Vérification fonctionnelle des systèmes numériques

Introduction

Yann Thoma

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

Février 2017

But de la vérification

Répondre à 2 questions

But de la vérification

Répondre à 2 questions

- Est-ce que ça marche?

But de la vérification

Répondre à 2 questions

- Est-ce que ça marche?
- Est-ce qu'on est sûr?

But de la vérification

Répondre à 2 questions

- Est-ce que ça marche?
- Est-ce qu'on est sûr?
- Vraiment?

But de la vérification

Répondre à 2 questions

- Est-ce que ça marche?
- Est-ce qu'on est sûr?
- Vraiment?
- Non, mais vraiment vraiment?

But de la vérification

Répondre à 2 questions

- Est-ce que ça marche?
- Est-ce qu'on est sûr?
- Vraiment?
- Non, mais vraiment vraiment?
- Sans plaisanter?

Vérification : oui, mais ...

On vérifie ...

- Quoi?
- Pourquoi?
- Quand?
- Comment?
  - Cette question sera centrale durant ce cours

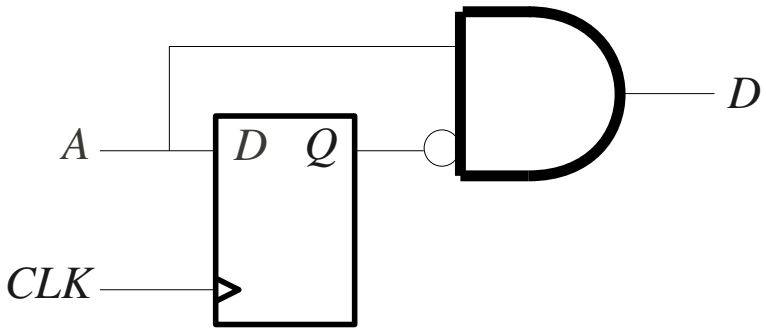
Première question: Quoi?

- Circuits programmables
- ASIC
- Dans le cadre de ce cours, seulement des circuits digitaux
- La vérification du logiciel est également importante
  - N'entre pas dans le cadre de ce cours
  - Offre plus de flexibilité et d'observabilité
  - Un patch, c'est si facile...
    - Vraiment?

Première question: Quoi?

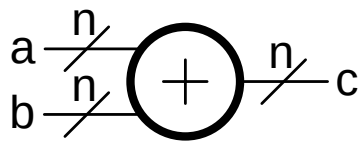
- Vérification du bon fonctionnement du système
- Qu'est-ce que cela signifie?
- Vérification de chaque module
  - Vérification exhaustive possible?

- Un détecteur de flanc:



  - Vérification exhaustive: OK

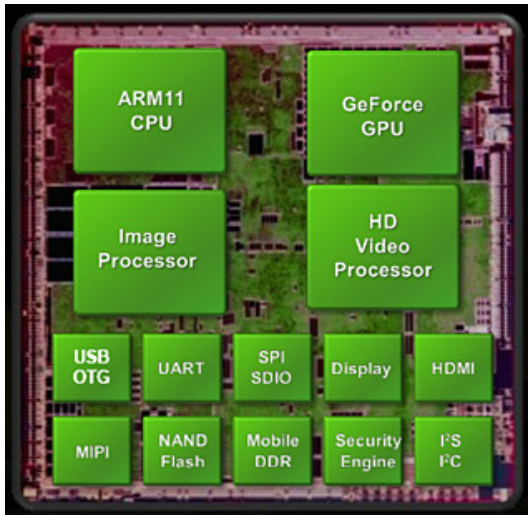
- Un additionneur 64-bits:



  - Vérification exhaustive: Non
  - $2^{128} = 3.4 \times 10^{38}$ combinaisons d'entrées...

Première question: Quoi?

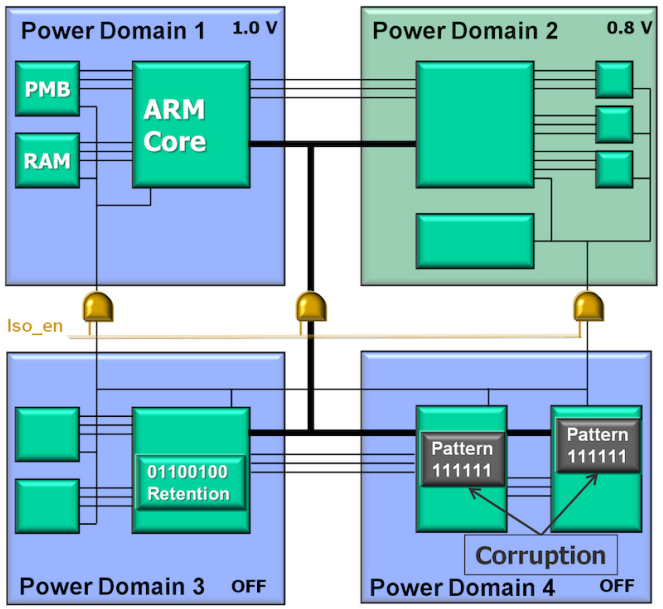
- Vérification du bon fonctionnement du système
- Qu'est-ce que cela signifie?
- Vérification du système entier (System on Chip, par exemple)
  - Impossible de faire une vérification exhaustive
  - $\Rightarrow$ Choix de cas intéressants
  - $\Rightarrow$ Vérification aléatoire



http://www.xbitlabs.com/news/mobile/display/20080603141353_Nvidia_Unleashes_Tegra_System_on_Chip_for_Handheld_Devices.html

Première question: Quoi?

- Vérification du bon fonctionnement du système
- Qu'est-ce que cela signifie?
- Tendance actuelle: Power aware verification
  - SOCs ont plusieurs alimentations
  - Certains sous-systèmes peuvent être déconnectés ponctuellement
  - A la frontière du digital et de l'analogique



http://www.techdesignforums.com/practice/technique/emulation-system-level-power-verification/

Question: Pourquoi?

- Dépannage d'un PLD programmé
  - Très coûteux
  - Voire impossible
- Dépannage d'un ASIC
  - Très coûteux (en temps et argent)
  - Refonte du circuit
- Vérification indispensable pour valider le système
  - Nécessite des outils performants
  - Une méthodologie rigoureuse
- La vérification est *le défi* de la conception des systèmes numériques

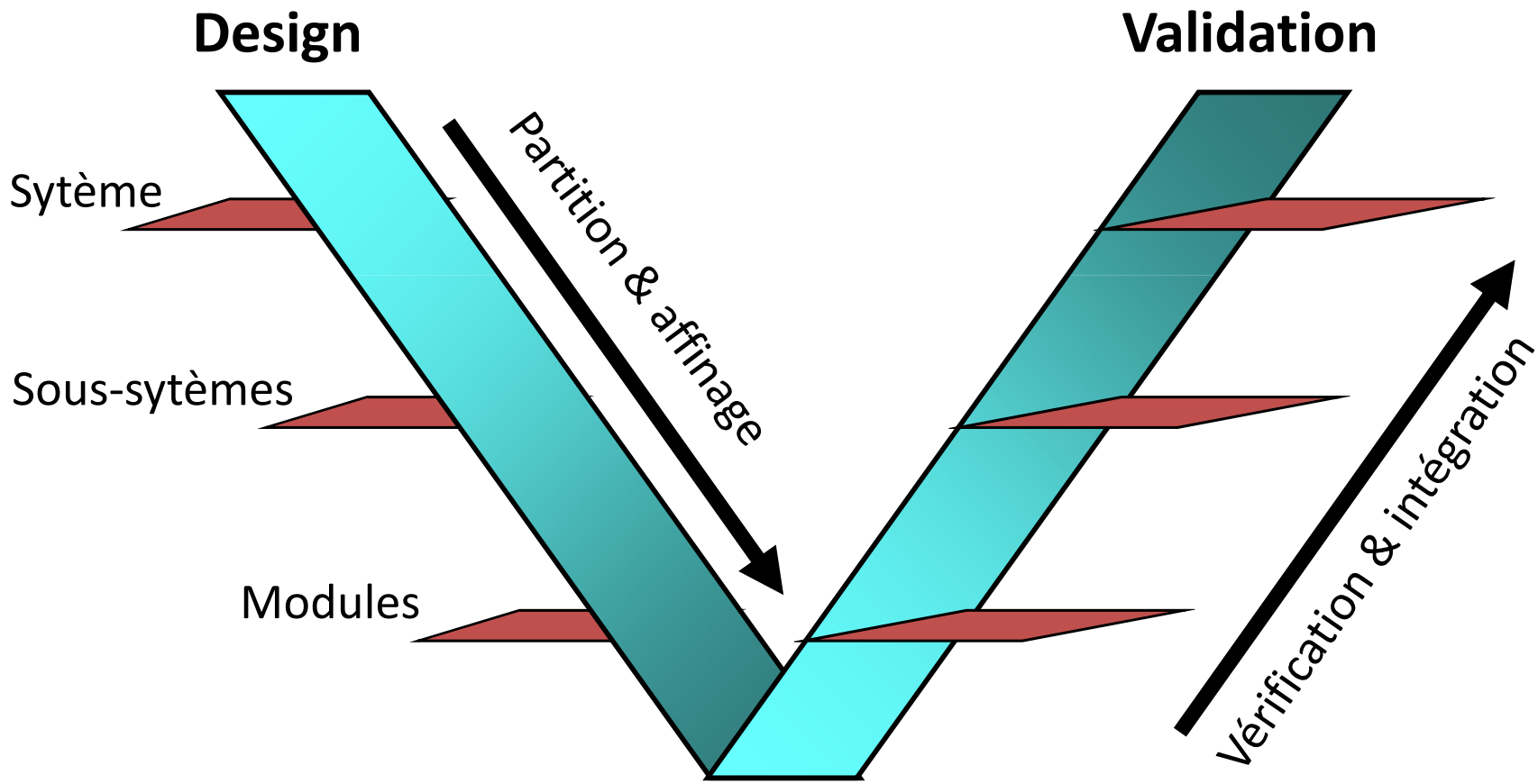
Vérification: concept

- Buts
  1. Vérifier la conformité du système avec le cahier des charges
  2. Détecter les erreurs au plus vite
  3. Assurer un fonctionnement correct après synthèse et intégration
- Difficulté
  1. Maîtriser les coûts de la vérification
     - Environ 50-60% du temps ingénieur
  2. Garantir un fonctionnement correct

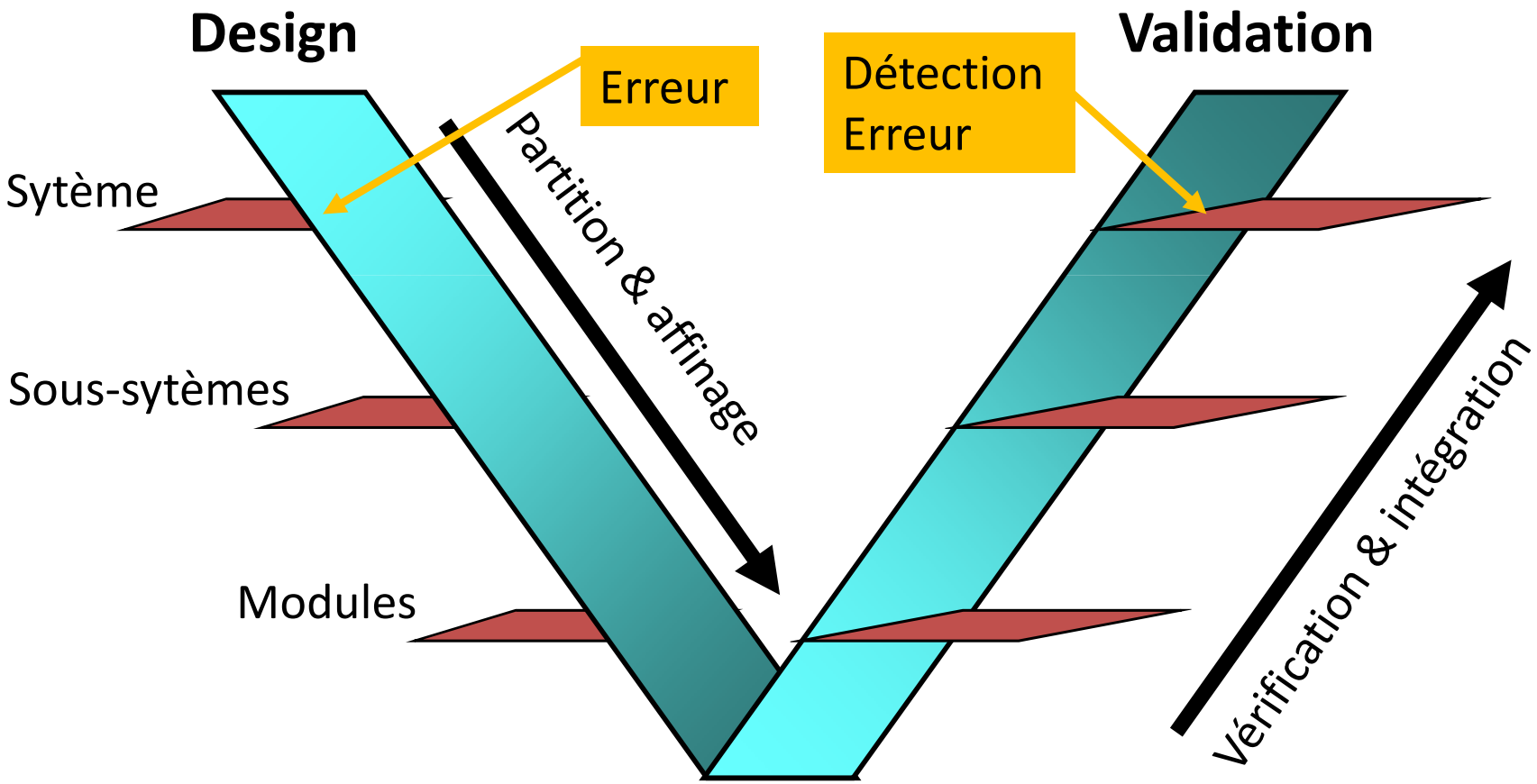
Question: Quand?

- Il semble évident d'essayer de détecter les erreurs le plus vite possible
- Tests unitaires sur les modules au fur et à mesure de la conception/implémentation
- Tests systèmes dès que possible
  - Nettement plus délicats à mettre en place
  - Temps de simulation importants
  - Quand? La nuit...
- Pour un ASIC
  - Lorsque le circuit est fondu, c'est trop tard...
  - Tous les tests doivent être fait avant
  - Prévoir la possibilité de tester après (cas d'un prototype notamment)
  - Concept intéressant: Design for Test
    - Le système embarque de la logique permettant de le tester durant son fonctionnement
    - Permet de faire du diagnostic
    - Exemple: JTAG, BIST

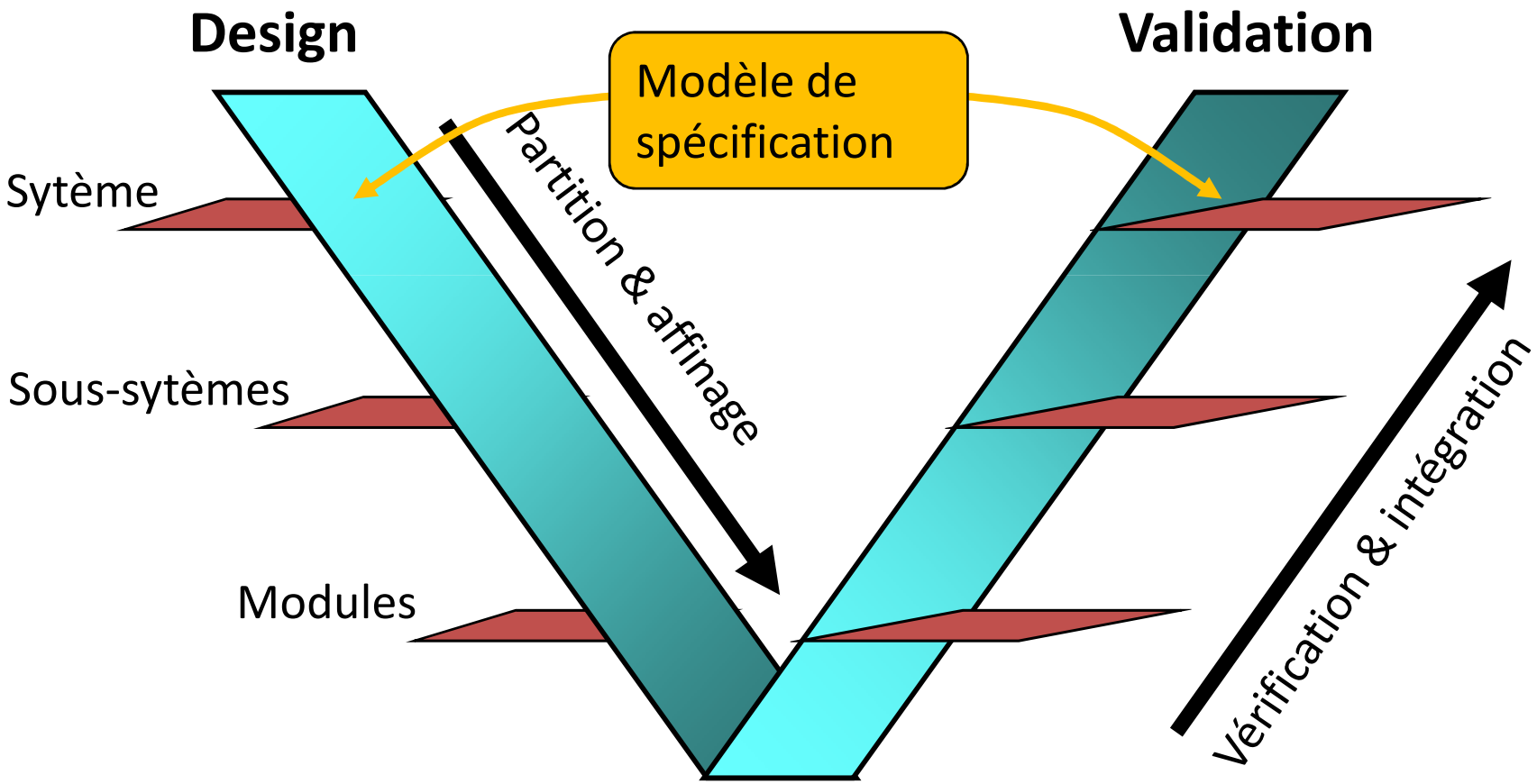
Conception/Vérification: décomposition



Conception/Vérification: décomposition



Conception/Vérification: décomposition



Vérification: les 2 questions

- Les deux grandes questions de la vérification

Vérification: les 2 questions

- Les deux grandes questions de la vérification
  1. Est-ce que ça marche?

Vérification: les 2 questions

- Les deux grandes questions de la vérification
  1. Est-ce que ça marche?
     - Est-ce que le système fonctionne correctement?

Vérification: les 2 questions

- Les deux grandes questions de la vérification
  1. Est-ce que ça marche?
     - Est-ce que le système fonctionne correctement?
     - Les erreurs sont-elles correctement détectées?

Vérification: les 2 questions

- Les deux grandes questions de la vérification
  1. Est-ce que ça marche?
     - Est-ce que le système fonctionne correctement?
     - Les erreurs sont-elles correctement détectées?

Exemple de banc de test qui passe tous les tests

```
entity MonSuperDesign_tb is
end MonSuperDesign_tb;

architecture MaSuperArchitecture of MonSuperDesign_tb is
begin
    process is
    begin
        report "Everything is fine";
    end process;
end MaSuperArchitecture;
```

Vérification: les 2 questions

- Les deux grandes questions de la vérification
  1. Est-ce que ça marche?
     - Est-ce que le système fonctionne correctement?
     - Les erreurs sont-elles correctement détectées?

Exemple de banc de test qui passe tous les tests

```
entity MonSuperDesign_tb is
end MonSuperDesign_tb;

architecture MaSuperArchitecture of MonSuperDesign_tb is
begin
    process is
    begin
        report "Everything is fine";
    end process;
end MaSuperArchitecture;
```

  2. Est-ce qu'on est sûr? (a-t-on fini?)

Vérification: les 2 questions

- Les deux grandes questions de la vérification
  1. Est-ce que ça marche?
     - Est-ce que le système fonctionne correctement?
     - Les erreurs sont-elles correctement détectées?

Exemple de banc de test qui passe tous les tests

```
entity MonSuperDesign_tb is
end MonSuperDesign_tb;

architecture MaSuperArchitecture of MonSuperDesign_tb is
begin
    process is
    begin
        report "Everything is fine";
    end process;
end MaSuperArchitecture;
```

  2. Est-ce qu'on est sûr? (a-t-on fini?)
     - A-t-on réalisé assez de tests?

Vérification: les 2 questions

- Les deux grandes questions de la vérification
  1. Est-ce que ça marche?
     - Est-ce que le système fonctionne correctement?
     - Les erreurs sont-elles correctement détectées?

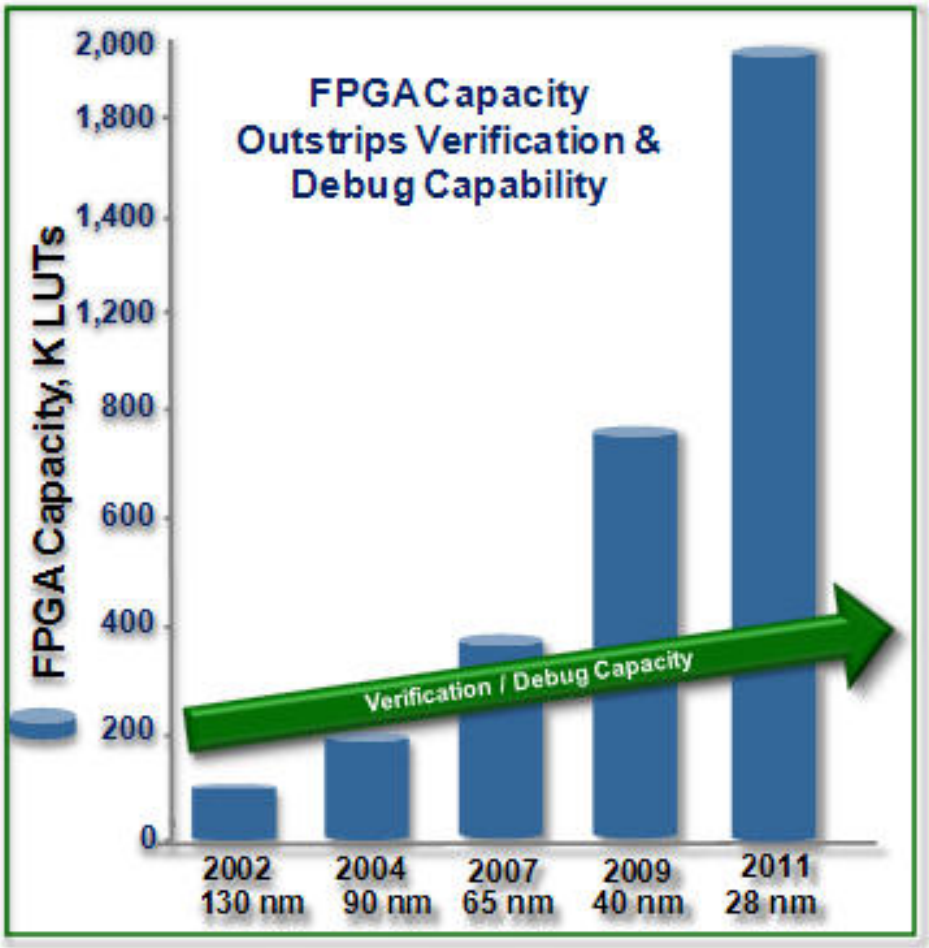
Exemple de banc de test qui passe tous les tests

```
entity MonSuperDesign_tb is
end MonSuperDesign_tb;

architecture MaSuperArchitecture of MonSuperDesign_tb is
begin
    process is
    begin
        report "Everything is fine";
    end process;
end MaSuperArchitecture;
```

  2. Est-ce qu'on est sûr? (a-t-on fini?)
     - A-t-on réalisé assez de tests?
     - Facile avec un petit design, très difficile avec un design complexe

Vérification: évolution de la complexité



http://www.eetimes.com/electronics-blogs/other/4209669/FPGAs-advance–but-verification-challenges-increase

Question: Comment?

- Choix du langage
  - VHDL
  - SystemVerilog
  - SystemC
  - ...
- Choix de la méthodologie
  - UVM
  - VMM
  - Spécifique
  - ...
- Choix technologique
  - Vérification formelle
  - Vérification basée sur la simulation
  - Vérification par émulation
  - ...
- Choix interactif
  - Manuel
  - Automatique

Question: Comment?

- Choix du langage
  - VHDL
  - SystemVerilog
  - SystemC
  - ...
- Choix de la méthodologie
  - UVM
  - VMM
  - Spécifique
  - ...
- Choix technologique
  - Vérification formelle
  - Vérification basée sur la simulation
  - Vérification par émulation
  - ...
- Choix interactif
  - Manuel
  - Automatique

Défis de la vérification: choix de méthode

- Le choix d'une solution pour la vérification doit prendre en compte les paramètres suivants:
  - Complétude
    - Maximiser les fonctionnalités (scénarios) de design testées
  - Réutilisabilité
    - Maximiser la réutilisabilité du code pour de futurs projets
  - Efficience
    - Minimiser l'effort à fournir et automatiser un maximum
  - Productivité
    - Maximiser le travail produit manuellement
  - Performance du code
    - Minimiser le temps de calcul nécessaire à la vérification

Vérification fonctionnelle: approches

- Vérification formelle
  - Tentative de prouver mathématiquement le bon fonctionnement du système
- Vérification basée sur la simulation
  - Tentative de prouver le bon fonctionnement du système en le simulant
- Vérification par émulation
  - Tentative de prouver le bon fonctionnement du système en l'émulant

Vérification formelle

- Concept: Prouver mathématiquement que le design est correct!
- Se base sur la logique mathématique
  - Logique propositionnelle
  - Logique des prédicats
- Semblable à la preuve automatique de théorèmes

Exemples de propositions

$A \Rightarrow B$ est équivalent à $\overline{A}$ ou B
modus ponens: Si A et $A \Rightarrow B$, alors B
modus ponens: peut s'écrire: (A et $A \Rightarrow B$)$\Rightarrow B$
contrapositive de l'implication: $(A \Rightarrow B) \Leftrightarrow (\overline{B} \Rightarrow \overline{A})$
Tiers exclu: A ou $\overline{A}$

Par la contrapositive de l'implication, la proposition "Tout ingénieur documente son code" est équivalente à "Tout code non documenté est écrit par un non-ingénieur"

Diagramme de Venn: A et B

Imaginez: $A \equiv$ est un animal, et $B \equiv$ est noir

A et B

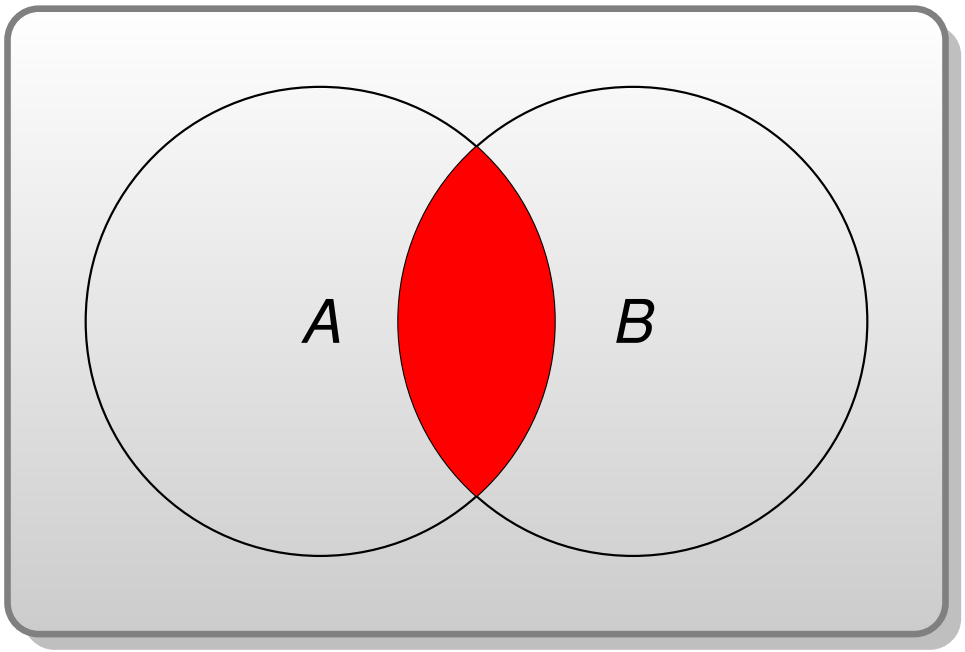


Diagramme de Venn: A ou B

Imaginez: $A \equiv$ est un animal volant, et $B \equiv$ est un animal qui nage

A ou B

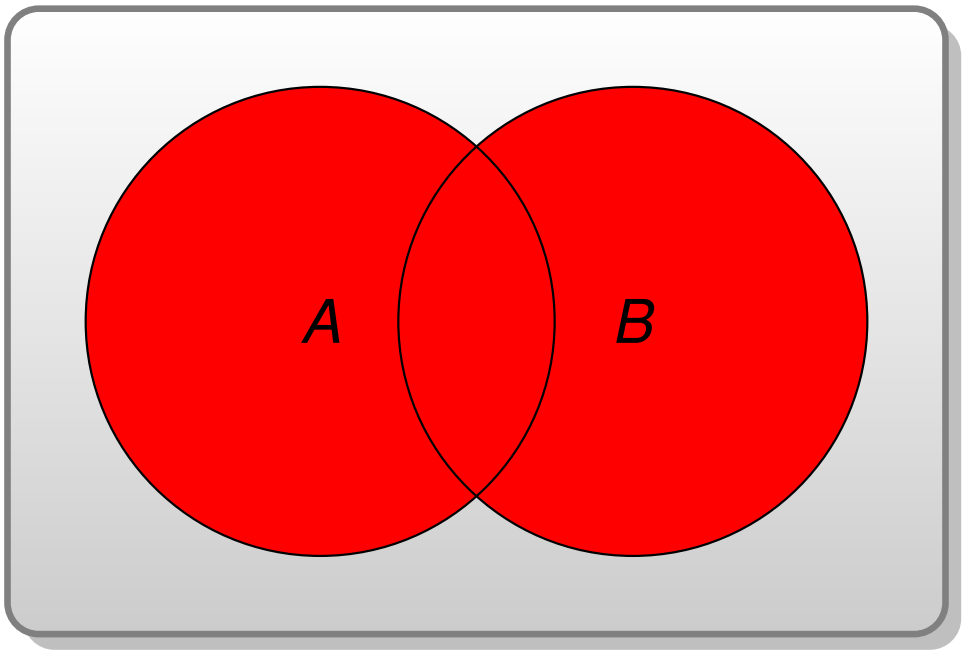
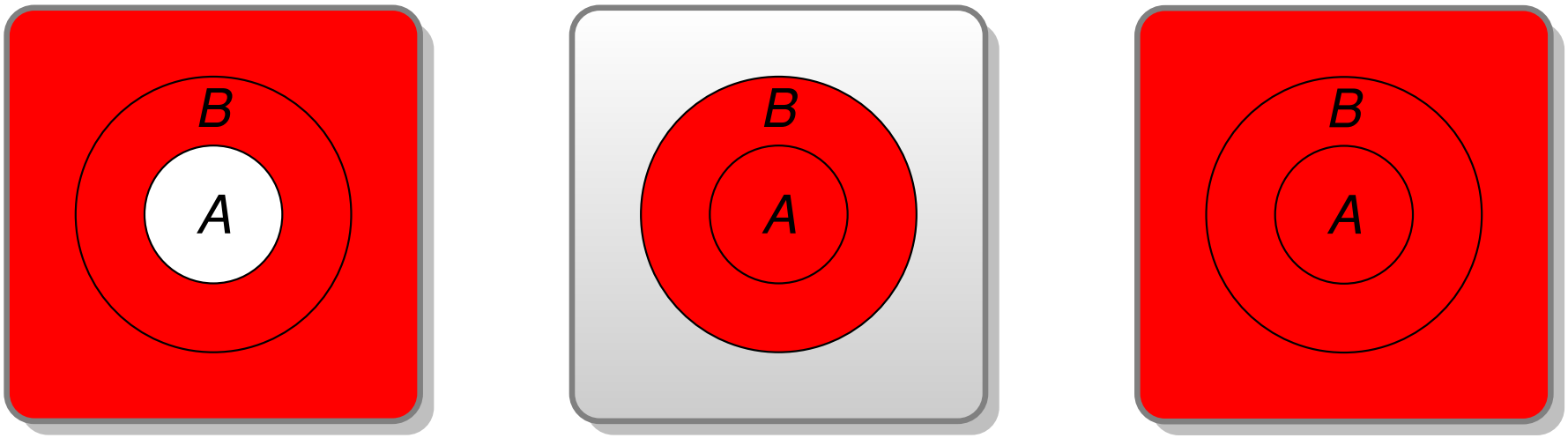


Diagramme de Venn: A $\Rightarrow$ B, $\overline{A}$ ou B

Imaginez: $A \equiv$ est un corbeau, et $B \equiv$ est noir

$\overline{A}$ B $\overline{A}$ ou B



Implication

Exemple de $A \Rightarrow B$ est équivalent à $\overline{A}$ ou B

Je suis étudiant de CSF $\Rightarrow$ J'ai suivi le cours BSL
est équivalent à
Je ne suis pas un étudiant de CSF OU j'ai suivi le cours BSL

Implication

Exemple de $A \Rightarrow B$ est équivalent à $\overline{A}$ ou B

Je suis étudiant de CSF $\Rightarrow$ J'ai suivi le cours BSL
est équivalent à
Je ne suis pas un étudiant de CSF OU j'ai suivi le cours BSL

- Bizarre, non?
- Attention, en logique il s'agit d'un OU non exclusif, alors que dans le langage courant nous utilisons plutôt le OU exclusif

Implication

Exemple de $A \Rightarrow B$ est équivalent à $\overline{A}$ ou B

Je suis étudiant de CSF $\Rightarrow$ J'ai suivi le cours BSL
est équivalent à
Je ne suis pas un étudiant de CSF OU j'ai suivi le cours BSL

- Bizarre, non?
- Attention, en logique il s'agit d'un OU non exclusif, alors que dans le langage courant nous utilisons plutôt le OU exclusif

De Morgan: $\overline{A} + B = \overline{A\overline{B}}$

Implication

Exemple de $A \Rightarrow B$ est équivalent à $\overline{A}$ ou B

Je suis étudiant de CSF $\Rightarrow$ J'ai suivi le cours BSL
est équivalent à
Je ne suis pas un étudiant de CSF OU j'ai suivi le cours BSL

- Bizarre, non?
- Attention, en logique il s'agit d'un OU non exclusif, alors que dans le langage courant nous utilisons plutôt le OU exclusif

De Morgan: $\overline{A} + B = \overline{A\overline{B}}$

Donc: $A \Rightarrow B$ est équivalent à $\overline{A\overline{B}}$
Je suis étudiant de CSF $\Rightarrow$ J'ai suivi le cours BSL
est équivalent à
Il est impossible que je sois un étudiant de CSF et que je n'aie pas suivi le cours BSL

Schémas de déduction élémentaires de la logique propositionnelle (1)

- Γ représente une liste quelconque d'hypothèses
- $\perp$ représente une proposition fausse

$\dfrac{}{\Gamma; A \vdash A}$	Vérité d'une hypothèse
$\dfrac{\Gamma \vdash B}{\Gamma; A \vdash B}$	Vérité a fortiori
$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \text{ et } B}$	Introduction de *et*
$\dfrac{\Gamma \vdash A \text{ et } B}{\Gamma \vdash A} \quad \dfrac{\Gamma \vdash A \text{ et } B}{\Gamma \vdash B}$	Elimination de *et*

Schémas de déduction élémentaires de la logique propositionnelle (2)

$\dfrac{\Gamma \vdash A}{\Gamma \vdash A \text{ ou } B} \quad \dfrac{\Gamma \vdash B}{\Gamma \vdash A \text{ ou } B}$	Introduction de *ou*
$\dfrac{\Gamma \vdash A \text{ ou } B \quad \Gamma; A \vdash C \quad \Gamma; B \vdash C}{\Gamma \vdash C}$	Disjonction des cas
$\dfrac{\Gamma; A \vdash B}{\Gamma \vdash A \Rightarrow B}$	Introduction de $\Rightarrow$
$\dfrac{\Gamma \vdash A \quad \Gamma \vdash A \Rightarrow B}{\Gamma \vdash B}$	Modus Ponens

Schémas de déduction élémentaires de la logique propositionnelle (3)

$\dfrac{\Gamma \vdash A \quad \Gamma \vdash \text{non}A}{\Gamma \vdash \bot}$	Introduction de $\bot$
$\dfrac{\Gamma \vdash \bot}{\Gamma \vdash B}$	Ex falso quodlibet
$\dfrac{\Gamma; A \vdash \bot}{\Gamma \vdash \text{non}A}$	Réduction à l'absurde J
$\dfrac{\Gamma; \text{non}A \vdash \bot}{\Gamma \vdash A}$	Réduction à l'absurde K

Schémas de déduction élémentaires de la logique propositionnelle (4)

$\dfrac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash B \Rightarrow A}{\Gamma \vdash A \Leftrightarrow B}$	Introduction de $\Leftrightarrow$
$\dfrac{\Gamma \vdash A \Leftrightarrow B}{\Gamma \vdash A \Rightarrow B} \quad \dfrac{\Gamma \vdash A \Leftrightarrow B}{\Gamma \vdash B \Rightarrow A}$	Elimination de $\Leftrightarrow$

Vérification formelle: Property checking

Un détecteur de flanc montant

```
entity Detector is
port(
    Clk_i       : in  std_logic;
    Input_i     : in  std_logic;
    Detection_o : out std_logic
);
end Detector;

architecture behave of Detector is
    signal reg_s: std_logic;
begin
    process(clk_i)
    begin
        if rising_edge(clk_i) then
            reg_s<=Input_i;
        end if;
    end process;

    Detection_o<=Input_i and not(reg_s);
end behave;
```

Vérification formelle: Property checking

Propriété à vérifier

Si $A = 1$ alors au cycle d'horloge suivant: $D = 0$

Vérification formelle: Property checking

Propriété à vérifier

Si $A = 1$ alors au cycle d'horloge suivant: $D = 0$
$\Leftrightarrow (A_{t-1} = 1) \Rightarrow D_t = 0$

Vérification formelle: Property checking

Propriété à vérifier

Si $A = 1$ alors au cycle d'horloge suivant: $D = 0$

$\Leftrightarrow (A_{t-1} = 1) \Rightarrow D_t = 0$

$\Leftrightarrow A_{t-1} \Rightarrow \overline{D_t}$

Vérification formelle: Property checking

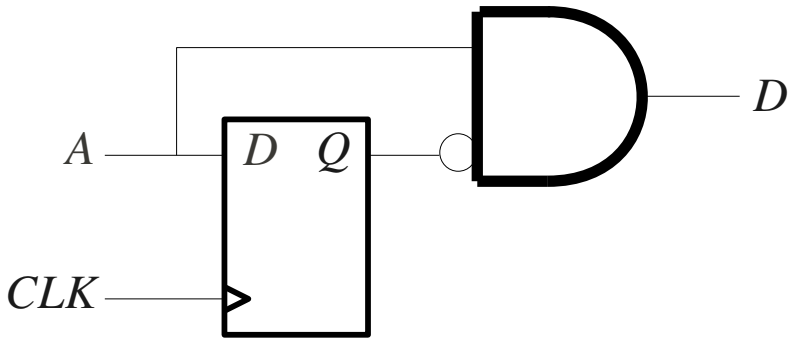
Propriété à vérifier

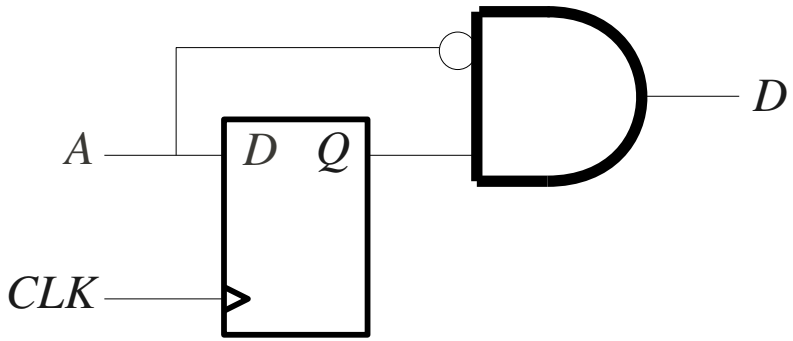
Si $A = 1$ alors au cycle d'horloge suivant: $D = 0$

$\Leftrightarrow (A_{t-1} = 1) \Rightarrow D_t = 0$

$\Leftrightarrow A_{t-1} \Rightarrow \overline{D_t}$

$\Leftrightarrow \overline{A_{t-1}} + \overline{D_t} = 1$

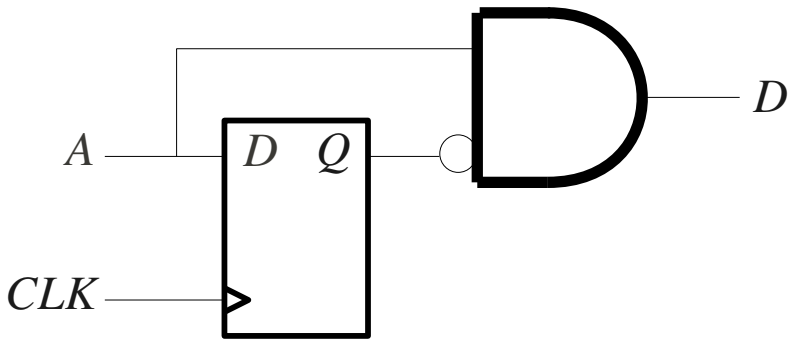




- Quel design est le bon?

Vérification formelle: Property checking

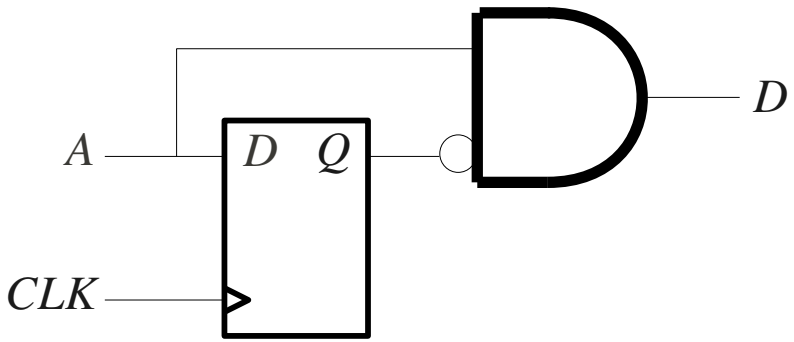
- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$



  - $D_t = A_t \cdot \overline{A_{t-1}}$

Vérification formelle: Property checking

- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$

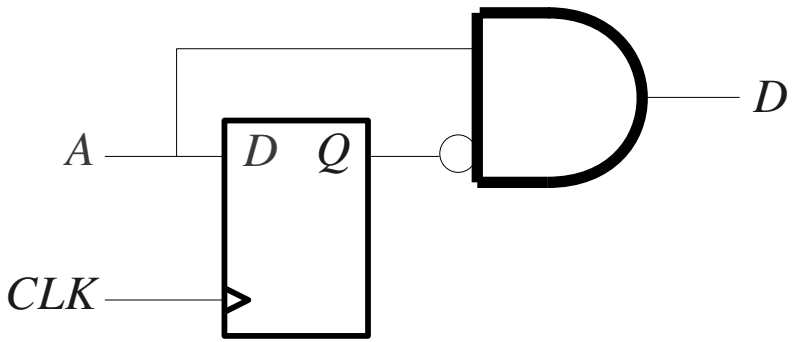


  - $D_t = A_t \cdot \overline{A_{t-1}}$
  - Donc

$$\overline{A_{t-1}} + \overline{D_t} \quad =$$

Vérification formelle: Property checking

- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$

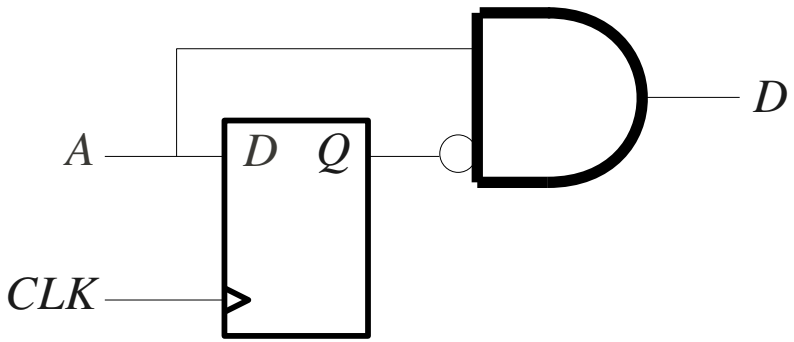


  - $D_t = A_t \cdot \overline{A_{t-1}}$
  - Donc

$$\overline{A_{t-1}} + \overline{D_t} = \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}}$$

Vérification formelle: Property checking

- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$

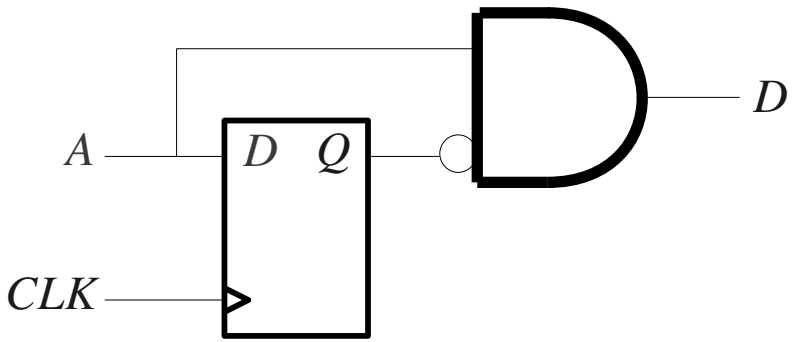


  - $D_t = A_t \cdot \overline{A_{t-1}}$
  - Donc

$$\begin{aligned} \overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}} \\ &= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \end{aligned}$$

Vérification formelle: Property checking

- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$

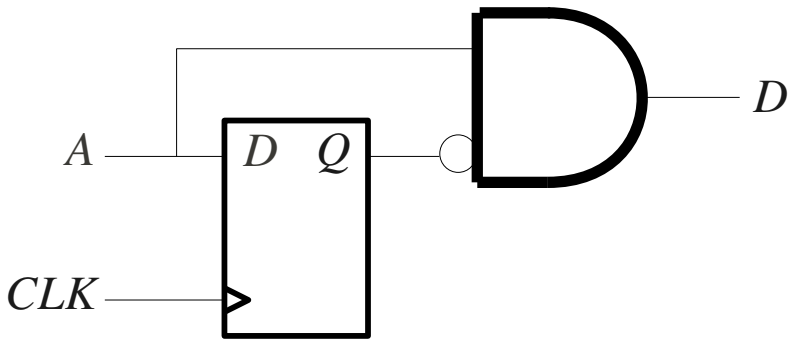


  - $D_t = A_t \cdot \overline{A_{t-1}}$
  - Donc

$$\begin{aligned}
\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}} \\
&= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \\
&= 1
\end{aligned}$$

Vérification formelle: Property checking

- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$



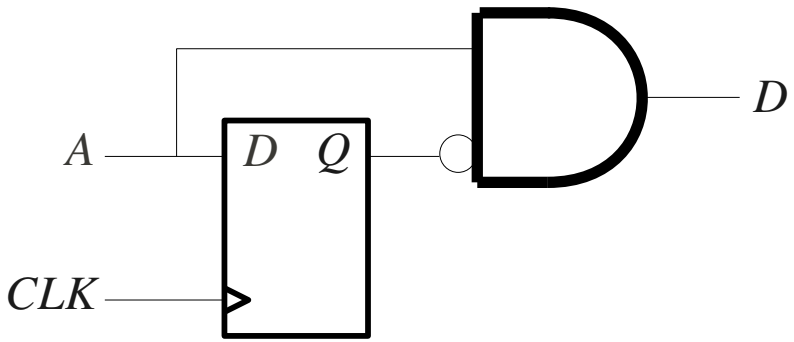
  - $D_t = A_t \cdot \overline{A_{t-1}}$
  - Donc

$$\begin{aligned} \overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}} \\ &= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \\ &= 1 \end{aligned}$$

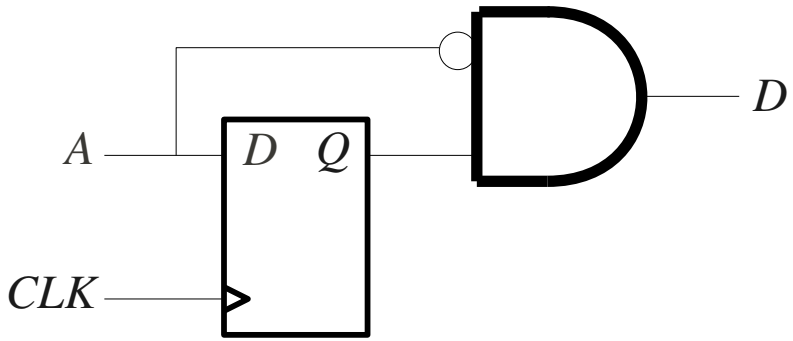
cqfd.

Vérification formelle: Property checking

- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$



- $D_t = A_t \cdot \overline{A_{t-1}}$
- Donc



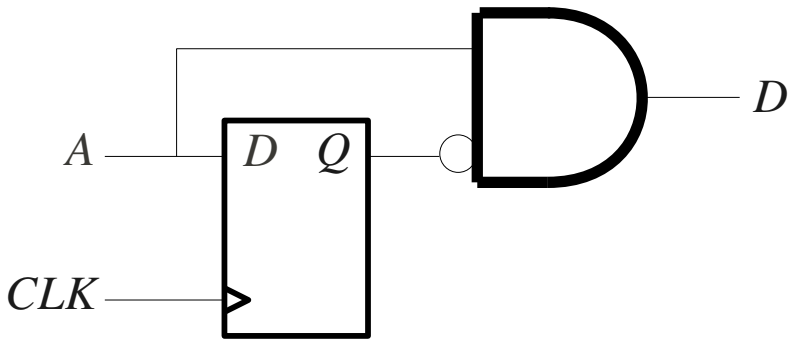
- $D_t = \overline{A_t} \cdot A_{t-1}$

$$\begin{aligned}
\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}} \\
&= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \\
&= 1
\end{aligned}$$

cqfd.

Vérification formelle: Property checking

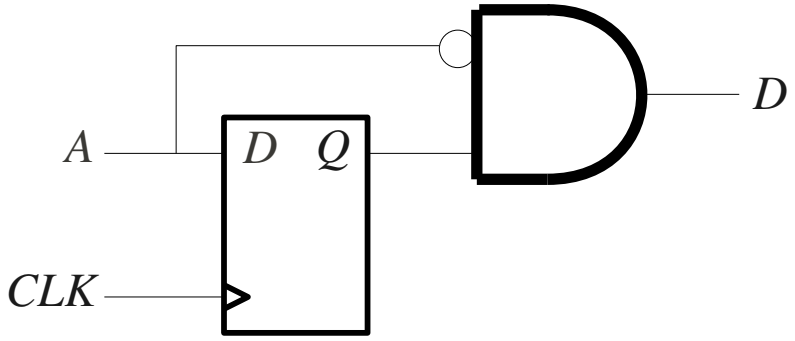
- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$



- $D_t = A_t \cdot \overline{A_{t-1}}$
- Donc

$$\begin{aligned}\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}} \\ &= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \\ &= 1\end{aligned}$$

cqfd.

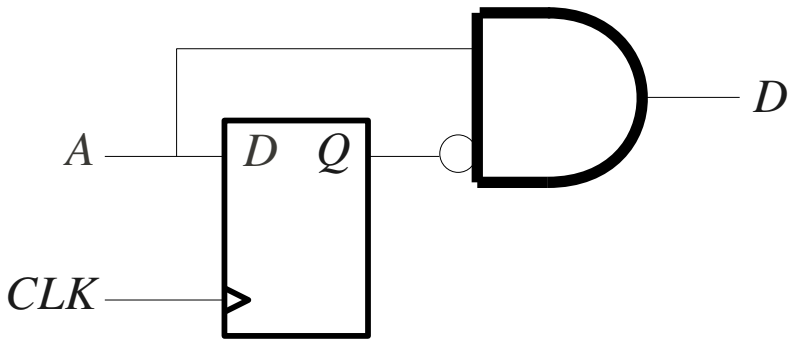


- $D_t = \overline{A_t} \cdot A_{t-1}$
- Donc

$$\overline{A_{t-1}} + \overline{D_t} =$$

Vérification formelle: Property checking

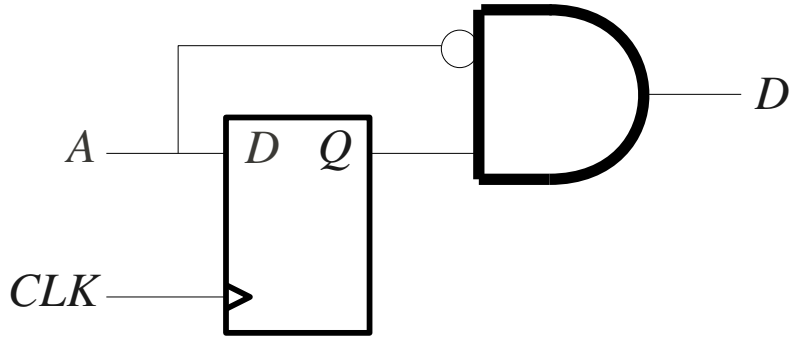
- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$



- $D_t = A_t \cdot \overline{A_{t-1}}$
- Donc

$$\begin{aligned}\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}} \\ &= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \\ &= 1\end{aligned}$$

cqfd.

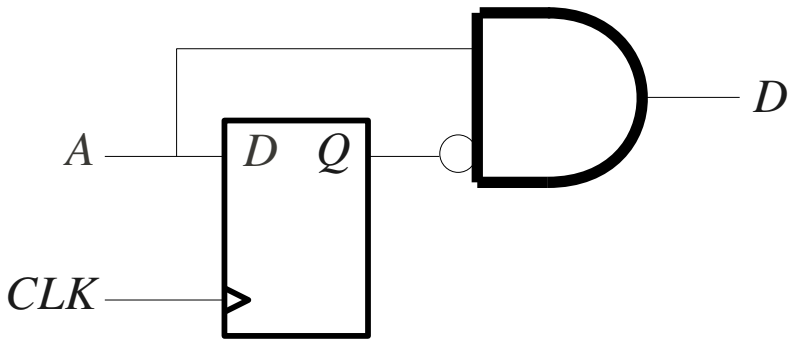


- $D_t = \overline{A_t} \cdot A_{t-1}$
- Donc

$$\overline{A_{t-1}} + \overline{D_t} = \overline{A_{t-1}} + \overline{\overline{A_t} \cdot A_{t-1}}$$

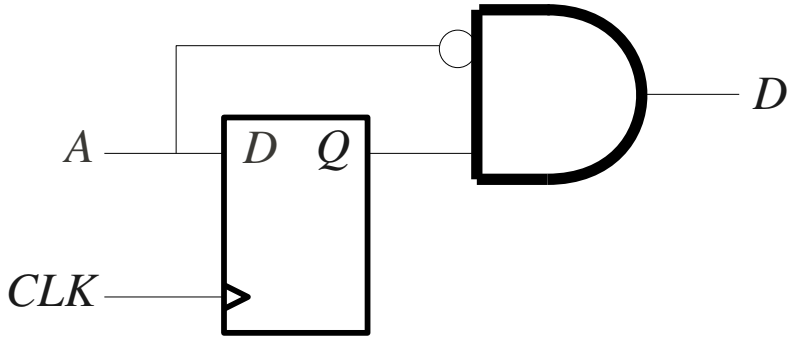
Vérification formelle: Property checking

- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$



- $D_t = A_t \cdot \overline{A_{t-1}}$
- Donc

$$\begin{aligned}\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}} \\ &= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \\ &= 1\end{aligned}$$



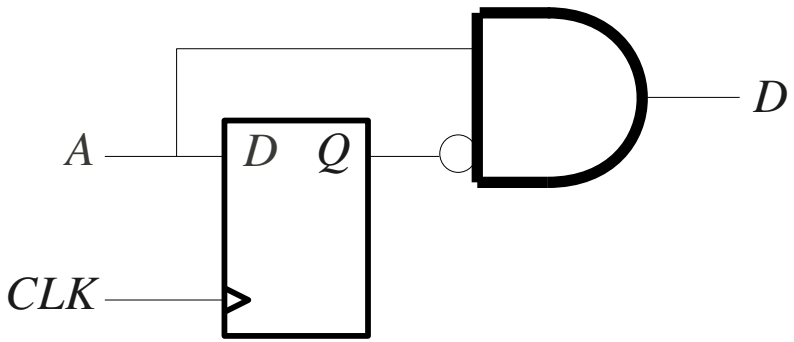
- $D_t = \overline{A_t} \cdot A_{t-1}$
- Donc

$$\begin{aligned}\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{\overline{A_t} \cdot A_{t-1}} \\ &= \overline{A_{t-1}} + A_t + \overline{A_{t-1}}\end{aligned}$$

cqfd.

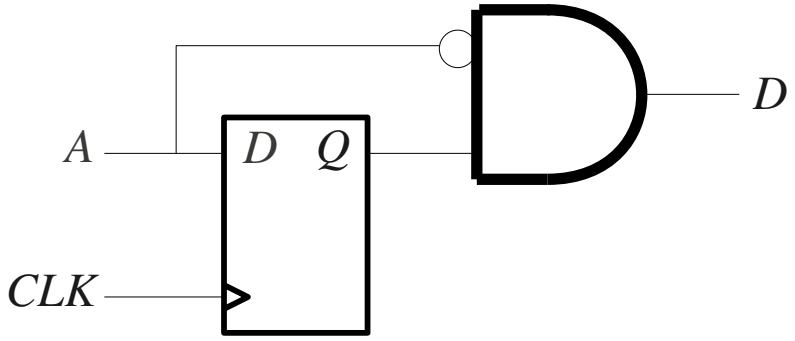
Vérification formelle: Property checking

- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$



- $D_t = A_t \cdot \overline{A_{t-1}}$
- Donc

$$\begin{aligned}\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}} \\ &= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \\ &= 1\end{aligned}$$



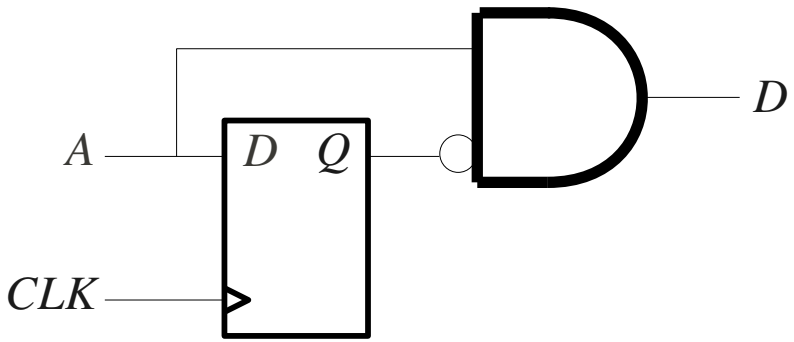
- $D_t = \overline{A_t} \cdot A_{t-1}$
- Donc

$$\begin{aligned}\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{\overline{A_t} \cdot A_{t-1}} \\ &= \overline{A_{t-1}} + A_t + \overline{A_{t-1}} \\ &= \overline{A_{t-1}} + A_t \neq 1\end{aligned}$$

cqfd.

Vérification formelle: Property checking

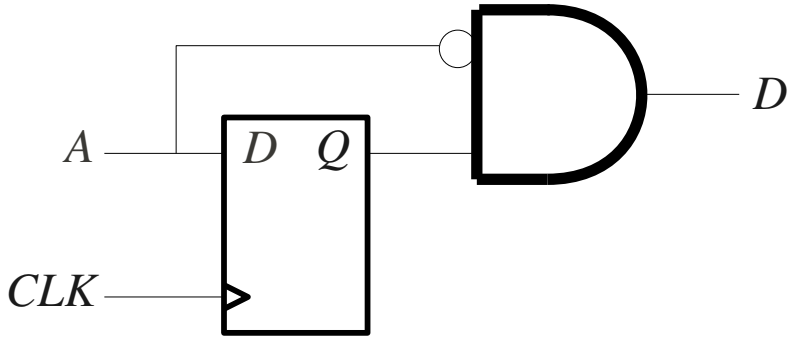
- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$



- $D_t = A_t \cdot \overline{A_{t-1}}$
- Donc

$$\begin{aligned}\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t \cdot \overline{A_{t-1}}} \\ &= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \\ &= 1\end{aligned}$$

cqfd.



- $D_t = \overline{A_t} \cdot A_{t-1}$
- Donc

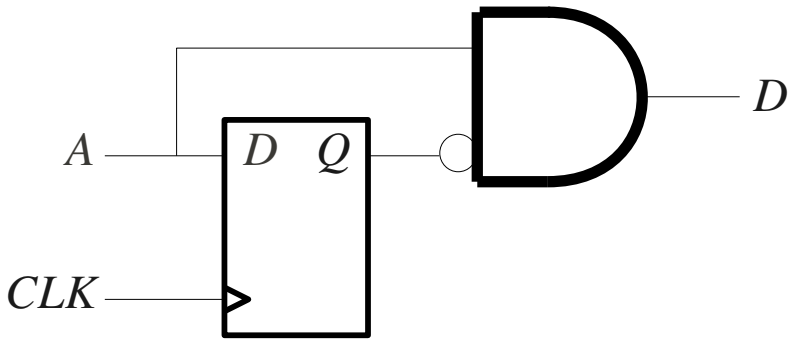
$$\begin{aligned}\overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{\overline{A_t} \cdot A_{t-1}} \\ &= \overline{A_{t-1}} + A_t + \overline{A_{t-1}} \\ &= \overline{A_{t-1}} + A_t \neq 1\end{aligned}$$

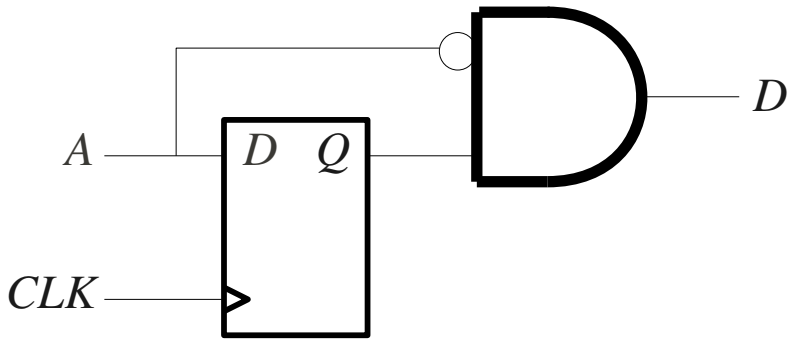
⚠ Propriété non vérifiée

Vérification formelle: Property checking

Propriété à vérifier

$$\overline{A_{t-1}} \cdot A_t \Rightarrow D_t$$



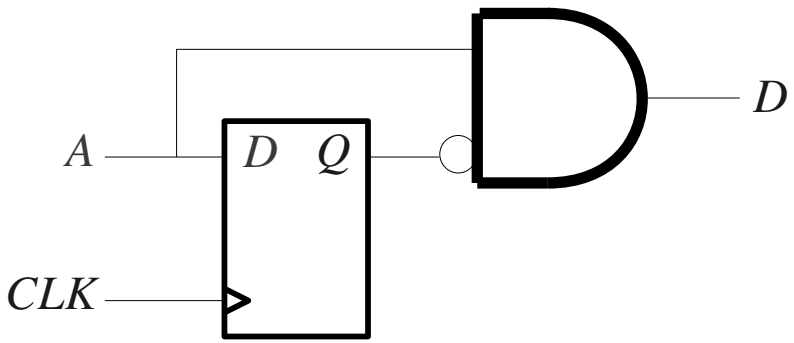


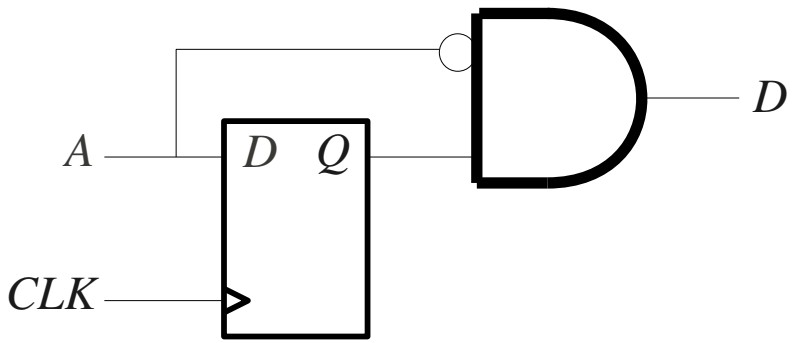
Vérification formelle: Property checking

2 Propriété suffisent

$$\overline{A_{t-1}} \cdot A_t \Rightarrow D_t$$

$$\overline{\overline{A_{t-1}} \cdot A_t} \Rightarrow \overline{D_t}$$





- Si on prouve ces deux propriétés, le système est formellement correct

Vérification formelle: Conclusion

- L'outil *QuestaFormal*, de Mentor, offre des possibilités de vérification formelle
  - Se base sur des assertions
  - Est capable de prouver le bon fonctionnement
  - Sinon offre le chronogramme menant à l'erreur
- La preuve mathématique reste la meilleure des preuves, mais...
- Non applicable pour des systèmes trop complexes
- Nécessite de spécifier le système avec des assertions
  - Nous aborderons les assertions
  - Exploitées durant la simulation
  - Et en preuve formelle!

Pour le plaisir: paradoxe de Hempel

- contrapositive de l'implication: $(A \Rightarrow B) \Leftrightarrow (\overline{B} \Rightarrow \overline{A})$
- Donc, pour prouver $A \Rightarrow B$, il suffit de prouver $\overline{B} \Rightarrow \overline{A}$
- Par la contrapositive de l'implication, la phrase "Tous les corbeaux sont noirs" est donc équivalente à "Tout ce qui n'est pas noir n'est pas un corbeau".
- Chaque fois que nous voyons un corbeau noir, cela confirme "Tous les corbeaux sont noirs".
- Donc:
- Chaque fois que nous voyons quelque chose qui n'est pas noir et n'est pas un corbeau, cela confirme que "Tous les corbeaux sont noirs"...

Vérification par simulation: approches

- Plusieurs manières d'aborder la simulation:
  - Bancs de test dirigés (Directed testbench)
    - Scénarios prévus
    - Par exemple: tests exhaustifs (!)
  - Basé sur la couverture et l'aléatoire (Coverage-driven random-based)
    - Génération de stimuli aléatoires
    - Test de couverture pour décider de la fin de la simulation
  - Vérification basée sur les assertions (assertion-based)
    - Assertions peuvent servir à une preuve formelle
    - Ou sont vérifiées par simulation
- Les outils (langages) sont plus ou moins adaptés à l'une ou l'autre approche

Objectifs de la simulation

- Vérifier le fonctionnement logique du module ou du système
- Vérifier les caractéristiques dynamiques du module ou du système (temps d'exécution, fréquence maximum, ...)
- Dans les deux cas, une preuve écrite du résultat de simulation doit être générée

Méthodologie de vérification

- Vérification exhaustive
  - Problème NP-dur
  - En général: impossibilité de tester toutes les possibilités, pour un système combinatoire
  - Pour un système séquentiel: encore pire
- Vérifications classiques
  - Test de la description (transparent)
    - Vérification de toutes les lignes de la description (taux de couverture)
  - Test du fonctionnement (boîte noire)
    - Vérification du fonctionnement du système sans tenir compte de la description interne
    - Devrait être réalisé par un autre ingénieur
- Nouvelles approches
  - Assertions, coverage-driven, ...

Méthodologie de vérification

- Il faut éviter qu'un banc de test ne détecte pas une erreur de fonctionnement ou de description
  - Il est facile de réaliser un banc de test qui ne détecte pas d'erreur...
  - Rigueur nécessaire pour la mise au point du banc de test
  - Auto-test du banc de test
    - Le banc de test modifie volontairement les sorties du design

Validation de la simulation

- En vérification dirigée, la validation est donnée par:
  - La liste des stimuli générés et les vérifications exécutées (fichier testbench)
  - Et la preuve écrite du résultat des vérifications (fenêtre log du simulateur, fichier texte, HTML, ...)
- En vérification aléatoire, la validation est donnée par:
  - La liste des scénarios testés et les vérifications exécutées (fichier testbench)
  - Le taux de couverture
  - Et la preuve écrite du résultat des vérifications (fenêtre log du simulateur, fichier texte, HTML, ...)

Vérification des timings

- La vérification des timings est compliquée avec une simulation VHDL
- Une meilleure méthodogie:
- Conception Full synchrone
  - Simulation fonctionnelle
  - Analyse statique des timings
- $\Rightarrow$ Fonctionnement garanti
  - Attention toutefois aux entrées/sorties

Manuelle vs. automatique

- Vérification manuelle
  - Forcer à la main les signaux d'entrée
  - Grands risques d'erreur
  - Difficulté à réexécuter le même scénario
  - Vérification visuelle par le développeur, par le chronogramme de simulation
  - Fastidieux (temps perdu)
- Vérification semi-automatique
  - Les signaux d'entrée sont forcés automatiquement
  - Vérification du fonctionnement par le développeur
  - Risque de mauvaise analyse par le "vérificateur"

Vérification automatique

- But: vérification la plus complète possible du système
- Résultat Go/No Go (OK, KO)
- Indication de l'instant d'une erreur
- Possibilité de stopper la simulation sur une erreur
- Indication du nombre d'erreurs total
- Permet une re-vérification complète (Go/no Go) après une modification ou une correction
- Les interactions utilisateur doivent être minimisées
  - Nécessaire pour garantir la similarité des tests effectués

Simulation automatique: scripts

- Une simulation doit pouvoir être lancée via des scripts
- Evite des erreurs de manipulation
- Simplifie la vie

Exemple: sim.do

```
# définition de la librairie de travail
vlib work
# compilation du design à tester
vcom counter.vhd
# compilation du banc de test
vcom counter_tb.vhd
# lancement de la simulation
vsim work.counter_tb
# ajout de tous les signaux au chronogramme
add wave -r *
# exécution de la simulation
run -all
```

Fonctionnement d'un banc de tests

- 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

Vérification black/white/gray box

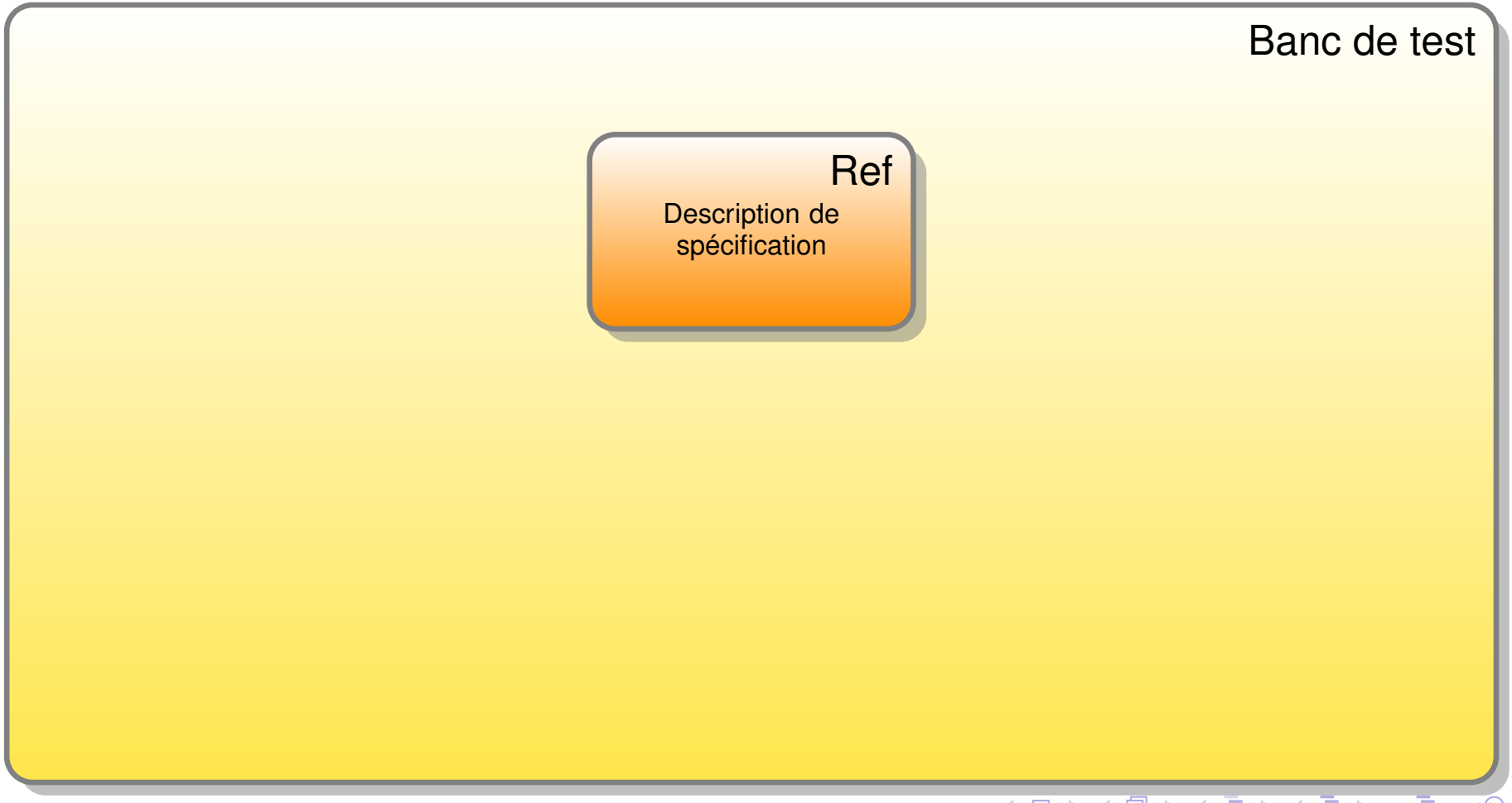
- But d'un design: que les sorties soient correctes
- Certaines erreurs du design pourraient être ignorées
  - Erreurs jamais activées
  - Erreurs qui ne se propagent pas jusqu'à la sortie
  - Erreurs qui s'annulent l'une l'autre
- Black box
  - La fonctionnalité n'est vérifiée qu'aux frontières du design
  - Typiquement comparées à un modèle de référence

Black box

- Seules les sorties sont analysées
  - Difficulté à trouver les sources d'erreurs
  - Nécessité d'avoir un modèle de référence

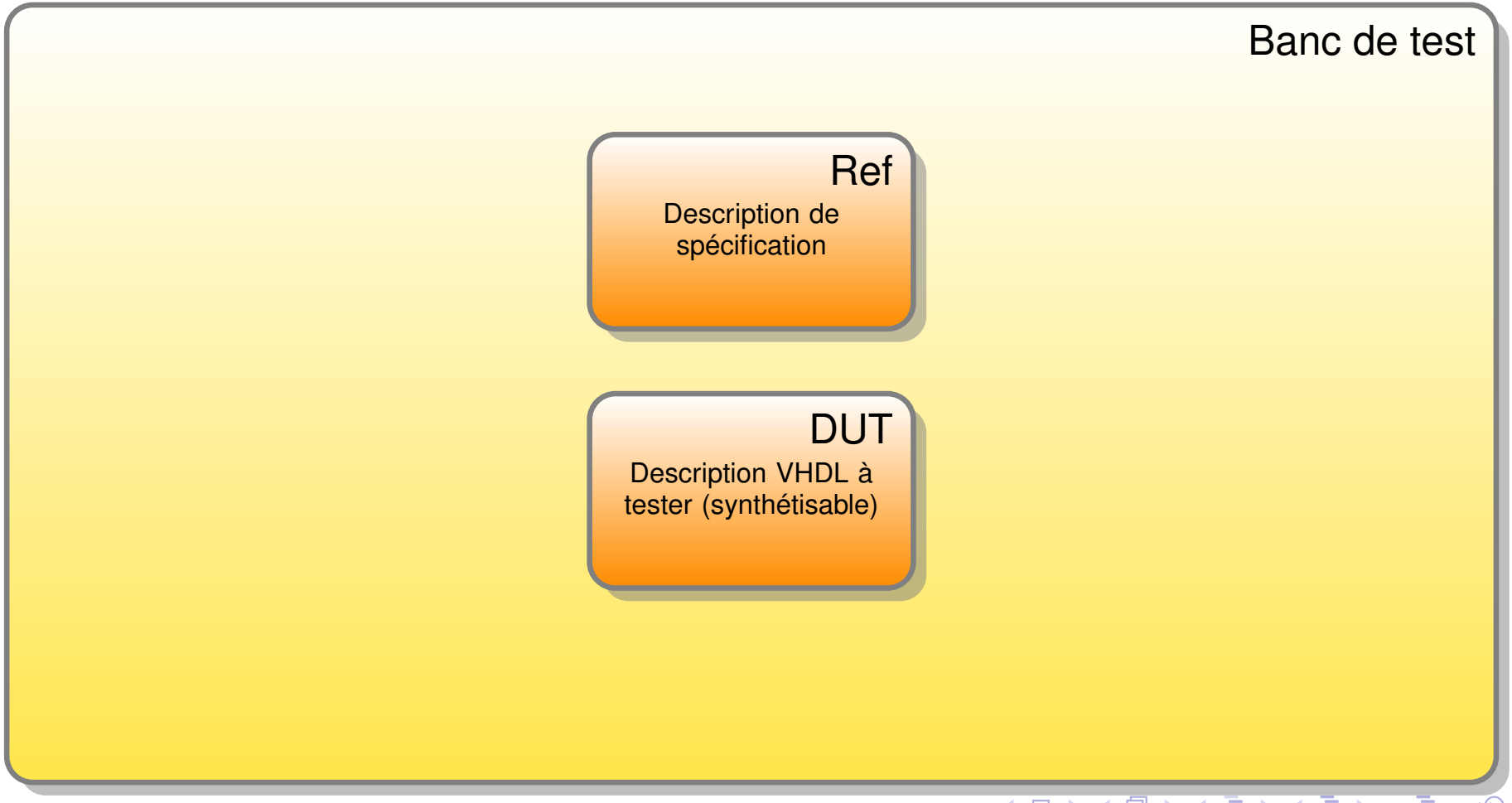
Black box

- Seules les sorties sont analysées
  - Difficulté à trouver les sources d'erreurs
  - Nécessité d'avoir un modèle de référence



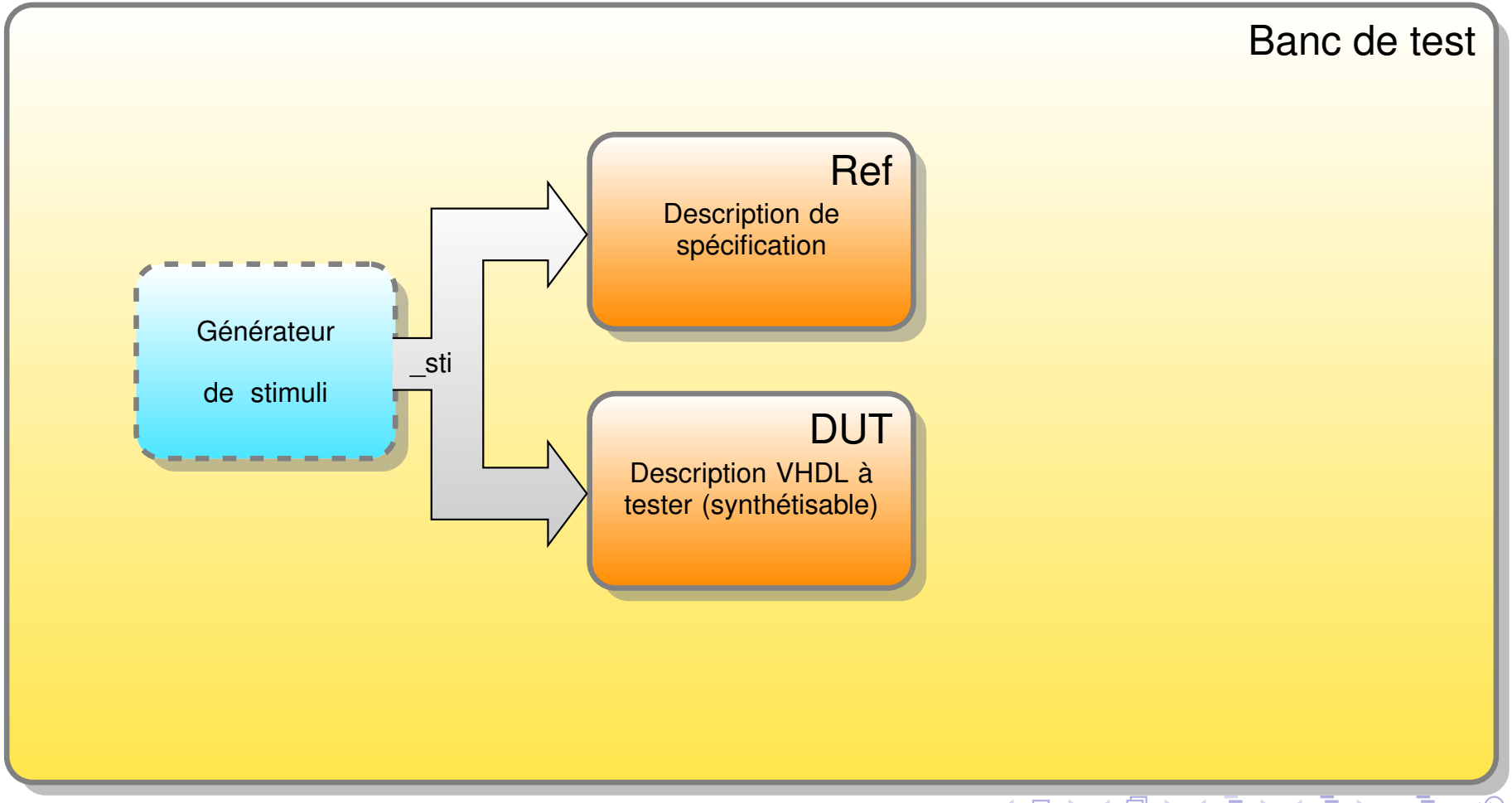
Black box

- Seules les sorties sont analysées
  - Difficulté à trouver les sources d'erreurs
  - Nécessité d'avoir un modèle de référence



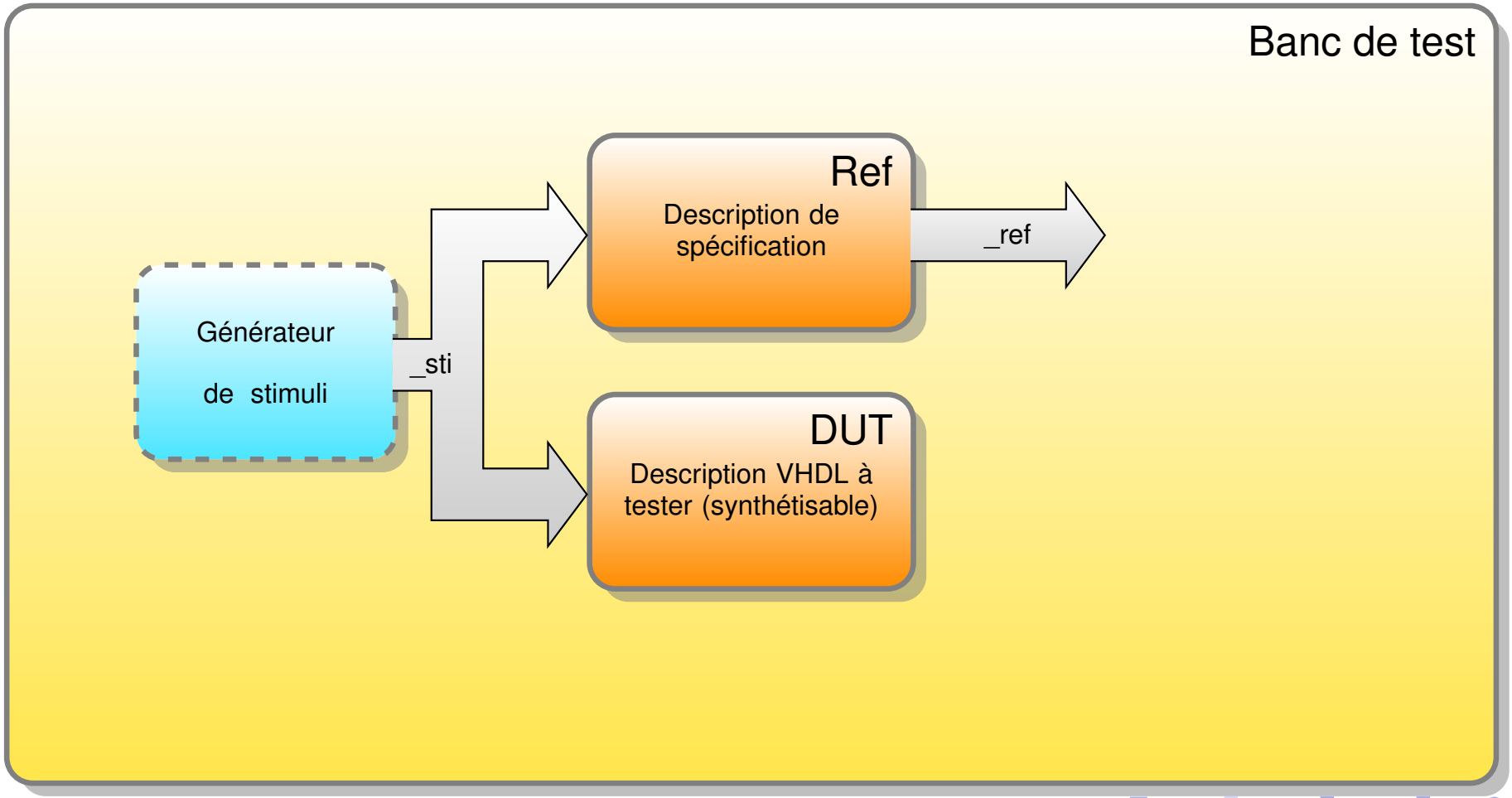
Black box

- Seules les sorties sont analysées
  - Difficulté à trouver les sources d'erreurs
  - Nécessité d'avoir un modèle de référence



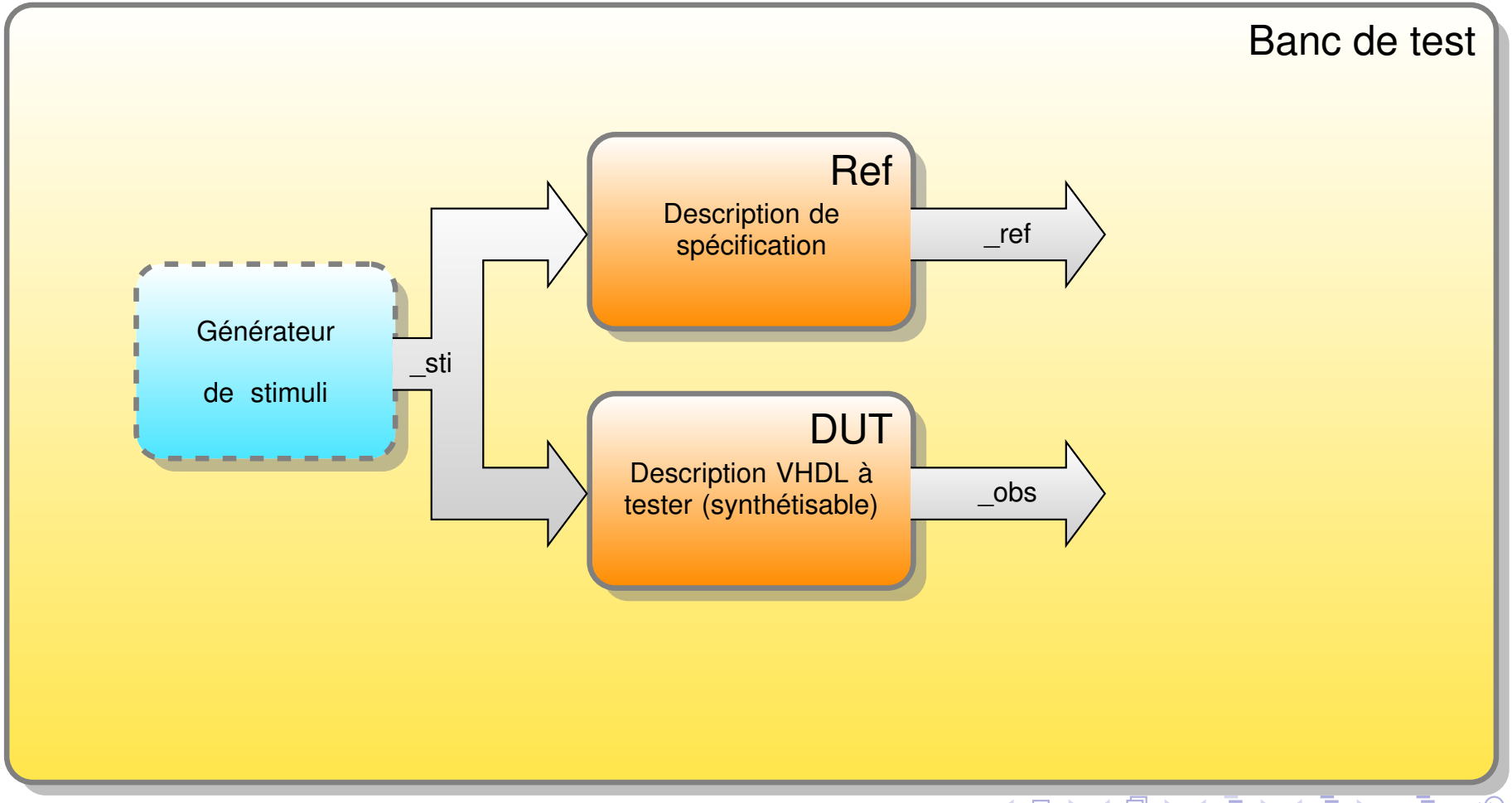
Black box

- Seules les sorties sont analysées
  - Difficulté à trouver les sources d'erreurs
  - Nécessité d'avoir un modèle de référence



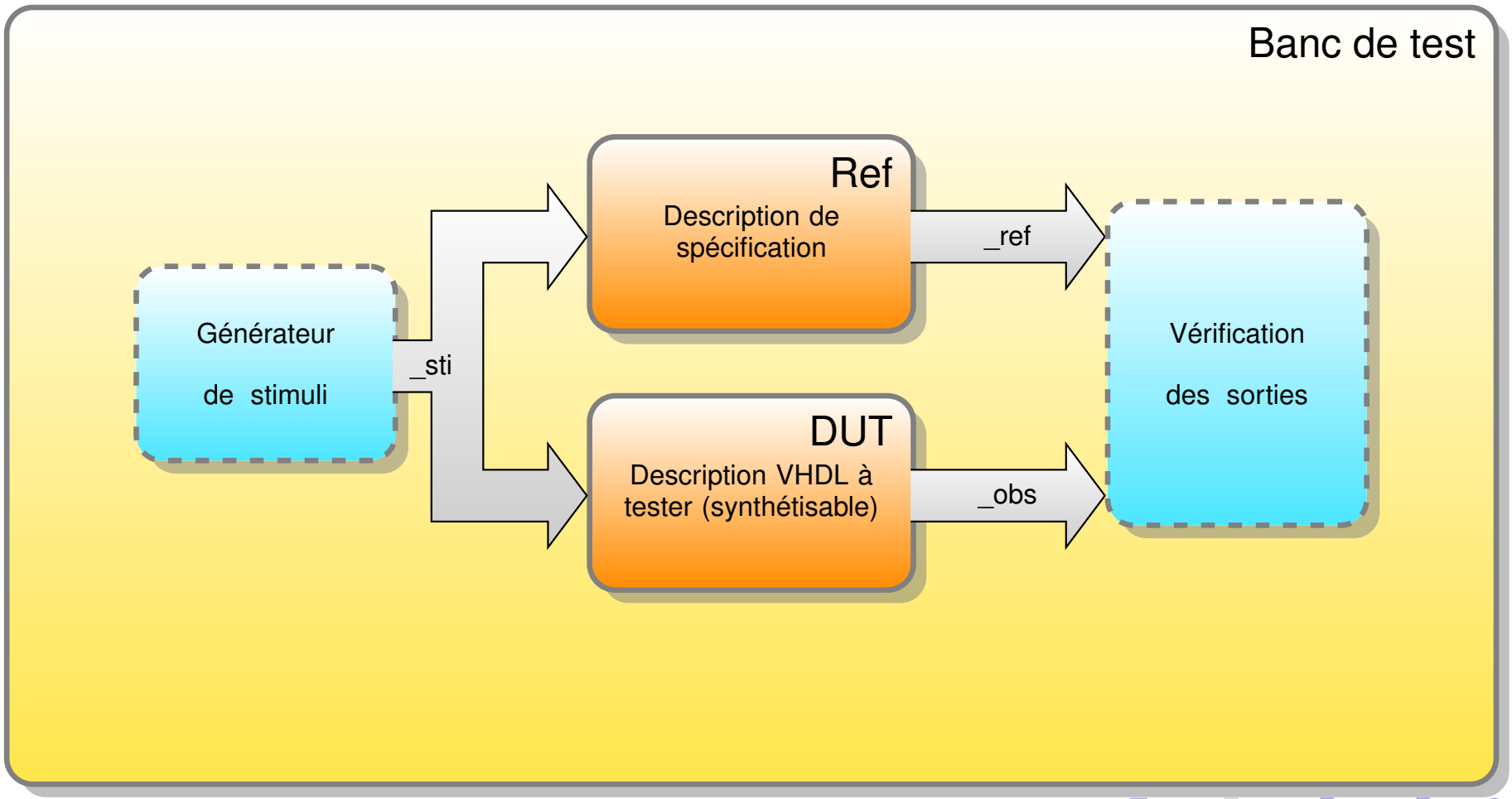
Black box

- Seules les sorties sont analysées
  - Difficulté à trouver les sources d'erreurs
  - Nécessité d'avoir un modèle de référence



Black box

- Seules les sorties sont analysées
  - Difficulté à trouver les sources d'erreurs
  - Nécessité d'avoir un modèle de référence



White box

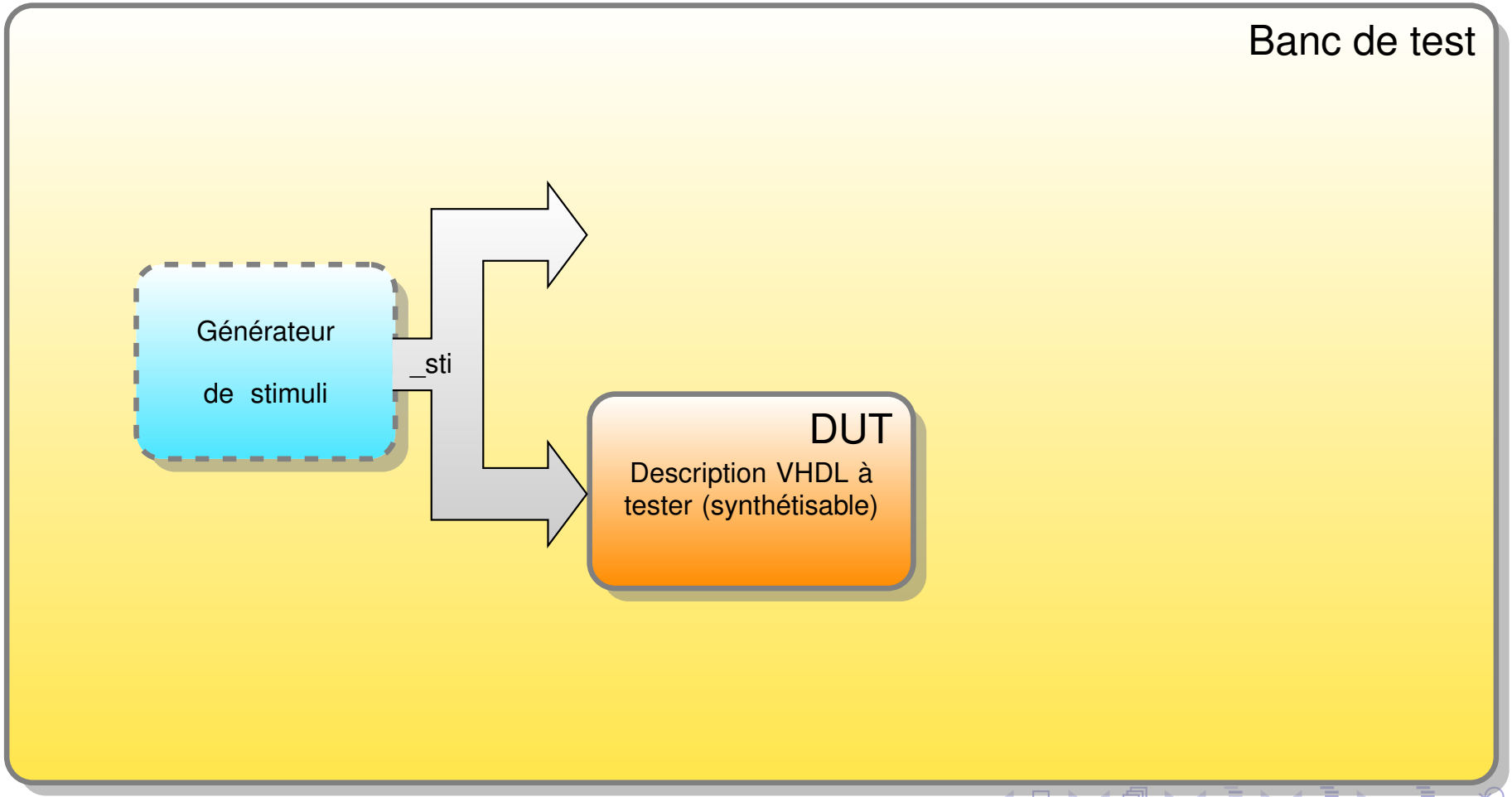
- L'intérieur du design est analysé
  - Recherche d'erreurs facilité
  - Peut être limité pour des grands systèmes

White box

- L'intérieur du design est analysé
  - Recherche d'erreurs facilité
  - Peut être limité pour des grands systèmes

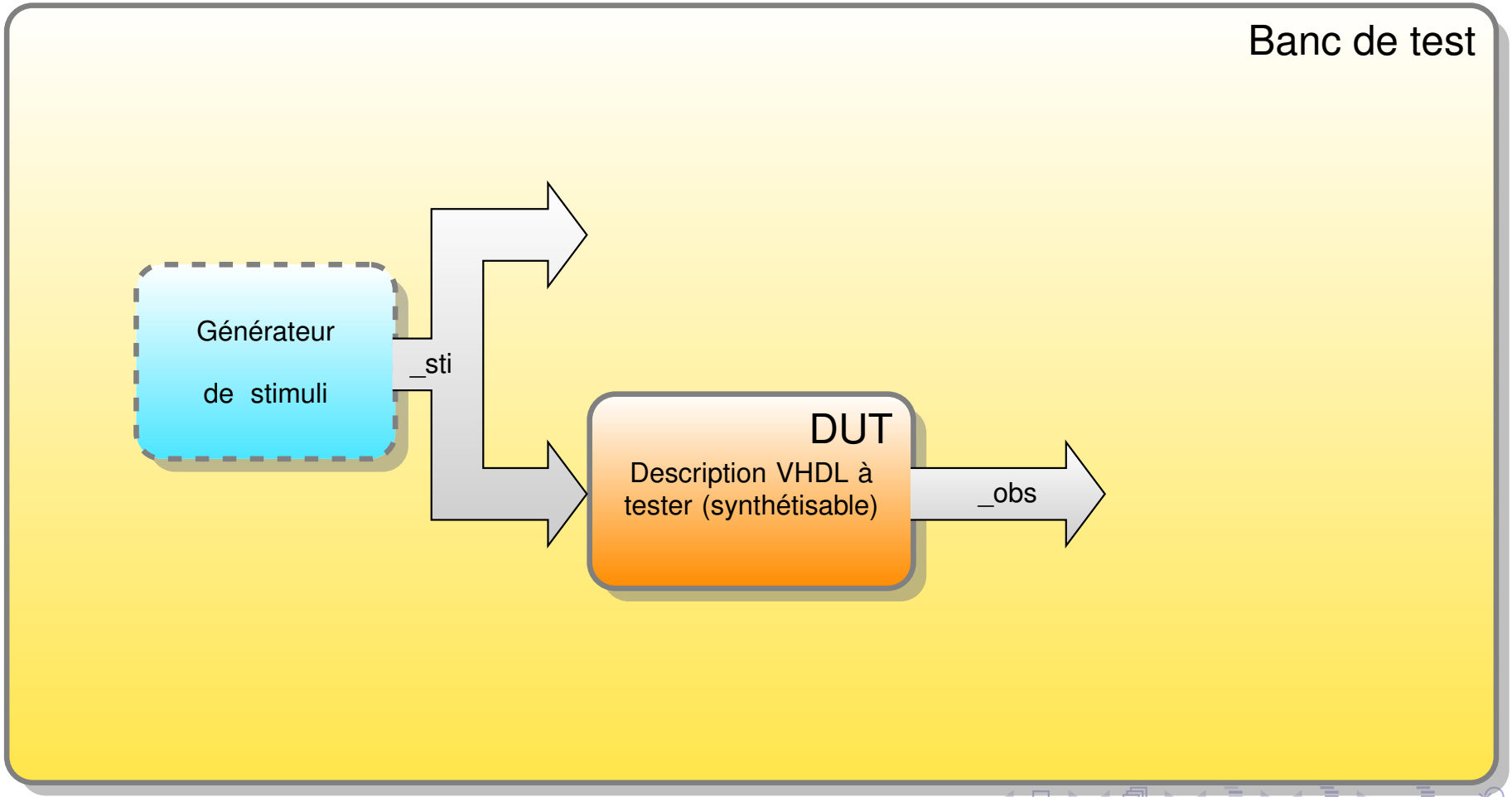
White box

- L'intérieur du design est analysé
  - Recherche d'erreurs facilité
  - Peut être limité pour des grands systèmes



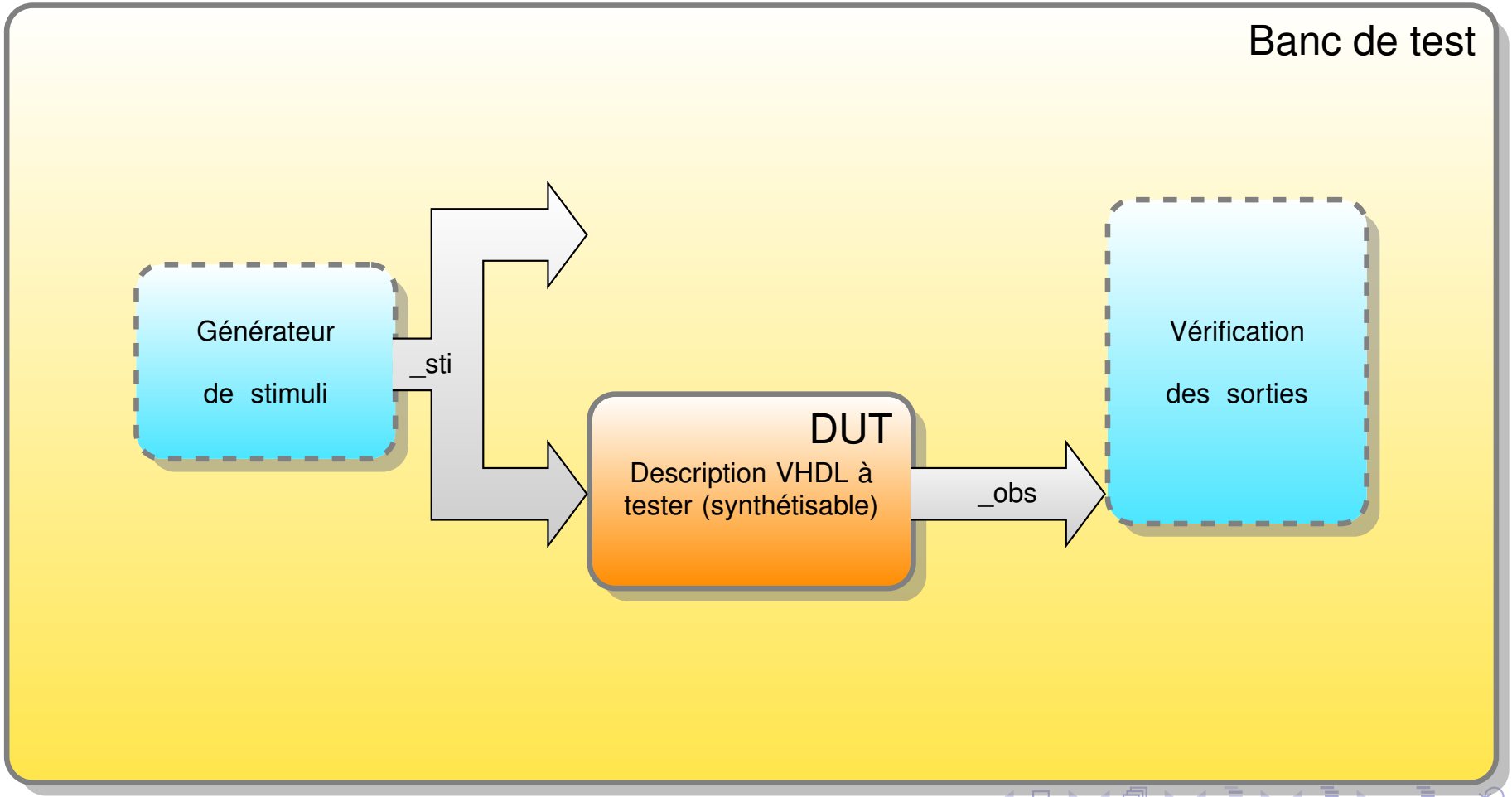
White box

- L'intérieur du design est analysé
  - Recherche d'erreurs facilité
  - Peut être limité pour des grands systèmes



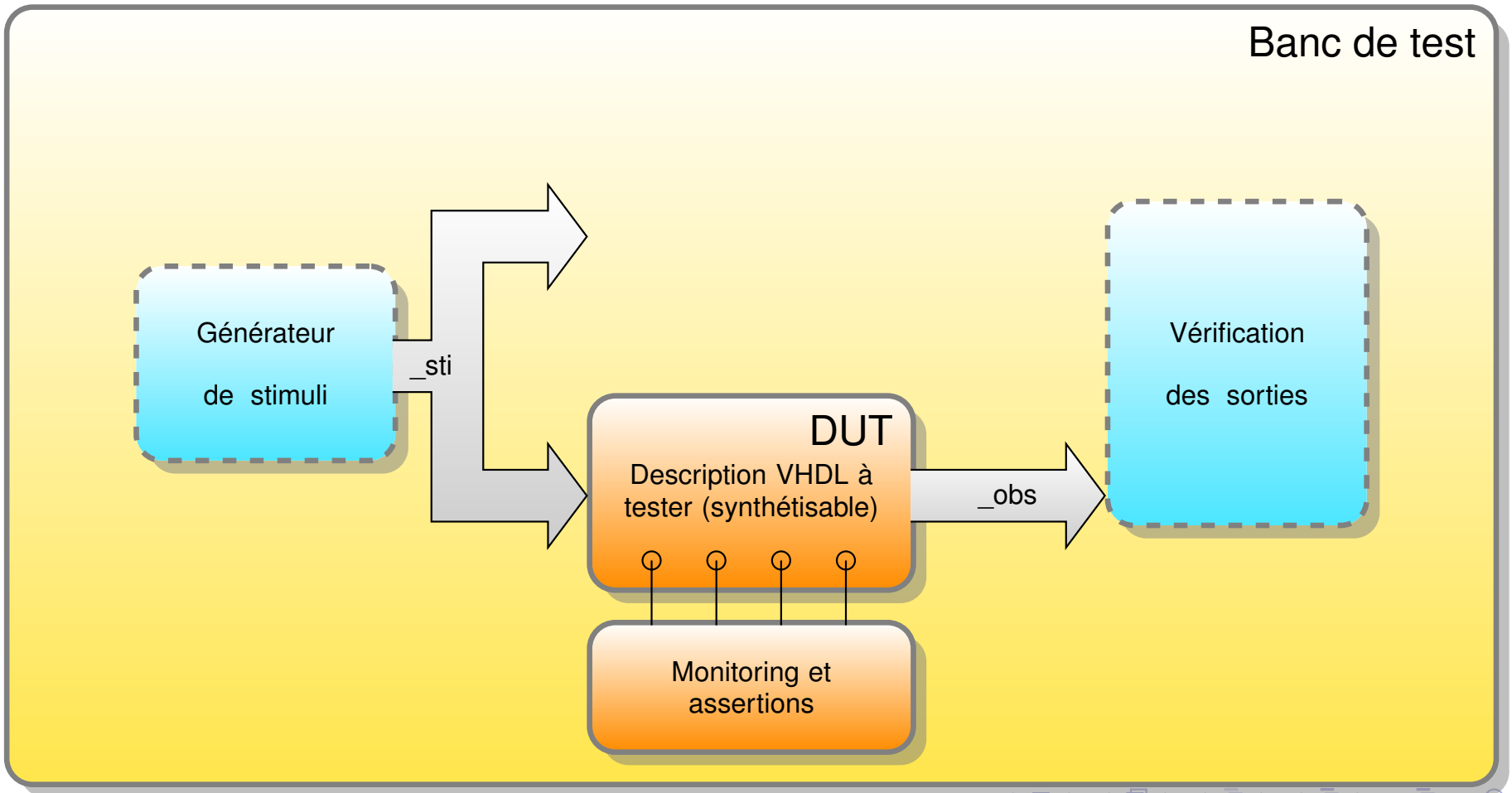
White box

- L'intérieur du design est analysé
  - Recherche d'erreurs facilité
  - Peut être limité pour des grands systèmes



White box

- L'intérieur du design est analysé
  - Recherche d'erreurs facilité
  - Peut être limité pour des grands systèmes

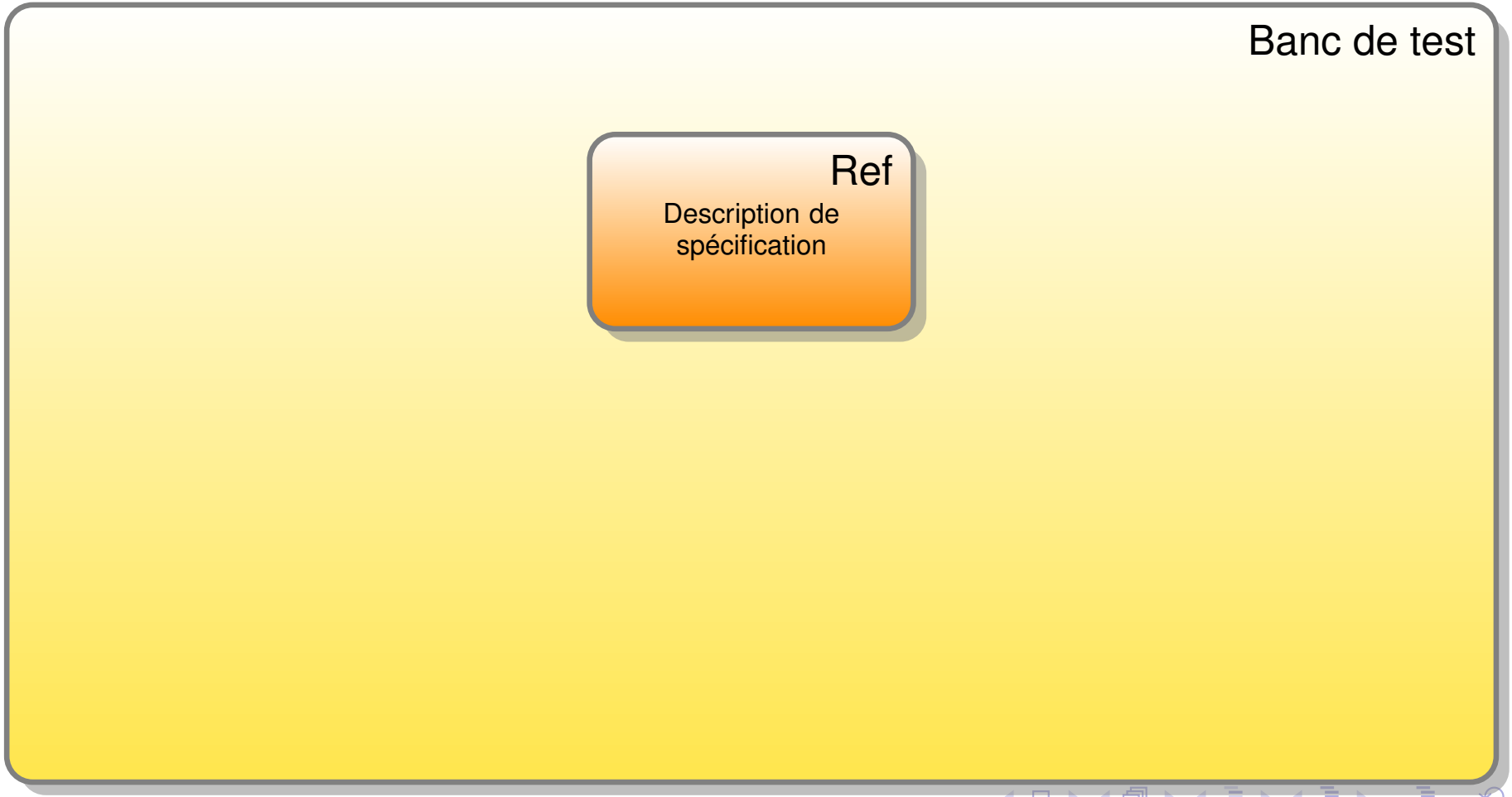


Grey box

- Les sorties et l'intérieur du design sont analysées
  - Compromis entre les deux autres approches
  - Permet de limiter la vérification des sorties par l'analyse interne

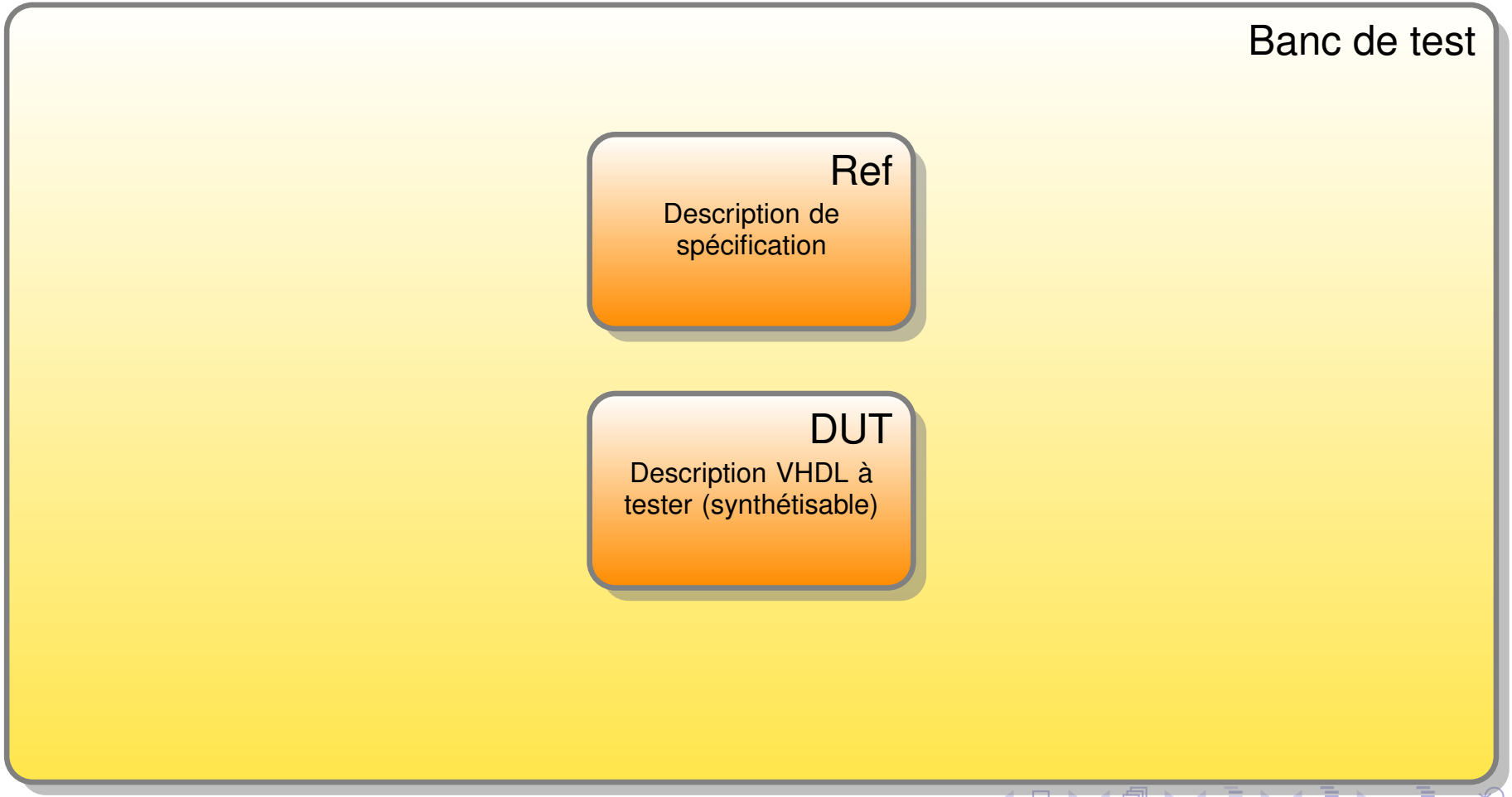
Grey box

- Les sorties et l'intérieur du design sont analysées
  - Compromis entre les deux autres approches
  - Permet de limiter la vérification des sorties par l'analyse interne



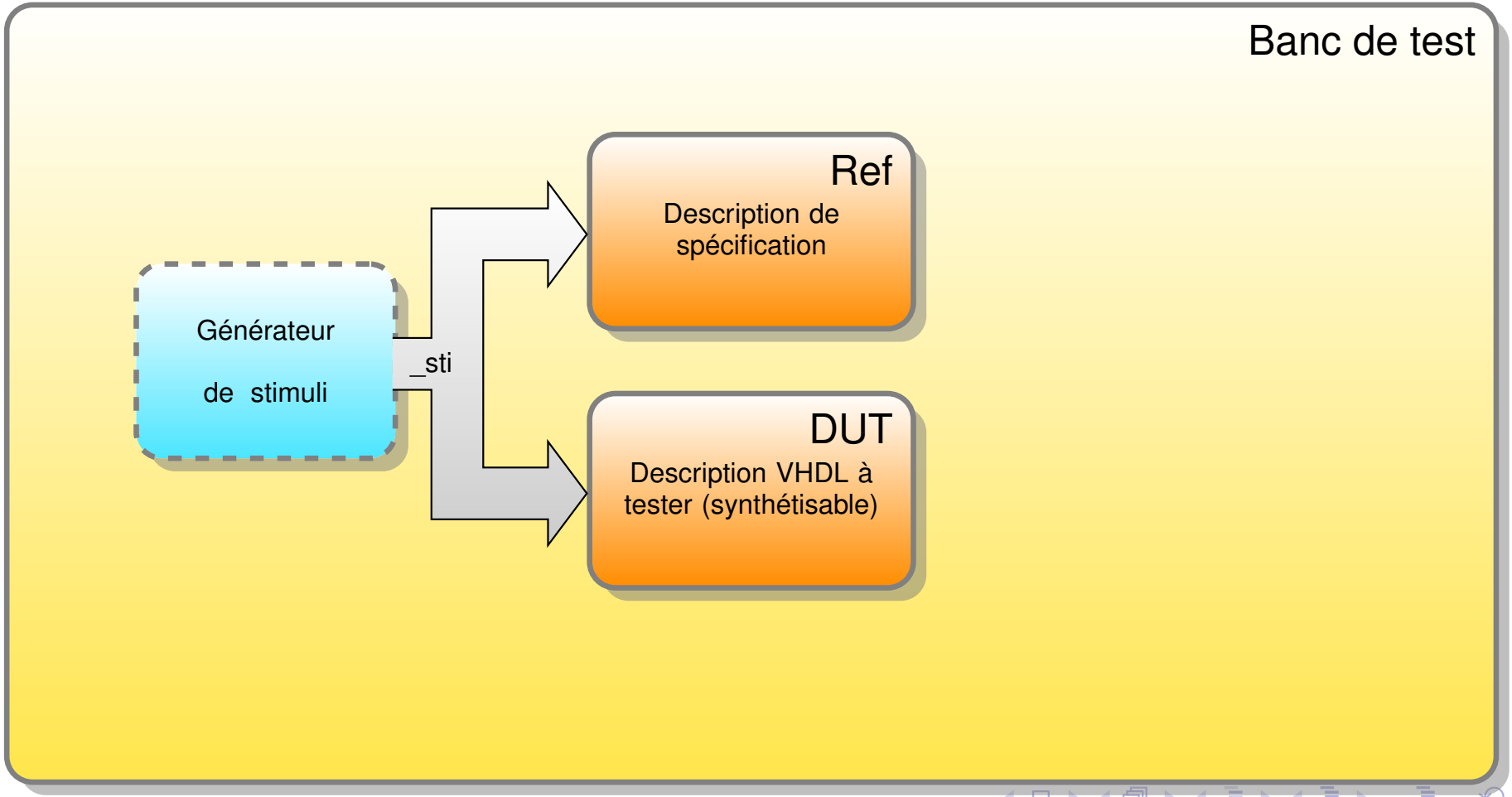
Grey box

- Les sorties et l'intérieur du design sont analysées
  - Compromis entre les deux autres approches
  - Permet de limiter la vérification des sorties par l'analyse interne



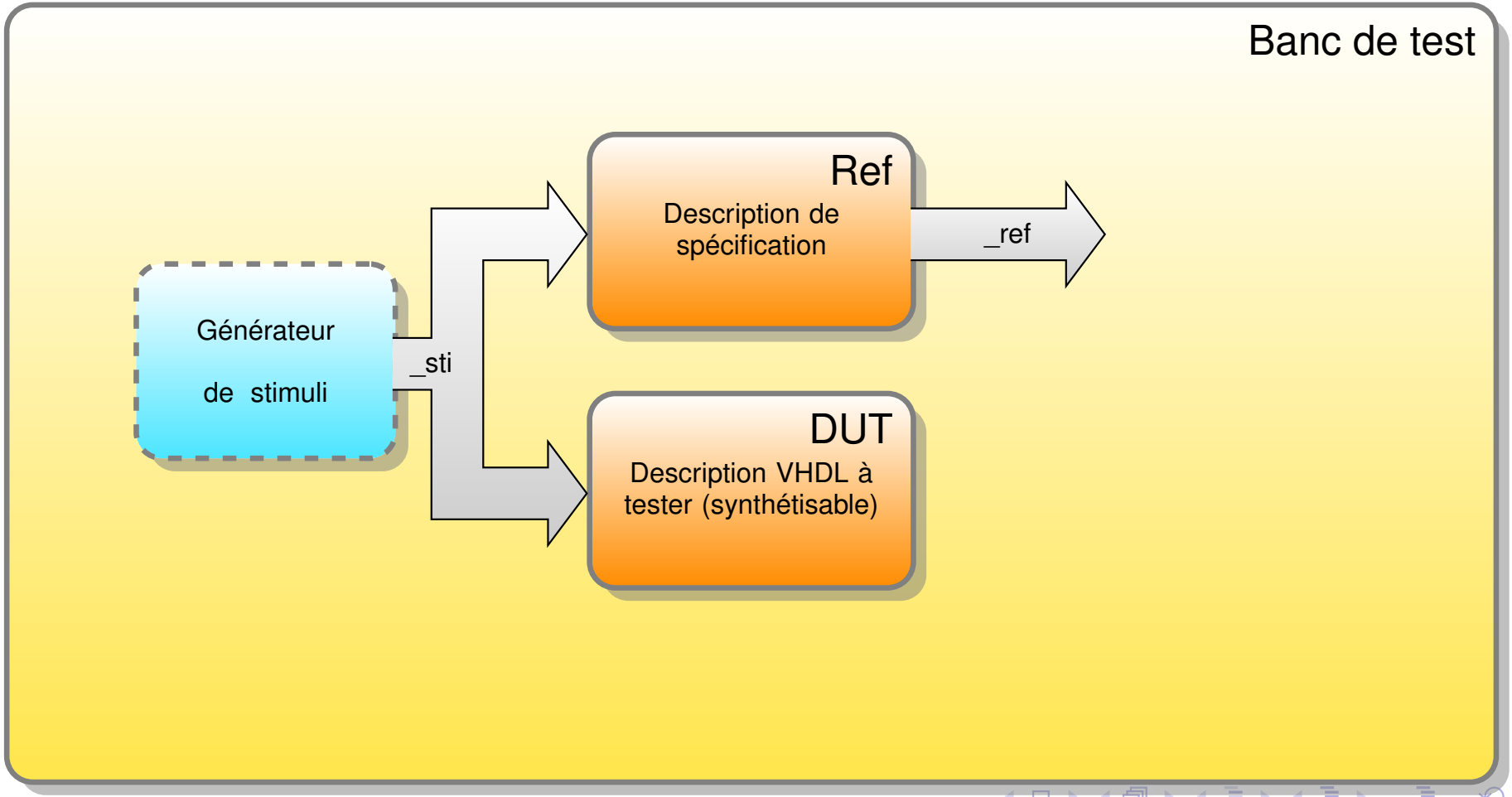
Grey box

- Les sorties et l'intérieur du design sont analysées
  - Compromis entre les deux autres approches
  - Permet de limiter la vérification des sorties par l'analyse interne



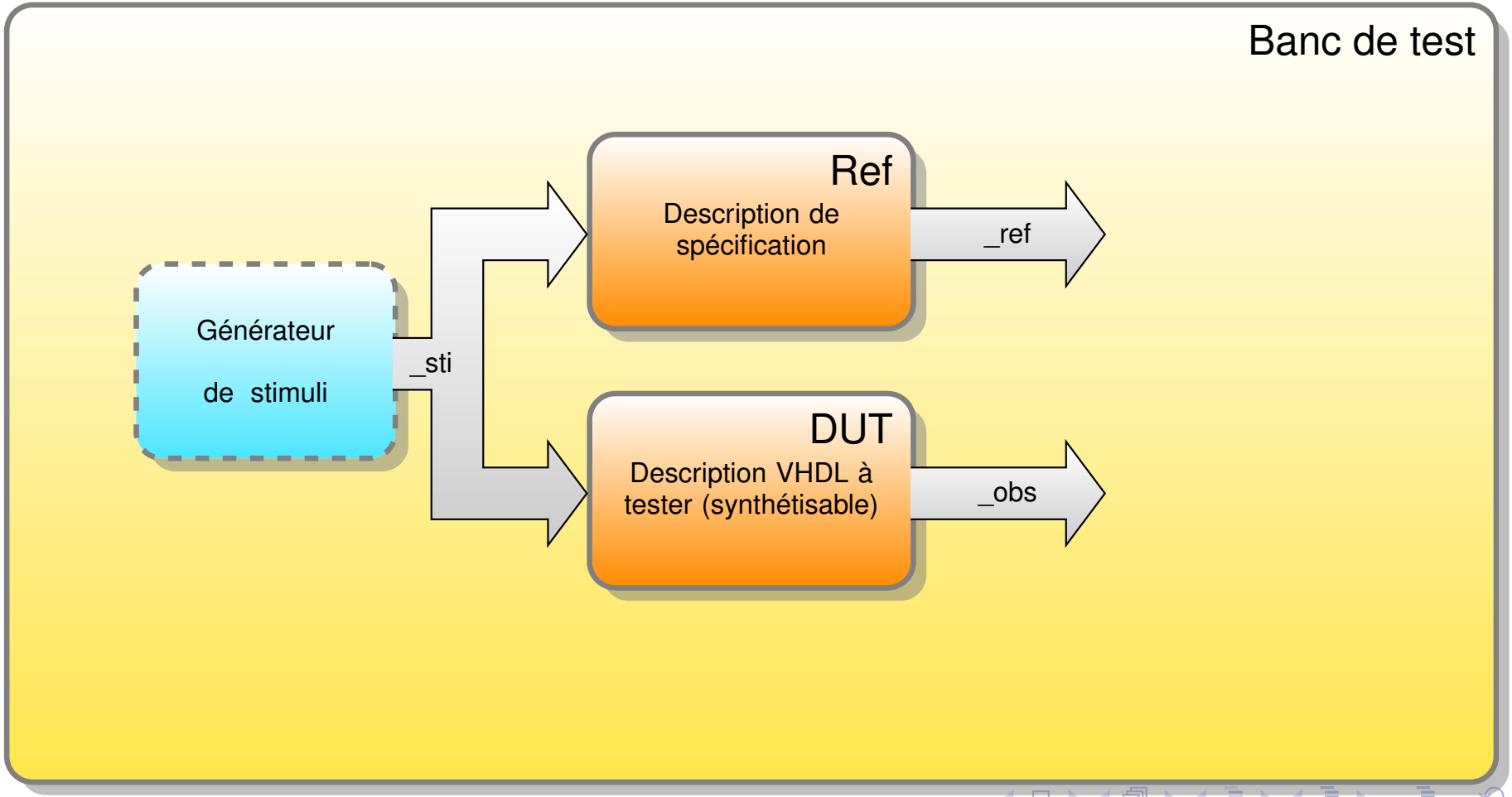
Grey box

- Les sorties et l'intérieur du design sont analysées
  - Compromis entre les deux autres approches
  - Permet de limiter la vérification des sorties par l'analyse interne



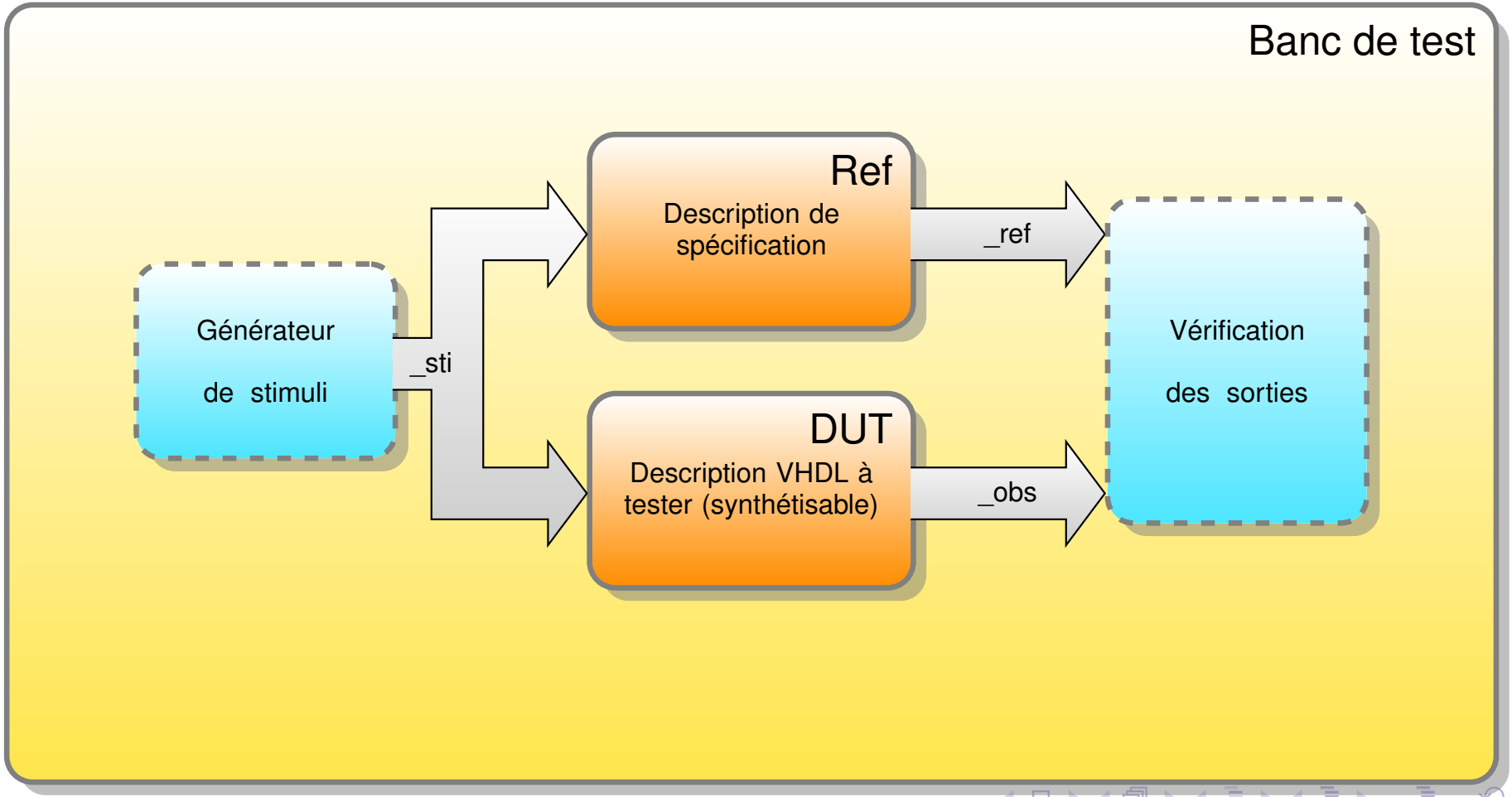
Grey box

- Les sorties et l'intérieur du design sont analysées
  - Compromis entre les deux autres approches
  - Permet de limiter la vérification des sorties par l'analyse interne



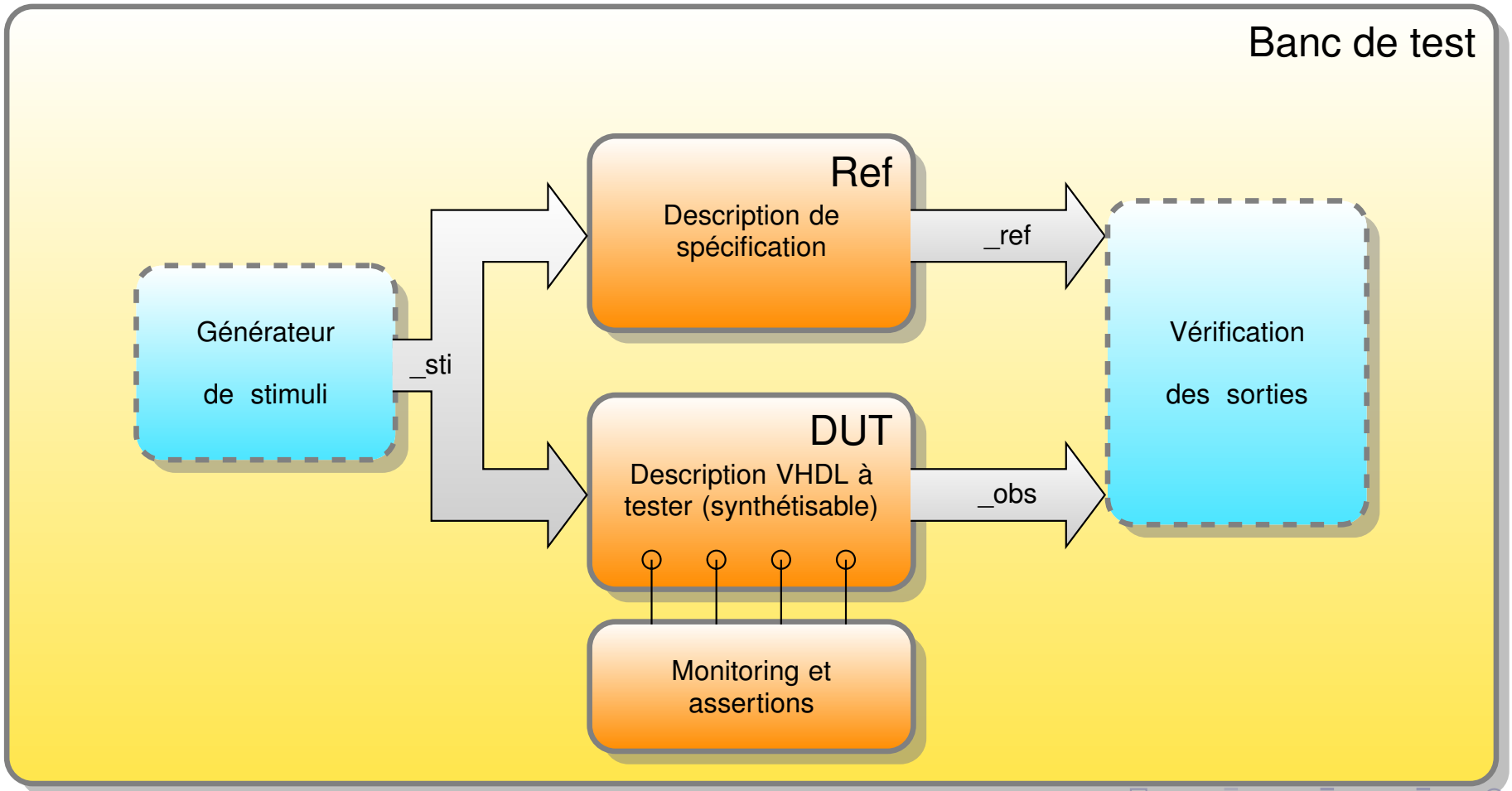
Grey box

- Les sorties et l'intérieur du design sont analysées
  - Compromis entre les deux autres approches
  - Permet de limiter la vérification des sorties par l'analyse interne



Grey box

- Les sorties et l'intérieur du design sont analysées
  - Compromis entre les deux autres approches
  - Permet de limiter la vérification des sorties par l'analyse interne



White/grey/black box: Effort

Challenge	**Black**	**White**	**Grey**
Création d'un modèle de référence	Haut	Non	Moyen
Ajout de moniteurs et assertions	Non	Haut	Bas
Tracer un bug de la sortie à la source	Haut	Bas	Bas
Vérifier l'implémentation des features	Haut	Bas	Bas

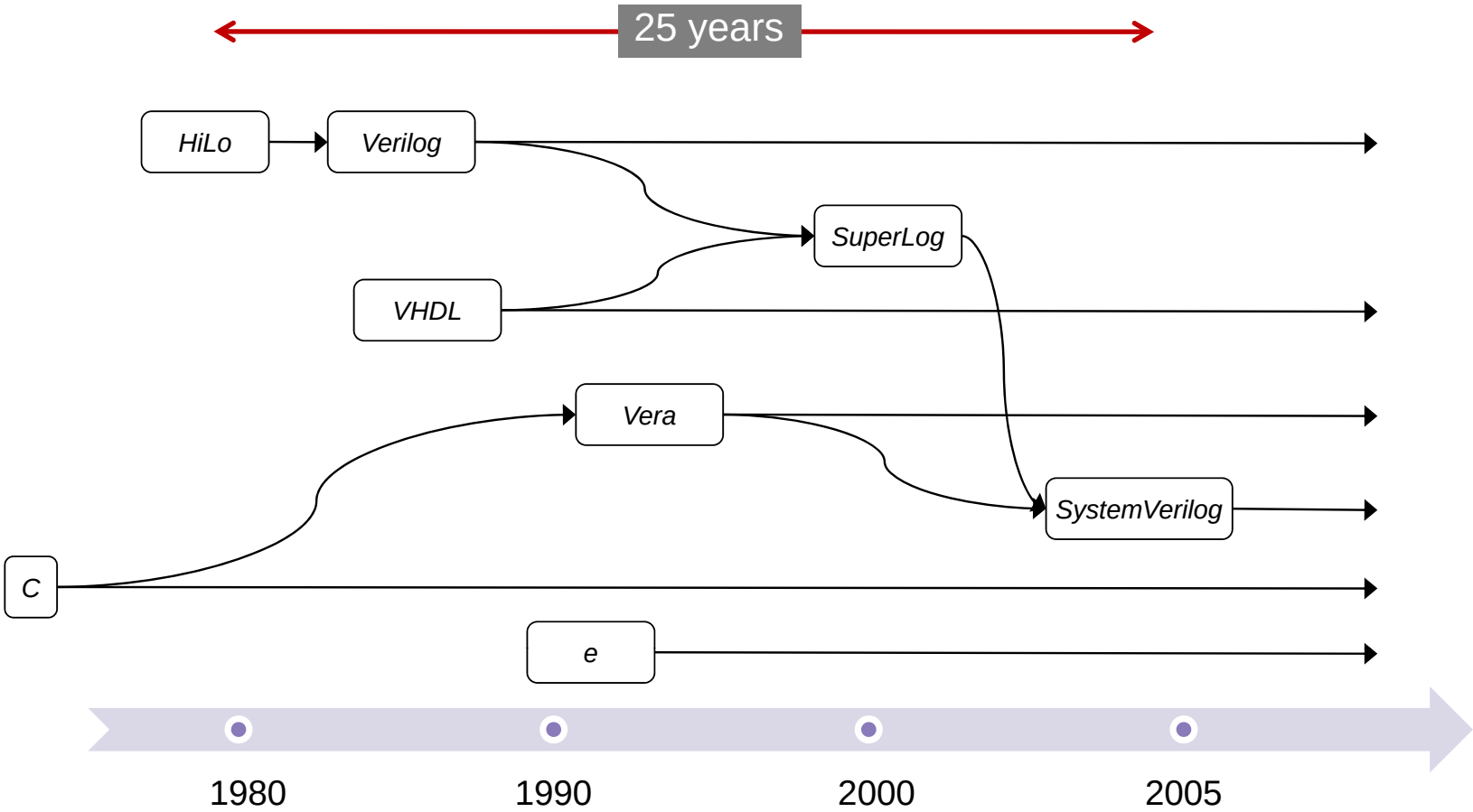
Vérification par émulation

- Constat: La simulation nécessite beaucoup de temps processeur
- Solution: Exploiter les FPGAs pour accélérer le processus
- Difficultés:
  - Temps de synthèse-placement-routage
  - Observabilité (nombre d'entrées/sorties)
  - Gestion des assertions
  - Se prête plutôt à des modèles *black box*
- Solutions commerciales:
  - Veloce, de Mentor
  - ZeBu, de Eve
  - Palladium series, de Cadence

Langages pour la vérification

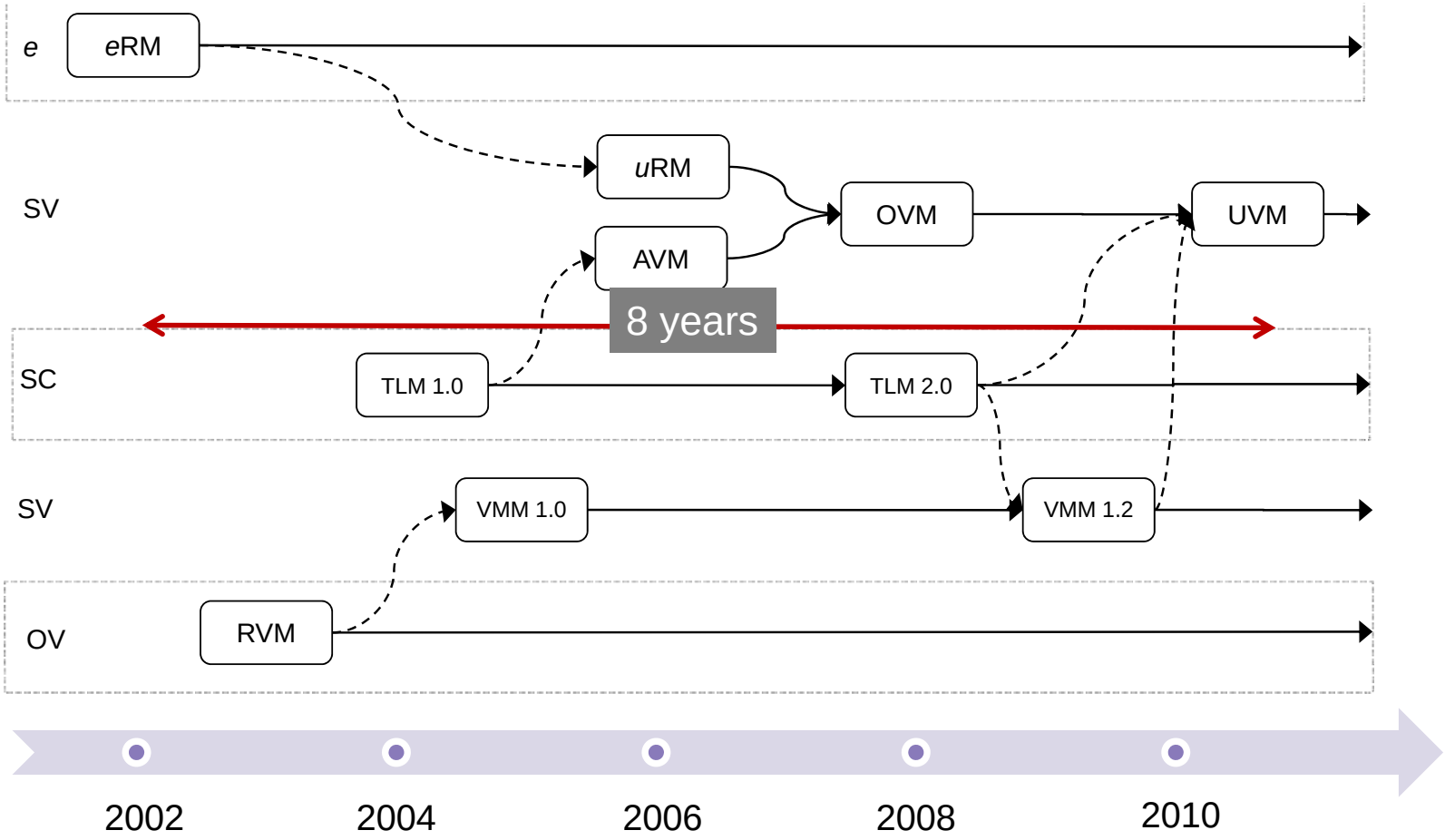
- Projets de petite et moyenne taille
  - Langage VHDL ⚠ Nouveauté: UVVM
- Projets de taille moyenne à élevée
  - Besoin de
    - Modèles décrits à très haut niveau (transaction)
    - Génération de stimuli aléatoires
    - Vérification par assertions
  - Solutions actuelles
    - SystemC
    - PSL (assertions)
  - Nouveau langage
    - SystemVerilog

Origine des langages



Functional Verification of Today's and Tomorrow's SoCs, Janick Bergeron, Synopsys, 2011

Origine des méthodologies



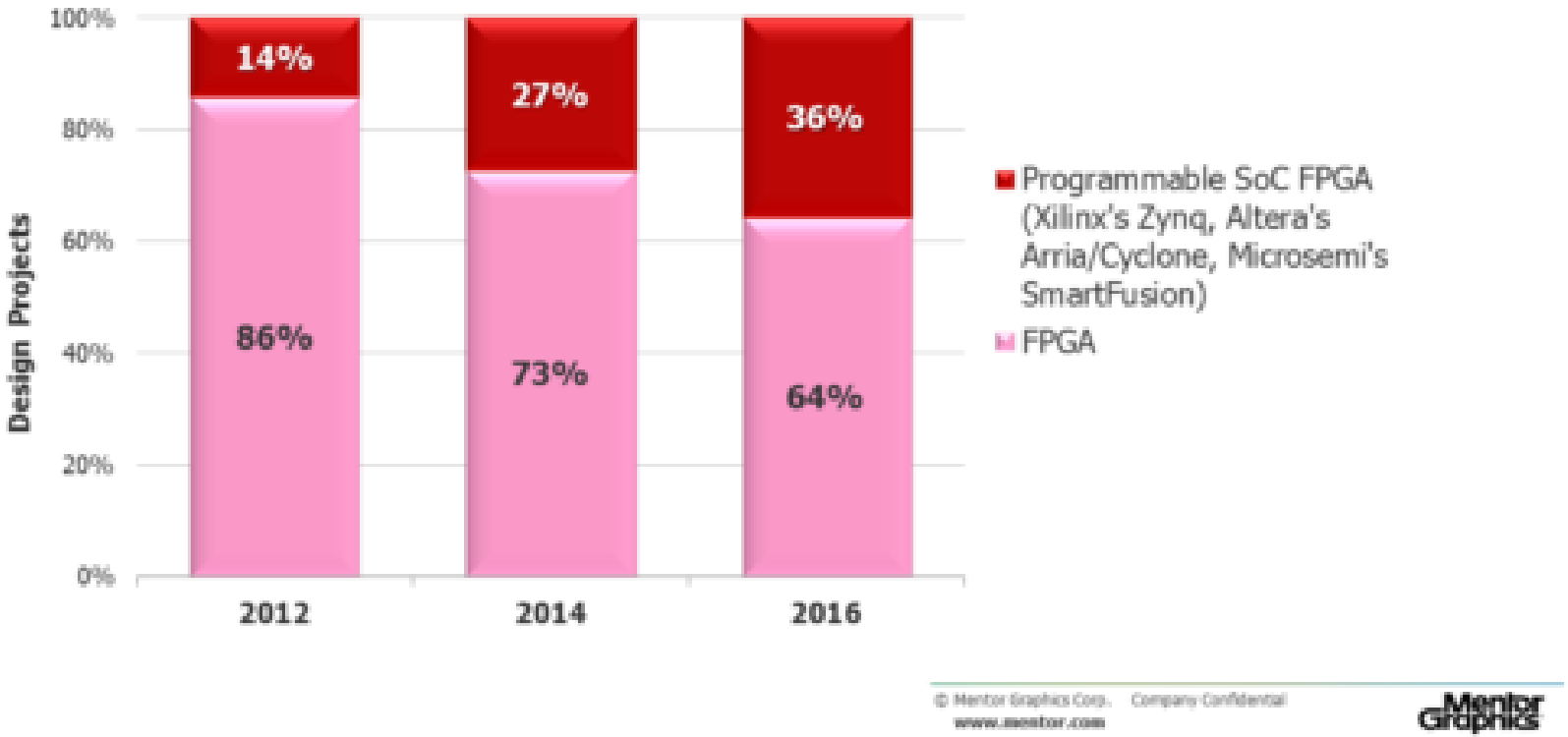
Functional Verification of Today's and Tomorrow's SoCs, Janick Bergeron, Synopsys, 2011

Etude sur l'évolution des usages

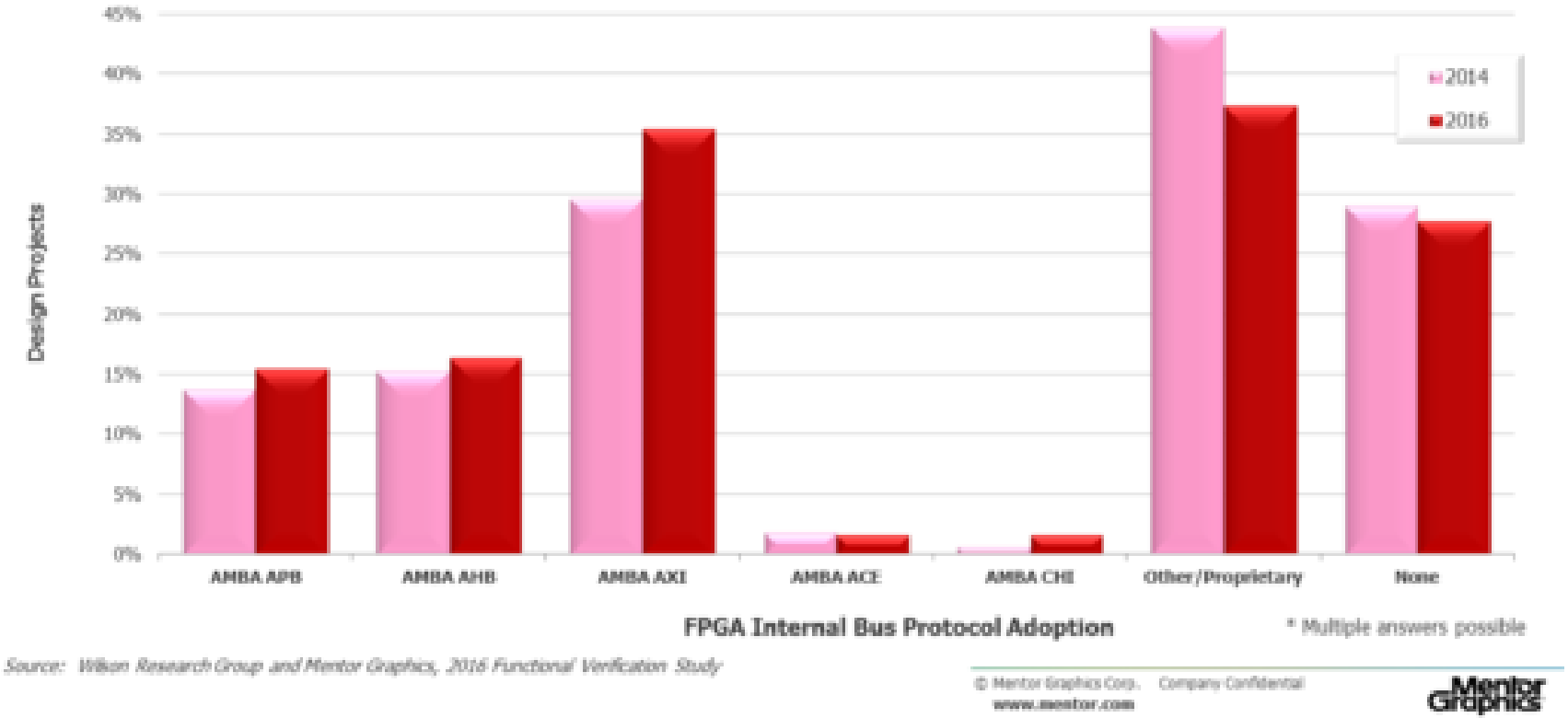
- Publiée par Mentor
- Comparaison entre 2012 et 2016
- Designs FPGA et non-FPGA (ASIC)

https://blogs.mentor.com/verificationhorizons/blog/2016/08/08/prologue-the-2016-wilson-research-group-functional-verification-study/

Type of FPGA Implementation

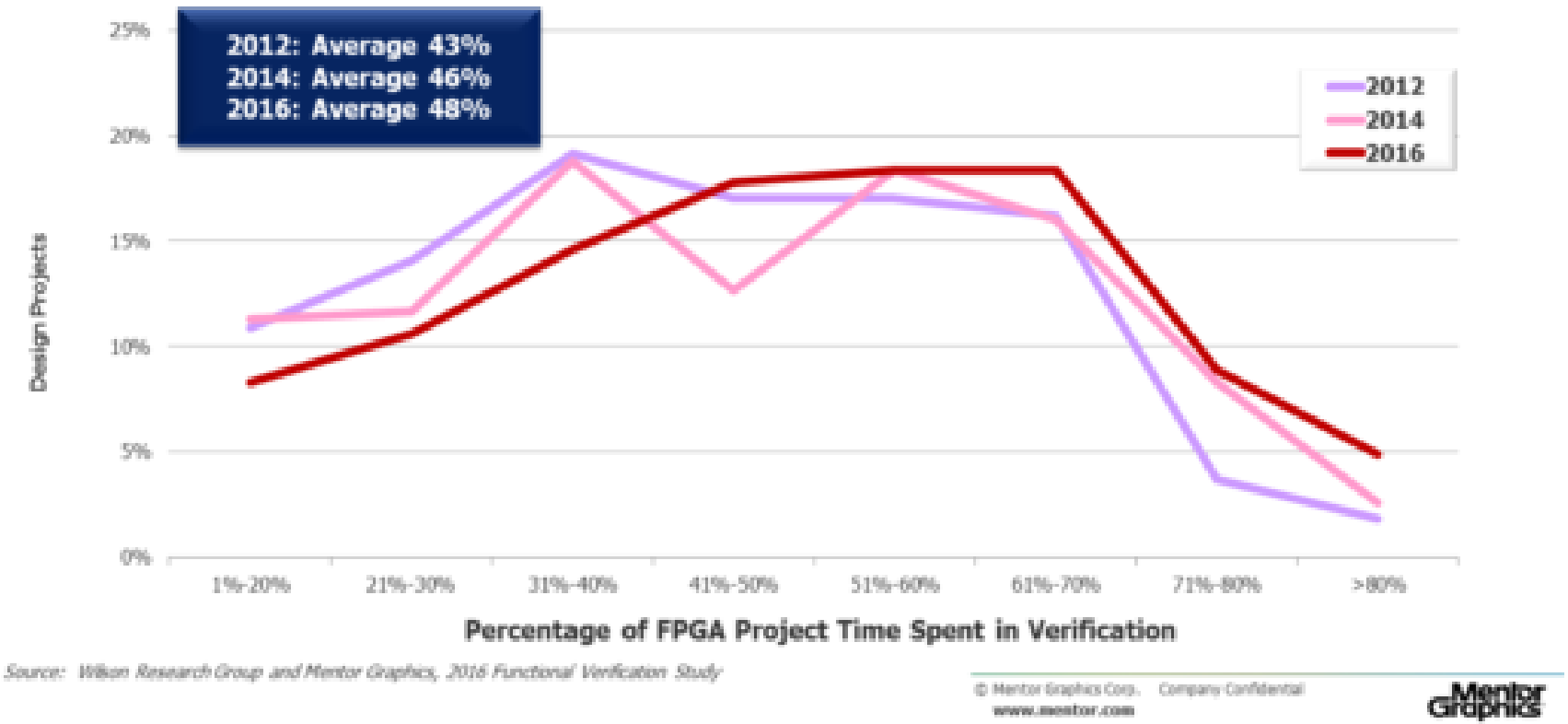


FPGA On-chip Bus Protocol Adoption



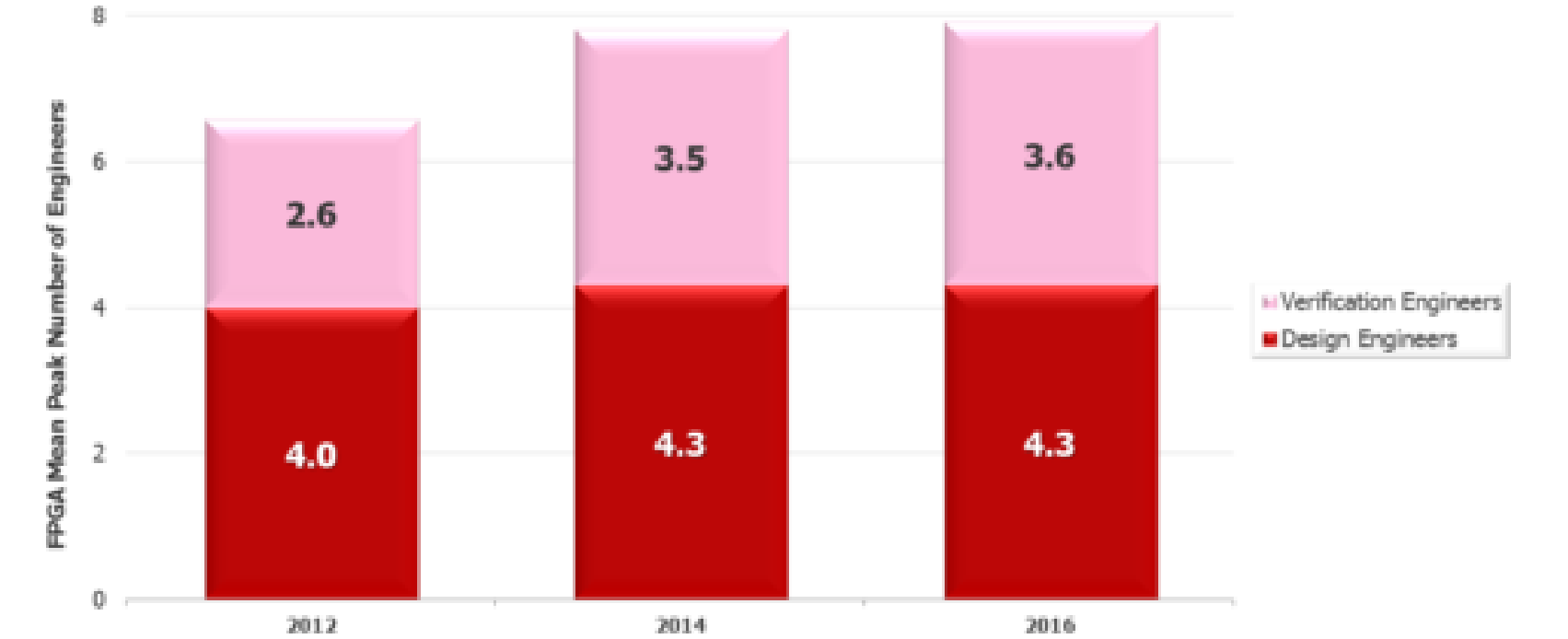
Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

FPGA Project Time Spent in Verification



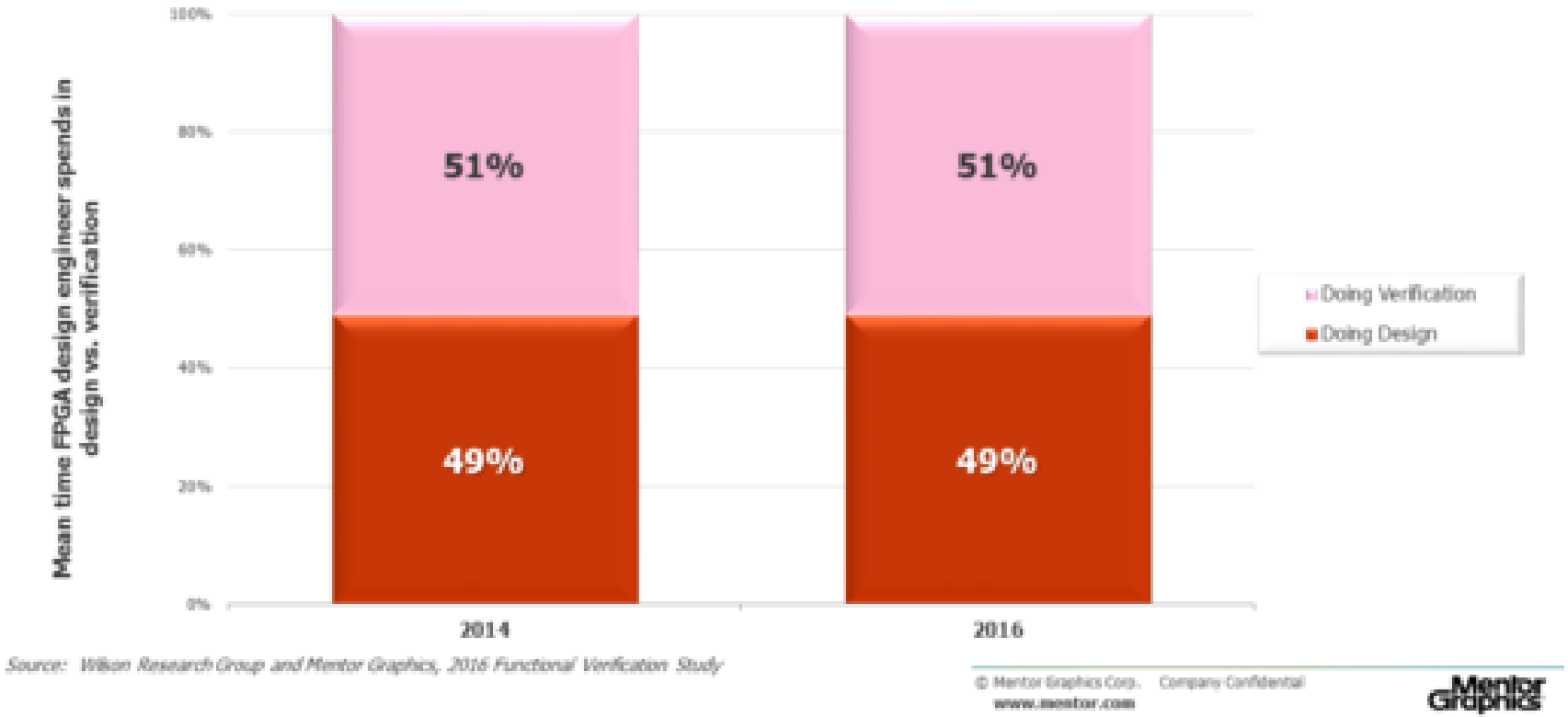
Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

Mean Peak Number of FPGA Engineers on a Project



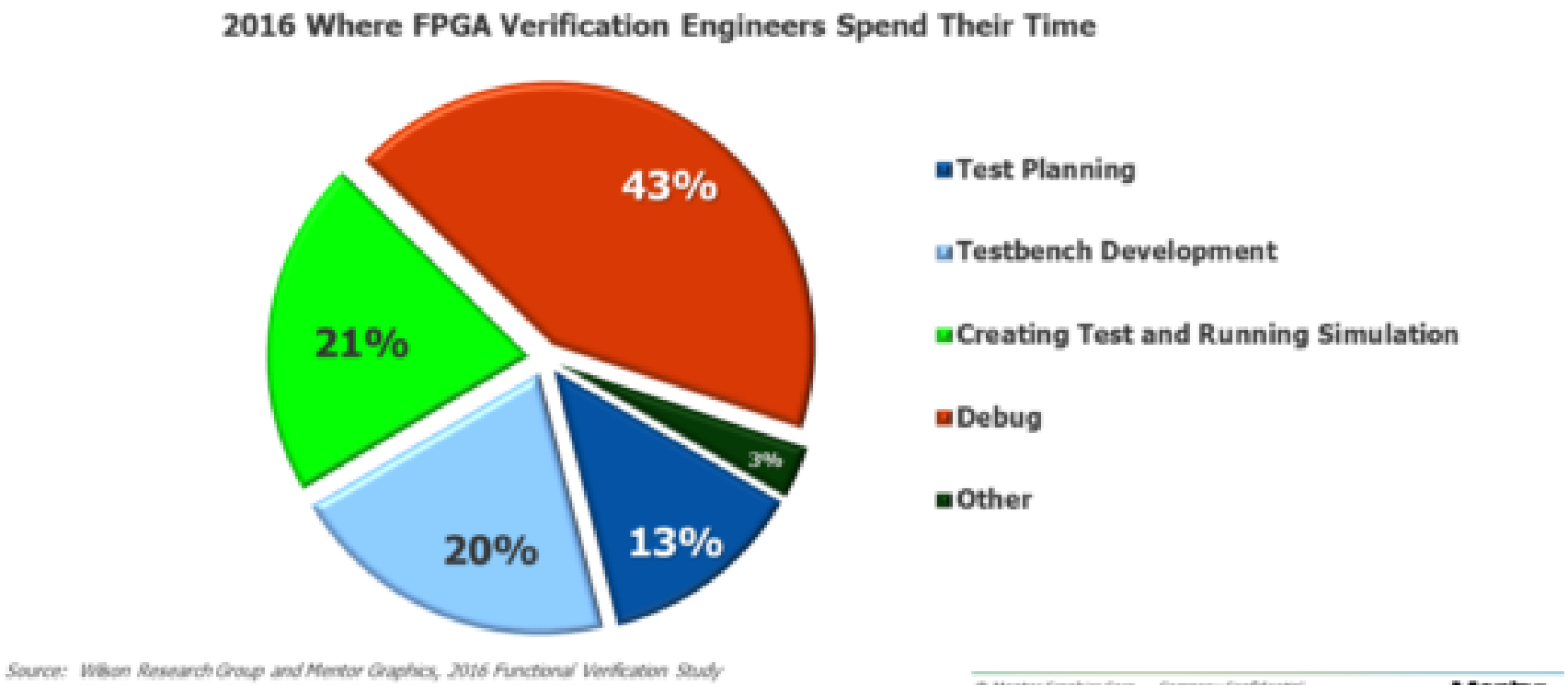
Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

Where FPGA Designers Spend Their Time

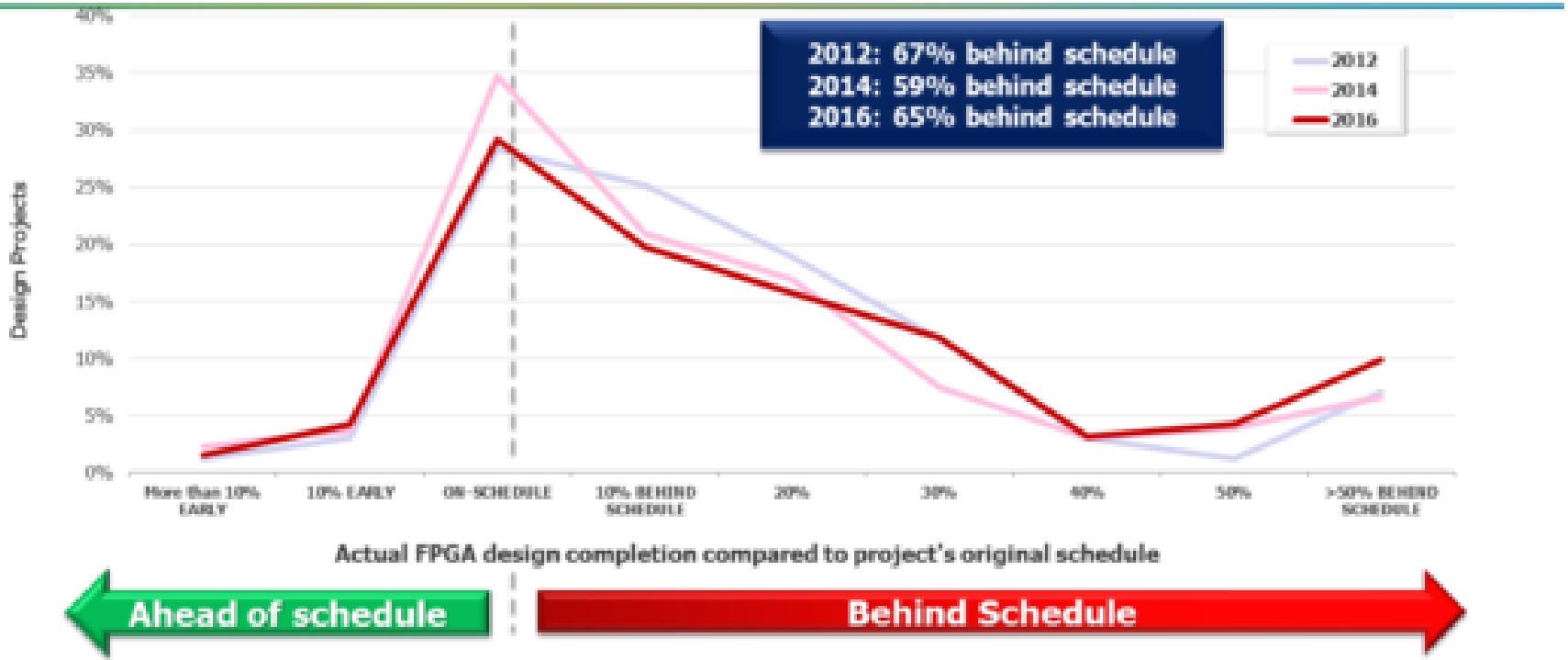


Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

Where FPGA Verification Engineers Spend Their Time

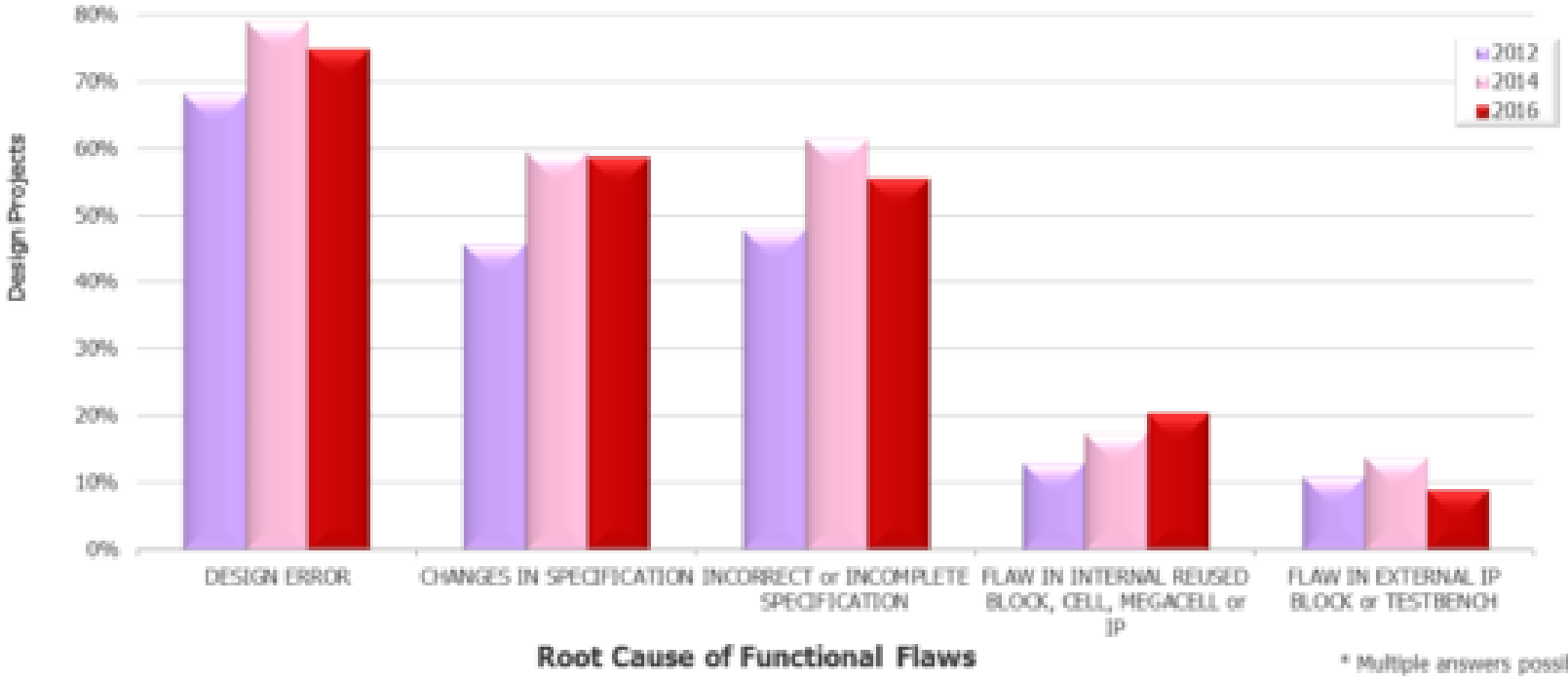


FPGA Project Completion to Original Schedule

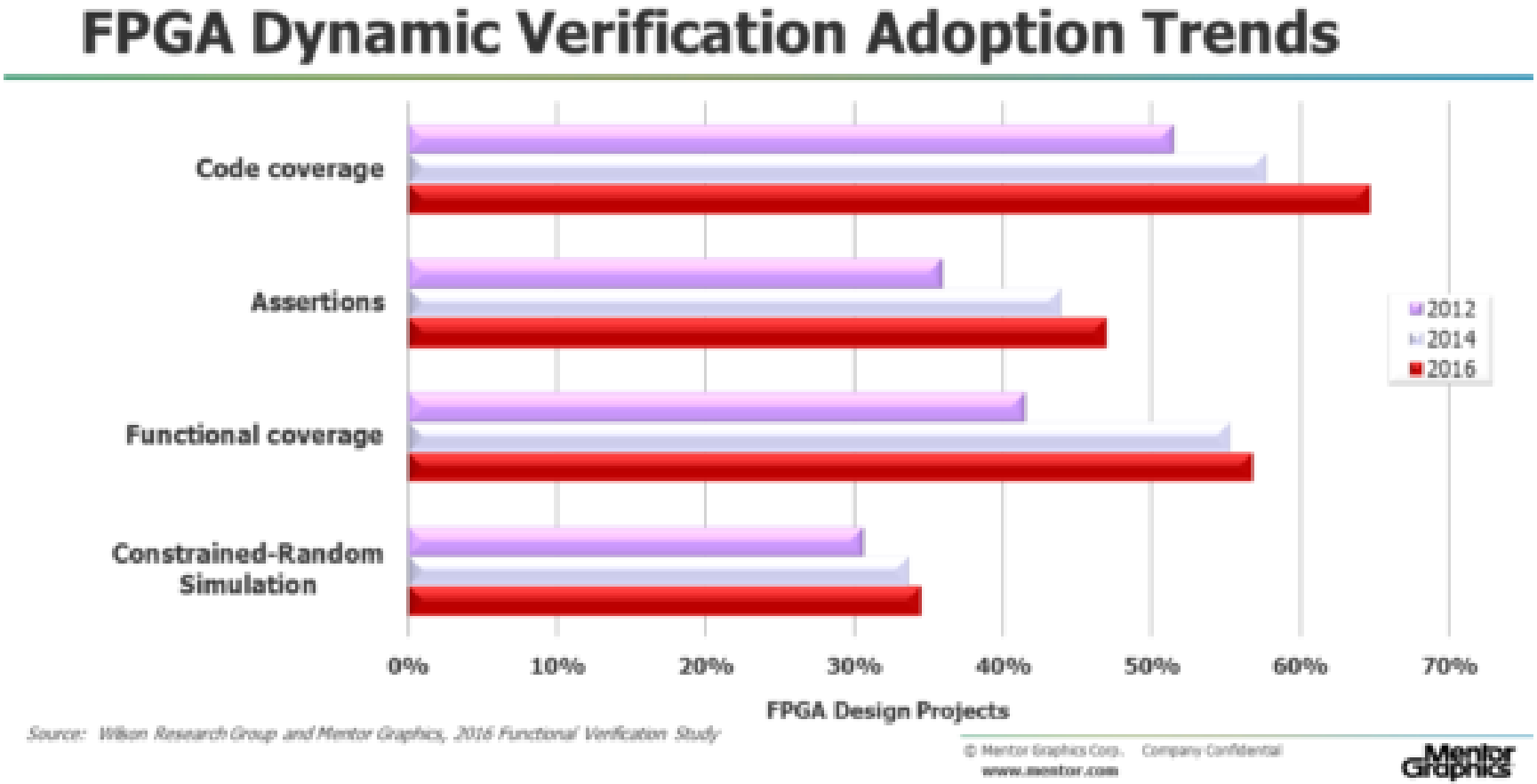


Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

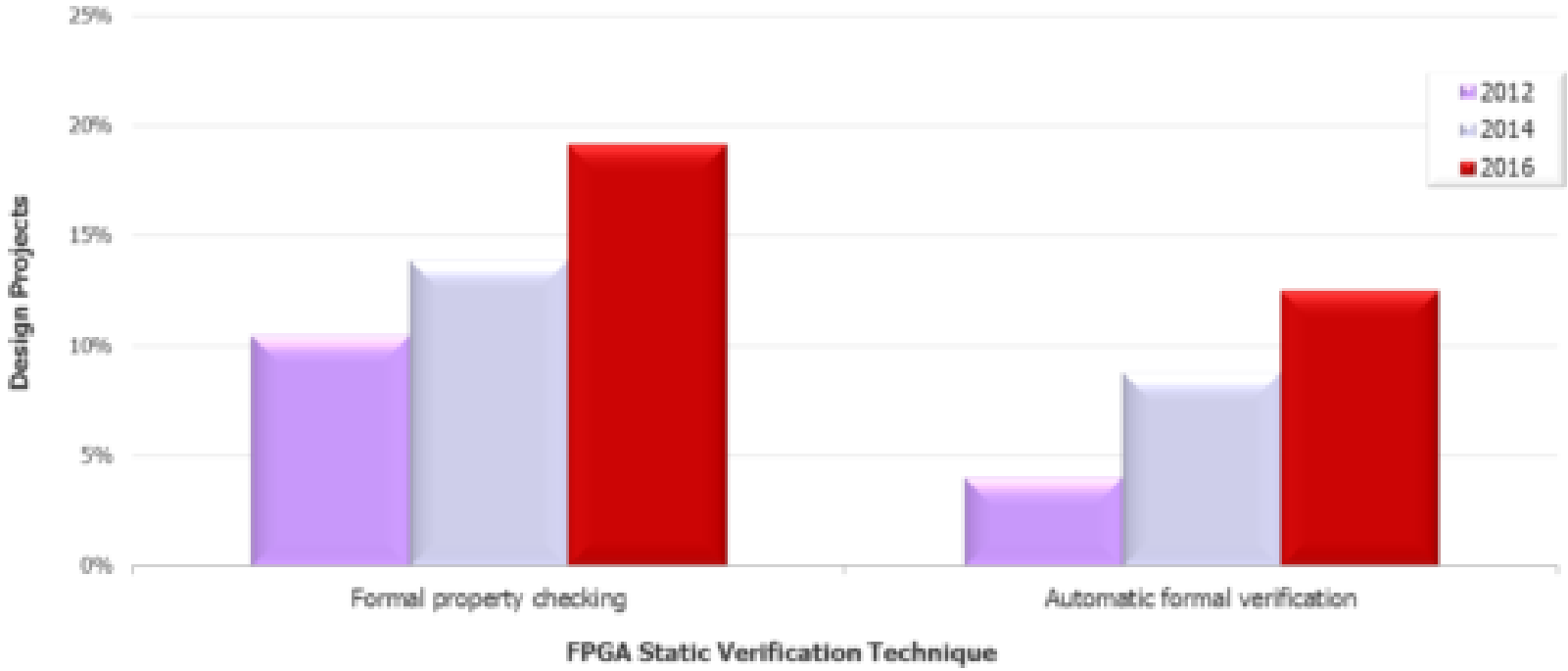
Root Cause of FPGA Functional Flaws



Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

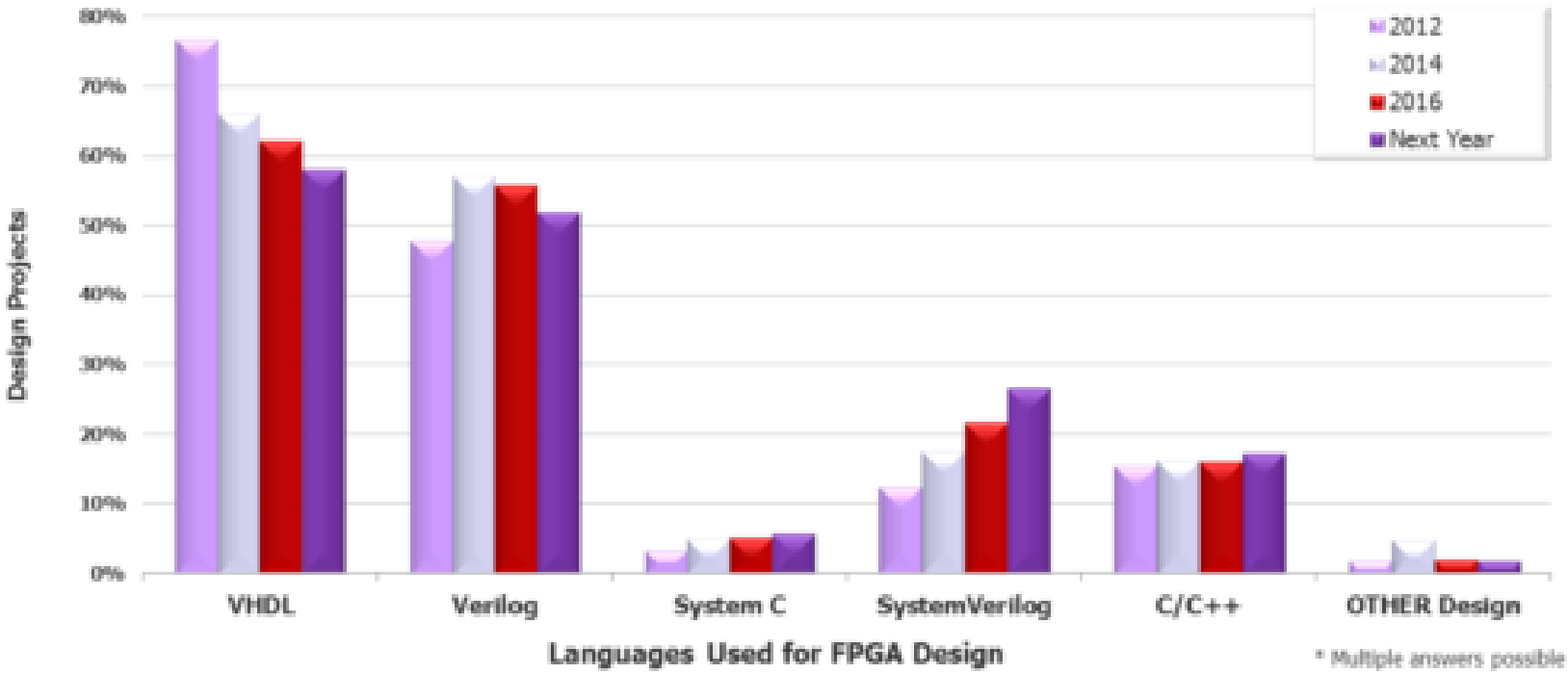

FPGA Dynamic Verification Adoption Trends
Code coverage
Assertions
Functional coverage
Constrained-Random Simulation
2012
2014
2016
0%
10%
20%
30%
40%
50%
60%
70%
FPGA Design Projects
Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study
© Mentor Graphics Corp. Company Confidential
www.mentor.com
Mentor Graphics

FPGA Formal Technology Adoption Trends



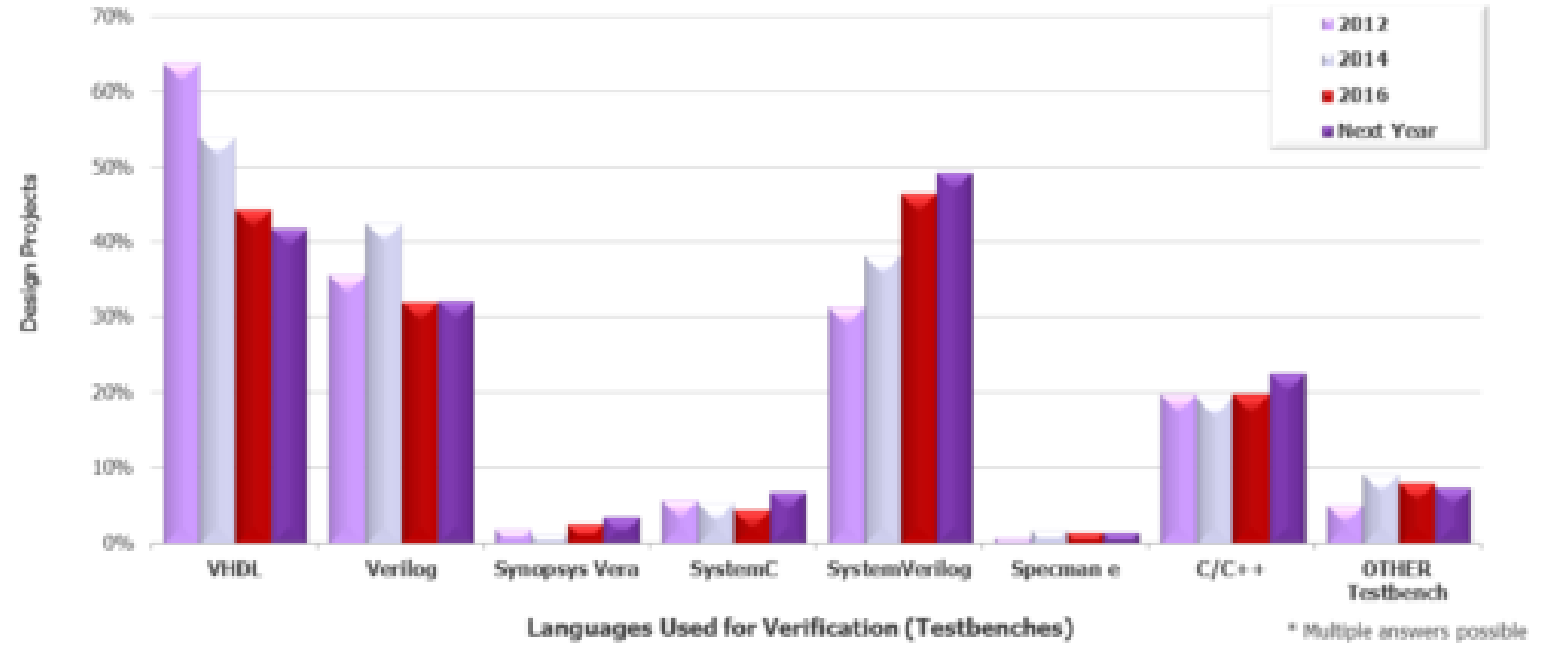
Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

FPGA Design Language Adoption Trends



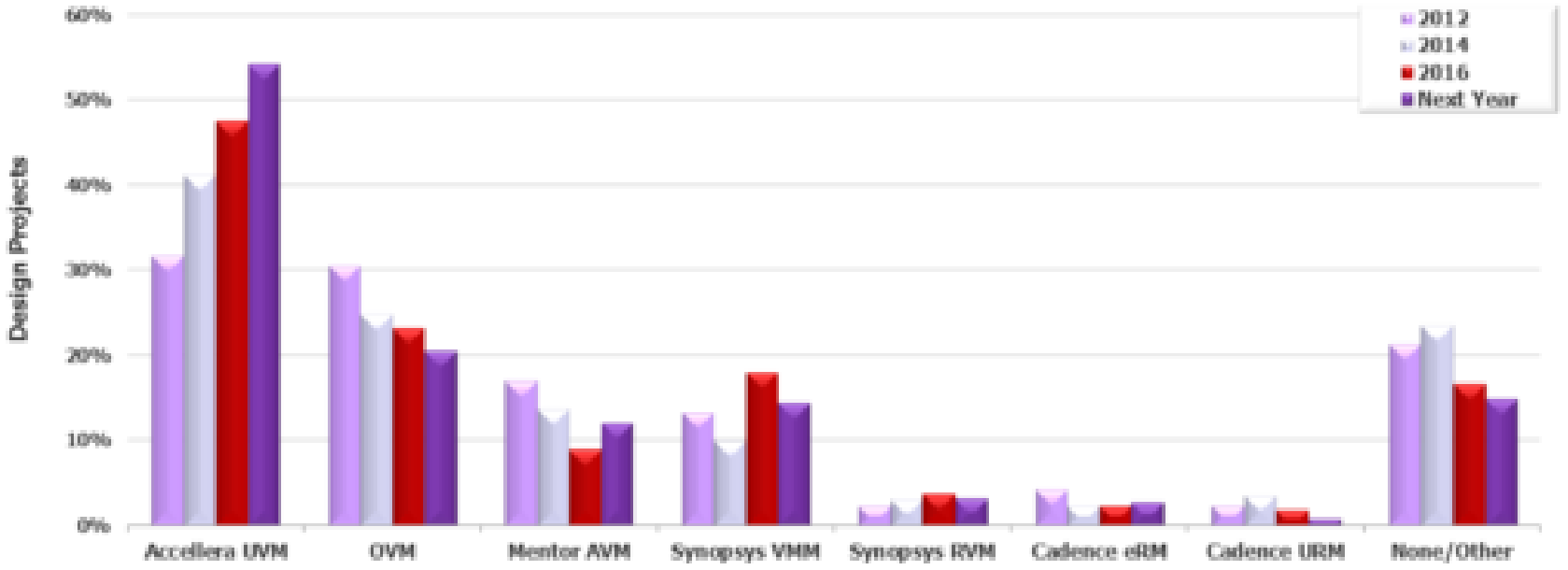
Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

FPGA Verification Language Adoption Trends



Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

FPGA Testbench Methodology Adoption Trends

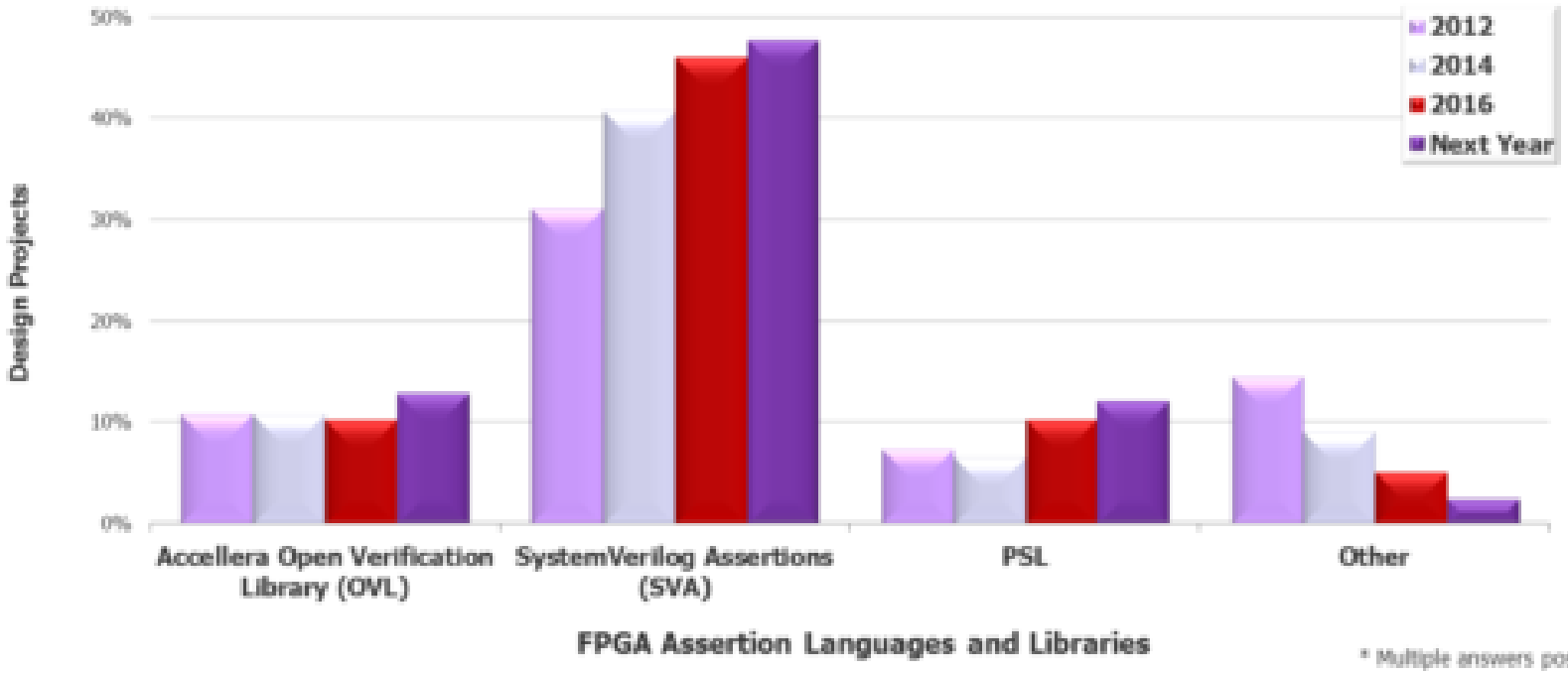


FPGA Methodologies and Testbench Base-Class Libraries

* Multiple answers possible

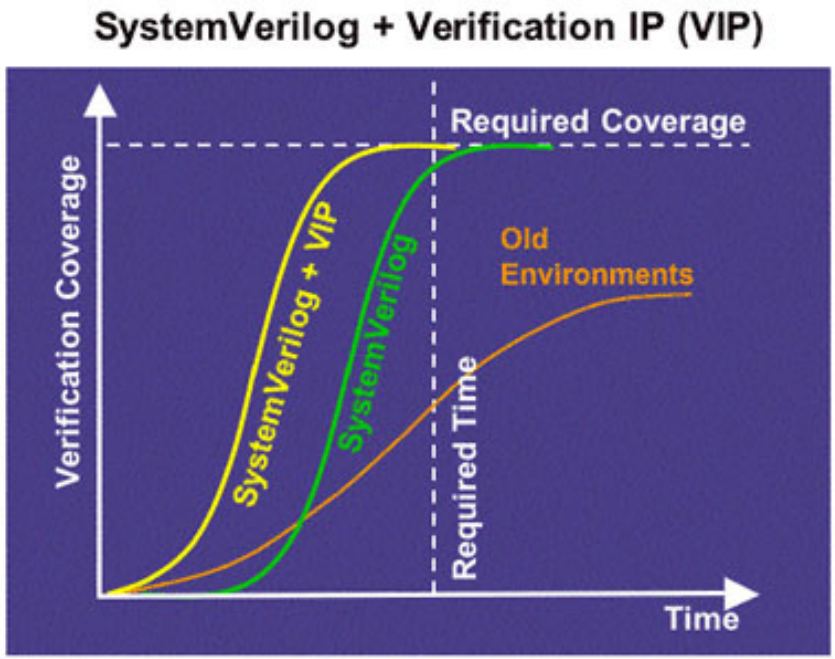
Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

FPGA Assertion Language Adoption



Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

SystemVerilog vs. VHDL



Source: `http://www.design-reuse.com/articles/7844/verification-ip-verification-ip--part-2-by-dr-aart-de-geus.html`

SystemVerilog

- Ce qu'il faut savoir de SystemVerilog pour la vérification
  - Langage orienté objet
  - Notion de classe (monde logiciel)
    - Héritage
    - Création dynamique d'objets
  - Notion de module (monde matériel)
    - Hiérarchie de composants
    - Création statique
  - Notion d'interface (entre-deux mondes)
    - Facilite la mise au point de bancs de tests

SystemVerilog

- Langage orienté objet
  - Réutilisation
- Génération aléatoire
  - Non contrainte
  - Contrainte
- Gestion des assertions
- Fonctions de couverture
- Idéal pour de la vérification aléatoire
- Des méthodologies ont été développées

Méthodologies SystemVerilog

- Cadence: VMM (Verification Methodology Manual)
  - Version 1.2
  - `http://www.vmmcentral.org/`
- Mentor: AVM (Advanced Verification Methodology)
  - Version 3.0
  - Terminé
- Ensemble: OVM (Open Verification Methodology)
  - Version 2.1
  - `http://www.ovmworld.org/`
- UVM (Universal Verification Methodology)
  - Unification de VMM et OVM
  - Version 1.1
  - `http://www.uvmworld.org/`
  - Sera présentée durant le cours

Niveaux d'abstractions

- Evolution des niveaux d'abstractions

Langages RTL: **VHDL/Verilog**

Niveaux d'abstractions

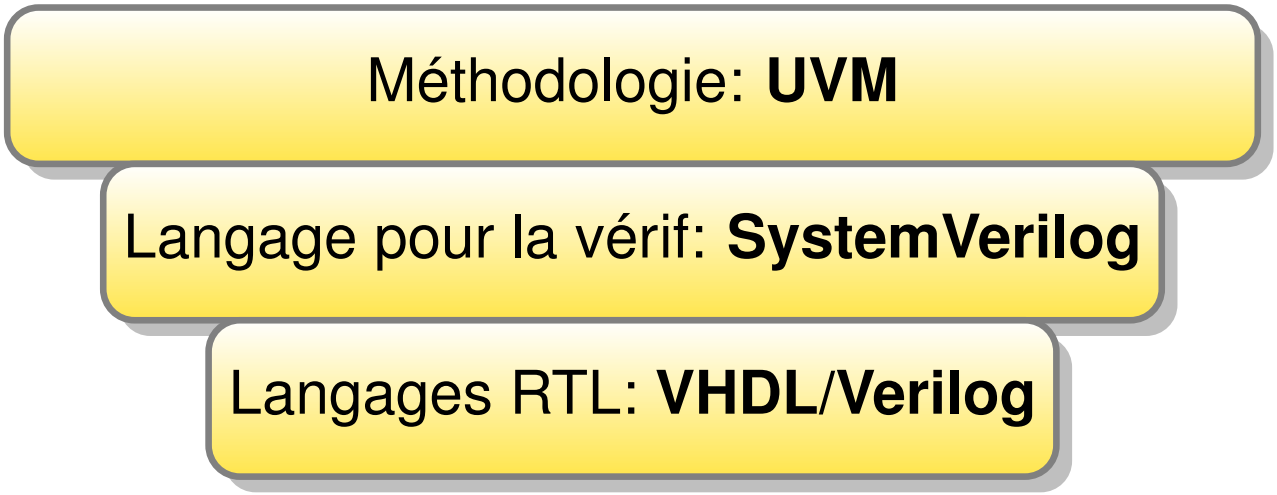
- Evolution des niveaux d'abstractions

Langage pour la vérif: **SystemVerilog**

Langages RTL: **VHDL/Verilog**

Niveaux d'abstractions

- Evolution des niveaux d'abstractions



Niveaux d'abstractions

- Evolution des niveaux d'abstractions

Méthodologie TLM

- Pour des systèmes complexes, la méthodologie *Transaction Level Modeling* permet:
  - Une meilleure modularité
  - Une meilleure réutilisabilité
  - De découpler le niveau RTL d'autres niveaux d'abstraction
- Concept:
  - Monter en abstraction dès que possible
  - Arriver dans le monde logiciel
- SystemVerilog est fait pour
- VHDL s'en approche avec UVVM
- L'avenir de la vérification est TLM