MicroPython

1 - Qui suis-je ?

1.1 - Qui suis-je ?

Vincent Poulailleau

https://github.com/vpoulailleau

vpoulailleau@gmail.com

https://www.linkedin.com/in/vpoulailleau/

https://www.lecalamar.fr/

L'essentiel : Qui suis-je ?

Vincent Poulailleau

https://www.linkedin.com/in/vpoulailleau/

https://www.lecalamar.fr/

2 - Introduction

2.1 - Python

Langage de programmation multi-fonction

Langage de programmation multi-paradigme

Typage dynamique

Clarté

Concision

2.2 - Efficacité de codage

- Le programmeur s'occupe des fonctionnalités
- Python s'occupe de l'implémentation
  - Allocation mémoire
  - Libération mémoire
  - Structures évoluées de données
    - Tableau à taille variable
    - Mémoire associative clé / valeur
    - ***FIFO***

2.3 - En C

Hello world en C

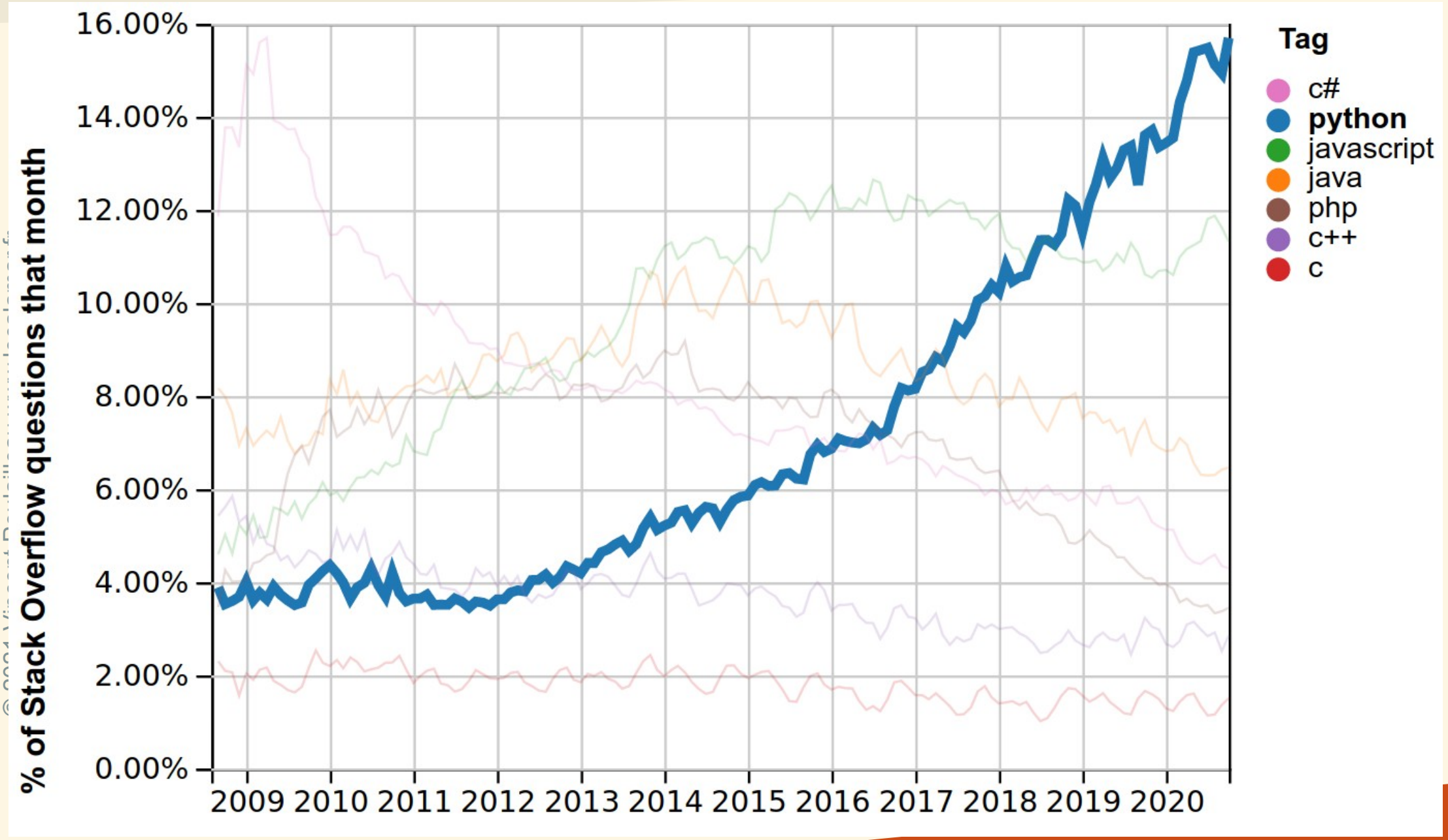
```c
#include <stdio.h>


int main(void)
{
    printf("hello, world\n");
    return 0;
}
```

2.4 - En Python

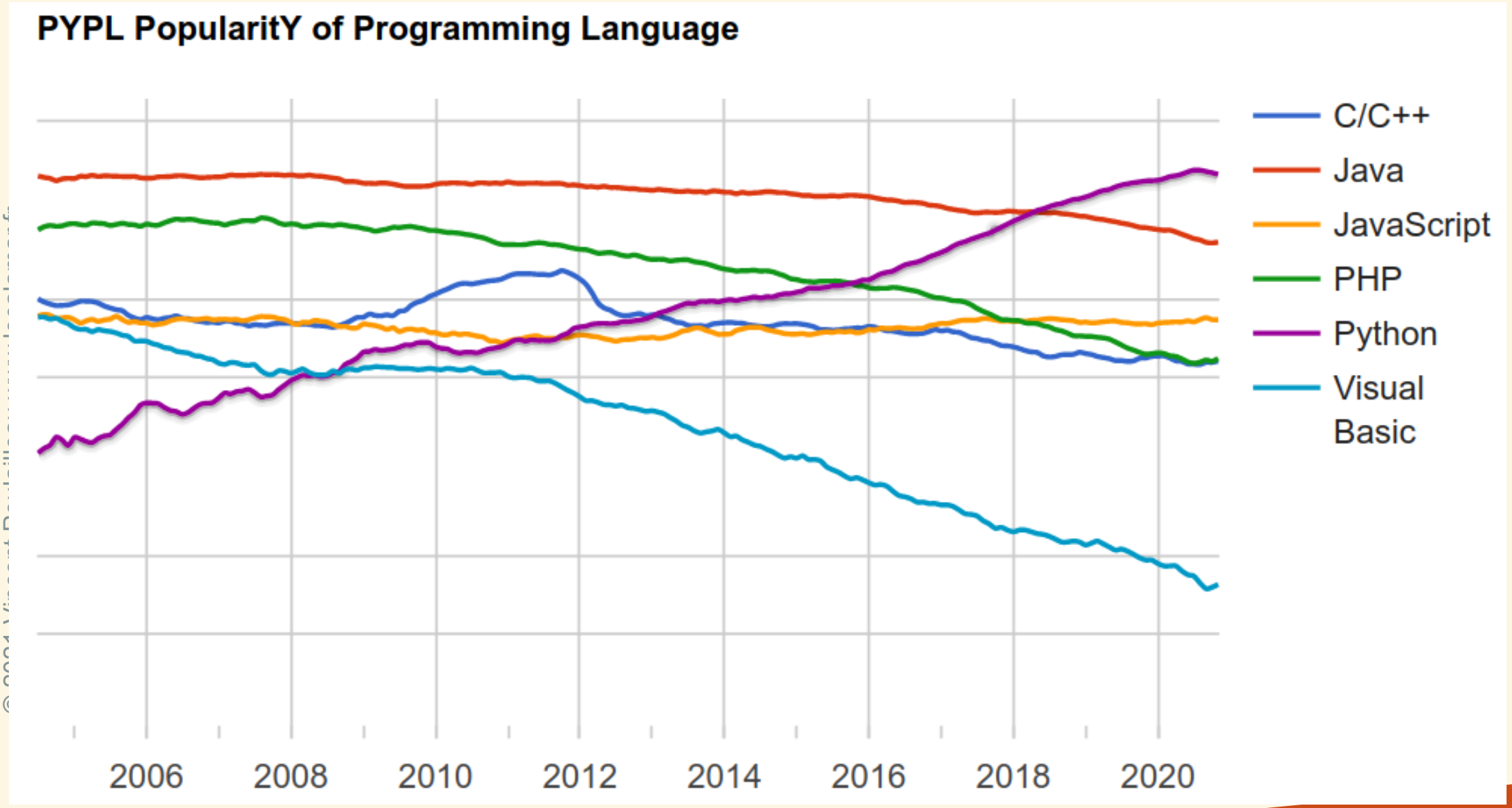
Hello world en Python

```
print("Hello world!")
```

2.5 - Stackoverflow



2.6 - Popularity of programming language (PyPL)



2.7 - Mais…

Python ne s'est pas intéressé de près au monde de l'embarqué

- Utilise beaucoup de ressources
- Pas compatible microcontrôleur

2.8 - Mais…

Damien Georges

- Correction quantique dans l'inflation de Higgs
- Étude de la masse des neutrinos
- Cosmologie extra-dimensionnelle
- Et…

2.9 - Mais…

2.10 - Mais…

En 2013, crowdfunding par Damien Georges

- Demande 15 000 £, reçoit 97 803 £
- Il va créer MicroPython : un interpréteur Python pour microcontrôleur
- Implémentation de Python 3.4/3.5
- Codé en C
- Licence MIT

En 2017, CircuitPython, fork pour l'enseignement

2.11 - Mais…

2.12 - Mais…

MicroPython n'est pas Python

- Il ne fournit pas tout
- Il fournit beaucoup
- Y compris des fonctionnalités allégées en consommation de ressources

Nécessite 16 Kio de RAM et 256 Kio de code

L'essentiel : Introduction

Langage de programmation multi-fonction et multi-paradigme

Langage clair et concis

Le programmeur s'occupe du quoi, pas du comment

Python est facile à écrire et à lire (facilite la maintenance et la recherche de bug)

La popularité de Python est en forte croissance

2013 : création de MicroPython, un Python pour microcontrôleur

3 - Installation de MicroPython

3.1 - Installation des outils d'installation !

Installez Python 3.7 ou plus

Sous Linux : `sudo apt install python3.8` ou équivalent

Sous Windows 10 : passez par le store d'applications

Sinon : www.python.org/downloads

3.2 - Installation des outils d'installation !

Téléchargez le programme de flashage

https://github.com/micropython/micropython/blob/master/tools/pydfu.py

Par exemple :

https://github.com/micropython/micropython/blob/cdaec0dcaffbfcb58e190738cf6e9d345414644f0/tools/pydfu.py

3.3 - Installation des outils d'installation !

Créez un environnement virtuel, avec un shell

```
cd le_dossier_qui_va_bien
python3.8 -m venv venv
```

Activez l'environnement virtuel

`source venv/bin/activate` sous Linux

`.\venv\Scripts\Activate.ps1` sous Windows

`(venv)` est ajouté au début du prompt

```
pip install pyusb==1.1.1
```

3.4 - Installation de MicroPython

Sur STM32F4 Discovery

https://www.micropython.org/download#other

Prenez la dernière version stable

Ou, à vos risques et périls, la version en cours de développement

3.5 - Installation de MicroPython

Débranchez la STM32F4 Discovery

Mettez un jumper entre VDD et BOOT0

Piquez le jumper J1 de la STM32F4 Discovery Shield (en dessous du bornier CAN)

VDD et BOOT0 sont sur le header de droite de la STM32F4 Discovery

3.6 - Installation de MicroPython

Branchez un câble mini-USB en haut de la STM32F4-Discovery

- Pour l'alimentation électrique
- Ou câblez la pin PA9 à la pin 5V

Branchez un câble micro-USB en bas de la STM32F4-Discovery

- Pour le transfert du fichier (flashage)

3.7 - Installation de MicroPython

Dans le terminal avec l'environnement virtuel activé

```
python pydfu.py --list
```

Liste des périphériques en mode DFU

```
    Bus 1 Device 009: ID 0483:df11
    Memory Layout
        0x8000000  4 pages of  16K bytes
        0x8010000  1 pages of  64K bytes
        0x8020000  7 pages of 128K bytes
```

3.8 - Installation de MicroPython

```
python pydfu.py --upload STM32F4DISC-20210222-v1.14.dfu
```

À adapter avec le nom du fichier téléchargé

Flashage

```
File: STM32F4DISC-20210222-v1.14.dfu
    b'DfuSe' v1, image size: 332349, targets: 1
    b'Target' 0, alt setting: 0, name: "ST...", size: 332064, elements: 2
      0, address: 0x08000000, size: 14592
      1, address: 0x08020000, size: 317456
    usb: 0483:df11, device: 0x0000, dfu: 0x011a, b'UFD', 16, 0xee0d55e3
Writing memory...
0x08000000   14592 [==========================] 100%
0x08020000  317456 [==========================] 100%
Exiting DFU...
Finished
```

3.9 - Installation de MicroPython

Ouvrez la nouvelle clé USB (MicroPython est vu comme une clé)

Ouvrez README.txt

3.10 - Installation de MicroPython

boot.py

```
# boot.py -- run on boot-up
# can run arbitrary Python, but best to keep it minimal

import machine
import pyb
pyb.country('FR')
pyb.main('main.py') # main script to run after this one
```

Et main.py vidé

3.11 - ATTENTION

Éjectez correctement la clé USB

Sinon, risque de corruption du système de fichier

- Ré-installation de MicroPython
- Potentielle perte du travail en cours

3.12 - Installation de MicroPython

Remettez le jumper à sa position initiale

Appuyez sur le bouton poussoir noir (Reset)

Démarrez un terminal MicroPython

`screen /dev/ttyACM0` sous Linux

TeraTerm sous Windows

3.13 - Installation de MicroPython

Appuyez sur entrée

Tapez « 3 + 4 » puis entrée

MicroPython répond 7 !

Tapez « help() » puis entrée

3.14 - REPL

Le mode actuel est REPL

Lancé quand main.py a fini son exécution

Read ***E***val ***P***rint ***L***oop, demande une commande, affiche le résultat

Permet de tester en temps réel

Pas de compilation, chargement, debug, recompilation…

Bienvenu dans le nouveau monde de l'informatique embarquée !

L'essentiel : Installation de MicroPython

`help()` pour avoir de l'aide

REPL : ***R***ead ***E***val ***P***rint ***L***oop

4 - Les bases de Python

4.1 - Les bases

Survol des notions de bases de Python

Suffisamment pour se débrouiller

Python couvre bien plus que cela

4.2 - Les commentaires

Ignorés par Python

Pour mieux comprendre le code

Aident les développeurs

N'aident pas les utilisateurs

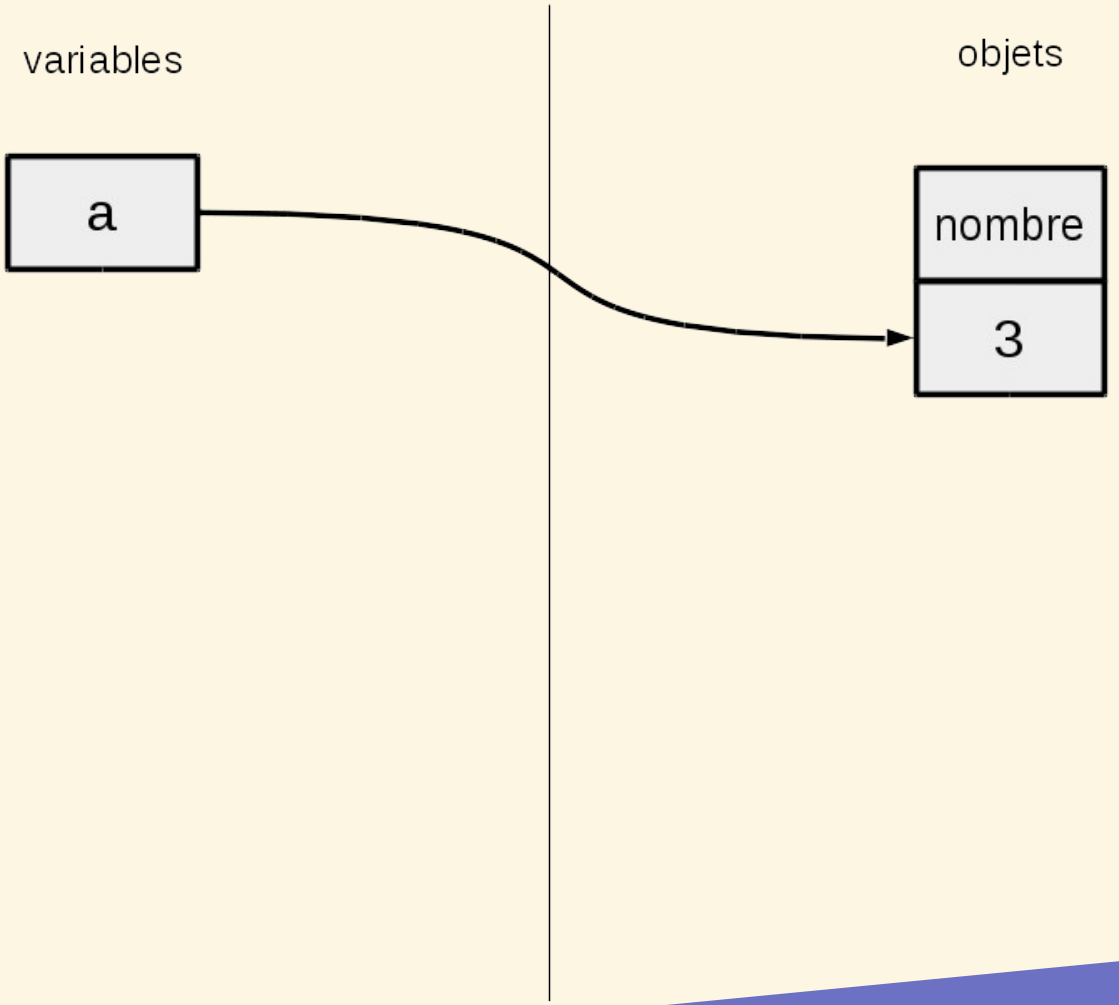
Commencent par #

Finissent à la fin de la ligne

4.3 - Les variables

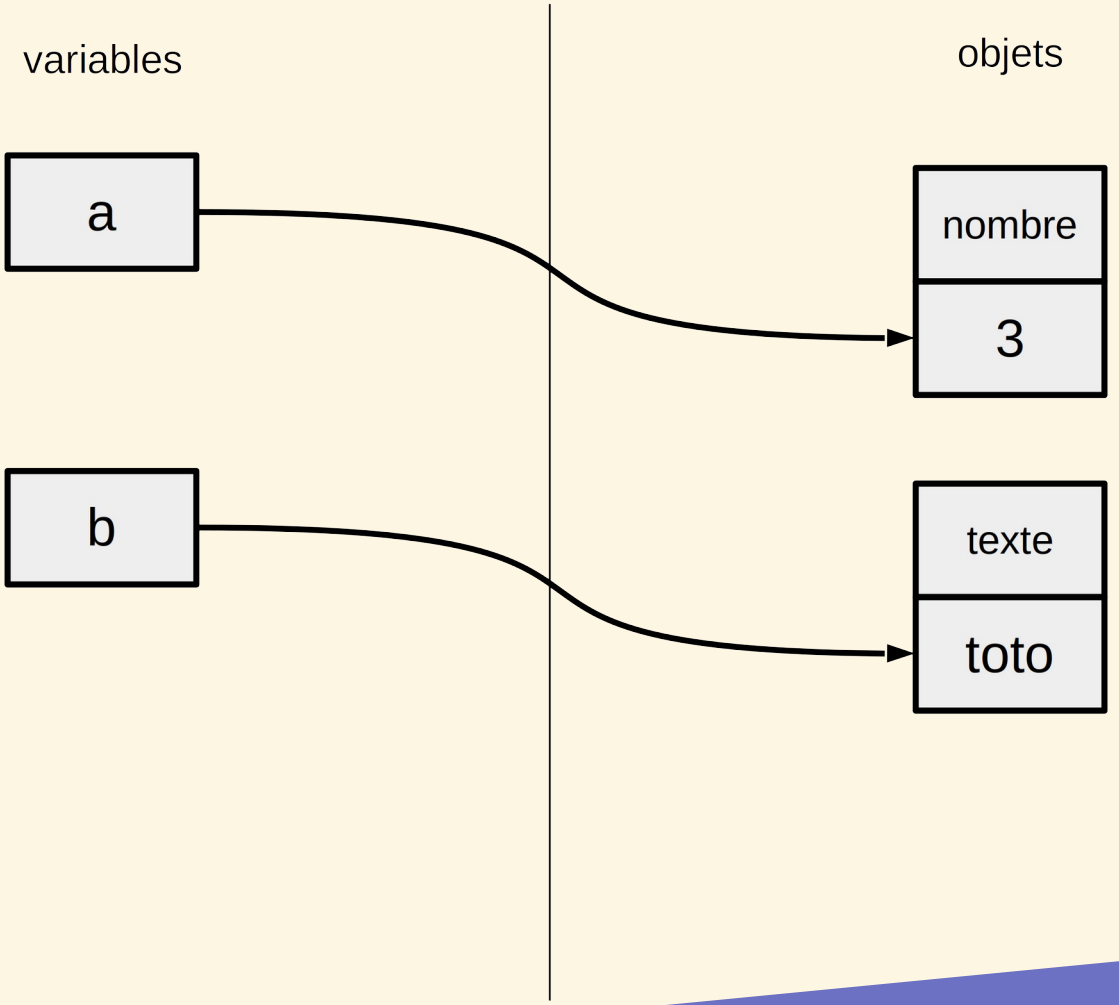
4.3.1 - Une référence

a = 3

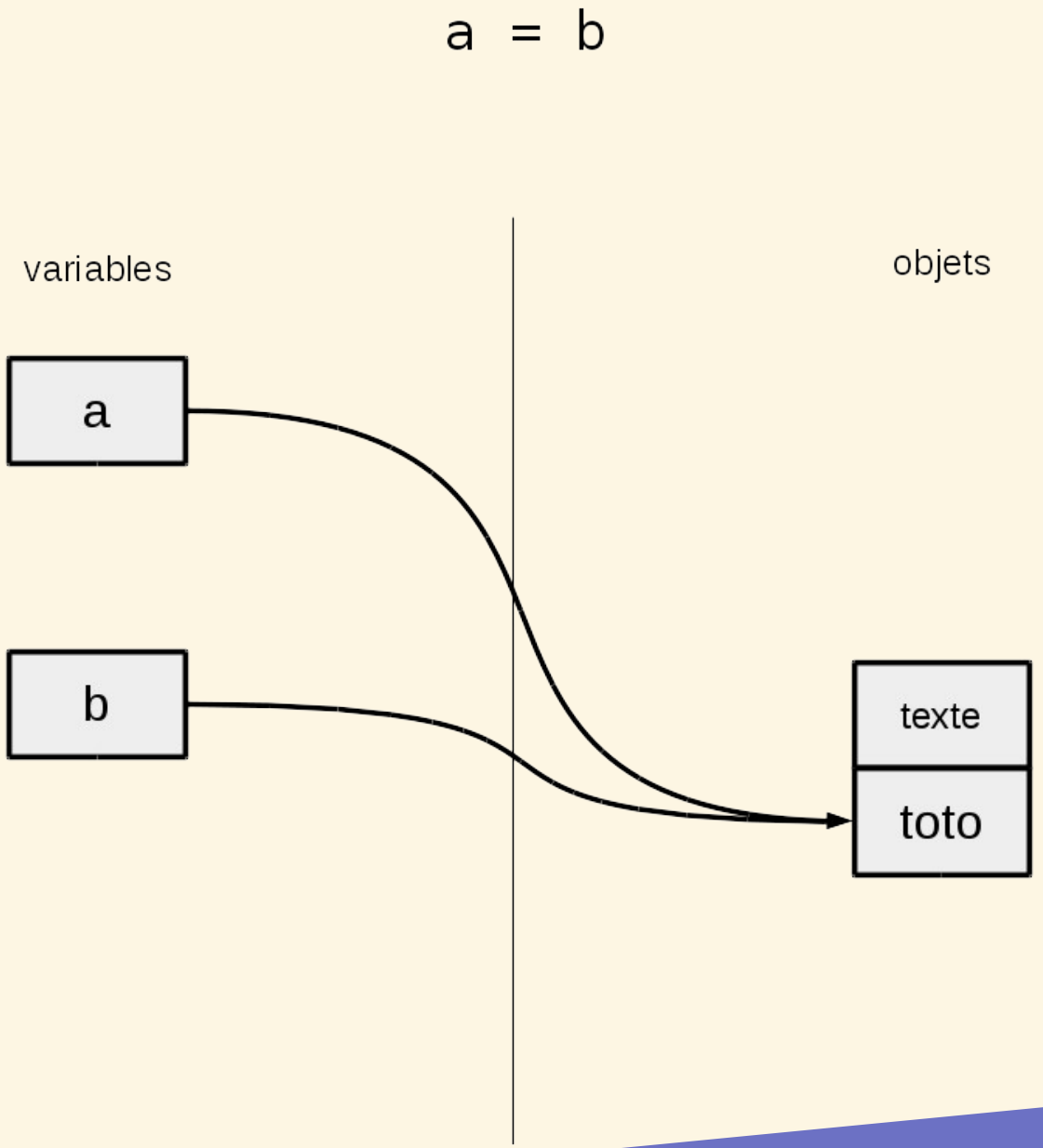


4.3.2 - Une référence

```
b = "toto"
```



4.3.3 - Une référence



4.3.4 - Les nombres

Les nombres

```
a = 42   # entier

b = 123.45   # IEEE754 double précision
```

4.3.5 - Constantes

`None` représente rien ! (le vide, l'absence)

`True` représente le booléen vrai

`False` représente le booléen faux

L'essentiel : Les variables

Un objet contient une valeur et un type

Une variable est une référence vers un objet

`None`, `True`, `False`

4.4 - Les opérateurs

4.4.1 - Les opérateurs arithmétiques

Exemple : `a = 5 + 3`

`+` pour l'addition

`-` pour la soustraction

`*` pour la multiplication

`/` pour la division normale

`//` pour la division entière (le résultat après la virgule est ignoré)

`%` pour le modulo (reste de la division normale)

`**` pour élever à la puissance (`x ** 3` signifie `x` au cube)

4.4.2 - Les opérateurs arithmétiques

`a = a + 3` peut aussi s'écrire `a += 3`

Affectation combinée avec un opérateur

```
a = 2 # a vaut 2

a += 5 # a vaut maintenant 7

a %= 3 # a vaut maintenant 1

a -= -1 # a vaut maintenant 2

a **= 4 # a vaut maintenant 16
```

4.4.3 - Les opérateurs bit à bit

`&` pour le ***et*** bit à bit

`|` pour le ***ou*** bit à bit

`~` pour le ***non*** bit à bit

`^` pour le ***xor*** bit à bit

`>>` pour le ***shift right***

`<<` pour le ***shift left***

Exemple : `a = 4 & 5`, ici `a` vaudra `4`

4.4.4 - Les opérateurs de comparaison

Exemple : `a < b` est vrai si `a` est inférieur à `b`

`==` pour l'égalité de valeur

`!=` pour la différence (non égalité)

`is` pour l'identité (les deux variables référencent le même objet)

`<` pour l'infériorité stricte

`<=` pour l'infériorité

`>` pour la supériorité stricte

`>=` pour la supériorité

`in` pour l'appartenance à un conteneur

4.4.5 - Opérateurs booléens

```
a = b and c

a = b or c

a = not b
```

L'essentiel : Les opérateurs

Opérateurs arithmétiques : `+`, `-`, `*`, `/`, `//`, `%`, `**`

Affectation combinée à un opérateur : `a += 3`

Opérateurs bit à bit : `&`, `|`, `~`, `^`, `>>`, `<<`

Opérateurs de comparaison : `==`, `!=`, **`is`**, `<`, `<=`, `>`, `>=`, **`in`**

Opérateurs booléens : **`and`**, **`or`**, **`not`**

4.5 - Types de base

4.5.1 - Les chaînes de caractères

4.5.1.1 - Les chaînes de caractères

Une chaîne de caractères est une séquence de caractères

En version simple, une chaîne de caractères est un texte

En anglais, se dit string

Les chaînes de caractères sont stockées en UTF-8

Exemple : `a = "hello world"`

4.5.1.2 - La concaténation dynamique

Concaténation dynamique

```
h = "hello "

w = "world!"

print(h + w)   # hello world!

w = "beautiful world!"

print(h + w)   # hello beautiful world!
```

4.5.1.3 - L'indexation classique

Python permet la même indexation que dans de nombreux langages

ATTENTION, les indices commencent à 0

ATTENTION, le dernier index est ***longueur - 1***

Indexation

```
word = "Python"

first_letter = word[0]

sixth_letter = word[5]
```

4.5.1.4 - L'indexation de Python

Python permet des choses que peu de langages supportent

Possibilité de compter à partir de la fin de la chaîne

Indexation

```
word = "Python"

first_letter = word[-6]

sixth_letter = word[-1]  # dernière lettre : indice -1
```

Les indices négatifs sont ajoutés à la longueur de la chaîne pour avoir les indices réels

4.5.1.5 - Le slicing de chaîne

Slicing de chaîne

```
word = "Python"

word[0:2]  # de 0 inclus, à 2 exclu => "Py"

word[2:6]  # de 2 inclus, à 6 exclu => "thon"

word[0:2] + word[2:6]  # => "Python"
```

4.5.1.6 - Le slicing de chaîne

Si le premier indice n'est pas donné, le slice démarre au début de la chaîne

Si le dernier indice n'est pas donné, le slice finit à la fin de la chaîne

Slicing de chaîne

```
word = "Python"

word[:2]  # du début, à 2 exclu => "Py"

word[2:]  # de 2 inclus, à la fin => "thon"

word[:2] + word[2:]  # => "Python"
```

4.5.1.7 - Slicing

Le slicing ne s'applique pas qu'aux chaînes

Le slicing s'applique à tous les éléments itérables

- Chaînes de caractères
- Listes
- Tuples
- Vue de dictionnaires
- …

4.5.1.8 - Méthodes

Python fournit de quoi manipuler les chaînes de caractères

Passer en majuscules

```
ma_chaine = "Vincent !"

print(ma_chaine.upper())   # VINCENT !
```

4.5.1.9 - Immuabilité

Les chaînes de caractères sont immuables

L'immuabilité des chaînes en pratique

```
ma_chaine = "Vincent !"
print(ma_chaine)   # Vincent !


ma_chaine.upper()   # ma_chaine n'est pas modifiée
print(ma_chaine)   # Vincent !


ma_chaine = ma_chaine.upper()   # ma_chaine est modifiée
print(ma_chaine)   # VINCENT !
```

4.5.1.10 - Et...

```
chaine.upper()

chaine.startswith() /
chaine.endswith()

chaine.strip() / chaine.lstrip() /
chaine.rstrip()

chaine.split() / chaine.join()

chaine.replace()
```

L'essentiel : Les chaînes de caractères

Une chaîne de caractères est un texte (en UTF-8)

Python permet la même indexation que dans de nombreux langages (les indices commencent à 0)

Les indices négatifs sont ajoutés à la longueur de la chaîne pour avoir les indices réels

Si le premier indice n'est pas donné, le slice démarre au début de la chaîne

Si le dernier indice n'est pas donné, le slice finit à la fin de la chaîne

Le slicing s'applique à tous les éléments itérables

```
chaine.upper()
```

4.5.2 - Les listes

4.5.2.1 - Les listes

Série d'éléments

Ordonnée

Indexable

Notée entre crochets

4.5.2.2 - Les listes

Les éléments d'une liste peuvent être de nature différente

- nombre
- chaîne de caractères
- liste
- ...

Listes

```
a = []  # liste vide
b = [1, "deux", 3.0]
c = [["4", 5], True, None]
d = list(range(1, 10, 2))  # [1, 3, 5, 7, 9]
```

4.5.2.3 - Indexation

Les listes s'indexent comme n'importe quel itérable, comme les chaines de caractères

Indexation de liste

```
# lst est une variable de type liste


lst[0]       # premier élément
lst[-1]      # dernier élément
lst[12:]     # sous-liste comprise entre indice 12 inclus et le dernier
lst[:3]      # 3 premiers éléments
lst[3:25:2]  # un élément sur 2 entre l'indice 3 inclus et l'indice 25 exclu
```

4.5.2.4 - Fonctions utiles

`len(ma_liste)` renvoie le nombre d'éléments

`sum(ma_liste)` renvoie la somme des éléments

`min(ma_liste)` renvoie le minimum des éléments

`max(ma_liste)` renvoie le maximum des éléments

`sorted(ma_liste)` renvoie une « liste » triée (un itérateur)

`sorted(ma_liste, reverse=True)` renvoie une « liste » triée en ordre inverse

`reversed(ma_liste)` renvoie une « liste » inversée (un itérateur)

4.5.2.5 - Méthodes utiles

`ma_liste.append(a)` ajoute l'élément `a` à la fin de la liste

`ma_liste.index(a)` renvoie l'indice de `a` dans la liste

`ma_liste.pop()` supprime et retourne le dernier élément de la liste

`ma_liste.sort()` trie la liste

`ma_liste.reverse()` inverse la liste

L'essentiel : Les listes

Liste d'éléments, ordonnée, indexable, notée entre `[]`

Les éléments d'une liste peuvent être de nature différente

Les listes s'indexent comme n'importe quel itérable, comme les chaines de caractères

Fonctions : `len()`, `sum()`, `min()`, `max()`, `sorted()`, `reversed()`...

Méthodes : `append()`, `index()`, `pop()`, `sort()`, `reverse()`

L'essentiel : Types de base

§ Les chaînes de caractères

§ Les listes

4.6 - Flot de contrôle

4.6.1 - if

```
if
    if condition1:
        instruction si la condition1 est vraie
    elif condition2:
        instruction si la condition1 est fausse
        et la condition2 est vraie
    else:
        instruction si les deux conditions sont fausses
```

4.6.2 - while

while est la boucle **while** habituelle du C, C++, Java, JavaScript…

Utilisation du while

```
i = 0
while i < 4:
    print("Python", i)
    i += 1
# affichera :
# Python 0
# Python 1
# Python 2
# Python 3
```

4.6.3 - for

Ce n'est pas la boucle `for` classique du C, C++, Java, JavaScript

C'est le `foreach` du PHP ou du C#

Permet d'agir sur chaque élément d'une séquence

for

```
for valeur in 1, "deux", 3.0:
    print(valeur)
```

4.6.4 - break

break interrompt une boucle

break

```
for i in range(10):
    if i > 2:
        break
    print(i)
# affichera :
# 0
# 1
# 2
```

4.6.5 - continue

continue saute au prochain tour de boucle

continue

```
for i in range(5):
    if i % 2:
        continue
    print(i)
# affichera :
# 0
# 2
# 4
```

4.6.6 - Fonctions

4.6.6.1 - Les fonctions

Bouts réutilisables de logiciel

Comme les fonctions mathématiques (cos, sin, ln…)

- Peuvent prendre zéro, un ou plusieurs paramètres
- Peuvent renvoyer une ou plusieurs valeurs

4.6.6.2 - Définition d'une fonction

Une fonction est définie avec le mot clé **def**

return permet de renvoyer la valeur résultat

Si aucun **return**, un **return None** est implicitement exécuté

Définition de fonction

```
def add(x, y):
    return x + y



def carre_cube(x):
    return x ** 2, x ** 3  # plusieurs valeurs
```

4.6.6.3 - Passage des paramètres

Paramètres

```
def add(x, y):
    return x + y



add(1, 2)
add(1, y=2)
add(x=1, y=2)
add(y=2, x=1)


# INTERDIT ! les paramètres nommés doivent être à la fin
add(x=1, 2)
```

L'essentiel : Fonctions

Les fonctions sont des bouts réutilisables de logiciel

Prendre des paramètres, renvoyer des valeurs

Une fonction est définie avec le mot clé `def`

`return` permet de renvoyer la valeur résultat

Paramètres positionnels et paramètres nommés

L'essentiel : Flot de contrôle

`if condition1:` + indentation

`elif condition2:` + indentation

`else:` + indentation

`while condition:` + indentation

`for element in sequence:` + indentation

`break` interrompt une boucle

`continue` saute au prochain tour de boucle

§ Fonctions

L'essentiel : Les bases de Python

Les commentaires aident à mieux comprendre le code pour les développeurs

Commencent par #, finissent à la fin de la ligne

§ Les variables

§ Les opérateurs

§ Types de base

§ Flot de contrôle

5 - MicroPython

5.1 - main.py

boot.py lance main.py

main.py est le point d'entrée de notre programme

S'il finit : REPL

Peut contenir tout notre logiciel

5.2 - machine

Dans boot.py : `import machine`

Code bas niveau pour utiliser le microcontrôleur

Ici : STM32F407VGT6

5.3 - pyb

Dans boot.py : `import pyb`

- Code bas niveau pour utiliser la carte électronique
- Ici : STM32F4 Discovery

C'est généralement notre point de départ

Mais spécifique à chaque carte

- API variable d'une carte à l'autre

5.4 - LED

5.4.1 - LED

4 LED sur la carte

`pyb.LED(1)` à `pyb.LED(4)`

```
une_led = pyb.LED(3)
    une_led.on()
    une_led.off()
    une_led.toggle()
```

`pyb.delay(1000)` attend 1000 millisecondes

5.4.2 - TP

5.4.2.1 - TP : chenillard

Faites un chenillard tournant avec une LED allumée

80 ms d'allumage par LED

Faites un chenillard tournant avec deux LEDs allumées

150 ms d'allumage par LED

Pour les plus avancés, faites osciller la vitesse de rotation

L'essentiel : LED

```
pyb.LED(1)

une_led.on(),une_led.off(),
une_led.toggle()

pyb.delay(1000)
```
attend 1000 millisecondes

5.5 - UART

5.5.1 - UART

Universal Asynchronous Receiver-Transmitter

Composant électronique

Gère des ***liaisons séries*** asynchrones

La version synchrone est USART

> Universal Synchronous and Asynchronous Receiver-Transmitter

5.5.2 - Configuration

Baud rate : vitesse de communication

Taille des mots

Nombre de bits de stop

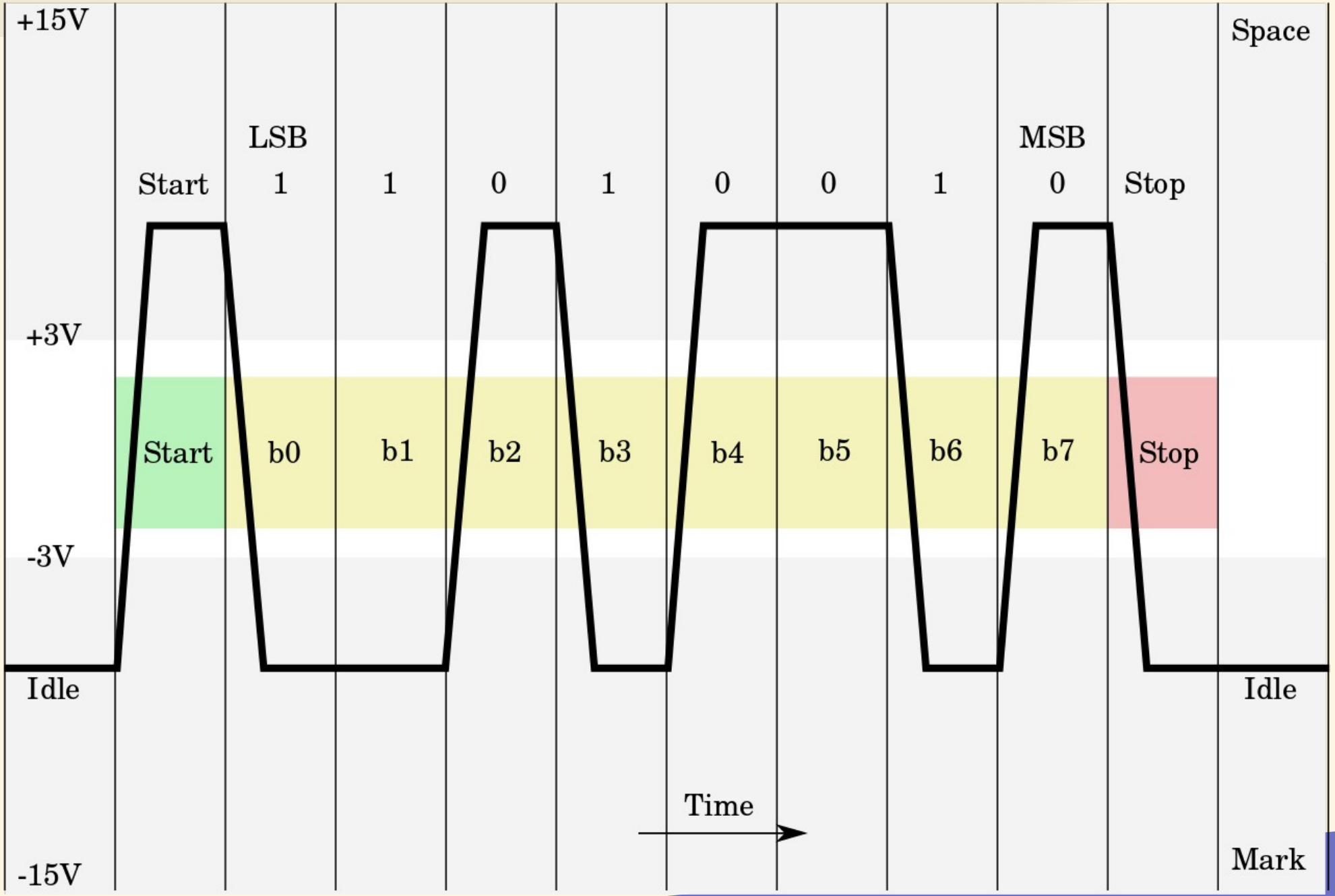
- Marque la fin d'un mot
- Plus le nombre est grand, plus il est facile de se synchroniser
- Plus le nombre est grand, plus le baud rate utile est faible

Parité

9600 8N1 : 9600 bits/seconde, 8 bits/mot, pas de parité, 1 bit de stop

- Configuration très classique

5.5.3 - 0x4B en RS-232



5.5.4 - RS-232

Protocole datant de 1960

Énormément utilisé

Encore aujourd'hui

Simple

Supporté par plein de matériels/logiciels

Protocole des ports série de PC

Remplacé par l'USB

5.5.5 - Limitations

Pratique à utiliser, mais

« Grosses tensions », consommation de puissance, et ralentissement de la vitesse

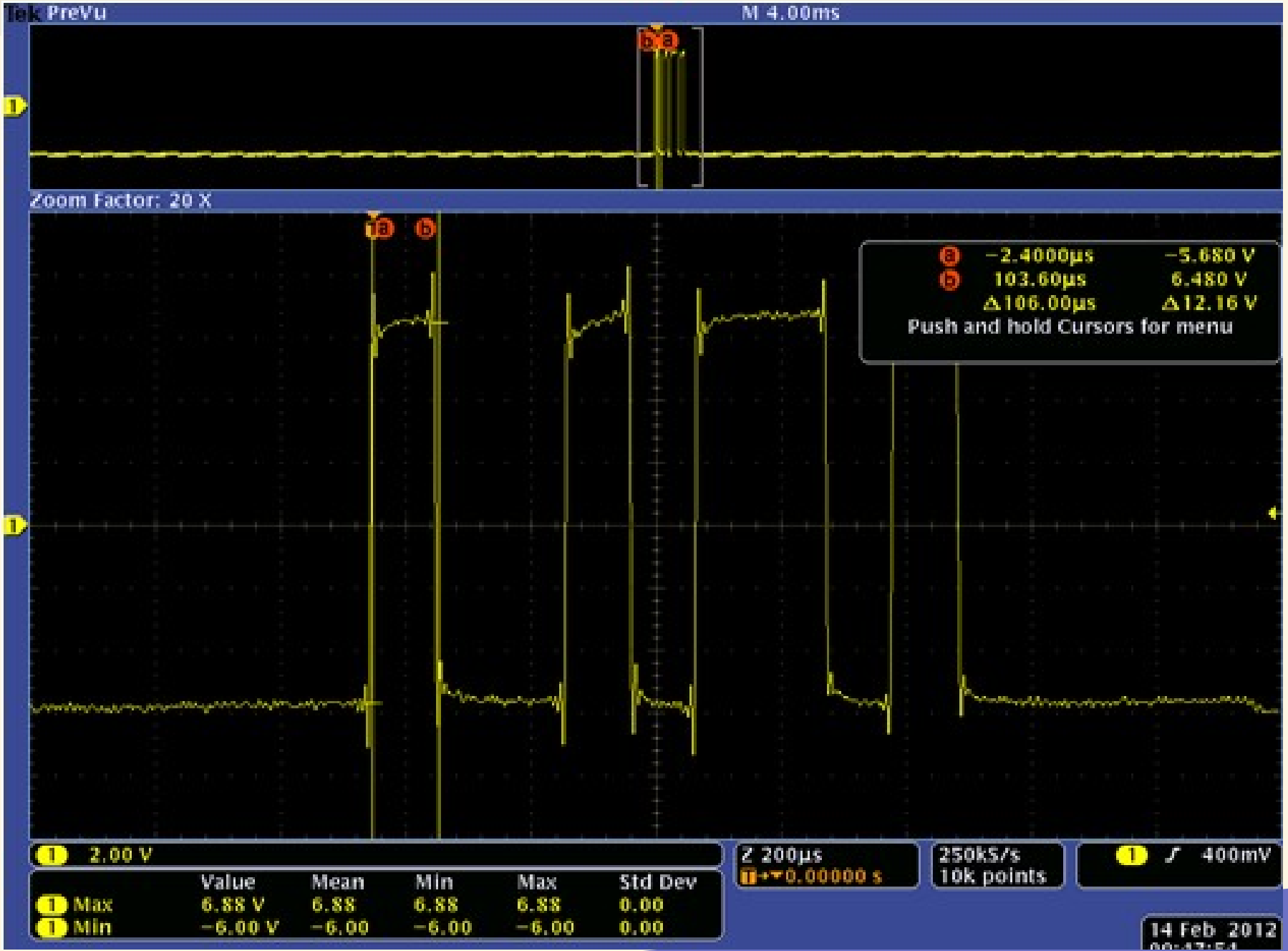
Masse partagée, pas différentiel, limite la robustesse au bruit et la distance de communication

Câbles NULL modem non standardisé (pour connecter deux PC)

Pas de transmission d'alimentation (arrive à faire fonctionner une souris)

Cela convient à plein d'applicati

5.5.6 - RS-232



5.5.7 - RS-232

5.5.8 - pyb.UART

Utilisation de l'UART

```
from pyb import UART


# UART STM32F4 Discovery Shield
numero_uart = 2


uart = UART(numero_uart)
uart.init(9600, bits=8, parity=None, stop=1)
```

5.5.9 - pyb.UART

`pyb.UART.init` prend comme paramètres :

- `baudrate` : vitesse de communication en bits par seconde
- `bits` : nombre de bits par mots (7, 8 ou 9)
- `parity` : **`None`**, `0` pour paire, `1` pour impaire
- `stop` : nombre de bits de stop (1, 2)

5.5.10 - pyb.UART

`uart.read()` : lit tous les octets reçus

`uart.read(10)` : lit au plus 10 octets

`uart.readinto(buf)` : lit tous les octets reçus et les stocke dans `buf`

Évite l'allocation dynamique

Ne fait pas de dépassement de capacité dans `buf` (taille respectée)

`uart.readinto(buf, 10)` : au plus 10 octets

`uart.readline()` : jusqu'à un retour à la ligne

`uart.write()` : pour envoyer des octets

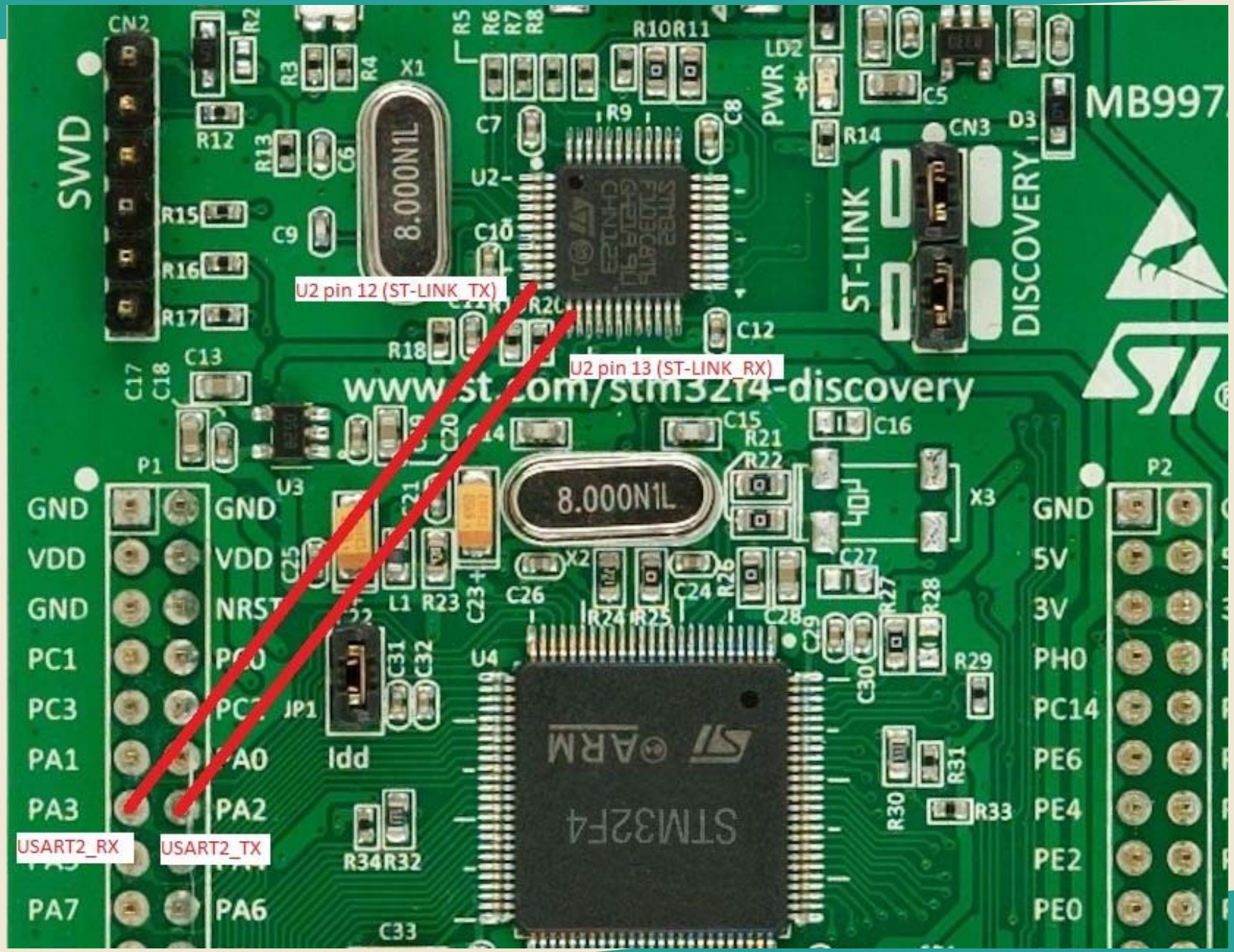
5.5.11 - TP

5.5.11.1 - TP

Envoyez un caractère sur l'UART

Observez le avec un oscilloscope

5.5.11.2 - TP



5.5.11.3 - TP

Affichez du texte sur la liaison série

Faites un driver VT100, et testez le

```
clear_screen()

move(x, y)

...
```

L'essentiel : UART

UART : liaisons séries asynchrones

USART : liaisons séries synchrones (et asynchrones)

Paramétrage : baud rate, taille des mots, nombre de bits de stop

RS-232 : protocole série souvent utilisé

Limitations : forte puissance électrique, masse partagée, pas de transmission d'alimentation

```
uart = pyb.UART(numero_uart)

uart.init(9600, bits=8, parity=None, stop=1)

uart.read(),uart.write()
```