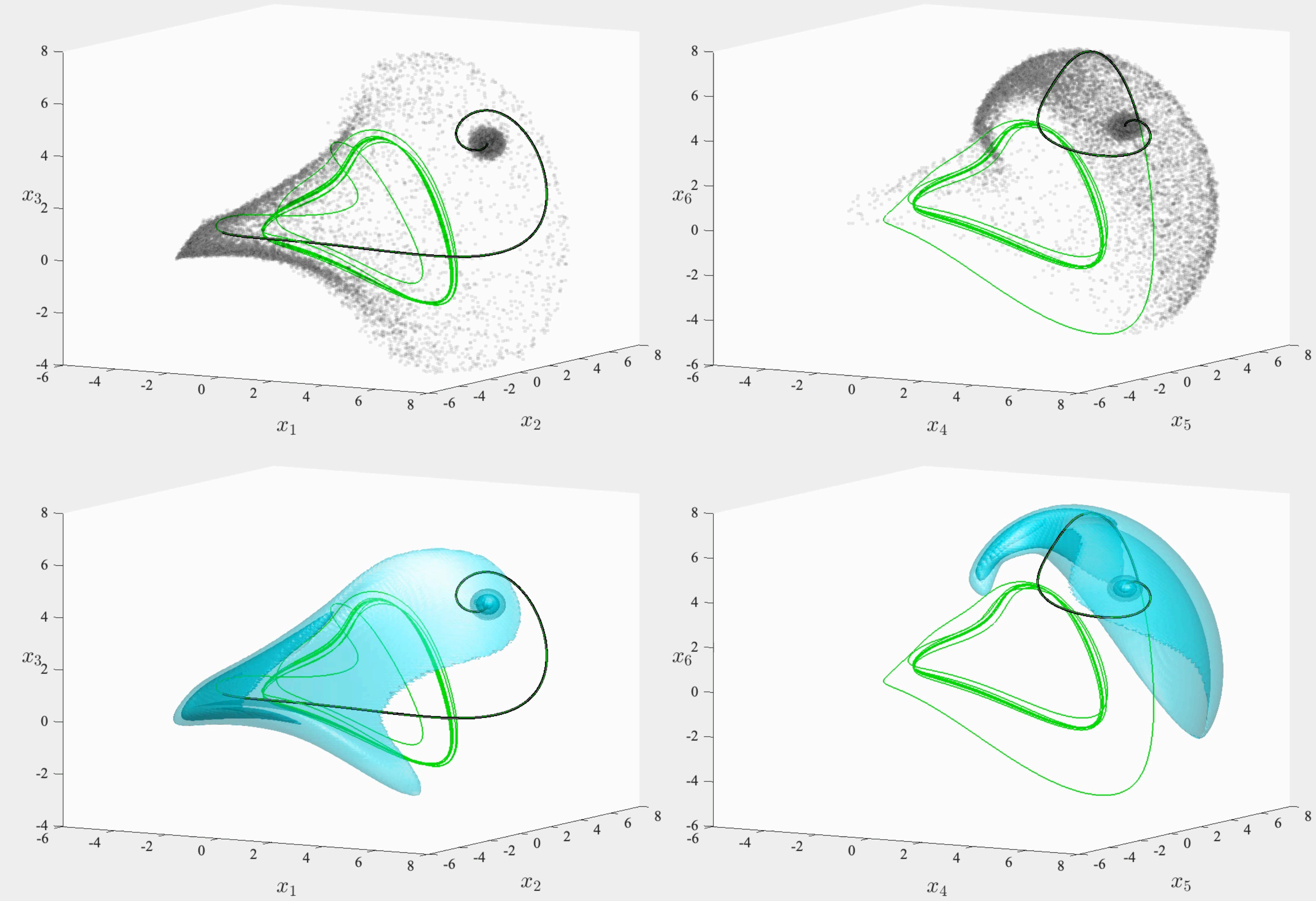


GBEES-GPU: An efficient parallel GPU algorithm for high-dimensional nonlinear uncertainty propagation

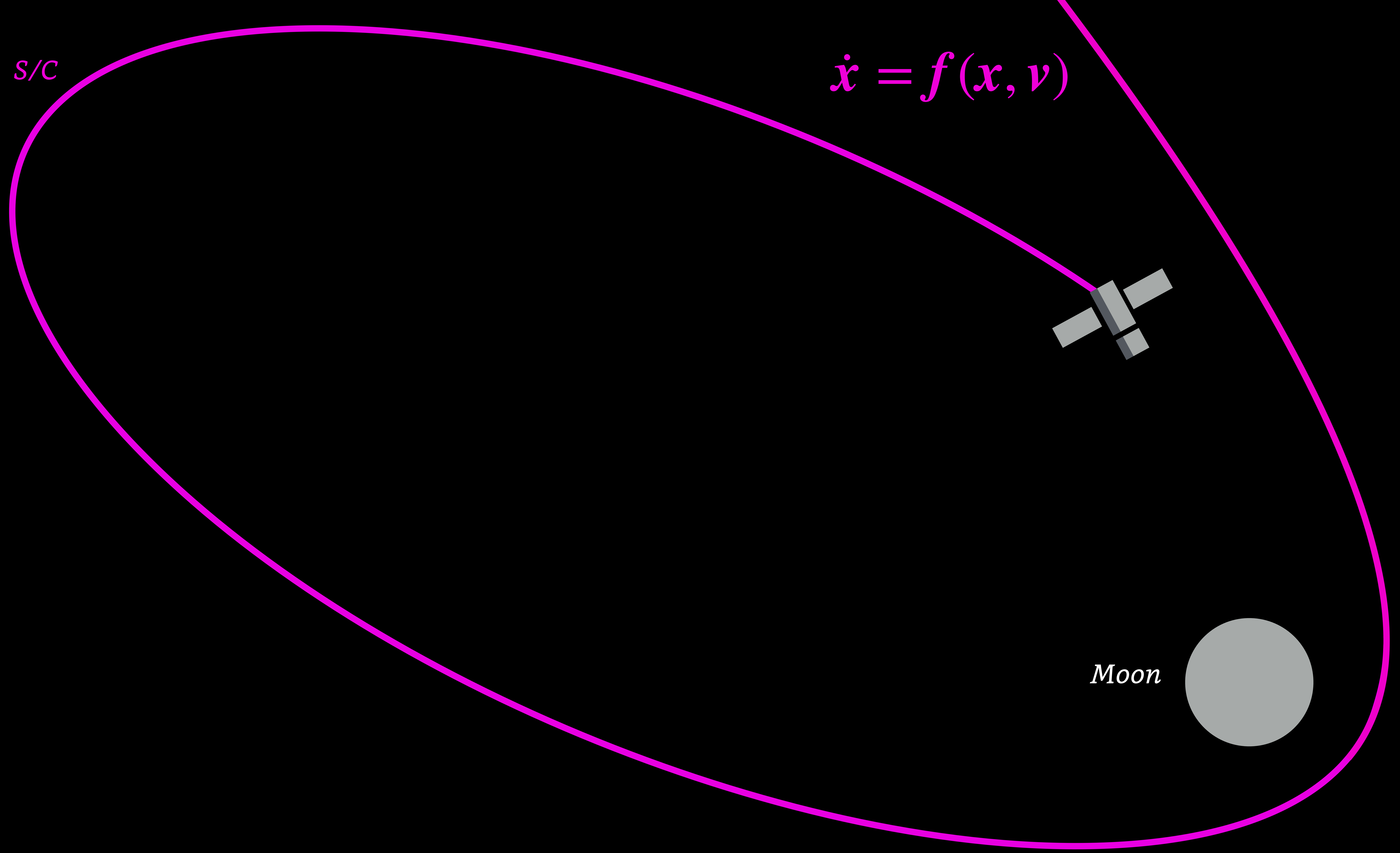
Computer Physics Communications,
10.1016/j.cpc.2025.109819

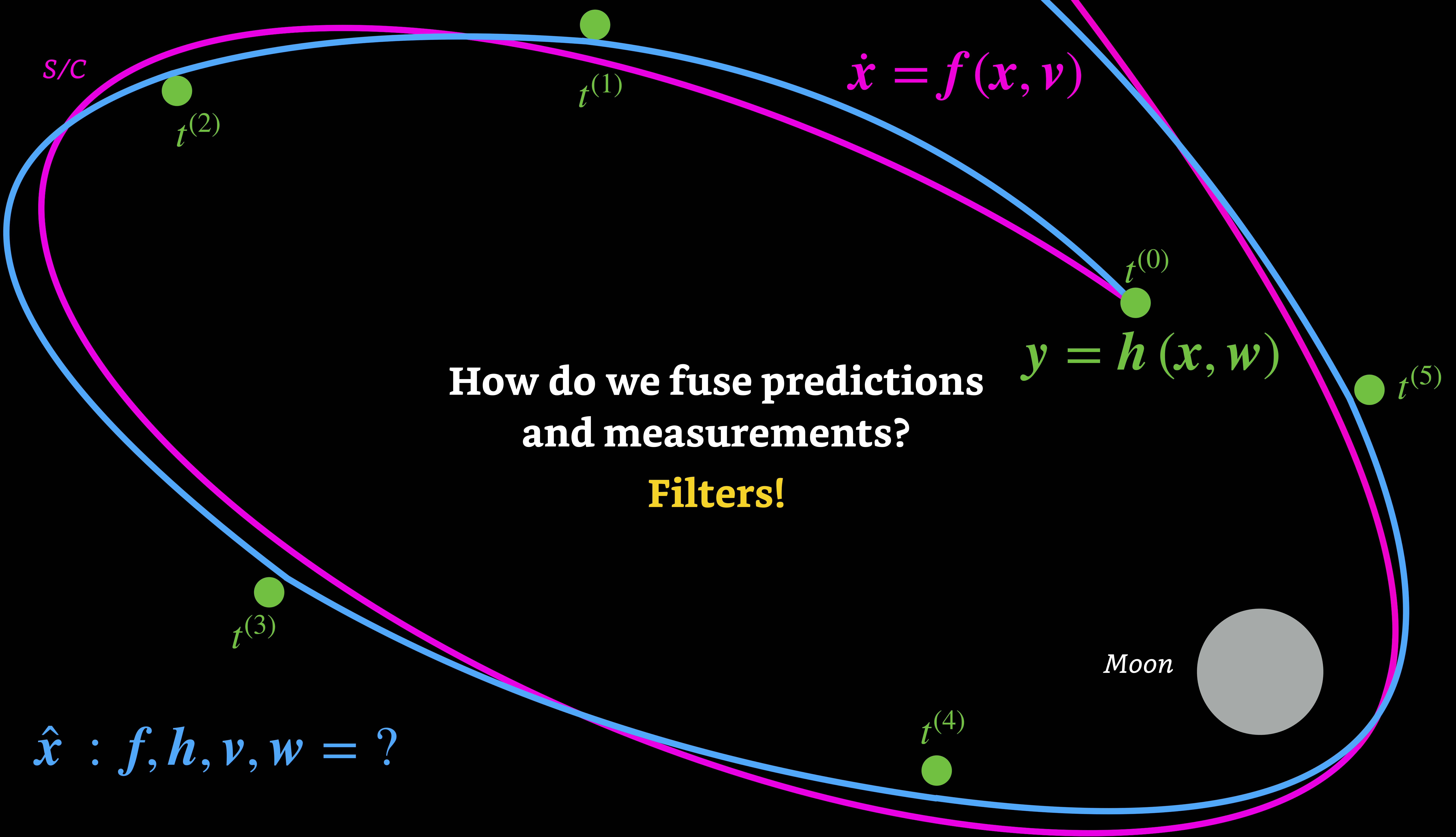
Benjamin L. Hanson
Ph.D. Student, Jacobs School of Engineering
Department of Mechanical and Aerospace Engineering
UC San Diego, La Jolla, CA

Dr. Carlos Rubio
Adjunct Professor, School of Engineering
Department of Mechanical, Computer and Aerospace Engineering
Universidad de León, León, Spain

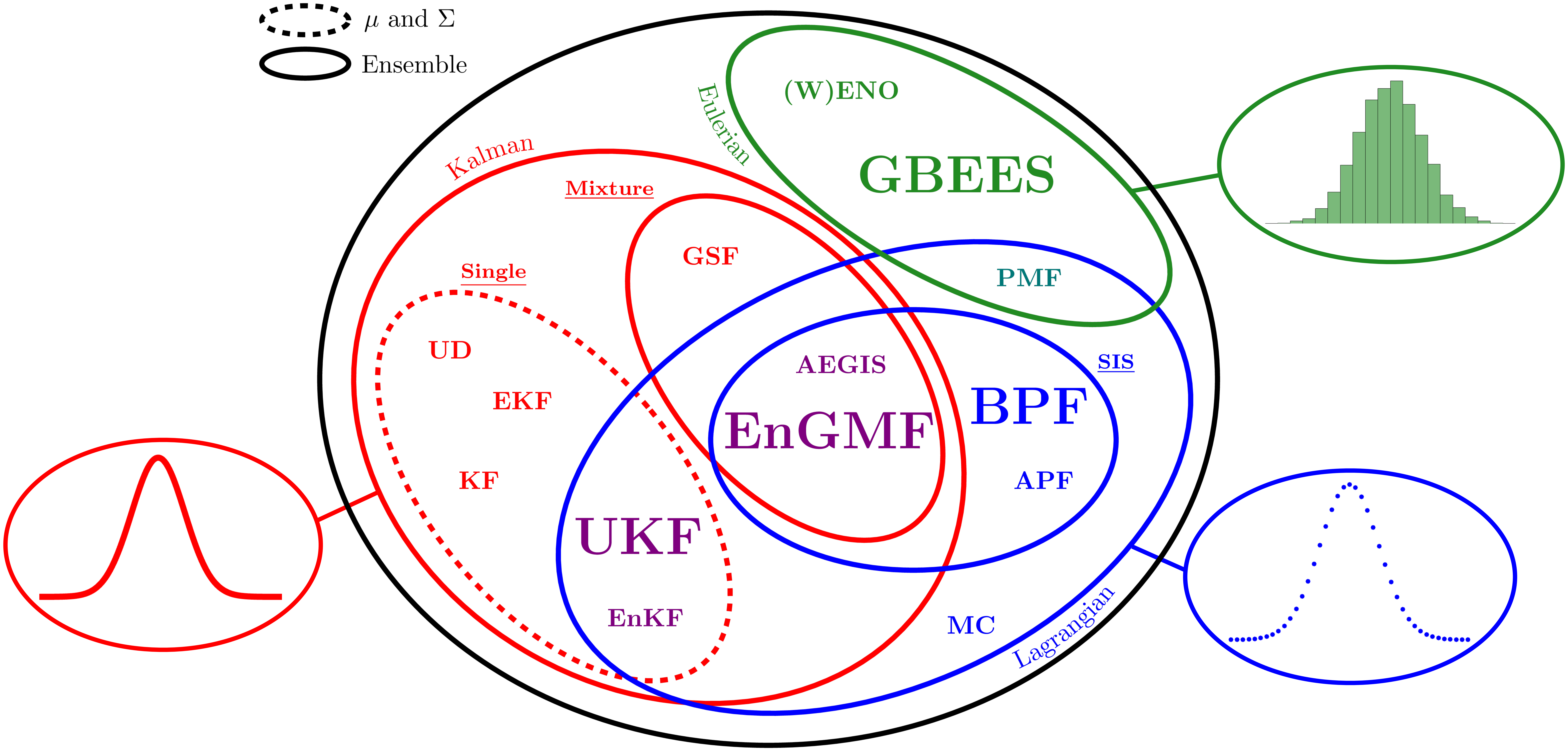
Dr. Adrián García-Gutiérrez
Associate Professor, School of Engineering
Department of Mechanical, Computer and Aerospace Engineering
Universidad de León, León, Spain

Dr. Thomas R. Bewley
Professor, Jacobs School of Engineering
Department of Mechanical and Aerospace Engineering
UC San Diego, La Jolla, CA


S/C
$\dot{x} = f(x, v)$
Moon


S/C
$t^{(1)}$
$t^{(2)}$
$\dot{x} = f(x, v)$
$t^{(0)}$
$y = h(x, w)$
How do we fuse predictions
and measurements?
Filters!
$t^{(5)}$
$t^{(3)}$
Moon
$t^{(4)}$
$\hat{x} \; : \; f, h, v, w = \, ?$

Current Landscape of Recursive Bayesian Filters



Current Landscape of Recursive Bayesian Filters

Kalman Approach

Pros
- Optimal when systems are linear
- Closed-form update equations (deterministic)
- Highly efficient and tractable

Cons
- Poor accuracy in the case of non-Gaussian posteriors
- Possibility of divergence when dynamics or measurement model are nonlinear

Lagrangian Approach

Pros
- Uses exact model definitions
- Easy to implement
- Capable of handling non-Gaussian posteriors

Cons
- Particle degeneracy without resampling
- High sample requirements in high dimensions
- Computationally expensive

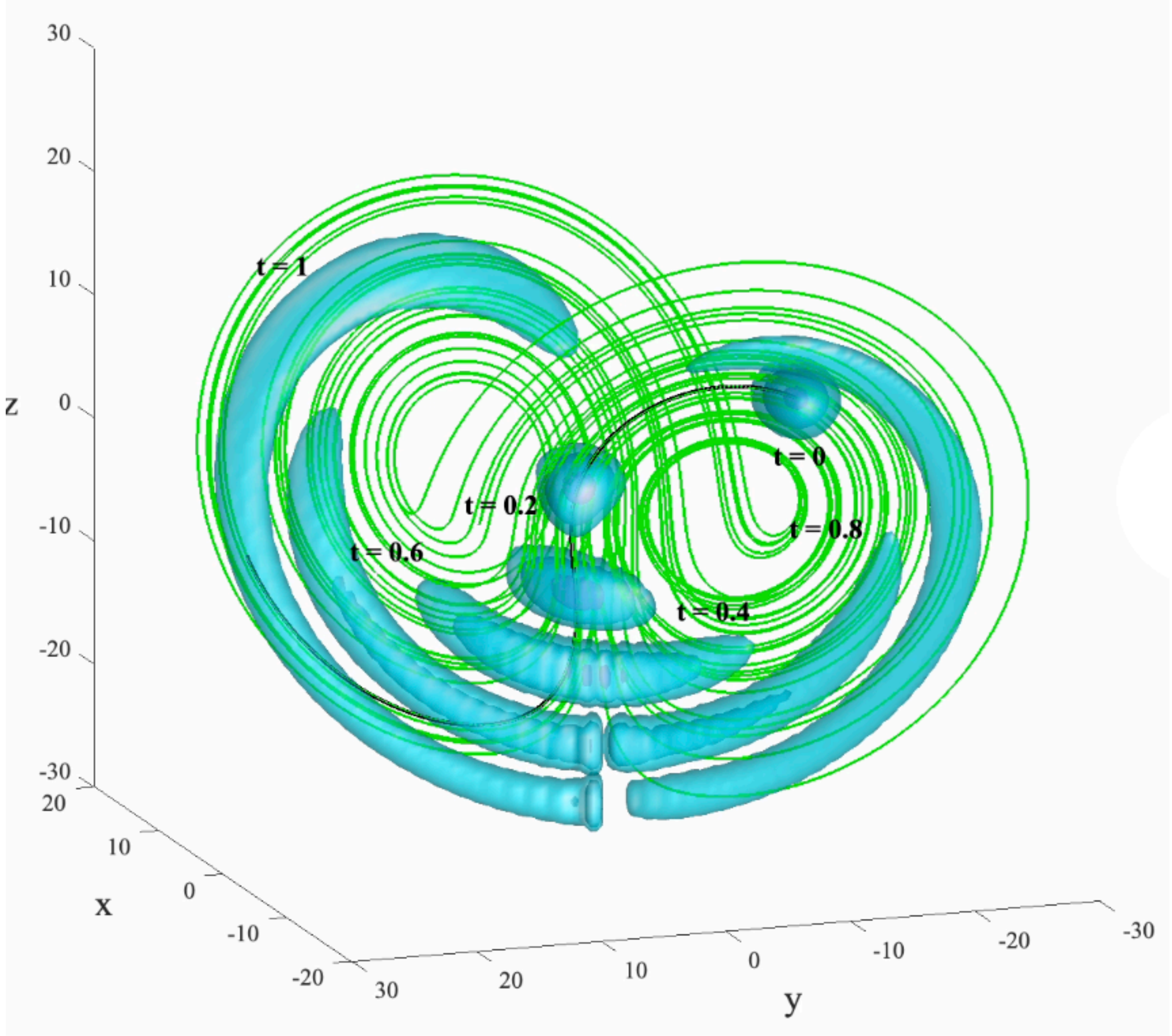
Eulerian Approach

Pros
- Uses exact model definitions
- Capable of handling non-Gaussian posteriors
- Avoids particle degeneracy by maintaining resolution

Cons
- Finite domain limitation for standard methods
- Computationally expensive

Grid-based Bayesian Estimation Exploiting Sparsity (GBEES)

- GBEES is a 2nd-order accurate, Godunov finite volume method that treats probability as a fluid, flowing the PDF through phase space subject to the dynamics of the system



Initially Gaussian uncertainty becoming highly non-Gaussian when subjected to the Lorenz '63 model

Where most Eulerian methods suffer and why GBEES doesn't

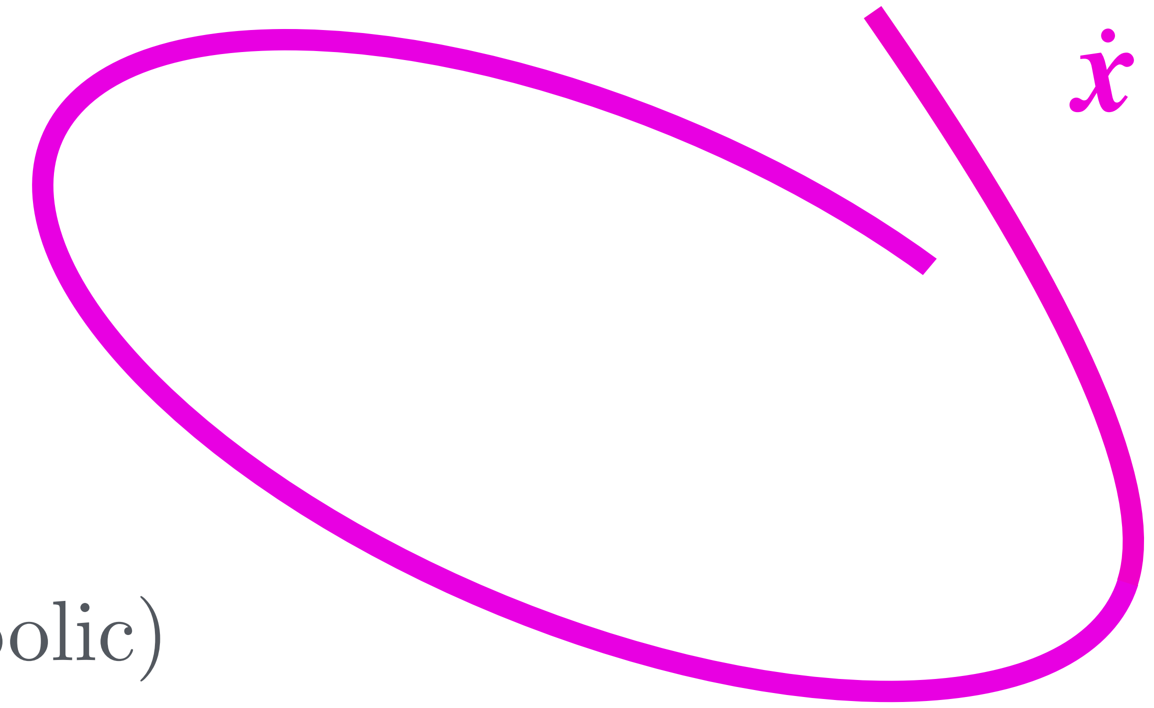
1. The finite domain limitation is circumvented by dynamically allocating grid cells in regions of non-negligible probability
2. The computational bottleneck of marching a full, discretized, high-dimensional PDF is overcome by exploiting the sparsity of that PDF in most of phase space

T.R. Bewley et al. (2012) Efficient grid-based Bayesian estimation of nonlinear low-dimensional systems with sparse non-Gaussian PDFs. Automatica, 48 (10.1016/j.automatica.2012.02.039)

Grid-based Bayesian Estimation Exploiting Sparsity (GBEES)

- GBEES consists of two distinct processes, one performed in **continuous-time**, the other in **discrete-time**:
  1. **Prediction:** $p(\mathbf{x}, t)$ is continuous-time marched via the **Fokker-Planck Equation**:

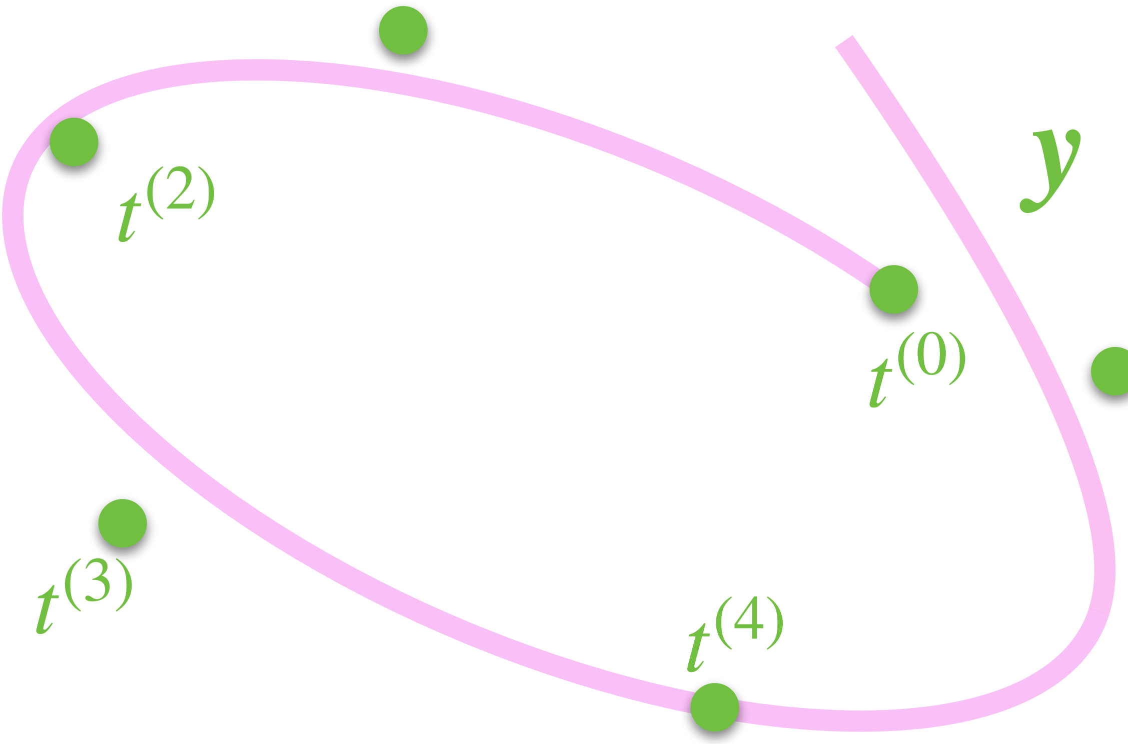
$$\frac{\partial p(\boldsymbol{x}, t)}{\partial t} = -\sum_{j=1}^{n} \frac{\partial f_j(\boldsymbol{x}, t) p(\boldsymbol{x}, t)}{\partial x_j} + \frac{1}{2} \sum_{j=1}^{n} \sum_{\ell=1}^{n} \frac{\partial^2 Q_{j\ell}(\boldsymbol{x}, t) p(\boldsymbol{x}, t)}{\partial x_j \partial x_\ell}$$



     - f_i: advection (EOMs) in the i^{th} dimension
     - q_{ij}: $(i, j)^{\text{th}}$ element of the spectral density ($Q(\boldsymbol{x}, t) \approx 0$, PDE is hyperbolic)

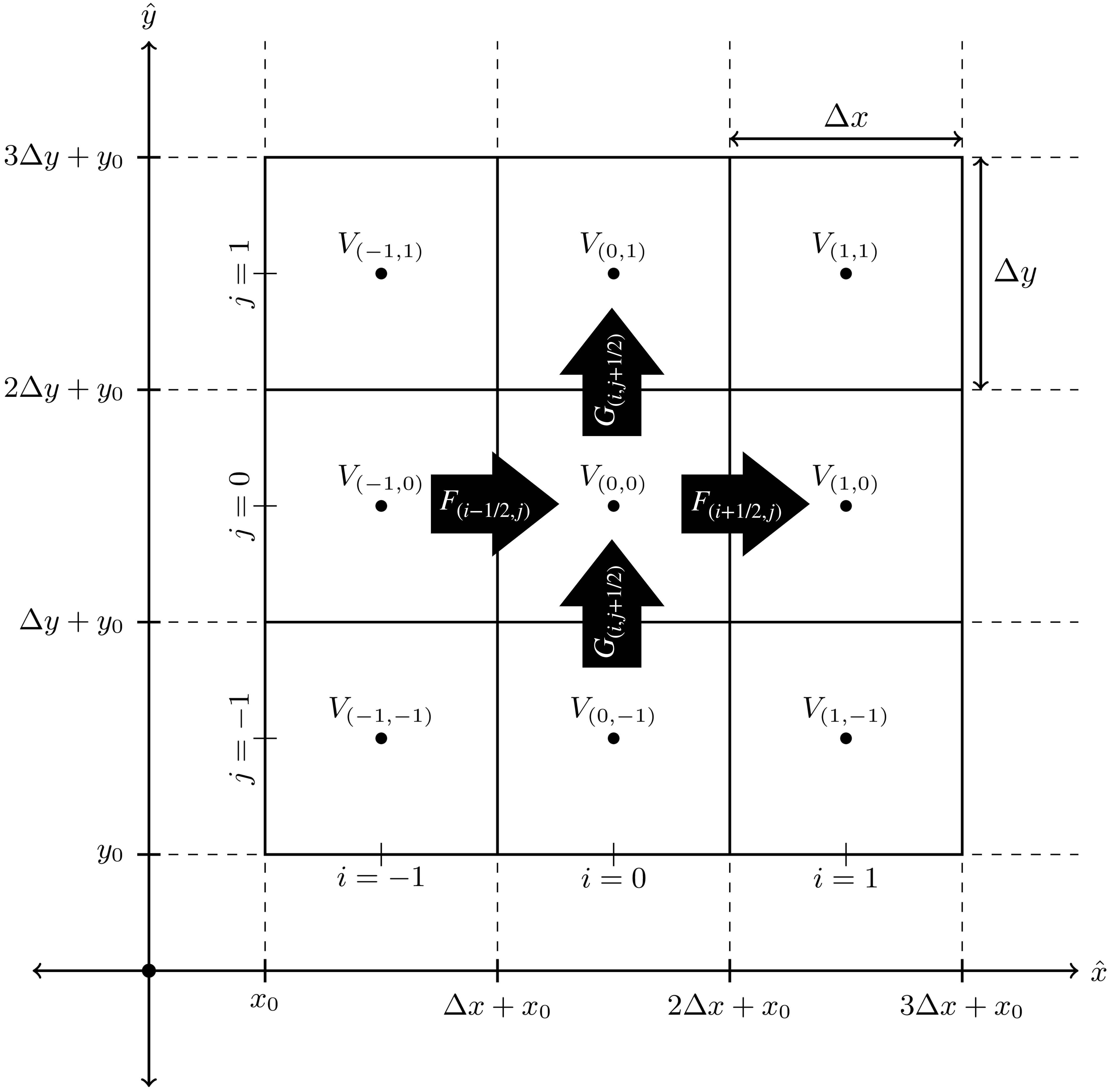
  2. **Correction:** at discrete-time interval $t^{(k)}$, measurement $\boldsymbol{y}^{(k)}$ updates $p(\boldsymbol{x}, t)$ via **Bayes' Theorem**:

$$p\left(\boldsymbol{x}, t^{(k+)}\right) = \frac{p\left(\boldsymbol{y}^{(k)} | \boldsymbol{x}\right) p\left(\boldsymbol{x}, t^{(k-)}\right)}{C}$$



     - $p\left(\mathbf{x}, t^{(k+)}\right)$: a posteriori distribution
     - $p\left(\mathbf{y}^{(k)} | \mathbf{x}\right)$: measurement distribution
     - $p\left(\mathbf{x}, t^{(k-)}\right)$: a priori distribution
     - C: normalization constant

Grid-based Bayesian Estimation Exploiting Sparsity (GBEES)



Godunov-type finite volume method implemented on a uniform Cartesian 2D mesh

- **Prediction:** assuming process noise is relatively small ($Q(\boldsymbol{x}, t) \approx 0$), the 2nd-order discrete approximation of

$$\frac{\partial p(\boldsymbol{x}, t)}{\partial t} = -\sum_{j=1}^{2} \frac{\partial f_j(\boldsymbol{x}, t) p(\boldsymbol{x}, t)}{\partial x_j} + \frac{1}{2} \sum_{j=1}^{2} \sum_{\ell=1}^{2} \frac{\partial^2 Q_{j\ell}(\boldsymbol{x}, t) p(\boldsymbol{x}, t)}{\partial x_j \partial x_\ell}$$

is

$$\frac{p^{(n+1)}_{(i,j)} - p^{(n)}_{(i,j)}}{\Delta t} = -\frac{F^{(n)}_{(i+1/2,j)} - F^{(n)}_{(i-1/2,j)}}{\Delta x} - \frac{G^{(n)}_{(i,j+1/2)} - G^{(n)}_{(i,j-1/2)}}{\Delta y},$$

where $t = t^{(n)}$ and

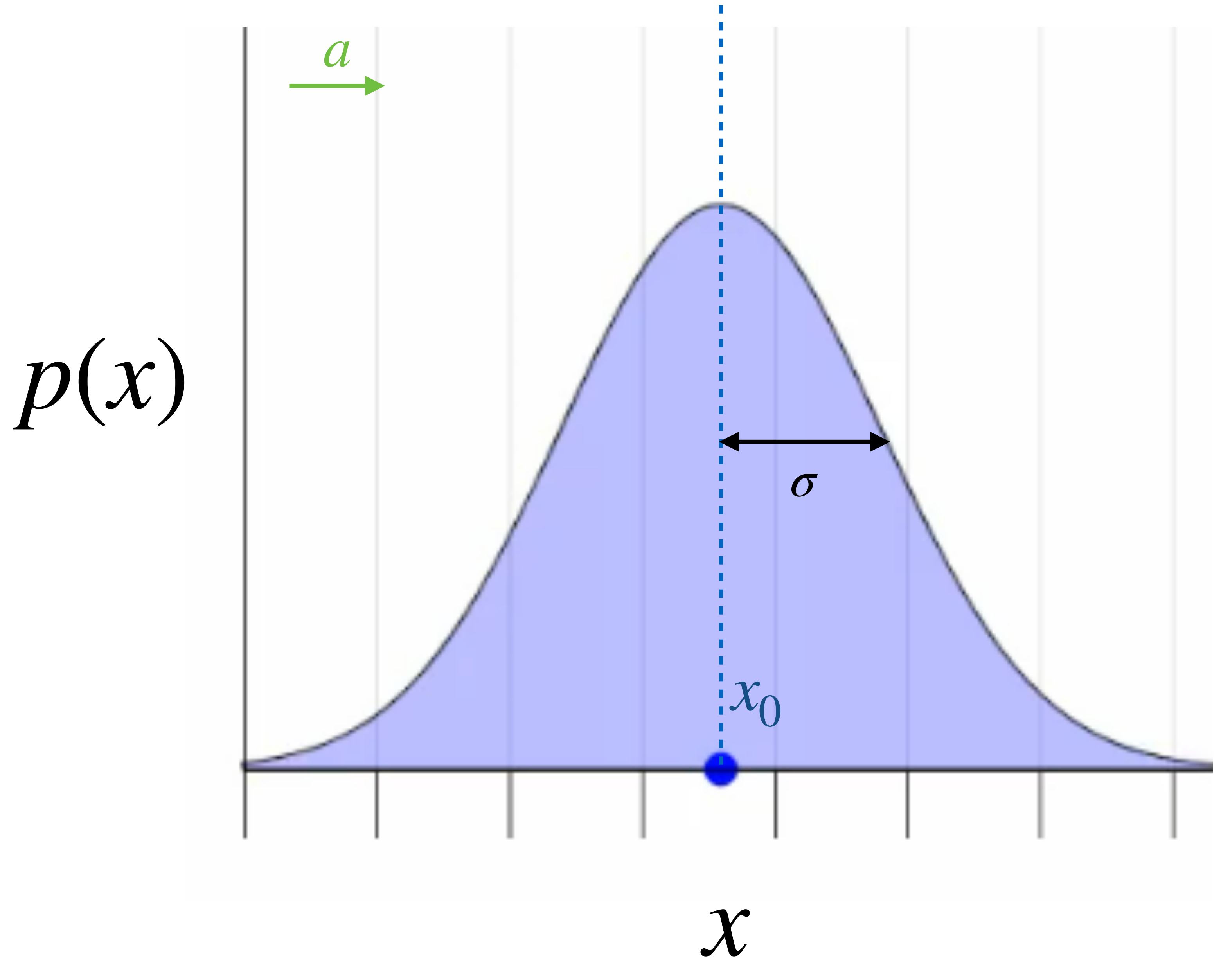
  - $p^{(n)}_{(i,j)}$ = probability at cell $V_{(i,j)}$
  - Δt = size of time step
  - $F^{(n)}_{(i-1/2,j)}$ = x-direction half-step backward flux
  - $F^{(n)}_{(i+1/2,j)}$ = x-direction half-step forward flux
  - $G^{(n)}_{(i,j-1/2)}$ = y-direction half-step backward flux
  - $G^{(n)}_{(i,j+1/2)}$ = y-direction half-step forward flux

- **Correction:** because we have the PDF defined over a grid, we can directly carry out a discretized implementation of Bayes' Theorem

- Consider a 1-dimensional, linear test example:

$$\boldsymbol{x} = x, \quad \frac{d\boldsymbol{x}}{dt} = a, \quad a > 0$$

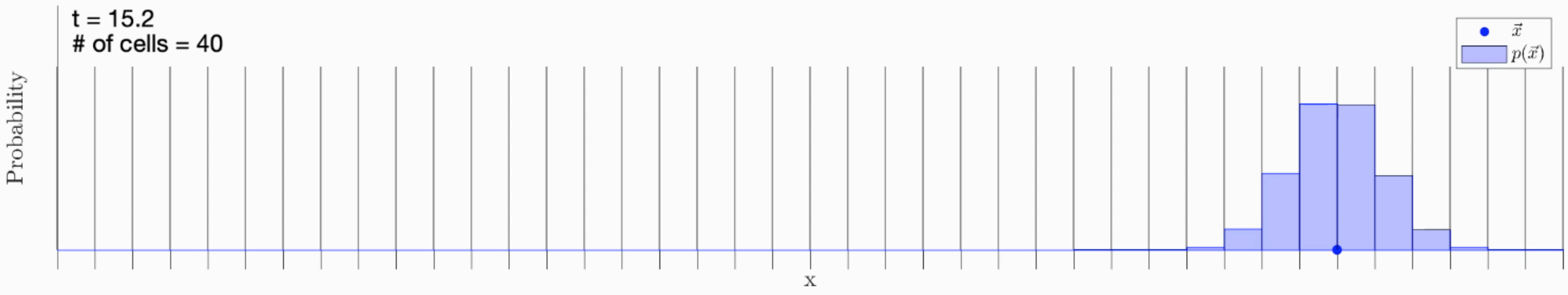
- Initial observation of $x(t)$ results in a Gaussian PDF $p(x)$ centered about x_0 with standard deviation σ



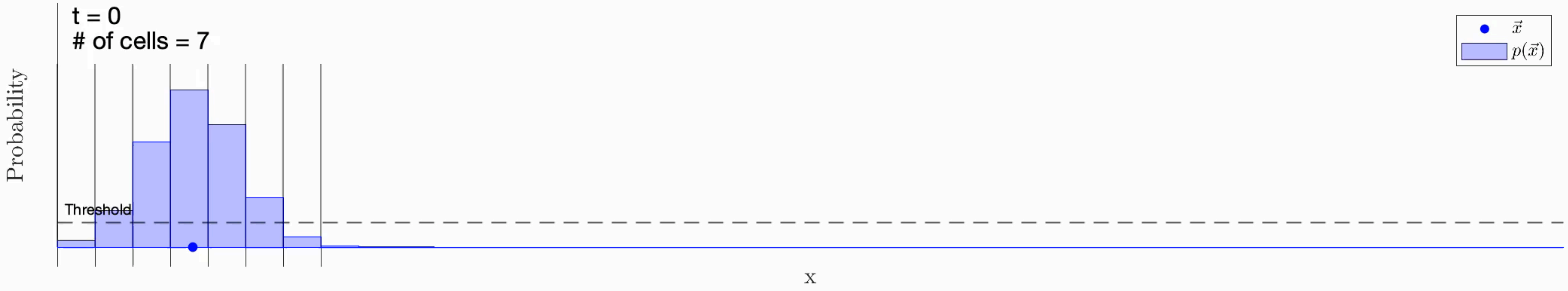
How does $p(x)$, governed by $d\boldsymbol{x}/dt$, change with respect to t?

Grid-based Bayesian Estimation **Exploiting Sparsity** (GBEES)

Ignoring sparsity



Exploiting sparsity



Not GBEES, just a visual aid

GBEES CPU-legacy Implementation

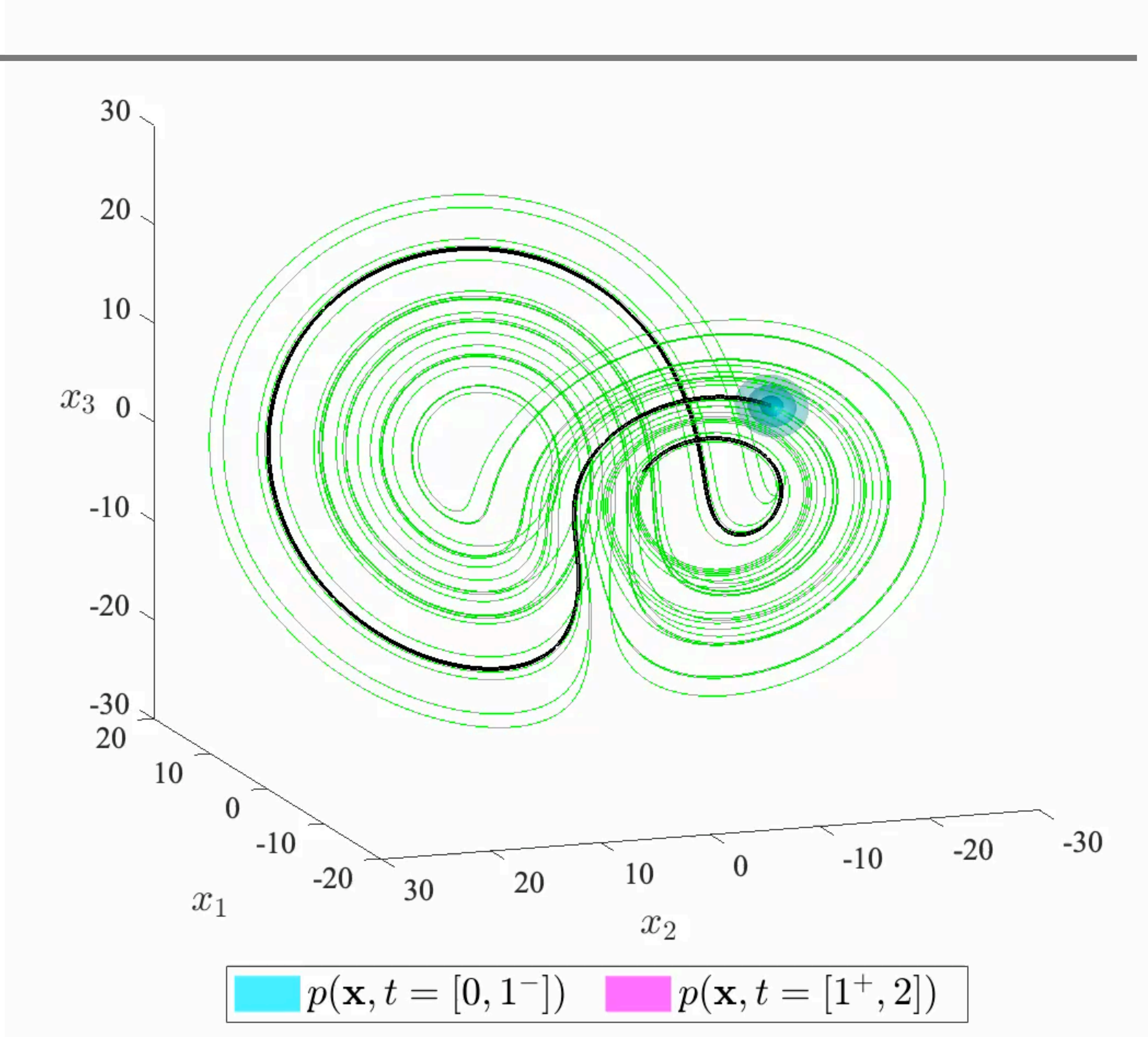
Application: Lorenz '63 Model

- State and equations of motion of the three-dimensional system:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \frac{d\boldsymbol{x}}{dt} = \begin{bmatrix} \sigma(x_2 - x_1) \\ -x_2 - x_1 x_3 \\ -b(x_3 + r) - x_1 x_2 \end{bmatrix},$$

where $\{\sigma, b, r\} = \{4,1,48\}$ results in the chaotic behavior seen in the right figure

- GBEES CPU-legacy runtime for propagating uncertainty from $t = [0,2]$ with $x_3 = -10$ measurement update at $t = 1$: **28.8 s**



Initial uncertainty of $\sigma_{x_j} = 1$ and grid width of $\Delta x_j = 0.5$ for $j = 1$, 2, and 3

Areas of improvement

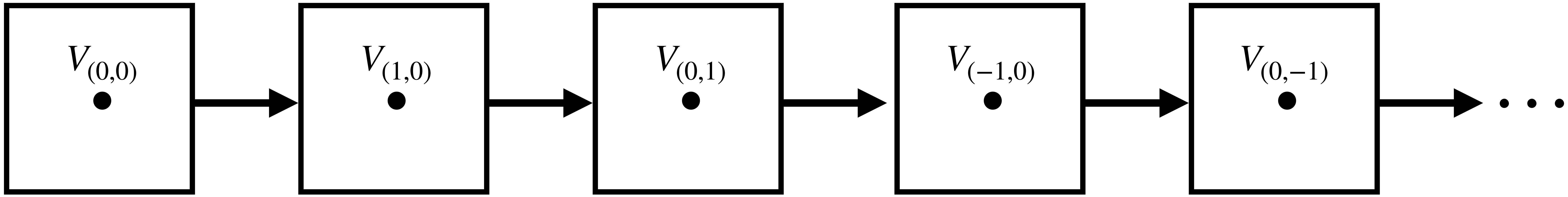
1. Grid data structure has an $\mathcal{O}\left(N^2\right)$ time complexity, where N is grid size
2. Over-conservative, fixed time step is required to maintain algorithm stability
3. No consideration for direction of upwind/downwind when creating/deleting cells
4. Parallelization by translating algorithm to CUDA and executing on GPU

GBEES CPU-optimized: Data structures

- The data structures where the n-dimensional grids are stored determine time complexity

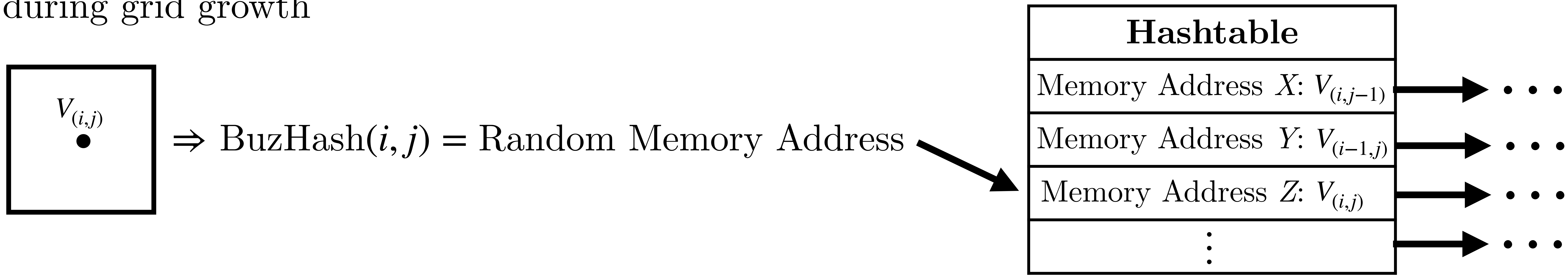
Legacy implementation

- GBEES CPU-legacy uses a linked list which results in an $\mathcal{O}\left(N^2\right)$ time complexity during grid growth



Optimized implementation

- GBEES CPU-optimized uses a hash table which results in an $\mathcal{O}(N)$ time complexity during grid growth



J.D. Cohen (1997) Recursive hashing functions for n-grams. ACM Trans. Inf. Syst., 15 (10.1145/256163.256168)

GBEES CPU-optimized: Adaptive time step

- In order to maintain stability, explicit finite volume methods must satisfy the Courant-Friedrichs-Lewy (CFL) condition:

$$C = \Delta t \left(\frac{F}{\Delta x} + \frac{G}{\Delta y} \right) \leq C_{\max},$$

where $C_{\max}$ is often chosen to be 1 for hyperbolic PDEs

Legacy implementation

- Uses an over-restrictive Δt so the CFL condition is always satisfied

R. Courant et al. (1967) On the partial difference equations of mathematical physics, IBM J. Res. Dev. 11 (10.1147/rd.112.0215)

GBEES CPU-optimized: Adaptive time step

- In order to maintain stability, explicit finite volume methods must satisfy the Courant-Friedrichs-Lewy (CFL) condition:

$$C = \Delta t \left(\frac{F}{\Delta x} + \frac{G}{\Delta y} \right) \leq C_{\max},$$

where $C_{\max}$ is often chosen to be 1 for hyperbolic PDEs

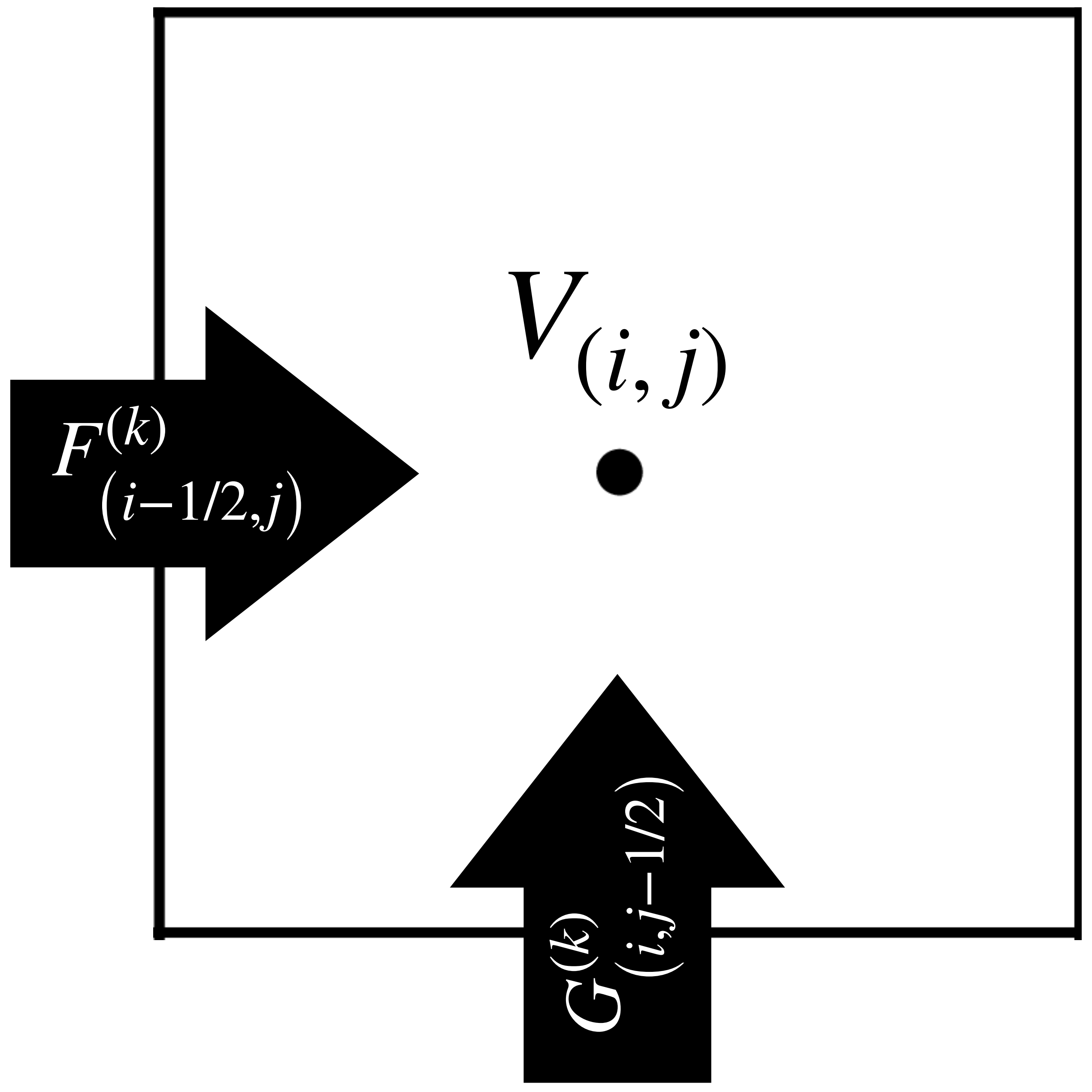
Legacy implementation

- Uses an over-restrictive Δt so the CFL condition is always satisfied

Optimized implementation

- Uses an adaptive, CFL-minimized time step for maximum efficiency

$$\Delta t^{(k)} = \min_{(i,j)\in \mathrm{grid}} \left[\left(\frac{F^{(k)}_{(i-1/2,j)}}{\Delta x} + \frac{G^{(k)}_{(i,j-1/2)}}{\Delta y} \right)^{-1} \right]$$

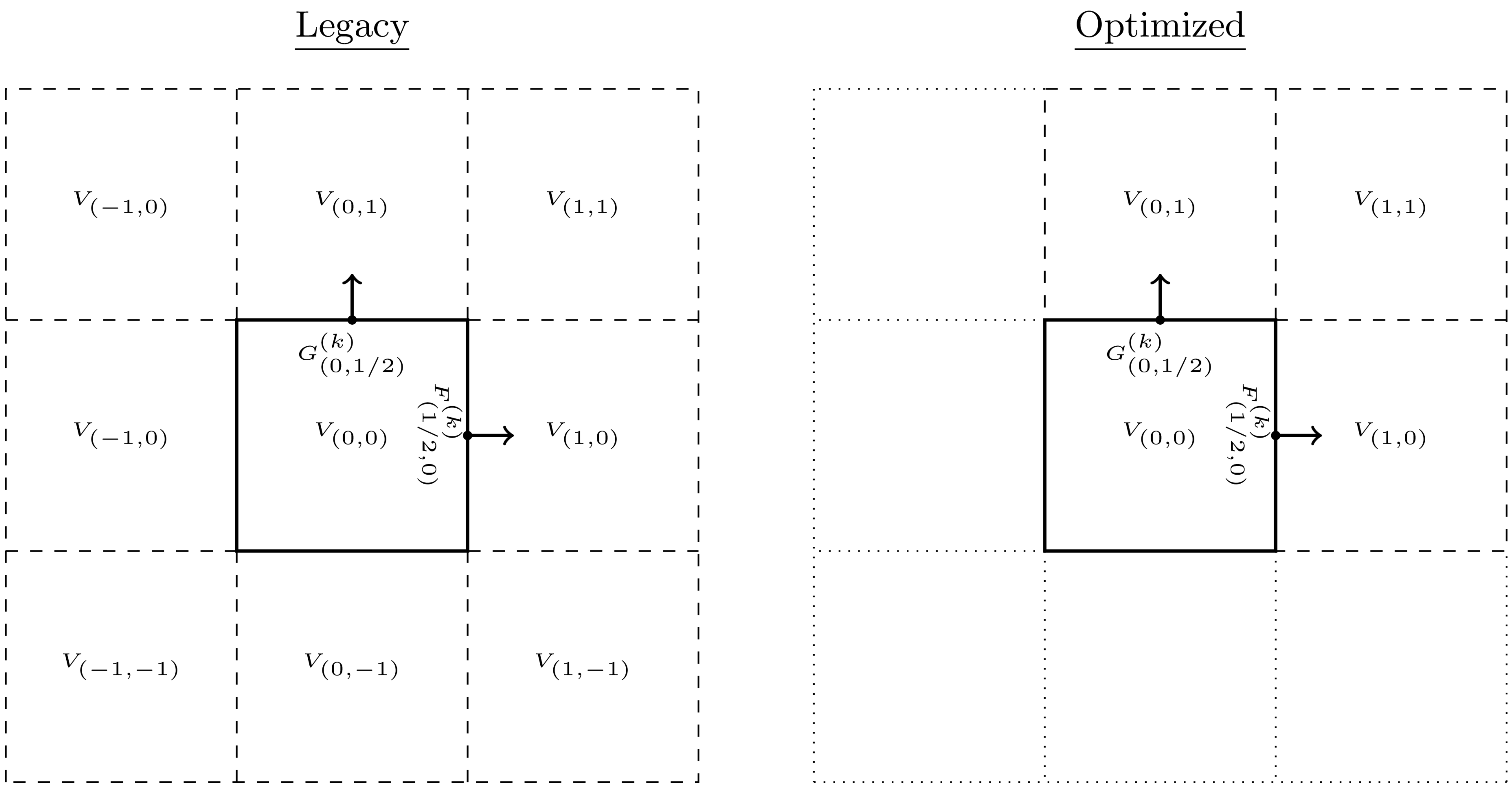


R. Courant et al. (1967) On the partial difference equations of mathematical physics, IBM J. Res. Dev. 11 (10.1147/rd.112.0215)

GBEES CPU-optimized: Directional growing/pruning

Directional Growing

- Legacy implementation has no consideration for fluxing direction when growing grid
- Optimized implementation only creates **downwind** grid cells when growing grid



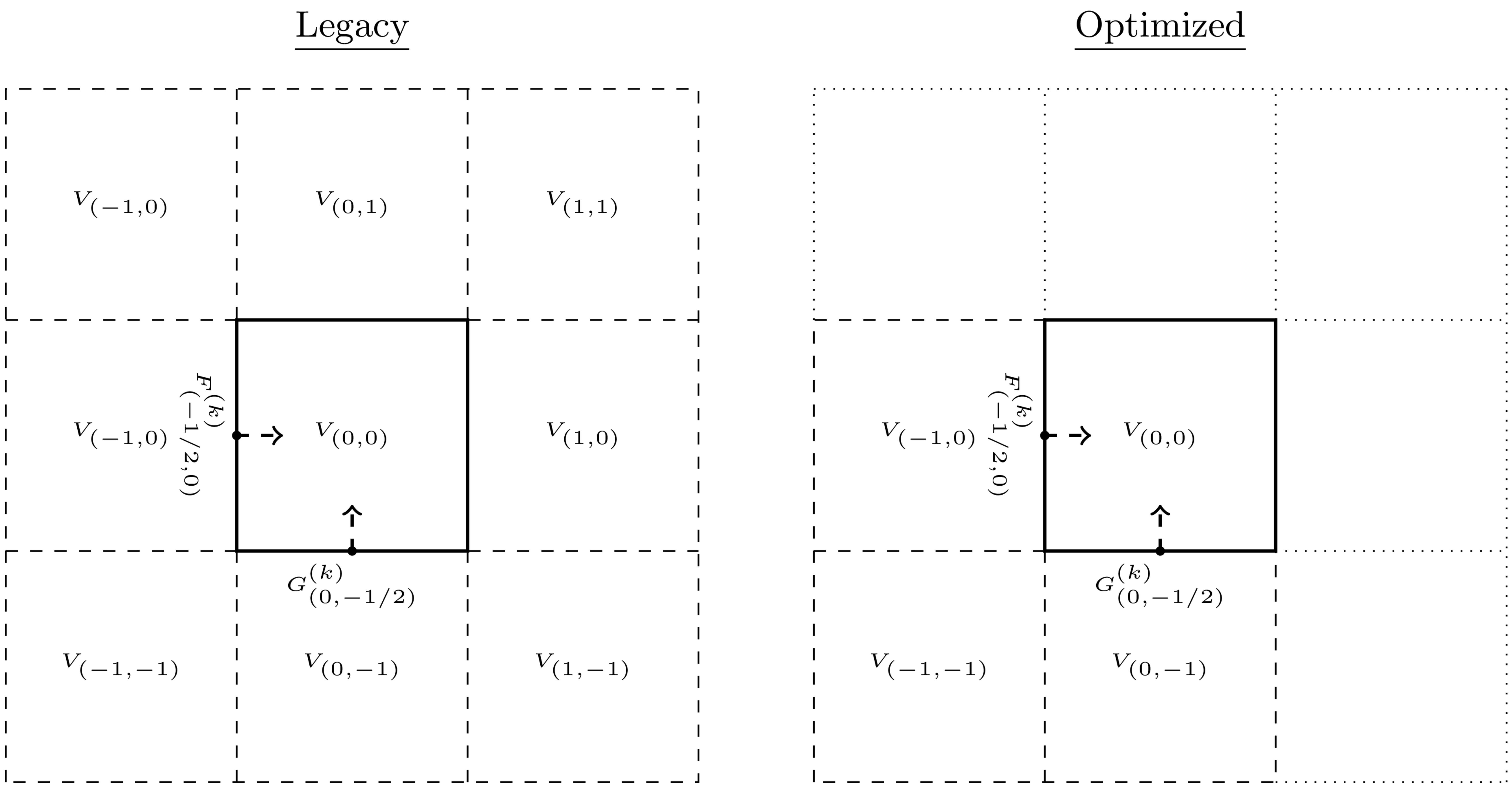
# of cells checked: $3^n - 1$

Max # of cells checked: $2^n - 1$

GBEES CPU-optimized: Directional growing/pruning

Directional Pruning

- Legacy implementation has no consideration for fluxing direction when pruning grid
- Optimized implementation only checks **upwind** grid cells when pruning grid



# of cells checked: $3^n - 1$

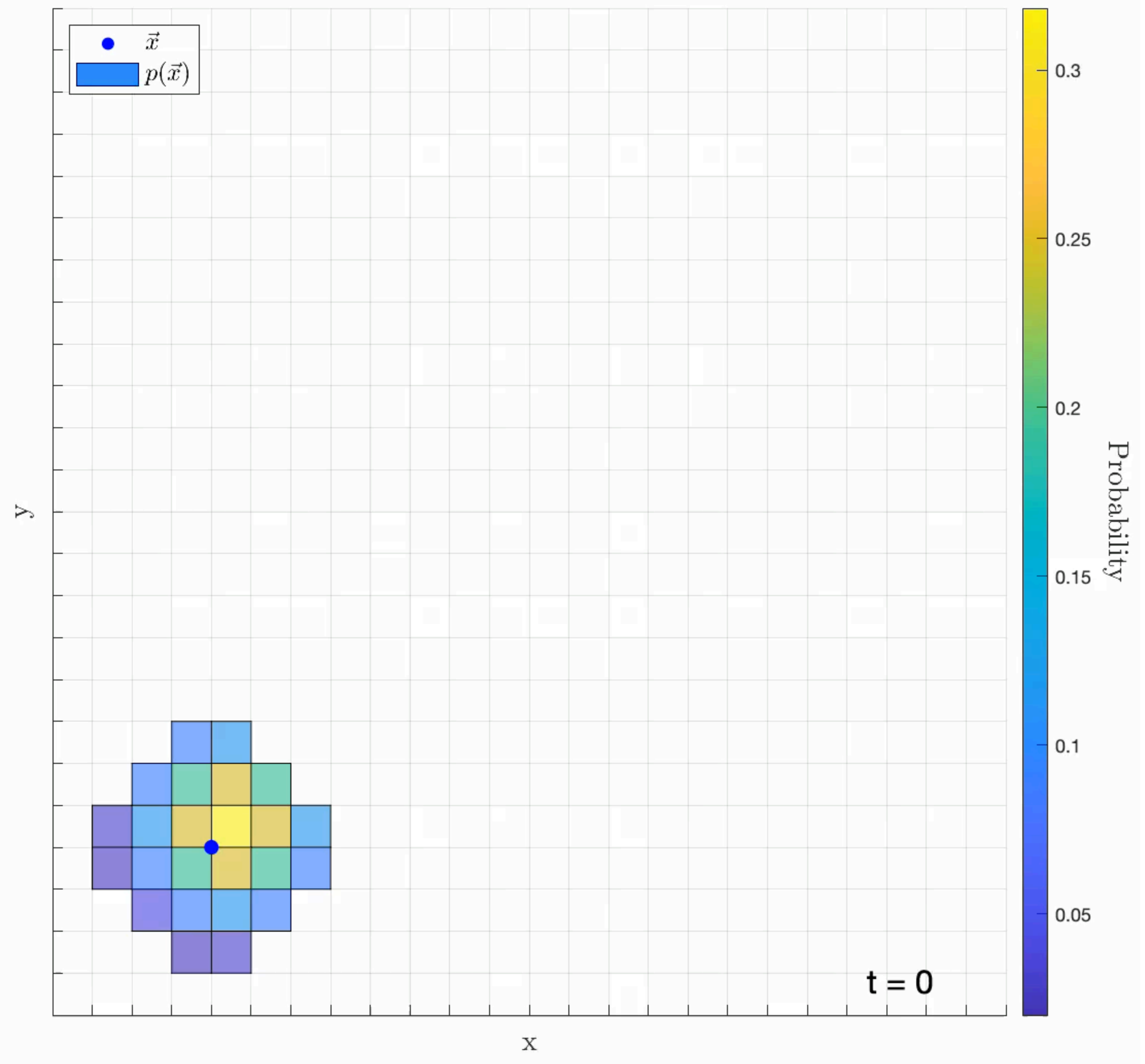
Max # of cells checked: $2^n - 1$

GBEES-GPU: Introduction

- Because GBEES exploits sparsity, parallelization of the dynamic grid is nontrivial

Traditional Approach to Grid Parallelization

- Subdomains are **statically assigned** to thread blocks
- Works for **low-dimensional problems** with predictable grid size
- **Problem:** number of cells grows exponentially with dimension, so static partitioning becomes infeasible



GBEES-GPU Approach to Grid Parallelization

- Utilization **of dynamic grid allocation** and **specialized data structures** (hashtables, used and free lists)
- **Flexible cell-to-thread assignment** and **extra synchronization algorithms** (atomic ops, barriers)
- **Parallel techniques** optimized for CUDA

Not GBEES, just a visual aid

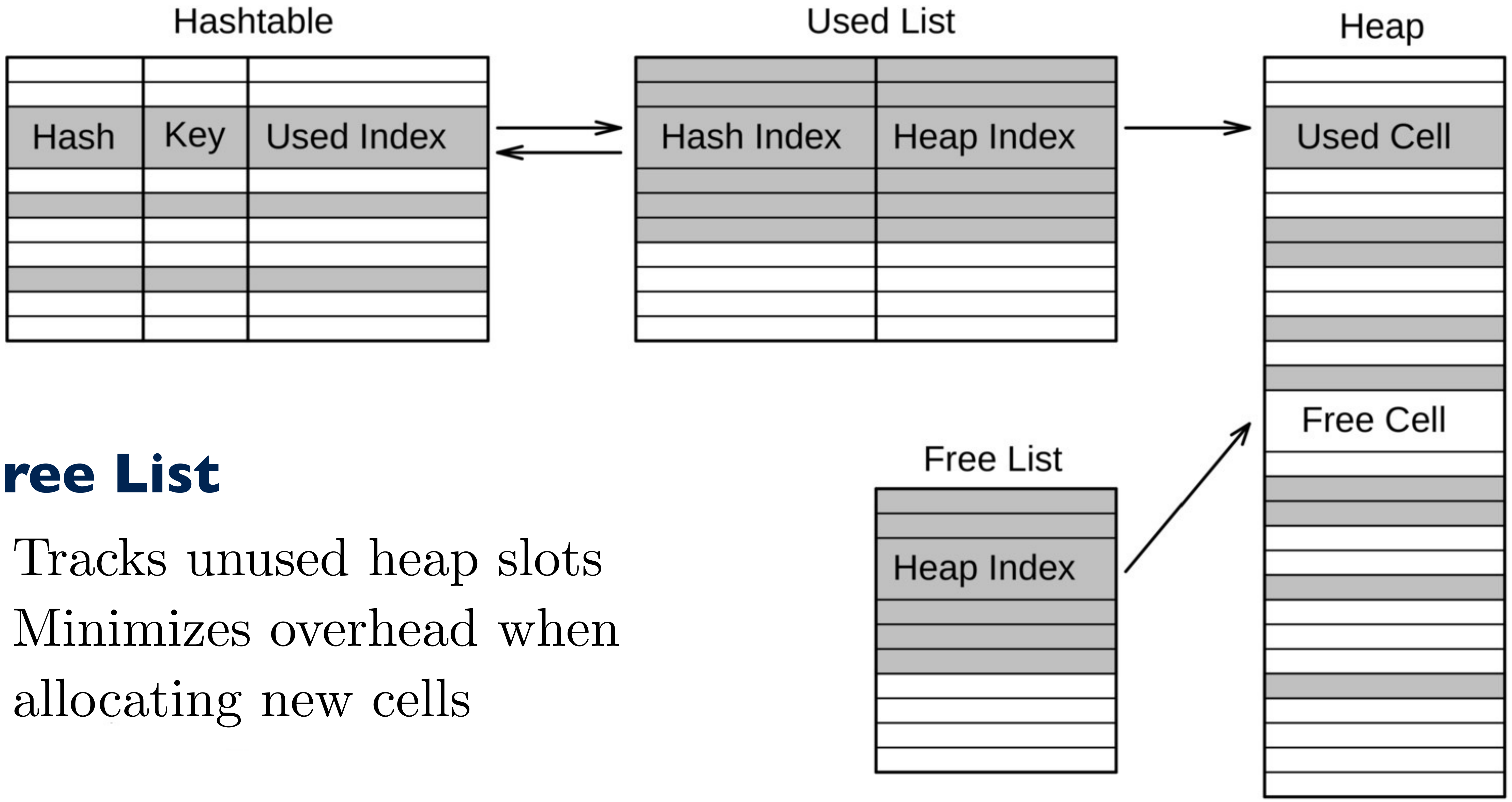
GBEES-GPU: Data structures

1. Hashtable

- Provides $\mathcal{O}(1)$ **random access** to cells by their grid index
- Enables fast neighbor lookups for cell-level operations and even workload distribution across active threads

2. Used List

- Maintains indices of active cells for efficient iteration during updates



3. Heap

- Stores the actual cell data
- Fixed-size allocation (due to CUDA constraints) sets the maximum number of cells per configuration

4. Free List

- Tracks unused heap slots
- Minimizes overhead when allocating new cells

GBEES-GPU: Implementation

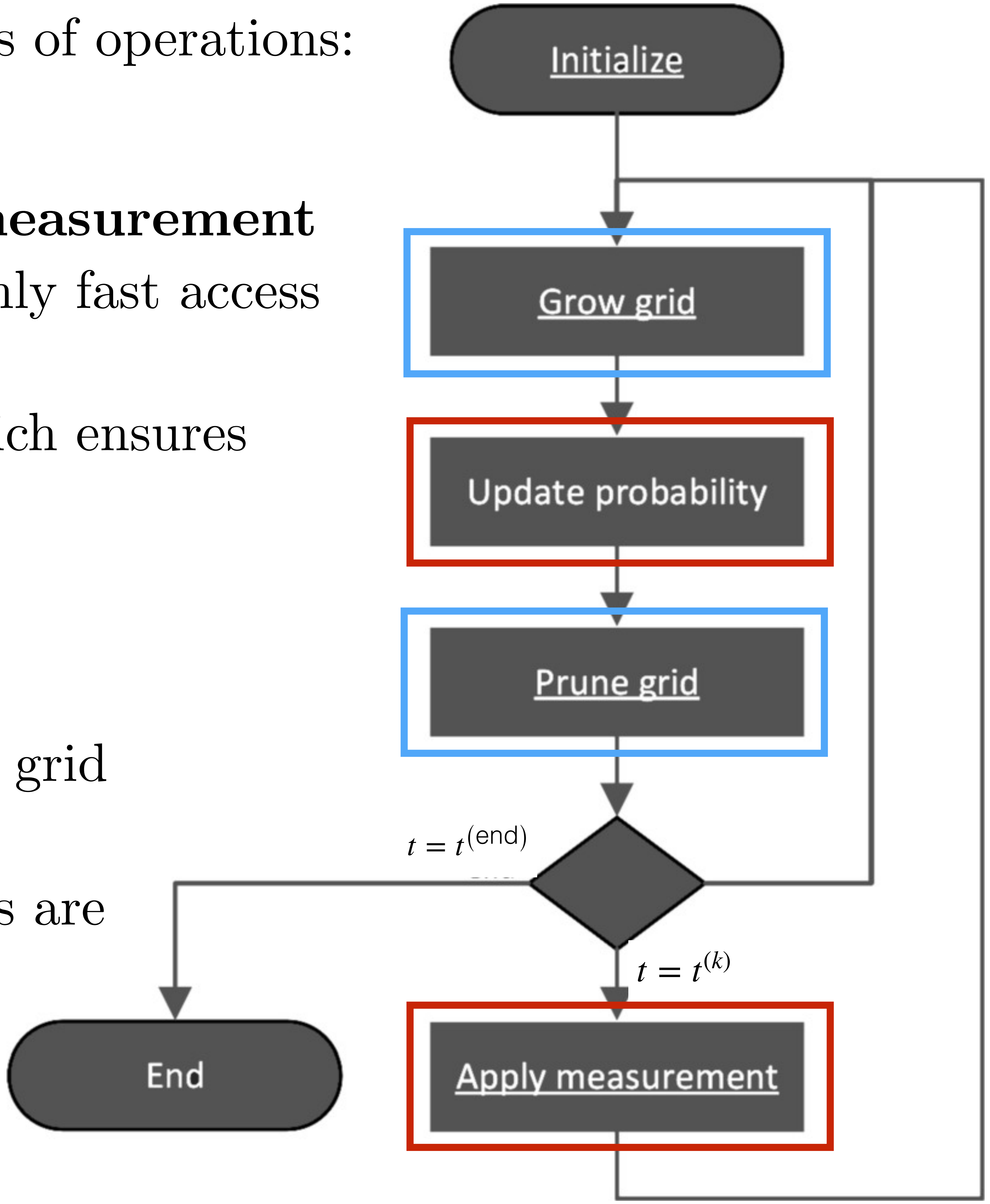
The implementation is divided into two main categories of operations:

1. Cell-level Operations

- Includes **updating probability** and **applying measurement**
- Each thread modifies its assigned cell, requiring only fast access to the cell itself and it's immediate neighbor
- This fast access is enabled via the **Used List**, which ensures memory locality and efficient thread execution

2. Grid-level Operations

- Includes **growing grid** and **pruning grid**
- Grid growth occurs at every step, but pruning the grid occurs every m steps defined by the user
- To maximize CUDA performance, these operations are coordinated using **atomic operations** and **synchronization barriers**



GBEES-GPU algorithm flow chart

GBEES-GPU: Synchronization Aspects

Grid Growing

- **Concurrent insertion** to avoid thread blocking
- Uses **callback initialization** for performance improvement after confirming cell uniqueness
- Race conditions prevented with **staged growth:**
  1. Forward axis → global sync
  2. Backward axis → global sync
  3. Diagonal directions → global sync

Algorithm 2 Concurrent Cell Creation.

```
Require: usedList and freeList are compact
hash ← BuzHash(i)
for count ∈ size(hashtable) do                                  ▷ linear probing
    hashIndex ← (hash + count) % size(hashtable)
    if hashtable[hashIndex] is free then                         ▷ current slot is empty
        usedIndex ← atomicAdd[size(usedList)]                    ▷ reserve used slot
        freeIndex ← atomicDec[size(freeList)]                    ▷ reserve free slot
        hashtable[hashIndex].i ← i                               ▷ update hashtable and lists
        usedList[usedIndex].heapIndex ← freeList[freeIndex]
        usedList[usedIndex].hashIndex ← hashIndex
        complete cell initialization with callback function
    else
        if hashtable[hashIndex].i is i then                      ▷ cell already exists
            break
        end if
    end if
end for
```

Grid Pruning

- Runs infrequently but needs **full parallelization**
  1. Mark low-probability cells for removal
  2. **Prefix sum (scan)** to compact Used List (double-buffer in shared memory)
  3. Add freed slots to Free List (atomic ops)
  4. **Rehash hash table** using double-buffer scheme

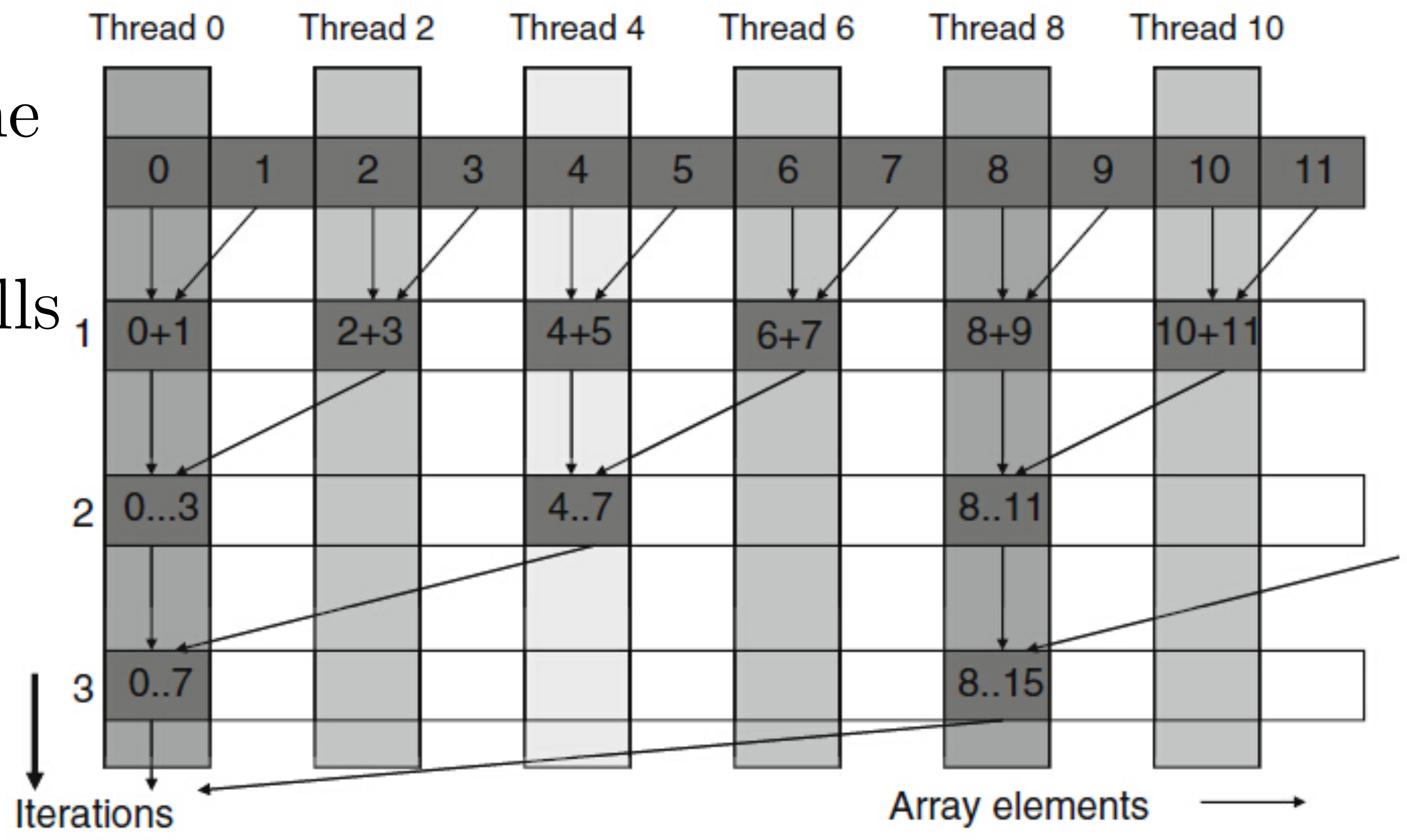
Algorithm 3 Grid Prune Operation.

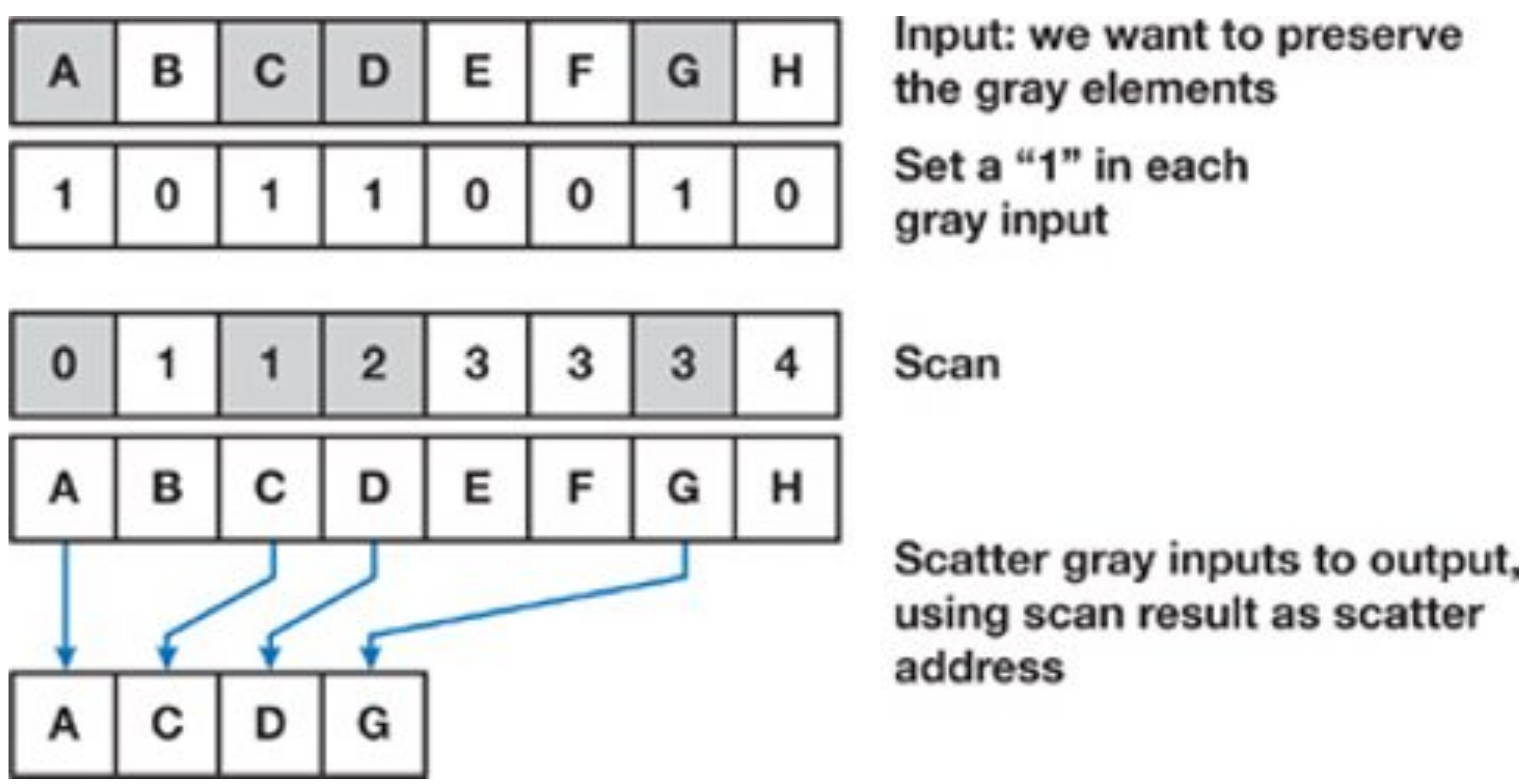
```
for i ∈ I do
    if i.p < p* and i is not a neighbor then
        i ← negligible
    end if
end for
perform a prefix sum process of usedList in shared memory
complete the prefix sum of usedList in global memory
compact usedList and update freeList
rehash hashtable
Ensure: perform a global synchronization at the end of each step.
```

GBEES-GPU: Parallel Reduction and Parallel Scan

Parallel Reduction — Normalization

- **Goal:** sum all grid-cell probabilities to normalize the distribution
- **Per-thread:** accumulate the sum of its assigned cells
- **Intra-block:** reduced in **shared memory** using **sequential addressing**
- **Outer reduction:** first thread of each block writes its block sum; followed by a **global reduction**
- **Output:** total probability





Parallel Scan — Prune/Compaction

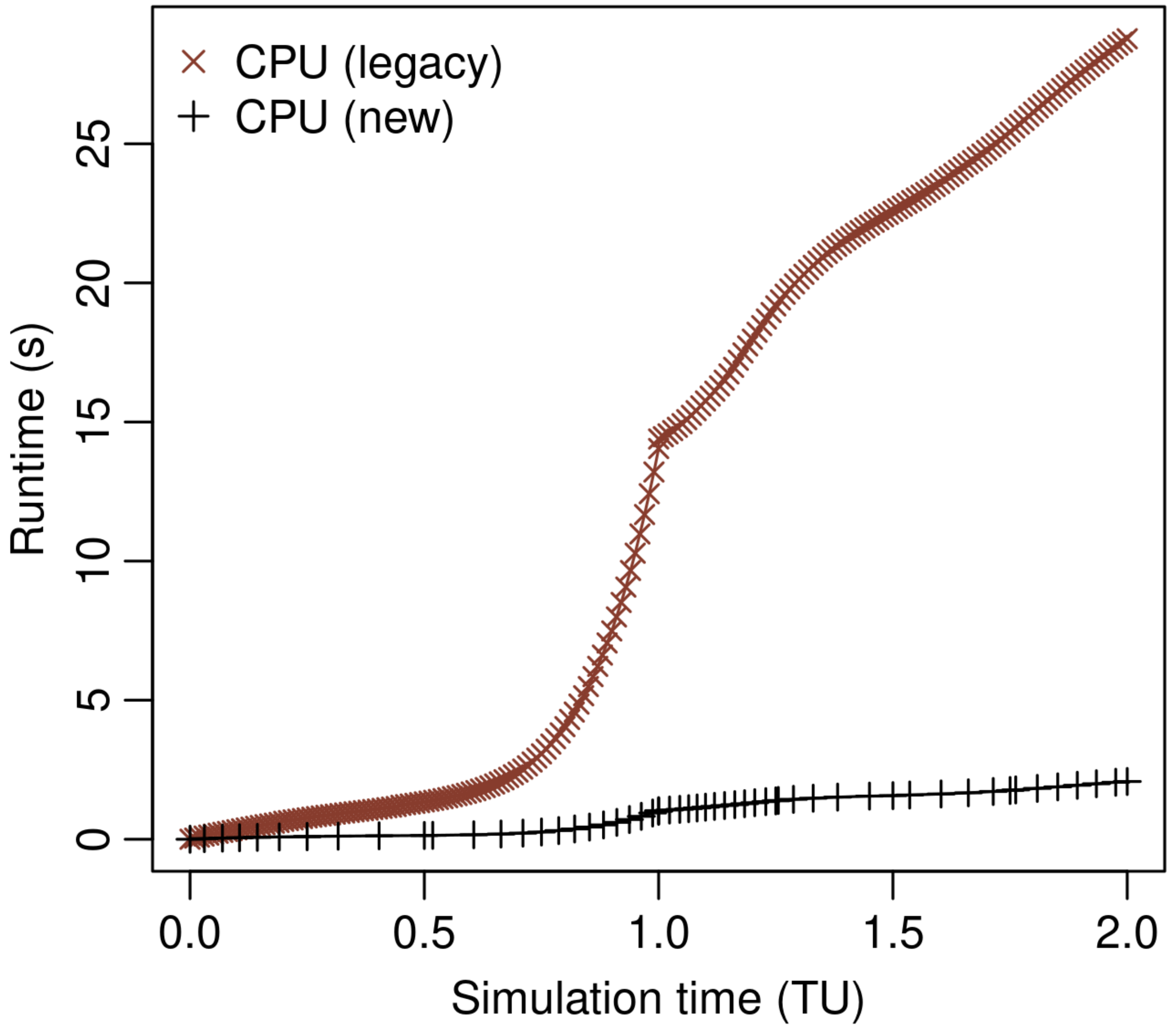
- **Goal:** compact the **Used List** (and update the **Free List**) during pruning
- **Intra-block: inclusive scan** with **sequential addressing** in shared memory using a **double buffer**
- **Accumulate the total of each block**
- **Outer exclusive scan**
- **Output:** compact Used List, updated Free List

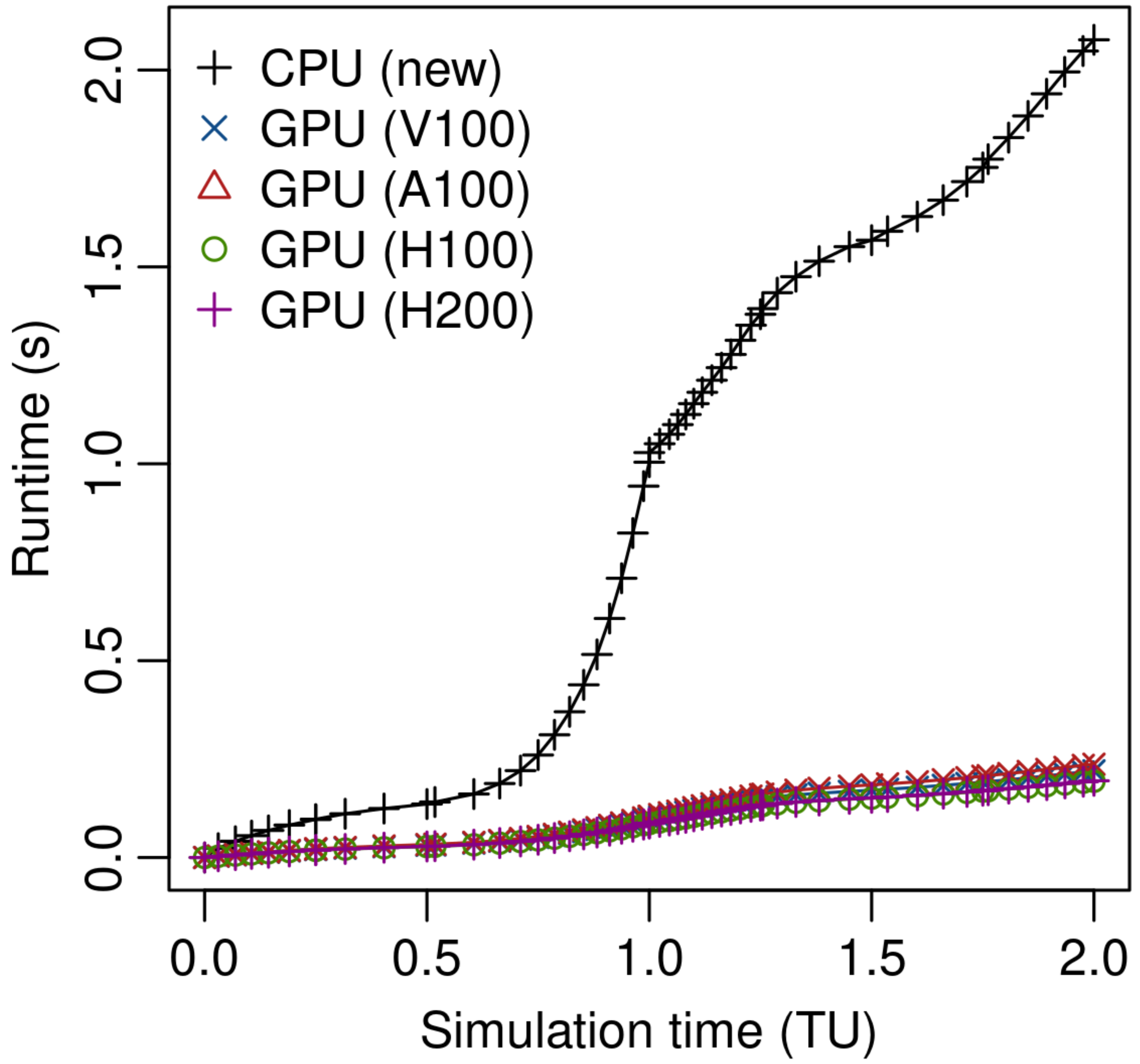
https://developer.nvidia.com/gpugems

Applications: Lorenz '63 Model

Revisiting Application: Lorenz '63 Model

- Analyzing the performance results, the CPU-optimized achieves a **13.9× speedup** compared to the CPU-legacy and the best GPU performance achieves a **9.2× speedup** when compared to the CPU-optimized





Device	Runtime (ms)	Cell/s	Speed-up
CPU-legacy Apple M2 MAX	28777	≈0.54M/s	0.072
CPU-optimized Apple M2 MAX	2077	≈3.13M/s	1
GPU 1: NVIDIA Tesla V100	244	≈26.6M/s	8.5
GPU 2: NVIDIA A100	258	≈25.2M/s	8.1
GPU 3: NVIDIA H100	226	≈28.8M/s	9.2
GPU 4: NVIDIA H200	230	≈28.3M/s	9.0

Applications: Lorenz '96 Model

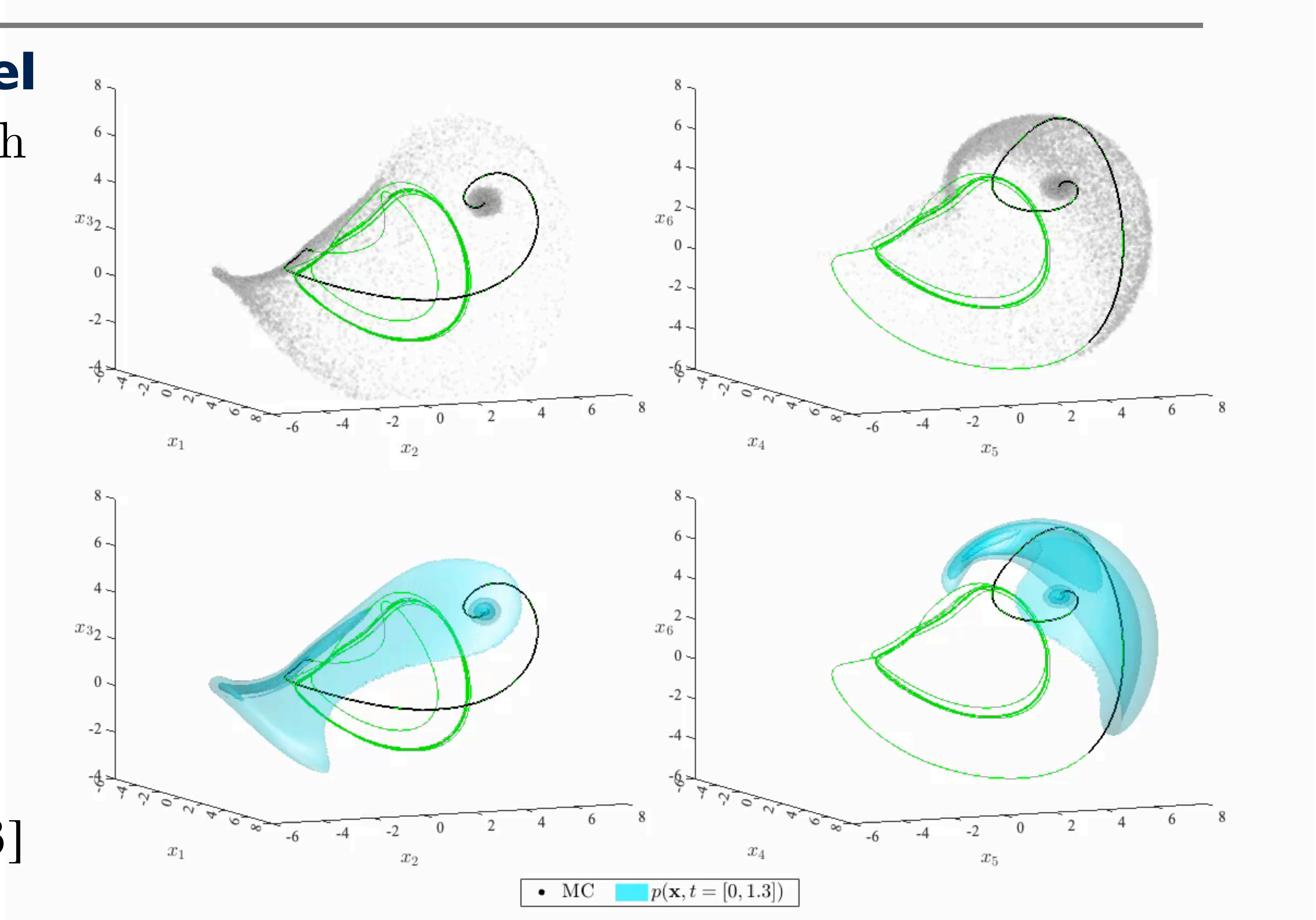
New Application: Lorenz '96 Model

- An n-dimensional chaotic attractor with equations of motion

$$\frac{dx_j}{dt} = \left(x_{j+1} + x_{j-2}\right) x_{j-1} - x_j + F,$$

where $\boldsymbol{x}^* = (F, \ldots, F)$ is an unstable equilibrium

- We use a 6D variation with $F = 4$ to compare our CPU-legacy, CPU-optimized and GPU versions, propagating uncertainty from $t = [0,1.3]$ with no measurement updates



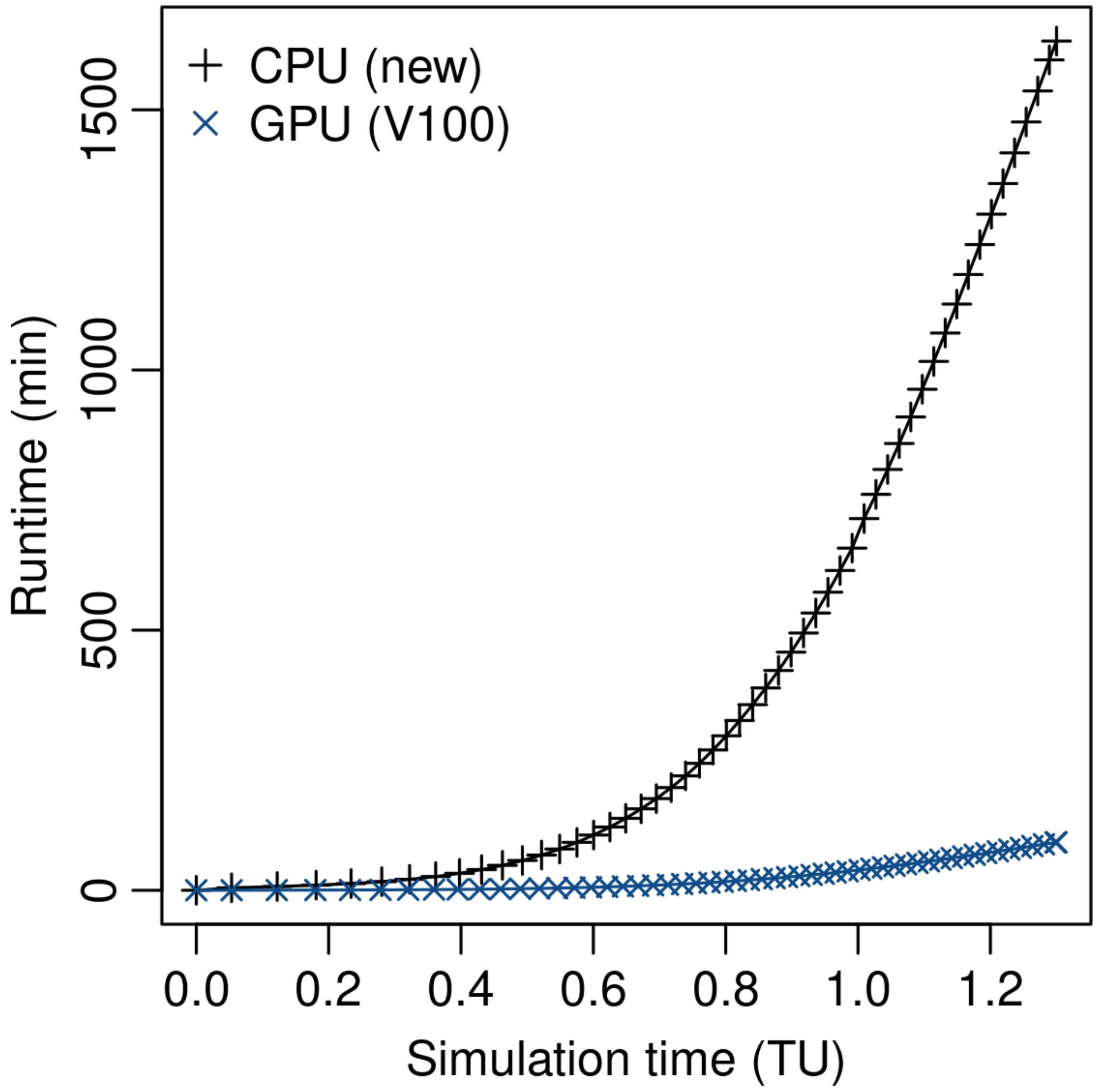
Initial uncertainty of $\sigma_{x_j} = 0.2$ and grid width of $\Delta x_j = 0.1$ for $j = 1, \ldots, 6$

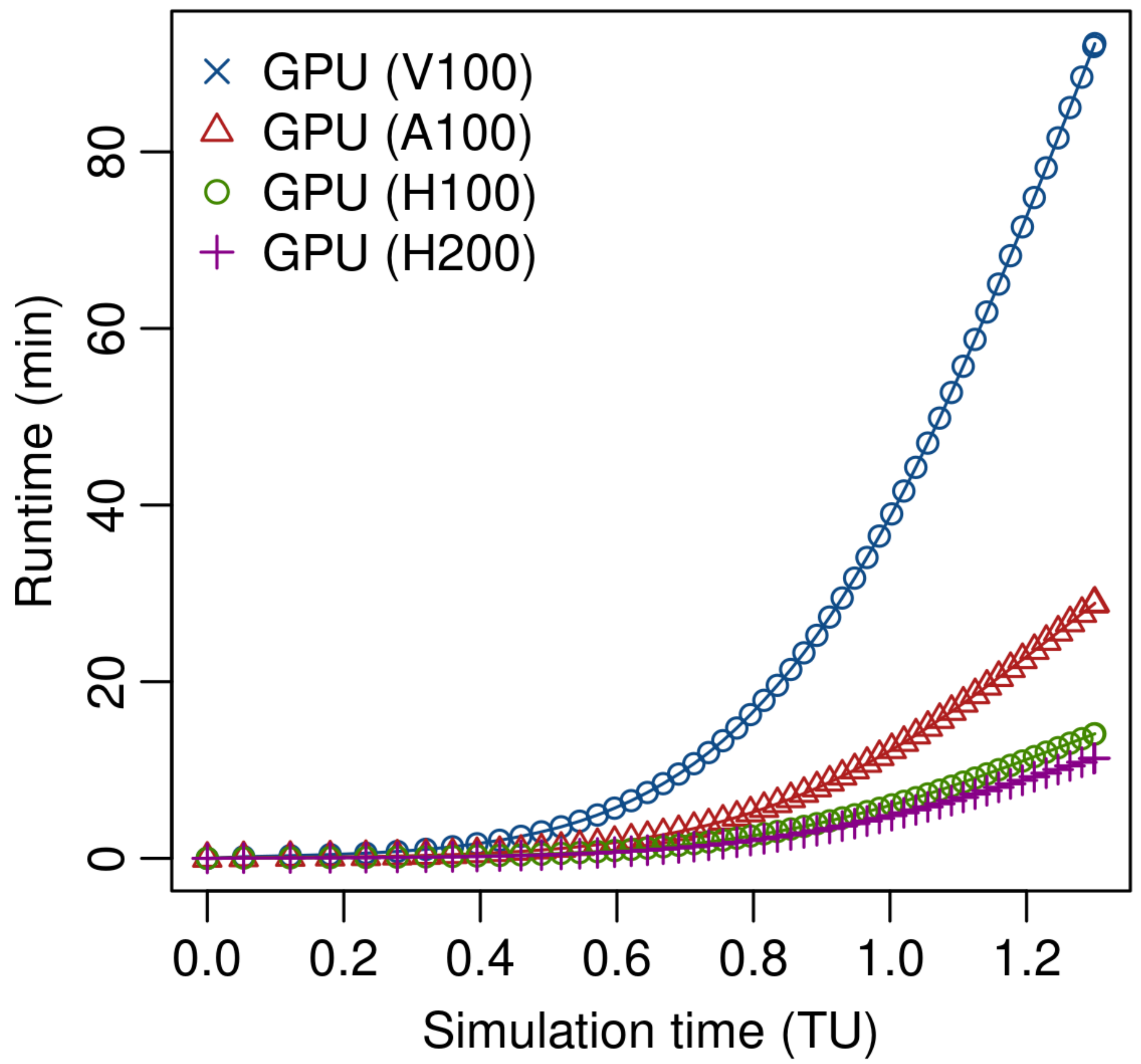
- To convert the discretized 6D PDFs into two, 3D PDFs, we numerically integrate:

$$p(x_1, x_2, x_3, t) = \int_{\min(x_6)}^{\min(x_6)} \int_{\min(x_5)}^{\min(x_5)} \int_{\min(x_4)}^{\min(x_4)} p(\boldsymbol{x}, t) dx_4\, dx_5\, dx_6 \quad \text{and} \quad p(x_4, x_5, x_6, t) = \int_{\min(x_3)}^{\min(x_3)} \int_{\min(x_2)}^{\min(x_2)} \int_{\min(x_1)}^{\min(x_1)} p(\boldsymbol{x}, t) dx_1\, dx_2\, dx_3$$

New Application: Lorenz ’96 Model

- Due to dimensionality, this example is computationally infeasible for CPU-legacy version, but the best GPU performance achieves a 132.5× **speedup** when compared to the CPU-optimized
- This implies a $\sim 10^3\times$ **speedup** when compared to the CPU-legacy

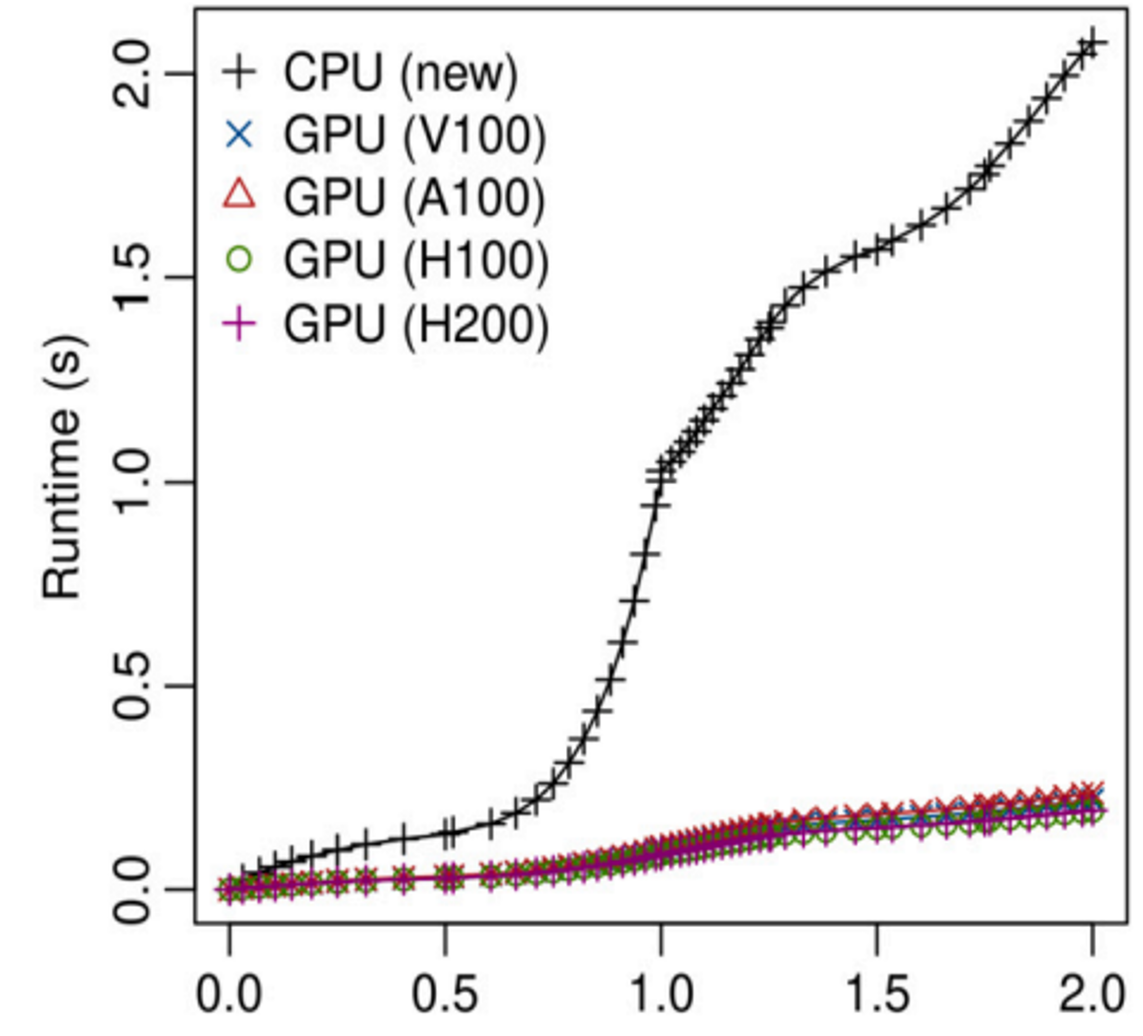




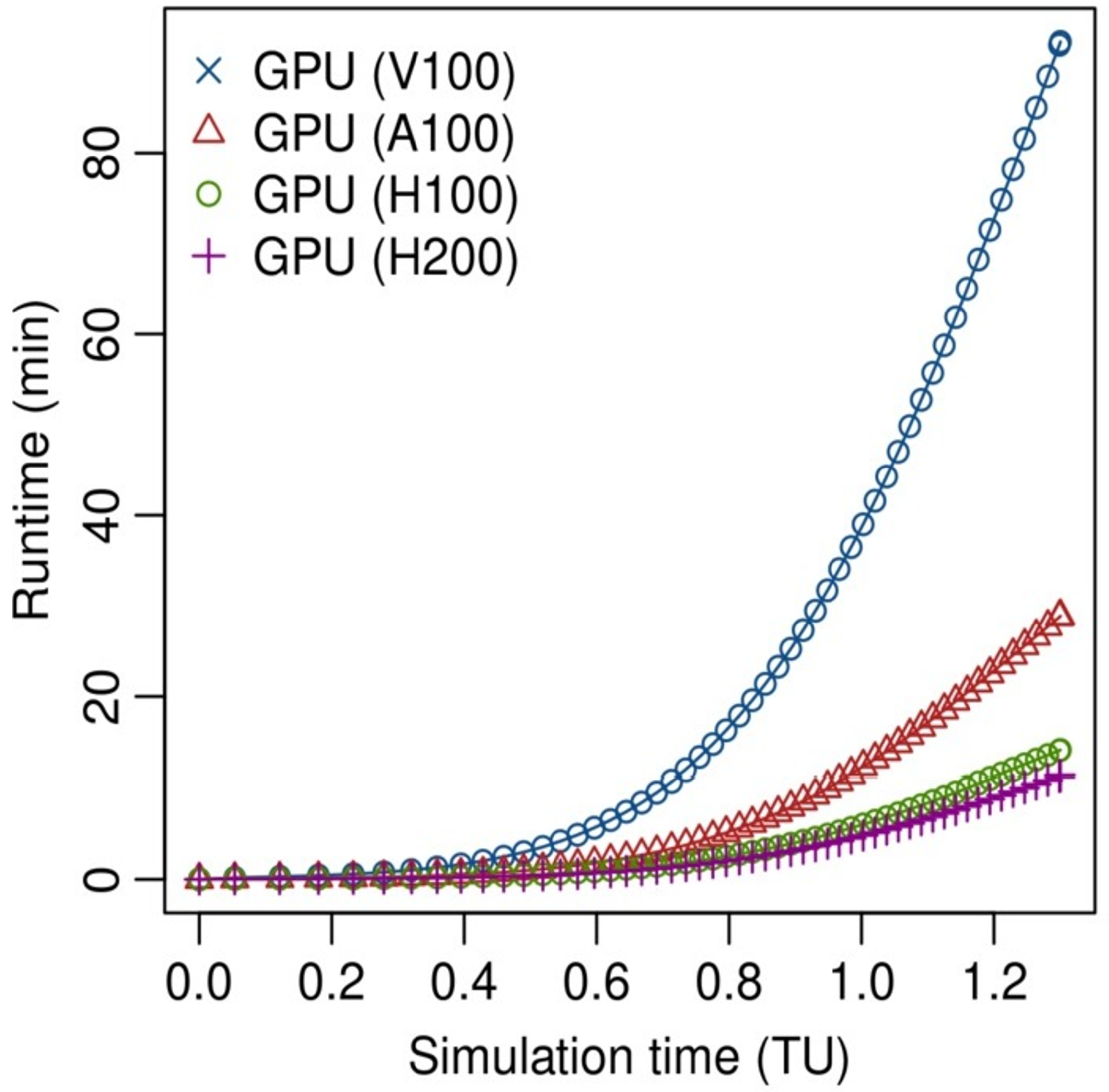
Device	Runtime (s)	Cell/s	Speed-up
CPU-optimized Apple M2 MAX	97927	≈0.3M/s	1
GPU 1: NVIDIA Tesla V100	5513	≈5.4M/s	17.8
GPU 2: NVIDIA A100	1736	≈17.3M/s	56.4
GPU 3: NVIDIA H100	919	≈32.6M/s	106.6
GPU 4: NVIDIA H200	739	≈40.6M/s	132.5

Conclusions

- CPU optimization and GPU execution make Eulerian uncertainty propagation for six-dimensional systems **computationally feasible**
- Performance of the new GBEES implementations depends on **grid size** and **GPU occupancy:**
  - **Lorenz '63:** grid too small for full GPU utilization → modest gains
    - CUDA version: **8.5-9.0× faster** than optimized by CPU
  - **Lorenz '96:** high computational load → fully exploits GPU parallelism
    - On V100: **17.8× faster** than optimized CPU
    - On A100: **56.4× faster**
    - On H100: **106.6× faster**
    - On H200: **132.5× faster**
- When compared to GBEES CPU-legacy, the results of the Lorenz '96 application are a ∼10× **speedup** in the **CPU-optimized** version and an implied $\sim 10^3\times$ **speedup** in the **GPU** version



Lorenz '63 model application



Lorenz '96 model application

This investigation was supported by the NASA Space Technology Graduate Research Opportunities Fellowship (Grant #80NSSC23K1219)

CPC Paper

GBEES CPU-optimized

GBEES-GPU

Thank you to everyone that attended this Cassyni CPC Seminar!