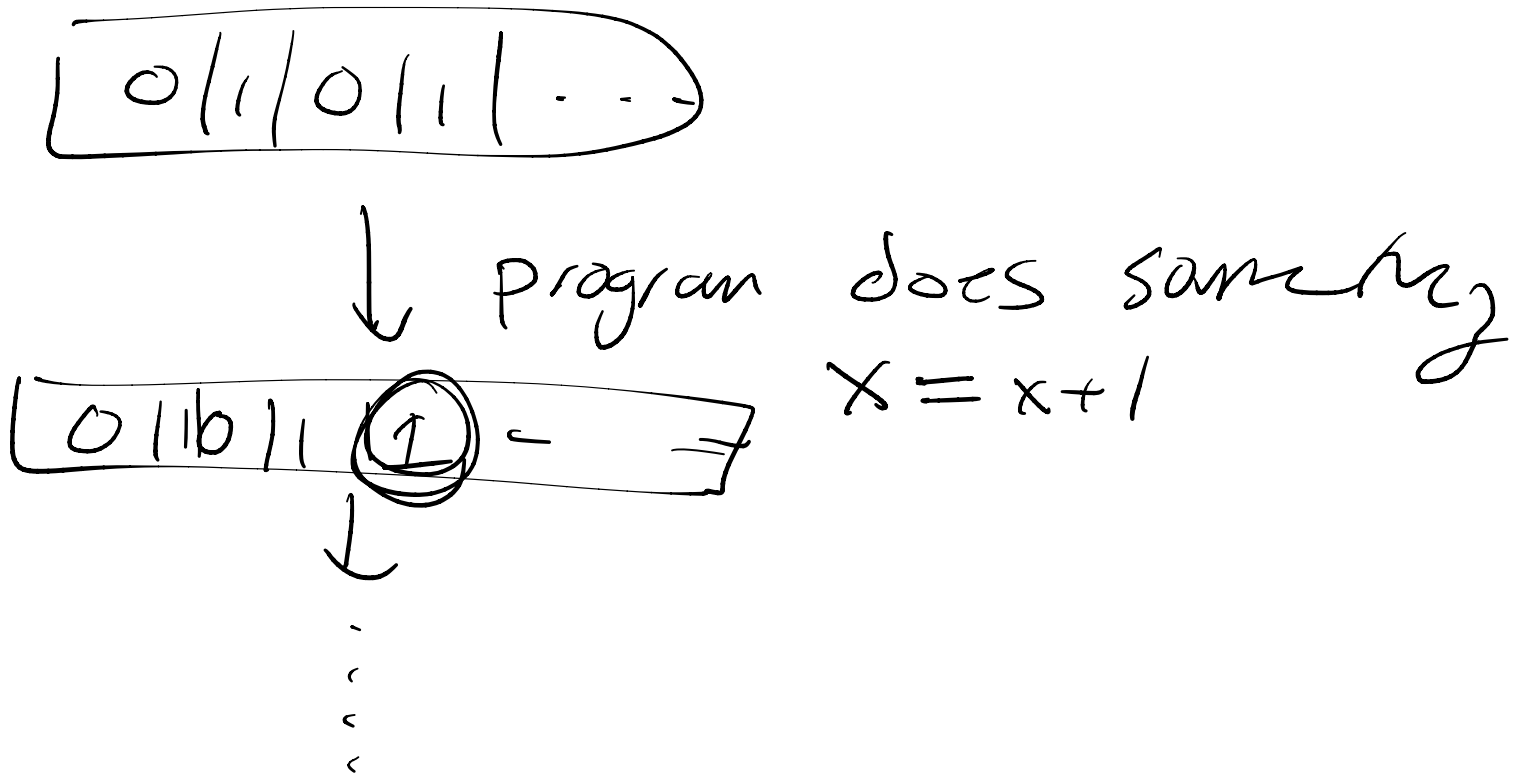
Lecture 2:

Computing by Calculation

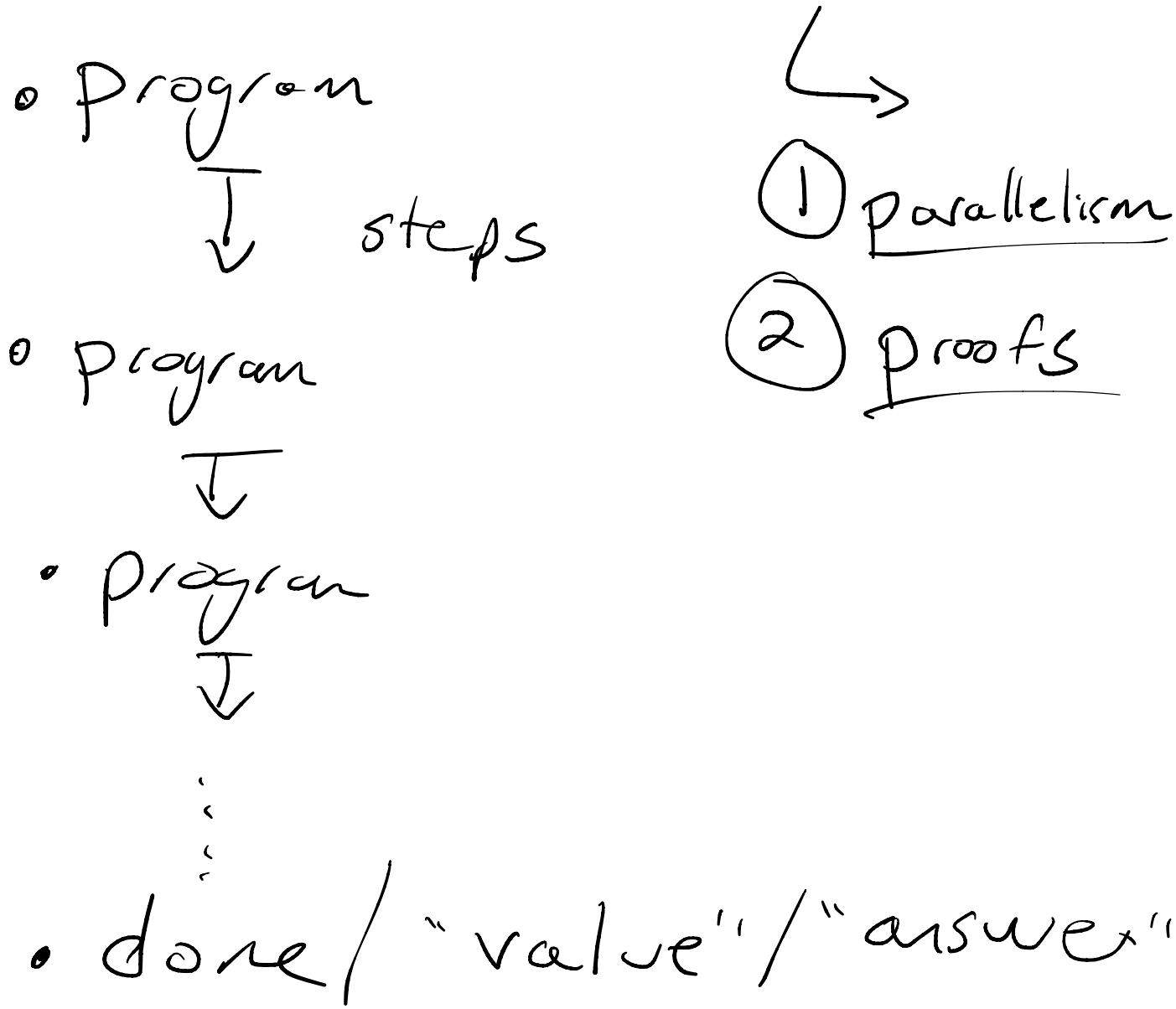
C / Python / ...

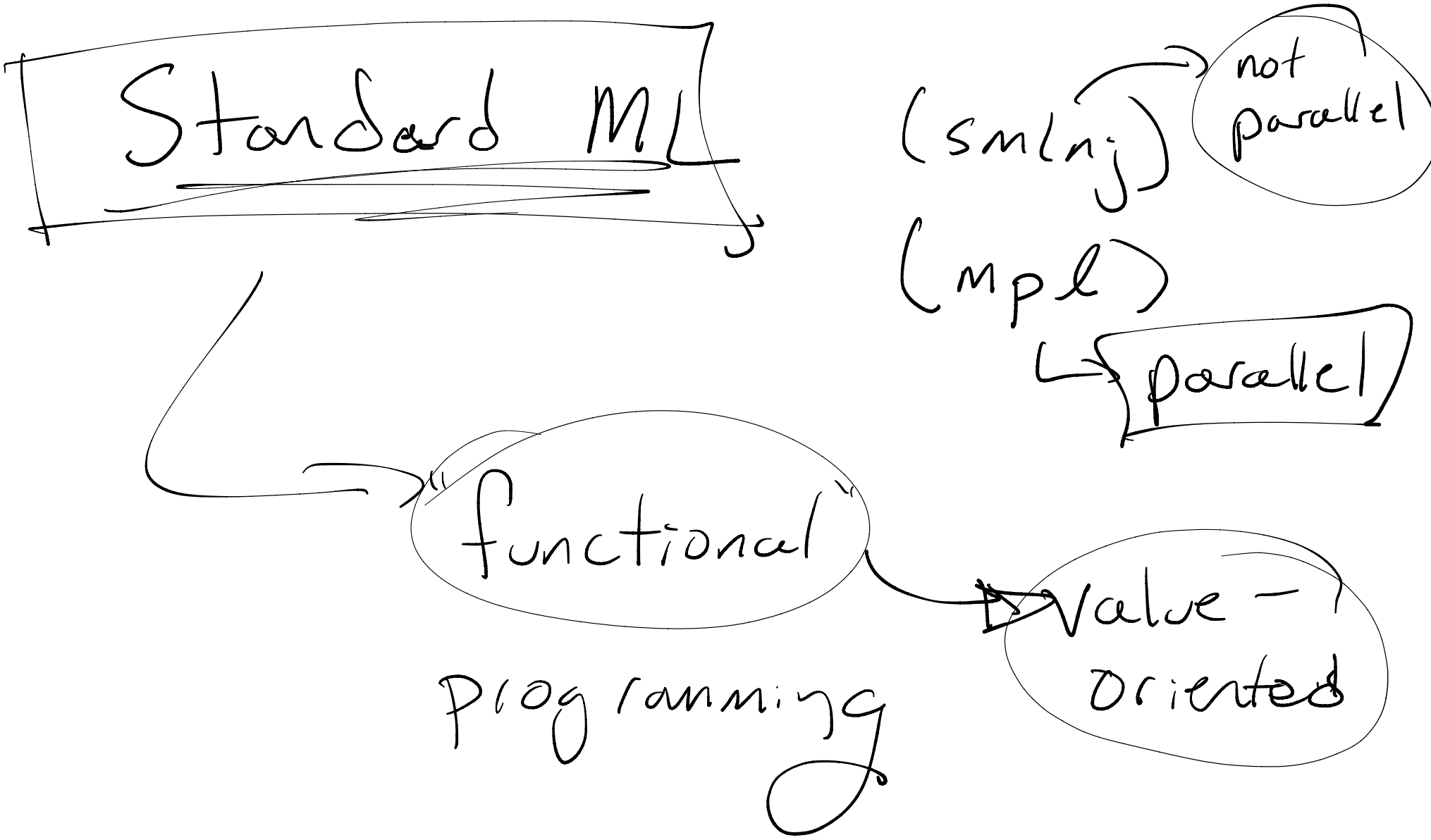
"imperative"

memory-based execution model



Program / expression - centric model




Standard ML
(smlnj)
not parallel
(mpl)
parallel
functional
Value - Oriented
programming

Expressions/programs	Types	Values/answers
2	int	2
1+1	int	2
(1+1) * (3+4)	int	14
"a"	string	"a"
intToString 5	string	"5"
"a" + 1 → 92+1=93="b" → "a1"	no type	no value
5 div 0	int	no value

5 div 0 = NaN (not a number)

int int int

↳ int DIV

float

5.0 / 0.0 = NaN

floating

5 div 2 = 2

int int

Type

Compile-time errors

↳ when you write

5 div 1 : int

5 div x : int

Run-time errors

↳ when user runs

e.g. 5 div 0 raise an exception Div

$(1+1)*(3+4)$

$\longmapsto 2*(3+4)$

"step"

$\longmapsto 2*7$

$\longmapsto 14$

Value / answer

sequential

$(1+1)*(3+4)$

$\Longmapsto 2*7$

parallel step

$\Longmapsto 14$

data-dependencies

parallel

Every type has a collection of values/answers and operations

int

values 0, 1, 2, 3, ~1, ~2 ...
(negative 1)

operations + * div intToString ...

String

values "a" "b" "ab"

Operations ^

Real

Floating Point

Values 0.0, 3.14, ≈1.17

Ops + * / - ...

/ division

Type checking

(1) each of values has the indicated type

(2) each operation is well-typed when the subexpressions are and have the correct types

ill-typed otherwise

$(3+7)*5 : int$ because

has type

$(3+7) : int$ b/c

$3 : int$ b/c value

$7 : int$ b/c value

$5 : int$ b/c value of that type

"a" + 1 is? int b/c

"a" is int no! "a" is a string value

1 is int ✓

type error!

"a" ^ "1"
intToString

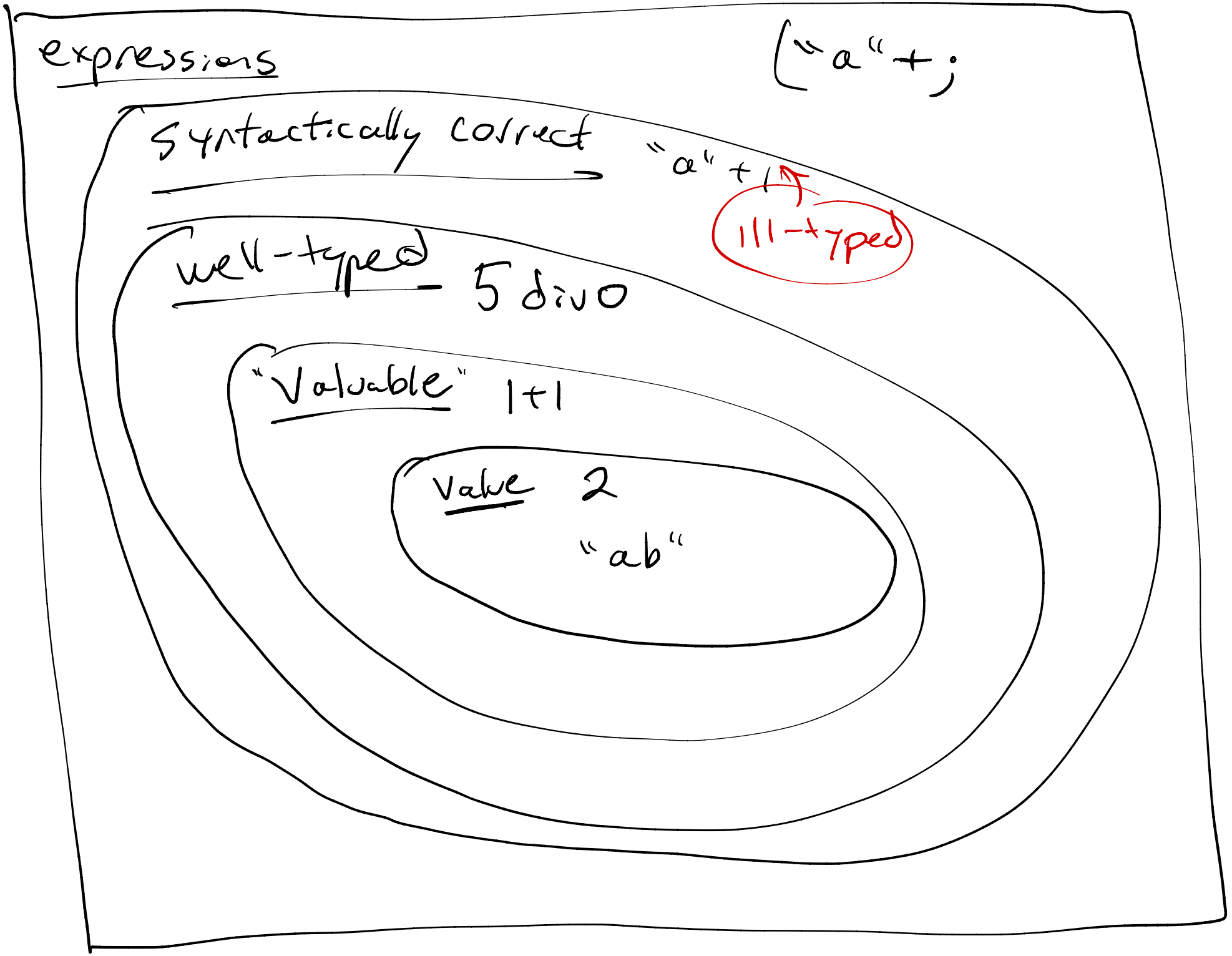
5 div 0 : int b/c

5 : int b/c value

0 : int ✓

---

run-time exception


expressions
("a"+;
syntactically correct
"a"+1
ill-typed
well-typed
5 div 0
"Valuable"
1+1
value
2
"ab"

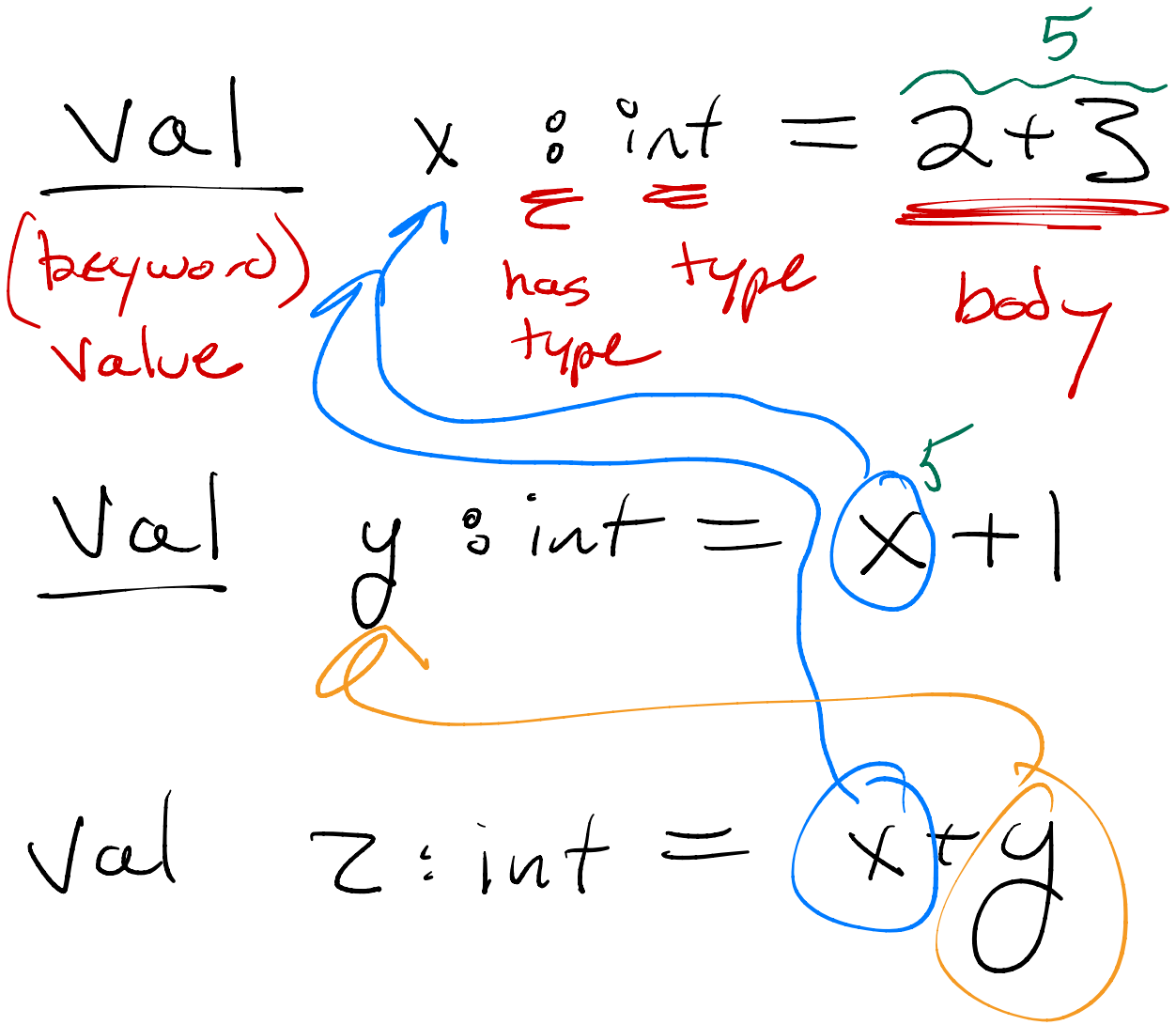
Variables

Imperative: (C) a variable is a name for a location in memory

Functional: a variable is a placeholder for a value

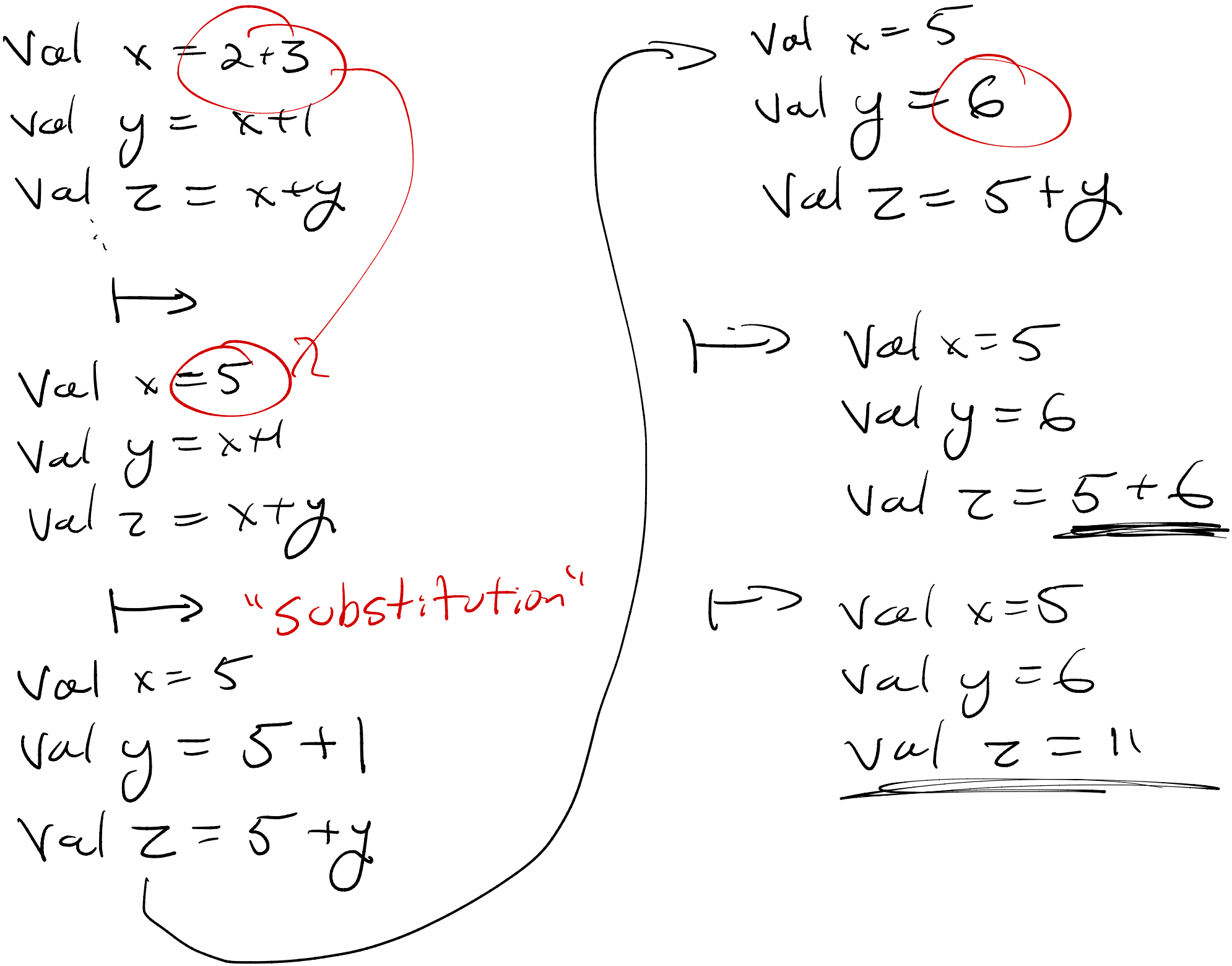
→ variables don't vary

Variable declarations



(1) body should have the type

(2) Assume variable has that type in rest of your code


val x = 2+3
val y = x+1
val z = x+y
↦
val x = 5
val y = x+1
val z = x+y
↦ "substitution"
val x = 5
val y = 5+1
val z = 5+y
val x = 5
val y = 6
val z = 5+y
↦
val x = 5
val y = 6
val z = 5+6
↦
val x = 5
val y = 6
val z = 11

```
val x = 5
val y = x + 1
val x = 3
val z = x + 1
```

Variables refer to the "nearest" "enclosing" binder declaration

```
val x = 5
val y = x + 1
val w = 3
val z = w + 1
```

Functions

capture a pattern of computation

math $f(x) = 2x + 6$

SML `fun f (x:int):int = (2 * x) + 6`

- `fun` — keyword
- `f` — name of fun
- `x` — name of input
- `int` (after `x:`) — type of input
- `int` (after `):`) — type of the output
- `(2 * x) + 6` — body

Type check:

1. variable is assumed to have input type in body
2. body must have the output type

```
fun f(x) = 2*x+6
```

Function application f(input)

f(3)

↦ (2*3)+6

↦ 6+6

↦ 12

substitute the input in for the input variable + step body

fun f(x) = 2*x + 6

f(1+1)

↦ 2*(1+1)+6

↦ 2*2+6

↦ 4+6

↦ 10

(call by name)

Haskell

f(1+1) "call by value"

↦ f(2)

↦ (2*2)+6

↦ 4+6

↦ 10

SML

evaluate the input to a function to a value, and then subst. that value into body

A function that takes fewer steps in CBV:

(some value)

A function that takes fewer steps in CBN:

Function application

f(a) has type B

If f is a fun f(x:A):B = ...

and a : A

e.g. fun intToString(x:int):string = ...

intToString(4) : string

:int