Type Linking

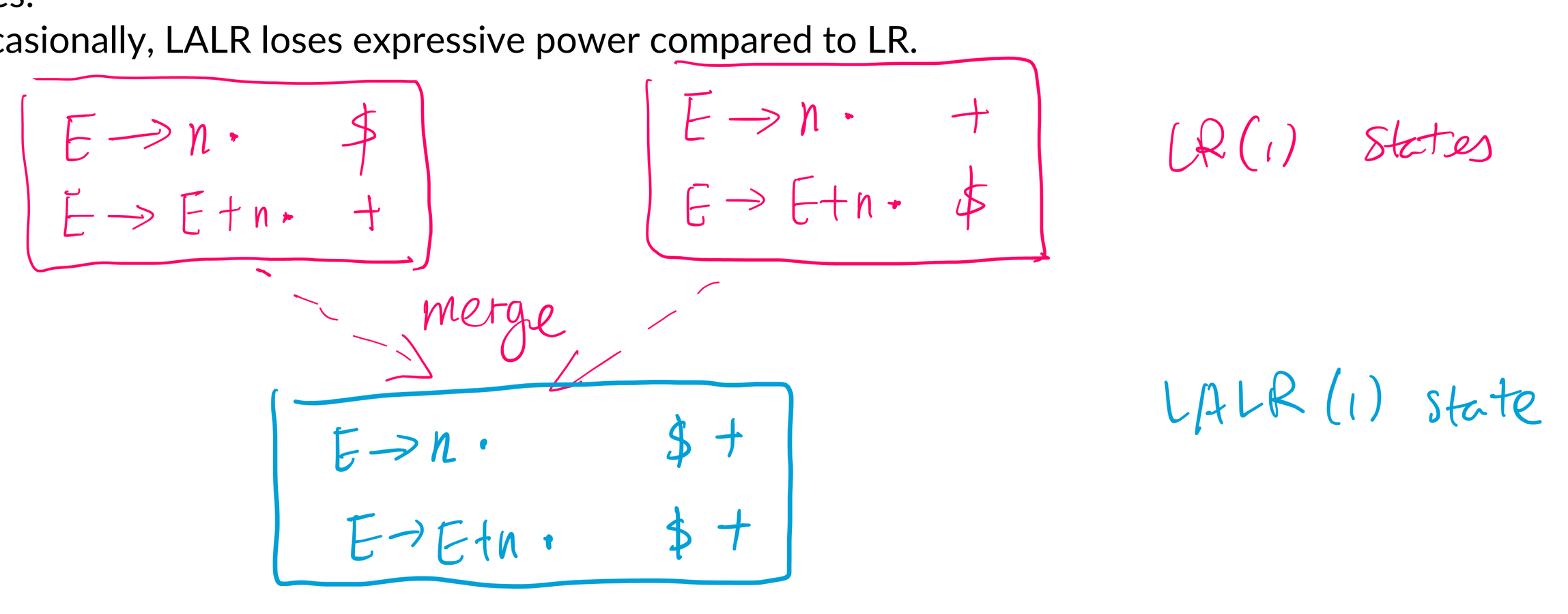
LALR(1)

LR(1): # automaton states can be large
end up having many states that are identical other than lookahead tokens

$[X \to \beta \cdot \gamma]$

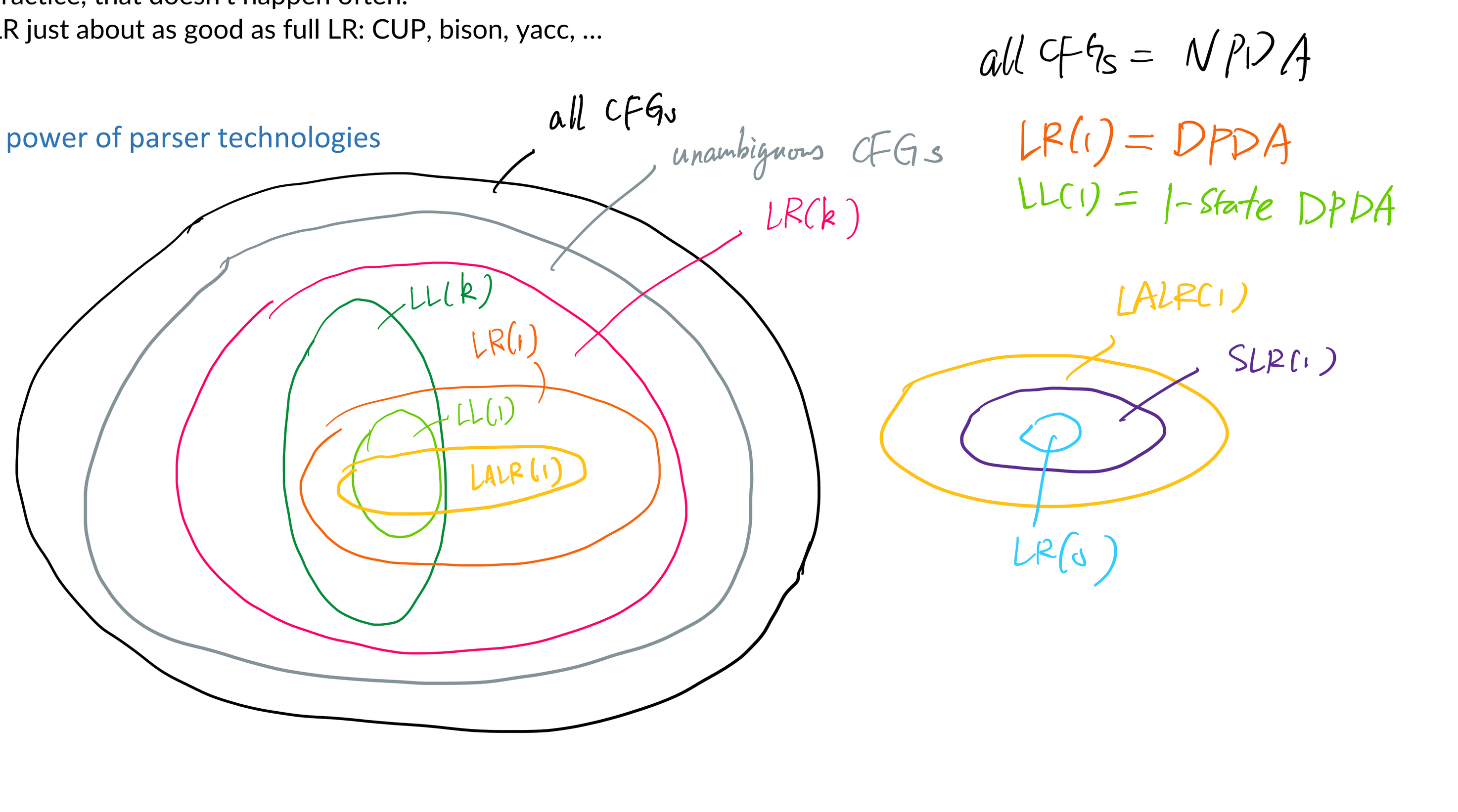
LALR(1): merge LR(1) states with same LR(0) items (lookahead sets merged too)
Actual implementation:
don't have to do merge after the fact; identify automaton states by LR(0) items

Outcomes:
- Occasionally, LALR loses expressive power compared to LR.



- In practice, that doesn't happen often.
  LALR just about as good as full LR: CUP, bison, yacc, ...

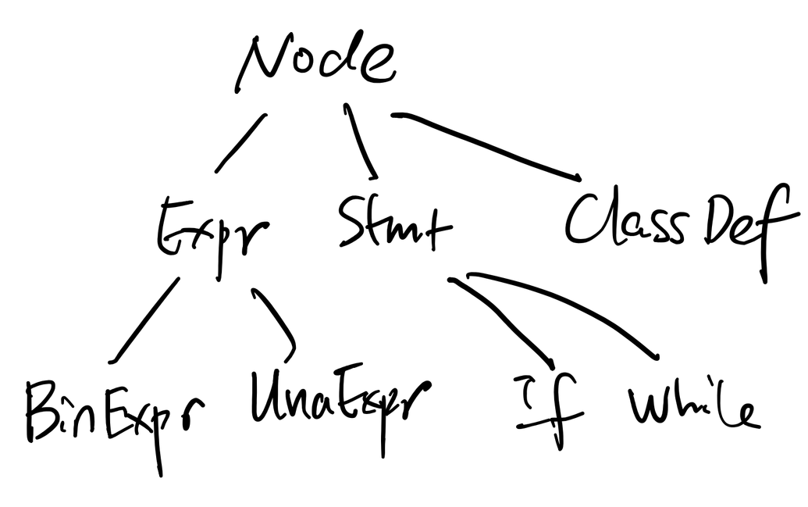
Relative power of parser technologies



AST design

Basic principle: you don't want to fight your language!

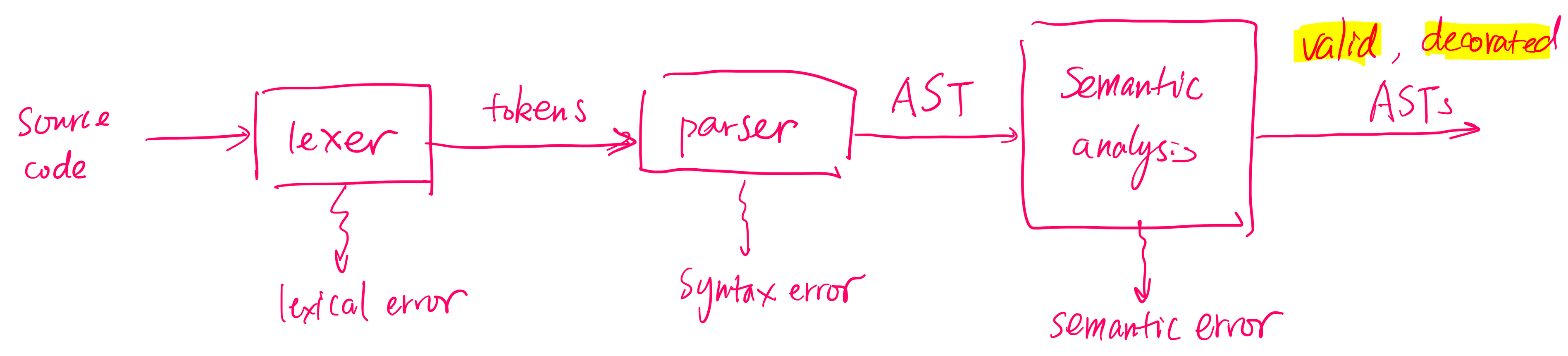
OOP. class hierarchy, method dispatch.



FP. Variant types, pattern matching.

type expr := BinExp of expr * expr
| UnaryExp of expr
| ...

type stmt := If of expr * stmt * stmt
| While of expr * stmt
| ...



Semantic Analysis

"valid" AST

"decorated" AST

Name resolution — resolve uses (aka references) to declarations

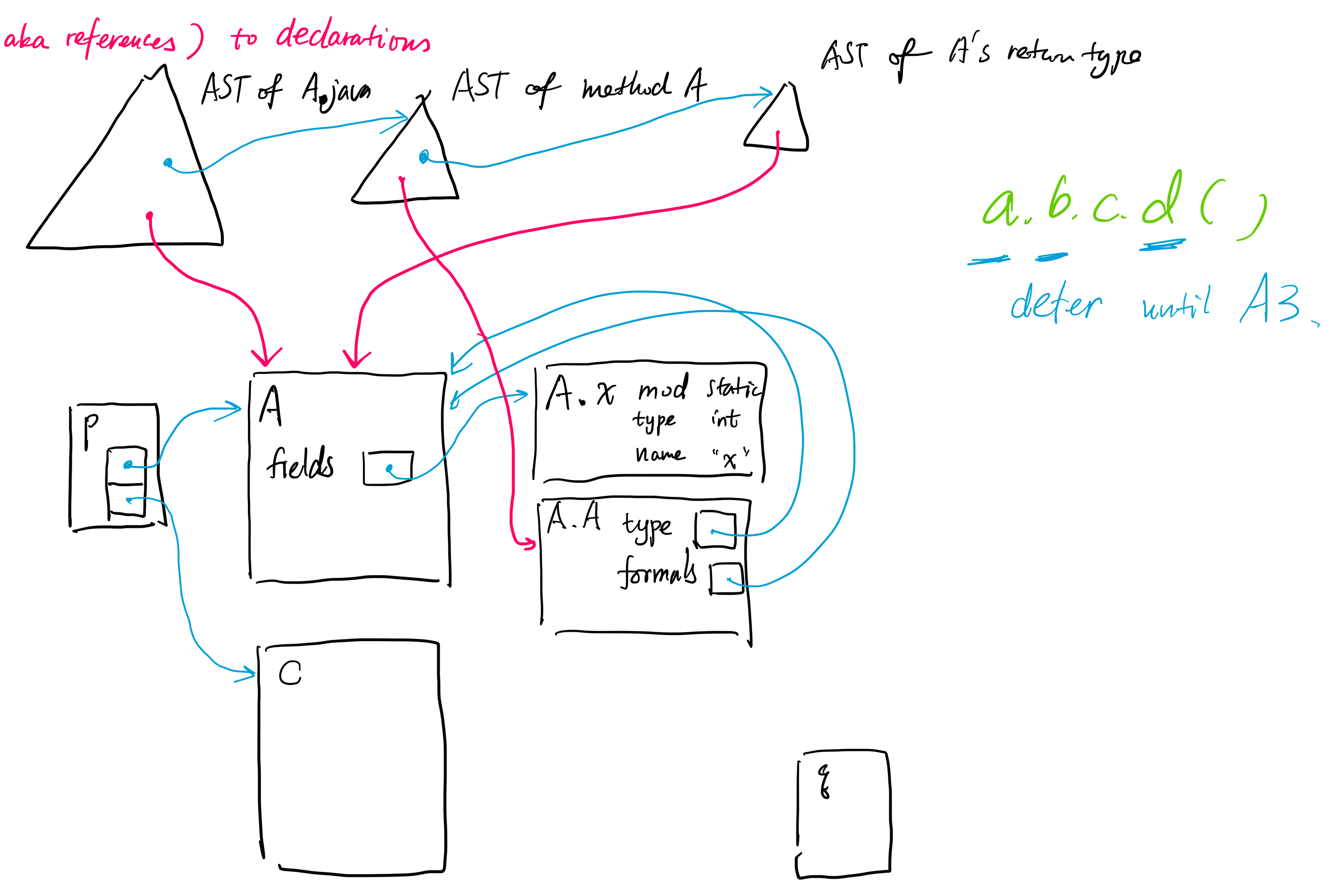
```
package p;
class A {
    static int x;
    A A(A A) {
        return (A)A.A(A);
    }

}

package p;
import q.*;
class C {
    D y;
    int z = A.x;
    … y.w.z …
}

package q;
class D extends C { C w; }

package q;
class C { String z; }
```



How to represent declarations?

Option 1: Decls as ASTs
- ASTs not always available
- error-prone
- Inefficient.

Option 2: Decls as package objects, class objects, method objects, field objects, local var objects, etc.

A2 first step: Build class environment

environment: aka contexts, symbol tables,
a map from names (strings) to declarations.

class environment aka class table

A2 second step: Resolving uses of class names (aka type linking)

class A { B x; } class B { A y; }

Combine construction of class table and type linking in a single AST traversal?

A2 third step: Sanity-check class hierarchies, field names, etc.

(copied from A2 specification)
1. A class must not extend an interface. (JLS 8.1.3)
2. A class must not implement a class. (JLS 8.1.4)
3. An interface must not be repeated in an implements clause or in an extends clause of an interface. (JLS 8.1.4)
4. A class must not extend a final class. (JLS 8.1.1.2, 8.1.3)
5. An interface must not extend a class. (JLS 9.1.2)
6. The hierarchy must be acyclic. (JLS 8.1.3, 9.1.2)
7. A class or interface must not declare two methods with the same signature (name and parameter types). (JLS 8.4, 9.4)
8. A class must not declare two constructors with the same parameter types (JLS 8.8.2)
9. A class or interface must not contain (declare or inherit) two methods with the same signature but different return types (JLS 8.1.1.1, 8.4, 8.4.2, 8.4.6.3, 8.4.6.4, 9.2, 9.4.1)
10. A class that contains (declares or inherits) any abstract methods must be abstract. (JLS 8.1.1.1)
11. A nonstatic method must not replace a static method. (JLS 8.4.6.1)
12. A static method must not replace a nonstatic method. (JLS 8.4.6.2)
13. A method must not replace a method with a different return type. (JLS 8.1.1.1, 8.4, 8.4.2, 8.4.6.3, 8.4.6.4, 9.2, 9.4.1)
    $\forall (m, m') \in Replace,\ type(m) = type(m')$
14. A protected method must not replace a public method. (JLS 8.4.6.3)
15. A method must not replace a final method. (JLS 8.4.3.3)

Conditions above are expressed informally, in English. How to turn it to code?

class A extends B implements C, D, E. $super(A) := \{B, C, D, E\}$

if implicit, B = java.lang.Object $super(java.lang.Object) = \emptyset$

interface F extends G, H $super(F) := \{G, H\}$

$Declare(T) :=$ methods and fields declared in T

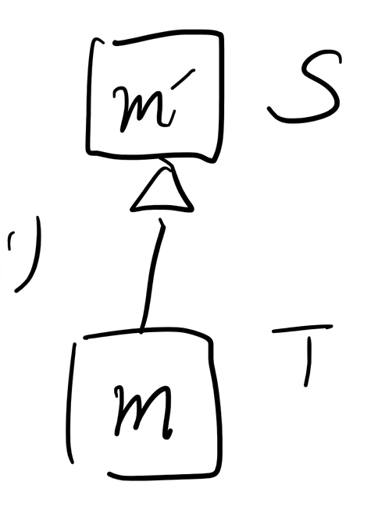
for $m \in Declare(T)$: $mod(m)$, $name(m)$, $sig(m)$: arg types + name, $type(m)$: return type

for $f \in Declare(T)$: $name(f)$, $type(f)$

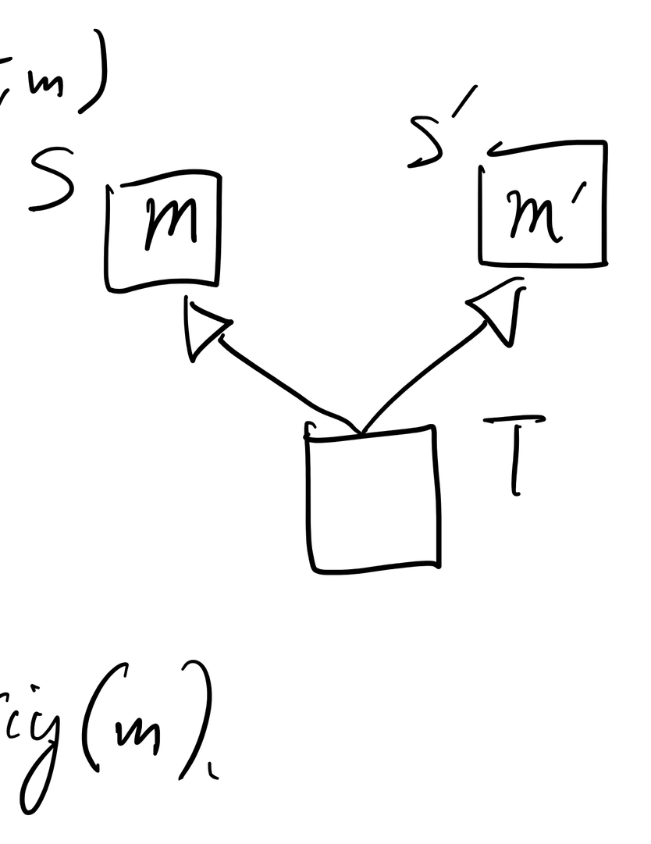
$Contain(T) := Declare(T) \cup Inherit(T)$

Replace(m, m')

$$\frac{S \in super(T) \quad m' \in contain(S) \quad m \in Declare(T) \quad sig(m) = sig(m')}{Replace(m, m')}$$



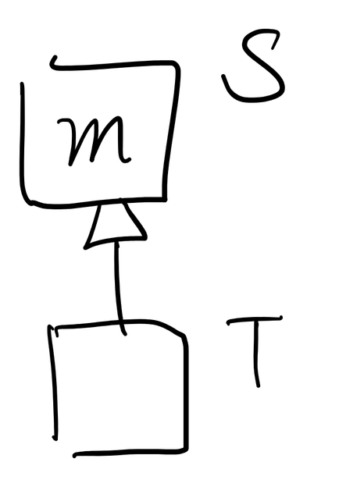
$$\frac{\begin{array}{l} S \in Super(T) \quad m \in Contain(S) \quad sig(m) = sig(m') \quad NoDecl(T, m) \\ S' \in super(T) \quad m' \in Contain(S') \quad abstract \in mod(m') \\ \qquad abstract \notin mod(m) \end{array}}{Replace(m, m')}$$



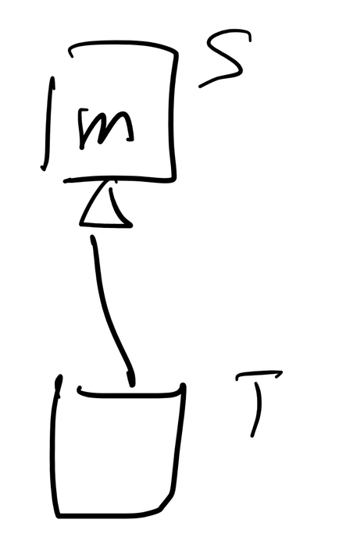
$$NoDecl(T, m) := \forall m' \in Declare(T),\ sig(m') \neq sig(m).$$

Inherit(T) := methods and fields inherited into T.

$$\frac{S \in super(T) \quad m \in Contain(S) \quad NoDecl(T, m) \quad abstract \notin mod(m)}{m \in Inherit(T)}$$



$$\frac{\begin{array}{l} S \in super(T) \quad m \in Contain(S) \quad NoDecl(T, m) \\ \forall S' \in super(T),\ \forall m' \in Contain(S'),\ sig(m') = sig(m) \Rightarrow abstract \in mod(m') \end{array}}{m \in Inherit(T)}$$



$$\frac{S \in super(T) \quad f \in Contain(S) \quad \forall f' \in Declare(T),\ name(f') \neq name(f)}{f \in Inherit(T)}$$