High-Confidence Java Card Applets and Runtime Environment

Alessandro Coglio

joint work with:

Matthias Anlauff David Cyrluk
Lindsay Errington Li-Mei Gilham
Lambert Meertens Stephen Westfold

Kestrel Institute

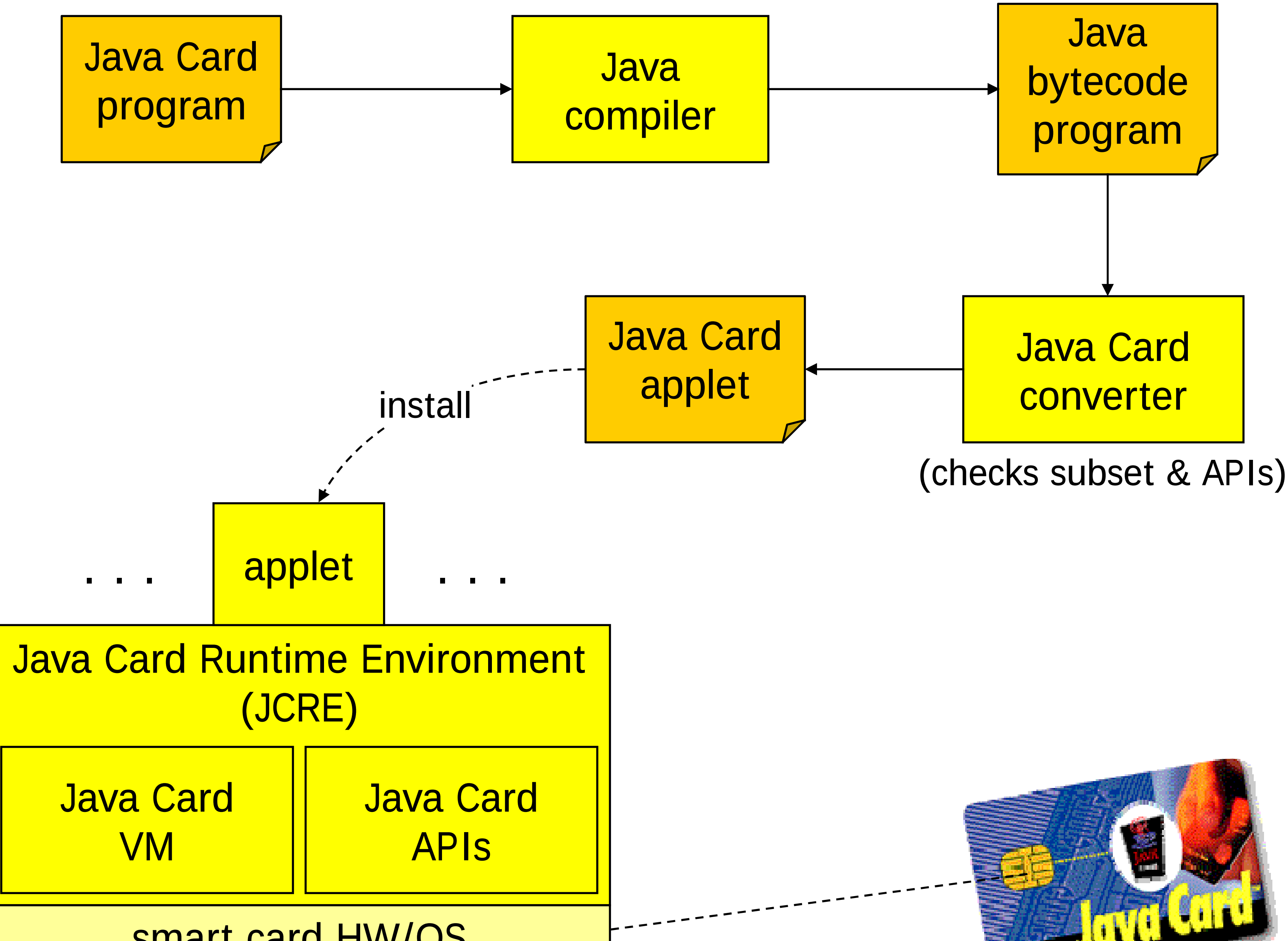
HCSS Conference, April 2006

Java Card

Java Card = Java for smart cards

- language subset
- different APIs

authentication,
banking,
telephony,
health care,
...



Java Card program

Java compiler

Java bytecode program

Java Card converter

(checks subset & APIs)

Java Card applet

install

. . .

applet

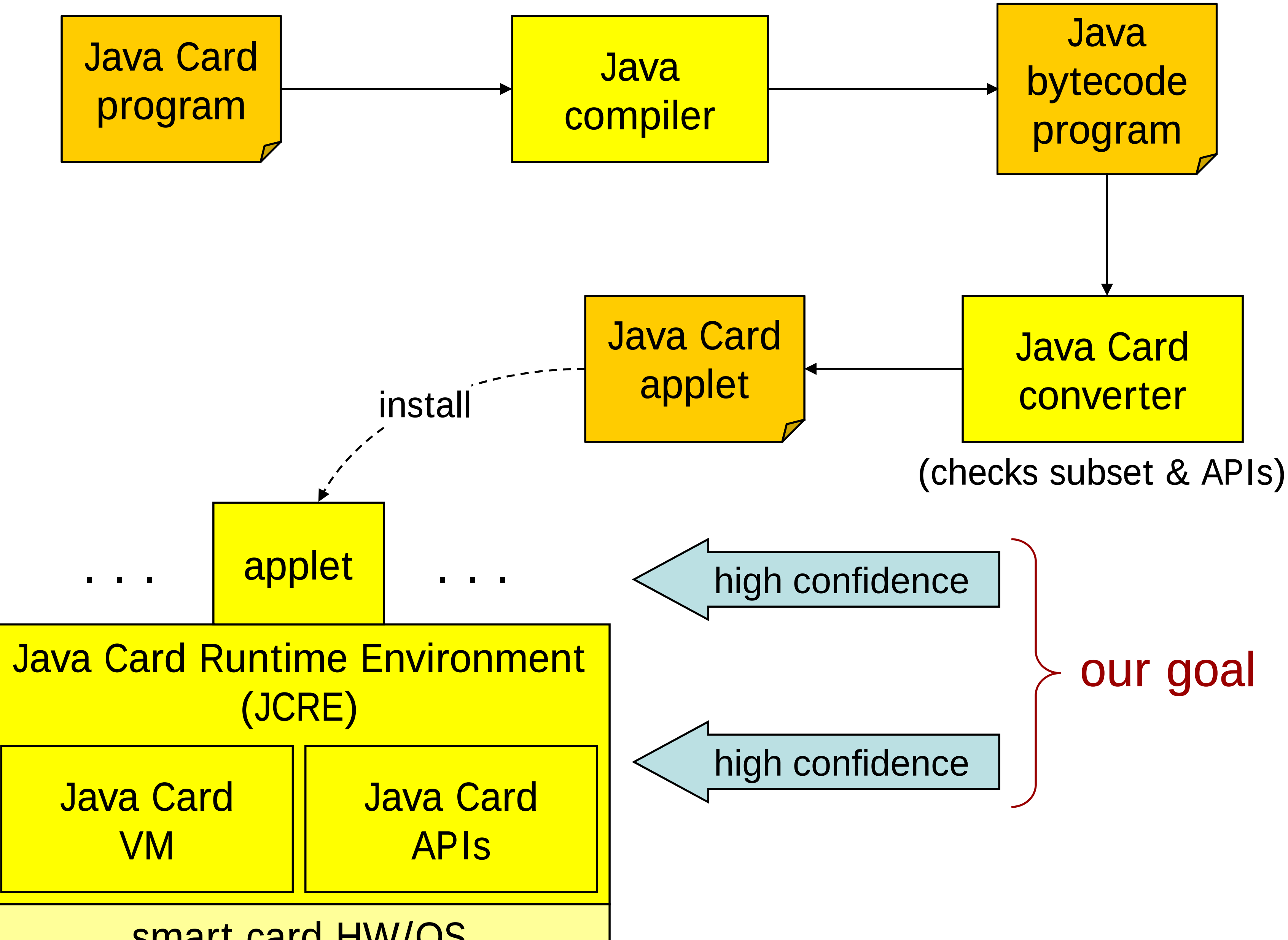
. . .

Java Card Runtime Environment (JCRE)

Java Card VM

Java Card APIs

smart card HW/OS



Java Card program

Java compiler

Java bytecode program

Java Card converter

(checks subset & APIs)

Java Card applet

install

. . . applet . . .

Java Card Runtime Environment (JCRE)

Java Card VM

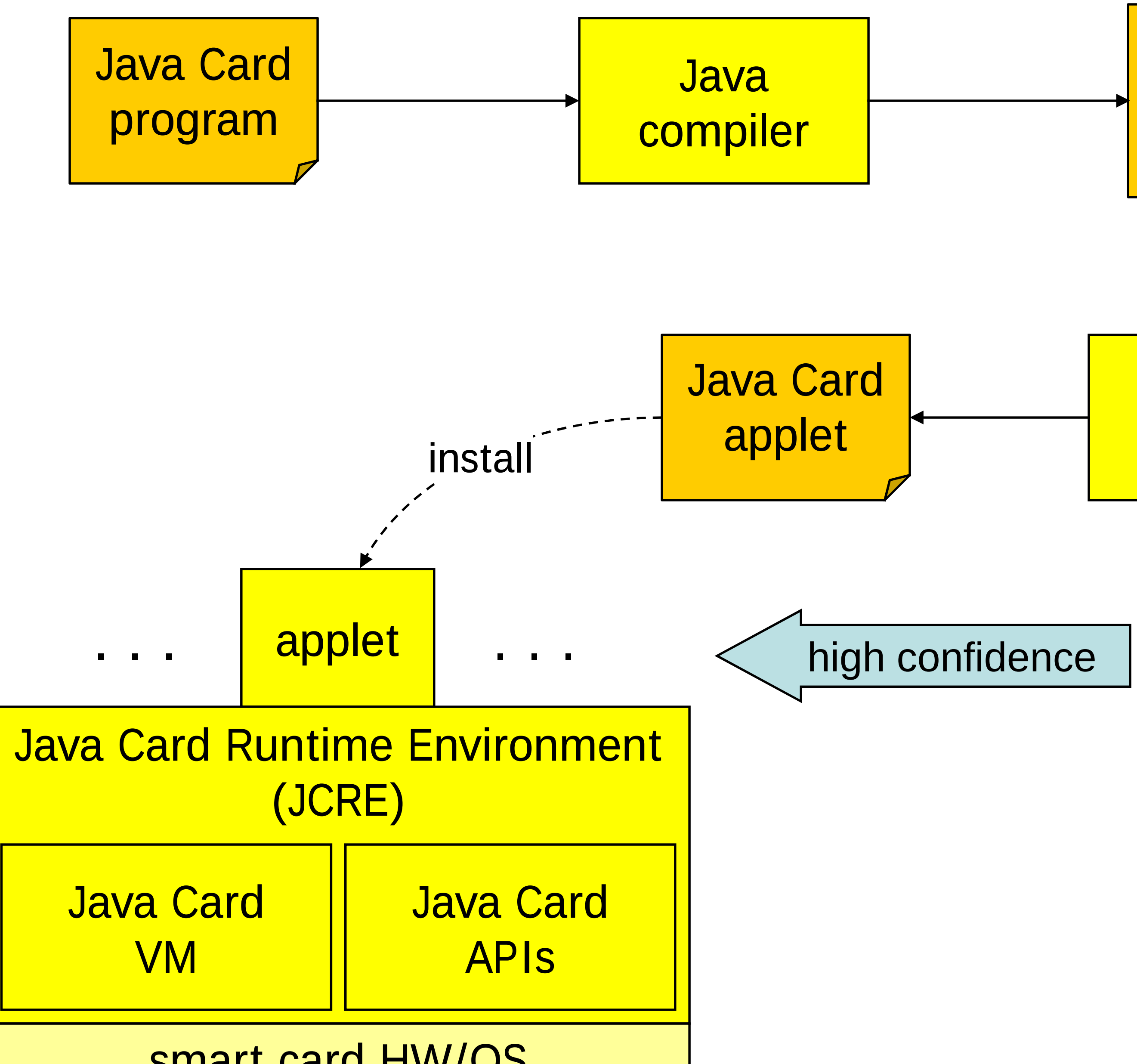
Java Card APIs

smart card HW/OS

high confidence

high confidence

our goal



Java Card program

Java compiler

Java bytecode program

Java Card converter

Java Card applet

install

applet

. . .

. . .

high confidence

Java Card Runtime Environment (JCRE)

Java Card VM

Java Card APIs

smart card HW/OS

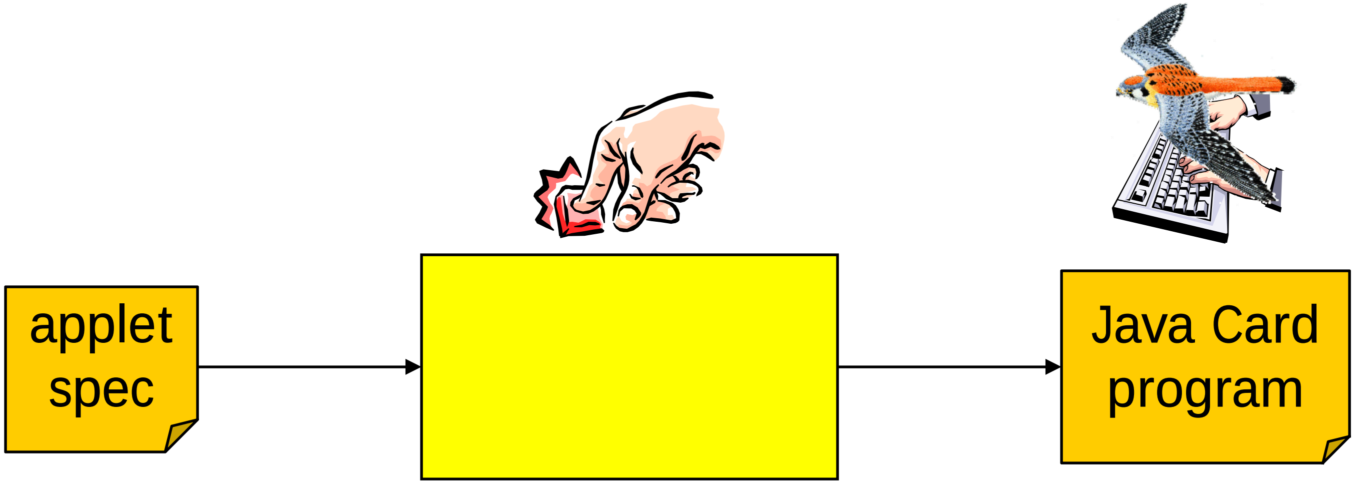
Java Card program

typically
written
by hand

Java Card program

somewhat
low-level

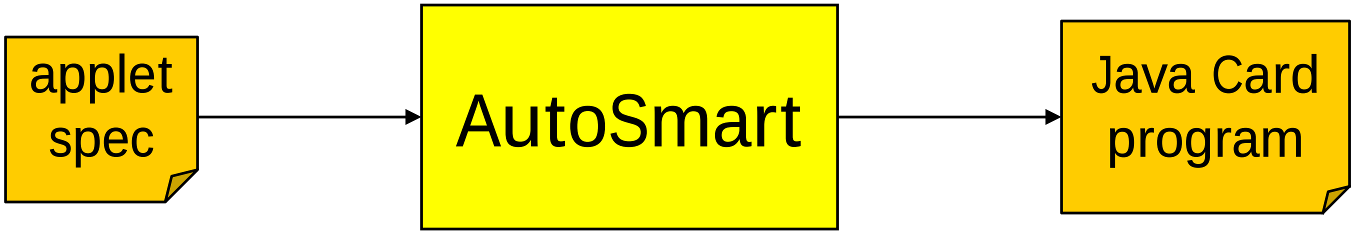
⇒ error-prone



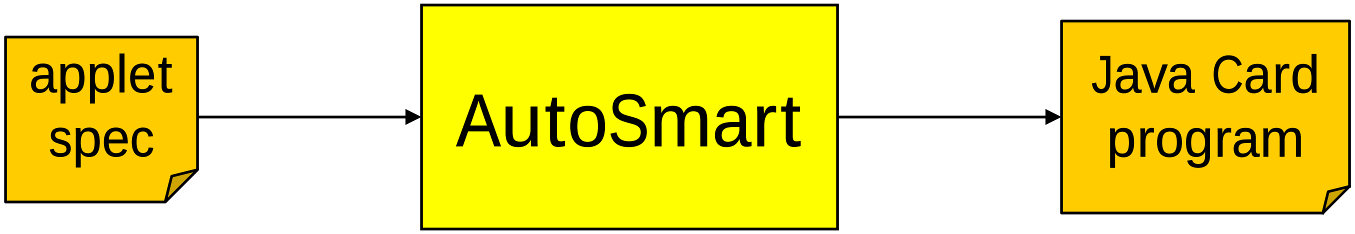
applet spec

Java Card program

automatic generator of smart card applets



- high confidence
- productivity



applet spec

AutoSmart

Java Card program

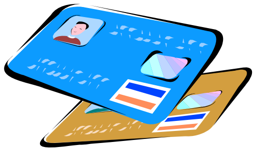
written in SmartSlang

SmartSlang

(smart card specification language)

SmartSlang
(smart card specification language)

domain-specific language



domain = smart cards

i.e. constructs specialized to smart card applications

evolving design

working version done,
additional constructs planned

precise semantics

in terms of state machines

$$applet : Command \times State \to Response \times State$$

but no formal background necessary

SmartSlang example: PKI

SmartSlang

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }
```

ord

built-in crypto key types

include size

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];
```

array type

includes allowed size(s)

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
```

word name parameter(s)

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
```

eyword to
nd response

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
```

built-in crypto function

static check: argument sizes match

also checked in the presence of variables
(via automated theorem proving)

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};
```

declarative encoding of command
cl. parameters) as low-level APDU bytes

semantics: decoding/dispatching
and range checking at run time

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
```

ord

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
```

size (in bytes)

determines strength

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
```

name (multiple PINs allowed)

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
axtries = 3;
```

efore blocking

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
axtries = 3;
rotected = {privSignDecrypt, ...};
```

explicit rule of security policy

commands that require
PIN p to be verified
before they can be used

semantics: state of PIN p checked
at run time, error response if not verified

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
axtries = 3;
rotected = {privSignDecrypt, ...};
erify apdu {0x80,0x20,0,0};
```

declarative APDU encoding of
command to verify PIN p

header only, body implicit (= supplied PIN value)

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
axtries = 3;
rotected = {privSignDecrypt, ...};
erify apdu {0x80,0x20,0,0};
.. }
```

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
axtries = 3;
rotected = {privSignDecrypt, ...};
erify apdu {0x80,0x20,0,0};
.. }

ure command generateKeyPair () {
enerate(privKey,pubKey);
pdu {0x84,0x46,0,0,{},0};
```

built-in key generation statement

SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
axtries = 3;
rotected = {privSignDecrypt, ...};
erify apdu {0x80,0x20,0,0};
.. }
```

modifier for Global Platform secure channels

```
ure command generateKeyPair () {
enerate(privKey,pubKey);
pdu {0x84,0x46,0,0,{},0};
```

channel must be established
(involving authentication)
for command to be used

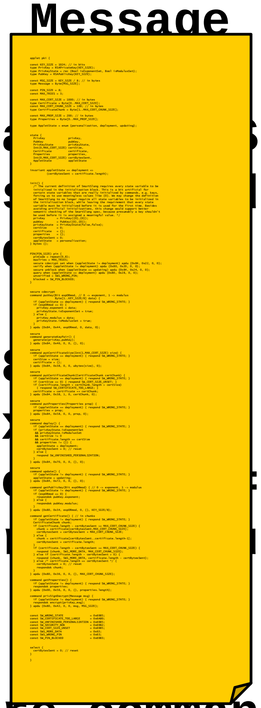
SmartSlang example: PKI

```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

mand privSignDecrypt (Message msg) {
espondok decrypt(key,msg);
pdu {0x80,0x42,0,0,msg,1024/8};

(8) p {
axtries = 3;
rotected = {privSignDecrypt, ...};
erify apdu {0x80,0x20,0,0};
.. }

ure command generateKeyPair () {
enerate(privKey,pubKey);
pdu {0x84,0x46,0,0,{},0};
```

PKI example

example PKI

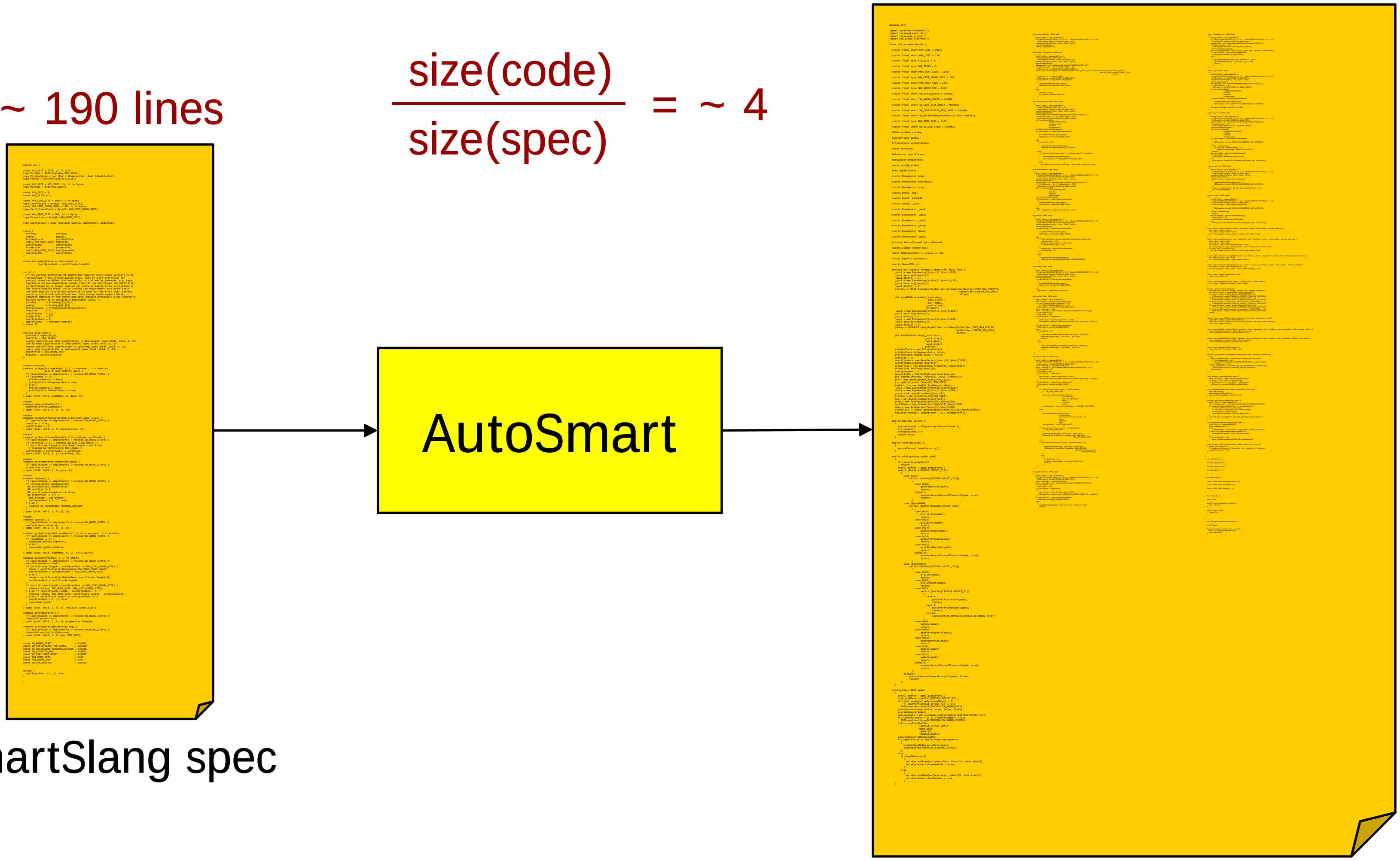
```
te { RSAPrivateKey(1024) privKey,
     RSAPublicKey(1024)  pubKey, ... }

e Message = Byte[1024/8];

ma        ignDecrypt (Message msg) {
es        ecrypt(key,msg);
pd        0x42,0,0,msg,1024/8};

(8
ax         3;
re       = {privSignDecrypt, ...};
er       u {0x80,0x20,0,0};
.

ure command generateKeyPair () {
enerate(privKey,pubKey);
pdu {0x84,0x46,0,0,{},0};
```



```
applet pki {

const KEY_SIZE = 1024; // in bits
type PrivKey = RSAPrivateKey(KEY_SIZE);
type PrivKeyState = rec {Bool isExponentSet, Bool isModulusSet};
type PubKey = RSAPublicKey(KEY_SIZE);

const MSG_SIZE = KEY_SIZE / 8; // in bytes
type Message = Byte[MSG_SIZE];

const PIN_SIZE = 8;
const MAX_TRIES = 3;

const MAX_CERT_SIZE = 1000; // in bytes
type Certificate = Byte[0..MAX_CERT_SIZE];
const MAX_CERT_CHUNK_SIZE = 100; // in bytes
type CertificateChunk = Byte[1..MAX_CERT_CHUNK_SIZE];

const MAX_PROP_SIZE = 200; // in bytes
type Properties = Byte[0..MAX_PROP_SIZE];

type AppletState = enum {personalization, deployment, updating};


state {
  PrivKey             privKey,
  PubKey              pubKey,
  PrivKeyState        privKeyState,
  Int(0,MAX_CERT_SIZE) certSize,
  Certificate         certificate,
  Properties          properties,
  Int(0,MAX_CERT_SIZE) certBytesSent,
  AppletState         appletState
}

invariant appletState == deployment =>
          (certBytesSent < certificate.length);


init() {
  /* The current definition of SmartSlang requires every state variable to be
  initialized in the initialization block. This is a bit artificial for
  certain state variables that are really initialized by commands, e.g. keys,
  forcing us to use meaningless values like {0}. We may change the definition
  of SmartSlang to no longer require all state variables to be initialized in
  the initialization block, while leaving the requirement that every state
  variable must be initialized before it is used for the first time. Besides
  avoiding artificial initializations, this change would support better
  semantic checking of the SmartSlang spec, because presumably a key shouldn't
  be used before it is assigned a meaningful value. */
  privKey       = PrivKey({0},{0});
  pubKey        = PubKey({0},{0});
  privKeyState  = PrivKeyState(false,false);
  certSize      = 0;
  certificate   = {};
  properties    = {};
  certBytesSent = 0;
  appletState   = personalization;
} bytes {};


PIN(PIN_SIZE) pin {
  pinCode = repeat(0,8);
  maxTries = MAX_TRIES;
  secure cdecrypt set when (appletState != deployment) apdu {0x84, 0x22, 0, 0};
  verify when (appletState != deployment) apdu {0x80, 0x20, 0, 0};
  secure unblock when (appletState == updating) apdu {0x84, 0x24, 0, 0};
  query when (appletState == deployment) apdu {0x80, 0x28, 0, 0};
  unverified = SW1_WRONG_PIN;
  blocked = SW_PIN_BLOCKED;
}


secure cdecrypt
command putKey(Bit expORmod, // 0 -> exponent, 1 -> modulus
               Byte[1..KEY_SIZE/8] data) {
  if (appletState == deployment) { respond SW_WRONG_STATE; }
  if (expORmod == 0) {
    privKey.exponent = data;
    privKeyState.isExponentSet = true;
  } else {
    privKey.modulus = data;
    privKeyState.isModulusSet = true;
  }
} apdu {0x84, 0x44, expORmod, 0, data, 0};

secure
command generateKeyPair() {
  generate(privKey,pubKey);
} apdu {0x84, 0x46, 0, 0, {}, 0};

secure
command putCertificateSize(Int(1,MAX_CERT_SIZE) size) {
  if (appletState == deployment) { respond SW_WRONG_STATE; }
  certSize = size;
  certificate = {};
} apdu {0x84, 0x38, 0, 0, ubytes(size), 0};

secure
command putCertificateChunk(CertificateChunk certChunk) {
  if (appletState == deployment) { respond SW_WRONG_STATE; }
  if (certSize == 0) { respond SW_CERT_SIZE_UNSET; }
  if (certificate.length + certChunk.length > certSize)
    { respond SW_CERTIFICATE_TOO_LARGE; }
  certificate = certificate ++ certChunk;
} apdu {0x84, 0x38, 1, 0, certChunk, 0};

secure
command putProperties(Properties prop) {
  if (appletState == deployment) { respond SW_WRONG_STATE; }
  properties = prop;
} apdu {0x84, 0x58, 0, 0, prop, 0};

secure
command deploy() {
  if (appletState == deployment) { respond SW_WRONG_STATE; }
  if (privKeyState.isExponentSet
   && privKeyState.isModulusSet
   && certSize != 0
   && certificate.length == certSize
   && properties != {}) {
    appletState = deployment;
    certBytesSent = 0; // reset
  } else {
    respond SW_UNFINISHED_PERSONALIZATION;
  }
} apdu {0x84, 0x70, 0, 0, {}, 0};

secure
command update() {
  if (appletState != deployment) { respond SW_WRONG_STATE; }
  appletState = updating;
} apdu {0x84, 0x72, 0, 0, {}, 0};

command getPublicKey(Bit expORmod) { // 0 -> exponent, 1 -> modulus
  if (appletState != deployment) { respond SW_WRONG_STATE; }
  if (expORmod == 0) {
    respondok pubKey.exponent;
  } else {
    respondok pubKey.modulus;
  }
} apdu {0x80, 0x34, expORmod, 0, {}, KEY_SIZE/8};

command getCertificate() { // in chunks
  if (appletState != deployment) { respond SW_WRONG_STATE; }
  CertificateChunk chunk;
  if (certificate.length - certBytesSent >= MAX_CERT_CHUNK_SIZE) {
    chunk = certificate[certBytesSent,MAX_CERT_CHUNK_SIZE];
    certBytesSent = certBytesSent + MAX_CERT_CHUNK_SIZE;
  } else {
    chunk = certificate[certBytesSent..certificate.length-1];
    certBytesSent = certificate.length;
  }
  if (certificate.length - certBytesSent >= MAX_CERT_CHUNK_SIZE) {
    respond {chunk, SW1_MORE_DATA, MAX_CERT_CHUNK_SIZE};
  } else if (certificate.length - certBytesSent > 0) {
    respond {chunk, SW1_MORE_DATA, certificate.length - certBytesSent};
  } else /* certificate.length == certBytesSent */ {
    certBytesSent = 0; // reset
    respondok chunk;
  }
} apdu {0x80, 0x36, 0, 0, {}, MAX_CERT_CHUNK_SIZE};

command getProperties() {
  if (appletState != deployment) { respond SW_WRONG_STATE; }
  respondok properties;
} apdu {0x00, 0x56, 0, 0, {}, properties.length};

command privSignDecrypt(Message msg) {
  if (appletState != deployment) { respond SW_WRONG_STATE; }
  respondok encrypt(privKey,msg);
} apdu {0x80, 0x42, 0, 0, msg, MSG_SIZE};


const SW_WRONG_STATE                = 0x6985;
const SW_CERTIFICATE_TOO_LARGE      = 0x6A80;
const SW_UNFINISHED_PERSONALIZATION = 0x6985;
const SW_SECURITY_NOK               = 0x6982;
const SW_CERT_SIZE_UNSET            = 0x6985;
const SW1_MORE_DATA                 = 0x63;
const SW1_WRONG_PIN                 = 0x63;
const SW_PIN_BLOCKED                = 0x6983;


select {
  certBytesSent = 0; // reset
}

}
```

martSlang spec

PKI example



~ 190 lines

~ 820 lines

$$\frac{\text{size(code)}}{\text{size(spec)}} = \sim 4$$

AutoSmart

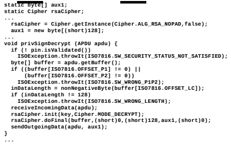
martSlang spec

Java Card code

PKI Java Card code

PKI

```
tic byte[] aux1;
tic Cipher rsaCipher;

saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];

d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```



```
...
static byte[] aux1;
static Cipher rsaCipher;
...
  rsaCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
  aux1 = new byte[(short)128];
...
void privSignDecrypt (APDU apdu) {
  if (! pin.isValidated())
    ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
  byte[] buffer = apdu.getBuffer();
  if ((buffer[ISO7816.OFFSET_P1] != 0) ||
      (buffer[ISO7816.OFFSET_P2] != 0))
    ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
  inDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
  if (inDataLength != 128)
    ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
  receiveIncomingData(apdu);
  rsaCipher.init(key,Cipher.MODE_DECRYPT);
  rsaCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
  sendOutgoingData(apdu, aux1);
}
...
```

Java Card code

PKI Java Card code

```
tic byte[] aux1;
tic Cipher rsaCipher;

saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];
```

method for command (after dispatching)

```
d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```

PKI Java Card code

```
tic byte[] aux1;
tic Cipher rsaCipher;

saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];

d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```

PIN check

PKI Java Card code

```
tic byte[] aux1;
tic Cipher rsaCipher;

saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];

d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```

command
APDU
decoding
and
checking

PKI Java Card code

```
tic byte[] aux1;
tic Cipher rsaCipher;

saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];

d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```

crypto operation

PKI Java Card code

```
tic byte[] aux1;
tic Cipher rsaCipher;

saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];

d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```

response sending

PKI Java Card code

```
tic byte[] aux1;
tic Cipher rsaCipher;
```

intermediate storage
for crypto operation

```
saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];

d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```

PKI Java Card code

```
tic byte[] aux1;
tic Cipher rsaCipher;
```

pre-allocated when applet is installed

```
saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];

d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```

storage re-used for every operation

because memory is scarce in smart cards

PKI Java Card code

```
tic byte[] aux1;
tic Cipher rsaCipher;

saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];

d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```

no run-time
“surprises”

guaranteed not to throw exceptions
thanks to AutoSmart’s static checking

PKI Java Card code

```
tic byte[] aux1;
tic Cipher rsaCipher;

saCipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,false);
ux1 = new byte[(short)128];

d privSignDecrypt (APDU apdu) {
f (! pin.isValidated())
 ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
yte[] buffer = apdu.getBuffer();
f ((buffer[ISO7816.OFFSET_P1] != 0) ||
   (buffer[ISO7816.OFFSET_P2] != 0))
 ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
nDataLength = nonNegativeByte(buffer[ISO7816.OFFSET_LC]);
f (inDataLength != 128)
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
eceiveIncomingData(apdu);
saCipher.init(privKey,Cipher.MODE_DECRYPT);
saCipher.doFinal(buffer,(short)0,(short)128,aux1,(short)0);
endOutgoingData(apdu, aux1);
```

idiomatic code, tuned for space efficiency

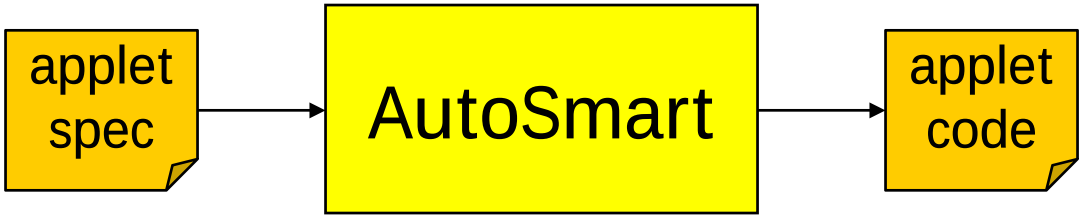
SmartSlang counterpart

```
command privSignDecrypt (Message msg) {
  respondok decrypt(key,msg);
} apdu {0x80,0x42,0,0,msg,1024/8};

PIN(8) p {
  protected = {privSignDecrypt, ...};
  ... }
```

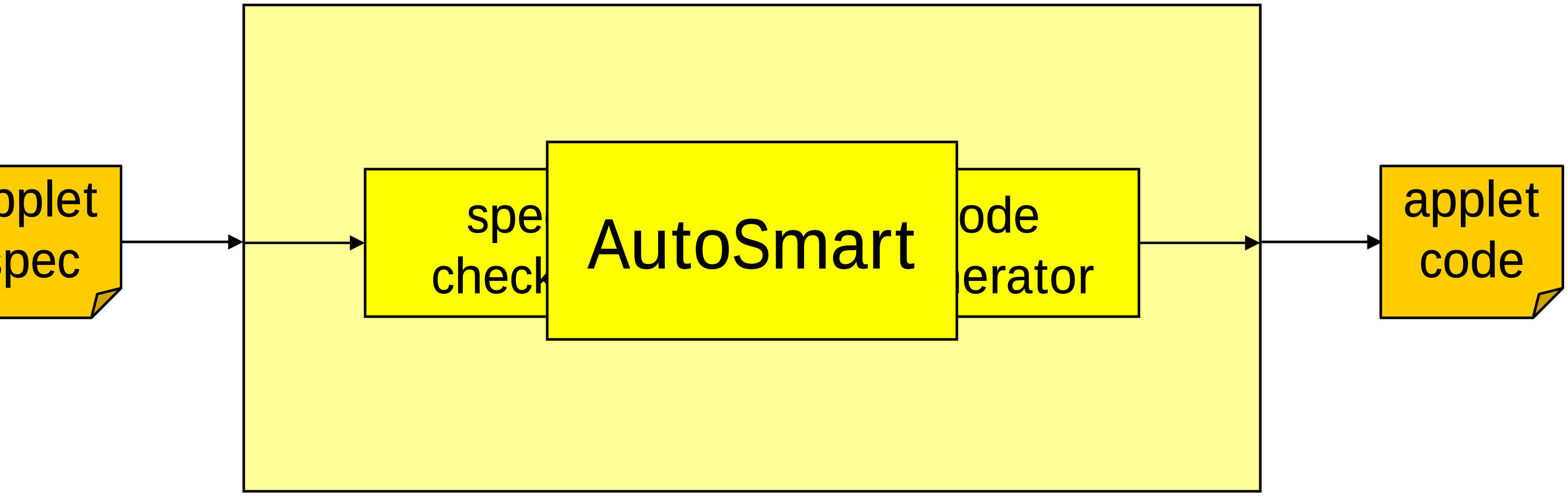
crypto result computed and returned “on the fly”

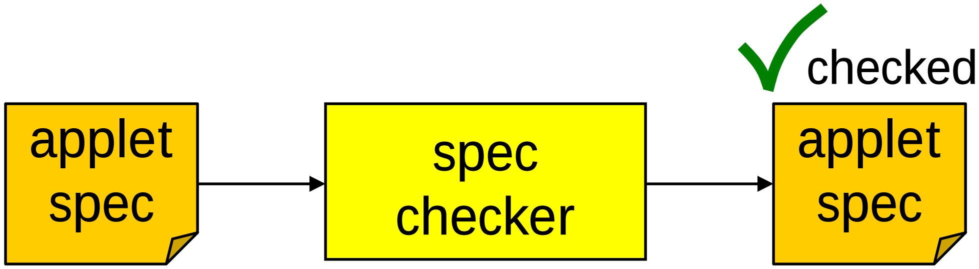
no concern for storage

taken care of by AutoSmart



AutoSmart


pplet
spec
spe
check
AutoSmart
ode
erator
applet
code

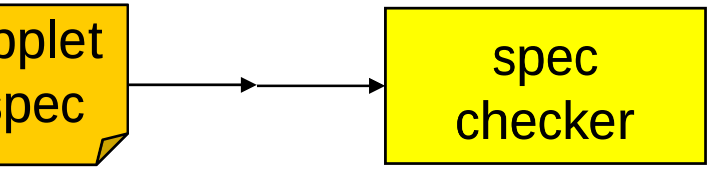


applet spec

spec checker

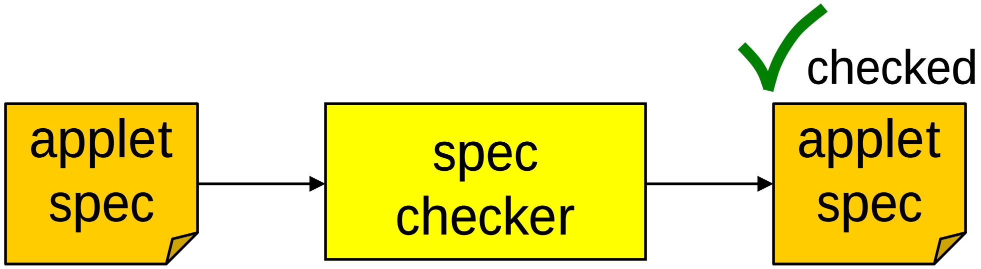
✓ checked

applet spec



pplet pec

spec checker



parses spec

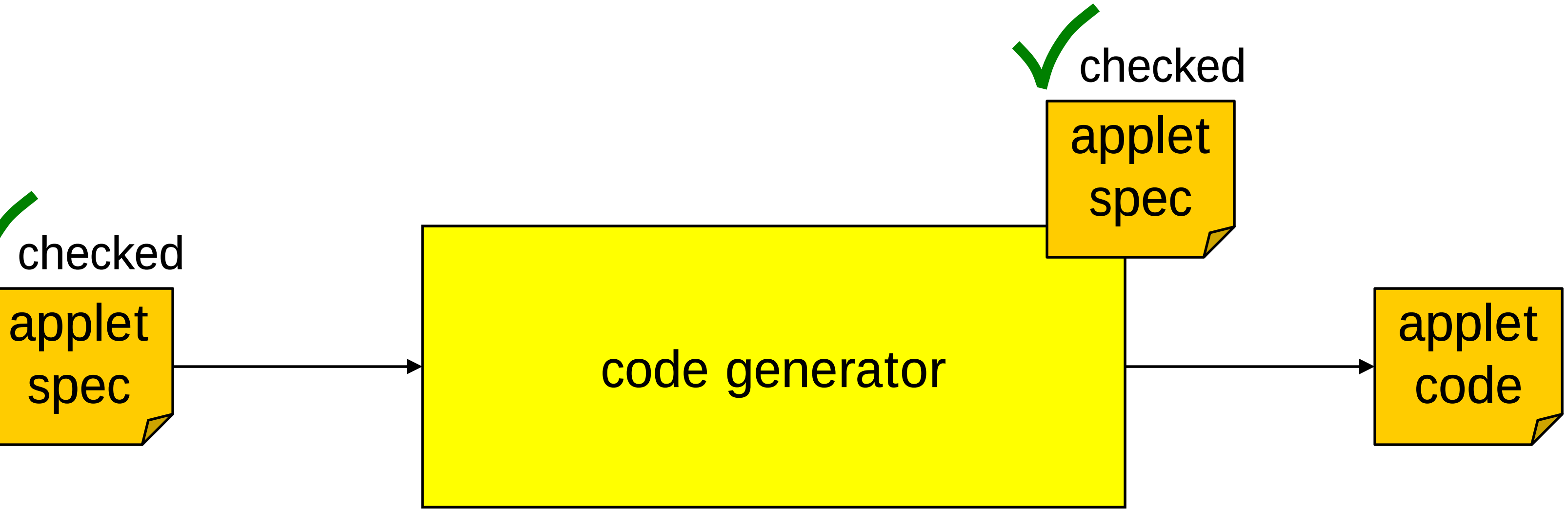
parser generated from grammar
via our own parser generator

checks type safety

includes Fourier-Motzkin decision
procedure for linear arithmetic

checks other properties

e.g. restrictions to allow generation
of code to check and decode APDUs



✓ checked

applet spec

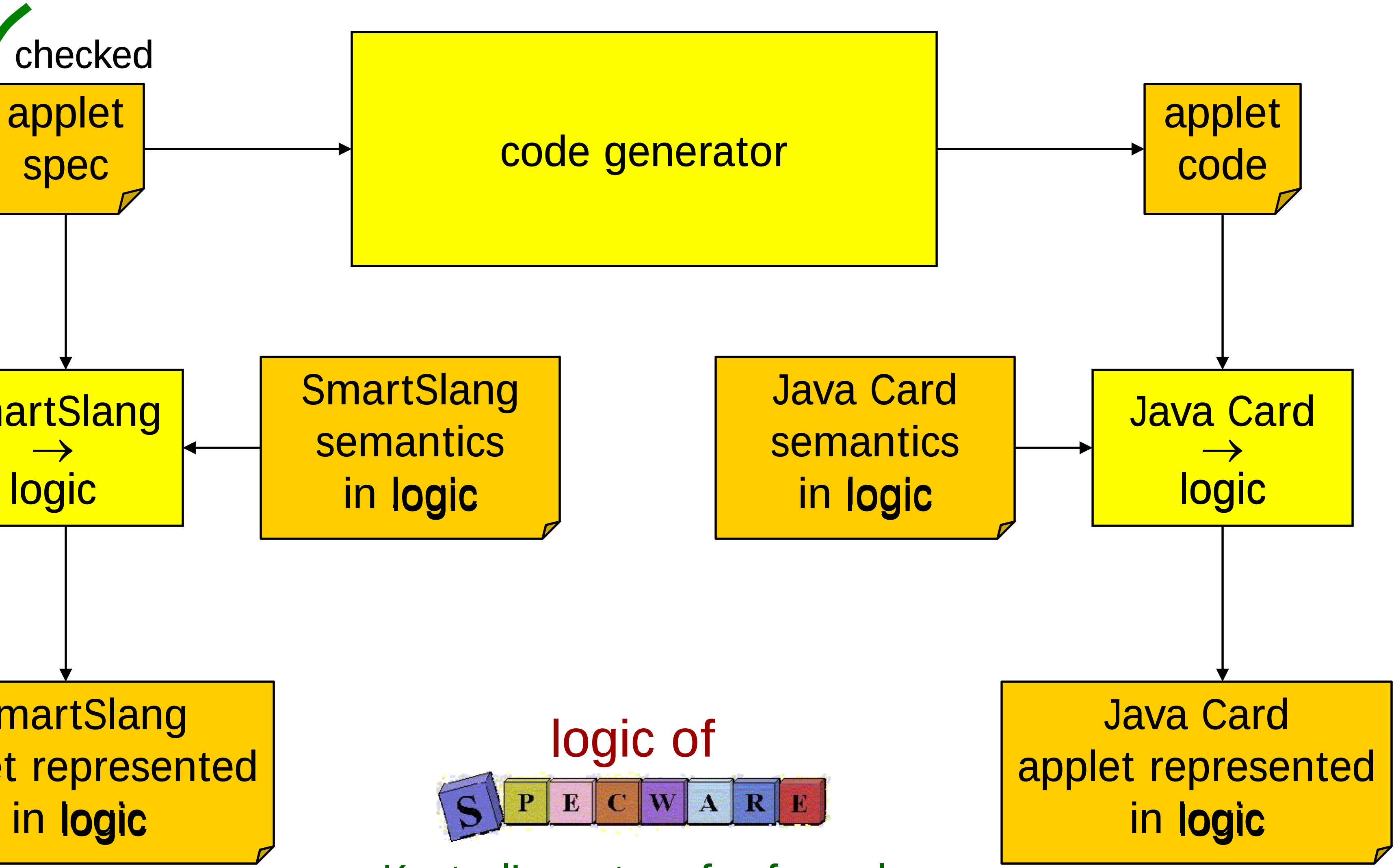
code generator

✓ checked

applet spec

applet code

correctness?



checked

applet
spec

code generator

applet
code

artSlang
→
logic

SmartSlang
semantics
in **logic**

Java Card
semantics
in **logic**

Java Card
→
logic

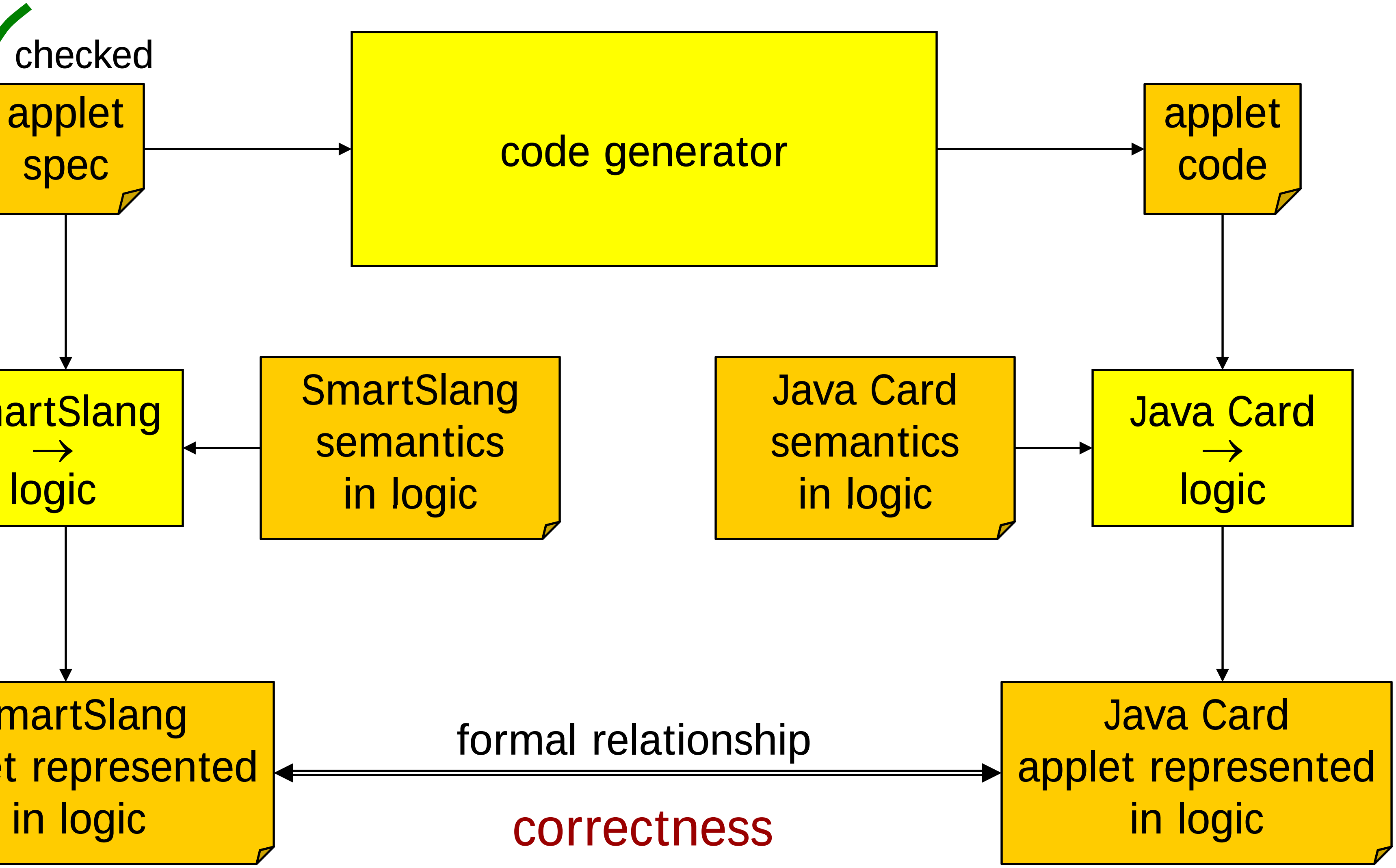
martSlang
t represented
in **logic**

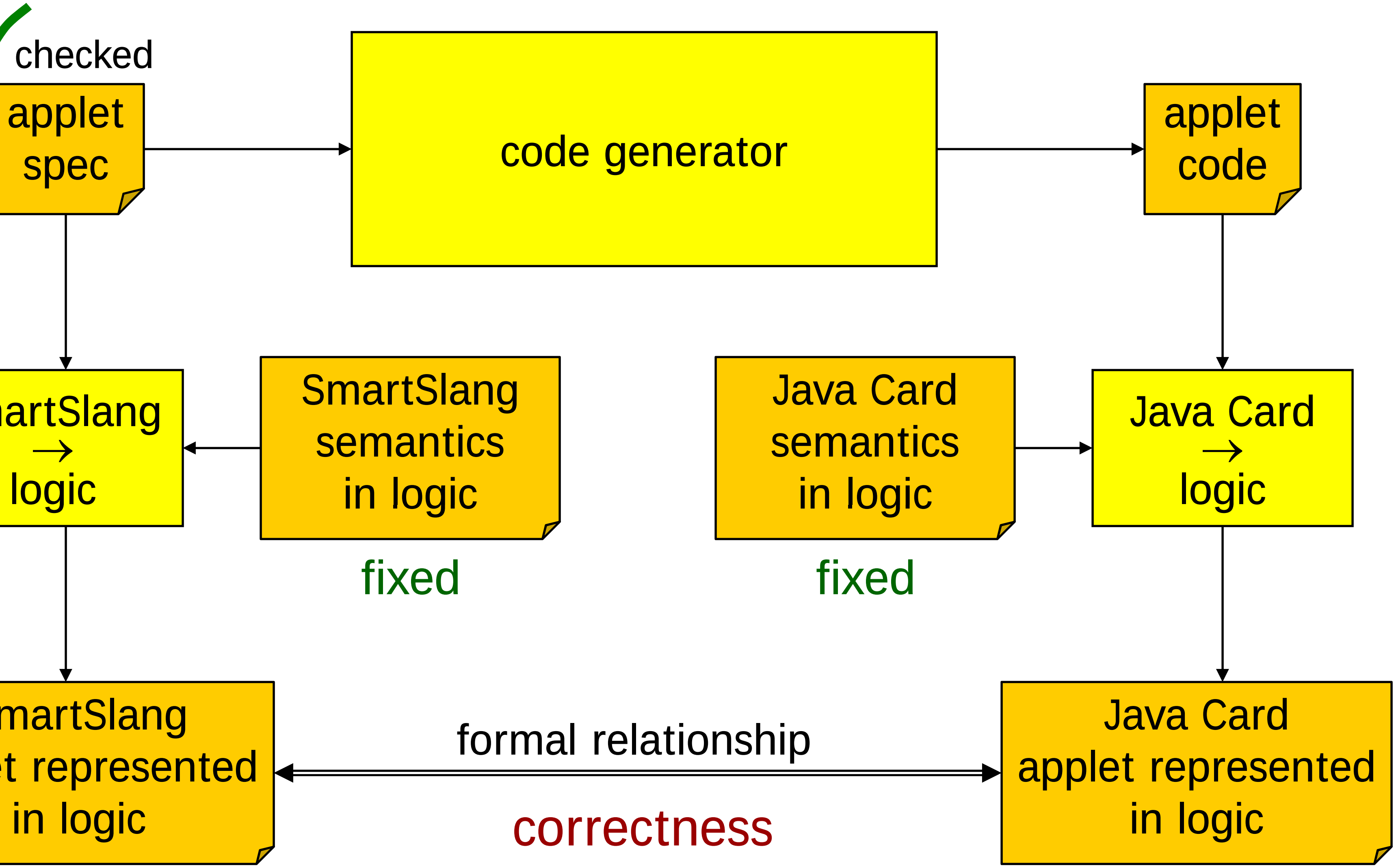
logic of

SPECWARE

Kestrel’s system for formal

Java Card
applet represented
in **logic**


checked
applet spec
code generator
applet code
artSlang → logic
SmartSlang semantics in logic
Java Card semantics in logic
Java Card → logic
martSlang
t represented
in logic
formal relationship
correctness
Java Card applet represented in logic



checked

applet spec

code generator

applet code

artSlang → logic

SmartSlang semantics in logic

fixed

Java Card semantics in logic

fixed

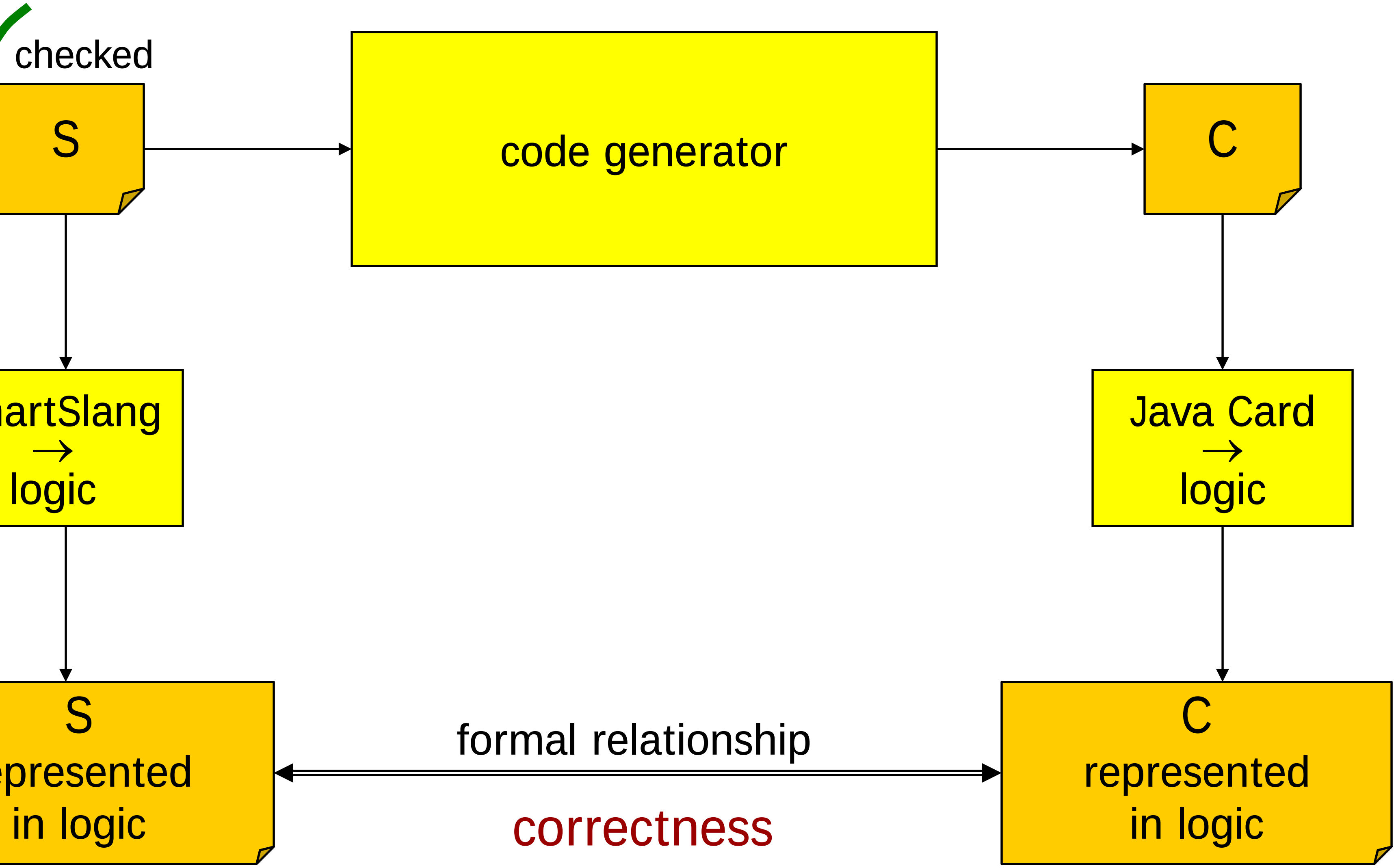
Java Card → logic

martSlang t represented in logic

formal relationship

correctness

Java Card applet represented in logic



✓ checked

S

code generator

C

artSlang → logic

Java Card → logic

S

epresented

in logic

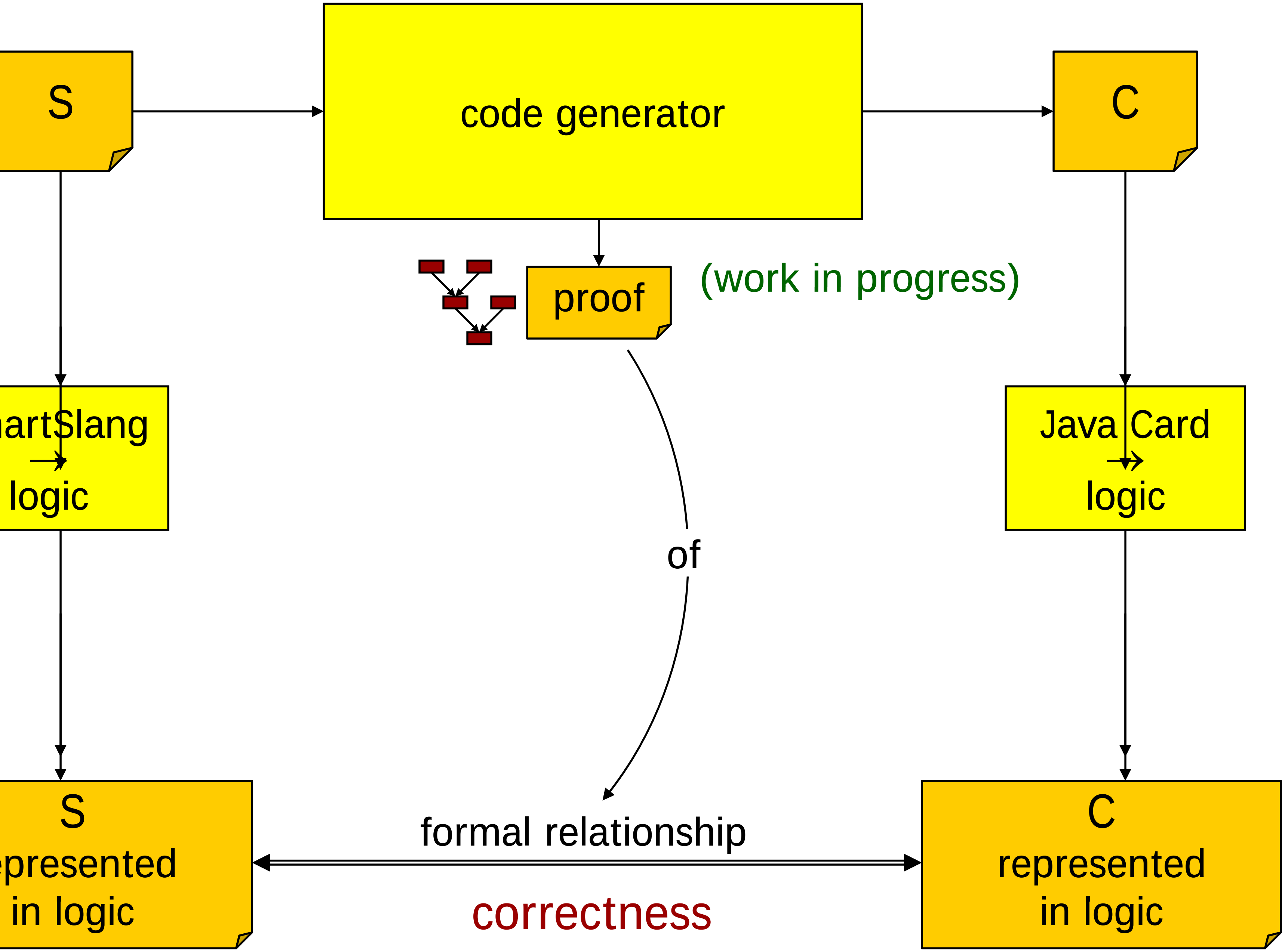
formal relationship

correctness

C

represented

in logic



S

code generator

C

proof

(work in progress)

artSlang
→
logic

Java Card
→
logic

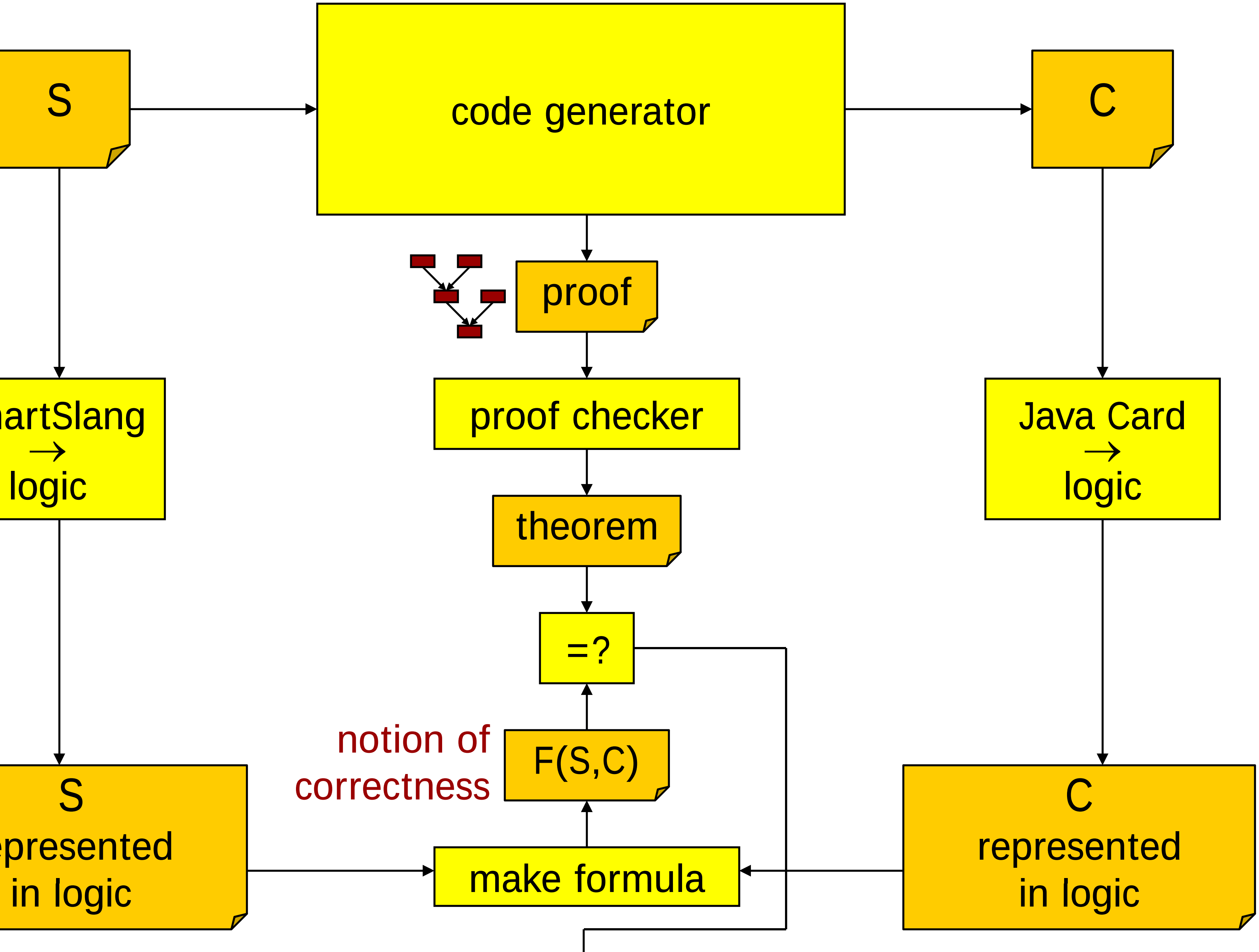
of

S
epresented
in logic

formal relationship

correctness

C
represented
in logic



S

code generator

C

proof

proof checker

theorem

=?

notion of correctness

F(S,C)

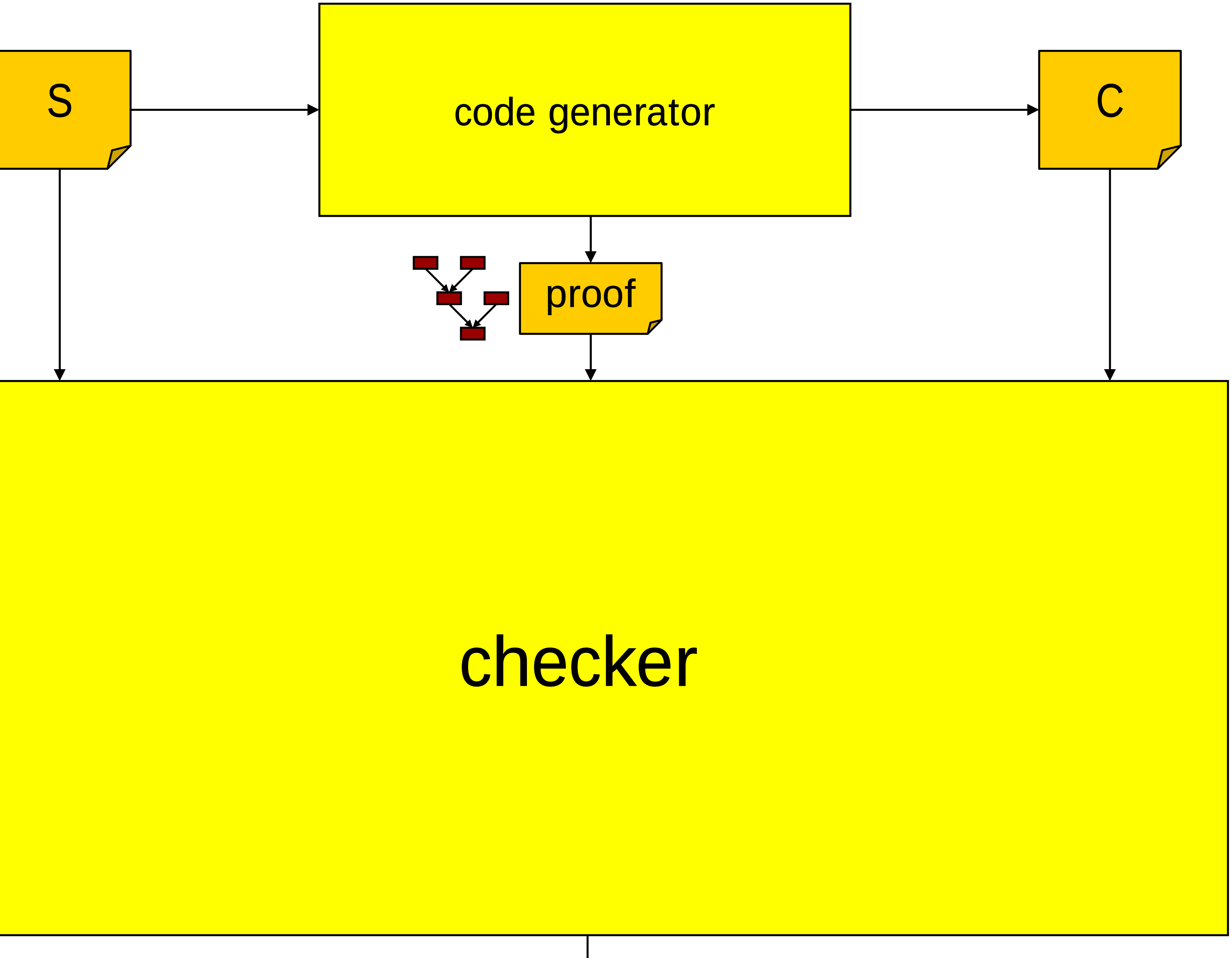
artSlang → logic

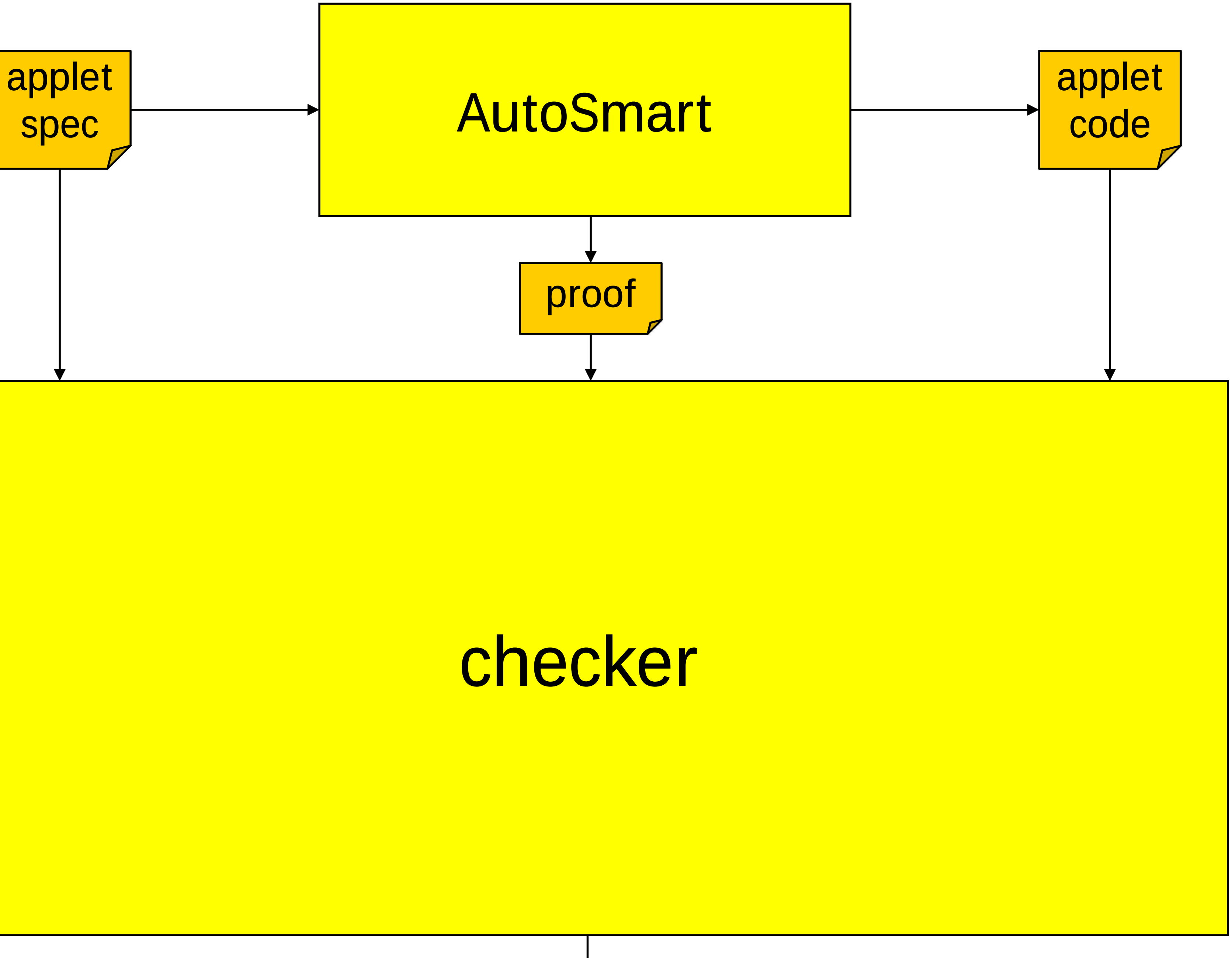
Java Card → logic

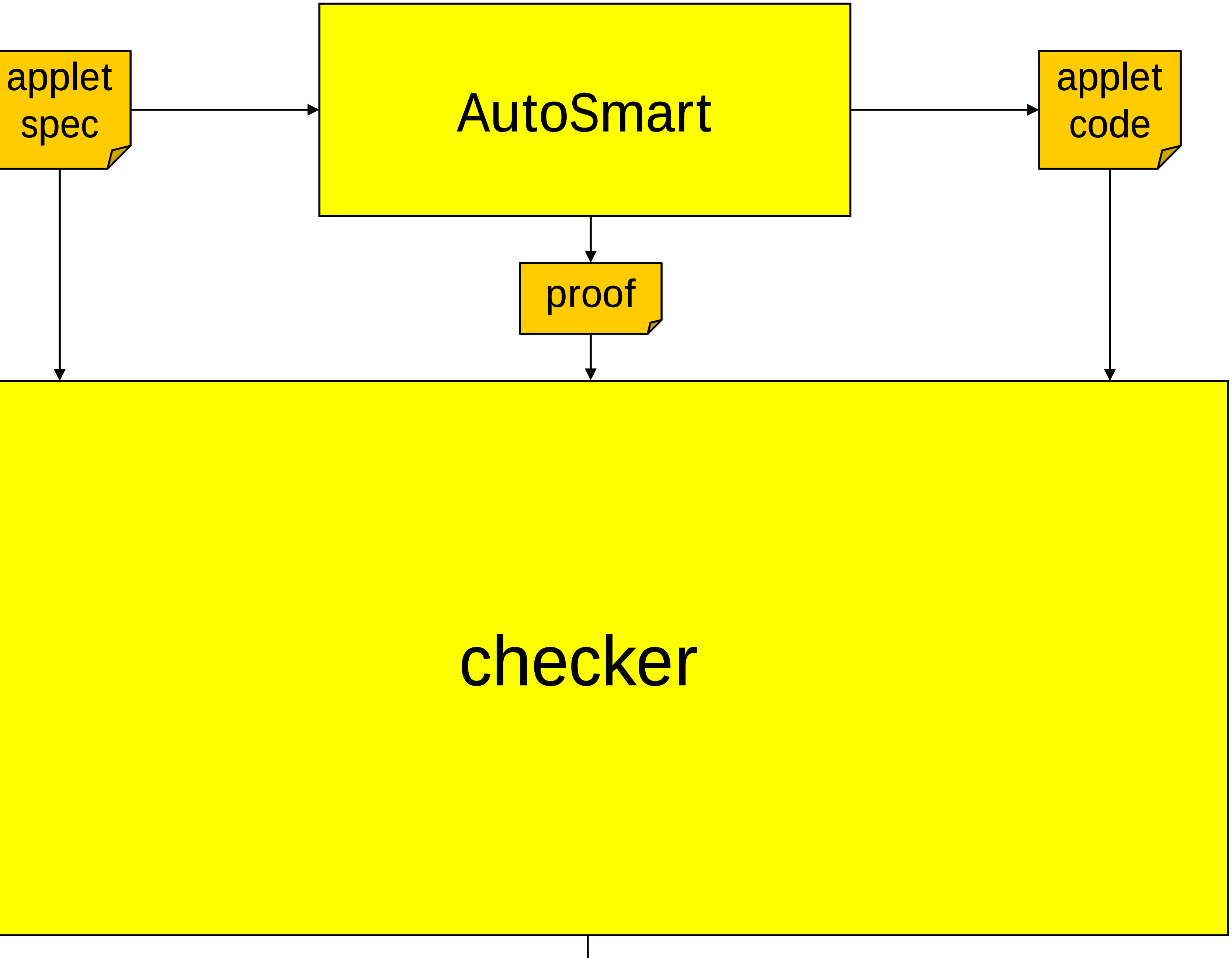
S epresented in logic

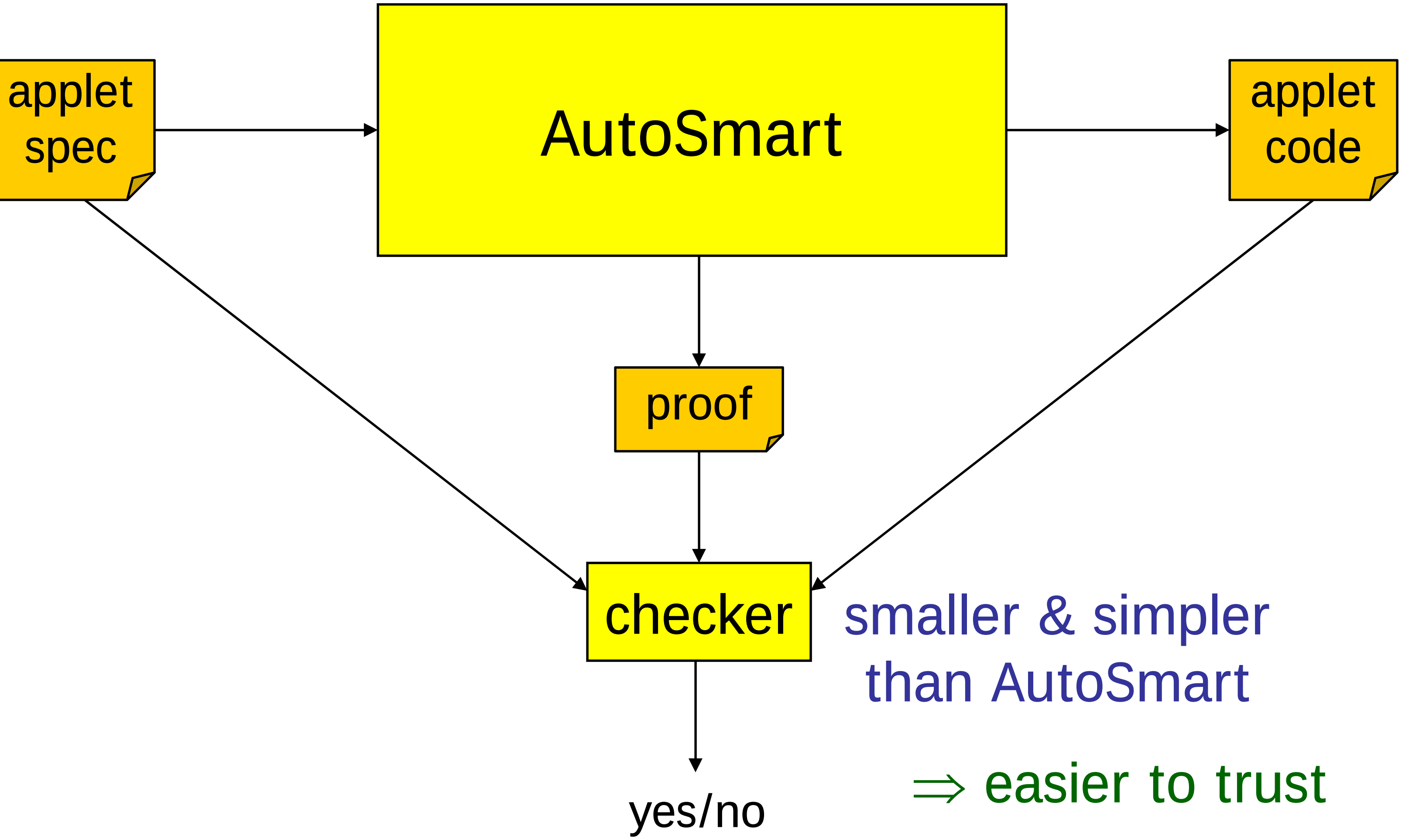
make formula

C represented in logic


S
code generator
C
proof
checker


applet spec
AutoSmart
applet code
proof
checker


applet
spec
AutoSmart
applet
code
proof
checker

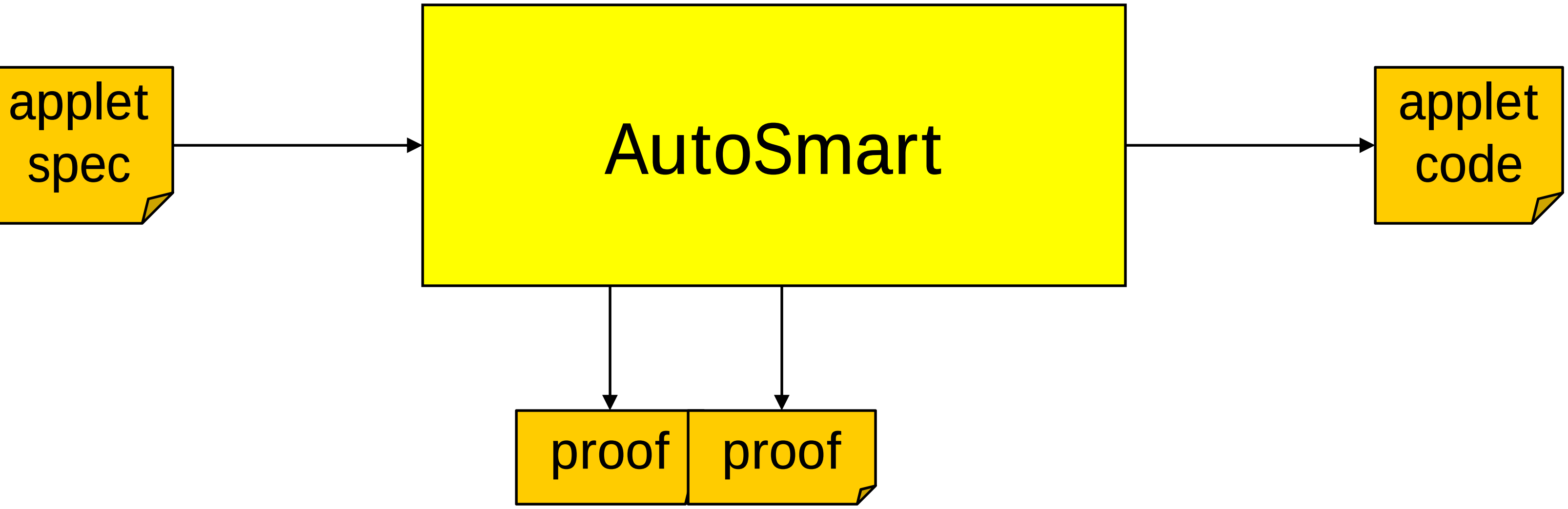


smaller & simpler
than AutoSmart

⇒ easier to trust

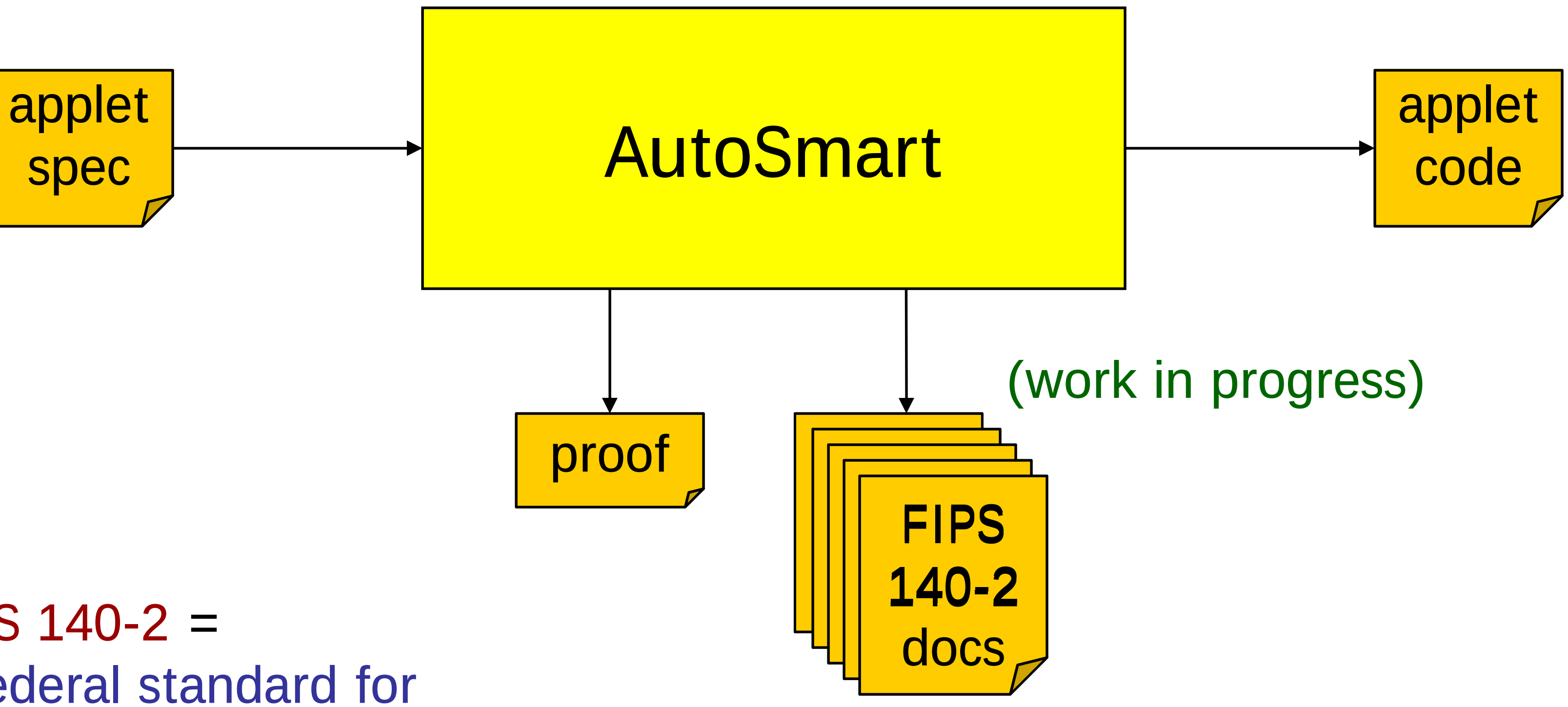
independent certification

independent certification



independent certification

independent certification


applet spec

AutoSmart

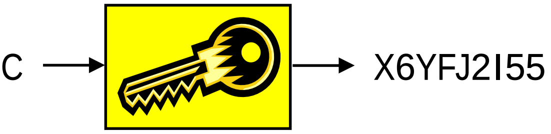
applet code

proof

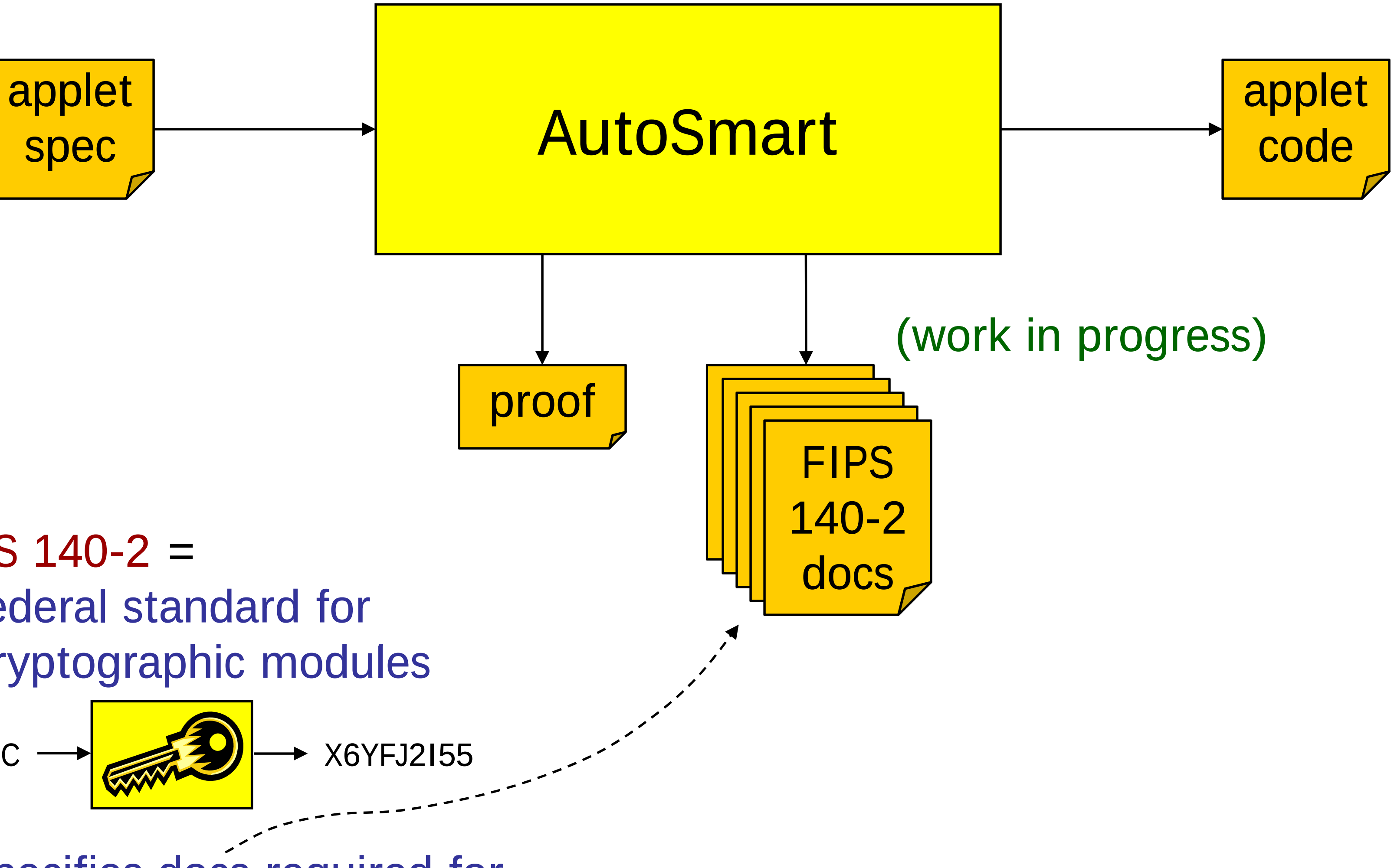
(work in progress)

FIPS 140-2 docs

S 140-2 =
ederal standard for
ryptographic modules



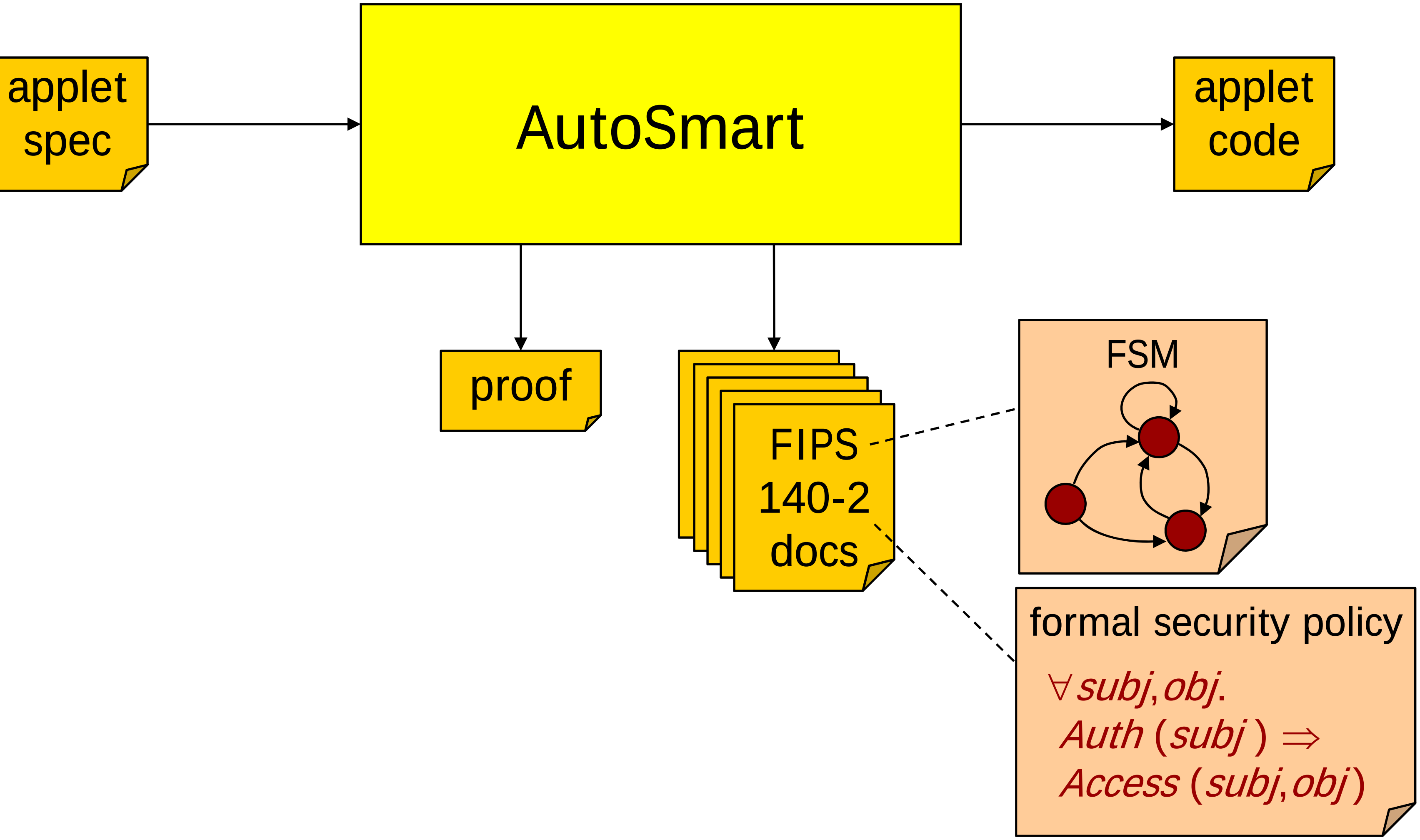
independent certification



S 140-2 =
ederal standard for
ryptographic modules

pecifies docs required for

independent certification



applet spec

AutoSmart

applet code

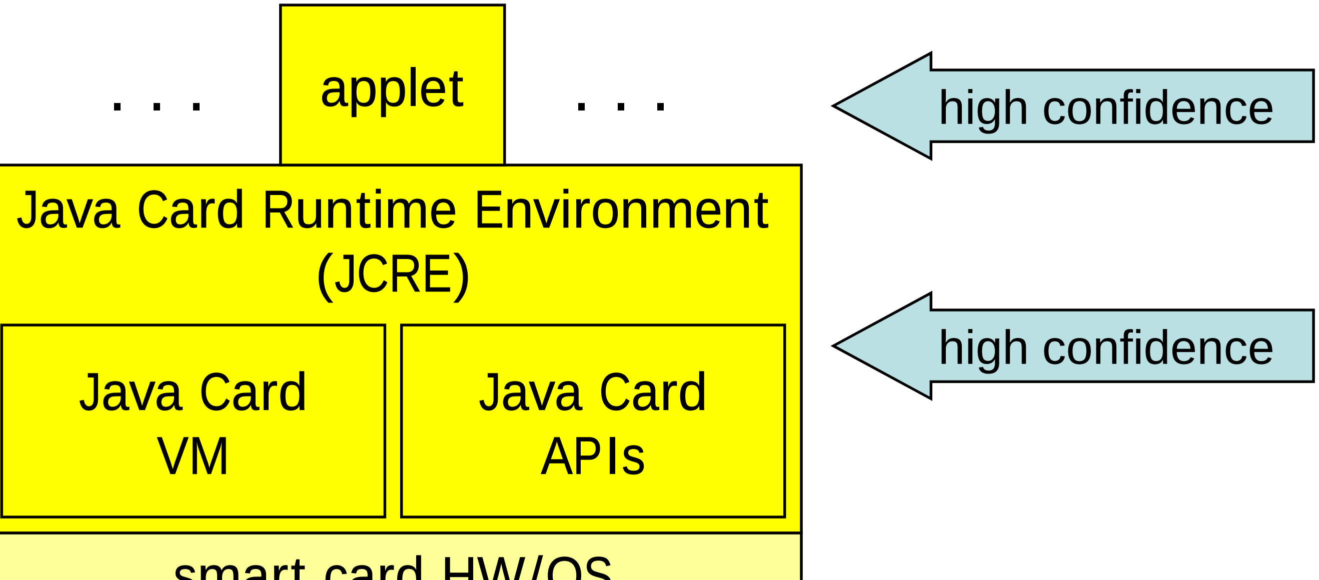
proof

FIPS 140-2 docs

FSM

formal security policy

$$\forall subj, obj.\ Auth(subj) \Rightarrow Access(subj, obj)$$



. . . applet . . .

high confidence

Java Card Runtime Environment (JCRE)

Java Card VM

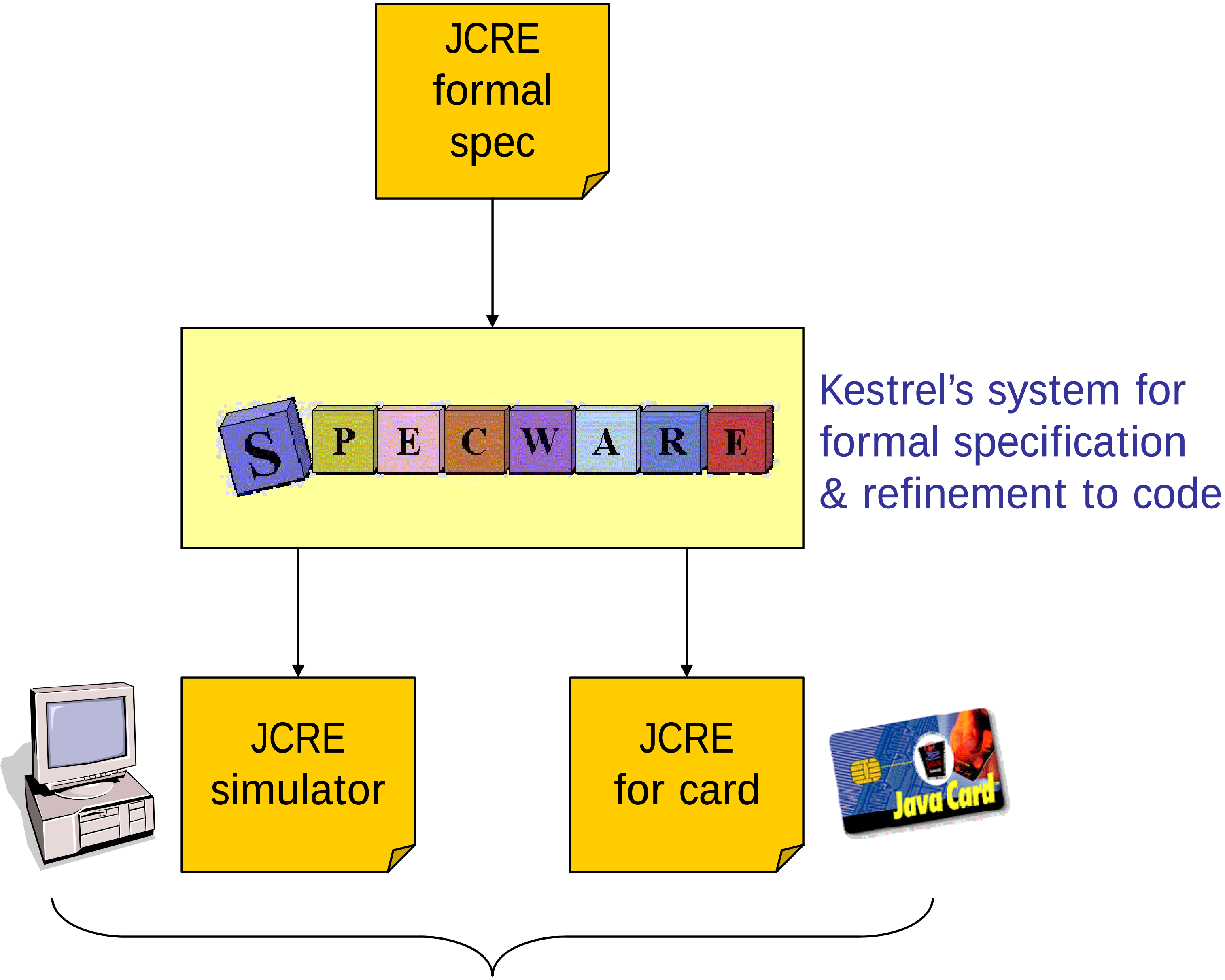
Java Card APIs

high confidence

smart card HW/OS

JCRE
formal
spec

JCRE



JCRE formal spec

Kestrel’s system for formal specification & refinement to code

JCRE simulator

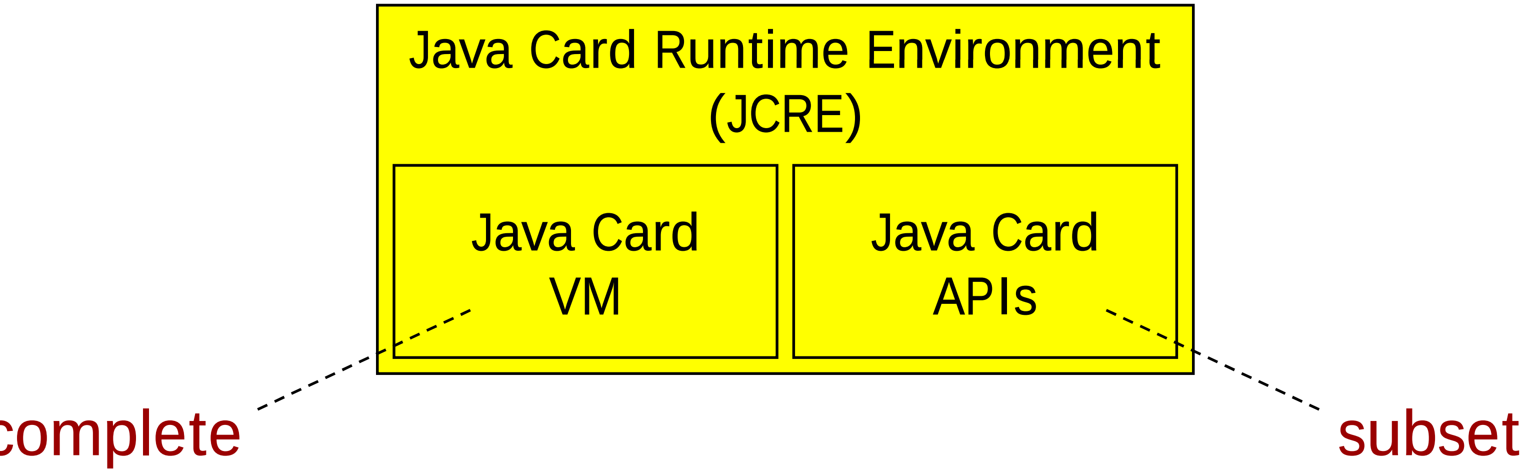
JCRE for card

guaranteed same behavior

JCRE formal spec

JCRE
formal
spec

JCRE formal spec



Java Card Runtime Environment (JCRE)

Java Card VM

Java Card APIs

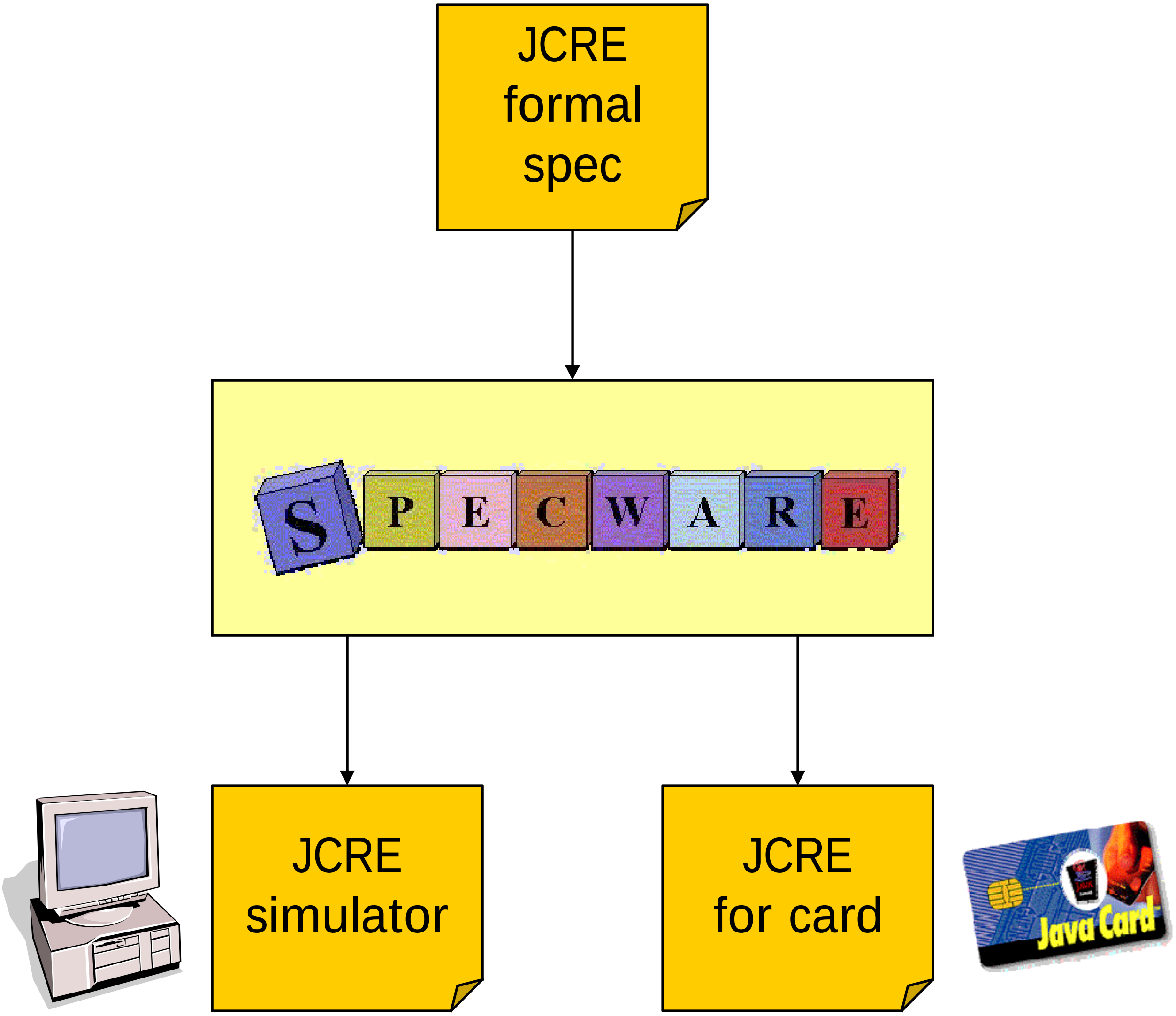
complete

subset

small but sufficient
to “run” some applets
(I/O & other basics)

size: ~ 12K lines
(~ 50% comments)

JCRE formal spec



JCRE formal spec

JCRE simulator

JCRE for card

JCRE simulator

JCRE
simulator

JCRE simulator

① refinement of formal spec

~ 8K lines

② automatic code generation from refined spec

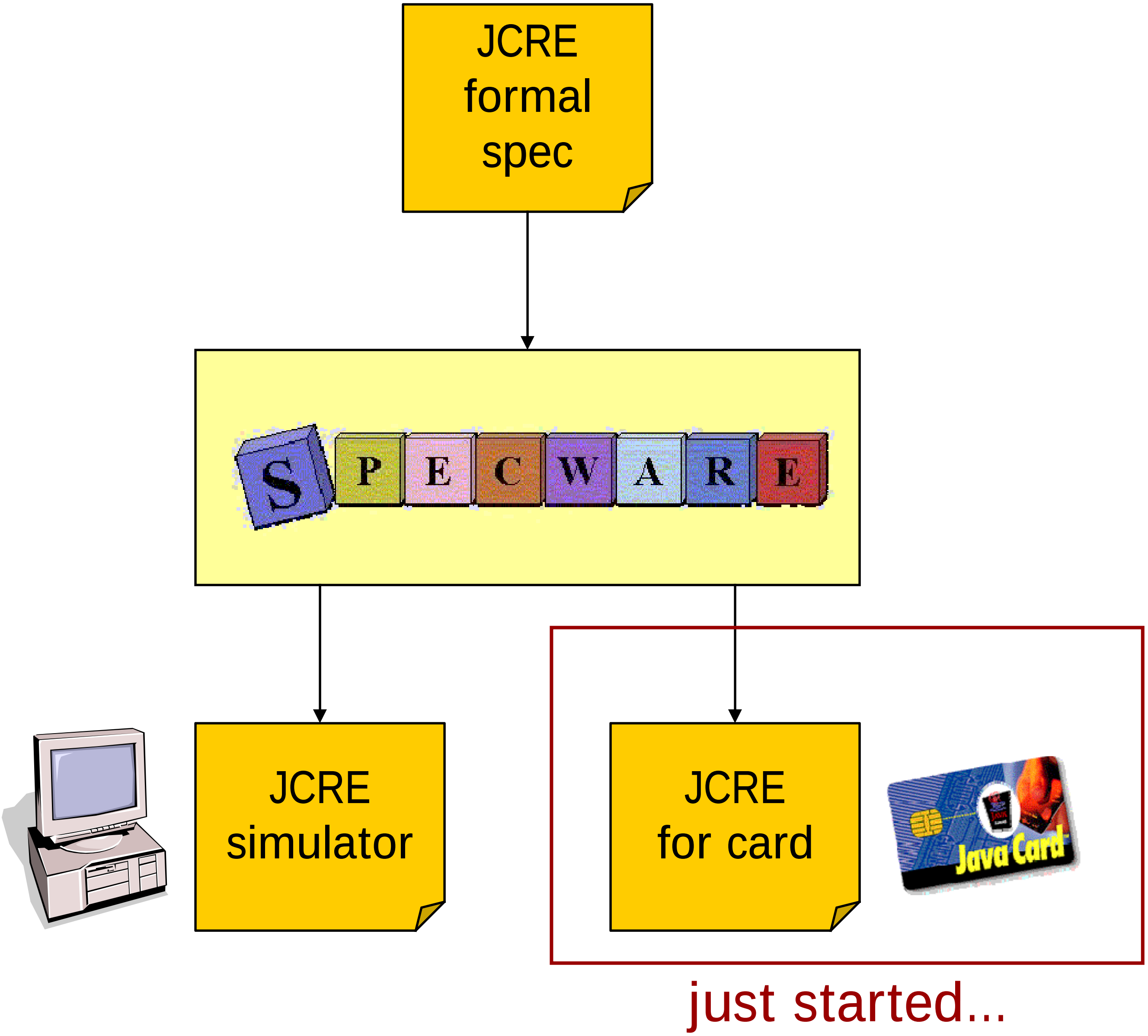
~ 35K lines

can be optimized via further refinement

successfully tested on a few applets

JCRE simulator

JCRE
simulator



JCRE formal spec

JCRE simulator

JCRE for card

just started...

recap

- ❑ Java Card
- ❑ AutoSmart
- ❑ SmartSlang example
- ❑ generated Java Card code
- ❑ proof generation & checking
- ❑ JCRE formal spec
- ❑ refinement to JCRE simulator