6.506: Algorithm Engineering

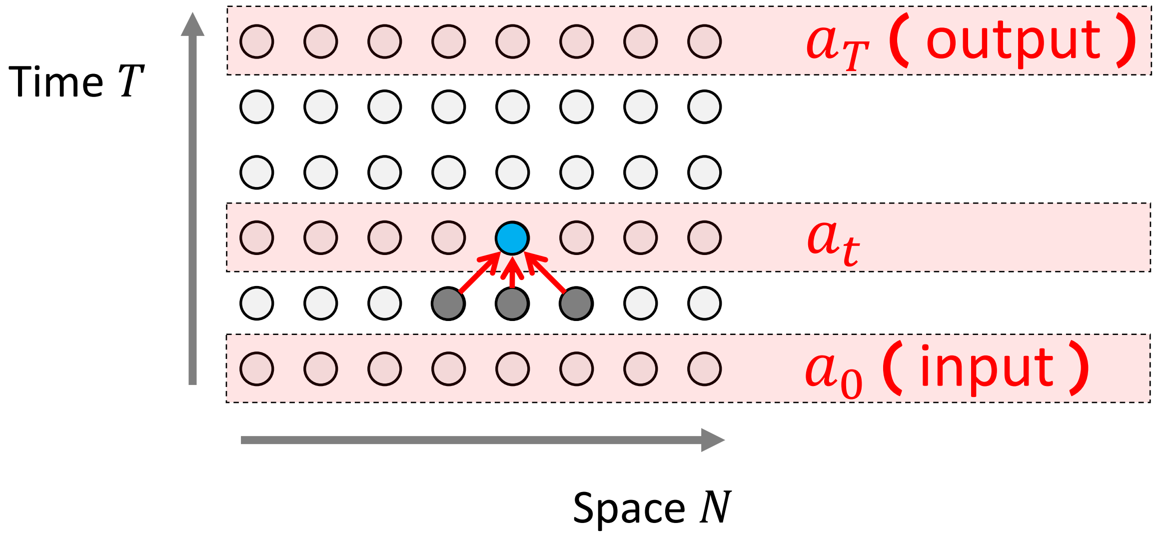
Lecture 5
(Stencil Computations)

Rezaul Chowdhury

Department of Computer Science
Stony Brook University

February 23, 2023

What is a Stencil Computation?



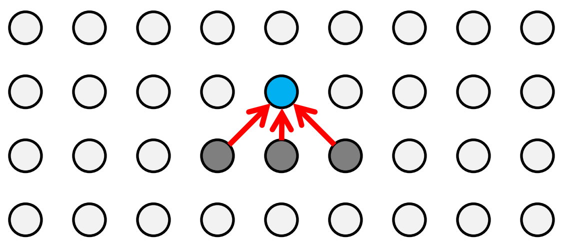
We are given spatial (input) grid data a_0 at time 0.

A stencil S computes a cell value in a spatial grid a_t at time t from nearby values at times before t.

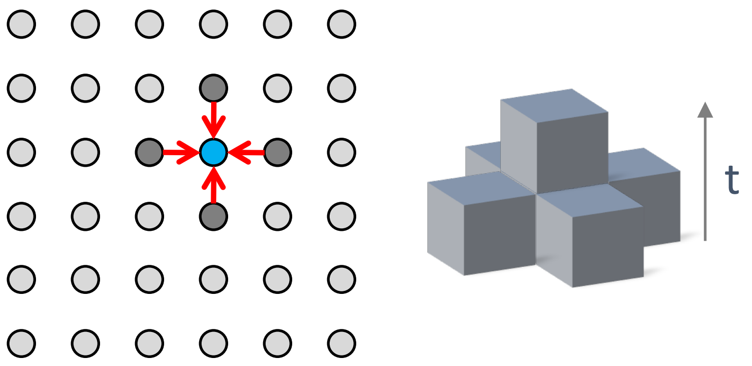
A stencil computation applies S to a_o for a given T number of timesteps to compute the output grid a_T.

Examples of Stencils

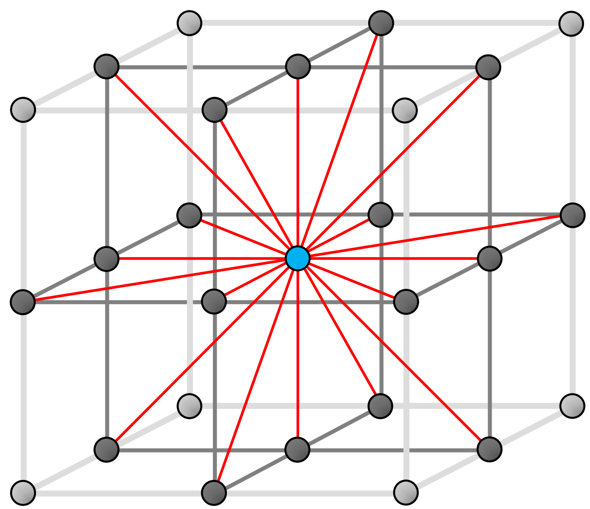
1D 3-point stencil



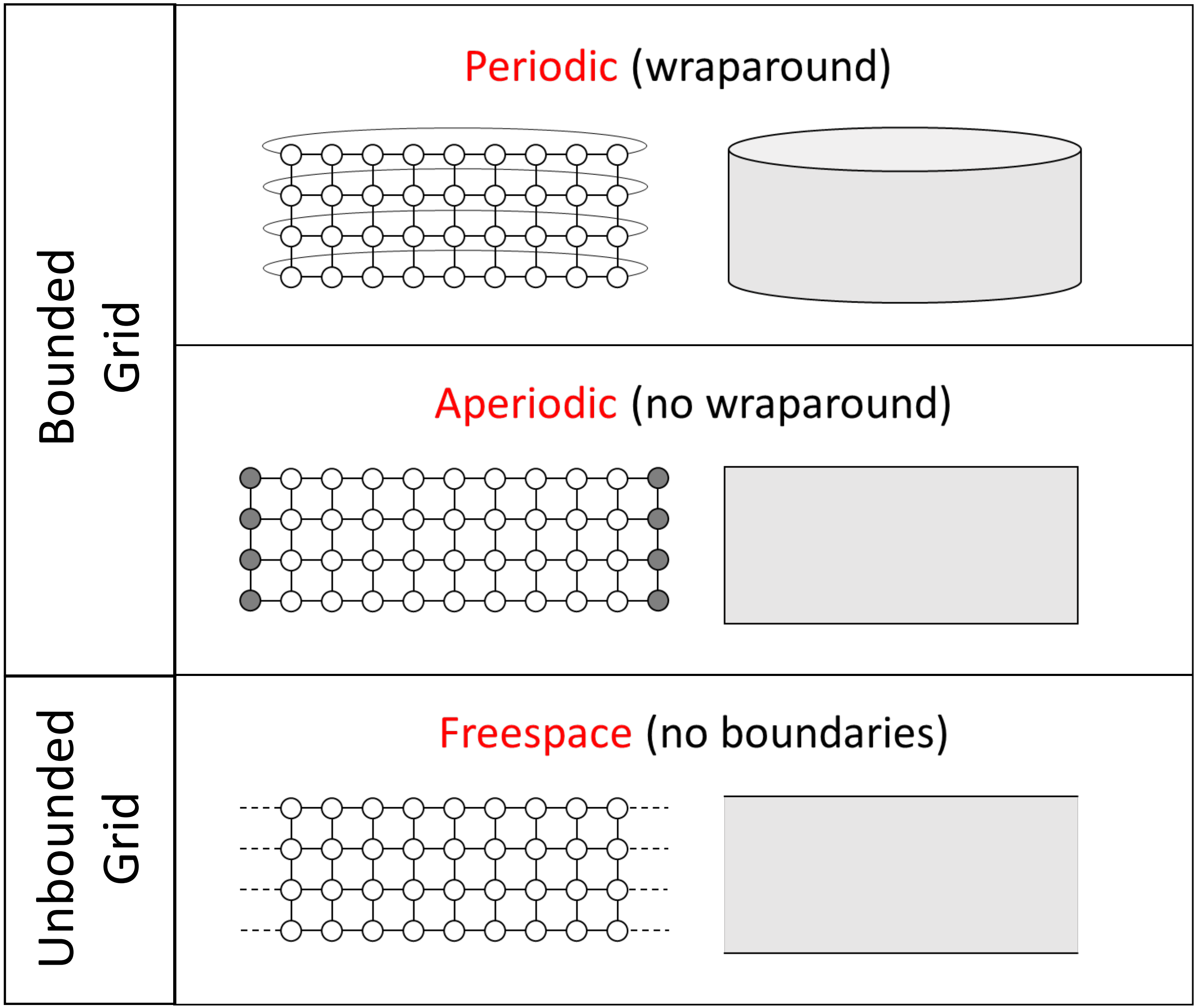
2D 5-point stencil



3D 19-point stencil

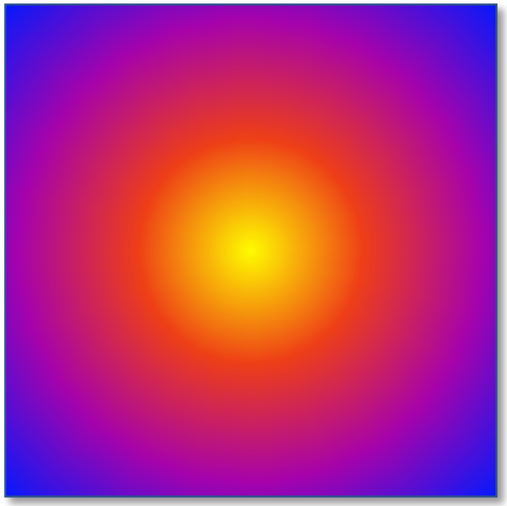


Types of Grids



Example: 2D Heat Diffusion

Let $h_t(x, y)$ be the heat at point (x, y) at time t.



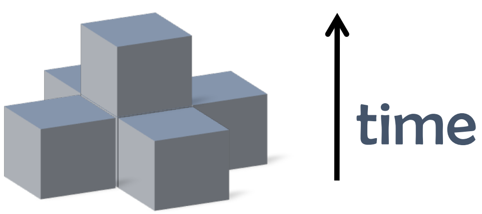
Heat Equation

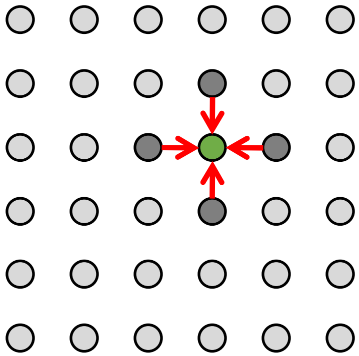
$$\frac{\partial h}{\partial t} = \alpha\left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2}\right), \quad \alpha = \text{thermal diffusivity}$$

Update Equation (on a discrete grid)

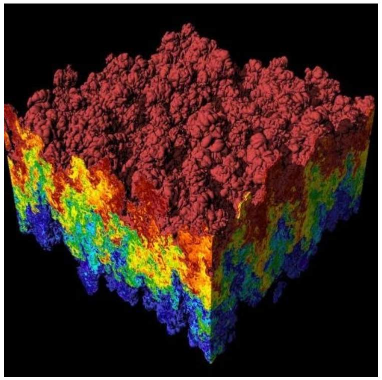
$$\begin{aligned} h_{t+1}(x, y) = h_t(x, y) & \\ & + c_x\big(h_t(x+1, y) - 2h_t(x, y) + h_t(x-1, y)\big) \\ & + c_y\big(h_t(x, y+1) - 2h_t(x, y) + h_t(x, y-1)\big) \end{aligned}$$

2D 5-point Stencil

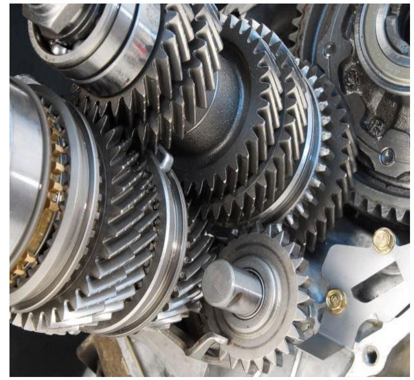




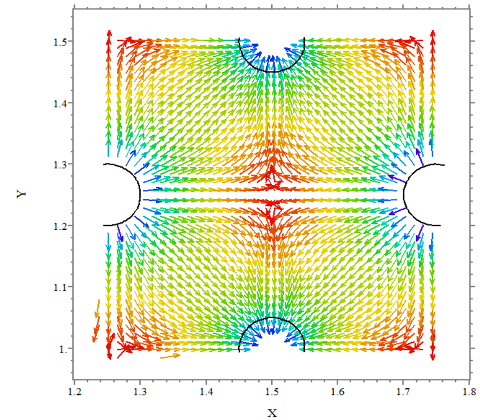
Stencil Applications



Fluid Dynamics



Mechanical Engg.



Electromagnetics

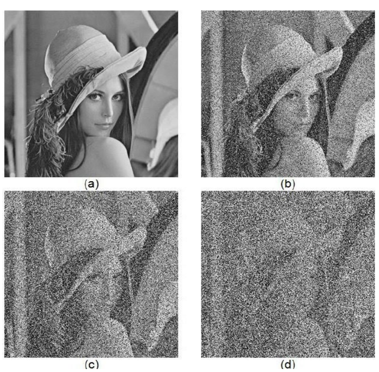
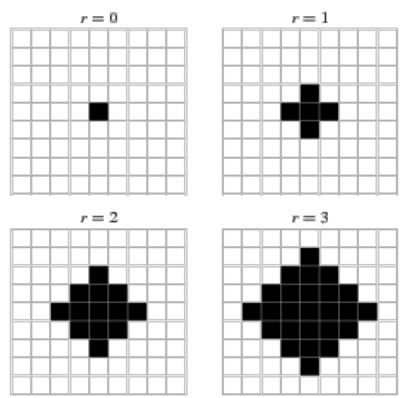


Image Processing

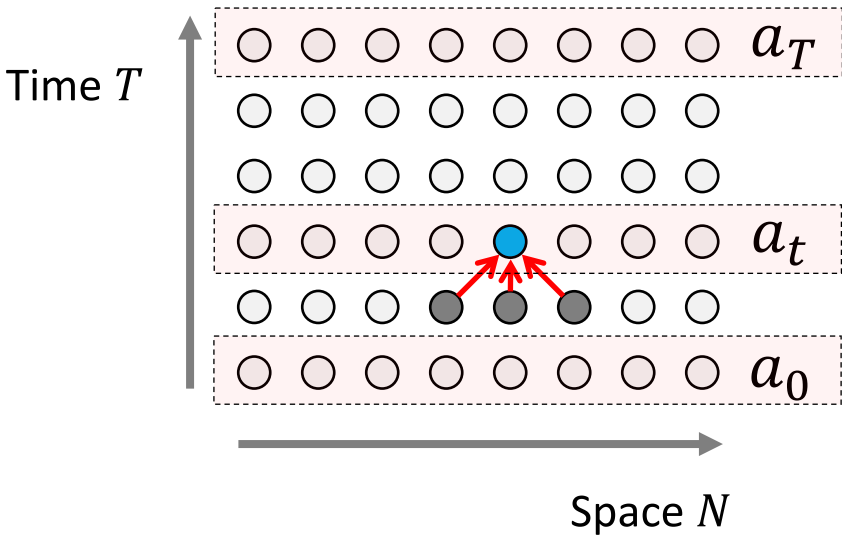


Cellular Automata

Meteorology

Standard $\Theta(NT)$ Work Stencil Algorithms

Standard Looping Algorithm

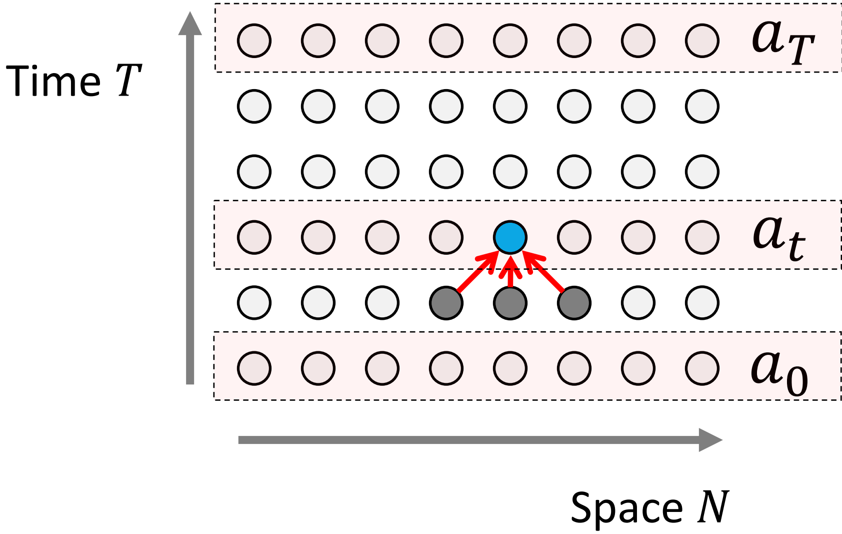



```
for t ← 1 to T do
    for i ← 0 to N − 1 do
        compute a_t[i] from a_{t−1}
        using the stencil
```

Implementation Trick: Reuse storage for odd and even time steps

```
for t ← 1 to T do
    k ← t mod 2
    for i ← 0 to N − 1 do
        compute a_k[i] from a_{1−k}
        using the stencil
```

Standard Looping Algorithm




```
for t ← 1 to T do
    k ← t mod 2
    par for i ← 0 to N − 1 do
        compute a_k[i] from a_{1−k}
        using the stencil
```

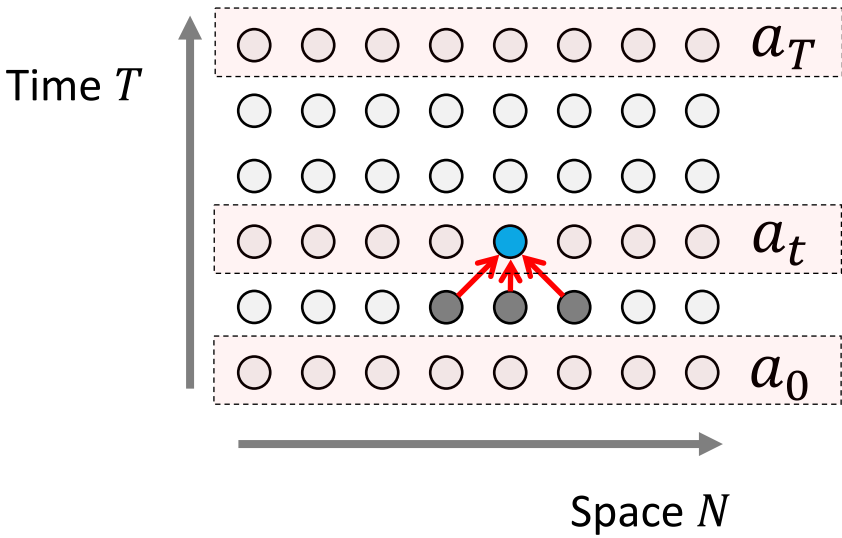
Work: Computes each of the TN cells exactly once in $\Theta(1)$ time.

$$\boxed{\text{Total work, } T_1(T,N) = \Theta(TN)}$$

Span: In each time step computes all N cells in parallel.

$$\boxed{\text{Total span, } T_\infty(T,N) = \Theta(T \log N)}$$

Standard Looping Algorithm




```
for t ← 1 to T do
    k ← t mod 2
    for i ← 0 to N − 1 do
        compute a_k[i] from a_{1−k}
        using the stencil
```

Serial Cache Complexity: It performs T sequential scans of a_0 and a_1.

$$\boxed{\text{Serial cache complexity, } Q_1(T,N) = O\left(\frac{TN}{B}\right),}$$

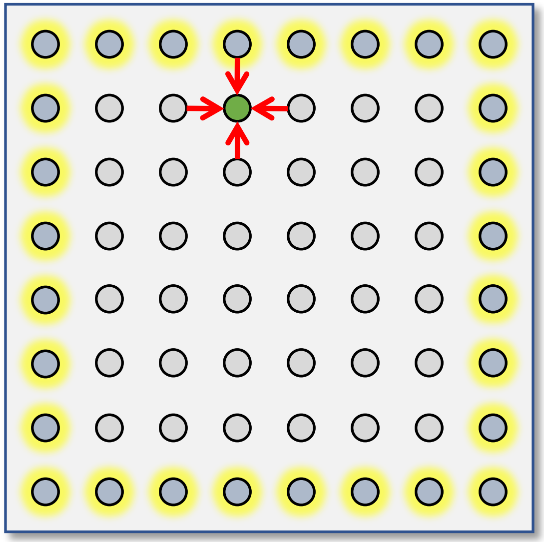
where, B = cache line size.

Standard Looping Algorithm

Implementation Tricks

– Reuse storage for odd and even time steps

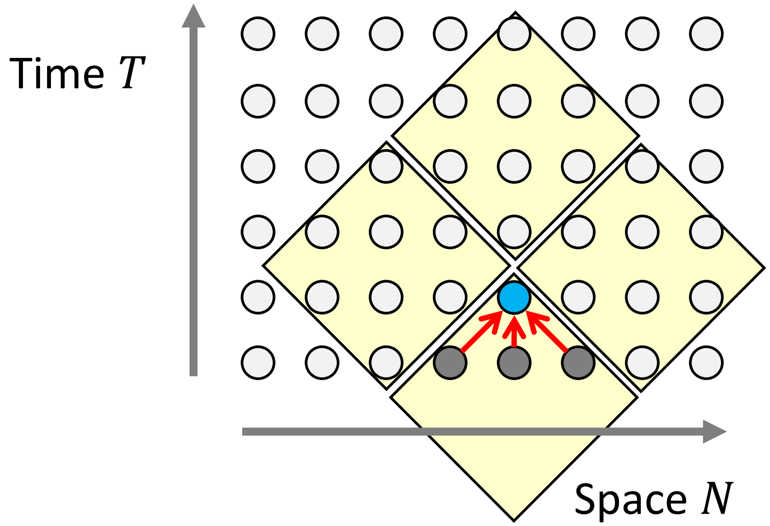
– Keep a halo of ghost cells around the array with boundary values



```
for ( int t = 0; t < T; ++t )
 {
  for ( int x = 1; x <= X; ++x )
    for ( int y = 1; y <= Y; ++y )
      g[x][y] = h[x][y]
              + cx * ( h[x+1][y] - 2 * h[x][y] + h[x-1][y] )
              + cy * ( h[x][y+1] - 2 * h[x][y] + h[x][y-1] );

  for ( int x = 1; x <= X; ++x )
    for ( int y = 1; y <= Y; ++y )
      h[x][y] = g[x][y];
 }
```

Tiled-looping Algorithm



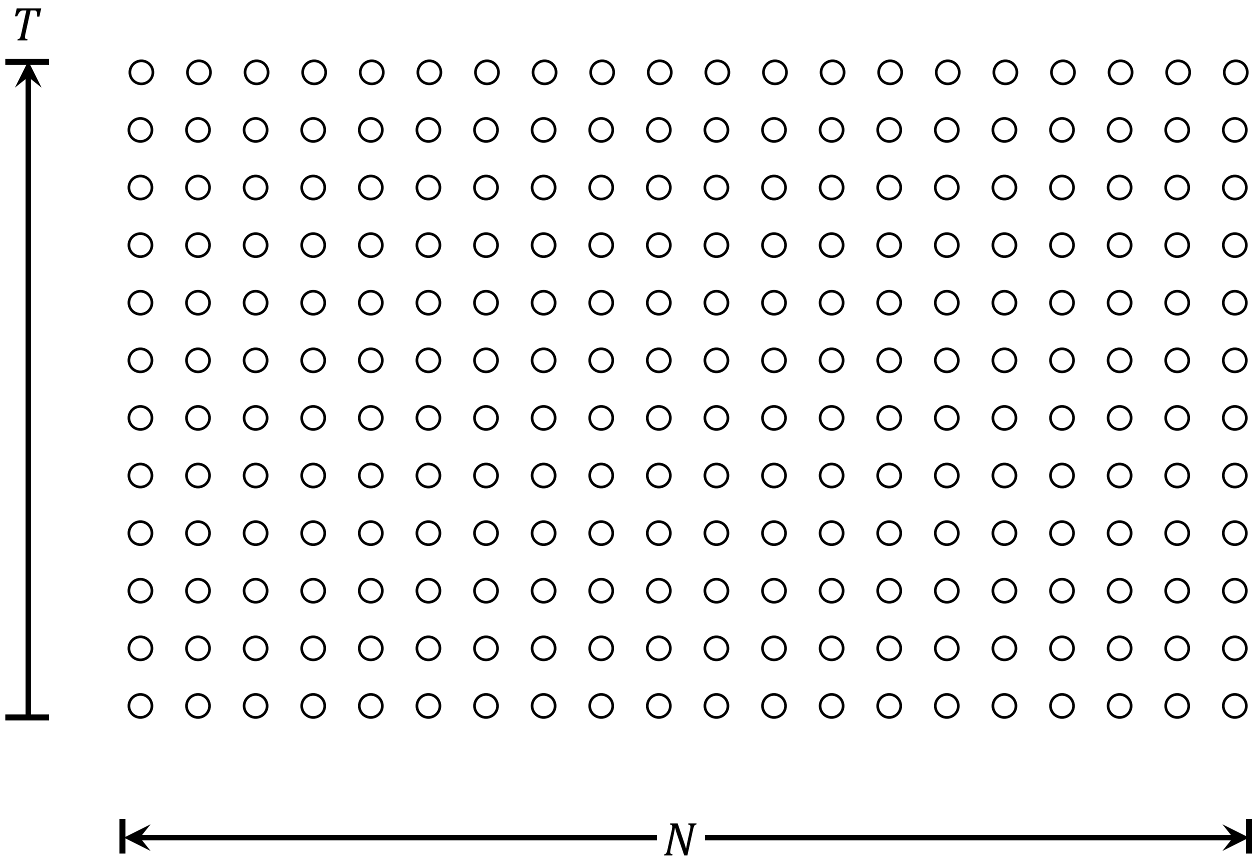
Cache-optimized version of the standard looping code.

The computation proceeds in blocks/tiles to achieve temporal data locality.

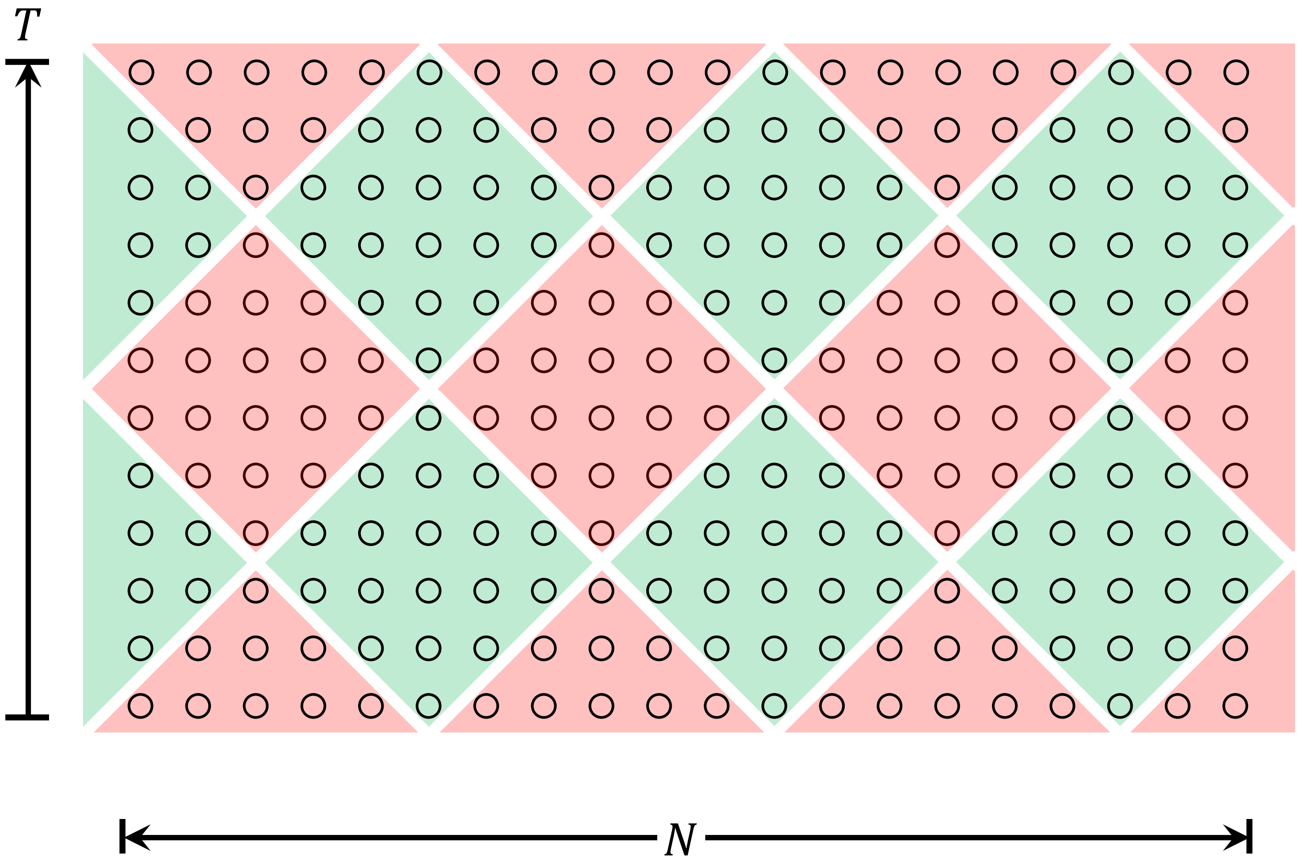
Cache-aware: cache size M must be known for blocking.

Code generators: PLuTo, Devito, etc.

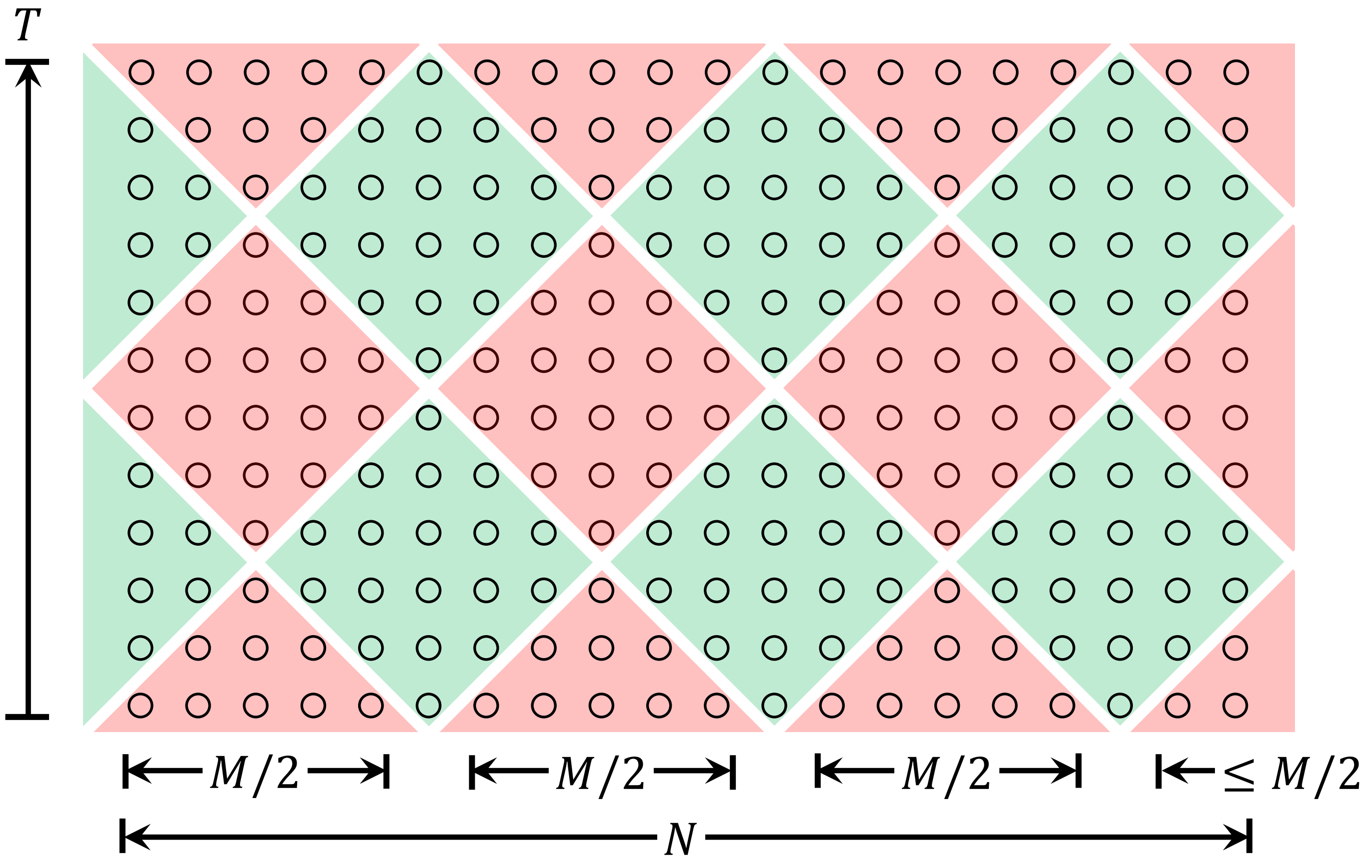
Tiled-looping Algorithm



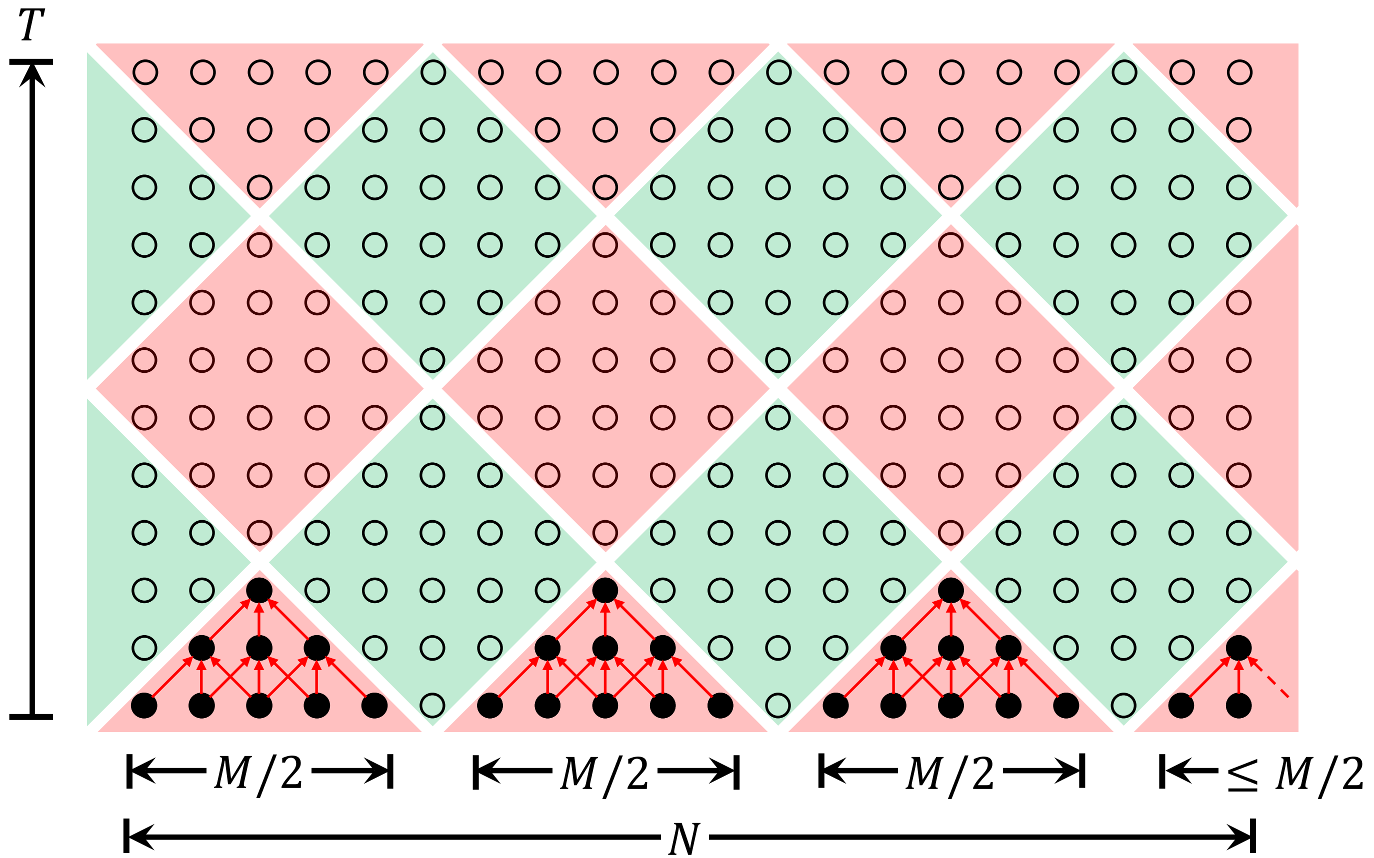
Tiled-looping Algorithm



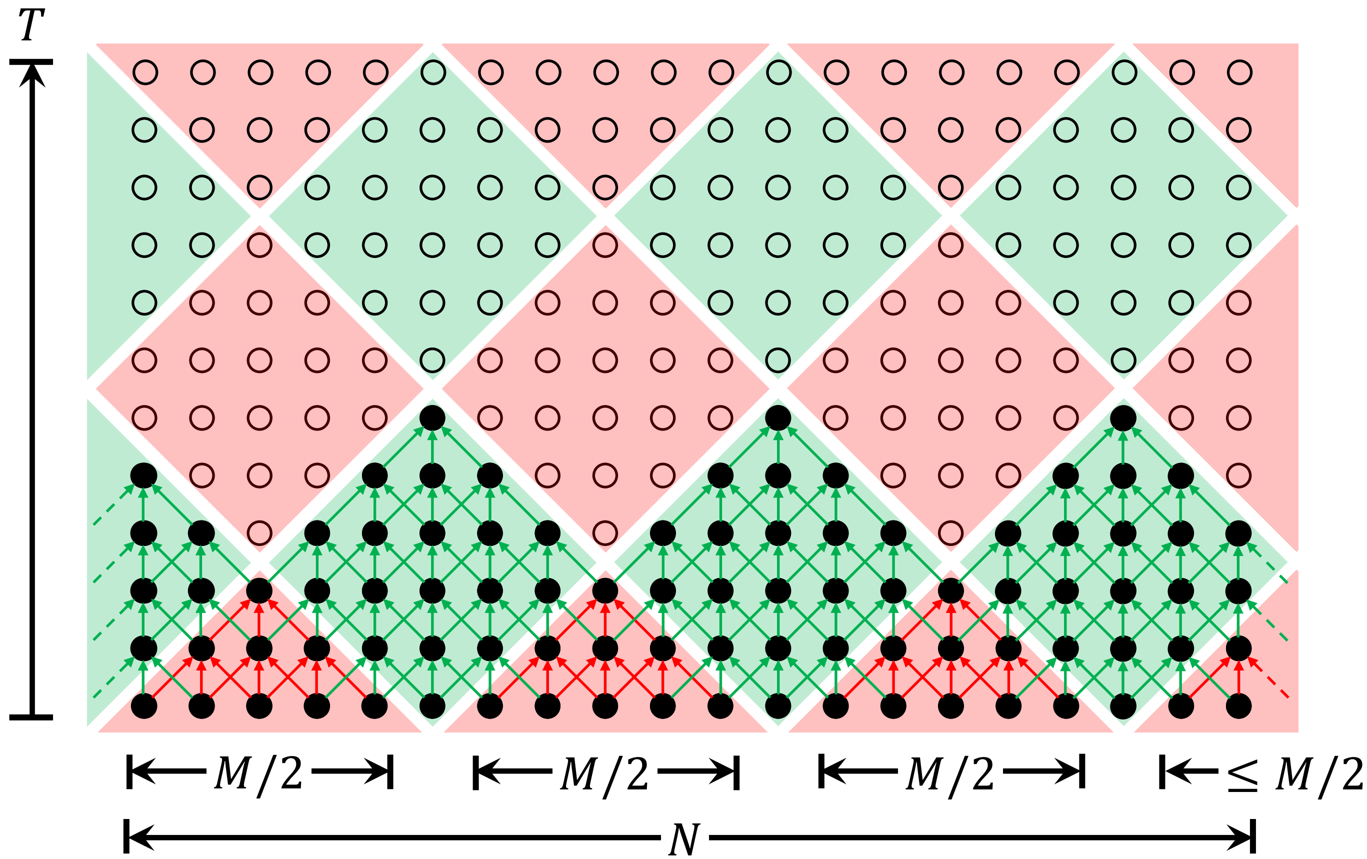
Tiled-looping Algorithm



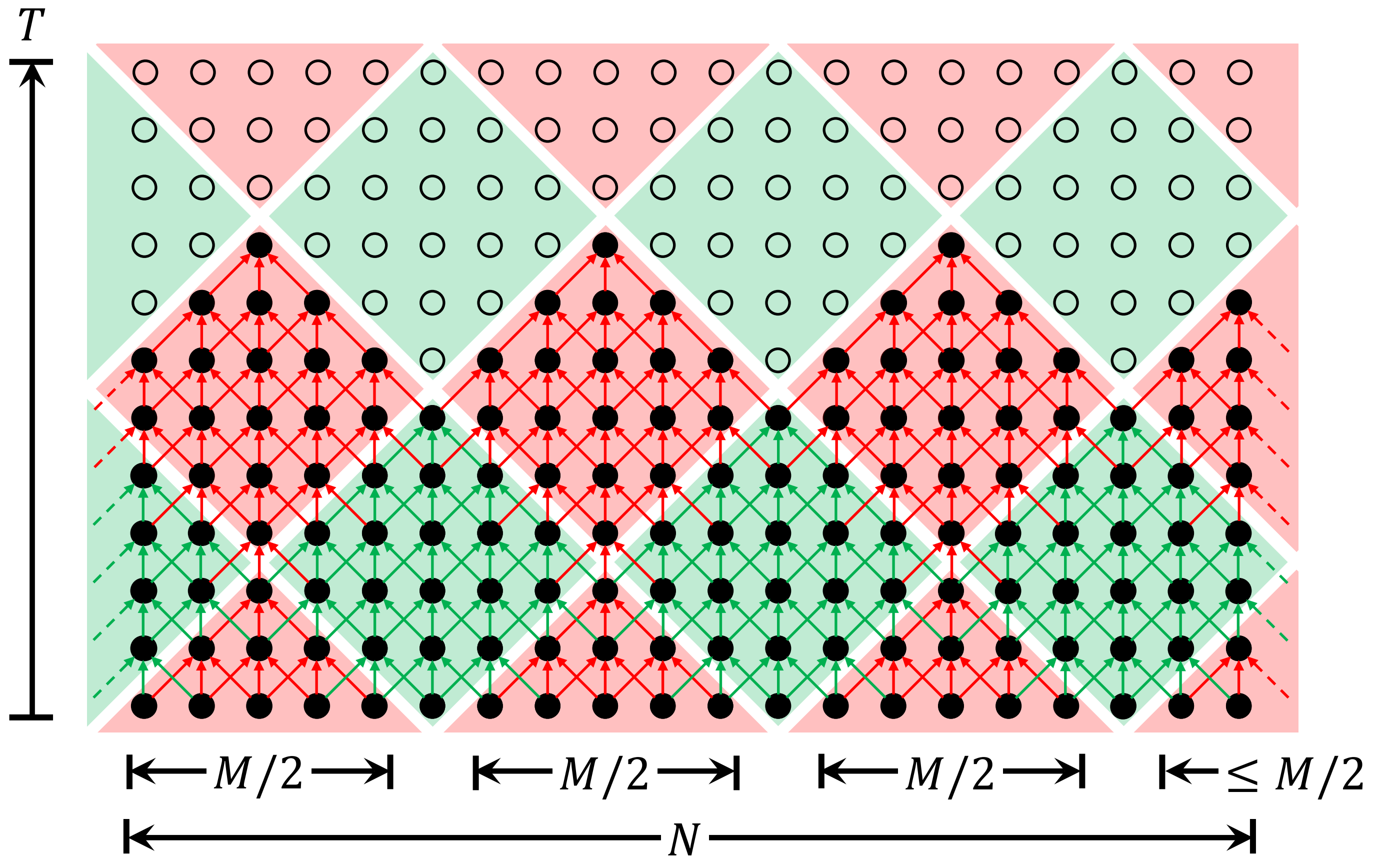
Tiled-looping Algorithm



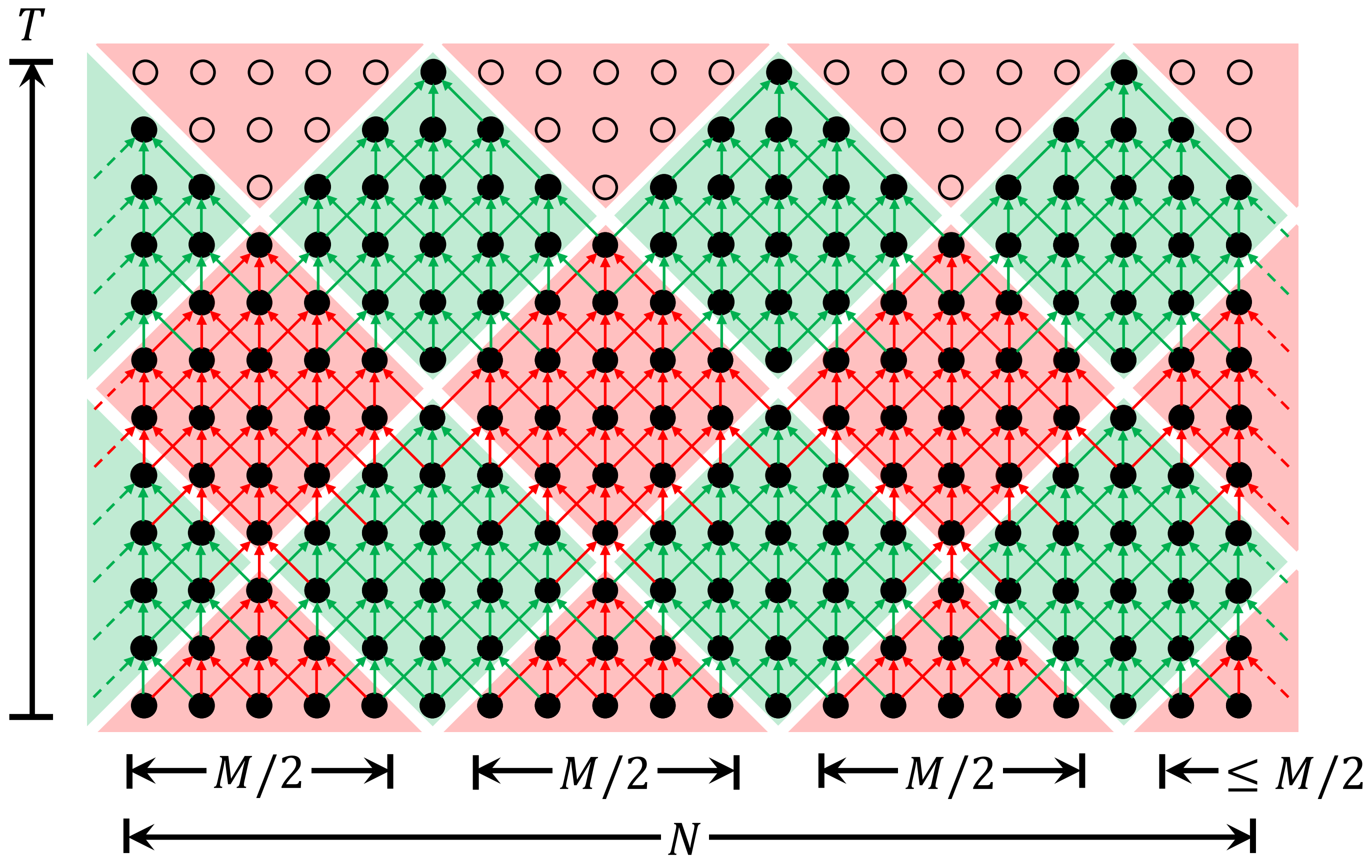
Tiled-looping Algorithm



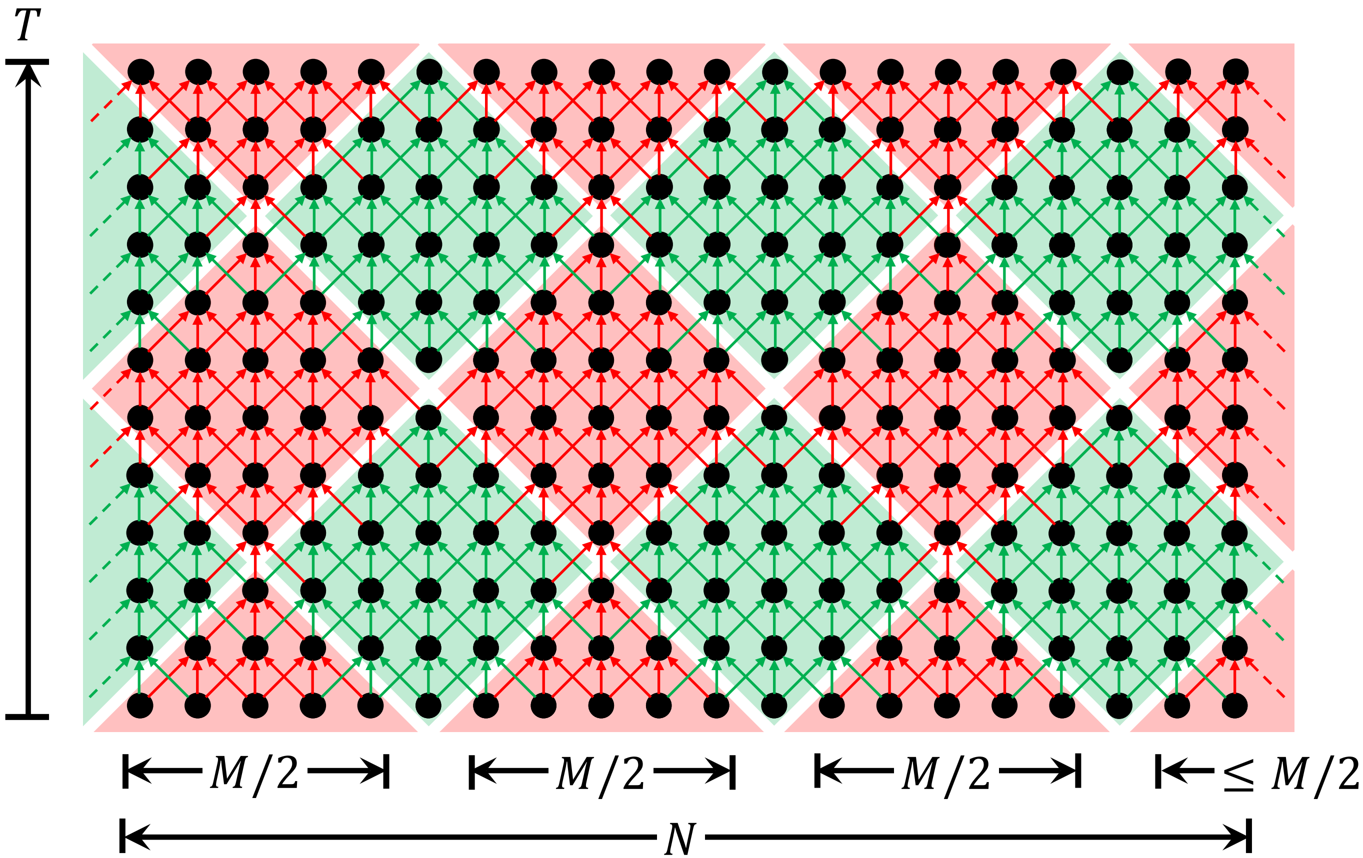
Tiled-looping Algorithm



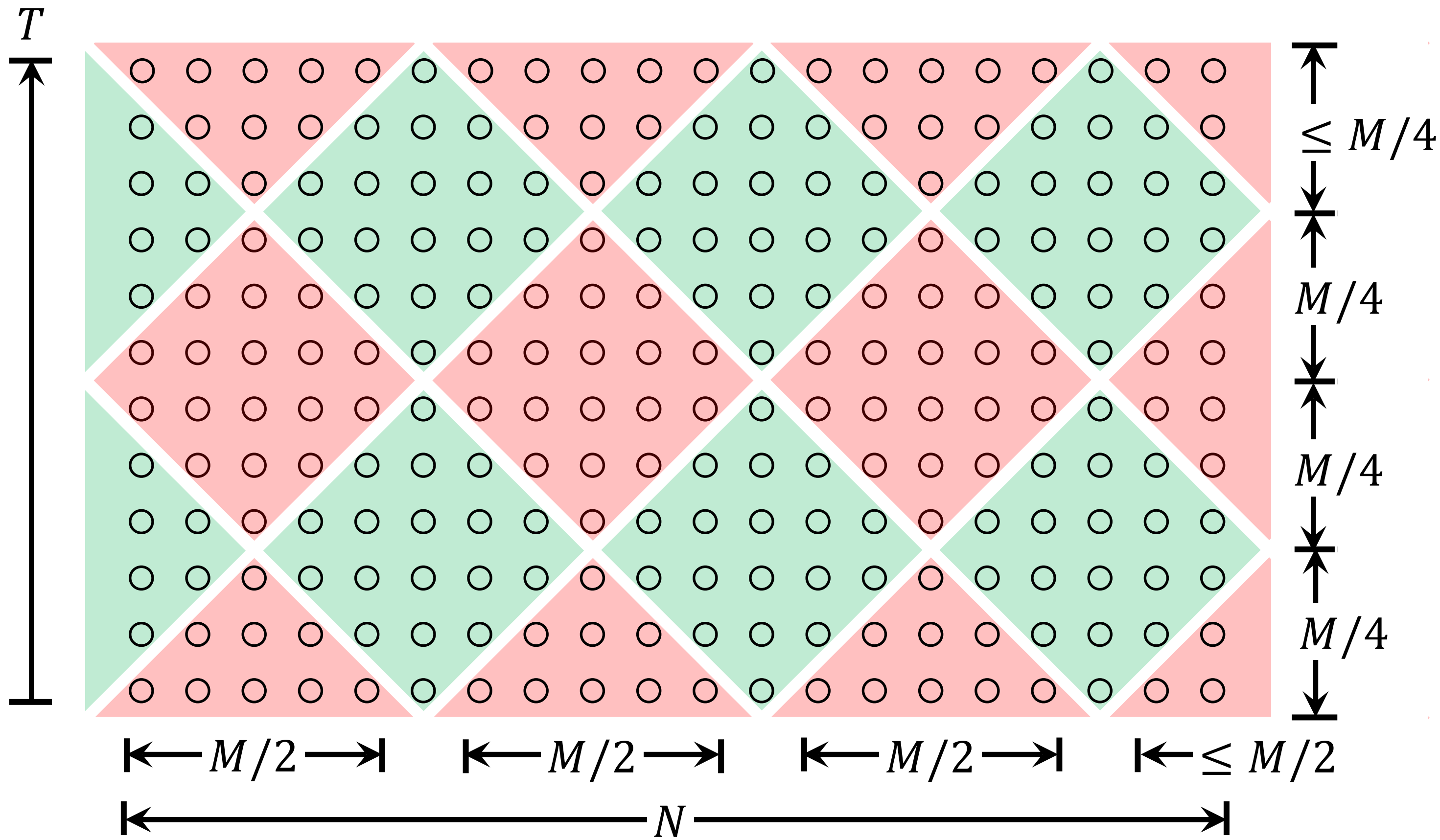
Tiled-looping Algorithm



Tiled-looping Algorithm



Tiled-looping Algorithm

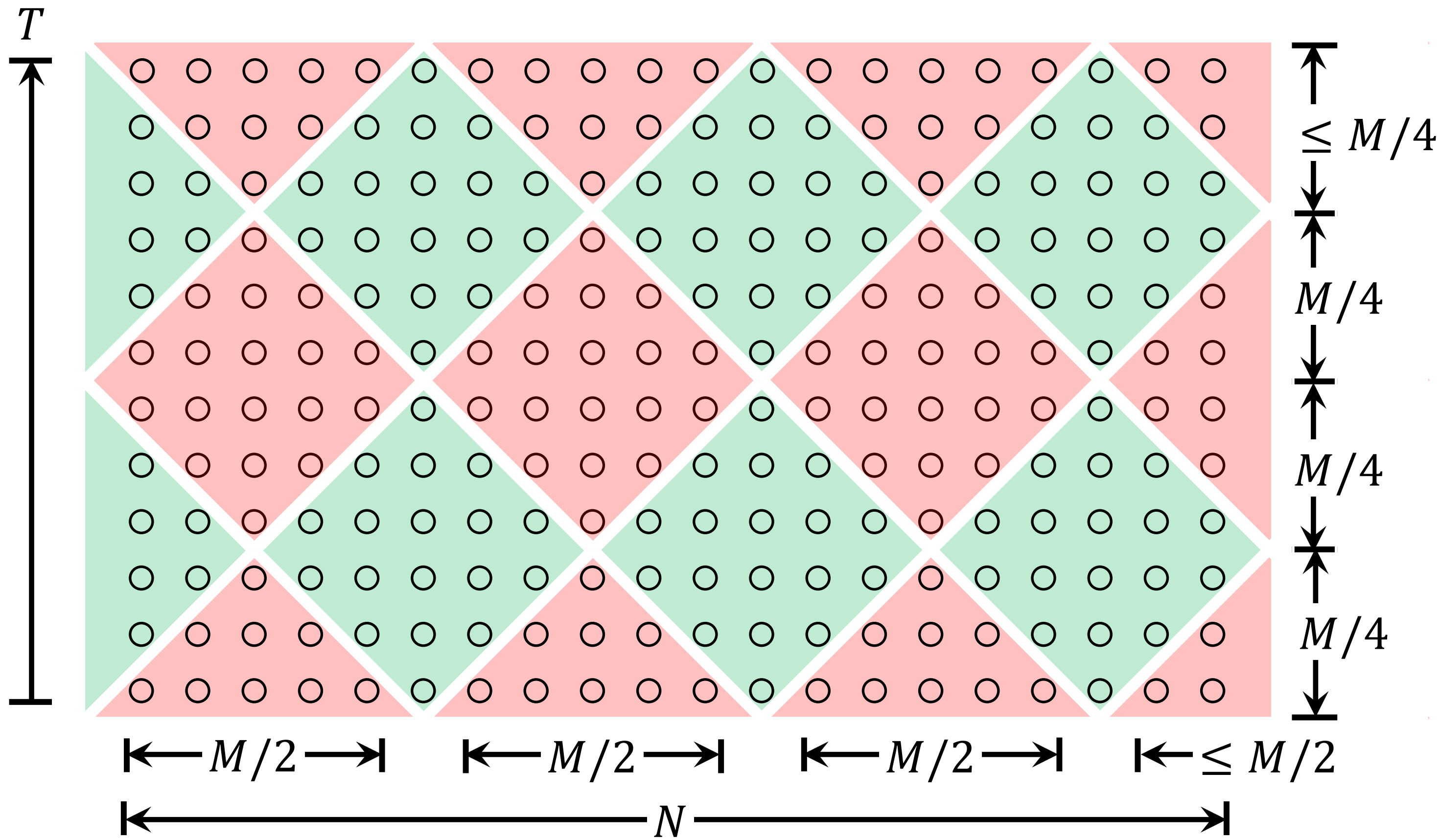


For a (full) tile: Area, $A = \Theta\left(M^2\right)$, Perimeter, $P = \Theta(M)$

Number of tiles, $K = \Theta\left(\frac{TN}{A}\right) = \Theta\left(\frac{TN}{M^2}\right)$

Number of rows of tiles, $R = \Theta\left(\frac{T}{M}\right)$

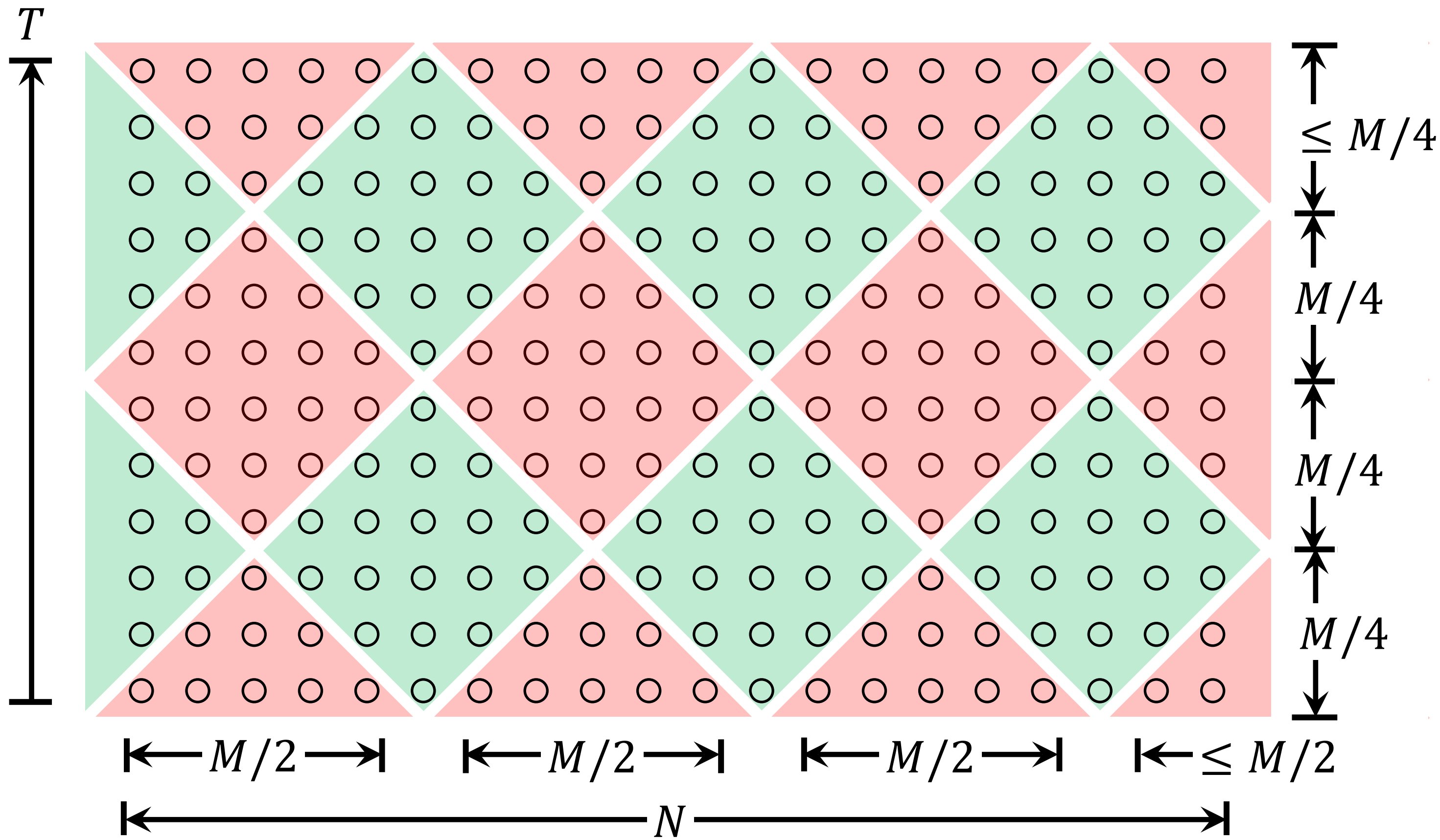
Tiled-looping Algorithm



Each grid cell is computed exactly once in $\Theta(1)$ work.

$\boxed{\text{Total work, } T_1(T, N) = \Theta(TN)}$

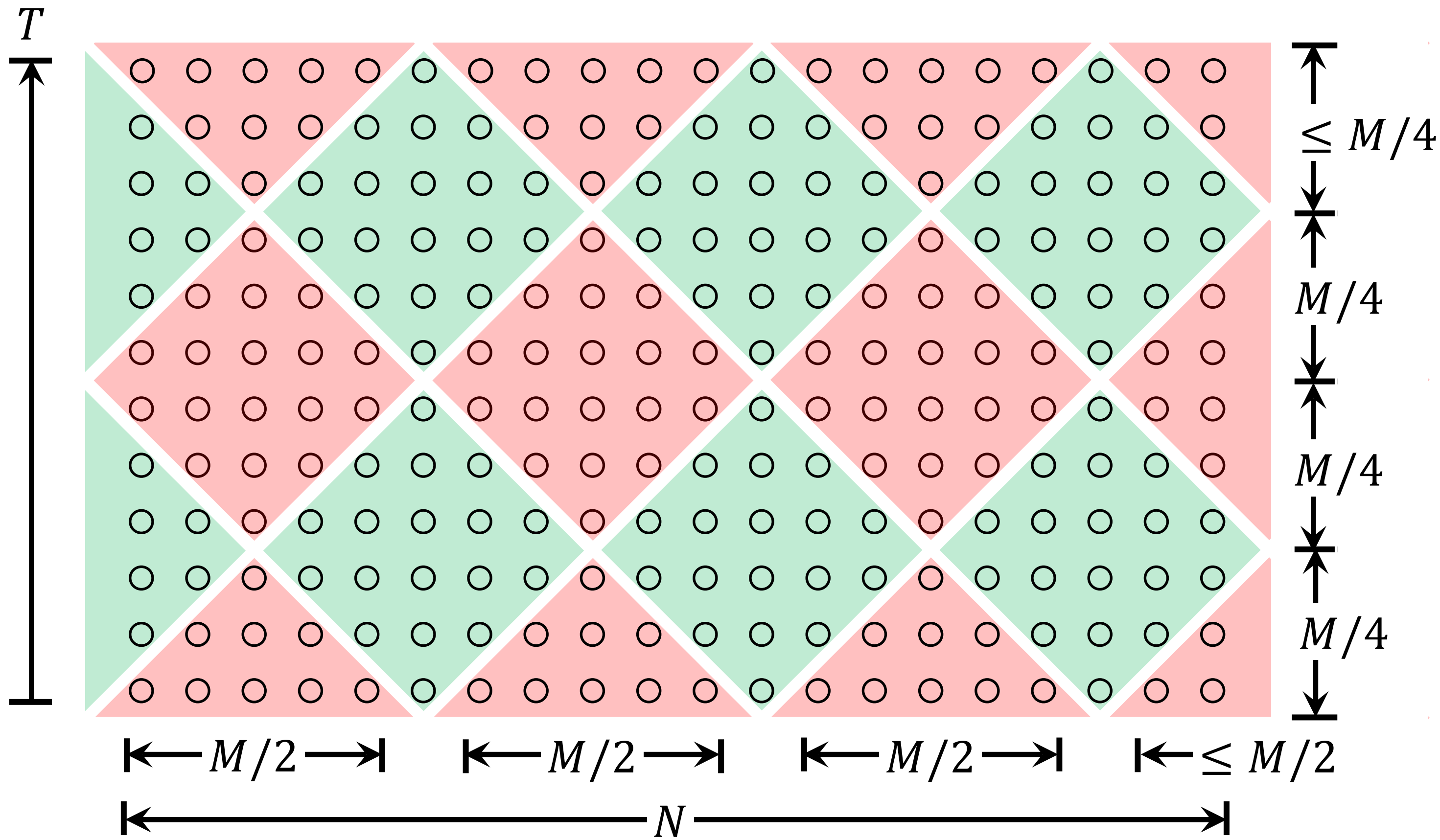
Tiled-looping Algorithm



Number of tiles in a row $= \Theta(K/R) = \Theta(N/M)$

$$\text{Total span, } T_\infty(T,N) = \Theta\left(R\left(\log\left(\frac{N}{M}\right) + M\log M\right)\right)$$
$$= \Theta\left(\left(\frac{T}{M}\right)\log\left(\frac{N}{M}\right) + T\log M\right)$$

Tiled-looping Algorithm

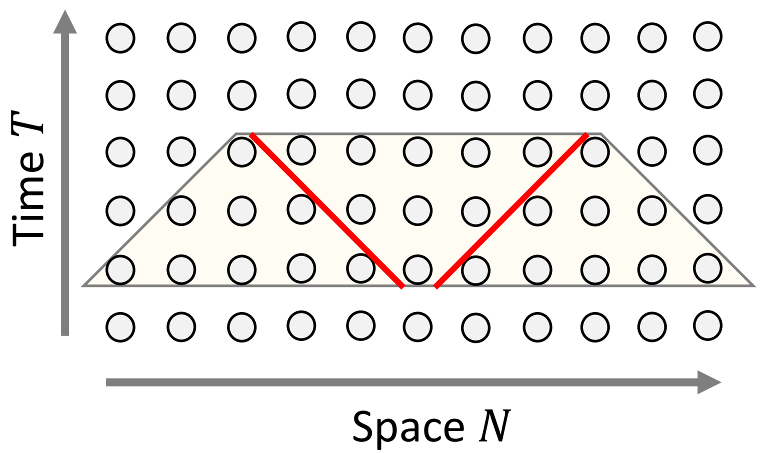


Serial cache complexity, $Q_1(T,N) = O\big(K(P/B)\big)$

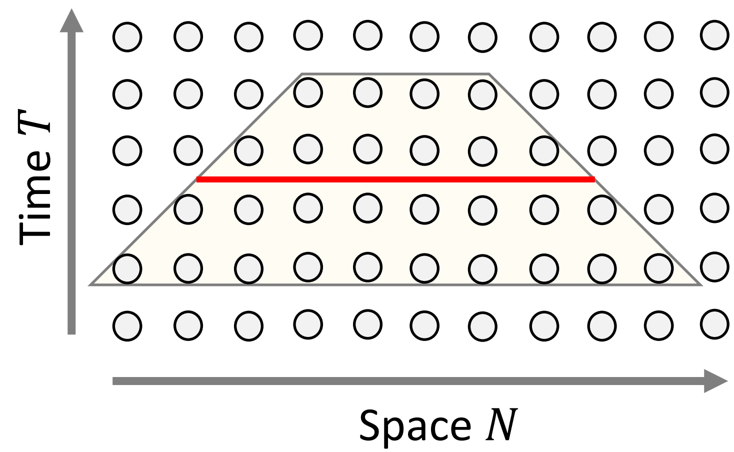
$$= O\left(\frac{TN}{MB}\right)$$

Trapezoidal Decomposition Algorithm

Space cut



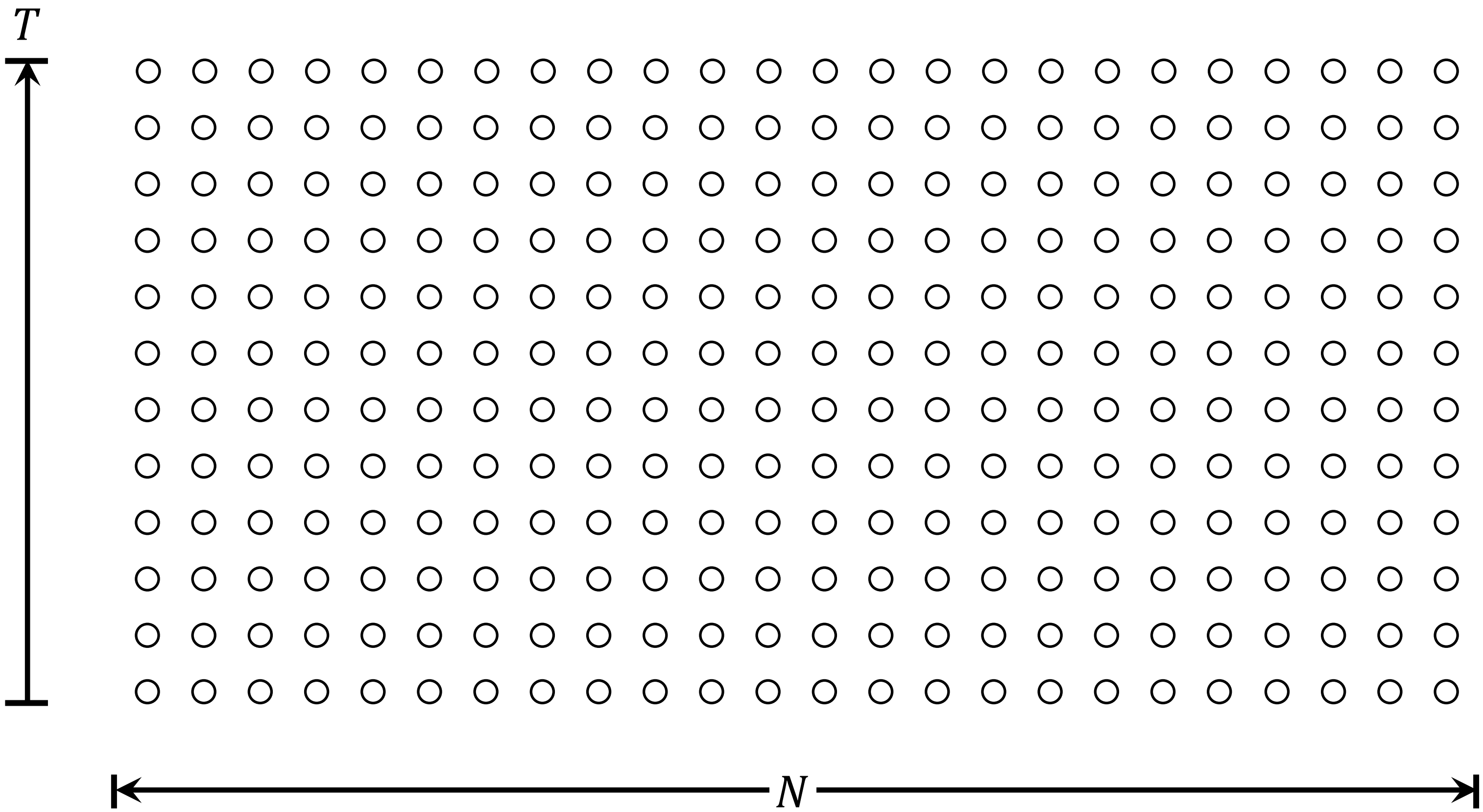
Time cut



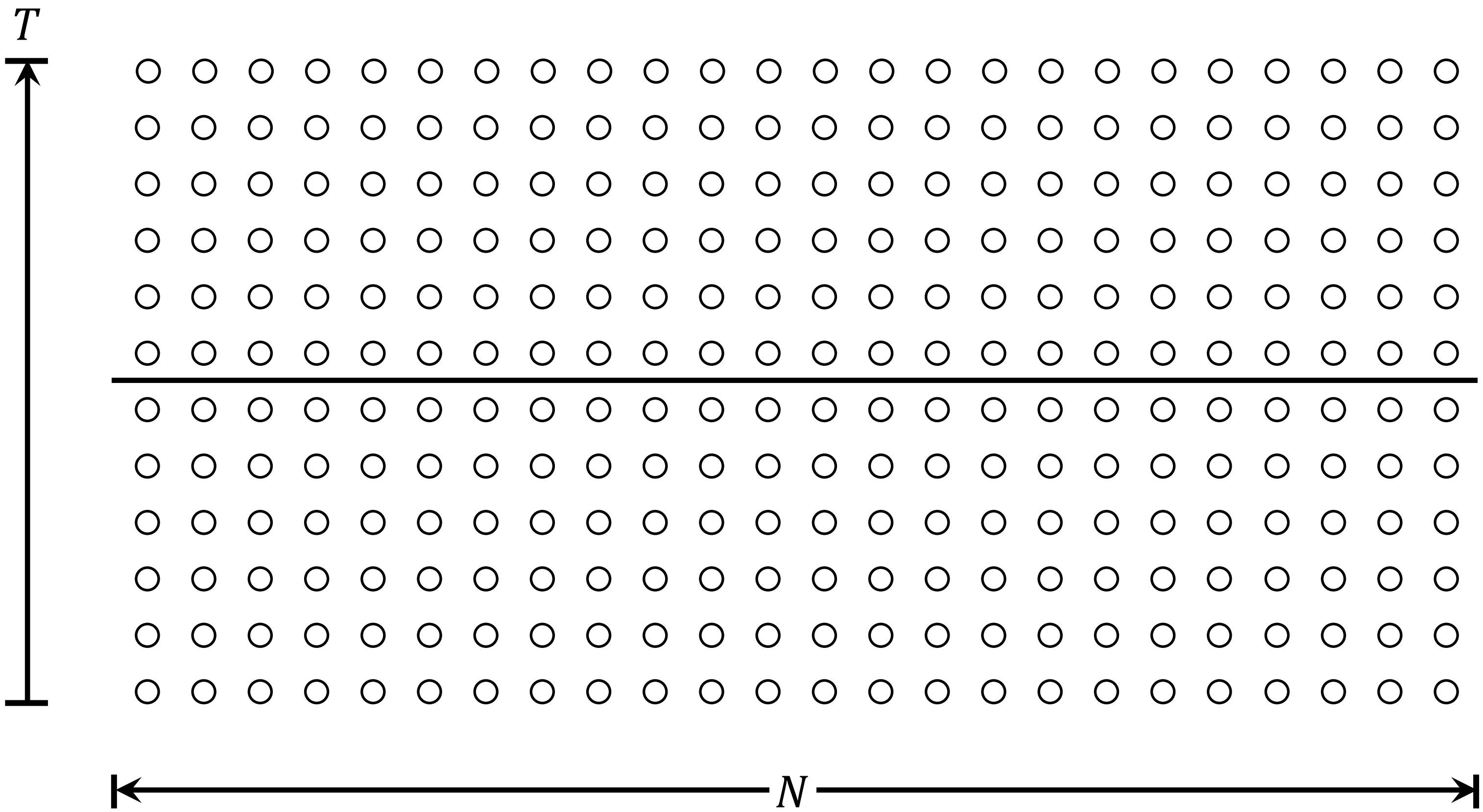
The algorithm exploits data locality cache-obliviously through recursive tiling.

Code generator: Pochoir (MIT/Fudan/SBU)

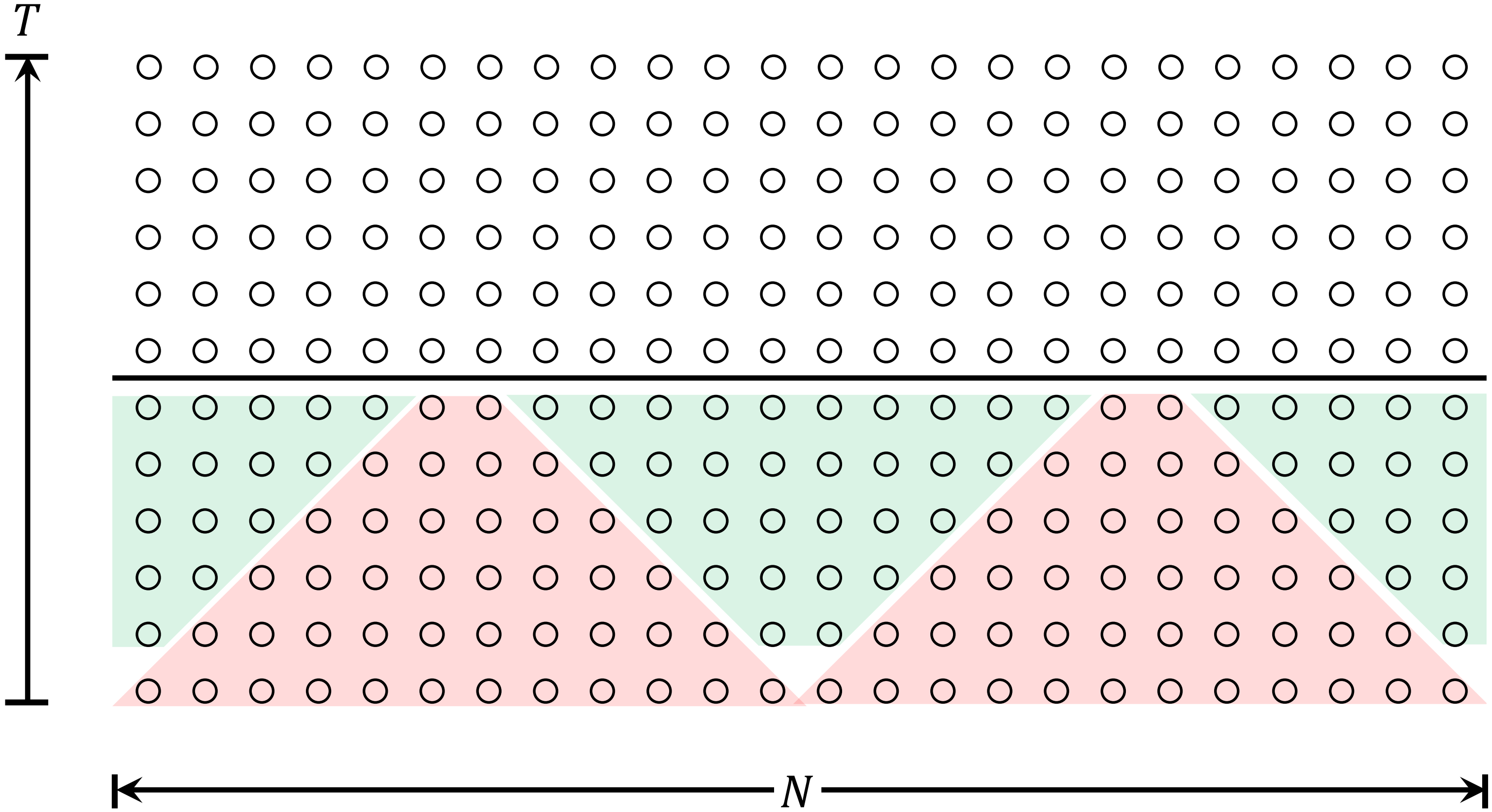
Trapezoidal Decomposition Algorithm



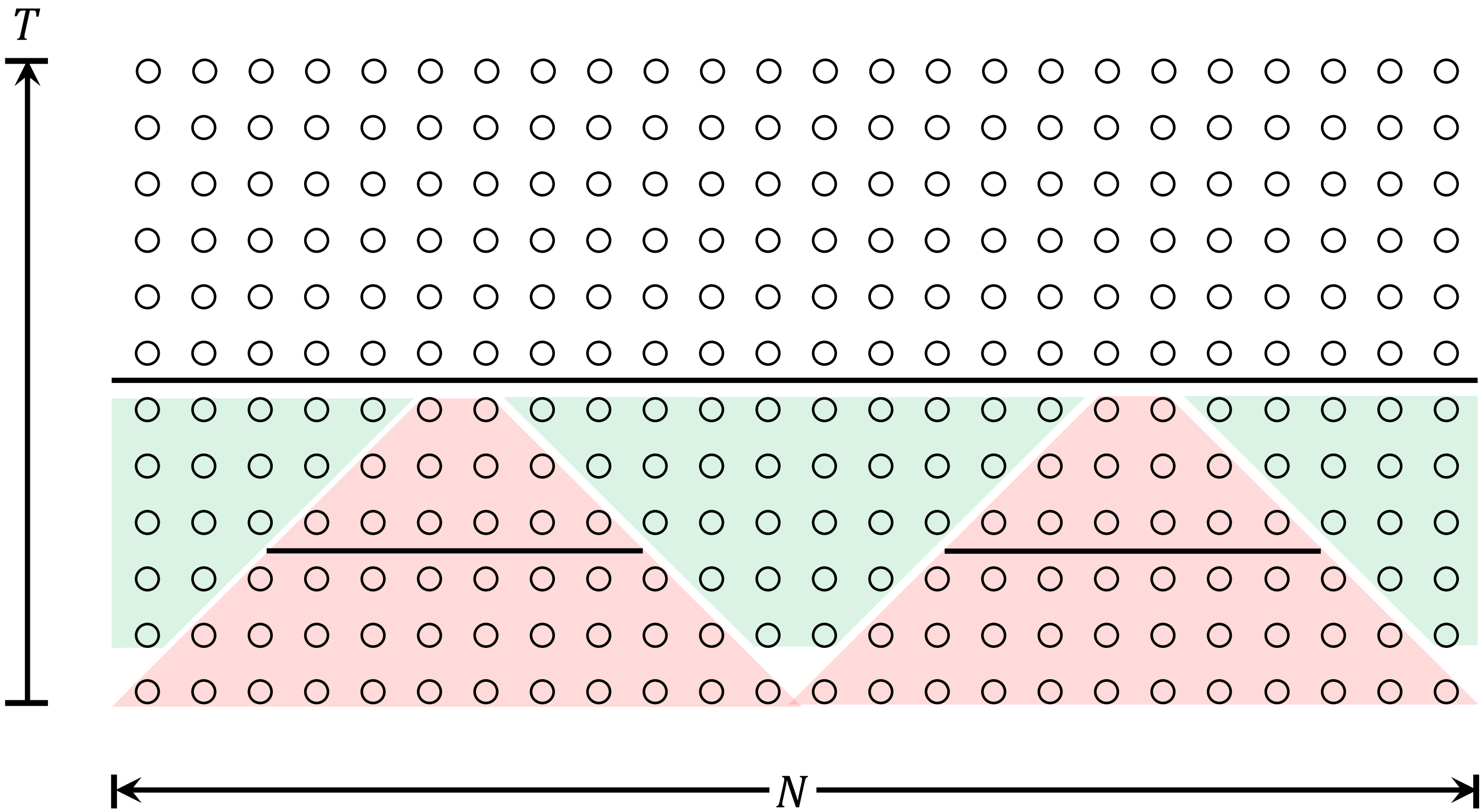
Trapezoidal Decomposition Algorithm



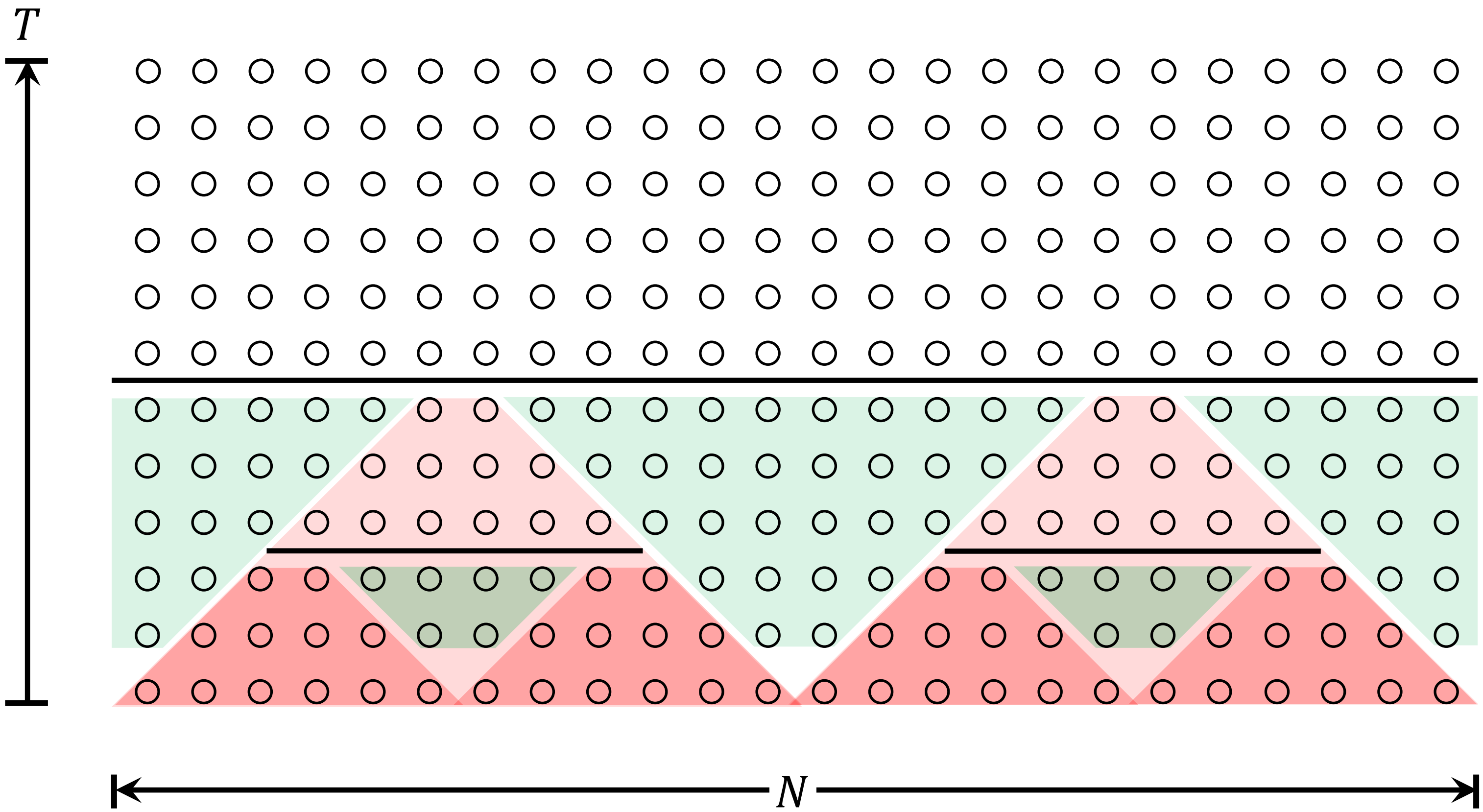
Trapezoidal Decomposition Algorithm



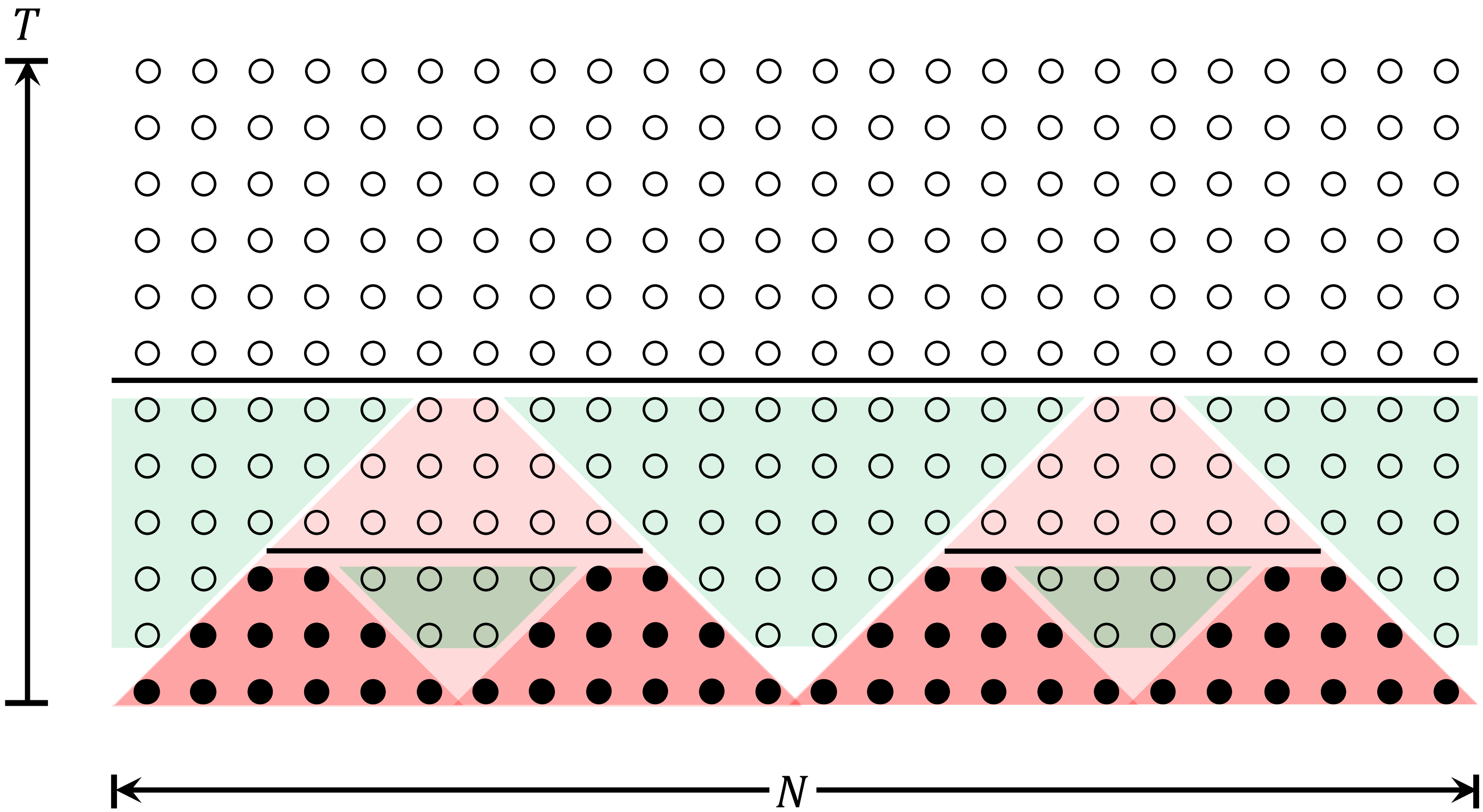
Trapezoidal Decomposition Algorithm



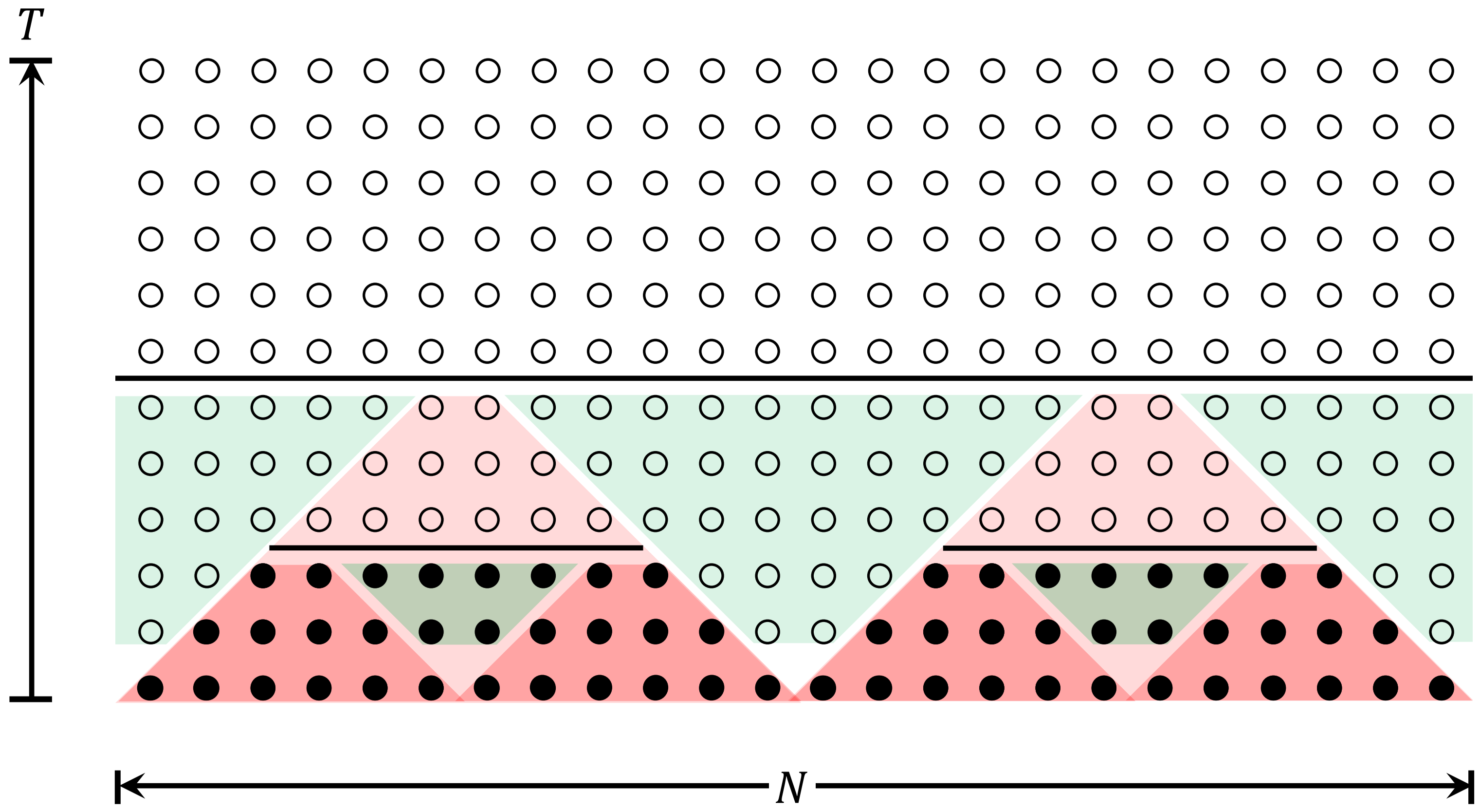
Trapezoidal Decomposition Algorithm



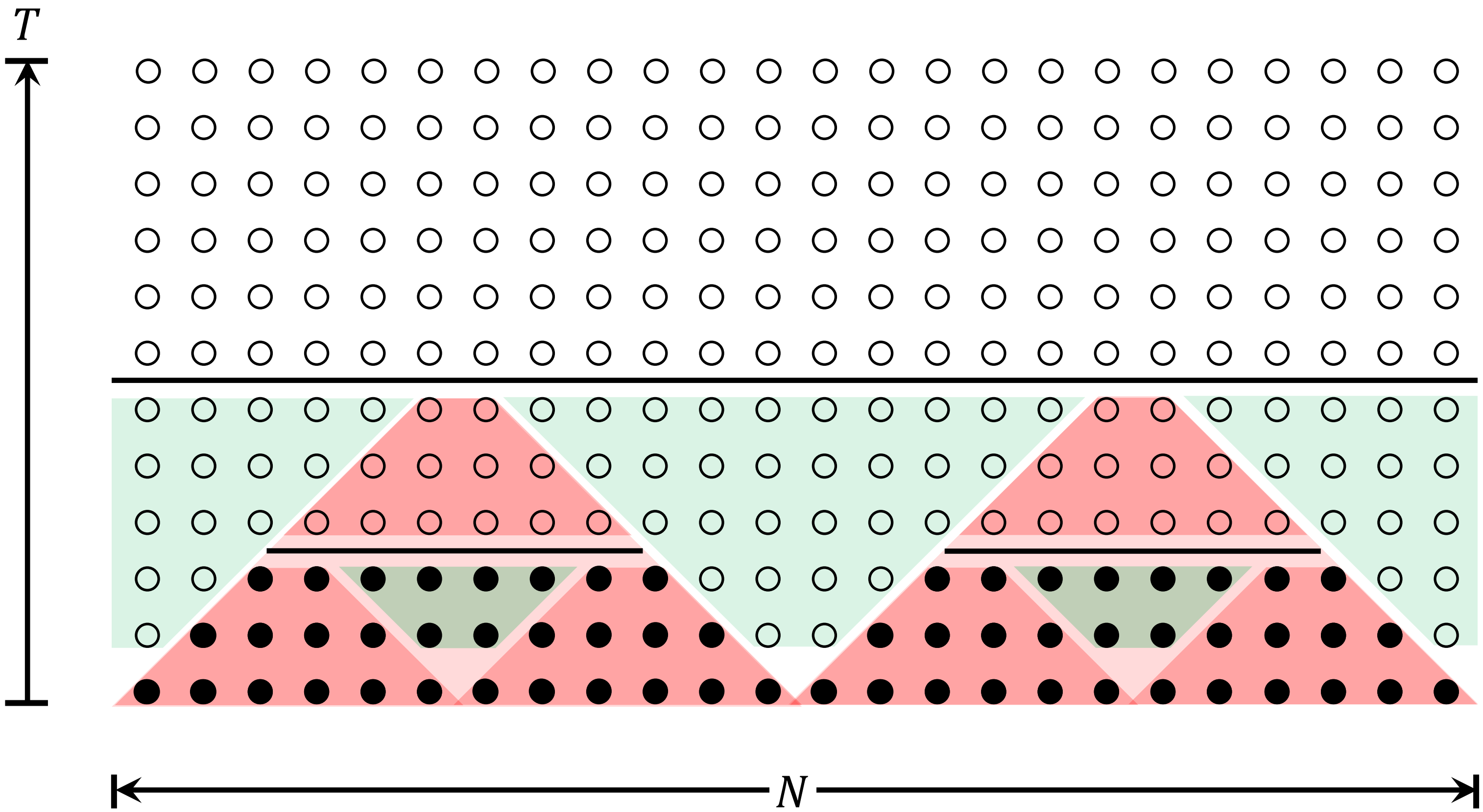
Trapezoidal Decomposition Algorithm



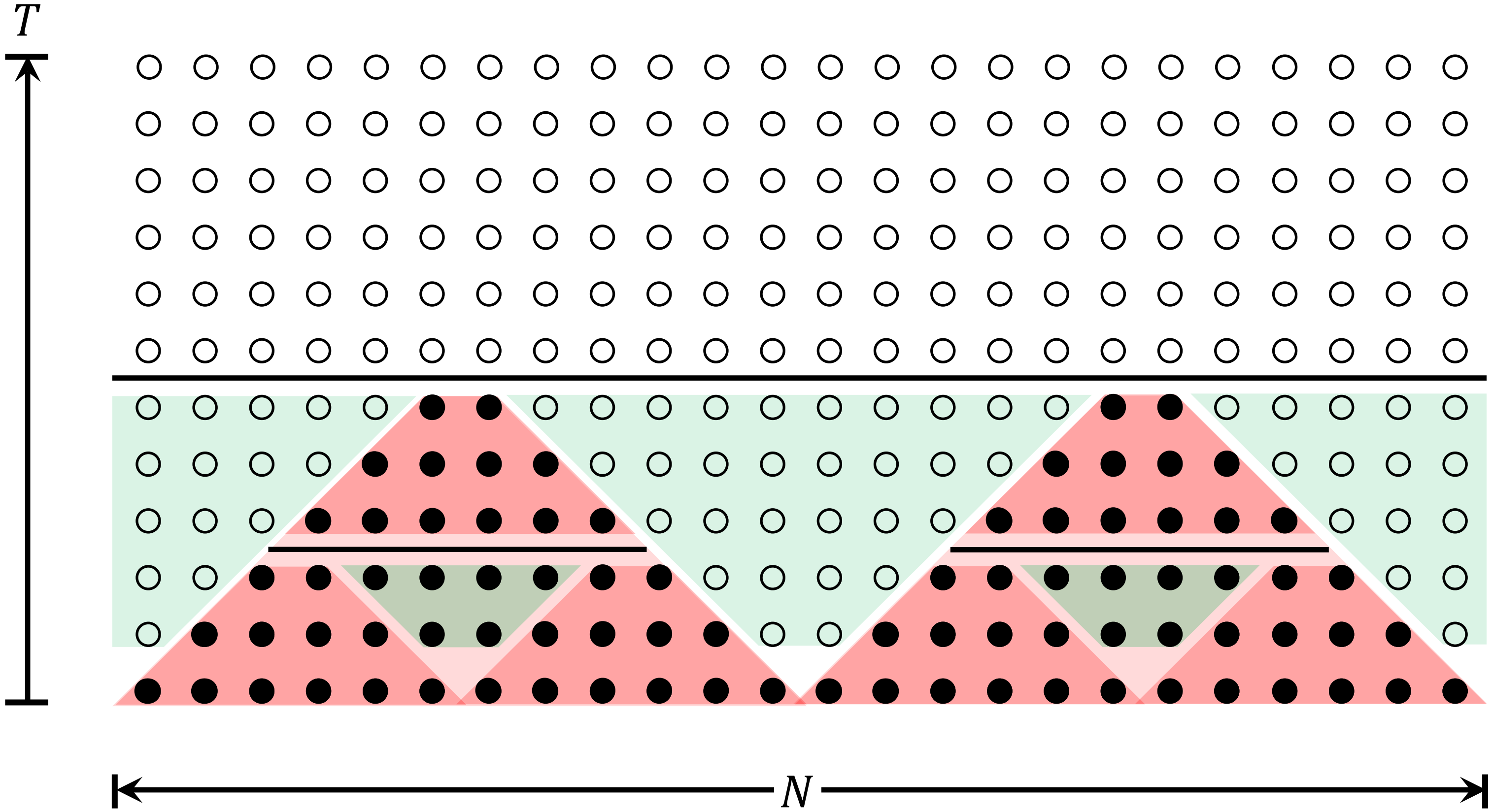
Trapezoidal Decomposition Algorithm



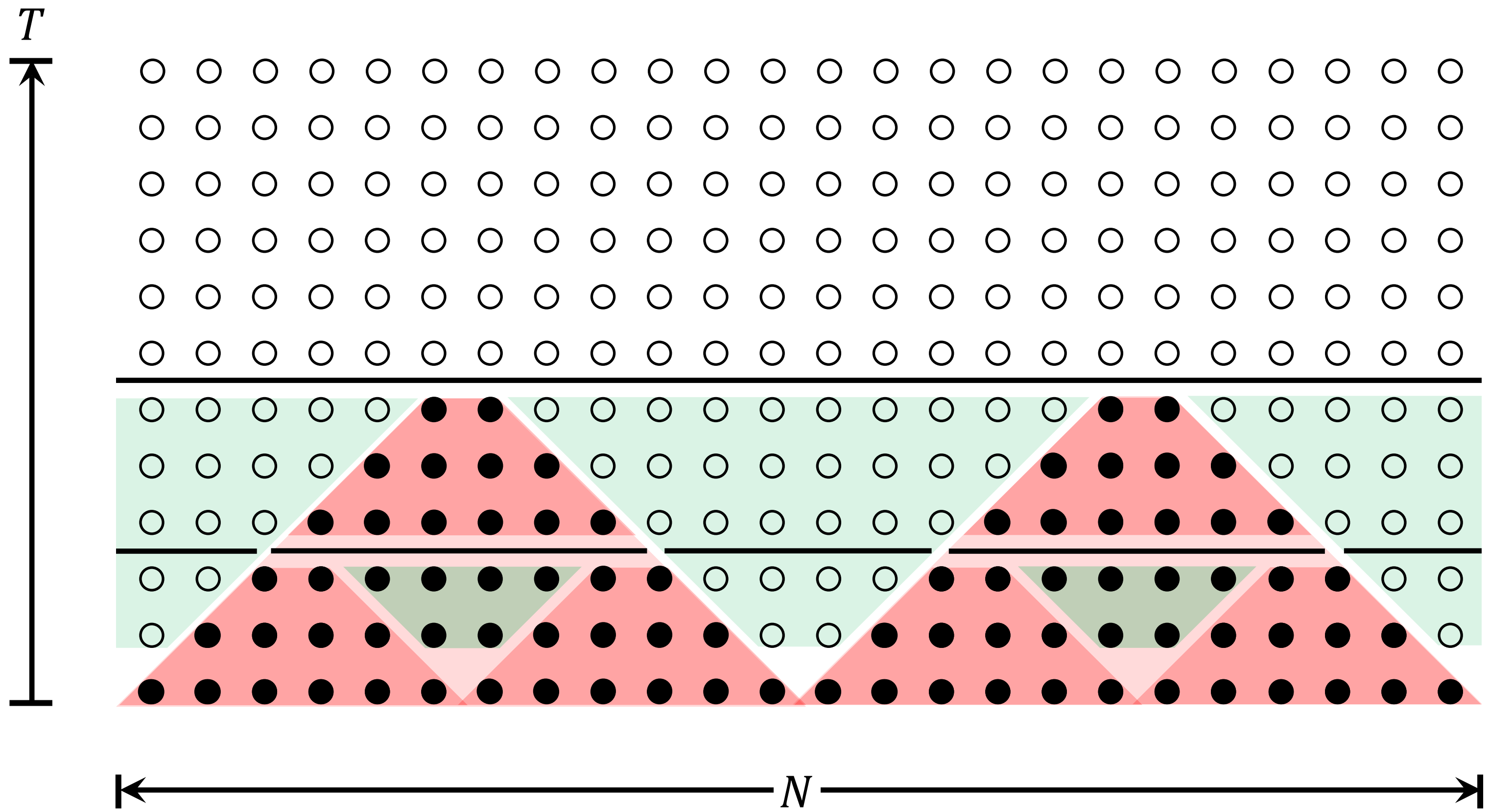
Trapezoidal Decomposition Algorithm



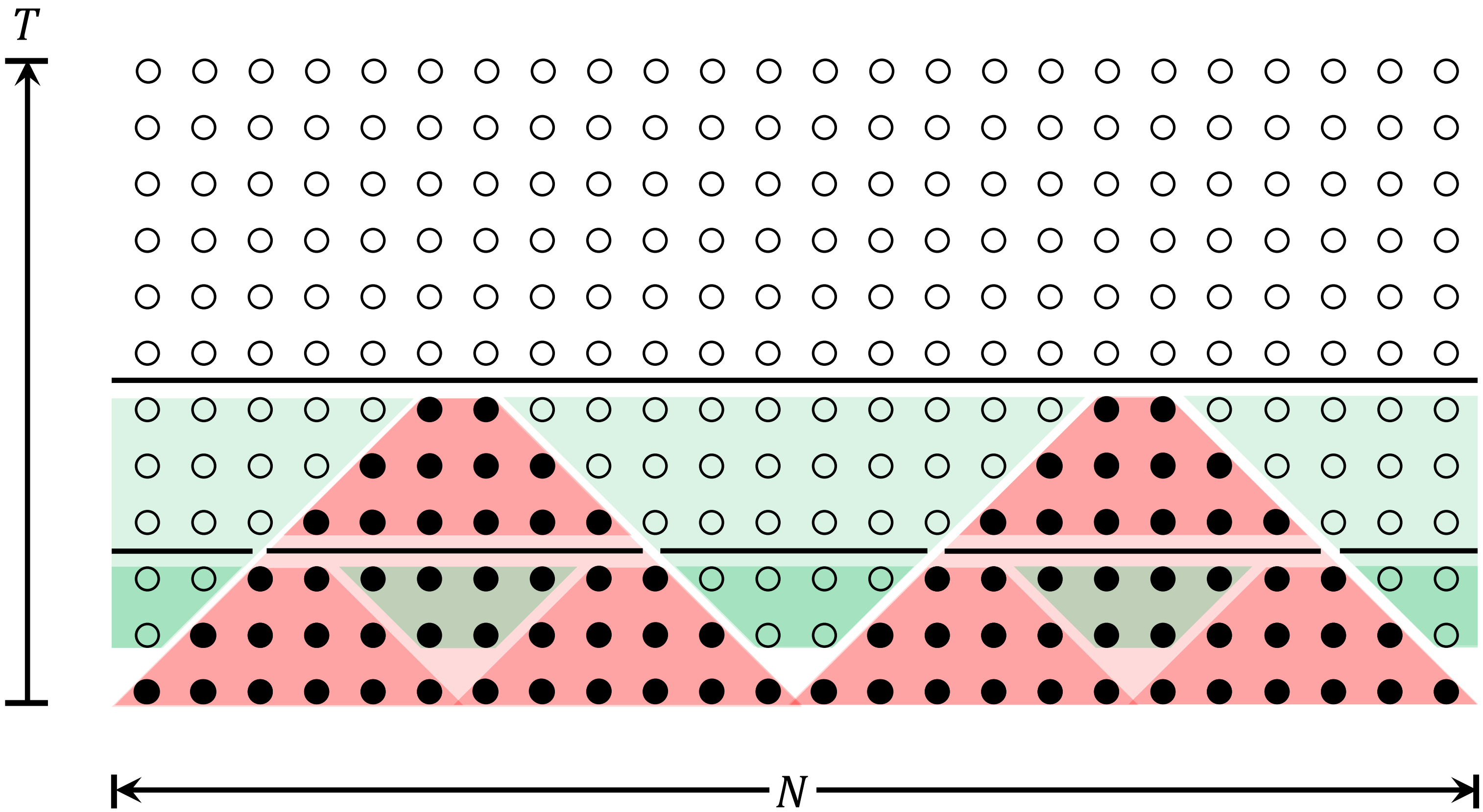
Trapezoidal Decomposition Algorithm



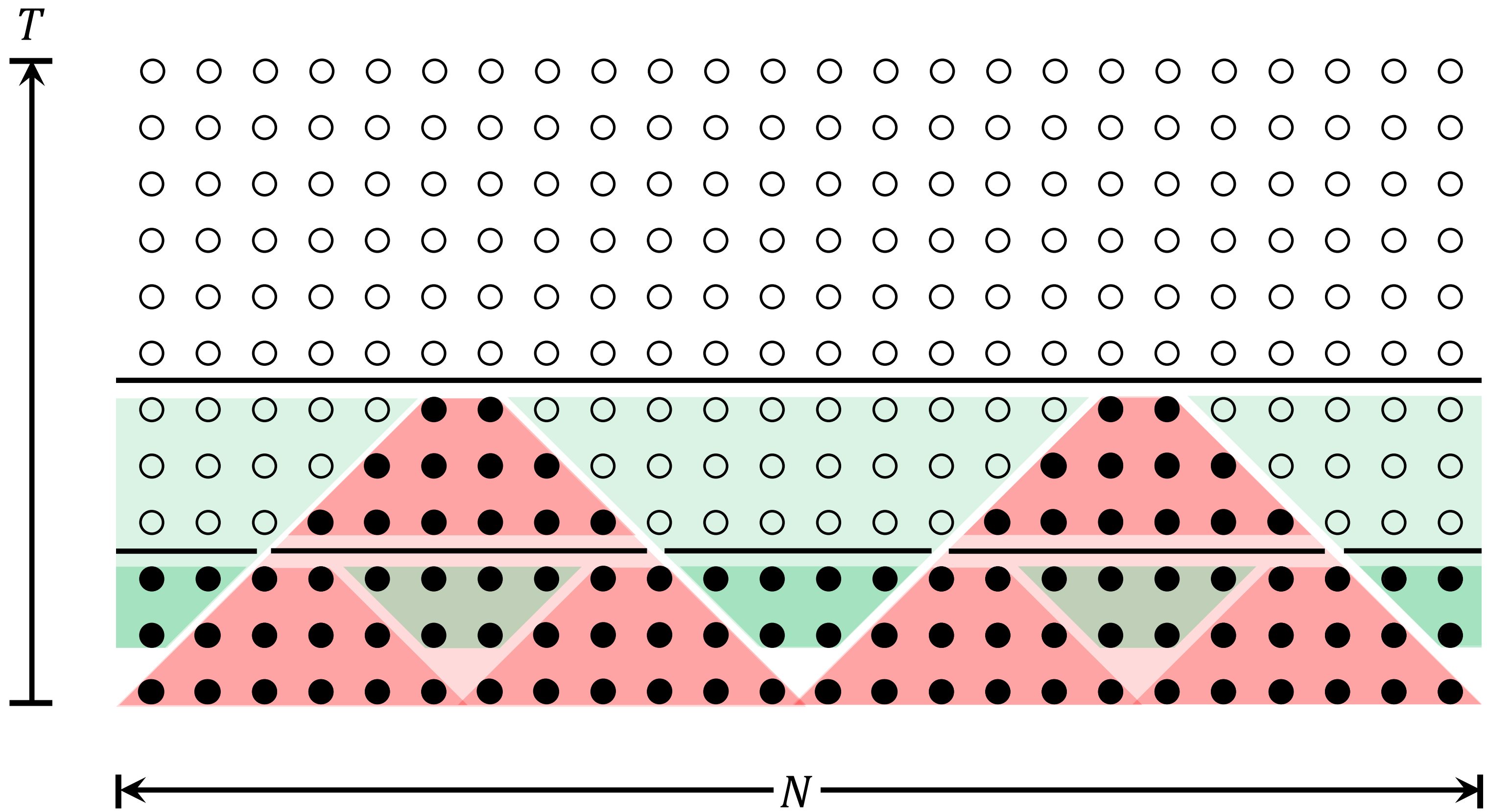
Trapezoidal Decomposition Algorithm



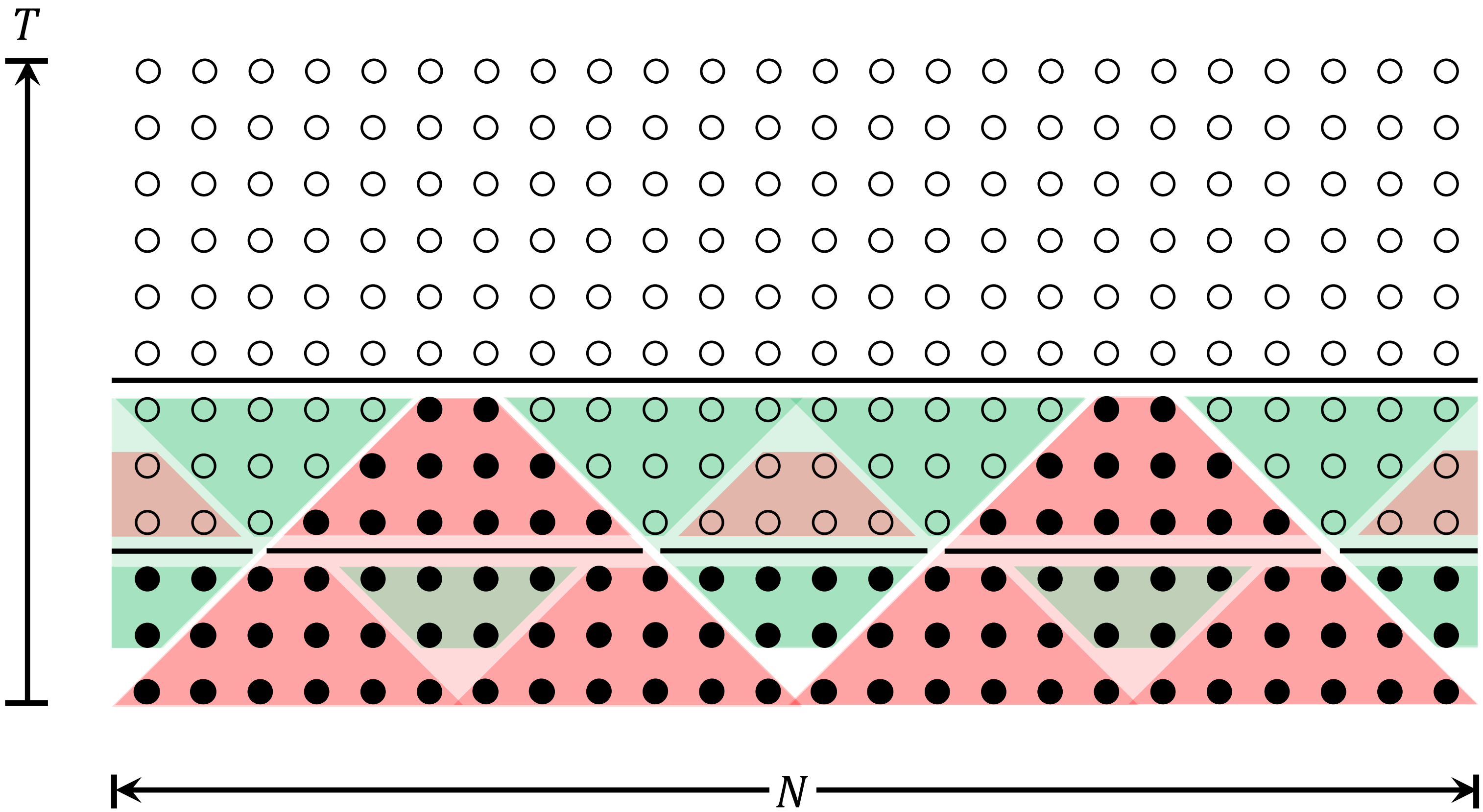
Trapezoidal Decomposition Algorithm



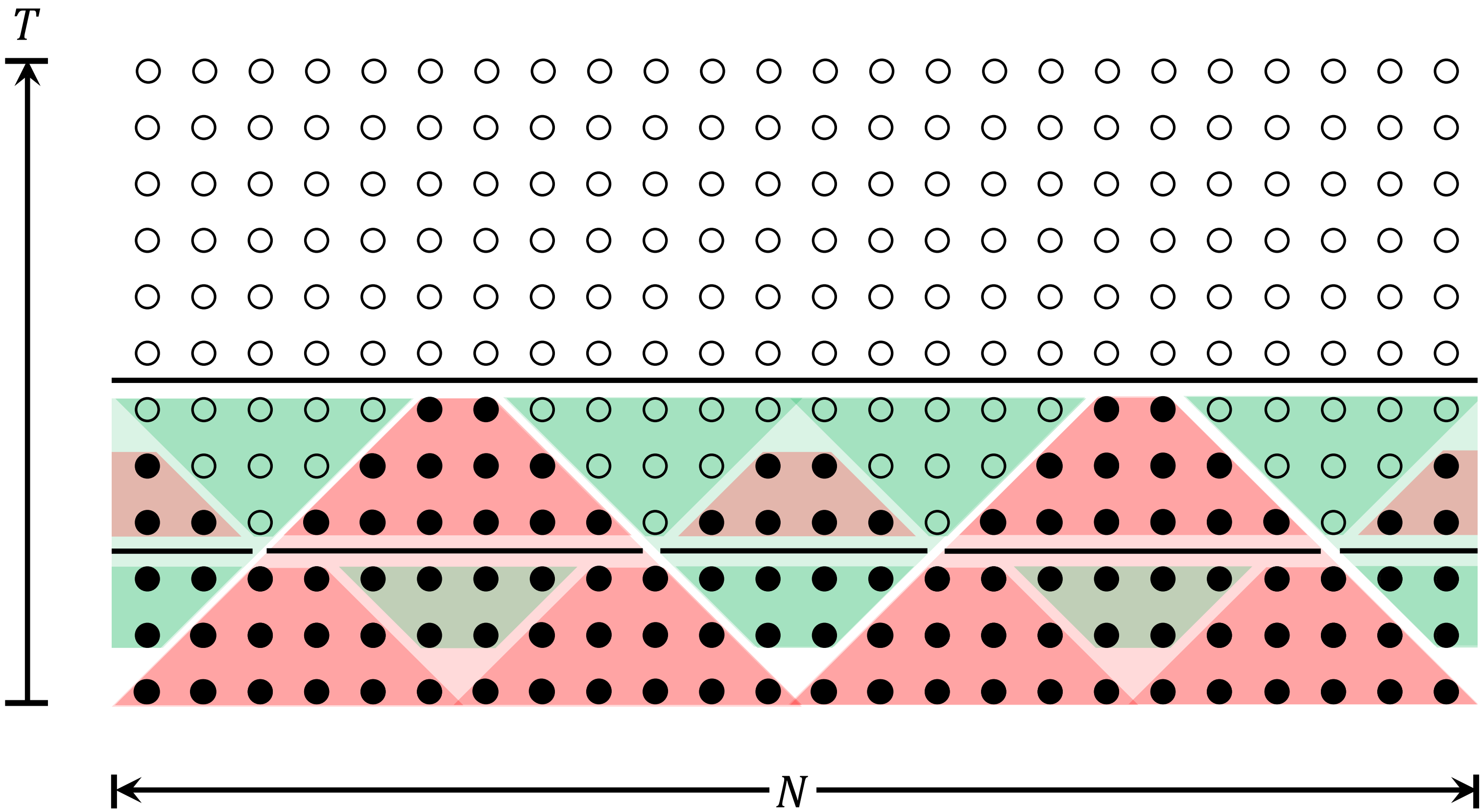
Trapezoidal Decomposition Algorithm



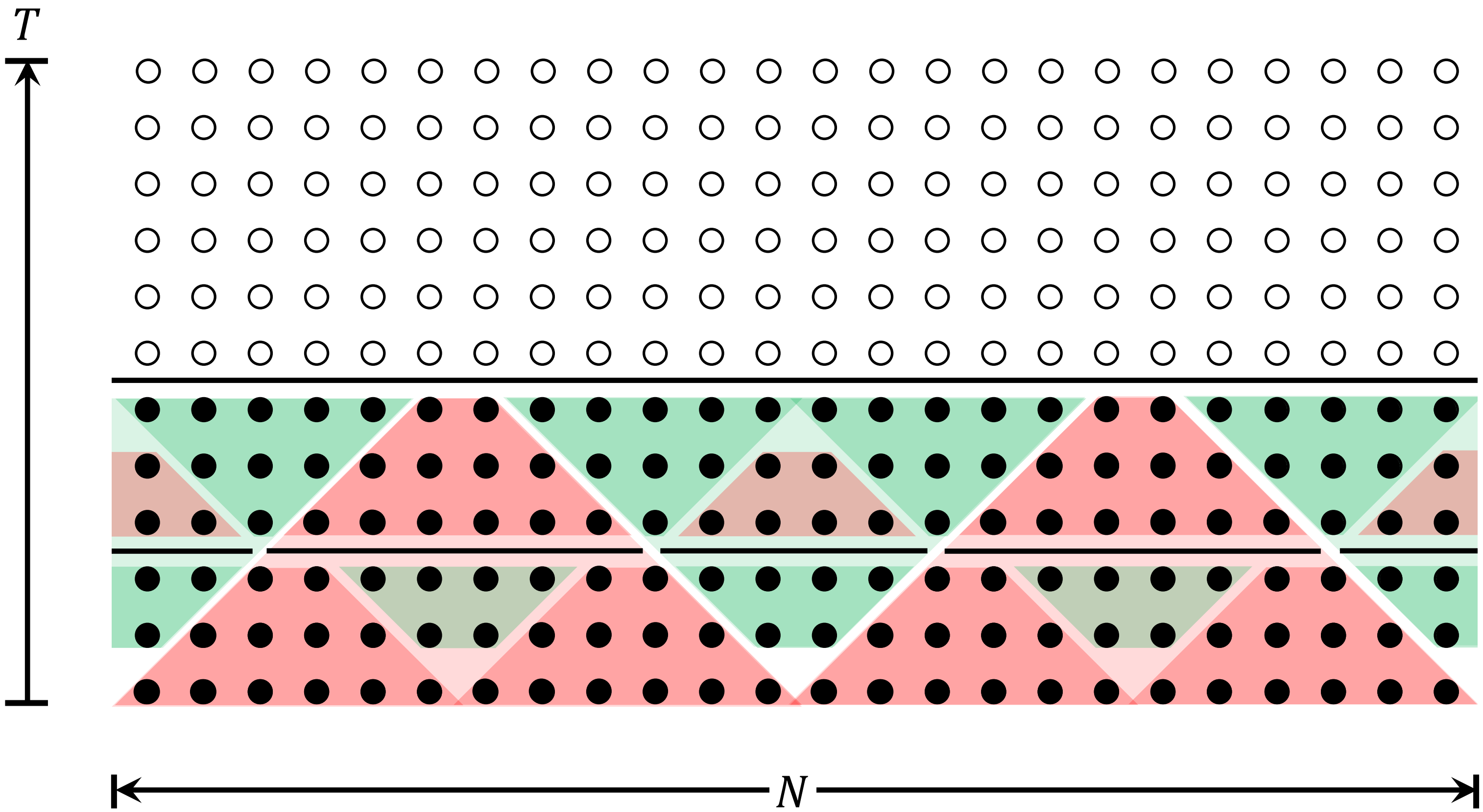
Trapezoidal Decomposition Algorithm



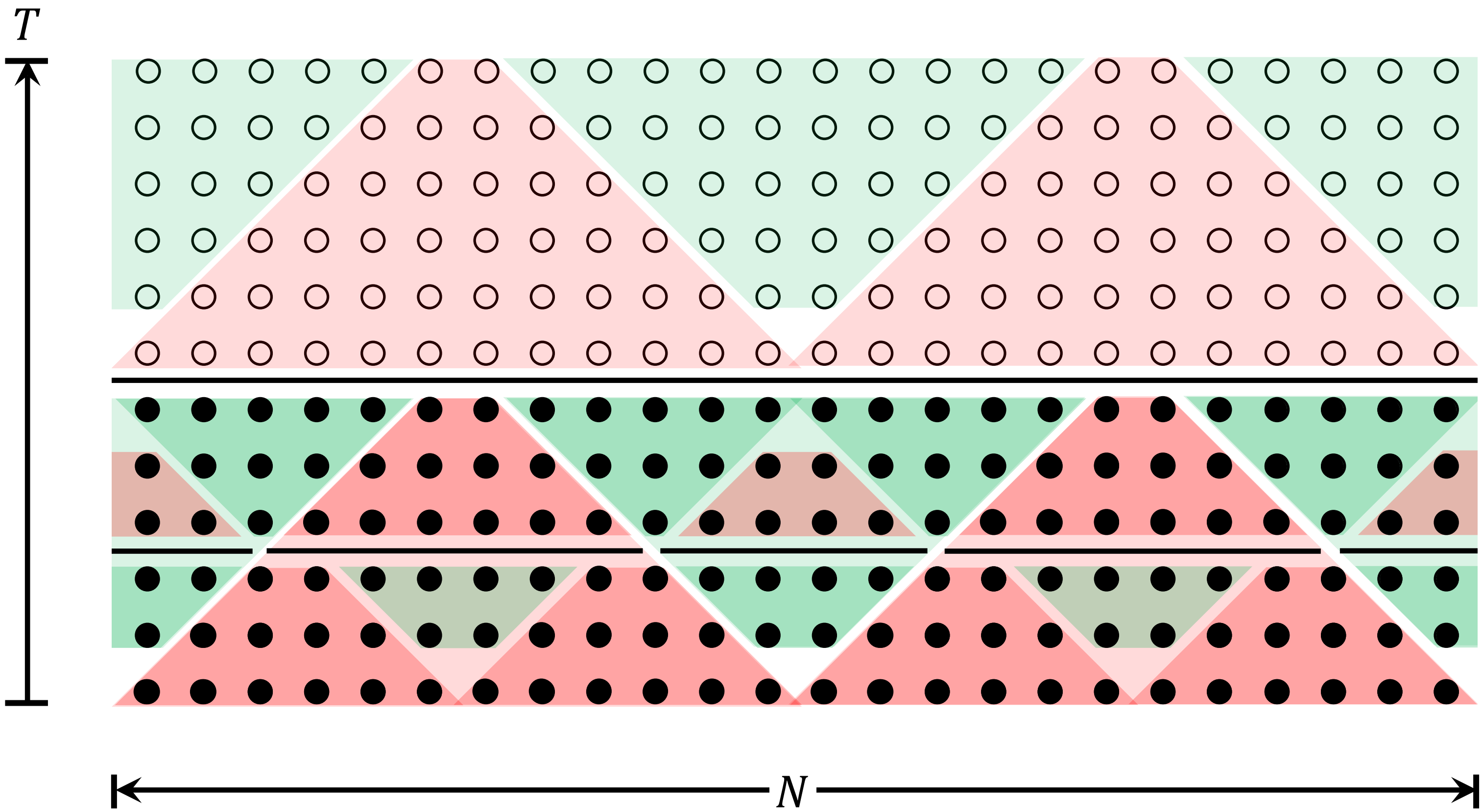
Trapezoidal Decomposition Algorithm



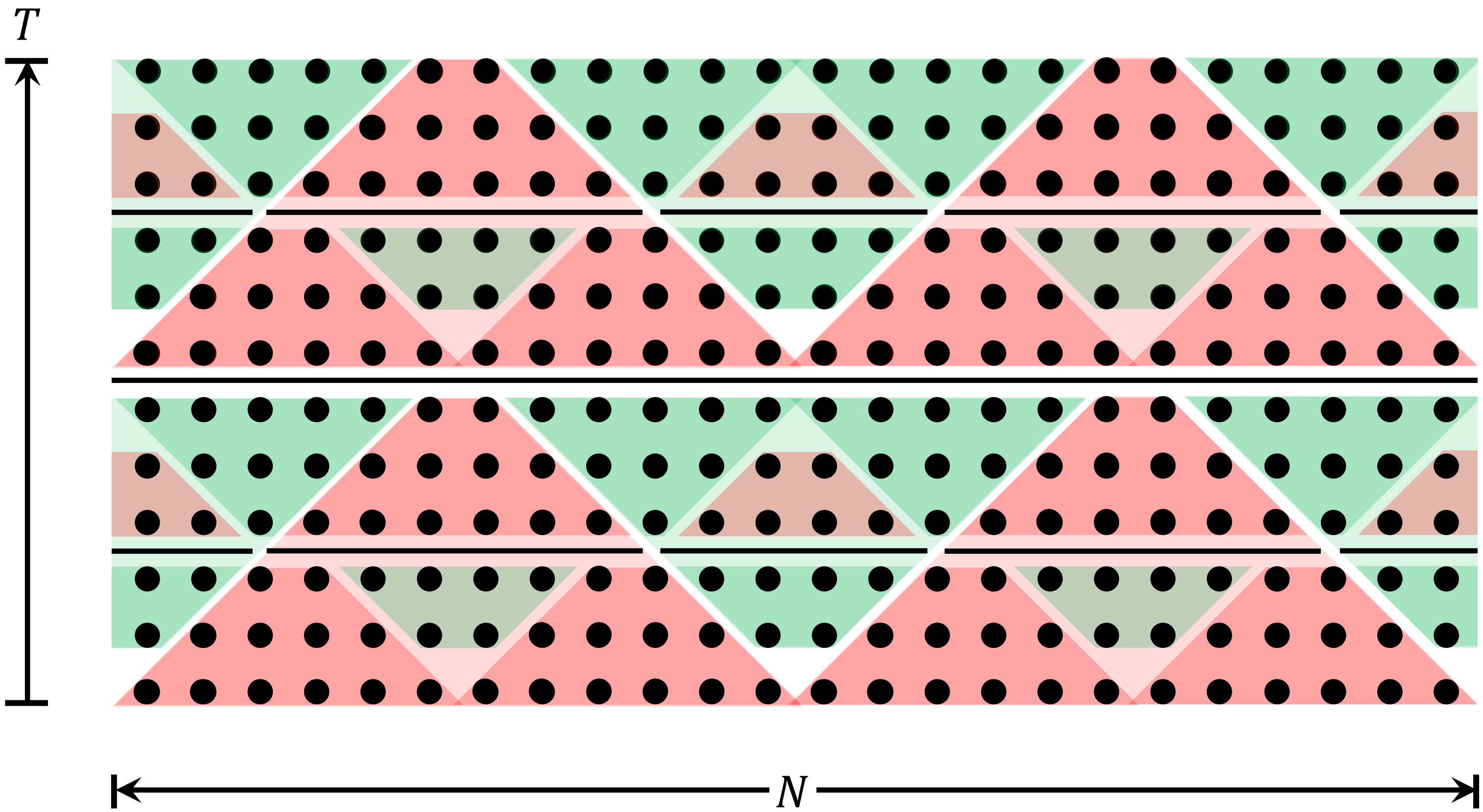
Trapezoidal Decomposition Algorithm



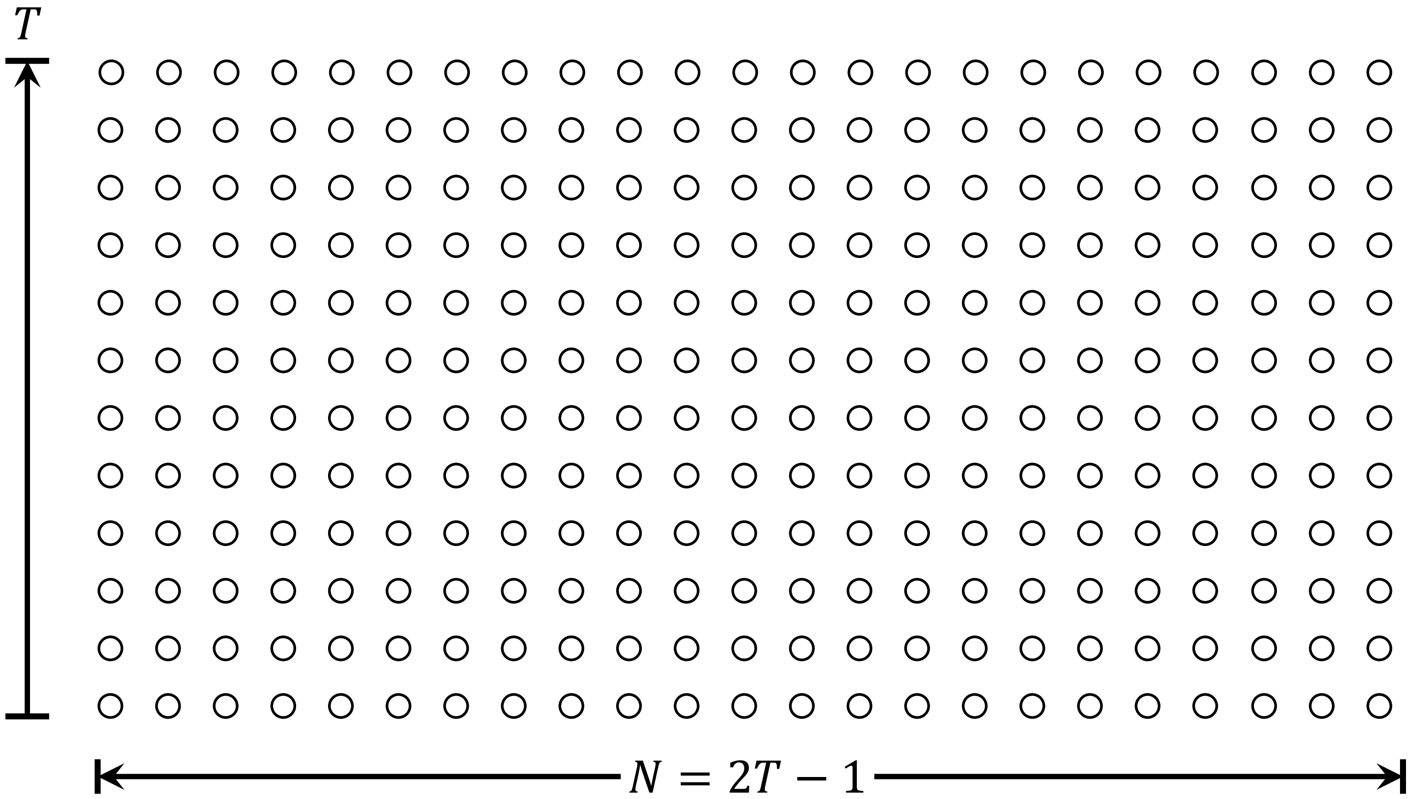
Trapezoidal Decomposition Algorithm



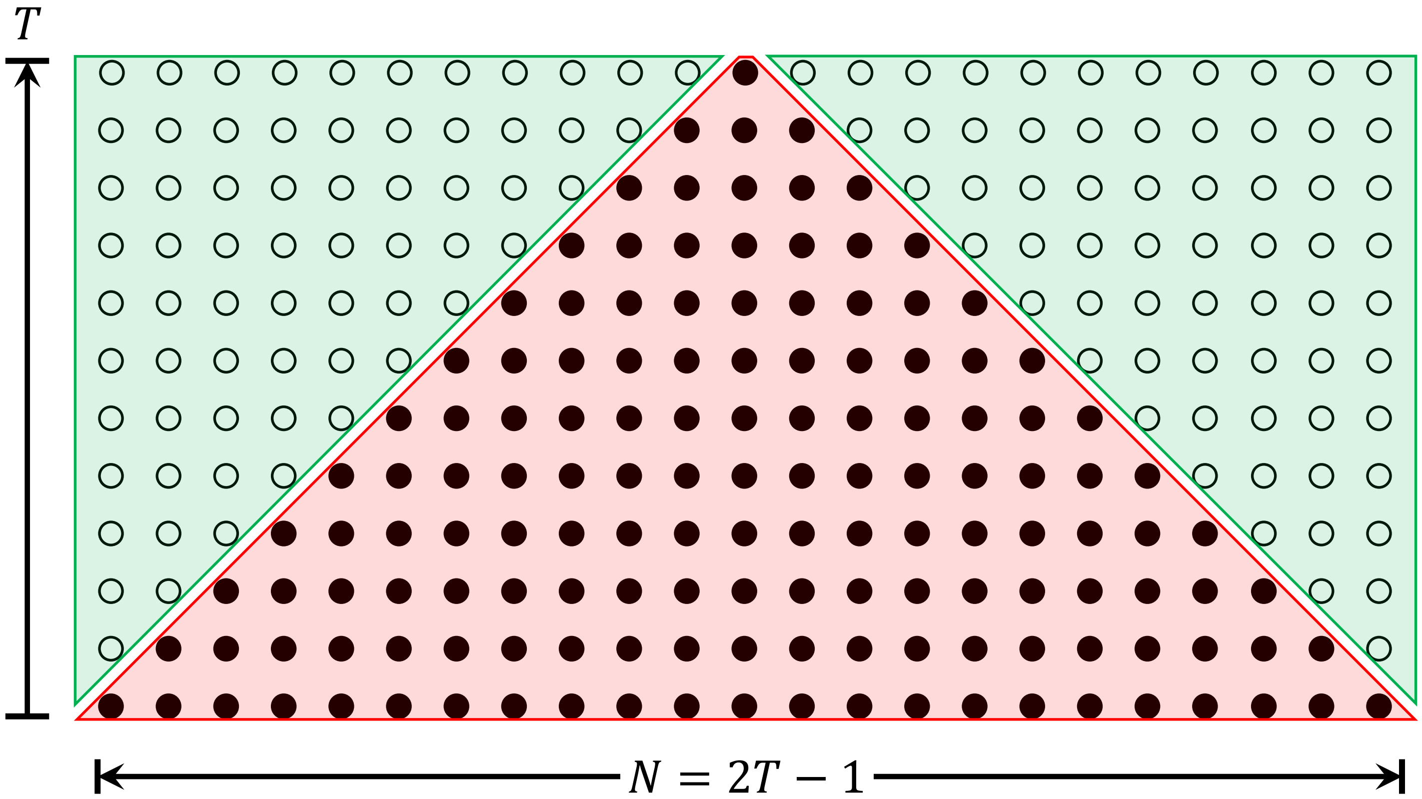
Trapezoidal Decomposition Algorithm



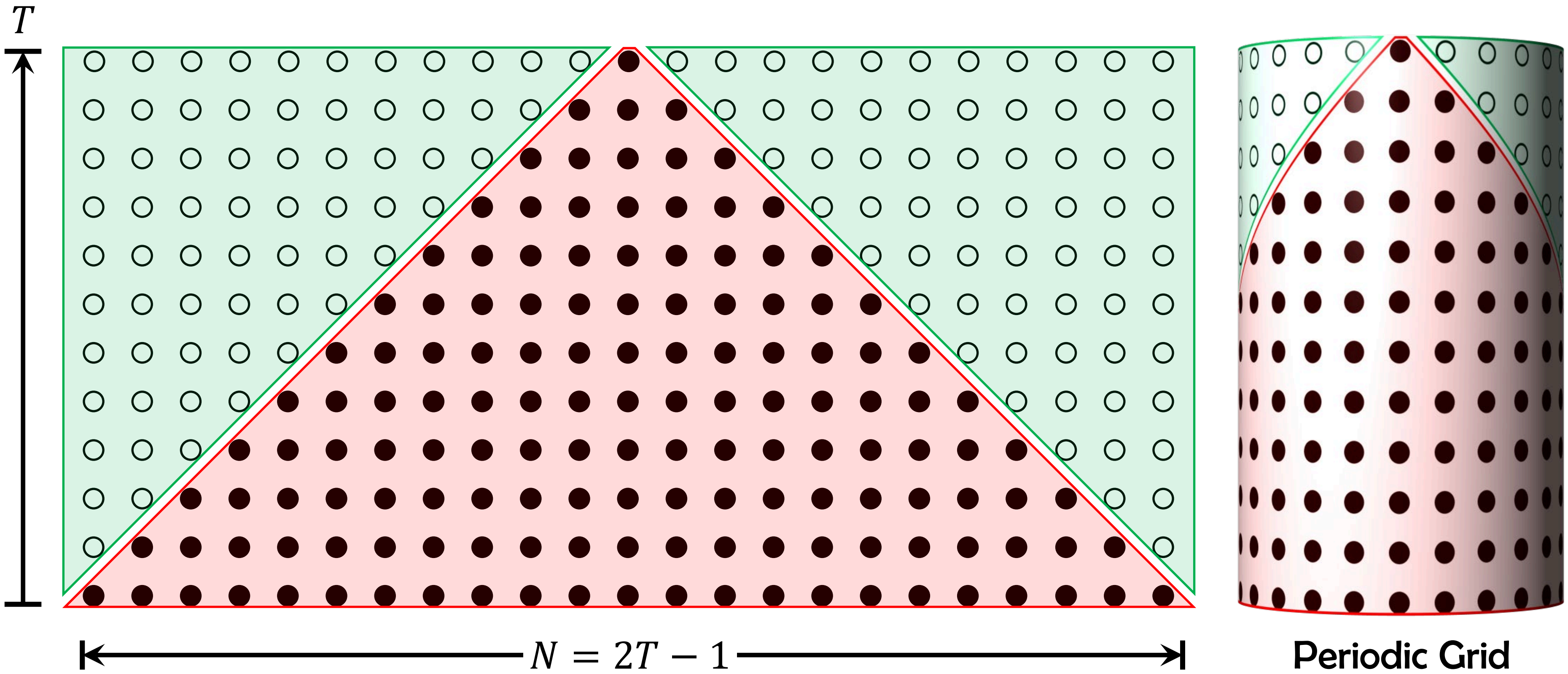
Trapezoidal Decomposition Algorithm



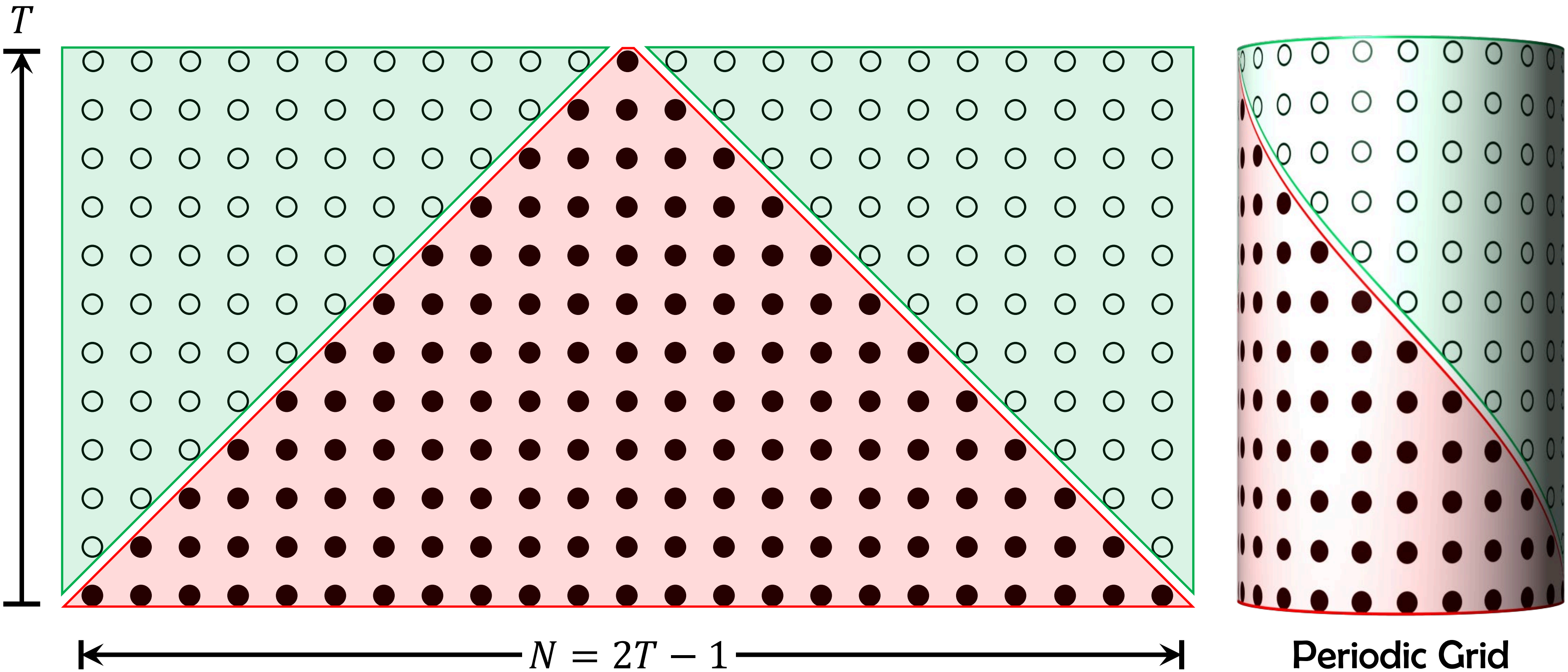
Trapezoidal Decomposition Algorithm



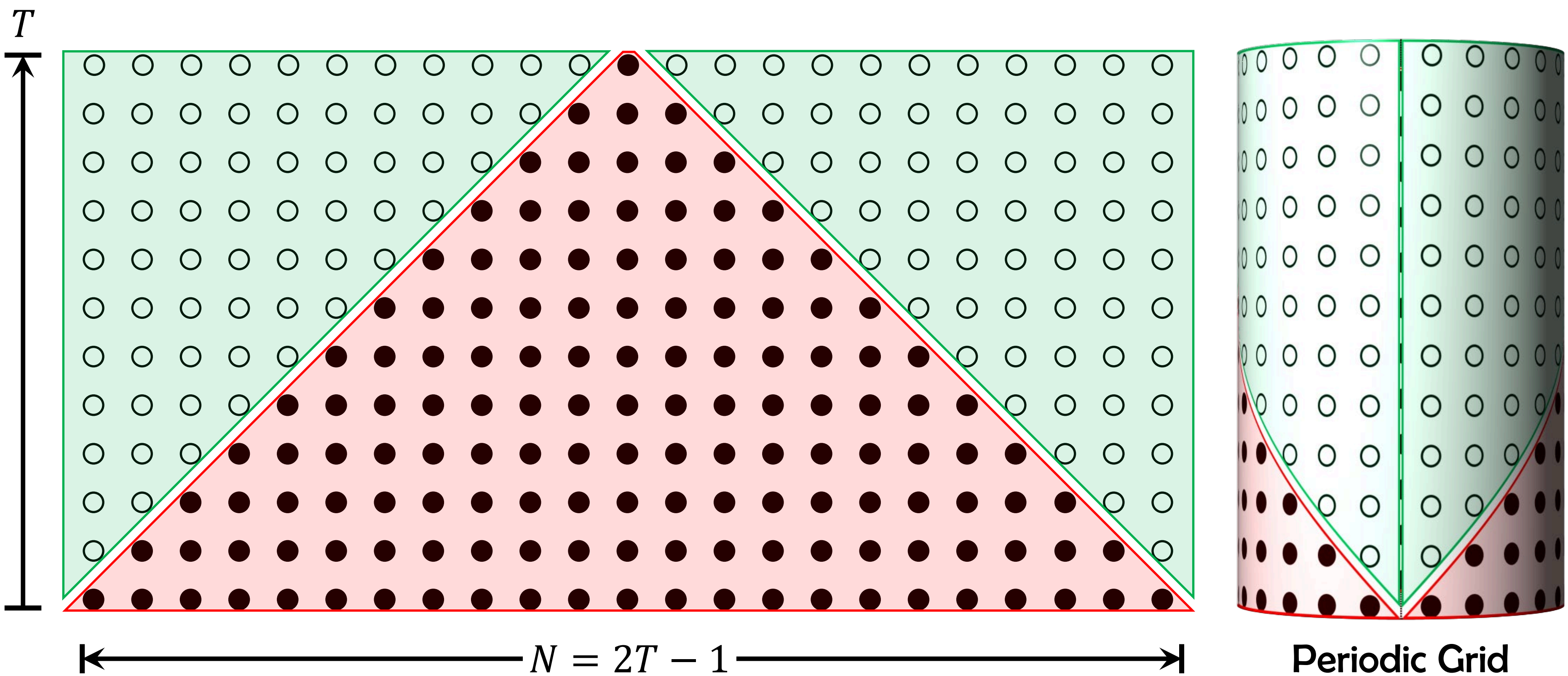
Trapezoidal Decomposition Algorithm



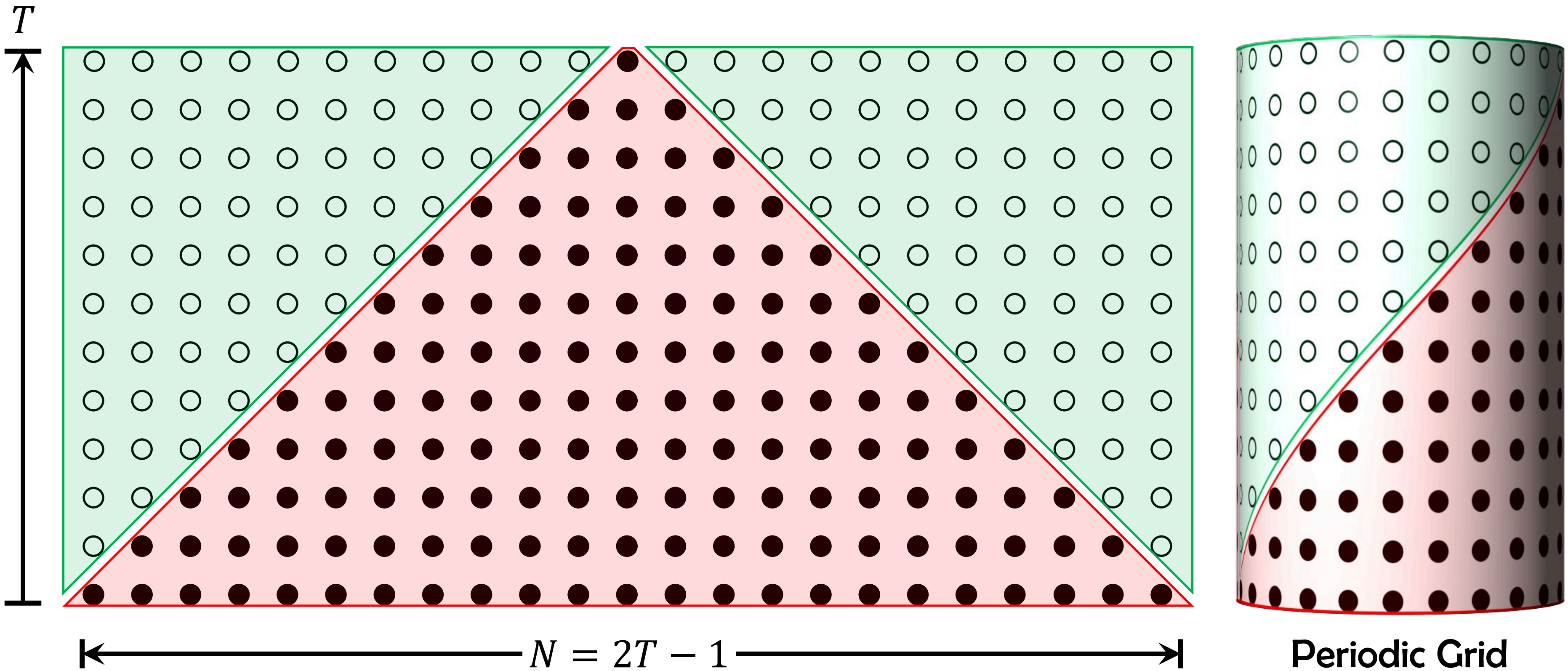
Trapezoidal Decomposition Algorithm



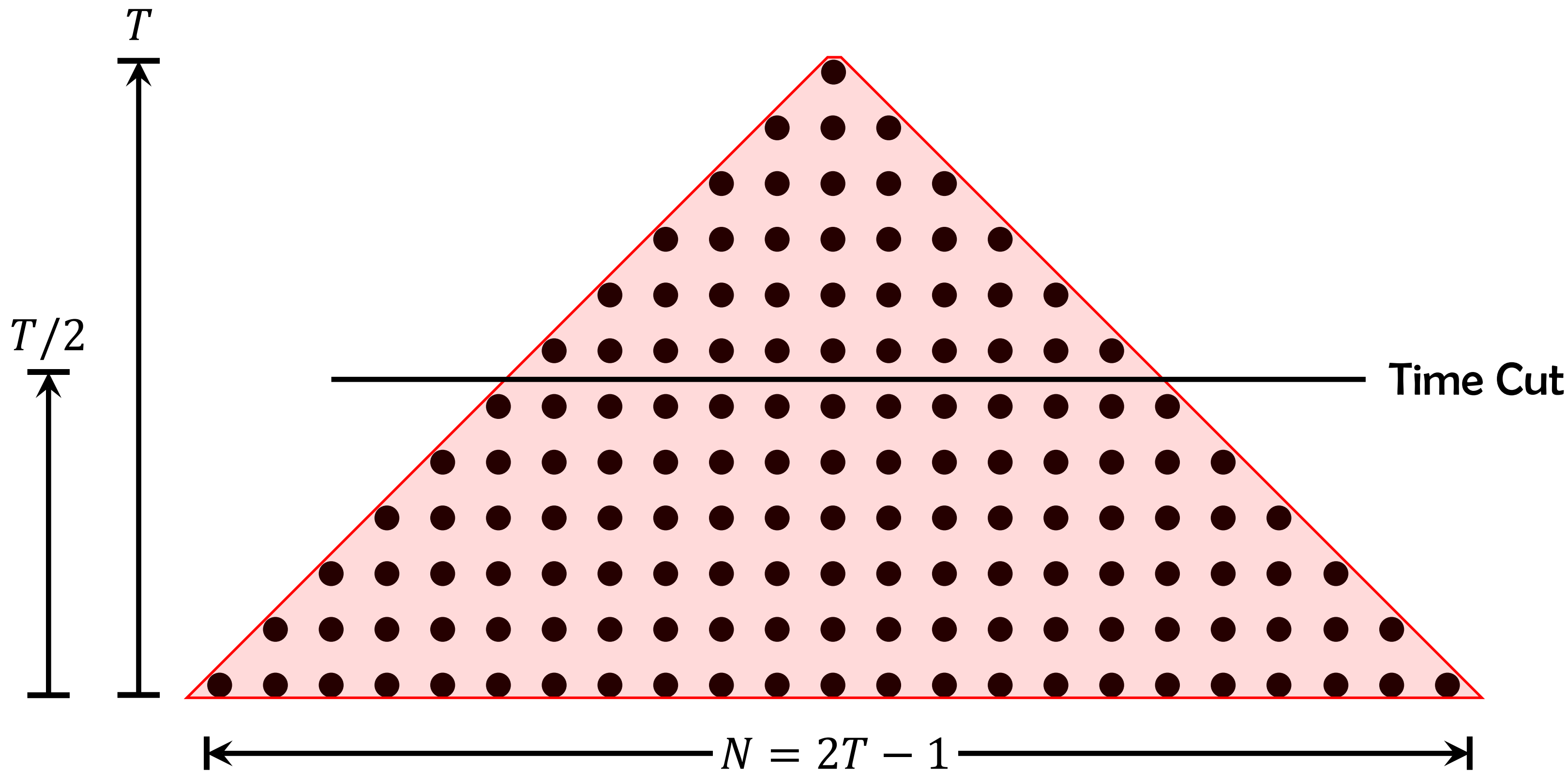
Trapezoidal Decomposition Algorithm



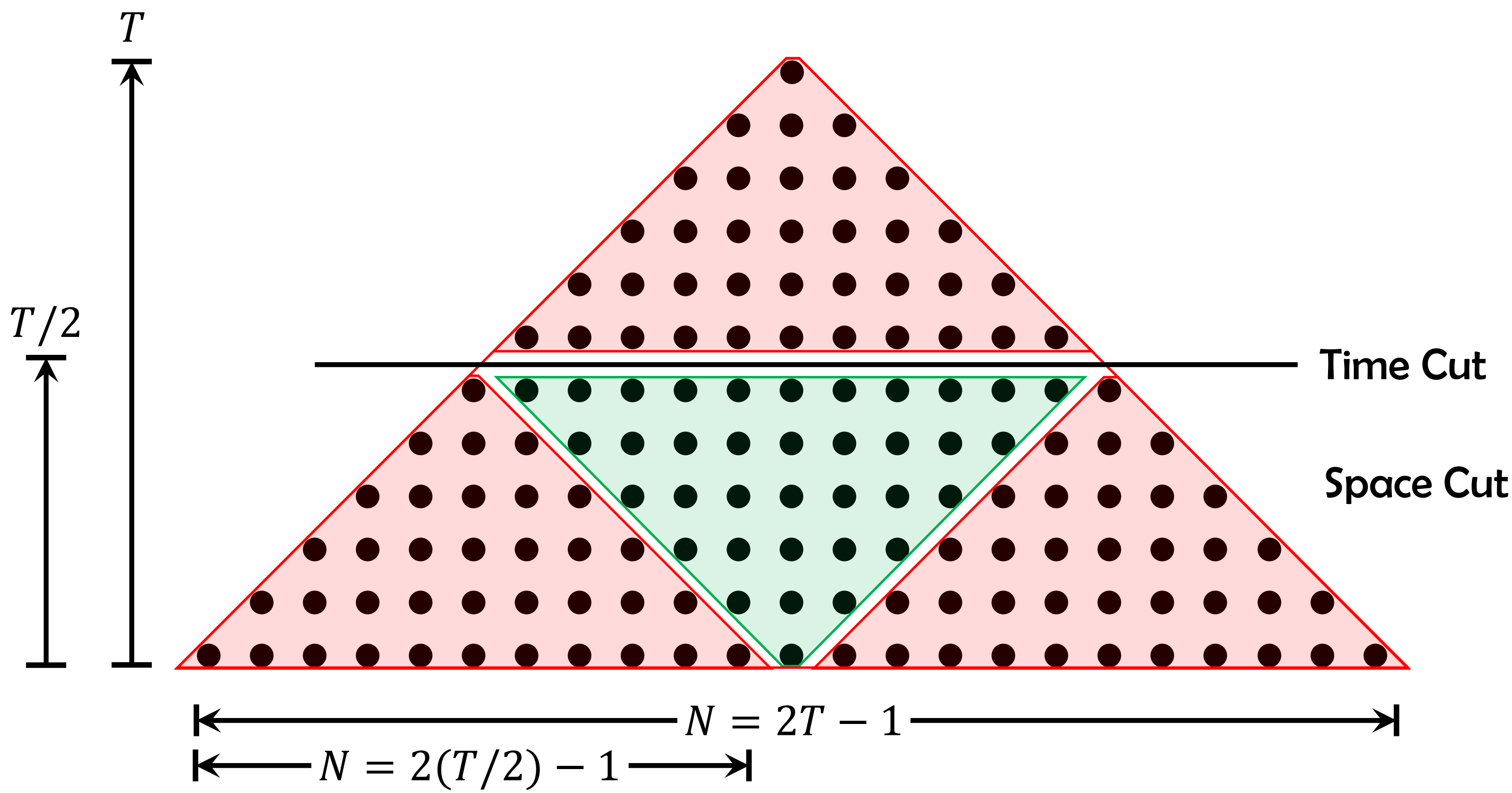
Trapezoidal Decomposition Algorithm



Trapezoidal Decomposition Algorithm



Trapezoidal Decomposition Algorithm

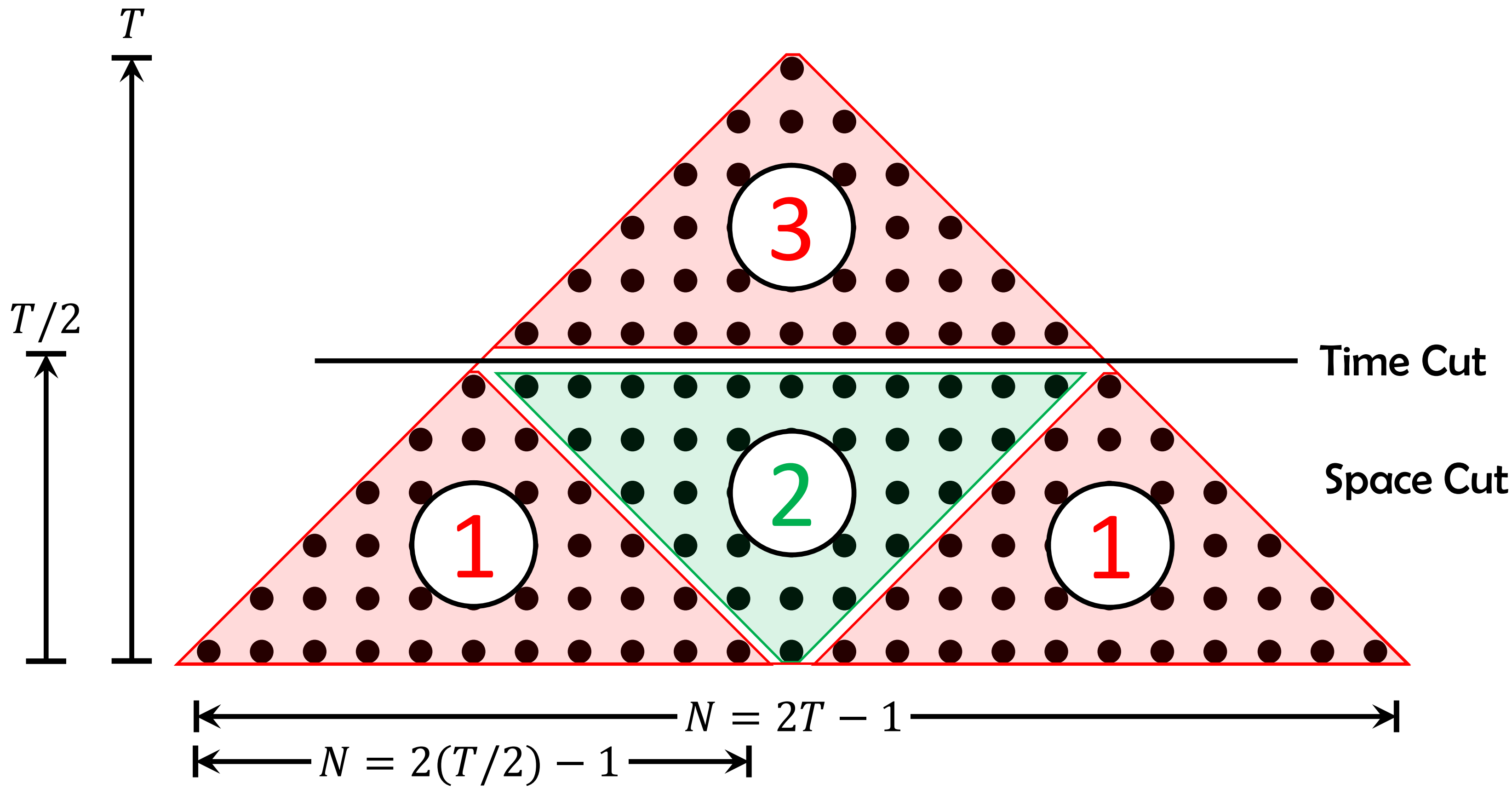


Work

$$T_1(T) = \begin{cases} \Theta(1), & if\ T \le c \\ 4T_1\left(\frac{T}{2}\right) + \Theta(1), & otherwise \end{cases}$$

$$= \Theta\left(T^2\right) = \Theta(NT)$$

Trapezoidal Decomposition Algorithm

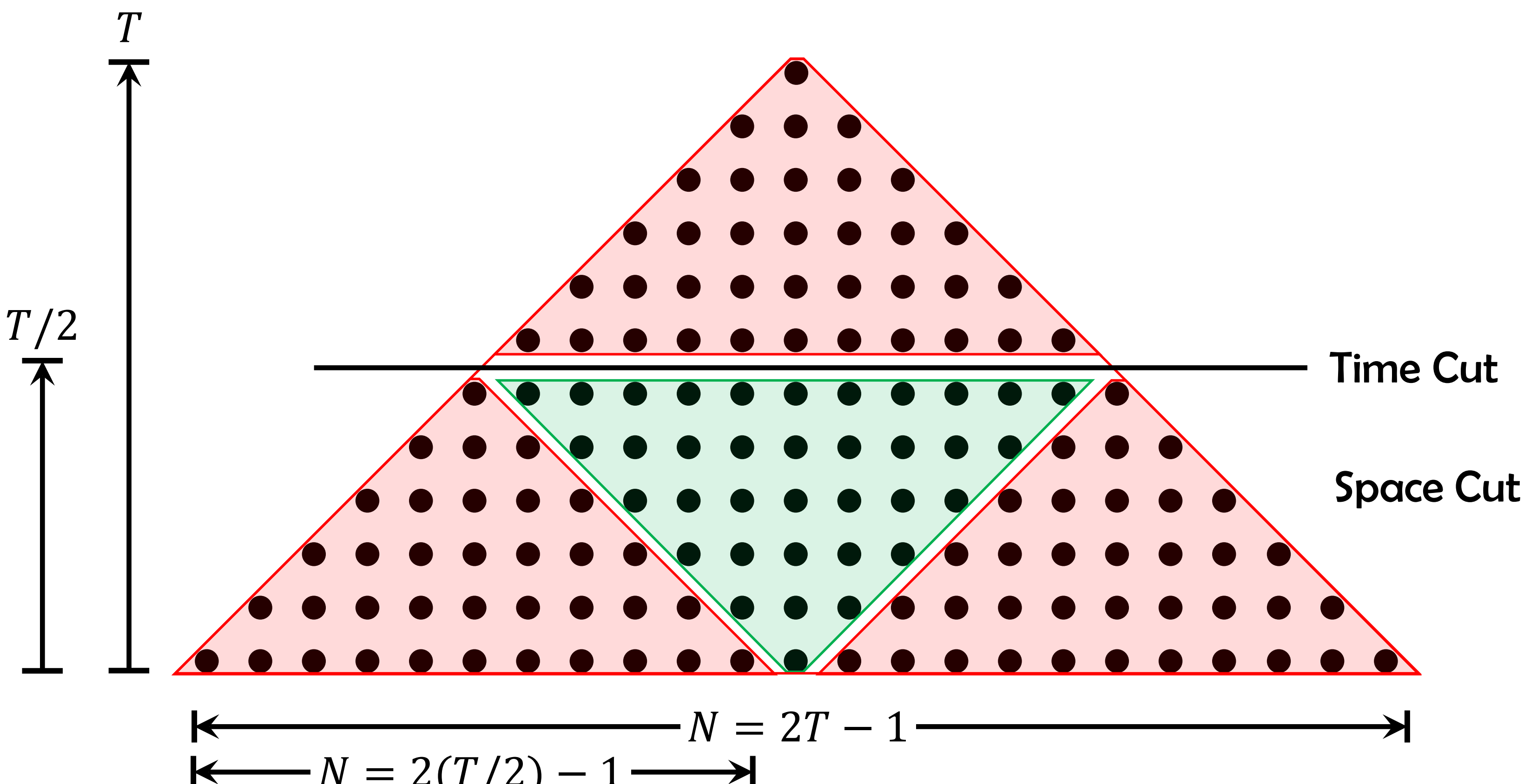


Span

$$T_\infty(T) = \begin{cases} \Theta(1), & if\ T \le c \\ 3T_\infty\left(\frac{T}{2}\right) + \Theta(1), & otherwise \end{cases}$$

$$= \Theta\left(T^{\log_2 3}\right) = \Theta\left(TN^{\log_2 3 - 1}\right)$$

Trapezoidal Decomposition Algorithm

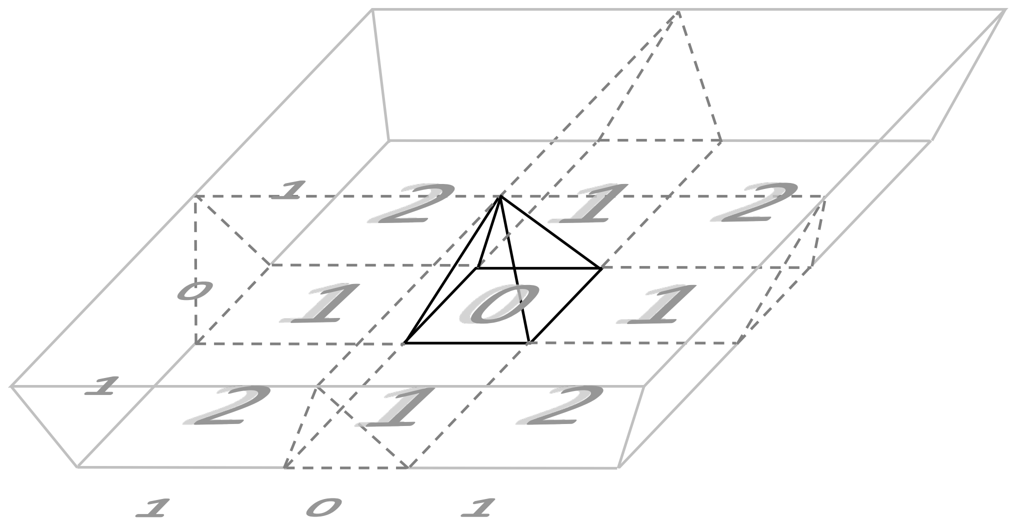


Serial Cache Complexity

$$Q_1(T) = \begin{cases} O(1 + M/B), & \textit{if } T \leq c'M \\ 4Q_1\left(\frac{T}{2}\right) + \Theta(1), & \textit{otherwise} \end{cases}$$

$$= O\left(\frac{T^2}{MB}\right) = O\left(\frac{NT}{MB}\right)$$

Trapezoidal Decomposition Algorithm



In grids of spatial dimension ≥ 2, a hyperspace cut simultaneously cuts as many spatial dimensions as possible.

All 3^k subtrapezoids created by a hyperspace cut on $k \geq 1$ of the $d \geq k$ spatial dimensions of a $(d+1)$-dimensional trapezoid can be evaluated in $k+1$ parallel steps.

Recent $o(NT)$ Work Stencil Algorithms

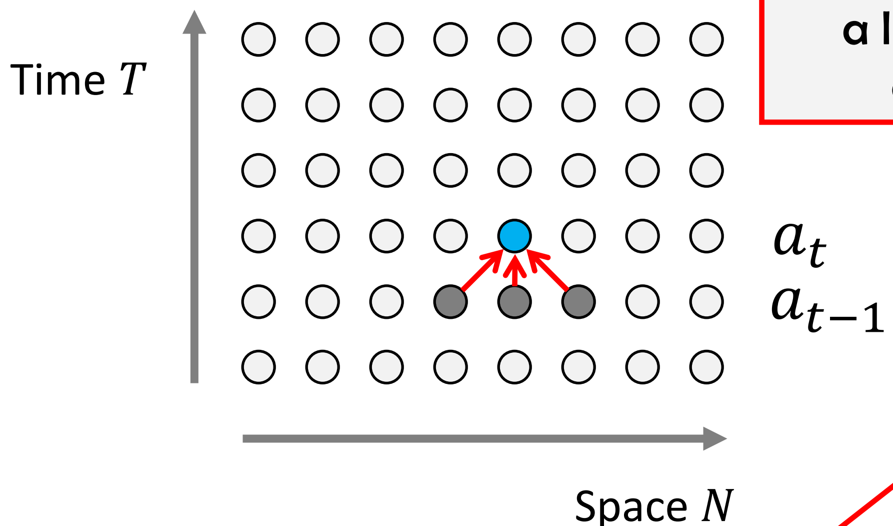
Theoretical Results

N: #cells in the spatial grid, T: #timesteps

Algorithm	Work	Span
Nested Loop	$\Theta(NT)$	$\Theta(T\log N)$
Tiled Loop		$\Theta\left(T\log M + \frac{T}{M^{1/d}}\log\frac{N}{M}\right)$
Recursive Tiling (Trapezoidal Decomp)		$\Theta\left(T\left(N^{1/d}\right)^{\log_2(d+2)-1}\right)$
FFT-Periodic (SPAA'21)	$\Theta(N\log(NT))$	$\Theta(\log T + \log N \log\log N)$
FFT-Aperiodic (SPAA'21)	$\Theta\left(TN^{1-\frac{1}{d}}\log\left(TN^{1-\frac{1}{d}}\right)\log T + N\log N\right)$	$\Theta(T)$ if $d = 1$ $\Theta(T\log N)$ if $d \geq 2$
Gaussian-Freespace (SPAA'22)	$\Theta\left(N\log^{O(d)}\left(\frac{1}{\epsilon}\right) + \log T\right)$	$\Theta\left(\log N\log^{O(d)}\left(\frac{1}{\epsilon}\right) + \log T\right)$

d: dimension of the spatial grid, M: tile size , ϵ: additive error

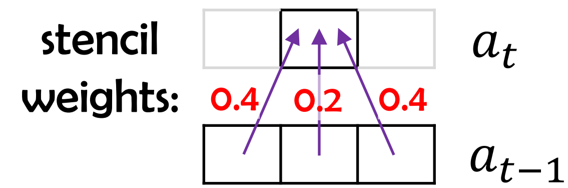
Applicability of the New Algorithms



each cell in a_t is computed as a linear combination of cells values in a_{t-1}

Linear Stencils

Example



$$a_t[i] = 0.4 \times a_{t-1}[i-1] + 0.2 \times a_{t-1}[i] + 0.4 \times a_{t-1}[i+1]$$

Can the techniques speed up nonlinear stencil computations?

We will see examples later where this happens.

Some Performance Numbers (Periodic and Aperiodic)

KNL (Intel Knight's Landing): 68 cores,   SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations

Benchmark		Stencil	N	T	Parallel runtime in seconds: PLuTo KNL	PLuTo SKX	Our algorithm KNL	Our algorithm SKX	Speedup factor over PLuTo: KNL	SKX
Periodic		heat1d	$1,600,000$	10^6	79	19	0.25	0.03	1754.7	759.6
		heat2d	$8,000 \times 8,000$	10^5	1,437	222	0.48	0.61	3,025.0	367.0
		seidel2d	$8,000 \times 8,000$	10^5	500	808	0.48	0.64	1,032.7	1268.6
		jacobi2d	$8,000 \times 8,000$	10^5	2,905	1017	0.48	0.68	6,084.7	1502.1
		heat3d	$800 \times 800 \times 800$	10^4	816	1466	4.98	5.48	163.9	267.3
		19pt3d	$800 \times 800 \times 800$	10^4	141	158	4.84	5.78	29.1	27.3
Aperiodic	Experiment 1	heat1d	$1,600,000$	10^6	50	35	5.85	6.69	8.5	5.2
		heat2d	$8,000 \times 8,000$	10^5	333	530	143.25	151.37	2.3	3.5
		seidel2d	$8,000 \times 8,000$	10^5	345	601	145.42	132.97	2.4	4.5
		jacobi2d	$8,000 \times 8,000$	10^5	567	456	249.04	273.46	2.3	1.7
		heat3d	$800 \times 800 \times 800$	10^4	513	763	395.10	605.89	1.3	1.3
		19pt3d	$800 \times 800 \times 800$	10^4	645	848	425.22	616.71	1.5	1.4
	Experiment 2	heat1d	$1,600,000$	N	32	23	5.63	6.87	5.7	3.3
		heat2d	$8,000 \times 8,000$	$\sqrt{N}$	210	312	92.78	121.70	2.3	2.6
		seidel2d	$8,000 \times 8,000$	$\sqrt{N}$	228	375	91.59	121.46	2.5	3.1
		jacobi2d	$8,000 \times 8,000$	$\sqrt{N}$	372	281	151.31	198.00	2.5	1.4
		heat3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	45	71	32.29	50.52	1.4	1.4
		19pt3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	61	71	33.82	52.27	1.8	1.4

Some Performance Numbers (Periodic and Aperiodic)

KNL (Intel Knight's Landing): 68 cores, SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations

Benchmark					Parallel runtime in seconds				Speedup factor over PLuTo	
					PLuTo		Our algorithm			
		Stencil	N	T	KNL	SKX	KNL	SKX	KNL	SKX
Periodic		heat1d	$1,600,000$	10^6	79	19	0.25	0.03	1754.7	759.6
Periodic		heat2d	$8,000 \times 8,000$	10^5	1,437	222	0.48	0.61	3,025.0	367.0
Periodic		seidel2d	$8,000 \times 8,000$	10^5	500	808	0.48	0.64	1,032.7	1268.6
Periodic		jacobi2d	$8,000 \times 8,000$	10^5	2,905	1017	0.48	0.68	6,084.7	1502.1
Periodic		heat3d	$800 \times 800 \times 800$	10^4	816	1466	4.98	5.48	163.9	267.3
Periodic		19pt3d	$800 \times 800 \times 800$	10^4	141	158	4.84	5.78	29.1	27.3
Aperiodic	Experiment 1	heat1d	$1,600,000$	10^6	50	35	5.85	6.69	8.5	5.2
Aperiodic	Experiment 1	heat2d	$8,000 \times 8,000$	10^5	333	530	143.25	151.37	2.3	3.5
Aperiodic	Experiment 1	seidel2d	$8,000 \times 8,000$	10^5	345	601	145.42	132.97	2.4	4.5
Aperiodic	Experiment 1	jacobi2d	$8,000 \times 8,000$	10^5	567	456	249.04	273.46	2.3	1.7
Aperiodic	Experiment 1	heat3d	$800 \times 800 \times 800$	10^4	513	763	395.10	605.89	1.3	1.3
Aperiodic	Experiment 1	19pt3d	$800 \times 800 \times 800$	10^4	645	848	425.22	616.71	1.5	1.4
Aperiodic	Experiment 2	heat1d	$1,600,000$	N	32	23	5.63	6.87	5.7	3.3
Aperiodic	Experiment 2	heat2d	$8,000 \times 8,000$	$\sqrt{N}$	210	312	92.78	121.70	2.3	2.6
Aperiodic	Experiment 2	seidel2d	$8,000 \times 8,000$	$\sqrt{N}$	228	375	91.59	121.46	2.5	3.1
Aperiodic	Experiment 2	jacobi2d	$8,000 \times 8,000$	$\sqrt{N}$	372	281	151.31	198.00	2.5	1.4
Aperiodic	Experiment 2	heat3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	45	71	32.29	50.52	1.4	1.4
Aperiodic	Experiment 2	19pt3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	61	71	33.82	52.27	1.8	1.4

Some Performance Numbers (Periodic and Aperiodic)

KNL (Intel Knight's Landing): 68 cores, SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations

Benchmark					Parallel runtime in seconds				Speedup factor over PLuTo	
					PLuTo		Our algorithm			
		Stencil	N	T	KNL	SKX	KNL	SKX	KNL	SKX
Periodic		heat1d	$1,600,000$	10^6	79	19	0.25	0.03	1754.7	759.6
		heat2d	$8,000 \times 8,000$	10^5	1,437	222	0.48	0.61	3,025.0	367.0
		seidel2d	$8,000 \times 8,000$	10^5	500	808	0.48	0.64	1,032.7	1268.6
		jacobi2d	$8,000 \times 8,000$	10^5	2,905	1017	0.48	0.68	6,084.7	1502.1
		heat3d	$800 \times 800 \times 800$	10^4	816	1466	4.98	5.48	163.9	267.3
		19pt3d	$800 \times 800 \times 800$	10^4	141	158	4.84	5.78	29.1	27.3
Aperiodic	Experiment 1	heat1d	$1,600,000$	10^6	50	35	5.85	6.69	8.5	5.2
		heat2d	$8,000 \times 8,000$	10^5	333	530	143.25	151.37	2.3	3.5
		seidel2d	$8,000 \times 8,000$	10^5	345	601	145.42	132.97	2.4	4.5
		jacobi2d	$8,000 \times 8,000$	10^5	567	456	249.04	273.46	2.3	1.7
		heat3d	$800 \times 800 \times 800$	10^4	513	763	395.10	605.89	1.3	1.3
		19pt3d	$800 \times 800 \times 800$	10^4	645	848	425.22	616.71	1.5	1.4
	Experiment 2	heat1d	$1,600,000$	N	32	23	5.63	6.87	5.7	3.3
		heat2d	$8,000 \times 8,000$	$\sqrt{N}$	210	312	92.78	121.70	2.3	2.6
		seidel2d	$8,000 \times 8,000$	$\sqrt{N}$	228	375	91.59	121.46	2.5	3.1
		jacobi2d	$8,000 \times 8,000$	$\sqrt{N}$	372	281	151.31	198.00	2.5	1.4
		heat3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	45	71	32.29	50.52	1.4	1.4
		19pt3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	61	71	33.82	52.27	1.8	1.4

Some Performance Numbers (Periodic and Aperiodic)

KNL (Intel Knight's Landing): 68 cores, SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations

Benchmark		Stencil	N	T	Parallel runtime in seconds: PLuTo KNL	PLuTo SKX	Our algorithm KNL	Our algorithm SKX	Speedup factor over PLuTo: KNL	SKX
Periodic		heat1d	$1,600,000$	10^6	79	19	0.25	0.03	1754.7	759.6
		heat2d	$8,000 \times 8,000$	10^5	1,437	222	0.48	0.61	3,025.0	367.0
		seidel2d	$8,000 \times 8,000$	10^5	500	808	0.48	0.64	1,032.7	1268.6
		jacobi2d	$8,000 \times 8,000$	10^5	2,905	1017	0.48	0.68	6,084.7	1502.1
		heat3d	$800 \times 800 \times 800$	10^4	816	1466	4.98	5.48	163.9	267.3
		19pt3d	$800 \times 800 \times 800$	10^4	141	158	4.84	5.78	29.1	27.3
Aperiodic	Experiment 1	heat1d	$1,600,000$	10^6	50	35	5.85	6.69	8.5	5.2
		heat2d	$8,000 \times 8,000$	10^5	333	530	143.25	151.37	2.3	3.5
		seidel2d	$8,000 \times 8,000$	10^5	345	601	145.42	132.97	2.4	4.5
		jacobi2d	$8,000 \times 8,000$	10^5	567	456	249.04	273.46	2.3	1.7
		heat3d	$800 \times 800 \times 800$	10^4	513	763	395.10	605.89	1.3	1.3
		19pt3d	$800 \times 800 \times 800$	10^4	645	848	425.22	616.71	1.5	1.4
	Experiment 2	heat1d	$1,600,000$	N	32	23	5.63	6.87	5.7	3.3
		heat2d	$8,000 \times 8,000$	$\sqrt{N}$	210	312	92.78	121.70	2.3	2.6
		seidel2d	$8,000 \times 8,000$	$\sqrt{N}$	228	375	91.59	121.46	2.5	3.1
		jacobi2d	$8,000 \times 8,000$	$\sqrt{N}$	372	281	151.31	198.00	2.5	1.4
		heat3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	45	71	32.29	50.52	1.4	1.4
		19pt3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	61	71	33.82	52.27	1.8	1.4

Some Performance Numbers (Periodic and Aperiodic)

KNL (Intel Knight's Landing): 68 cores, SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations

Benchmark		Stencil	N	T	Parallel runtime in seconds: PLuTo KNL	PLuTo SKX	Our algorithm KNL	Our algorithm SKX	Speedup factor over PLuTo: KNL	SKX
Periodic		heat1d	$1,600,000$	10^6	79	19	0.25	0.03	1754.7	759.6
Periodic		heat2d	$8,000 \times 8,000$	10^5	1,437	222	0.48	0.61	3,025.0	367.0
Periodic		seidel2d	$8,000 \times 8,000$	10^5	500	808	0.48	0.64	1,032.7	1268.6
Periodic		jacobi2d	$8,000 \times 8,000$	10^5	2,905	1017	0.48	0.68	6,084.7	1502.1
Periodic		heat3d	$800 \times 800 \times 800$	10^4	816	1466	4.98	5.48	163.9	267.3
Periodic		19pt3d	$800 \times 800 \times 800$	10^4	141	158	4.84	5.78	29.1	27.3
Aperiodic	Experiment 1	heat1d	$1,600,000$	10^6	50	35	5.85	6.69	8.5	5.2
Aperiodic	Experiment 1	heat2d	$8,000 \times 8,000$	10^5	333	530	143.25	151.37	2.3	3.5
Aperiodic	Experiment 1	seidel2d	$8,000 \times 8,000$	10^5	345	601	145.42	132.97	2.4	4.5
Aperiodic	Experiment 1	jacobi2d	$8,000 \times 8,000$	10^5	567	456	249.04	273.46	2.3	1.7
Aperiodic	Experiment 1	heat3d	$800 \times 800 \times 800$	10^4	513	763	395.10	605.89	1.3	1.3
Aperiodic	Experiment 1	19pt3d	$800 \times 800 \times 800$	10^4	645	848	425.22	616.71	1.5	1.4
Aperiodic	Experiment 2	heat1d	$1,600,000$	N	32	23	5.63	6.87	5.7	3.3
Aperiodic	Experiment 2	heat2d	$8,000 \times 8,000$	$\sqrt{N}$	210	312	92.78	121.70	2.3	2.6
Aperiodic	Experiment 2	seidel2d	$8,000 \times 8,000$	$\sqrt{N}$	228	375	91.59	121.46	2.5	3.1
Aperiodic	Experiment 2	jacobi2d	$8,000 \times 8,000$	$\sqrt{N}$	372	281	151.31	198.00	2.5	1.4
Aperiodic	Experiment 2	heat3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	45	71	32.29	50.52	1.4	1.4
Aperiodic	Experiment 2	19pt3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	61	71	33.82	52.27	1.8	1.4

Some Performance Numbers (Periodic and Aperiodic)

KNL (Intel Knight's Landing): 68 cores, SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations

Benchmark					Parallel runtime in seconds				Speedup factor over PLuTo	
					PLuTo		Our algorithm			
		Stencil	N	T	KNL	SKX	KNL	SKX	KNL	SKX
Periodic		heat1d	$1,600,000$	10^6	79	19	0.25	0.03	1754.7	759.6
Periodic		heat2d	$8,000 \times 8,000$	10^5	1,437	222	0.48	0.61	3,025.0	367.0
Periodic		seidel2d	$8,000 \times 8,000$	10^5	500	808	0.48	0.64	1,032.7	1268.6
Periodic		jacobi2d	$8,000 \times 8,000$	10^5	2,905	1017	0.48	0.68	6,084.7	1502.1
Periodic		heat3d	$800 \times 800 \times 800$	10^4	816	1466	4.98	5.48	163.9	267.3
Periodic		19pt3d	$800 \times 800 \times 800$	10^4	141	158	4.84	5.78	29.1	27.3
Aperiodic	Experiment 1	heat1d	$1,600,000$	10^6	50	35	5.85	6.69	8.5	5.2
Aperiodic	Experiment 1	heat2d	$8,000 \times 8,000$	10^5	333	530	143.25	151.37	2.3	3.5
Aperiodic	Experiment 1	seidel2d	$8,000 \times 8,000$	10^5	345	601	145.42	132.97	2.4	4.5
Aperiodic	Experiment 1	jacobi2d	$8,000 \times 8,000$	10^5	567	456	249.04	273.46	2.3	1.7
Aperiodic	Experiment 1	heat3d	$800 \times 800 \times 800$	10^4	513	763	395.10	605.89	1.3	1.3
Aperiodic	Experiment 1	19pt3d	$800 \times 800 \times 800$	10^4	645	848	425.22	616.71	1.5	1.4
Aperiodic	Experiment 2	heat1d	$1,600,000$	N	32	23	5.63	6.87	5.7	3.3
Aperiodic	Experiment 2	heat2d	$8,000 \times 8,000$	$\sqrt{N}$	210	312	92.78	121.70	2.3	2.6
Aperiodic	Experiment 2	seidel2d	$8,000 \times 8,000$	$\sqrt{N}$	228	375	91.59	121.46	2.5	3.1
Aperiodic	Experiment 2	jacobi2d	$8,000 \times 8,000$	$\sqrt{N}$	372	281	151.31	198.00	2.5	1.4
Aperiodic	Experiment 2	heat3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	45	71	32.29	50.52	1.4	1.4
Aperiodic	Experiment 2	19pt3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	61	71	33.82	52.27	1.8	1.4

Some Performance Numbers (Periodic and Aperiodic)

KNL (Intel Knight's Landing): 68 cores, SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations

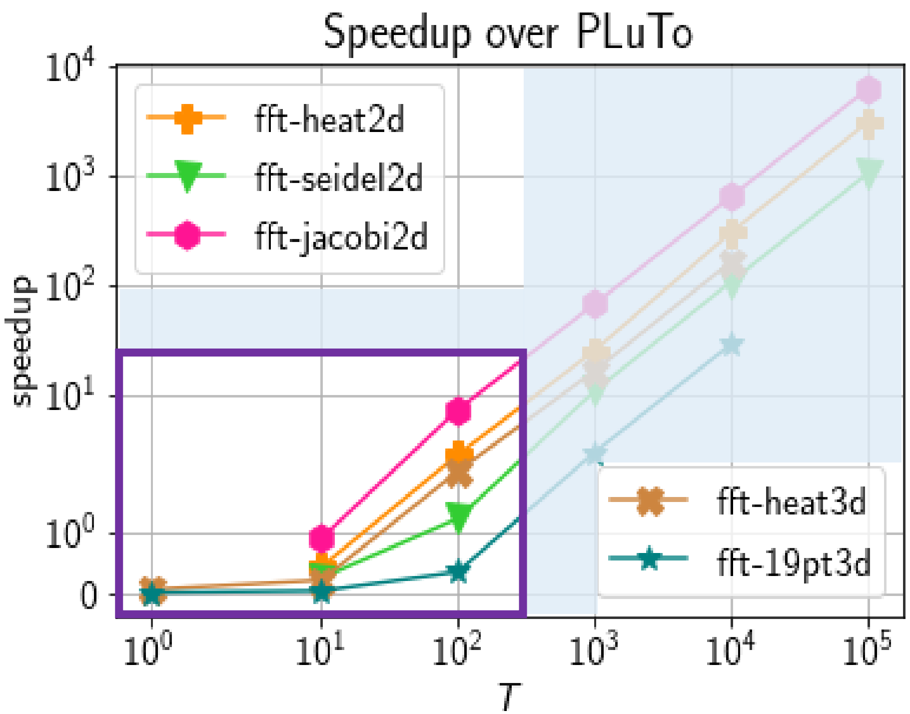
Benchmark		Stencil	N	T	Parallel runtime in seconds: PLuTo KNL	PLuTo SKX	Our algorithm KNL	Our algorithm SKX	Speedup factor over PLuTo: KNL	SKX
Periodic		heat1d	$1,600,000$	10^6	79	19	0.25	0.03	1754.7	759.6
Periodic		heat2d	$8,000 \times 8,000$	10^5	1,437	222	0.48	0.61	3,025.0	367.0
Periodic		seidel2d	$8,000 \times 8,000$	10^5	500	808	0.48	0.64	1,032.7	1268.6
Periodic		jacobi2d	$8,000 \times 8,000$	10^5	2,905	1017	0.48	0.68	6,084.7	1502.1
Periodic		heat3d	$800 \times 800 \times 800$	10^4	816	1466	4.98	5.48	163.9	267.3
Periodic		19pt3d	$800 \times 800 \times 800$	10^4	141	158	4.84	5.78	29.1	27.3
Aperiodic	Experiment 1	heat1d	$1,600,000$	10^6	50	35	5.85	6.69	8.5	5.2
Aperiodic	Experiment 1	heat2d	$8,000 \times 8,000$	10^5	333	530	143.25	151.37	2.3	3.5
Aperiodic	Experiment 1	seidel2d	$8,000 \times 8,000$	10^5	345	601	145.42	132.97	2.4	4.5
Aperiodic	Experiment 1	jacobi2d	$8,000 \times 8,000$	10^5	567	456	249.04	273.46	2.3	1.7
Aperiodic	Experiment 1	heat3d	$800 \times 800 \times 800$	10^4	513	763	395.10	605.89	1.3	1.3
Aperiodic	Experiment 1	19pt3d	$800 \times 800 \times 800$	10^4	645	848	425.22	616.71	1.5	1.4
Aperiodic	Experiment 2	heat1d	$1,600,000$	N	32	23	5.63	6.87	5.7	3.3
Aperiodic	Experiment 2	heat2d	$8,000 \times 8,000$	$\sqrt{N}$	210	312	92.78	121.70	2.3	2.6
Aperiodic	Experiment 2	seidel2d	$8,000 \times 8,000$	$\sqrt{N}$	228	375	91.59	121.46	2.5	3.1
Aperiodic	Experiment 2	jacobi2d	$8,000 \times 8,000$	$\sqrt{N}$	372	281	151.31	198.00	2.5	1.4
Aperiodic	Experiment 2	heat3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	45	71	32.29	50.52	1.4	1.4
Aperiodic	Experiment 2	19pt3d	$800 \times 800 \times 800$	$\sqrt[3]{N}$	61	71	33.82	52.27	1.8	1.4

Some Performance Numbers (Periodic)

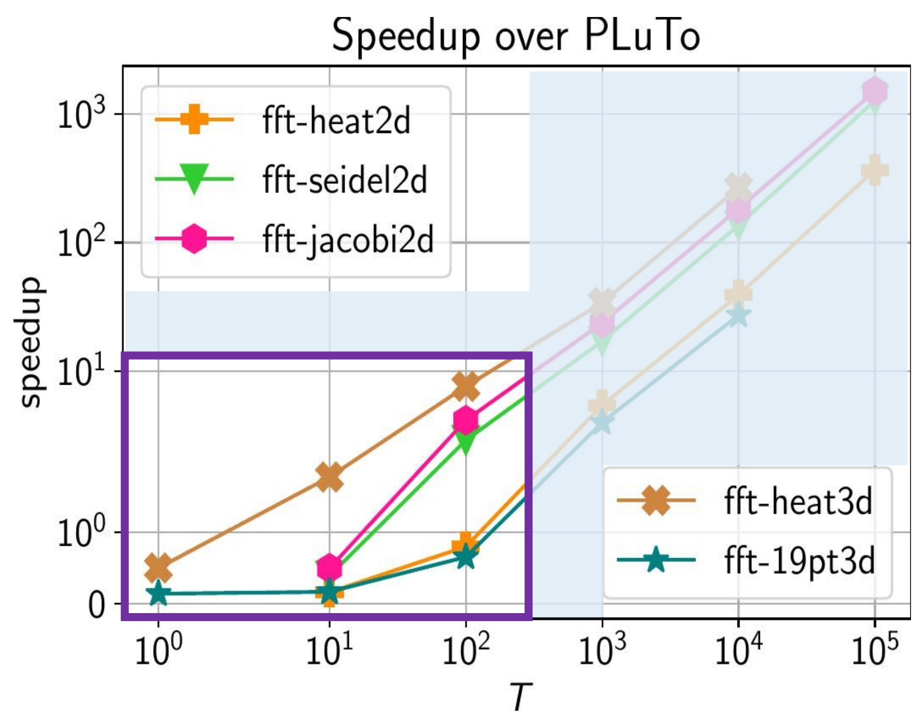
KNL (Intel Knight's Landing): 68 cores, SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations



KNL



SKX

2D grid size: 8000×8000

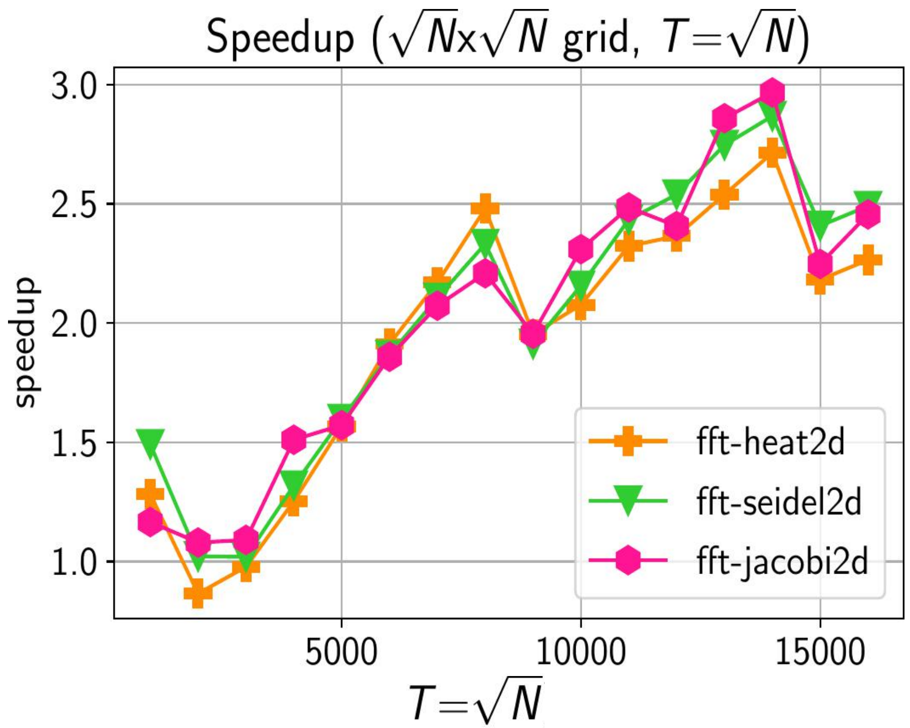
3D grid size: $800 \times 800 \times 800$

Some Performance Numbers (Aperiodic, $T = N^{1/d}$)

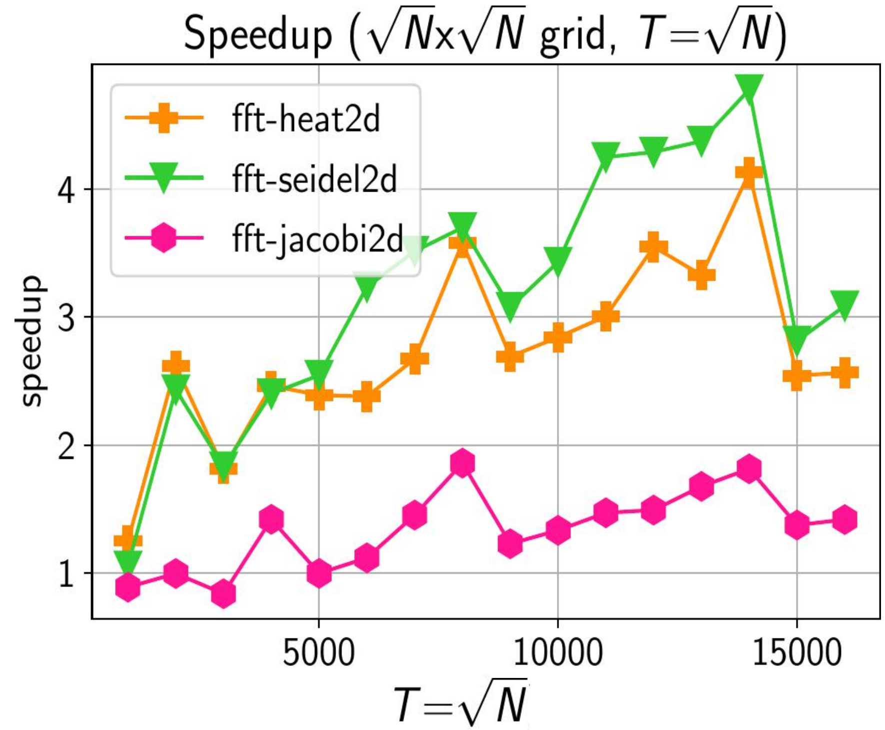
KNL (Intel Knight's Landing): 68 cores, SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations



KNL



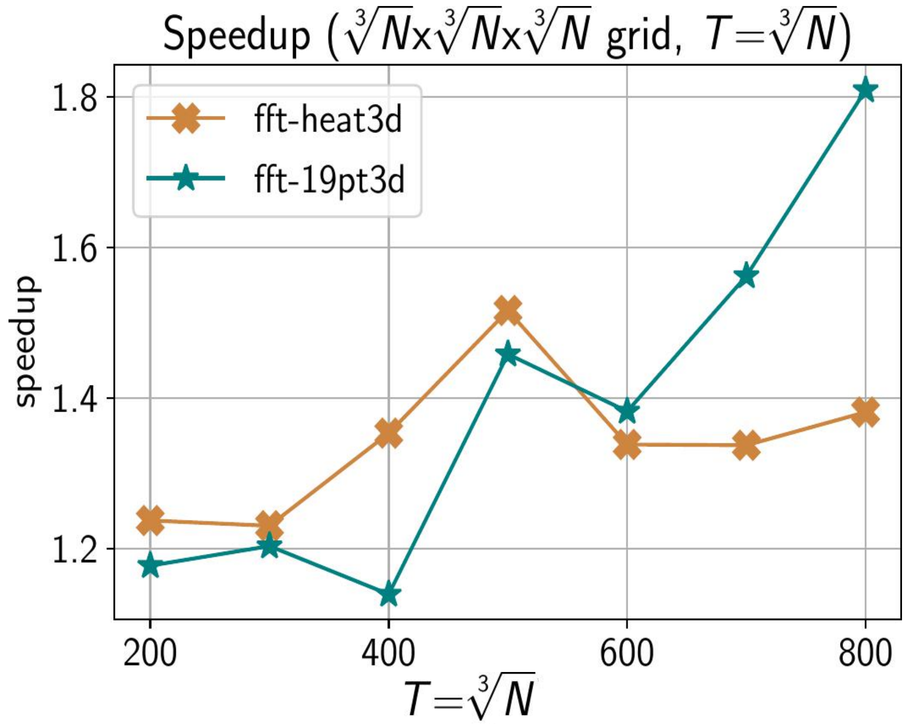
SKX

Some Performance Numbers (Aperiodic, $T = N^{1/d}$)

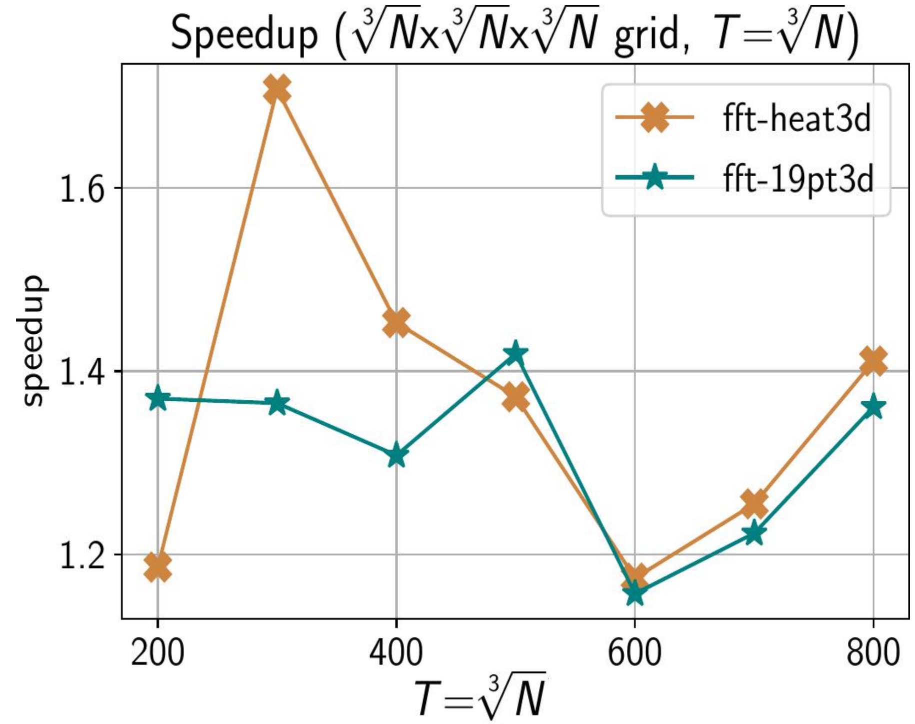
KNL (Intel Knight's Landing): 68 cores, SKX (Intel Skylake): 48 cores

PLuTo: State-of-the-art Tiled Looping Code Generator for Multicores

Intel MKL was used for FFT Implementations



KNL



SKX

Some Performance Numbers (Freespace)

SKX (Intel Skylake)
- cores: 48 (24 / socket)
- caches: 32 KB L1 / core, 1 MB L2 / core, 33 MB L3 / socket
- RAM: 192 GB DDR4

Stencil	N	T	Running Time (sec)			Error (w.r.t. FFT)			
			Gaussian	PLuTo	FFT	Mean Squared	Max Absolute	Mean Relative	Max Relative
heat 1d (3 pt)	1.6×10^6	10^3	0.10	0.49	0.02	1.13×10^{-12}	2.29×10^{-10}	1.20×10^{-9}	3.03×10^{-5}
		10^4	0.11	2.10	0.02	2.01×10^{-13}	2.29×10^{-11}	5.27×10^{-10}	3.03×10^{-6}
		10^5	0.11	8.73	0.02	3.58×10^{-14}	2.29×10^{-12}	2.50×10^{-10}	3.03×10^{-7}
		10^6	0.11	217.56	0.04	6.44×10^{-15}	2.30×10^{-13}	1.15×10^{-10}	3.05×10^{-8}
		10^7	0.10	$> 10\ m$	0.32	1.57×10^{-14}	5.03×10^{-14}	1.41×10^{-9}	6.74×10^{-9}
heat 2d (5 pt)	1000×1000	10^3	0.45	1.58	0.08	7.29×10^{-11}	3.34×10^{-10}	5.91×10^{-6}	6.36×10^{-5}
		10^4	0.42	53.74	4.55	3.12×10^{-11}	4.22×10^{-11}	3.41×10^{-6}	6.30×10^{-6}
		10^5	0.15	out of memory					
Seidel 2d (9 pt)	1000×1000	10^3	0.40	2.41	0.08	1.23×10^{-10}	5.31×10^{-10}	1.19×10^{-5}	9.36×10^{-5}
		10^4	0.40	567.46	5.59	2.13×10^{-11}	6.12×10^{-11}	2.14×10^{-6}	8.72×10^{-6}
		10^5	0.15	out of memory					
Jacobi 2d (25 pt)	1000×1000	10^3	0.60	4.24	0.22	1.30×10^{-7}	1.61×10^{-7}	1.35×10^{-2}	2.08×10^{-2}
		10^4	0.51	out of memory					
heat 3d (7 pt)	$100 \times 100 \times 100$	$10^{2.7}$	10.35	271.52	11.81	2.21×10^{-10}	5.28×10^{-10}	2.76×10^{-5}	1.08×10^{-4}
		10^3	3.31	out of memory					

Input cell values are in $(0, 1]$

Some Performance Numbers (Freespace)

SKX (Intel Skylake)
- cores: 48 (24 / socket)
- caches: 32 KB L1 / core, 1 MB L2 / core, 33 MB L3 / socket
- RAM: 192 GB DDR4

Stencil	N	T	Running Time (sec)			Error (w.r.t. FFT)			
			Gaussian	PLuTo	FFT	Mean Squared	Max Absolute	Mean Relative	Max Relative
heat 1d (3 pt)	1.6×10^6	10^3	0.10	0.49	0.02	1.13×10^{-12}	2.29×10^{-10}	1.20×10^{-9}	3.03×10^{-5}
		10^4	0.11	2.10	0.02	2.01×10^{-13}	2.29×10^{-11}	5.27×10^{-10}	3.03×10^{-6}
		10^5	0.11	8.73	0.02	3.58×10^{-14}	2.29×10^{-12}	2.50×10^{-10}	3.03×10^{-7}
		10^6	0.11	217.56	0.04	6.44×10^{-15}	2.30×10^{-13}	1.15×10^{-10}	3.05×10^{-8}
		10^7	0.10	$> 10\ m$	0.32	1.57×10^{-14}	5.03×10^{-14}	1.41×10^{-9}	6.74×10^{-9}
heat 2d (5 pt)	1000×1000	10^3	0.45	1.58	0.08	7.29×10^{-11}	3.34×10^{-10}	5.91×10^{-6}	6.36×10^{-5}
		10^4	0.42	53.74	4.55	3.12×10^{-11}	4.22×10^{-11}	3.41×10^{-6}	6.30×10^{-6}
		10^5	0.15	out of memory					
Seidel 2d (9 pt)	1000×1000	10^3	0.40	2.41	0.08	1.23×10^{-10}	5.31×10^{-10}	1.19×10^{-5}	9.36×10^{-5}
		10^4	0.40	567.46	5.59	2.13×10^{-11}	6.12×10^{-11}	2.14×10^{-6}	8.72×10^{-6}
		10^5	0.15	out of memory					
Jacobi 2d (25 pt)	1000×1000	10^3	0.60	4.24	0.22	1.30×10^{-7}	1.61×10^{-7}	1.35×10^{-2}	2.08×10^{-2}
		10^4	0.51	out of memory					
heat 3d (7 pt)	$100 \times 100 \times 100$	$10^{2.7}$	10.35	271.52	11.81	2.21×10^{-10}	5.28×10^{-10}	2.76×10^{-5}	1.08×10^{-4}
		10^3	3.31	out of memory					

Input cell values are in $(0, 1]$

Some Performance Numbers (Freespace)

SKX (Intel Skylake)
- cores: 48 (24 / socket)
- caches: 32 KB L1 / core, 1 MB L2 / core, 33 MB L3 / socket
- RAM: 192 GB DDR4

Stencil	N	T	Running Time (sec)			Error (w.r.t. FFT)			
			Gaussian	PLuTo	FFT	Mean Squared	Max Absolute	Mean Relative	Max Relative
heat 1d (3 pt)	1.6×10^6	10^3	0.10	0.49	0.02	1.13×10^{-12}	2.29×10^{-10}	1.20×10^{-9}	3.03×10^{-5}
		10^4	0.11	2.10	0.02	2.01×10^{-13}	2.29×10^{-11}	5.27×10^{-10}	3.03×10^{-6}
		10^5	0.11	8.73	0.02	3.58×10^{-14}	2.29×10^{-12}	2.50×10^{-10}	3.03×10^{-7}
		10^6	0.11	217.56	0.04	6.44×10^{-15}	2.30×10^{-13}	1.15×10^{-10}	3.05×10^{-8}
		10^7	0.10	$> 10\,m$	0.32	1.57×10^{-14}	5.03×10^{-14}	1.41×10^{-9}	6.74×10^{-9}
heat 2d (5 pt)	1000×1000	10^3	0.45	1.58	0.08	7.29×10^{-11}	3.34×10^{-10}	5.91×10^{-6}	6.36×10^{-5}
		10^4	0.42	53.74	4.55	3.12×10^{-11}	4.22×10^{-11}	3.41×10^{-6}	6.30×10^{-6}
		10^5	0.15	out of memory					
Seidel 2d (9 pt)	1000×1000	10^3	0.40	2.41	0.08	1.23×10^{-10}	5.31×10^{-10}	1.19×10^{-5}	9.36×10^{-5}
		10^4	0.40	567.46	5.59	2.13×10^{-11}	6.12×10^{-11}	2.14×10^{-6}	8.72×10^{-6}
		10^5	0.15	out of memory					
Jacobi 2d (25 pt)	1000×1000	10^3	0.60	4.24	0.22	1.30×10^{-7}	1.61×10^{-7}	1.35×10^{-2}	2.08×10^{-2}
		10^4	0.51	out of memory					
heat 3d (7 pt)	$100 \times 100 \times 100$	$10^{2.7}$	10.35	271.52	11.81	2.21×10^{-10}	5.28×10^{-10}	2.76×10^{-5}	1.08×10^{-4}
		10^3	3.31	out of memory					

Input cell values are in $(0, 1]$

Some Performance Numbers (Freespace)

SKX (Intel Skylake)
- cores: 48 (24 / socket)
- caches: 32 KB L1 / core, 1 MB L2 / core, 33 MB L3 / socket
- RAM: 192 GB DDR4

Stencil	N	T	Running Time (sec)			Error (w.r.t. FFT)			
			Gaussian	PLuTo	FFT	Mean Squared	Max Absolute	Mean Relative	Max Relative
heat 1d (3 pt)	1.6×10^6	10^3	0.10	0.49	0.02	1.13×10^{-12}	2.29×10^{-10}	1.20×10^{-9}	3.03×10^{-5}
		10^4	0.11	2.10	0.02	2.01×10^{-13}	2.29×10^{-11}	5.27×10^{-10}	3.03×10^{-6}
		10^5	0.11	8.73	0.02	3.58×10^{-14}	2.29×10^{-12}	2.50×10^{-10}	3.03×10^{-7}
		10^6	0.11	217.56	0.04	6.44×10^{-15}	2.30×10^{-13}	1.15×10^{-10}	3.05×10^{-8}
		10^7	0.10	$> 10\ m$	0.32	1.57×10^{-14}	5.03×10^{-14}	1.41×10^{-9}	6.74×10^{-9}
heat 2d (5 pt)	1000×1000	10^3	0.45	1.58	0.08	7.29×10^{-11}	3.34×10^{-10}	5.91×10^{-6}	6.36×10^{-5}
		10^4	0.42	53.74	4.55	3.12×10^{-11}	4.22×10^{-11}	3.41×10^{-6}	6.30×10^{-6}
		10^5	0.15	out of memory					
Seidel 2d (9 pt)	1000×1000	10^3	0.40	2.41	0.08	1.23×10^{-10}	5.31×10^{-10}	1.19×10^{-5}	9.36×10^{-5}
		10^4	0.40	567.46	5.59	2.13×10^{-11}	6.12×10^{-11}	2.14×10^{-6}	8.72×10^{-6}
		10^5	0.15	out of memory					
Jacobi 2d (25 pt)	1000×1000	10^3	0.60	4.24	0.22	1.30×10^{-7}	1.61×10^{-7}	1.35×10^{-2}	2.08×10^{-2}
		10^4	0.51	out of memory					
heat 3d (7 pt)	$100 \times 100 \times 100$	$10^{2.7}$	10.35	271.52	11.81	2.21×10^{-10}	5.28×10^{-10}	2.76×10^{-5}	1.08×10^{-4}
		10^3	3.31	out of memory					

Input cell values are in $(0, 1]$

Some Performance Numbers (Freespace)

SKX (Intel Skylake)
- cores: 48 (24 / socket)
- caches: 32 KB L1 / core, 1 MB L2 / core, 33 MB L3 / socket
- RAM: 192 GB DDR4

Stencil	N	T	Running Time (sec)			Error (w.r.t. FFT)			
			Gaussian	PLuTo	FFT	Mean Squared	Max Absolute	Mean Relative	Max Relative
heat 1d (3 pt)	1.6×10^6	10^3	0.10	0.49	0.02	1.13×10^{-12}	2.29×10^{-10}	1.20×10^{-9}	3.03×10^{-5}
		10^4	0.11	2.10	0.02	2.01×10^{-13}	2.29×10^{-11}	5.27×10^{-10}	3.03×10^{-6}
		10^5	0.11	8.73	0.02	3.58×10^{-14}	2.29×10^{-12}	2.50×10^{-10}	3.03×10^{-7}
		10^6	0.11	217.56	0.04	6.44×10^{-15}	2.30×10^{-13}	1.15×10^{-10}	3.05×10^{-8}
		10^7	0.10	$> 10\ m$	0.32	1.57×10^{-14}	5.03×10^{-14}	1.41×10^{-9}	6.74×10^{-9}
heat 2d (5 pt)	1000×1000	10^3	0.45	1.58	0.08	7.29×10^{-11}	3.34×10^{-10}	5.91×10^{-6}	6.36×10^{-5}
		10^4	0.42	53.74	4.55	3.12×10^{-11}	4.22×10^{-11}	3.41×10^{-6}	6.30×10^{-6}
		10^5	0.15	out of memory					
Seidel 2d (9 pt)	1000×1000	10^3	0.40	2.41	0.08	1.23×10^{-10}	5.31×10^{-10}	1.19×10^{-5}	9.36×10^{-5}
		10^4	0.40	567.46	5.59	2.13×10^{-11}	6.12×10^{-11}	2.14×10^{-6}	8.72×10^{-6}
		10^5	0.15	out of memory					
Jacobi 2d (25 pt)	1000×1000	10^3	0.60	4.24	0.22	1.30×10^{-7}	1.61×10^{-7}	1.35×10^{-2}	2.08×10^{-2}
		10^4	0.51	out of memory					
heat 3d (7 pt)	$100 \times 100 \times 100$	$10^{2.7}$	10.35	271.52	11.81	2.21×10^{-10}	5.28×10^{-10}	2.76×10^{-5}	1.08×10^{-4}
		10^3	3.31	out of memory					

Input cell values are in $(0, 1]$

Some Performance Numbers (Freespace)

SKX (Intel Skylake)
- cores: 48 (24 / socket)
- caches: 32 KB L1 / core, 1 MB L2 / core, 33 MB L3 / socket
- RAM: 192 GB DDR4

Stencil	N	T	Running Time (sec)			Error (w.r.t. FFT)			
			Gaussian	PLuTo	FFT	Mean Squared	Max Absolute	Mean Relative	Max Relative
heat 1d (3 pt)	1.6×10^6	10^3	0.10	0.49	0.02	1.13×10^{-12}	2.29×10^{-10}	1.20×10^{-9}	3.03×10^{-5}
		10^4	0.11	2.10	0.02	2.01×10^{-13}	2.29×10^{-11}	5.27×10^{-10}	3.03×10^{-6}
		10^5	0.11	8.73	0.02	3.58×10^{-14}	2.29×10^{-12}	2.50×10^{-10}	3.03×10^{-7}
		10^6	0.11	217.56	0.04	6.44×10^{-15}	2.30×10^{-13}	1.15×10^{-10}	3.05×10^{-8}
		10^7	0.10	$> 10\, m$	0.32	1.57×10^{-14}	5.03×10^{-14}	1.41×10^{-9}	6.74×10^{-9}
heat 2d (5 pt)	1000×1000	10^3	0.45	1.58	0.08	7.29×10^{-11}	3.34×10^{-10}	5.91×10^{-6}	6.36×10^{-5}
		10^4	0.42	53.74	4.55	3.12×10^{-11}	4.22×10^{-11}	3.41×10^{-6}	6.30×10^{-6}
		10^5	0.15	out of memory					
Seidel 2d (9 pt)	1000×1000	10^3	0.40	2.41	0.08	1.23×10^{-10}	5.31×10^{-10}	1.19×10^{-5}	9.36×10^{-5}
		10^4	0.40	567.46	5.59	2.13×10^{-11}	6.12×10^{-11}	2.14×10^{-6}	8.72×10^{-6}
		10^5	0.15	out of memory					
Jacobi 2d (25 pt)	1000×1000	10^3	0.60	4.24	0.22	1.30×10^{-7}	1.61×10^{-7}	1.35×10^{-2}	2.08×10^{-2}
		10^4	0.51	out of memory					
heat 3d (7 pt)	$100 \times 100 \times 100$	$10^{2.7}$	10.35	271.52	11.81	2.21×10^{-10}	5.28×10^{-10}	2.76×10^{-5}	1.08×10^{-4}
		10^3	3.31	out of memory					

Input cell values are in $(0, 1]$

Some Performance Numbers (Freespace)

SKX (Intel Skylake)

- cores: 48 (24 / socket)
- caches: 32 KB L1 / core, 1 MB L2 / core, 33 MB L3 / socket
- RAM: 192 GB DDR4

Stencil	N	T	Running Time (sec)			Error (w.r.t. FFT)			
			Gaussian	PLuTo	FFT	Mean Squared	Max Absolute	Mean Relative	Max Relative
heat 1d (3 pt)	1.6×10^6	10^3	0.10	0.49	0.02	1.13×10^{-12}	2.29×10^{-10}	1.20×10^{-9}	3.03×10^{-5}
		10^4	0.11	2.10	0.02	2.01×10^{-13}	2.29×10^{-11}	5.27×10^{-10}	3.03×10^{-6}
		10^5	0.11	8.73	0.02	3.58×10^{-14}	2.29×10^{-12}	2.50×10^{-10}	3.03×10^{-7}
		10^6	0.11	217.56	0.04	6.44×10^{-15}	2.30×10^{-13}	1.15×10^{-10}	3.05×10^{-8}
		10^7	0.10	$> 10\ m$	0.32	1.57×10^{-14}	5.03×10^{-14}	1.41×10^{-9}	6.74×10^{-9}
heat 2d (5 pt)	1000×1000	10^3	0.45	1.58	0.08	7.29×10^{-11}	3.34×10^{-10}	5.91×10^{-6}	6.36×10^{-5}
		10^4	0.42	53.74	4.55	3.12×10^{-11}	4.22×10^{-11}	3.41×10^{-6}	6.30×10^{-6}
		10^5	0.15	out of memory					
Seidel 2d (9 pt)	1000×1000	10^3	0.40	2.41	0.08	1.23×10^{-10}	5.31×10^{-10}	1.19×10^{-5}	9.36×10^{-5}
		10^4	0.40	567.46	5.59	2.13×10^{-11}	6.12×10^{-11}	2.14×10^{-6}	8.72×10^{-6}
		10^5	0.15	out of memory					
Jacobi 2d (25 pt)	1000×1000	10^3	0.60	4.24	0.22	1.30×10^{-7}	1.61×10^{-7}	1.35×10^{-2}	2.08×10^{-2}
		10^4	0.51	out of memory					
heat 3d (7 pt)	$100 \times 100 \times 100$	$10^{2.7}$	10.35	271.52	11.81	2.21×10^{-10}	5.28×10^{-10}	2.76×10^{-5}	1.08×10^{-4}
		10^3	3.31	out of memory					

Input cell values are in $(0, 1]$

Periodic Grids:

Application of a Linear Stencil Can be Viewed as Multiplying Two Polynomials (Computing Convolution)

Stencils and Polynomial Multiplication

$$A_t(x) = v_0 + v_1 x + v_2 x^2 + v_3 x^3 + v_4 x^4 + v_5 x^5 + v_6 x^6 + v_7 x^7$$

a_t	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Stencil	s_{-1}	s_0	s_1

a_{t+1}	$s_{-1}v_7 + s_0 v_0 + s_1 v_1$	$s_{-1}v_0 + s_0 v_1 + s_1 v_2$	$s_{-1}v_1 + s_0 v_2 + s_1 v_3$	$s_{-1}v_2 + s_0 v_3 + s_1 v_4$	$s_{-1}v_3 + s_0 v_4 + s_1 v_5$	$s_{-1}v_4 + s_0 v_5 + s_1 v_6$	$s_{-1}v_5 + s_0 v_6 + s_1 v_7$	$s_{-1}v_6 + s_0 v_7 + s_1 v_0$

Stencils and Polynomial Multiplication

$$A_t(x) = v_0 + v_1 x + v_2 x^2 + v_3 x^3 + v_4 x^4 + v_5 x^5 + v_6 x^6 + v_7 x^7$$

a_t	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Stencil	s_{-1}	s_0	s_1

a_{t+1}	$s_{-1}v_7 + s_0 v_0 + s_1 v_1$	$s_{-1}v_0 + s_0 v_1 + s_1 v_2$	$s_{-1}v_1 + s_0 v_2 + s_1 v_3$	$s_{-1}v_2 + s_0 v_3 + s_1 v_4$	$s_{-1}v_3 + s_0 v_4 + s_1 v_5$	$s_{-1}v_4 + s_0 v_5 + s_1 v_6$	$s_{-1}v_5 + s_0 v_6 + s_1 v_7$	$s_{-1}v_6 + s_0 v_7 + s_1 v_0$

Stencils and Polynomial Multiplication

$$A_t(x) = v_0 + v_1x + v_2x^2 + v_3x^3 + v_4x^4 + v_5x^5 + v_6x^6 + v_7x^7$$

a_t							
v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

$$S(x) = s_1x^{-1} + s_0 + s_{-1}x$$

Flipped Stencil

s_1	s_0	s_{-1}

a_{t+1}							
$s_{-1}v_7 + s_0v_0 + s_1v_1$	$s_{-1}v_0 + s_0v_1 + s_1v_2$	$s_{-1}v_1 + s_0v_2 + s_1v_3$	$s_{-1}v_2 + s_0v_3 + s_1v_4$	$s_{-1}v_3 + s_0v_4 + s_1v_5$	$s_{-1}v_4 + s_0v_5 + s_1v_6$	$s_{-1}v_5 + s_0v_6 + s_1v_7$	$s_{-1}v_6 + s_0v_7 + s_1v_0$

Stencils and Polynomial Multiplication

$$A_t(x) = v_0 + v_1 x + v_2 x^2 + v_3 x^3 + v_4 x^4 + v_5 x^5 + v_6 x^6 + v_7 x^7$$

a_t

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

$$S(x) = s_1 x^7 + s_0 + s_{-1} x$$ (wrap around)

Flipped Stencil

s_1	s_0	s_{-1}

a_{t+1}

$s_{-1}v_7 + s_0 v_0 + s_1 v_1$	$s_{-1}v_0 + s_0 v_1 + s_1 v_2$	$s_{-1}v_1 + s_0 v_2 + s_1 v_3$	$s_{-1}v_2 + s_0 v_3 + s_1 v_4$	$s_{-1}v_3 + s_0 v_4 + s_1 v_5$	$s_{-1}v_4 + s_0 v_5 + s_1 v_6$	$s_{-1}v_5 + s_0 v_6 + s_1 v_7$	$s_{-1}v_6 + s_0 v_7 + s_1 v_0$

Stencils and Polynomial Multiplication

$$A_t(x) = v_0 + v_1 x + v_2 x^2 + v_3 x^3 + v_4 x^4 + v_5 x^5 + v_6 x^6 + v_7 x^7$$

a_t

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

$$S(x) = s_1 x^7 + s_0 + s_{-1} x$$ (wrap around)

Flipped Stencil

s_1	s_0	s_{-1}

a_{t+1}

$s_{-1}v_7 + s_0 v_0 + s_1 v_1$	$s_{-1}v_0 + s_0 v_1 + s_1 v_2$	$s_{-1}v_1 + s_0 v_2 + s_1 v_3$	$s_{-1}v_2 + s_0 v_3 + s_1 v_4$	$s_{-1}v_3 + s_0 v_4 + s_1 v_5$	$s_{-1}v_4 + s_0 v_5 + s_1 v_6$	$s_{-1}v_5 + s_0 v_6 + s_1 v_7$	$s_{-1}v_6 + s_0 v_7 + s_1 v_0$

$S(x)\ A_t(x)$

$$= (s_0 v_0) + \begin{pmatrix} s_{-1}v_0 \\ +s_0 v_1 \end{pmatrix} x + \begin{pmatrix} s_{-1}v_1 \\ +s_0 v_2 \end{pmatrix} x^2 + \begin{pmatrix} s_{-1}v_2 \\ +s_0 v_3 \end{pmatrix} x^3 + \begin{pmatrix} s_{-1}v_3 \\ +s_0 v_4 \end{pmatrix} x^4 + \begin{pmatrix} s_{-1}v_4 \\ +s_0 v_5 \end{pmatrix} x^5 + \begin{pmatrix} s_{-1}v_5 \\ +s_0 v_6 \end{pmatrix} x^6 + \begin{pmatrix} s_{-1}v_6 \\ +s_0 v_7 \\ +s_1 v_0 \end{pmatrix} x^7$$

$$+ \begin{pmatrix} s_{-1}v_7 \\ +s_1 v_1 \end{pmatrix} x^8 + (s_1 v_2)\ x^9 + (s_1 v_3)\ x^{10} + (s_1 v_4)\ x^{11} + (s_1 v_5)\ x^{12} + (s_1 v_6)\ x^{13} + (s_1 v_7)\ x^{14}$$

Stencils and Polynomial Multiplication

$$A_t(x) = v_0 + v_1 x + v_2 x^2 + v_3 x^3 + v_4 x^4 + v_5 x^5 + v_6 x^6 + v_7 x^7$$

a_t	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

$$S(x) = s_1 x^7 + s_0 + s_{-1} x$$ (wrap around)

Flipped Stencil	s_1	s_0	s_{-1}

a_{t+1}	$s_{-1}v_7 + s_0v_0 + s_1v_1$	$s_{-1}v_0 + s_0v_1 + s_1v_2$	$s_{-1}v_1 + s_0v_2 + s_1v_3$	$s_{-1}v_2 + s_0v_3 + s_1v_4$	$s_{-1}v_3 + s_0v_4 + s_1v_5$	$s_{-1}v_4 + s_0v_5 + s_1v_6$	$s_{-1}v_5 + s_0v_6 + s_1v_7$	$s_{-1}v_6 + s_0v_7 + s_1v_0$

$S(x)\, A_t(x)$

$$= (s_0v_0) + \begin{pmatrix} s_{-1}v_0 \\ +s_0v_1 \end{pmatrix} x + \begin{pmatrix} s_{-1}v_1 \\ +s_0v_2 \end{pmatrix} x^2 + \begin{pmatrix} s_{-1}v_2 \\ +s_0v_3 \end{pmatrix} x^3 + \begin{pmatrix} s_{-1}v_3 \\ +s_0v_4 \end{pmatrix} x^4 + \begin{pmatrix} s_{-1}v_4 \\ +s_0v_5 \end{pmatrix} x^5 + \begin{pmatrix} s_{-1}v_5 \\ +s_0v_6 \end{pmatrix} x^6 + \begin{pmatrix} s_{-1}v_6 \\ +s_0v_7 \\ +s_1v_0 \end{pmatrix} x^7$$

$$+ \begin{pmatrix} s_{-1}v_7 \\ +s_1v_1 \end{pmatrix} x^8 + (s_1v_2)\, x^9 + (s_1v_3)\, x^{10} + (s_1v_4)\, x^{11} + (s_1v_5)\, x^{12} + (s_1v_6)\, x^{13} + (s_1v_7)\, x^{14}$$

$$\begin{pmatrix} s_{-1}v_7 \\ +s_0v_0 \\ +s_1v_1 \end{pmatrix} + \begin{pmatrix} s_{-1}v_0 \\ +s_0v_1 \\ +s_1v_2 \end{pmatrix} x + \begin{pmatrix} s_{-1}v_1 \\ +s_0v_2 \\ +s_1v_3 \end{pmatrix} x^2 + \begin{pmatrix} s_{-1}v_2 \\ +s_0v_3 \\ +s_1v_4 \end{pmatrix} x^3 + \begin{pmatrix} s_{-1}v_3 \\ +s_0v_4 \\ +s_1v_5 \end{pmatrix} x^4 + \begin{pmatrix} s_{-1}v_4 \\ +s_0v_5 \\ +s_1v_6 \end{pmatrix} x^5 + \begin{pmatrix} s_{-1}v_5 \\ +s_0v_6 \\ +s_1v_7 \end{pmatrix} x^6 + \begin{pmatrix} s_{-1}v_6 \\ +s_0v_7 \\ +s_1v_0 \end{pmatrix} x^7$$

$$= A_{t+1}(x)$$

Stencils and Polynomial Multiplication

$$A_T(x) = \overbrace{S(x)S(x)\ldots S(x)S(x)}^{T \text{ instances of } S(x)} A_0(x)$$

$$= \underbrace{\left(S(x)\right)^T}_{\text{This can be computed using repeated squaring}} A_0(x)$$

This can be computed using repeated squaring

($\mathrm{O}(\log T)$ polynomial products)

Stencils and Polynomial Multiplication

Computational complexity of multiplying two polynomials of degree bound N:

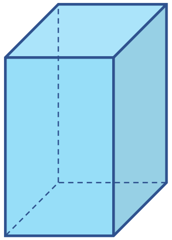
Classical Algorithm: $\Theta(N^2)$

Karatsuba's Algorithm: $\Theta(N^{\log_2 3}) \in O(N^{1.59})$

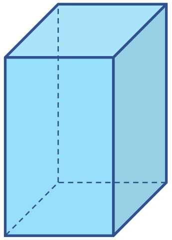
Toom-Cook Algorithms

$\boxed{\text{Fast Fourier Transform (FFT):} \quad \Theta(N \log N)}$

Algorithm for Periodic Grids

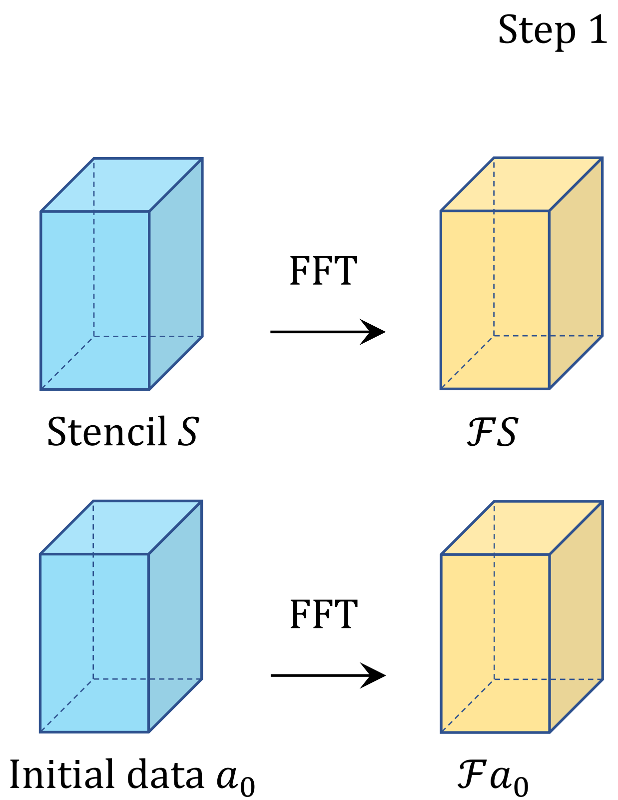


Stencil S

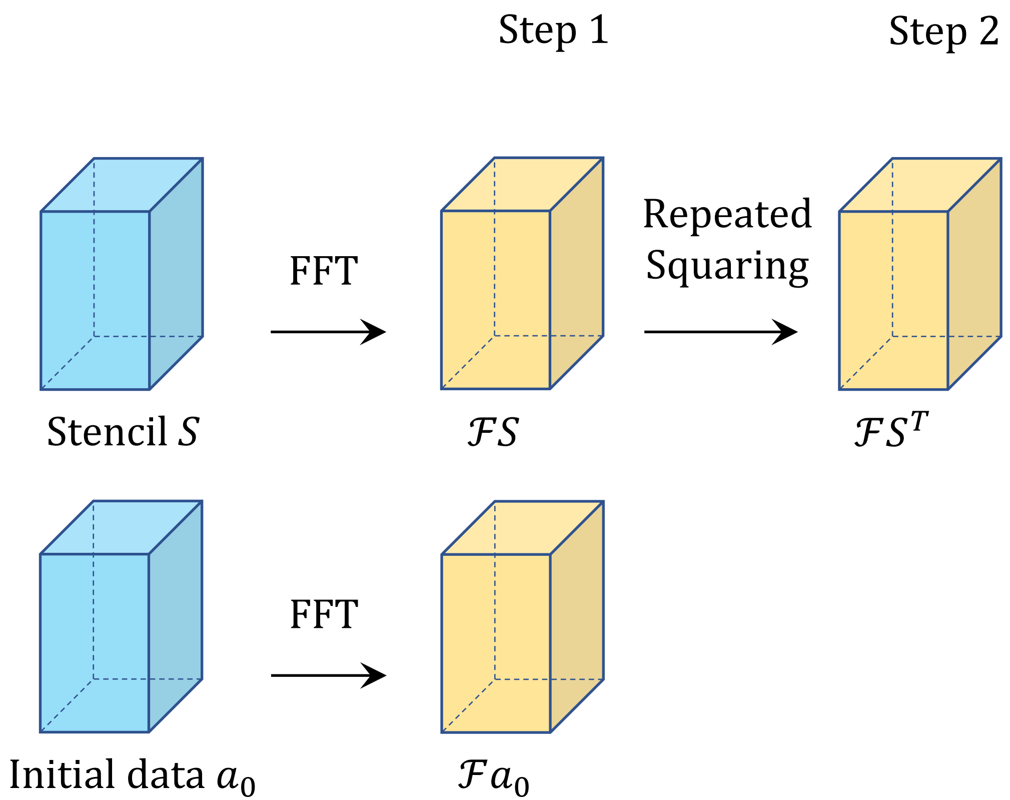


Initial data a_0

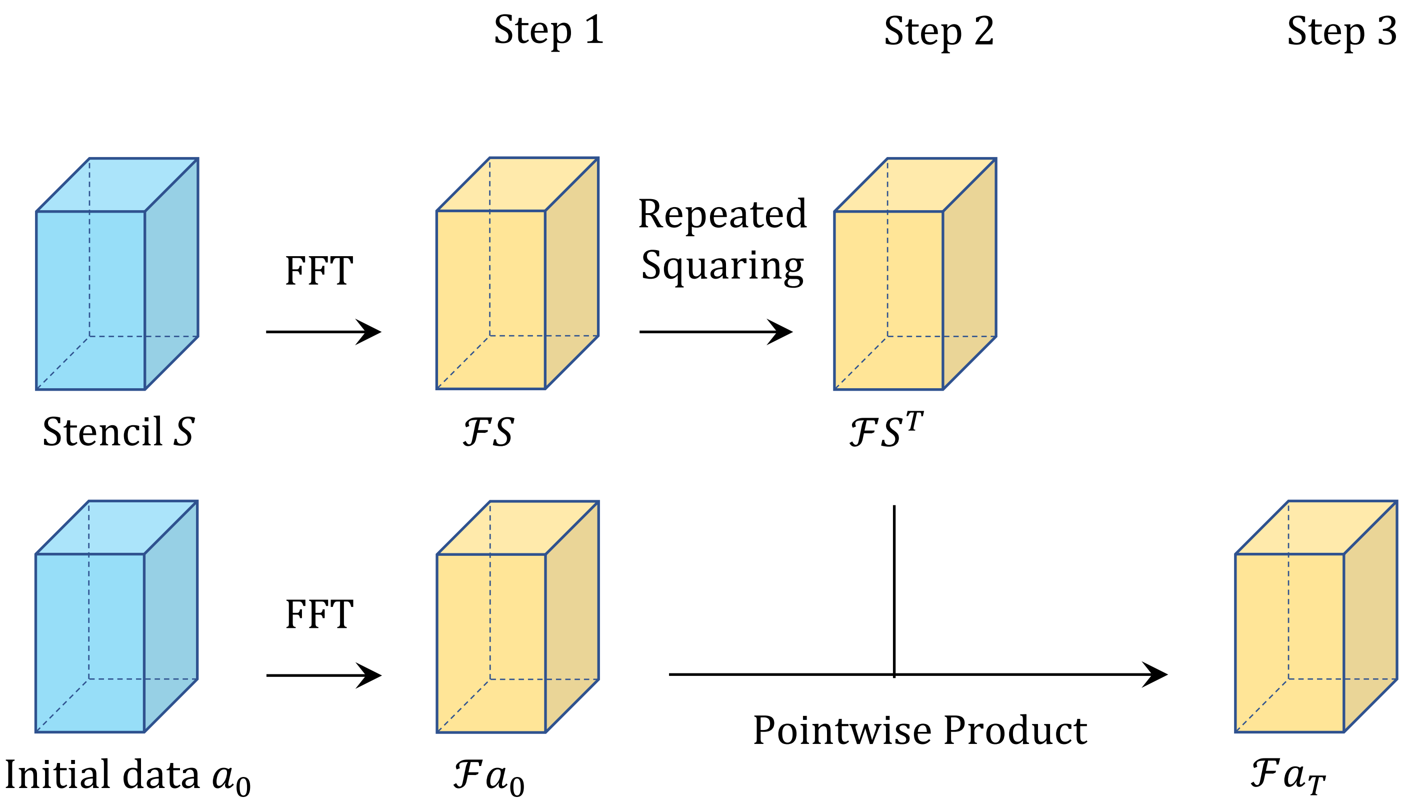
Algorithm for Periodic Grids



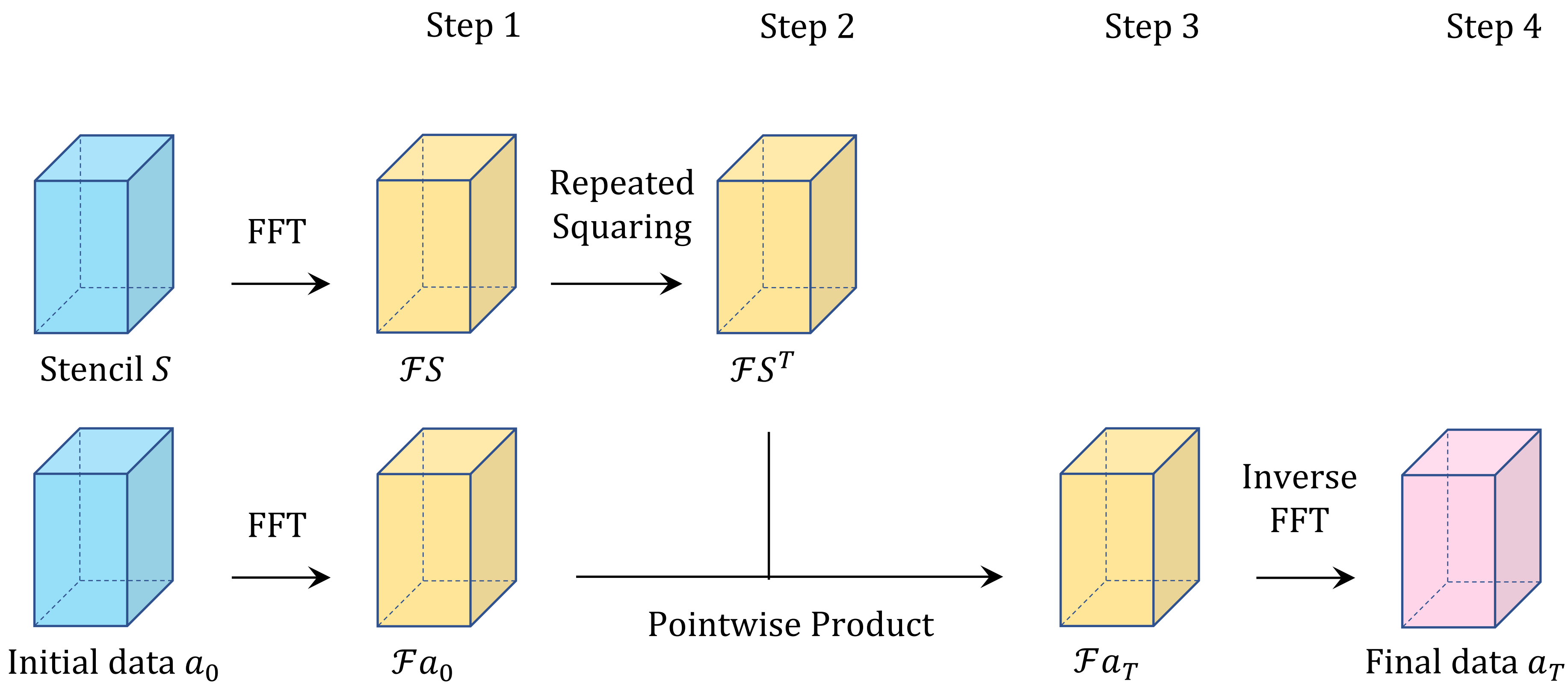
Algorithm for Periodic Grids



Algorithm for Periodic Grids



Algorithm for Periodic Grids



Computational Complexity: $\Theta(N \log(NT))$

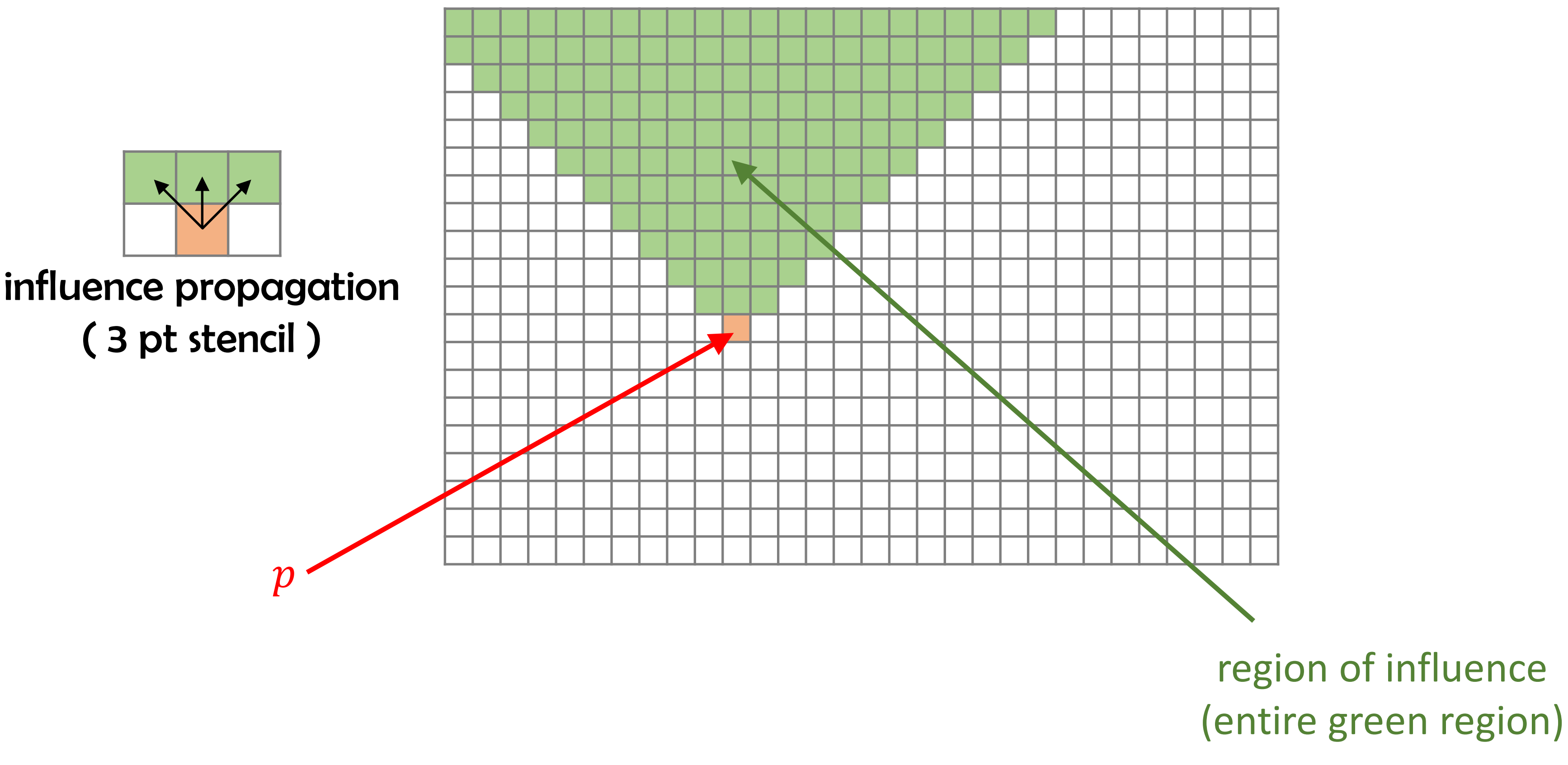
(for a d-dimensional $N^{\frac{1}{d}} \times N^{\frac{1}{d}} \times \cdots \times N^{\frac{1}{d}}$ grid with constant d)

Aperiodic Grids:

Aperiodic Algorithm is a Repeated Application of Periodic Algorithm via Divide-and-Conquer

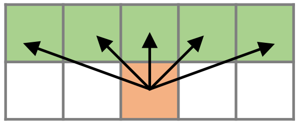
Algorithm for Aperiodic Grids

The region of influence of a grid point p consists of the set of grid points whose values can depend on the value at point p.

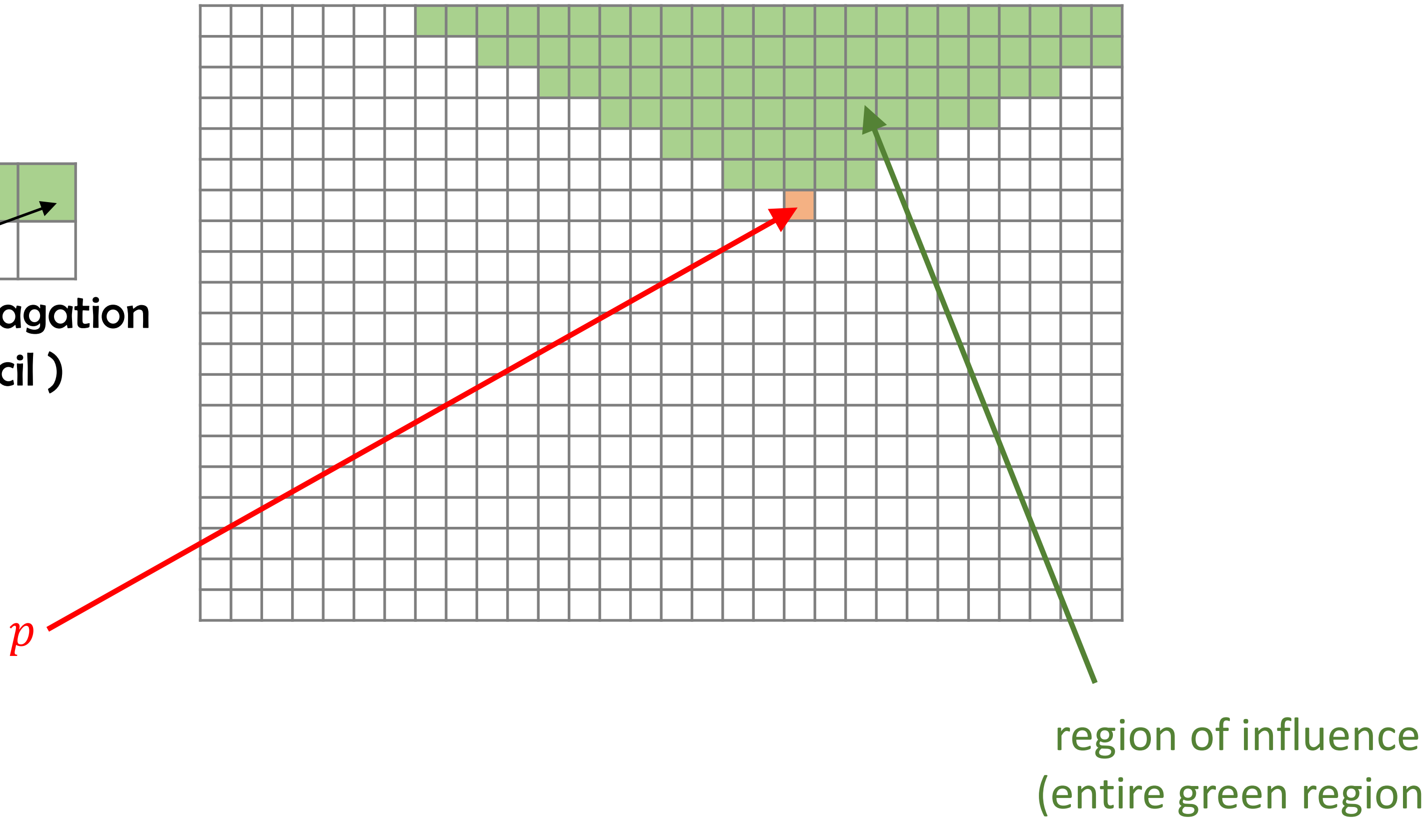


Algorithm for Aperiodic Grids

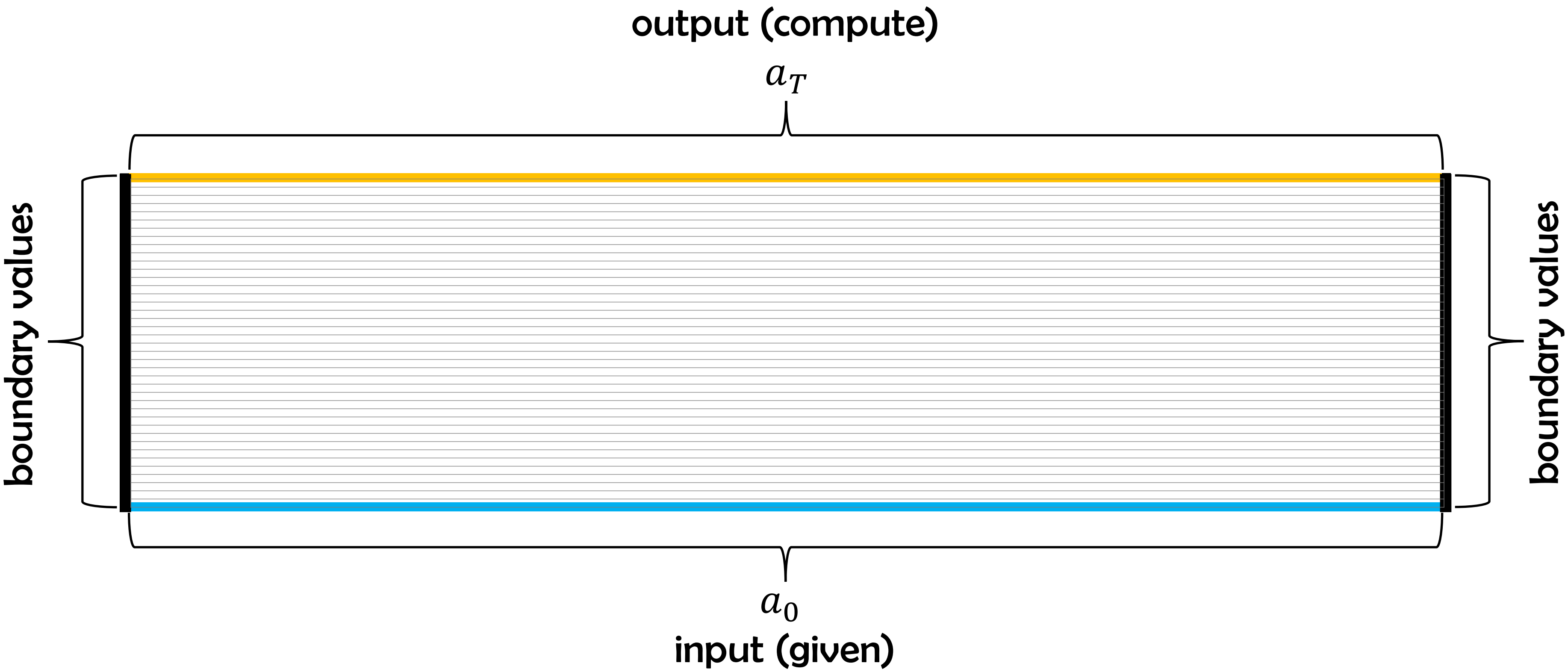
The region of influence of a grid point p consists of the set of grid points whose values can depend on the value at point p.



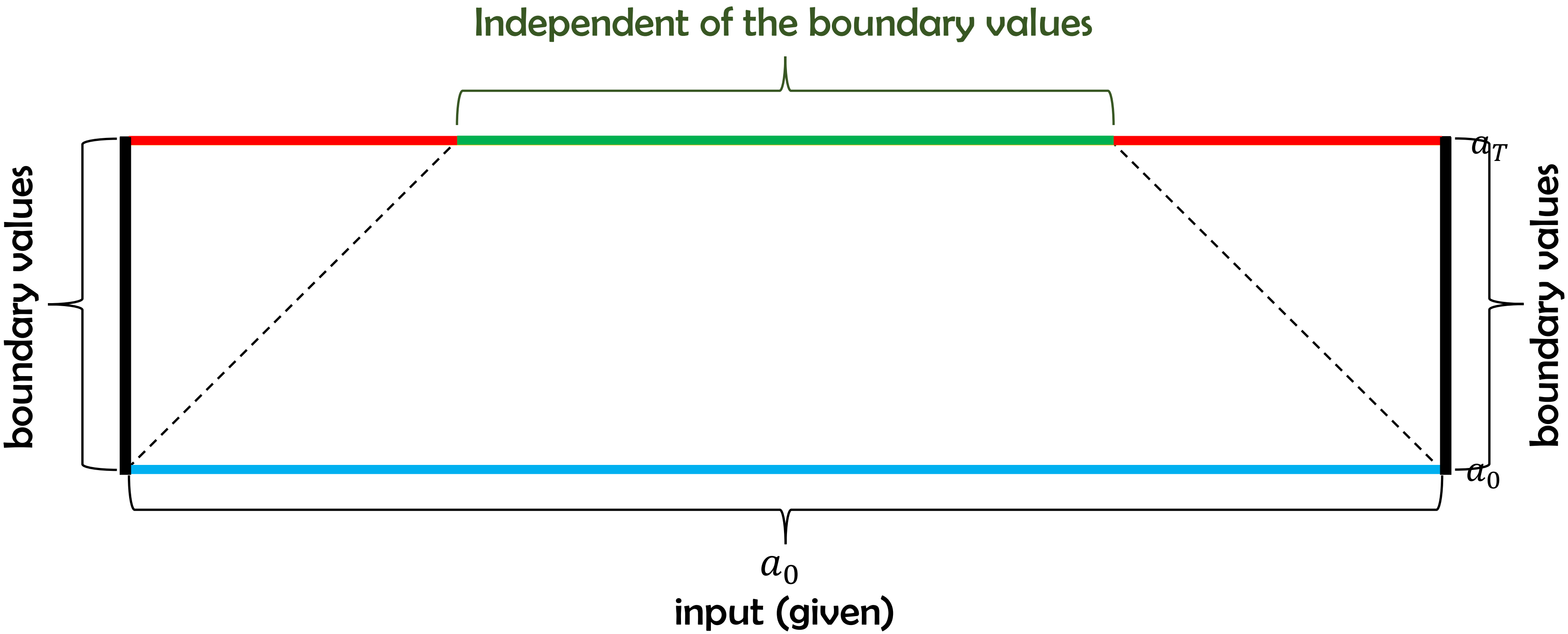
influence propagation
(5 pt stencil)



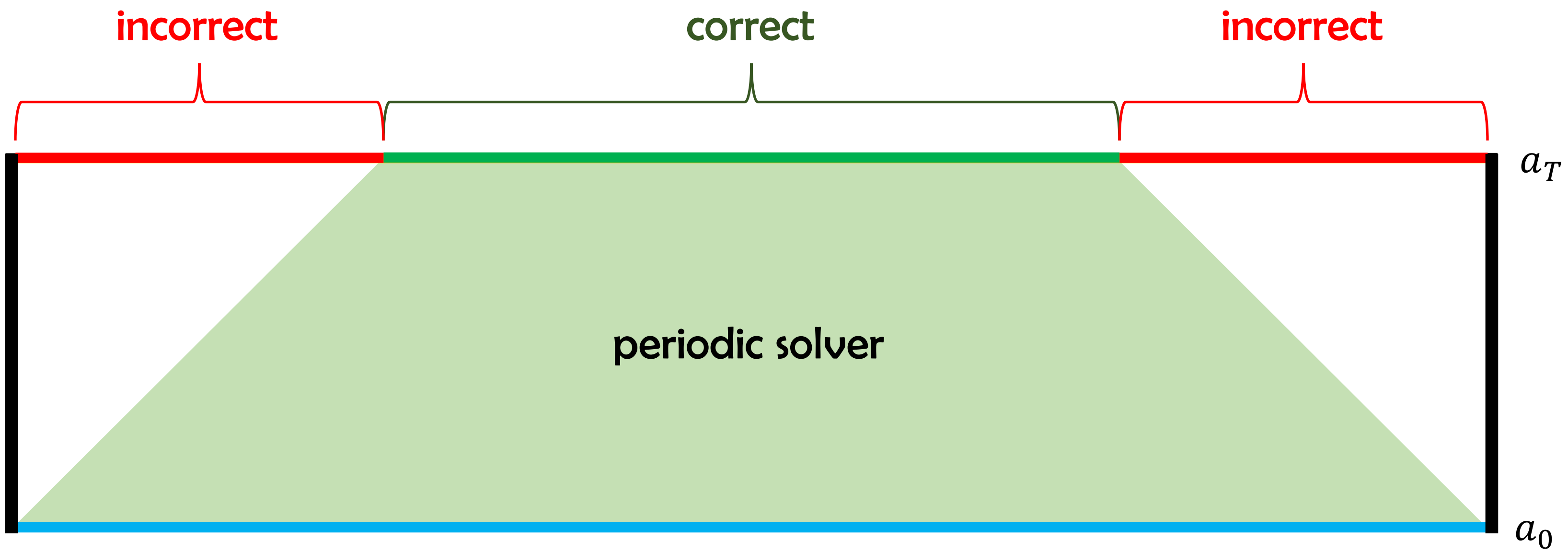
Algorithm for Aperiodic Grids



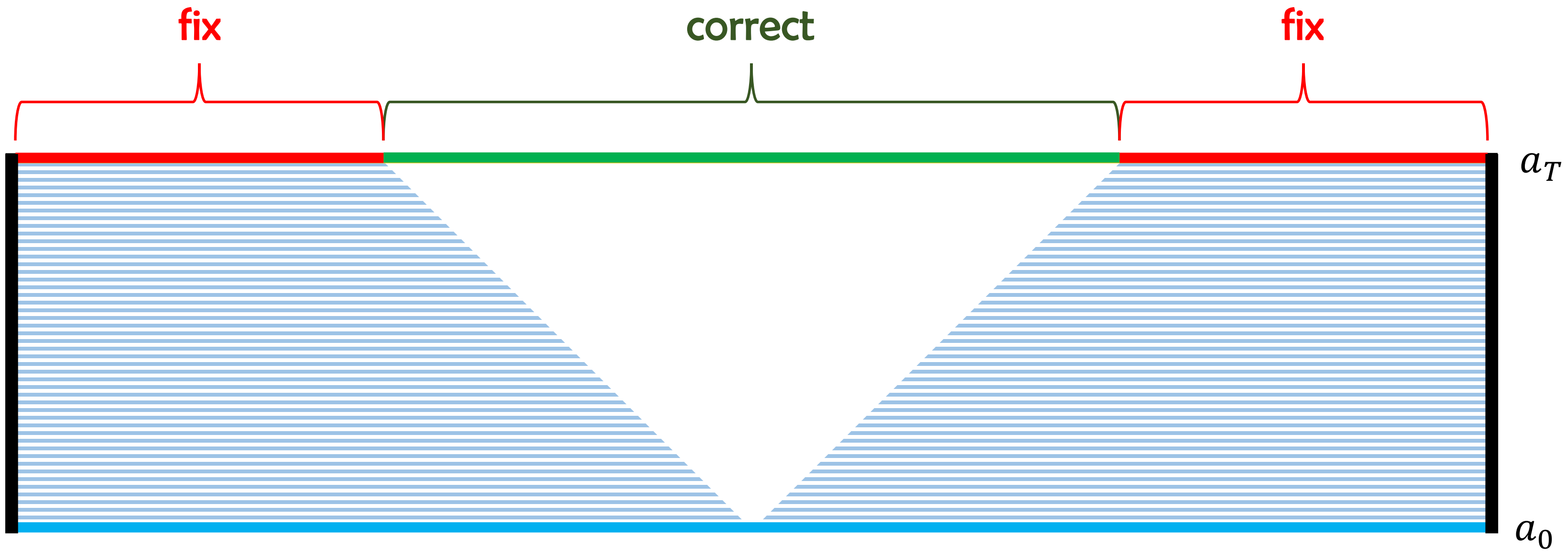
Algorithm for Aperiodic Grids



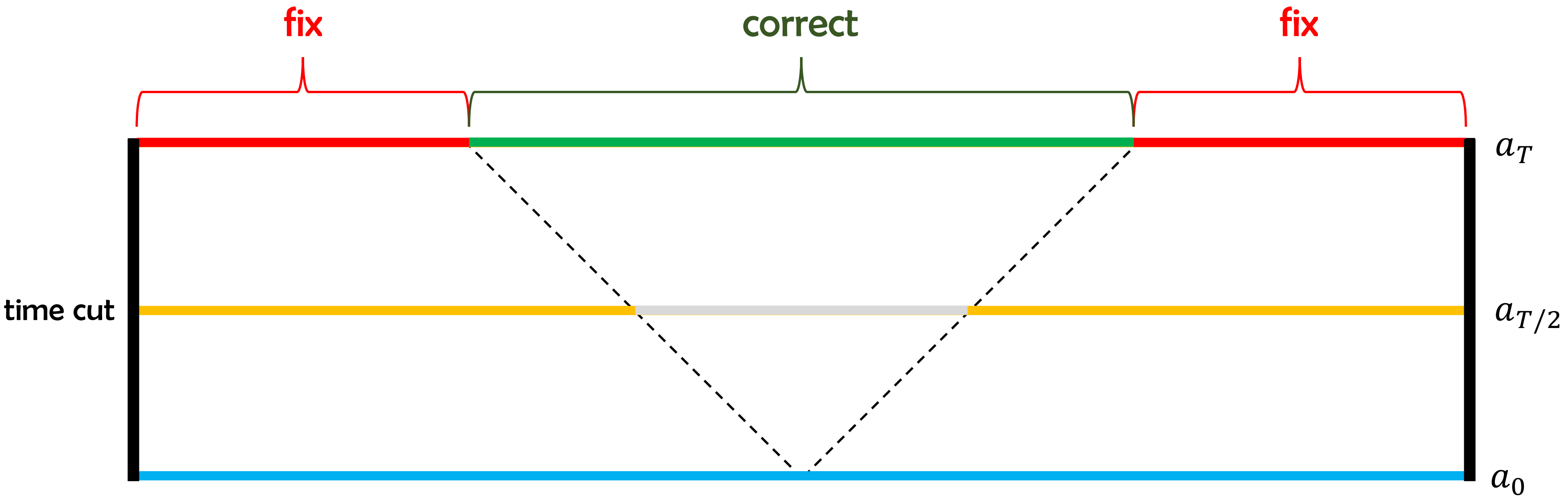
Algorithm for Aperiodic Grids



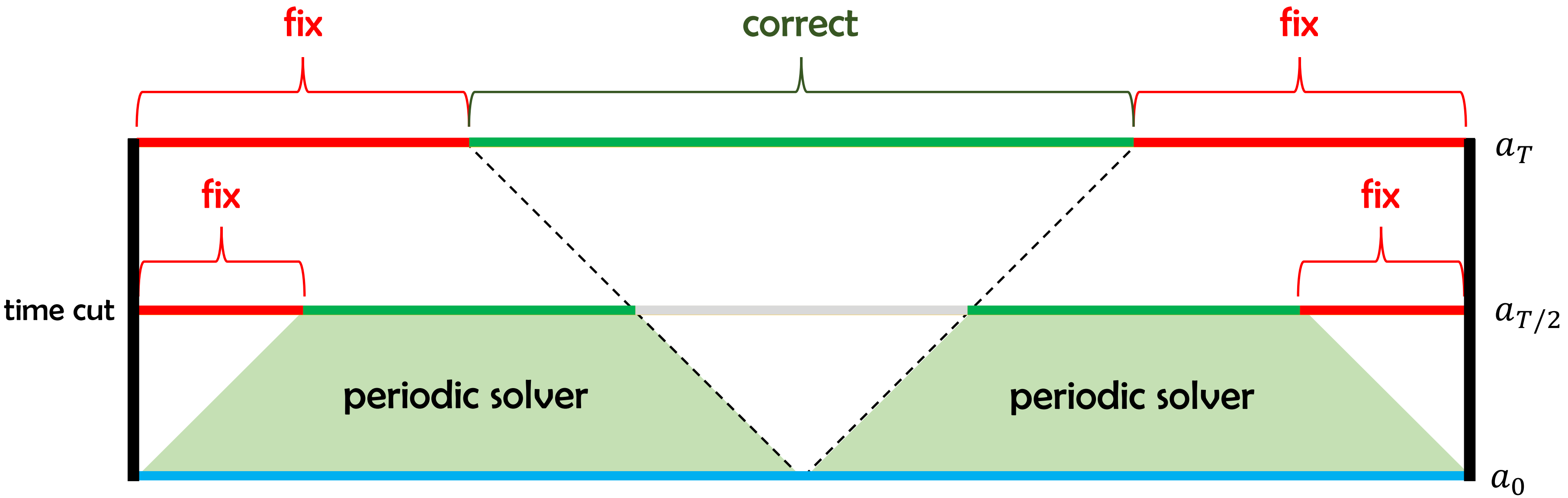
Algorithm for Aperiodic Grids



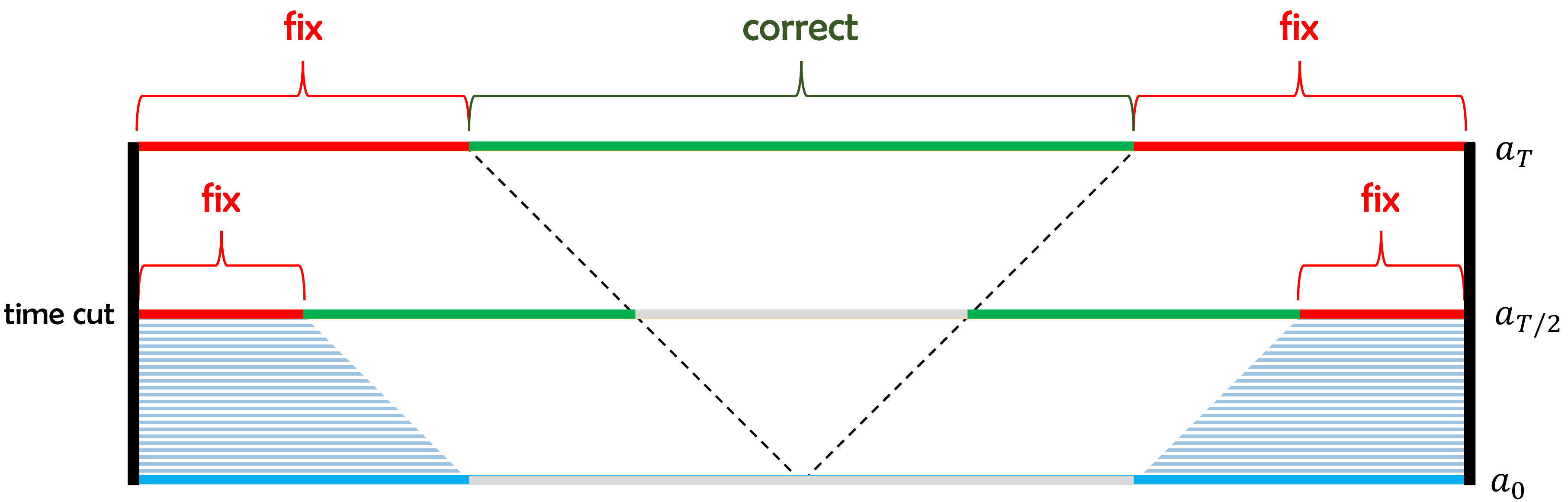
Algorithm for Aperiodic Grids

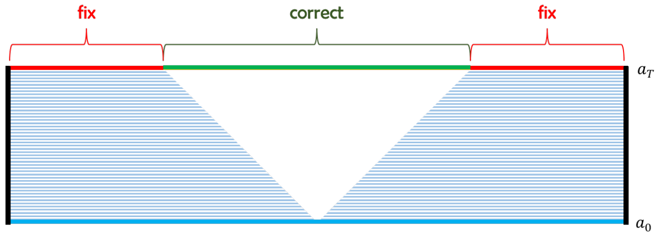


Algorithm for Aperiodic Grids



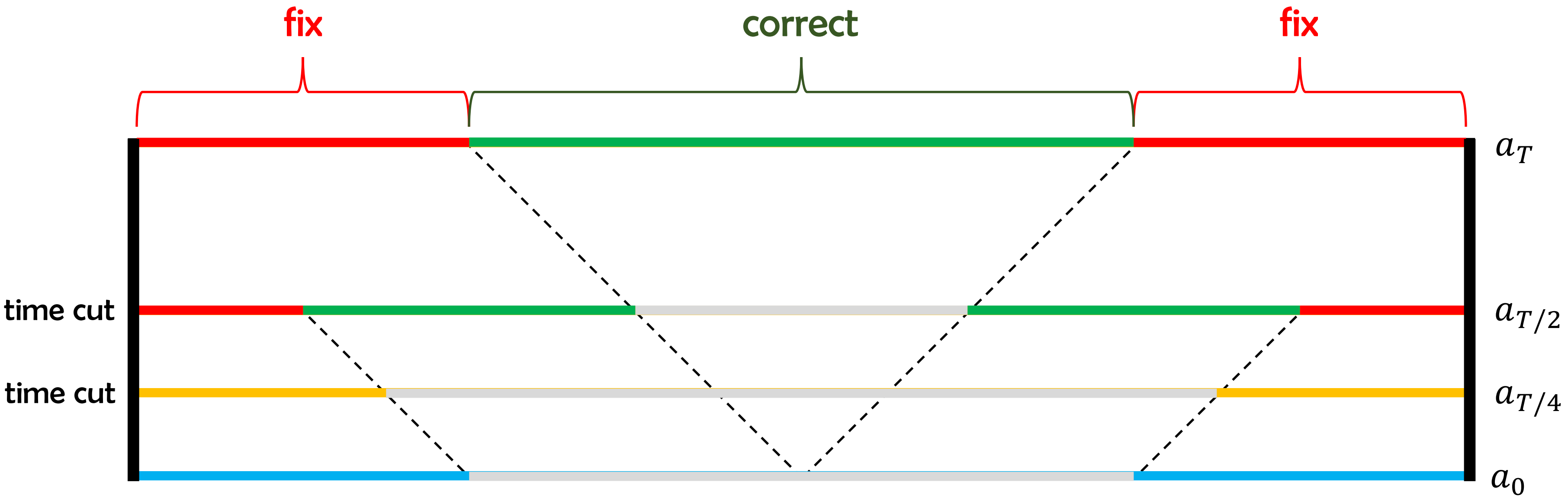
Algorithm for Aperiodic Grids



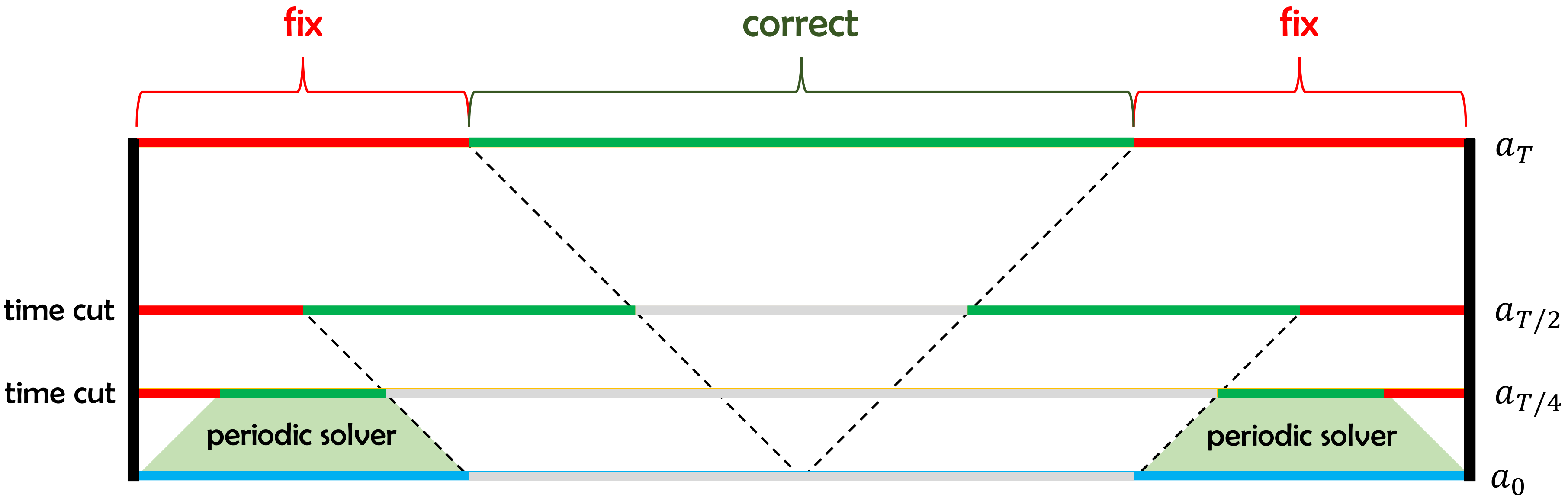


encountered before!

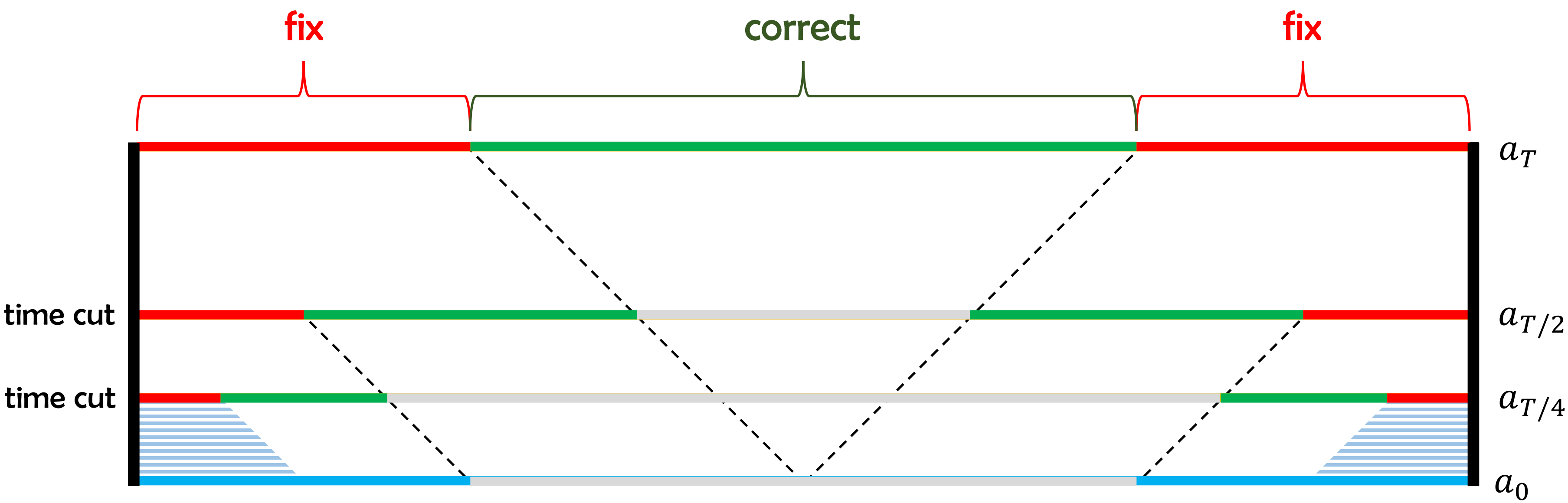
Algorithm for Aperiodic Grids

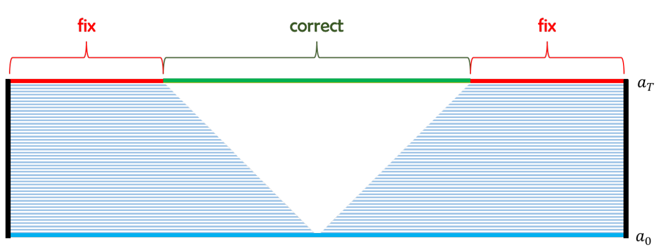


Algorithm for Aperiodic Grids



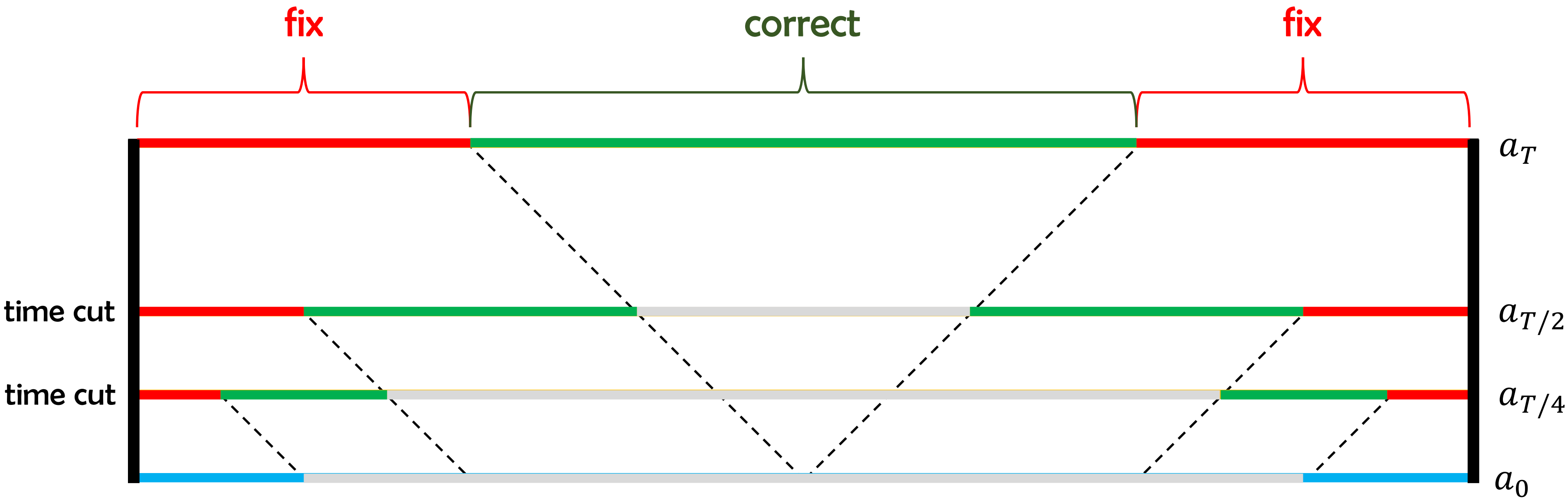
Algorithm for Aperiodic Grids



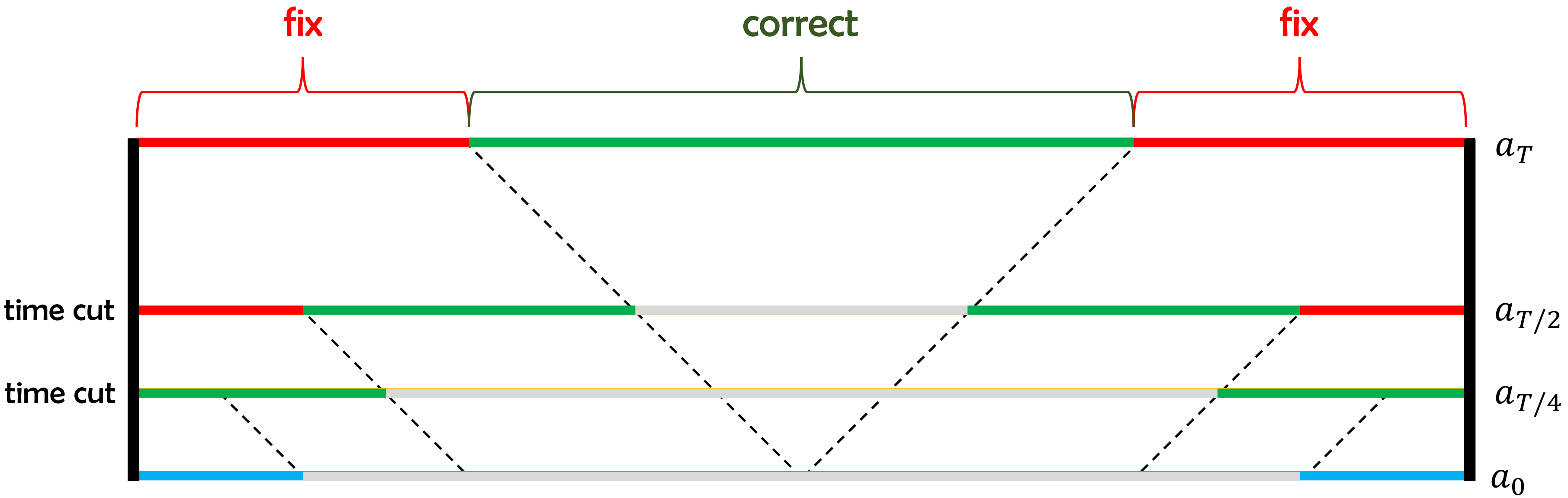


encountered before!

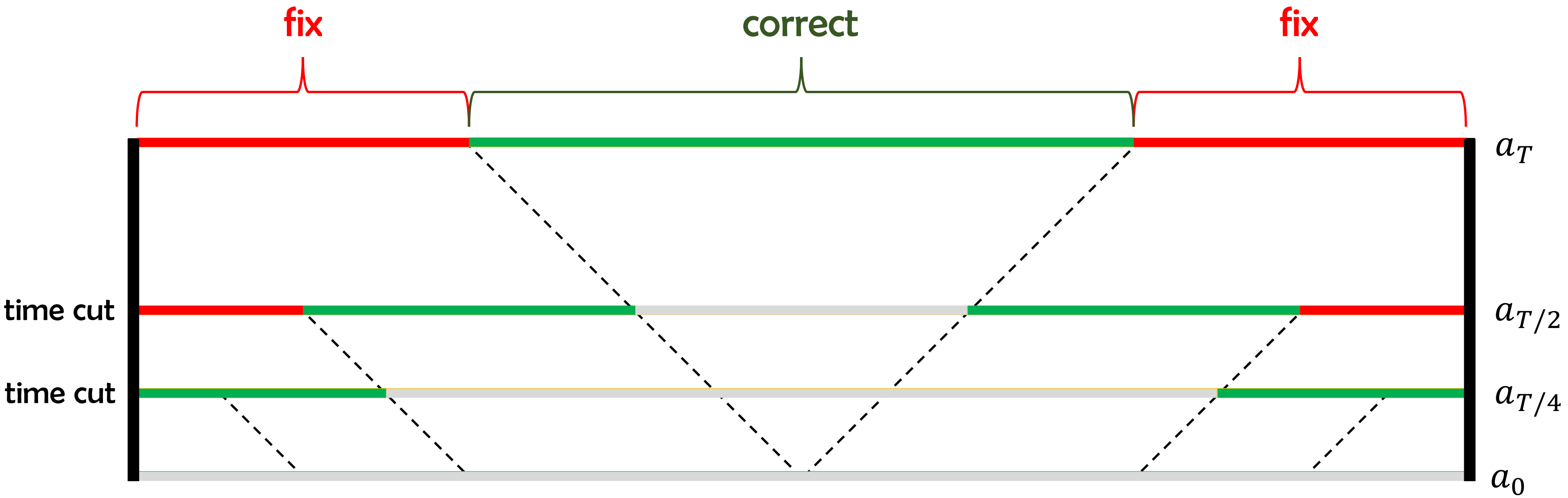
Algorithm for Aperiodic Grids



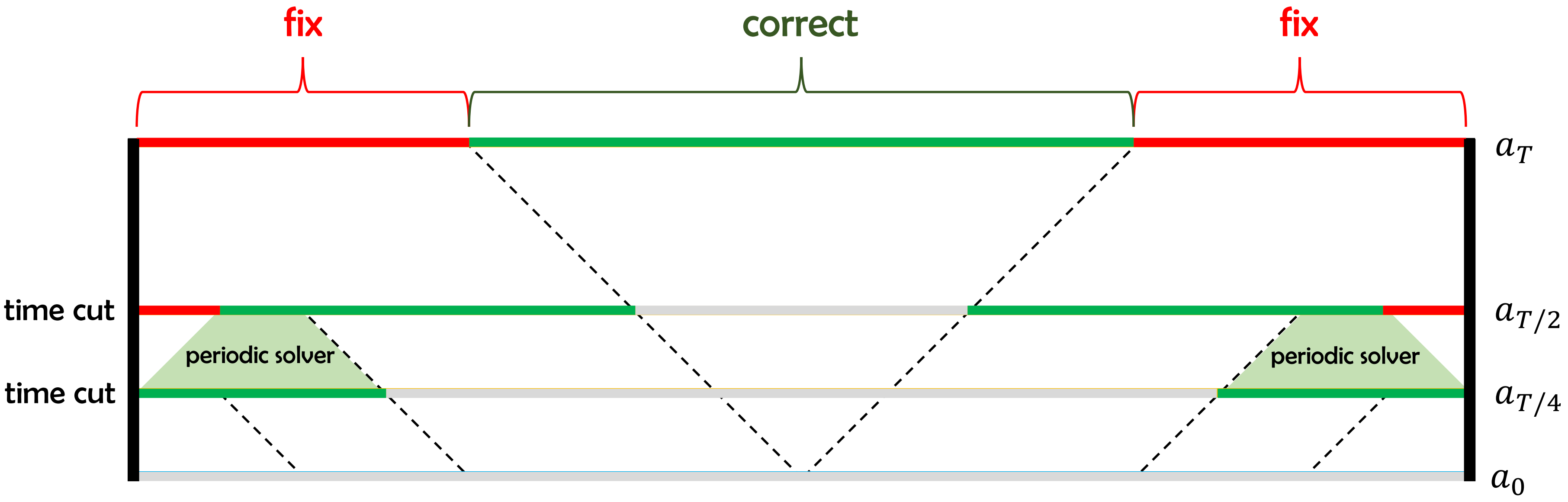
Algorithm for Aperiodic Grids



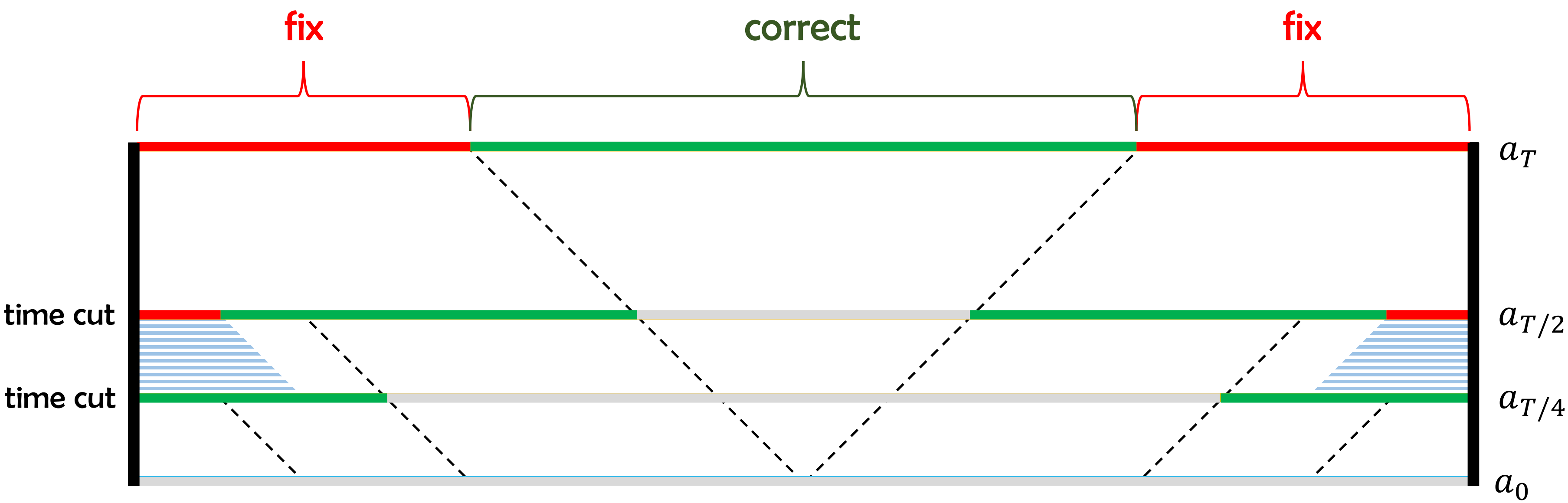
Algorithm for Aperiodic Grids

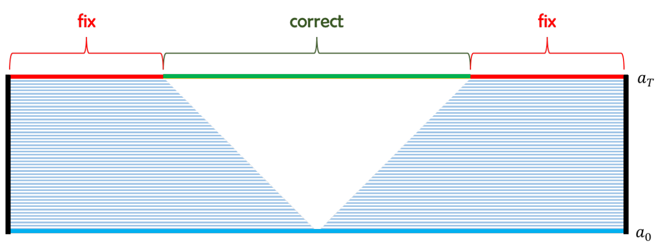


Algorithm for Aperiodic Grids



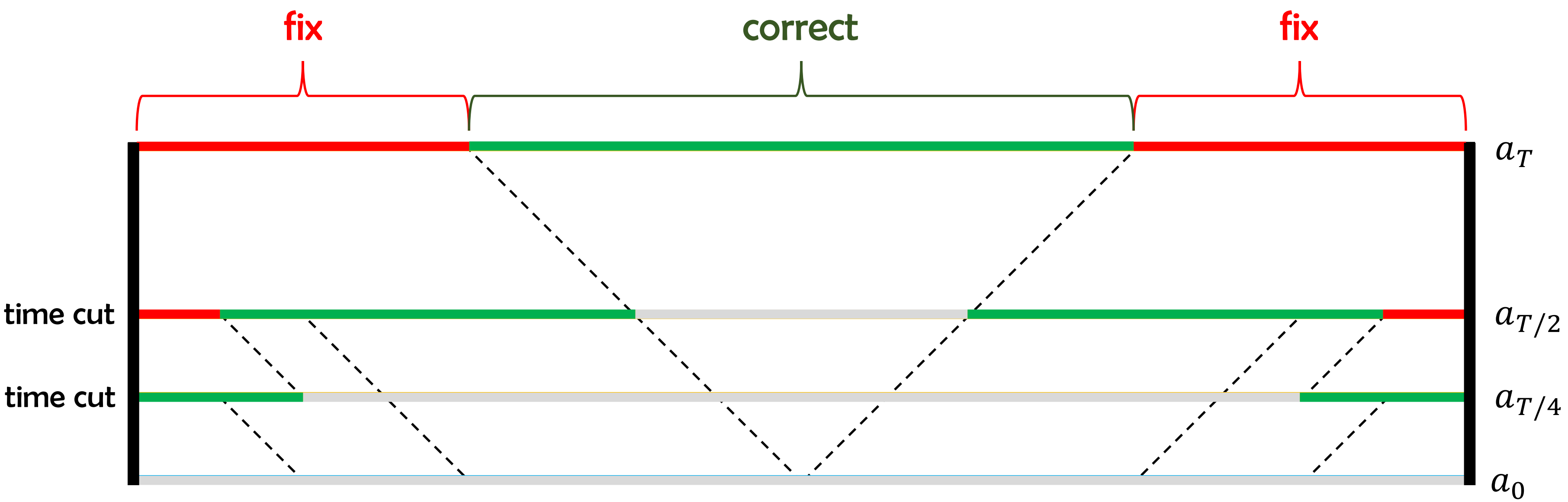
Algorithm for Aperiodic Grids



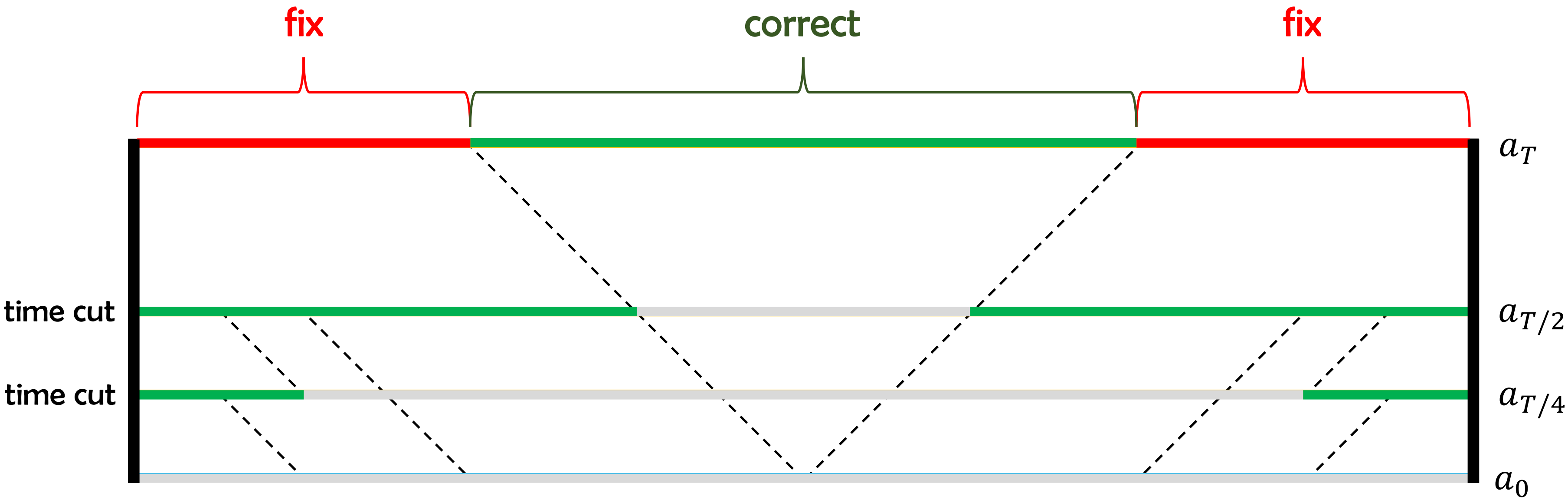


encountered before!

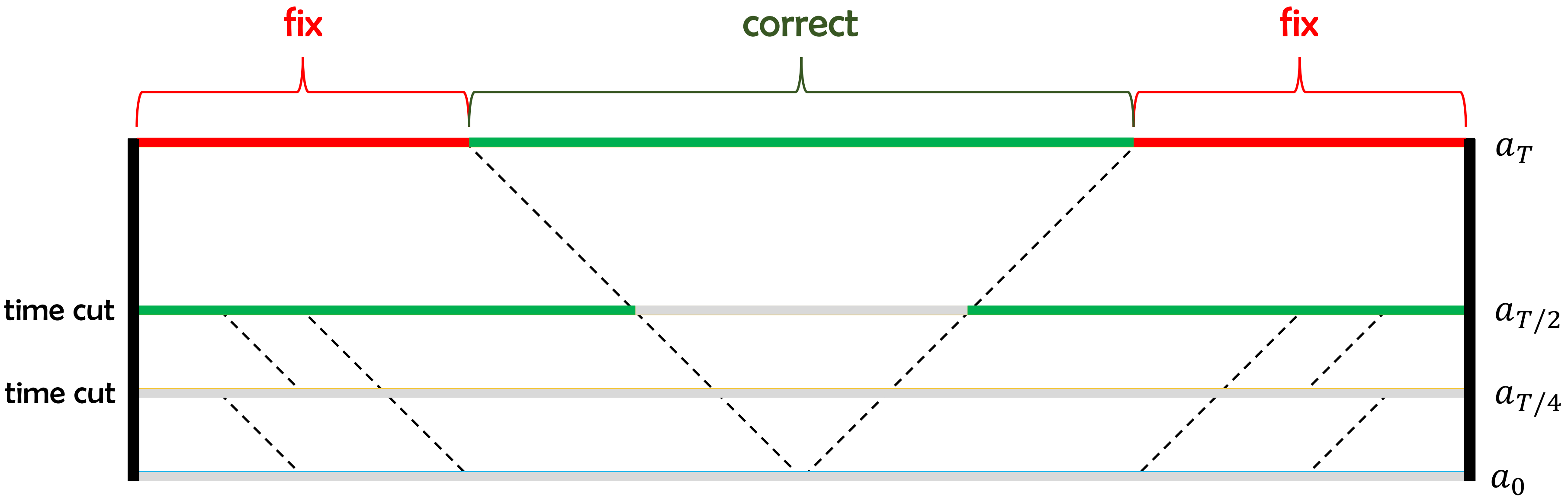
Algorithm for Aperiodic Grids



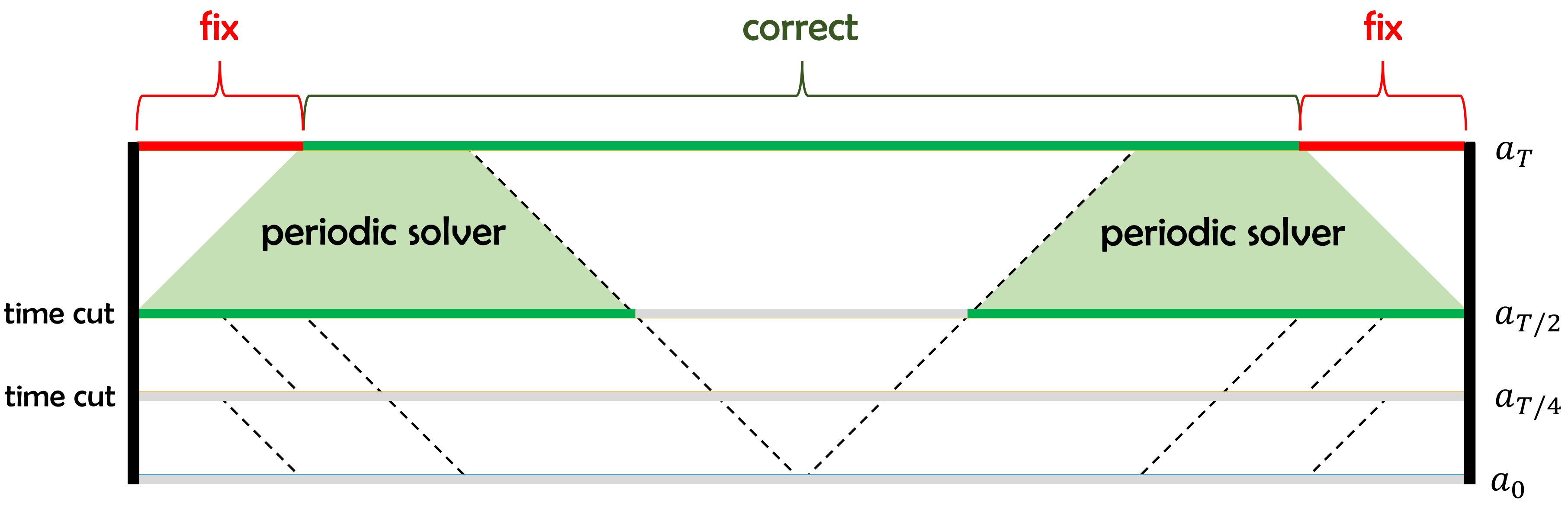
Algorithm for Aperiodic Grids



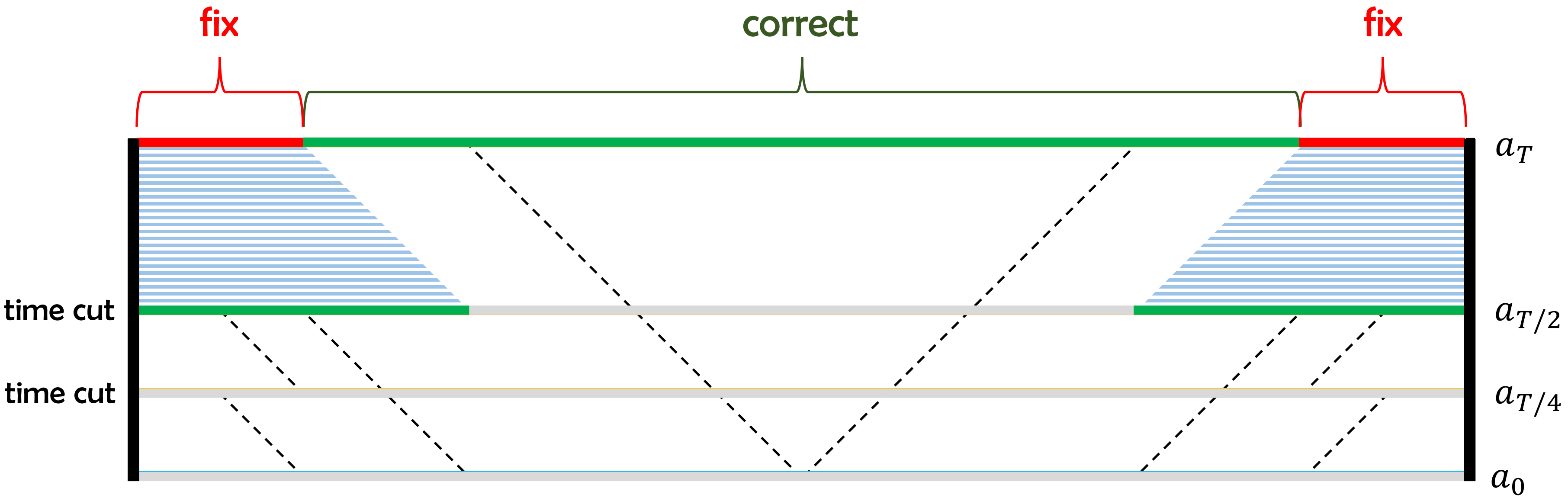
Algorithm for Aperiodic Grids

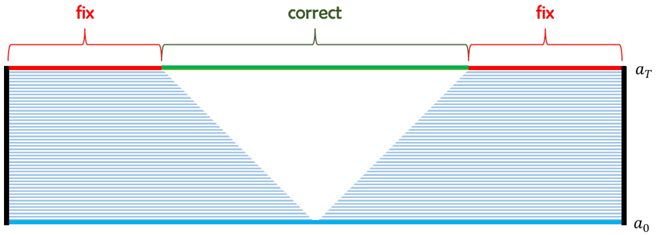


Algorithm for Aperiodic Grids



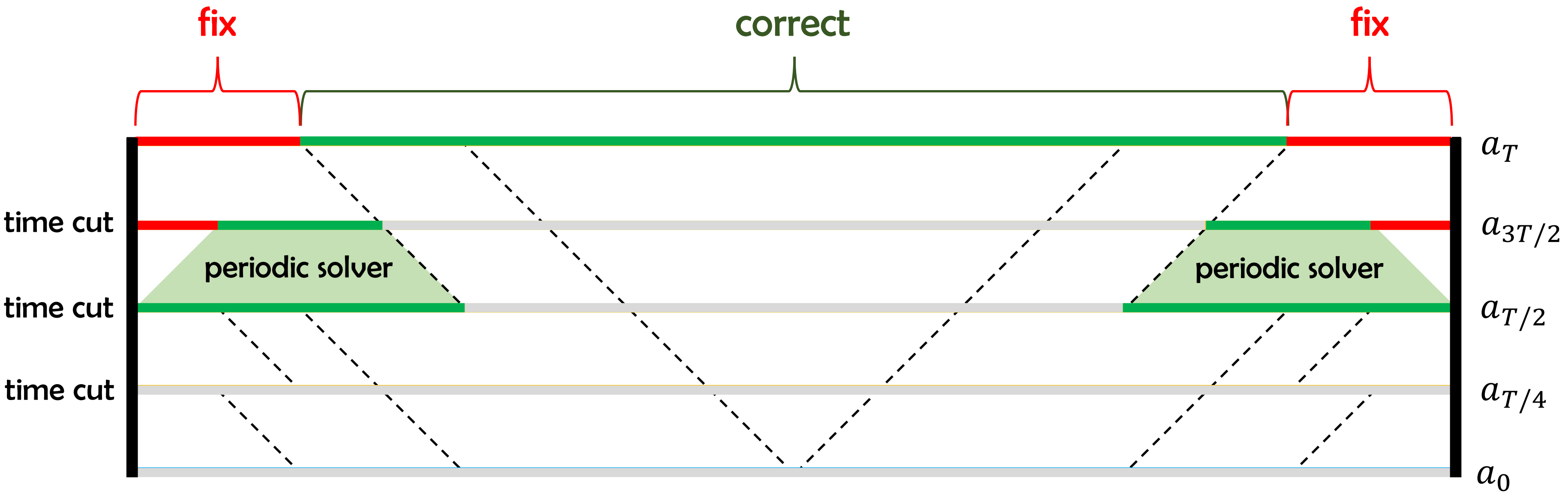
Algorithm for Aperiodic Grids



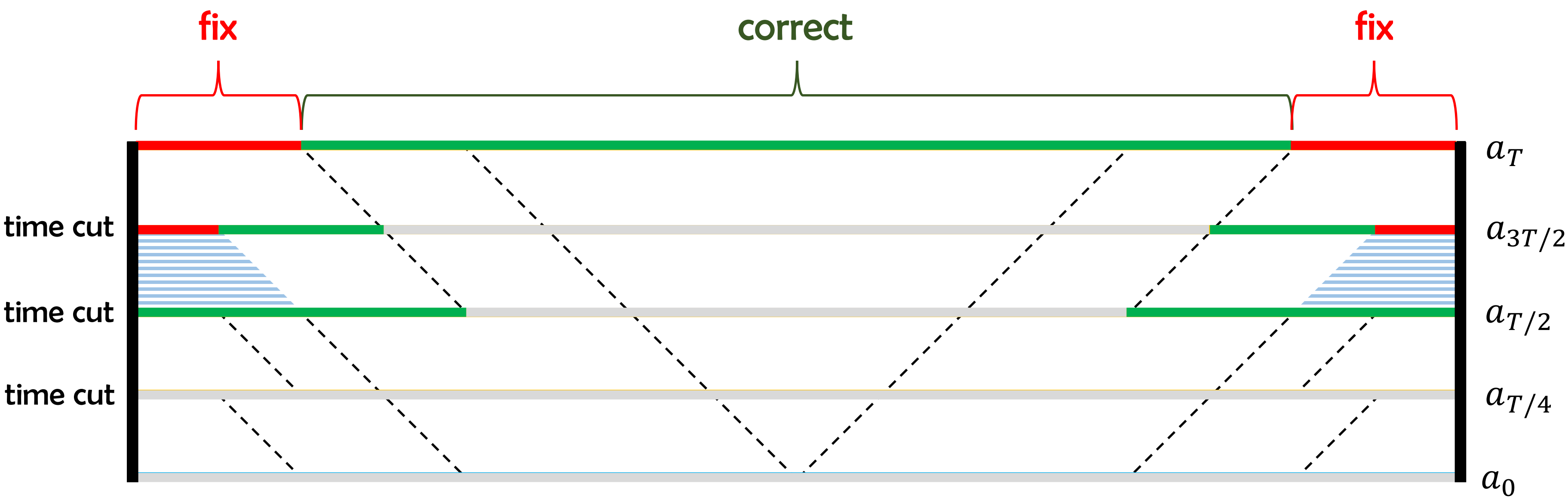


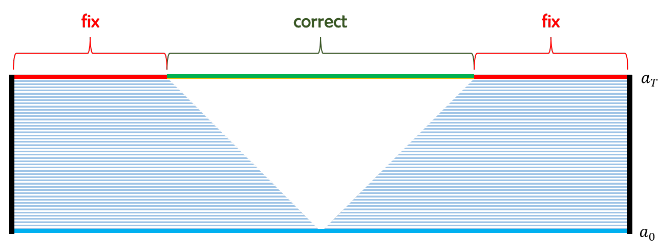
encountered before!

Algorithm for Aperiodic Grids



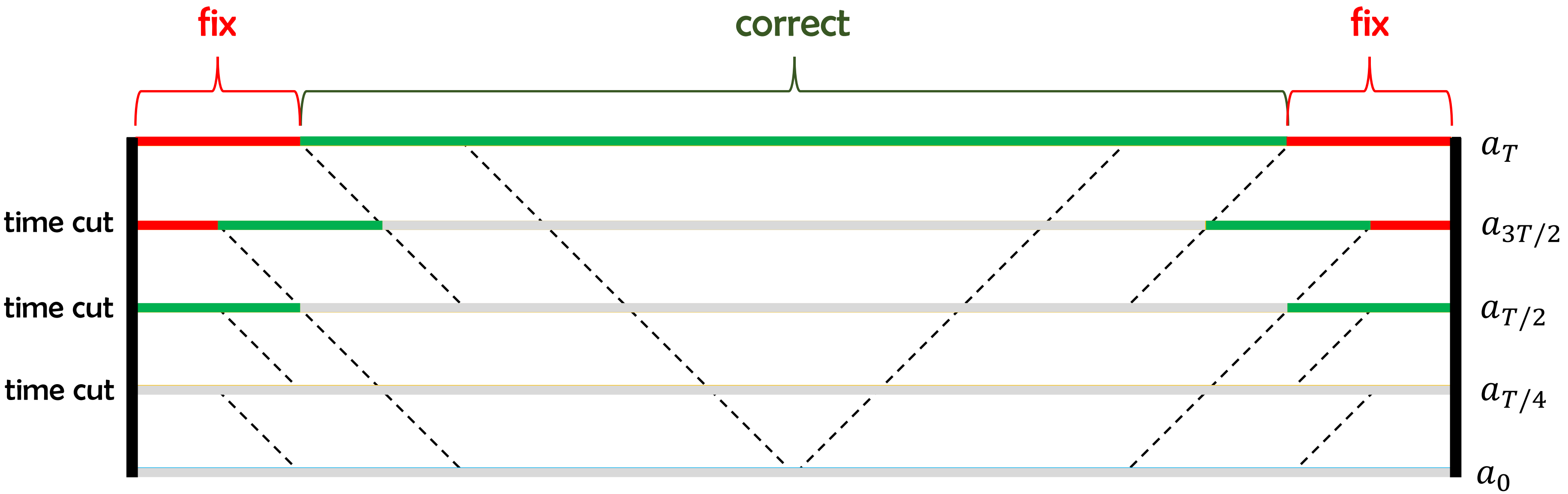
Algorithm for Aperiodic Grids



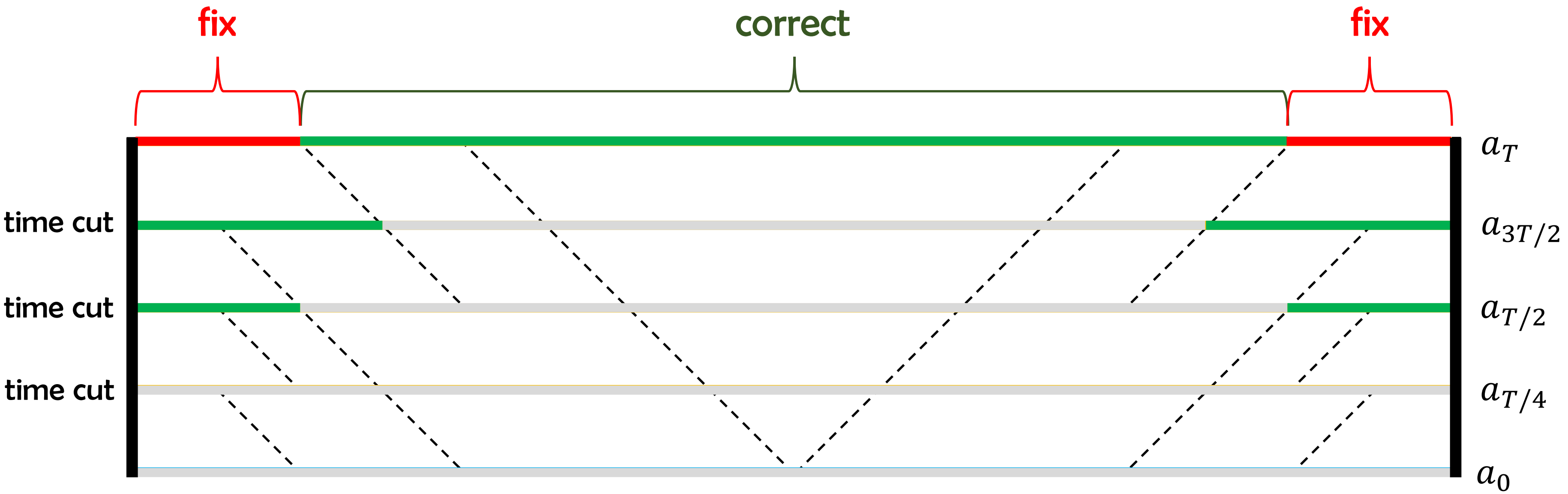


encountered before!

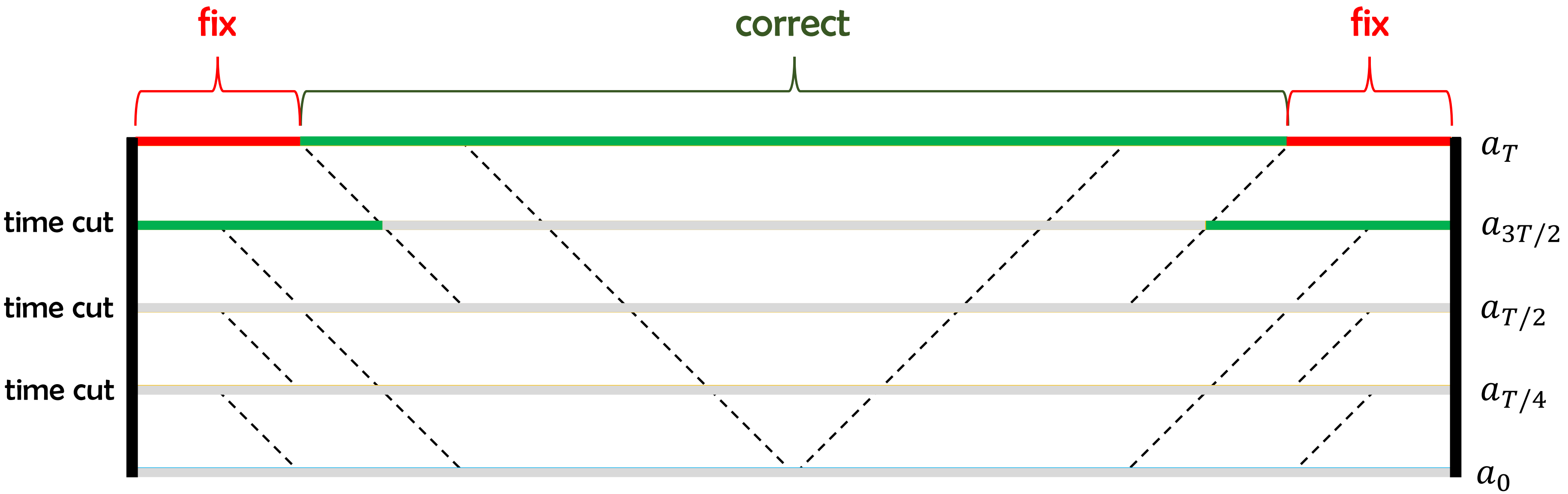
Algorithm for Aperiodic Grids



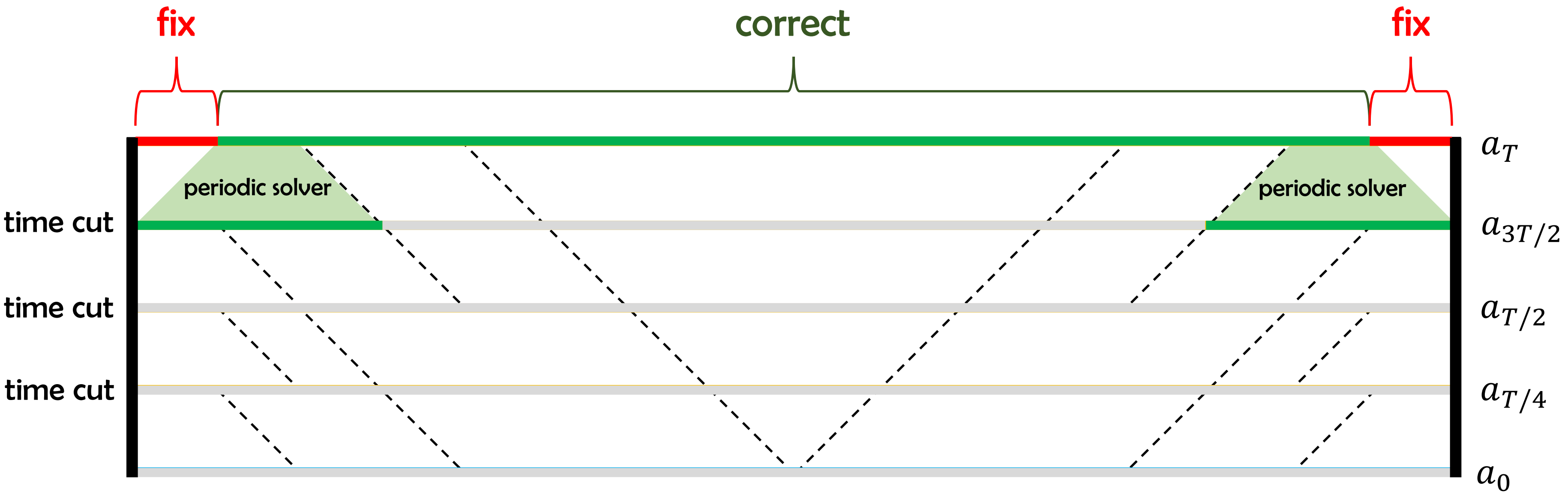
Algorithm for Aperiodic Grids



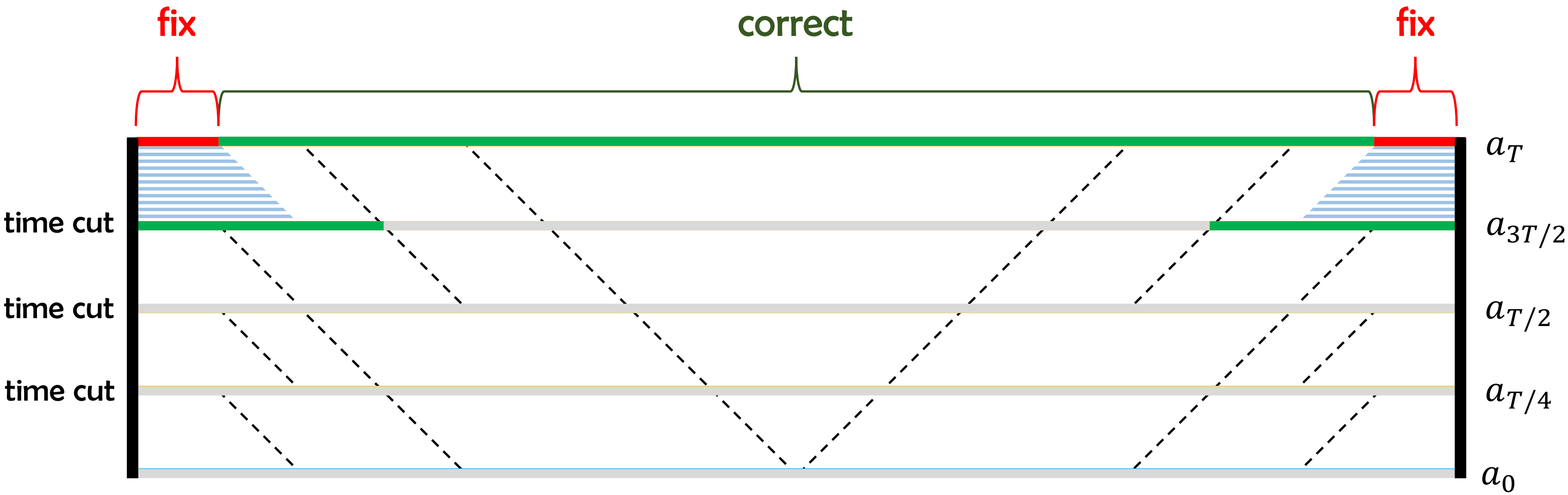
Algorithm for Aperiodic Grids

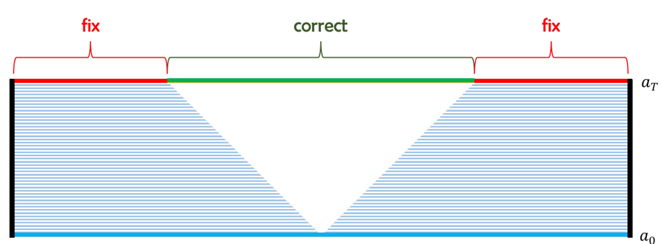


Algorithm for Aperiodic Grids



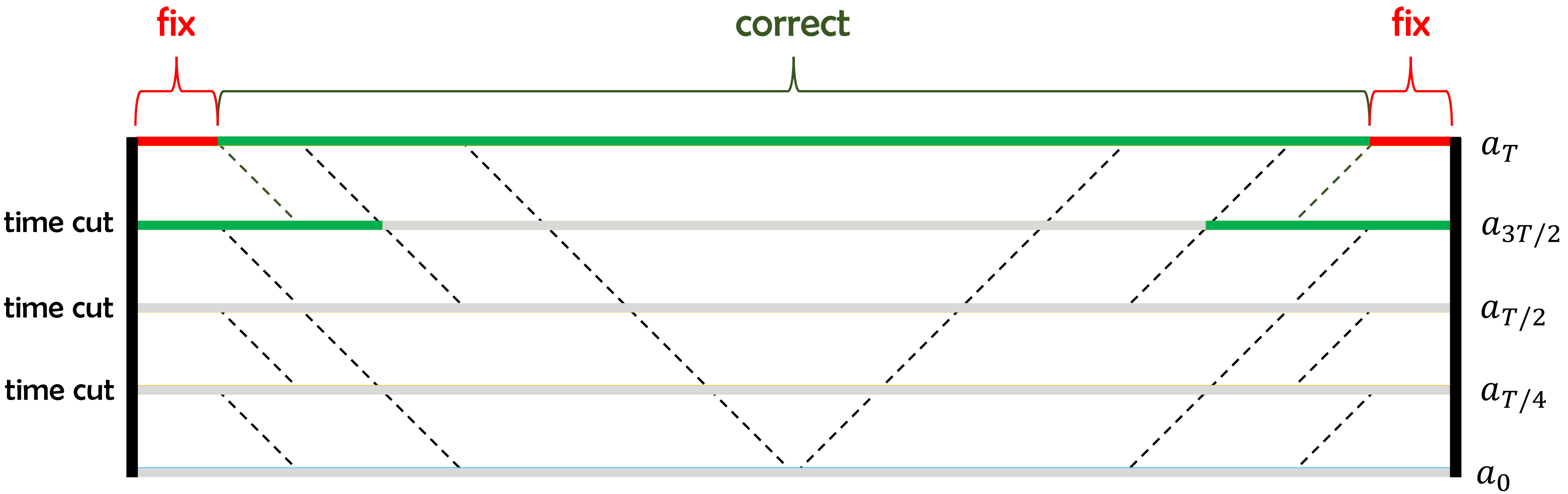
Algorithm for Aperiodic Grids



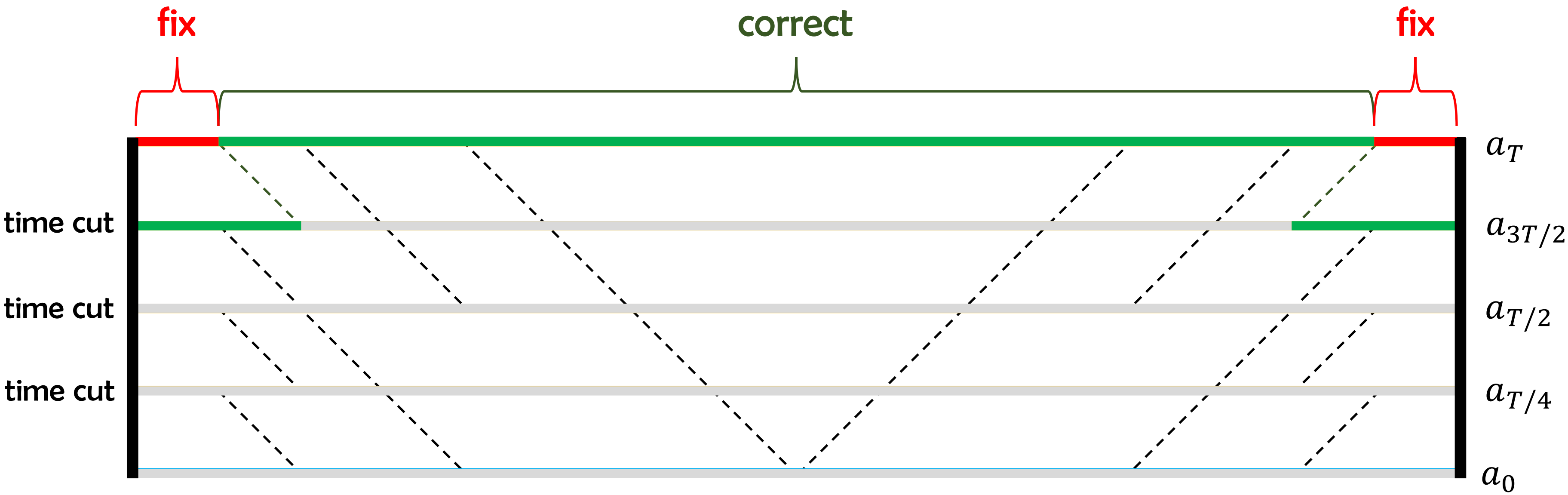


encountered before!

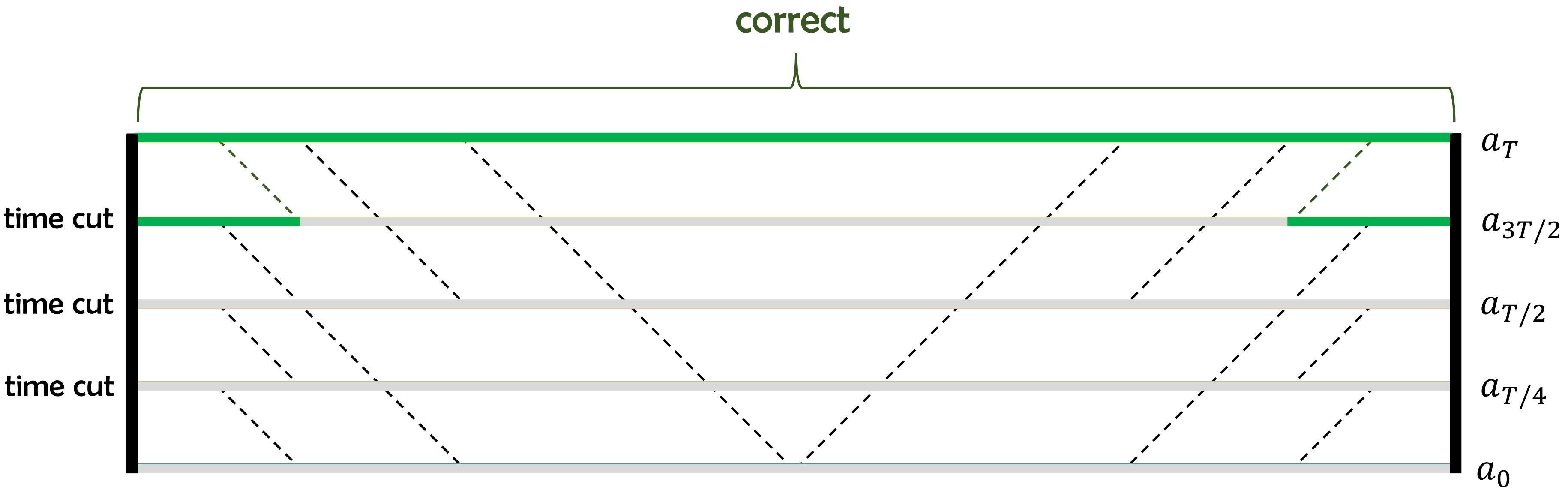
Algorithm for Aperiodic Grids



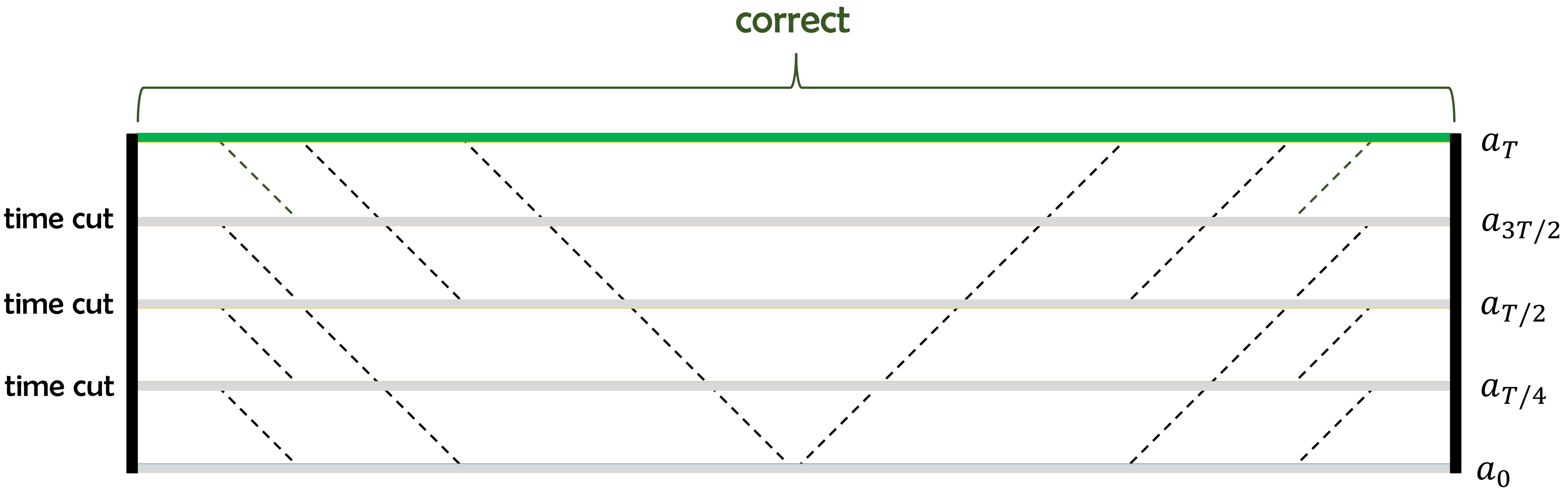
Algorithm for Aperiodic Grids



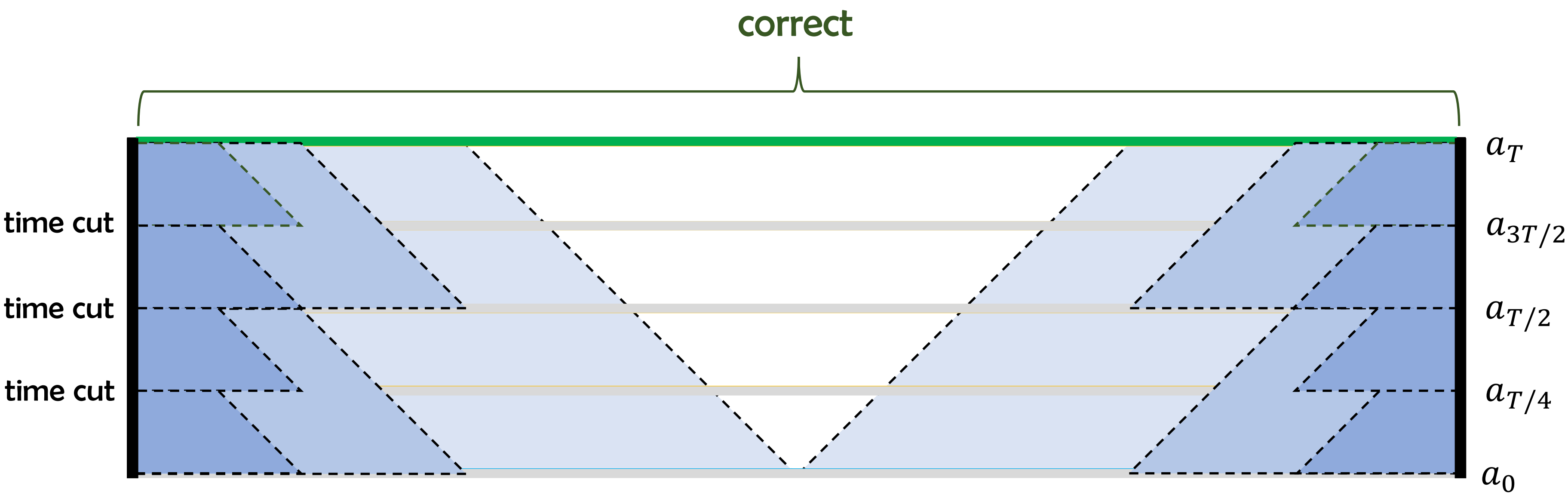
Algorithm for Aperiodic Grids



Algorithm for Aperiodic Grids



Algorithm for Aperiodic Grids



Computational Complexity:

$$\Theta\left(TN^{1-\frac{1}{d}}\log\left(TN^{1-\frac{1}{d}}\right)\log T + N\log N\right)$$

(for a d-dimensional $N^{\frac{1}{d}} \times N^{\frac{1}{d}} \times \cdots \times N^{\frac{1}{d}}$ grid with constant d)

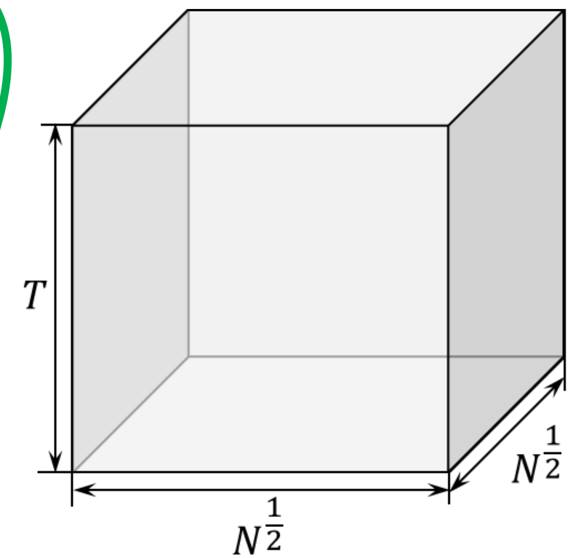
How Good is the Aperiodic Algorithm?

Computational Complexity for a d-dimensional hypercubic $\left(N^{\frac{1}{d}} \times N^{\frac{1}{d}} \times \cdots \times N^{\frac{1}{d}}\right)$ grid (assuming constant d):

$$\Theta\left(\boxed{TN^{1-\frac{1}{d}}} \log\left(TN^{1-\frac{1}{d}}\right) \log T + N \log N\right)$$

But when T timesteps are executed on such a grid, the number of cells on the boundary is $2d\,TN^{1-\frac{1}{d}} = \Theta\left(TN^{1-\frac{1}{d}}\right)$.

If the boundary cells can take arbitrary values, one must read every boundary cell for correctness.

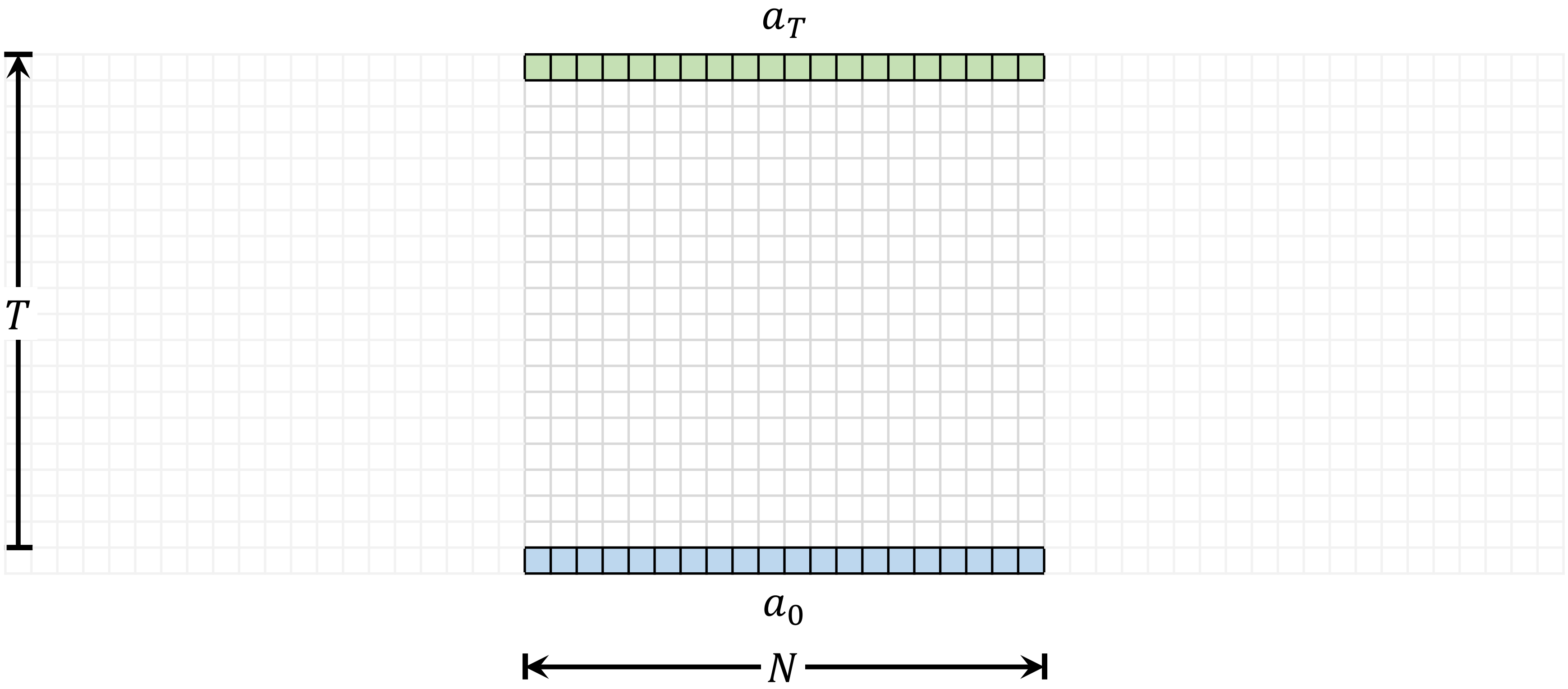


Thus, one is forced to do $\Omega\left(\boxed{TN^{1-\frac{1}{d}}}\right)$ work in such a case.

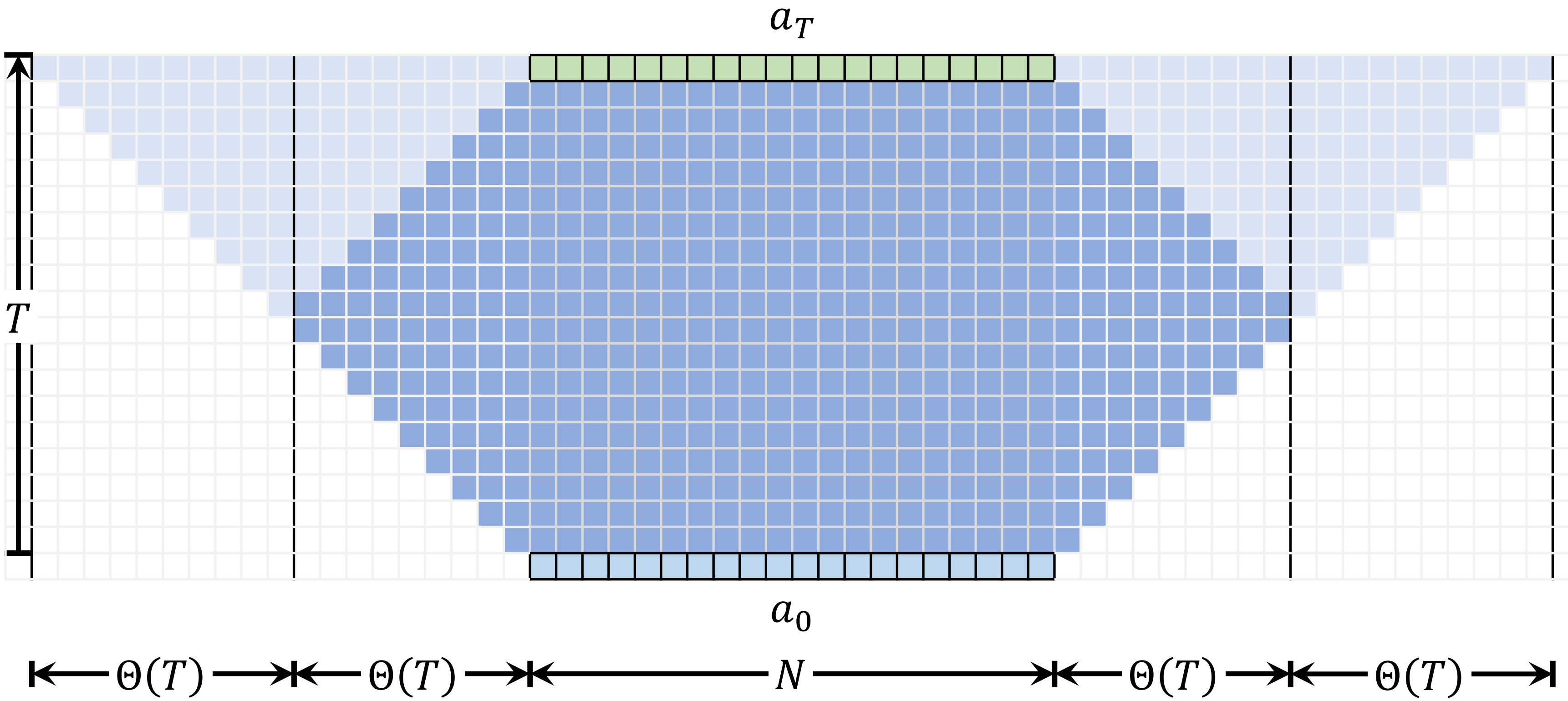
Hence, for large T and arbitrary boundary conditions, the aperiodic algorithm is within polylog factor of optimal.

Freespace Grids:
Freespace has a Space Problem

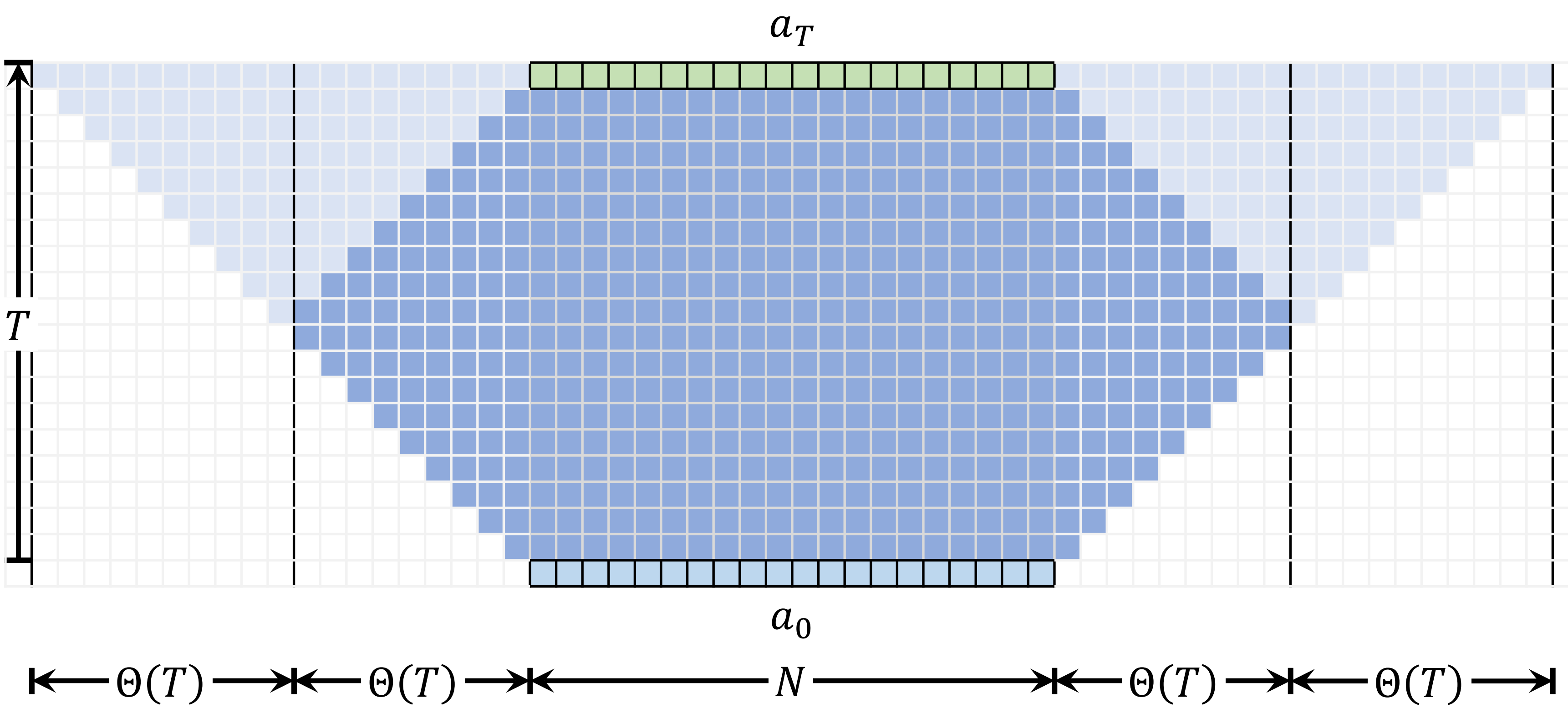
Freespace May Blow up Space (and Time)



Freespace May Blow up Space (and Time)



Freespace May Blow up Space (and Time)



Algorithm	Space	Work
Standard (Loops, Tiled Loops, Recursive Tiling)	$\Theta(N + T^d)$	$\Theta\left((N + T^d)\, T\right)$
FFT-based	$\Theta(N + T^d)$	$\Theta\left((N + T^d) \log(NT)\right)$

for d-dimensional hypercubic $\left(N^{\frac{1}{d}} \times N^{\frac{1}{d}} \times \cdots \times N^{\frac{1}{d}}\right)$ grids with constant d

Freespace Grids:

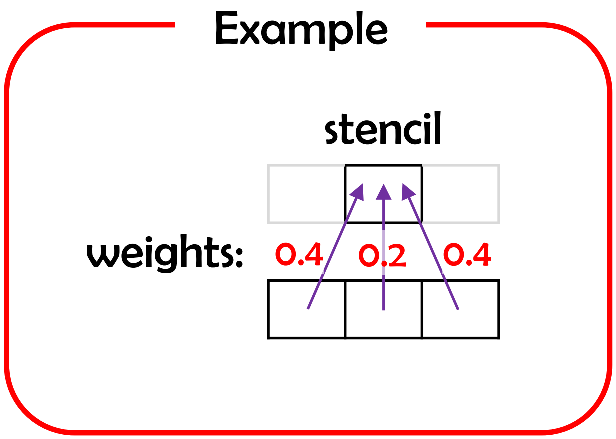
S^{∞} is a Gaussian

S^{T} Approximates a Gaussian

Normalized Nonnegative Stencils

We assume that Stencil S is

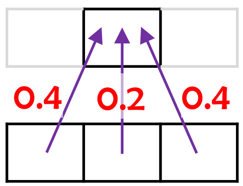
- linear,
- composed of nonnegative weights only, and
- normalized so that all weights add up to 1.



Normalized Nonnegative Stencils and Random Walks

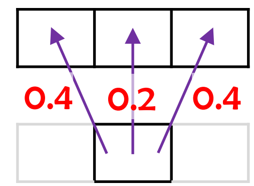
stencil

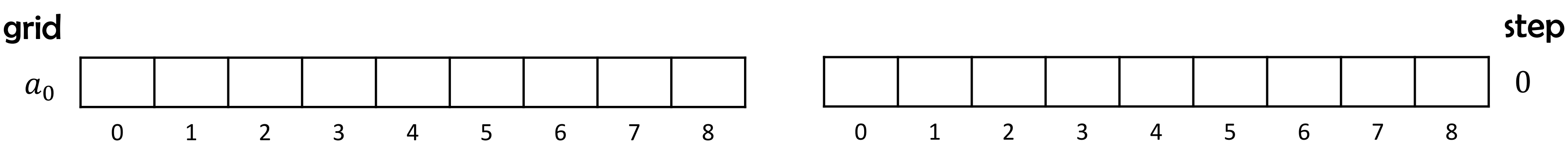
weights:



random walk

probabilities:





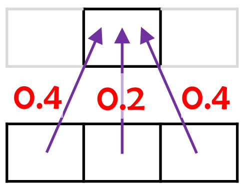
grid cell values
after various timesteps

walker location probabilities
after various number of steps

Normalized Nonnegative Stencils and Random Walks

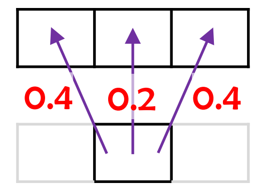
stencil

weights:



random walk

probabilities:



grid

	0	1	2	3	4	5	6	7	8
a_0	0.000	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	0.000

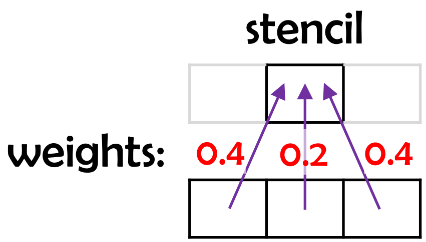
step

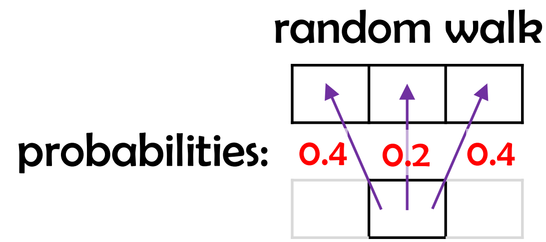
0	1	2	3	4	5	6	7	8	
0.000	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	0.000	0

grid cell values
after various timesteps

walker location probabilities
after various number of steps

Normalized Nonnegative Stencils and Random Walks





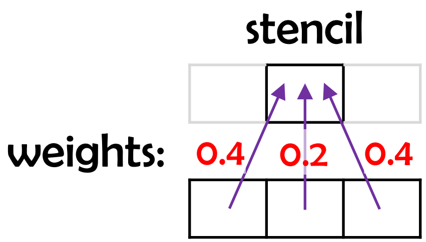
grid									
a_1	0.000	0.000	0.000	**0.400**	**0.200**	**0.400**	0.000	0.000	0.000
a_0	0.000	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	0.000
	0	1	2	3	4	5	6	7	8

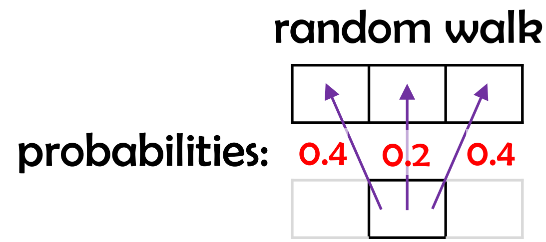
grid cell values
after various timesteps

									step
0.000	0.000	0.000	**0.400**	**0.200**	**0.400**	0.000	0.000	0.000	1
0.000	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	0.000	0
0	1	2	3	4	5	6	7	8	

walker location probabilities
after various number of steps

Normalized Nonnegative Stencils and Random Walks





grid									
a_2	0.000	0.000	**0.160**	**0.160**	**0.360**	**0.160**	**0.160**	0.000	0.000
a_1	0.000	0.000	0.000	**0.400**	**0.200**	**0.400**	0.000	0.000	0.000
a_0	0.000	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	0.000
	0	1	2	3	4	5	6	7	8

grid cell values
after various timesteps

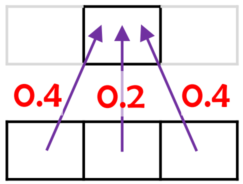
									step
0.000	0.000	**0.160**	**0.160**	**0.360**	**0.160**	**0.160**	0.000	0.000	2
0.000	0.000	0.000	**0.400**	**0.200**	**0.400**	0.000	0.000	0.000	1
0.000	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	0.000	0
0	1	2	3	4	5	6	7	8	

walker location probabilities
after various number of steps

Normalized Nonnegative Stencils and Random Walks

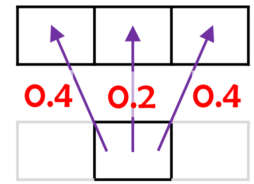
stencil

weights:



random walk

probabilities:



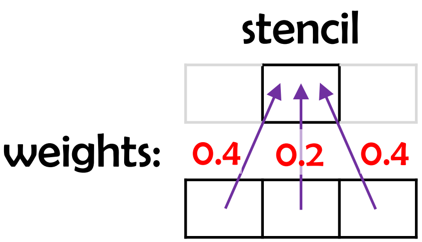
grid	0	1	2	3	4	5	6	7	8
a_3	0.000	**0.064**	**0.096**	**0.240**	**0.200**	**0.240**	**0.096**	**0.064**	0.000
a_2	0.000	0.000	**0.160**	**0.160**	**0.360**	**0.160**	**0.160**	0.000	0.000
a_1	0.000	0.000	0.000	**0.400**	**0.200**	**0.400**	0.000	0.000	0.000
a_0	0.000	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	0.000

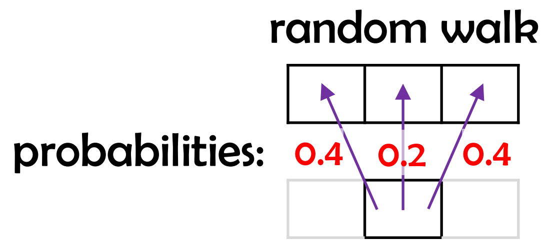
grid cell values
after various timesteps

0	1	2	3	4	5	6	7	8	step
0.000	**0.064**	**0.096**	**0.240**	**0.200**	**0.240**	**0.096**	**0.064**	0.000	3
0.000	0.000	**0.160**	**0.160**	**0.360**	**0.160**	**0.160**	0.000	0.000	2
0.000	0.000	0.000	**0.400**	**0.200**	**0.400**	0.000	0.000	0.000	1
0.000	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	0.000	0

walker location probabilities
after various number of steps

Normalized Nonnegative Stencils and Tracking a Bunch of Drunk Walkers





grid	0	1	2	3	4	5	6	7	8
a_3	0.000	**0.320**	**0.480**	**1.200**	**1.000**	**1.200**	**0.480**	**0.320**	0.000
a_2	0.000	0.000	**0.800**	**0.800**	**1.800**	**0.800**	**0.800**	0.000	0.000
a_1	0.000	0.000	0.000	**2.000**	**1.000**	**2.000**	0.000	0.000	0.000
a_0	0.000	0.000	0.000	0.000	**5.000**	0.000	0.000	0.000	0.000

grid cell values
after various timesteps

expected #walkers in each cell after 3 steps

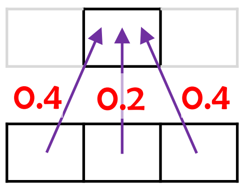
0	1	2	3	4	5	6	7	8	step
0.000	**0.320**	**0.480**	**1.200**	**1.000**	**1.200**	**0.480**	**0.320**	0.000	3
0.000	0.000	**0.800**	**0.800**	**1.800**	**0.800**	**0.800**	0.000	0.000	2
0.000	0.000	0.000	**2.000**	**1.000**	**2.000**	0.000	0.000	0.000	1
0.000	0.000	0.000	0.000	**5.000**	0.000	0.000	0.000	0.000	0

expected #walkers in each cell
after various number of steps

Normalized Nonnegative Stencils and Tracking a Bunch of Drunk Walkers

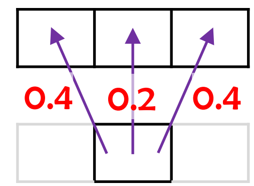
stencil

weights:



random walk

probabilities:



grid									
a_3	0.128	0.512	1.216	1.984	2.440	2.192	1.568	0.704	0.256
a_2	0.000	0.320	1.120	2.160	2.760	2.560	1.440	0.640	0.000
a_1	0.000	0.000	0.800	2.400	3.400	2.800	1.600	0.000	0.000
a_0	0.000	0.000	0.000	2.000	5.000	4.000	0.000	0.000	0.000
	0	1	2	3	4	5	6	7	8

grid cell values
after various timesteps

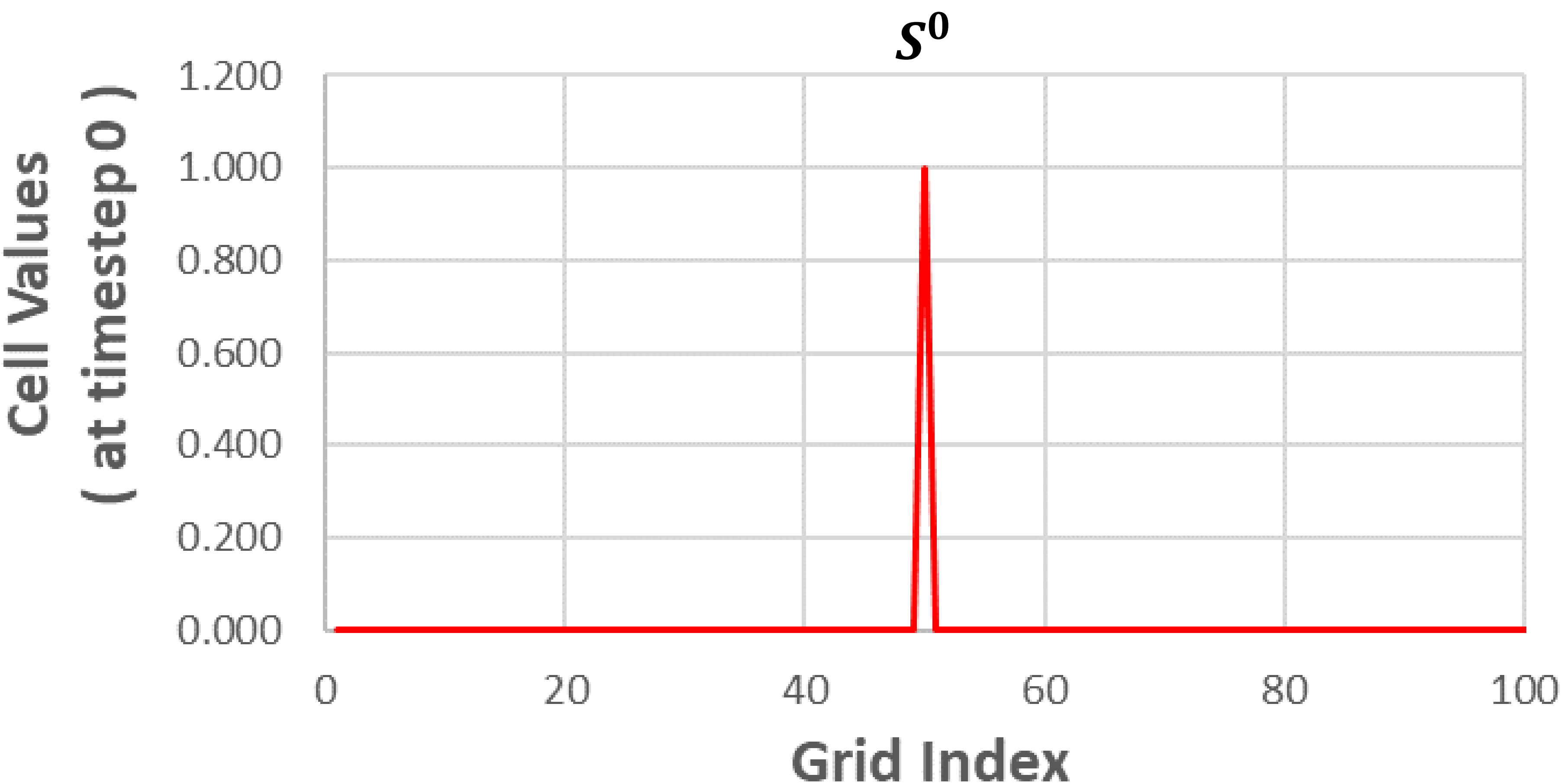
expected #walkers in each cell after 3 steps

									step
0.128	0.512	1.216	1.984	2.440	2.192	1.568	0.704	0.256	3
0.000	0.320	1.120	2.160	2.760	2.560	1.440	0.640	0.000	2
0.000	0.000	0.800	2.400	3.400	2.800	1.600	0.000	0.000	1
0.000	0.000	0.000	2.000	5.000	4.000	0.000	0.000	0.000	0
0	1	2	3	4	5	6	7	8	

expected #walkers in each cell
after various number of steps

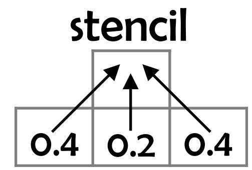
S^T Approximates a Gaussian

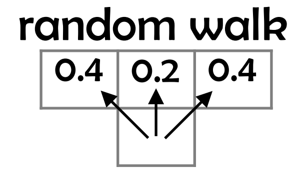
3-point symmetric stencil:

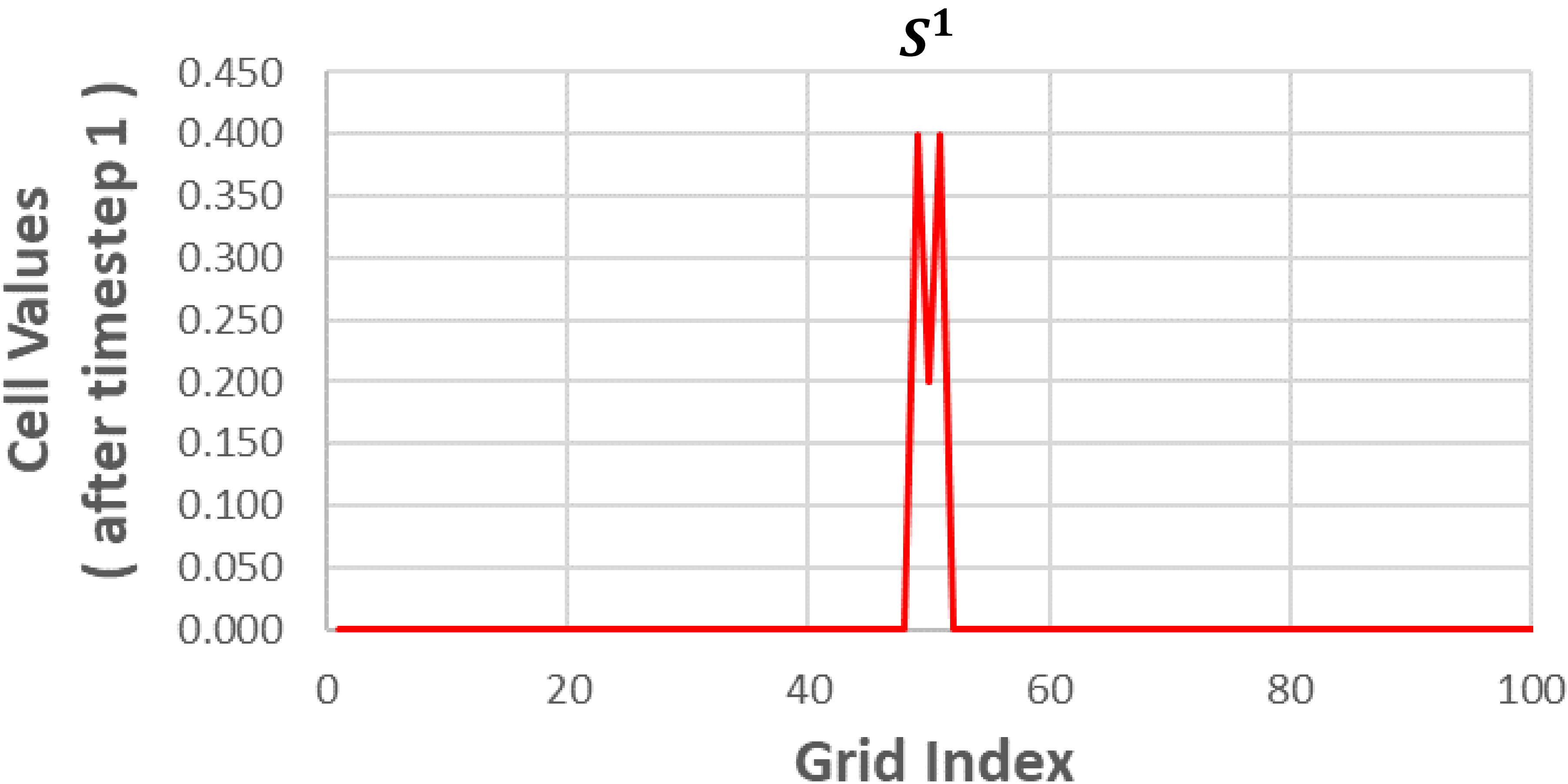


S^T Approximates a Gaussian

3-point symmetric stencil:

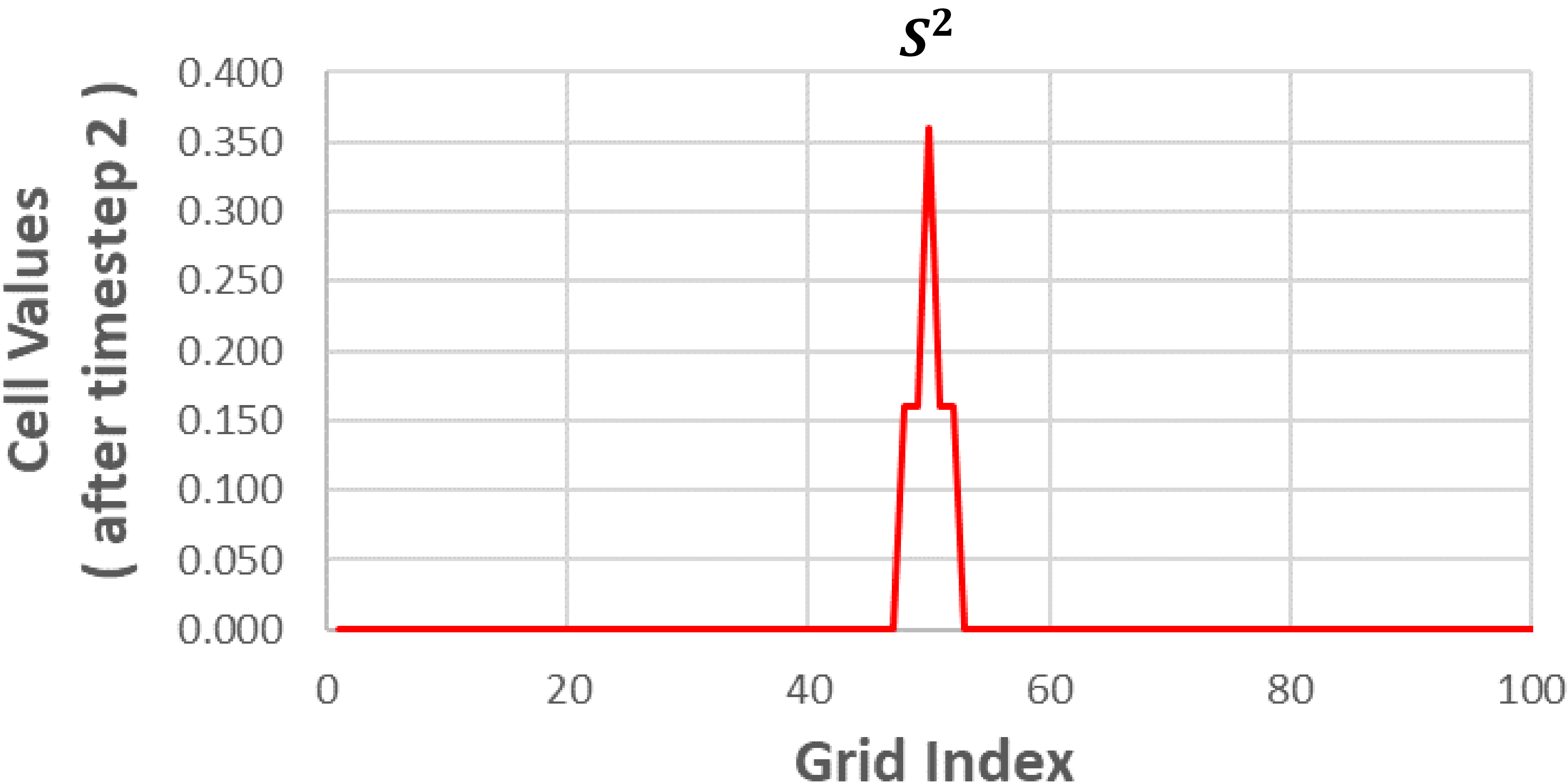






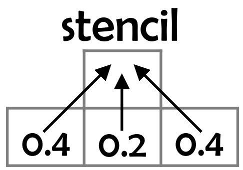
S^T Approximates a Gaussian

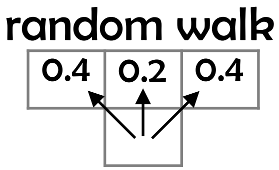
3-point symmetric stencil:

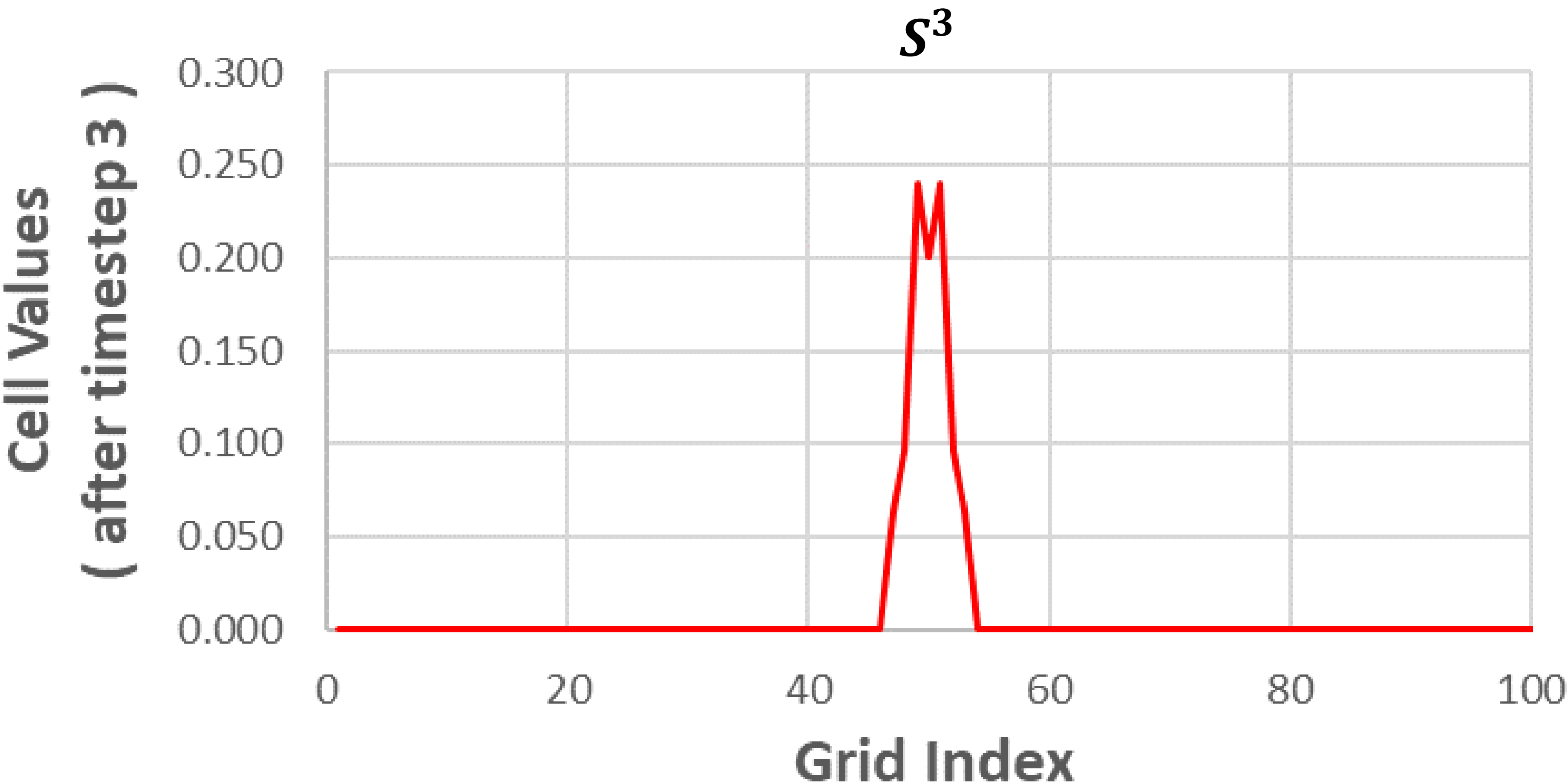


S^T Approximates a Gaussian

3-point symmetric stencil:

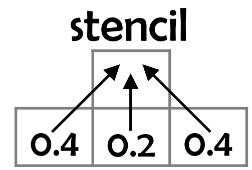


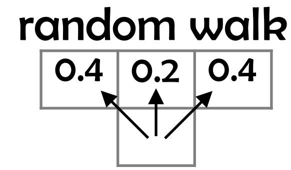


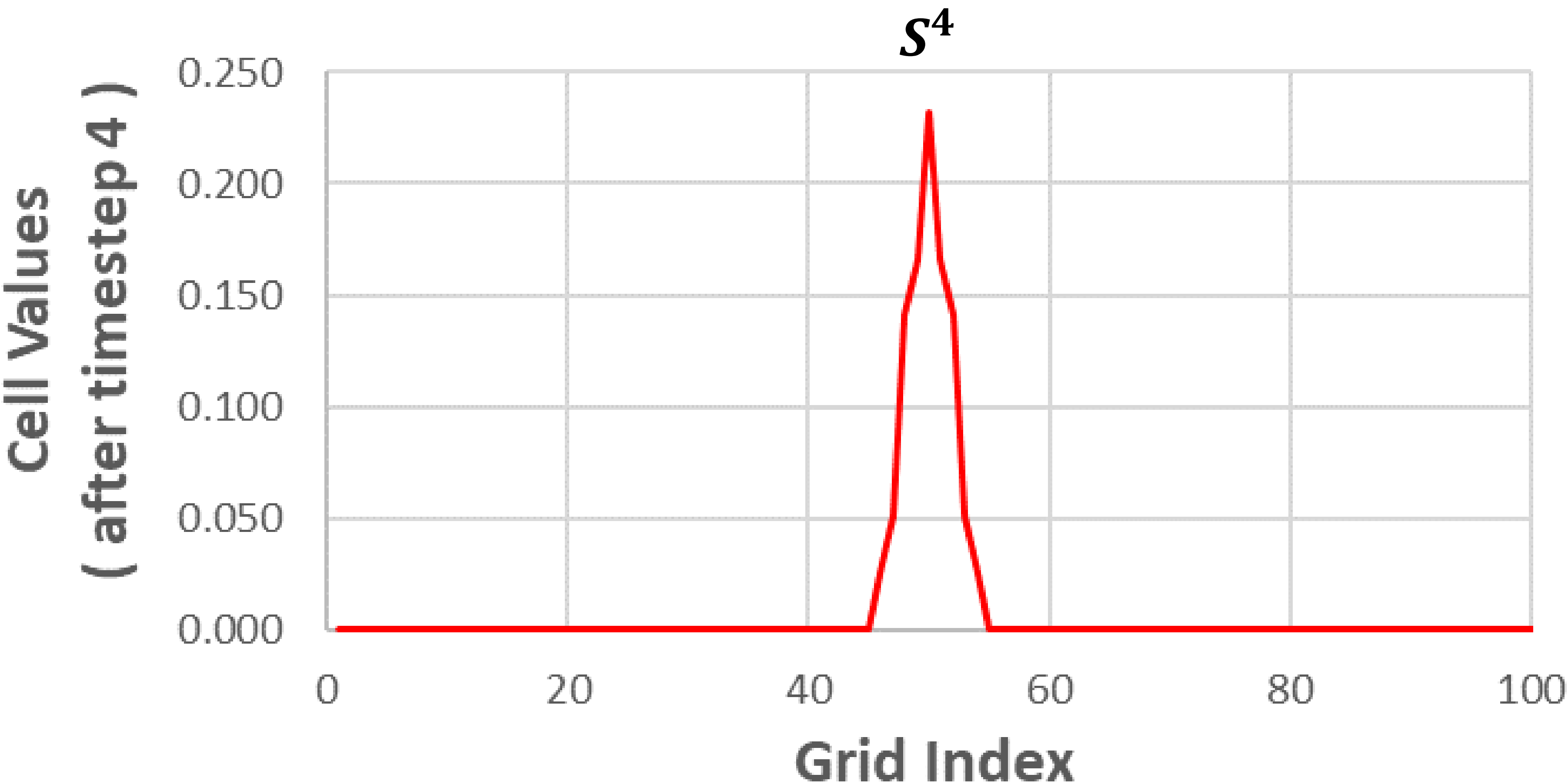


S^T Approximates a Gaussian

3-point symmetric stencil:



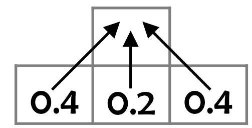




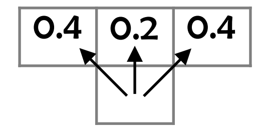
S^T Approximates a Gaussian

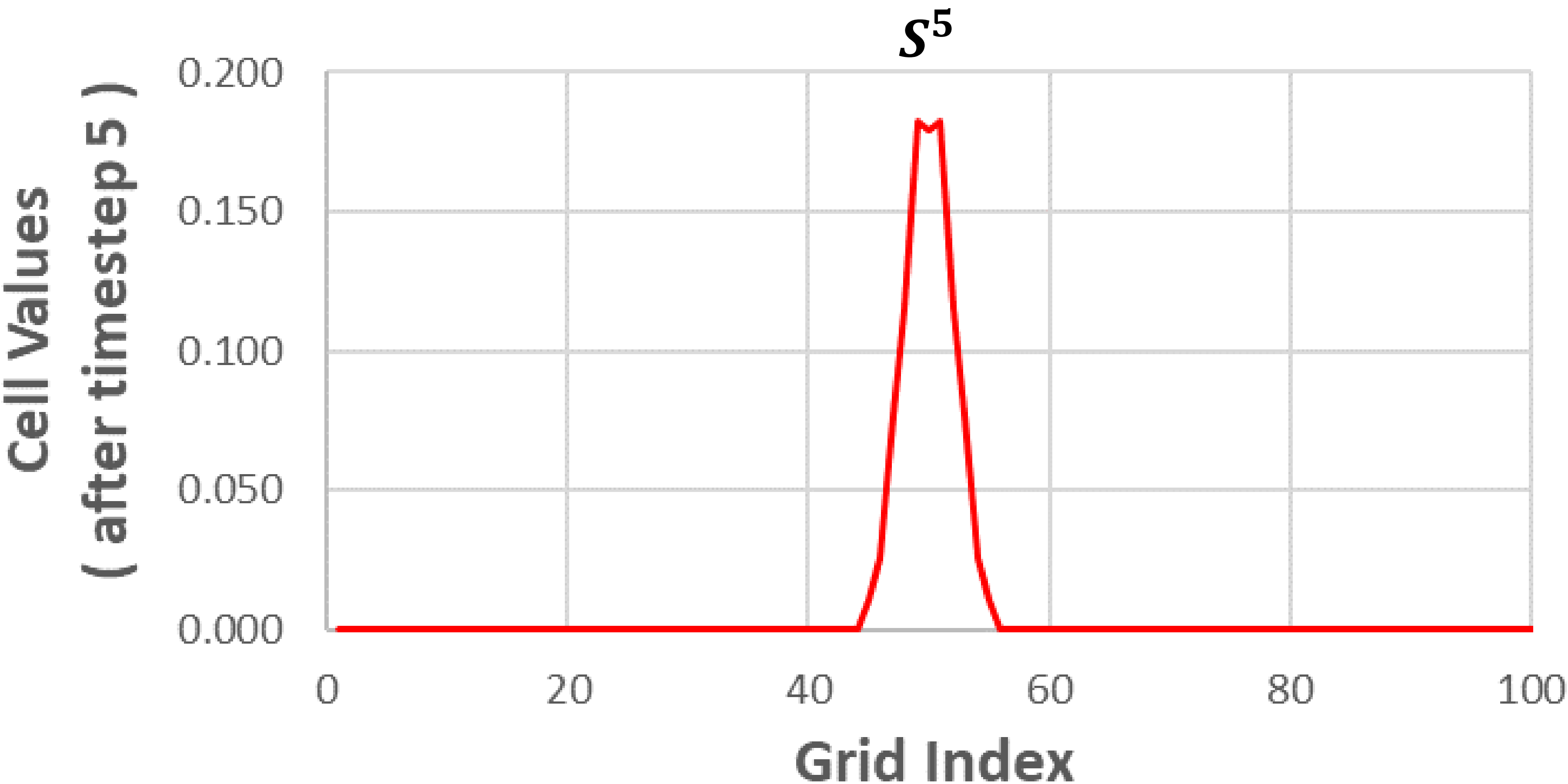
3-point symmetric stencil:

stencil



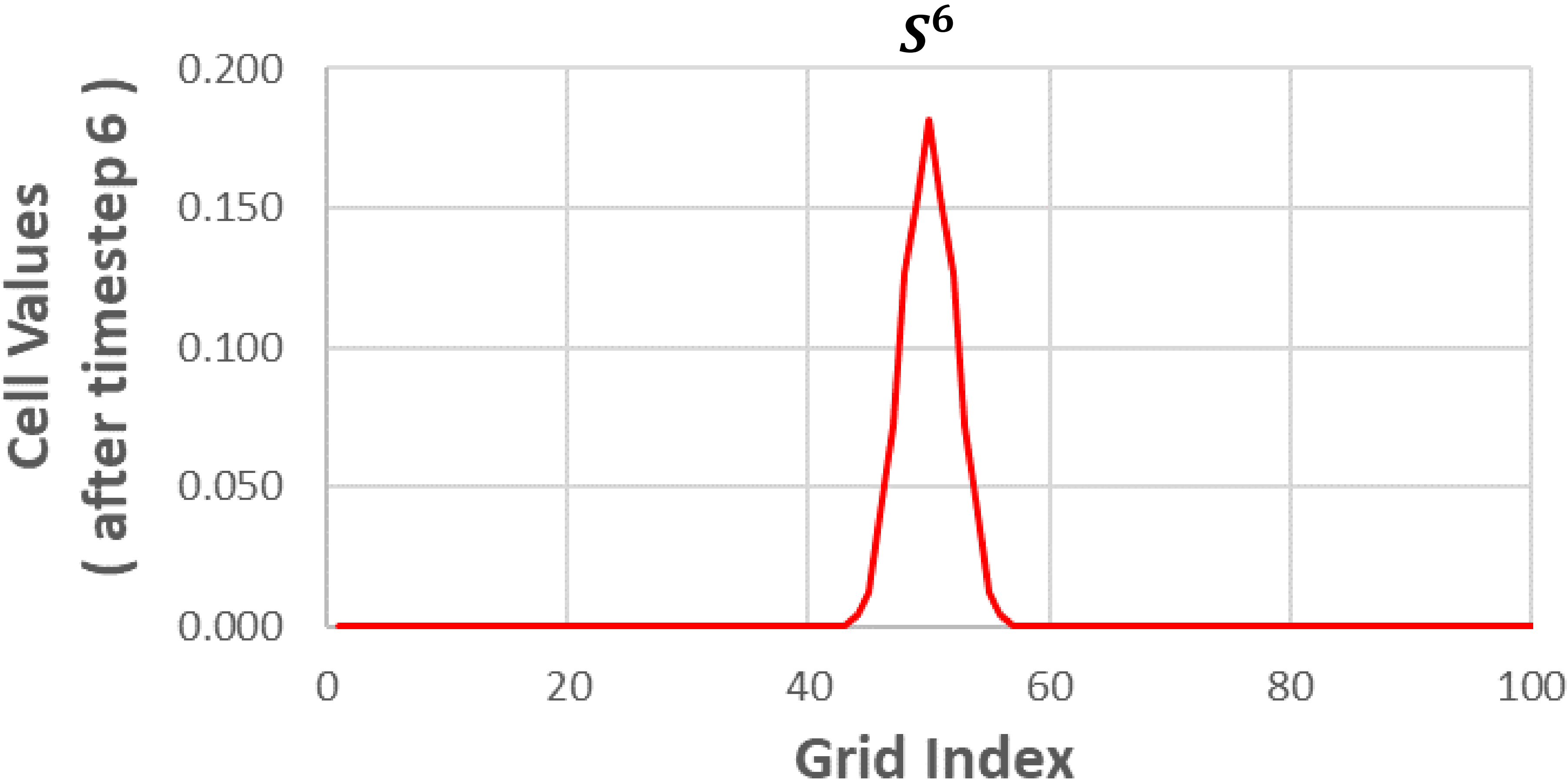
random walk





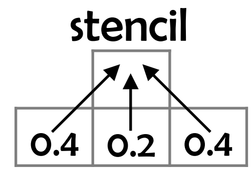
S^T Approximates a Gaussian

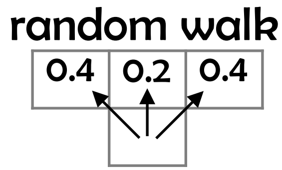
3-point symmetric stencil:

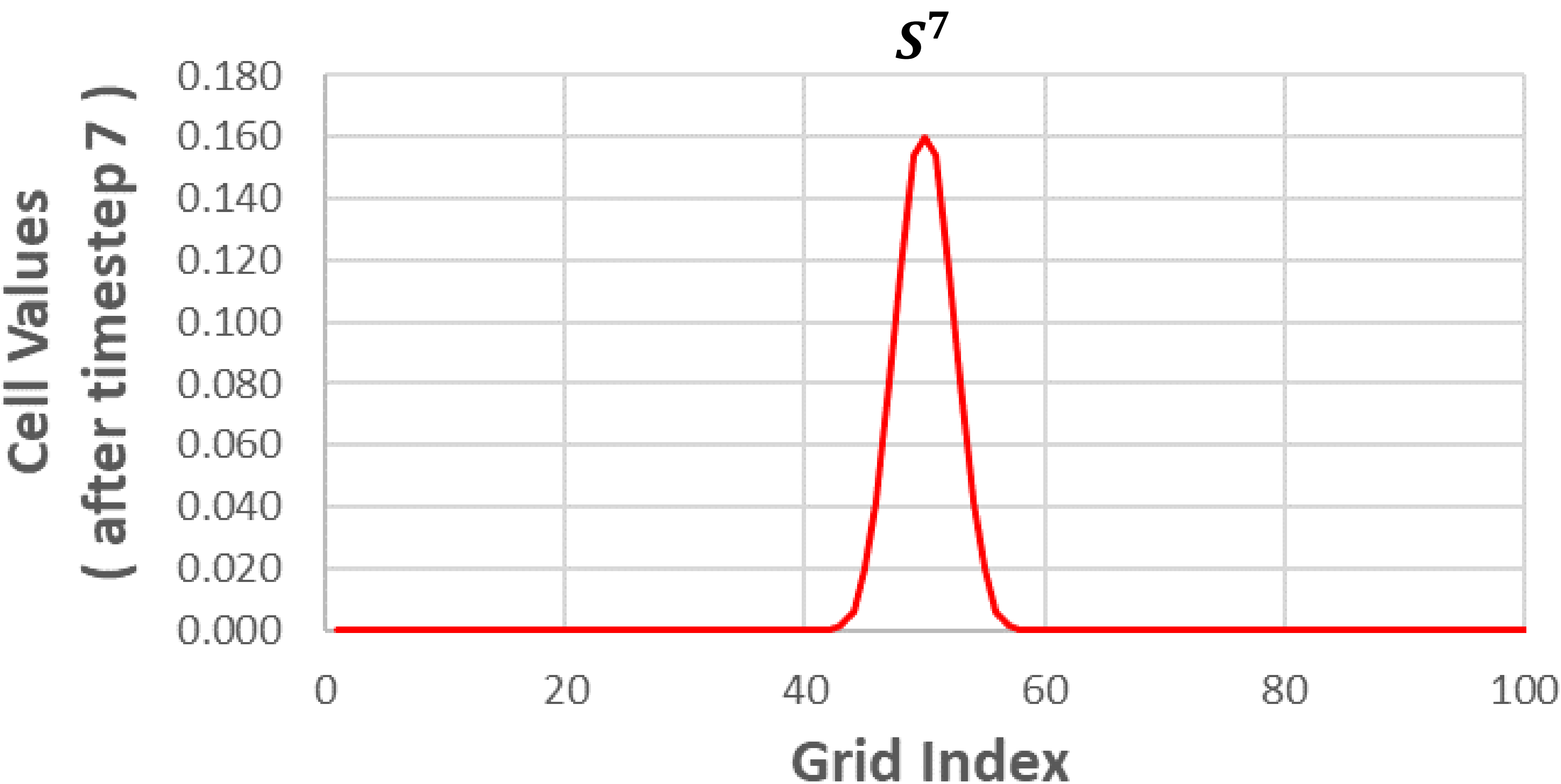


S^T Approximates a Gaussian

3-point symmetric stencil:

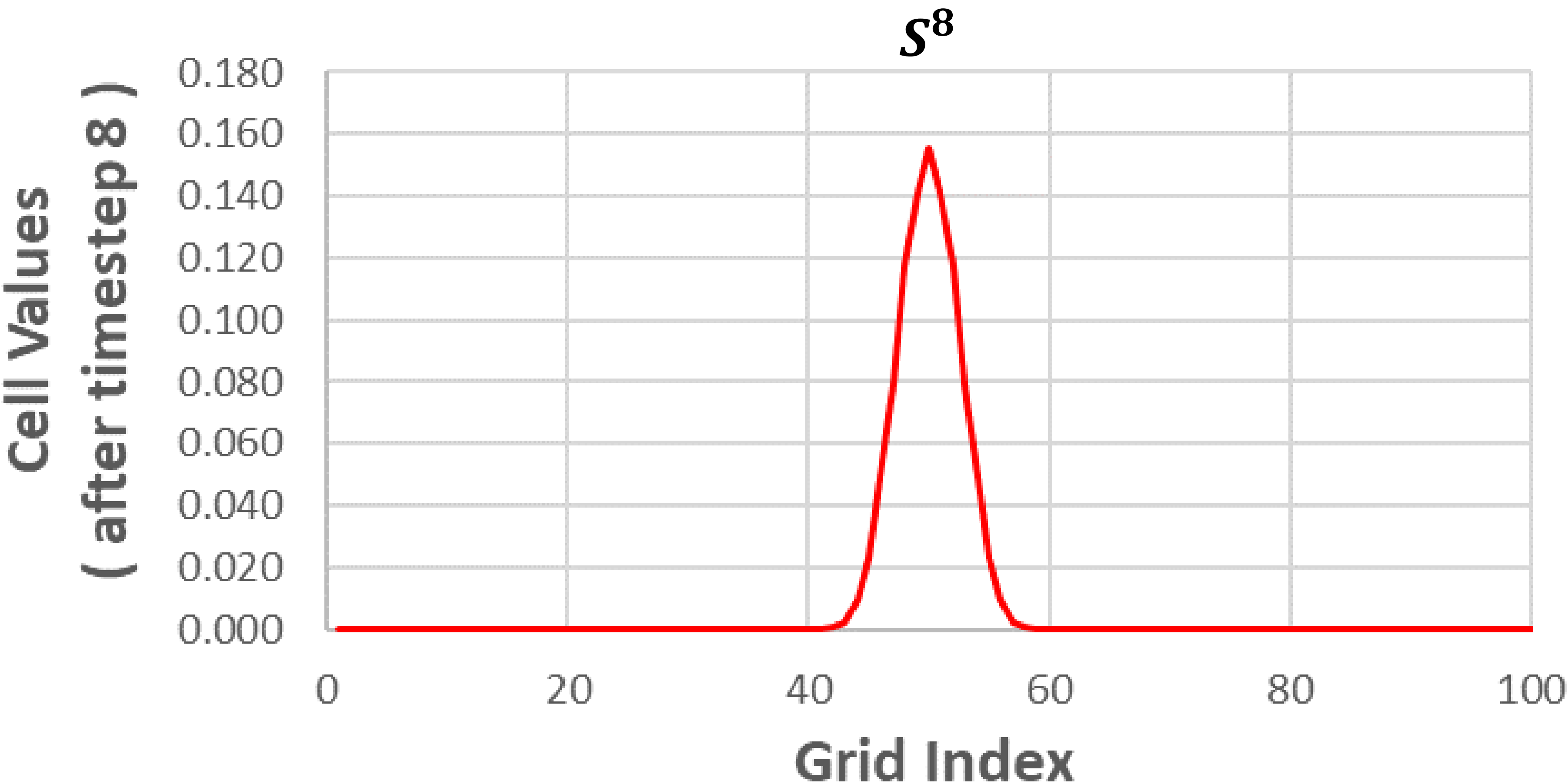






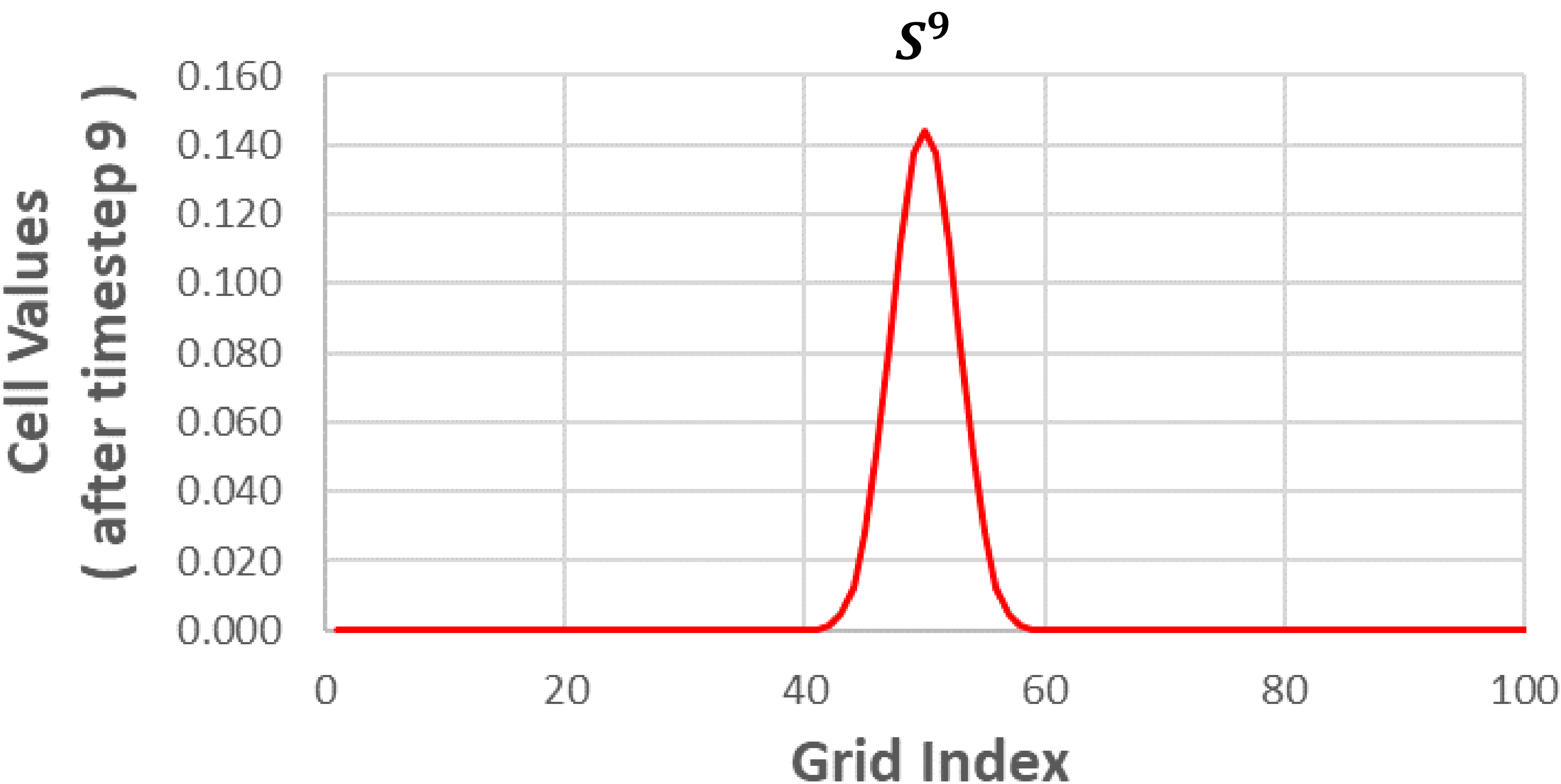
S^T Approximates a Gaussian

3-point symmetric stencil:



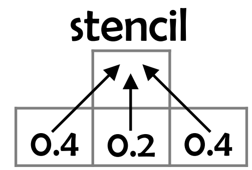
S^T Approximates a Gaussian

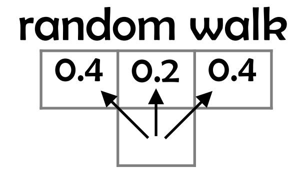
3-point symmetric stencil:

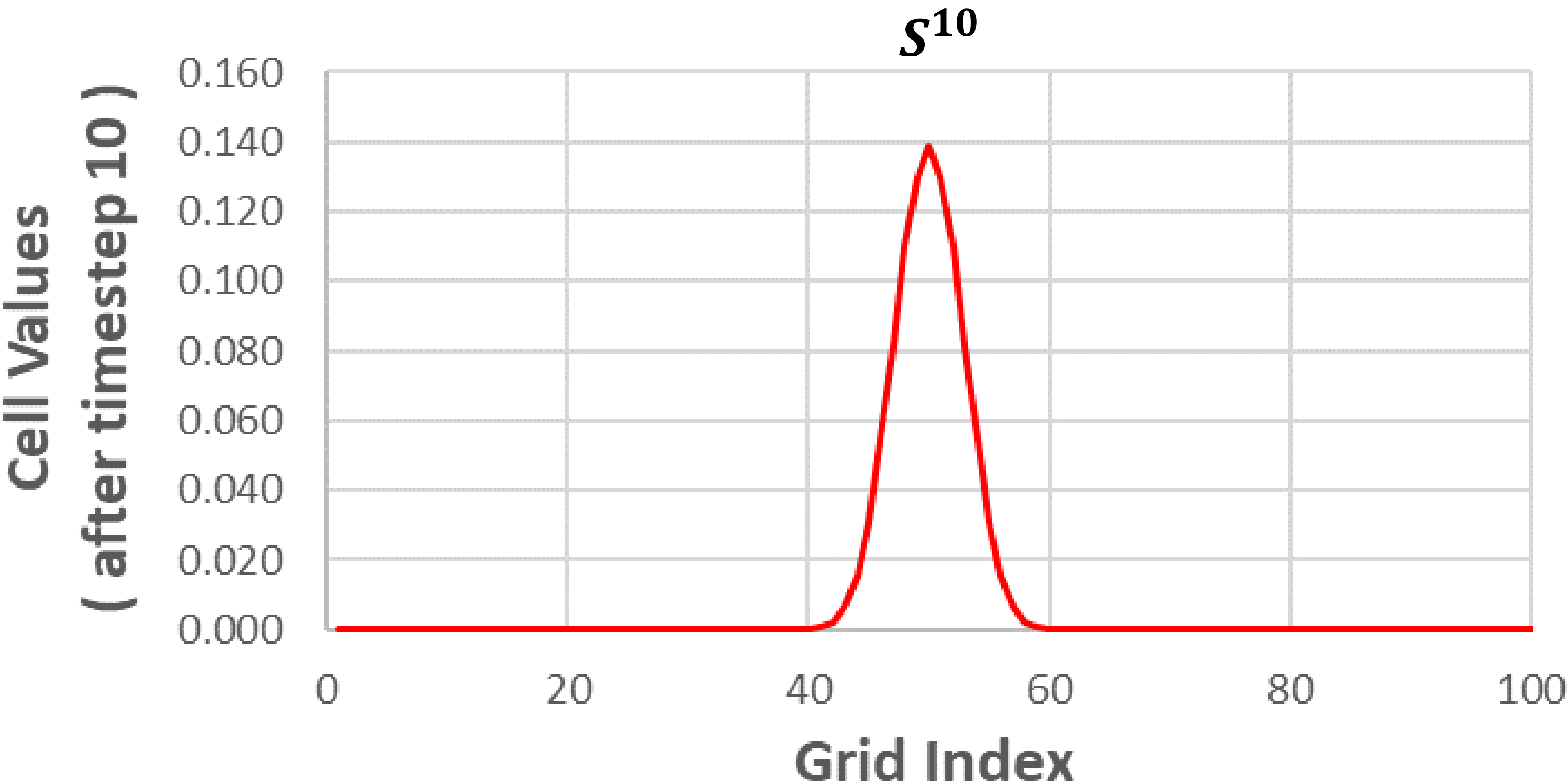


S^T Approximates a Gaussian

3-point symmetric stencil:

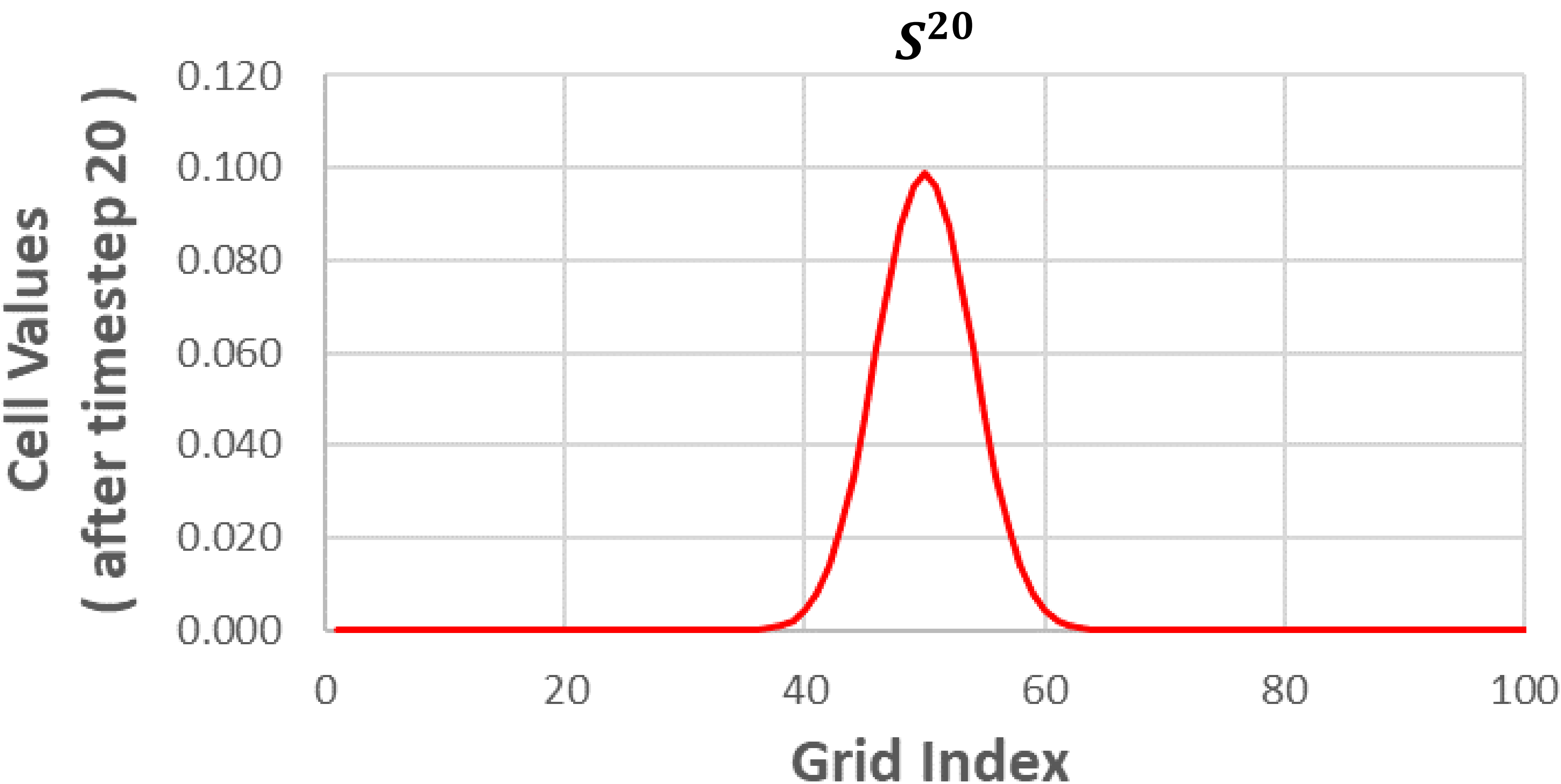






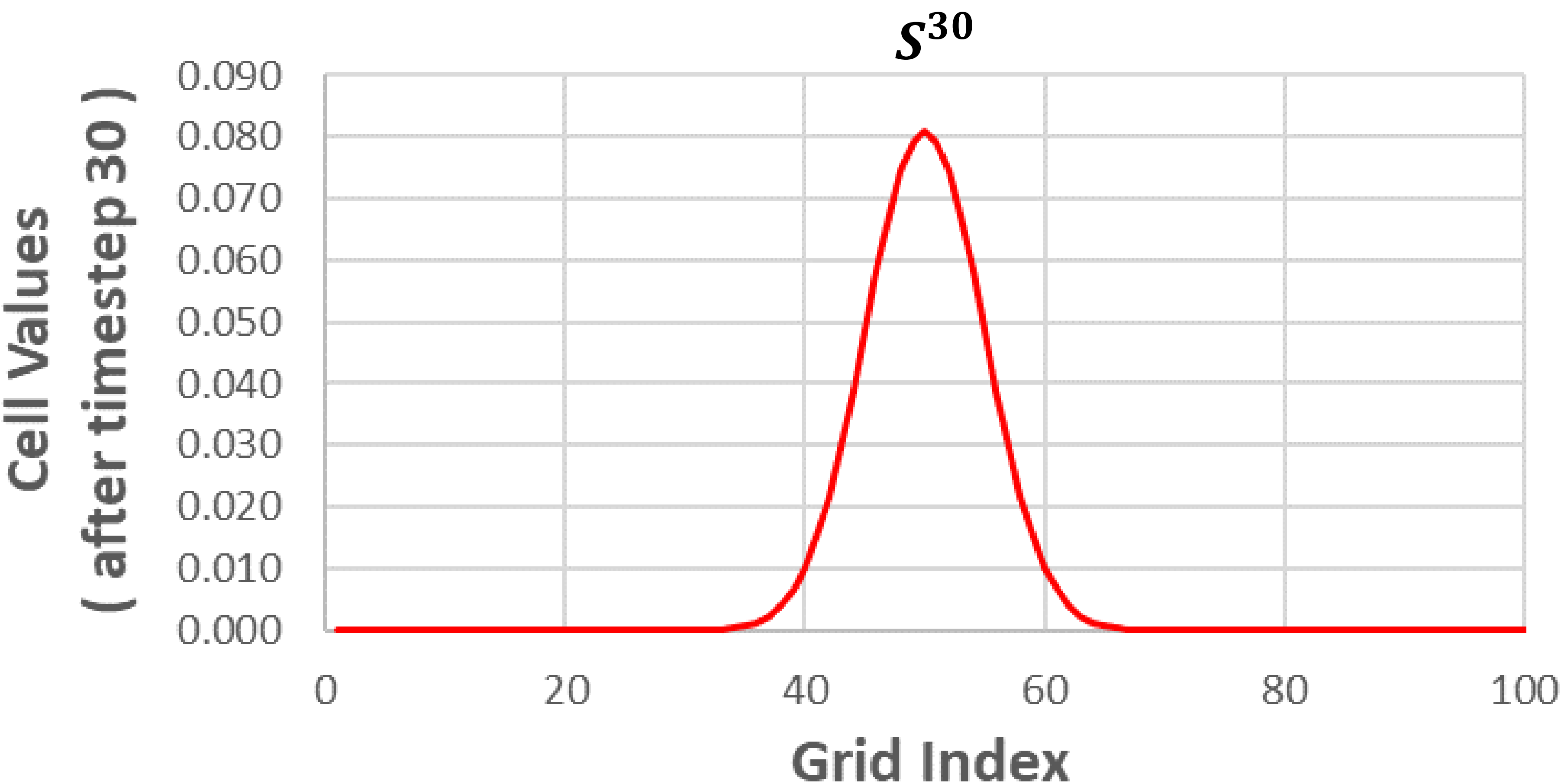
S^T Approximates a Gaussian

3-point symmetric stencil:



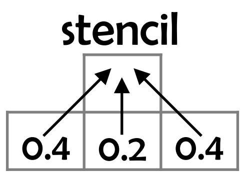
S^T Approximates a Gaussian

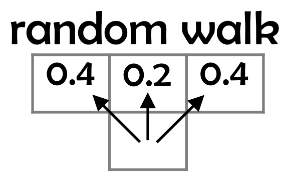
3-point symmetric stencil:

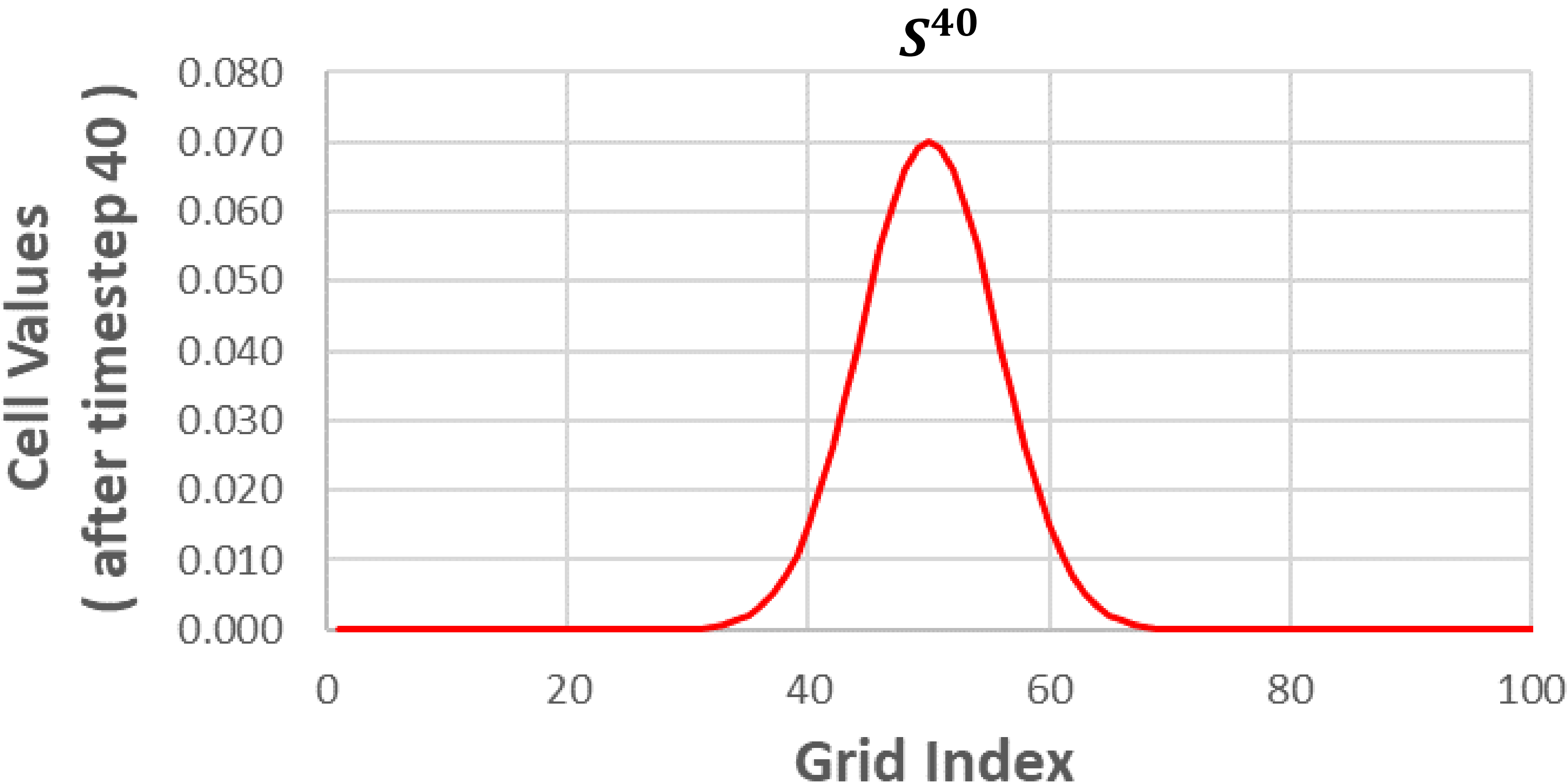


S^T Approximates a Gaussian

3-point symmetric stencil:

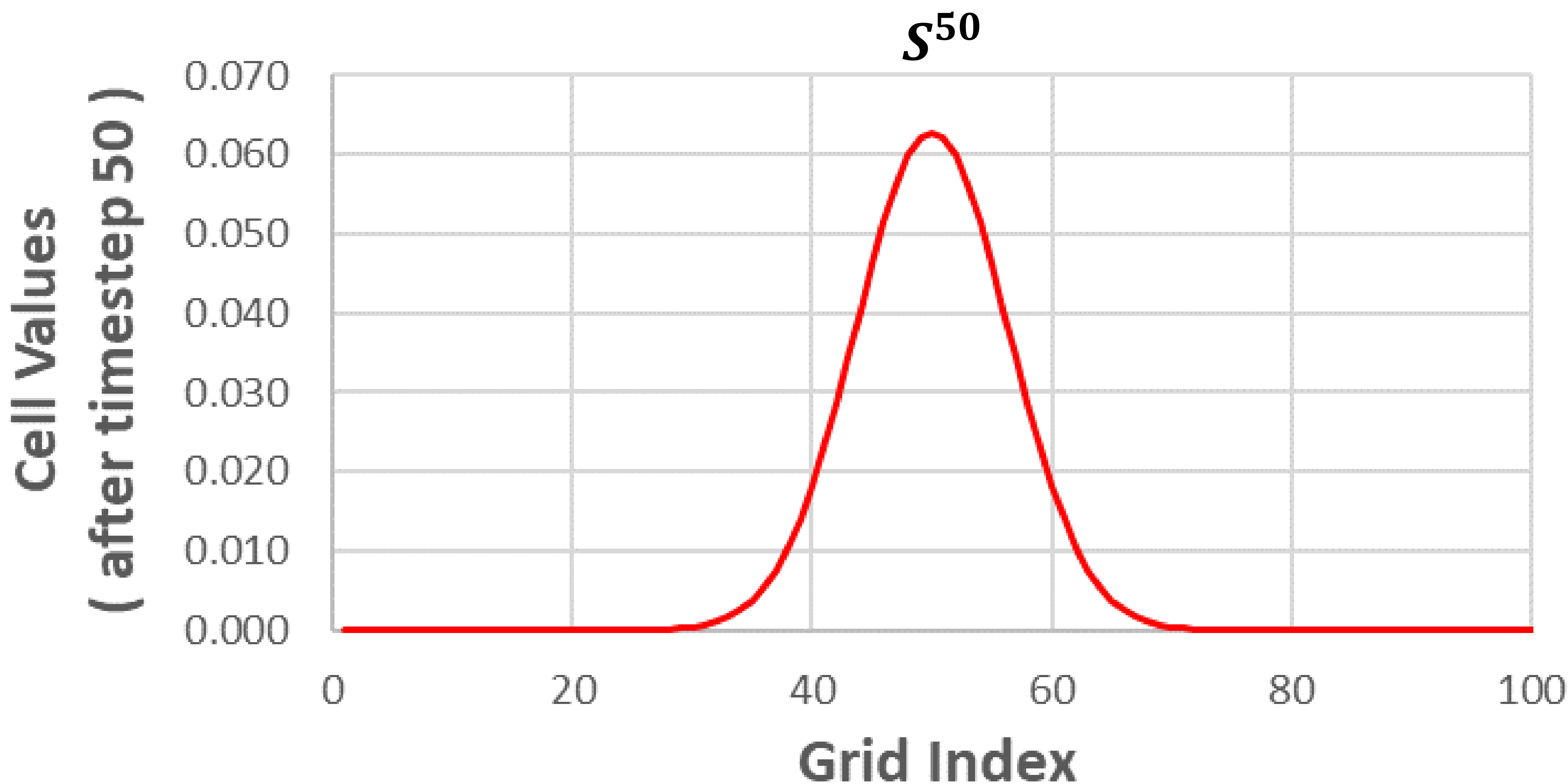






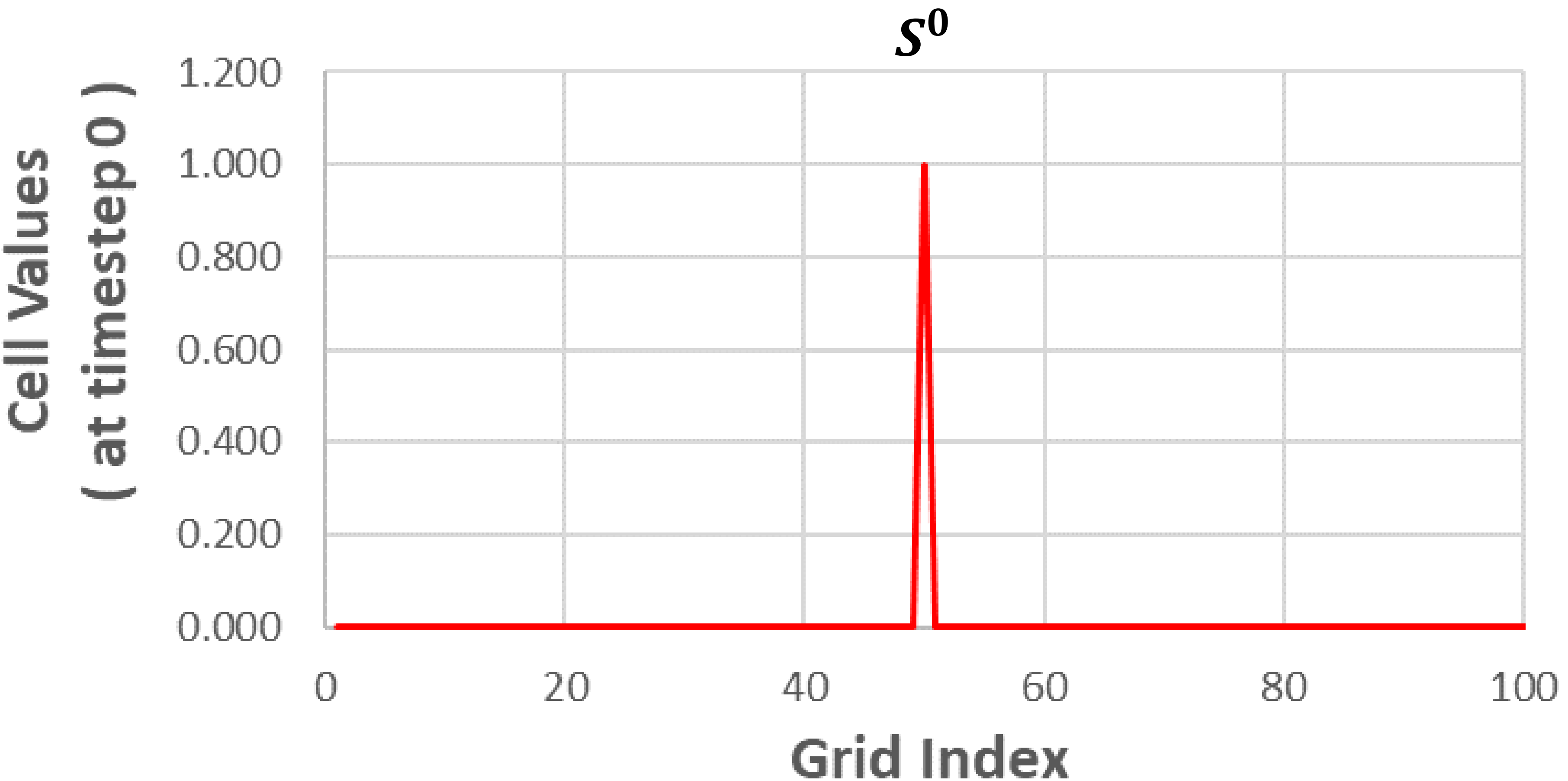
S^T Approximates a Gaussian

3-point symmetric stencil:



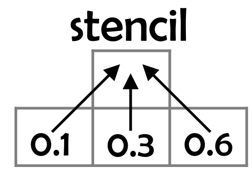
S^T Approximates a Gaussian

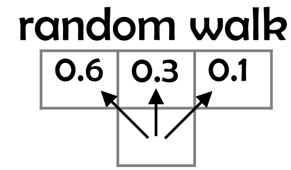
3-point asymmetric stencil:

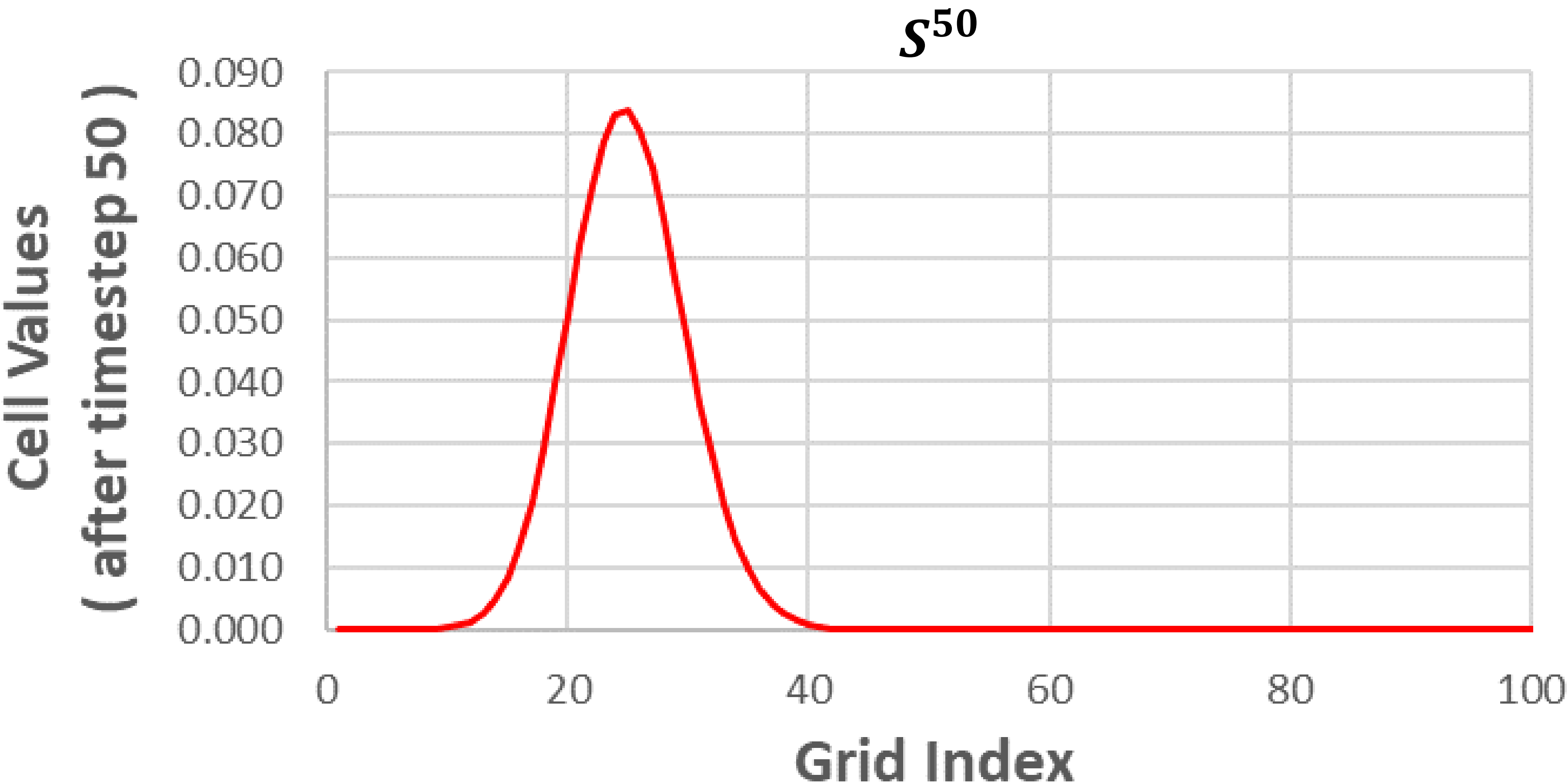


S^T Approximates a Gaussian

3-point asymmetric stencil:

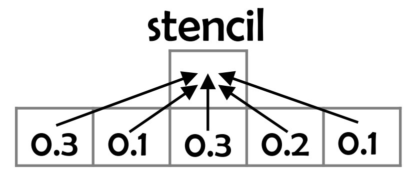


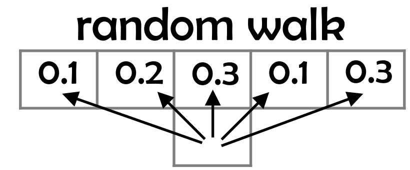


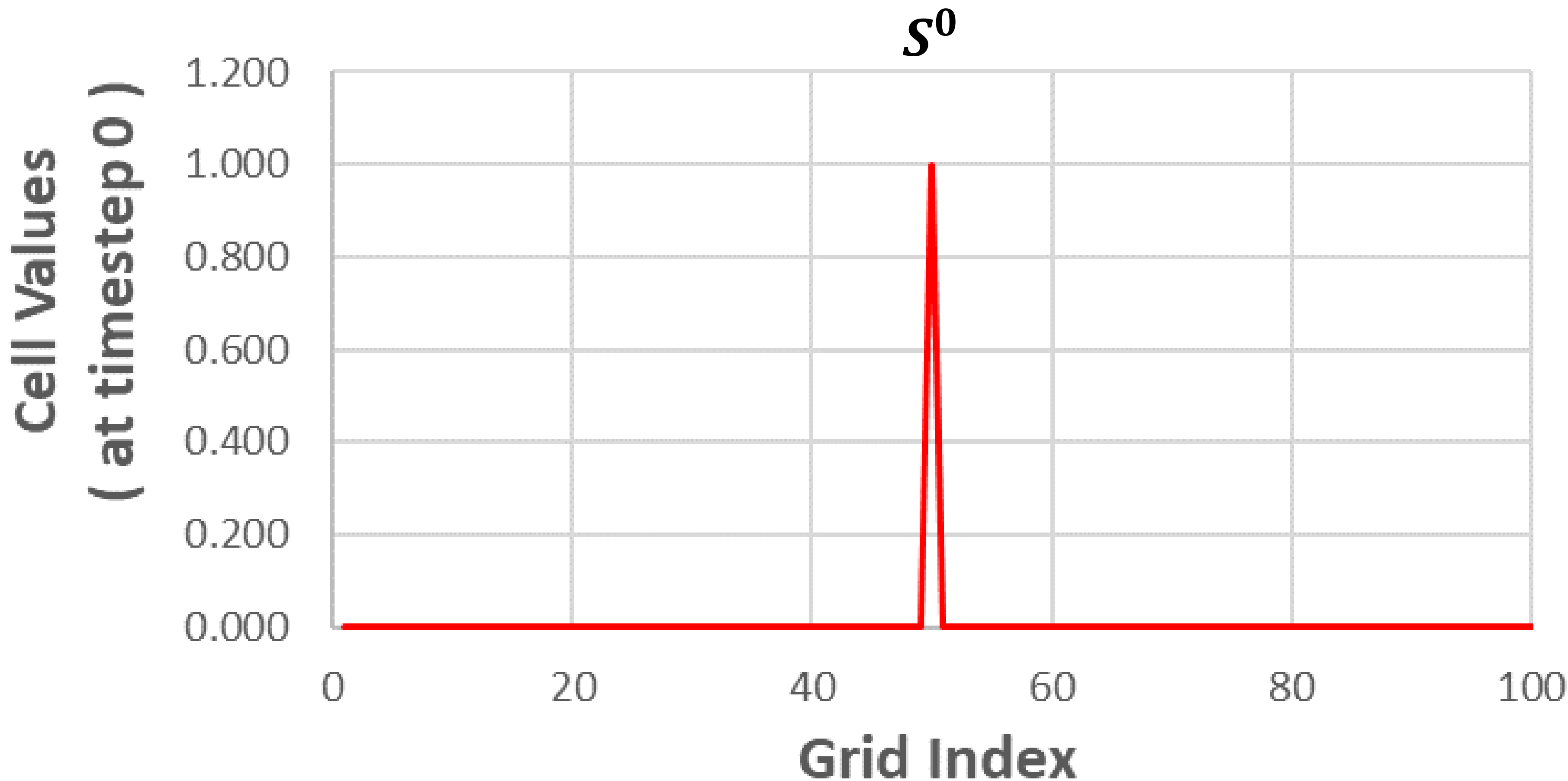


S^T Approximates a Gaussian

5-point asymmetric stencil:

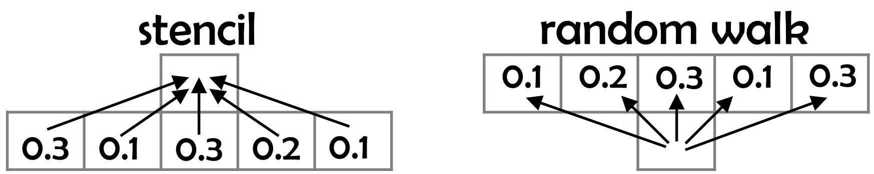


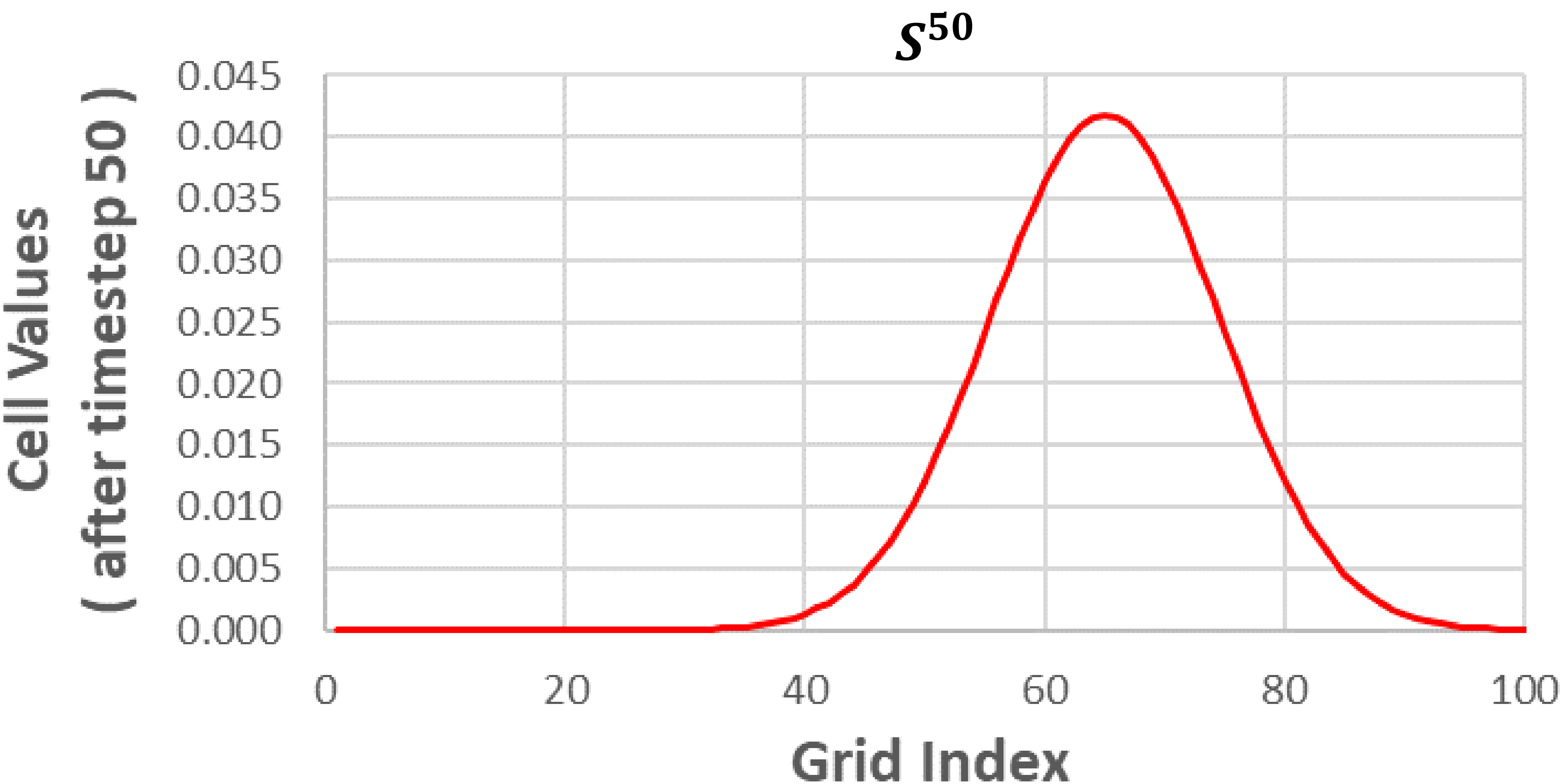




S^T Approximates a Gaussian

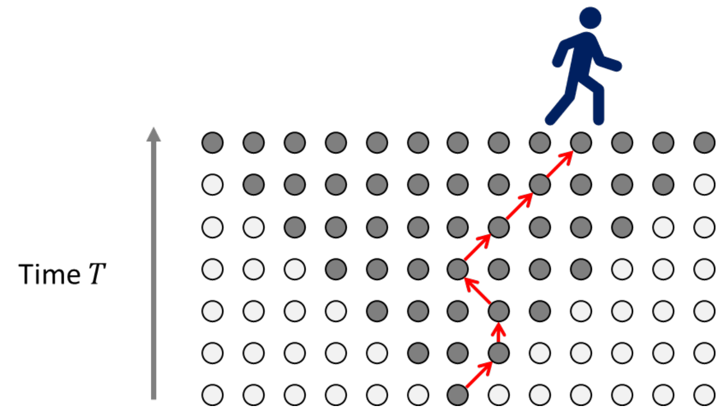
5-point asymmetric stencil:





S^T Approximates a Gaussian

A linear stencil with nonnegative weights can be modeled as a random walk on an integer grid.



Given a d-dimensional stencil S we can compute the mean vector μ_T and covariance matrix Σ_T of S^T in $O(1)$ time.

Then S^T approximates the following Gaussian:

$$G_{\mu_T,\Sigma_\mathrm{T}}(x) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma_\mathrm{T})}} e^{-\frac{1}{2}(x-\mu_T)^{trans}\,\Sigma_\mathrm{T}^{-1}(x-\mu_T)}$$

S^T Approximates a Gaussian

Let S be a linear stencil with

expectation 0 (symmetric)

variance σ^2

skewness ρ

Let $G_{T\sigma^2}(x)$ be a Gaussian with

expectation 0

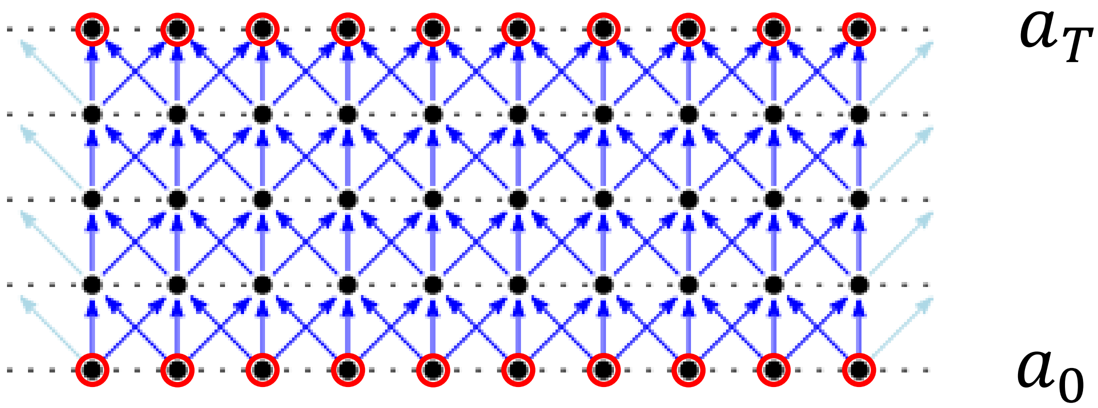
variance $T\sigma^2$

Then for any integer x:

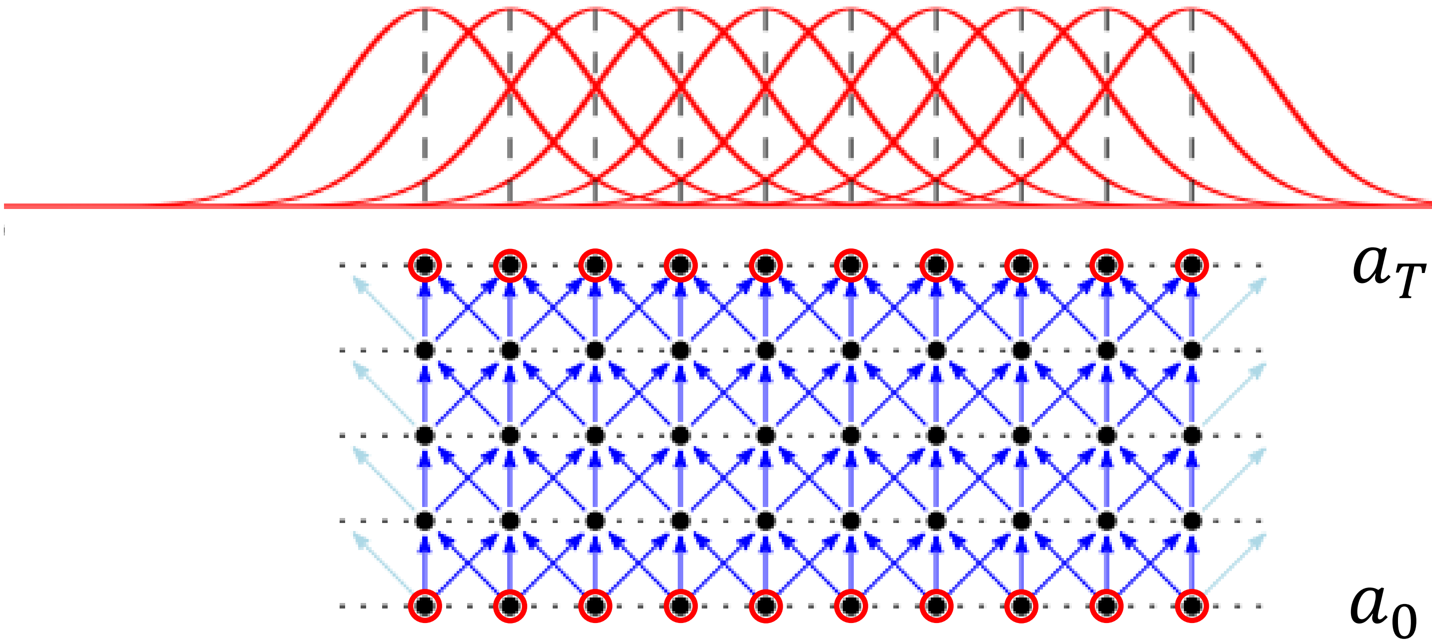
$$\boxed{\left|S^T[x] - G_{T\sigma^2}(x)\right| \leq \frac{1}{\sigma\sqrt{T}}\left(\frac{2C\rho}{\sigma} + \frac{1}{\sqrt{2\pi e}}\right)}$$

where, $C \leq 0.4748$.

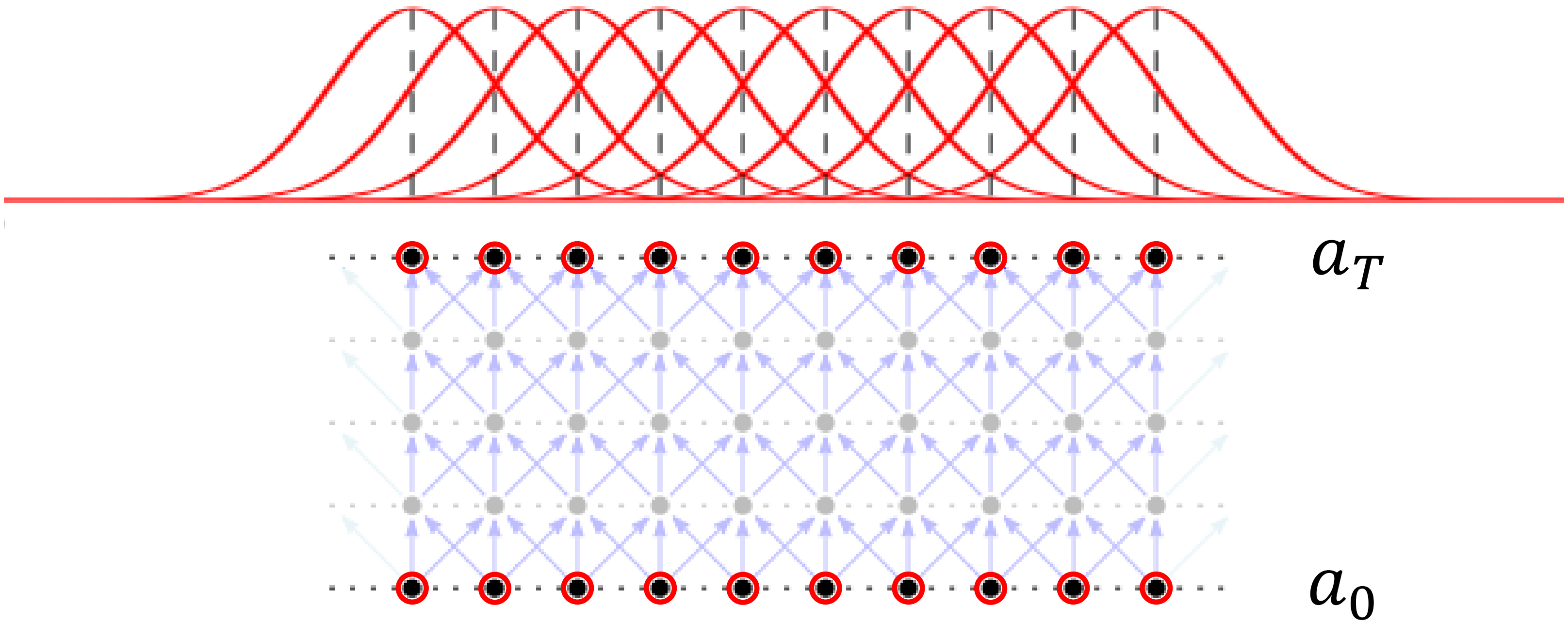
Algorithm for Freespace Grids



Algorithm for Freespace Grids



Algorithm for Freespace Grids

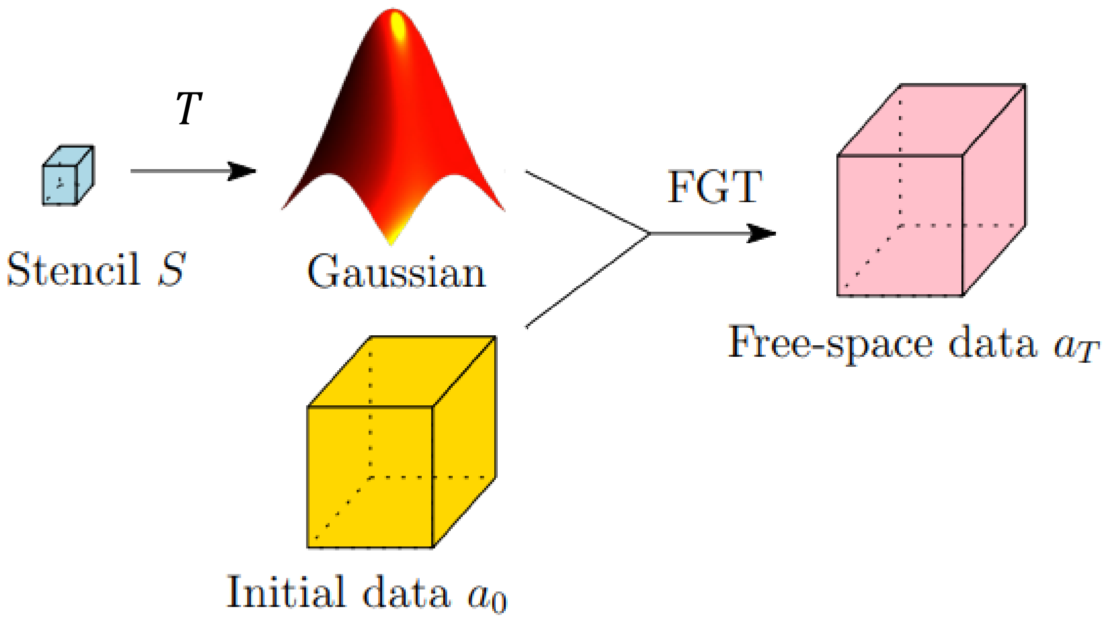


This is a very structured version of the n-body problem with a Gaussian interaction function!

So, we can use a fast multipole type algorithm to find an approximate solution efficiently.

The algorithm works by approximating the Gaussian S^T approximates!

Algorithm for Freespace Grids



Computational Complexity: $\Theta\left(N \log^{O(d)}\left(\frac{1}{\epsilon}\right) + \log T\right)$

(for a d-dimensional $N^{\frac{1}{d}} \times N^{\frac{1}{d}} \times \cdots \times N^{\frac{1}{d}}$ grid with constant d)

How About Nonlinear Stencils?

Binomial and Black-Scholes-Merton Option Pricing (American Options)

- Modeled as an aperiodic nonlinear 1D stencil computation problem with a max operator
- Requires $\mathrm{O}\left(N^2\right)$ work using standard algorithms, where N = #timesteps (= grid size)
- Can be reduced into linear 1D stencil computation problems through recursive decomposition
- The FFT-based periodic linear stencil algorithm for 1D grids $\Rightarrow$ the nonlinear American option pricing problem can be solved in $\mathrm{O}\left(N \log^2 N\right)$ work

Stencil Weights are Functions of Space and/or Time Coordinates

- Some of these problems can be reduced to a sequence of linear stencil computation problems
- Inside each linear stencil computation phase, we use fixed approximate stencil weights based on the actual space/time-dependent weights of the original stencil
- The final output approximates the final grid values

Some Open Problems

- Efficient algorithms for classes of nonlinear stencils.
- Low-span algorithms for aperiodic stencils.
- Algorithms for inhomogenous stencils.
- Stencil code generator that combines trapezoidal decomposition, tiling, FFT, Gaussians, etc. to achieve the best performance under a given set of constraints (e.g., error tolerance, space bound).

  We already have FOURST — a very primitive FFT-based generator.

GITHUB Repository for FFT-based Stencil Codes

https://github.com/TEAlab/FFTStencils