VISUALIZATION OF CIRCLE COLLISIONS USING QUAD- AND OCTA-TREES

HPC-PROJECT

BY ELIAH AND EVGENI

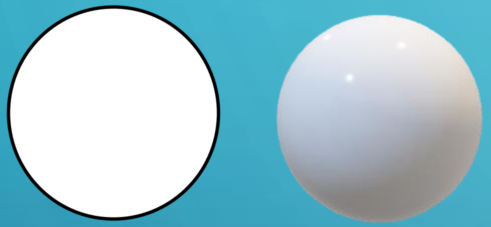
PRESENTATION OUTLINE

- MOTIVATION
- PROBLEM DESCRIPTION
- APPROACHES
- PERFORMANCE ANALYSIS
- CONCLUSION
- OUTLOOK

MOTIVATION

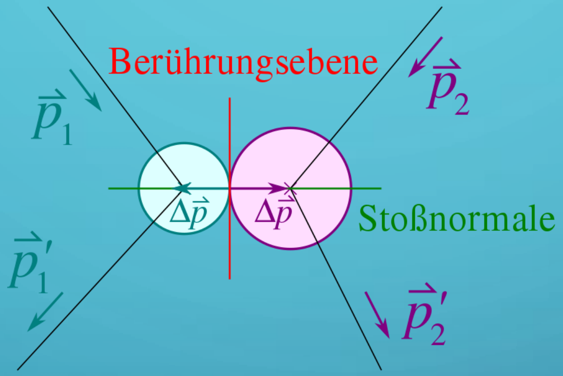
- Physics Simulations
- Swarm Robotics
  - Drones, Cars
- Computer Games
- It's fun

PROBLEM DESCRIPTION



2D INSTEAD OF 3D

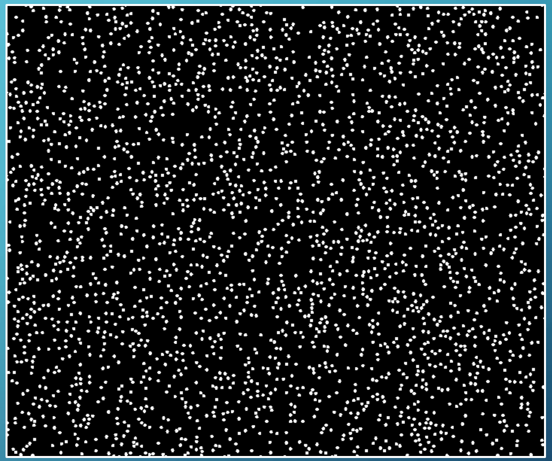
We reduced the dimensions to 2



COLLISIONS

Check the collision between circles and with the wall

All circles have the same size and same mass



THE AMOUNT OF CIRCLES

The amount of cirlces increases the amount of collision checks drastically

APPROACHES

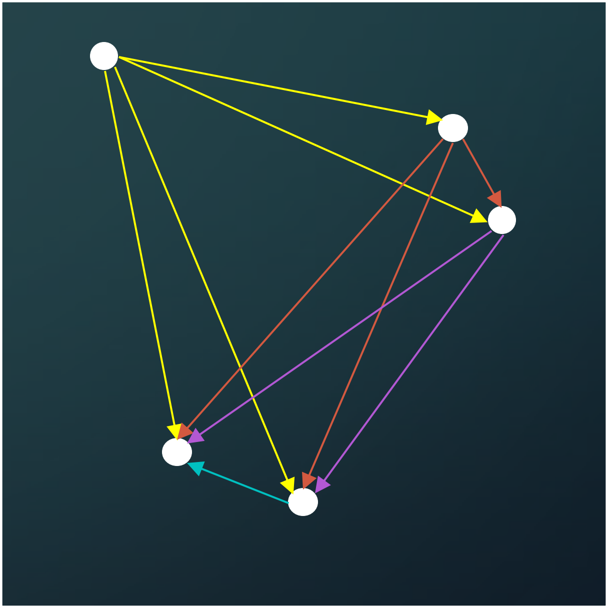
NAÏVE IMPLEMENTATION

Main Loop

```
for i from 0 to numCircles-1:
|  for j from i + 1 to numCircles:
|  |  checkCollision(i, j)

for i from 0 to numCircles:
|  move(i)
```

Check Collision

```
if distance(circle1, circle2) < 2r:
|  calcVelocities(circle1, circle2)
|  resolveOverlap(circle1, circle2)
```

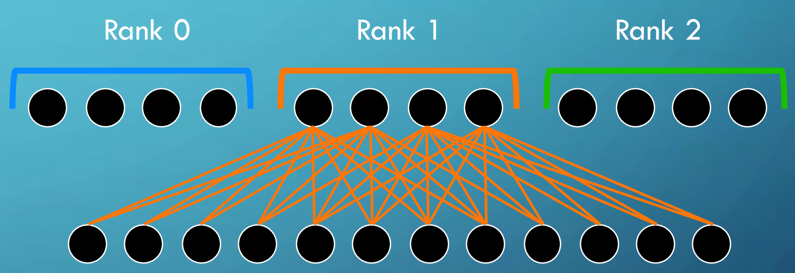


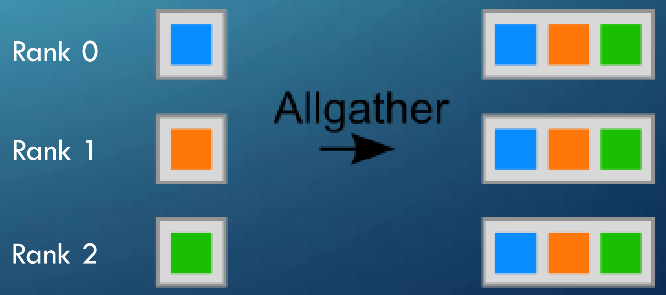
NAÏVE – WITH MPI

- Parallelize the outer main loop
- Each process only calculates the collisions for its own circles

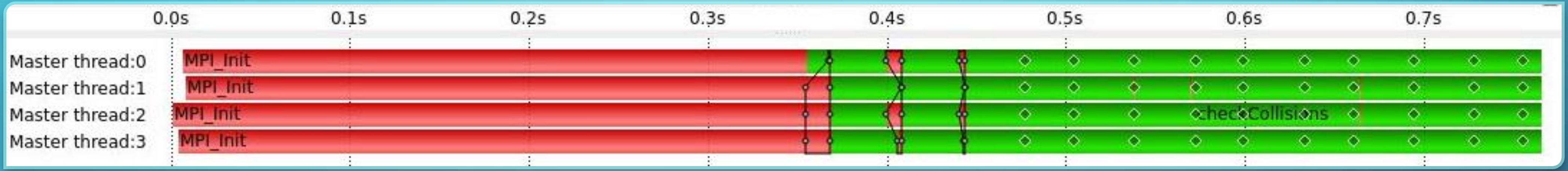
Main Loop

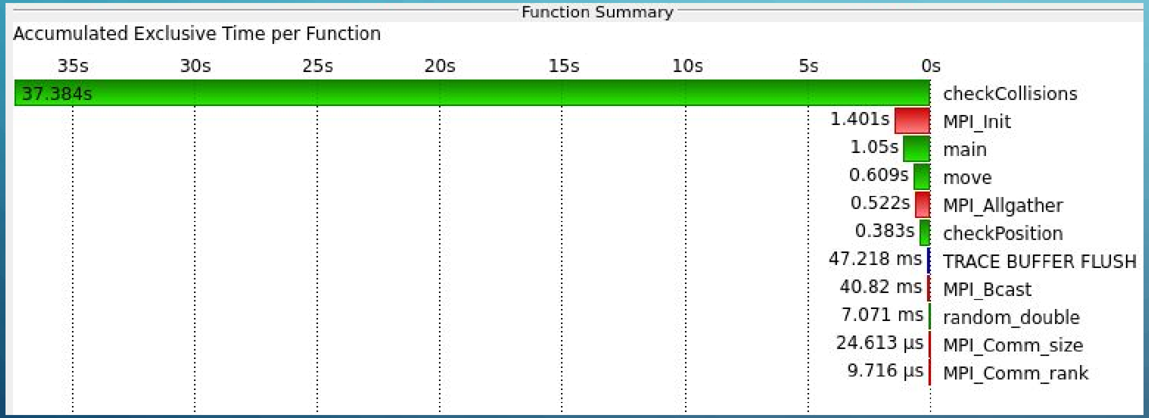
```
h = numCircles / world_size
for i from rank * h to (rank+1) * h:
| for j from 0 to numCircles:
| | checkCollision(i, j)

for i from 0 to numCircles:
|  move(i)
MPI_Allgather(&circles[rank * h], h)
```





NAÏVE – WORKLOAD DISTRIBUTION



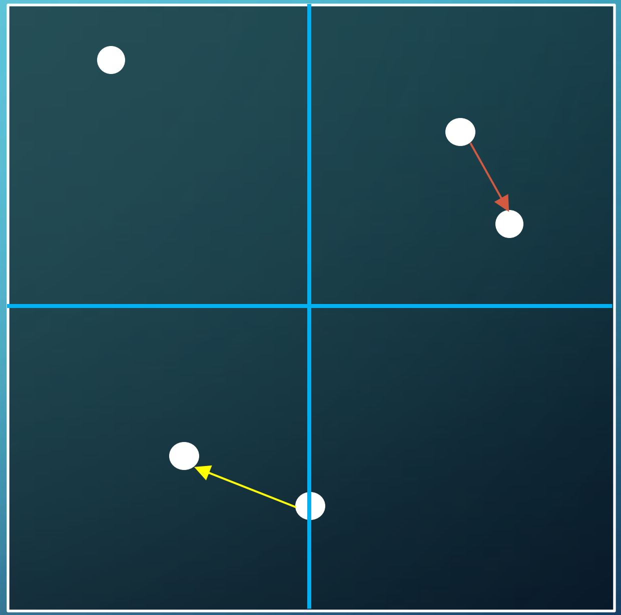


- Most of the calculation is in the checkCollisions function
- Communication is only a small overhead

THE TREE

- Split field in cells
- Assign circles to cells
- Check collisions in every cell

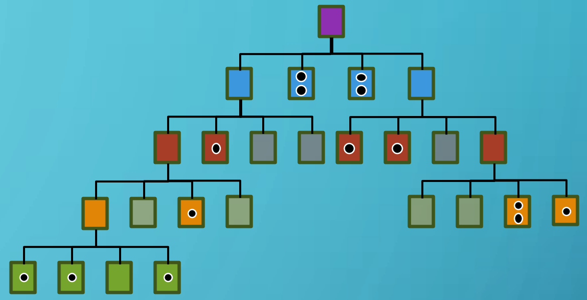
Benefits	Drawbacks
Less collision checks	*Tree creation*

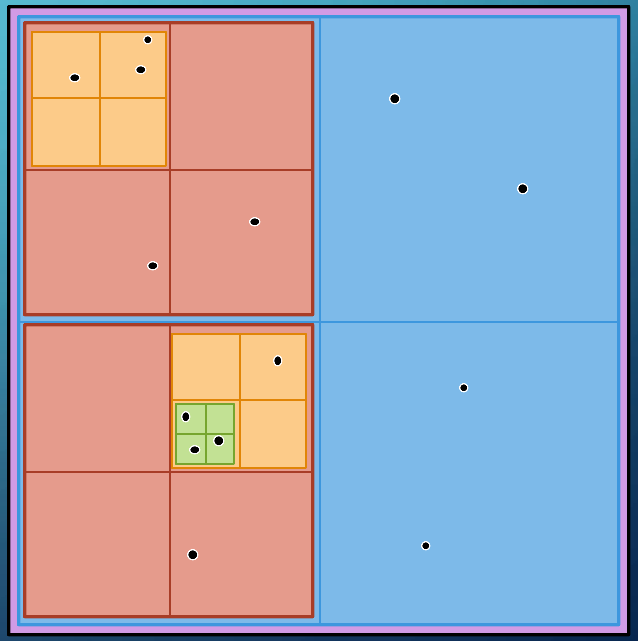


STATIC TREE

Add Circle To Tree (circle, cell)

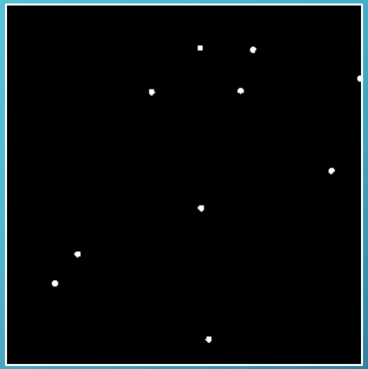
```
if cell.isLeaf:
|   if cell.numCirclesInCell < 2:
|   |   cell.circles.append(circle)
|   else:
|   |   split(cell)
|   |   addCircleToTree(circle, cell)
else:
|   for i from 0 to 4:
|   |   if circleOverCell(circle, subcell[i])
|   |   |   addCircleToTree(circle, subcell[i])
```

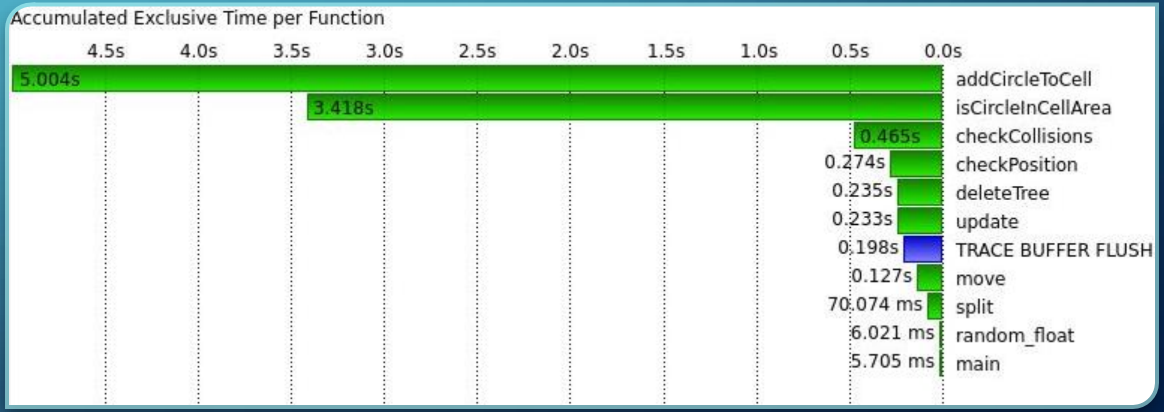




STATIC TREE - WORKLOAD DISTRIBUTION

- Main workload on tree building
- checkCollision needs much less time
- Wasteful to delete the tree and rebuild it for every frame





DYNAMIC TREE

The idea:

- Don't reconstruct the tree

→ Reuse and update: split/collapse

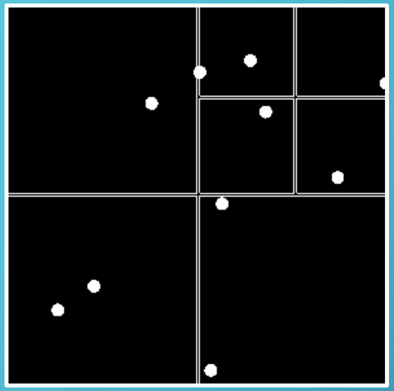
- Don't "waste" perfectly fine subtrees

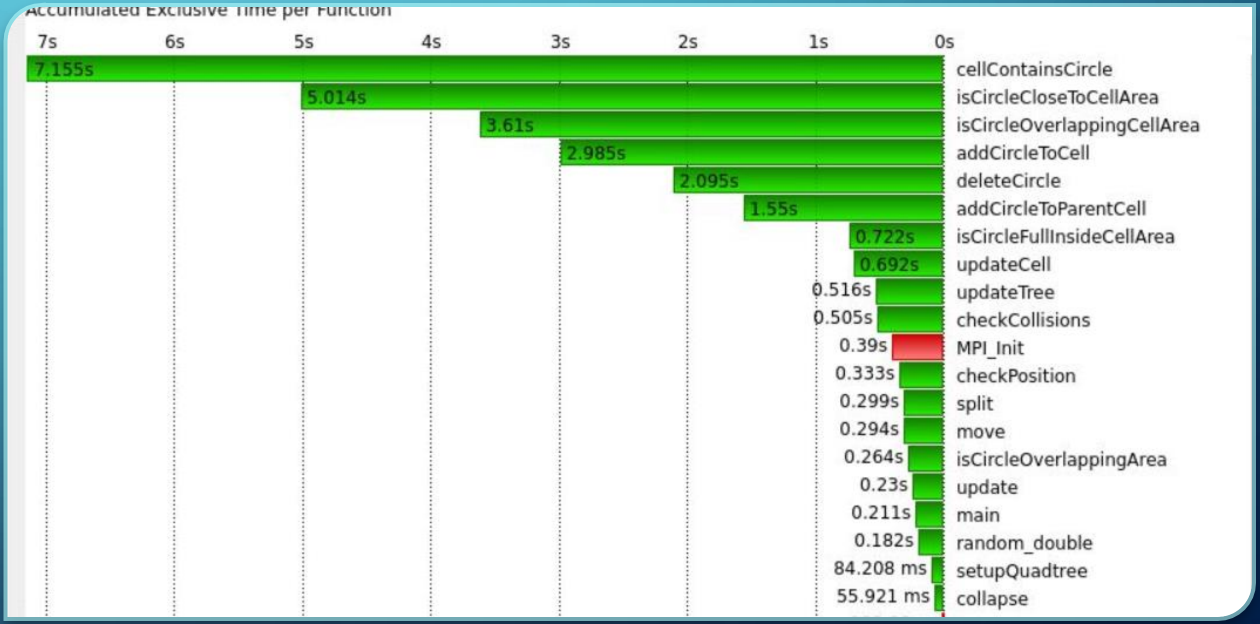
Problems:

- Implementation
- Recursive functions are harder to debug
- Exception thrown multiple iterations after error occured

DYNAMIC TREE - WORKLOAD DISTRIBUTION

- The dynamic functions are costing a lot of time
- Multiple tree traversals per frame
- There might be a better implementation we haven't found yet

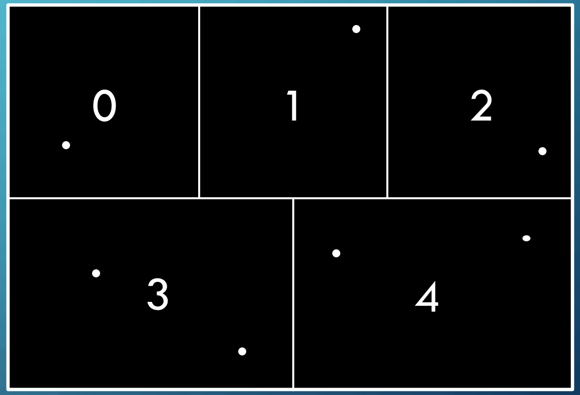


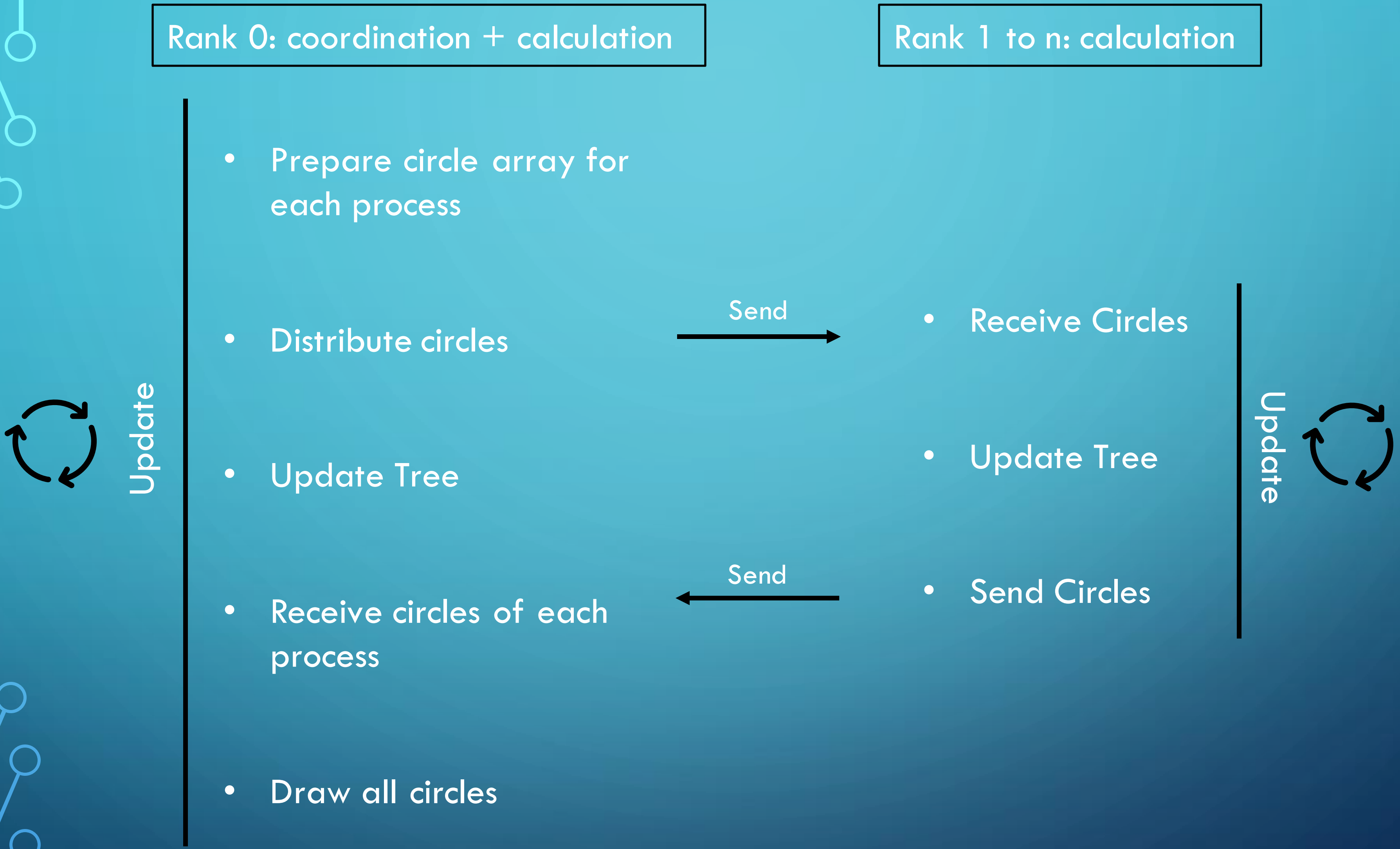


MPI WITH TREES

- Split the window in multiple subfields
- Assign one to each process
- Every process manages a tree with dimensions of its subfield
- Process 0 coordinates circle distribution

Window subfields with process numbers





Rank 0: coordination + calculation

Update

- Prepare circle array for each process
- Distribute circles
- Update Tree
- Receive circles of each process
- Draw all circles

Send

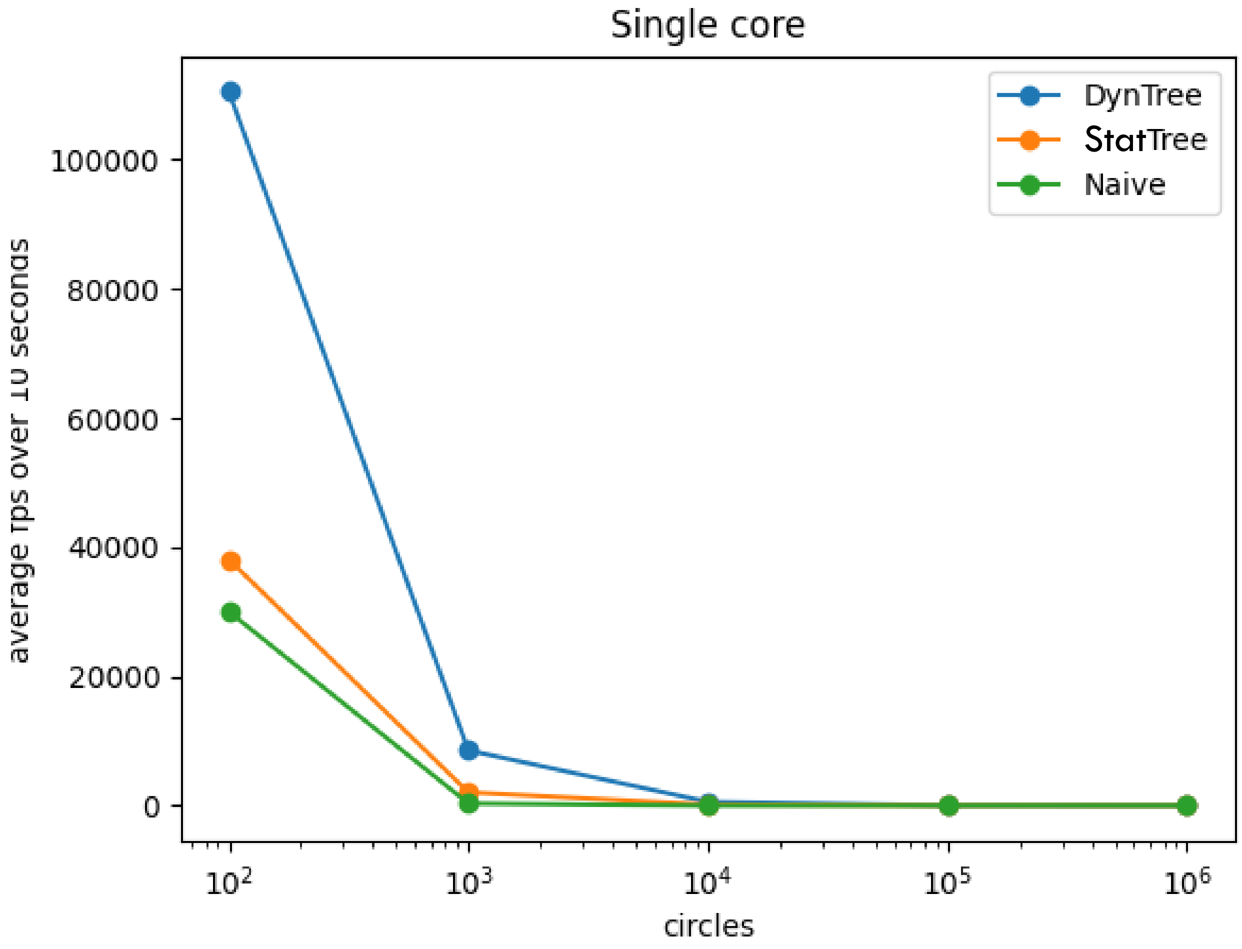
Send

Rank 1 to n: calculation

- Receive Circles
- Update Tree
- Send Circles

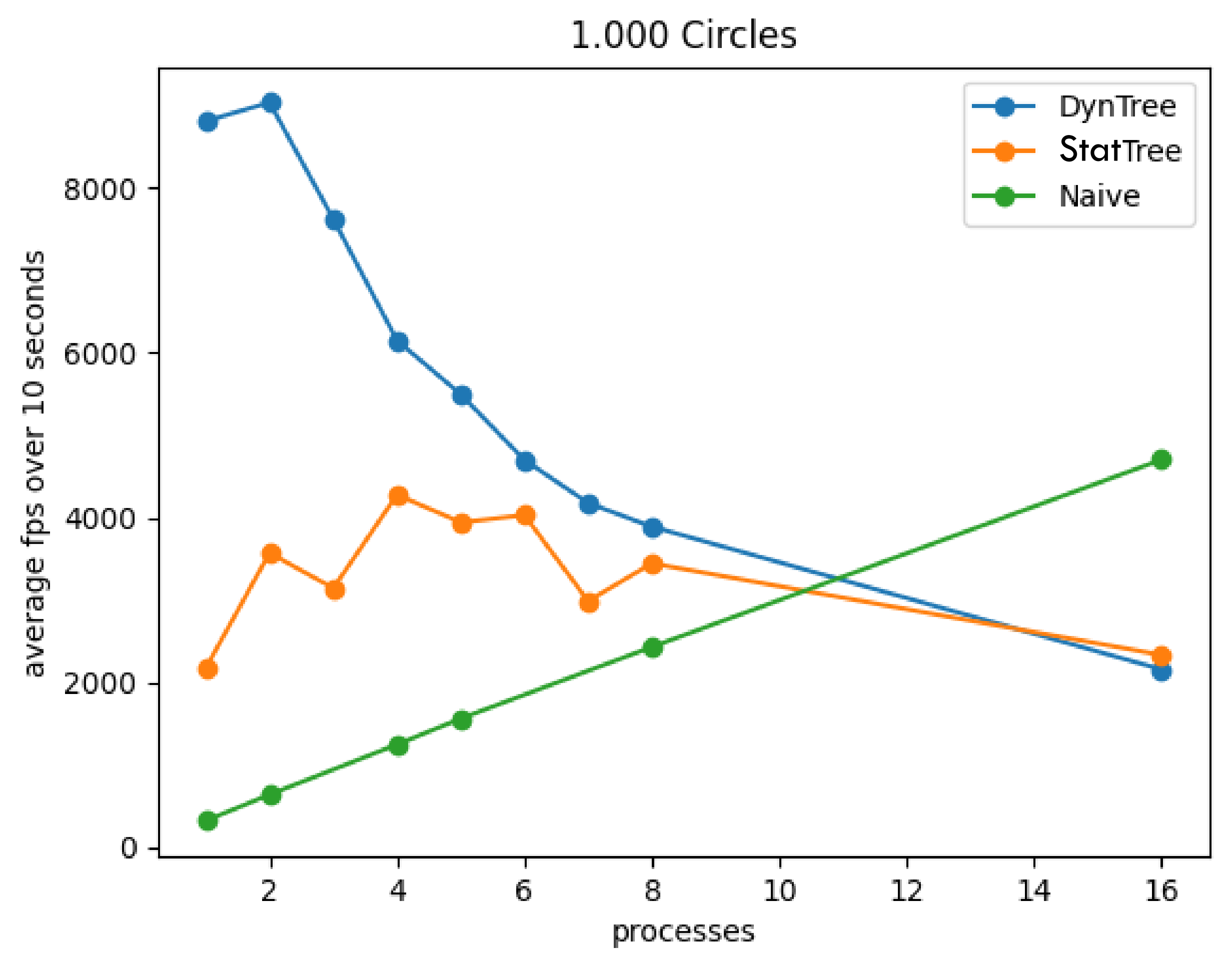
Update

PERFORMANCE COMPARISON



- Naïve works slower than both tree approaches
- With more circles the fps improvement gets smaller
- Dynamic tree shows best results

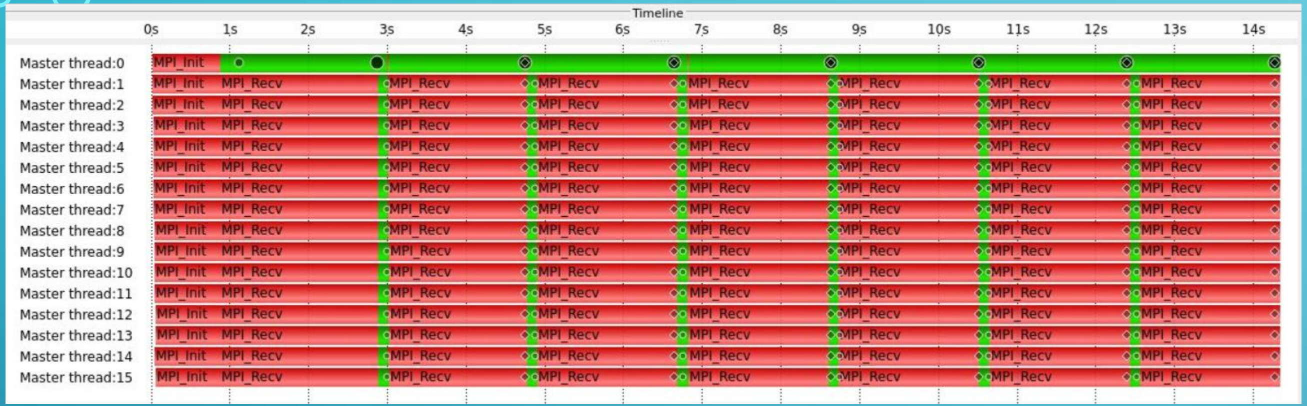
Performance comparison

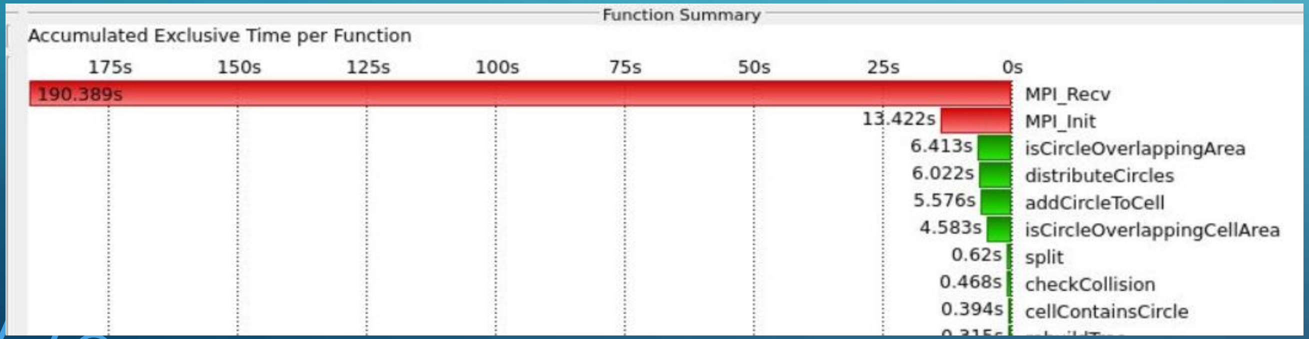


- The naïve implementation scales nearly optimal
- Both trees do not scale
- The tree solutions get worse with more processes

Performance comparison

THE PROBLEM (STATIC TREE)

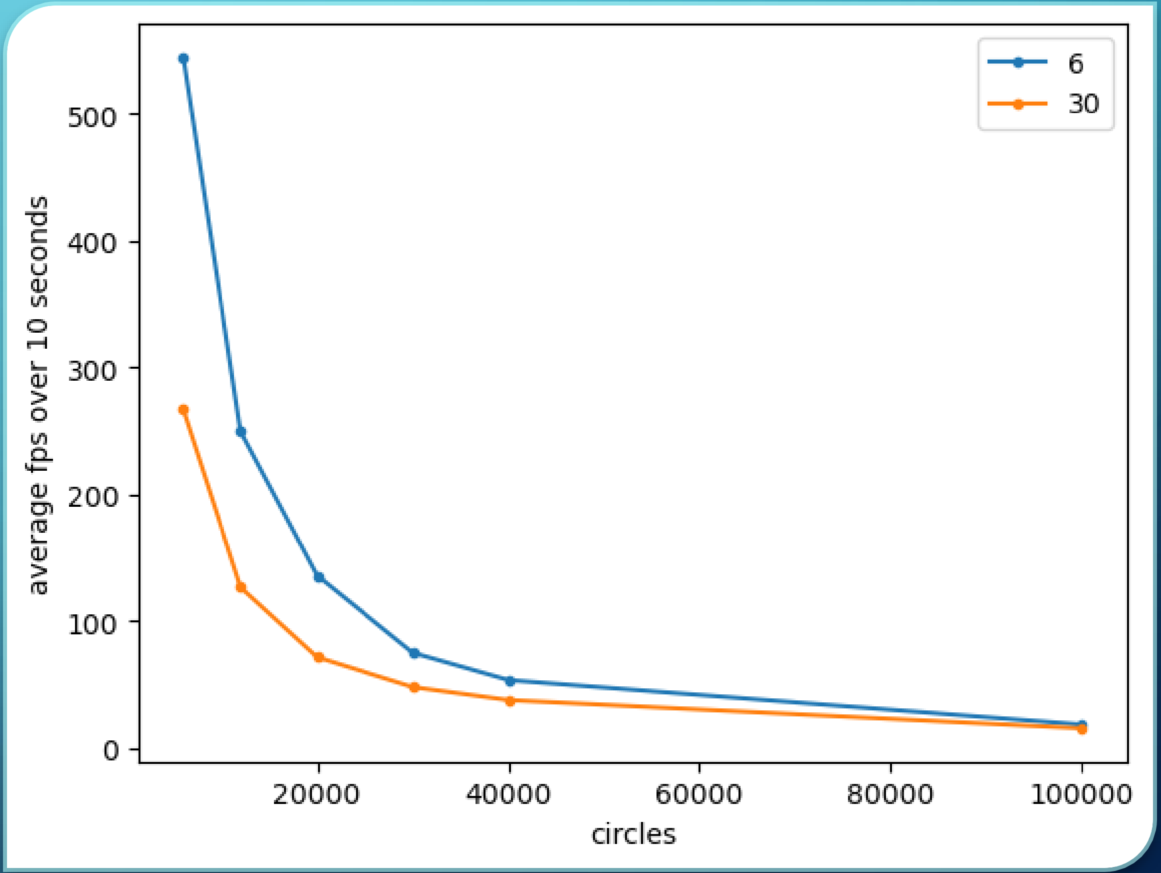




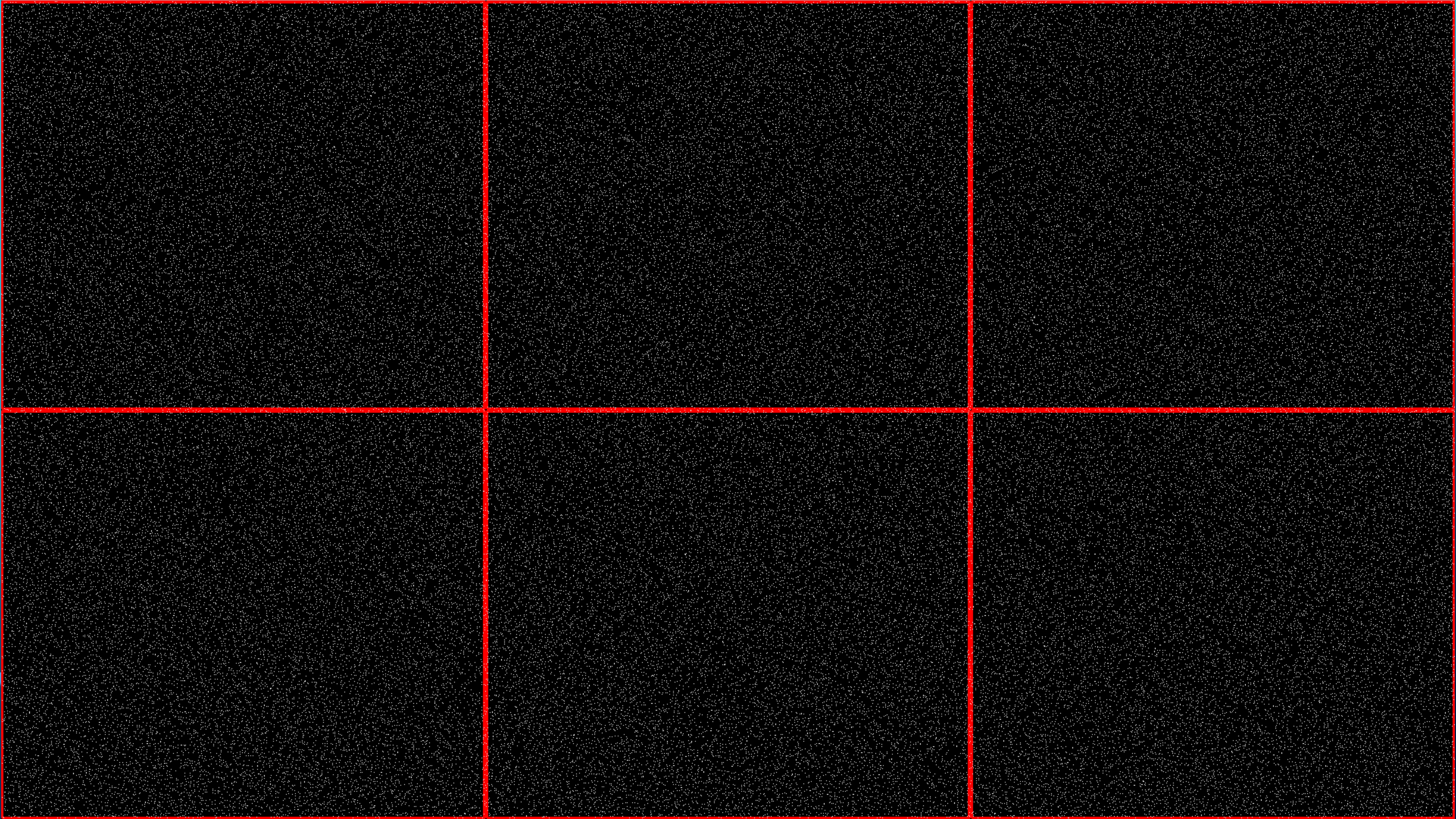
- All processes are waiting until process 0 distributes updated circles
- The overhead for process 0 is bigger than the time advantage we get with less collision checks
  - Only if number of circles is low

STATIC TREE

- static tree approach works better with less processes
- Even if we increase the amount of circles
- At around 100k circles we get a turn point
- The collision check gets more complex compared to the work process 0 gets



Performance comparison



CONCLUSION

Naïve:

- Gets slow with growing circle amounts
- Scales nearly optimal with increasing process size

- Should be used if you have a high amount of processes

Trees:

- Trees work way better in single process applications
- MPI is harder to implement and scales worse

- Should be used in single or with low amount of processes

OUTLOOK

- Different circle sizes and differente masses
- Find a better dynamic tree implmentation
- Find a better way of circles coordination (direct messaging)
- Find a better implementation where processes get a circle range instead of subfields
  - Reduces message, comparable with the naïve messaging
  - Scales better if circles are not equally distributed (e.g. gravity)