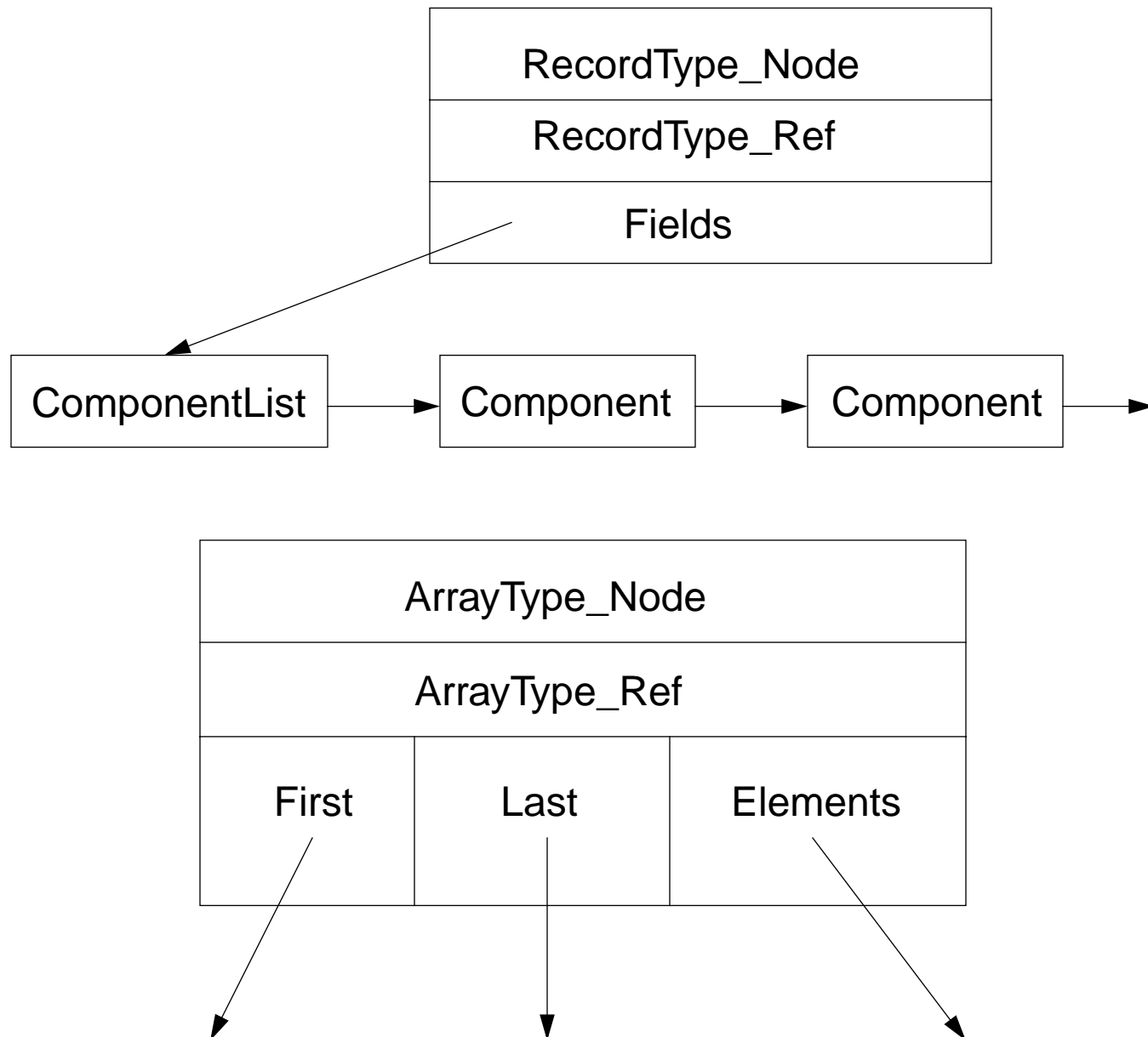


# AST Nodes for Arrays and Records

---



# Processing Record Declarations

---

RecordType\_Node.TypeSemantics( )

    Create a RecordTypeDescriptor for the new record type with:

        Size initialized to 0

        Fields initialized with a newly created symbol table

    Set RecordType\_Ref to point to this RecordTypeDescriptor

    For each of the components referenced by Fields,

        Declare each of the Ids as fields of this record by putting it in the symbol table RecordType\_Ref.Fields

        Associate a FieldAttributes record with each Id with it indicating:

            IdType = the type included in the component declaration

            Offset = RecordType\_Ref.Size

        Allocate space for the field by adding IdType.Size to RecordType\_Ref.Size

*Note 1: The pseudocode above allocates offsets for the fields within the record as they are declared. This could be deferred until the code generation pass.*

*Note 2: The result of this processing is a type descriptor referenced from the tree, NOT an attributes record for a type name (since there is not necessarily one). The Attributes records created are in the symbol table that describes the record type itself.*

# Processing Array Declarations

---

ArrayType\_Node.TypeSemantics( )

  If First and Last describe a valid range then

    Create an ArrayTypeDescriptor for the new type with:

      Lower = value of First

      Upper = value of Last

      ElementType = Elements.TypeSemantics()

      Size = ElementType.Size \* (Upper–Lower+1)

    Set ArrayType\_Ref to point to this TypeDescriptor

  else

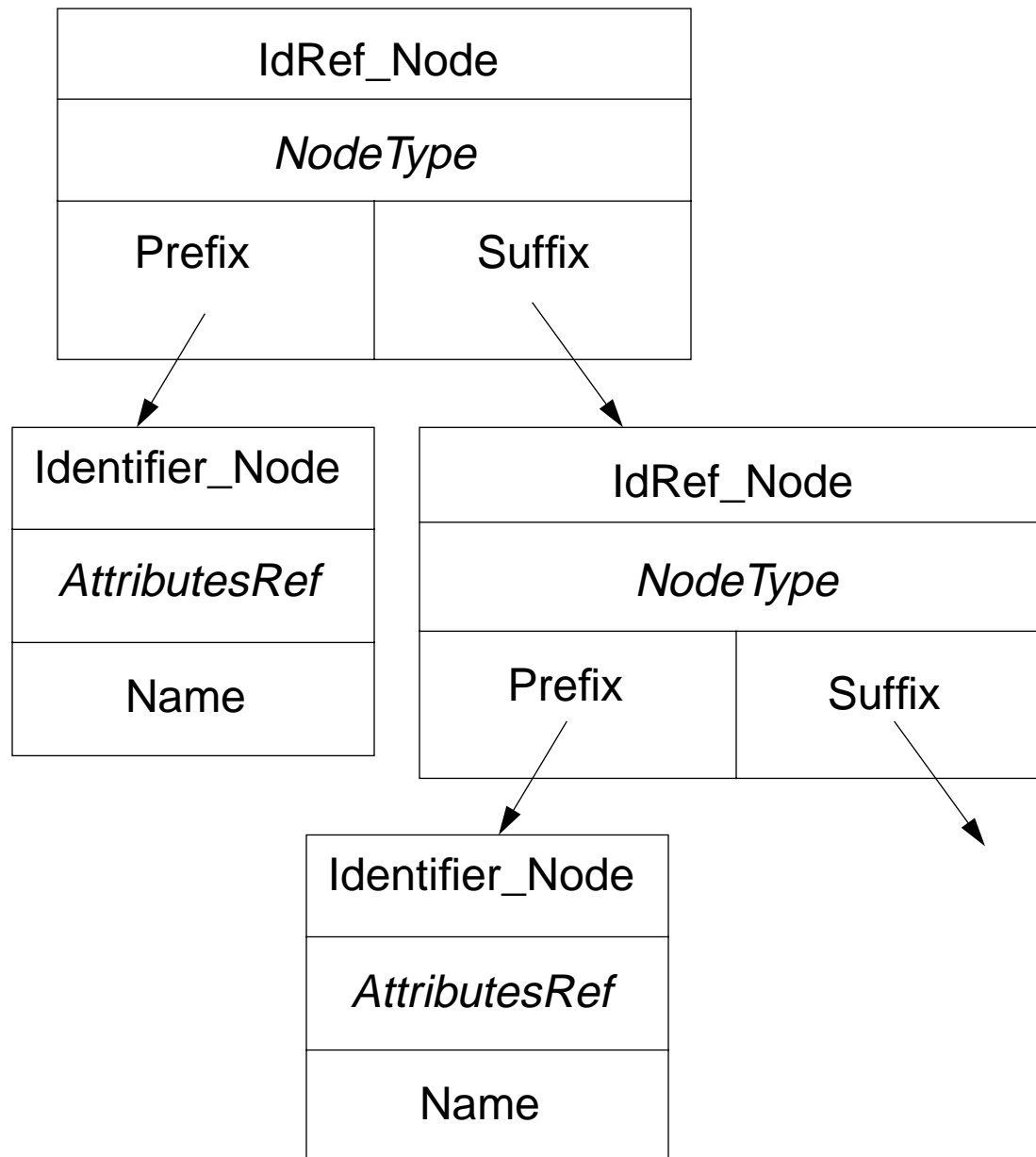
    Produce any appropriate error message

    Set ArrayType\_Ref to point to an ErrorType descriptor

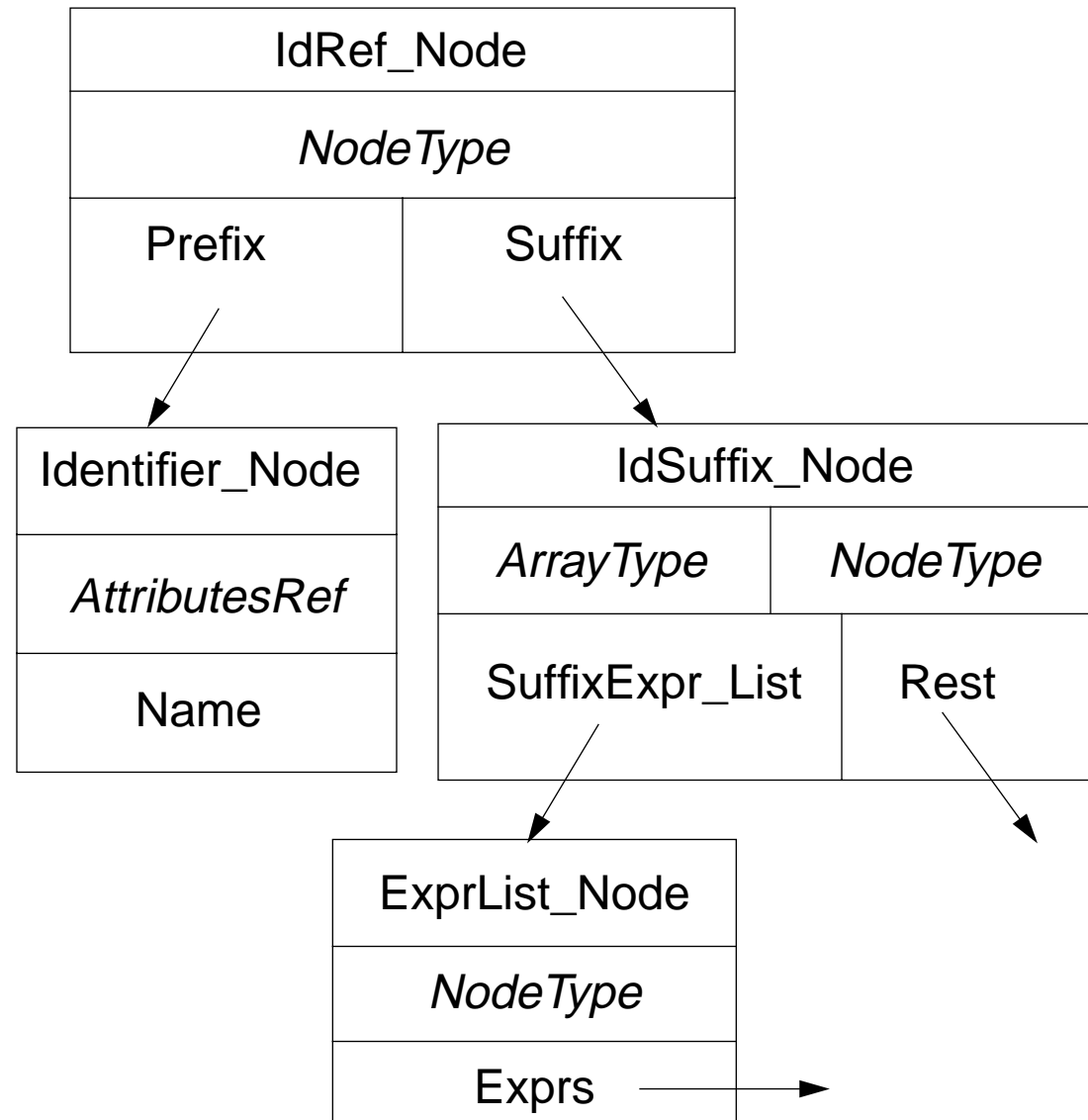
*Note: The pseudocode above computes the size of the array type as the type descriptor is created. This step could be deferred until the code generation pass.*

# AST for Record References

---



# AST for Array References



# Semantics for Record References

---

IdRef\_Node.ExprSemantics()

Prefix.Semantics()

If Suffix = NULL then

Set NodeType from Prefix.NodeType

else

Suffix.IdSemantics(Prefix.NodeType)

Set NodeType from Suffix.NodeType

IdRef\_Node.IdSemantics(LeftType)

if LeftType is not a record type then

error (Name before a “.” is does not denote a record)

Set NodeType to ErrorType

else

Prefix.FieldSemantics(LeftType.Fields)

If Suffix = NULL then

Set NodeType from Prefix.NodeType

else

Suffix.IdSemantics(Prefix.NodeType)

Set NodeType from Suffix.NodeType

*Identifier\_Node.FieldSemantics is left for you to supply.*

# Semantics for Array References

---

IdSuffix\_Node.IdSemantics(LeftType)

Set ArrayType to the value of LeftType

SuffixExpr\_List.ArraySemantics(LeftType)

If Rest = NULL then

Set NodeType from SuffixExpr\_List.NodeType

else

Rest.IdSemantics(SuffixExpr\_List.NodeType)

Set NodeType from Rest.NodeType

ExprList\_Node.ArraySemantics(LeftType)

[Local variable: LocalType]

Initialize LocalType to LeftType

For each ExprNode, E, on the list

If LocalType does not denote an array type then

error (Name before an index expression does not denote an array)

Set LocalType to ErrorType

exit for loop

else

E.ExprSemantics()

Check that E.NodeType is IntegerType

Set LocalType to LocalType.ElementType

Set NodeType to LocalType

## Code Generation for Record References

---

IdRef\_Node.ExprCodeGen()

Set Addr to the address returned by Prefix.ExprCodeGen()

If Suffix = NULL then

Result  $\leftarrow$  Addr

else

Result  $\leftarrow$  Suffix.IdCodeGen(Addr)

IdRef\_Node.IdCodeGen(LeftAddr)

Set Addr to the address returned by Prefix.IdCodeGen(LeftAddr)

If Suffix = NULL then

Result  $\leftarrow$  Addr

else

Result  $\leftarrow$  Suffix.IdCodeGen(Addr)

Identifier\_Node.IdCodeGen(LeftAddr)

Let Offset be the offset for the field specified by this node

If LeftAddr is a variable address then

Result  $\leftarrow$  LeftAddr with Offset added to its Var\_Offset field

else [LeftAddr must be a register address]

Generate code to add Offset to the value in the register denoted by LeftAddr

Result  $\leftarrow$  LeftAddr

## Code Generation for Array References

---

IdSuffix\_Node.IdCodeGen(LeftAddr)

Set Addr to the address returned by

SuffixExpr\_List.ArrayCodeGen(LeftAddr)

If Rest = NULL then

Result  $\leftarrow$  Addr

else

Result  $\leftarrow$  Rest.IdCodeGen(Addr)

ExprList\_Node.ArrayCodeGen(LeftAddr)

[Local variables: LocalAddr LocalType]

Initialize LocalAddr to LeftAddr

Initialize LocalType to ArrayType

For each ExprNode, E, on the list

Set IndexAddr to the address returned by E.ExprCodeGen()

Generate code to test that the value in IndexAddr is within the bounds defined in LocalType

Generate code to add (Index-LocalType.First)\*LocalType.Size to LocalAddr updating LocalAddr as necessary

*LocalAddr is now an indirect address and must be so marked.*

Set LocalType to LocalType.ElementType

Result  $\leftarrow$  LocalAddr