Code Generation for High-Assurance Java Card Applets

Alessandro Coglio

Kestrel Institute

HCSS Conference
April 1-3, 2003

Summary

- Java Card
- Kestrel's work
- Applet generator
- Code generator
  - approach
  - details
  - checking correctness
  - accomplishments
- Current and future work

Summary

- Java Card
- Kestrel's work
- Applet generator
- Code generator
  - approach
  - details
  - checking correctness
  - accomplishments
- Current and future work

What Is Java Card?

What Is Java Card?

A version of Java for smart cards

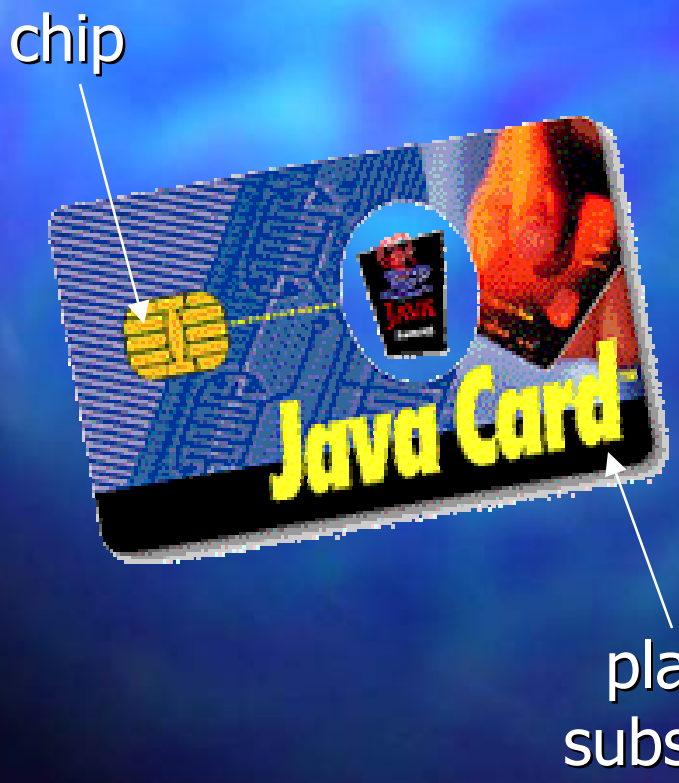
What Is Java Card?

A version of Java for smart cards

plastic
substrate

What Is Java Card?

A version of Java for smart cards



What Is Java Card?

A version of Java for smart cards

chip

plastic substrate

authentication,
banking,
telephony,
health care,
…

Java Card Technology

Java Card Technology

Java Card program

Java Card Technology

Java Card program

1) subset of Java

Java Card Technology

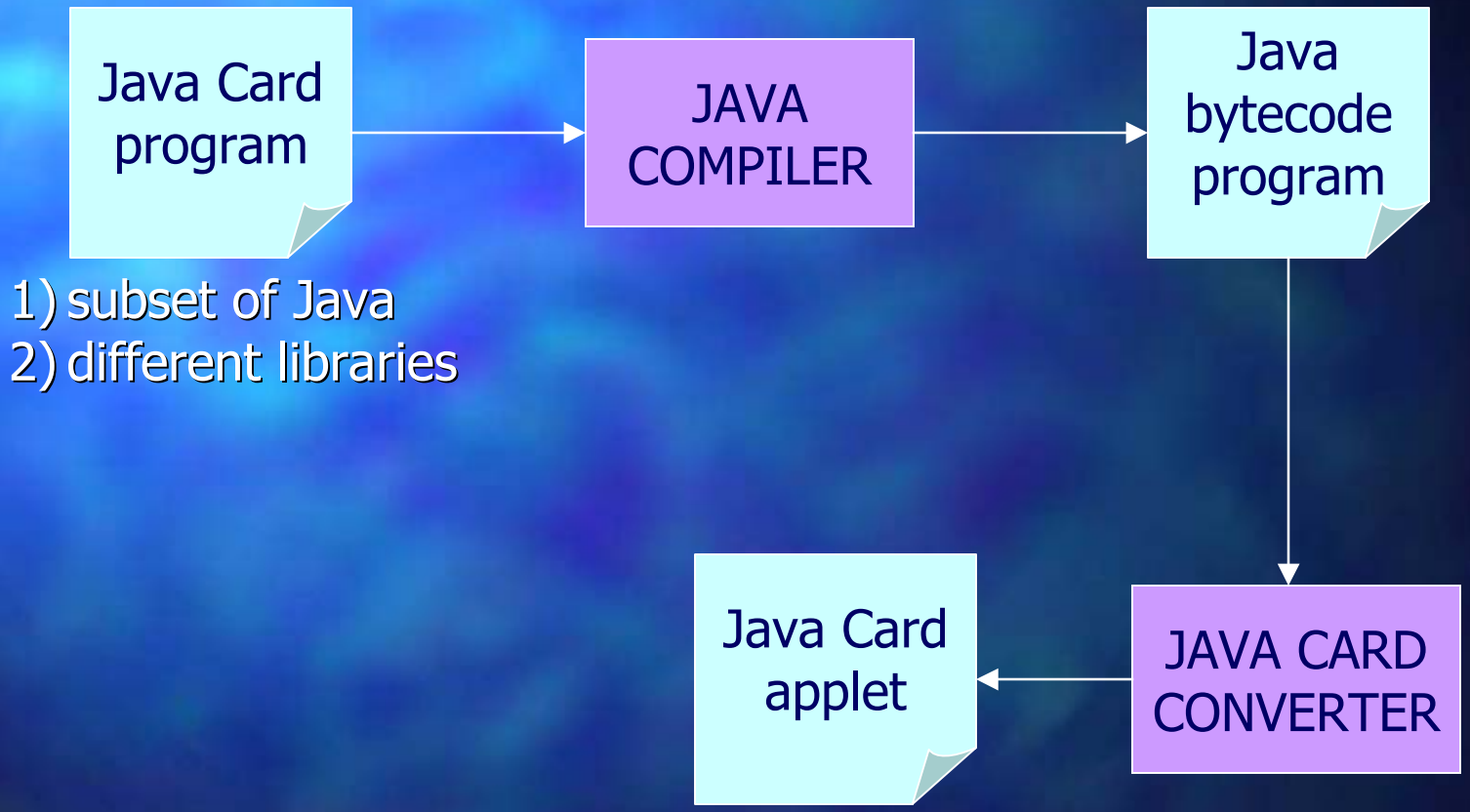
Java Card program

1) subset of Java
2) different libraries

Java Card Technology

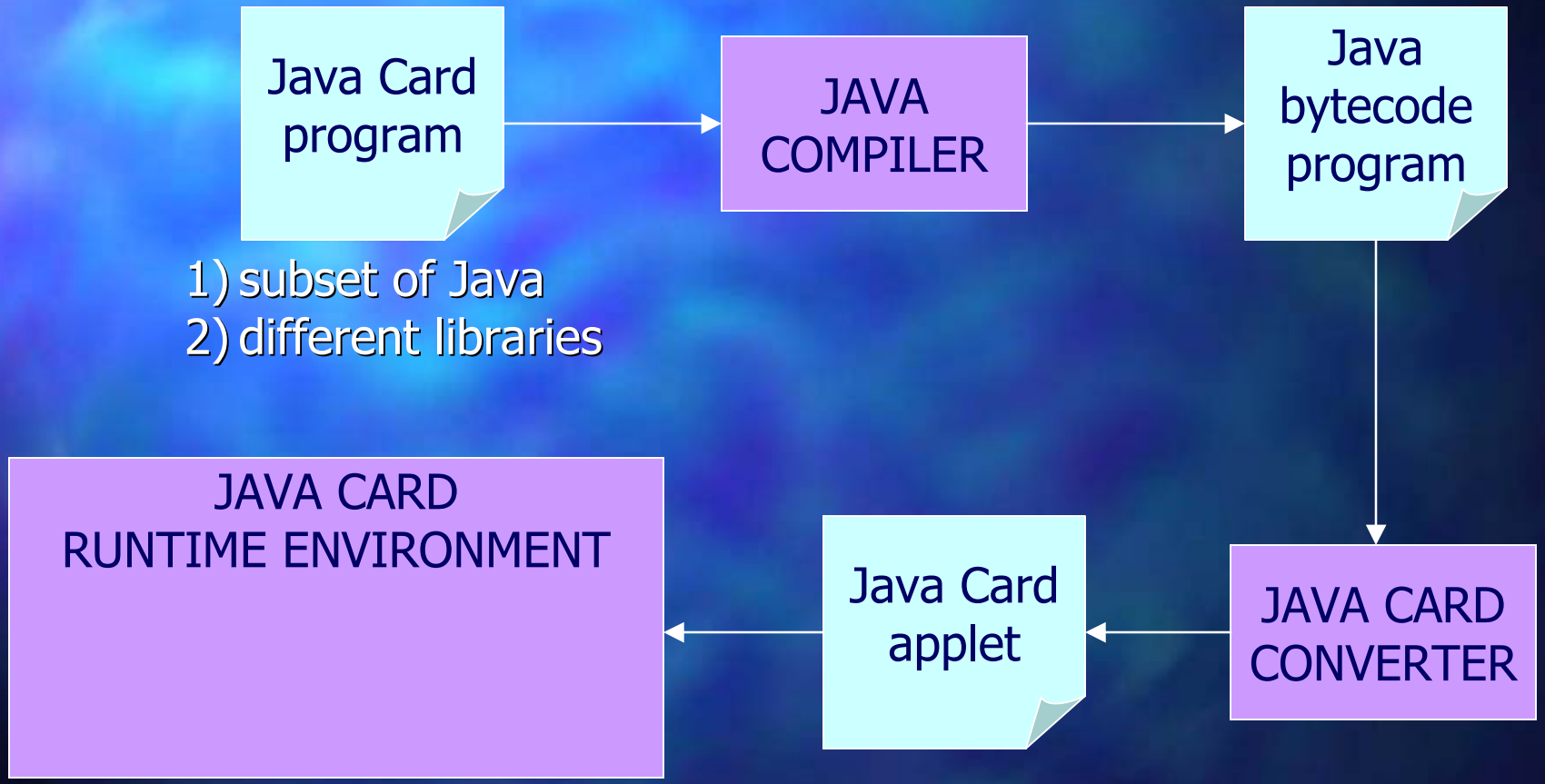
1) subset of Java
2) different libraries

Java Card Technology

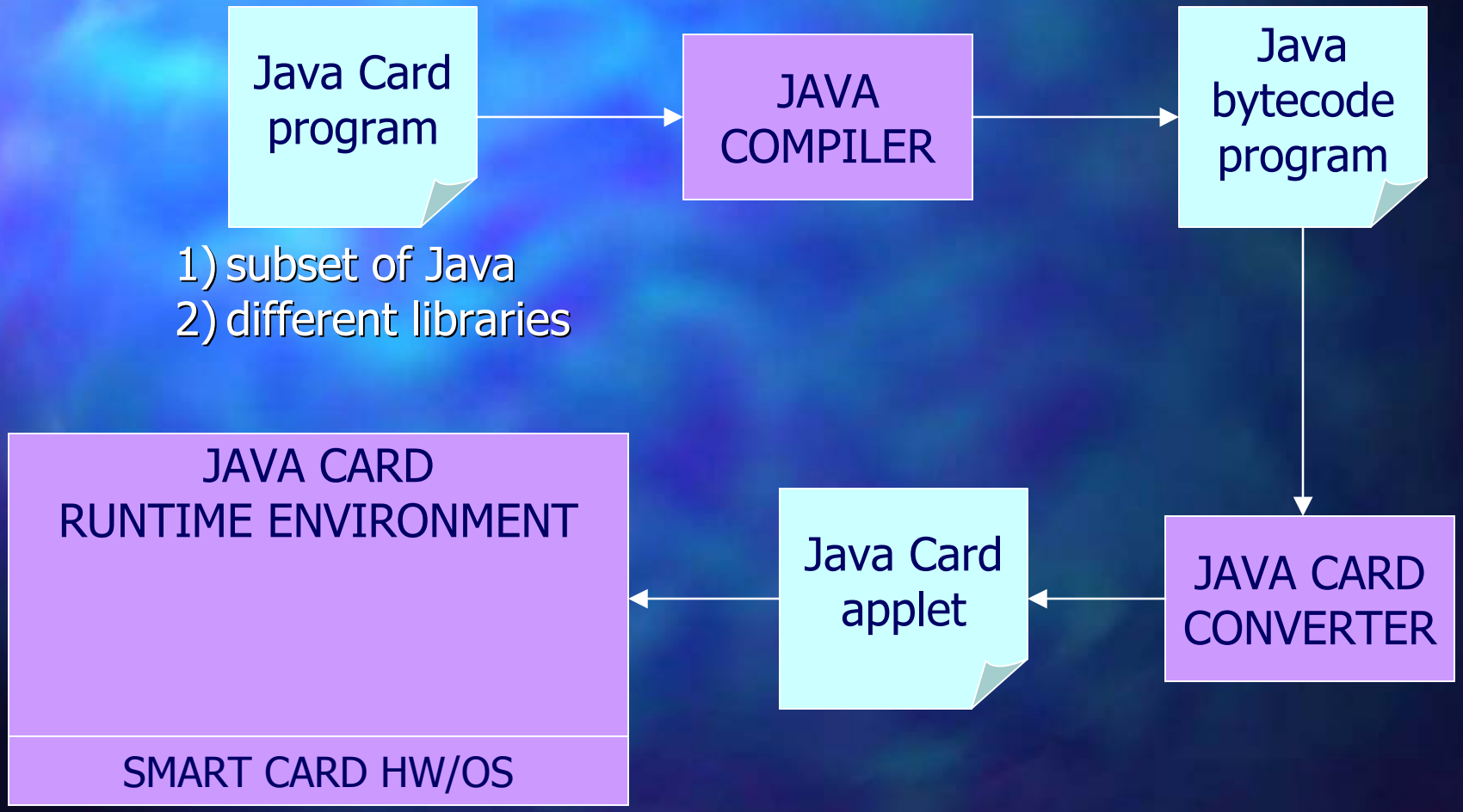


1) subset of Java
2) different libraries

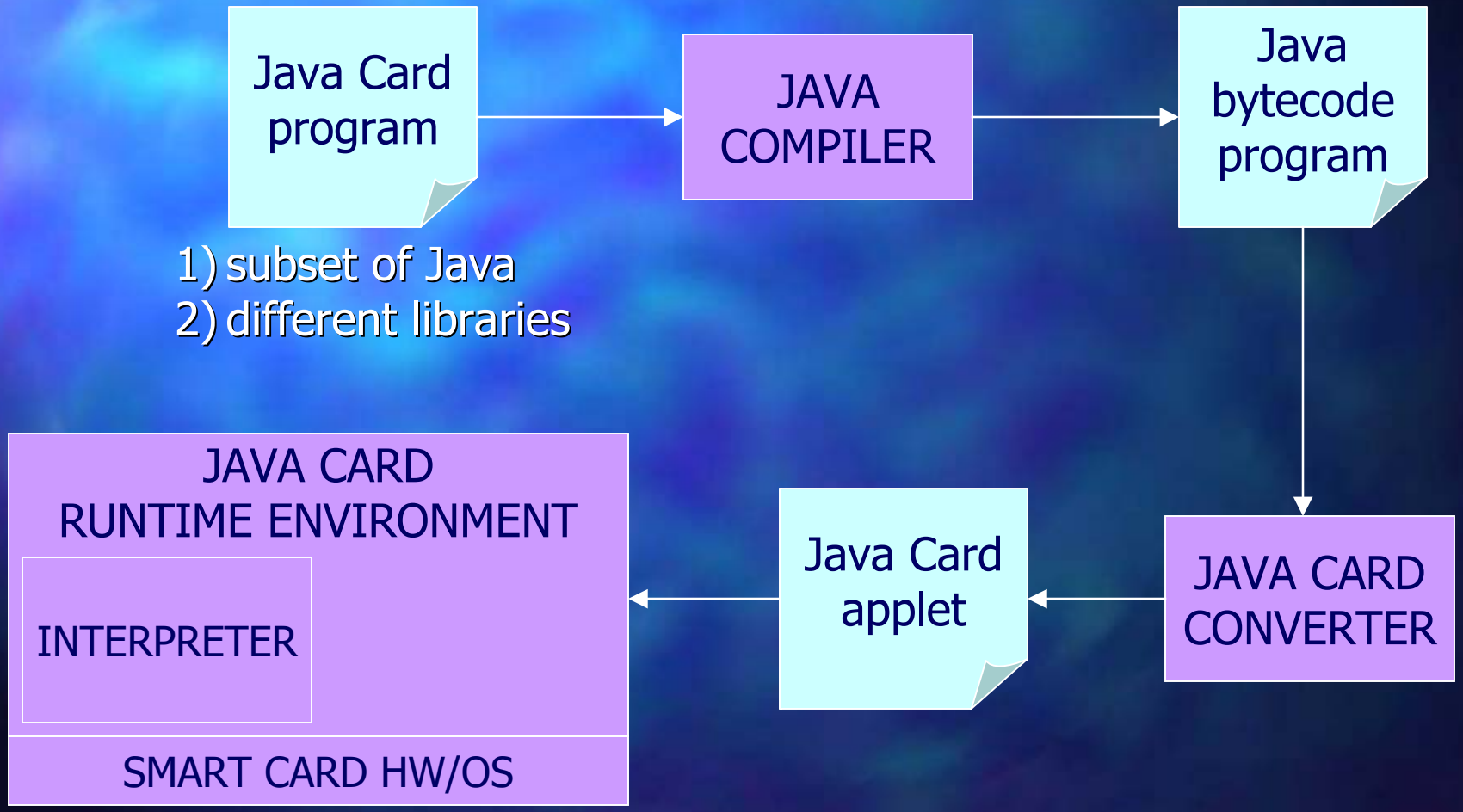
Java Card Technology



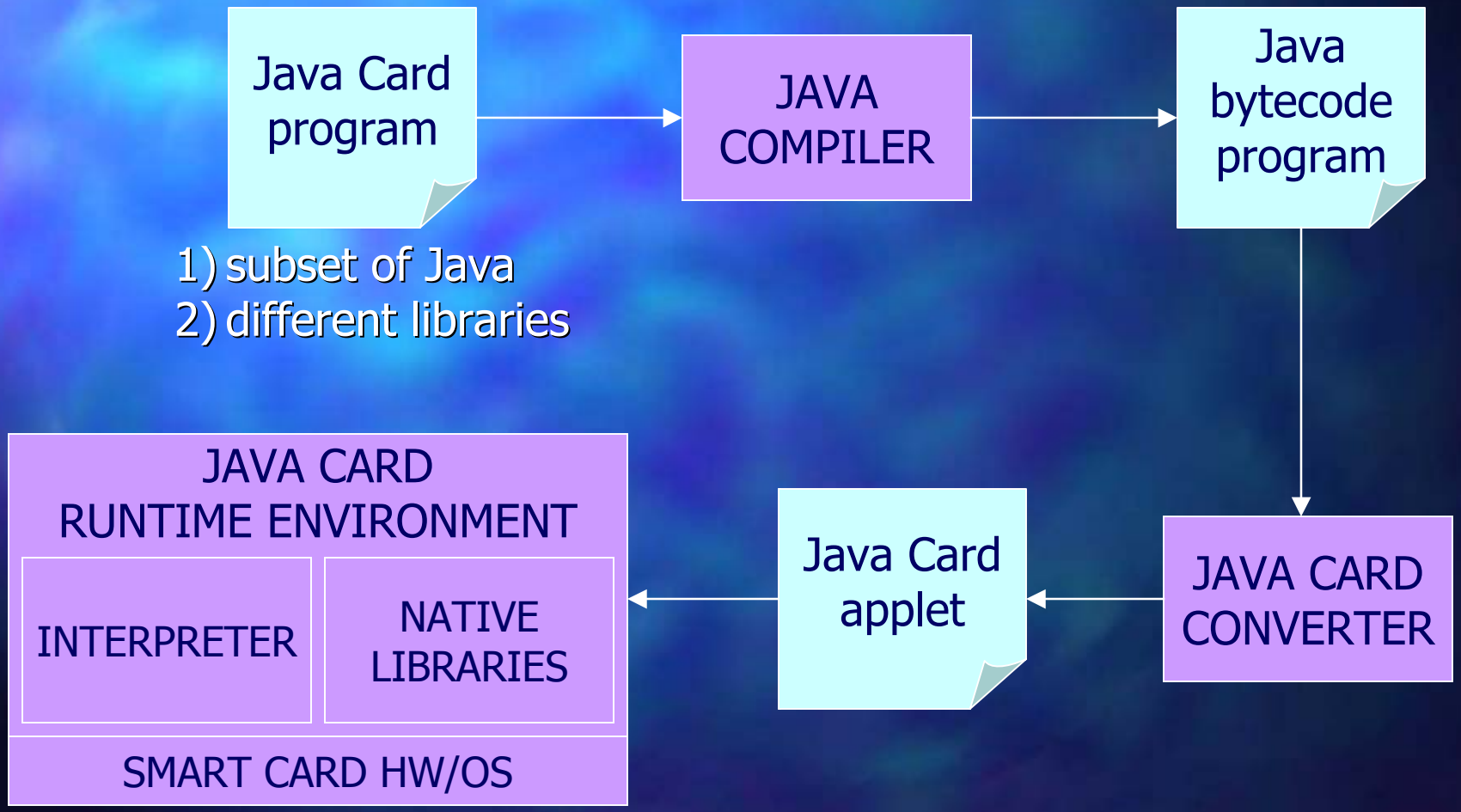
Java Card Technology



Java Card Technology



Java Card Technology



High Assurance

- Critical requirement for smart cards
- Pursued by smart card vendors
- Measurable (Common Criteria, FIPS-140)
- Focus of Kestrel Institute's research
  - automated synthesis ("specs to code")
  - formal analysis

Summary

- Java Card
- Kestrel's work
- Applet generator
- Code generator
  - approach
  - details
  - checking correctness
  - accomplishments
- Current and future work

Kestrel's Synthesis Systems

- Specware
  - formal specs
  - refinement
  - composition
  - code generation
- Designware
  - libraries of specs and refinements embodying software design knowledge (algorithms, optimizations, …)
  - tactics for automated refinement in Specware
- Planware
  - automatic generator of high-performance, complex resource systems (allocation, transportation schedulers, …)
  - on top of Specware
- MoBIES, HARBINGER, SVA, protocols, …

Our Work on Java Card

- Platform
  - synthesis (specs to code) of
    - Java Card Runtime Environment
    - simulator
    - possibly tools (off-card verifier, ...)
- Applets
  - applet generator

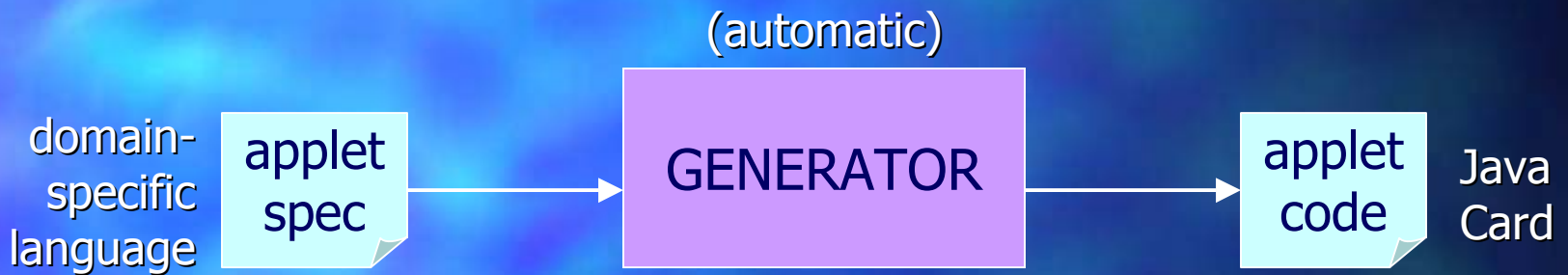
Our Work on Java Card

- Platform
  - synthesis (specs to code) of
    - Java Card Runtime Environment
    - simulator
    - possibly tools (off-card verifier, ...)
- Applets
  - applet generator

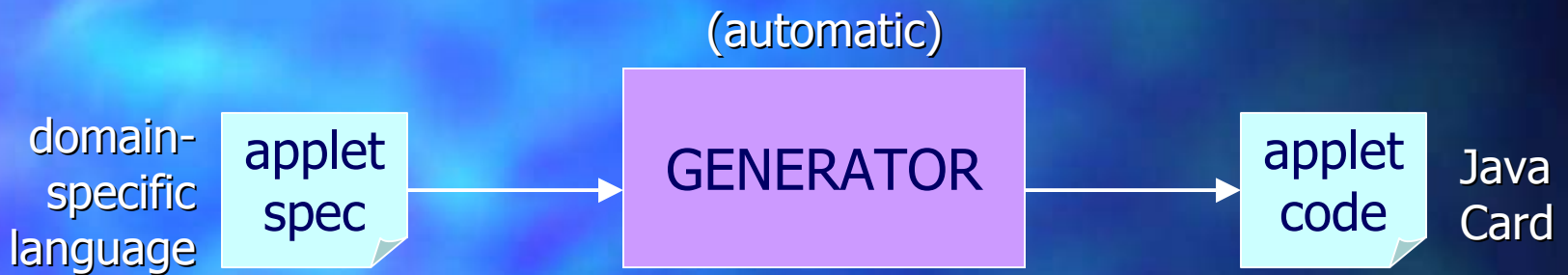
Summary

- Java Card
- Kestrel's work
- Applet generator
- Code generator
  - approach
  - details
  - checking correctness
  - accomplishments
- Current and future work

Applet Generator

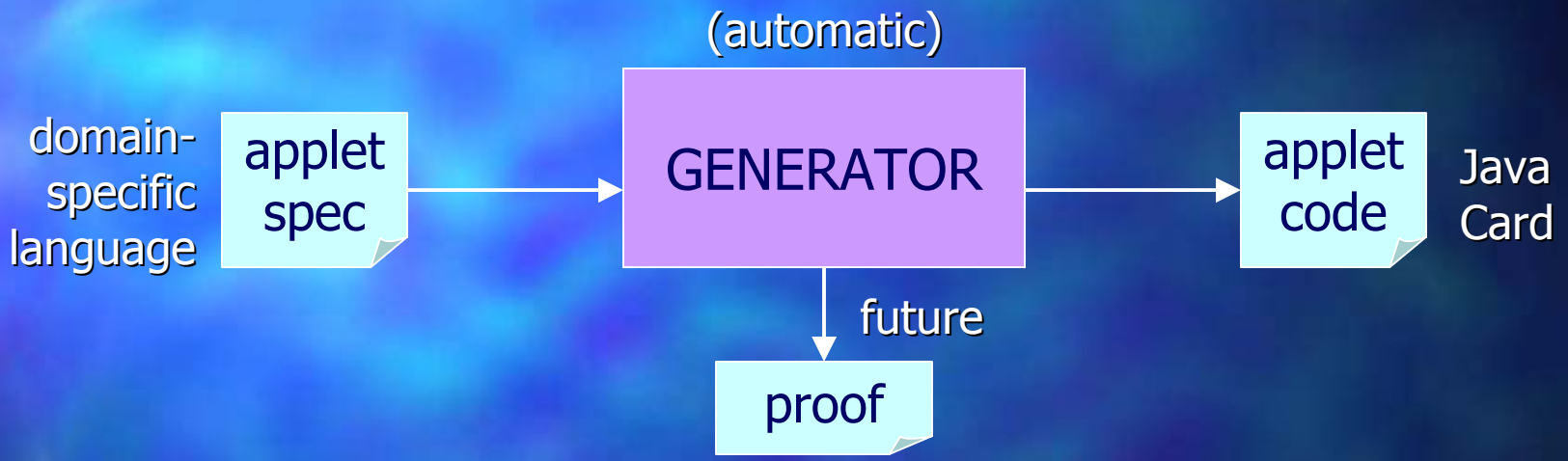


Applet Generator



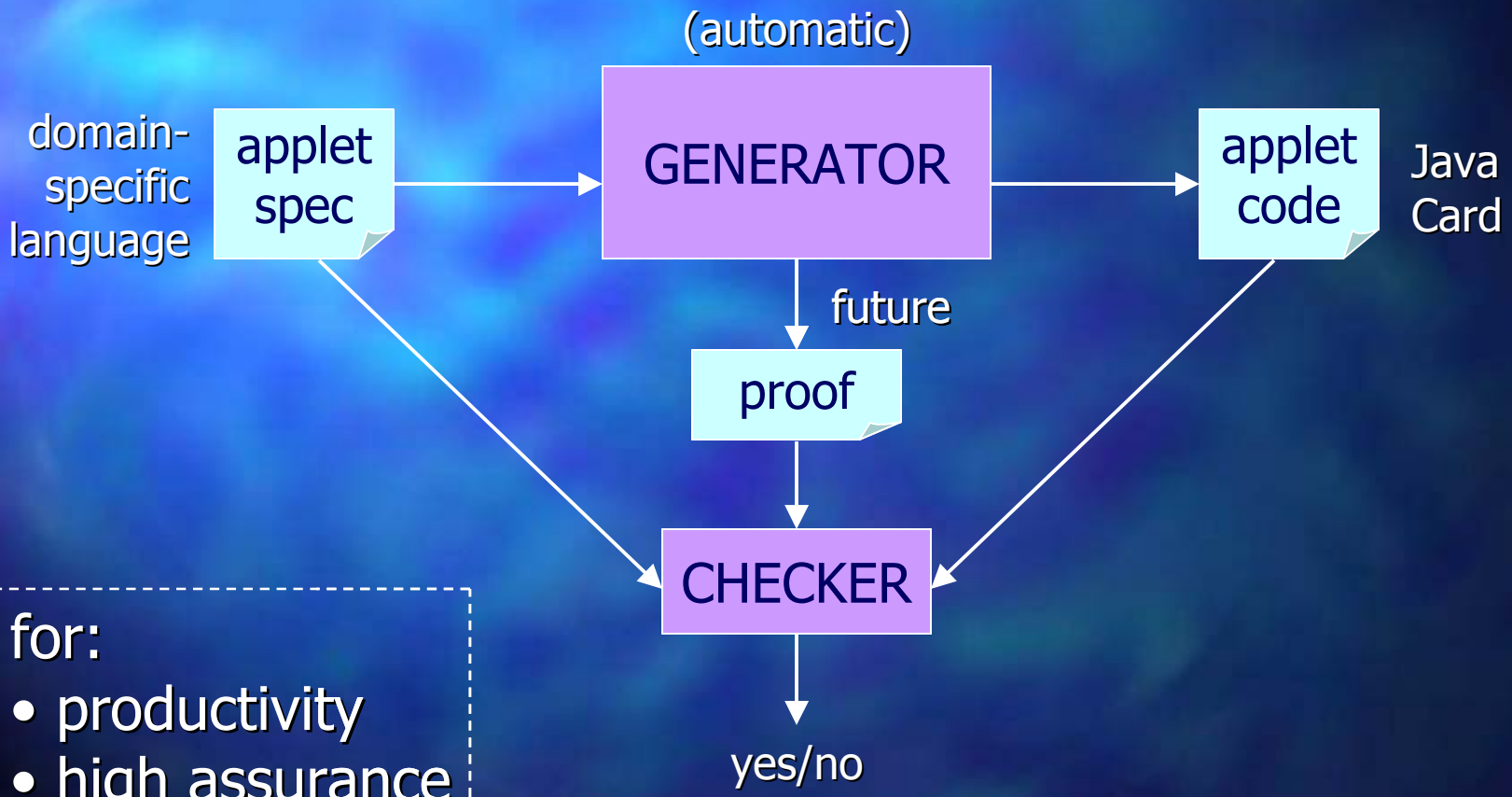
for:
- productivity
- high assurance

Applet Generator



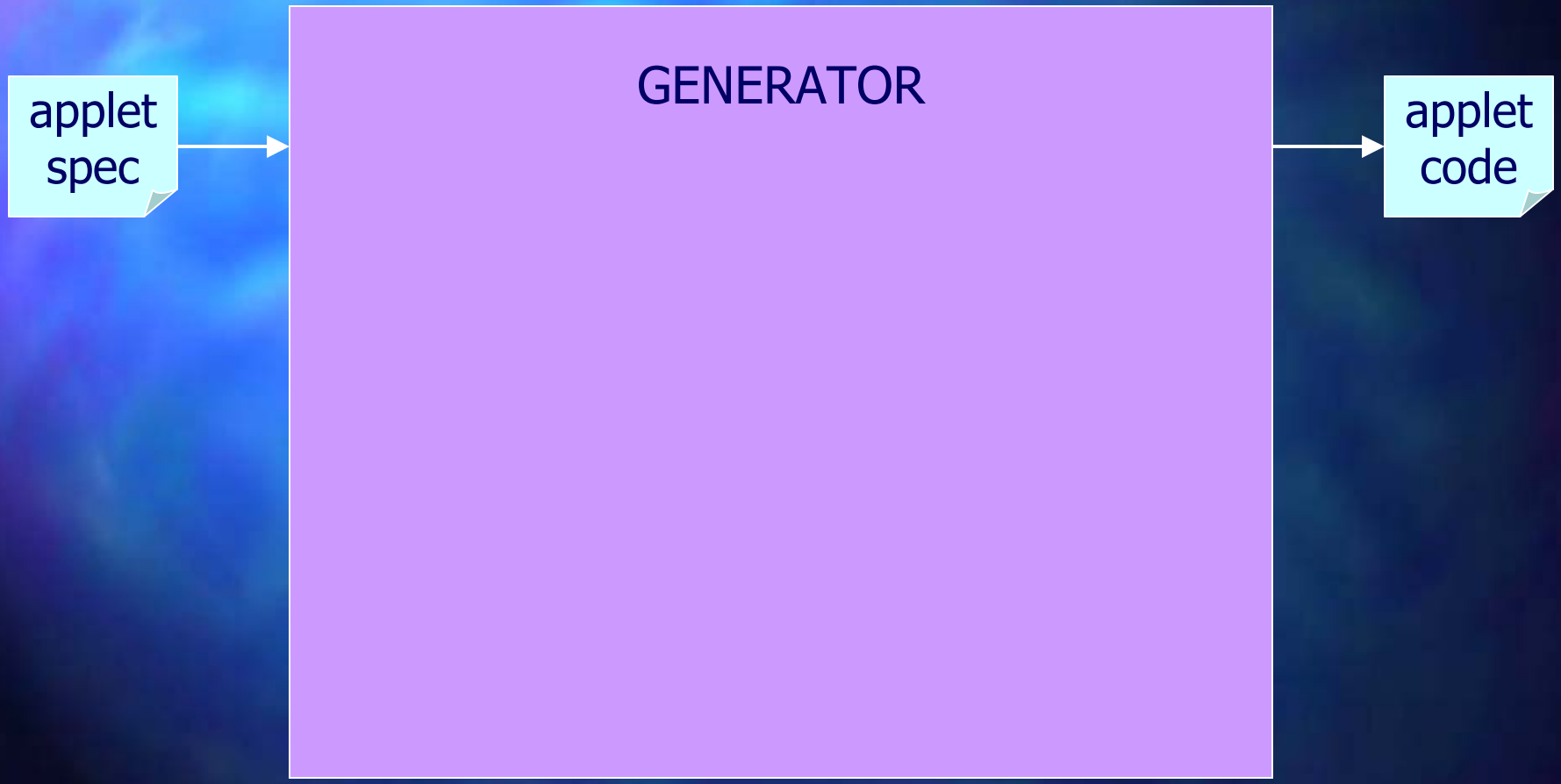
for:
- productivity
- high assurance

Applet Generator

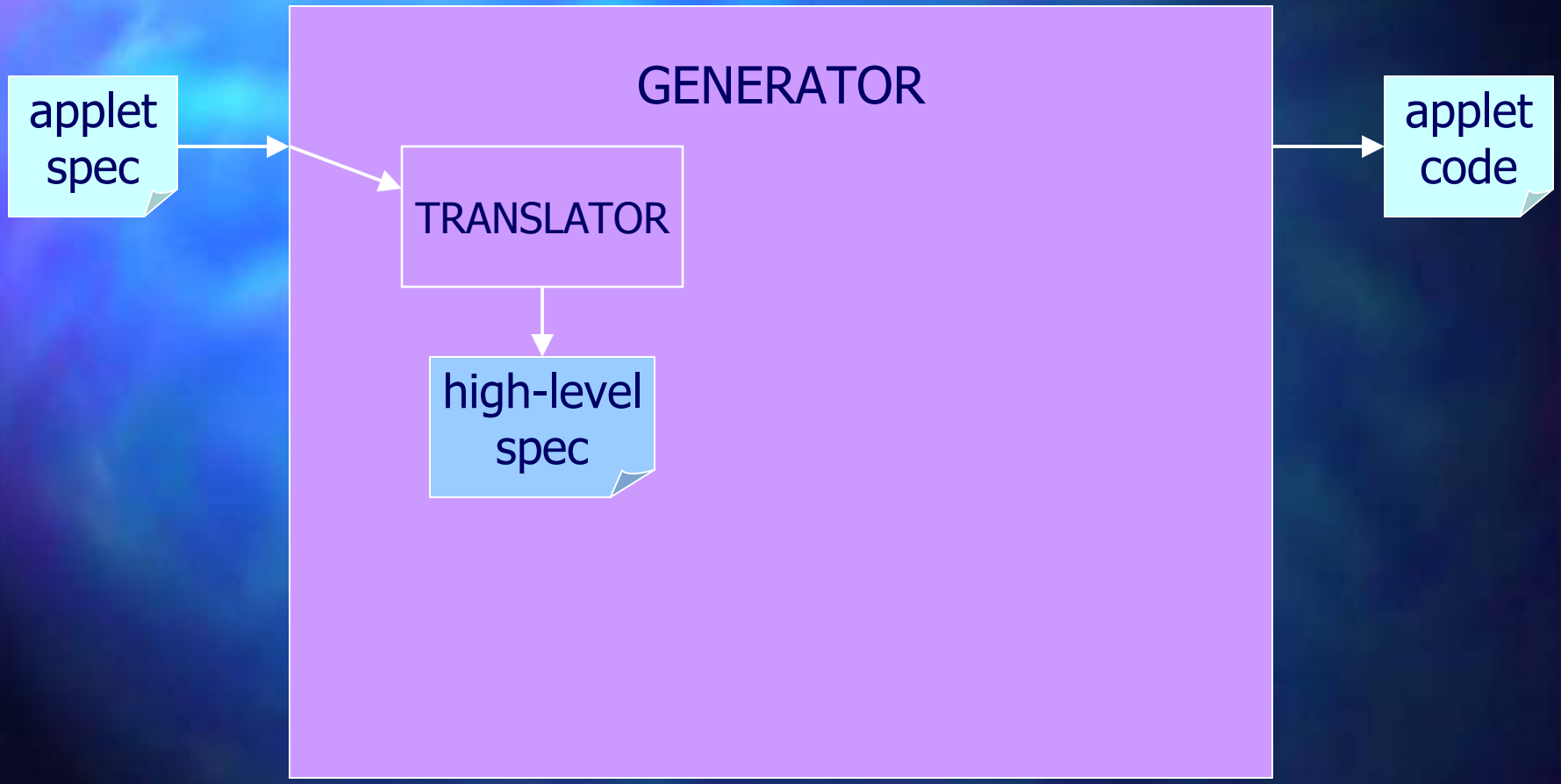


for:
- productivity
- high assurance

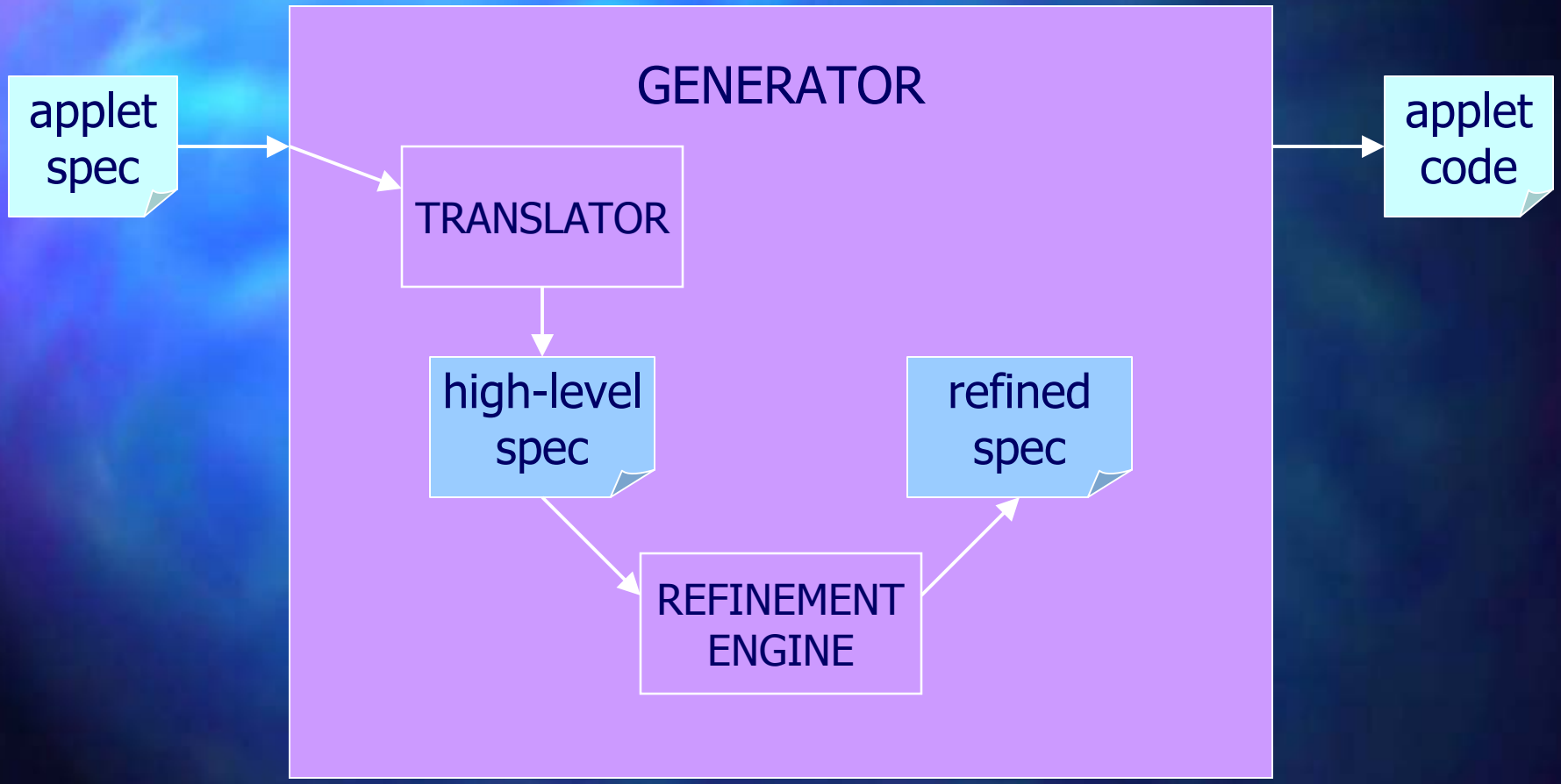
Inside the Generator



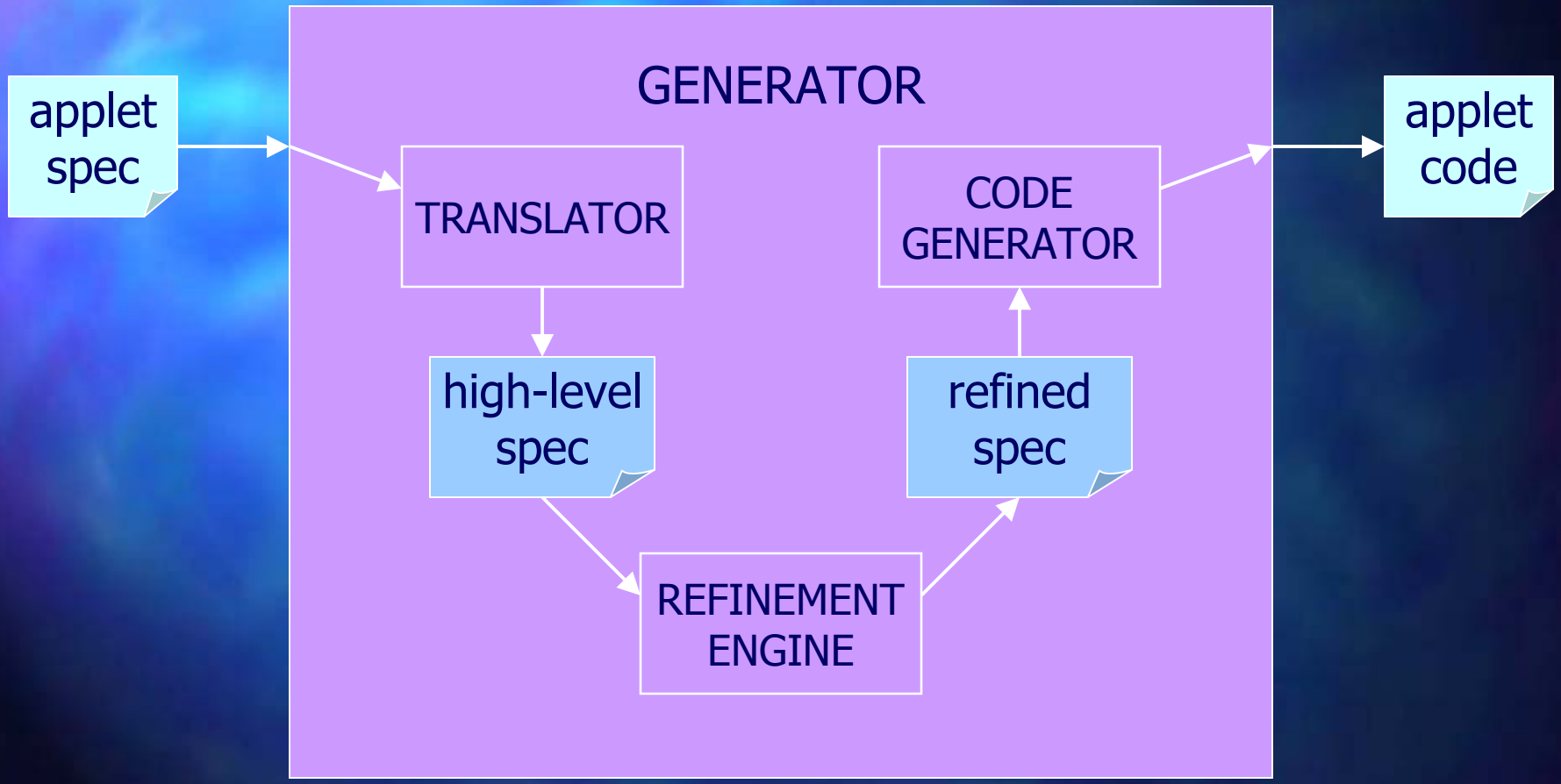
Inside the Generator



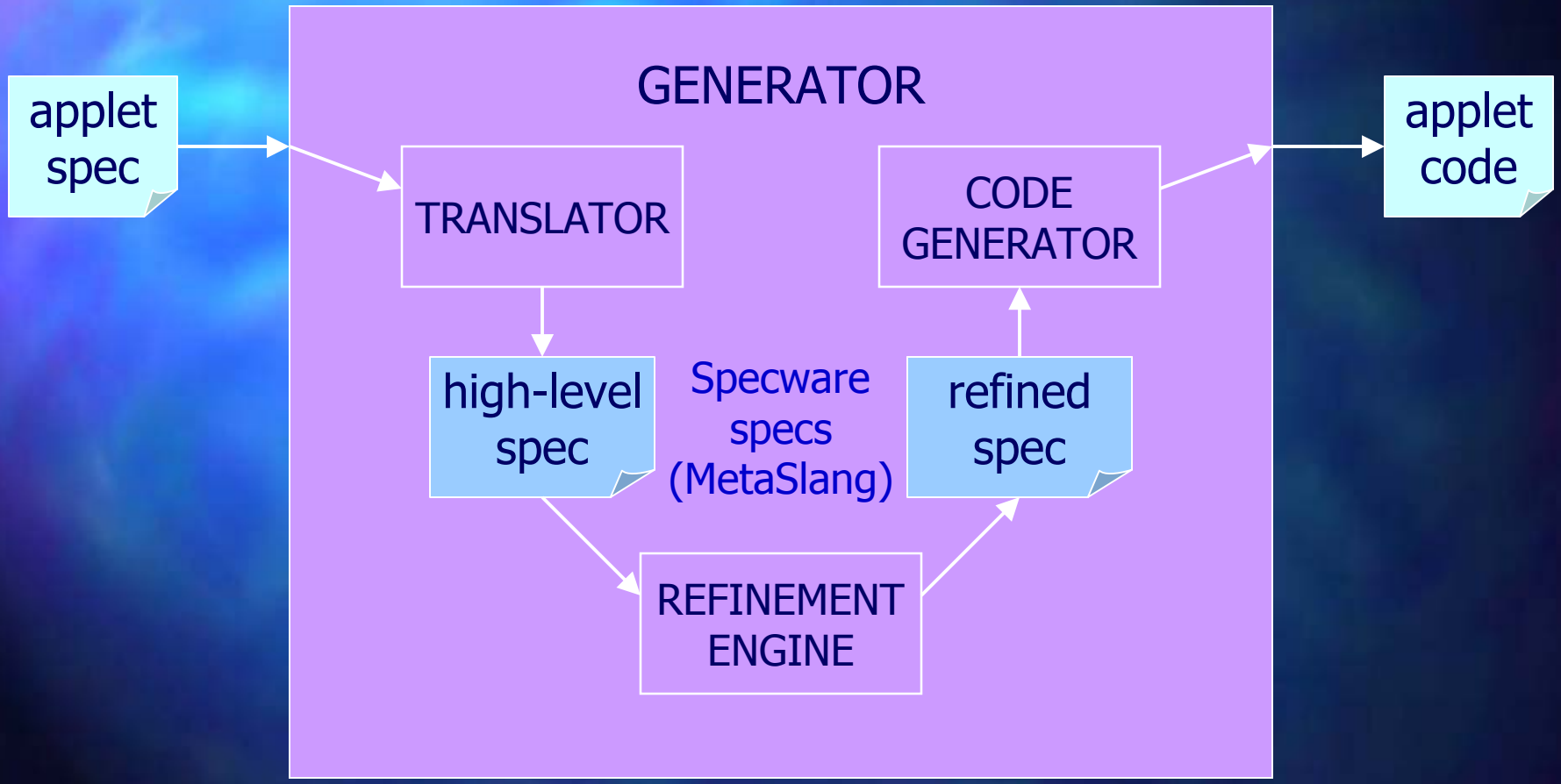
Inside the Generator



Inside the Generator

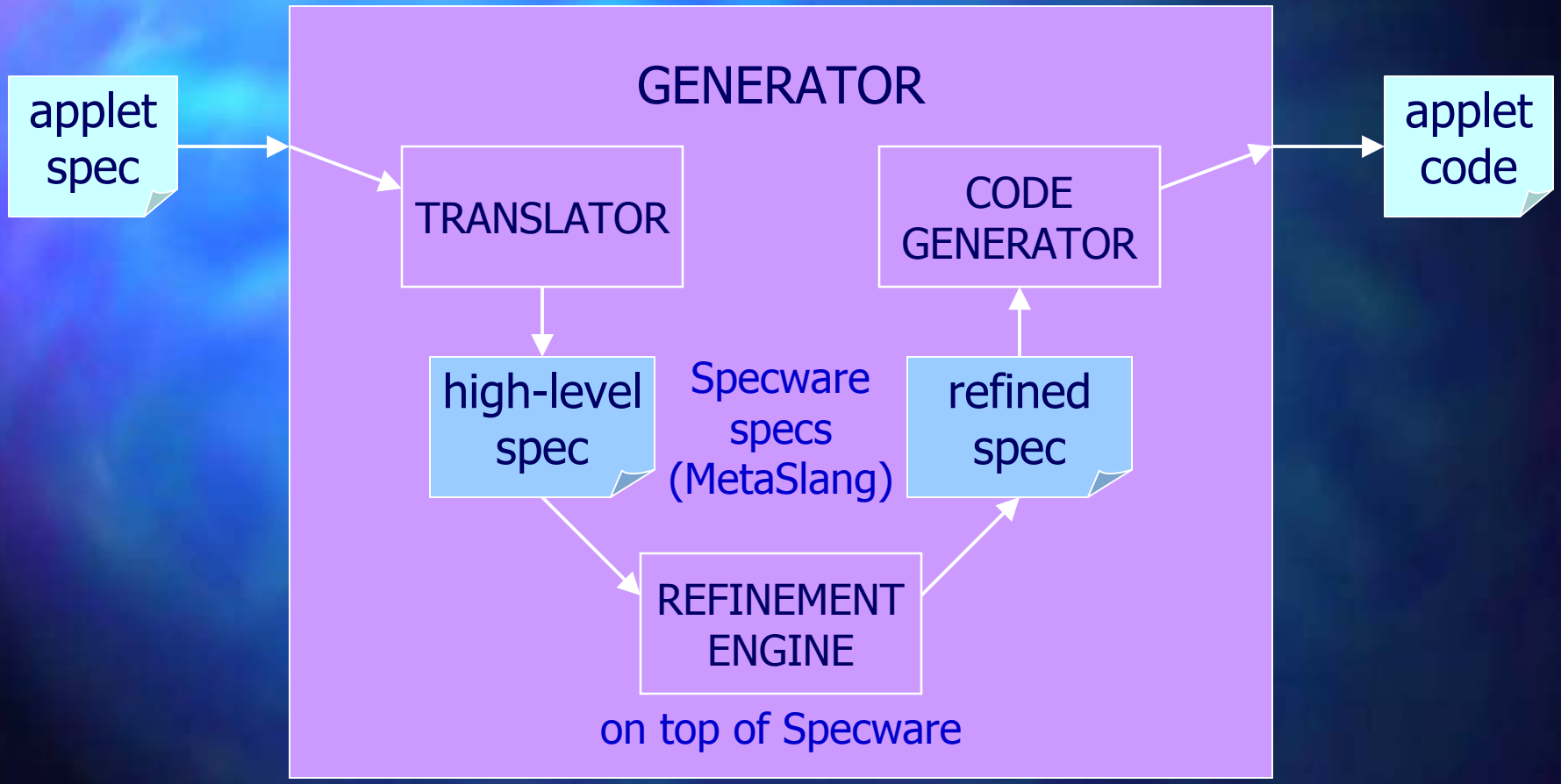


Inside the Generator



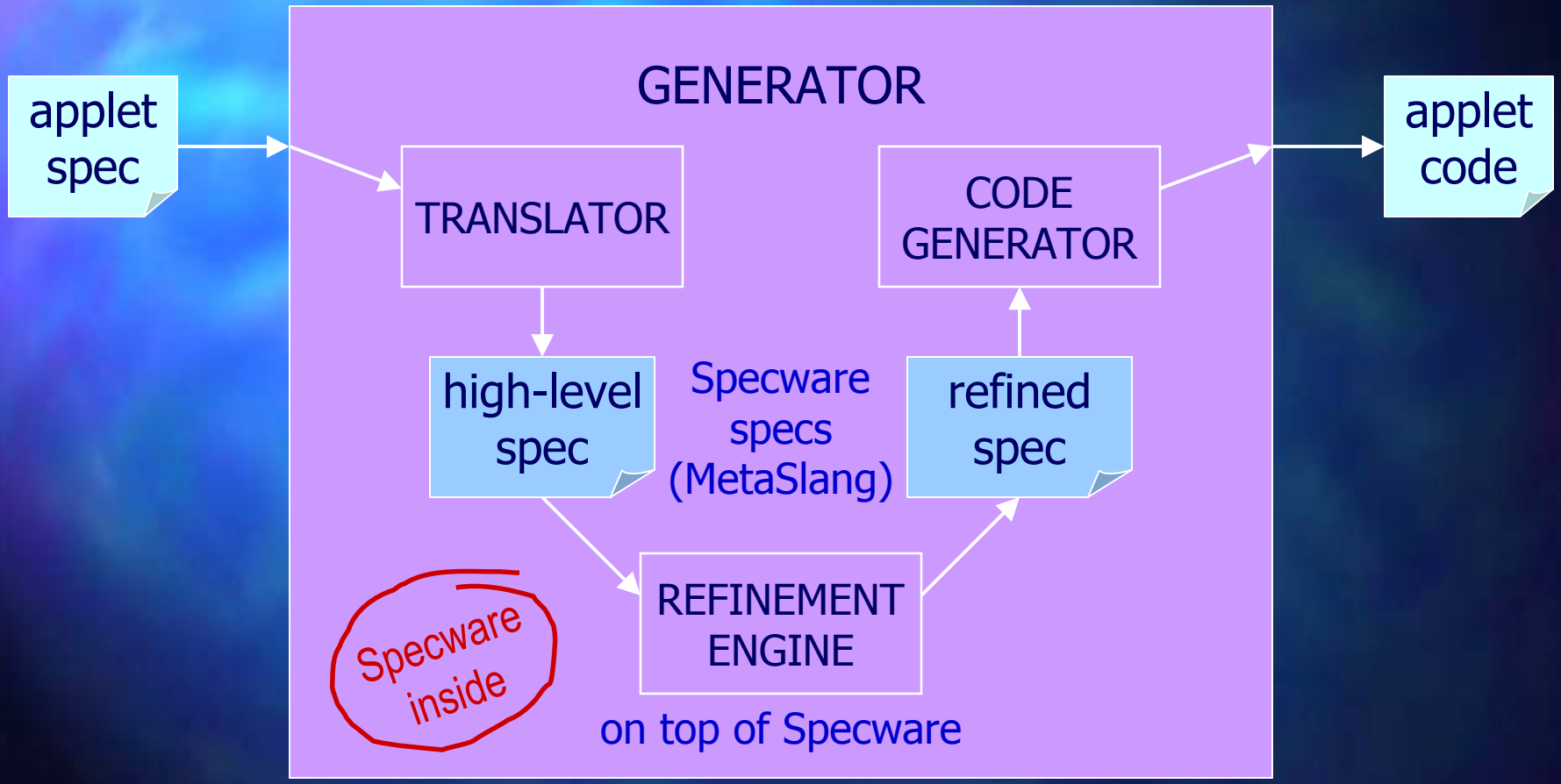
MetaSlang = spec language of Specware; version of higher-order logic

Inside the Generator



MetaSlang = spec language of Specware; version of higher-order logic

Inside the Generator



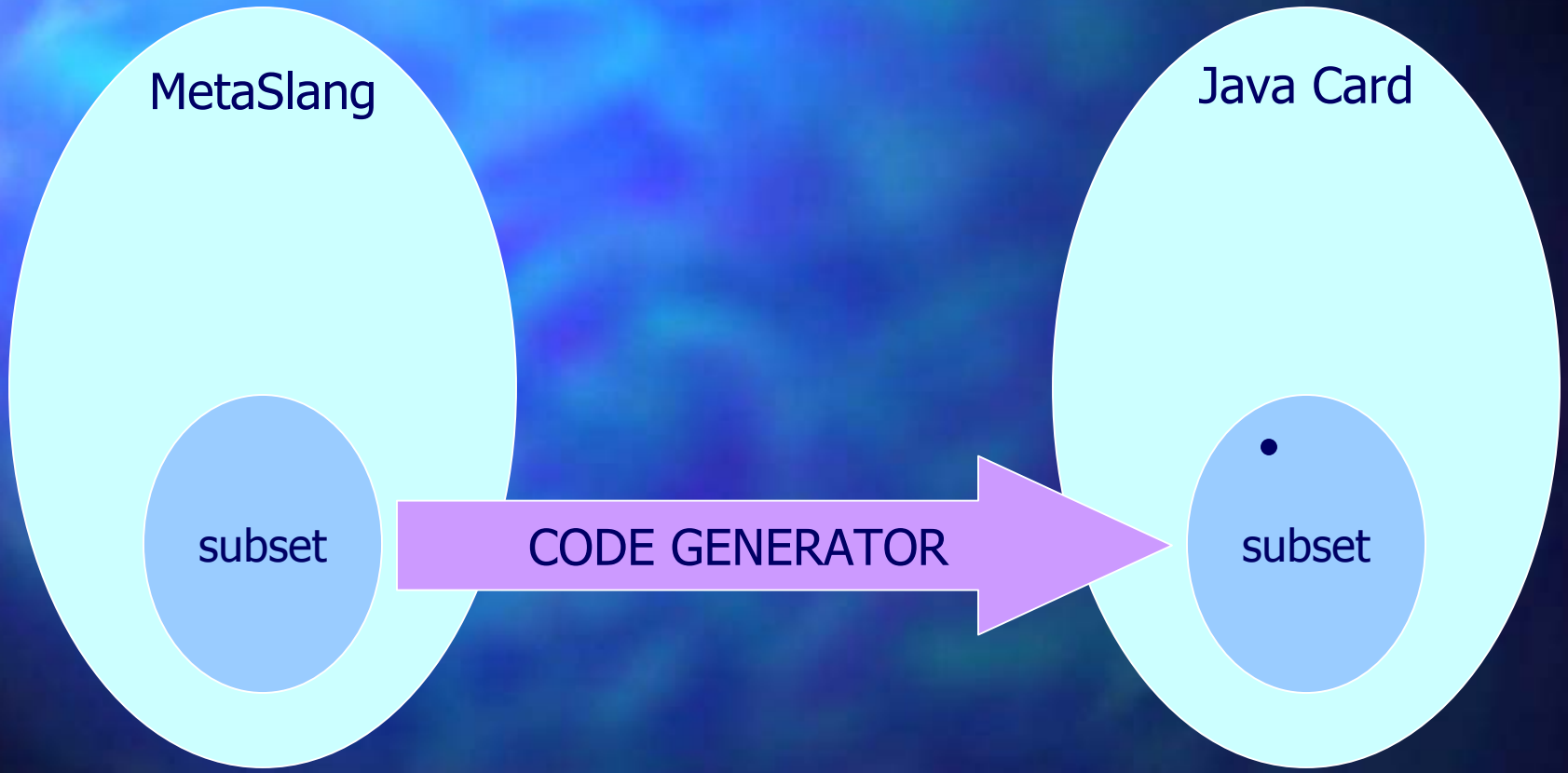
MetaSlang = spec language of Specware; version of higher-order logic

Summary

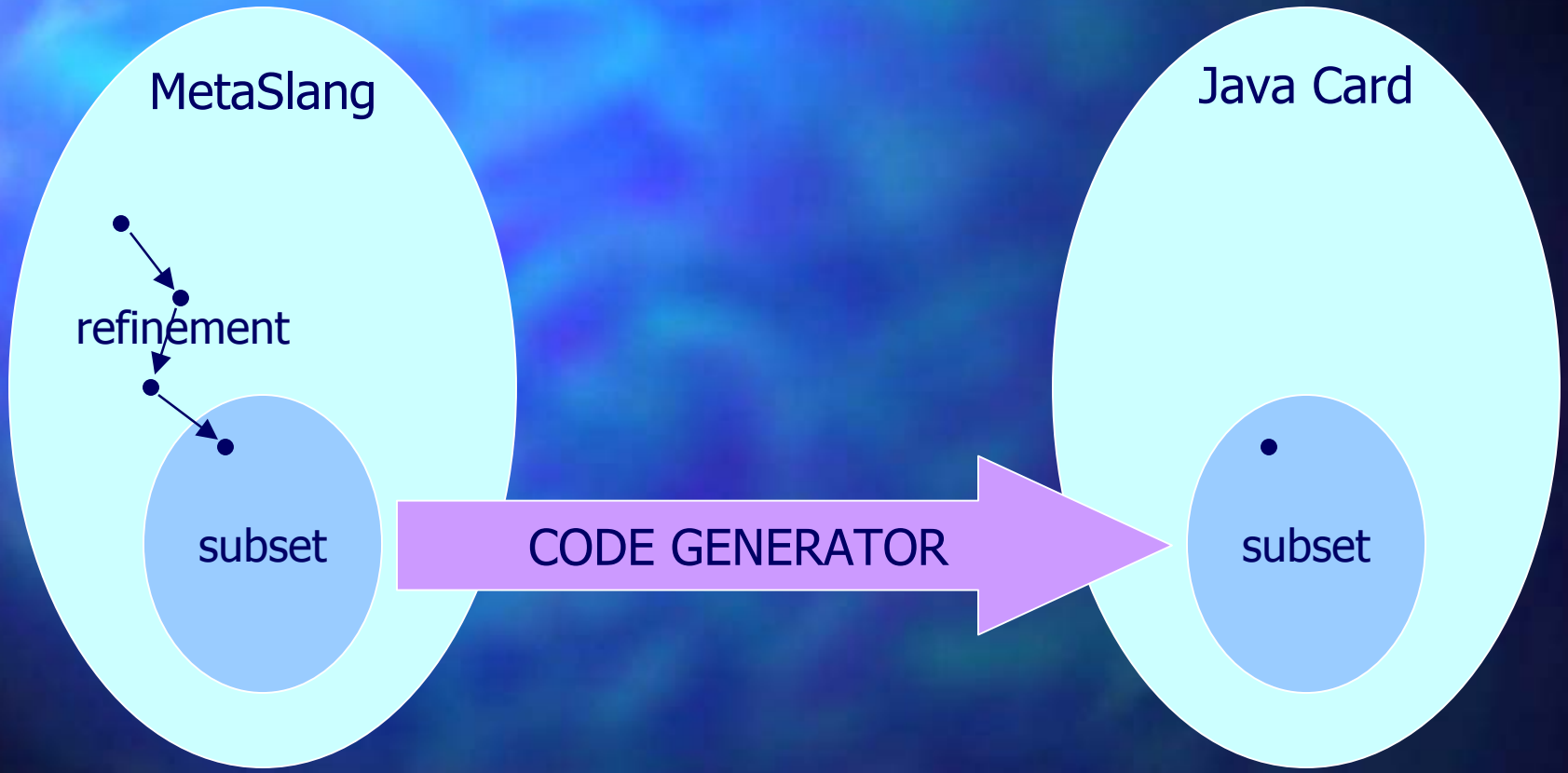
- Java Card
- Kestrel's work
- Applet generator
- Code generator
  - approach
  - details
  - checking correctness
  - accomplishments
- Current and future work

Code Generator

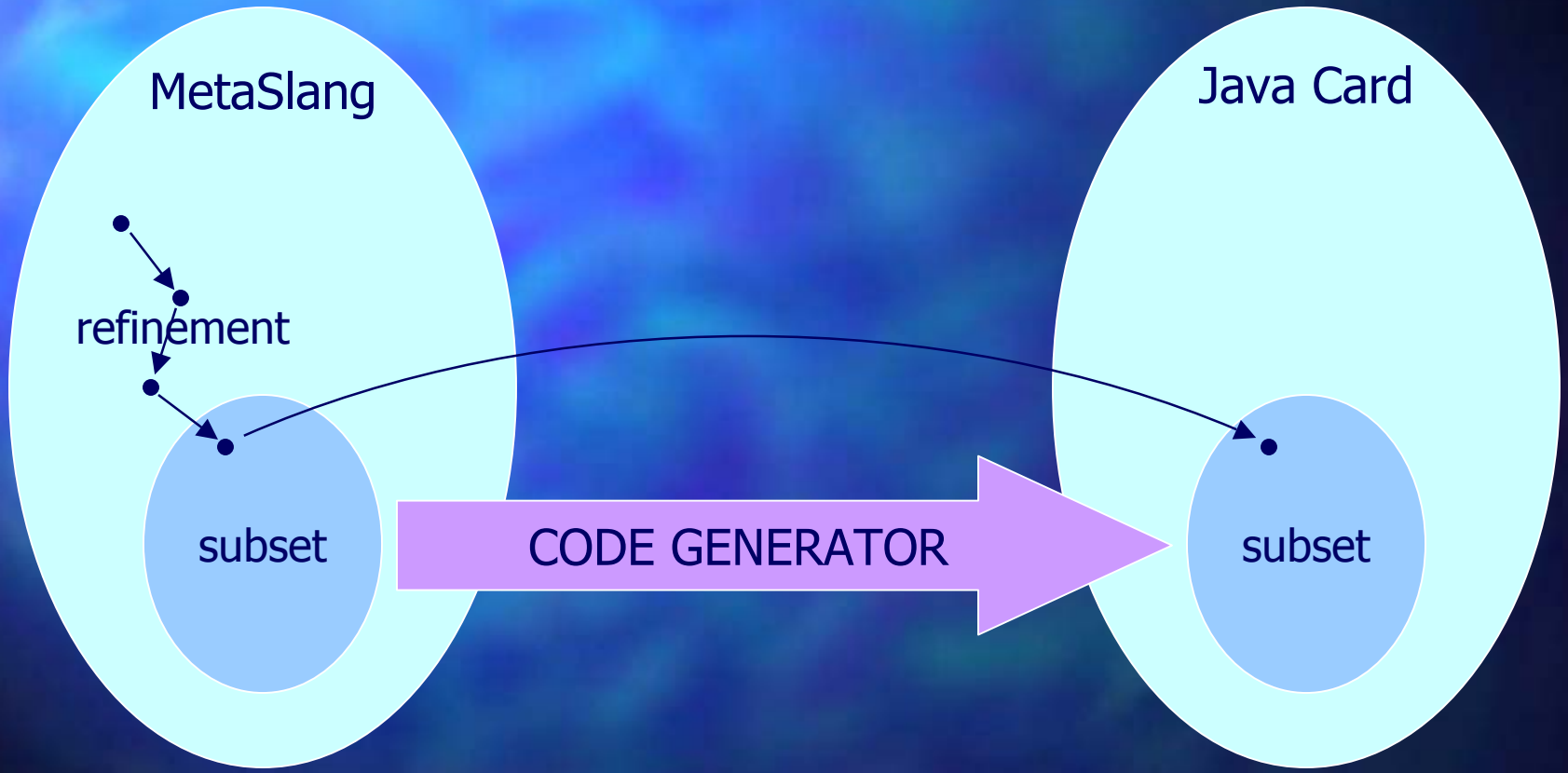
Code Generator



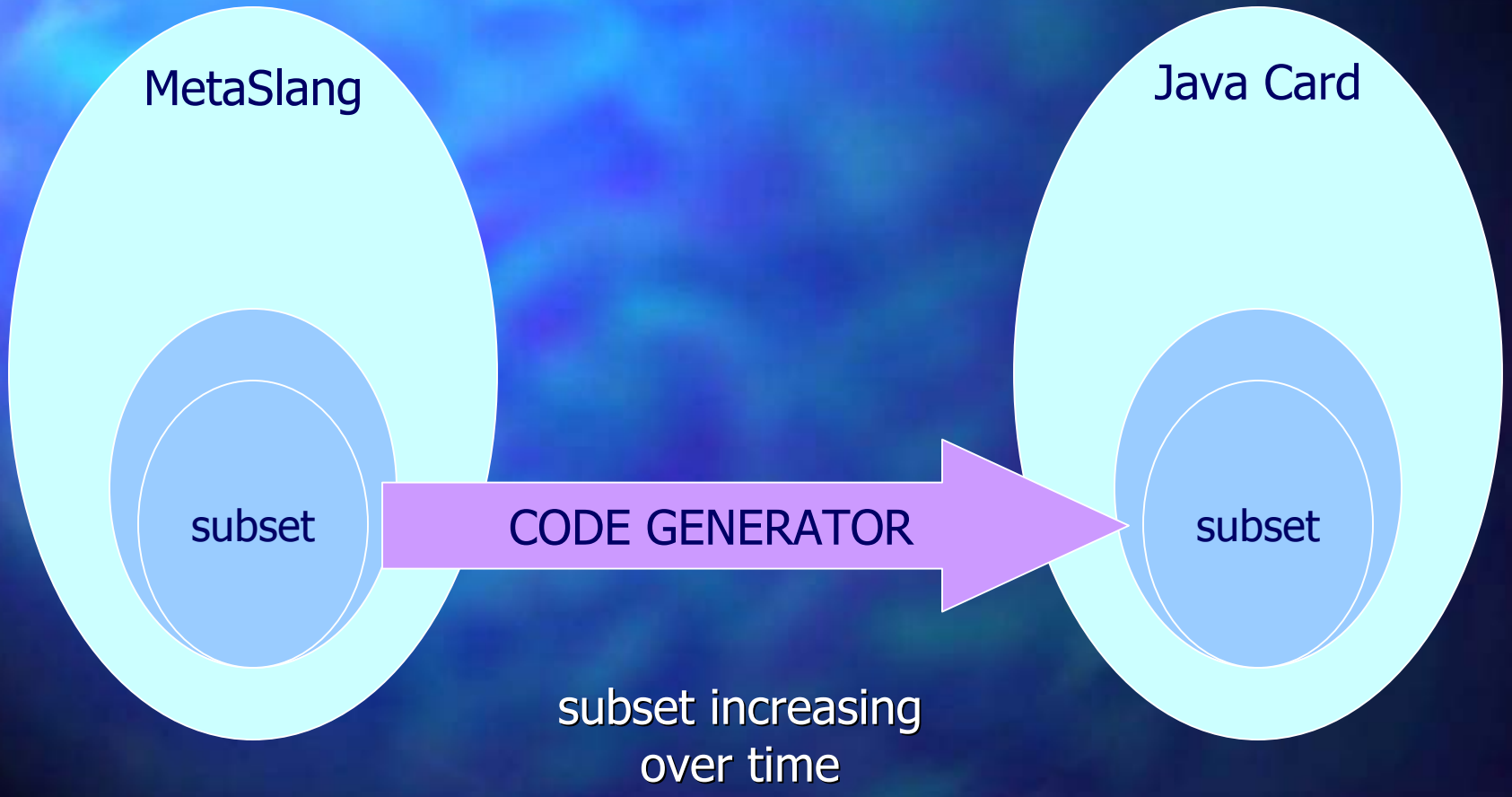
Code Generator



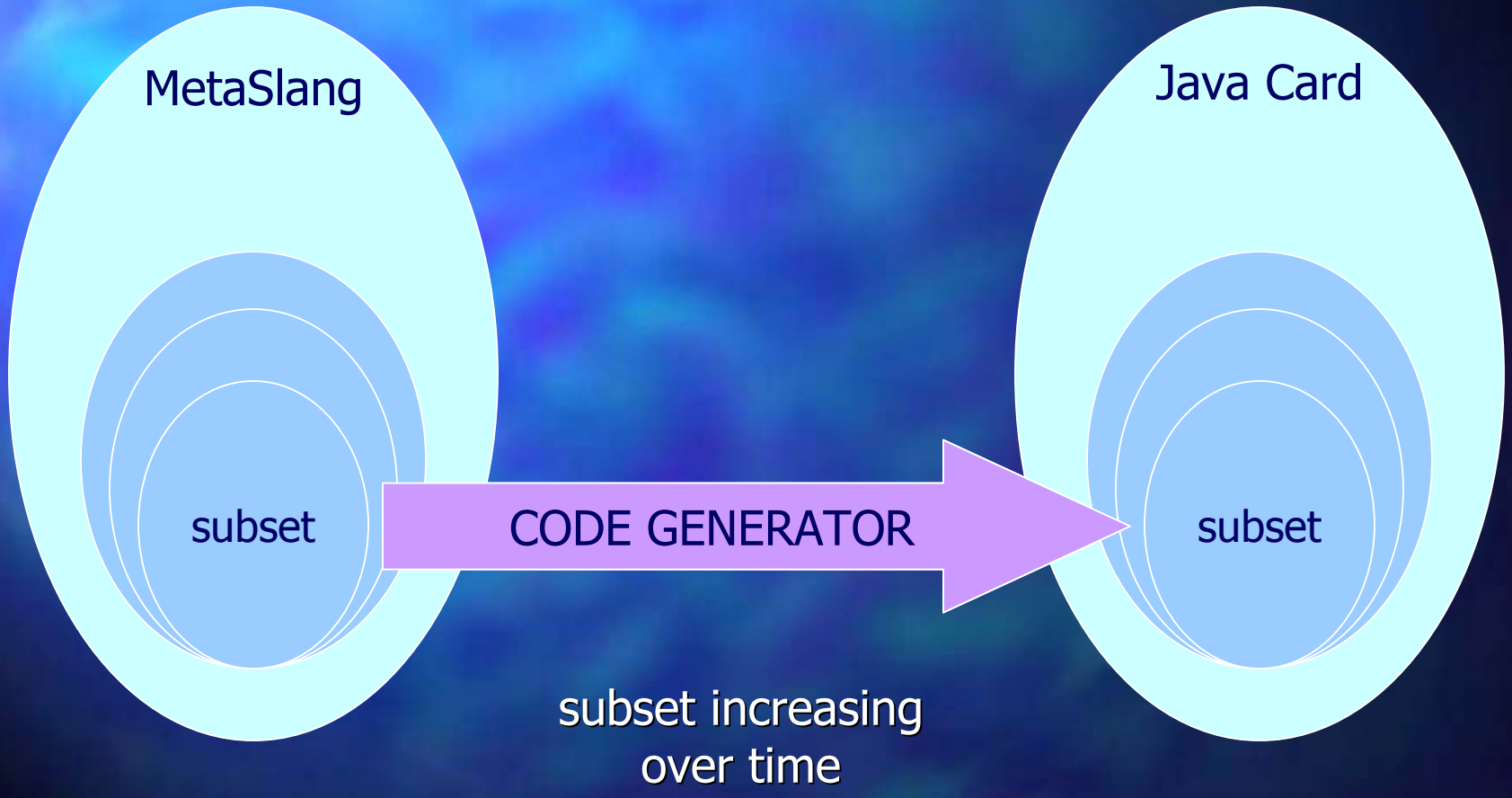
Code Generator



Code Generator



Code Generator



MetaSlang Subset

- $\text{MetaSlang}_{\text{JavaCard}} \subset \text{MetaSlang}$
- Specs in $\text{MetaSlang}_{\text{JavaCard}}$ represent Java Card programs
  - classes
  - fields
  - methods (incl. code)
- "Represent program" = "represent semantics of program"
- Code generator "extracts" Java Card program from its representation in $\text{MetaSlang}_{\text{JavaCard}}$

Summary

- Java Card
- Kestrel's work
- Applet generator
- Code generator
  - approach
  - details
  - checking correctness
  - accomplishments
- Current and future work

Language Embedding

- Def.: representation of a language J in a language M (e.g. interpreter of J written in M)
- Well-researched topic
- Terminology
  - deep embedding:
    syntax and semantics of J represented in M
  - shallow embedding:
    semantics (but no syntax) of J represented in M

Deep vs. Shallow: Example

- Boolean expressions
- Syntax: variables + boolean operators
  - (x AND y) OR z
- Semantics: given a boolean value for each variable, expression evaluates to boolean value
  - x = T, y = T, z = F
  - (x AND y) OR z evaluates to T

Deep Embedding: Example

```
sort Expr = | var Var
            | and Expr * Expr
            | ...
sort Env = Var -> Bool
op eval : Expr -> (Env -> Bool)
...
axiom fa(exp1,exp2)
        eval(and(exp1,exp2)) =
        fn env -> eval(exp1)(env) &
                  eval(exp2)(env)
```

syntax

semantics

Deep Embedding: Example

```
sort Expr = | var Var
            | and Expr * Expr
            | ...
sort Env = Var -> Bool
op eval : Expr -> (Env -> Bool)
...
axiom fa(exp1,exp2)
        eval(and(exp1,exp2)) =
        fn env -> eval(exp1)(env) &
                  eval(exp2)(env)
```

syntax

semantics

boolean expressions are assigned semantics

Shallow Embedding: Example

```
sort Env = Var -> Bool
sort Expr = Env -> Bool
...
op and : Expr * Expr -> Expr
axiom fa(exp1,exp2)
          and(exp1,exp2) =
          fn env -> exp1(env) & exp2(env)
```

semantics only

Shallow Embedding: Example

```
sort Env = Var -> Bool
sort Expr = Env -> Bool
...
op and : Expr * Expr -> Expr
axiom fa(exp1,exp2)
        and(exp1,exp2) =
        fn env -> exp1(env) & exp2(env)
```

semantics only

boolean expressions are identified with their semantics

MetaSlang$_{\text{JavaCard}}$

- Shallow embedding of (subset of) Java Card into MetaSlang
- Subset of Java Card chosen
  - sufficient for non-trivial applets
  - will grow over time
- Why not deep embedding?
  - because shallow embedding is simpler (smaller)
  - we may switch to deep embedding in the future
- Includes Java Card APIs

Chosen Java Card Subset

- Includes
  - primitive values and types
  - classes and limited interfaces
  - instance fields, final static fields
  - methods
  - most expressions and statements
  - objects (class instances & arrays)
  - exceptions
- Excludes
  - full interfaces
  - inheritance, hiding, overriding
  - access modifiers (public, private, …)
  - packages
  - optional type int

Java Card Language Embedding (1)

```
sort Class    % classes in program
sort Field    % fields  in program
sort Method   % methods in program
op field_owner : Field -> Class
op method_owner : Method -> Class
sort Type = | boolean | byte | short
            | class Class | ...
op field_type : Field -> Type
op method_arg_types : Method -> FSeq(Type)
op method_res_type : Method ->
                     | void | nonvoid Type
...
```

Java Card Language Embedding (2)

```
sort ByteValue = {b : Int | -128 <= b & b < 128}
...
sort Value = | byte ByteValue
             | ref Reference
             | ...
sort ClassInstance = ...
sort Array = ...
sort Object = | clins ClassInstance | arr Array
sort Heap = FMap(Reference,Object)
...
```

Java Card Language Embedding (3)

```
sort LocalVariableStore = ...
sort State = {heap  : Heap,
              local : LocalVariableStore}
sort Expression = ...   % later
sort Statement = ...    % later
op plus : Expression * Expression -> Expression
axiom def_of_plus is ...
op while : Expression * Statement -> Statement
axiom def_of_while is ...
op method_body : Method -> Statement
...
```

Expressions and Statements

- May have side effects (i.e., cause state changes)
- May throw exceptions
- May return a value
- May cause input/output events
  - Java:
```
if (infile.read() == 3)
outfile.print("hello");
```
  - Java Card: `apdu.setIncomingAndReceive()`
- State changes may be non-deterministic
  - Java Card: `random.generateData()`

Expression and Statement Embedding: Attempt #1

```
sort Expression =
  State -> State * ExprResult

sort Statement =
  State -> State * StatResult
```

Expression and Statement Embedding: Attempt #1

```
sort Expression =
  State -> State * ExprResult

sort Statement =
  State -> State * StatResult
```

but: deterministic, no input/output events

Expression and Statement Embedding: Attempt #2a

```
sort Expression =
  State * InputEvent ->
  State * OutputEvent * ExprResult

sort Statement =
  State * InputEvent ->
  State * OutputEvent * StatResult
```

Expression and Statement Embedding: Attempt #2a

```
sort Expression =
  State * InputEvent ->
  State * OutputEvent * ExprResult

sort Statement =
  State * InputEvent ->
  State * OutputEvent * StatResult
```

but: deterministic

Expression and Statement Embedding: Attempt #2b

```
sort Expression =
  State * State * ExprResult -> Boolean

sort Statement =
  State * State * StatResult -> Boolean
```

Expression and Statement Embedding: Attempt #2b

```
sort Expression =
  State * State * ExprResult -> Boolean

sort Statement =
  State * State * StatResult -> Boolean
```

but: no input/output events

Expression and Statement Embedding: Attempt #3

```
sort Expression =
  State * InputEvent *
  State * OutputEvent * ExprResult ->
  Boolean


sort Statement =
  State * InputEvent *
  State * OutputEvent * StatResult ->
  Boolean
```

Expression and Statement Embedding: Attempt #3

```
sort Expression =
  State * InputEvent *
  State * OutputEvent * ExprResult ->
  Boolean

sort Statement =
  State * InputEvent *
  State * OutputEvent * StatResult ->
  Boolean
```

OK: input/output events, non-deterministic!

Expression and Statement Embedding (1)

```
sorts InputEvent, OutputEvent
sort SideEffect = {old : State,
                   new : State,
                   in  : FSeq(InputEvent),
                   out : FSeq(OutputEvent)}
sort ExprResult = ...
sort StatResult = ...
sort Expression =  % relation
     SideEffect * ExprResult -> Boolean
sort Statement =   % relation
     SideEffect * StatResult -> Boolean
```

Expression and Statement Embedding (2)

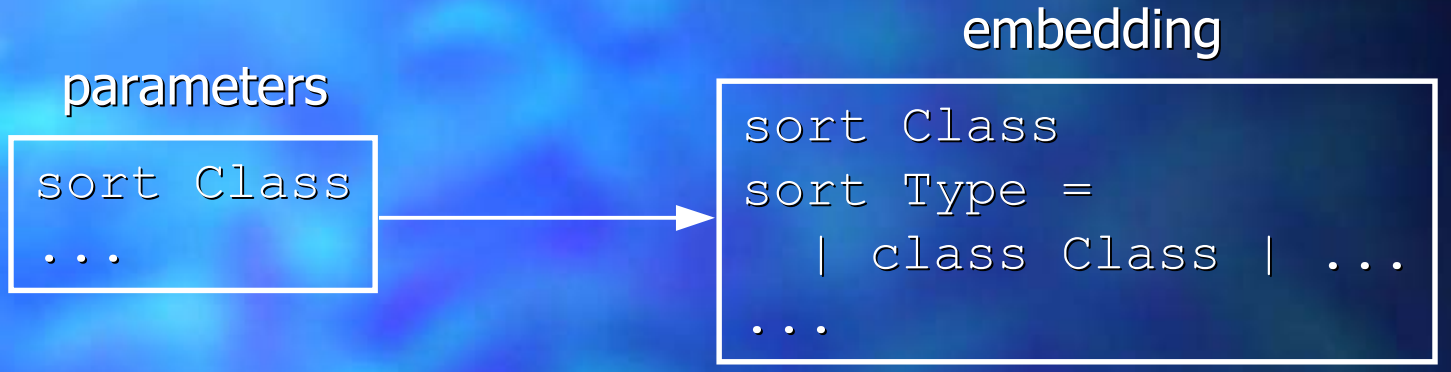
```
op plus : Expression * Expression -> Expression
axiom def_of_plus is
  fa(exp1,exp2)
    plus(exp1,exp2) =
    fn(seff,eres) ->
      (ex(seff1,eres1,seff2,eres2)
         exp1(seff1,eres1) &
         exp2(seff2,eres2) &
         seff.old = seff1.old &
         seff1.new = seff2.old &
         seff2.new = seff.new &
         seff.in = seff1.in || seff2.in &
         seff.out = seff1.out || seff2.out &
         eres = eres1 + eres2)
```

Program Representation by Instantiation of Embedding

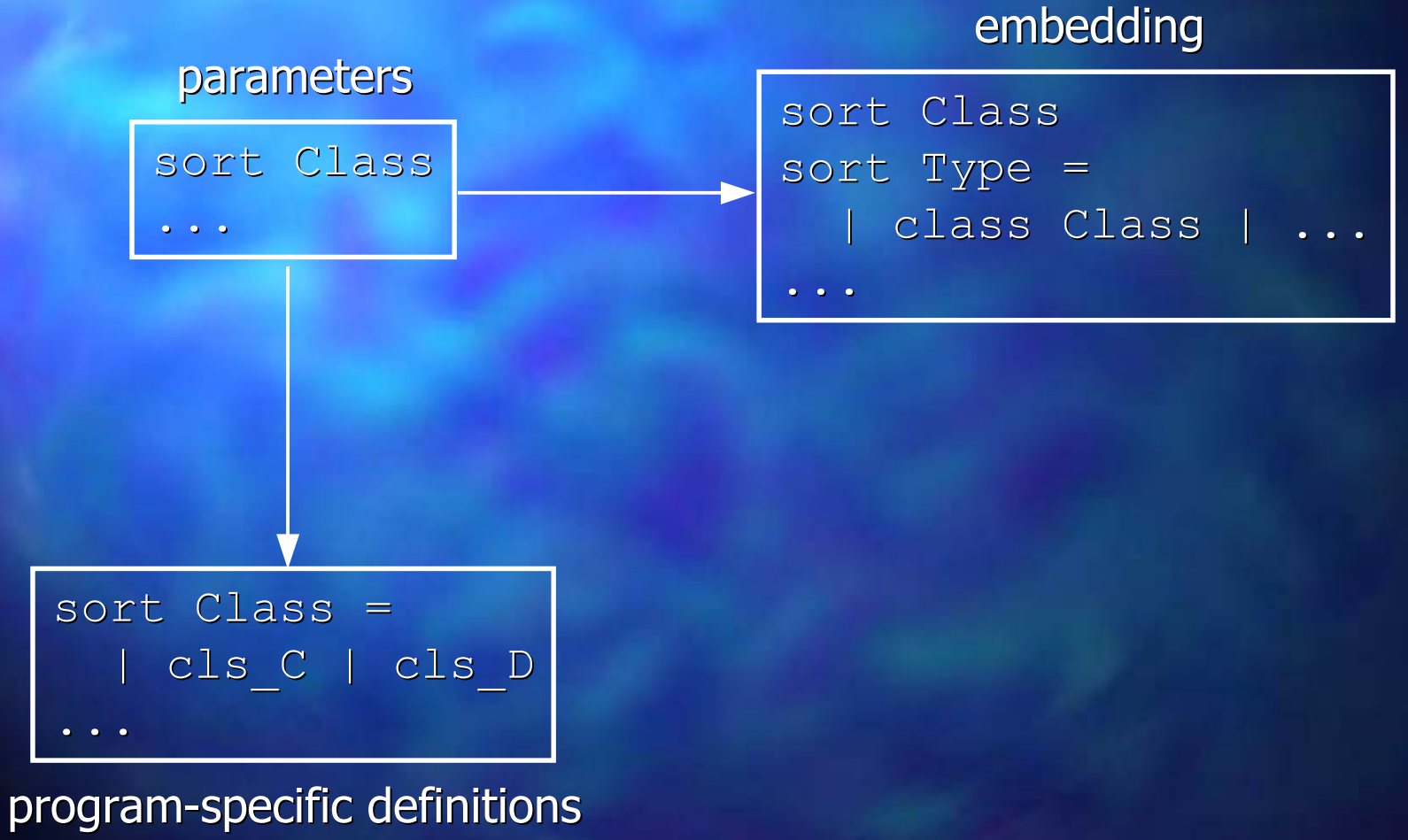
embedding

```
sort Class
sort Type =
  | class Class | ...
...
```

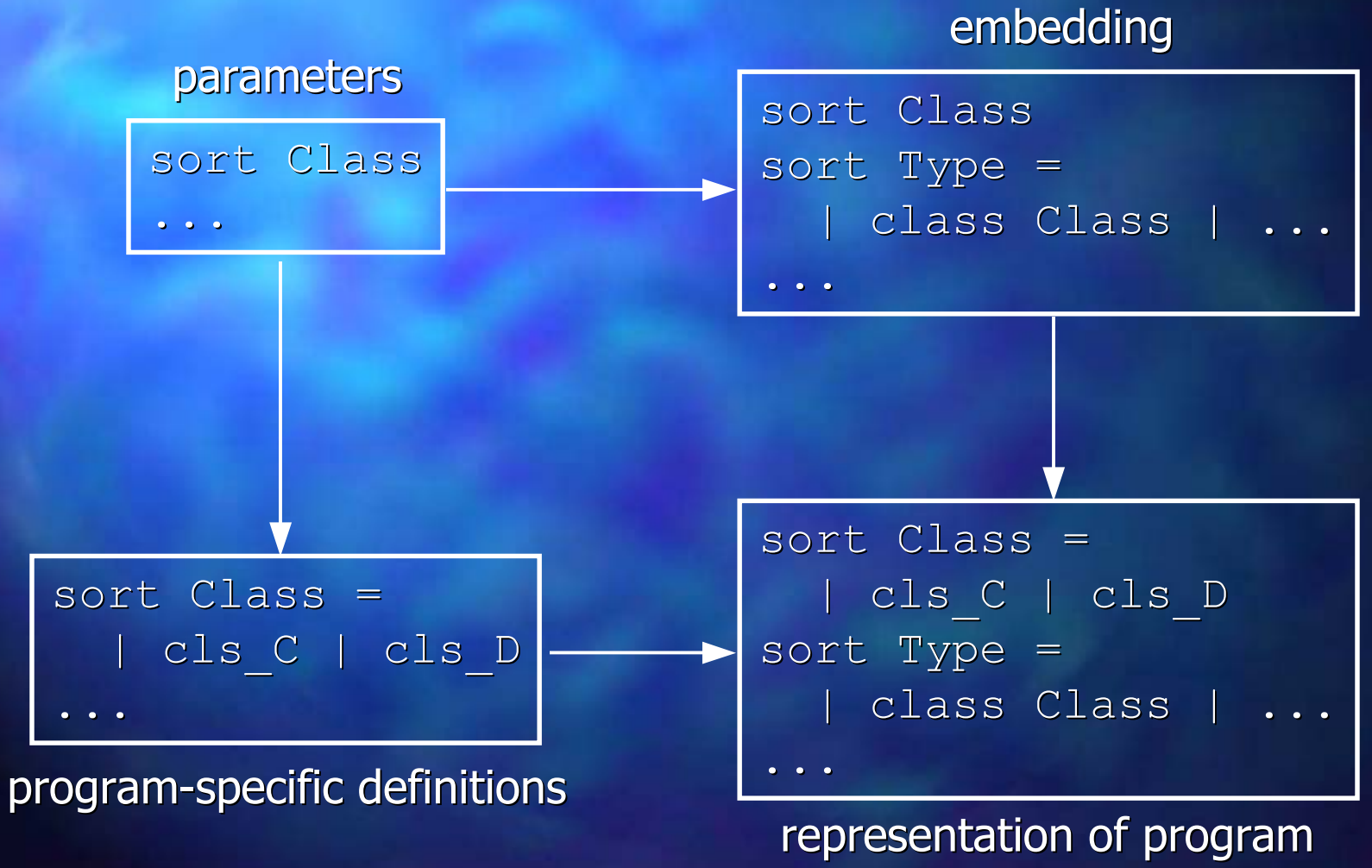
Program Representation by Instantiation of Embedding



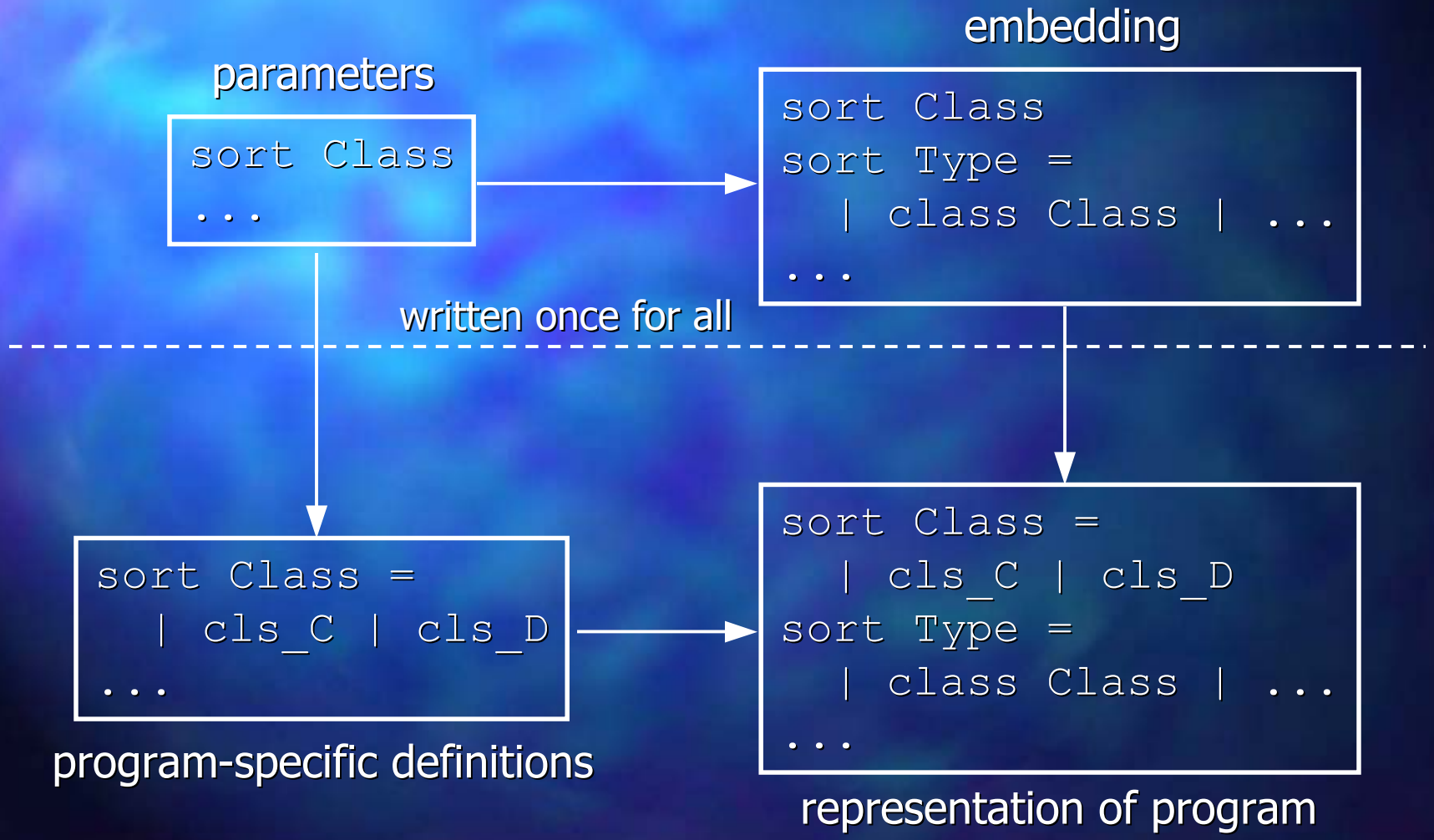
Program Representation by Instantiation of Embedding



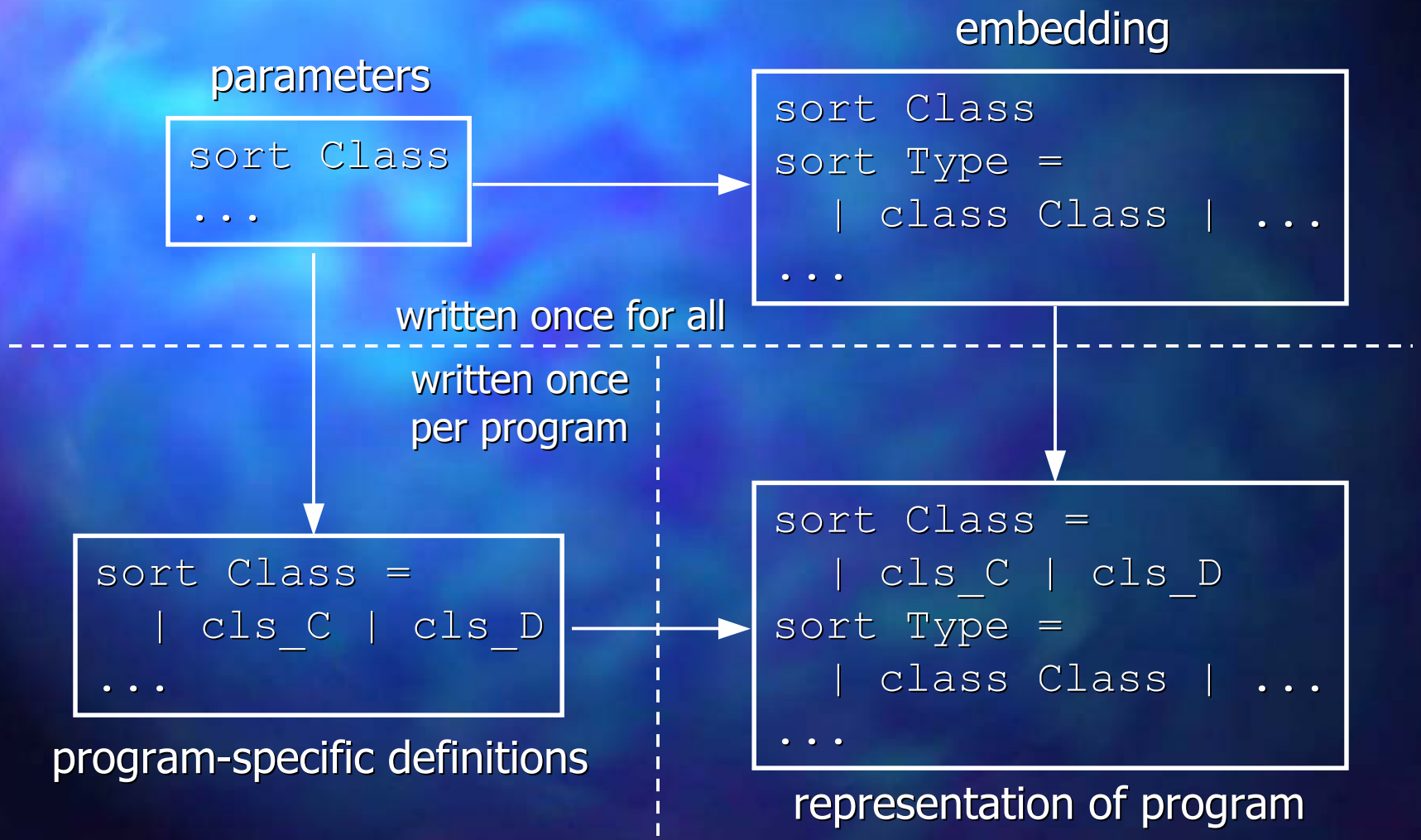
Program Representation by Instantiation of Embedding



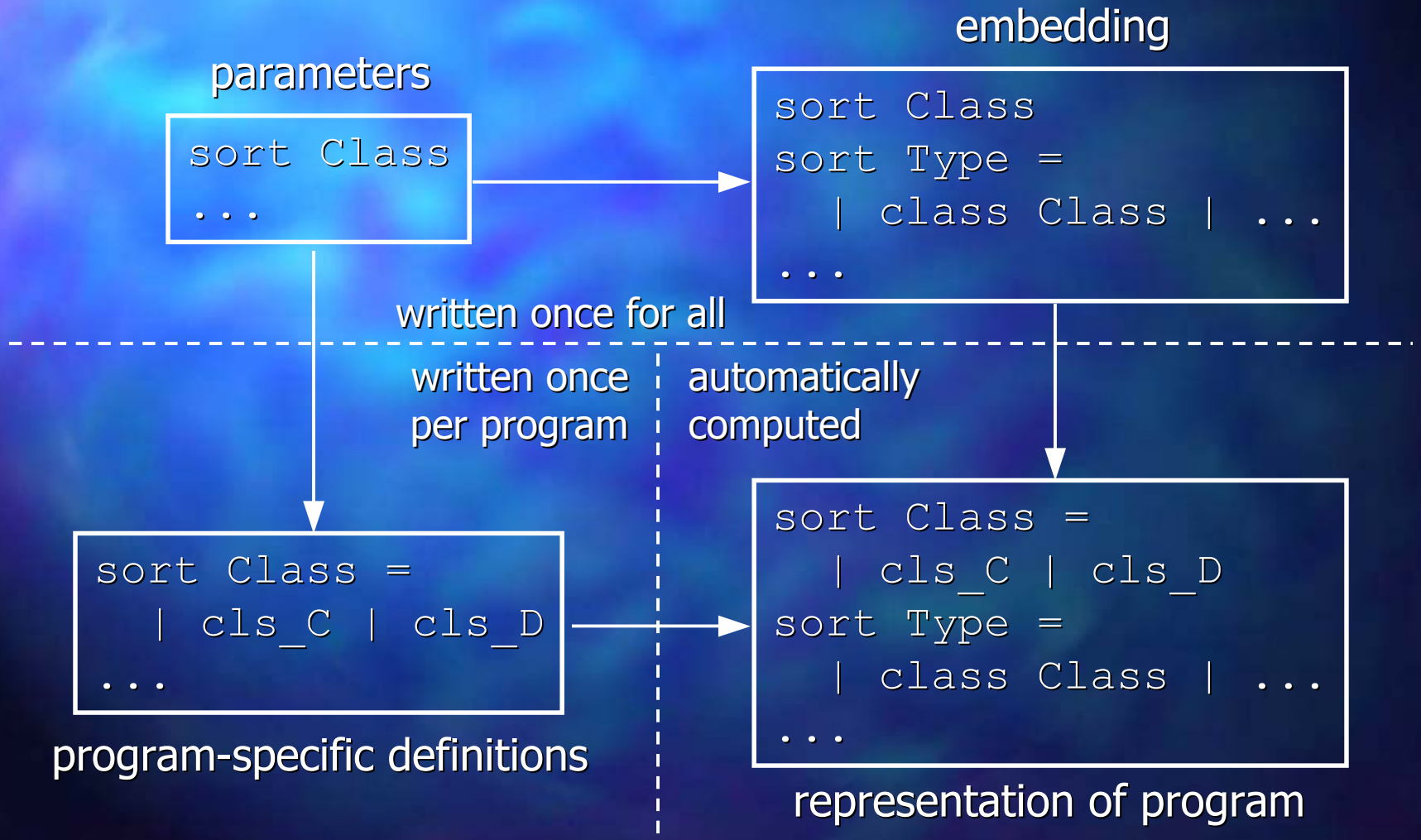
Program Representation by Instantiation of Embedding



Program Representation by Instantiation of Embedding



Program Representation by Instantiation of Embedding



Example

```
class C {
  byte f;
}

class D {
  C g;
  byte m() {
    return g.f;
  }
}
```

```
sort Class = | cls_C | cls_D
sort Field = | fld_f | fld_g
sort Method = | mth_m
axiom field_owner(fld_f) = cls_C
axiom field_type(fld_f) = prim(byte)
axiom field_owner(fld_g) = cls_D
axiom field_type(fld_g) = class(cls_C)
axiom method_owner(mth_m) = cls_D
...
axiom
  method_body(mth_m) =
  stat_return
    (field_access
      (field_access expr_this_D fld_g)
      fld_f)
```

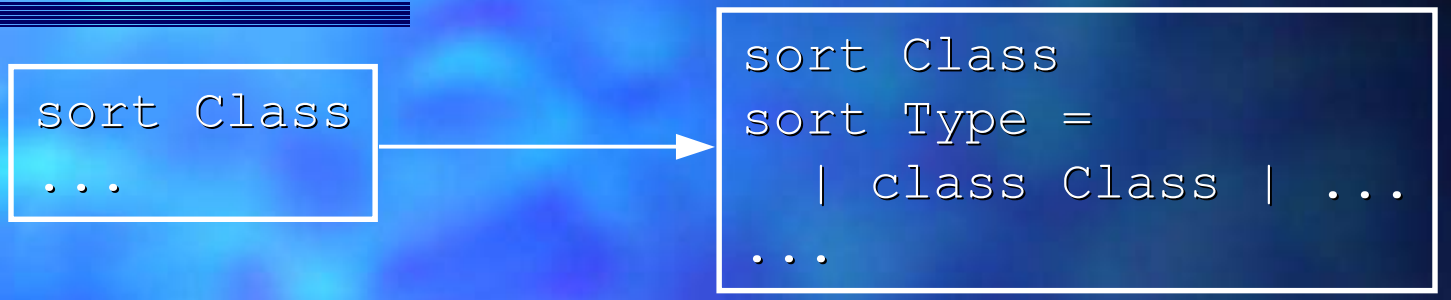
Chosen Java Card APIs

- Formalized substantial subsets of
  - APDU (most complex class)
  - ISOException
  - DESKey
  - RSAPrivateKey
  - RandomData
  - Cipher
  - KeyBuilder
  - ISO7816
  - Key

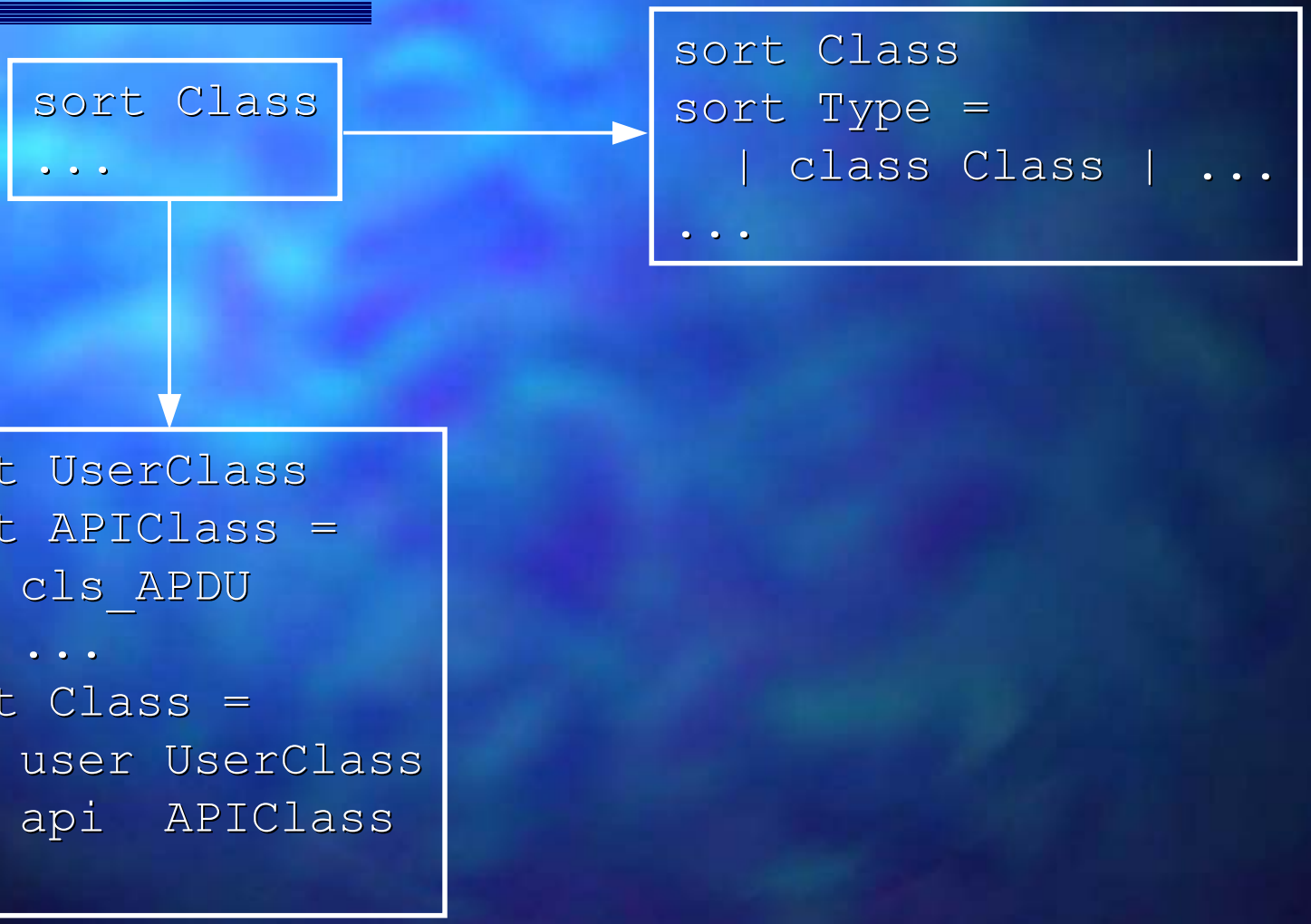
API Formalization

```
sort APIClass = | cls_APDU | cls_Cipher | ...
sort APDUState = | received_header
                 | receiving_data
                 | received_data
                 | sending_data
                 | sent_data
sort RSAKey = {modulus  : FSeq ByteValue,
               exponent : FSeq ByteValue}
...
op APDU_sendBytes : APIMethod
axiom APDU_sendBytes = ...
```

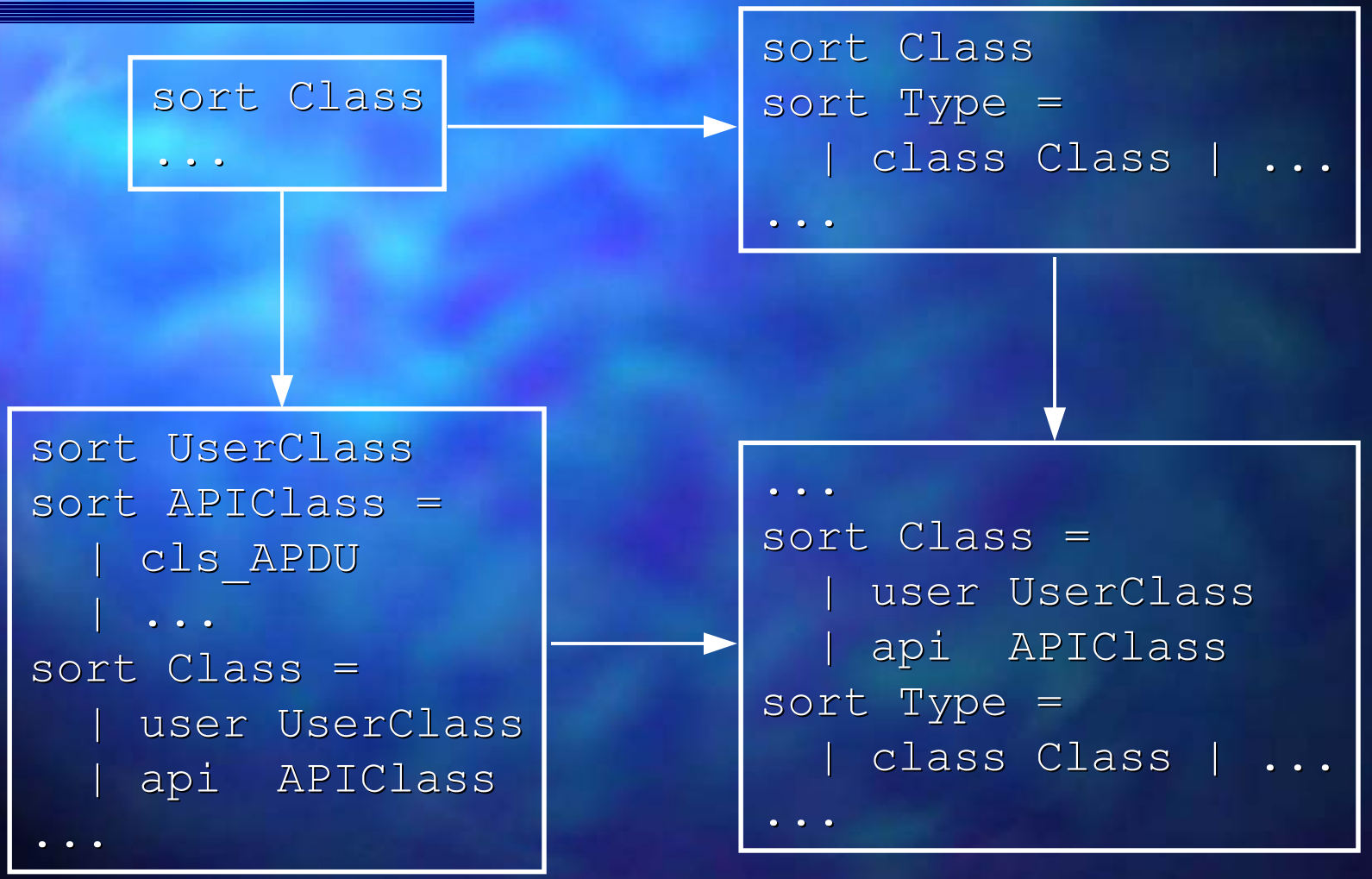
API Formalization by Partial Instantiation of Embedding



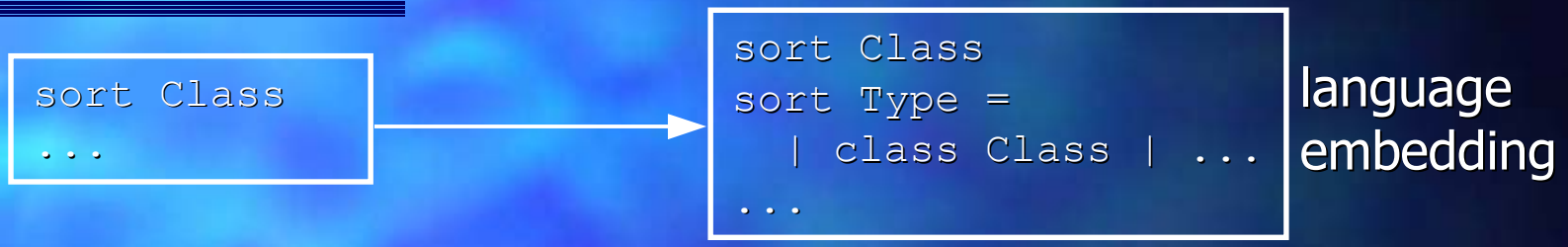
API Formalization by Partial Instantiation of Embedding



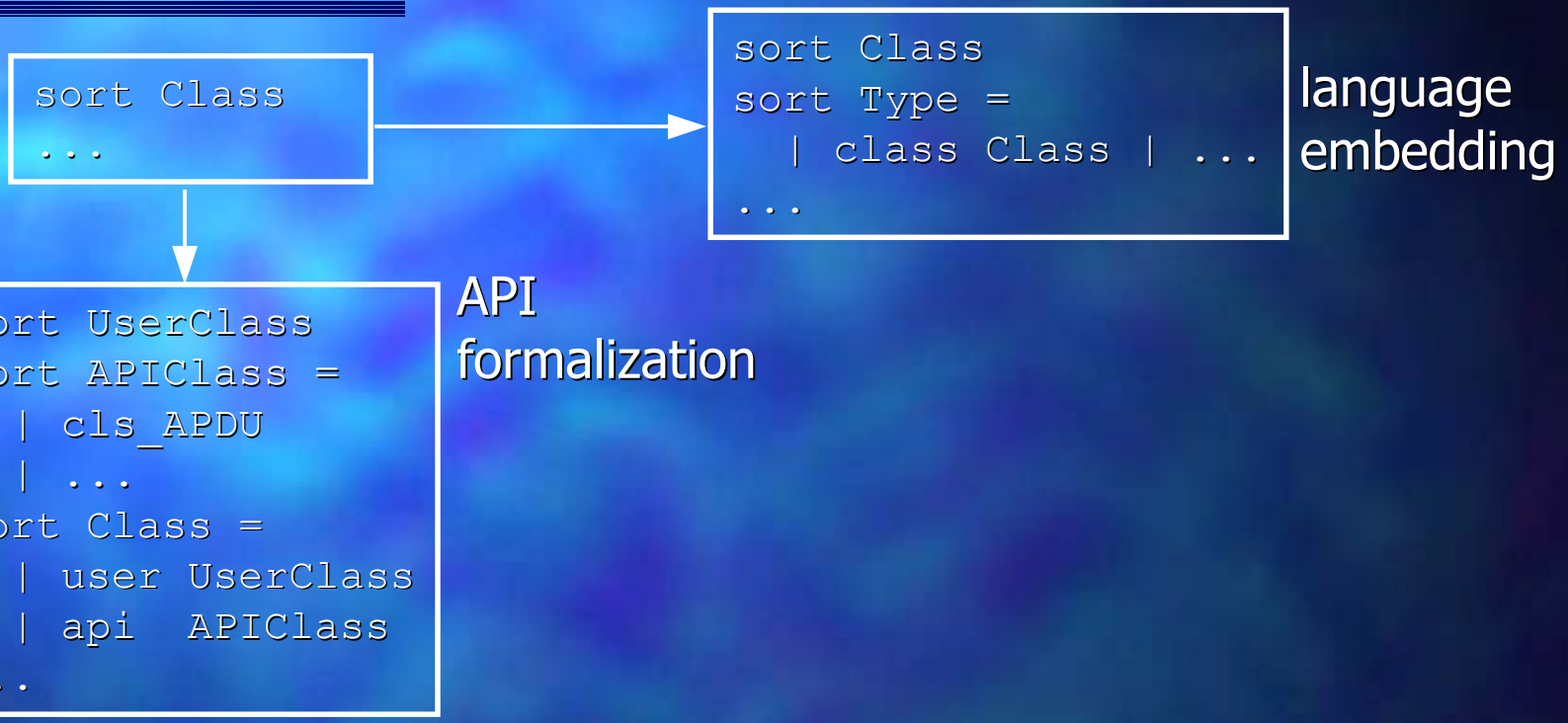
API Formalization by Partial Instantiation of Embedding



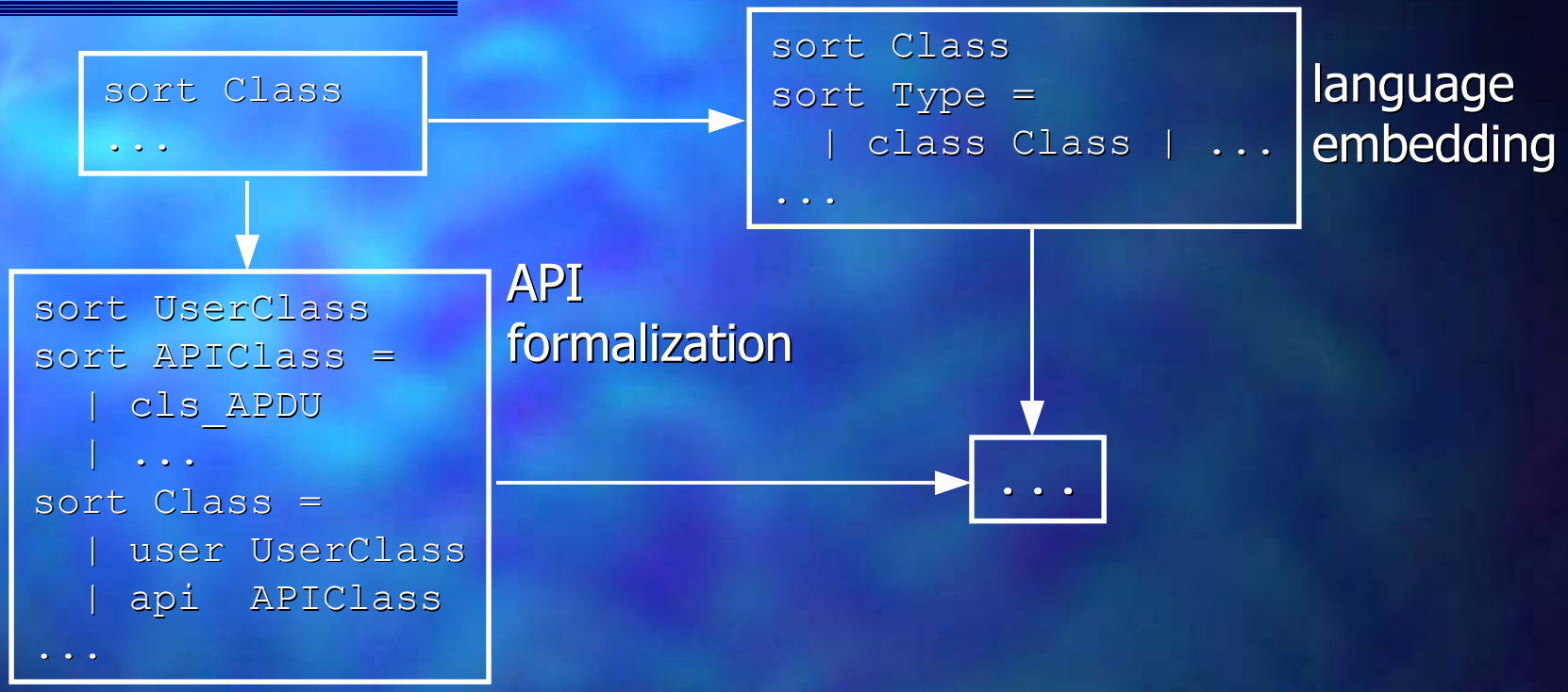
Program Representation by Complete Instantiation



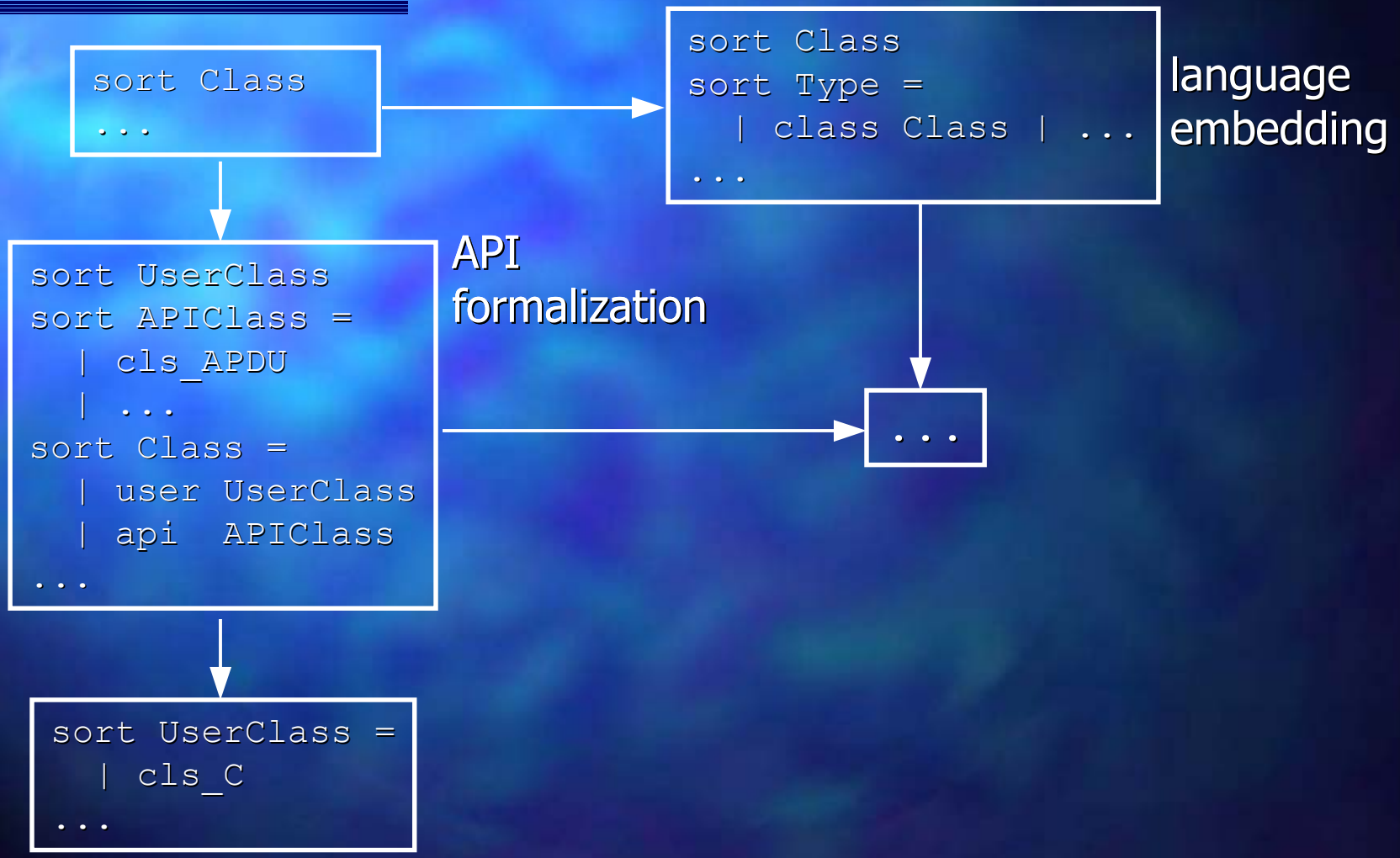
Program Representation by Complete Instantiation



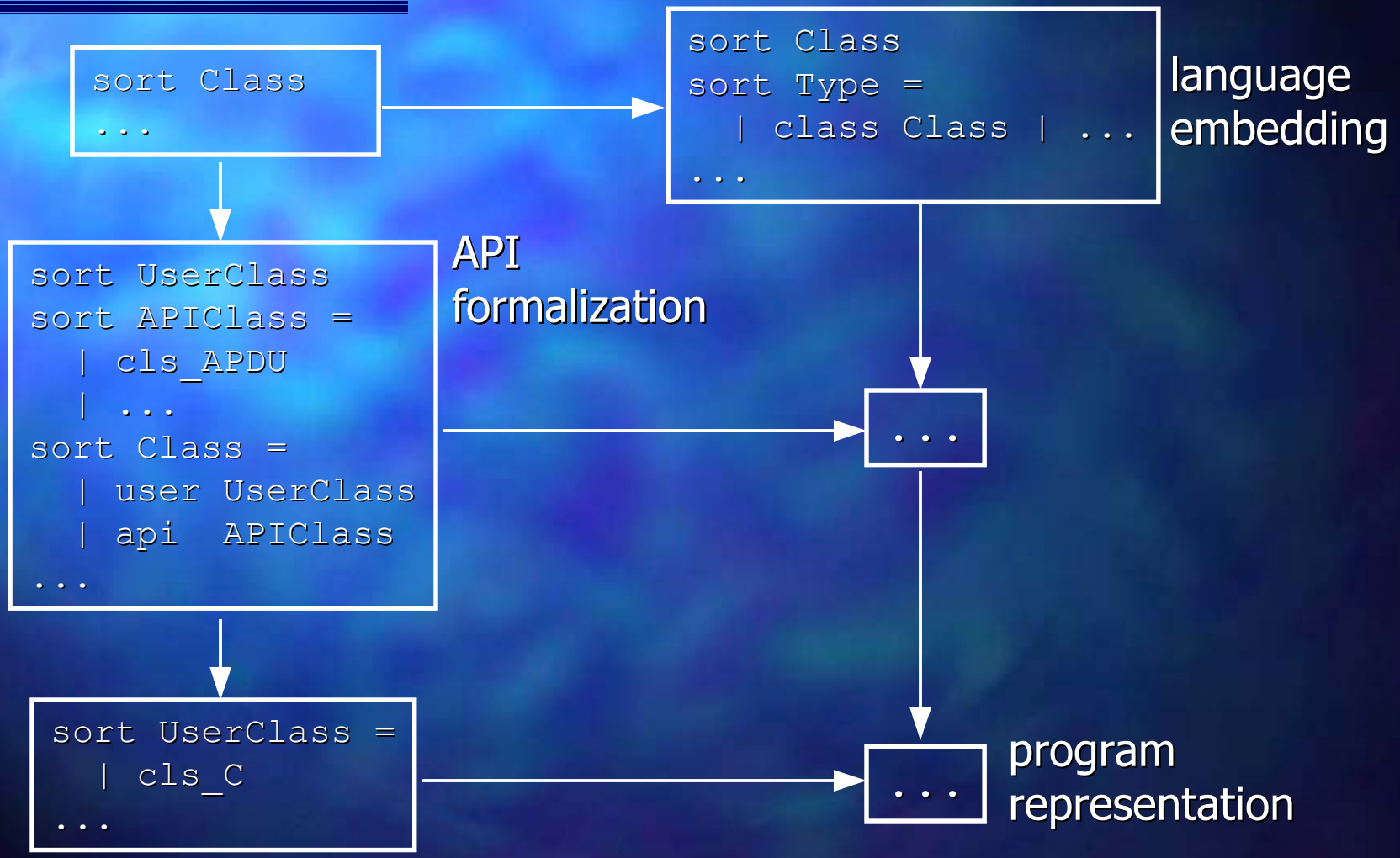
Program Representation by Complete Instantiation



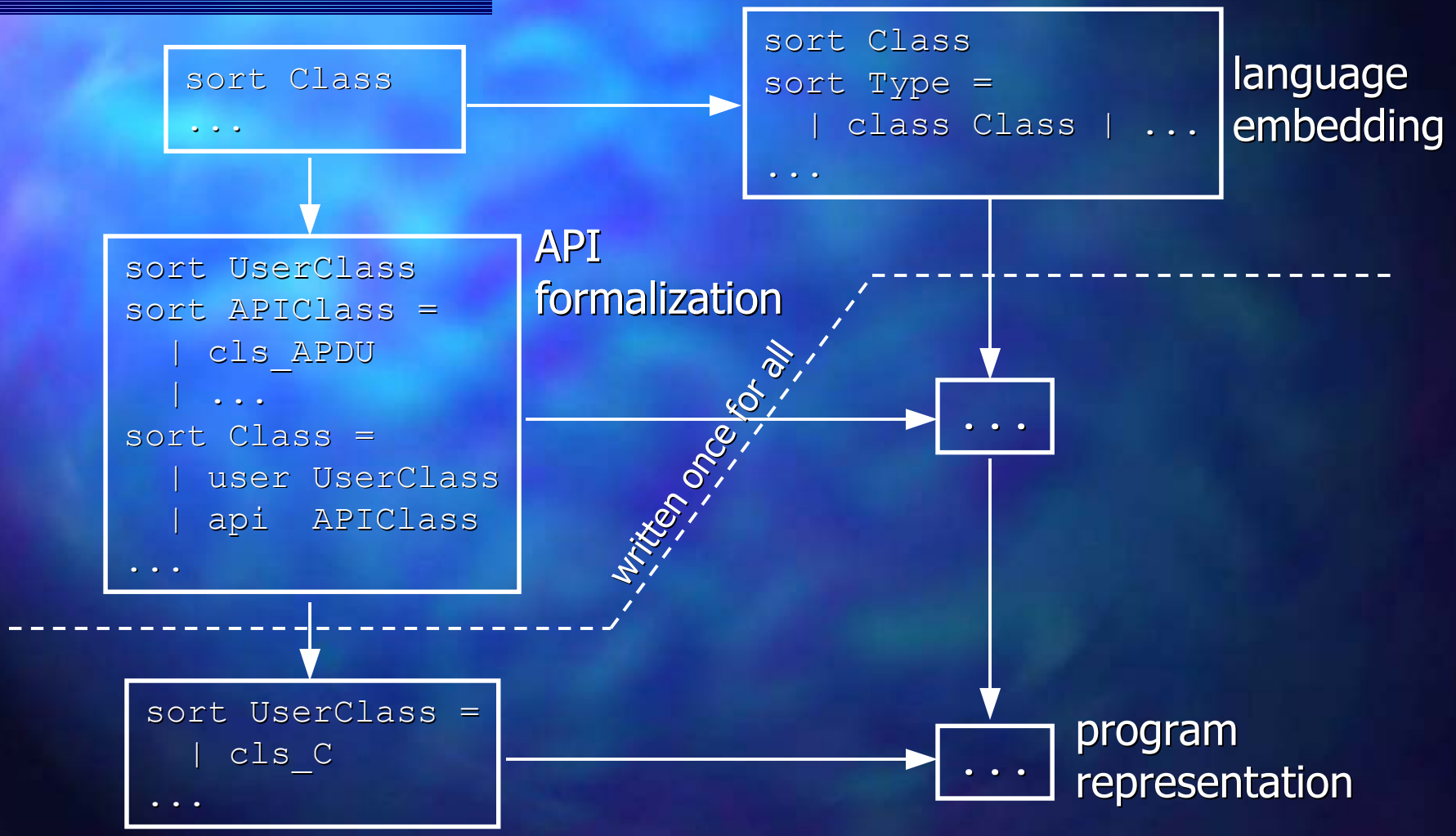
Program Representation by Complete Instantiation



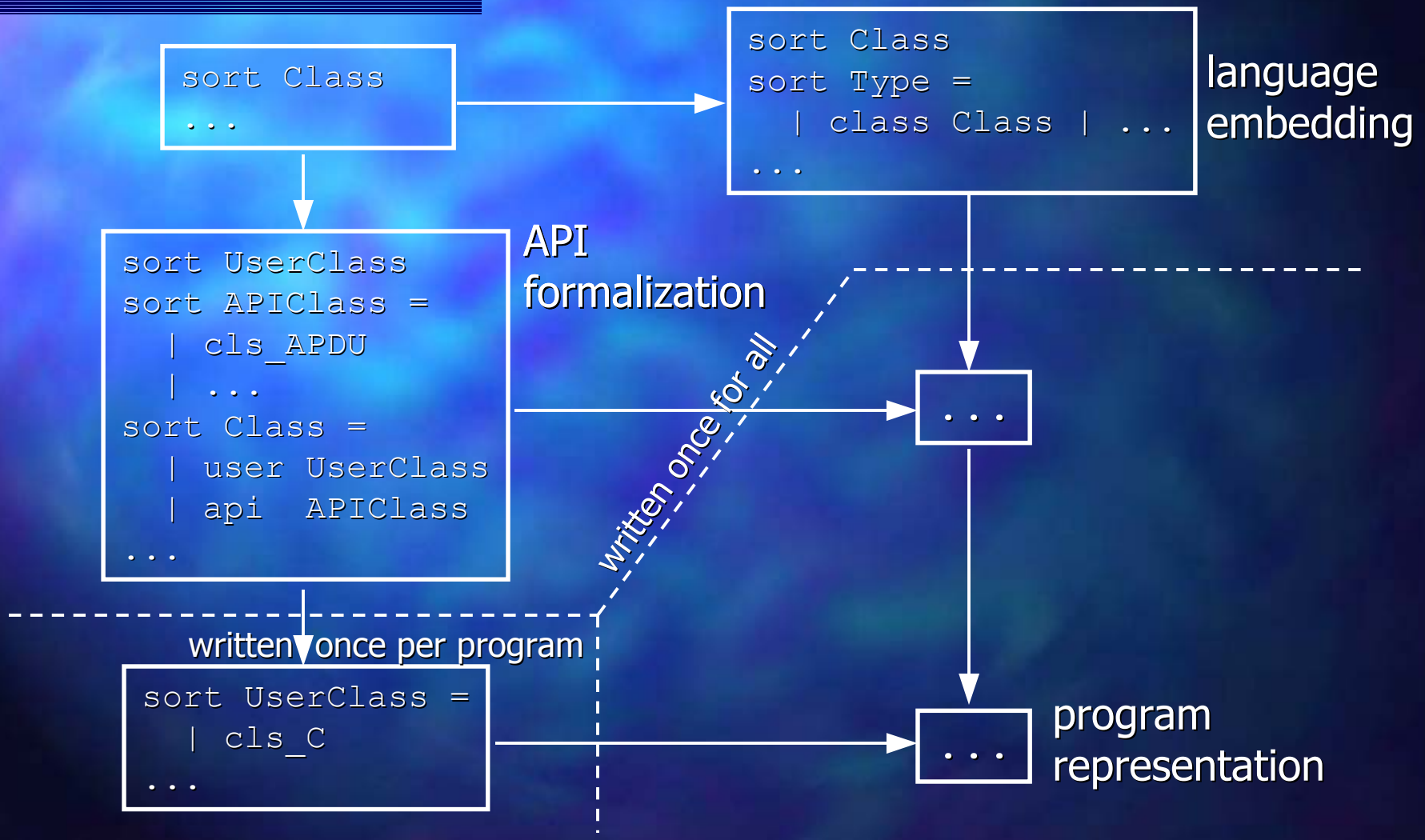
Program Representation by Complete Instantiation



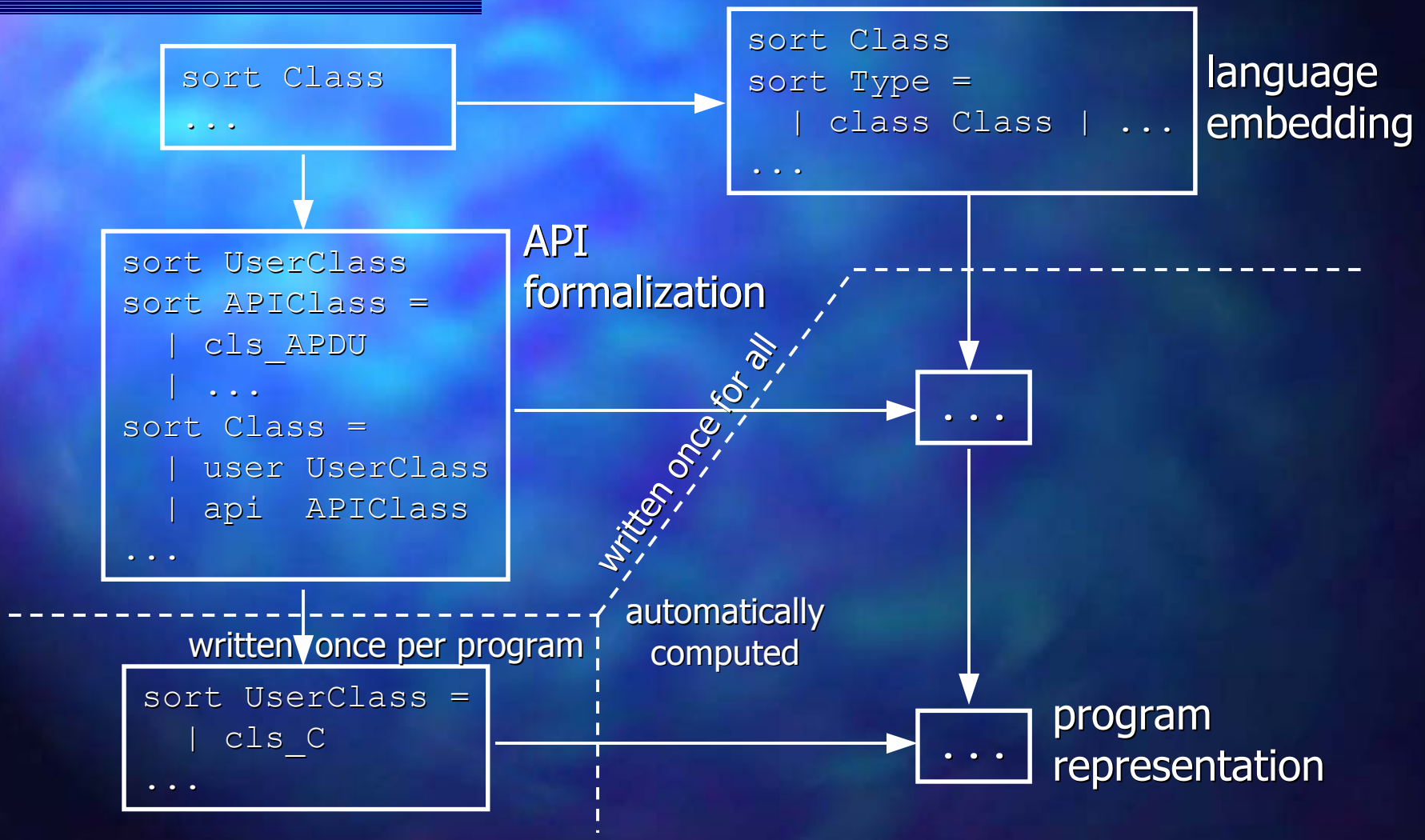
Program Representation by Complete Instantiation



Program Representation by Complete Instantiation



Program Representation by Complete Instantiation



Example

```
class C {
 void m(APDU apdu)
 {
  byte[] buf;
  buf =
    apdu.getBuffer();
 }
}
```

```
sort UserClass = | cls_C
sort UserLocalVariable =
   | locvar_apdu
   | locvar_buf
sort Method = | mth_m
...
axiom
  method_body(mth_m) =
  stat_assignment
     expr_locvar_buf
     (expr_invoke_APDU_getBuffer
        expr_locvar_apdu)
```

Summary

- Java Card
- Kestrel's work
- Applet generator
- Code generator
  - approach
  - details
  - checking correctness
  - accomplishments
- Current and future work

One-to-one Correspondence

used by the checker

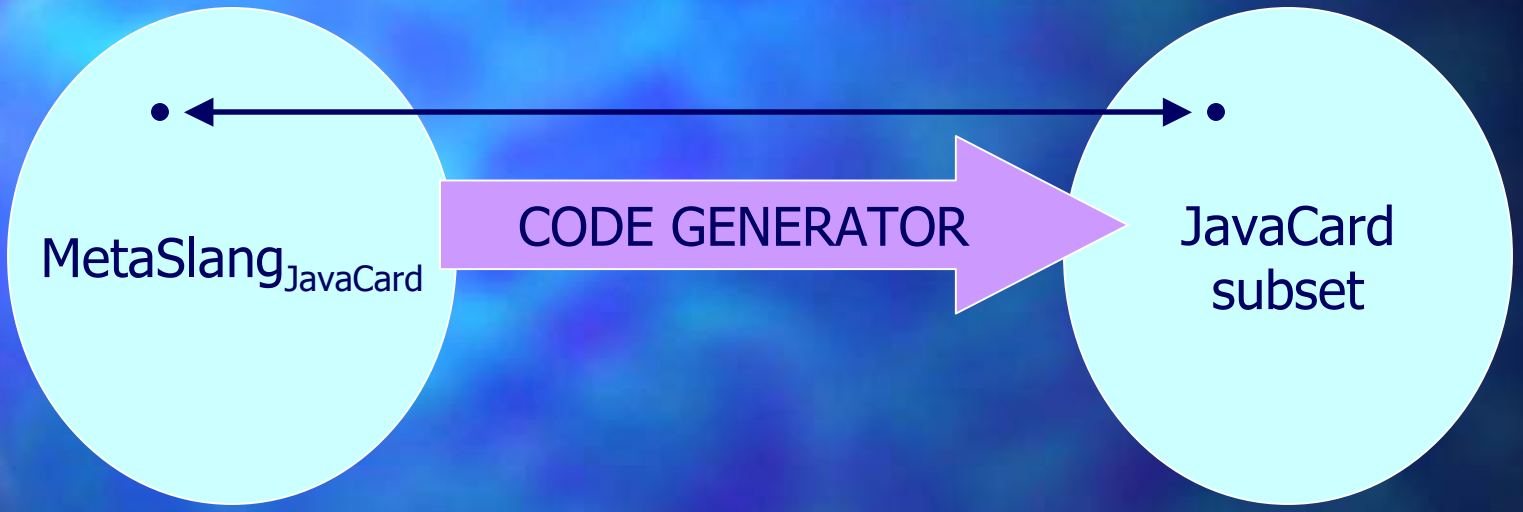
code generator checks that spec is in MetaSlang$_{JavaCard}$ and maps it to corresponding Java Card program

One-to-one Correspondence

used by the checker

code generator checks that spec is in MetaSlang$_{JavaCard}$
and maps it to corresponding Java Card program

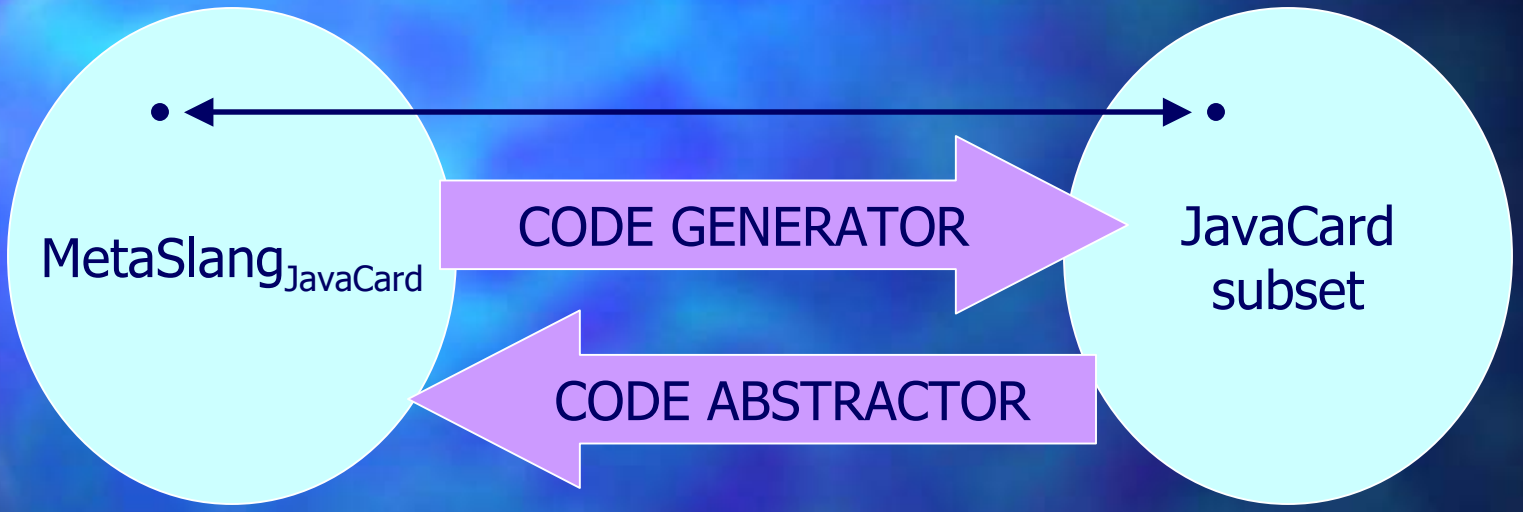
One-to-one Correspondence



used by the checker

code generator checks that spec is in MetaSlang$_{JavaCard}$
and maps it to corresponding Java Card program

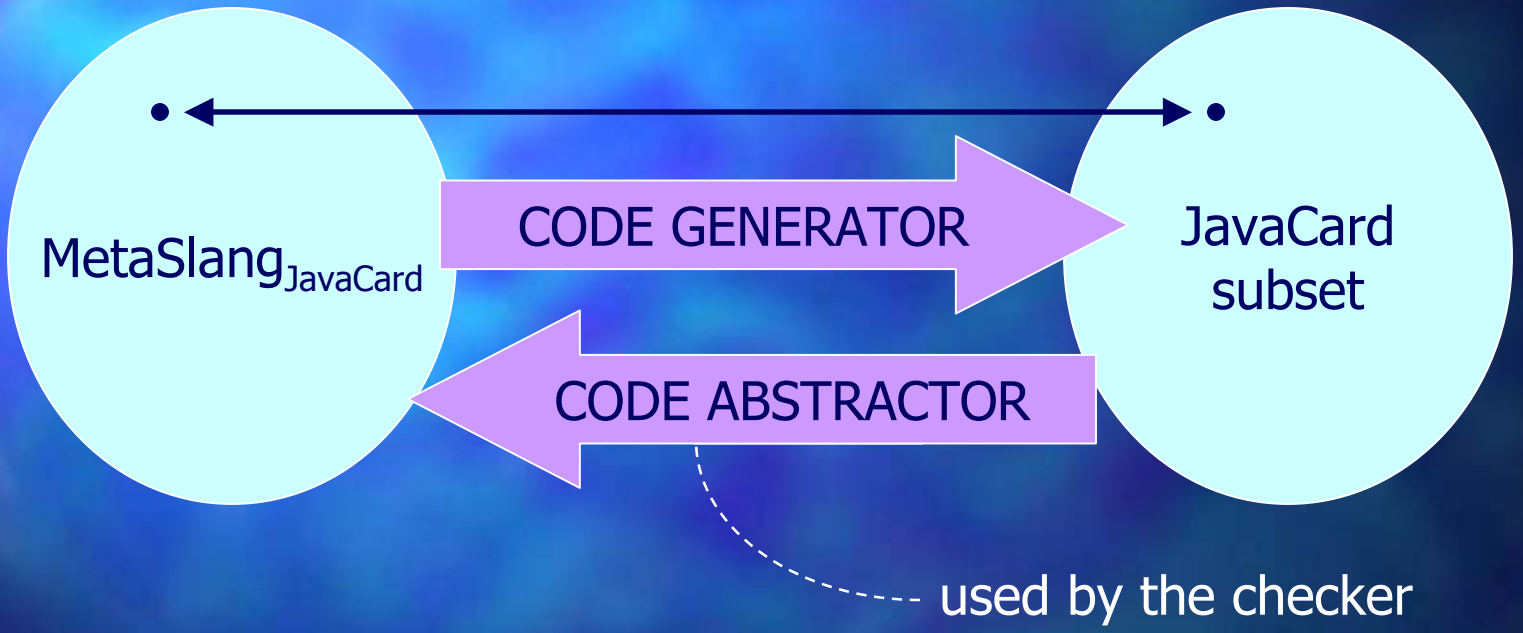
One-to-one Correspondence



used by the checker

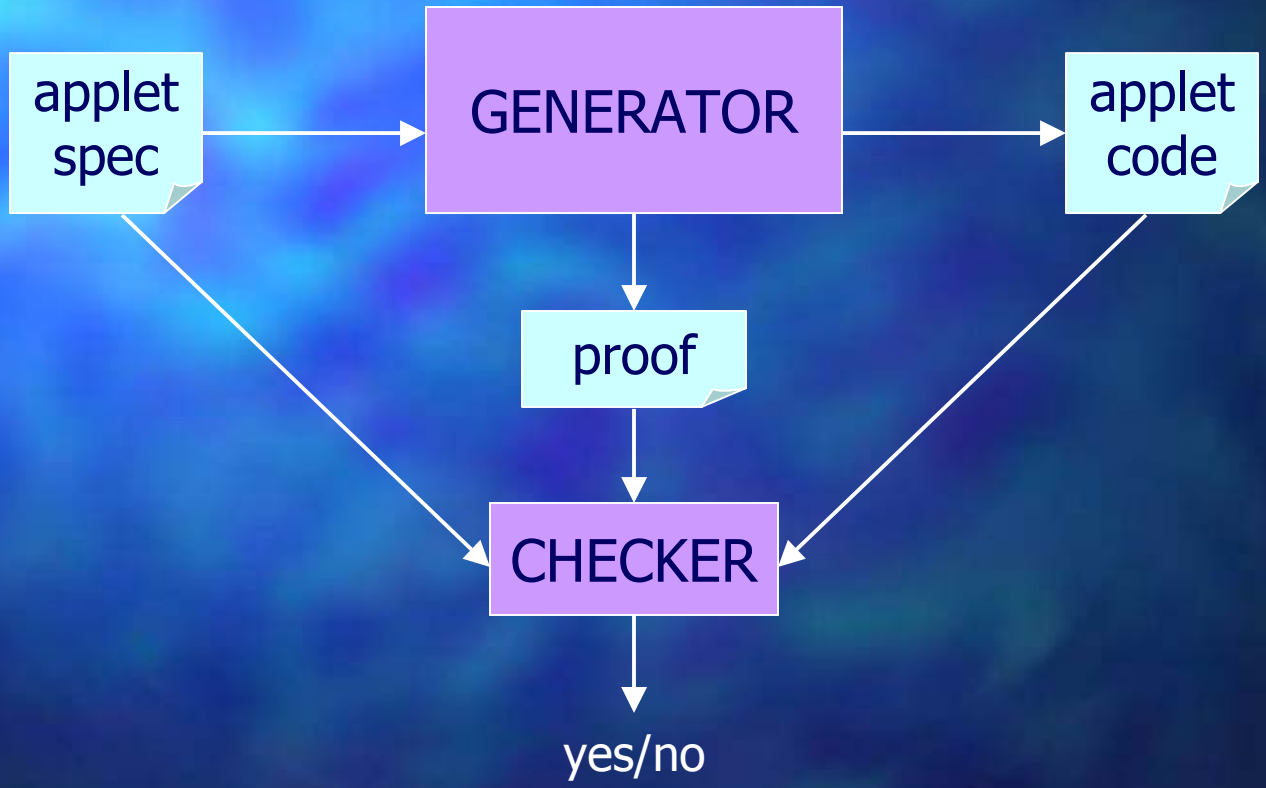
code generator checks that spec is in MetaSlang$_{JavaCard}$ and maps it to corresponding Java Card program

One-to-one Correspondence

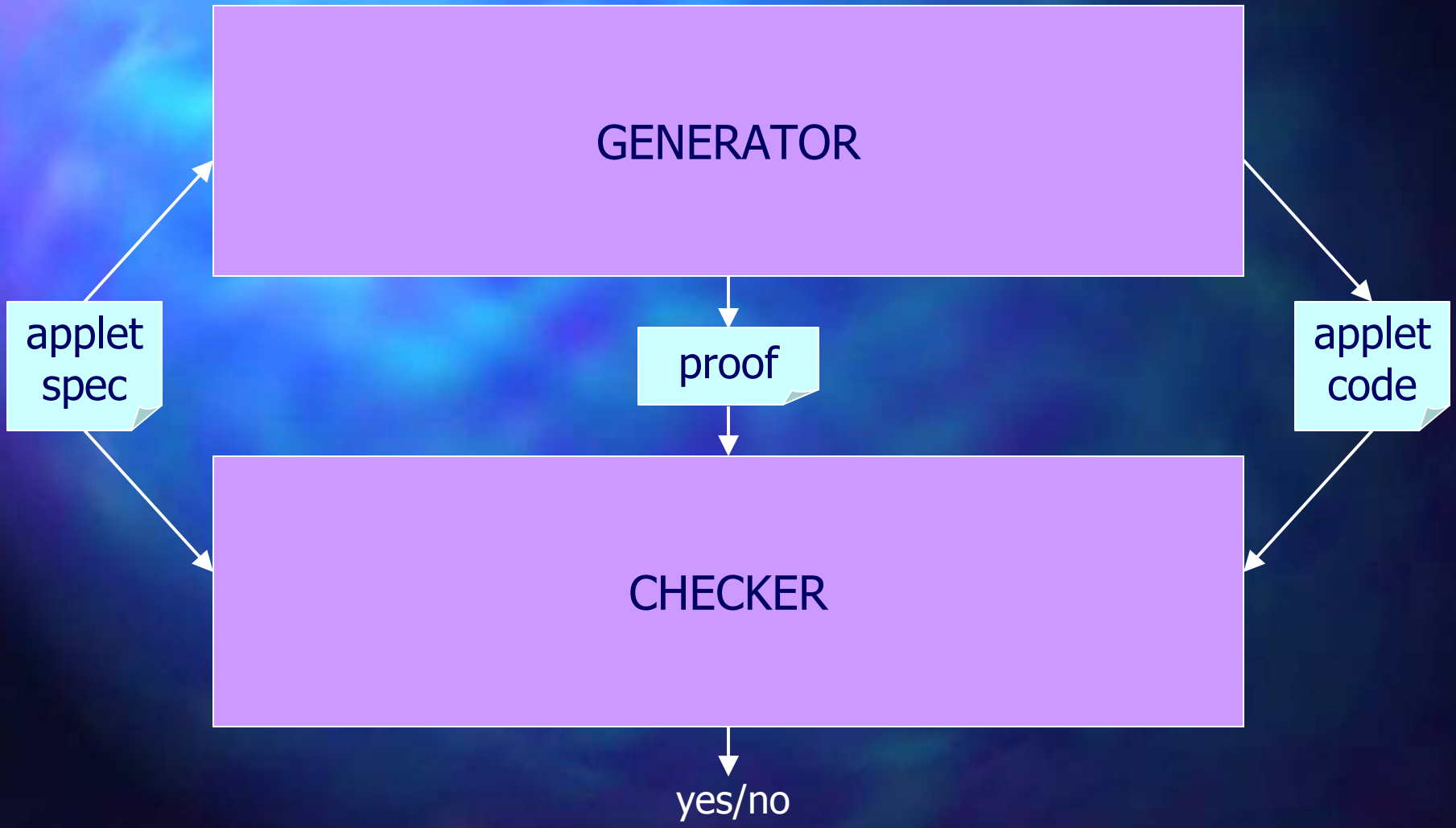


code generator checks that spec is in MetaSlang$_{JavaCard}$ and maps it to corresponding Java Card program

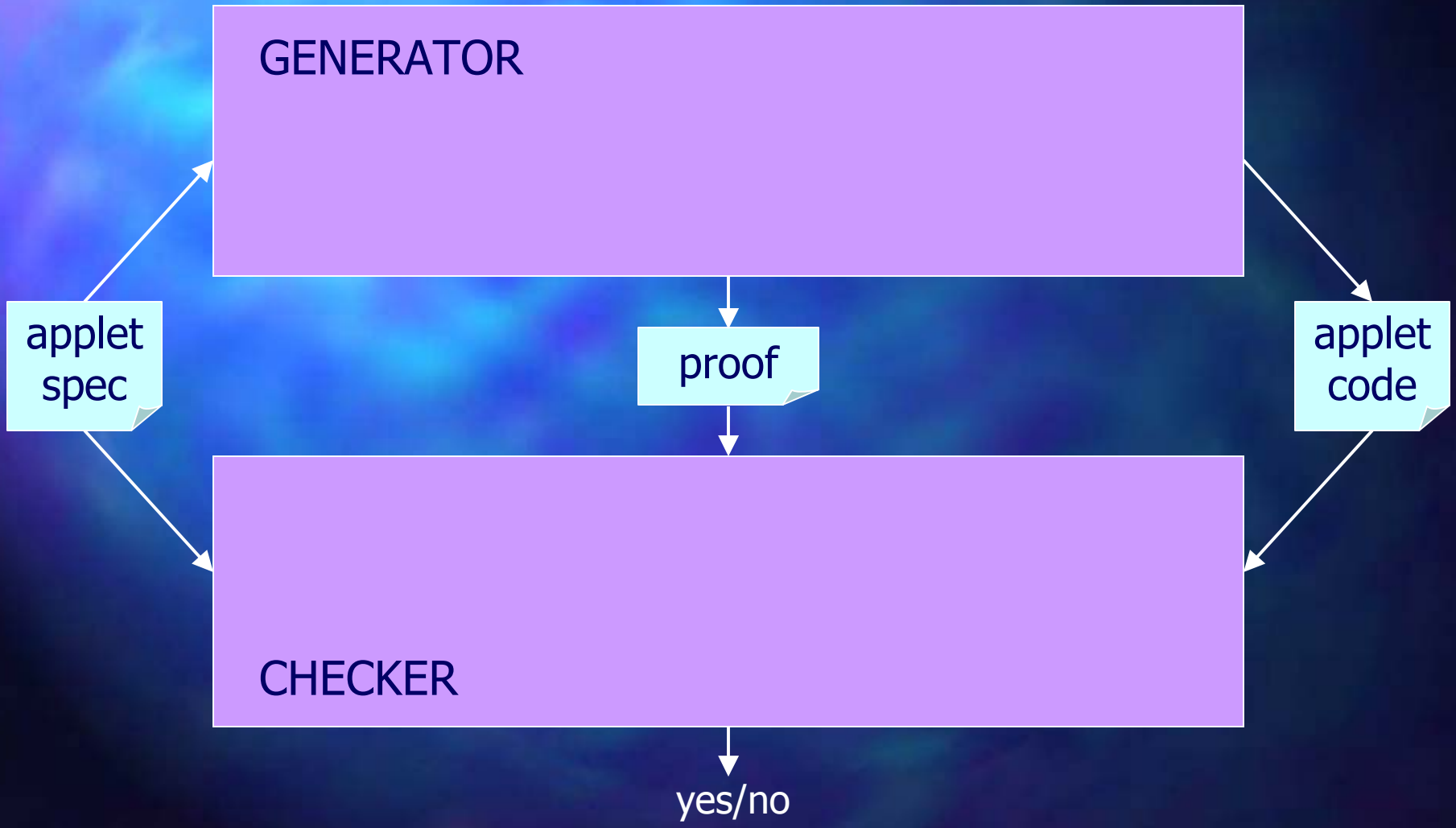
Code Abstractor



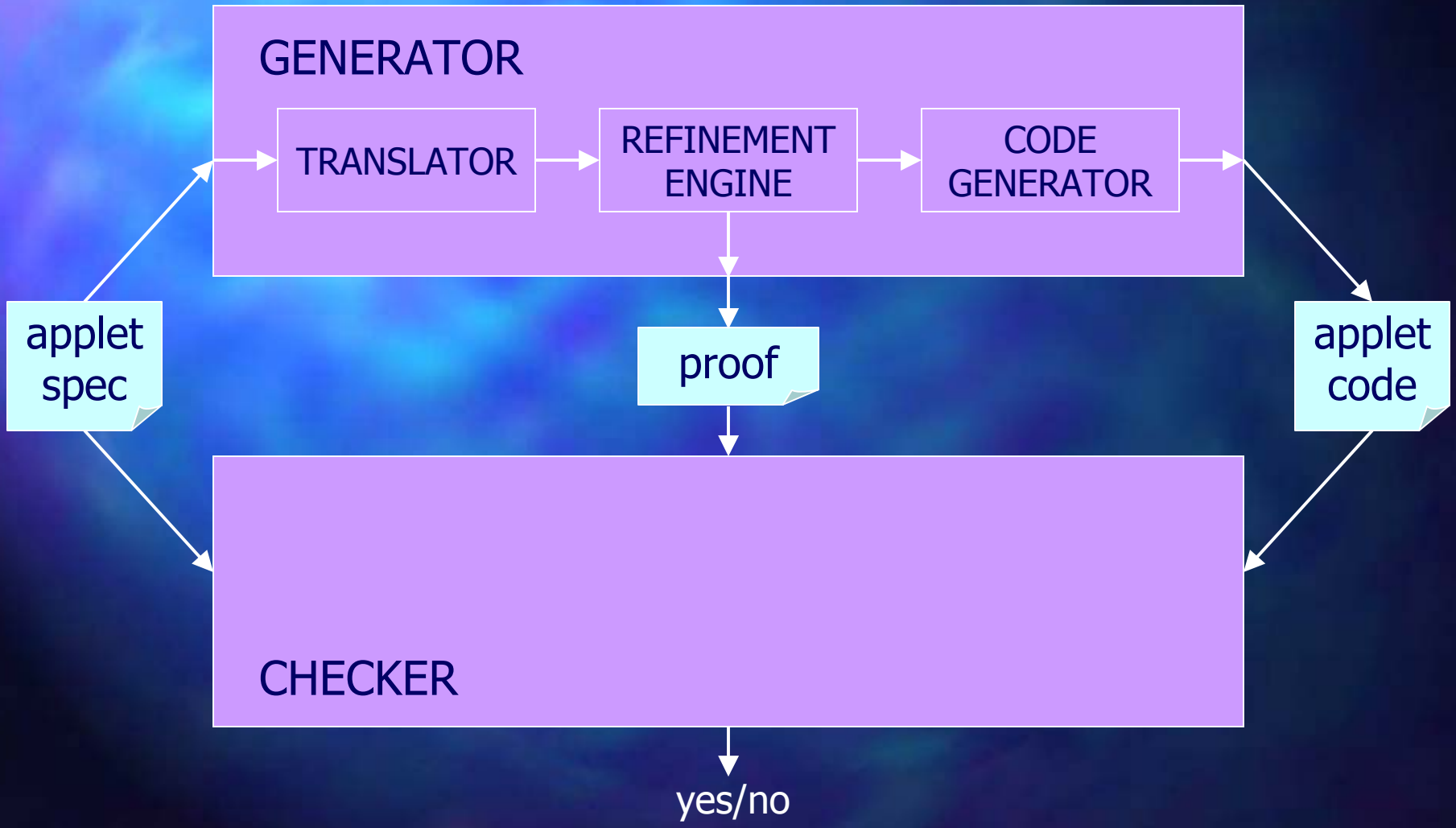
Code Abstractor



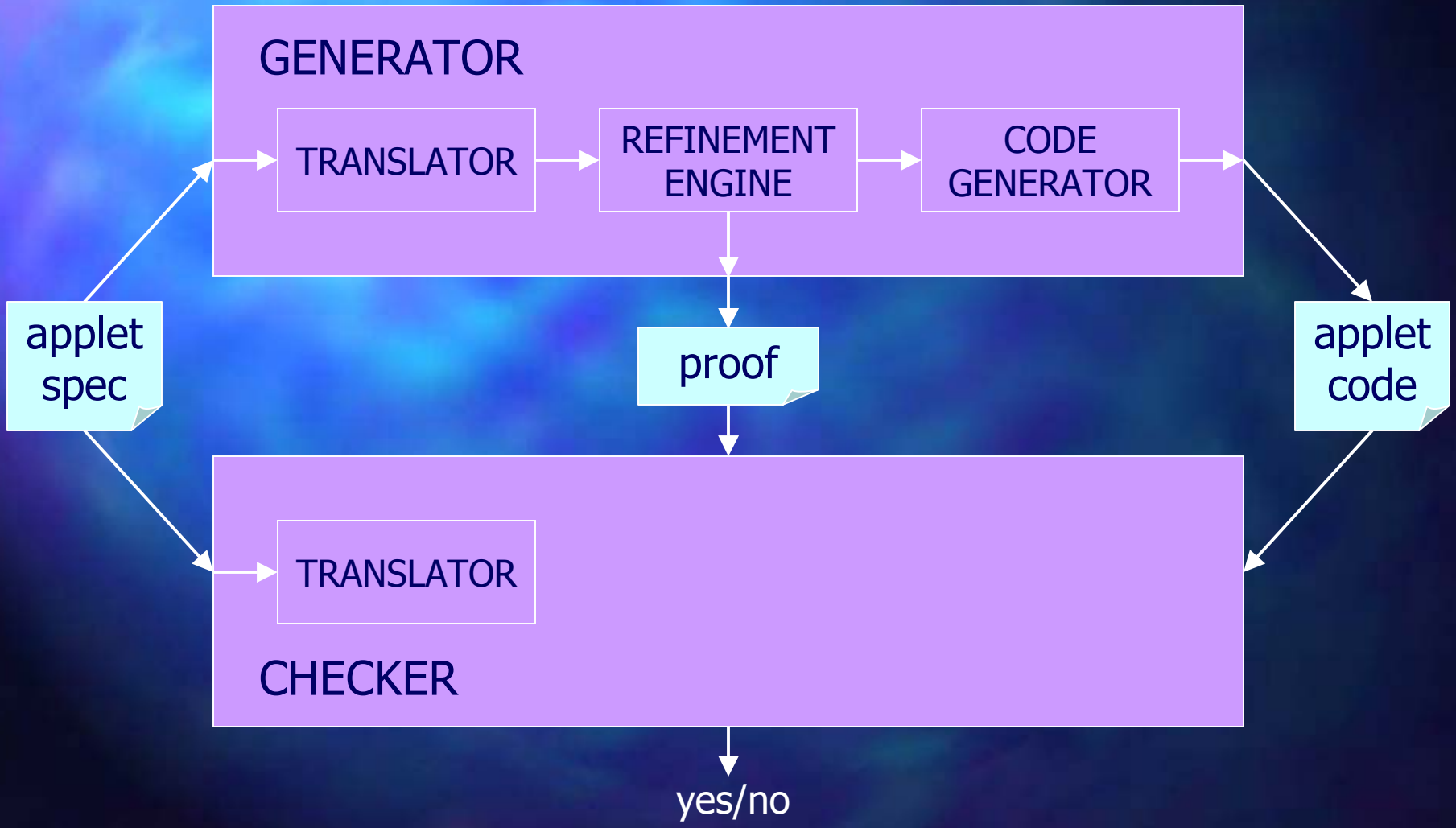
Code Abstractor



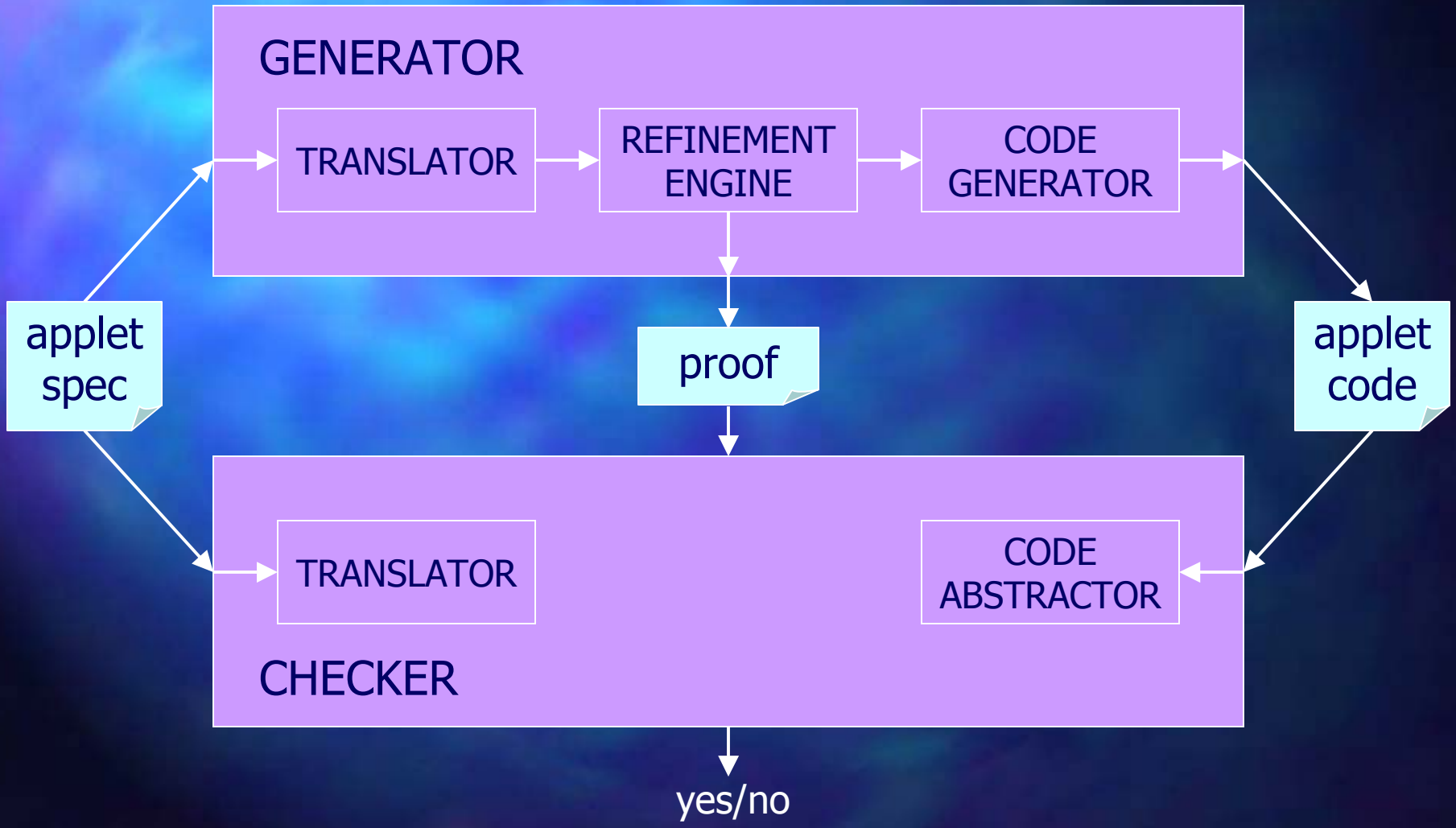
Code Abstractor



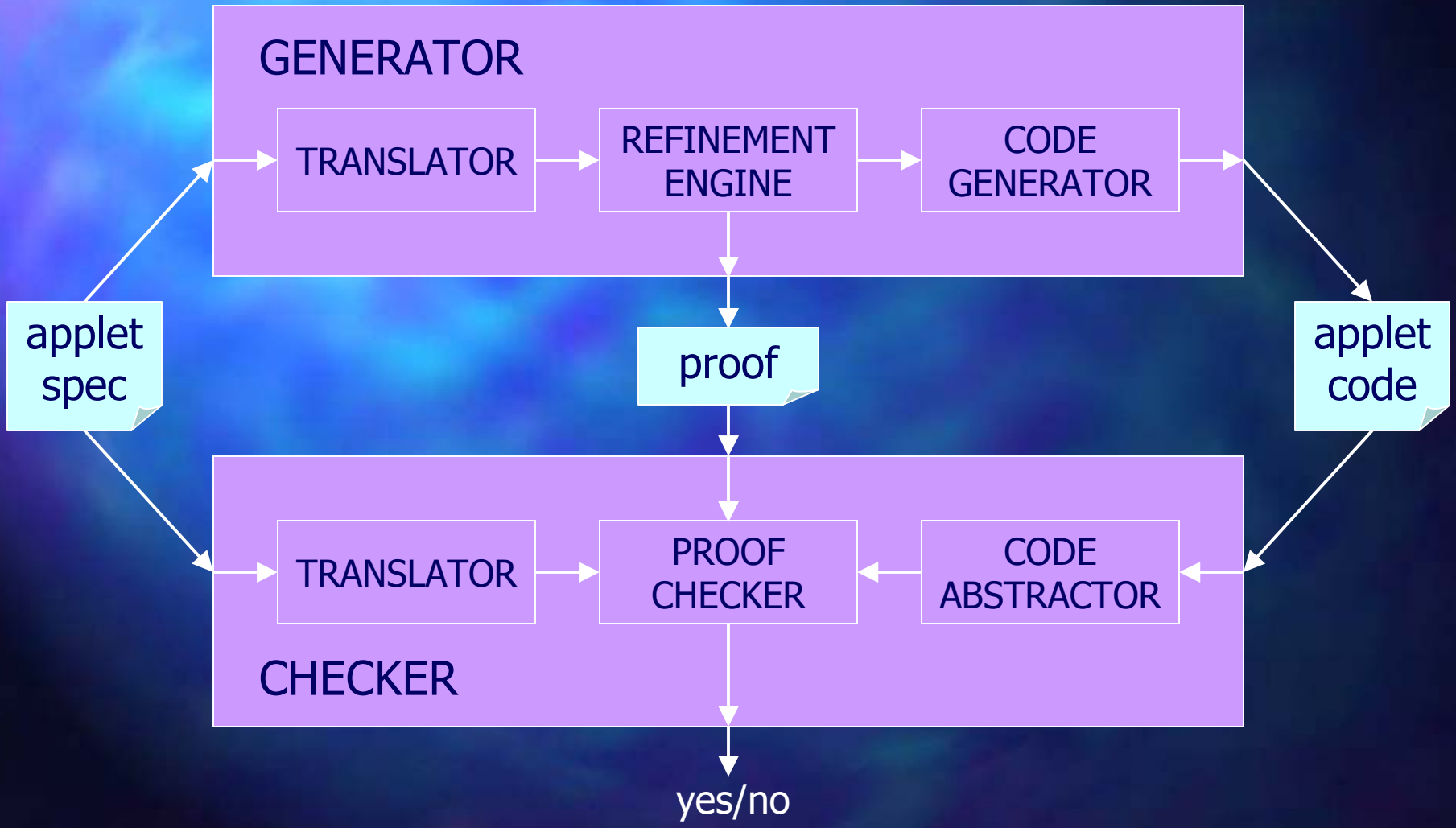
Code Abstractor



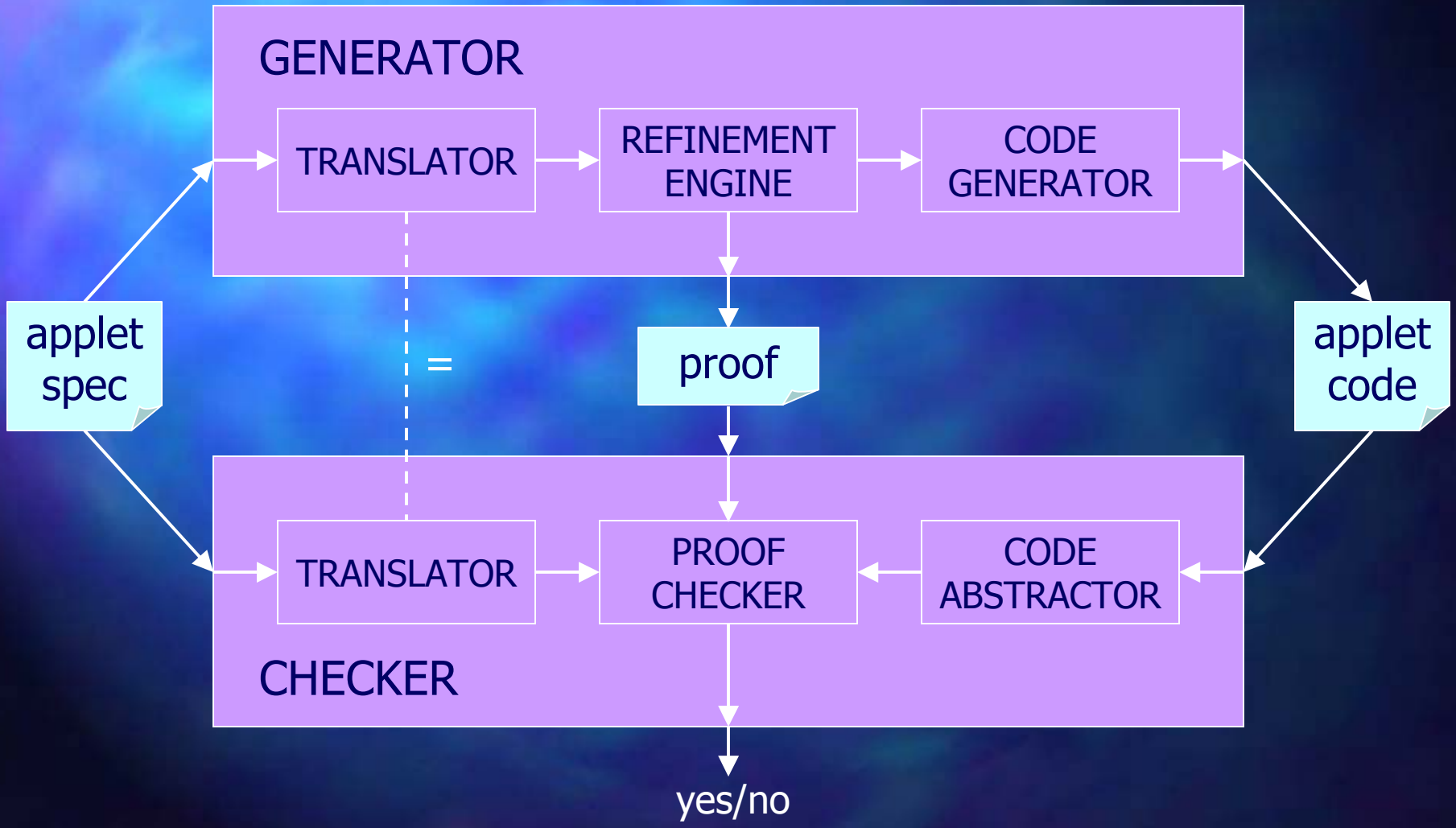
Code Abstractor



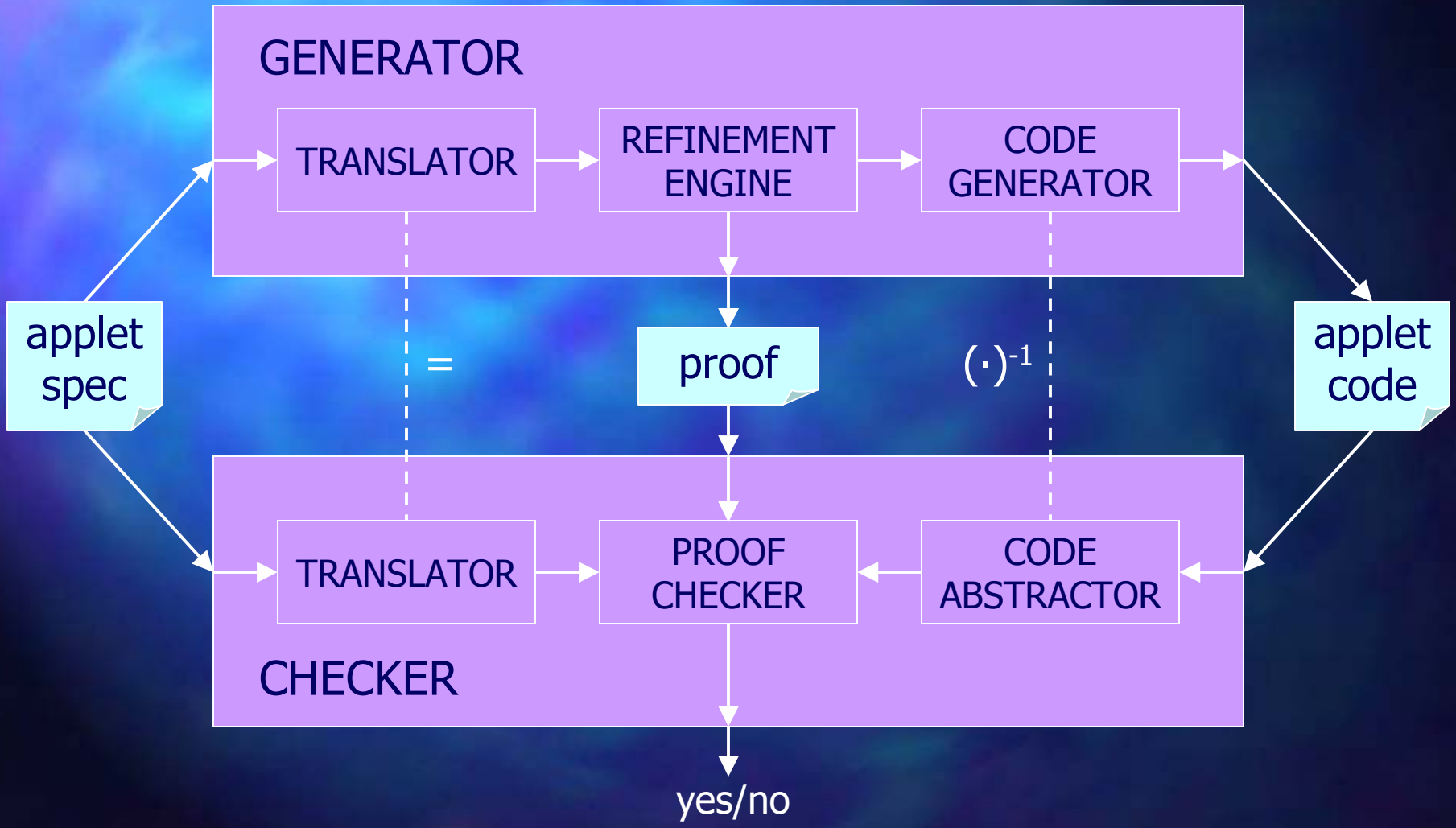
Code Abstractor



Code Abstractor



Code Abstractor



A Possible Question

If $\text{MetaSlang}_{\text{JavaCard}}$ is one-to-one with Java Card programs, why bother?

A Possible Question

If $\text{MetaSlang}_{\text{JavaCard}}$ is one-to-one with Java Card programs, why bother?

- Correctness of applet generator is expressed by a formal relationship between DSL and Java Card
- A formal relationship between two languages is expressed by bringing the languages into a common mathematical language (e.g. MetaSlang)
- So, $\text{MetaSlang}_{\text{JavaCard}}$ is needed for formal proofs

Summary

- Java Card
- Kestrel's work
- Applet generator
- Code generator
  - approach
  - details
  - checking correctness
  - accomplishments
- Current and future work

Accomplishments

- Java Card language and API specs
  - ~7K lines of MetaSlang
    - ~3K for the language
    - ~4K for the APIs
  - re-usable piece of formalized knowledge (e.g., for Java Card platform specification)
- Instantiated specs to represent PKI applet for Common Access Cards (CAC)
- Implemented code generator
- Generated PKI applet code

Summary

- Java Card
- Kestrel's work
- Applet generator
- Code generator
  - approach
  - details
  - checking correctness
  - accomplishments
- Current and future work

Current Work

- Definition of DSL
  - higher-level than Java card
  - specs shorter than corresponding Java Card programs (~ 2-5 x)
  - captures boilerplates
  - early spec error detection
- Specification of PKI applet in DSL (done)
- Implementation of translator DSL $\rightarrow$ MetaSlang

Future Work

- Design & implement refinement engine
- Integrate components (translator, refinement engine, code generator) into applet generator
- Applet checker
  - implement code abstractor
  - implement proof checker

Questions?