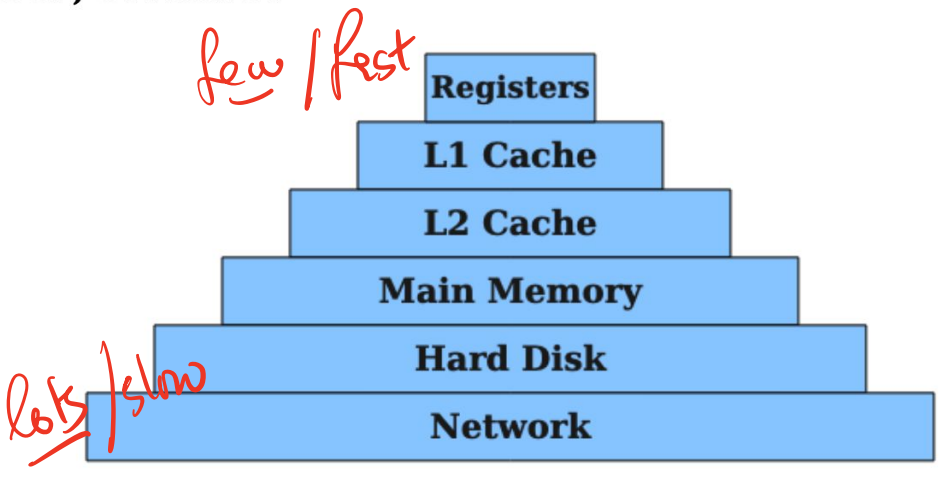
Let's add **register allocation** to our compiler

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]
```

```
mov rbx, 184
add rbx, 2
add rbx, 2
add rbx, 2
add rbx, 2
add rbx, 2
mov rax, rbx
```

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

fn ANF : (&Expr) → Expr

in out

Alloc

Var ↦ Loc

Reg (RAX, RBX...)

Stak (i32)

Example 1

Ex 1

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

```
mov rax, 10
add rax, 10
mul rax, 2
mul rax, 3
mul rax, 10
```

(* 10 (* 3 (* 2 (+ 10 10))))
a3 a2 a1 a0

Ex A

Example 2

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

```
mov rax, 5
mul rax, 5
mov rbx, 6
mul rbx, 6
add rax, 1
add rbx, 1
add rax, rbx
```

n ↦ rax
m ↦ rbx

x ↦ rax
y ↦ rbx

Example 3

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

a ↦ rbp+16
x ↦ rax

```
mov rax, [rbp+16]
mul rax, 2
add rax, 7
```

(if e1 e2 e3)

f (+1 3) (let...) (...)

```
<e1>
cmp rax, (TRUE)
jne else_cond
then_cond:
  <e2>
  jmp exit
else_cond:
  <e3>
exit:
```

Example 4

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

3 regs x, y, g

(if (= x y) ...)

```
(defn (foo x y)
  (if (let (b (=xy)) b)
      (let (a (+x1)) (*a 99))
      (* y 10)))
```

ANF

But … what if the programmer *instead* wrote

Example A

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

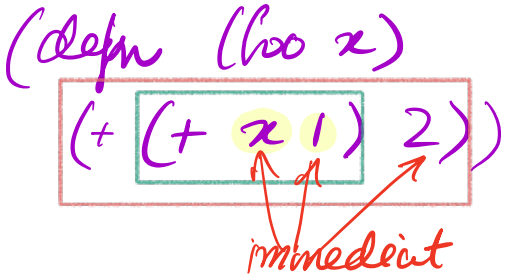
Example B

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

Example C

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

$x_1 \rightarrow$ rax
$x_2 \rightarrow$ rbx
$x_3 \rightarrow$ rbx
a_1
a_K



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)`	`(let (t0 (+ 1 2)) (+ t0 3))`
`(+ (+ (+ 1 2) 3) 4)`	`(let ((t0 (+ 1 2)) (t1 (+ t0 3))) (+ t1 4))`
`(+ (+ (+ 1 2) 3) (+ (+ 4 5) 6))`	`(let* ((t0 (+ 1 2)) (t1 (+ t0 3)) (t2 (+ 4 5)) (t3 (+ t2 6))) (+ t1 t3))`

imm (Exp) → Vec<(tmp, Expr)>, Imm

[(t0, (+ 1 2))], t0

imm

2. Compiling with Allocations

x r8 y r9

(+ i1 i2)

dst → r8, r10

let reg = imm_reg(env, dst)

```
mov  rax, <i1>
add  rax, <i2>
mov  <dst>, rax
```

```
mov? reg, <i1>
add  reg, <i2>
mov? dst, reg
```

i2r i1

x ← a + b

3. Computing Allocations by Graph Coloring