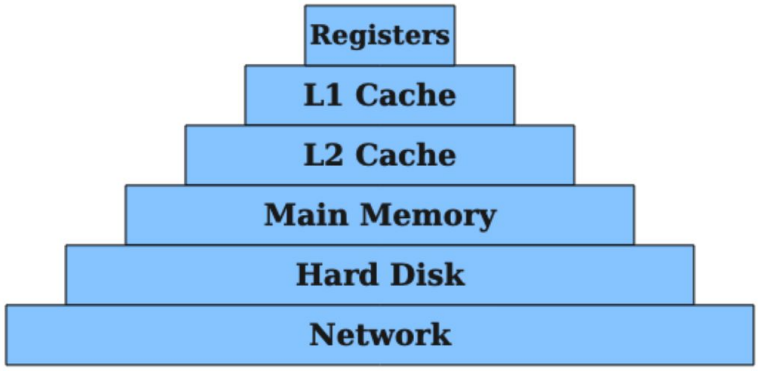
Let's add **register allocation** to our compiler

faster, smaller



256B - 8KB	0.25 – 1ns
16KB – 64KB	1ns – 5ns
1MB - 4MB	5ns – 25ns
4GB – 256GB	25ns – 100ns
500GB+	3 – 10ms
HUGE	10 – 2000ms

slower, bigger

(via Max New)

So far: all variables/values stored on stack (or heap) (easy, but slow)

Next: Use the REGISTERS 3-10x performance gains, variable access are ubiquitous!

```
(let ((a0 92)
      (a1 (add1 a0))
      (a2 (add1 a1))
      (a3 (add1 a2))
      (a4 (add1 a3))
      (a5 (add1 a4)))
 a5)
```

```
mov rax, 184
mov [rbp - 8*2], rax
mov rax, [rbp - 8*2]
add rax, 2
mov [rbp - 8*3], rax
mov rax, [rbp - 8*3]
add rax, 2
mov [rbp - 8*4], rax
mov rax, [rbp - 8*4]
add rax, 2
mov [rbp - 8*5], rax
mov rax, [rbp - 8*5]
add rax, 2
mov [rbp - 8*6], rax
mov rax, [rbp - 8*6]
add rax, 2
mov [rbp - 8*7], rax
mov rax, [rbp - 8*7]
```

Let's add **register allocation** to our compiler

Example 1

```
(let ((a1 (+ 10 10))
      (a2 (* 2 a1))
      (a3 (* 3 a2)))
  (* 10 a3))
```

Example 2

```
(let ((n (* 5 5))
      (m (* 6 6))
      (x (+ n 1))
      (y (+ m 1)))
 (+ x y)
)
```

Example 3

```
(defn (f a)
  (let ((x  (* a 2))
        (y  (+ x 7)))
    y))
```

Example 4

```
(defn (f a)
  (let ((x  (* a 2))
        (y  (+ x 7)))
    (g x y)))
```

But … what if the programmer *instead* wrote

Example A

```
(* 10 (* 3 (* 2 (+ 10 10))))
```

Example B

```
(+ (+ (* 5 5) 1) (+ (* 6 6) 1))
```

Example C

```
(defn (f a)
  (+ (* 2 a) 7))
```

1. Administrative Normal Form (ANF)

Immediate expressions: whose values don't require any computation!

- **Constant**, e.g. `1, true, false`,
- **Variable,** e.g. `x, y, z` (whose value is on the stack/reg)

An expression is in ANF when all **primitive operations** have **immediate** arguments

QUIZ: ANF? Yes or No : Example 1, 2, 3, 4, A, B, C

Expr	ANF Expr

```
(+ (+ 1 2) 3)
```

```
(+ (+ (+ 1 2) 3) 4)
```

```
(+ (+ (+ 1 2) 3)
   (+ (+ 4 5) 6))
```

2. Compiling with `Alloc`ations

3. Computing Allocations by Graph Coloring