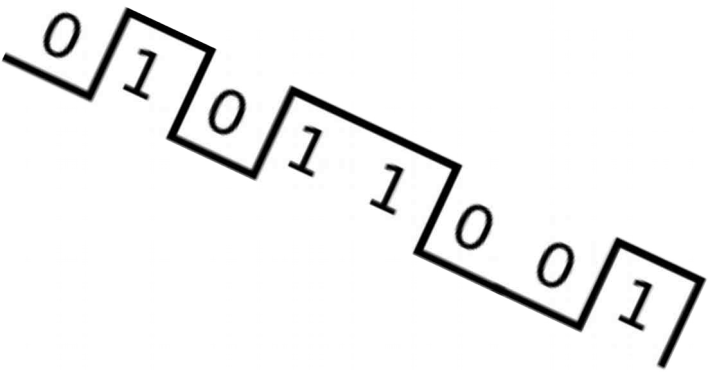
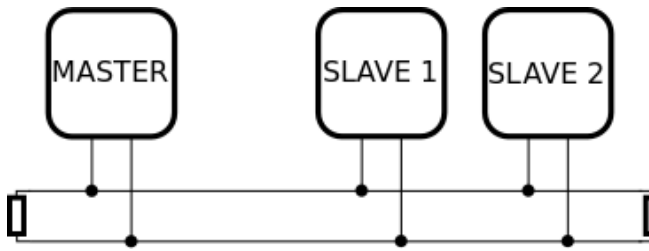
Module Réseau 2 : Préparation du projet Communication Ethernet entre PC(QT) et Microcontrôleur PIC32



Bertrand VANDEPORTAELE
IUT GEII Toulouse
2019

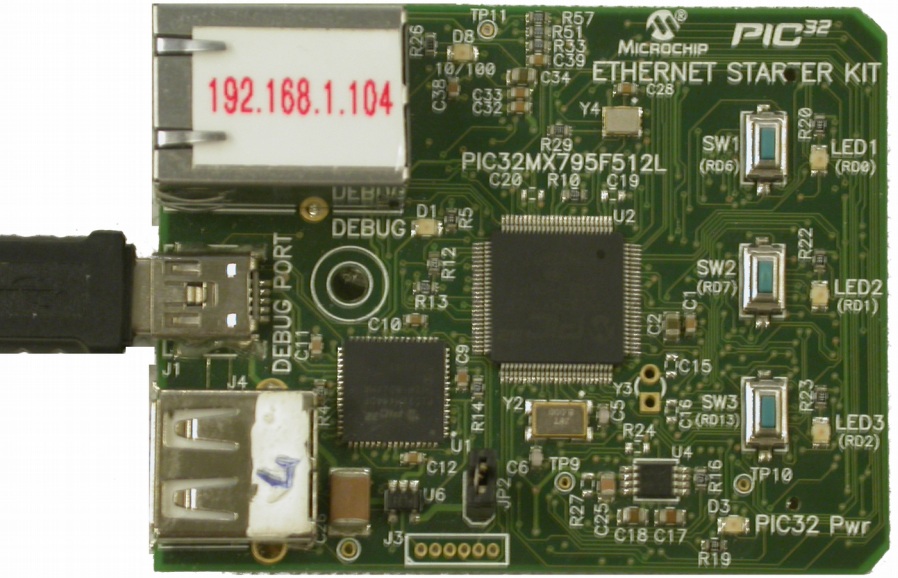


Objectif et déroulement du projet

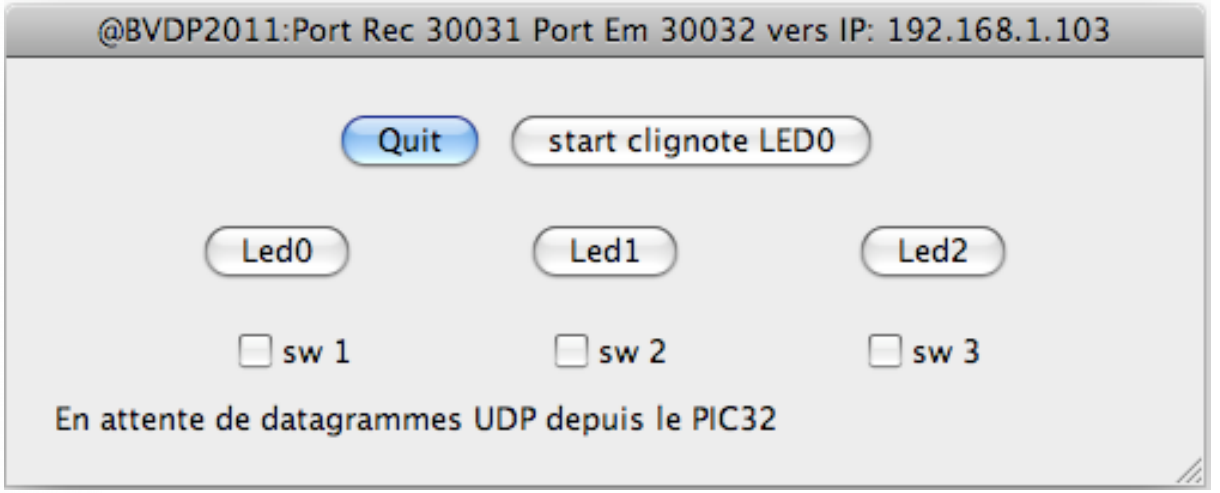
- 2h de CM: présentation
- 2h de CM: Programmation Orientée Objet C++
- 2*2h de TP: Programmation du microcontrôleur PIC32
- 2*2h de TP: Programmation du PC sous Qt

Objectif et déroulement du projet

Mise en réseau de plusieurs cartes et PC



Carte microcontroleur

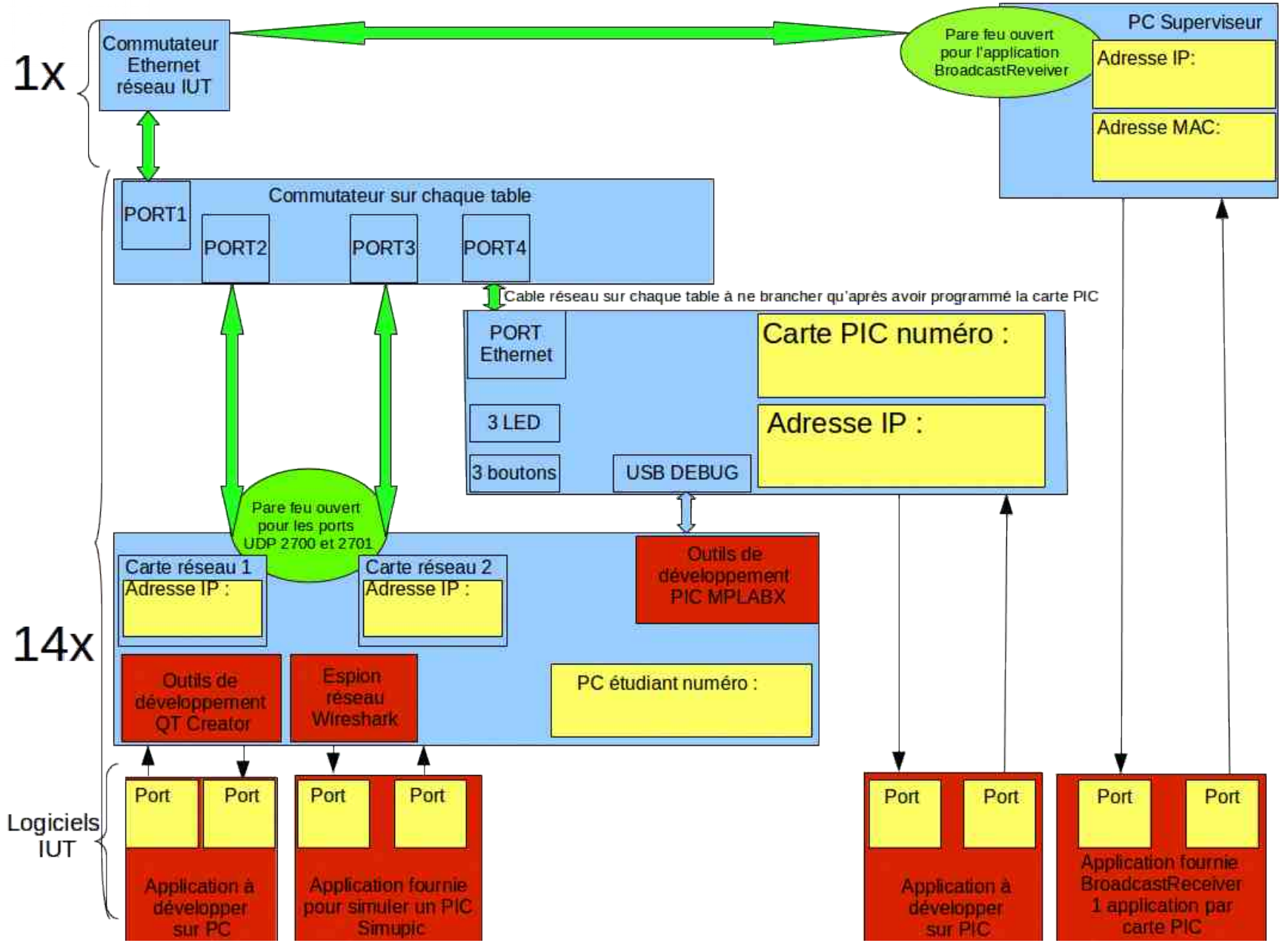


Interface graphique de l'application sur PC

Objectif et déroulement du projet

- Établir une communication entre des dispositifs «industriels» de type micro-contrôleurs et des ordinateurs via Ethernet
- 1 PC superviseur commun à tous les étudiants
  - Utilisé dans un premier temps pour que vous n'ayez à traiter que la carte microcontrôleur
- Puis 1 PC par étudiant
  - A vous de faire une application graphique "moderne"
- Les PC utilisent :
  - 2 adaptateurs réseau sur chaque pc pour séparer les trafics
  - 1 même lien physique à travers un VLAN mais équivalent à de 2 réseaux distincts pour l'utilisateur

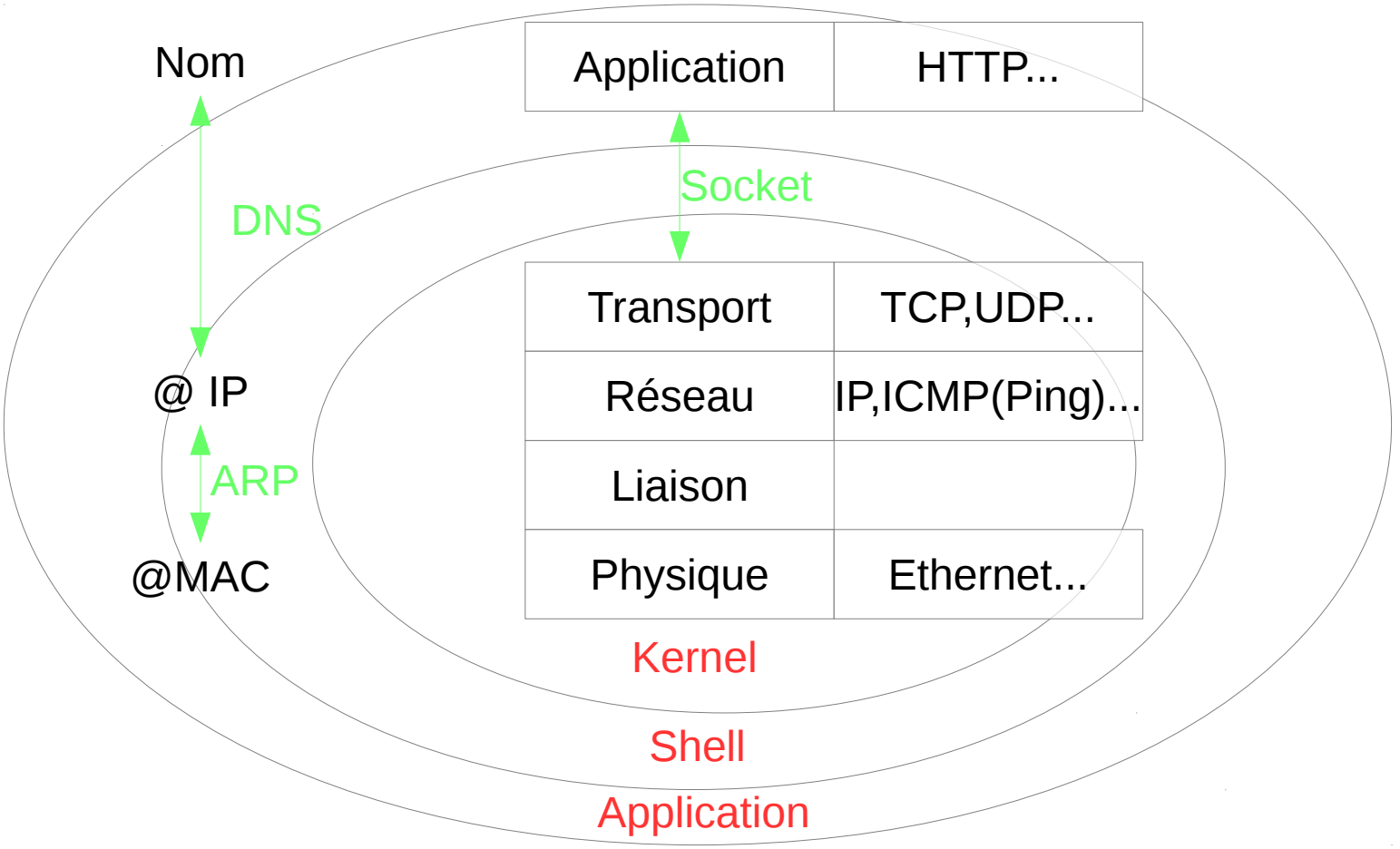
Présentation des réseaux



Socket

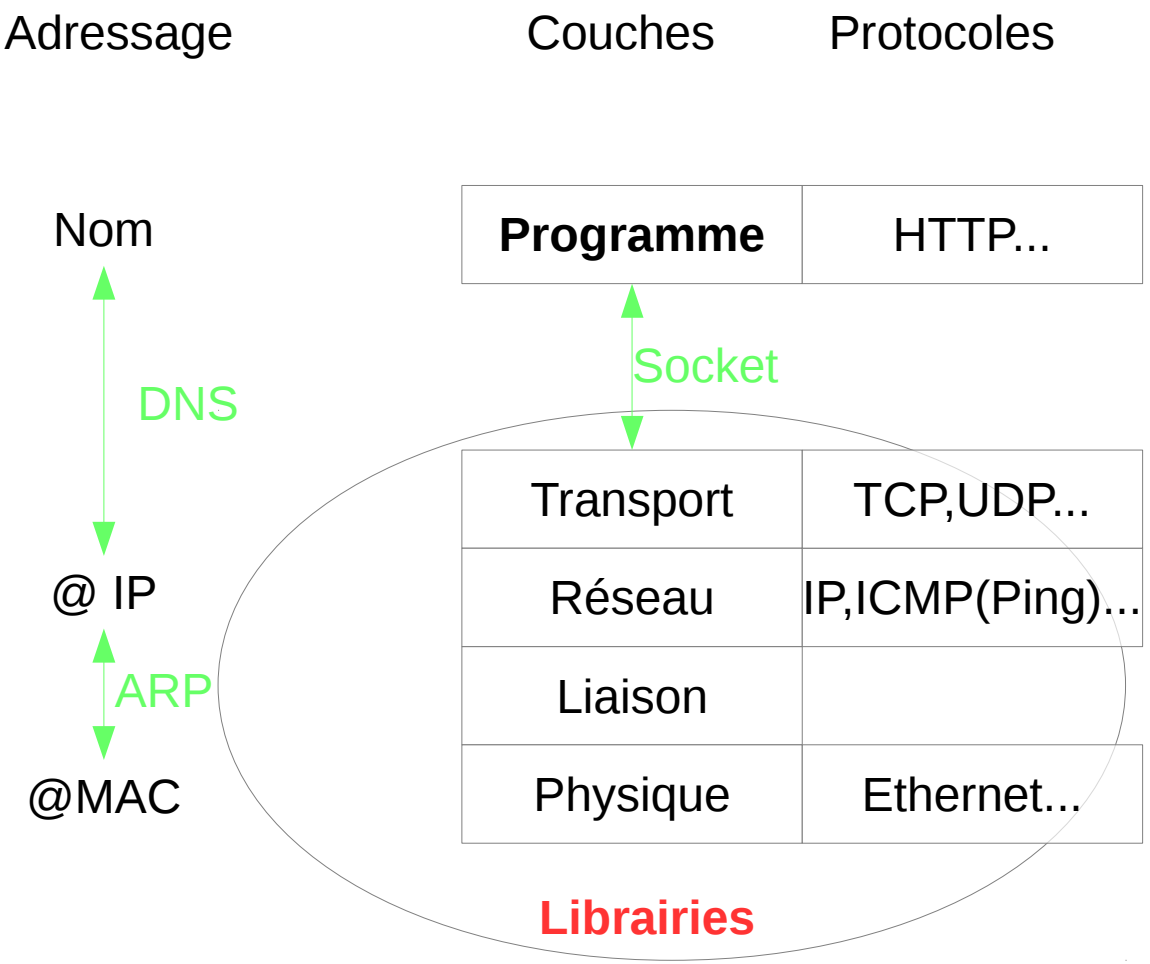
- Vision pour le développeur d'une application **avec** système d'exploitation :

Adressage Couches Protocoles



Socket

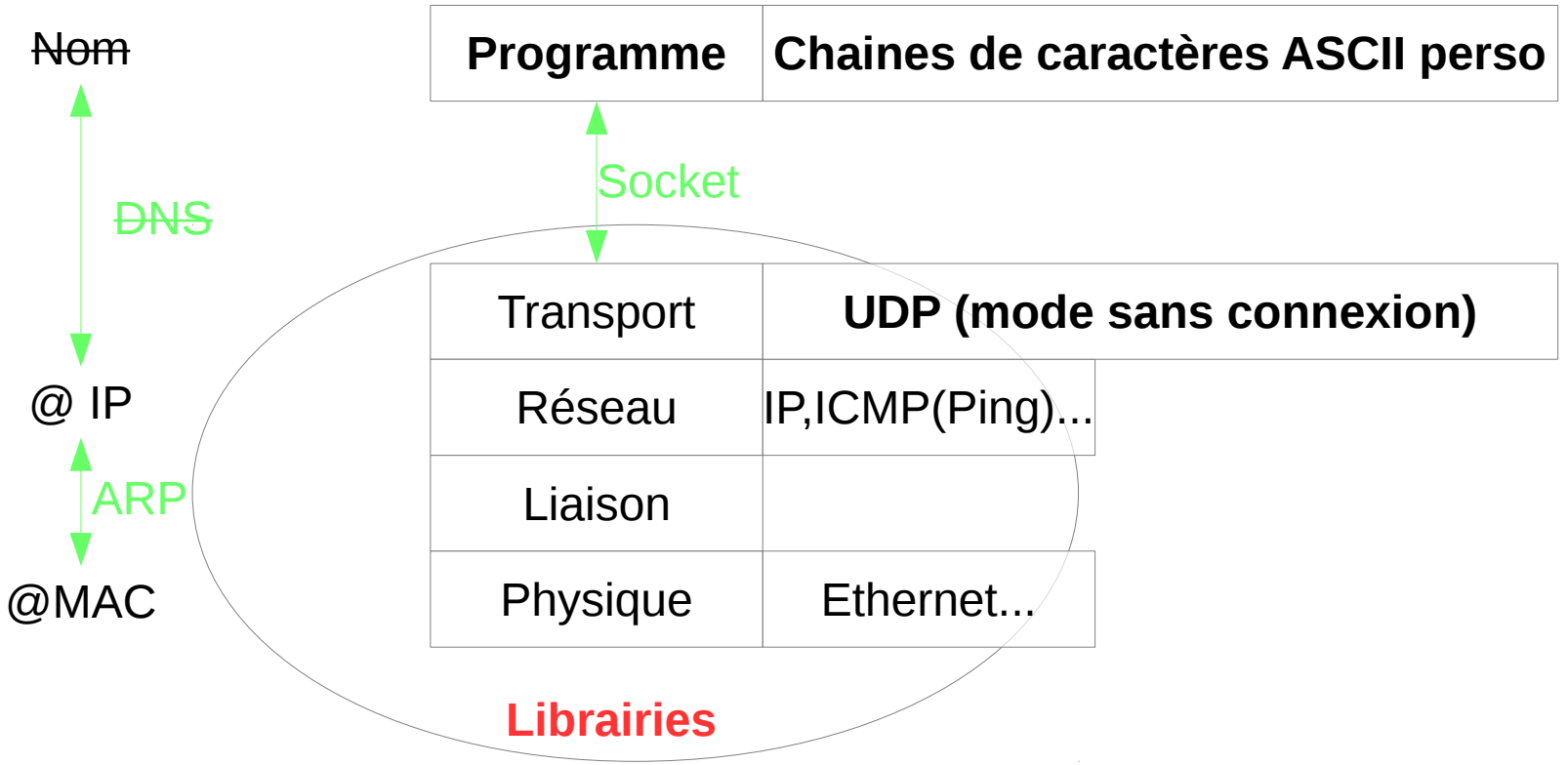
- Vision pour le développeur d'une application **sans** système d'exploitation :



Socket

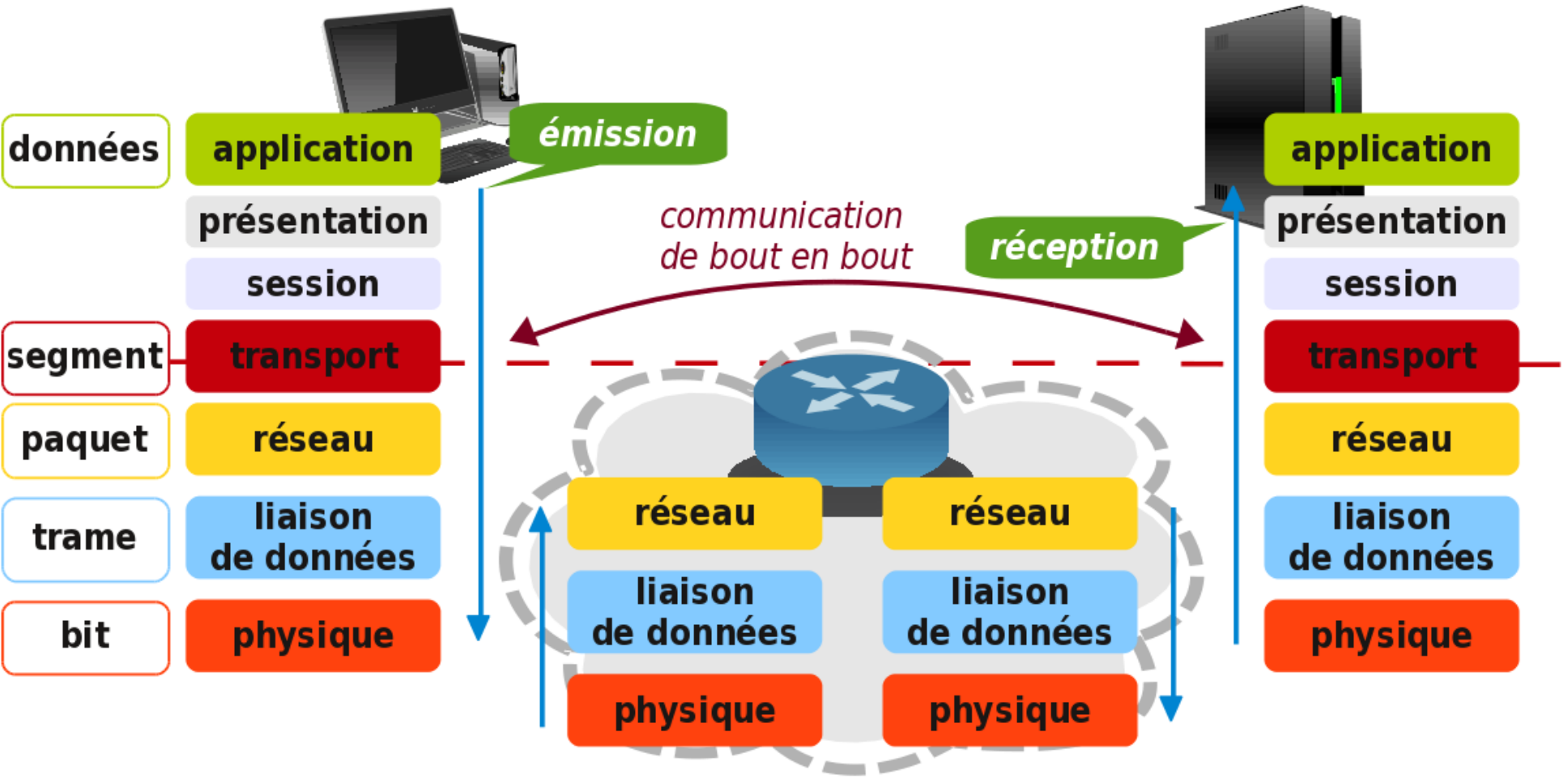
- Vision pour le développeur d'une application **sans** système d'exploitation **pour le TP**:

Adressage Couches Protocoles



Socket

- Vision du système:

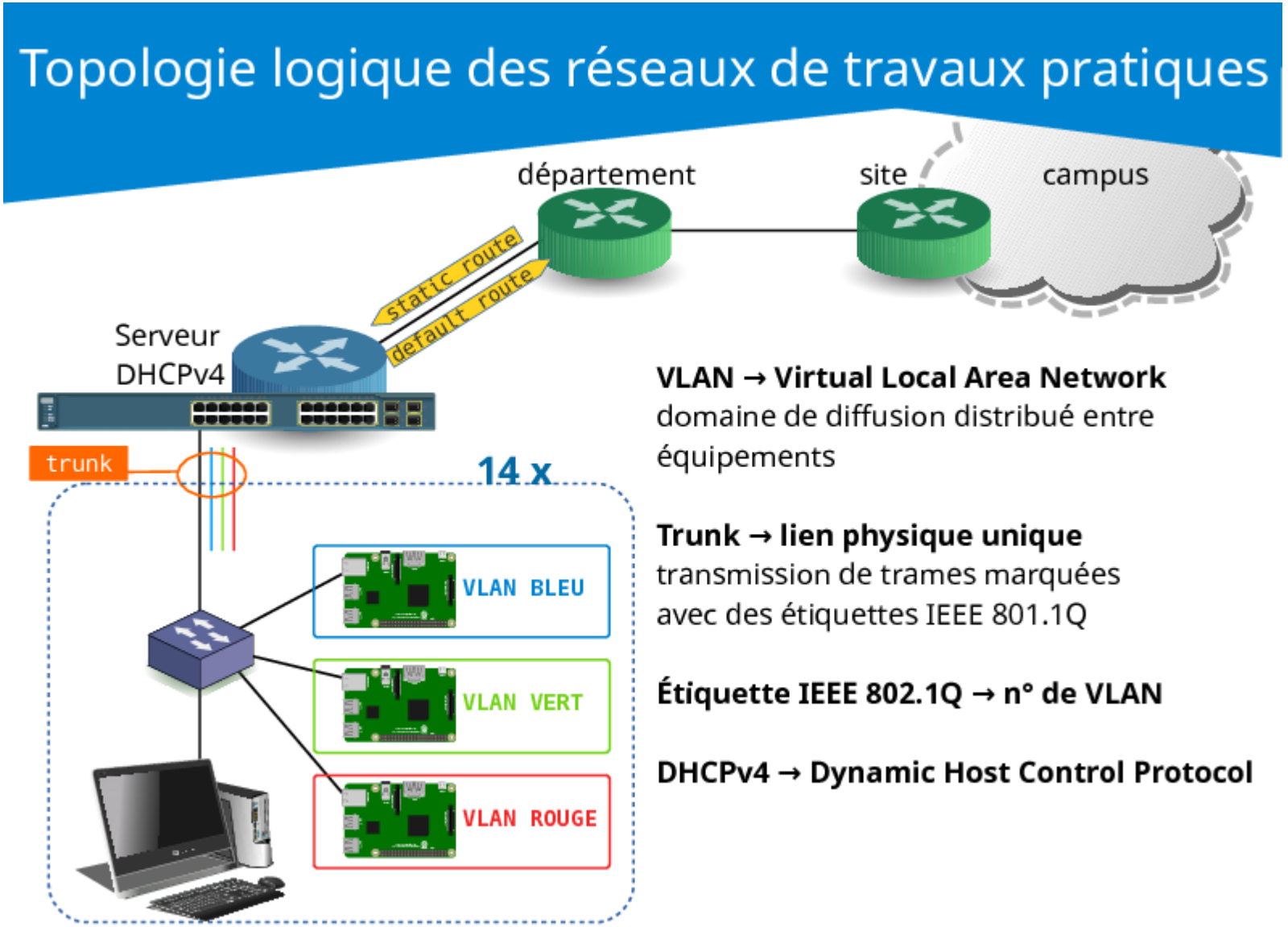


Éventuellement les 2 applications peuvent fonctionner sur la même machine :
c'est un moyen simple pour faire de la communication entre plusieurs processus
(et exploiter une architecture multicoeurs)

Socket

- Vision du système pour notre application:

VLAN : Réseau local virtuel



VLAN → Virtual Local Area Network
domaine de diffusion distribué entre équipements

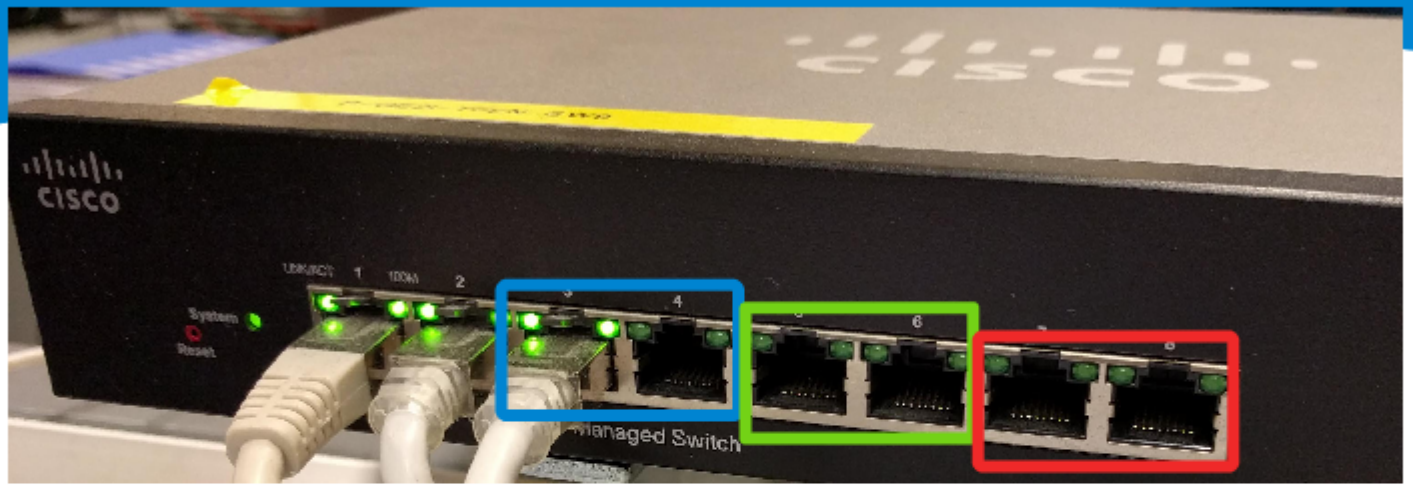
Trunk → lien physique unique
transmission de trames marquées avec des étiquettes IEEE 801.1Q

Étiquette IEEE 802.1Q → n° de VLAN

DHCPv4 → Dynamic Host Control Protocol

VLAN

Salle EREN



VLAN BLEU

VLAN n°1600 – Préfixe réseau 172.16.0.0/26 – Routeur 172.16.0.1
Plage DHCP 172.16.0.34 → 62

VLAN VERT

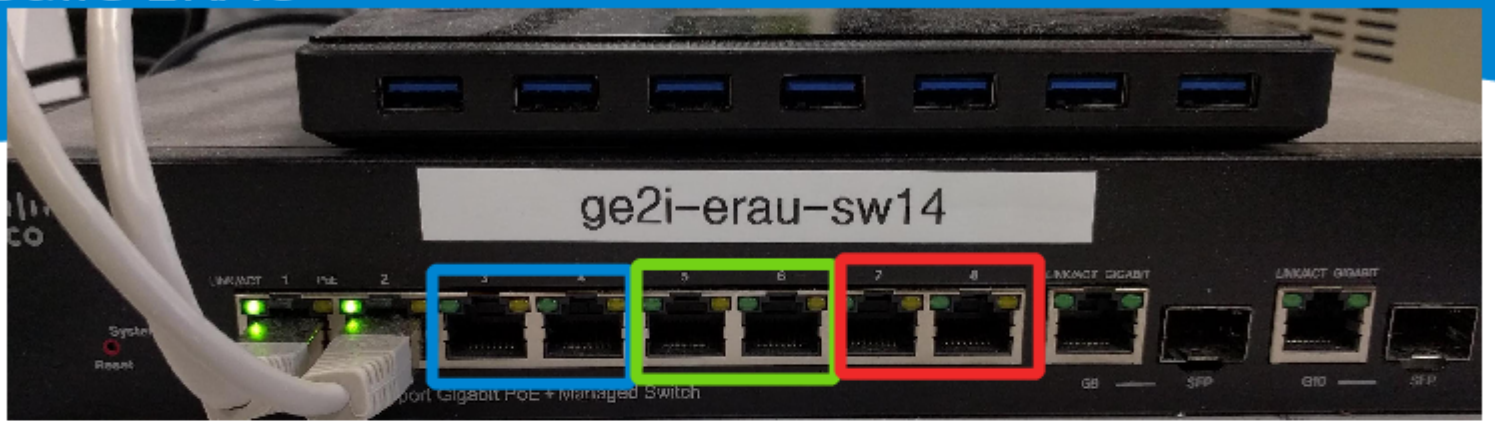
VLAN n°1601 – Préfixe réseau 172.16.1.64/26 – Routeur 172.16.1.65
Plage DHCP 172.16.1.98 → 126

VLAN ROUGE

VLAN n°1602 – Préfixe réseau 172.16.2.128/26 – Routeur 172.16.2.129
Plage DHCP 172.16.2.162 → 190

VLAN

Salle ERAU



VLAN BLEU

VLAN n°1606 – Préfixe réseau 172.16.6.0/26 – Routeur 172.16.6.1
Plage DHCP 172.16.6.34 → 62

VLAN VERT

VLAN n°1607 – Préfixe réseau 172.16.7.64/26 – Routeur 172.16.7.65
Plage DHCP 172.16.7.98 → 126

VLAN ROUGE

VLAN n°1608 – Préfixe réseau 172.16.8.128/26 – Routeur 172.16.8.129
Plage DHCP 172.16.8.162 → 190

Détermination des adresses

- Nommer les hôtes
  - Requiert un annuaire (serveur DNS), non utilisé dans le TP
- adresses IP v4
  - Attribution dynamique pour les PC par un serveur DHCP (Bauds statiques ou dynamiques)
  - Configuration statique pour les cartes microcontrôleurs
  - /26 → 26bits pour le n° de réseau et 6 pour l'hôte
  - Adressage logique, une adresse est attribuée par choix
  - Utilisée pour permettre l'adressage hors du réseau local grâce au routage
- Adresses MAC
  - Adressage matériel :Gravées dans les composants
  - Utilisée pour identifier les hôtes sur le réseau local
- Table ARP
  - Stocke les correspondances IP/MAC après interrogation par requête ARP

    ? (192.168.1.28) à 50:e5:49:3d:e3:13 [ether] sur enp0s25

Protocole

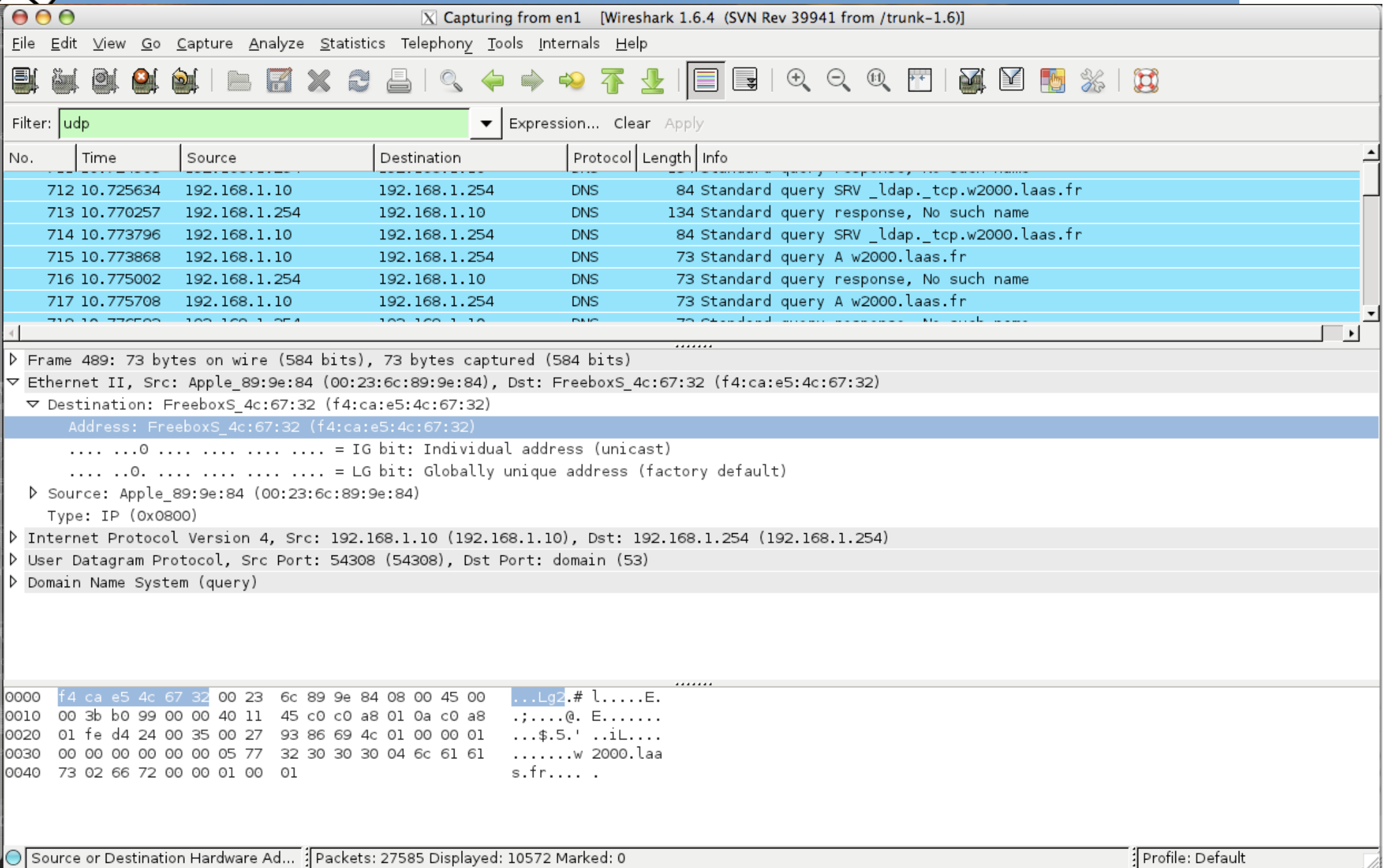
- Communication de bout en bout à l’aide de sockets UDP
  - Choix d’une socket par sens de communication pour séparer les trafics
  - Numéros de ports différents pour chaque carte pour que le PC superviseur puisse distinguer les trafics provenant des différentes cartes

Protocole

- Échange via des chaînes de caractères ASCII
- État des boutons envoyé par la carte microcontrôleur :
  - **sprintf(chaine,"compteur= %6d b1:%d b2:%d b3:%d\n",compteuretudiant,b1state,b2state,b3state);**
  - Décodage (Buffer Parsing) à l'aide de la fonction **sscanf**
  - Comptage du nombre de caractères contenus dans la chaîne avec la fonction **strlen**

    int strlen(char* chainecarac);
- Demande de commutation des LED envoyée par le PC :
  - **sprintf(chaine,"%d",i) ;**
  - Décodage (Buffer Parsing) en interprétant le contenu des cases du tableau stockant les caractères reçus

Outils d'analyse réseau : Wireshark



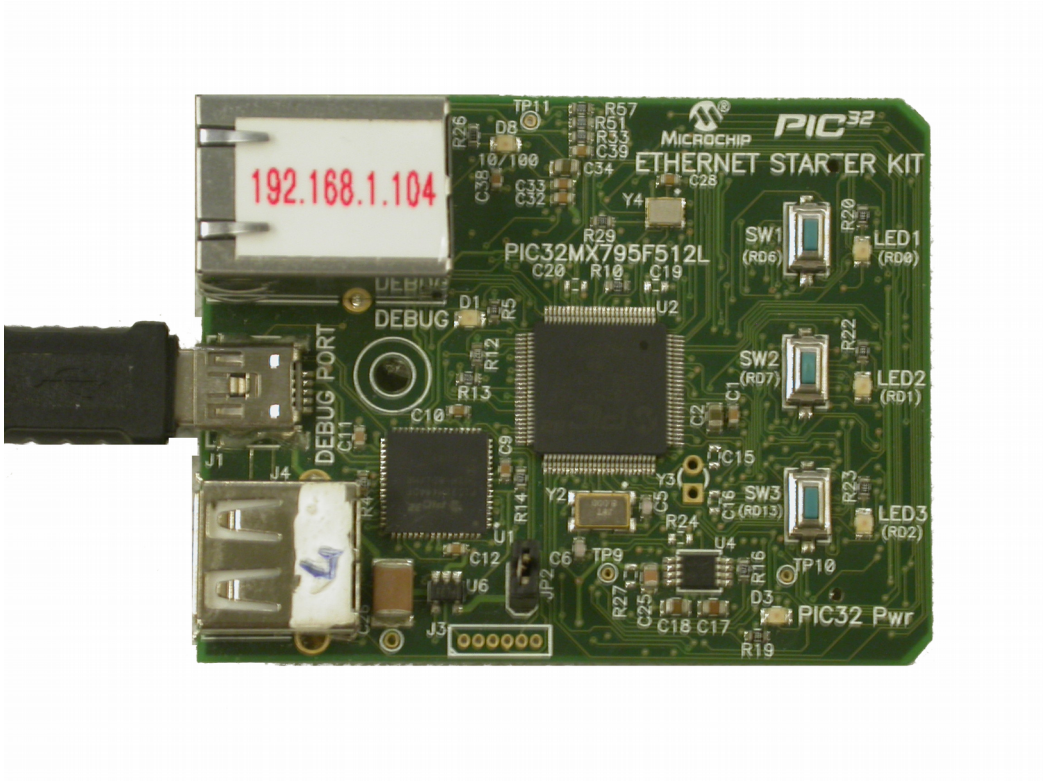
Plateforme matérielle microcontrôleur

- Carte de développement « PIC32 Ethernet Starter Kit »
- Microcontrôleur 32 bits
- Pas de système d’exploitation
  - Application monolithique : 1 binaire contenant le main() et toutes les librairies
- Pas de MMU (Memory Management Unit)
  - Attention aux fuites mémoires

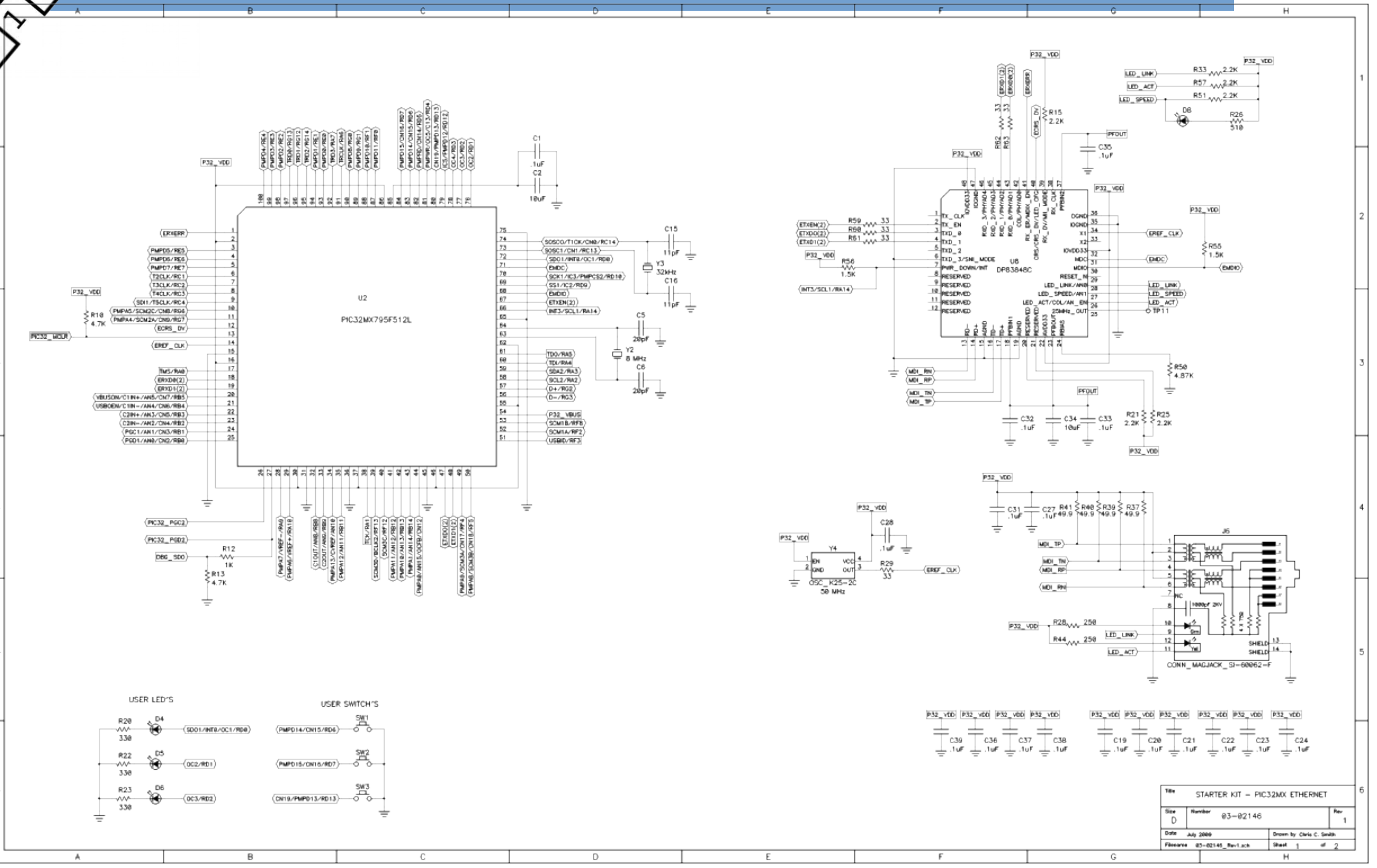
Plateforme matérielle microcontrôleur

- Carte de développement « PIC32 Ethernet Starter Kit »
- Grand nombre d'interfaces de communication dont
  - Interfaces USB
    - Device (bootloader, alimentation et debug)
    - OTG (Host+Device)
  - Interface ethernet intégrée
    - MAC (Media Access Control) interface matérielle intégrée dans le microcontrôleur (niveau 2 OSI : couche liaison)
    - PHY (PHYsique) dans un composant externe DP83848C sur la carte de développement
- Librairie de fonctions
  - pile TCP/IP Microchip

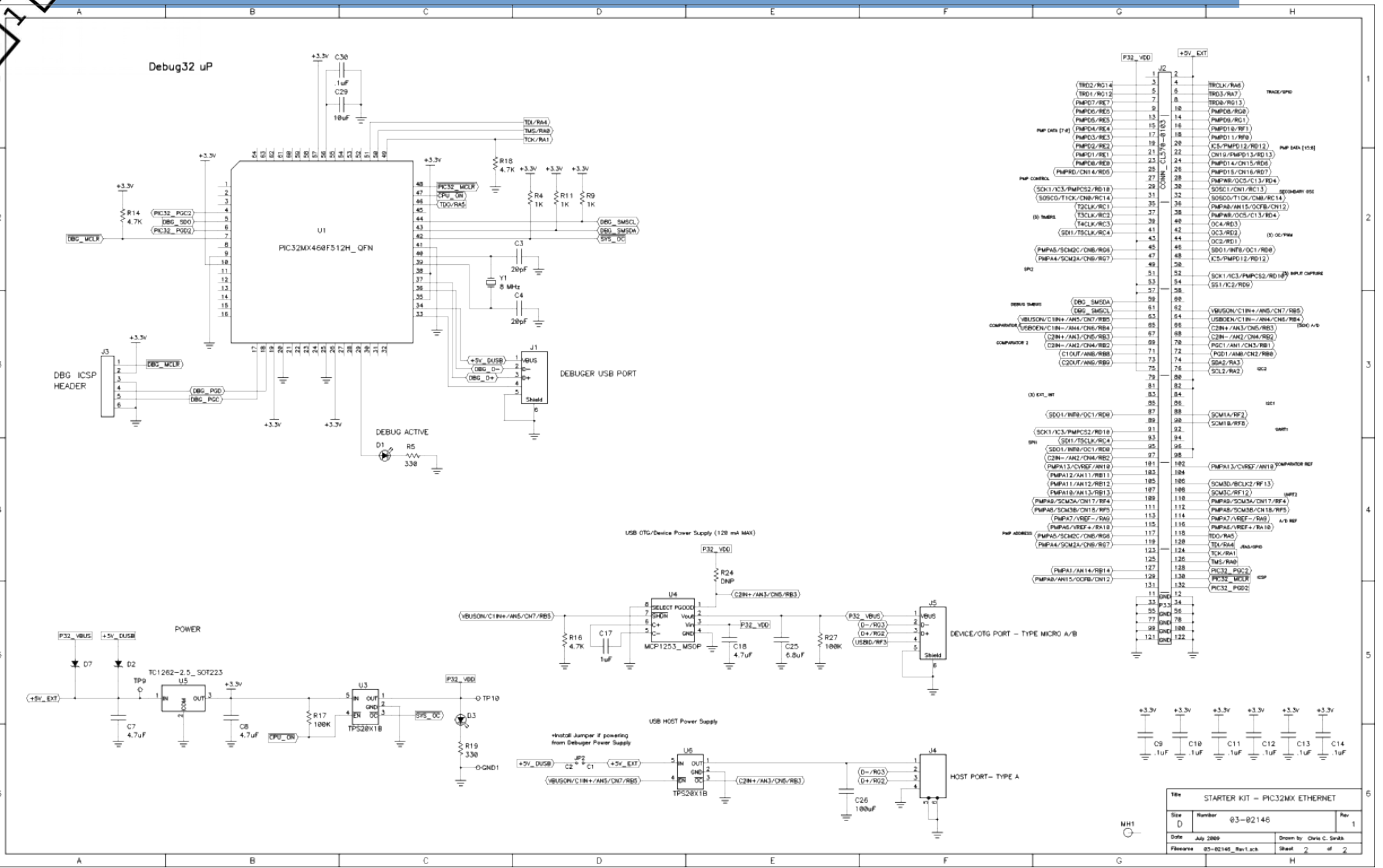
Plateforme matérielle microcontrôleur



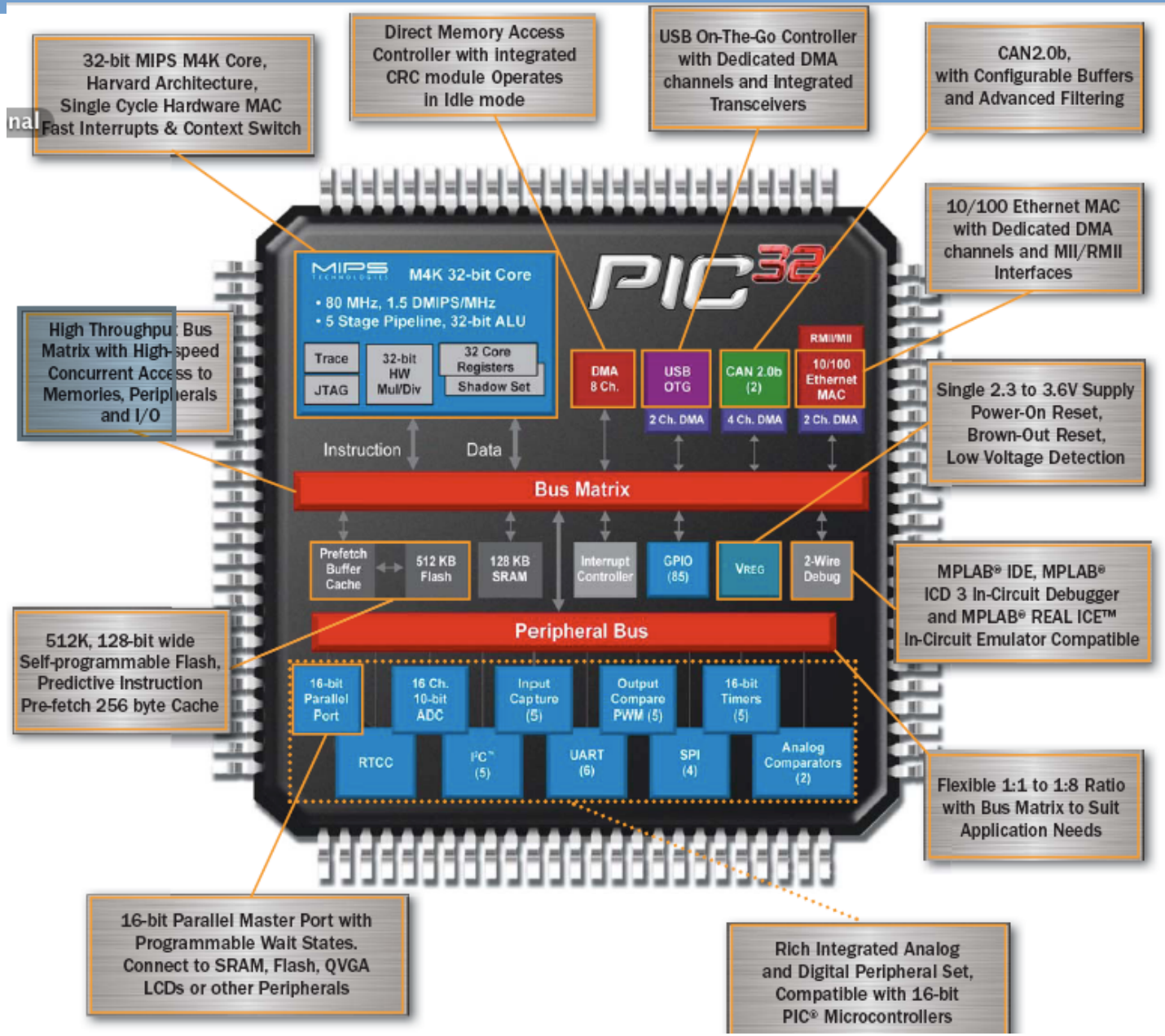
Plateforme matérielle microcontrôleur



Plateforme matérielle microcontrôleur



Plateforme matérielle microcontrôleur



Outils de développement MPLABX

- Démo de l’outils
  - CTRL+click gauche !
- Présentation de la structure des fichiers du projet
- Fonctions à implémenter (équivalent à setup() et loop() sur Arduino) :
  - void etudiantSocketAppInit();
  - void etudiantSocketAppTask();

Définitions de types

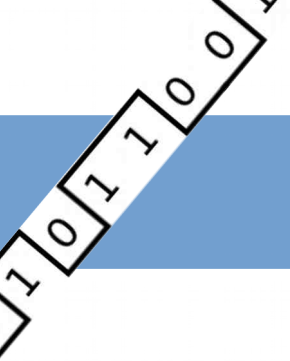
- Définitions de types “standard”

```
typedef signed int          INT;
typedef signed char         INT8;
typedef signed short int    INT16;
typedef signed long int     INT32;
typedef unsigned int        UINT;
typedef unsigned char       UINT8;
typedef unsigned short int  UINT16;
```

Définitions de types

- Définitions de types “alternatifs”

```
typedef char                CHAR8;
typedef unsigned char       UCHAR8;
typedef unsigned char       BYTE;
typedef unsigned short int   WORD
typedef unsigned long       DWORD;
```



Définitions de types

- Définitions de types pour divers usages

  UDP_PORT

  NODE_INFO

  UDP_SOCKET

Type Union pour manipuler les @IP v4

```
• typedef union
  {
     UINT32 Val;
     UINT16 w[2] __PACKED;
     UINT8  v[4] __PACKED;
     struct __PACKED
     {
        UINT16 LW;
        UINT16 HW;
     } word;
     struct __PACKED
     {
        UINT8 LB;
        UINT8 HB;
        UINT8 UB;
        UINT8 MB;
     } byte;
     struct __PACKED
     {
        UINT16_VAL low;
        UINT16_VAL high;
     }wordUnion;
     struct __PACKED
     {
        __EXTENSION UINT8 b0:1;
…..
        __EXTENSION UINT8 b31:1;
    } bits;
  } UINT32_VAL;
```

Un type union permet d'accéder à un même ensemble de données avec des granularités différentes

Outils de développement MPLABX

- Démo de l’outils
  - CTRL+click gauche !
- Présentation de la structure des fichiers du projet
- Fonctions à implémenter
  - void etudiantSocketAppInit();
  - void etudiantSocketAppTask();

Debogage

- Debogage avec pas à pas
  - Points d’arrêts (breakpoints)
  - Stoppe la lecture (et donc le vidage) de la FIFO de réception
- fonction DBPRINTF(char texte[]);
  - pour afficher dans la console debug des messages courts indiquant l'état du programme. (proche de Serial.print() sur arduino)
  - A utiliser avec parcimonie car très lente.
  - La chaine passée en paramètre doit être terminée par \n pour déclencher l'envoi effectif de la chaine sur l'interface debug. (Vidage FIFO d’émission). Dans le cas contraire, vous pourrez faire un appel à: DBPRINTF("\n");

Accés aux périphériques

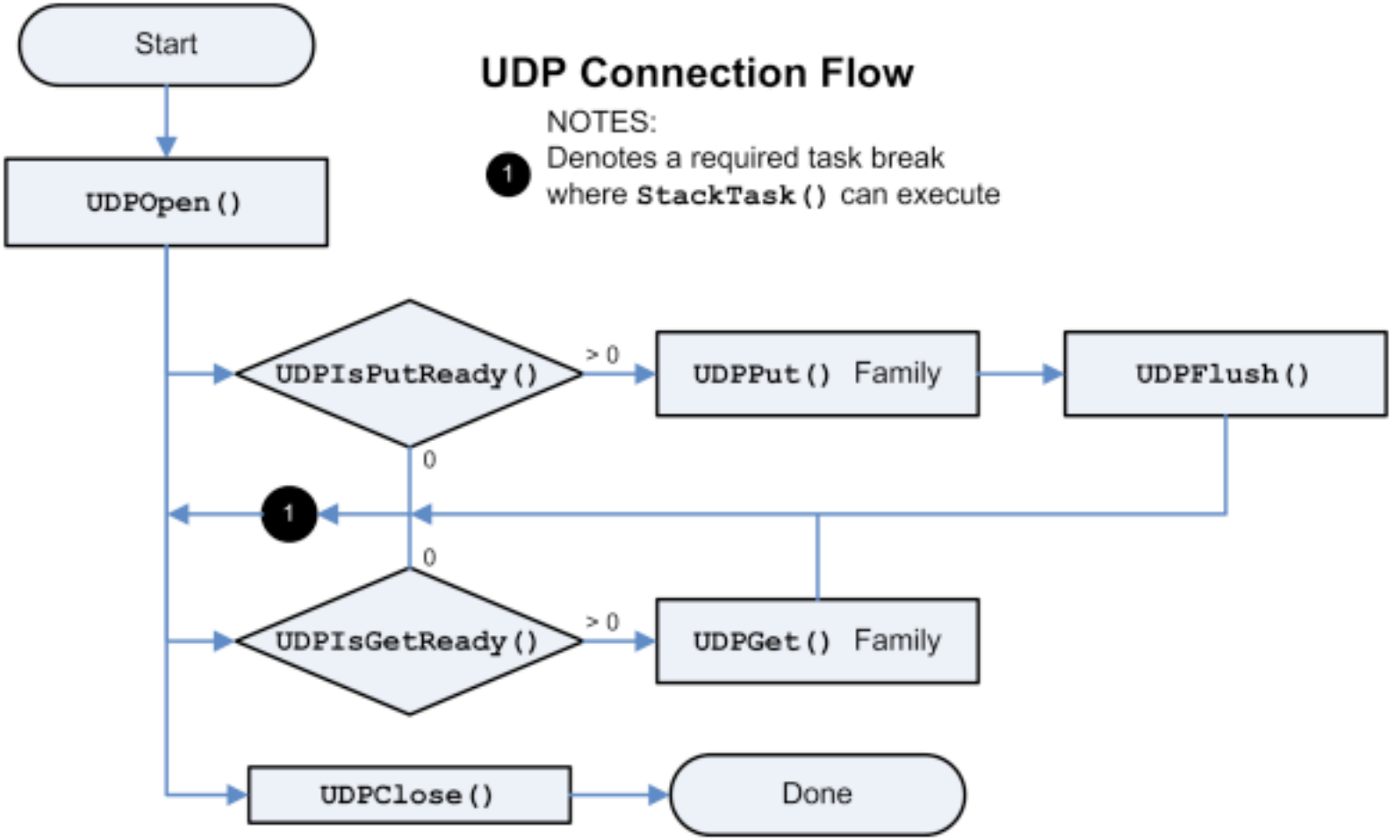
- Lecture de l’état d’un des boutons:

```
b1state=(int)!PORTReadBits(IOPORT_D, BIT_6);
```

- Pilotage d’une LED

```
LED0_IO = 1;
```

Gestion des sockets



Gestion des sockets

- UDP_SOCKET UDPOpen(UDP_PORT localPort, NODE_INFO * ptr_remoteNode, UDP_PORT remotePort);
- WORD UDPIsPutReady(UDP_SOCKET s);
- WORD UDPPutArray(BYTE * cData, WORD wDataLen);
- void UDPFlush();

- WORD UDPIsGetReady(UDP_SOCKET s);
- WORD UDPGetArray(BYTE * cData, WORD wDataLen);

- UDPClose(UDP_SOCKET s); ne sera pas utilisée

Gestion du temps

- Utilisation d’un timer par **scrutation** :

```
#define ETUDIANTSOCKETAPP_OCCURENCE_PAR_SEC 5ul

void etudiantSocketAppTask()
{
if(TickGet() -tim >= TICK_SECOND/
ETUDIANTSOCKETAPP_OCCURENCE_PAR_SEC)
  {
  tim = TickGet();
  compteur++; //Tâche à exécuter, incrémentation d'un compteur par exemple....
  }
}
```

Gestion de version avec **git**

- **git** est un logiciel de gestion de versions
  - Stocke les différentes versions des fichiers
  - Permet de savoir qui a modifié quoi, quand et pourquoi
  - **décentralisé** (pair à pair): chacun possède tout l'historique
- Gestion d'arborescence avec différentes branches (**branch**) ou fourchettes (**fork**) avec possibilité de fusion (**merge**)
  - Permet à un développeur de créer des nouvelles fonctionnalités qui empêcherait les anciennes de fonctionner sans gêner les autres
- Outils graphique pour la visualisation: **gitk**

Git, Quelques commandes (wikipedia)

- **git init** crée un nouveau dépôt ;
- **git clone** clone un dépôt distant ;
- **git add** ajoute de nouveaux objets blobs dans la base des objets pour chaque fichier modifié depuis le dernier commit. Les objets précédents restent inchangés ;
- **git commit** intègre la somme de contrôle SHA-1 d'un objet tree et les sommes de contrôle des objets commits parents pour créer un nouvel objet commit ;
- **git branch** liste les branches ;
- **git merge** fusionne une branche dans une autre ;
- **git log** affiche la liste des commits effectués sur une branche ;
- **git push** publie les nouvelles révisions sur le Remote. (La commande prend différents paramètres) ;
- **git pull** récupère les dernières modifications distantes du projet (depuis le Remote) et les fusionner dans la branche courante ;
- **git stash** stocke de côté un état non commité afin d'effectuer d'autres tâches.