Defeating APT10 Compiler-level Obfuscations

Takahiro Haruyama

Threat Analysis Unit

Carbon Black

Who am I?

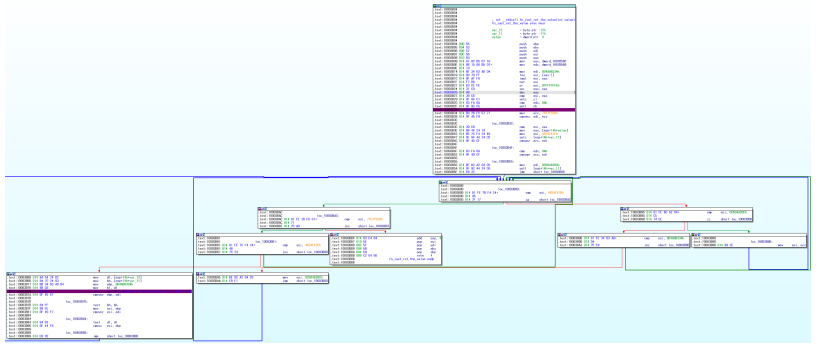
- Takahiro Haruyama (@cci_forensics)
  - Principal Threat Researcher
    - Carbon Black's Threat Analysis Unit (TAU)
  - Reverse-engineering cyber espionage malware
    - linked to PRC/Russia/DPRK
  - Past public research presentations
    - binary diffing, Winnti/PlugX malware research
    - forensic software exploitation, memory forensics

Overview

- Motivation and Approach
- Microcode
- Opaque Predicates
- Control Flow Flattening
- IDA 7.2 Issues and 7.3 Improvements
- Wrap-up

Motivation and Approach

Question

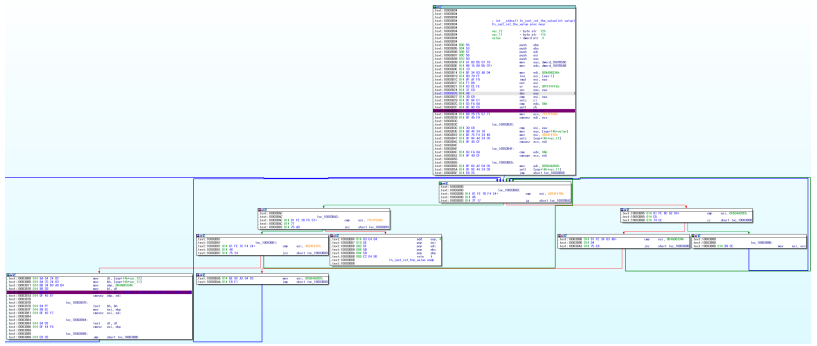


```
12  v1 = 0xD4A06334;
13  v2 = ~(dword_1007B58C * (dword_1007B58C - 1)) | -2u;
14  v3 = 0x7157F526;
15  if ( (v2 == -1) != dword_1007B588 < 10 )
16    v1 = 0x7157F526;
17  v4 = v2 == -1;
18  result = value;
19  b_cmp = 0x4624F47C;
20  if ( !v4 )
21    v3 = v1;
22  if ( dword_1007B588 >= 10 )
23    v3 = v1;
24  v8 = dword_1007B588 < 10;
25  while ( 1 )
26  {
27    while ( b_cmp <= 0x4624F47B )
28    {
29      if ( b_cmp == 0xC504A26C )
30        b_cmp = v3;
31      else
32        b_cmp = 0xC504A26C;
33    }
34    if ( b_cmp == 0x7157F526 )
35      break;
36    v7 = 0xD4A06334;
37    if ( v8 != v4 )
38      v7 = 0xC504A26C;
39    b_cmp = v7;
40    if ( v8 )
41      b_cmp = 0xC504A26C;
42    if ( !v4 )
43      b_cmp = v7;
44  }
45  return result;
```

This function just returns the value

Question

Opaque Predicates



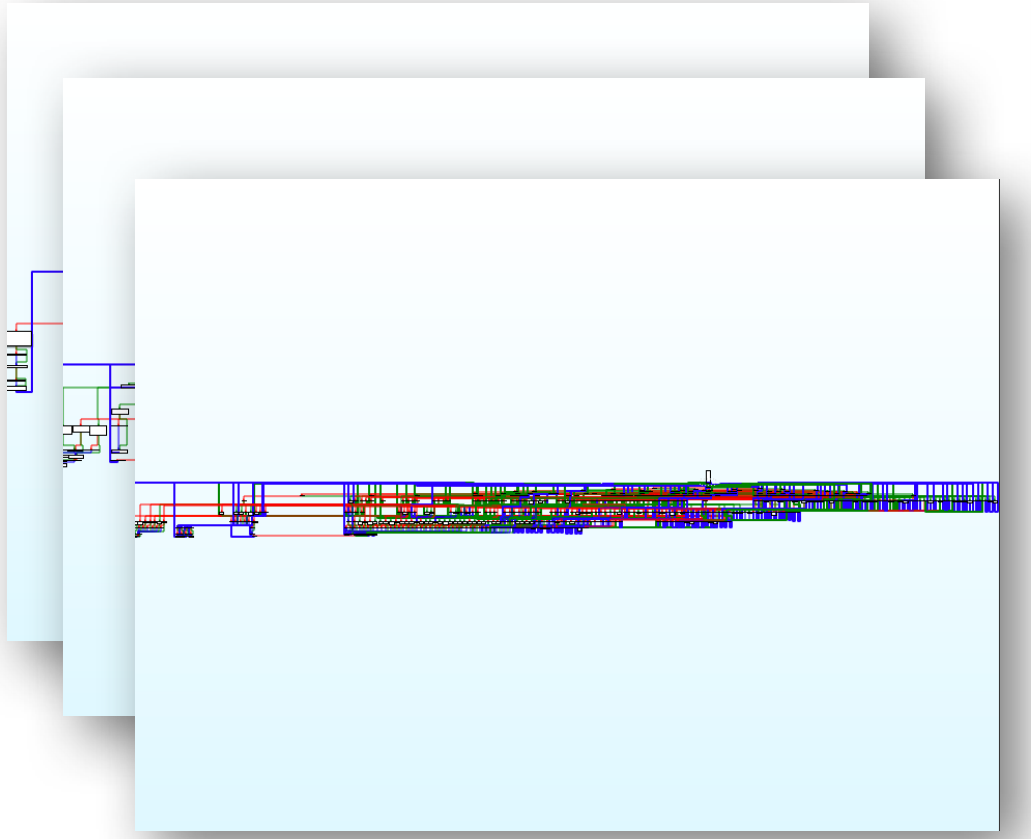
```
12  v1 = 0xD4A06334;
13  v2 = ~(dword_1007B58C * (dword_1007B58C - 1)) | -2u;
14  v3 = 0x7157F526;
15  if ( (v2 == -1) != dword_1007B588 < 10 )
16    v1 = 0x7157F526;
17  v4 = v2 == -1;
18  result = value;
19  b_cmp = 0x4624F47C;
20  if ( !v4 )
21    v3 = v1;
22  if ( dword_1007B588 >= 10 )
23    v3 = v1;
24  v8 = dword_1007B588 < 10;
25  while ( 1 )
26  {
27    while ( b_cmp <= 0x4624F47B )
28    {
29      if ( b_cmp == 0xC504A26C )
30        b_cmp = v3;
31      else
32        b_cmp = 0xC504A26C;
33    }
34    if ( b_cmp == 0x7157F526 )
35      break;
36    v7 = 0xD4A06334;
37    if ( v8 != v4 )
38      v7 = 0xC504A26C;
39    b_cmp = v7;
40    if ( v8 )
41      b_cmp = 0xC504A26C;
42    if ( !v4 )
43      b_cmp = v7;
44  }
45  return result;
```

Control Flow Flattening

APT10 ANEL [1][2]

- RAT program used by APT10
  - observed in Japan uniquely
- ANEL version 5.3.0 or later are obfuscated with
  - opaque predicates
  - control flow flattening

Examples



```
v4 = this;
v5 = -1155786945;
v39 =
v28 =
v38 =
while
{
  wh
  {
    while
    {
      whi
      {
        w
        {
```

```
v5 = 543224093;
v43 = 0;
v42 = a2;
v44 = a1;
v46 = a
while (
{
  while
  {
    whi
    {
      w
      {
```

```
v3 = this;
v4 = -1989932919;
v107 = this;
v149 = (~(dword_1007B56C * (dword_1007B56C - 1)) | 0xFFFFFFFE) == -1;
v108 = this + 13;
v148 = dword_1007B568 < 10;
v109 = this + 3;
v99 = this + 2;
while ( 1 )
{
  while ( 1 )
  {
    while ( 1 )
    {
      while ( 1 )
      {
        while ( 1 )
        {
          while ( 1 )
          {
            while ( 1 )
            {
              while ( v4 <= -92467710 )
              {
                if ( v4 <= -1496265648 )
                {
                  if ( v4 <= -1818342062 )
                  {
                    if ( v4 > -1996574571 )
                    {
                      if ( v4 <= -1862869731 )
                      {
                        if ( v4 > -1925852450 )
                        {
                          if ( v4 != -1925852449 )
                          {
                            v4 = 304258603;
                            v13 = 304258603;
                            v127 = v117;
                            v12 = dword_1007B568;
                            v41 = (dword_1007B56C * (dword_1007B56C - 1)) & ((dword_1007B56
                            v15 = v41 == 0;
                            v16 = v41 == 0;
                            v17 = -348560582;
                            if ( !v16 )
                              v13 = -348560582;
                            goto LABEL_470;
                          }
                          v4 = 1899001619;
                          v153[1] = *(v156 + 1);
                          v81 = v156;
                          v82 = *(v153 + 68);
                          *(v153 + 68) = *(v156 + 68);
                          v81[68] = v82;
                          v126 = v94;
```

We need an automated de-obfuscation method

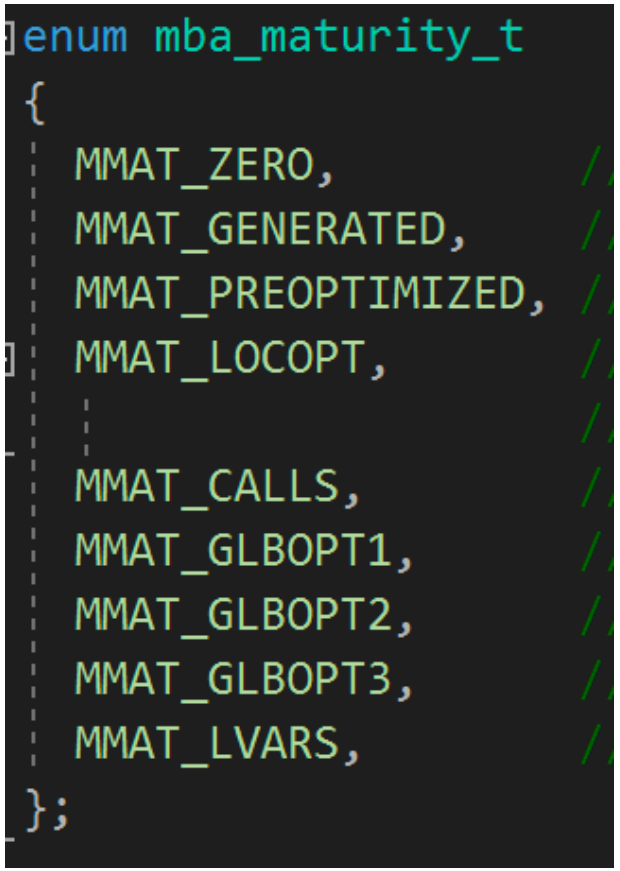
Motivation and Approach

- automate ANEL code de-obfuscations
  - The obfuscations looked similar to the ones described in Hex-Rays blog [3]
  - The IDA plugin HexRaysDeob [4] didn't work
    - It was made for another variant of the obfuscations
  - I investigated the causes then modified HexRaysDeob to work for ANEL samples [8]

Microcode

Microcode

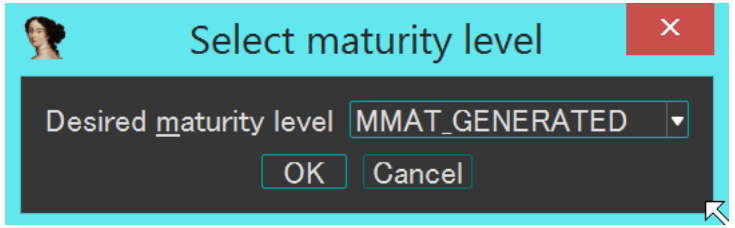
low

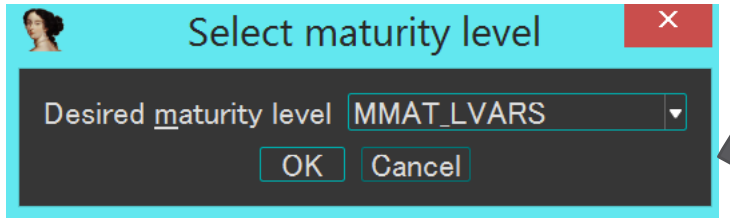


high

- intermediate representation (IR) used by IDA Pro decompiler
- optimized in 9 maturity levels
  - transformed from low-level to high-level IRs [3]

Microcode Explorer [4]






```
12.12 mov    cs.2, seg.2
12.13 mov    #0x75449F57.4, eoff.4
12.14 call   $fn_blowfish_encdec
12.14
13. 0 ; ????-BLOCK 13 PROP PUSH [STAR
13. 0 ; USE-DEF LISTS ARE NOT READY
13. 0 mov    ebp.4, eoff.4
13. 1 mov    ss.2, seg.2
13. 2 sub    eoff.4, #0x10.4, eoff.4
13. 3 ldx    seg.2, eoff.4, t1.1
13. 4 xdu    t1.1, eax.4
13. 5 mov    #8.1, t1.1
13. 6 cfshl  eax.4, t1.1, cf.1
13. 7 shl    eax.4, t1.1, eax.4
```

over 150 instructions

just 8 instructions

```
7. 0 call   $fn_blowfish_encdec <spec:"_DWORD *a1" sbox.4,"_BYTE *a2" &(var10{22}).4,"unsigned __int8
7. 1 xdu    varC@1.1, ecx7.4{23}    ; 75457FC2 u=sp+18.4    d=ecx.4
7. 2 stx    (xdu.4(var10@3.1) | ((xdu.4(var10@2.1) | ((xdu.4(var10@1.1) | (xdu.4(var10.1) <<l #8.1)) <
7. 3 or     xdu.4(varC@3.1), ((xdu.4(varC@2.1) | ((ecx7.4{23} | (xdu.4(varC.1) <<l #8.1)) <<l #8.1)) <
7. 4 stx    result.4{26}, ds.2{24}, edi5.4{25} ; 75457FE4 u=eax.4,edi.4,ds.2 d=(GLBLOW,sp+14..,GLBHIGH
7. 5 add    edi5.4{25}, #8.4, edi5.4 ; 75457FE6 u=edi.4      d=edi.4
7. 6 sub    var4.4{27}, #1.4, var4.4{28} ; 75457FE9 u=sp+20.4    d=sp+20.4
7. 7 jnz    var4.4{28}, #0.4, @7    ; 75457FEC u=sp+20.4
```

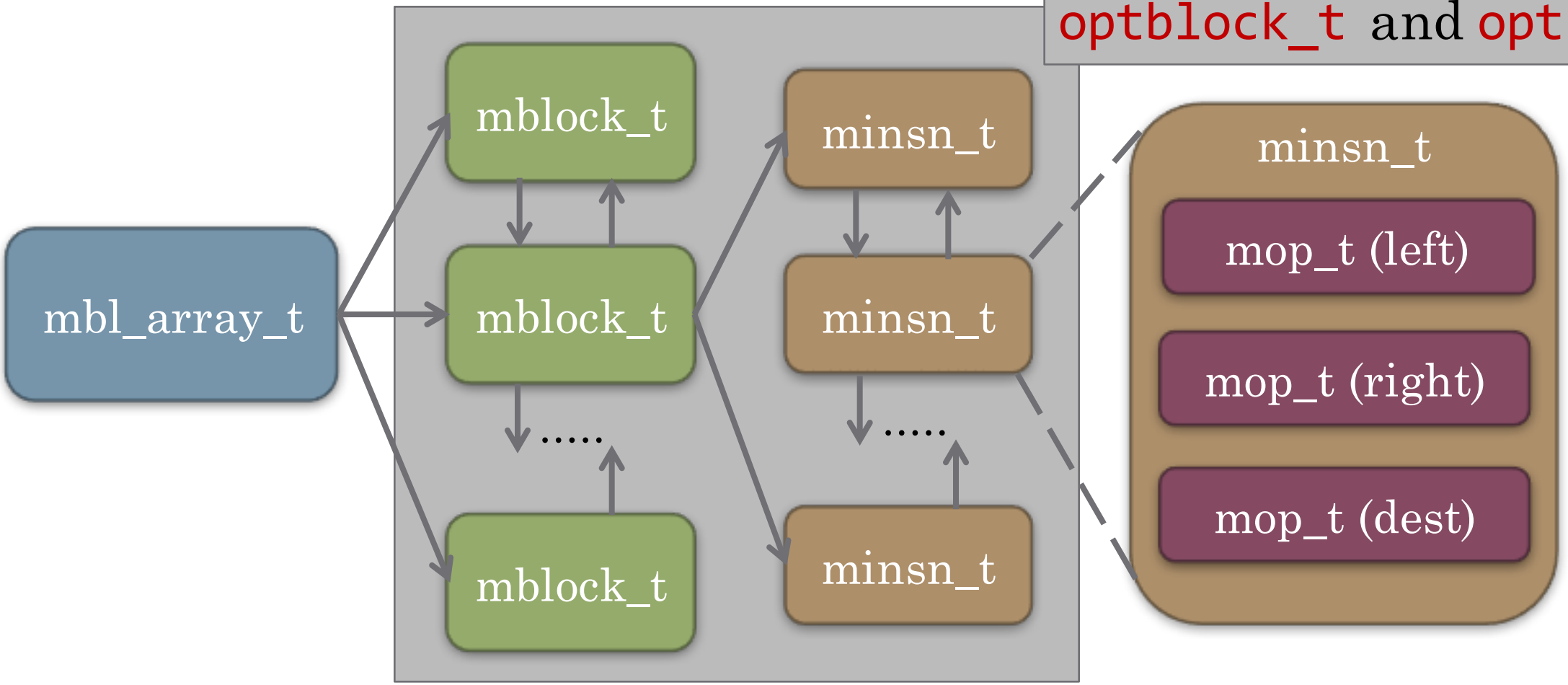
Microcode Explorer [4]

```
do
{
  fn_blowfish_encdec(sbox, &v12, (unsigned __int8 *)&v12);
  v10 = BYTE1(v13);
  *(v9 - 1) = HIBYTE(v12) | ((BYTE2(v12) | ((BYTE1(v12) | ((unsigned __int8)v12 << 8)
  result = HIBYTE(v13) | ((BYTE2(v13) | ((v10 | ((unsigned __int8)v13 << 8)) << 8)) <
  *v9 = result;
  v9 += 2;
  --v15;
}
while ( v15 );
--v14;
```

```
7. 0 call   $fn_blowfish_encdec <spec:"_DWORD *a1" sbox.4,"_BYTE *a2" &(var10{22}).4,"unsigned __int8
7. 1 xdu    varC@1.1, ecx7.4{23}     ; 75457FC2 u=sp+18.4    d=ecx.4
7. 2 stx    (xdu.4(var10@3.1) | ((xdu.4(var10@2.1) | ((xdu.4(var10@1.1) | (xdu.4(var10.1) <<I #8.1)) <
7. 3 or     xdu.4(varC@3.1), ((xdu.4(varC@2.1) | ((ecx7.4{23} | (xdu.4(varC.1) <<I #8.1)) <<I #8.1)) <
7. 4 stx    result.4{26}, ds.2{24}, edi5.4{25} ; 75457FE4 u=eax.4,edi.4,ds.2 d=(GLBLOW,sp+14..,GLBHIGH
7. 5 add    edi5.4{25}, #8.4, edi5.4 ; 75457FE6 u=edi.4      d=edi.4
7. 6 sub    var4.4{27}, #1.4, var4.4{28} ; 75457FE9 u=sp+20.4    d=sp+20.4
7. 7 jnz    var4.4{28}, #0.4, @7     ; 75457FEC u=sp+20.4
```

Key Structures [5]

HexRaysDeob installs two optimizer callbacks: `optblock_t` and `optinsn_t`



CFG and Instructions in Microcode Explorer

CFG (mblock_t)

block number

```
6. 0 mov    #0x80.4, var4.4          ; 7

7. 0 call   $fn_blowfish_encdec <spec:"_DWORD *a1" sbox.4,"_BYTE *a2" &(var10{22}).4,"unsigned __int8 *a3" &(var10{22}).4>.0 ; 75457F9D u=ebx.
7. 1 xdu    varC@1.1, ecx7.4{23}     ; 75457FC2 u=sp+18.4    d=ecx.4
7. 2 stx    (xdu.4(var10@3.1) | ((xdu.4(var10@2.1) | ((xdu.4(var10@1.1) | (xdu.4(var10.1) <<I #8.1)) <<I #8.1)) <<I #8.1)), ds.2{24}, (edi5.4{
7. 3 or     xdu.4(varC@3.1), ((xdu.4(varC@2.1) | ((ecx7.4{23} | (xdu.4(varC.1) <<I #8.1)) <<I #8.1)) <<I #8.1), result.4{26} ; 75457FE2 u=ecx.
7. 4 stx    result.4{26}, ds.2{24}, edi5.4{25} ; 75457FE4 u=eax.4,edi.4,ds.2 d=(GLBLOW,sp+14..,GLBHIGH)
7. 5 add    edi5.4{25}, #8.4, edi5.4 ; 75457FE6 u=edi.4      d=edi.4
7. 6 sub    var4.4{27}, #1.4, var4.4{28} ; 75457FE9 u=sp+20.4    d=sp+20.4
7. 7 jnz    var4.4{28}, #0.4, @7     ; 75457FEC u=sp+20.4

8. 0 sub    var8.4{29}, #1.4, var8.4{30} ;
8. 1 jnz    var8.4{30}, #0.4, @6     ; 7545
```

top-level instruction

```
m_or  mop_d,mop_d,mop_l)
or    xdu.4(varC@3.1), ((xdu.4(varC@2.1) | ((ecx7.4{23} | (xdu.4(varC.1) <<I #8.1)) <<I #8.1)) <<I #8.1), result.4{26}

mop_d
xdu.4(varC@3.1)

mop_d
((xdu.4(varC@2.1) | ((ecx7.4{23} | (xdu.4(varC.1) <<I #8.1)) <<I #8.1)) <<I #8.1)

mop_l
result.4{26}

m_xdu(mop_l,mop_z)
xdu   varC@3.1, .4

sub instructions

m_shl mop_d,mop_n,mop_z)
shl   (xdu.4(varC@2.1) | ((ecx7.4{23} | (xdu.4(varC.1) <<I #8.1)) <<I #8

mop_l
varC@3.1

mop_d
(xdu.4(varC@2.1) | ((ecx7.4{23} | (xdu.4(varC.1) <<I #8.1)) <<I #8.1))

mop_n
#8.1

m_or  mop_d,mop_d,mop_z)
```

nested instructions (minsn_t)

Opaque Predicates

Opaque Predicates Summary

```
case m_or:
    iLocalRetVal = pat_OrAndNot(ins, blk);
    if (!iLocalRetVal)
        iLocalRetVal = pat_OrViaXorAnd(ins, blk);
    if (!iLocalRetVal)
        iLocalRetVal = pat_OrNegatedSameCondition(ins,
    if (!iLocalRetVal)
        iLocalRetVal = pat_LogicAnd1(ins, blk);
    if (!iLocalRetVal)
        iLocalRetVal = pat_MulSub2(ins, blk); // added
    break;
```

- `optinsn_t::func` replaces an opaque predicate pattern with another expression
  - called from `MMAT_ZERO` to `MMAT_GLBOPT2`
- ANEL samples require 2 more patterns and data-flow tracking

Pattern1: ~(x * (x - 1)) | -2

- In the example below,
  - `dword_745BB58C` = either even or odd
  - `dword_745BB58C * (dword_745BB58C - 1)` = always even
  - the lowest bit of the negated value becomes 1
  - OR by -2 (`0xFFFFFFFE`) will always produce the value -1
- The pattern `x * (x-1)` will be replaced with 2

```
13  v2 = ~((_BYTE)dword_745BB58C * ((_BYTE)dword_745BB58C - 1)) | 0xFFFFFFFE;
14  v3 = 0x7157F526;
```

Pattern2: read-only global variable >= 10 or < 10

- dword_72DBB588 is always 0
  - without a value (will be initialized with 0)
  - only read accesses
- the pattern matching function replaces the global variable with 0
- other variants
  - the variable - 10 < 0
  - the immediate value can be different, not 10 (e.g., 9)

```
22  if ( dword_72DBB588 >= 10 )
23    v3 = v1;
24  v8 = dword_72DBB588 < 10;
```

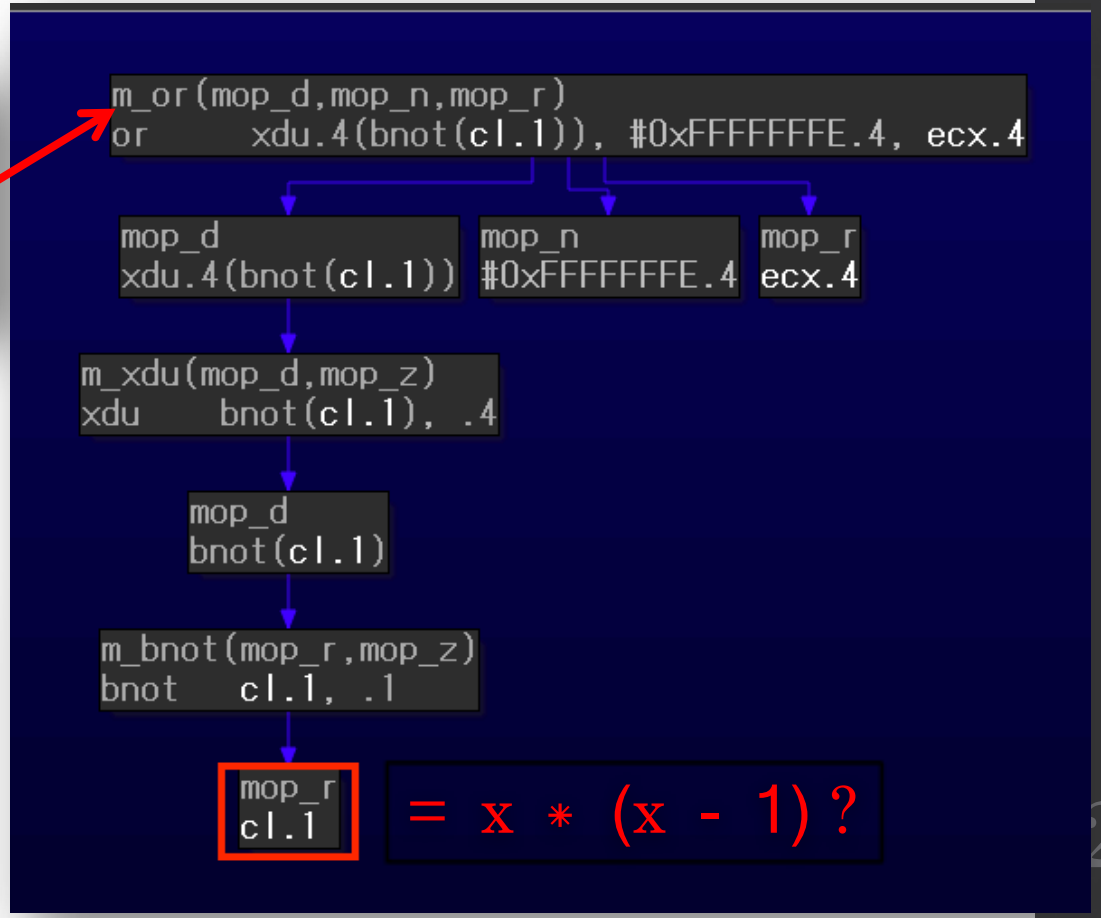
Data-flow tracking for the patterns

- trace back the `minsn_t` / `mblock_t` linked lists

```
72  v5 = dword_72DBB504 * (dword_72DBB504 - 1);
73  v6 = 2017184256;
74  while ( 2 )
75  {
76    v7 = (~(_BYTE)v5 | 0xFFFFFFFE) == -1;
```

```
v7 = dword_1007B4A4;
v11 = dword_1007B4A4 - 1;
_58:
v19 = ~(v7 * v11) | 0xFFFFFFFE;
```

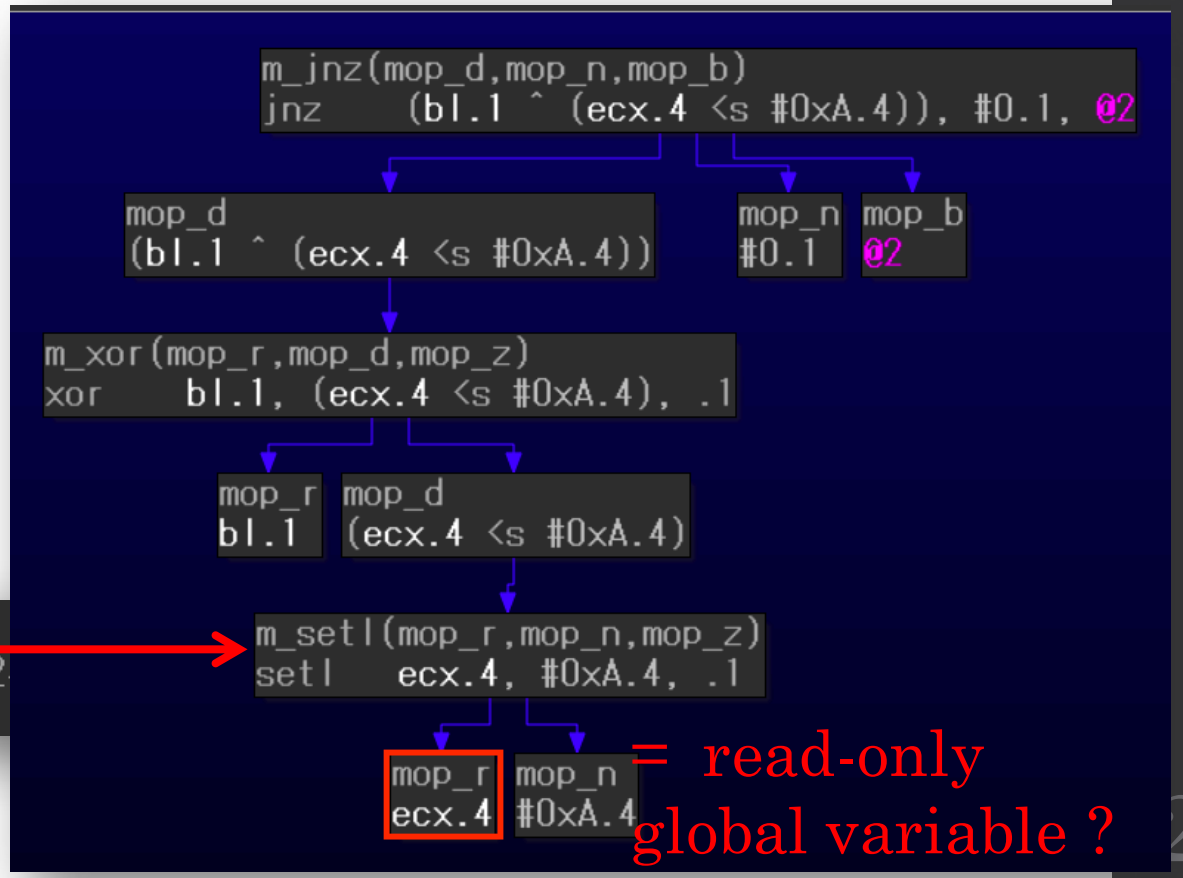
```
v11 = dword_1007B56C;
v28 = -252310204;
v12 = dword_1007B56C;
v76 = *(*v32 + 68) == 1;
9:
v5 = (~(v11 * (v12 - 1)) | 0xFFFFFFFE) == -1;
```



Data-flow tracking for the patterns (Cont.)

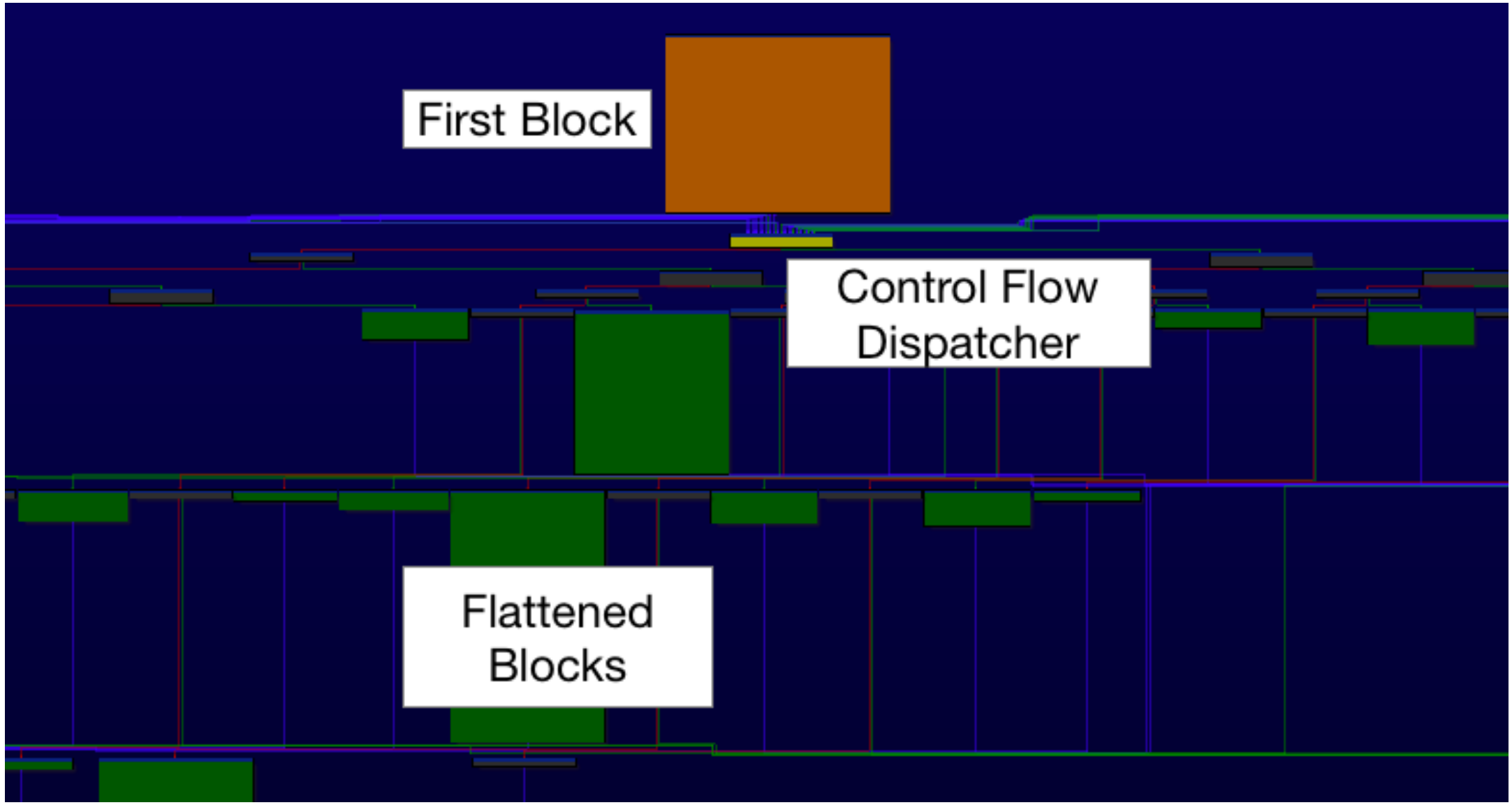
- `optinsn_t::func` passes a null `mblock_t` pointer if an instruction is not **top-level**
  - An additional code traces from `jnz` then passes the pointer to `setl`

```
17    v2 = dword_72DBB5F8;
18    for ( i = 0x1C5CF5E5; v2 < 10 && i > 0x856E12
19    {
```

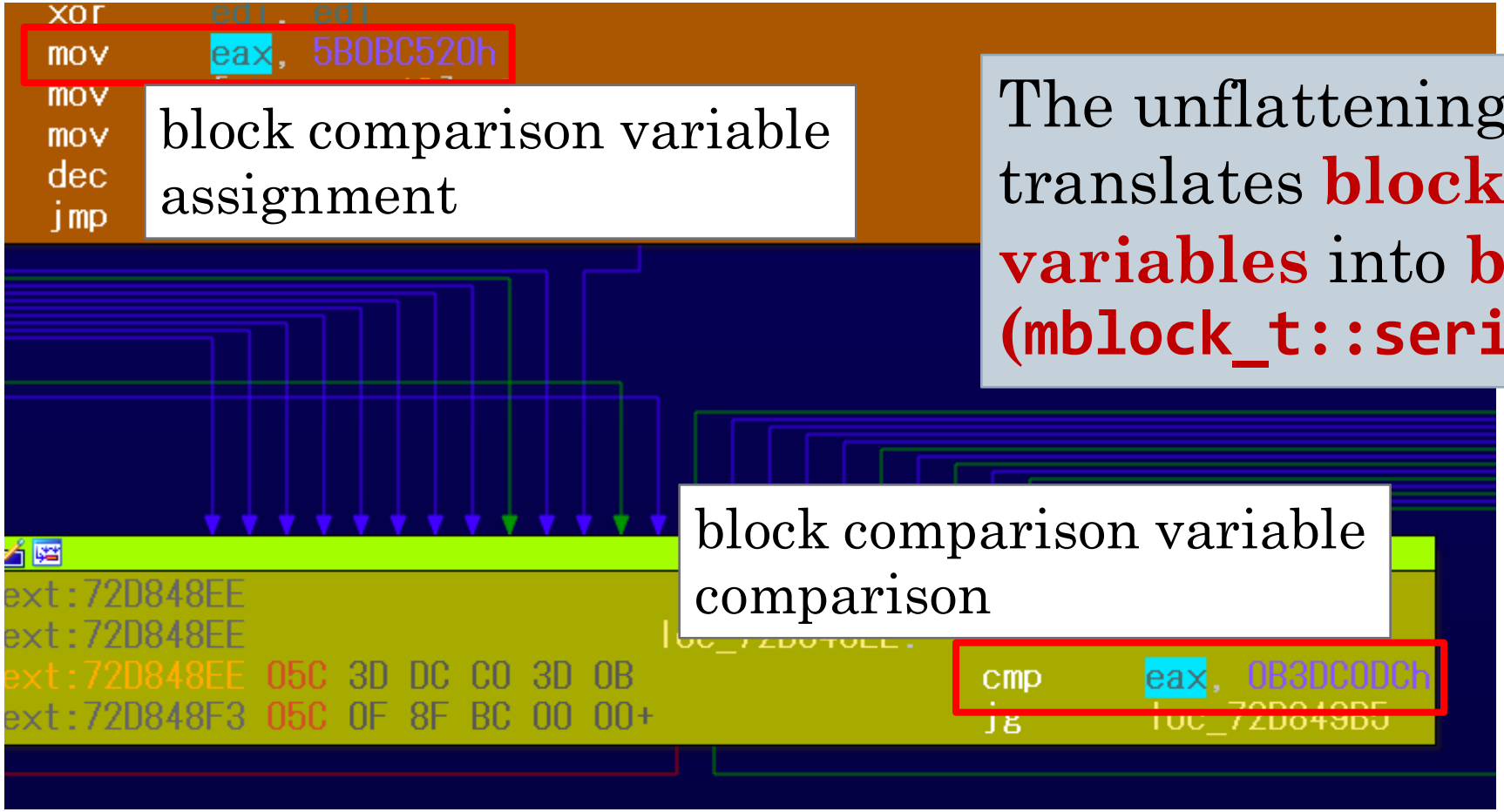


Control Flow Flattening

Control Flow Flattening: Summary



Control Flow Flattening: block comparison variable



The unflattening code translates **block comparison variables** into **block numbers (`mblock_t::serial`)**

Control Flow Flattening: Modifications

- three main modifications
  - Unflattening in multiple maturity levels
  - Control flow handling with multiple dispatchers
  - Implementation for various jump cases

Unflattening in Multiple Maturity Levels

```
  v1 = 0x7157F526;
always_true = always_minus1 == -1;
result = value;
b_cmp = 0x4624F47C;
if ( !always_true )
  v3 = v1;
if ( dword_72DBB588 >= 10 )
  v3 = v1;
v8 = dword_72DBB588 < 10;
while ( 1 )
{
  while ( b_cmp <= 0x4624F47B )
  {
    if ( b_cmp == 0xC504A26C )
    {
      b_cmp = v3;
    }
```

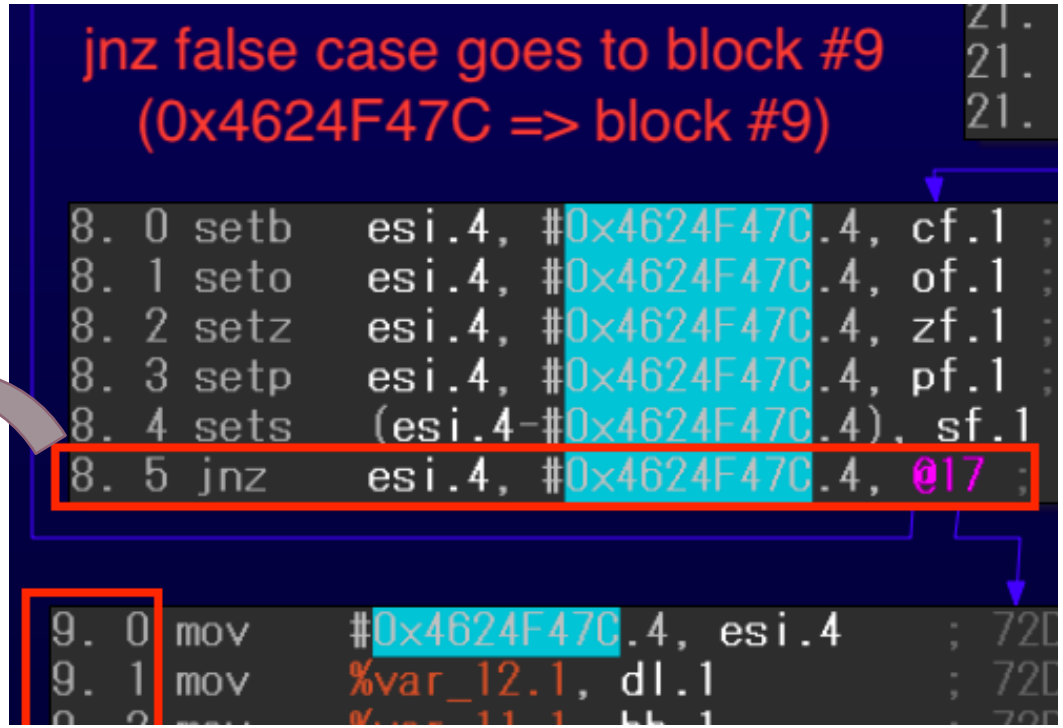
- The original implementation works in `MMAT_LOCOPT`
  - due to "Odd Stack Manipulations" obfuscation
- I had to unflatten the ANEL code in later maturity levels
  - The block comparison variable heavily depends on opaque predicate conditions

Unflattening in Multiple Maturity Levels (Cont.)

- The loop becomes simpler once opaque predicates are broken
- Unflattening in later maturity levels makes another problem

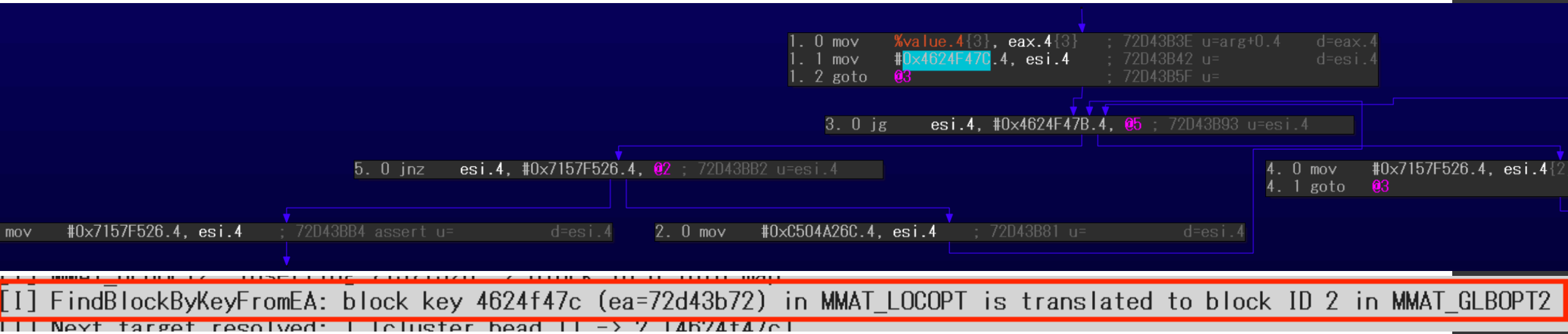
```
result = value;
for ( b_cmp = 0x4624F47C; ; b_cmp = 0xC504A26C )
{
  while ( b_cmp <= 0x4624F47B )
    b_cmp = 0x7157F526;
  if ( b_cmp == 0x7157F526 )
    break;
}
return result;
}
```

In MMAT_LOCOPT,
The block comparison variable
0x4624F47C is translated into block #9



Unflattening in Multiple Maturity Levels (Cont.)

- The block will be eliminated in later maturity levels
- The modified code
  - Links between block comparison variables and block addresses in MMAT_LOCOPT
  - Guesses the block numbers in later maturity levels by using each block and instruction addresses



Control Flow Handling with Multiple Dispatchers

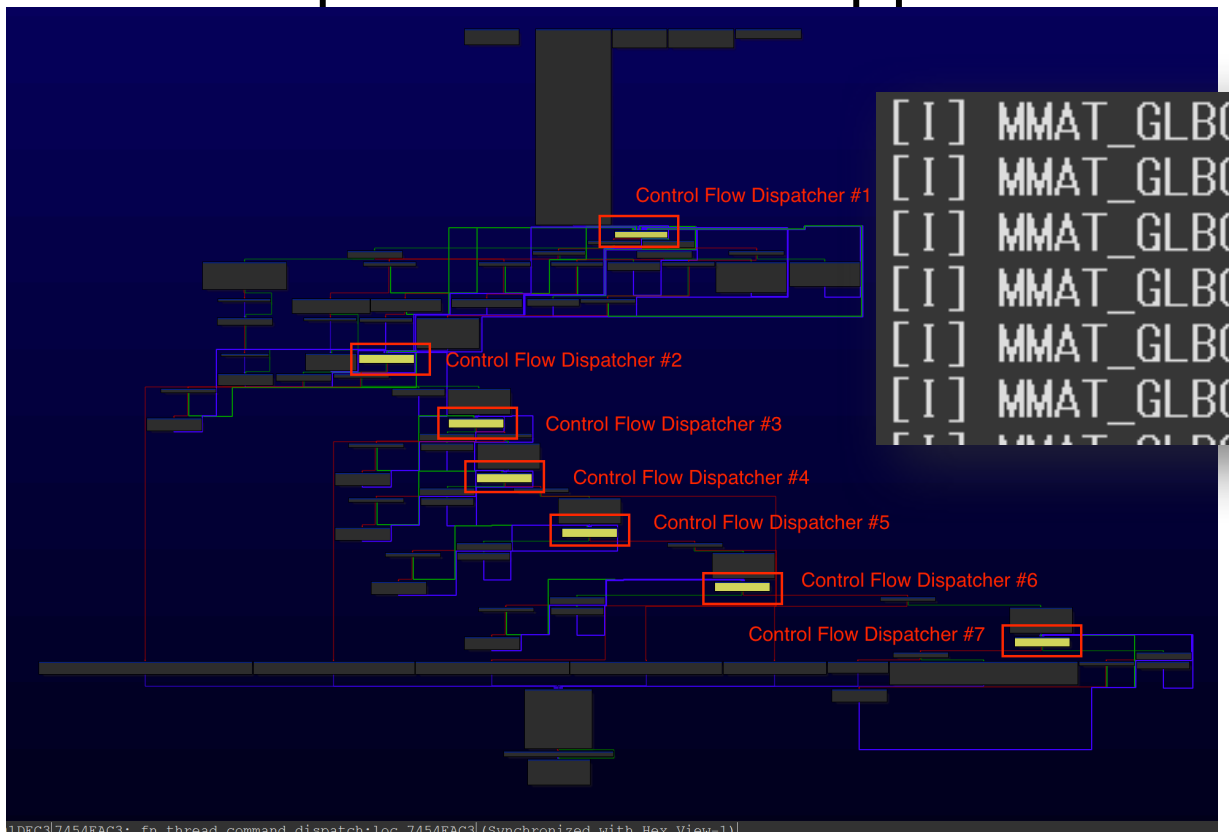
- The original implementation assumes an obfuscated function has only one control flow dispatcher
- Some functions in the ANEL sample have multiple dispatchers
  - up to seven dispatchers in one function

Control Flow Handling with Multiple Dispatchers (Cont.)

- The modified code
  - catches the `hxe_prealloc` event then calls the `optblock_t::func`
    - This event occurs several times in `MMAT_GLBOPT1` and `MMAT_GLBOPT2`
  - utilizes different algorithms
    - control flow dispatcher / first block detection
    - block comparison variable validation

Control Flow Handling with Multiple Dispatchers (Cont.)

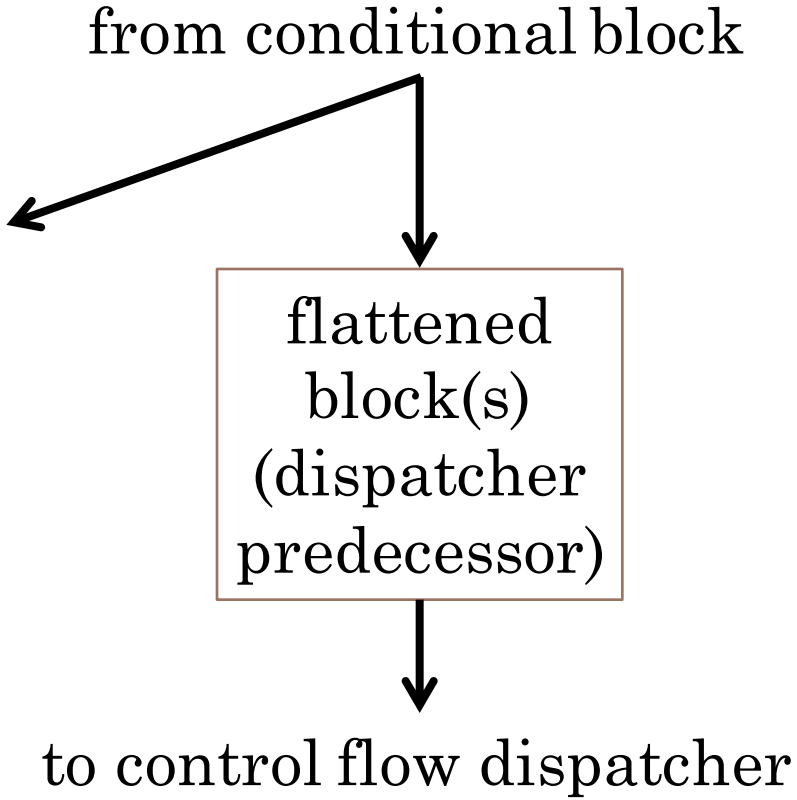
- The modified code detects block comparison variable duplications and applies the most likely variable



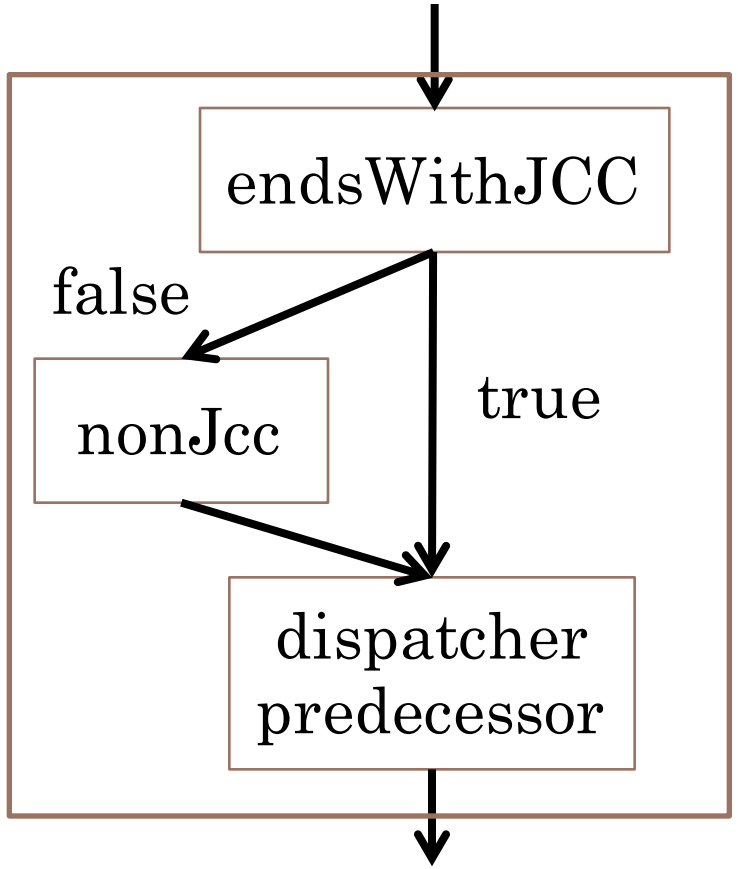

```
[I] MMAT_GLBOPT1: Inserting 27319357 -> block ID 13 into map
[I] MMAT_GLBOPT1: Inserting 4b5f6a19 -> block ID 10 into map
[I] MMAT_GLBOPT1: Inserting 258f71af -> block ID 15 into map
[I] MMAT_GLBOPT1: Ignoring 27319357 -> block ID 23 due to mu
[I] MMAT_GLBOPT1: Ignoring 4b5f6a19 -> block ID 20 due to mu
[I] MMAT_GLBOPT1: Ignoring 27319357 -> block ID 33 due to mu
```

Implementation for Various Jump Cases: The Originals

(1) goto case for normal block

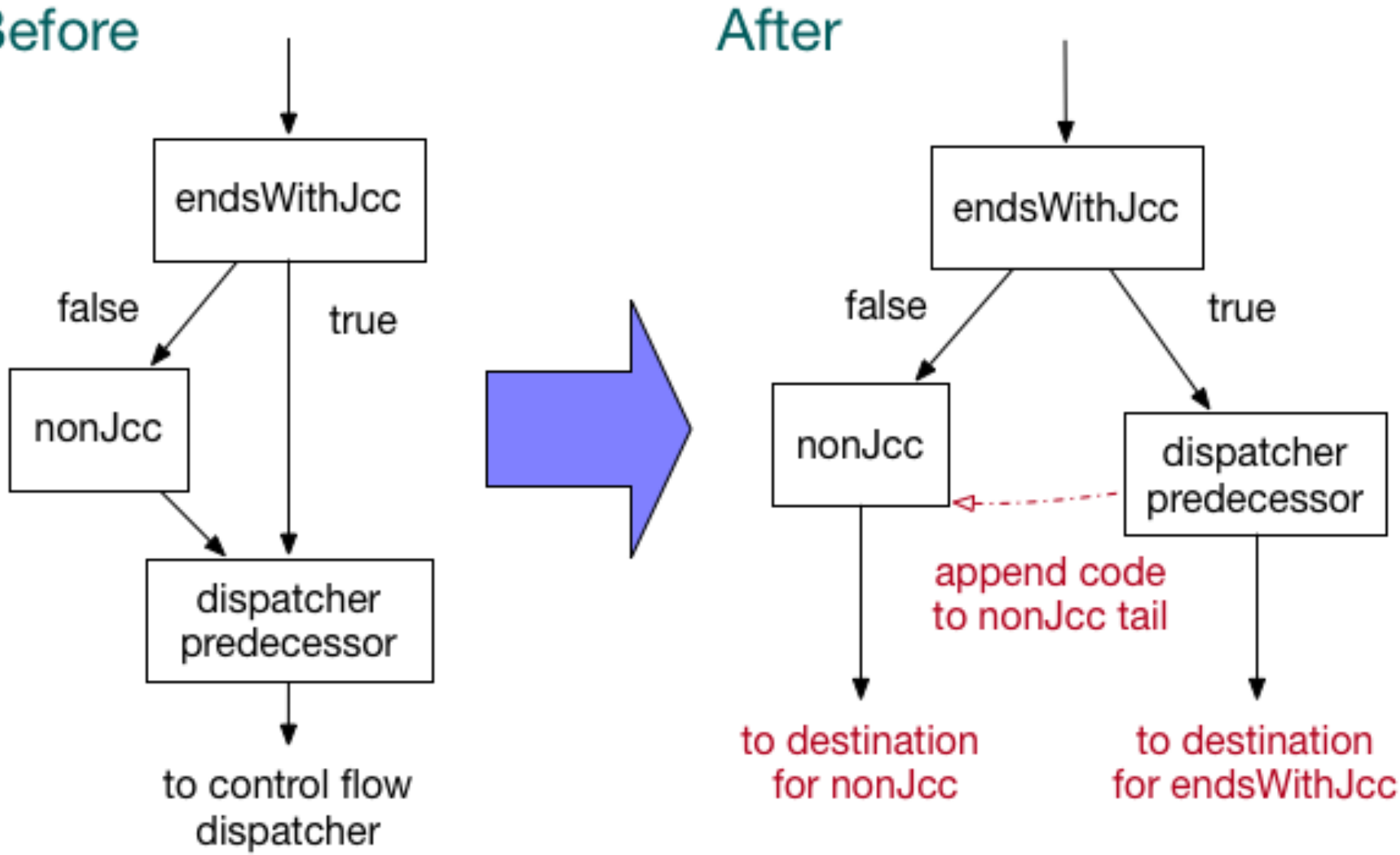


(2) conditional jump case for flattened if-statement block

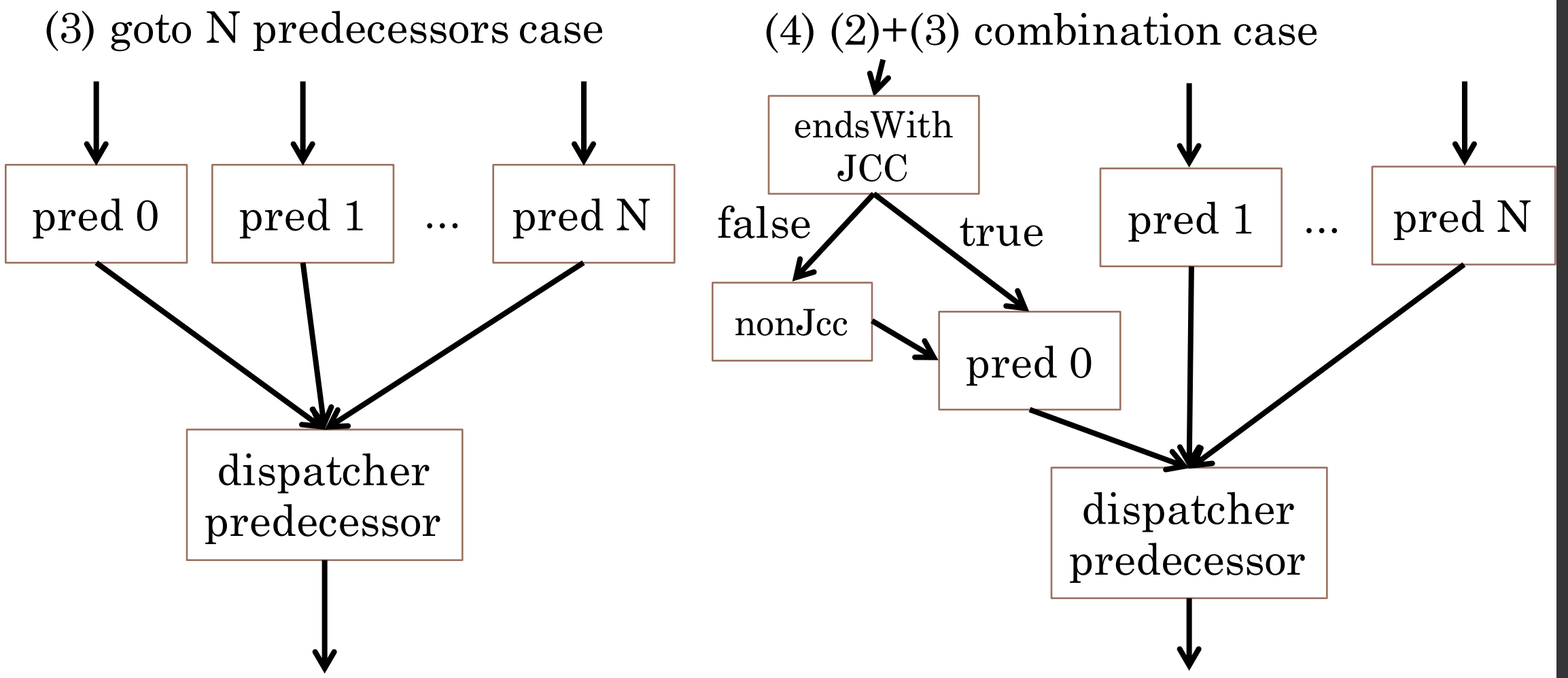


Implementation for Various Jump Cases: The Originals (Cont.)

(2)

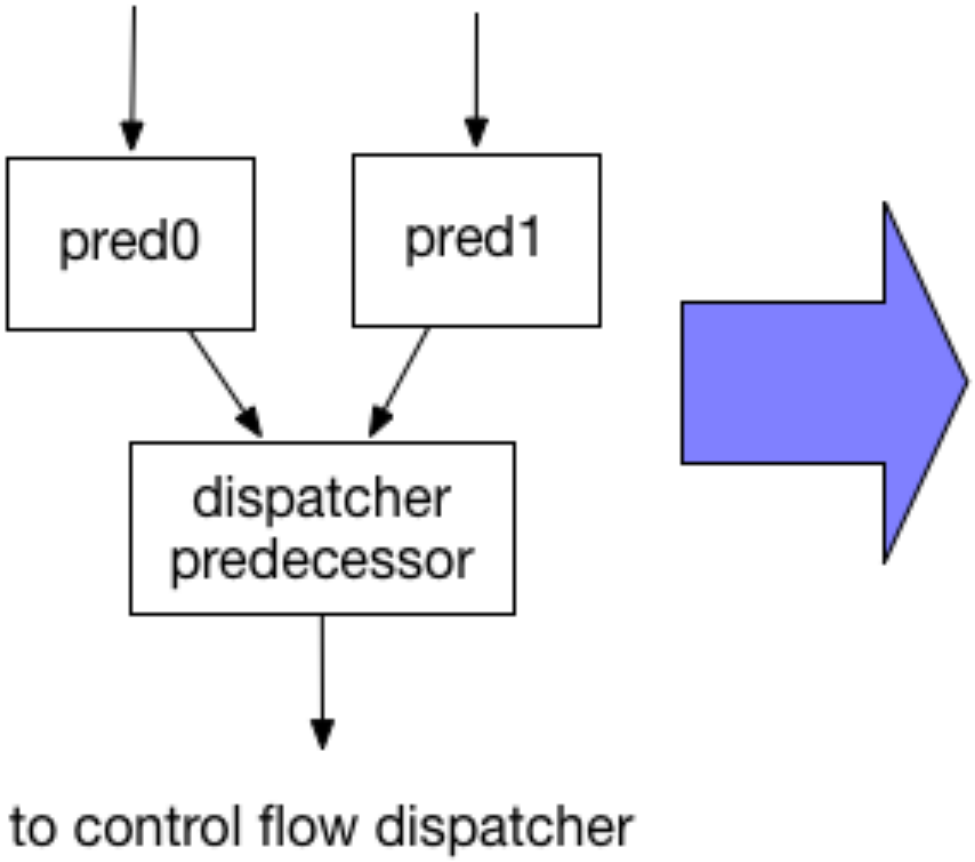


Implementation for Various Jump Cases: The Additions

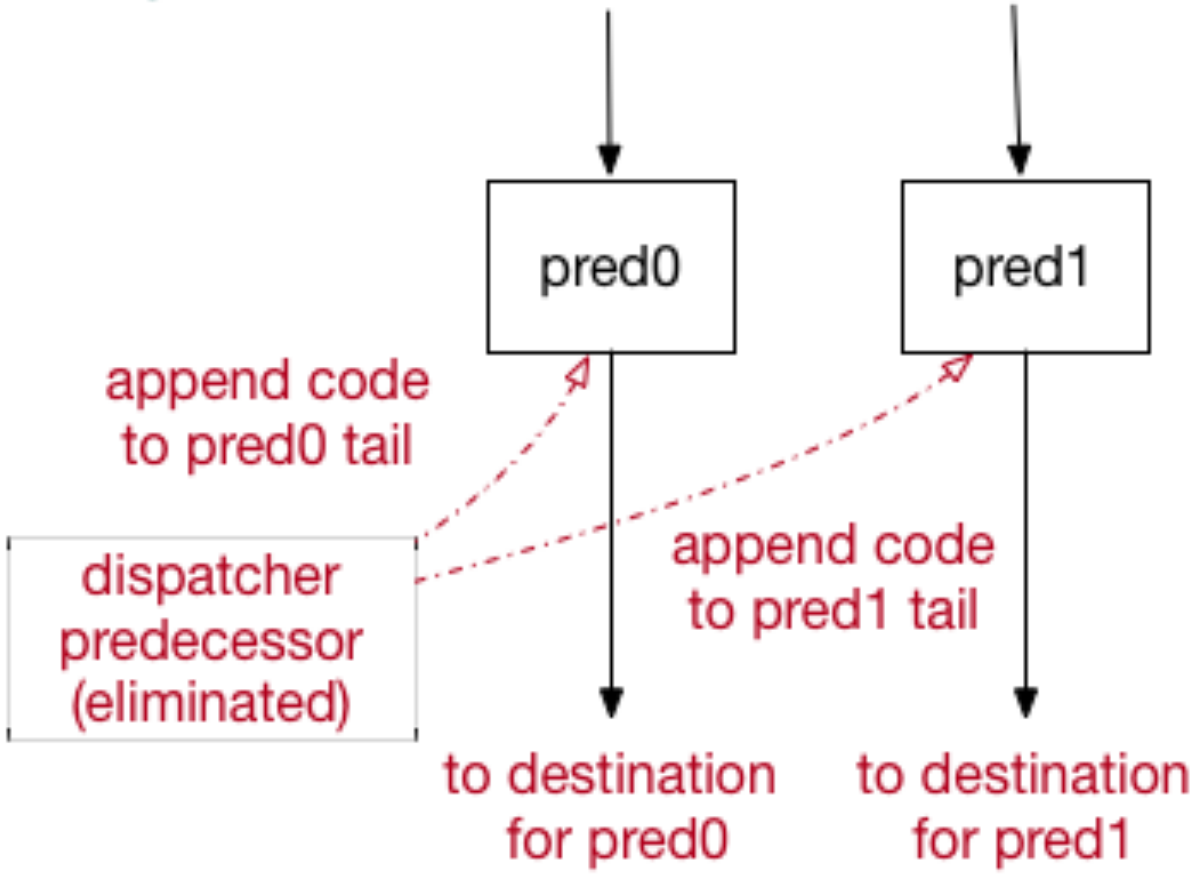


Implementation for Various Jump Cases: The Additions (Cont.)

(3) Before

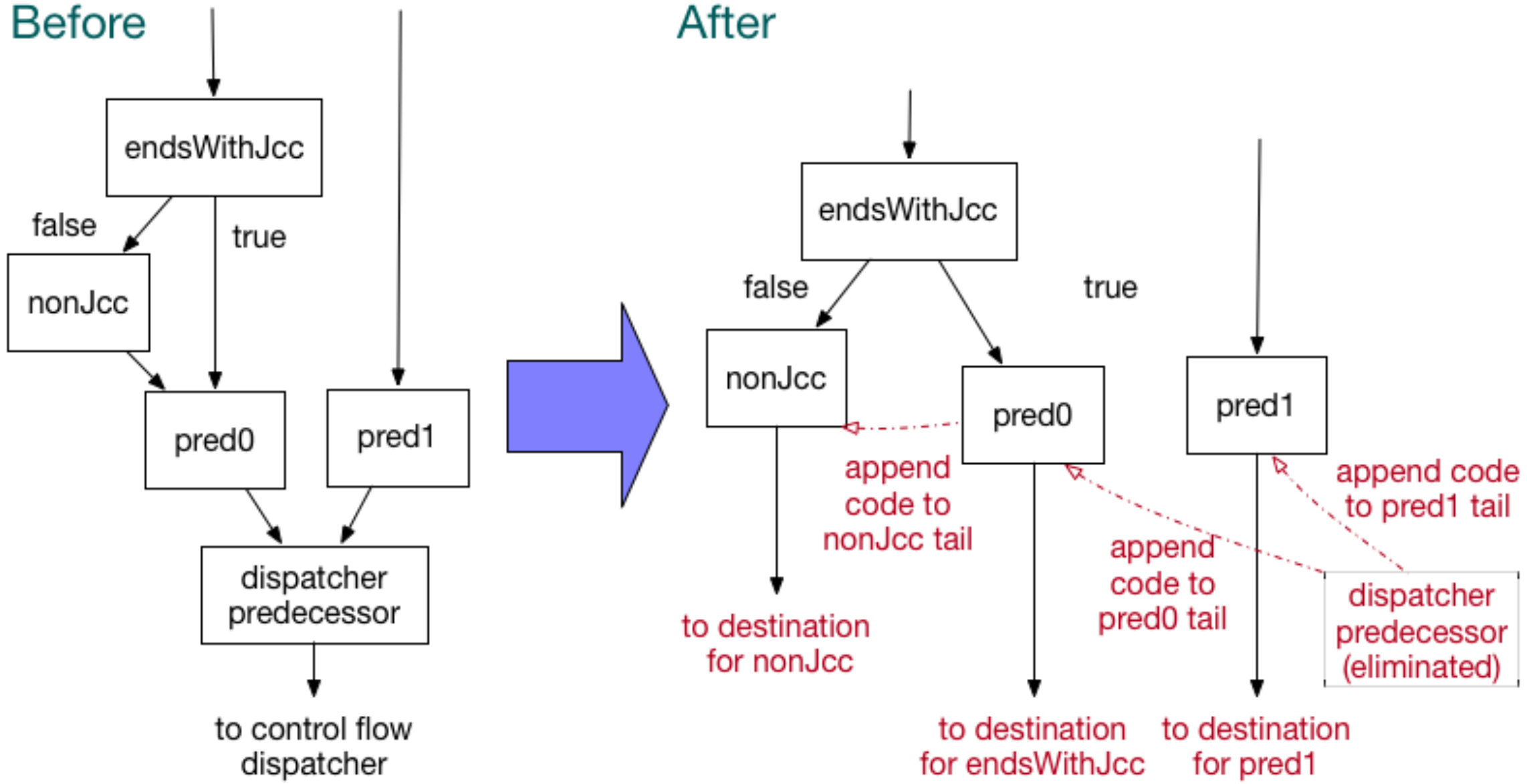


After

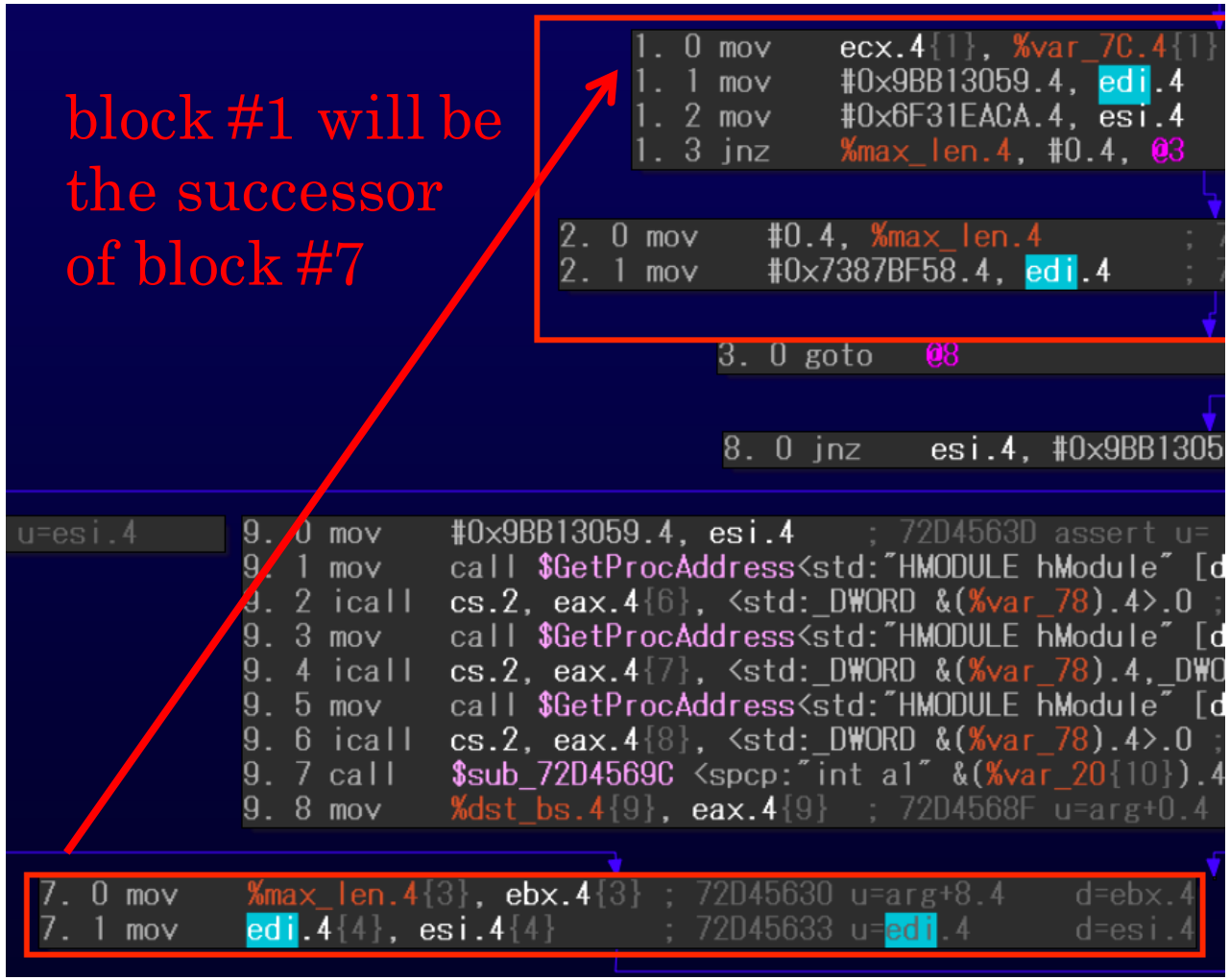


Implementation for Various Jump Cases: The Additions (Cont.)

(4)



Implementation for Various Jump Cases: The Additions (Cont.)



- (5) Block comparison variables are assigned in the first blocks
  - The modified code reconnects first blocks as successors of the flattened block
- I saw up to three assignments of the case in one function

IDA 7.2 Issues and 7.3 Improvements

Evaluation on IDA 7.2

- Tested ANEL samples
  - 5.4.1 payload [1]
    - 3d2b3c9f50ed36bef90139e6dd250f140c373664984b97a97a5a70333387d18d
  - 5.5.0 rev1 loader DLL [6]
    - f333358850d641653ea2d6b58b921870125af1fe77268a6fdfeda3e7e0fb636d
- The modified tool could deobfuscate 92% of the obfuscated functions that we encountered in the 5.4.1 payload

Evaluation on IDA 7.2 (Cont.)

- The causes of the failures
  - The next block number guessing algorithm failed

resolved in this case

  - Propagations of opaque predicates deobfuscation failed
  - No method to handle a conditional jump of a dispatcher predecessor with multiple predecessors

resolved in IDA 7.3

IDA 7.3: Propagation of Opaque Predicates Deobfuscation

7.2

```
true1 = 1;
true2 = 1;
v2 = 0x1D3E02CA;
if ( true2 )
  v2 = 0xC1A18C30;
if ( !true1 )
  v2 = 0x1D3E02CA;
if ( true2 == true1 && v2 > 0x1D3E02C9 )
{
  savedregs = 0x1D3E02CA;
  v5 = fn_get_ptr_from_bs(src_bs);
  StrToIntA(v5);
  v6 = _gmtime64(&v8);
  strftime(&v9, 0x50u, "%a, %d %b %Y %X",
  fn_w_bs_make_from_str(dst_bs, &v9);
}
savedregs = 0xC1A18C30;
v3 = fn_get_ptr_from_bs(src_bs);
StrToIntA(v3);
v4 = _gmtime64(&v8);
strftime(&v9, 0x50u, "%a, %d %b %Y %X", v
fn_w_bs_make_from_str(dst_bs, &v9);
return dst_bs;
```

aliased stack slots

always 0xC1A18C30 (signed)

7.3

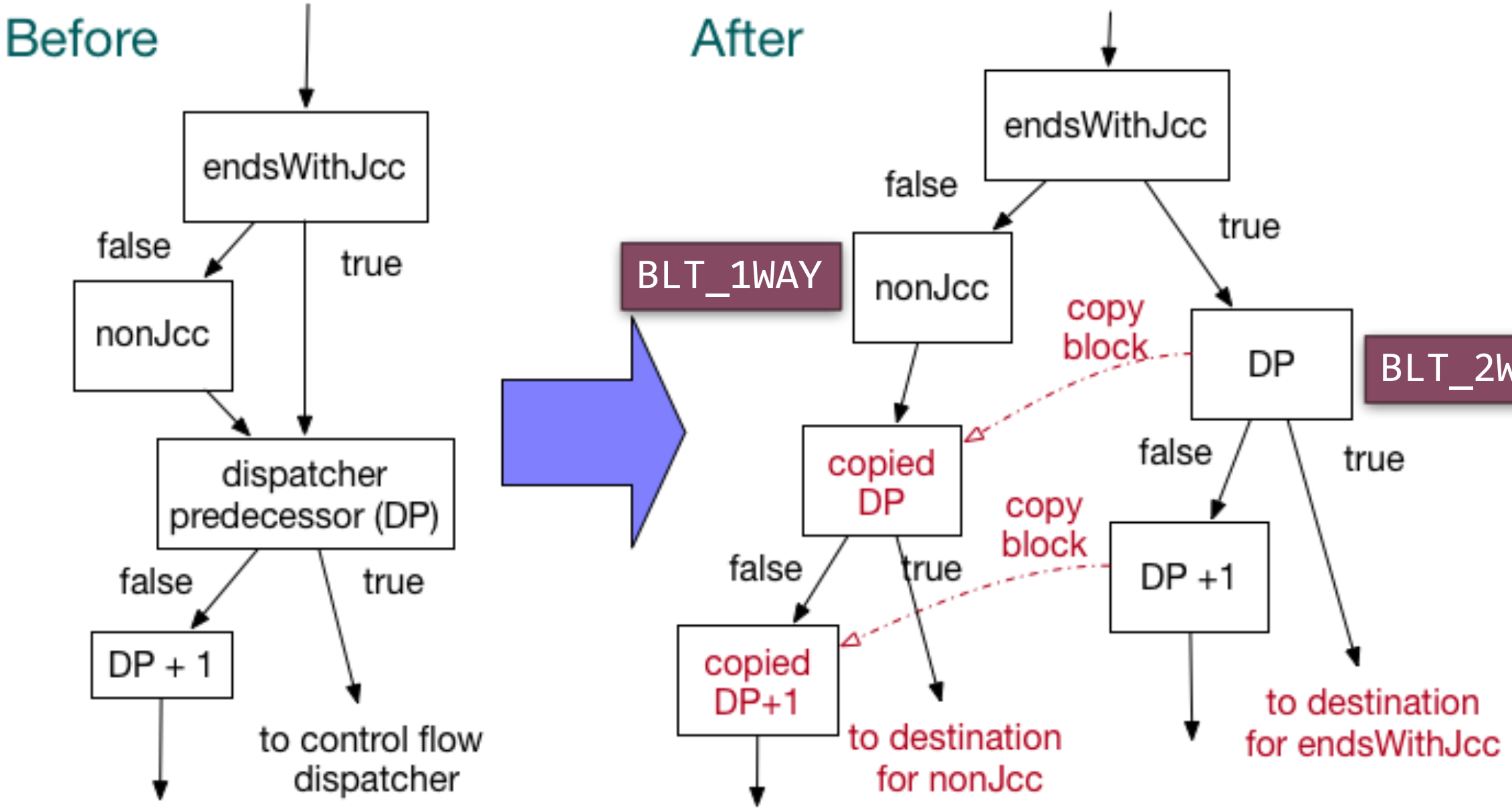
```
true1 = 1;
true2 = 1;
savedregs = 0xC1A18C30;
v2 = fn_get_ptr_from_bs(src_bs);
StrToIntA(v2);
v3 = _gmtime64(&v5);
strftime(&v6, 0x50u, "%a, %d %b %Y %X
fn_w_bs_make_from_str(dst_bs, &v6);
return dst_bs;
```

IDA7.3: Handling a Conditional Jump of a Dispatcher Predecessor

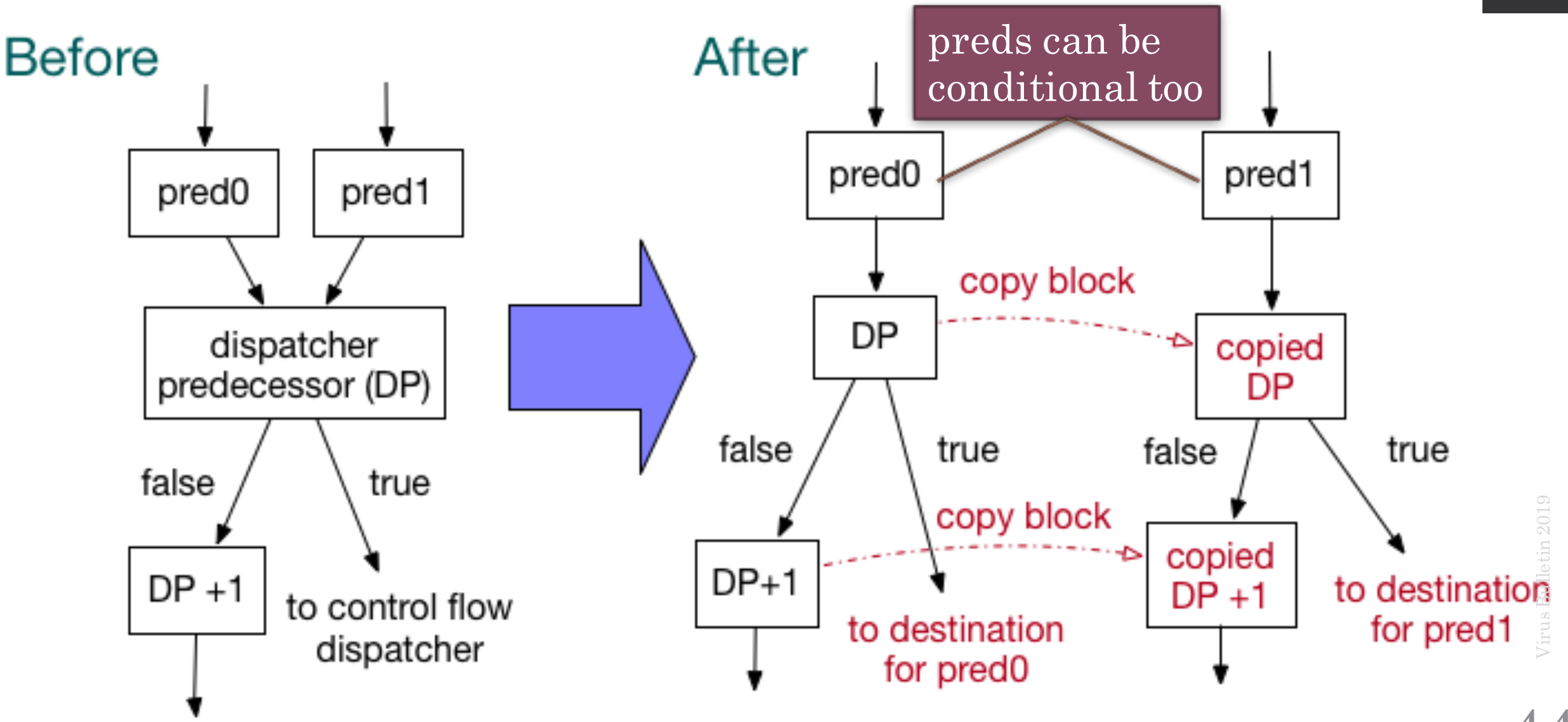
- All jump cases (1)-(5) can be conditional
  - (2)-(4) cases require a mblock_t duplication
- IDA 7.3 provides the option
  - clear the flag `MBA2_NO_DUP_CALLS`
  - use `mbl_array_t::insert_block` API then copy instructions and other information
  - adjust destinations of the blocks passing a control to the exit block whose block type is `BLT_STOP`

```
// enable mblock_t copy for later maturity levels
mba->clr_mba_flags2(MBA2_NO_DUP_CALLS);
```

Conditional Jump Case (2)

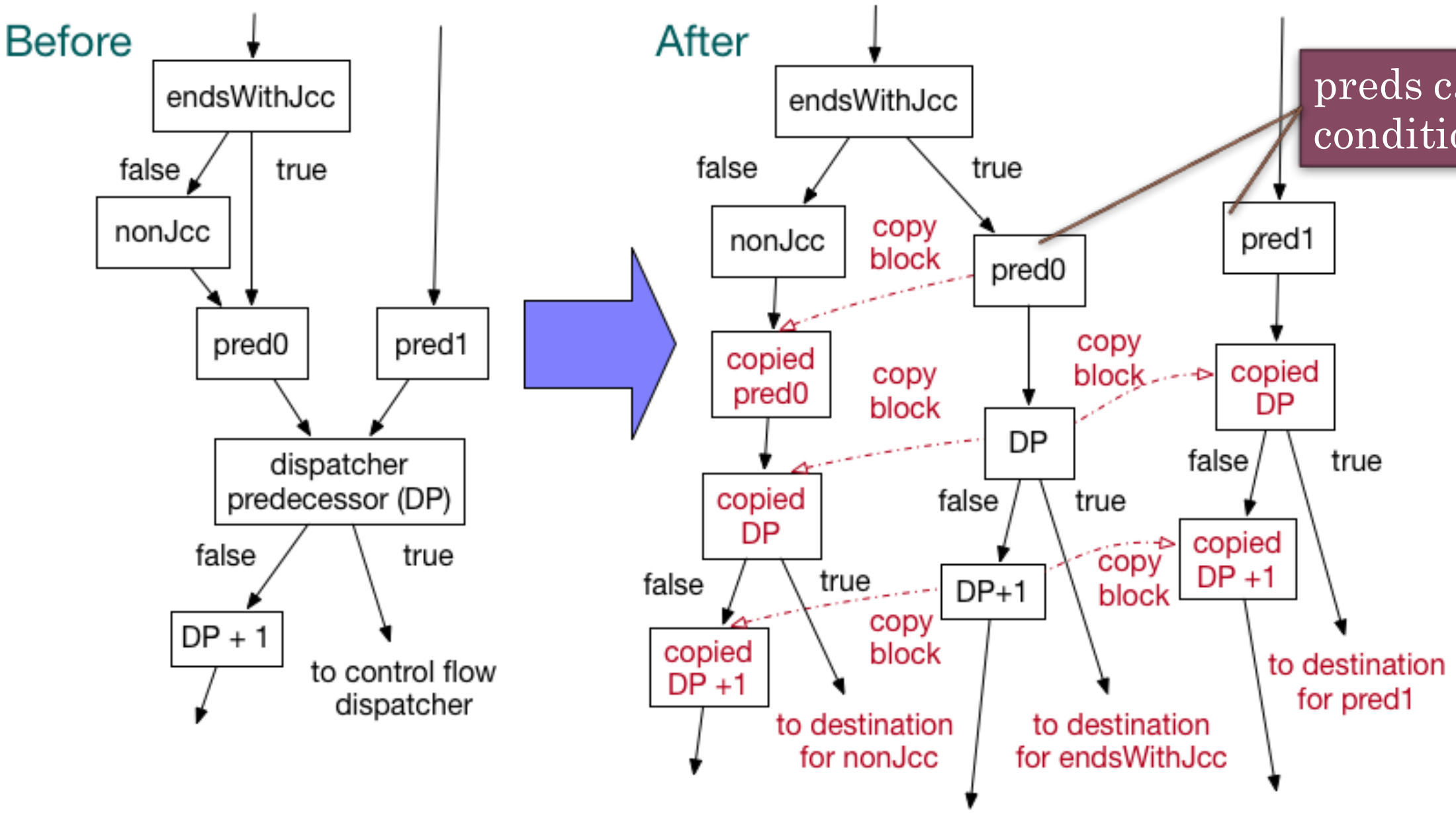


Conditional Jump Case (3)



Conditional Jump Case (4)

not seen in the tested samples :-)



Workaround in Control Flow Unflattening Failure

- The plugin execution with `0xdead` deobfuscates only opaque predicates in the current selected function

```
1 size_t __cdecl fn_cause_
2 {
3   bool v5; // [esp+57h]
4
5   if ( !a1 )
6     return a3;
7   v5 = (a3 & 0xF) != 0;
8   while ( !v5 )
9     ;
10  return 0;
11 }
```

```
idc.load_and_run_plugin("HexRaysDeob", 0xdead)
```

```
idc.load_and_run_plugin("HexRaysDeob", 0xf001)
```

```
51 while ( 1 )
52 {
53   while ( 1 )
54   {
55     while ( 1 )
56     {
57       while ( 1 )
58       {
59         while ( 1 )
60         {
61           v15 = v8;
62           if ( v8 <= 21690082 )
63             break;
64           if ( v8 > 1127530844 )
65           {
66             if ( v8 <= 1518054240 )
67             {
68               if ( v8 > 1278155936 )
69               {
70                 if ( v8 == 1278155937 )
71                 {
72                   memset(&v23, v39, v3
73                   v25 = &v26[-52857152
74                   v14 = memcmp(&v23, v2
75                   v8 = -35098432;
```

Wrap-up

Wrap-up

- The compiler-level obfuscations are starting to be observed in the wild
  - The automated deobfuscation is needed
- The modified code is available publically [7]
  - 1570 insertions(+), 450 deletions(-)
  - It works for almost every obfuscated function of APT10 ANEL on IDA 7.3

Acknowledgement

- Hex-Rays
- Rolf Rolles
- TAU members
  - especially Jared Myers and Brian Baskin

References

- [1] https://www.fireeye.com/blog/threat-research/2018/09/apt10-targeting-japanese-corporations-using-updated-ttps.html
- [2] https://jsac.jpcert.or.jp/archive/2019/pdf/JSAC2019_6_tamada_jp.pdf
- [3] http://www.hexblog.com/?p=1248
- [4] https://github.com/RolfRolles/HexRaysDeob
- [5] https://www.hexblog.com/?p=1232
- [6] https://www.secureworks.jp/resources/at-bronze-riverside-updates-anel-malware
- [7] https://github.com/carbonblack/HexRaysDeob
- [8] https://www.carbonblack.com/2019/02/25/defeating-compiler-level-obfuscations-used-in-apt10-malware/

Questions?

- [Q1] What's the obfuscating compiler?
  - [A1] Not sure but it may be Obfuscator-LLVM
- [Q2] This tool works for other samples with similar obfuscations?
  - [A2] Yes only if
    - Q1 is resolved
    - the compiler algorithm and implementation have been thoroughly investigated