High-Performance and Scalable Agent-Based Simulation With BioDynaMo

Lukas Breitwieser, Ahmad Hesam, Fons Rademakers,
Juan Gómez Luna, and Onur Mutlu

https://arxiv.org/abs/2301.06984, https://doi.org/10.1145/3572848.3577480

ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming
27 February 2023, Montreal Canada

CERN ETH zürich SAFARI TU Delft

Key problem

Most existing agent-based simulation platforms exhibit **low performance**

Impact of low performance

Impact of low performance

Limits the model size and complexity

Impact of low performance

Impact of low performance

Limits the model size and complexity

Increases the development time

Limits the capability to optimize parameters

→ might lead to less optimal solution

Impact of low performance

Limits the model size and complexity

Increases the development time

Limits the capability to optimize parameters

→ might lead to less optimal solution

Increases costs

Our solution: **BioDynaMo**

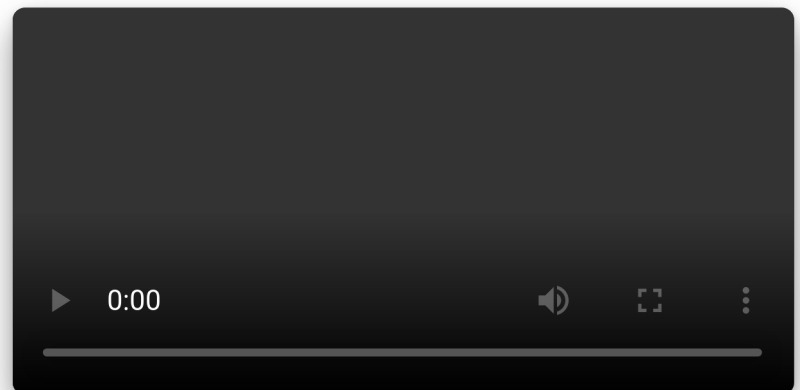
BioDynaMo is a **modular and high-performance** agent-based simulation platform written in C++.

https://biodynamo.org

Background

Example agent-based model

Modeling a swarm of birds



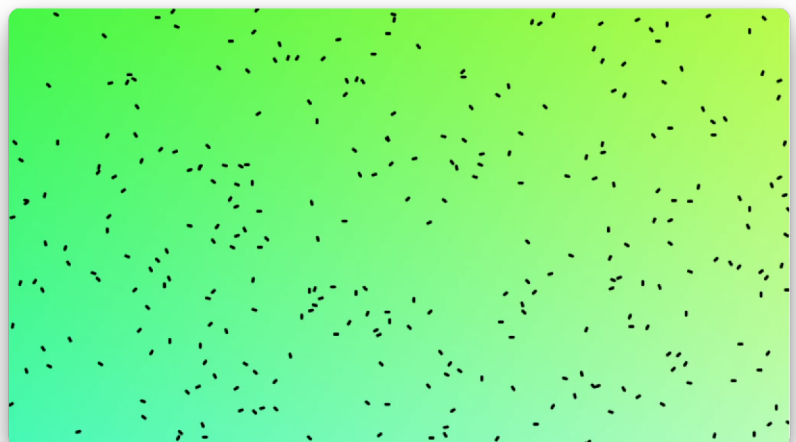
Source: Craig Reynolds, 1987, https://doi.org/10.1145/37401.37406, **Simulator:** Thierry Escande https://github.com/tescande/boids.git

Example agent-based model

Modeling a swarm of birds

Agent is a bird with the following attributes:

- position
- velocity



Source: Craig Reynolds, 1987, https://doi.org/10.1145/37401.37406, **Simulator:** Thierry Escande https://github.com/tescande/boids.git

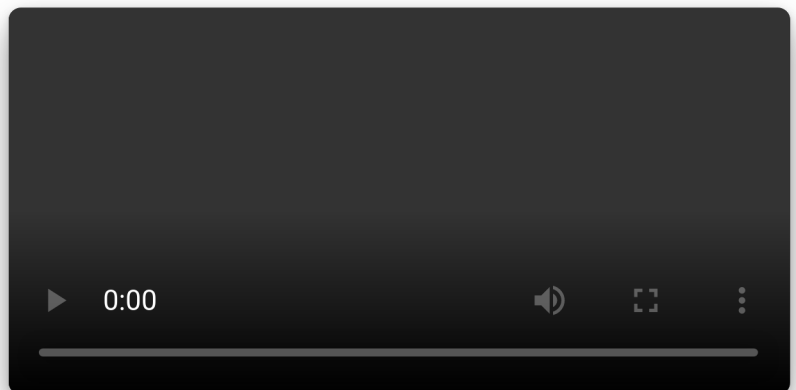
Example agent-based model

Modeling a swarm of birds

Agent is a bird with the following attributes:

- position
- velocity

and three behaviors



Source: Craig Reynolds, 1987, https://doi.org/10.1145/37401.37406, **Simulator:** Thierry Escande https://github.com/tescande/boids.git

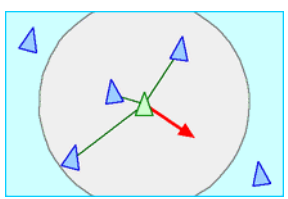
Example agent-based model

Modeling a swarm of birds

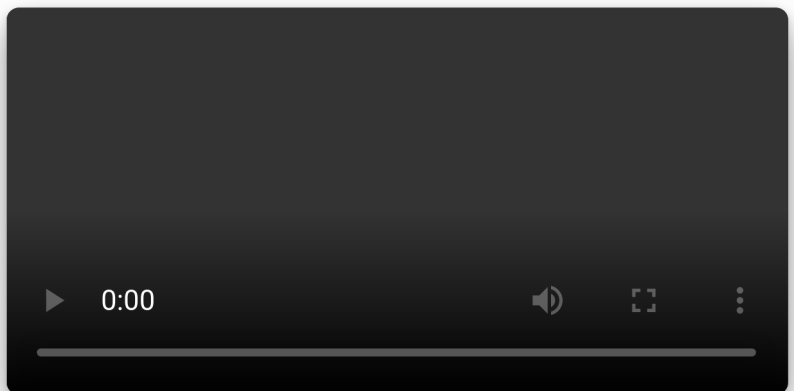
Agent is a bird with the following attributes:

- position
- velocity

and three behaviors



Separation



Source: Craig Reynolds, 1987, https://doi.org/10.1145/37401.37406, **Simulator:** Thierry Escande https://github.com/tescande/boids.git

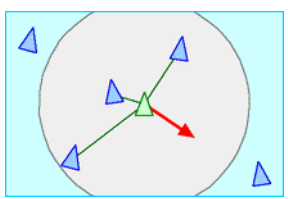
Example agent-based model

Modeling a swarm of birds

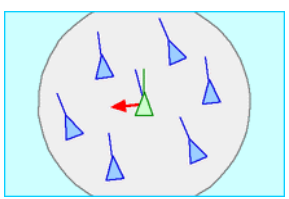
Agent is a bird with the following attributes:

- position
- velocity

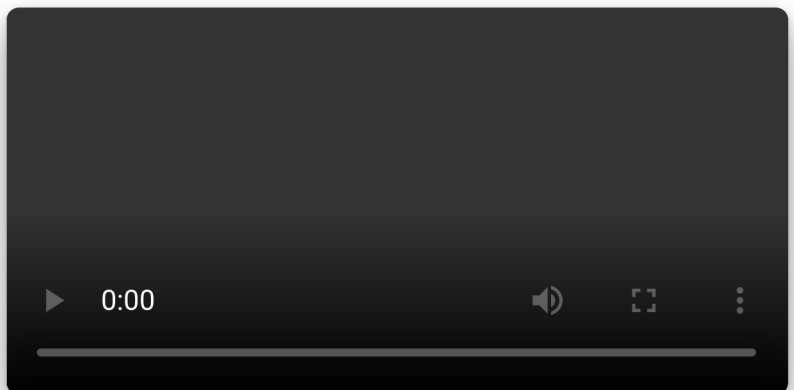
and three behaviors



Separation



Alignment



Source: Craig Reynolds, 1987, https://doi.org/10.1145/37401.37406, **Simulator:** Thierry Escande https://github.com/tescande/boids.git

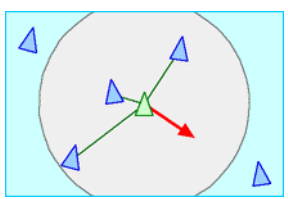
Example agent-based model

Modeling a swarm of birds

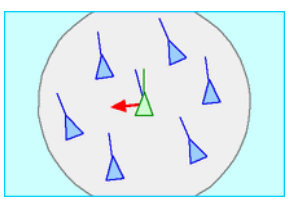
Agent is a bird with the following attributes:

- position
- velocity

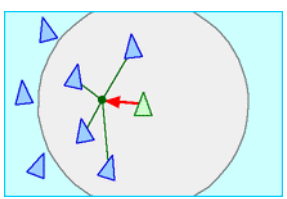
and three behaviors



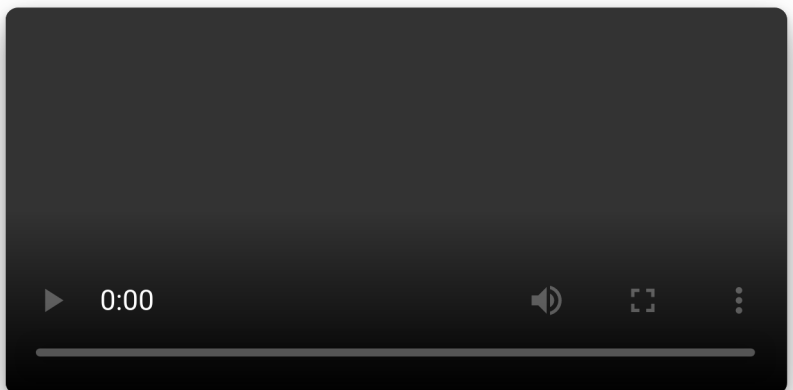
Separation



Alignment

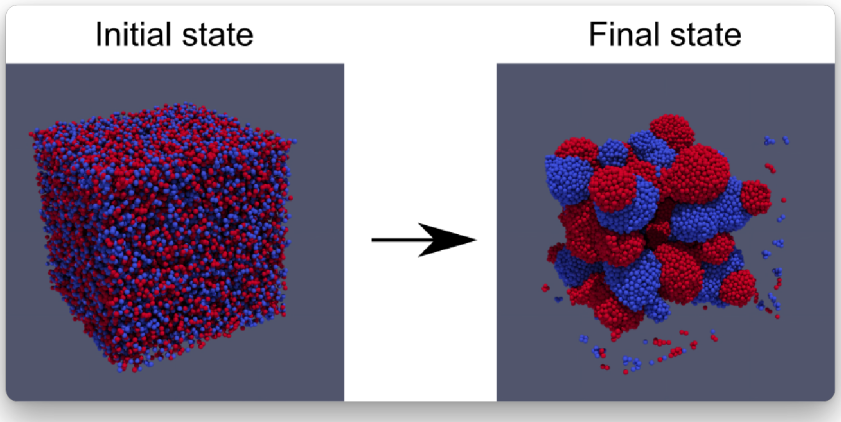


Cohesion



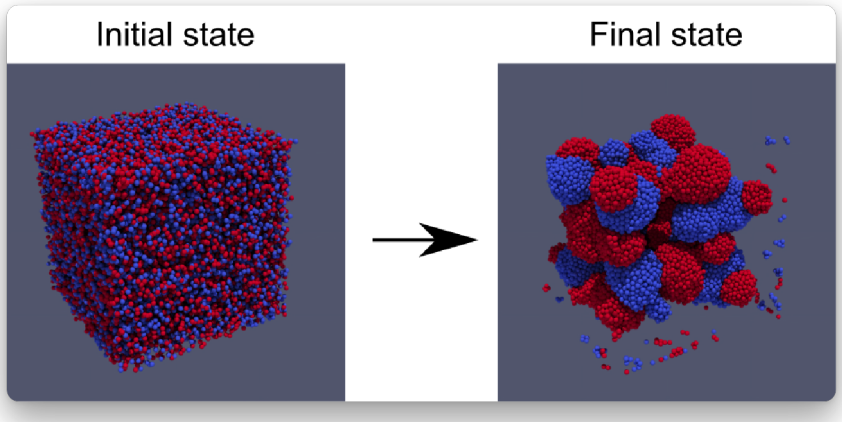
Source: Craig Reynolds, 1987, https://doi.org/10.1145/37401.37406, **Simulator:** Thierry Escande https://github.com/tescande/boids.git

Cell sorting in BioDynaMo



Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Cell sorting in BioDynaMo



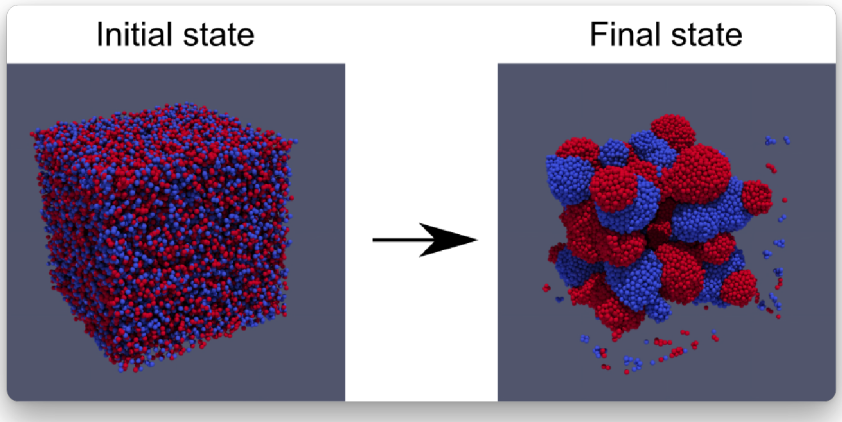
Agent

Spherical cell with type attribute (red and blue)

Behaviors

Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

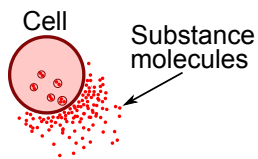
Cell sorting in BioDynaMo



Agent

Spherical cell with type attribute (red and blue)

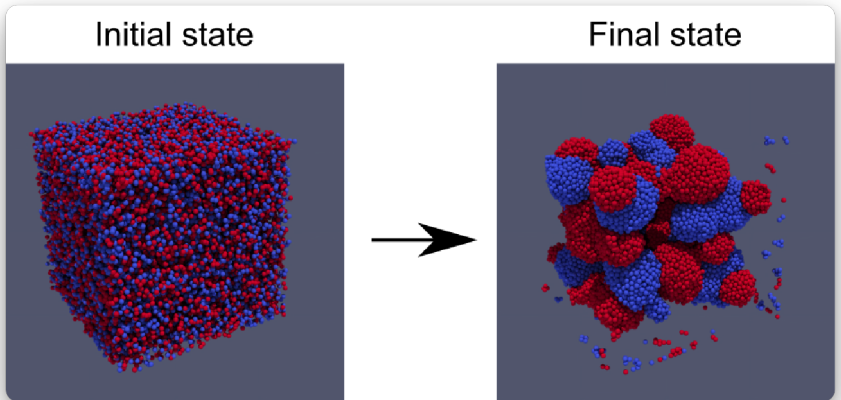
Behaviors



Substance secretion

Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

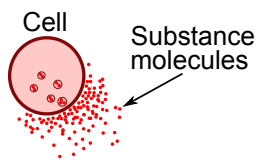
Cell sorting in BioDynaMo



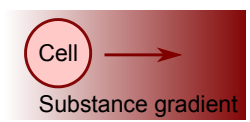
Agent

Spherical cell with type attribute (red and blue)

Behaviors



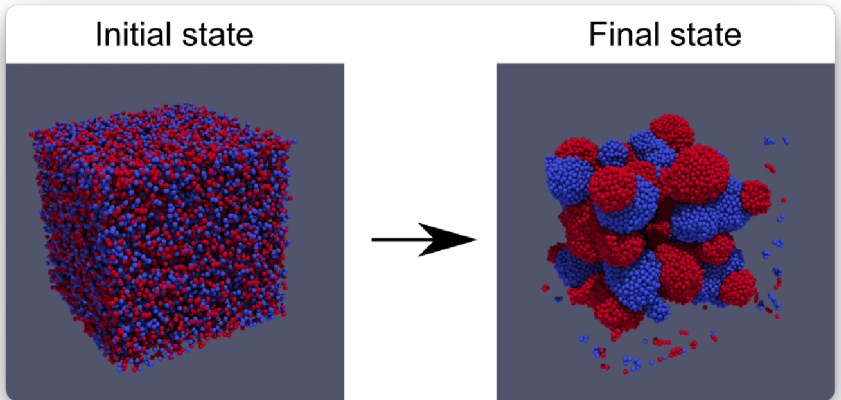
Substance secretion



Chemotaxis

Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

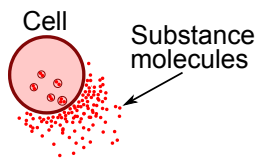
Cell sorting in BioDynaMo



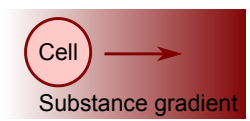
Agent

Spherical cell with type attribute (red and blue)

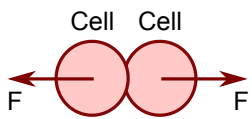
Behaviors



Substance secretion



Chemotaxis



Mechanical forces

Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Cell sorting implementation

```
// Model initialization
AddSubstance('purple');
AddSubstance('light-blue');
CreateAgentsRandom(number, 'red');
CreateAgentsRandom(number, 'blue');

// Simulation
for each iteration {
   UpdateNeighborSearchIndex();

   for each agent {
     SecreteSubstance(agent);
     FollowSubstanceGradient(agent);
     CalculateForces(agent);
   }

   DiffuseSubstance('purple');
   DiffuseSubstance('light-blue');

   Visualize();
}
```

Cell sorting implementation

```
// Model initialization
AddSubstance('purple');
AddSubstance('light-blue');
CreateAgentsRandom(number, 'red');
CreateAgentsRandom(number, 'blue');

// Simulation
for each iteration {
   UpdateNeighborSearchIndex();

   for each agent {
     SecreteSubstance(agent);
     FollowSubstanceGradient(agent);
     CalculateForces(agent);
   }

   DiffuseSubstance('purple');
   DiffuseSubstance('light-blue');

   Visualize();
}
```

Cell sorting implementation

```
// Model initialization
AddSubstance('purple');
AddSubstance('light-blue');
CreateAgentsRandom(number, 'red');
CreateAgentsRandom(number, 'blue');

// Simulation
for each iteration {
   UpdateNeighborSearchIndex();

   for each agent {
     SecreteSubstance(agent);
     FollowSubstanceGradient(agent);
     CalculateForces(agent);
   }

   DiffuseSubstance('purple');
   DiffuseSubstance('light-blue');

   Visualize();
}
```

Cell sorting implementation

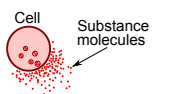
```
// Model initialization
AddSubstance('purple');
AddSubstance('light-blue');
CreateAgentsRandom(number, 'red');
CreateAgentsRandom(number, 'blue');

// Simulation
for each iteration {
   UpdateNeighborSearchIndex();

   for each agent {
     SecreteSubstance(agent);
     FollowSubstanceGradient(agent);
     CalculateForces(agent);
   }

   DiffuseSubstance('purple');
   DiffuseSubstance('light-blue');

   Visualize();
}
```

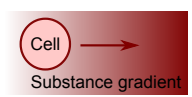
Cell sorting implementation

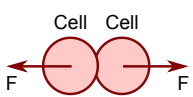
```
// Model initialization
AddSubstance('purple');
AddSubstance('light-blue');
CreateAgentsRandom(number, 'red');
CreateAgentsRandom(number, 'blue');

// Simulation
for each iteration {
   UpdateNeighborSearchIndex();

   for each agent {
     SecreteSubstance(agent);
     FollowSubstanceGradient(agent);
     CalculateForces(agent);
   }

   DiffuseSubstance('purple');
   DiffuseSubstance('light-blue');

   Visualize();
}
```

Cell sorting implementation

```
1   // Model initialization
2   AddSubstance('purple');
3   AddSubstance('light-blue');
4   CreateAgentsRandom(number, 'red');
5   CreateAgentsRandom(number, 'blue');
6
7   // Simulation
8   for each iteration {
9      UpdateNeighborSearchIndex();
10
11     for each agent {
12       SecreteSubstance(agent);
13       FollowSubstanceGradient(agent);
14       CalculateForces(agent);
15     }
16
17     DiffuseSubstance('purple');
18     DiffuseSubstance('light-blue');
19
20     Visualize();
21  }
```







Cell sorting implementation

```
// Model initialization
AddSubstance('purple');
AddSubstance('light-blue');
CreateAgentsRandom(number, 'red');
CreateAgentsRandom(number, 'blue');

// Simulation
for each iteration {
   UpdateNeighborSearchIndex();

   for each agent {
     SecreteSubstance(agent);
     FollowSubstanceGradient(agent);
     CalculateForces(agent);
   }

   DiffuseSubstance('purple');
   DiffuseSubstance('light-blue');

   Visualize();
}
```

Cell sorting implementation

```
// Model initialization
AddSubstance('purple');
AddSubstance('light-blue');
CreateAgentsRandom(number, 'red');
CreateAgentsRandom(number, 'blue');

// Simulation
for each iteration {
   UpdateNeighborSearchIndex();

   for each agent {
     SecreteSubstance(agent);
     FollowSubstanceGradient(agent);
     CalculateForces(agent);
   }

   DiffuseSubstance('purple');
   DiffuseSubstance('light-blue');

   Visualize();
}
```

Cell sorting implementation

```
// Model initialization
AddSubstance('purple');
AddSubstance('light-blue');
CreateAgentsRandom(number, 'red');
CreateAgentsRandom(number, 'blue');

// Simulation
for each iteration {
   UpdateNeighborSearchIndex();

   for each agent {
     SecreteSubstance(agent);
     FollowSubstanceGradient(agent);
     CalculateForces(agent);
   }

   DiffuseSubstance('purple');
   DiffuseSubstance('light-blue');

   Visualize();
}
```

Fundamental operations

Cell sorting implementation

```
1   // Model initialization
2   AddSubstance('purple');
3   AddSubstance('light-blue');
4   CreateAgentsRandom(number, 'red');
5   CreateAgentsRandom(number, 'blue');
6
7   // Simulation
8   for each iteration {
9      UpdateNeighborSearchIndex();
10
11     for each agent {
12       SecreteSubstance(agent);
13       FollowSubstanceGradient(agent);
14       CalculateForces(agent);
15     }
16
17     DiffuseSubstance('purple');
18     DiffuseSubstance('light-blue');
19
20     Visualize();
21  }
```

Fundamental operations

- Iterate over agents and execute a function

Cell sorting implementation

```
1   // Model initialization
2   AddSubstance('purple');
3   AddSubstance('light-blue');
4   CreateAgentsRandom(number, 'red');
5   CreateAgentsRandom(number, 'blue');
6
7   // Simulation
8   for each iteration {
9      UpdateNeighborSearchIndex();
10
11     for each agent {
12       SecreteSubstance(agent);
13       FollowSubstanceGradient(agent);
14       CalculateForces(agent);
15     }
16
17     DiffuseSubstance('purple');
18     DiffuseSubstance('light-blue');
19
20     Visualize();
21  }
```

Fundamental operations

- Iterate over agents and execute a function
- Determine neighbors

Cell sorting implementation

```
1   // Model initialization
2   AddSubstance('purple');
3   AddSubstance('light-blue');
4   CreateAgentsRandom(number, 'red');
5   CreateAgentsRandom(number, 'blue');
6
7   // Simulation
8   for each iteration {
9      UpdateNeighborSearchIndex();
10
11     for each agent {
12       SecreteSubstance(agent);
13       FollowSubstanceGradient(agent);
14       CalculateForces(agent);
15     }
16
17     DiffuseSubstance('purple');
18     DiffuseSubstance('light-blue');
19
20     Visualize();
21  }
```

Fundamental operations

- Iterate over agents and execute a function
- Determine neighbors
- Iterate over neighbors and execute a function

Cell sorting implementation

```
// Model initialization
AddSubstance('purple');
AddSubstance('light-blue');
CreateAgentsRandom(number, 'red');
CreateAgentsRandom(number, 'blue');

// Simulation
for each iteration {
   UpdateNeighborSearchIndex();

   for each agent {
     SecreteSubstance(agent);
     FollowSubstanceGradient(agent);
     CalculateForces(agent);
   }

   DiffuseSubstance('purple');
   DiffuseSubstance('light-blue');

   Visualize();
}
```

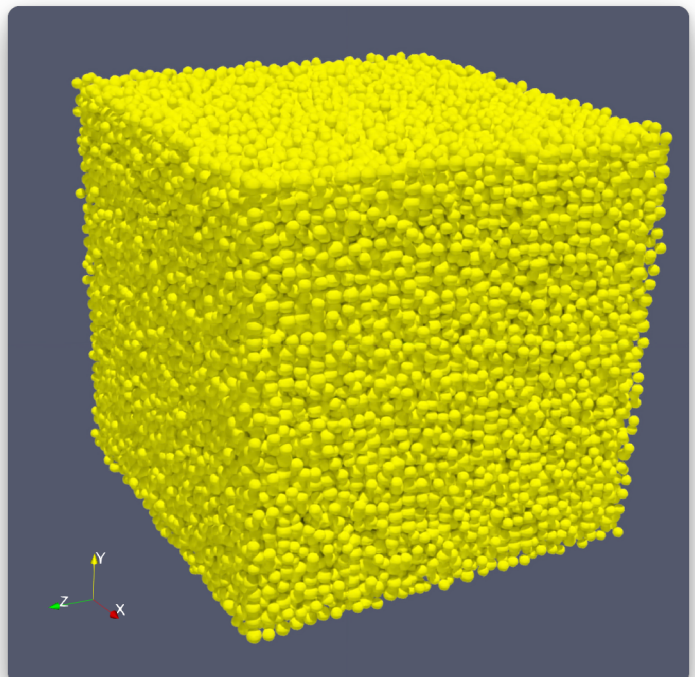
Fundamental operations

- Iterate over agents and execute a function
- Determine neighbors
- Iterate over neighbors and execute a function
- Add and remove agents

Benchmark simulations

The selected simulations cover a broad spectrum of performance related simulation characteristics

Characteristic	Cell proliferation	Cell clustering	Epidemiology	Neuroscience	Oncology
Create new agents during simulation	✗			✗	✗
Delete agents during simulation					✗
Agents modify neighbors				✗	
Load imbalance			✗	✗	
Agents move randomly			✗		✗
Simulation uses diffusion		✗		✗	
Simulation has static regions				✗	
Number of iterations	500	1000	1000	500	288
Number of agents (in millions)	12.6	2	10	9	10
Number of diffusion volumes	0	54m	0	65k	0

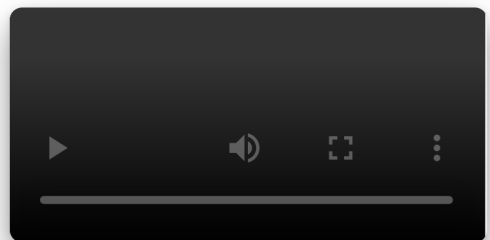


Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Benchmark simulations

The selected simulations cover a broad spectrum of performance related simulation characteristics

Characteristic	Cell proliferation	Cell clustering	Epidemiology	Neuroscience	Oncology
Create new agents during simulation	✗			✗	✗
Delete agents during simulation					✗
Agents modify neighbors				✗	
Load imbalance			✗	✗	
Agents move randomly			✗		✗
Simulation uses diffusion		✗		✗	
Simulation has static regions				✗	
Number of iterations	500	1000	1000	500	288
Number of agents (in millions)	12.6	2	10	9	10
Number of diffusion volumes	0	54m	0	65k	0

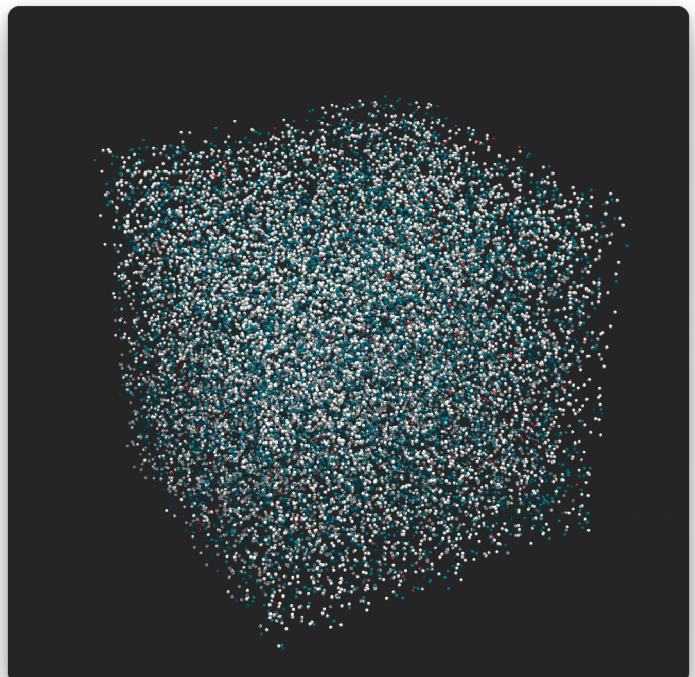


Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Benchmark simulations

The selected simulations cover a broad spectrum of performance related simulation characteristics

Characteristic	Cell proliferation	Cell clustering	Epidemiology	Neuroscience	Oncology
Create new agents during simulation	✗			✗	✗
Delete agents during simulation					✗
Agents modify neighbors				✗	
Load imbalance			✗	✗	
Agents move randomly			✗		✗
Simulation uses diffusion		✗		✗	
Simulation has static regions				✗	
Number of iterations	500	1000	1000	500	288
Number of agents (in millions)	12.6	2	10	9	10
Number of diffusion volumes	0	54m	0	65k	0

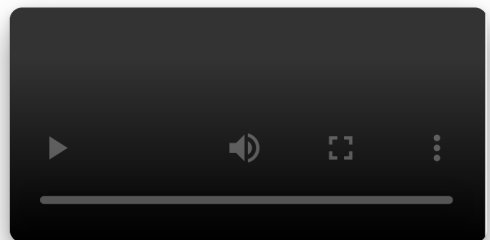


Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Benchmark simulations

The selected simulations cover a broad spectrum of performance related simulation characteristics

Characteristic	Cell proliferation	Cell clustering	Epidemiology	Neuroscience	Oncology
Create new agents during simulation	✗			✗	✗
Delete agents during simulation					✗
Agents modify neighbors				✗	
Load imbalance			✗	✗	
Agents move randomly			✗		✗
Simulation uses diffusion		✗		✗	
Simulation has static regions				✗	
Number of iterations	500	1000	1000	500	288
Number of agents (in millions)	12.6	2	10	9	10
Number of diffusion volumes	0	54m	0	65k	0

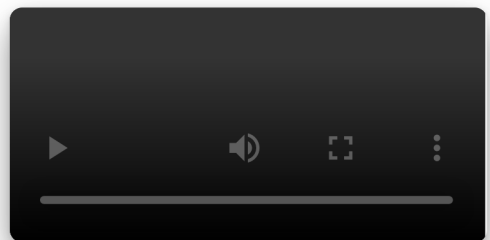


Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Benchmark simulations

The selected simulations cover a broad spectrum of performance related simulation characteristics

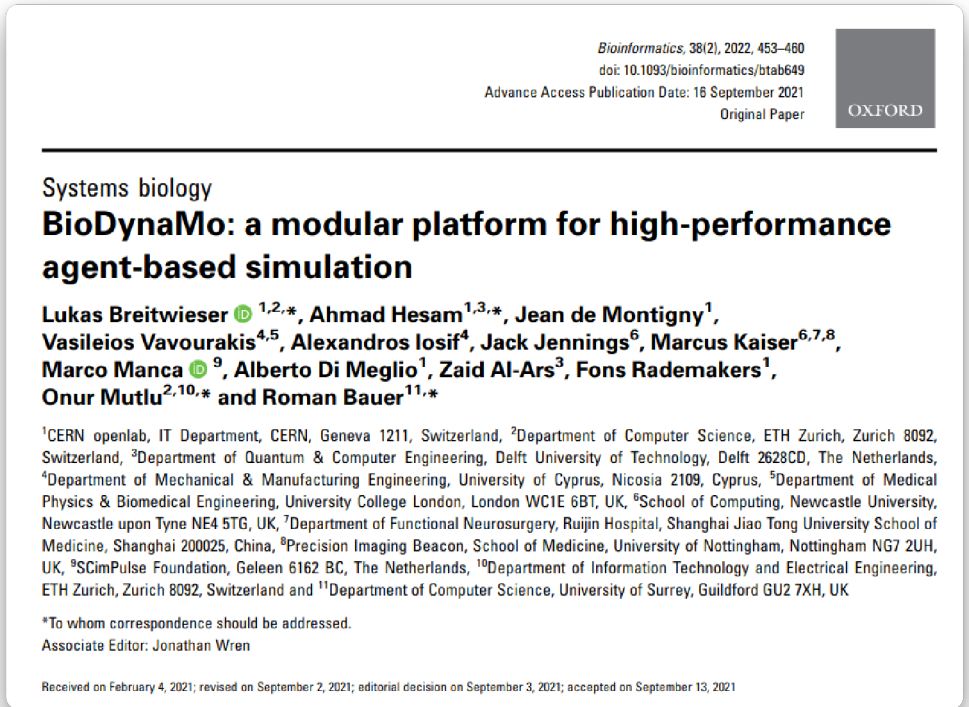
Characteristic	Cell proliferation	Cell clustering	Epidemiology	Neuroscience	Oncology
Create new agents during simulation	✗			✗	✗
Delete agents during simulation					✗
Agents modify neighbors				✗	
Load imbalance			✗	✗	
Agents move randomly			✗		✗
Simulation uses diffusion		✗		✗	
Simulation has static regions				✗	
Number of iterations	500	1000	1000	500	288
Number of agents (in millions)	12.6	2	10	9	10
Number of diffusion volumes	0	54m	0	65k	0



Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

More details about the **simulations** and BioDynaMo's **modeling features**

https://doi.org/10.1093/bioinformatics/btab649



Bioinformatics, 38(2), 2022, 453–460
doi: 10.1093/bioinformatics/btab649
Advance Access Publication Date: 16 September 2021
Original Paper

OXFORD

Systems biology

BioDynaMo: a modular platform for high-performance agent-based simulation

Lukas Breitwieser [1,2,*], Ahmad Hesam[1,3,*], Jean de Montigny[1], Vasileios Vavourakis[4,5], Alexandros Iosif[4], Jack Jennings[6], Marcus Kaiser[6,7,8], Marco Manca [9], Alberto Di Meglio[1], Zaid Al-Ars[3], Fons Rademakers[1], Onur Mutlu[2,10,*] and Roman Bauer[11,*]

[1]CERN openlab, IT Department, CERN, Geneva 1211, Switzerland, [2]Department of Computer Science, ETH Zurich, Zurich 8092, Switzerland, [3]Department of Quantum & Computer Engineering, Delft University of Technology, Delft 2628CD, The Netherlands, [4]Department of Mechanical & Manufacturing Engineering, University of Cyprus, Nicosia 2109, Cyprus, [5]Department of Medical Physics & Biomedical Engineering, University College London, London WC1E 6BT, UK, [6]School of Computing, Newcastle University, Newcastle upon Tyne NE4 5TG, UK, [7]Department of Functional Neurosurgery, Ruijin Hospital, Shanghai Jiao Tong University School of Medicine, Shanghai 200025, China, [8]Precision Imaging Beacon, School of Medicine, University of Nottingham, Nottingham NG7 2UH, UK, [9]SCimPulse Foundation, Geleen 6162 BC, The Netherlands, [10]Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich 8092, Switzerland and [11]Department of Computer Science, University of Surrey, Guildford GU2 7XH, UK

*To whom correspondence should be addressed.

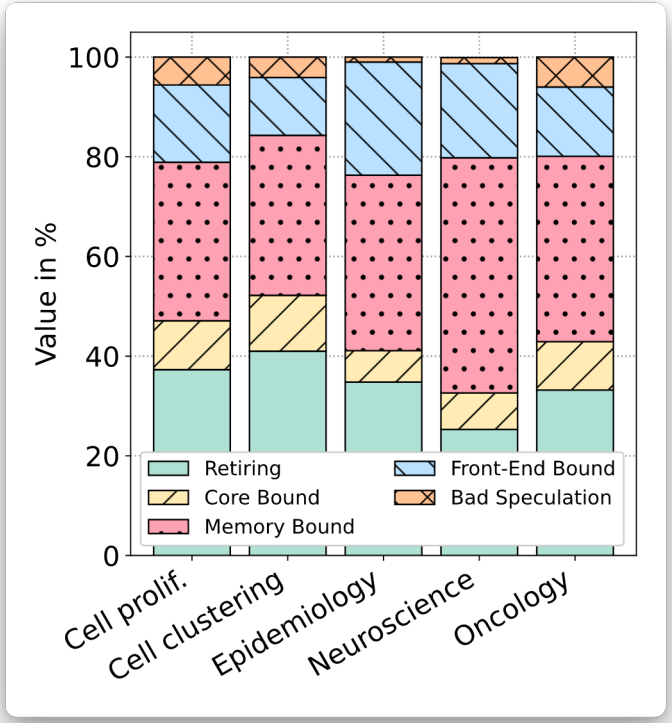
Associate Editor: Jonathan Wren

Received on February 4, 2021; revised on September 2, 2021; editorial decision on September 3, 2021; accepted on September 13, 2021

Optimizations

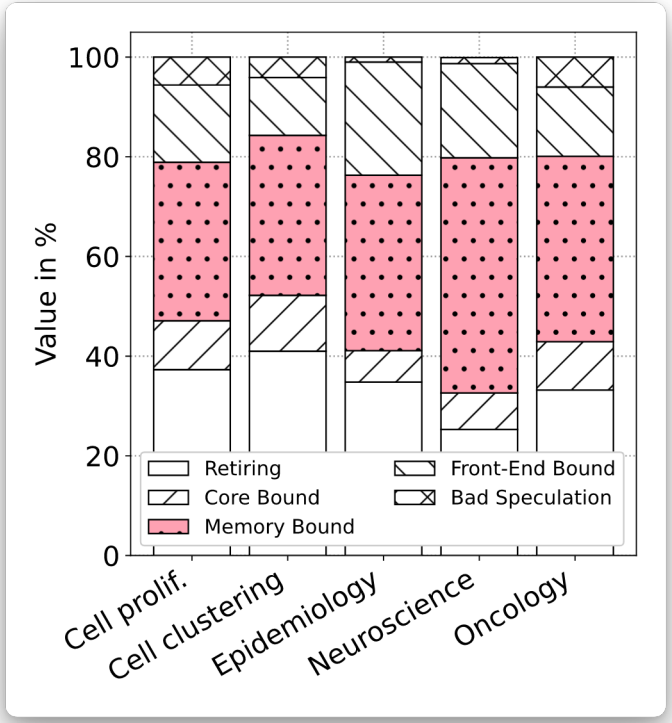
Key observation:
Agent-based simulations are often **memory-bound**

Microarchitecture analysis for five simulations



Key observation:
Agent-based simulations are often **memory-bound**

Microarchitecture analysis for five simulations



Key challenges and improvements

Improved fundamental operations:

- Determine neighbors
- Add and remove agents
- Iterate over agents and execute a function
- Iterate over neighbors and execute a function

Key challenges and improvements

I: Maximize parallelization

- Optimize radial neighbor search
- Parallelize the addition and removal of agents

Improved fundamental operations:

- Determine neighbors
- Add and remove agents
- Iterate over agents and execute a function
- Iterate over neighbors and execute a function

Key challenges and improvements

I: Maximize parallelization

- Optimize radial neighbor search
- Parallelize the addition and removal of agents

II: Memory layout

- NUMA-aware iteration
- Agent sorting and balancing
- Pool-based memory allocator

Improved fundamental operations:

- Determine neighbors
- Add and remove agents
- Iterate over agents and execute a function
- Iterate over neighbors and execute a function

Key challenges and improvements

I: Maximize parallelization

- Optimize radial neighbor search
- Parallelize the addition and removal of agents

II: Memory layout

- NUMA-aware iteration
- Agent sorting and balancing
- Pool-based memory allocator

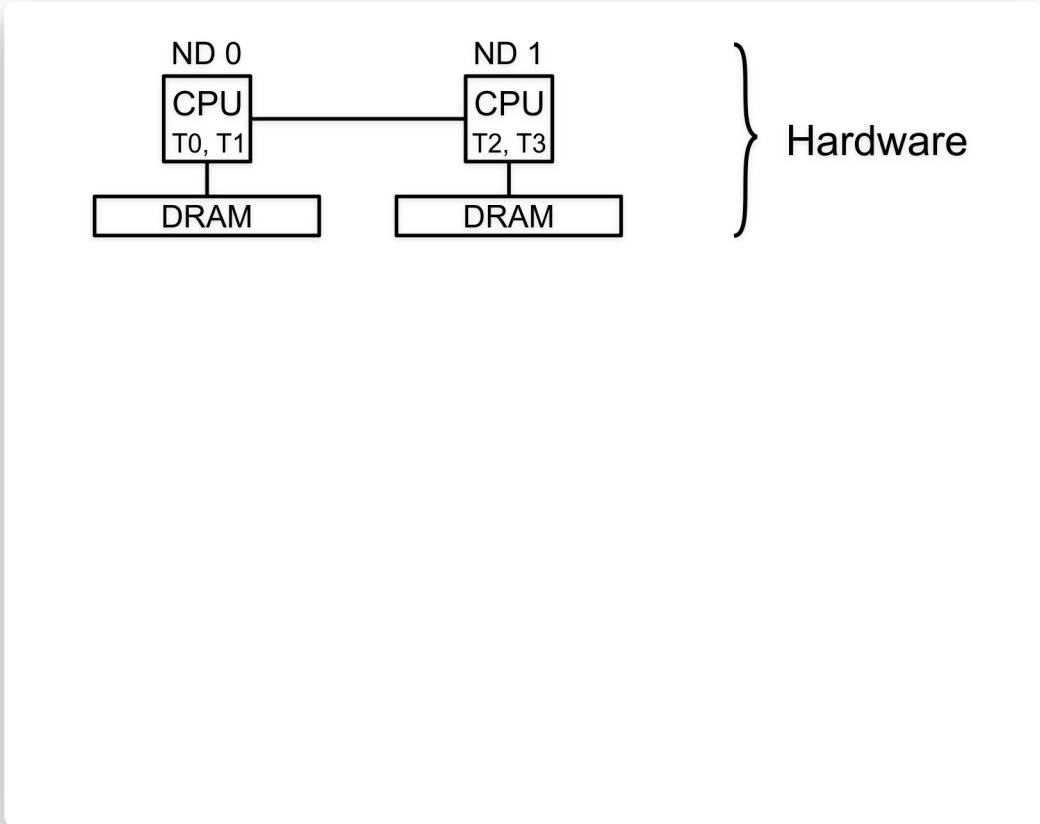
III: Avoid unnecessary work

- Pair-wise force calculation for static regions

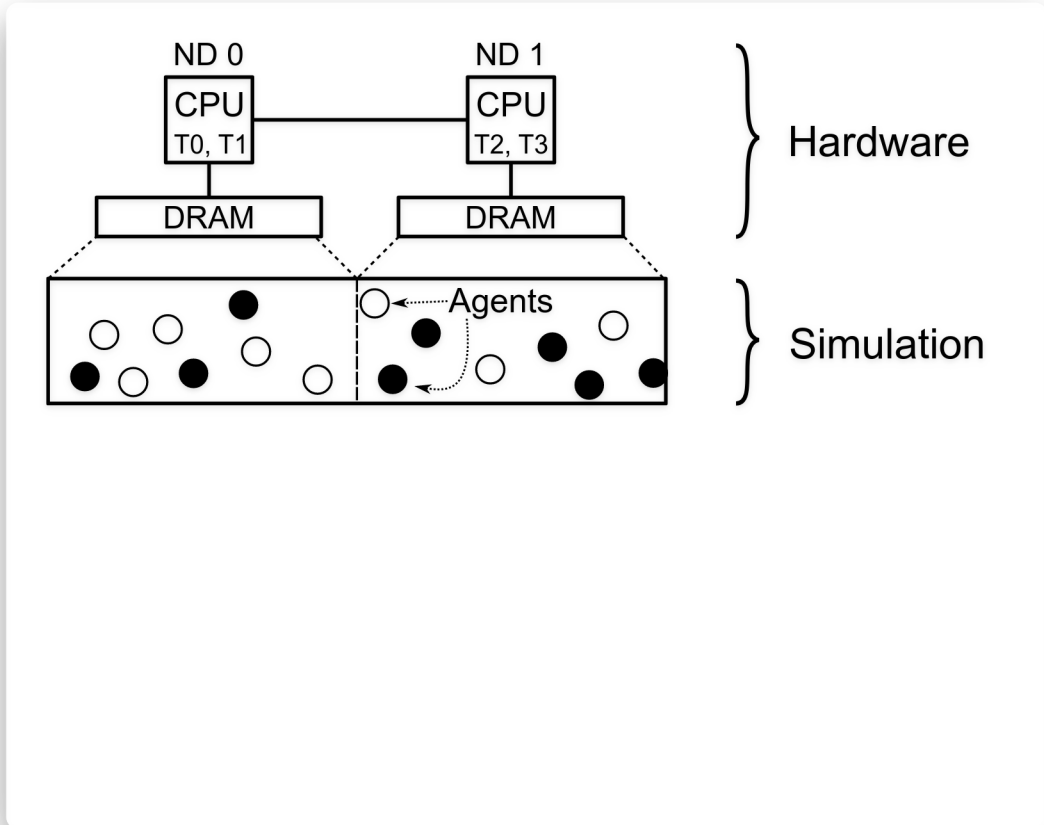
Improved fundamental operations:

- Determine neighbors
- Add and remove agents
- Iterate over agents and execute a function
- Iterate over neighbors and execute a function

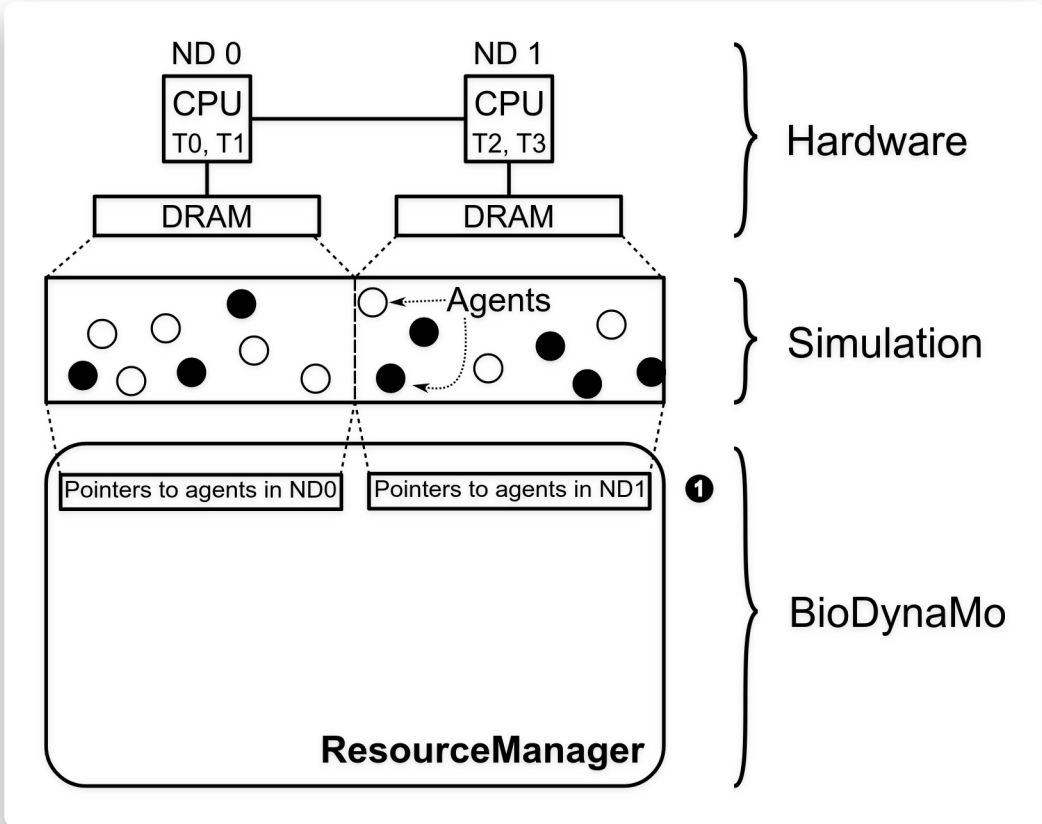
NUMA-aware iteration



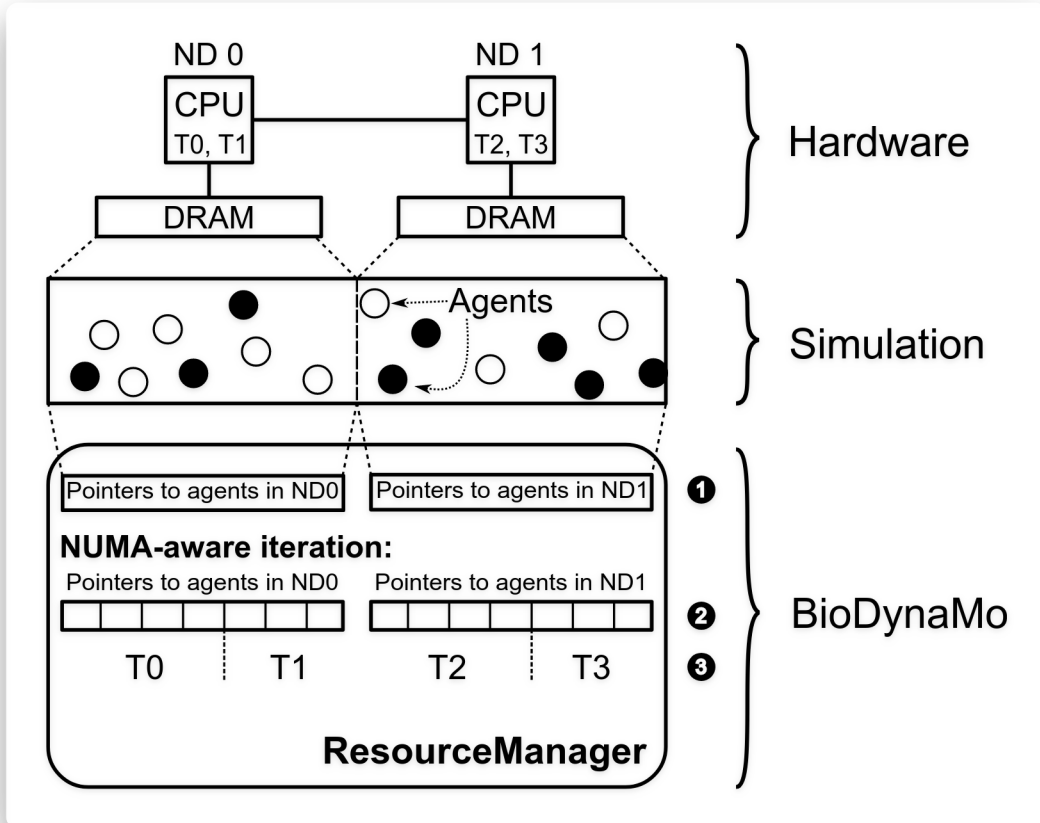
NUMA-aware iteration



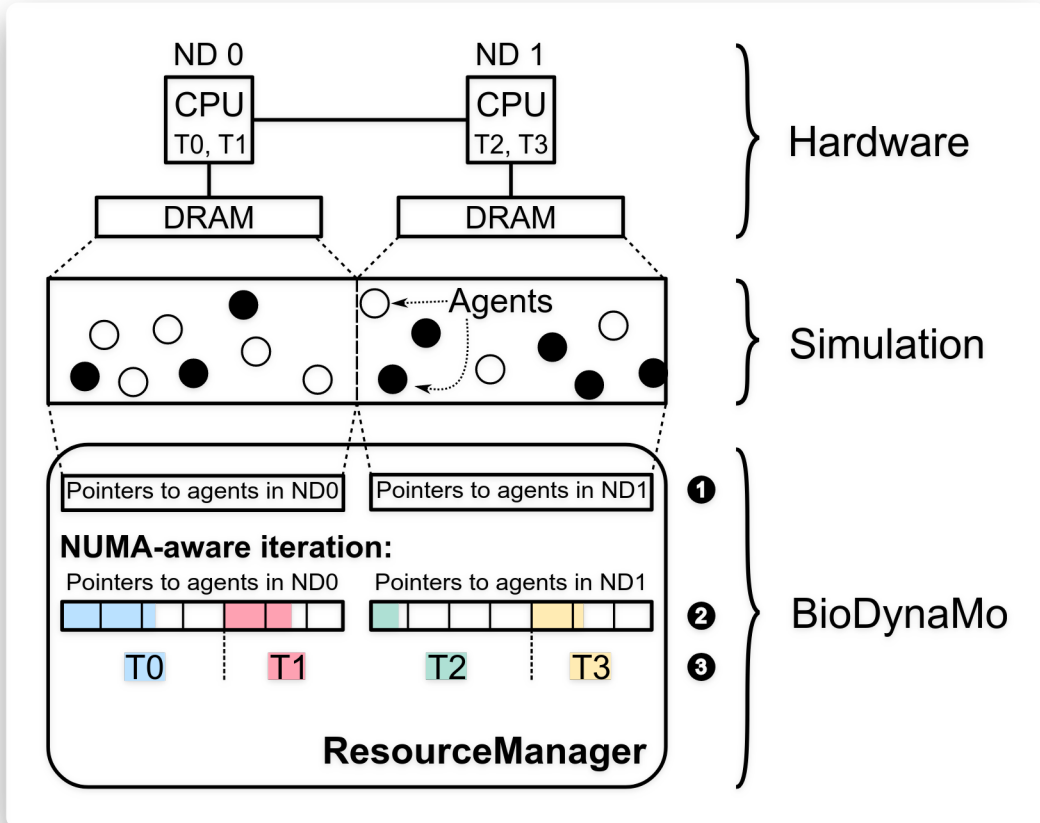
NUMA-aware iteration



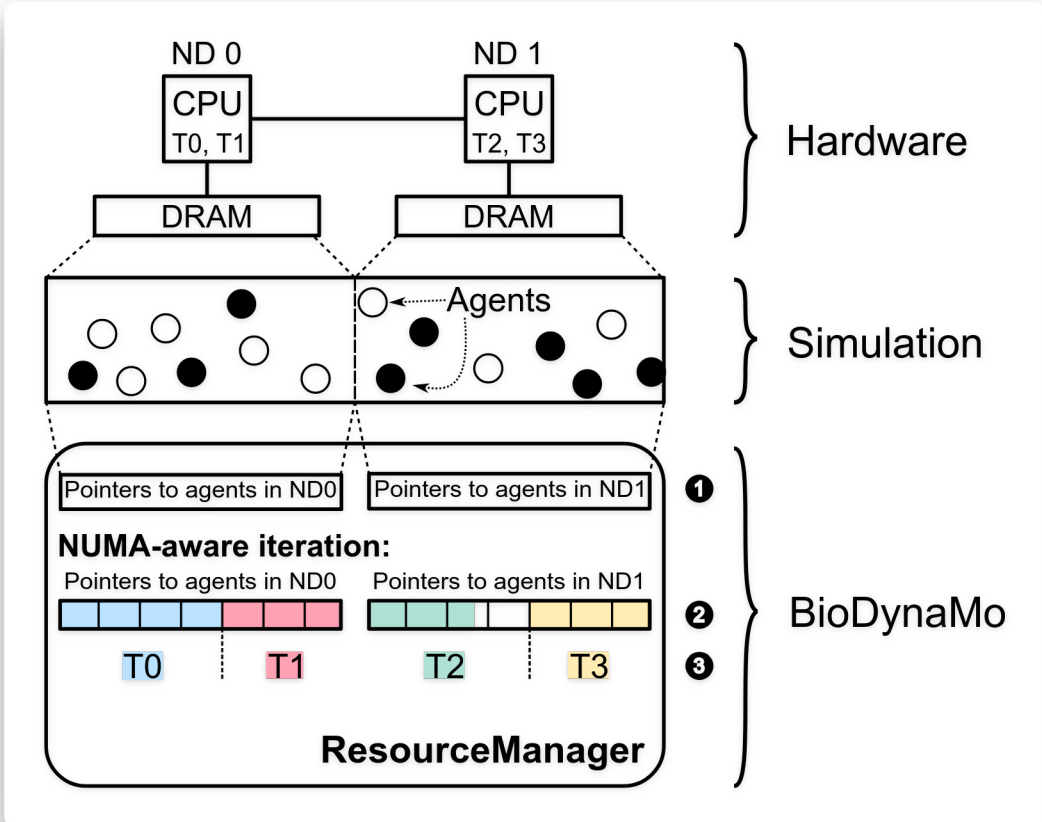
NUMA-aware iteration



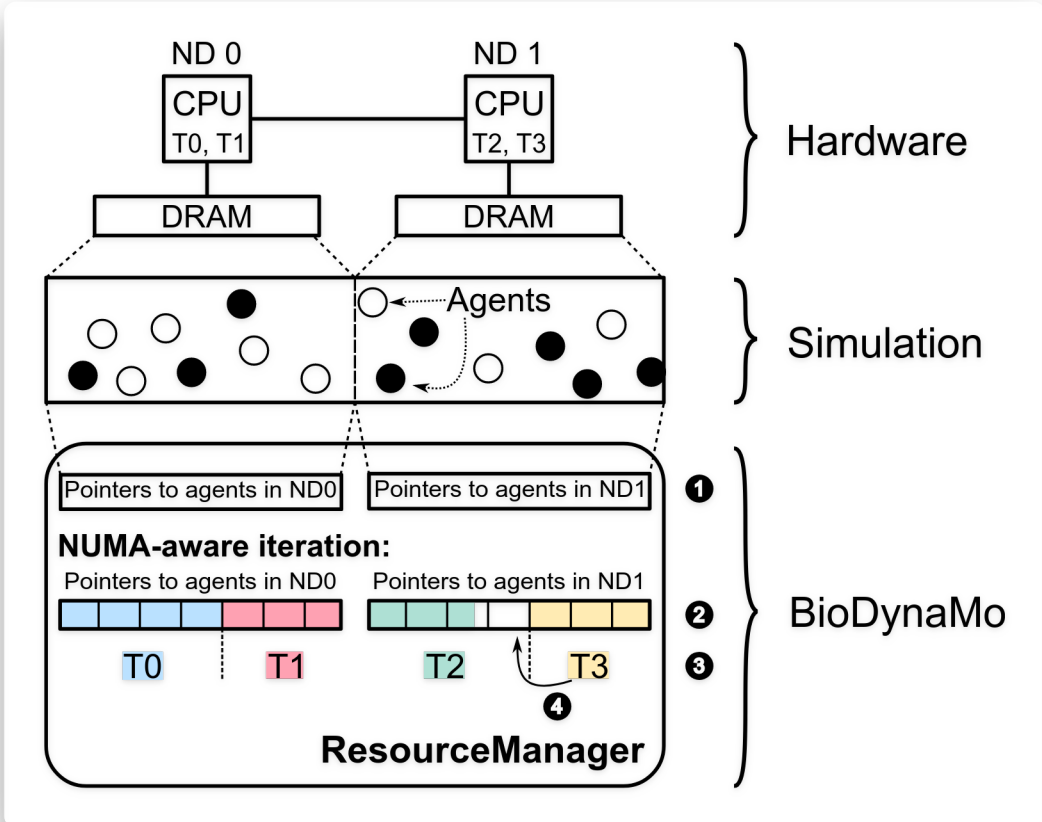
NUMA-aware iteration



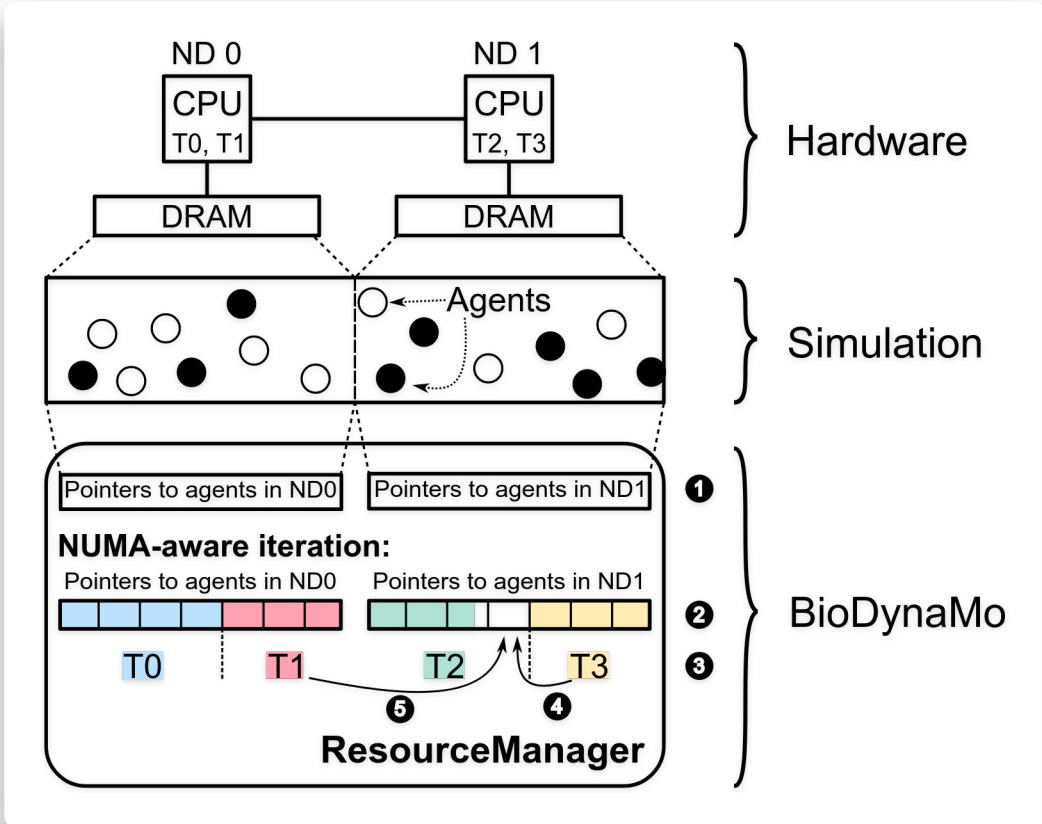
NUMA-aware iteration



NUMA-aware iteration

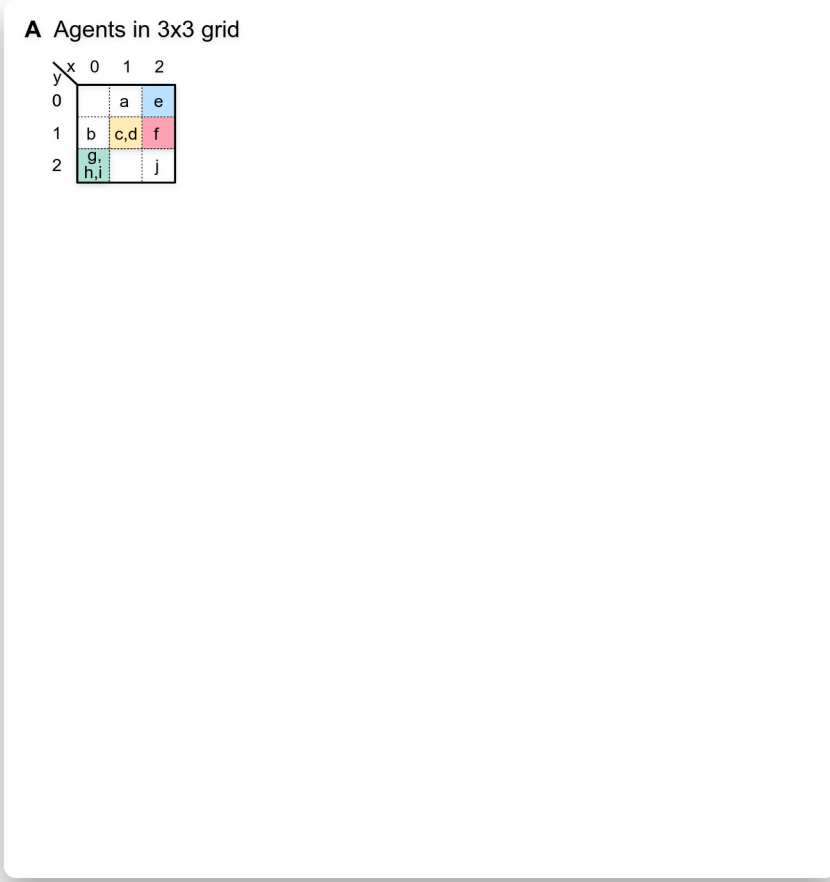


NUMA-aware iteration

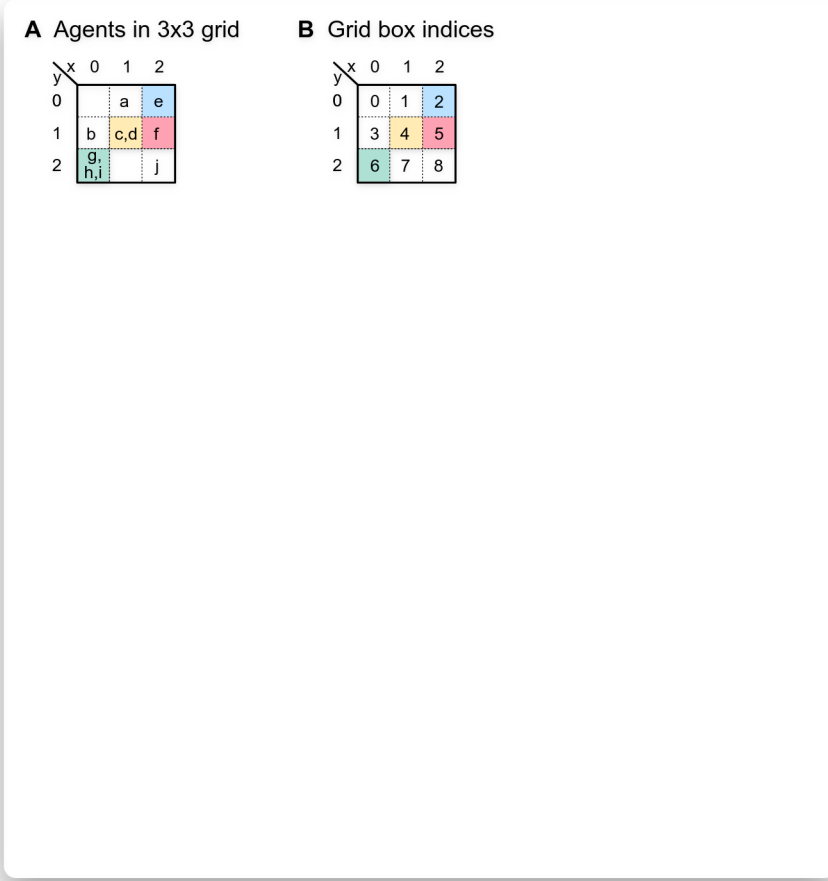


NUMA-aware iteration

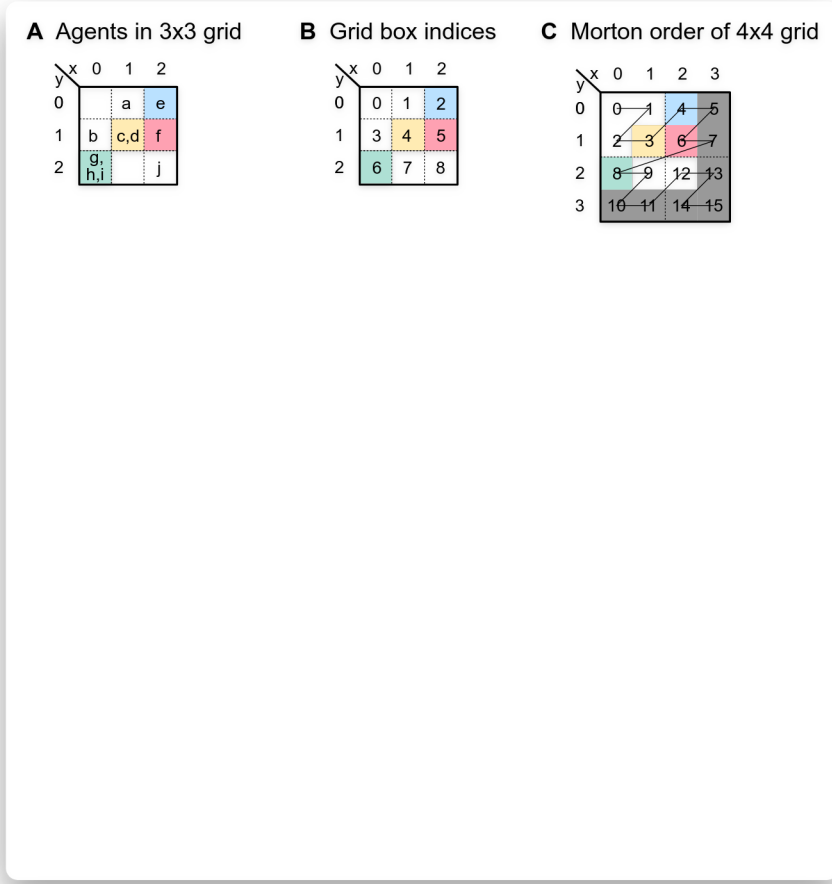
Agent sorting and balancing



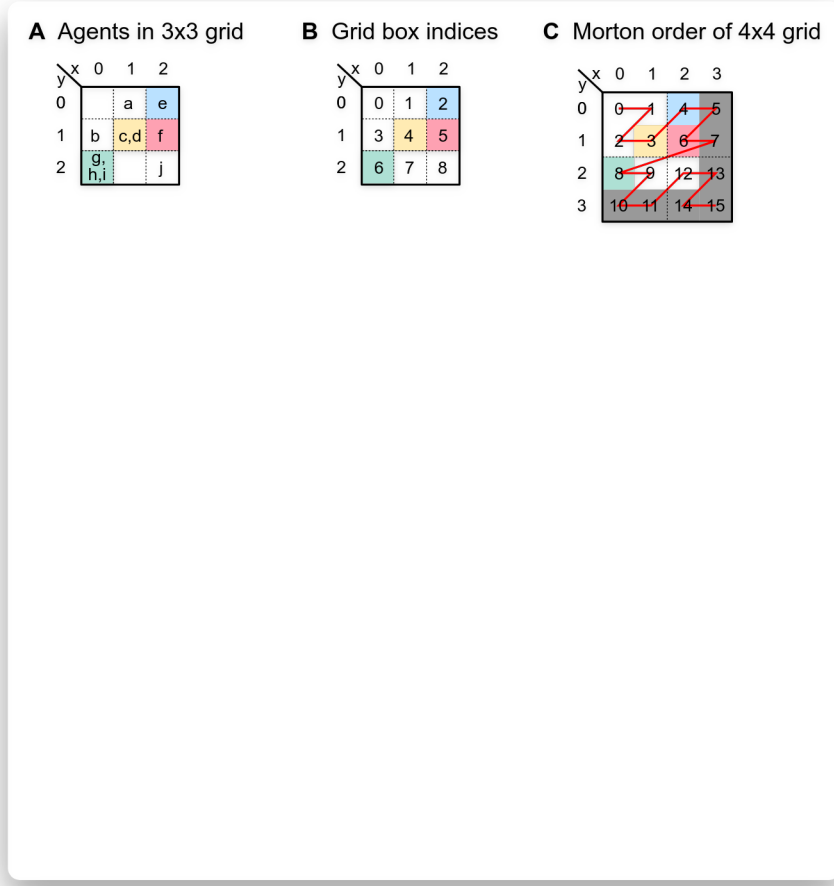
Agent sorting and balancing



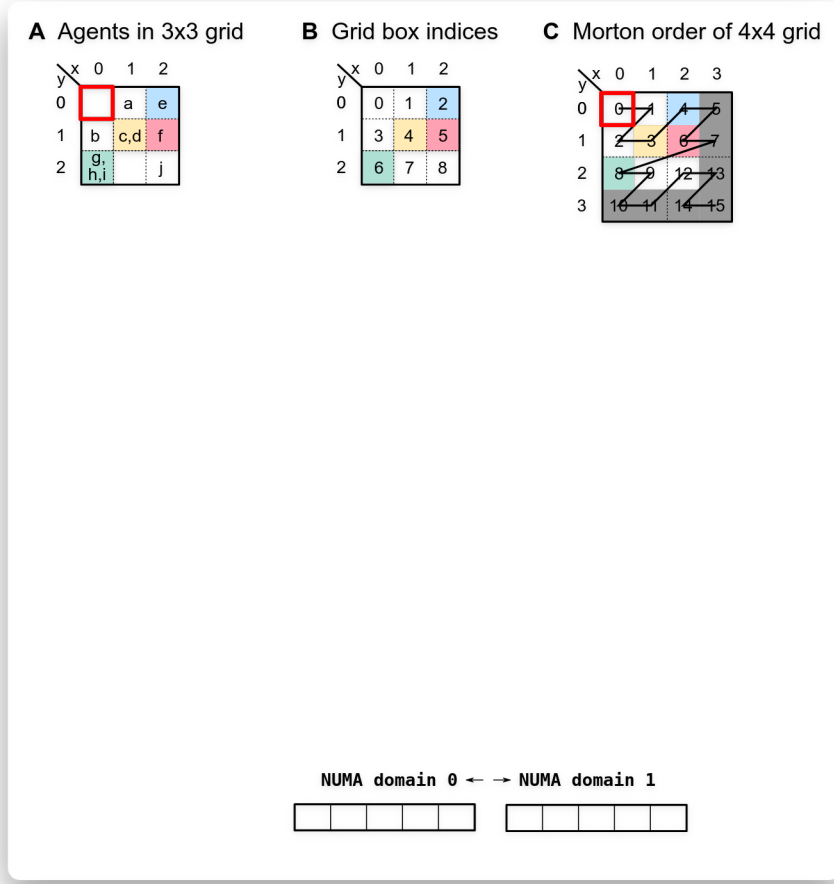
Agent sorting and balancing



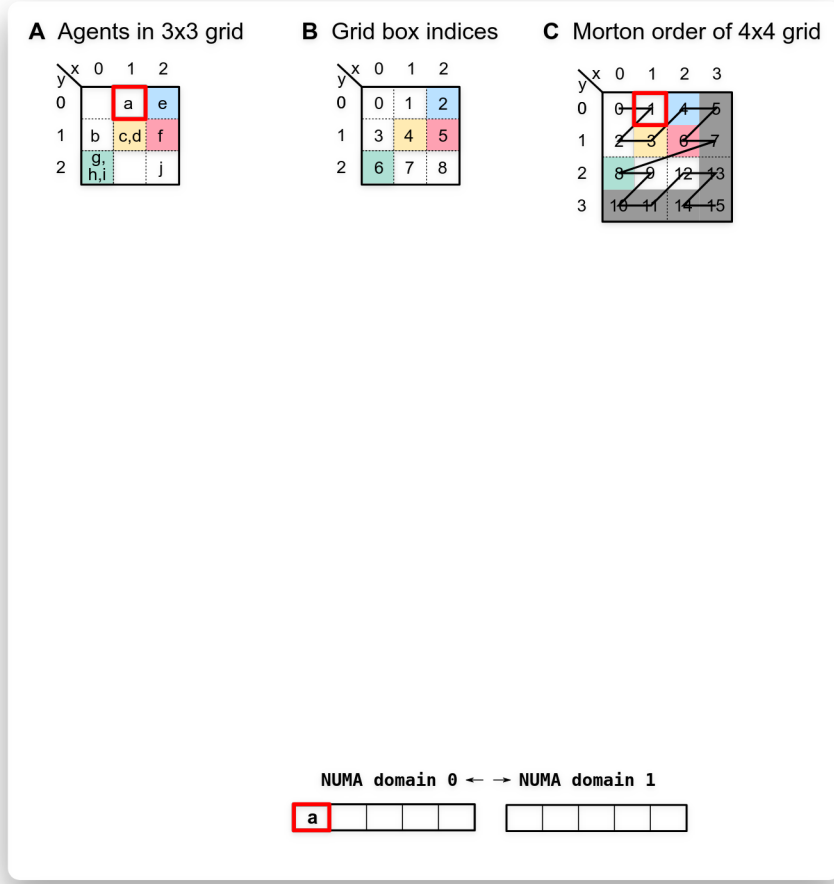
Agent sorting and balancing



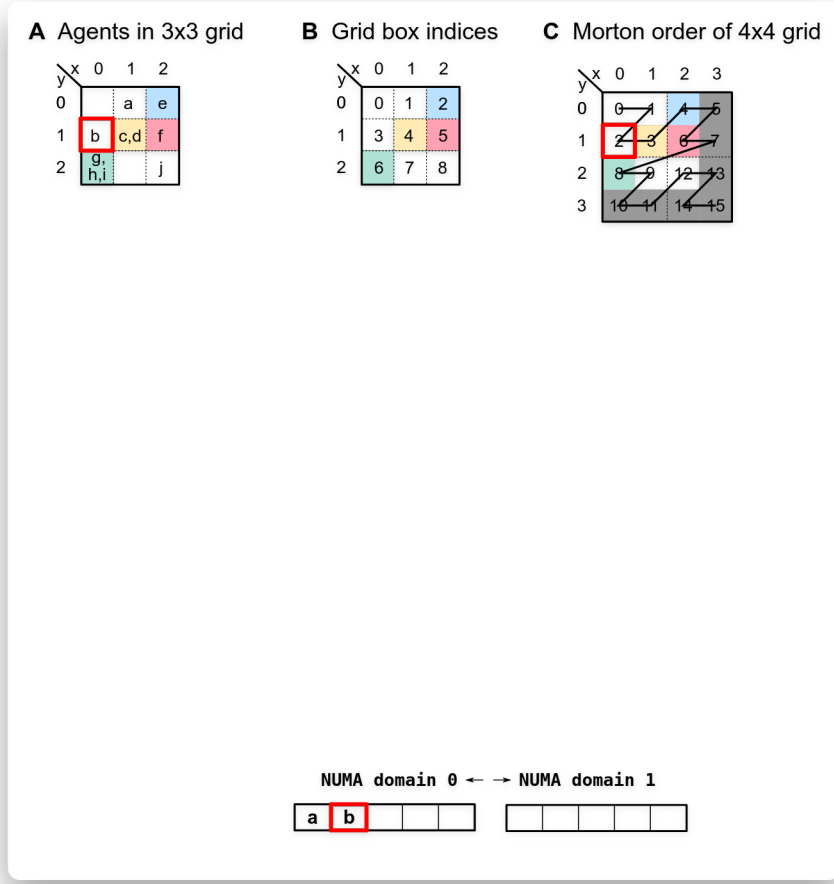
Agent sorting and balancing



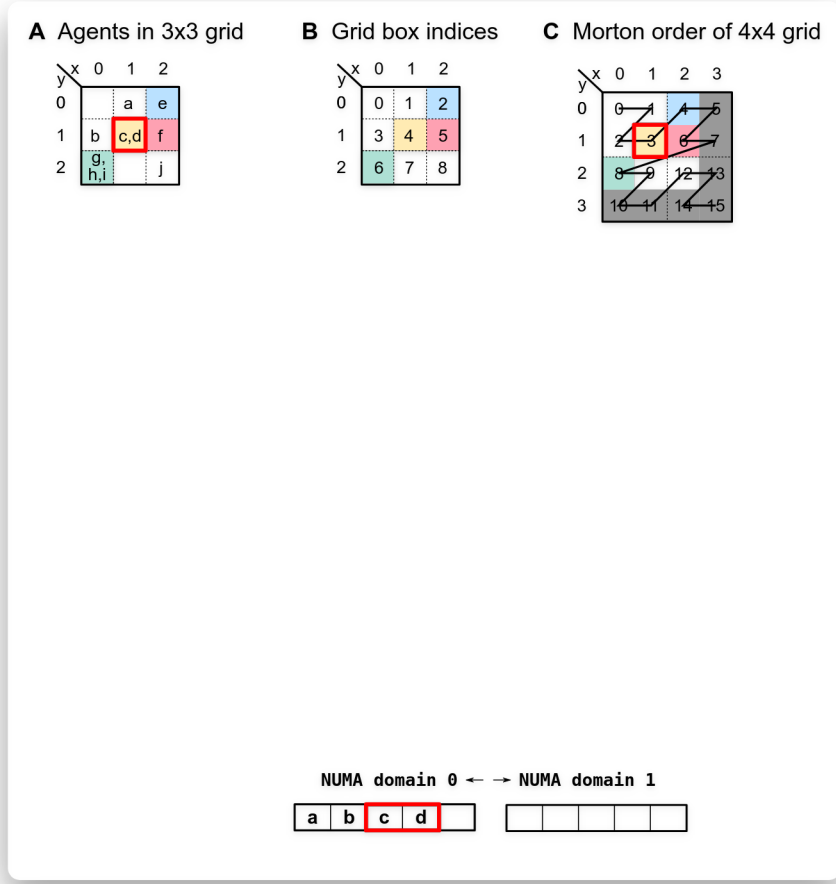
Agent sorting and balancing



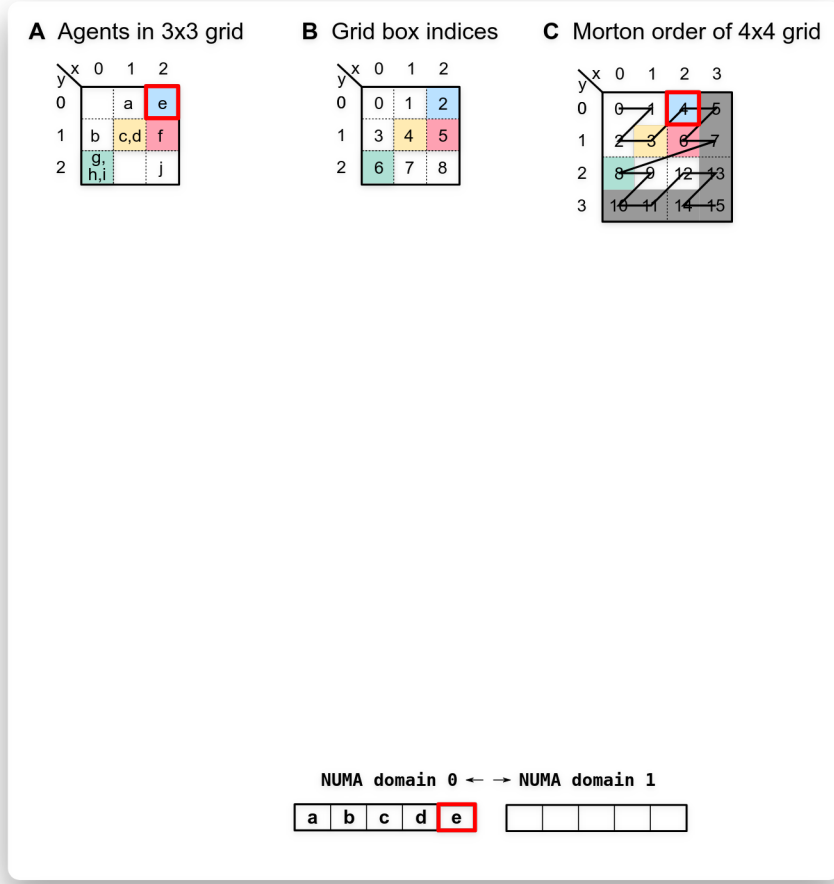
Agent sorting and balancing



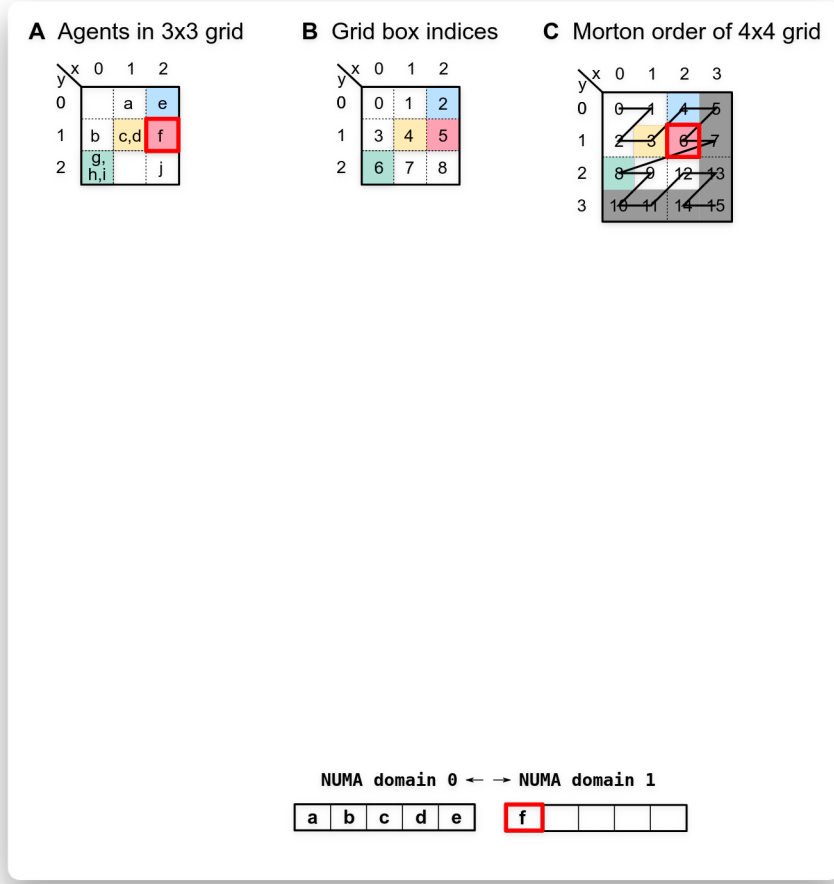
Agent sorting and balancing



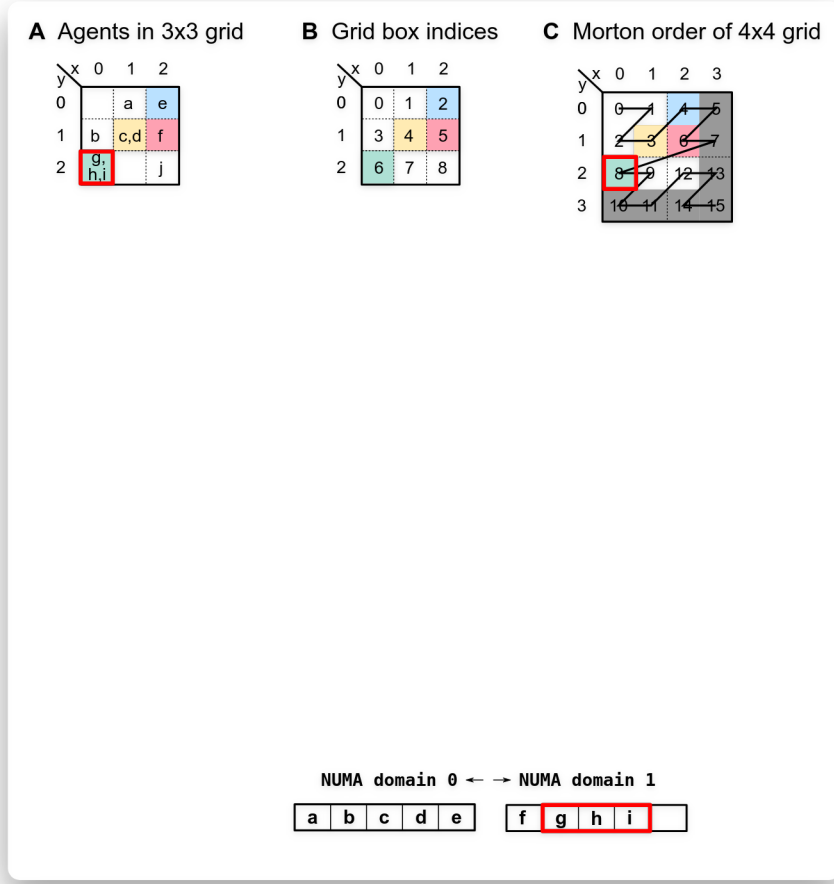
Agent sorting and balancing



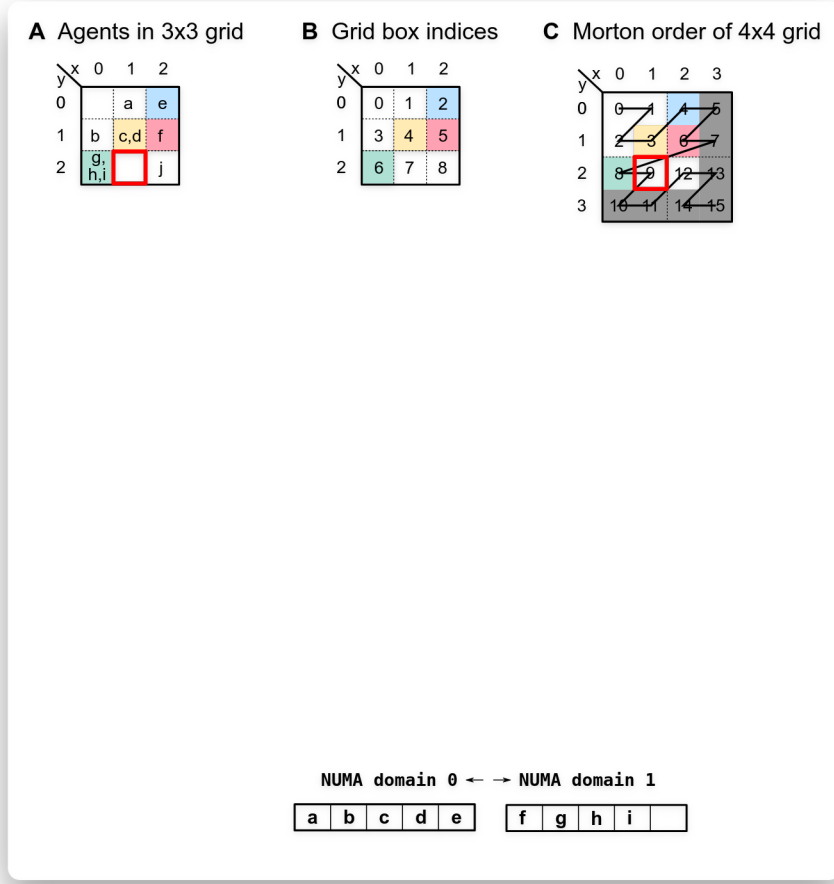
Agent sorting and balancing



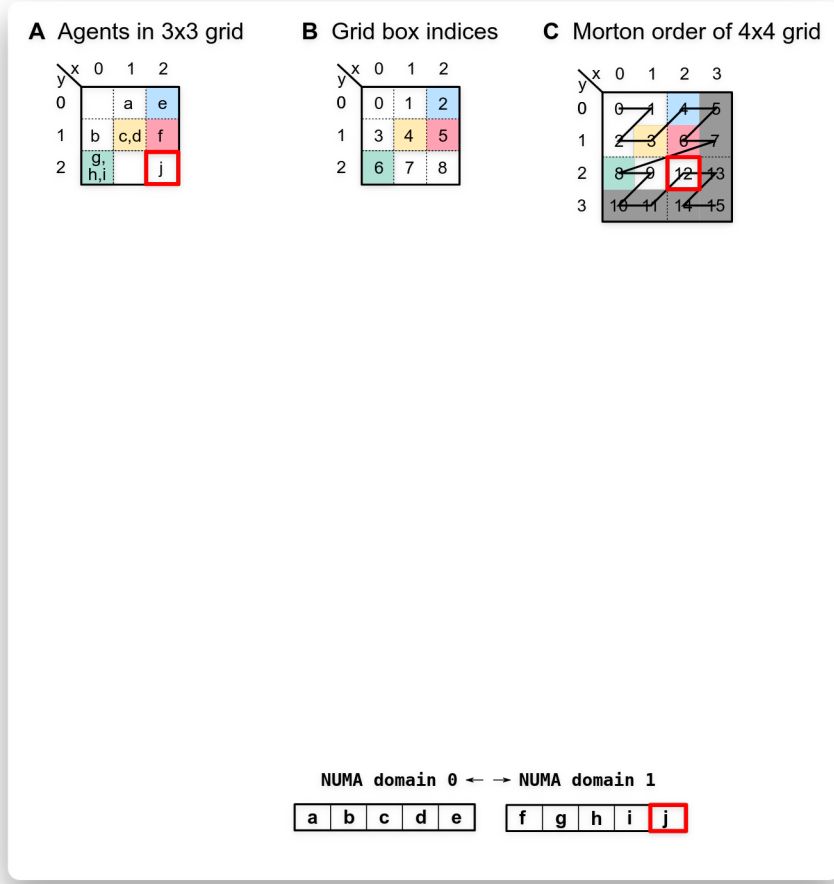
Agent sorting and balancing



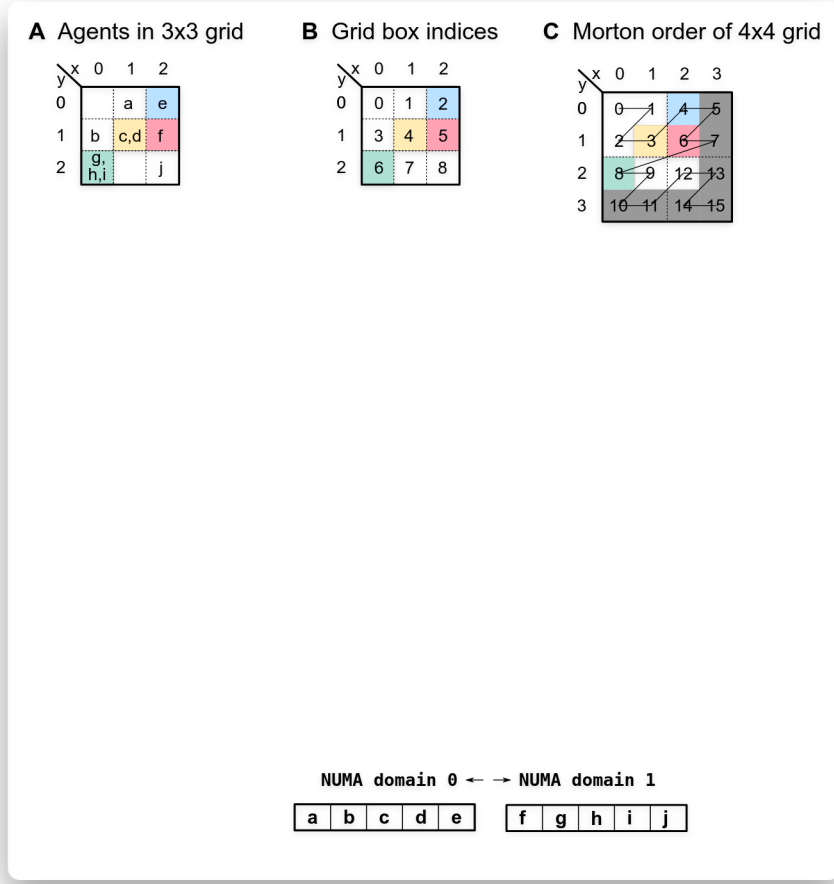
Agent sorting and balancing



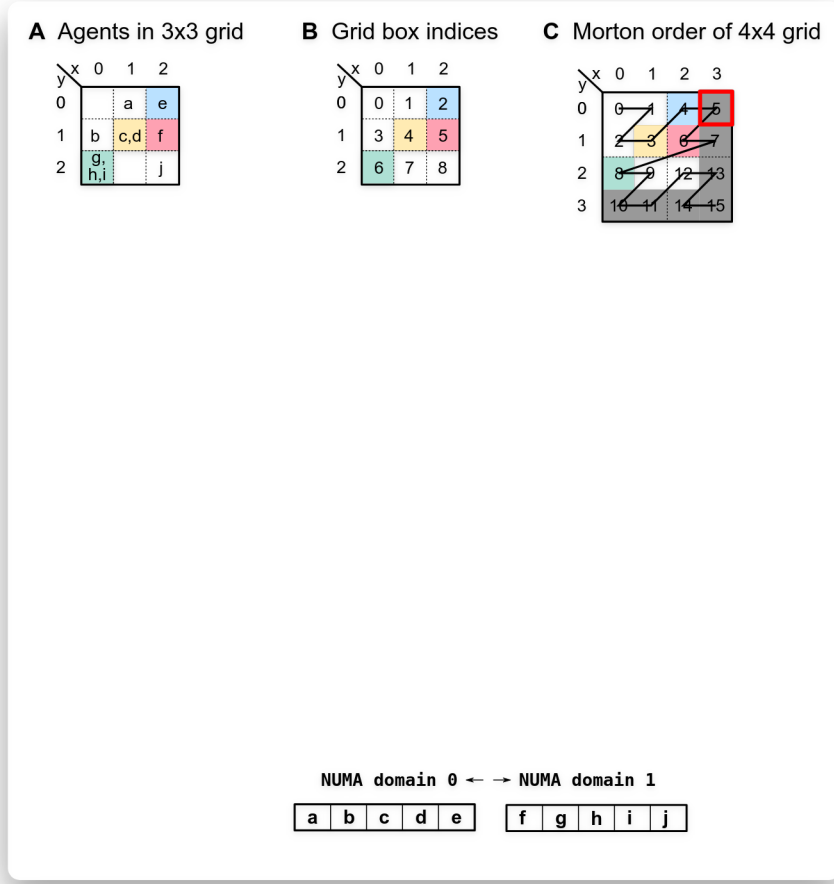
Agent sorting and balancing



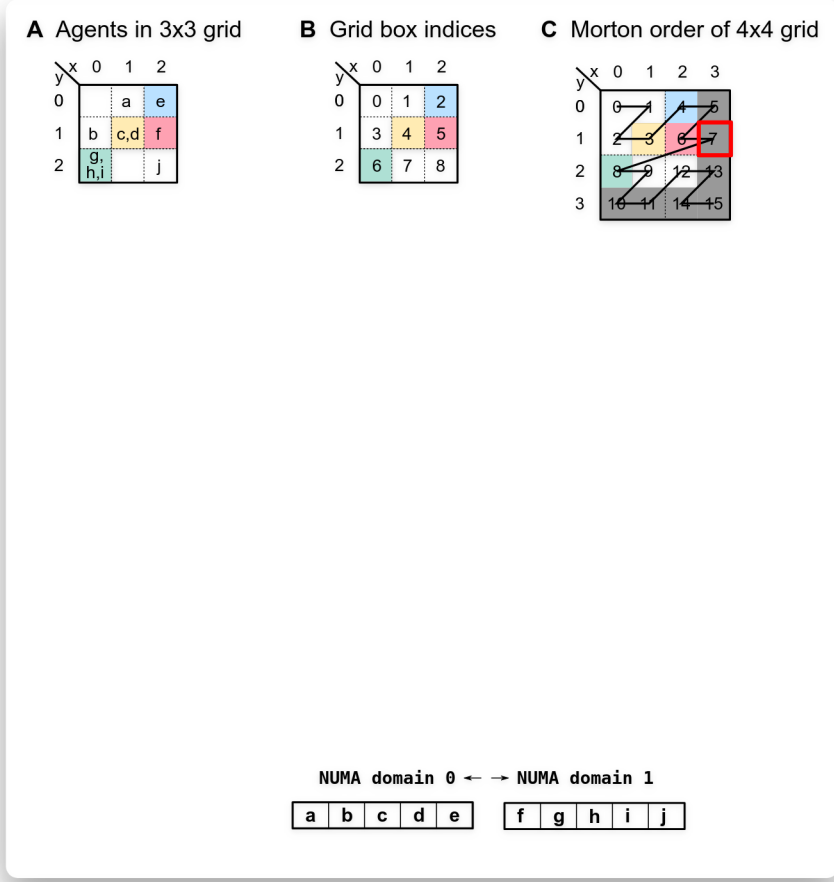
Agent sorting and balancing



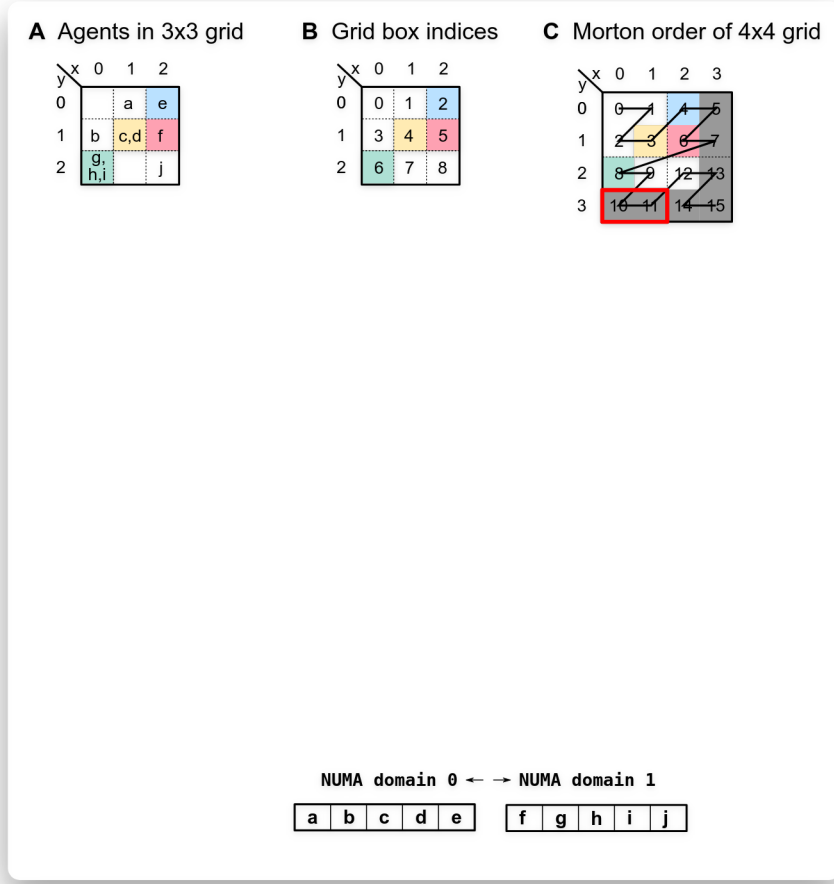
Agent sorting and balancing



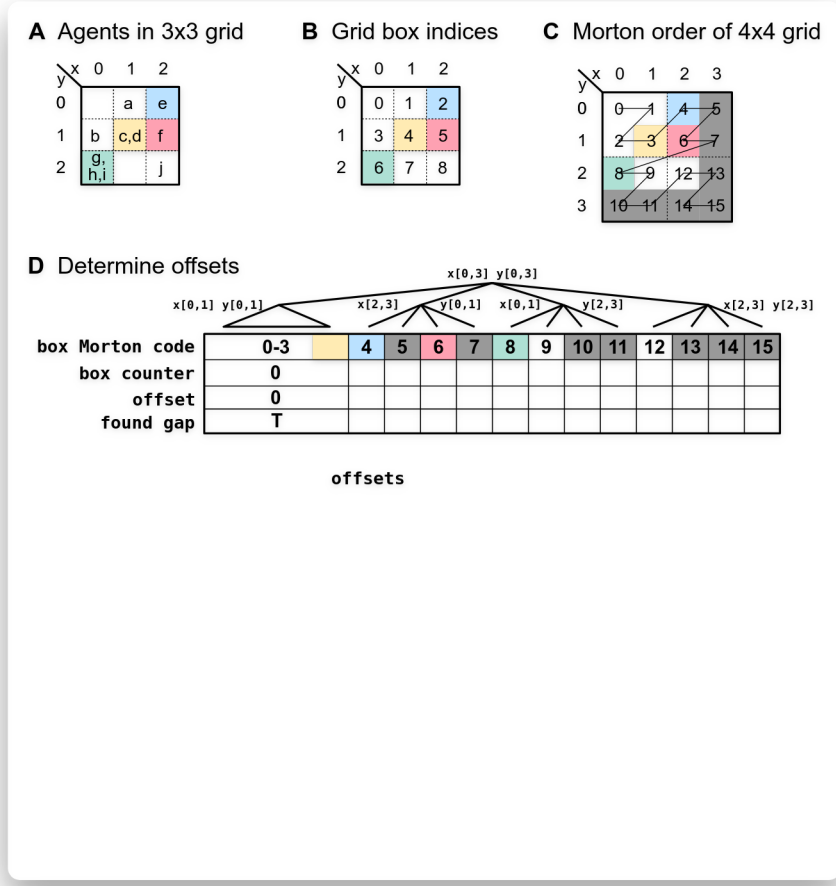
Agent sorting and balancing



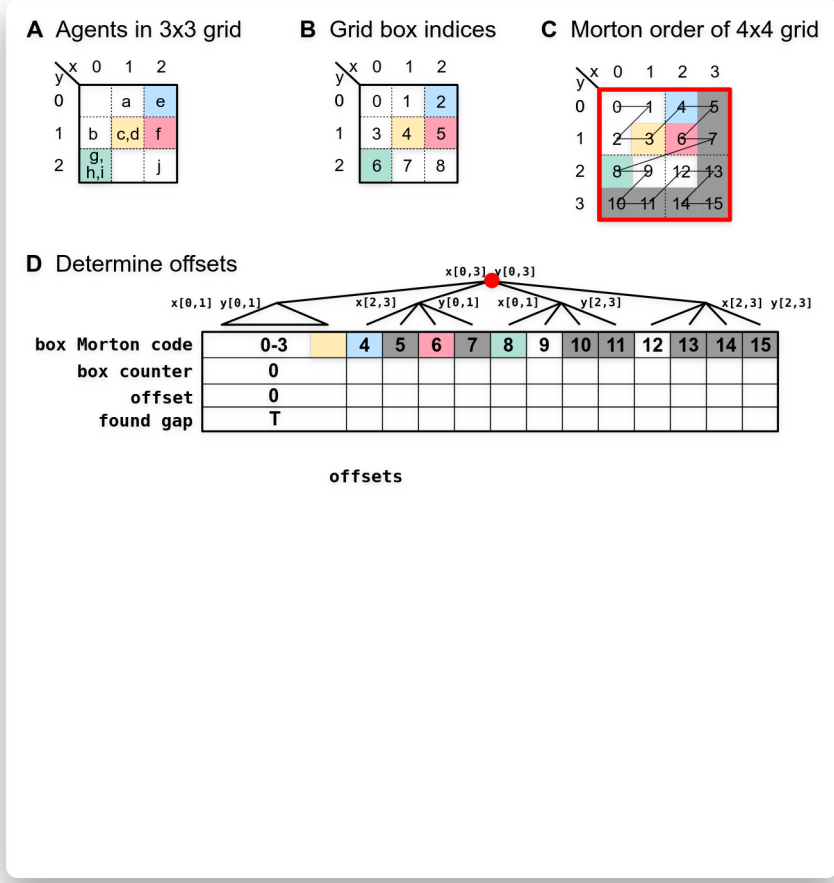
Agent sorting and balancing



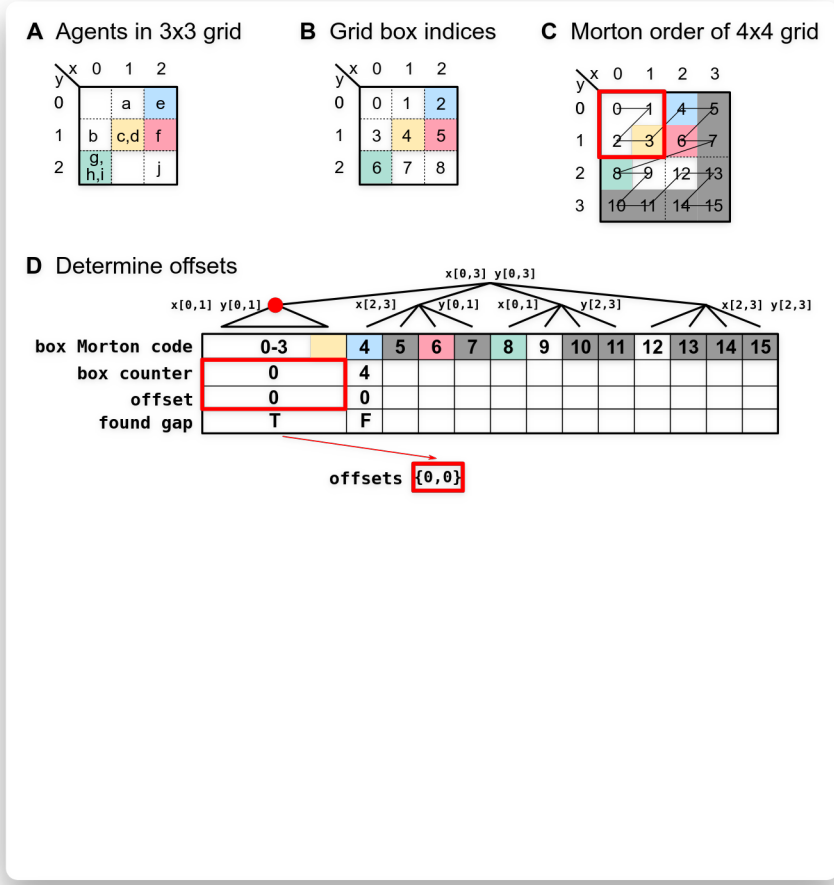
Agent sorting and balancing



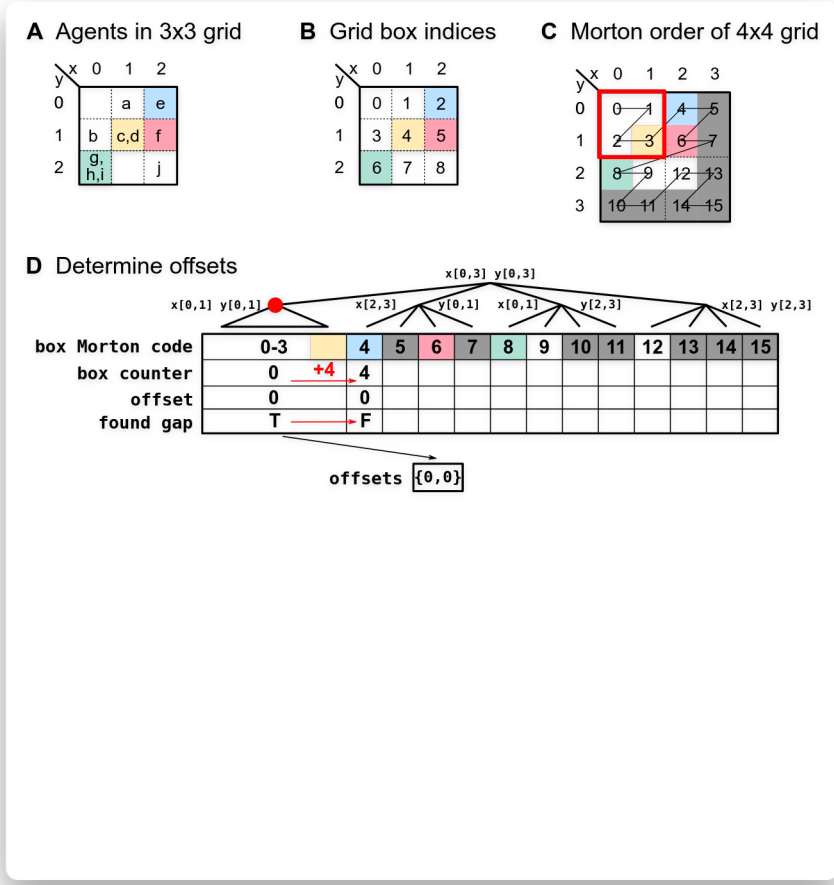
Agent sorting and balancing



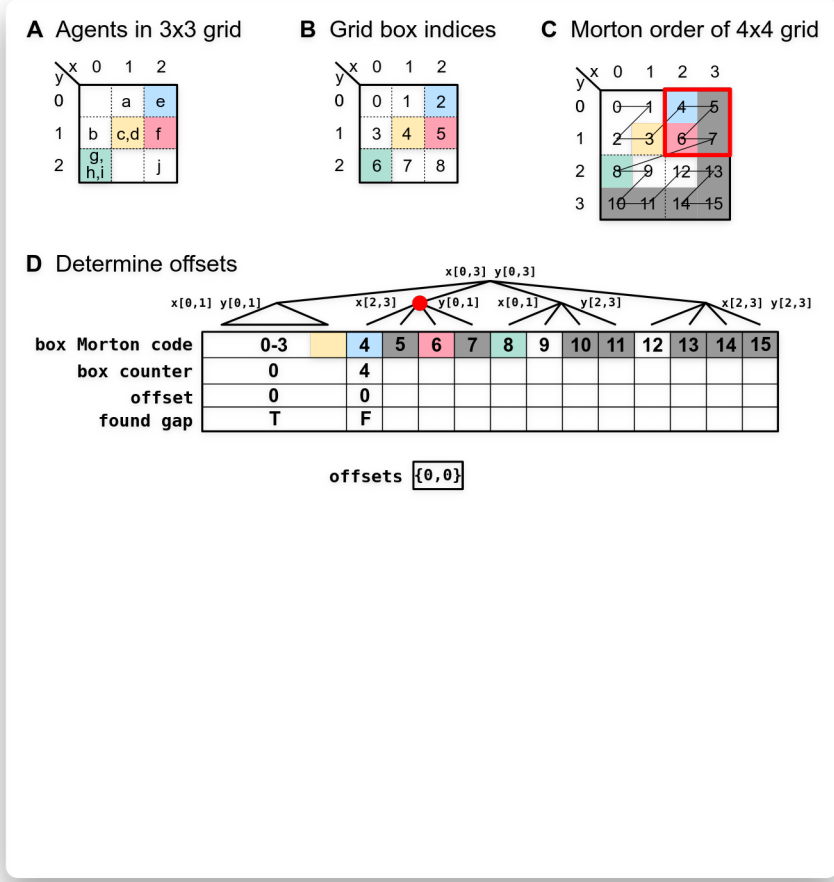
Agent sorting and balancing



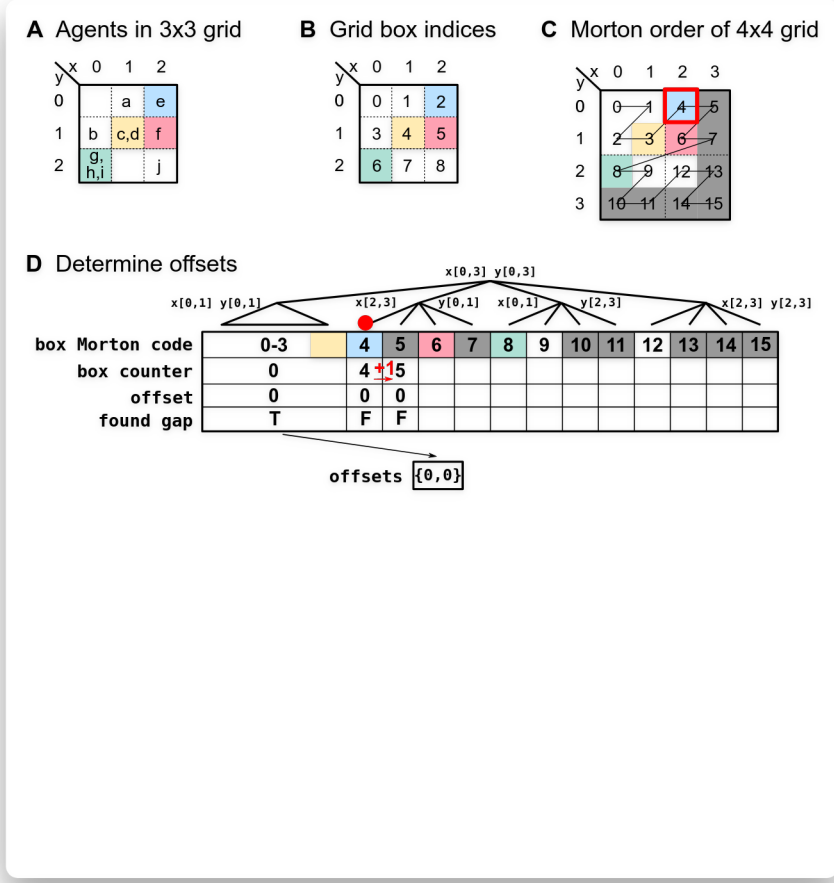
Agent sorting and balancing



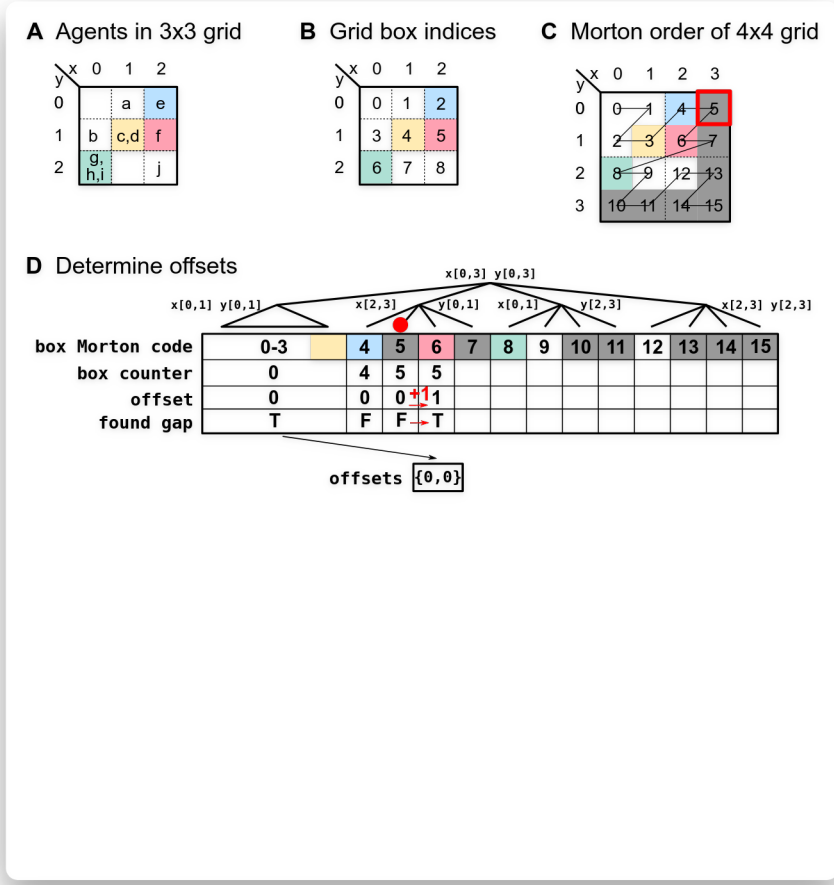
Agent sorting and balancing



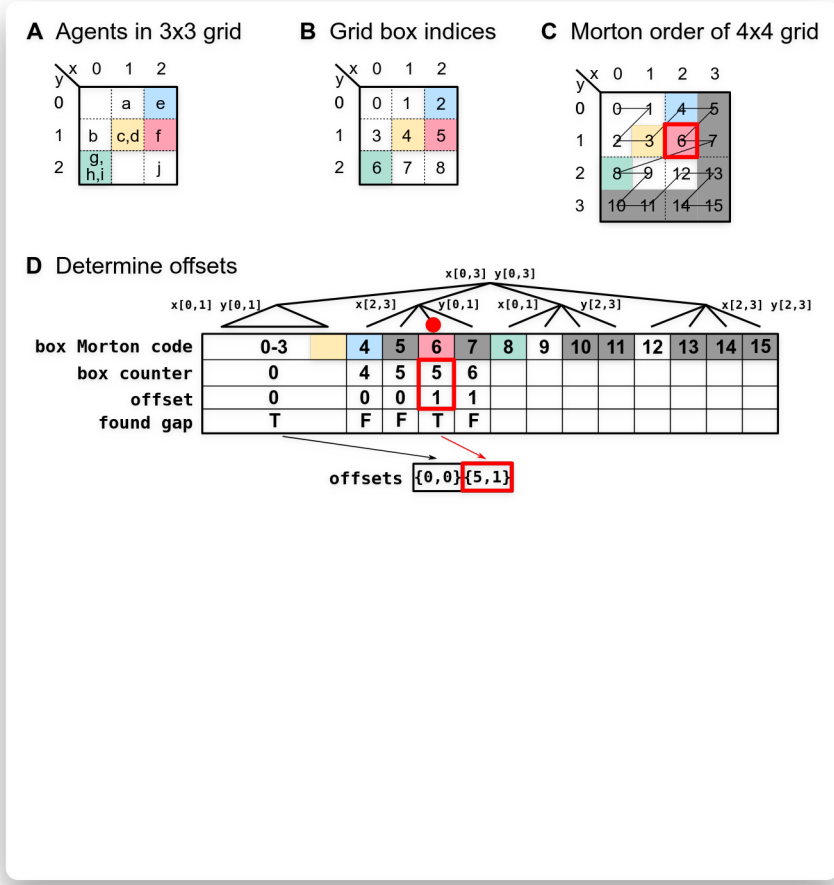
Agent sorting and balancing



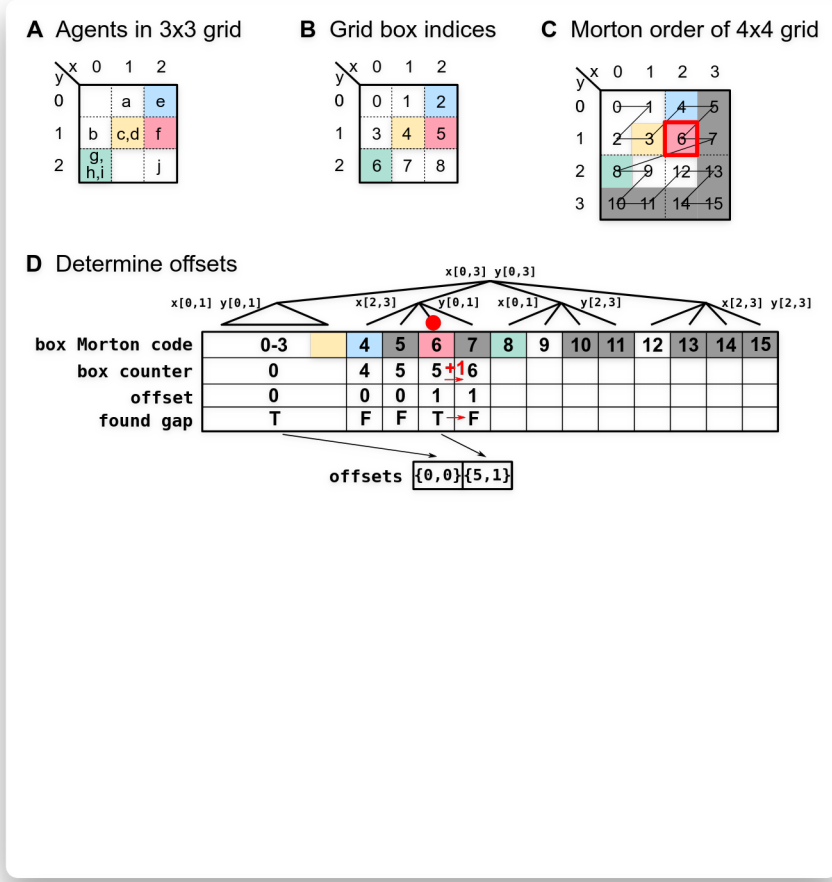
Agent sorting and balancing



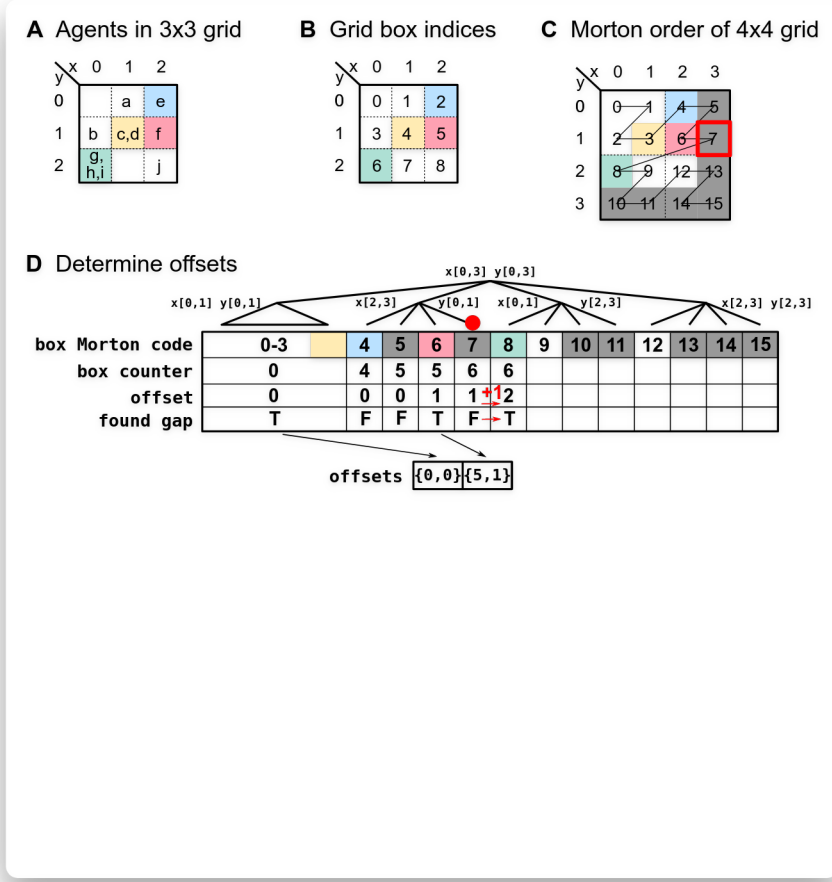
Agent sorting and balancing



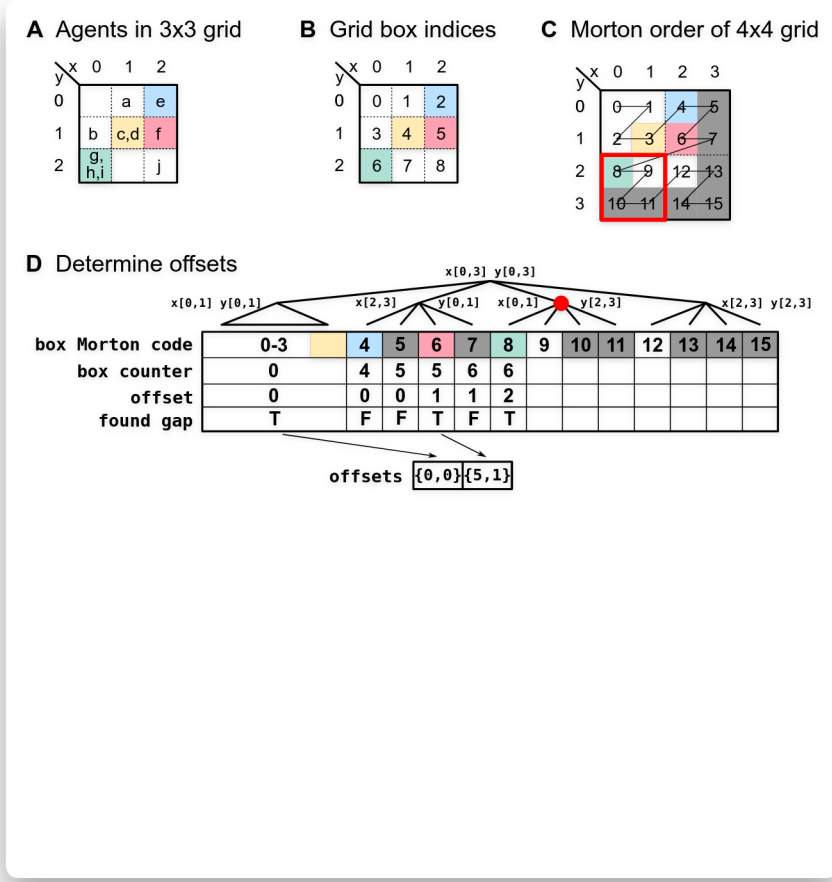
Agent sorting and balancing



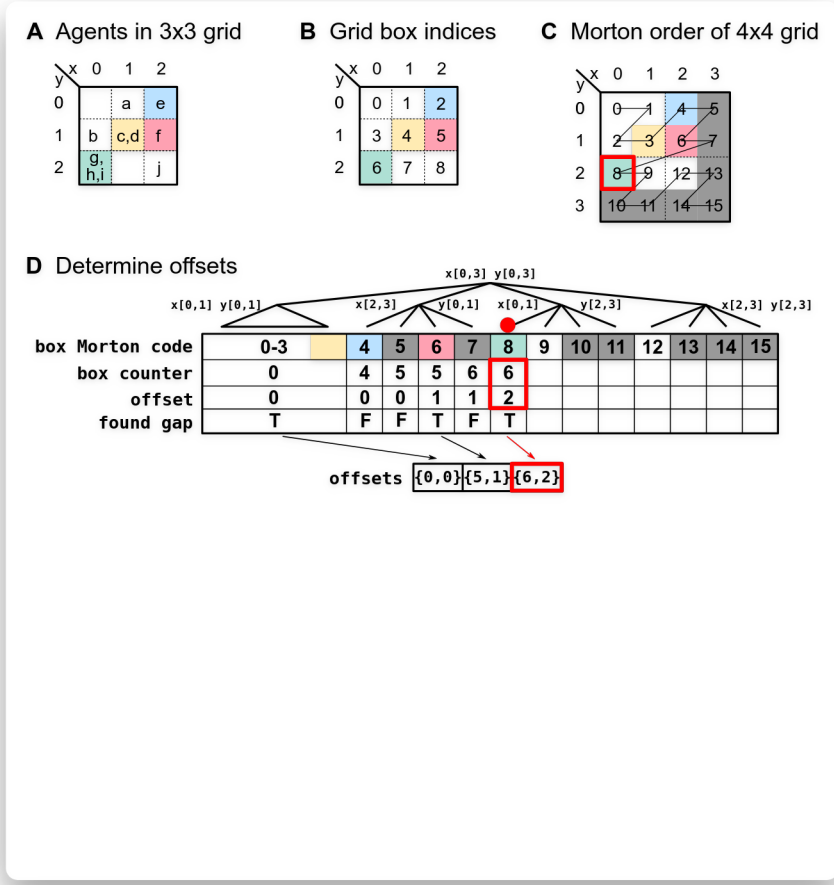
Agent sorting and balancing



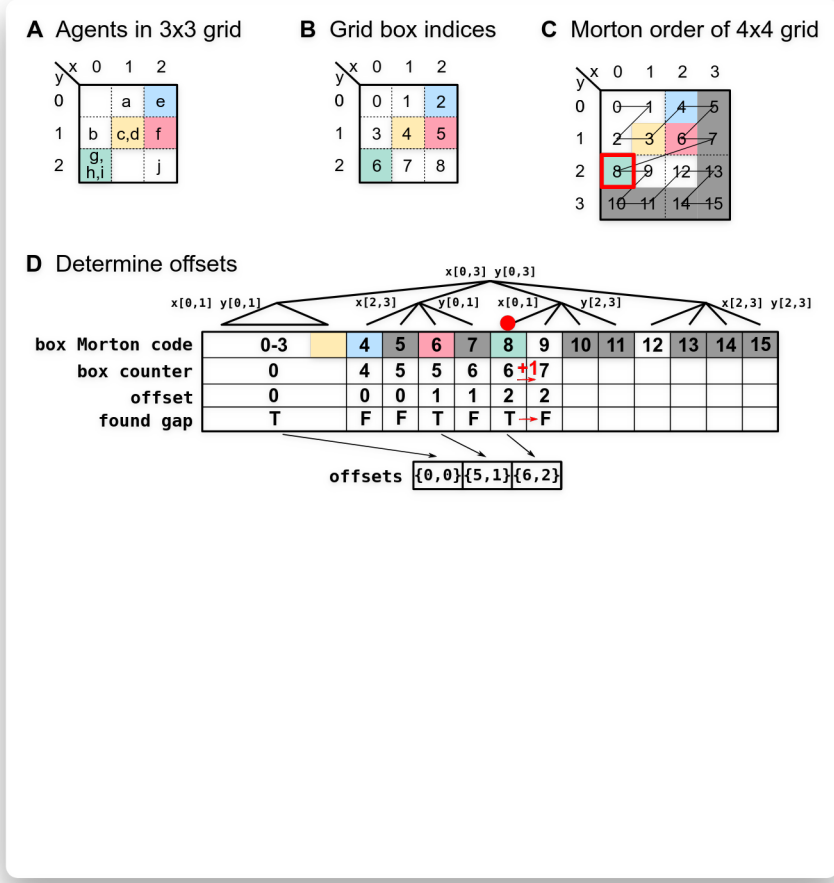
Agent sorting and balancing



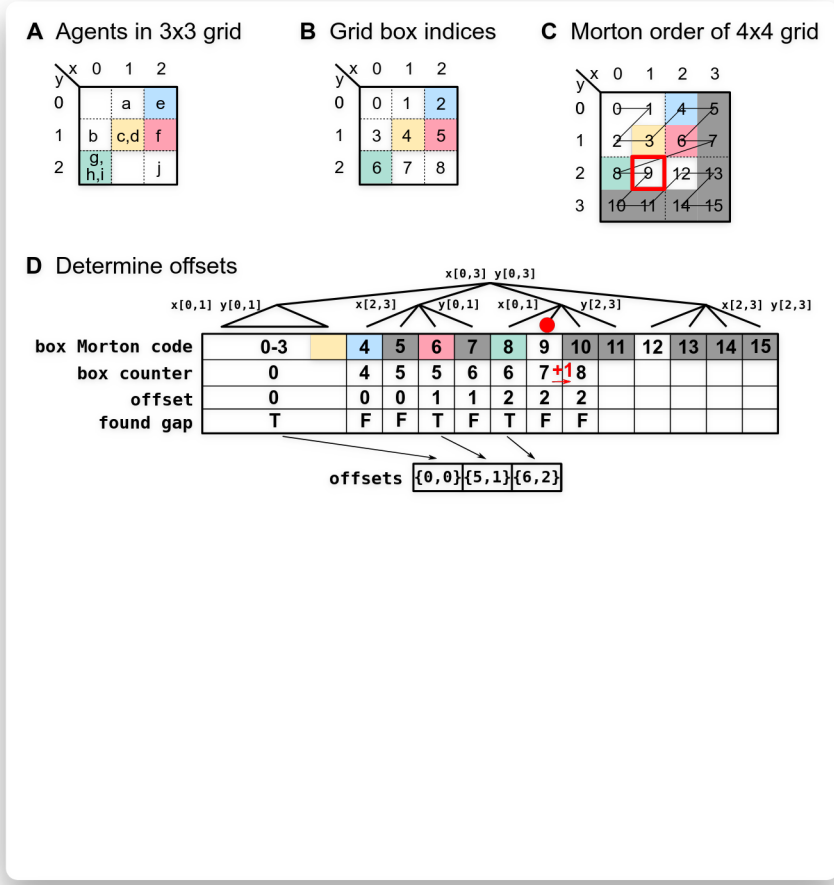
Agent sorting and balancing



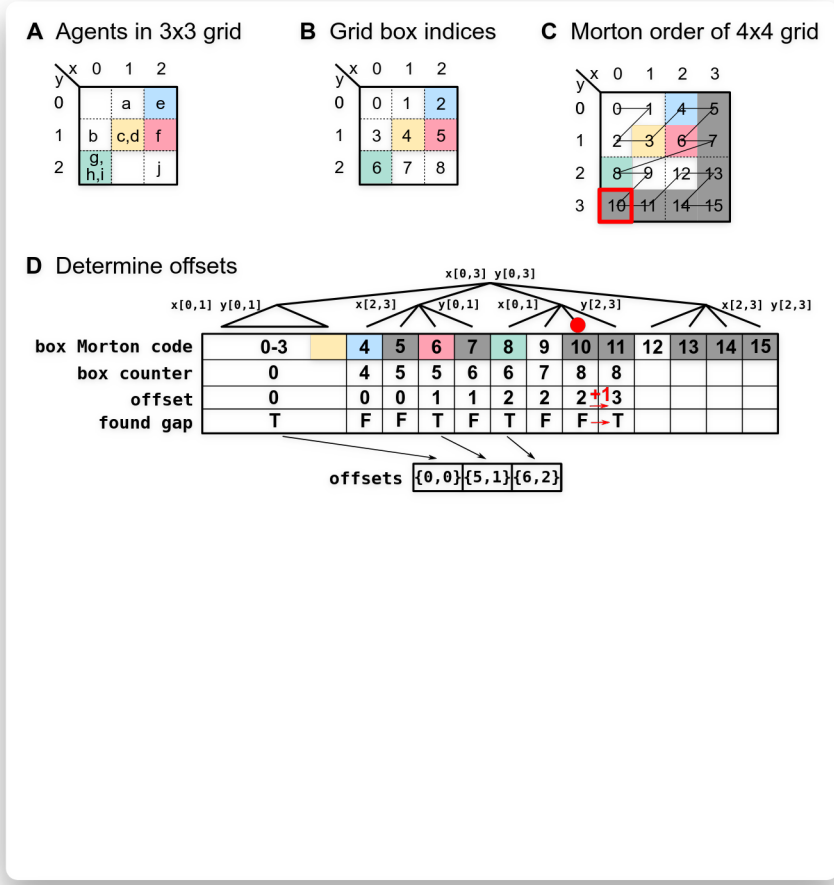
Agent sorting and balancing



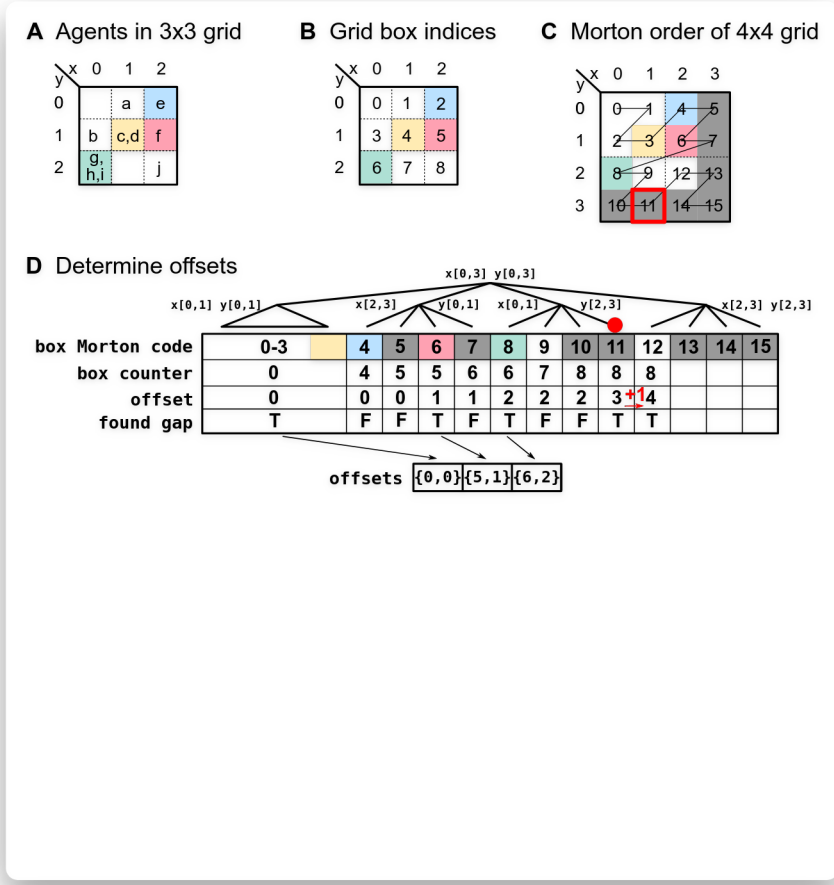
Agent sorting and balancing



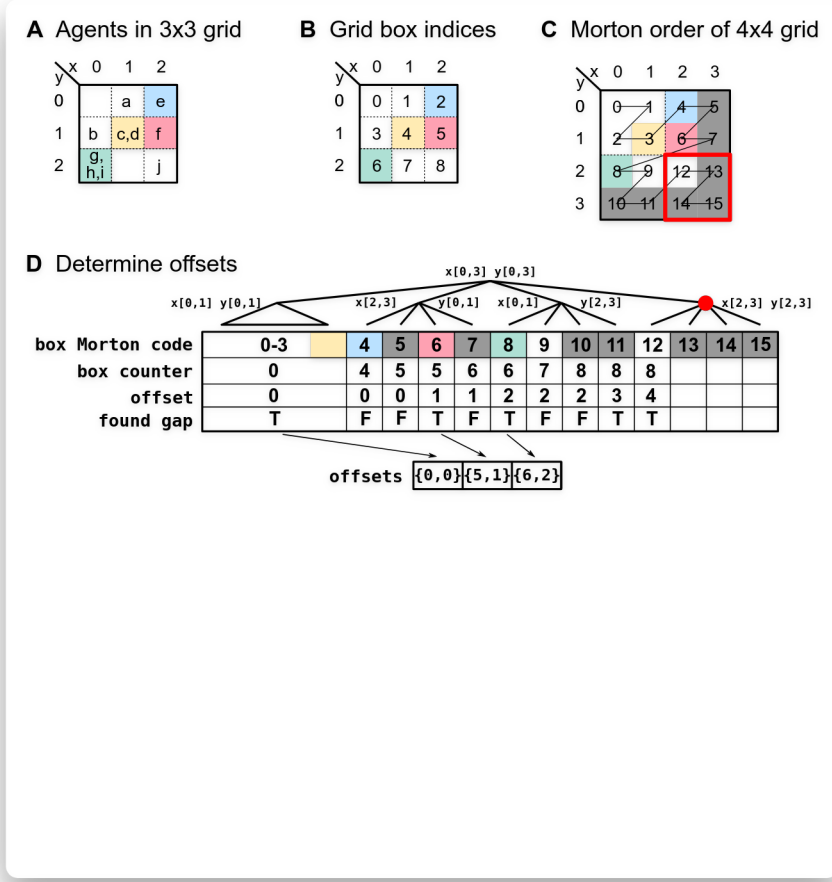
Agent sorting and balancing



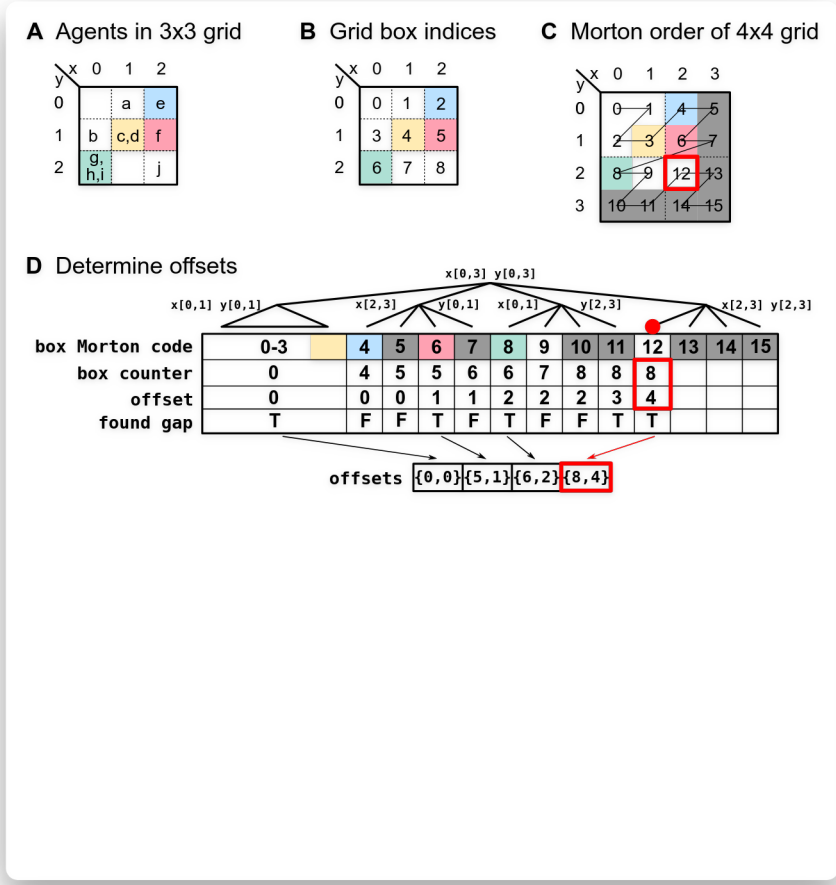
Agent sorting and balancing



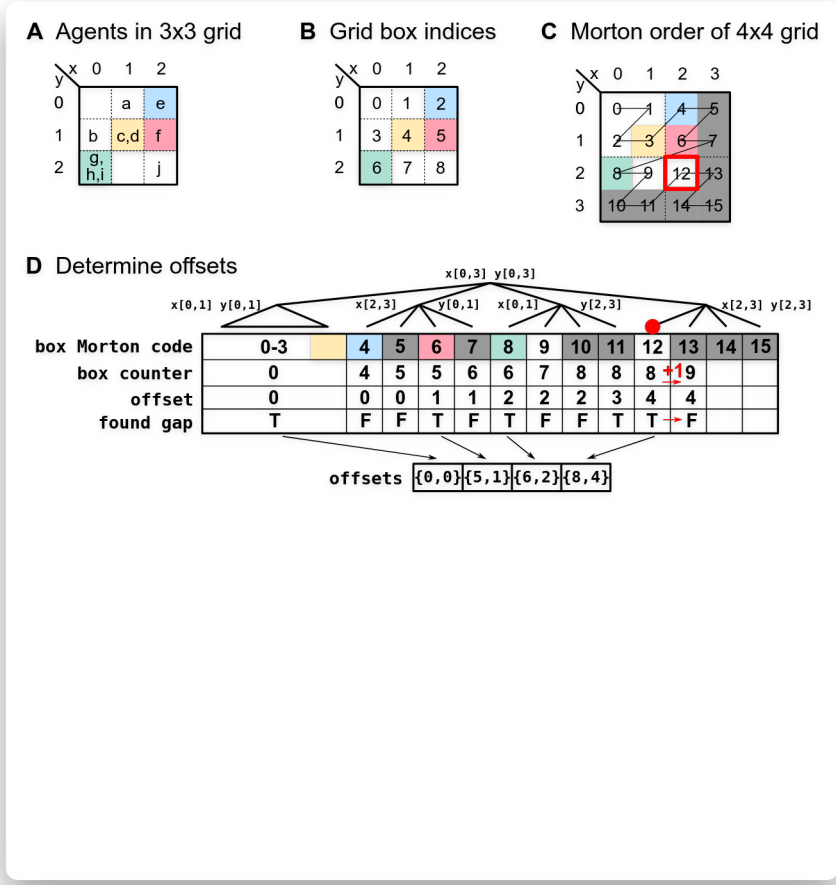
Agent sorting and balancing



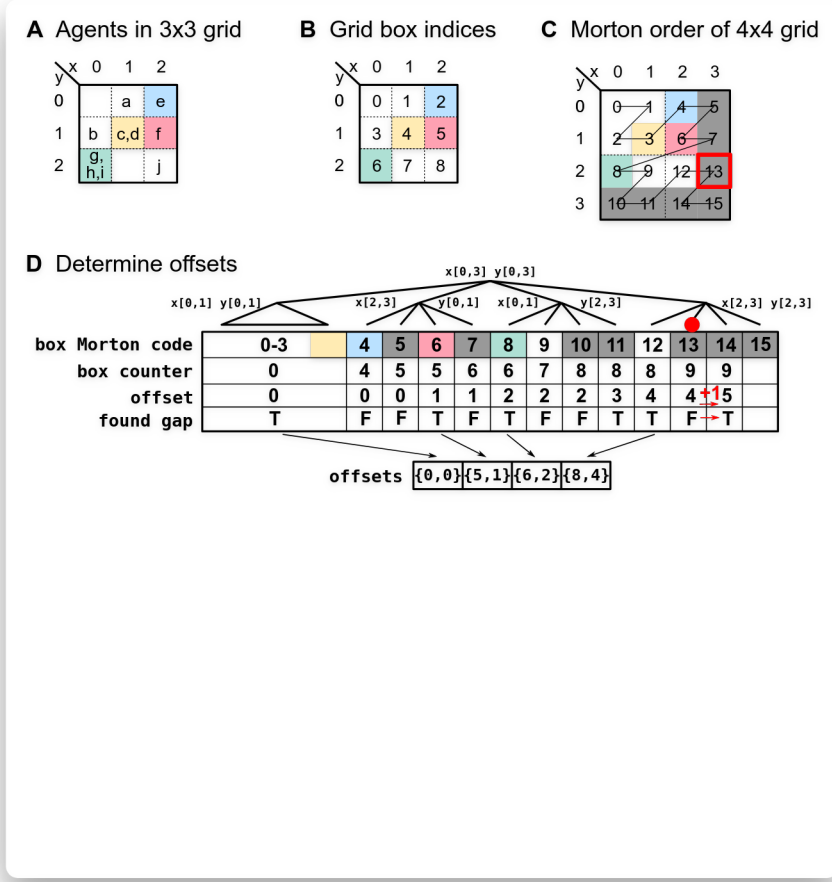
Agent sorting and balancing



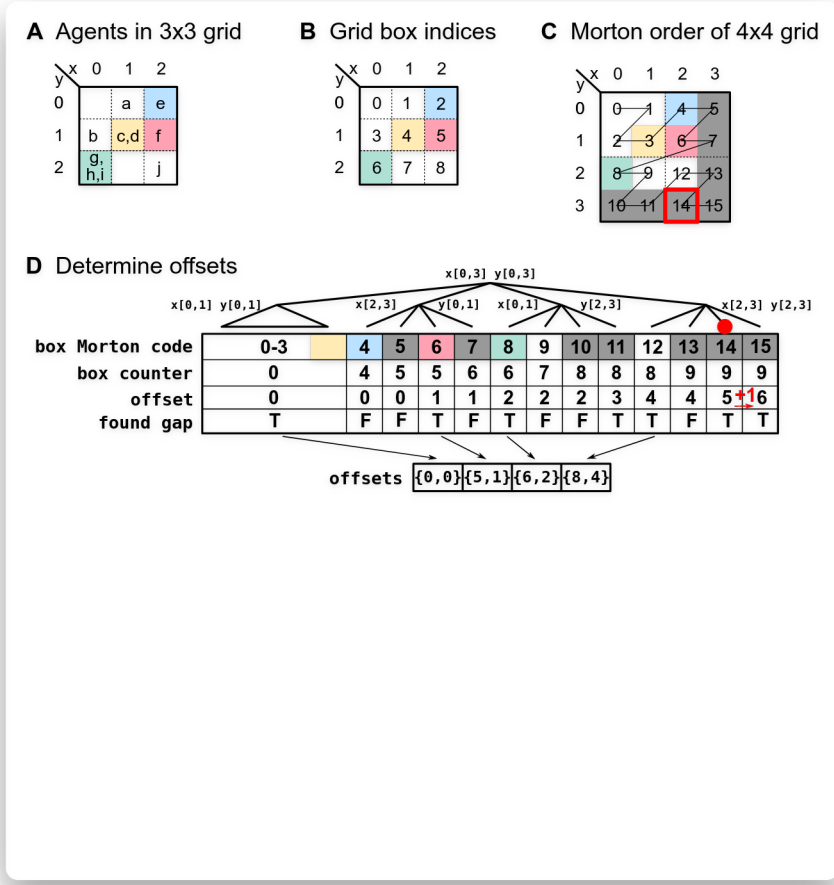
Agent sorting and balancing



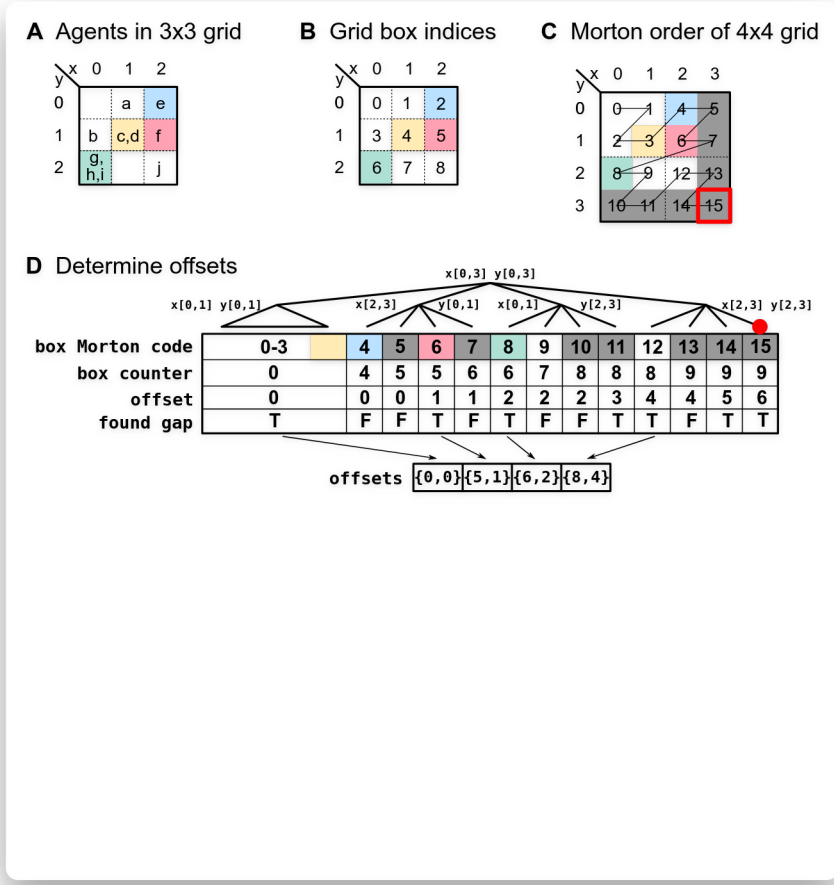
Agent sorting and balancing



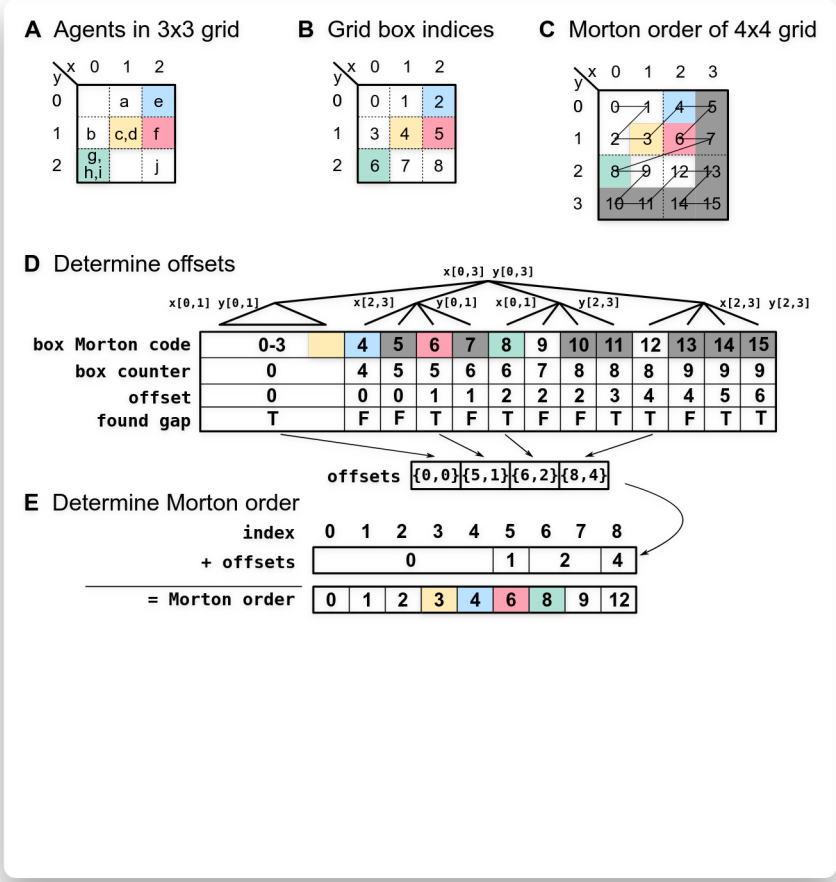
Agent sorting and balancing



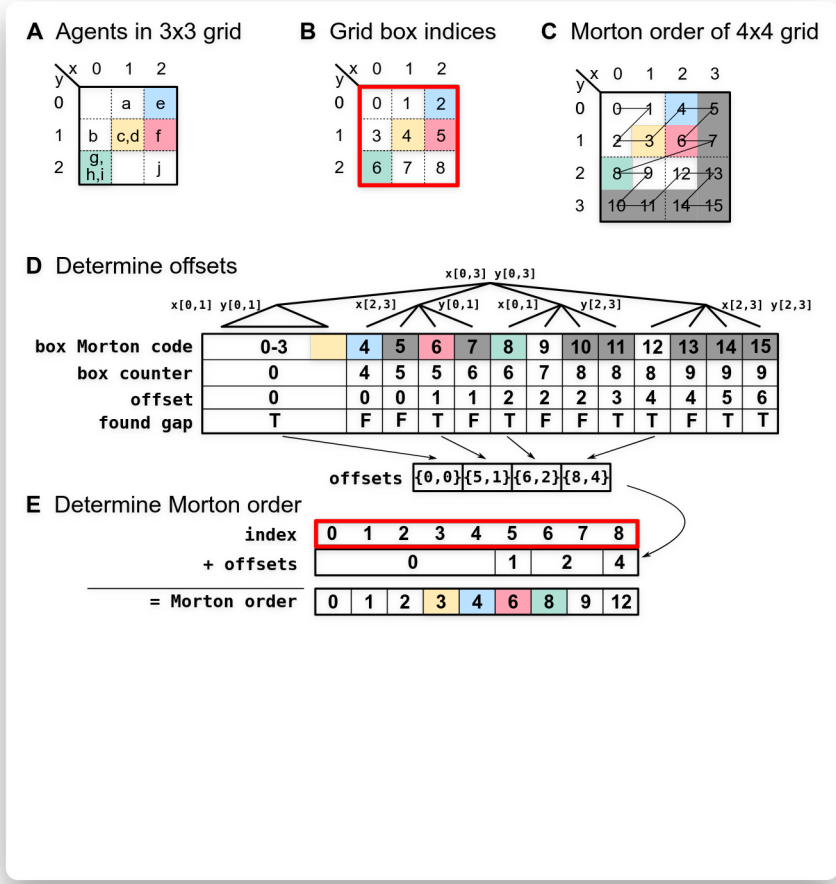
Agent sorting and balancing



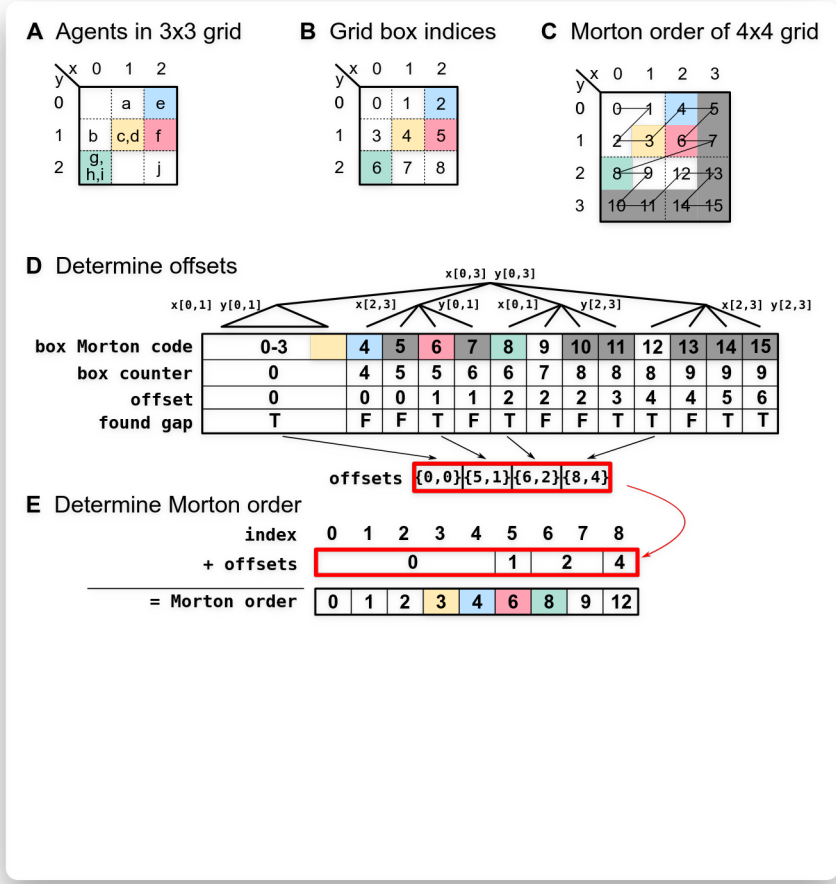
Agent sorting and balancing



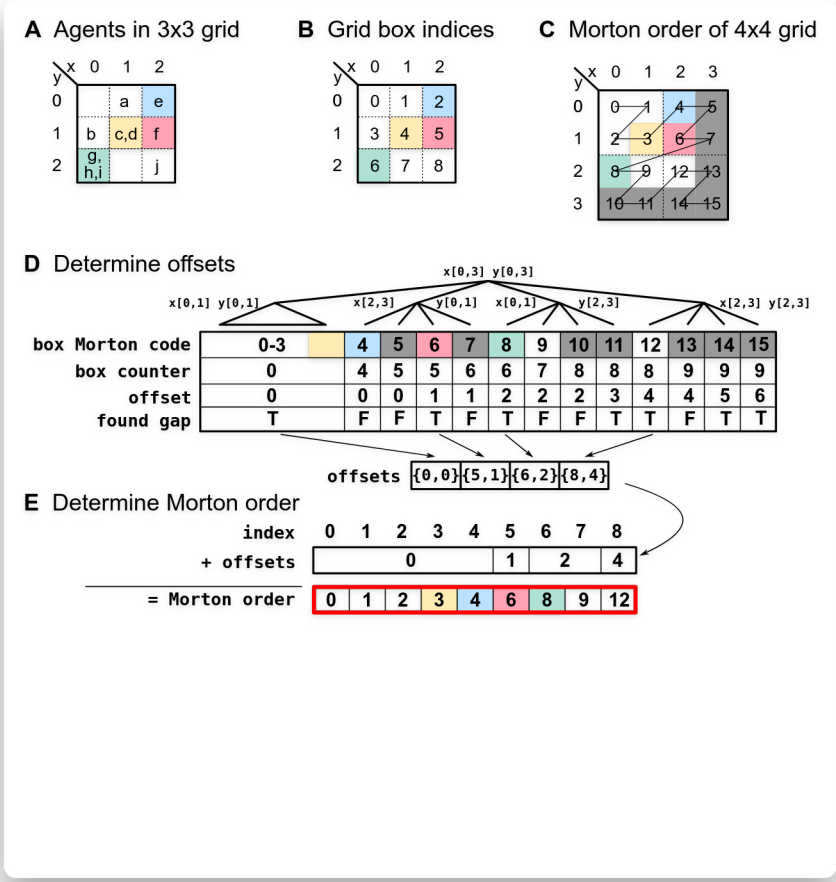
Agent sorting and balancing



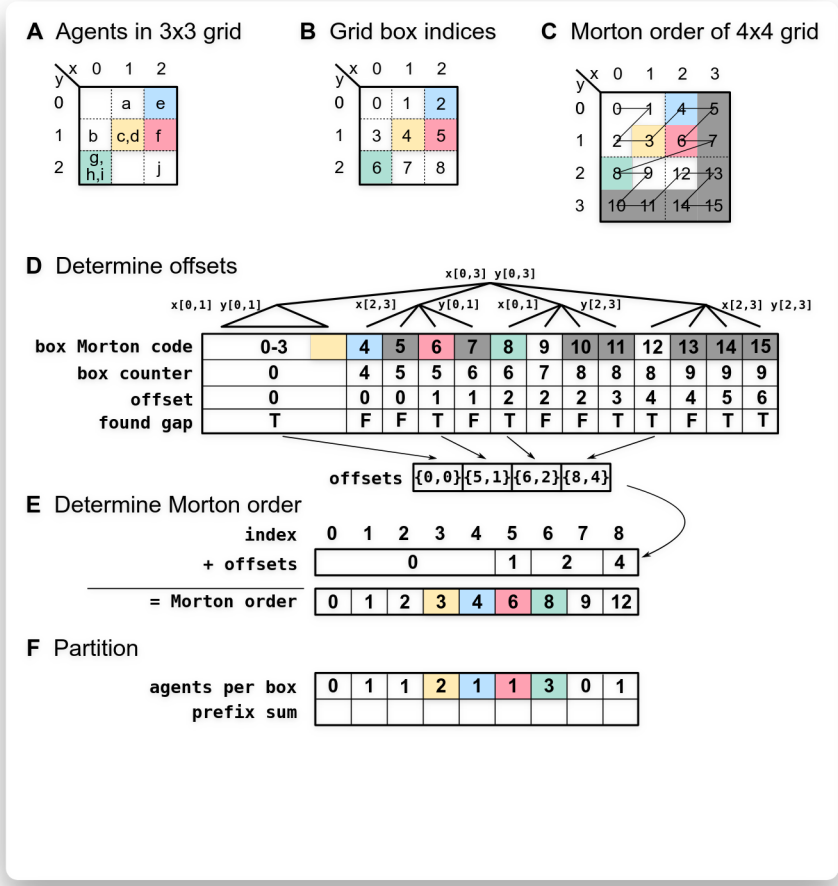
Agent sorting and balancing



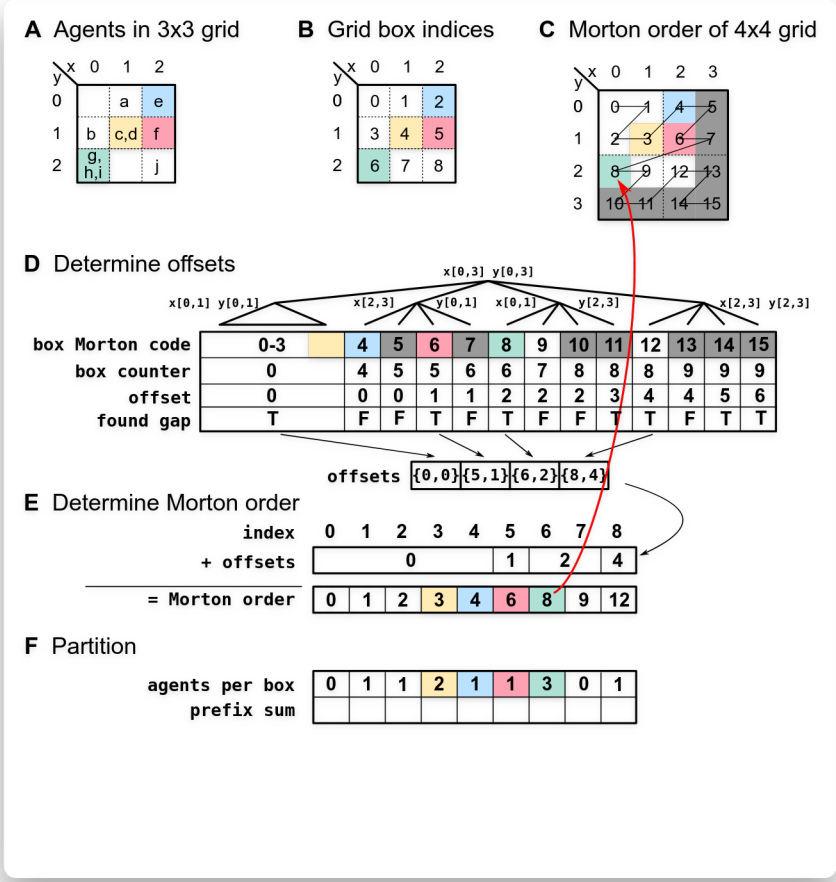
Agent sorting and balancing



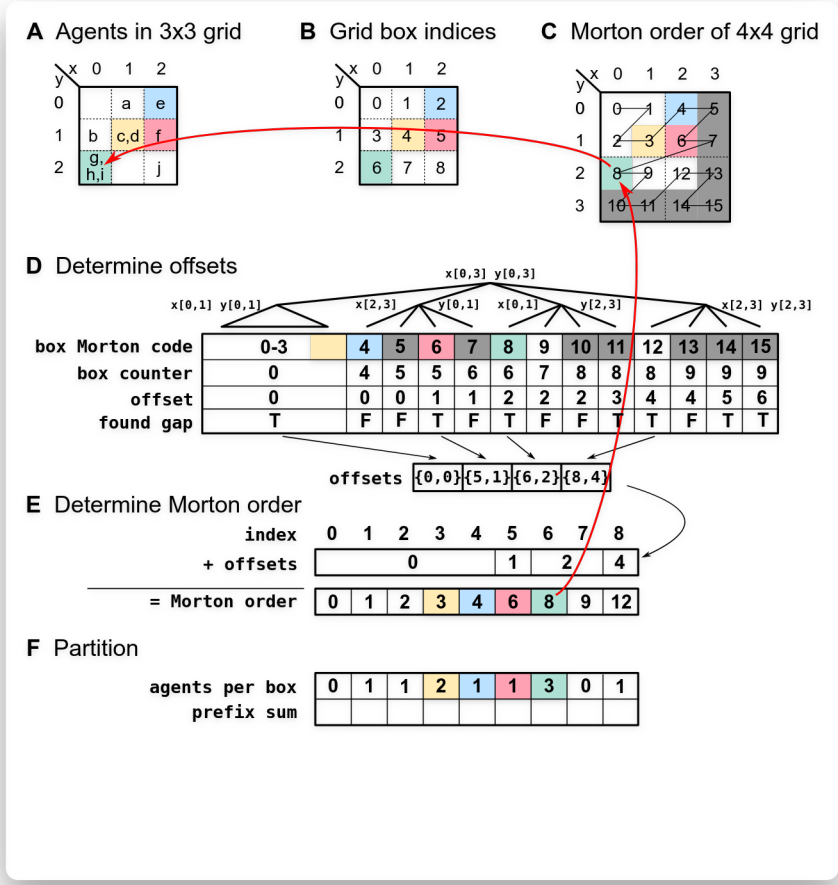
Agent sorting and balancing



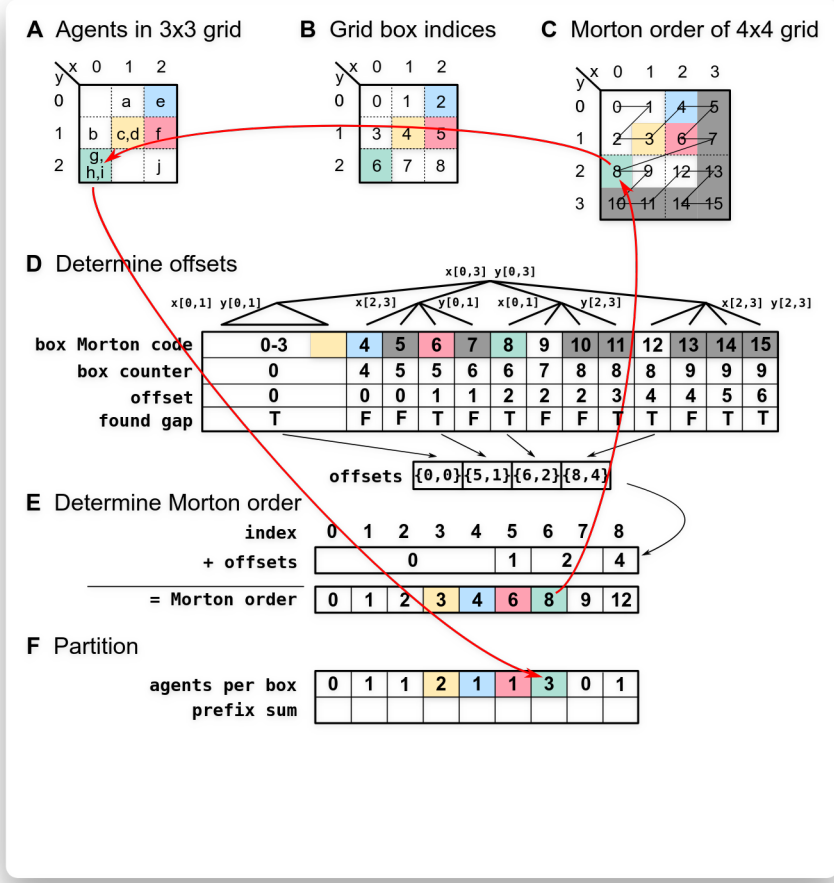
Agent sorting and balancing



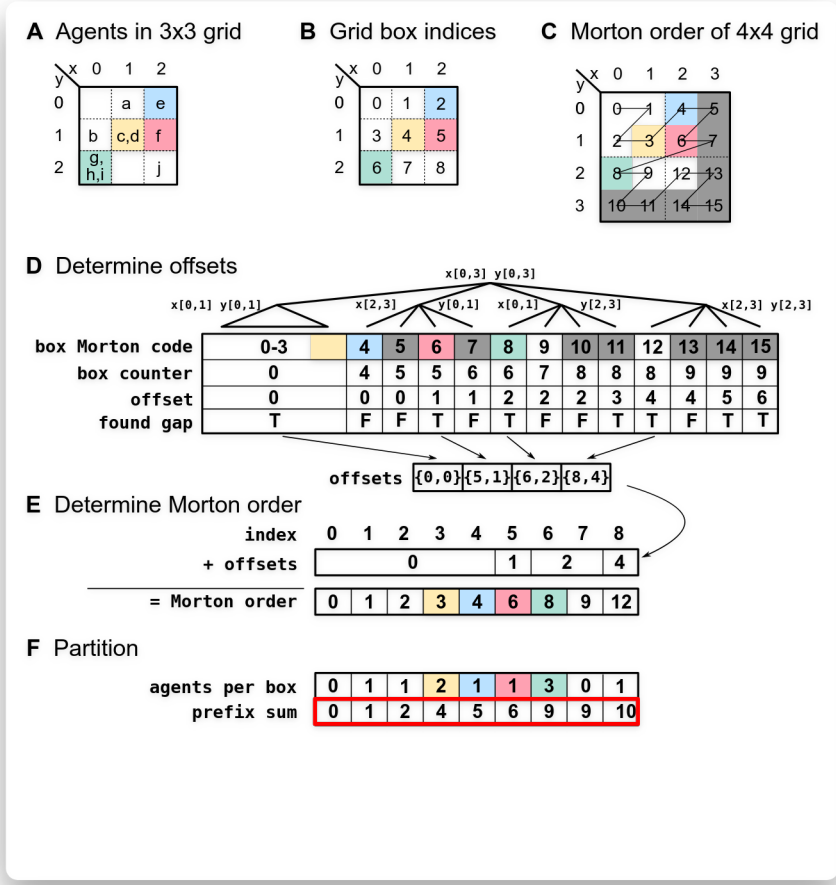
Agent sorting and balancing



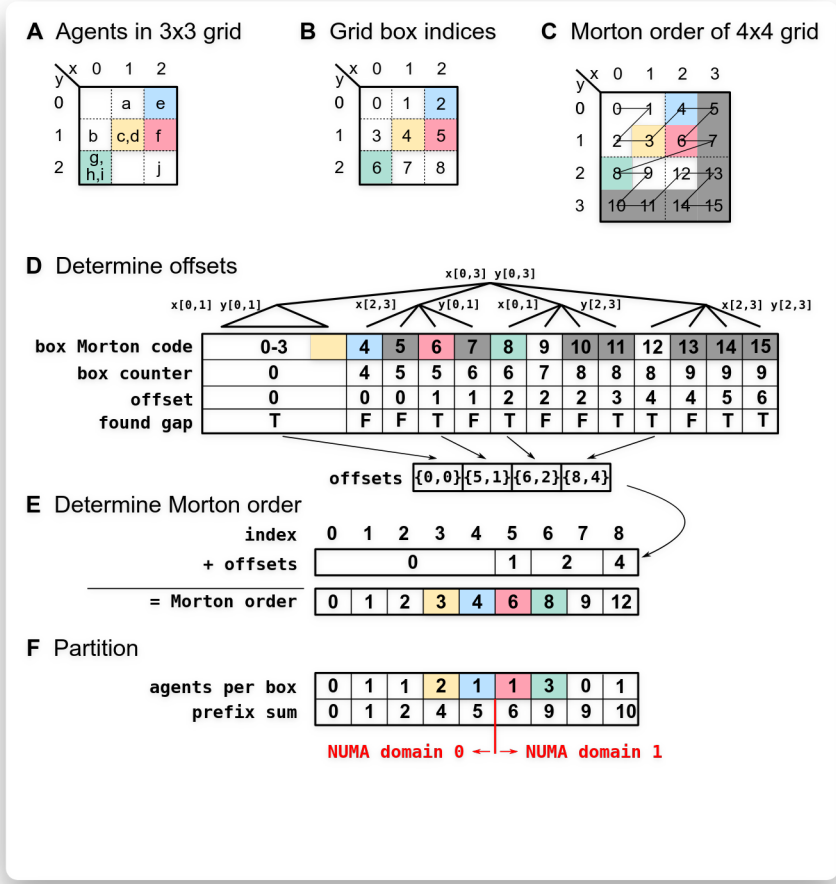
Agent sorting and balancing



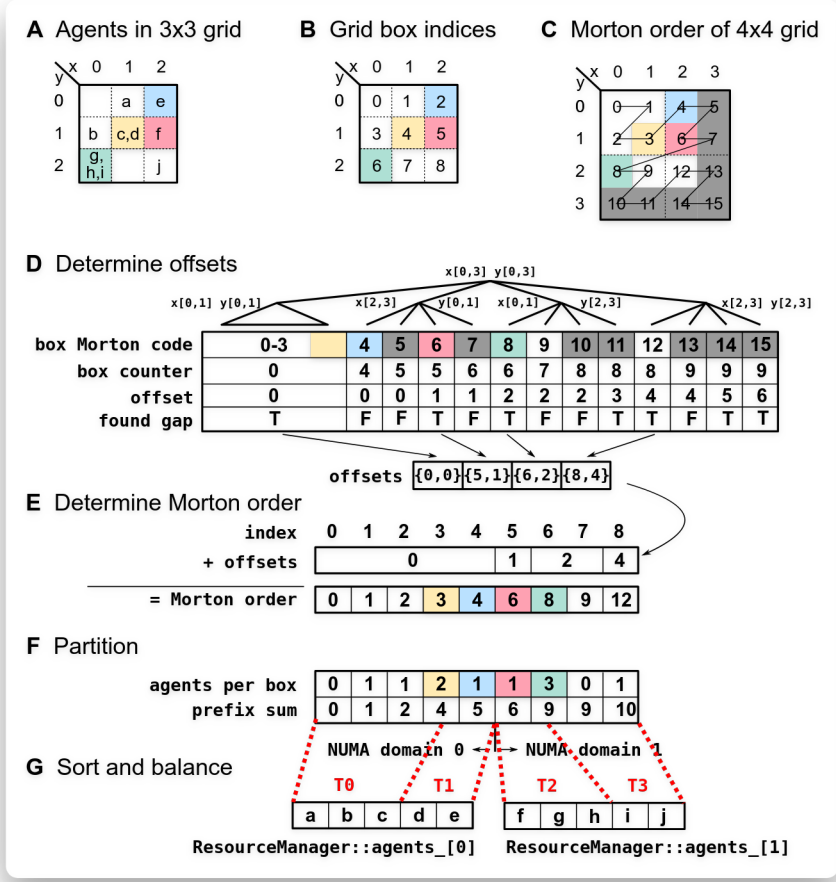
Agent sorting and balancing



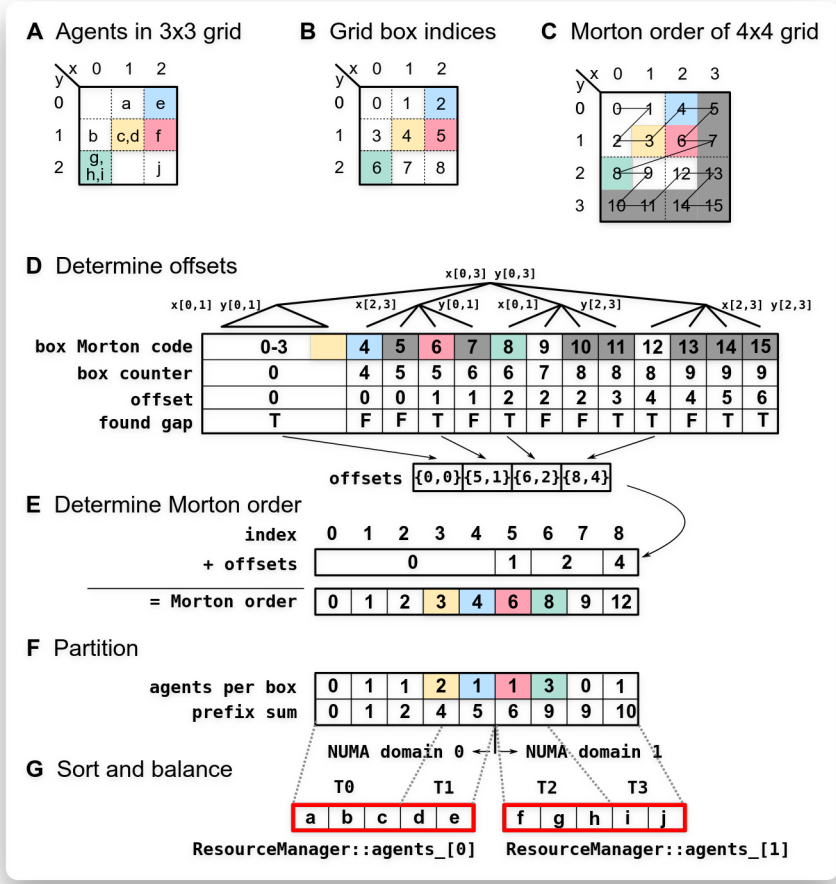
Agent sorting and balancing



Agent sorting and balancing



Agent sorting and balancing



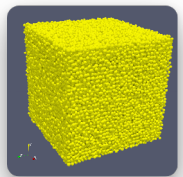
Evaluation

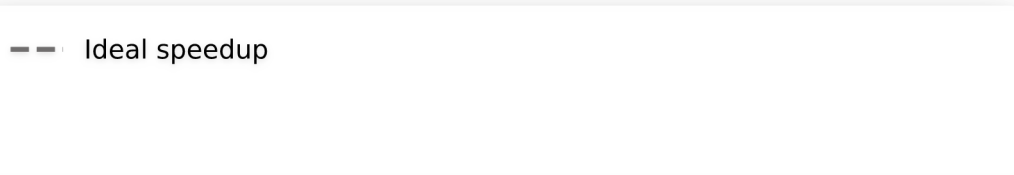
Benchmark hardware

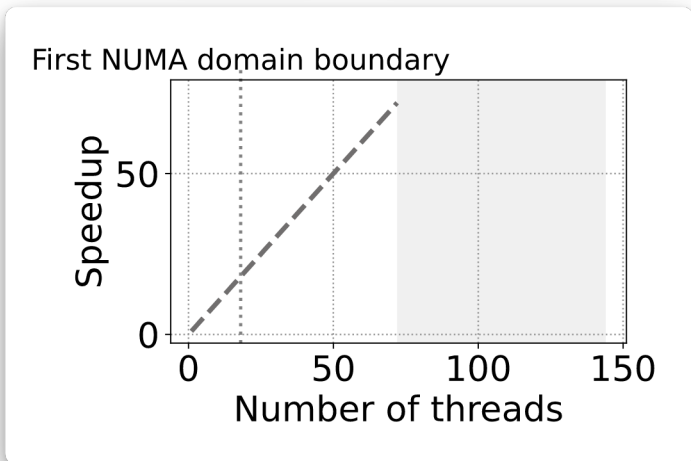
System	Memory	CPU	OS
A	504 GB	Four Intel(R) Xeon(R) E7-8890 v3 CPUs @ 2.50GHz with a total of 72 physical cores, two threads per core and four NUMA domains.	CentOS 7.9.2009
B	1008 GB		
C	62 GB	Two Intel(R) Xeon(R) E5-2683 v3 CPUs @ 2.00GHz with a total of 28 physical cores, two threads per core and two NUMA domains.	CentOS Stream 8

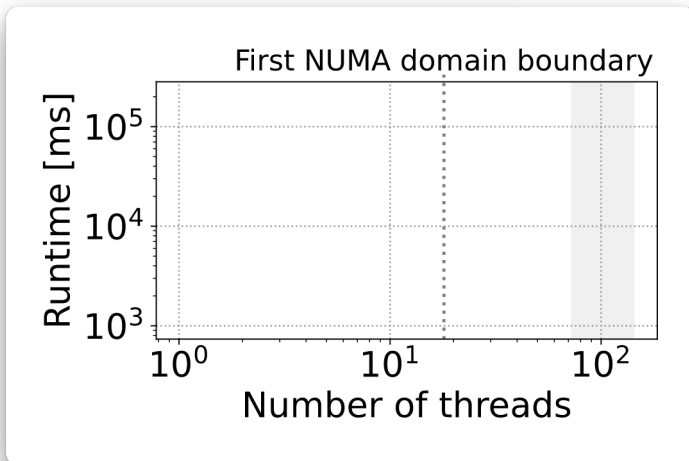
BioDynaMo **scales well** across NUMA nodes & large CPU core counts

Cell proliferation simulation



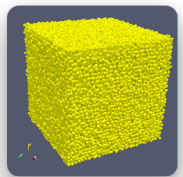






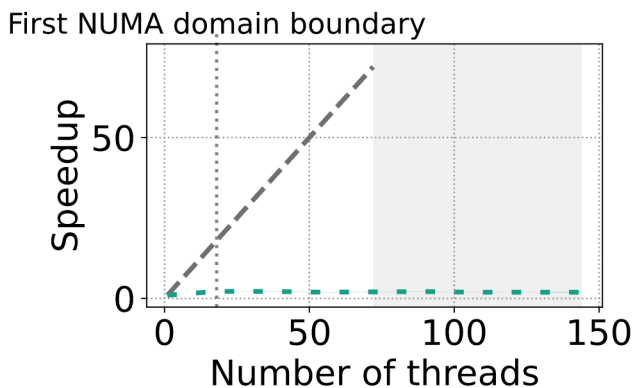
BioDynaMo **scales well** across NUMA nodes & large CPU core counts

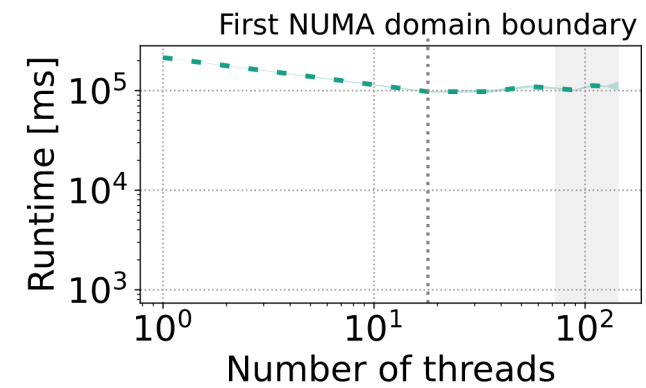
Cell proliferation simulation



Ideal speedup

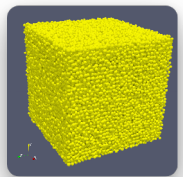
BDM standard implementation



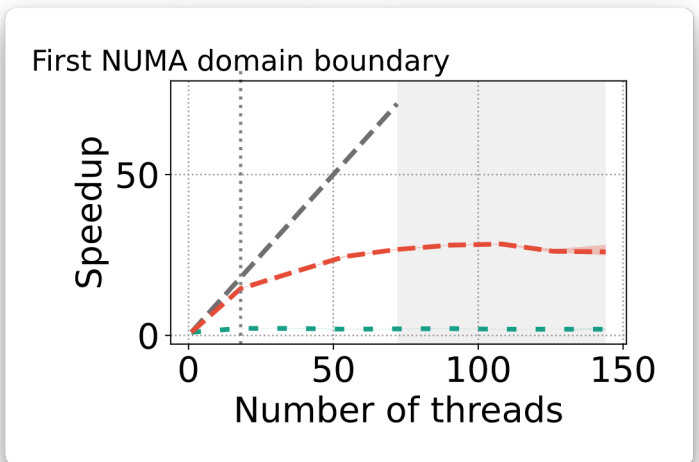


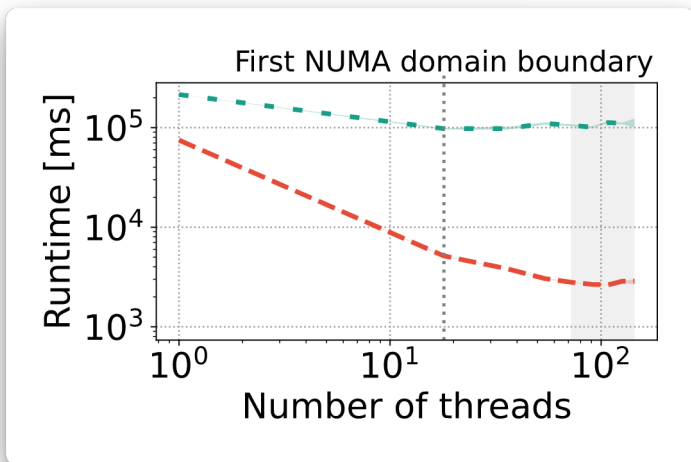
BioDynaMo **scales well** across NUMA nodes & large CPU core counts

Cell proliferation simulation



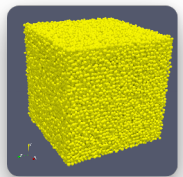
Ideal speedup
BDM standard implementation
Plus BDM uniform grid



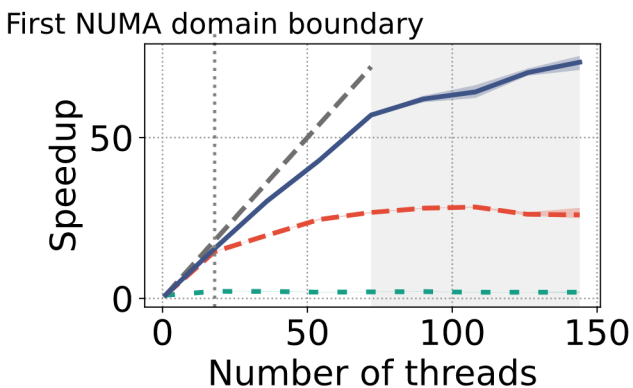


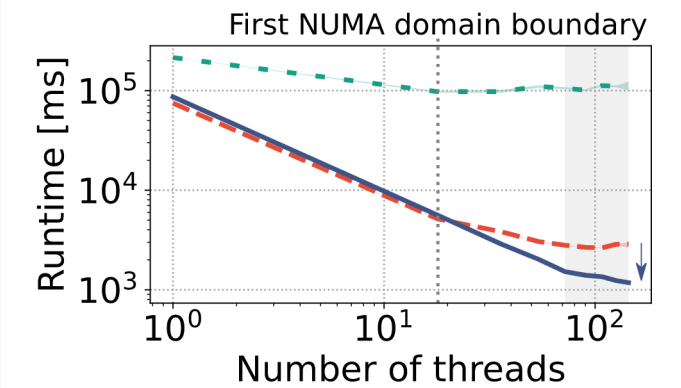
BioDynaMo **scales well** across NUMA nodes & large CPU core counts

Cell proliferation simulation



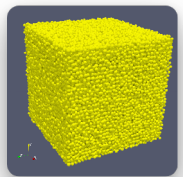
Ideal speedup
BDM standard implementation
Plus BDM uniform grid
Plus memory improvements



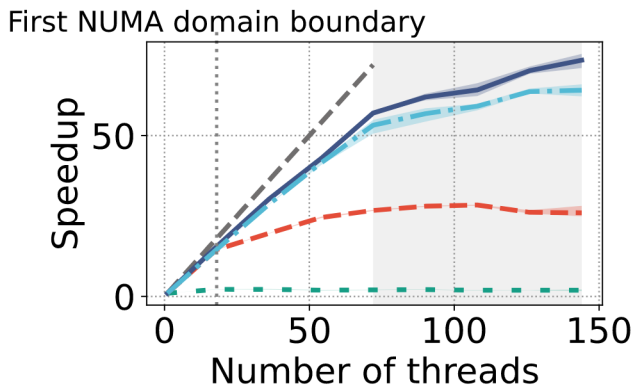


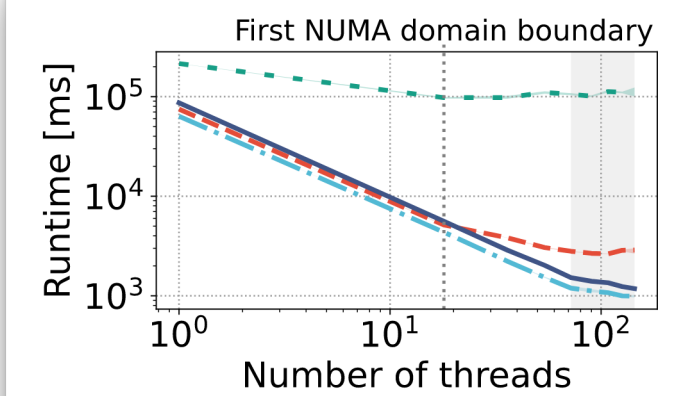
BioDynaMo **scales well** across NUMA nodes & large CPU core counts

Cell proliferation simulation



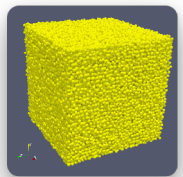
Ideal speedup
BDM standard implementation
Plus BDM uniform grid
Plus memory improvements
Plus extra memory



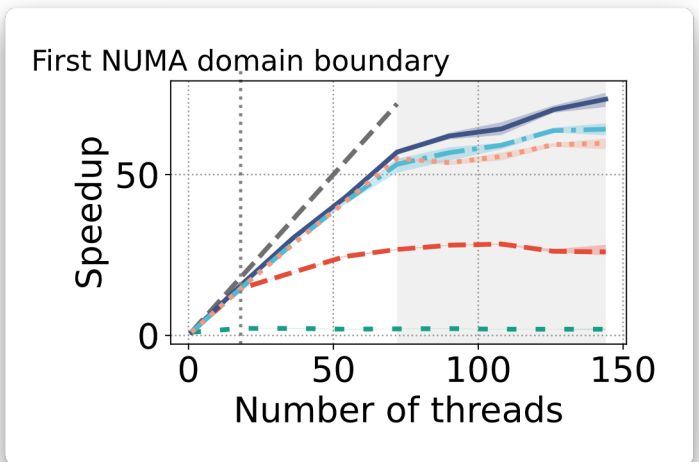


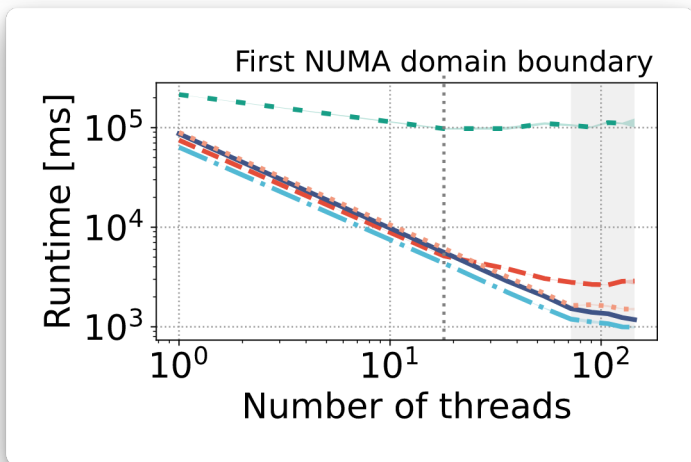
BioDynaMo **scales well** across NUMA nodes & large CPU core counts

Cell proliferation simulation



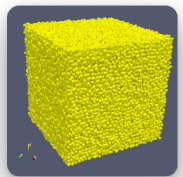
Ideal speedup
BDM standard implementation
Plus BDM uniform grid
Plus memory improvements
Plus extra memory
Plus static agents detection



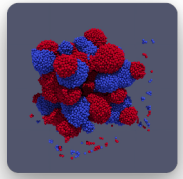


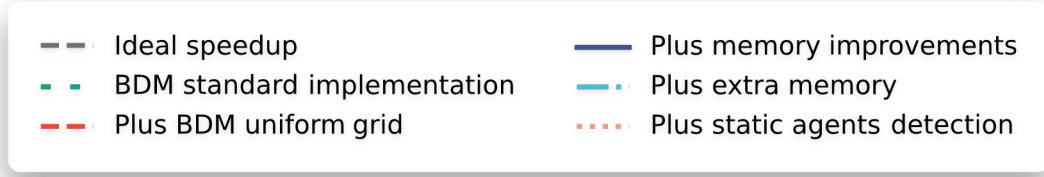
BioDynaMo **scales well** across NUMA nodes & large CPU core counts

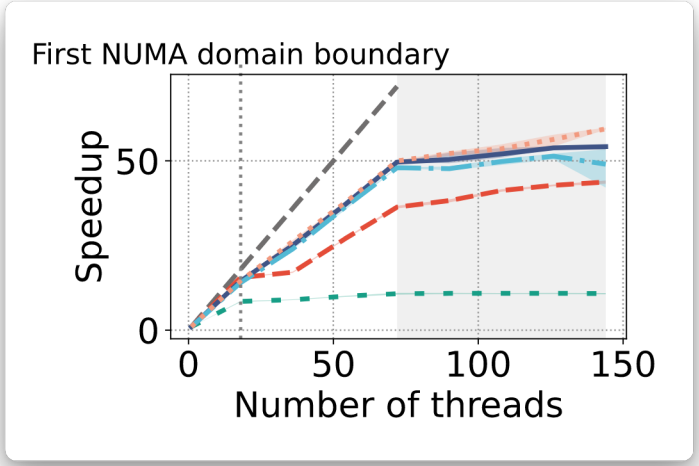
Cell proliferation simulation

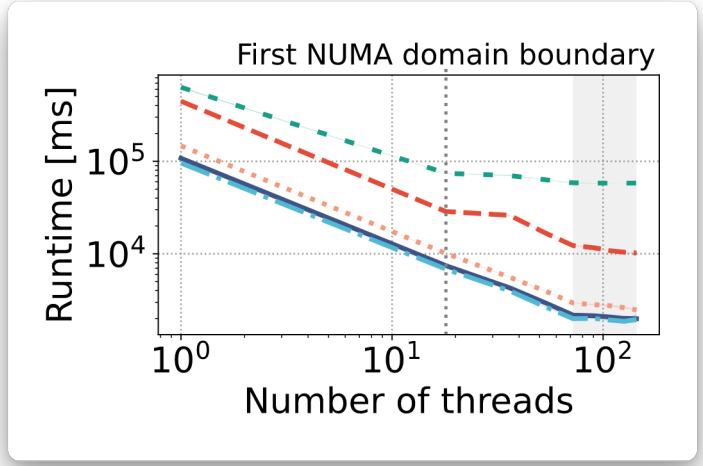


Cell clustering benefits from BioDynaMo's **memory optimizations** for any number of CPU cores

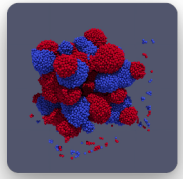




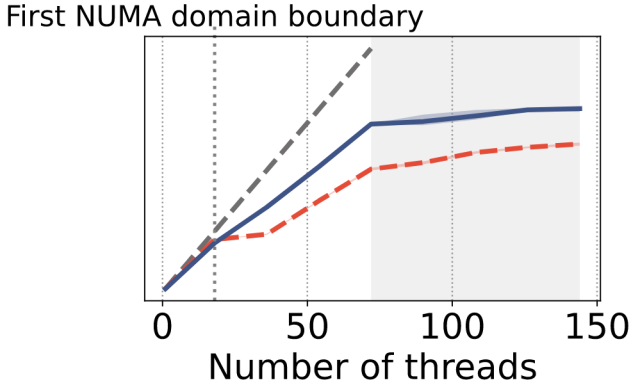


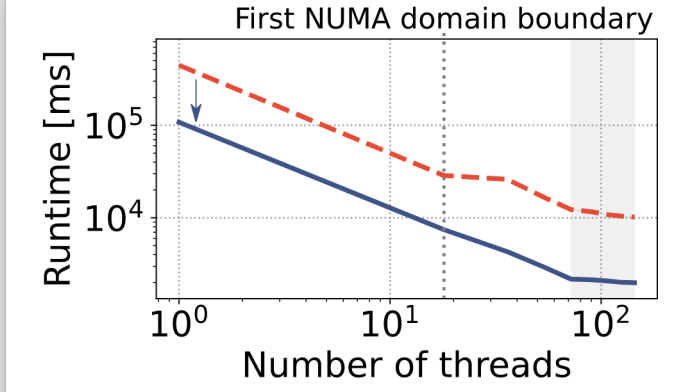


Cell clustering benefits from BioDynaMo's **memory optimizations** for any number of CPU cores

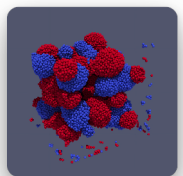


Ideal speedup
BDM standard implementation
Plus BDM uniform grid
Plus memory improvements
Plus extra memory
Plus static agents detection



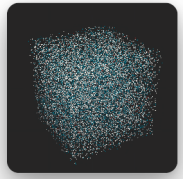


Cell clustering benefits from BioDynaMo's **memory optimizations** for any number of CPU cores

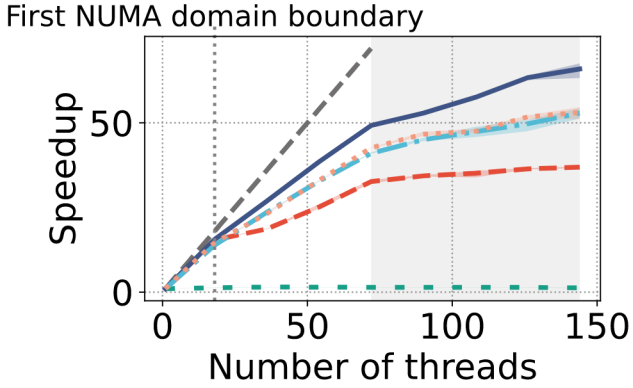


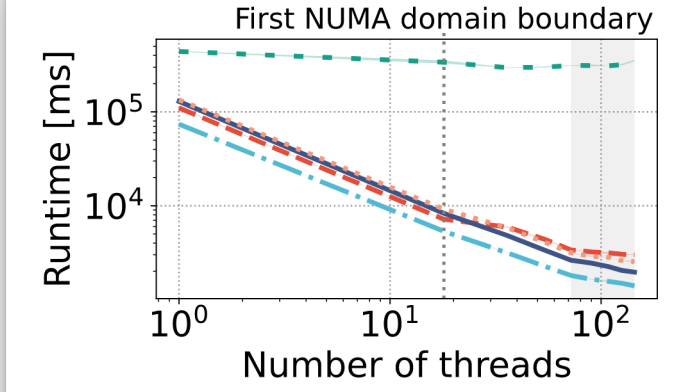
Ideal speedup
BDM standard implementation
Plus BDM uniform grid
Plus memory improvements
Plus extra memory
Plus static agents detection

The epidemiology use case benefits from using **additional memory**

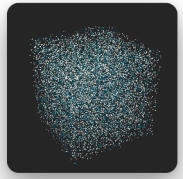


Ideal speedup
BDM standard implementation
Plus BDM uniform grid
Plus memory improvements
Plus extra memory
Plus static agents detection

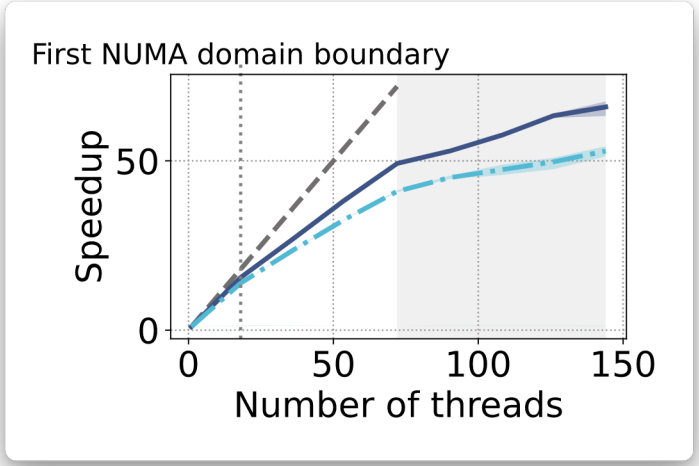


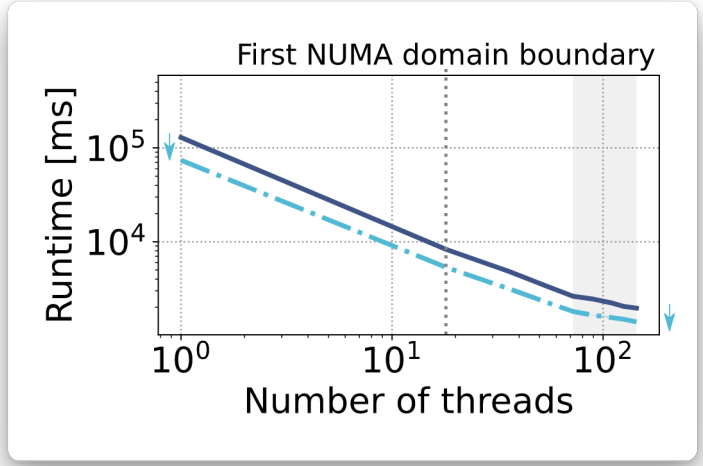


The epidemiology use case benefits from using **additional memory**

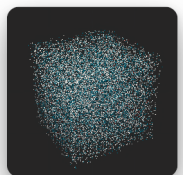


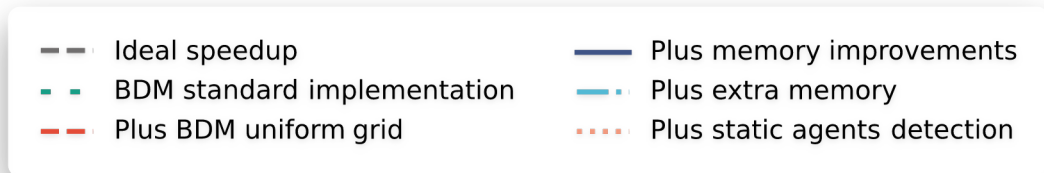
Ideal speedup
BDM standard implementation
Plus BDM uniform grid
Plus memory improvements
Plus extra memory
Plus static agents detection



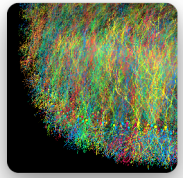


The epidemiology use case benefits from using **additional memory**

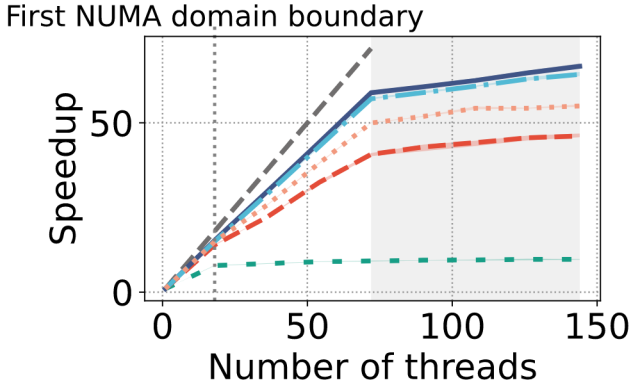


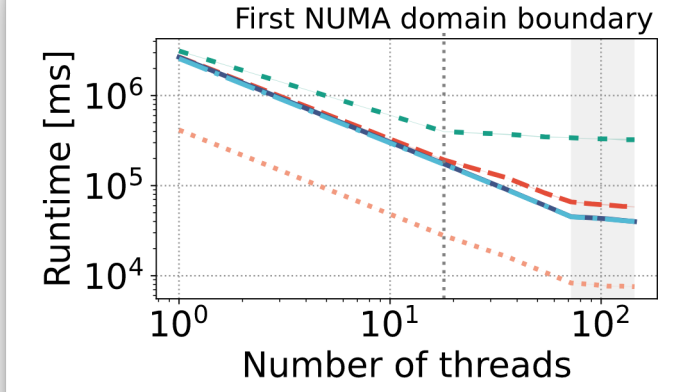


The neuroscience use case benefits substantially from **static region detection**

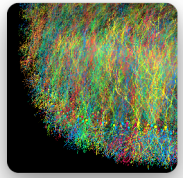


Ideal speedup
BDM standard implementation
Plus BDM uniform grid
Plus memory improvements
Plus extra memory
Plus static agents detection

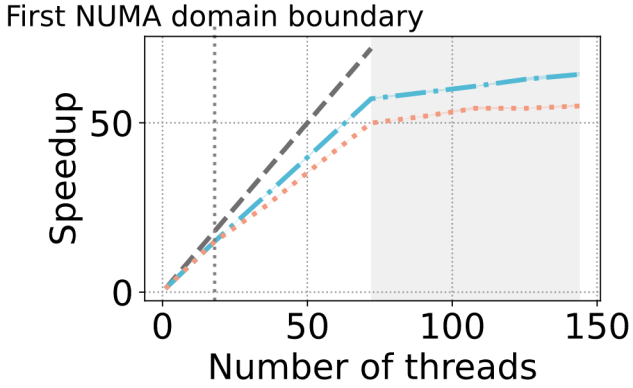


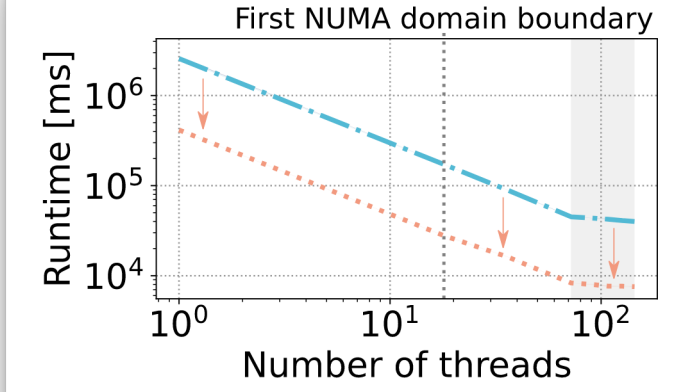


The neuroscience use case benefits substantially from **static region detection**

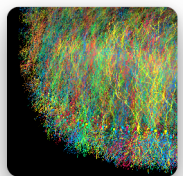


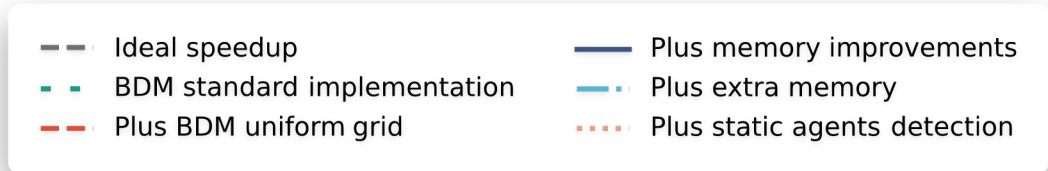
Ideal speedup
BDM standard implementation
Plus BDM uniform grid
Plus memory improvements
Plus extra memory
Plus static agents detection





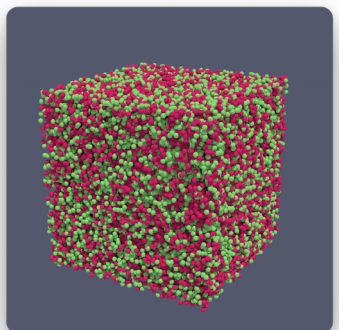
The neuroscience use case benefits substantially from **static region detection**



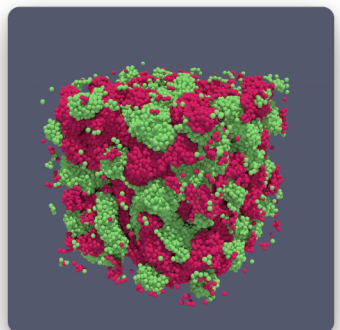


BioDynaMo is up to 9× **more efficient** than Biocellion

We implement Biocellions cell clustering simulation in BioDynaMo



Initial state

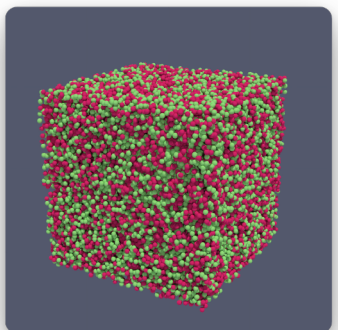


Final state

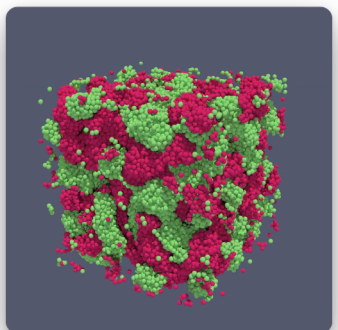
Biocellion paper: Kang et al., 2014, https://doi.org/10.1093/bioinformatics/btu498

BioDynaMo is up to 9× **more efficient** than Biocellion

We implement Biocellions cell clustering simulation in BioDynaMo



Initial state



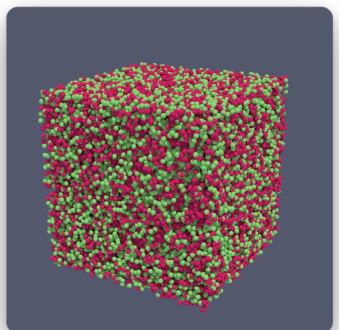
Final state

- Small-scale with **26.8 million** cells
  - 16 CPU cores (Biocellion & BioDynaMo)
  - Result: **BioDynaMo is 4.2× faster**

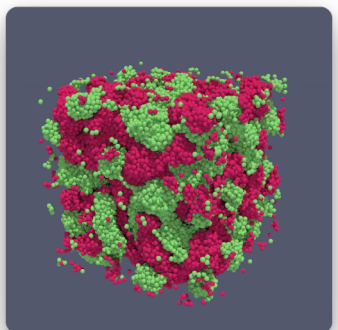
Biocellion paper: Kang et al., 2014, https://doi.org/10.1093/bioinformatics/btu498

BioDynaMo is up to 9× **more efficient** than Biocellion

We implement Biocellions cell clustering simulation in BioDynaMo



Initial state

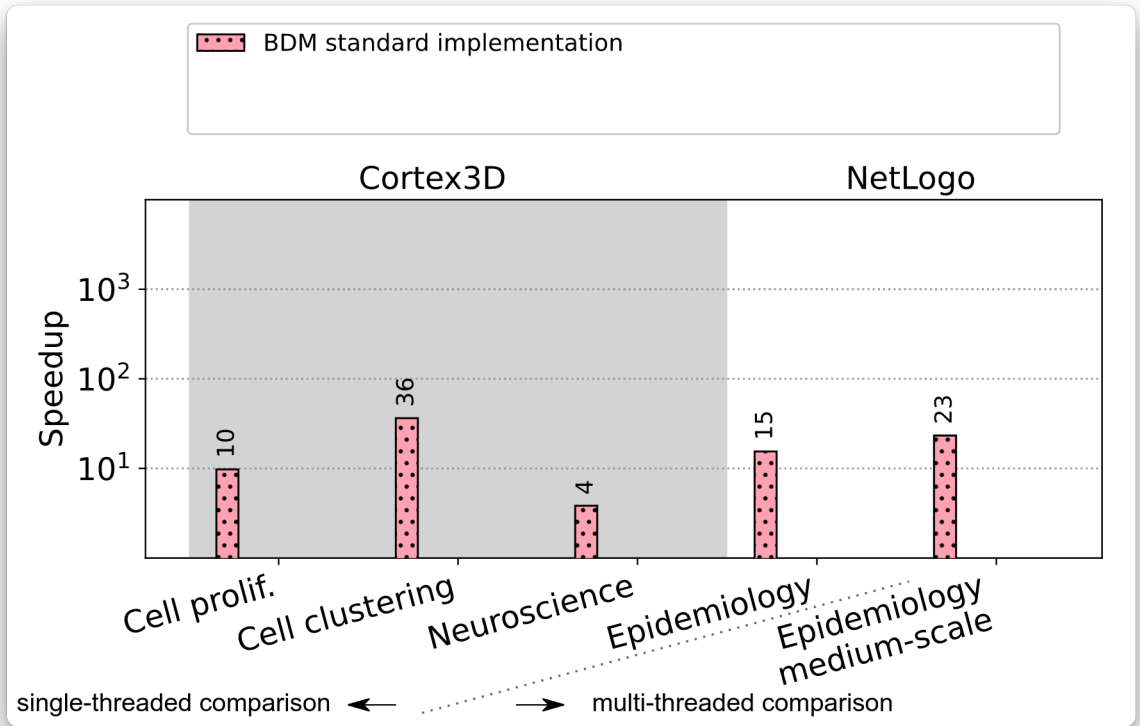


Final state

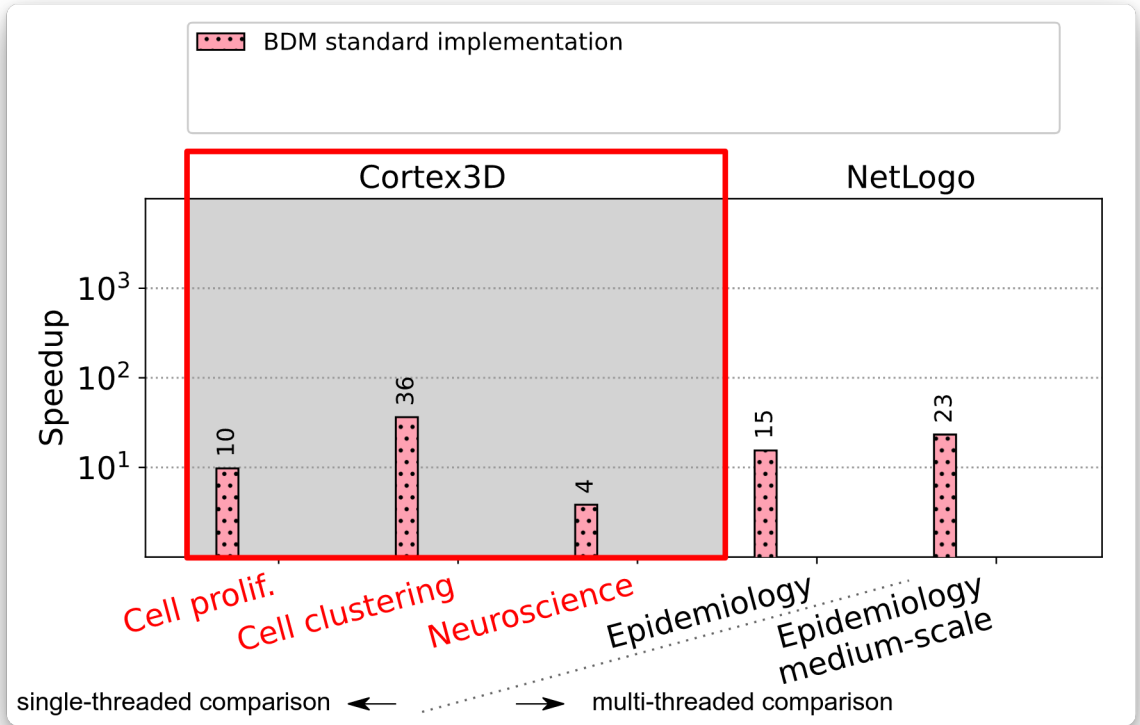
- Small-scale with **26.8 million** cells
  - 16 CPU cores (Biocellion & BioDynaMo)
  - Result: **BioDynaMo is 4.2× faster**
- Large-scale with 281 million cells
  - Biocellion: 21 nodes with 672 CPU cores
  - BioDynaMo: 1 node with 72 CPU cores
  - Result: same runtime, but BioDynaMo uses **9.3× fewer CPU cores**

Biocellion paper: Kang et al., 2014, https://doi.org/10.1093/bioinformatics/btu498

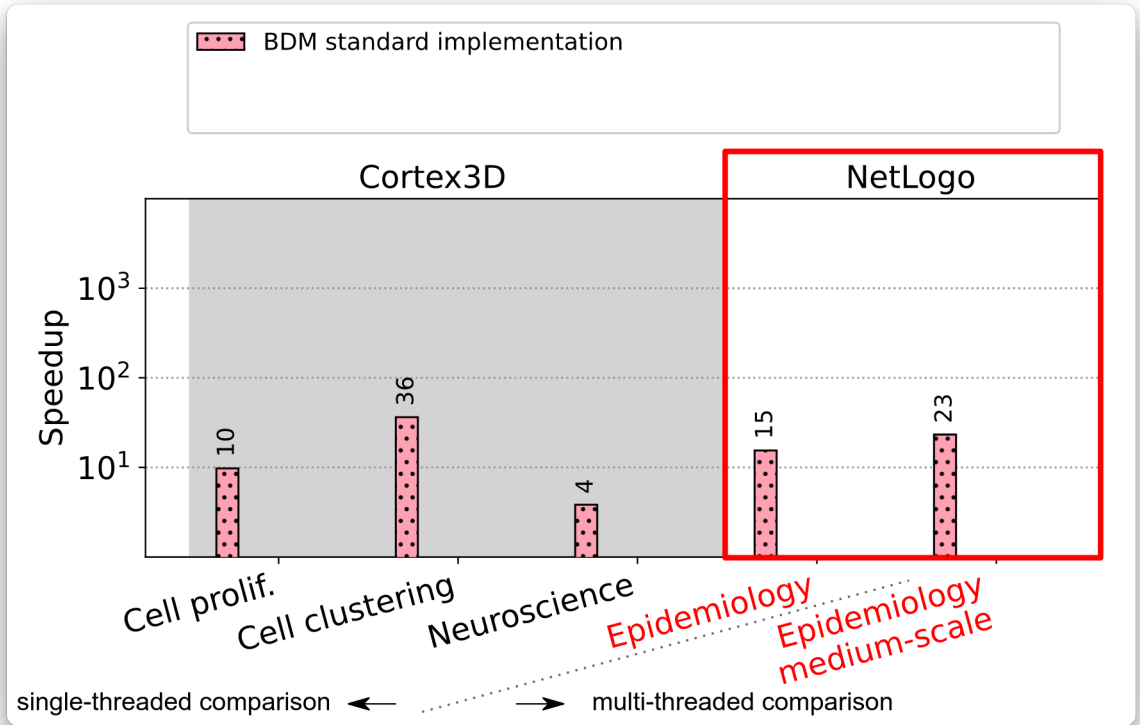
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



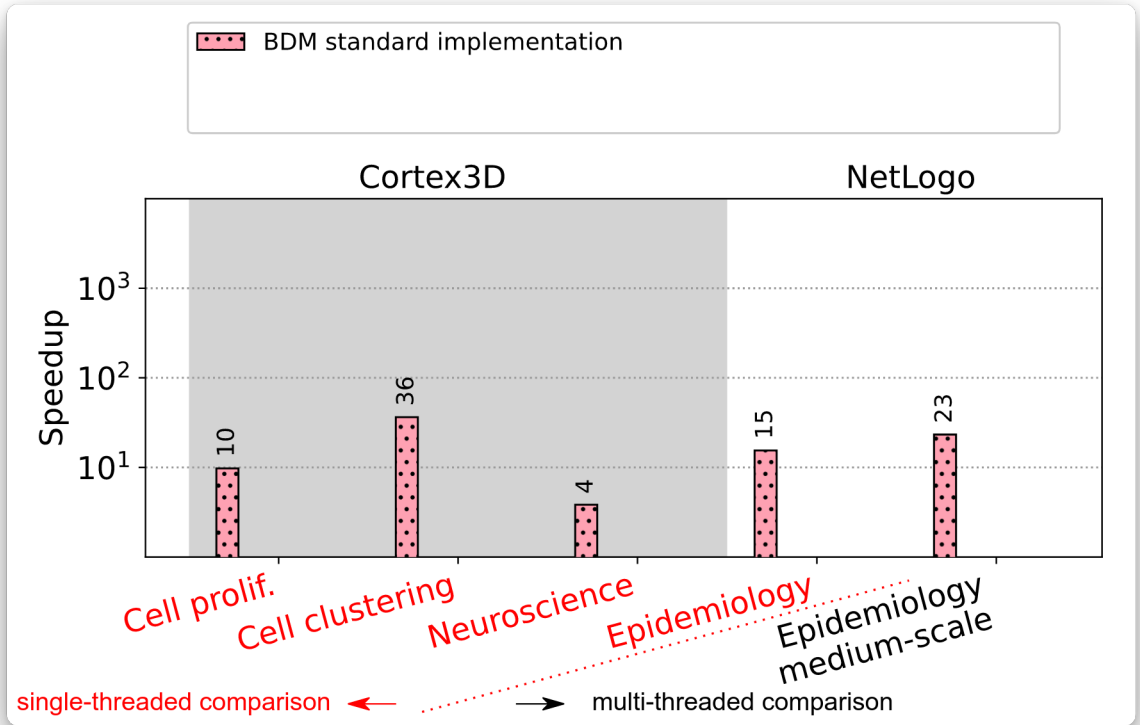
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



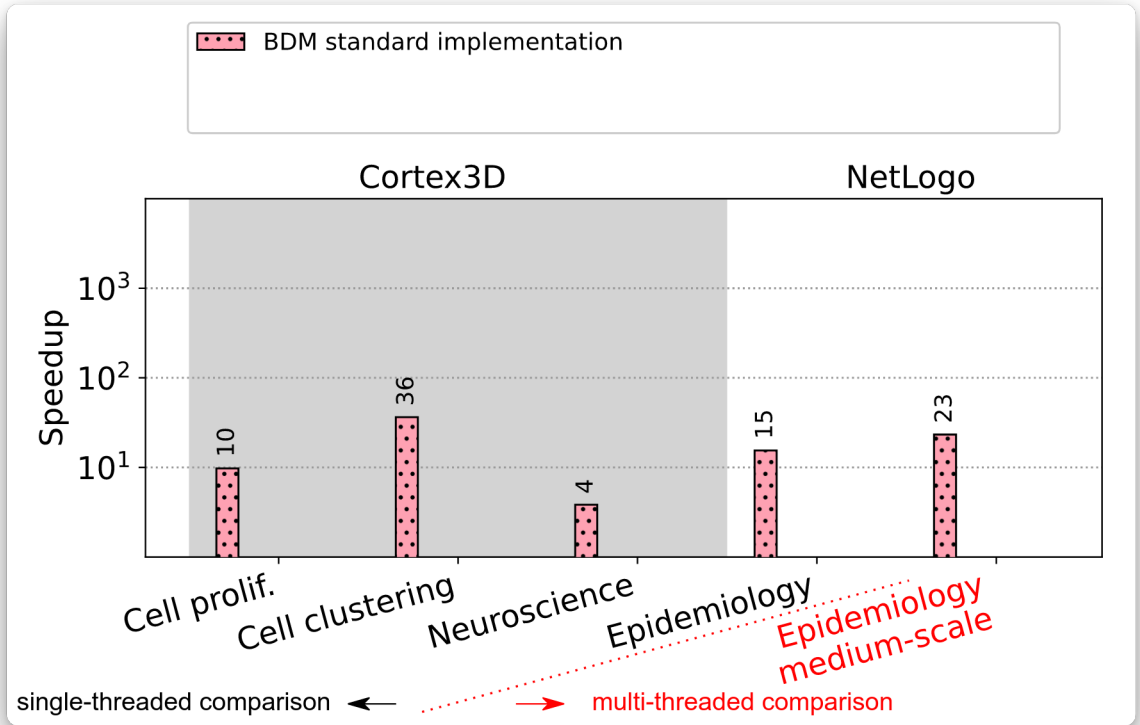
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



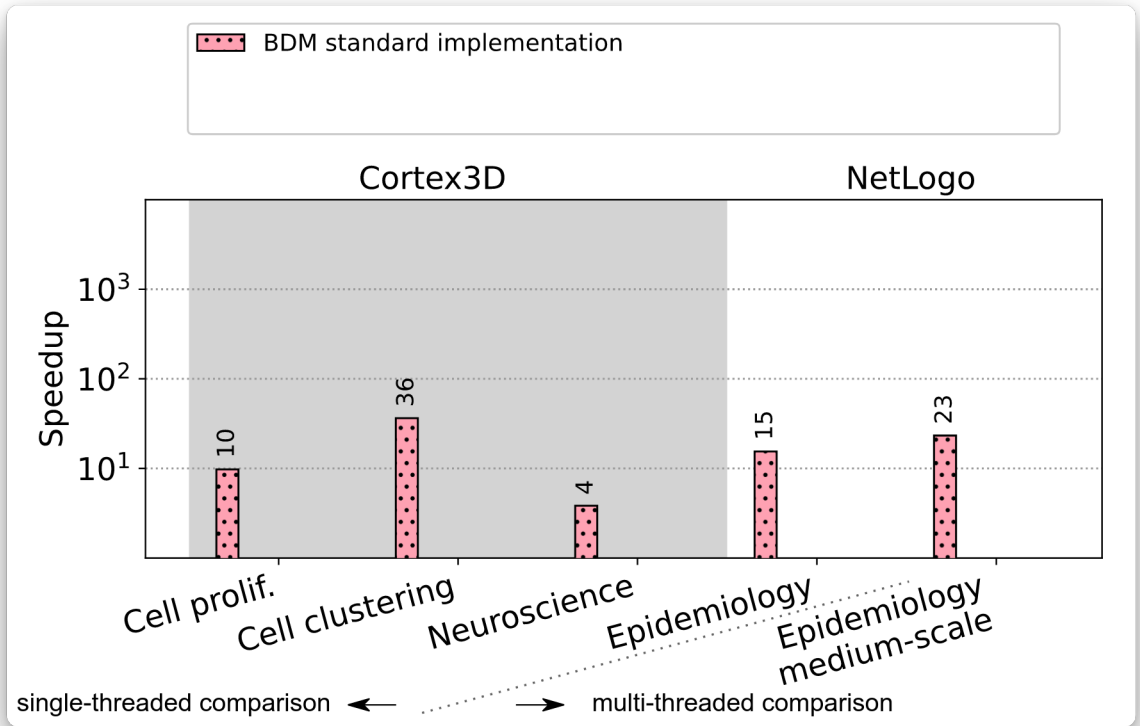
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



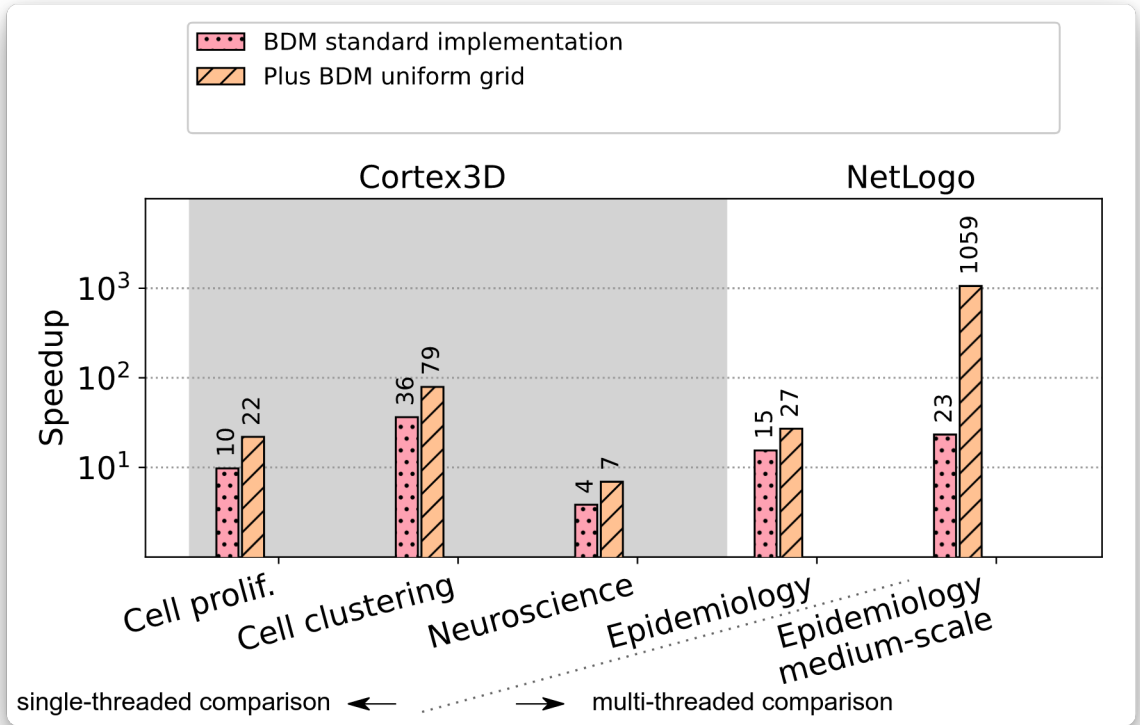
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



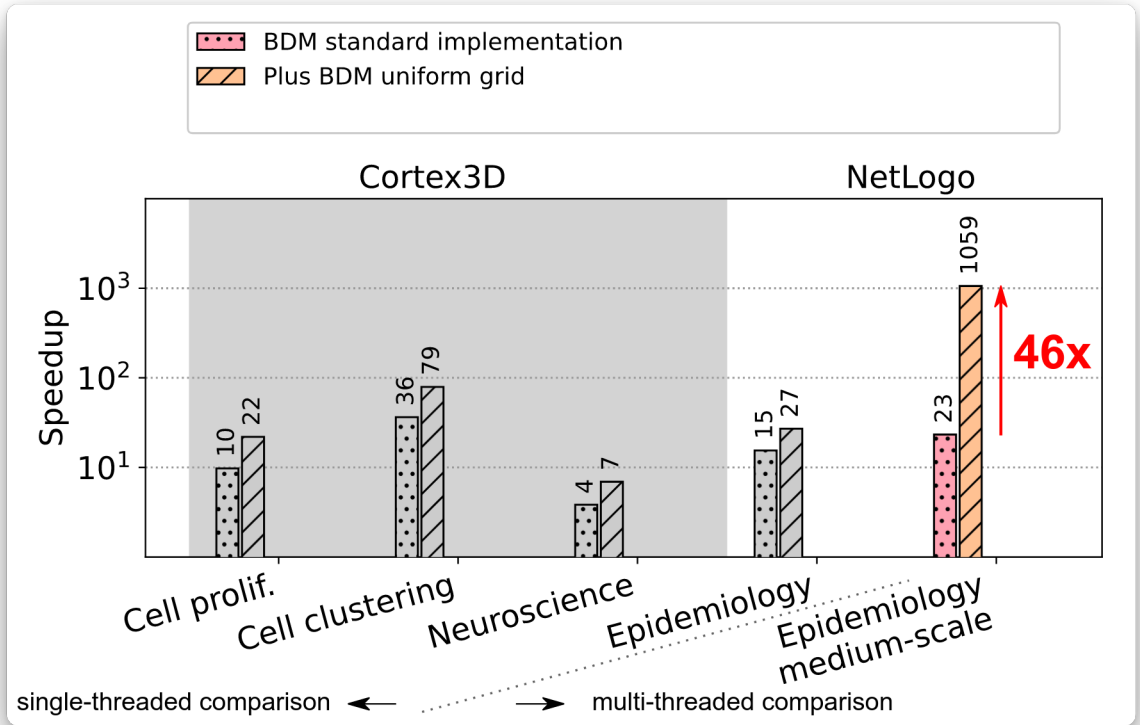
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



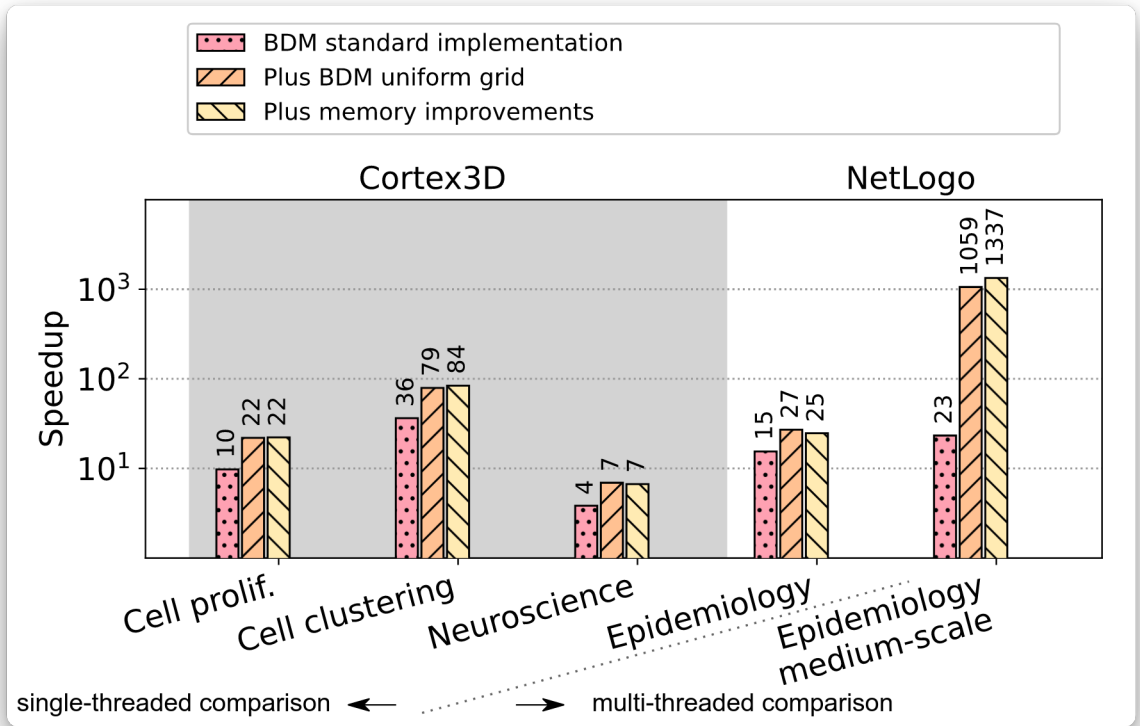
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



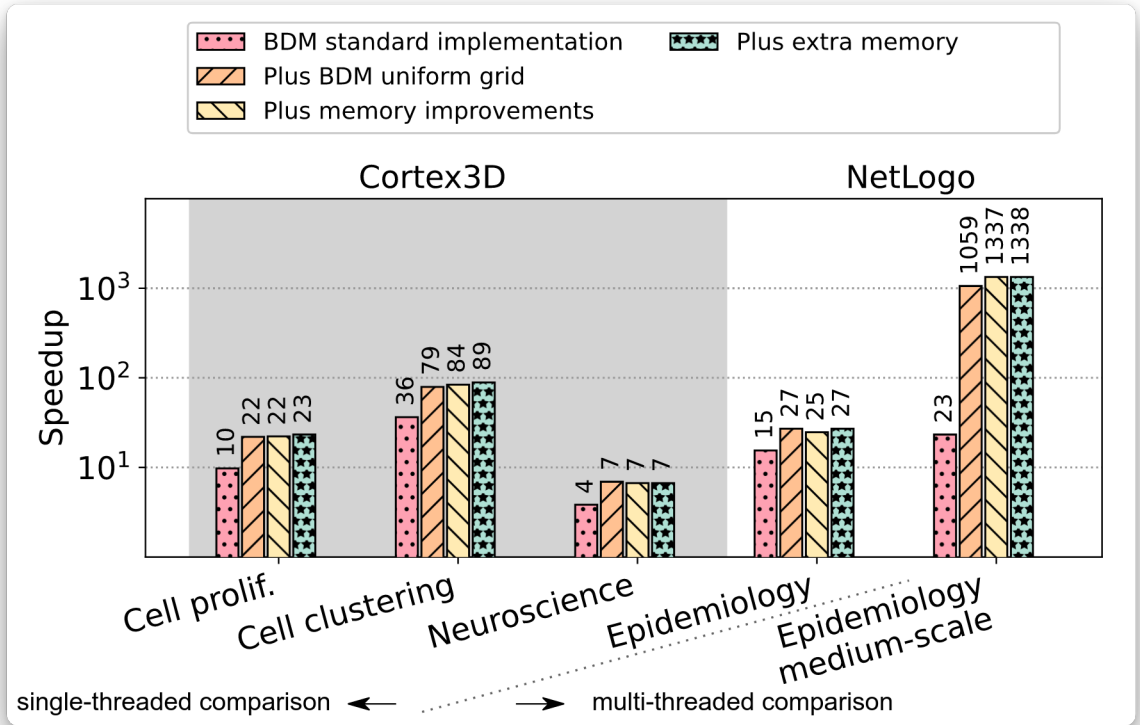
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



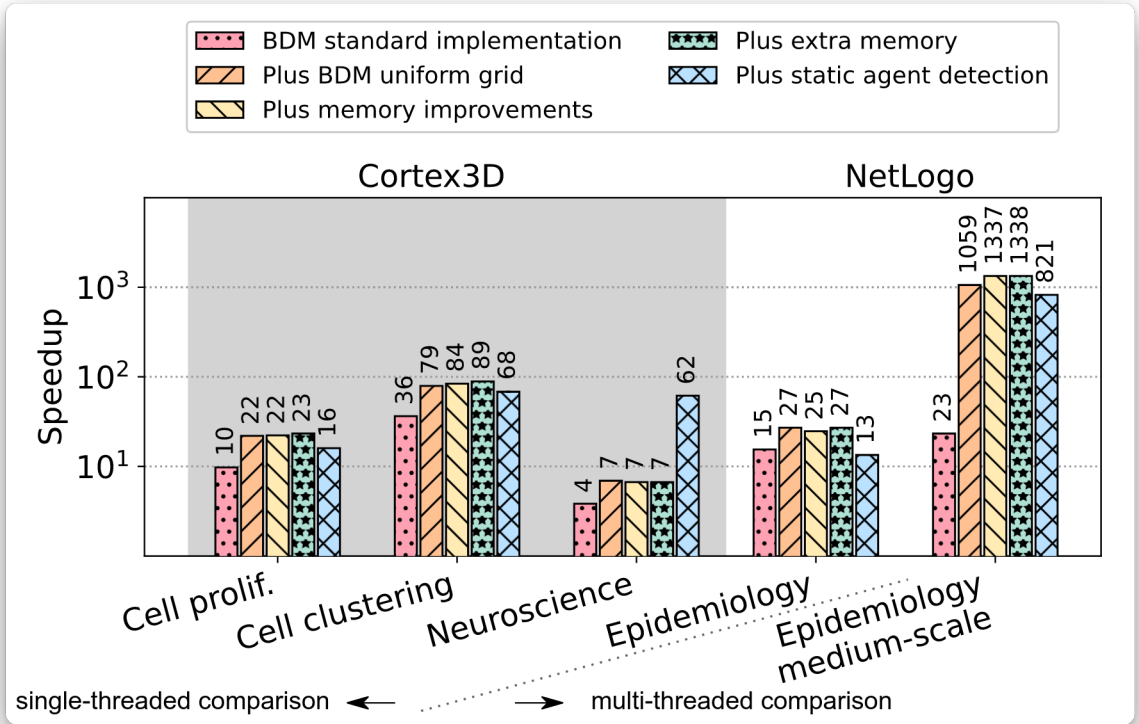
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



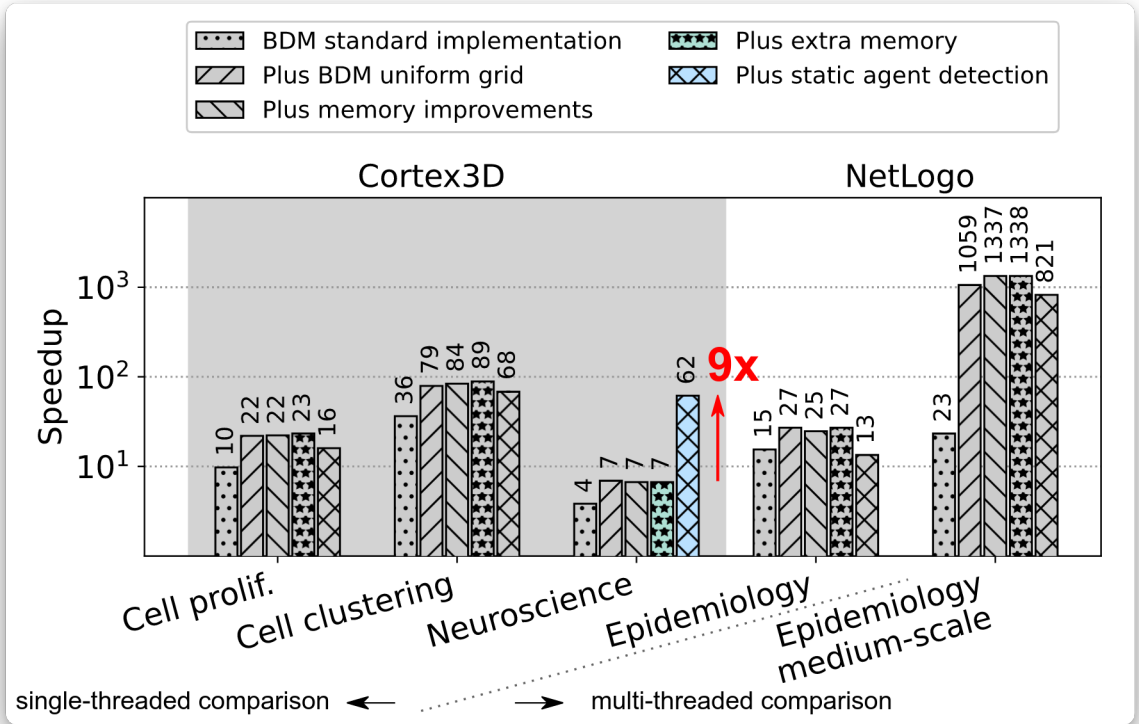
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



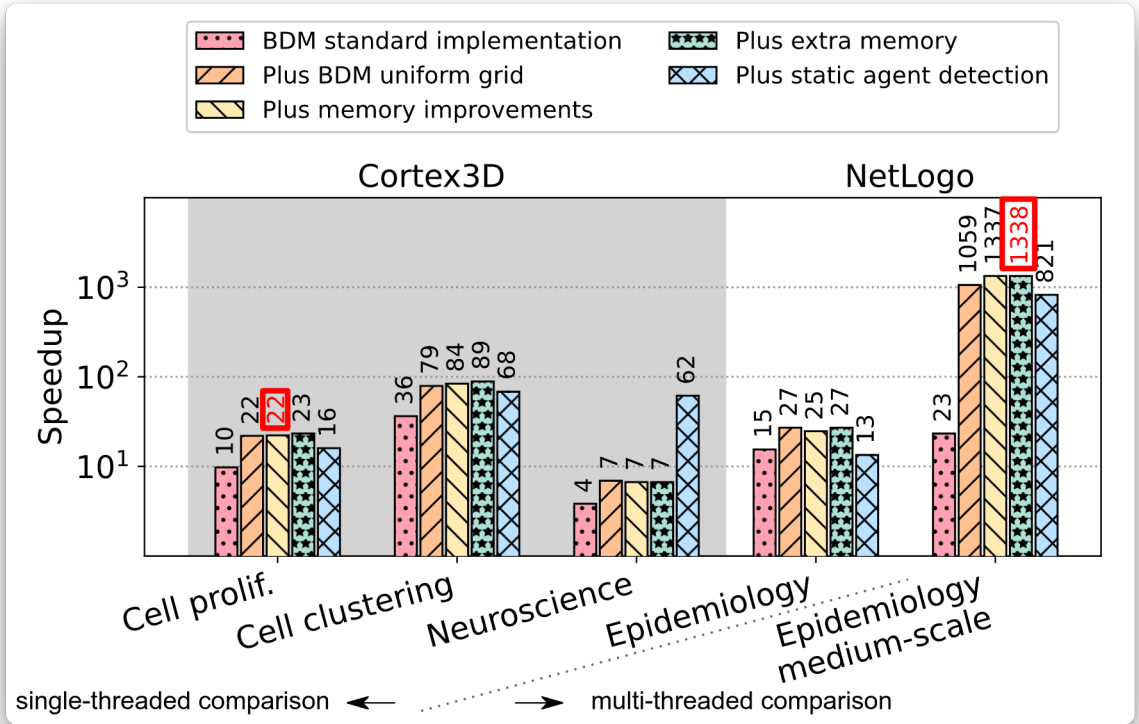
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



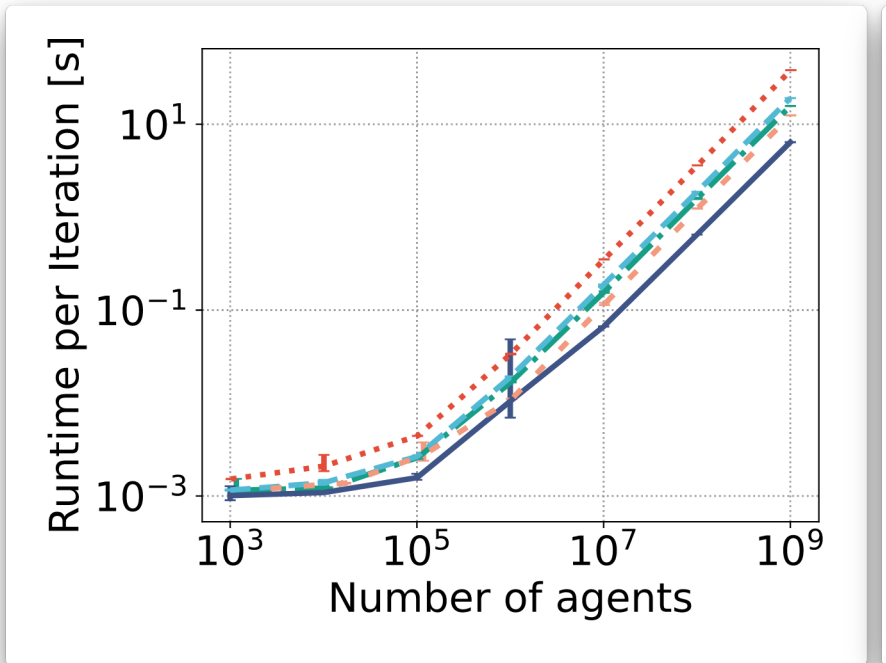
BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo

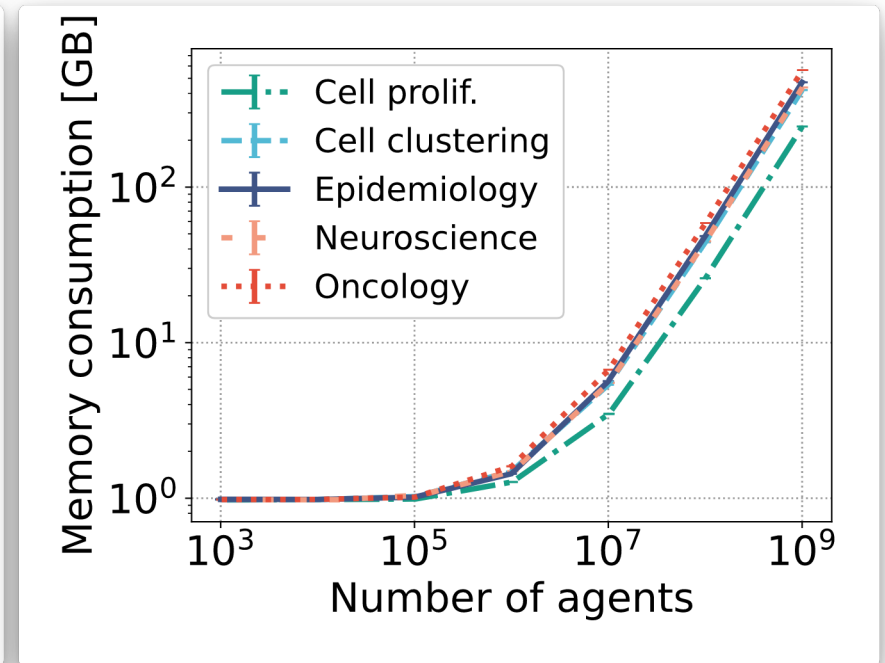


BioDynaMo is **1000×** **faster** than Cortex3D and NetLogo



BioDynaMo's runtime and memory consumption increases **linearly** with model size





More in the paper

https://arxiv.org/abs/2301.06984, https://doi.org/10.1145/3572848.3577480

High-Performance and Scalable Agent-Based Simulation with BioDynaMo

Lukas Breitwieser*
CERN, Switzerland
ETH Zurich, Switzerland

Ahmad Hesam
Delft University of Technology,
The Netherlands

Fons Rademakers
CERN, Switzerland

Juan Gómez Luna
ETH Zurich, Switzerland

Onur Mutlu†
ETH Zurich, Switzerland

Abstract

Agent-based modeling plays an essential role in gaining insights into biology, sociology, economics, and other fields. However, many existing agent-based simulation platforms are not suitable for large-scale studies due to the low performance of the underlying simulation engines. To overcome this limitation, we present a novel high-performance simulation engine.

We identify three key challenges for which we present the following solutions. First, to maximize parallelization, we present an optimized grid to search for neighbors and parallelize the merging of thread-local results. Second, we reduce

1 Introduction

Agent-based modeling (ABM) allows to simulate complex dynamics in a wide range of research fields. ABM has been used to answer research questions in biology [30, 43, 73], sociology [18], economics [65], technology [50], business [56], and more fields [39]. *Agents* are individual entities that, among others, can represent subcellular structures to simulate the growth of a neuron, a cell to investigate cancer development, or a person to simulate the spread of infectious diseases [10]. The actions of an agent are defined through instances of class *behavior*. To stay with the examples from before, possible behaviors are neurite bifurcation, uncontrolled cell division,

Supplementary Materials

Provides **additional evaluations** and all files required to **reproduce** the results (PDF):

- https://doi.org/10.5281/zenodo.6463816
- https://github.com/CMU-SAFARI/BioDynaMo

Supplementary Materials

Provides **additional evaluations** and all files required to **reproduce** the results ( ):

- https://doi.org/10.5281/zenodo.6463816
- https://github.com/CMU-SAFARI/BioDynaMo

```
# extract the provided code archive
mkdir reproduce
tar -xzf <path>/SF2-code.tar.gz -C reproduce
cd reproduce/bdm-paper-examples

# load the provided self-contained docker image
docker/load.sh <path>/SF3-bdm-publication-image.tar.gz

# execute the scripts
docker/run.sh ./run-main.sh
```

Supplementary Materials

Provides **additional evaluations** and all files required to **reproduce** the results ( ):

- https://doi.org/10.5281/zenodo.6463816
- https://github.com/CMU-SAFARI/BioDynaMo

```
# extract the provided code archive
mkdir reproduce
tar -xzf <path>/SF2-code.tar.gz -C reproduce
cd reproduce/bdm-paper-examples

# load the provided self-contained docker image
docker/load.sh <path>/SF3-bdm-publication-image.tar.gz

# execute the scripts
docker/run.sh ./run-main.sh
```

Supplementary Materials

Provides **additional evaluations** and all files required to **reproduce** the results (PDF):

- https://doi.org/10.5281/zenodo.6463816
- https://github.com/CMU-SAFARI/BioDynaMo

```
# extract the provided code archive
mkdir reproduce
tar -xzf <path>/SF2-code.tar.gz -C reproduce
cd reproduce/bdm-paper-examples

# load the provided self-contained docker image
docker/load.sh <path>/SF3-bdm-publication-image.tar.gz

# execute the scripts
docker/run.sh ./run-main.sh
```

Supplementary Materials

Provides **additional evaluations** and all files required to **reproduce** the results (PDF):

- https://doi.org/10.5281/zenodo.6463816
- https://github.com/CMU-SAFARI/BioDynaMo

```
# extract the provided code archive
mkdir reproduce
tar -xzf <path>/SF2-code.tar.gz -C reproduce
cd reproduce/bdm-paper-examples

# load the provided self-contained docker image
docker/load.sh <path>/SF3-bdm-publication-image.tar.gz

# execute the scripts
docker/run.sh ./run-main.sh
```

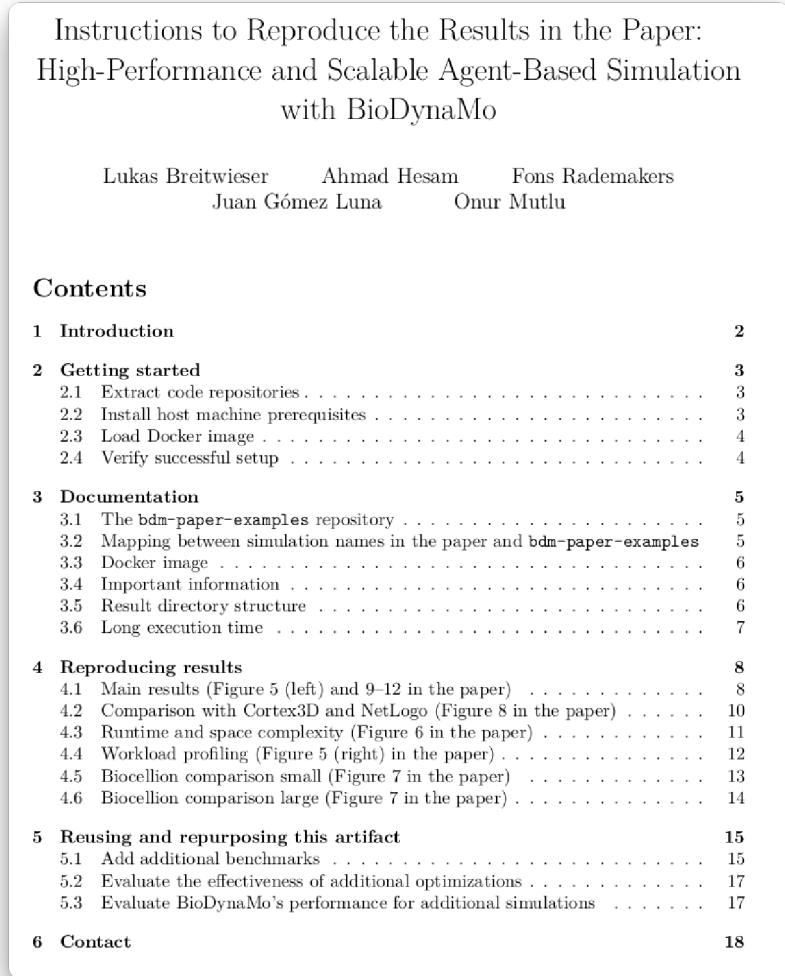
Supplementary Materials

Provides **additional evaluations** and all files required to **reproduce** the results (PDF):

- https://doi.org/10.5281/zenodo.6463816
- https://github.com/CMU-SAFARI/BioDynaMo

```
# extract the provided code archive
mkdir reproduce
tar -xzf <path>/SF2-code.tar.gz -C reproduce
cd reproduce/bdm-paper-examples

# load the provided self-contained docker image
docker/load.sh <path>/SF3-bdm-publication-image.tar.gz

# execute the scripts
docker/run.sh ./run-main.sh
```

More details in file `SF1-readme.pdf` →



Instructions to Reproduce the Results in the Paper: High-Performance and Scalable Agent-Based Simulation with BioDynaMo

Lukas Breitwieser Ahmad Hesam Fons Rademakers
Juan Gómez Luna Onur Mutlu

Contents

Supplementary Materials

Provides **additional evaluations** and all files required to **reproduce** the results (PDF):

- https://doi.org/10.5281/zenodo.6463816
- https://github.com/CMU-SAFARI/BioDynaMo

```
# extract the provided code archive
mkdir reproduce
tar -xzf <path>/SF2-code.tar.gz -C reproduce
cd reproduce/bdm-paper-examples

# load the provided self-contained docker image
docker/load.sh <path>/SF3-bdm-publication-image.tar.gz

# execute the scripts
docker/run.sh ./run-main.sh
```

More details in file `SF1-readme.pdf` →

Instructions to Reproduce the Results in the Paper: High-Performance and Scalable Agent-Based Simulation with BioDynaMo

Lukas Breitwieser Ahmad Hesam Fons Rademakers
Juan Gómez Luna Onur Mutlu

Contents

Summary

Our optimizations to maximize parallelism, improve the memory layout, and avoid unnecessary work are effective and give BioDynaMo the following performance characteristics.

Summary

Our optimizations to maximize parallelism, improve the memory layout, and avoid unnecessary work are effective and give BioDynaMo the following performance characteristics.

BioDynaMo **scales well** across a large number of CPU cores

Summary

Our optimizations to maximize parallelism, improve the memory layout, and avoid unnecessary work are effective and give BioDynaMo the following performance characteristics.

BioDynaMo **scales well** across a large number of CPU cores

Runtime and memory consumption **increases linearly** with the number of agents

Summary

Our optimizations to maximize parallelism, improve the memory layout, and avoid unnecessary work are effective and give BioDynaMo the following performance characteristics.

BioDynaMo **scales well** across a large number of CPU cores

Runtime and memory consumption **increases linearly** with the number of agents

BioDynaMo is more than **1000× faster** than Cortex3D and NetLogo and up to **9× more efficient** than Biocellion

Summary

Our optimizations to maximize parallelism, improve the memory layout, and avoid unnecessary work are effective and give BioDynaMo the following performance characteristics.

BioDynaMo **scales well** across a large number of CPU cores

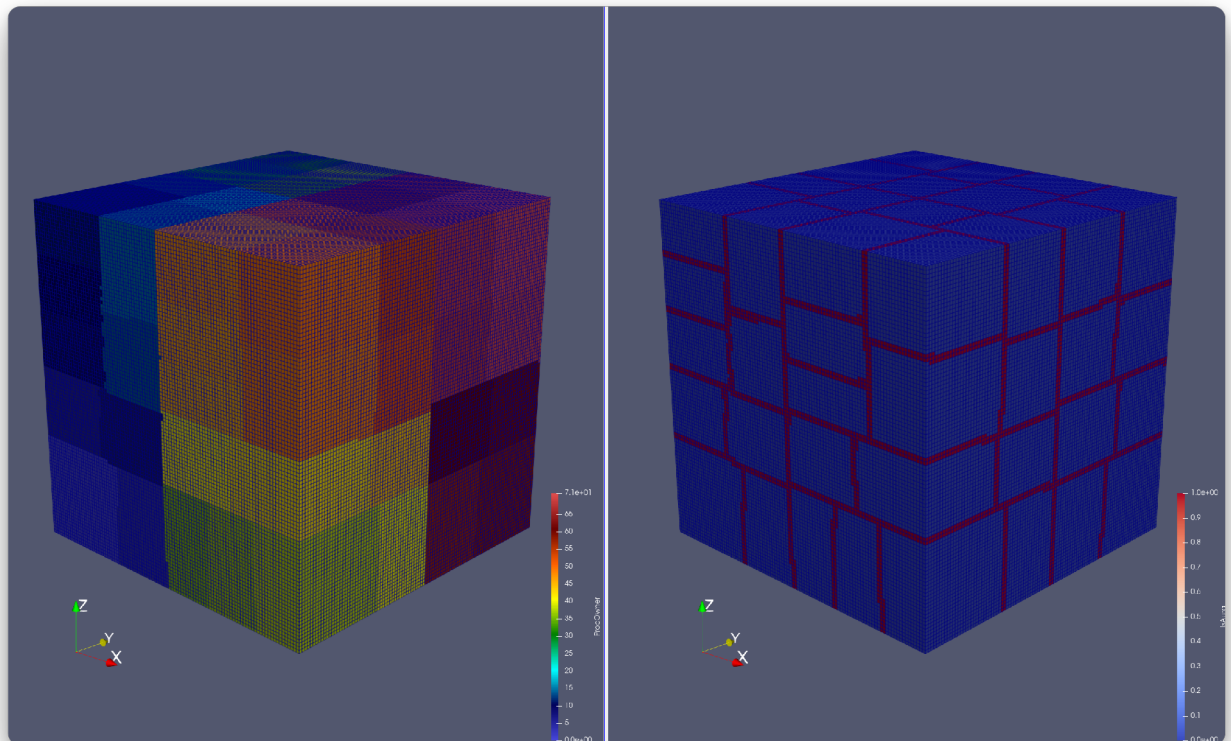
Runtime and memory consumption **increases linearly** with the number of agents

BioDynaMo is more than **1000× faster** than Cortex3D and NetLogo and up to **9× more efficient** than Biocellion

These performance characteristics enable **simulations with billions of agents** on a single server.

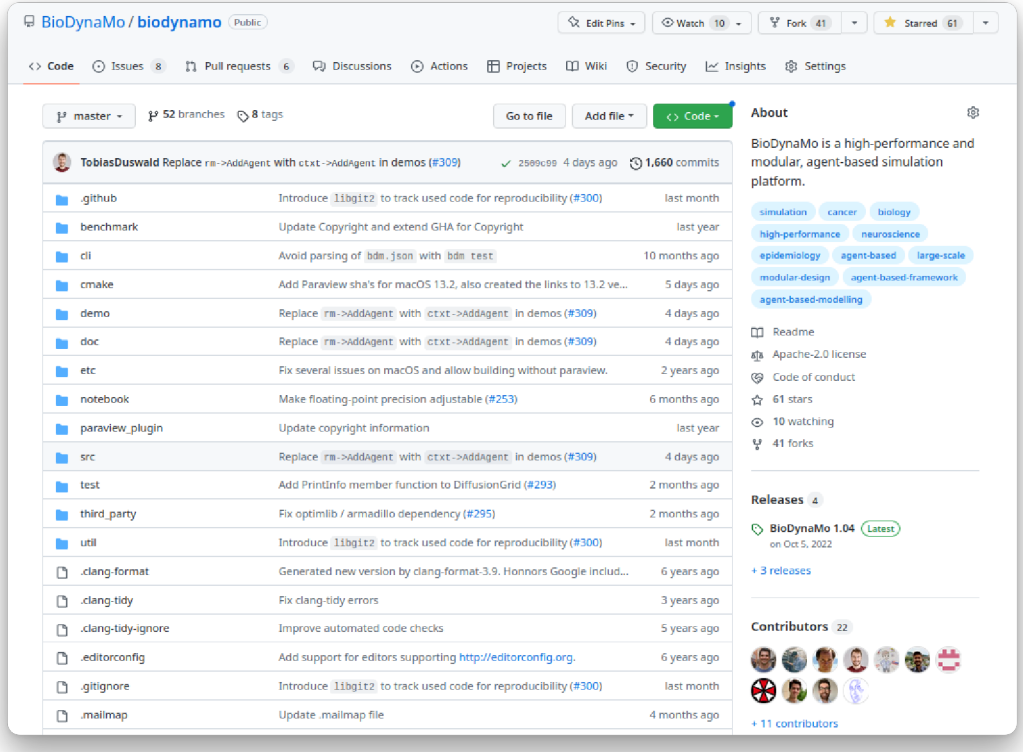
Future work

Distributed simulation engine



BioDynaMo is available under the permissive Apache 2.0 **open source** license

https://github.com/BioDynaMo/biodynamo



Join us in pushing the limits of agent-based modeling

BioDynamo is an **open, welcoming, and collaborative project**: https://biodynamo.org

Besides the organizations behind this paper,

BioDynamo is developed by the **BioDynamo collaboration**,

and further organizations.

More in the paper

https://arxiv.org/abs/2301.06984, https://doi.org/10.1145/3572848.3577480

High-Performance and Scalable Agent-Based Simulation with BioDynaMo

Lukas Breitwieser*
CERN, Switzerland
ETH Zurich, Switzerland

Ahmad Hesam
Delft University of Technology,
The Netherlands

Fons Rademakers
CERN, Switzerland

Juan Gómez Luna
ETH Zurich, Switzerland

Onur Mutlu†
ETH Zurich, Switzerland

Abstract

Agent-based modeling plays an essential role in gaining insights into biology, sociology, economics, and other fields. However, many existing agent-based simulation platforms are not suitable for large-scale studies due to the low performance of the underlying simulation engines. To overcome this limitation, we present a novel high-performance simulation engine.

We identify three key challenges for which we present the following solutions. First, to maximize parallelization, we present an optimized grid to search for neighbors and parallelize the merging of thread-local results. Second, we reduce the memory access latency with a NUMA-aware agent iterator, agent sorting with a space-filling curve, and a custom heap memory allocator. Third, we present a mechanism to omit the collision force calculation under certain conditions.

Our evaluation shows an order of magnitude improvement over Biocellion, three orders of magnitude speedup over Cortex3D and NetLogo, and the ability to simulate 1.72 billion agents on a single server.

Supplementary Materials, including instructions to reproduce the results, are available at: https://doi.org/10.5281/zenodo.6463816

1 Introduction

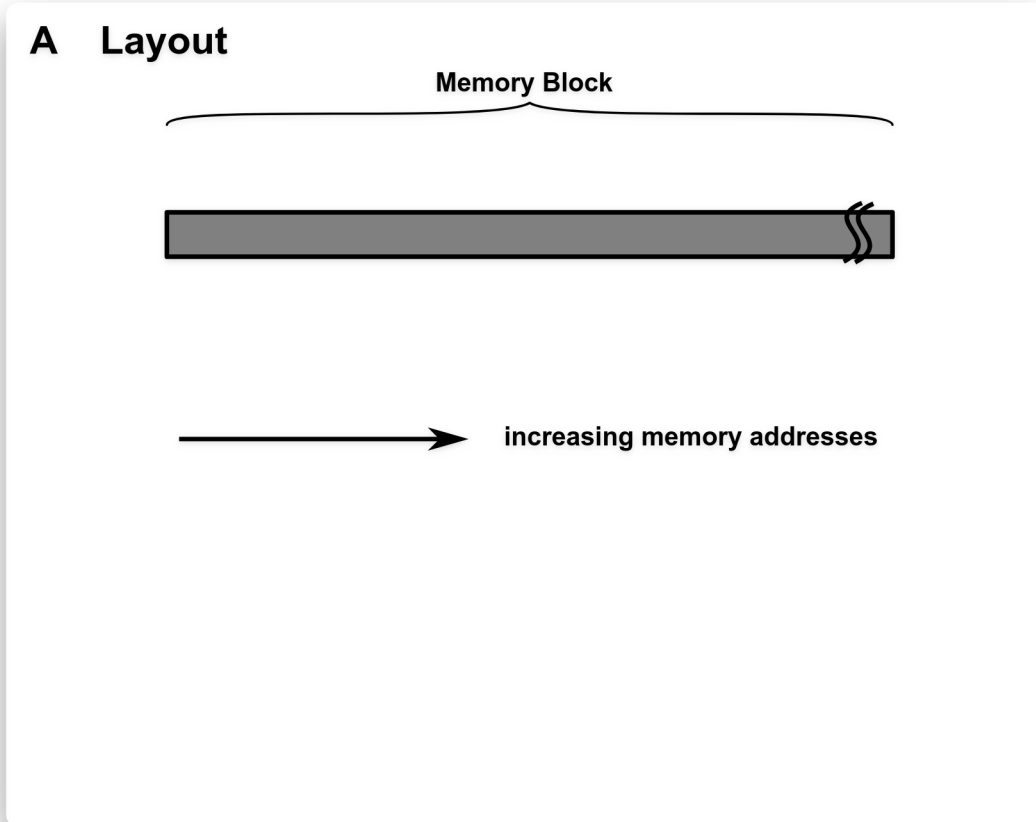
Agent-based modeling (ABM) allows to simulate complex dynamics in a wide range of research fields. ABM has been used to answer research questions in biology [30, 43, 73], sociology [18], economics [65], technology [50], business [56], and more fields [39]. *Agents* are individual entities that, among others, can represent subcellular structures to simulate the growth of a neuron, a cell to investigate cancer development, or a person to simulate the spread of infectious diseases [10]. The actions of an agent are defined through instances of class *behavior*. To stay with the examples from before, possible behaviors are neurite bifurcation, uncontrolled cell division, or infection.

Agent-based models are developed in an iterative way, during which an initial model is increasingly refined until it matches with observed data [58, 67]. Model parameters that cannot be derived from the literature are determined through optimization. An optimization algorithm generates a parameter set, executes the model, and evaluates the error with respect to observed data until the error converges to a local or global minimum. This loop might also contain an uncertainty analysis to evaluate the robustness of a solution [40]. Consequently, the model must be simulated many times.

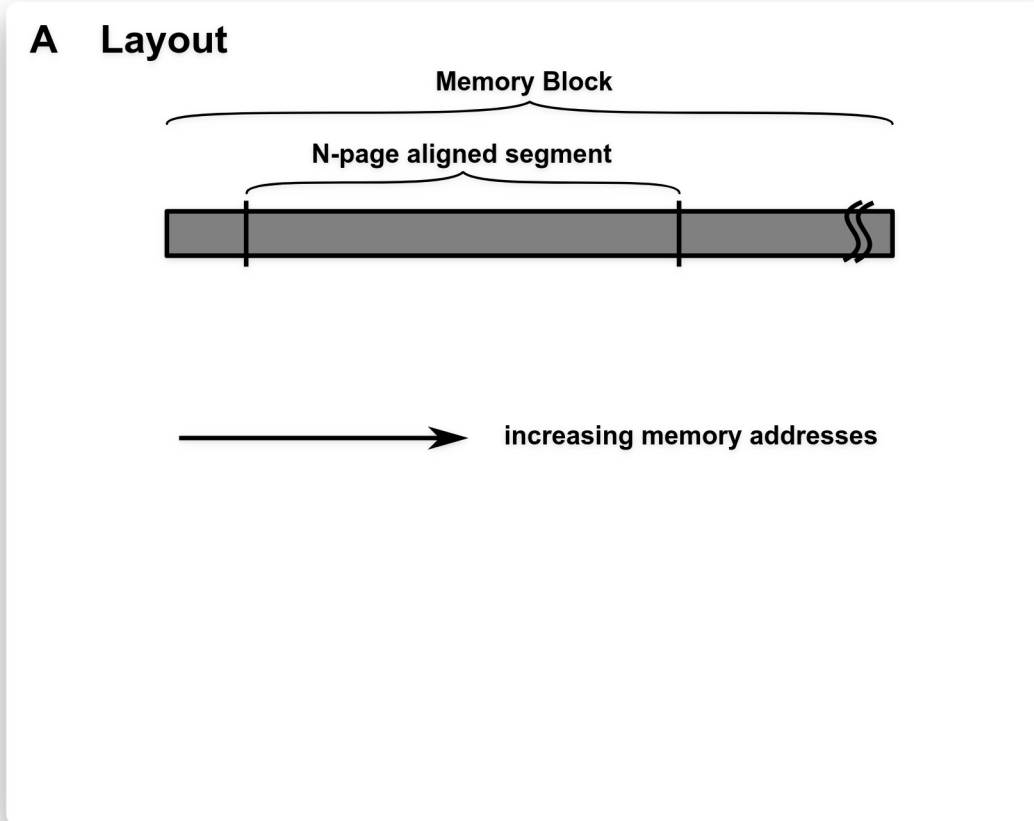
The simulation engine's performance limits the scale of the

Appendix

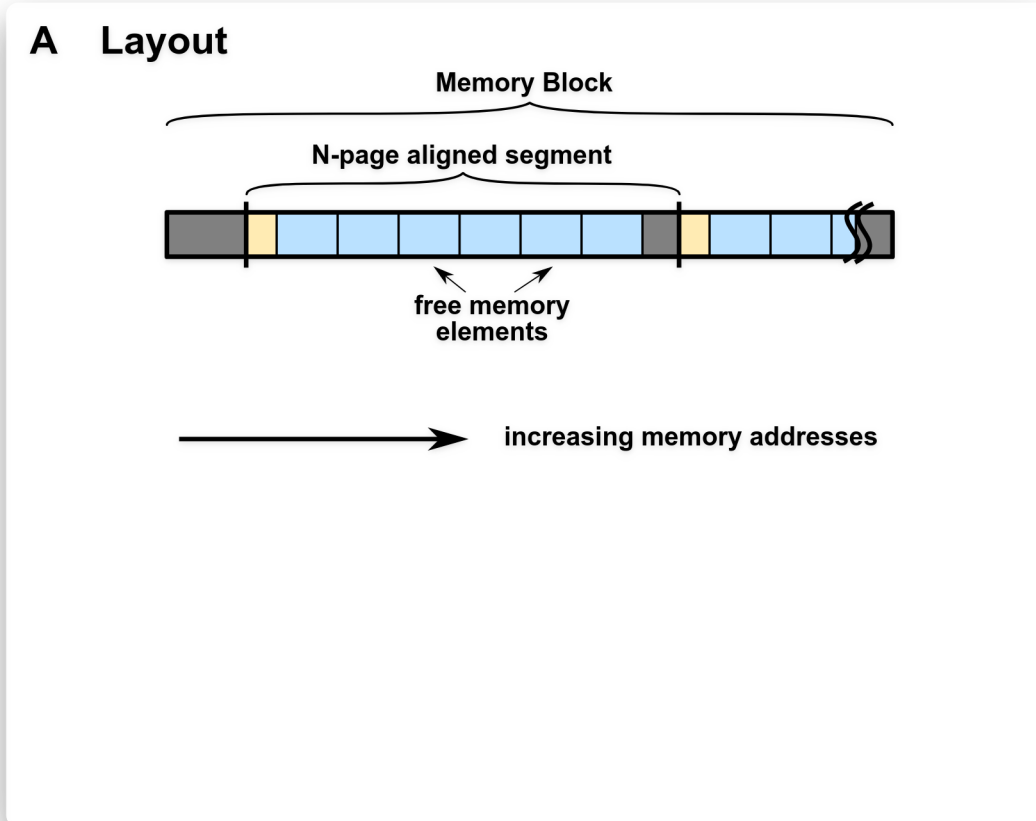
Memory allocation



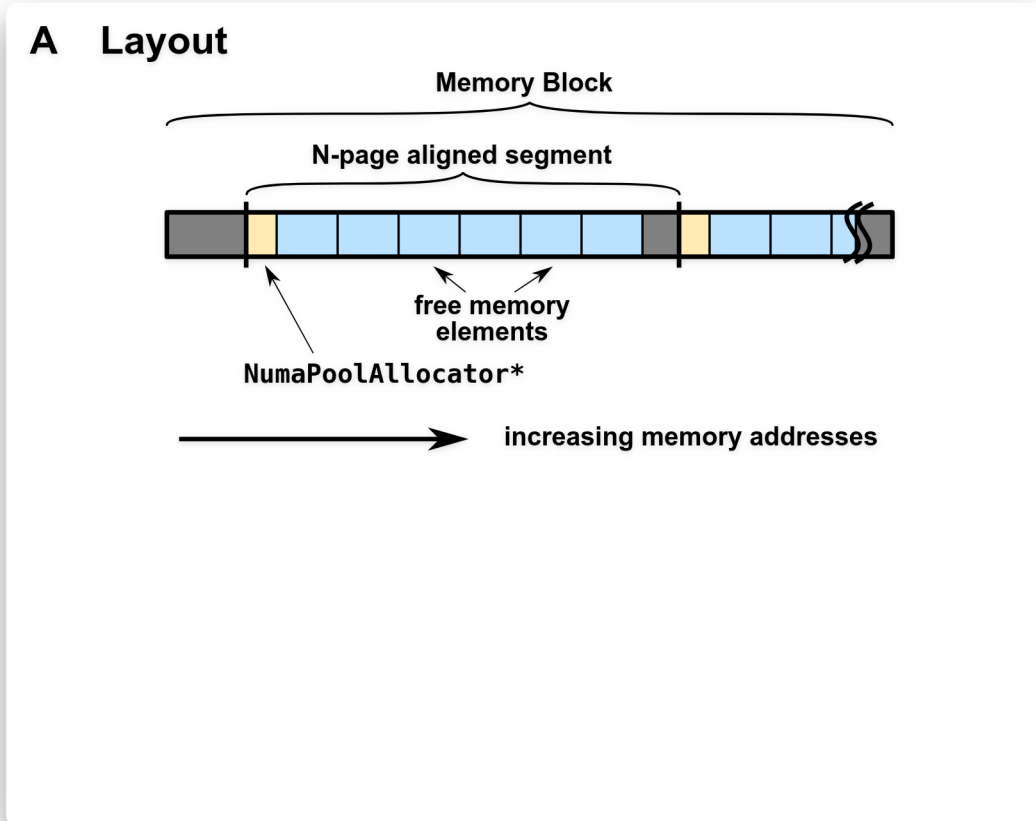
Memory allocation



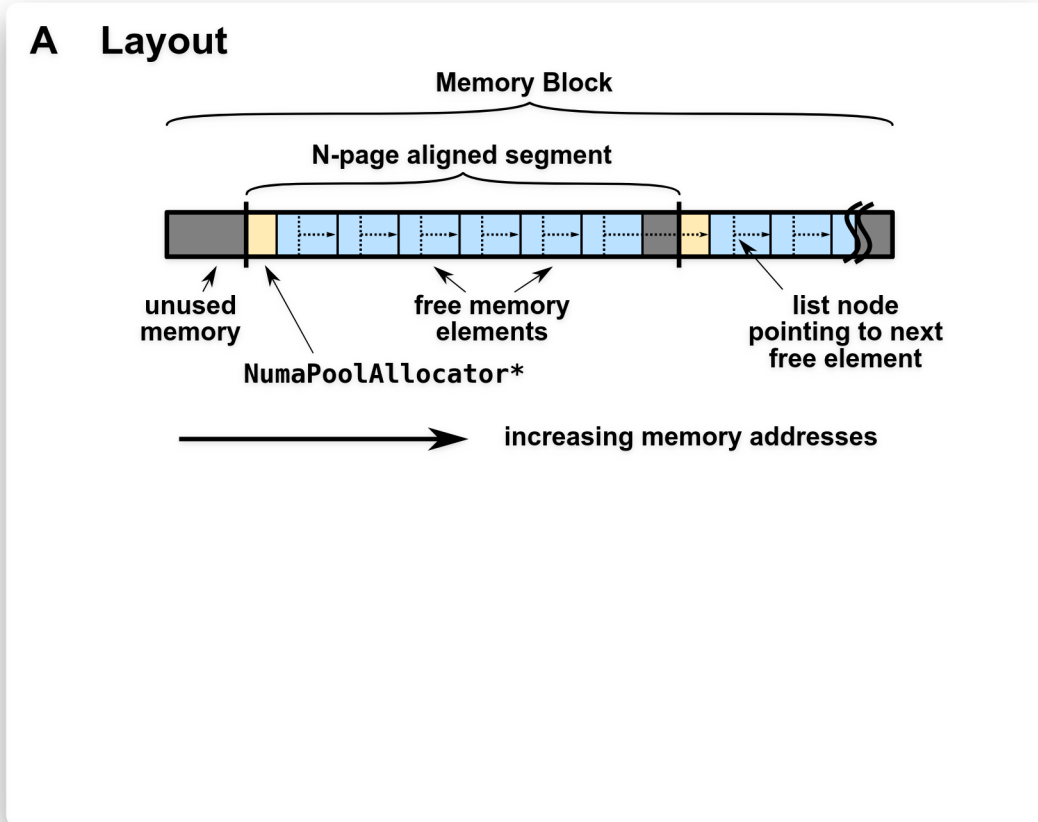
Memory allocation



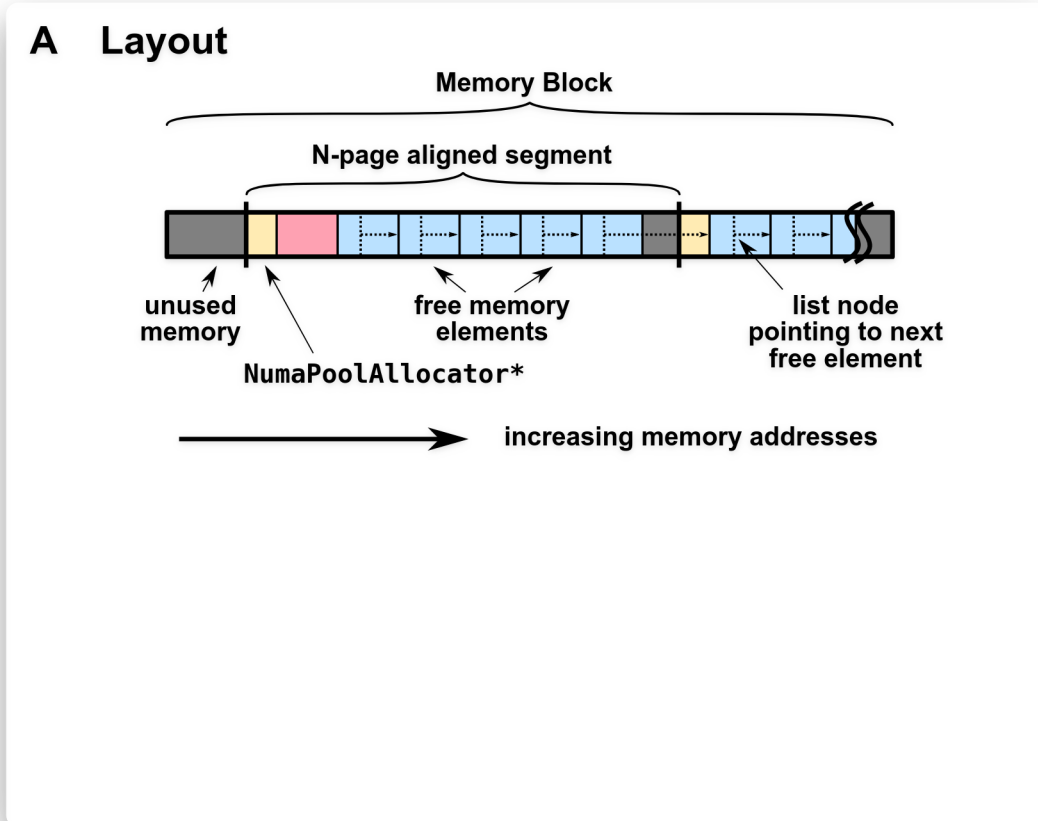
Memory allocation



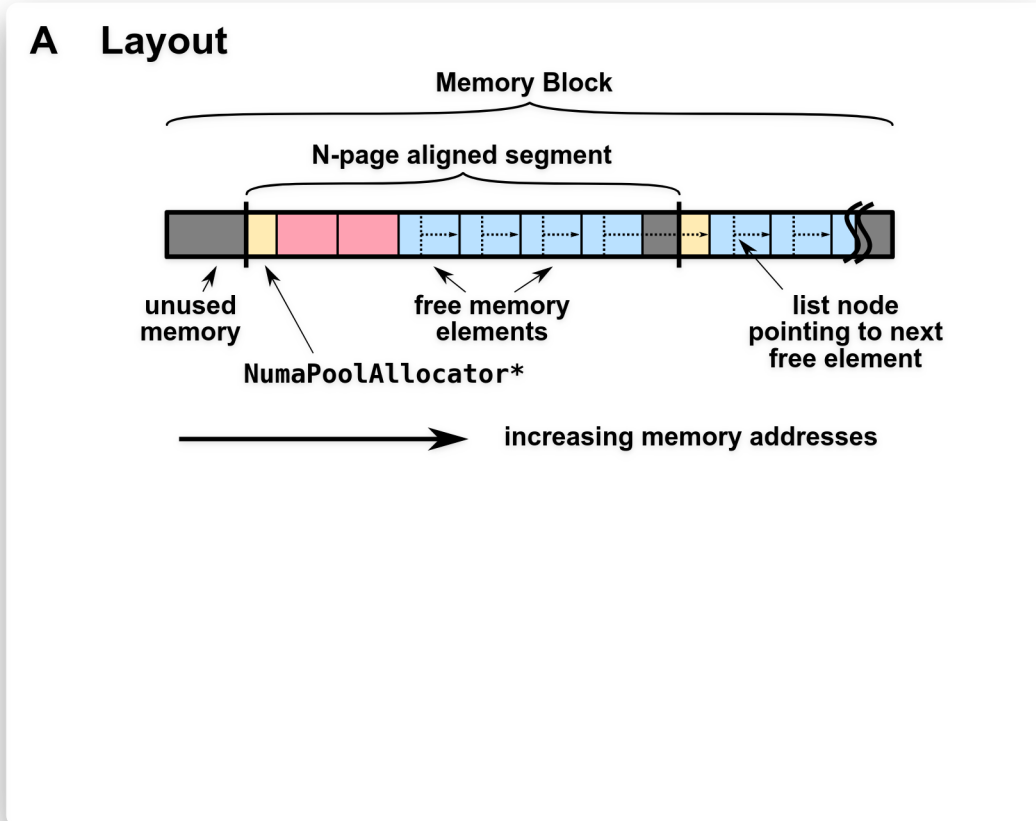
Memory allocation



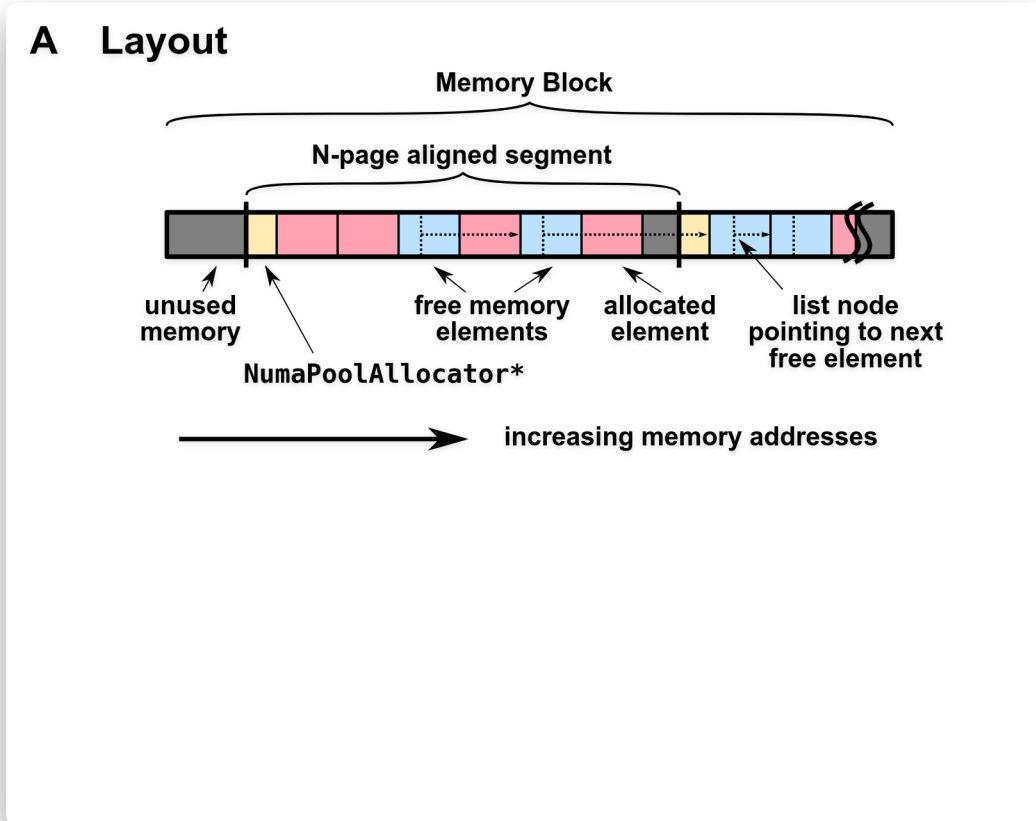
Memory allocation



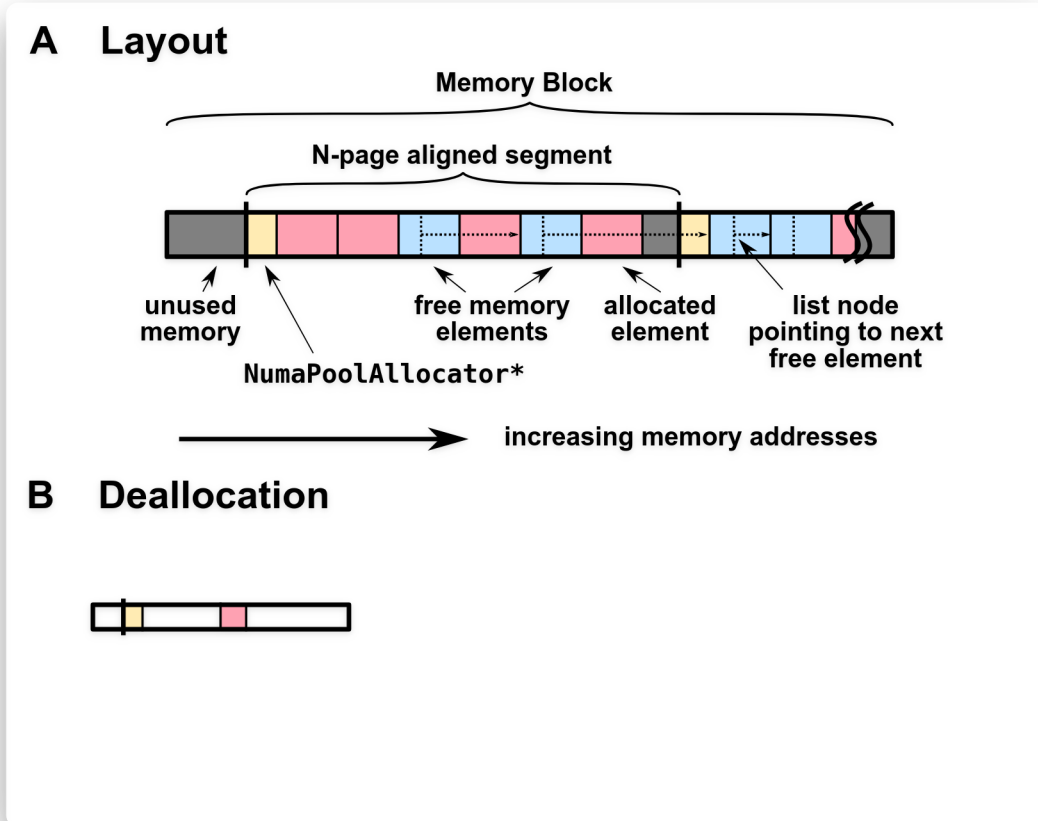
Memory allocation



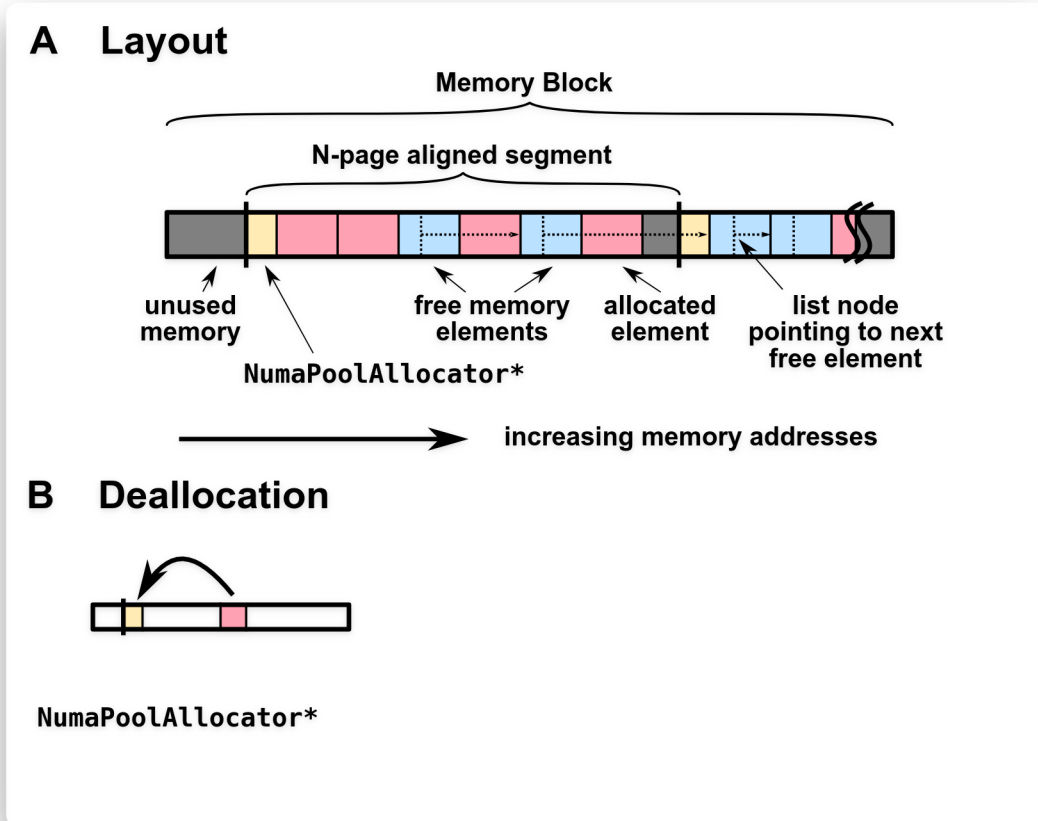
Memory allocation



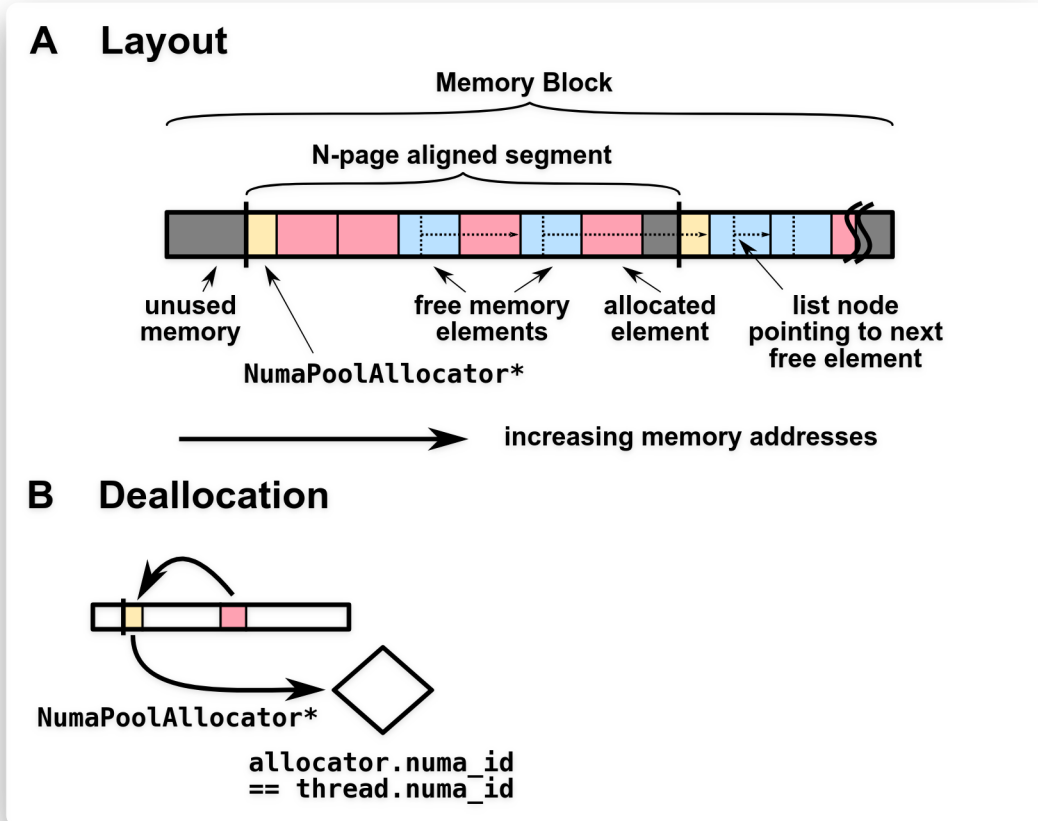
Memory allocation



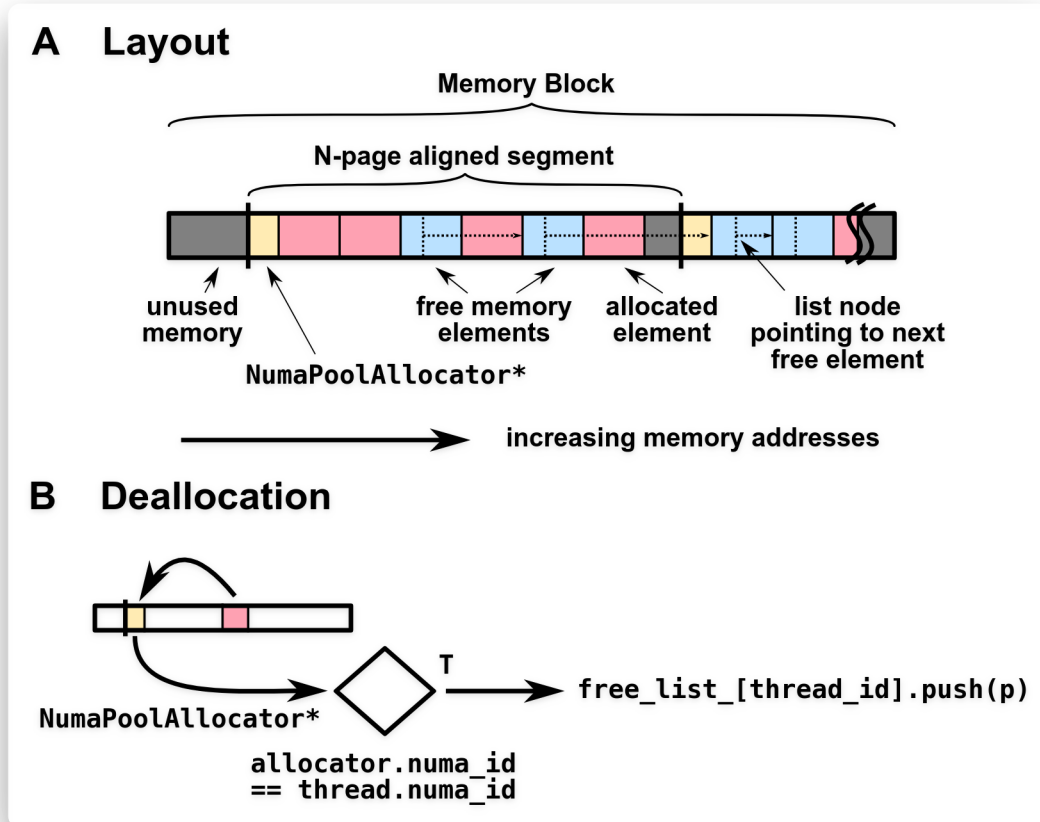
Memory allocation



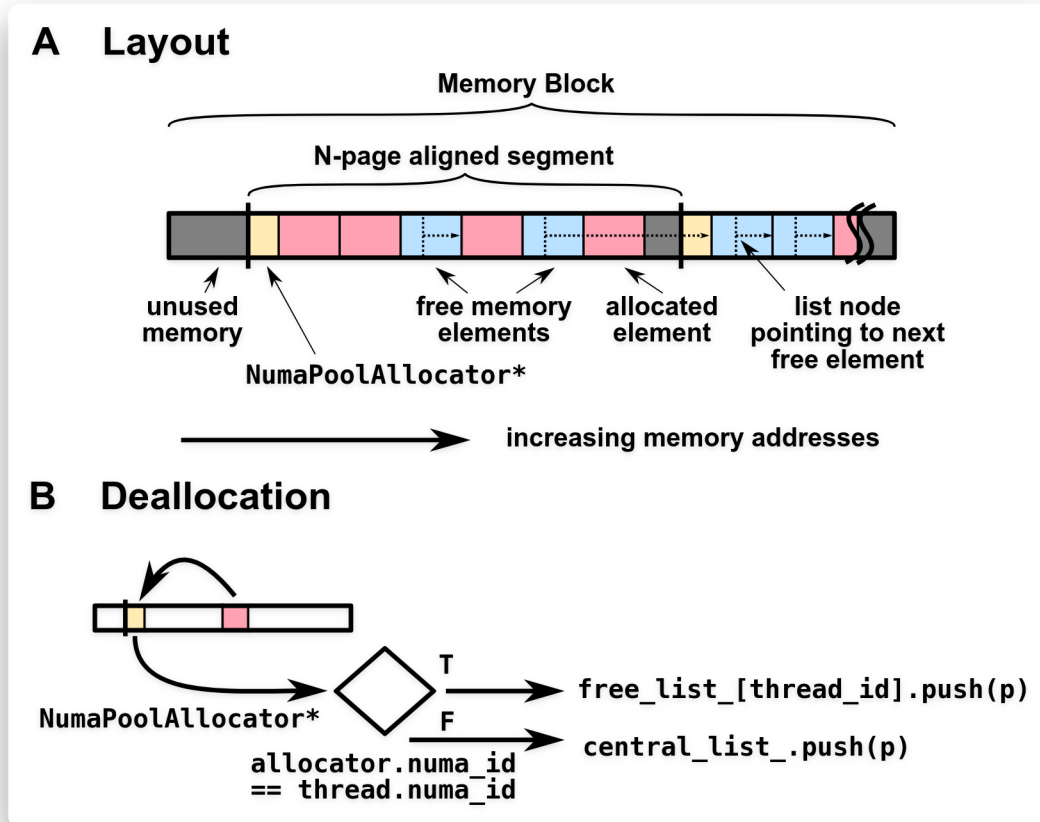
Memory allocation



Memory allocation

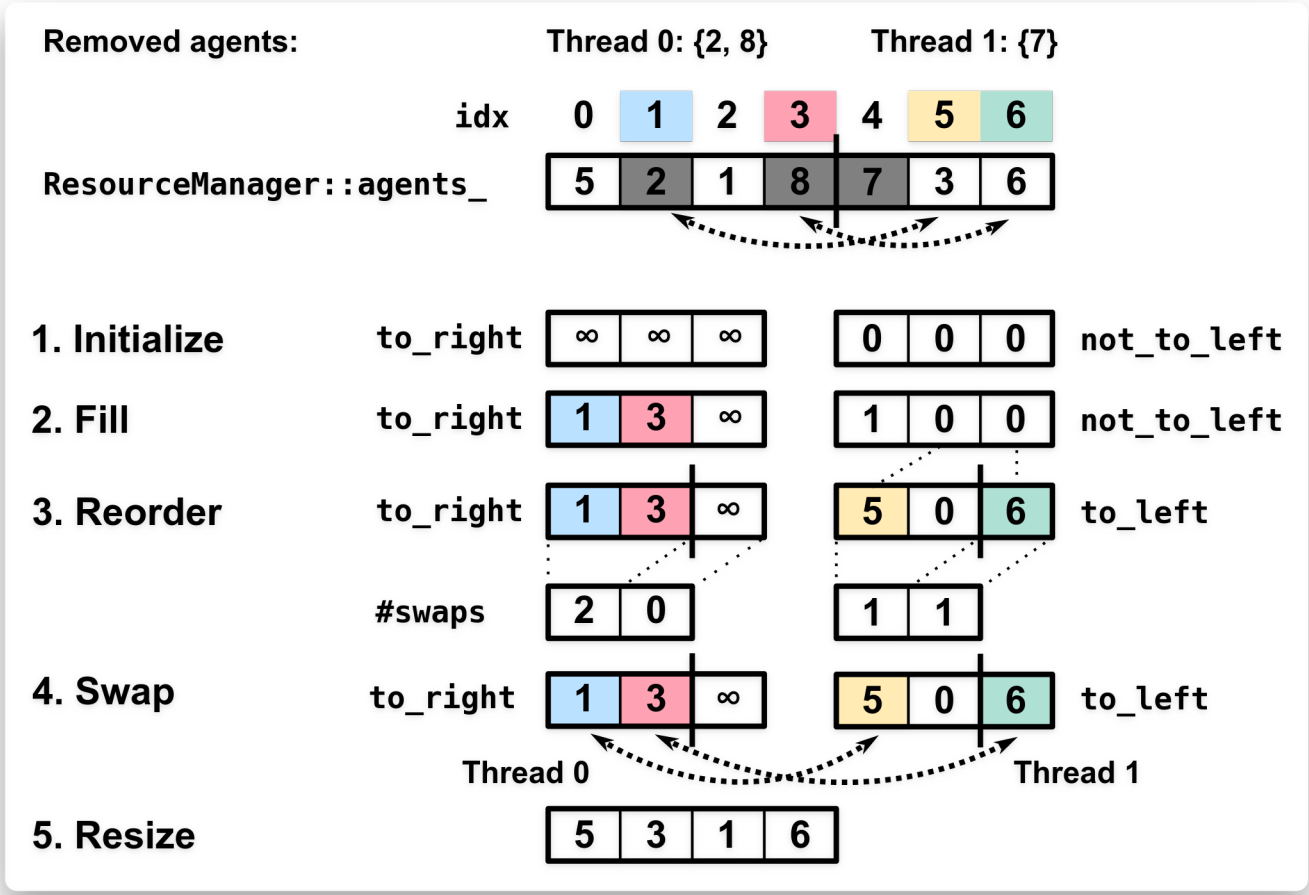


Memory allocation



Memory allocation

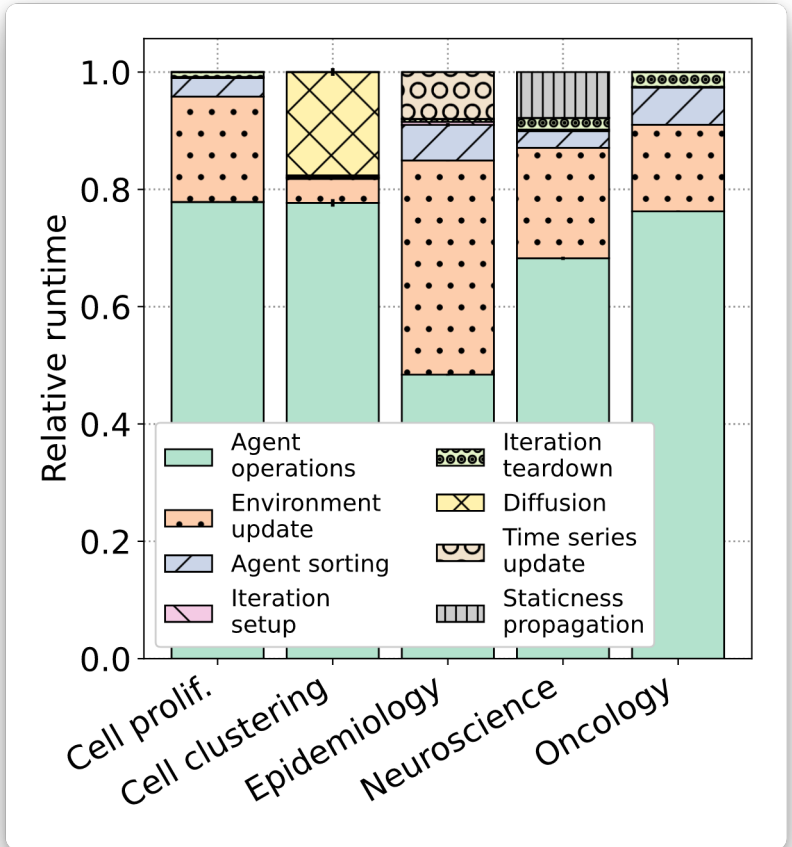
Parallel agent removal



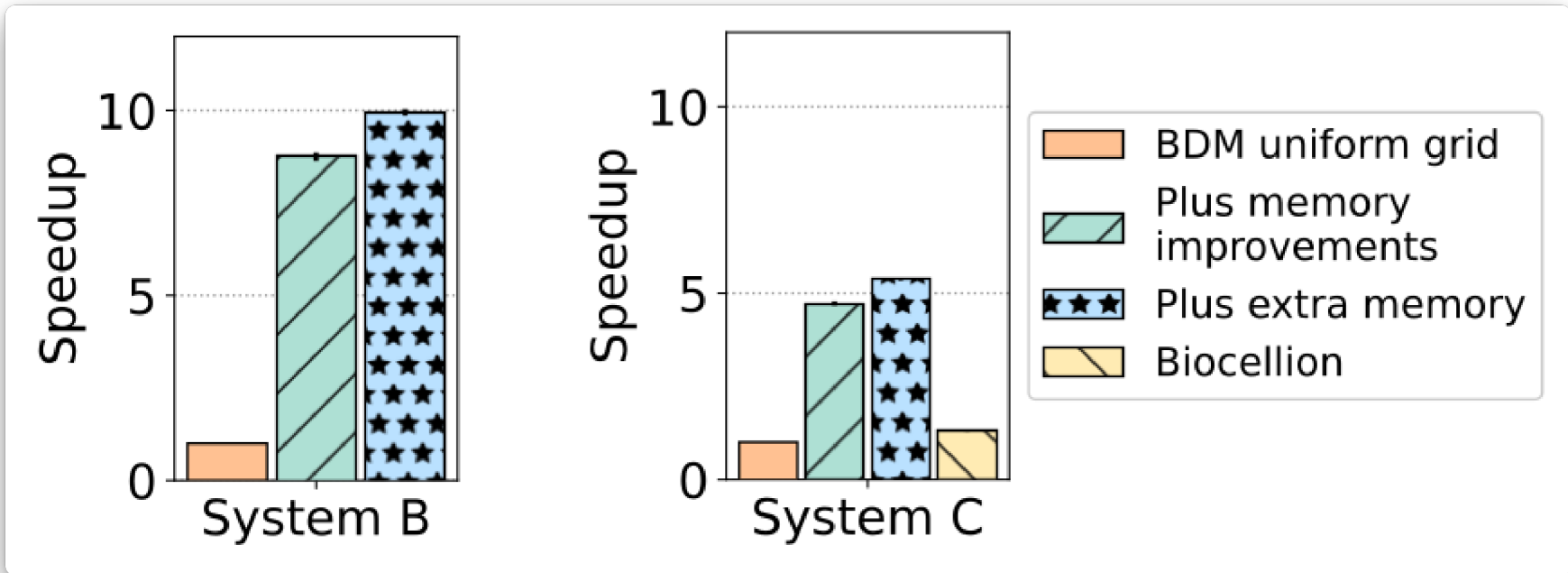
Performance-related simulation characteristics

Characteristic	Cell proliferation	Cell clustering	Epidemiology	Neuroscience	Oncology
Create new agents during simulation	✗			✗	✗
Delete agents during simulation					✗
Agents modify neighbors				✗	
Load imbalance			✗	✗	
Agents move randomly			✗		✗
Simulation uses diffusion		✗		✗	
Simulation has static regions				✗	
Number of iterations	500	1000	1000	500	288
Number of agents (in millions)	12.6	2	10	9	10
Number of diffusion volumes	0	54m	0	65k	0

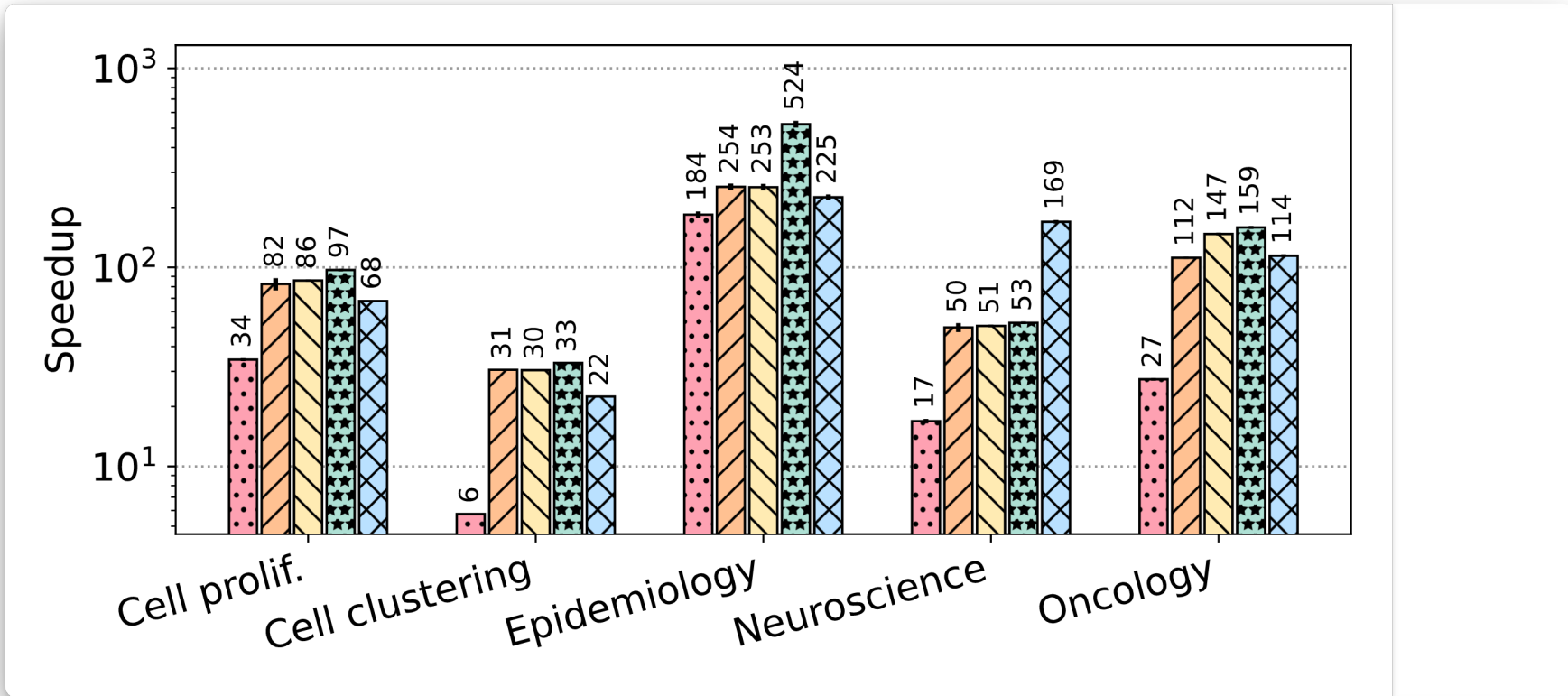
Operation runtime breakdown



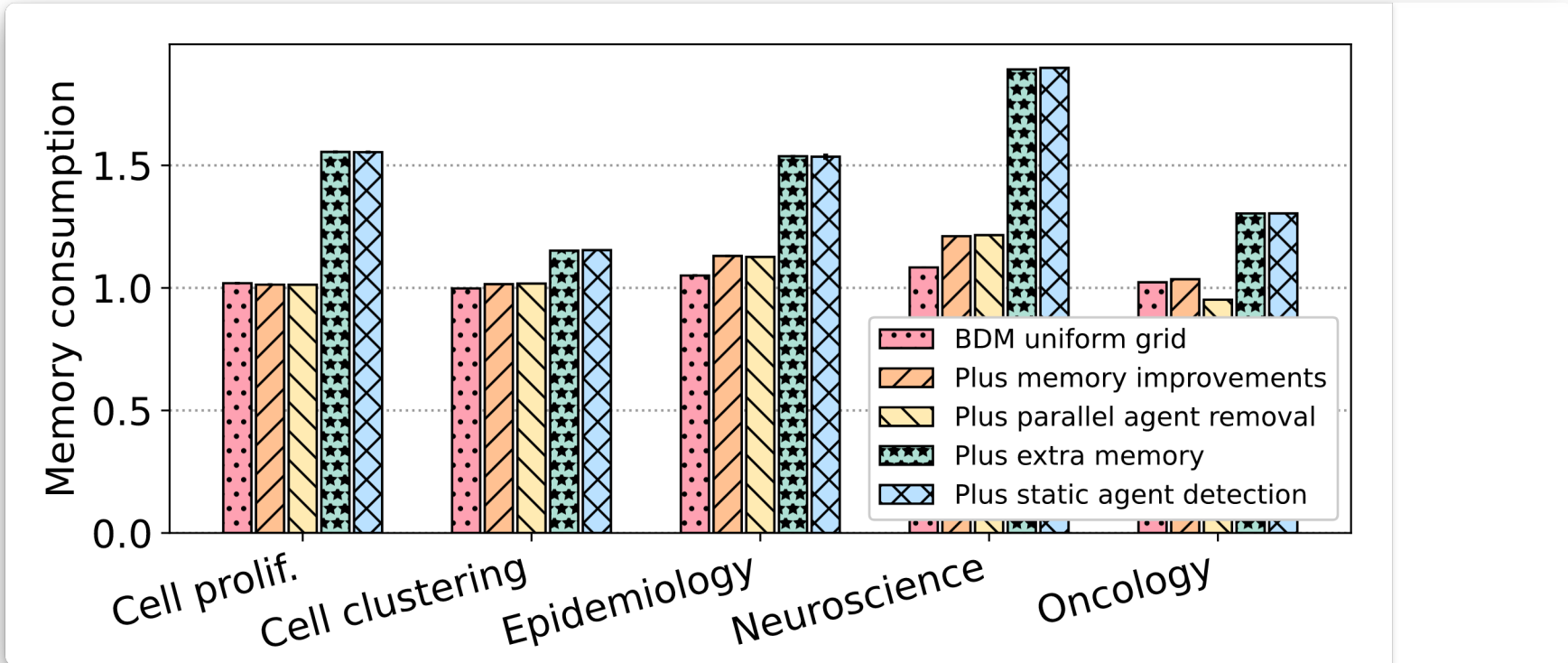
BioDynaMo / Biocellion comparison analysis



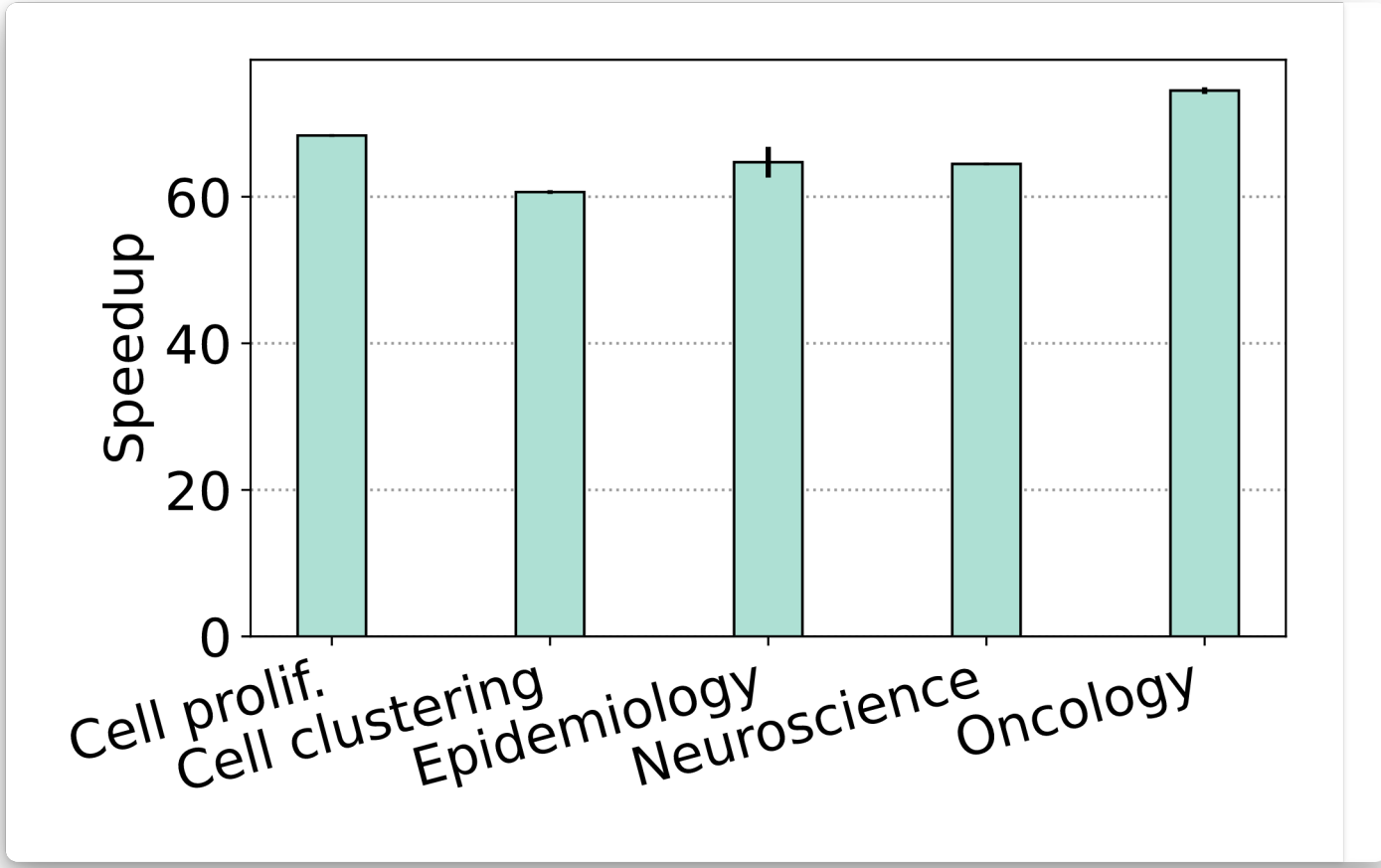
Optimization overview for larger scale simulations



Optimization overview for larger scale simulations

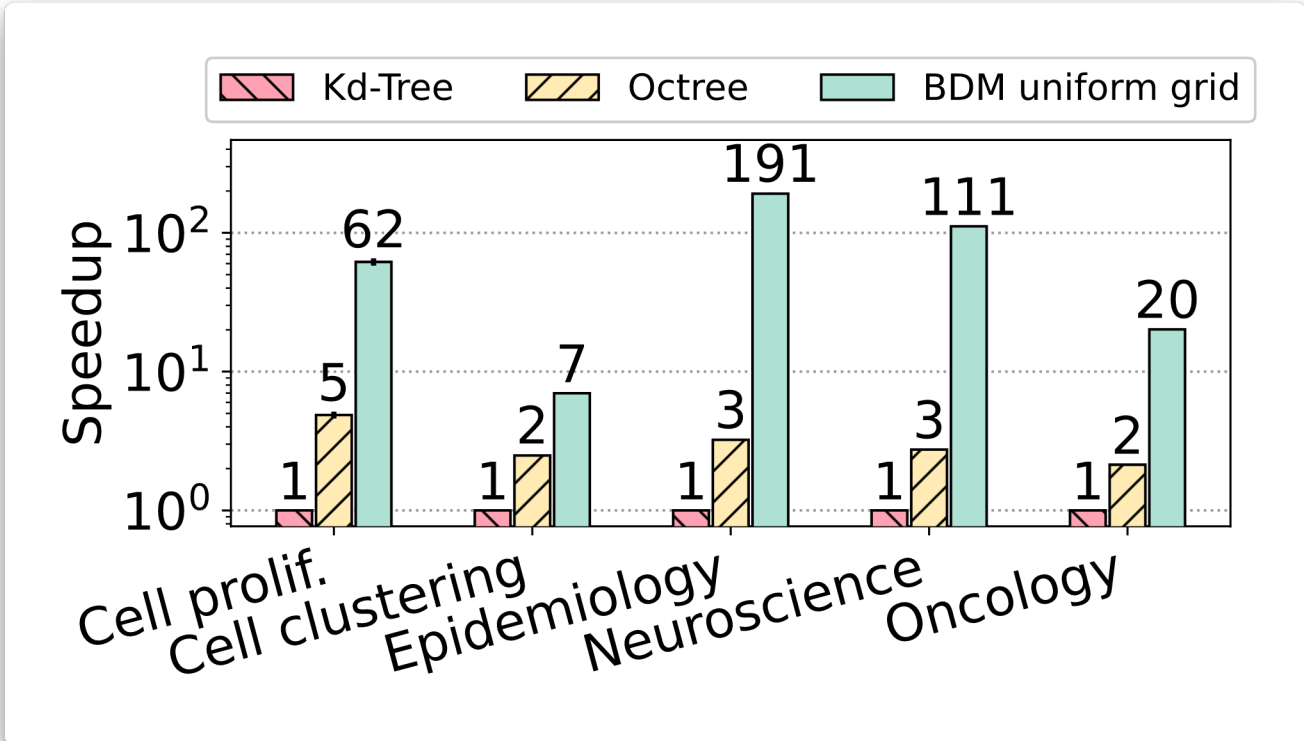


Full simulation scalability



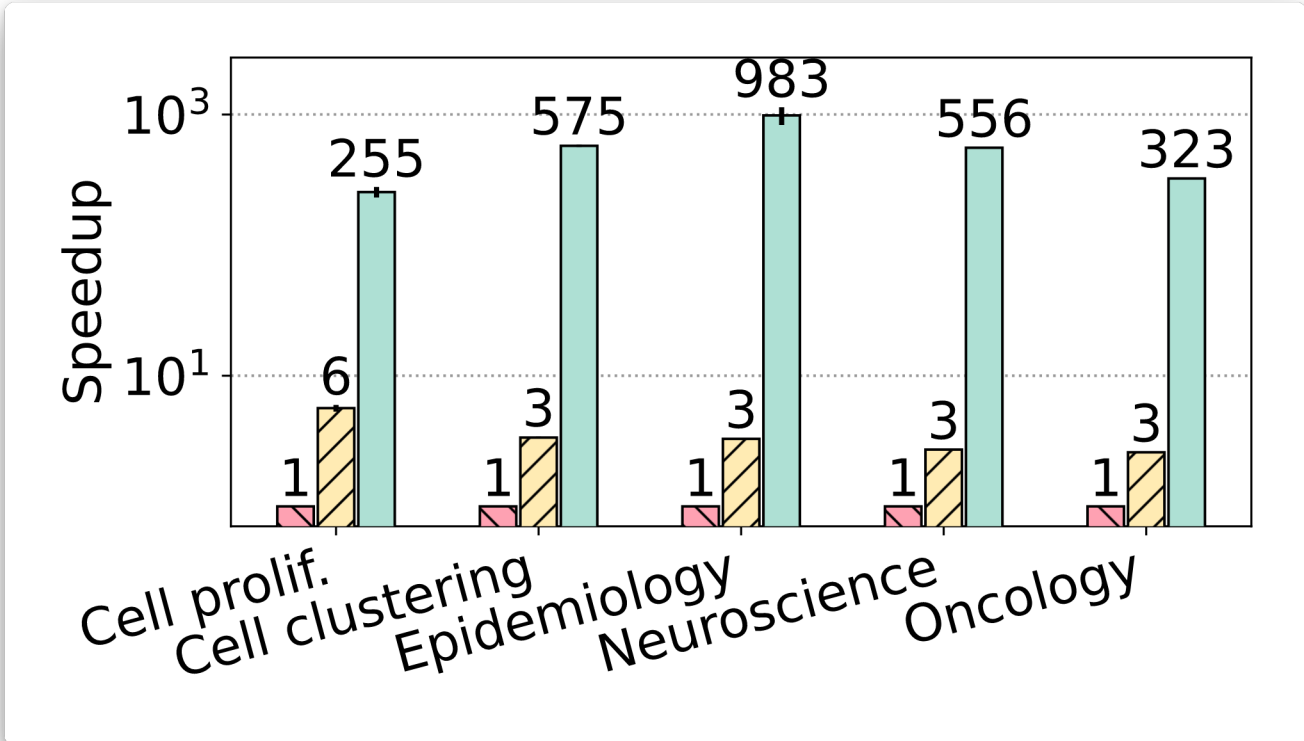
Radial neighbor search comparison

Whole simulation (144 threads and 4 NUMA domains)



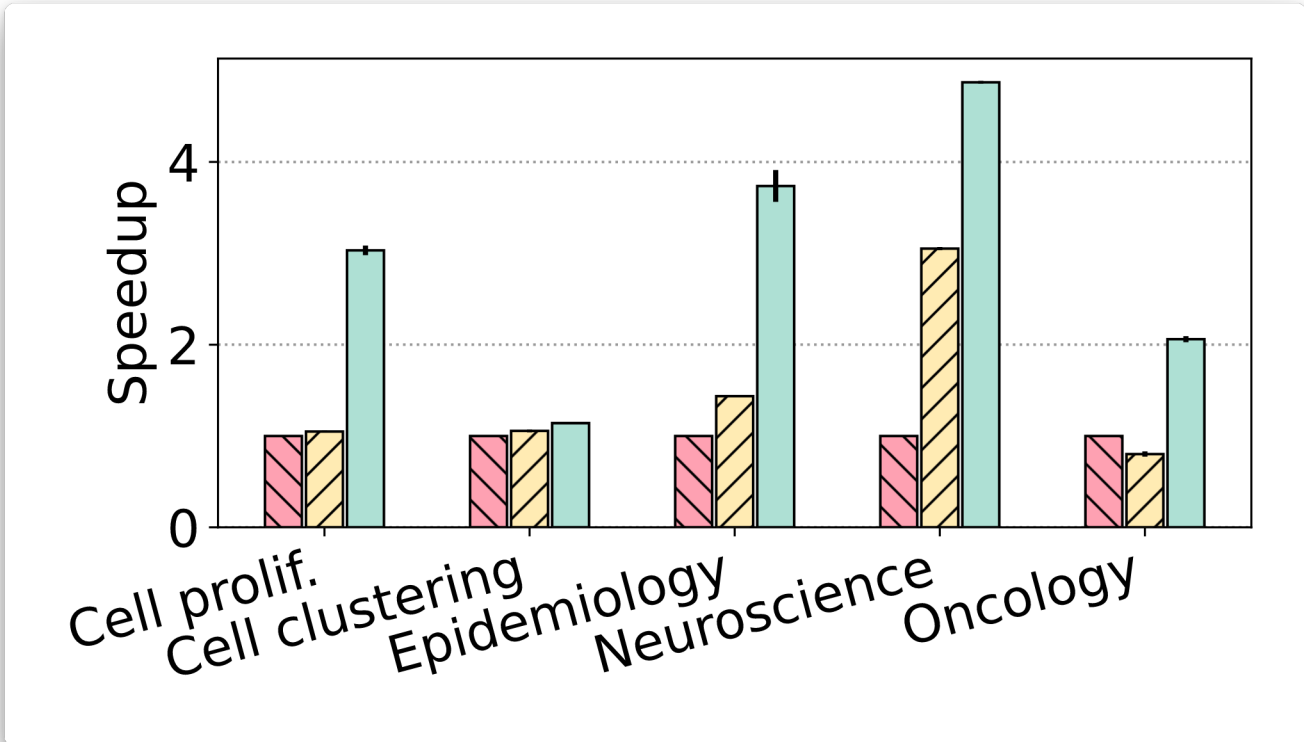
Radial neighbor search comparison

Build time (144 threads and 4 NUMA domains)



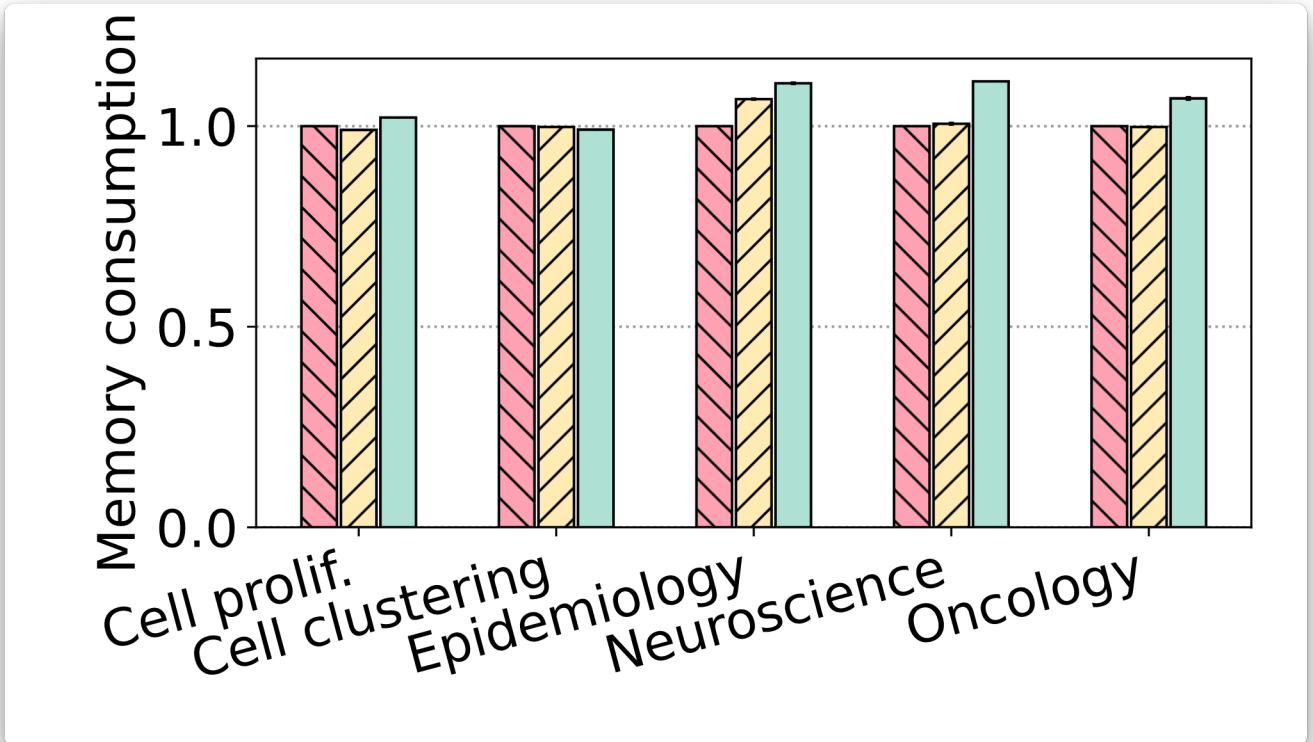
Radial neighbor search comparison

Search time indirect (144 threads and 4 NUMA domains)



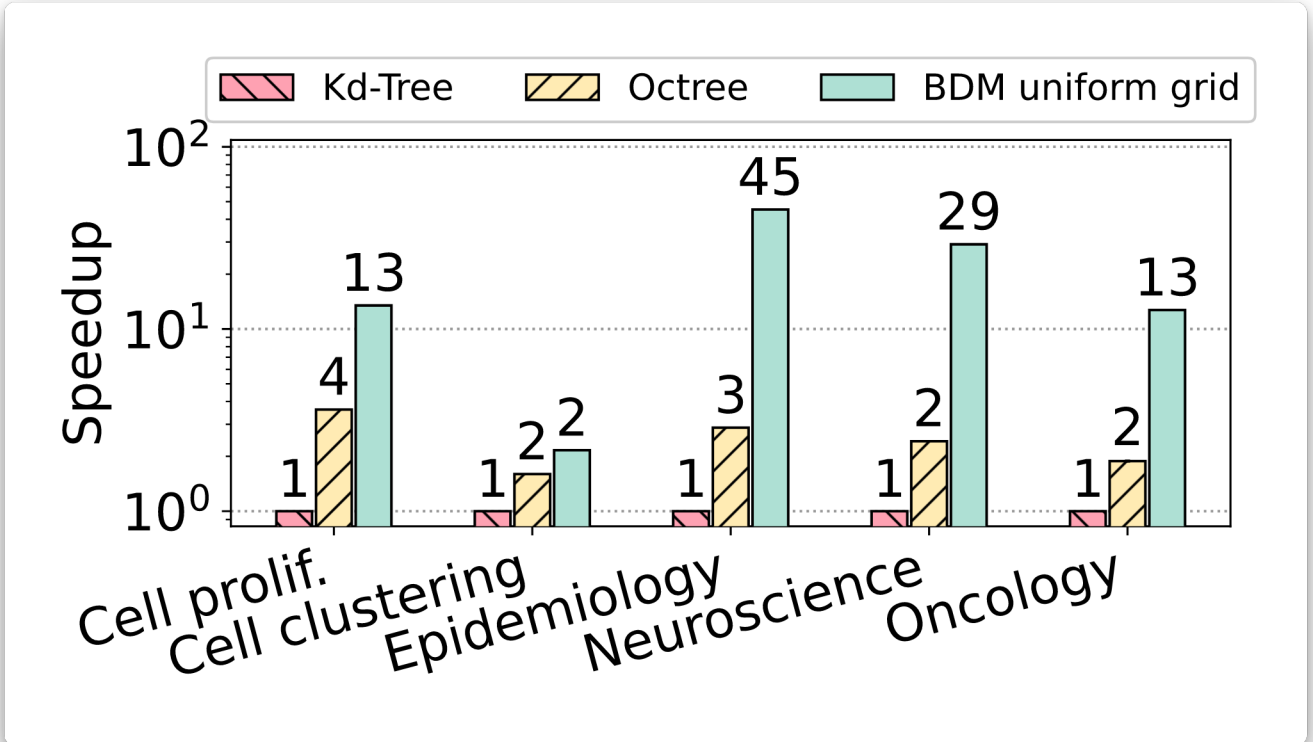
Radial neighbor search comparison

Memory consumption (144 threads and 4 NUMA domains)



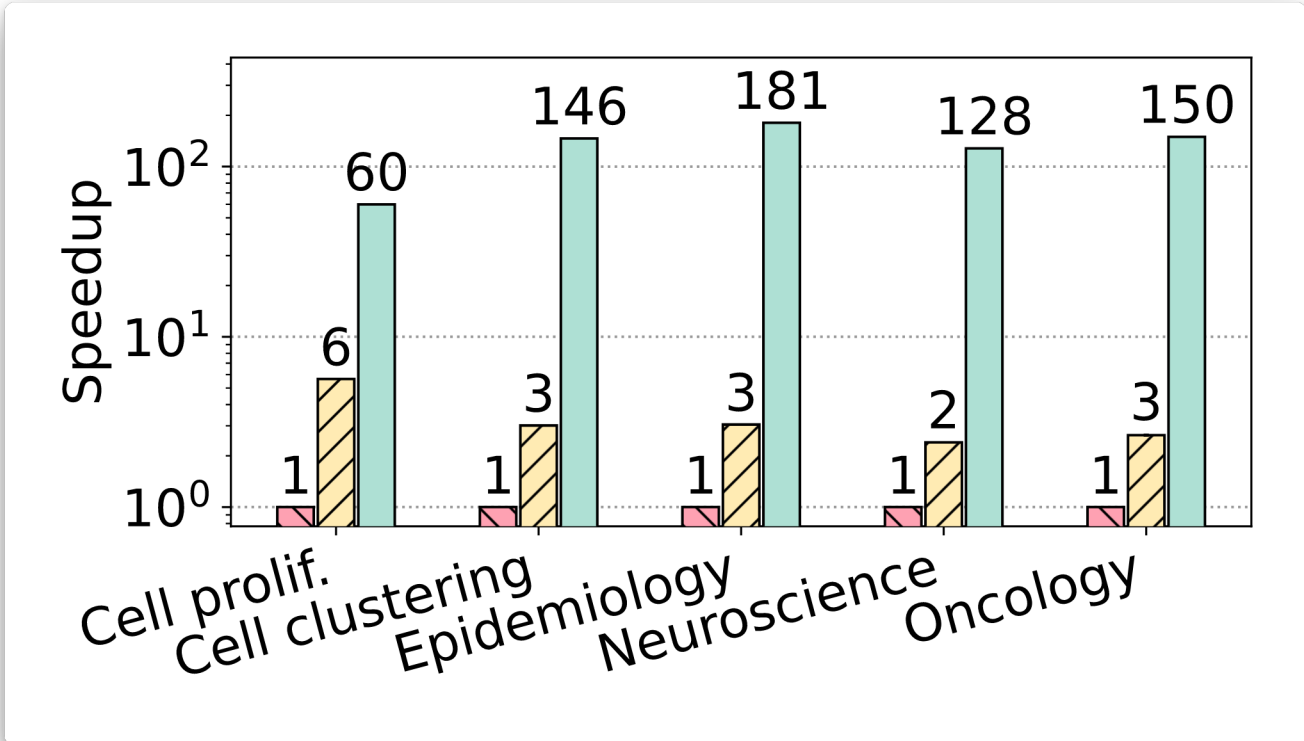
Radial neighbor search comparison

Whole simulation (18 threads and one NUMA domain)



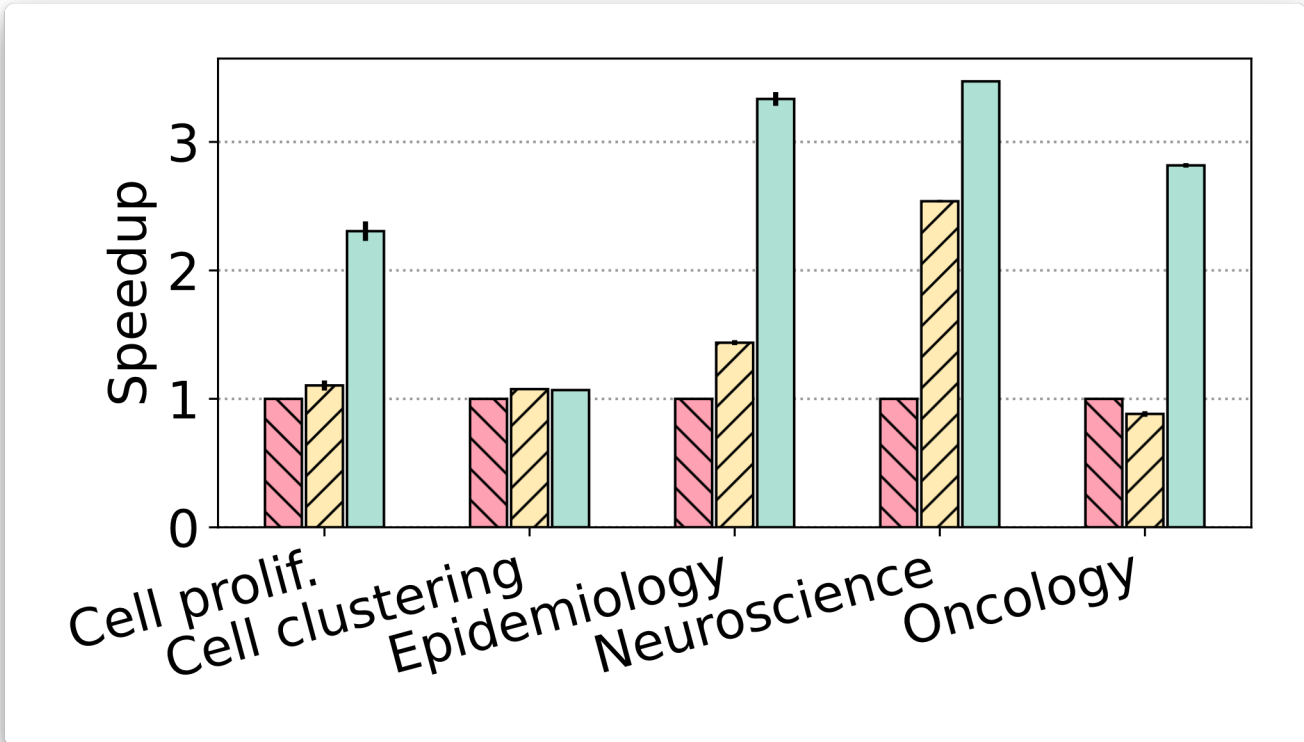
Radial neighbor search comparison

Build time (18 threads and one NUMA domain)



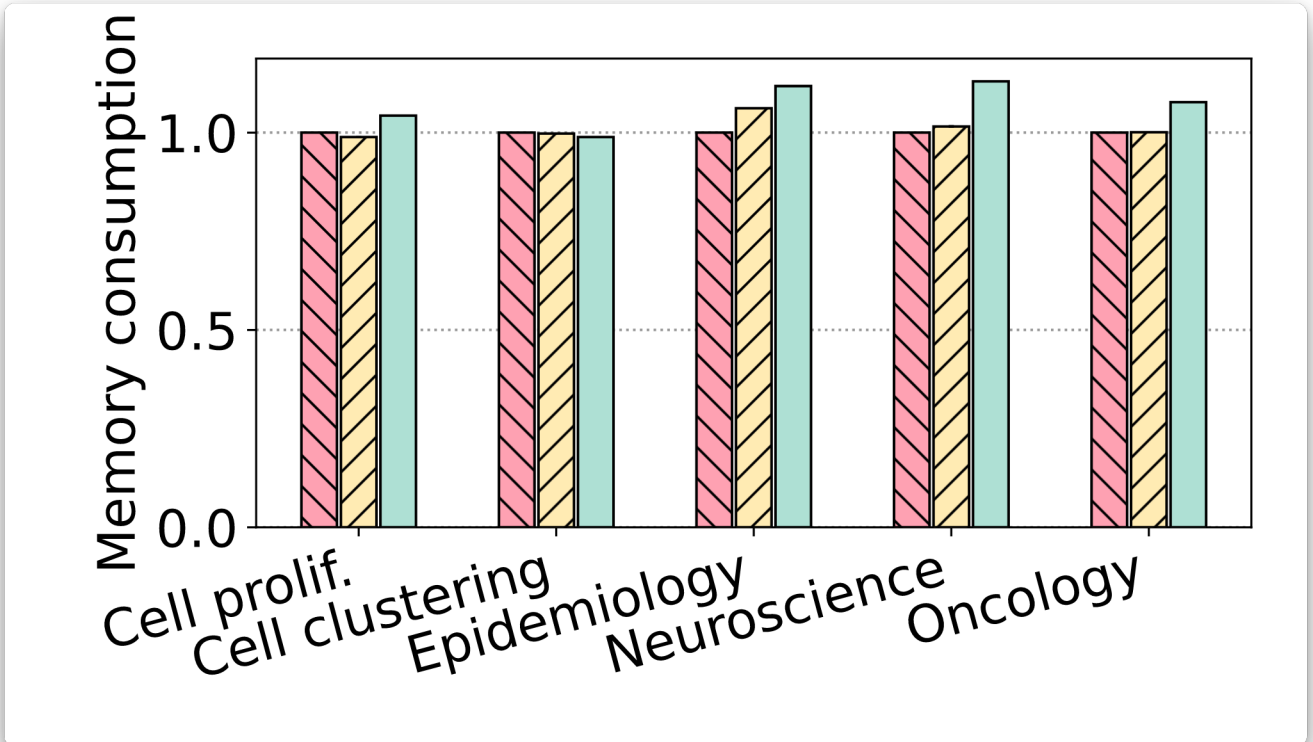
Radial neighbor search comparison

Search time indirect (18 threads and one NUMA domain)

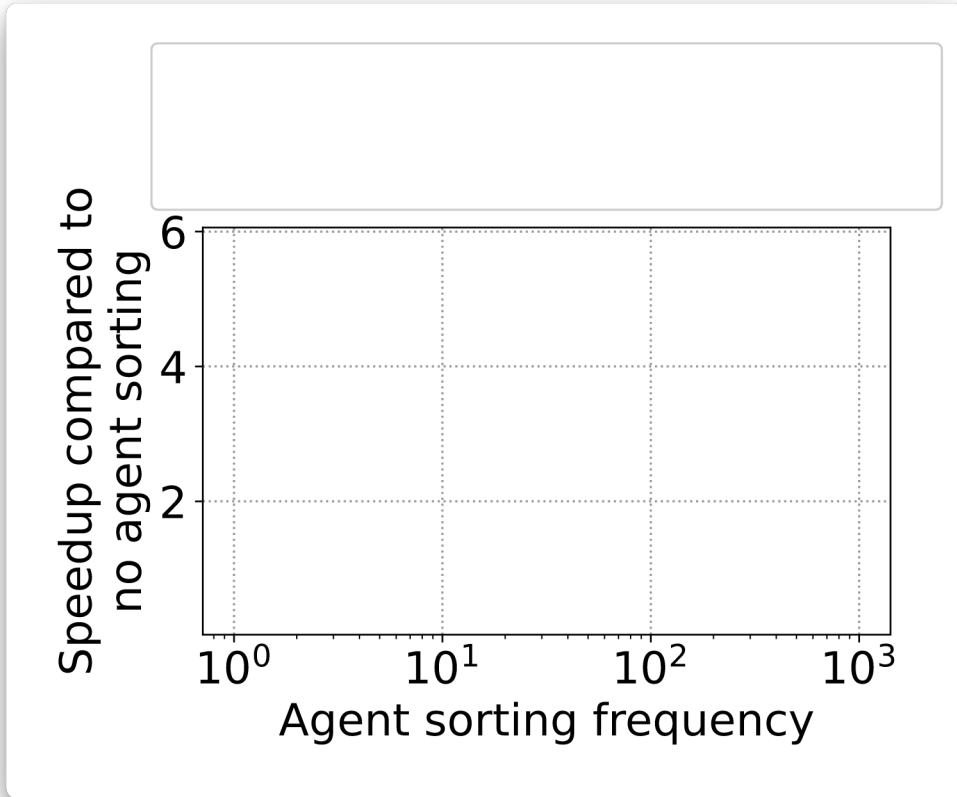


Radial neighbor search comparison

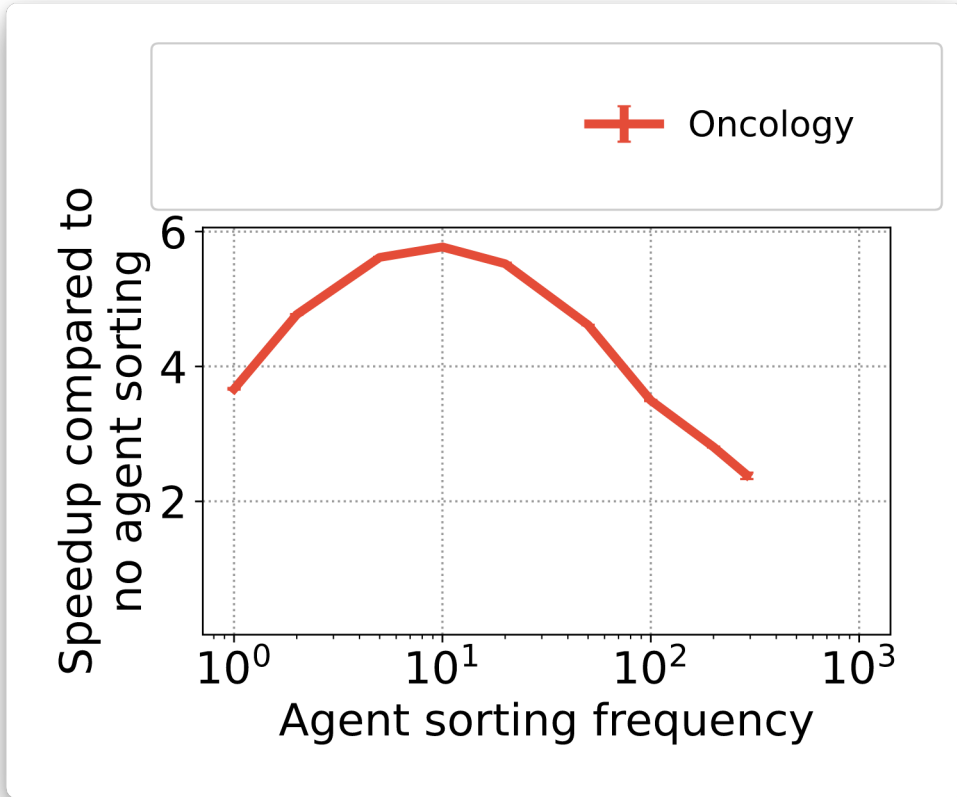
Memory consumption (18 threads and one NUMA domain)



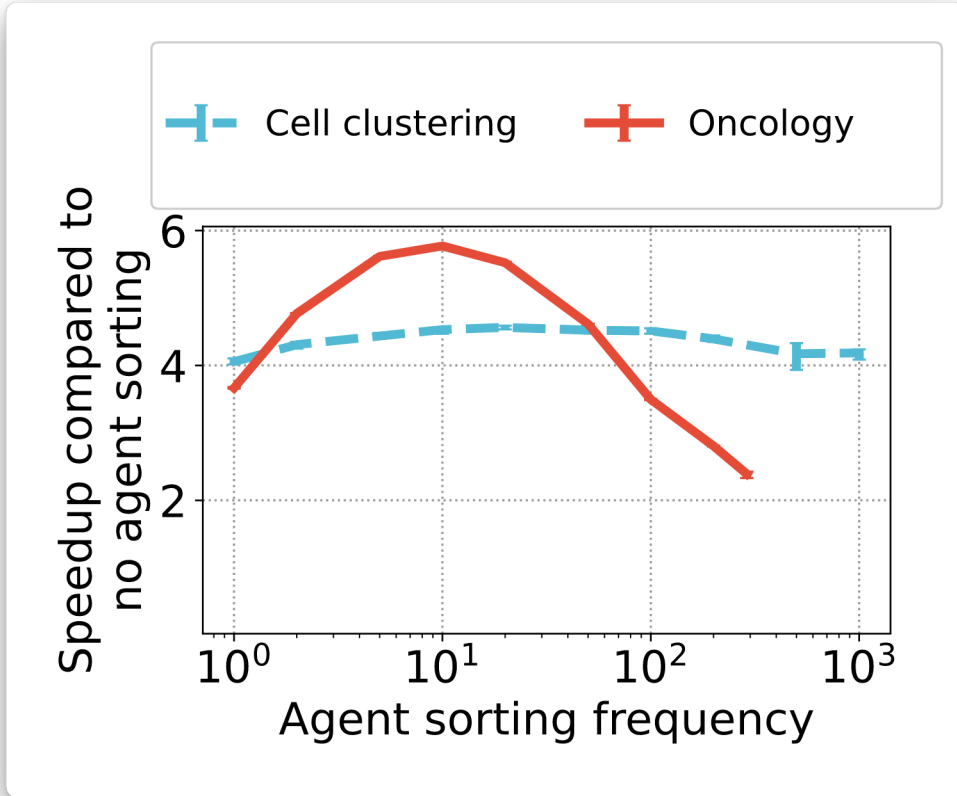
Agent sorting speeds up simulations up to **6**×



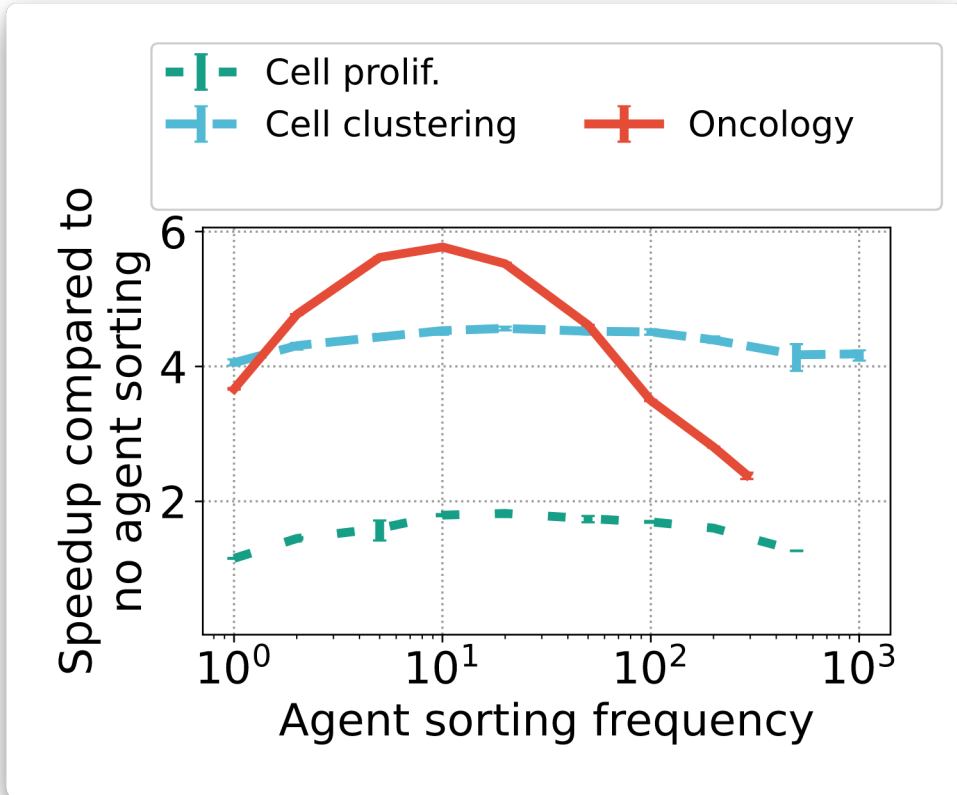
Agent sorting speeds up simulations up to **6**×



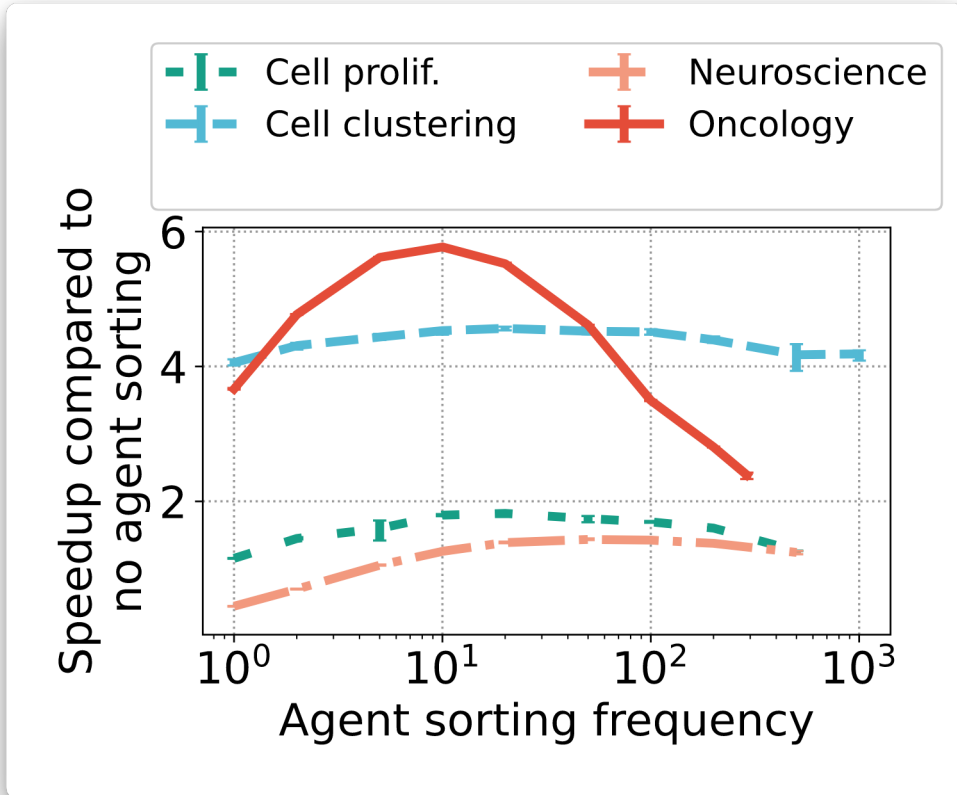
Agent sorting speeds up simulations up to **6**×



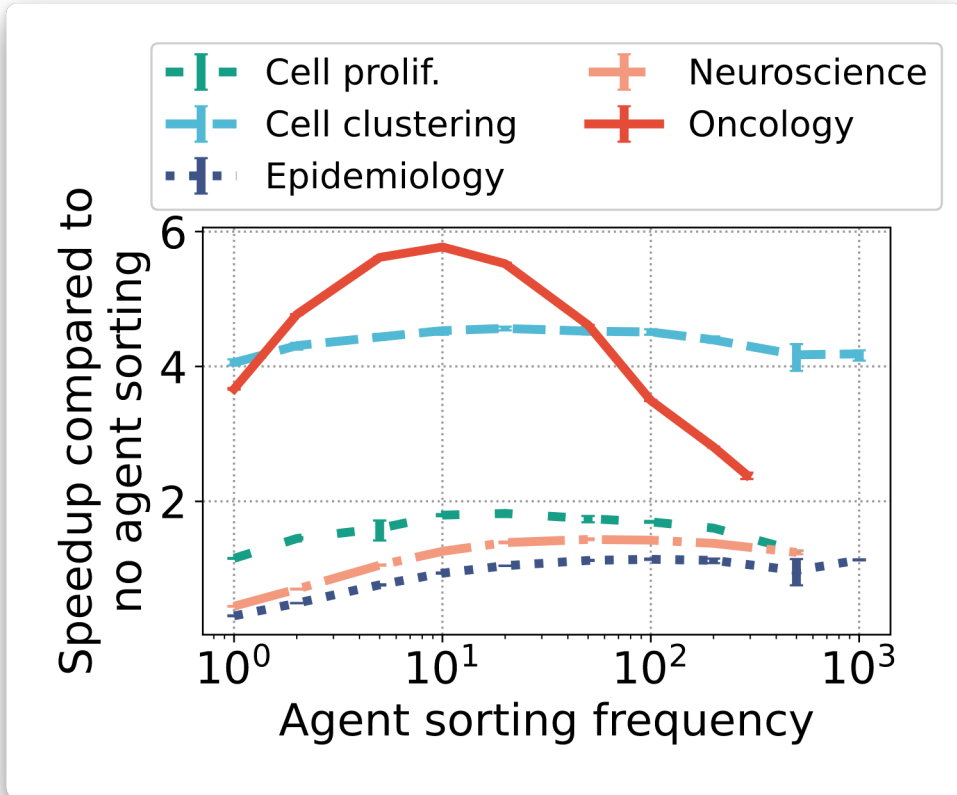
Agent sorting speeds up simulations up to 6×



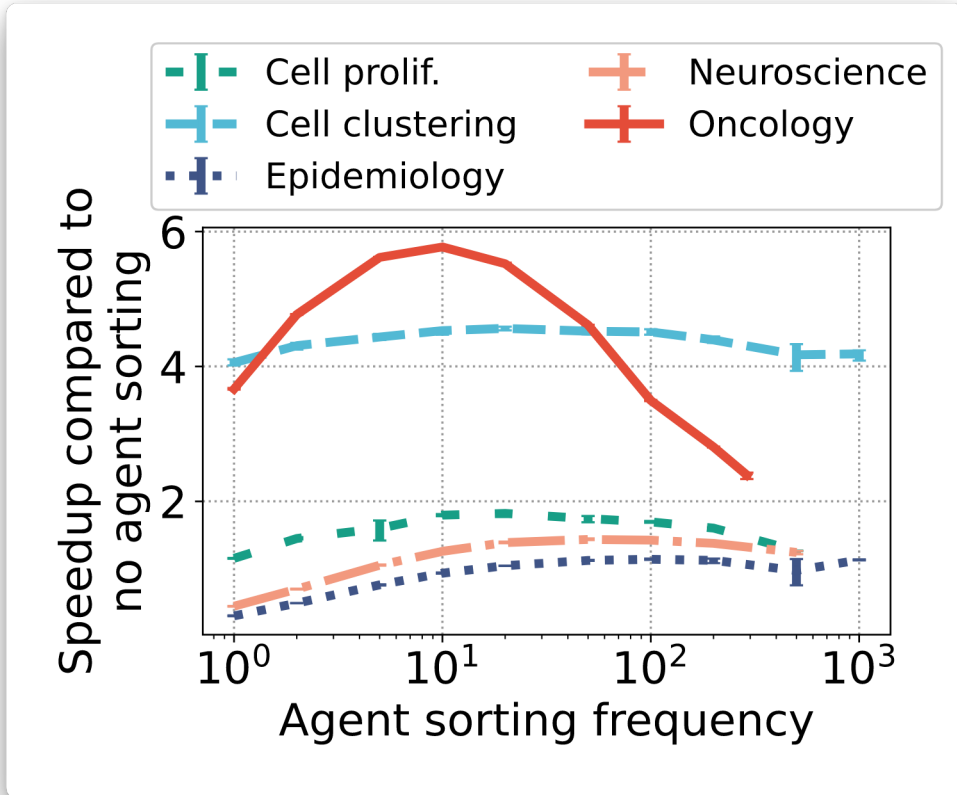
Agent sorting speeds up simulations up to 6×



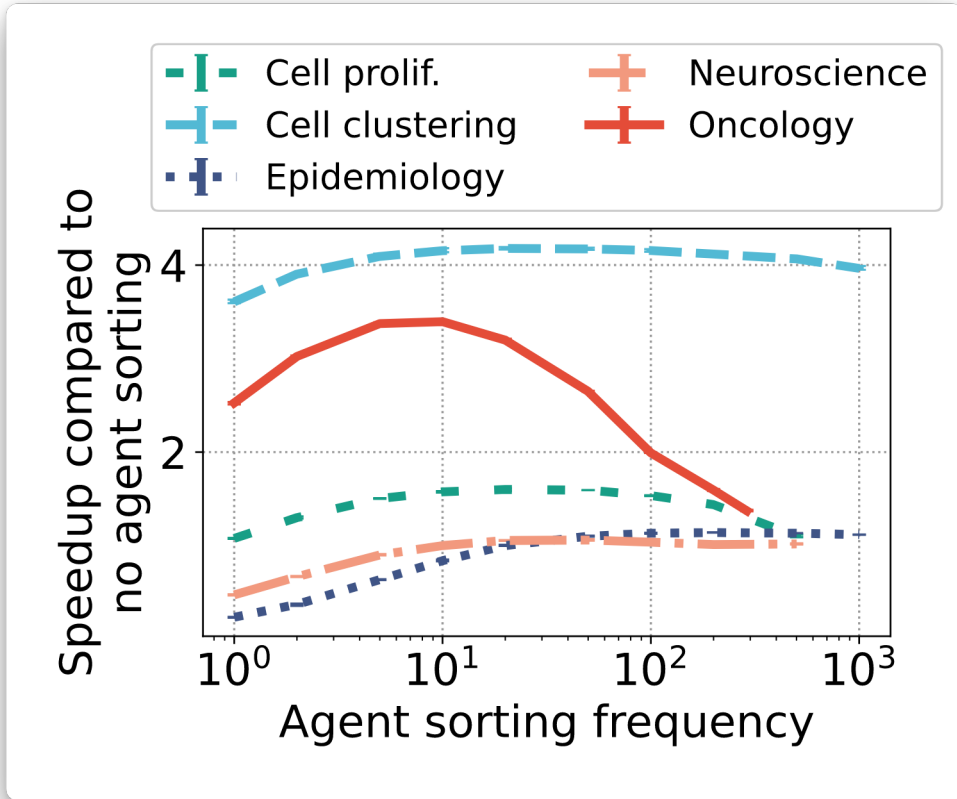
Agent sorting speeds up simulations up to 6×



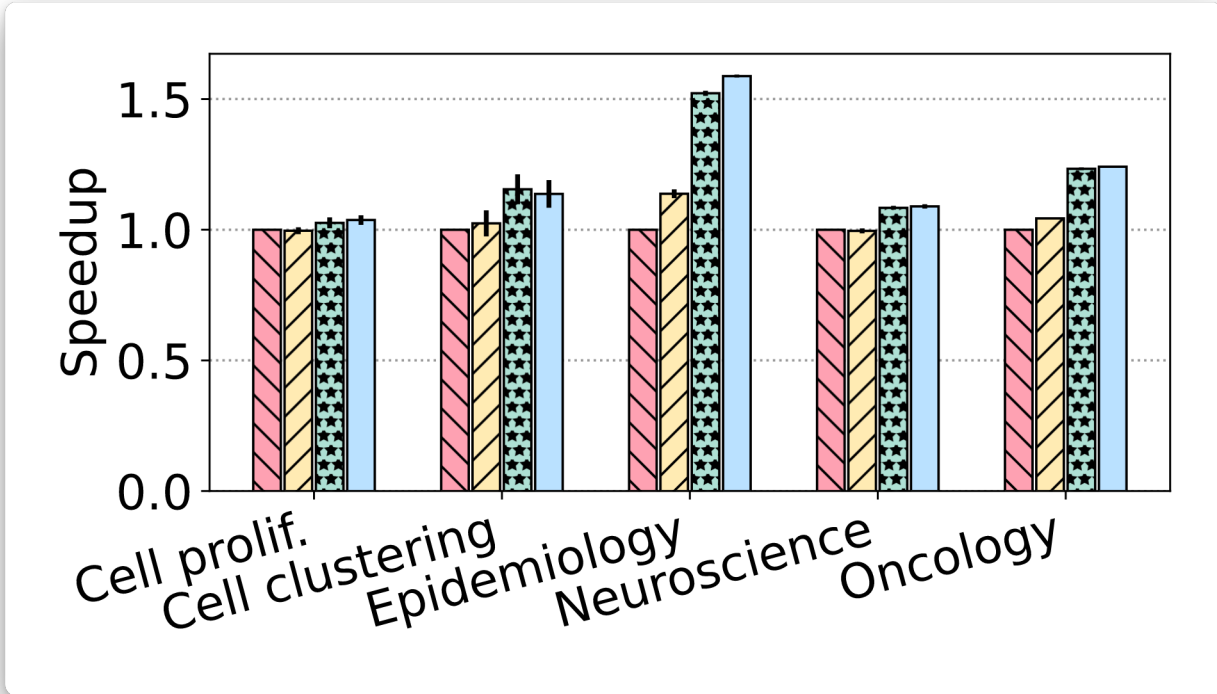
Agent sorting speeds up simulations up to 6×



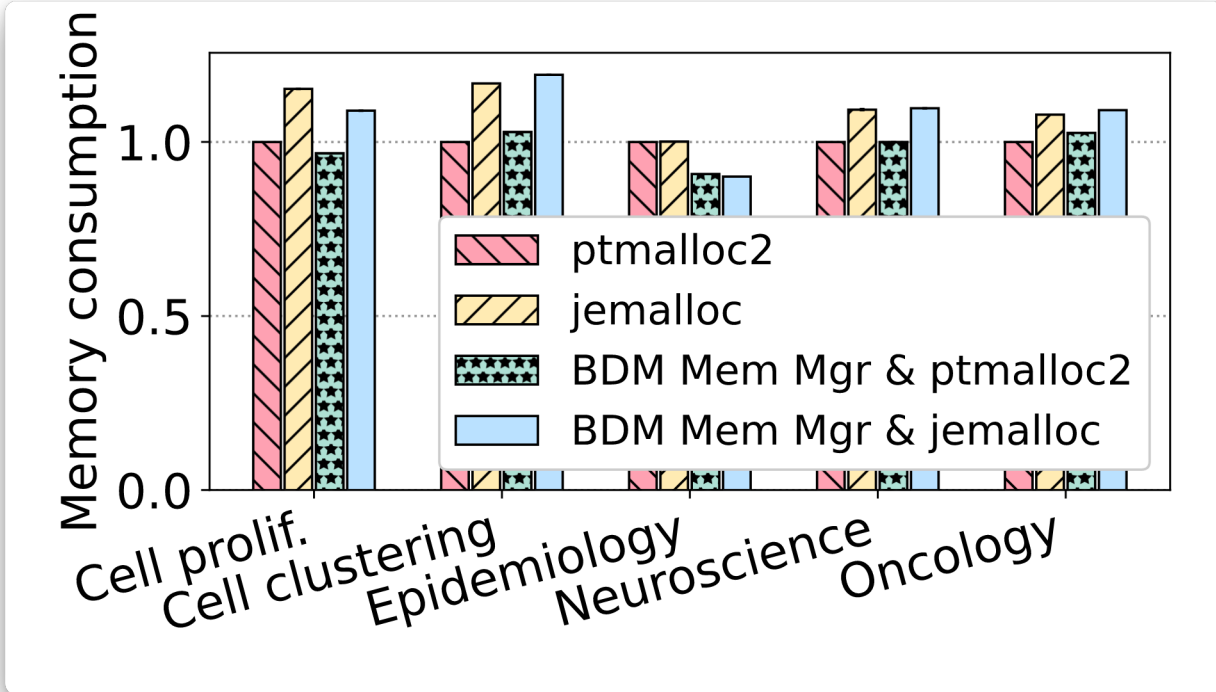
Agent sorting speedup for one NUMA domain



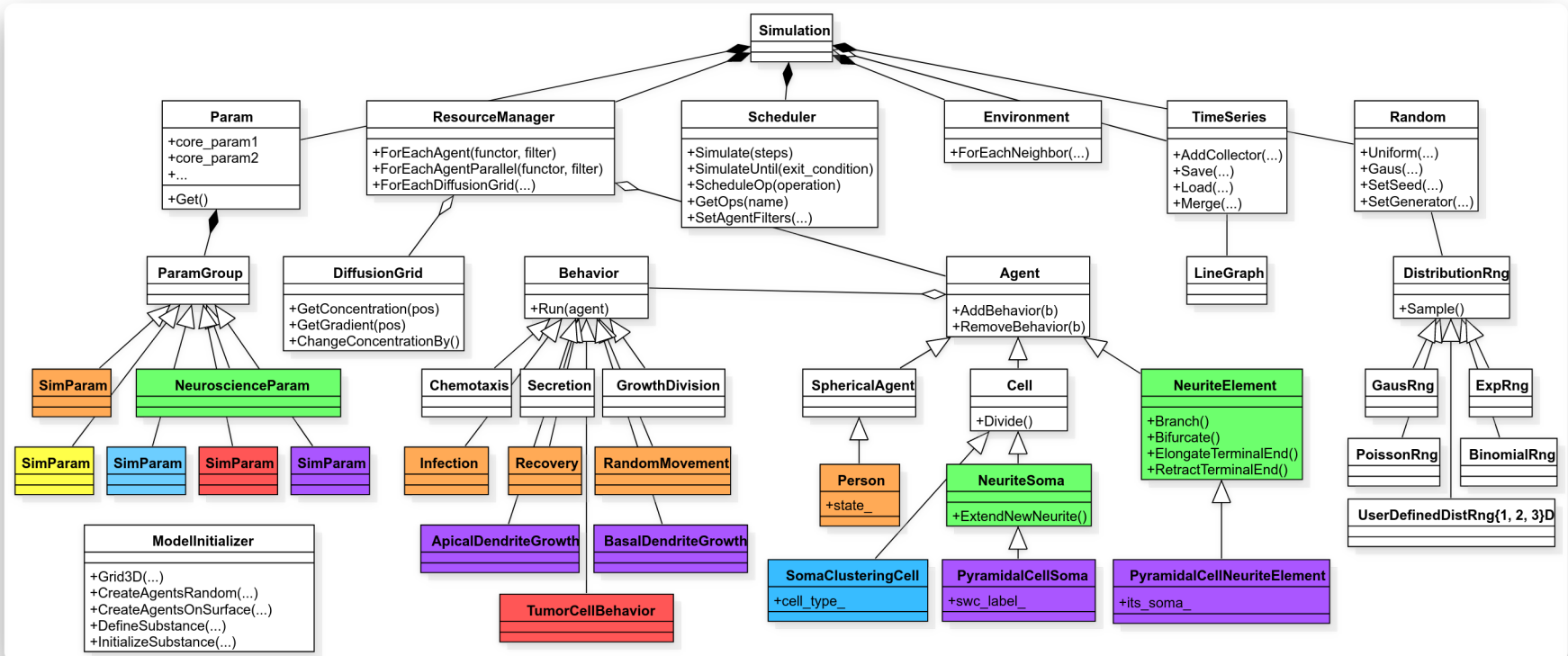
Memory allocator comparison



Memory allocator comparison

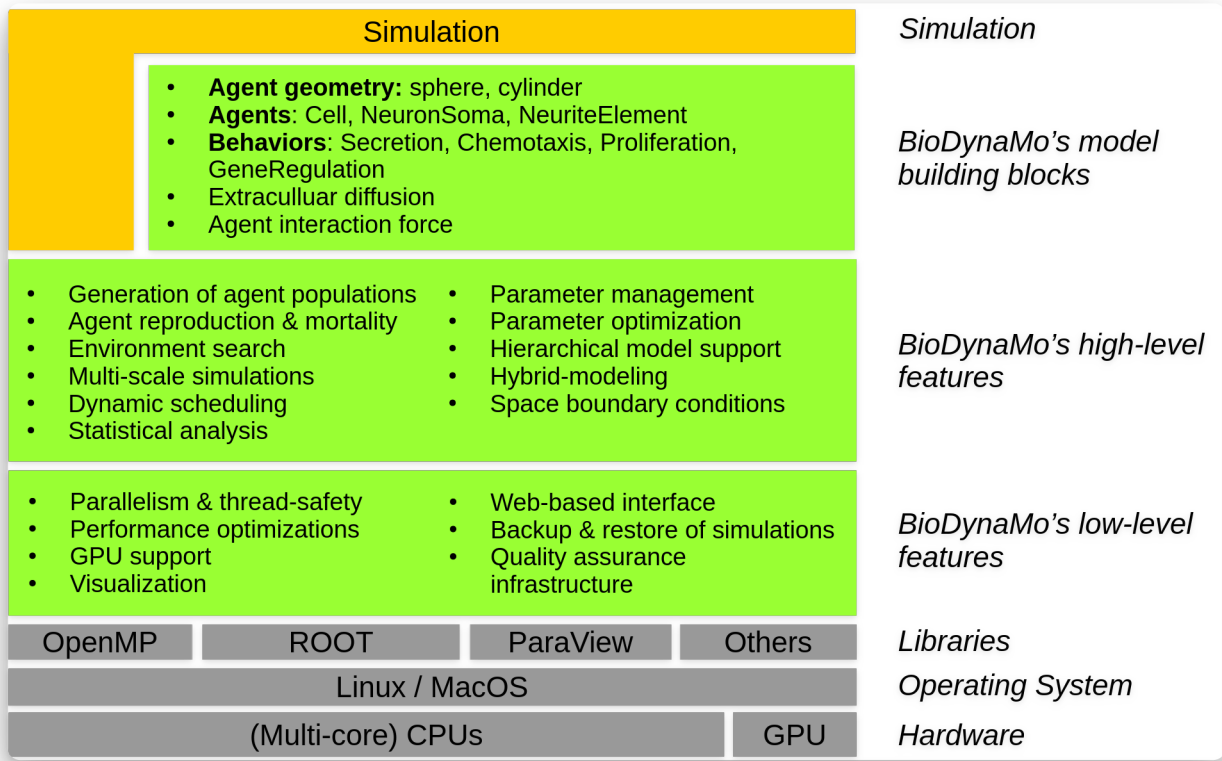


Modularity



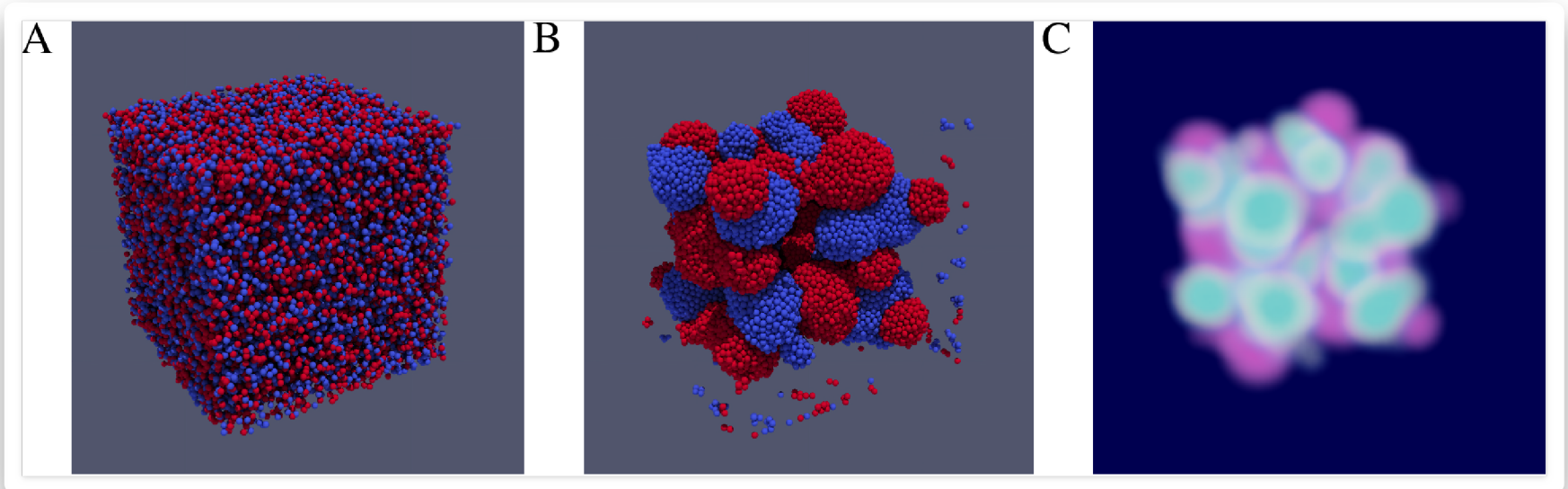
Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Abstraction layers and modeling features



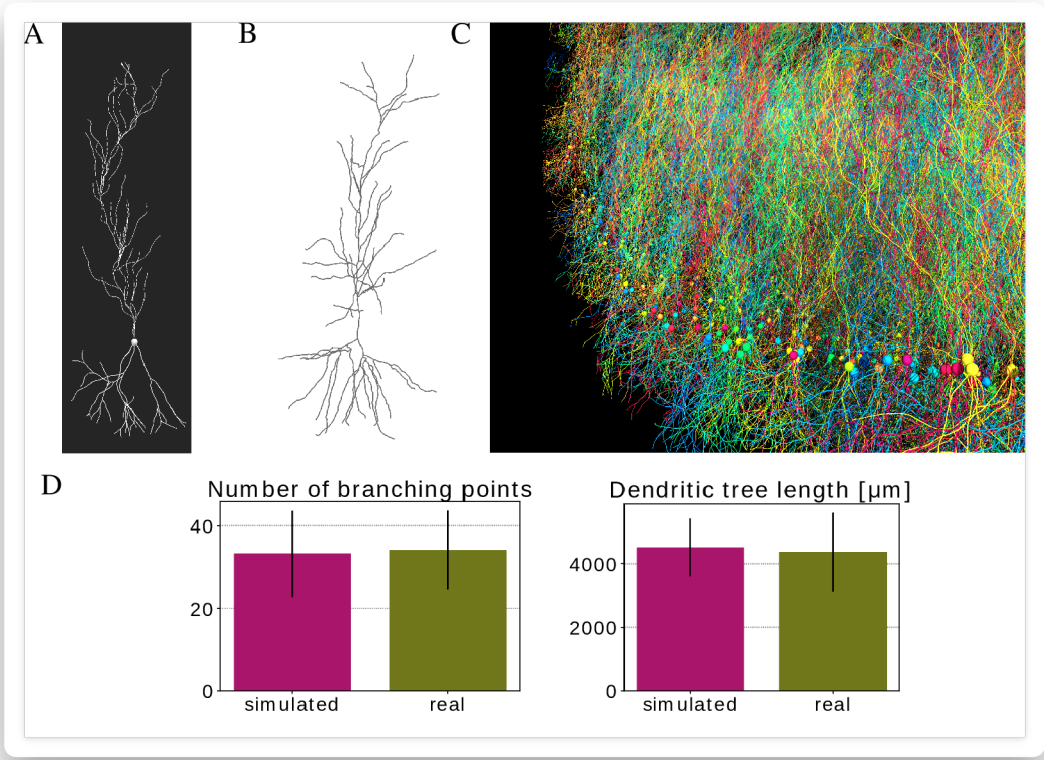
Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Cell sorting



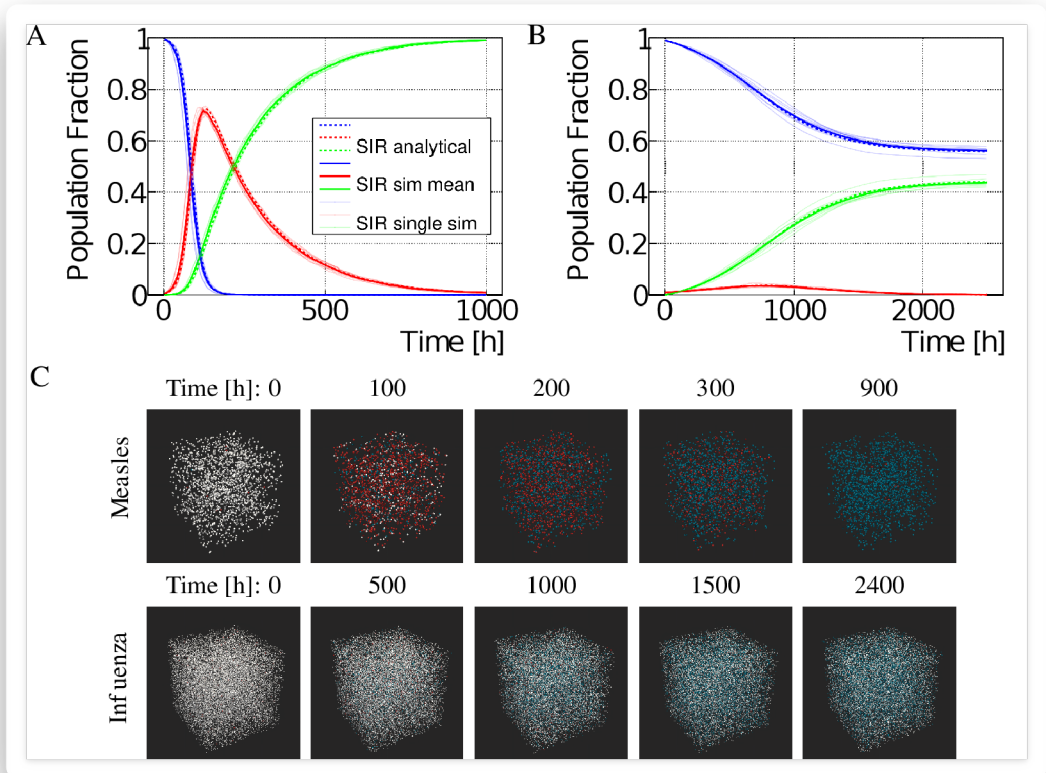
Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Neuroscience use case



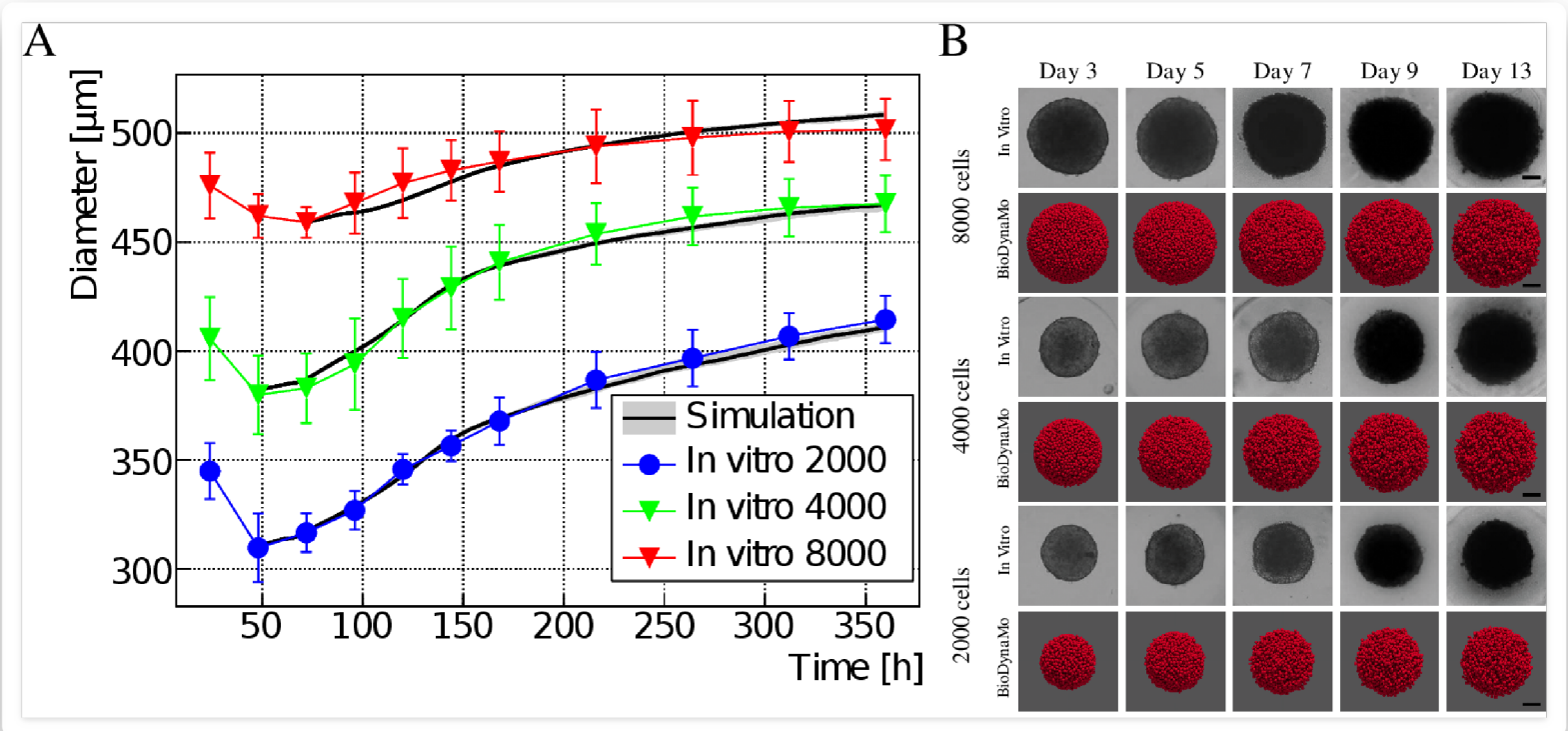
Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Epidemiology use case



Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Oncology use case



Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

Pyramid[illegible]vth video

Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

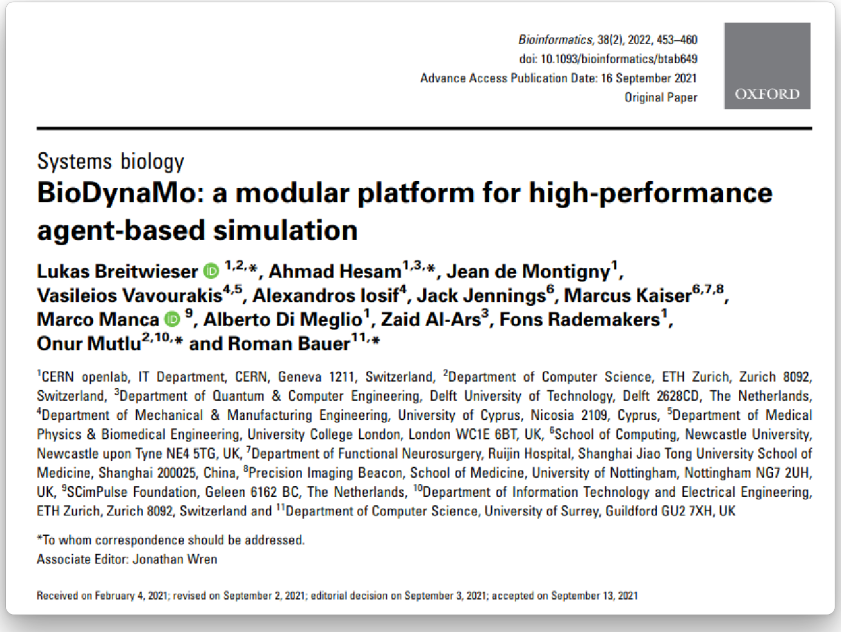
Performance data table

Table 6. **Performance data.** The values in column "Agents" and "Diffusion volumes" are taken from the end of the simulation. Runtime measures the wall-clock time to simulate the number of iterations. It excludes the time for simulation setup and visualization.

Simulation	Agents	Diffusion volumes	Iterations	System (Table 5)	Physical CPUs	Runtime	Memory
Neuroscience use case							
Single (Figure 4A in the main manuscript)	1 494	250	500	A	1	0.16 s	382 MB
				D	1	0.12 s	479 MB
Large-scale (Figure 4C in the main manuscript)	9 036 986	65 536	500	A	72	35 s	6.47 GB
				D	2	11 min 28 s	5.37 GB
Very-large-scale	1 018 644 154	5 606 442	500	B	72	1 h 24 min	438 GB
Oncology use case (Figure 5 in the main manuscript)							
2000 initial cells	4 177	0	312	A	1	1.05 s	382 MB
				D	1	0.832 s	480 MB
4000 initial cells	5 341	0	312	A	1	1.76 s	382 MB
				D	1	1.34 s	480 MB
8000 initial cells	7 861	0	288	A	1	3.27 s	384 MB
				D	1	2.60 s	482 MB
Large-scale	1 000 3925	0	288	A	72	1 min 42 s	7.42 GB
				D	2	43 min 56 s	5.84 GB
Very-large-scale	986 054 868	0	288	B	72	6 h 21 min	604 GB
Epidemiology use case (Figure 6C in the main manuscript)							
Measles	2 010	0	1000	A	1	0.53 s	381 MB
				D	1	0.42 s	479 MB
Seasonal Influenza	20 200	0	2500	A	1	16.41 s	383 MB
				D	1	16.40 s	479 GB
Medium-scale (measles)	100 500	0	1000	A	72	1.36 s	1 GB
Large-scale (measles)	10 050 000	0	1000	A	72	59.19 s	5.87 GB
				D	2	19 min 18 s	5.41 GB
Very-large-scale (measles)	1 005 000 000	0	1000	B	72	2 h 0 min	495 GB
Soma clustering (Figure 2)	32 000	1 240 000	6 000	A	72	12.91 s	1.02 GB
				D	2	2 min 7 s	522 MB

Source: Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649

BioDynaMo's GPU offloading capabilities



Bioinformatics, 38(2), 2022, 453–460
doi: 10.1093/bioinformatics/btab649
Advance Access Publication Date: 16 September 2021
Original Paper

OXFORD

Systems biology

BioDynaMo: a modular platform for high-performance agent-based simulation

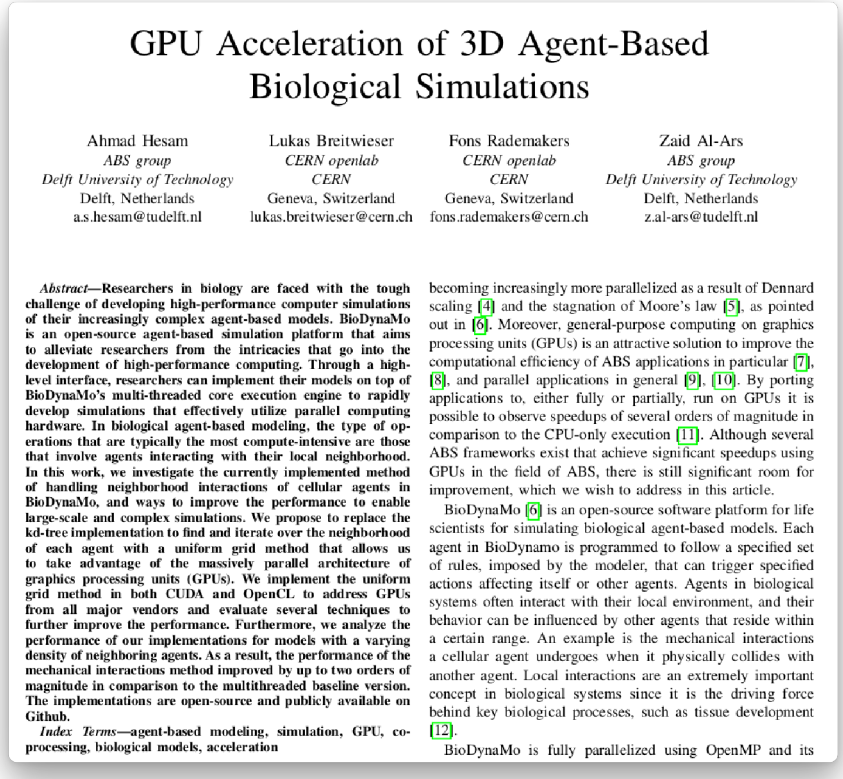
Lukas Breitwieser [1,2,*], Ahmad Hesam[1,3,*], Jean de Montigny[1], Vasileios Vavourakis[4,5], Alexandros Iosif[4], Jack Jennings[6], Marcus Kaiser[6,7,8], Marco Manca [9], Alberto Di Meglio[1], Zaid Al-Ars[3], Fons Rademakers[1], Onur Mutlu[2,10,*] and Roman Bauer[11,*]

[1]CERN openlab, IT Department, CERN, Geneva 1211, Switzerland, [2]Department of Computer Science, ETH Zurich, Zurich 8092, Switzerland, [3]Department of Quantum & Computer Engineering, Delft University of Technology, Delft 2628CD, The Netherlands, [4]Department of Mechanical & Manufacturing Engineering, University of Cyprus, Nicosia 2109, Cyprus, [5]Department of Medical Physics & Biomedical Engineering, University College London, London WC1E 6BT, UK, [6]School of Computing, Newcastle University, Newcastle upon Tyne NE4 5TG, UK, [7]Department of Functional Neurosurgery, Ruijin Hospital, Shanghai Jiao Tong University School of Medicine, Shanghai 200025, China, [8]Precision Imaging Beacon, School of Medicine, University of Nottingham, Nottingham NG7 2UH, UK, [9]SCimPulse Foundation, Geleen 6162 BC, The Netherlands, [10]Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich 8092, Switzerland and [11]Department of Computer Science, University of Surrey, Guildford GU2 7XH, UK

*To whom correspondence should be addressed.

Associate Editor: Jonathan Wren

Received on February 4, 2021; revised on September 2, 2021; editorial decision on September 3, 2021; accepted on September 13, 2021



GPU Acceleration of 3D Agent-Based Biological Simulations

Ahmad Hesam
ABS group
Delft University of Technology
Delft, Netherlands
a.s.hesam@tudelft.nl

Lukas Breitwieser
CERN openlab
CERN
Geneva, Switzerland
lukas.breitwieser@cern.ch

Fons Rademakers
CERN openlab
CERN
Geneva, Switzerland
fons.rademakers@cern.ch

Zaid Al-Ars
ABS group
Delft University of Technology
Delft, Netherlands
z.al-ars@tudelft.nl

***Abstract*—Researchers in biology are faced with the tough challenge of developing high-performance computer simulations of their increasingly complex agent-based models. BioDynaMo is an open-source agent-based simulation platform that aims to alleviate researchers from the intricacies that go into the development of high-performance computing. Through a high-level interface, researchers can implement their models on top of BioDynaMo's multi-threaded core execution engine to rapidly develop simulations that effectively utilize parallel computing hardware. In biological agent-based modeling, the type of operations that are typically the most compute-intensive are those that involve agents interacting with their local neighborhood. In this work, we investigate the currently implemented method of handling neighborhood interactions of cellular agents in BioDynaMo, and ways to improve the performance to enable large-scale and complex simulations. We propose to replace the kd-tree implementation to find and iterate over the neighborhood of each agent with a uniform grid method that allows us to take advantage of the massively parallel architecture of graphics processing units (GPUs). We implement the uniform grid method in both CUDA and OpenCL to address GPUs from all major vendors and evaluate several techniques to further improve the performance. Furthermore, we analyze the performance of our implementations for models with a varying density of neighboring agents. As a result, the performance of the mechanical interactions method improved by up to two orders of magnitude in comparison to the multithreaded baseline version. The implementations are open-source and publicly available on Github.**

***Index Terms*—agent-based modeling, simulation, GPU, co-processing, biological models, acceleration**

becoming increasingly more parallelized as a result of Dennard scaling [4] and the stagnation of Moore's law [5], as pointed out in [6]. Moreover, general-purpose computing on graphics processing units (GPUs) is an attractive solution to improve the computational efficiency of ABS applications in particular [7], [8], and parallel applications in general [9], [10]. By porting applications to, either fully or partially, run on GPUs it is possible to observe speedups of several orders of magnitude in comparison to the CPU-only execution [11]. Although several ABS frameworks exist that achieve significant speedups using GPUs in the field of ABS, there is still significant room for improvement, which we wish to address in this article.

BioDynaMo [6] is an open-source software platform for life scientists for simulating biological agent-based models. Each agent in BioDynamo is programmed to follow a specified set of rules, imposed by the modeler, that can trigger specified actions affecting itself or other agents. Agents in biological systems often interact with their local environment, and their behavior can be influenced by other agents that reside within a certain range. An example is the mechanical interactions a cellular agent undergoes when it physically collides with another agent. Local interactions are an extremely important concept in biological systems since it is the driving force behind key biological processes, such as tissue development [12].

BioDynaMo is fully parallelized using OpenMP and its

- Lukas Breitwieser et al., 2022, DOI: 10.1093/bioinformatics/btab649
- Ahmad Hesam et al., 2021, DOI: 10.1109/IPDPSW52791.2021.00040