HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

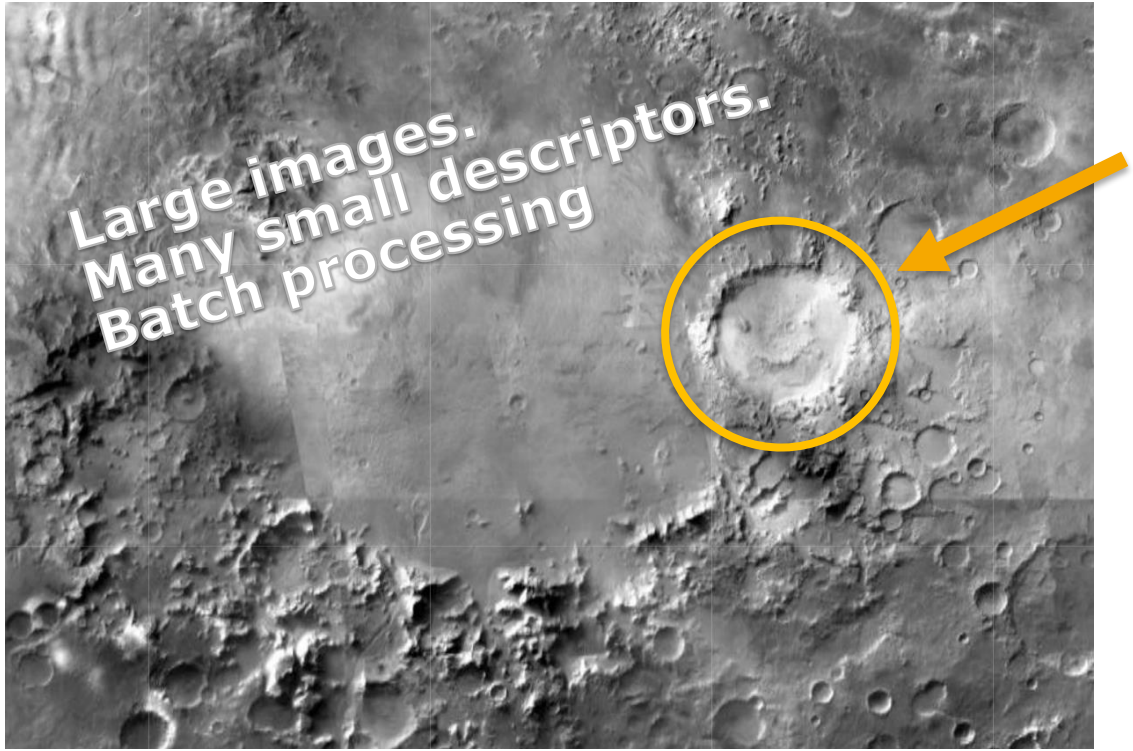
Scalable SIFT for NUMA with Actors

Frank Feinbube, Lena Herscheid,
Christoph Neijenhuis, Peter Tröger

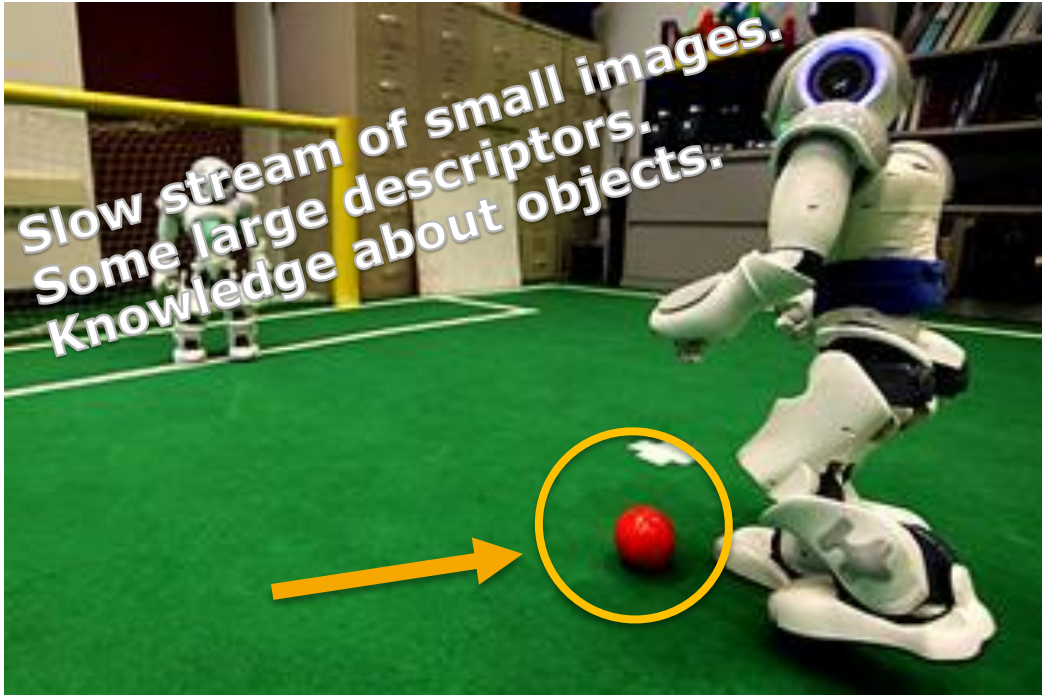
Hasso Plattner Institute for IT Systems Engineering

What is Scale Invariant Feature Transform (SIFT) good for?

This is what SIFT was used for:



This is what SIFT was used for:



This is what we wanted to use SIFT for:

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

"Distinctive Image Features from Scale-Invariant Keypoints"

International Journal of Computer Vision, 2004

David Lowe

Computer Science Department
2366 Main Mall
University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada

E-mail: lowe@cs.ubc.ca

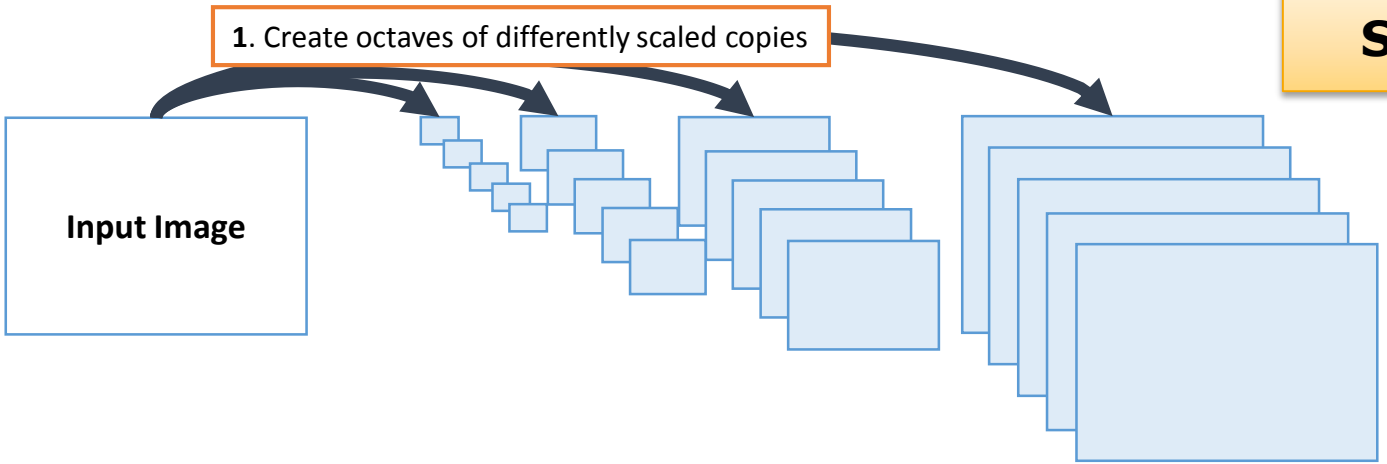
How does Scale Invariant Feature Transform (SIFT) work?

Hasso Plattner Institut

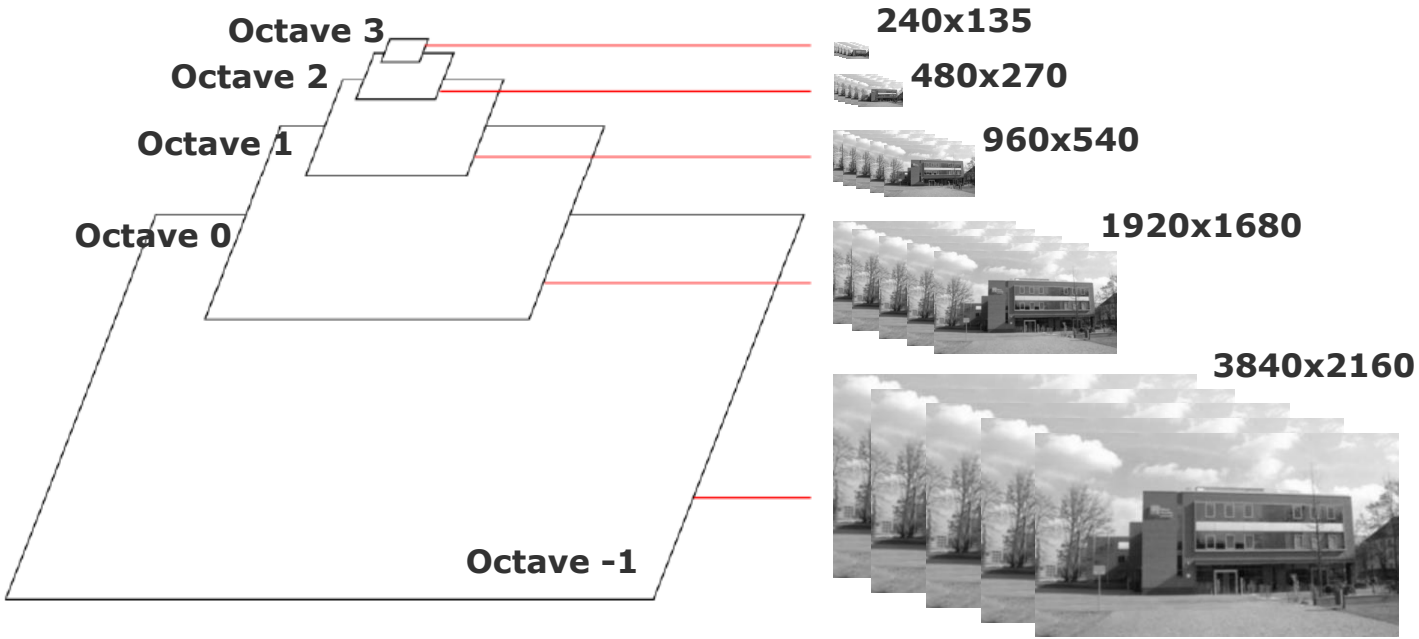
Scalable SIFT for NUMA with Actors

Frank Feinbube,
Research Assistant

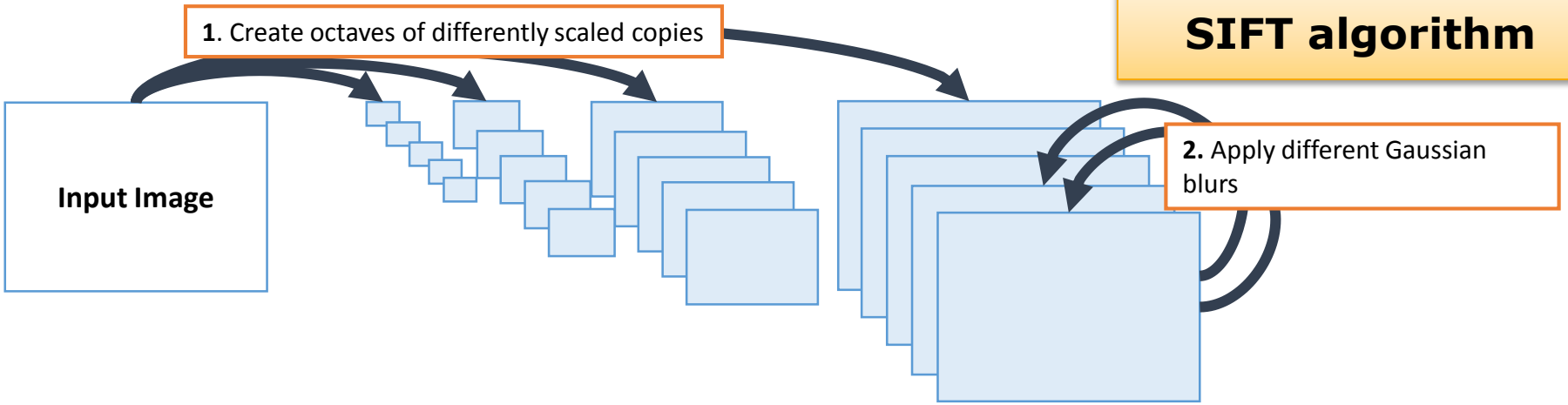
SIFT algorithm



1. Create octaves of differently scaled copies

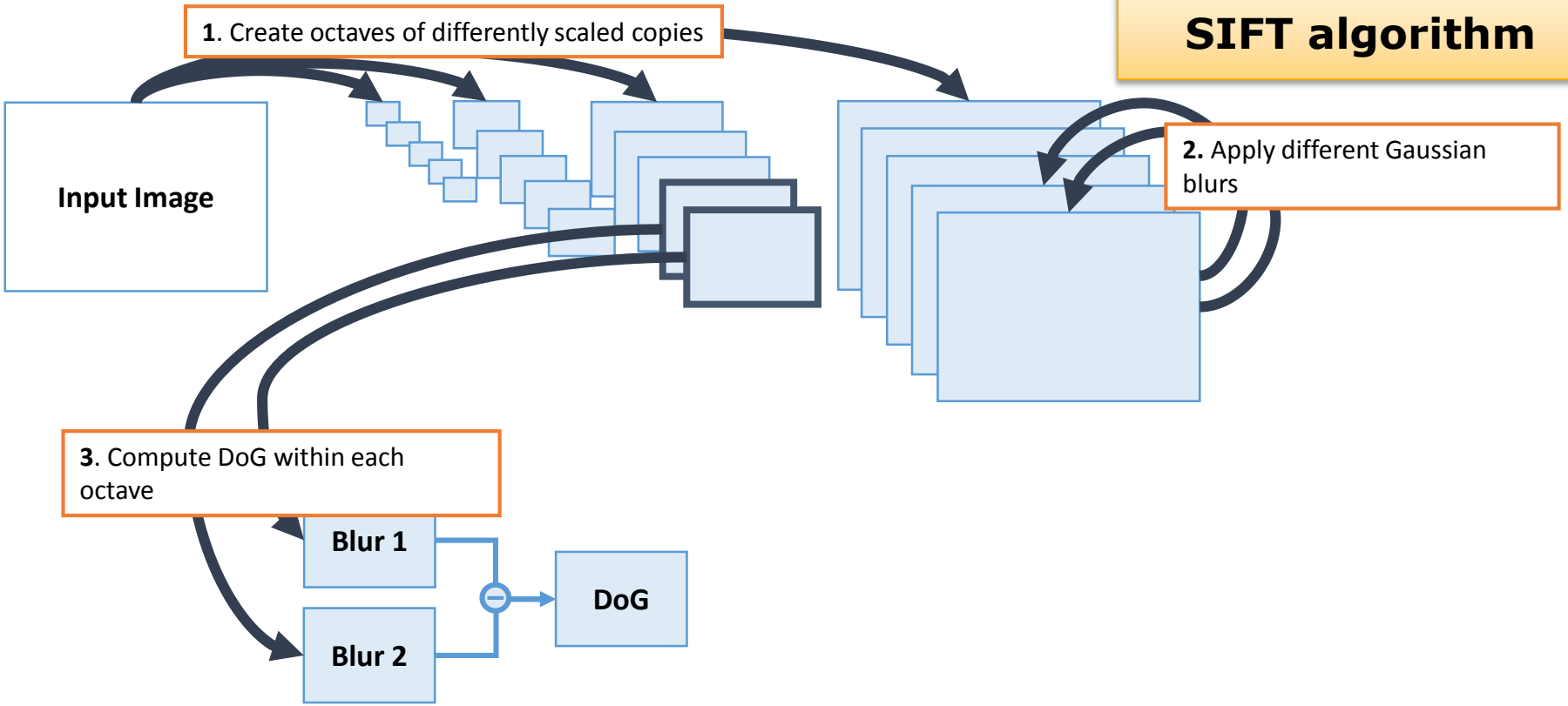


SIFT algorithm



2. Apply different Gaussian blurs

SIFT algorithm



3. Compute DoG (difference of gaussians)

Two different Gaussian blur filters are applied to the same image.

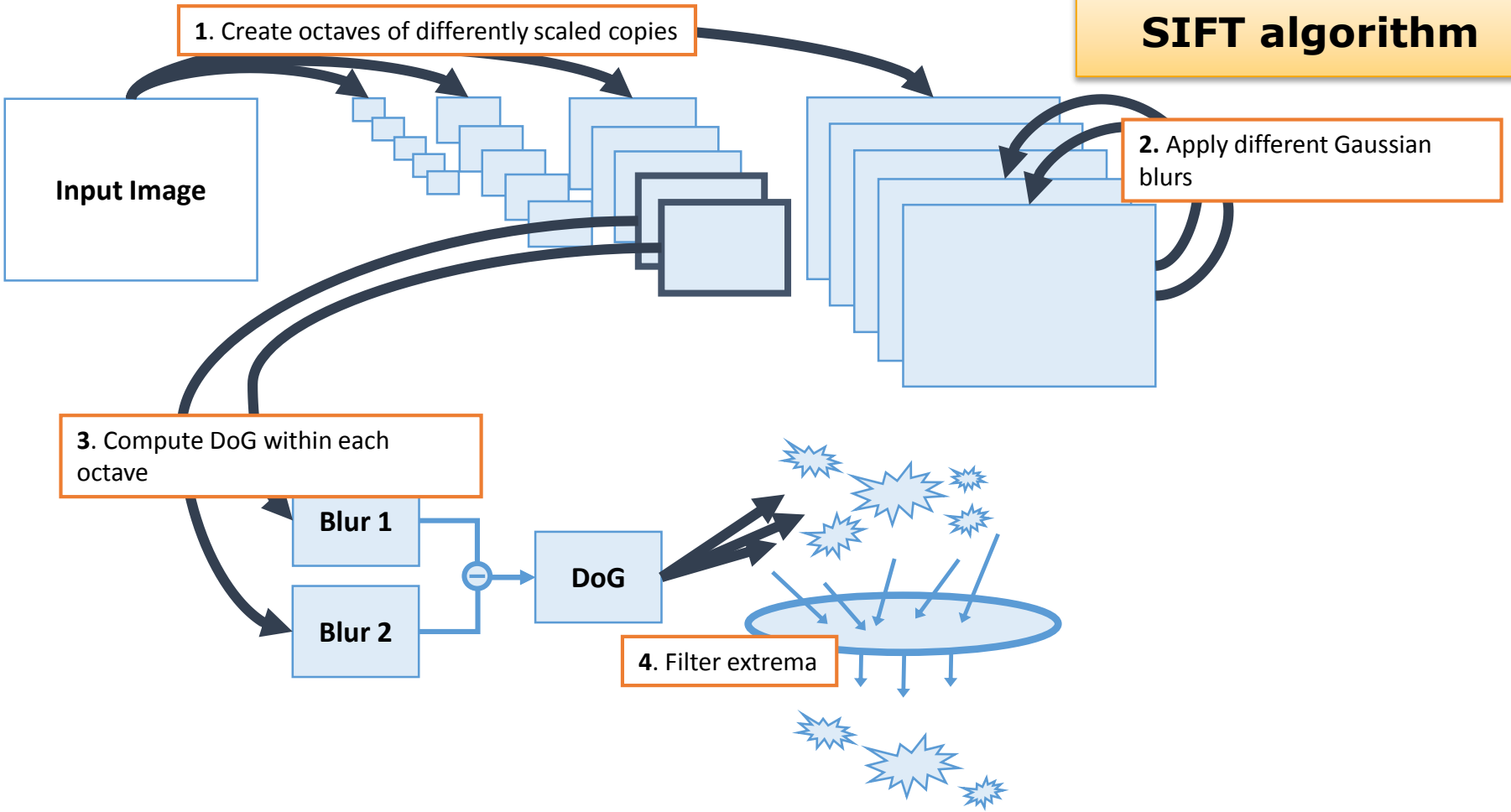
The difference between the two resulting images highlights the main image characteristics.

3. Compute DoG within each octave

Two different Gaussian blur filters are applied to the same image.

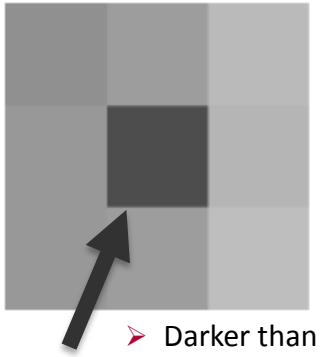
The difference between the two resulting images highlights the main image characteristics.

SIFT algorithm

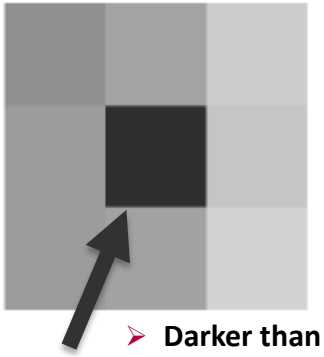


4. Filter extrema

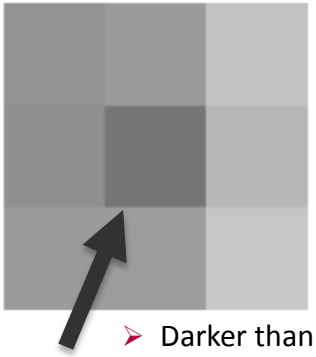
Extrema Detection:



- Darker than it‘s neighbors



- **Darker than it‘s neighbors**
- **Darker than the ones in the other scales!**

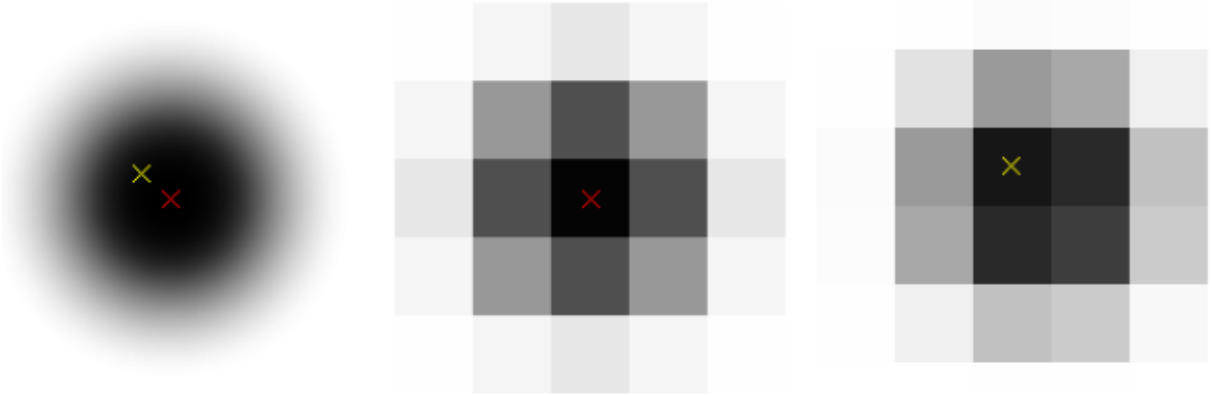


- Darker than it‘s neighbors

4. Filter extrema

Extrema Filtering:

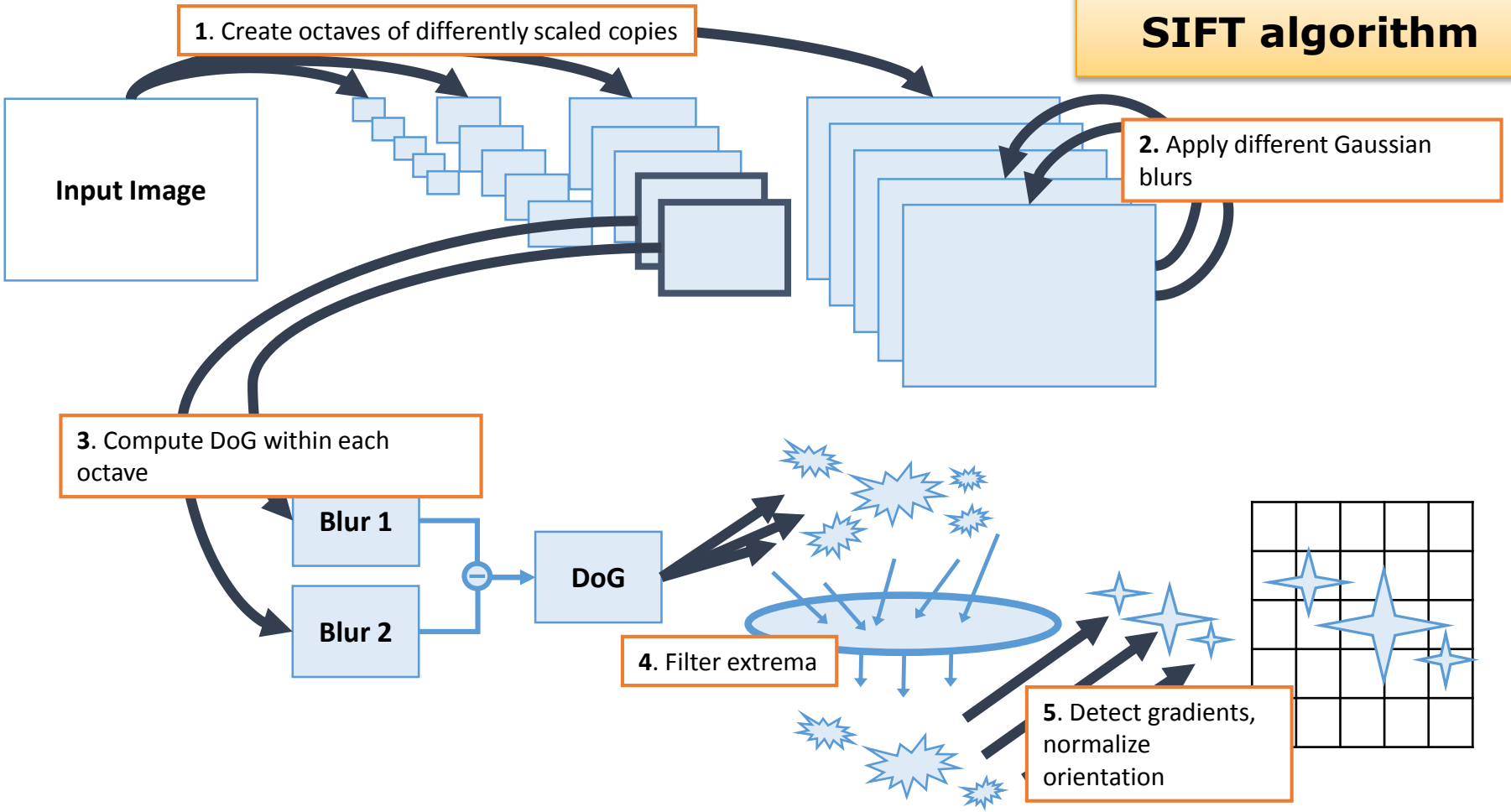
- Due to rasterization extrema might be located at different pixels leading to different descriptors.



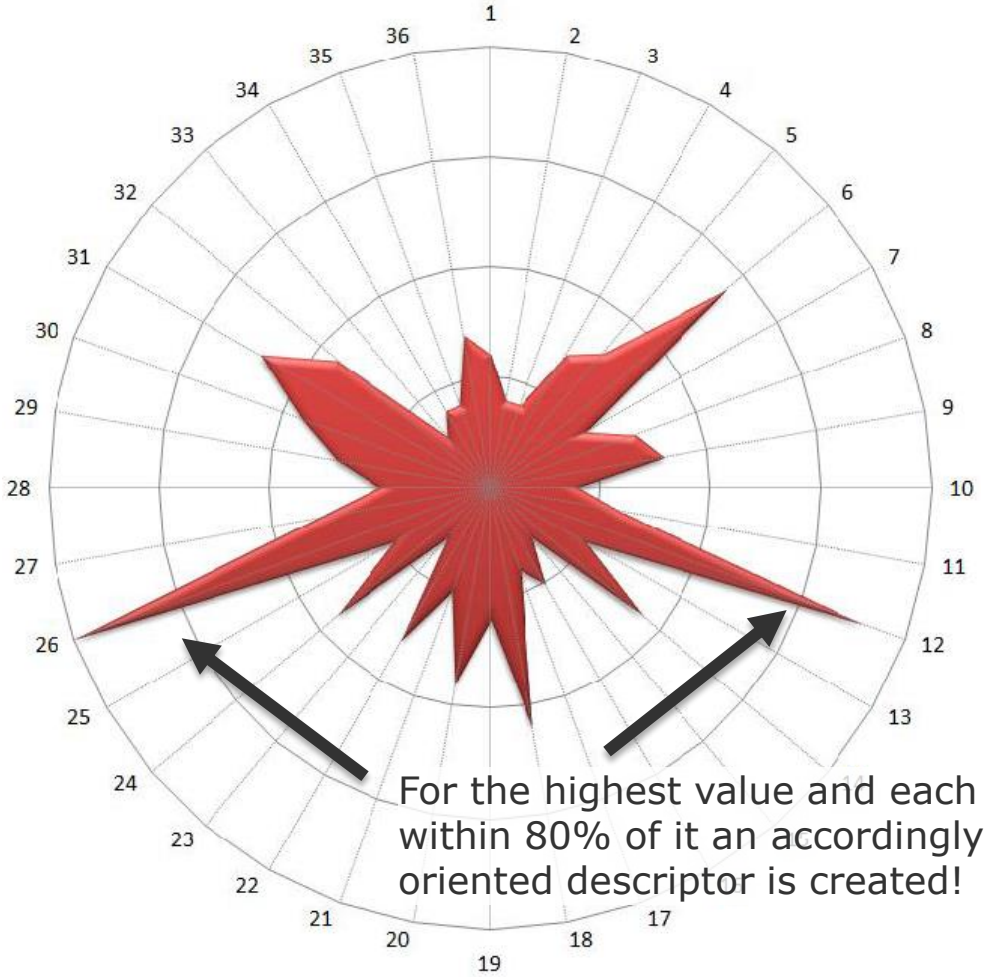
Using sub-pixel positions and sub-scale positions for interpolation increases the probability to recognize a detector about 10% to 25%.

[M. Brown and D. G. Lowe, „Invariant features from interest point groups," in British Machine Vision Conference, 2002.]

SIFT algorithm



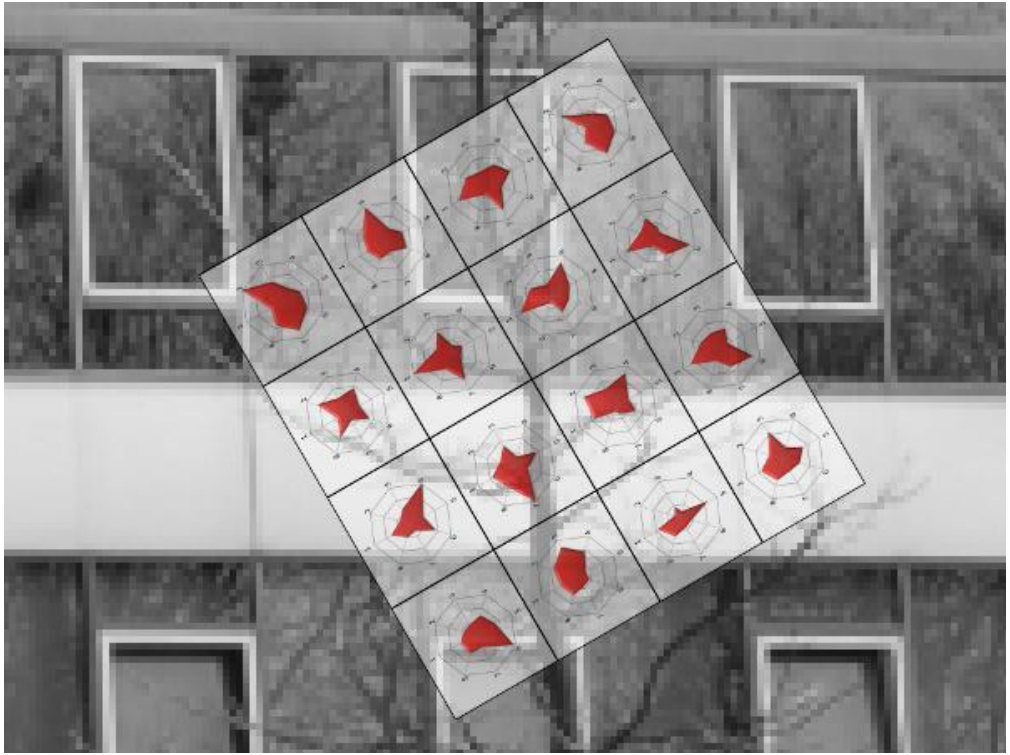
5. Detect gradients, normalize orientation



Gradient histogram: the change of brightness in that direction

For the highest value and each value within 80% of it an accordingly oriented descriptor is created!

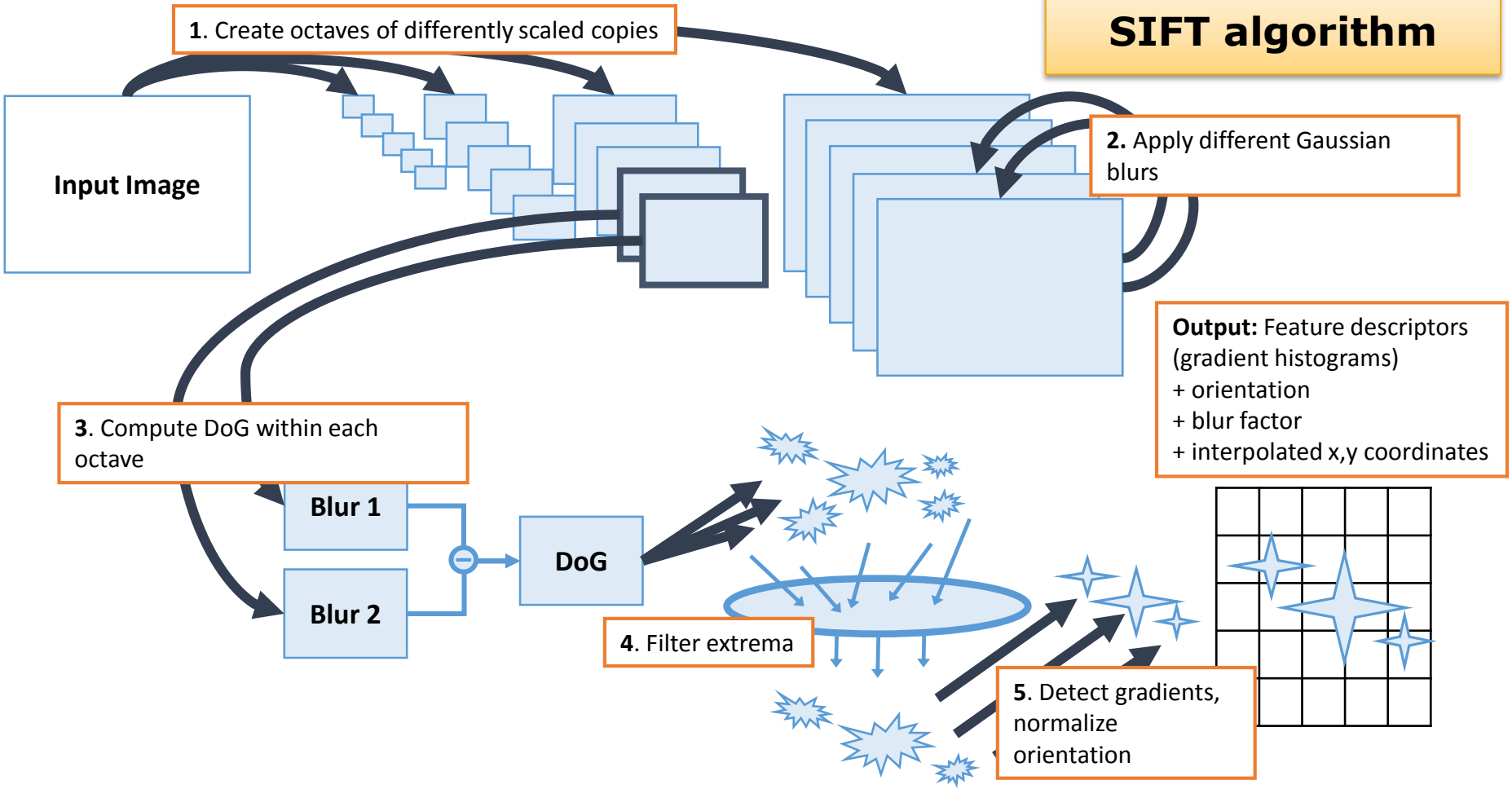
5. Detect gradients, normalize orientation



Visualization of a single descriptor

A descriptor comprises 4x4 gradient histograms describing the relative change of brightness in the area of a feature.

SIFT algorithm



Hasso Plattner Institut

Scalable SIFT for NUMA with Actors

Frank Feinbube,
Research Assistant

What did we contribute?

Implementation in Scala

Scala

- Productivity-focused high-level programming language
- Designed to allow for a high degree of parallelization and scalability

- Extensive actor library (**Akka**)
- Effortless distribution across multiple nodes
- Runs on JVM

Faster than OpenCV (C/C++)

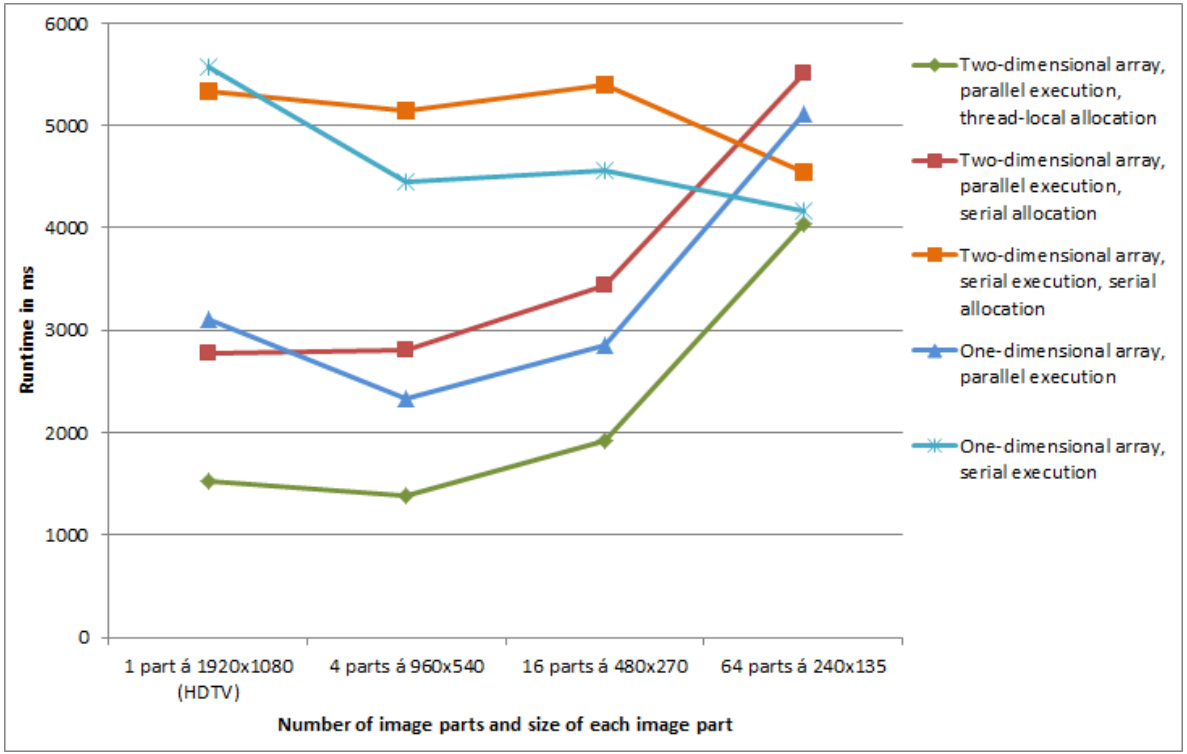
- SIFT in OpenCV is optimized C/C++, but still sequential
- We benchmarked our Scala-based implementation in sequential mode

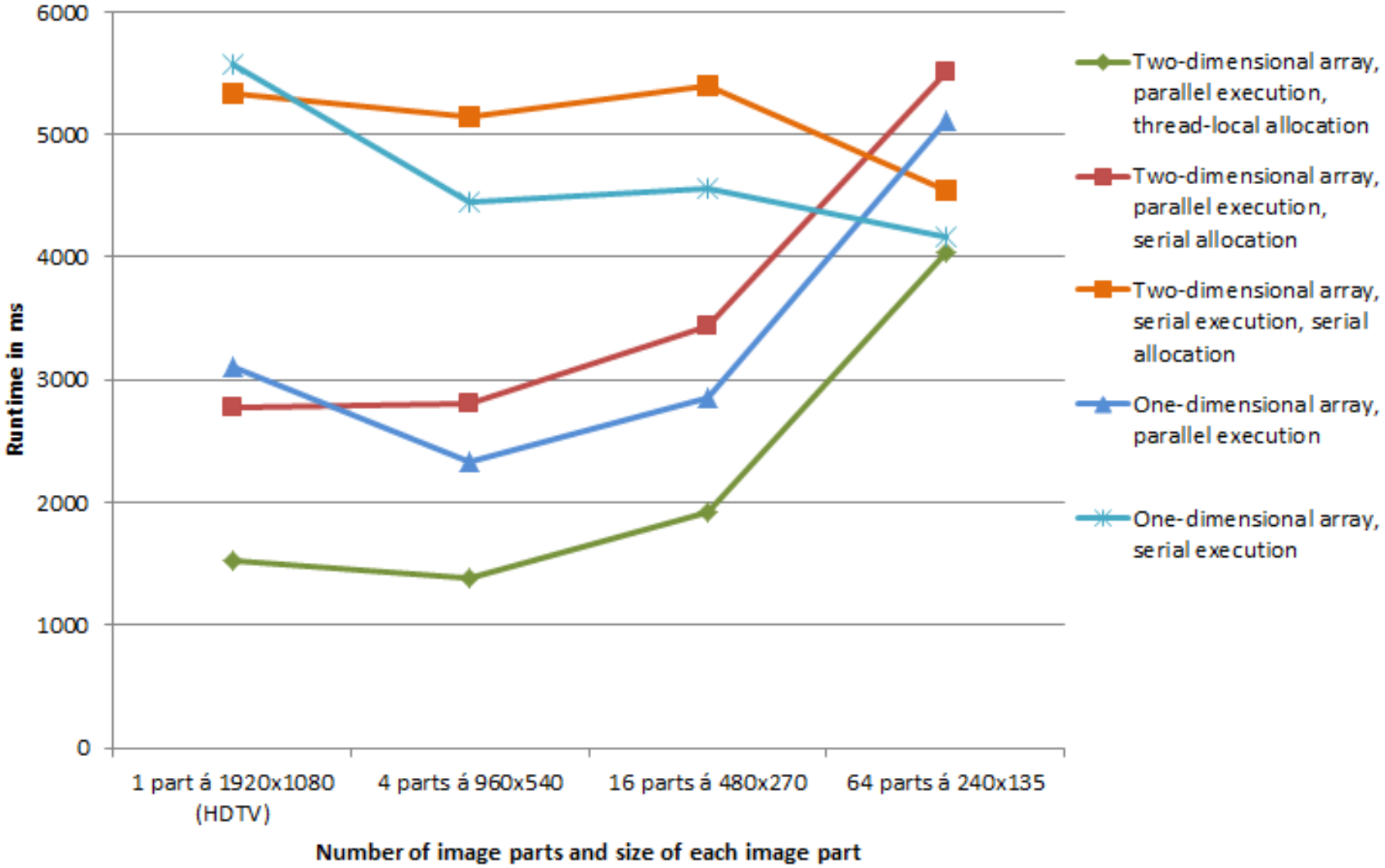
	OpenCV (C/C++)		**Our Scala implementation**	
	Runtime	Features	Runtime	Features
1920x1080	7670 ms	9460	5960 ms	9697
800x600	1330 ms	1316	1130 ms	1552

1.29 times faster for 1920x1080

1.18 times faster for 800x600

Data structure optimization – 2D array allocation



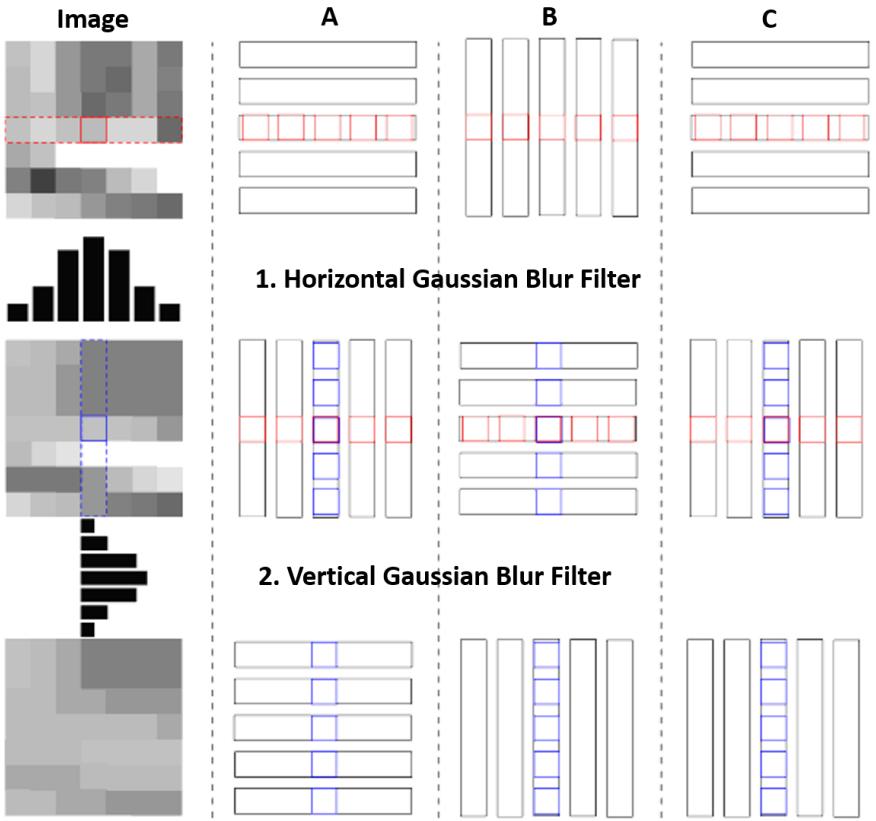

6000
5000
4000
3000
2000
1000
0
Runtime in ms
1 part á 1920x1080 (HDTV)
4 parts á 960x540
16 parts á 480x270
64 parts á 240x135
Number of image parts and size of each image part
Two-dimensional array, parallel execution, thread-local allocation
Two-dimensional array, parallel execution, serial allocation
Two-dimensional array, serial execution, serial allocation
One-dimensional array, parallel execution
One-dimensional array, serial execution

Optimization of the order of SIFT stages: L3 cache

- Different strategies for various combinations of image size and cache size
  - If less than 3 images fit into cache: order doesn’t matter
  - 3 images: all blurs one after another -> all substracts -> …
  - 4-6 images: all blurs one after another -> all substracts one after another (in backwards order)
  - >6 images: blur single image -> substract -> blur next -> …
  - 16 images: order doesn’t matter

- Has to be considered for each octave, since images are smaller for higher octaves

Algorithmic optimization – image flipping



Our experiment show that A and C perform similarly when executed serially, but B is 67% slower.

With six threads, B is still 35% slower than A, while C is 16% faster.

Optimization of the order of SIFT stages

- L2 cache: execute right after one another in one processing step
  - Extrema detection and interpolation
  - Computation of the orientation and the descriptor
- Smaller images = more blurred
  - Have a smaller amount of extrema -> less descriptors to compute
    - Probably even less than cores available
  - Collect extrema for all octaves first

Work distribution on NUMA nodes

- JVM
  - No control over NUMA environment (thread affinity)
    - Uncontrollable memory access latencies
  - Runtime and object management centralized in JVM instance

- We start one JVM per NUMA node

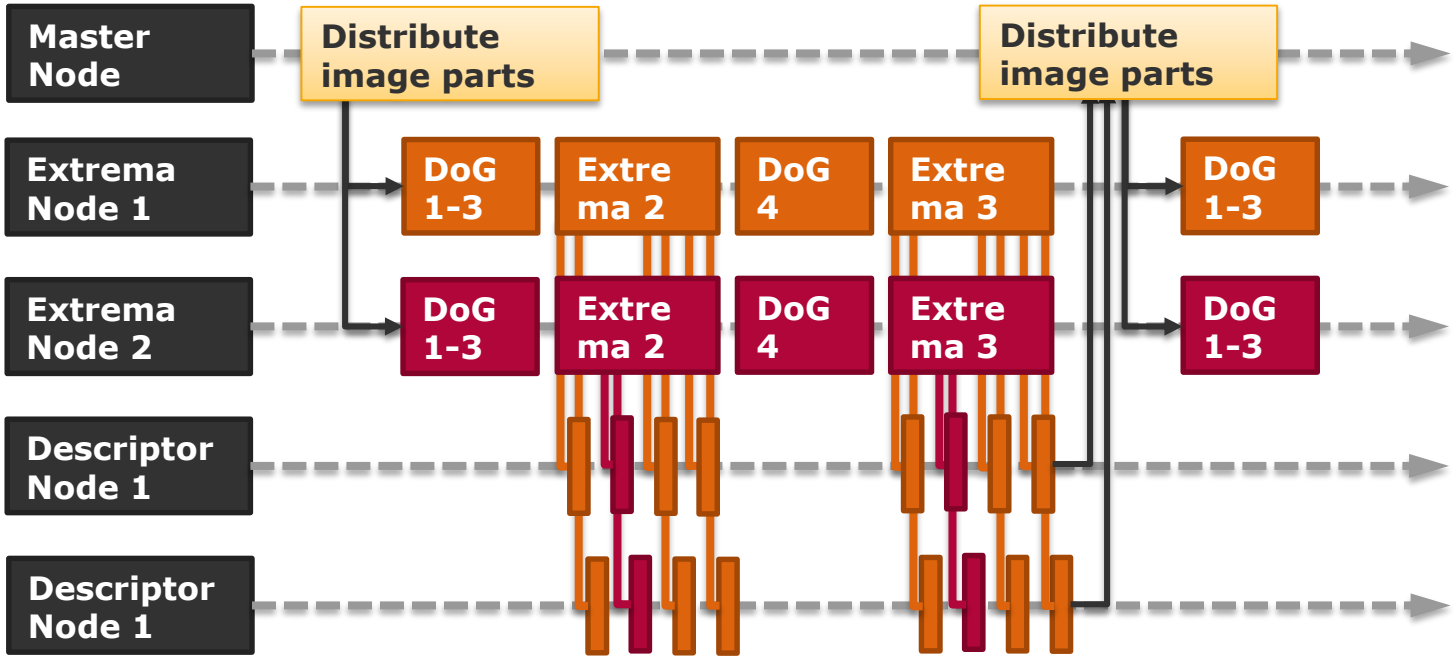
- Performance improvements:
  - 54% when using 2 JVMs instead of 1 on two NUMA nodes
  - 79% when using 4 JVMs instead of 1 on four NUMA nodes

Work distribution on NUMA nodes

- Actor model: distribution on multiple CPUs or multiple systems
  - One actor per JVM
    - Cache-aware and parallelized
  - Master actor decodes video stream and distributes frames
  - Work actors perform SIFT stages

- Video decoding is fast, disk access speed is the bottleneck for master

Work Distribution Strategy (3 types of actors)



Related work and our contribution

	Image Size	Features	Processors	Cores / Processor	Cores	Speedup	Speedup Factor
Feng et al. [3]	HDTV	1038	4	4	16	9.7	0.61
	720x576	700	4	4	16	11	0.69
	HDTV	1038	1	64	64	52	0.81
	720x576	700	1	64	64	39	0.61
Zhang et al. [13]	640x480	200-1000	2	4	8	5.9-6.7	0.73-0.84
Warn et al. [11]	4136x1424	40000	2	4	8	2-3	0.25-0.38
Our	854x480	1400-2700	1	6	6	7.15	**1.19**
Approach	854x480	1400-2700	2	6	12	13.21	**1.10**
(150 video frames)	854x480	1400-2700	4	6	24	24.23	**1.01**

Related Work

- Feng et al. [3]: OpenMP, performance optimizations, 4x4 HP DL580 G5
  - SIMD optimizations can halve the runtime
  - Thread affinity, false sharing removal and synchronization reduction yields a 25% performance improvement
  - Speedup factors of 9.7 for large pictures and 11 for small pictures
  - Scalability investigated with CMP simulator, 64 cores, shared L2
  - Speedup of 52 for large pictures and 39 for small pictures
- Zhang et al. [13]: OpenMP, 2x4 HP DL380 G5
  - Speedup of 5.9-6.7 depending on the feature density in the images
  - For 640x480 images speedup factor is slightly higher than that of Feng et al.'s implementation.

Related Work

- Warn et al. [11]: OpenMP, parallelization of the most expensive loops
  - Works best with large satellite pictures
  - Speedup of factor 2 on the 8-core test system.
- Several SIFT implementations for GPU accelerators [5, 9, 11, 12]
  - Bottleneck: data copy / move overhead
  - Warn et al. [11]: execute only Gaussian blurring on the GPU
    - Copying overhead = 90% of the execution time
    - Still, GPU version is 13 times faster than the CPU version
- Absolute performance over various hardware architectures is not well comparable

Related work and our contribution

	Image Size	Features	Processors	Cores / Processor	Cores	Speedup	Speedup Factor
Feng et al. [3]	HDTV	1038	4	4	16	9.7	0.61
	720x576	700	4	4	16	11	0.69
	HDTV	1038	1	64	64	52	0.81
	720x576	700	1	64	64	39	0.61
Zhang et al. [13]	640x480	200-1000	2	4	8	5.9-6.7	0.73-0.84
Warn et al. [11]	4136x1424	40000	2	4	8	2-3	0.25-0.38
Our	854x480	1400-2700	1	6	6	7.15	**1.19**
Approach	854x480	1400-2700	2	6	12	13.21	**1.10**
(150 video frames)	854x480	1400-2700	4	6	24	24.23	**1.01**

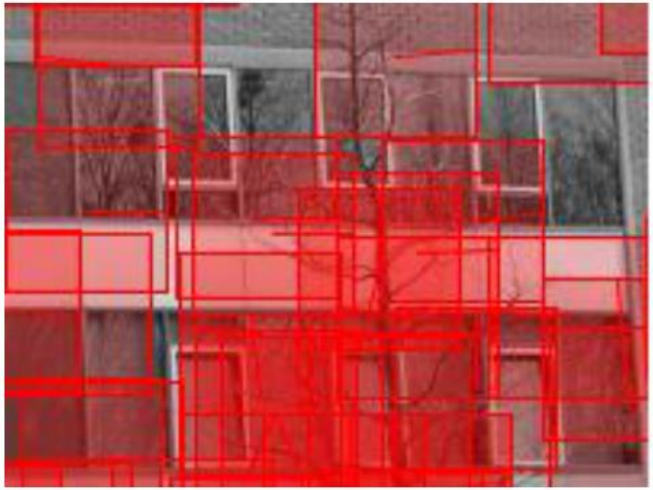
Distribution over multiple systems / into the cloud

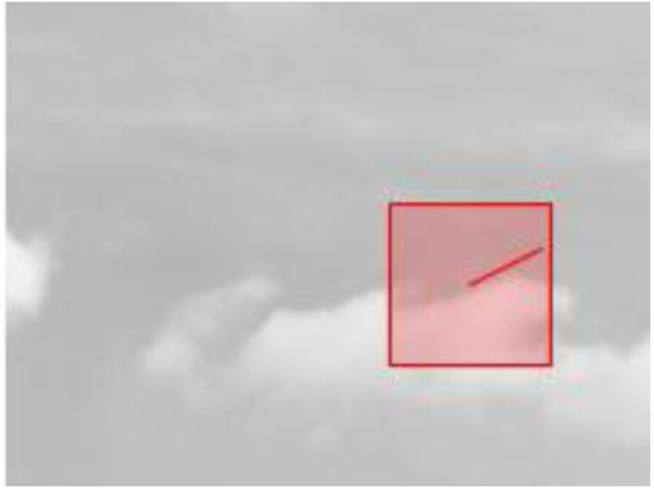
- No modifications needed due to application of actor programming model
  - Effortless reuse of such an implementation in various execution environments
- Communication / synchronization overhead
- Increased image throughput
- Widely known I/O optimizations for cluster computing to reduce latency
  - Fast interconnects, parallel file system, …

- When distributed across 5 machines, we achieved a **speedup of 3.74**

Imbalanced workload

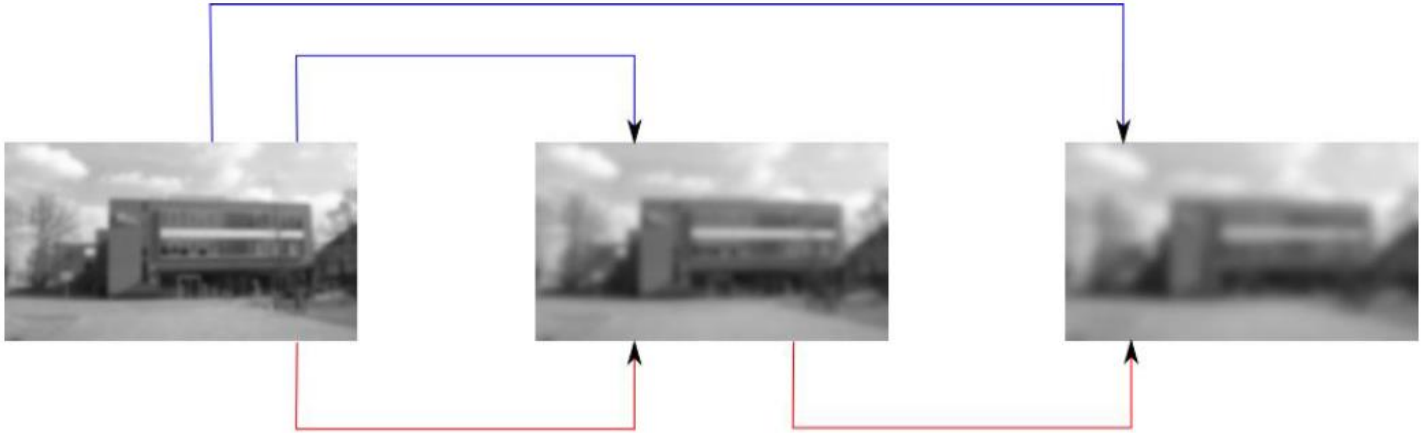
- Work per actor depends on feature density
  - We make sure that each work actor always has 2 images (enough to hide memory latency effects)





Iterative vs. original-based blurring

- Iterative blurring (red) has a smaller filter size (less computations) and is thus favored by many implementations (almost similar accuracy)
- Leads to huge increase in pixel transfer overhead (51% for 16 tiles)
- Use original-based blurring in distributed scenarios



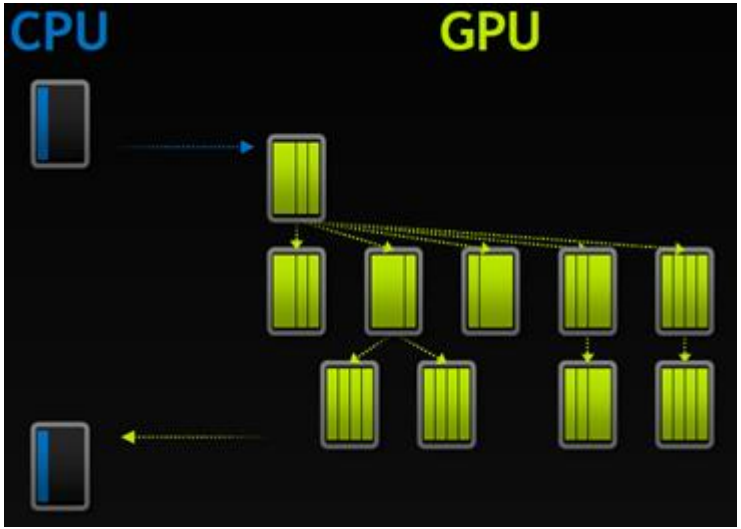
Single image distribution

- Example: shared belt of 45 pixels (ghost cells)
  - Left: one of 4 image tiles, 13% overhead
  - Right: one of 64 image tiles, 79% overhead

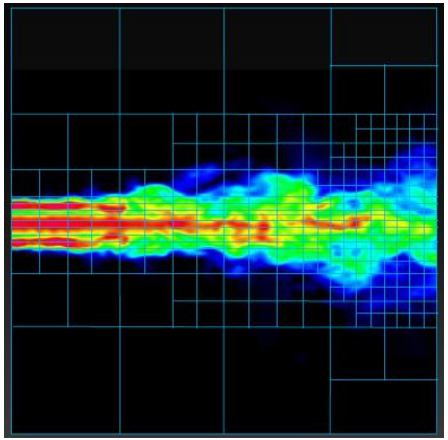
Where do we go from here?

Future Work: GPU / Accelerator Implementation

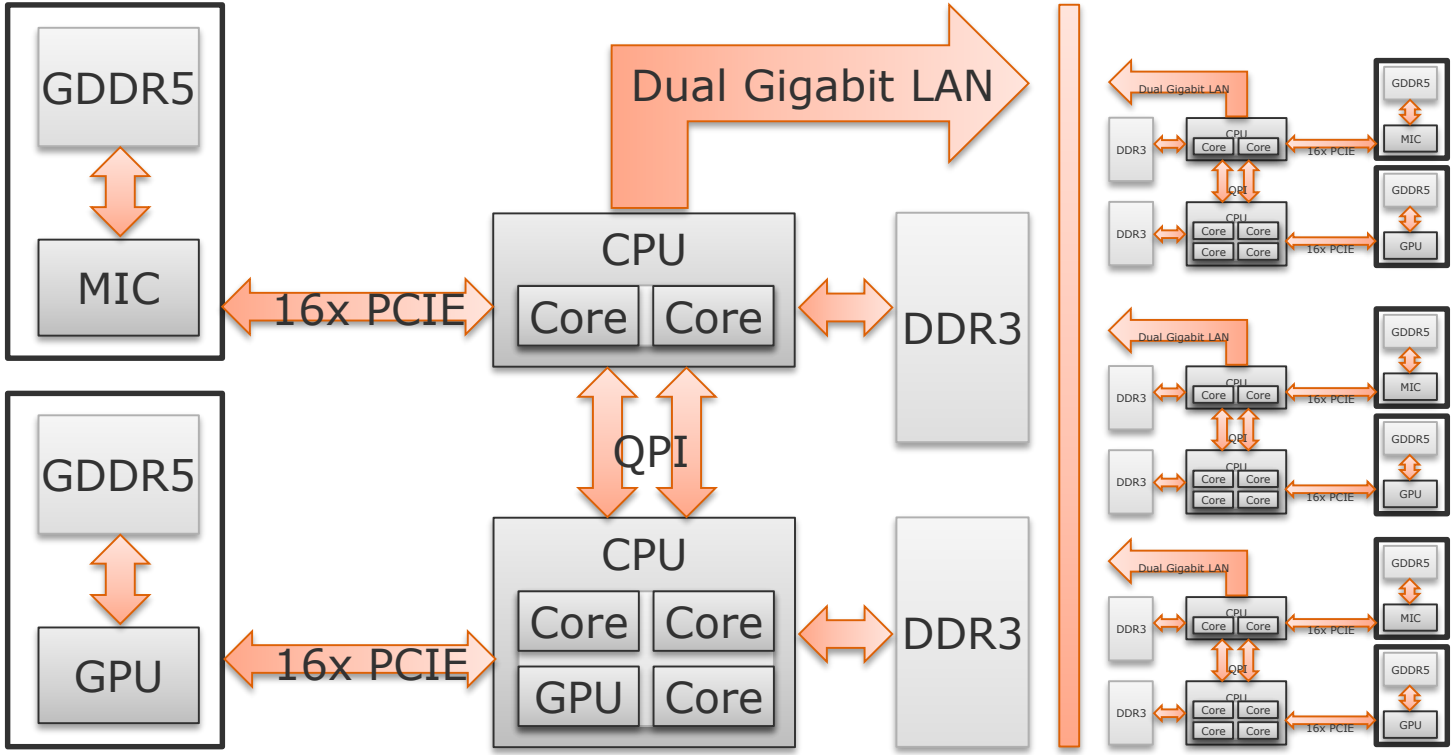
Dynamic Parallelism?



Xeon Phi?



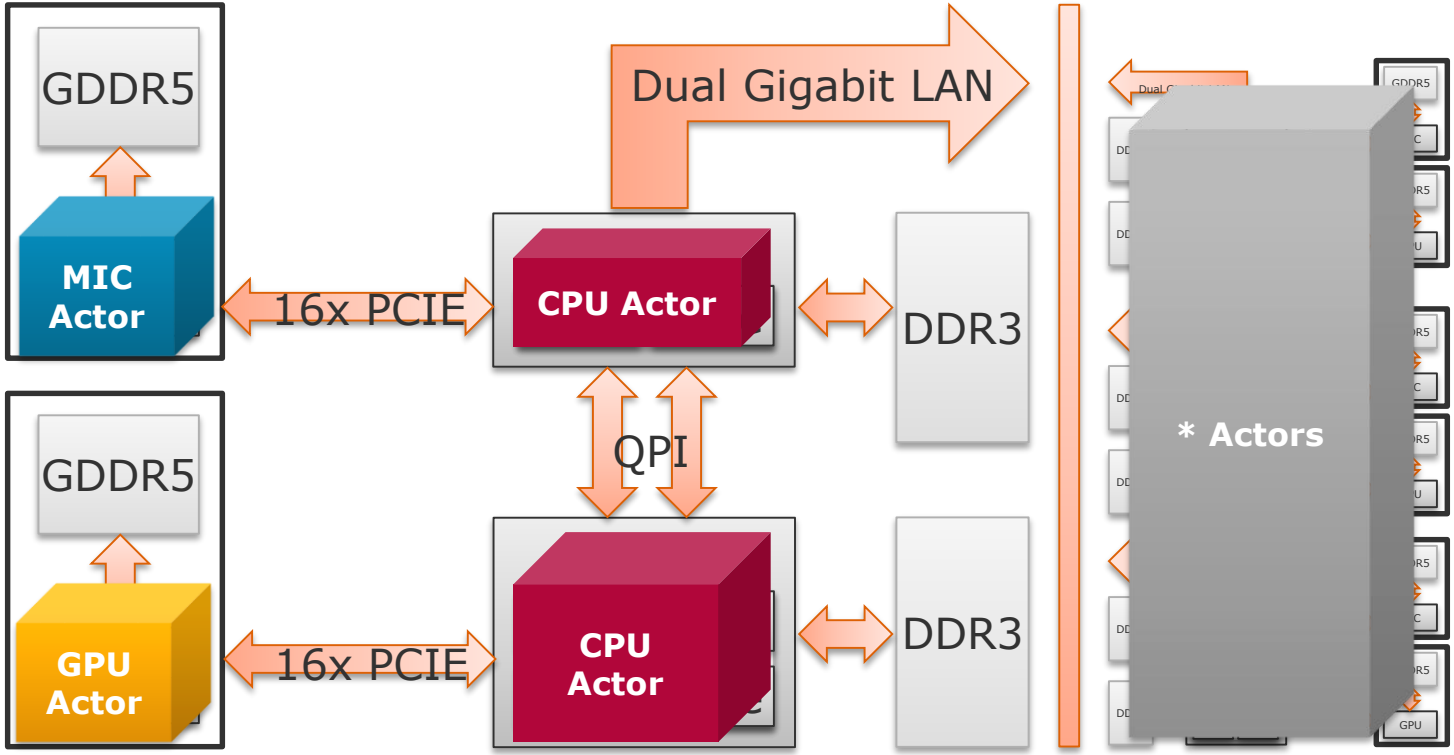
Future Work



Scalable SIFT for NUMA with Actors

Frank Feinbube,
Research Assistant

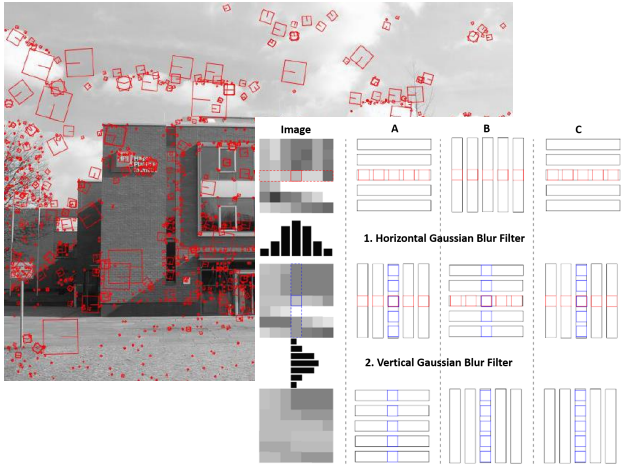
Future Work: Distributed, heterogeneous Actors



Scalable SIFT for NUMA with Actors

Frank Feinbube,
Research Assistant

Scalable SIFT for NUMA with Actors



- Data structure optimization: 2D array allocation
- Optimization of the order of SIFT stages and caching
- Algorithmic optimization: image flipping
- NUMA node work distribution strategies
- Multi-System work distribution
- …

Our	854x480	1400-2700	1	6	6	7.15	**1.19**
Approach	854x480	1400-2700	2	6	12	13.21	**1.10**
(150 video	854x480	1400-2700	4	6	24	24.23	**1.01**
frames)							

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Scalable SIFT for NUMA with Actors

Frank Feinbube, Lena Herscheid,
Christoph Neijenhuis, Peter Tröger

Hasso Plattner Institute for IT Systems Engineering