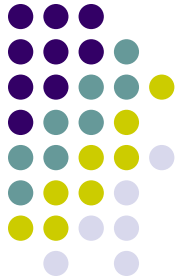


Managing Software Development

Day 10 Debugging and Testing

A pragmatic method for quality control

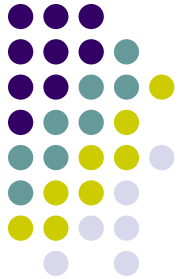
Charoen Vongchumyen

Email : kvocharo@kmitl.ac.th

04/2021

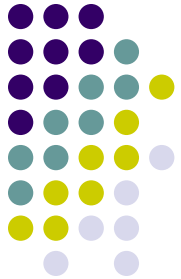
Thank to Tsuneo Yamaura

Differences between testing and debugging



- Similar but completely different!
  - Testing (by QA)
    - → The process to prove that the software is NOT correct.
    - → If a very serious bug is found, Stop further testing.
  - Debugging (by programmer)
    - → The process that prove that the software is CORRECT.
    - → You have to check all the functions.

Input domain space and test space (testability)



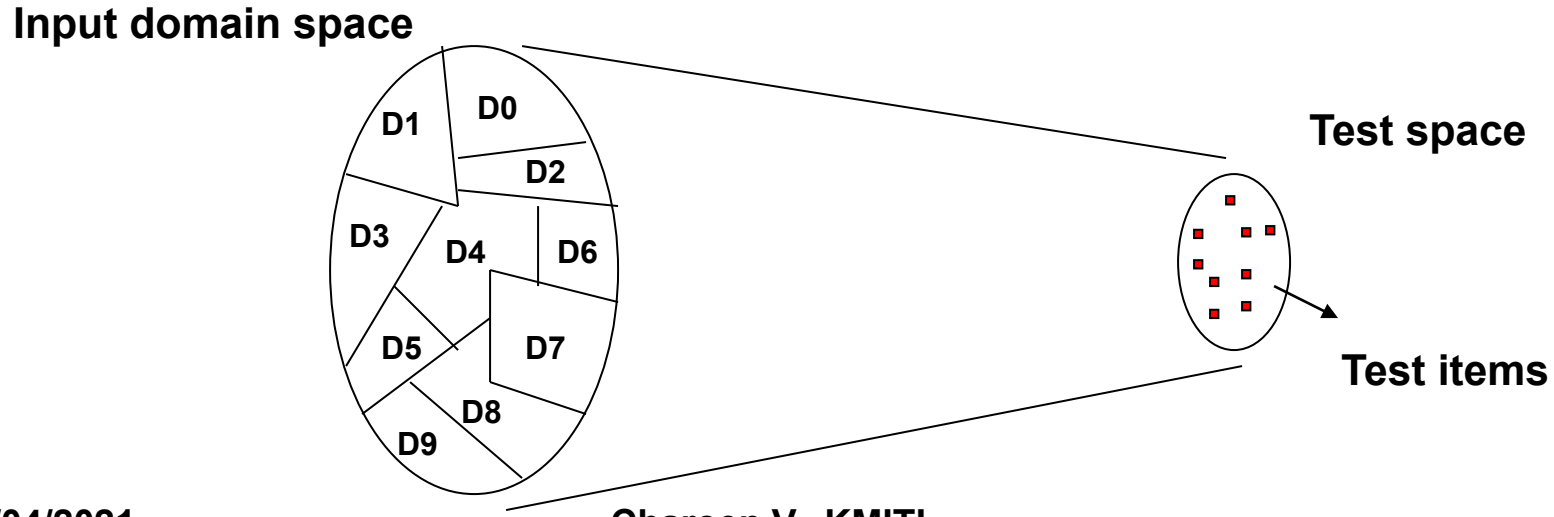
- A simple program has an enormous number of paths.
  - It takes 1,000,000 years to run all the paths even if you take only 1 second to execute a test case.
  - You cannot test all the possible combinations in the reasonable amount of resource.

$$5 + 5^2 + 5^3 + \ldots + 5^{18} = 4.77 \times 10^{12}$$

"All paths execution" cannot detect "missing logic"

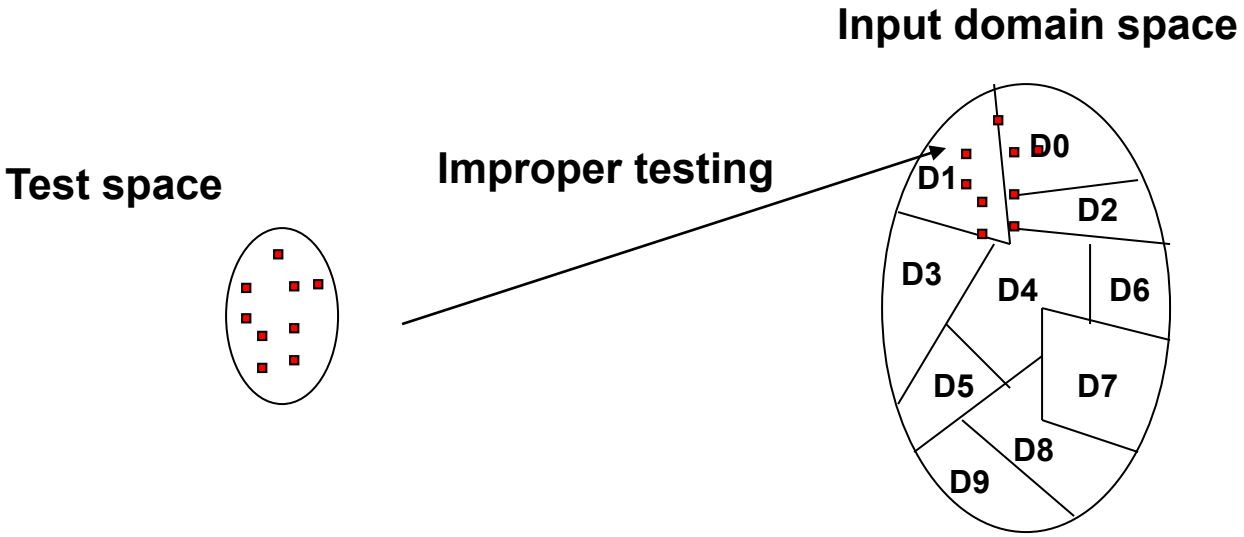
Input domain space and test space (testability)

- The test space is too small compared to the input the input domain space.
  - You have to design test cases that will test the huge input domain space effectively. This ability required the expert test engineers. (quite high ability)



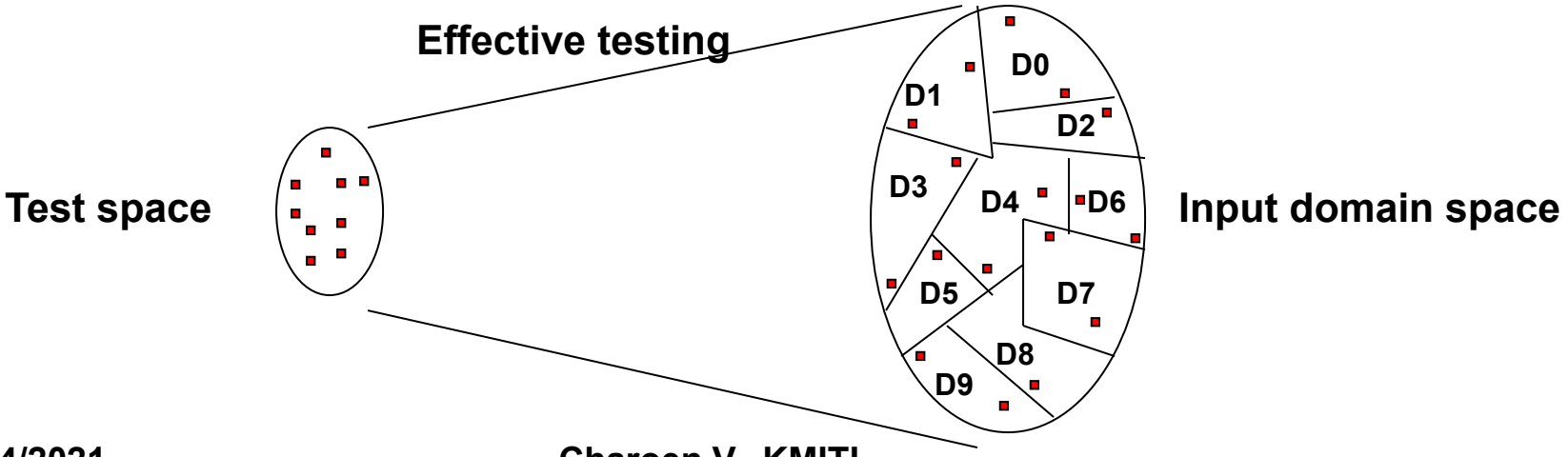
Input domain space and test space (testability)

- If perfect testing is impossible, it is quite important to effectively assign the test space to the input domain space.

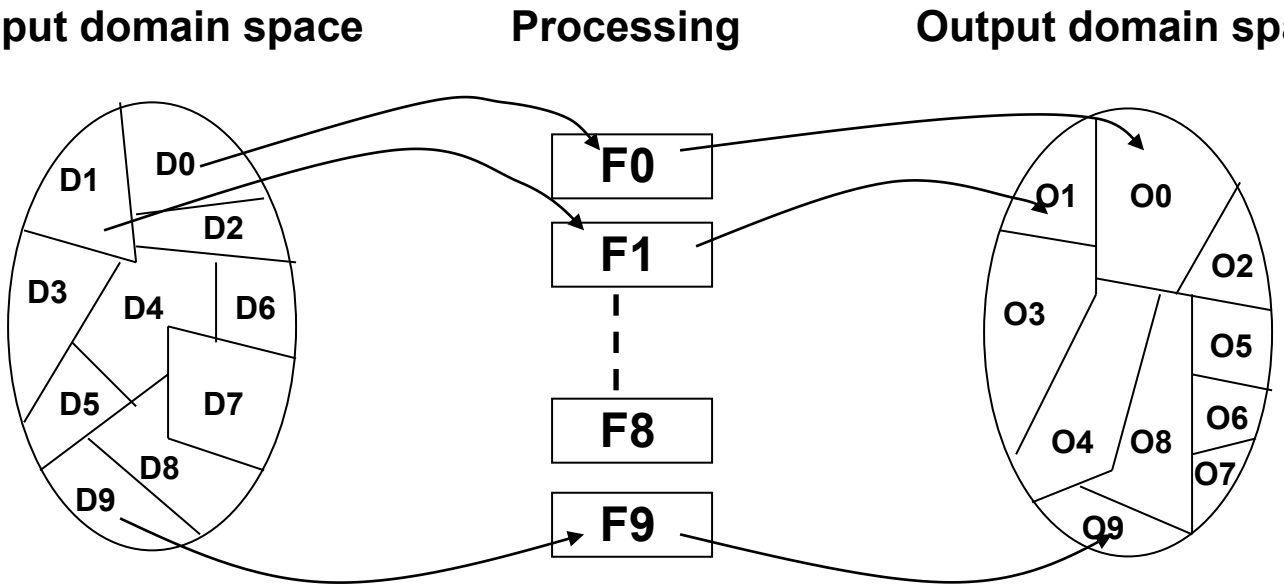


Input domain space and test space (testability)

- You have to cover all the input domain spaces.
  - You have to know the functions and structures of the targeting program.
- You have to extensively test the portion where the bugs likely exists.
  - You need to have quality assurance expertise and experience.

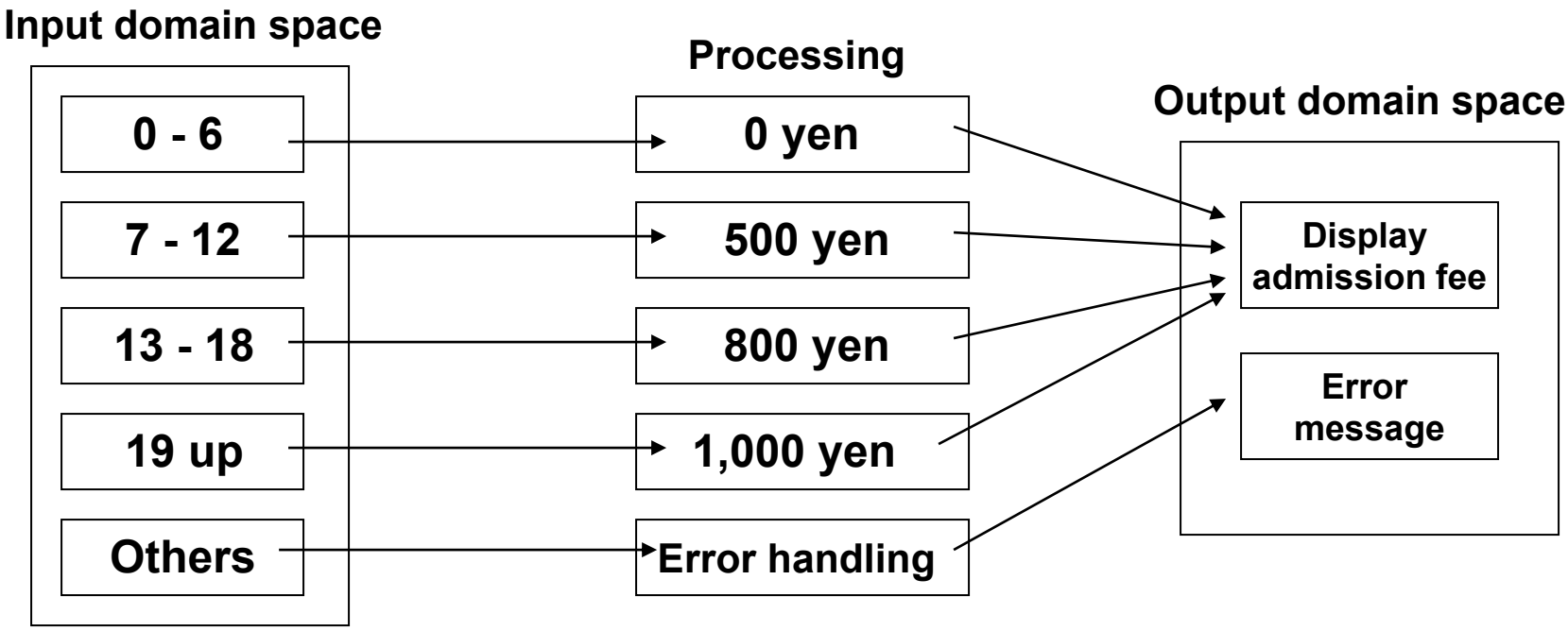


Conceptual model of program



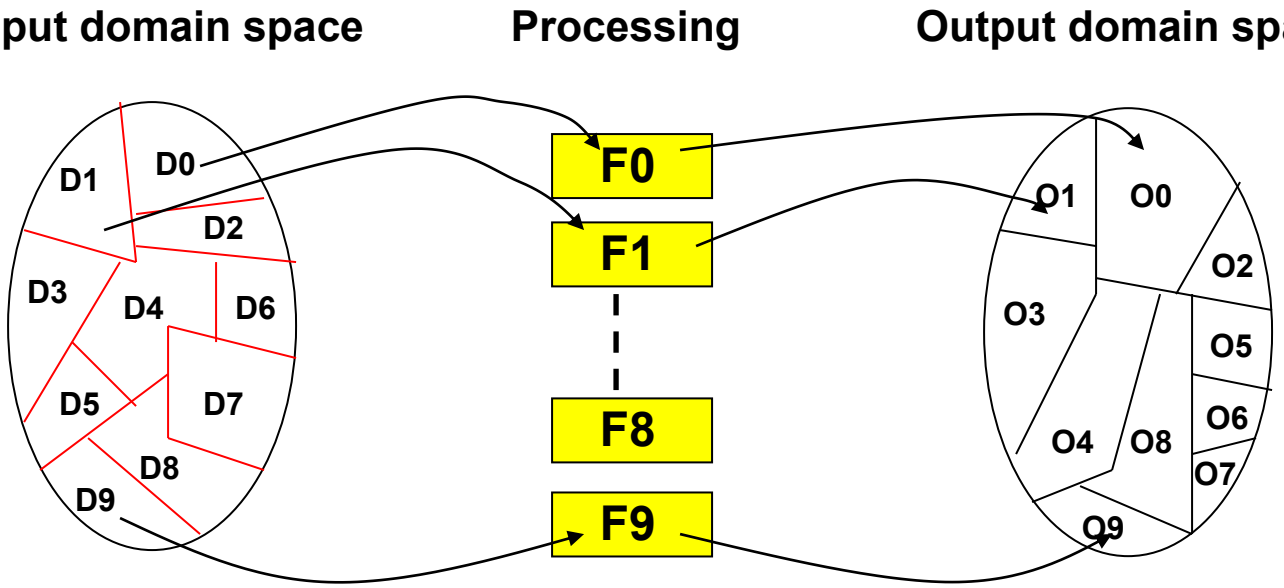
Conceptual model of program

Admission fee : Below 6 year old : free, 7 – 12 : 500 yen,
13 – 18 : 800 yen, 19 years or older : 1,00 yen



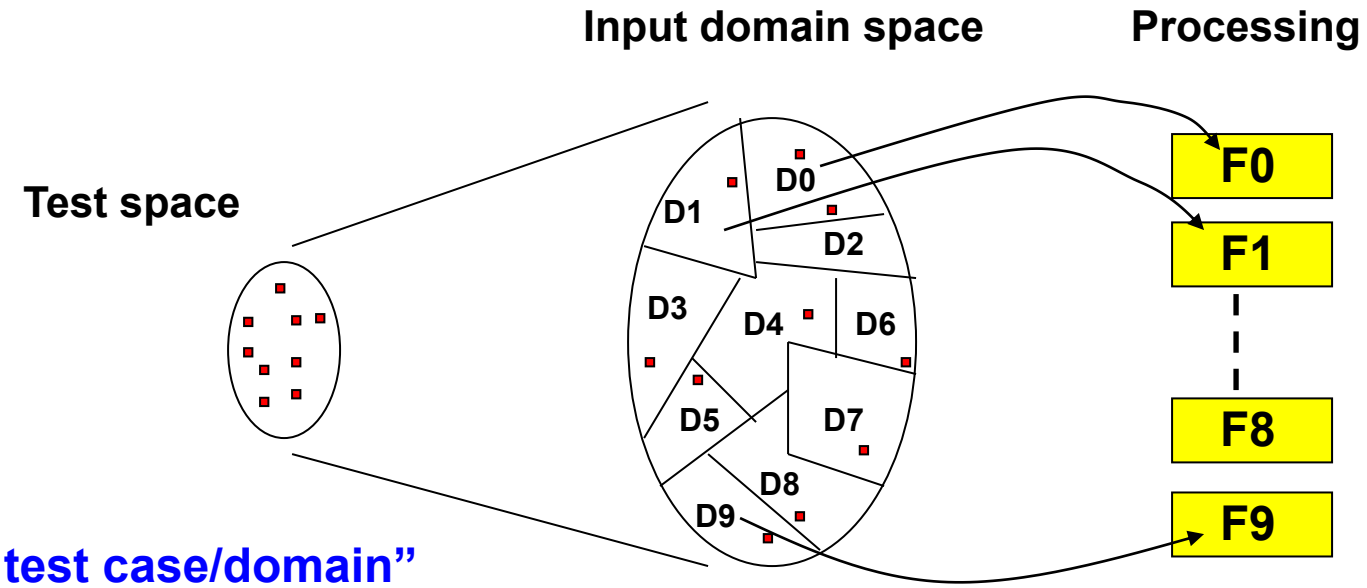
Conceptual model of program

- Only 2 types of bugs :
  - Wrong “processing”
  - Wrong “Input domain” (Boundary)



Detecting wrong "processing"

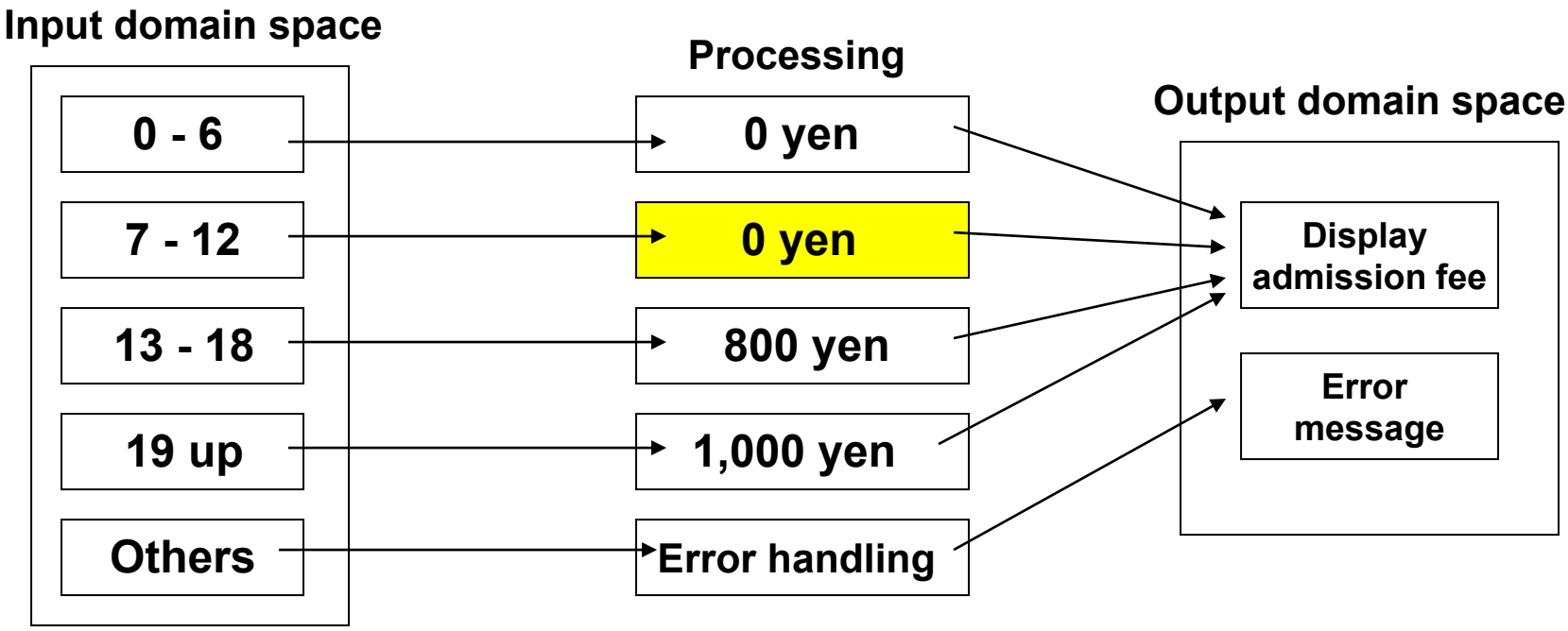
- Design the test items for each input domain space, and validate the spaces.
  - There is possibility of coincidental correctness.



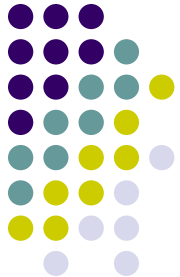
"Just 1 test case/domain"

Detecting wrong “processing”

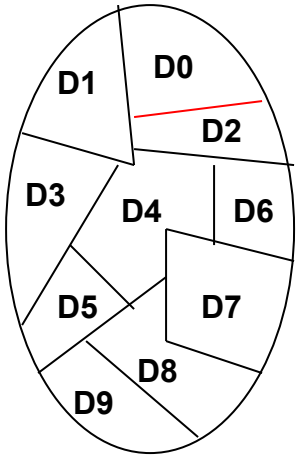
Admission fee : Below 6 year old : free, 7 – 12 : 500 yen,
13 – 18 : 800 yen, 19 years or older : 1,00 yen

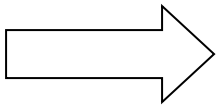


Detecting wrong "input domain" boundary

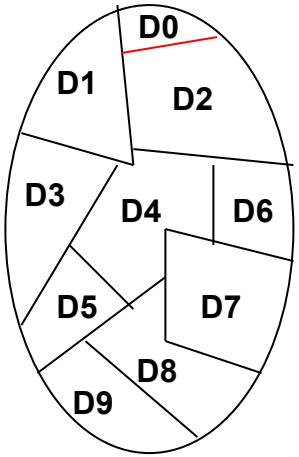


Wrong input domain boundary



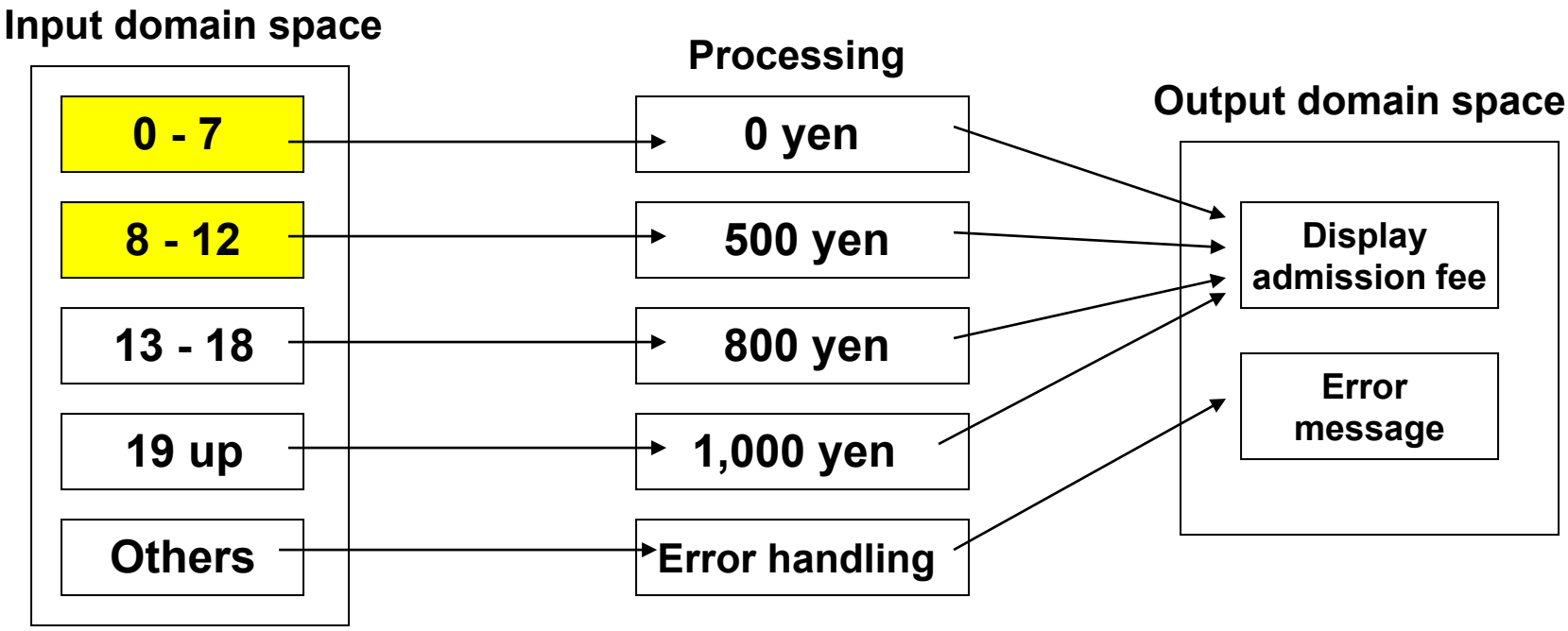


Correct input domain boundary

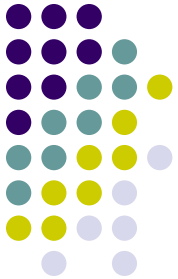


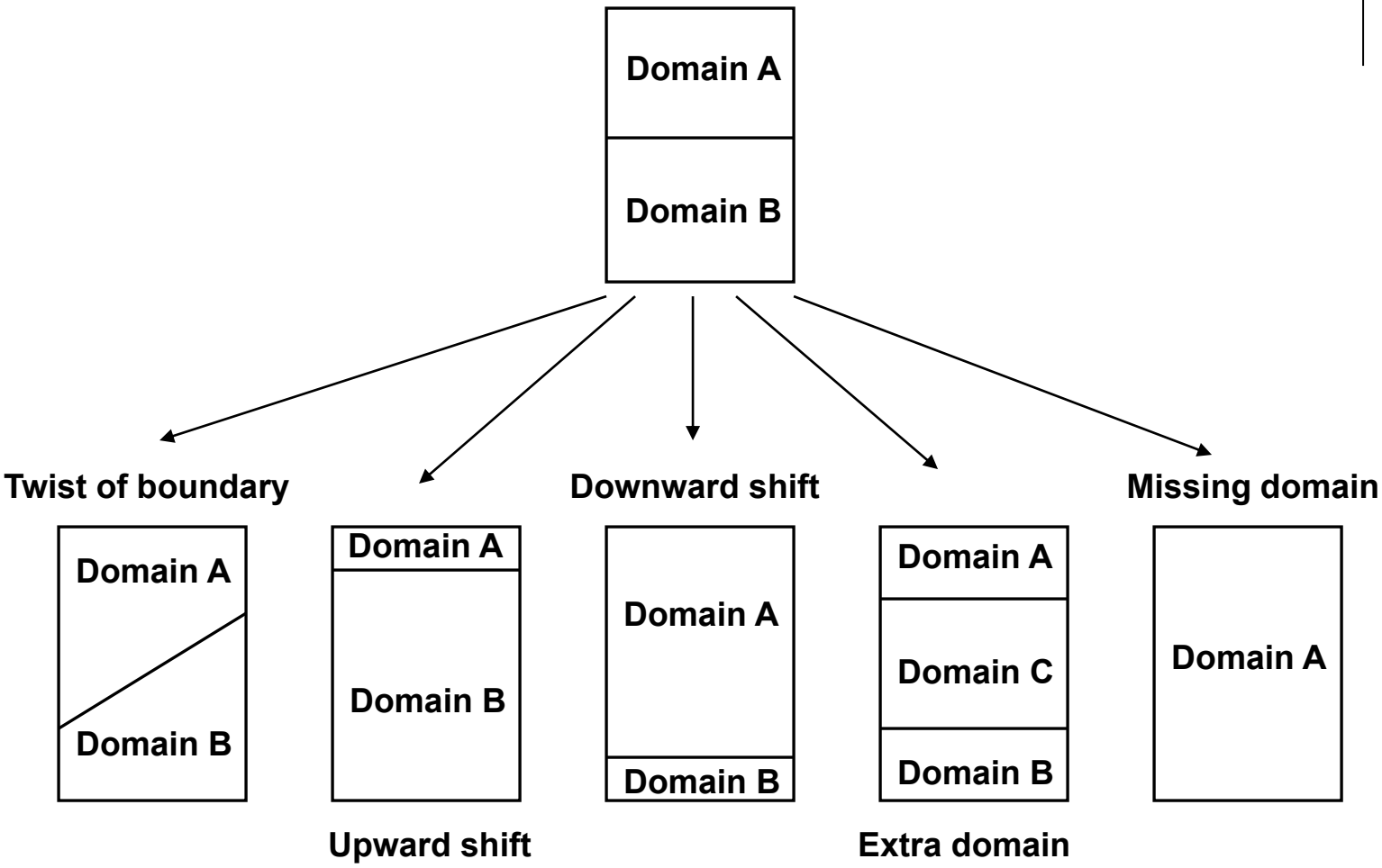
Detecting wrong "input domain" boundary

Admission fee : Below 6 year old : free, 7 – 12 : 500 yen,
13 – 18 : 800 yen, 19 years or older : 1,00 yen



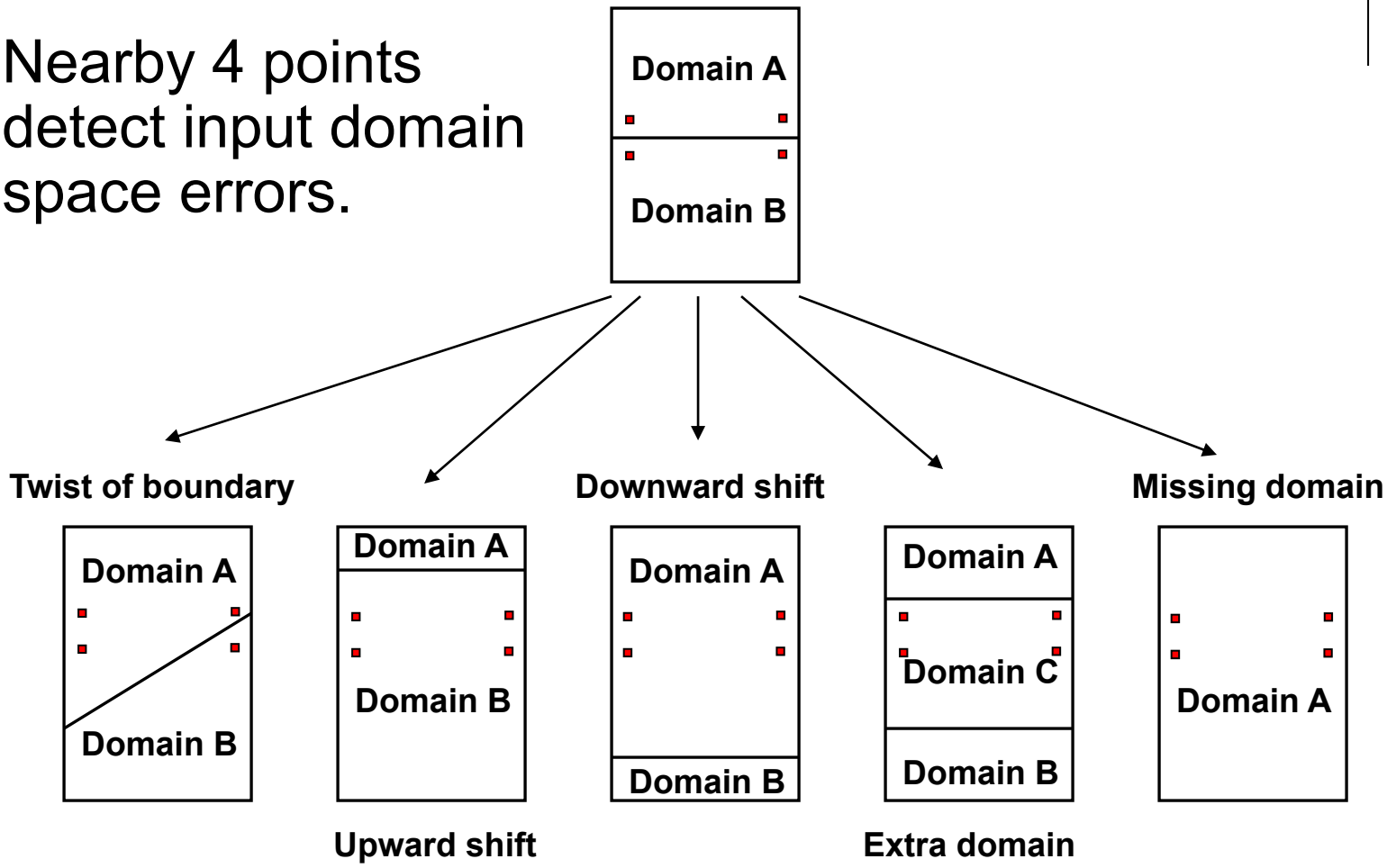
Detecting the input domain space boundary errors



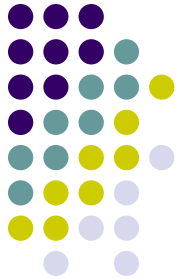


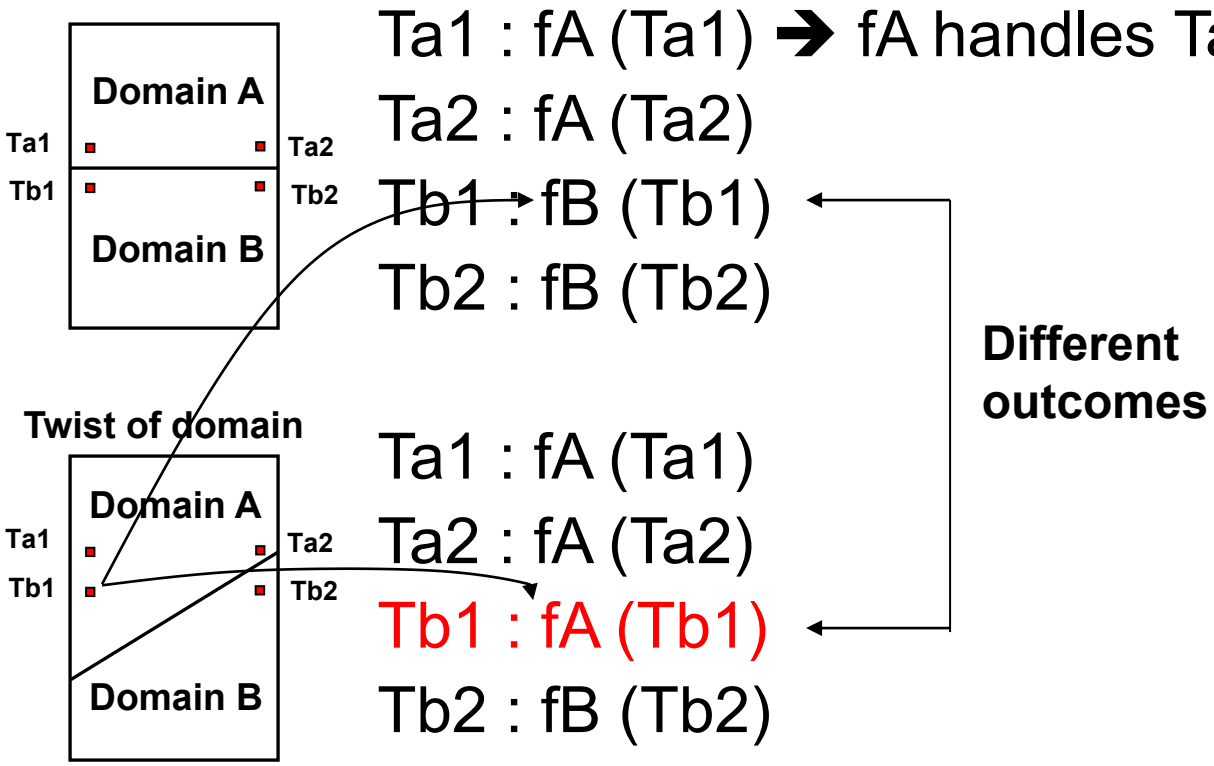
Types of the input domain space boundary errors

- Nearby 4 points detect input domain space errors.

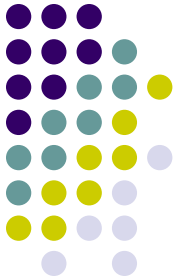


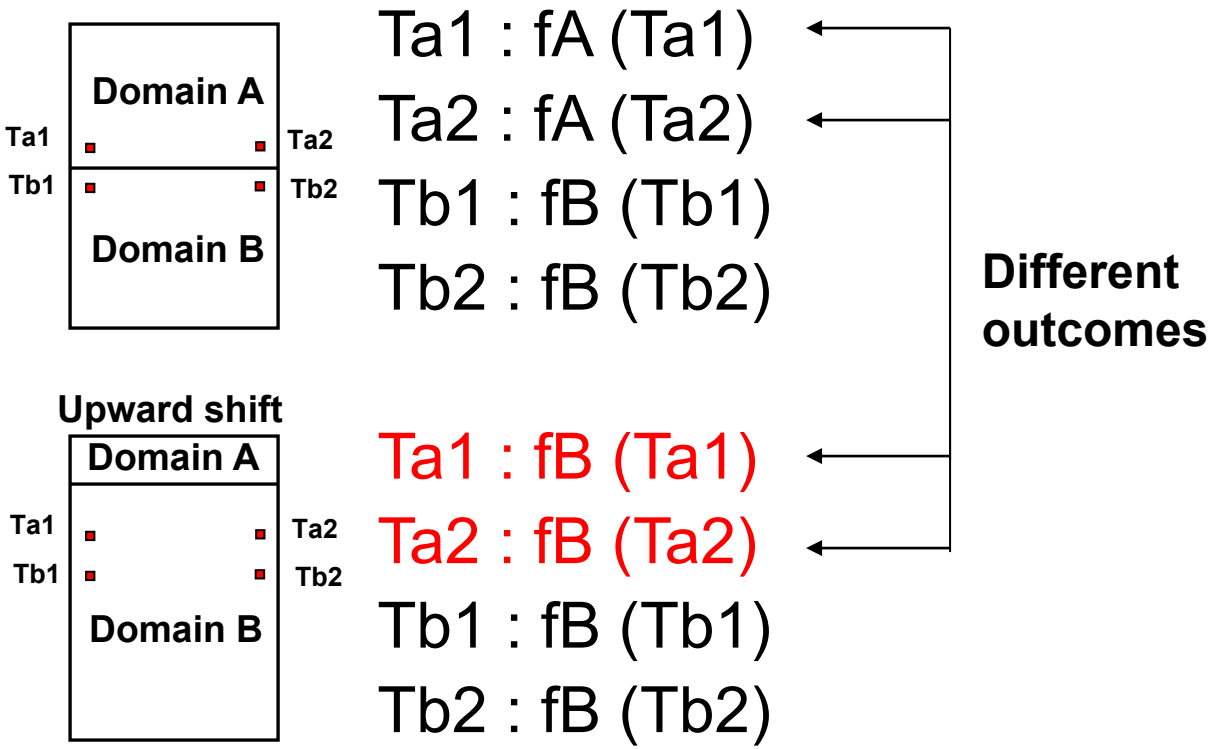
Detecting the input domain space boundary errors (Part 1)



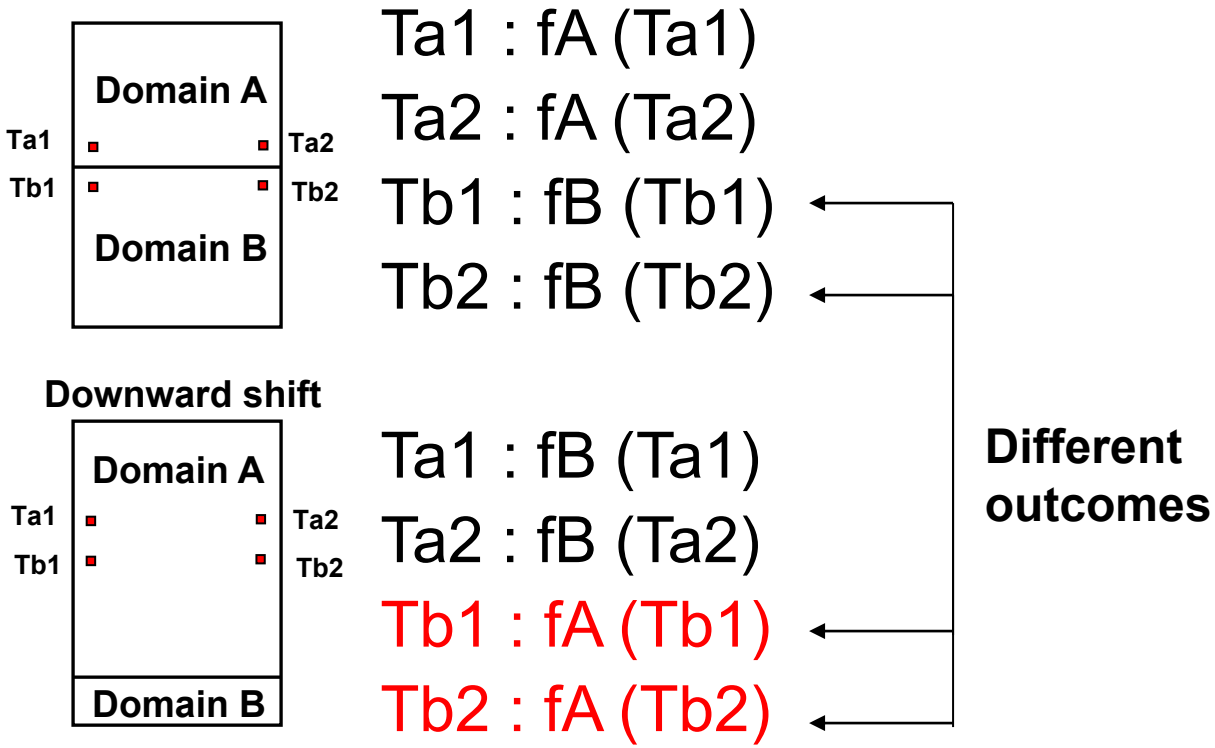


Detecting the input domain space boundary errors (Part 2)

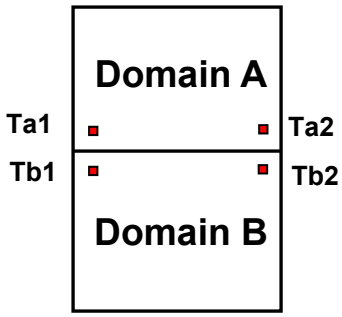




Detecting the input domain space boundary errors (Part 3)



Detecting the input domain space boundary errors (Part 3)



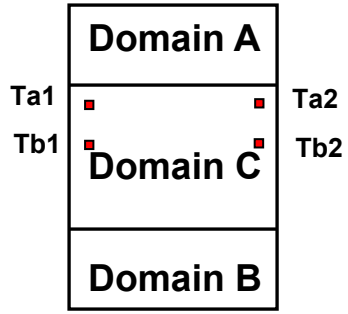
Ta1 : fA (Ta1)

Ta2 : fA (Ta2)

Tb1 : fB (Tb1)

Tb2 : fB (Tb2)

Extra domain



Ta1 : fC (Ta1)

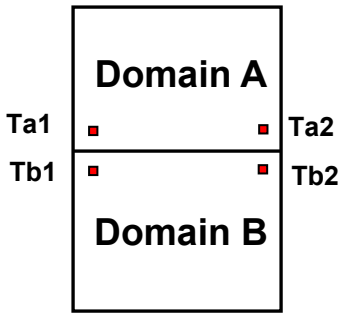
Ta2 : fC (Ta2)

Tb1 : fC (Tb1)

Tb2 : fC (Tb2)

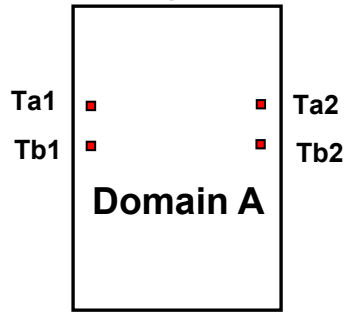
Different outcomes

Detecting the input domain space boundary errors (Part 3)



Ta1 : fA (Ta1)
Ta2 : fA (Ta2)
Tb1 : fB (Tb1)
Tb2 : fB (Tb2)

Missing domain

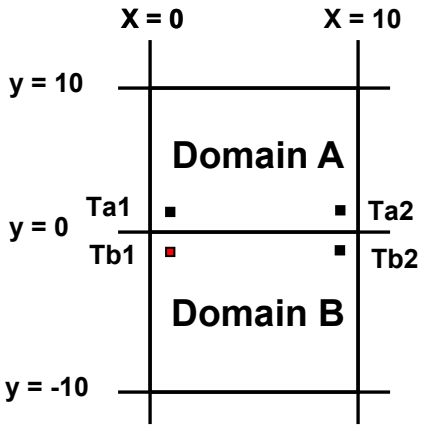


Ta1 : fB (Ta1)
Ta2 : fB (Ta2)
Tb1 : fA (Tb1)
Tb2 : fA (Tb2)

Different outcomes

*** Same as Downward shift domain**

Coincidental correctness



Ta1 = (1,1) : fA(x,y) = x + 2y ➔ 3

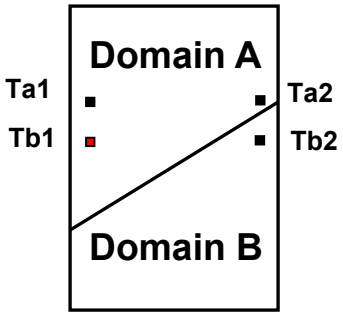
Ta2 = (9,1) ➔ 11

Tb1 = (1,-1) : fB (x,y) = xy ➔ -1

Tb2 = (9,1) ➔ -9

The processing is different, but the outcome coincidentally match.

Twist of domain



Ta1: (1,1) : fA(x,y) = x+ 2y ➔ 3

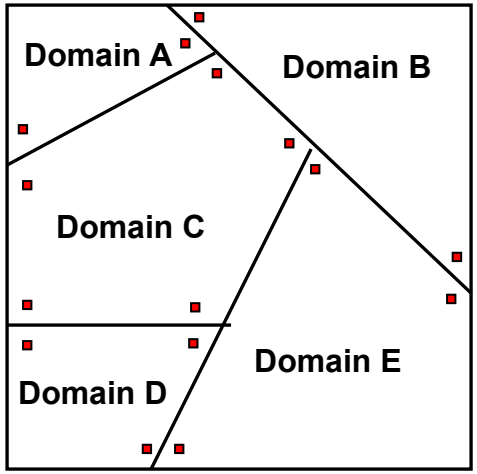
Ta2: (9,1) : fA(x,y) = x+ 2y ➔ 11

Tb1: (1,-1): fA(x,y) = x+ 2y ➔ -1

Tb2: (9,-1): fA(x,y) = xy ➔ -9

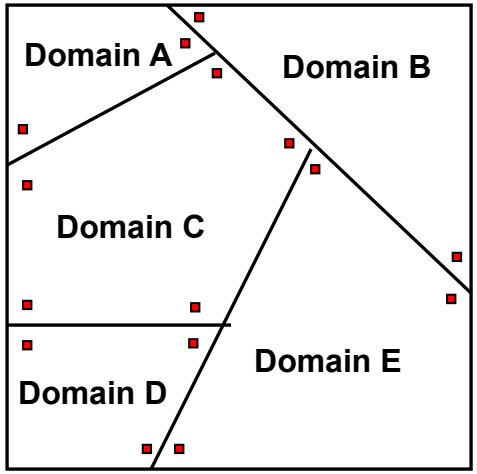
Guidelines of test case designing

- Generate at least 1 test item to each input domain space.
  - This is for validation of processing.
- Generate nearby test points for each input domain space.
  - This is for validation of domain boundary.



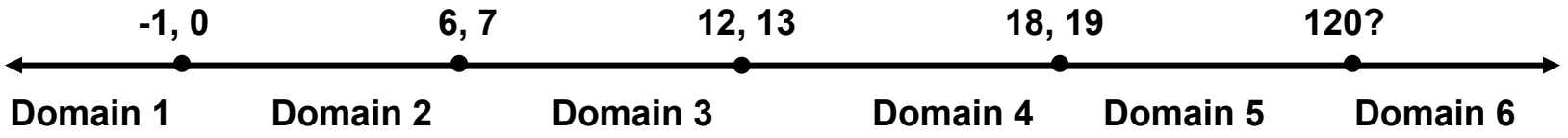
Black box testing and Guidelines of test case designing

- Generate at least 1 test item to each input domain space.
  - Equivalent partitioning in black box testing.
- Generate nearby test points for each input domain space.
  - Boundary analysis in black box testing.



Test case design in equivalent partitioning

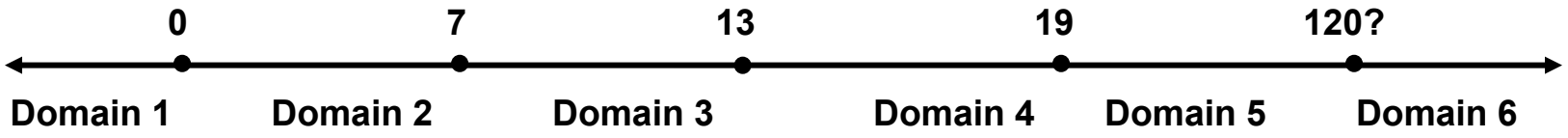
Admission fee : Below 6 year old : free, 7 – 12 : 500 yen,
13 – 18 : 800 yen, 19 years or older : 1,00 yen



- Pick up only on test case per each domain.
- For example,
  - -2 for domain 1, 5 for domain2, 8 for domain 3, 15 for domain 4, 100 for domain 5, 150 for domain 6.
  - Do not make too many test cases.

Test case design in boundary analysis

Admission fee : Below 6 years old : free, 7 – 12 : 500 yen,
13 – 18 : 800 yen, 19 years or older : 1,00 yen

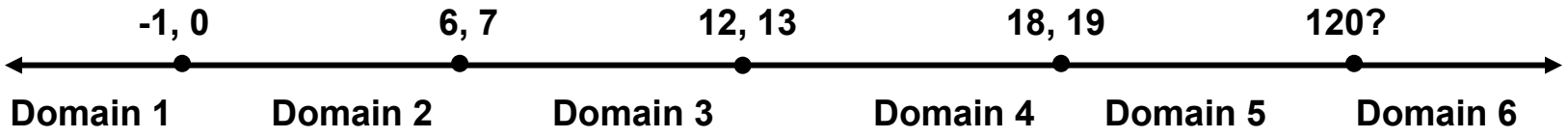


- Pick up boundary values.
- In the above case, 0, 7, 13, 19, (120?)

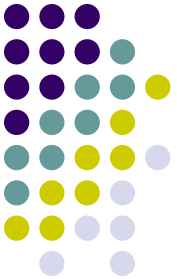
Test case designing

- In order to design test cases to detect wrong processing and wrong boundary in black box testing.
  - Test cases (equivalent partitioning + boundary analysis) =
  - Test cases (both ends of the boundaries)
  - In the below cases, -1, 0, 6, 7, 12, 13, 18, 19, (120?)

Admission fee : Below 6 year old : free, 7 – 12 : 500 yen,
13 – 18 : 800 yen, 19 years or older : 1,00 yen

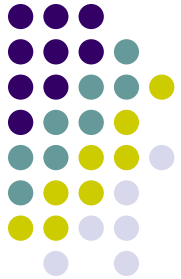


What you have to specify in the test cases



- Must for test cases
  - Test environment (if needed special environment)
  - Input data values
  - Expected outcomes
- Desirable for test cases (for validation test case quality)
  - Types of test cases
    - Normal, Error, Boundary, Environmental
  - Desk checking / Machine checking

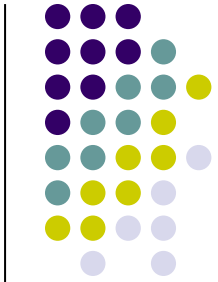
Test case designing process



- You can find many bugs while you are designing test cases, like Missing functions (hard to detect in debugging).
- Most likely they will sneak into the final product(= expensive bugs)
- Test case designing
  - = Process that defines the req. spec. with actual values.
  - = Checking req. spec.
  - = Seeing the req. spec. from different point of view

"It is not just generate test case,
but also for debugging the req. spec."

Designing test cases (part 1)



- Specify by natural language
  - You can describe complex conditions.
  - It is not easy to cover all the possible conditions.

Admission fee : Below 6 year old : free, 7 – 12 : 500 yen,
13 – 18 : 800 yen, 19 years or older : 1,00 yen

1 Checking the format of 'age'

1-1 Invalid format

1-1-1 : Enter 'PR' as 'age' and check if an error message is displayed.

1-1-2 : Enter 'PR' as 'age and check if a retry screen is displayed.

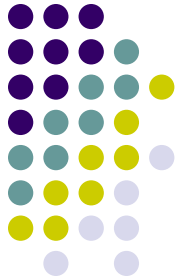
1-2 Valid format

1-2-1 : Enter '6' as 'age' and check if 0 is displayed as an admission fee.

1-2-2 : Enter '7' as 'age' and check if 500 is displayed as an admission fee.

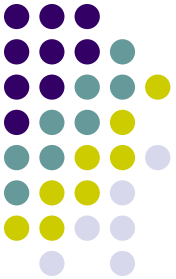
1-2-3 : Enter '13' as 'age' and check if 800 is displayed as an admission fee.

Designing test cases (part 2)



- Specify with matrix
  - It is easy to cover all the possible conditions.
  - It is easy to generate test cases
    (sometime, generate too many)
  - You may on be able to describe complex conditions.

Designing test cases (part 2)

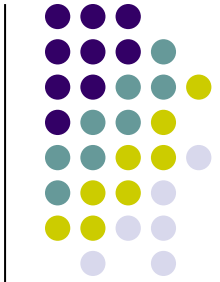


Specify with matrix

Admission fee : Below 6 year old : free, 7 – 12 : 500 yen, 13 – 18 : 800 yen, 19 years or older : 1,00 yen

Test IDs		AD001	AD002	AD003	AD004	AD005	AD006	AD007	AD008
Age	ZZ	X							
	-1		X						
	0			X					
	6				X				
	12					X			
	18						X		
	19							X	
	999								X
fee	0			X	X				
	500					X			
	800						X		
	1000							X	
	error	X	X						X
type		err	bound	norm	norm	norm	norm	norm	err
desk checking		5/05	5/05	5/05	5/05	5/05	5/05	5/05	5/05
machine checking		6/01	6/01	6/02	6/02	6/02	6/02	6/03	6/03

Practice



Design test cases for the Triangle program using matrix based + natural language