Vérification

approches, bancs de test et méthodologie

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Février 2017

Fonctionnement d'un banc de test

- Assignation des entrées du circuit à tester
- Détermination des valeurs des sorties attendues (références)
- Attente d'un délai
  - Permet l'évaluation des sorties par le circuit (temps de propagation)
- Vérification des sorties
  - Comparaison avec la référence
  - Indication claire en cas d'erreur

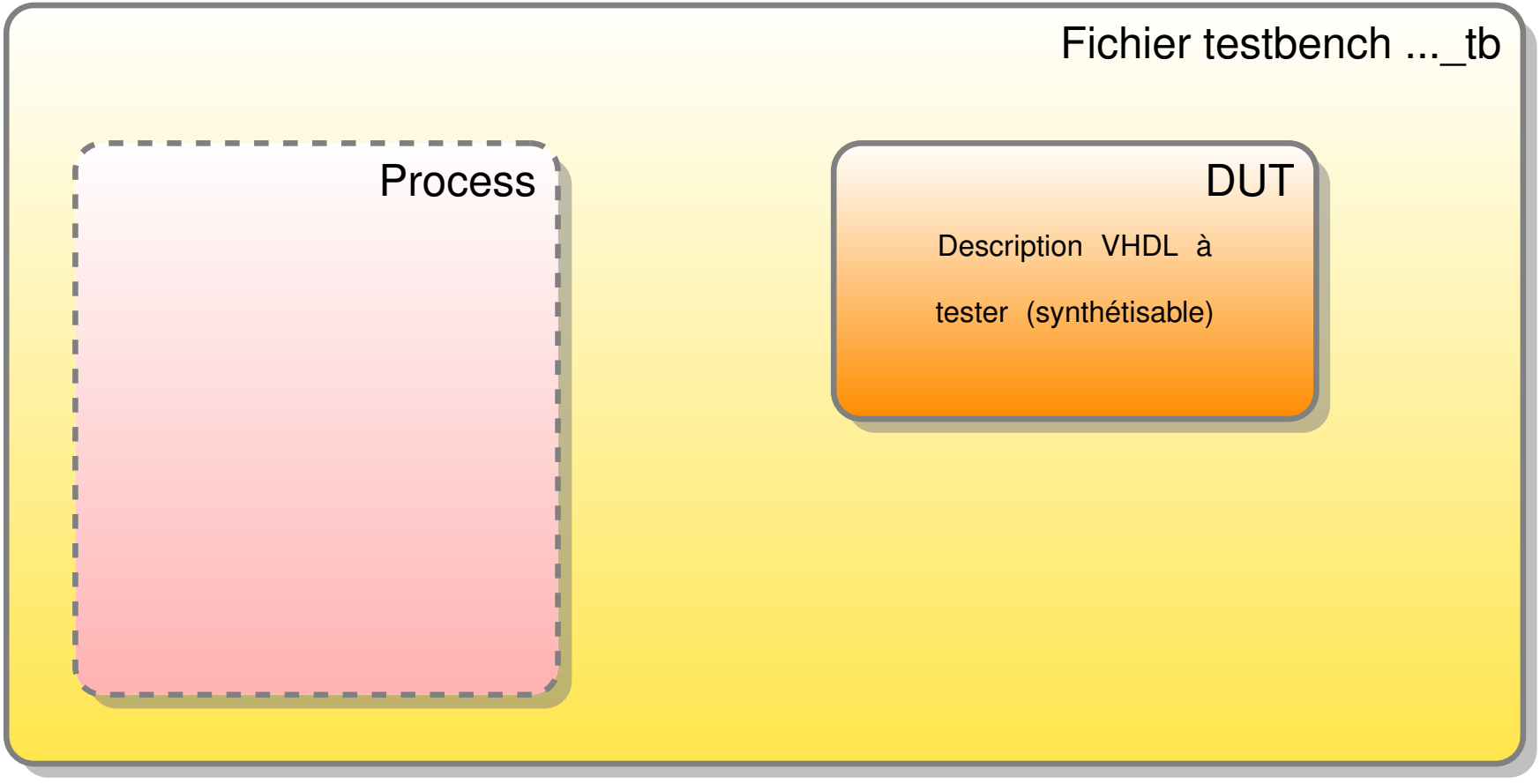
Structure de banc de test

- Solution applicable pour un design standard
- Un seul processus pour la stimulation/vérification... Attention, vite limité

Structure de banc de test

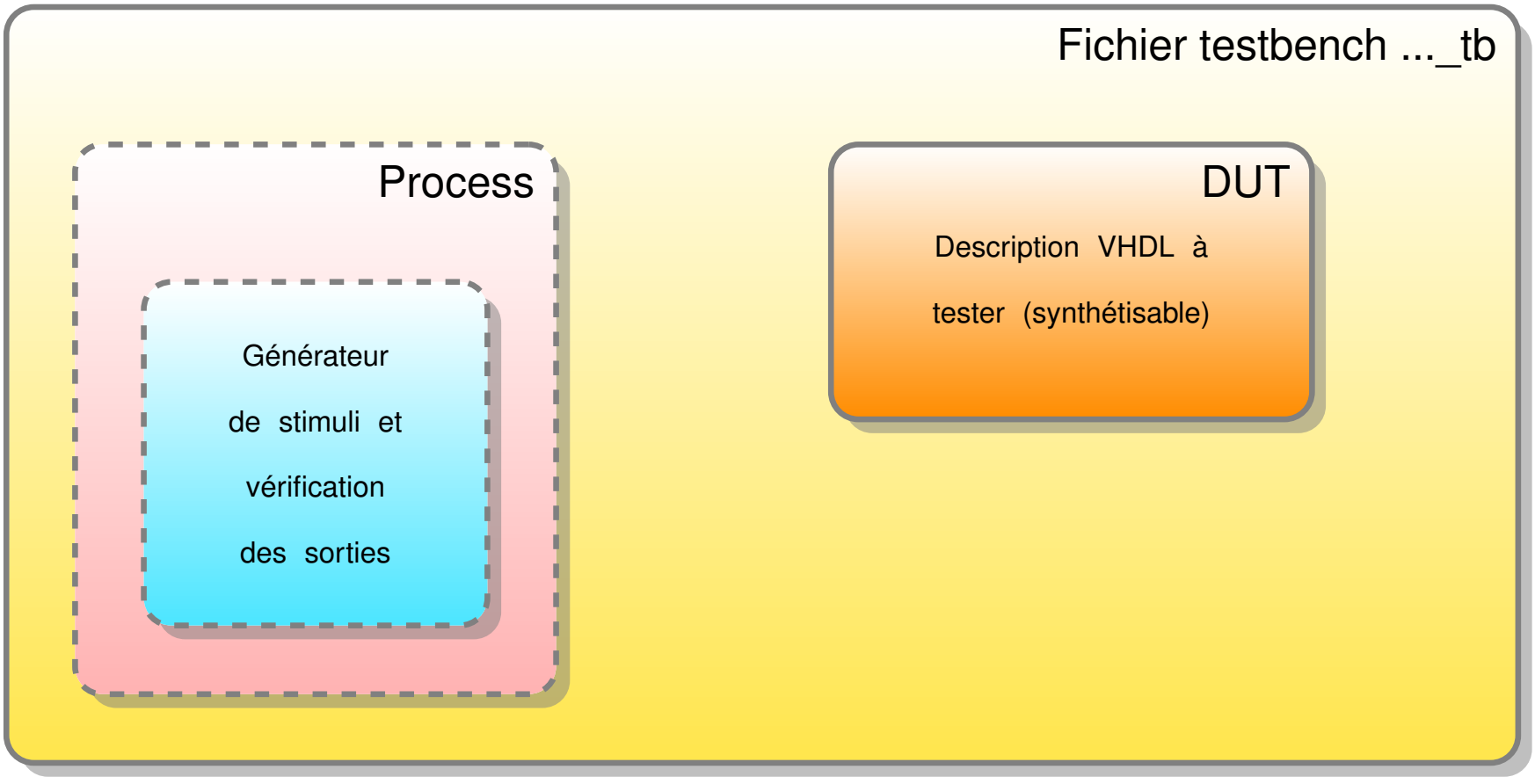
- Solution applicable pour un design standard
- Un seul processus pour la stimulation/vérification... Attention, vite limité

Structure de banc de test



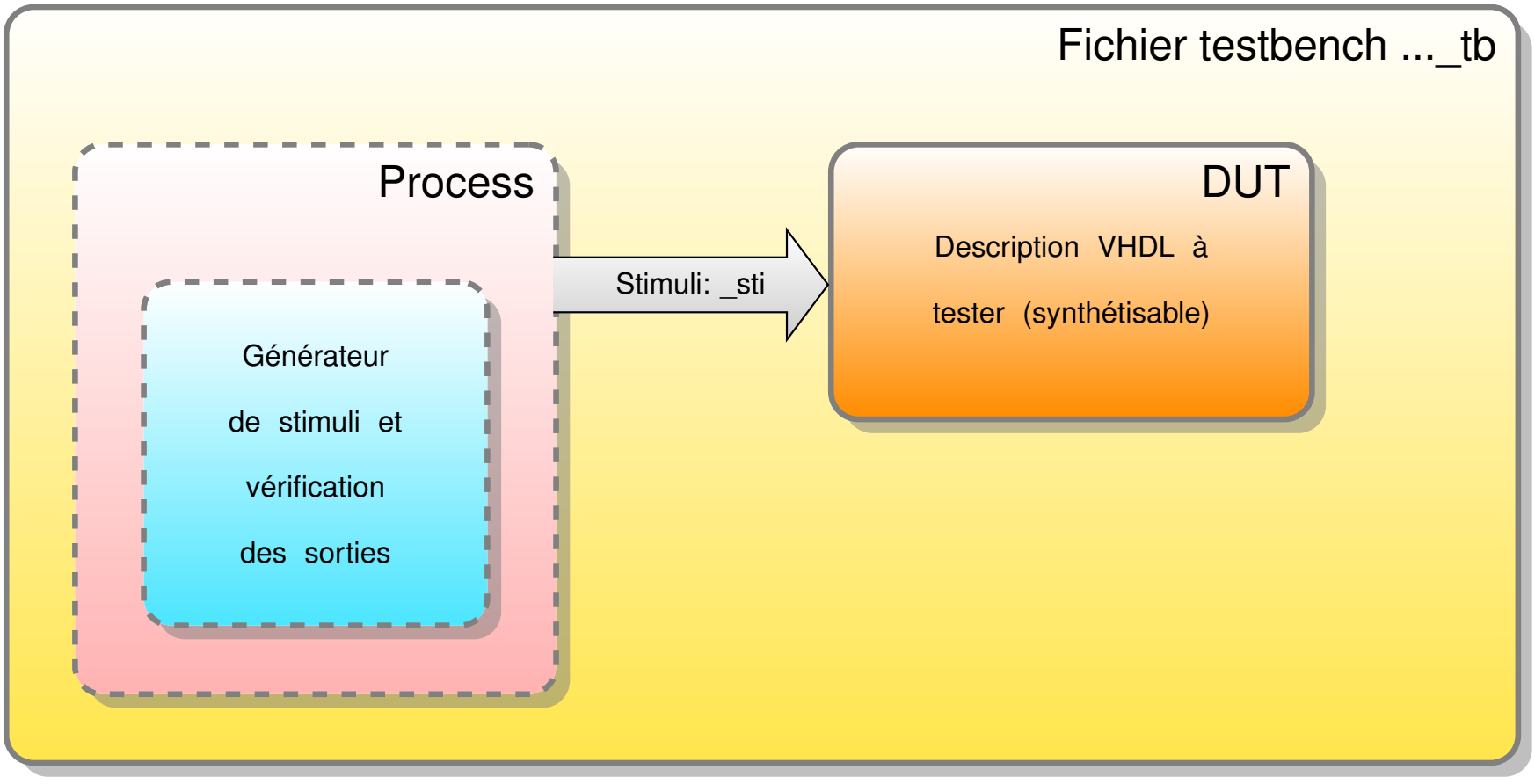
- Solution applicable pour un design standard
- Un seul processus pour la stimulation/vérification... Attention, vite limité

Structure de banc de test



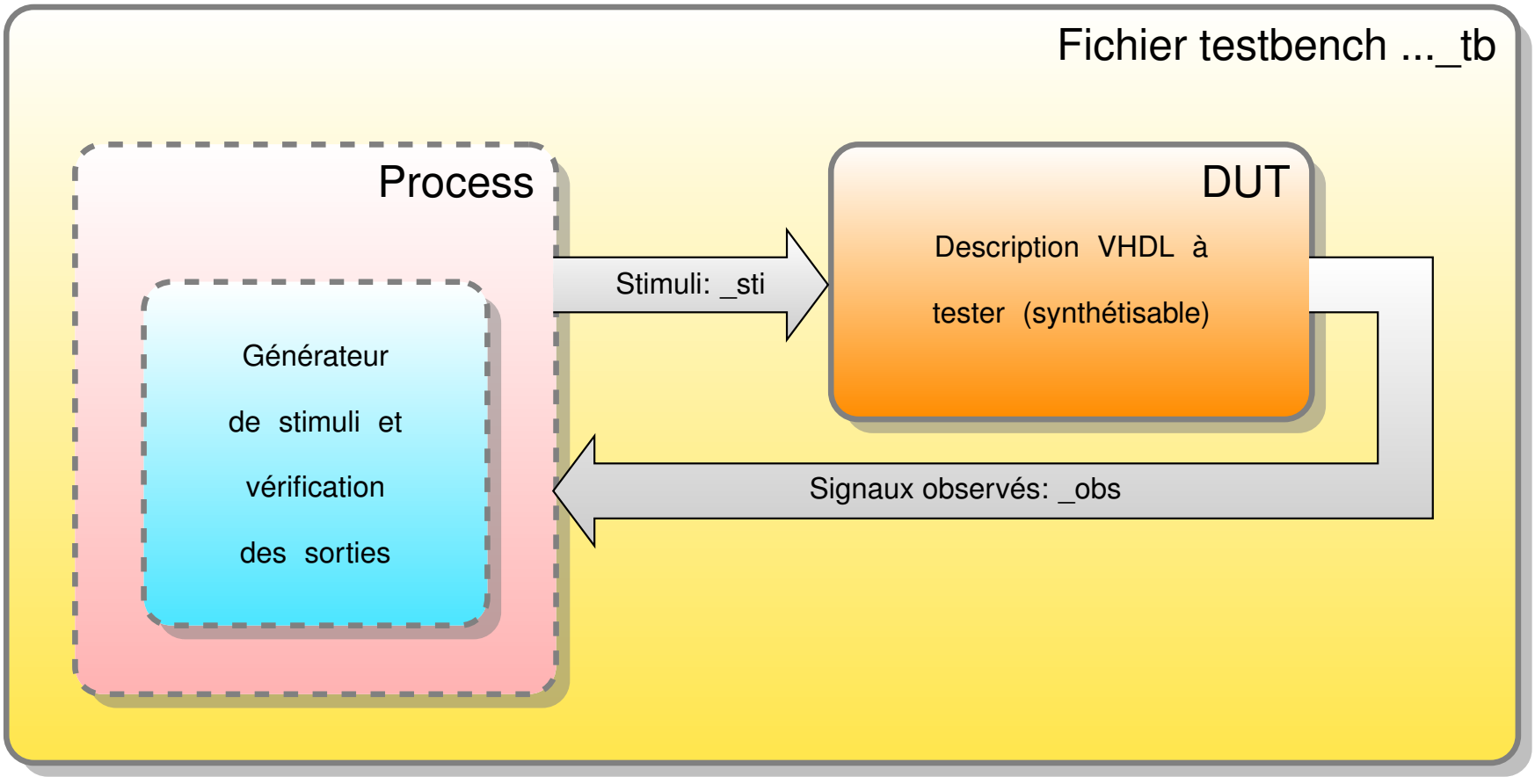
- Solution applicable pour un design standard
- Un seul processus pour la stimulation/vérification... Attention, vite limité

Structure de banc de test



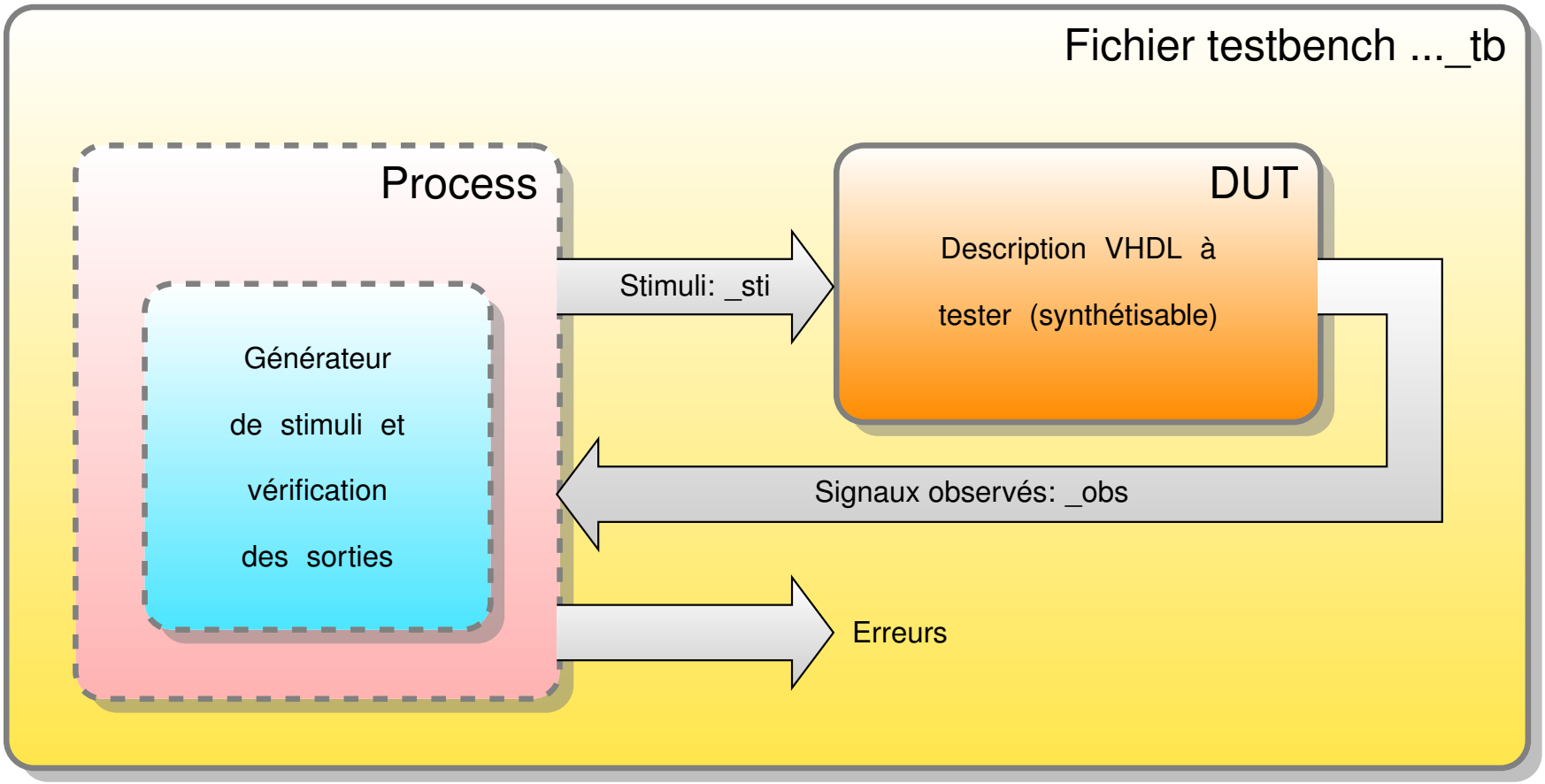
- Solution applicable pour un design standard
- Un seul processus pour la stimulation/vérification... Attention, vite limité

Structure de banc de test



- Solution applicable pour un design standard
- Un seul processus pour la stimulation/vérification... Attention, vite limité

Structure de banc de test



- Solution applicable pour un design standard
- Un seul processus pour la stimulation/vérification... Attention, vite limité

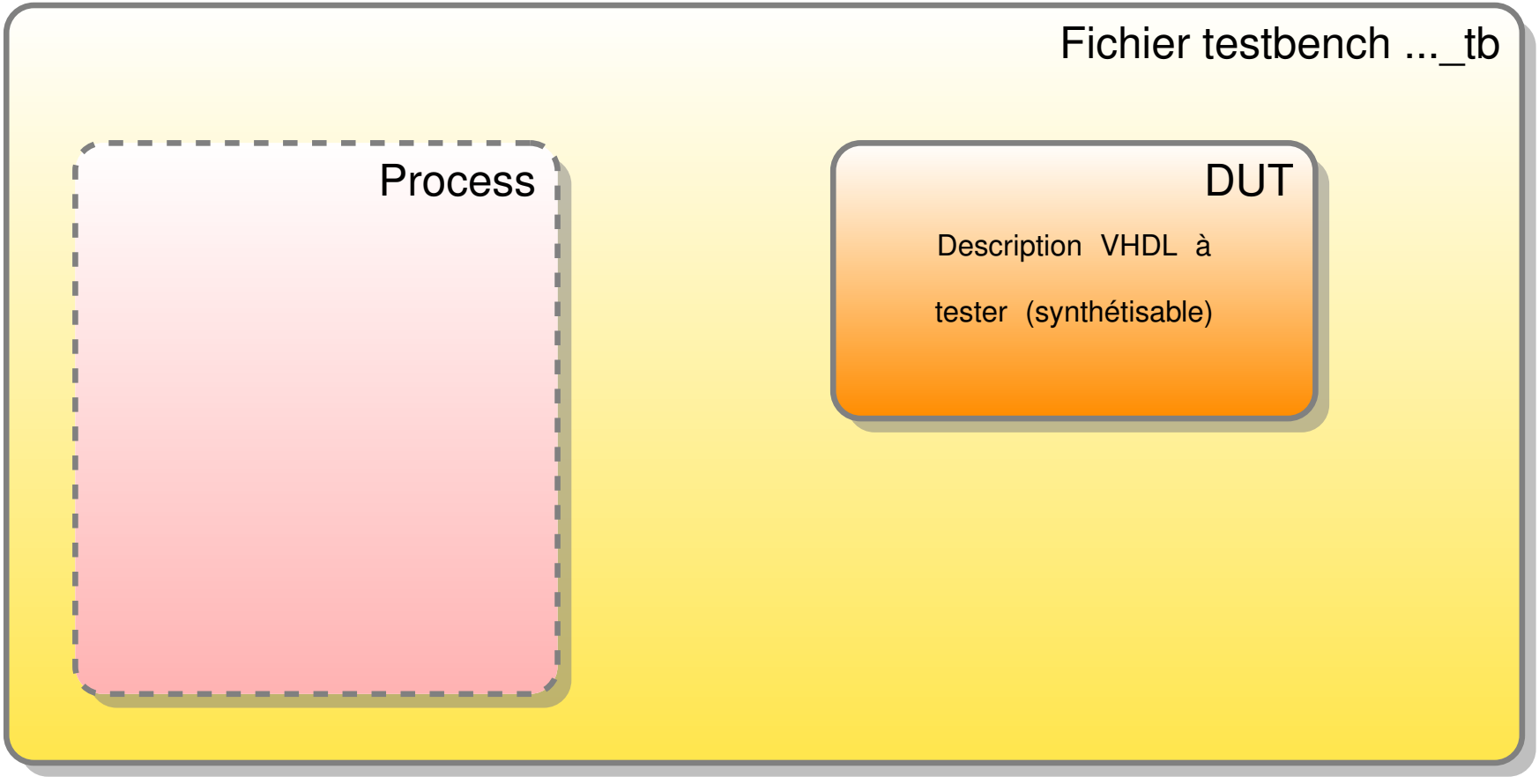
Structure de banc de test

- Solution applicable pour un design standard
- Les processus de stimulation/vérification sont dans le même fichier

Structure de banc de test

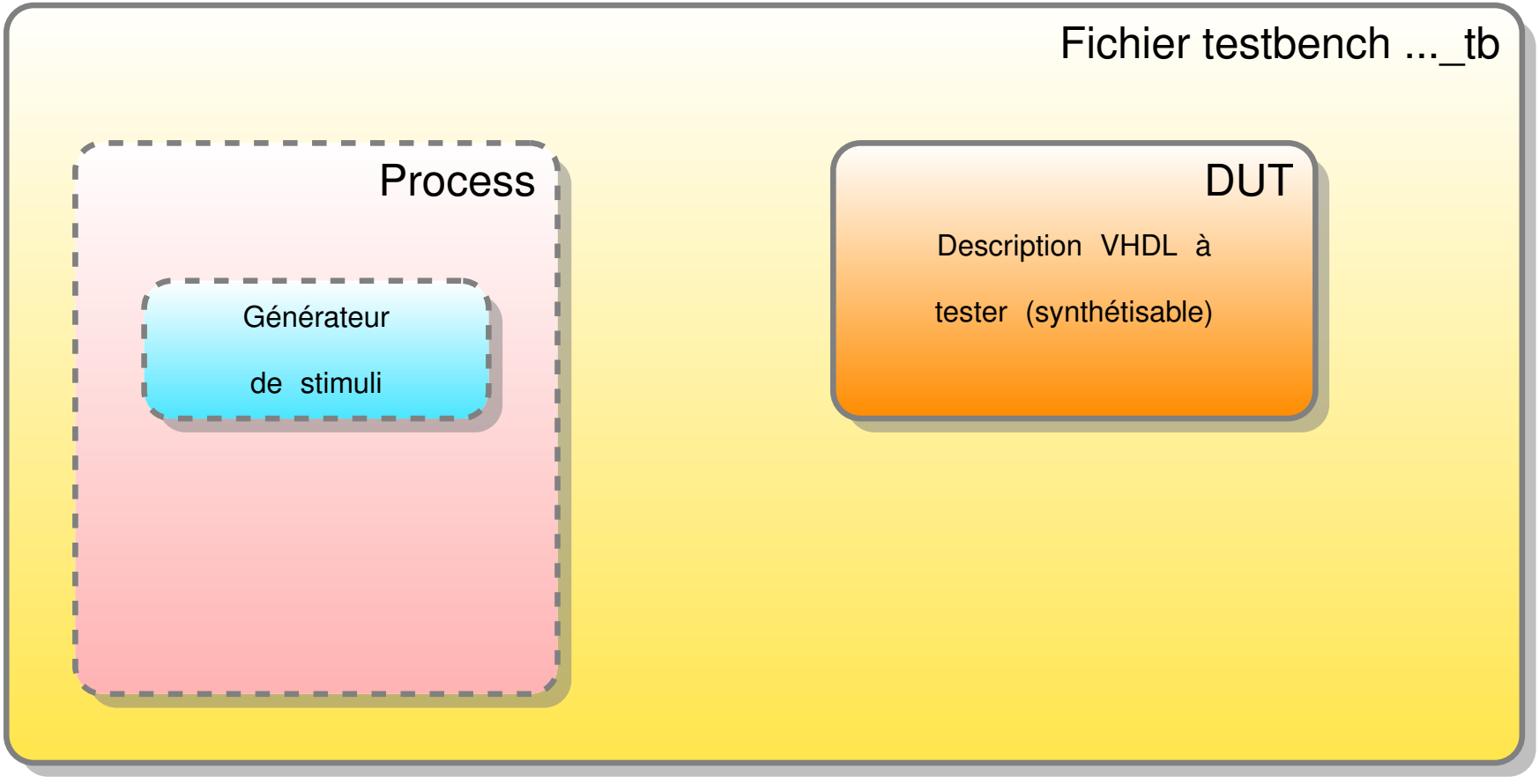
- Solution applicable pour un design standard
- Les processus de stimulation/vérification sont dans le même fichier

Structure de banc de test



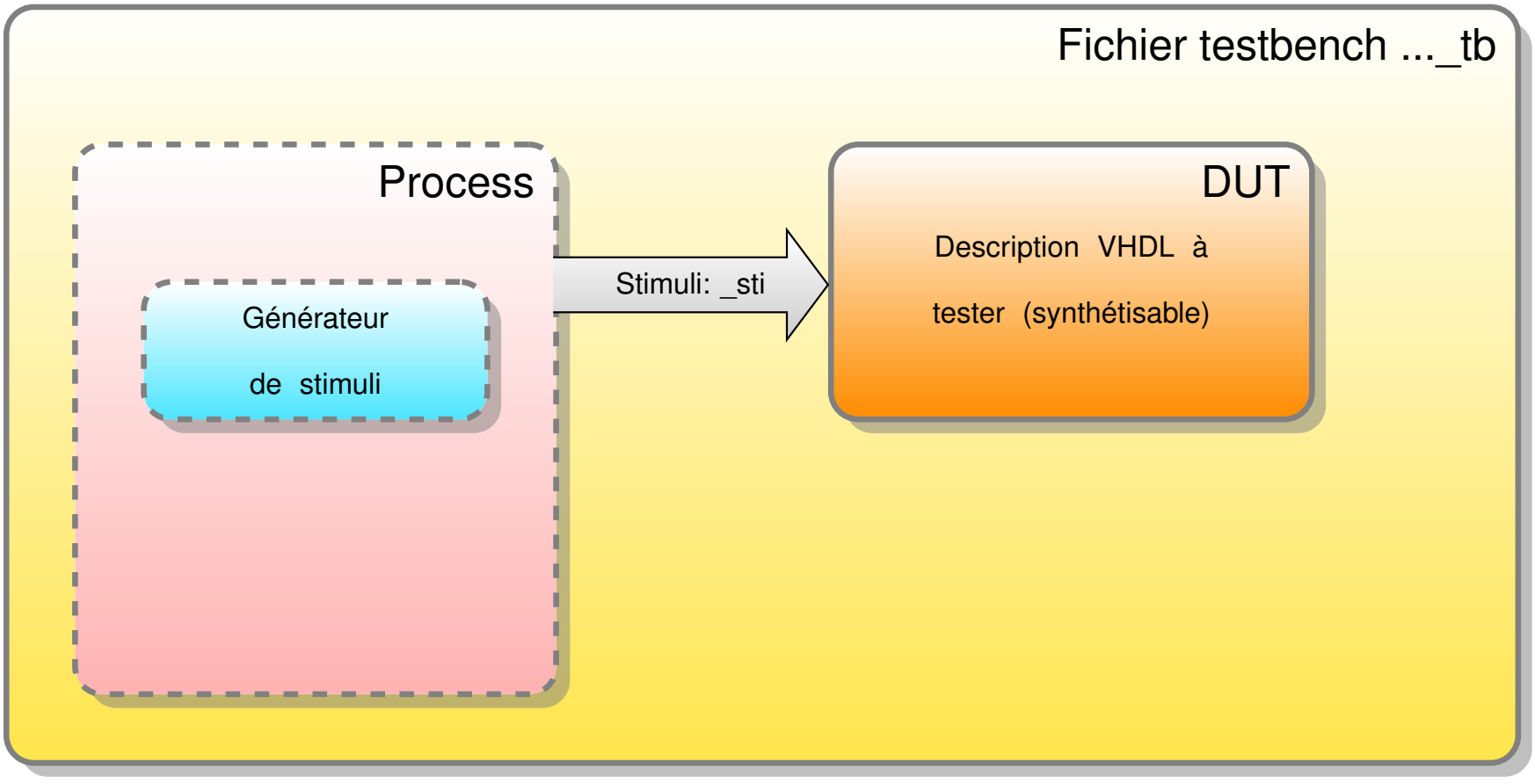
- Solution applicable pour un design standard
- Les processus de stimulation/vérification sont dans le même fichier

Structure de banc de test



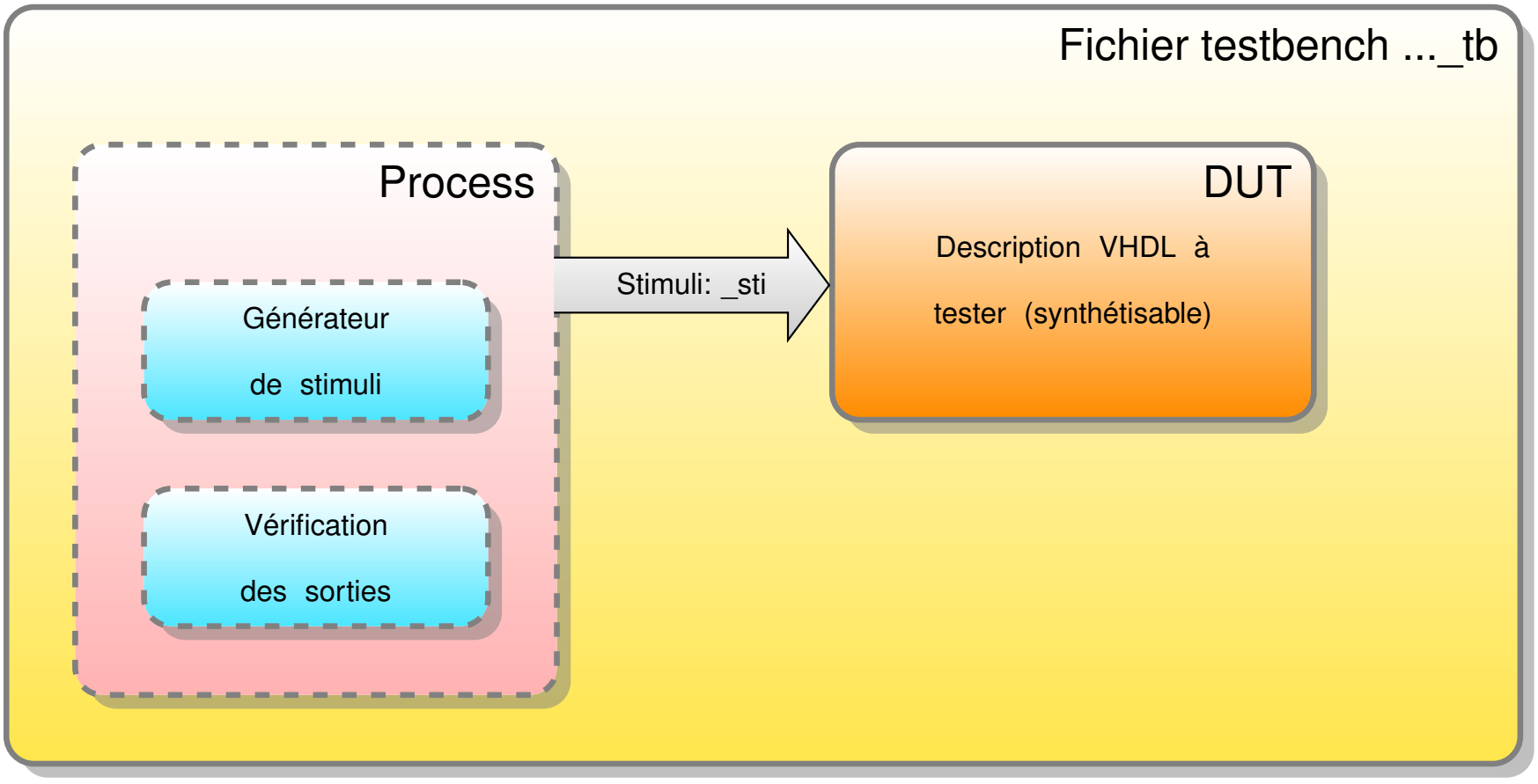
- Solution applicable pour un design standard
- Les processus de stimulation/vérification sont dans le même fichier

Structure de banc de test



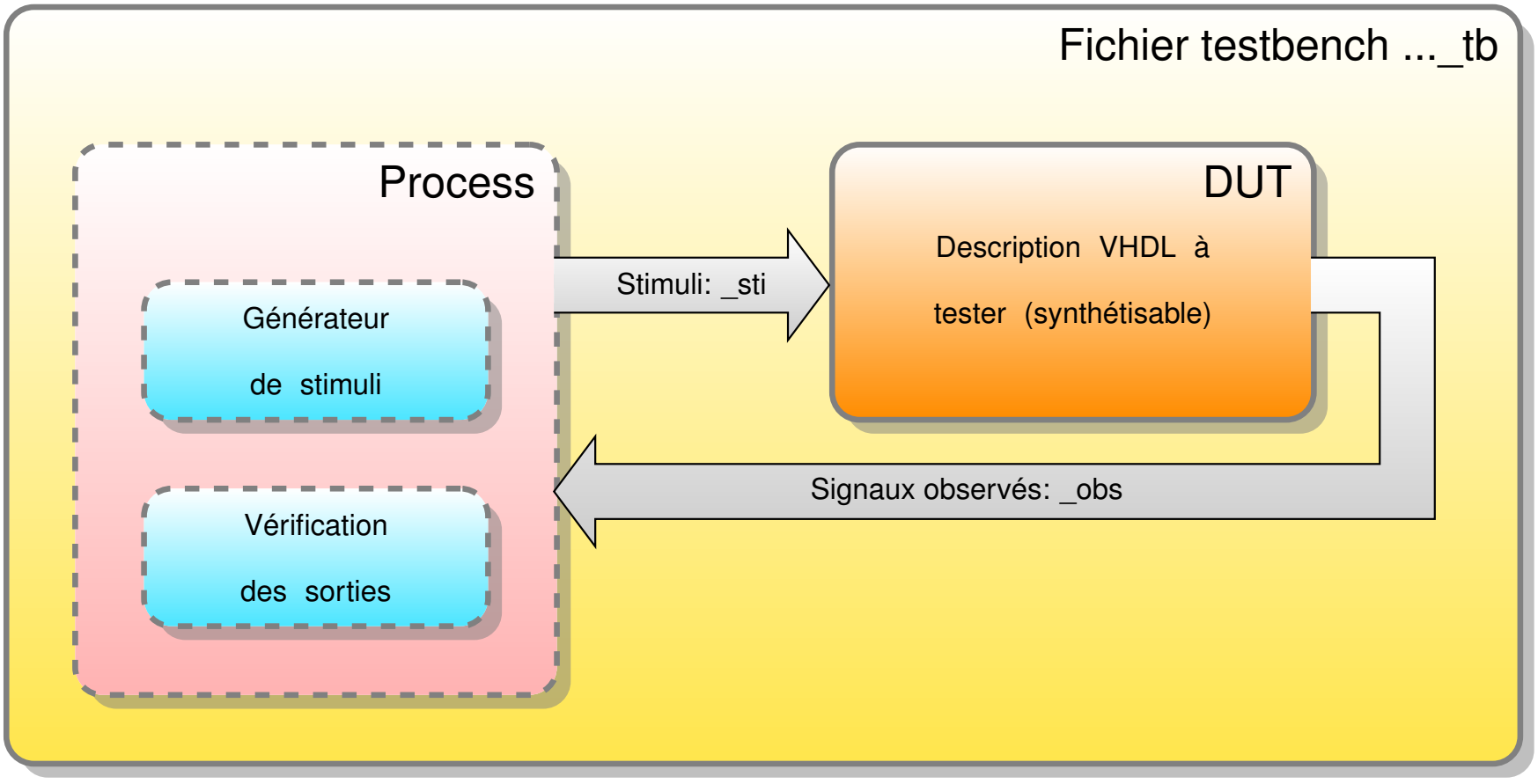
- Solution applicable pour un design standard
- Les processus de stimulation/vérification sont dans le même fichier

Structure de banc de test



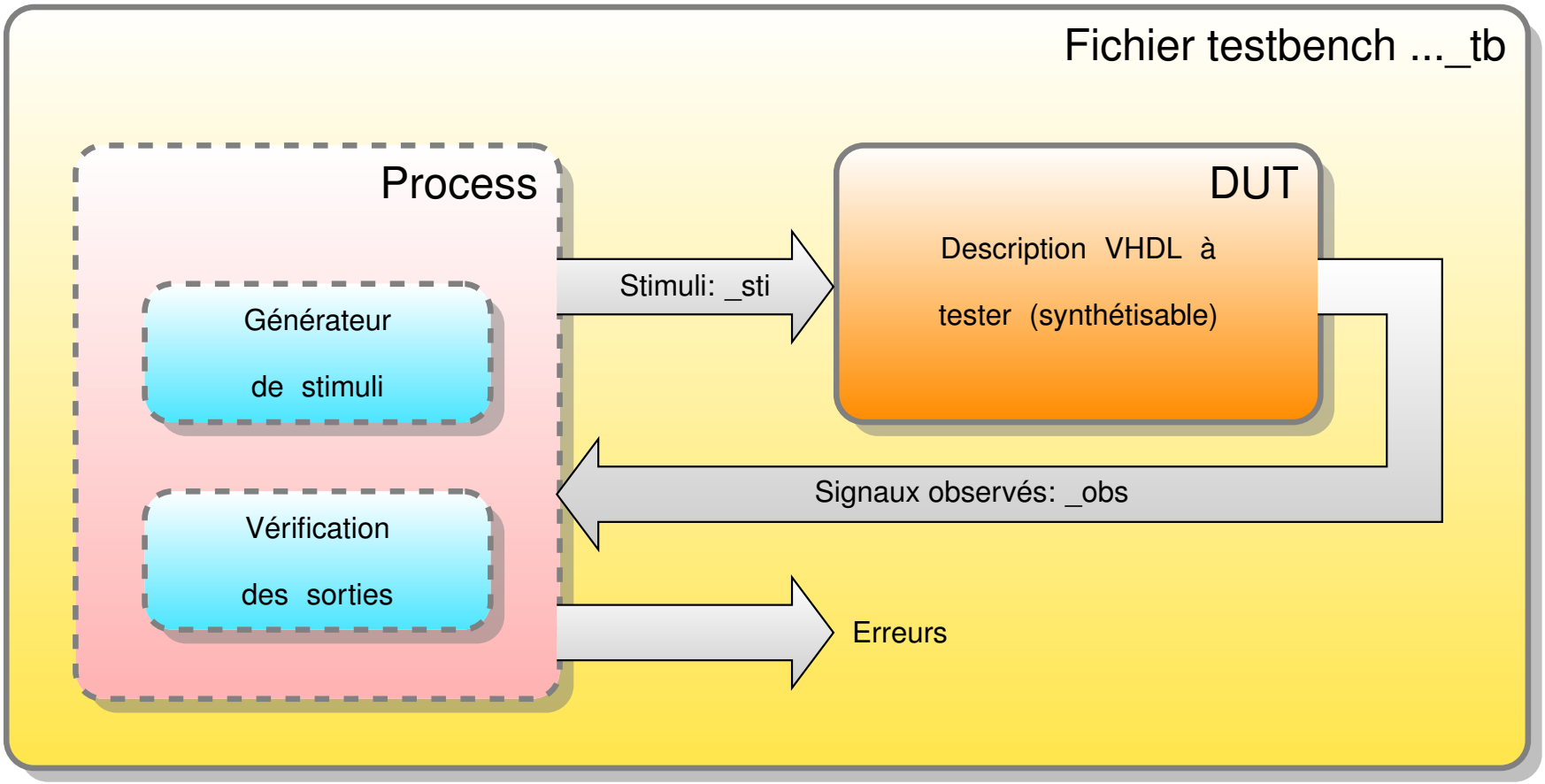
- Solution applicable pour un design standard
- Les processus de stimulation/vérification sont dans le même fichier

Structure de banc de test



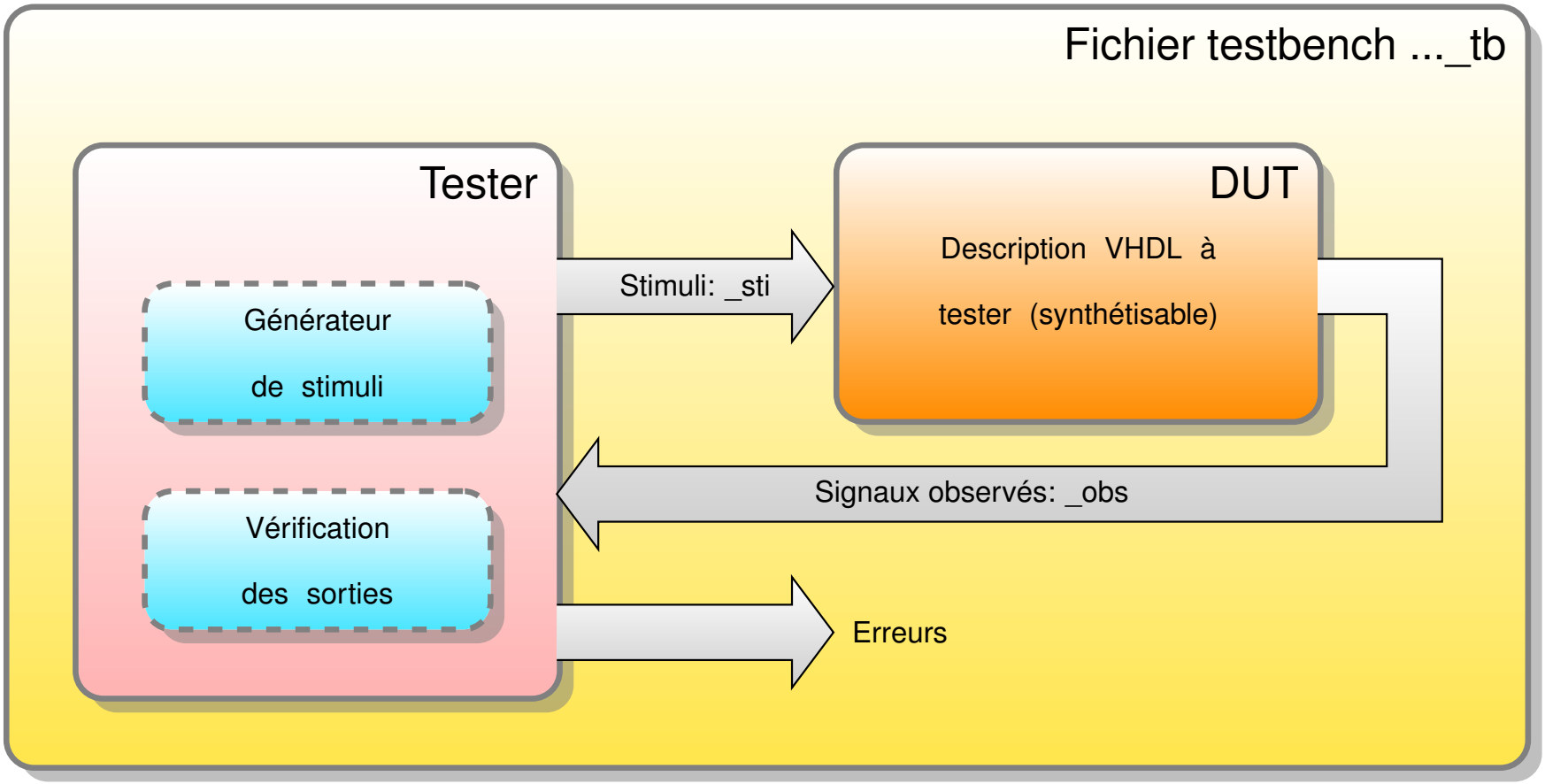
- Solution applicable pour un design standard
- Les processus de stimulation/vérification sont dans le même fichier

Structure de banc de test



- Solution applicable pour un design standard
- Les processus de stimulation/vérification sont dans le même fichier

Structure de banc de test: solution hiérarchique

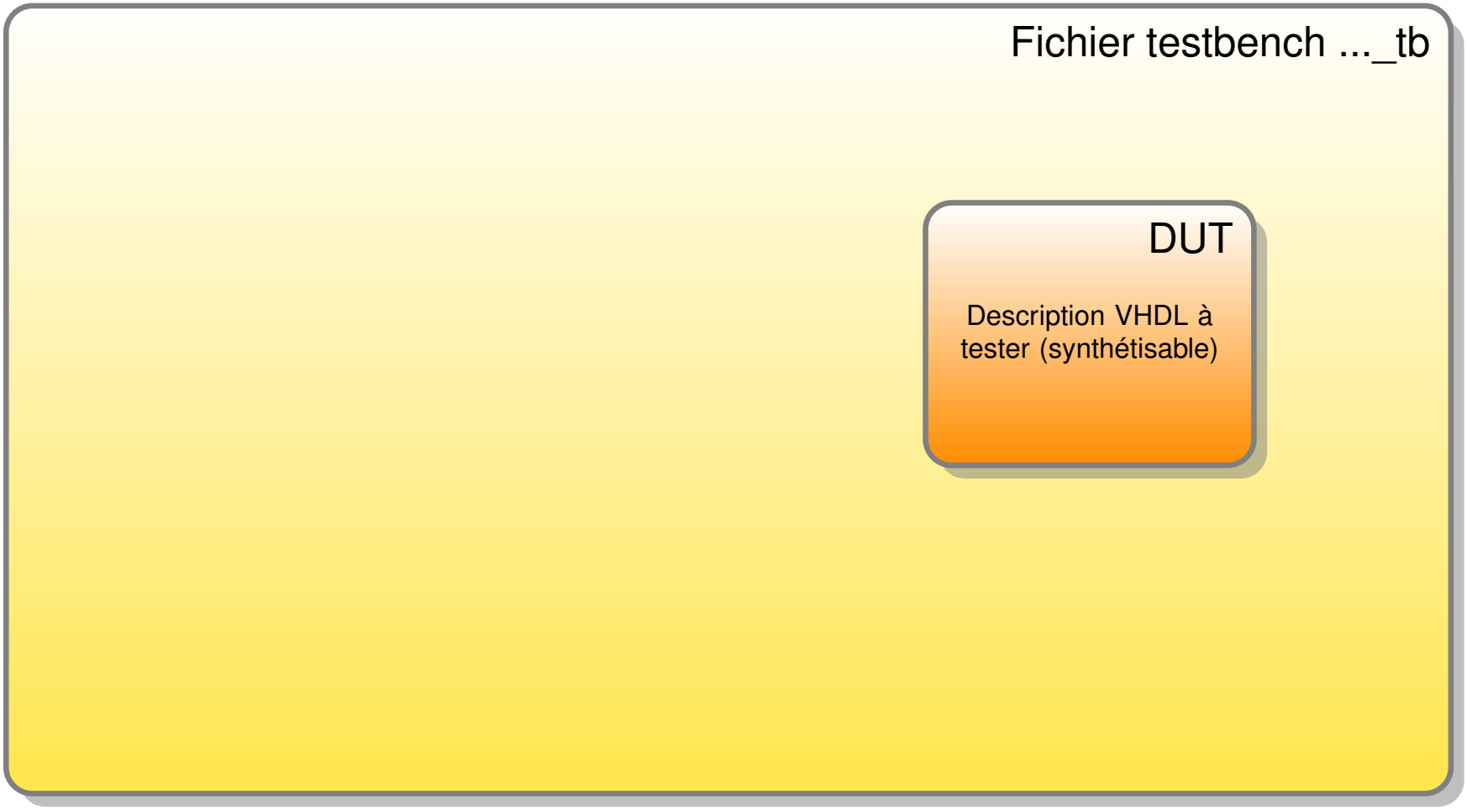


- Solution utilisée avec des outils EDA graphiques (type HDL Designer) ou pour des circuits complexes
- Le fichier de simulation peut comprendre plusieurs modules VHDL

Structure de banc de test

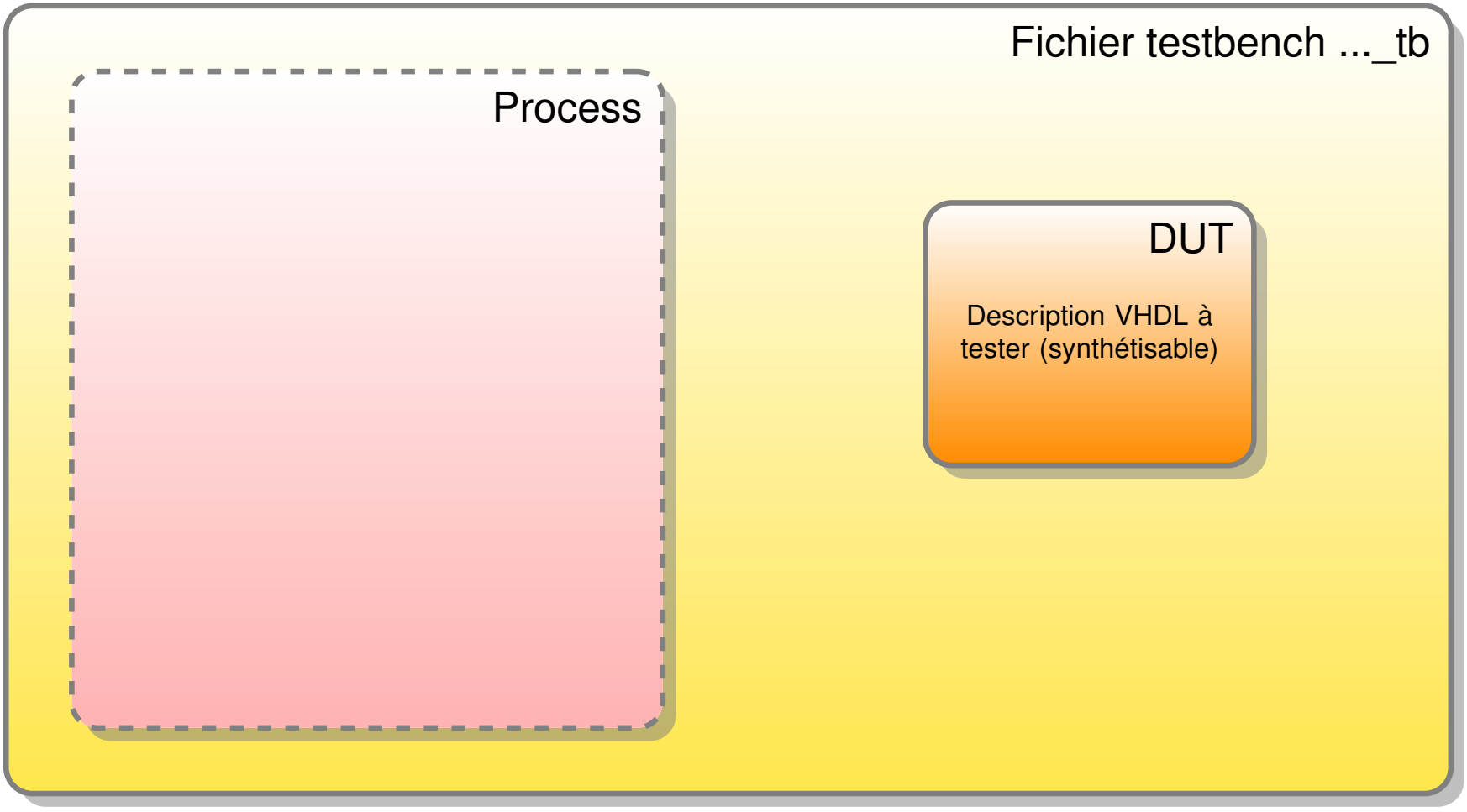
- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test



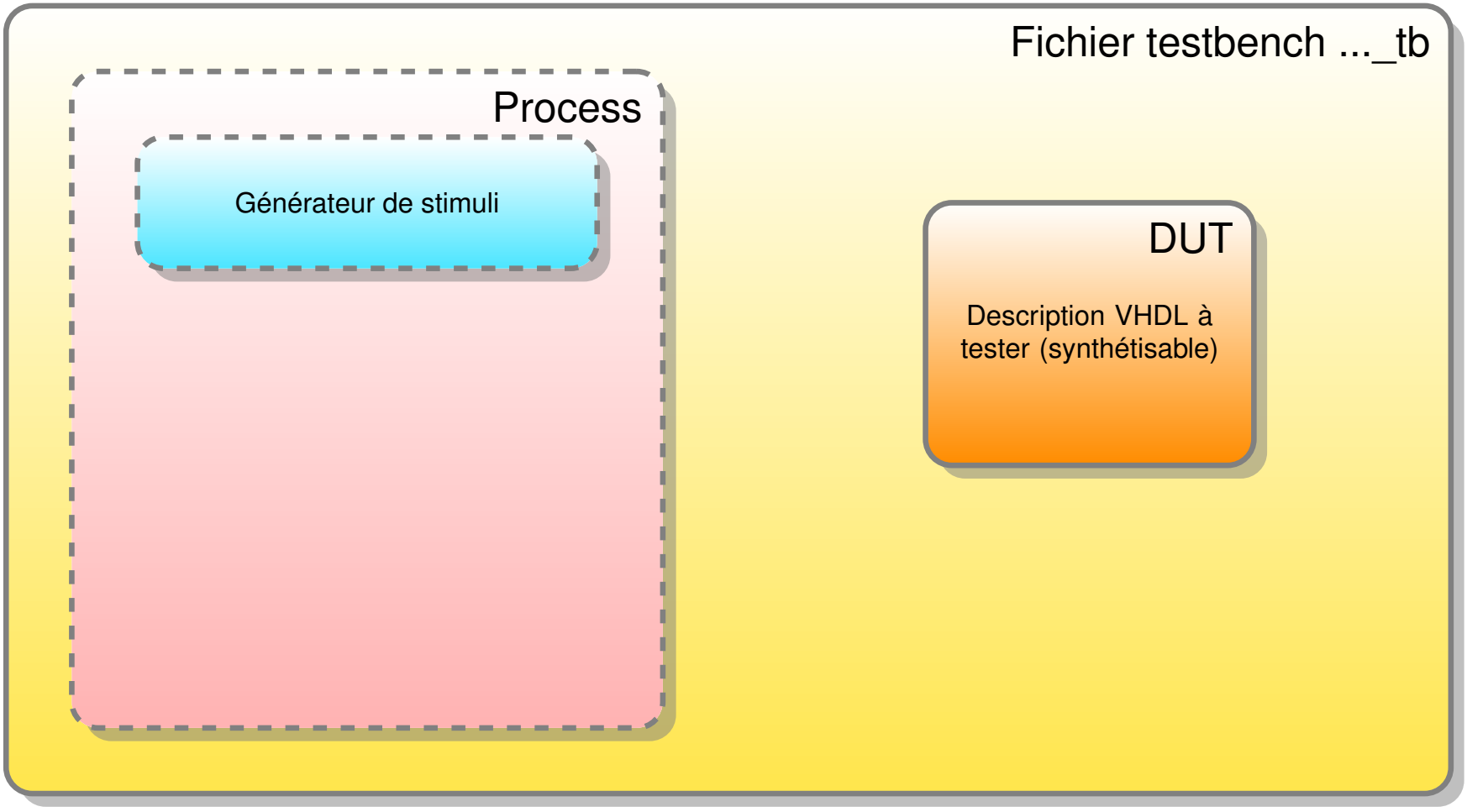
- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test



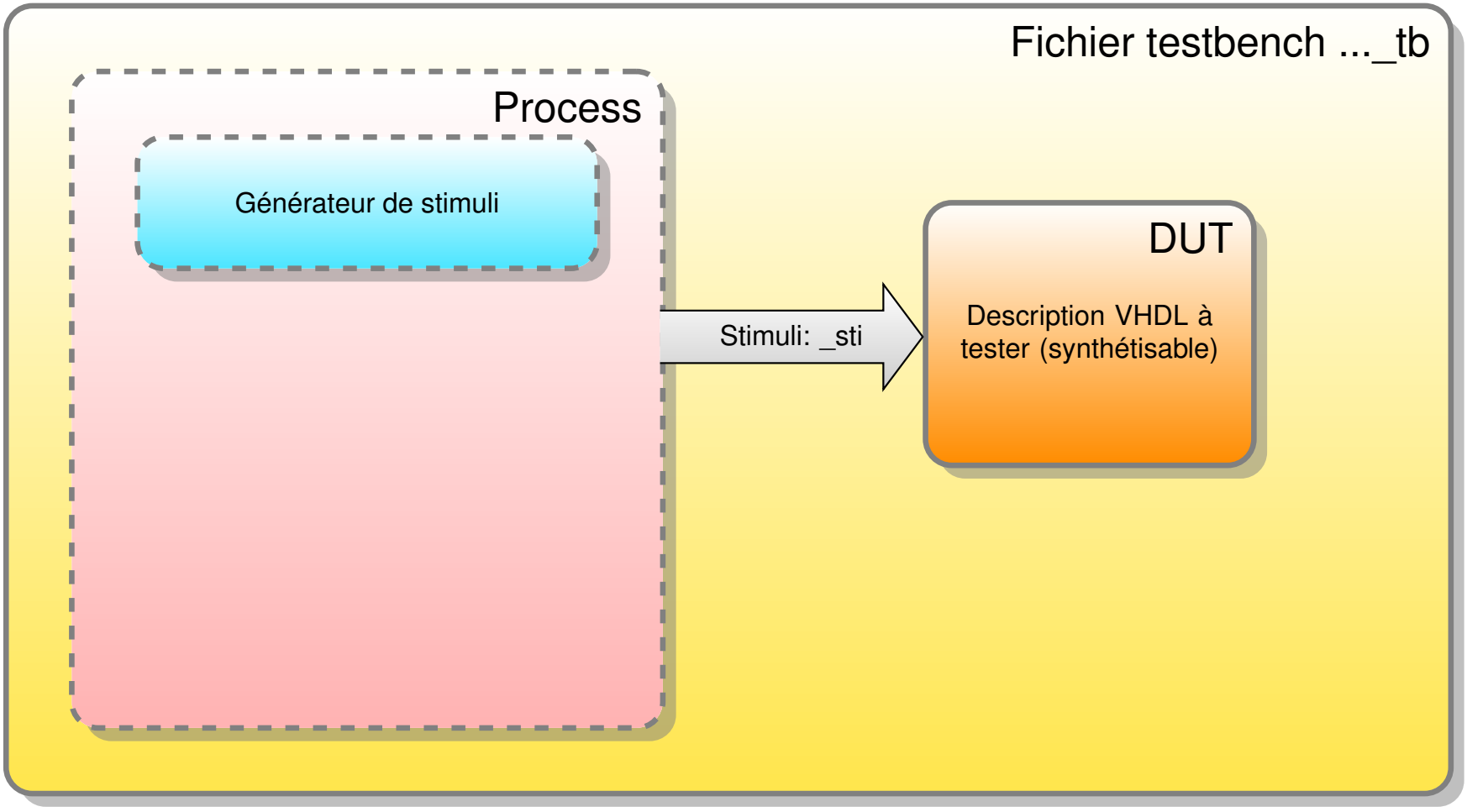
- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test



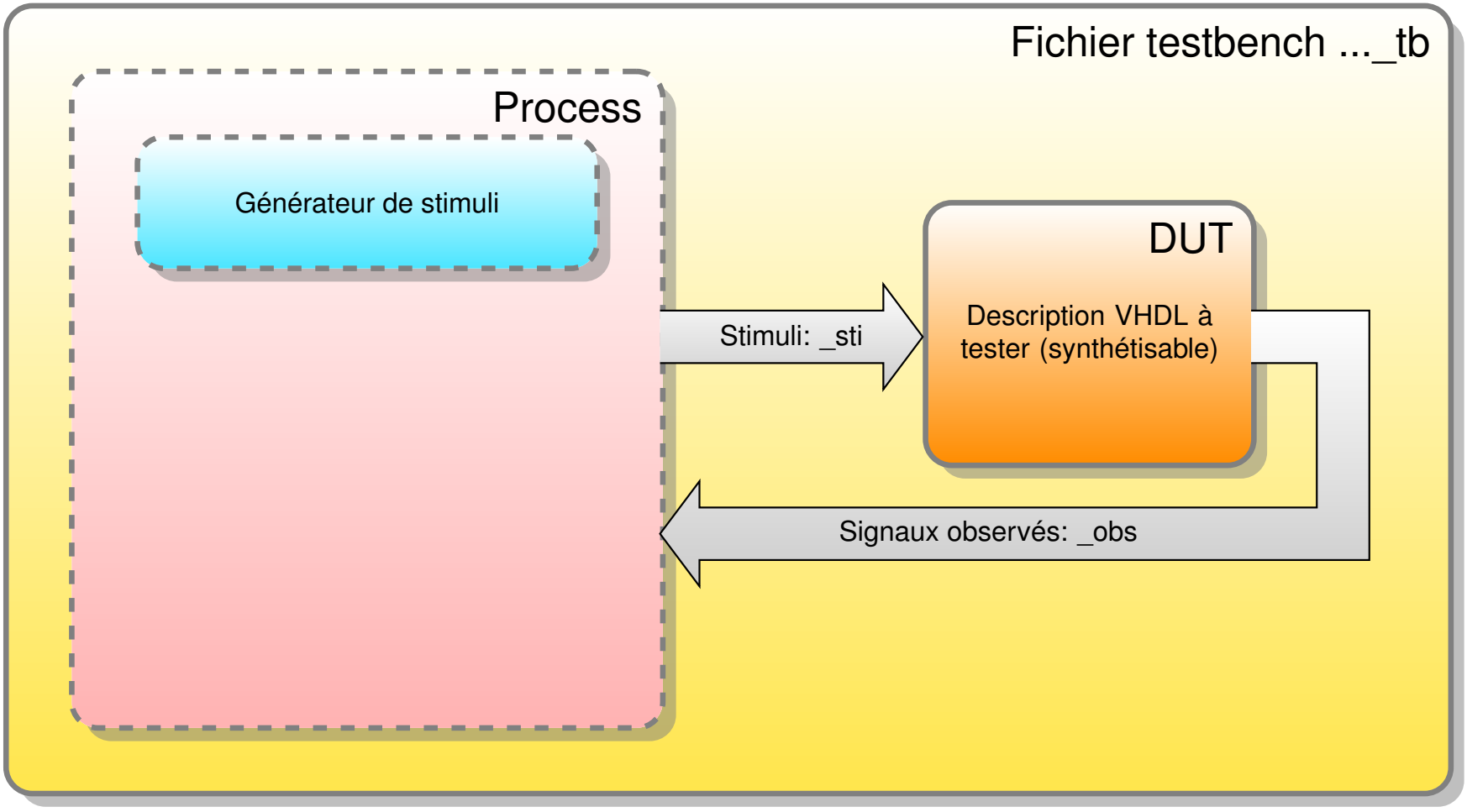
- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test



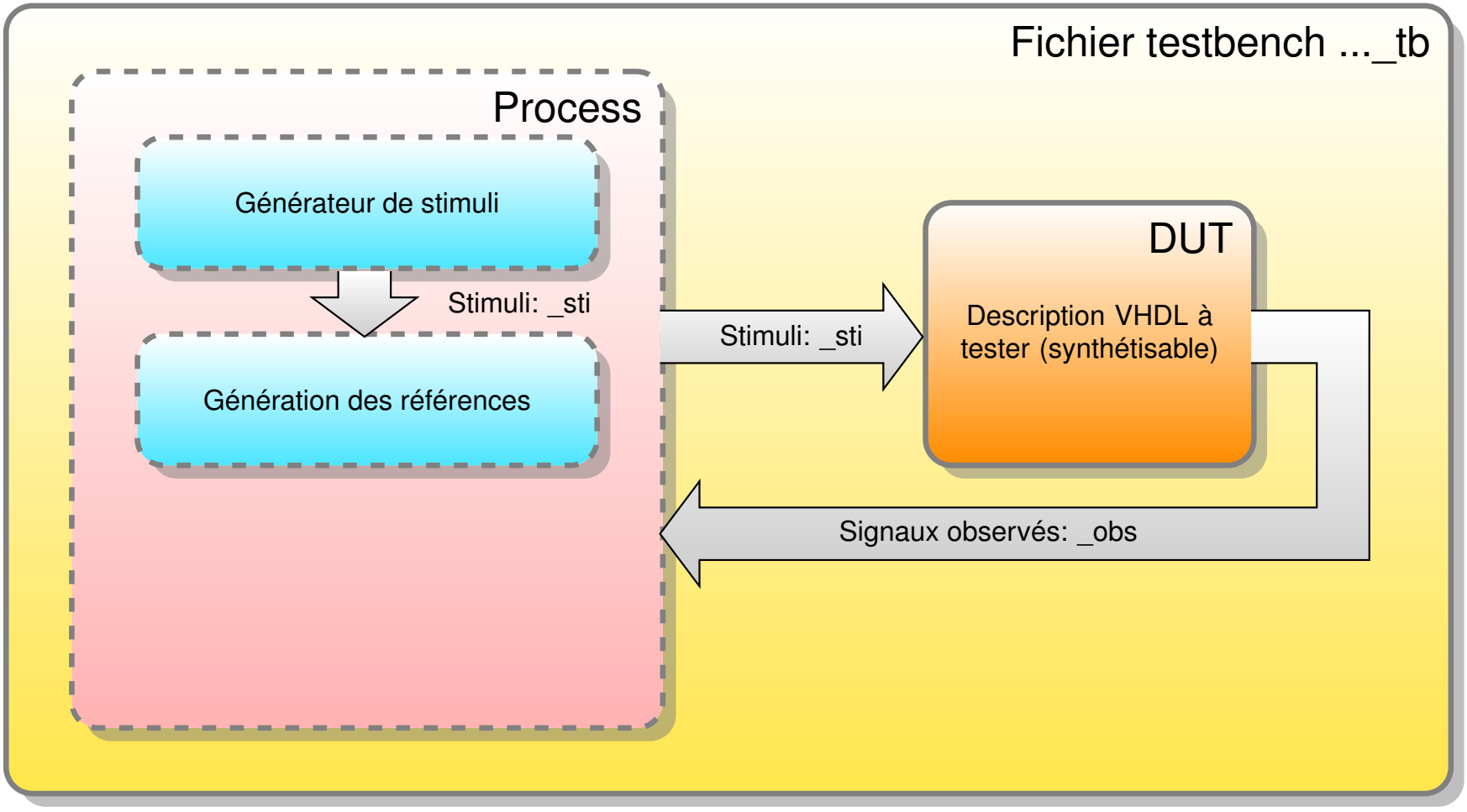
- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test



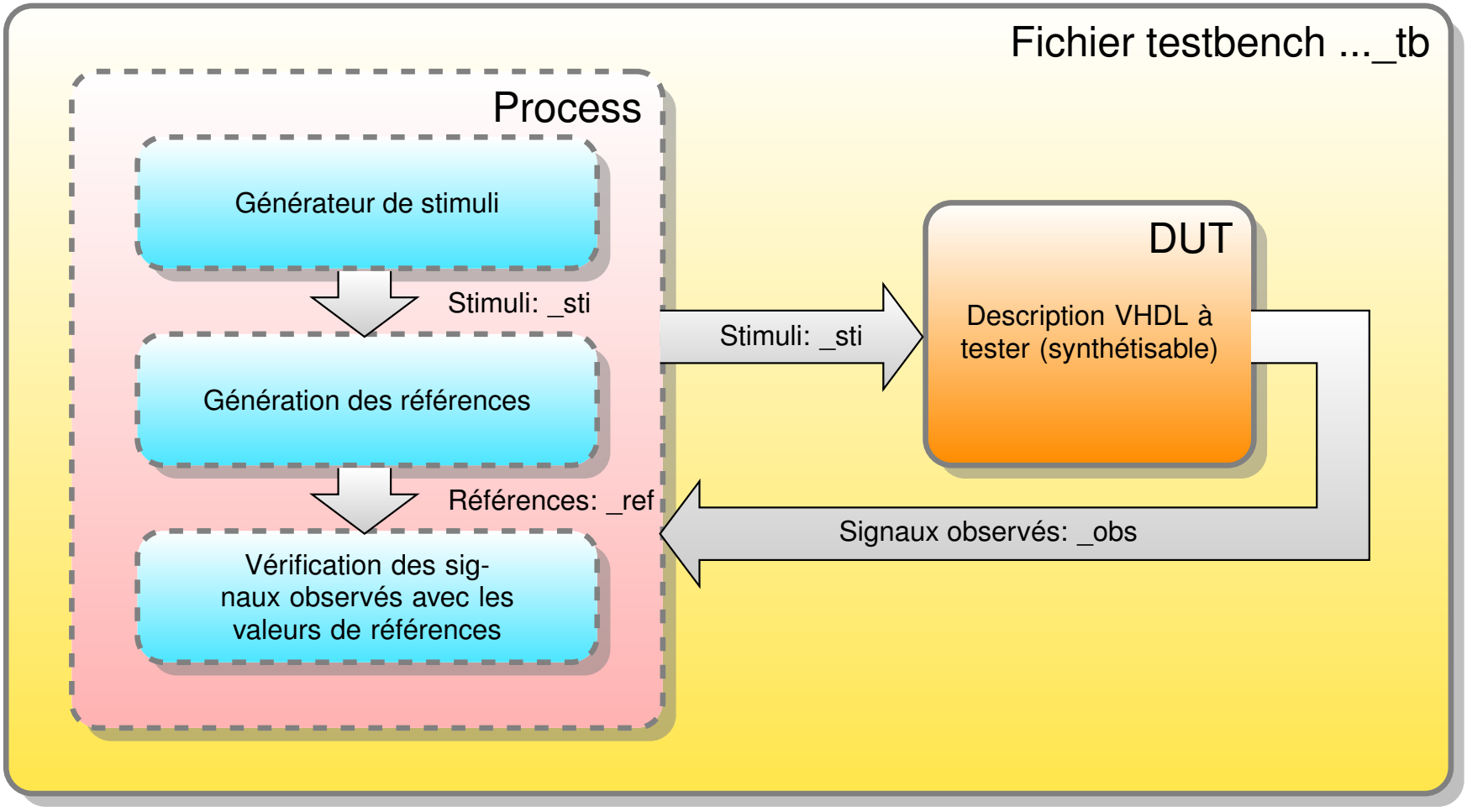
- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test



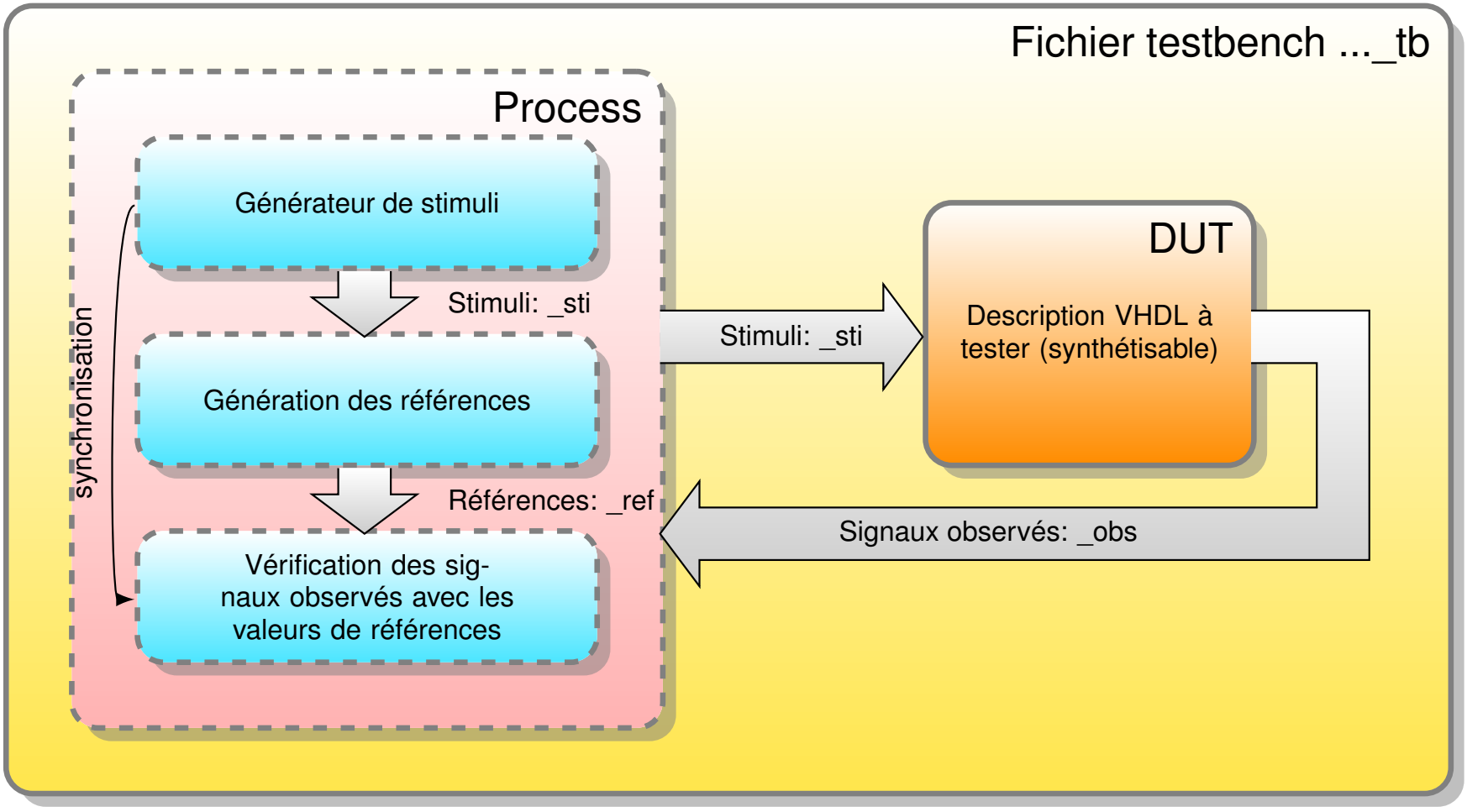
- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test



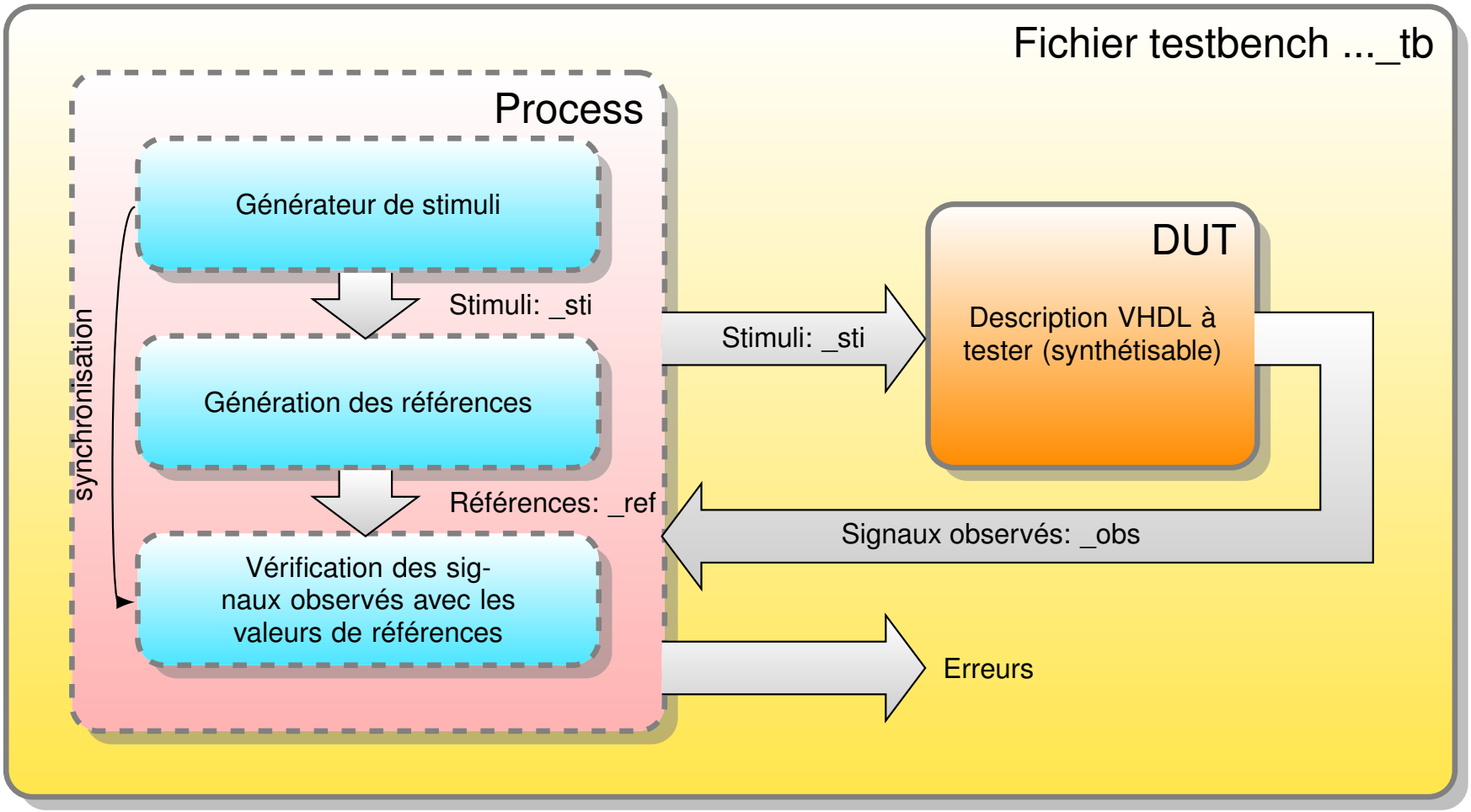
- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test



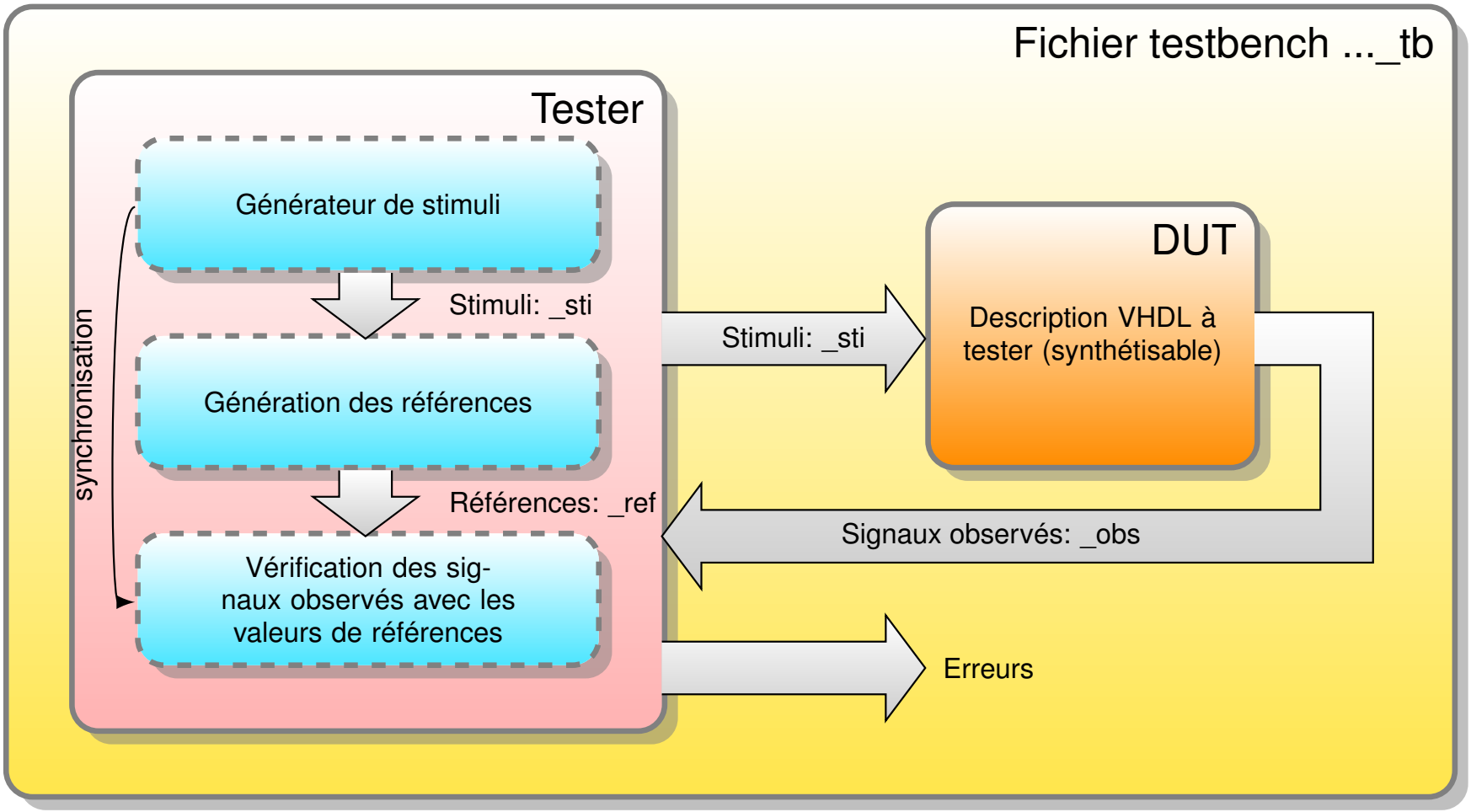
- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test



- Variante avec calcul des références par algorithme
- Structure sans hiérarchie

Structure de banc de test

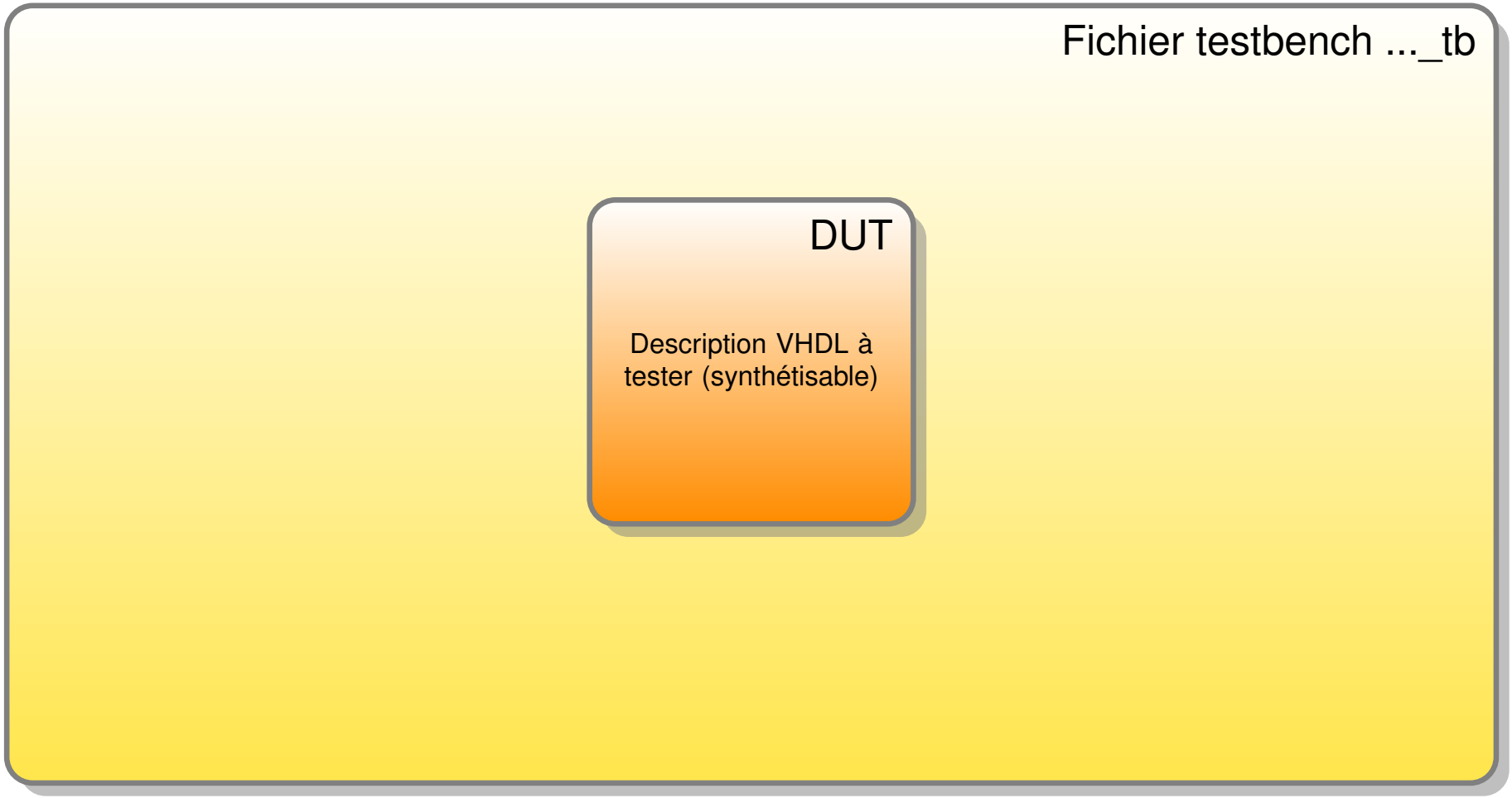


- Variante avec calcul des références par algorithme
- Structure avec hiérarchie

Structure de banc de test

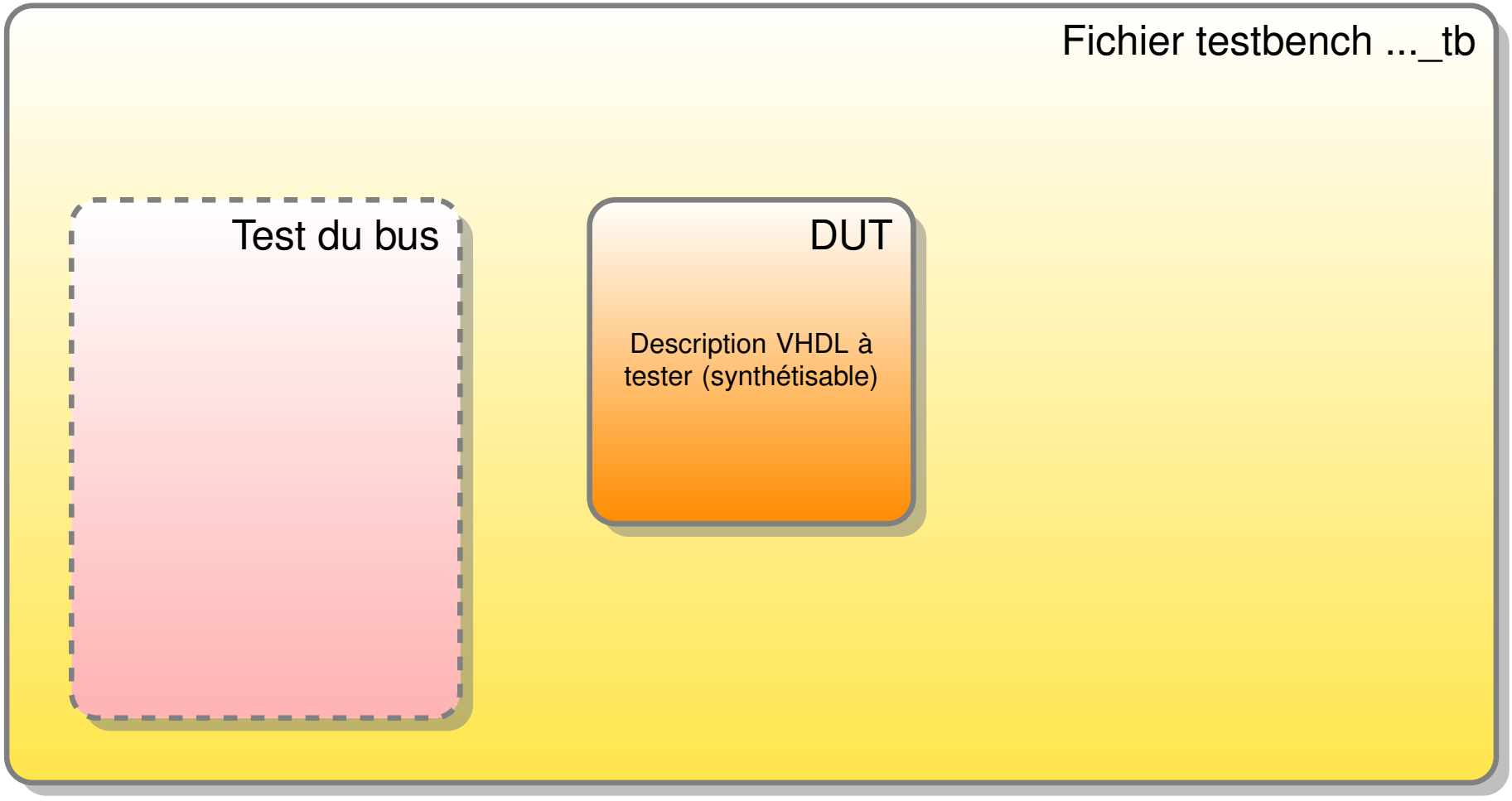
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



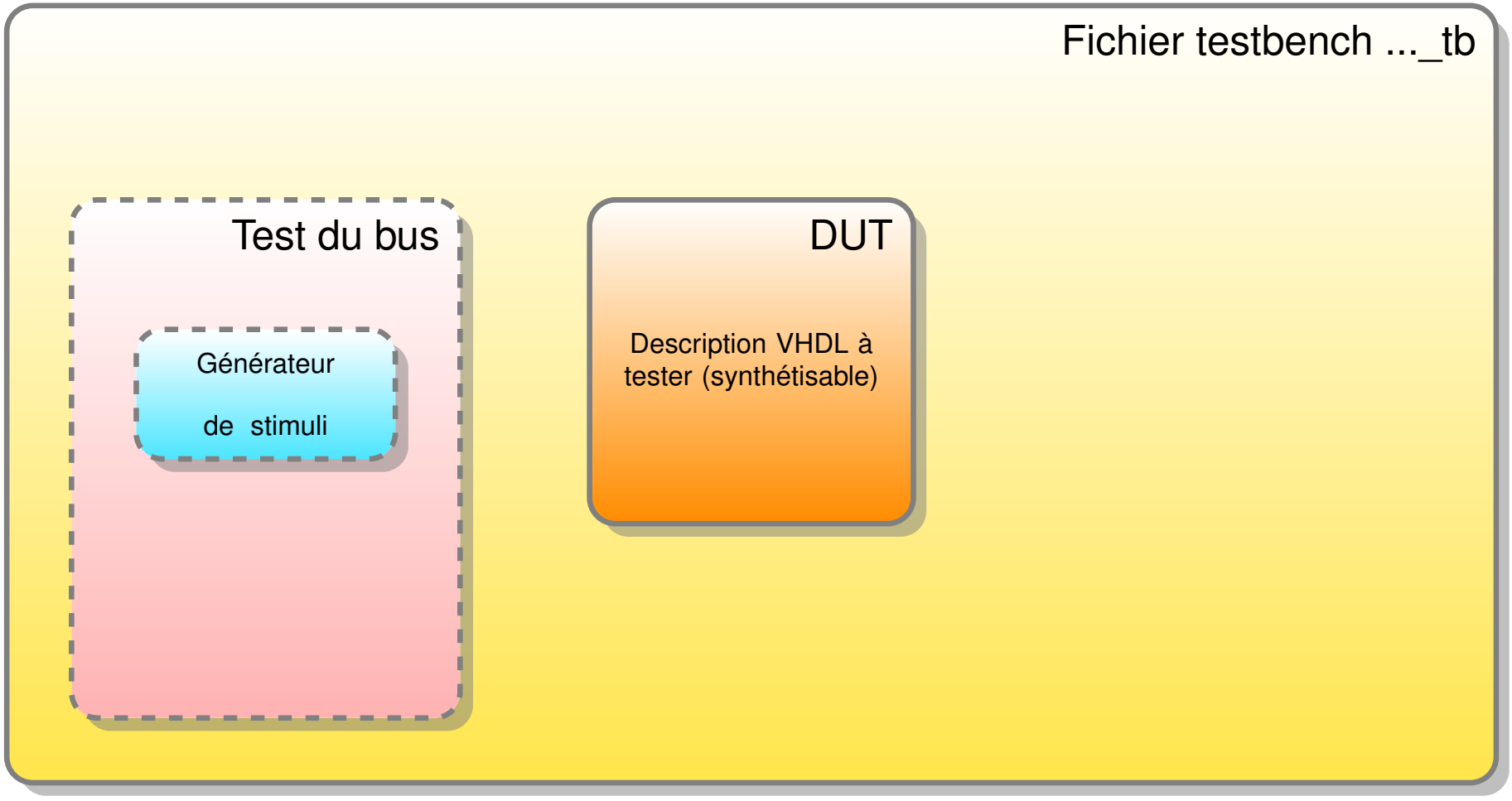
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



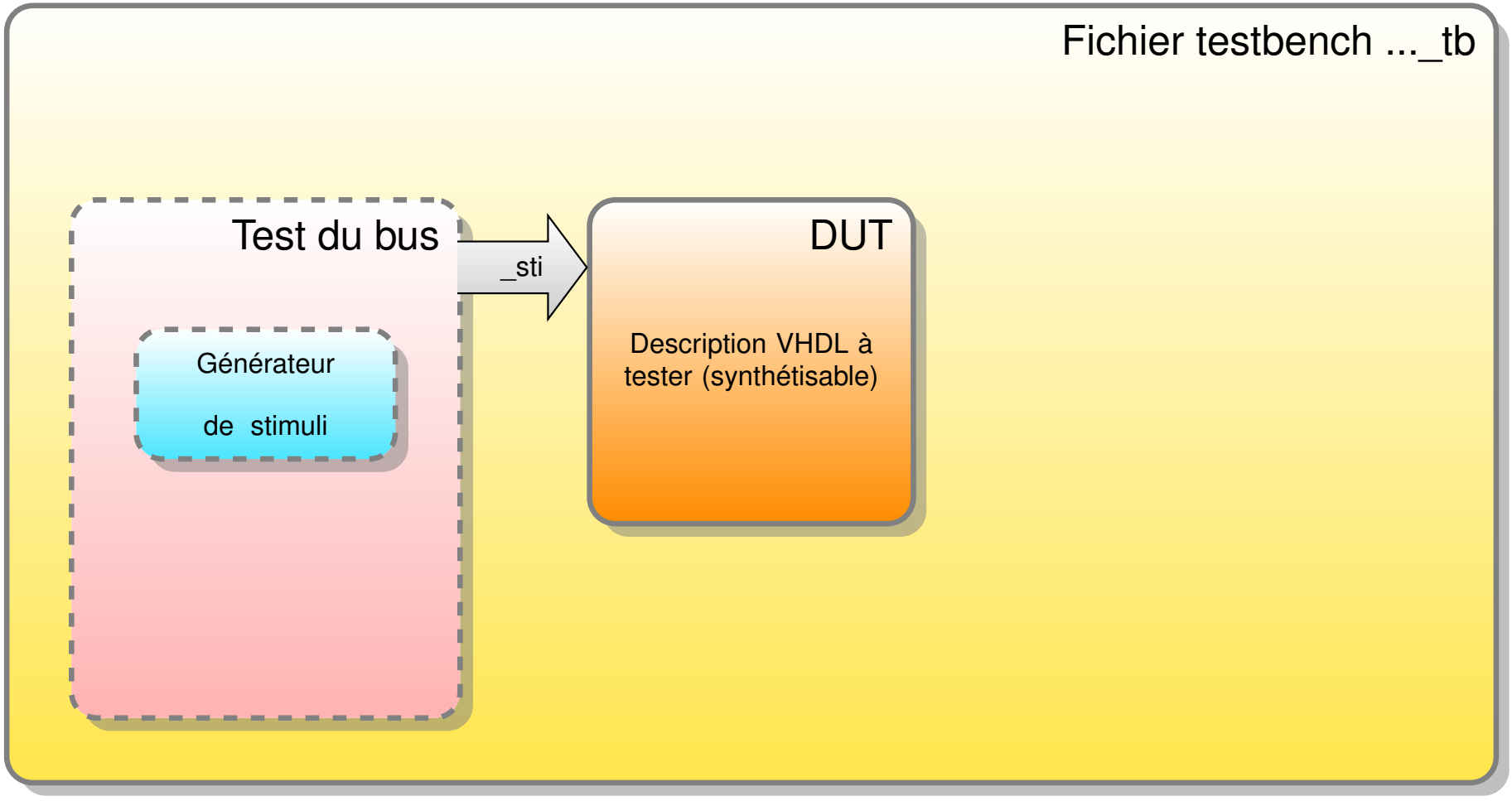
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



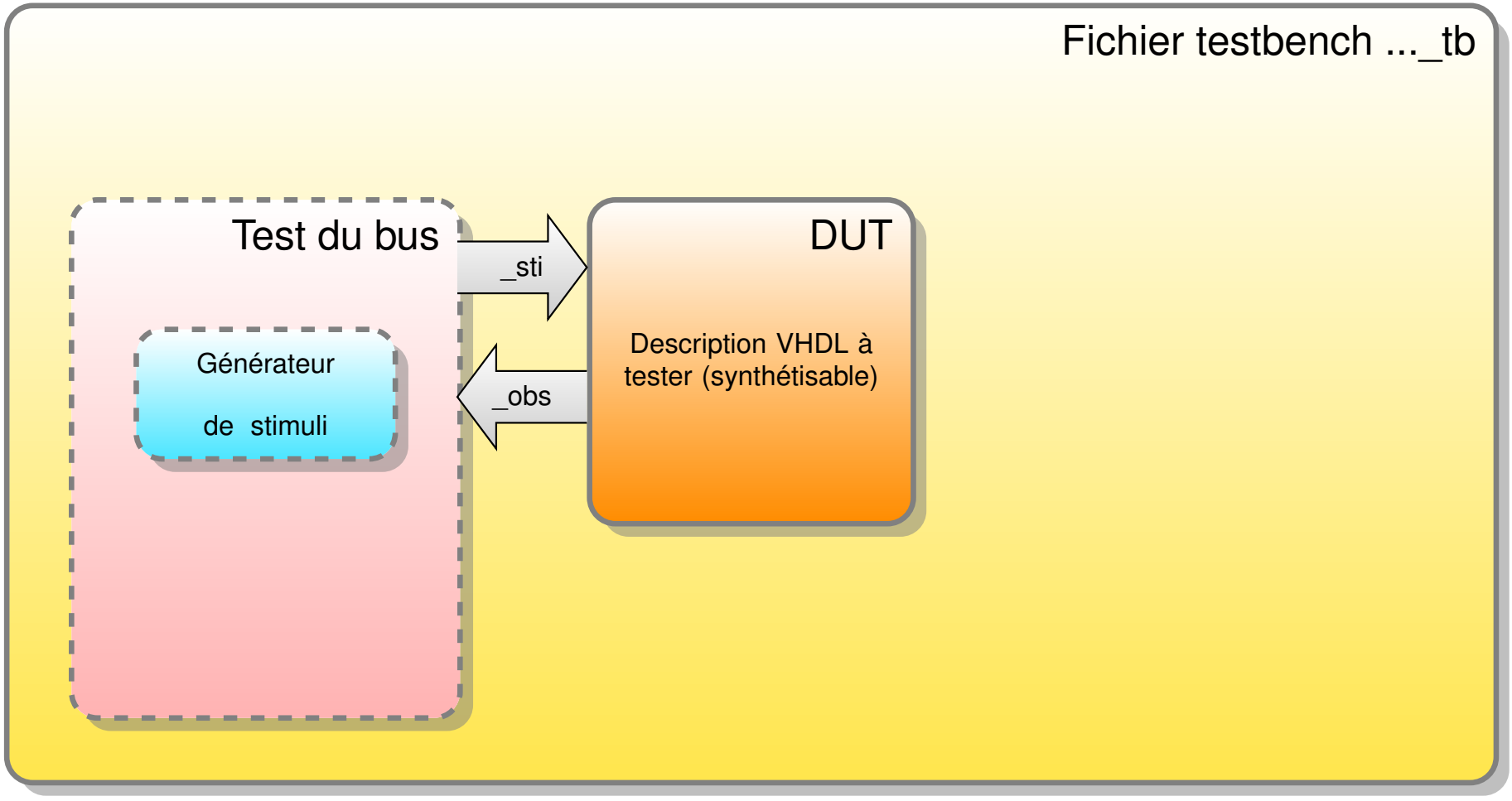
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



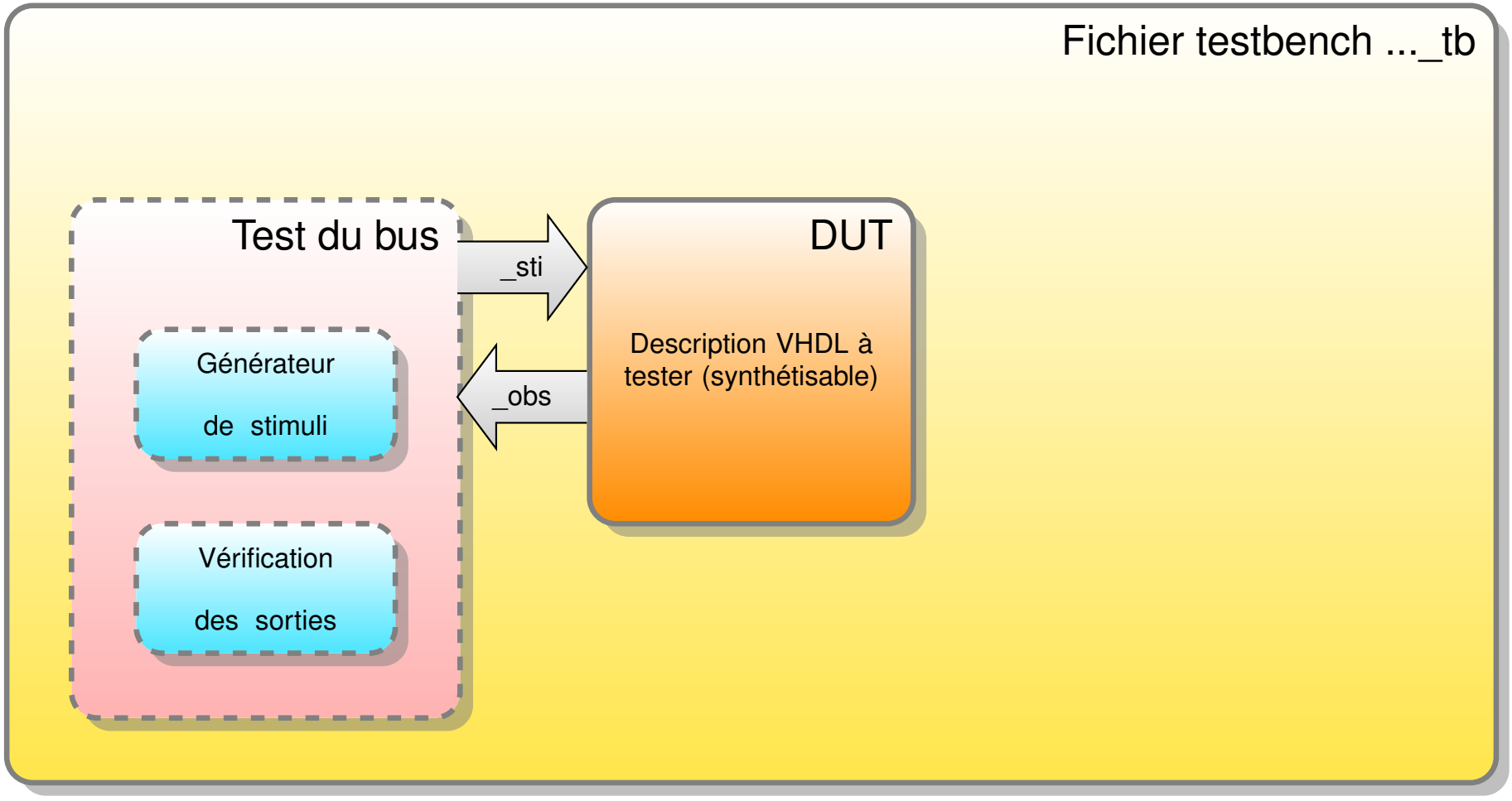
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



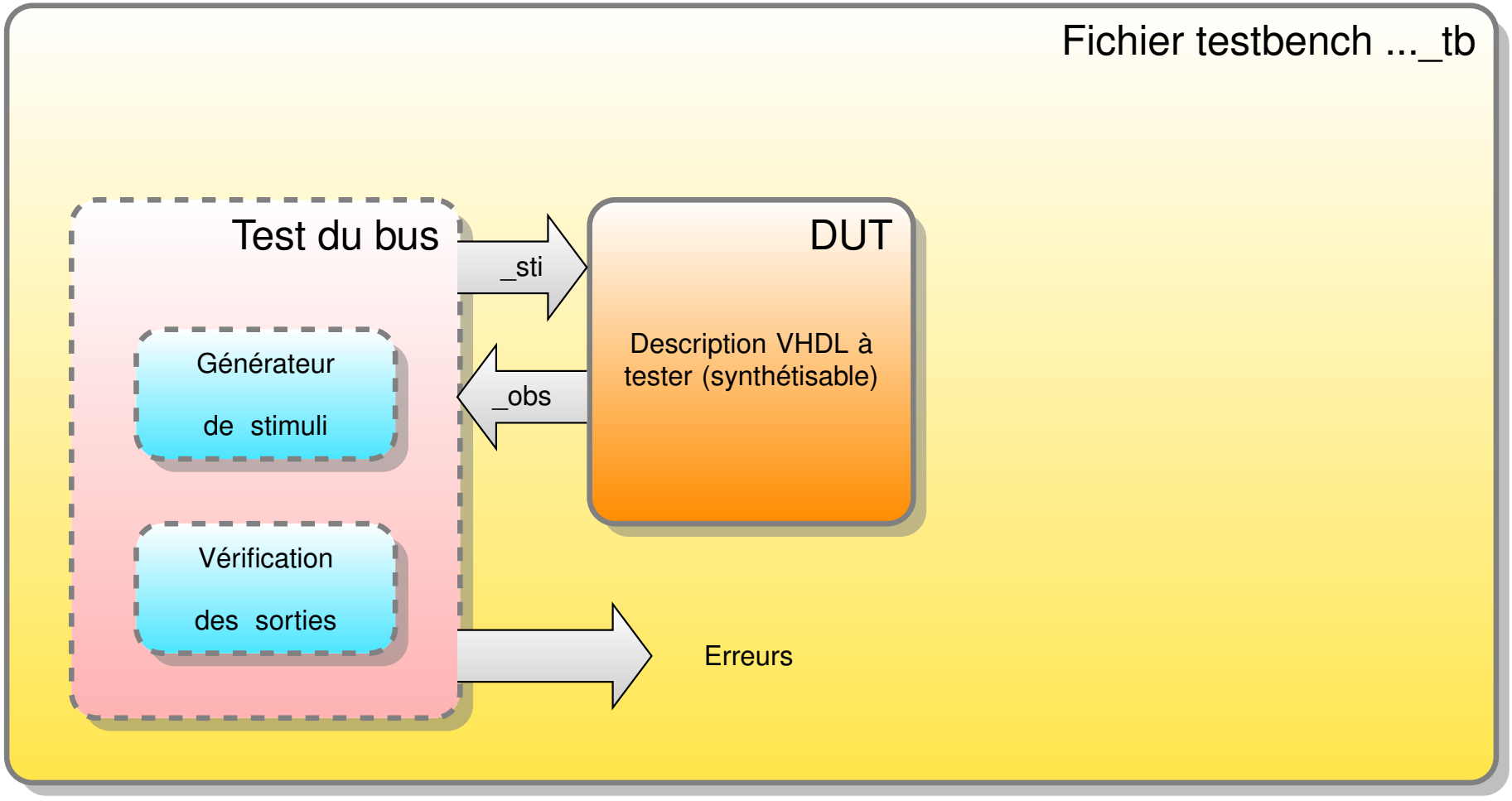
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



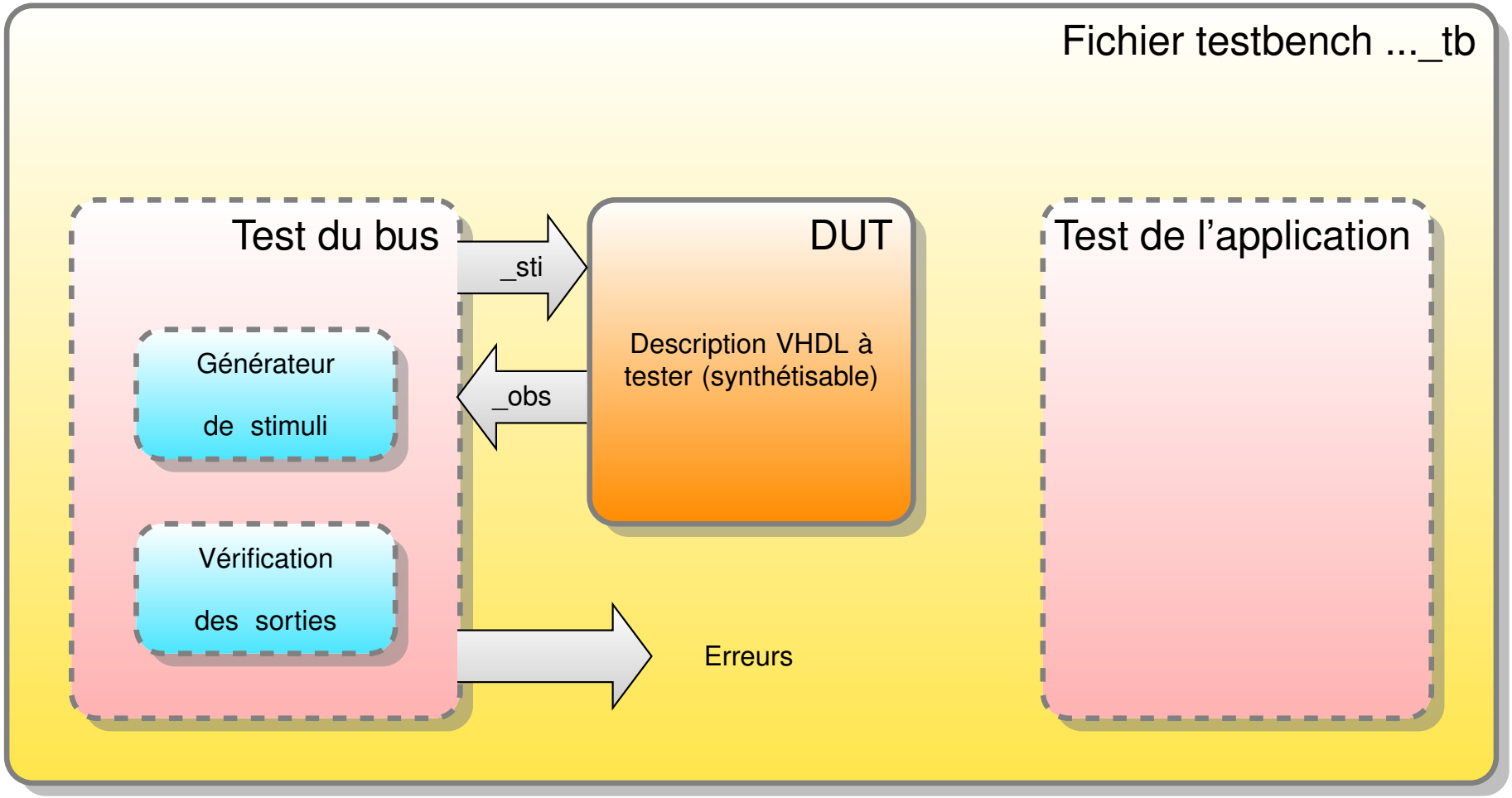
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



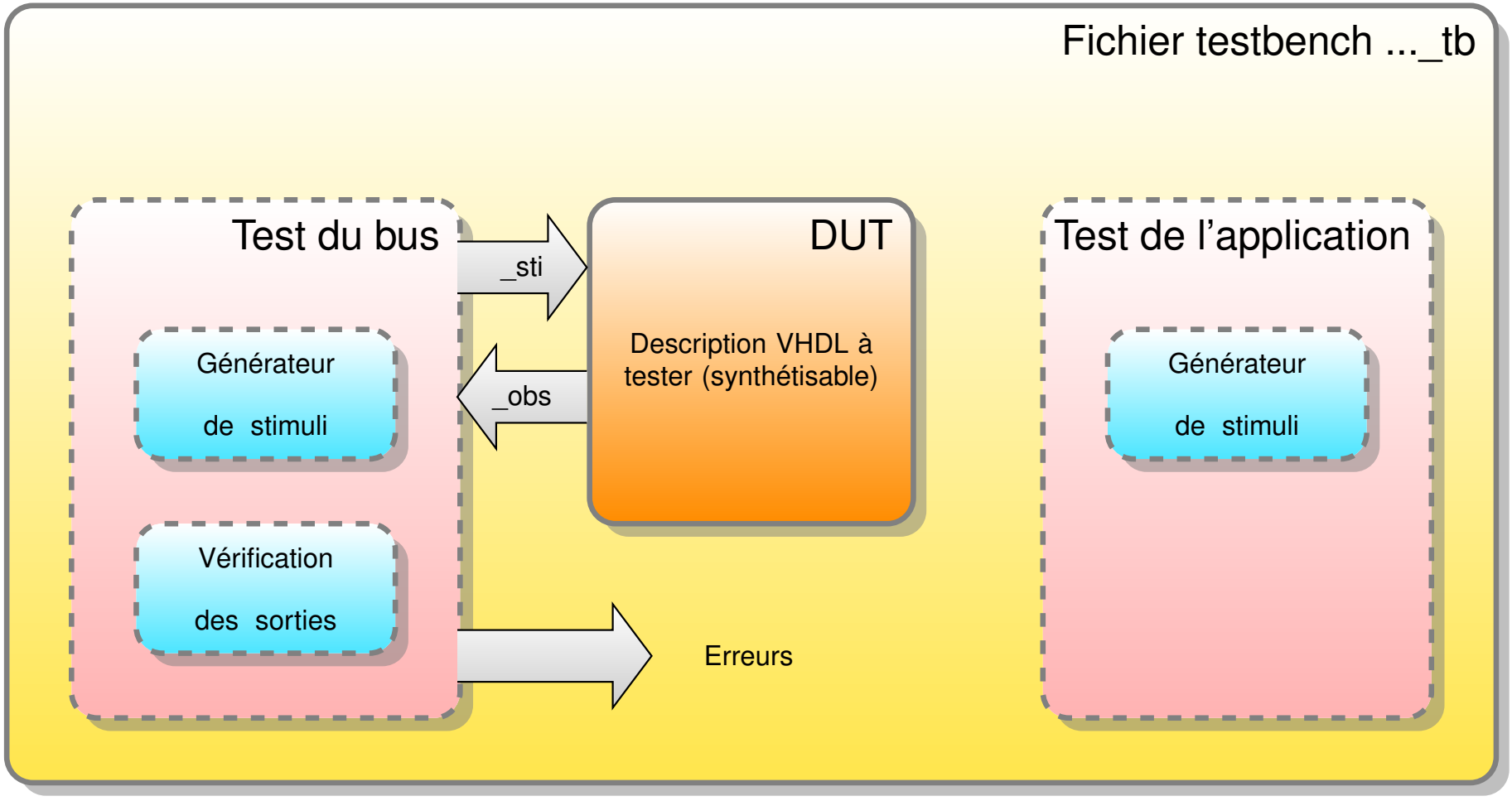
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



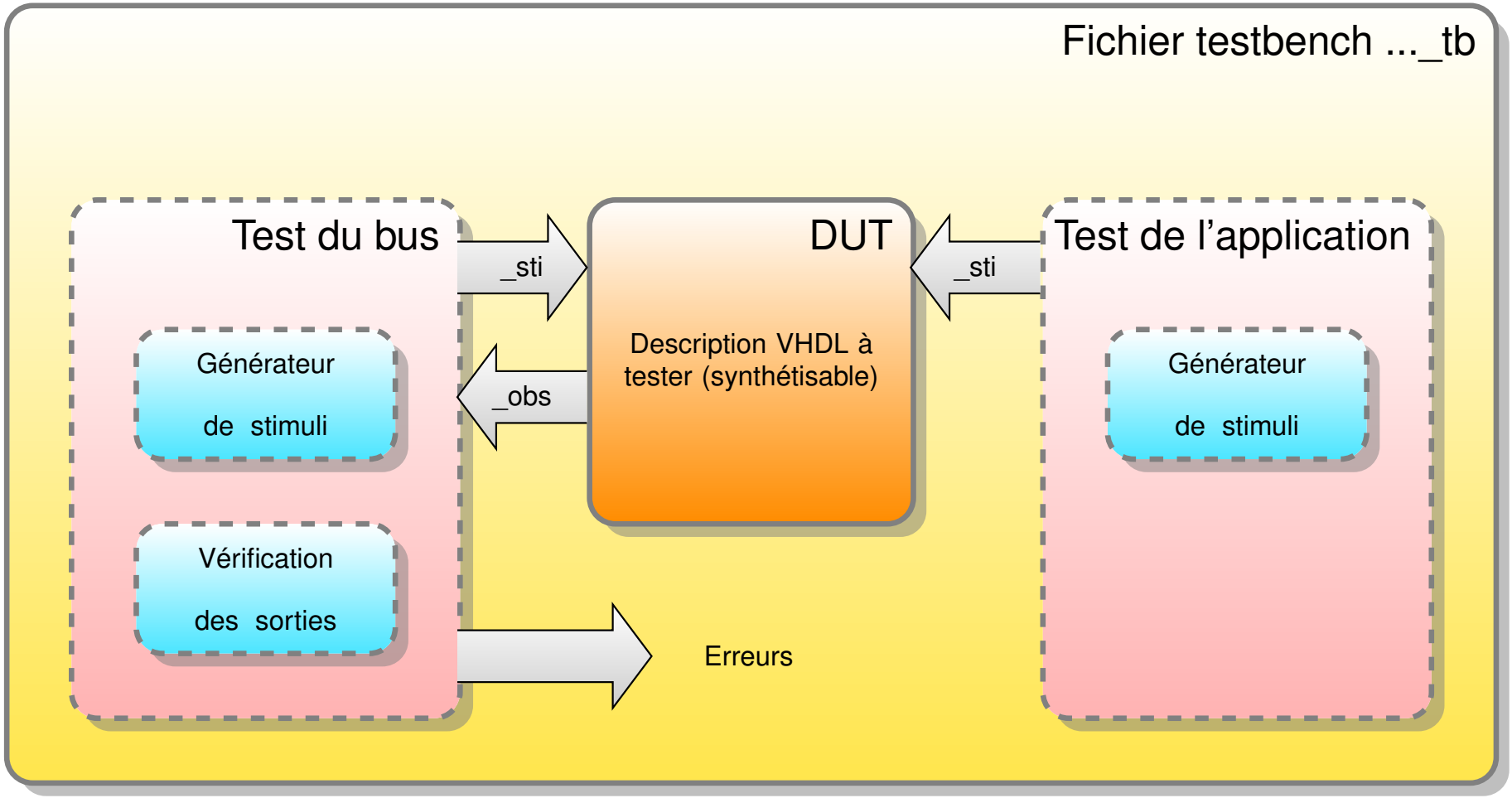
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



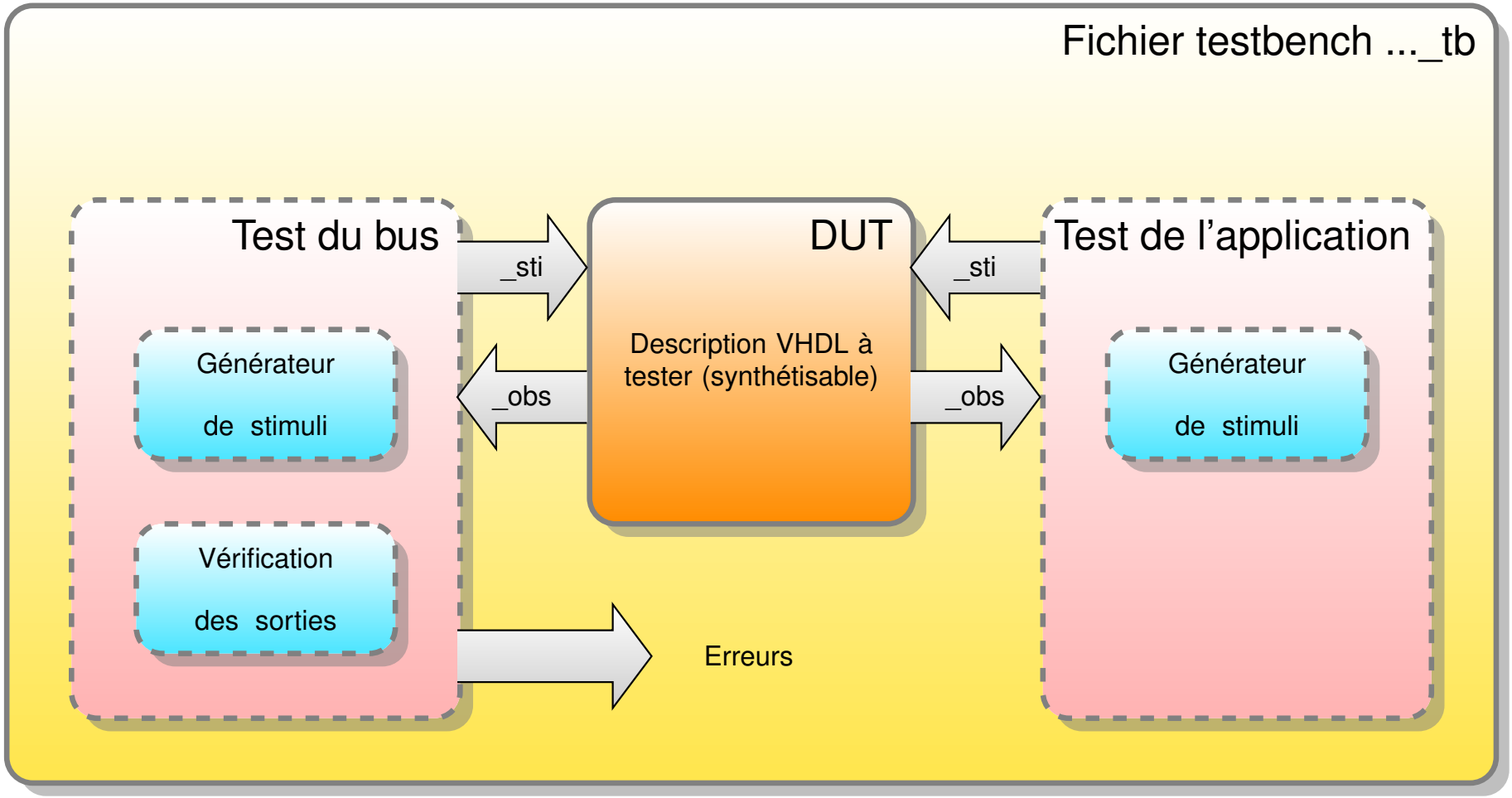
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



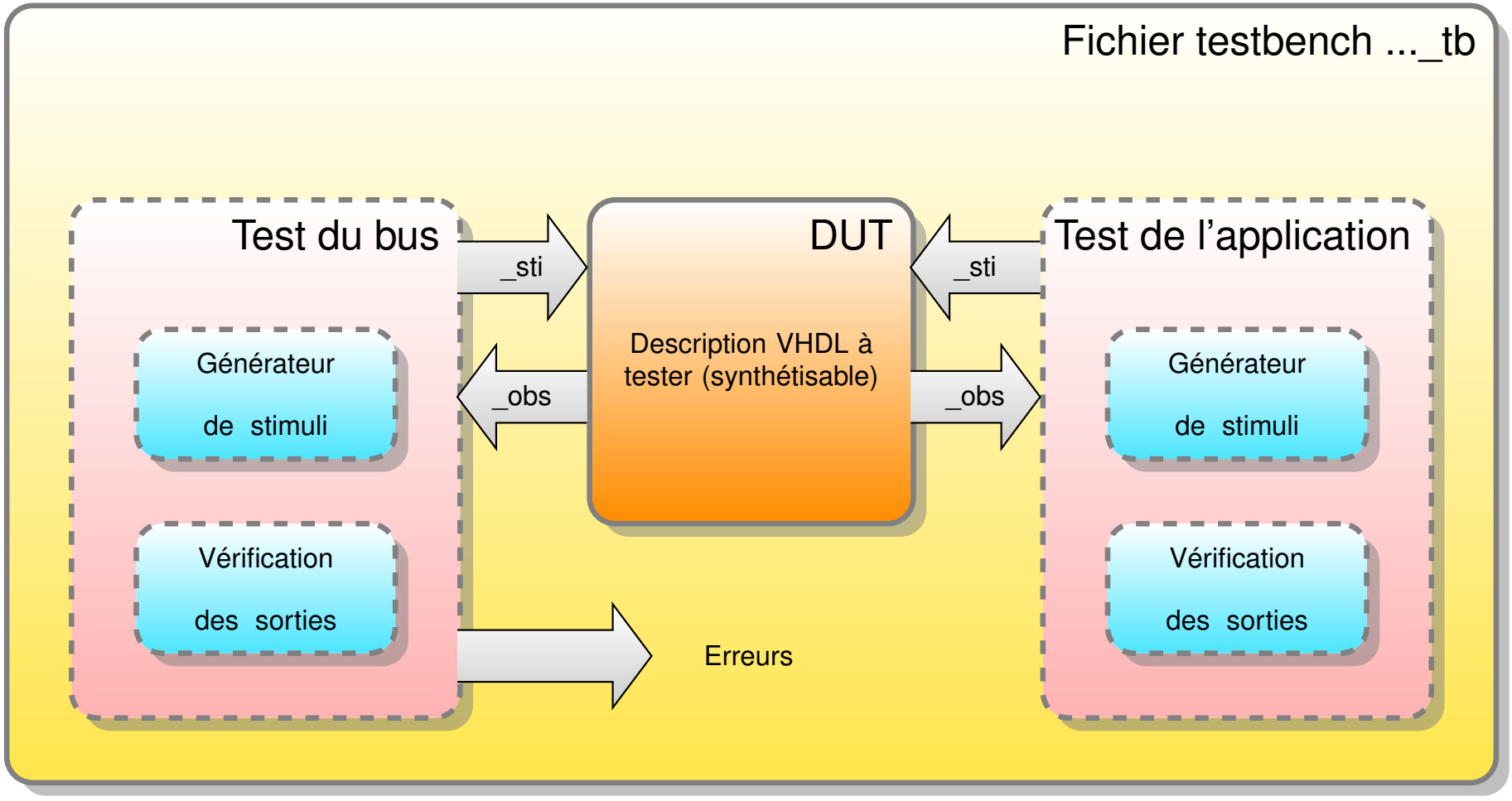
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



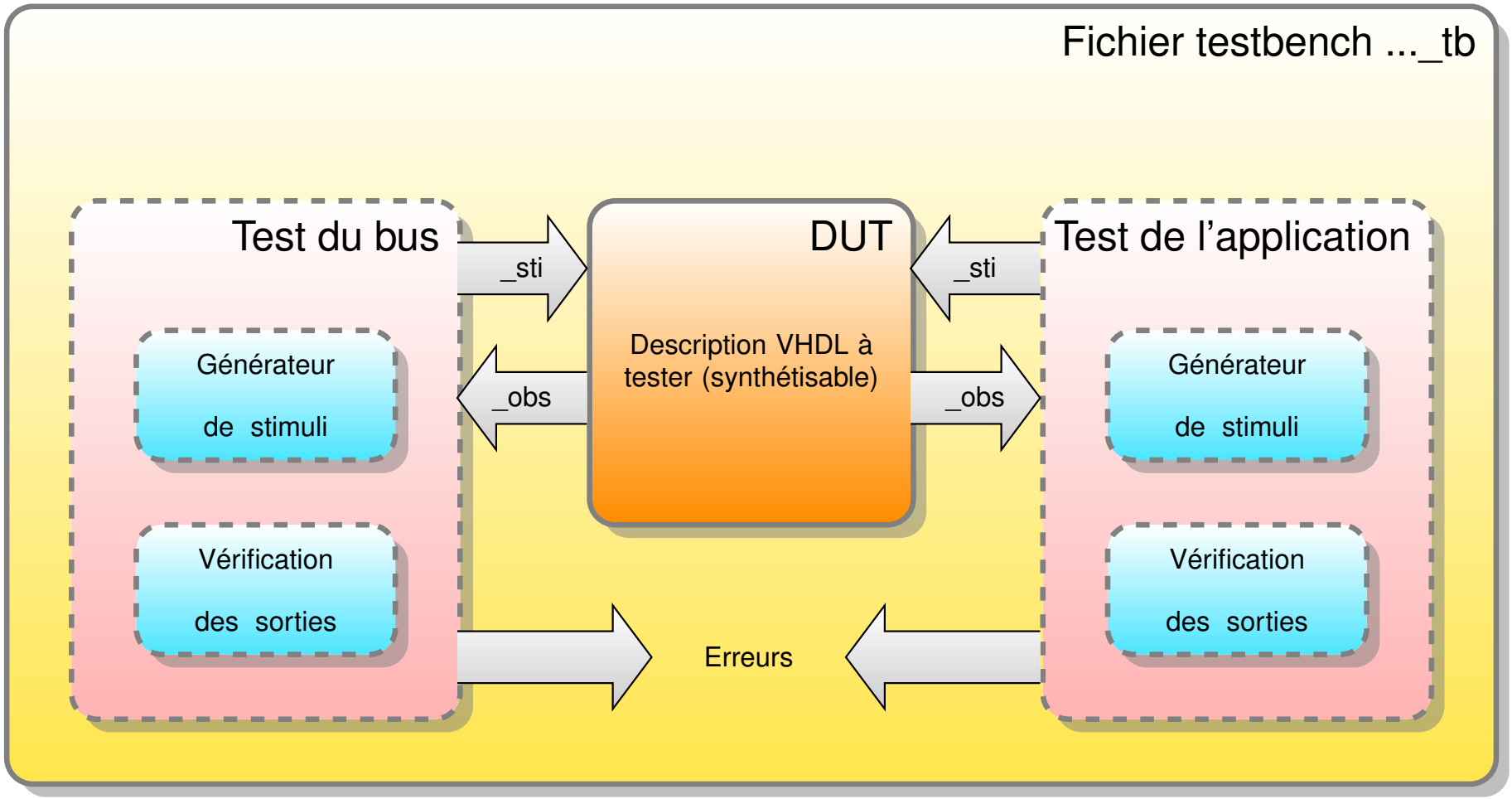
- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

Structure de banc de test



- Solution pour applications utilisant un bus
- Le générateur de stimuli du bus est indépendant de l'application réalisée

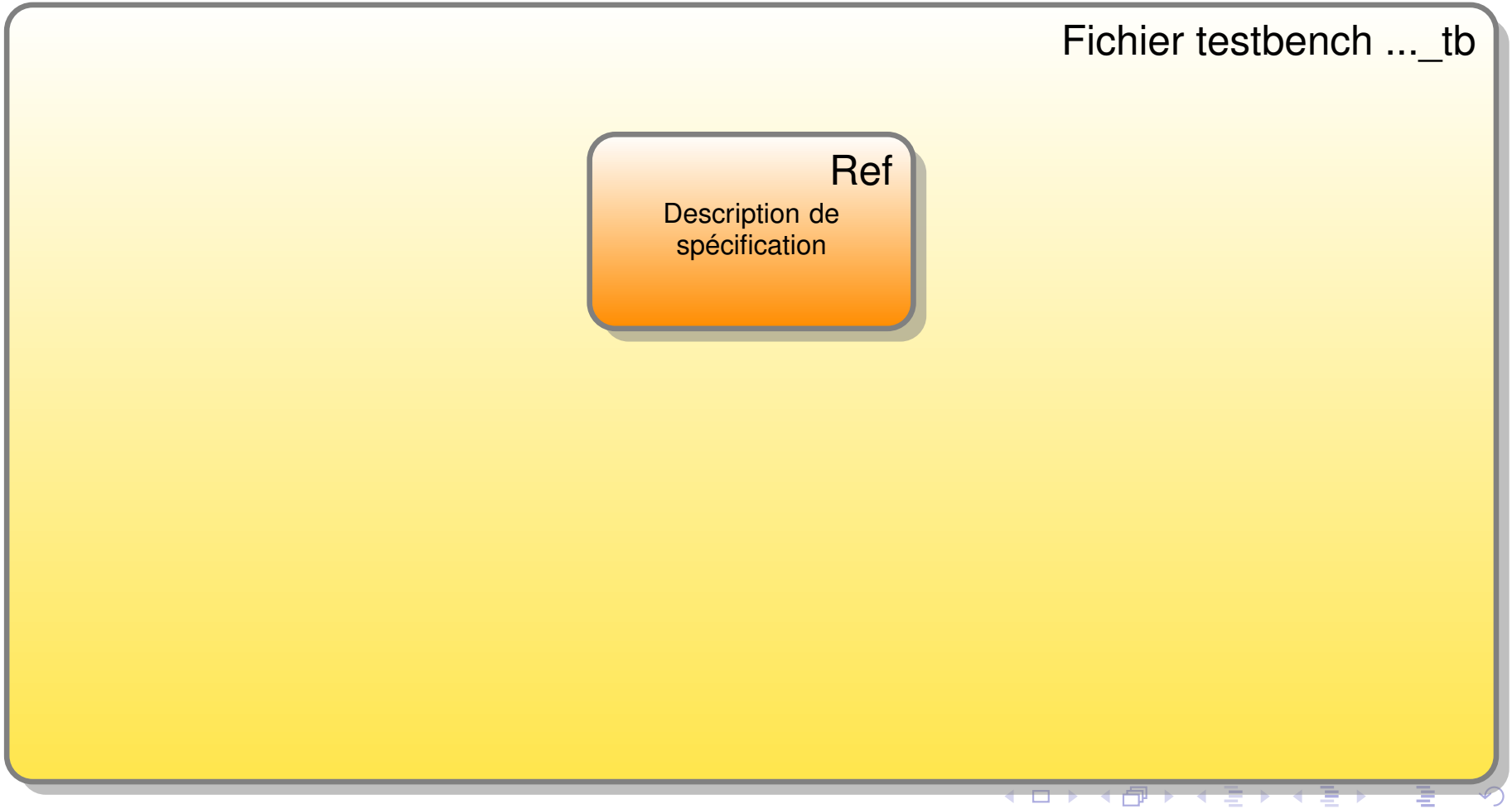
Structure des bancs de test

- Structure avec description de spécification
  - Phase de spécification du projet
  - Permet de valider le fonctionnement du système avec le client

Fichier testbench ..._tb

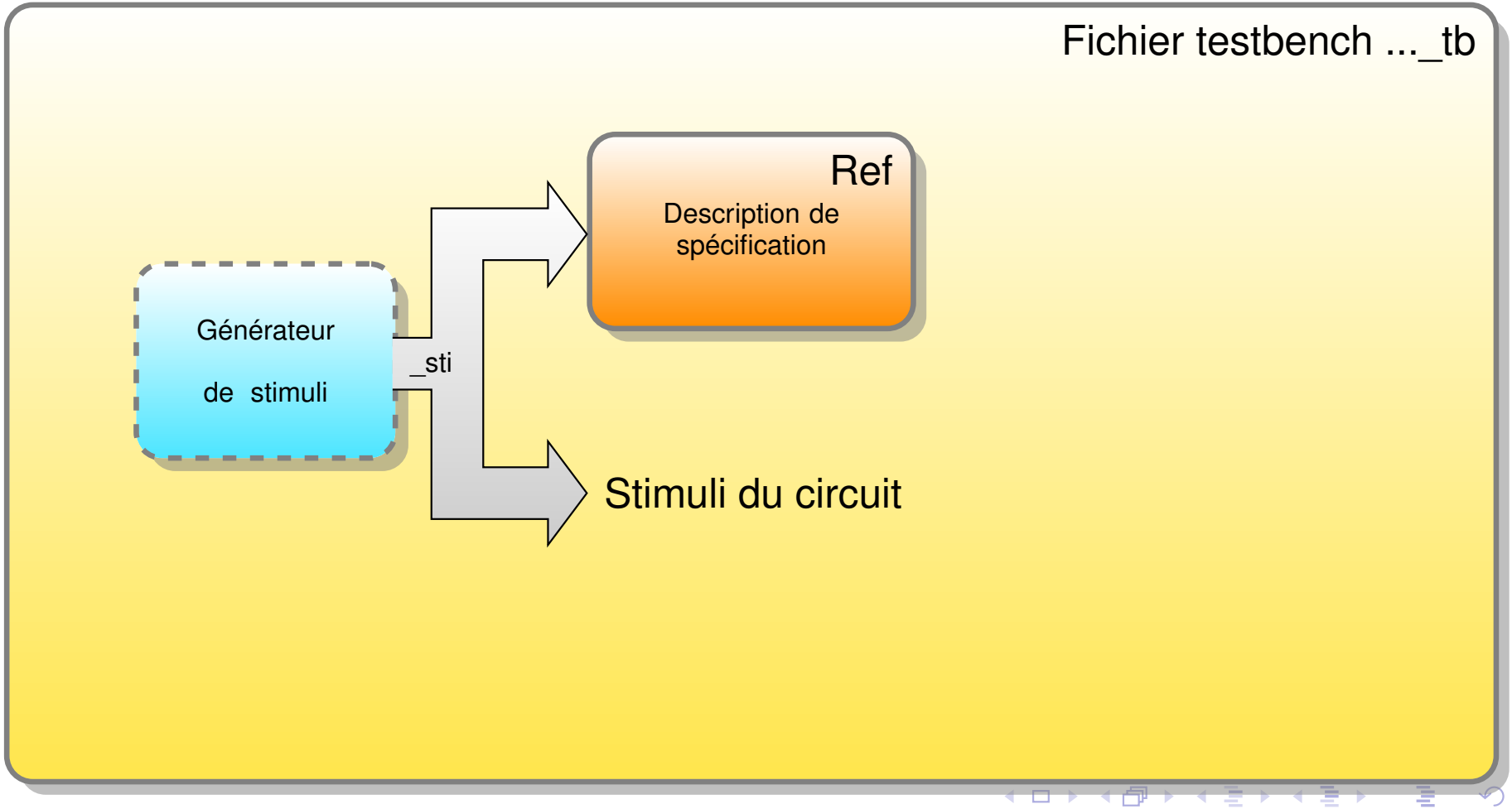
Structure des bancs de test

- Structure avec description de spécification
  - Phase de spécification du projet
  - Permet de valider le fonctionnement du système avec le client



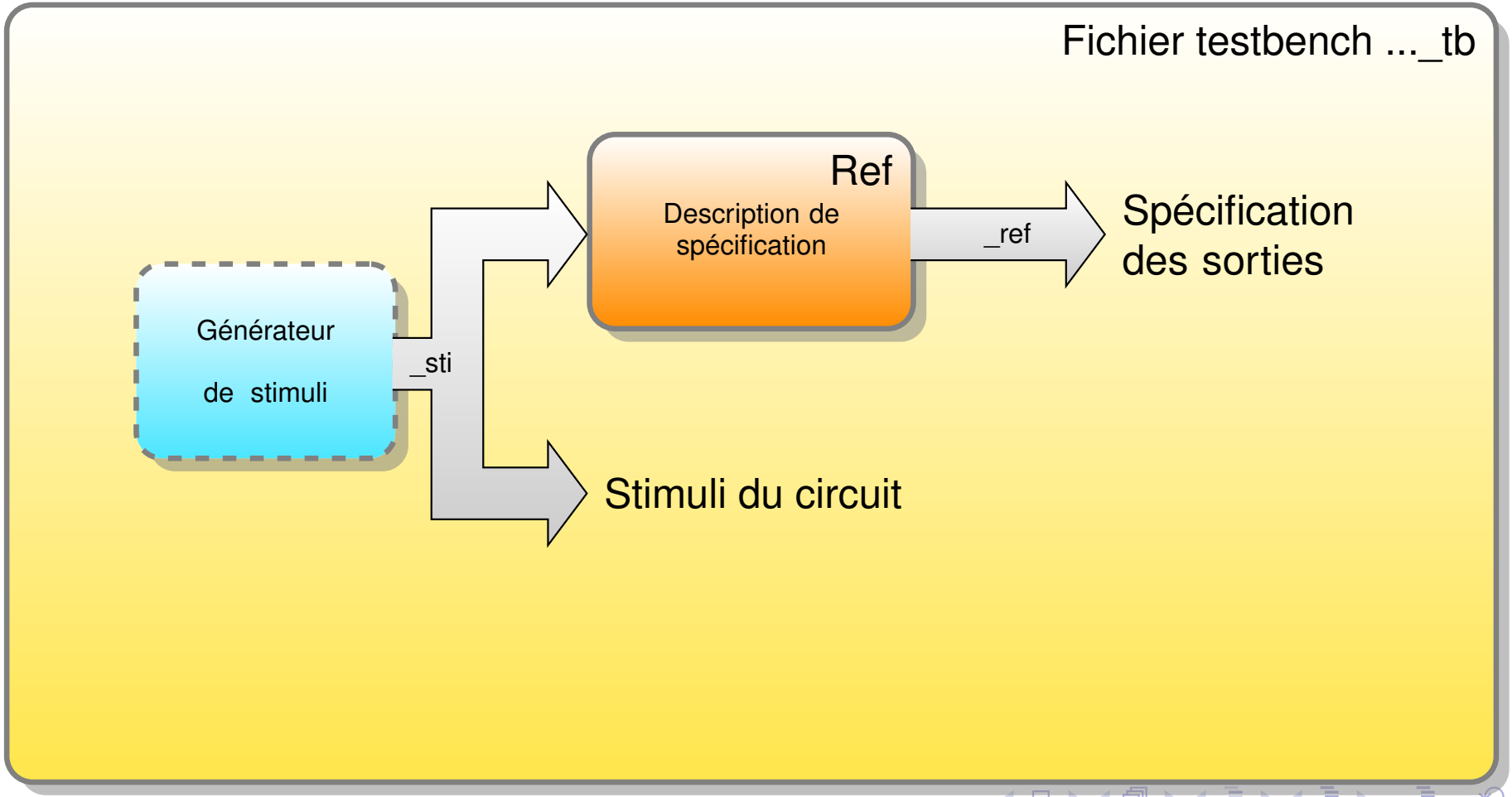
Structure des bancs de test

- Structure avec description de spécification
  - Phase de spécification du projet
  - Permet de valider le fonctionnement du système avec le client



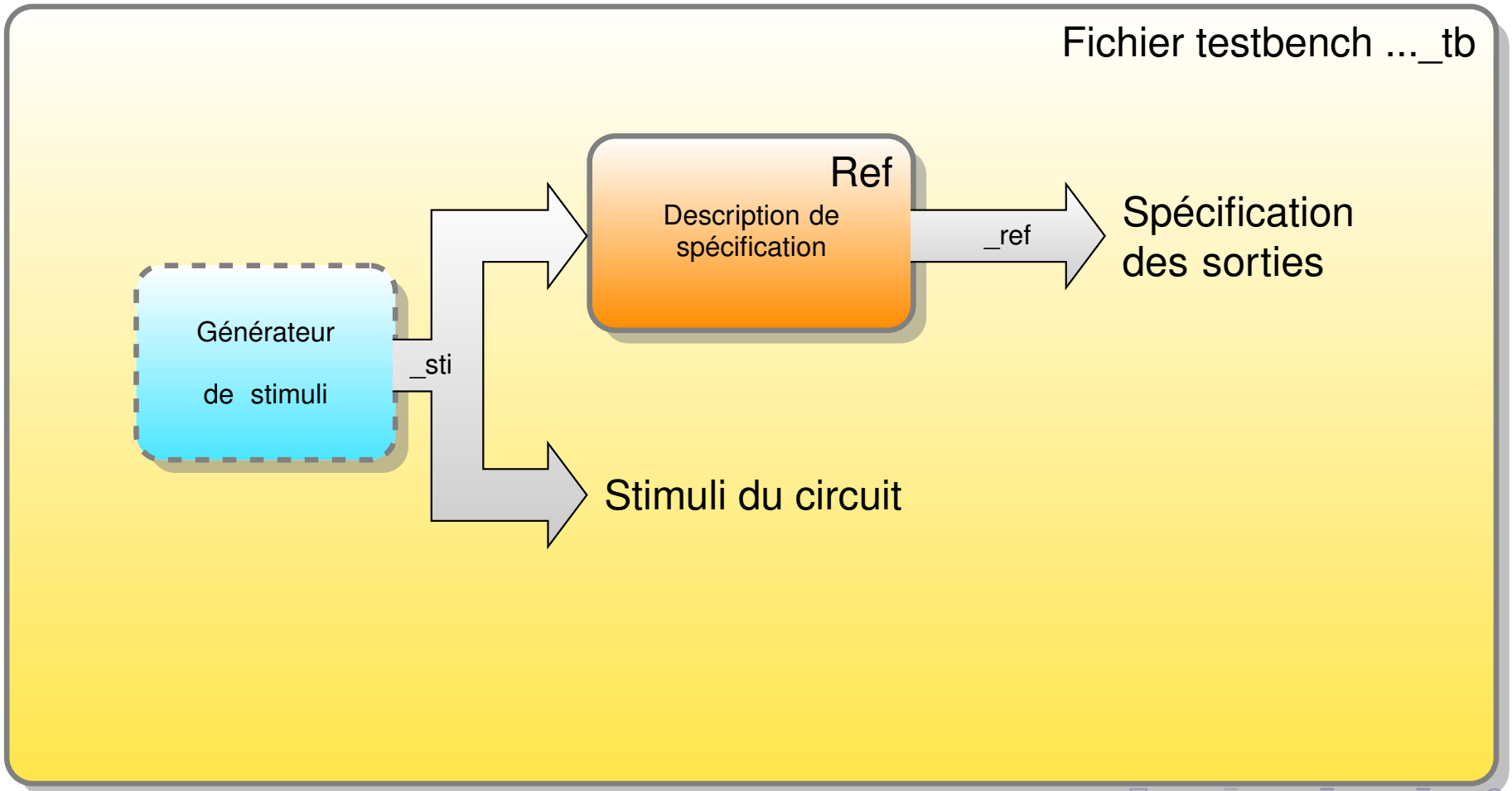
Structure des bancs de test

- Structure avec description de spécification
  - Phase de spécification du projet
  - Permet de valider le fonctionnement du système avec le client



Structure des bancs de test

- Structure avec description de spécification
  - Phase de spécification du projet
  - Permet de valider le fonctionnement du système avec le client



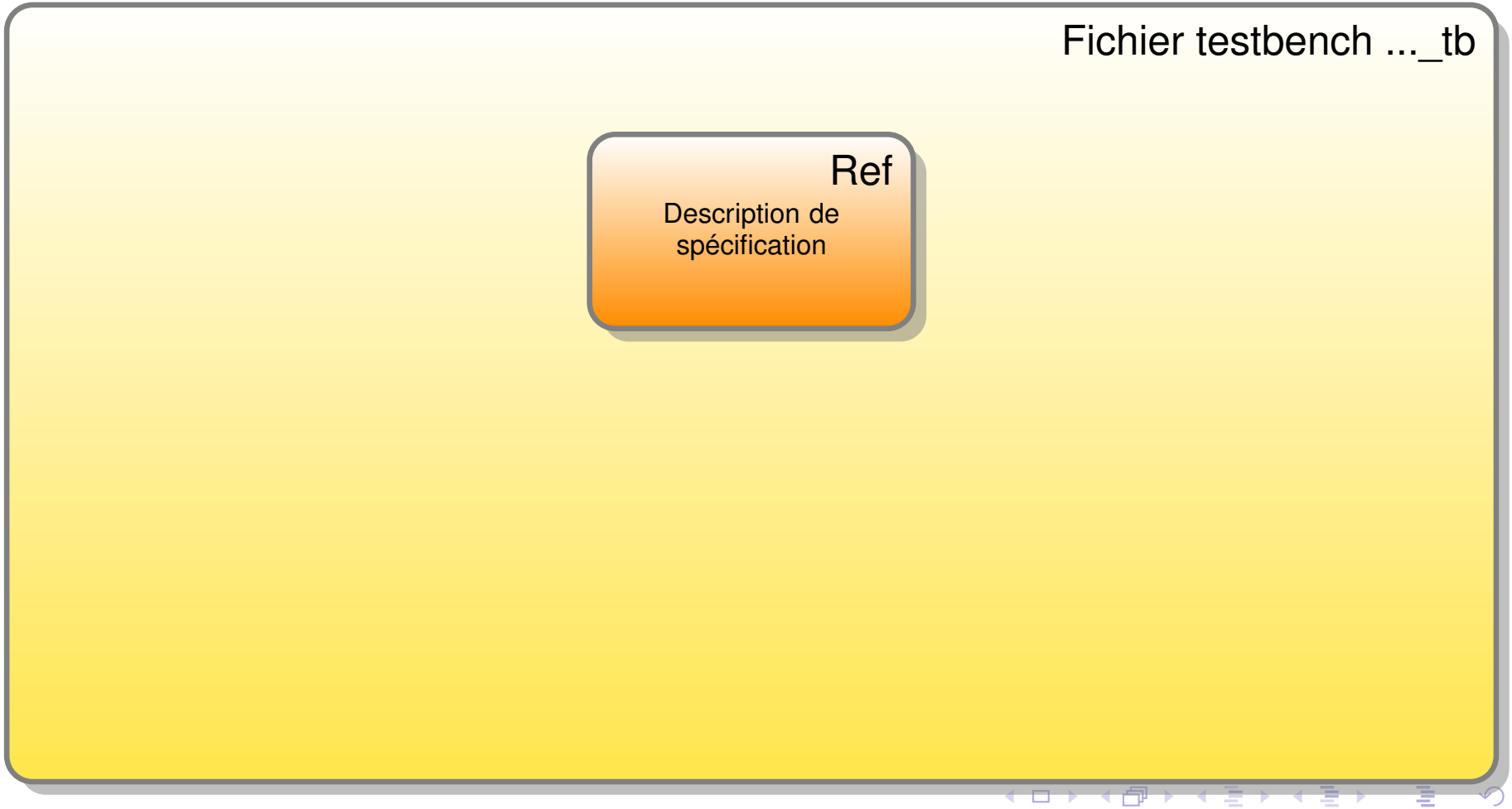
Structure des bancs de test

- Structure avec description de spécification
  - Phase de réalisation du projet
  - Permet de valider le fonctionnement du système

Fichier testbench ..._tb

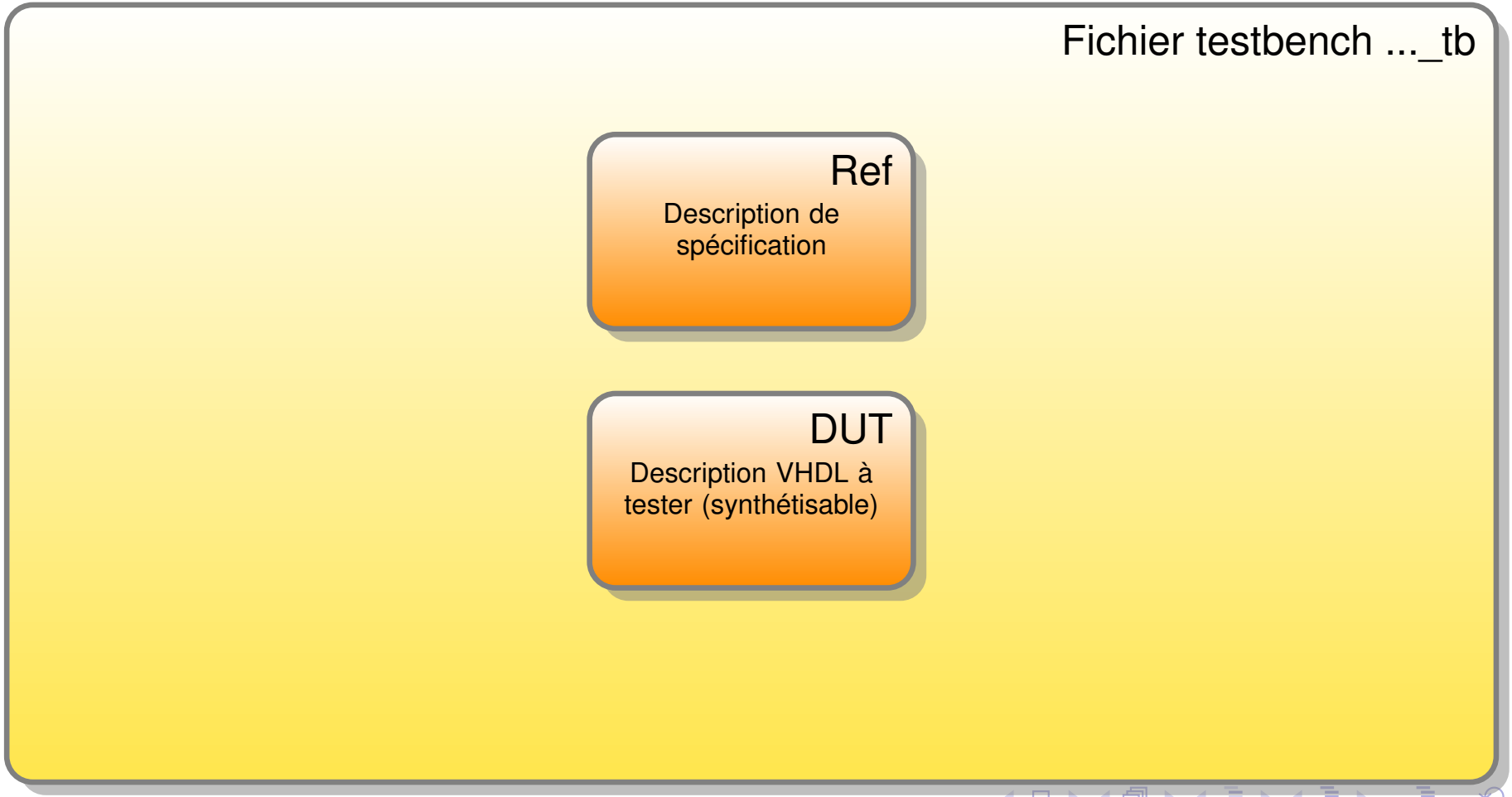
Structure des bancs de test

- Structure avec description de spécification
  - Phase de réalisation du projet
  - Permet de valider le fonctionnement du système



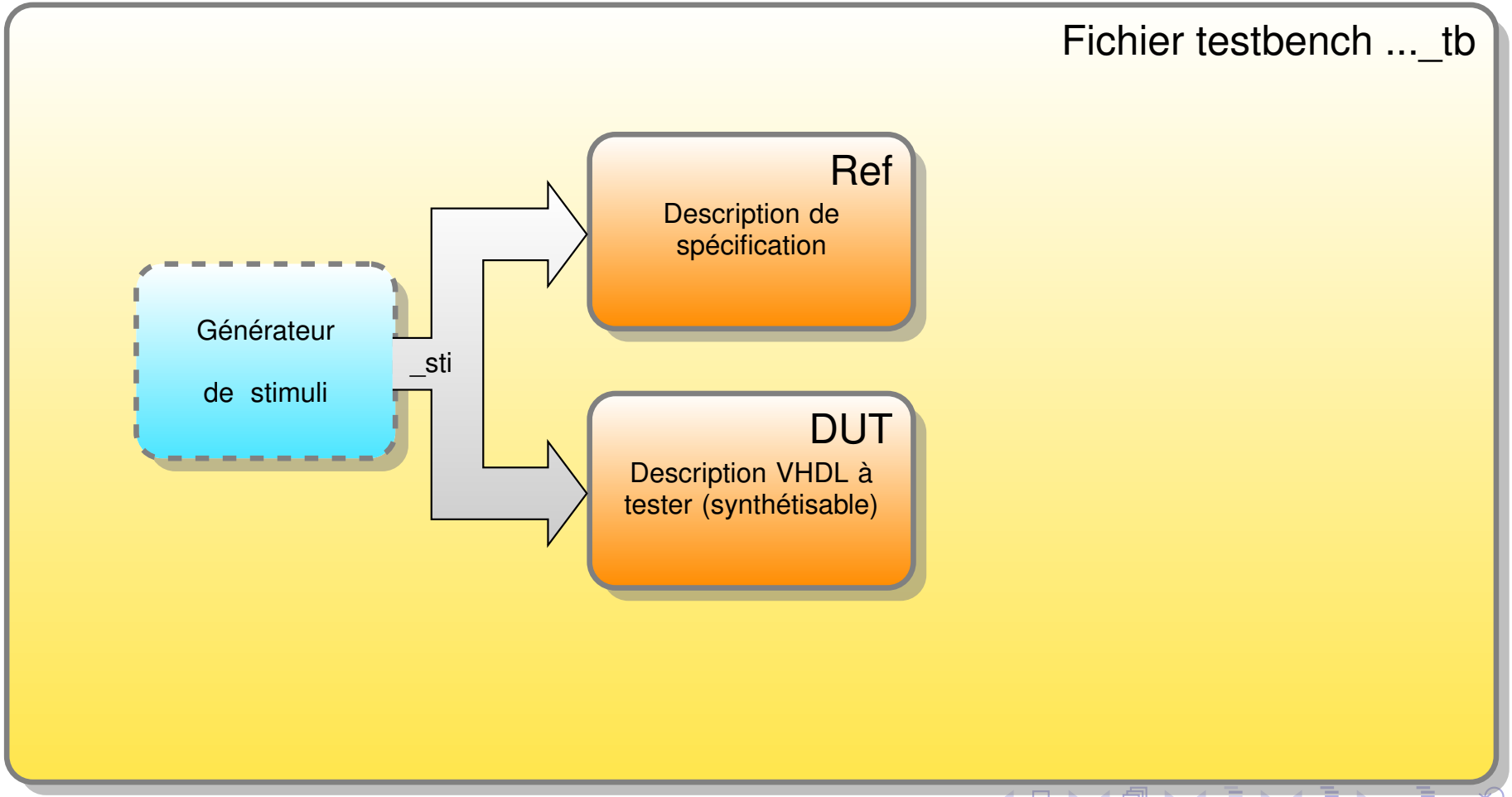
Structure des bancs de test

- Structure avec description de spécification
  - Phase de réalisation du projet
  - Permet de valider le fonctionnement du système



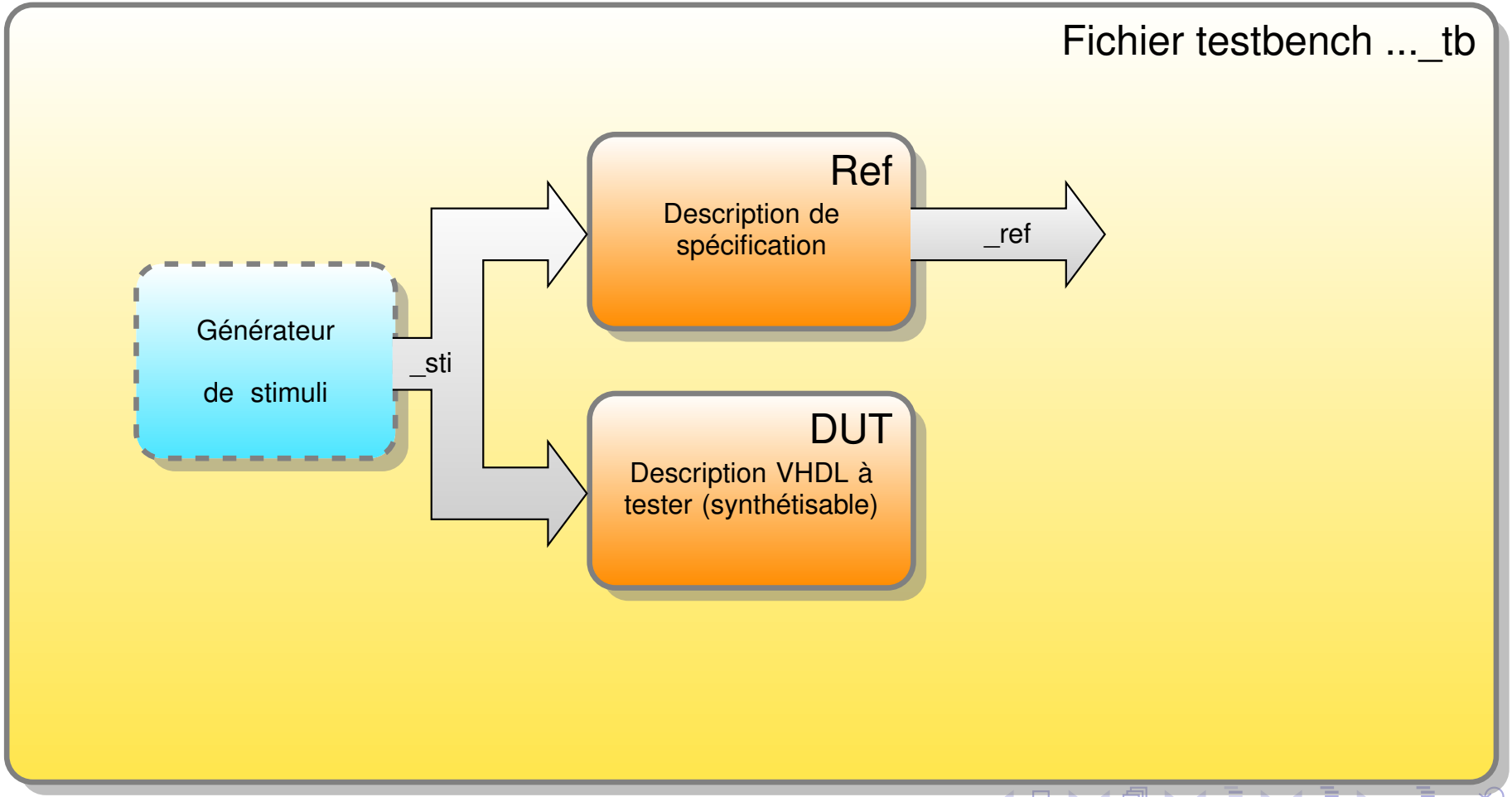
Structure des bancs de test

- Structure avec description de spécification
  - Phase de réalisation du projet
  - Permet de valider le fonctionnement du système



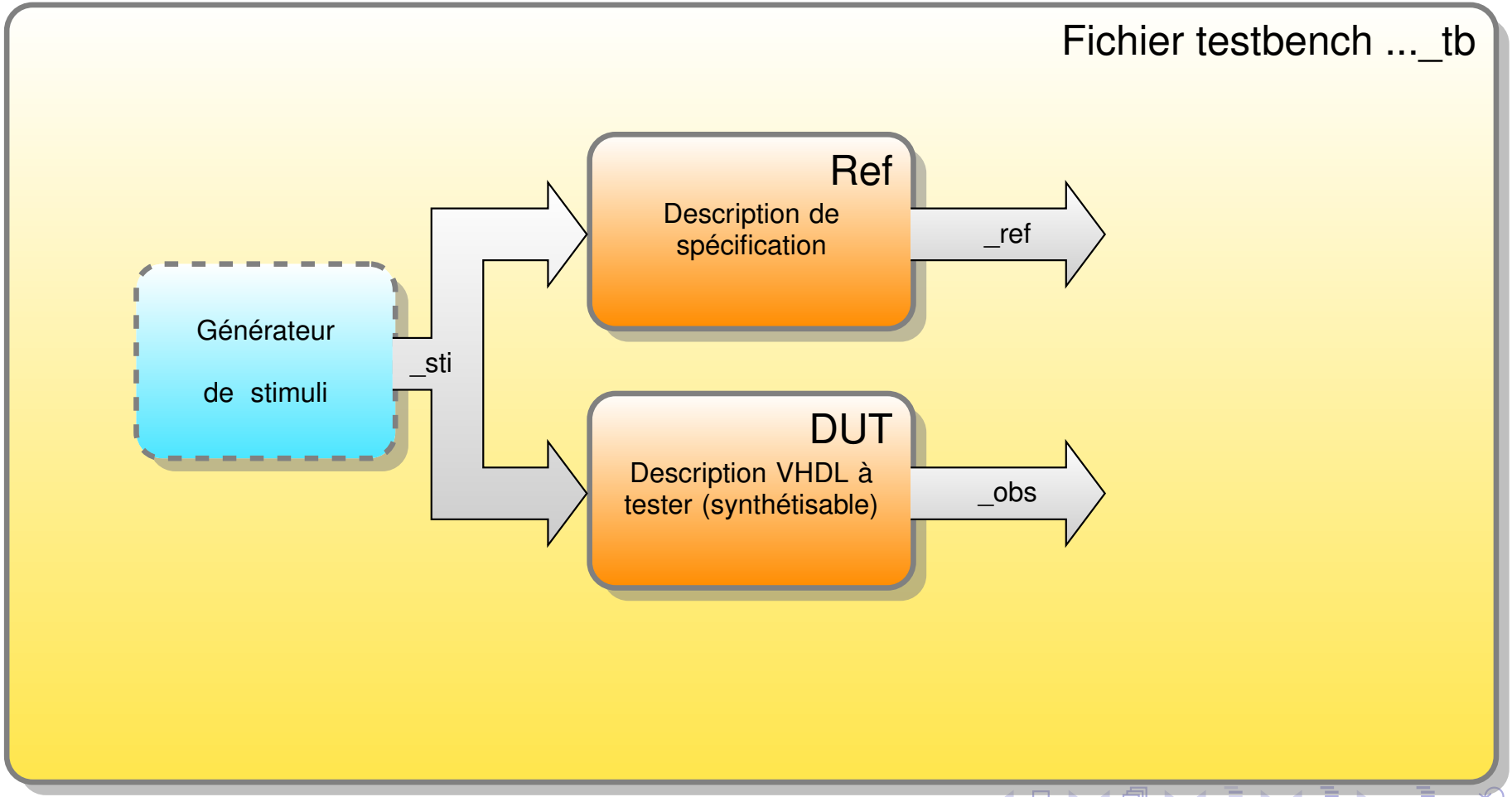
Structure des bancs de test

- Structure avec description de spécification
  - Phase de réalisation du projet
  - Permet de valider le fonctionnement du système



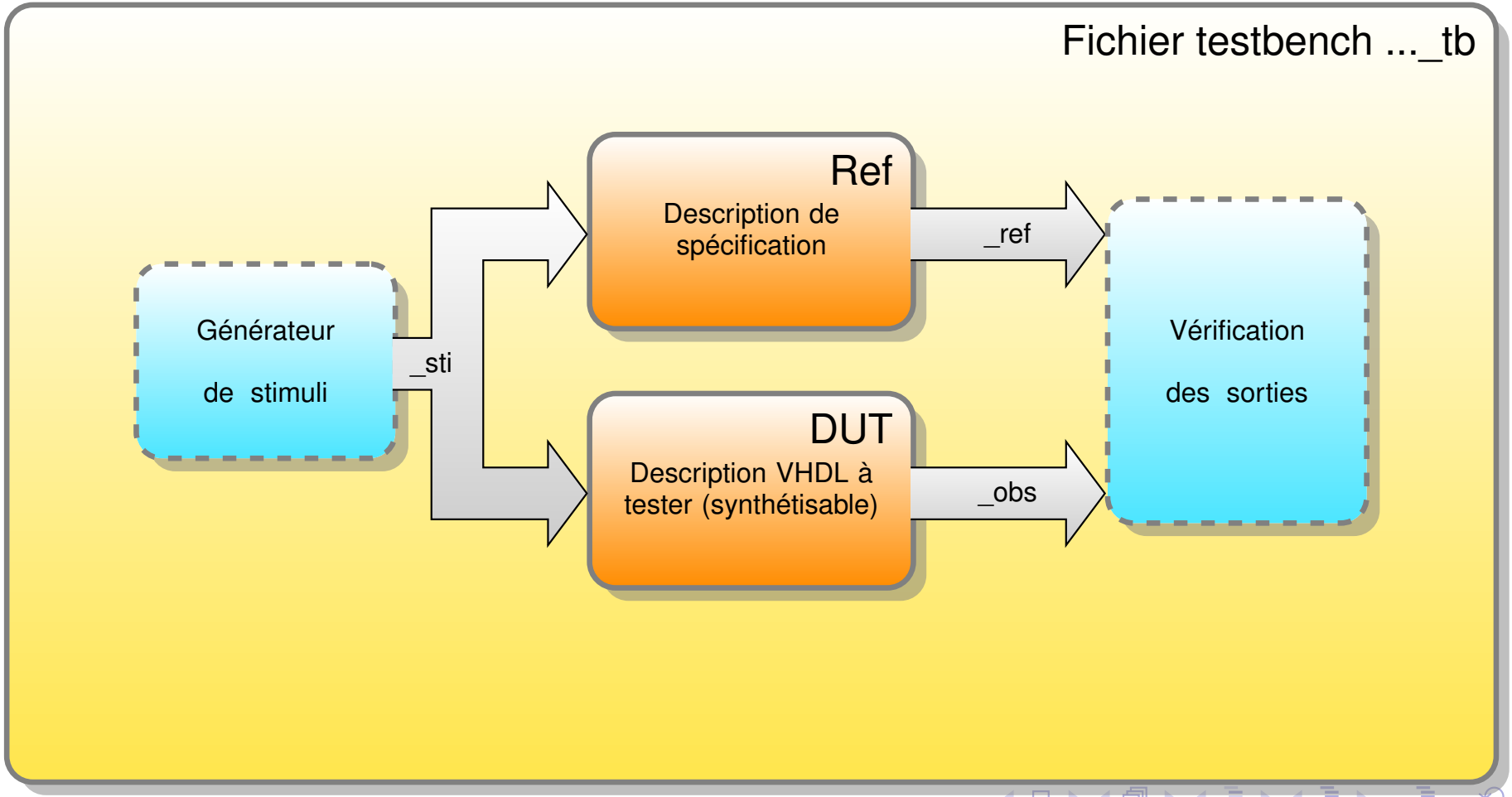
Structure des bancs de test

- Structure avec description de spécification
  - Phase de réalisation du projet
  - Permet de valider le fonctionnement du système



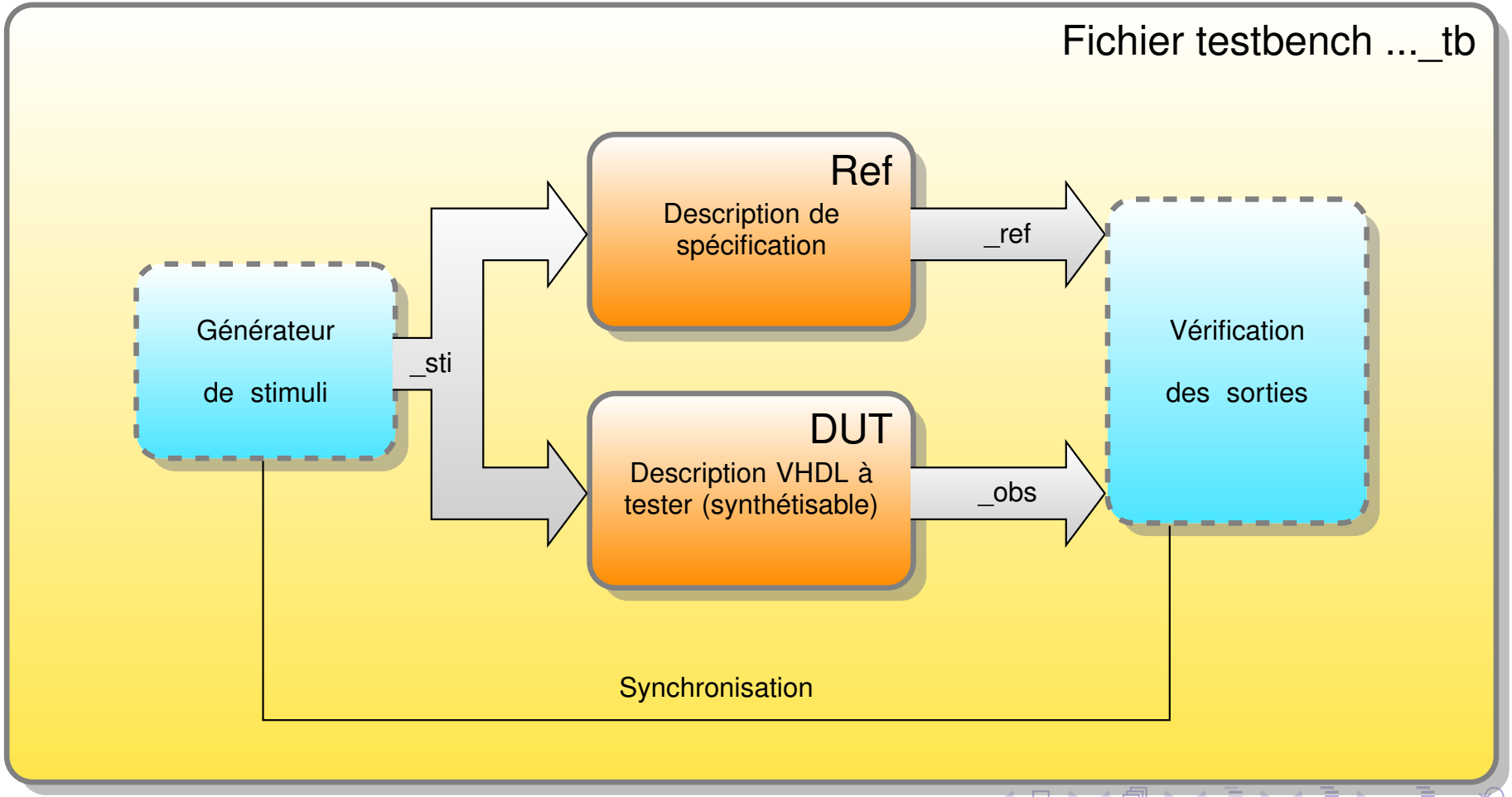
Structure des bancs de test

- Structure avec description de spécification
  - Phase de réalisation du projet
  - Permet de valider le fonctionnement du système



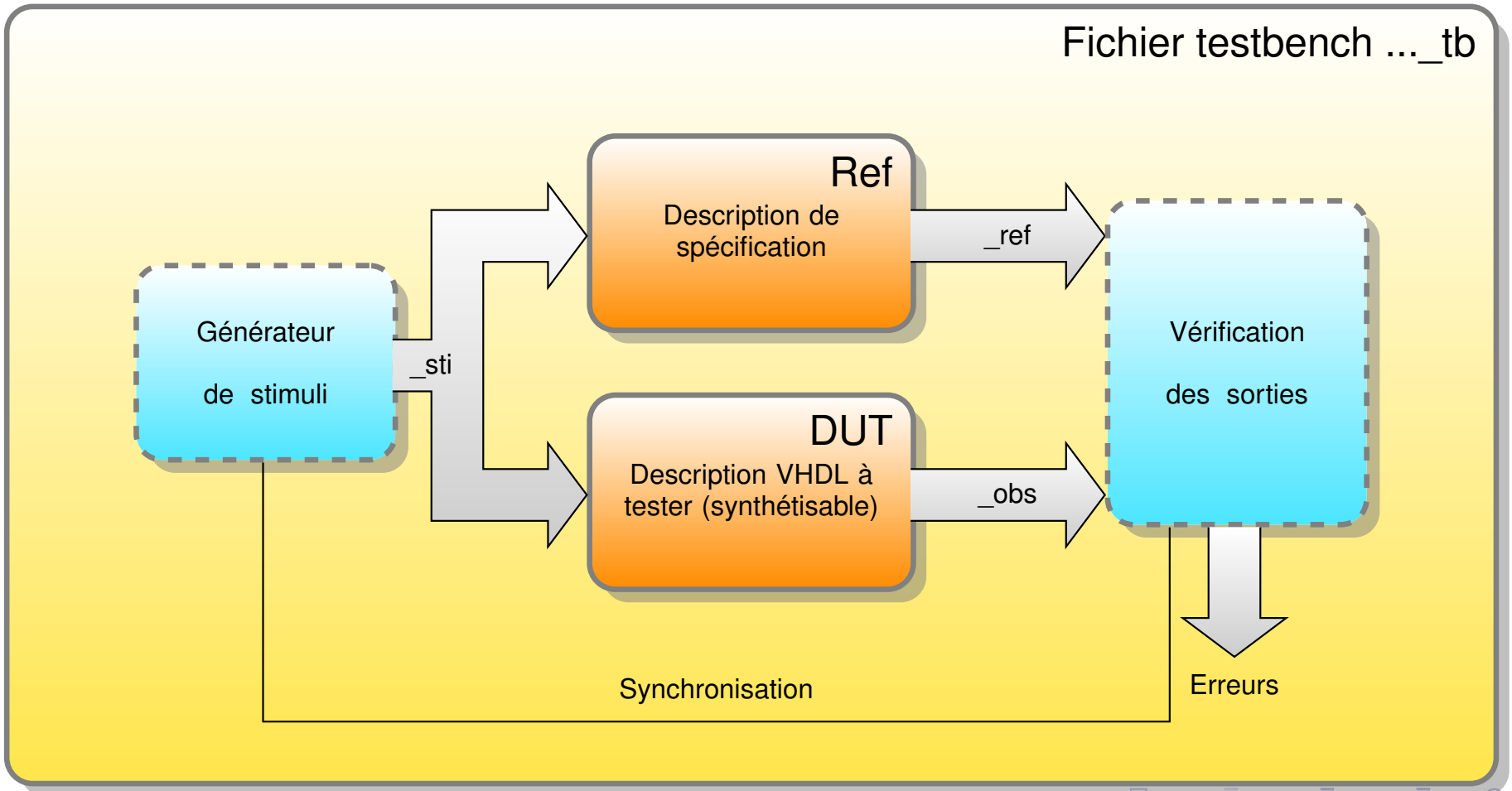
Structure des bancs de test

- Structure avec description de spécification
  - Phase de réalisation du projet
  - Permet de valider le fonctionnement du système



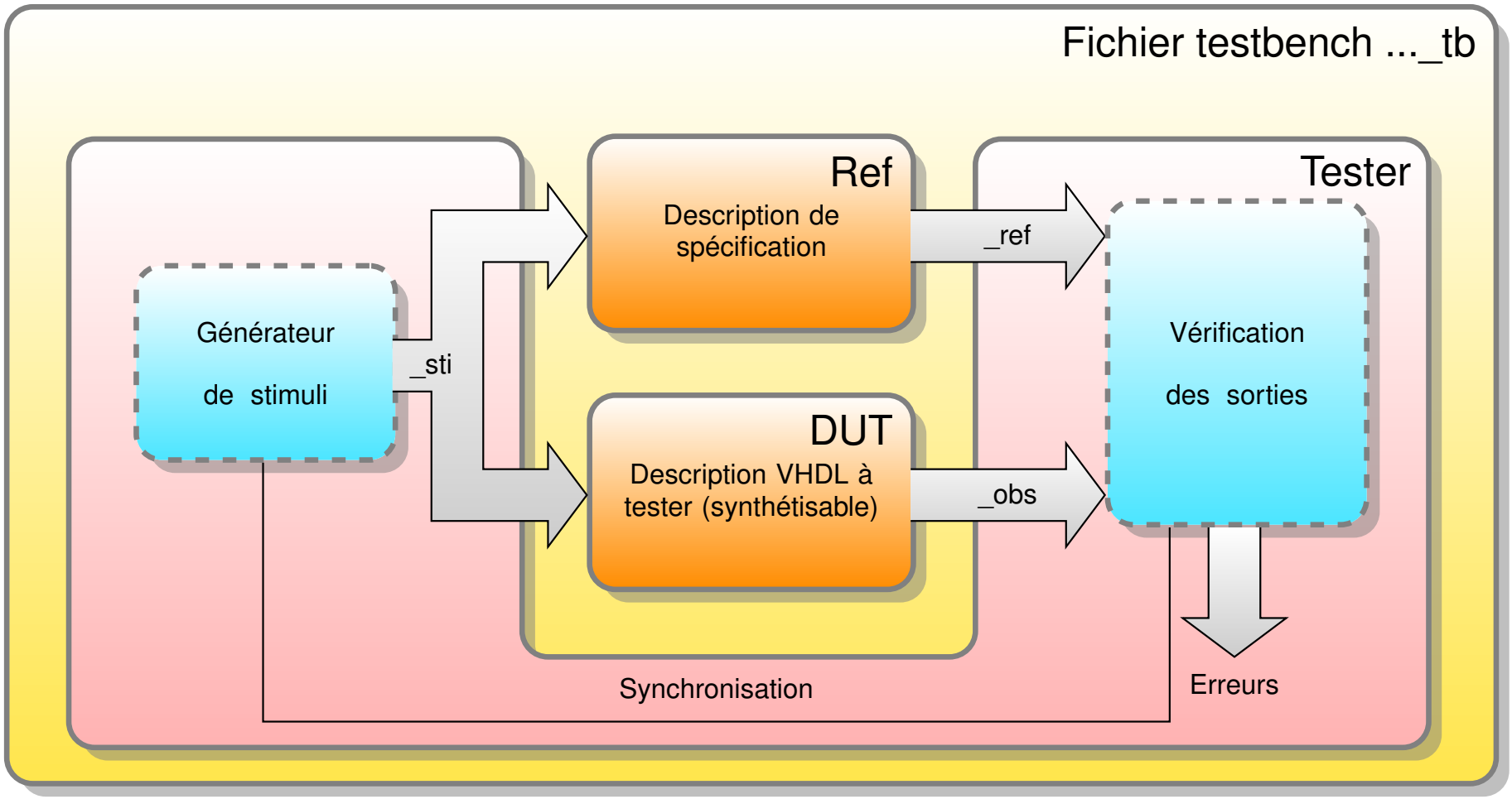
Structure des bancs de test

- Structure avec description de spécification
  - Phase de réalisation du projet
  - Permet de valider le fonctionnement du système



Structure des bancs de test

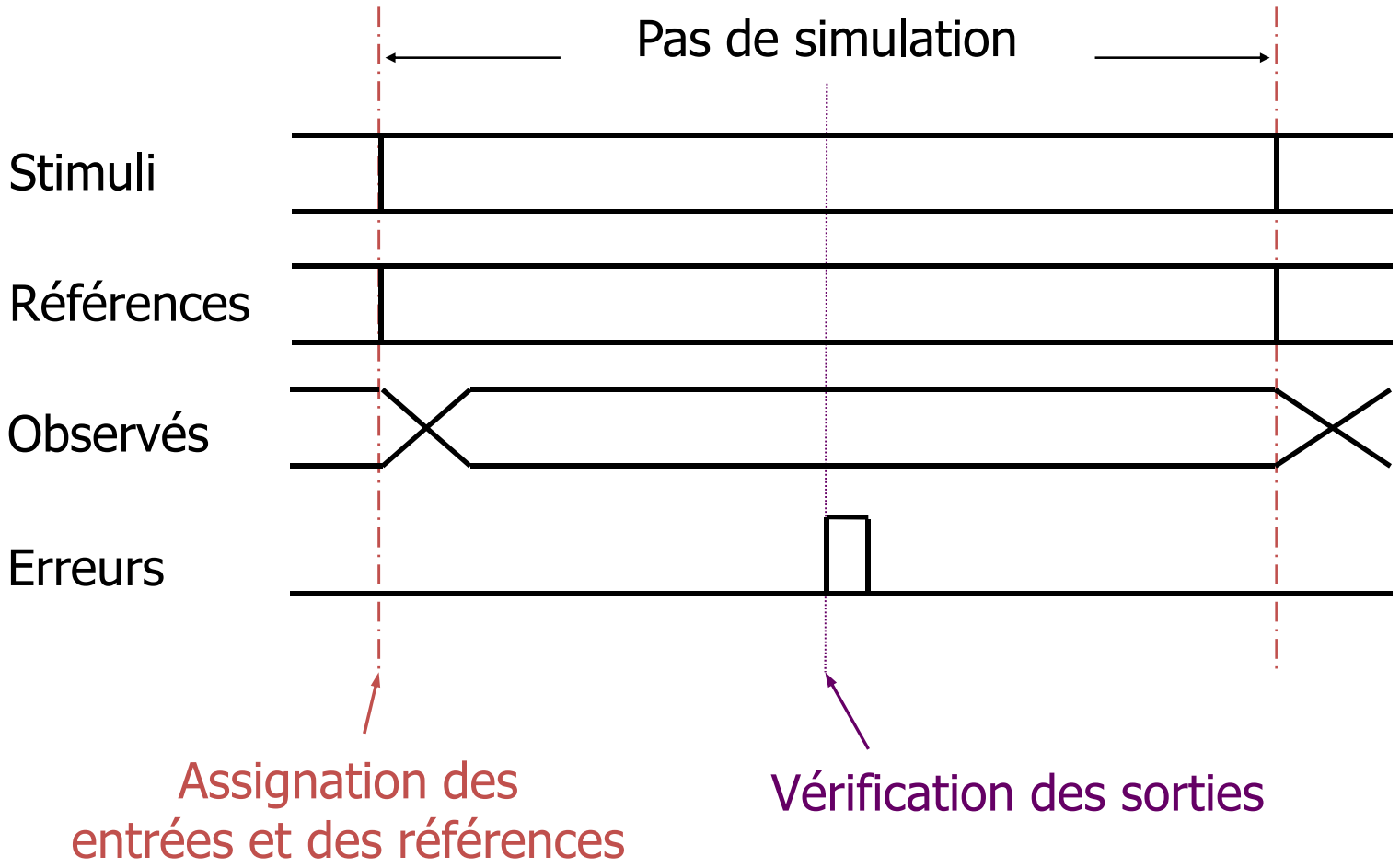
- Structure hiérarchique avec spécification



Simulation d'un système combinatoire

- Concept:
  - On place des valeurs sur les entrées
  - On attend que les sorties soient stabilisées
  - On vérifie que les sorties correspondent
- Les tests peuvent être:
  - Dirigés
  - Aléatoires
  - Un mélange des deux

Simulation d'un système combinatoire



Exemple: simulation d'un multiplexeur (1)

```
entity mux4_1_tb is
end mux4_1_tb;


architecture testbench of mux4_tb is --VHDL93
    component mux4
    port (sel_i            : in  std_logic_vector (1 downto 0);
        a_i,b_i,c_i,d_i : in  std_logic;
        y_o                : out std_logic   );
    end component;
```

Exemple: simulation d'un multiplexeur (2)

```
--declaration de constantes
constant sim_step  : time := 100 ns;
constant pulse     : time := 4 ns;

--signal de détection des erreurs
signal error_signal : std_logic := '0';

--signaux intermédiaires pour la simulation

-- <nom>_sti : Stimuli pour les entrées
signal sel_sti : std_logic_Vector(1 downto 0);
signal a_sti, b_sti, c_sti, d_sti : std_logic;

-- <nom>_obs : Observation des sorties
signal y_obs : std_logic;

-- <nom>_ref : References pour verifier les sorties
signal y_ref : std_logic;
```

Exemple: simulation d'un multiplexeur (3)

```
begin

    -- Interconnexion du module VHDL a simuler
    -- dut :  Design Under Test
    dut: mux4 port map (sel_i =>  sel_sti,
                        a_i   =>  a_sti,
                        b_i   =>  b_sti,
                        c_i   =>  c_sti,
                        d_i   =>  d_sti,
                        y_o   =>  y_obs);
```

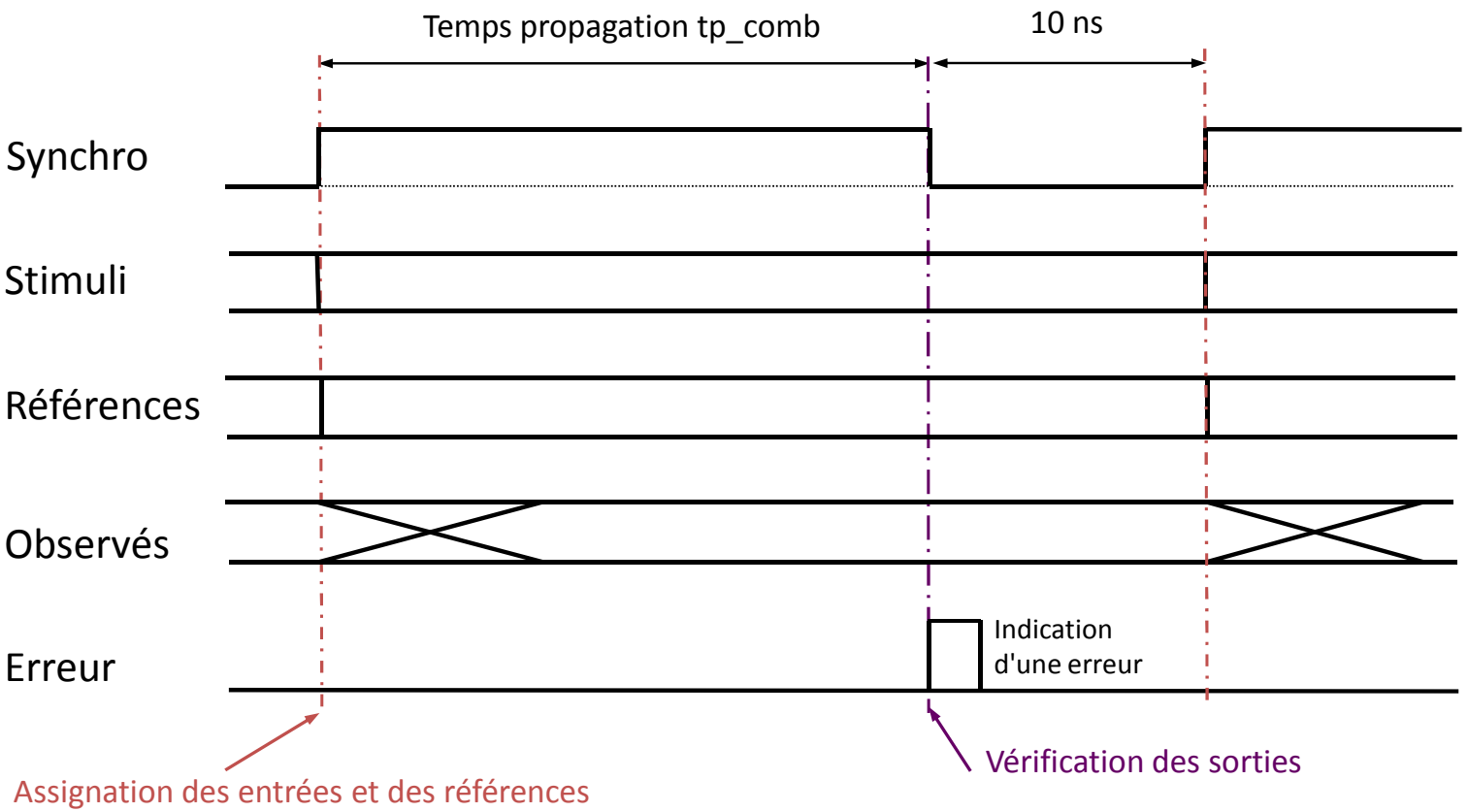
```
-- Debut des pas de simulation
process
    variable nb_errors : Natural;
begin

    nb_errors := 0; -- initialise nombre d'erreurs
    -- debut de la simulation
    report ">>Debut de la simulation";

    --Test sélection 0, A = 0
    sel_sti <= "00";
    a_sti <= '0';  b_sti <= '1';
    c_sti <= '1';  d_sti <= '1';

    y_ref <= '0'; --calcul reference

    wait for sim_step/2;
    if y_obs /= y_ref then
        error_signal <= '1', '0' after pulse;
        nb_errors := nb_errors + 1;
        report ">>Erreur pour Sel = 00 et Y = 0"
               severity ERROR;
    end if;
    wait for sim_step/2;
    --      ...
```

```
        --Test selection 3, D = 1
        sel_sti <= "11";
        a_sti <= '0';  b_sti <= '0';
        c_sti <= '0';  d_sti <= '1';
        y_ref <= '1'; --calcul reference

        wait for sim_step/2;
        if y_obs /= y_ref then
            error_signal <= '1', '0' after pulse;
            nb_errors := nb_errors + 1;
            report ">>Erreur pour Sel = 11 et Y = 1"
                   severity ERROR;
        end if;
        wait for sim_step/2;

        -- fin de la simulation
        report ">>Nombre total erreur=" & integer'image(nb_errors);
        report ">>Fin de la simulation";
        wait; --Attente infinie, stop la simulation
    end process;
end testbench;
```

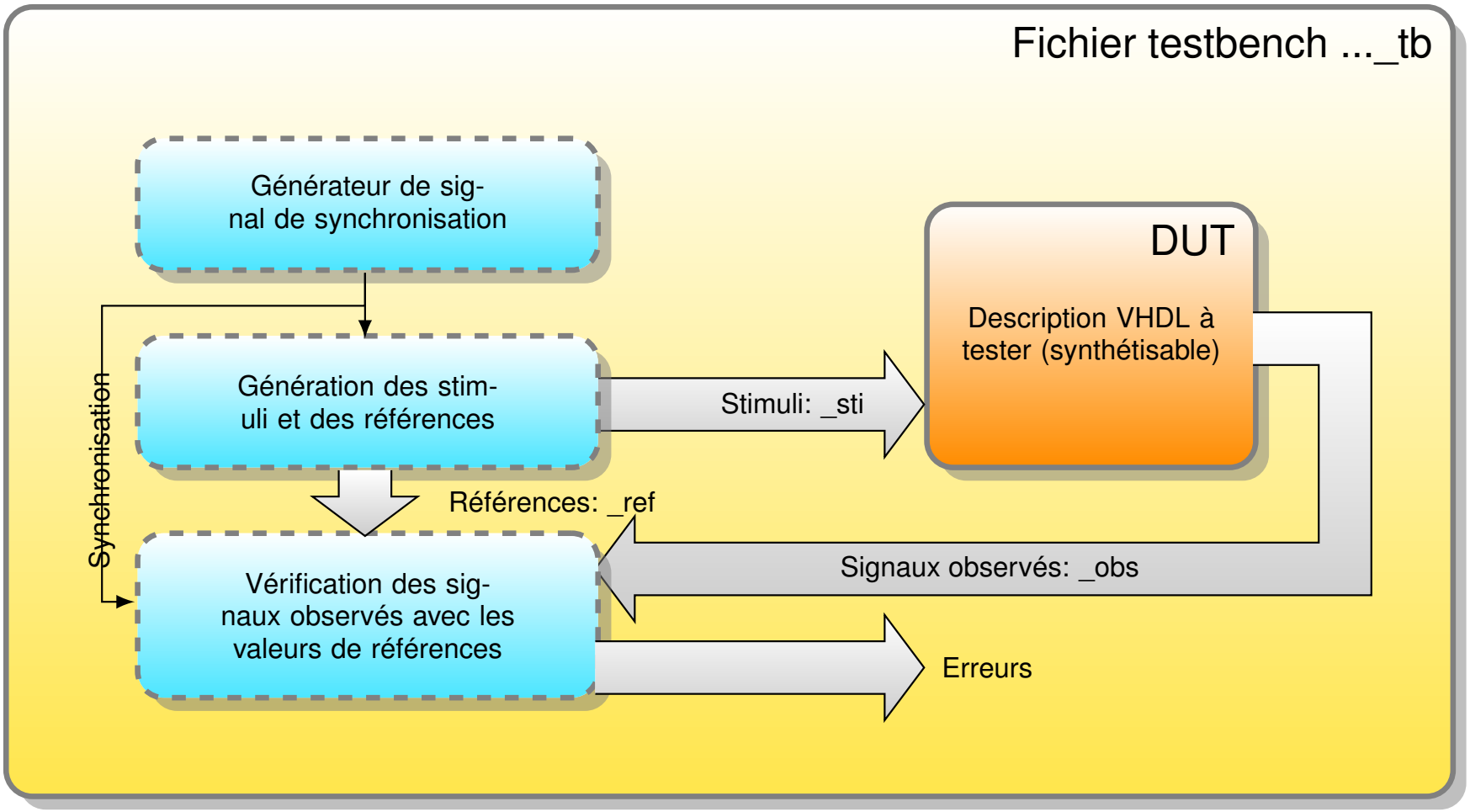
Scrutation de toutes les combinaisons d'un SLC

```
-- Simulation de tous les cas avec une boucle for
for I in 0 to (2**Nbr_E)-1 loop  --simule toutes combin.
  Entrees_Sti <= Std_logic_Vector(To_Unsigned(I,Nbr_E));
   --Calcul de l'etat des sorties avec un algorithme
  Sorties_Ref <= . . . .
   wait for Pas_Sim/2;
  if Sorties_obs /= Sorties_ref then
    Erreur <= '1', '0' after Pulse;
    Nbr_Err := Nbr_Err + 1;
    report ">>Erreur pour I = " & integer'image(I);
       severity ERROR;
  end if;
  wait for Pas_Sim/2;
end loop;
```

Simulation avec signal de synchronisation

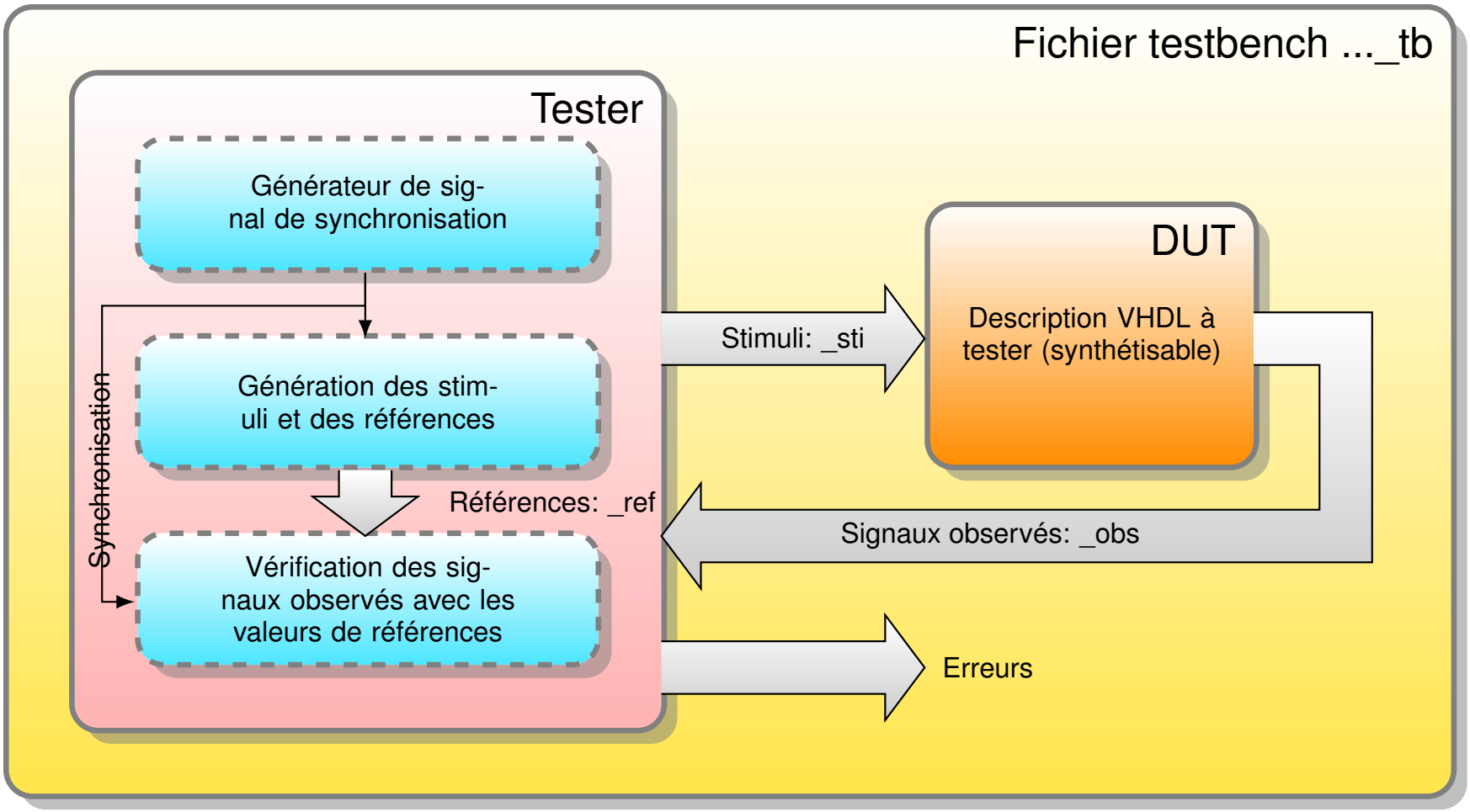


Simulation avec signal de synchronisation



- Structure sans hiérarchie

Simulation avec signal de synchronisation



- Structure avec hiérarchie

Génération du signal de synchronisation

```
process
begin
  Synchro_s <= '0';
  wait for 10 ns;
  while not Sim_End_s loop
    Synchro_s <= '1',
                 '0' after tpcomb;
    wait for (tpcomb + 10 ns);
  end loop;
  wait; --stop le process
end process;
```

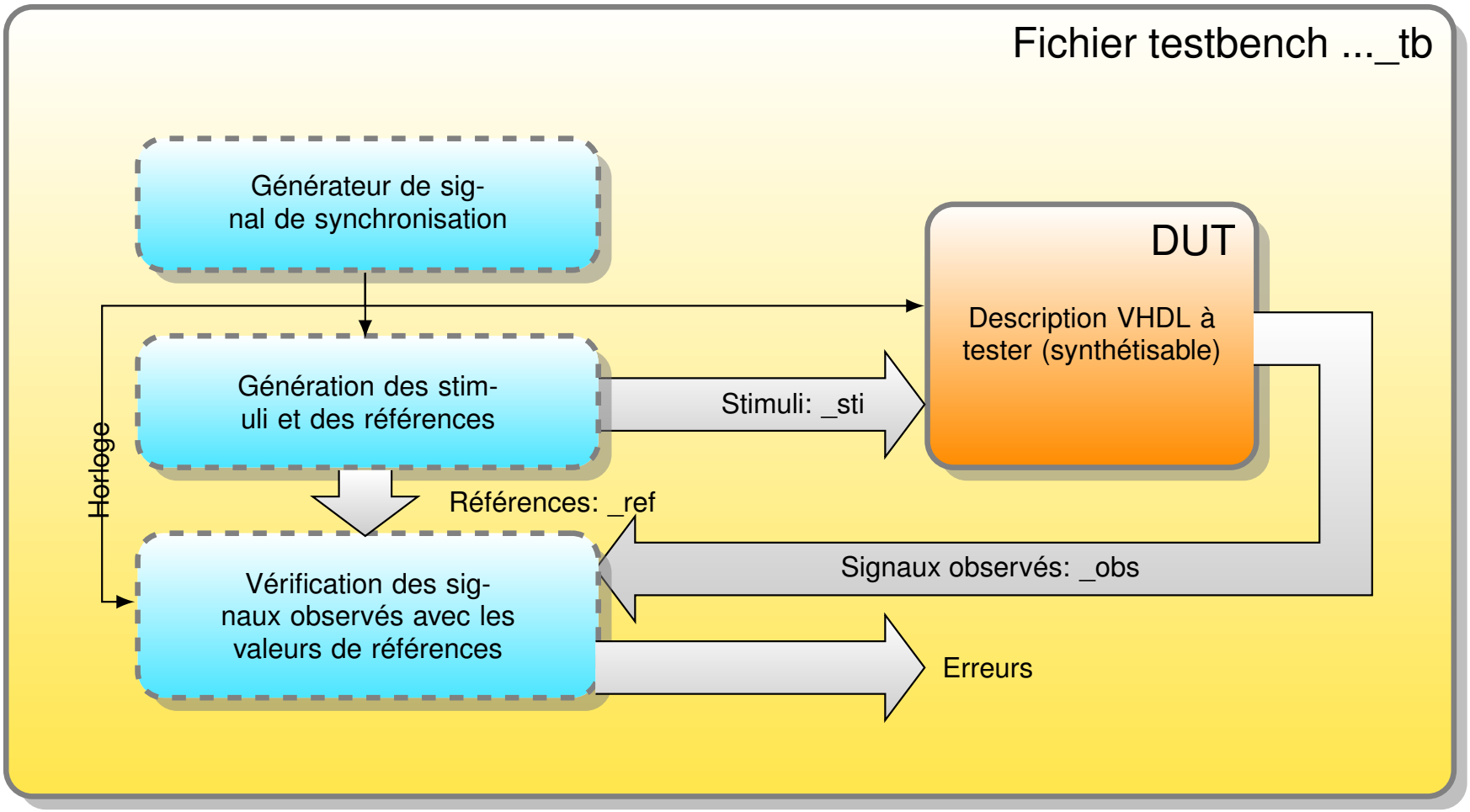
Processus de vérification

```
process
begin
  while not Sim_End_s loop
    wait until Falling_Edge(Synchro_s) or (Sim_End_s);
    if Signal_obs /= Signal_ref then
      Erreur <= '1', '0' after Pulse;
      Nbr_Err := Nbr_Err + 1;
      report "Lors verification Signal"
         severity ERROR;
    end if;
  end loop;
  wait;
end process;
```

Simulation d'un système synchrone

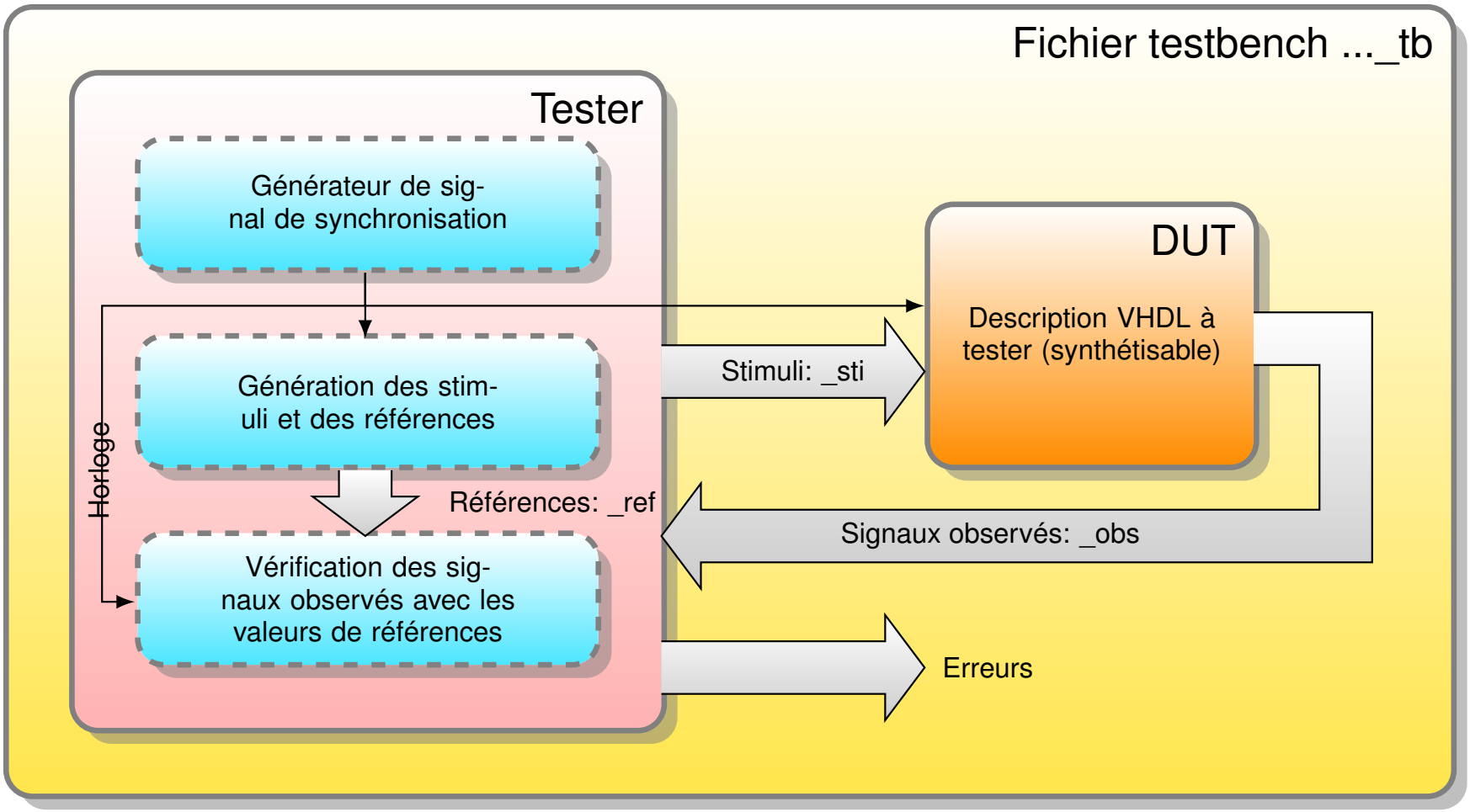
- La génération d'une horloge est nécessaire
- Plus grande difficulté à synchroniser la vérification des sorties
  - Latence du système
- Les assertions peuvent offrir un bon moyen de vérifier les éléments d'un protocole

Simulation d'un système synchrone



- Structure sans hiérarchie

Simulation d'un système synchrone

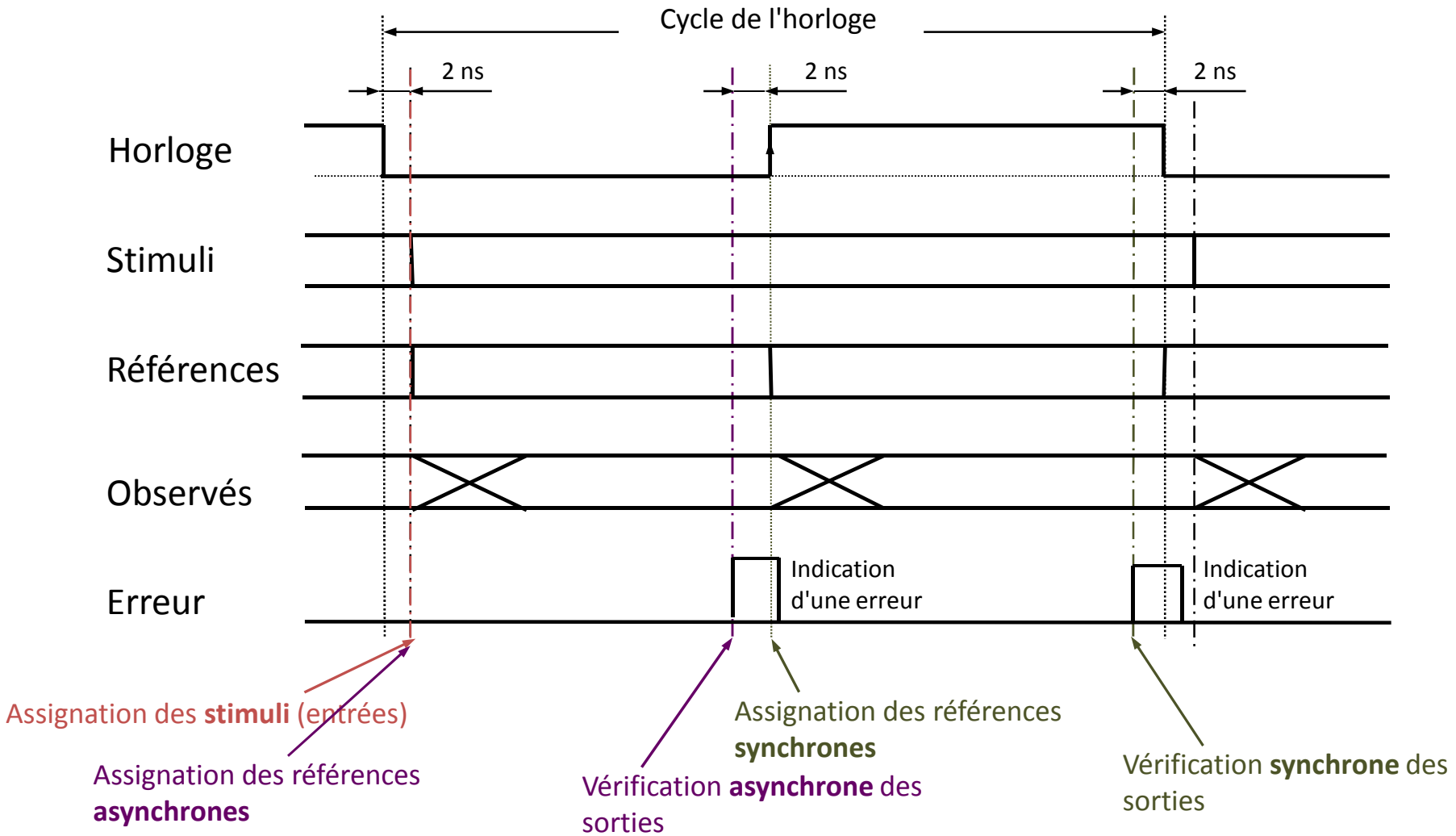


- Structure avec hiérarchie

Simulation d'un système séquentiel

- L'horloge sert de signal de synchronisation
- Vérification fonctionnelle
  - L'horloge n'est pas celle à fréquence réelle
  - Les stimuli sont affectés au même instant
  - Toutes les phases (assignation des stimuli, flanc, vérifications) sont réalisées dans une seule période d'horloge
- Les stimulis sont activés hors des transitions de l'horloge

Simulation d'un système séquentiel



Génération d'une horloge

Géré par un processus concurrent

```
process
begin
  while not Sim_End_s loop
    Clk_sti <= '1',
               '0' after Periode/2;
    wait for Periode;
  end loop;
  wait;
end process;
```

Processus de vérification (1)

Géré par un processus concurrent

```
process
 begin
   while not Sim_End_s loop
     wait until Falling_Edge(Clk_sti);
     wait for (Periode/2)- 2 ns;
     --Verification asynchrone
     if Signal_obs /= Signal_ref then
       Erreur <= '1', '0' after Pulse;
       Nbr_Err := Nbr_Err + 1;
       report"Lors verification Signal, asynch"
         severity ERROR;
     end if;
```

Processus de vérification (2)

Géré par un processus concurrent

```
      wait until Rising_Edge(Clk_sti);
      wait for (Periode/2)- 2 ns;
      --Verification synchrone
      if Signal_obs /= Signal_ref then
        Erreur <= '1', '0' after Pulse;
        Nbr_Err := Nbr_Err + 1;
        report"Lors verification Signal, synch"
          severity ERROR;
      end if;
    end loop;
    wait;
  end process;
```

- Souvent les modules à tester permettent de voir la vérification à un niveau d'abstraction plus élevé
- Accès au module via des appels de procédures
- Les procédures gèrent le protocole d'accès au niveau RTL
- Les paramètres d'entrée/sortie des procédures sont au niveau *transactionnel*
- Ces procédures correspondent au *Bus Functional Model (BFM)*
- Exemple:
  - Mémoire
  - Bus PCIe
  - Fifo
  - ...

Exemple d'une mémoire

- Une mémoire est accédée en lecture ou écriture

$\Rightarrow$ Une procédure d'écriture et une de lecture

- Permet de découpler le protocole bas niveau du fonctionnement haut niveau

Mémoire: write

```
procedure writemem(addr: in std_logic_vector(ADDR_SIZE-1 downto 0);
                   data: in std_logic_vector(DATA_SIZE-1 downto 0)) is
begin
    wait until falling_edge(clk_sti);
    addr_sti    <= addr;
    wr_data_sti <= data;
    wr_sti      <= '1';
    rd_sti      <= '0';
    loop
        wait until falling_edge(clk_sti);
        if busy_obs = '0' then
            exit;
        end if;
    end loop;
    wr_sti <='0';
end writemem;
```

Mémoire: read

```
procedure readmem(addr: in std_logic_vector(ADDR_SIZE-1 downto 0);
                  data: out std_logic_vector(DATA_SIZE-1 downto 0)) is
begin
    wait until falling_edge(clk_sti);
    addr_sti    <= addr;
    wr_sti      <= '0';
    rd_sti      <= '1';
    loop
        wait until falling_edge(clk_sti);
        if data_ready_obs = '1' then
            data := rd_data_obs;
            exit;
        end if;
    end loop;
    rd_sti <='0';
end readmem;
```

Mémoire: test case

```
procedure testcase1 is
    variable data_out : std_logic_vector(DATA_SIZE-1 downto 0);
begin
    writemem(x"0000",x"0001");
    writemem(x"0001",x"0004");
    writemem(x"0002",x"0007");

    readmem(x"0001", data_out);
    check(data_out, x"0004");

    readmem(x"0002", data_out);
    check(data_out, x"0007");

    readmem(x"0000", data_out);
    check(data_out, x"0001");

end testcase1;
```

Mémoire: check

```
procedure check(value1: in std_logic_vector(DATA_SIZE-1 downto 0);
                value2: in std_logic_vector(DATA_SIZE-1 downto 0)) is
begin
    assert value1 = value2 report "Aie, bad error" severity error;
end testcase1;
```

BFM: Avantages

- Les test cases sont découplés du protocole bas niveau
- Dans le cas d'une mémoire, il serait possible de réexploiter le test case pour un autre type de mémoire
  - Seul le contenu des procédures du BFM doit être modifié
- L'ingénieur qui décrit les tests n'a pas besoin de connaître le protocole au niveau RTL $\Rightarrow$ peut être laisser à la responsabilité de quelqu'un d'autre

Déclaration du BFM

- Limitation de VHDL: Pour pouvoir accéder aux signaux en écriture il faut que la procédure soit déclarée dans le processus

```
stimuli_proc: process is

    procedure writemem(addr: in std_logic_vector(ADDR_SIZE-1 downto 0);
                       data: in std_logic_vector(DATA_SIZE-1 downto 0)) is
    begin
        ...
    end writemem;
    ....
begin
    testcase1;
end process;
```

  - $\Rightarrow$ Ceci limite la réutilisabilité du code
- Autre option:
  - Passer les signaux RTL aux procédures, qui peuvent alors être placées dans un paquetage (par exemple)

Interfaces asynchrones

- Exemples d'interfaces asynchrones :
  - Mémoire avec accès asynchrone
    - DRAM
    - DDR
  - Communication inter-chips
    - FX2/3
    - FTDI
  - Communication asynchrone
    - UART
    - SPI
    - I^2C

Interfaces asynchrones

- Les interfaces asynchrones demandent une autre approche, car elles ne possèdent pas d'horloge, tout en faisant appel à la notion de temps
- Utilisation du `wait`, `wait on` et `wait until`
- Gestion de différents types de signaux:
  - Signaux de contrôle
  - Signaux de données
- Vérifications typiques:
  - Les données ont été stables suffisamment longtemps
  - Un signal de requête a été asserté le temps nécessaire
  - Deux requêtes sont séparées par un temps suffisant

Méthodologie

- Il faut émuler le système communiquant avec le DUT
  - Respect des timings asynchrones par la partie émulée
  - Attention à ne pas bloquer le banc de test si le DUT ne respecte pas les timings ou la datasheet
    - Exemple: Le DUT ne donne jamais d'acknowledge à une requête et le banc de test attend indéfiniment sur celui-ci

    $\Rightarrow$ Le test ne se termine jamais (pour un banc de test automatique)
- Il faut vérifier que le DUT respecte les différentes contraintes de timing

Bancs de tests génériques

- Dans l'optique d'utiliser des scripts pour lancer les simulations, des paramètres génériques du banc de tests générique peuvent:
  - Offrir un nom de fichier de Log
  - Influencer la configuration du design
    - En passant des paramètres génériques au design
  - Influencer la génération de stimuli
  - Influencer la vérification apportée

Banc de test générique: exemple

- L'entité du banc de test
  - N'a pas de ports
  - Peut avoir des paramètres génériques

Entité

```
entity mon_design_tb is
    generic(
        CONFIGINFO   : integer := 0;
        NBLINES      : integer := 1;
        VERIFDATA    : boolean := true;
        VERIFCONTROL : boolean := true;
        LOGFILENAME  : string
    );
end mon_design_tb;
```

Banc de test générique: exemple

Architecture

```
architecture testbench of mon_design_tb is

    FILE log_file : TEXT open WRITE_MODE is LOGFILENAME;

begin

    un_process: process
        variable l: line;
    begin
        write(l,string'("Starting simulation"));
        writeline(log_file,l);
        write(l,string'("CONFIGINFO = "),left,10);
        write(l,CONFIGINFO,Left,10);
        writeline(log_file,l);
        ...
    end process;

    ...
```

Banc de test générique: exemple

Architecture

```
        ...

        -- Instanciation du DUT, avec paramètre générique
        dut: Mon_Design
        generic map (NBLINES => NBLINES)
        port map (
            clk_i => clk_sti,
            ...
        );

    end testbench;
```

Banc de test générique: lancement

- Une modification des valeurs génériques ne nécessite pas de recompilation
- Les paramètres peuvent être fournis au lancement de la simulation (élaboration)

Exemple

```
vsim -t 1ns -novopt -GCONFIGINFO=1
                    -GVERIFCONTROL=false
                    -GLOGFILENAME="unfichier.txt"
                    work.mon_design_tb
```

Banc de test générique: automatisation

- Possibilité de lancer automatiquement plusieurs simulations

Exemple (dans sim.do)

```
vsim -t 1ns -novopt -GCONFIGINFO=1
            -GLOGFILENAME="unfichier.txt"
            work.mon_design_tb
run -all

vsim -t 1ns -novopt -GCONFIGINFO=2
            -GLOGFILENAME="unautrefichier.txt"
            work.mon_design_tb
run -all

...
```

Banc de test générique: modularité

- Et le tout dans une fonction...

Exemple (dans sim.do)

```
proc sim_start {logfile configinfo} {
    vsim -t 1ns -novopt -GCONFIGINFO=$configinfo
         -GLOGFILENAME=$logfile work.mon_design_tb
    run -all
}

...

sim_start "fichier1.txt" 1
sim_start "fichier2.txt" 8
sim_start "fichier3.txt" 15
...
```

Fin de tests

- Un banc de tests automatique doit se terminer lui-même
- Système standard: un signal pour la détection de fin
  - Généré par le processus de génération ou vérification
  - Permet d'arrêter l'horloge

Déclaration du signal

```
architecture testbench of mon_testbench is

    signal sim_end_s : boolean := false;

    ...
```

Fin de tests

Processus de génération d'horloge

```
clk_proc: process is
begin
    loop
        clk_sti <= '0';
        wait for CLK_PERIOD/2;
        clk_sti <= '1';
        wait for CLK_PERIOD/2;
        if (sim_end_s) then
            wait;
        end if;
    end loop;
end process;
```

Fin de tests

Processus de génération de stimuli

```
sti_proc: process is
begin
    for i in 1 to 1000 loop
        wait until falling_edge(clk_sti);
        input_sti <= ...;
    end loop;

    -- Fin de simulation
    sim_end_s <= '1';
    wait;
end process;
```

- La simulation s'arrête automatiquement avec l'arrêt de l'horloge

Fin de tests

- Problème si deux processus doivent déclencher une fin de simulation!
- Deux processus ne peuvent pas agir sur un même signal booléen
- Solution: un signal par processus

Déclaration du signal

```
architecture testbench of mon_testbench is

    signal sim_end1_s : boolean := false;
    signal sim_end2_s : boolean := false;

    ...
```

Fin de tests

Processus de génération d'horloge

```
clk_proc: process is
begin
    loop
        clk_sti <= '0';
        wait for CLK_PERIOD/2;
        clk_sti <= '1';
        wait for CLK_PERIOD/2;
        if (sim_end1_s or sim_end2_s) then
            wait;
        end if;
    end loop;
end process;
```

- Problème de scalabilité et de dépendance du code

Objections

- Les objections sont une solution!
- Chaque processus peut:
  - Lever une objection à son démarrage
  - La relâcher lorsqu'il autorise la fin de simulation
- La génération d'horloge peut vérifier s'il y a encore une objection levée ou non
- Pour ce faire, utilisation de `objection_pkg.vhd` (développé au REDS)

Utilisation du paquetage

```
use work.objection_pkg.all;
```

Paquetage objection_pkg

```
package objection_pkg is

    -- This type corresponds to an objection manager. It offers 4 function
    -- to raise, drop, and check the objections.
    type objection_type is protected

        -- Raises a number of objections (default: 1)
        procedure raise_objection(nb_obj: integer := 1);

        -- Drops a number of objections (default: 1)
        procedure drop_objection(nb_obj: integer := 1);

        -- Drops all objections
        procedure drop_all_objections;

        -- Returns true if there is no more objection raised
        impure function no_objection return boolean;

    end protected objection_type;

    ...
```

Paquetage objection_pkg

```
    ...

    -- The 4 following subprograms access a single objection object, and s
    -- can be used by the entire testbench, without the need of declaring
    -- objection object. Basically they are offered for convenience.

    -- Raises an objection on the singleton
    procedure raise_objection(nb_obj: integer := 1);

    -- Drops an objection on the singleton
    procedure drop_objection(nb_obj: integer := 1);

    -- Drops all objections on the singleton
    procedure drop_all_objections;

    -- Indicates if all objections have been dropped on the singleton
    impure function no_objection return boolean;

end objection_pkg;
```

Fin de tests

Processus de génération d'horloge

```
-- Generation of a clock, while there is at least an objection raised
clk_proc: process is
begin
    clk_sti <= '0';
    wait for CLK_PERIOD/2;
    clk_sti <= '1';
    wait for CLK_PERIOD/2;
    if no_objection then
        wait;
    end if;
end process;
```

Fin de tests

Processus de génération de stimuli

```
-- Stimuli process number 1
sti1_proc: process is
begin
    -- We do not want to be interrupted
    raise_objection;

    -- Let's stimulate the design
    wait for 200 ns;
    an_output_o <= '1';
    wait for 50 ns;

    -- OK, we are over with the stimulation from this process
    drop_objection;
    wait;
end process;
```

- La simulation s'arrête automatiquement avec le relâchement de toutes les objections

Fin de tests

- Autre option: les battements de coeur
- Idée: les éléments du banc de test peuvent indiquer qu'ils sont en vie
- Si plus aucun ne l'est et s'il n'y a plus d'objections, alors la simulation peut s'arrêter
- Implémentation: paquetage `heartbeat_pkg`, développé au REDS

Fin de tests

- Exploitation des objections et des battements de coeur pour arrêter la simulation
- Implémentation: paquetage `simulation_end_pkg`, développé au REDS
- Permet d'instancier un moniteur de fin de simulation
- Et de faire appeler une procédure lorsque c'est le cas

Fin de tests

Contenu de `simulation_end_pkg`

```
-- This type allows the reporting function to know why the simulation
-- Because of:
--      - No objection raised any more and drain time elapsed
--      - No beat event so the simulation seems to be stuck somewhere
--      - None of the above, should not occur
type finish_status_t is (NO_OBJECTION, NO_BEAT, UNKNOWN_ENDING);

component simulation_monitor is
generic (
    -- Drain time for objections
    DRAIN_TIME : time := 0 ns;
    -- Beat time for the heartbeat
    BEAT_TIME  : time := 0 ns;
    -- Function called at the end of the simulation
    procedure final_reporting(finish_status: finish_status_t);
    -- If yes, then the simulator exits when the simulation is over
    should_finish : boolean := true);
end component;
```

Fin de test

Exemple d'utilisation

```
architecture testbench of my_testbench is

    procedure end_of_simulation(finish_status: finish_status_t) is
    begin
        report "I finished, yepee";
    end end_of_simulation;

begin

    monitor: simulation_monitor
    generic map (drain_time => 50 ns,
                 beat_time => 400 ns,
                 final_reporting => end_of_simulation);

    ...

end testbench;
```

Librairie Tlmvm

- *Transaction Level Modeling VHDL Methodology*: TLMVM
- Ces différents éléments:
  - Fifos de transactions
  - Objections
  - Battements de coeur
  - Détection de fin de simulation
- Sont dans la librairie `tlmvm`
- Disponible sur le site du cours
  - Fichiers sources
  - Exemples directement compilables via des scripts