==0) k_3_2;
  else if (t%2==0 && i%2==1 && j%2==1) k_3_3;
  else if (t%2==1 && i%2==0 && j%2==0) k_3_4;
  else if (t%2==1 && i%2==0 && j%2==1) k_3_5;
  else if (t%2==1 && i%2==1 && j%2==0) k_3_6;
  else if (t%2==1 && i%2==1 && j%2==1) k_3_7;}
if g4 {
  if (t%2==0 && i%2==0) k_4_0;
  else if (t%2==0 && i%2==1) k_4_1;
  else if (t%2==1 && i%2==0) k_4_2;
  else if (t%2==1 && i%2==1) k_4_3;}
if g5 {
  if (t%2==0) k_5_0;
  else if (t%2==1) k_5_1;}
```

15.9 conditional to check / kernel

```
if (t%2==0) {
  if (i%2==0) {
    if (j%2==0) if g0 k_0_0;
    else if (j%2==1) if g0 k_0_1;
    if g1 k_1_0;
  } else if (i%2==1) {
    if (j%2==0) if g0 k_0_2;
    else if (j%2==1) if g0 k_0_3;
    if g1 k_1_1;
  }
  if g2 k_2_0;
  if (i%2==0) {
    if (j%2==0) if g0 k_3_0;
    else if (j%2==1) if g0 k_3_1;
    if g4 k_4_0;
  } else if (i%2==1) {
    if (j%2==0) if g3 k_3_2;
    else if (j%2==1) if g3 k_3_3;
    if g4 k_4_1;
  }
  if g5 k_5_0;
} else if (t%2==1) {
  ……
}
```

4.6 conditional to check / kernel

Can we do better?

## Contributions

- Simple, concise, declarative, and easily verifiable DSL embedded in C++, with Intel Cilk Plus extension.
- Arbitrary shaped, arbitrary depth stencil on arbitrary d-dimensional space-time grid, with complex boundary condition.
- Multiple inhomogeneous kernels (possibly overlapping)
  - Macroscopic inhomogeneity
  - Microscopic inhomogeneity
- Generalized dependency
  - From orthogonal grid to general graph
- JIT compiler for stencil
  - Balance the # if conditionals in the inner-most loop and # kernels to generate