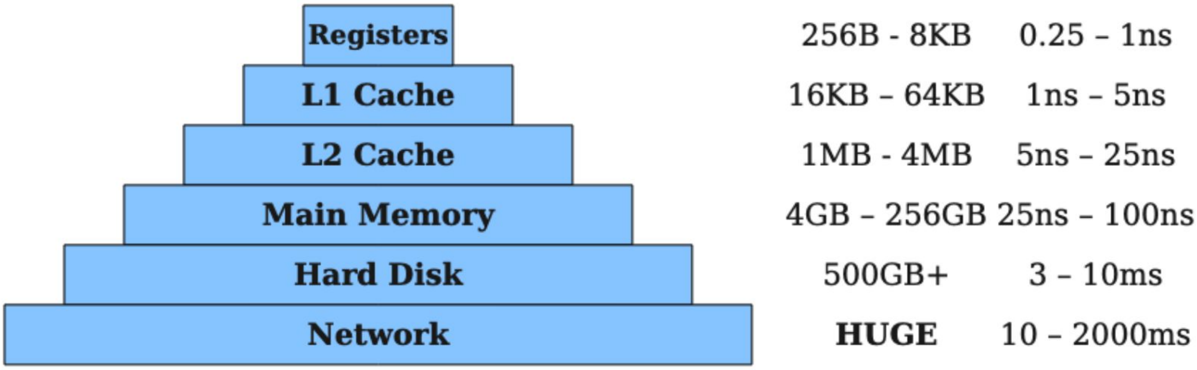
Let’s add **register allocation** to our compiler

faster, smaller



| | | |
|---|---|---|
| Registers | 256B - 8KB | 0.25 – 1ns |
| L1 Cache | 16KB – 64KB | 1ns – 5ns |
| L2 Cache | 1MB - 4MB | 5ns – 25ns |
| Main Memory | 4GB – 256GB | 25ns – 100ns |
| Hard Disk | 500GB+ | 3 – 10ms |
| Network | **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
```

# 2. Compiling with Allocations

```
fn compile_expr(e: &Expr, env: &Alloc, count: &mut i32, brk: &str, dst: &Loc)
   -> String {
   match e {
      Expr::Num(_) | Expr::True | Expr::False | Expr::Var(_) | Expr::Input => {


      }
      Expr::Add1(i) => {



      }
      Expr::Plus(i1, i2) => {



      }
      Expr::Eq(i1, i2) => {





      }
      Expr::Call(f, is) => {





      }
}
```

# 3. Computing Allocations by Graph Coloring

## 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)))
```

# 3. Computing Allocations by Graph Coloring

```
fn live(
   graph: &mut ConflictGraph,
   e: &Expr,
   binds: &HashSet<String>,
   params: &HashSet<String>,
   out: &HashSet<String>,
) -> HashSet<String> {
   match e {
      Expr::Num(_) | Expr::True | Expr::False | Expr::Input => {

      }
      Expr::Var(x) => {

      }
      Expr::Plus(i1, i2) => {

      }
      Expr::If(e1, e2, e3) => {

      }
      Expr::Let(x, e1, e2) => {

      }
      Expr::Call(f, is) => {

      }
      Expr::Loop(e) => {


      }
}
```