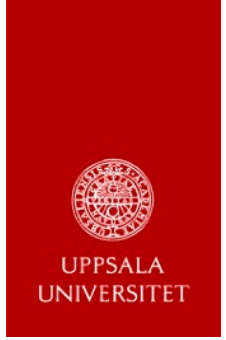
UPPSALA
UNIVERSITET

Low Level Parallel Programming

David Håkansson & Christopher Kjellqvist



Introduction

- Optimizing for SIMD operations
- Design
- Performance

UPPSALA UNIVERSITET

Problems

- Square roots are expensive
- Divides are also expensive
- ‘for’ loops introduce expensive control

```
double diffX = destination->getx() - x;
double diffY = destination->gety() - y;
double len = sqrt(diffX * diffX + diffY * diffY);
desiredPositionX = (int)round(x + diffX / len);
desiredPositionY = (int)round(y + diffY / len);
```

$$\Delta x = d_x - x$$
$$\Delta y = d_y - y$$
$$l = \sqrt{\Delta x^2 + \Delta y^2}$$
$$m_x = \Delta x \div l, m_y = \Delta y \div l$$
$$x \Leftarrow x + m_x, y \Leftarrow y + m_y$$

UPPSALA UNIVERSITET

Problems

$$\Delta x = d_x - x$$
$$\Delta y = d_y - y$$
$$l = \sqrt{\Delta x^2 + \Delta y^2}$$
$$m_x = \Delta x \div l > .5 \quad m_y = \Delta y \div l > .5$$
$$x \Leftarrow x + m_x, y \Leftarrow y + m_y$$

UPPSALA UNIVERSITET

Problems

$$\Delta x = d_x - x$$
$$\Delta y = d_y - y$$
$$l = \sqrt{\Delta x^2 + \Delta y^2}$$
$$m_x = \Delta x \div l > .5$$

$$x \Leftarrow x + m_x$$

UPPSALA UNIVERSITET

Problems

$$\Delta x = d_x - x$$
$$\Delta y = d_y - y$$
$$m_x = \frac{\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} > .5$$
$$x \Leftarrow x + m_x$$

UPPSALA UNIVERSITET

Problems

$$\Delta x = d_x - x$$
$$\Delta y = d_y - y$$
$$m_x = \Delta x > .5 \cdot \sqrt{\Delta x^2 + \Delta y^2}$$

$$x \Leftarrow x + m_x$$

UPPSALA UNIVERSITET

Problems

$$\Delta x = d_x - x$$
$$\Delta y = d_y - y$$
$$m_x = \Delta x^2 > .25 \cdot (\Delta x^2 + \Delta y^2)$$
$$x \Leftarrow x + m_x$$

UPPSALA UNIVERSITET

Problems

$$\Delta x = d_x - x$$
$$\Delta y = d_y - y$$
$$m_x = 4 \cdot \Delta x^2 > \Delta x^2 + \Delta y^2$$
$$x \Leftarrow x + m_x$$

UPPSALA UNIVERSITET

Problems

$$\Delta x = d_x - x$$
$$\Delta y = d_y - y$$
$$m_x = 3 \cdot \Delta x^2 > \Delta y^2$$
$$x \Leftarrow x + m_x$$

Originally was {-1, 0, 1}, which was removed because Δx is now squared. Create a bitmask to be 1 if Δx is > 0 or -1 if Δx is < 0. Multiply by the bitmask

UPPSALA UNIVERSITET

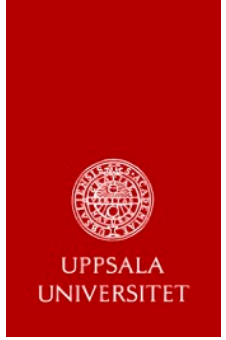
Problems

$$\Delta x = d_x - x$$
$$\Delta y = d_y - y$$
$$m_x = 3 \cdot \Delta x^2 > \Delta y^2$$
$$x \Leftarrow x + m_x$$

3 add/subs
3 multiplies, (2 of which can be used later)
1 compare
0 sqrt
0 divides
A few bitmasks

```
double diffX = destination->getx() - x;
double diffY = destination->gety() - y;
double len = sqrt(diffX * diffX + diffY * diffY);
desiredPositionX = (int)round(x + diffX / len);
```

4 add/subs
2 multiplies
1 sqrt
1 divide



SIMD Implementation

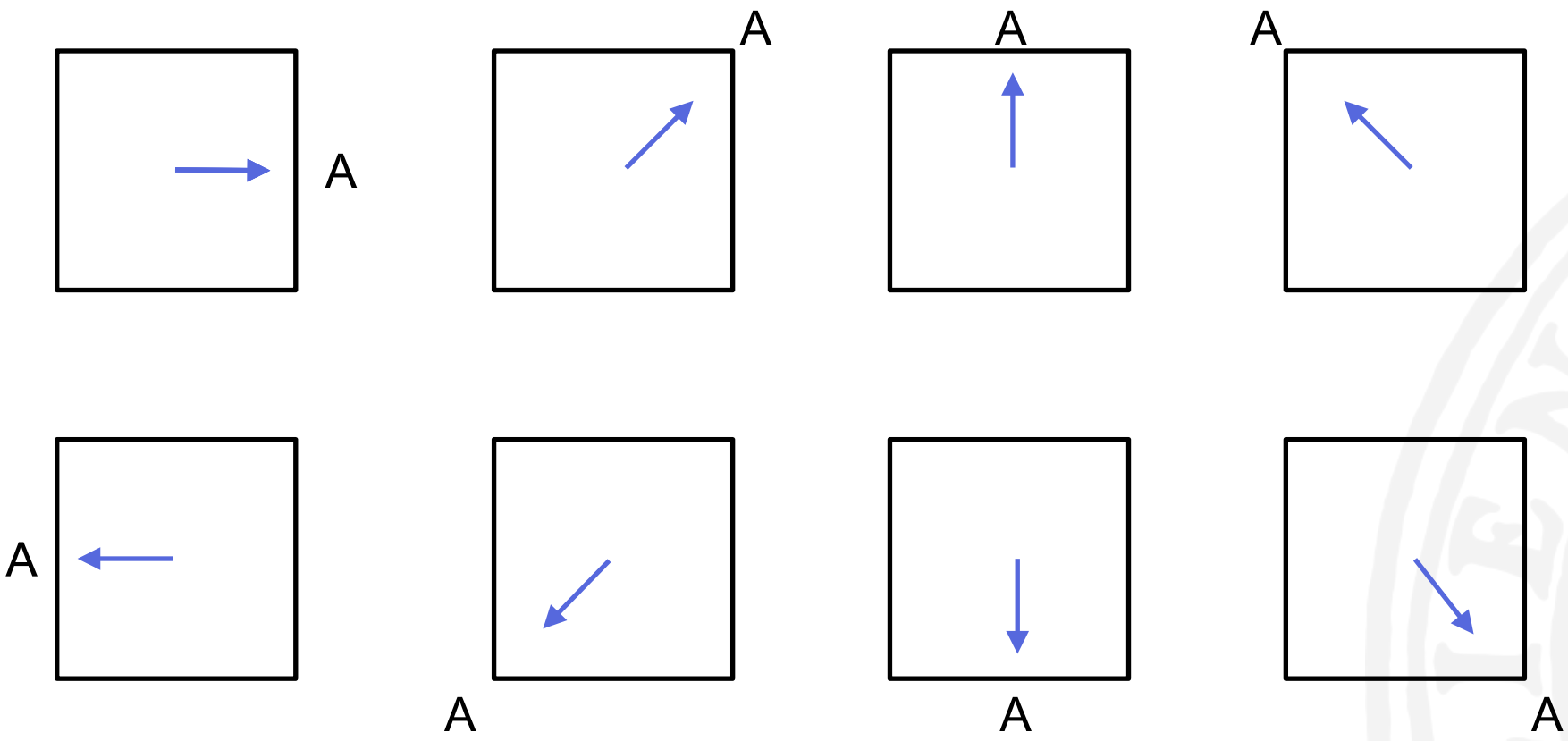
- Simplified Operations
- 16-bit ints (also support 32-bit ints)
- No lists
  - Agent has a maximum of 2 waypoints on all tests so only need 1 bit to store current waypoint
  - Use bitmasks

UPPSALA UNIVERSITET

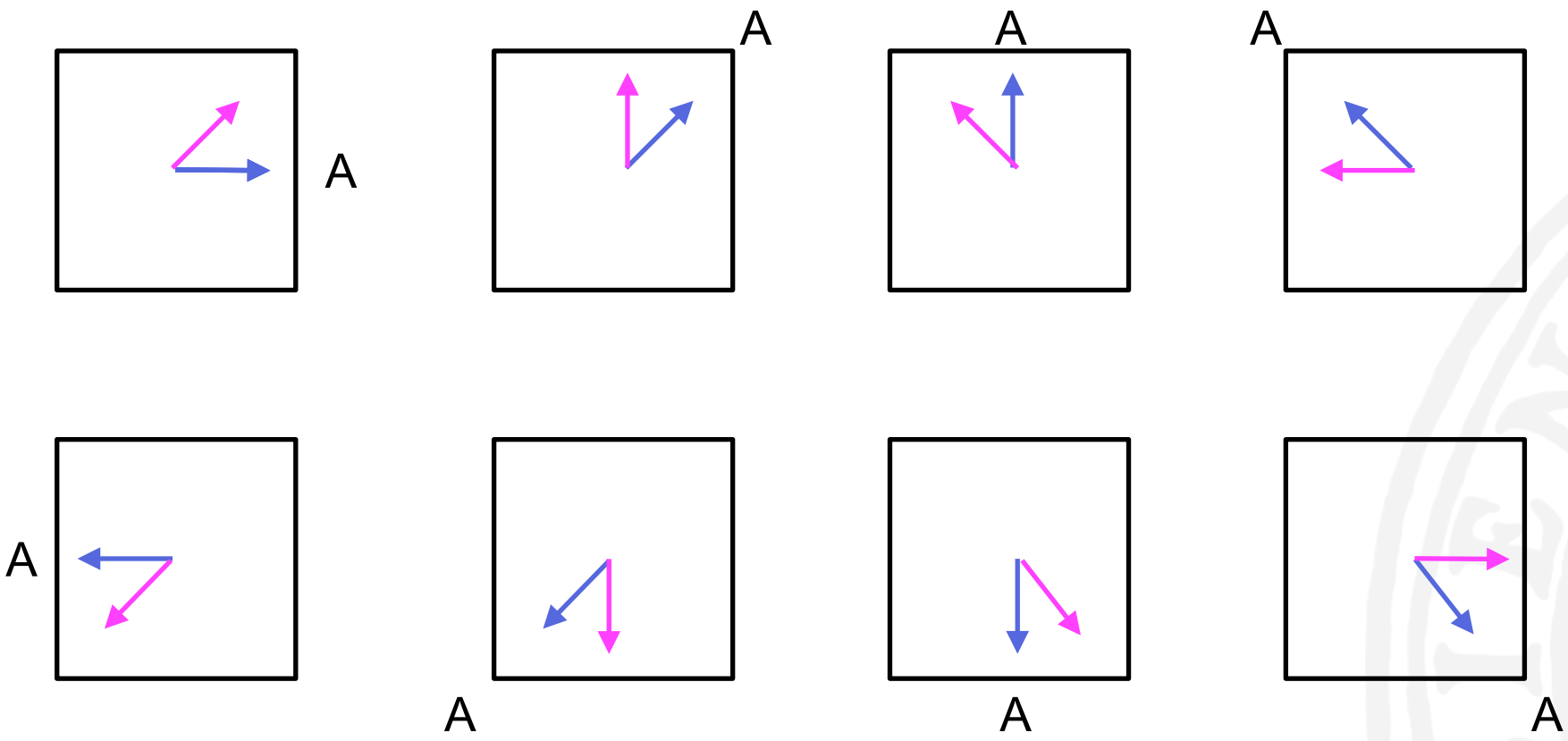
Problem - Preferred Positions

- Uses for loop and lists to generate positions
- Not entirely obvious what positions are being generated for each direction

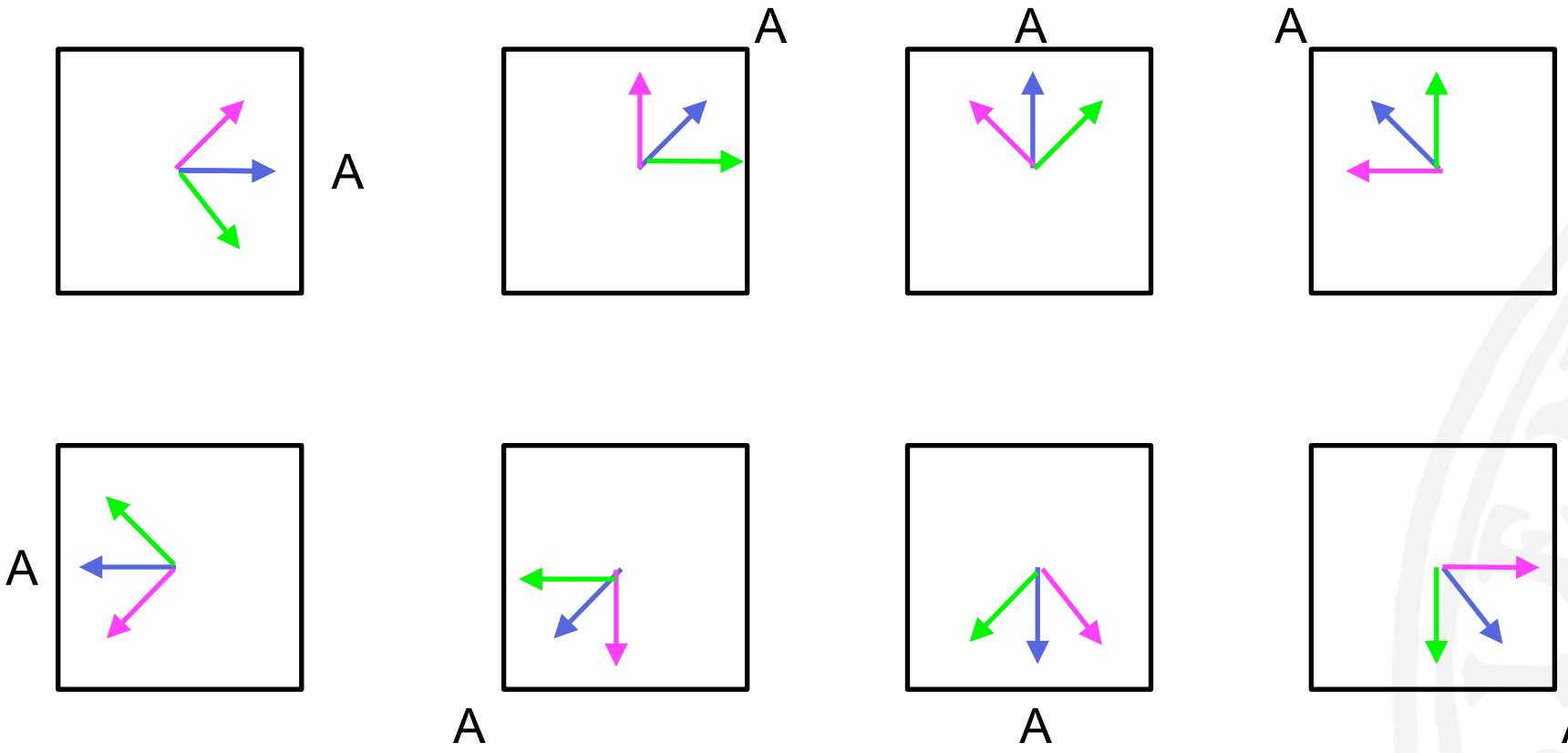
Problem - Preferred Positions



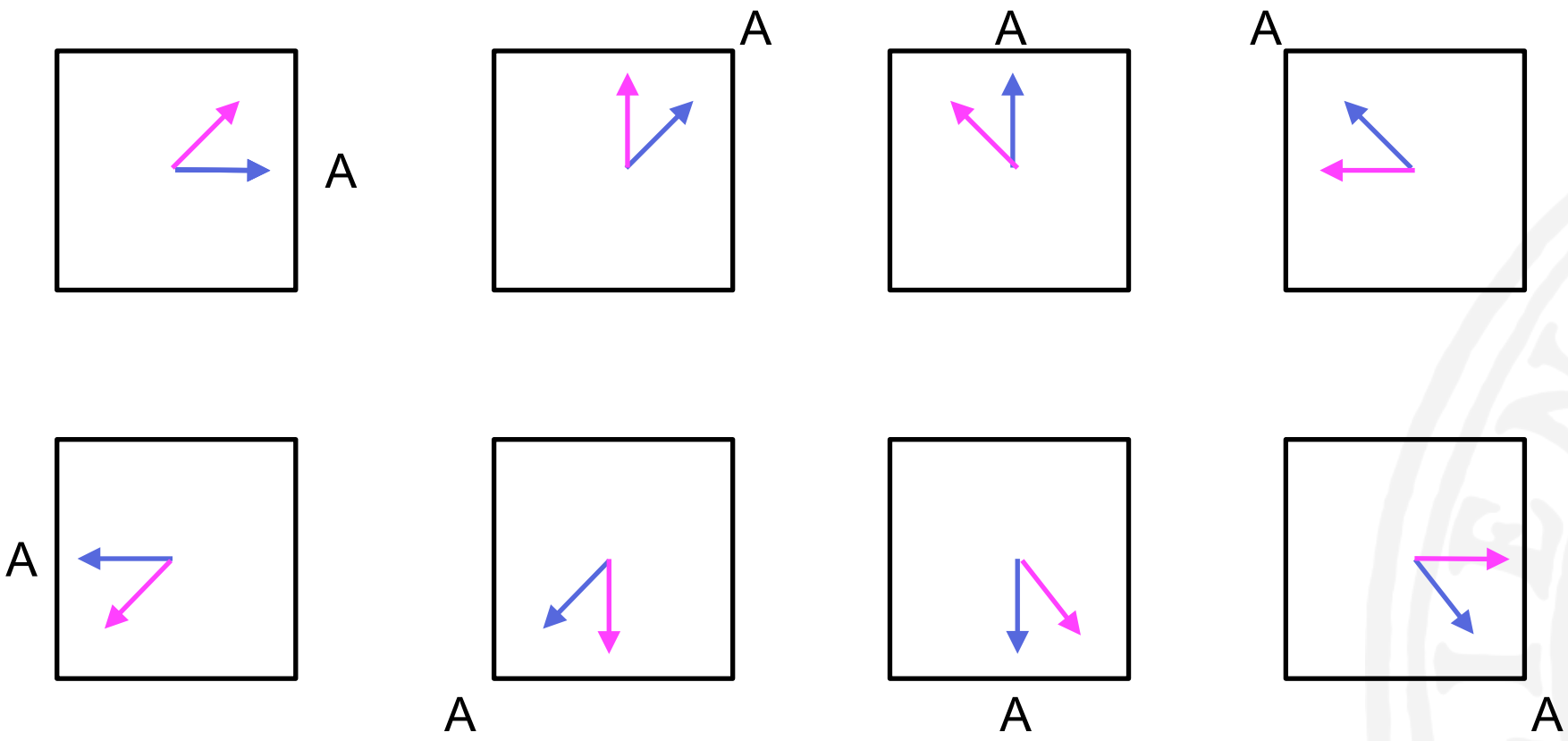
Problem - Preferred Positions



Problem - Preferred Positions



Problem - Preferred Positions



UPPSALA UNIVERSITET

Problem - Preferred Positions

dx	dy	Bx	By
-1	-1	0	-1
-1	0	-1	-1
-1	1	-1	0
0	-1	1	-1
0	0	?	?
0	1	-1	1
1	-1	1	0
1	0	1	1
1	1	0	1

Problem - Preferred Positions

dx	dy	Bx	By
-1	-1	0	-1
-1	0	-1	-1
-1	1	-1	0
0	-1	1	-1
0	0	?	?
0	1	-1	1
1	-1	1	0
1	0	1	1
1	1	0	1

$$-1_{10} = 111\ldots1_2$$
$$1_{10} = 0000\ldots01_2$$
$$0_{10} = 0000\ldots0_2$$

Since we only have 3 cases, we can condense this to 2 bit numbers

$$-1_{10} = 11_2$$
$$1_{10} = 01_2$$
$$0_{10} = 00_2$$

Problem - Preferred Positions

dx	dy	Bx	By
-1	-1	0	-1
-1	0	-1	-1
-1	1	-1	0
0	-1	1	-1
0	0	?	?
0	1	-1	1
1	-1	1	0
1	0	1	1
1	1	0	1

Hi-bits

dx	dy	Bx	By
11	11	0	1
11	0	1	1
11	1	1	0
0	11	0	1
0	0	?	?
0	1	1	0
1	11	0	0
1	0	0	0
1	1	0	0

Lo-bits

dx	dy	Bx	By
11	11	0	?
11	0	?	?
11	1	?	0
0	11	1	?
0	0	?	?
0	1	?	1
1	11	1	0
1	0	1	1
1	1	0	1

UPPSALA UNIVERSITET

Problem - Preferred Positions

Lo-bits

dx	dy	Bx	By
11	11	0	?
11	0	?	?
11	1	?	0
0	11	1	?
0	0	?	?
0	1	?	1
1	11	1	0
1	0	1	1
1	1	0	1

UPPSALA UNIVERSITET

Problem - Preferred Positions

Lo-bits

dx	dy	Bx
11	11	0
11	0	?
11	1	?
0	11	1
0	0	?
0	1	?
1	11	1
1	0	1
1	1	0

Problem - Preferred Positions

Lo-bits

dx	dy	Bx
11	11	0
11	0	?
11	1	?
0	11	1
0	0	?
0	1	?
1	11	1
1	0	1
1	1	0

dy

dx

	11	10	00	01
11	0	?	?	?
10	?	?	?	?
00	1	?	?	?
01	1	?	1	0

Problem - Preferred Positions

Lo-bits

dx	dy	Bx
11	11	0
11	0	?
11	1	?
0	11	1
0	0	?
0	1	?
1	11	1
1	0	1
1	1	0

dy

dx

	11	10	00	01
11	0	?	?	?
10	?	?	?	?
00	1	?	?	?
01	1	?	1	0

Bx_lo = (dy != 1) & (dx > -1)

Alternatively, Bx_lo = dx != dy

Problem - Preferred Positions

Lo-bits

dx	dy	Bx
11	11	0
11	0	?
11	1	?
0	11	1
0	0	?
0	1	?
1	11	1
1	0	1
1	1	0

dy

dx

	11	10	00	01
11	0	?	?	?
10	?	?	?	?
00	1	?	?	?
01	1	?	1	0

Bx_lo = (dy != 1) & (dx > -1)

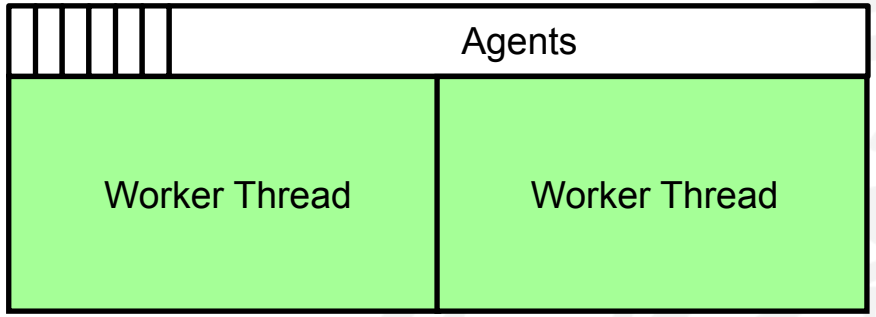
Alternatively, Bx_lo = dx != dy

Finally, calculate Bx_hi, such that it either equals 0 or -1, and OR it with Bx_lo

UPPSALA UNIVERSITET

Design - Collisions

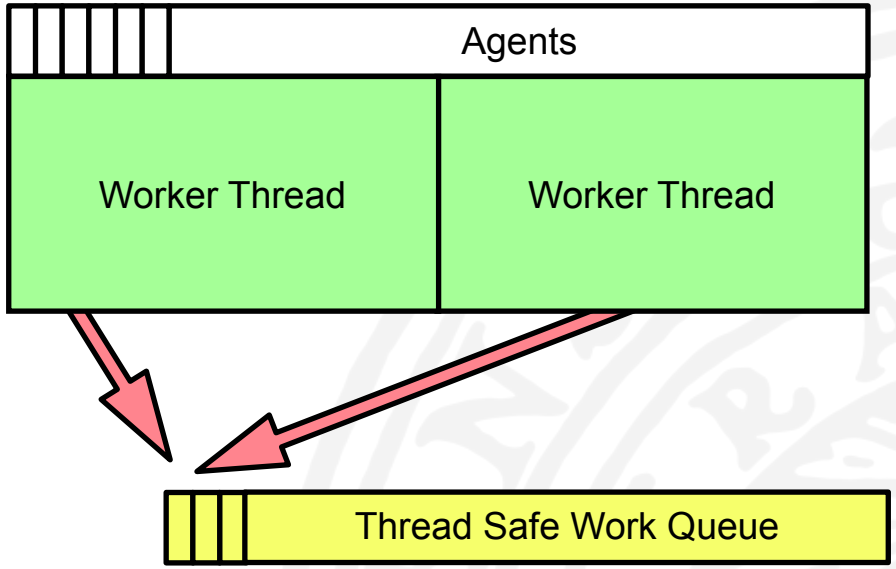
- Producer Threads (worker)
  - Operations take constant time
  - static work allocation
- Consumer Threads



UPPSALA UNIVERSITET

Design - Collisions

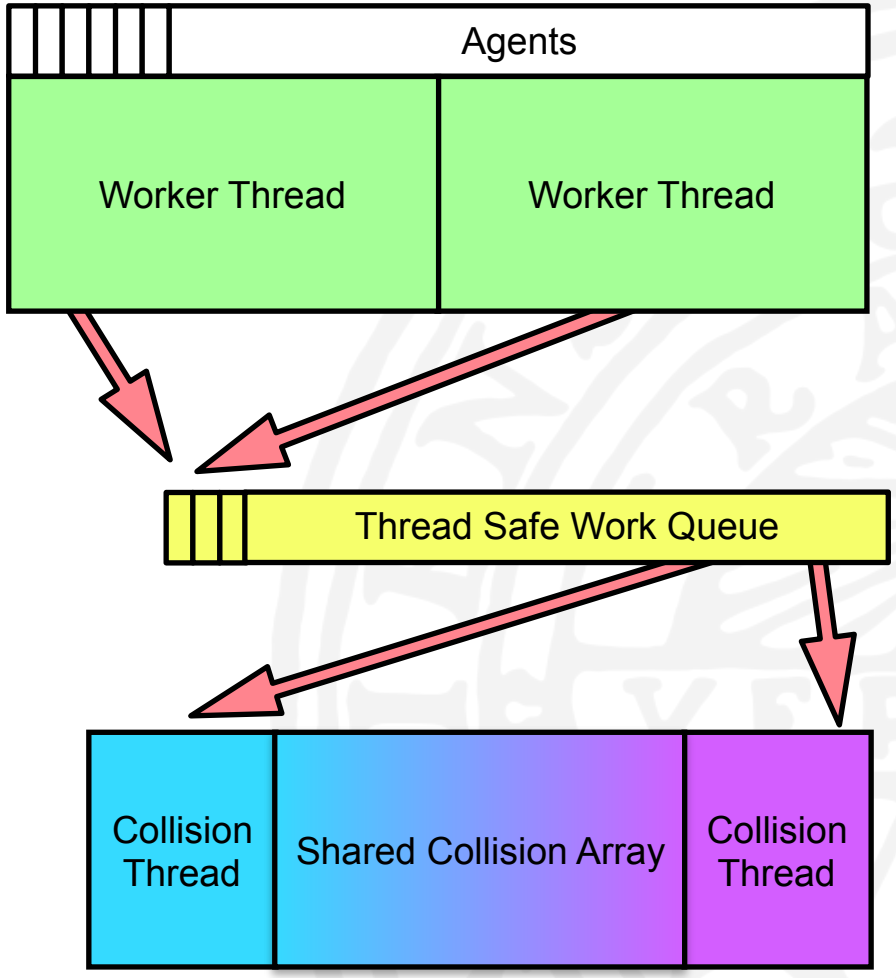
- Producer Threads (worker)
  - Operations take constant time
  - static work allocation
- Consumer Threads
  - Non constant time ops
  - Non static work allocation



UPPSALA UNIVERSITET

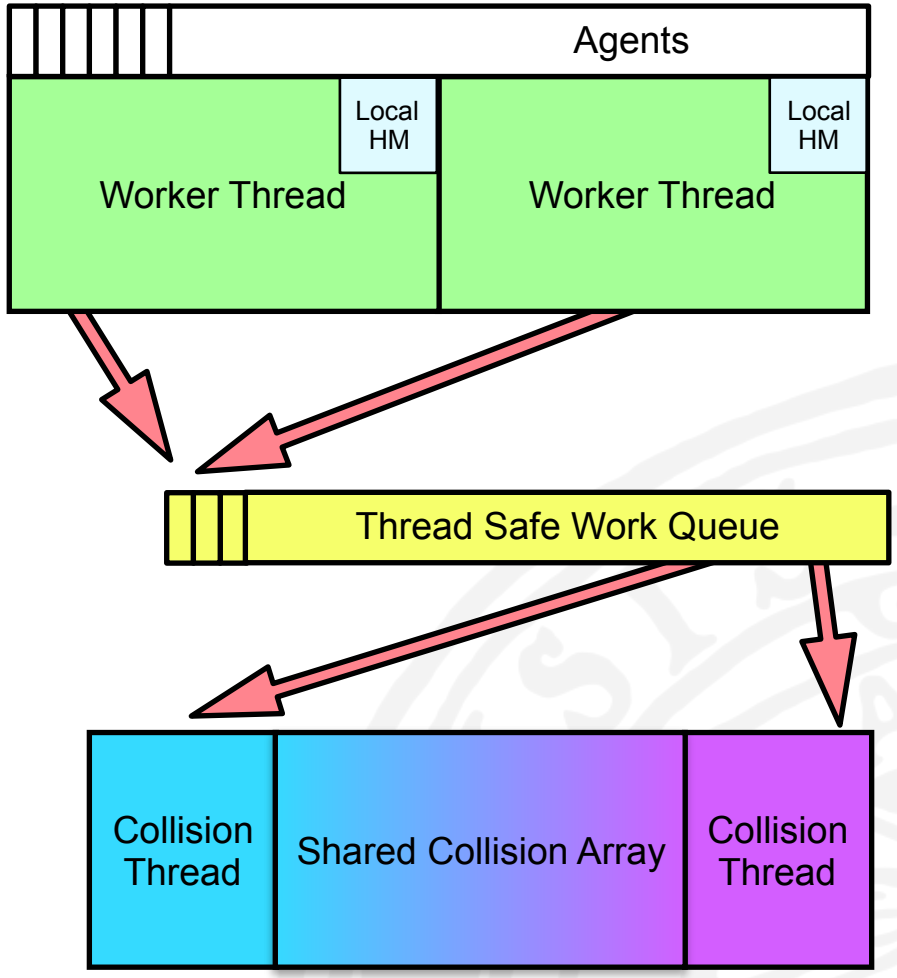
Design - Collisions

- Producer Threads (worker)
  - Operations take constant time
  - static work allocation
- Consumer Threads
  - Non constant time ops
  - Non static work allocation



UPPSALA UNIVERSITET

Design - Heatmap

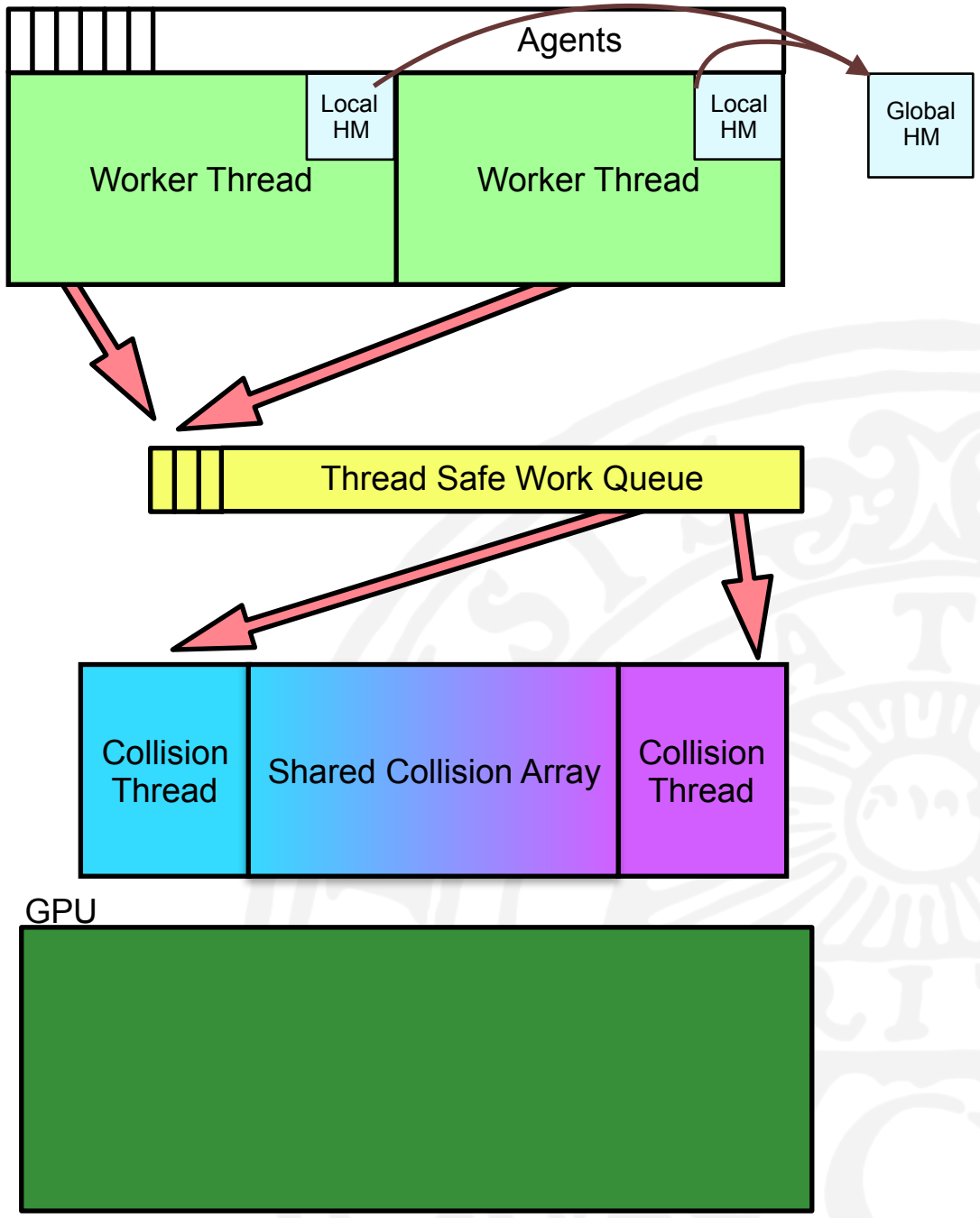


- Producer Threads (worker)
  - Operations take constant time
  - static work allocation
- Consumer Threads
  - Non constant time ops
  - Non static work allocation

UPPSALA UNIVERSITET

Design - Heatmap

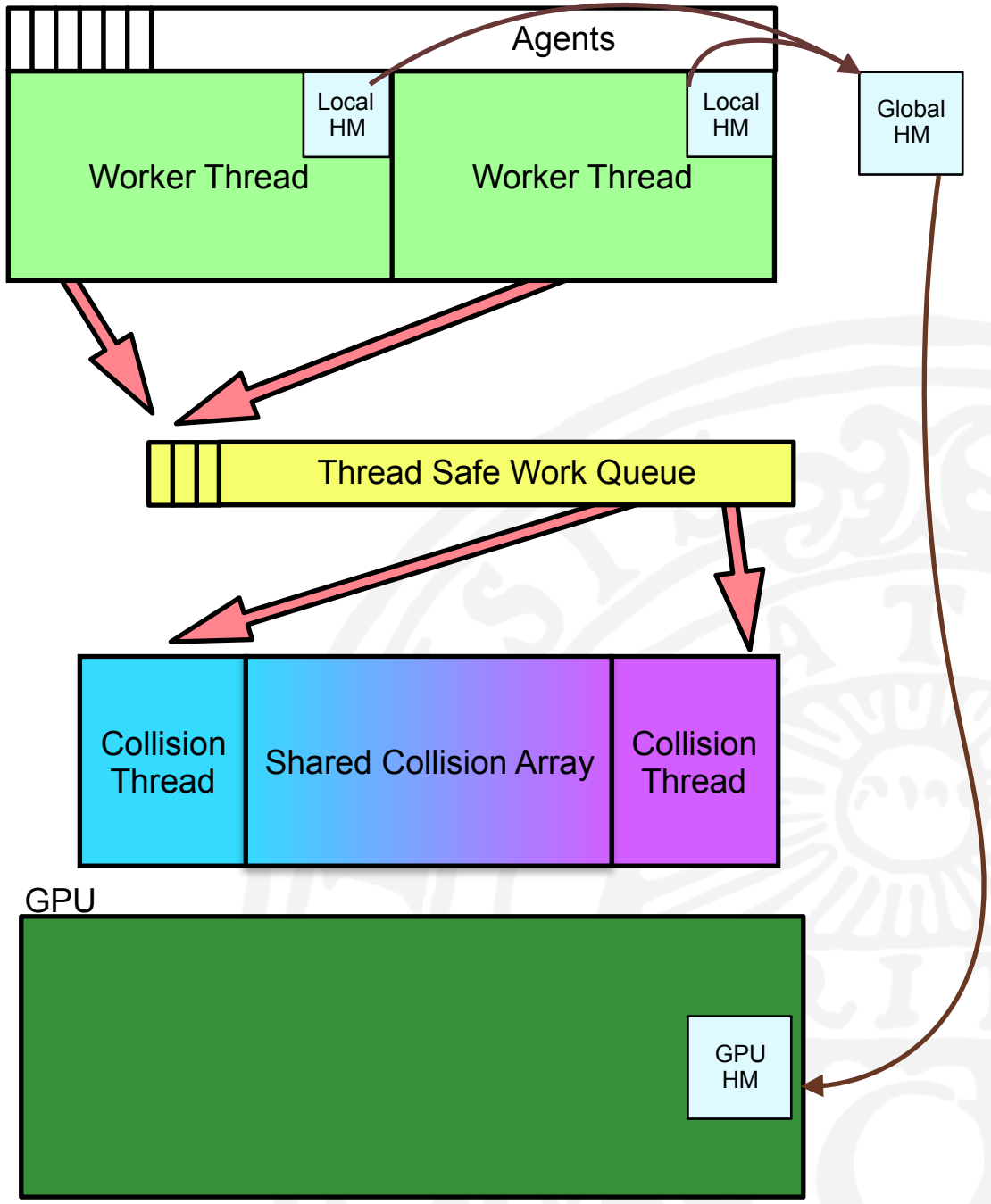
- Producer Threads (worker)
  - Operations take constant time
  - static work allocation
- Consumer Threads
  - Non constant time ops
  - Non static work allocation



UPPSALA UNIVERSITET

Design - Heatmap

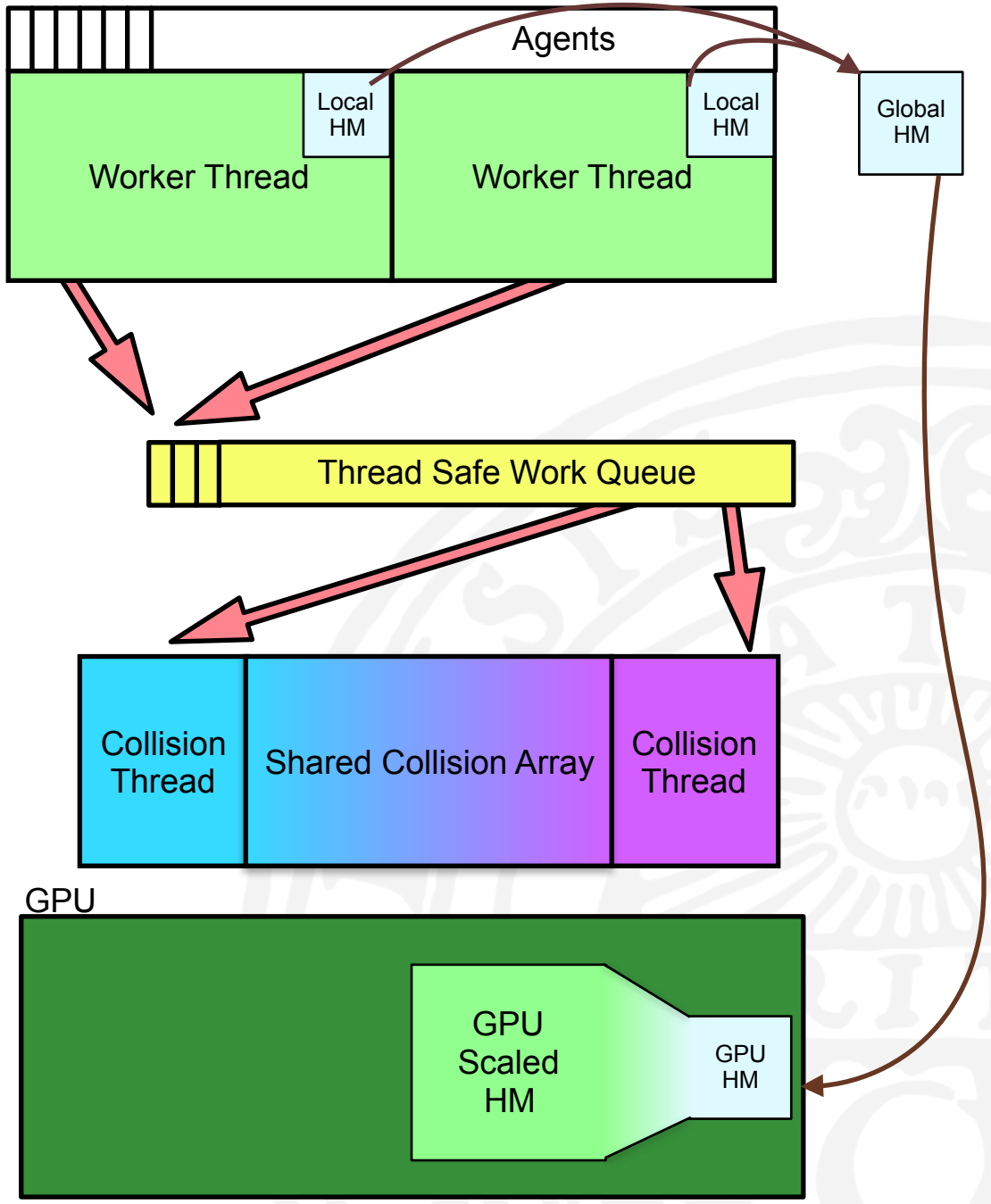
- Producer Threads (worker)
  - Operations take constant time
  - static work allocation
- Consumer Threads
  - Non constant time ops
  - Non static work allocation



UPPSALA UNIVERSITET

Design - Heatmap

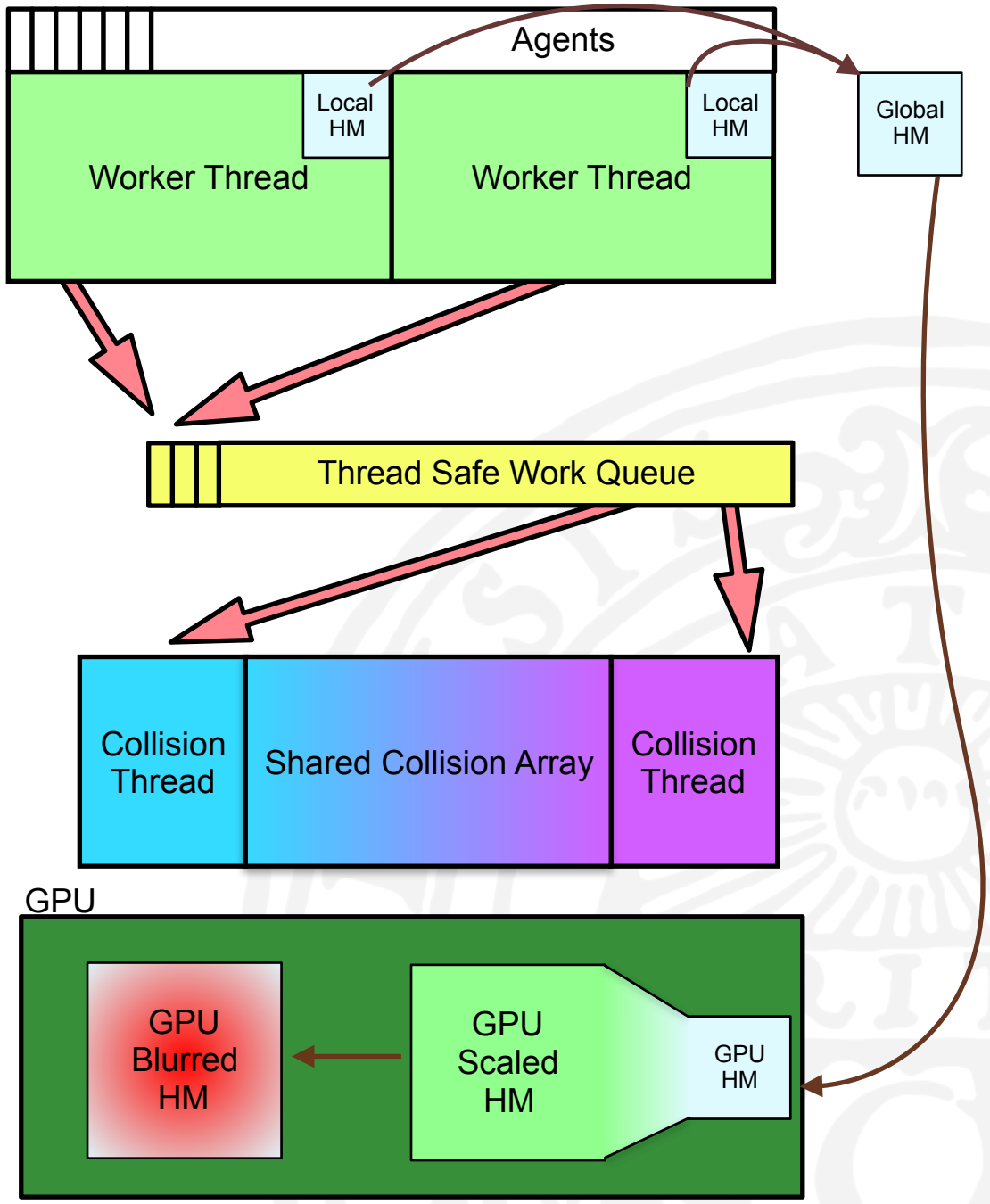
- Producer Threads (worker)
  - Operations take constant time
  - static work allocation
- Consumer Threads
  - Non constant time ops
  - Non static work allocation



UPPSALA UNIVERSITET

Design - Heatmap

- Producer Threads (worker)
  - Operations take constant time
  - static work allocation
- Consumer Threads
  - Non constant time ops
  - Non static work allocation



UPPSALA UNIVERSITET

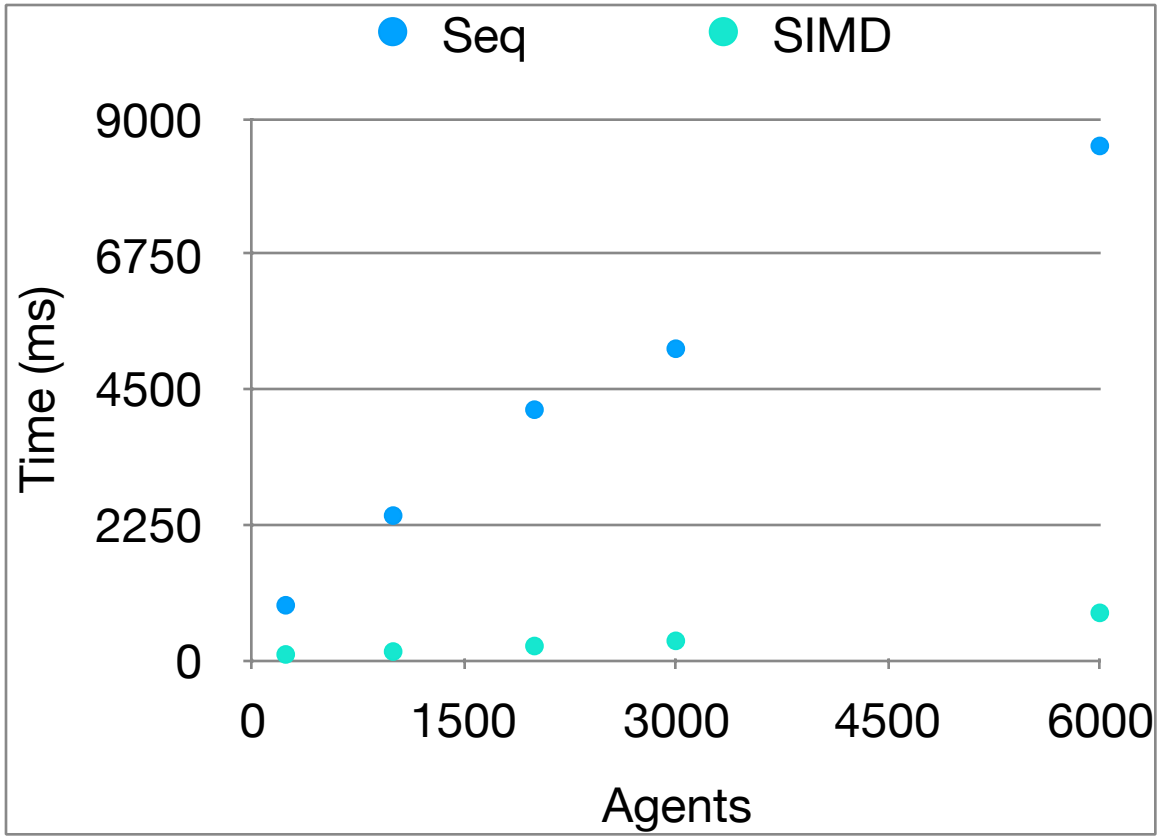
Performance

- All numbers with 50 000 iterations done on varying number of agents (provided 50 000 iterations would take a reasonable amount of time)
- Used scenario.xml setup

UPPSALA UNIVERSITET

Performance

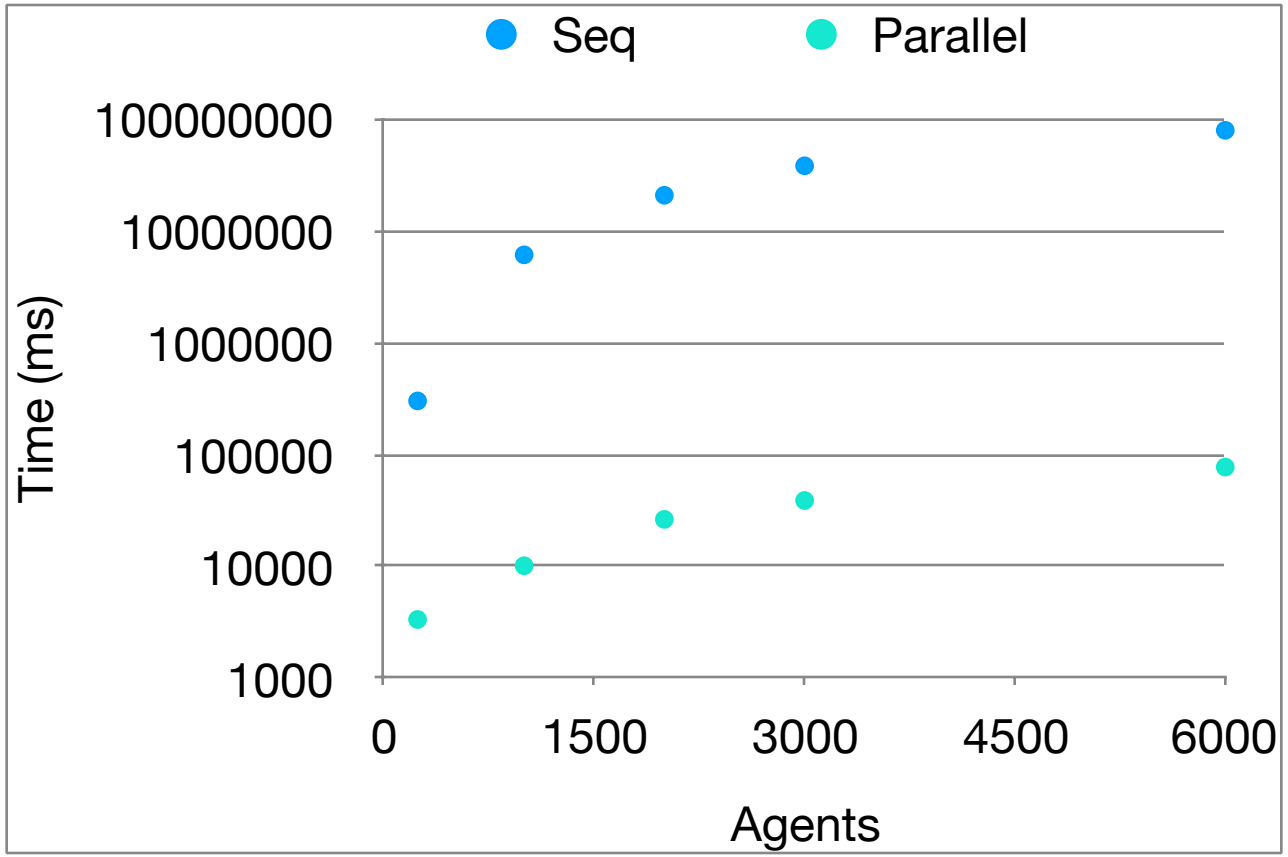
- SIMD gives us 10x speedup over original version



Agents	Seq(slow)	SIMD
240	908	91
1000	2398	139
2000	4159	231
3000	5172	317
6000	8542	782

Performance

- 1000x speedup with collisions

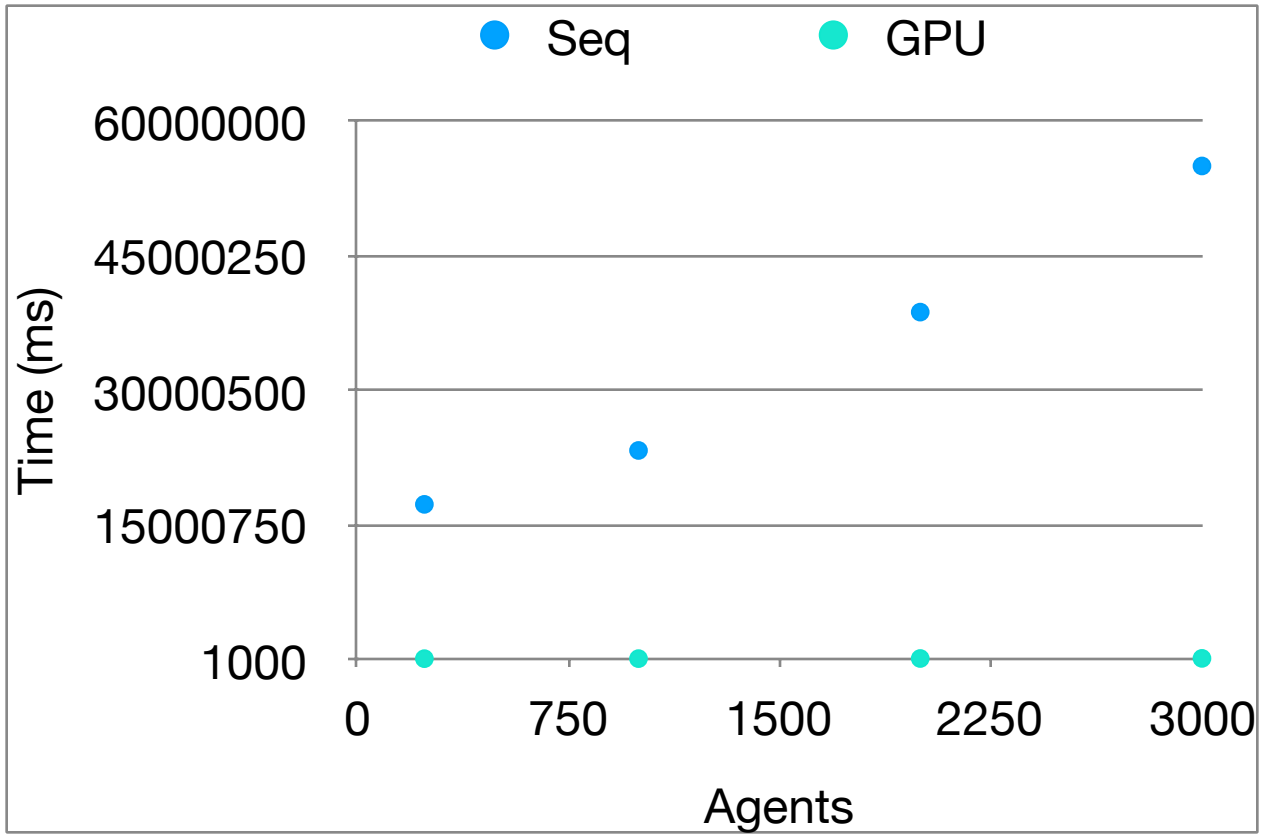


Agents	Coll_Thread4	Coll_seq
240	3334	303030
1000	10109	6172839
2000	26301	21097046
3000	38850	38759689
6000	77399	80645161

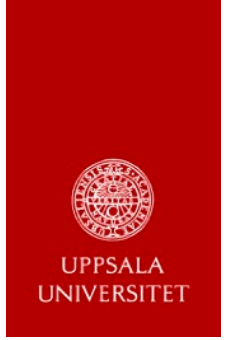
UPPSALA UNIVERSITET

Performance

- 385x speedup with heat map



Agents	GPU	Seq
240	110132	17314997
1000	115740	23325029
2000	129533	38750077
3000	142857	55044982



Takeaways

- “Effective code isn’t always good looking code”
- SIMD
- OpenMP