Implementing Classes

TICAE

types

```
{class posn extends object
  x : num   y : num
  {mdist : num -> num
         {+ {get this x} {get this y}}}
  {addDist : posn -> num
           {+ {send this mdist 0} {send arg mdist 0}}}}
{class posn3D extends posn
  z : num
  {mdist : num -> num
         {+ {get this z} {super mdist arg}}}}
{send {new posn3D 7 5 3} mdist 0}
```

⬇

ICAE

inheritance

super

```
{class posn extends object
  x y
  {mdist {+ {get this x} {get this y}}}
  {addDist {+ {send this mdist 0} {send arg mdist 0}}}}
{class posn3D extends posn
  z
  {mdist {+ {get this z} {super mdist arg}}}}
{send {new posn3D 7 5 3} mdist 0}
```

⬇

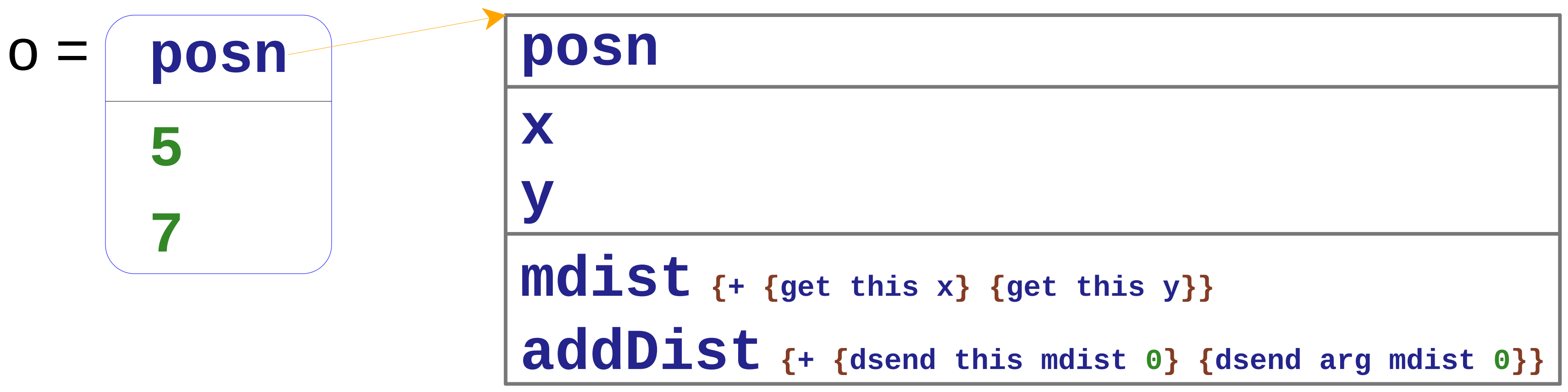
CAE

method dispatch

fields

```
{class posn
  x y
  {mdist {+ {get this x} {get this y}}}
  {addDist {+ {dsend this mdist 0} {dsend arg mdist 0}}}}
{class posn3D
  x y z
  {mdist {+ {get this z} {ssend this posn mdist arg}}}
  {addDist {+ {dsend this mdist 0} {dsend arg mdist 0}}}}
{dsend {new posn3D 7 5 3} mdist 0}
```

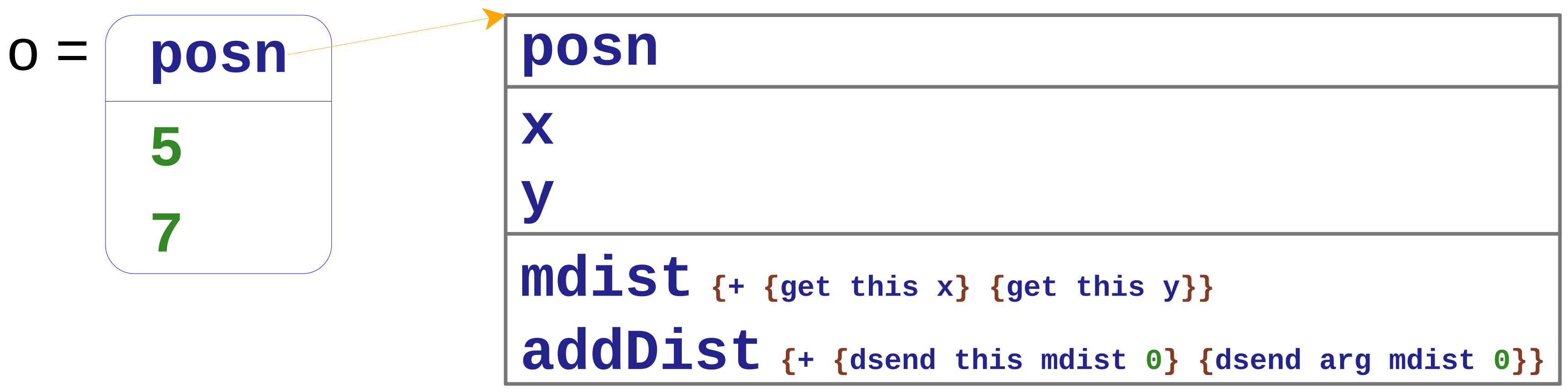
Run-Time Dispatch by Name


o =
posn
5
7
posn
x
y
mdist {+ {get this x} {get this y}}
addDist {+ {dsend this mdist 0} {dsend arg mdist 0}}

```
(dsend o mdist 0)
```

dsend follows reference to class table, searches method list

Run-Time Dispatch by Name



o =
posn
5
7
posn
x
y
mdist {+ {get this x} {get this y}}
addDist {+ {dsend this mdist 0} {dsend arg mdist 0}}

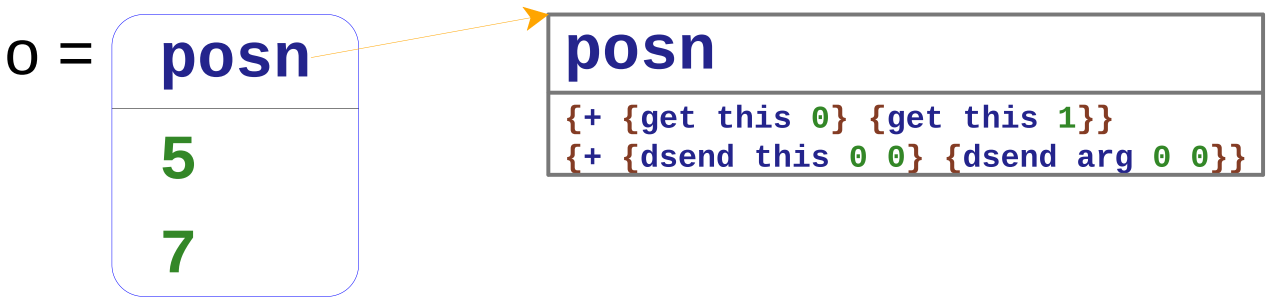
```
(dsend o mdist 0)
```

```
{class posn extends object
  x : num   y : num
  {mdist : num -> num
         {+ {get this x} {get this y}}}
  {addDist : posn -> num
           {+ {send this mdist 0} {send arg mdist 0}}}}
```

$\Rightarrow$ typechecking ensures search will succeed

If we order methods in expansion, method will always be first in list

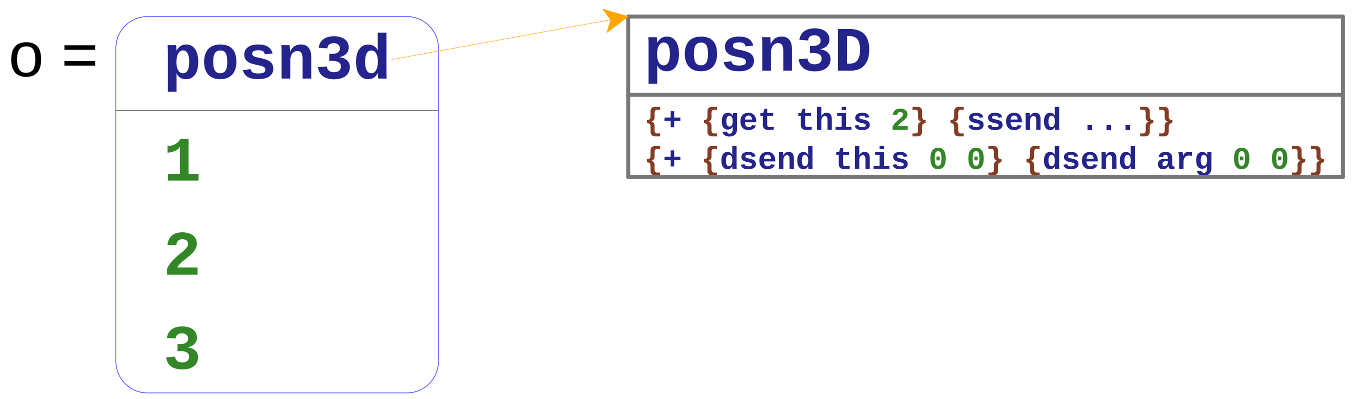
Run-Time Dispatch by Position



```
{+ {get this 0} {get this 1}}
{+ {dsend this 0 0} {dsend arg 0 0}}
```

```
(dsend o 0 0)
```

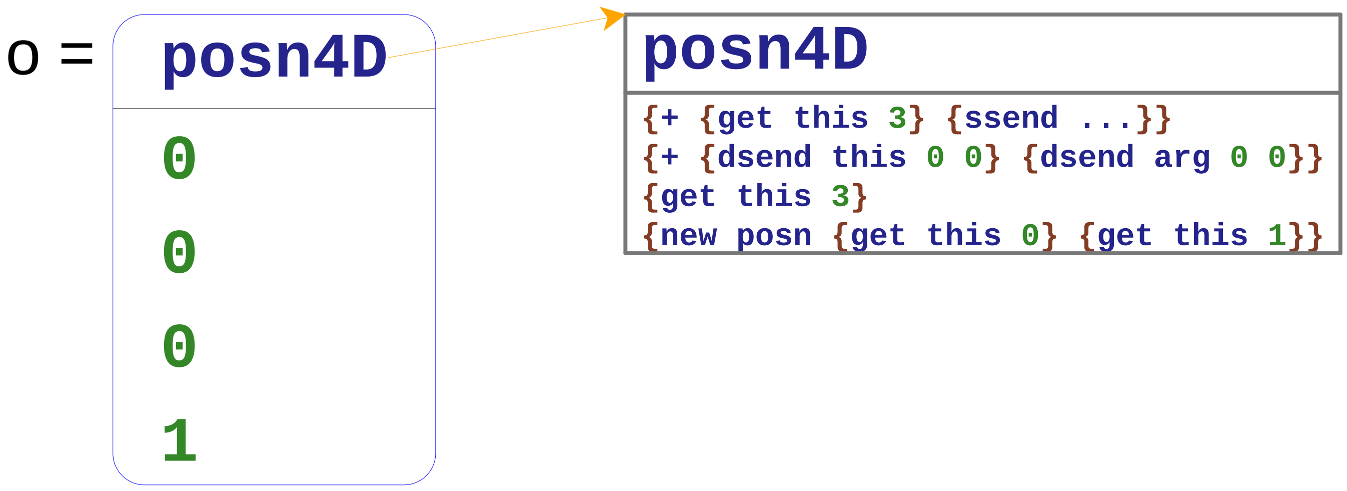
Run-Time Dispatch by Position



```
{+ {get this 2} {ssend ...}}
{+ {dsend this 0 0} {dsend arg 0 0}}
```

```
(dsend o 0 0)
```

Run-Time Dispatch by Position



```
posn4D
{+ {get this 3} {ssend ...}}
{+ {dsend this 0 0} {dsend arg 0 0}}
{get this 3}
{new posn {get this 0} {get this 1}}
```

```
(dsend o 0 0)
```

Compiling Classes

TICAE

no change

```
{class posn extends object
  x : num  y : num
  {mdist : num -> num
         {+ {get this x} {get this y}}}
  {addDist : posn -> num
           {+ {send this mdist 0} {send arg mdist 0}}}}
{class posn3D extends posn
  z : num
  {mdist : num -> num
         {+ {get this z} {super mdist arg}}}}
{send {new posn3D 7 5 3} mdist 0}
```

CICAE

name class in each method call

```
{class posn extends object
  x y
  {mdist {+ {get this x} {get this y}}}
  {addDist {+ {send this posn mdist 0} {send arg posn mdist 0}}}}
{class posn3D extends posn
  z
  {mdist {+ {get posn3d this z} {super mdist arg}}}}
{send {new posn3D 7 5 3} posn3d mdist 0}
```

CCAE

methods and fields as positions

```
{class posn 2
  {mdist {+ {get this 0} {get this 1}}}
  {addDist {+ {dsend this 0 0} {dsend arg 0 0}}}}
{class posn3D 3
  {mdist {+ {get this z} {ssend this {+ {get this 0} {get this 1}}
                                arg}}}
  {addDist {+ {dsend this 0 0} {dsend arg 0 0}}}}
{dsend {new posn3D 7 5 3} 0 0}
```

CCAE Revised Datatypes

```
type cae =
    ...
  | Get of cae * int
  | DSend of cae * int * cae
  | SSend of cae * cae * cae

and cdecl = Class of string * int * cae list
```

CCAE Revised Interpreter

```
let rec interp : (cae * cdecl list * caeValue * caeValue
                 -> caeValue )
    = function (expr, cdecls, this, arg) ->
      let recur = fun e -> interp(e, cdecls, this, arg)
      in match expr with
      …
    | Get(expr, n) ->
        (match recur expr with
          ObjV(_, vals) ->
            List.nth vals n
        | _ -> raise (Failed "not an object for get"))
    | DSend(expr, n, argExpr) ->
        (match recur expr with
          (ObjV(Class(_, _, methods), _) as this) ->
            let body = List.nth methods n
            in interp(body, cdecls, this, recur argExpr)
        | _ -> raise (Failed "not an object for send"))
    | SSend(expr, body, argExpr) ->
        let this = recur expr
        in interp(body, cdecls, this, recur argExpr)
```

CICAE Revised Datatypes

```
type icae =
     …
   | IGet of icae * string * string
   | ISend of icae * string * string * icae
     …
```

CICAE Revised Compiler

```
let rec compileExpr = function
    (expr, thisClass, idecls) ->
      let recur = fun expr -> compileExpr(expr, thisClass, idecls)
      in match expr with
        …
      | IGet(expr, cname, fname) ->
          let IClass(_, sname, fields, _) = findIClass cname idecls
          in Get(recur expr, ((locateIField fname fields)
                               + classFieldCount(sname, idecls)))
      | ISend(expr, cname, mname, argExpr) ->
          let IClass(_, _, _, methods) = findIClass cname idecls
          in DSend(recur expr,
                   locateIMethod mname methods,
                   recur argExpr)
      | ISuper(mname, expr) ->
          let IClass(_, sname, _, _) = thisClass
          in let super = findIClass sname idecls
          in let IClass(_, _, _, methods) = super
          in let IMethod(_, body) = findIMethod mname methods
          in SSend(This, compileExpr(body, super, idecls), recur expr)
```

CICAE Helpers

```
let rec locate = fun what nameOf name vals ->
  match vals with
    [] -> raise (NoSuch (what, name))
  | a::rest ->
      if (name = nameOf(a))
      then 0
      else 1 + (locate what nameOf name rest)

let locateIField = (locate "field"
                       (fun (IField(name)) -> name))
let locateIMethod = (locate "method"
                        (fun (IMethod(name, _)) -> name))

let rec classFieldCount = function
    (cname, idecls) ->
      if (cname = "object")
      then 0
      else let IClass(_, sname, fields, _)
                = findIClass cname idecls
      in (List.length fields) + classFieldCount(sname, idecls)
```

CICAE Revised Compiler: Methods

```
let rec compileMethods = function
    (sdecl, idecls) ->
      let IClass(name, superName, fields, methods) = sdecl
      in Class(name,
               List.length fields,
               List.map
                 (fun (IMethod(name, expr)) ->
                   compileExpr(expr,
                               sdecl,
                               idecls))
                 methods)
```

CICAE Revised Compiler: Flattening

```
let rec flattenClassNames  : (cdecl * idecl list * cdecl list
                               -> cdecl * string list) = function
    (Class(name, fields, methods), idecls, cdecls) ->
      let IClass(_, superName, _, imethods) = findIClass name idecls
      in let (Class(_, superFields, superMethods), superMNames)
          = if (superName = "object")
          then (Class("object", 0, []), [])
          else flattenClassNames(findClass superName cdecls,
                                 idecls, cdecls)
      in let (methods, names)
          = addReplaceMethods(superMethods,
                              superMNames,
                              methods,
                              (List.map
                                 (fun (IMethod(name, _)) -> name)
                                 imethods))
      in (Class(name,
                superFields + fields,
                methods),
          names)

let flattenClass = function x ->
  let (c, names) = flattenClassNames x
  in c
```

CICAE Revised Compiler: Flattening - Methods

```
let rec addReplaceMethods : (cae list * string list
                              * cae list * string list
                            -> cae list * string list) = function
    (methods, names, [], []) -> (methods, names)
  | (methods, names, meth::mrest, name::nrest) ->
      let (methods, names) = addReplaceMethod(methods, names, meth, name)
      in addReplaceMethods(methods,
                           names,
                           mrest,
                           nrest)
  | _ -> raise (Failed "shouldn't happen")

and addReplaceMethod : (cae list * string list
                        * cae * string
                       -> cae list * string list) = function
    ([], [], bmeth, bname) -> ([bmeth], [bname])
  | (ameth::arest, aname::arestnames, bmeth, bname)
    -> if (aname = bname)
    then (bmeth::arest, bname::arestnames)
    else let (meths, names)
        = addReplaceMethod (arest, arestnames, bmeth, bname)
    in (ameth::meths, aname::names)
  | _ -> raise (Failed "shouldn't happen")
```

TICAE Revised Type Checker

```
let rec typecheckExpr = function
    (expr, tdecls, argTy, thisClass) ->
      let recur = fun expr ->
                    typecheckExpr(expr, tdecls, argTy, thisClass)
      in match expr with
        …
      | IGet(expr, getCName, fname) ->
          (match (recur expr) with
            ObjT(cname) ->
              if not (isSubClass(cname, getCName, tdecls))
              then raise (NoType(expr, "field class mismatch"))
              else …
            …
      | ISend(expr, sendCName, mname, argExpr) ->
          (match (recur expr) with
            ObjT(cname) ->
              if not (isSubClass(cname, sendCName, tdecls))
              then raise (NoType(expr, "method class mismatch"))
              else …
            …
        …
```