Title: PSI 2015/2016 - Computational/Julia Talk

Date: Apr 12, 2016 11:30 AM

URL: http://pirsa.org/16040101

Abstract:

Computational Physics

Erik Schnetter

Perimeter Institute, April 12, 2016

Physics Computing

Computational Physics

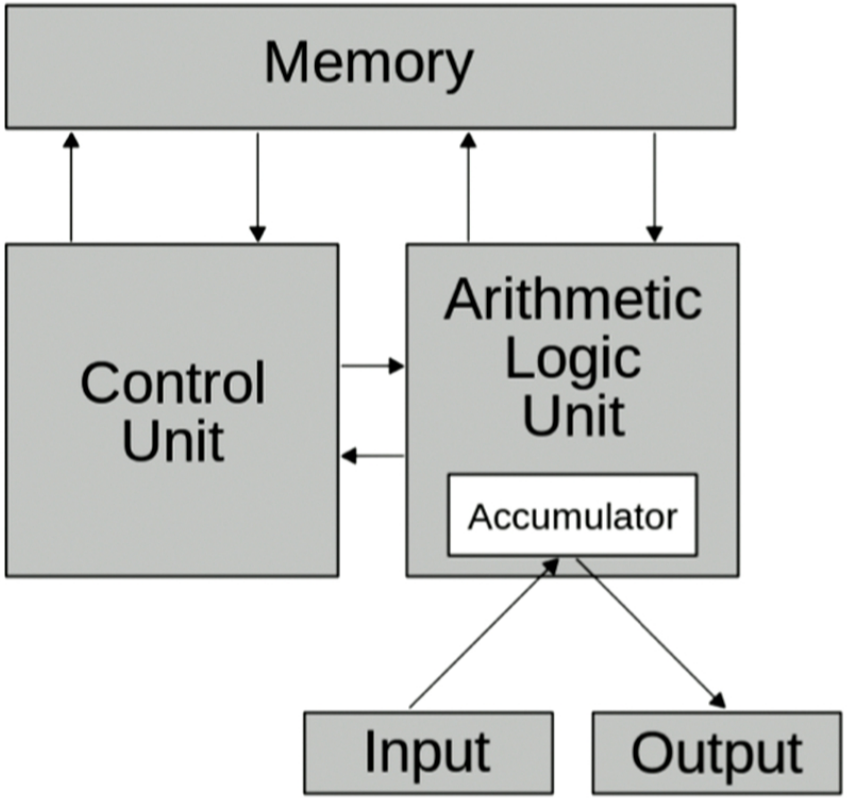
- High-level introduction to computational methods

1. Computing architecture ("what is a computer and how does it work")
2. Numerical analysis ("how do I solve a PDE on a computer")

Algorithm Development

- Naïve idea:
  - Von Neumann (and others) developed a theoretical model of computing
  - Take an algorithm ("constructive proof"), map it to this model, implement it in a computer language, done
- In practice:
  - What is possible (and what is efficient) is not determined by an abstract model, but by current-day hardware technology
  - "Good" algorithms today look very different than 20 years ago
  - (... not because people today are more clever!)

Von Neumann Architecture



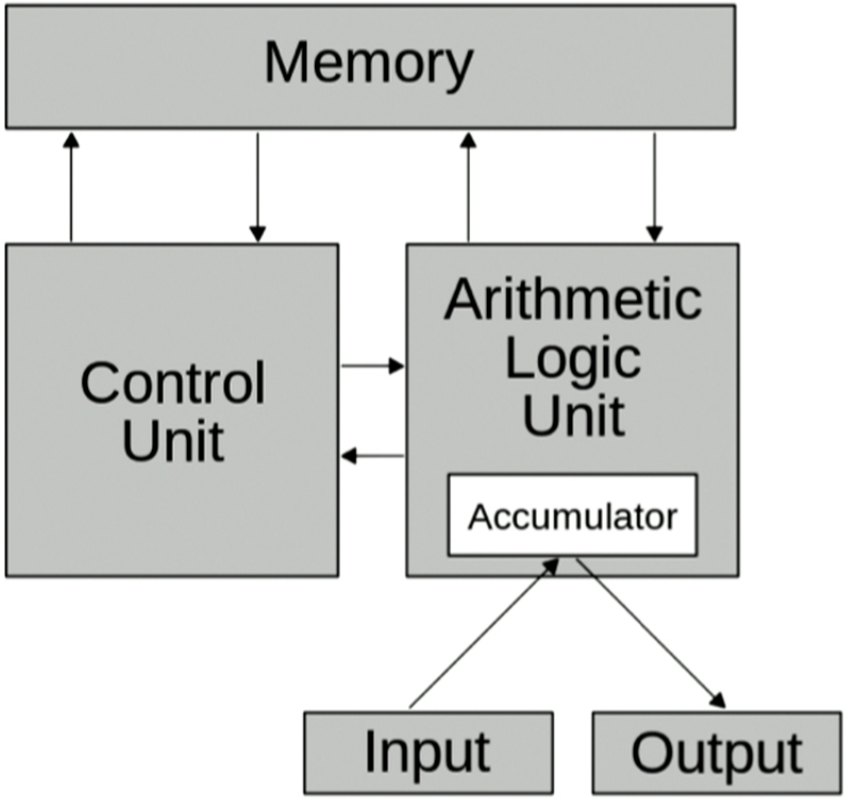
Sequential Execution:

x := a + b

1. read instruction
2. read a
3. read b
4. add
5. write x

[Wikipedia]

Von Neumann Architecture



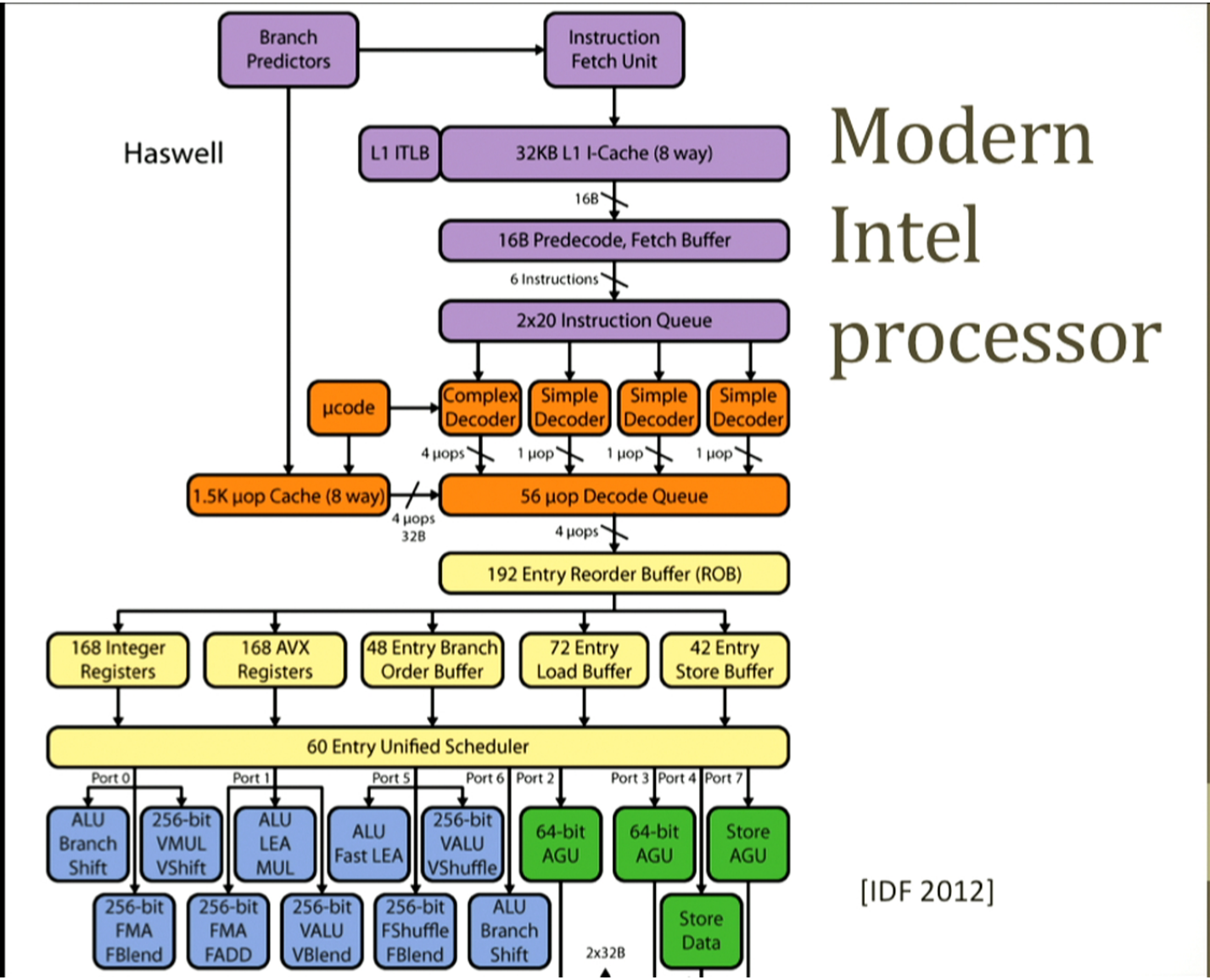
Sequential Execution:

x := a + b

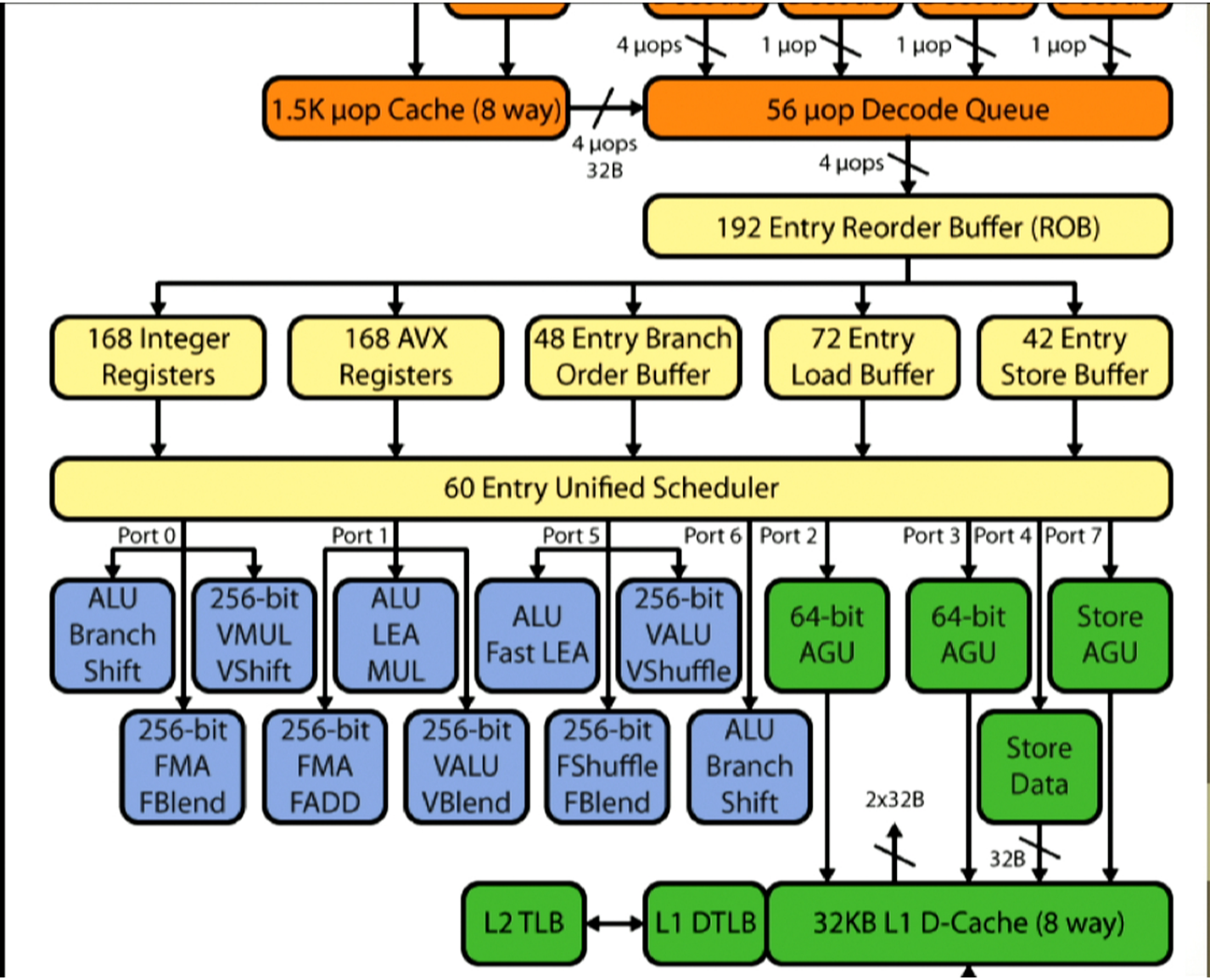
1. read instruction
2. read a
3. read b
4. add
5. write x

[Wikipedia]

Modern Intel processor



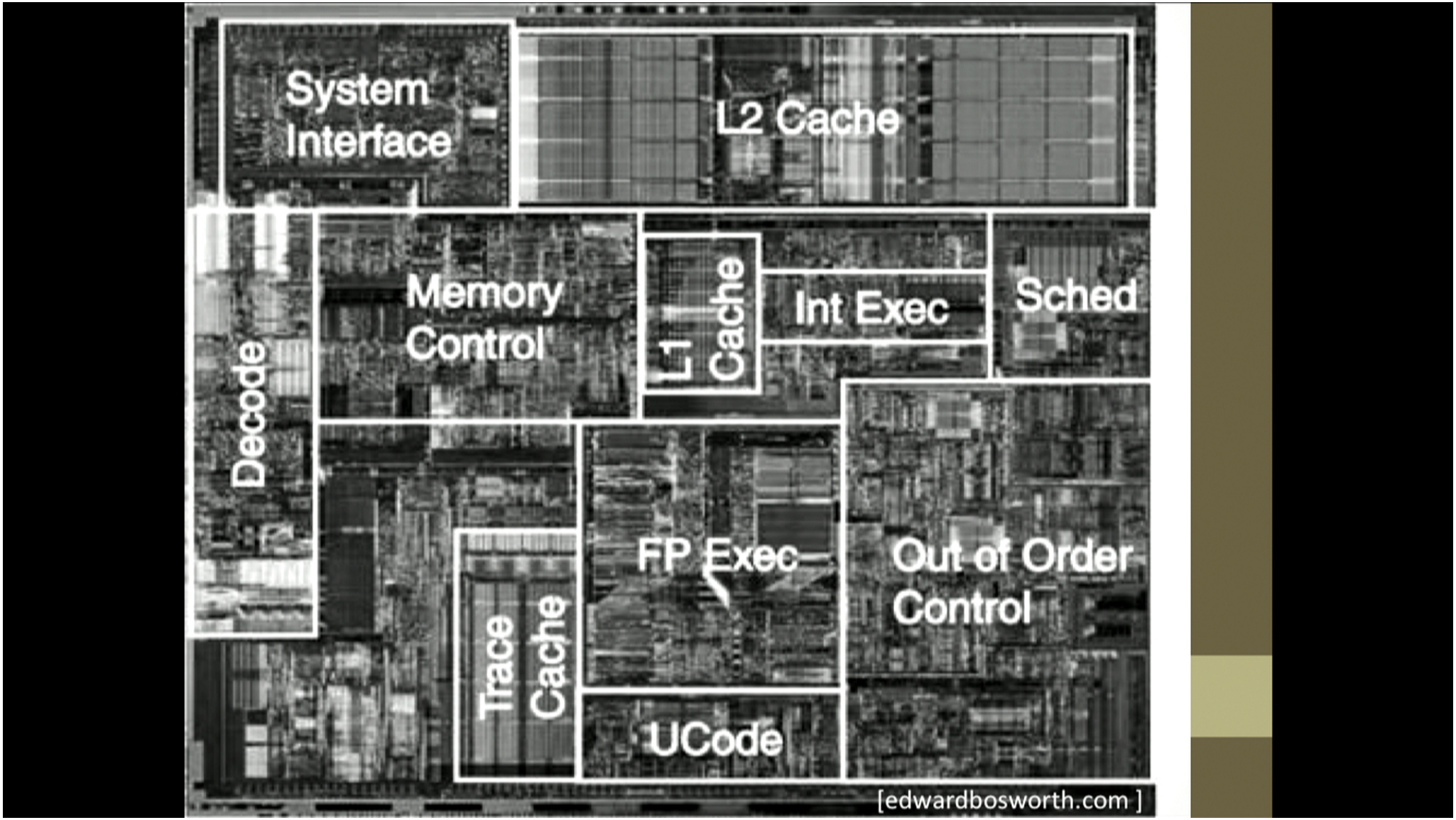
[IDF 2012]

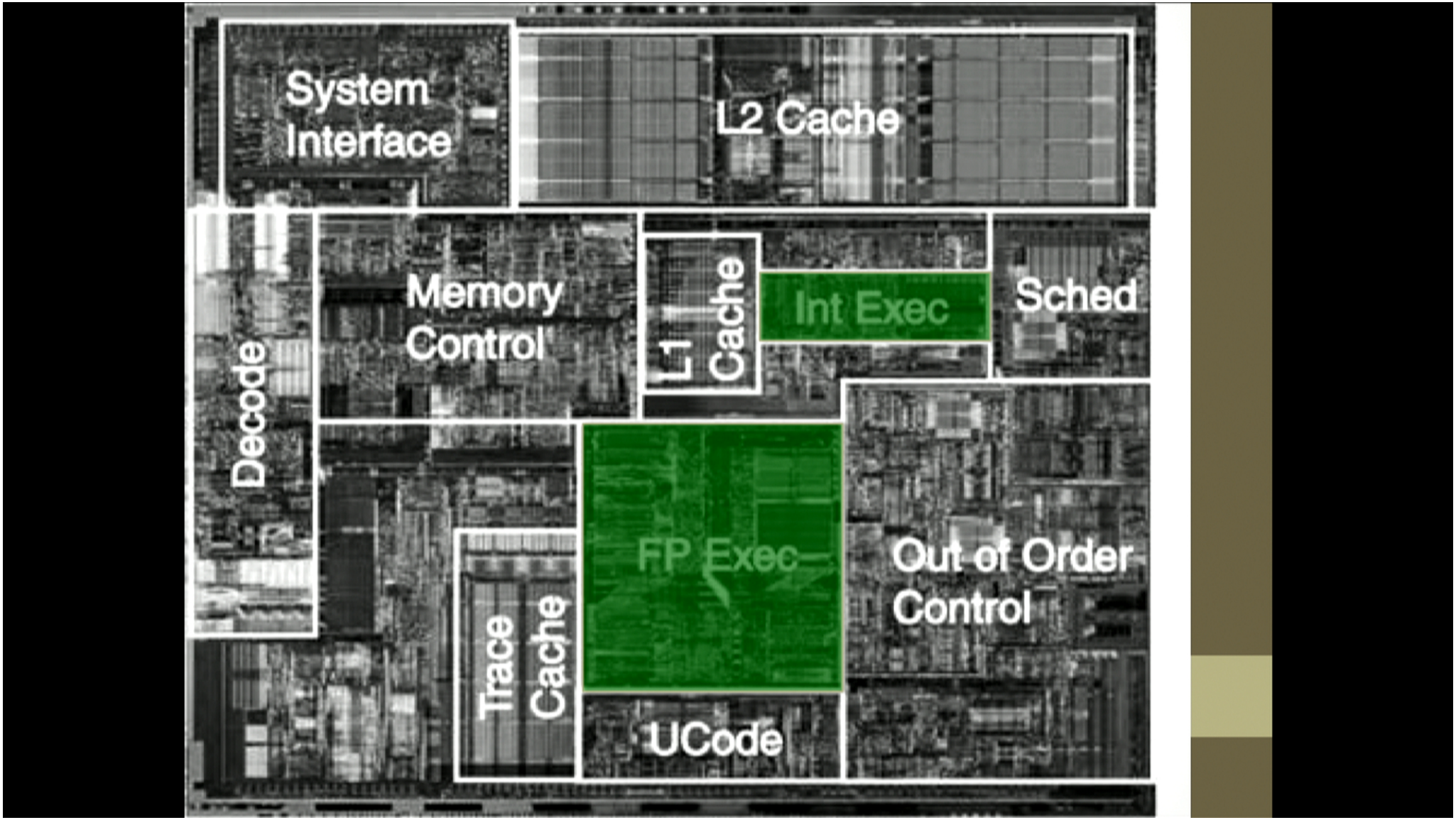

4 µops
1 µop
1 µop
1 µop
1.5K µop Cache (8 way)
4 µops
32B
56 µop Decode Queue
4 µops
192 Entry Reorder Buffer (ROB)
168 Integer Registers
168 AVX Registers
48 Entry Branch Order Buffer
72 Entry Load Buffer
42 Entry Store Buffer
60 Entry Unified Scheduler
Port 0
Port 1
Port 5
Port 6
Port 2
Port 3
Port 4
Port 7
ALU Branch Shift
256-bit VMUL VShift
ALU LEA MUL
ALU Fast LEA
256-bit VALU VShuffle
64-bit AGU
64-bit AGU
Store AGU
256-bit FMA FBlend
256-bit FMA FADD
256-bit VALU VBlend
256-bit FShuffle FBlend
ALU Branch Shift
Store Data
2x32B
32B
L2 TLB
L1 DTLB
32KB L1 D-Cache (8 way)

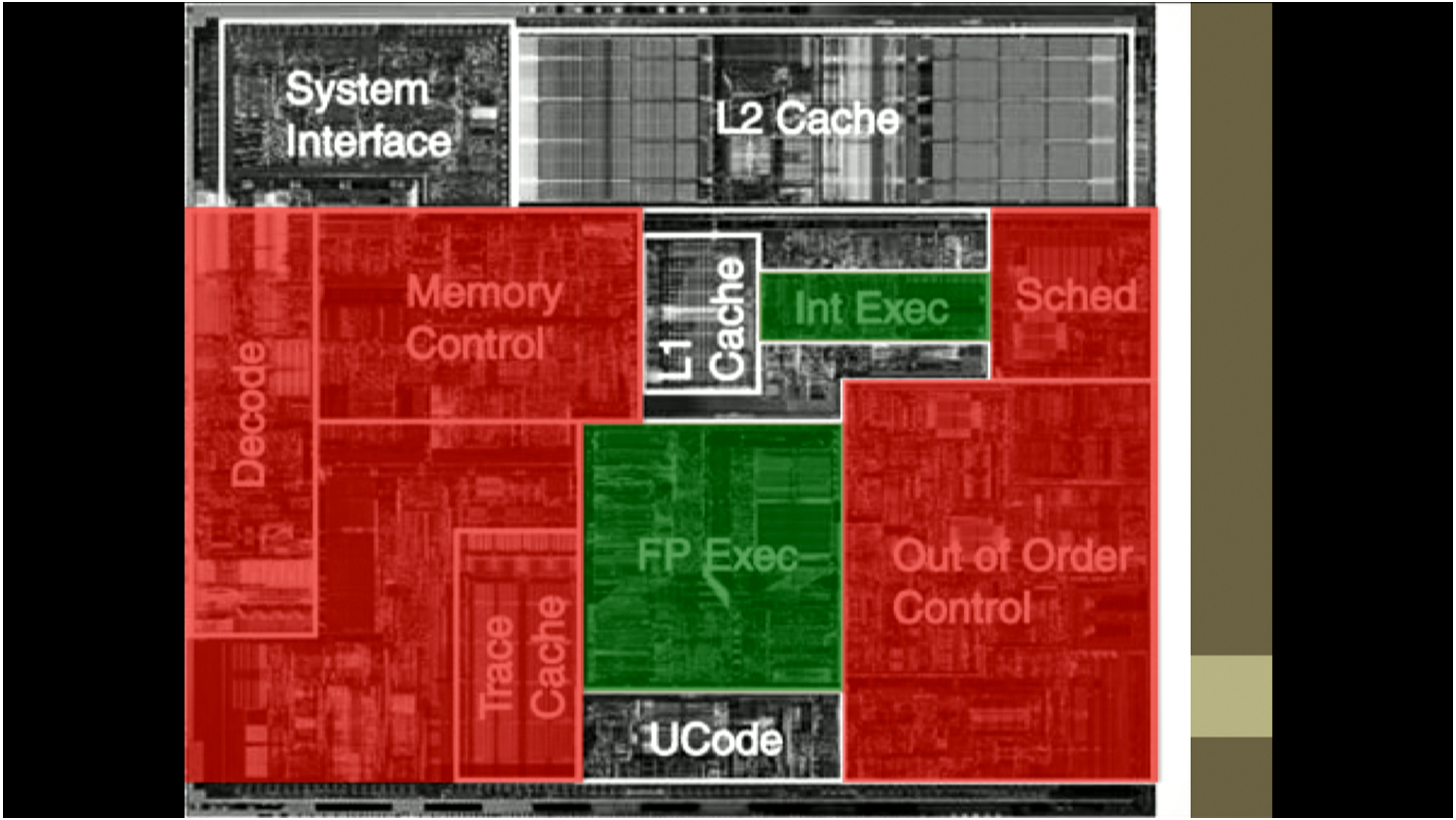
Sequential vs. Parallel

- Modern CPUs are highly parallel
  - Many cores (28)
  - Each core is superscalar (10…100)
  - Vector (“SIMD”) operations (4)
  - FMA (a*b+c) (2)

- Algorithms need to be parallel (many independent steps), not sequential (each step depending on the previous)
  - Bad: Update a vector element-by-element
  - Good: Create a new vector from an existing one, each element independently


System Interface
L2 Cache
Memory Control
L1 Cache
Int Exec
Sched
Decode
Trace Cache
FP Exec
Out of Order Control
UCode
[edwardbosworth.com]


System Interface
L2 Cache
Decode
Memory Control
L1 Cache
Int Exec
Sched
FP Exec
Out of Order Control
Trace Cache
UCode


System Interface
L2 Cache
Memory Control
L1 Cache
Int Exec
Sched
Decode
Trace Cache
FP Exec
Out of Order Control
UCode

Performance Measures

- *Theoretical Peak Performance*
  1. Computing: Floating point operations per second (Flop/sec)
  2. Data access: Bytes per second (Byte/sec)

- Typical numbers for a modern workstation:
  - 1,000 GFlop/sec (Giga, 10^9)
  - 100 GByte/sec
  - Ideal ratio: 0.1 Byte/Flop
  - Double precision add: 24 Byte/Flop

- Memory (RAM): 100 GByte

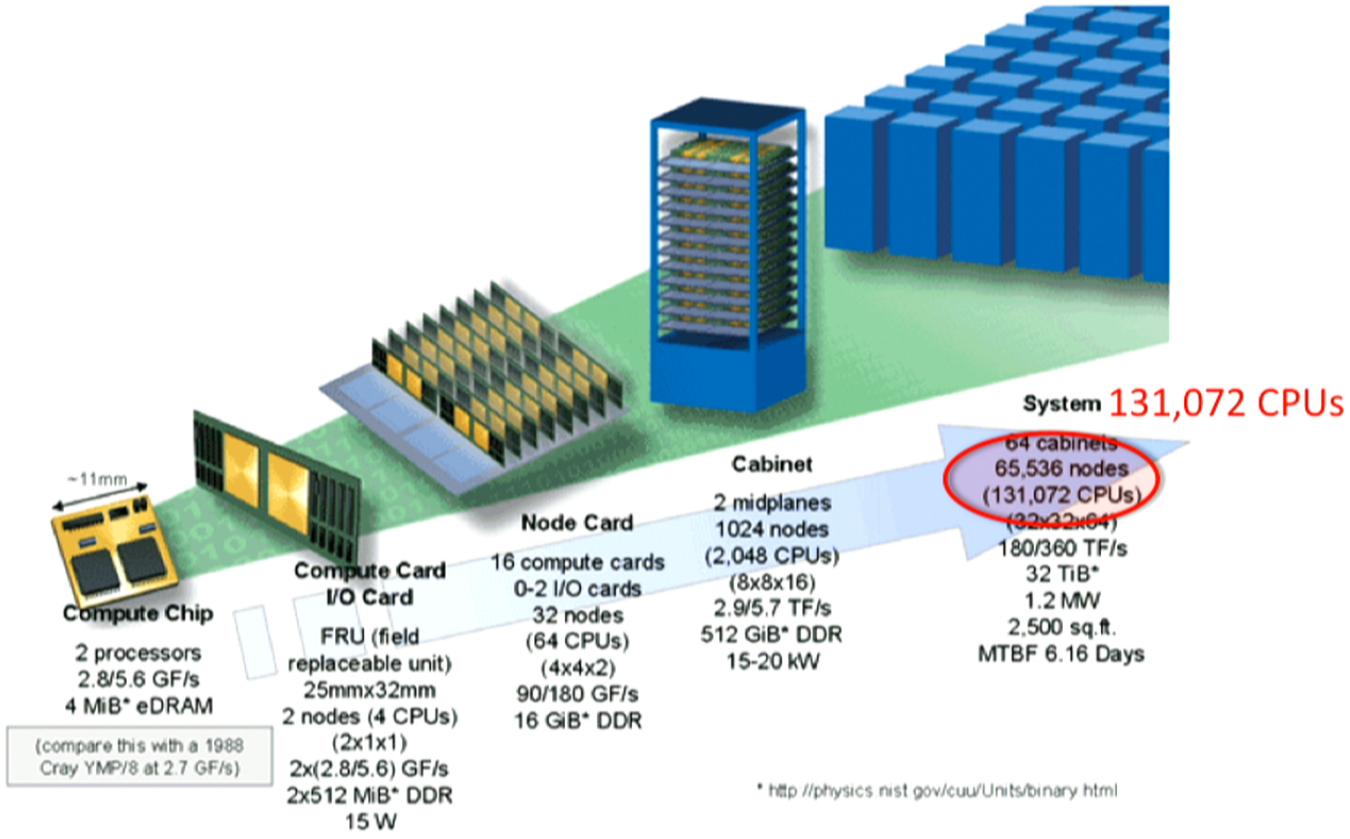
Algorithm characteristics

- Algorithms can be characterized by the number of calculations they perform (Flop) and amount of data they process (Byte)

- Matrix multiplication (n*n matrices, 8 Byte per number):
  - $2n^3$ Flop
  - $24n^2$ Byte
  - 12/n Byte/Flop
  - Thus: compute bound even for moderate n

- Matrix addition:
  - n^2 Flop
  - $24n^2$ Byte
  - Ratio: 24 Byte/Flop
  - Thus: memory bound

Supercomputer architecture

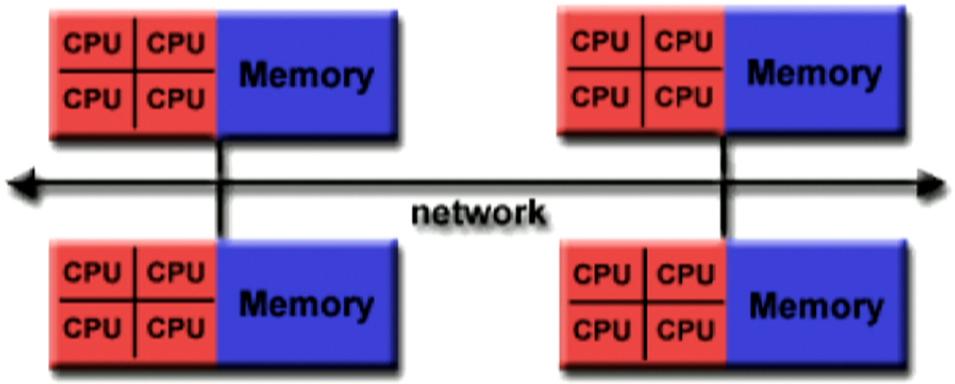


Supercomputer?

Top500

[top500.org]

Distributed Memory



[computing.llnl.gov]

- Need to *partition* data structures
- Need to *exchange information* at (artificial) inter-node boundaries
- Permeates the whole program, quite difficult in practice

Software Architecture

- Every decade or two, there is a paradigm shift in software architecture
  - 1980s: modular programming (“information hiding”)
  - 1990s: object oriented programming (generic programming)
  - 2000s: parallel programming (multi-threading)
  - 2010s: (?) functional (declarative) programming

Parallel Programming Algebra

- Problems: Access conflicts, deadlocks
  - Naively parallel programs do not form a group
- Relatively recent solution, found today e.g. in C++, Python, Julia:
  - *Future*: "black box" holding the result of a calculation that is not yet finished
  - Accessing a future automatically waits until the result is ready
  - *Async*: (asynchronous execution): "function call" returning a future, without waiting for the result
  - C++: Future is a "container" (or could be)

Parallel Programming Algebra

- Problems: Access conflicts, deadlocks
  - Naively parallel programs do not form a group
- Relatively recent solution, found today e.g. in C++, Python, Julia:
  - *Future*: "black box" holding the result of a calculation that is not yet finished
  - Accessing a future automatically waits until the result is ready
  - *Async*: (asynchronous execution): "function call" returning a future, without waiting for the result
  - C++: Future is a "container" (or could be)

Functional Programming

- Dichotomy between mathematics and programming:
  1. Math is about eternal truths (there is no "time" in a proof)
  2. Programs execute sequentially
- How can one prove statements about programs?

- Functional programming:
  - Design programs to be order-independent (as much as possible)
  - Remove distinction between data and functions

```
let x = 1
    sin(x)
end
```

```
let f = sin
    f(1)
end
```

julia

- From juliacode.org:
  - high-level, high-performance dynamic programming language for technical computing
  - sophisticated compiler
  - distributed parallel execution
  - numerical accuracy
  - extensive mathematical function library
  - mature, best-of-breed open source C and Fortran libraries for linear algebra, random number generation, …
  - powerful browser-based graphical notebook interface
- Note: only few years old, still immature

Summary

- Efficiency of algorithms depends on available computing architecture
  - Parallelism (independent operations, not sequential)
  - Data locality (operate on “neighbouring” data)
- Large problems require distributed memory
  - Partition data structure
- Modern software engineering:
  - Powerful concepts from group theory, category theory