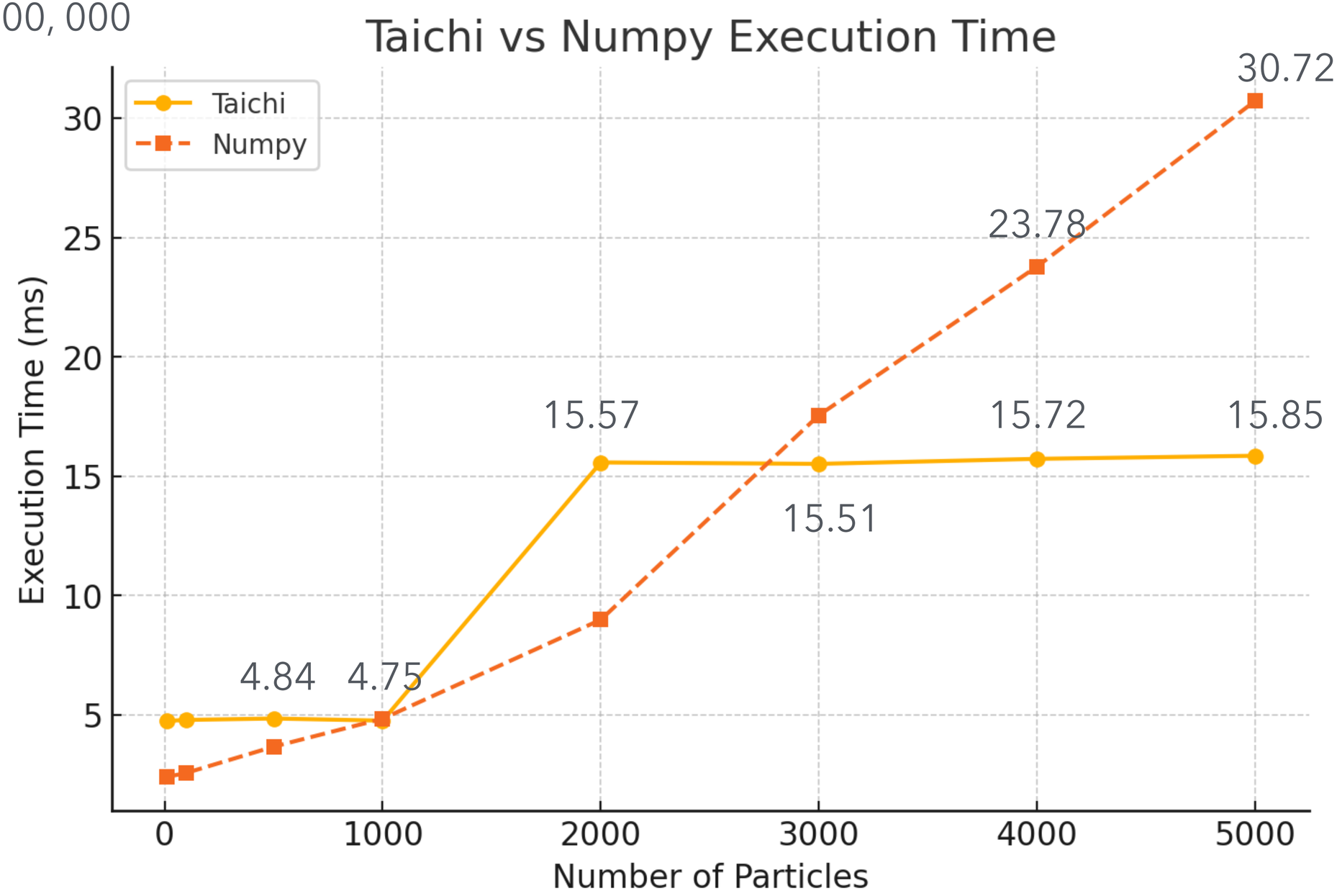
Московский Государственный Университет им. М.В. Ломоносова
факультет Вычислительной математики и кибернетики
кафедра Суперкомпьютеров и Квантовой информатики


# Возможности пакета TaiChi для математического моделирования на современных вычислительных архитектурах

Выполнила:
студентка группы 538, Ши Хуэй

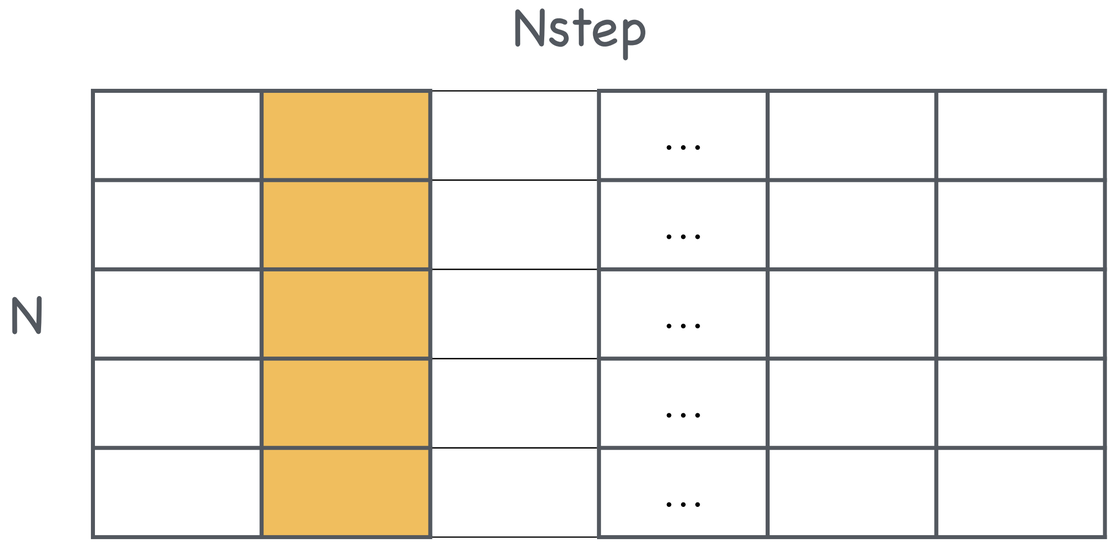
Руководитель:
доцент кафедры СКИ, к.т.н., Русол А.В.

numSteps: 300, 000



```
x = np.zeros((n,nstep+1),float)
```

```
for i in range(nstep):
    dx = np.diff(x[:,i])-l
    F = -C*np.append(0.0, dx) + C*np.append(dx ,0.0) - m*g
    F[0] = F[0] - Cs*x[0,i]*(x[0,i]<0)
    a = F/m
    vx[:,i+1] = vx[:,i] + a*dt
    x[:,i+1] = x[:,i] + vx[:,i+1]*dt
```




$-2^{31} <= \text{Index} < 2^{31} - 1$

```
x = ti.field(dtype=float, shape=(N, nstep + 1))
```

```
@ti.kernel
def substep(i:int):
    for j in range(N - 1):
        dx[j] = (x[j + 1, i] - x[j, i]) - l

    for j in range(N):
        left = -C * dx[j - 1] if j > 0 else 0.0
        right = C * dx[j] if j < N - 1 else 0.0

        contact = -Cs * x[j, i] if j == 0 and x[j, i] < 0.0 else 0.0

        F[j] = left + right - B * vx[j, i] + contact - m * g

    for j in range(N):
        a = F[j] / m
        vx[j, i + 1] = vx[j, i] + a * dt
        x[j, i + 1] = x[j, i] + vx[j, i + 1] * dt

for i in range(nstep):
    substep(i)
```

# Numpy vs Taichi

```
for j in range(n - 1):
    dx[j] = x[j + 1] - x[j] - l

for j in range(n):

    left = np.array([0.0, 0.0, 0.0])
    right = np.array([0.0, 0.0, 0.0])
    contact = np.array([0.0, 0.0, 0.0])

    if j > 0:
        left = -C * dx[j - 1]
    if j < n - 1:
        right = C * dx[j]
    if j == 0 and x[j][1] < 0.0:
        contact = -Cs * x[j]

    F[j] = left + right - B * v[j] + contact - m * np.array([0, 9.8, 0])

for j in range(n):
    a = F[j] / m
    v[j] = v[j] + a * dt
    x[j] = x[j] + v[j] * dt
```

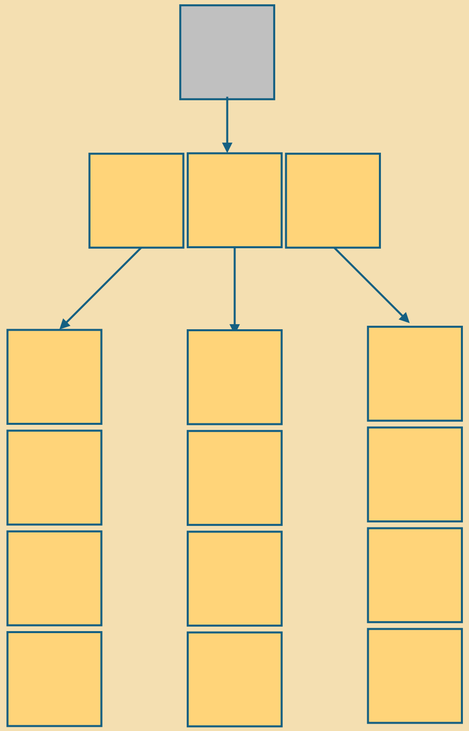
```
@ti.kernel
def substep():
    for j in range(n - 1):
        dx[j] = x[j + 1] - x[j] - ti.Vector([0, l, 0])

    for j in range(n):

        left = ti.Vector([0.0, 0.0, 0.0])
        right = ti.Vector([0.0, 0.0, 0.0])
        contact = ti.Vector([0.0, 0.0, 0.0])

        if j > 0:
            left = -C * dx[j - 1]
        if j < n - 1:
            right = C * dx[j]
        if j == 0 and x[j][1] < 0.0:
            contact = -Cs * x[j]

        F[j] = left + right - B * v[j] + contact - m * ti.Vector([0, 9.8, 0])

    for j in range(n):
        a = F[j] / m
        v[j] = v[j] + a * dt
        x[j] = x[j] + v[j] * dt
```

<table>
<tr><td>1</td><td>cpu_max_num_threads = 1,<br>Block_size = 1</td><td rowspan="4">ti.root.dense(ti.i,grid_size).dense(ti.i,block_size)</td></tr>
<tr><td>2</td><td>cpu_max_num_threads = 8,<br>Block_size = 16</td></tr>
<tr><td>3</td><td>cpu_max_num_threads = 8,<br>Block_size = 32</td></tr>
<tr><td>4</td><td>cpu_max_num_threads = 16,<br>Block_size = 64</td></tr>
<tr><td>5</td><td rowspan="3">cpu_max_num_threads = 8,<br>Block_size = 16</td><td>ti.root.dense(ti.i,grid_size).dense(ti.i,block_size)</td></tr>
<tr><td>6</td><td>ti.root.pointer(ti.i,grid_size).dense(ti.i,block_size)</td></tr>
<tr><td>7</td><td>ti.Vector.field(n)</td></tr>
<tr><td>8</td><td>Numpy</td><td>np.array()</td></tr>
</table>

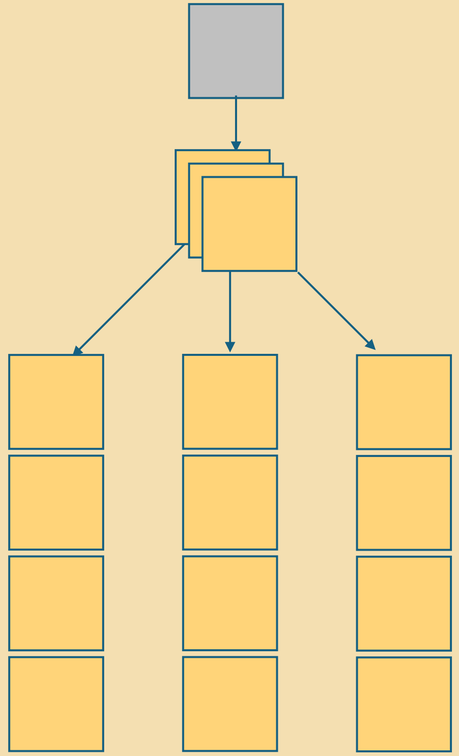
ti.Vector

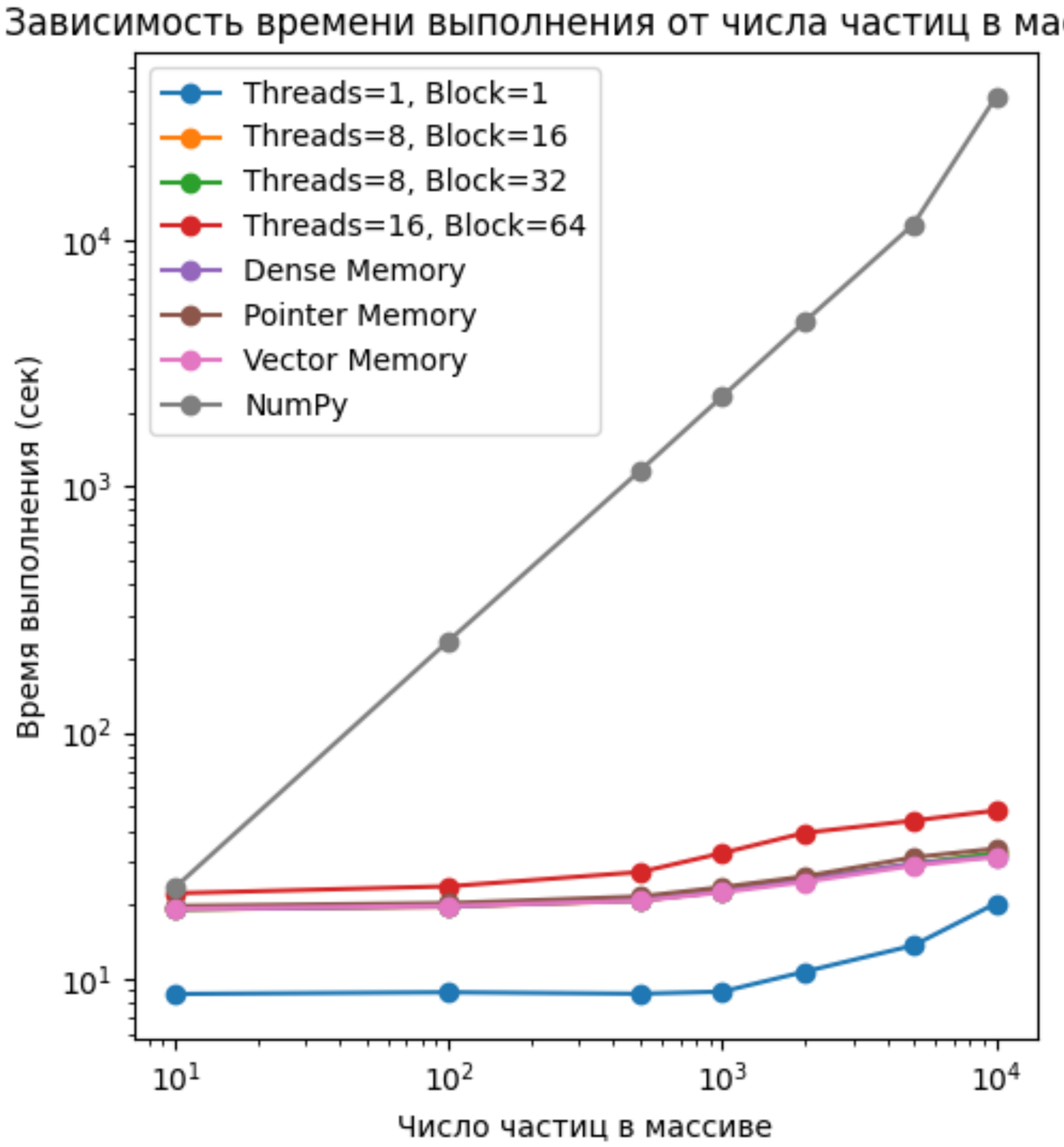
ti.root.pointer(ti.i, grid_size).dense(ti.i, block_size).place(x)

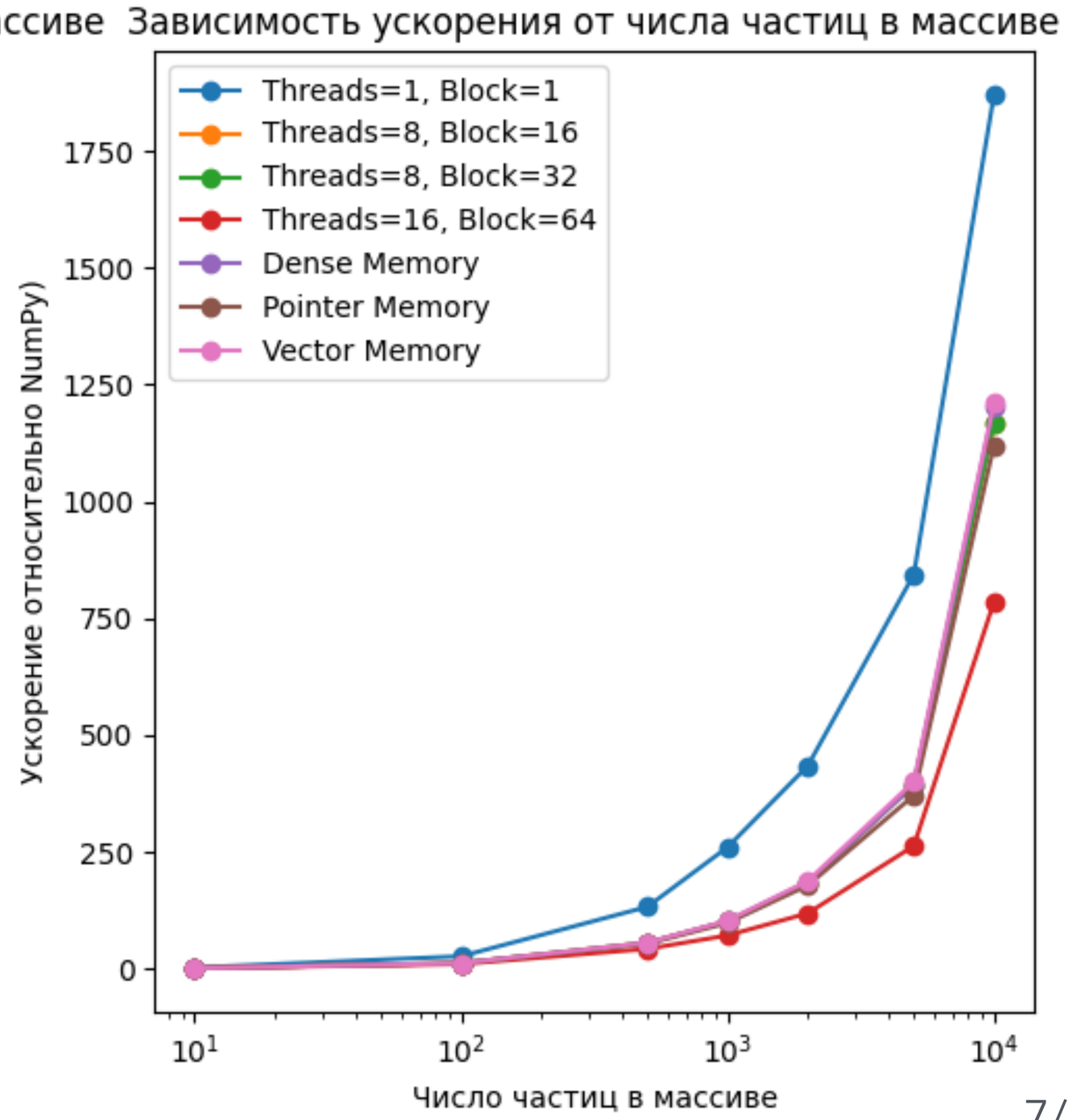
ti.dense

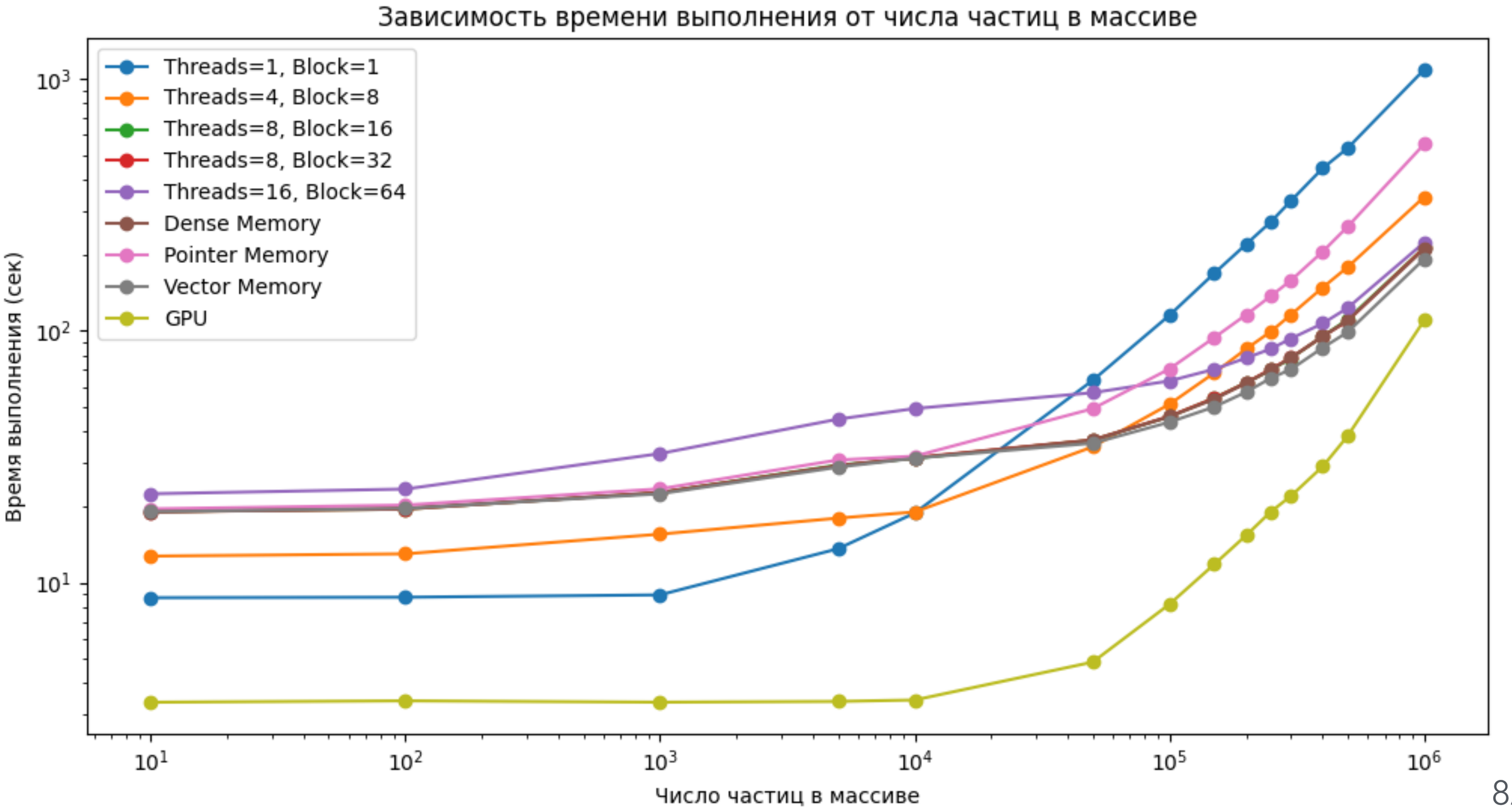


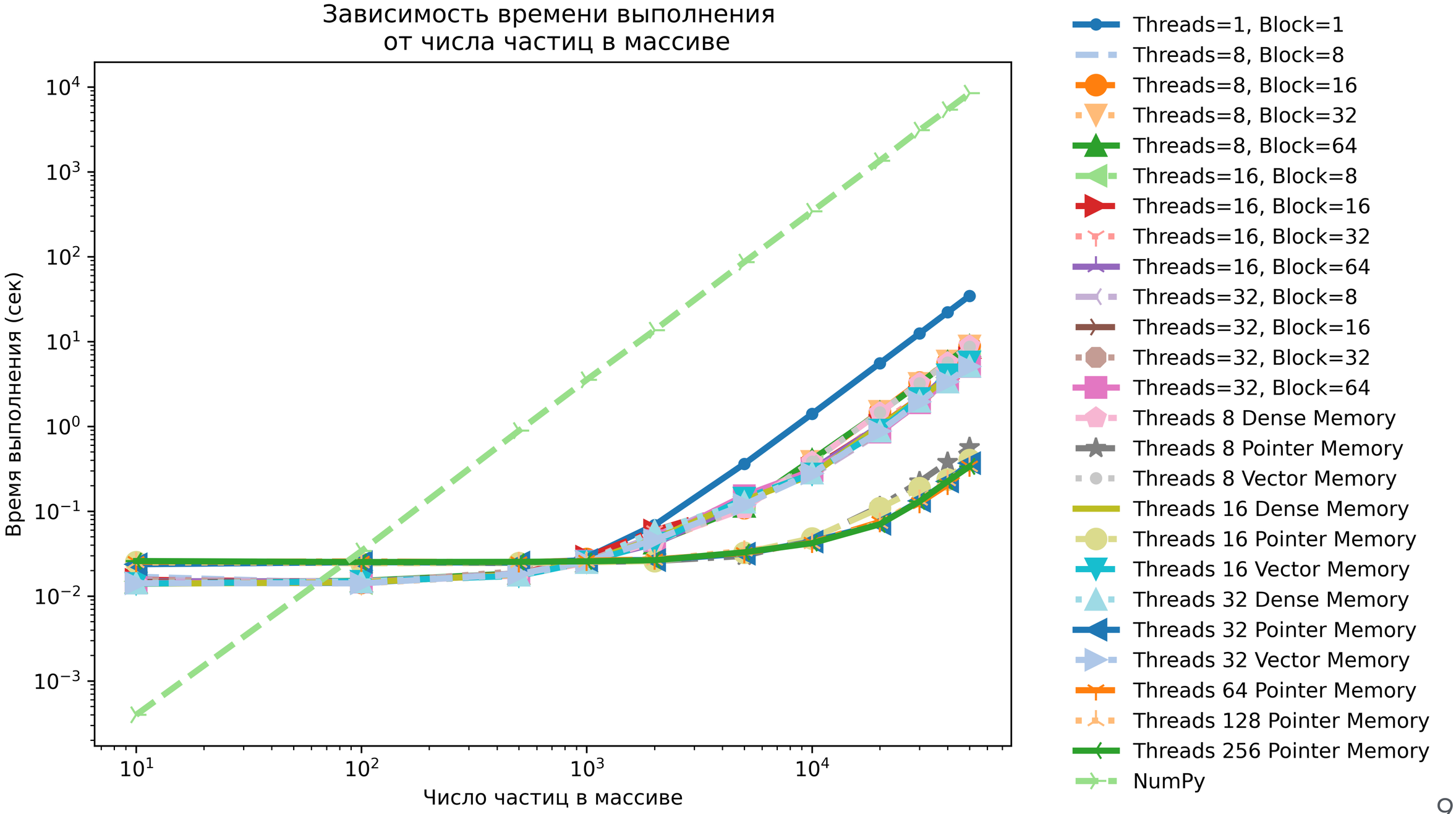
ti.pointer

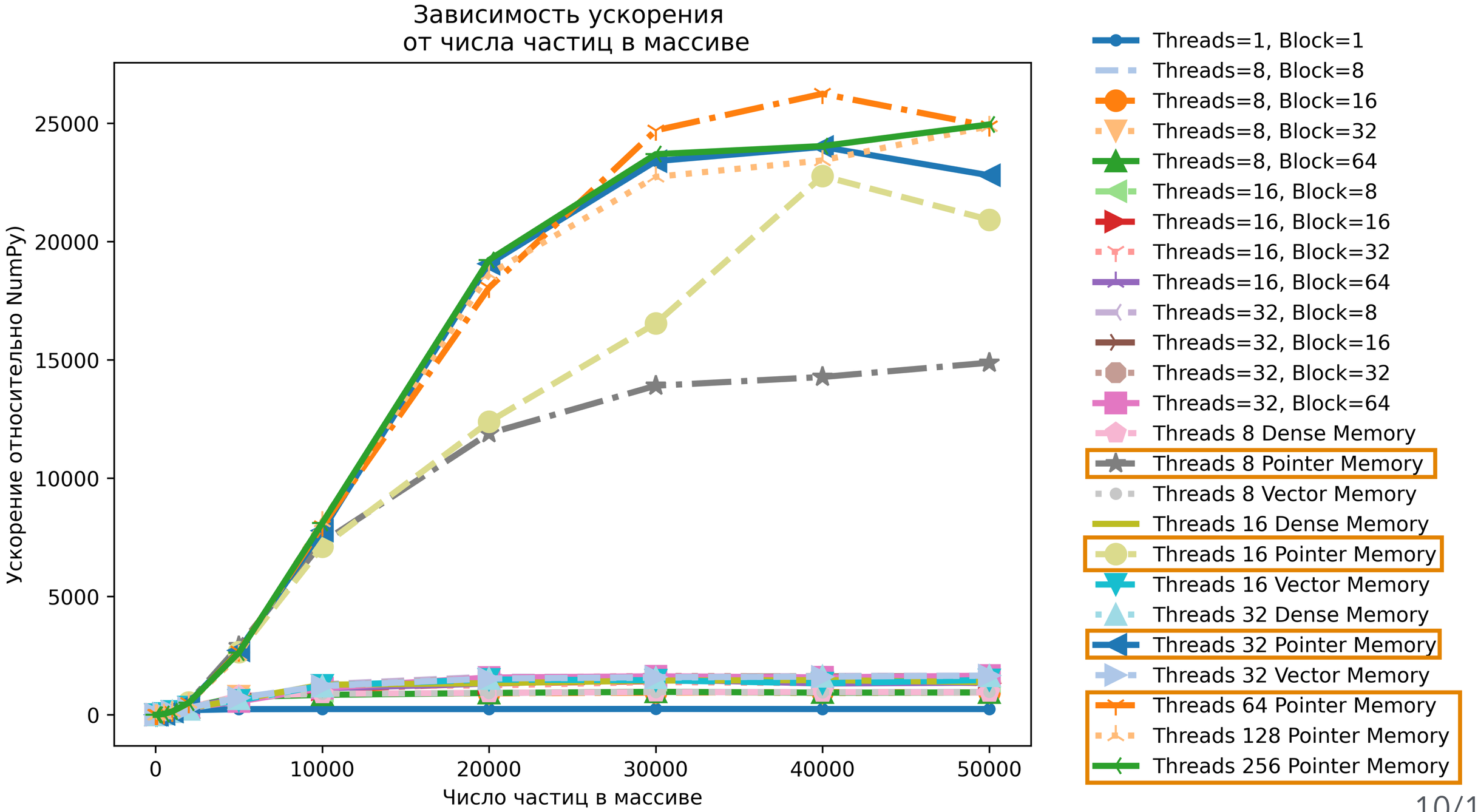



Зависимость времени выполнения от числа частиц в массиве
Threads=1, Block=1
Threads=8, Block=16
Threads=8, Block=32
Threads=16, Block=64
Dense Memory
Pointer Memory
Vector Memory
NumPy
Время выполнения (сек)
$10^4$
$10^3$
$10^2$
$10^1$
$10^1$
$10^2$
$10^3$
$10^4$
Число частиц в массиве


Зависимость ускорения от числа частиц в массиве
Threads=1, Block=1
Threads=8, Block=16
Threads=8, Block=32
Threads=16, Block=64
Dense Memory
Pointer Memory
Vector Memory
Ускорение относительно NumPy)
1750
1500
1250
1000
750
500
250
0
$10^1$
$10^2$
$10^3$
$10^4$
Число частиц в массиве


Зависимость времени выполнения от числа частиц в массиве
Threads=1, Block=1
Threads=4, Block=8
Threads=8, Block=16
Threads=8, Block=32
Threads=16, Block=64
Dense Memory
Pointer Memory
Vector Memory
GPU
Время выполнения (сек)
Число частиц в массиве

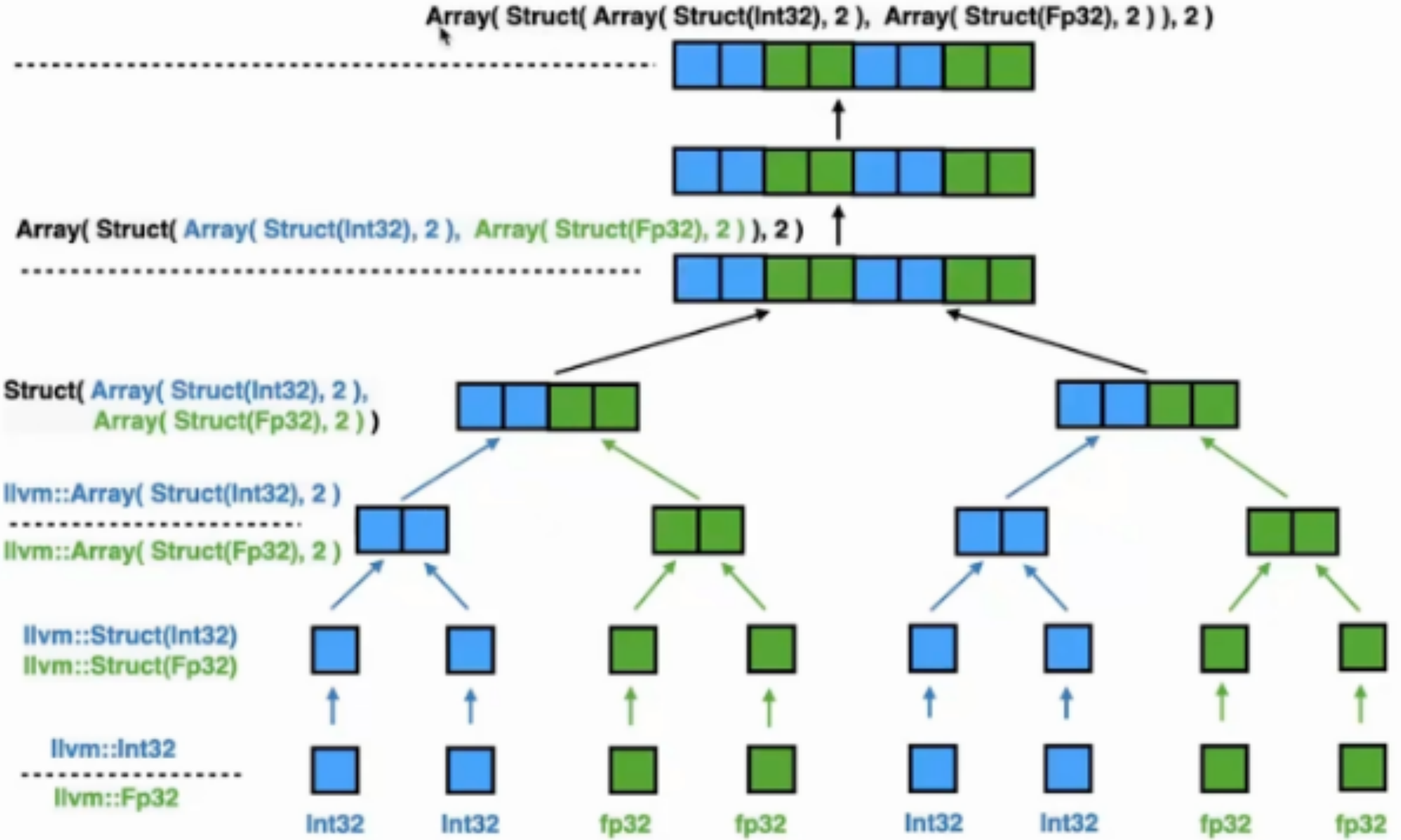

Зависимость времени выполнения
от числа частиц в массиве
Время выполнения (сек)
Число частиц в массиве
$10^{4}$
$10^{3}$
$10^{2}$
$10^{1}$
$10^{0}$
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{1}$
$10^{2}$
$10^{3}$
$10^{4}$
Threads=1, Block=1
Threads=8, Block=8
Threads=8, Block=16
Threads=8, Block=32
Threads=8, Block=64
Threads=16, Block=8
Threads=16, Block=16
Threads=16, Block=32
Threads=16, Block=64
Threads=32, Block=8
Threads=32, Block=16
Threads=32, Block=32
Threads=32, Block=64
Threads 8 Dense Memory
Threads 8 Pointer Memory
Threads 8 Vector Memory
Threads 16 Dense Memory
Threads 16 Pointer Memory
Threads 16 Vector Memory
Threads 32 Dense Memory
Threads 32 Pointer Memory
Threads 32 Vector Memory
Threads 64 Pointer Memory
Threads 128 Pointer Memory
Threads 256 Pointer Memory
NumPy


Зависимость ускорения
от числа частиц в массиве
Ускорение относительно NumPy)
Число частиц в массиве
0
5000
10000
15000
20000
25000
0
10000
20000
30000
40000
50000
Threads=1, Block=1
Threads=8, Block=8
Threads=8, Block=16
Threads=8, Block=32
Threads=8, Block=64
Threads=16, Block=8
Threads=16, Block=16
Threads=16, Block=32
Threads=16, Block=64
Threads=32, Block=8
Threads=32, Block=16
Threads=32, Block=32
Threads=32, Block=64
Threads 8 Dense Memory
Threads 8 Pointer Memory
Threads 8 Vector Memory
Threads 16 Dense Memory
Threads 16 Pointer Memory
Threads 16 Vector Memory
Threads 32 Dense Memory
Threads 32 Pointer Memory
Threads 32 Vector Memory
Threads 64 Pointer Memory
Threads 128 Pointer Memory
Threads 256 Pointer Memory

## Python Code

```
x = ti.field(ti.i32)
y = ti.field(ti.f32)
root = ti.root


common = root.dense(2)


dense_x = common.dense(2)
dense_y = common.dense(2)


dense_x.place(x)
dense_y.place(y)
```

## SNode Container - Cell

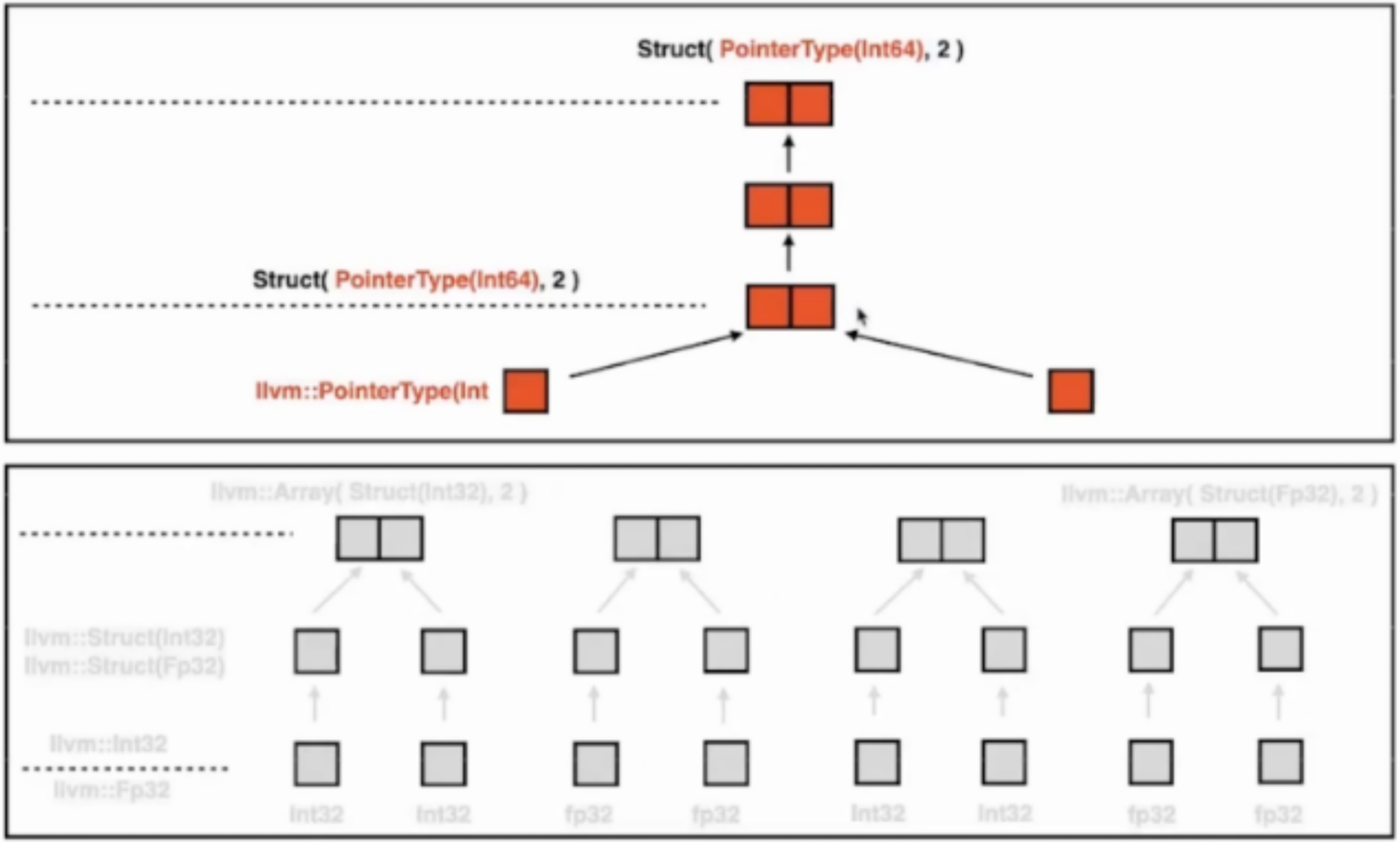


# Sparse Field

## Sparse Field

```
x = ti.field(ti.i32)
y = ti.field(ti.f32)
root = ti.root

common = root.pointer(2)

dense_x = common.dense(2)
dense_y = common.dense(2)

dense_x.place(x)
dense_y.place(y)
```

## SNode Container - Cell



Спасибо за внимание