MANCHESTER 1824
The University of Manchester

SCAMP-5: Vision Sensor
with Pixel Parallel SIMD Processor Array

Piotr Dudek
School of Electronic & Electrical Engineering
The University of Manchester

CVPR 2019 workshop, Long Beach, 17/06/2019

Embedded machine vision

Vision is hard…

Even if you got the algorithms that work,
often there is not enough CPU, GPU, battery.

Portable systems, interacting with environment in real-time
require **TOPS** performance @ **mW** of power

Vision System Hardware

Image sensor, ADC

"lots of images"

Memory

lots of data movement!

GPU

CPU

Memory

Vision System Hardware

Vision Chip

"meaningful data"

low-power microcontroller or low-cost CPU

Putting processing where the data is:

High performance (speed, latency, improved capabilities)
at much reduced system power, cost & size

“Event Cameras”

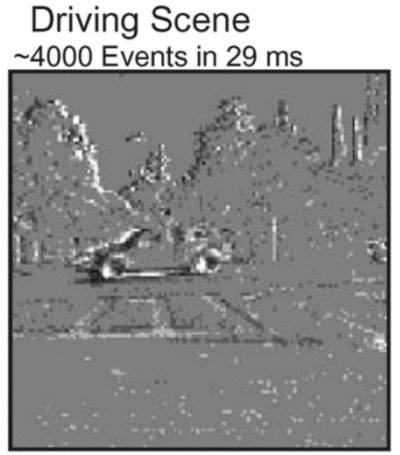
- sparse, high-temporal resolution data extracted from images

- log-intensity changes (DVS, DAVIS, ATIS, Celex, etc.)
  - are only **one type** of temporal feature
  - and perhaps not always the most useful one…
  - NB. Biology does a lot more on the retina!

- how about **other spatial & temporal features?**
  e.g. events corresponding to moving edges,
  corners, general convolution filters, etc.

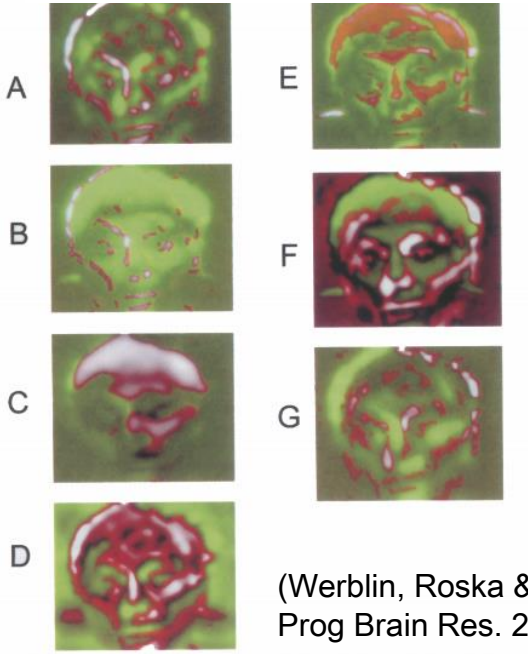
- how about **more complex information extraction**?
  e.g. object centroids, compressive sensing,
  optical flow, tracking, visual odometry, neural net, etc…

- how can we construct a vision sensor that can **do this all**?



(Lichtsteiner et al, JSSC 2008)

spatio-temporal filtering at outputs of different ganglion cell groups



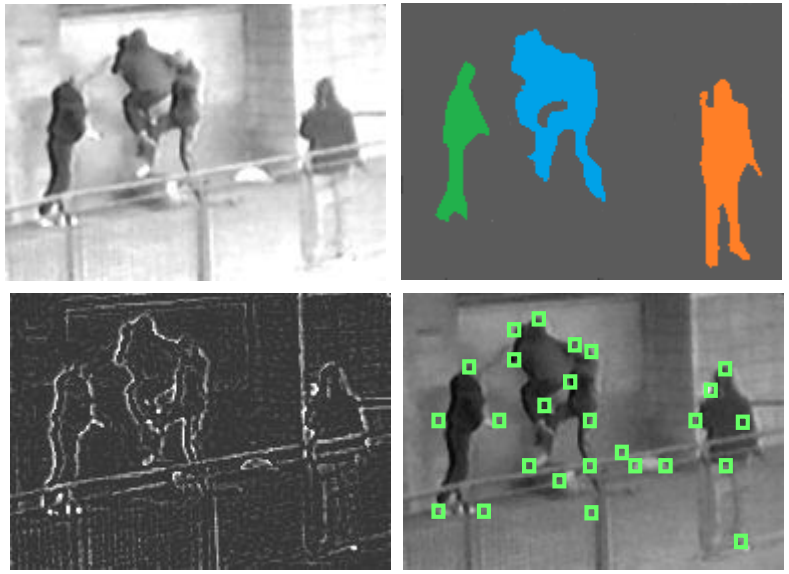
(Werblin, Roska & Balya, Prog Brain Res. 2001)

Early Vision – Pixel-Parallel Operations

a variety of tasks/algorithms:

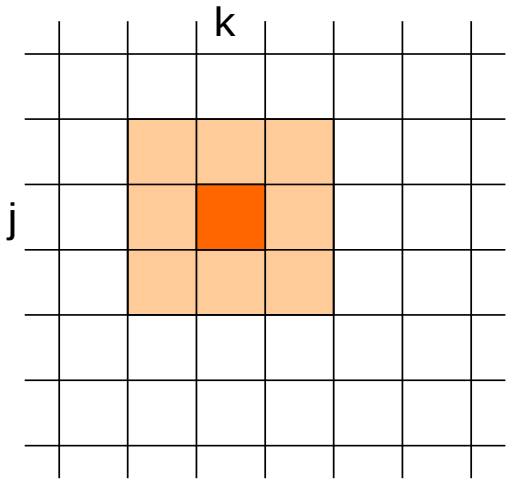
- spatio-temporal filters, convolutions,
- segmentation, optic flow,
- background subtraction,
- adaptive/HDR mapping,
- feature/keypoint extraction,
- object localization & tracking
- CNNs, etc…

"you need a computer for that!"

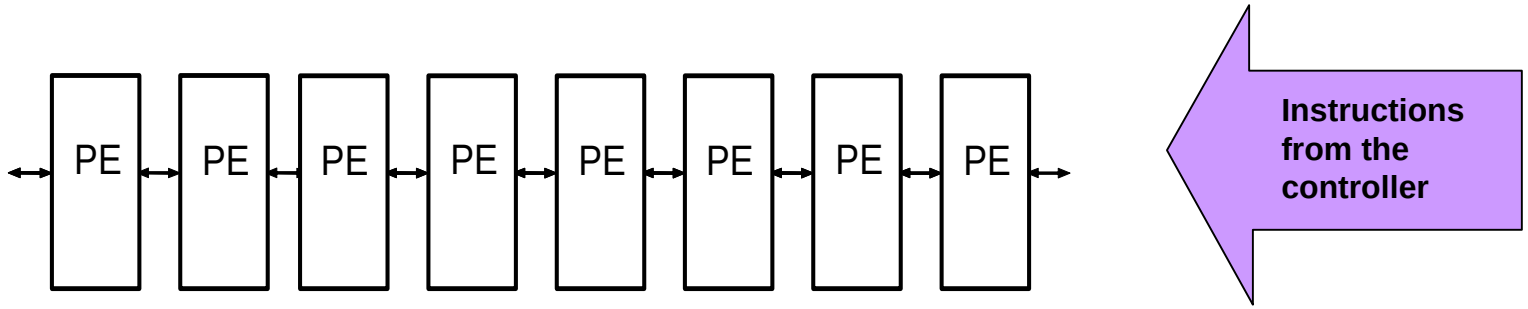


NB. These tasks are **compute intensive** because of the amount of data, many pixels!

- relatively *simple* operations,
- *identical* for each image pixel,
- *localised* (nearest neighbour)

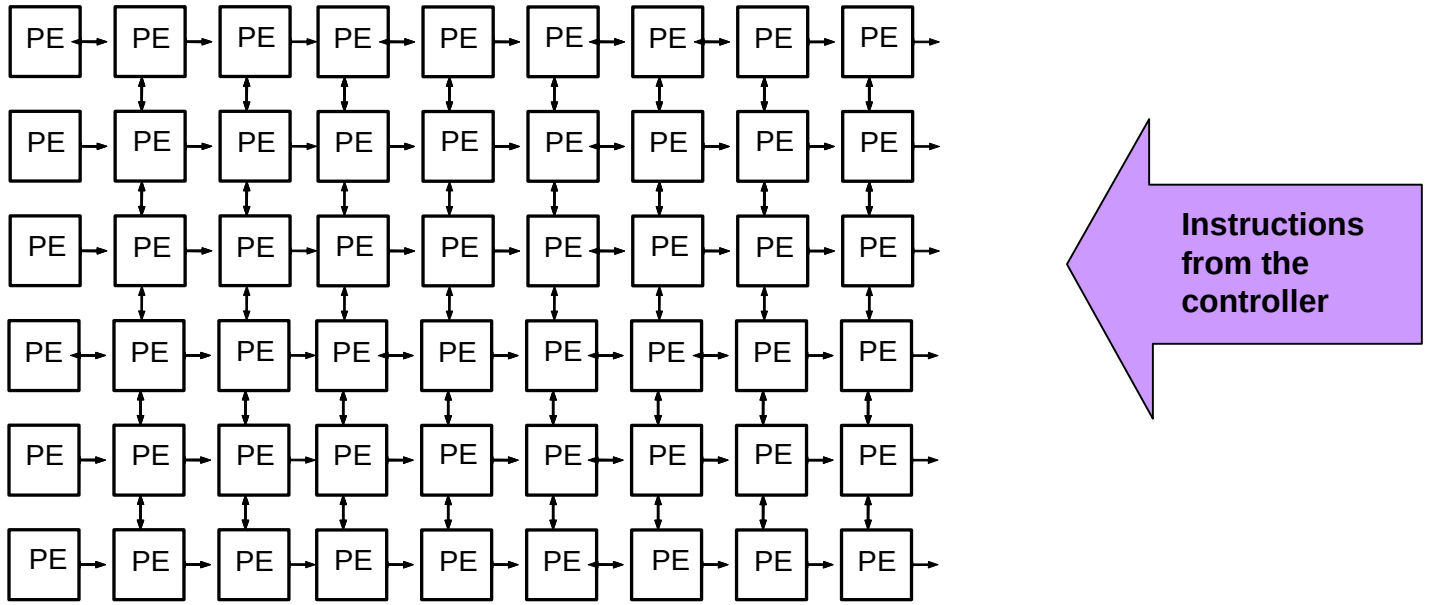


Massively Parallel SIMD Processor Array



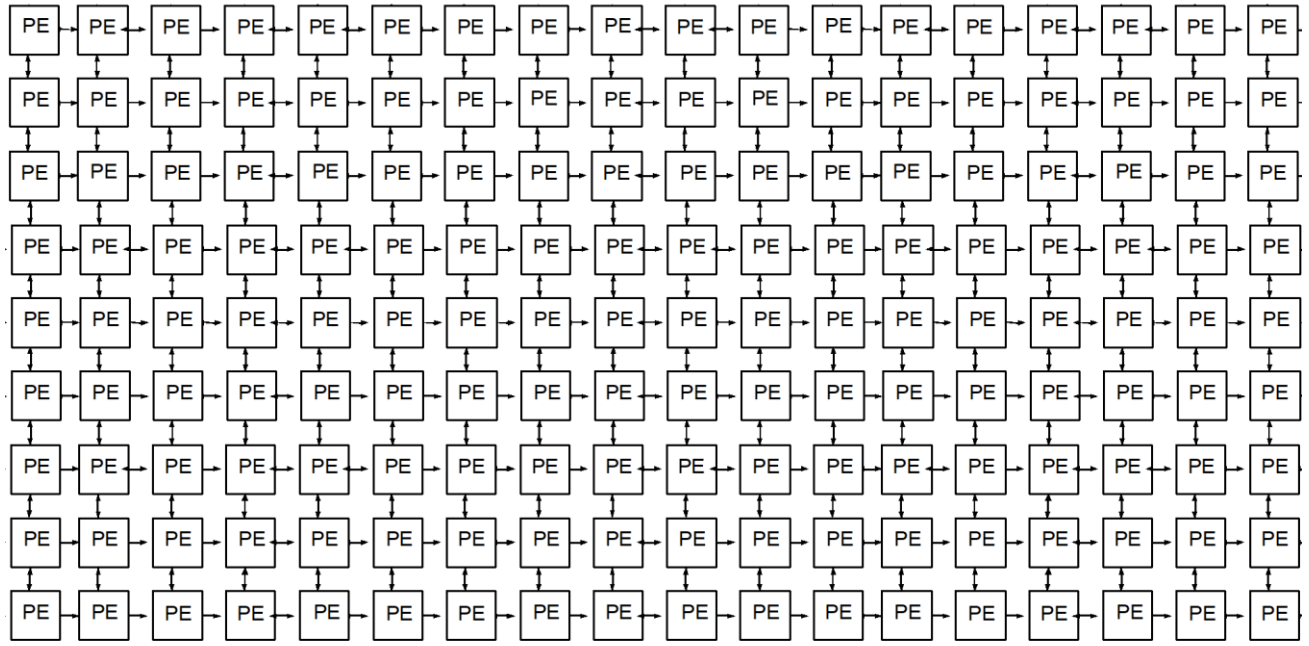
- All processors (PEs) perform the same operation (**Single Instruction**) operating on their local data (**Multiple Data**)

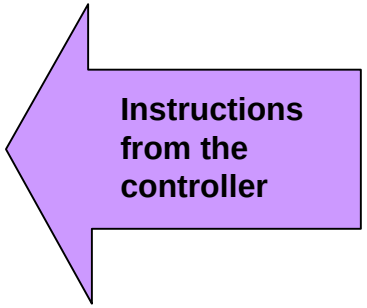
Massively Parallel SIMD Processor Array

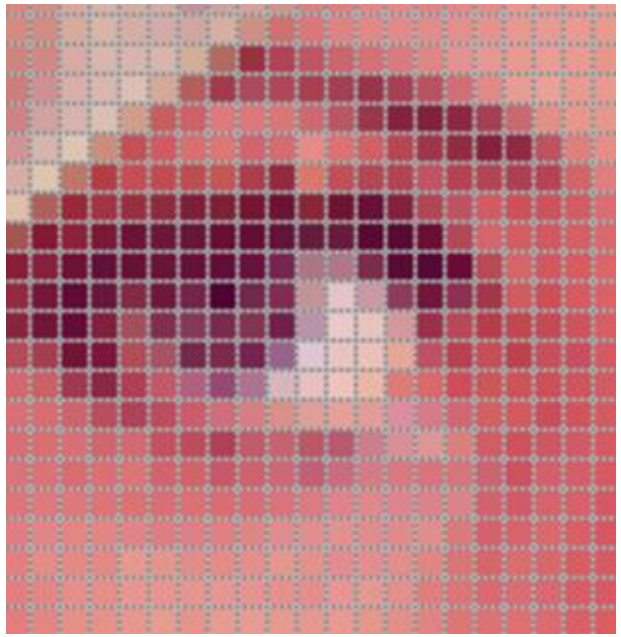


- All processors (PEs) perform the same operation (**Single Instruction**) operating on their local data (**Multiple Data**)
- The more parallel the better…

Massively Parallel SIMD Processor Array



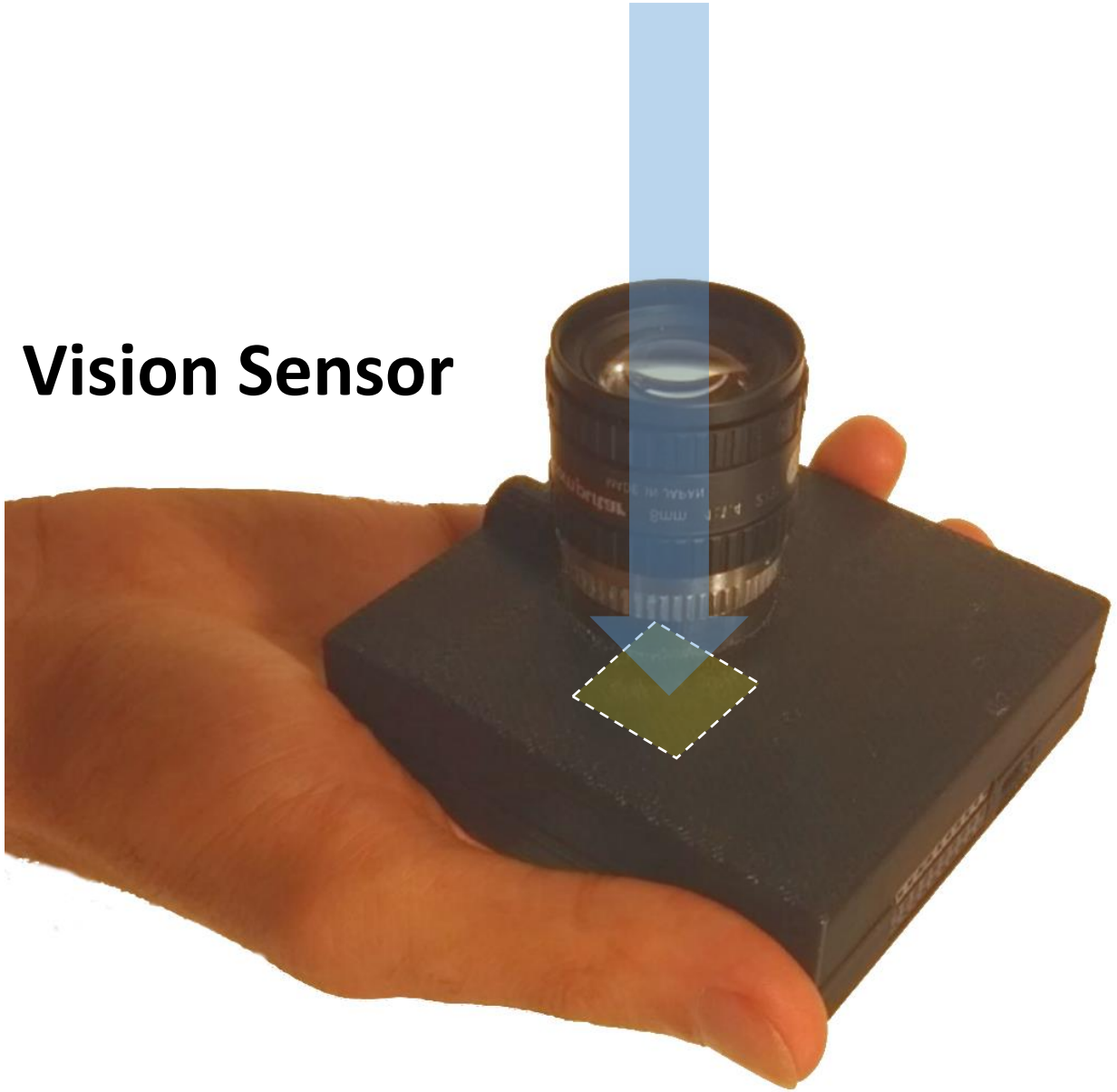




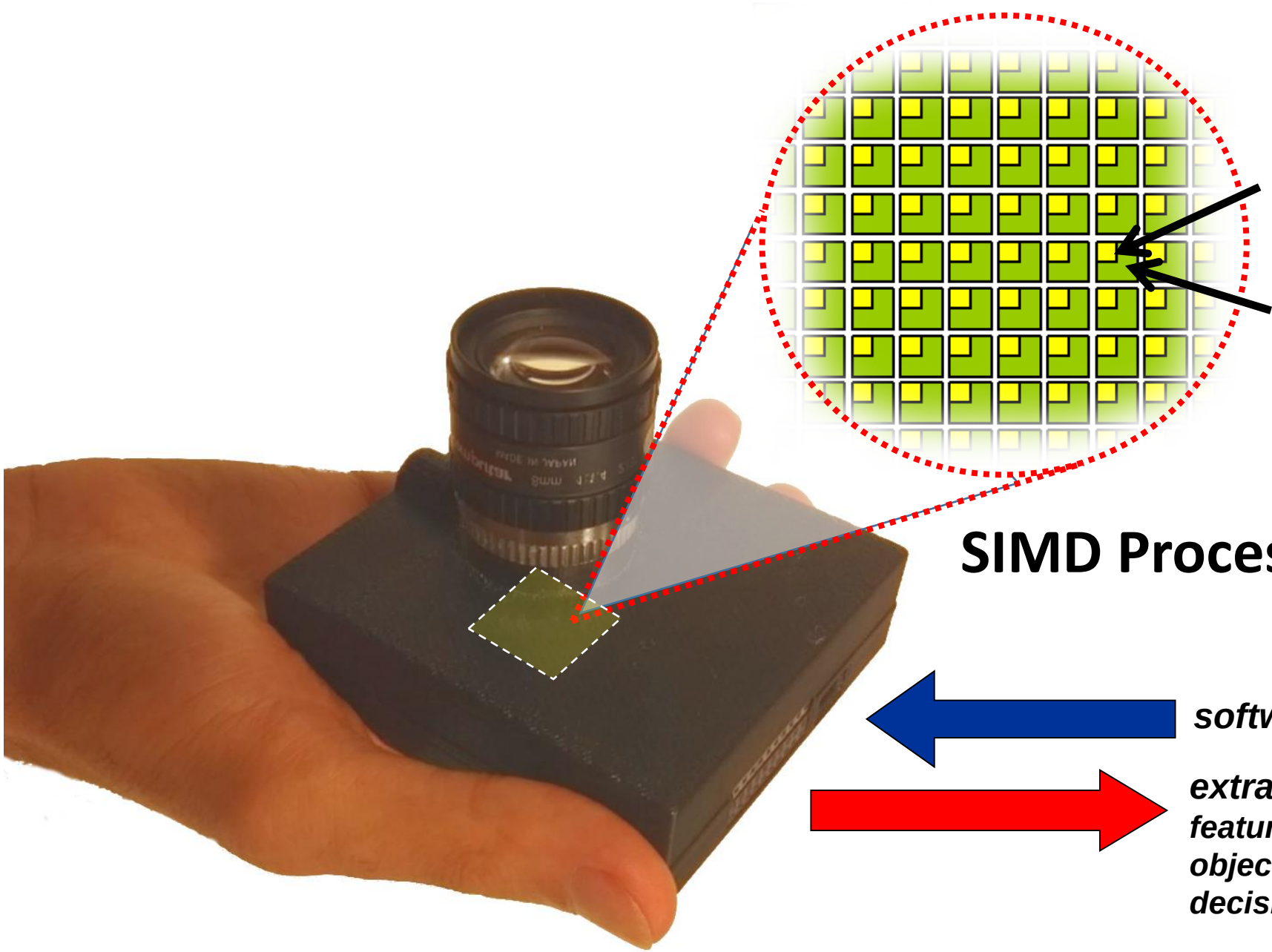
- All processors (PEs) perform
  the same operation (**Single Instruction**)
  operating on their local data (**Multiple Data**)
- The more parallel the better…
- Ultimate parallelism - **one processor per one pixel !**

INSIDE

Vision Sensor with Pixel Processor Array



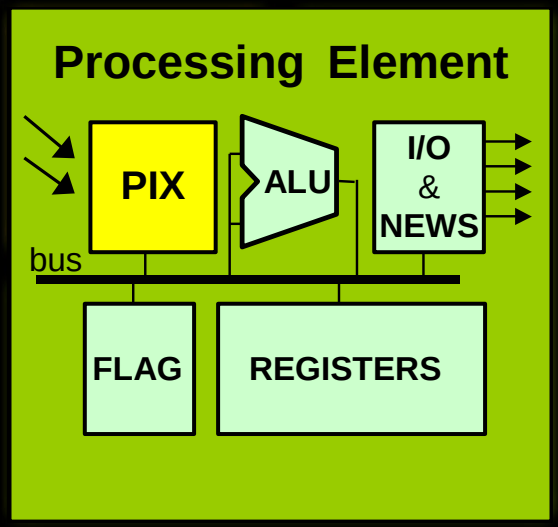
Vision Sensor with Pixel Processor Array

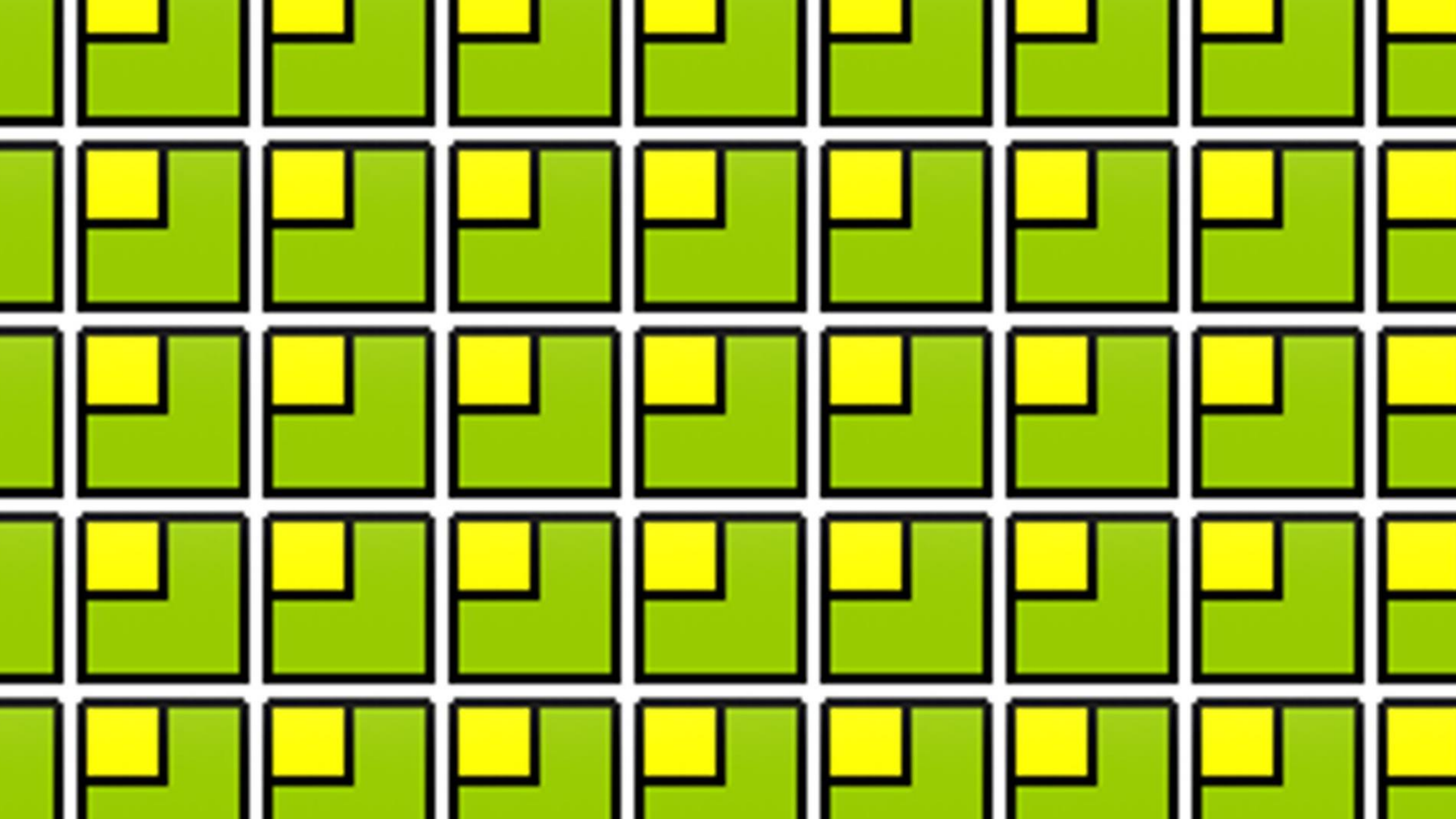


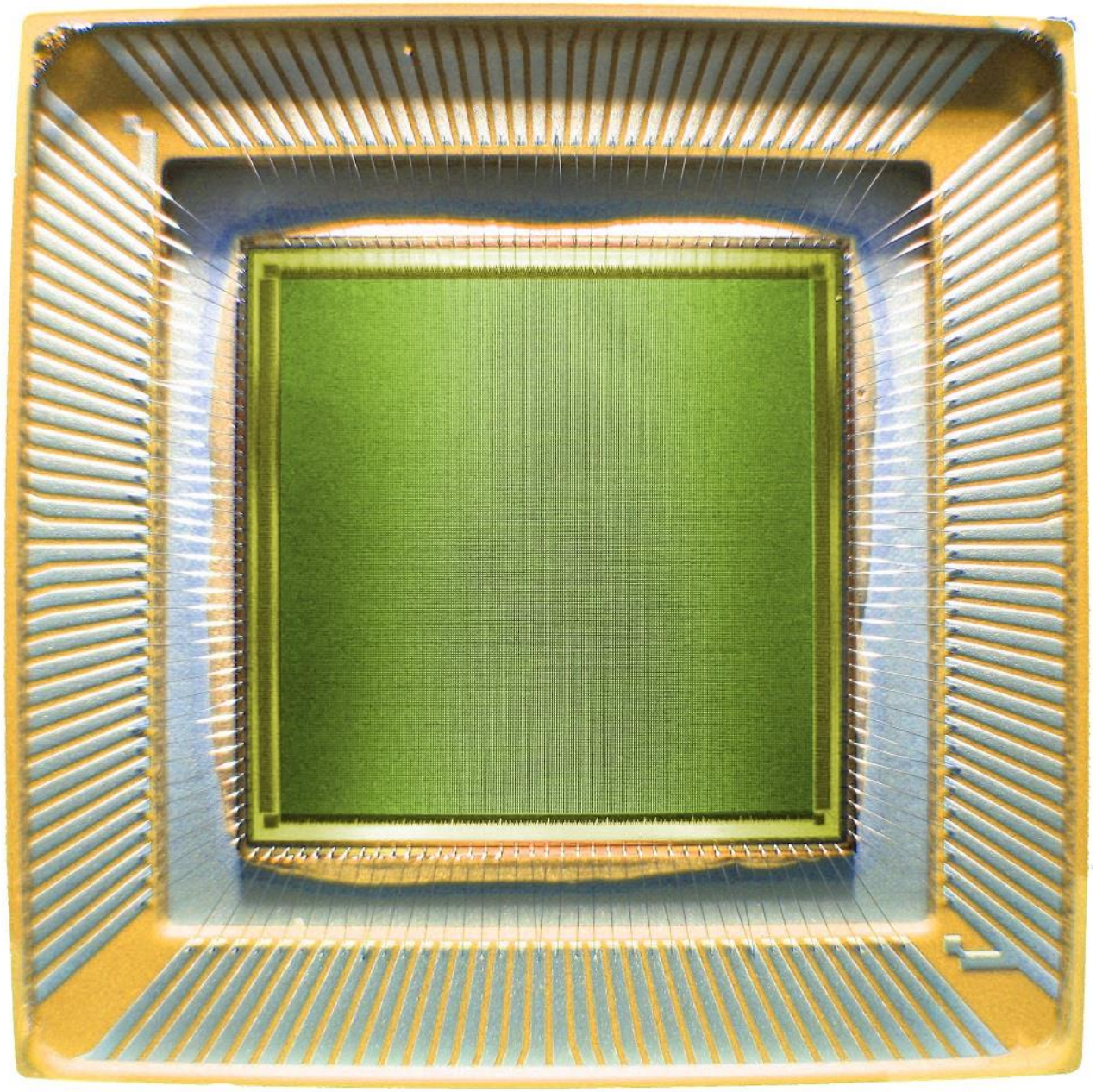
```
rpix(B,C)

usb.aout(C,window_1)

A = copy(C)

s0 = usb.slider(slider_1)
_sub(s0,127) // will set the carry flag if s0 < 127
_jump(c, #begin_loop_shift_west)
_jump(z, #end_loop_shift_h)

#begin_loop_shift_east

    A = west(A)

_sub(s0, 1)
_jump(nz,#begin_loop_shift_east)
_jump(#end_loop_shift_h)

#begin_loop_shift_west

    A = east(A)
```

```
rpix(B,C)

// output the original image
usb.aout(C,window_1)

// fill A with -128 (darkest level)
A = in(-128)

// get the threshold value from the slider in the host UI
s0 = usb.slider(slider_1)

// calculate: E = C - D, after D is filled with our threshold value
D = in(s0)
E = sub(C,D)

// flag those PEs where
where(E)
    // only those flag
    // fill A with 127
    A = in(127)
all
```

```
R11 = 0
R10 = 0

// dark foreground bright background

s0 = usb.slider(slider_0)
E = in(s0)
A = add(C,E)
_call(#algorithm_FAST16_R9)
_nop

R10 = R9

// bright foreground dark background

E = neg(C)
C = copy(E)

s0 = usb.slider(slider_0)
E = in(s0)
A = add(C,E)
_call(#algorithm_FAST16_R9)
_nop

// merge the two
```







SCAMP-5 vision chip

256X256 Pixel Processor Array

Parameter	Value
No. of Processors	65,536
PE cell size	32um x 32um
Memory per PE	7 analog + 14 bits
Clock	10 MHz
Peak performance	655 GOPS
Power consumption	1.23 W (peak) 0.2 mW (idle)
Efficiency	**535 GOPS/W**

in **180nm** CMOS!
(20-yrs old tech)

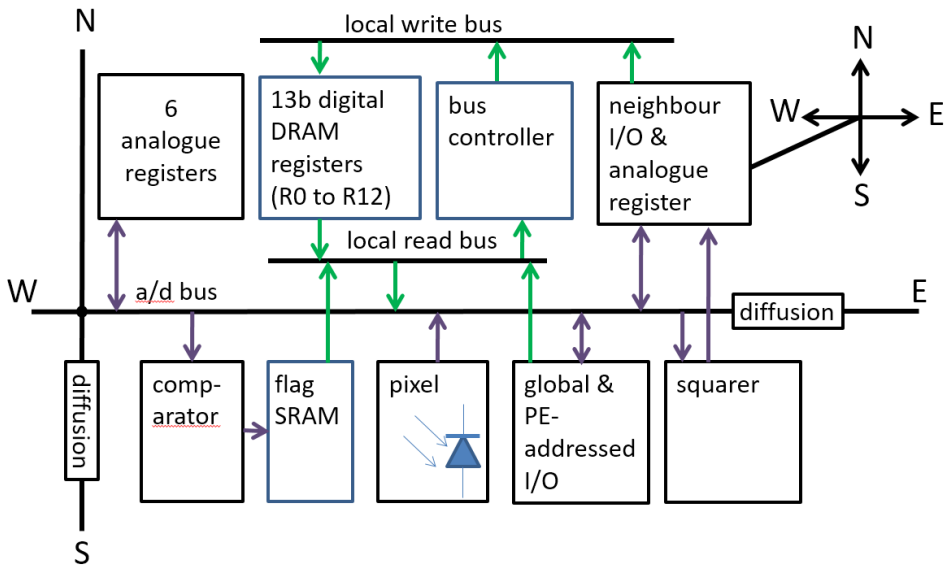
(Carey, Lopich, Barr, Wang and Dudek, VLSI Symp 2013)

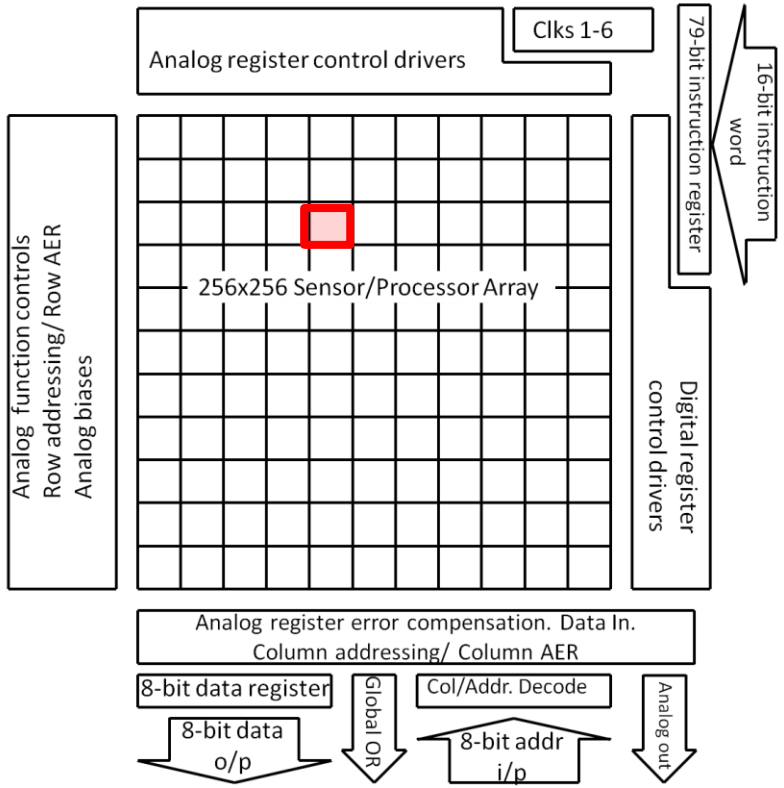
SCAMP-5 Architecture

- Mixed analog/digital datapath
- Local memory 6 analog + 13 bits per PE
- Custom instruction set:
  `mov, add, sub, div, where-all, ...`

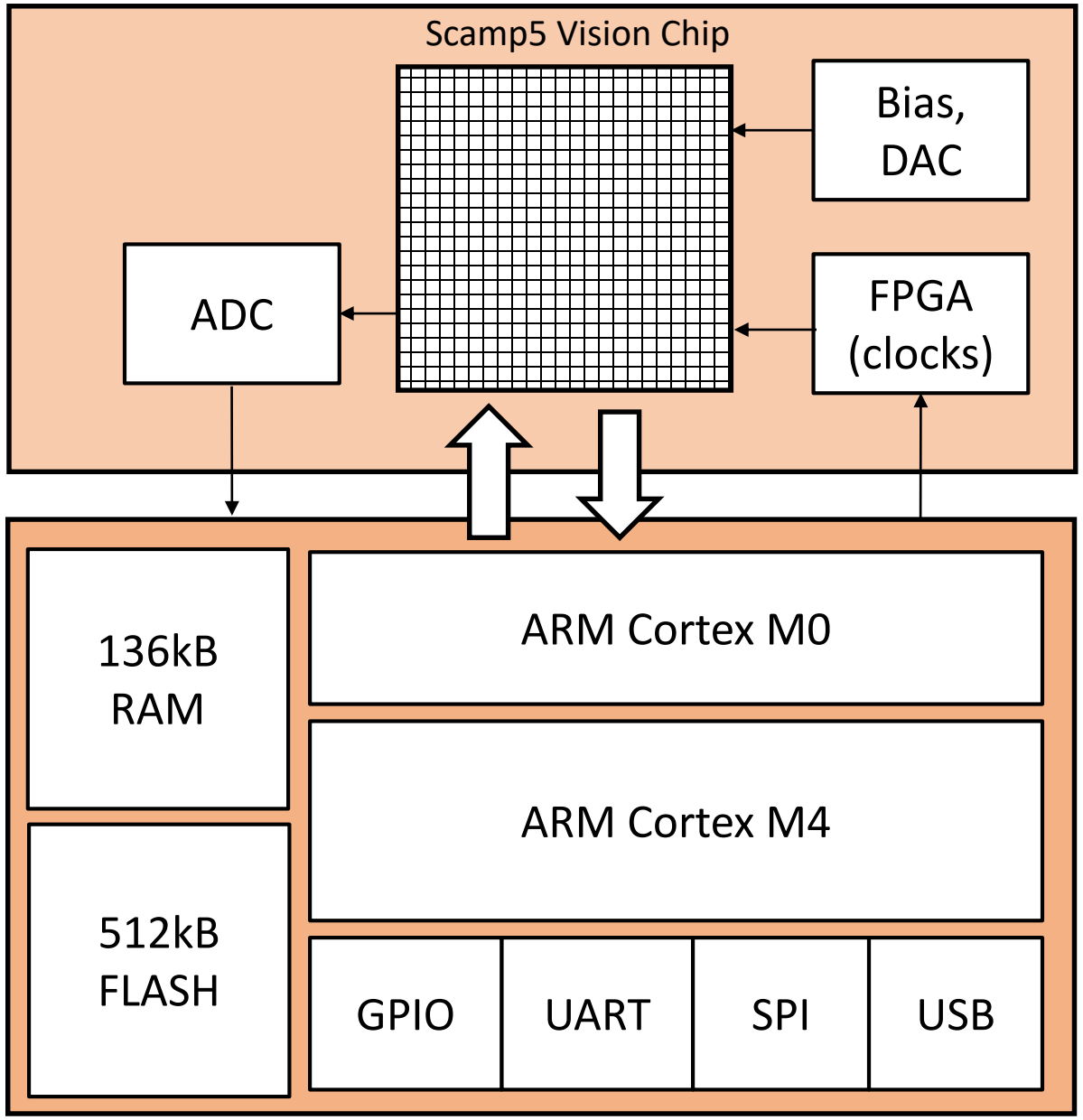
- 256x256 processor array
- periphery: control, interface (external sequencer)

- Random access I/O
  - analog & binary
- Global I/O:
  - block sum, block OR
- Pixel-driven I/O
  - extracting active pixel addresses **(events)**





Scamp5c Smart Camera System

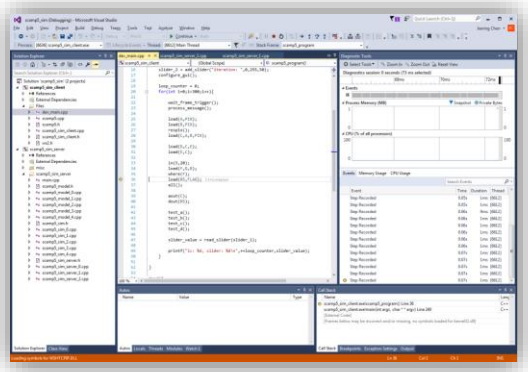


Host Processor
(Odroid, RaspberryPI, Arduino, etc…)
SCAMP Interface API
ROS interface
Host GUI, Code dev & Debugging on a PC

```
#include <scamp5.hpp>
using namespace SCAMP5_PE;

int main(){

vs_init(); //initialise and configure interface
auto display_1 = vs_gui_add_display("result",0,0,2); //output channels
auto slider_1 = vs_gui_add_slider ("Count",0,5,2,&sliderCount); //parameters

while(1){
        vs_process_message();
        vs_wait_frame_trigger();

        scamp5_kernel_begin();
            get_image(C,D);
            where(C);
                add(A,E,C);
                mov (C,F);
            all();
        scamp5_kernel_end();

        for (int i=0; i<sliderCount; i++){
                scamp5_kernel_begin();
                    mov(B,C,west);
                    add(C,B,C);
                scamp5_kernel_end();
        }
        scamp5_output_image(B,display_1);  //output via USB/API to external host
        vs_loop_counter_inc();
    }
return 0;}
```

Software Development Kit

Visual Studio



Simulation Binary (EXE)

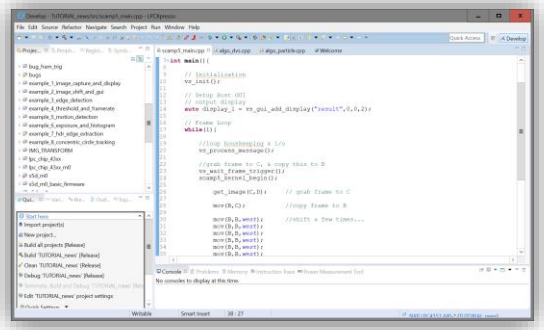
Scamp Simulation Library

Vision Algorithm in C/C++

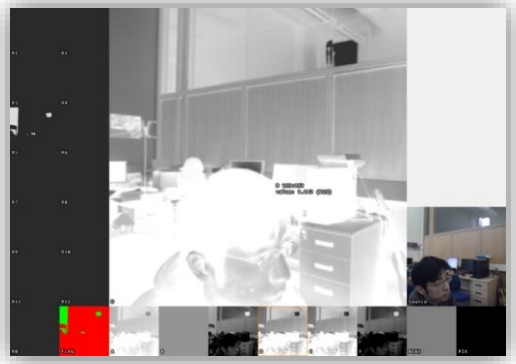
Scamp5d Cortex-M0 Library

LPC4357-M0 Binary (AXF)

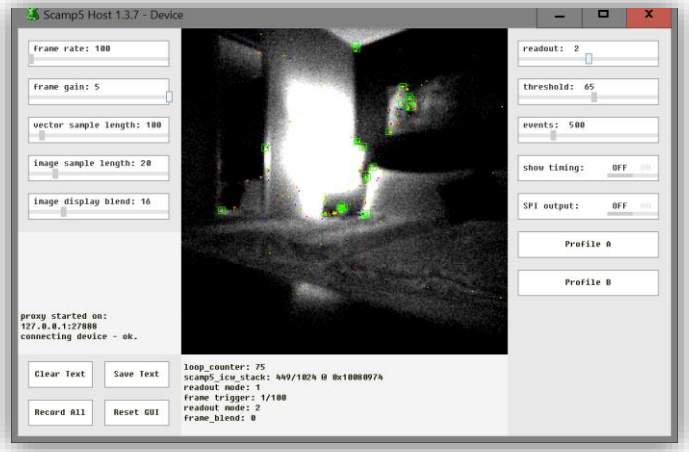
LPCXpresso (Eclipse)



Scamp Sim Server



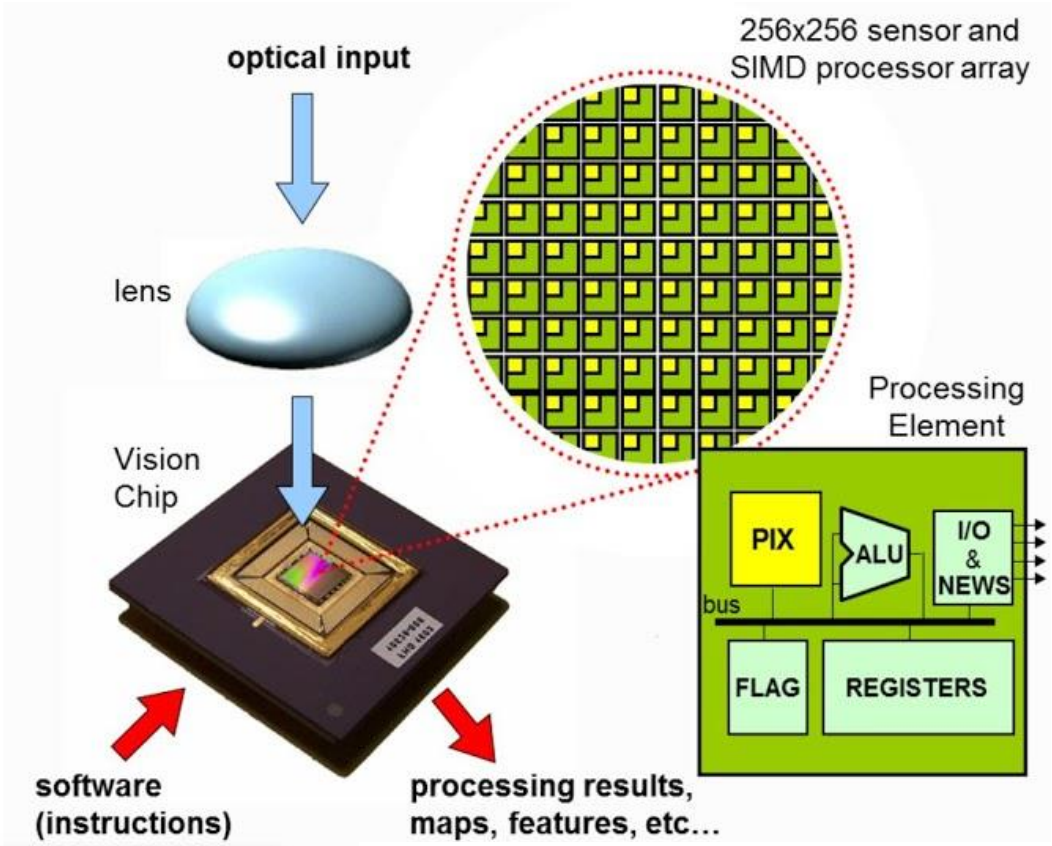
Scamp Host

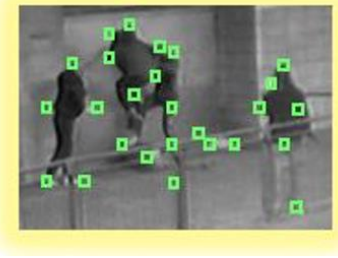


Scamp5 System

Recap – Pixel Processor Array

- each image pixel contains a processor
- processors are software-programmable
  - they execute code
- all computations are done **on the sensor**
  - in a pixel-parallel array
  - no sensor/processor/memory data movements
  - high speed & low power
- operation is *“frame based”*
  - but the frame rate can be **very high – many kHz**
- Only interesting data is transmitted off chip
  - results of computations,
  - e.g. 2D maps, global measures, address-events of points of interest,…





Example Basic Algorithms

Edge detector
execution time 5.8 μs

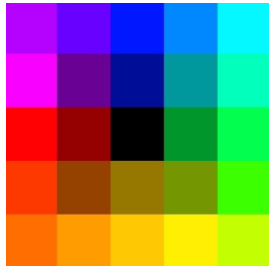
Power @ 20 fps
(excluding I/O) :
340 μW total,
i.e.5.2 nW/pixel

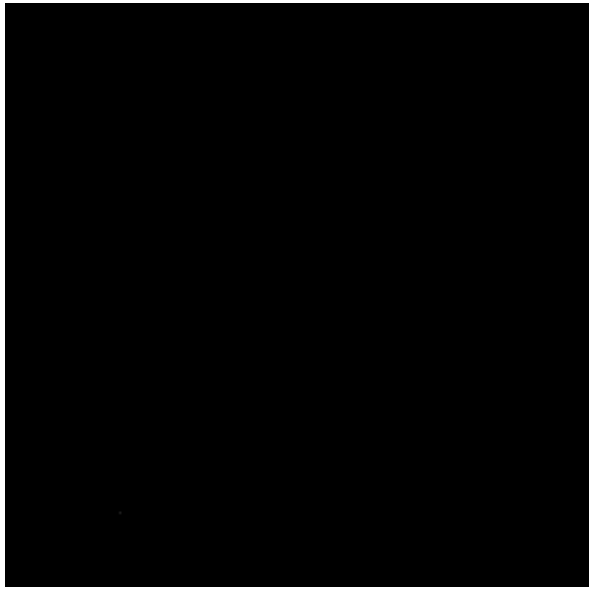
input image

result (SCAMP-5)

result (MATLAB)

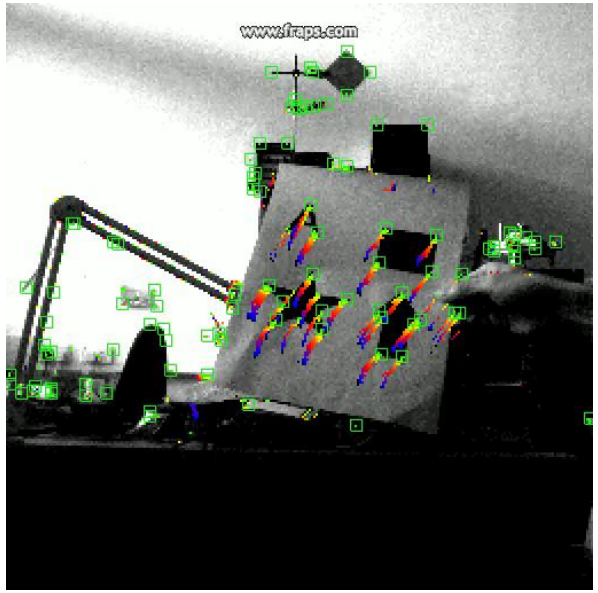
Example Basic Algorithms

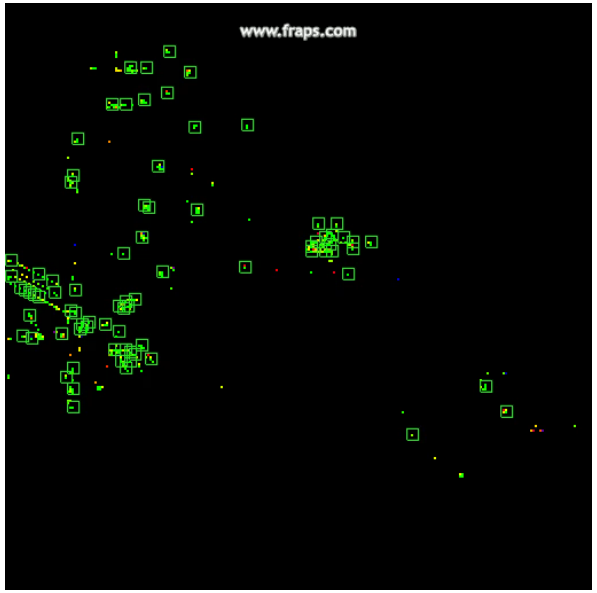




Optical Flow

- Block matching using L1-norm
- Determines best match
  - moving a 5x5 kernel of pixels
  - over a 5x5 grid
- Algorithm time ~0.4 ms

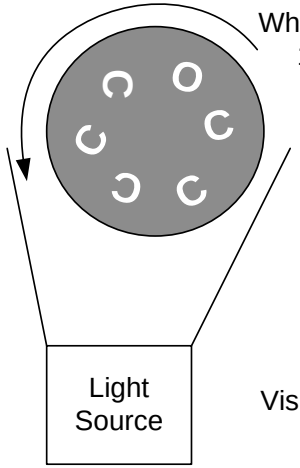


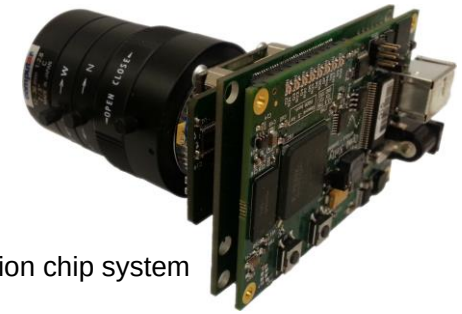


Corner detector

- FAST algorithm
- up to 480 fps with some gray-level image output (1 bit per frame)
- up to **2000 fps** when returning only **address-events** (corner coordinates)

Object identification and tracking at 100,000 frames per second





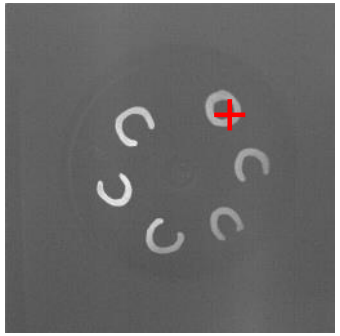
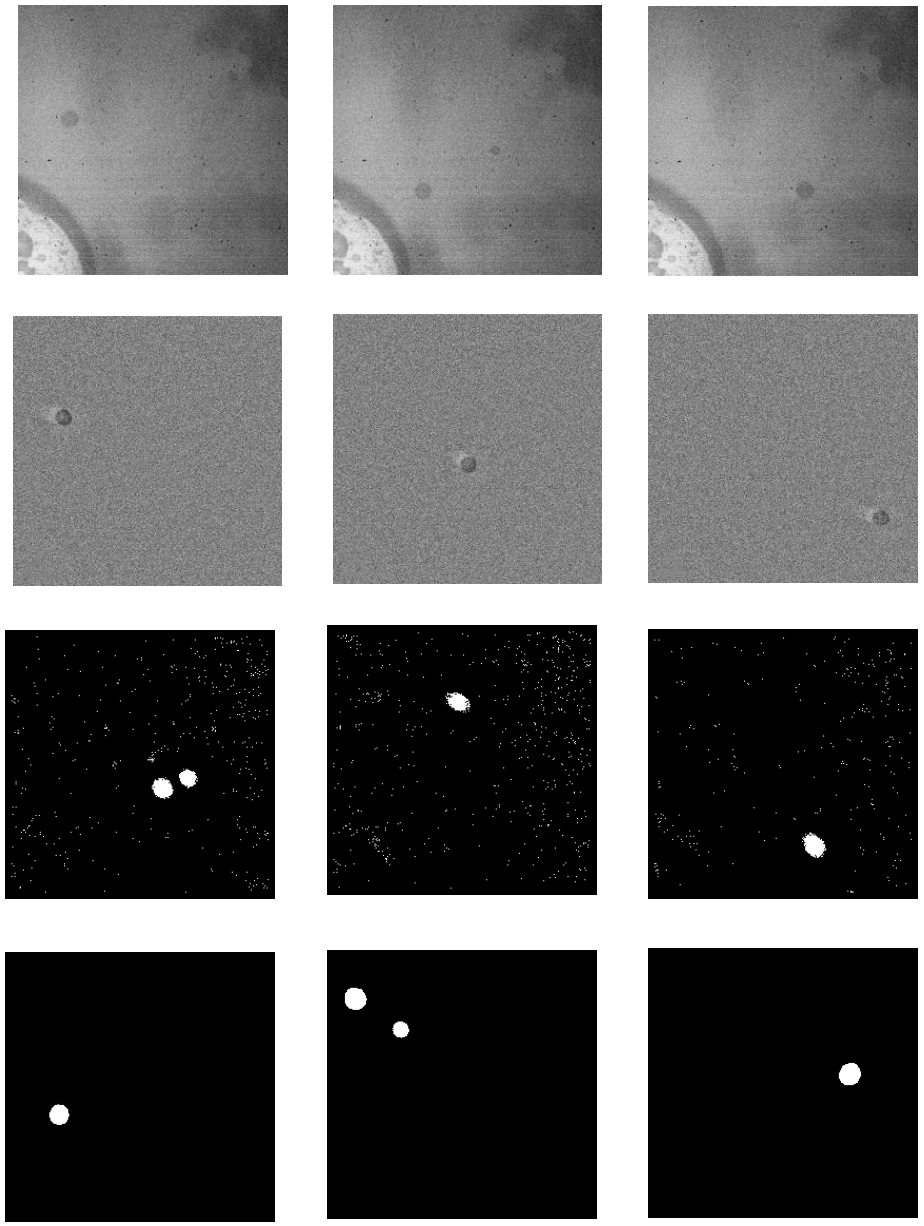


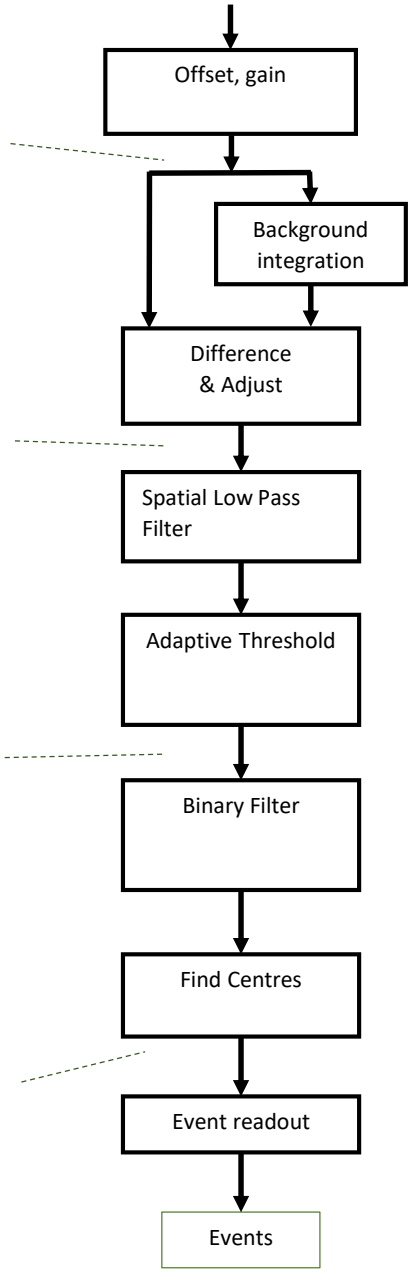
Image acquired by the sensor array

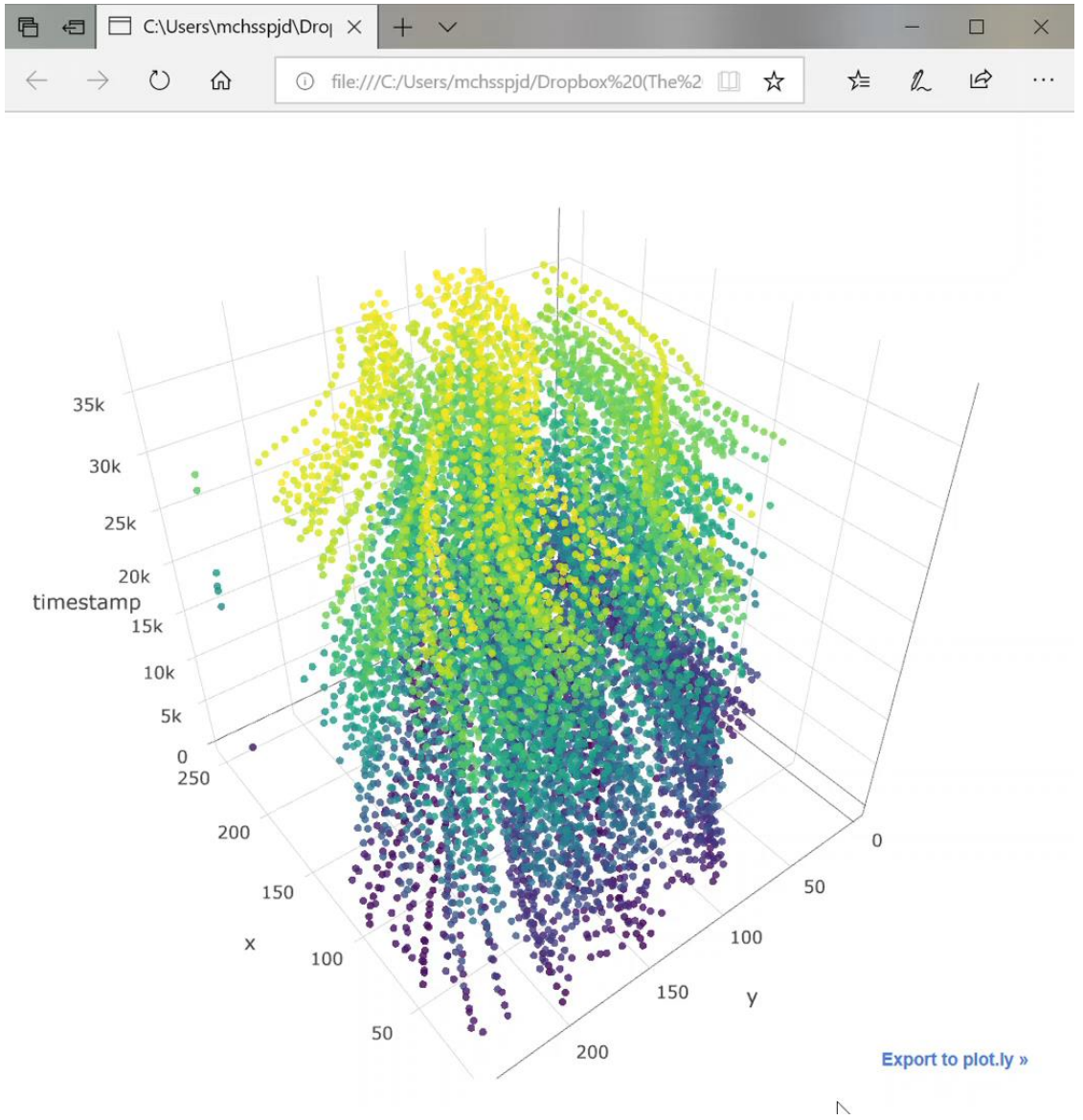
- Identify object of interest 'O' amongst distractors 'C'
- Track the object - extract {x,y} coordinates
- **10 us** timing resolution – 100,000 fps processing
- (6GBps raw image data rate upon which a non-trivial image processing algorithms need to be executed)

S.J.Carey et al, ISCAS 2014

Particle tracker - microfluidics



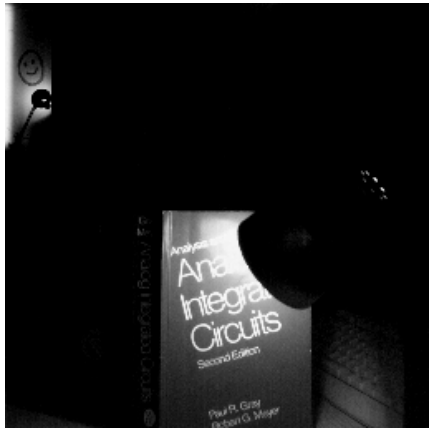




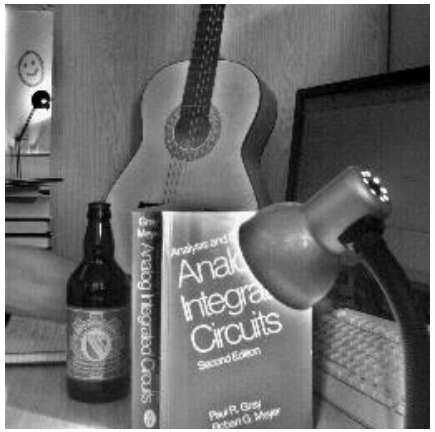
Example frames, running at 2000 fps

(Berthelon & Dudek, unpublished)

“Sub-frame” computation


short exposure


long exposure


HDR (exposure per pixel)

High Dynamic Range (HDR) Sensing

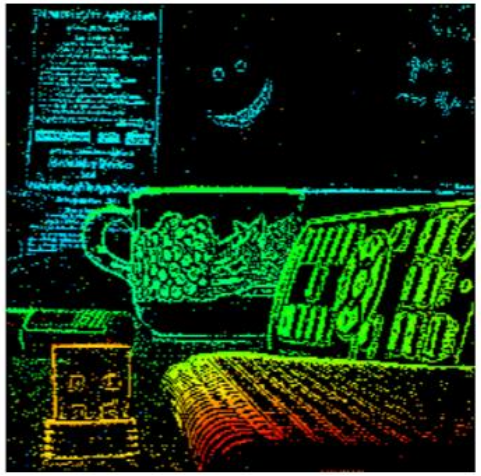
120dB dynamic range

- Tone-mapped, Composite frame combines 100s of exposure times
- Computations during frame integration - zero latency

J.Martel et al, ISCAS 2016


sub-frames from one sweep


“all in focus” image


depth map

Depth from Focus

- Liquid lens oscillating at 25Hz
- Depth determined by max local contract across focus sweep
- 32 levels @ 25 fps
- 800fps processing

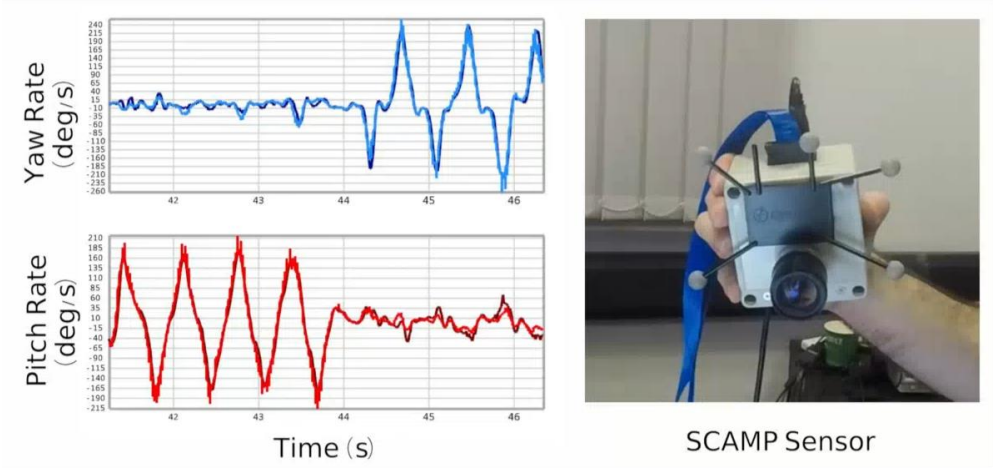
J.Martel et al, ISCAS 2017

Agile Robotics

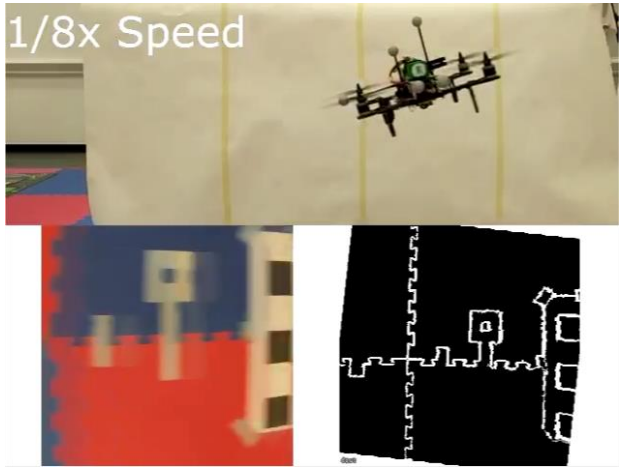
University of BRISTOL

MANCHESTER 1824
The University of Manchester

Visual odometry

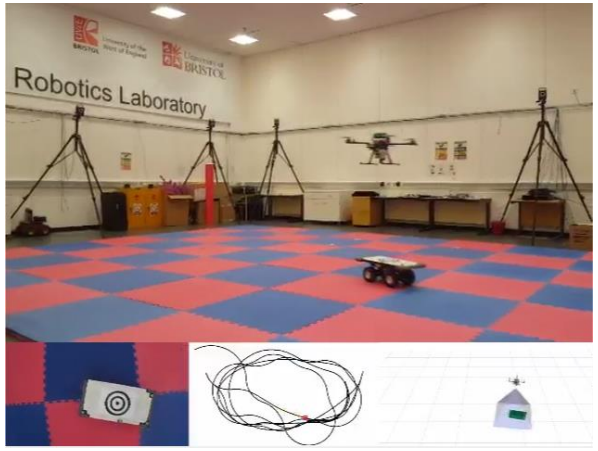


(Bose et. al ICCV 2017)



(Greatwood et al. IROS 2018)

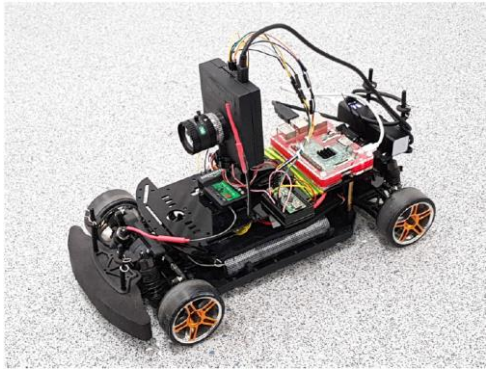
Visual target tracking



(Greatwood et al. IROS 2017)

High-speed slalom

(Liu et al. IROS 2019,submitted)



see videos at https://sites.google.com/view/project-agile

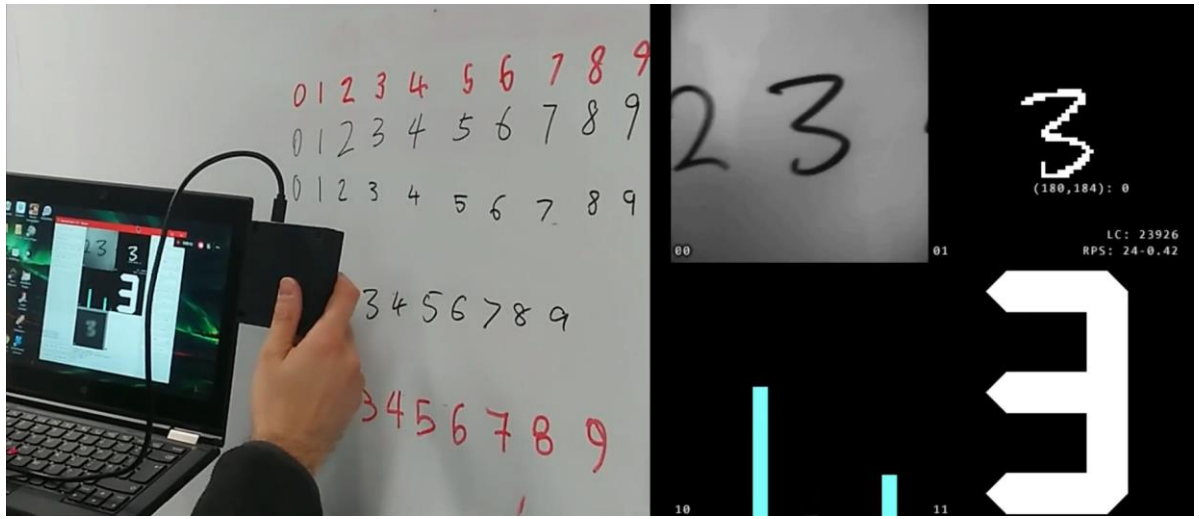
Convolutional Neural Networks (CNNs)

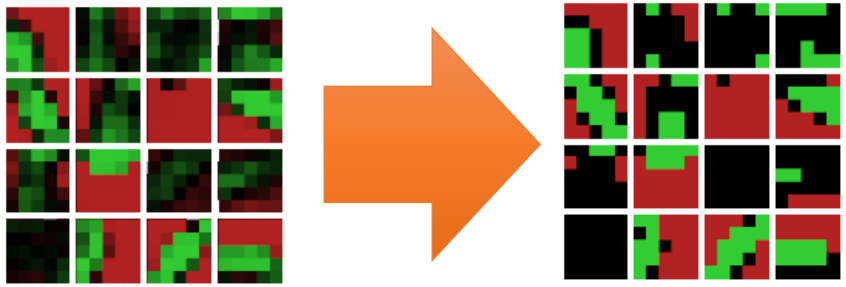
University of BRISTOL

MANCHESTER 1824
The University of Manchester

See our demo!

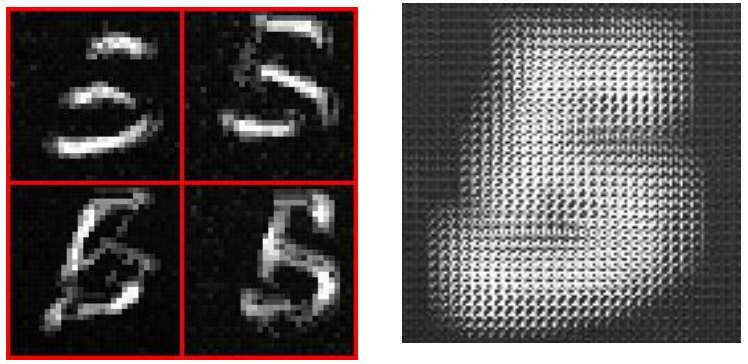
L.Bose et al., CVPR 2019





Ternary Convolution Layers

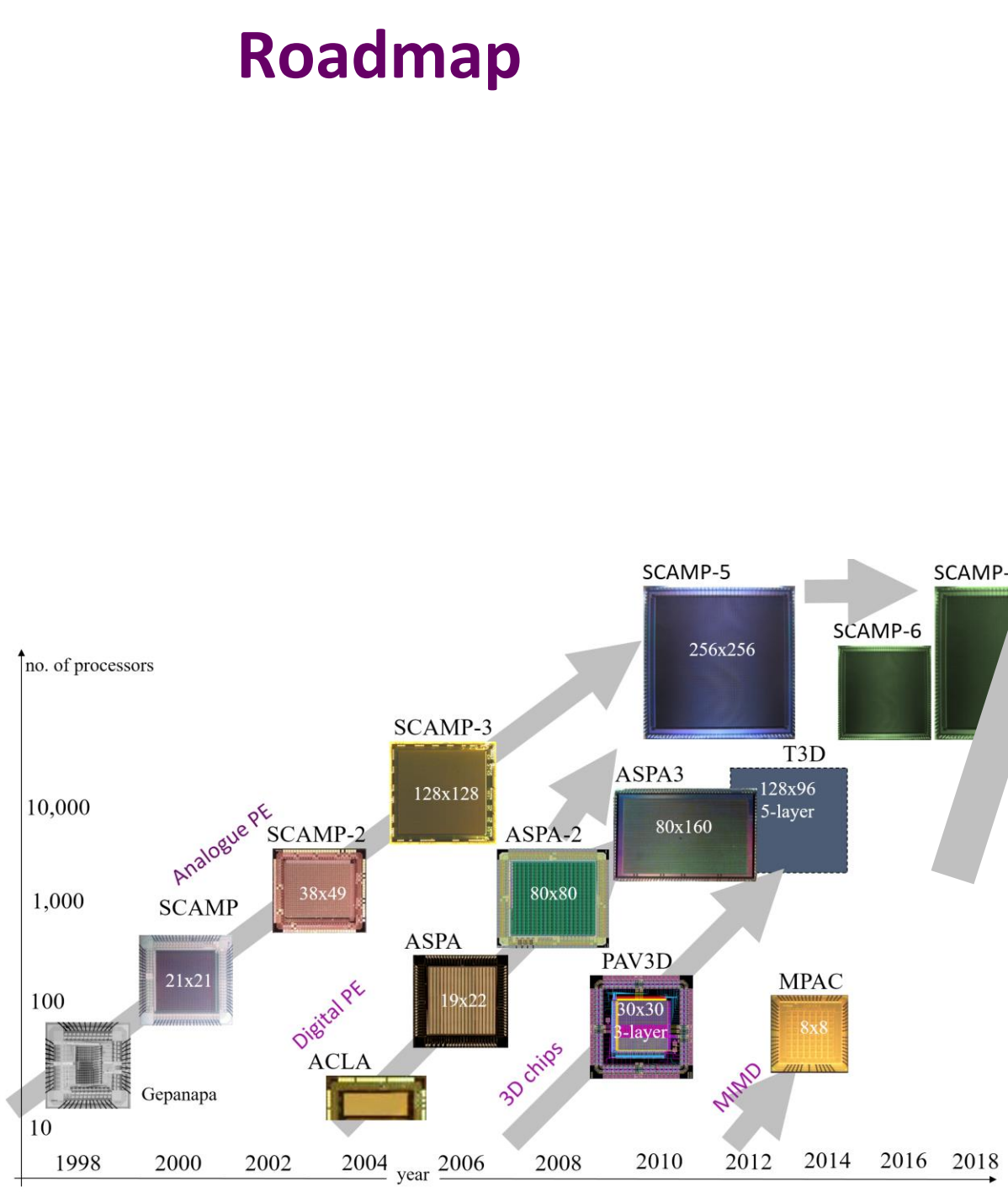
computed using parallel image operations: addition(+1), subtraction(-1) and image shifting.



Max pooling Layers

multiple images are stored within a single array using a checker boarding scheme

Roadmap



28 nm analog
14 nm digital
chip stacking

This is in
180 nm
CMOS…

Technology
state-of-the-art
ca. 1998

What Next…

What Next…

Get in touch if you would like to use SCAMP-5 in your application !

MANCHESTER 1824
The University of Manchester

Acknowledgements

People:

Manchester Team: Steve Carey, Jianing Chen, Alex Lopich, David Barr, …

Collaborators: Laurie Bose, Walterio Mayol-Cuevas, Yanan Liu, Colin Greatwood, Tom Richardson, Lorenz Muller, Julien Martel, Xavier Berthelon, …

+ others

Funding:

Project AGILE:

EPSRC (Engineering and Physical Sciences Research Council)

previous projects:

EPSRC, DSTL (MoD)

© 2019 by Piotr Dudek, The University of Manchester – pdudek@manchester.ac.uk