High Performance Computing in Julia

from the ground up.

Hardware and Software Basics

1/10

Module Aims

- Provide a **mental model** of how a computer works
- Make the best use of the hardware given (optimisation)
- Introduce parallel computing:
  - Hardware SIMD
  - Multithreading
  - Multiprocessing
  - GPU Programming
- Provide a mix of theory & hands-on practice with Julia
- Give you the skills to learn more

Module Overview

- Hybrid sessions to support those in other Universities
- Only assessed for PhD students (**no** credits for undergrad)
- Module is pass/fail, graded via **unit testing**
- Each week has a programming assignment, delivered via GitHub Classroom.

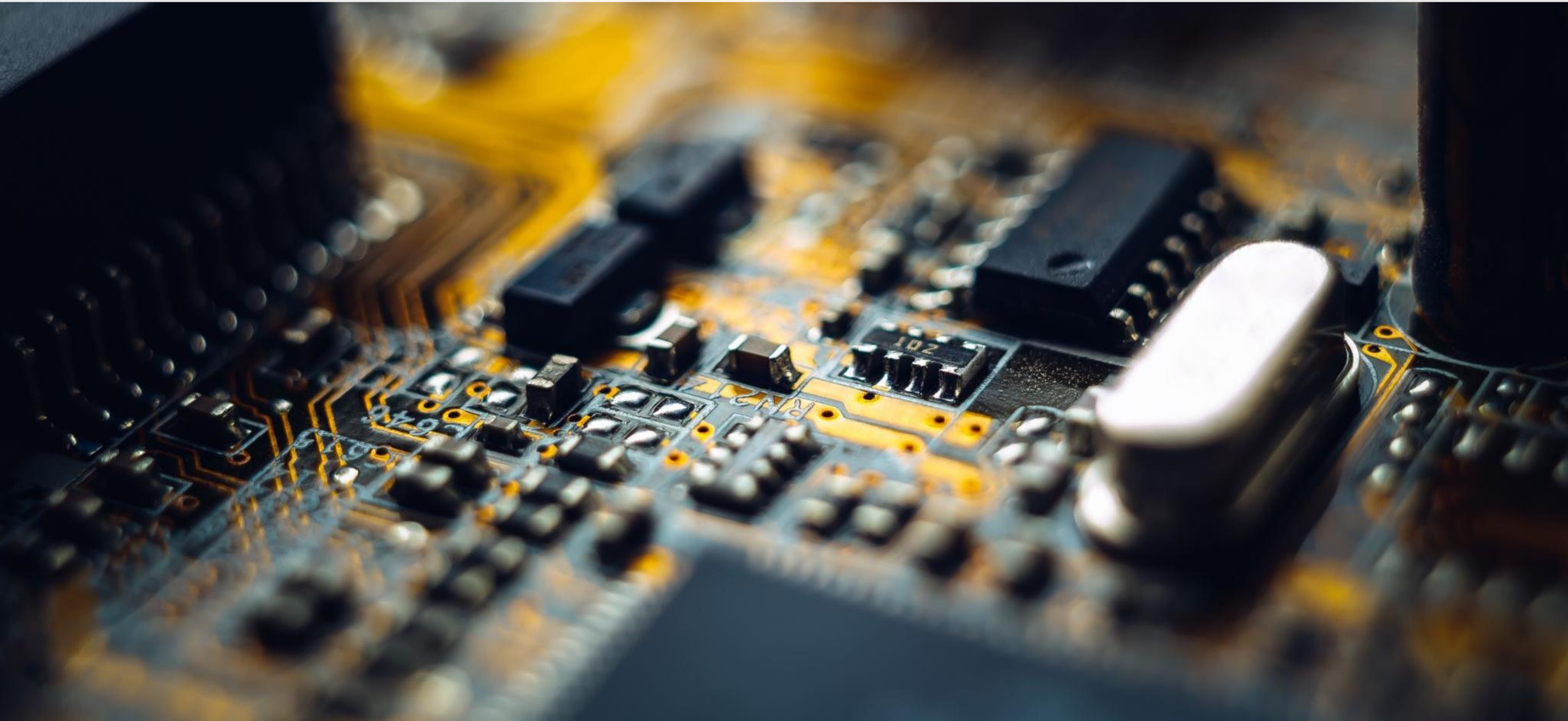
Resources

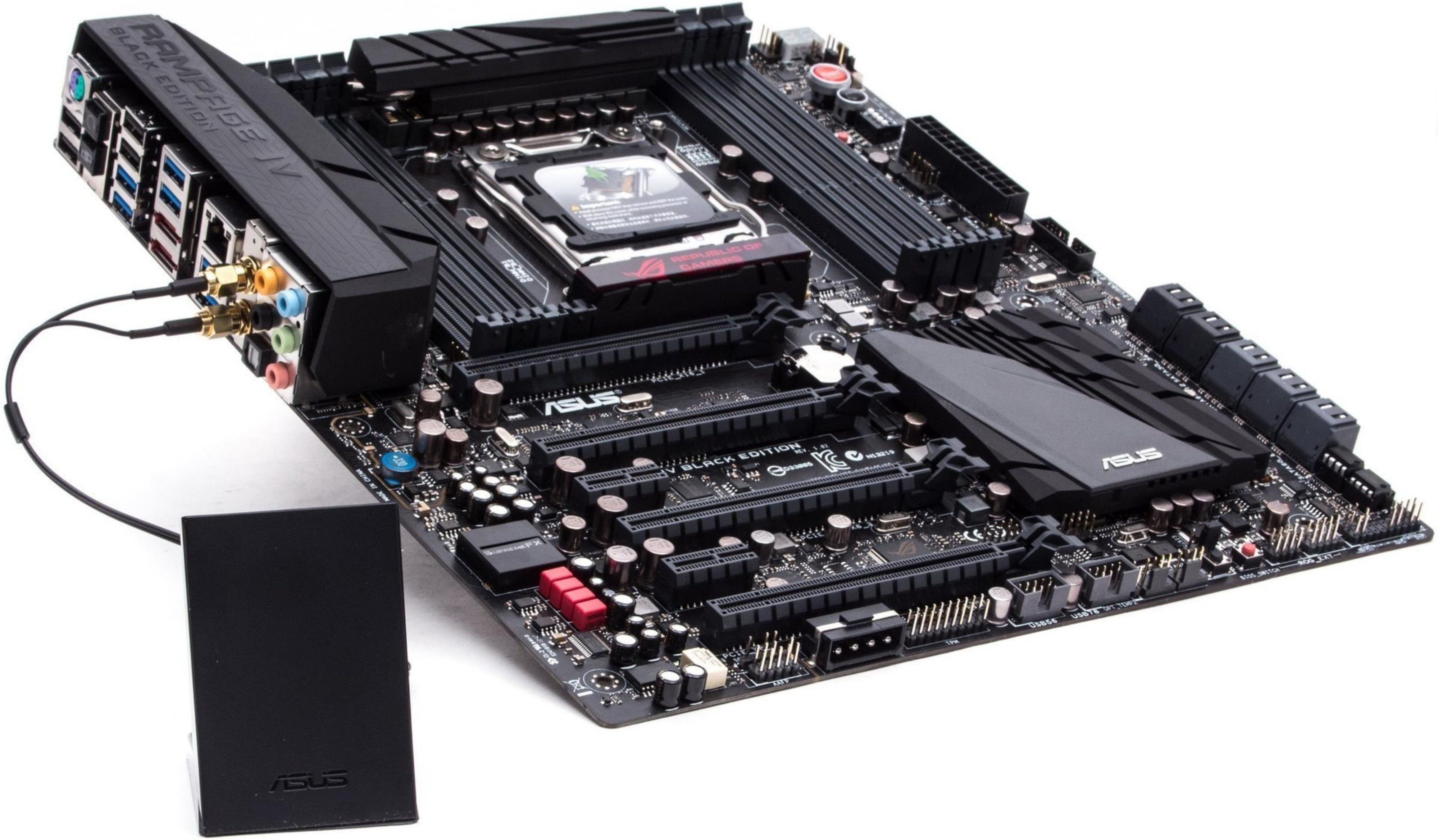
- Website - https://jamiemair.github.io/mpags-high-performance-computing/overview/
- Lecture Notes (PDF) – on the website
    - Contains all lecture material, but still a WIP
    - Lots of **code examples** in **Julia**
    - Additional section on writing professional code (i.e. **version control**, **documentation**, **unit testing** etc)
- GitHub Classroom Assignments
- Help with learning Julia:
    - Look at the lecture notes
    - Read the Julia manual (https://docs.julialang.org/en/v1/manual/getting-started/)
    - Ask questions on the Julia Discourse (https://discourse.julialang.org/)

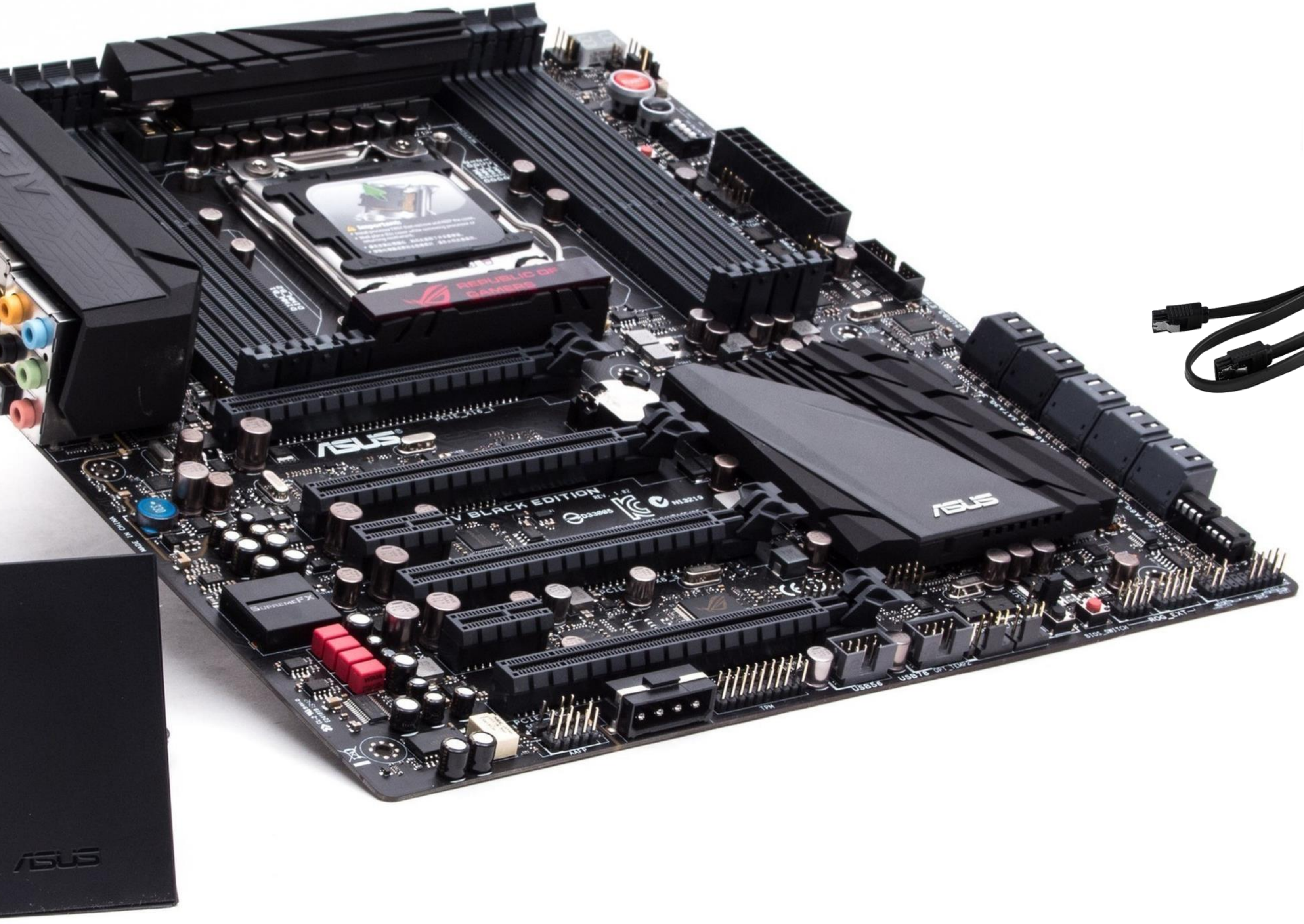
Topics

- **Week 1 - Hardware & Software Basics**
- Week 2 - Optimisation
- Week 3 - Parallel Computing & Multithreading
- Week 4 - Multiprocessing
- Week 5 - GPU Programming with CUDA

Hardware – what's inside a computer?

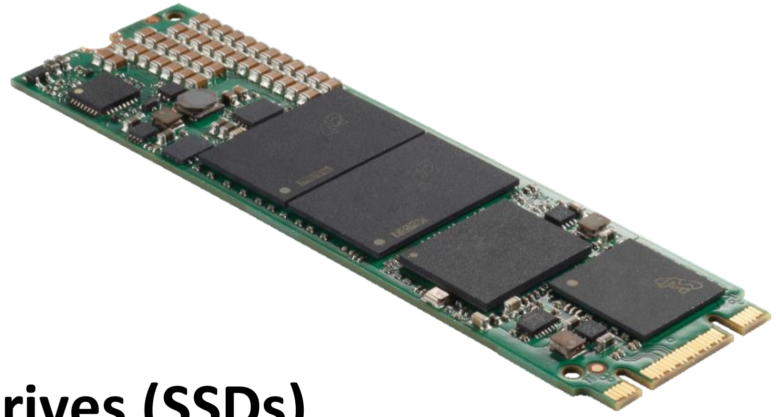



RAMPAGE IV
BLACK EDITION
REPUBLIC OF GAMERS
ASUS
ASUS
ASUS



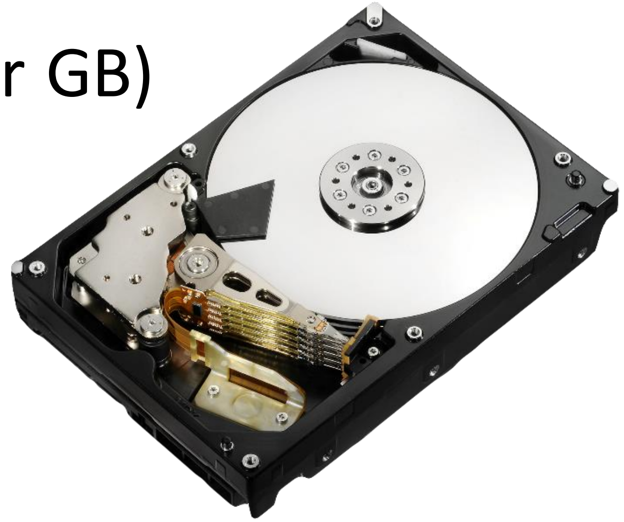
Permanent Storage
e.g. Hard Disk Drive (HDD)

Permanent Storage



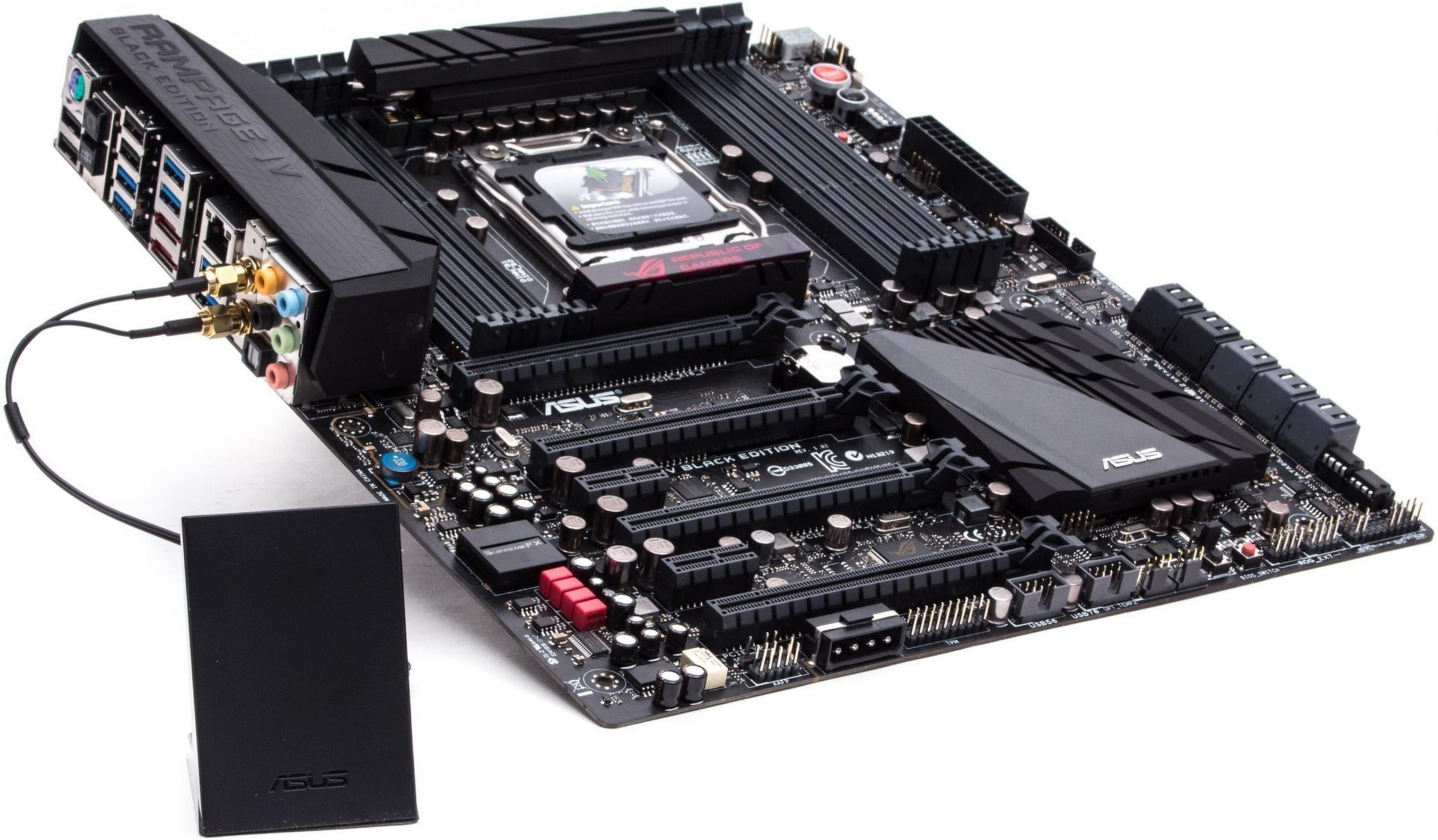
Hard Disk Drives (HDDs)

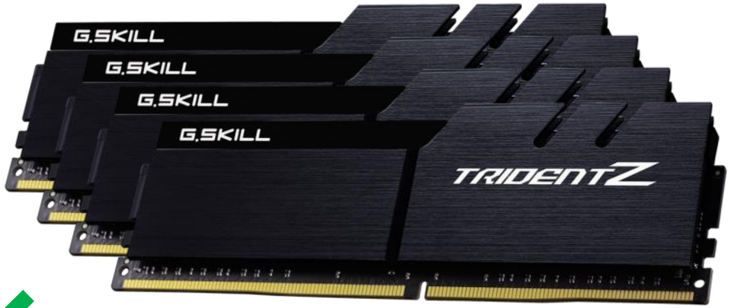
- High Capacity (up to ~30TB)
- Slow read/write (~150MB/s)
- High Latency
- Cheap (~3p per GB)



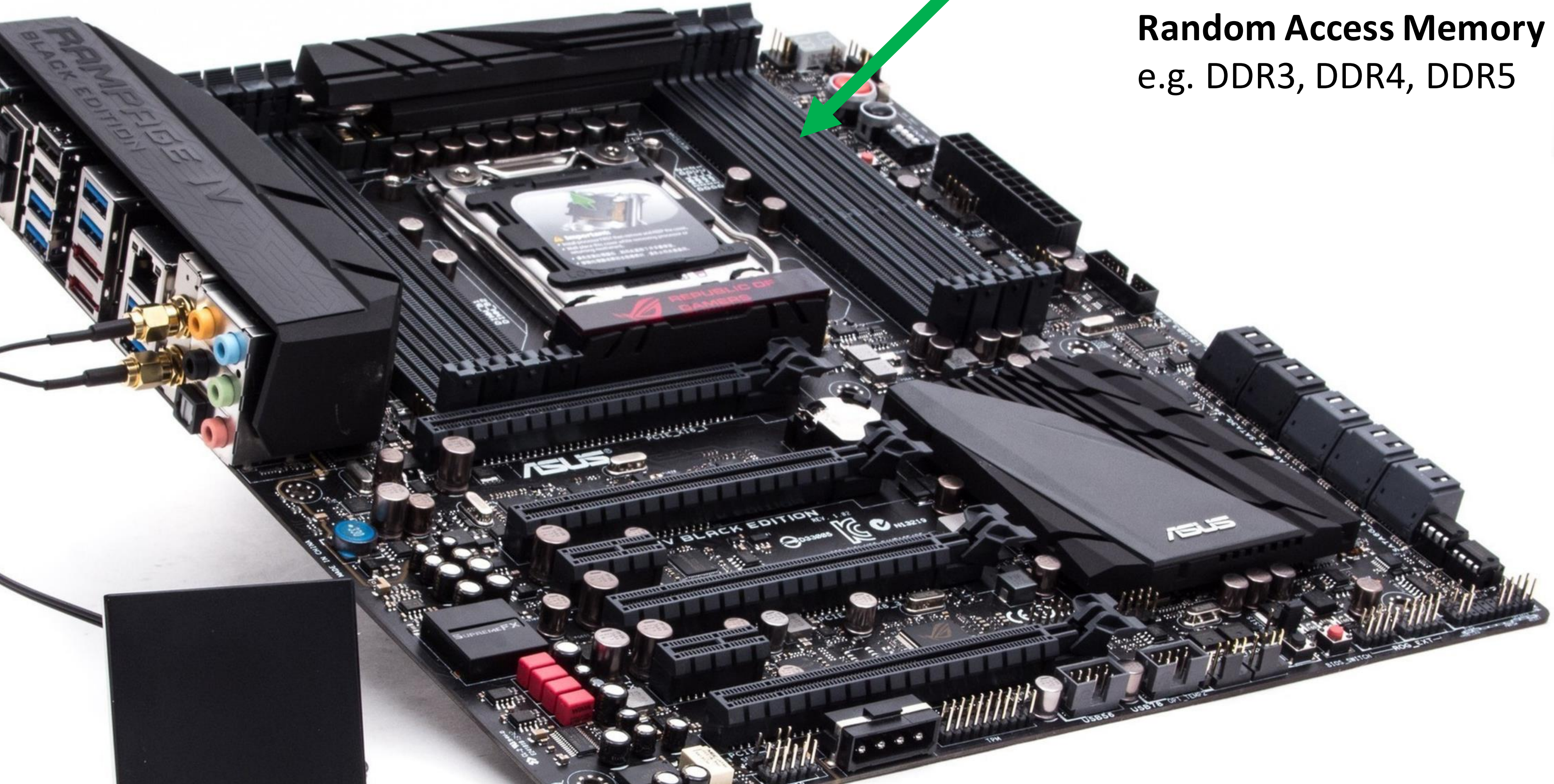
Solid State Drives (SSDs)

- Medium-High Capacity (up to ~8TB)
- Fast read/write (up to ~4-5GB/s)
- Medium Latency
- More expensive (~20p per GB)


RAMPAGE IV
BLACK EDITION
REPUBLIC OF GAMERS
ASUS
ASUS
ASUS

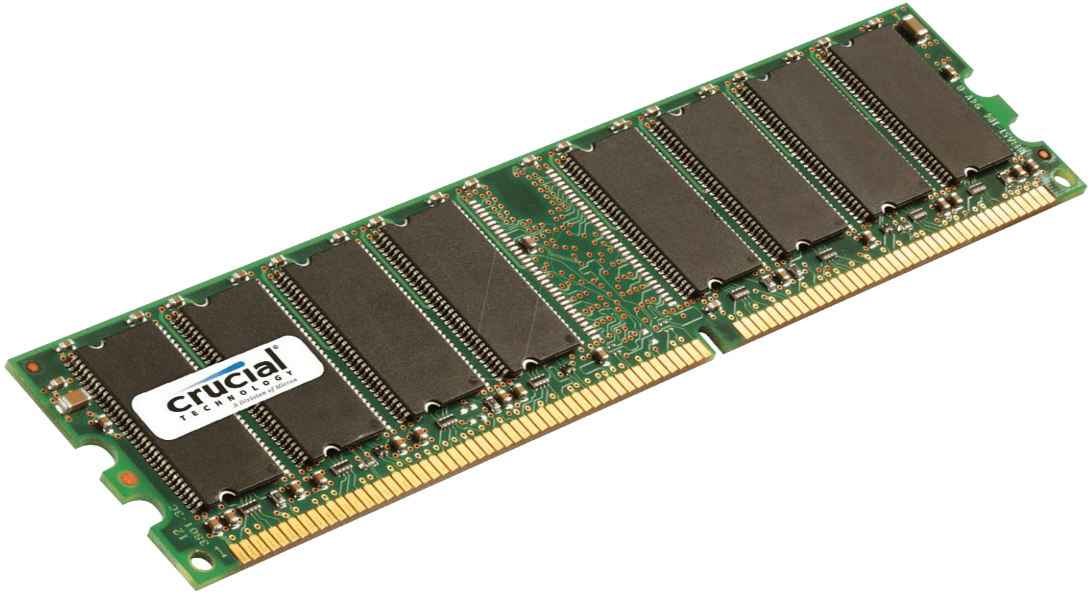


Random Access Memory (RAM)
e.g. DDR3, DDR4, DDR5



Random Access Memory (RAM)

- **Volatile** – loses information when powered off
- Expensive
- High speed (60GB/s)
- Low capacity (~10-100s of GB)
- Low latency (~10-20ns)
- Stores activate programs **and** data

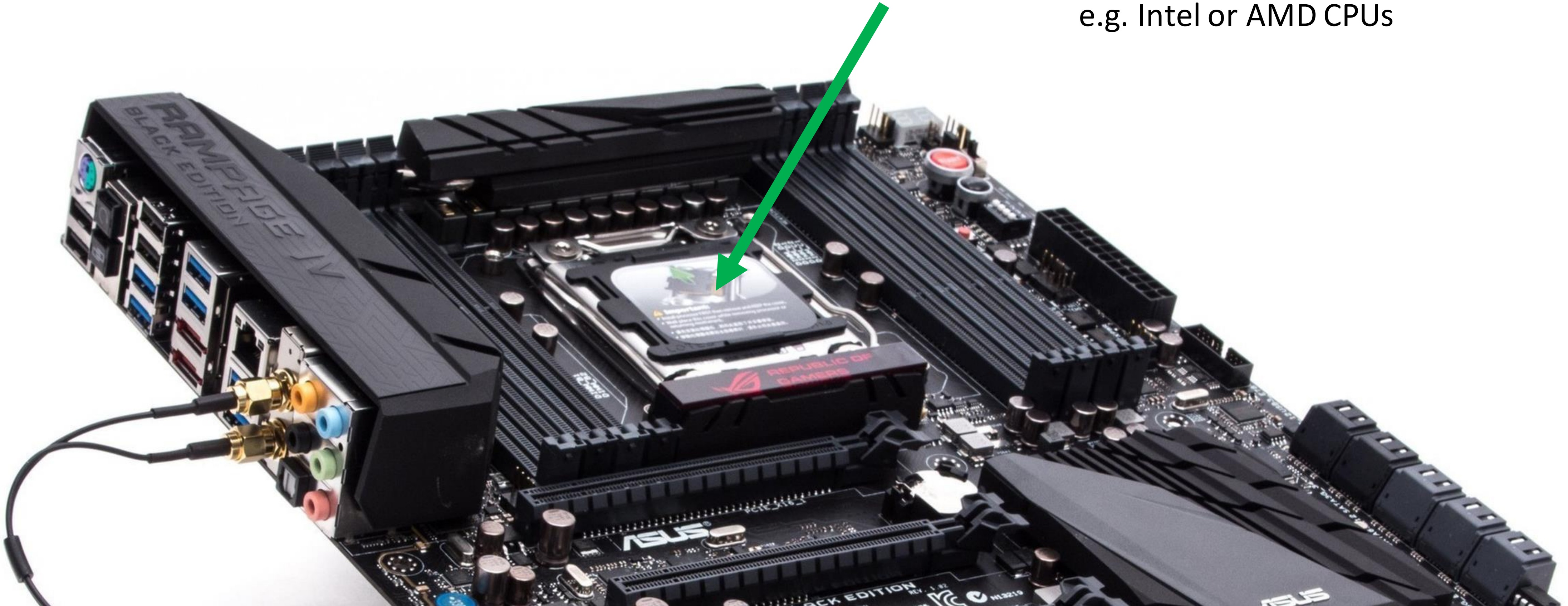


Memory

- "Table" of information stored in **bytes** (8 bits)
- 1 byte is the **smallest** addressable unit of memory
- Each **byte** has a numeric **address**
- **Pointer** refers to data that contains the address of other data

Address	Data
0000	01001001
0001	11011011
0010	00000000
…	…
1110	10000001
1111	00000101

Central Processing Unit (CPU)
e.g. Intel or AMD CPUs



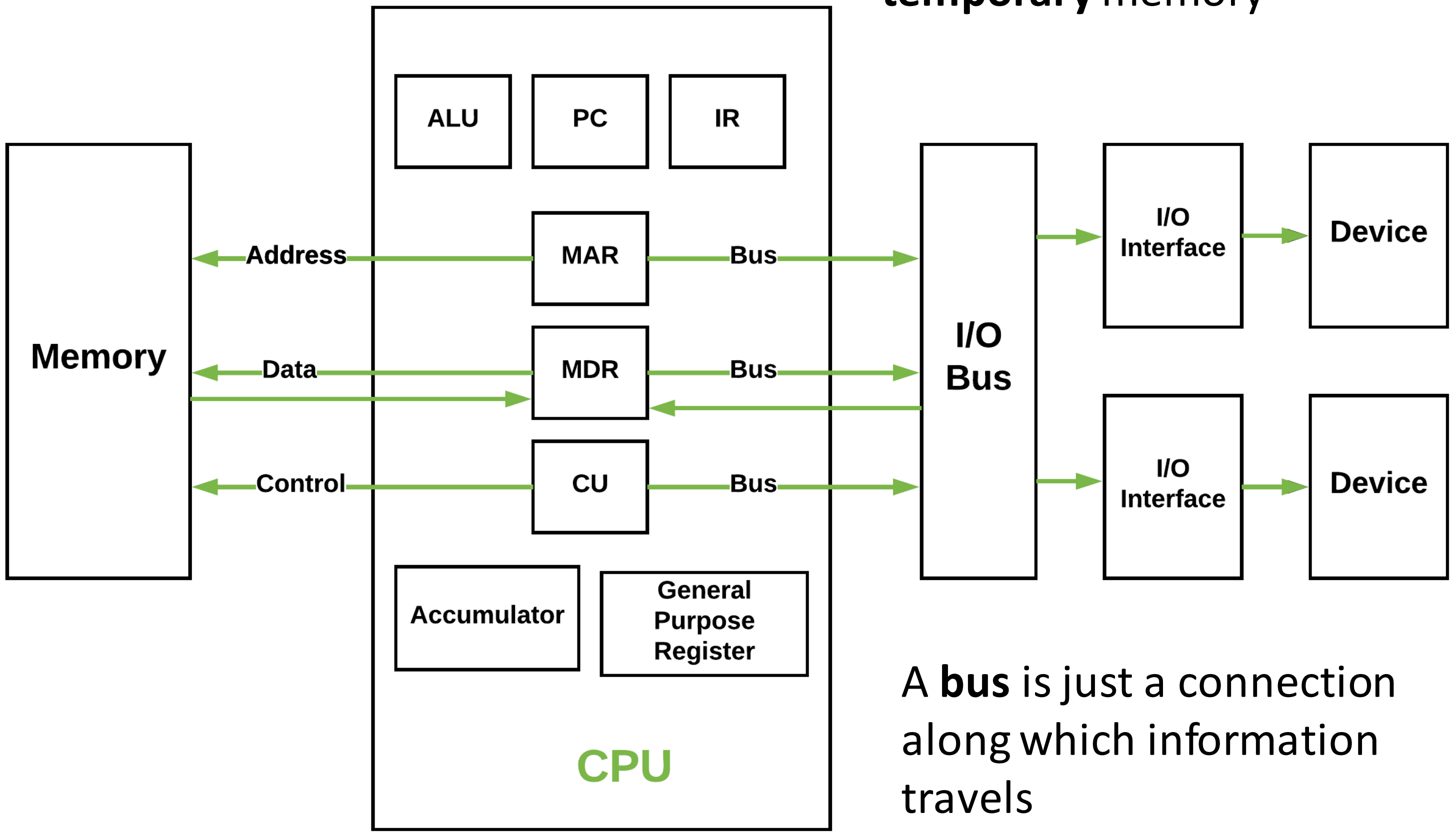
Central Processing Unit (CPU)

- Microprocessors – containing billions of transistors
- Performs logical computations e.g. Arithmetic
- Interfaces with memory/storage and I/O
- Multiple **cores**
- Contains its own memory (L1-L3 cache)

Von Neumann Architecture

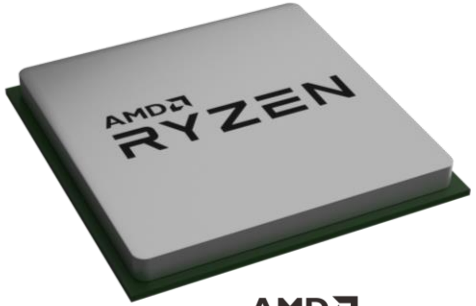
Registers are used for **temporary** memory

Data and **instructions** are stored together in memory



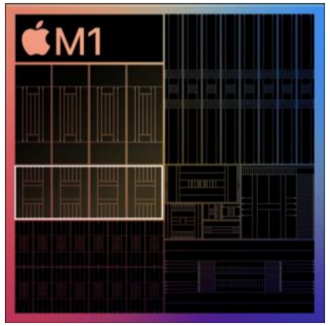
A **bus** is just a connection along which information travels

Modern CPU Architectures



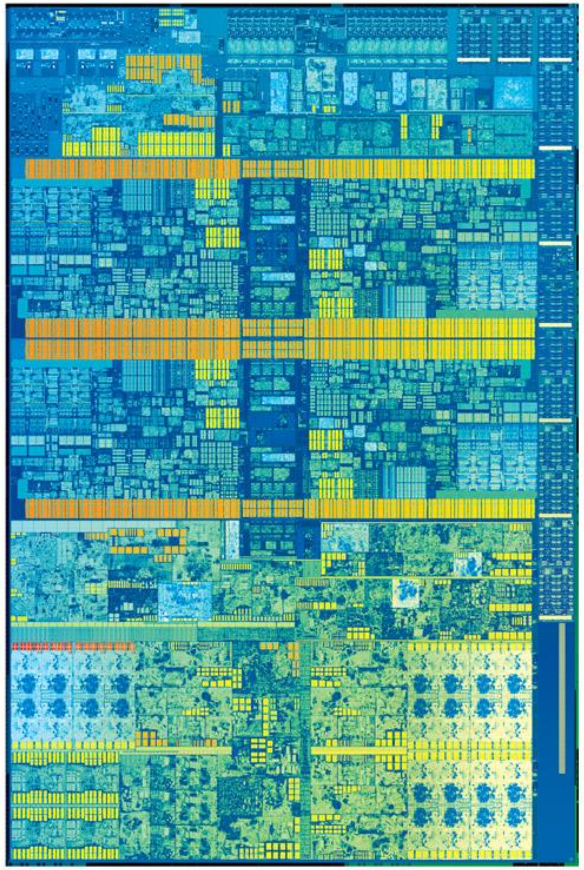
intel

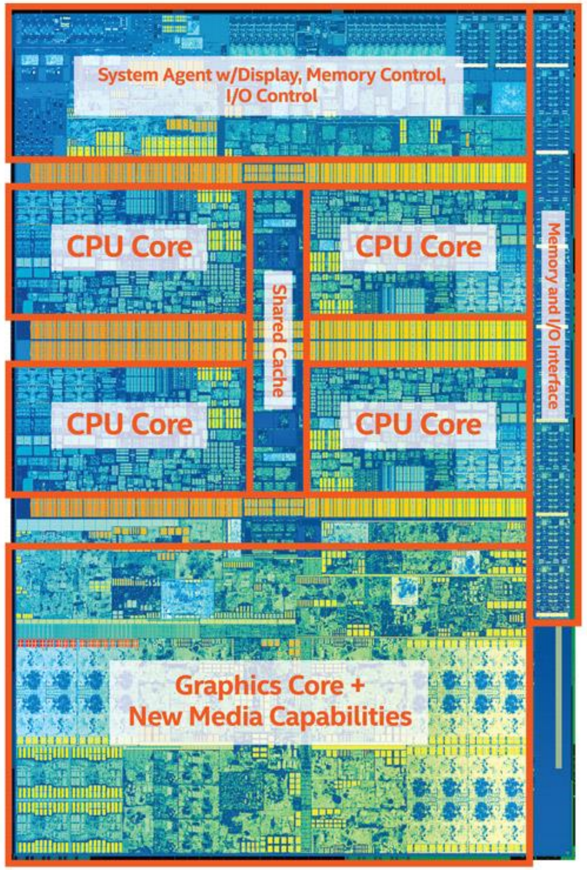
AMD
RYZEN

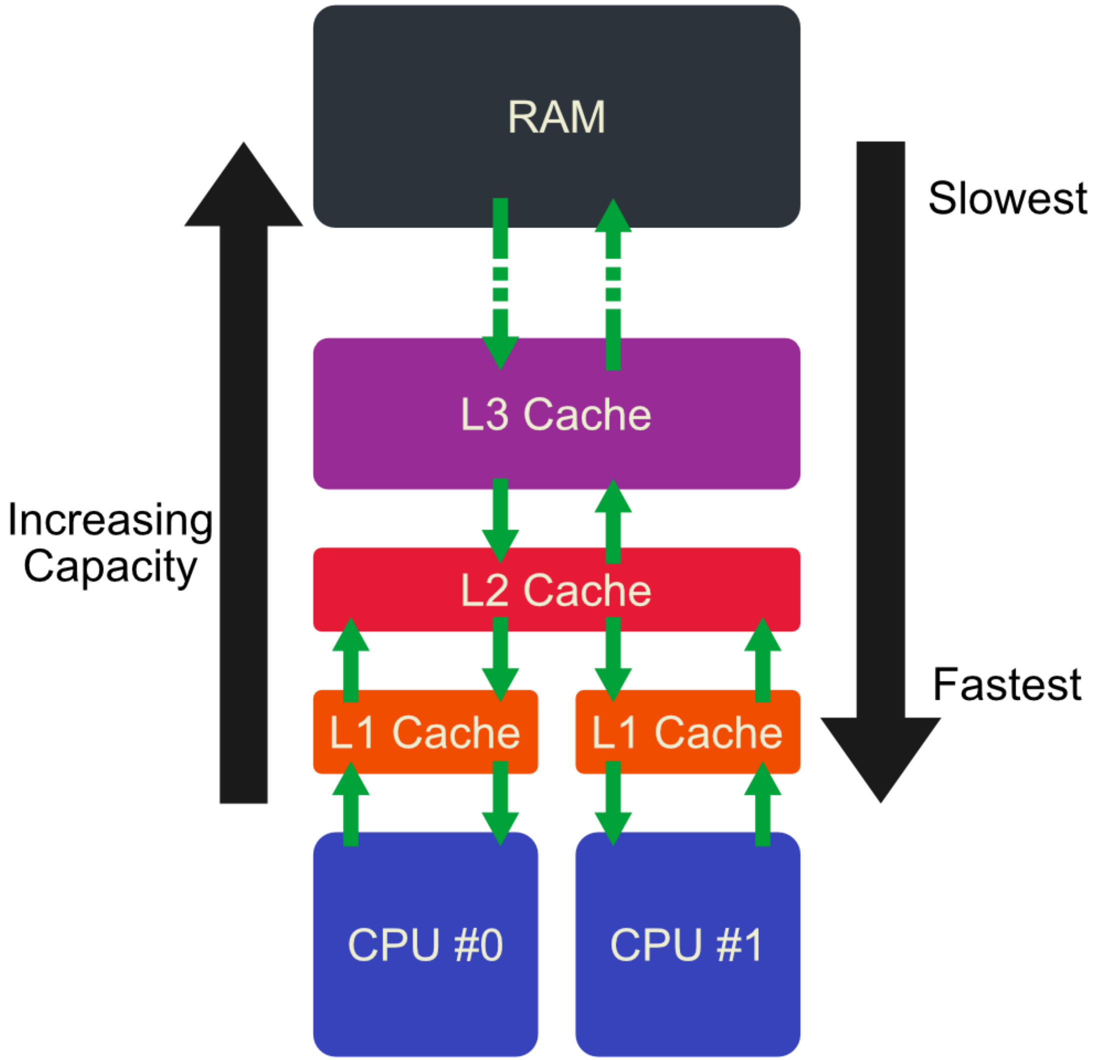


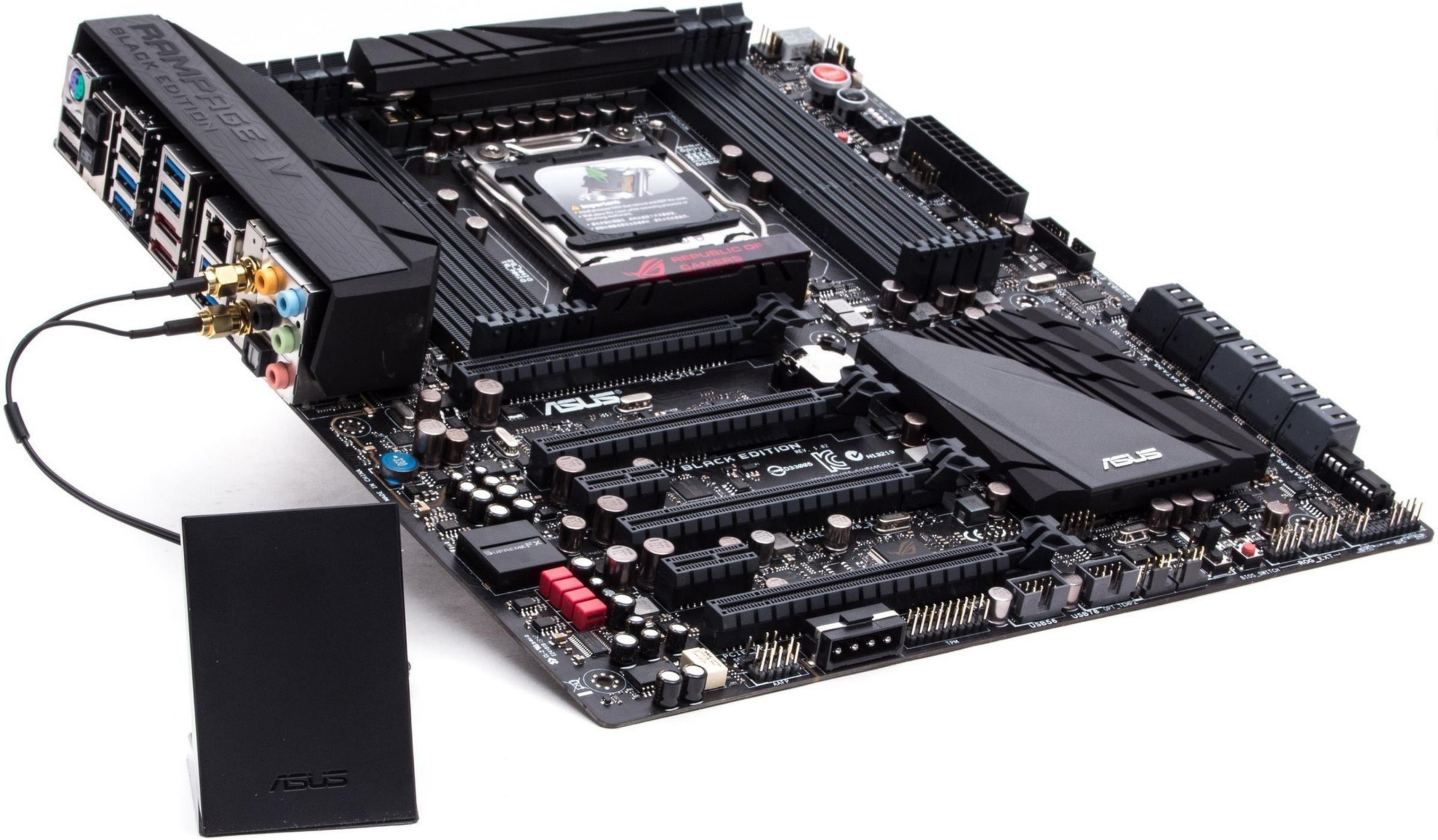
ARM

x86 (32bit) or **x86_64** (64 bit)



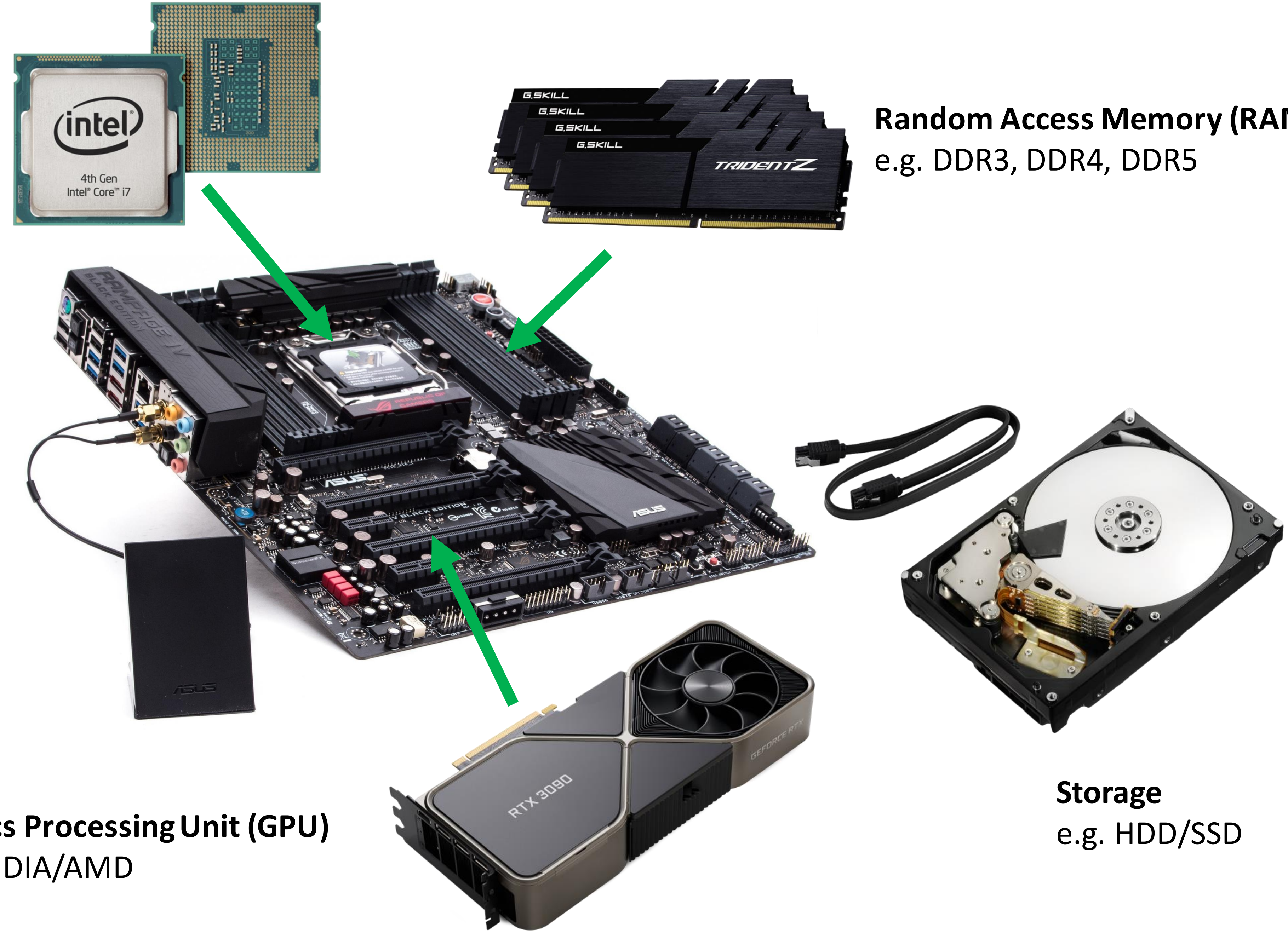

System Agent w/Display, Memory Control,
I/O Control
CPU Core
CPU Core
Shared Cache
Memory and I/O Interface
CPU Core
CPU Core
Graphics Core +
New Media Capabilities


RAM
Slowest
L3 Cache
Increasing
Capacity
L2 Cache
Fastest
L1 Cache
L1 Cache
CPU #0
CPU #1


RAMPAGE IV
BLACK EDITION
REPUBLIC OF GAMERS
ASUS
ASUS
ASUS

Central Processing Unit (CPU)
e.g. Intel or AMD CPUs

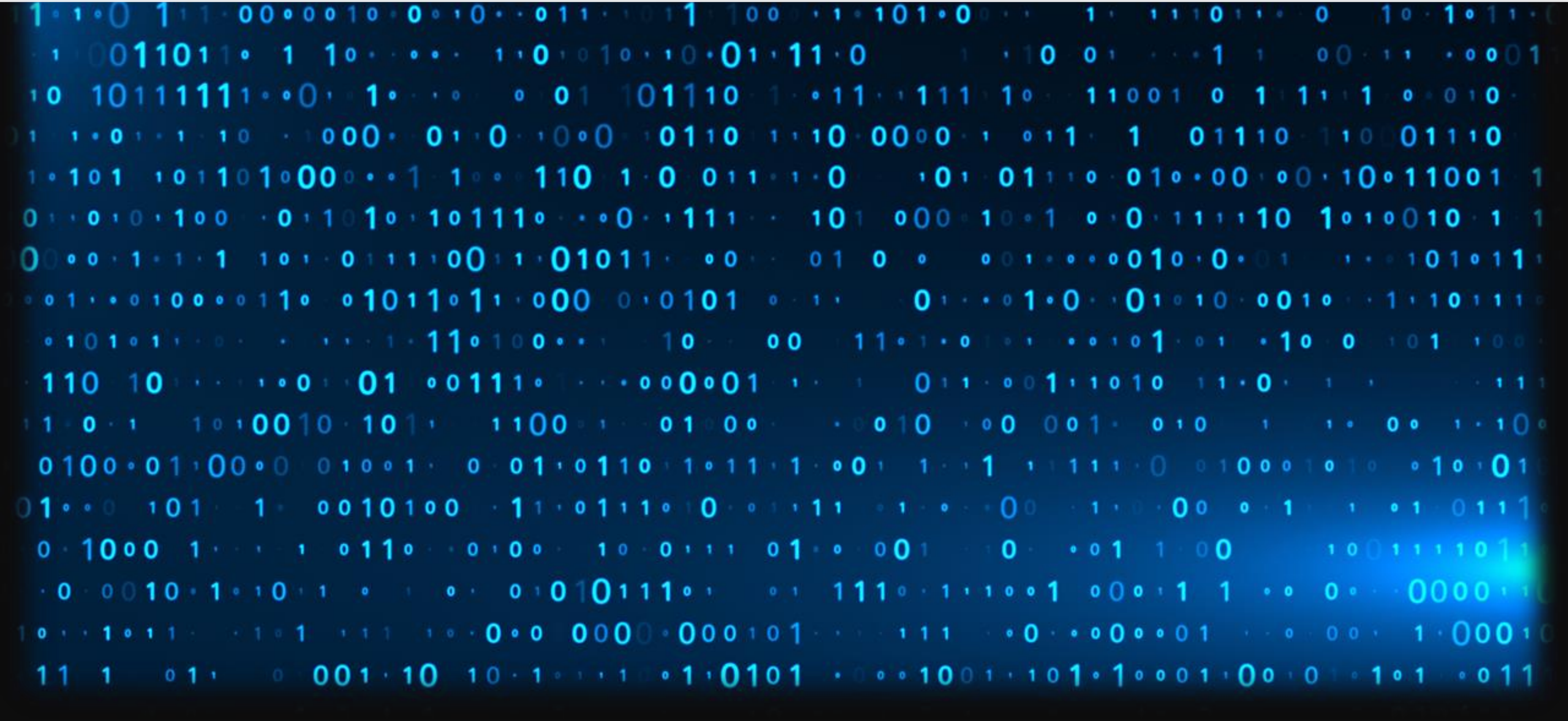
Random Access Memory (RAM)
e.g. DDR3, DDR4, DDR5



Graphics Processing Unit (GPU)
e.g. NVIDIA/AMD

Storage
e.g. HDD/SSD

Software – how does a computer run?

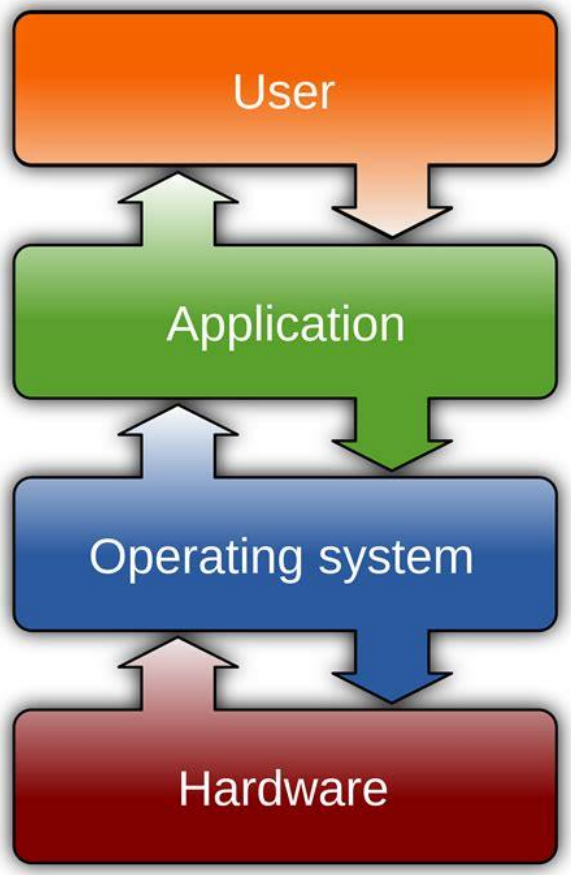


Operating Systems

- Software which manages hardware & software resources
- Provides common utilities to software (e.g. disk & network I/O)
- In control of **loading** software and **scheduling** the CPU to run the software
- Provides **virtual memory** to processes, so each process has its own memory space
- 3 main OS families – **Windows**, **Linux** and **MacOS**

Operating Systems

- Software which manages hardware & software resources
- Provides common utilities to software (e.g. disk & network I/O)
- In control of **loading** software and **scheduling** the CPU to run the software
- Provides **virtual memory** to processes, so each process has its own memory space
- 3 main OS families – **Windows**, **Linux** and **MacOS**



Encoding Data

- Machines only understand binary
- Types are needed to **interpret** data
- Operations change based on the type

- Common types:
  - Floating Point Numbers (Float)
  - Integers (Int or UInt)
  - Boolean
  - Characters (ASCII or Unicode)
  - Have a fixed size due to the **registers** in the CPU

Integer Data Types

- Direct representation in binary for unsigned integers
- Signed integers use the leading bit as a negative

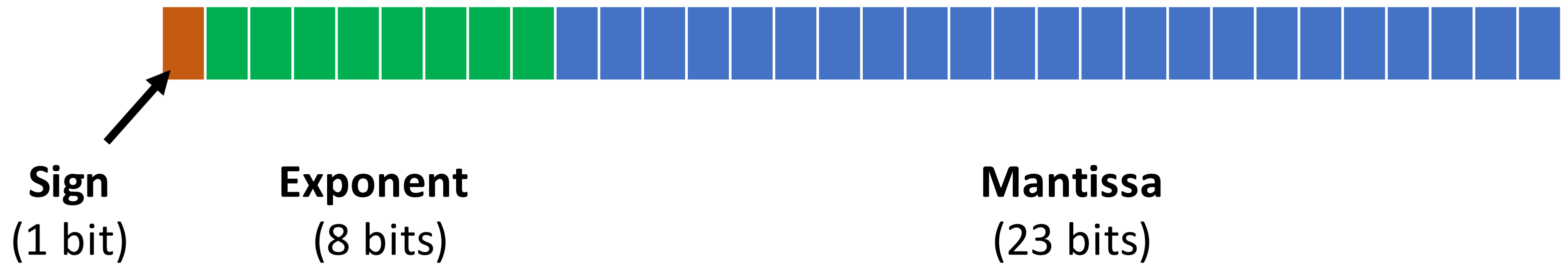
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	1	0	1	1
1	0	0	0	0	0	0	1

- Uses a fixed amount of bits 8/16/32/64 etc

Floating Point Types

- Represent decimal values
- Can be **16**, **32** or **64** bits
  Also known as **half**, **single** or **double** precision
- Partitions the bits into **sign**, **exponent** and **mantissa**
- Similar to scientific notation
- Most use IEEE 754 Standard
- Mantissa holds **significant digits**
- Exponent holds the power of two

$$n = (-1)^s \times (1 + m/2^{23}) \times 2^{e-127}$$



How do we write software?

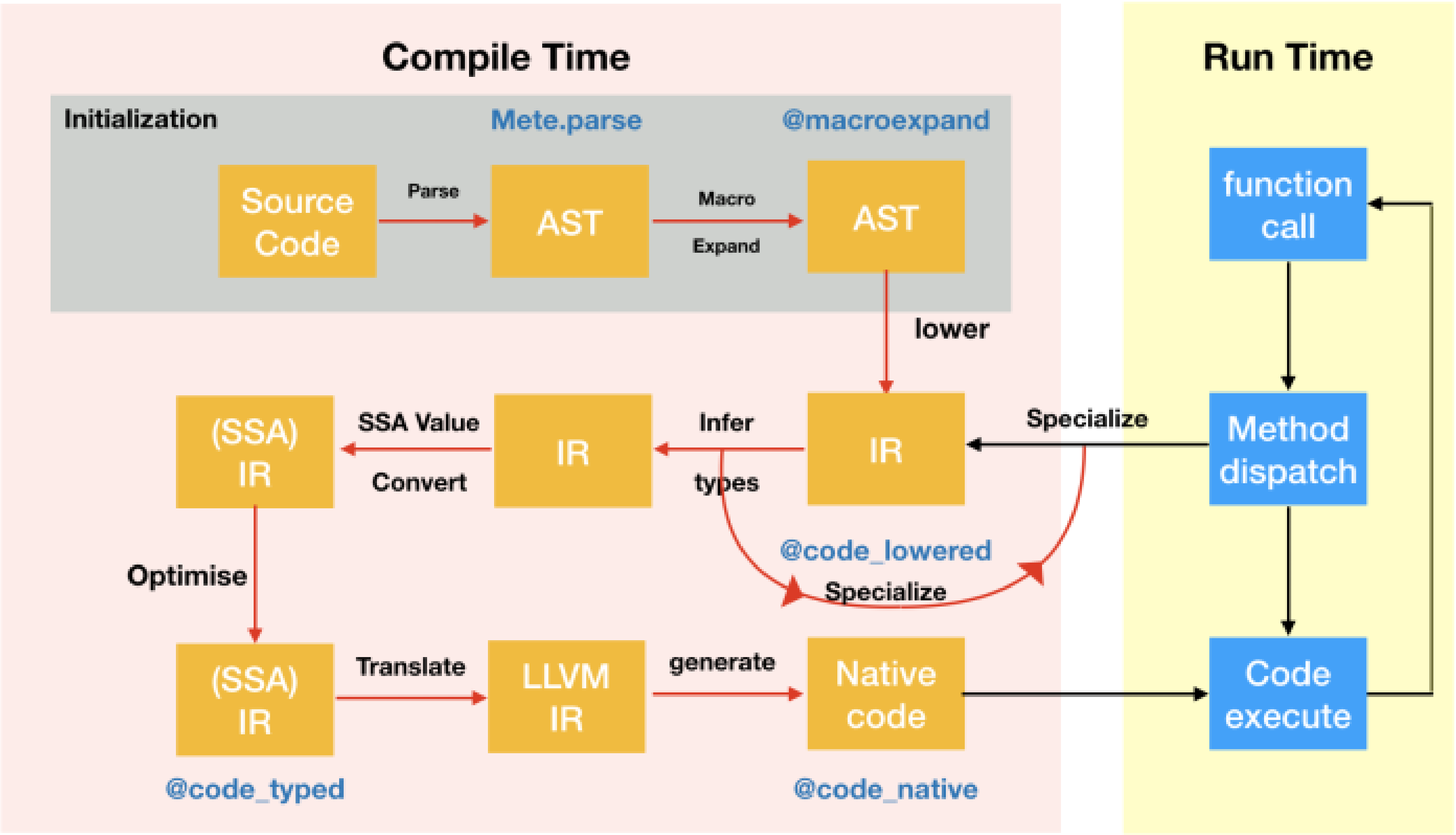
Source Code (Julia)

```julia
f(x) = 5*x*x - 2*x + 1
```

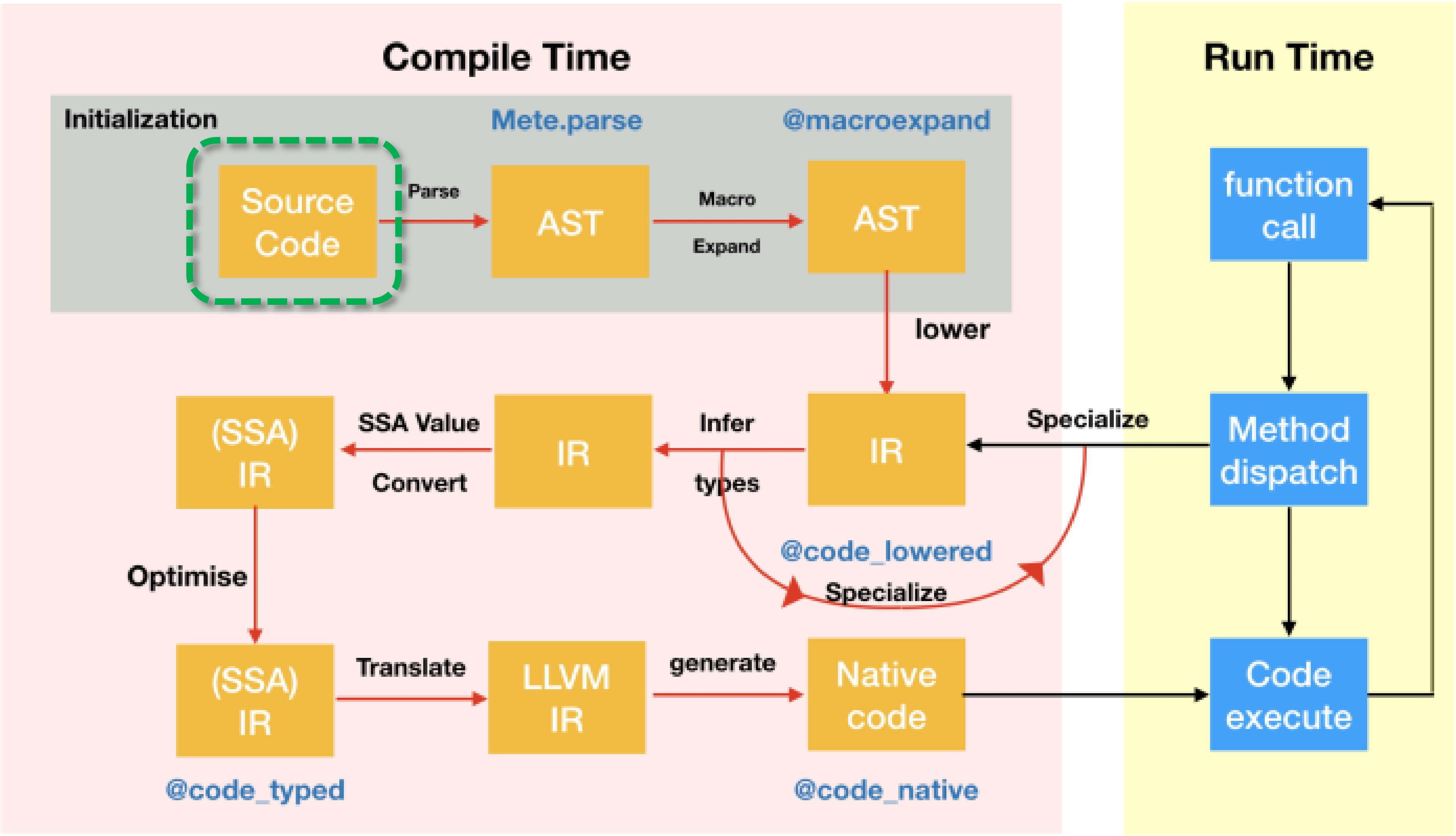
Machine Code (x86 Assembly)

```
pushq                   %rbp
.cfi_def_cfa_offset     16
.cfi_offset             %rbp,              -16
movq                    %rsp,              %rbp
.cfi_def_cfa_register   %rbp
leaq                    -2(%rcx,%rcx,4),   %rax
imulq                   %rcx,              %rax
incq                    %rax
popq                    %rbp
retq
```

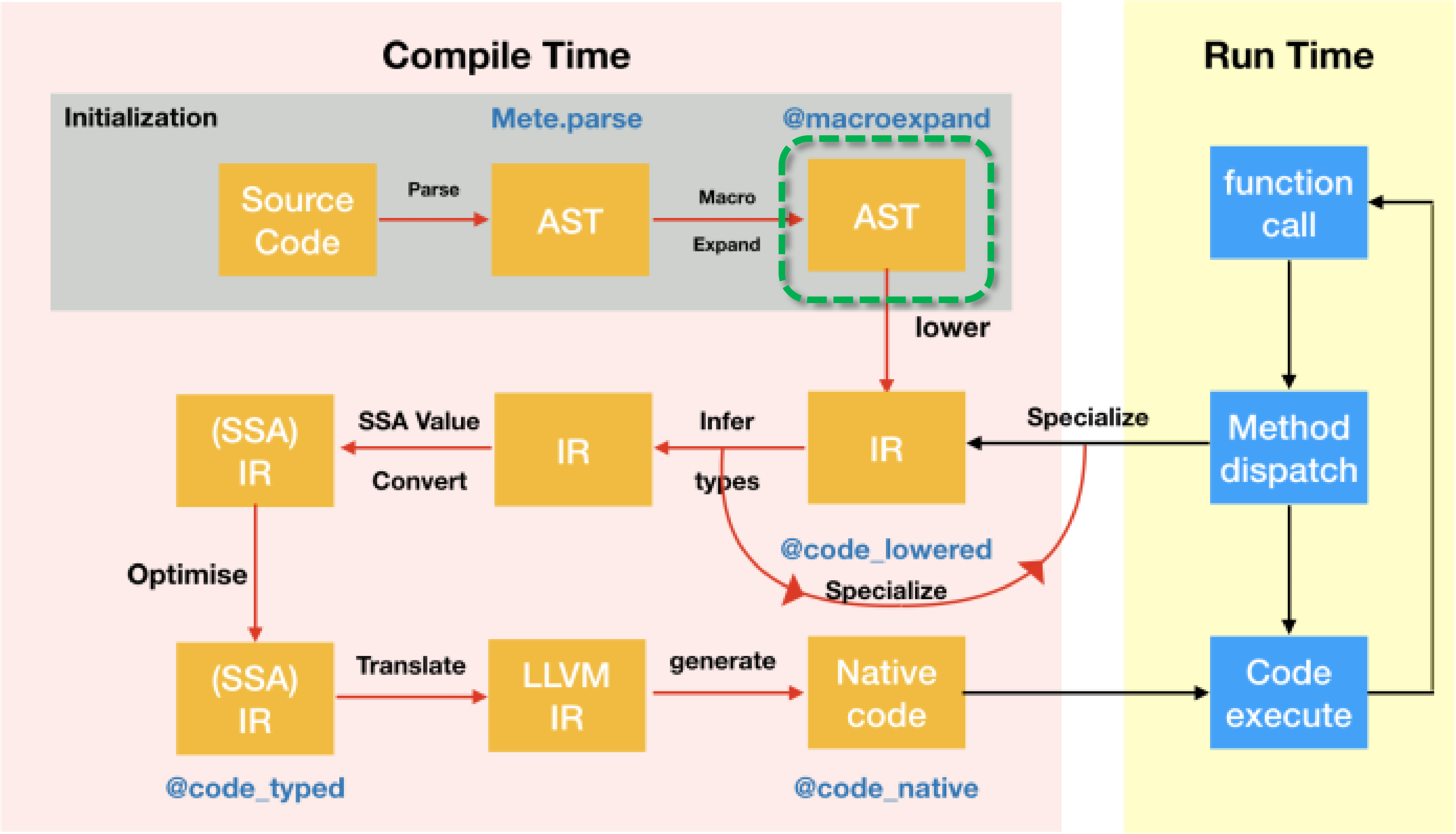
Compilation (in Julia)



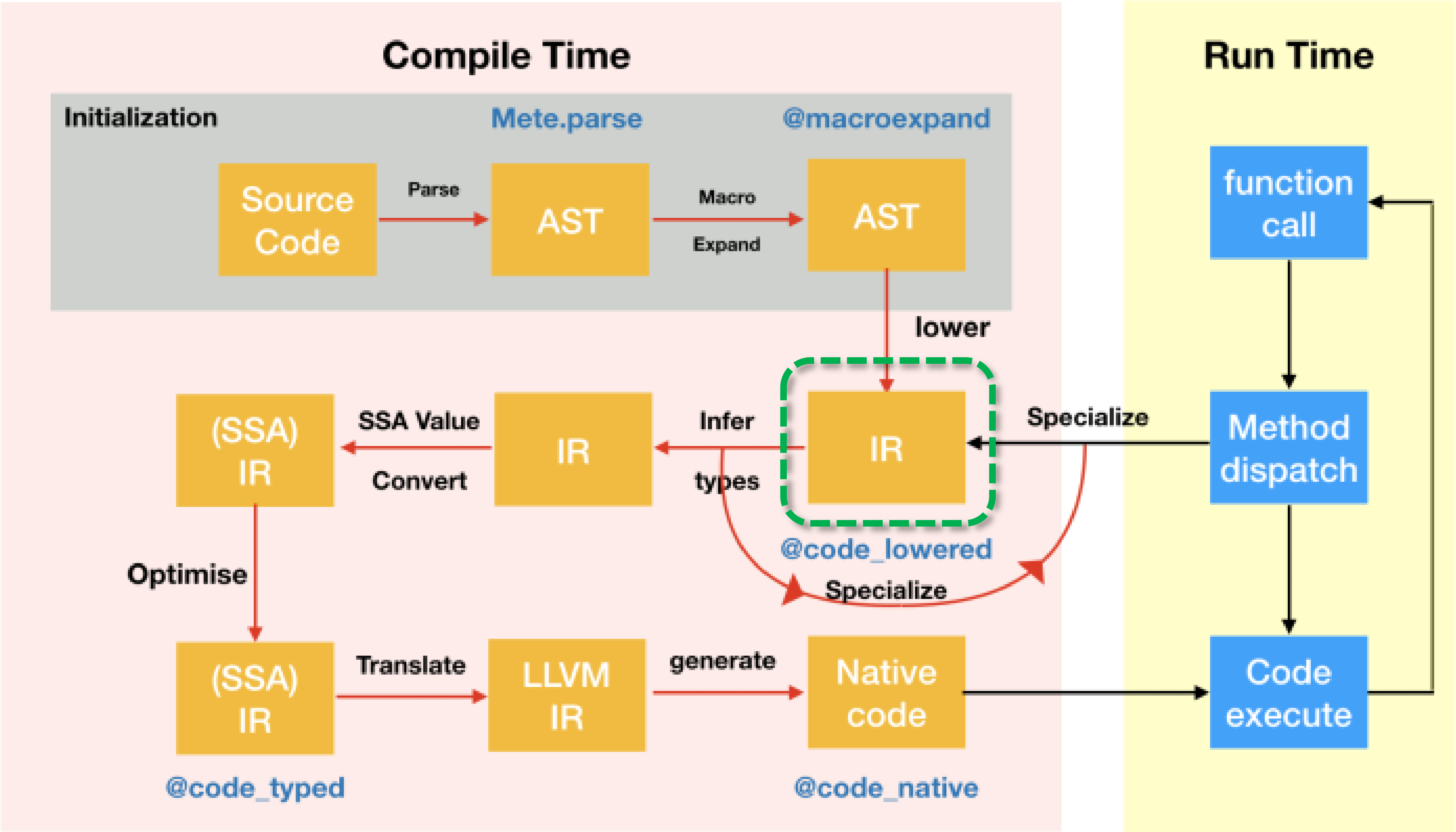
Compilation (in Julia)



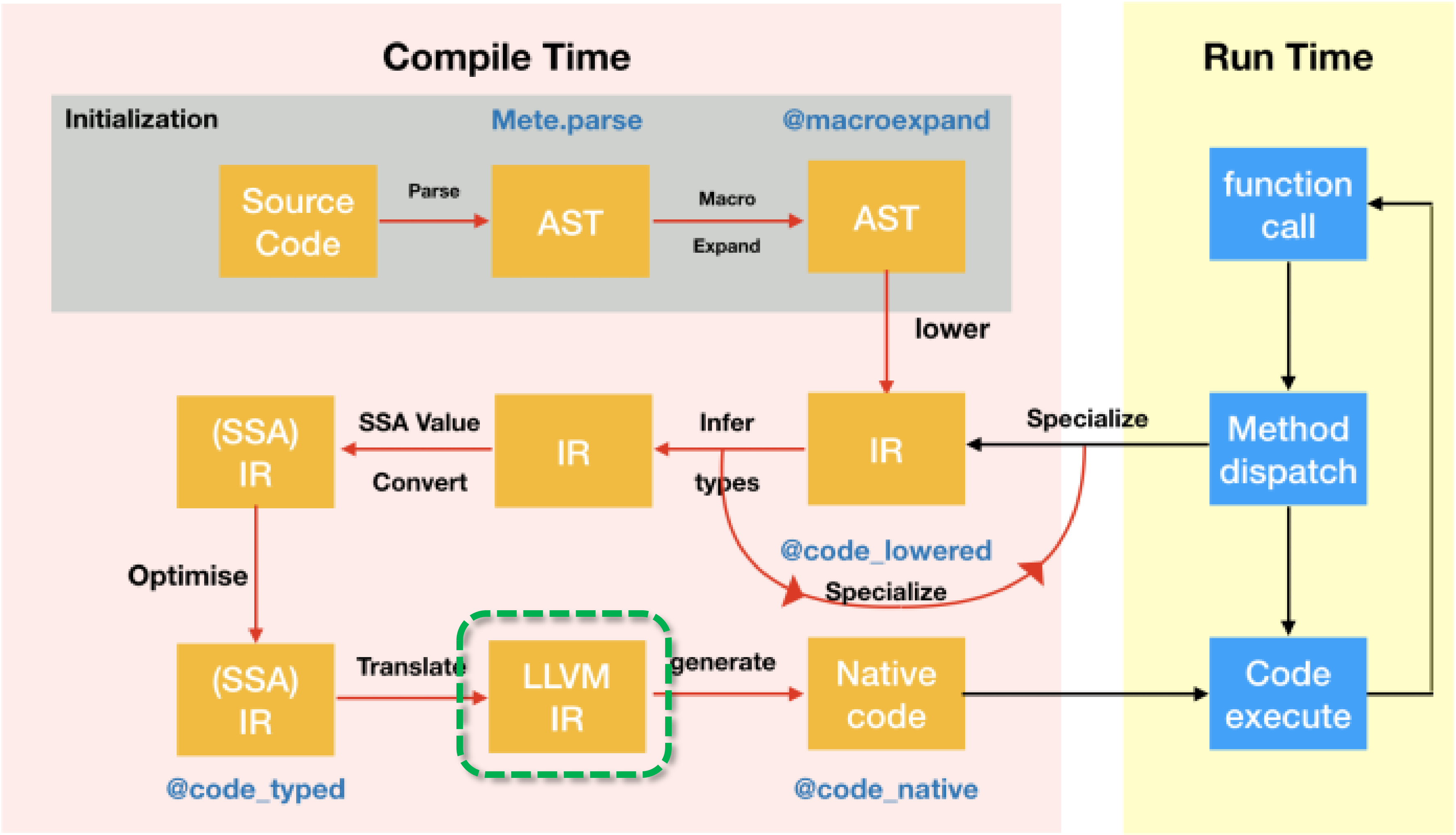
Compilation (in Julia)



Compilation (in Julia)



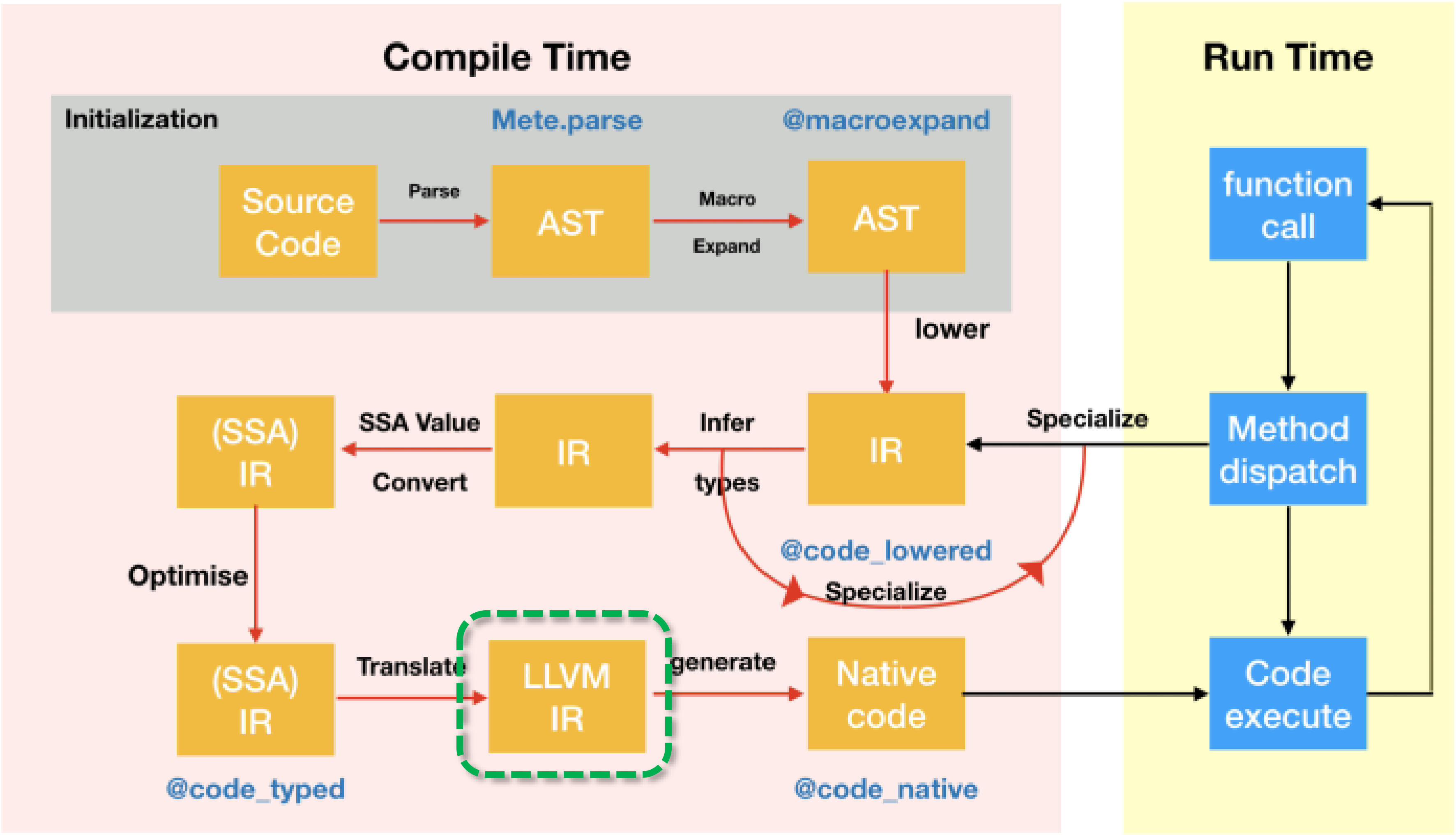
Compilation (in Julia)



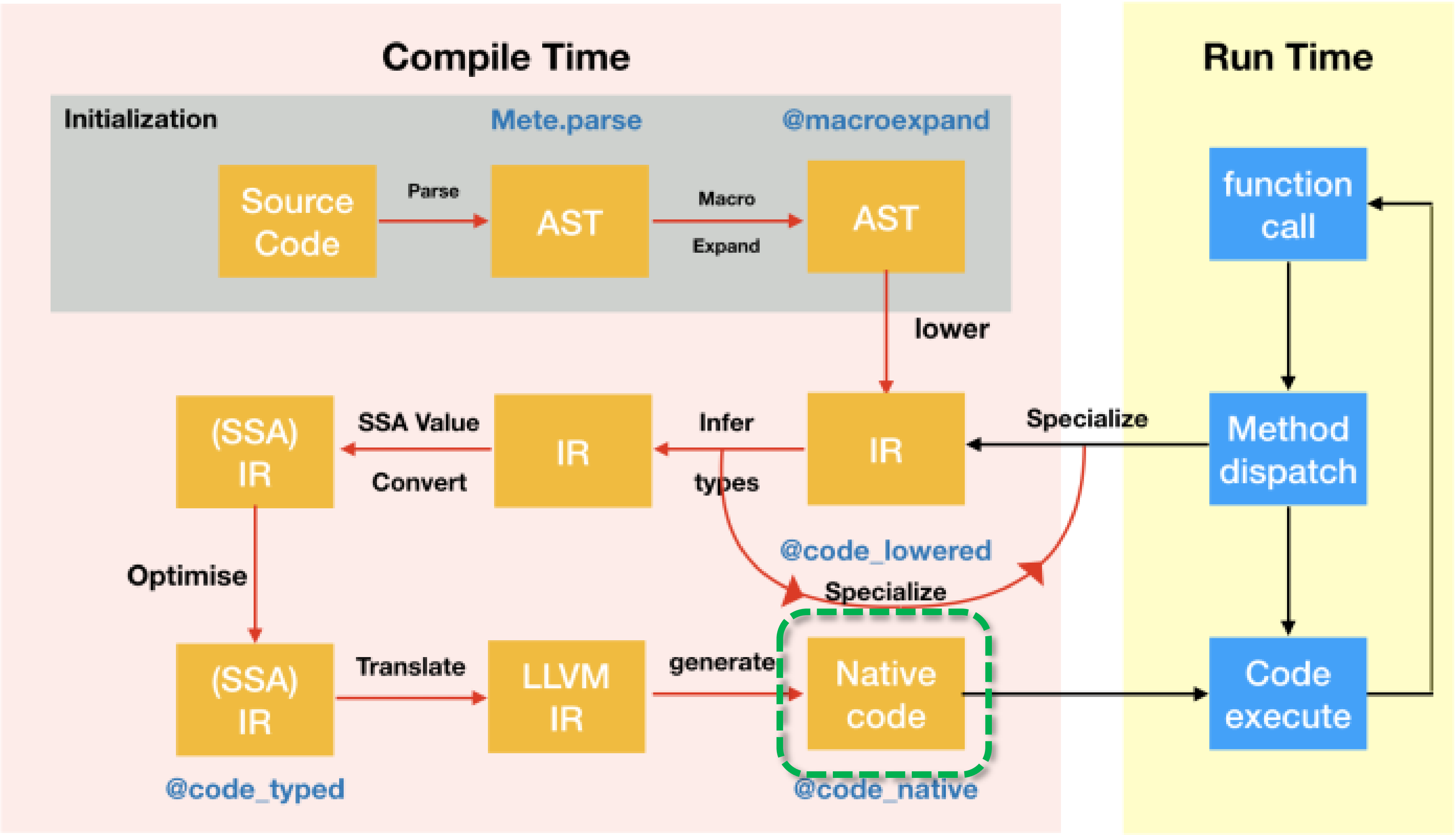
LLVM

- Collection of modular and reusable toolchain technology
- Provides a “front-end” library to generate LLVM IR
- Provides backends for a huge range of architectures (x86, ARM etc)
- Not an acronym – just the name of the project

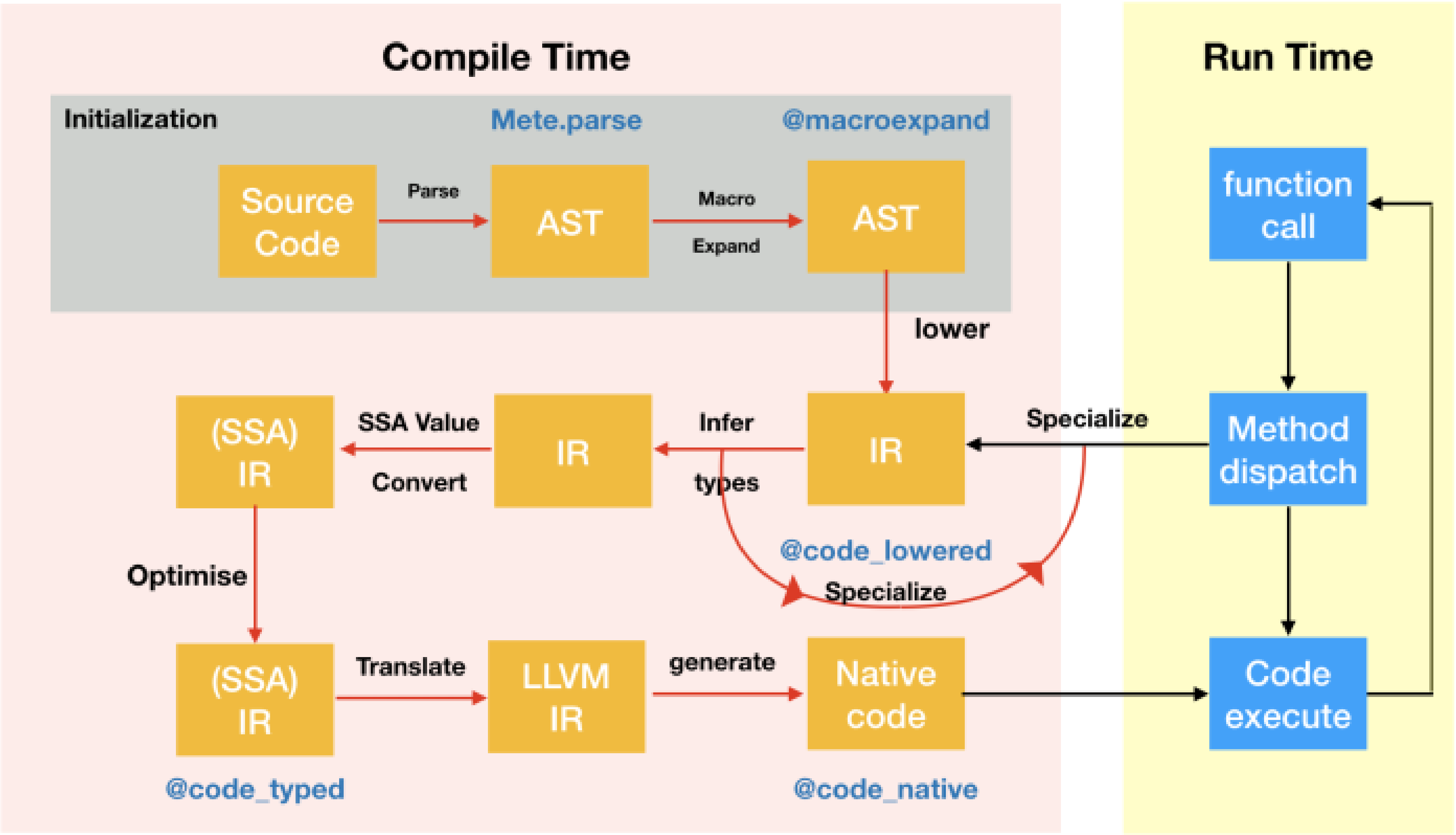
Compilation (in Julia)



Compilation (in Julia)



Compilation (in Julia)



Live Demonstration

Next Session – Thursday 19th Jan

Physics B5

Bring your laptops!

Tasks:

- Create GitHub account
- Accept assignment - https://classroom.github.com/a/3bYk2x83
- Follow instructions (in README or Lecture Notes):
  - Install Git & GitHub Desktop
  - Install Julia
  - Install VS Code & Julia extension