SURF SARA

HPC center of the Netherlands

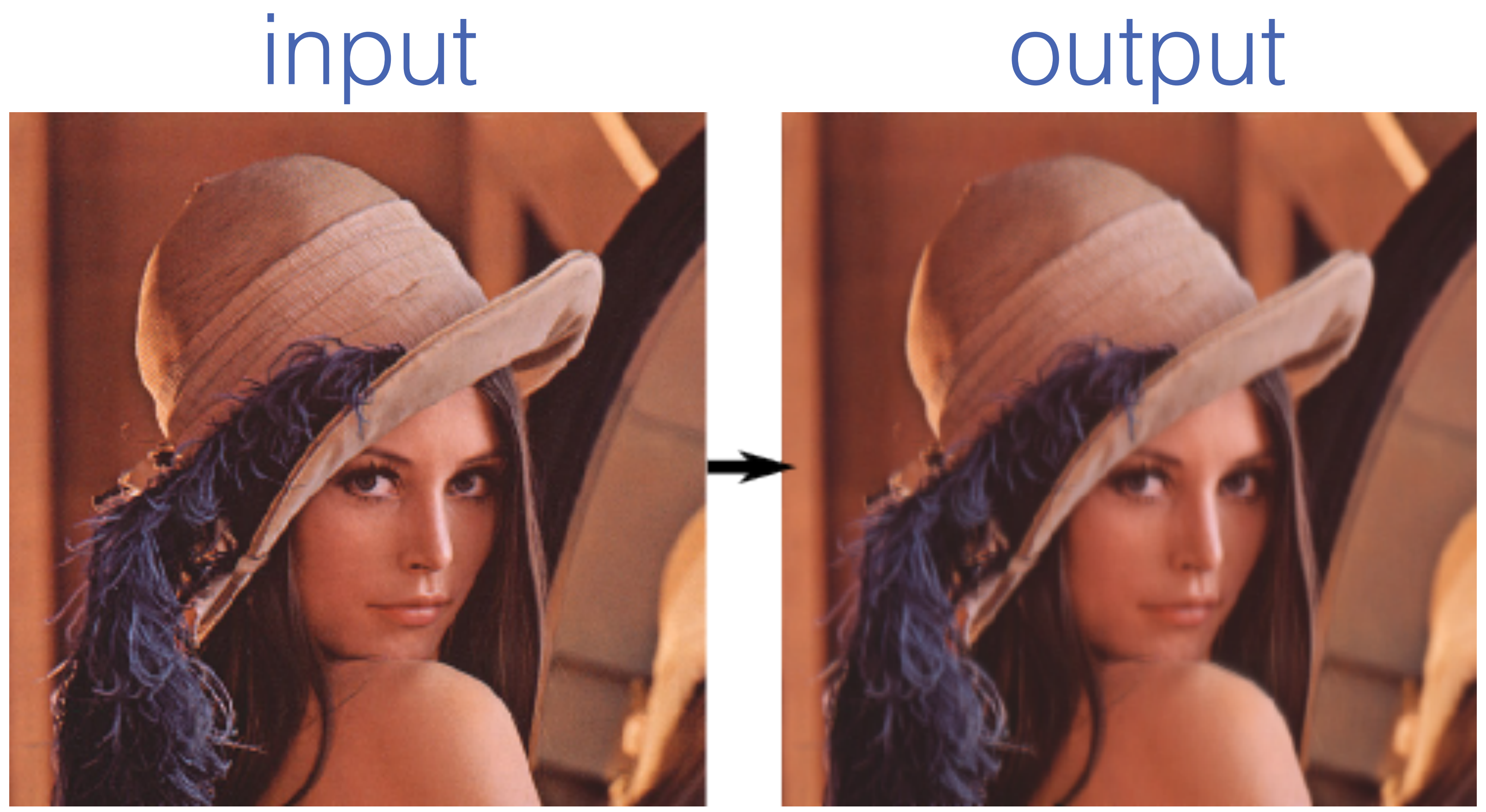
CLTune:
A Generic Auto-Tuner for OpenCL Kernels

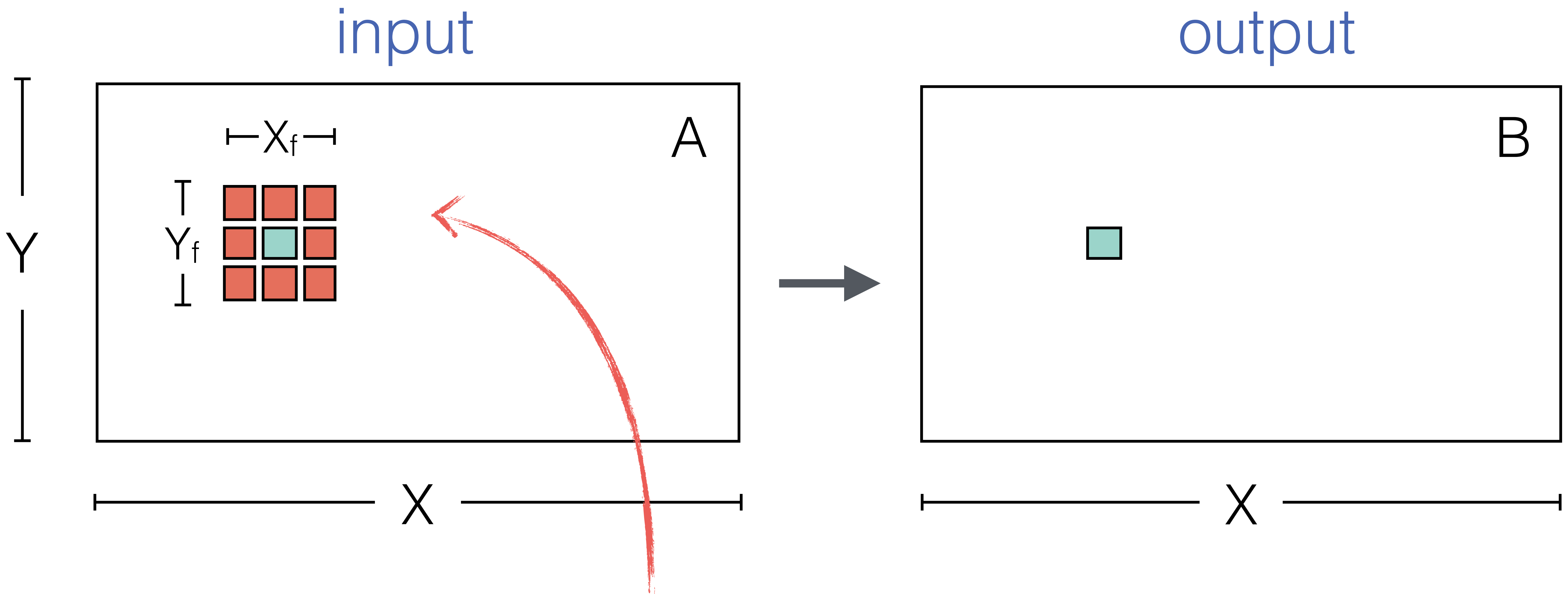
IEEE MCSoC
September 24, 2015

Cedric Nugteren (presenter), Valeriu Codreanu

Example: convolution

Example: blur filter





filter size: X_f by Y_f

filter coefficients

$$B_{x,y} = w \cdot \sum_{i=-1}^{i\leq 1} \sum_{j=-1}^{j\leq 1} F_{i,j} A_{x+i,y+j}$$

example: 3 by 3 filter

Targets:

- GPUs
- Multi-core CPUs
- Other OpenCL-capable devices

OpenCL 2D convolution

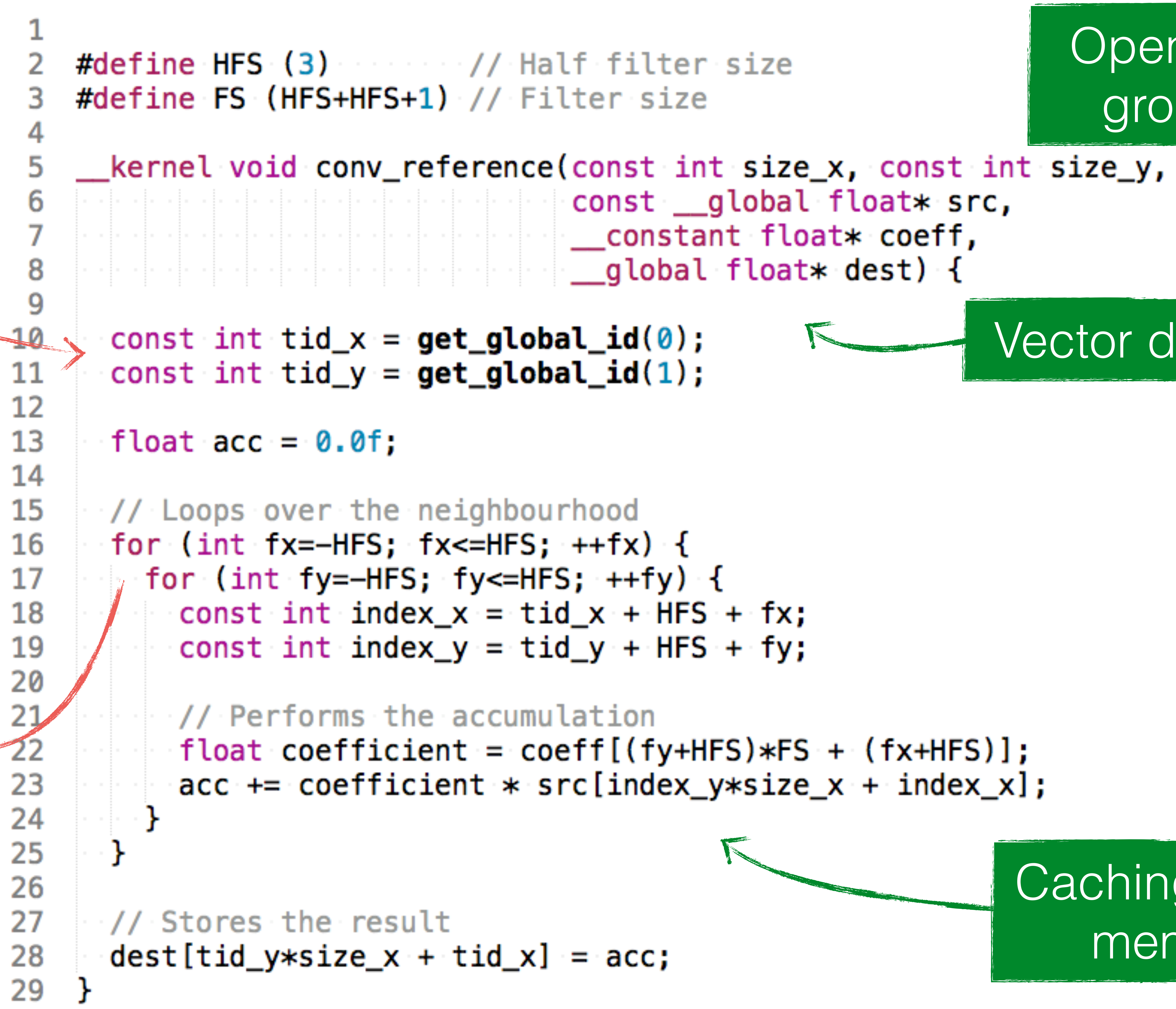
each thread: one output pixel

Thread coarsening (2D)?

$$B_{x,y} = w \cdot \sum_{i=-1}^{i\leq 1} \sum_{j=-1}^{j\leq 1} F_{i,j} A_{x+i,y+j}$$

double for-loop

Unroll loops?



```

#define HFS (3)        // Half filter size
#define FS (HFS+HFS+1) // Filter size

__kernel void conv_reference(const int size_x, const int size_y,
                             const __global float* src,
                             __constant float* coeff,
                             __global float* dest) {

  const int tid_x = get_global_id(0);
  const int tid_y = get_global_id(1);

  float acc = 0.0f;

  // Loops over the neighbourhood
  for (int fx=-HFS; fx<=HFS; ++fx) {
    for (int fy=-HFS; fy<=HFS; ++fy) {
      const int index_x = tid_x + HFS + fx;
      const int index_y = tid_y + HFS + fy;

      // Performs the accumulation
      float coefficient = coeff[(fy+HFS)*FS + (fx+HFS)];
      acc += coefficient * src[index_y*size_x + index_x];
    }
  }

  // Stores the result
  dest[tid_y*size_x + tid_x] = acc;
}
```

OpenCL work-group size?

Vector data-types?

Caching in local memory?

Search-space explosion

Thread coarsening (2D)?

OpenCL work-group size?

Caching in local memory?

16 x 2 x 16 x 4 x 5 = 10240

Unroll loops?

Vector data-types?

filter illegal configurations

3424 configurations

Large search-space:

- Not feasible to explore manually
- Perhaps not even feasible automatically?

Why do we need an auto-tuner?

Large search-space:

- Not feasible to explore manually
- Perhaps not even feasible automatically?

Caching in local memory?

OpenCL work group size?

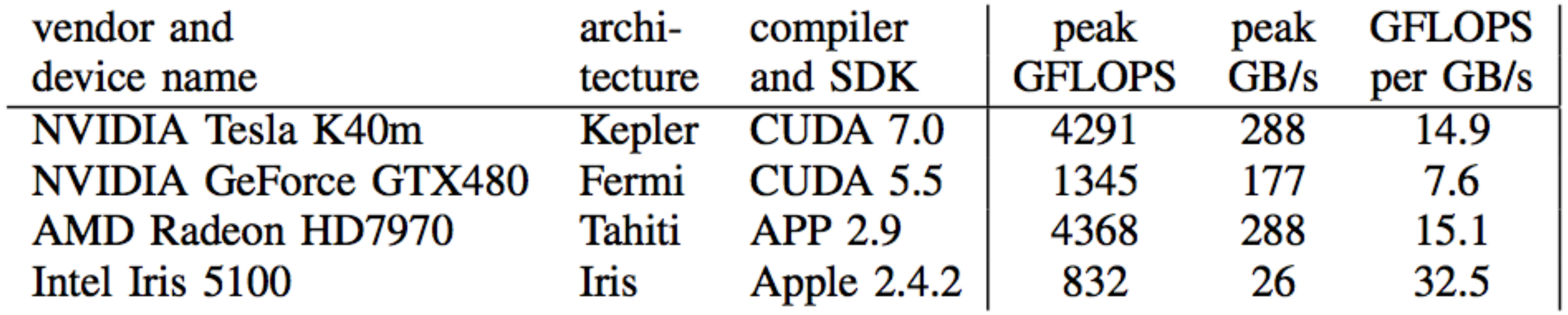
Thread coarsening (2D)?

Unroll loops?

Vector data-types?

Wide variety of devices:

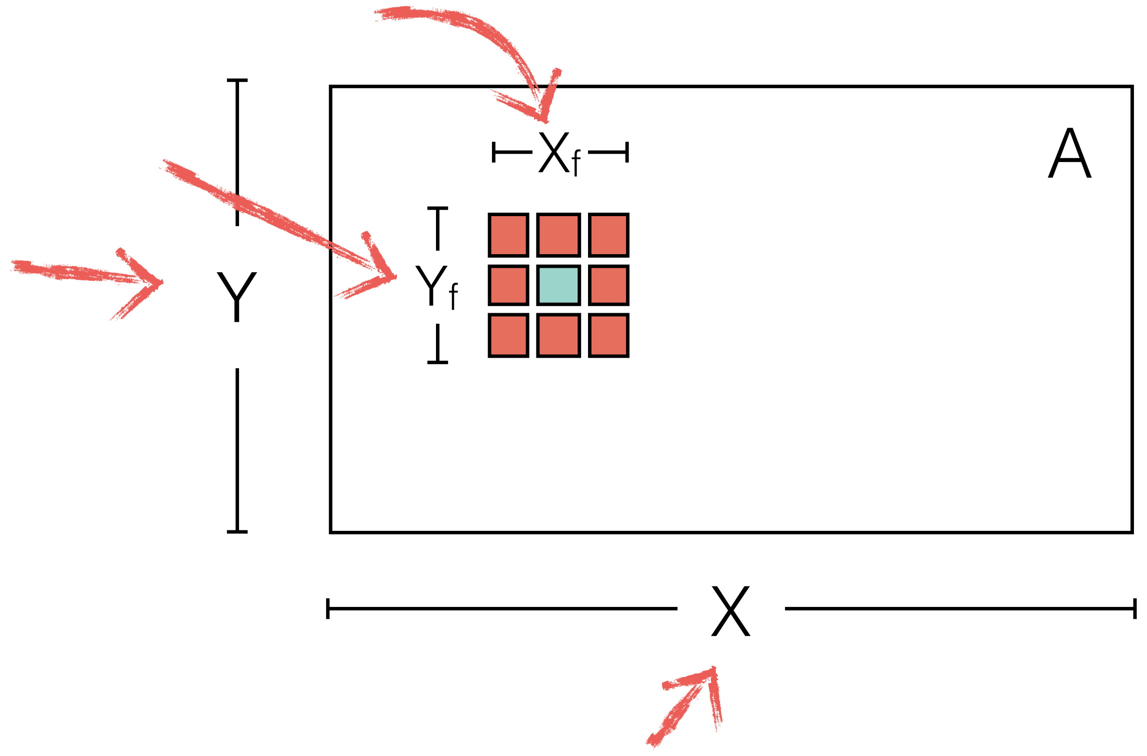
- Different optimal kernels
- Even from the same vendor



vendor and device name	archi-tecture	compiler and SDK	peak GFLOPS	peak GB/s	GFLOPS per GB/s
NVIDIA Tesla K40m	Kepler	CUDA 7.0	4291	288	14.9
NVIDIA GeForce GTX480	Fermi	CUDA 5.5	1345	177	7.6
AMD Radeon HD7970	Tahiti	APP 2.9	4368	288	15.1
Intel Iris 5100	Iris	Apple 2.4.2	832	26	32.5

User-parameter dependent:

- Examples: matrix sizes, image size, filter sizes, etc.

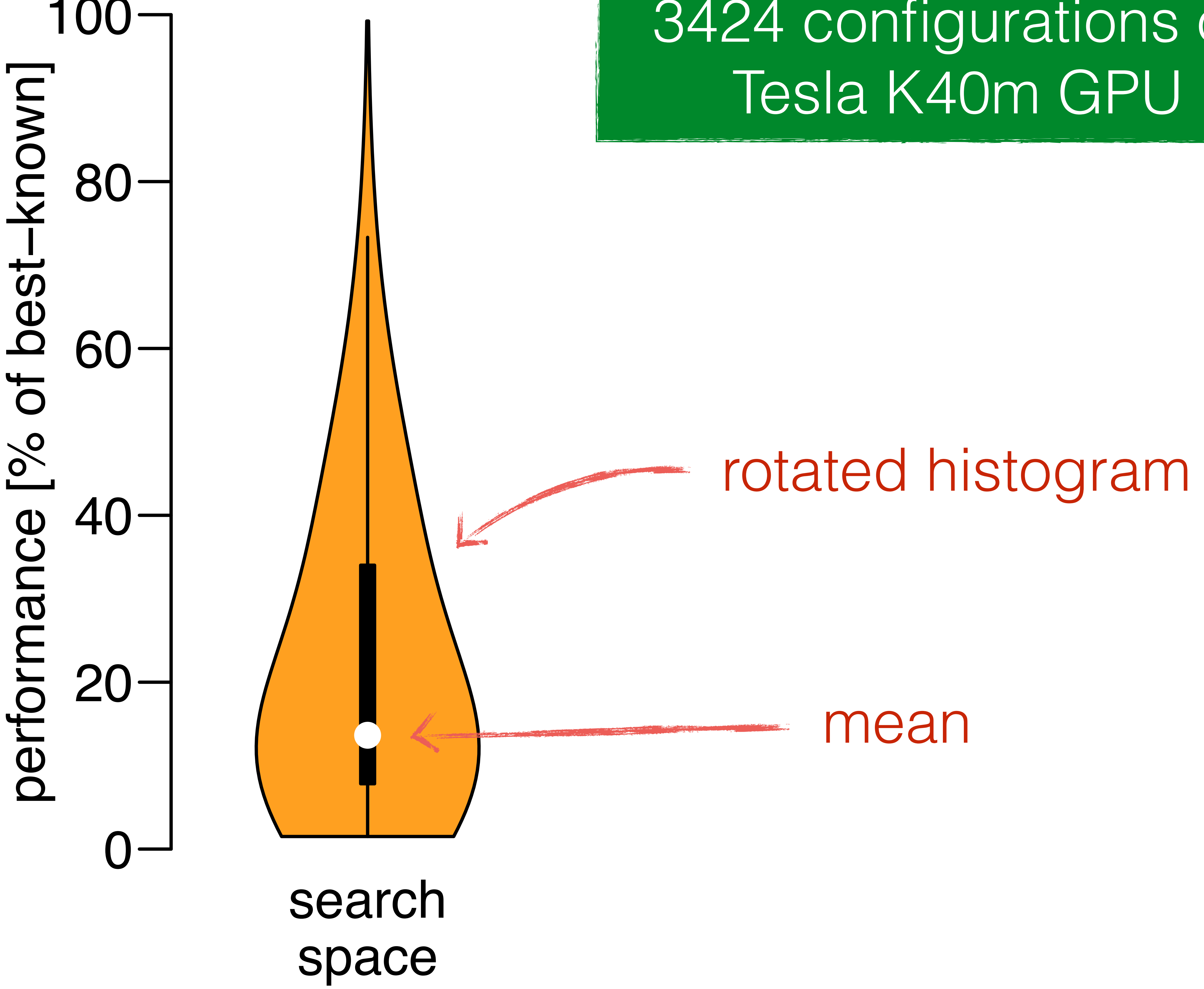


Search strategies

Option 0: Full search

☺ Finds optimal solution

☹ Explores all options



3424 configurations on
Tesla K40m GPU

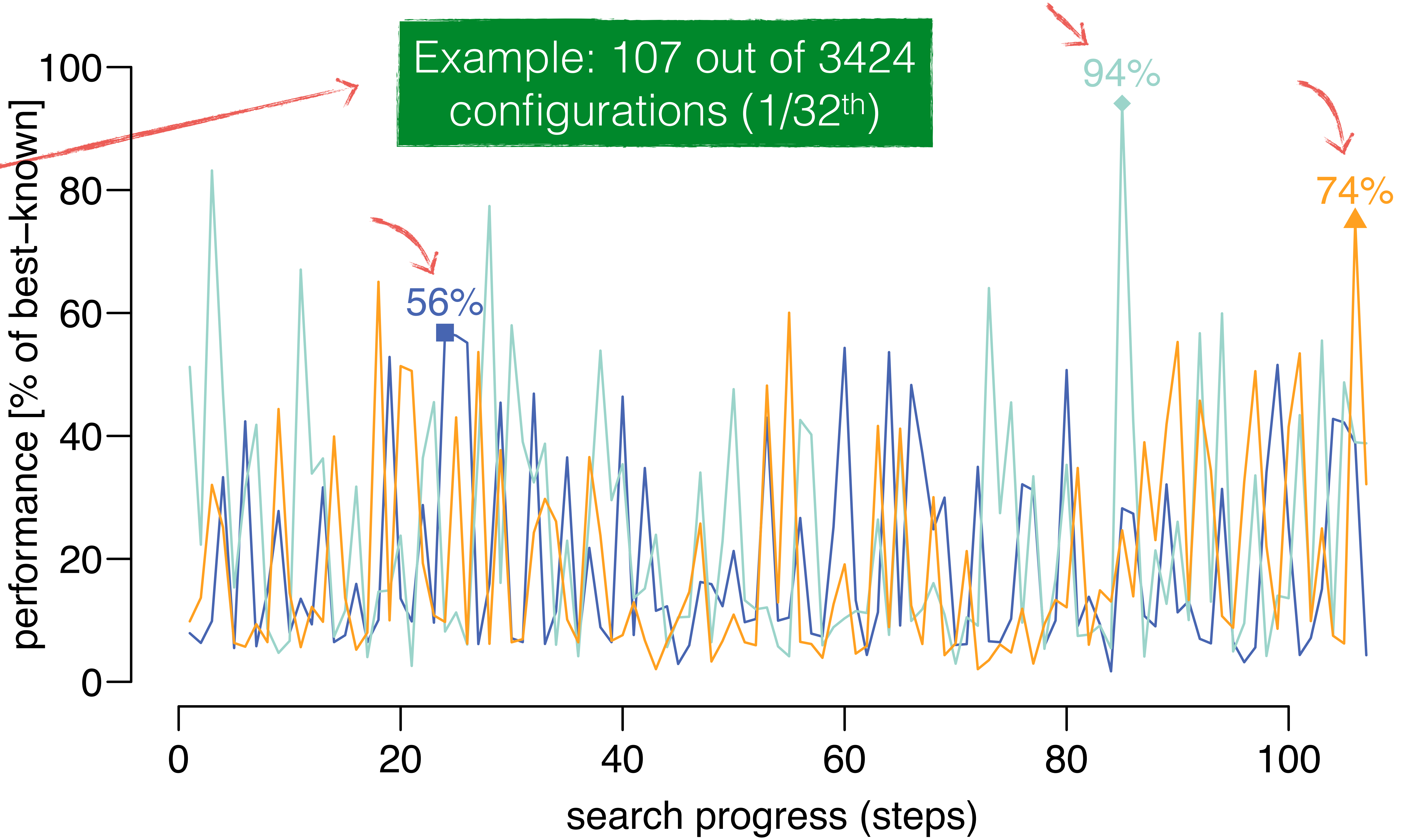
Search strategies

Option 0: Full search

Option 1: Random search

- ☺ Explores arbitrary fraction
- ☹ Performance varies

Example: 107 out of 3424 configurations (1/32th)



Colours: 3 example runs

Search strategies

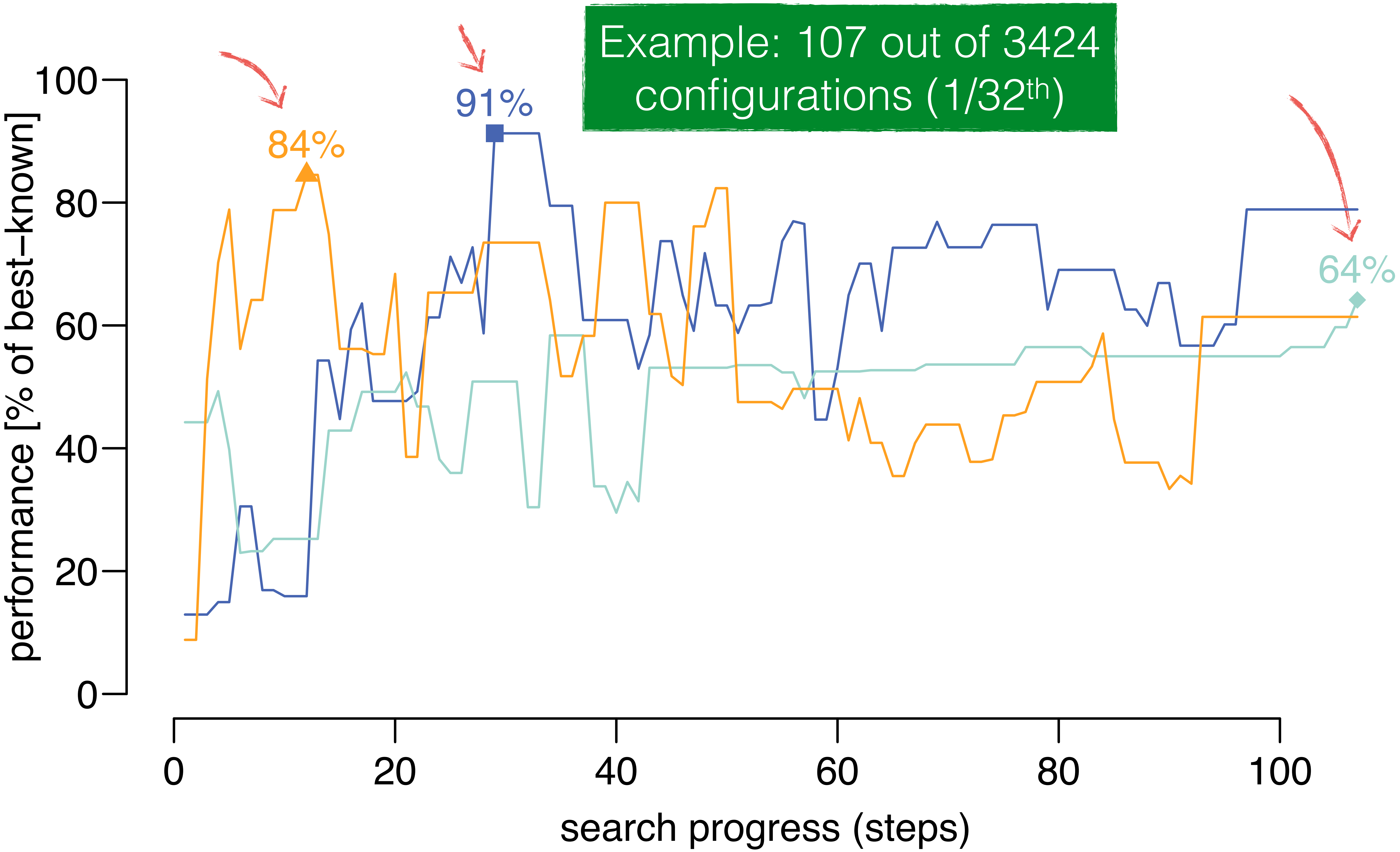
Option 0: Full search

Option 1: Random search

Option 2: Simulated annealing

- ☺ Explores arbitrary fraction
- ☹ Performance varies
- ☹ Meta-parameter
- ☹ Local optima

Colours: 3 example runs



Example: 107 out of 3424 configurations (1/32th)

Search strategies

Option 0: Full search

Option 1: Random search

Option 2: Simulated annealing

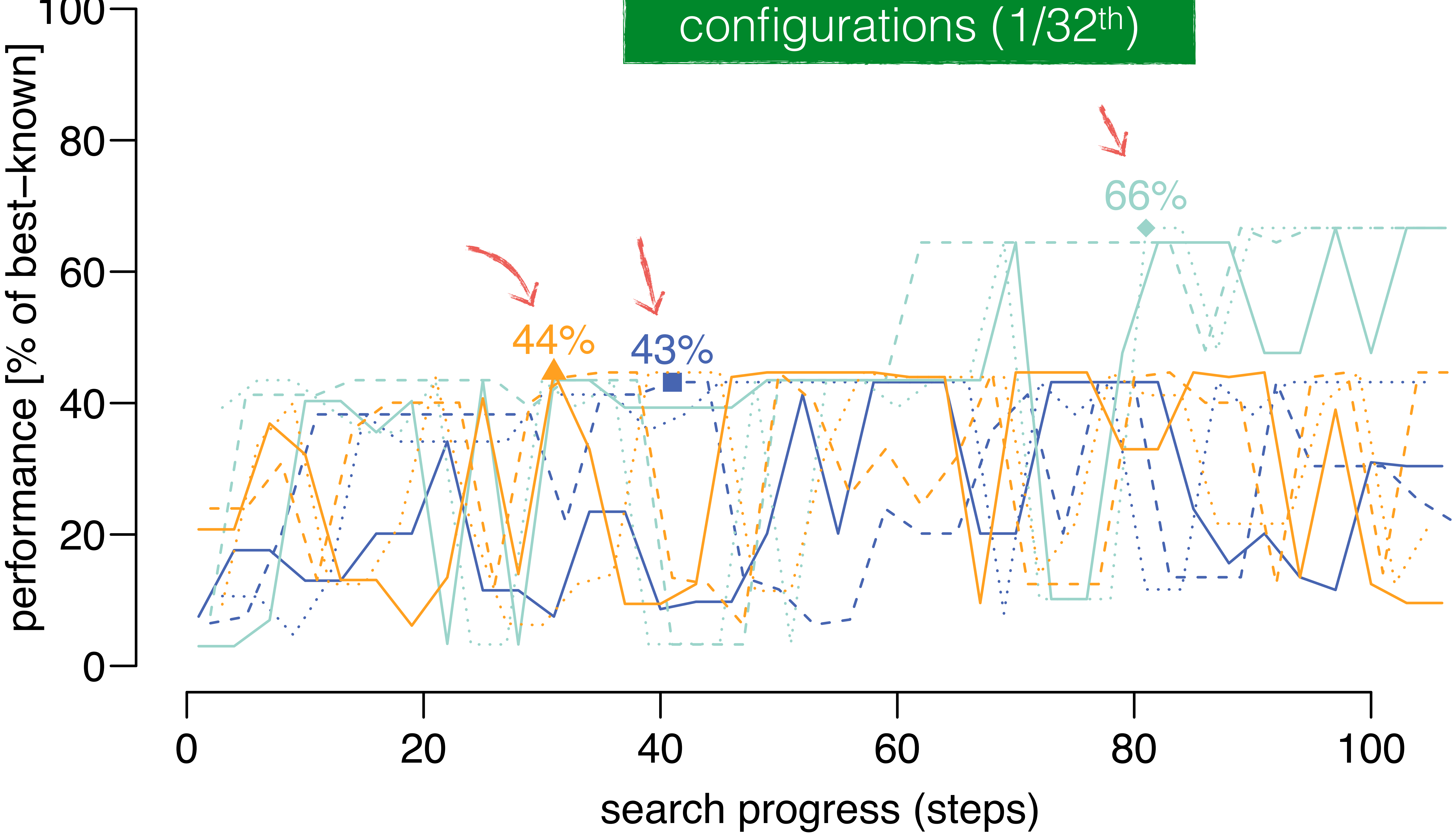
Option 3: Particle swarm optimisation

- ☺ Explores arbitrary fraction
- ☹ Performance varies
- ☹ Meta-parameter
- ☹ Local optima

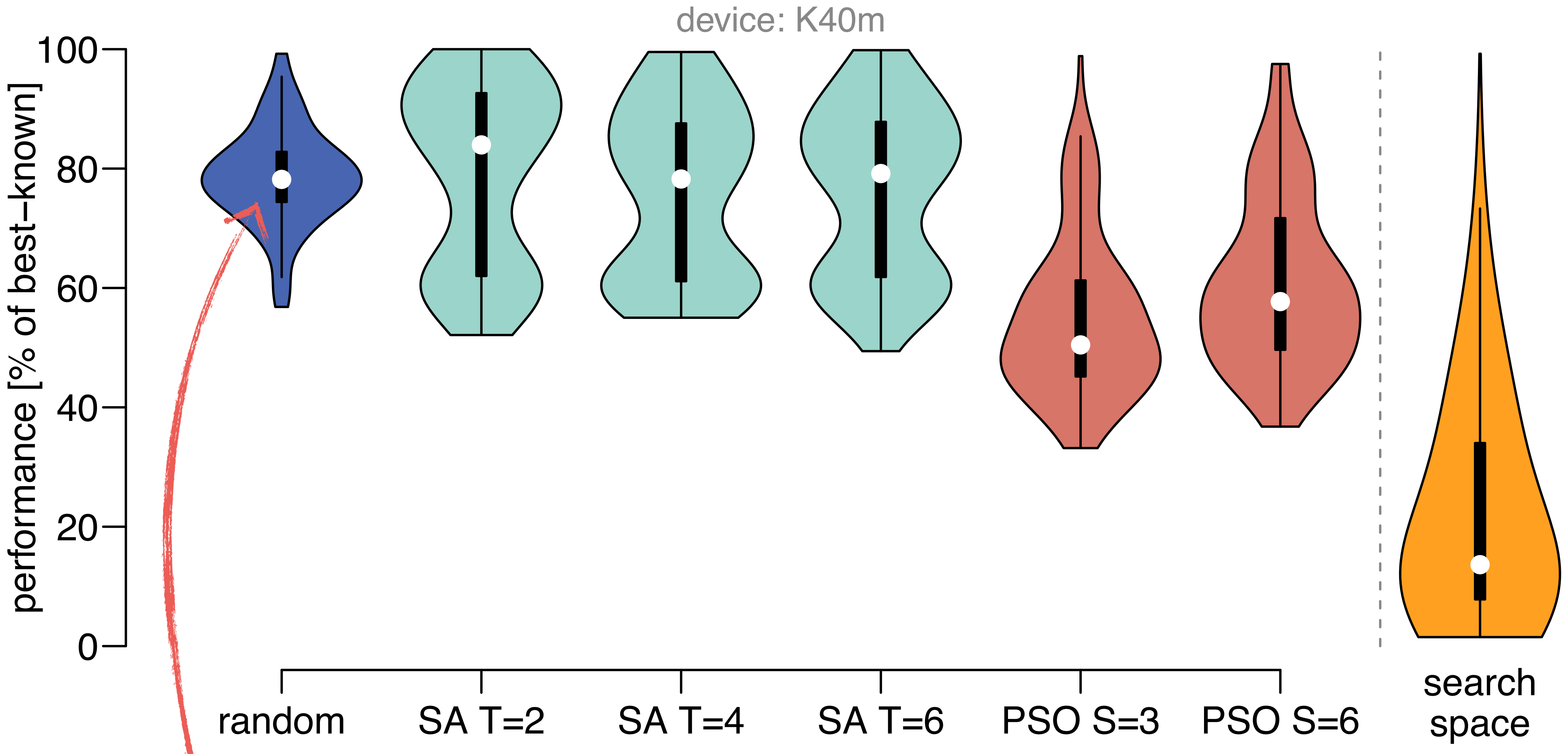
Colours: 3 example runs

Line-types: 3 swarms

Example: 107 out of 3424 configurations (1/32th)



Search strategies evaluation

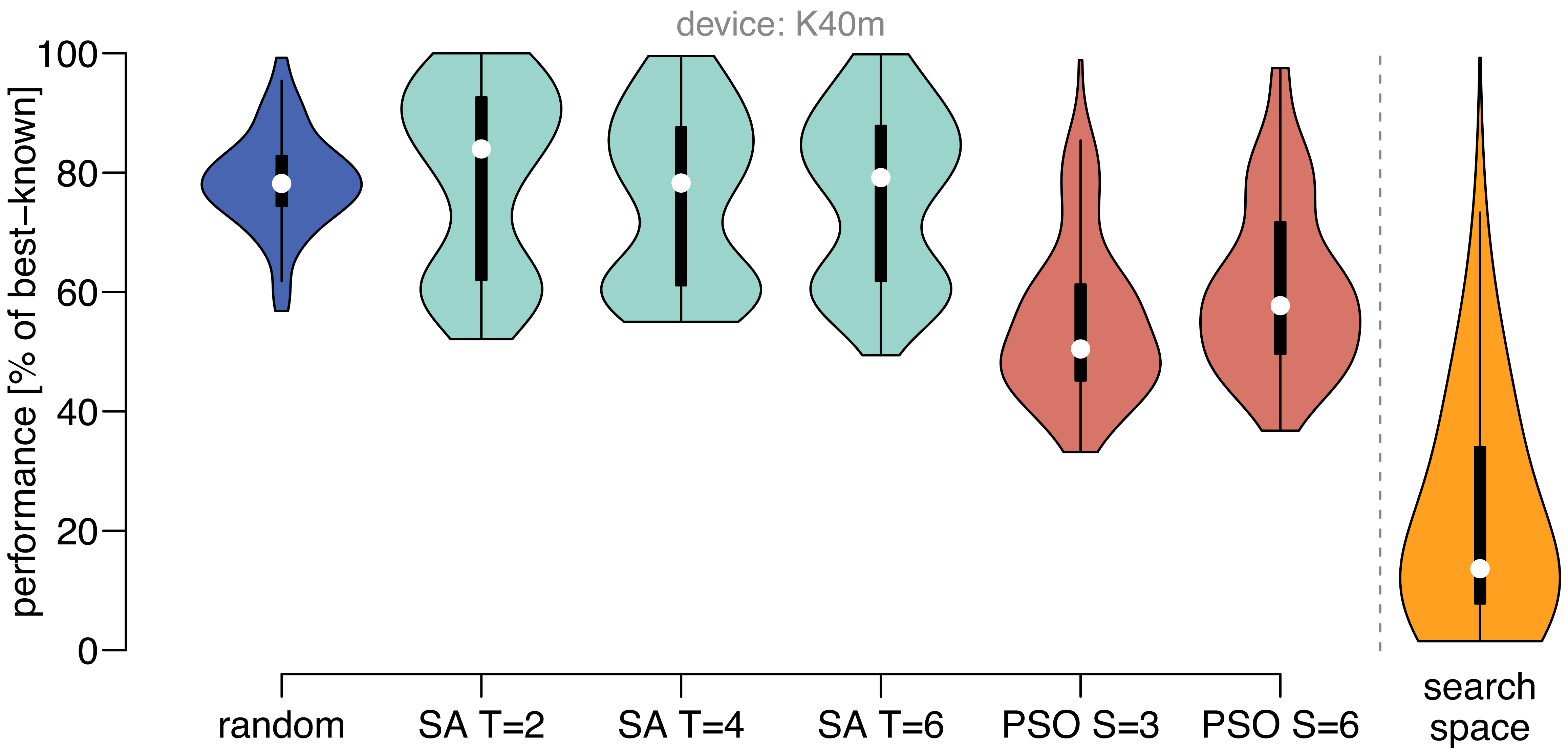


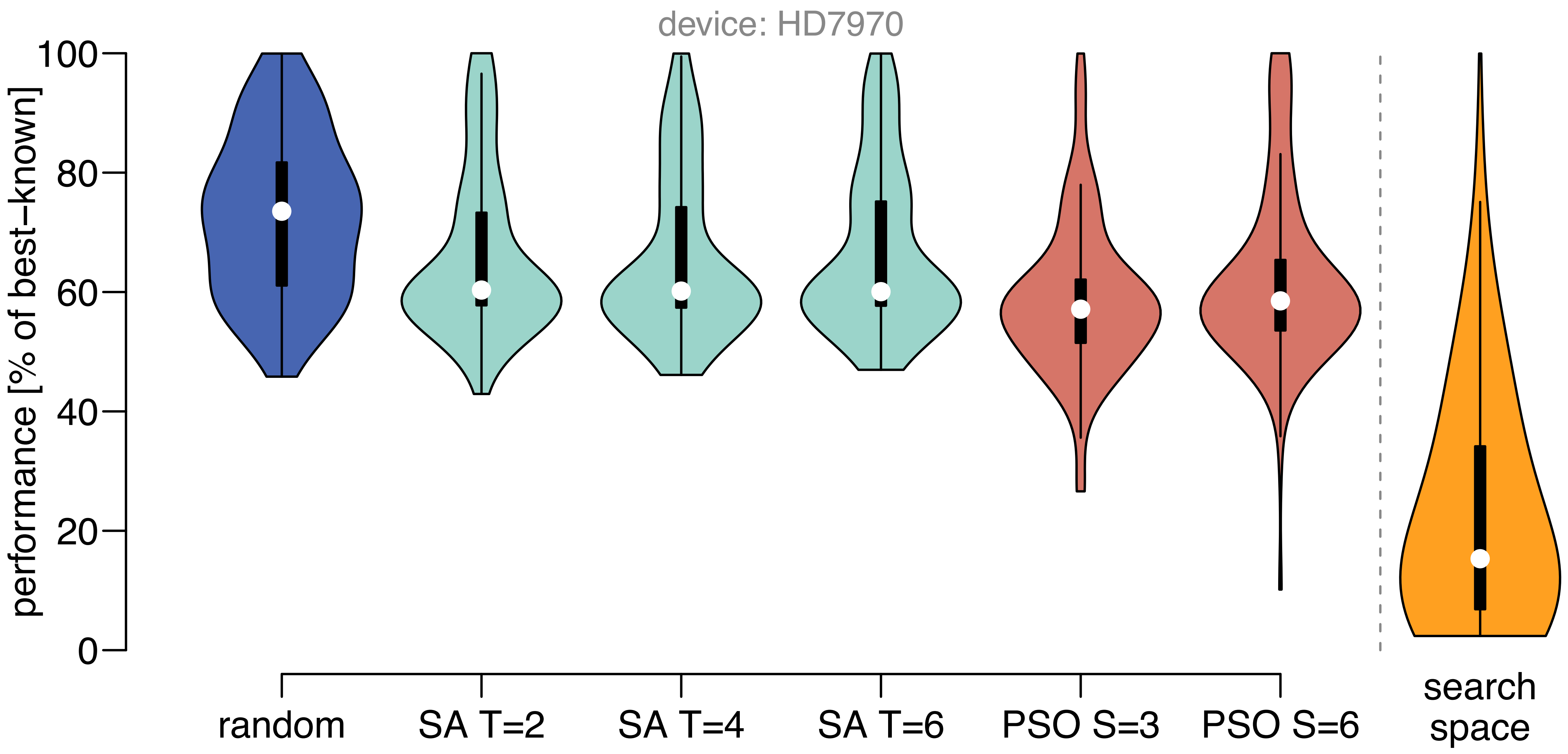
average best result
of 128 searches

meta-parameters
for SA and PSO

Each search: 107 out of 3424 configurations (1/32th)

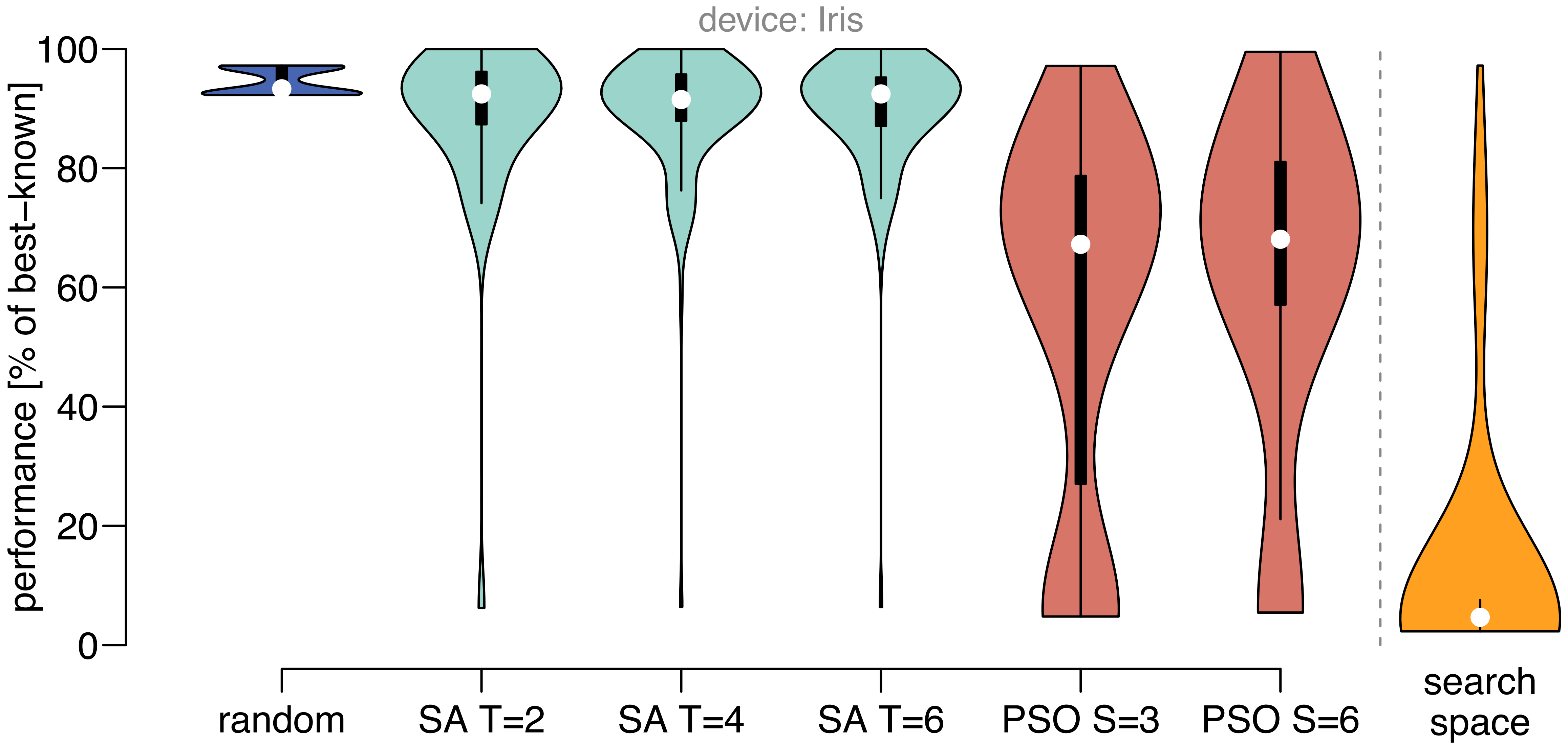
Search strategies evaluation



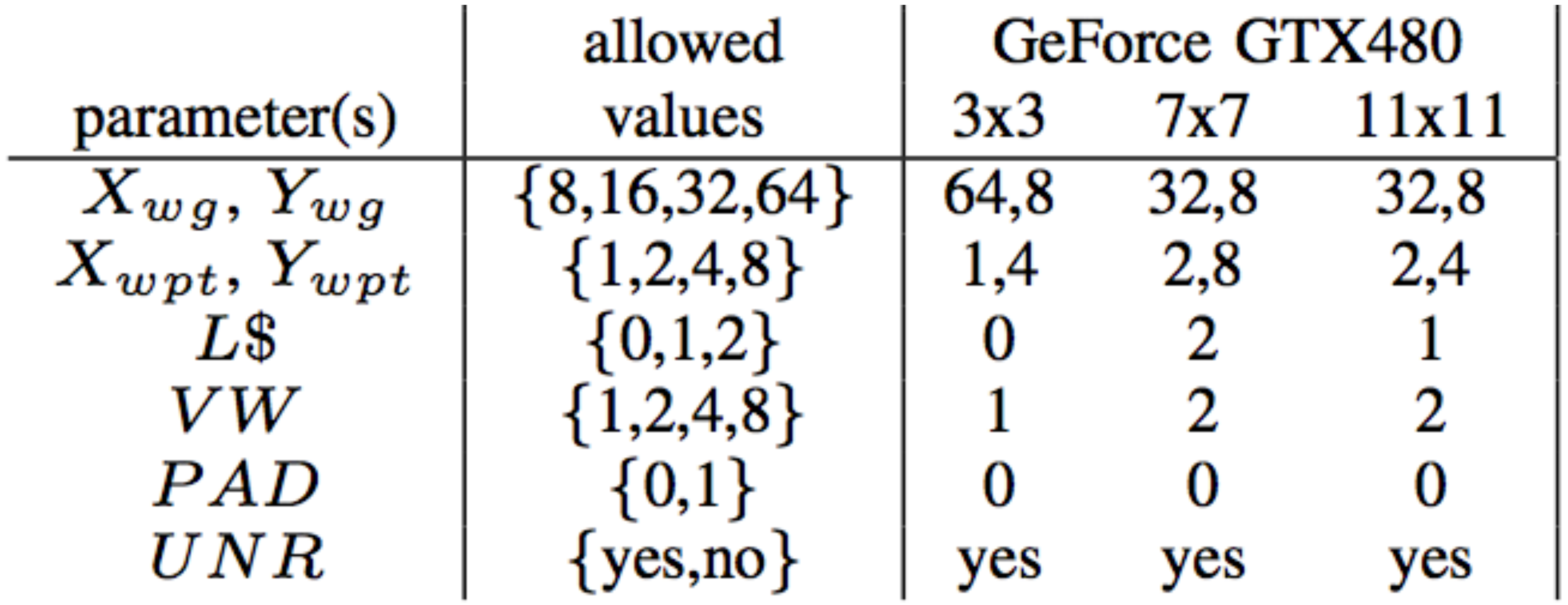


Conclusions:

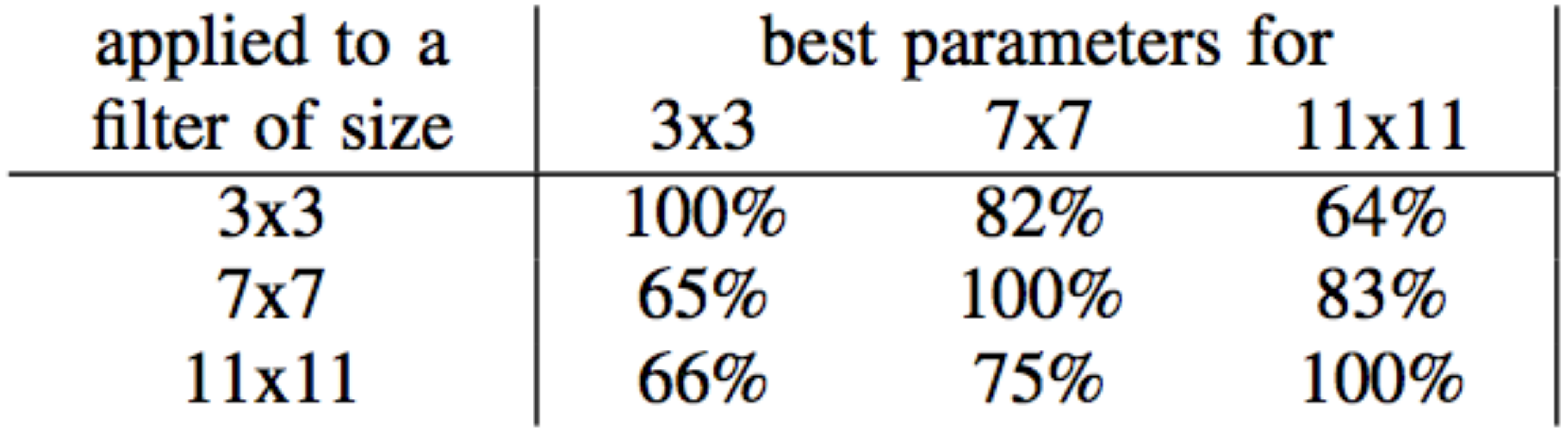
- Different per device
- PSO performs poorly
- Random search and SA perform well



Convolution case-study



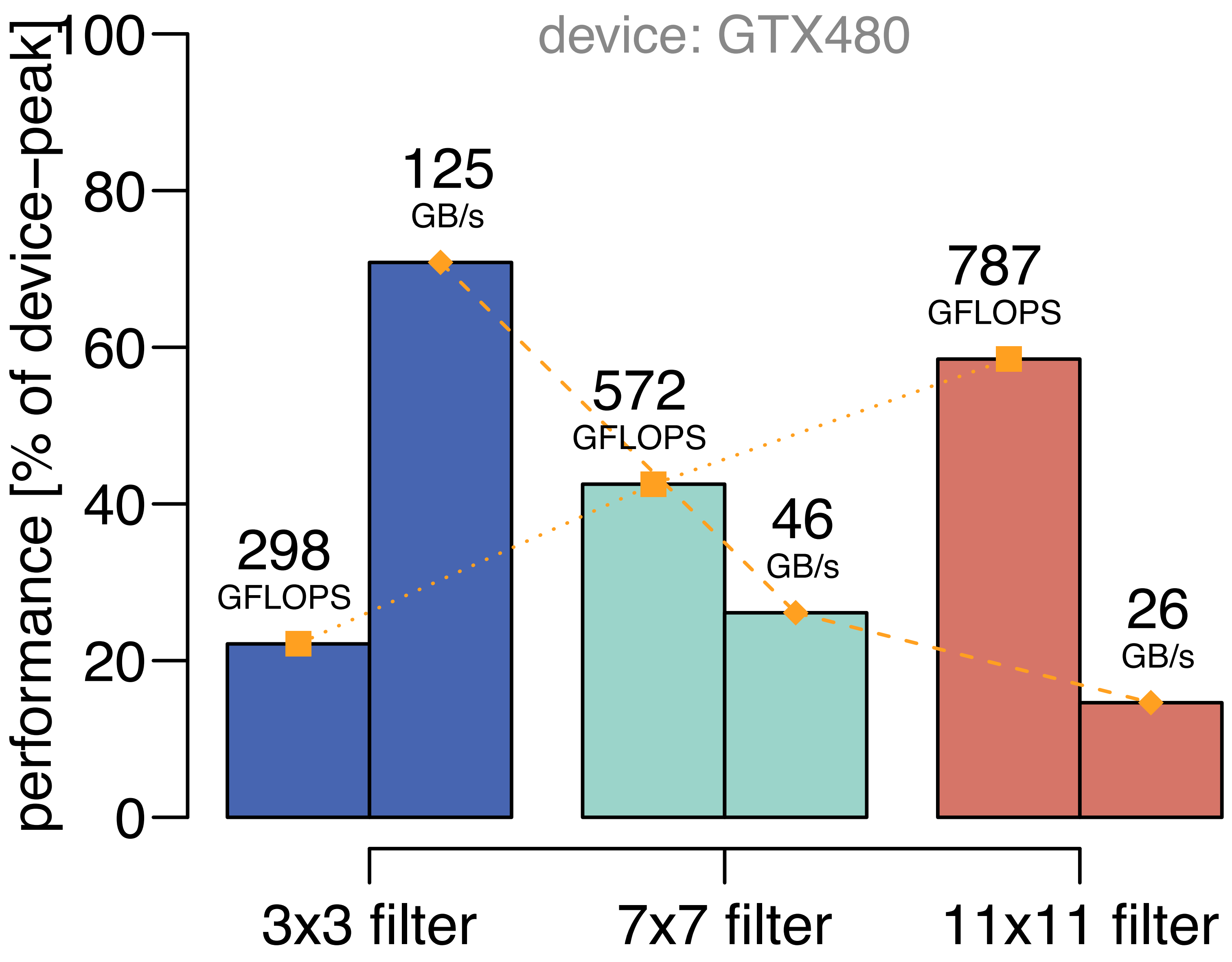
parameter(s)	allowed values	GeForce GTX480		
		3x3	7x7	11x11
X_{wg}, Y_{wg}	{8,16,32,64}	64,8	32,8	32,8
X_{wpt}, Y_{wpt}	{1,2,4,8}	1,4	2,8	2,4
$L\$$	{0,1,2}	0	2	1
VW	{1,2,4,8}	1	2	2
PAD	{0,1}	0	0	0
UNR	{yes,no}	yes	yes	yes



applied to a filter of size	best parameters for		
	3x3	7x7	11x11
3x3	100%	82%	64%
7x7	65%	100%	83%
11x11	66%	75%	100%

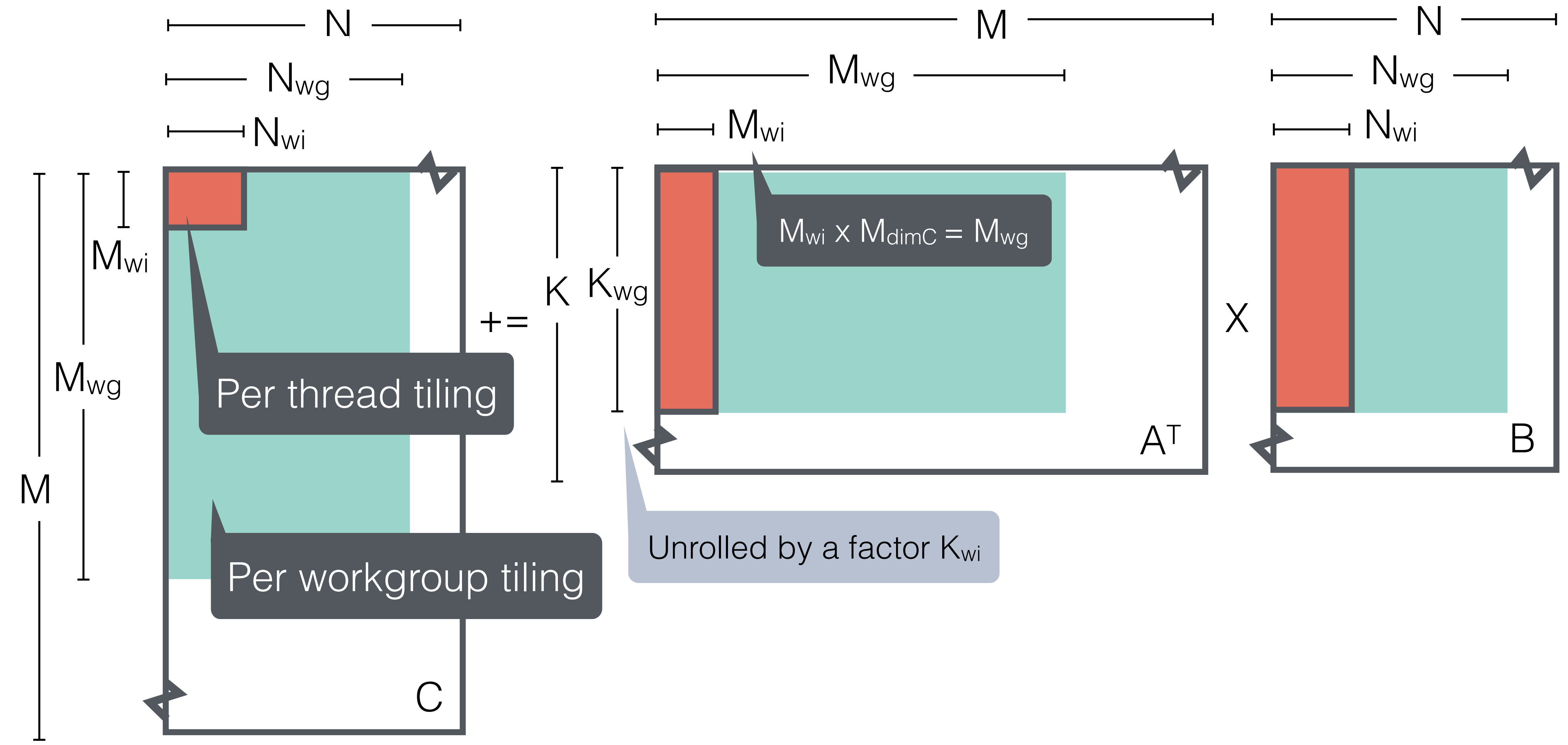
Conclusions:

- Different best parameters for different:
  - devices (see paper)
  - filter-sizes
- Performance equal or better than the state-of-the-art [1]



[1]: B. Van Werkhoven, J. Maassen, H.E. Bal, and F.J. Seinstra. Optimizing Convolution Operations on GPUs Using Adaptive Tiling.

GEMM case-study



Per thread tiling

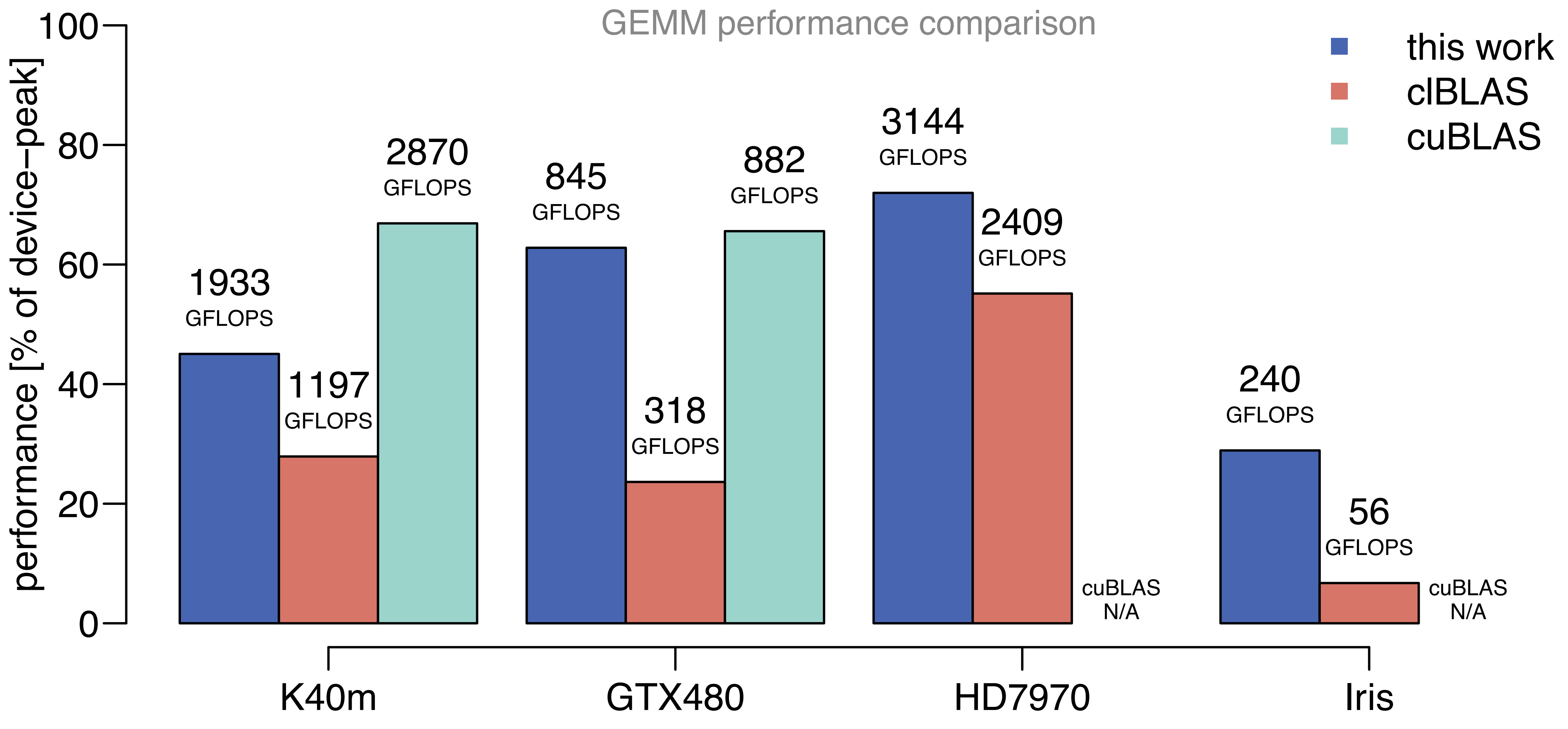
Per workgroup tiling

Unrolled by a factor Kwi

parameter(s)	allowed values	best parameters per device			
		K40m	GTX480	HD7970	Iris
M_{wg}, N_{wg}, K_{wg}	{16,32,64,128}	128,128,16	64,64,32	128,128,32	64,64,16
M_{dimC}, N_{dimC}	{8,16,32}	16,16	8,16	16,16	8,8
$L\$_A, L\$_B$	{yes,no}	yes, yes	yes, yes	yes, yes	yes, yes
M_{dimA}, N_{dimB}	{8,16,32}	32,16	32,32	32,32	8,16
M_{stride}, N_{stride}	{yes,no}	yes, no	yes, no	no, yes	yes, yes
M_{vec}, N_{vec}	{1,2,4,8}	2,1	2,2	4,4	4,4
K_{wi}	{2,8}	8	8	2	8

Conclusions:

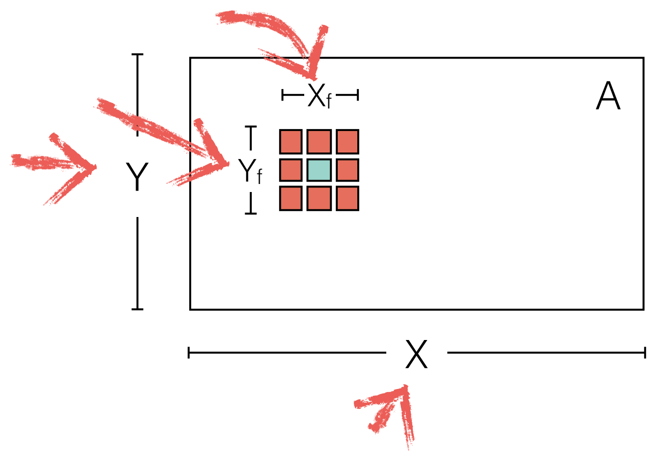
- Different best parameters for different devices
- Performance better than clBLAS, but not as good as assembly-tuned cuBLAS



CLTune:
A Generic Auto-Tuner for OpenCL Kernels

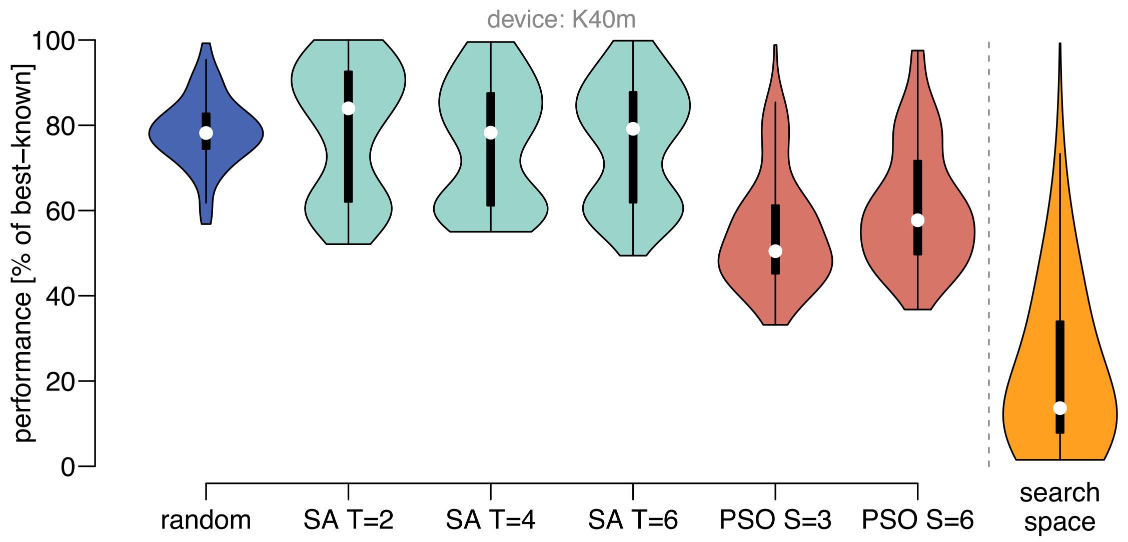
Auto-tuning OpenCL kernels:

- Large search-space
- Wide variety of devices
- User-parameter dependent



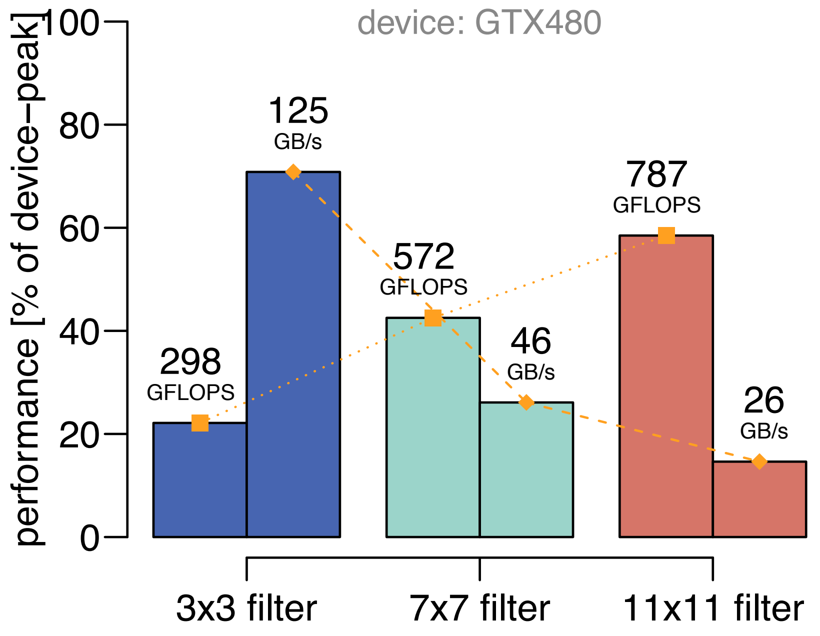
Advanced search strategies:

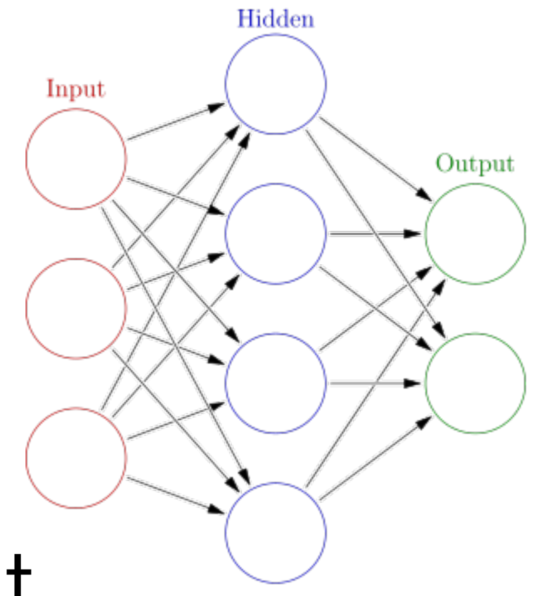
- Simulated annealing
- Particle swarm optimisation



Case-studies:

- Fastest 2D convolution
- Fast matrix-multiplication





Future: machine-learning [2]

- Train a model on a small subset
- Use the model to predict the remainder

Source-code on GitHub:
https://github.com/CNugteren/CLTune

[2]: T.L. Falch and A.C. Elster. Machine Learning Based Auto-tuning for Enhanced OpenCL Performance Portability.