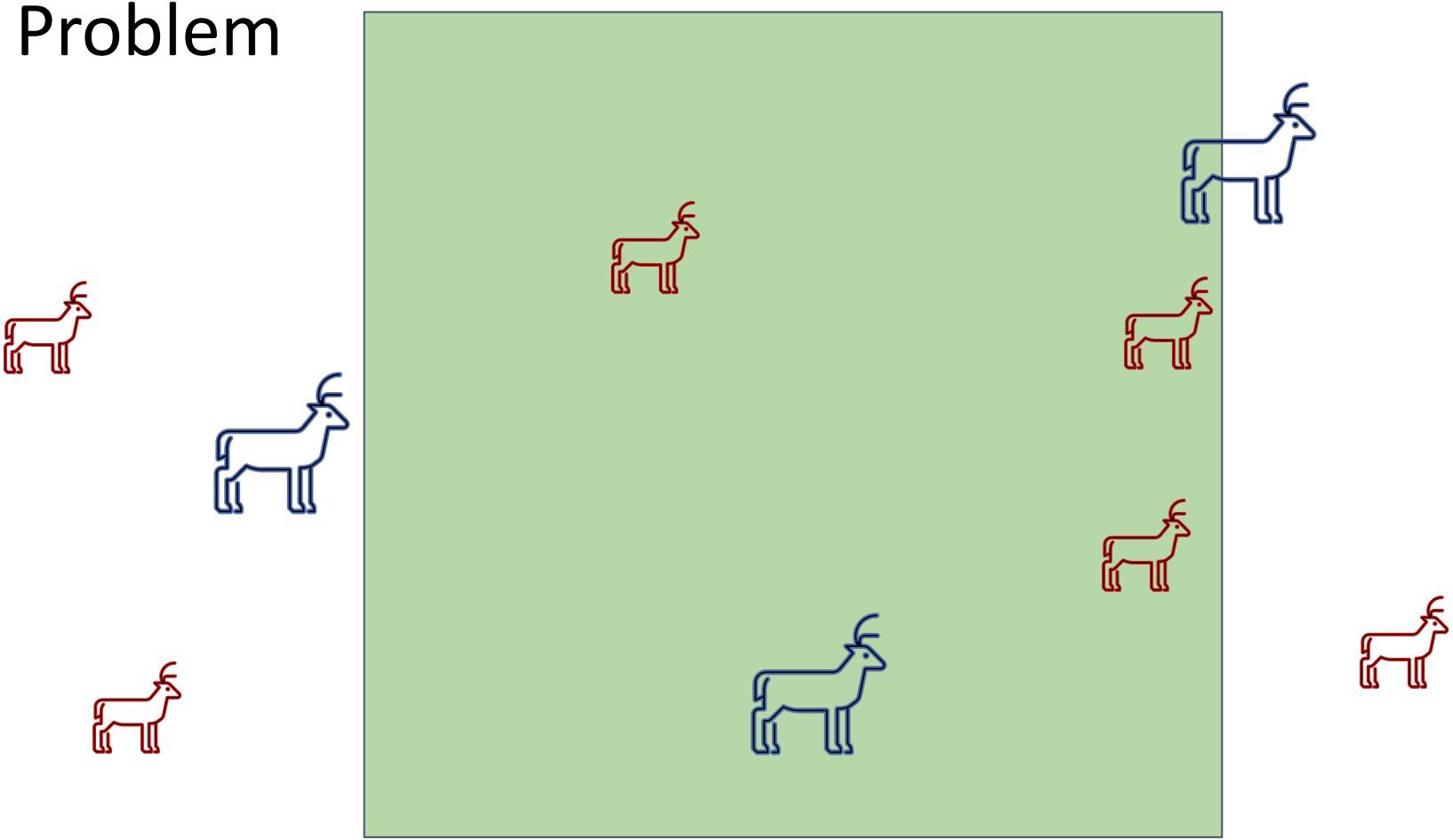
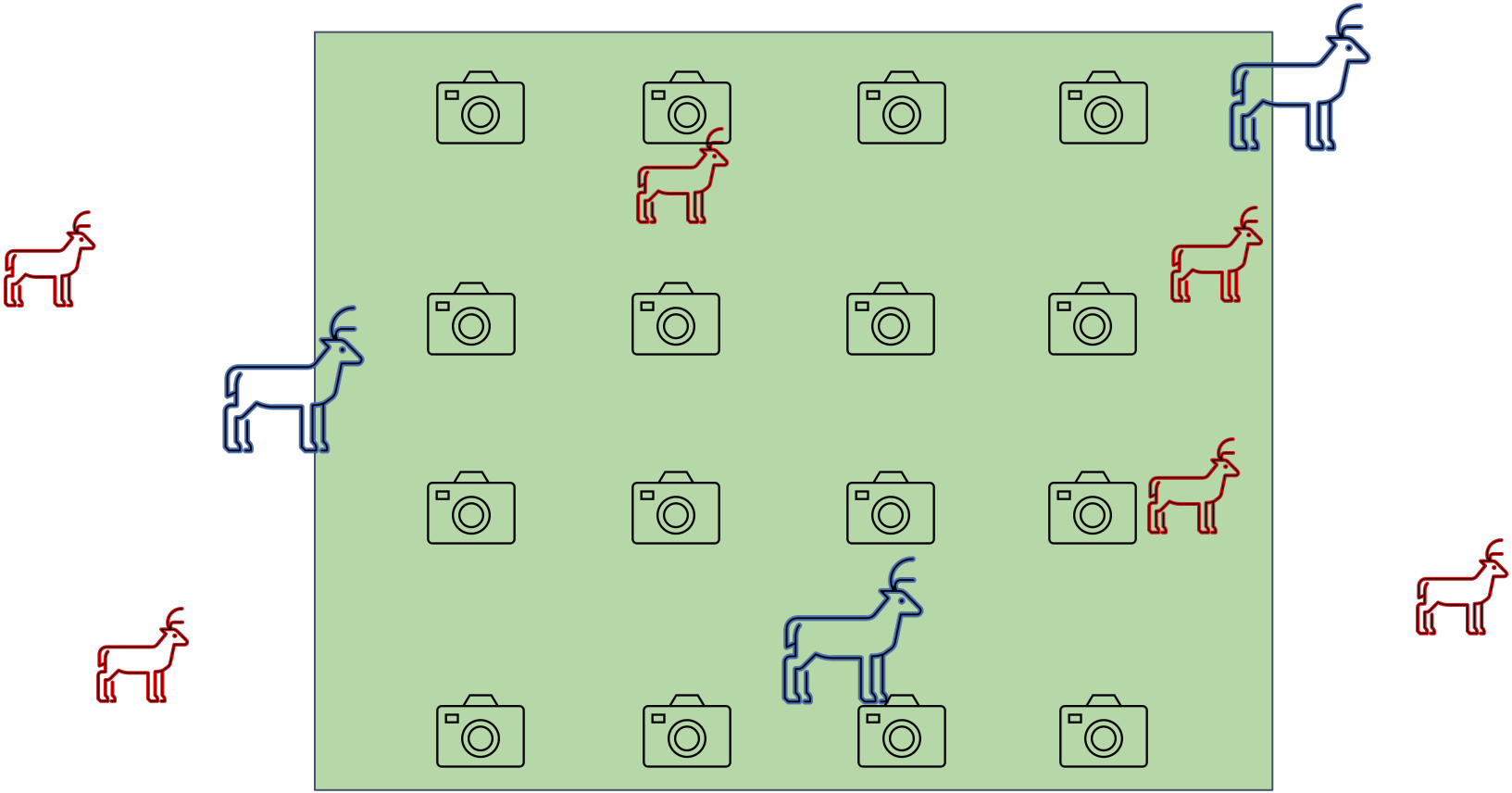
Estimating the population density of unmarked animals using camera traps

A simulation-based evaluation

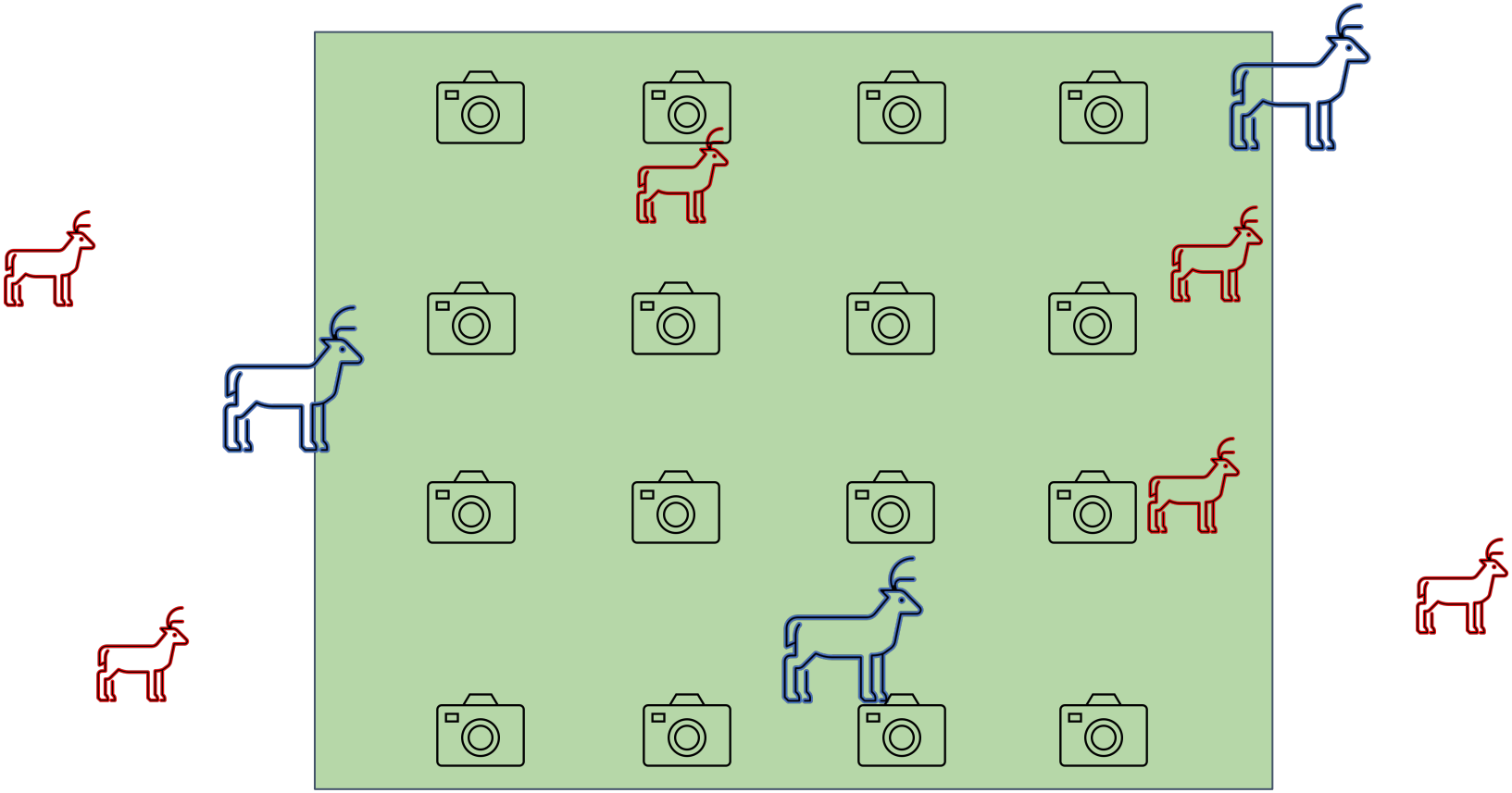
Louis von Leitner, Thomas Hay

Problem





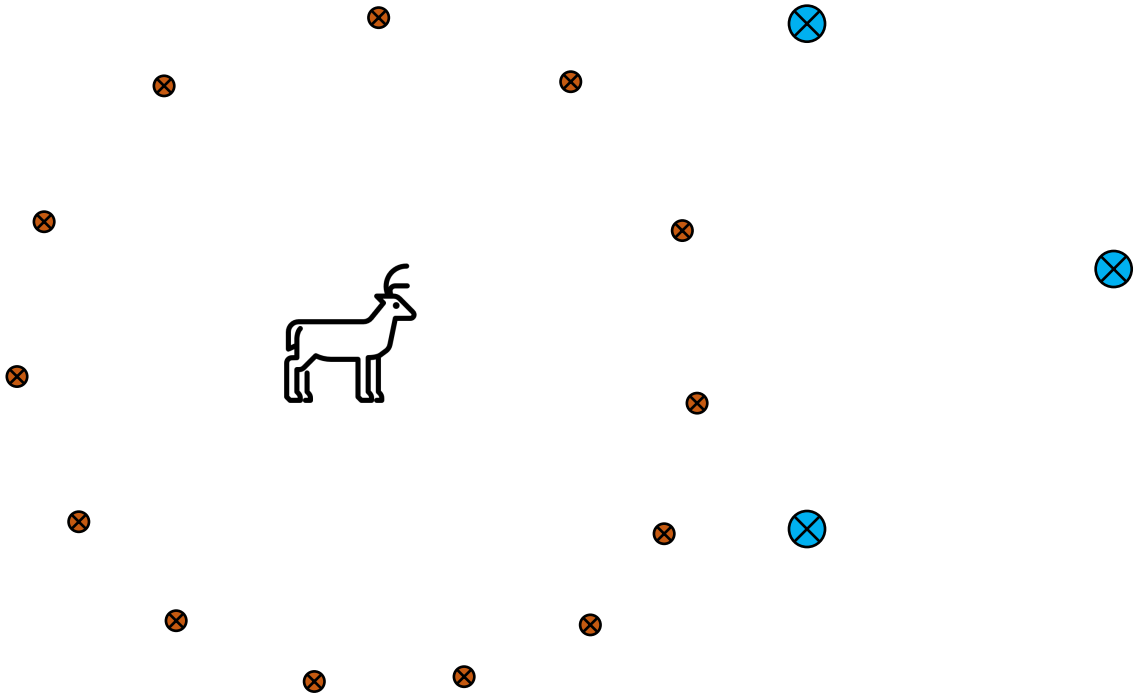
How many deer are in the forest?



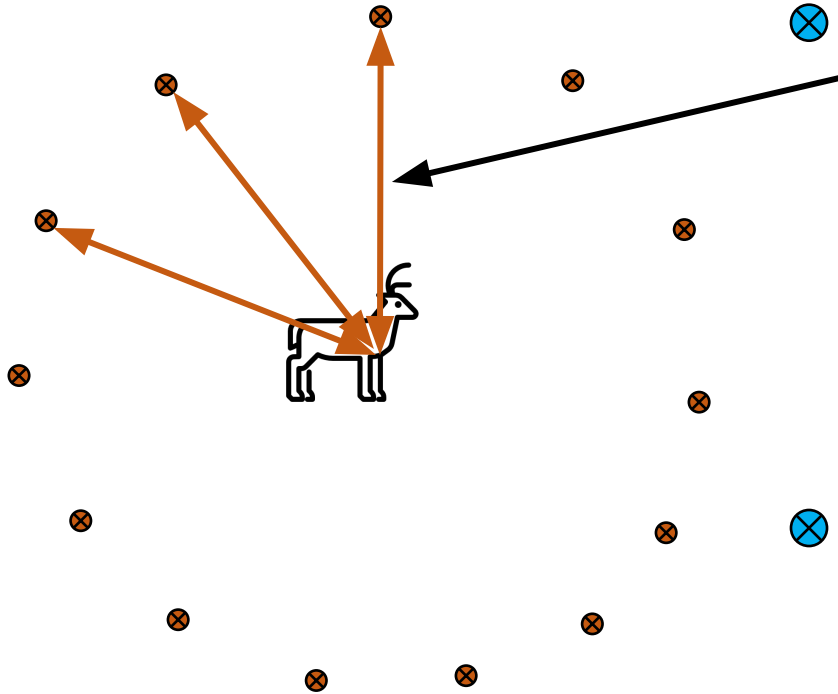
Simulations!

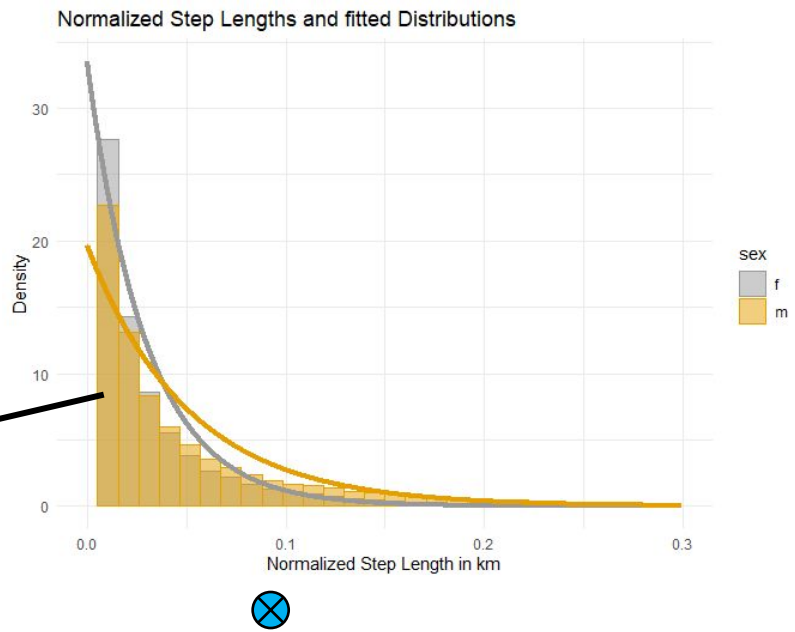
Movement Model

Movement Model



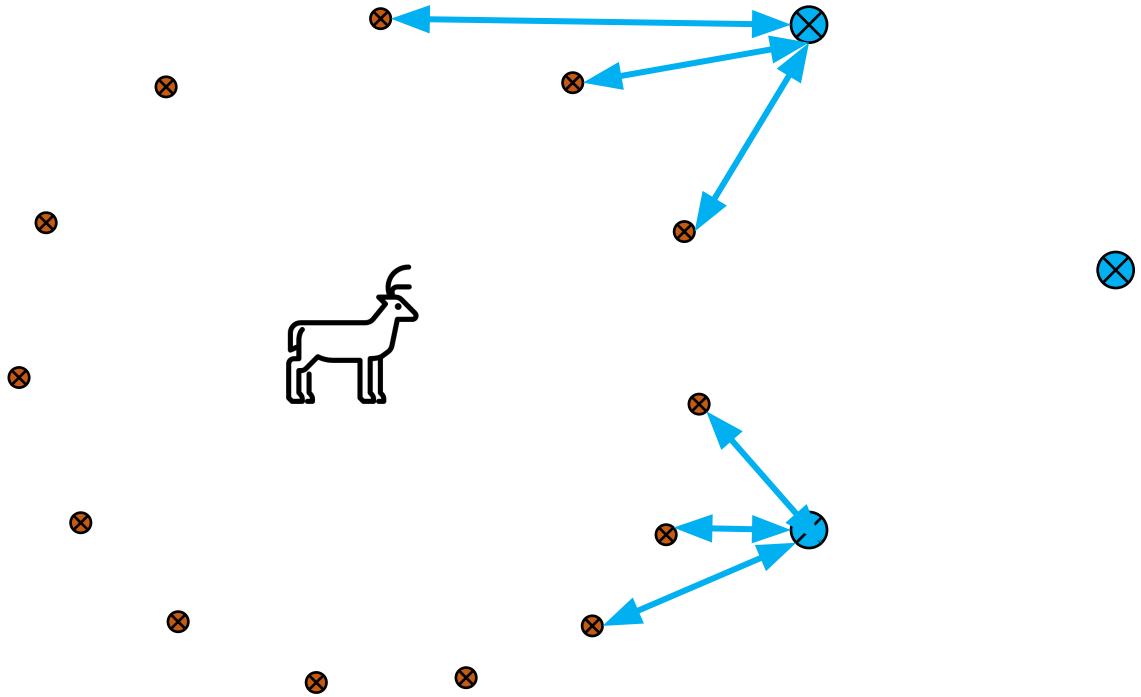
Movement Model





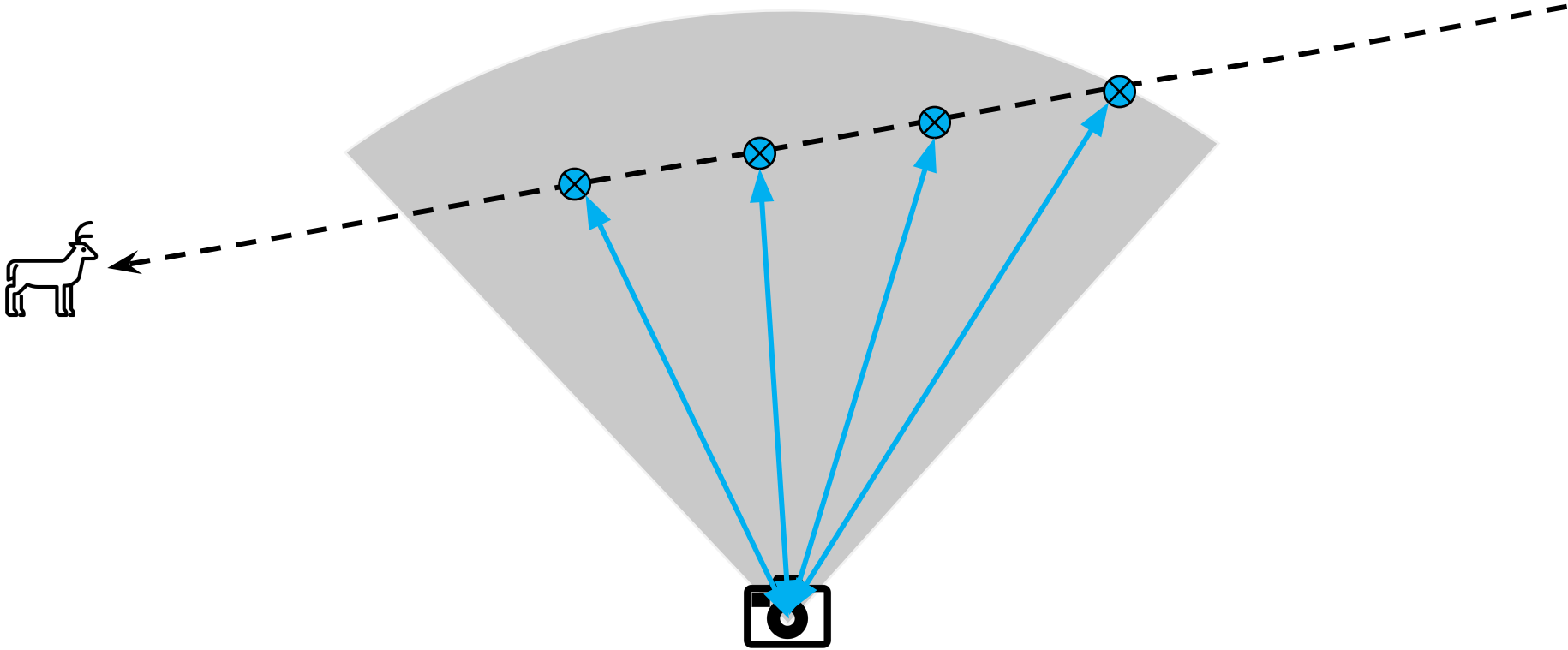
Movement Model

Probability = 1/a * exp (d*b)

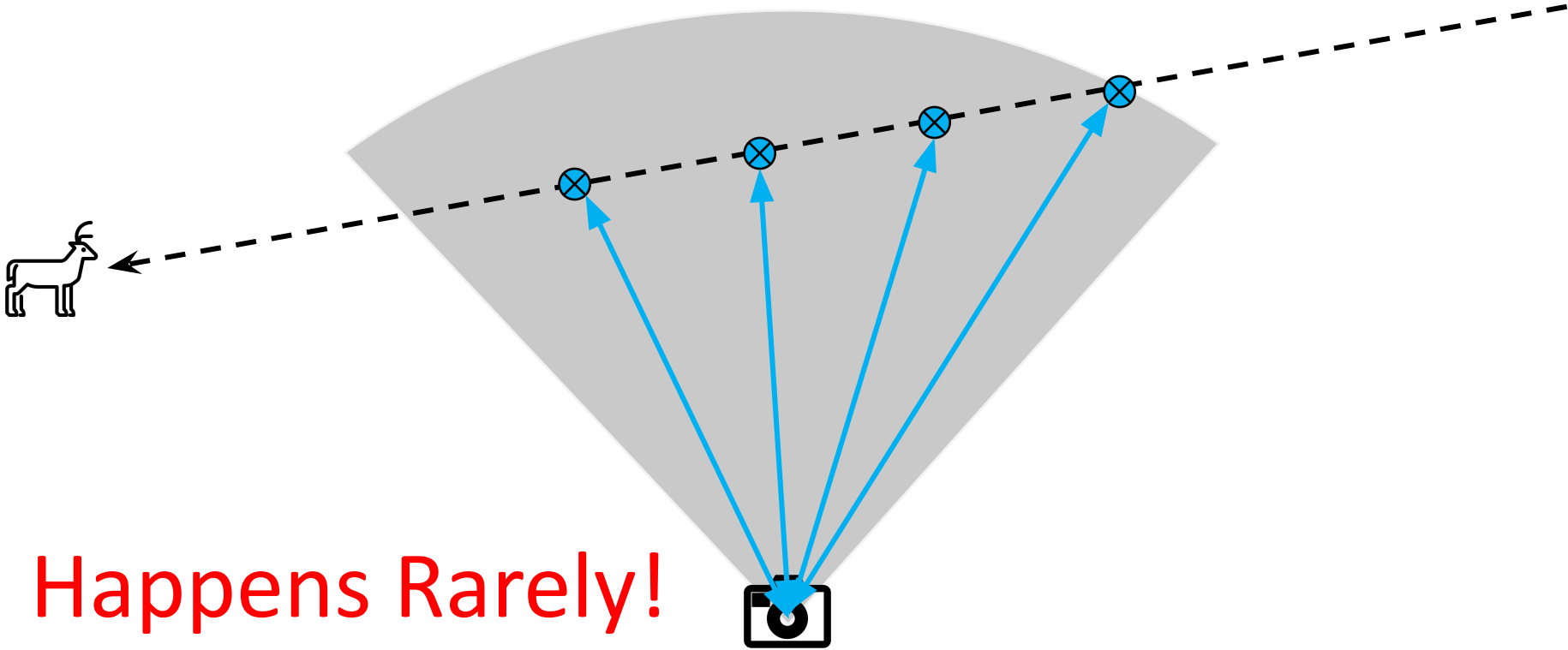


Weight computation takes ⅔ of the runtime!

Camera trapping



Camera trapping



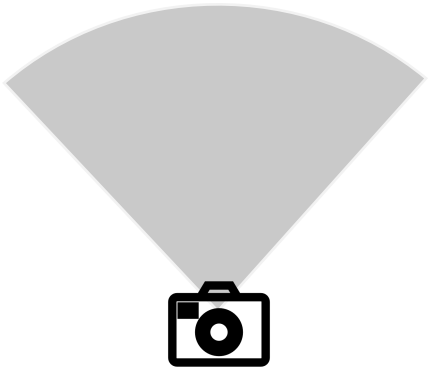
Happens Rarely!

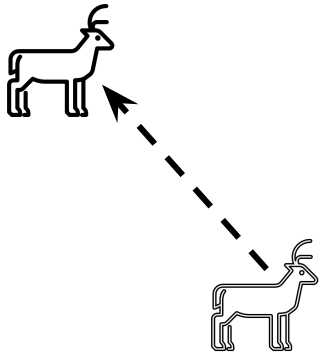
Camera Trapping

- Happens rarely
- Takes on average double the time as doing one animal step

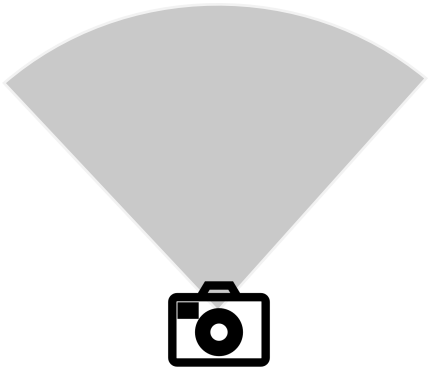
Sequential Solution Ideas

Small steps and always check everything

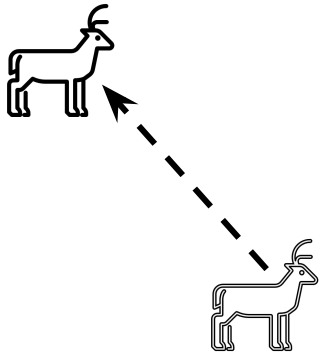




Small steps and always check everything

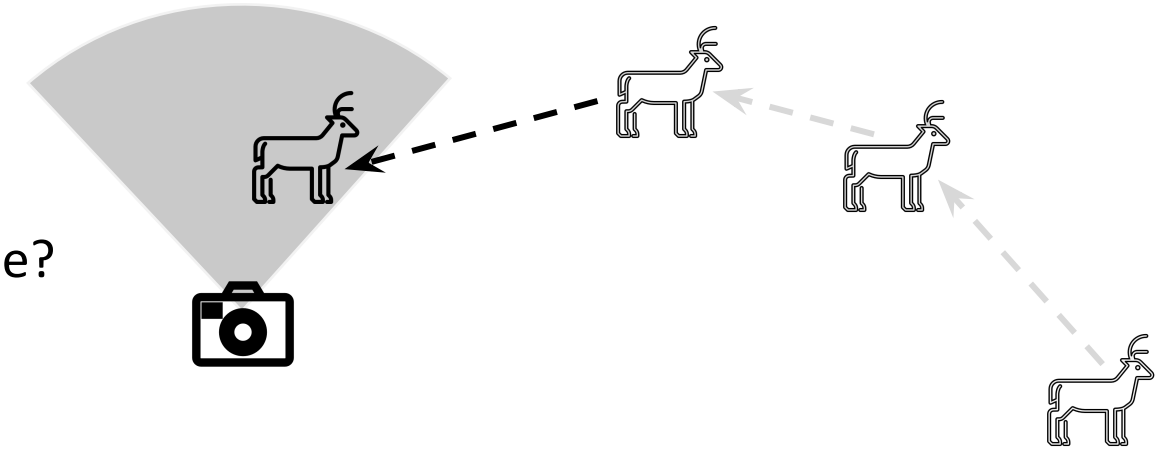


Animal inside?



Small steps and always check everything

Animal inside?



Remarks

Idea:

- No path camera tracking necessary → decrease runtime?

Answer:

- This increases runtime by a lot

Quick Math

ca. 6 month period simulation

One step every minute:
⇒ 2000 steps x 120 = 240.000

x 100 animals = 24.000.000
steps in total

One step every 2 hours:
⇒ 2000 steps
x 100 animals = 200.000
steps in total

Average time of camera trapping = 2x time of moving animal

Quick Math

ca. 6 month period simulation

⇒ 24.000.000 - 200.000 = 23.800.000

23.800.000 / 2 = 11.900.000

Amount of Photos to be taken so that simulations take the same amount of time

Average of about 800 photos per simulation (max 200.000)

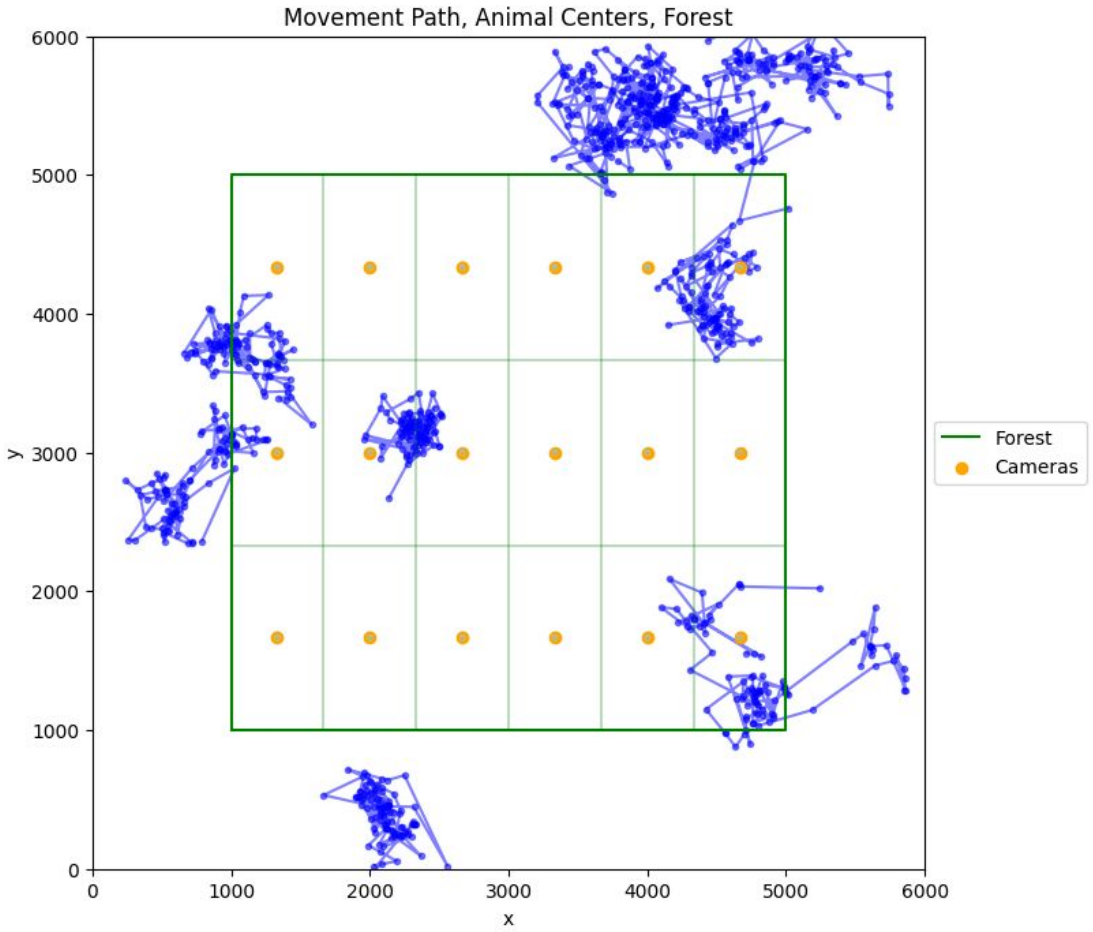
Quick Math

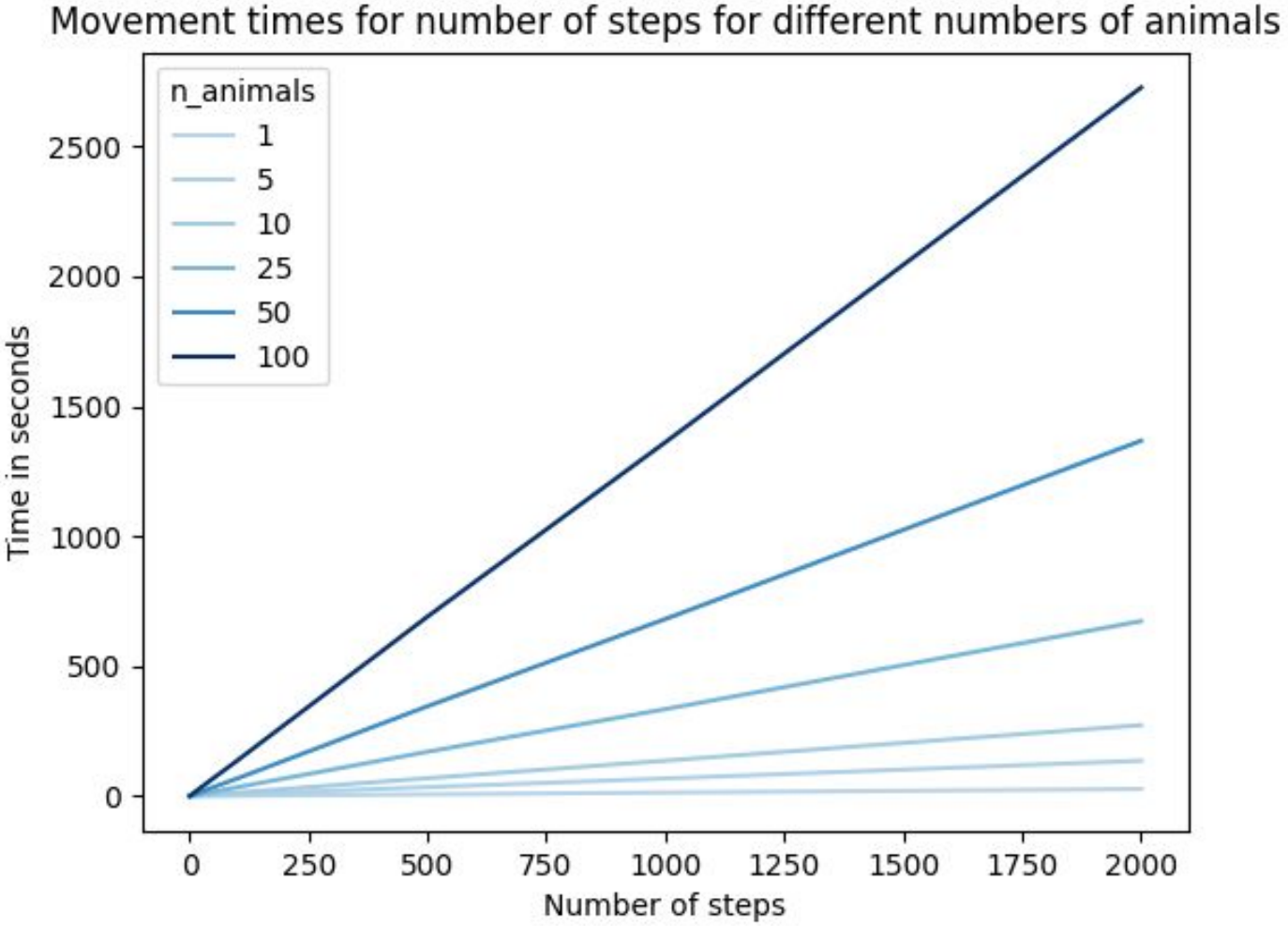
ca. 6 month period simulation

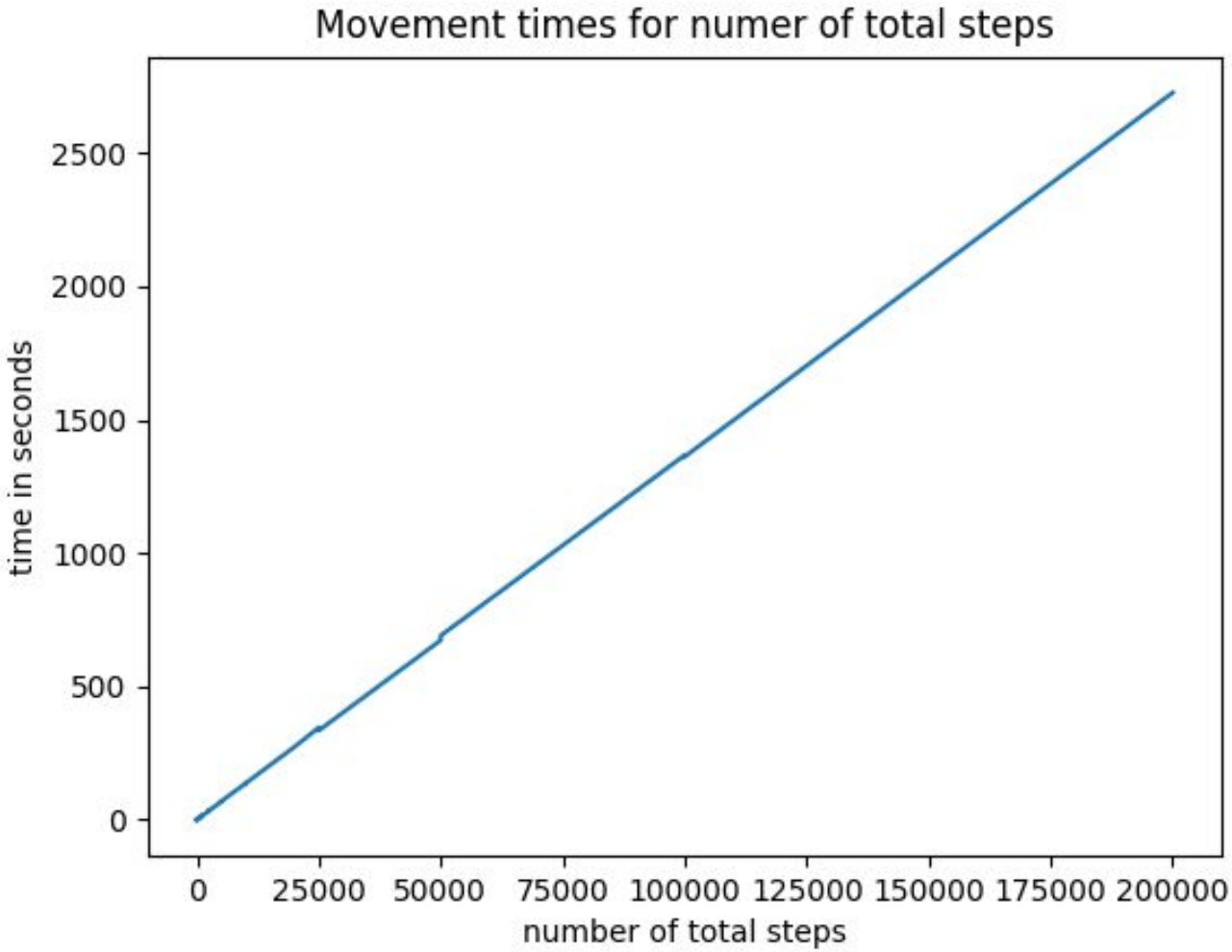
Sparse modelling is a lot more efficient!

Back to our method!

10 animals
100 steps




Movement times for number of steps for different numbers of animals
n_animals
1
5
10
25
50
100
Time in seconds
0
500
1000
1500
2000
2500
0
250
500
750
1000
1250
1500
1750
2000
Number of steps


Movement times for numer of total steps
time in seconds
2500
2000
1500
1000
500
0
0
25000
50000
75000
100000
125000
150000
175000
200000
number of total steps

Performance in Sequential

Linear Scaling with amount of total steps

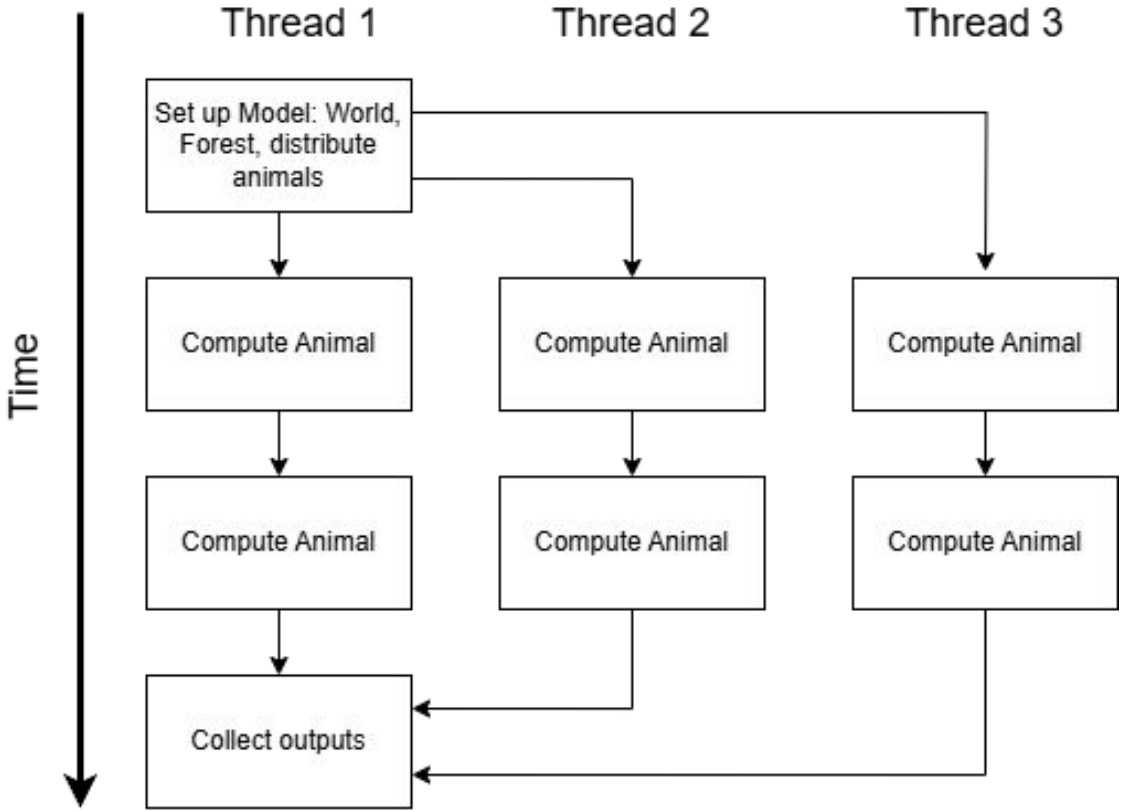
2000 Steps for 100 Animals takes 45 minutes.
→ Too long!

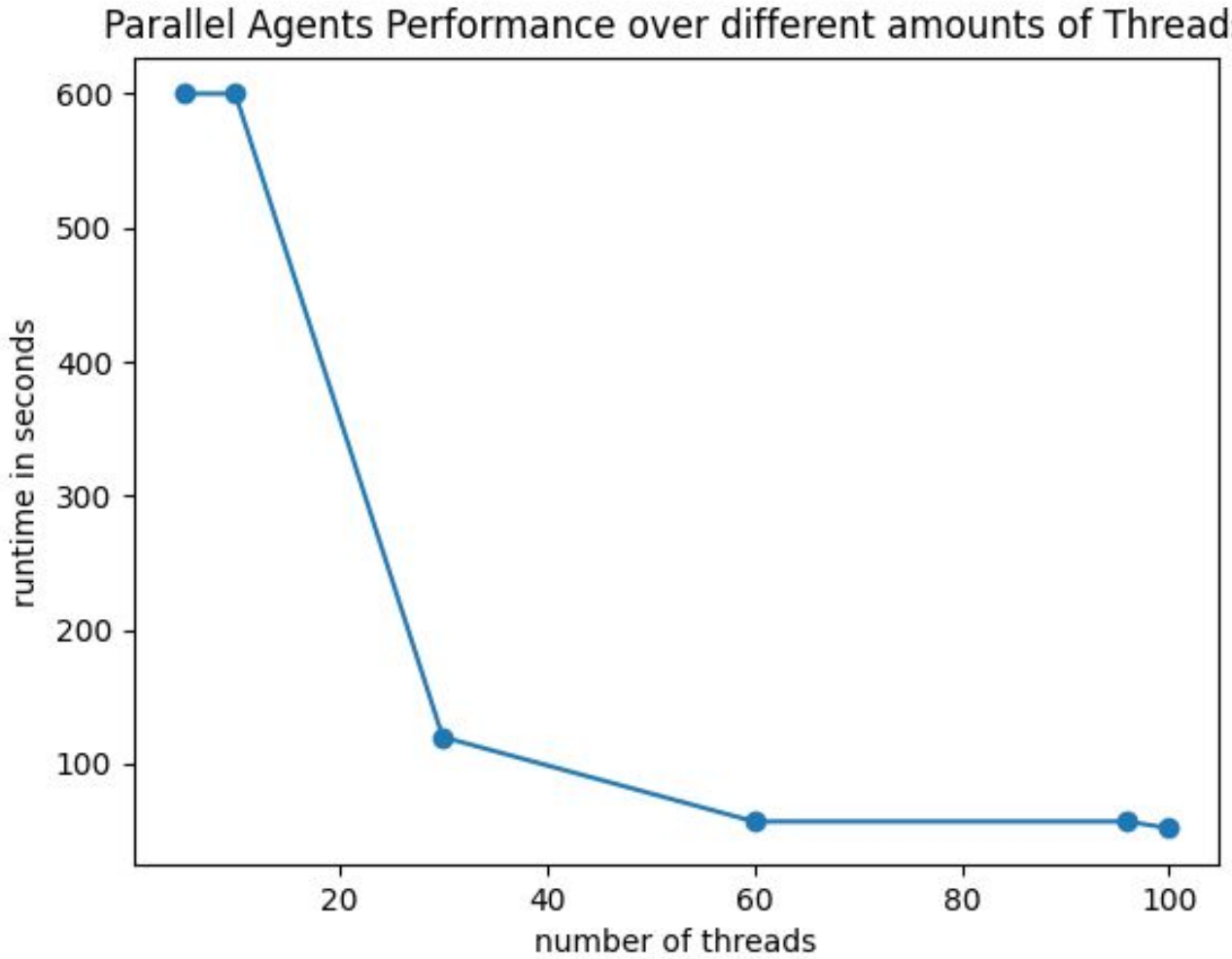
First parallelization Idea: Parallel Agents

- Initialize multiple threads
- Distribute animals evenly to threads
- Let every thread compute steps for their animals
- Gather results (photos, location data, …)

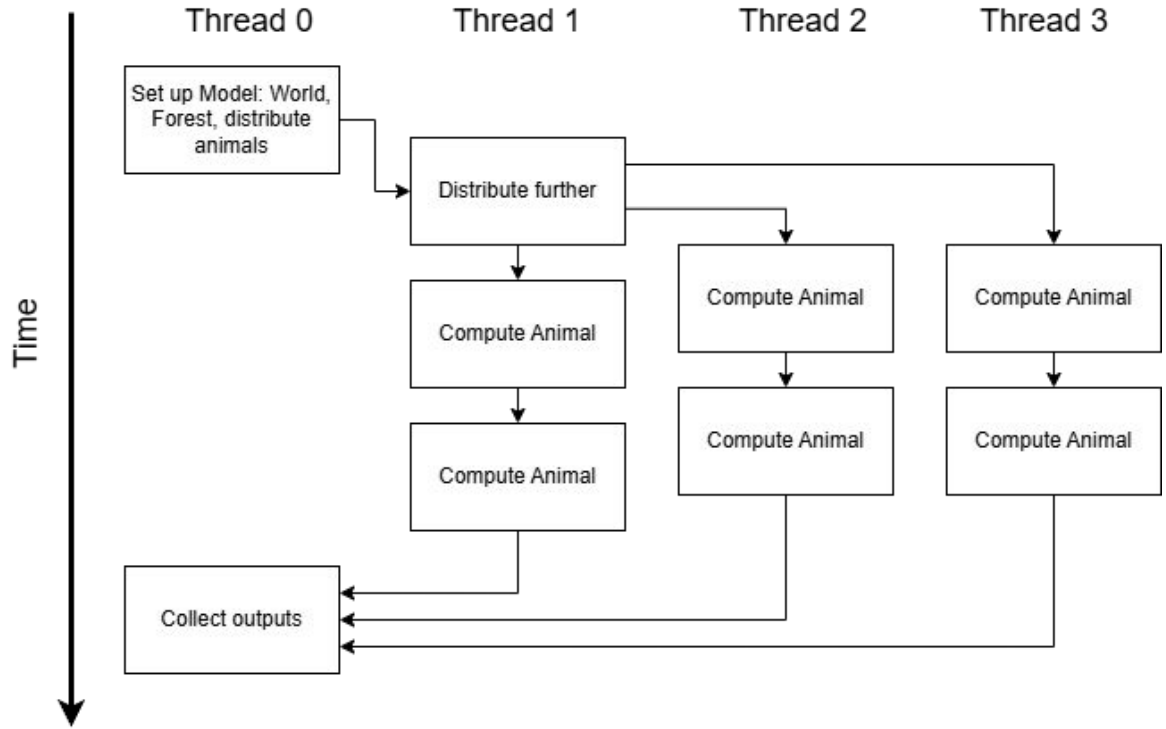
(like the computation of pi)

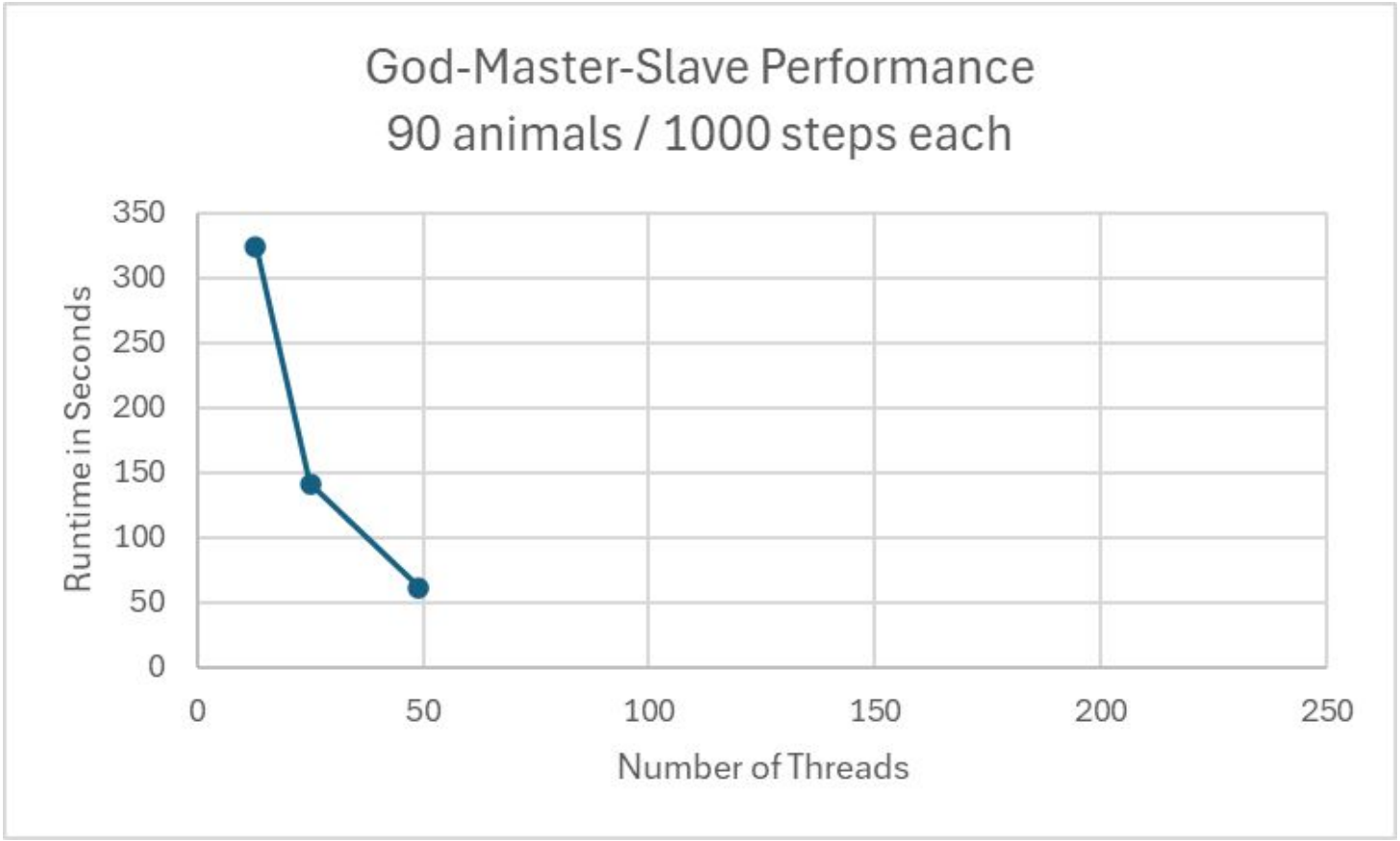
Parallel Agents: Master-Slave




Parallel Agents Performance over different amounts of Threads
runtime in seconds
600
500
400
300
200
100
20
40
60
80
100
number of threads

Parallel Agents: God-Master-Slaves




God-Master-Slave Performance
90 animals / 1000 steps each
Runtime in Seconds
350
300
250
200
150
100
50
0
0
50
100
150
200
250
Number of Threads

Performance Analysis

- looks exponential
- expectation: linear

45 mins = 2700 seconds

2700 seconds / 100 threads = 27 seconds

- our model run took 52 seconds → not linear

What's going on?

Animal partitioning

- animals are discrete units and cannot be split in half
- with 100 animals and 99 threads:
  thread 1 computes 2 animals
  thread 2-100 compute 1 animal
  → Thread 2-100 are idle half of the time

Parallel Agents

- Number of nodes and thus performance capped by number of animals (100)
- embarrassingly parallel!
- Not linear scaling with amount of threads used

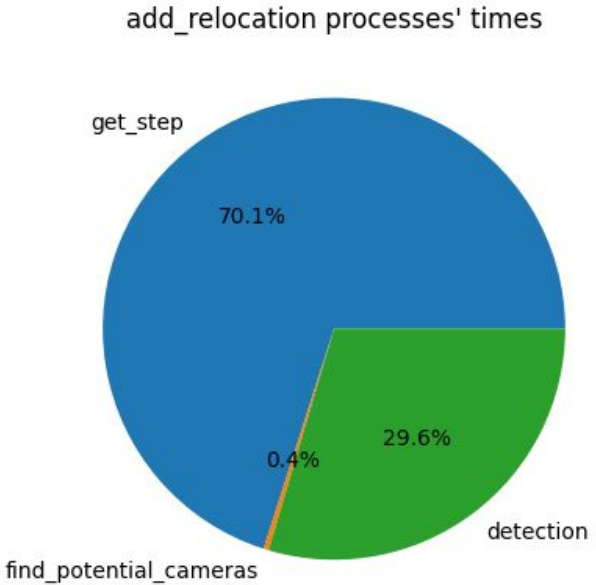
Gain: Good base comparison!

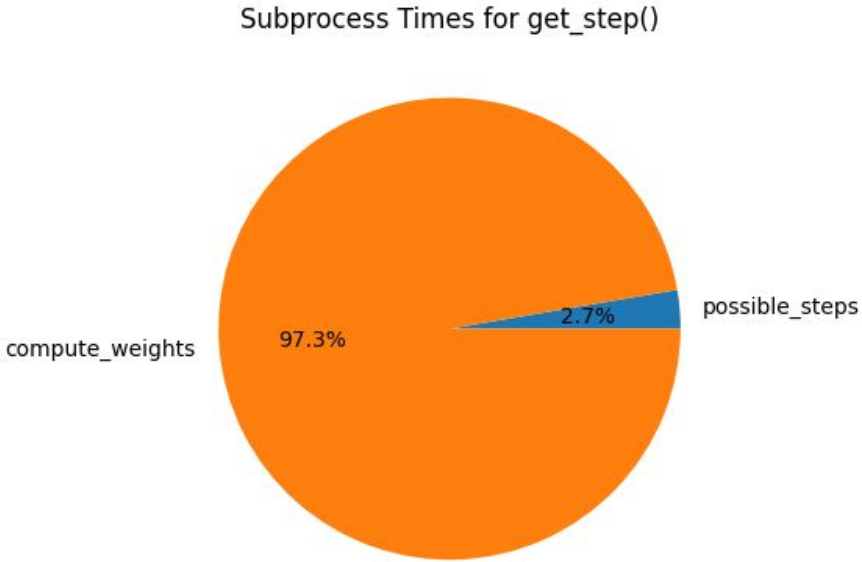
Other Idea: Optimization of thread use

Given n threads for computation, how can you leverage them best for most efficient computation?

Optimization of thread use

From last time:





Optimization of thread use: Idea

Workmaster and worker nodes

Workmaster:

- Generate potential steps
- Send steps to Workers
- Do Bookkeeping (Camera detections and more data collection)
- Get Weights from Workers and determine next step

Worker:

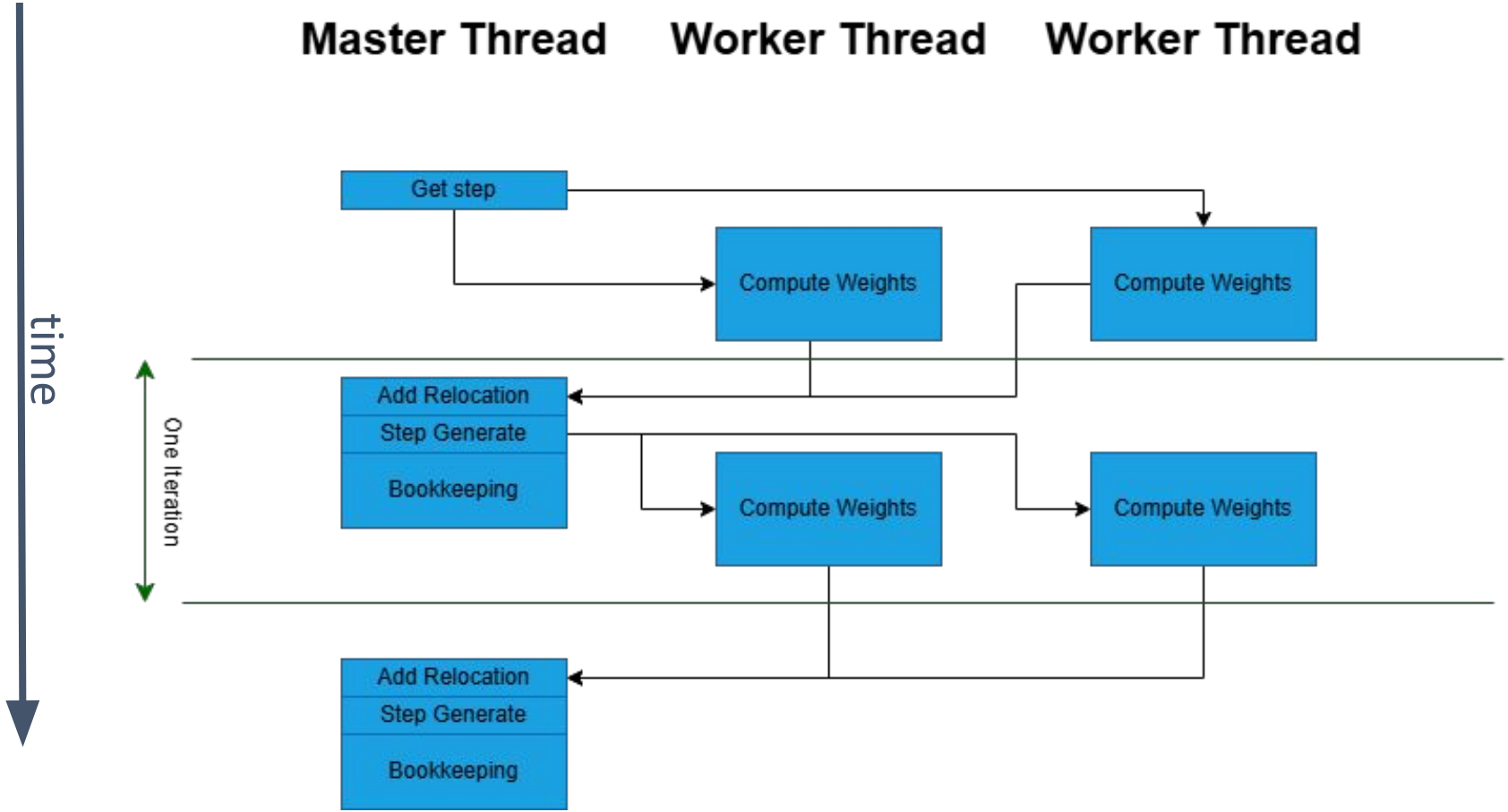
- Get potential steps from Workmaster
- Calculate Weights
- Send Weights for steps to Workmaster

Optimization of thread use: Idea

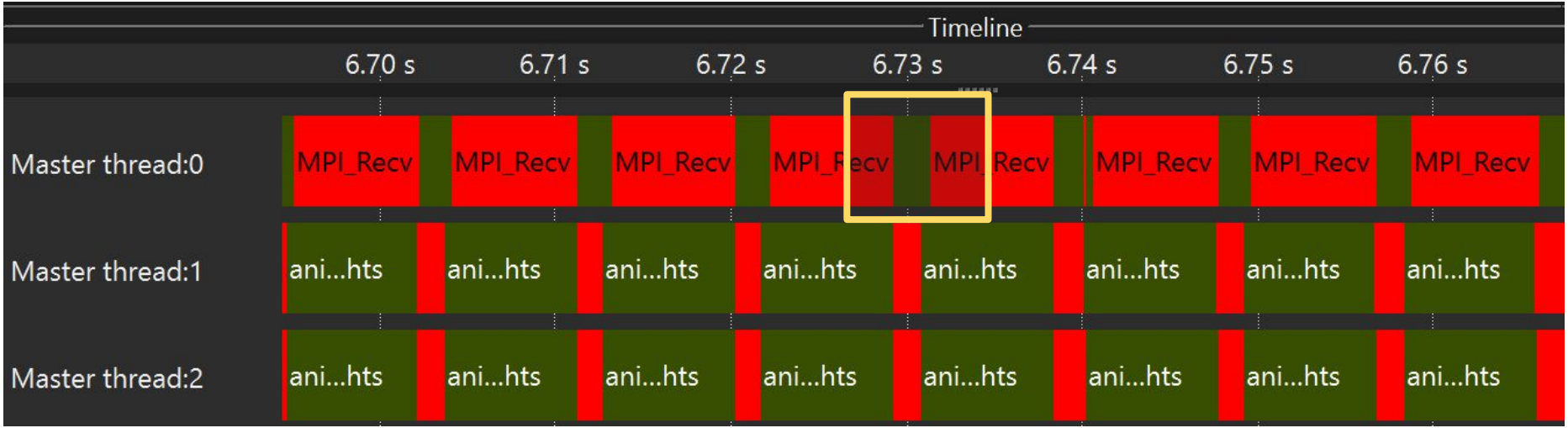
Workmaster and worker nodes

Weight calculation ⅔ of time ⇒ 1 Master, 2 Workers

Master Worker Approach



Vampir Performance Analysis

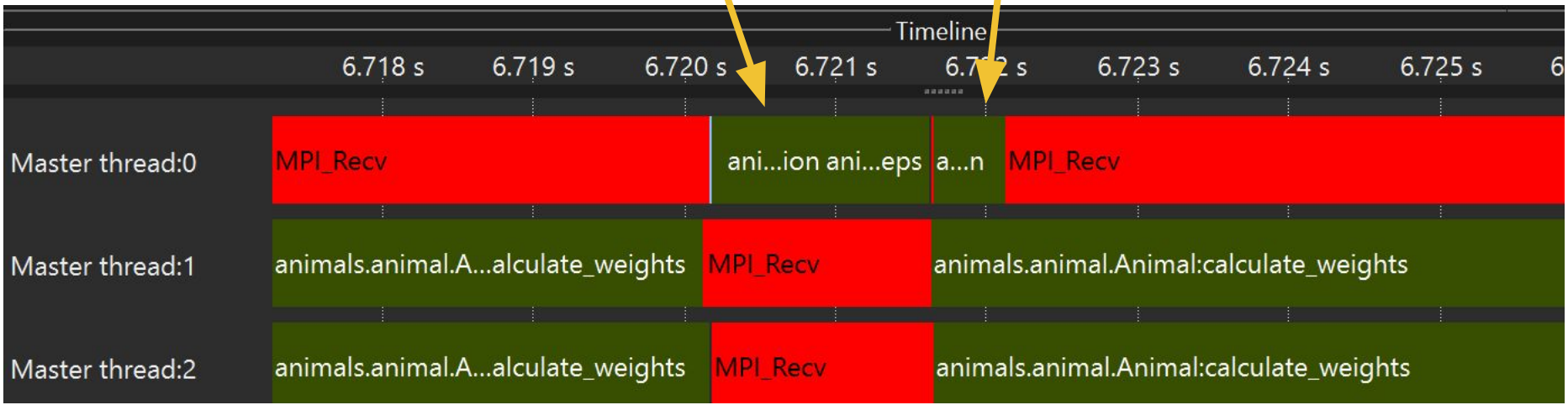


= waiting

= doing something

computing next position and generating steps

bookkeeping



still a lot of waiting

A lot of waiting?

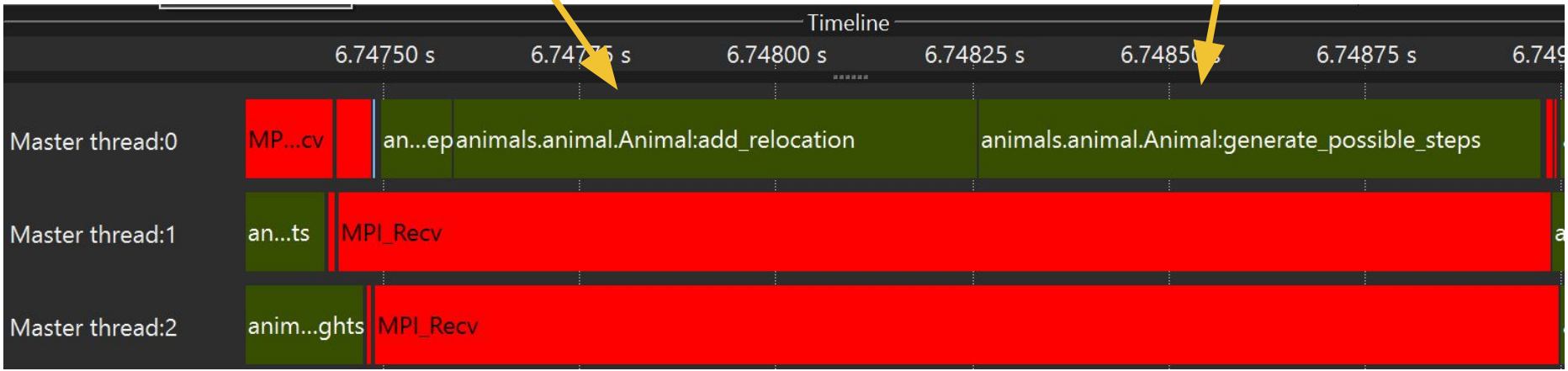
0.002 seconds of waiting x 2000 steps = 4 seconds of waiting per animal

⇒ 4 seconds per animal x 100 animals = 400 seconds of waiting

Master Worker v2

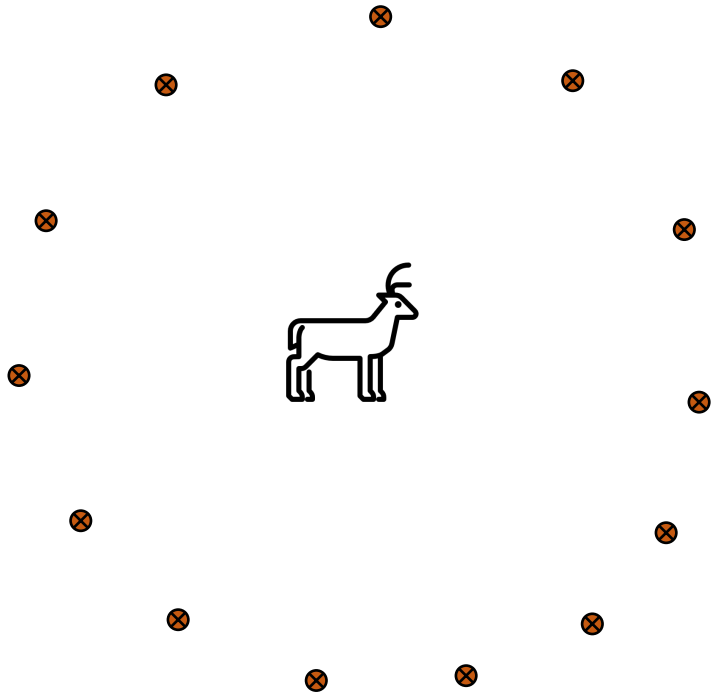
depends on weights

depends on location

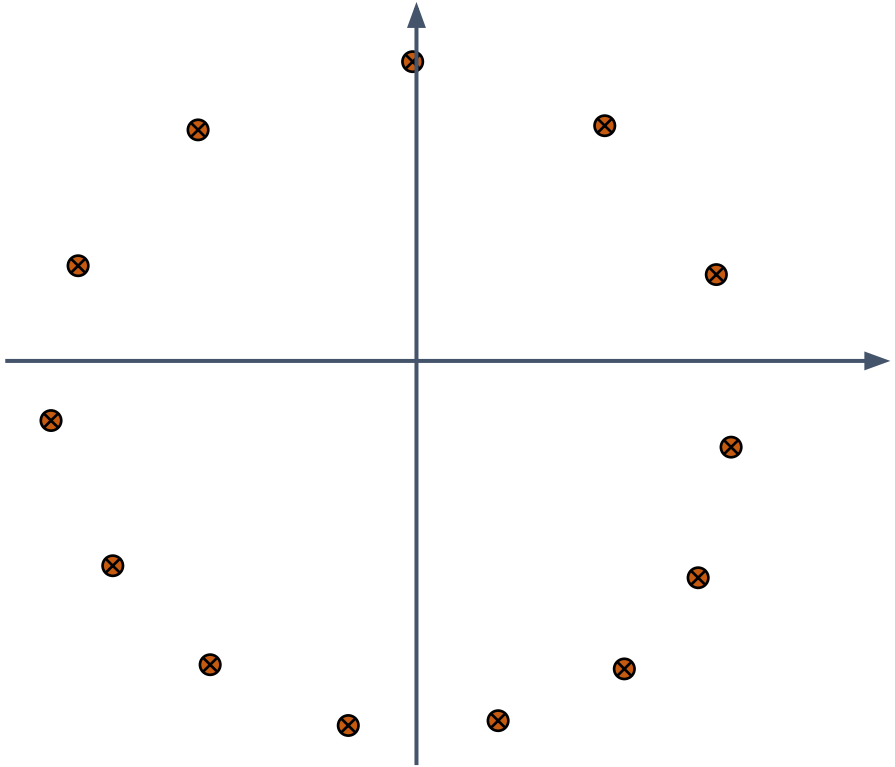


Idea: Can Master do preparatory substeps while he is waiting?

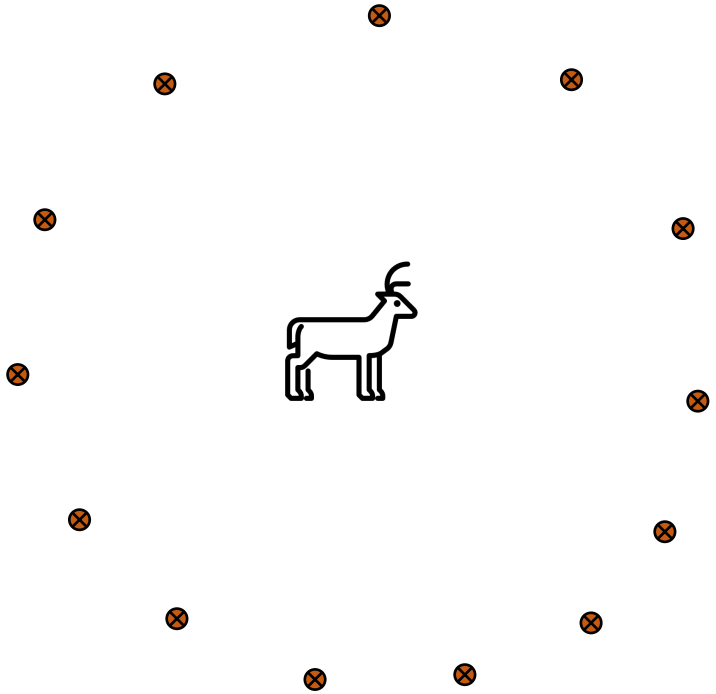
Movement Model

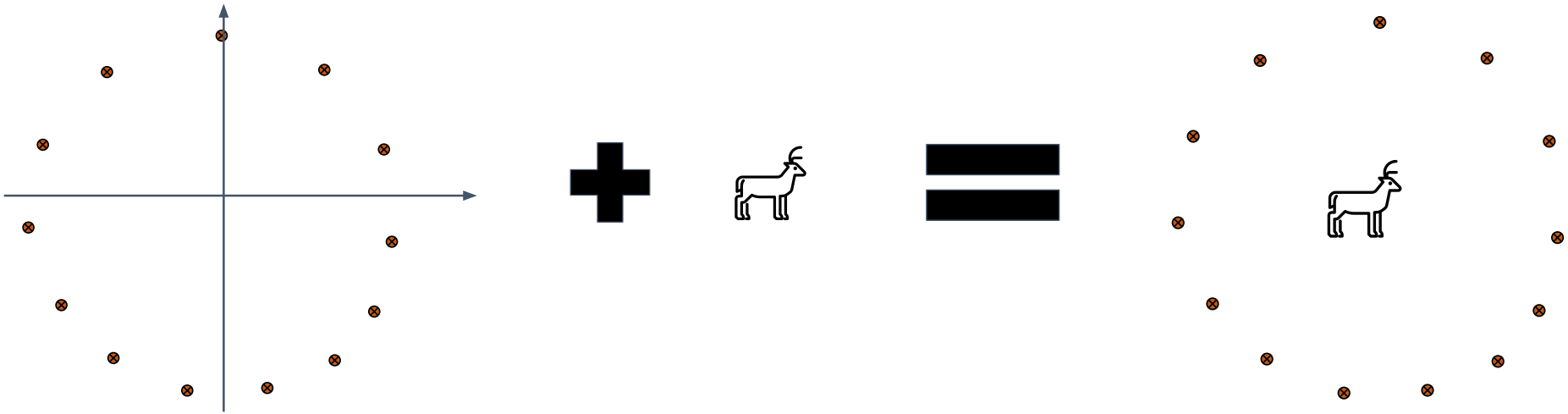


Movement Model

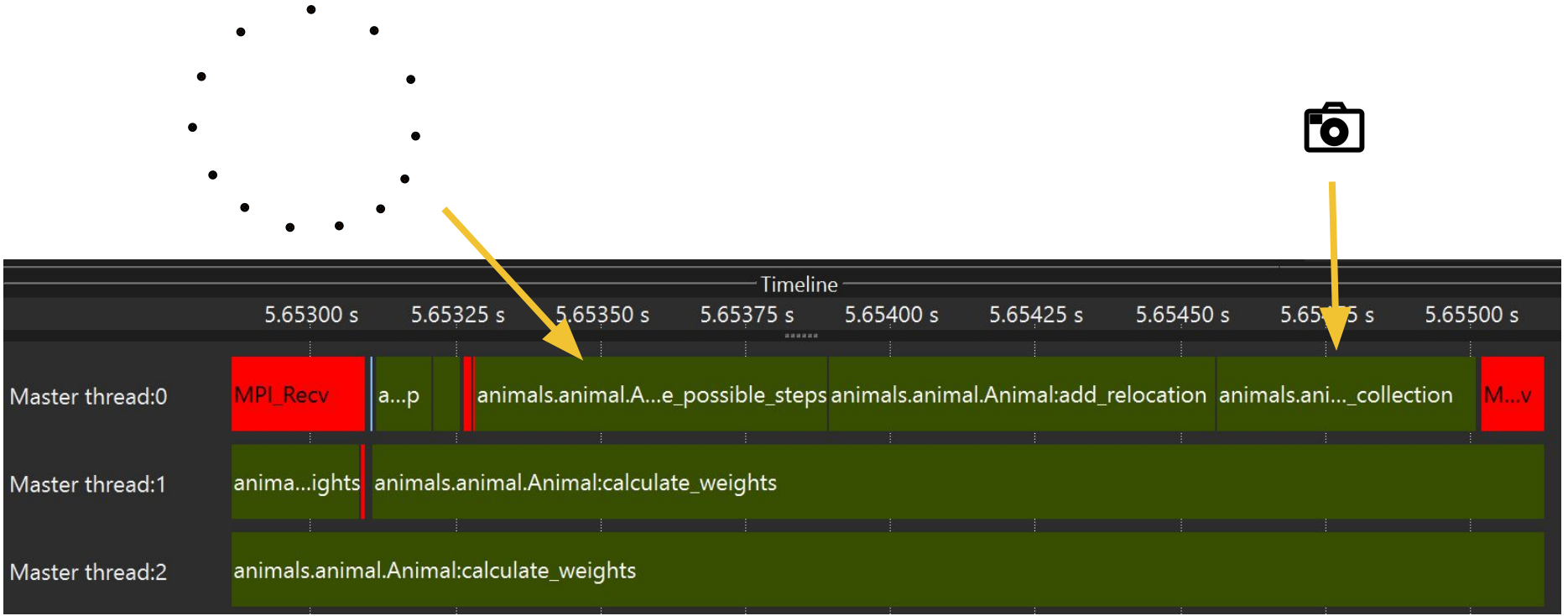


Movement Model

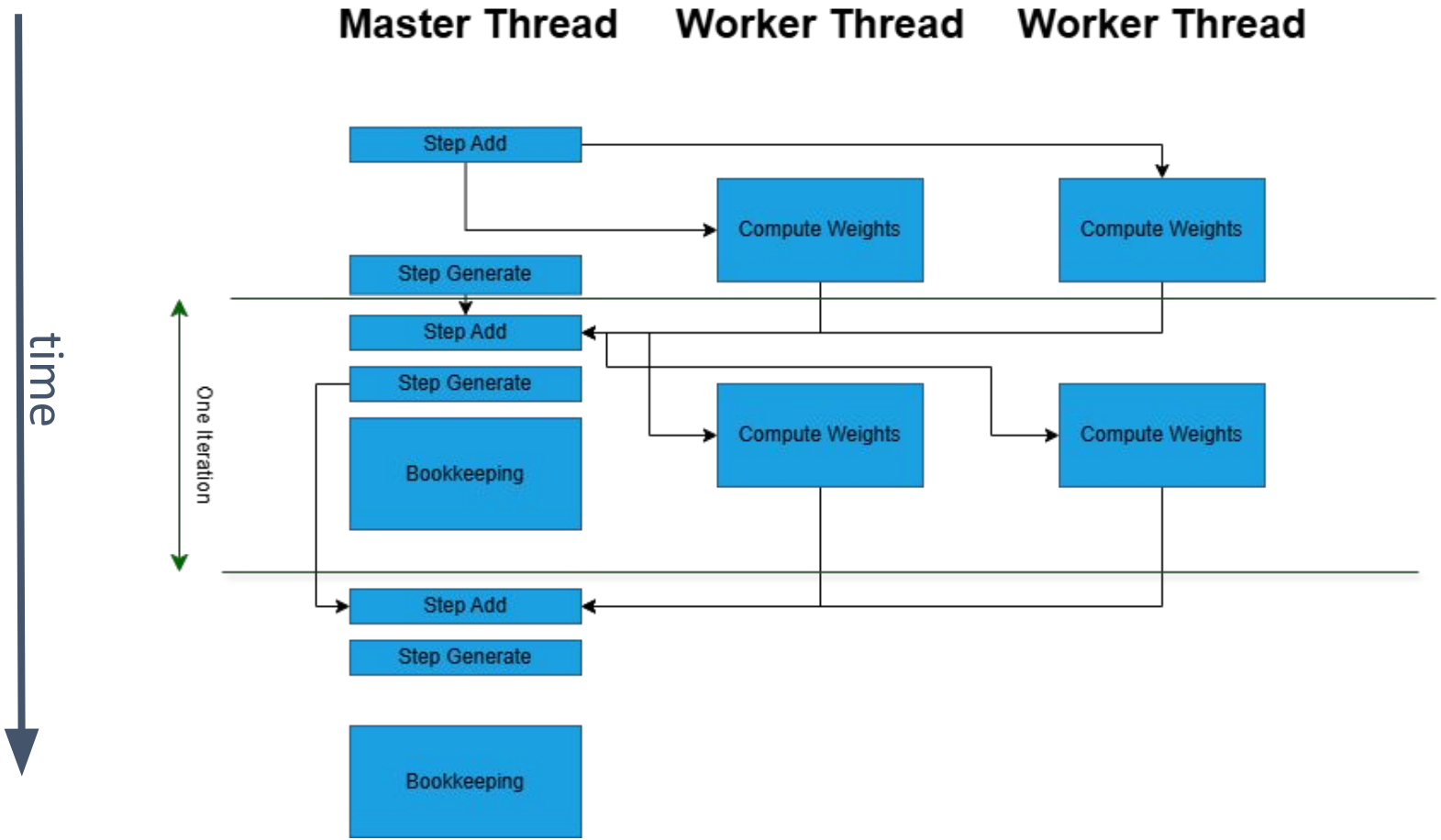




generate possible steps during weight calculation



Our Chart looks like this by now …



What we achieved:

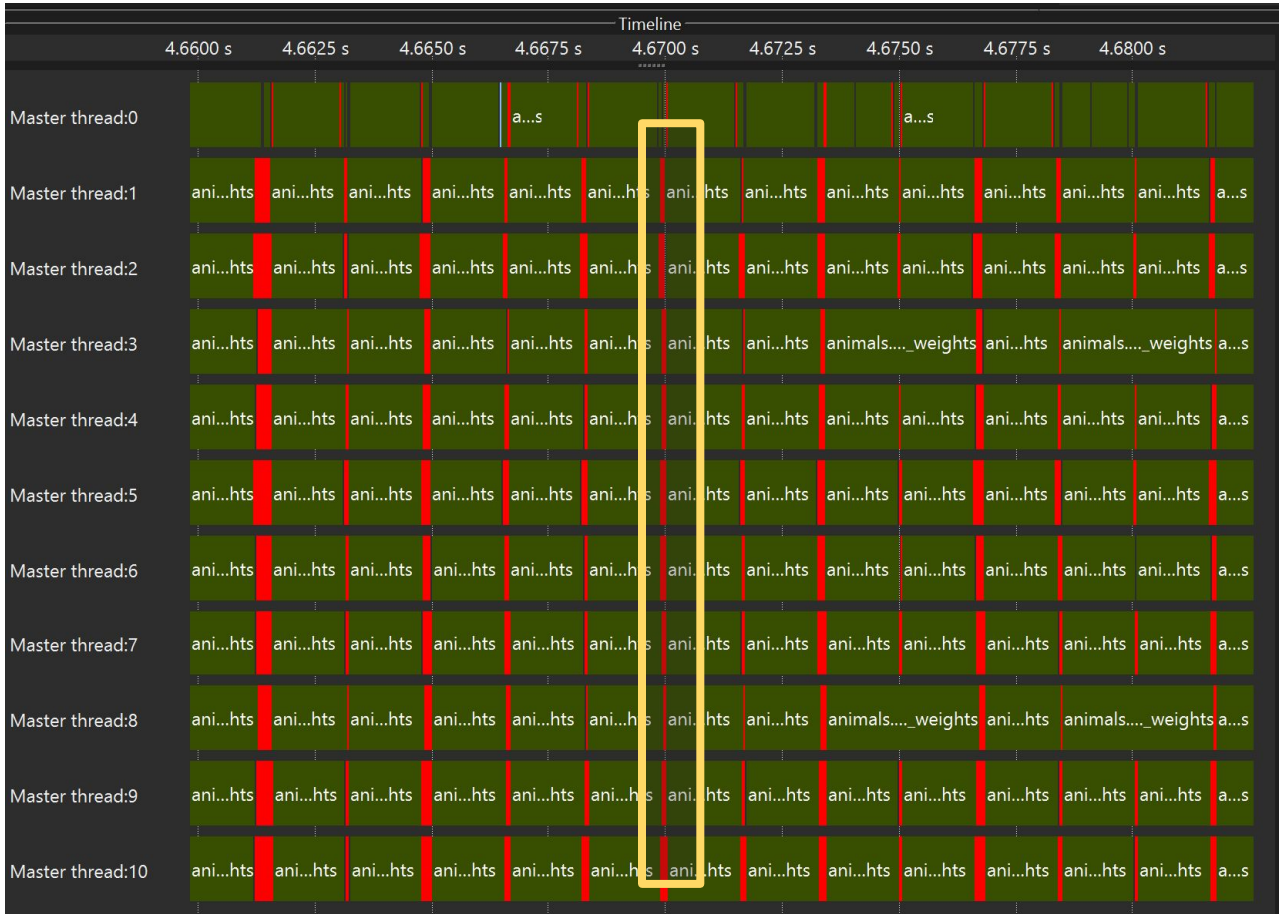
Workers are barely idle

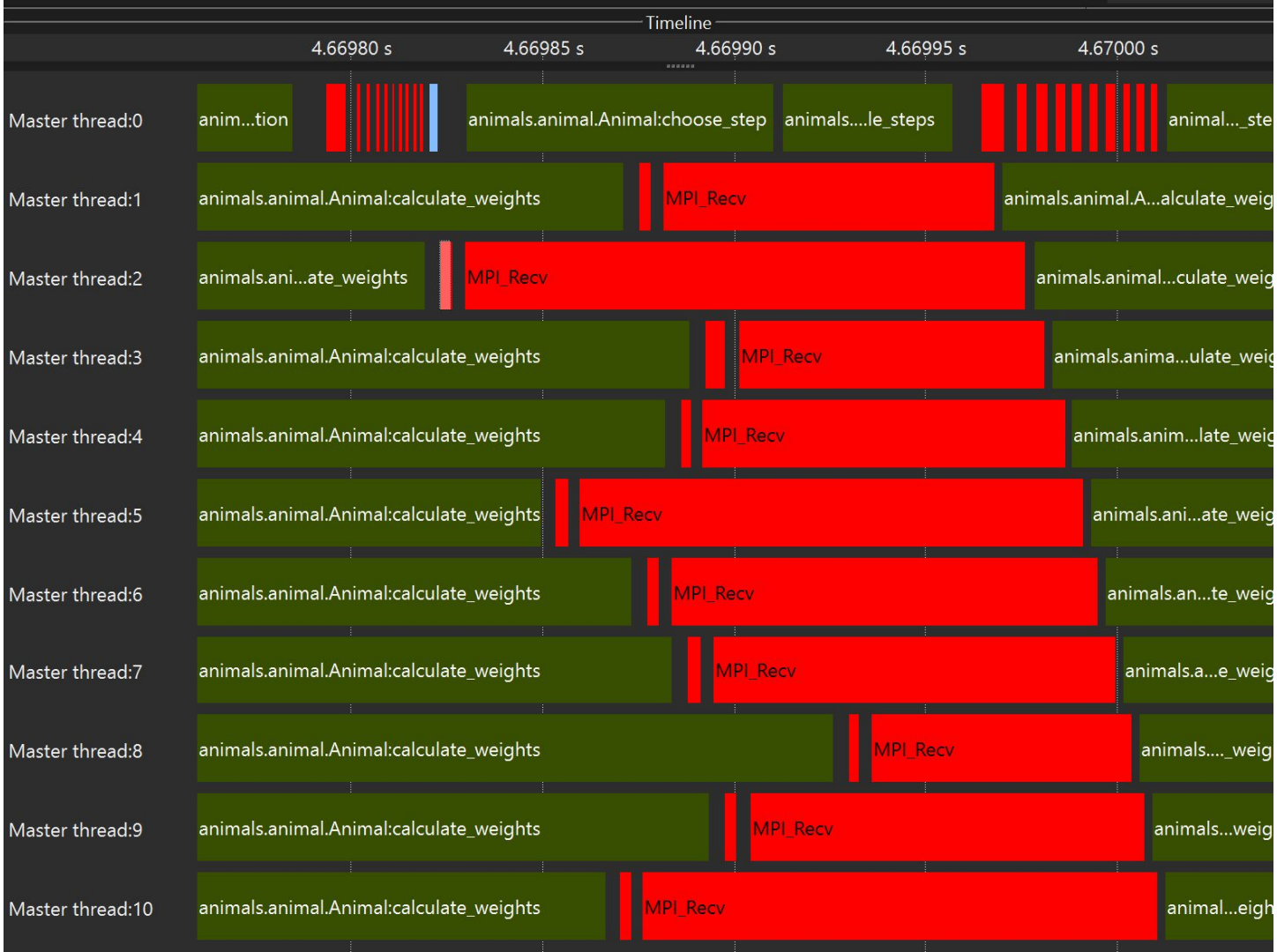
but Master is Idle a lot

Solution: Find optimal amount of Workers per Master

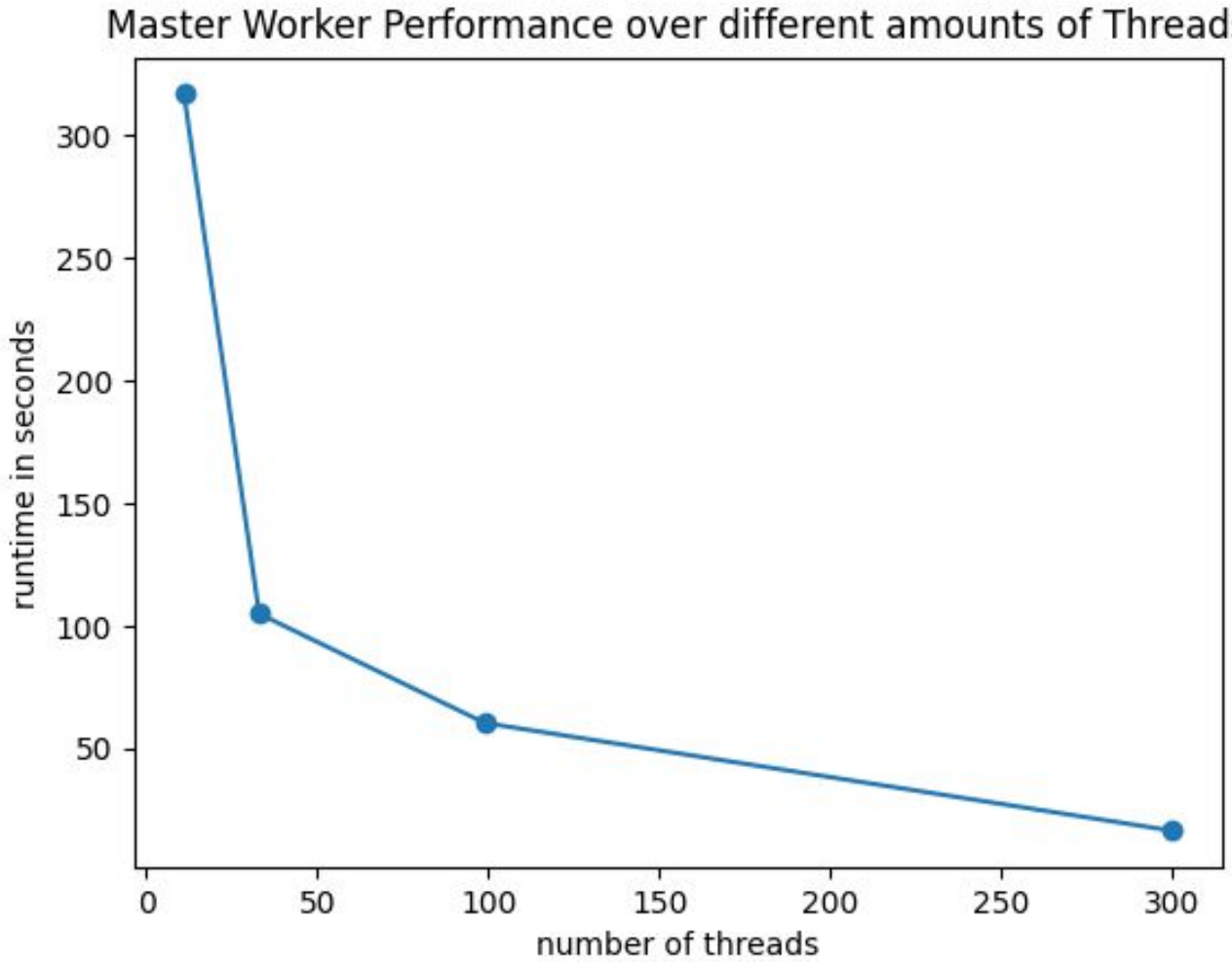
10 Workers per Master is optimal

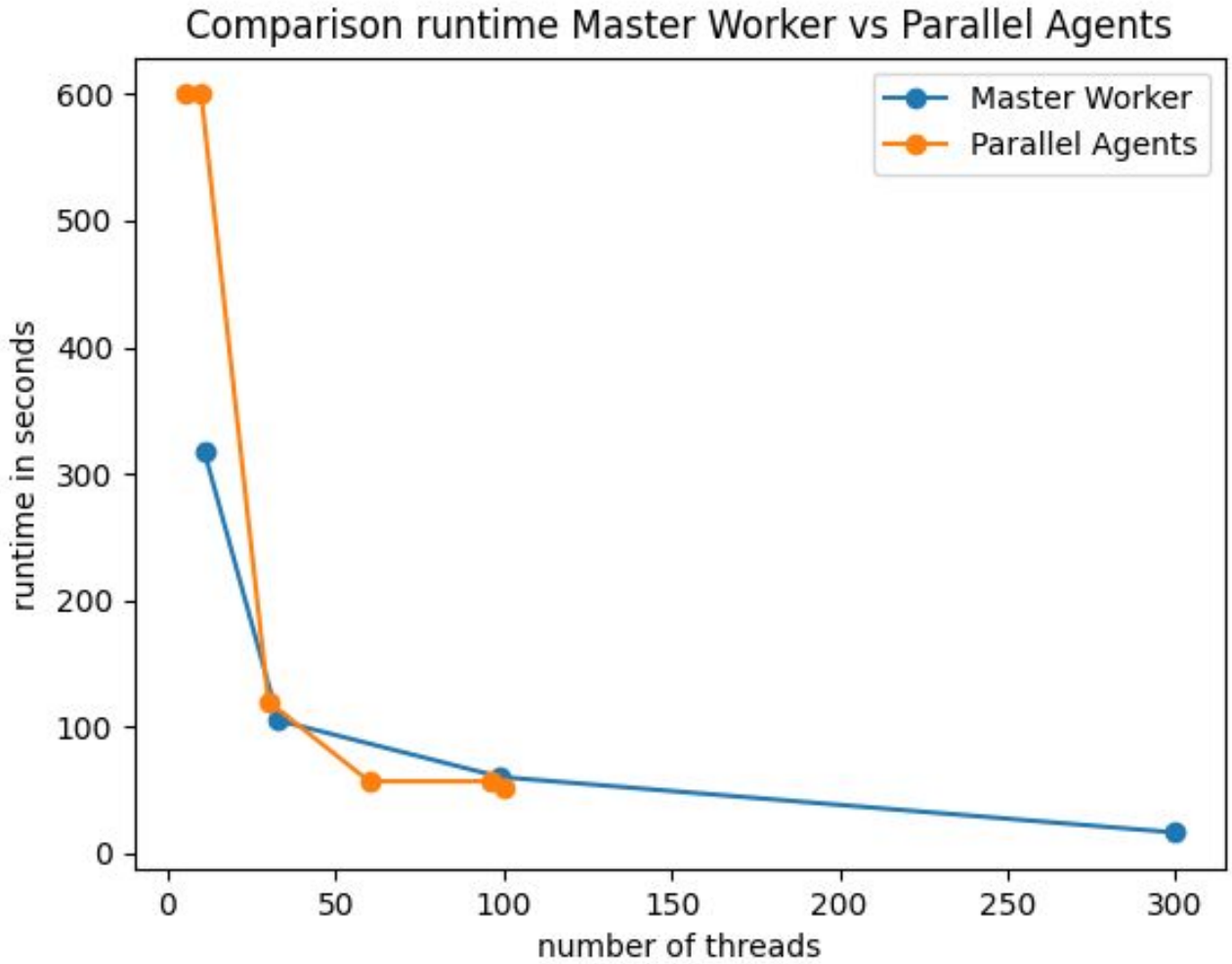
Lots of green! (little idling)



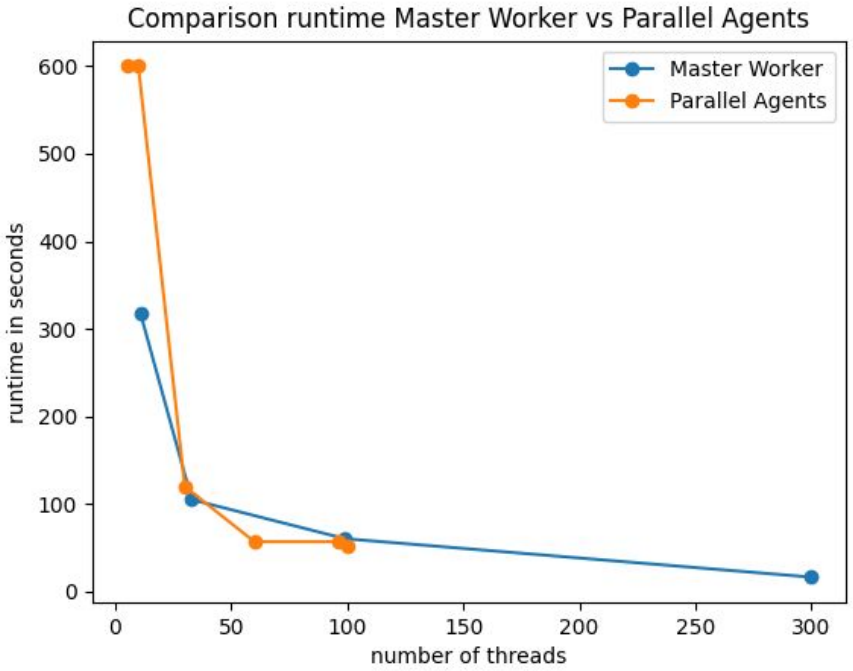

Timeline
4.66980 s
4.66985 s
4.66990 s
4.66995 s
4.67000 s
Master thread:0
anim...tion
animals.animal.Animal:choose_step
animals....le_steps
animal..._ste
Master thread:1
animals.animal.Animal:calculate_weights
MPI_Recv
animals.animal.A...alculate_weig
Master thread:2
animals.ani...ate_weights
MPI_Recv
animals.animal...culate_weig
Master thread:3
animals.animal.Animal:calculate_weights
MPI_Recv
animals.anima...ulate_weig
Master thread:4
animals.animal.Animal:calculate_weights
MPI_Recv
animals.anim...late_weig
Master thread:5
animals.animal.Animal:calculate_weights
MPI_Recv
animals.ani...ate_weig
Master thread:6
animals.animal.Animal:calculate_weights
MPI_Recv
animals.an...te_weig
Master thread:7
animals.animal.Animal:calculate_weights
MPI_Recv
animals.a...e_weig
Master thread:8
animals.animal.Animal:calculate_weights
MPI_Recv
animals...._weig
Master thread:9
animals.animal.Animal:calculate_weights
MPI_Recv
animals...weig
Master thread:10
animals.animal.Animal:calculate_weights
MPI_Recv
animal...eigh

Performance


Master Worker Performance over different amounts of Threads
runtime in seconds
300
250
200
150
100
50
0
50
100
150
200
250
300
number of threads


Comparison runtime Master Worker vs Parallel Agents
Master Worker
Parallel Agents
runtime in seconds
number of threads
0
100
200
300
400
500
600
0
50
100
150
200
250
300

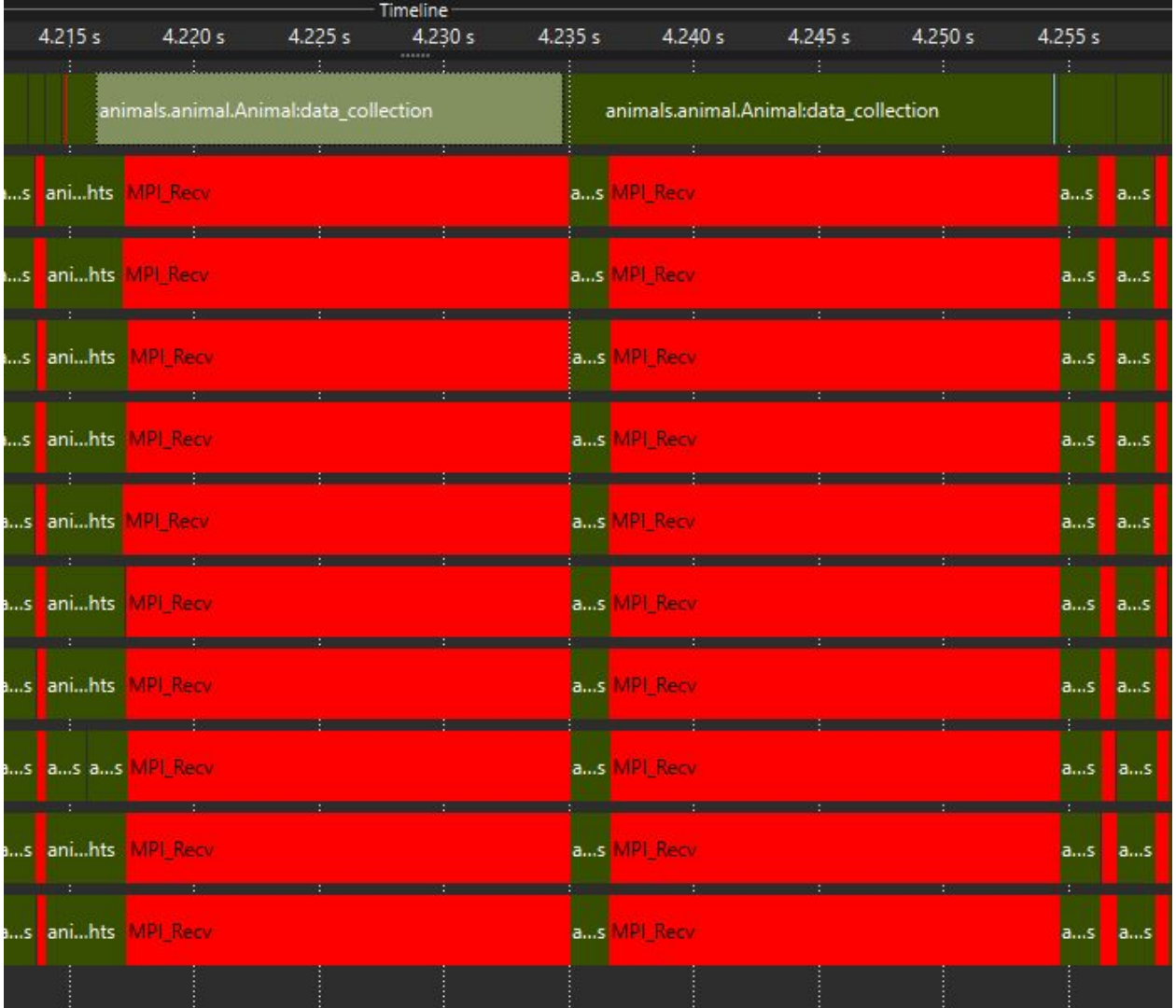
- Master Worker more efficient for lower thread numbers
- Master Worker possibility for more threads employed
- Parallel Agents more efficient around 50-100 threads



Performance

- Not linear either
- Capped by number of Animals x Threads per Workgroup (100 x 11 = 1100)
- Number of Threads must be divisible by Threads per Workgroup
- 300 threads $\longrightarrow$ 24 seconds runtime

What about photos?


Timeline
4.215 s
4.220 s
4.225 s
4.230 s
4.235 s
4.240 s
4.245 s
4.250 s
4.255 s
animals.animal.Animal:data_collection
animals.animal.Animal:data_collection
MPI_Recv

Problem

10 Workers waiting for 1 Master for a “long” time

Idea:
For multiple Workgroups (Master + 10 Workers) assign one “Photographer”

Master Worker v3

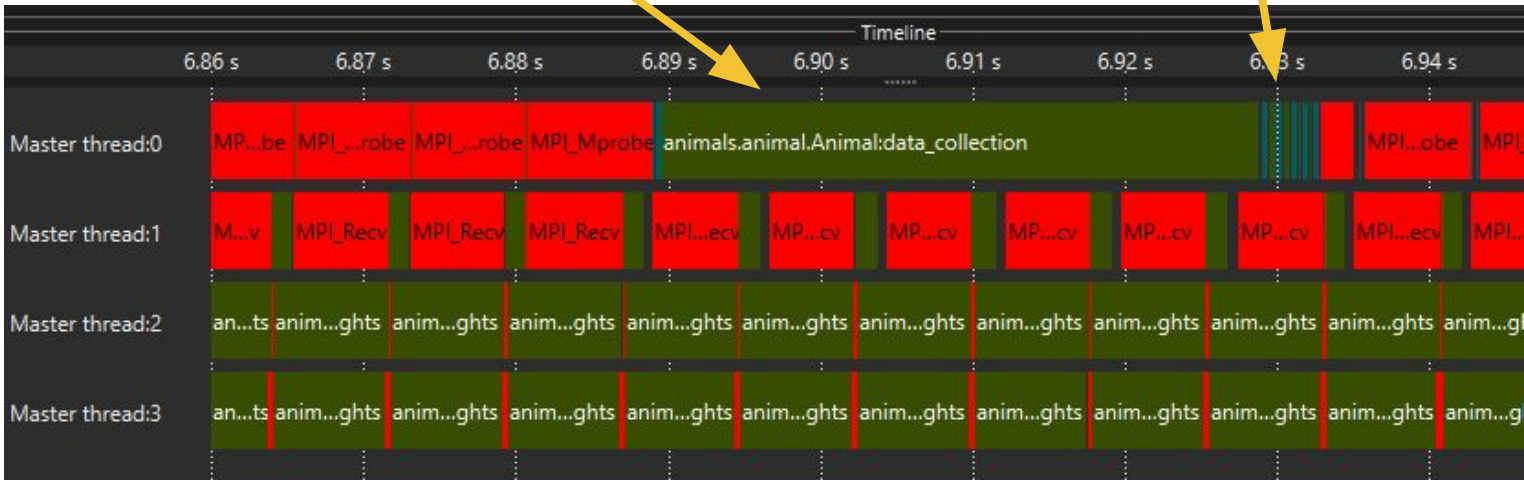
taking photo while animals keep running

catching up with other bookkeeping

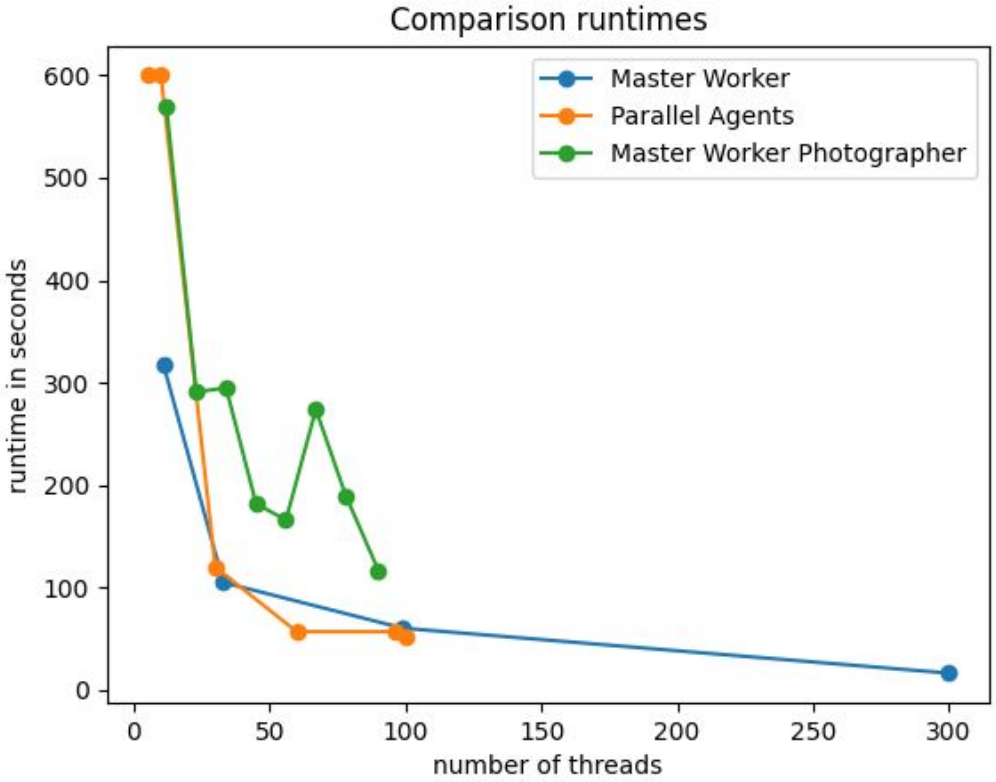
Photographer

Master

Workers



Is this quicker?



Is this quicker?

No.

Challenges:

- a lot of communication overhead
- volatile and instable
- hard to implement (MPI buffer overwriting problem)

⇒ Previous solution was better

Conclusion

Conclusion

- didn’t get linear growth in speed by resources
- reached goal: 24 seconds < 45 seconds
- Parallel Agents approach is already very quick
- Master Worker v2 is the most efficient and can scale longer

Key Lessons

- parallel programming is highly variable and unpredictable (race conditions, etc.)
  ⇒ Make the program Waterproof
- Vampir is great to see weaknesses and if the program even works properly
- environment setup on the cluster is complex (python)
- Often the easiest solutions are the best (80:20 rule)
- dividing into subfunctions is useful for parallelizing already existing sequential code
- parallel programming is fun

Thank you!

Questions?