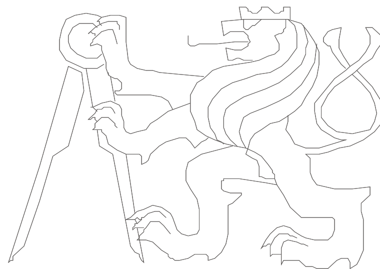
Architektury počítačů

IEEE754



České vysoké učení technické, Fakulta elektrotechnická

Ztráta přesnosti

25. února 1991 při válce v Zálivu nedokázala americká raketová baterie sledovat a zachytit iráckou raketu typu Scud. Scud udeřil do kasáren americké armády, zabil 28 vojáků a zranil asi 100 dalších lidí. Příčinou byl nepřesný výpočet času kvůli aritmetickým chybám v počítači.

Konkrétně, **čas v desetinách sekundy se čítal vnitřními integer hodinami systému a násobil se float 1/10**, aby se obdržel čas v sekundách. Nepřesná hodnota float 1/10 vynásobená velkým číslem času v desetinách sekundy vedla k velké chybě.

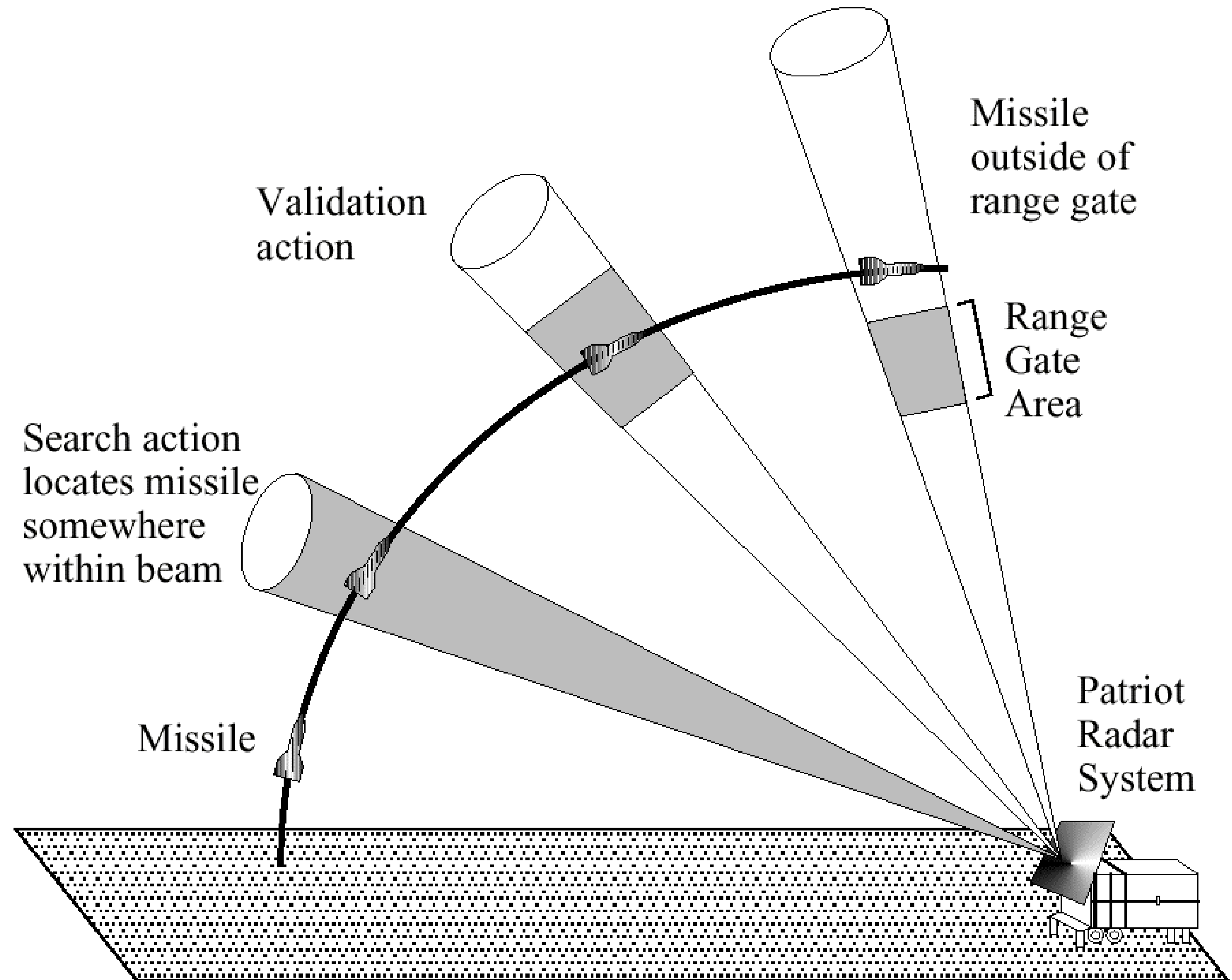
Po 100 hodinách provozu činila časová chyba **0,34 sekundy**. Raketa Scud cestuje rychlostí asi 1 676 metrů za sekundu, a tak za tuto dobu urazila více než půl kilometru, což už leželo mimo „dosahovou bránu“, kterou systém Patriot sledoval.

Více: https://sdqweb.ipd.kit.edu/publications/pdfs/saglam2016a.pdf

Ztráta přesnosti v realite

Podle Generálního úřadu vlády USA byla ztráta přesnosti odpovědná za selhání protiraketové baterie Patriot.

Došlo k ní vynásobením integer čítače času float konstantou 0.1.



Zdroj: University of Illinois Urbana-Champaign

Kvíz: Rozhodněte o platnosti vztahů

```
int x = …;
float f = …;
double d = …;
```

Předpokládejme, že `d` a `f` nejsou NAN

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (float) d`
- `f == -(-f)`
- `2/3 == 2/3.0`
- `d < 0.0  ⇒ ((d*2) < 0.0)`
- `d > f  ⇒ -f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

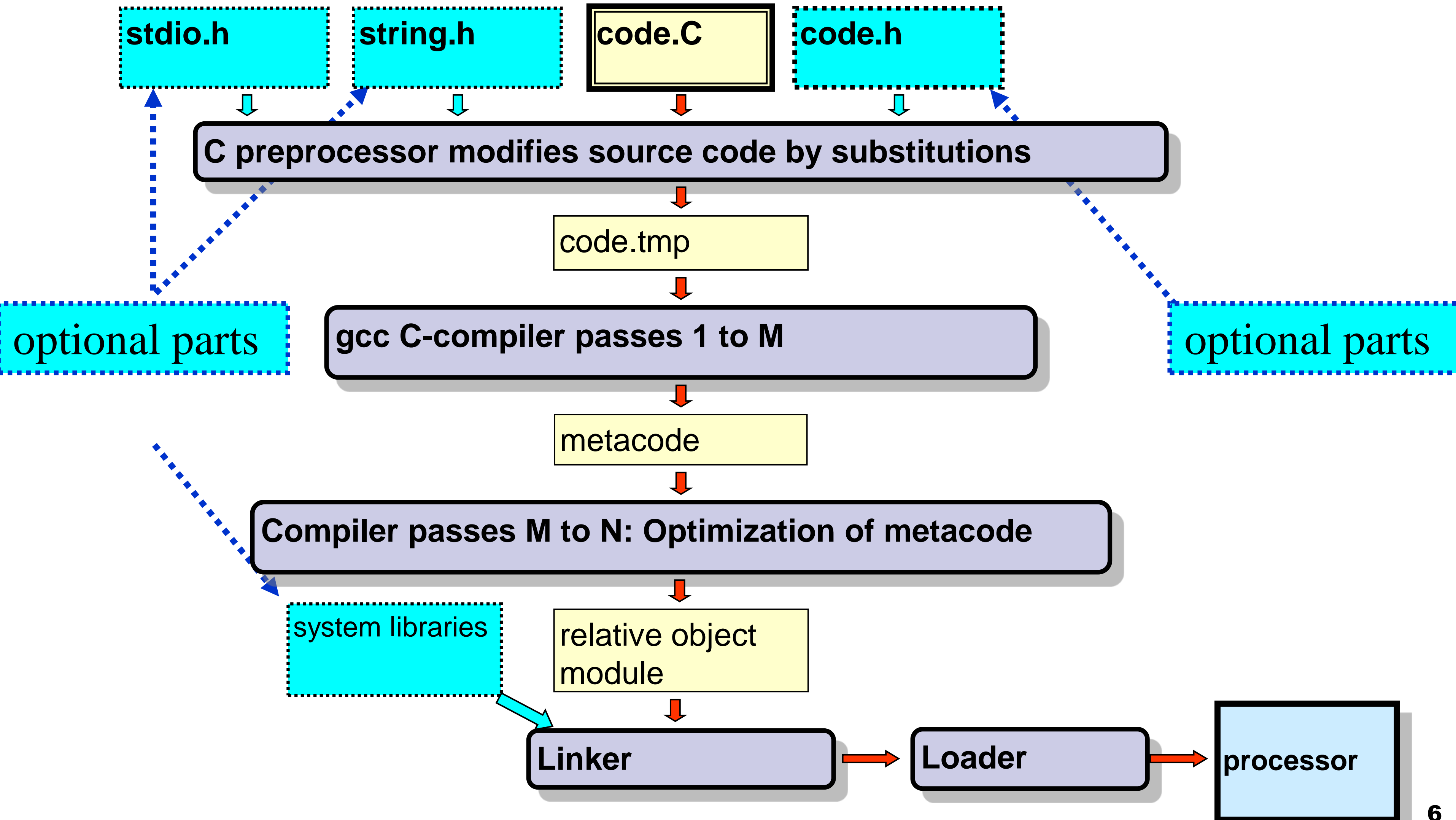
Odpovědi na kvíz

```
int x = …;
float f = …;
double d = …;
```

Předpokládejme, že `d` a `f` nejsou NAN

- `x == (int)(float) x` — Ne: 24 významých bitů
- `x == (int)(double) x` — Ano: 53 významých bitů
- `f == (float)(double) f` — Ano : zvýšení přesnosti
- `d == (float) d` — Ne: ztráta přesnosti
- `f == -(-f);` — Ano : pouhá změna znaménka
- `2/3 == 2/3.0` — Ne: 2/3 == 0
- `d < 0.0 ⇒ ((d*2) < 0.0)` — Ano!
- `d > f ⇒ -f > -d` — Ano!
- `d * d >= 0.0` — Ano!
- `(d+f)-d == f` — Ne: Není asociativní

Basic Steps of C Compiler

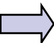


C primitive types

Size	C	C alternative	Range
1	any integer, true if !=0	BOOL[1]	0 to !=0
8	char[2]	signed char	–128 to +127
8	unsigned char	BYTE[1]	0 to 255
16	int	signed short	–32768 to +32767
16	unsigned short		0 to + 65535
32	int	signed int	-2^31 to 2^31-1
32	unsigned int	DWORD[1]	0 to 2^32-1
64	long	long int	-2^63 to 2^63-1
64	unsigned long	LWORD[1]	0 to 2^64-1

1) BOOL, DWORD, LWORD nejsou standardní typy jazyka C, ale pouhá rozšíření u některých implementacích, jinde se zavádějí například přes "#define" makro

2) Výchozí typ je **signed**.



Definition of BYTE and BOOL

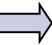
// textové substituční pravidlo bez koncového ;

- #define BYTE unsigned char
- #define BOOL int

// nový typ vyžaduje koncový ; - jde o příkaz jazyka C

- typedef unsigned char BYTE;
- typedef int BOOL;

Jazyk C nemá striktní rozlišení #define ~ typedef,
ale typedef se lépe integruje do překladače.



Parametrizované makro

```
#define PRINT_MEM(a) print_mem((unsigned char*)&(a), sizeof(a))
```

- Makra se mohou definovat s parametry, které nemají žádné typy – pouhá substituce textu za text.
- Syntaxe:

```
#define MACRONAME(parameter_list) text
```

  - Nesmí být mezera před (

Příklady:

```
#define MAXVAL(A,B) ((A) > (B)) ? (A) : (B)

#define PRINT(e1,e2) printf("%c\t%d\n",(e1),(e2));

#define putchar(x) putc(x, stdout)
```

#define PRINT_MEM(a) print_mem((unsigned char*)&(a), sizeof(a))

Vedlejší efekty!!!

Špatná definice:

```
#define PROD1(A,B) A * B
```

výsledek:

```
PROD1(1+3,2) → 1+3 * 2
```

Oprava chyby ()

```
#define PROD2(A,B) (A) * (B)

PROD2(1+3,2) → (1+3) * (2)
```

Opakování C: Sign Extension (znaménkové rozšíření)

Prerekvizita APOLOS – kapitola 3.2.5

Příklad:

```
short int x =  15213;
int      ix = (int) x;
short int y = -15213;
int      iy = (int) y;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 C4 92	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

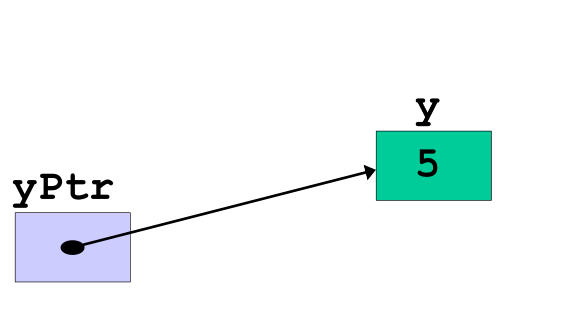
& (address operator)

Vrací adresu začátku proměnné v adresovém prostoru.

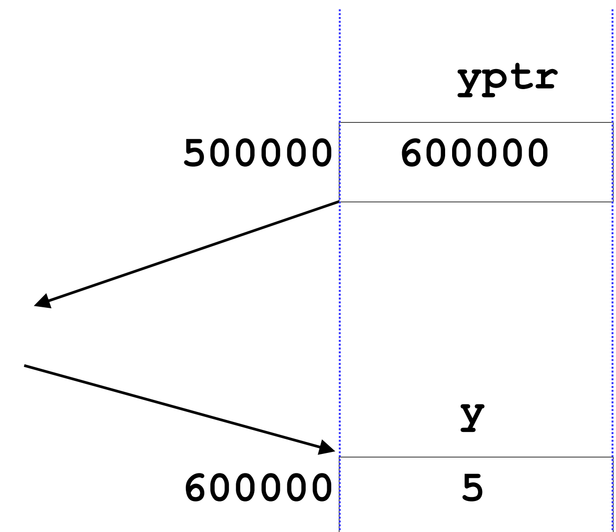
Příklad

```
int y = 5;
int *yPtr;
yPtr = &y;     // yPtr obsahuje adresu y
```

yPtr "ukazuje na" **y**



adresa y se stala hodnotou yptr



Opakování C: Operace s ukazateli

& (address operator)

vrací adresu operandu

* dereference address

hodnota uložená na adrese

***** and **&** jsou inverzní

(*ale ne vždy aplikovatelné*)

```
*&myVar == myVar
```

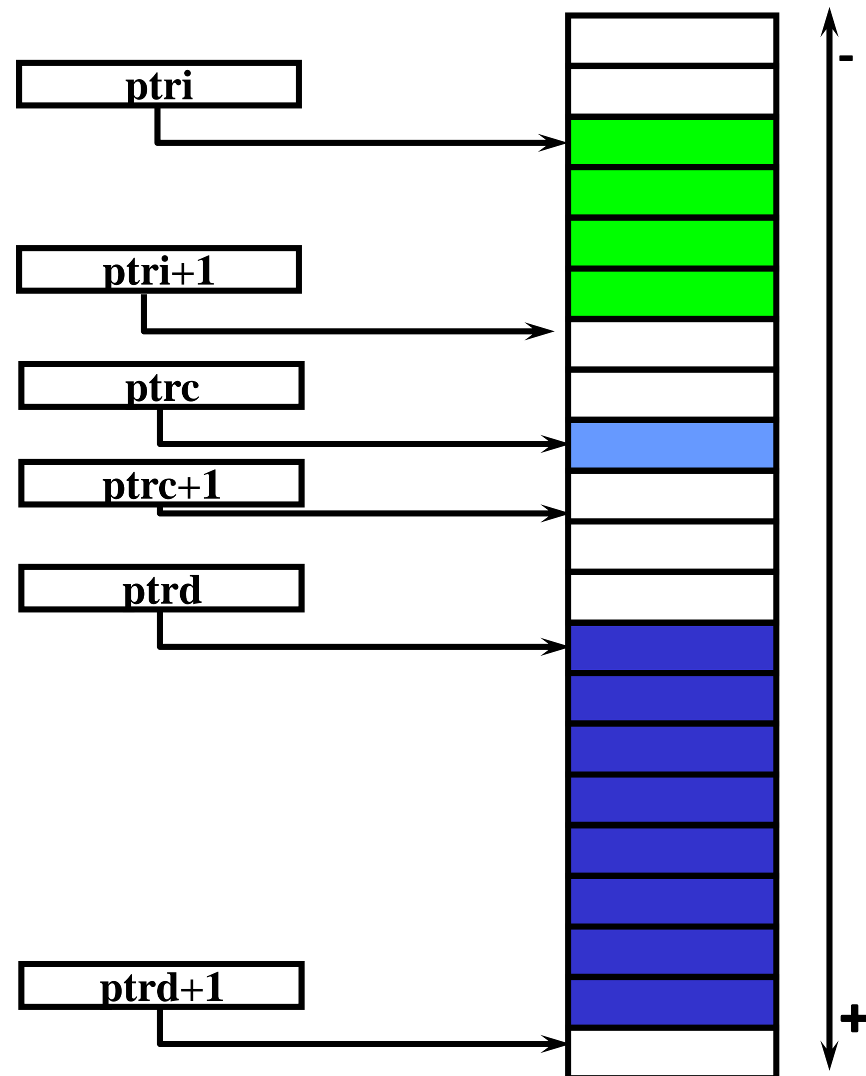
and

```
&*yPtr == yPtr
```

Opakování C: Velikost ukazatele v C-kódu

```
int * ptri;
char * ptrc;
double * ptrd;
```

*ptrx ≡ ptrx[0]
*(ptrx+1) ≡ ptrx[1]
*(ptrx+n) ≡ ptrx[n]
*(ptrx-n) ≡ ptrx[-n]




```
nr1 = sizeof (double);
nr2 = sizeof (double*);
```

nr1 != nr2

Opakování C: Ukazatel s **const** v C-kódu

int x, y;

int * lpio = &y;

*lpio = 1;   x=*lpio;   lpio++;

const int * lpCio = &y;

~~*lpCio = 1;~~   x=*lpCio;   lpCio++;

int * **const** lpioC = &y;

*lpioC = 1;   x=*lpioC;   ~~lpioC++;~~

const int * **const** lpCioC = &y;

~~*lpCioC = 1;~~   x=*lpCioC;   ~~lpCioC++;~~