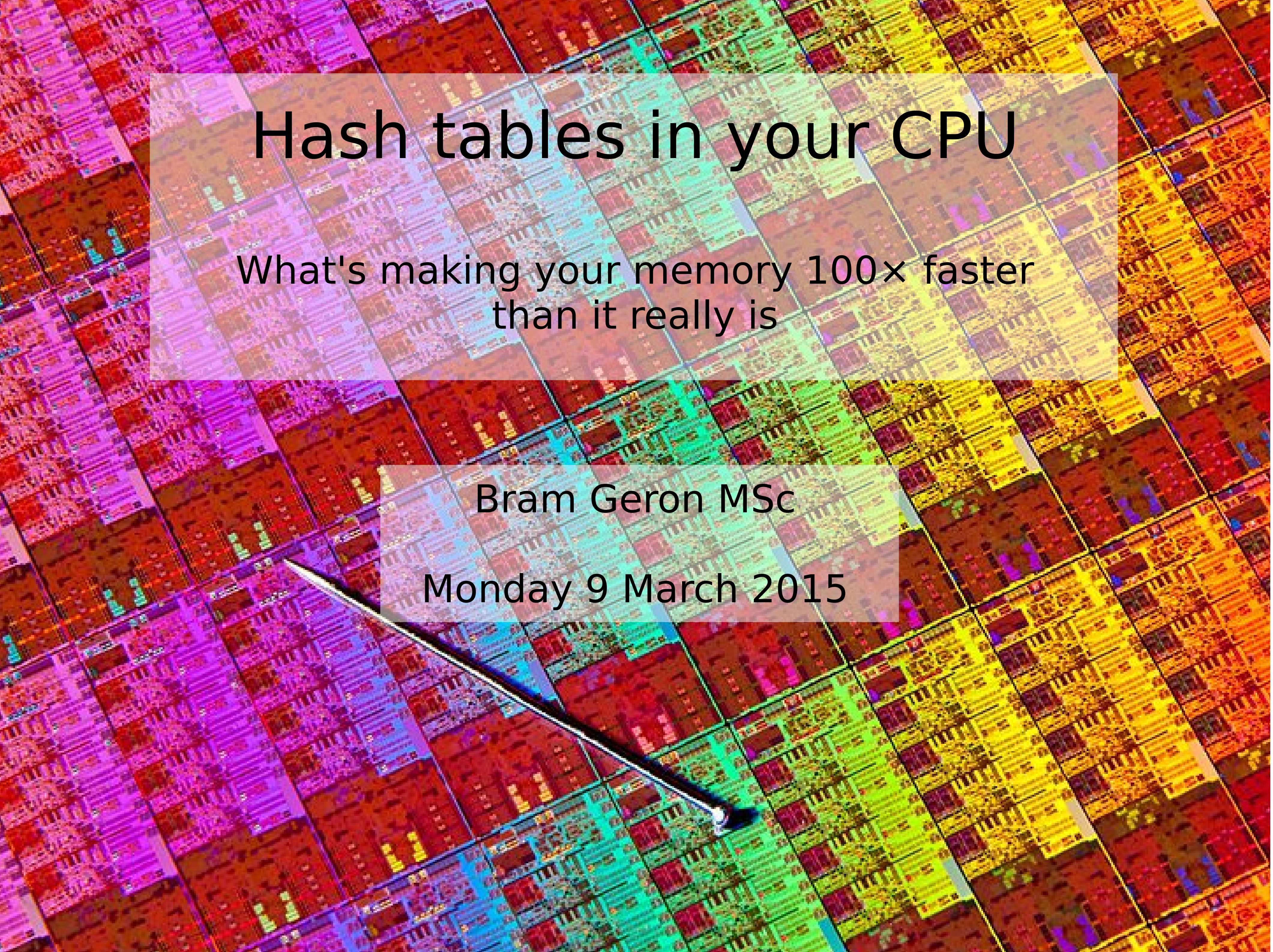


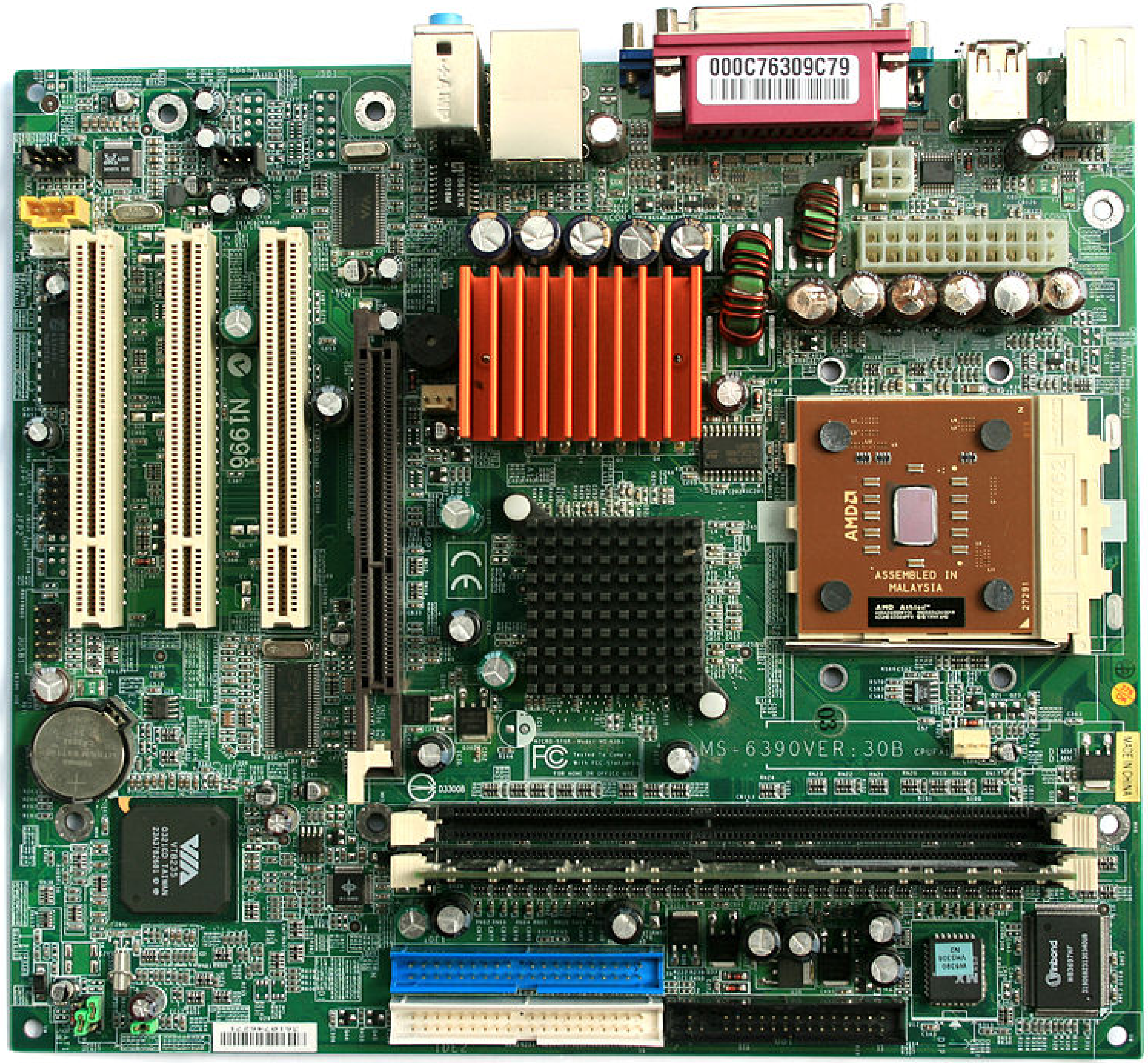
Hash tables in your CPU

What's making your memory 100× faster
than it really is

Bram Geron MSc

Monday 9 March 2015

Computer architecture



Me, myself & I

Bram Geron

Graduated in 2013 from Eindhoven Uni. of Tech., NL

Now: PhD student here

Researching programming languages between imperative and functional

Helping in FoCS, Functional Programming, Data Structures

What to expect from this lecture

- Nothing on exam
- No assignment

You can thank John later.

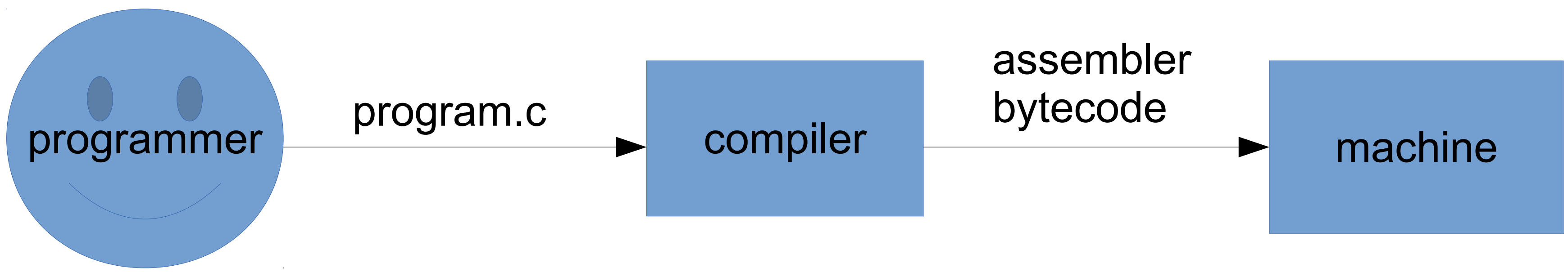
- How a simple program is <u>really</u> executed by a computer
- Why we kind of need a CPU cache
- Recap hash tables
- How caches are implemented

- Rationale and design of a real-world algorithm that uses hash tables

Summing an array of numbers

```
int sum(int array[], int count) {
    int result = 0;
    for (int i = 0; i < count; i++) {
        result += array[i];
    }
    return result;
}
```

What's really going on



Demo: compile!

Summing an array of numbers

```
int sum(int array[], int count) {


    int result = 0;

    for (int i = 0; i < count; i++) {
        result += array[i];

    }
    return result;
}
```

```
int sum(int array[], int count) {

    int pointer* = array;
    int result = 0;

    while (count -= 1; count != 0) {
        result += *pointer;
        pointer = pointer + 1;
    }
    return result;
}
```

Summing an array of numbers

```
int sum(int array[], int count) {
    int pointer* = array;
    int result = 0;


    while (count -= 1; count != 0) {

        result += *pointer;
        pointer = pointer + 1;

    }



    return result;
}
```

```
int sum(int array[], int count)
    pointer = array;
    int result = 0
    if (count == 0) {
        goto end
    }
begin:
    result += *pointer // Look up element
    pointer += 4   // Move one integer right
    count -= 1     // Decrement by one
    if (count != 0) {
        goto begin;
    }
end:
    return result;
}
```

Summing an array of numbers

```
int sum(int array[], int count)
    pointer = array;
    int result = 0
    if (count == 0) {
        goto end
    }
begin:
    result += *pointer // Look up element
    pointer += 4   // Move one integer right
    count -= 1     // Decrement by one
    if (count != 0) {
        goto begin;
    }
end:
    return result;
}
```

```
sum:
    pointer = array
    xor     result, result
    test    count, count
    je      end

begin:
    add     result, [pointer]
    add     pointer, 4
    dec     count
    jne     begin


end:
    ret    result
```

Summing an array of numbers

```
sum:
    pointer = array
    xor     result, result
    test    count, count
    je      end

begin:
    add     result, [pointer]
    add     pointer, 4
    dec     count
    jne     begin


end:
    ret    result
```

```
sum:

    xor     eax, eax
    test    rsi, rsi
    je      end

begin:
    add     eax, dword ptr [rdi]
    add     rdi, 4
    dec     rsi
    jne     begin


end:
    ret
```

Instruction types

1. Set register to a value
2. If a register is 0, jump to another location in the code
3. Add a memory location to a register
4. Add 4 to a register
5. Subtract 1 from a register
6. Exit from the function

All of them are really really fast, except for one.

This is due to chemical properties of memory.

The processor memory gap

[picture removed for copyright reasons]

This picture shows that from 1980 to 2000, CPU speeds increased ~1000×, while RAM speed only increased ~5×.

See http://dx.doi.org/10.1109/40.592312 page 2 ; SRAM = Static RAM follows the CPU line.

From: Patterson et al, A Case for Intelligent RAM, in: IEEE Micro 17:2, pp 34.

Timings on modern computers

Approximate timing for various operations on a typical PC:

Task	Time
execute typical instruction	1/1,000,000,000 sec = **1 nanosec**
fetch from L1 cache memory	**0.5 nanosec**
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	**100 nanosec**
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

Read 16 bytes sequentially: ~1ns

Source: Peter Norvig, Teach yourself programming in ten years, http://norvig.com/21-days.html

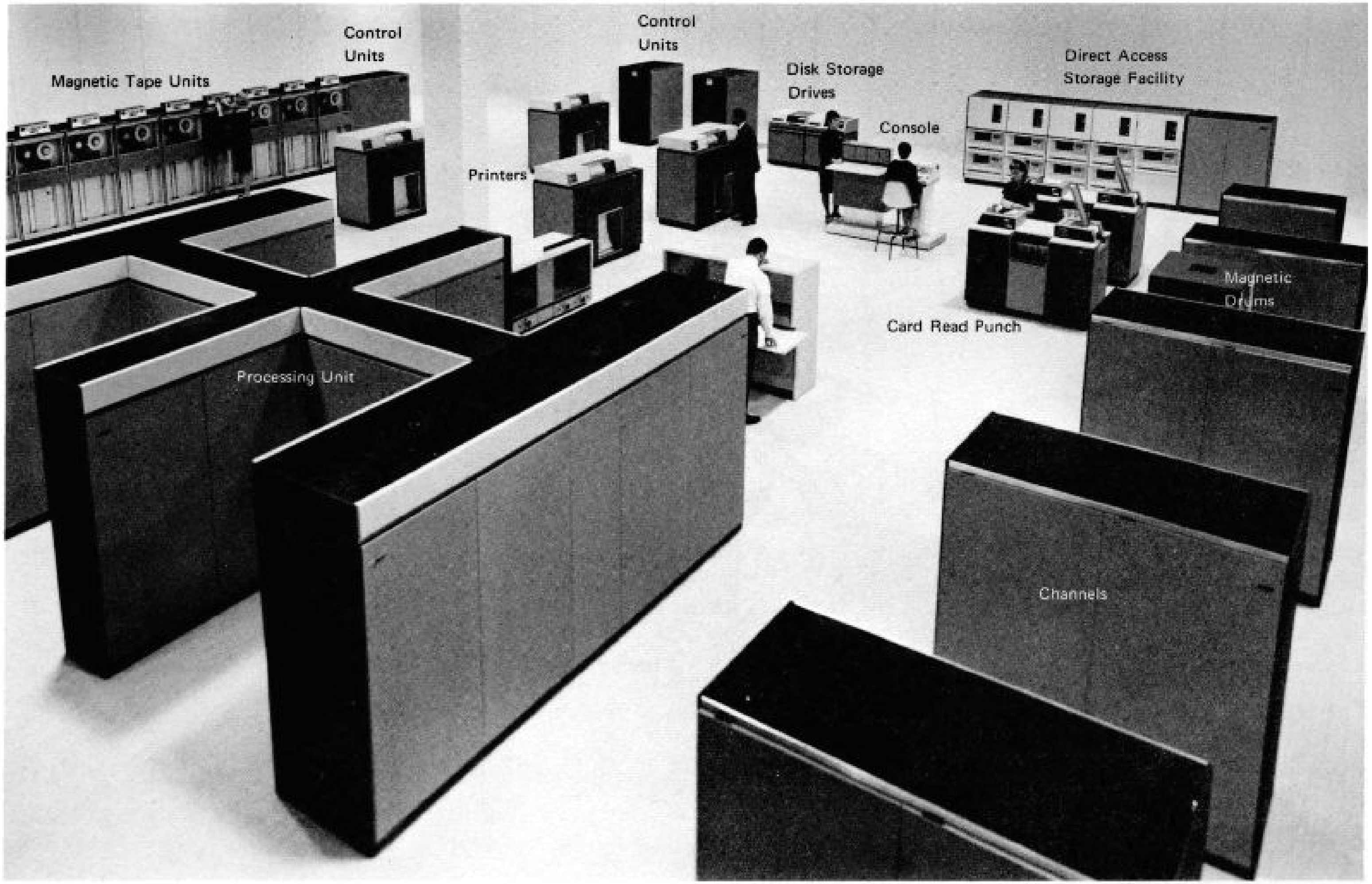
Memory cache: the idea

- Memory accesses are usually close together
- Put recently-accessed memory in faster memory

Requirements:

- Store various fragments of main memory
- Be super fast

1969: introduction of “buffer storage”: 16 kB



IBM System/360 Model 85

ARK | Intel® Core™ i7-5550U Processor (4M Cache, up to 3.00 GHz) - Chromium

ARK | Intel® Cor ×

ark.intel.com/products/84992/Intel-Core-i7-5550U-Processor-4M-Cache-up-to-3_00-GHz?_ga=1.16197017

intel Menu Find Content | Search USA (English)

Type Here to Search Products

ARK Menu ▶

Intel® Core™ i7-5550U Processor (4M Cache, up to 3.00 GHz)

Compare +

Specifications
- Essentials
- Performance
- Memory Specifications
- Graphics Specifications
- Expansion Options
- Package Specifications
- Advanced Technologies
- Intel® Data Protection Technology
- Intel® Platform Protection Technology

Ordering / sSpecs / Steppings

Specifications

Essentials

Status	Launched
Launch Date	Q1'15
DMI2	5 GT/s
Processor Number	i7-5550U
Intel® Smart Cache	4 MB
Instruction Set	64-bit
Instruction Set Extensions	SSE4.1/4.2, AVX 2.0
Embedded Options Available	No
Lithography	14 nm
Recommended Customer Price	TRAY: $426.00
Datasheet	Link
Conflict Free	Yes

Performance

# of Cores	2

Related Products
- 5th Generation Intel® Core™ i7 Processors
- Intel® Core™ i7-5500 Mobile Processor Series
- Products formerly Broadwell

Learn how Intel is pursuing conflict-free technology. >

Quick Links
- Export Full Specifications >
- Search Distributors >
- Support Overview >
- Search all of intel.com >

Hash tables: recap

- Set of keys
- Usually: set of values
- A size *s*
- A hash function
  from keys to {0, .., *s*-1}
- A collision strategy:
  none / direct chaining
  / linear probing / double hashing
  / buckets

(example)

Num	Memory
0	false
1	true
2	first prime
4	$= 2+2 = 2 \times 2 = 2^2 = ...$
6	perfect
7	first non-fib prime
42	answer to something important
314	a bit like π
1989	I was born
2015	now

Hash table: CPU cache

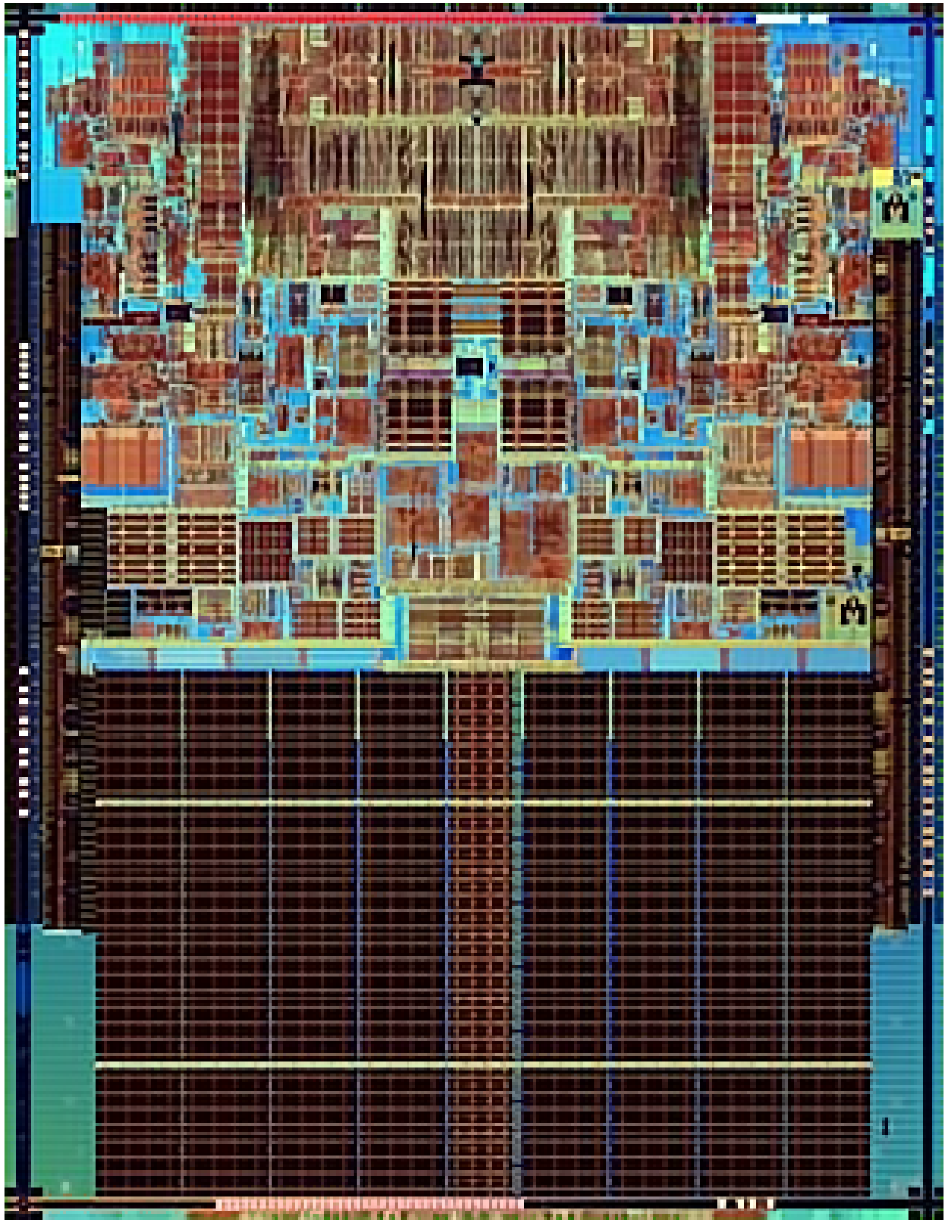
Keys: memory block numbers
Hash: last digit
Value: block of memory
Collision strategy: none

(example)

Block no = address **div** 10
Hash function = block no **mod** 10

To read a byte of memory:

- Look in the CPU cache in the corresponding hash table slot
- If it’s not there:
  - read from DRAM (slow) a memory block, and
  - replace the entry in the CPU cache
- Look up the right bytes from the cache



Picture: Intel Conroe die, probably in Core 2 Duo

2-way associative CPU cache

Keys: memory block numbers
Hash: last digit
Value: time of access + block of memory
Collision strategy: buckets

(example)

Block no = still address **div** 10
Hash function = still block no **mod** 10

But uses more memory!

In the same way: 4-way associative cache, 8-way associative.

To read a byte of memory:

- Look in the CPU cache in the corresponding hash table slot
- If it's not there:
  - Choose a block to **evict** from the cache
  - Read from DRAM (slow) a memory block, and
  - Replace the entry in the CPU cache
- Look up the right bytes from the cache
- Update the time last accessed

Caches, caches, caches

Typical recent-ish system:

- Level 1 cache: 64 kB[1], 4-way associative[2]
- Level 2 cache: 256 kB[1], 8-way associative[2]

  (Same block size*, more entries = different hash fn)
- Level 3 cache: 8 MB[1], 8-way associative[3]

** Blocks are actually called cache lines.*

[1] For Core 2.
http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/2nd-gen-core-desktop-vol-1-datasheet.pdf

[2] For original Pentium 4. Gene Cooperman, "Cache basics", 2003.
http://www.ccs.neu.edu/course/com3200/parent/NOTES/cache-basics.html

[3] Guesstimate.

Thanks for your attention!

That's all, folks. Hope you learned something.

Questions?

Ask me anything.

Feedback is welcome at http://bram.xyz/feedback .

Acknowledgements

- Haswell wafer photo CC BY 2.0 from James086 and Intel Free Press
- Motherboard photo by Jonathan Zander (photography.jznet.org), CC BY-SA 3.0
- Conroe die: supposedly public domain, https://commons.wikimedia.org/wiki/File:Conroe_die.png, uploaded by Xoqolatl
- CPU/memory graph from DOI 10.1109/40.592312 (“A case for Intelligent RAM” by Patterson et al), who reference J.L. Hennessy and D.A. Patterson, Computer Organization and Design, 2nd ed., Morgan Kaufmann Publishers, San Francisco, 1997.