CS/ECE 3810: Computer Systems Architecture

Lecture 1: Introduction

Anton Burtsev
September, 2022

Class details

- Undergraduate
  - 203 students
- Instructor: Anton Burtsev

Who am I?

- I build operating systems
  - Since I was your age, i.e., since 2000
- Bits of L4 microkernel, micro-ITRON, XenTT, LCDs, KSplit
  - https://www.cs.utah.edu/~aburtsev/

Prospective students

I am looking for students interested in operating systems at all levels from undergraduate to PhD, if you have relevant skills send me an email.

We are building three new operatings systems

RedLeaf: a clean-slate operating system in Rust designed to support formal verification of functional correctness (project web page).

Redshift: a new operating system aimed to support heterogeneous hardware, e.g., FPGAs, GPUs, TPUs, near storage, and near network cores, etc., as first class citizens (project web page).

Horizon: a new secure hypervisor and secure cloud in which users own their data. Horizon is developed in Rust, and relies on novel techniques of hardware and software isolation, and will implement cloud-wide information flow control (project web page).

Class details

- Undergraduate
  - 203 students
- Instructor: Anton Burtsev
- Meeting time: 11:50am-1:10pm (Mon/Wed)
- 4 TAs
  - Send us a private message on Piazza
- Web page
  - https://www.cs.utah.edu/~aburtsev/3810

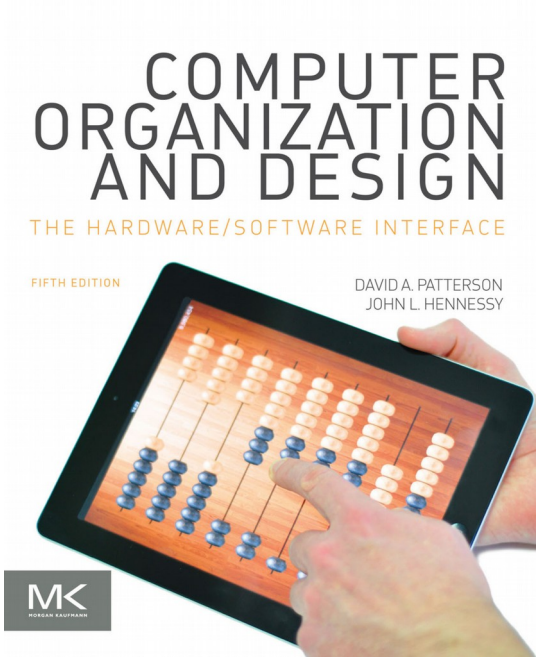
More details

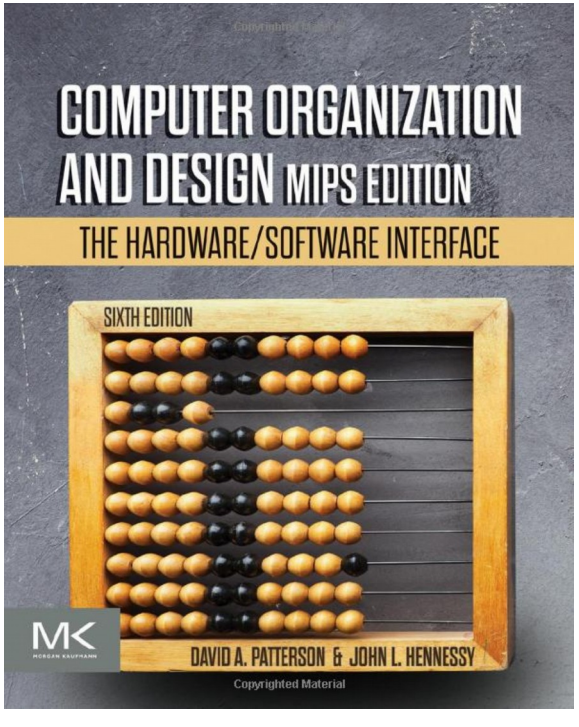
- 8-10 small homework assignments
- 1-2 lab-like assignments
  - Requires basics familiarity with UNIX
  - Shell, C
- Midterm
- Final
- Grades are curved
  - Homework: 30%, midterm exam: 30%, final exam: 40% of your grade.
  - You can drop two assignments
  - Late submissions are 0

This course

- Book: Hennessy and Patterson's
  - Computer Organization and Design

- Topics
  - Understanding performance/cost/power
  - Assembly language
  - Computer arithmetic
  - Pipelining
  - Using predictions
  - Memory hierarchies
  - Accelerators
  - Reliability and Security



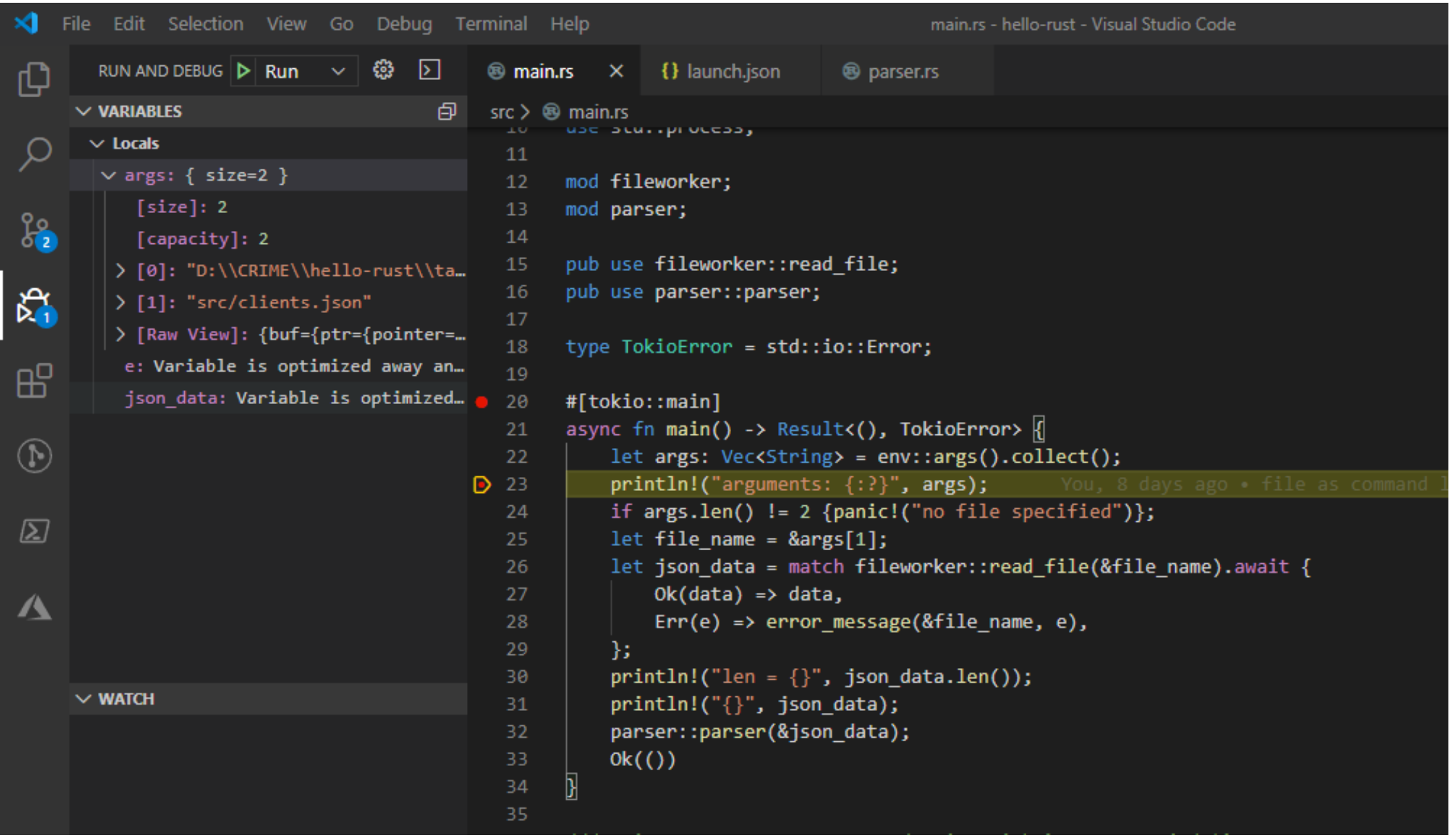


Course organization

- Lectures
  - High level concepts and abstractions
  - Recorded (and hopefully live stream)
- Reading
  - Hennessy and Patterson
  - Bits of additional notes
- Homeworks
- Exams
  - Open notes (but might include open questions)

Questions?

Why this class?


File Edit Selection View Go Debug Terminal Help
main.rs - hello-rust - Visual Studio Code
RUN AND DEBUG
Run
VARIABLES
Locals
args: { size=2 }
[size]: 2
[capacity]: 2
[0]: "D:\\CRIME\\hello-rust\\ta…
[1]: "src/clients.json"
[Raw View]: {buf={ptr={pointer=…
e: Variable is optimized away an…
json_data: Variable is optimized…
WATCH
main.rs
launch.json
parser.rs
src > main.rs
use std::process;
mod fileworker;
mod parser;
pub use fileworker::read_file;
pub use parser::parser;
type TokioError = std::io::Error;
#[tokio::main]
async fn main() -> Result<(), TokioError> {
let args: Vec<String> = env::args().collect();
println!("arguments: {:?}", args);
You, 8 days ago • file as command l
if args.len() != 2 {panic!("no file specified")};
let file_name = &args[1];
let json_data = match fileworker::read_file(&file_name).await {
Ok(data) => data,
Err(e) => error_message(&file_name, e),
};
println!("len = {}", json_data.len());
println!("{}", json_data);
parser::parser(&json_data);
Ok(())
}

But can you do it fast?
… or secure

Example 1: Database Join

Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware

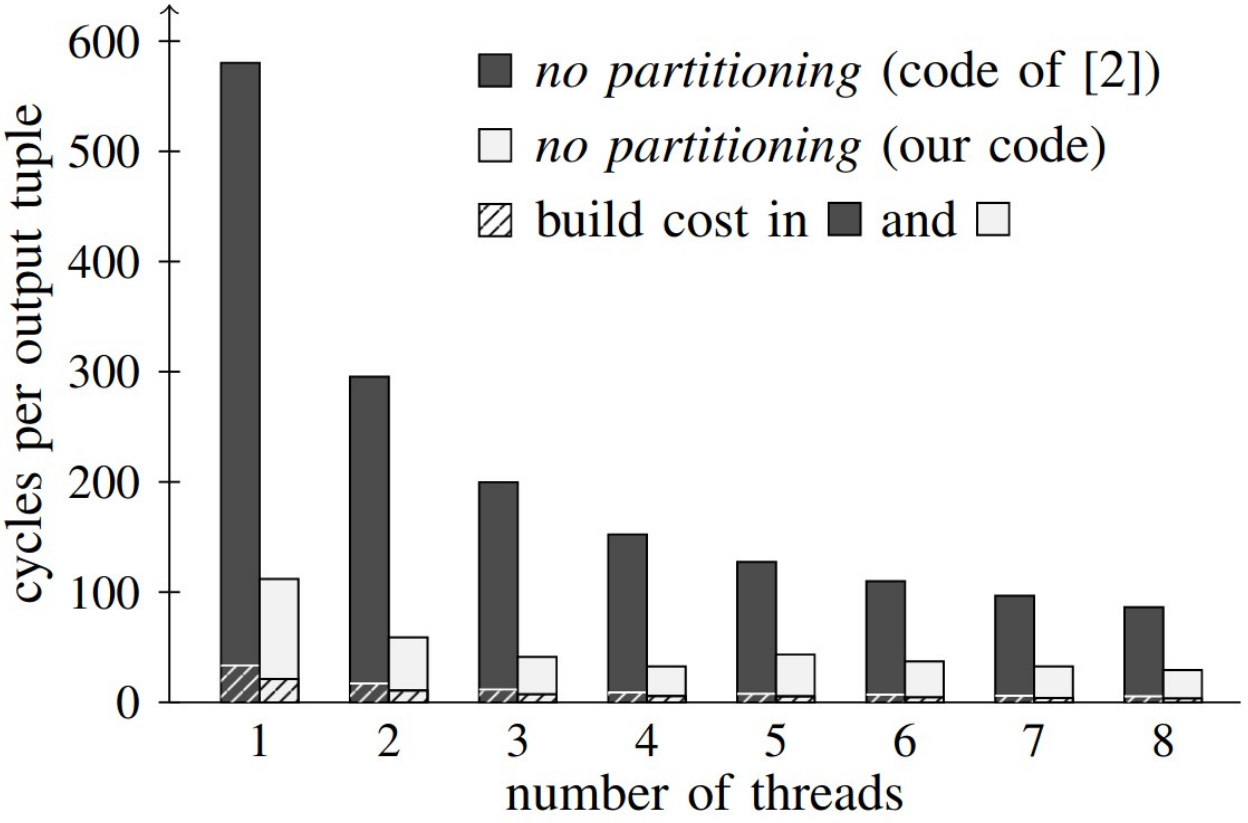
Cagri Balkesen [#1], Jens Teubner [#2], Gustavo Alonso [#3], M. Tamer Özsu [*4]

[#] *Systems Group, Department of Computer Science, ETH Zurich, Switzerland*
[1, 2, 3] {name.surname}@inf.ethz.ch

[*] *University of Waterloo, Canada*
[4] tamer.ozsu@uwaterloo.ca

- https://15721.courses.cs.cmu.edu/spring2016/papers/balkesen-icde2013.pdf

Example 1: Database Join



Example 1: Database Join

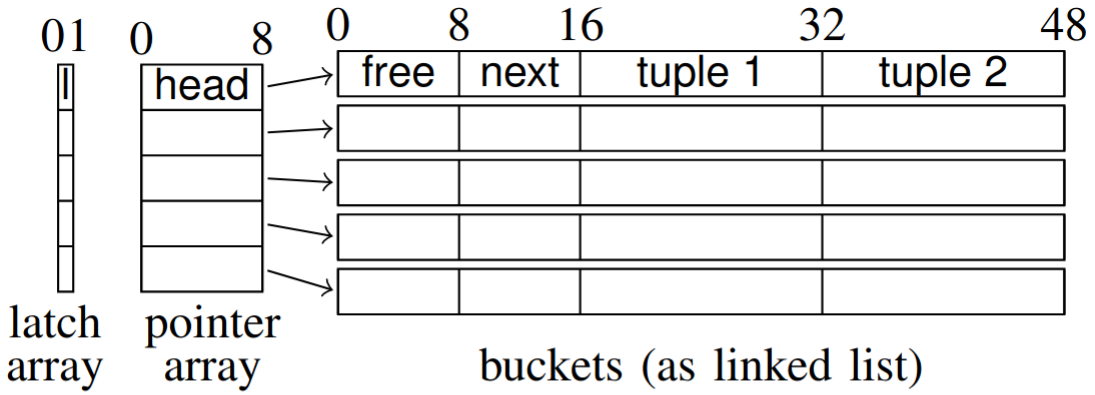


Fig. 6. Original hash table implementation.

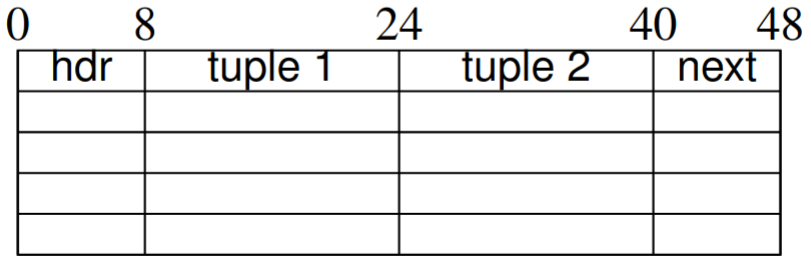


Fig. 7. Our hash table implementation.

Example 2: Virtualization

Virtualization Without Direct Execution or Jitting: Designing a Portable Virtual Machine Infrastructure

Darek Mihocka
Emulators
darekm@emulators.com

Stanislav Shwartsman
Intel Corp.
stanislav.shwartsman@intel.com

- https://bochs.sourceforge.io/Virtualization_Without_Hardware_Final.pdf

Example 2: Virtualization

	1000 MHz Pentium III	2533 MHz Pentium 4	2666 MHz Core 2 Duo
Bochs 2.3.5	882	595	180
Bochs 2.3.6	609	533	157
Bochs 2.3.7	457	236	81

Table 3.1: Windows XP boot time on different hosts

Example 2: Virtualization

3.3 Host branch misprediction as biggest cause of slow emulation performance

Every pipelined processor features branch prediction logic used to predict whether a conditional branch in the instruction flow of a program is likely to be taken or not. Branch predictors are crucial in today's modern, superscalar processors for achieving high performance.

Modern CPU architectures implement a set of sophisticated branch predictions algorithms in order to achieve highest prediction rate, combining both static and dynamic prediction methods. When a branch instruction is executed, the branch history is stored inside the processor. Once branch history is available, the processor can predict branch outcome – whether the branch should be taken and the branch target.

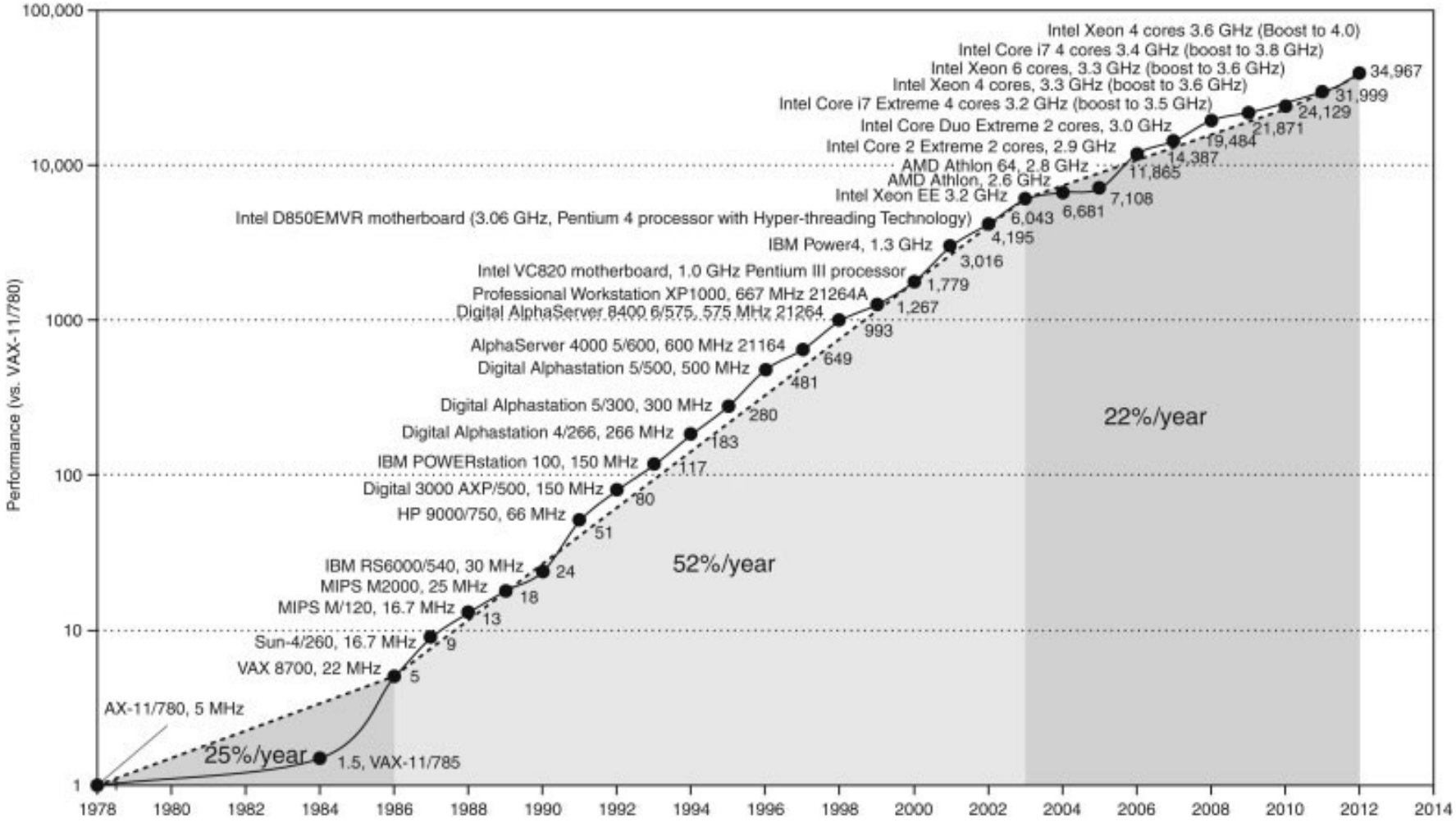
A typical Bochs instruction handler method:

```
void BX_CPU_C::SUB_EdGd(bxInstruction_c *i)
{
  Bit32u op2_32, op1_32, diff_32;

  op2_32 = BX_READ_32BIT_REG(i->nnn());

  if (i->modC0()) {    // reg/reg format
    op1_32 = BX_READ_32BIT_REG(i->rm());
    diff_32 = op1_32 - op2_32;
    BX_WRITE_32BIT_REGZ(i->rm(), diff_32);
  }
  else {               // mem/reg format
    read_RMW_virtual_dword(i->seg(),
        RMAddr(i), &op1_32);
    diff_32 = op1_32 - op2_32;
    Write_RMW_virtual_dword(diff_32);
  }
  SET_LAZY_FLAGS_SUB32(op1_32, op2_32,
        diff_32);
}
```

Listing 3.1: A typical Bochs instruction handler

Why is it so hard?

Microprocessor Performance



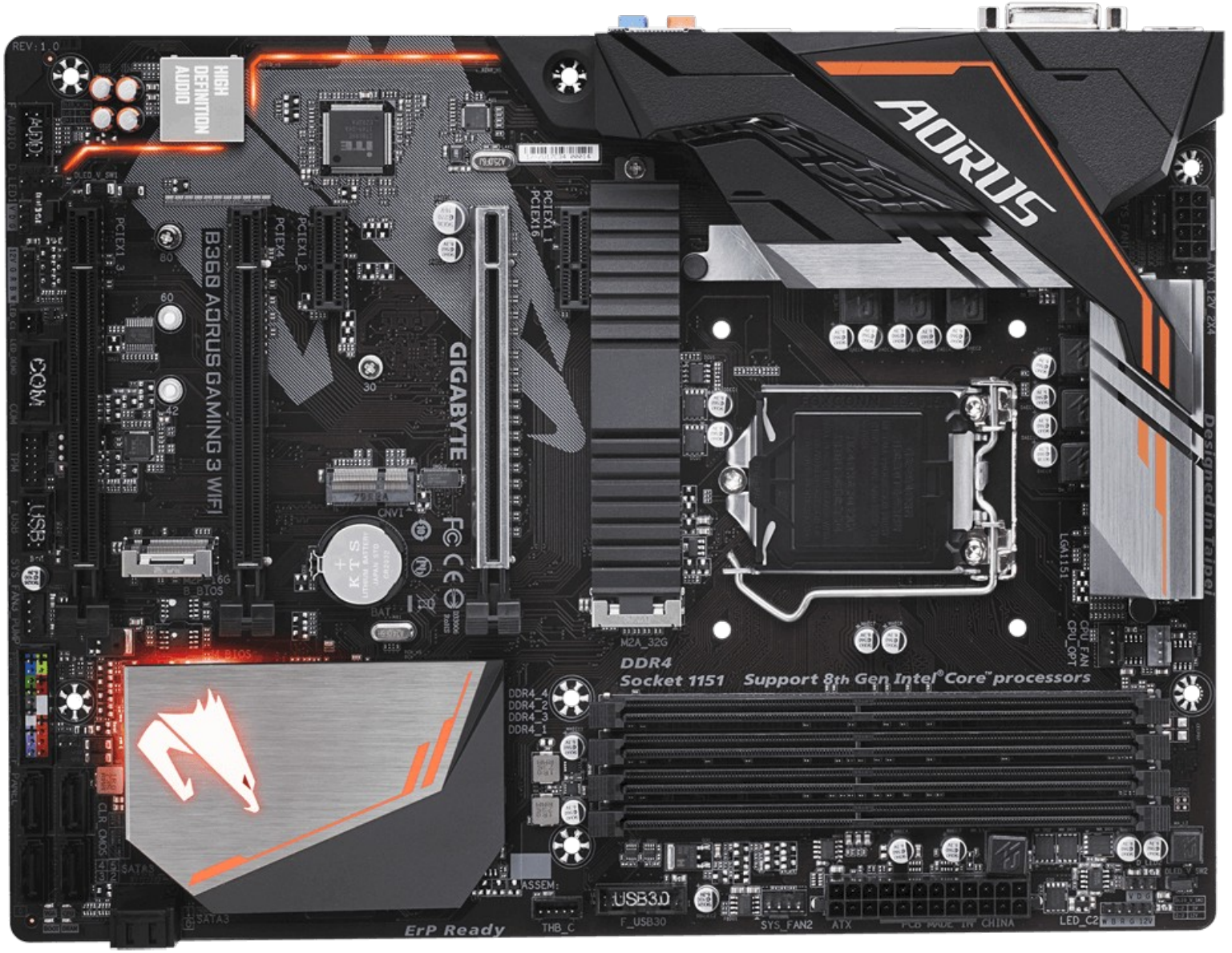
Source: H&P Textbook

50% improvement every year!!

What contributes to this improvement?

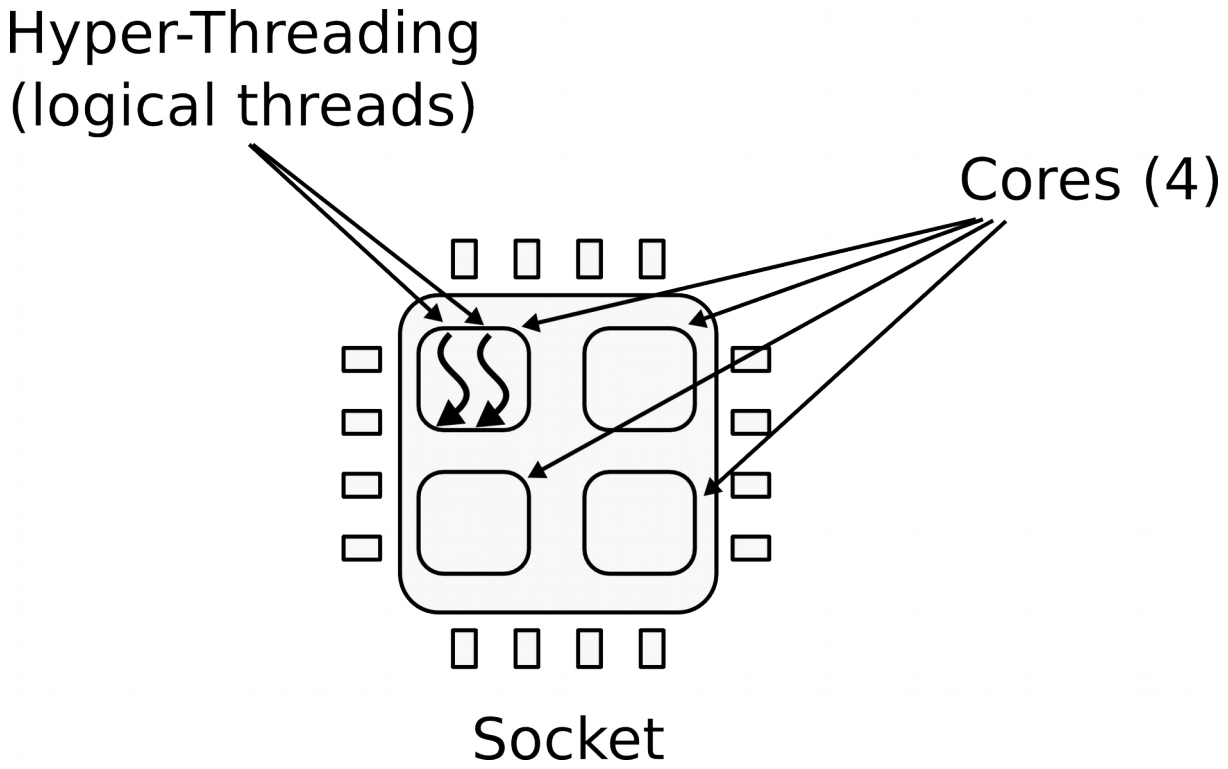
What’s inside a typical machine?

B360 AORUS Motherboard

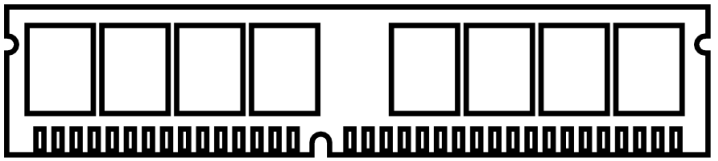


CPU

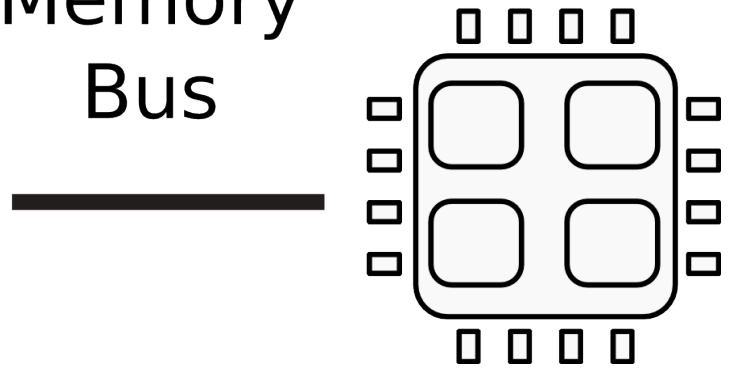
- 1 CPU socket
  - 4 cores
  - 2 logical threads each (hyperthreads)

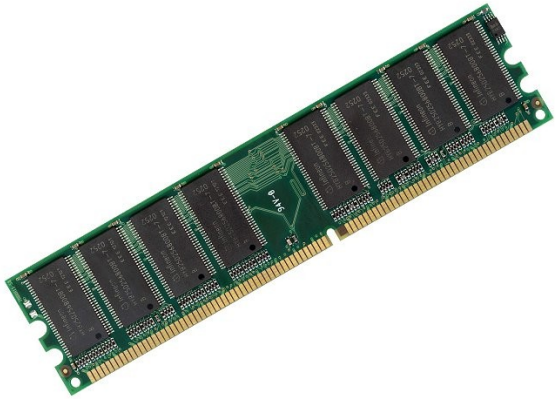


Memory



Memory
Bus

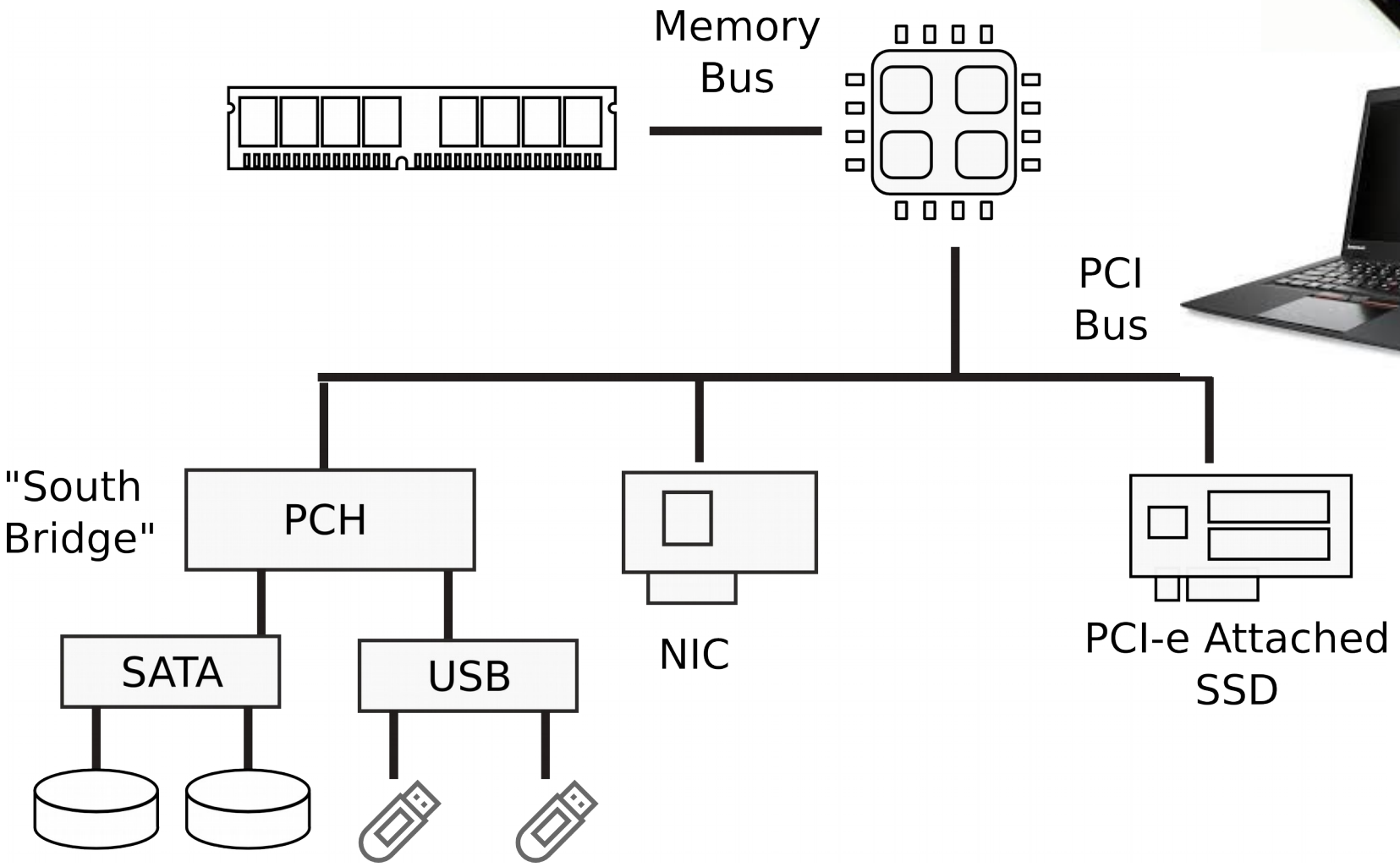




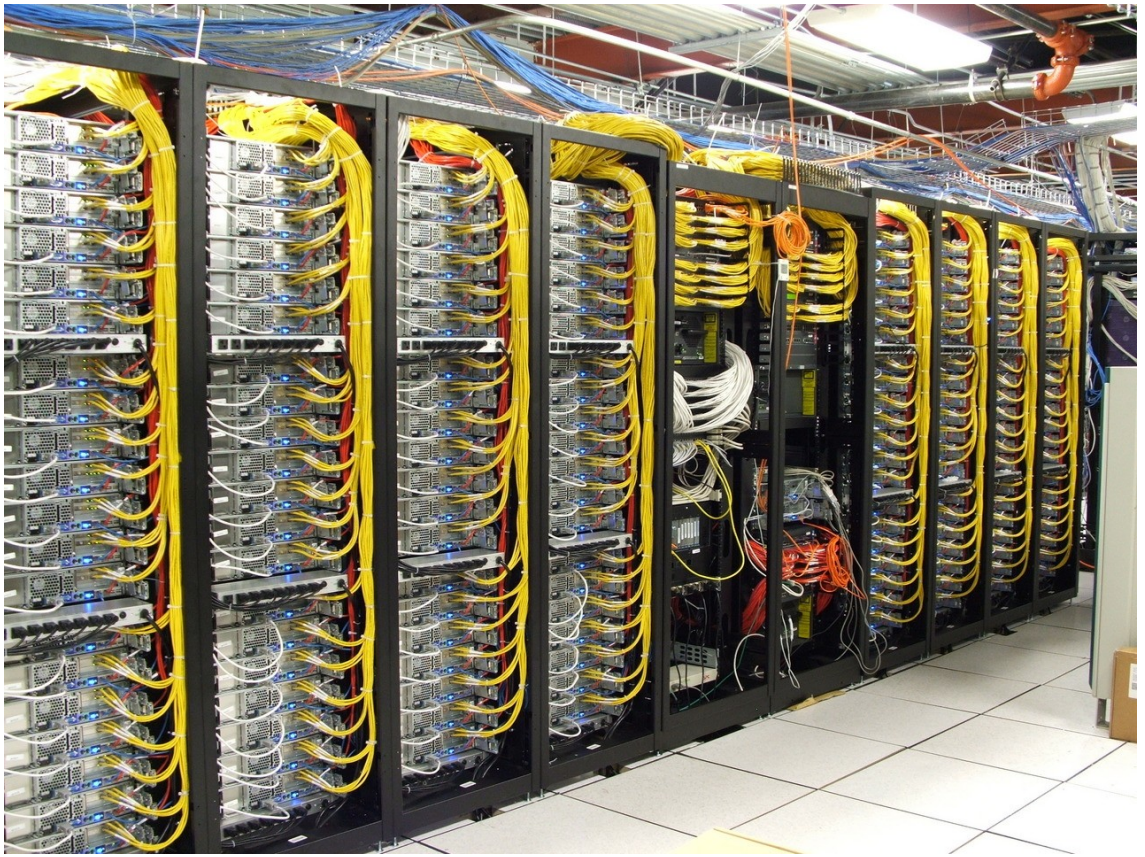
Memory abstraction

WRITE(*addr*, *value*) $\rightarrow \varnothing$ Store *value* in the storage cell identified by *addr*.
READ(*addr*) $\rightarrow$ *value* Return the *value* argument to the most recent WRITE call referencing *addr*.

I/O Devices


Memory
Bus
PCI
Bus
"South
Bridge"
PCH
SATA
USB
NIC
PCI-e Attached
SSD

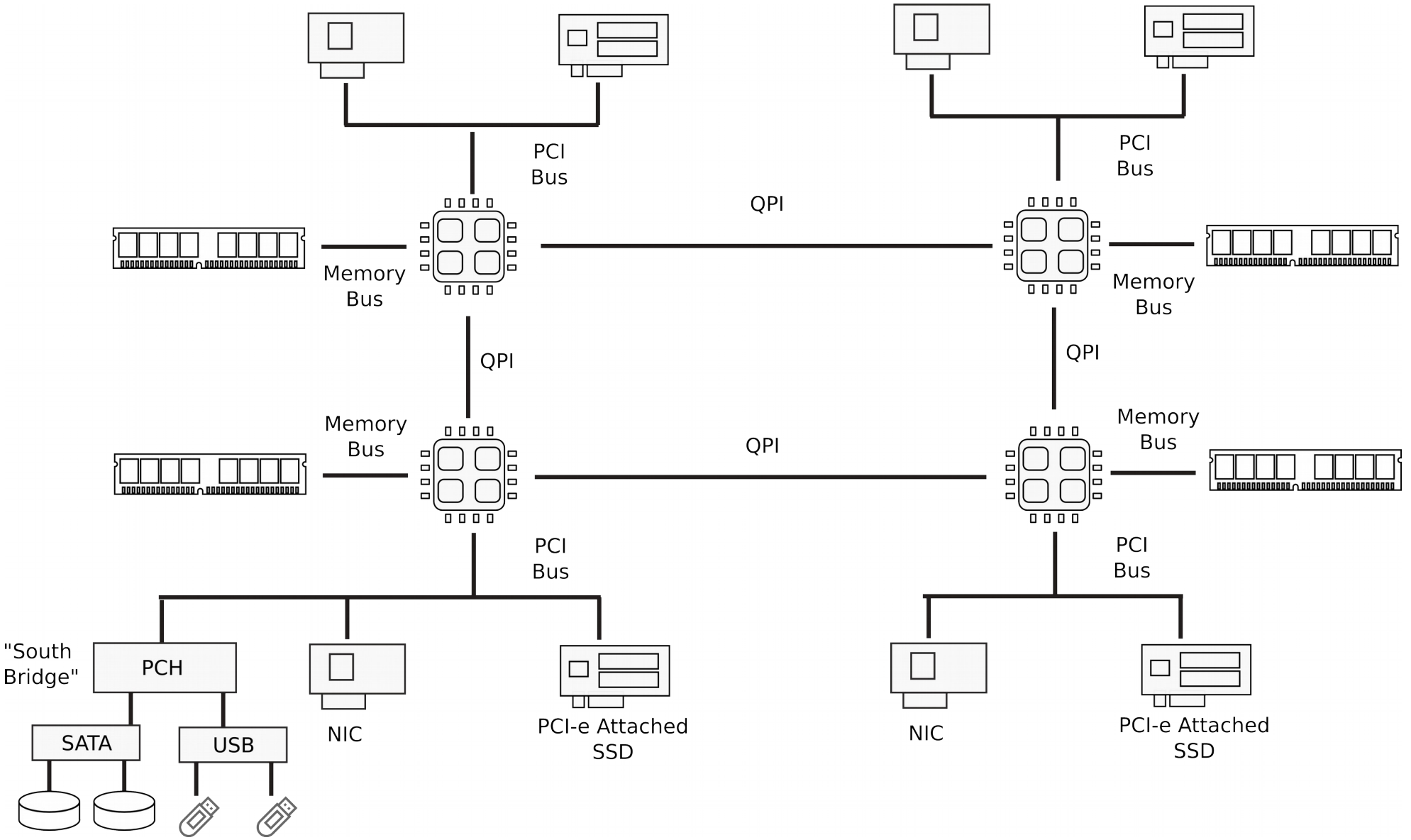
Dell R830 4-socket server



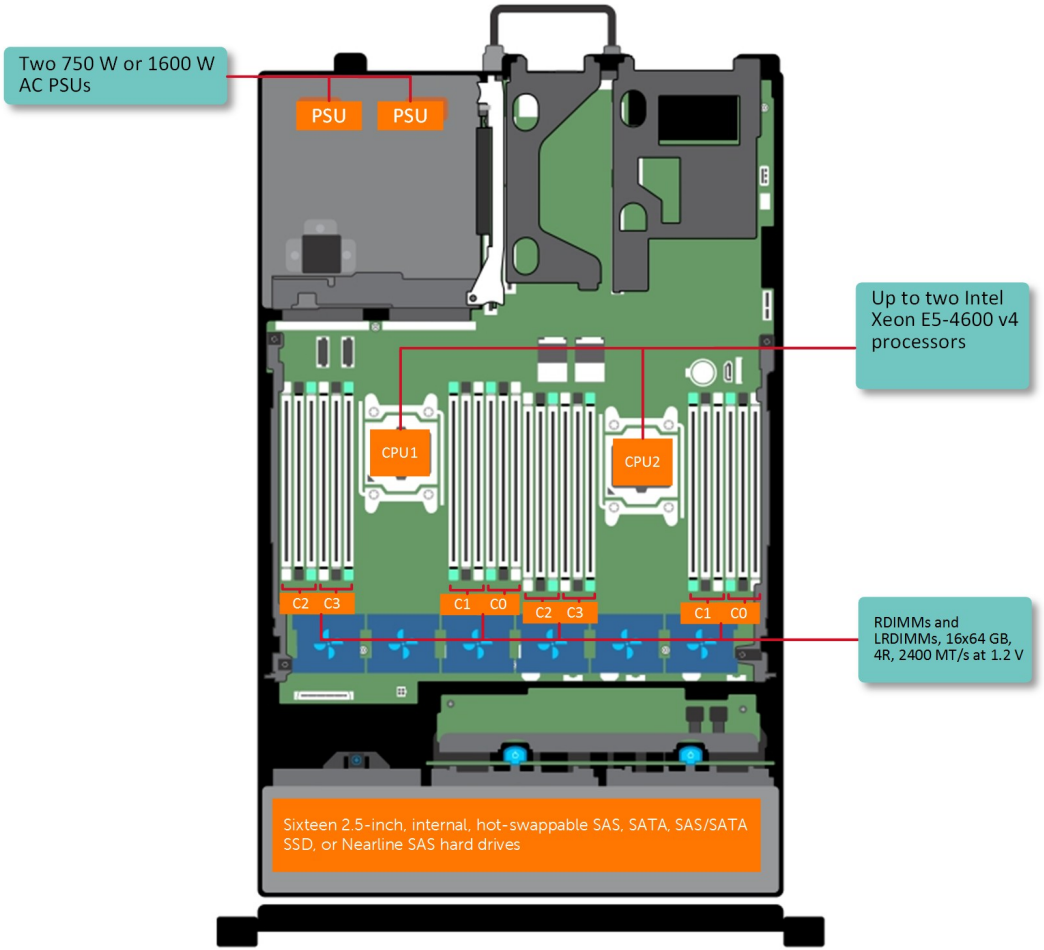
Dell Poweredge R830 System Server with 2 sockets
on the main floor and 2 sockets on the expansion

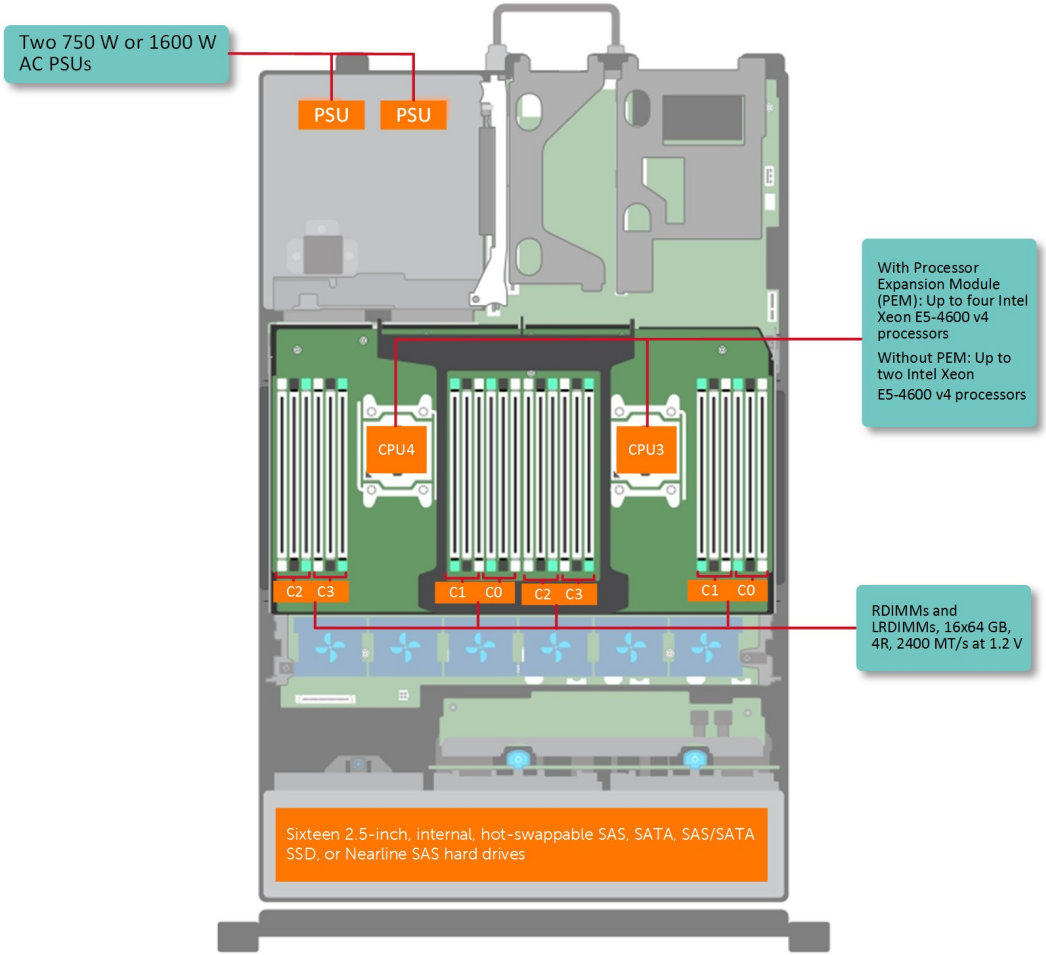
http://www.dell.com/support/manuals/us/en/19/poweredge-r830/r830_om/supported-configurations-for-the-poweredge-r830-system?guid=guid-01303b2b-f884-4435-b4e2-57bec2ce225a&lang=en-us

Multi-socket machines



Dell R830 4-socket server

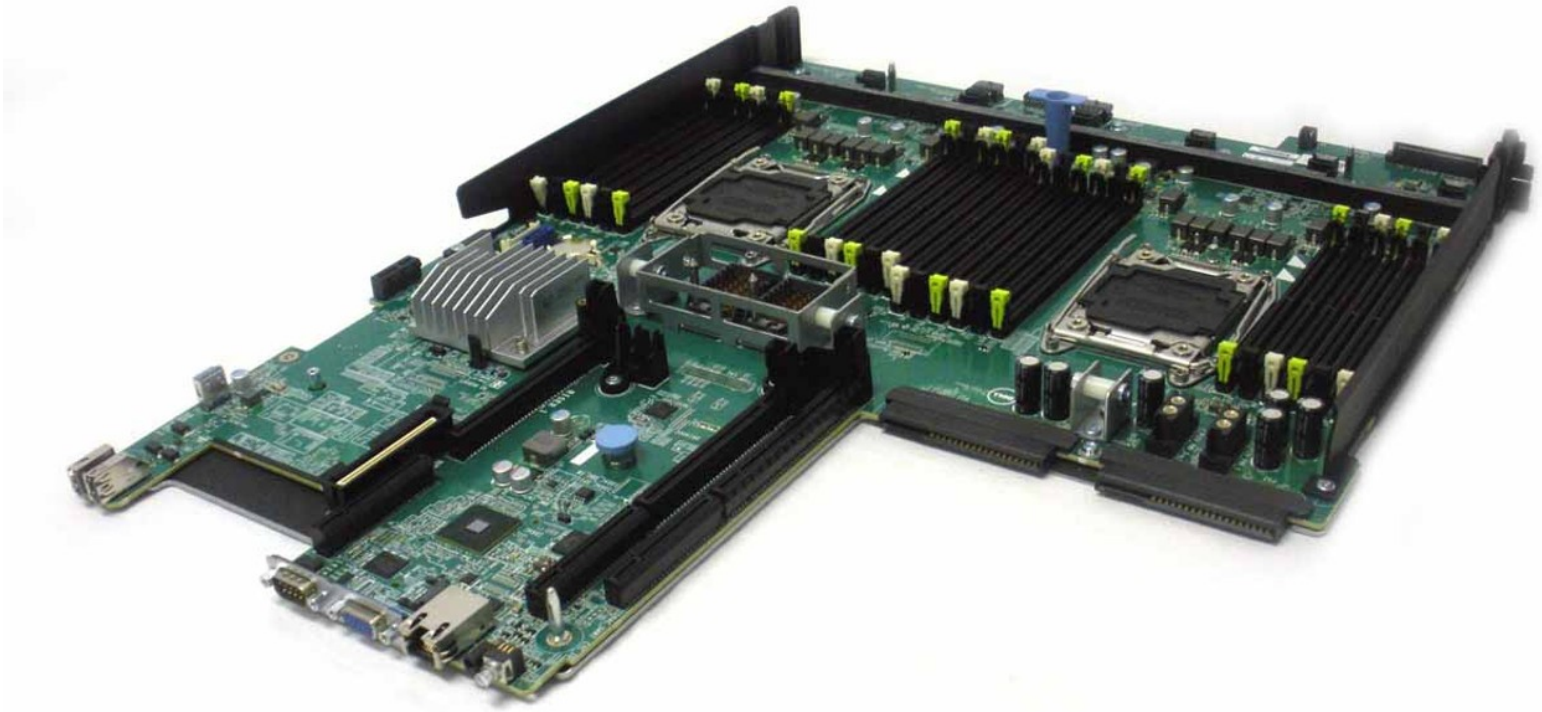




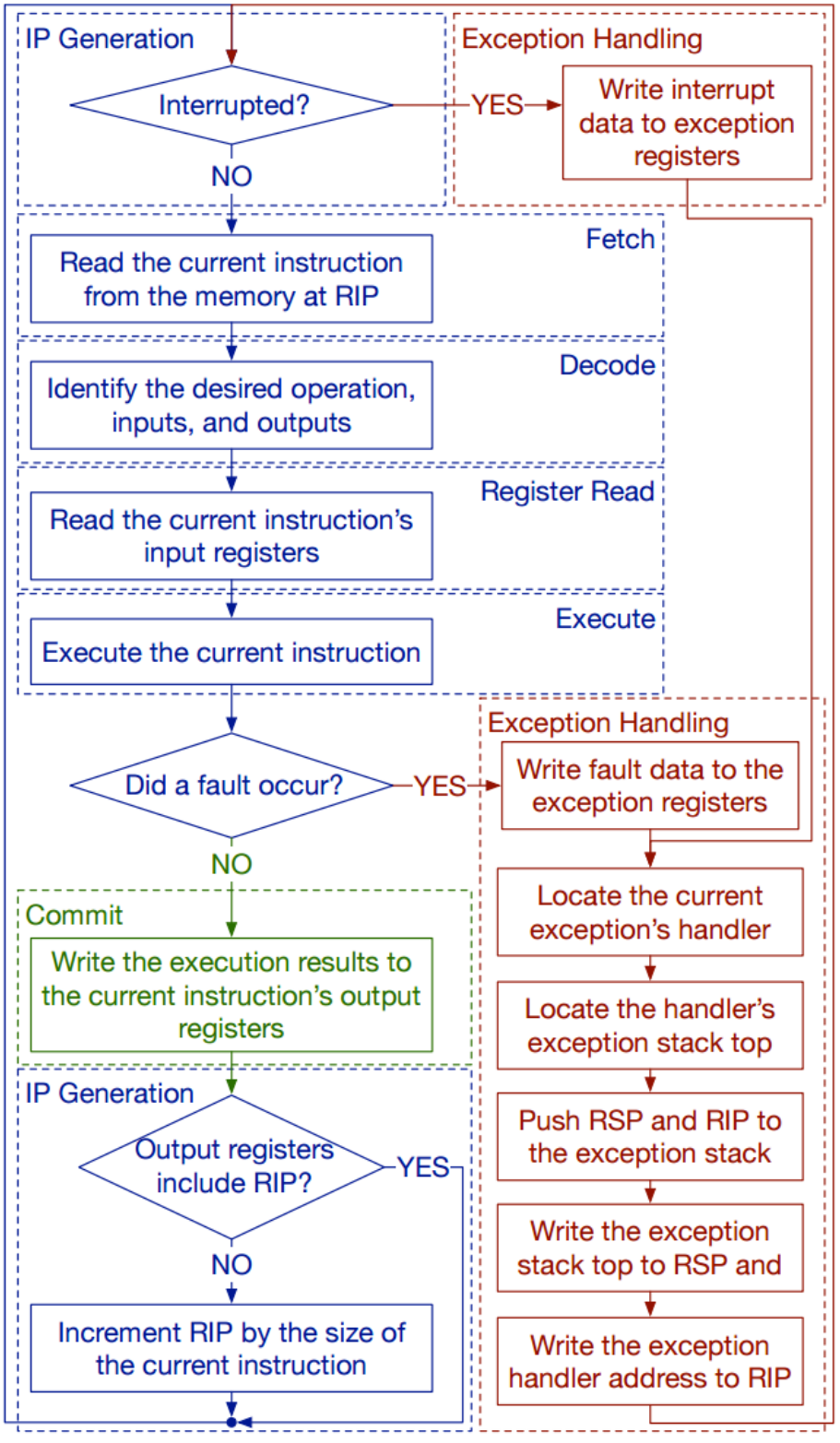
Dell Poweredge R830 System Server with 2 sockets on the main floor and 2 sockets on the expansion

http://www.dell.com/support/manuals/us/en/19/poweredge-r830/r830_om/supported-configurations-for-the-poweredge-r830-system?guid=guid-01303b2b-f884-4435-b4e2-57bec2ce225a&lang=en-us

Dell R830 Motherboard



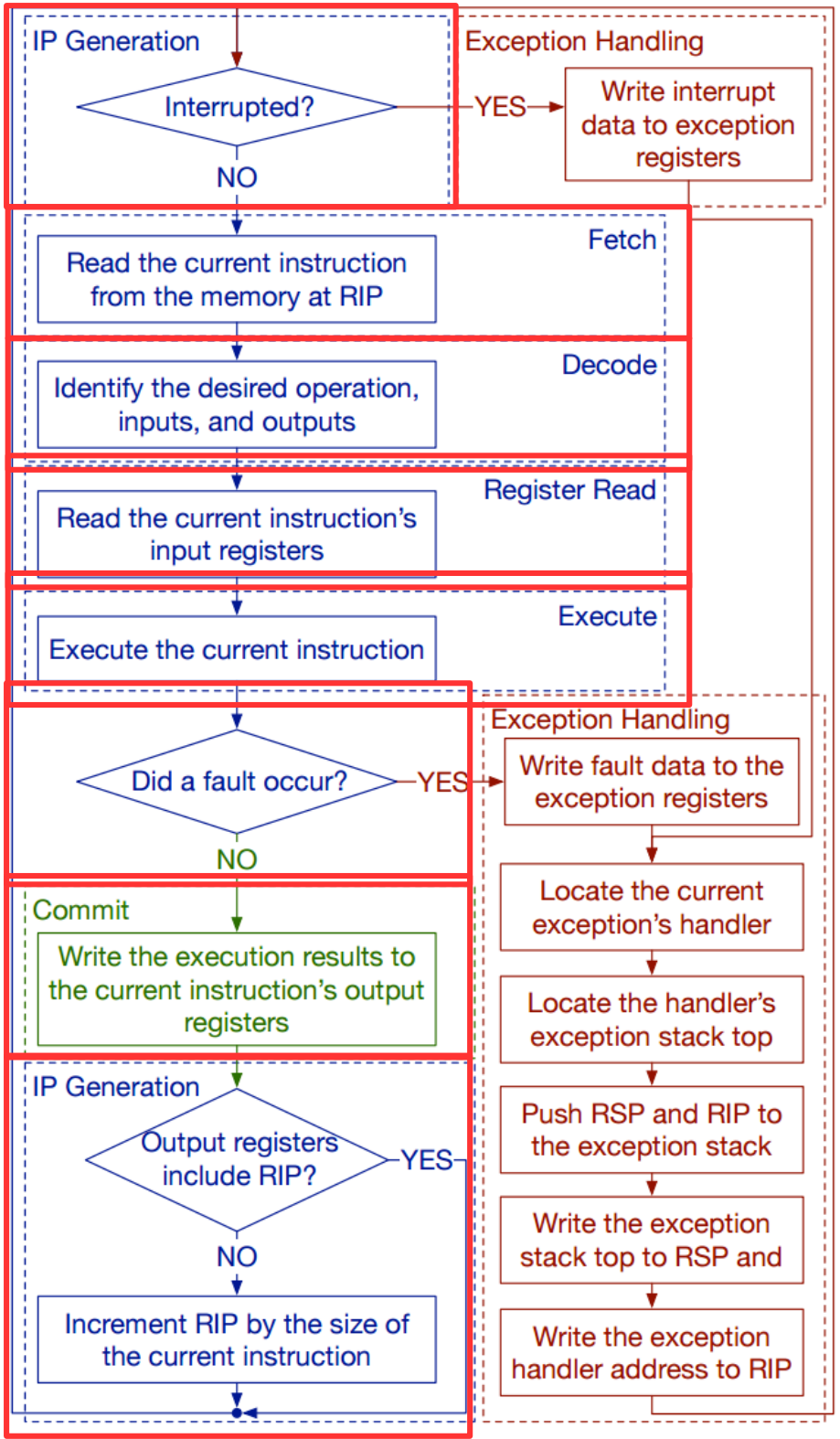
What does CPU do internally?

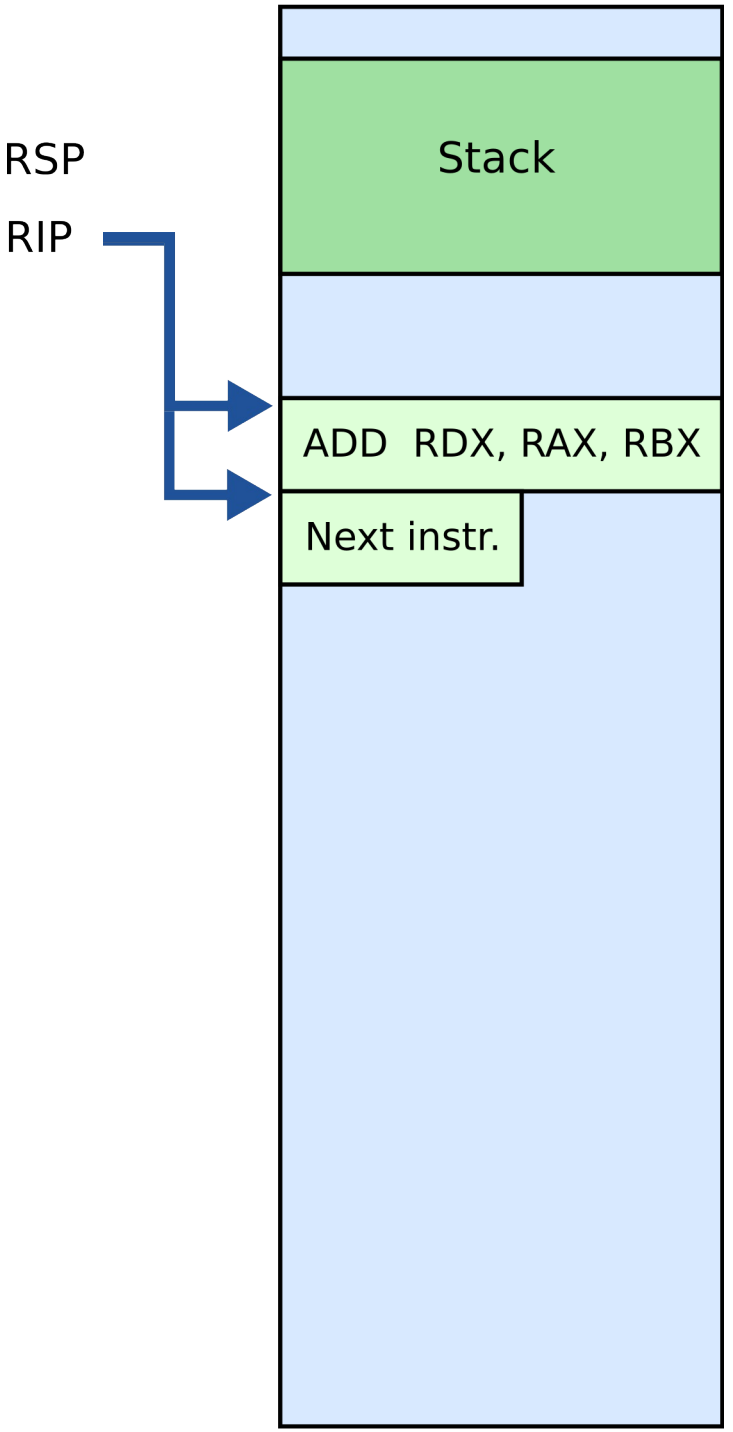


CPU execution loop

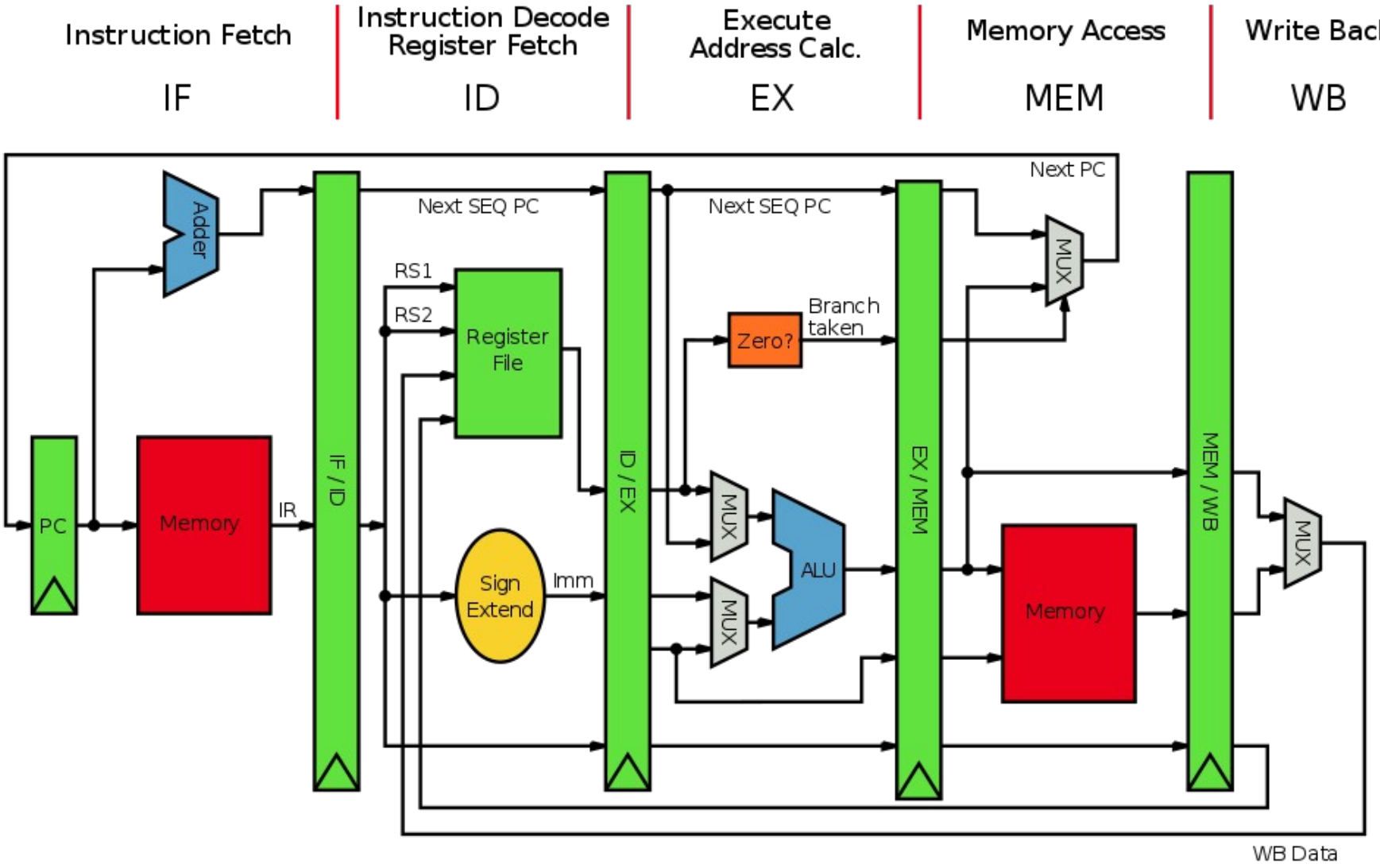
- CPU repeatedly reads instructions from memory
- Executes them
- Example

```
ADD EDX, EAX

// EDX = EAX + EDX
```


IP Generation
Interrupted?
YES
NO
Exception Handling
Write interrupt data to exception registers
Fetch
Read the current instruction from the memory at RIP
Decode
Identify the desired operation, inputs, and outputs
Register Read
Read the current instruction's input registers
Execute
Execute the current instruction
Did a fault occur?
YES
NO
Exception Handling
Write fault data to the exception registers
Locate the current exception's handler
Locate the handler's exception stack top
Push RSP and RIP to the exception stack
Write the exception stack top to RSP and
Write the exception handler address to RIP
Commit
Write the execution results to the current instruction's output registers
IP Generation
Output registers include RIP?
YES
NO
Increment RIP by the size of the current instruction


RSP
RIP
Stack
ADD RDX, RAX, RBX
Next instr.

A simple 5-stage pipeline



Thank you!