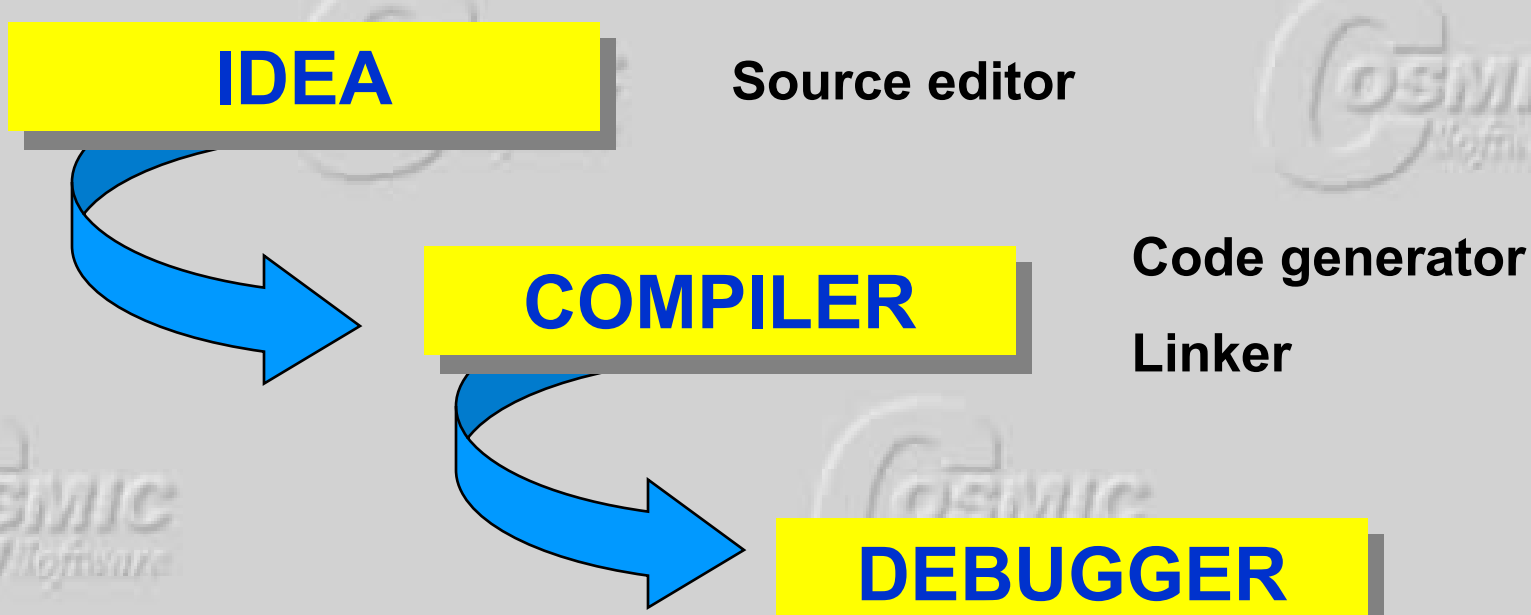
Supporting Embedded Innovation
Since 1983

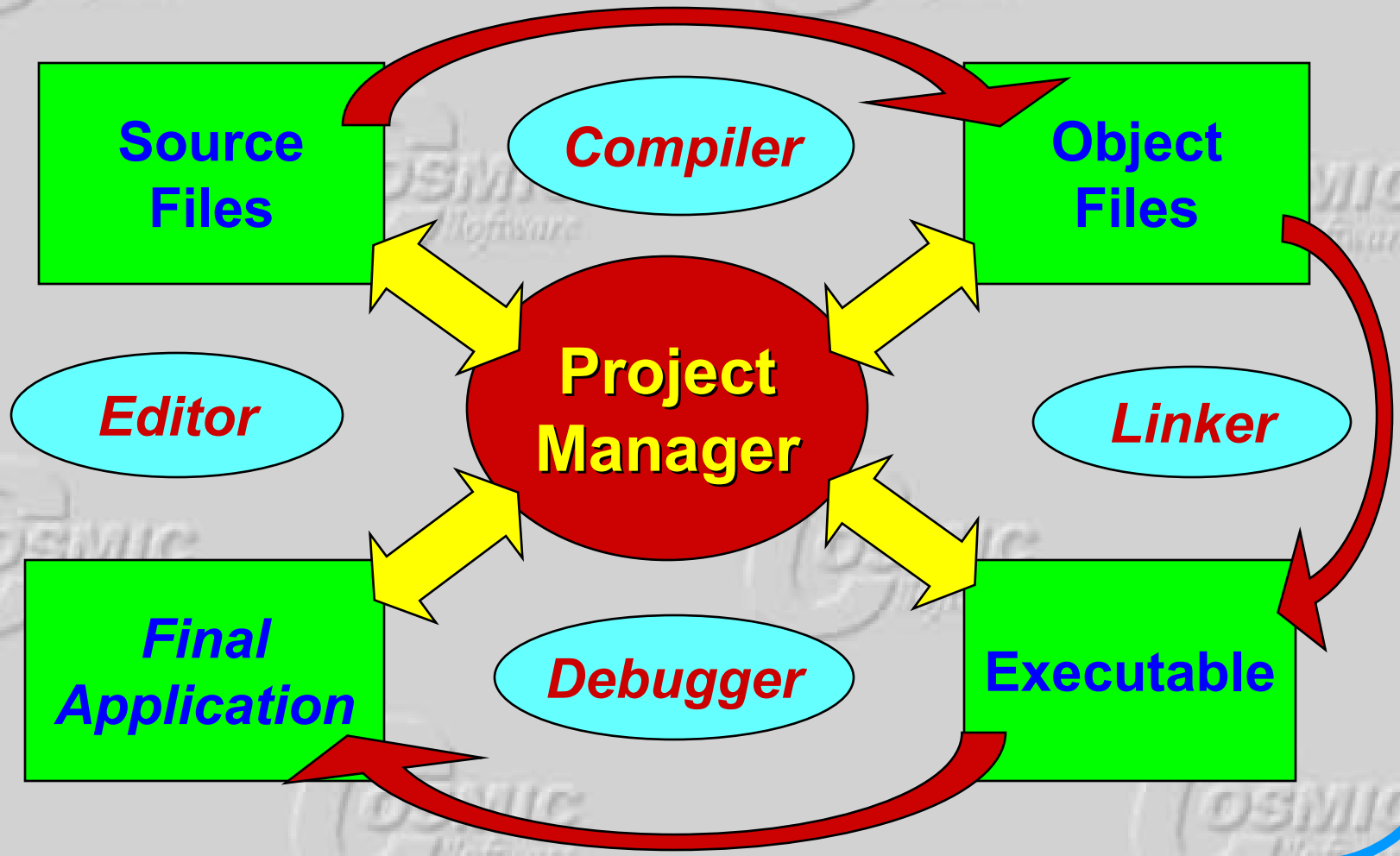
ST7 C Compiler



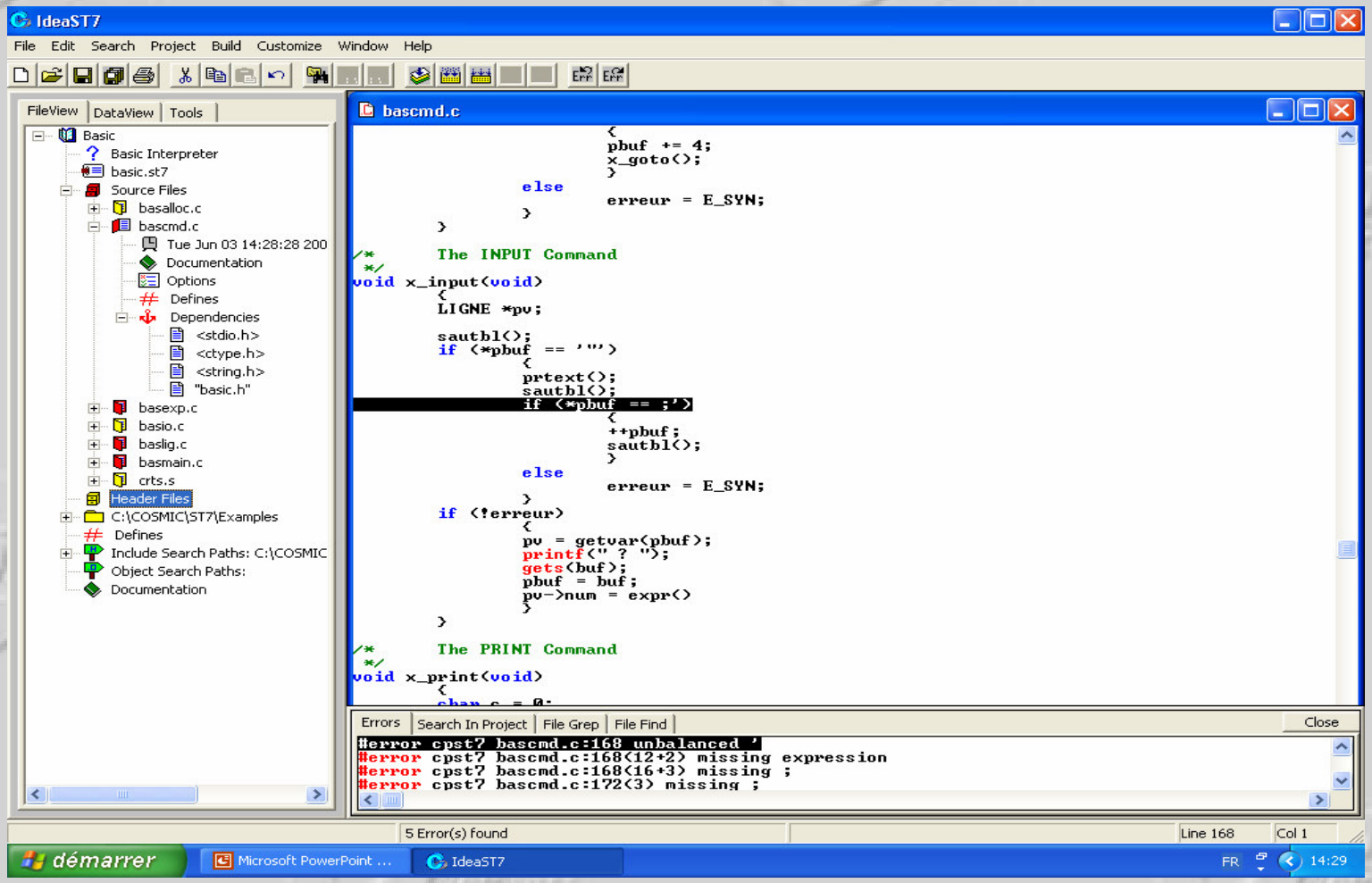
IDEA

- Windows Integrated environment
- Integrates with C and Assembler
- Project management
- Automatic make and build
- Integrated Editor
- Automatic Error Handling
- Integration of documentation

IDEA Flowchart



IDEA Main Window



ST7 C Compiler Features

- ***ANSI / ISO Compliant***
  - Full language implementation
  - Standard C libraries (subset for embedded)
  - Compatible with native compilers
- ***Memory Spaces***
  - Short range (zero page) / Long range
  - Direct access to I/O Registers
  - EEPROM support

ST7 C Compiler Features

- ***Full Floating Point Support***
  - Single Precision IEEE-754
- ***C and Assembly Library Source Code***
  - Optional Integer only library support
- ***Bit Optimizations***
  - Extensive use of bit instructions
  - Bit variables
  - Option to reverse bit ordering
  - 8-Bit Bitfield support

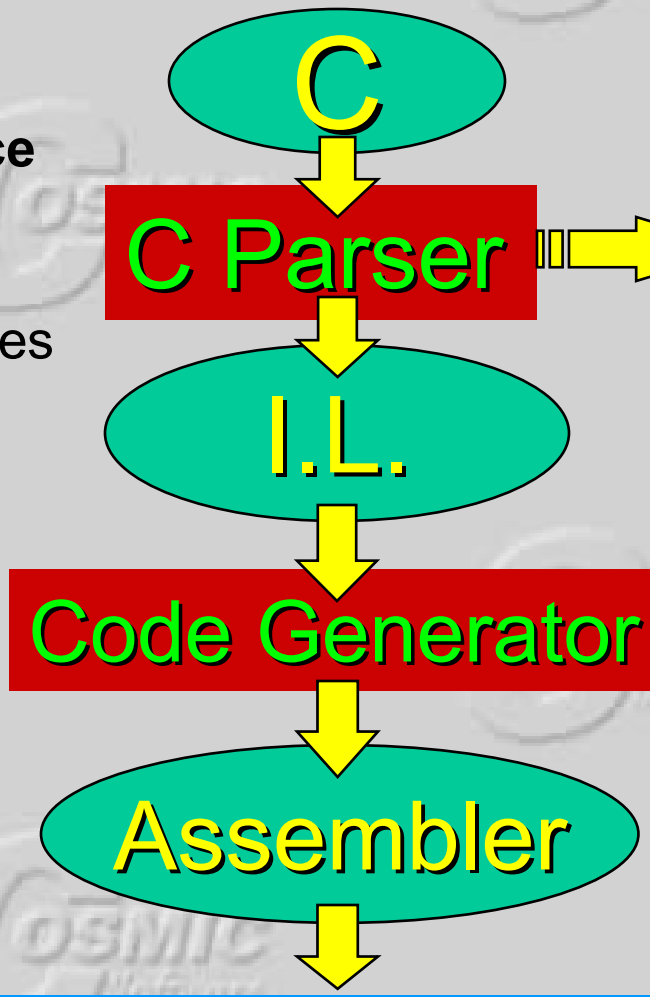
ST7 C Compiler Features

- ***Stack Implementation***
  - Physical Stack (**@stack**)
    - fully recursive and reentrant
  - Overlay Static Memory (**@nostack**)
    - Optimized Memory Allocation by Linker
  - Both allowed in the same application
- ***And Also***
  - Overflow control for signed compares
  - Absolute C and Assembler Listings

Compiler Architecture

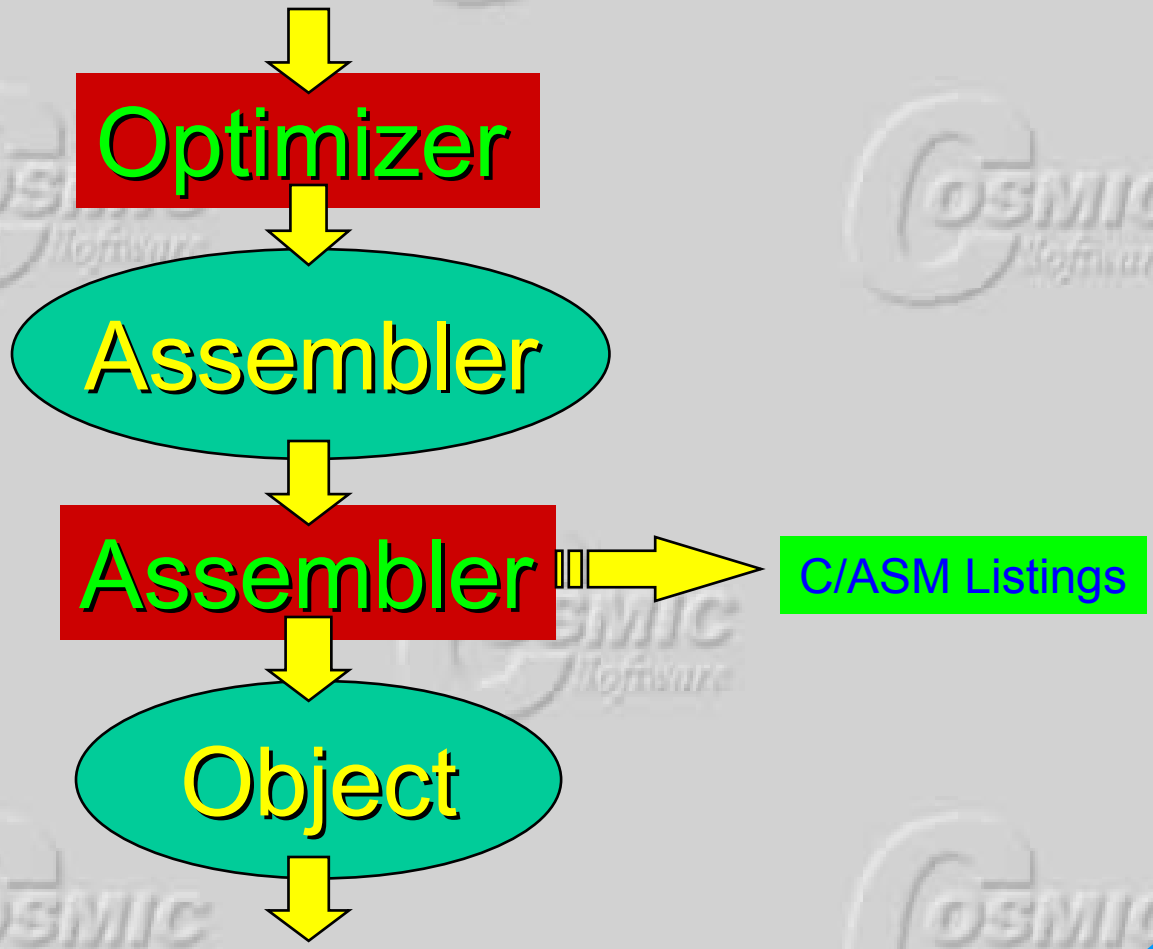
Switches to enforce
- Prototyping
- Strict checking for code inconsistencies
- ...

Switches to tune
- first Levels of Optimization
- listings
- ...

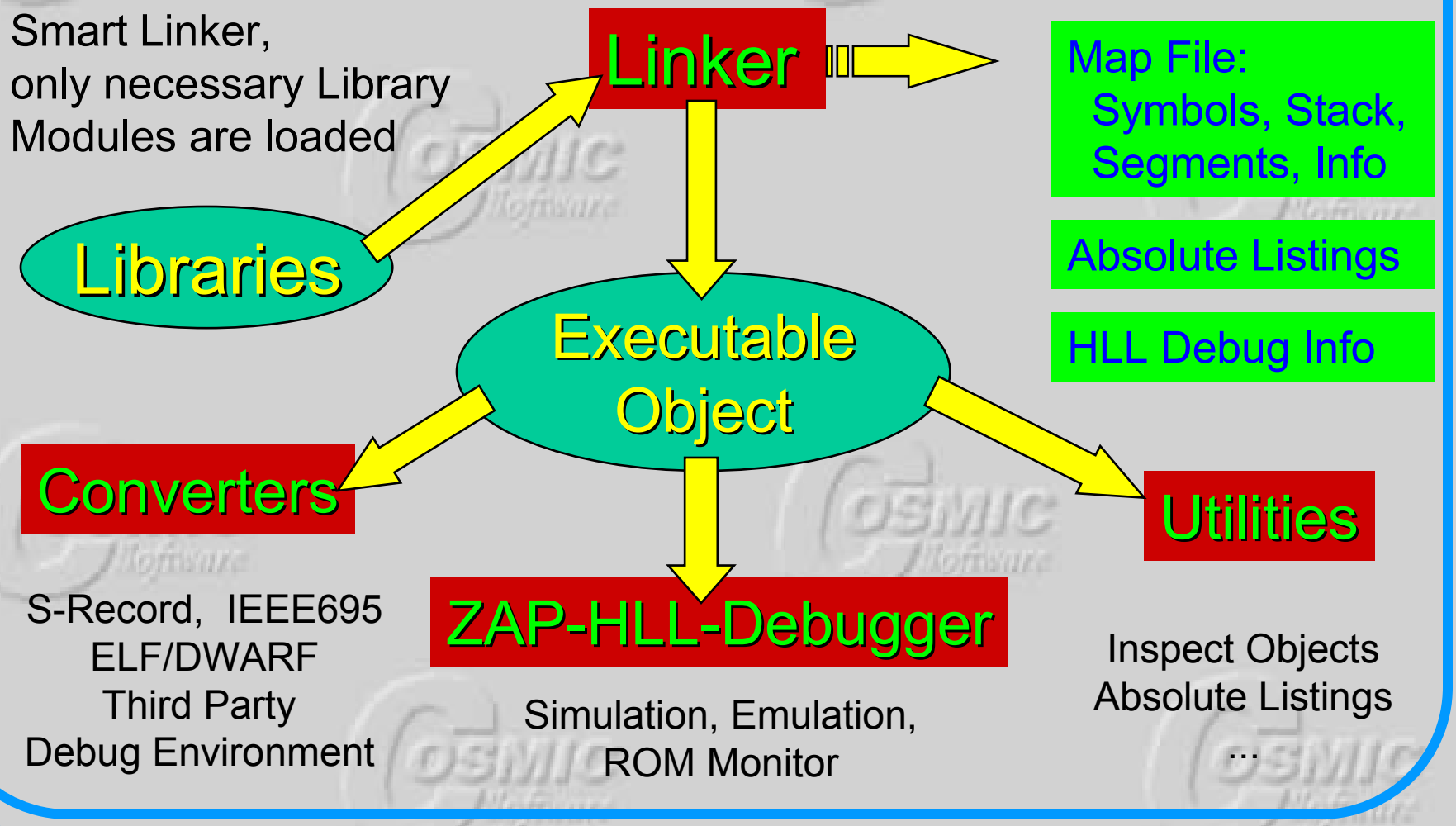


Compiler Architecture

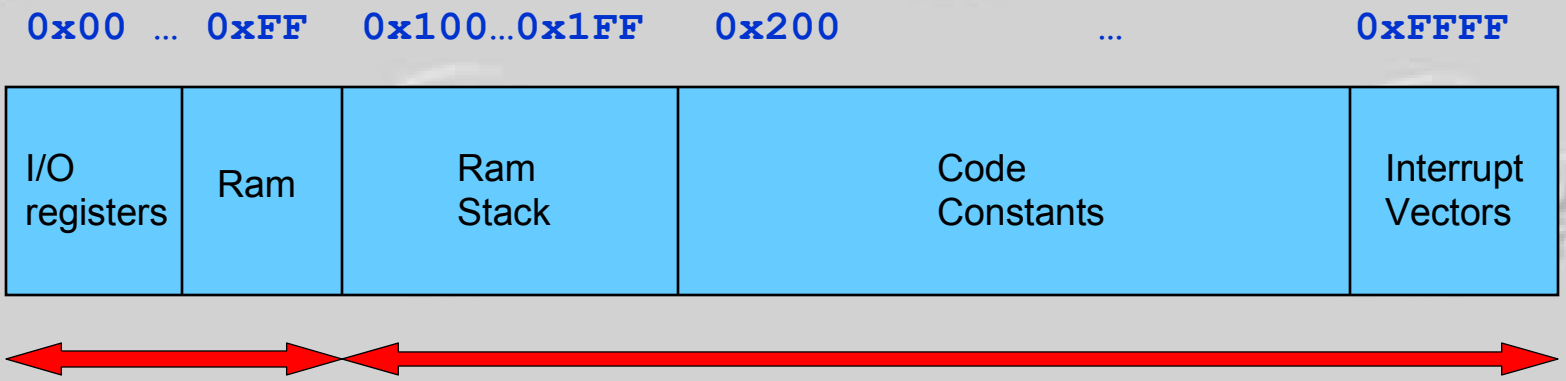
Full Control on any optimization Feature



Compiler Architecture



Memory Spaces



Static byte access:

```
ld a,adr
```

Stack byte access:

```
ld x,s
ld a,(0x100,x)
```

Memory Models

- ***Physical Stack***
  - Stack long +modsl
  - Stack short +mods
- ***Static Memory***
  - Memory large +modml
  - Memory medium +modmm
  - Memory small +modms
  - Memory short +modm
  - Memory compact +modc

Memory Models

Model	Stack	Globals	Pointers
◆ modsl	phys	long	16 bits
◆ mods	phys	short	16 bits
◆ modml	long	long	16 bits
◆ modmm	long	short	16 bits
◆ modms	short	long	16 bits
◆ modm	short	short	16 bits
◆ modc	short	short	8 bits

Mixing Models

- One default model for the whole application
- One matching library type for the whole application
- Possible mixing in the same application of
  - **mods** and **modsl**
  - **modm** and **modms**
  - **modmm** and **modml**
- Use modifiers to adapt behaviour:
  - **@stack** forces arguments and locals on the *physical* stack
  - **@nostack** forces arguments and locals in *simulated* stack
- Interrupt functions defaulted to *stack* model for nested interrupts

C Language Extensions

- ***Absolute addressing***

```
unsigned char PORTB @0x03;
```

- ***Short range addressing***

```
@tiny char shvar;
```

- ***Long range addressing***

```
@near int lgvar;
```

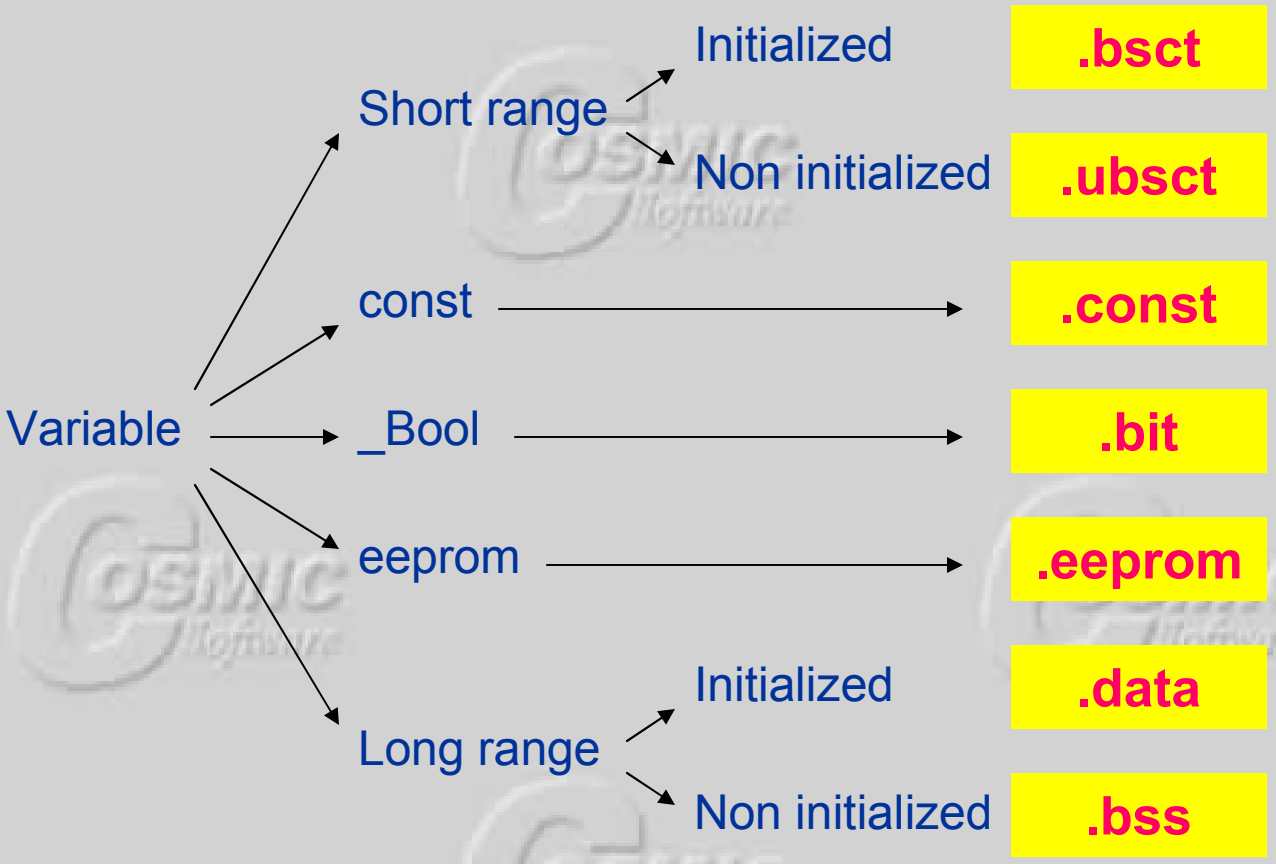
- ***Internal EEPROM***

```
@eeprom short eevar;
```

- ***Bit variables***

```
_Bool bitvar;
```

Data Allocation




```
@tiny char shvar;

const int cvar;

_Bool bvar;

@eeprom int eevar;

@near char lgvar;
```

Code Allocation

- **+split** one function per section
- **+nocst** literals and const in code section
- **+nobss** all data considered as initialized

Section Renaming

- ***Code section***

```
#pragma section (sname)
```

.text ⟶ .sname

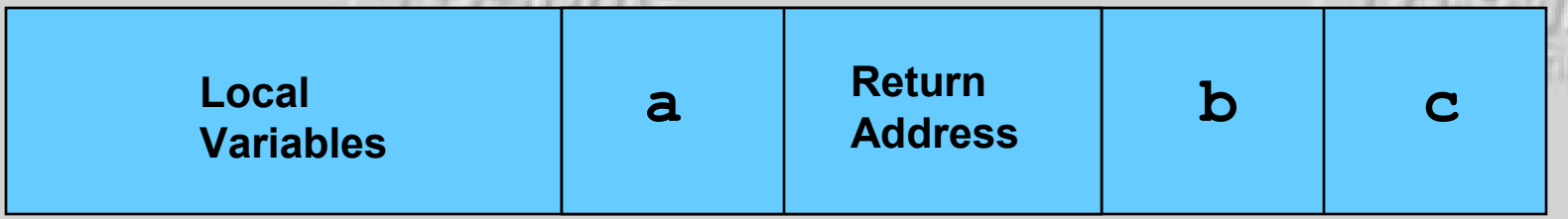
- ***Data section***

```
#pragma section @tiny [name]
#pragma section @near {sname}
```

const, @eeprom, _Bool

Function call

```
x = f(a, b, c);
```



return in **A** (char) or **X:A** (int, pointer) or **c_lreg** (long, float)

Simulated stack:

```
ld a,_f$L-3
```

Physical stack:

```
ld x,s
ld a,(0x103,x)
```

Inline assembly

```
#asm
      rim
      ld a,_var
#endasm
```

```
#pragma asm
      sim
      jp _main
#pragma endasm
```

- inside or outside a function
- connection with global C objects
- no connection with local C objects

Inline assembly

```
#asm
 $N:
        dec     a
        jrne    $L
#endasm
```

- $N creates a new label
- $L uses the current label

Inline assembly

```
result = _asm(« asm code », input);
```

Result copied from returned value in **A** (*char*) or **X:A** (*int*)

Assembler code

Input expression evaluated in **A** (*char*) or **X:A** (*int*)

```
crc = _asm(« add a,#$80\n rlc a », crc);
```

```
LD      A,_crc
ADD     A,#$80
RLC     A
LD      _crc,A
```

Inline code

- ***library functions***

```
@inline char *memcpy(char*,char*,int);
@inline char *memset(char*,char,int);
@inline char *strlen(char*);
```

- ***user functions***

```
@inline void func(int arg);
```

  - cannot return any value
  - any kind and number of arguments
  - replace function call by its body

Bit Variables

```
_Bool Bitvar;
```

- Conform to ANSI standard C99 (C9X)
- Global and local bit variables packed into bytes
- Argument and function return values 1 bit in 1 byte
- Localized bit definition

```
_Bool PA3 @PORTA:3;
```

Interrupt Functions

```
@interrupt void it_func(void);
@interrupt @nostack void it_func(void);
```

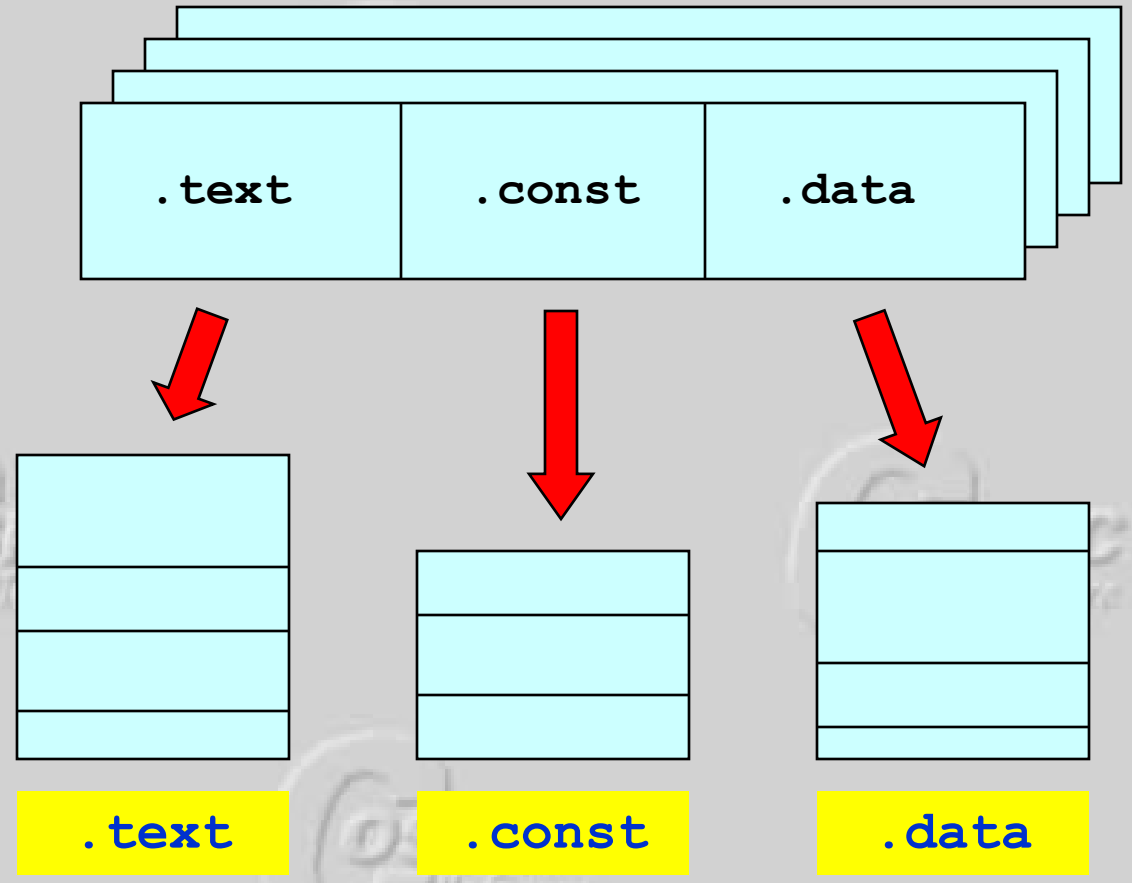
- **c_x** (2 bytes) used for extending **X** register to 16 bits
- **c_y** (2 bytes) used for extending **Y** register to 16 bits
- **c_lreg** (4 bytes) used for long and float operations

Automatically and *selectively* saved with the **Y** register by interrupt functions

If there is a function call inside the interrupt routine:

- **Y**, **c_x**, **c_y** saved even if not explicitly used, unless **@nosvf** is specified
- **c_lreg** *NOT* saved if not explicitly used, unless **@svlreg** is specified

Linker

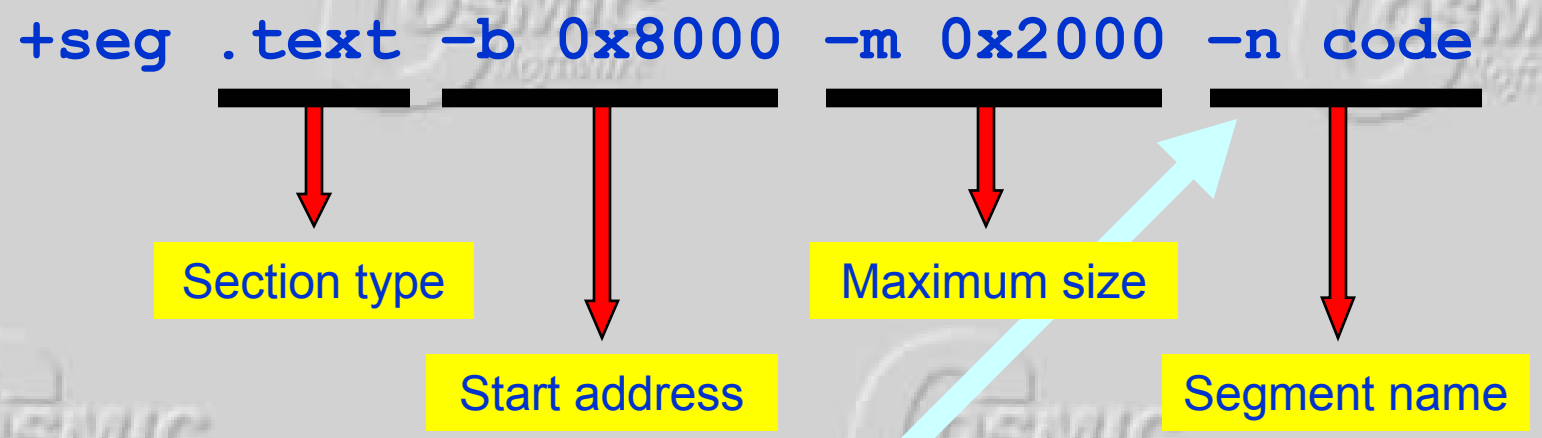


Sections in object files

Segments in executable file

Linker Controls

Segment definition:

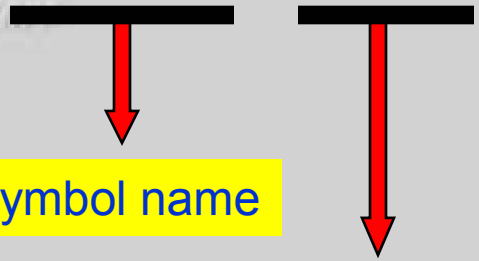



```
+seg .const -a code -n const
```

Linker Controls

Symbol definition:

```
+def symbol=value
```



`+def _ssize=0x80`	Absolute value
`+def _hstart=_ssize`	Other symbol
`+def _bstart=@.bss`	Section reference
`+def _cstart=start(code)`	Segment reference

```
start()
end()
size()
```

Linker Controls

Linker command file:

```
+seg .text -b 0xe000 -m 0x1ff0 -n code
+seg .const -a code -n const
+seg .bsct -b 0x80 -m 0x80 -n ram
crts.o                          Startup code
appli.o                         Application code
libm.st7                        library
+seg .const -b 0xfff0 -n vectors
vectors.o                       Interrupt vectors
```

Supporting Embedded Innovation
Since 1983