Dynamic Type Checking

what happens?

```
.section .text
LA t1, 0x20000000
LI t2, 17               ;t2 = 17
SW t2, 0(t1)            ;*t1 = t2
FLW f1, 0(t1)           ;f1 = *t1
```

In a “real” machine:

t2 = 0000 0000 0000 0000 0000 0001 0001 (17 in binary)

so

f1 = 0000 0000 0000 0000 0000 0001 0001 (2.38e-44 in floating point)

what happens?

```
.section .text
LA t1, 0x20000000
LI t2, 17              ;t2 = 17
SW t2, 0(t1)           ;*t1 = t2
FLW f1, 0(t1)          ;f1 = *t1
```

On our simulator:

AssertionError: Value in memory not of type <class 'float'>

what happens?

Our simulator does some basic **dynamic type checking**

- Keeps track of the type of data stored in memory
- Makes sure that loads and stores respect that type
  - Cannot load an integer value into a floating point register, and vice versa

what is dynamic type checking?

- Types constrain behavior of a program
- If those constraints are not respected, a program can produce weird behavior
  - Or worse, have a security vulnerability!
- Dynamic type checking *checks those constraints at runtime* to turn constraint violations into runtime errors
- Which constraints are checked, and where, is up to the language/runtime

dynamic checks in python

- Makes sure that operations only work on valid types

  10 + “x” → TypeError: unsupported operand type(s) for +: 'int' and ‘str’

- Makes sure that list accesses are valid

  x = 5 * [0] ; x[6] → IndexError: list index out of range

- *Doesn’t* check that functions are called with the right types (why?)

how does dynamic type checking work?

- Data carries along *meta-data* that specifies type information
  - Data type, lengths of strings, sizes of arrays, whether a reference is null, etc.

- At **run-time** this meta-data is used to check constraints before performing operations that might give bad behavior if constraints are violated
  - Not all constraints are checked all the time!

what to check?

- Different languages make different choices about what to check
  - Java will check that array access are in bounds, C will not
  - C++ will (sort of) check that downcasts succeed, Java will give a better runtime error
- What happens if a constraint is not checked?
  - Can cause an error lower in the system stack, e.g., a segmentation fault
  - Can cause silent problems (lots of security vulnerabilities!)

when to check?

- Dynamic type checking requires *run-time* processing
  - Adds overhead!
  - Array accesses in Java are much slower than array accesses in C
- In some circumstances, can offload some of the work of type checking to the compiler, check *before the program even runs*
- This is called **static type checking**!