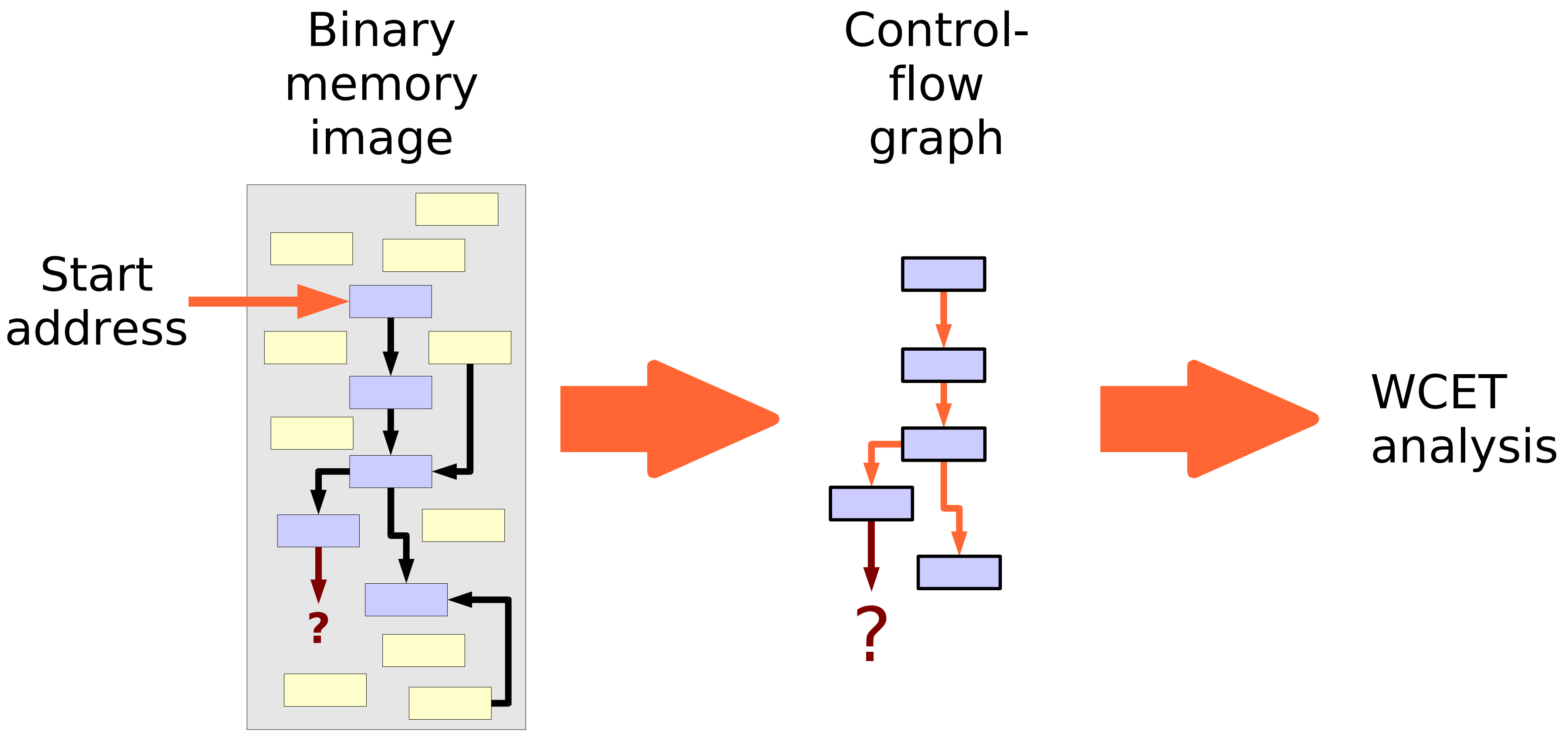
Analysing Switch-Case Tables by Partial Evaluation

Niklas Holsti

Tidorum Ltd

www.tidorum.fi

From binary file to control-flow graph



Problem: dynamic transfer of control, DTC

for example jump via register

Overview

- Analysing DTC from a switch-case statement
- When compiled into a switch table interpreted by a switch handler routine
- Partial evaluation (PE) of switch handler
- Example

Switch tables and switch handlers

Switch-case statement

```
switch (k) {
    case 4: ...
    case 8: case 9: case 11: ...
    default: ... }
```

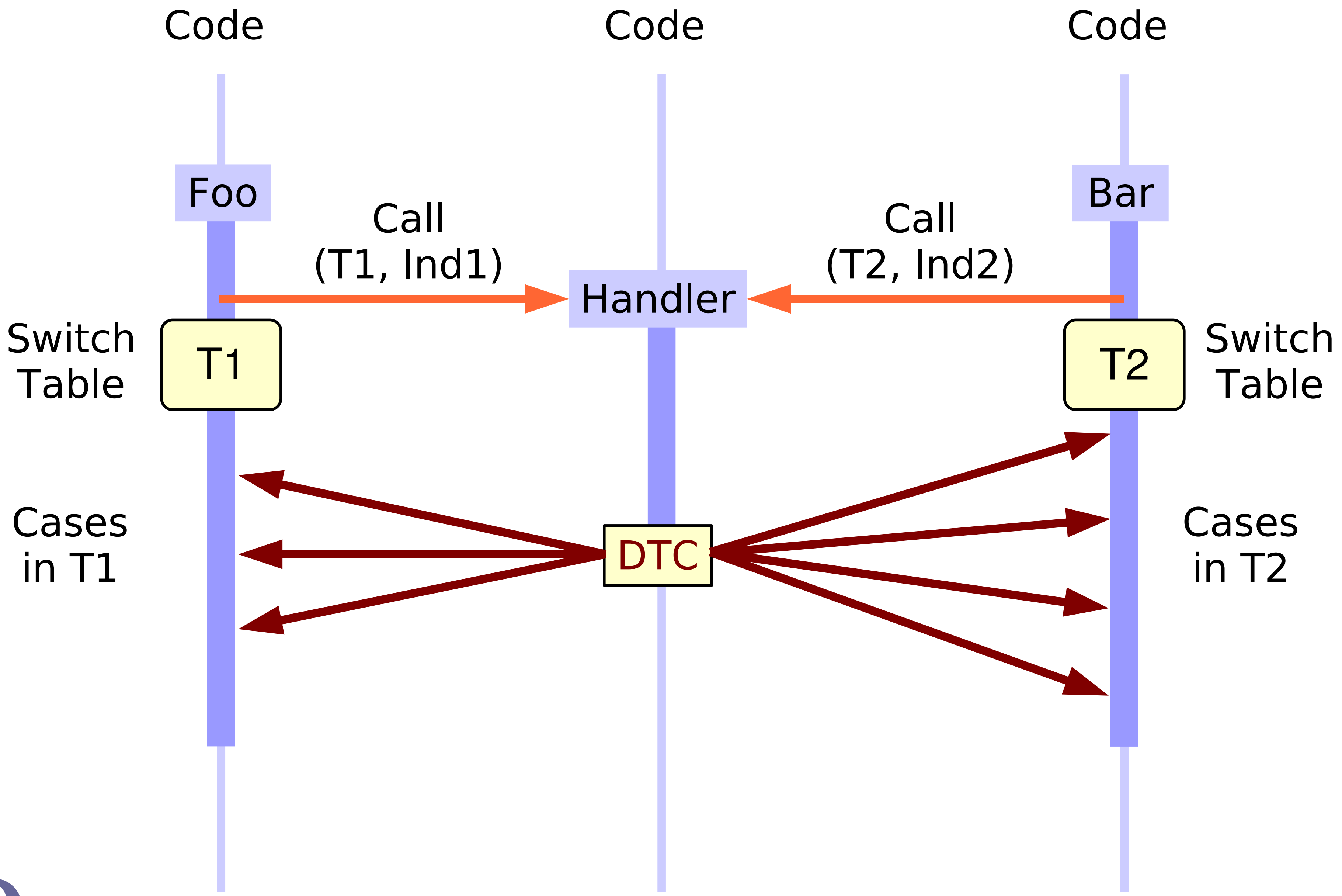
Switch table

A constant table that maps index value to code address:

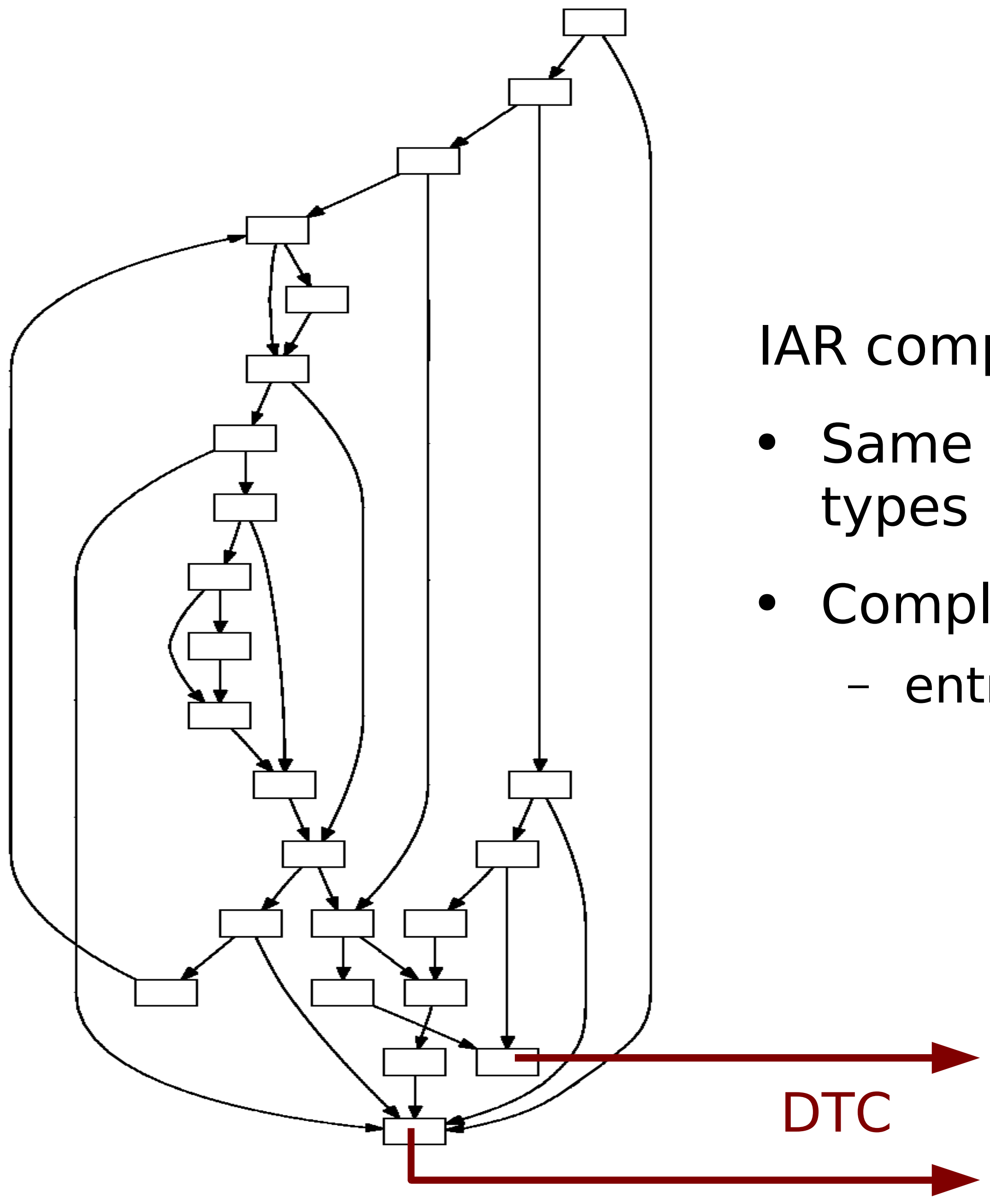
4	→ A: <case 4>
8, 9, 11	→ B: <case 8, 9, 11>
others	→ C: <default>

- Various forms of switch tables
  - depending on compiler, index type, dense/sparse, ...
- Compiler generates:
  - switch table T
  - call or jump to switch handler (Table ⇒ T, Index ⇒ k)
- Switch handler
  - looks up Index in Table
    - jumps to that case using DTC

One handler – many switches



A real switch handler

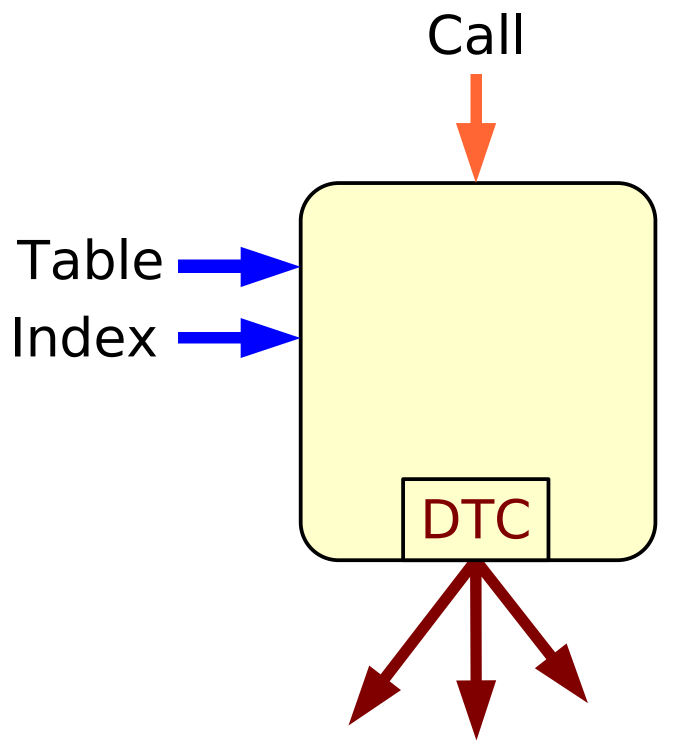


IAR compiler for Atmel AVR

- Same handler for many index types (8, 16, 32 bits)
- Complex table structure
  - entries of variable length

Partial Evaluation of switch handlers

Switch handler for any Table and any Index



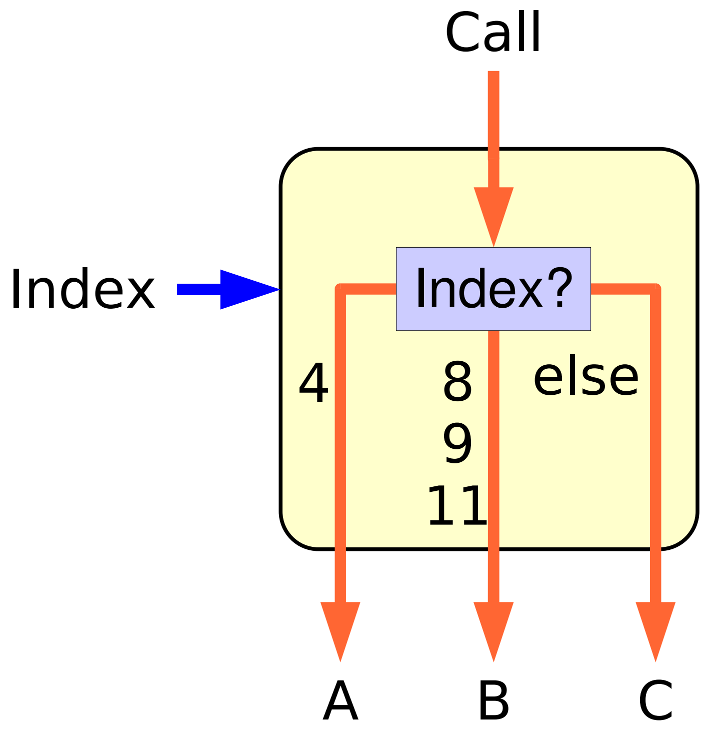
Addresses defined in the Table

Partially evaluate the switch handler with respect to the known switch table T for a given switch-case statement

Table T

Index	Addr
4	A
8,9,11	B
default	C

Residual switch handler for table T and any Index



DTC is resolved into static jumps

Eureka

- The analysis “runs” the switch handler
- The switch handler itself decodes the switch table

The example

- Here shown on a symbolic level
  - paper shows AVR machine code

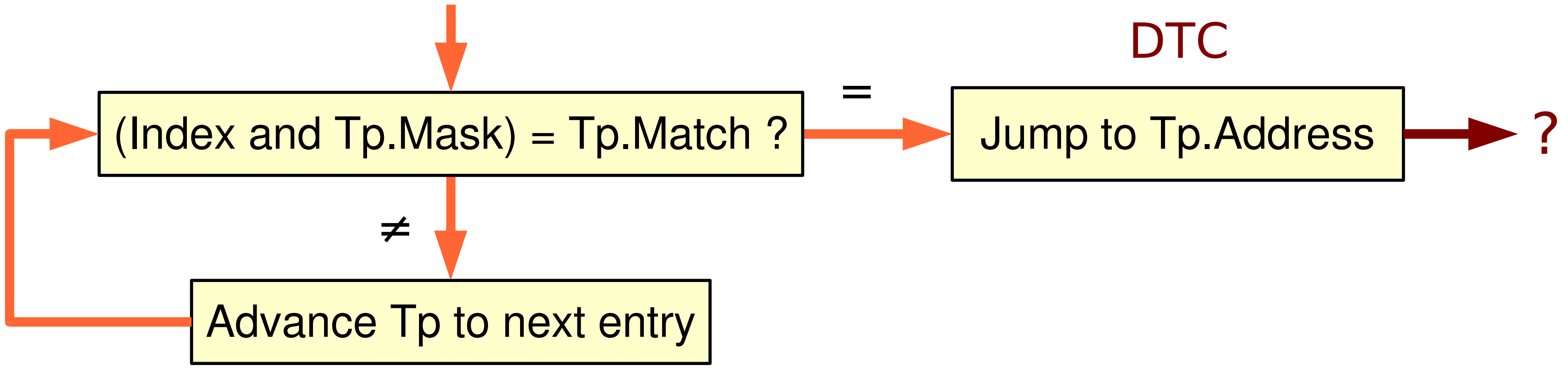
- Partial Evaluation as implemented in Bound-T
  - on the fly while building flow-graph
  - data state: some variables bound to constants

Simple 8-bit switch table & handler

```
switch (k) {
    case 4: ...
    case 8: case 9: case 11: ...
    default: ... }
```

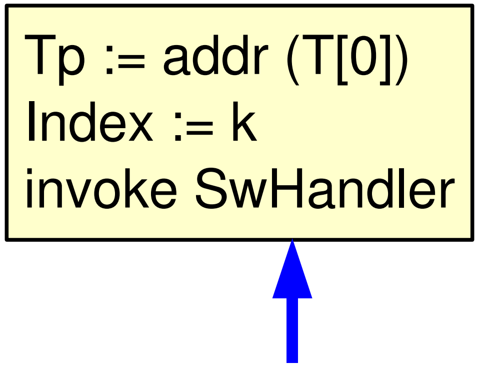
Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

SwHandler parameters:
- pointer Tp to switch table
- 8-bit Index (k) of switch-case

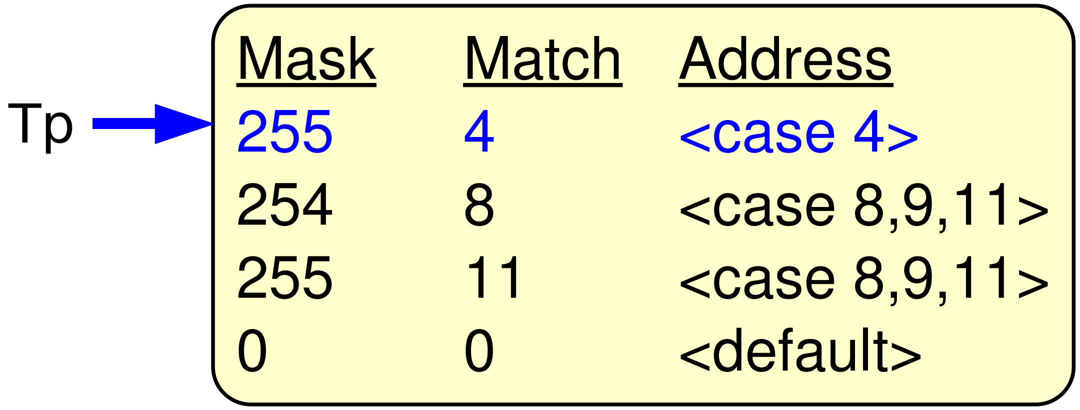


0. Detect invocation of switch handler

Flow graph (1 node so far)



Aha!
Entering switch handler!



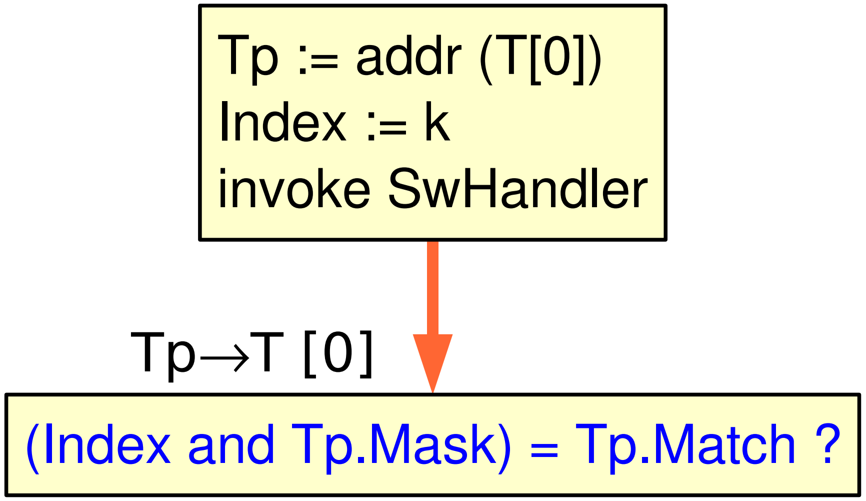
Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

- Start PE with initial data-state

  $Tp \rightarrow T\ [0]$

- Expand SwHandler “in-line”

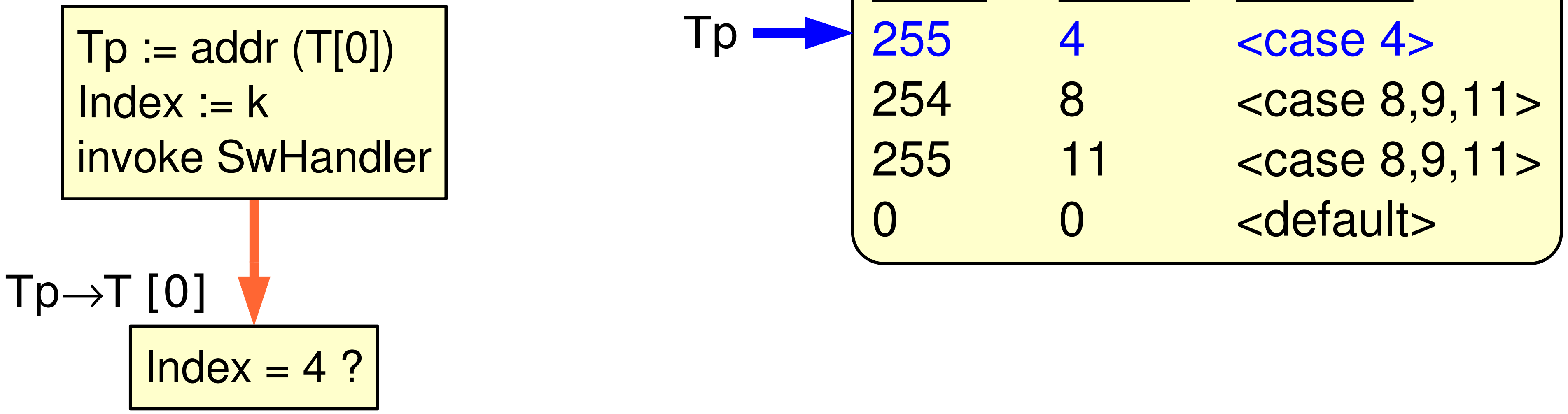
1. Add first step in *SwHandler*, with data state



Tp →

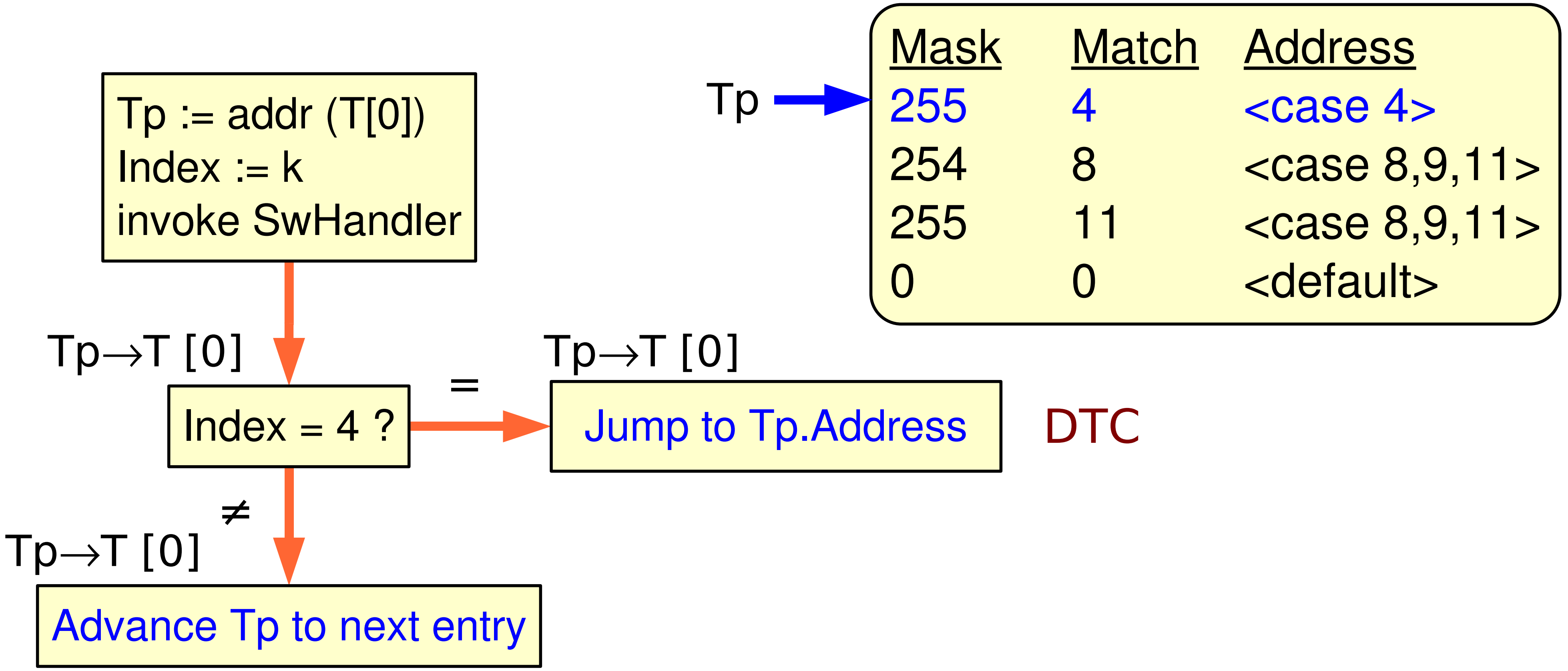
Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

2. Partially evaluate wrt data state



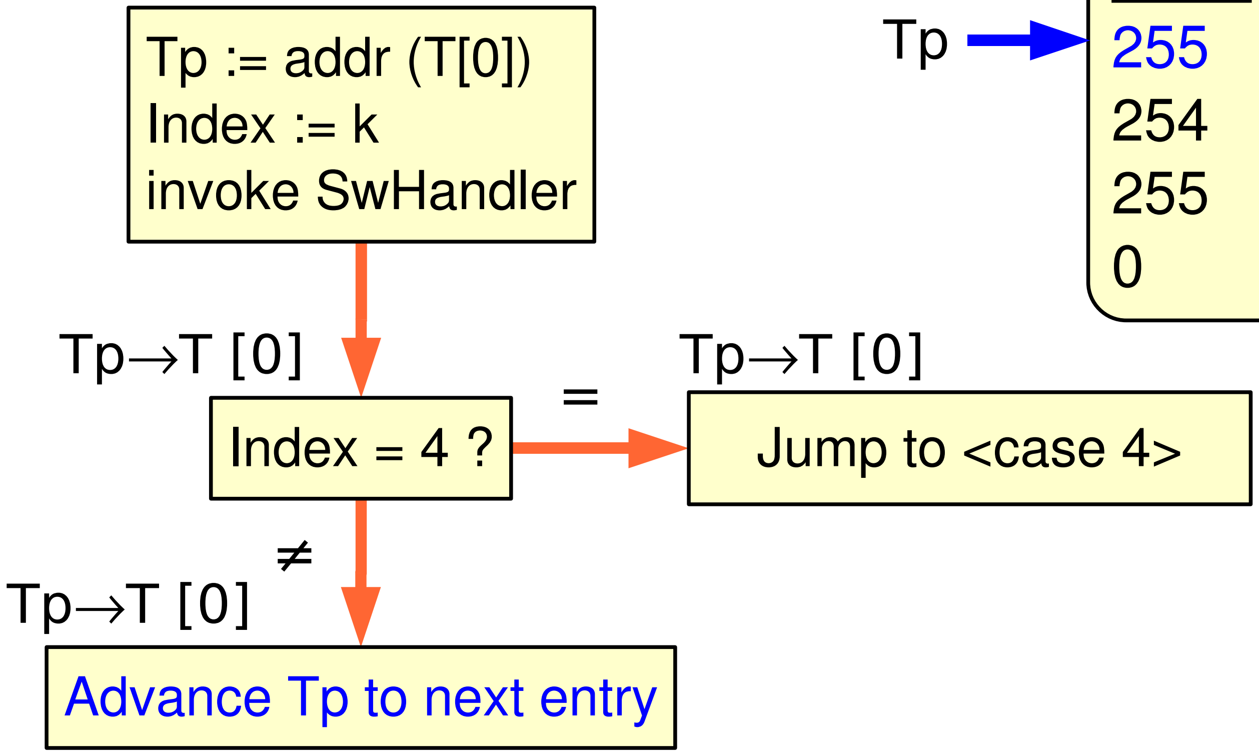
Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

3. Generate successors (PC, data state) & add



Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

4a. Partially evaluate wrt data state



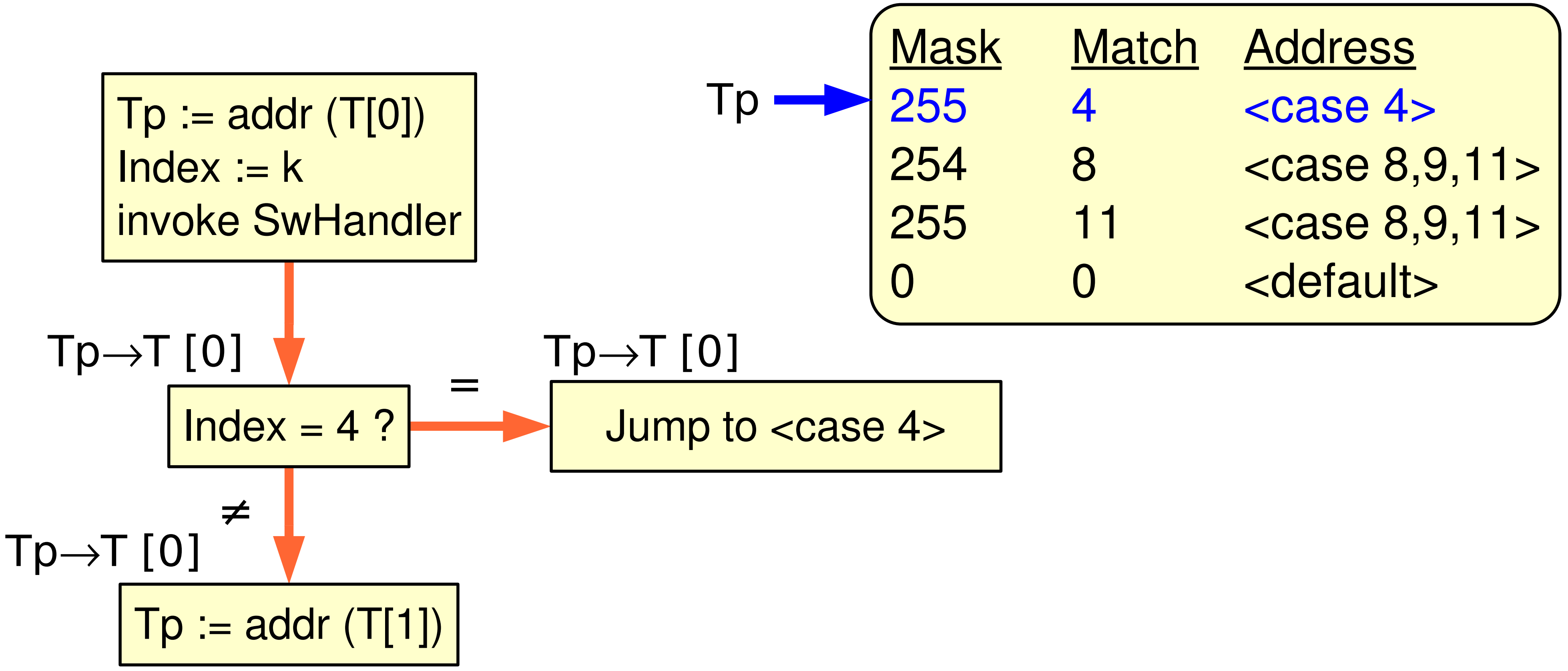
Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

DTC resolved

PE ends on this path

CFG building continues (shown later)

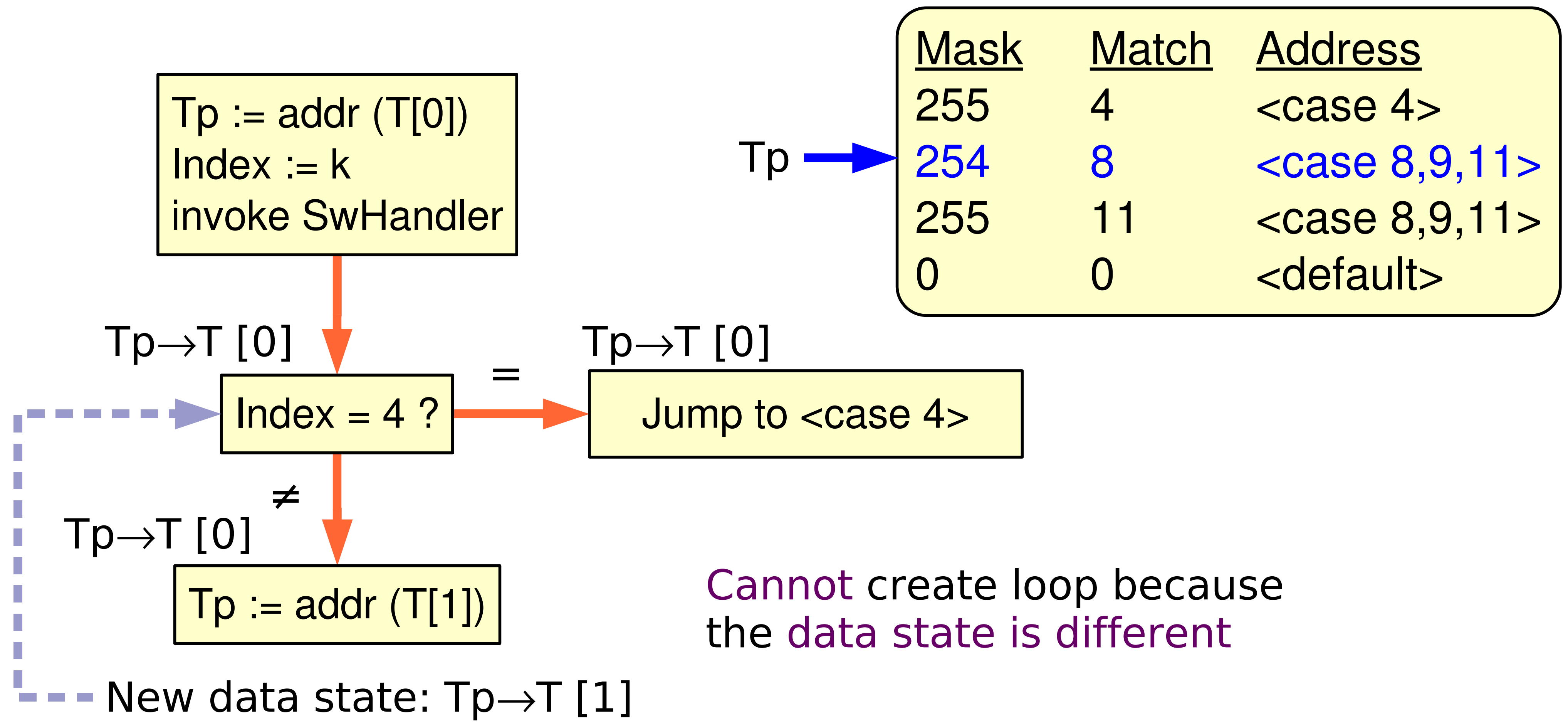
4b. Partially evaluate wrt data state



Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

New data state: Tp→T [1]

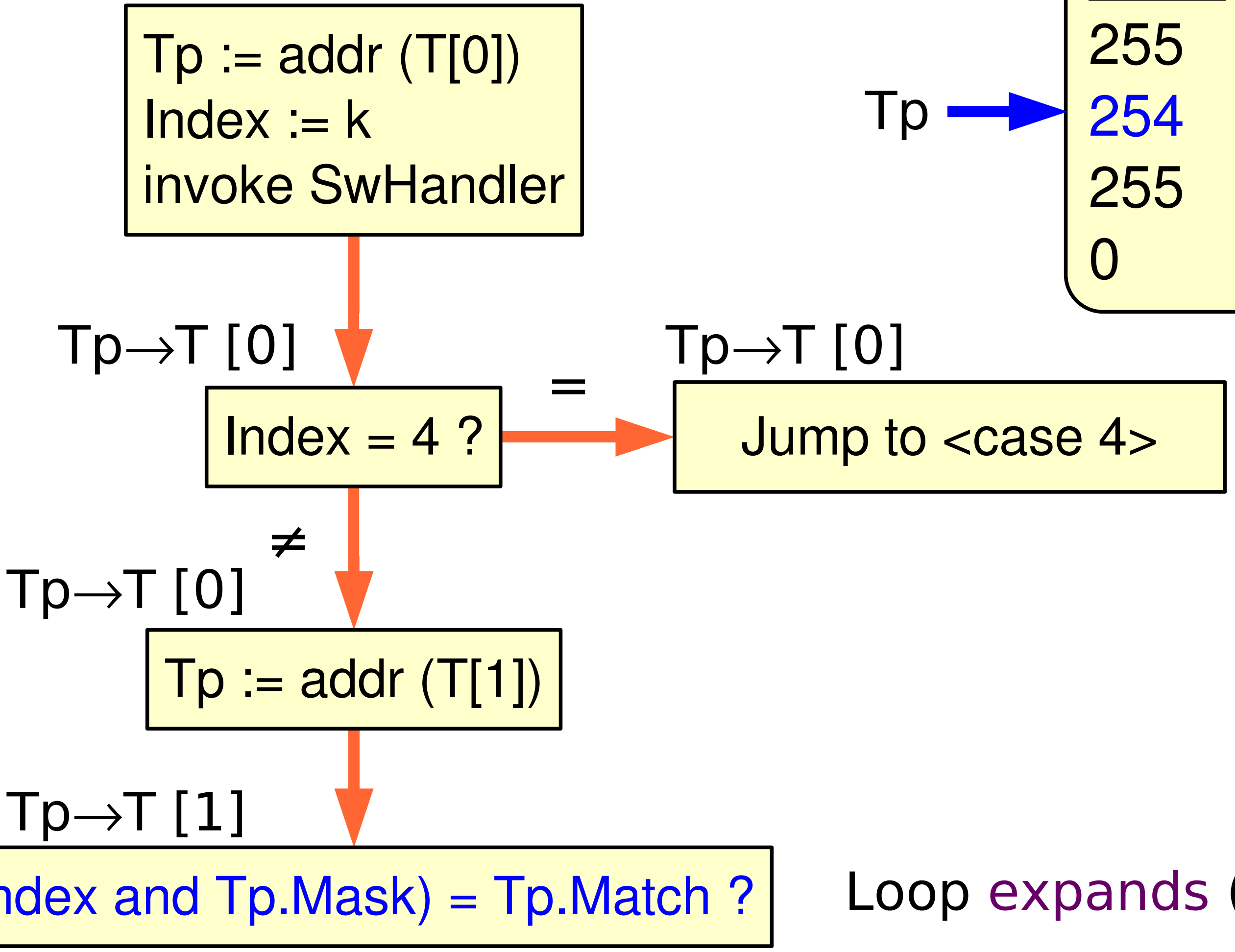
Generate successors: can we loop?



Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

Cannot create loop because the data state is different

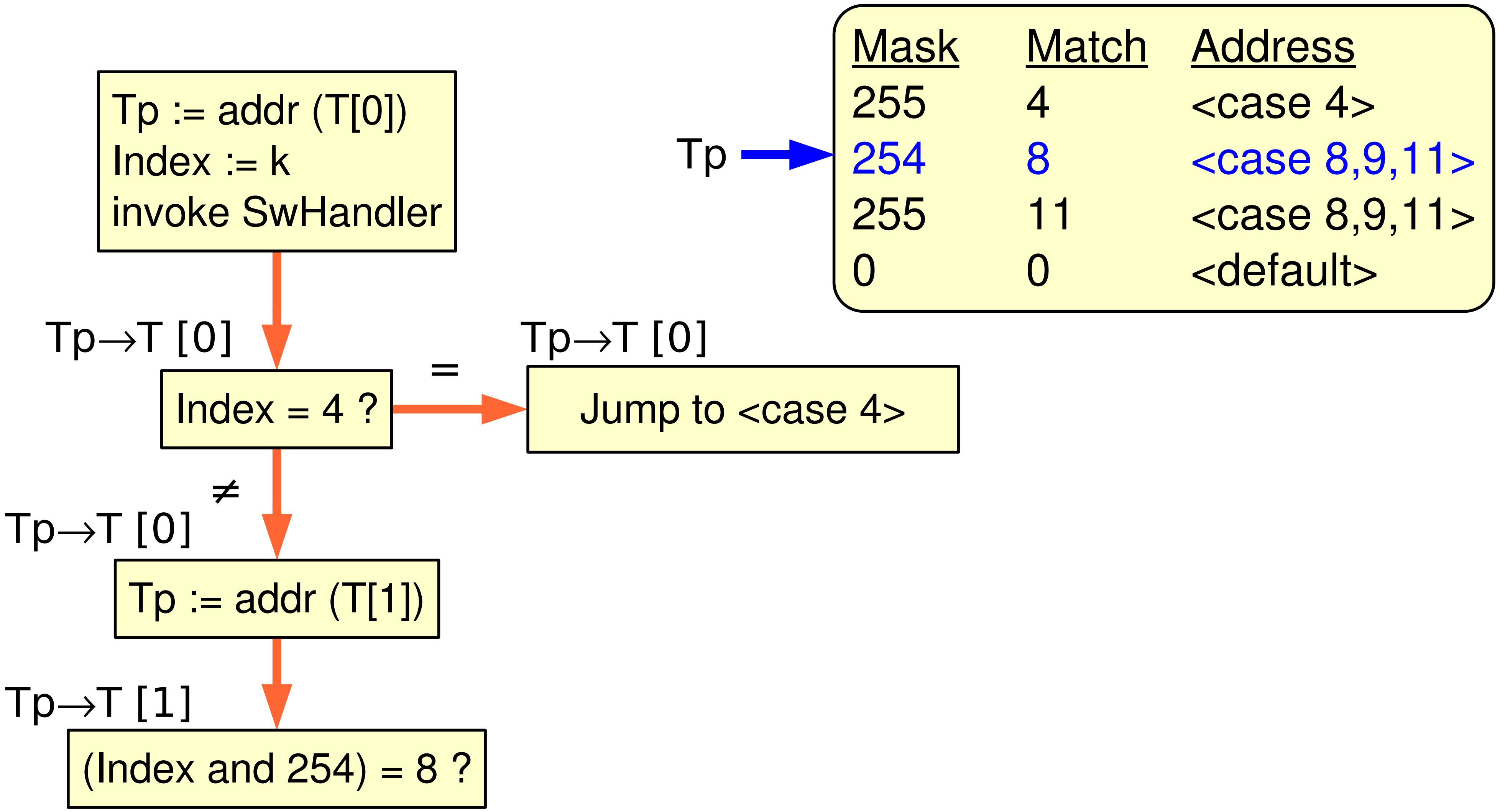
5. Generate successors (PC, data state) & add



Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

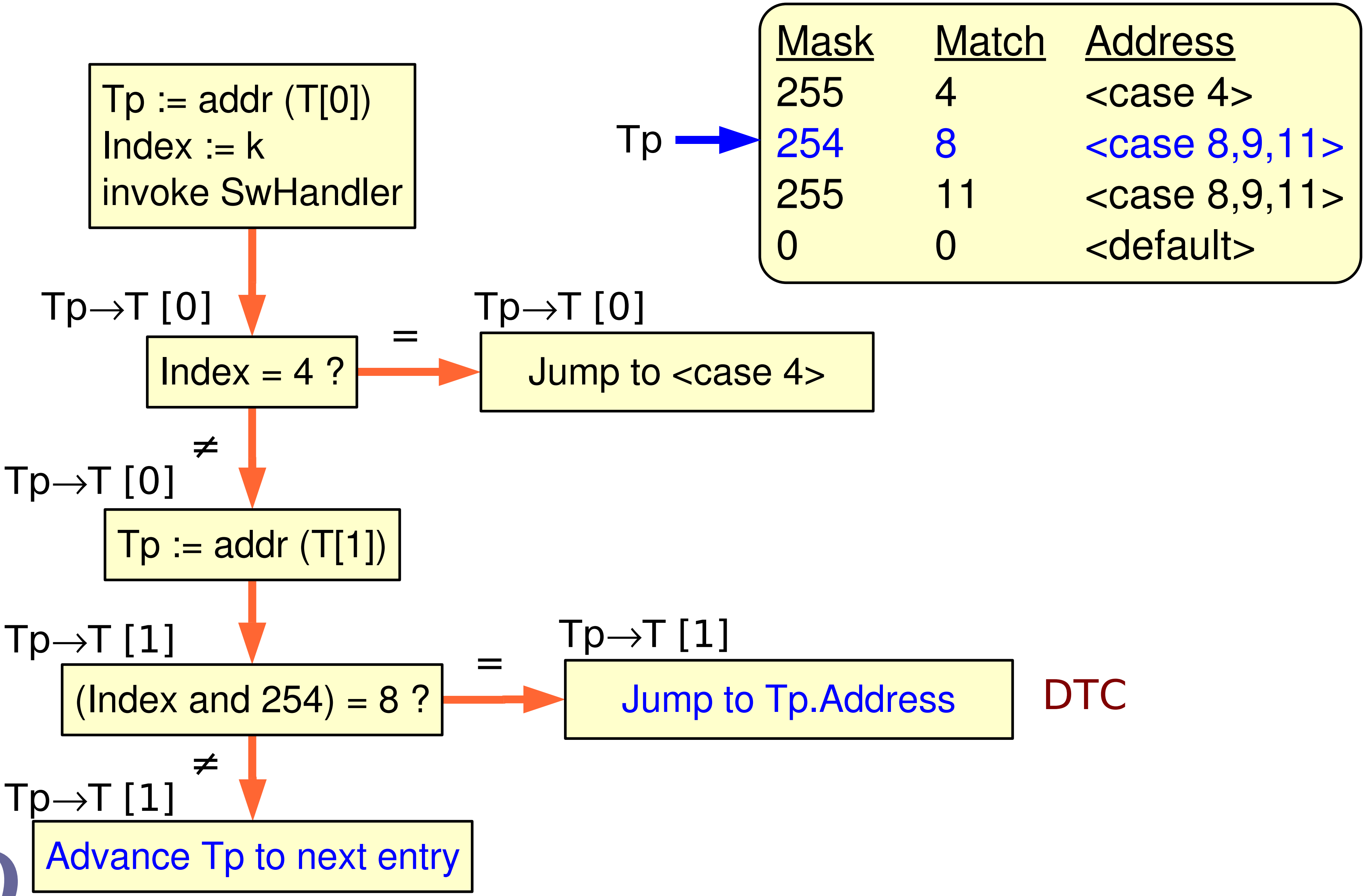
Loop expands (unrolls) itself

6. Partially evaluate wrt data state



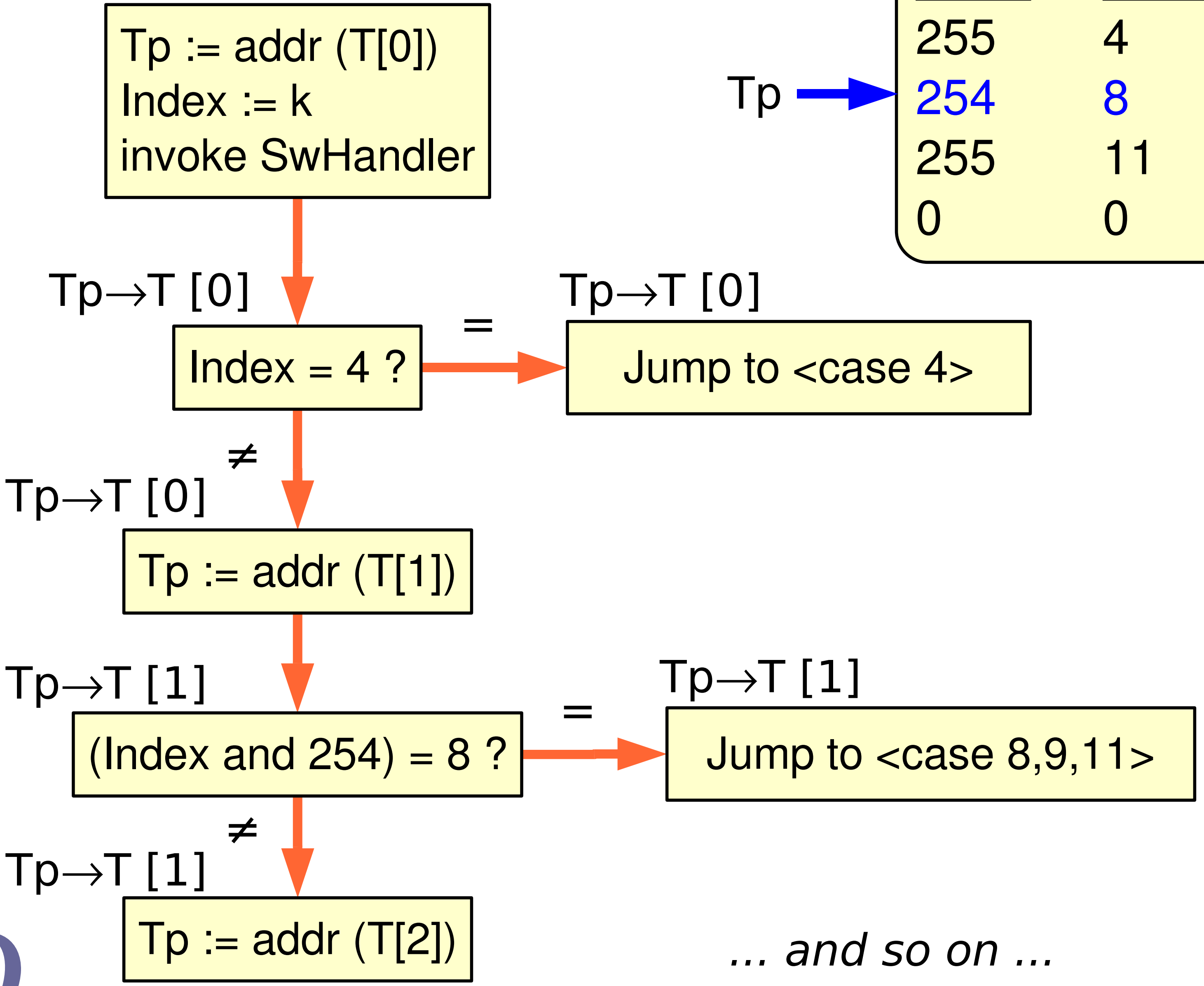
Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

7. Generate successors (PC, data state) & add



Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

8. Partially evaluate wrt data state, etc.

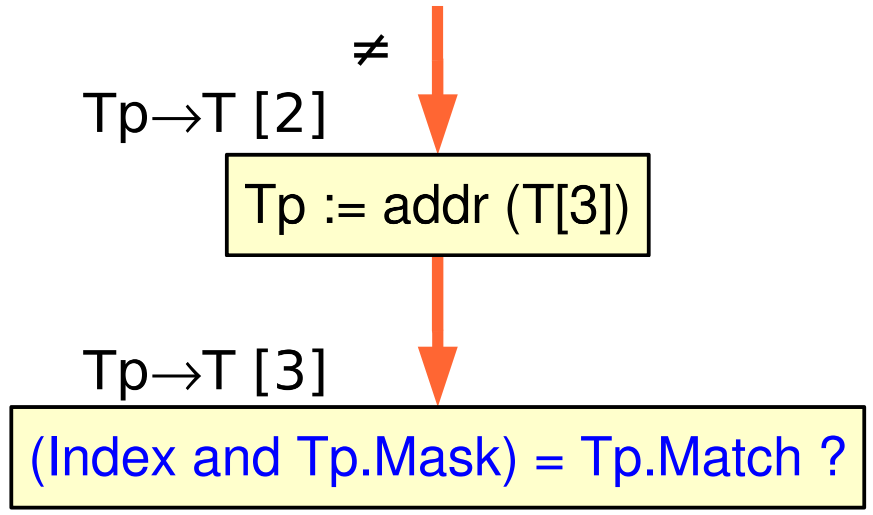


Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

DTC resolved

... and so on ...

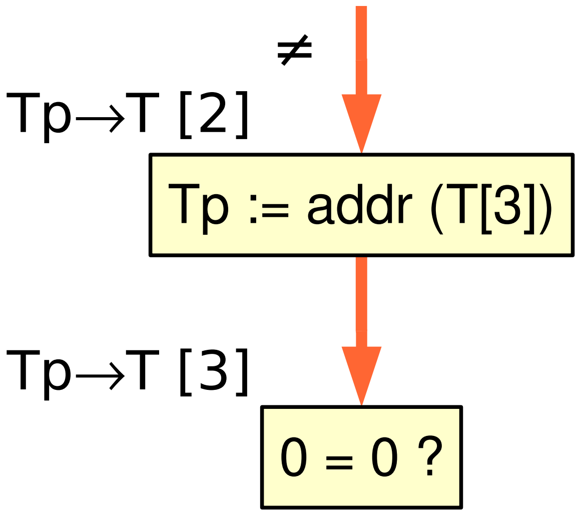
n. Add successors for last table entry



Tp →

Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

n+1. Partially evaluate wrt data state

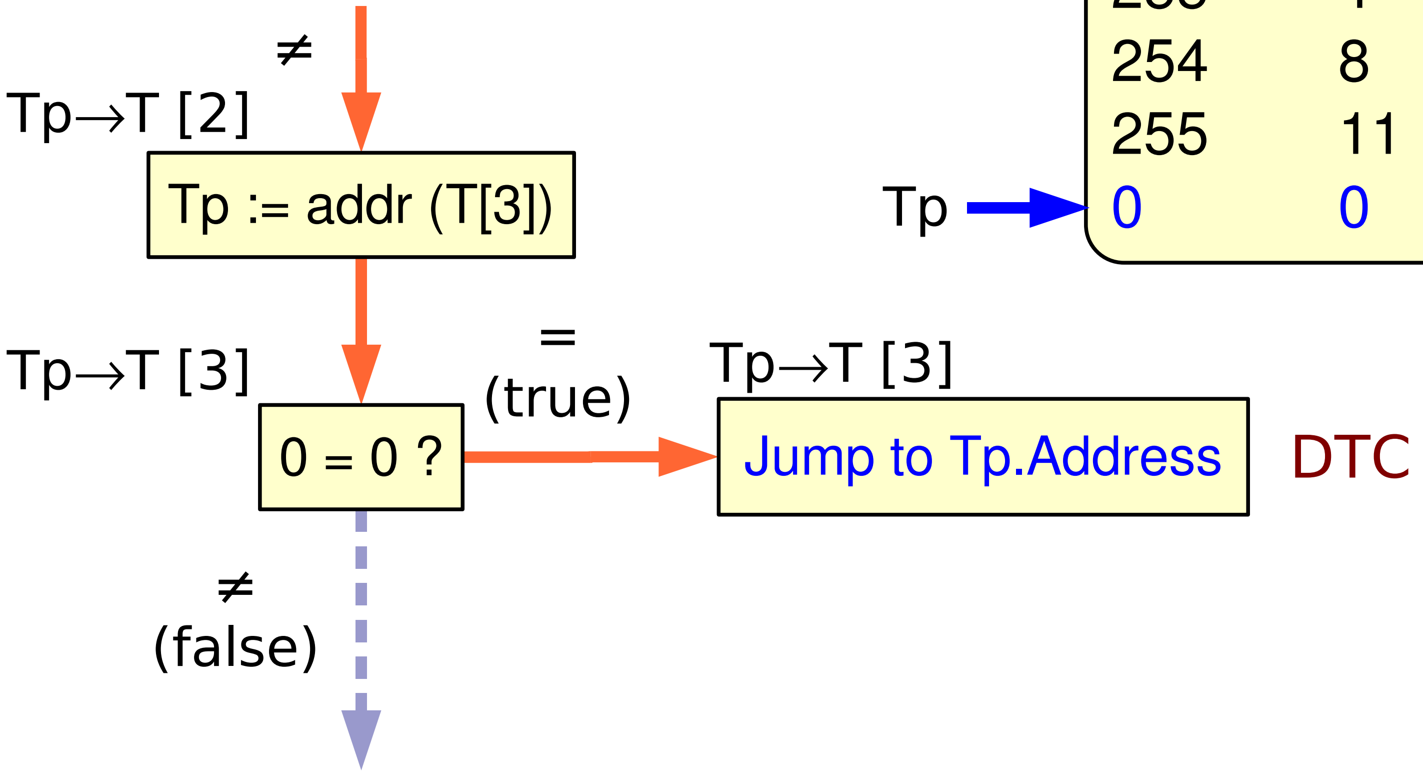


Tp →

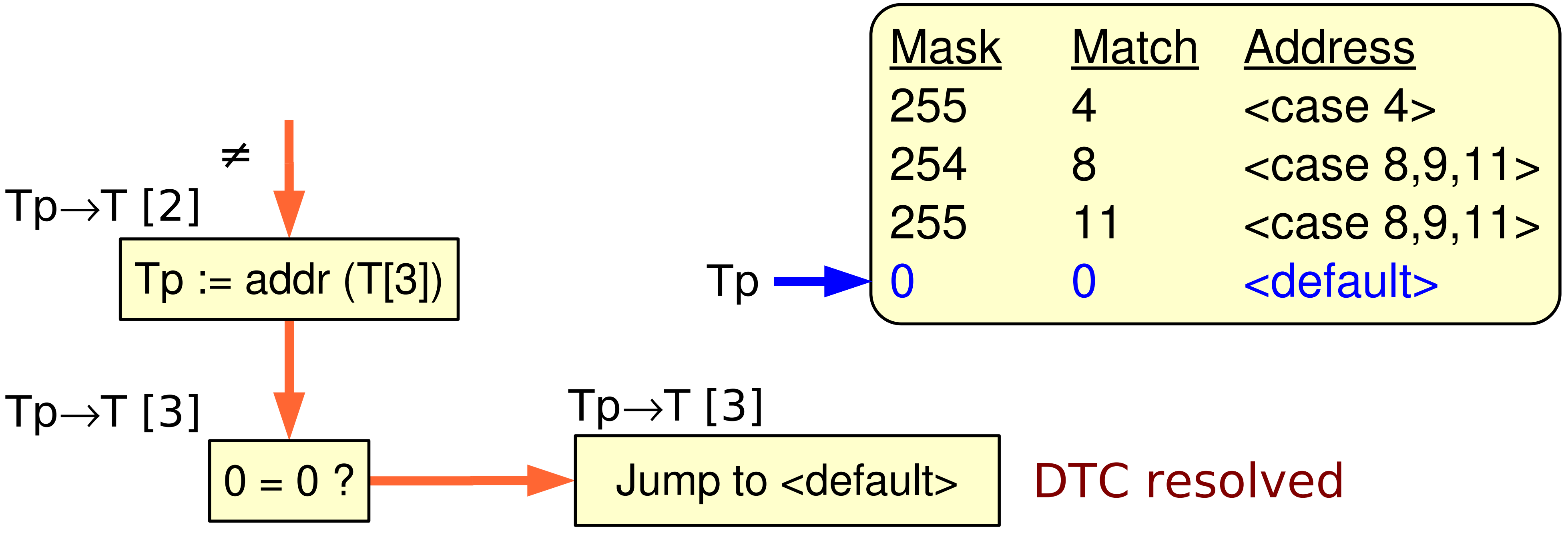
Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

n+2. Generate & add *feasible* successors

Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>



n+3. Partially evaluate wrt data state

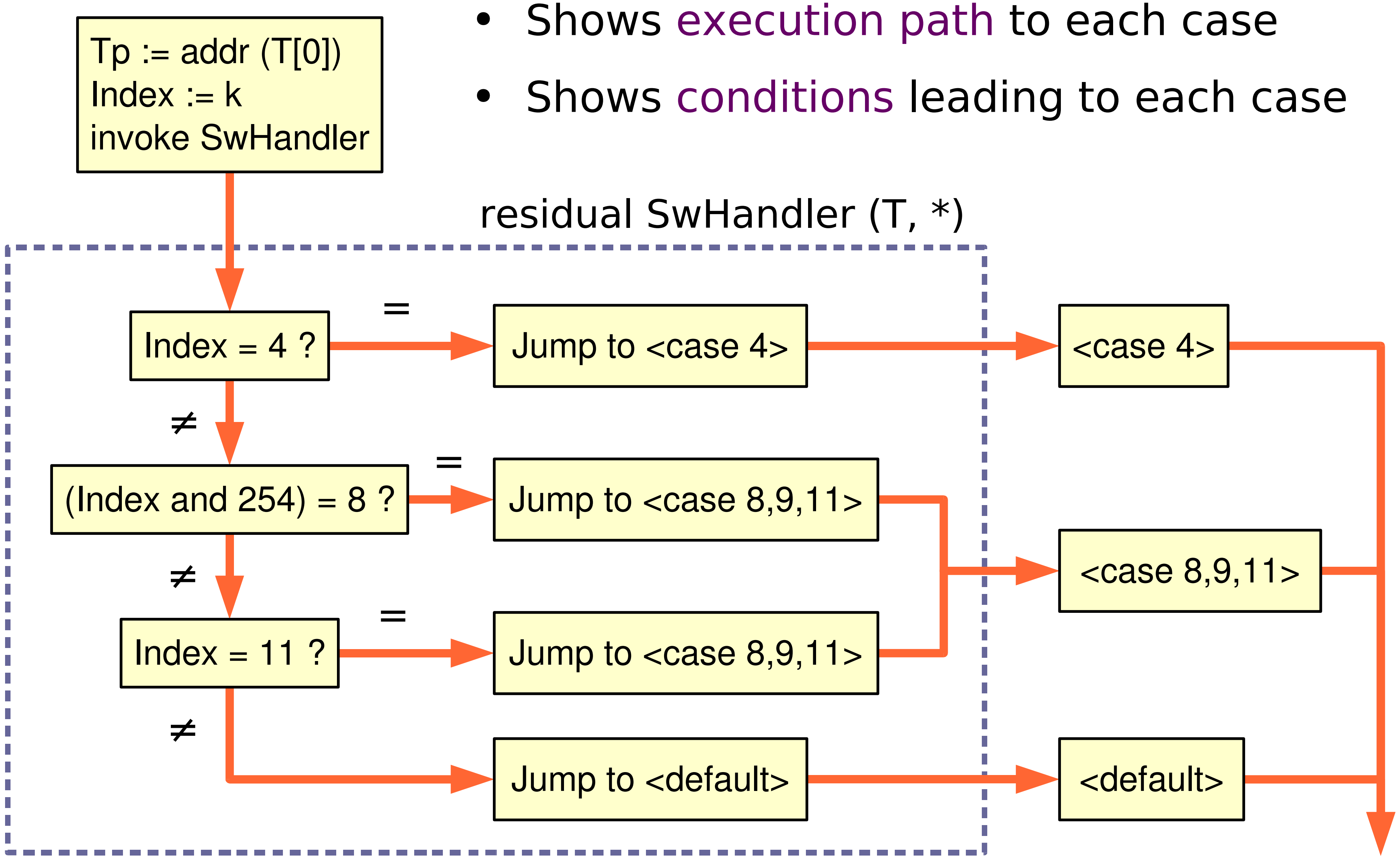


Mask	Match	Address
255	4	<case 4>
254	8	<case 8,9,11>
255	11	<case 8,9,11>
0	0	<default>

- All paths end with resolved DTC
- Expansion of switch handler completed
  - for this switch-case statement

Completed flow-graph

- Shows execution path to each case
- Shows conditions leading to each case



Summary

- Traditional flow-graph construction:
  - enumerate only PC values
- Flow-graph construction with PE:
  - choose relevant data state components (... but how ?)
  - enumerate product domain (PC, data state)
- PE applied here to switch handlers
  - easy to choose the relevant data:
    - the switch table and anything derived from it
- Other PE applications in WCET analysis?