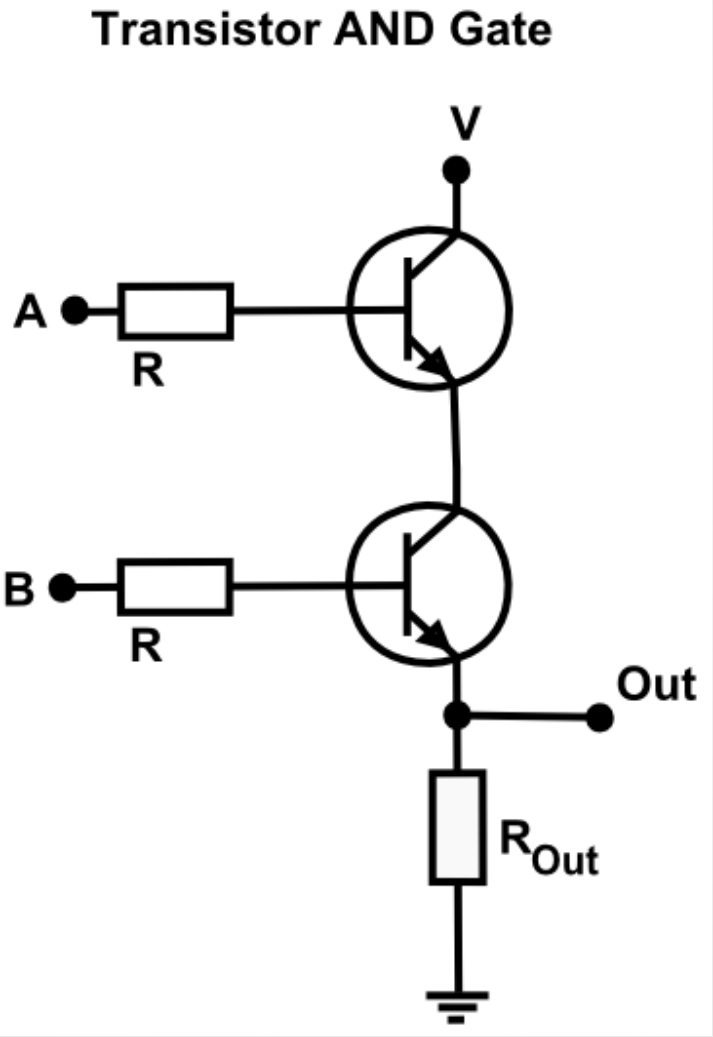
Synchronous Message Exchange

Concurrent and Distributed Systems

2018-12-18 Niels Bohr Institute

Kenneth Skovhede

Binary logic in hardware



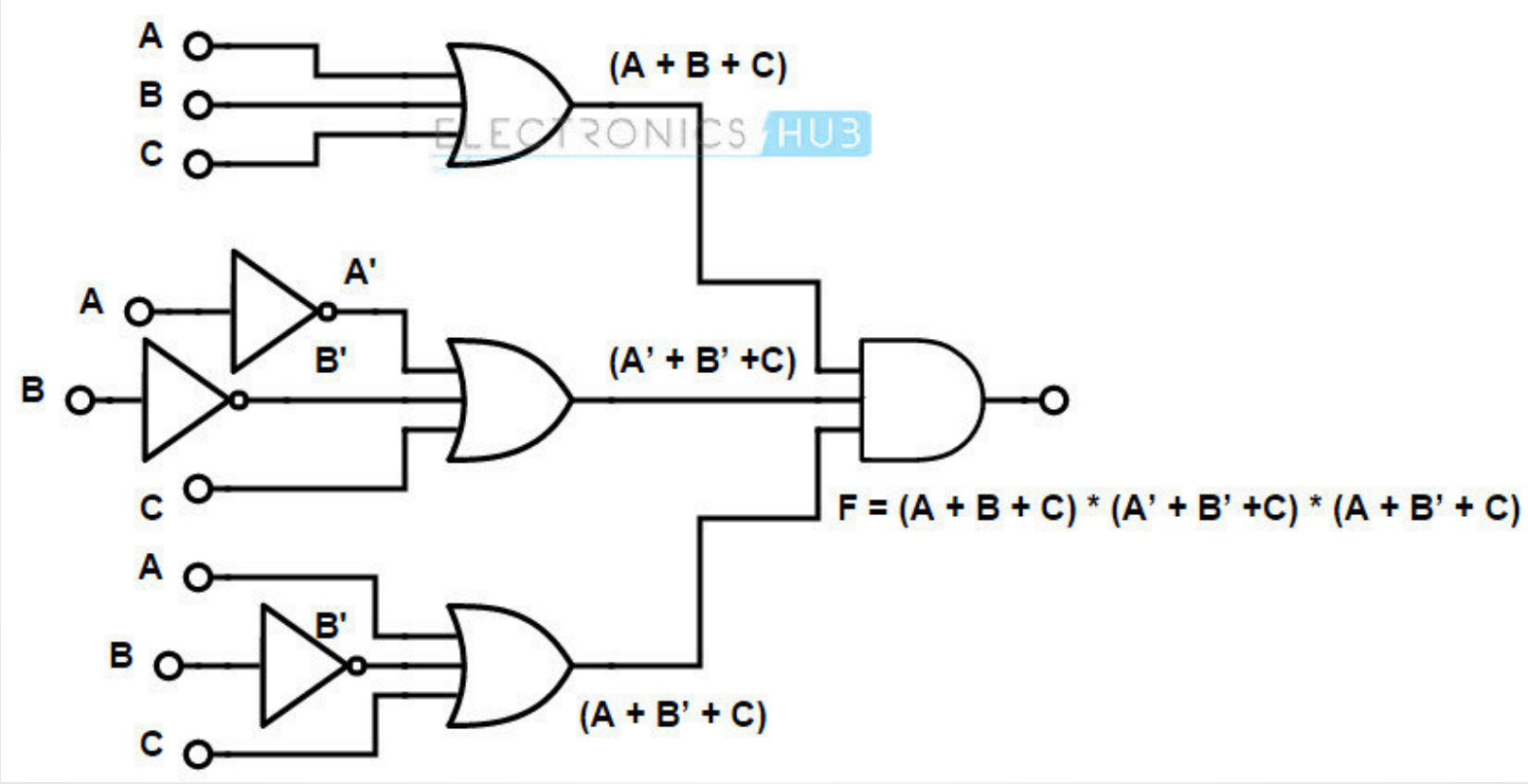
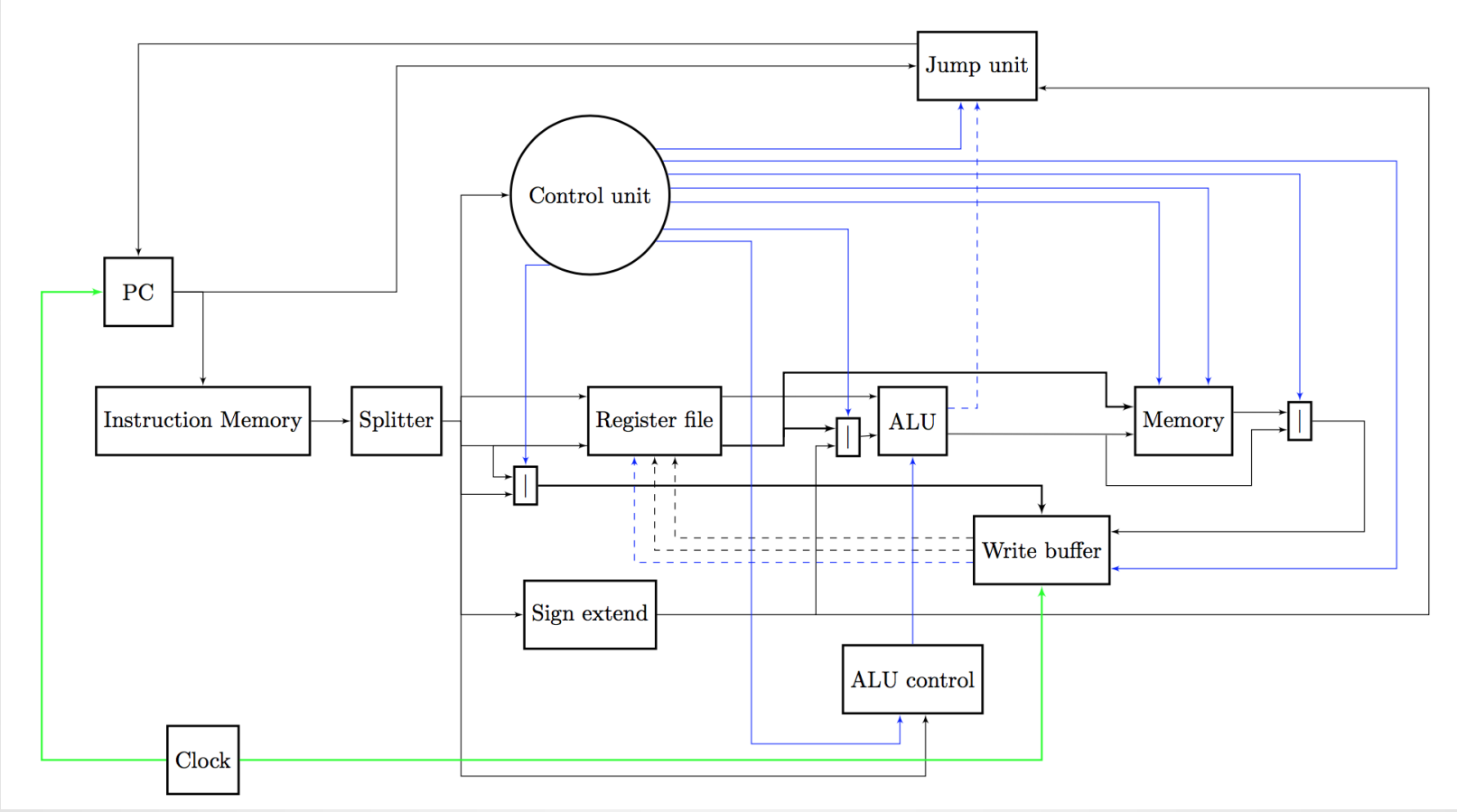
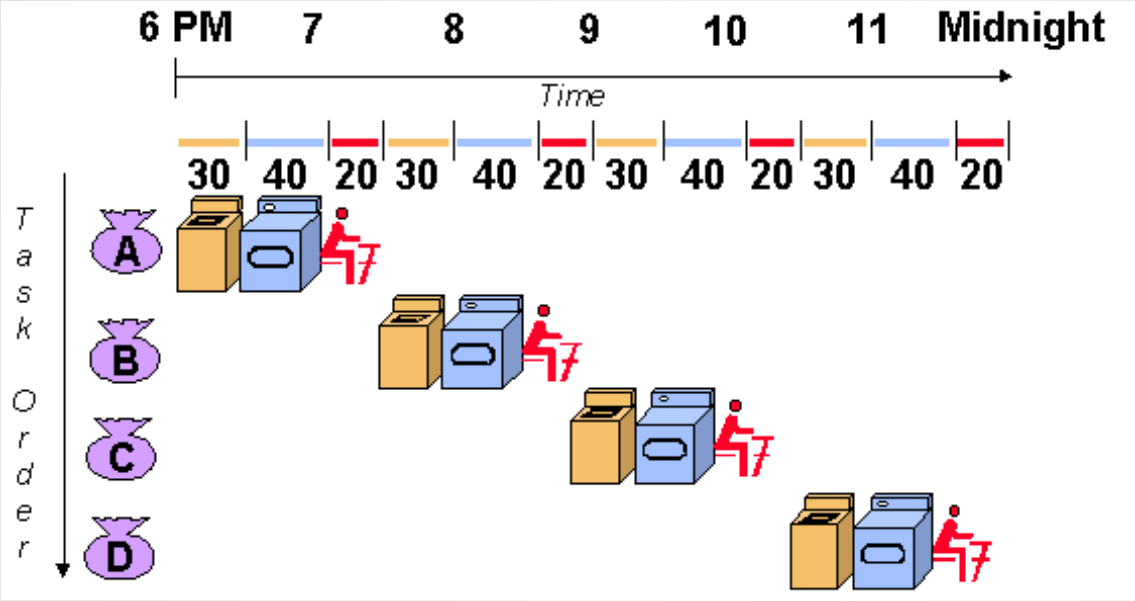
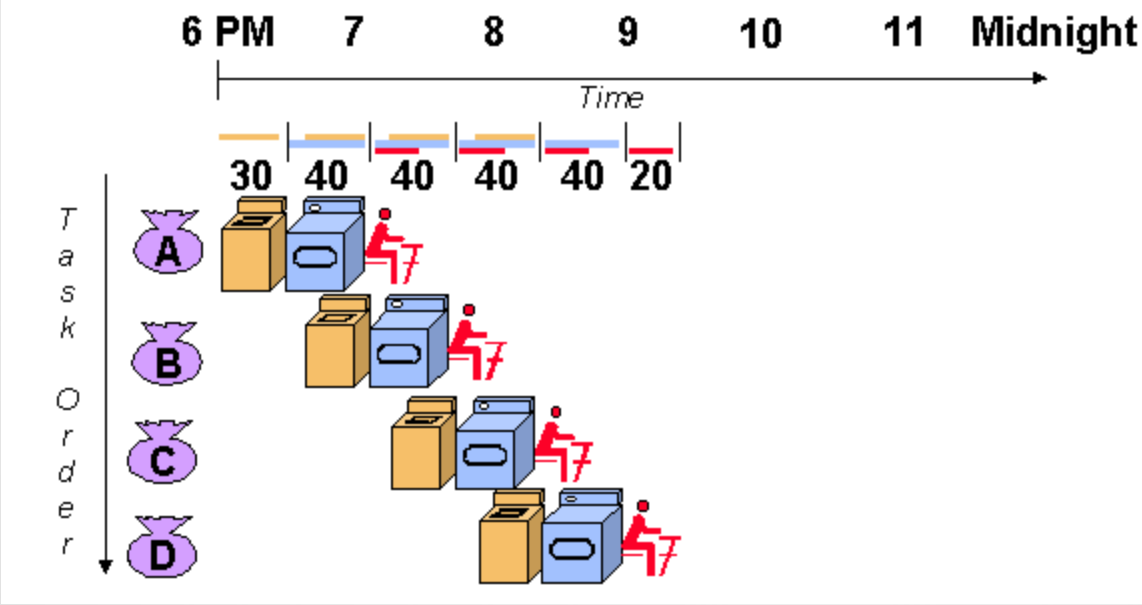


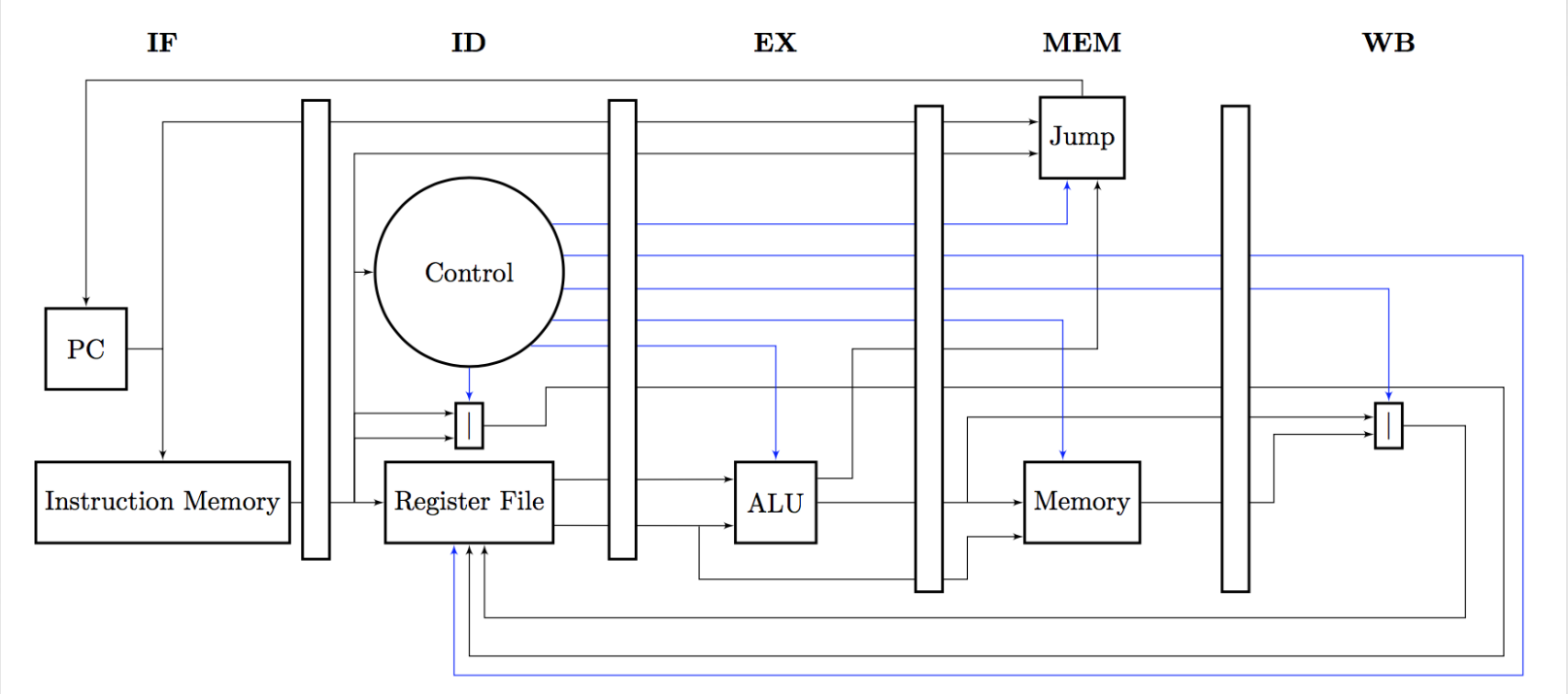
Image from electronicshub.com


Jump unit
Control unit
PC
Instruction Memory
Splitter
Register file
ALU
Memory
Write buffer
Sign extend
ALU control
Clock





https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/pipelining/index.html

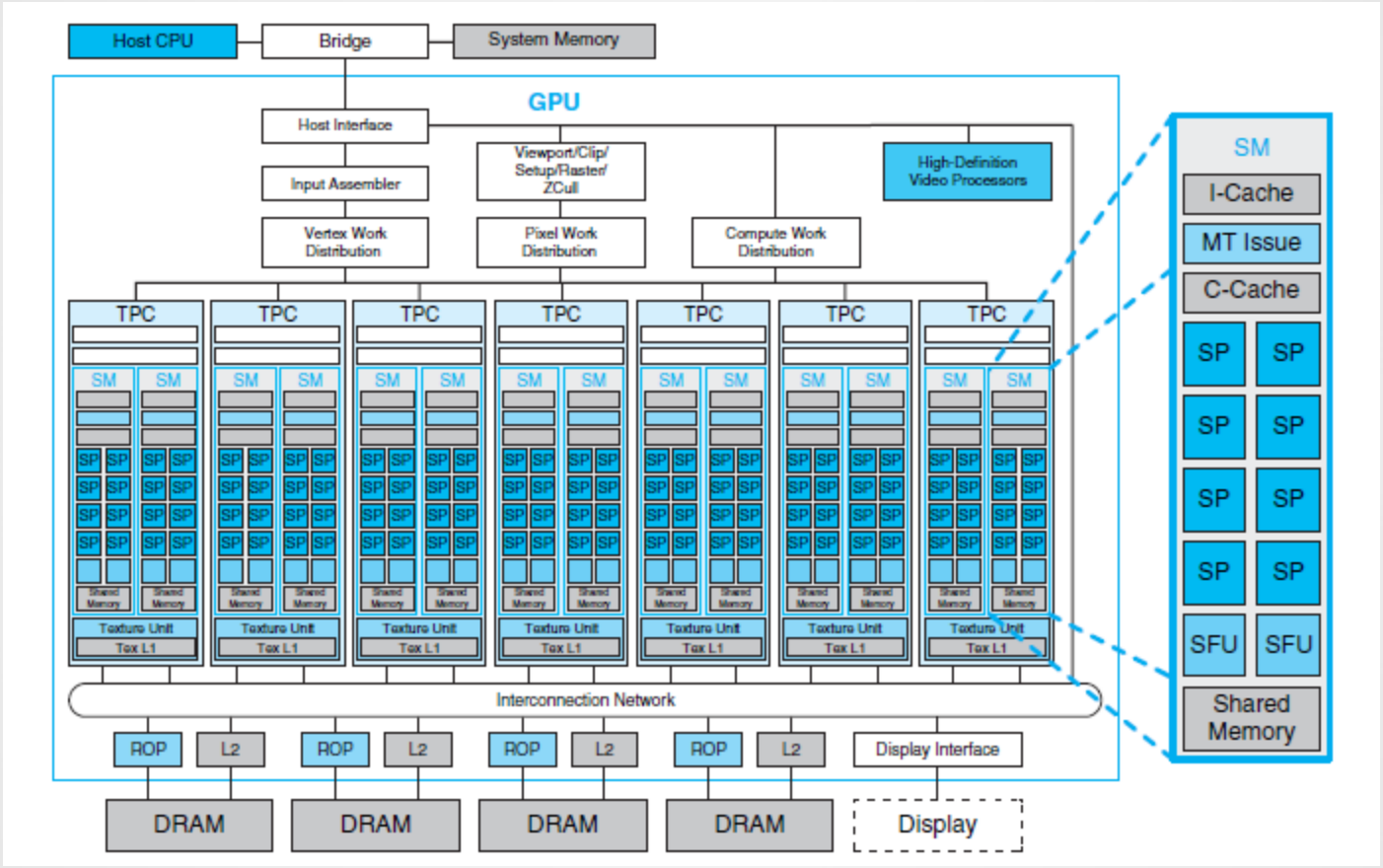


c = a + b

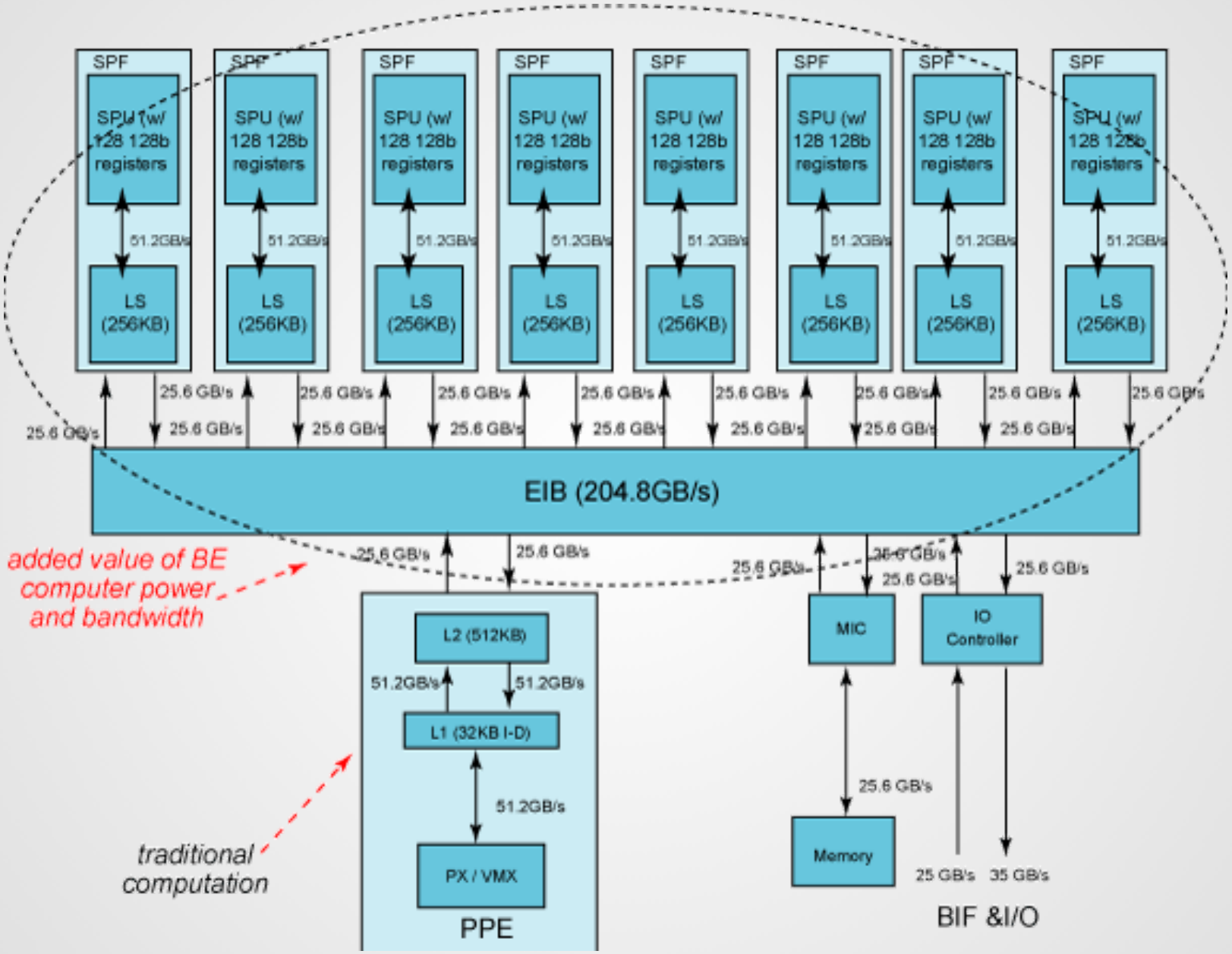
d = a * b

c = a + b

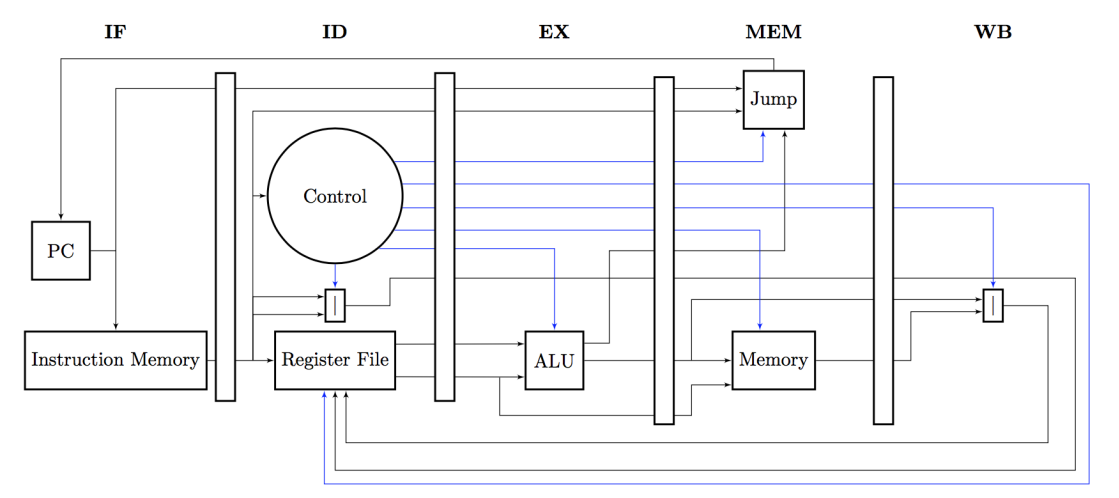
d = c * b



https://tatourian.blog/2013/09/03/nvidia-gpu-architecture-cuda-programming-environment/


SPF
SPU (w/ 128 128b registers
51.2GB/s
LS (256KB)
25.6 GB/s
EIB (204.8GB/s)
added value of BE computer power and bandwidth
L2 (512KB)
L1 (32KB I-D)
PX / VMX
PPE
traditional computation
MIC
IO Controller
Memory
25 GB/s
35 GB/s
BIF &I/O

Running a program




```
function odd_or_even(int a, int b) {
  if ((a + b) % 2 == 0)
    return 1;
  else
    return 0;
}
```

```
lw   $2, addr_a
lw   $3, addr_b
add  $4, $2, $3
li   $5, 2
mod  $6, $4, $5
beq  $6, $0, even
li   $2, 0
b    done
even:
li   $2, 1
done:
```

A number game

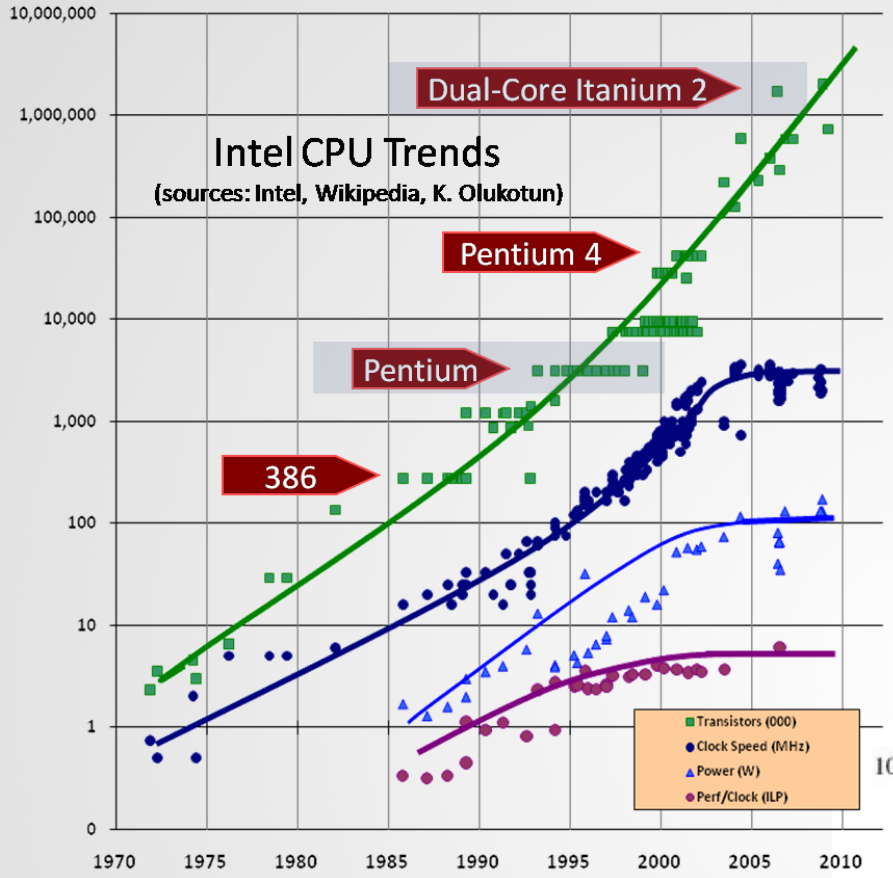
Speed of light: 299.792.458 m/s

Speed of electricity: ~150.000.000 m/s

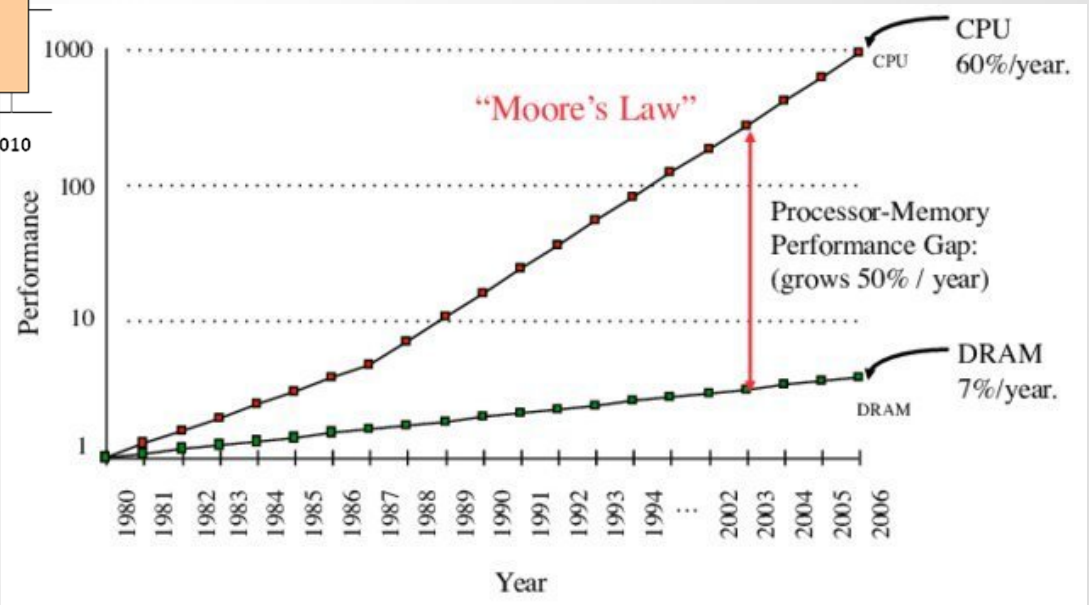
A regular CPU is: ~1 cm across

Max clock rate : ~150.000.000 * 100 ~= 15GHz

Heat and power limits it to around 5GHz

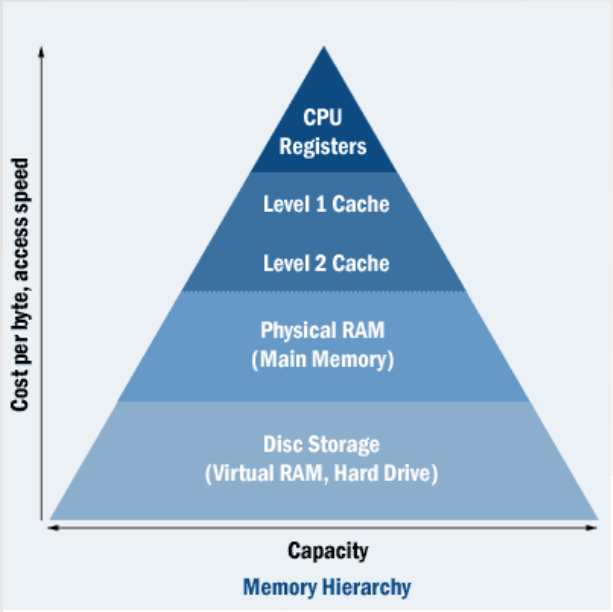


http://www.gotw.ca

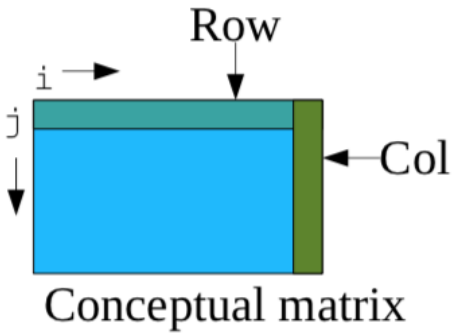


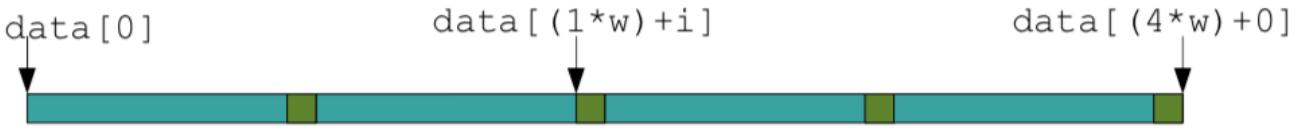
Source: High Performance by Exploiting Information Locality through Reverse Computing

UMA vs NUMA

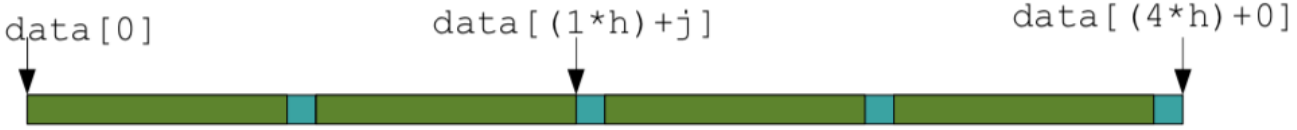



```
float sum(float* data, size_t w, size_t h)
{
    float result = 0.0;
    for(size_t j = 0; j < h; j++)
        for(size_t i = 0; i < w; i++)
            result += data[(j * w) + i];
    return result;
}
```

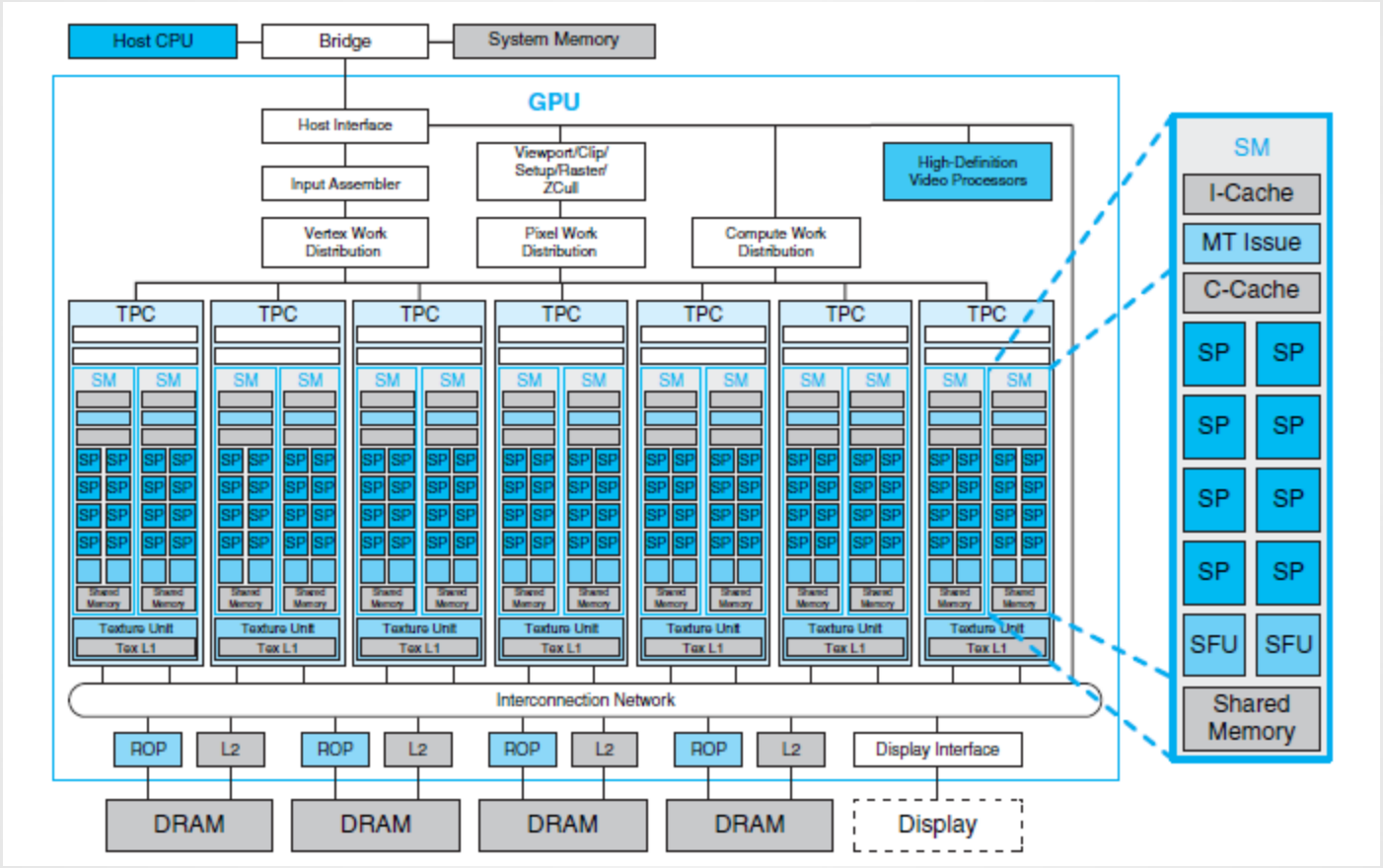




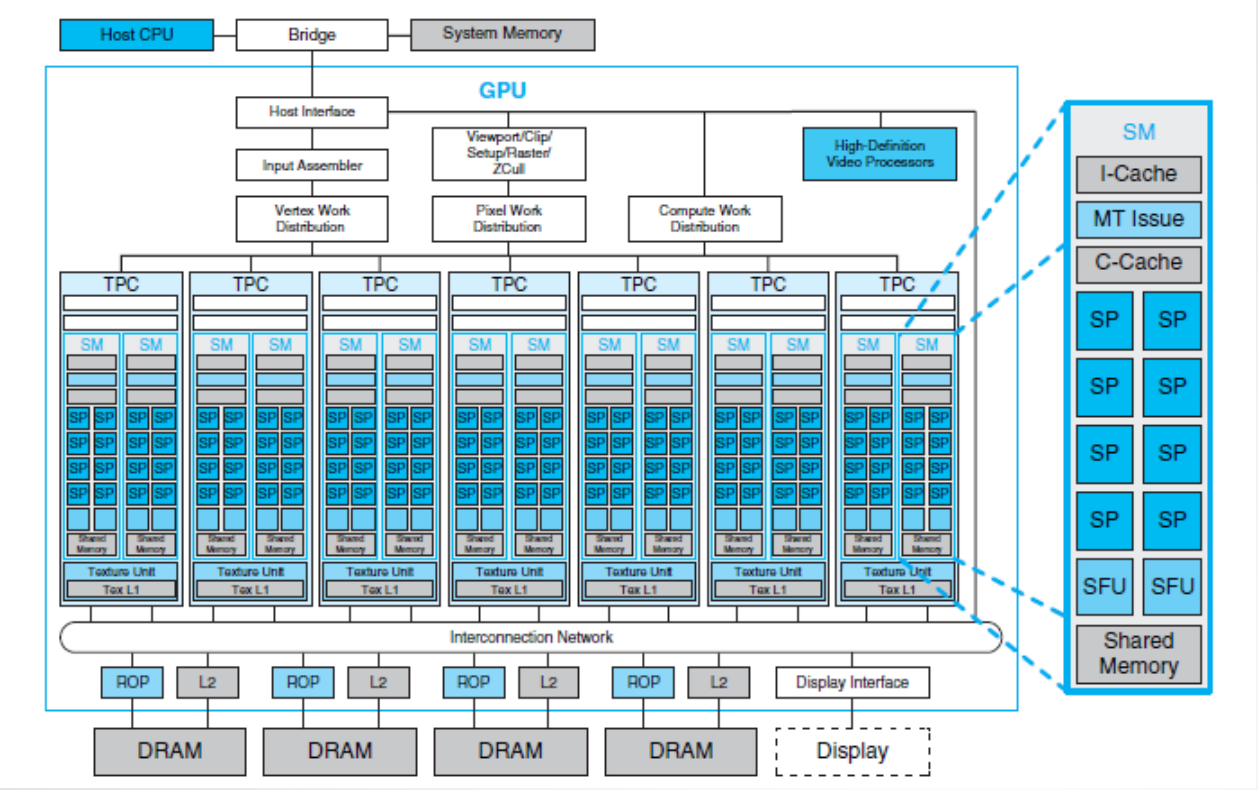
row major layout



column major layout


Host CPU
Bridge
System Memory
GPU
Host Interface
Input Assembler
Viewport/Clip/
Setup/Raster/
ZCull
High-Definition
Video Processors
Vertex Work
Distribution
Pixel Work
Distribution
Compute Work
Distribution
TPC
SM
SP
Shared
Memory
Texture Unit
Tex L1
Interconnection Network
ROP
L2
Display Interface
DRAM
Display
SM
I-Cache
MT Issue
C-Cache
SP
SFU
Shared
Memory

Nvidia




```
function odd_or_even(int a, int b) {
  if ((a + b) % 2 == 0)
    return 1;
  else
    return 0;
}
```

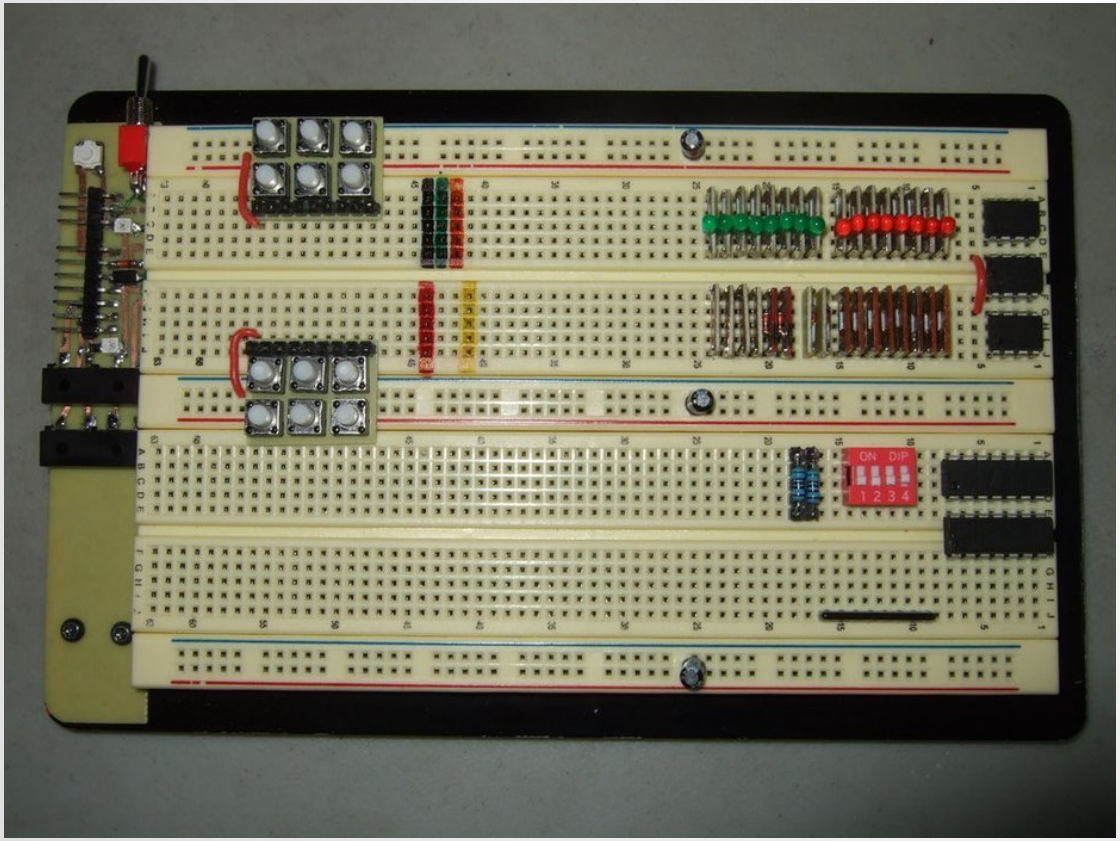
```
lw    $2, addr_a
lw    $3, addr_b
add   $4, $2, $3
li    $5, 2
mod   $6, $4, $5
beq   $6, $0, even
li    $2, 0
b     done
even:
li    $2, 1
done:
```

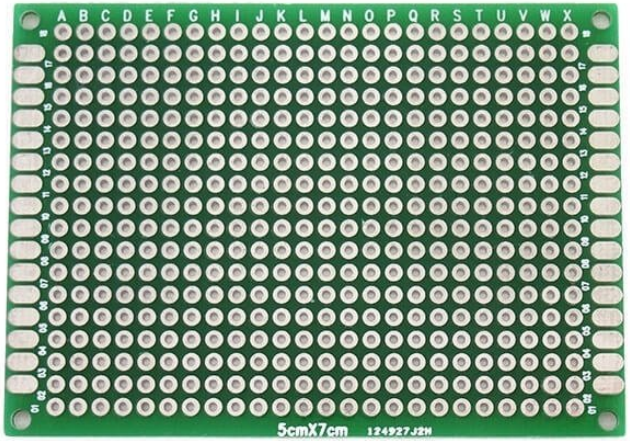
FPGA and ASIC

Field **P**rogrammable **G**ate **A**rray

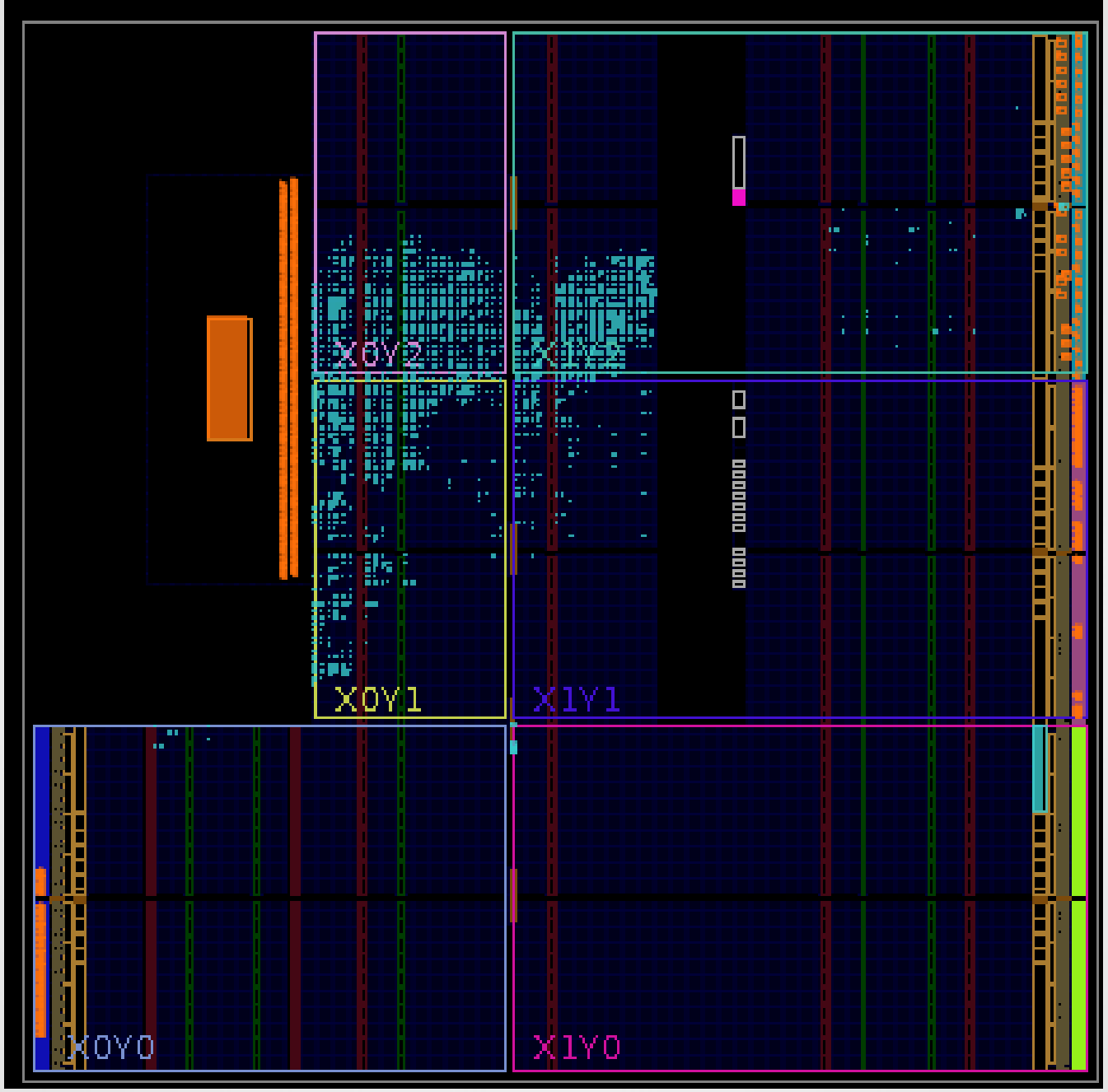
Application **S**pecific **I**ntegrated **C**ircuit

FPGA

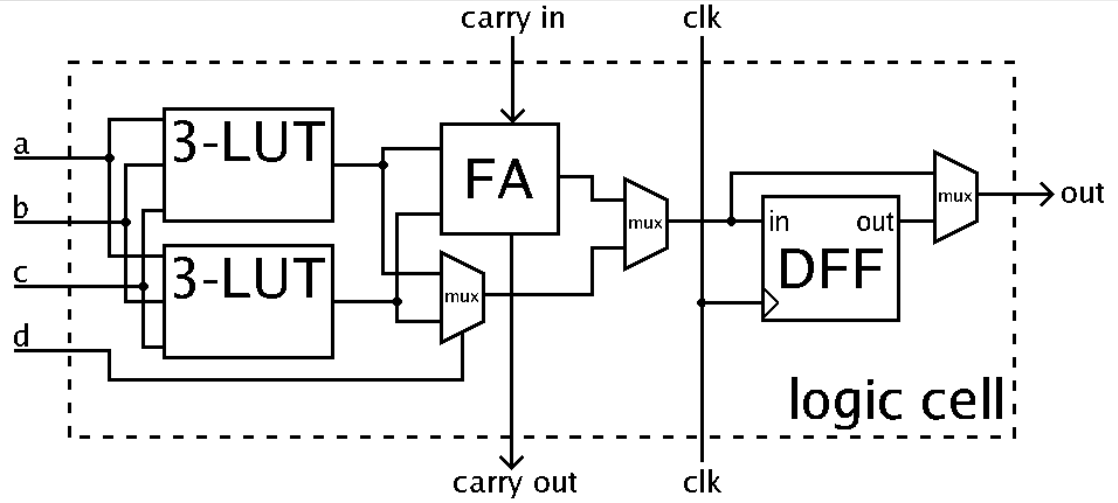




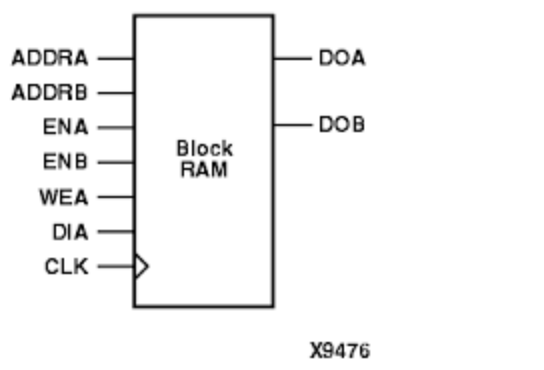
https://www.instructables.com/id/My-Top-Ten-Most-Useful-Breadboard-Tips-and-Tricks/



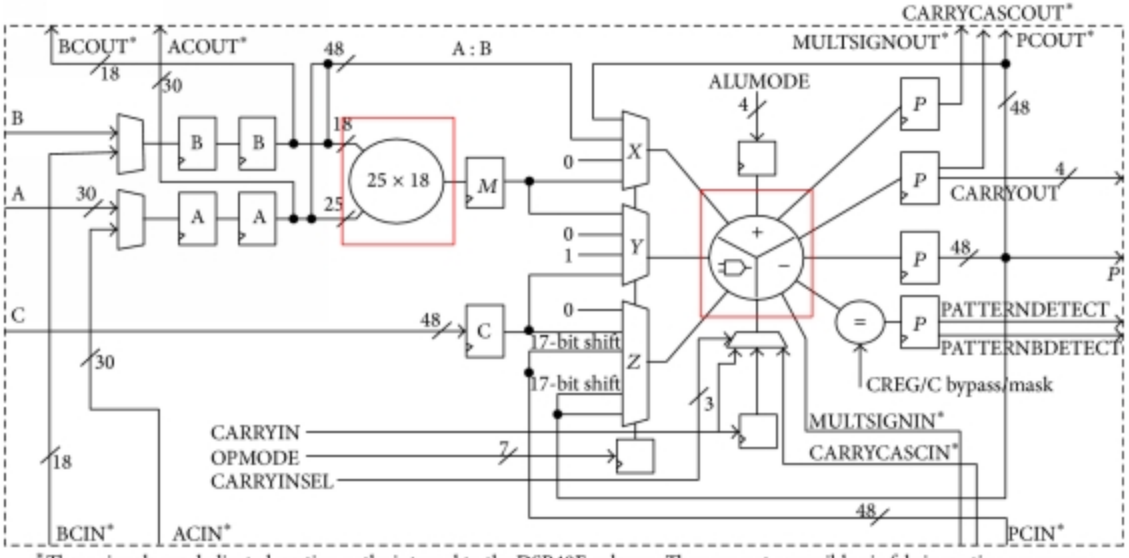
https://www.parallella.org/2015/03/23/new-parallella-elink-fpga-design-project-now-available-in-vivado/



Wikipedia



Xilinx



Xilinx

When to use CPU, GPU, FPGA, and ASIC?

GPGPU

Pros:

- Fast memory access
- Low price pr flops
- Massive throughput

Cons:

- Needs structured data
- Only uses SIMD
- PCI transfer is bottleneck
- Needs host PC

CPU

Pros:

- Low unit price
- Flexible (runs many programs)
- Massive amount of tools

Cons:

- "Low" throughput
- Average price pr flop
- "Unpredictable" throughput

FPGA

Pros:

- Very flexible
- Very reliable
- Predictable throughput
- Predictable latency
- Low power usage

Cons:

- Higher price pr flop
- No dynamic resourses
- Hard to program

ASIC

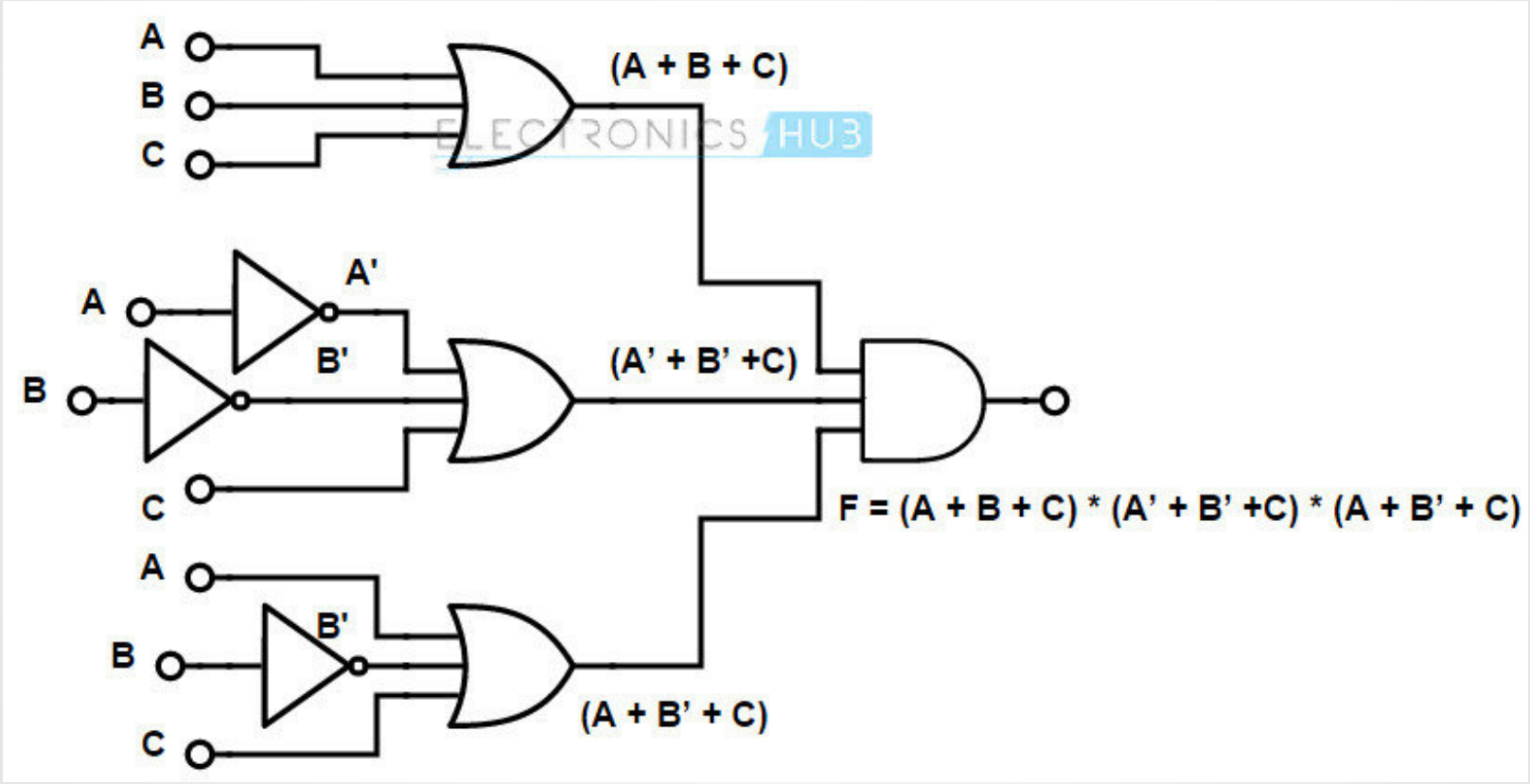
Pros:

- Made-to-order
- Extremely reliable
- Predictable high throughput
- Predictable latency
- Very low power usage

Cons:

- Hefty price tag
- Need to make 100.000's of units
- Very hard to program and test

From Software to Hardware


A
B
C
(A + B + C)
ELECTRONICS HUB
A'
A
B'
B
(A' + B' +C)
C
F = (A + B + C) * (A' + B' +C) * (A + B' + C)
A
B'
B
(A + B' + C)
C

Hardware Description Languages

VHDL (VLSI HDL, Very high speed integrated circuit Hardware Description Language)

(System) Verilog

System C

- Low level
- Event based simulation
- From the 1980's

HLS

High Level Synthesis

```
int temp;
for(int i2=0; i2<=length; i2++)
{
  for(int j=0; j<length; j++)
  {
    if(array[j]>array[j+1])
    {
      temp=array[j];
      array[j]=array[j+1];
      array[j+1]=temp;
    }
  }
}
```

Bubble sort in C++

Static loop bounds

Map array to HW

Loop dependencies

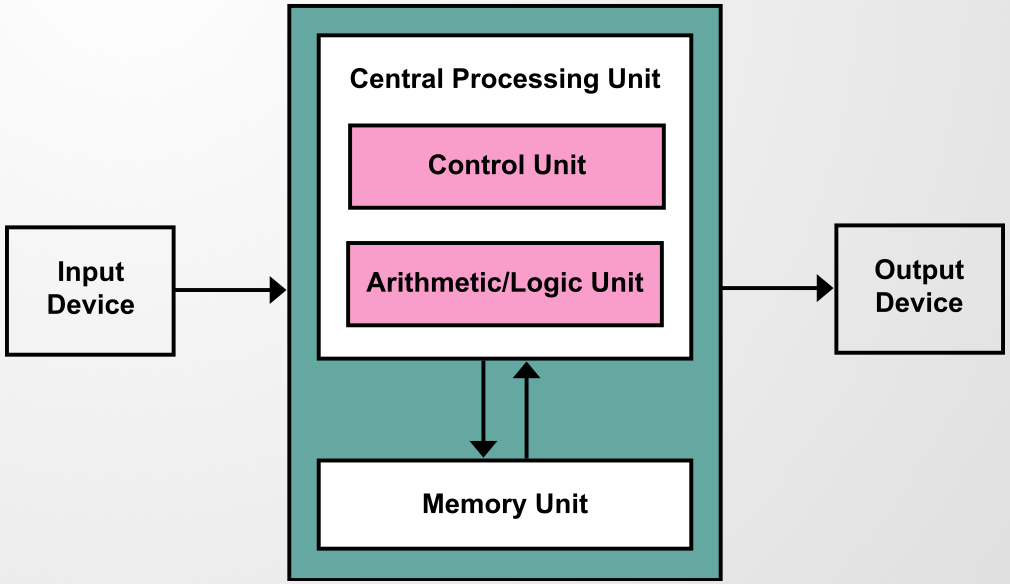
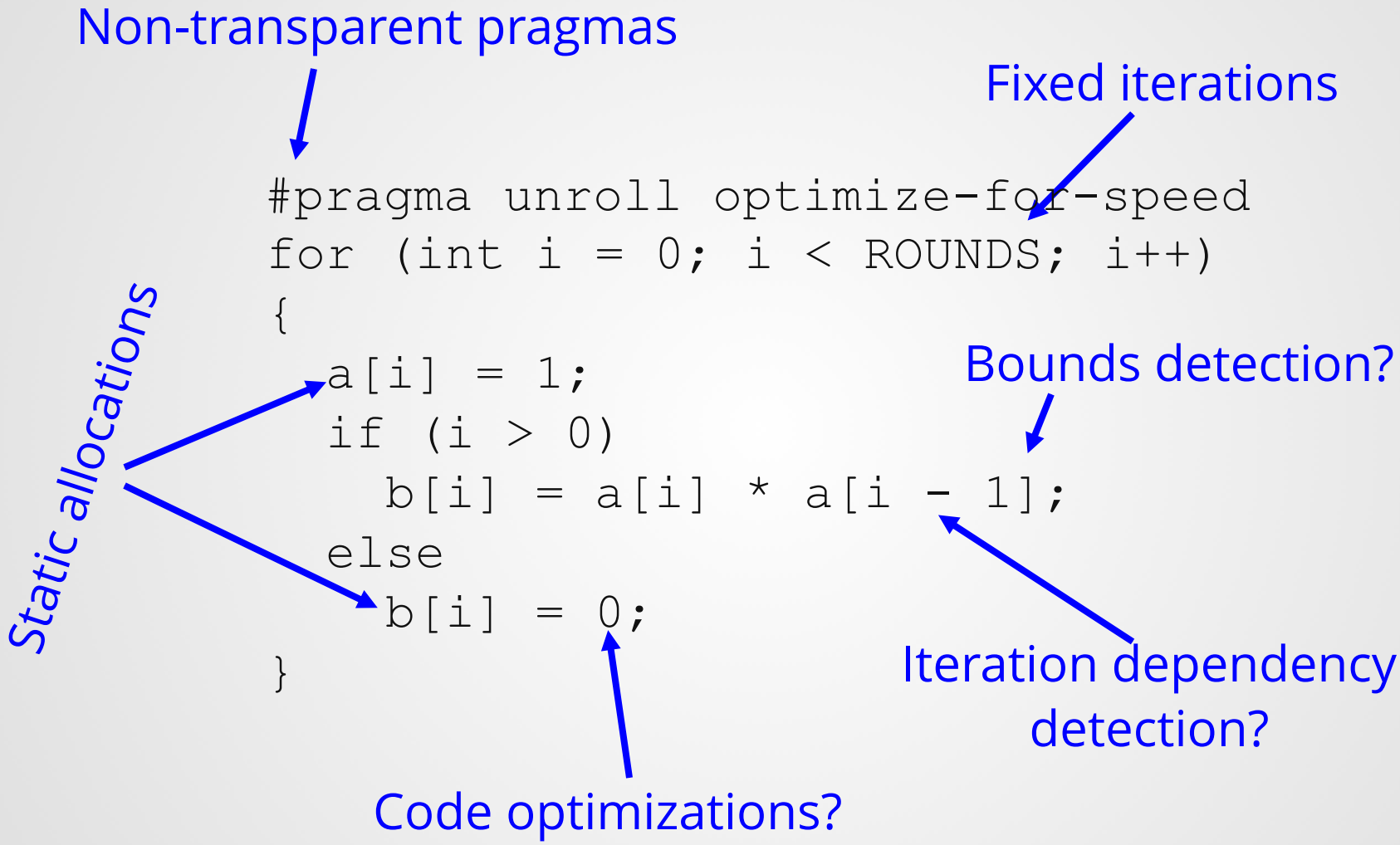


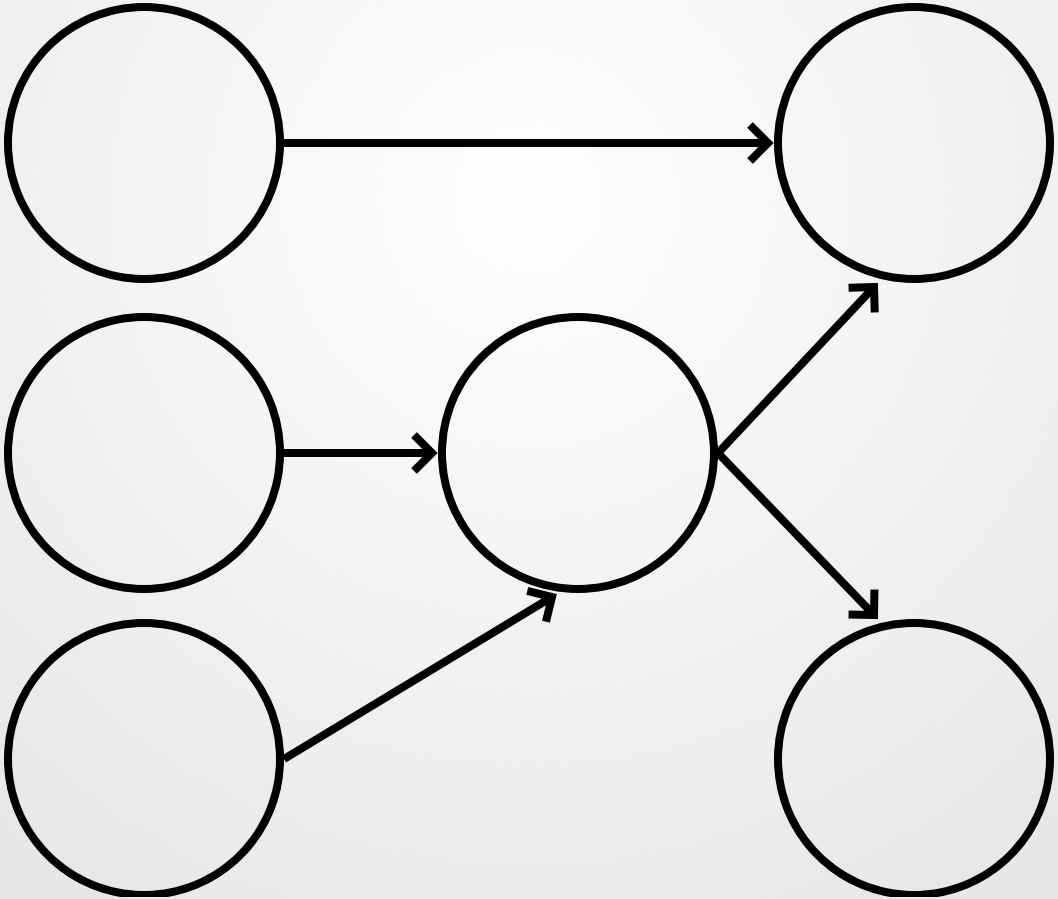
Image from Wikipedia

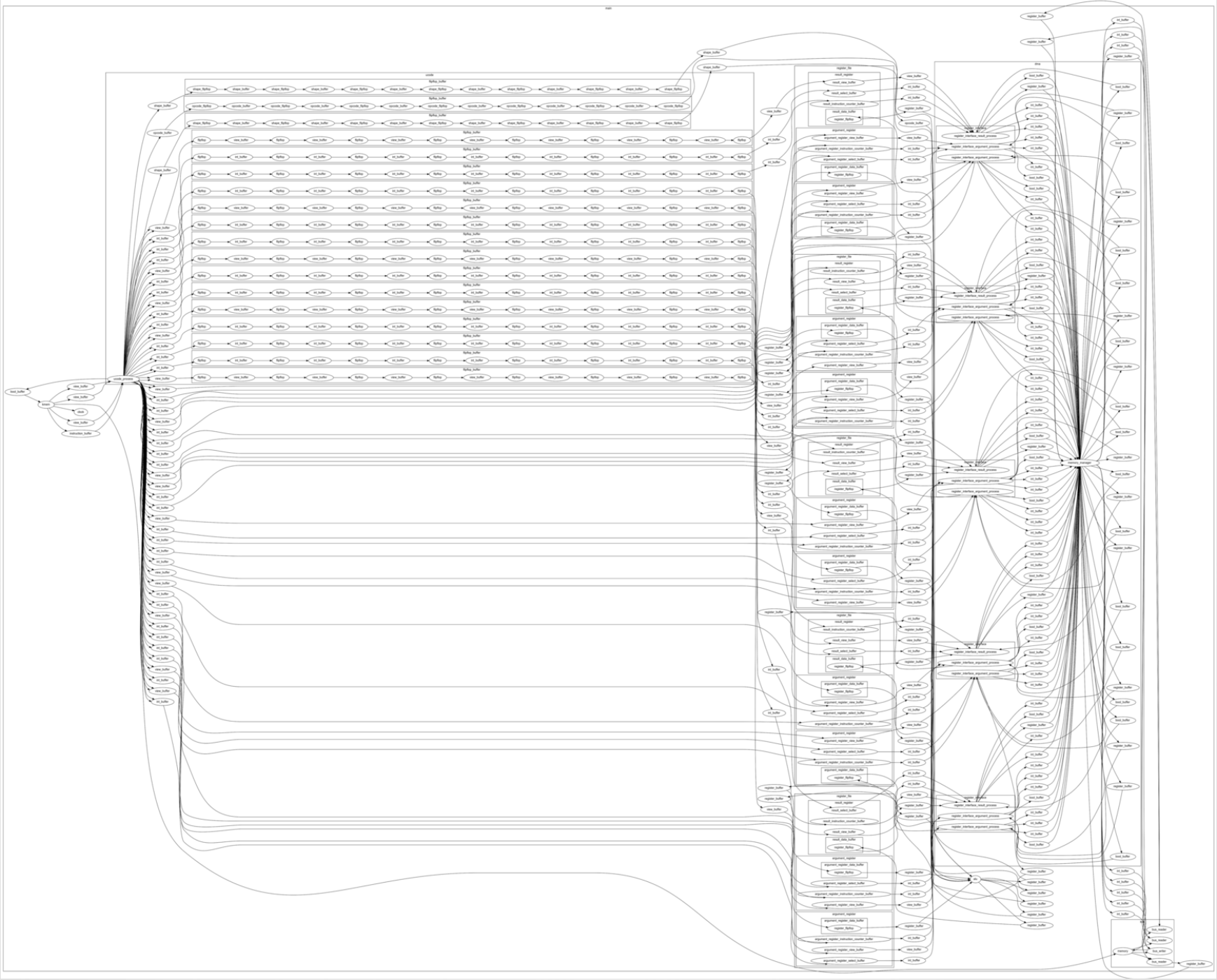

Non-transparent pragmas
Fixed iterations
#pragma unroll optimize-for-speed
for (int i = 0; i < ROUNDS; i++)
{
a[i] = 1;
if (i > 0)
b[i] = a[i] * a[i - 1];
else
b[i] = 0;
}
Static allocations
Bounds detection?
Iteration dependency detection?
Code optimizations?

SME: Synchronous Message Exchange

Basic idea:

Use CSP to model massively concurrent hardware




main
ocode
flpfkp_buffer
shape_flpfkp
shape_buffer
opcode_flpfkp
opcode_buffer
flpfkp
view_buffer
int_buffer
register_buffer
boot_buffer
instruction_buffer
tokenizer
clock
register_file
result_register
result_instruction_counter_buffer
result_view_buffer
result_select_buffer
result_data_buffer
argument_register
argument_register_data_buffer
argument_register_select_buffer
argument_register_instruction_counter_buffer
argument_register_view_buffer
register_interface
register_interface_result_process
register_interface_argument_process
alu
dma
memory_manager
bus
bus_reader
bus_writer
memory

SME

Components

Process

Equivalent to CSP process

Bus

A multi-value channel

(or a collection of channels)

SME

Processes

Processes are

- Sequential
- Shared nothing
- Keeps state
- Invoked when inputs are ready
- Invoked exactly once pr. clock tick

SME

Communication

Communication

- Via channel-like signals
- Signals have propagation delays
- Signals are grouped into busses

SME

From CSP

- Sequential programs
- Naturally concurrent
- Transparent/logic hardware mapping
- Compositional

SME

Not from CSP

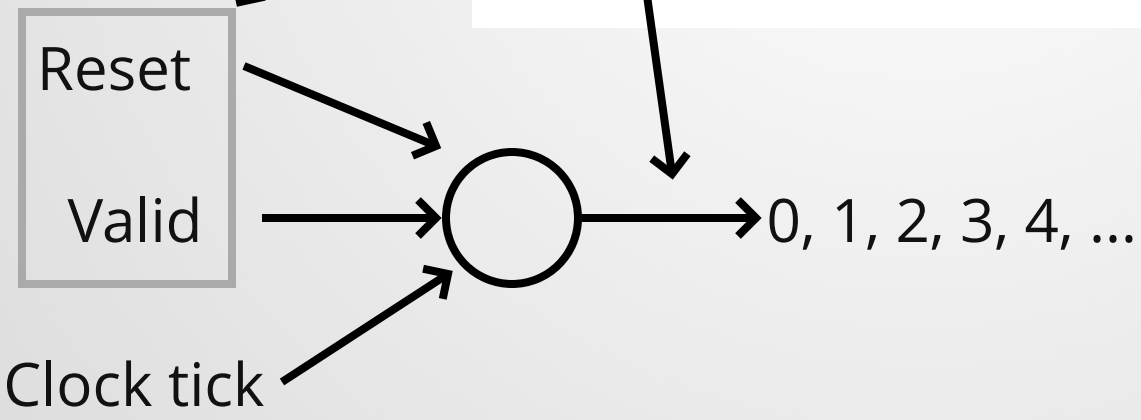
- External choice
- Broadcast
- Single writer

SME Examples

SME

Counter example - communication

```
public interface ICounterControl : IBus
{
  [Initial(false)]
  bool Reset { get; set; }

  [Initial(false)]
  bool Valid { get; set; }
}

public interface ICounterData : IBus
{
  [Initial(0)]
  int Value { get; set; }
}
```



SME

Counter example - logic

```
public interface ICounterControl : IBus {
  [InitialValue(false)] bool Reset { get; set
  [InitialValue(false)] bool Valid { get; set
}

public interface ICounterData : IBus {
  [InitialValue(0)] int Value { get; set; }
}
```

```
using SME;

public class Counter : SimpleProcess {
  public readonly ICounterControl m_control = Scope.CreateBus<ICounterControl>();
  public readonly ICounterData m_data = Scope.CreateBus<ICounterData>();

  protected override void OnTick() {
    if (m_control.Reset) {
      m_data.Value = 0;
    } else {
      m_data.Value++;
      //m_data.Value = m_data.Value + 1;
    }
  }
}
```

SME

Larger example

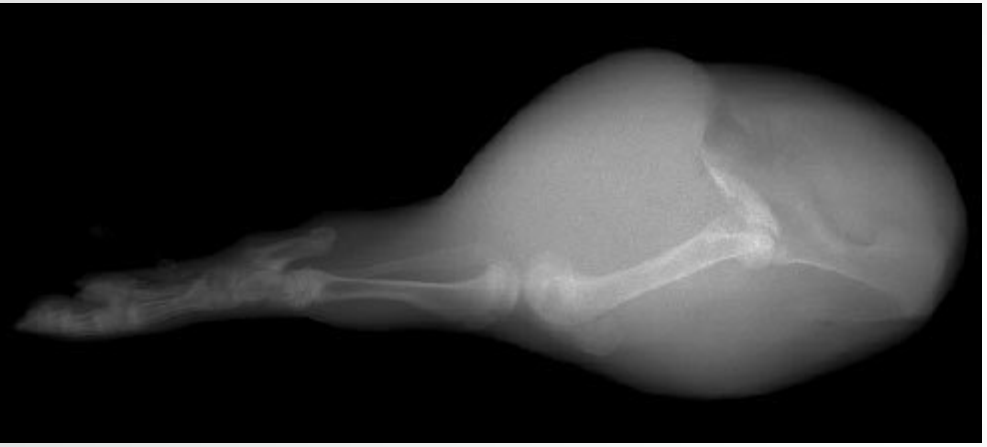


Image from FOSS

$$I = R * 0.299 + G * 0.587 + B * 0.144$$

Image from FOSS

SME

Larger example

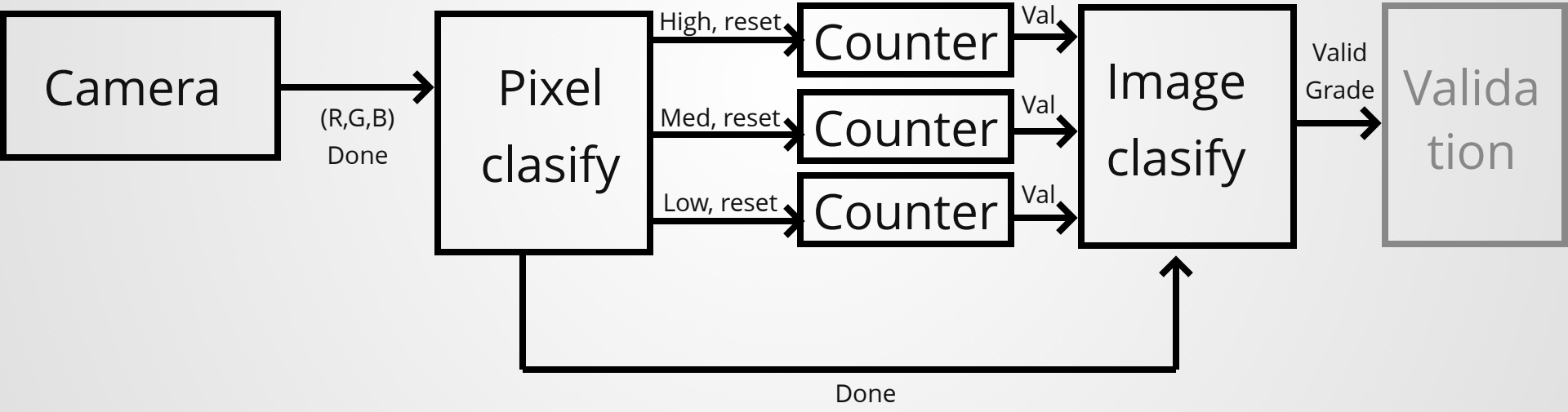


Image from FOSS

SME

Larger example

```
[TopLevelInputBus] public interface ICamData : IBus {
    [InitialValue(false)] bool Valid { get; set; }
    [InitialValue(false)] bool Done { get; set; }
    byte R { get; set; }
    byte G { get; set; }
    byte B { get; set; }
}

public interface IClassifierForward : IBus {
    [InitialValue(false)] bool Done { get; set; }
}
```

```
public class PixelClassifier : SimpleProcess {
  public readonly ICamData m_camera = Scope.CreateOrLoadBus<ICamData>();
  public readonly IClassifierForward m_forward = Scope.CreateBus<IClassifierForward>();
  readonly ICounterControl m_counterLow; readonly ICounterControl m_counterMed; readonly ICounterControl m_counterHigh;

  private bool m_isActive = false;

  public PixelClasifier(ICounterControl low, ICounterControl med, ICounterControl high) {
    m_counterLow = low; m_counterMed = med; m_counterHigh = high;
  }

  protected override void OnTick() {
    if (!m_isActive && m_camera.Valid) {
      m_isActive = true;
    }

    m_forward.Done = false;
    m_counterLow.Reset = m_counterMed.Reset = m_counterHigh.Reset = !m_isActive;

    if (m_isActive && m_camera.Valid) {
      var i = (m_camera.R * 299u + m_camera.G * 587u + m_camera.B * 144u) / 1000u;;

      m_counterLow.Valid = m_counterMed.Valid = m_counterHigh.Valid = false;
      if (i > TH_HIGH)
        m_counterHigh.Valid = true;
      else if (i > TH_MED)
        m_counterMed.Valid = true;
      else if (i > TH_LOW)
        m_counterLow.Valid = true;

      if (m_camera.Done)
        m_forward.Done = true;
    }
  }
}
```

Image from FOSS

SME

Larger example

```
[TopLevelOutputBus]
public interface IGraderResult : IBus {
    [InitialValue(false)] bool Valid { get; set; }
    byte Grade { get; set; }
}
```

```
public class ImageClassifier : SimpleProcess {
  public readonly IGraderResult m_result = Scope.CreateBus<IGraderResult>();
  readonly IClassifierForward m_forward;
  readonly ICounterData m_counterLow;
  readonly ICounterData m_counterMed;
  readonly ICounterData m_counterHigh;

  private bool m_isActive = false;

  public ImageClasifier(IClassifierForward forward, ICounterData low, ICounterData med, ICounterData high)
    m_forward = forward;
    m_counterLow = low; m_counterMed = med; m_counterHigh = high;
  }

  protected override void OnTick() {
    m_result.Valid = false;
    if (m_forward.Done) {
      m_result.Valid = true;
      if (m_counterHigh.Value > 20)
        m_result.Grade = 2;
      else if (m_counterMed.Value > 20)
        m_result.Grade = 1;
      else if (m_counterLow.Value > 10)
        m_result.Grade = 0;
      else
        m_result.Grade = 3;
    }
  }
}
```

Image from FOSS

Testing

```
public class CameraSimulator : SimulationProcess {
  public readonly ICamData m_camera = Scope.CreateBus<ICamData>();

  public override async Task Run()
  {
    await ClockAsync();

    using (var img = System.Drawing.Image.FromFile(file))
    using (var bmp = new System.Drawing.Bitmap(img, new Size(20, 20)))
    {
      m_camera.Valid = true;

      for (var i = 0; i < bmp.Height; i++)
      {
        for (var j = 0; j < bmp.Width; j++)
        {
          var pixel = bmp.GetPixel(j, i);
          m_camera.R = pixel.R;
          m_camera.G = pixel.G;
          m_camera.B = pixel.B;
          m_camera.Done = i == bmp.Height - 1 && j == bmp.Width - 1;

          await ClockAsync();
        }
      }
    }

    m_camera.Done = false;
    await ClockAsync();
    m_camera.Valid = true;
  }
}
```

Image from FOSS

Validating

```
public class Validator : SimulationProcess {
  public readonly IGraderResult m_result = Scope.LoadBus<IGraderResult>();

  public override async Task Run()
  {
    while(!m_result.Valid)
      await ClockAsync();

    Console.WriteLine($"Grade is: {m_result.Grade}");
  }
}
```

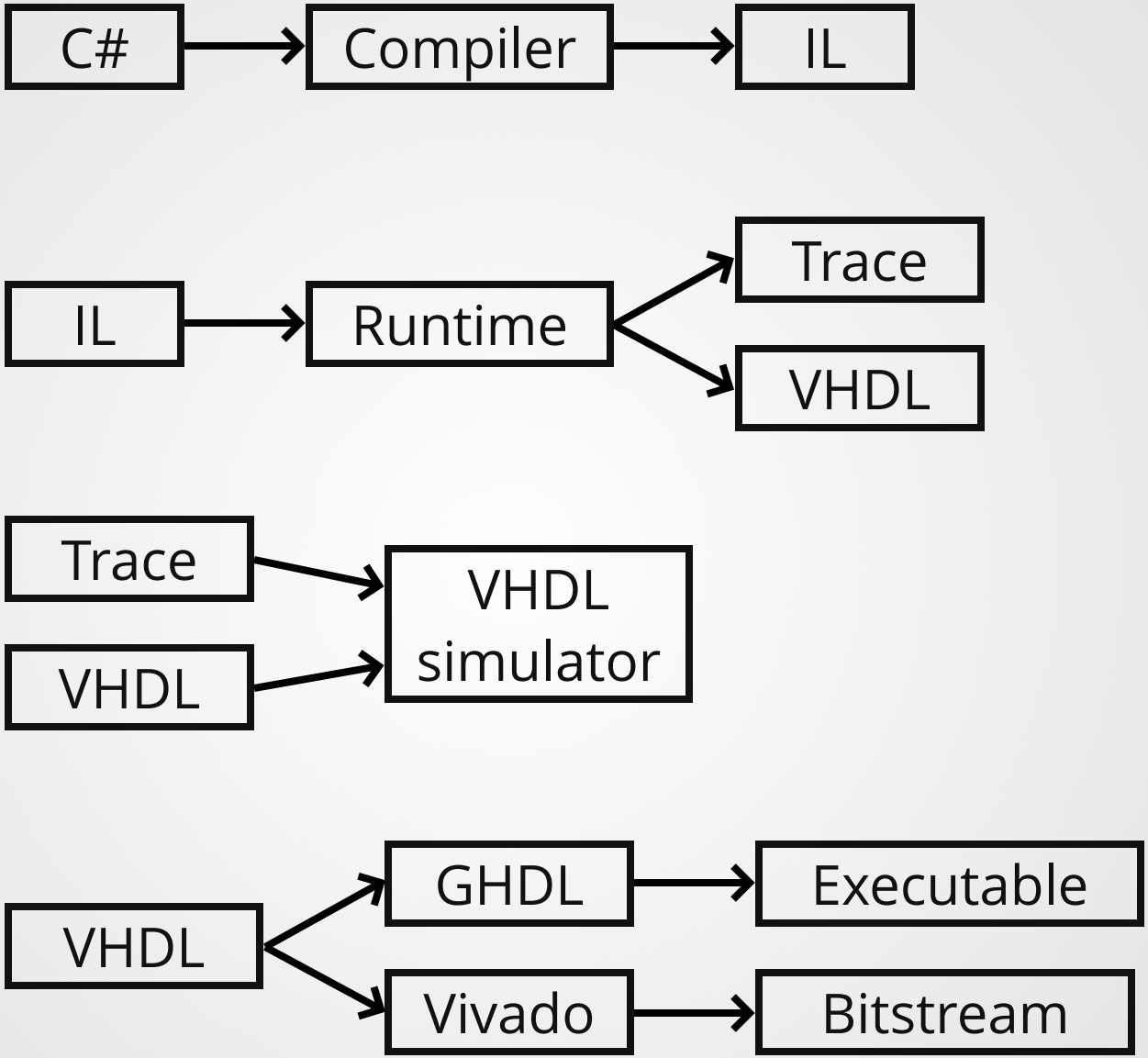
Image from FOSS

SME

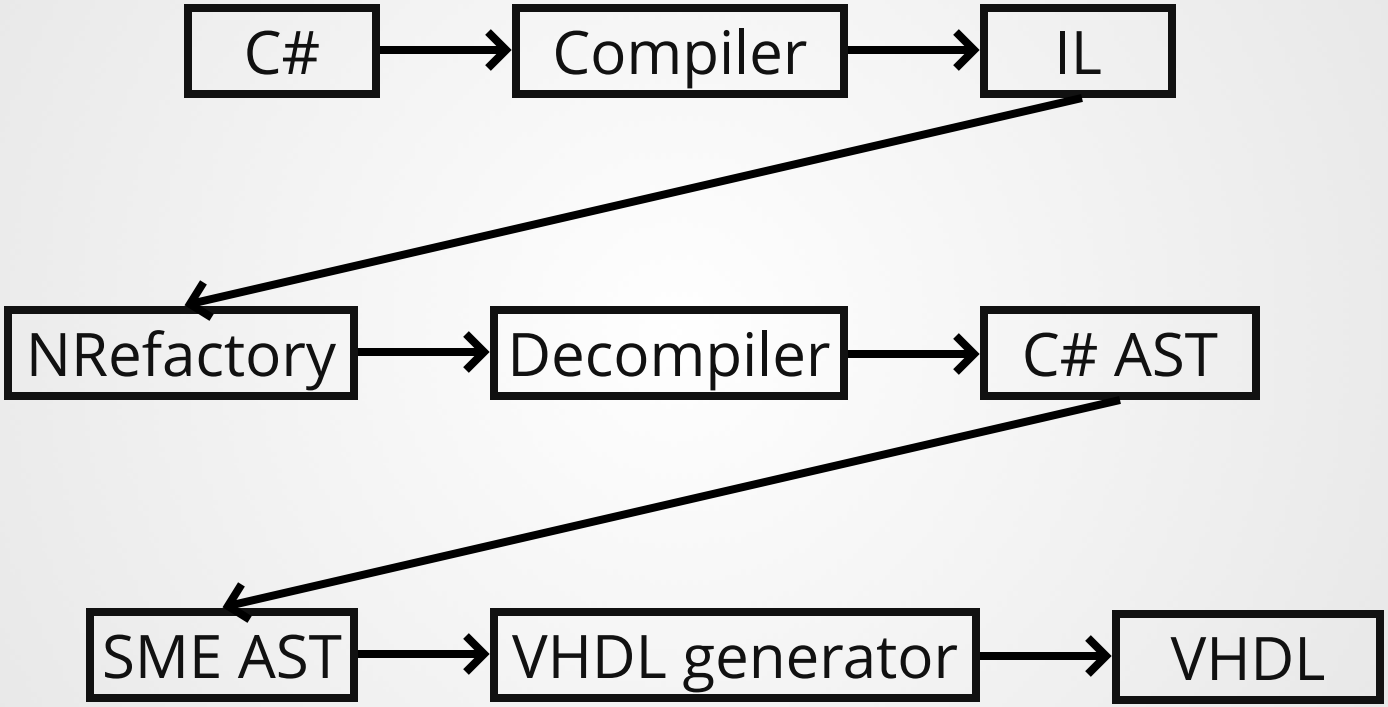
Larger example

```
public class Program {
  static void Main() {
    using(var sim = new Simulation()) {
      sim
        .BuildCSVFile()
        .BuildGraph()
        .BuildVHDL();


      var simulator = new CameraSimulator();

      var counterLow = new Counter();
      var counterMed = new Counter();
      var counterHigh = new Counter();
      var pixelClasifier = new PixelClassifier(simulator.m_camera, counterLow, counterMed, counterHigh);
      var imageGrader = new ImageClassifer(pixelClasifier.m_forward, counterLow, counterMed, counterHigh);

      var validator = new Validator();


      sim.Run();
  }
}
```

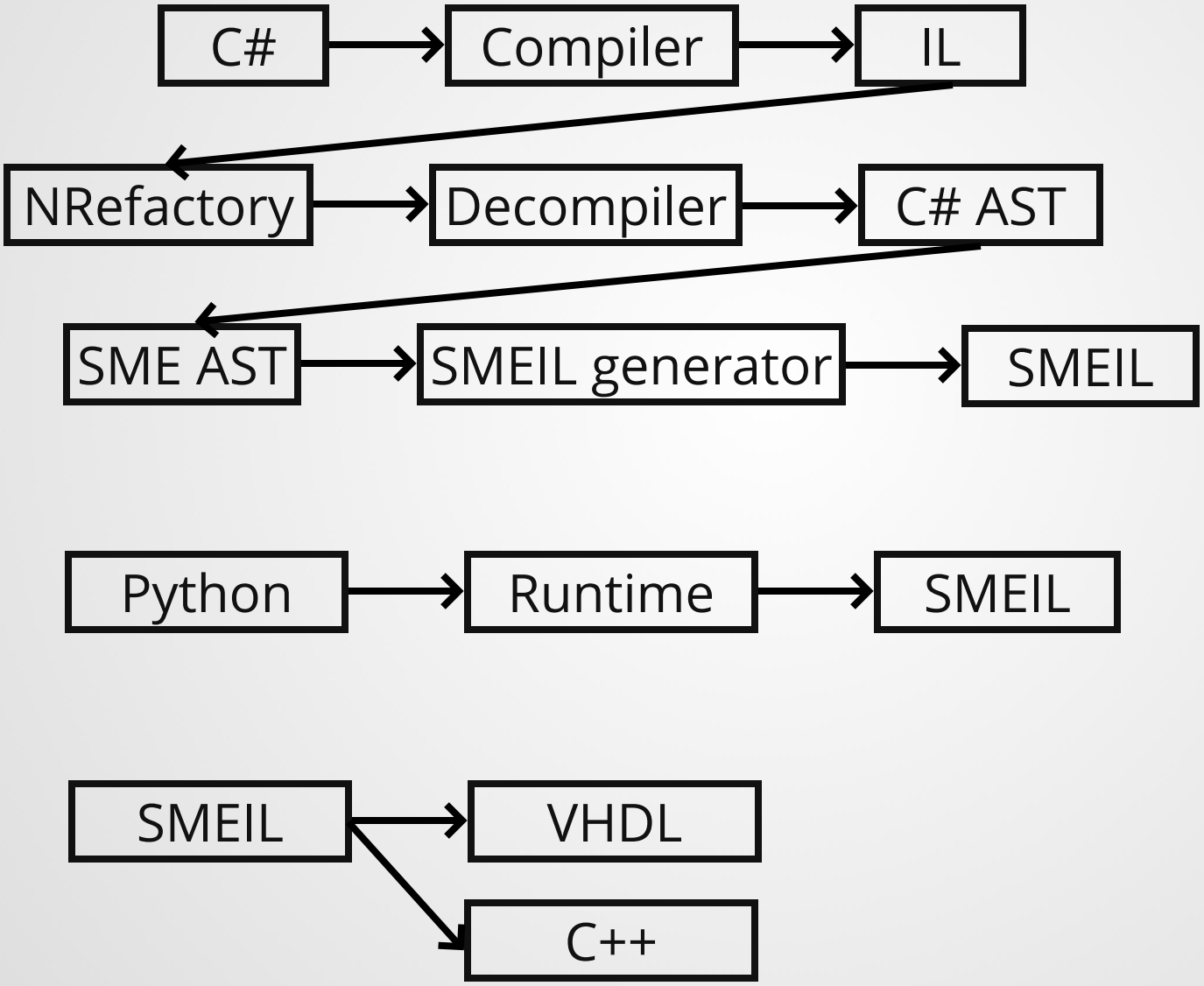
SME internals


C#
Compiler
IL
IL
Runtime
Trace
VHDL
Trace
VHDL
VHDL
simulator
VHDL
GHDL
Executable
Vivado
Bitstream

Current approach



New approach



Demo

- VS Code
- Mono or .Net Framework
- NuGet
- GHDL
- GTK Wave