

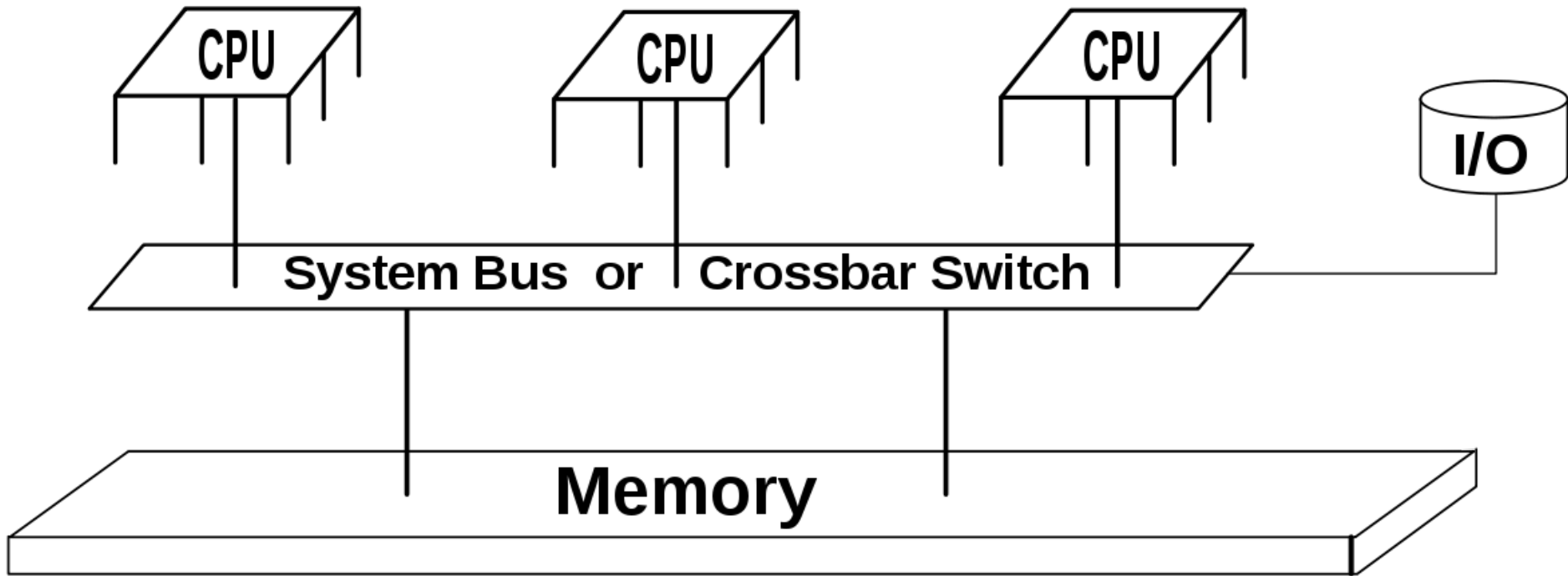
Parallel Architectures

Memory Access

- Multiple processing units
- Potentially multiple memory units
- Does each PU have its own mem.?
- Is it shared with others?
- What is access time between PU and mem.?
 - When it is not shared
 - When it is shared

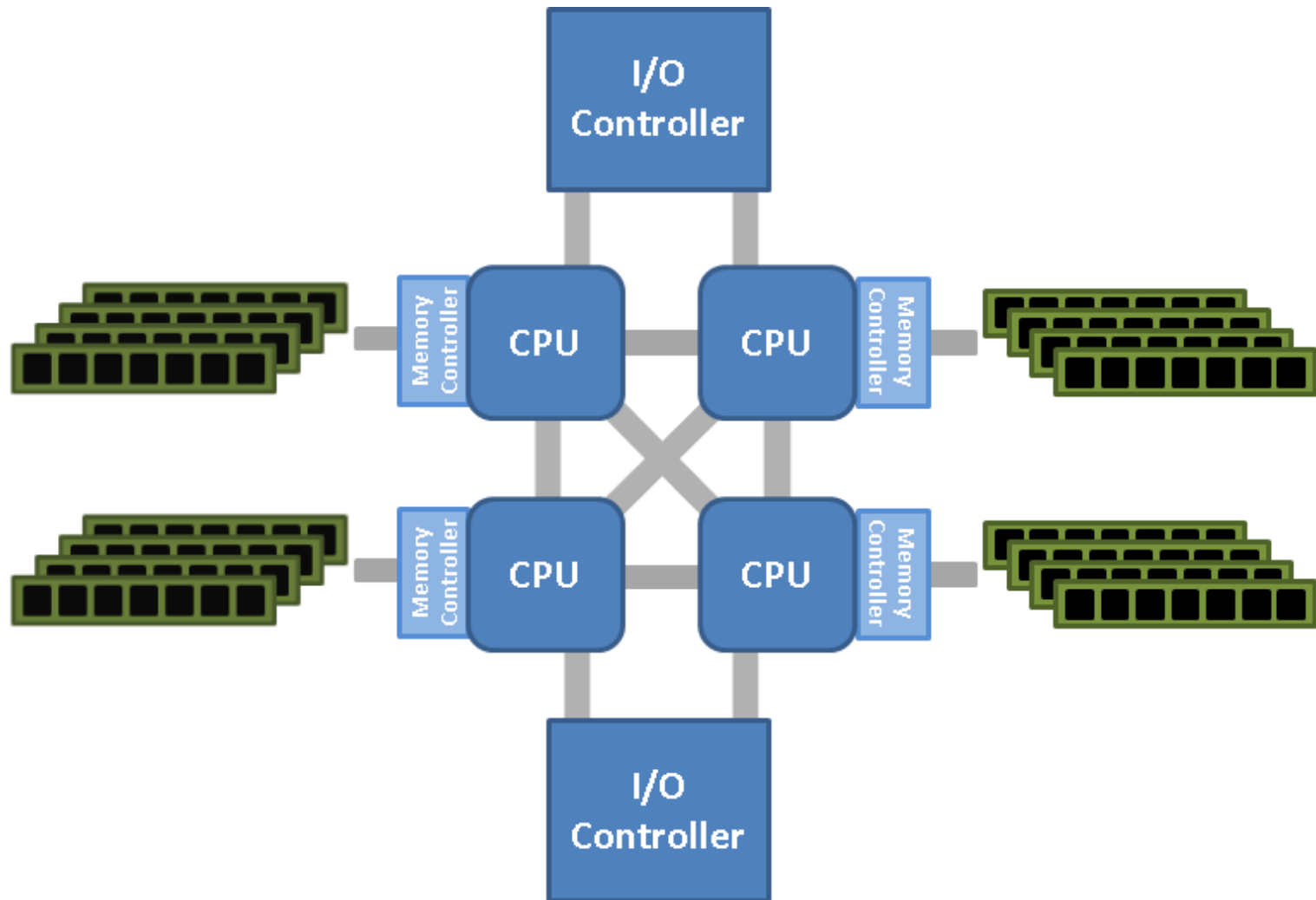
Memory Access

Uniform Memory Access (UMA)



Memory Access

Non-Uniform Memory Access (NUMA)



Memory Access

	UMA	NUMA
Latency	Same	Different
Bandwidth	Same	Different
Memory	Shared	Distributed

Memory Access

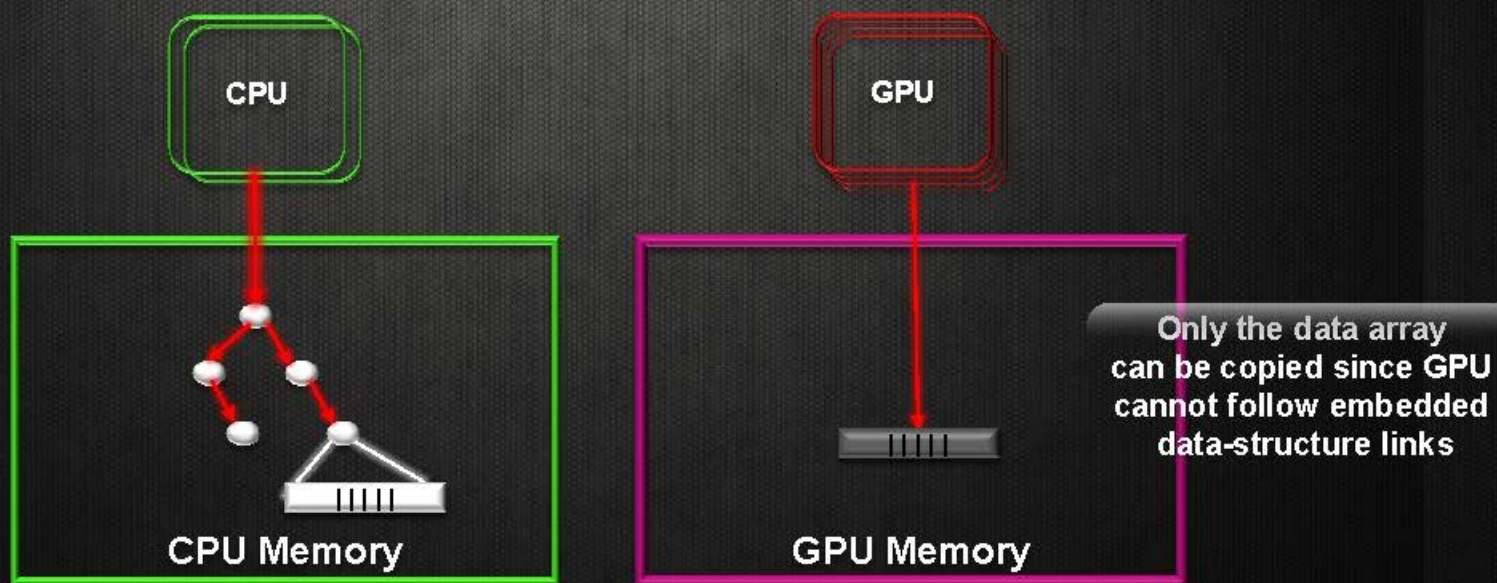
heretogenous Uniform Memory Access (hUMA)

WITHOUT POINTERS* AND DATA SHARING



Without hUMA:

- CPU explicitly copies data to GPU memory
- GPU completes computation
- CPU explicitly copies result back to CPU memory



*A Pointer is a named variable that holds a memory address. It makes it easy to reference data or code segments by a name and eliminates the need for the developer to know the actual address in memory. Pointers can be manipulated by the same expressions used to operate on any other variable

Memory Access

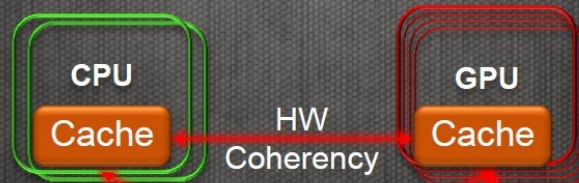
heretogenous Uniform Memory Access (hUMA)

hUMA KEY FEATURES



Coherent Memory:

Ensures CPU and GPU caches both see an up-to-date view of data



Pageable memory:

The GPU can seamlessly access virtual memory addresses that are not (yet) present in physical memory

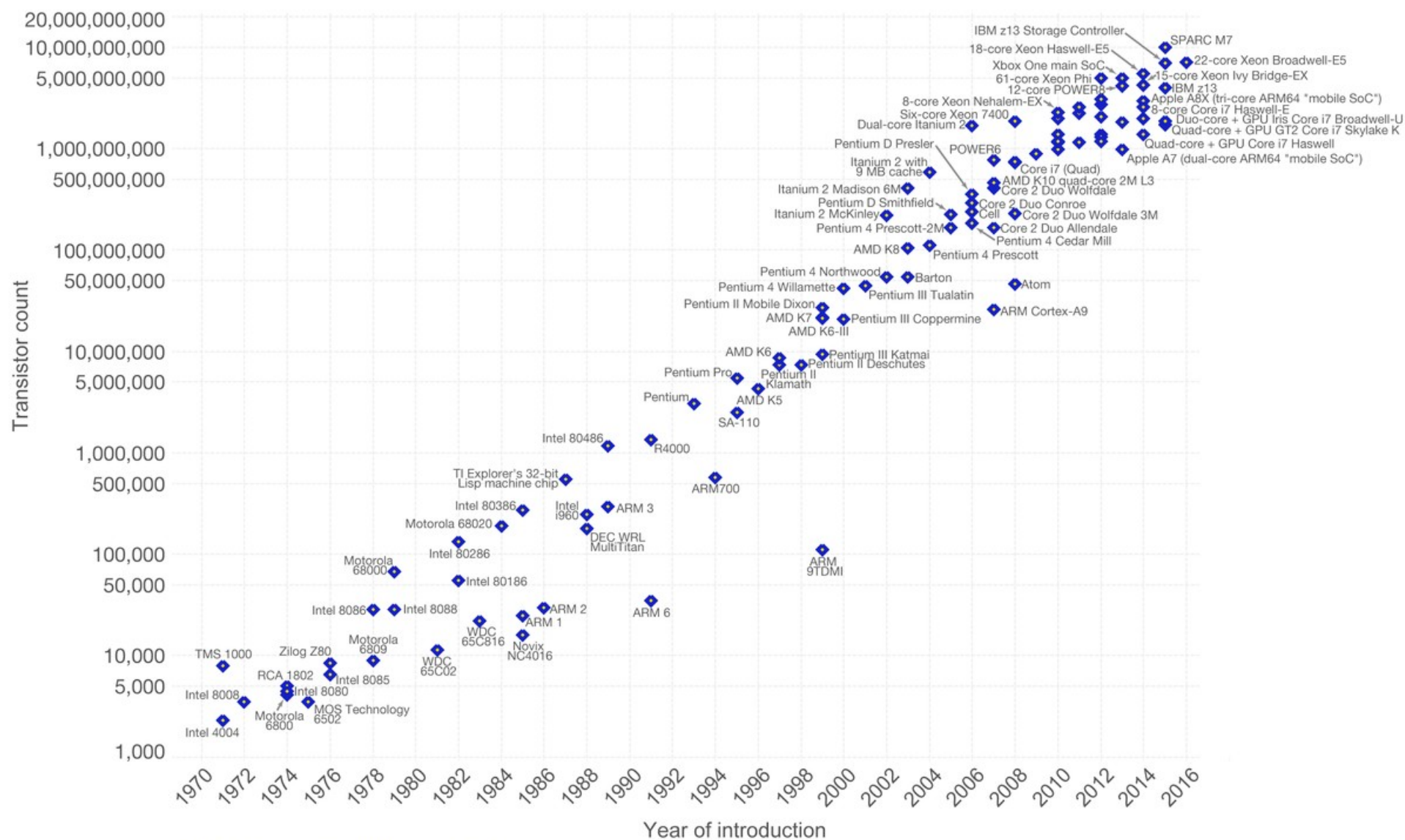


Entire memory space:

Both CPU and GPU can access and allocate any location in the system's virtual memory space

Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



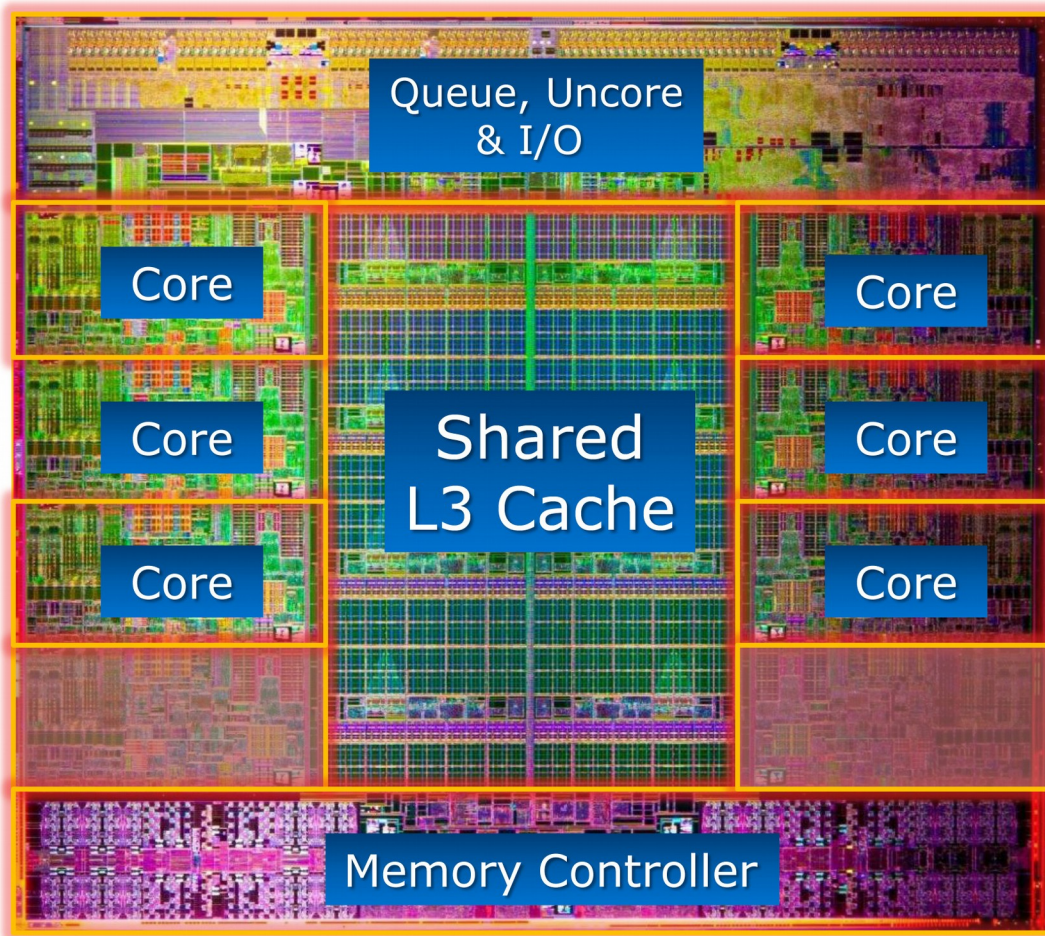
Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)

The data visualization is available at [OurWorldinData.org](https://www.ourworldindata.org). There you find more visualizations and research on this topic.

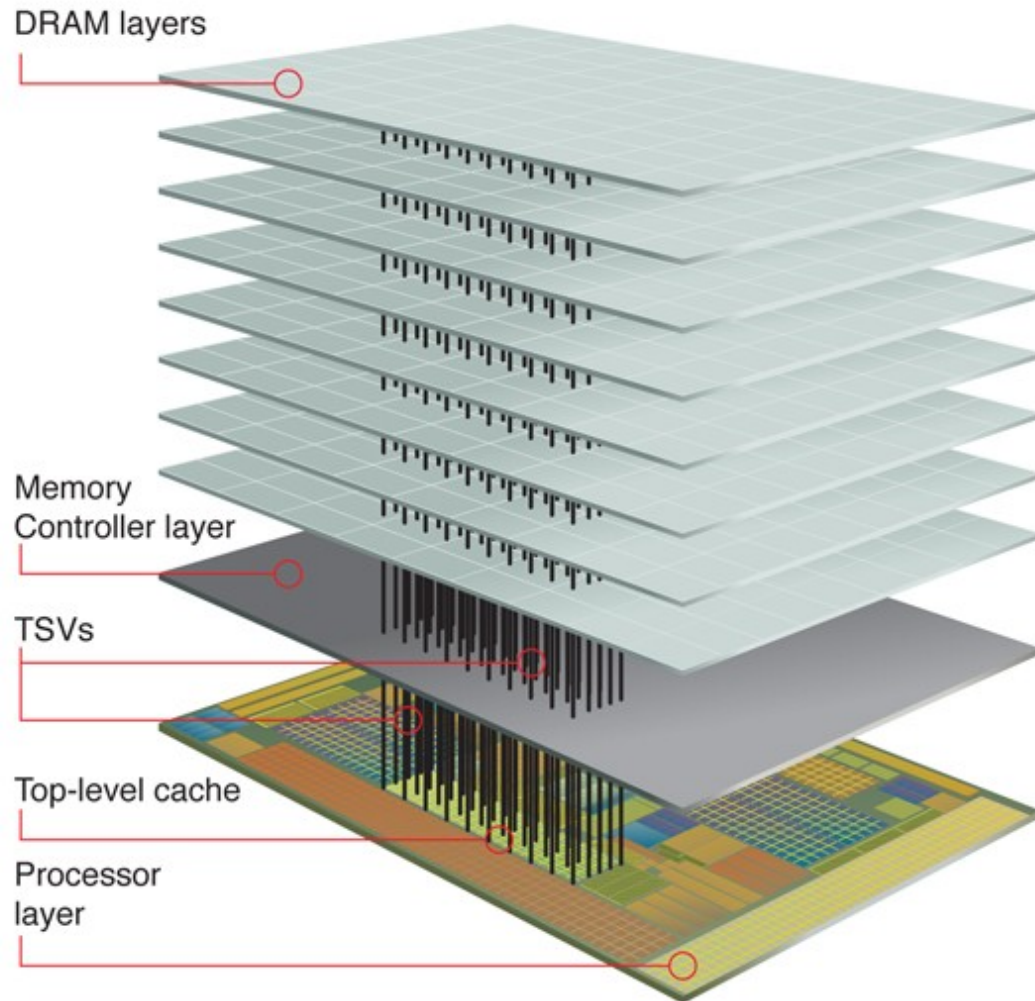
Licensed under CC-BY-SA by the author Max Roser.

Intel Core i7 3960X Sandy-Bridge E

3.3GHz (3.9Ghz Turbo) | 6core | 15MB L3 | 130W TDP



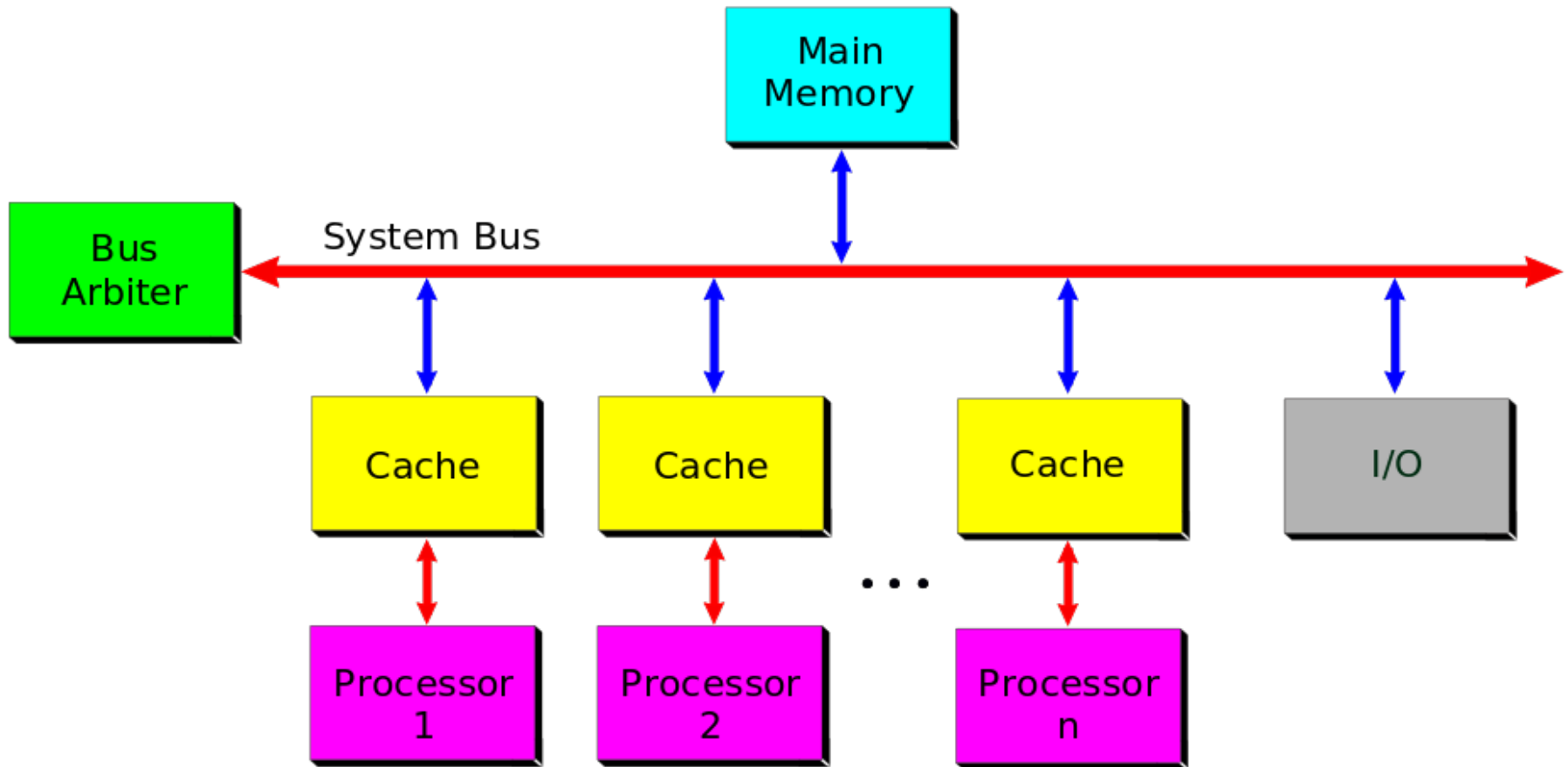
3D Processors



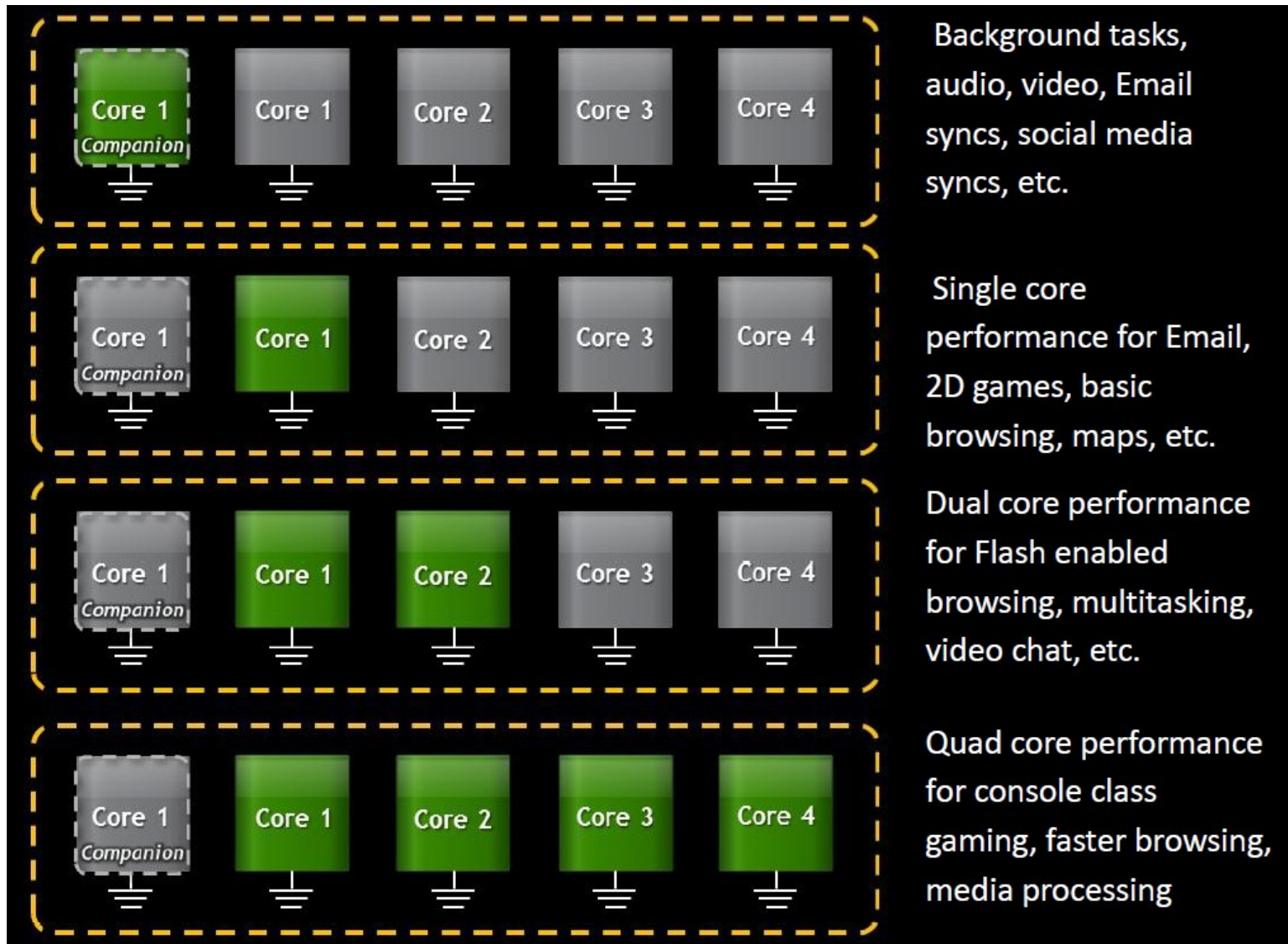
Symmetric vs Asymmetric

- 2+ identical processors connected to single shared memory --> SMP
- Most multiprocessors use SMP
- For OS, all processors are treated same
- Tightly coupled (connected at bus level)
- If processors are not treated same, then it is Asymmetric
- ASMP is expensive, hence rarer

SMP - Symmetric Multiprocessor System



variable SMP (vSMP)

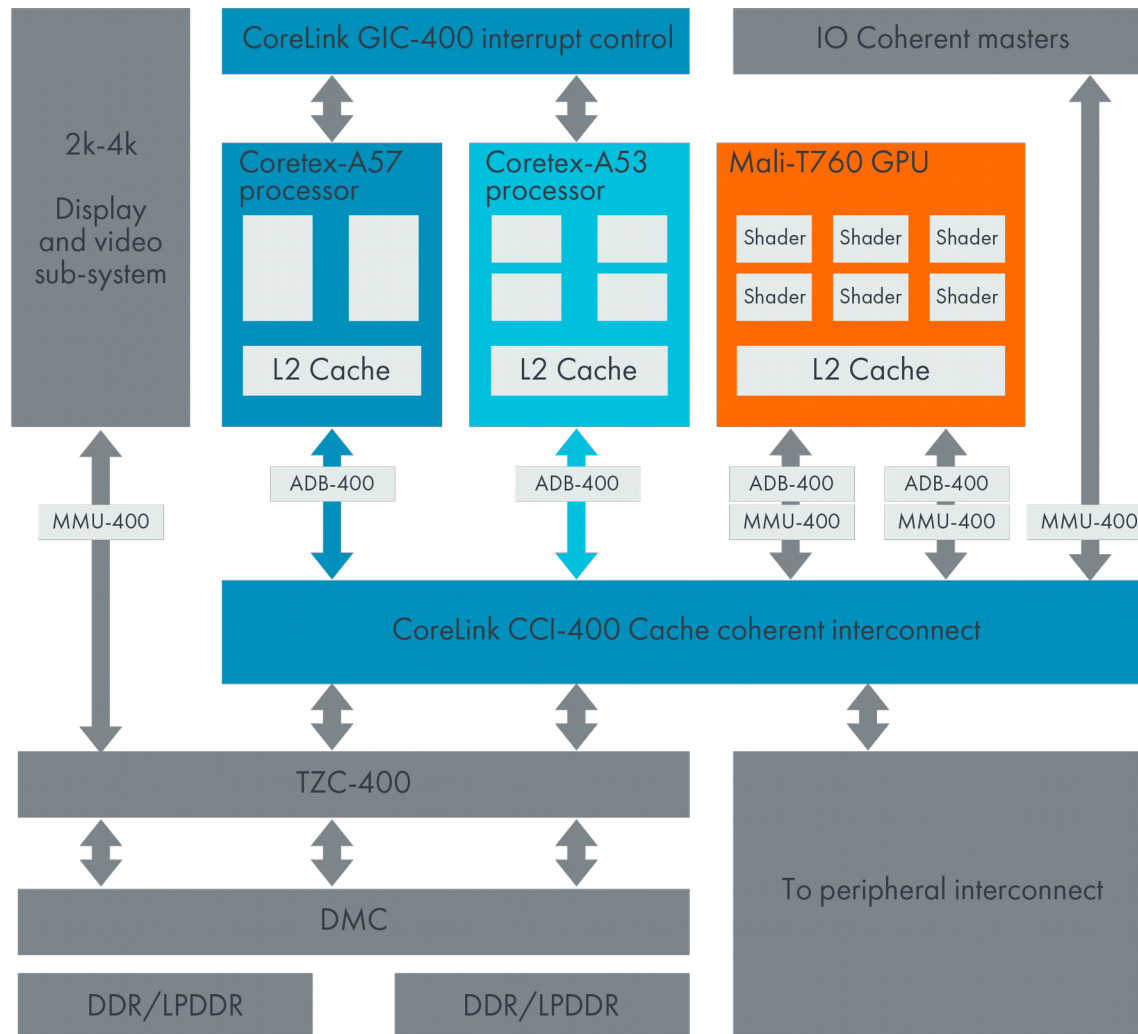


Multicore Processors

- May or may not share cache
- May implement message passing or IPC
- Cores can be connected in -
 - bus, ring, 2D mesh, crossbar
- Homogenous or Heterogenous

big.LITTLE

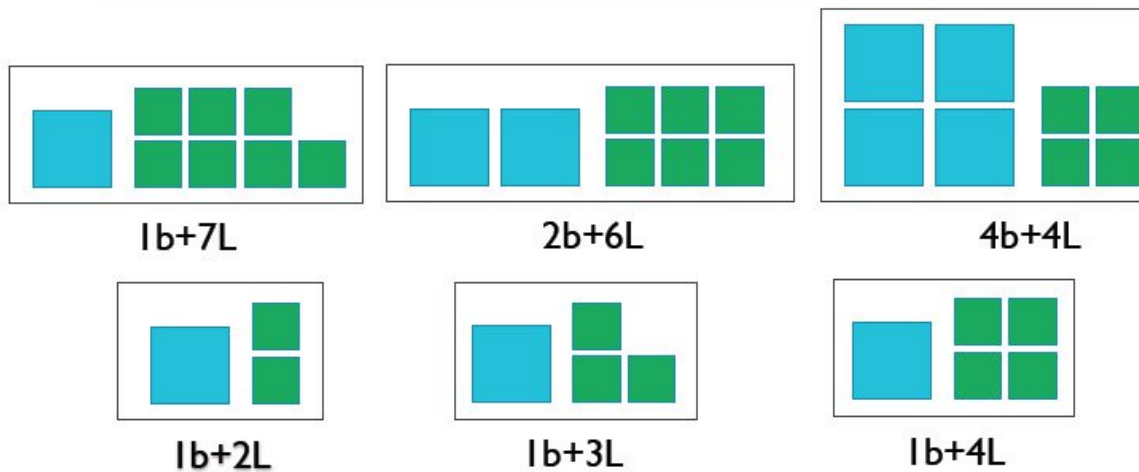
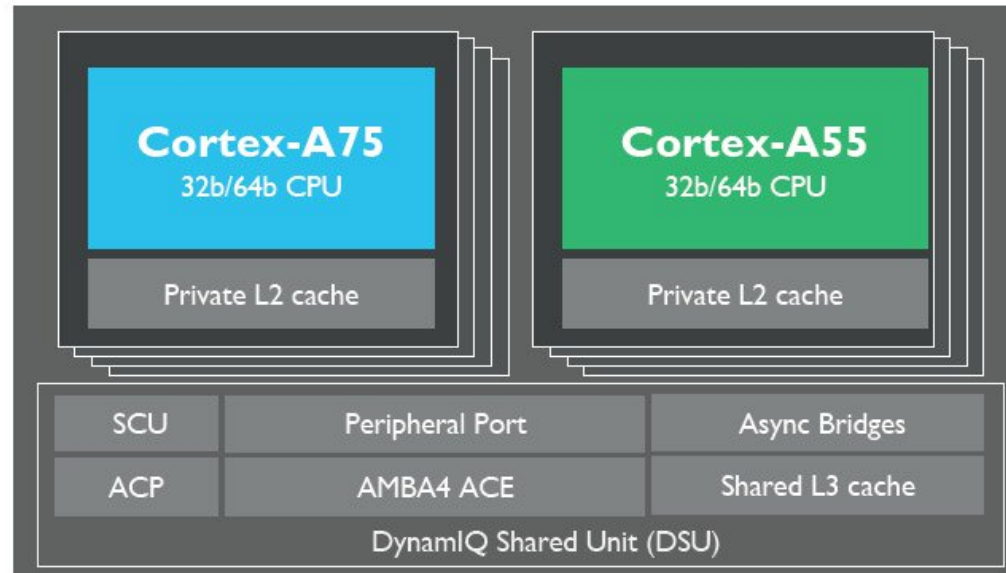
ARM architecture



big.LITTLE

- Finer-grained control of workloads
- Implementation in the schedule
 - Clustered switching
 - In-kernel switcher (CPU migration)
 - Heterogeneous multi-processing
(global task scheduling)
- Easily support non-symmetrical SoCs
- Use all cores simultaneously to provide improved peak performance

DynamiQ



DynamiQ

- Combines big and LITTLE cores into single, fully integrated cluster
- Better power and memory efficiency
- 1-8 Cortex A-* CPUs in one cluster
- Great for Artificial Intelligence and Machine Learning processing
- Various configurations

Instruction Level Parallelism (ILP)

- How many instructions can be executed simultaneously? --> *measure with ILP*
- hardware (dynamic parallelism)
 - Decide at runtime what to execute
 - Pentium (and all else)
- software (static parallelism)
 - Compiler decides what to parallelise
 - Itanium (and server cores)

Instruction Pipelining

- Within single processor
- Keep every part of processor busy
- Divide instructions
- Execute in parallel
- Fetch-Decode-Execute cycle

Pipeline Braching

- If a branch is not taken, wasted resources
- Causes delay in execution --> *bubble*
- Branch prediction
 - Algorithm to predict which branch might be taken to prevent bubbles
 - Very complex to execute accurately

Patent US7069426 (Intel)

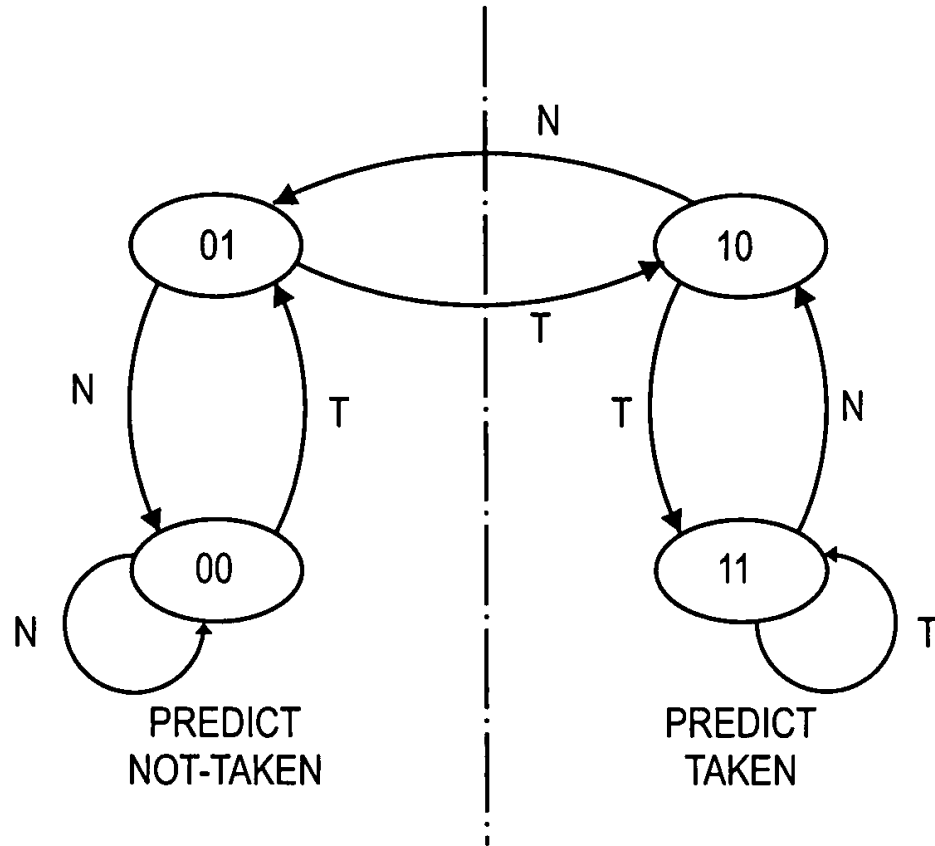


FIG. 2

```

const unsigned arraySize = 32768;
int data[arraySize];
for (unsigned c = 0; c < arraySize; ++c)
    data[c] = std::rand() % 256;

for (unsigned i = 0; i < 100000; ++i)
{
    // Primary loop
    for (unsigned c = 0; c < arraySize; ++c)
    {
        if (data[c] >= 128)
            sum += data[c];
    }
}

// execution time --> 11.54s

```

```

std::sort(data, data + arraySize);
for (unsigned i = 0; i < 100000; ++i)
{
    // Primary loop
    for (unsigned c = 0; c < arraySize; ++c)
    {
        if (data[c] >= 128)
            sum += data[c];
    }
}

// execution time --> 1.93s

```



```

const unsigned arraySize = 32768;
int data[arraySize];
for (unsigned c = 0; c < arraySize; ++c)
    data[c] = std::rand() % 256;

```

```

for (unsigned i = 0; i < 100000; ++i)
{
    // Primary loop
    for (unsigned c = 0; c < arraySize; ++c)
    {
        if (data[c] >= 128)
            sum += data[c];
    }
}

```

// execution time --> 11.54s

```

std::sort(data, data + arraySize);
for (unsigned i = 0; i < 100000; ++i)
{
    // Primary loop
    for (unsigned c = 0; c < arraySize; ++c)
    {
        if (data[c] >= 128)
            sum += data[c];
    }
}

```

// execution time --> 1.93s

```

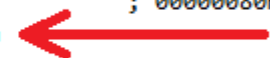
if (data[c] >= 128)
    sum += data[c];

```

```

2876     movsxd  rdx, DWORD PTR [rcx]
2877     cmp     edx, 128
2878     jle     SHORT $LN3@main
2879     add     rbx, rdx
2880 $LN3@main:

```



T = branch taken

N = branch not taken

data[] = 0, 1, 2, 3, 4, ... 126, 127, 128, 129, 130, ... 250, 251, 252, ...

branch = N N N N N ... N N T T T ... T T T ...
 = NNNNNNNNNNN ... NNNNNNTTTTTTTTTT ... TTTTTTTTTT (easy to predict)

gcc -O3 or gcc -ftreevectorise

<https://stackoverflow.com/questions/11227809/>

Superscalar

- Scalar – each instruction manipulates {1,2} data items at a time
- Superscalar – Execute more than one instruction at a time
- How? --> multiple simultaneous instructions to different execution units
- More throughput per clock cycle
- Flynn's Taxonomy
 - SISD for single core (or SIMD for vector ops)
 - MIMD for multiple cores