Software Performance Engineering

LECTURE 14
Cache-Oblivious Algorithms

Xuhao Chen

November 13, 2025

Acknowledgment

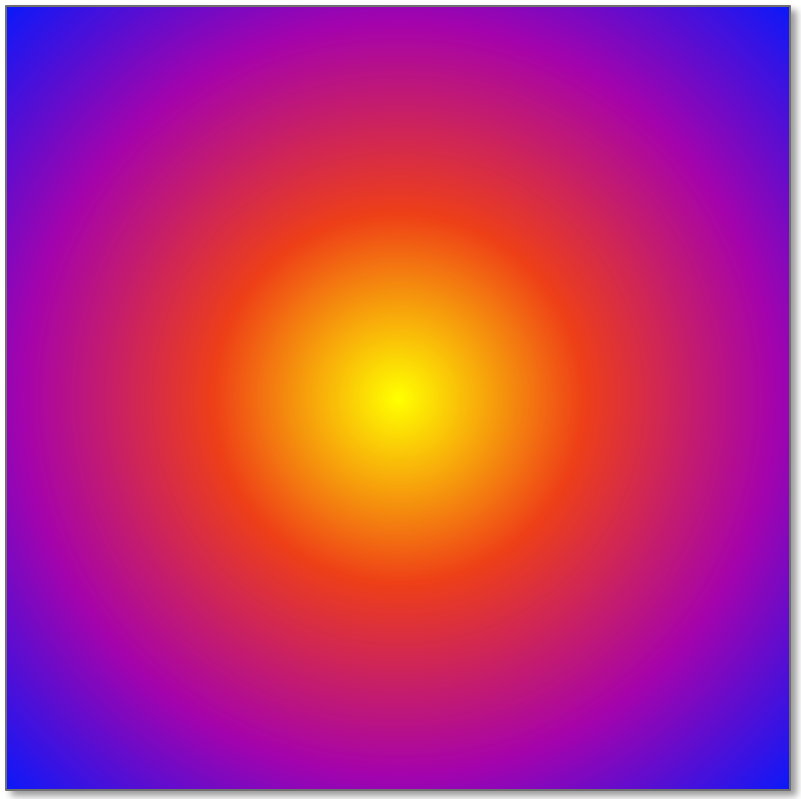
A presentation by Matteo Frigo inspired some of these slides.

Outline

- Simulation of Heat Diffusion → Stencil Computation
- Cache-oblivious Stencil Computation
- Parallelizing Cache-oblivious Stencil Computation
- Summary

Heat Diffusion

- 2D heat equation*



The heat function `u(t,x,y)` is the heat at time `t` of a point `(x,y)`.

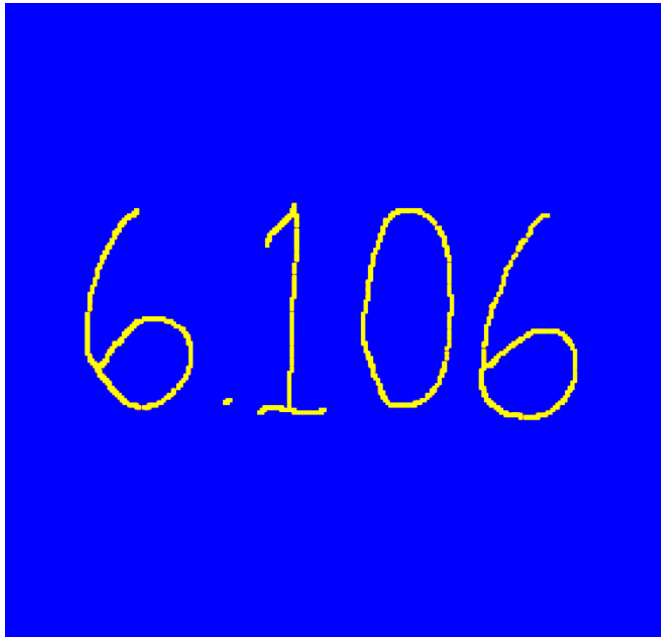
$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

Laplacian

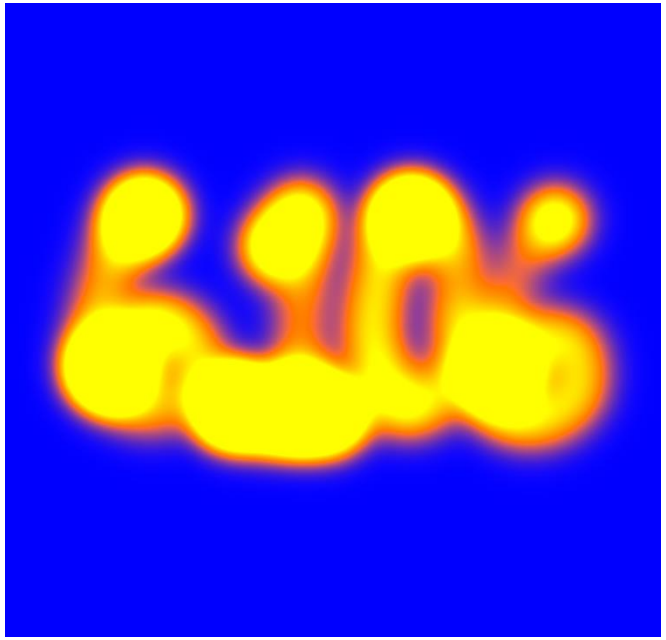
α is the thermal diffusivity.

* originally formulated by Joseph Fourier in 1807

2D Heat-Diffusion Simulation



Before



After

1D Heat Equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

boundary condition (hot)

boundary condition (cold)

Finite-Difference Method

- The famous Swiss mathematician Leonhard Euler (1707–1783) invented the finite-difference method around 1768.

We owe Euler credit for the notations

- $f(x)$ for a function,
- e for the base of the natural logarithm,
- i for the square root of -1,
- π for the area of a unit circle,
- $\sum$ for summation,
- Δ for finite differences.

Finite-Difference Approximation

$$\frac{\partial}{\partial t}u(t,x) \approx \frac{u(t+\Delta t,x)-u(t,x)}{\Delta t},$$

$$\frac{\partial}{\partial x}u(t,x) \approx \frac{u(t,x)-u(t,x-\Delta x)}{\Delta x},$$

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

1D heat equation

$$\frac{\partial^2}{\partial x^2}u(t,x) \approx \frac{\frac{\partial}{\partial x}u(t,x+\Delta x)-\frac{\partial}{\partial x}u(t,x)}{\Delta x}$$

$$\approx \frac{\dfrac{u(t,x+\Delta x)-u(t,x)}{\Delta x} - \dfrac{u(t,x)-u(t,x-\Delta x)}{\Delta x}}{\Delta x}$$

$$\approx \frac{u(t,x+\Delta x)-2u(t,x)+u(t,x-\Delta x)}{(\Delta x)^2}.$$

Discretized Heat Equation

$$\frac{u(t+\Delta t,x)-u(t,x)}{\Delta t}=\alpha\left(\frac{u(t,x+\Delta x)-2u(t,x)+u(t,x-\Delta x)}{(\Delta x)^2}\right)$$

Now, put the one term involving $t+\Delta t$ on the left and the other terms involving just t on the right:

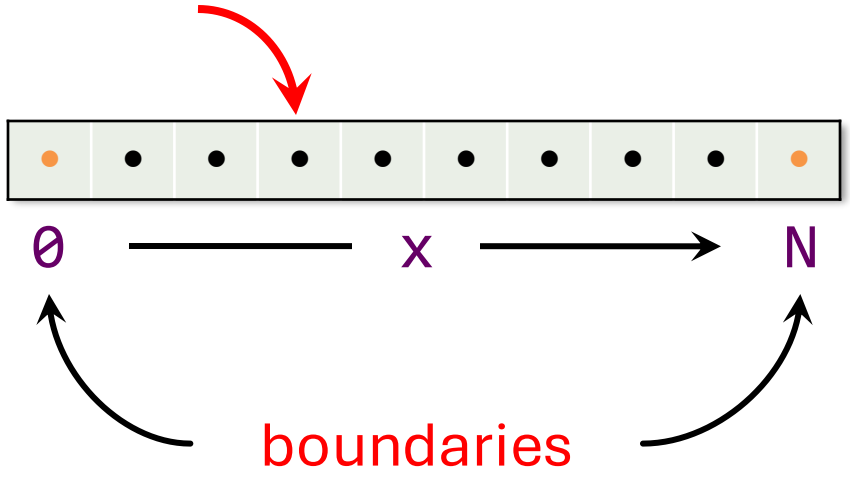
$$u(t+\Delta t,x)=u(t,x)+\alpha\Delta t\left(\frac{u(t,x+\Delta x)-2u(t,x)+u(t,x-\Delta x)}{(\Delta x)^2}\right)$$

Assuming that $\Delta t = 1$ and $\Delta x = 1$, we obtain the following code for the update rule:

```
u[t+1][x] = u[t][x] + ALPHA * (u[t][x+1] - 2*u[t][x] + u[t][x-1]);
```

3-Point Stencil

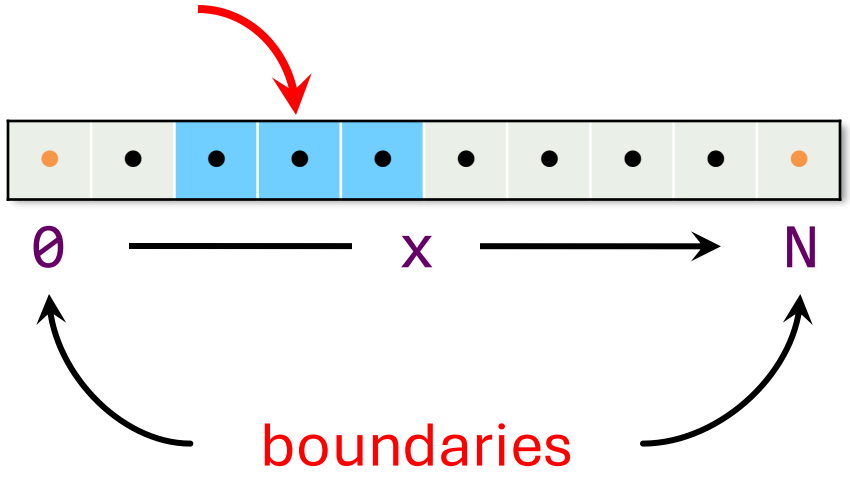
```
u[t+1][x] = u[t][x] + ALPHA * (u[t][x+1] - 2*u[t][x] + u[t][x-1]);
```



A stencil computation updates each point in an array by a fixed pattern, called a stencil.

3-Point Stencil

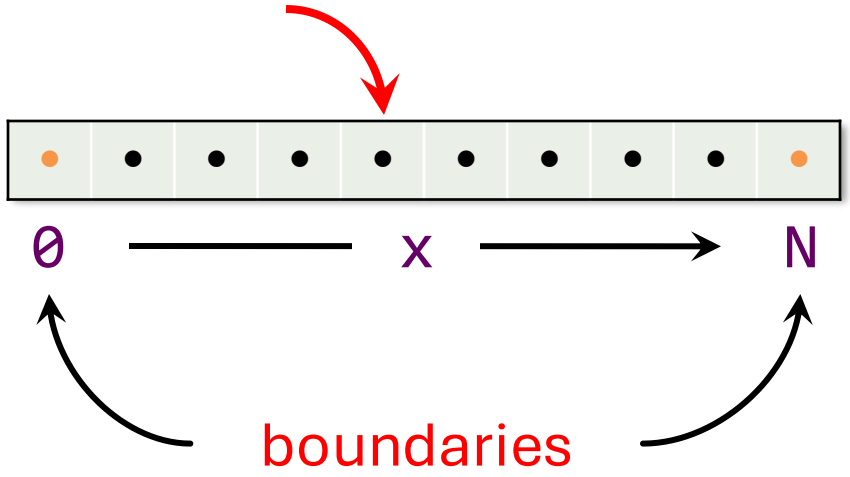
```
u[t+1][x] = u[t][x] + ALPHA * (u[t][x+1] - 2*u[t][x] + u[t][x-1]);
```



A stencil computation updates each point in an array by a fixed pattern, called a stencil.

3-Point Stencil

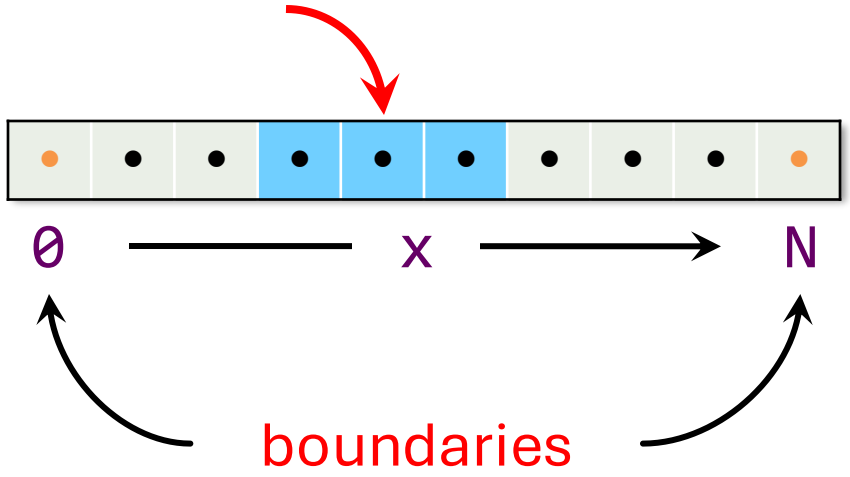
```
u[t+1][x] = u[t][x] + ALPHA * (u[t][x+1] - 2*u[t][x] + u[t][x-1]);
```



A stencil computation updates each point in an array by a fixed pattern, called a stencil.

3-Point Stencil

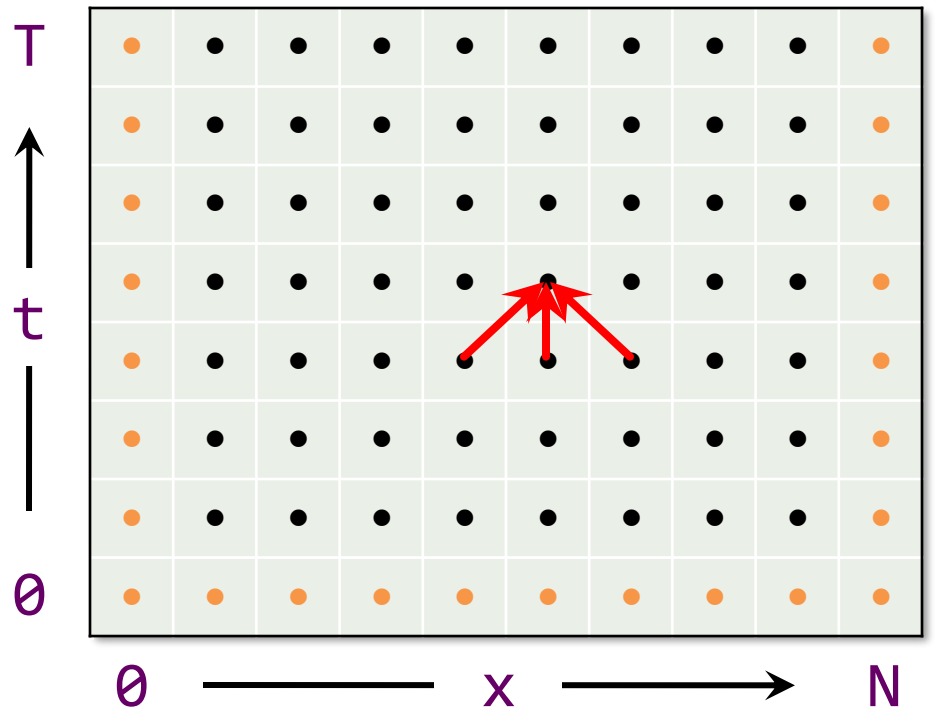
```
u[t+1][x] = u[t][x] + ALPHA * (u[t][x+1] - 2*u[t][x] + u[t][x-1]);
```



A stencil computation updates each point in an array by a fixed pattern, called a stencil.

3-Point Stencil

```
u[t+1][x] = u[t][x] + ALPHA * (u[t][x+1] - 2*u[t][x] + u[t][x-1]);
```

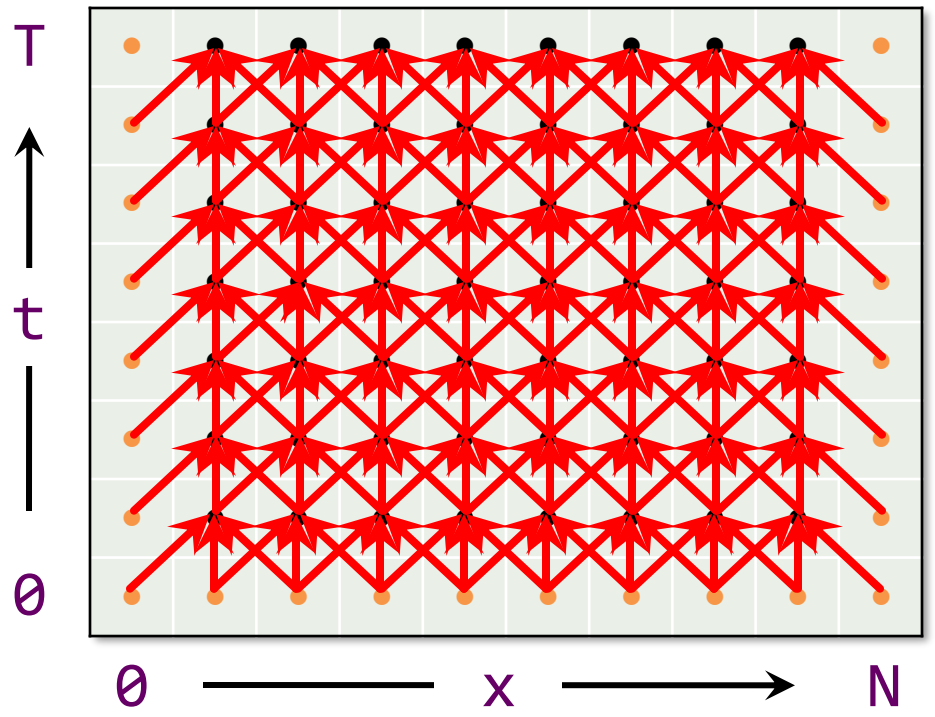


A stencil computation updates each point in an array by a fixed pattern, called a stencil.

iteration space

3-Point Stencil

```
u[t+1][x] = u[t][x] + ALPHA * (u[t][x+1] - 2*u[t][x] + u[t][x-1]);
```



A stencil computation updates each point in an array by a fixed pattern, called a stencil.

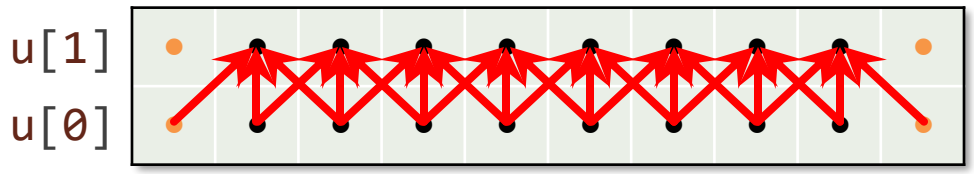
iteration space

3-Point Stencil Code

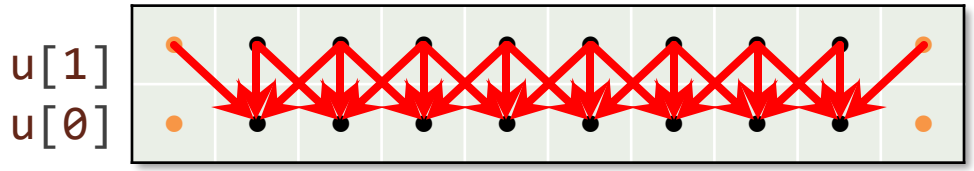
```
double u[2][N]; // even-odd trick

static inline double kernel(double * w) {
    return w[0] + ALPHA * (w[-1] - 2*w[0] + w[1]);
}

for (size_t t = 0; t < T-1; ++t) { // time loop
  for(size_t x = 1; x < N; ++x)  // space loop
    u[(t+1)%2][x] = kernel( &u[t%2][x] );
```

pointer

Even time step



Odd time step

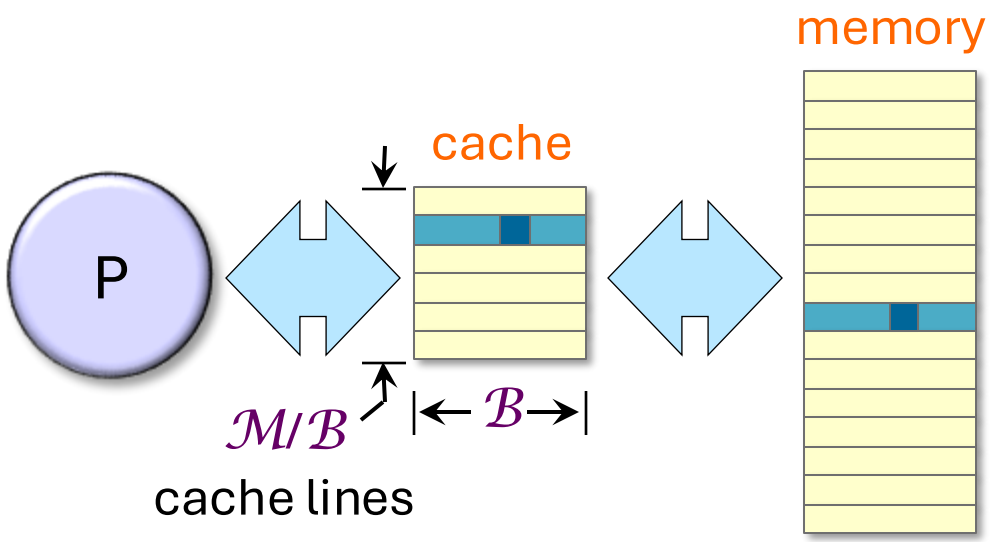


Outline

- Simulation of Heat Diffusion → Stencil Computation
- Cache-oblivious Stencil Computation
- Parallelizing Cache-oblivious Stencil Computation
- Summary

Recall: Ideal-Cache Model

- Parameters
  - Two-level hierarchy
  - Cache size of $\mathcal{M}$ bytes
  - Cache-line size of $\mathcal{B}$ bytes
  - Fully associative
  - Optimal omniscient replacement, or LRU.

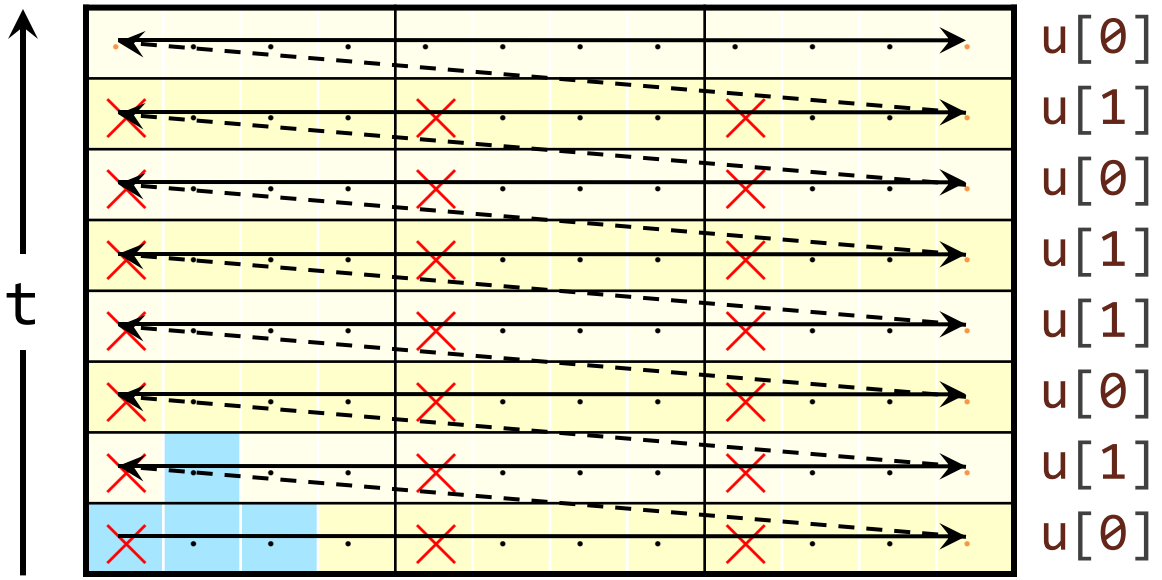


Performance Measures

- work T_1 (ordinary running time)
- cache misses Q

Cache Behavior of Looping

```
double u[2][N]; // even-odd trick

static inline double kernel(double * w) {
    return w[0] + ALPHA * (w[-1] - 2*w[0] + w[1]);
}

for (size_t t = 0; t < T-1; ++t) { // time loop
  for(size_t x = 1; x < N; ++x)  // space loop
    u[(t+1)%2][x] = kernel( &u[t%2][x] );
```



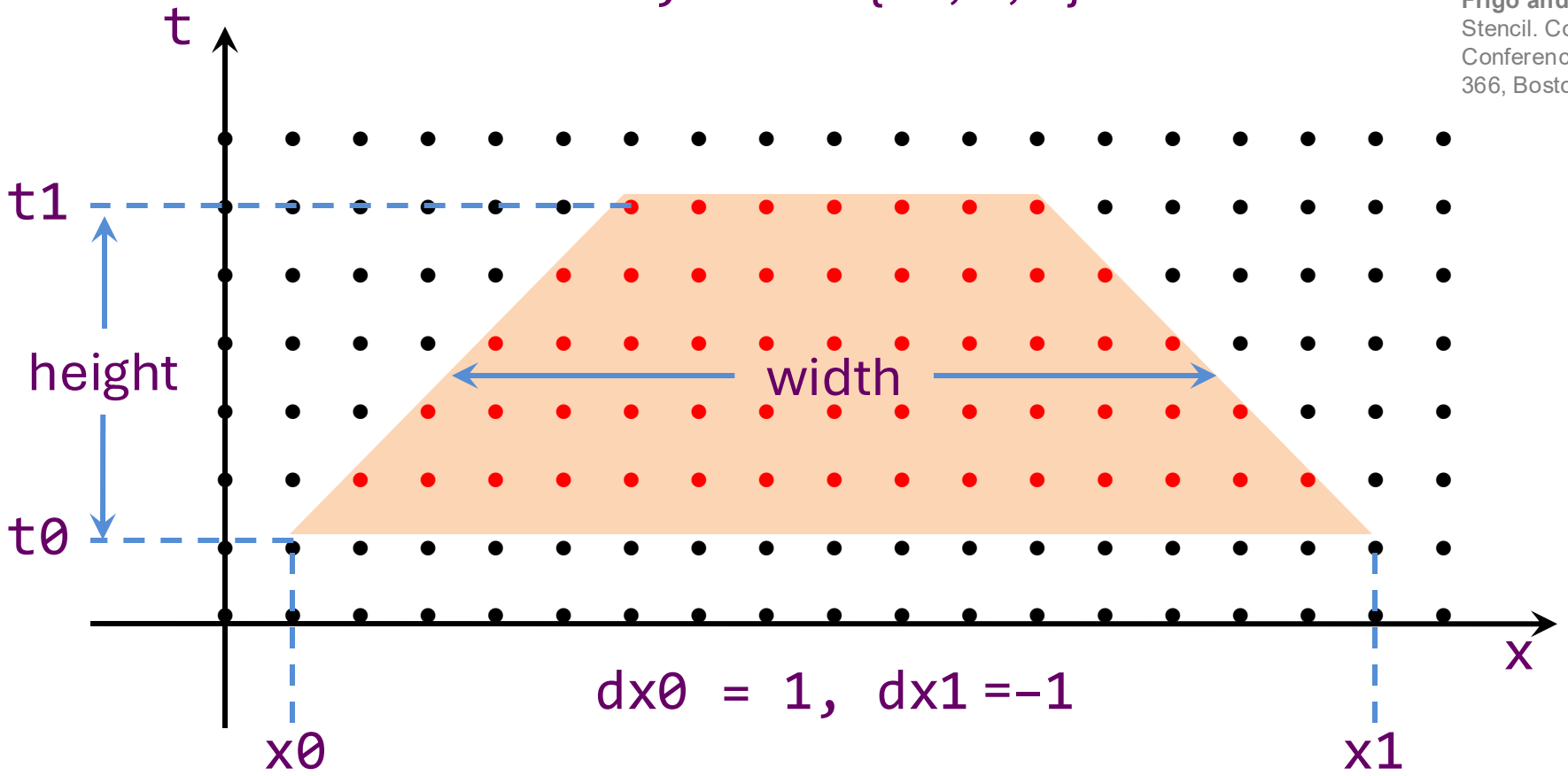
Assume that $N > \mathcal{M}$ and we use LRU replacement

Then $Q = \Theta(NT/\mathcal{B})$.

Cache-Oblivious 3-Point Stencil

Recursively traverse trapezoidal regions of space-time points (t,x) such that

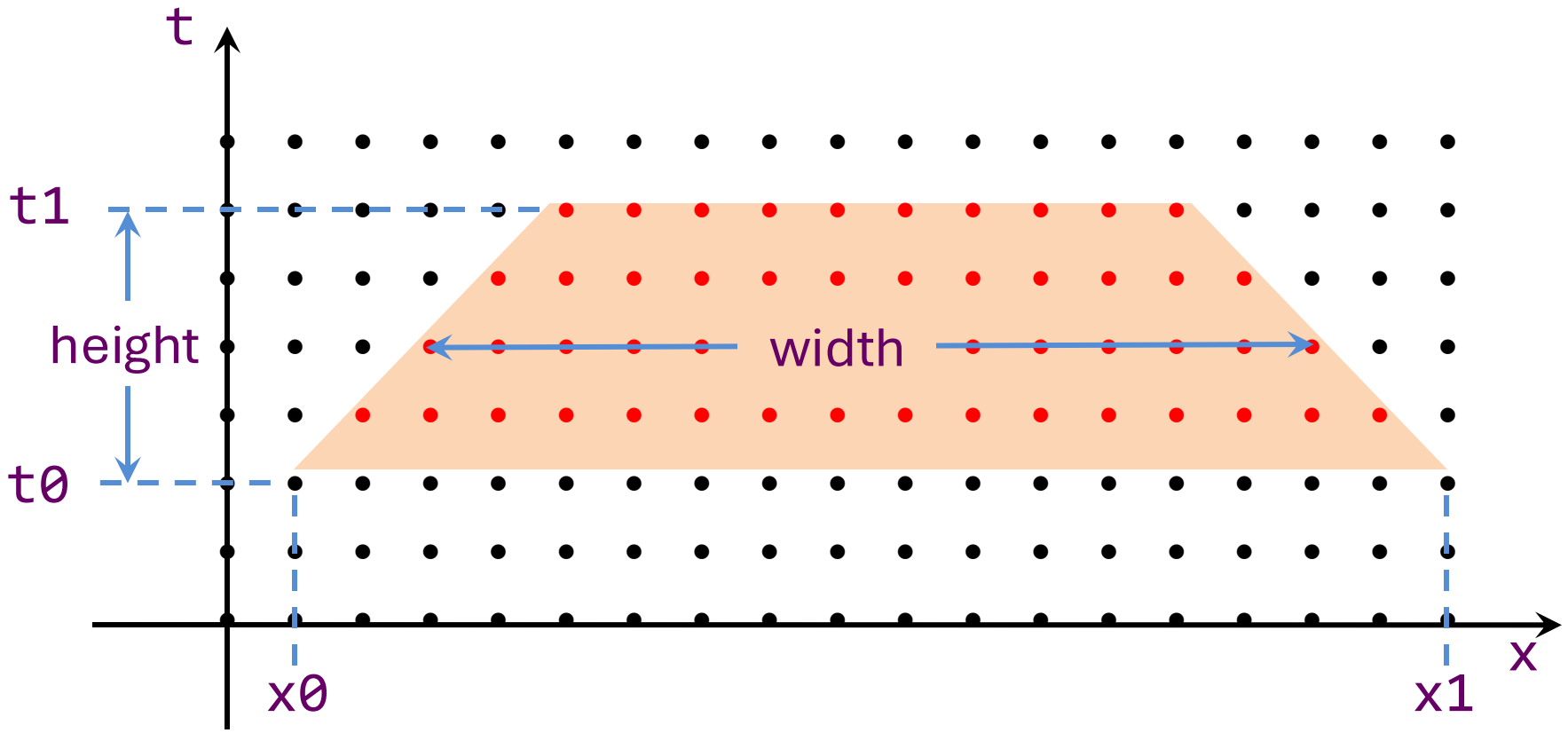
$$t_0 \le t < t_1$$
$$x_0 + dx_0(t-t_0) \le x < x_1 + dx_1(t-t_0)$$
$$dx_0,\ dx_1 \in \{-1, 0, 1\}$$



Frigo and V. Strumpen. Cache Oblivious Stencil. Computations. In International Conference on. Supercomputing, pages 361–366, Boston, Massachusetts, June 2005

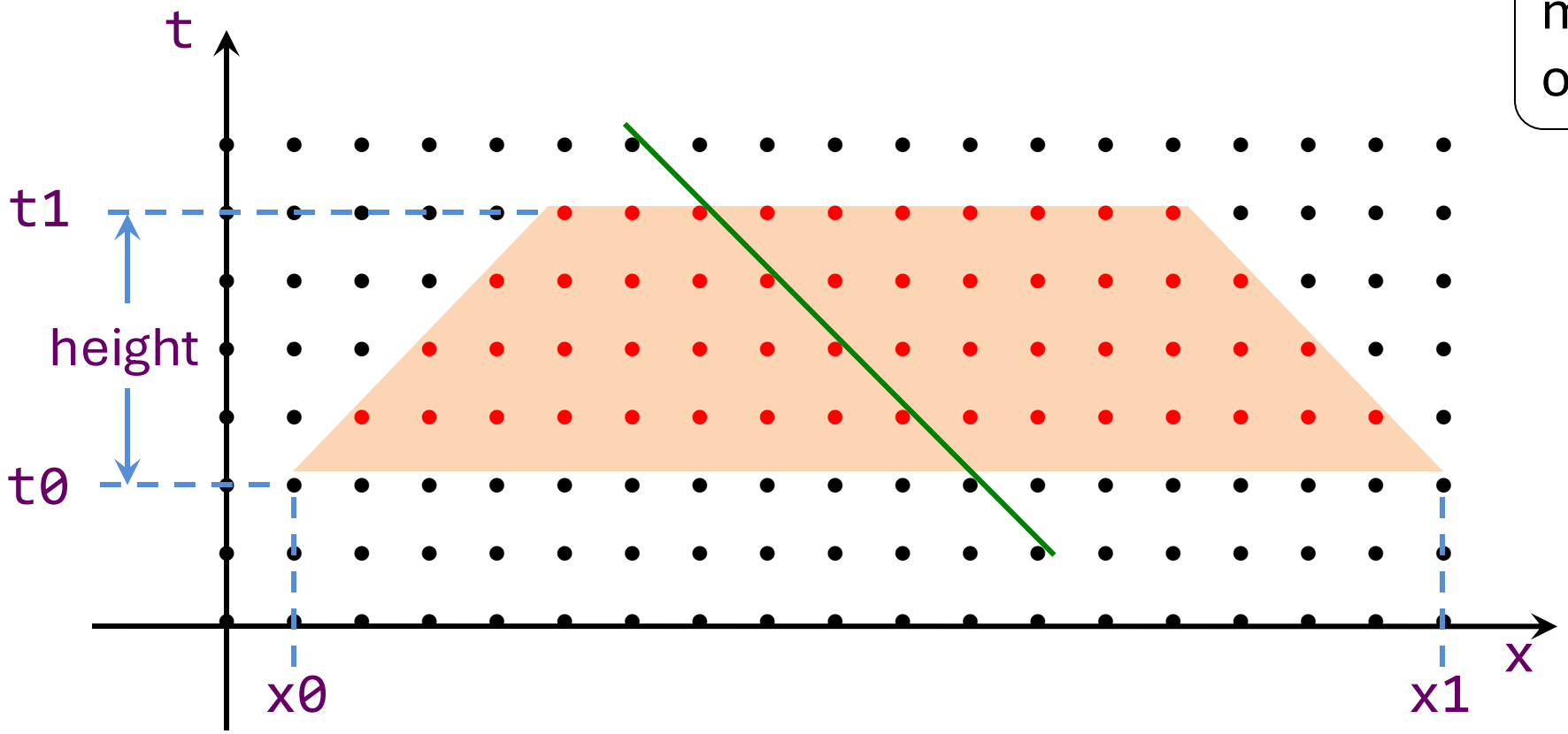
Squat Trapezoid: Space Cut

- If width ≥ 2·height, cut the trapezoid



Squat Trapezoid: Space Cut

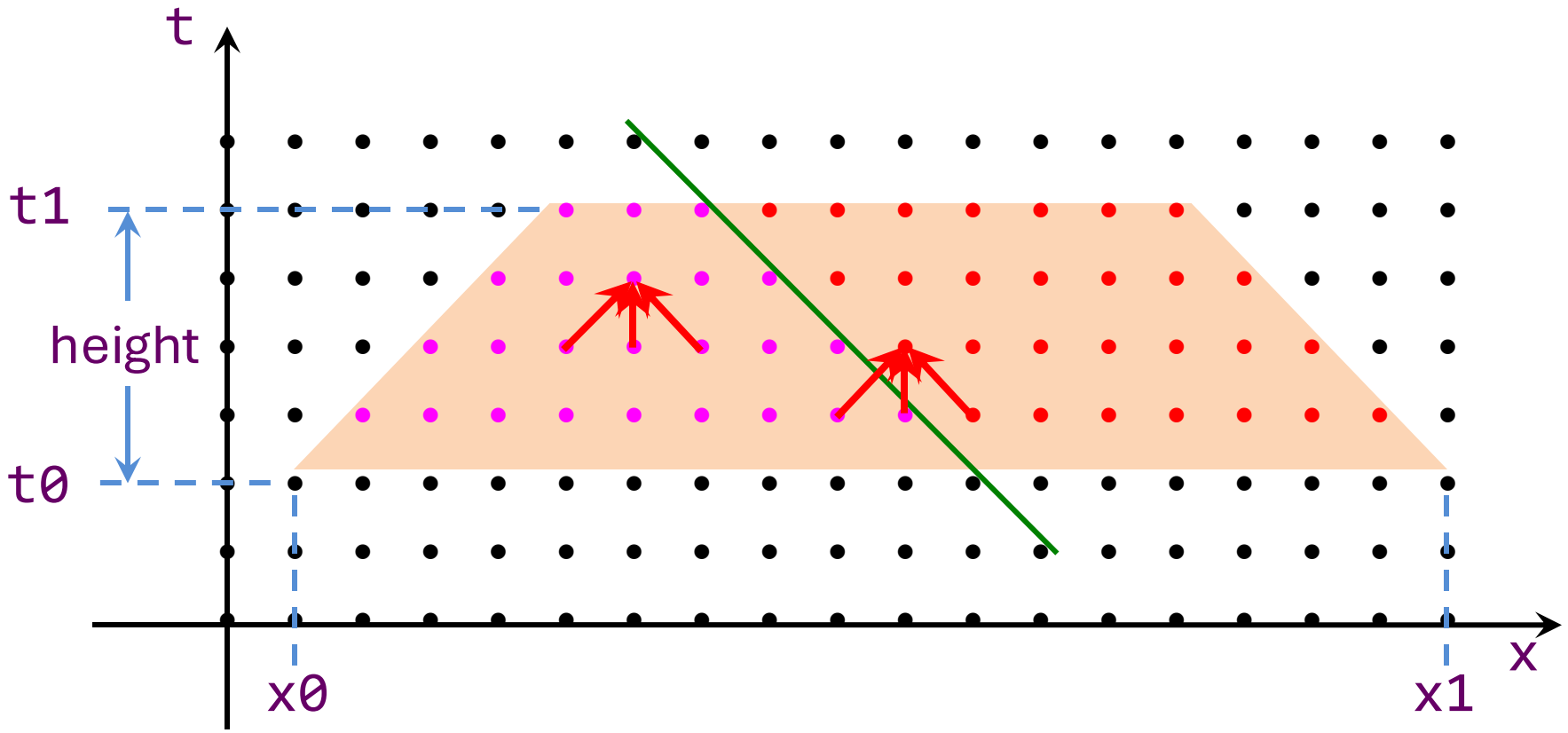
- If width ≥ 2·height, cut the trapezoid with a line of slope -1 through the center



middle point of middle row

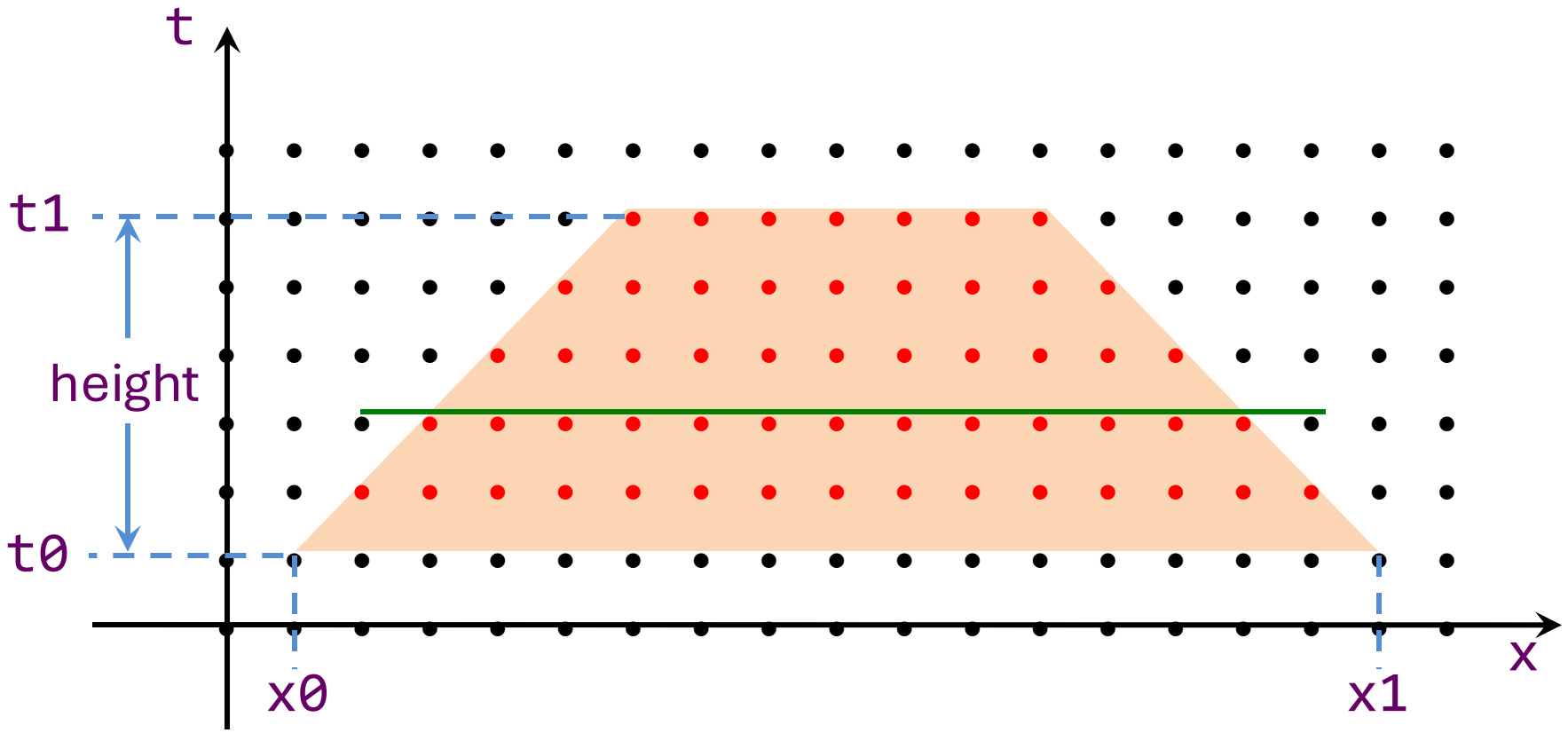
Squat Trapezoid: Space Cut

- If width ≥ 2·height, cut the trapezoid with a line of slope −1 through the center
- Traverse the trapezoid on the left first, and then the one on the right.



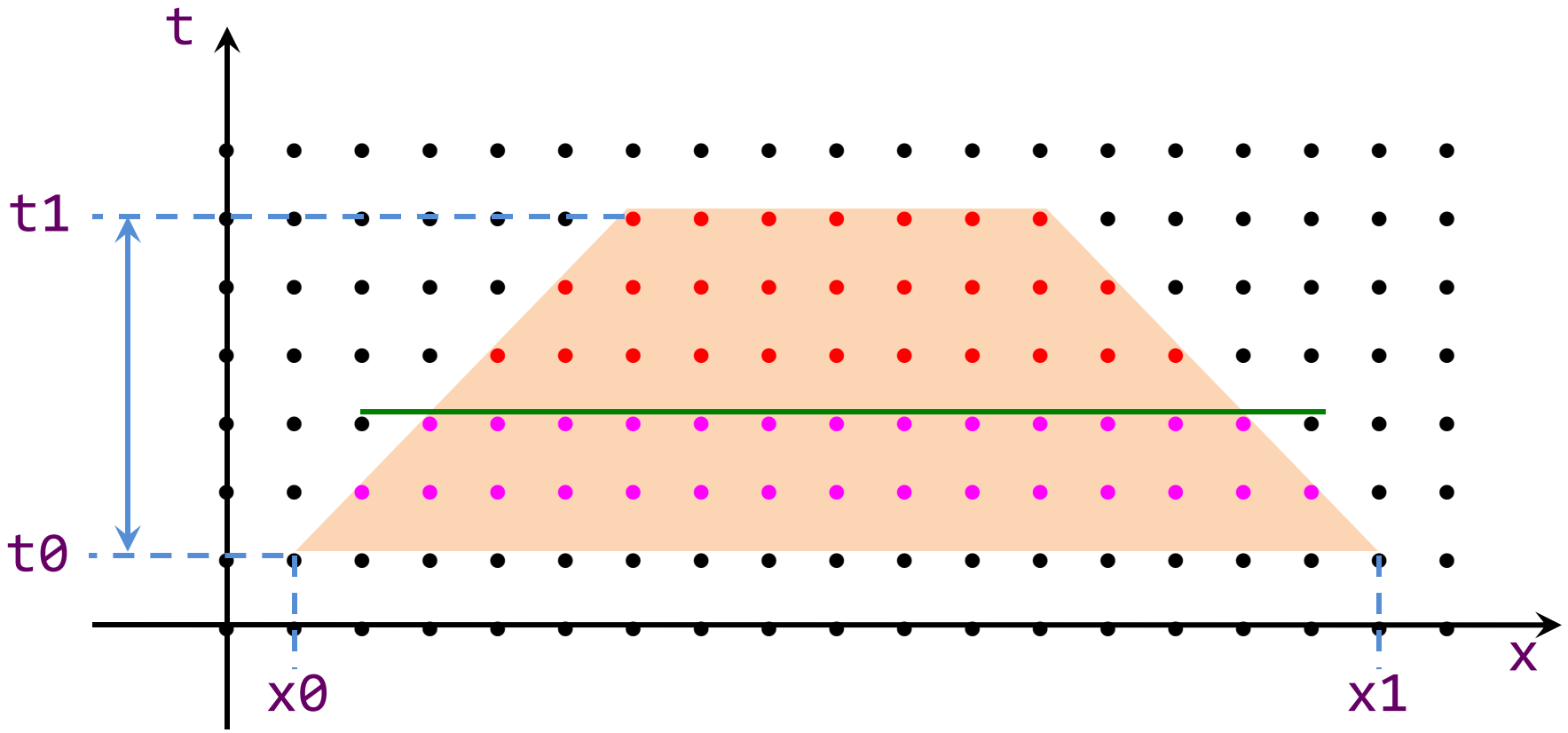
Tall Trapezoid: Time Cut

- If width < 2·height, cut the trapezoid with a horizontal line through the center



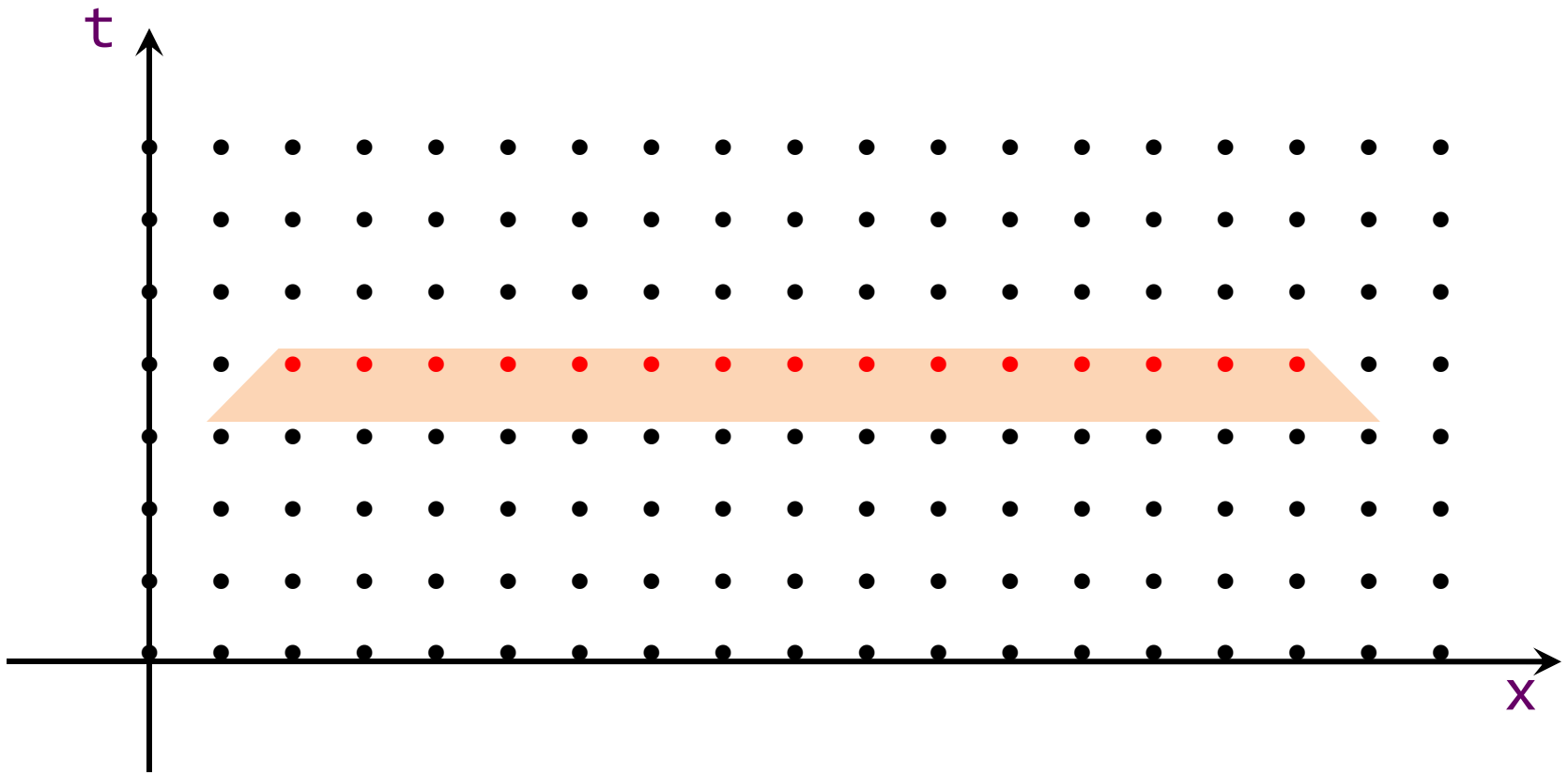
Tall Trapezoid: Time Cut

- If width < 2·height, cut the trapezoid with a horizontal line through the center
- Traverse the bottom trapezoid first, and then the top one



Base Case

- If height = 1, compute all space-time points in the trapezoid.
- Any order of computation is valid, since no point depends on another

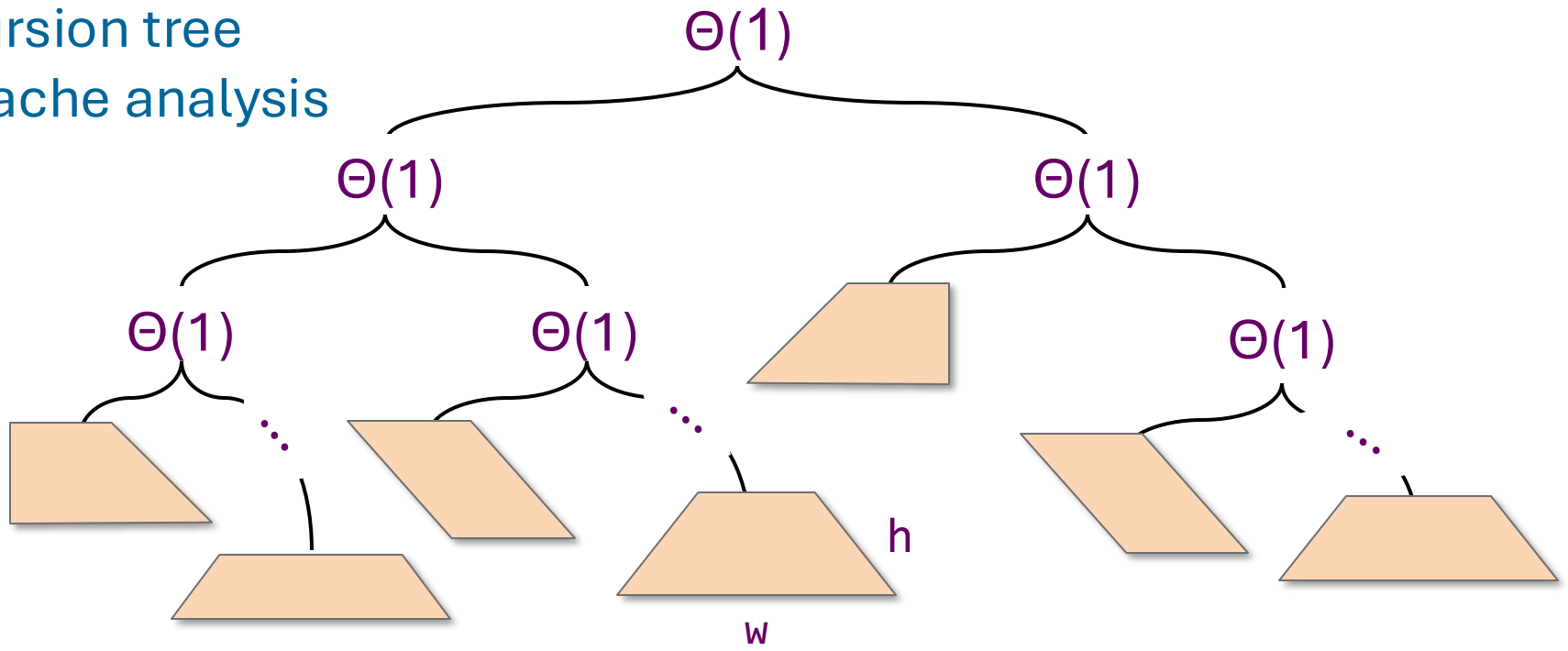


C Implementation

```
void trapezoid(int64_t t0, int64_t t1, //time start and end
               int64_t x0, int64_t dx0, //left pt of base & "slope"
               int64_t x1, int64_t dx1){//rt pt of base & "slope"
  int64_t h = t1 - t0; //trapezoid height
  if (h == 1) { //base case
      for (int64_t x = x0; x < x1; x++)
        u[t1%2][x] = kernel( &u[t0%2][x] ); //same as in looping
  } else if (h > 1) {
    if (2*(x1 - x0) + (dx1 - dx0) * h >= 4*h) { //space cut
      int64_t xm = (2*(x0 + x1) + (dx0 + dx1 + 2)*h) / 4;
      trapezoid(t0, t1, x0, dx0, xm, -1); //left
      trapezoid(t0, t1, xm, -1, x1, dx1); //right
    } else { //time cut
      int64_t half_h = h / 2;
      trapezoid(t0, t0 + half_h, x0, dx0, x1, dx1); //bottom
      trapezoid(t0 + half_h, t1,
                x0 + dx0 * half_h,
                dx0, x1 + dx1 * half_h, dx1); //top
    }
  }
}
```

Work and Cache Analysis

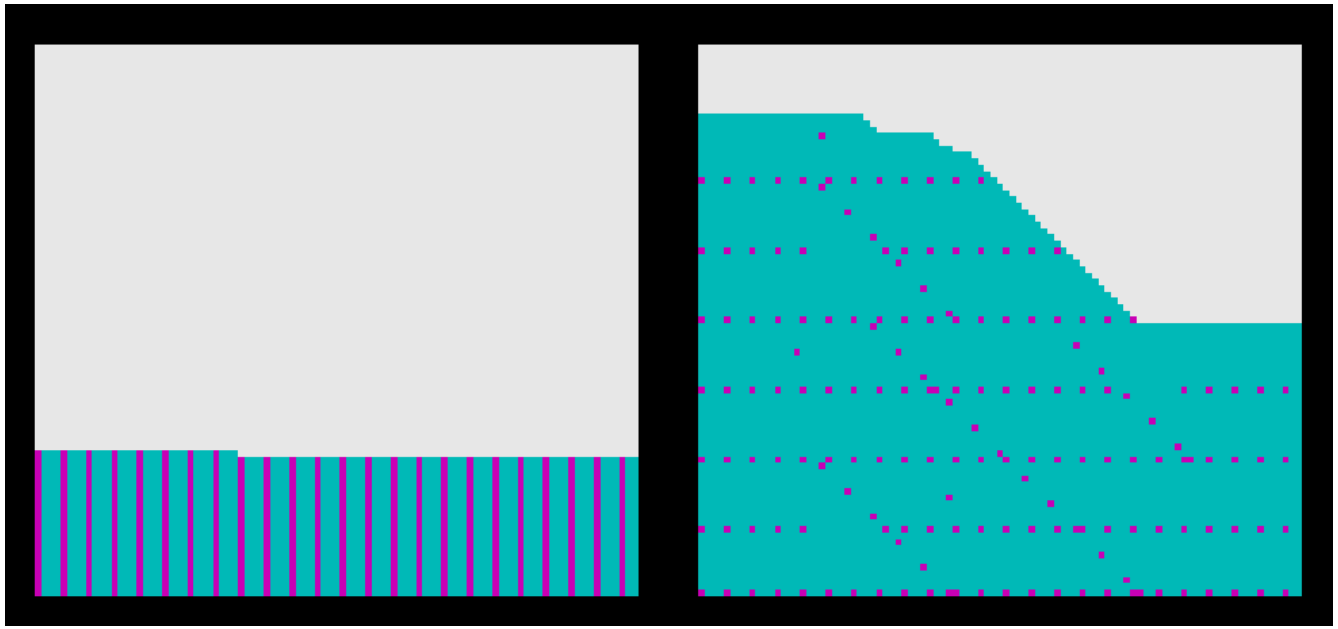
Recursion tree
for cache analysis



- The bottom of a leaf trapezoid just fits in the cache, so $w = \Theta(\mathcal{M})$.
- A leaf trapezoid contains $\Theta(hw) = \Theta(w^2)$ points and $\Theta(w^2)$ work.
- Since $w \le \mathcal{M}$, a leaf incurs $\Theta(w/\mathcal{B})$ cache misses.
- There are $\Theta(NT/hw) = \Theta(NT/w^2)$ leaves and internal nodes.
- The internal nodes contribute little to both work and cache misses.
- Work = $\Theta(NT/w^2) \cdot \Theta(w^2) = \Theta(NT)$.
- Cache misses = $\Theta(NT/w^2) \cdot \Theta(w/\mathcal{B}) = \Theta(NT/\mathcal{B}w) = \Theta(NT/\mathcal{B}\mathcal{M})$.

Simulation: 3-Point Stencil

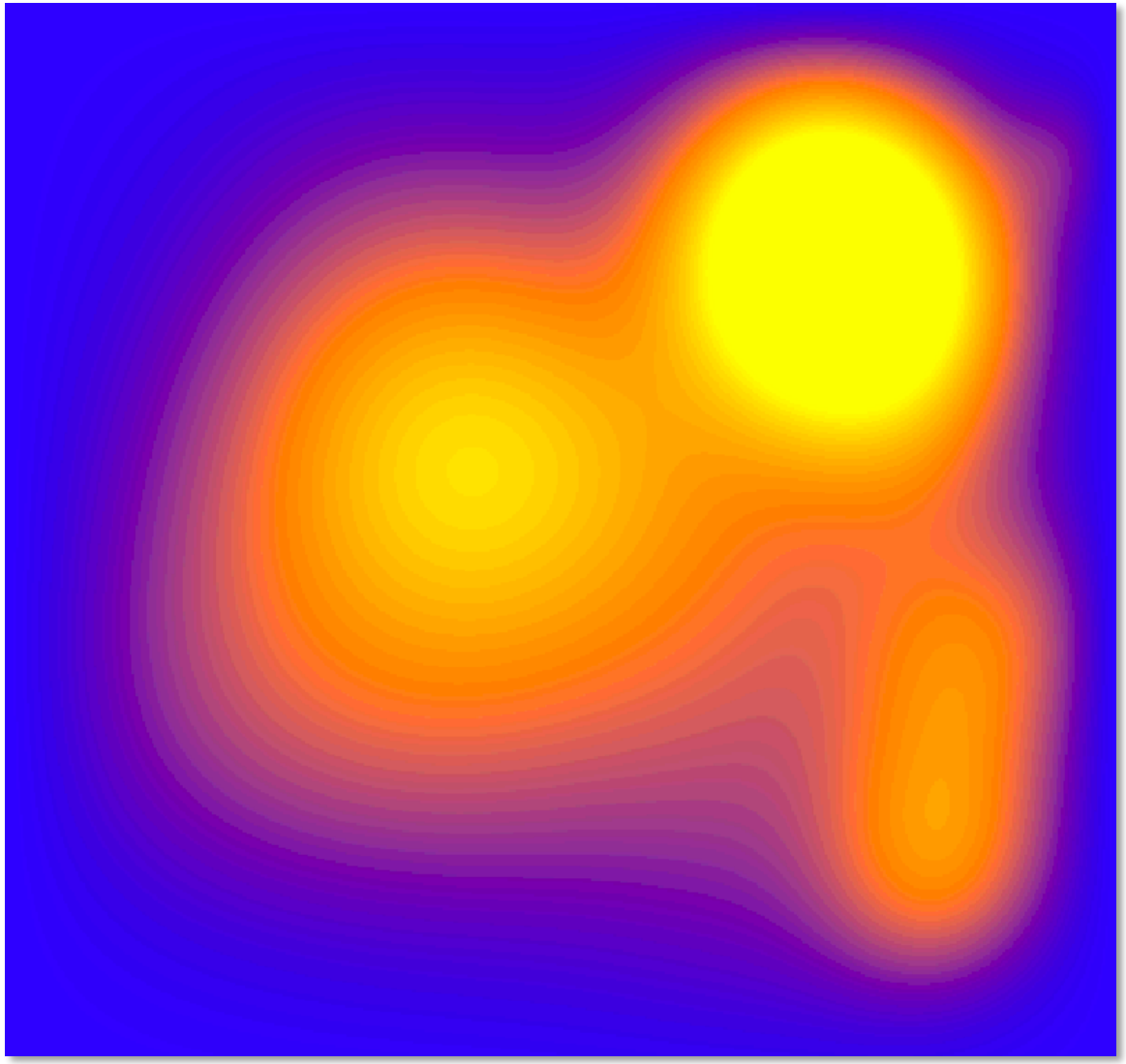
- Rectangular region
  - N = 95
  - T = 87



Looping Divide-and-conquer

- Fully associative LRU cache
  - cache line $\mathcal{B}$ = 4 points
  - cache size $\mathcal{M}$ = 32 points = 8 cache lines
- Cache-hit latency = 1 cycle
- Cache-miss latency = 10 cycles

Looping vs. Trapezoid on Heat



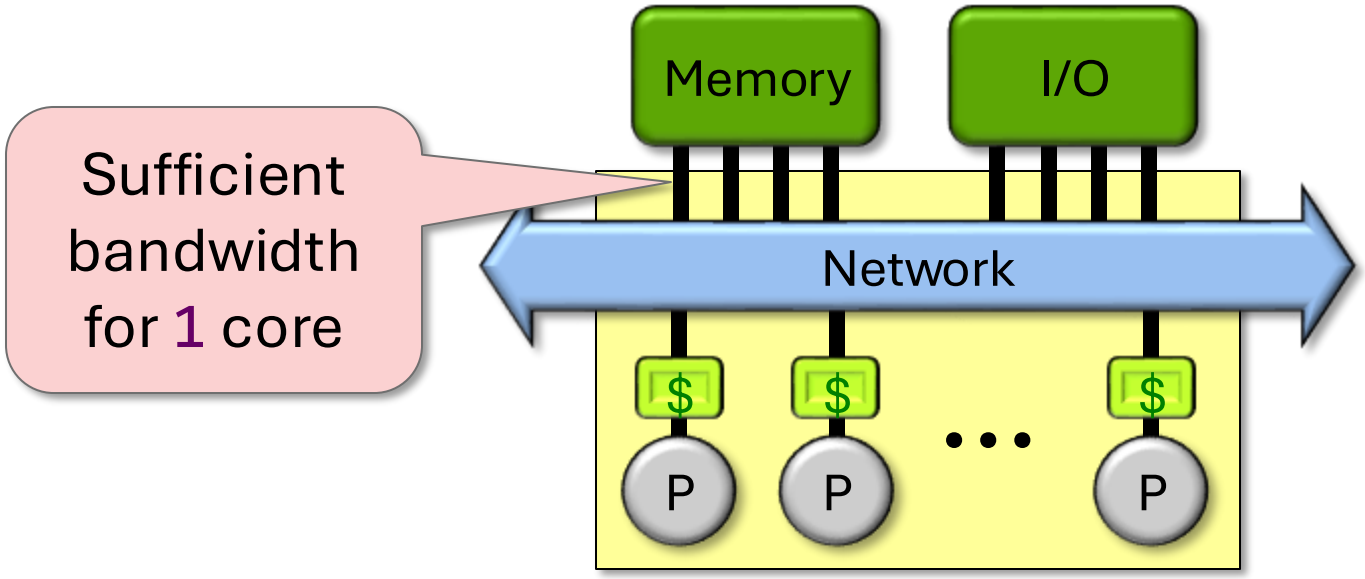
Impact on Performance

cache-oblivious algorithm incurs way fewer cache misses than looping version

Q. Why no performance gain?

A. Prefetching and a good memory architecture.

The memory bandwidth for one core largely suffices.

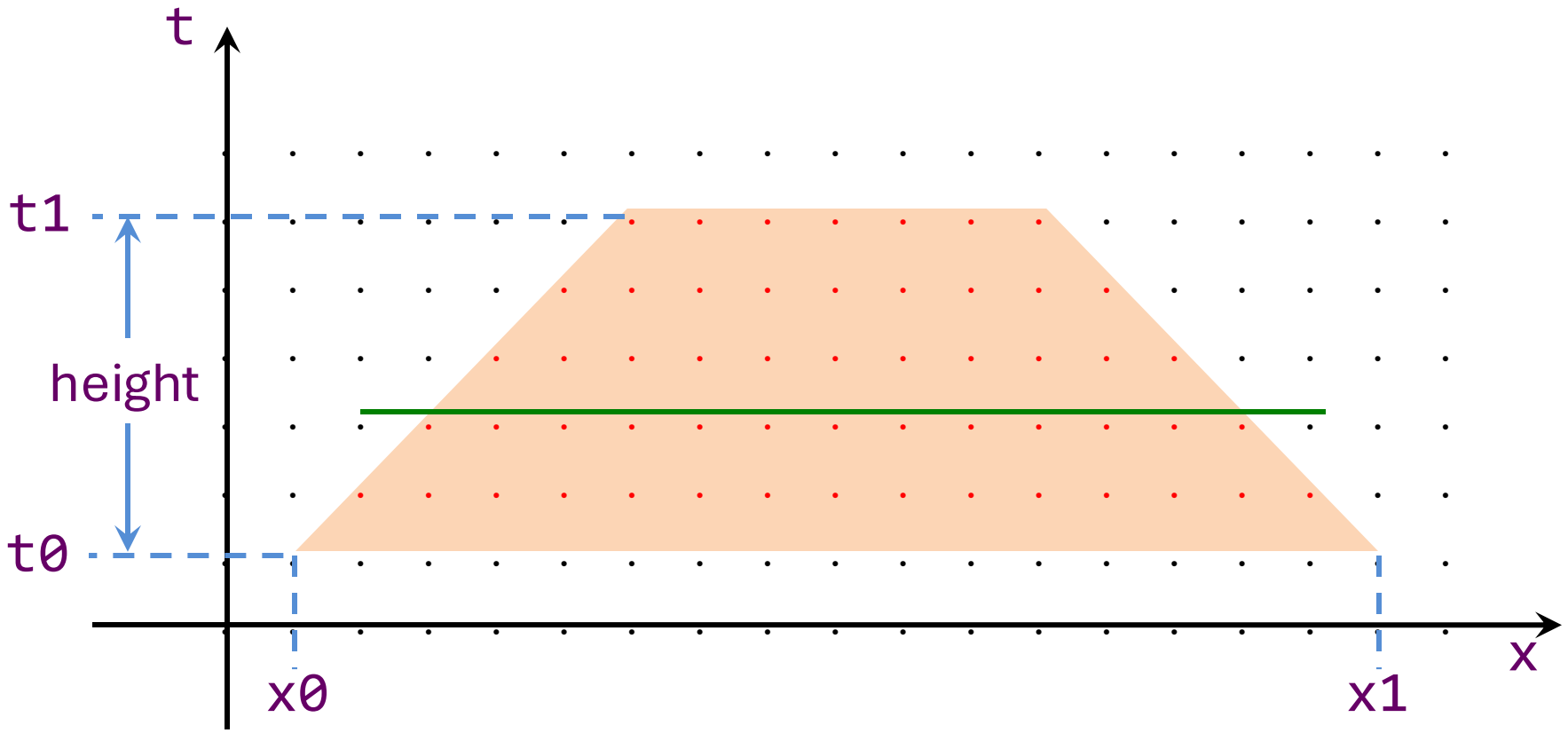


Outline

- Simulation of Heat Diffusion → Stencil Computation
- Cache-oblivious Stencil Computation
- Parallelizing Cache-oblivious Stencil Computation
- Summary

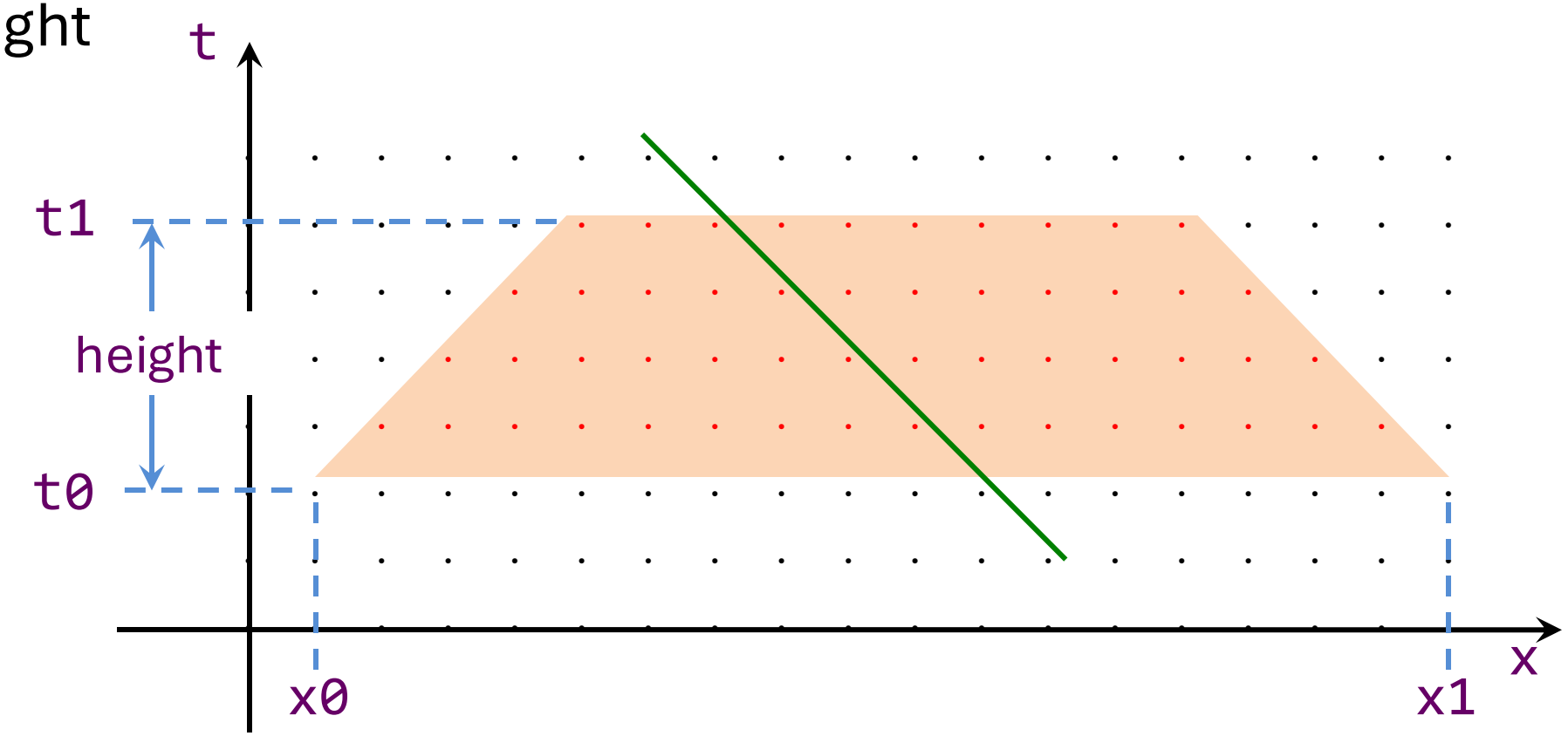
Time Cuts Don’t Parallelize

- There’s no way to parallelize a time cut
- The bottom trapezoid must be traversed first, and then the top one

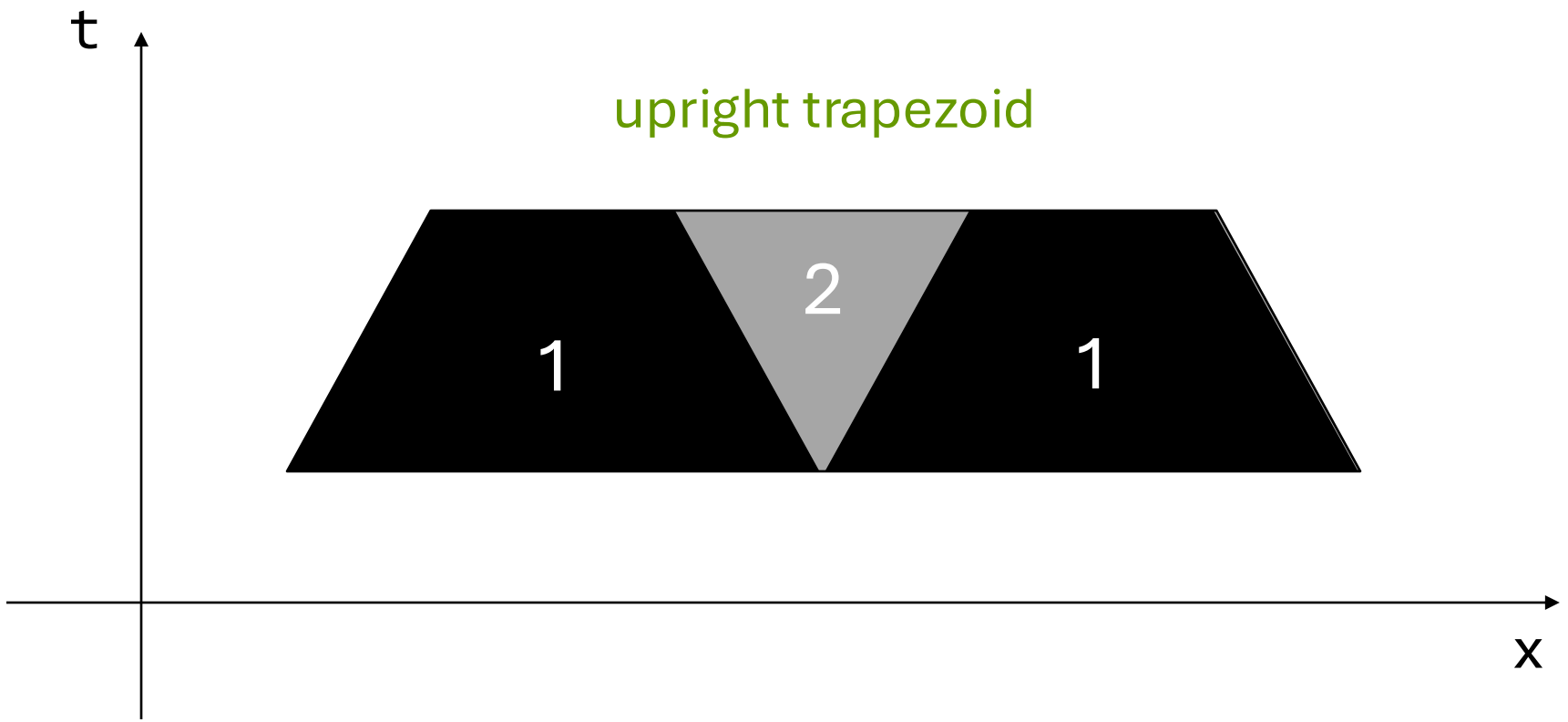


Space Cuts Don't Parallelize, or Do They?

- A space cut poses a similar problem
- Must traverse the trapezoid on the left before the one on the right

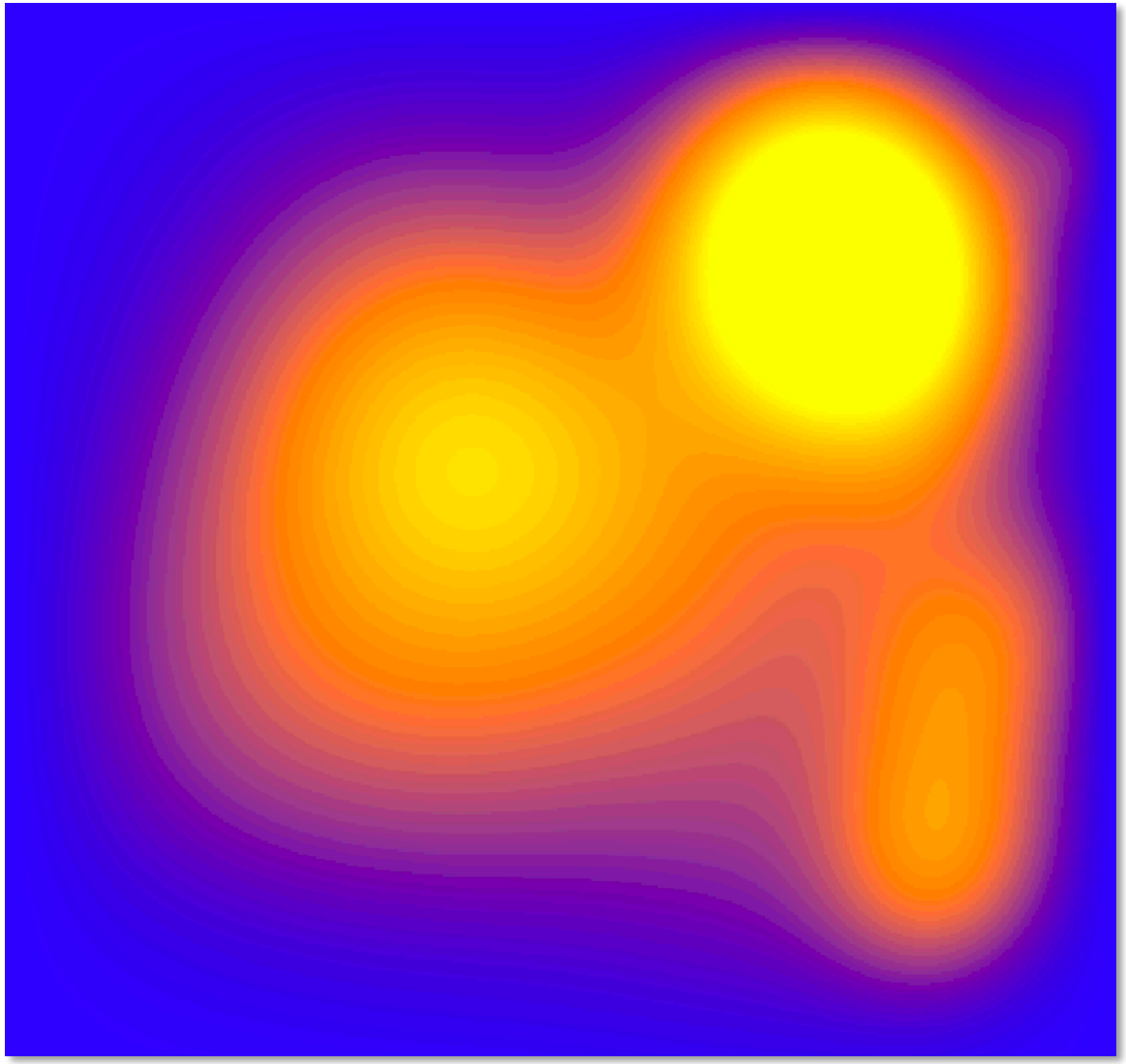


Parallel Space Cuts

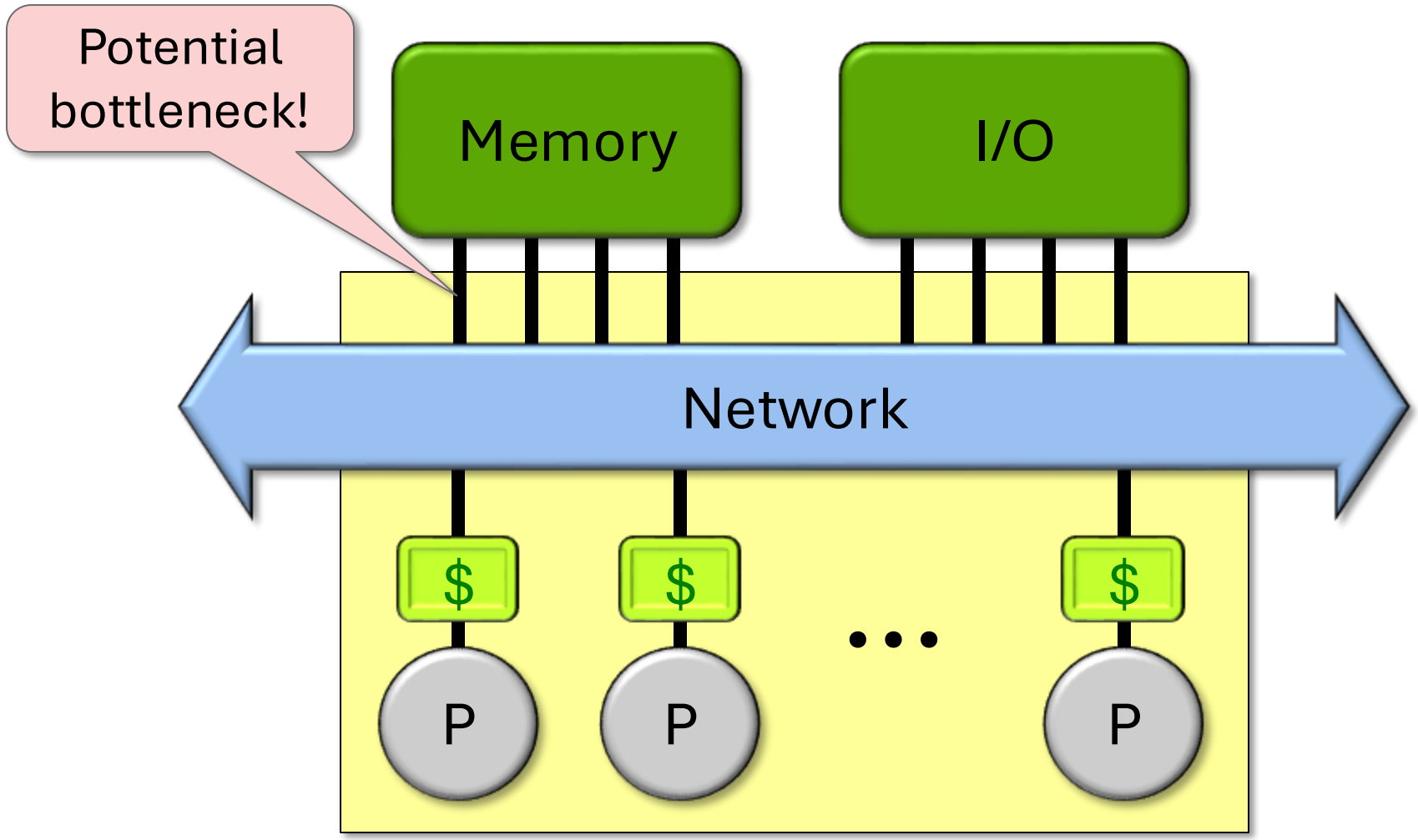


A parallel space cut produces two upright trapezoids (black) that can be executed in parallel and a third "inverted" trapezoid (gray) that must execute in series after the two upright trapezoids.

Parallel Looping vs. Parallel D&C



Memory Bandwidth



Impediments to Speedup

- ☑ Insufficient parallelism
- ☑ Scheduling overhead
- ☑ Lack of memory bandwidth
- ☐ Contention (locking and true/false sharing)

Cilkscale can diagnose the first two problems.

Q. How can we diagnose lack of memory bandwidth?

Impediments to Speedup

- ☑ Insufficient parallelism
- ☑ Scheduling overhead
- ☑ Lack of memory bandwidth
- ☐ Contention (locking and true/false sharing)

Cilkscale can diagnose the first two problems.

Q. How can we diagnose lack of memory bandwidth?

A. Run P identical copies of the serial projection in parallel — if you have enough memory.

- Tools exist to detect lock contention, but not the potential for lock contention
- Potential for true and false sharing is even harder to detect
  - although you shouldn't have true sharing if your code is free of determinacy races

Other C-O Algorithms

Matrix Transposition/Addition $\Theta(1+mn/\mathcal{B})$

Straightforward recursive algorithm.

Strassen's Algorithm $\Theta(n + n^2/\mathcal{B} + n^{\lg 7}/\mathcal{B}\mathcal{M}^{(\lg 7)/2 - 1})$

Straightforward recursive algorithm.

Fast Fourier Transform $\Theta(1 + (n/\mathcal{B})(1 + \log_{\mathcal{M}} n))$

Variant of Cooley-Tukey [CT65] using cache-oblivious matrix transpose.

LUP-Decomposition $\Theta(1 + n^2/\mathcal{B} + n^3/\mathcal{B}\mathcal{M}^{1/2})$

Recursive algorithm due to Sivan Toledo [T97].

C-O Data Structures

Ordered-File Maintenance

$O(1 + (\lg^2 n)/\mathcal{B})$

INSERT/DELETE anywhere in file while maintaining $O(1)$-sized gaps. Amortized bound [BDFC00], later improved in [BCDFC02].

B-Trees

INSERT/DELETE:	$O(1+\log_{\mathcal{B}+1} n+(\lg^2 n)/\mathcal{B}$
SEARCH:	$O(1+\log_{\mathcal{B}+1} n)$
TRAVERSE:	$O(1+k/\mathcal{B})$

Solution [BDFC00] with later simplifications [BDIW02], [BFJ02].

Priority Queues

$O(1+(1/\mathcal{B})\log_{\mathcal{M}/\mathcal{B}}(n/\mathcal{B}))$

Funnel-based solution [BF02]. General scheme based on buffer trees [ABDHMM02] supports INSERT/DELETE.

Take-Aways

- Cache-oblivious algorithms can use cache as well as cache-aware algorithms without resorting to voodoo parameters
- Cache-efficiency matters more for parallel code
- The base case of divide-and-conquer algorithms must be engineered to account for hardware acceleration (prefetching, etc.)
  - along the fast-moving index of a multidimensional array