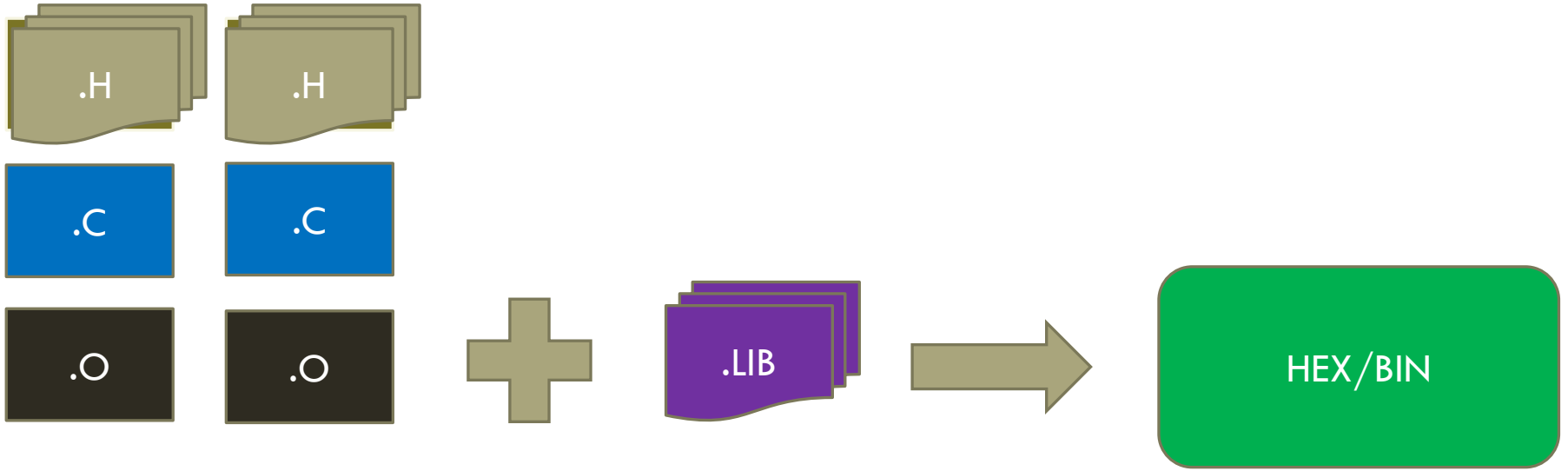
FINESSES DU C

Yves Masur
Microclub, le 8.2.2019

COMPILATION – VARIABLE GLOBALE

Compilation de plusieurs modules C (ou CPP), et link final:



Attention: Arduino triche !!

MACRO D'INCLUSION, CLASSE DE VARIABLE

```
#ifndef BEFA_IO_H
#define BEFA_IO_H
#if defined BEFA_IO_C
    #define CLASS
#else
    #define CLASS extern ← garantie d'un affection au module BEFA_IO.C
#endif
```

USAGE DE CLASSE DE STOCKAGE

Dans un fichier .H, selon le module, la classe de stockage est ‘extern’ ou locale

…

CLASS char infile[MAXPATH];

CLASS char outfile[MAXPATH];

…

MACRO NOMBRE D'ÉLÉMENTS + BORNE

Programmation défensive!!

#define NELEM(x) (sizeof(x)/sizeof(x[0])) ← nb d'éléments du tableau

```
#define BOUND(var,min,max)    \
    {                         \
    if (var < min) var = min;  \
    if (var > max) var = max;  \
    }
```

USAGE DE BOUND ET NELEM

```
BOUND(p, 0, 31); // be sure staying in the range
    c = *(str_psp[n] + p); // read the GF sign
    for (i=0; i<NELEM(GF_online) && GF_online[i][0]; i++)
    {
        if (c == GF_online[i][0]) //Q:char match with the table?
        {
            rc = GF_online[i][1]; //A:yes, get the TS bits
            break;
        }
    }
```

UTILISATION D'UN AUTRE COMPILATEUR

Pour remplacer les parts manquantes (fonctions, variables) du code final

```
#include <string.h>
#if defined(__GNUC__) ← le compilateur du PC a une définition __GNUC__
    #include "_gnu_.h"   ← définitions bidon, pour tester
#else
    #include "projdefs.h"  ← p. exemple des ports I/O
    #include "debug.h"   ← sur PC, on a le debugger!!
#endif
```

MACRO POUR MULTITÂCHE

```
#ifdef WIN32
    #ifndef _MT
    #define _MT /* multi threading specialities */
    #endif
    CRITICAL_SECTION fl_CriticalSection;
    #define disable() EnterCriticalSection(&fl_CriticalSection)
#else   /* !WIN32 mode */
```

USAGE MULTITÂCHE

Modifier une variable par une tâche sans être perturbé par l'interruption:

Les fonctions enable() et disable() sont propres:

- au compilateur
- au CPU

Il faut les redéfinir par macro.

```
CLASS TIMER Timer;
#define SET_TIMER(x,val) { disable(); x=val; enable(); }
```

OPÉRATION ATOMIQUE

Pour partage d'information entre processus indépendants, inattendus avec interruptions hardware:

class Counter{ public: void clear_all(); unsigned int get_live(); //{return c_live;}

… }

DANGER !! L'entier 16 bits est constitué de 2 octets:

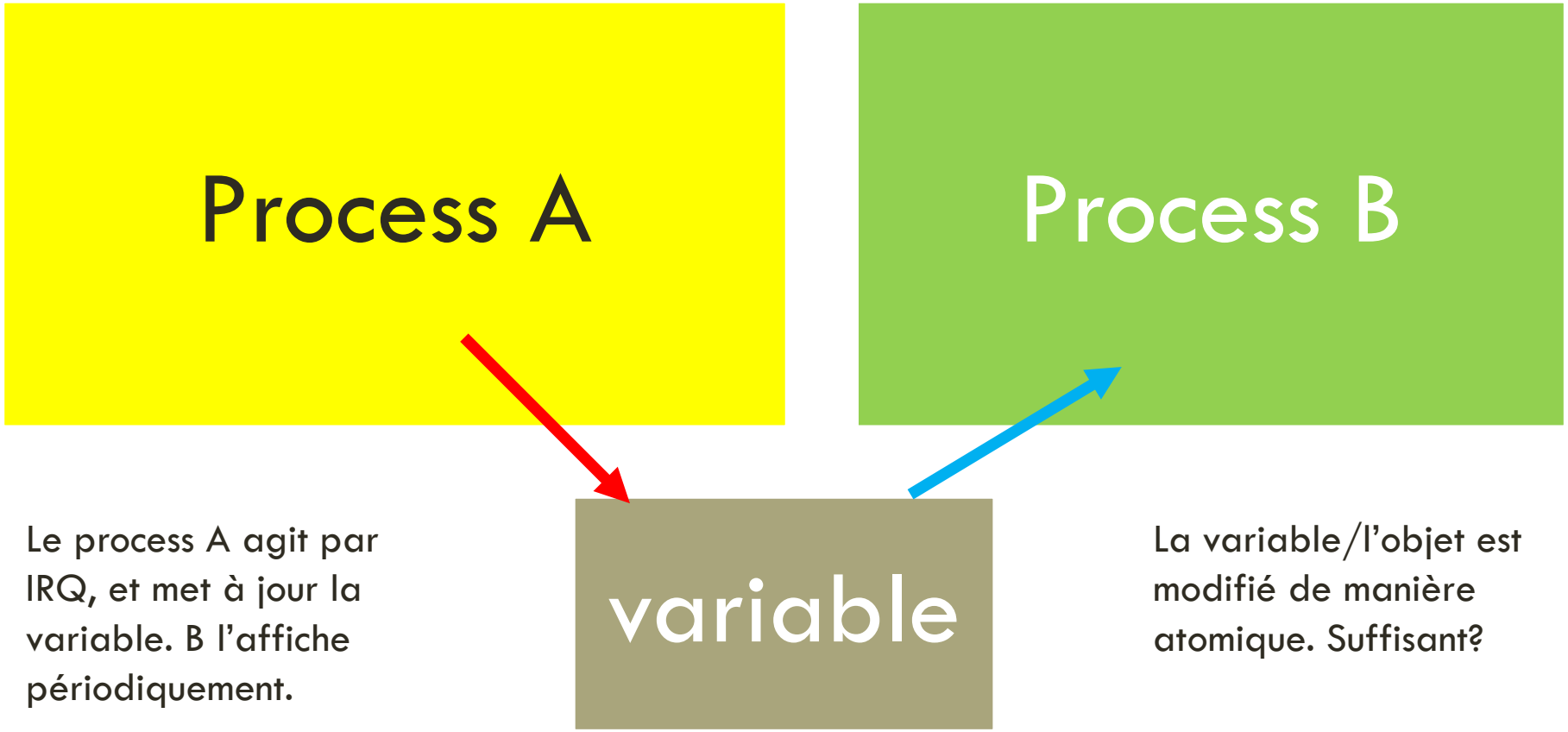
La fct doit bloquer toute modification tierce avant de lire la variable

EXEMPLE DE LECTURE

```
#include <Arduino.h>
#include "Geiger_counter.h"

unsigned int Counter::get_live()
{
  register int c;        ← remarquez le modificateur register
  noInterrupts();
  c = c_live;
  interrupts();
  return c;
}
```

VARIABLE VOLATILE



Le process A agit par IRQ, et met à jour la variable. B l'affiche périodiquement.

La variable/l'objet est modifié de manière atomique. Suffisant?

BLOCAGE POSSIBLE

Code

volatile int c_live;

Process A:

```
void add_live() { c_live++; }
```

Process B:

```
while (c_live == 0);      //attente événenement

{ … } // suite conditionnelle
```

Effet

Un compilateur optimise les lectures en RAM; et met des variables dans des registres du CPU.

La variable est à relire à chaque fois avant une opération.

Ceci garantit une exécution correcte.

VARIABLE : MASQUAGE ET STATIQUE

masquage

```
fct()
{
        int a = 1; // sur le stack
        {
                int a = 10;  // masque la version 1 de a
                // traitement
        }
        return a+1;
}
```

Que rend la focntion?

- au 1er appel?

- au second?

Et si : static int a = 1;

RÉSUMÉ

Macro: pour fiabiliser le code de manière défensive

Garantir la taille d'un tableau et son parcours

Gérer l'affectation des variables à un module

Travailler/tester des modules sur compilateurs différents

Multitâche: protéger les écritures et lectures de manière atomique

Bon usage des modificateurs register et volatile