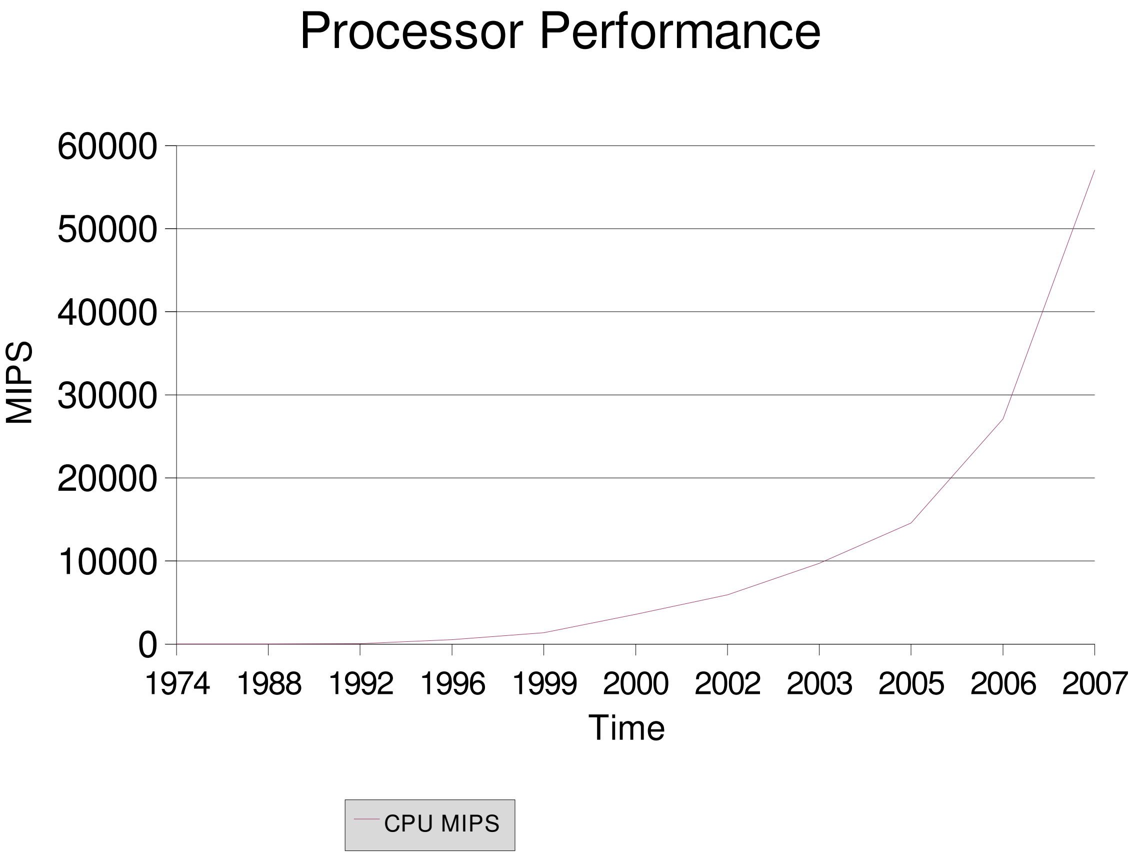
Of Programmers And Hardware: Transcending The Gap

Ulrich Drepper

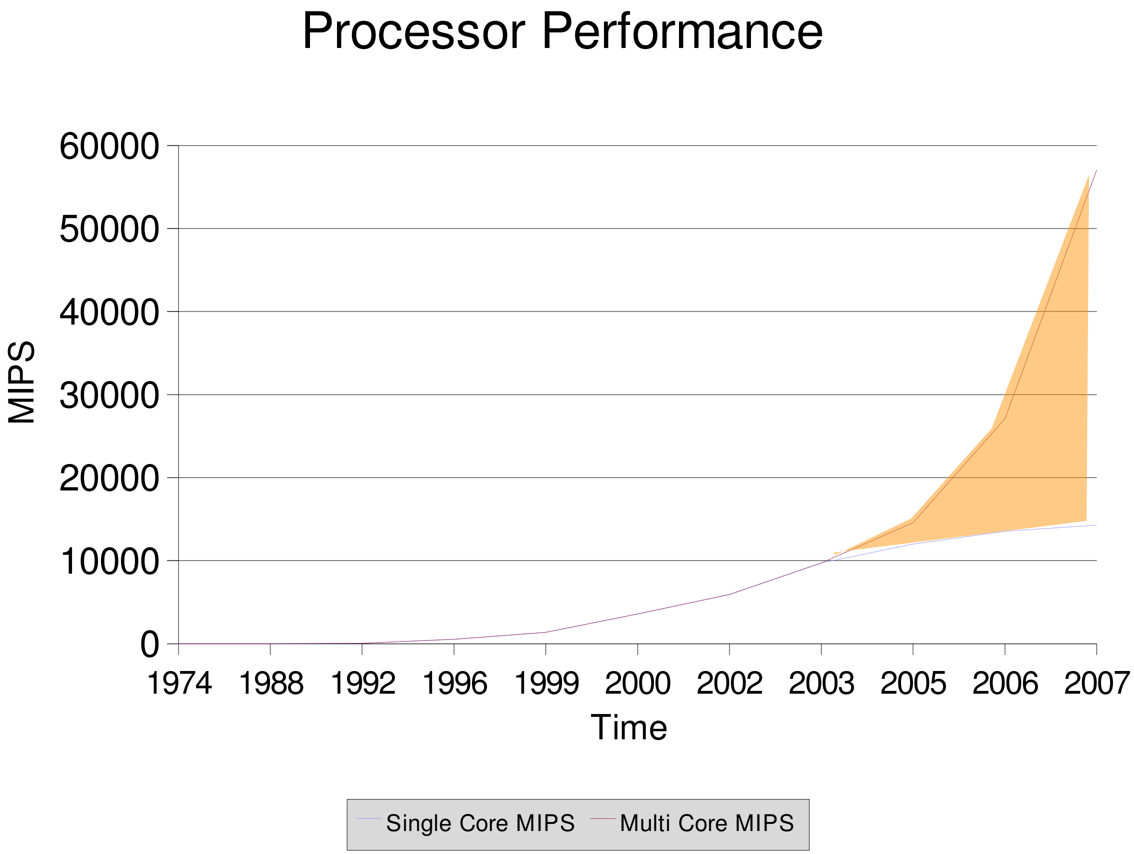
Advertisement Claims

- What the CPU manufacturers want you to believe



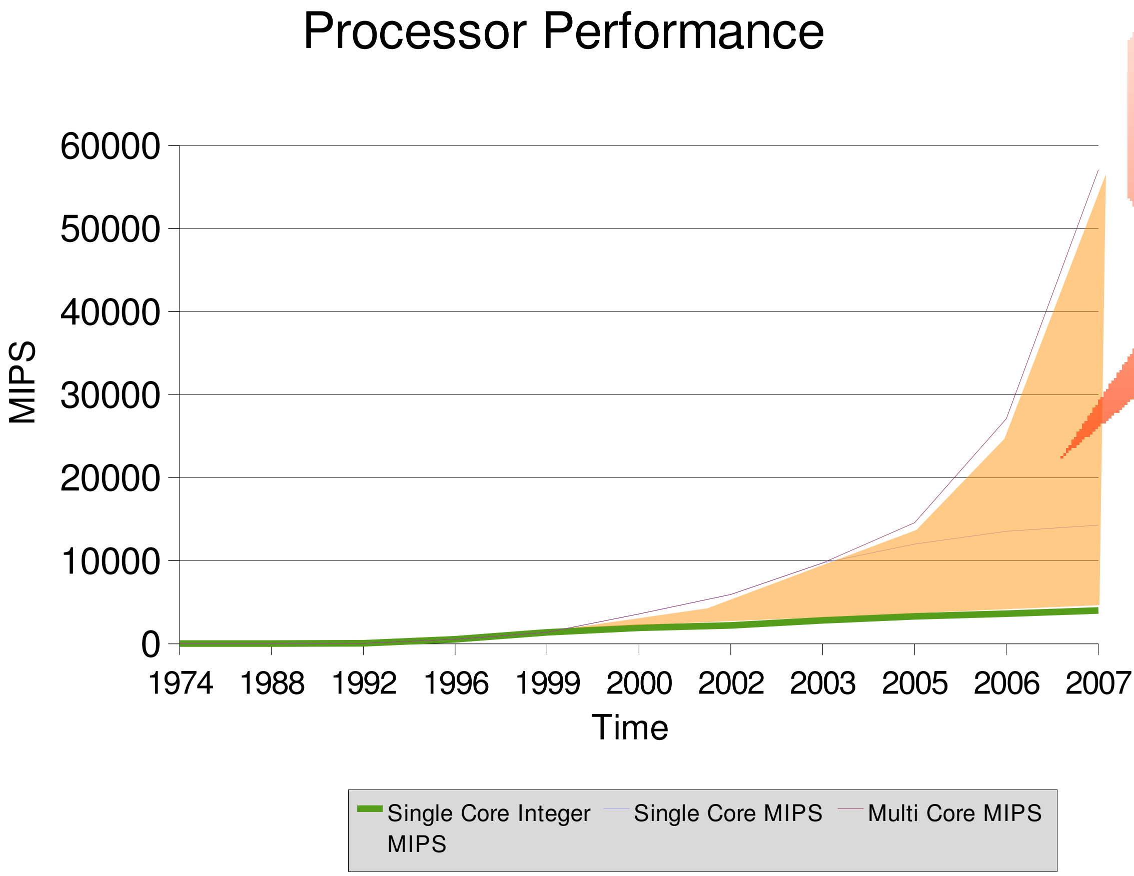
Advertisement Claims

- But most programs are not multi-threaded...



Advertisement Claims

- Ignoring SIMD performance

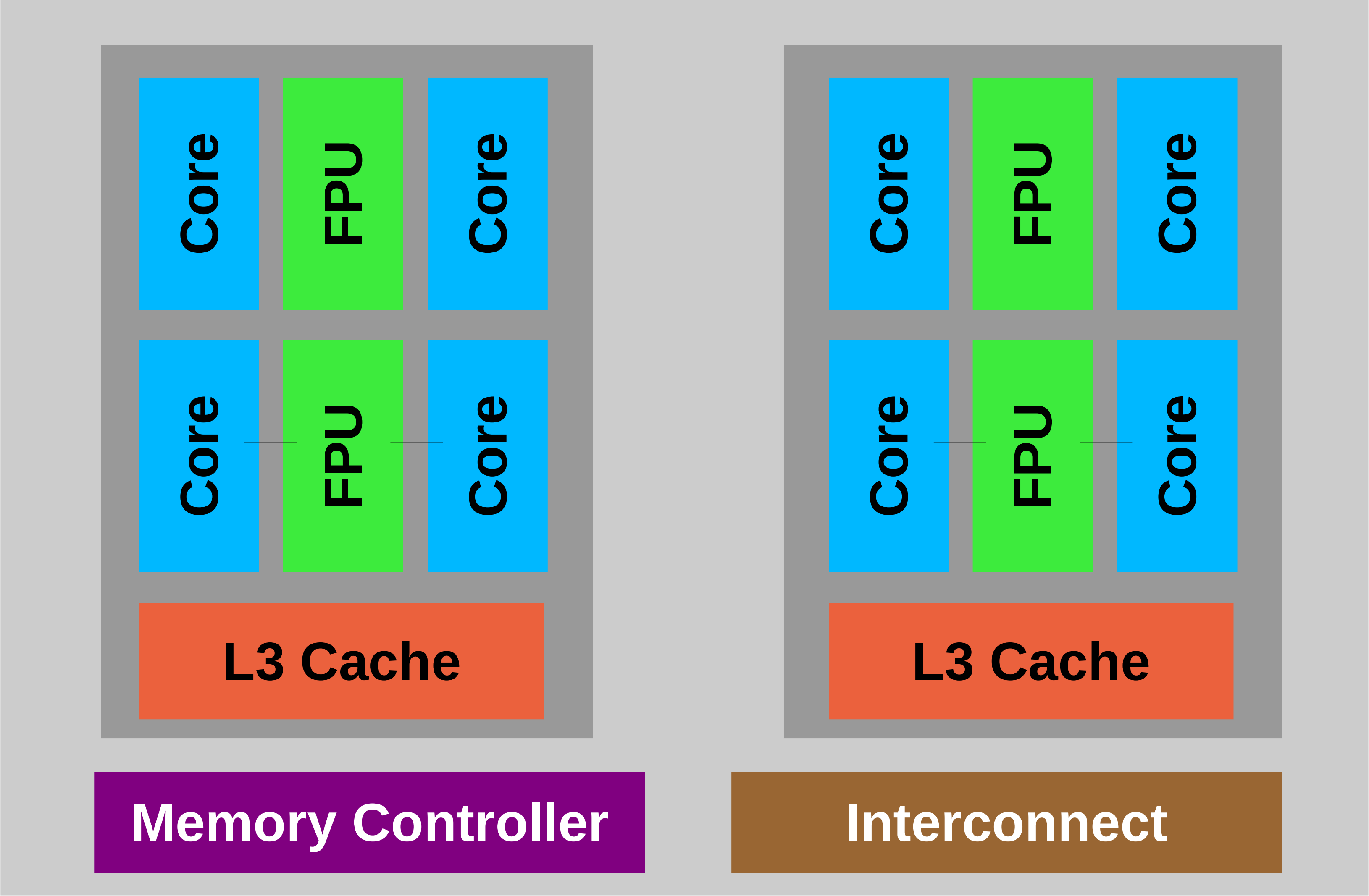


We have to work for this!

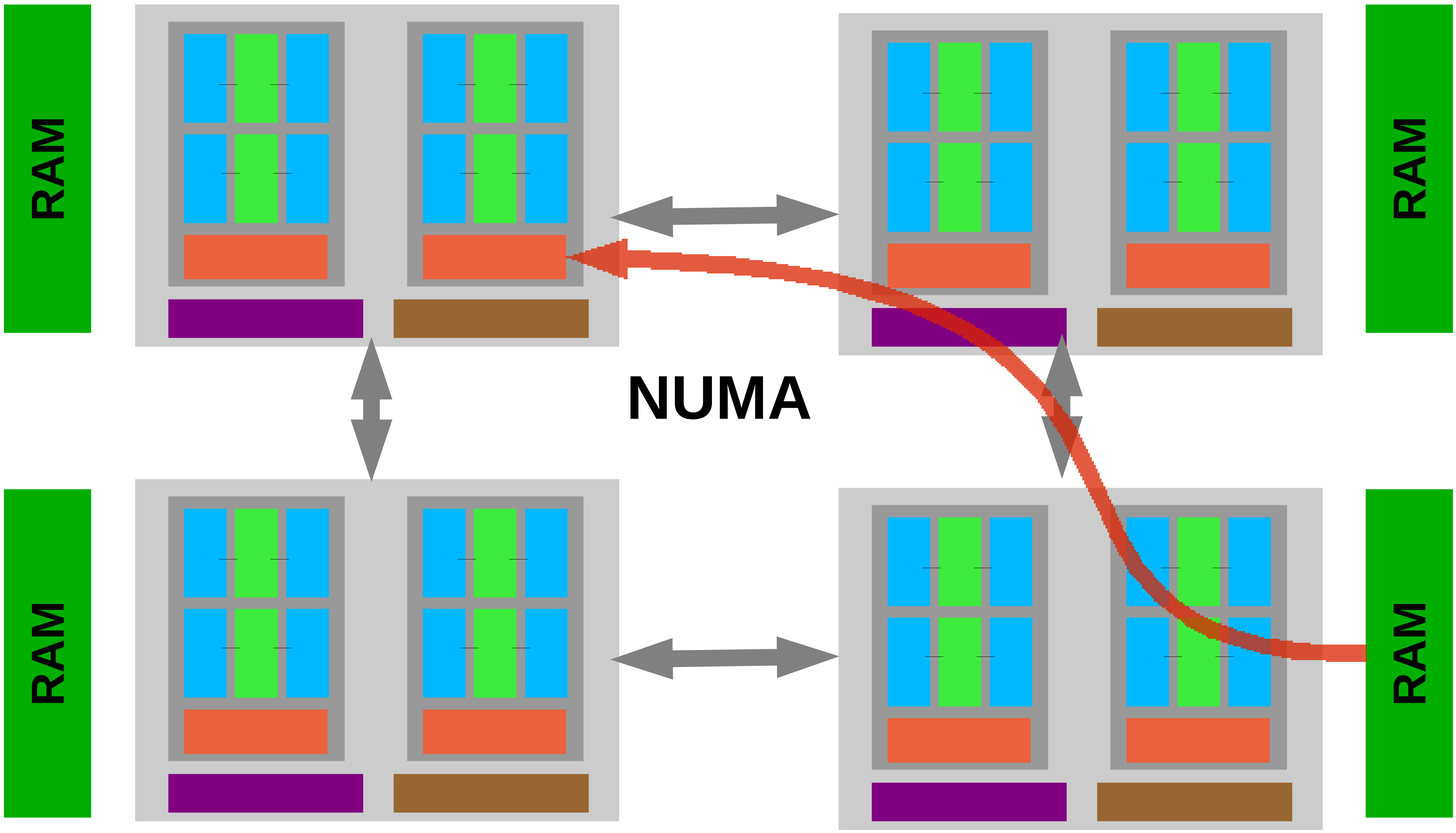
Parallelism

- Two types
  - Multi-core
  - Hyperthreads
- Not all cores in a package equally connected
- Planning concurrent execution
  - How much data has to be shared?
  - Functional units shared between cores/threads

CPU Structure

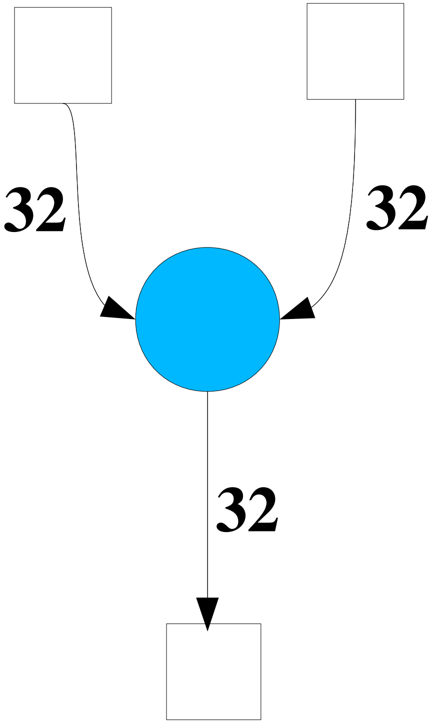


System Structure



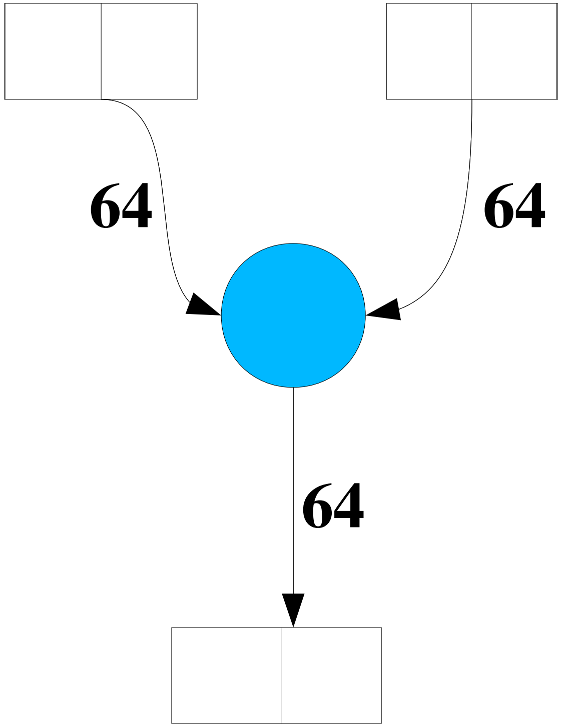
SIMD

- Single-Instruction/Multiple Data
  - Normal Arithmetic:



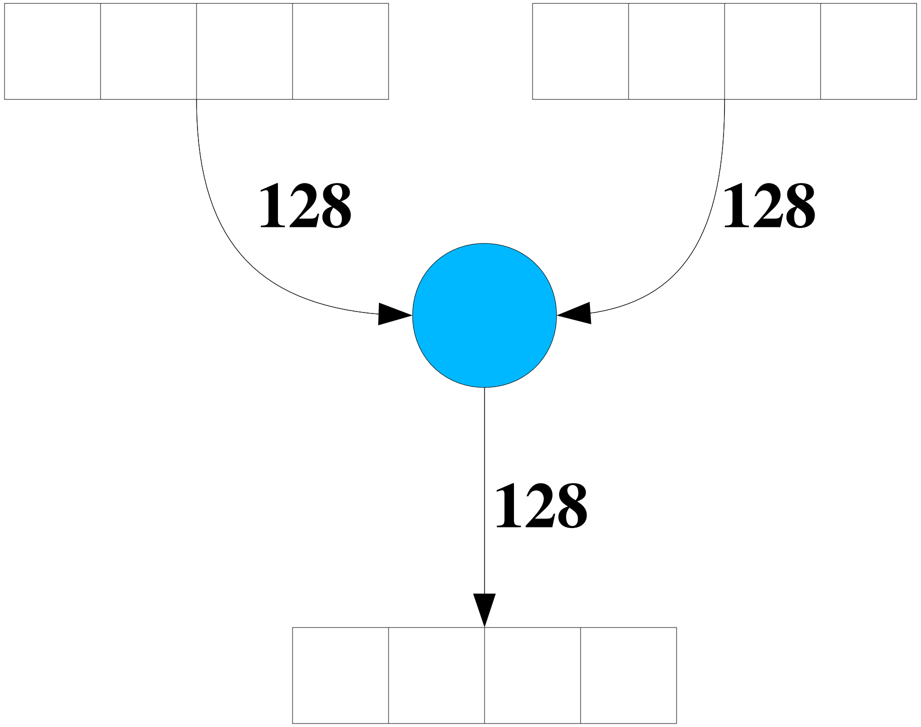
SIMD

- Single-Instruction/Multiple Data
  - Intel: MMX



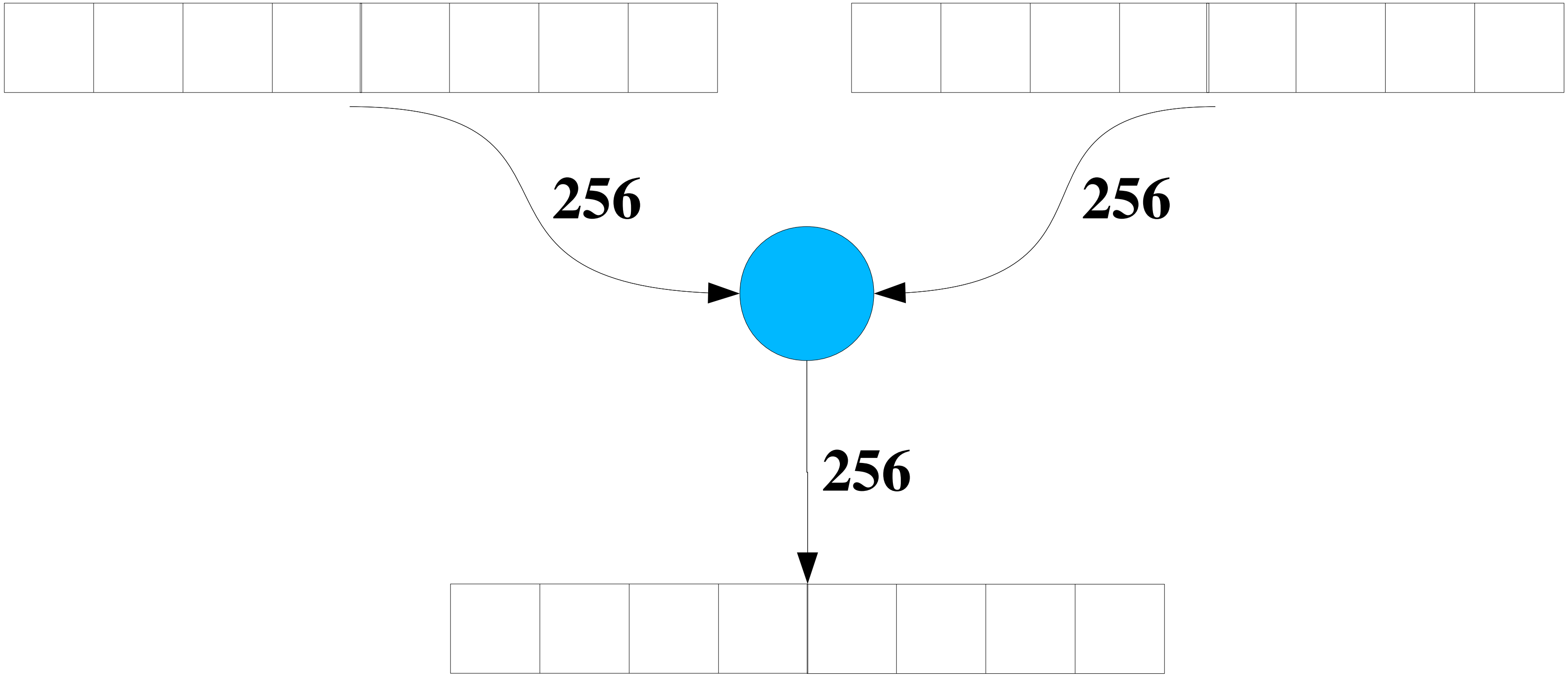
SIMD

- Single-Instruction/Multiple Data
  - Intel: SSE



SIMD

- Single-Instruction/Multiple Data
  - Intel: AVX



Streaming Support

- Covers multiple data types, integer and floating-point
- Incomplete coverage
- Core:
  - Arithmetic
  - For video, audio, and photo processing
- Later extensions:
  - Comparisons
  - Logic operations
  - Vector operations

Non-Streaming Example

- Original code

```
void levelscale(vec_float &dst, const vec_float &src)
{
  for (int i = 0; i < N; ++i)
    if (src[i] > 10)
      dst[i] = 10 + (src[i] - 10) * 9 / 10;
    else
      dst[i] = src[i];
}
```

Streaming Example

- Using SSE

```
void levelscale(vec_float &dst, const vec_float &src)
{
  __m128 v10 = _mm_set_ps1(10.0f);
  __m128 v09 = _mm_set_ps1(0.9f);
  for (int i = 0; i < N / 4; ++i) {
    __m128 cmp = _mm_cmpgt_ps(src.f[i], v10);
    __m128 tmp = _mm_add_ps(v10,
           _mm_mul_ps(_mm_sub_ps(src.f[i], v10),  v09));
    dst.f[i] = _mm_blendv_ps(src.f[i], tmp, cmp);
  }
}
```

Utilize Processor completely

- Determine program factors
  - How much parallelism?
  - Which functional units?
  - Memory use:
    - Working set versus cache size
    - Memory bandwidth requirement
  - Synchronization requirements
  -

Scheduling Decisions

- Two parties responsible:
  - Kernel:
    - Scheduling without insight into program
    - Optimal memory bandwidth
    - Cache sharing
    - Minimal energy use
  - Userlevel
    - Influence scheduling through affinity
    - Needs insight into CPU topology:
      - Connecting caches
      - Socket connections

Core-Memory Gap

	Core	*Array*	
intel inside pentium II	233 MHz	66 MHz	EDO
intel Pentium 4 inside	3.7 GHz	133 MHz to 266 MHz	DDR2
intel CORE i7 inside	3.3 GHz	100 MHz to 233 MHz	DDR3

Core-Memory Gap

	Core	*Array*	*Ratio*
intel inside pentium II	233 MHz	66 MHz	3.5:1
intel Pentium 4 inside	3.7 GHz	133 MHz to 266 MHz	14:1 to 28:1
intel CORE i7 inside	3.3 GHz	100 MHz to 233 MHz	14:1 to 33:1

Memory Handling Decision

- Kernel
  - Implement memory policy
- Userlevel
  - Use cache lines efficiently
  - Use cache levels efficiently
  - Share caches where possible and useful
  - Prefetch cache lines
  - Use local memory
  - Needs insight into memory topology
  - Needs control over memory placement

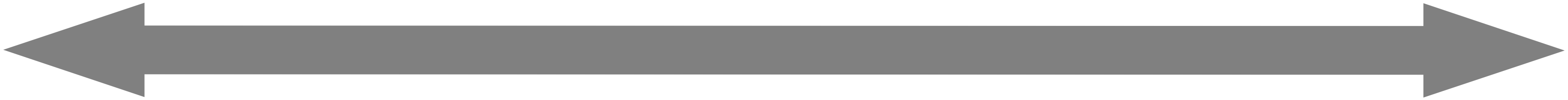
Programming Language Effects

- For best performance:
  - Access the execution units (threads)
  - Access to kernel facilities (affinity)
  - Control over object placement
    - Cache line utilization
    - Alignment issues (cache associativity)
  - Fixed address space regions
    - For node binding

Language Spectrum

Little Control

Much Control



Perl

Python

Java

C

C++

Fortran

Programmer Progress

- OK to use scripting languages
  - Do not expect performance
- Automatic memory handling
  - Nice for fast programming
  - … and safe programs
  - Does not allow memory optimizations
- Start with scripting
- ***Learn about hardware details***
- Replace performance critical parts with C/C++
  - Help through OpenMP, streaming libraries, etc
- Proceed rewriting until performance goal is met

Questions?

drepper@redhat.com | people.redhat.com/drepper