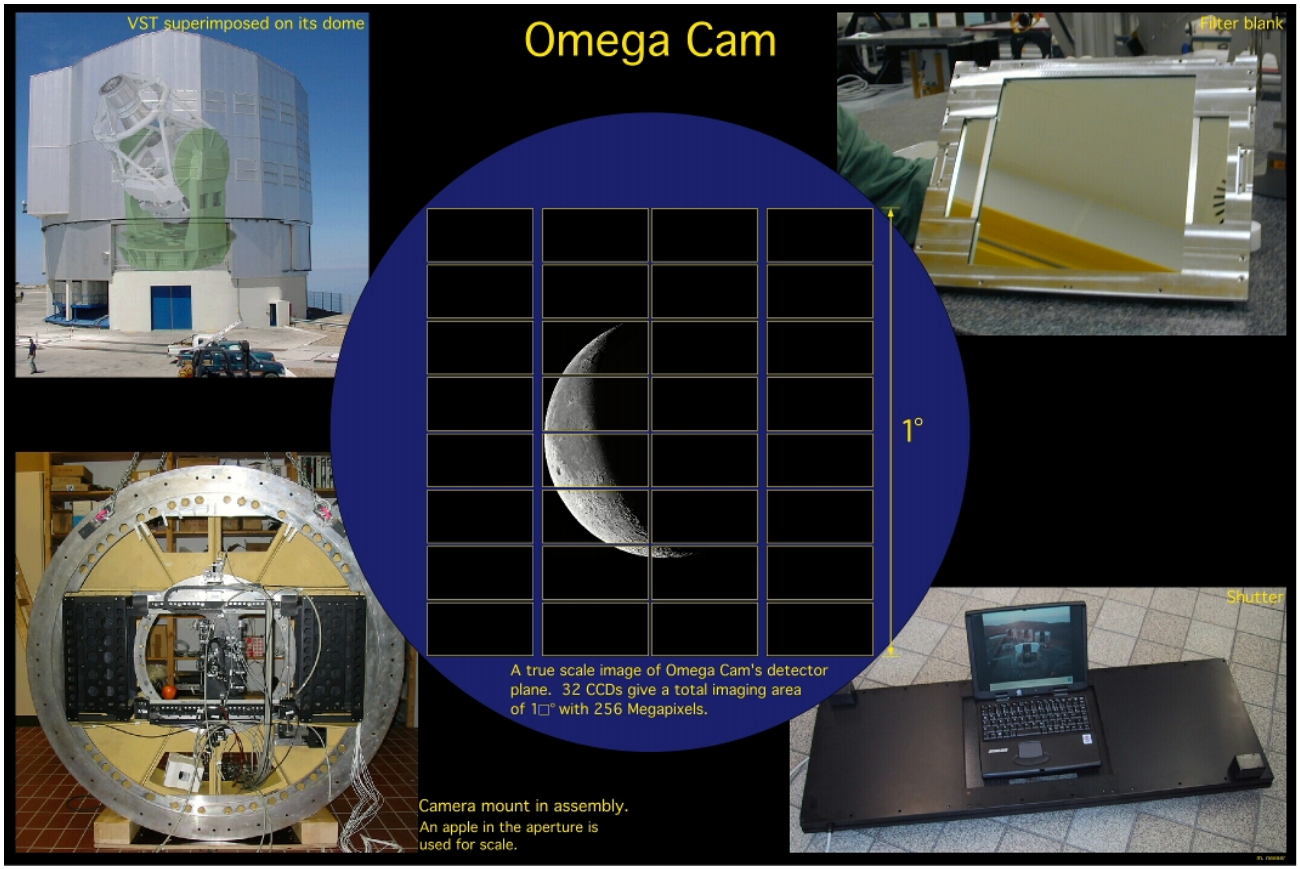
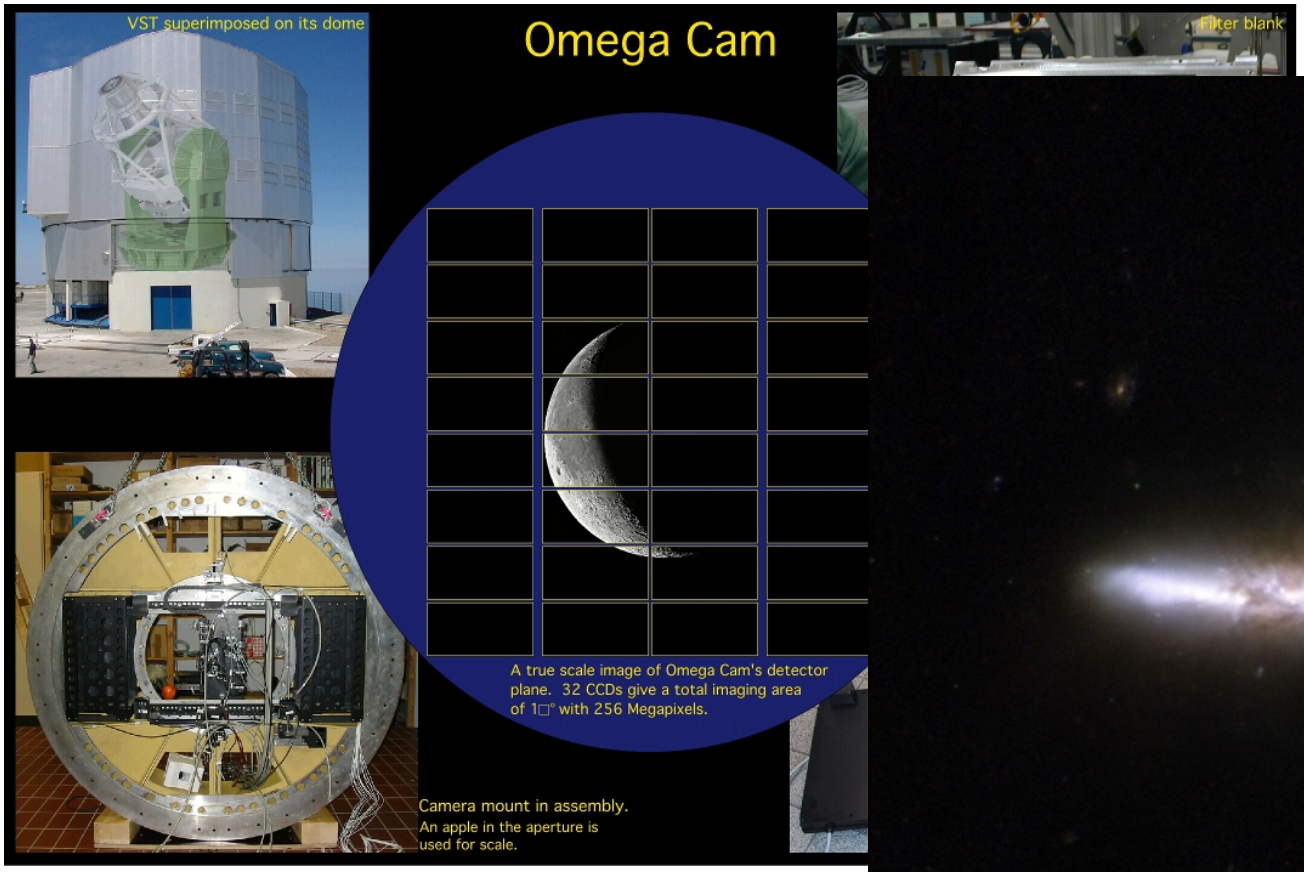
university of groningen

Supercomputing for Giga- and Tera-Pixel Image Scales

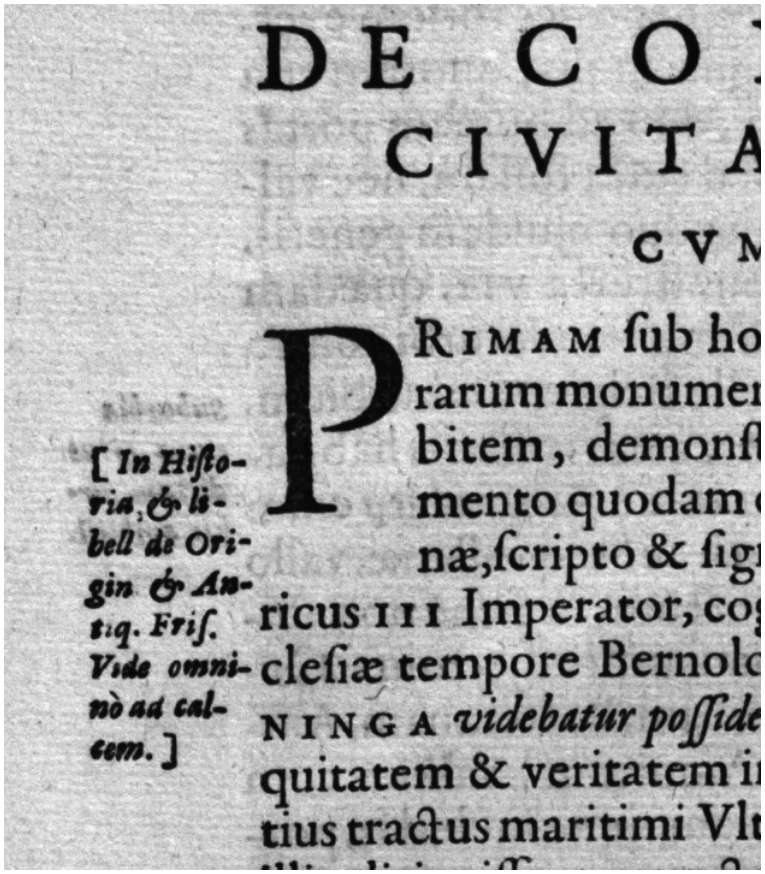
Michael H. F. Wilkinson

Johann Bernoulli Institute
University of Groningen


VST superimposed on its dome
Omega Cam
Filter blank
1°
A true scale image of Omega Cam's detector plane. 32 CCDs give a total imaging area of 1□° with 256 Megapixels.
Shutter
Camera mount in assembly.
An apple in the aperture is used for scale.


VST superimposed on its dome
Omega Cam
Filter blank
A true scale image of Omega Cam's detector plane. 32 CCDs give a total imaging area of 1□° with 256 Megapixels.
Camera mount in assembly.
An apple in the aperture is used for scale.

[Ouzounis & Wilkinson, IEEE Trans. PAMI 2011]



DE CO

CIVITA

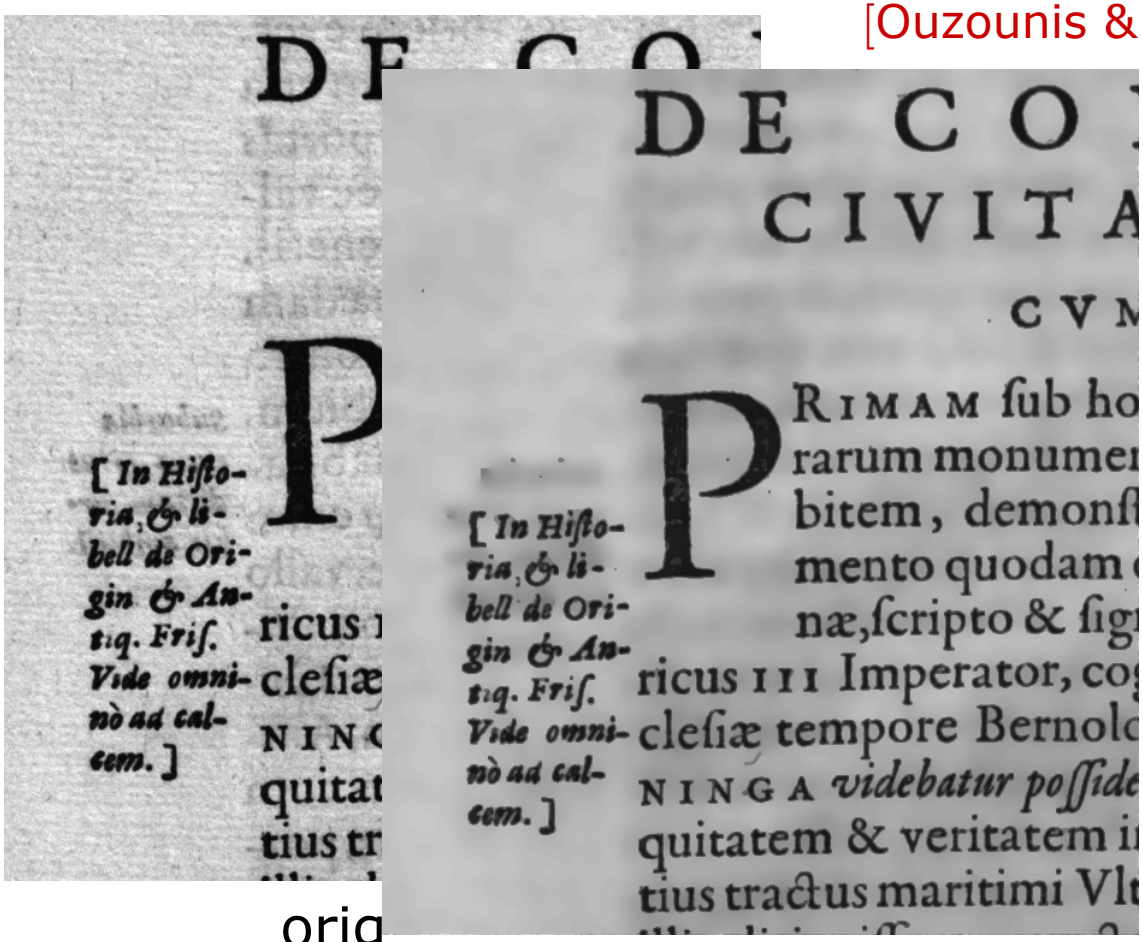
CVM

PRIMAM ſub ho
rarum monume
bitem, demonſ
mento quodam
næ, ſcripto & ſig
ricus III Imperator, co
cleſiæ tempore Bernol
NINGA *videbatur poſſide*
quitatem & veritatem i
tius tractus maritimi Vl

[*In Hiſtoria, & libell de Origin & Antiq. Friſ. Vide omninò ad calcem.*]

original

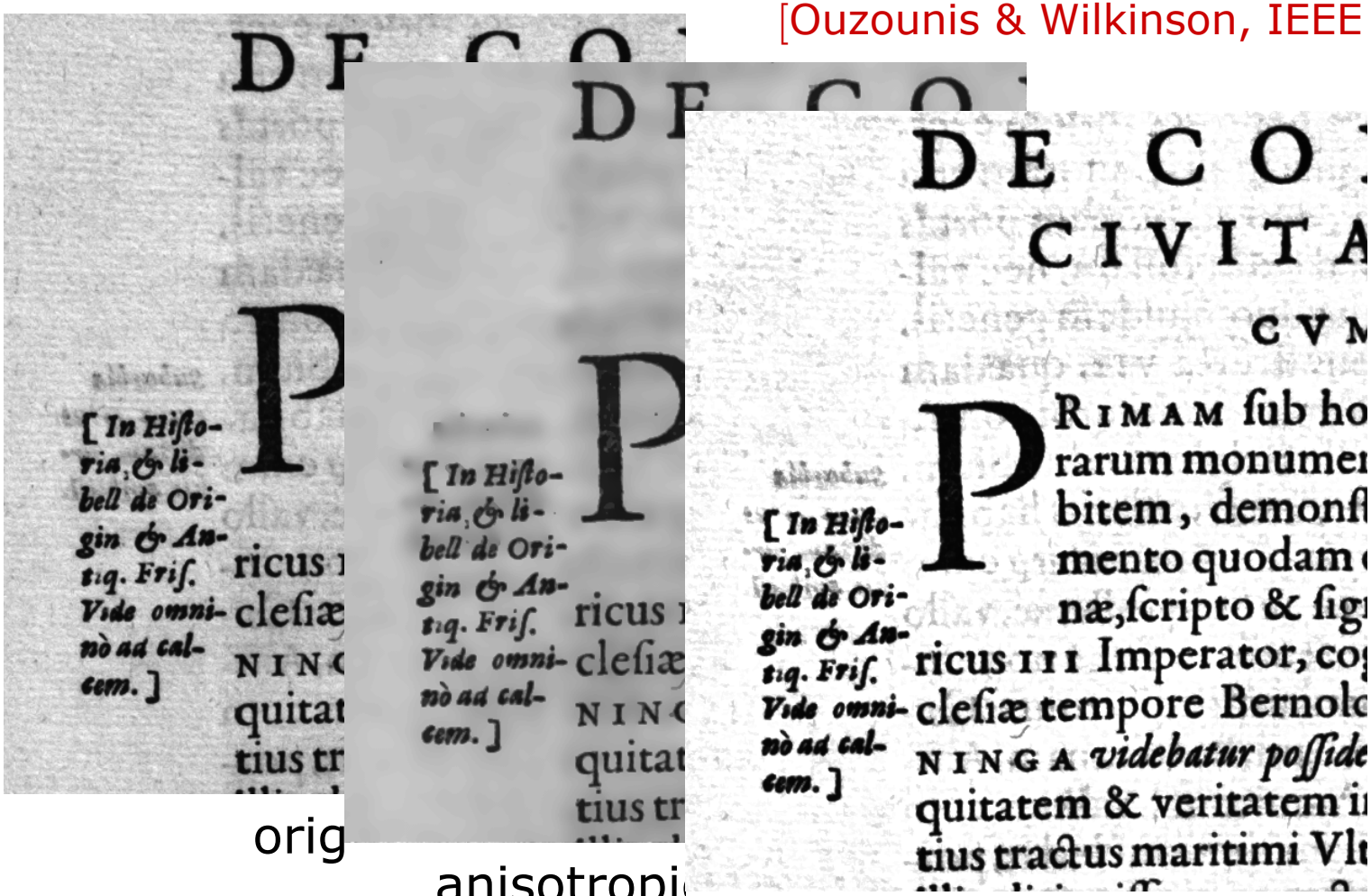
[Ouzounis & Wilkinson, IEEE Trans. PAMI 2011]



orig

anisotropic diffusion

[Ouzounis & Wilkinson, IEEE Trans. PAMI 2011]

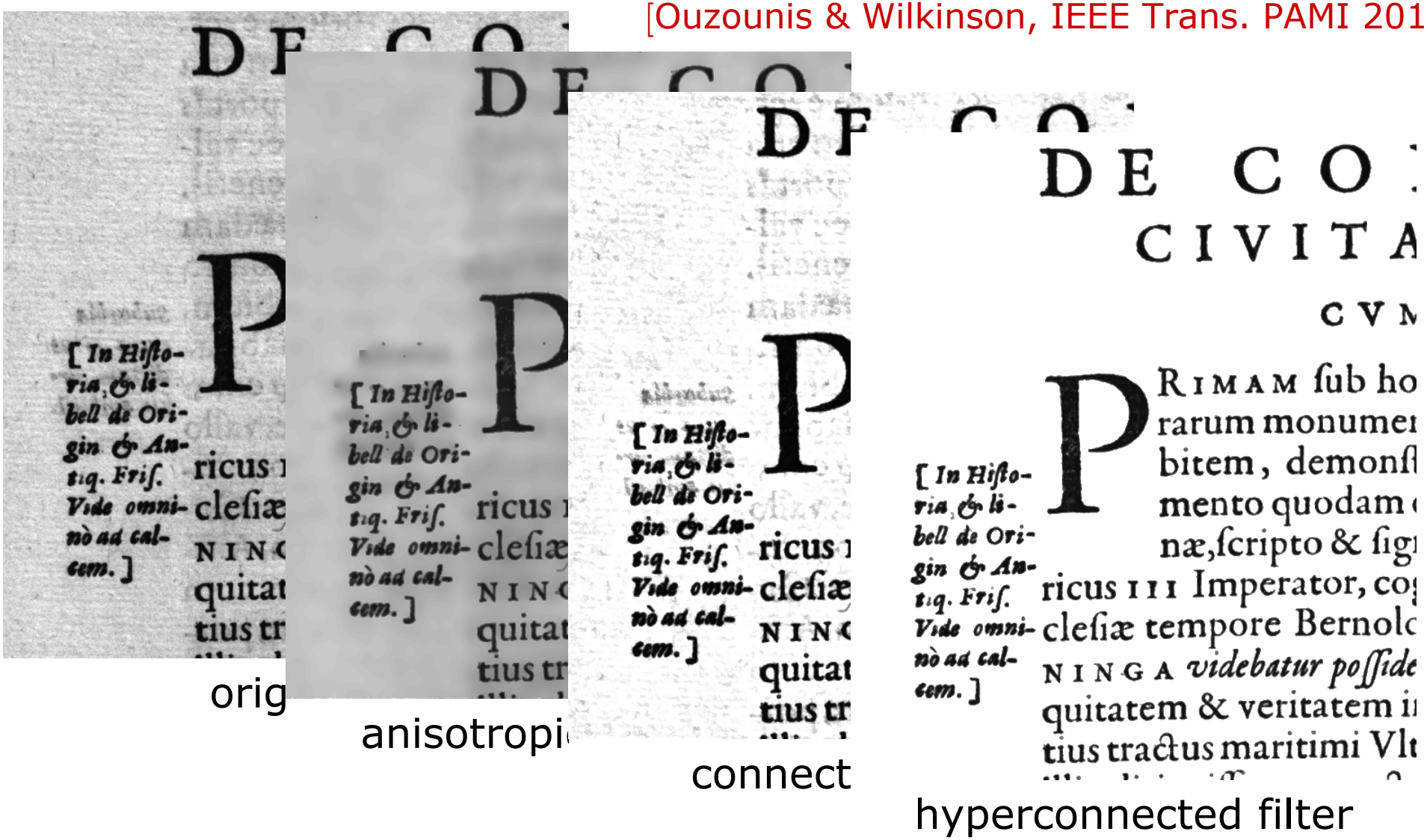


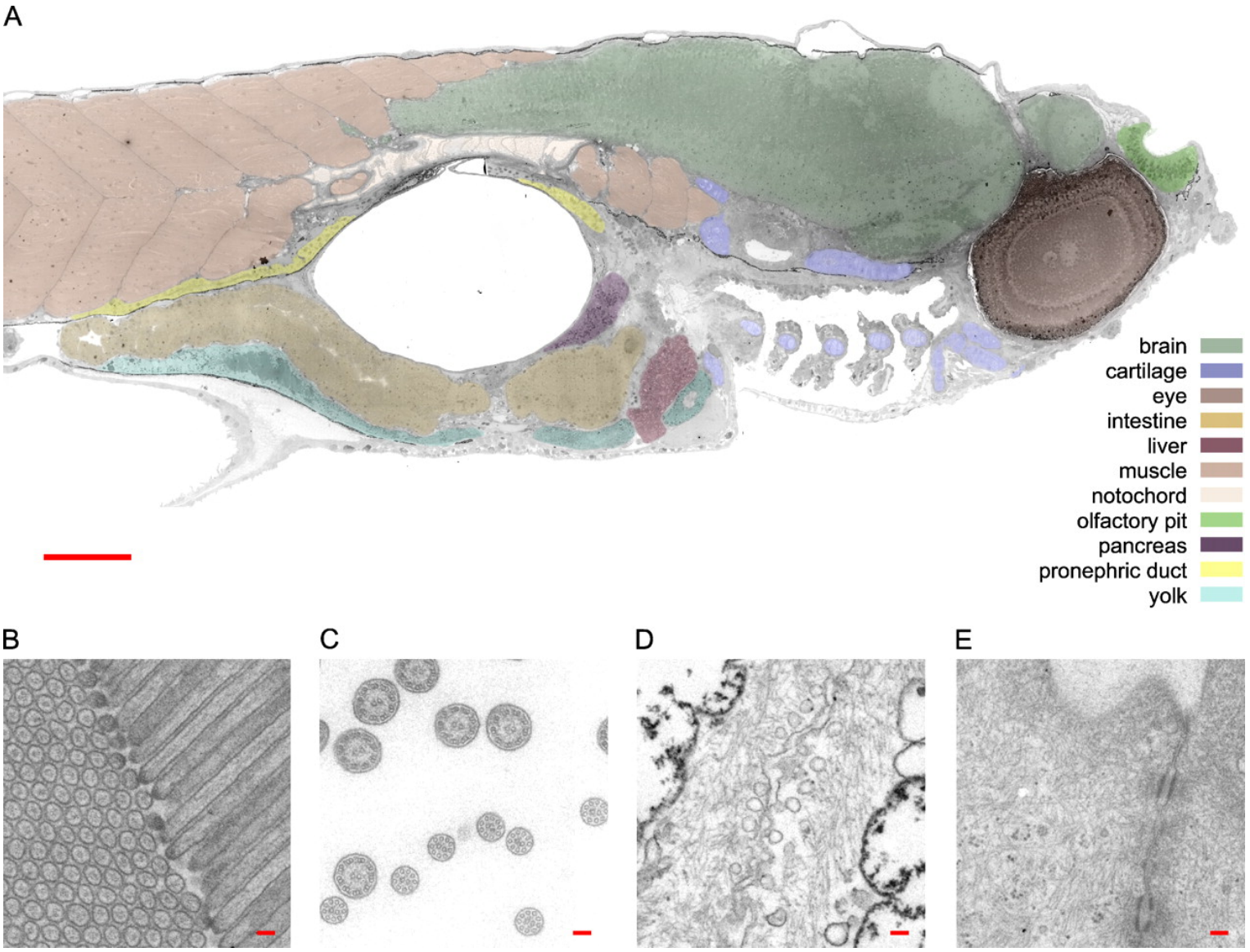
orig

anisotropi

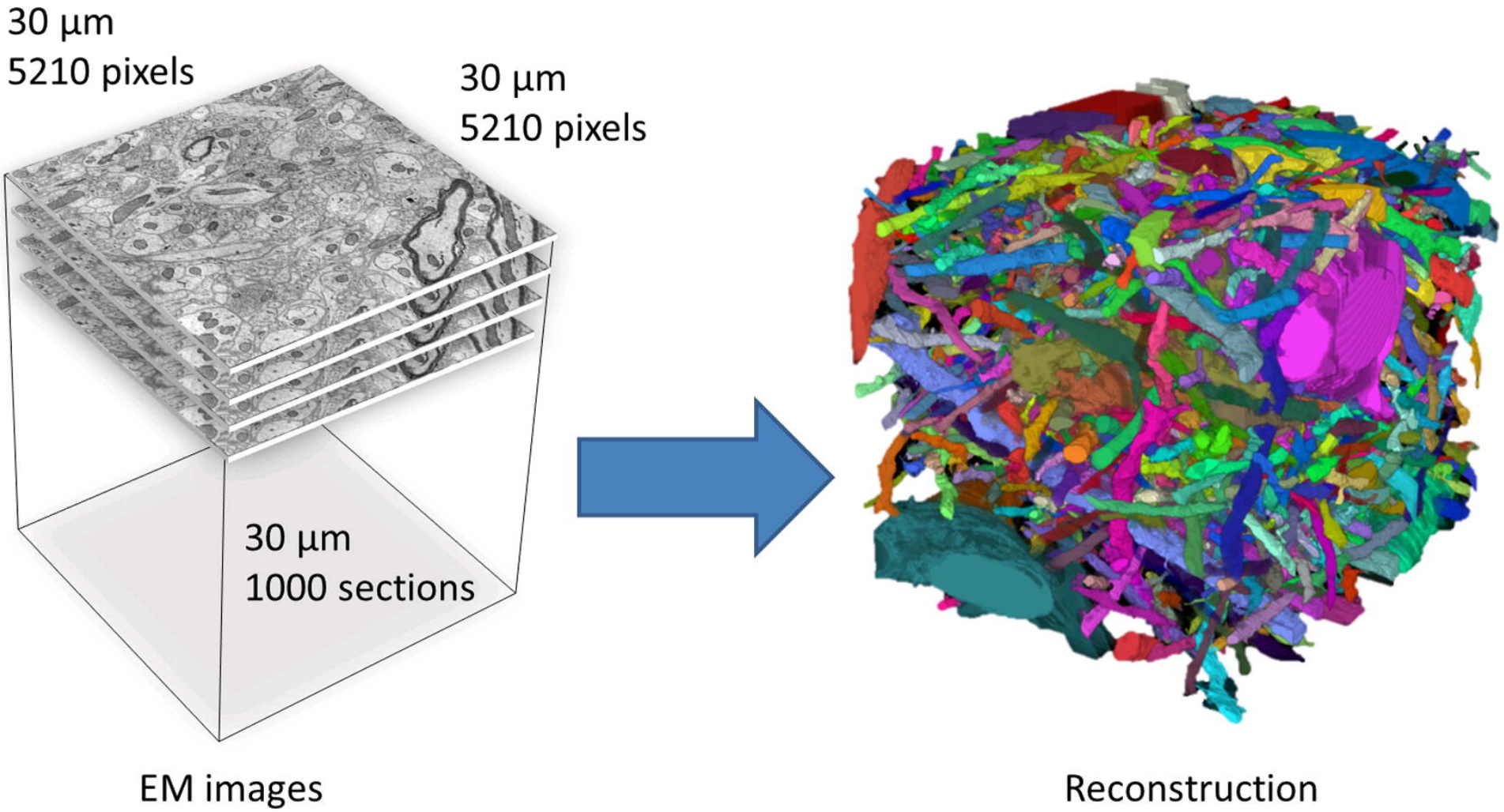
connected filter

[Ouzounis & Wilkinson, IEEE Trans. PAMI 2011]

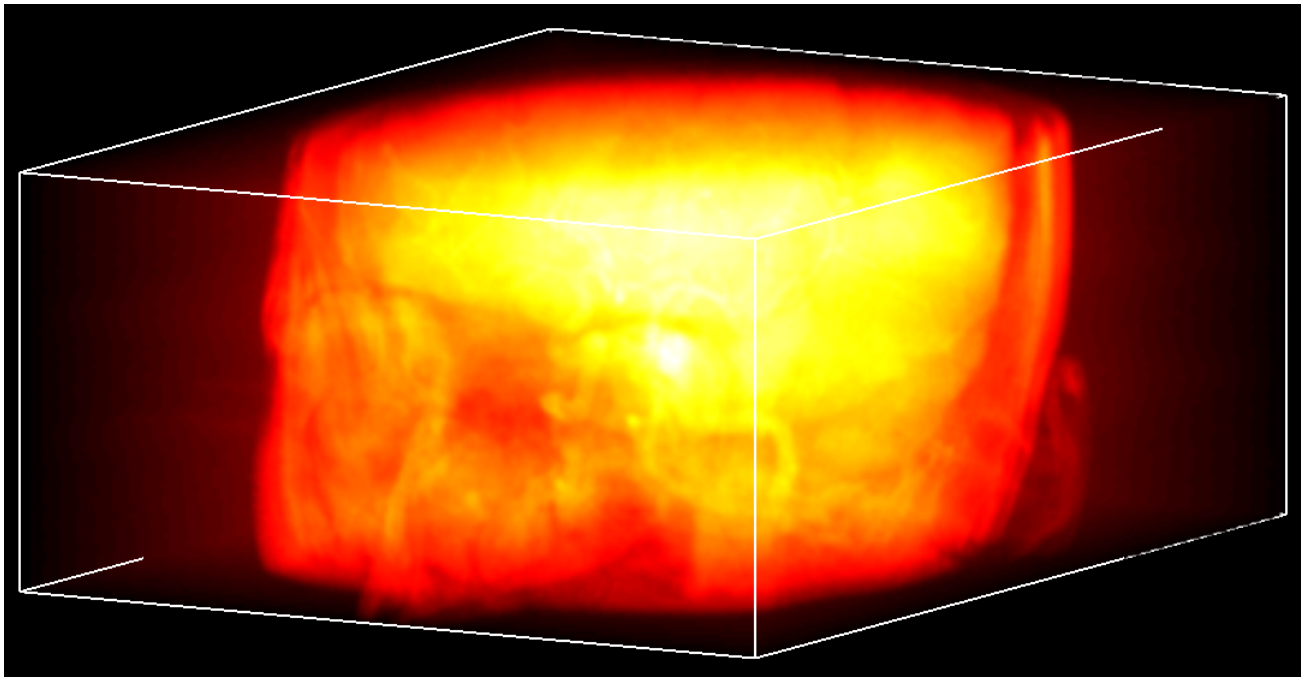




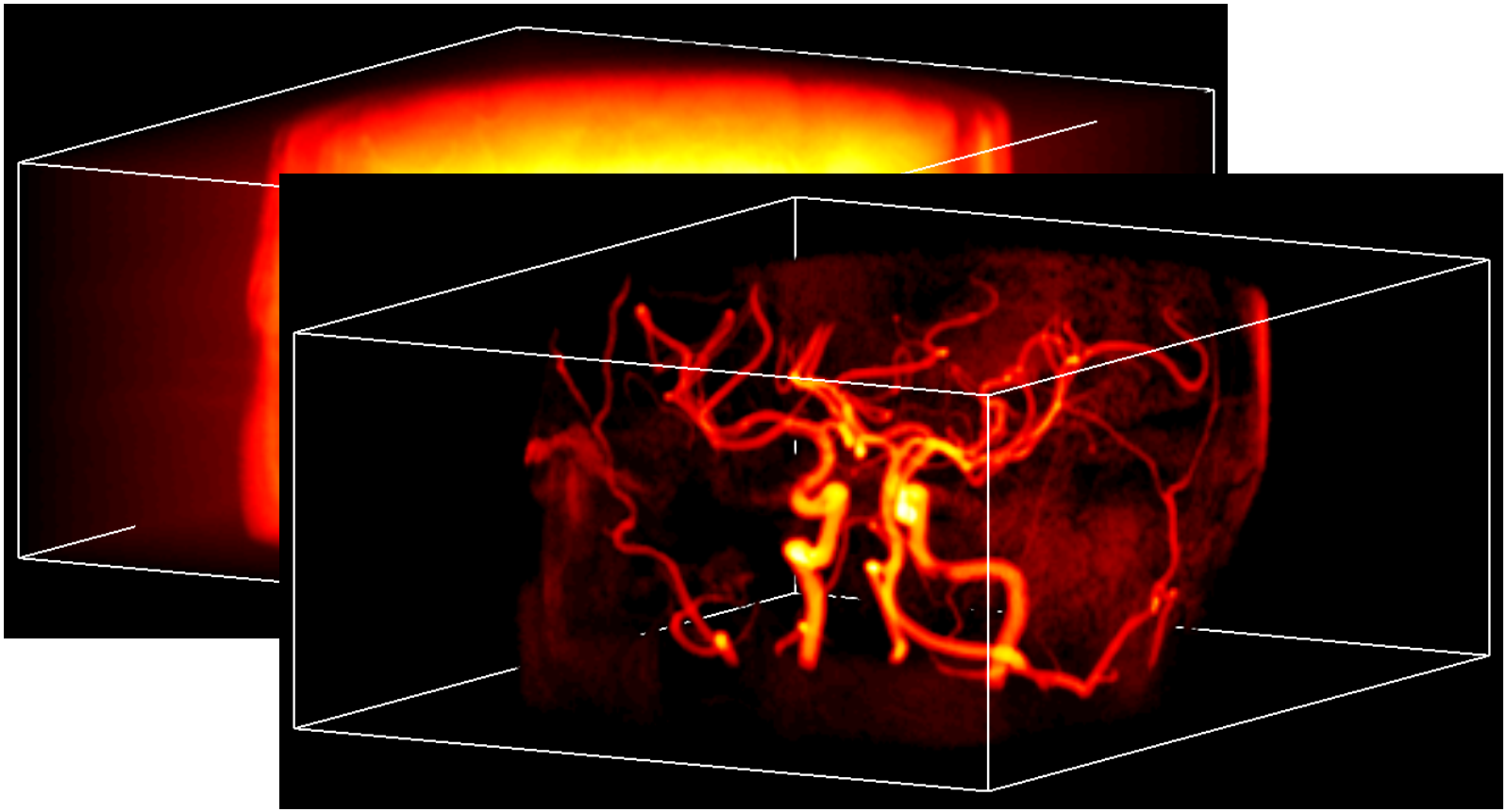
Faas et al (2012)



Kaynig et al (2015)

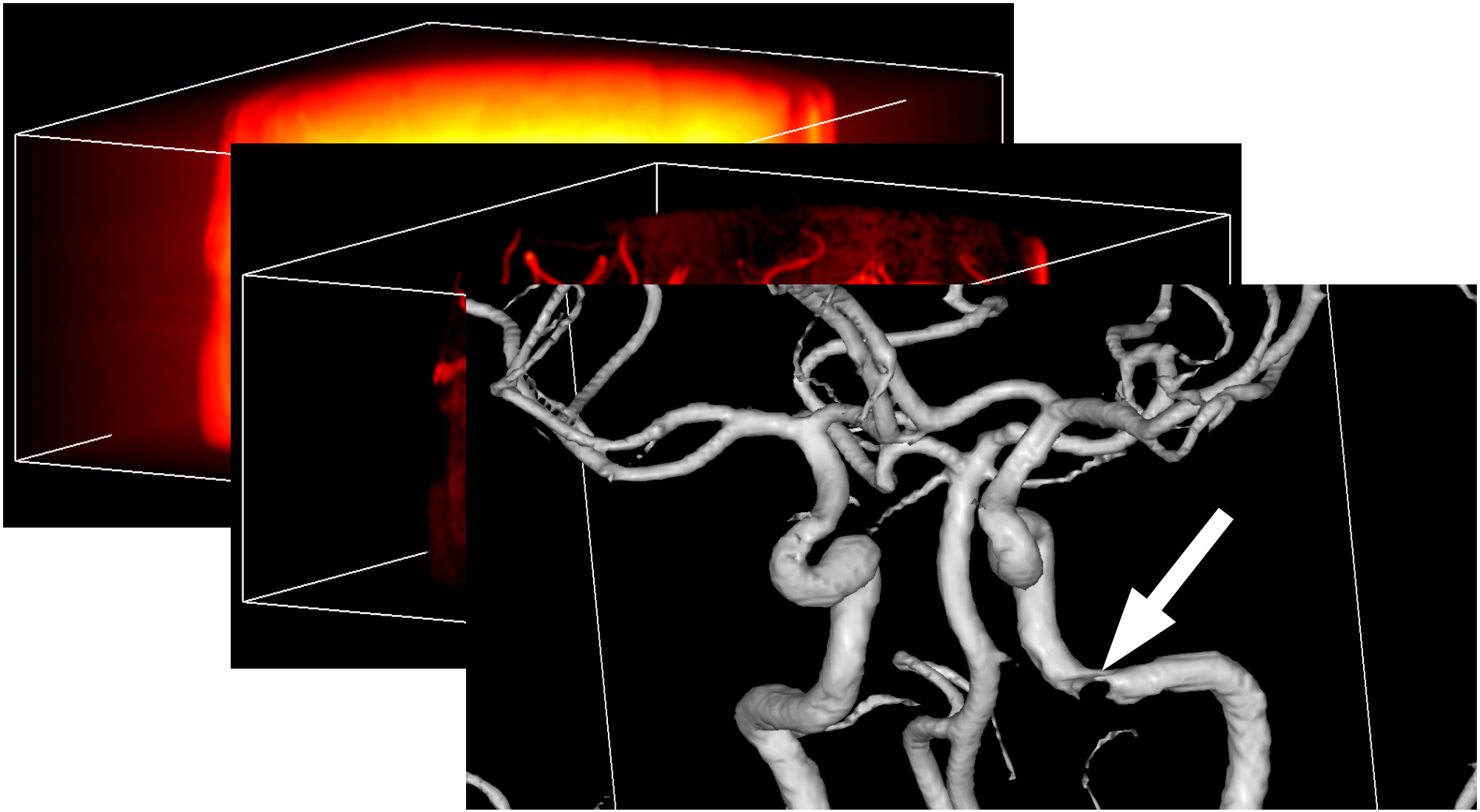


Raw data



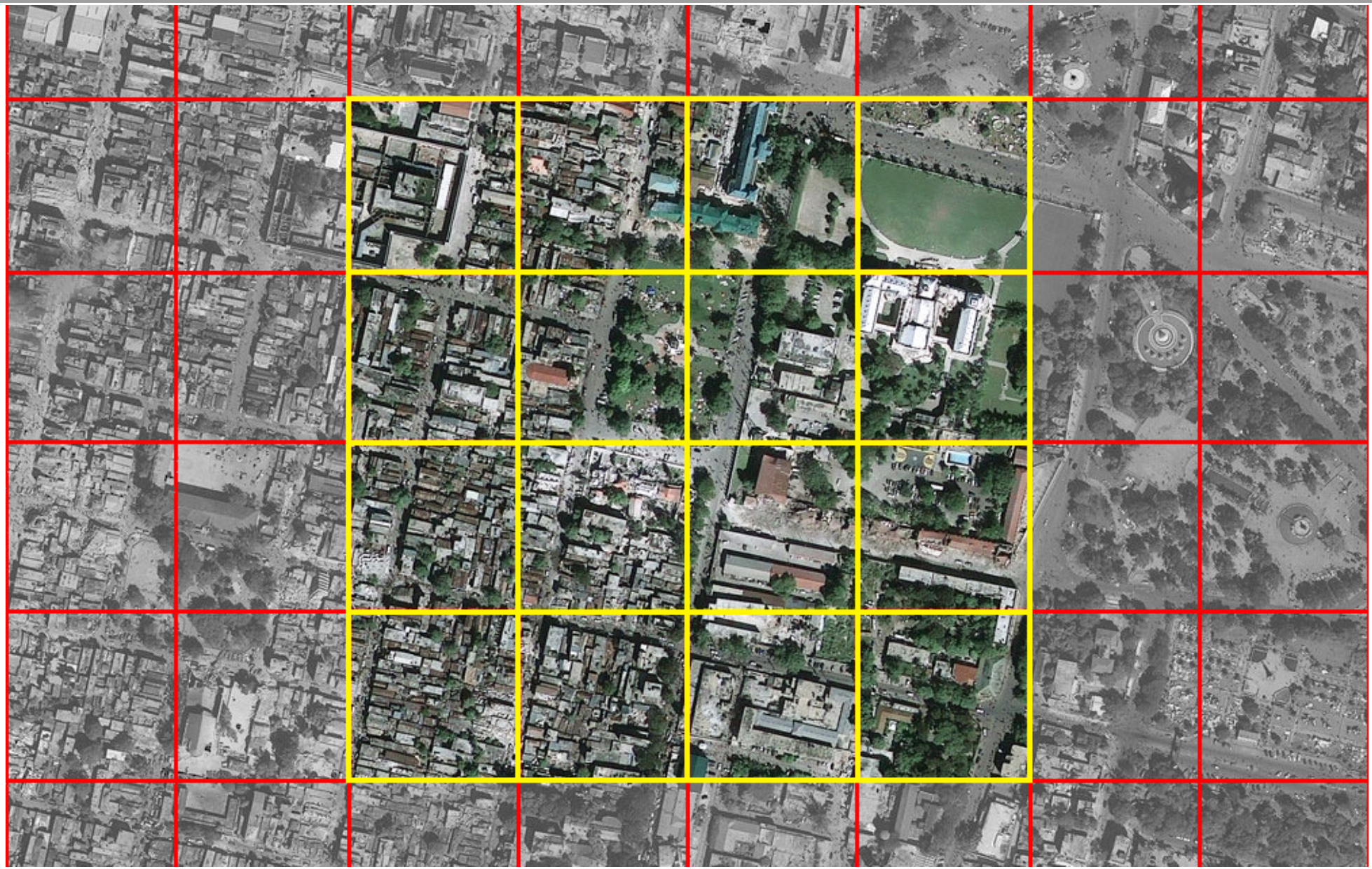
Connected filter

Angiography



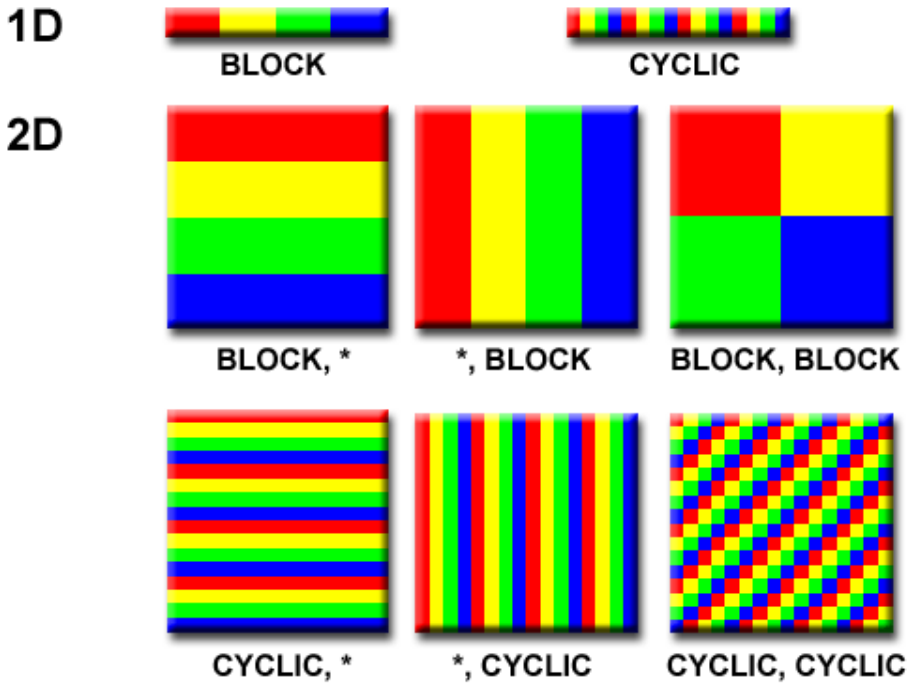
Blood clot?

Big, URGENT Data

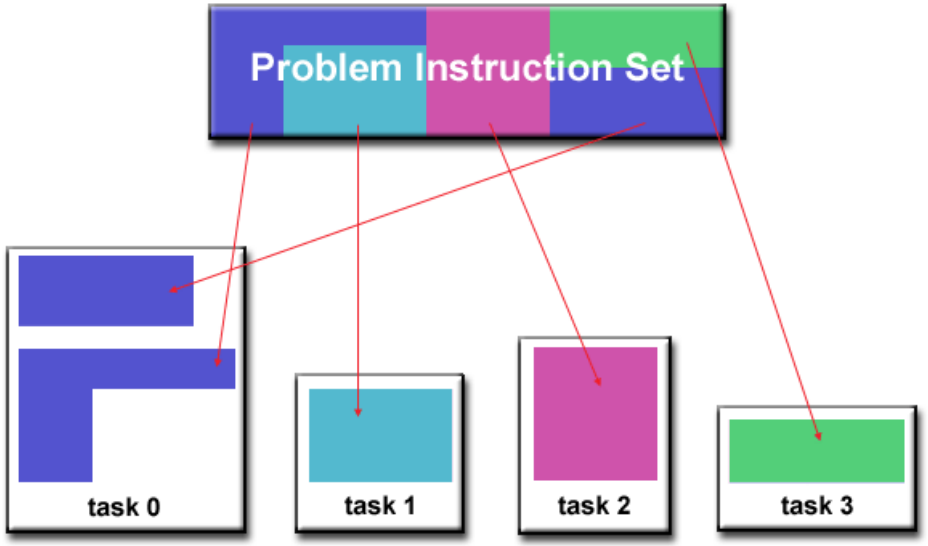


Enter (Super)computers

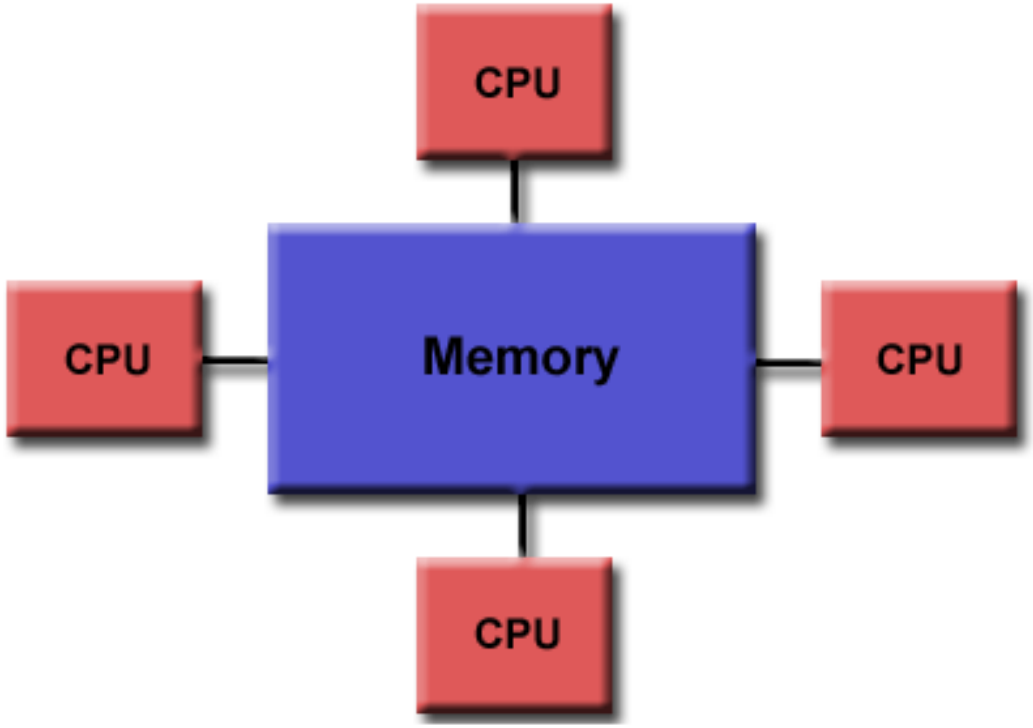
- Most computers are parallel machines
- Partitioning the task is essential



spatial

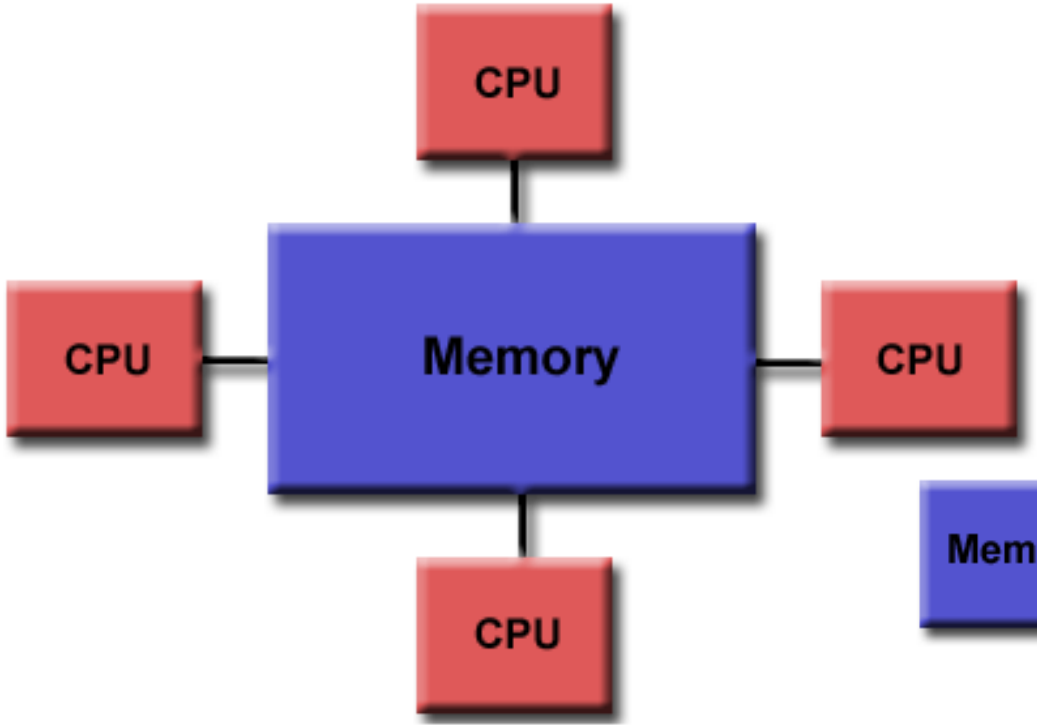


functional

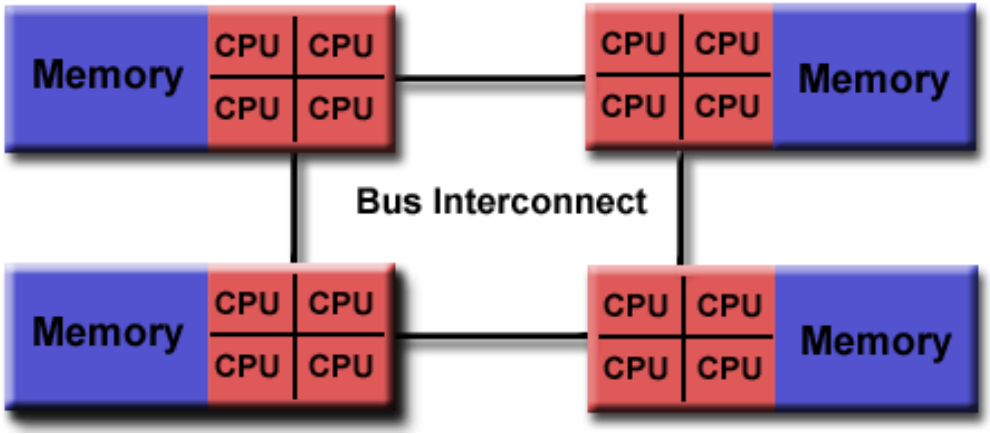


Simple shared memory

Shared Memory

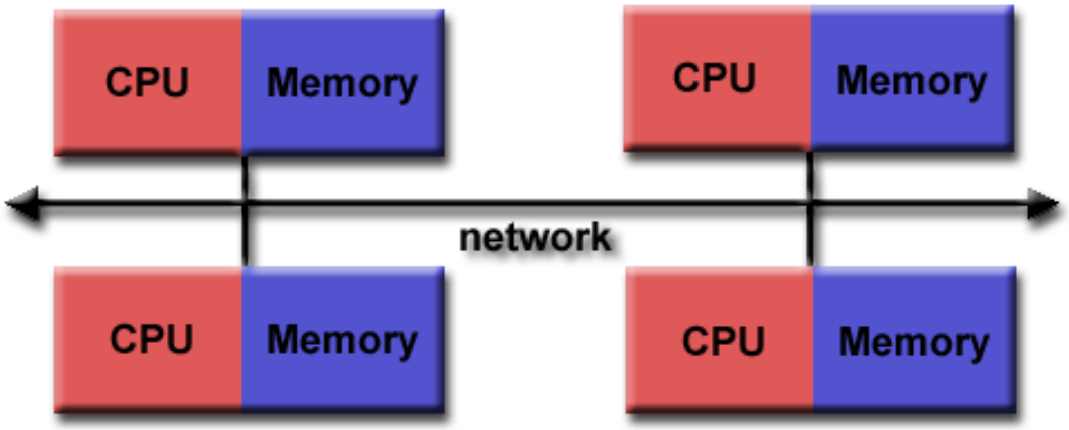


Simple shared memory



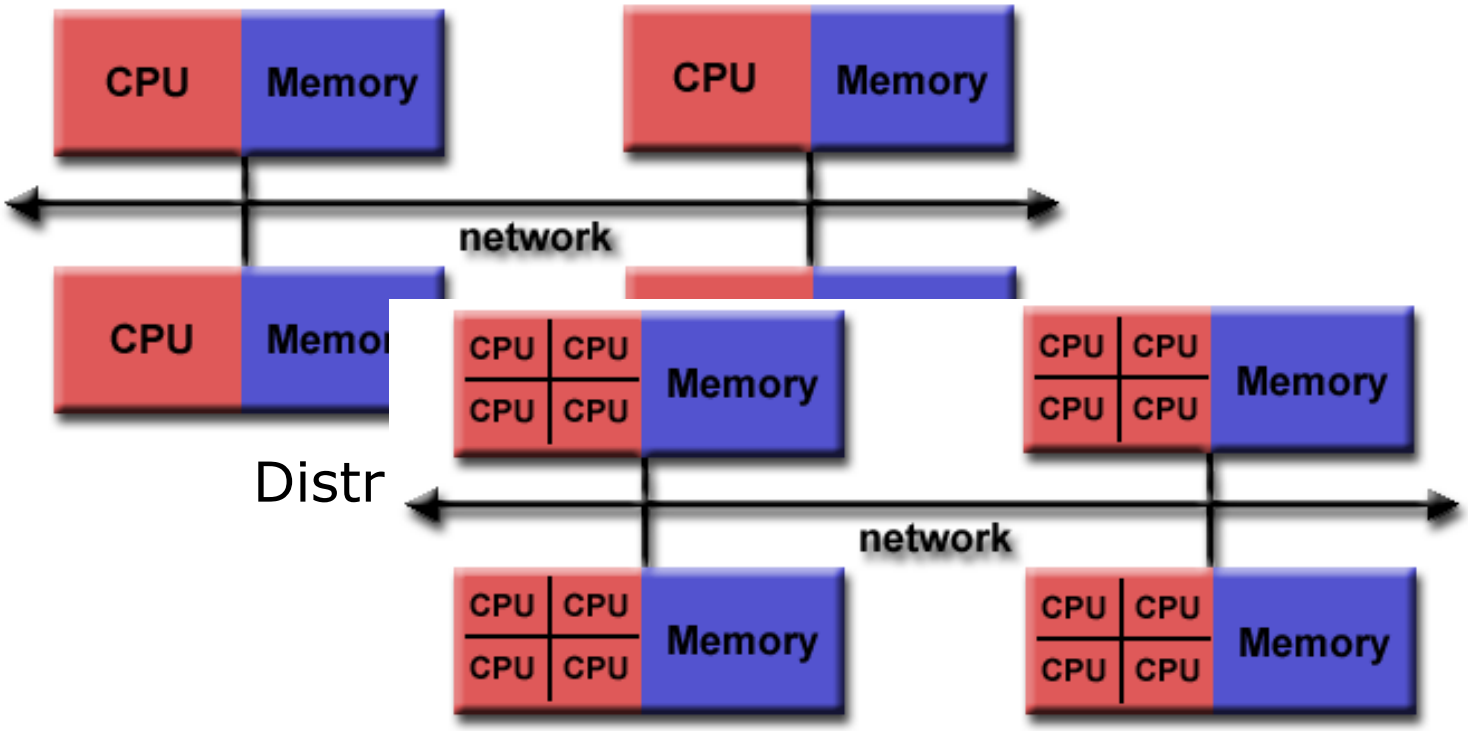
NUMA

Distributed Memory



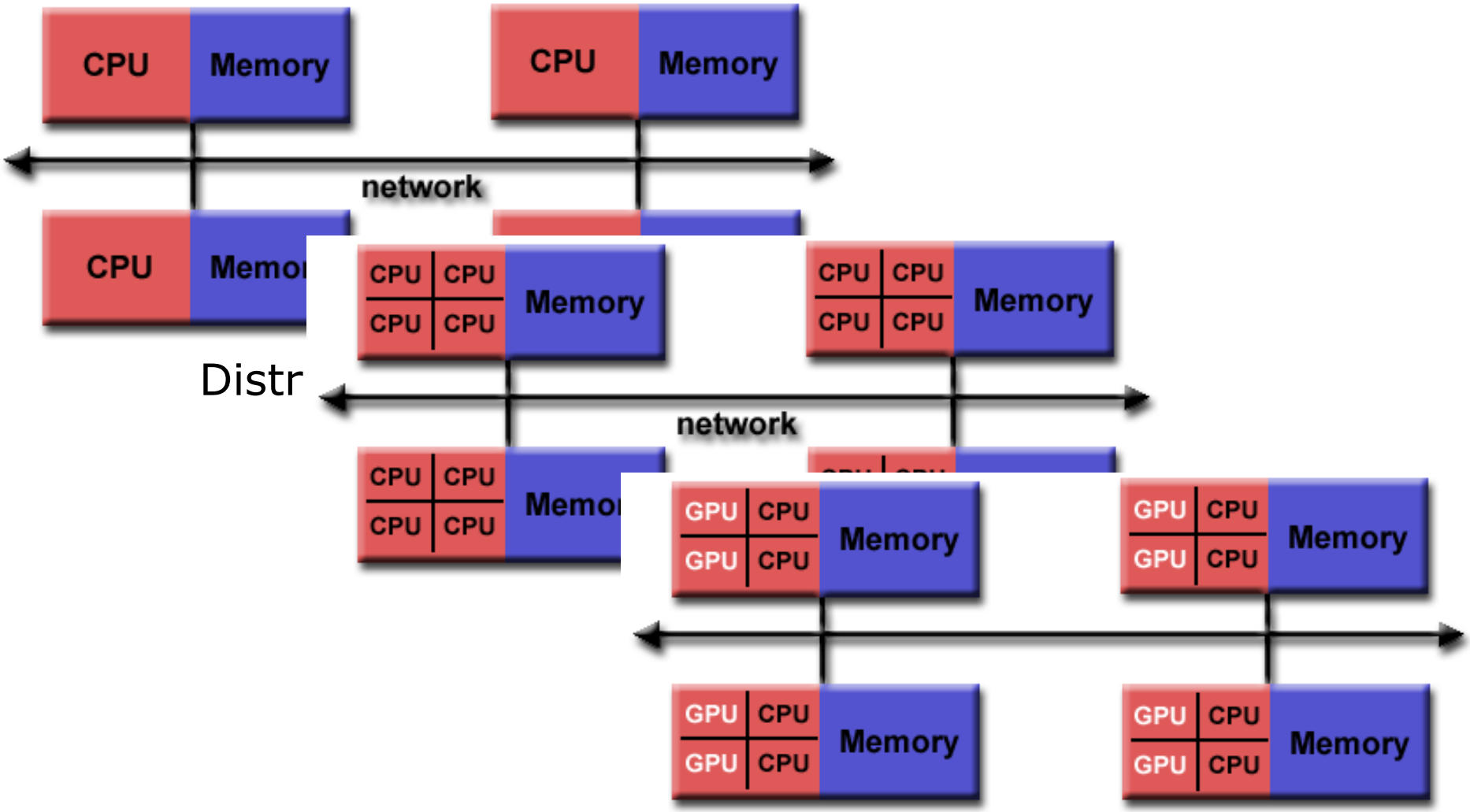
Distributed memory

Distributed Memory



Hybrid

Distributed Memory



Hybrid with GPUs

Parallellization Strategy

- The big challenge is to deal with many different architectures
- The approach most used is to identify algorithmic patterns
- Build a framework for generic programming based on these patterns
- Express new methods in terms of easily parallellized basic operations
- We then only need to develop specific parallel algorithms for these basic operators for each architecture
- A key problem can be domain decomposition

Classes of Image Operators

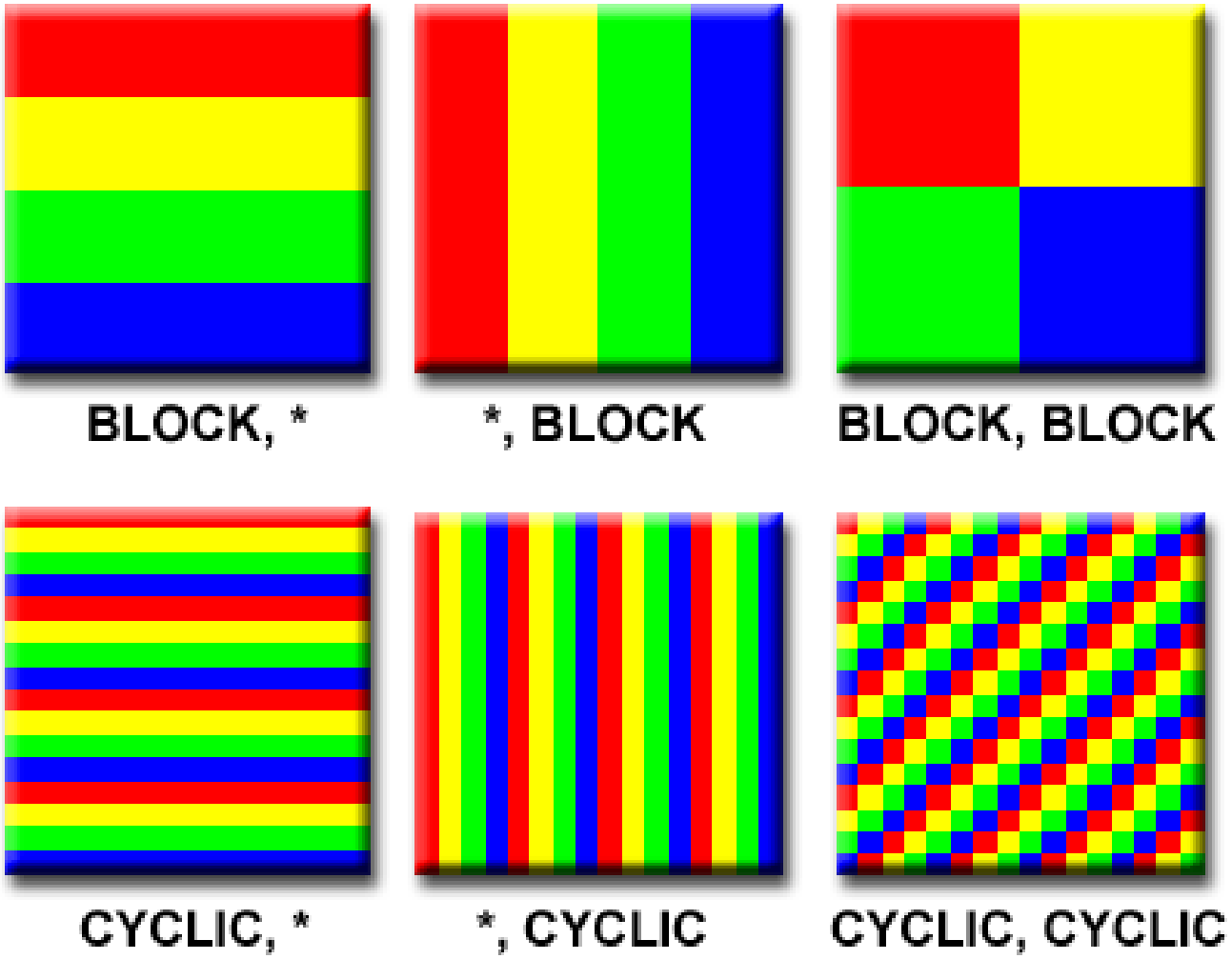
- Point operators
- Neighbourhood operators
- Geometric operators
- Global image $\rightarrow$ image operators
  - Separable
  - Non-separable
- Global reduction operators

Point Operators

- Unary: Look-up-table (LUT) operators
- Image + constant operators
  - Thresholding
  - Arithmetic functions
- Binary: Arithmetic/logic operators
  - Addition/subtraction
  - Max/min
  - Image comparison
- Trivially parallellized

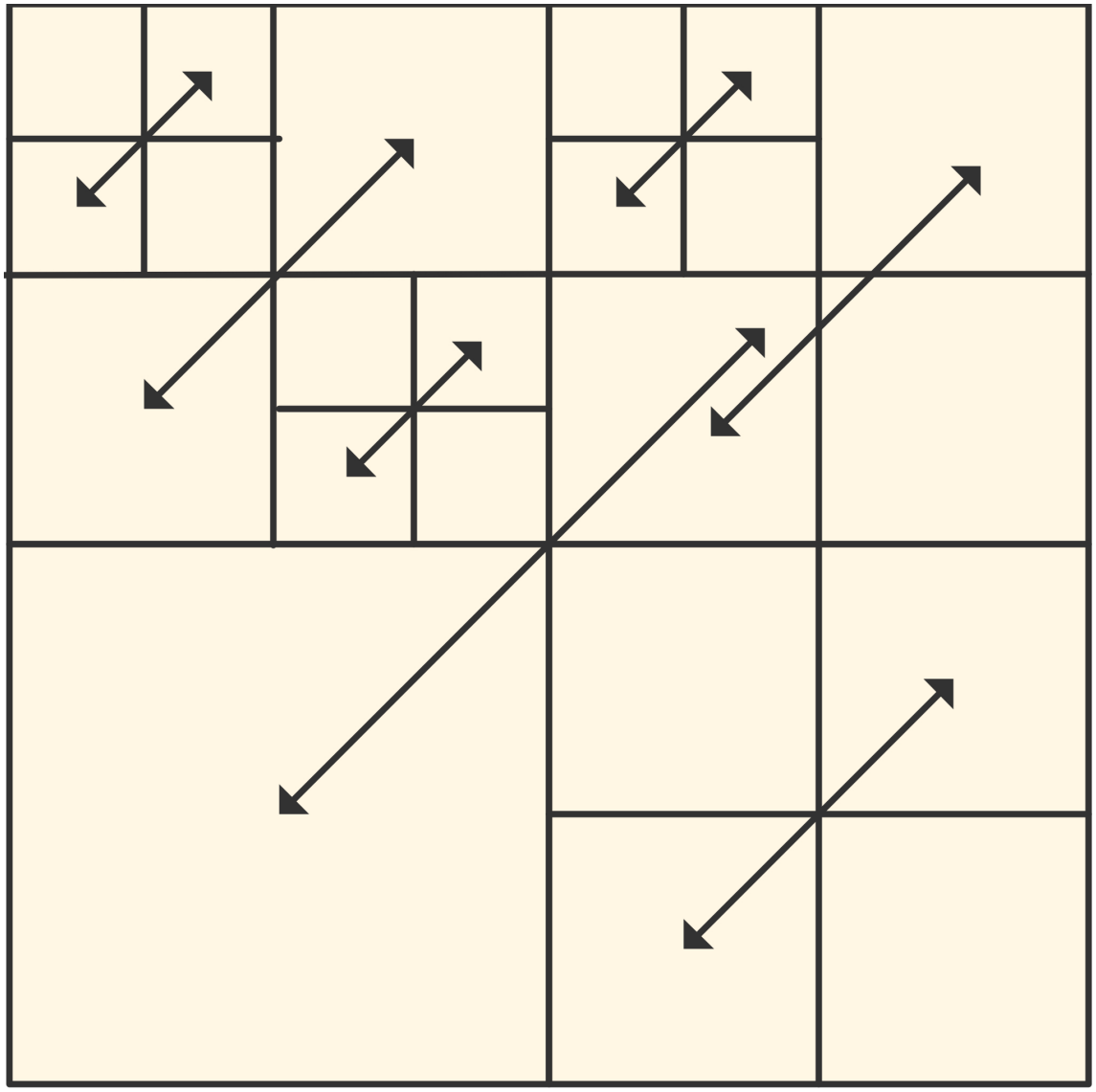
Divide and Conquer!

Any division will do



Geometric Operators

- Translation: one-to-one
- Scaling: neighbourhood
- Transposition: one-to-one
- Rotation
  - 90°: one-to-one
  - arbitrary angle: neighbourhood
- Shearing: neighbourhood
- Generally memory-bound



Neighbourhood Operators

- Convolutions with small kernels
- Morphological SE operators
  - dilations/erosions
  - openings/closings
  - hit-or-miss transforms
- Very suitable for SIMD machines

Generalized Convolutions

- Linear filters are implemented through convolutions

$$g(x,y) = (f * h)(x,y) = \sum_{(i,j)\in N} f(x-i, y-j)h(i,j) \tag{1}$$

- Morphological filters are based on Minkowski addition and subtraction

$$g(x,y) = (f \oplus h)(x,y) = \bigvee_{(i,j)\in N} f(x-i, y-j) + h(i,j) \tag{2}$$

and

$$g(x,y) = (f \ominus h)(x,y) = \bigwedge_{(i,j)\in N} f(x+i, y+j) - h(i,j) \tag{3}$$

Distributivity rules

- Convolutions are linear, shift-invariant, with simple distributivity

$$f * g * h = f * (g * h) = (f * g) * h \tag{4}$$

- For dilations we have

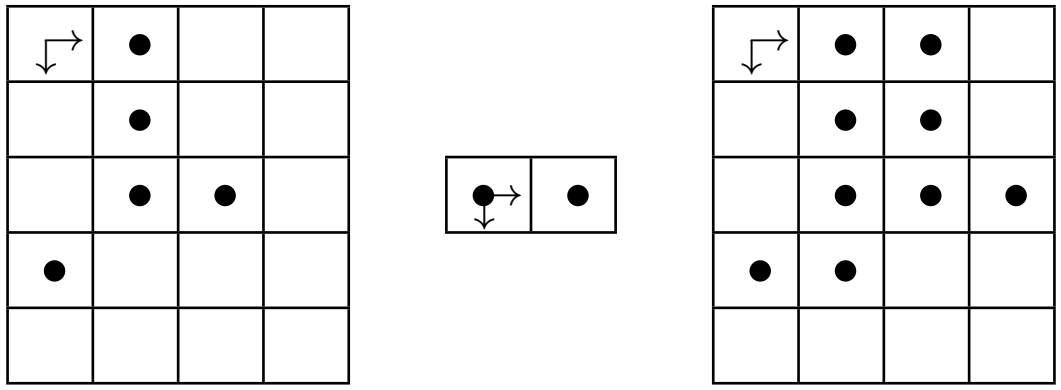
$$f \oplus g \oplus h = f \oplus (g \oplus h) = (f \oplus g) \oplus h \tag{5}$$

- For erosions things are slightly different

$$(f \ominus g) \ominus h = f \ominus (g \oplus h) \tag{6}$$

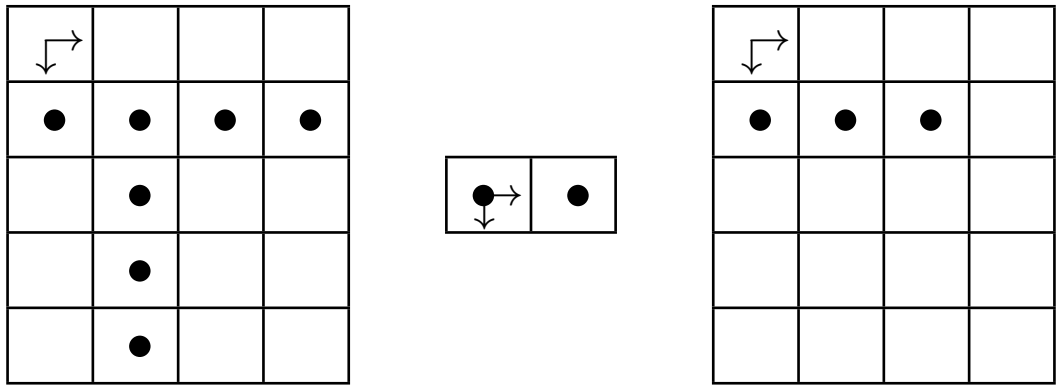
- All rules allow decomposition of kernels or structuring elements (SEs) into a number of smaller ones

Dilation: discrete case



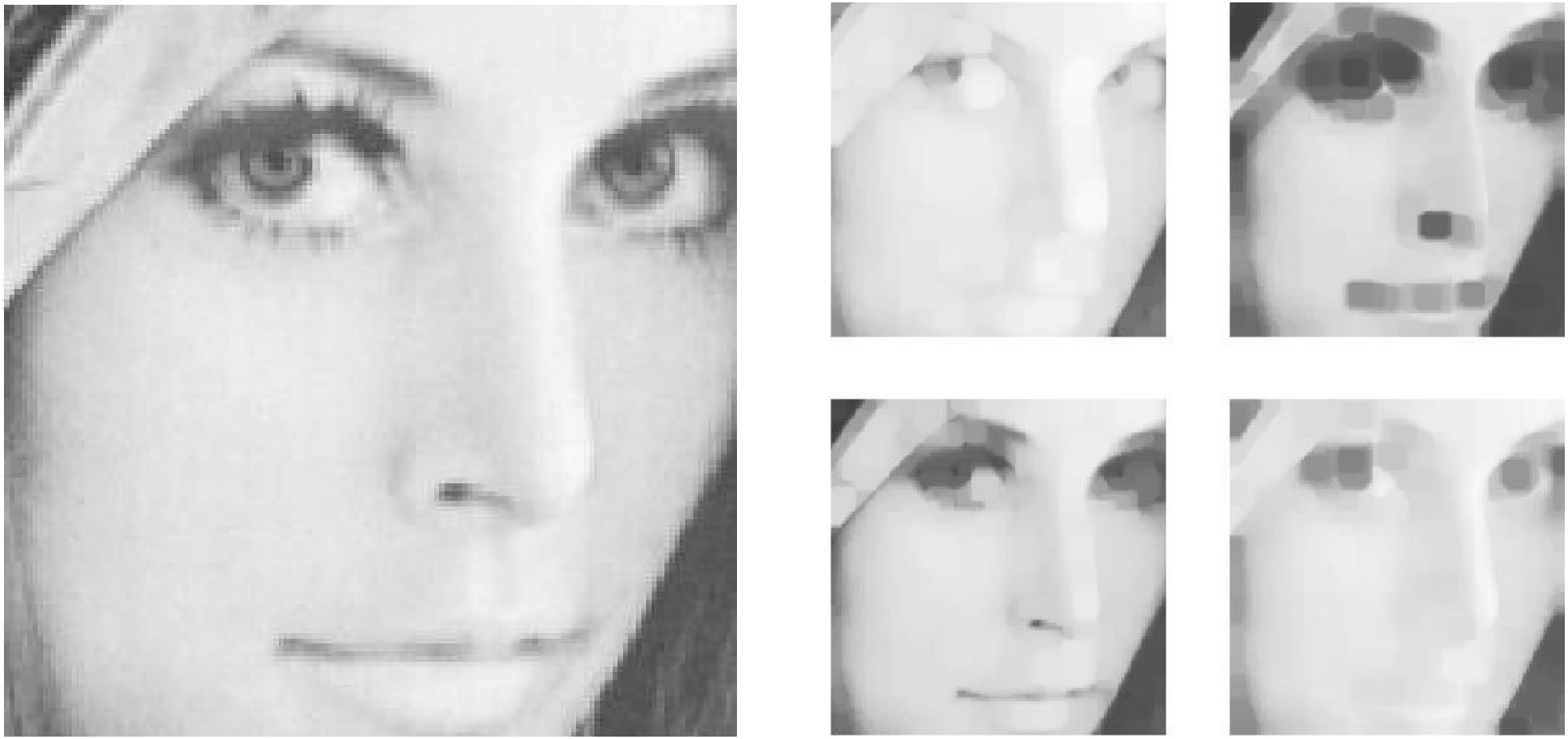
Left: binary image X. Middle: structuring element A. Right: dilation of X by A.

Erosion: discrete case



Left: binary image X. Middle: structuring element A. Right: erosion of X by A.

Basic Operators in Grey Scale



Left: Lenna, the most famous woman in computer vision. Right panel: upper left: dilation; upper right: erosion; lower left: opening; lower right: closing.

Basic Max-Filter Algorithm

- At every pixel we must find the minimum of maximum of all pixels within the S.E..
- Finding the maximum of N unsorted numbers kan be done as follows:

```
i = 1;
maximum = a[i];

while (i <= N) {
  if (a[i] > maximum) {
     maximum = a[i];
  }
  i++ ;
}
```

- How many comparisons must I perform?

Basic Max-Filter Algorithm II

- The entire algorithm for an image of $H \times W$ pixels is:

```
for ( y = 0; y <  H - 1; y++ ){
  for ( x = 0; x < W; x++ ){
     move S.E. to (x,y)
     result[x][y] = maximum(S.E.);
  }
}
```

- The number of comparisons for an $L \times L$ square S.E. is

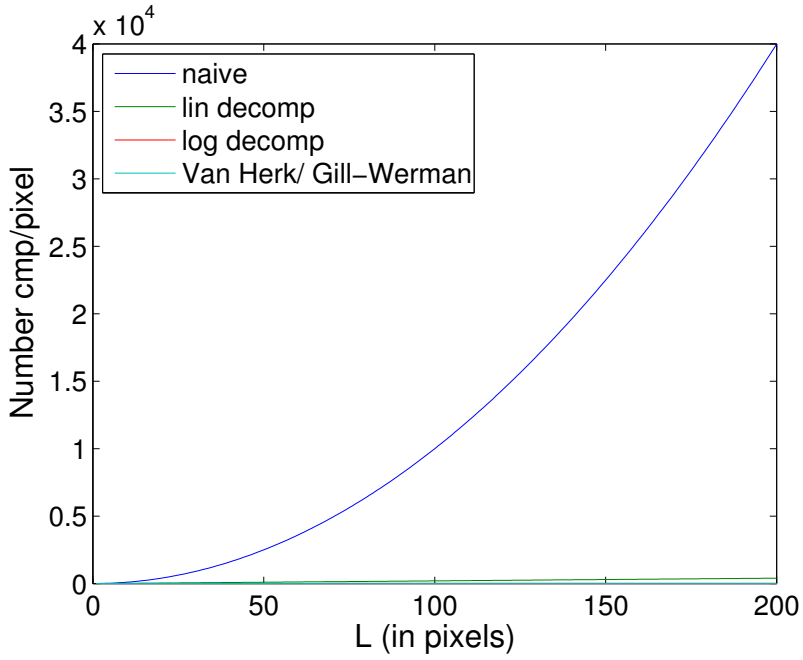
$$W \times H \times (L^2 - 1) \tag{7}$$

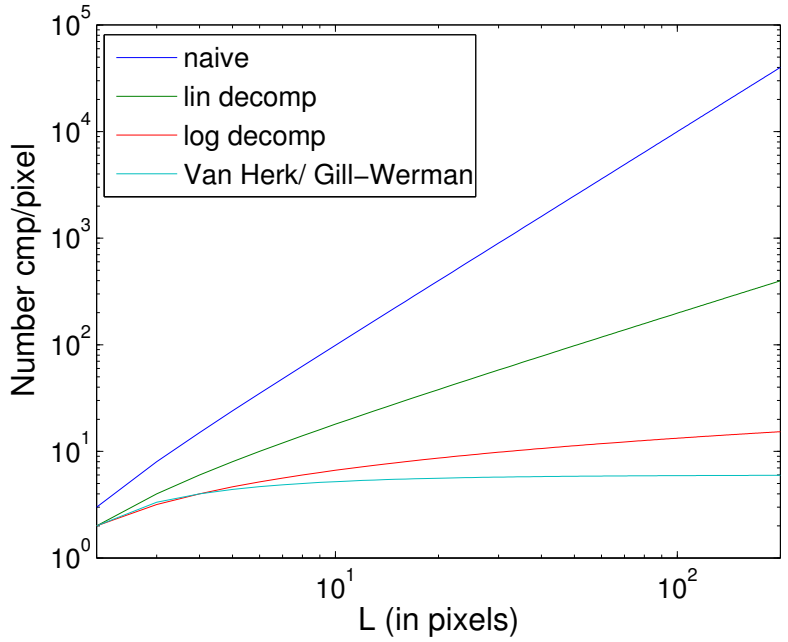
- Can we do better?

- The basic algorithm has complexity

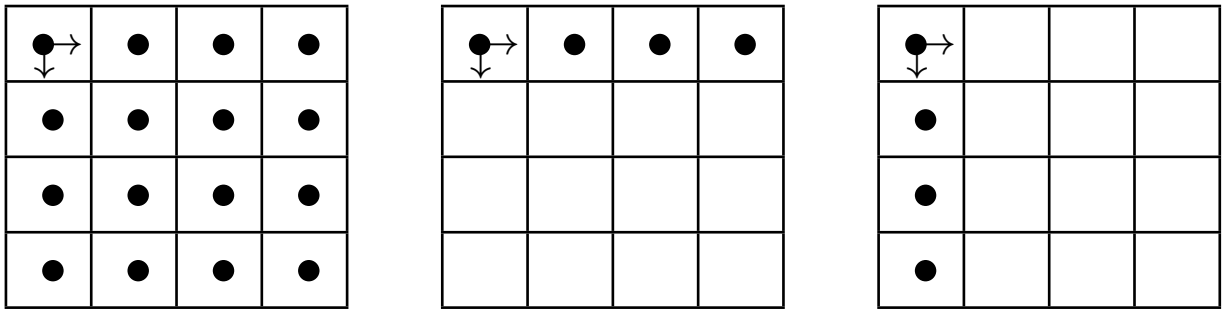
$$O(W \times H \times L^2) \tag{8}$$

- Thus it is *linear* in W en H, and *quadratic* in L.





- A max-filter with square S.E. can be computed from two max-filters with linear S.E. :



- The number of comparisons is reduced to

$$2 \times W \times H \times (L - 1) \tag{9}$$

- The complexity is

$$O(W \times H \times L) \tag{10}$$

Log-Decomposition

- A linear S.E. cna be decomposed into $k = \lceil \log_2(L) \rceil$ S.E.'s wit only points in each S.E.

$$L^i = L^i_0 \oplus L^i_1 \oplus \ldots L^i_{k-1}, \qquad i = r, c$$

with

$$L^r_k = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 2^k \\ 0 \end{pmatrix} \right\}, \qquad L^c_k = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 2^k \end{pmatrix} \right\}.$$

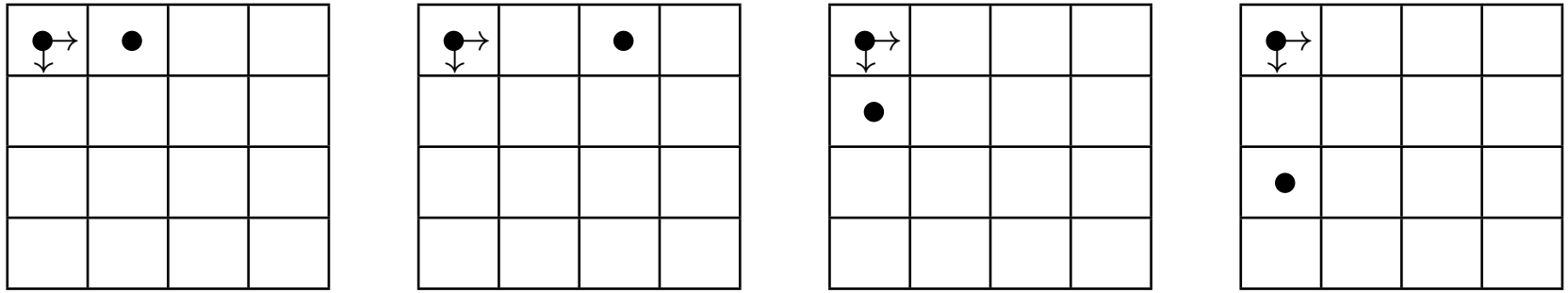
A square is decomposed as:

$$B_k = (L^r_0 \oplus L^r_1 \oplus \ldots L^r_{k-1}) \oplus (L^c_0 \oplus L^c_1 \oplus \ldots L^c_{k-1})$$

with a total of $2k$ max-filters.

Log-Decomposition II

- Example for $L = 4$:



- The number of comparisons reduces to

$$2 \times W \times H \times \log_2(L) \tag{11}$$

- The complexity is

$$O(W \times H \times \log L) \tag{12}$$

- Instead of different decompositions, use different algorithm
- Divide the image row (column) into overlapping segments of $2L-1$ pixels, centred on:

$$j = L-1, 2L-1, 3L-1, \ldots, kL-1 \tag{13}$$

- Every pixel j is in L distinct shifted S.E.
- For every j compute arrays R_j and S_j:

$$R_j(k) = \max(p(j), p(j-1), \ldots, p(j-k)) \tag{14}$$

$$S_j(k) = \max(p(j), p(j+1), \ldots, p(j+k)) \tag{15}$$

with $k \in \{0, 1, \ldots, L-1\}$, and $p(i)$ the grey value at point i in the row (column).

5	4	6	2	1	4	11	10	8	2	0

j

6	6	6	2	1

R_j

S_j	1	4	11	11	11

- Note that we can compute $R_j(k)$ en $S_j(k)$ as

$$R_j(k) = \begin{cases} \max(R_j(k-1), p(j-k)) & \text{if } k > 0 \\ p(j) & \text{if } k = 0 \end{cases} \tag{16}$$

$$S_j(k) = \begin{cases} \max(S_j(k-1), p(j+k)) & \text{if } k > 0 \\ p(j) & \text{if } k = 0 \end{cases} \tag{17}$$

- Thus, we need $2(L-1)$ comparisons.

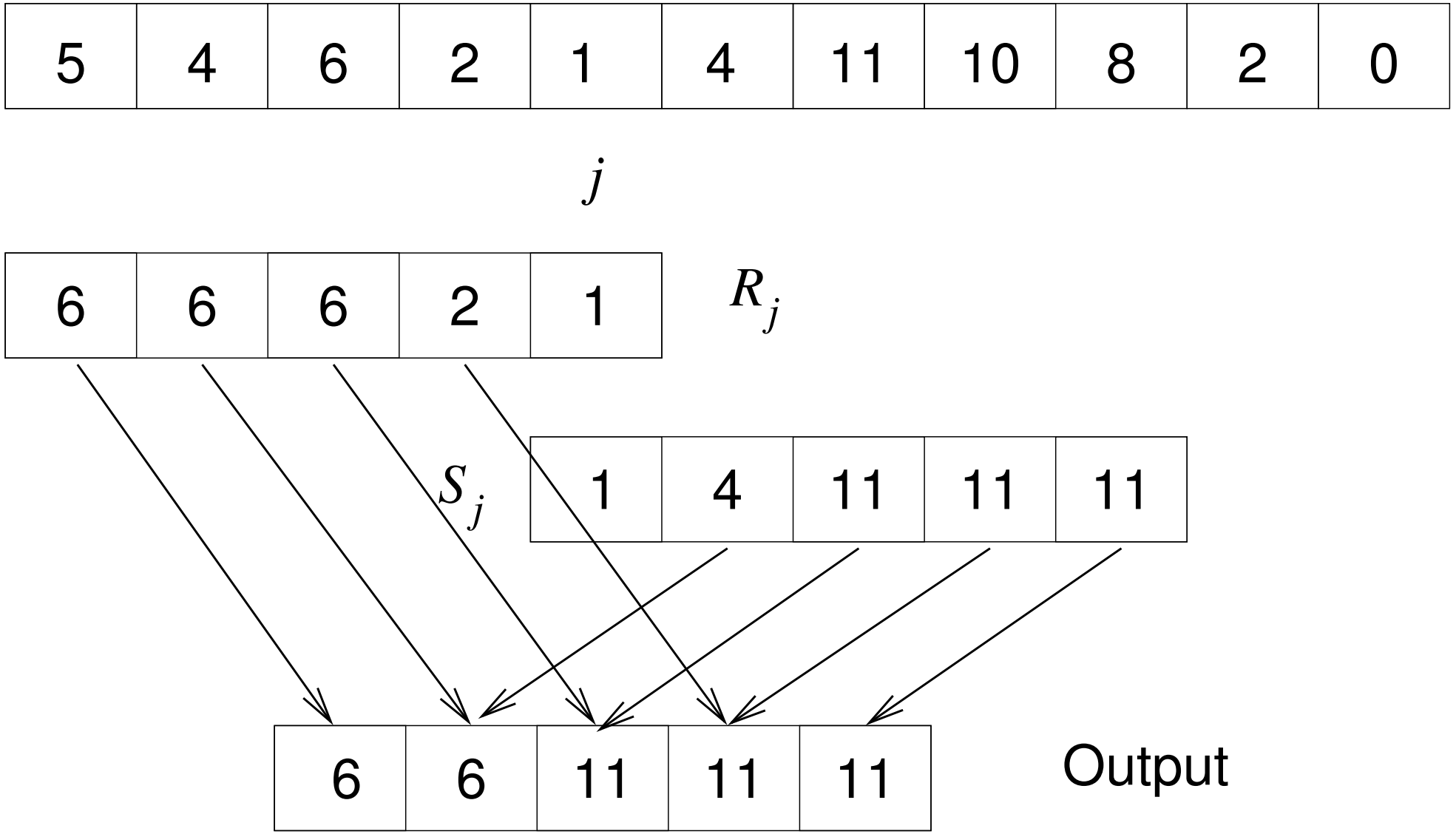
- Another $L-2$ comparisons to compute L ppoints in the final result

$$\max(p(j-k),\ldots,p(j+L-k-1)) = \max(R_j(k), S_j(L-k-1) \tag{18}$$

voor $k = 1, 2, \ldots, p-2$

$$\max(p(j-p-1),\ldots,p(j)) = R_j(p-1) \tag{19}$$

$$\max(p(j),\ldots,p(j+p-1)) = S_j(p-1) \tag{20}$$


5 4 6 2 1 4 11 10 8 2 0
j
6 6 6 2 1
R_j
S_j
1 4 11 11 11
6 6 11 11 11
Output

- To compute L results, we need a number of comparisons equal to

$$2(L-1)+L-1 \tag{21}$$

or, per pixel

$$\frac{2(L-1)+L-1}{L} = 3 - \frac{4}{L} \tag{22}$$

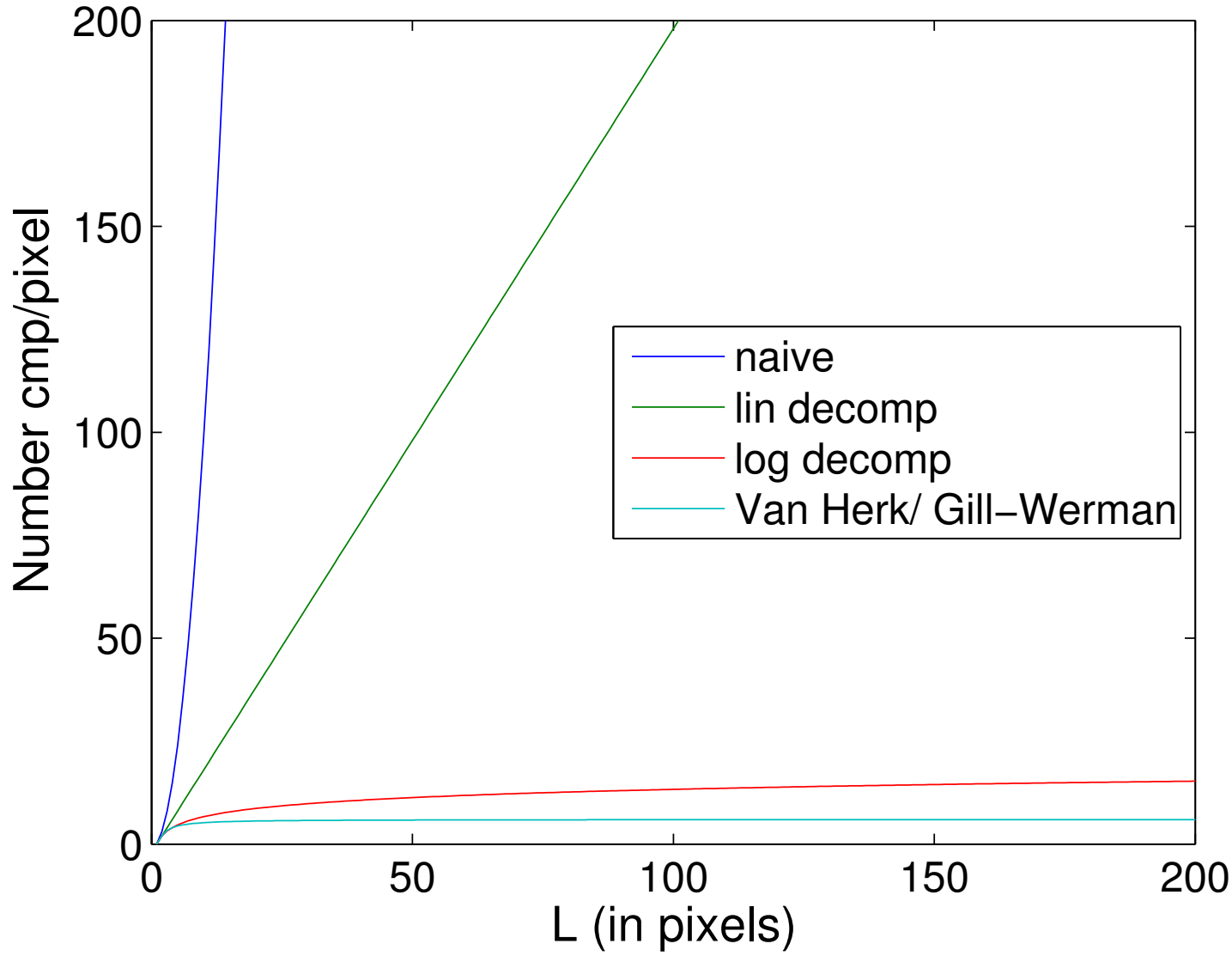
- The total number of comparisons becomes

$$W \times H \times (6 - \frac{8}{L}) \tag{23}$$

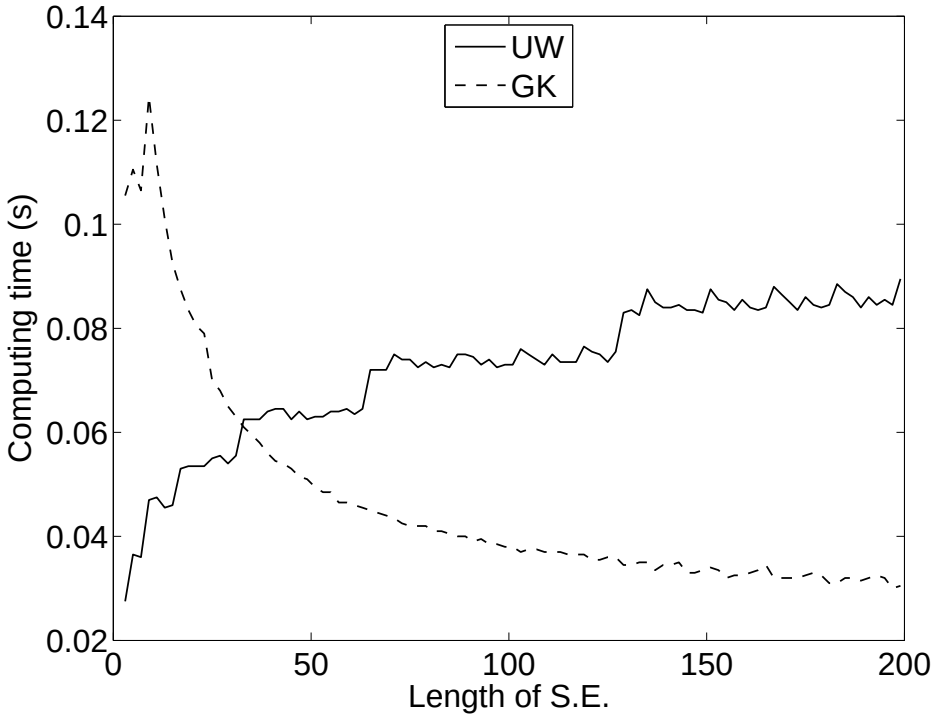
- And the complexity

$$O(W \times H). \tag{24}$$

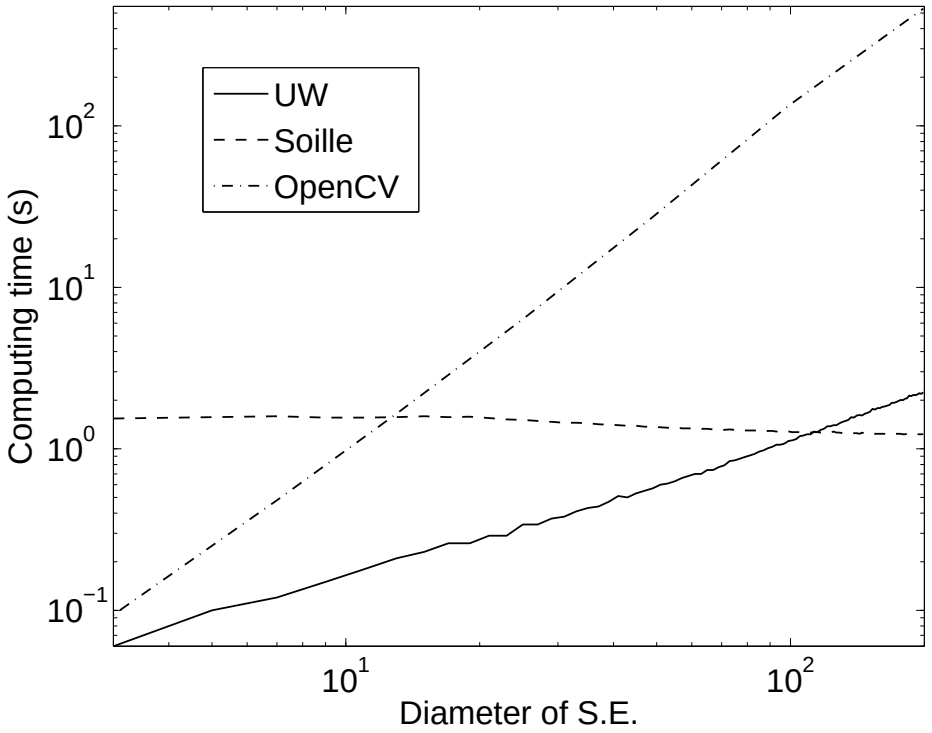
What does it matter?



A few measurements



Lines



Circles

Neighbourhood Filters: Algorithms

- Distribution with overlap
  - Overlap increases with kernel/SE size
  - Can be countered by kernel/SE decomposition
- Shift + combine strategies
  - Makes good use of easily parallellized point operations
  - Requires more barriers

Divide and Conquer Once More

The Cost of Overlap

- On shared-memory machines:
  - No overlap costs incurred
  - Data from adjacent block can be read directly
- On distributed-memory machines:
  - Data needs to be copied
  - Memory or communication cost $O(\sqrt{N_p}(W \times kheight + H \times kwidth))$ in 2D
  - Rises steeply for higher-dimensional data

Basic Convolution Algorithm

A simple sequential algorithm for convolution can be used per block for small kernels:

```
for ( y = 0; y <  H - 1; y++ ){
  for ( x = 0; x < W; x++ ){

     result[x][y] = 0;

     for (j = -kheight/2; j <= kheight/2; j++){
       for (i = -kwidth/2; i <= kwidth/2; i++){
         result[x][y] += f[x-i] [y-j] * h[i][j];
       }
     }
  }
}
```

Complexity: $O(W \times H \times kwidth \times kheight)$, or $O(N^2)$ if the kernel is as large as the image.

Shift + Combine Algorithm

By moving the inner loops over the kernel outwards we can reuse parallel point operators and shifts:

```
ParImageSet(result,0);              // sets all pixels of result to zero

for (j = -kheight/2; j <= kheight/2; j++){
  for (i = -kwidth/2; i <= kwidth/2; i++){
    ParShiftImage(f,temp,i,j);          //creates copy temp of f
                                        // shifted by (i,j)
    ParMulImageConst(temp,h[i][j]);     // multiply temp by h[i][j]
    ParAddImages(result,temp,result); // adds to result
  }
}
```

Shift + Combine Algorithm II

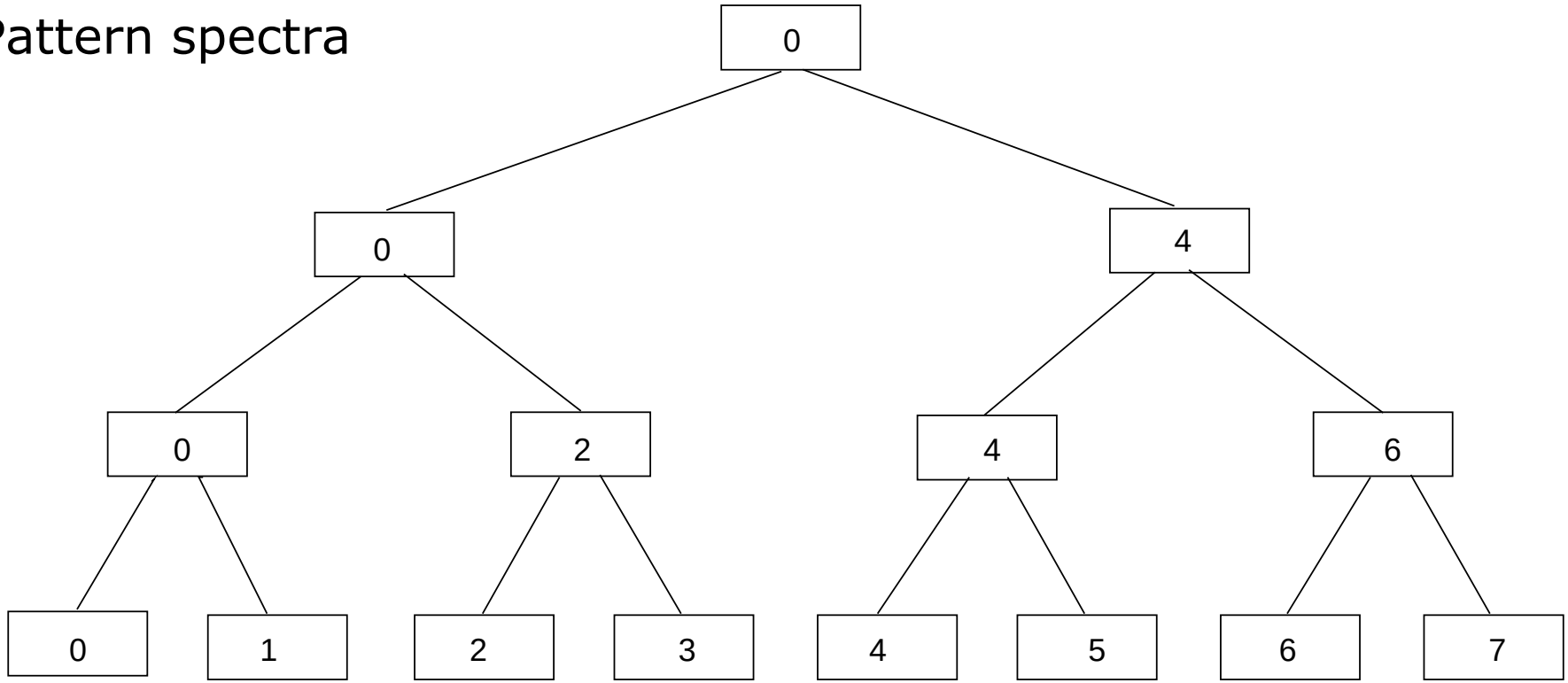
For dilation we obtain:

```
ParImageSet(result,-MAXINT);   // sets all pixels of result to smallest v

for (j = -kheight/2; j <= kheight/2; j++){
  for (i = -kwidth/2; i <= kwidth/2; i++){
    ParShiftImage(f,temp,i,j);          //creates copy temp of f
                                        // shifted by (i,j)
    ParAddImageConst(temp,h[i][j]);     // add h[i][j] to temp
    ParMaxImages(result,temp,result); // compute pixelwise maximum
  }
}
```

Global reduction operators

- Finding global minimum/maxmimum
- Histograms
- Pattern spectra



Global Image → Image Operators

- Large kernel convolutions
- Large, non-flat SE-based morphological operators
- Fourier Transforms
- Distance transforms
- Watershed transforms
- Connected component labelling
- Connected filters

Convolution by FFT

- For large kernels we use the following property:

$$f * h = \mathcal{F}^{-1}(\mathcal{F}(f)\mathcal{F}(h)) \tag{25}$$

  with $\mathcal{F}$ the Fourier transform, and $\mathcal{F}^{-1}$ its inverse

- Using the Fast Fourier Transform we obtain a complexity of $O(N \log N)$.
- Parallel computation of the FFT can simply be obtained through separability
- Many parallel FFT algorithms for specific architectures exist

The convolution algorithm becomes

```
ParFFT(f, ft);                          //Fourier transform f, store in ft
ParFFT(h, ht);                          //Fourier transform h, store in ht


ParMulComplexImages(ht,ft,ft);  // multiply and store in ft
ParInvFFT(ft,g);                        // compute inverse FFT, store in g
```

For added speed, precompute FFTs of frequently used kernels.

Separable global operators

- All methods which allow treatment per dimension
  - Fourier transform
  - Gaussian convolution
  - Parabolic dilation/erosion
- Distance Maps

- The 2D Discrete Fourier transform is given by

$$F[k,l] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[m,n] e^{i2\pi(\frac{ml}{M}+\frac{nk}{N})} \tag{26}$$

- This can readily be rewritten as

$$F[k,l] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[m,n] e^{i2\pi\frac{ml}{M}} e^{i2\pi\frac{nk}{N}}$$

or

$$F[k,l] = \sum_{n=0}^{N-1} e^{i2\pi\frac{nk}{N}} \left(\sum_{m=0}^{M-1} f[m,n] e^{i2\pi\frac{ml}{M}} \right) \tag{27}$$

Gaussian convolution

- The 2D Gaussian kernel kan be separated into an x and y kernel

$$k_{x,\sigma}(x,y) = \frac{\delta_y}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \quad k_{y,\sigma}(x,y) = \frac{\delta_x}{\sigma\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma^2}} \tag{28}$$

- Using the fact that

$$(k_{x,\sigma} * k_{y,\sigma})(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = k_\sigma(x,y) \tag{29}$$

  we have

$$f * k_\sigma = f * (k_{x,\sigma} * k_{y,\sigma}) = f * k_{x,\sigma} * k_{y,\sigma} \tag{30}$$

- 1-D Gaussian convolution is very rapidly done using recursive filters (6 multiplies and adds per pixel) [Van Vliet and Young]

Parabolic Dilation

- The 2D parabolic SE kan be separated into an x and y SEs

$$P_x(x,y) = \overline{\delta}_y + ax^2 \quad P_y(x,y) = \overline{\delta}_x + ay^2 \tag{31}$$

with

$$\overline{\delta}_x = \begin{cases} 0 & \text{if } x = 0, \\ -\infty & \text{otherwise.} \end{cases}$$

- We then have

$$P_x \oplus P_y = a(x^2 + y^2) = ar^2 = P_r \tag{32}$$

and thus

$$f \oplus P_x \oplus P_y = f \oplus P_r \tag{33}$$

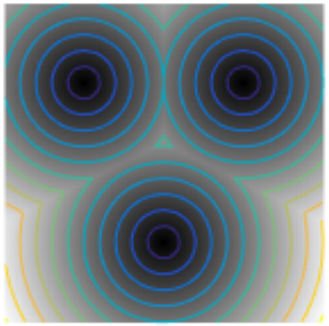
Separable operators: general approach

1. Distribute rows over processors
2. Perform transform per row
3. Rotate or transpose image
4. Perform transform per row
5. Rotate back or transpose image

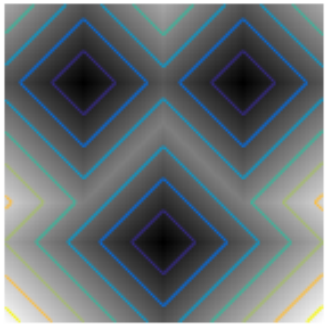
Distance maps

- Turn a binary image into a grey-scale image
- Each grey value indicates the distance to the nearest background pixel

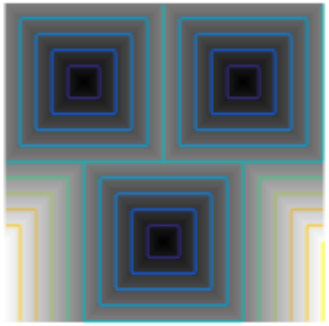
Euclidean



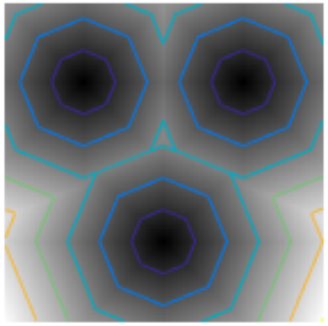
City block



Chessboard



Quasi-Euclidean



Distance Map: An odd-one-out

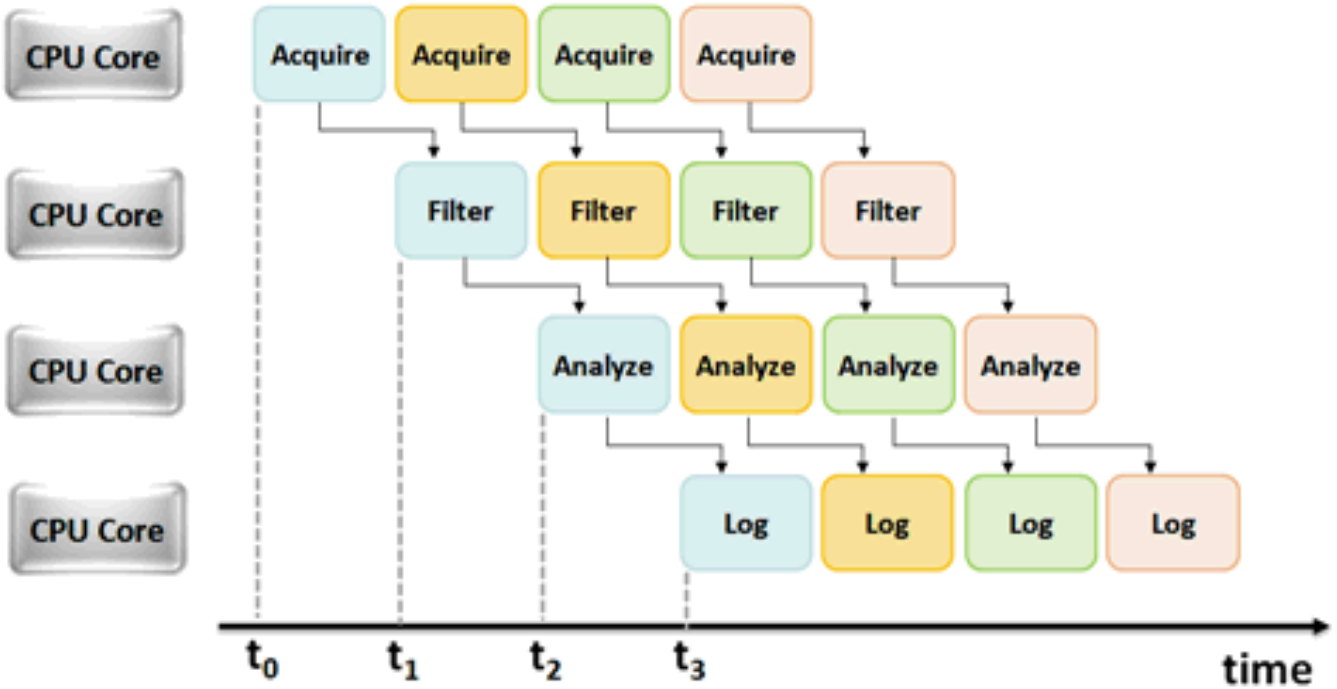
- An efficient, general algorithm has been developed
- It can handle Euclidean, Manhattan, and Chessboard distances
- Can be parallellized effectively by a three-pass method
  - One forward row-wise sweep
  - One reverse row-wise sweep
  - One column-wise sweep

Some Algorithmic Issues

- On shared memory machines:
  - No transposition strictly needed
  - Look out for cache-hostile scan orientations
- On distributed+hybrid machines:
  - Transposition can be costly
  - Tune transposition algorithm carefully to architecture

Other parallel strategies

- Sequence processing
  - Pipe-lines: functional decomposition in time
  - Interleaving/multiplexing: temporal decomposition
- PDE-based filters:
  - PDE solvers readily expressed in matrix-vector operation
  - Use libraries like LINPACK, HPL, etc.
- Convolutional Neural Nets
  - Parallel by nature
  - Very heavy compute and memory load


CPU Core
CPU Core
CPU Core
CPU Core
Acquire
Acquire
Acquire
Acquire
Filter
Filter
Filter
Filter
Analyze
Analyze
Analyze
Analyze
Log
Log
Log
Log
t_0
t_1
t_2
t_3
time

Non-separable, global operators

- Watershed transforms
- Connected component labelling
- Connected filters
- Stuff for next time

Questions