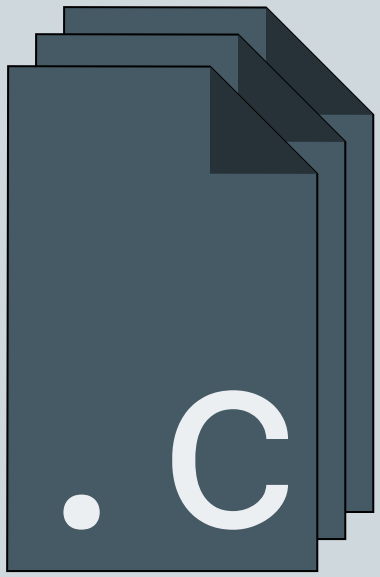
Semantics of Linker Scripts
(a static analysis perspective)

KAREEM KHAZEM

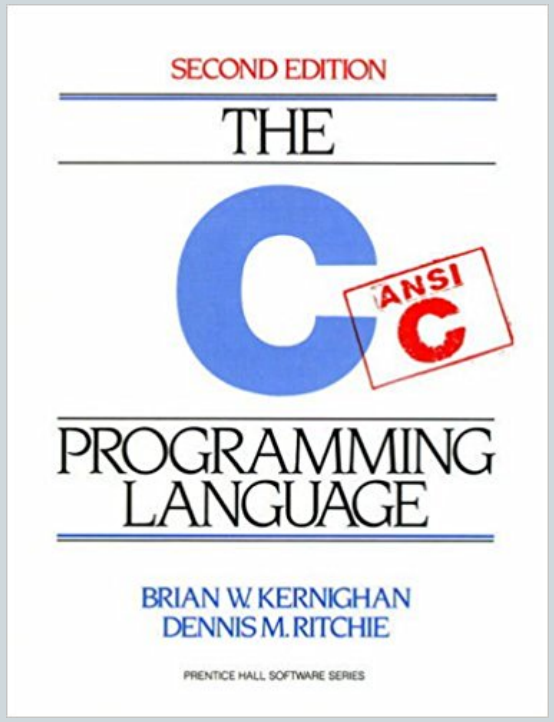
Program Behaviour

Program Behaviour



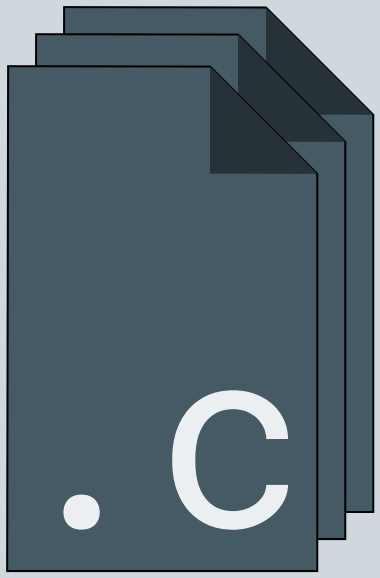
source code

semantics



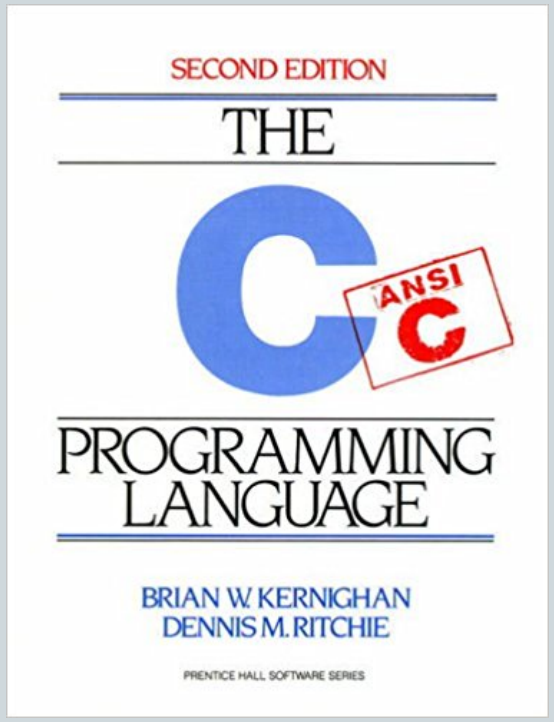
program inputs

Program Behaviour



source code

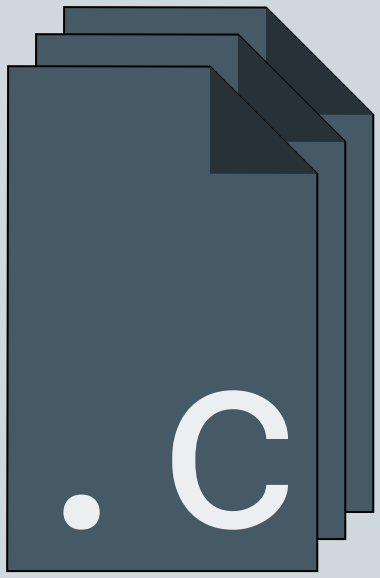
semantics



program inputs

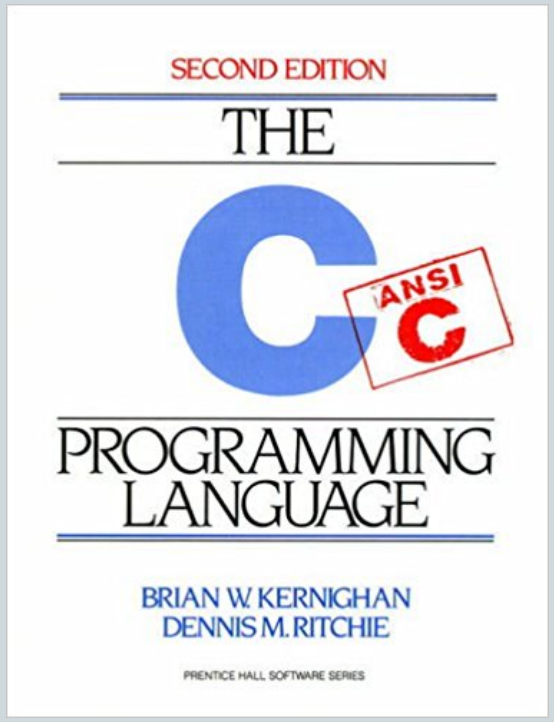
Next Talk: other things

Program Behaviour



source code

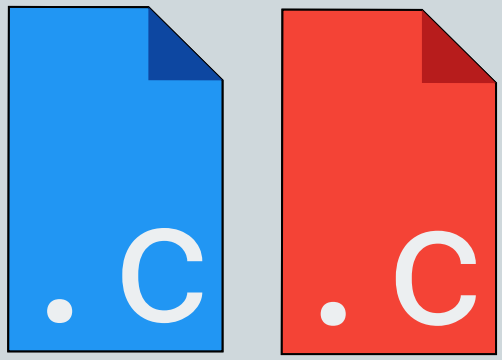
semantics

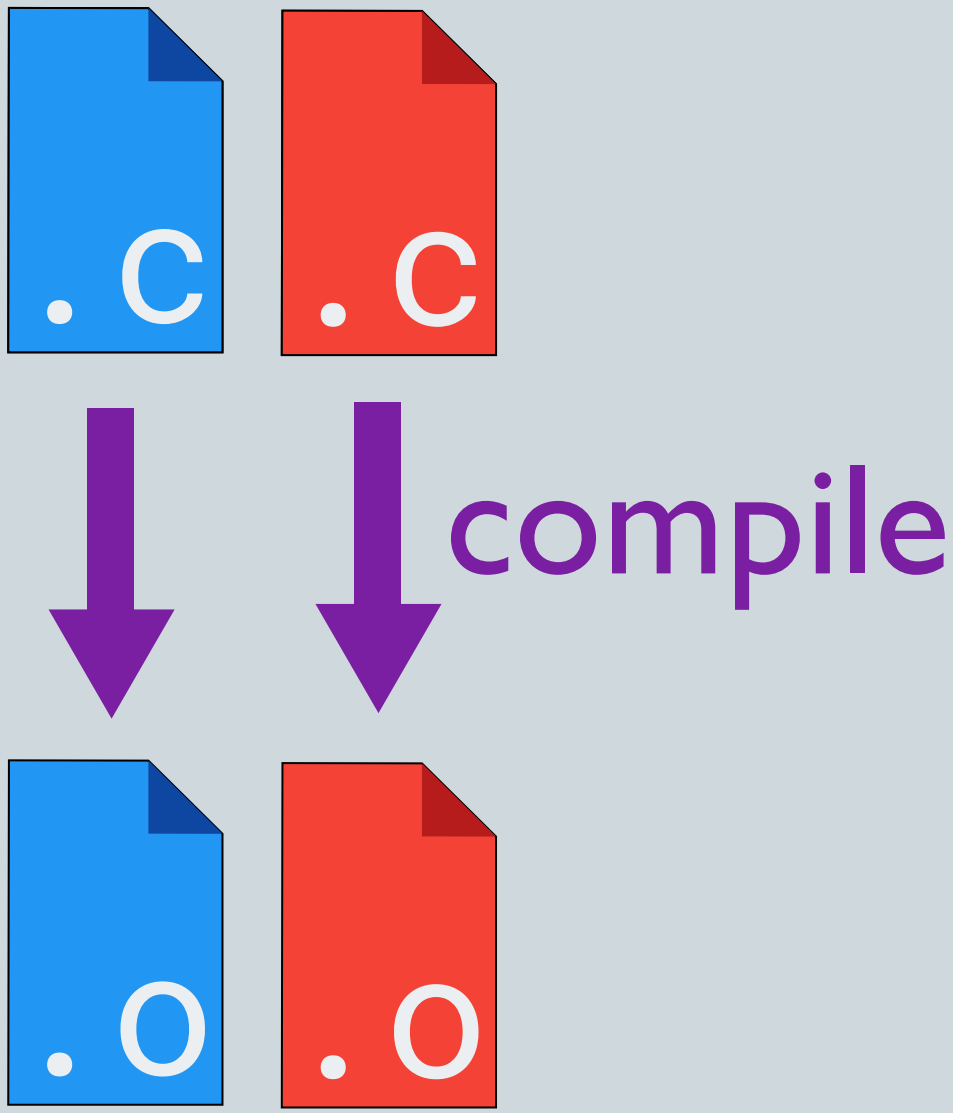


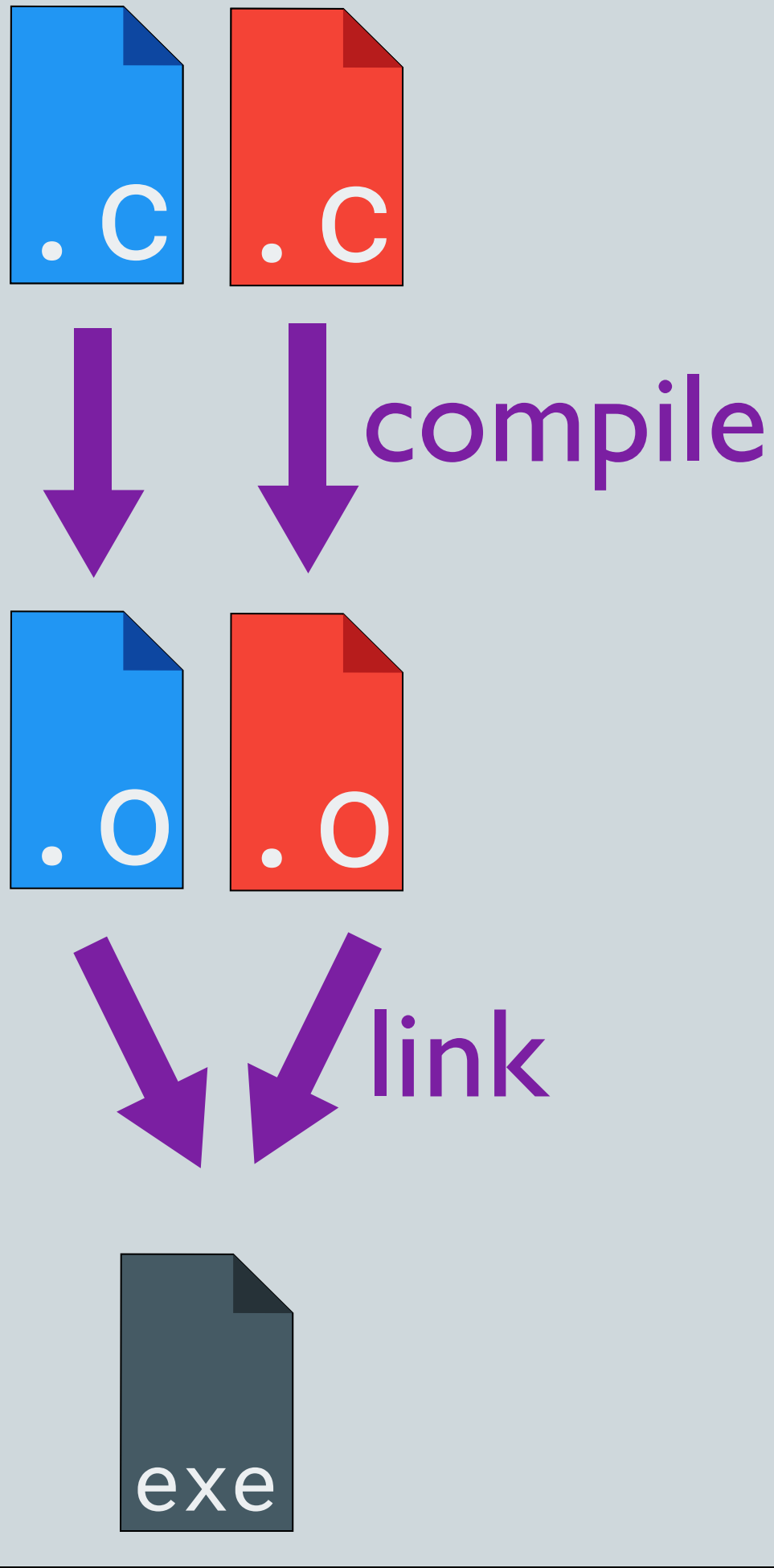
program inputs

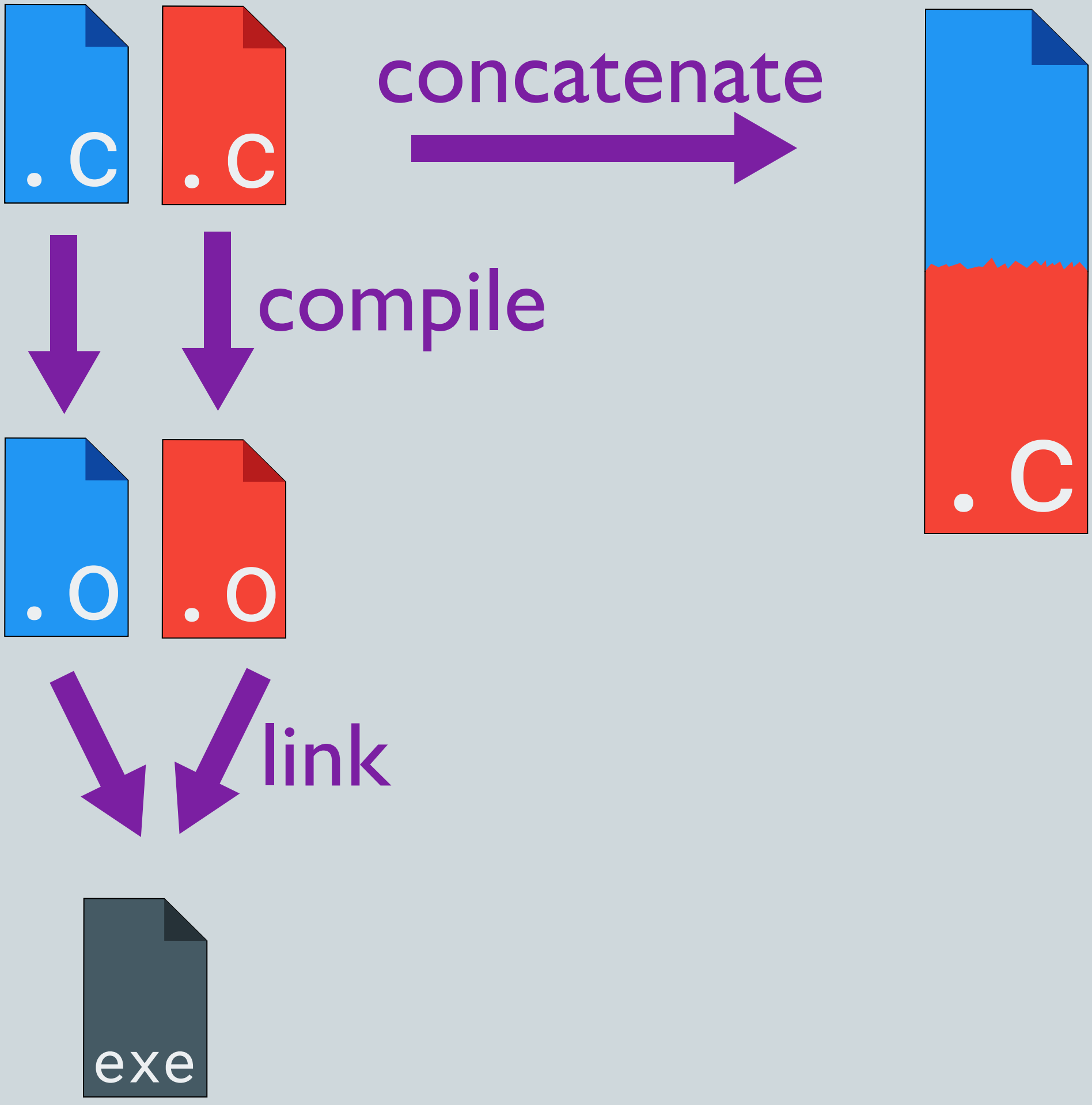
Next Talk: other things

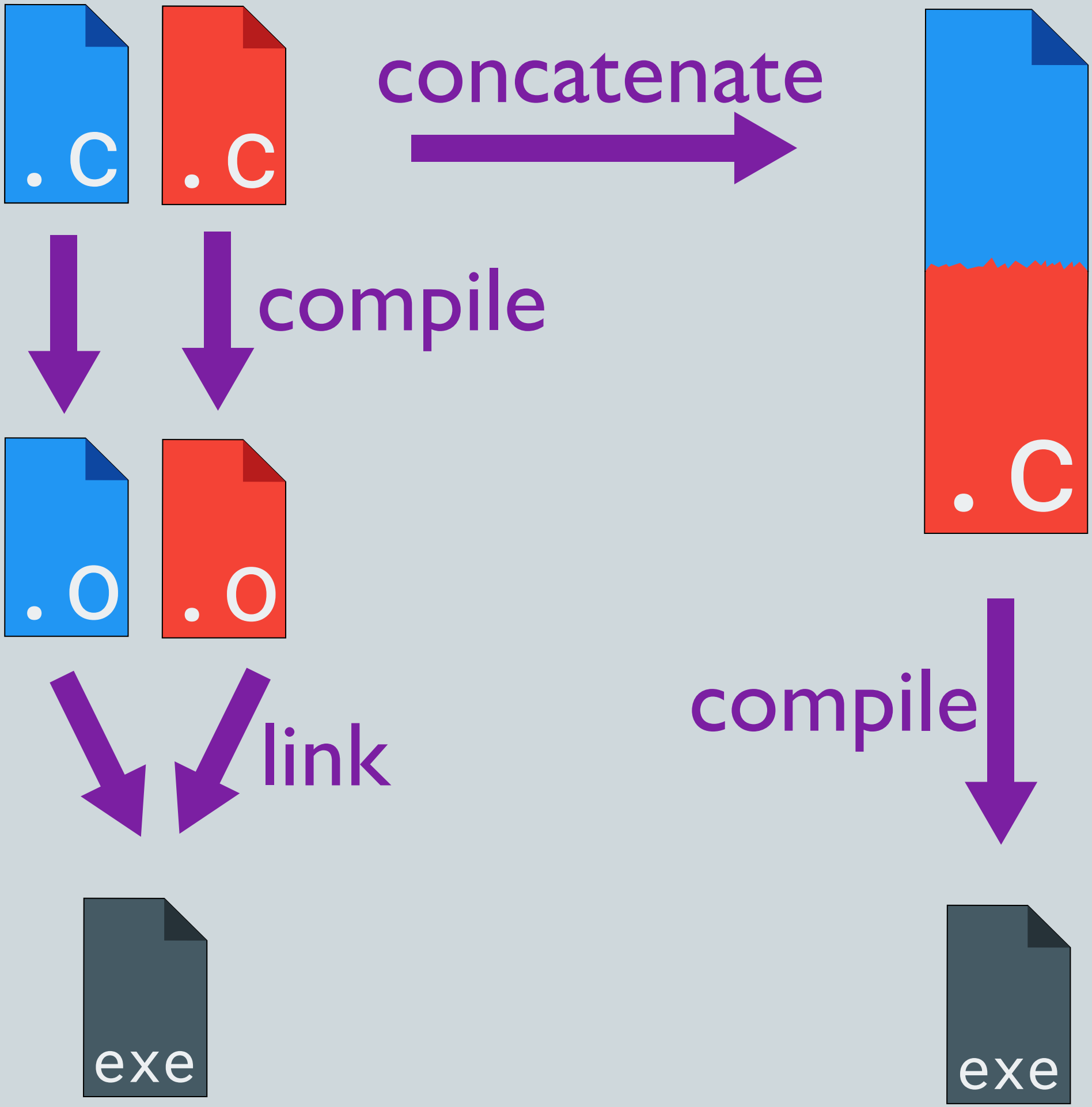
This Talk: the linker

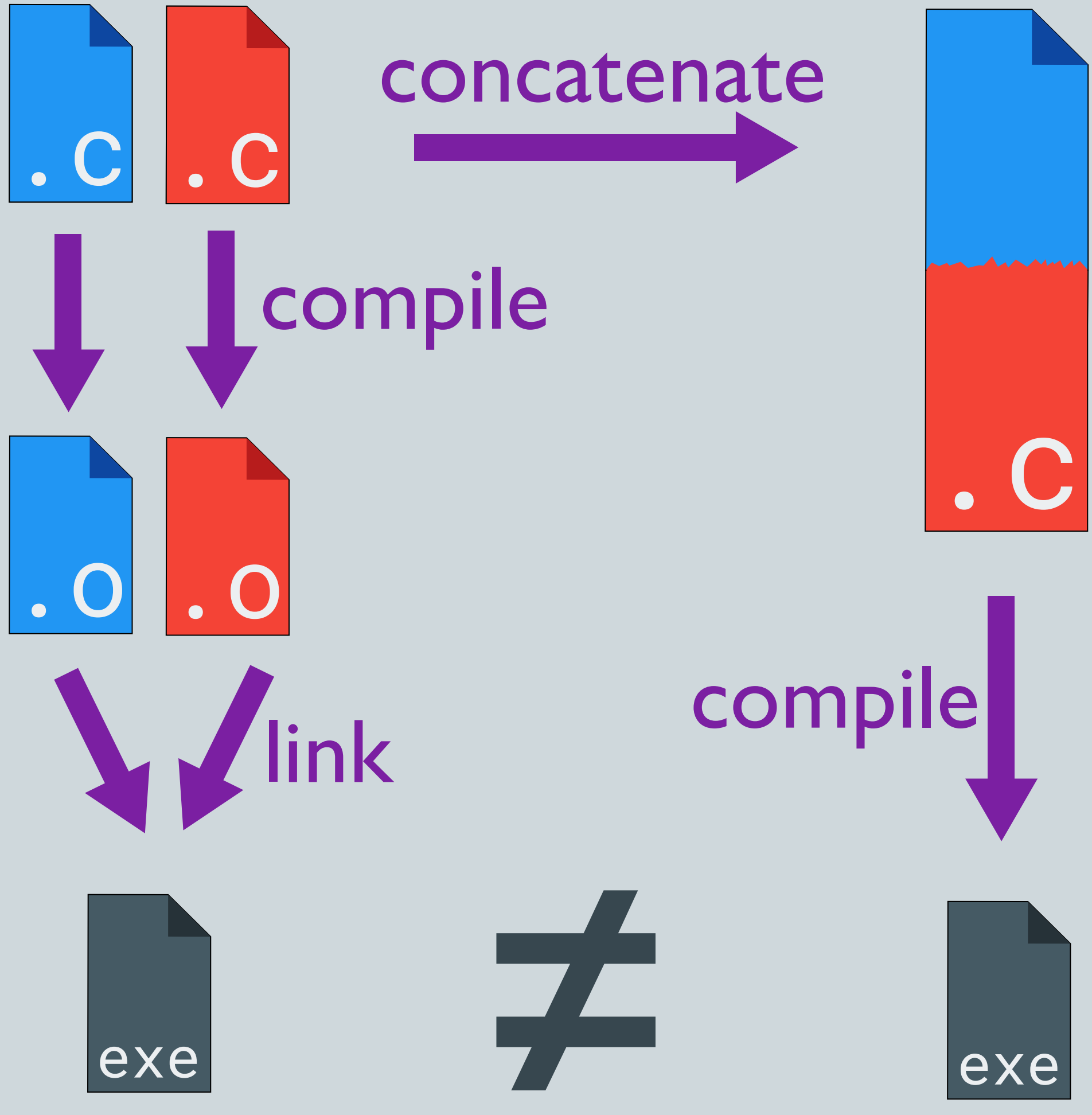

.c
.c


.c
.c
compile
.o
.o


.c
.c
compile
.o
.o
link
exe


.c
.c
concatenate
compile
.o
.o
link
exe
.c


.c
.c
concatenate
compile
.o
.o
link
exe
compile
exe

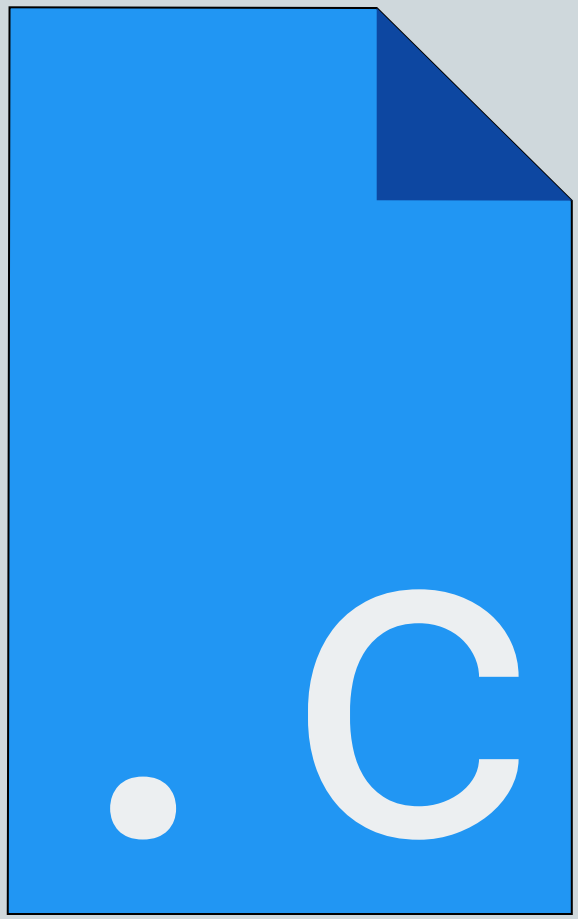

.c
.c
concatenate
compile
.o
.o
.c
link
compile
exe
≠
exe

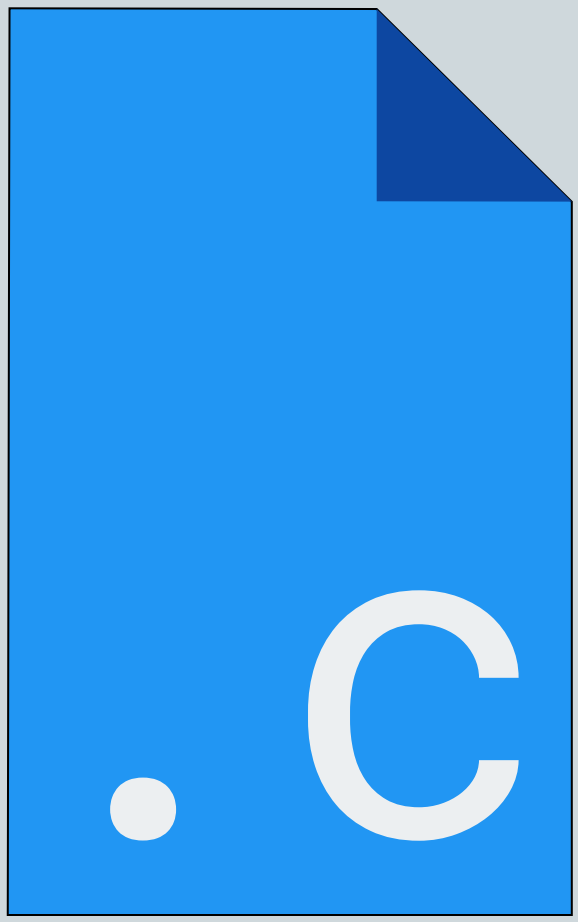
Correctness

can depend on

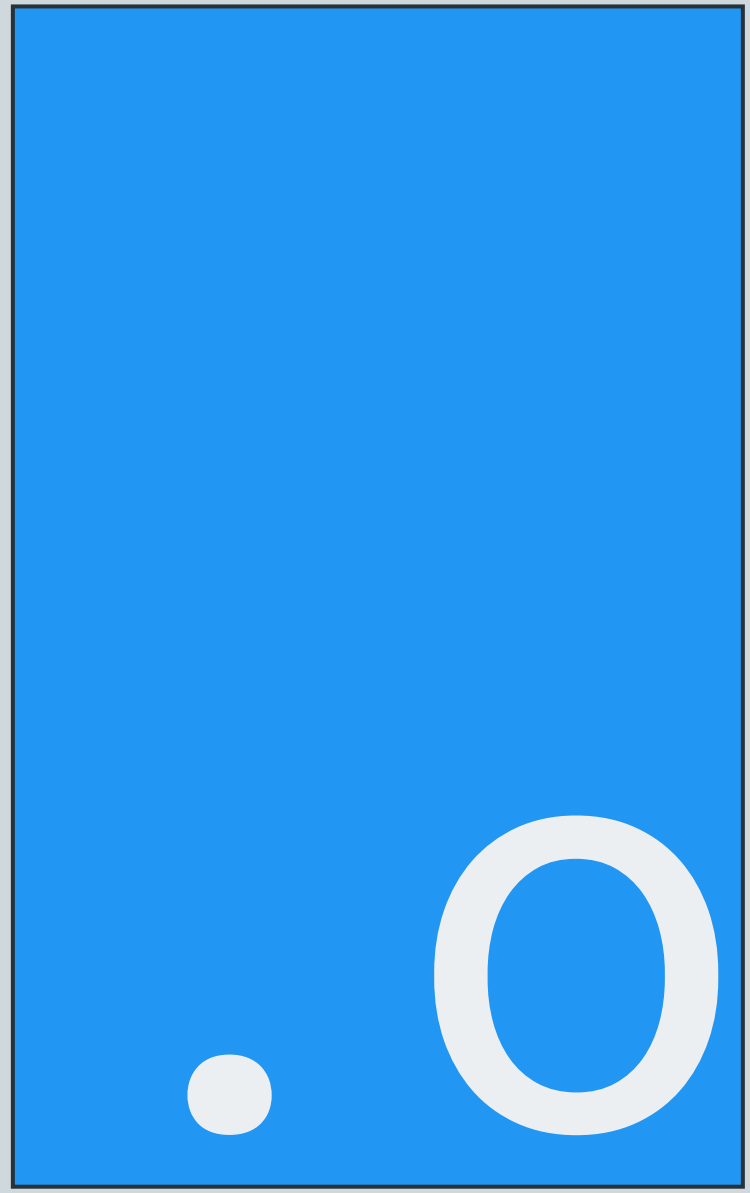
how the program

was linked

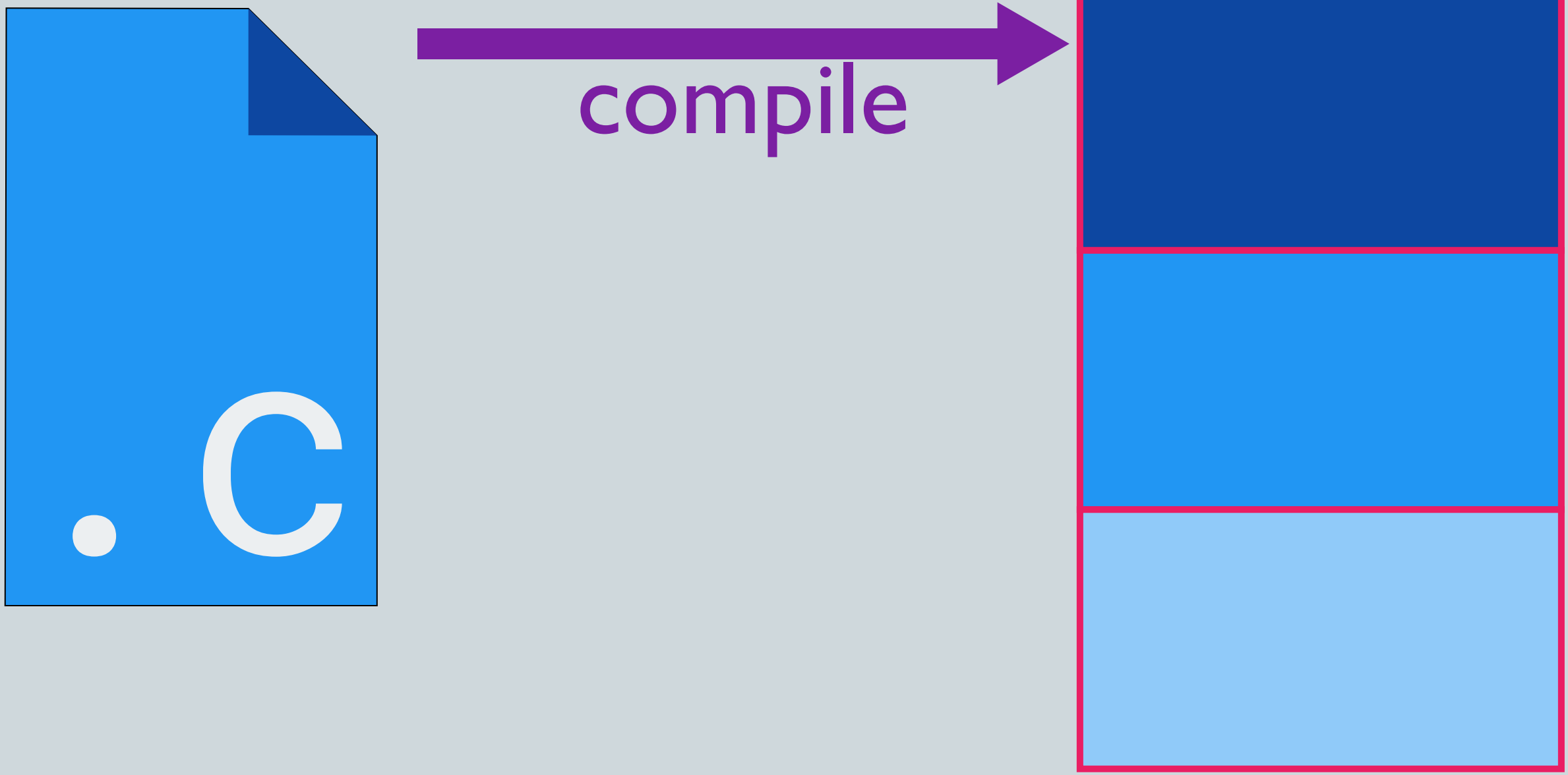

.c



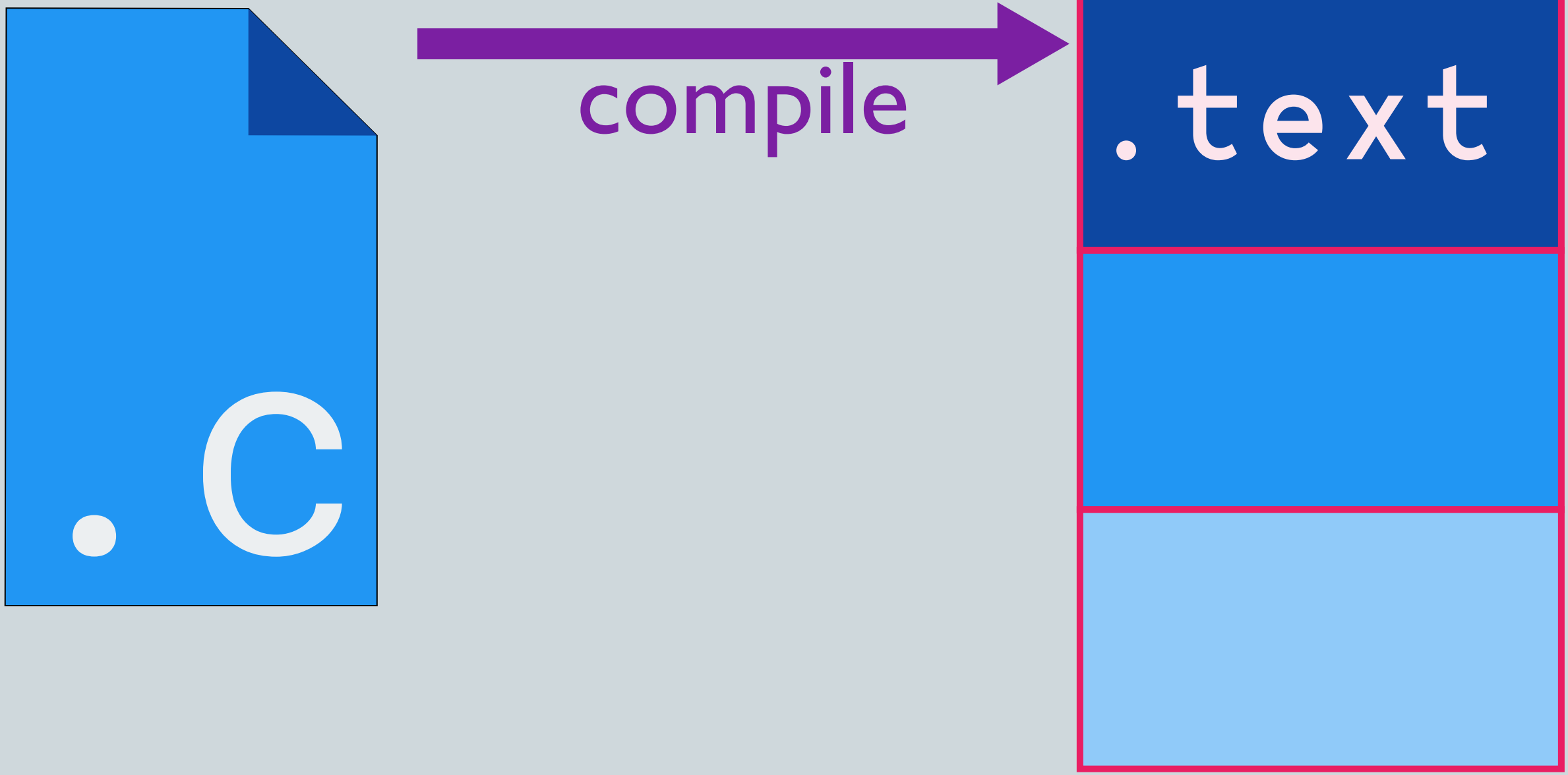
compile



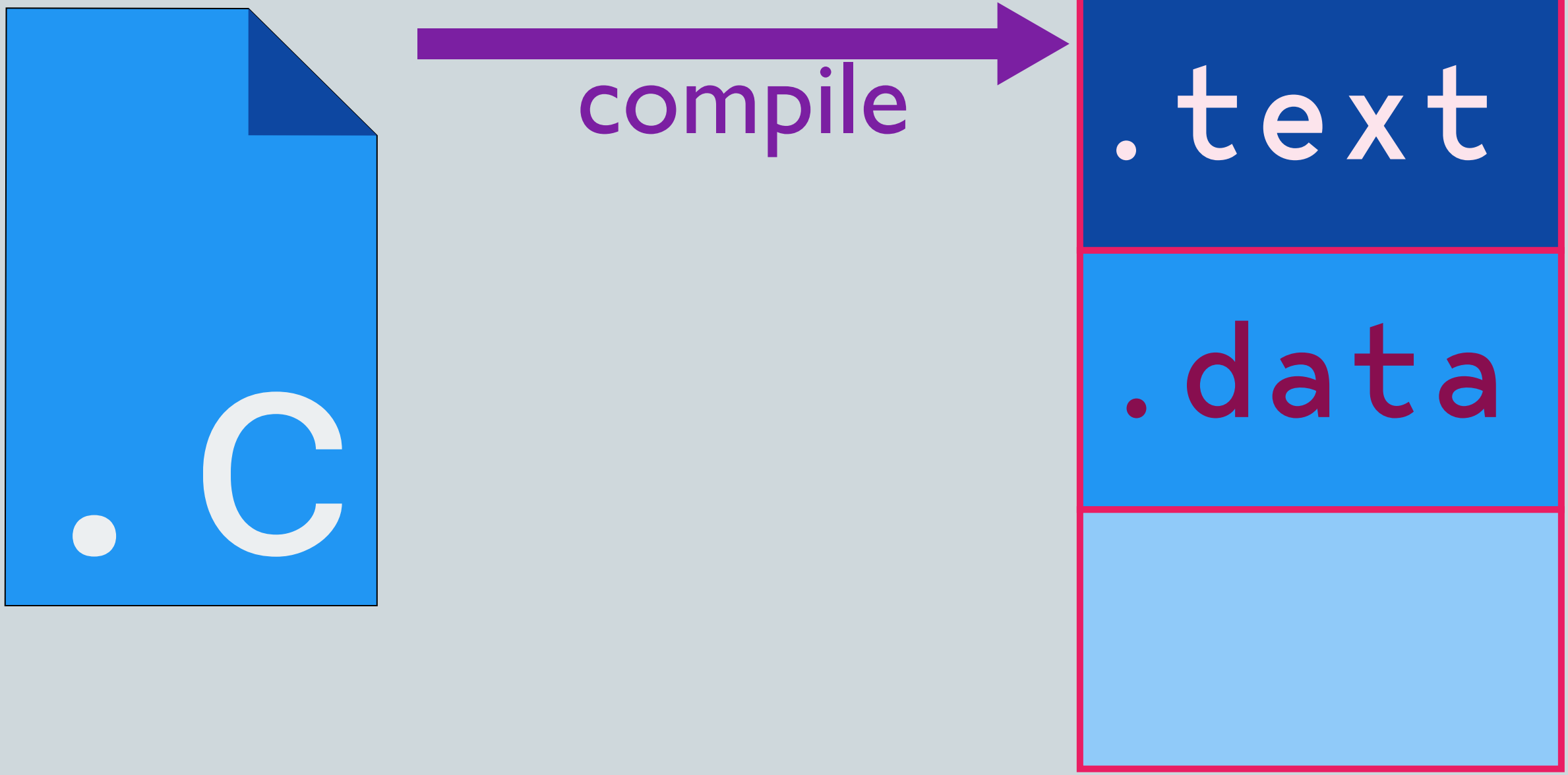
Sections



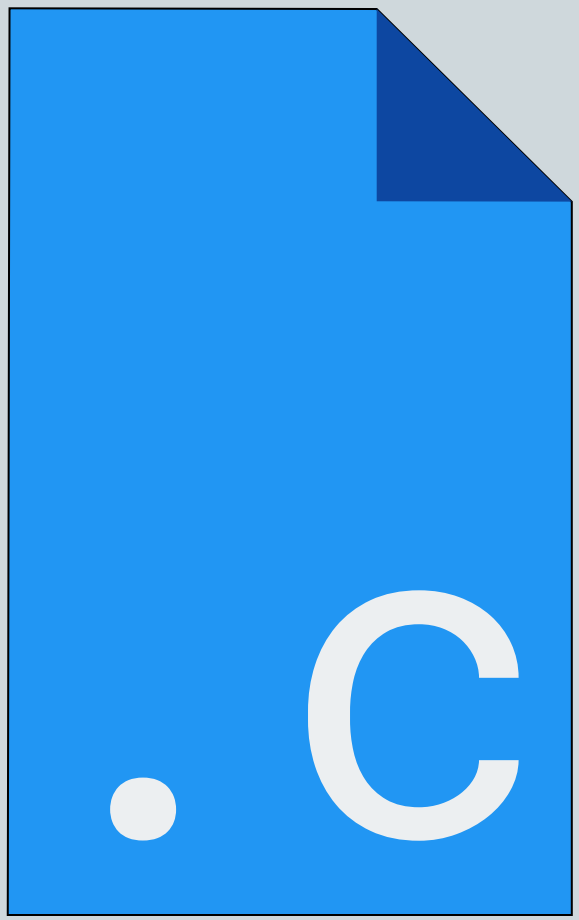
Sections



Sections



Sections



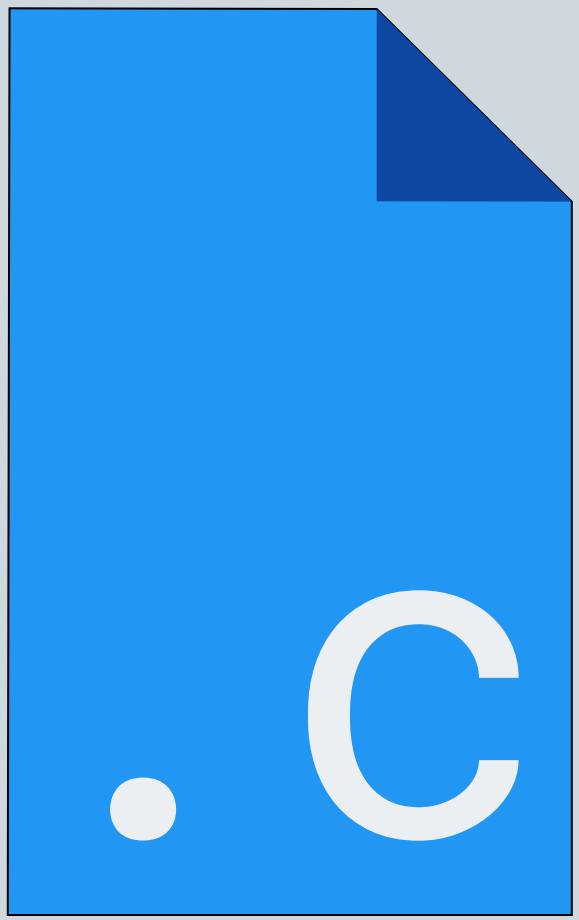
compile

.text

.data

.debug

Sections

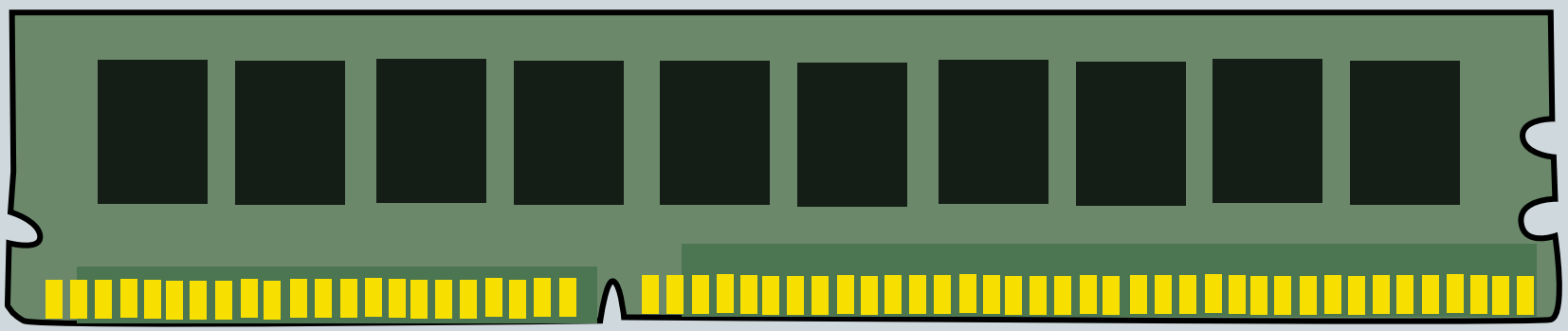


compile

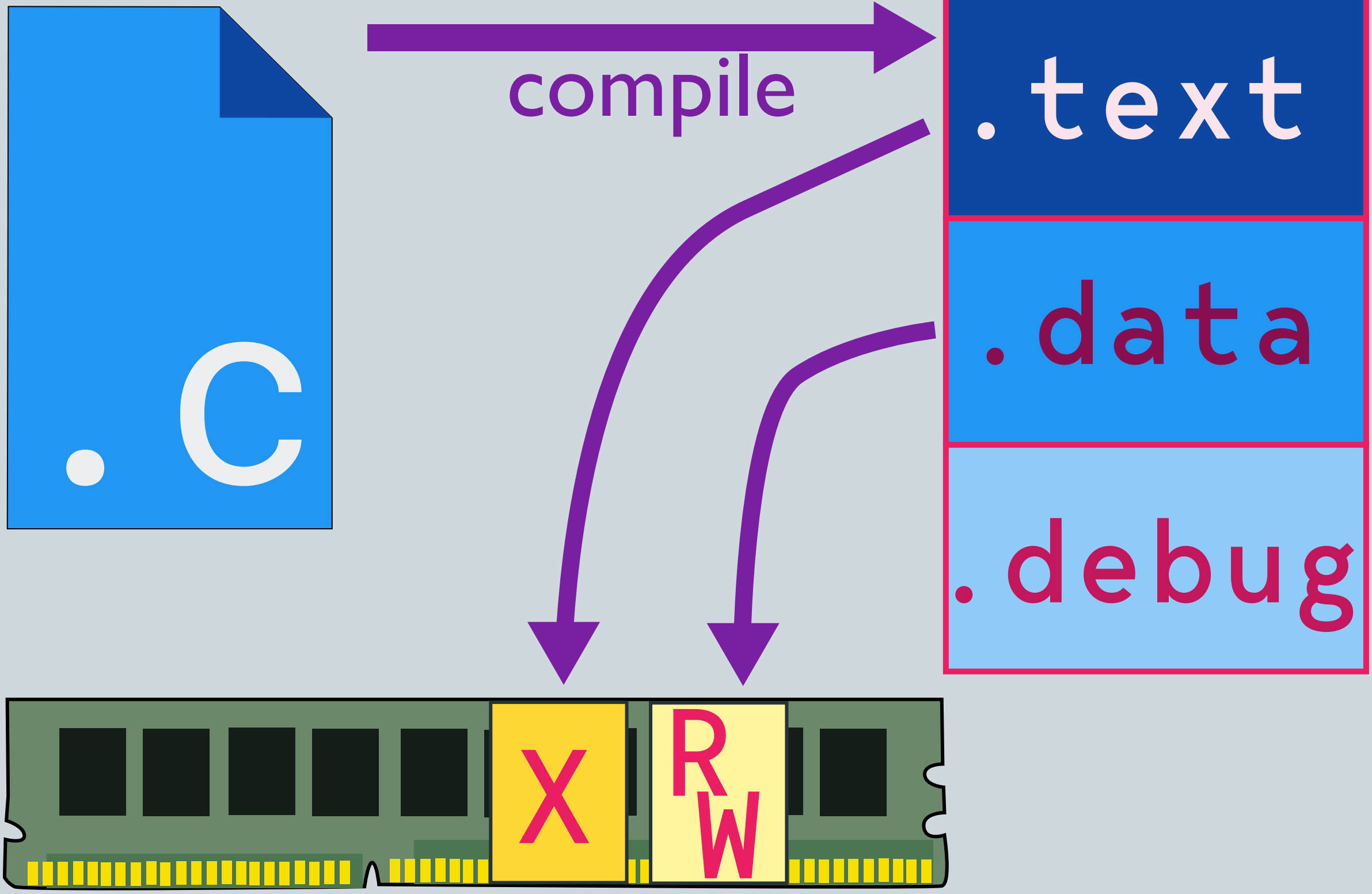
.text

.data

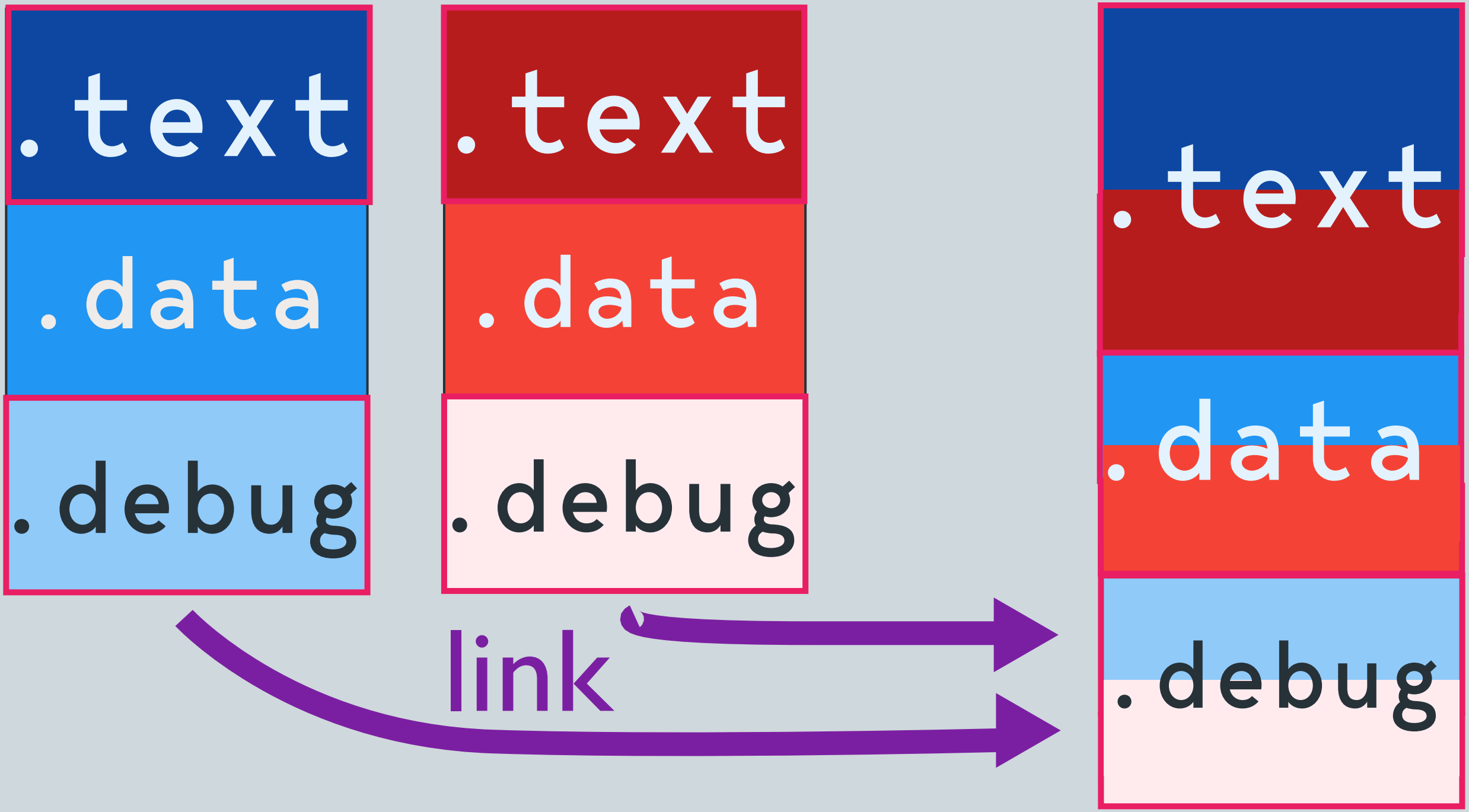
.debug

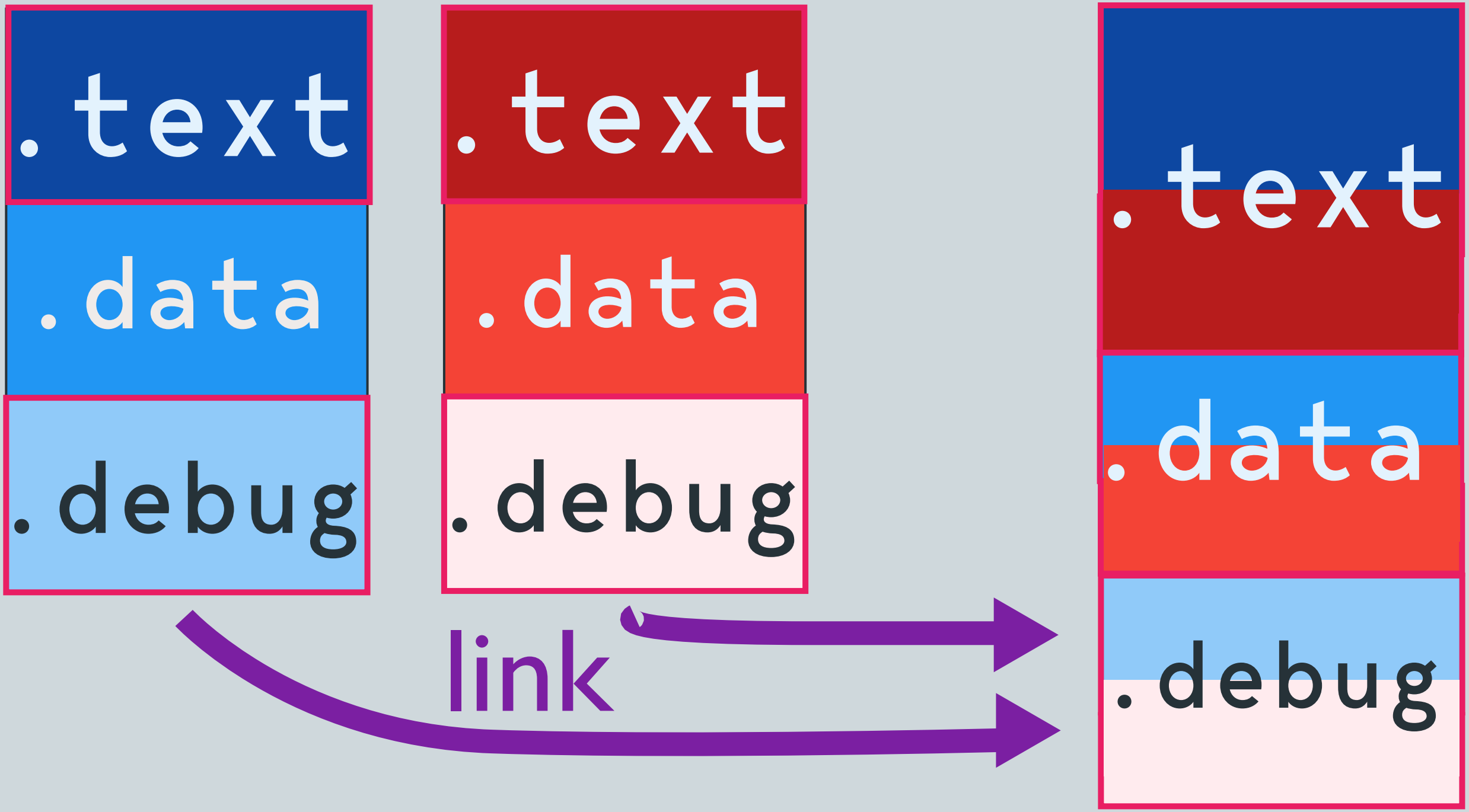


Sections

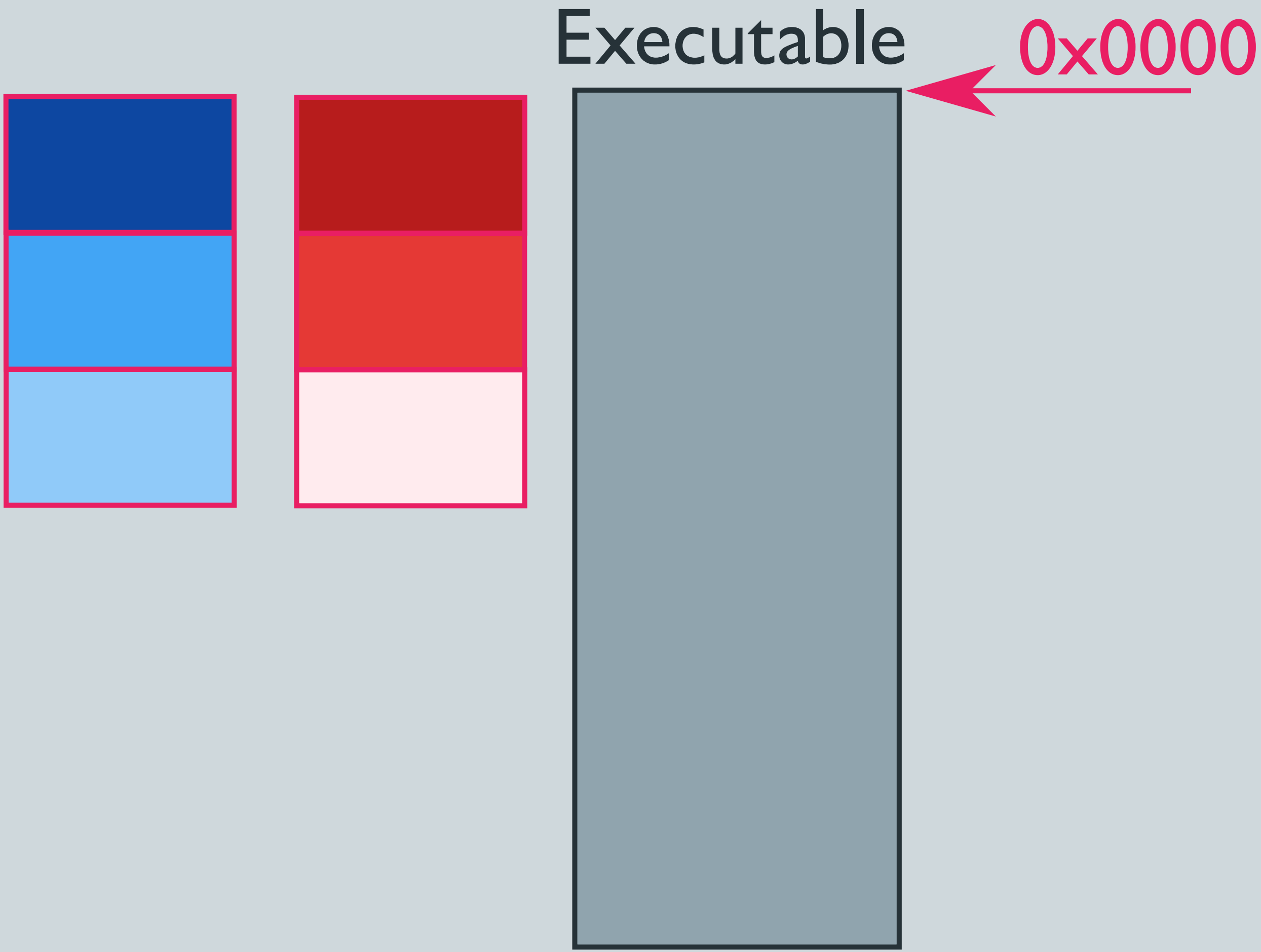


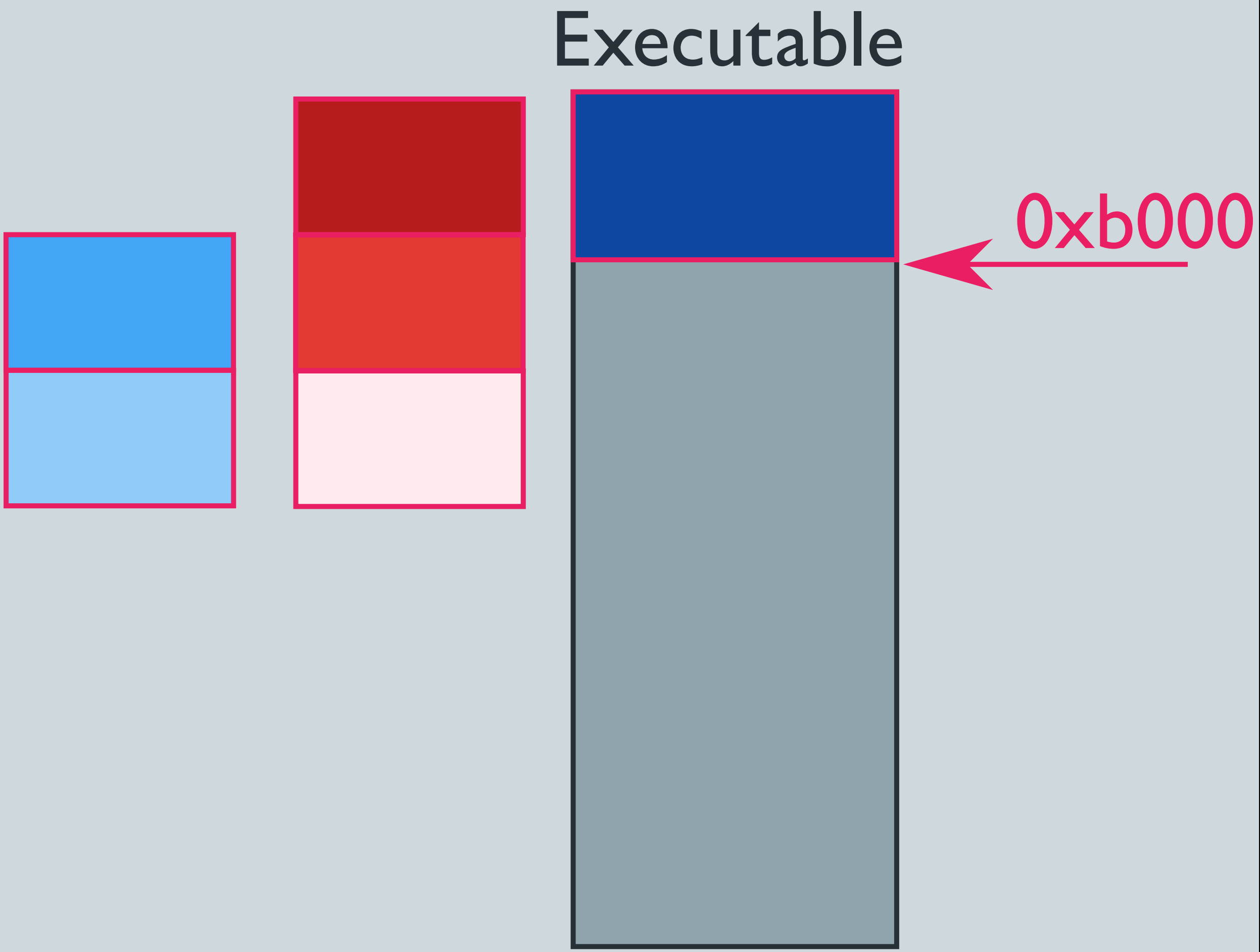
.text
.data
.debug
.text
.data
.debug


.text
.data
.debug
.text
.data
.debug
link
.text
.data
.debug

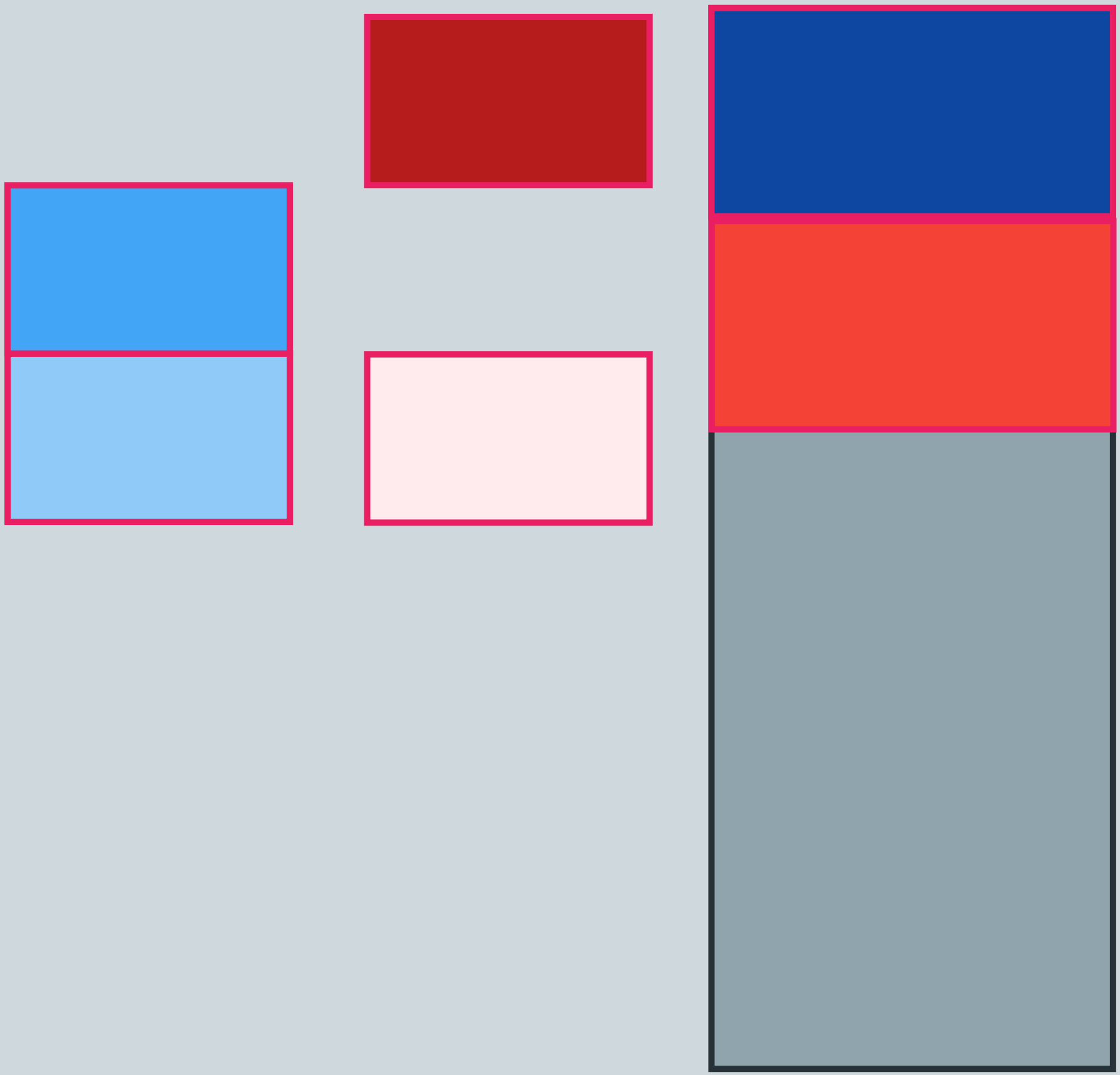


- Join sections together
- Resolve symbols

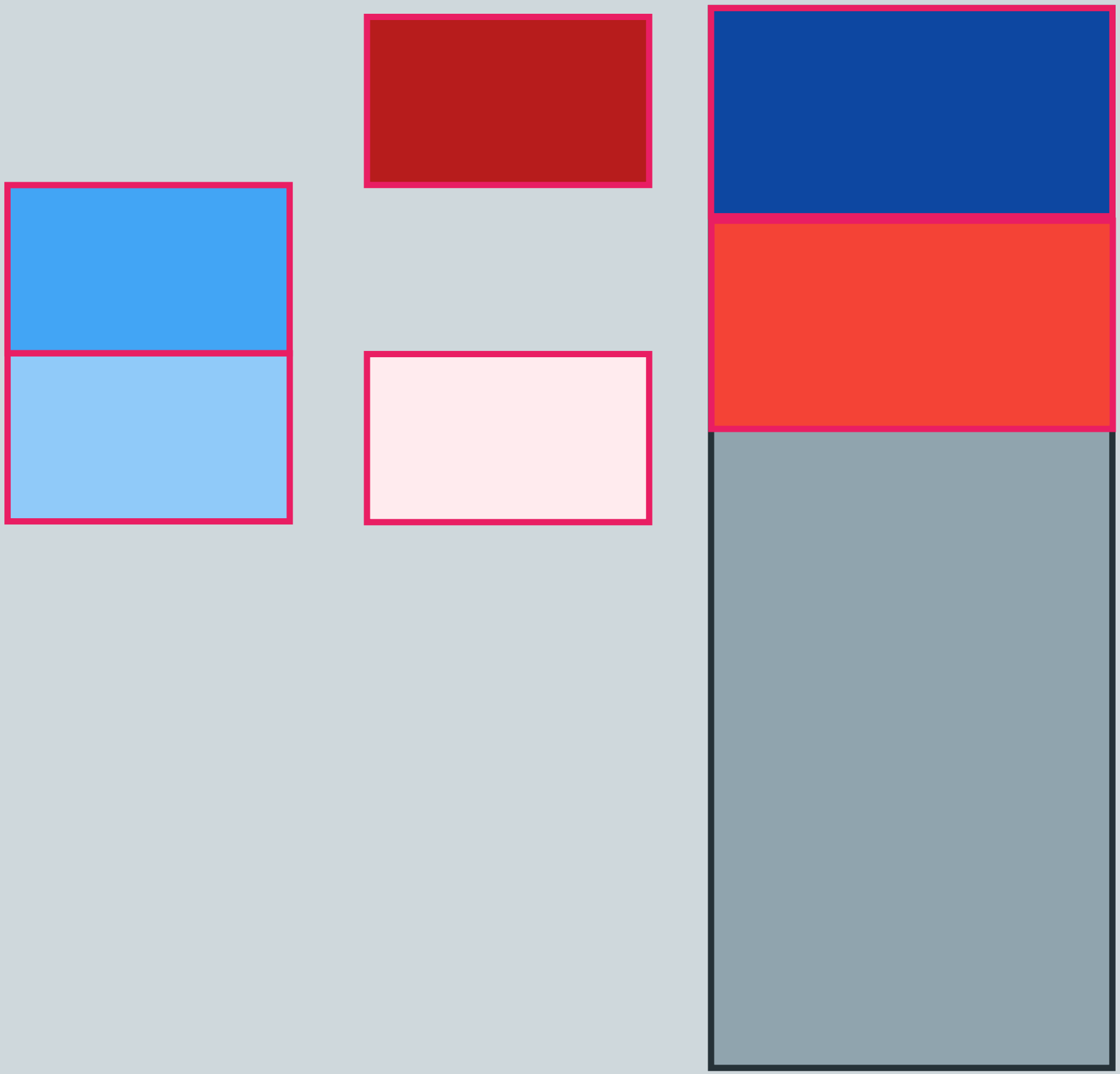

Executable
0x0000


Executable
0xb000

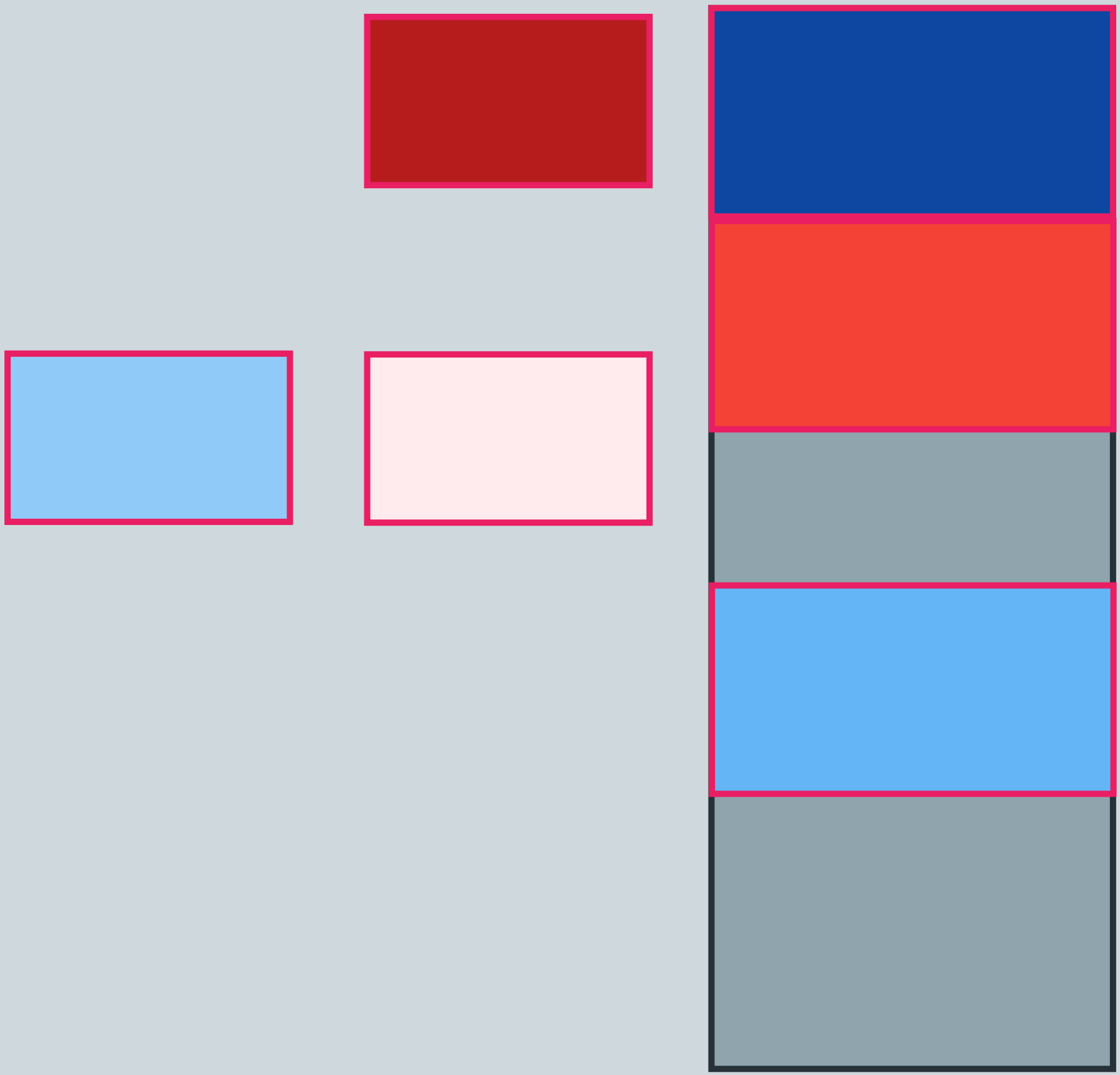
Executable



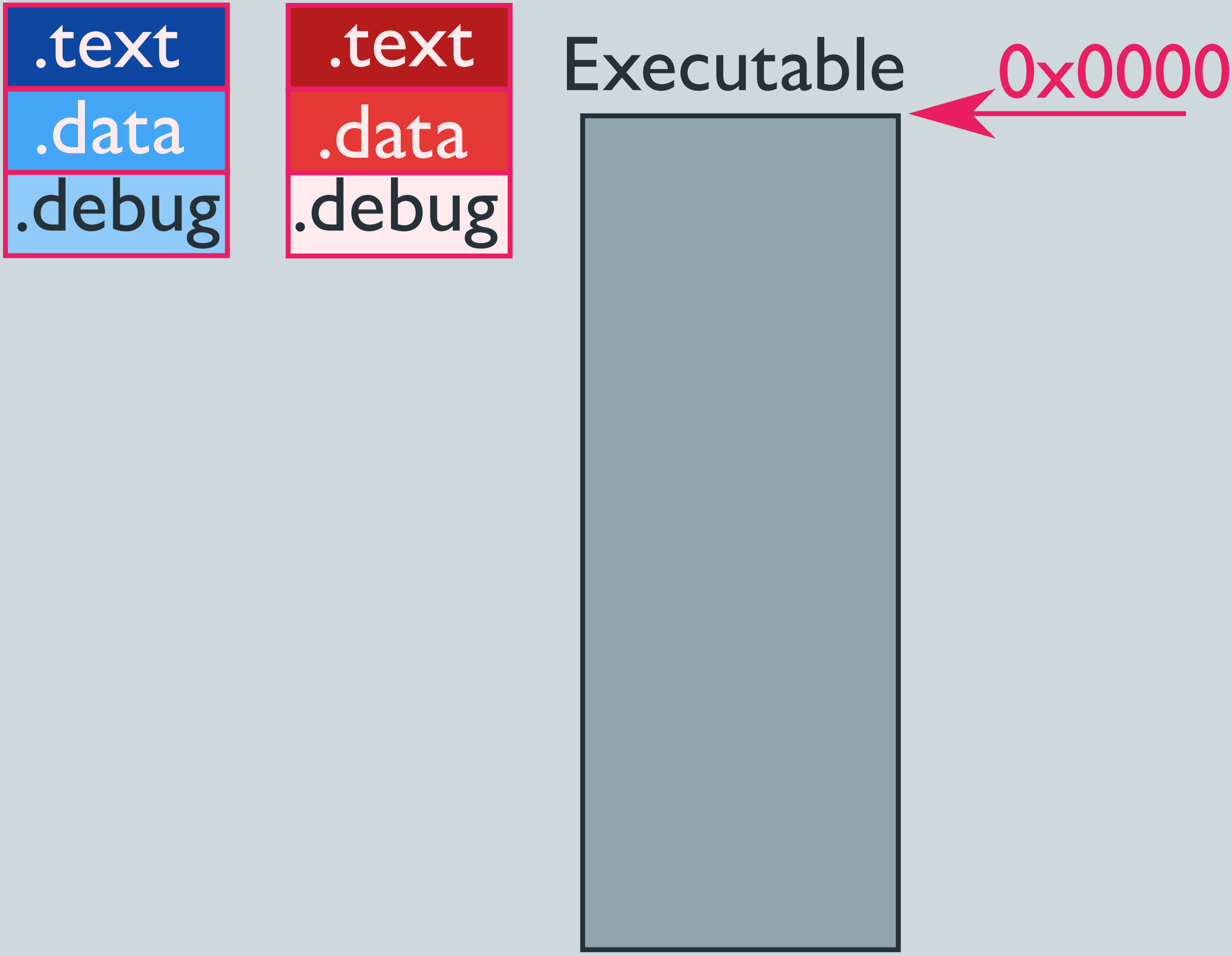
Executable

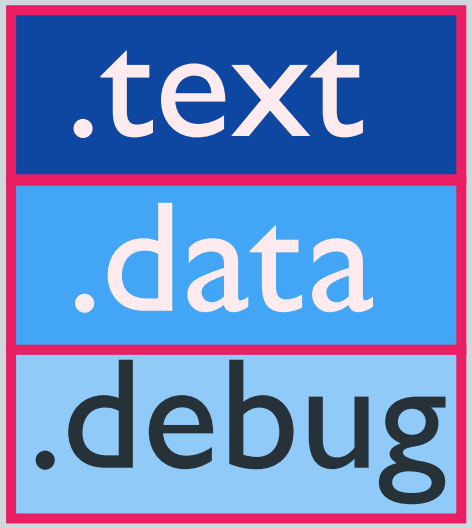


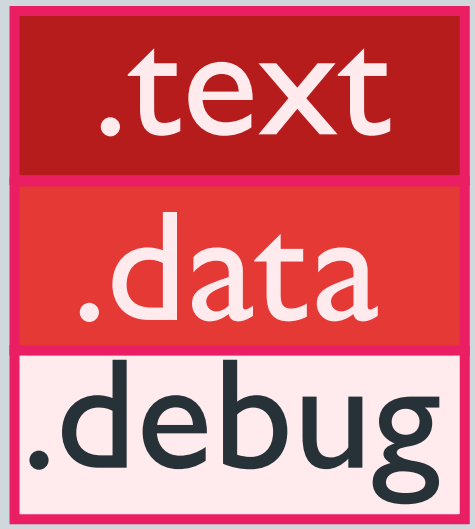
Executable



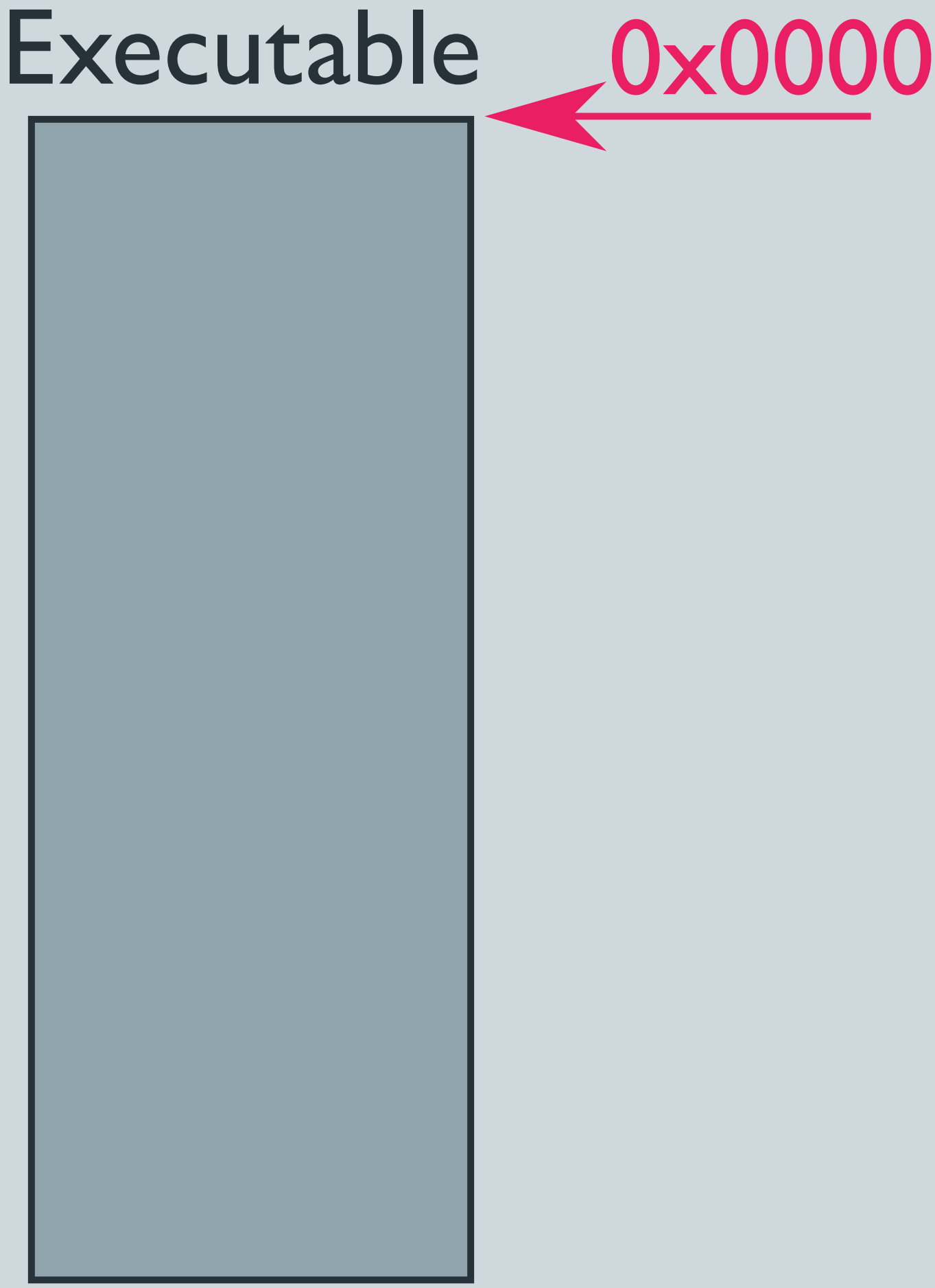
0xca00


.text
.data
.debug
.text
.data
.debug
Executable
0x0000






```
.text : {
    *(.text*)
}
.  =  . + 0x400
.data : {
    *(.data*)
}
```



.text

.data

.debug

.text

.data

.debug

Executable

0x0000

```
.text : {
    *(.text*)
}
. = . + 0x400
.data : {
    *(.data*)
}
```

.text

.data

.debug

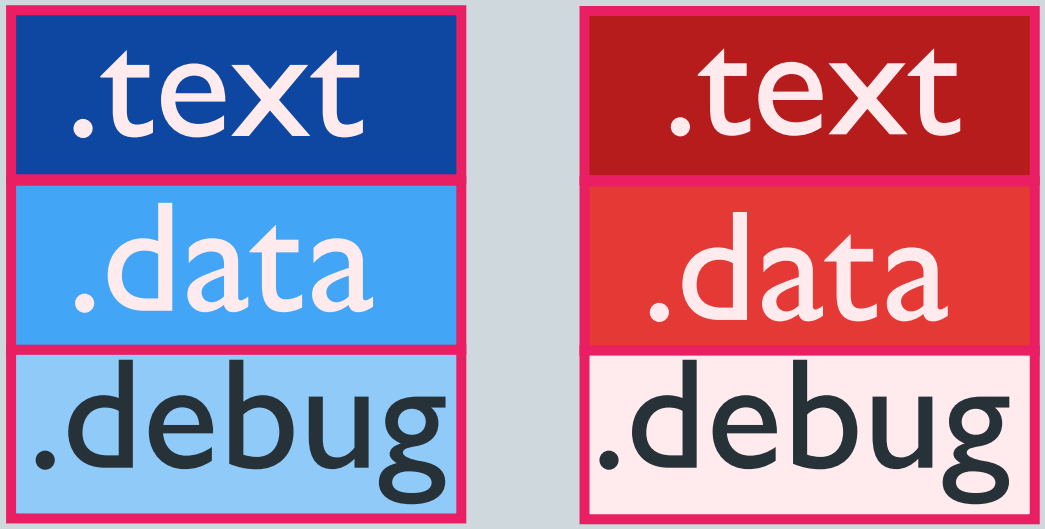
.text

.data

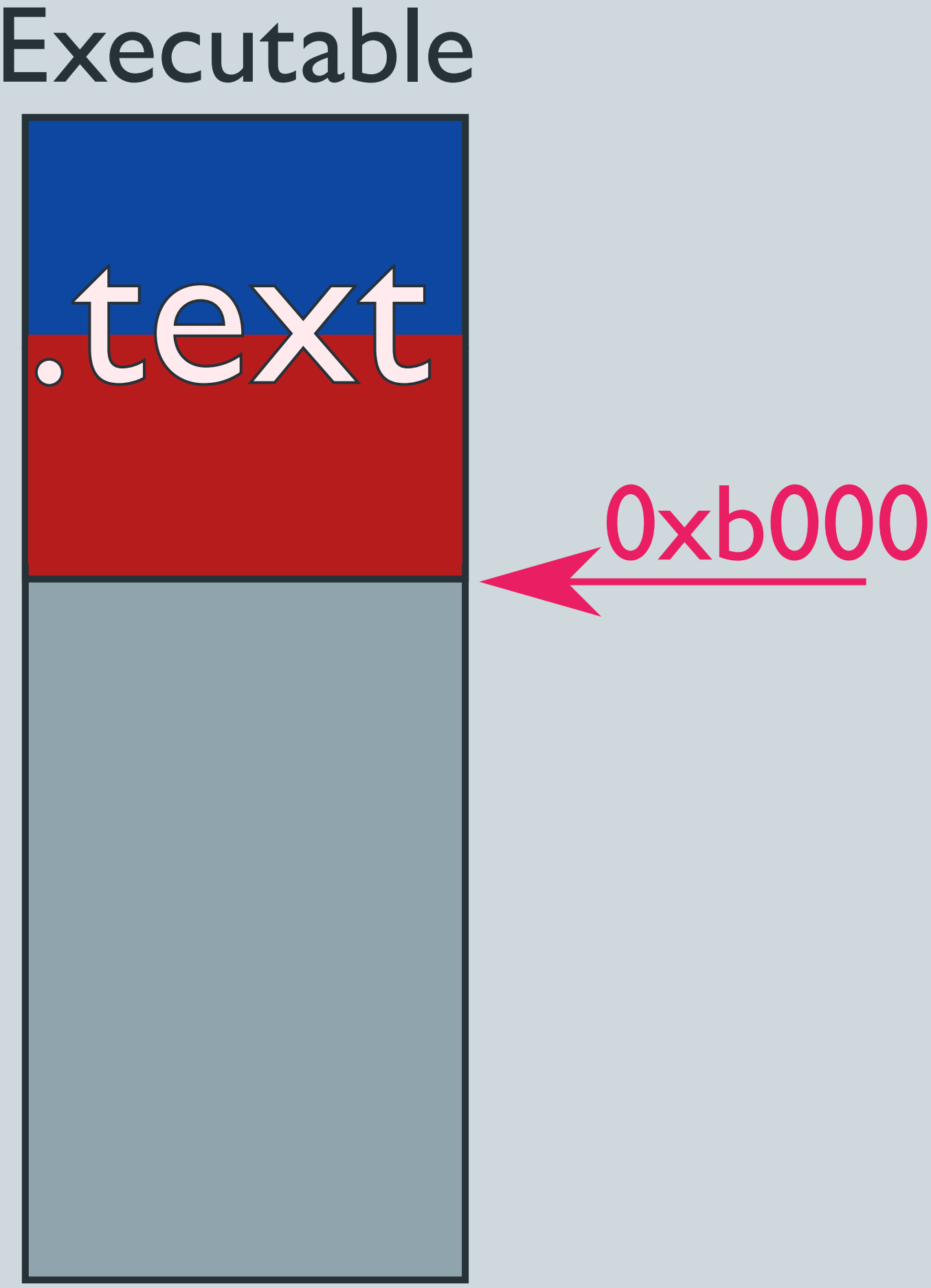
.debug

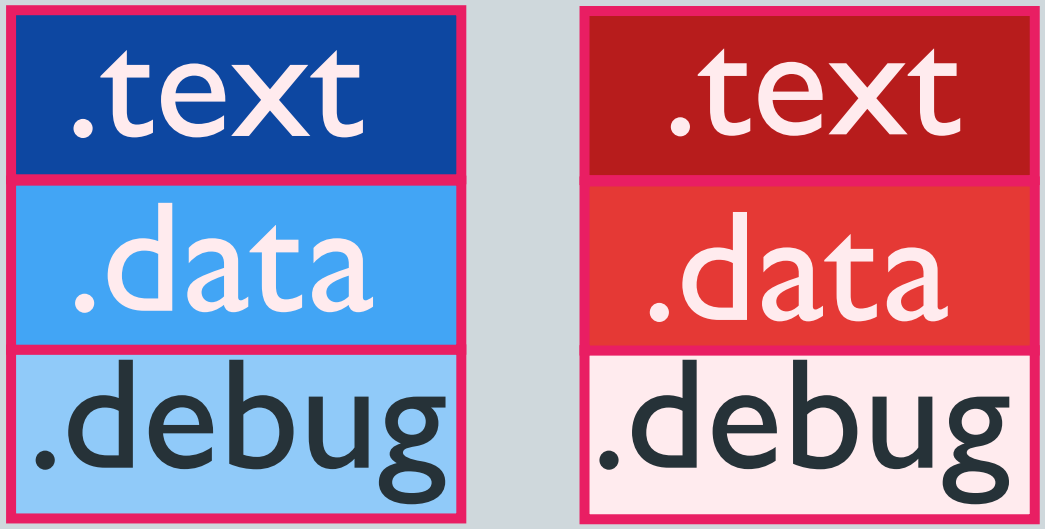
Executable

```
.text : {
    *(.text*)
}
. = . + 0x400
.data : {
    *(.data*)
}
```

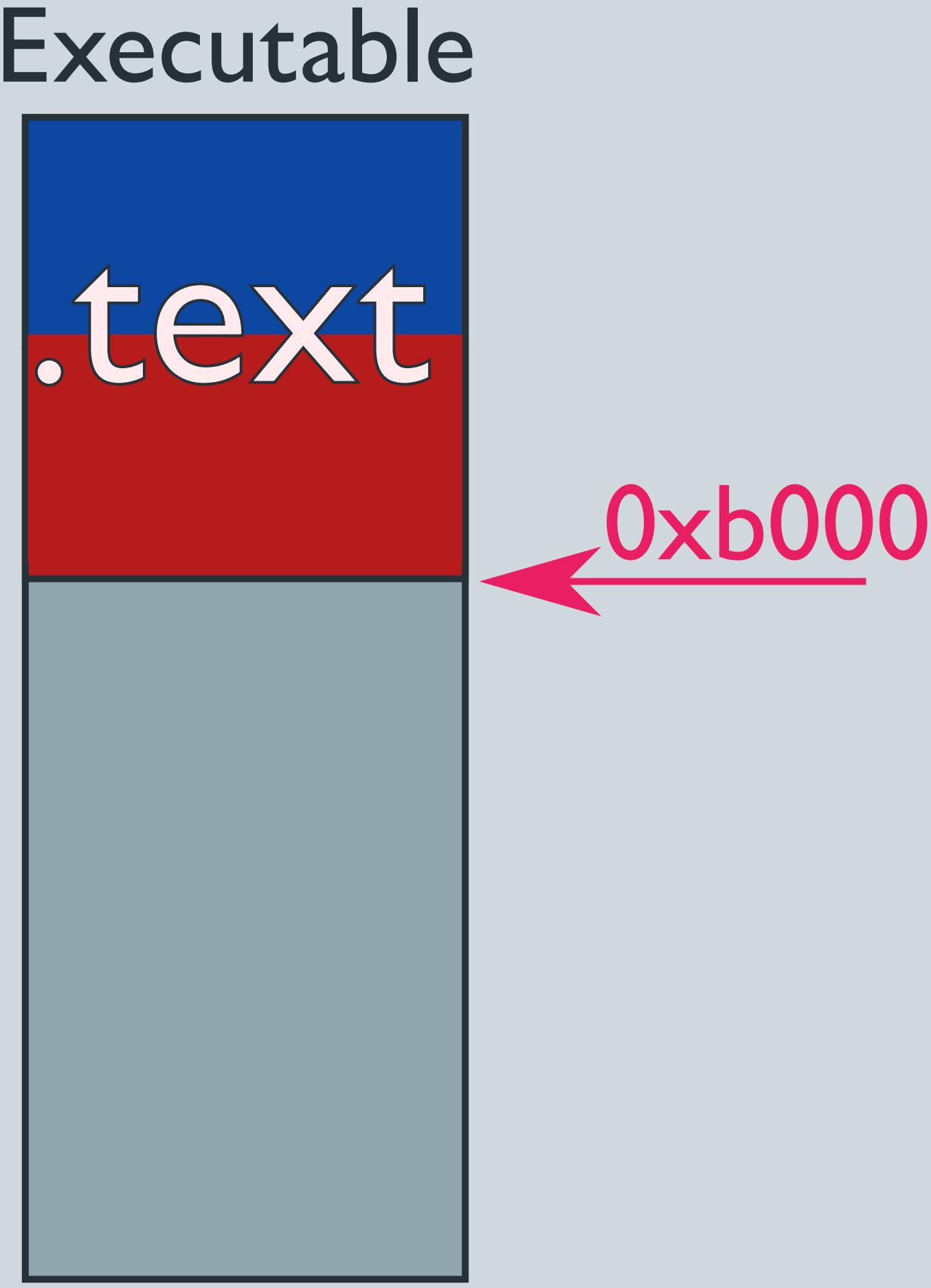


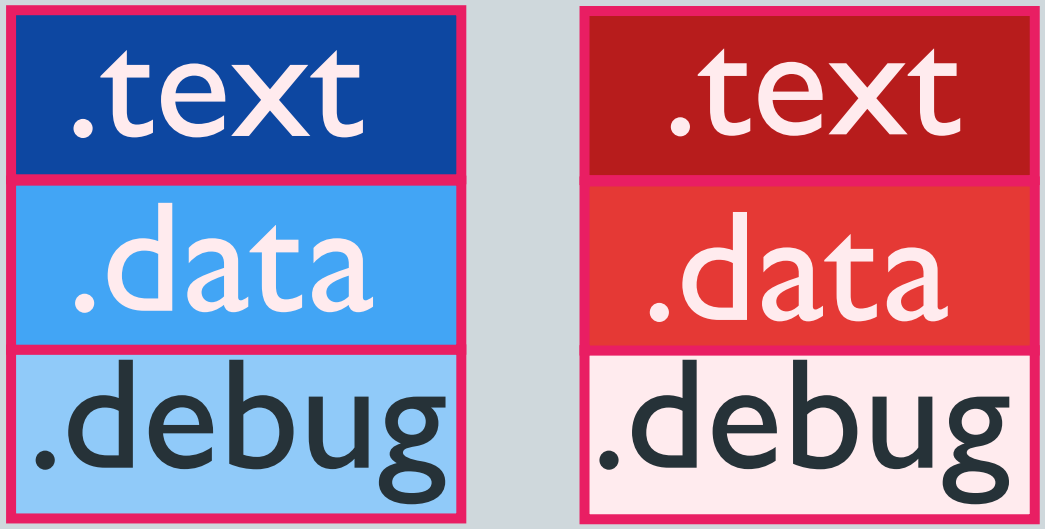

```
.text : {
    *(.text*)
}
.  =  . + 0x400
.data : {
    *(.data*)
}
```



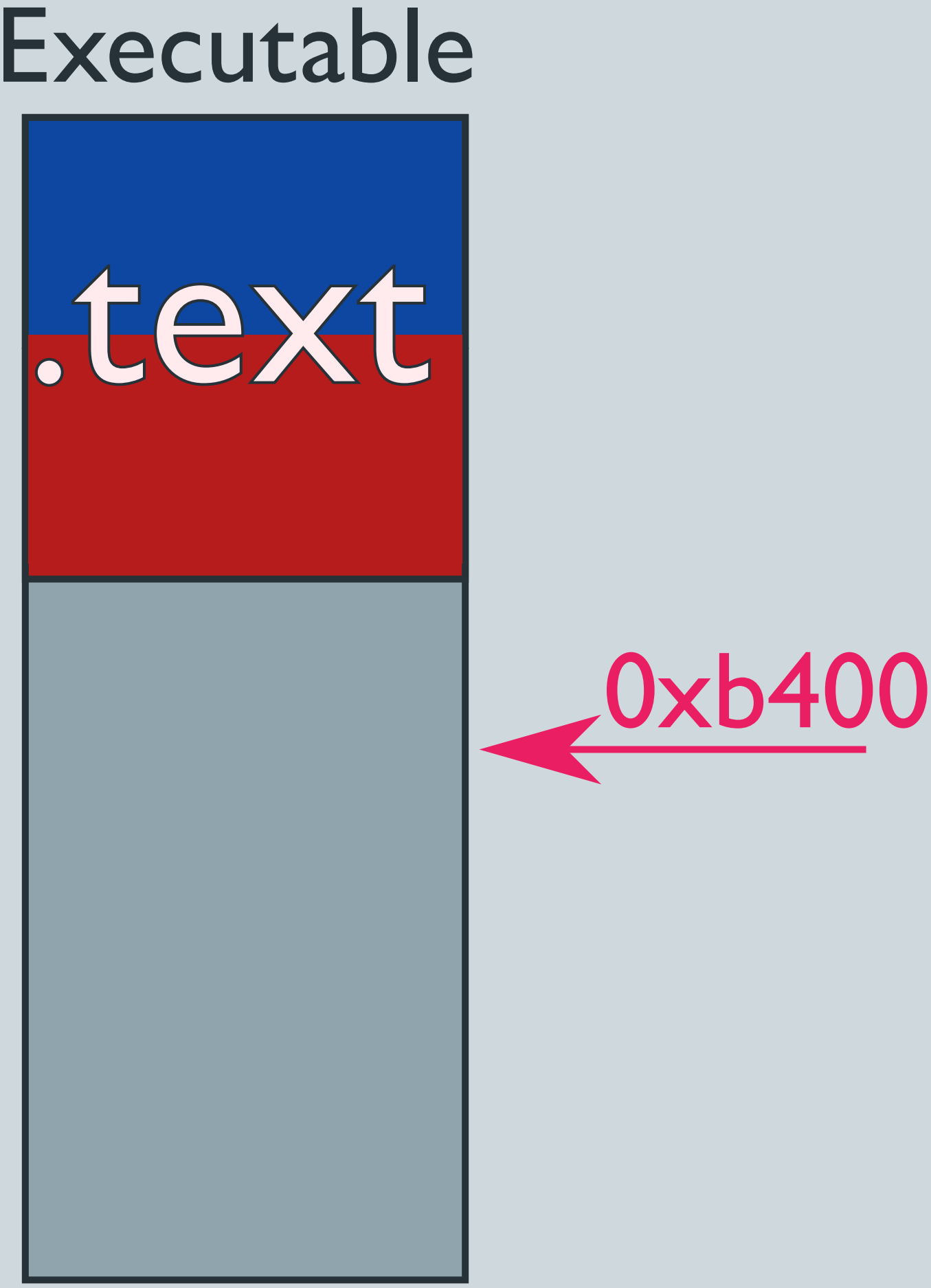


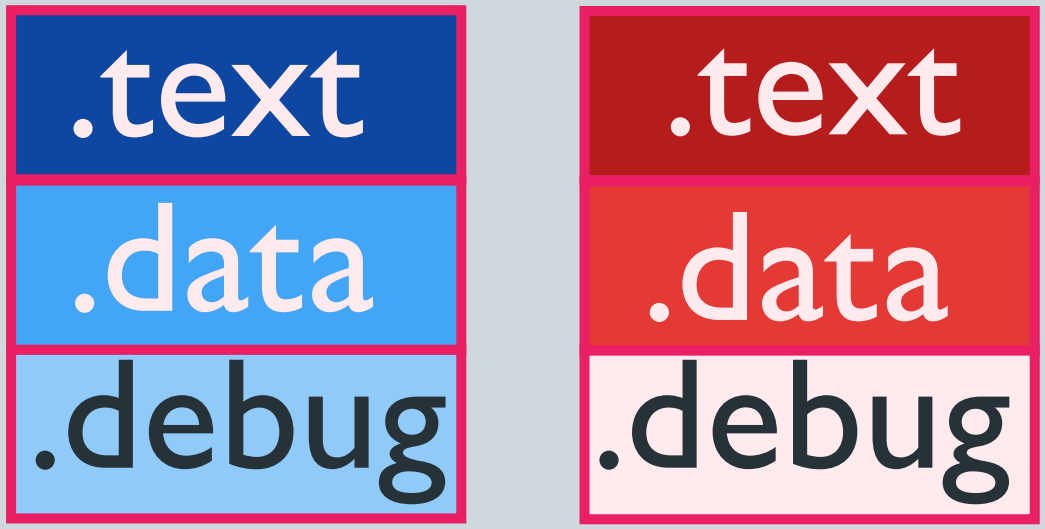

```
.text : {
    *(.text*)
}
. = . + 0x400
.data : {
    *(.data*)
}
```





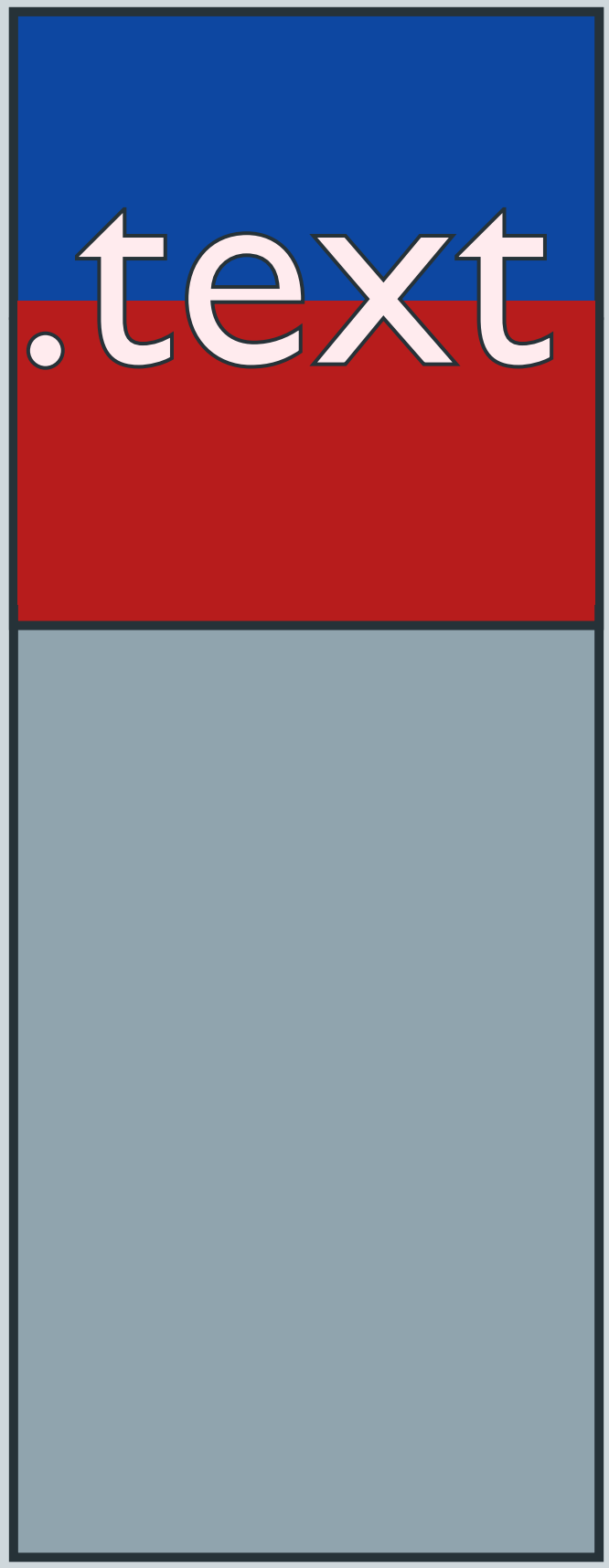

```
.text : {
    *(.text*)
}
.  =  . + 0x400
.data : {
    *(.data*)
}
```






```
.text : {
    *(.text*)
}
. = . + 0x400
.data : {
    *(.data*)
}
```

Executable



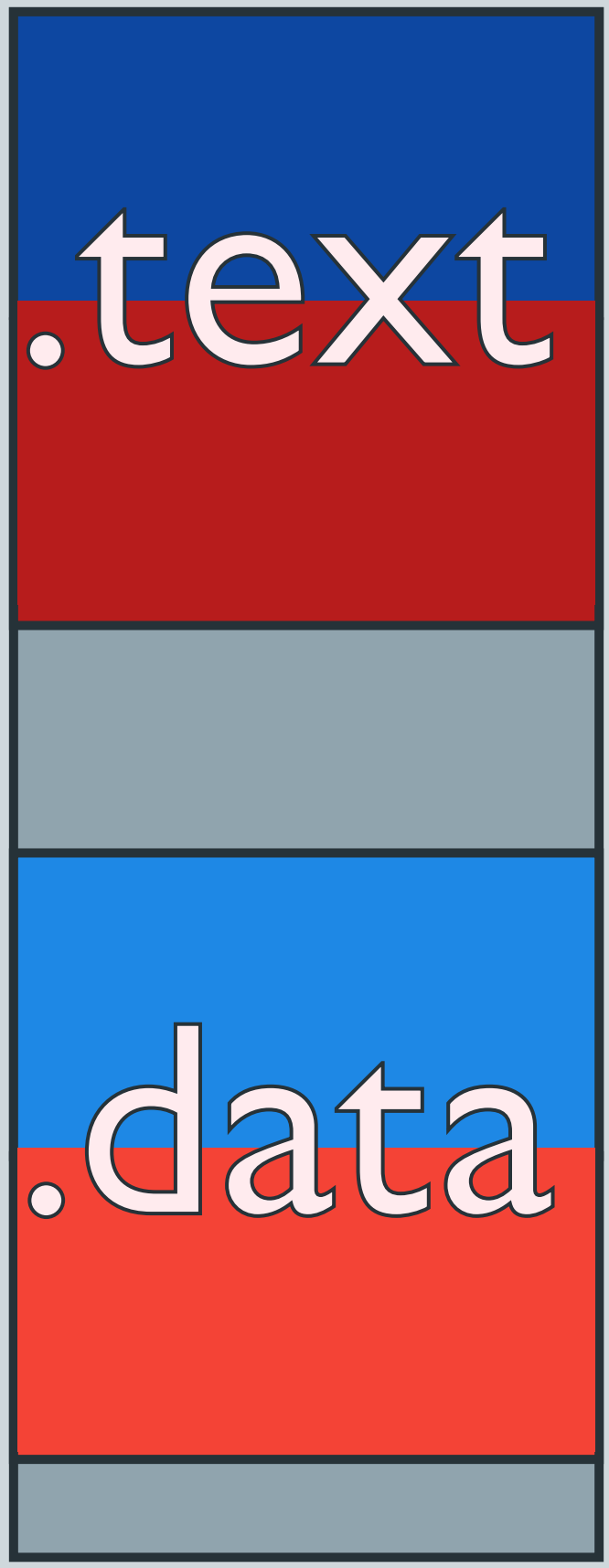
0xb400

.text
.data
.debug

.text
.data
.debug

```
.text : {
    *(.text*)
}
. = . + 0x400
.data : {
    *(.data*)
}
```

Executable

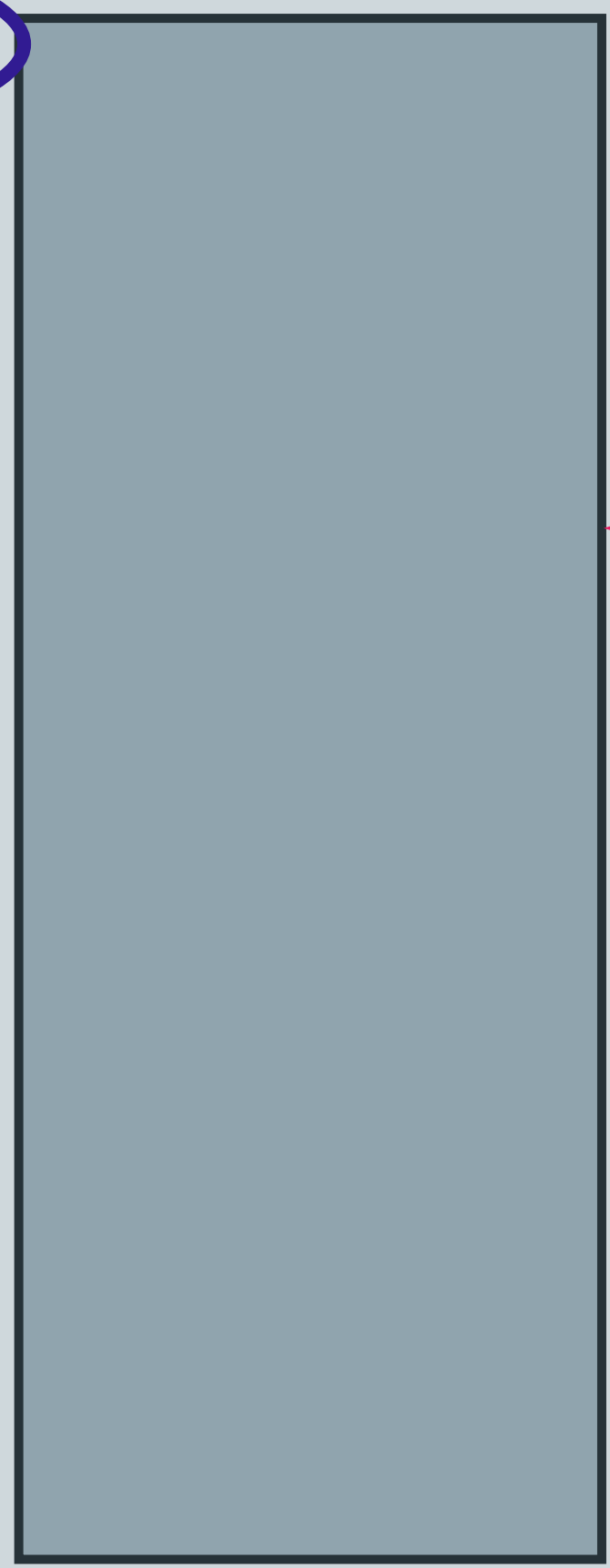


```
.text : {
    text_start = .
    *(.text*)
    text_end = .
}

text_size =
  SIZEOF(.text)
```

```
.text : {
    text_start = .
    *(.text*)
    text_end = .
}

text_size =
  SIZEOF(.text)
```

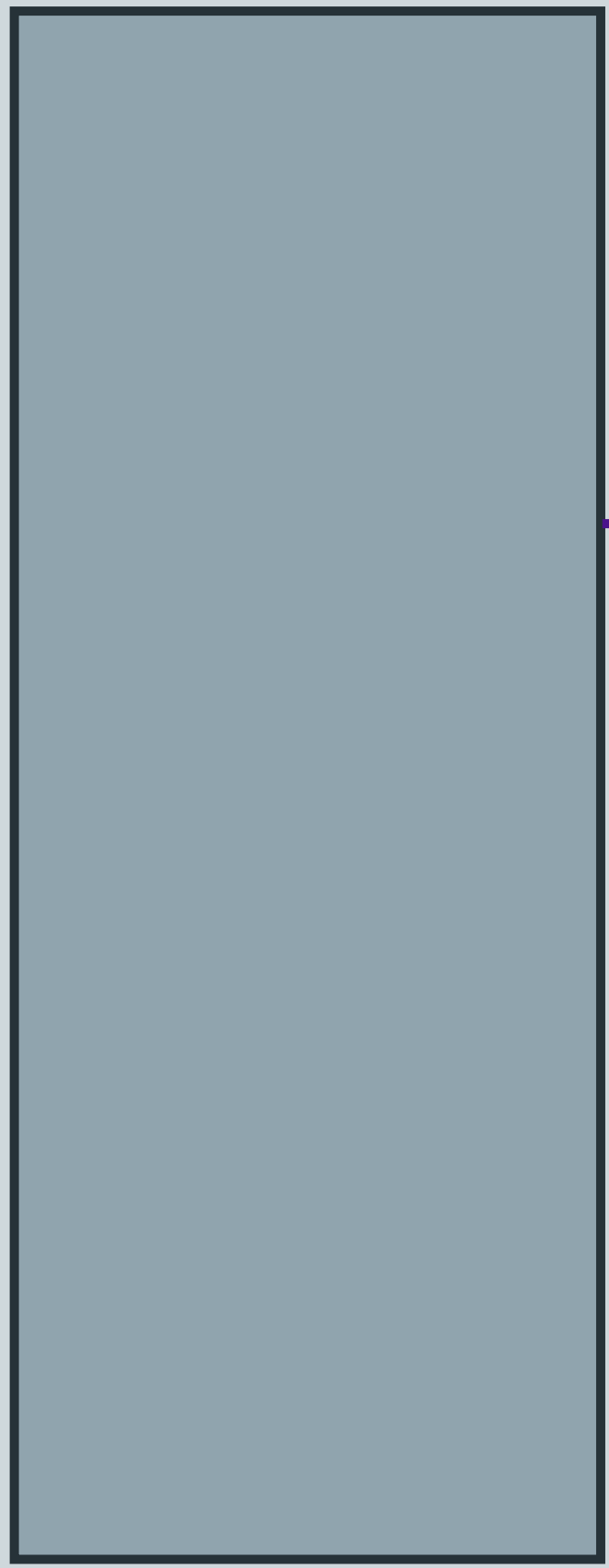


```
.text : {
    text_start = .
    *(.text*)
    text_end = .
}

text_size =
  SIZEOF(.text)
```

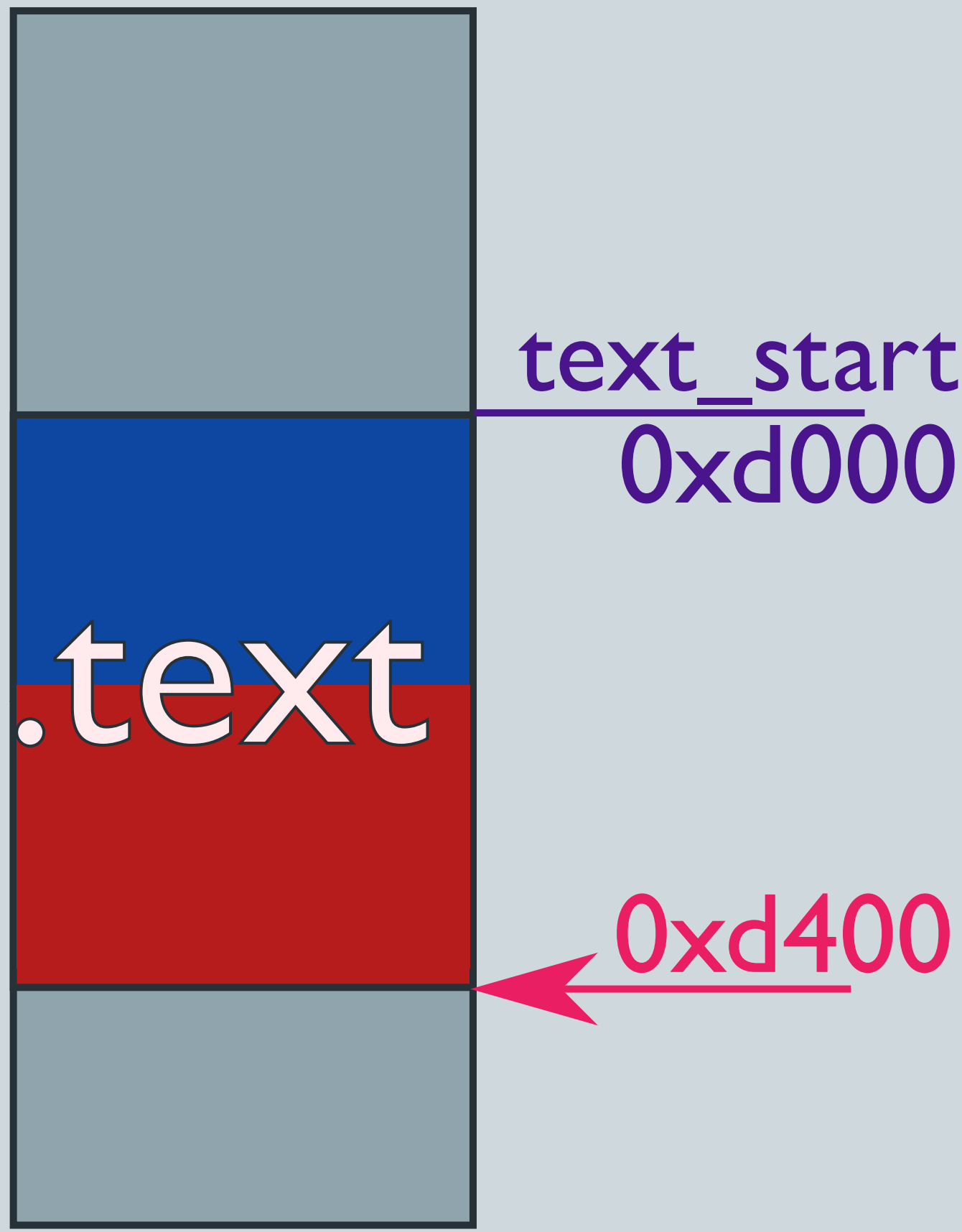
```
.text : {
    text_start = .
   *(.text*)
    text_end = .
}

text_size =
  SIZEOF(.text)
```



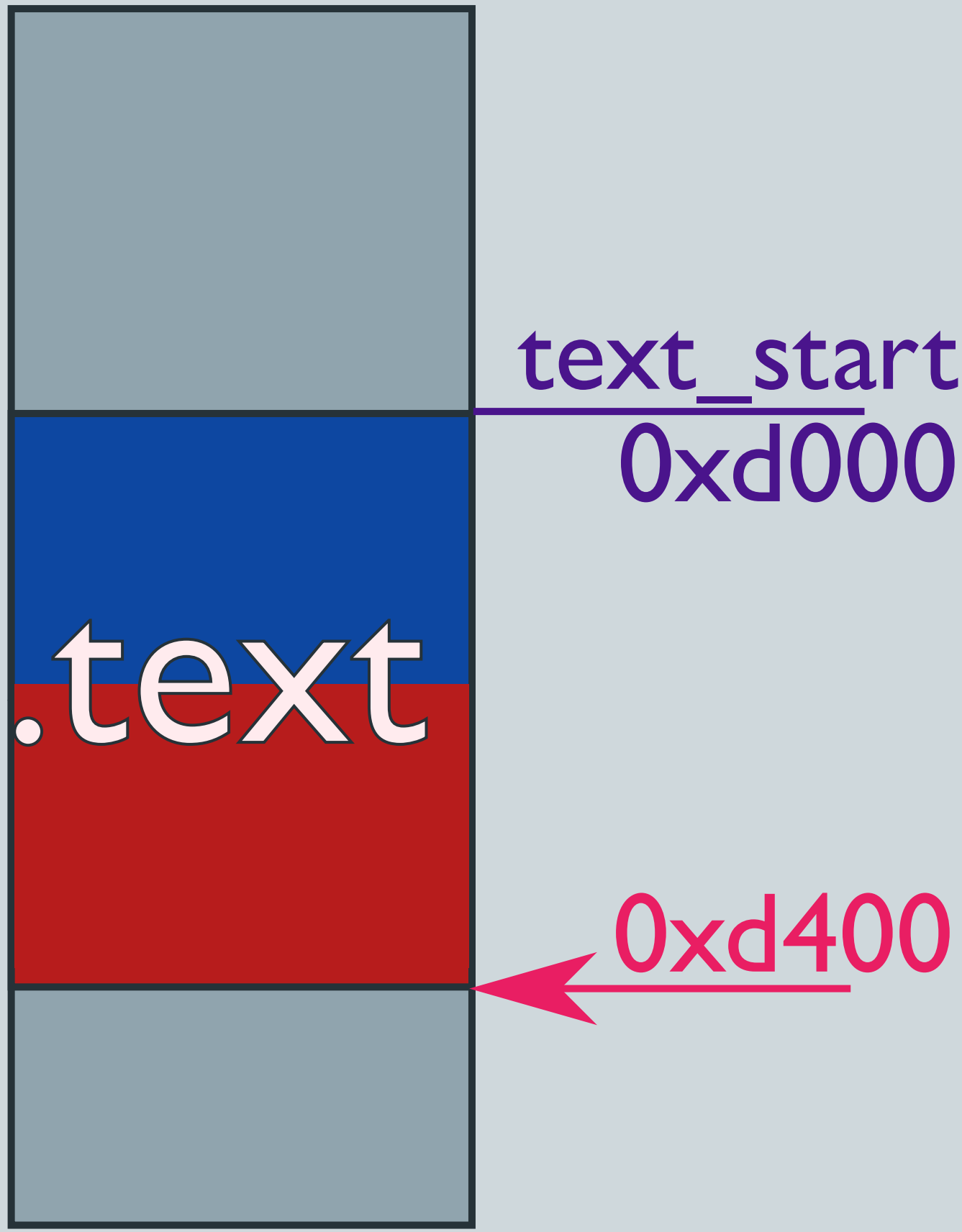
```
.text : {
    text_start = .
    *(.text*)
    text_end = .
}

text_size =
  SIZEOF(.text)
```



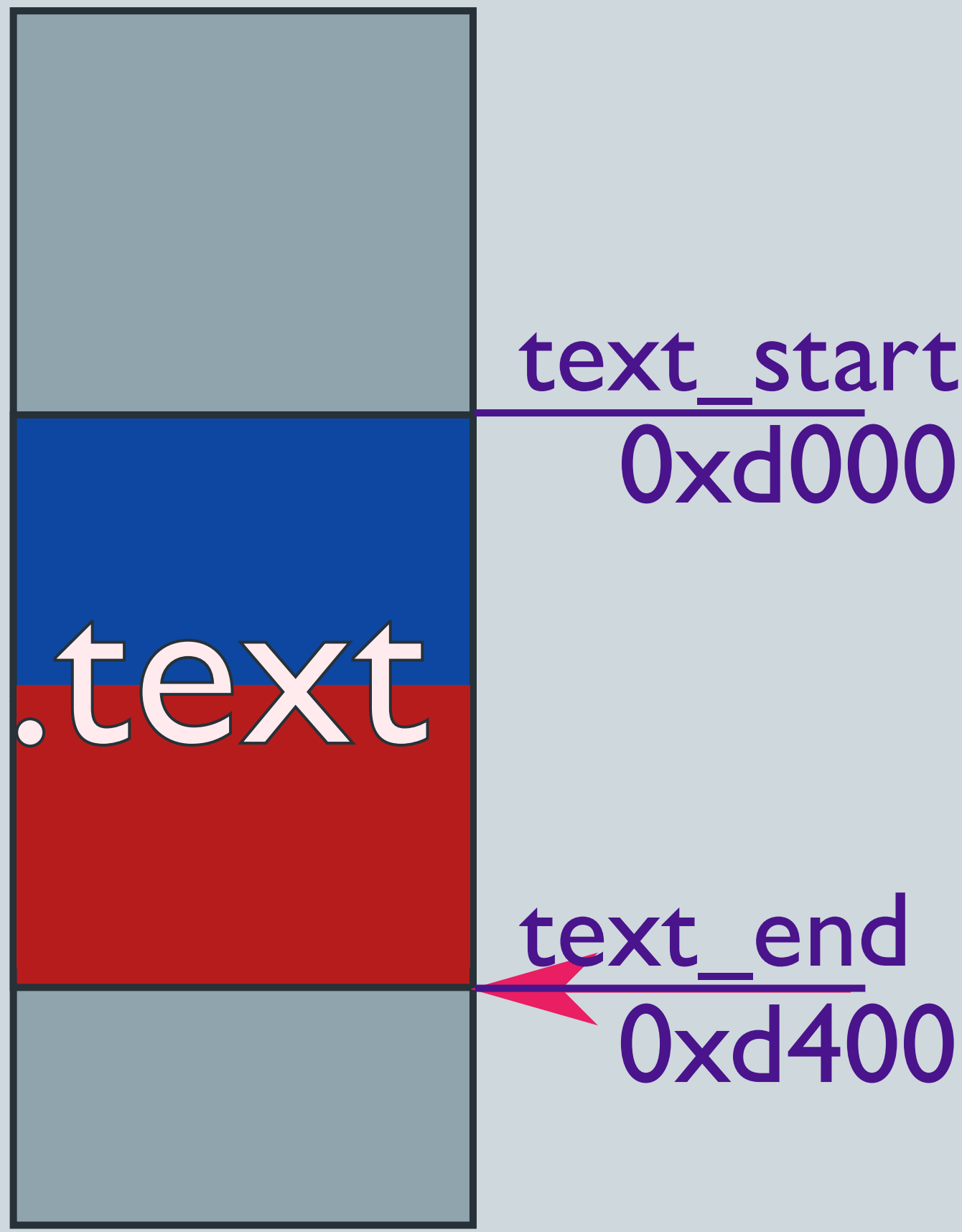
```
.text : {
    text_start = .
    *(.text*)
    text_end = .
}

text_size =
  SIZEOF(.text)
```



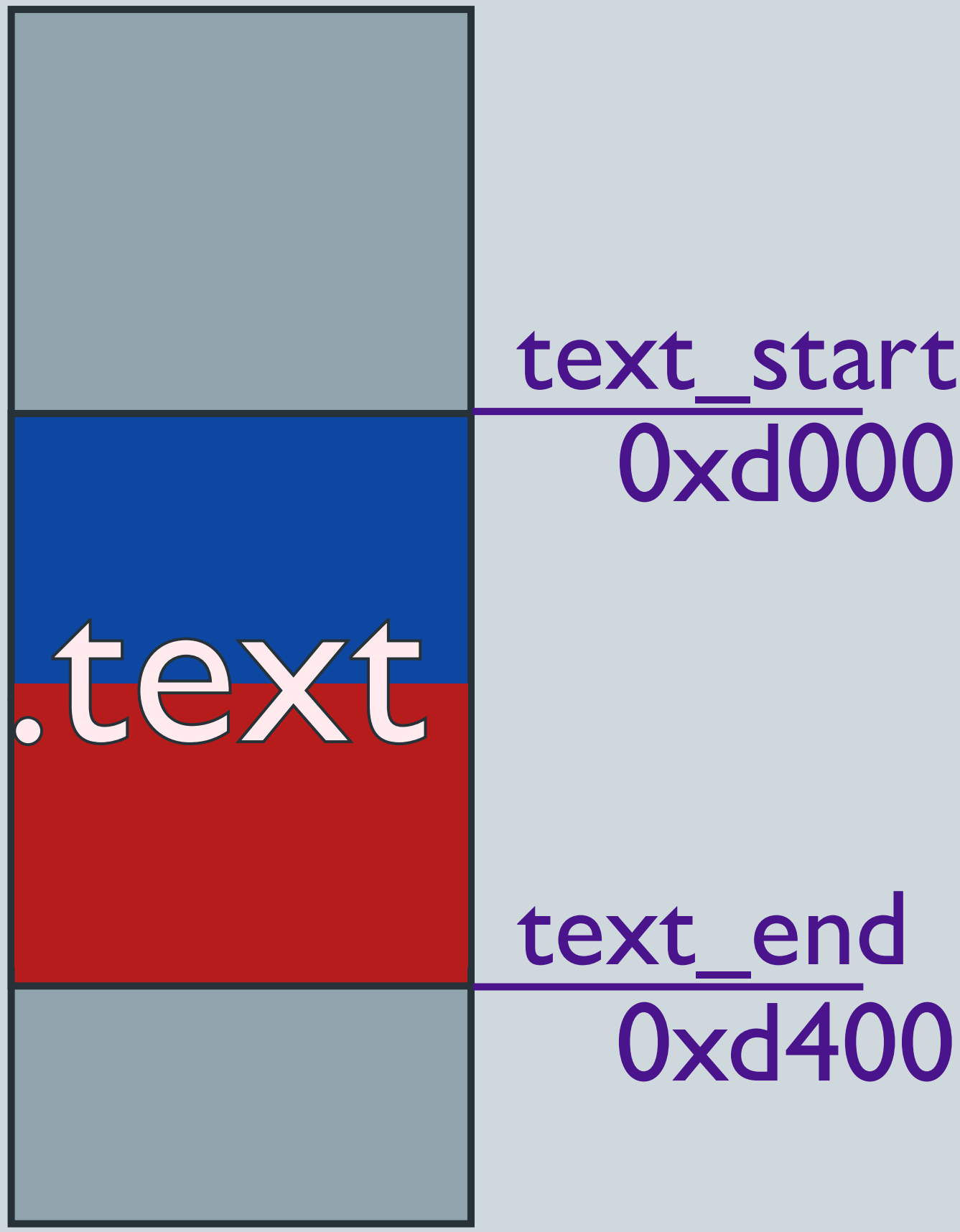
```
.text : {
    text_start = .
    *(.text*)
    text_end = .
}

text_size =
  SIZEOF(.text)
```

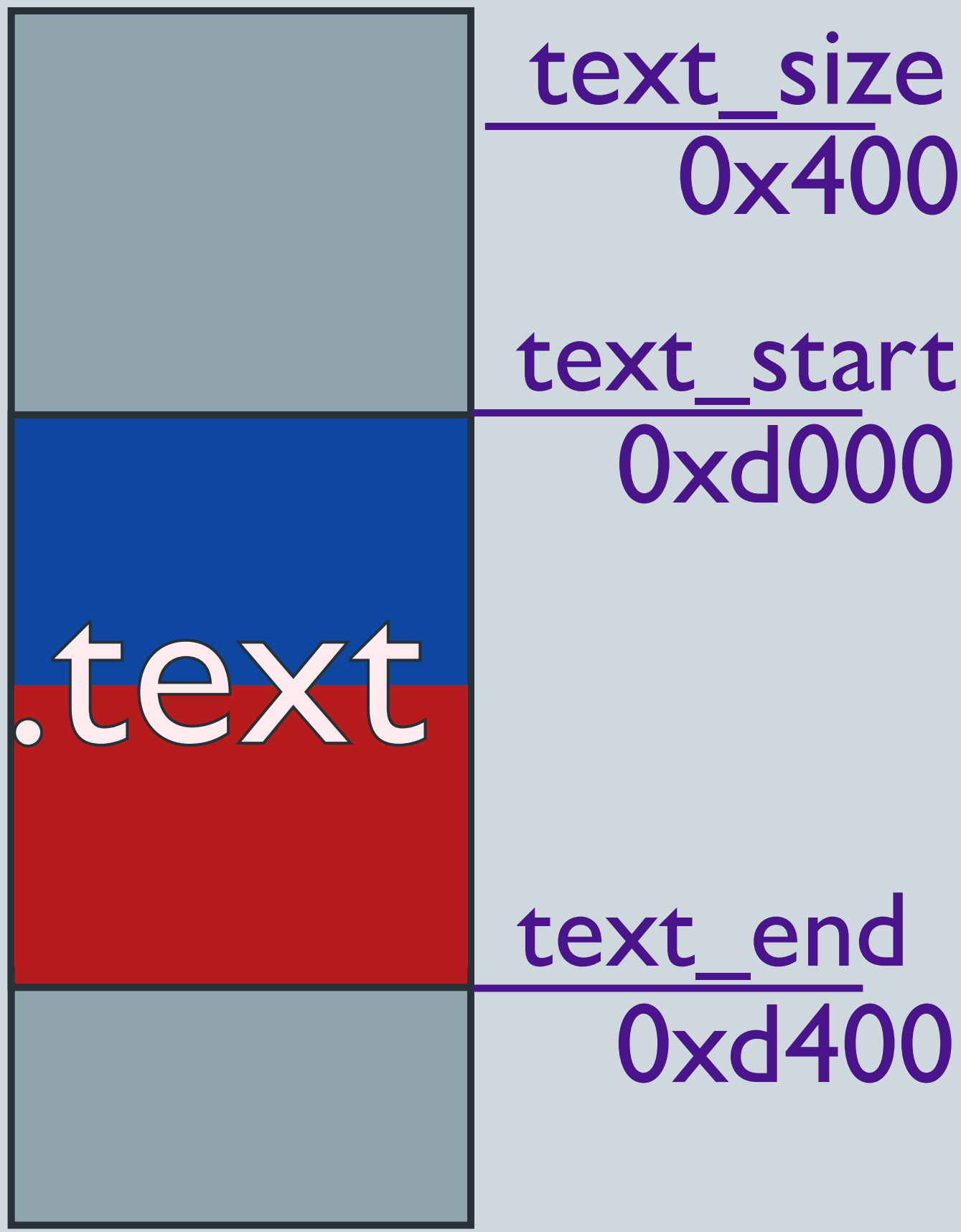


```
.text : {
    text_start = .
    *(.text*)
    text_end = .
}
```

text_size =
SIZEOF(.text)



```
.text : {
    text_start = .
    *(.text*)
    text_end = .
}
```

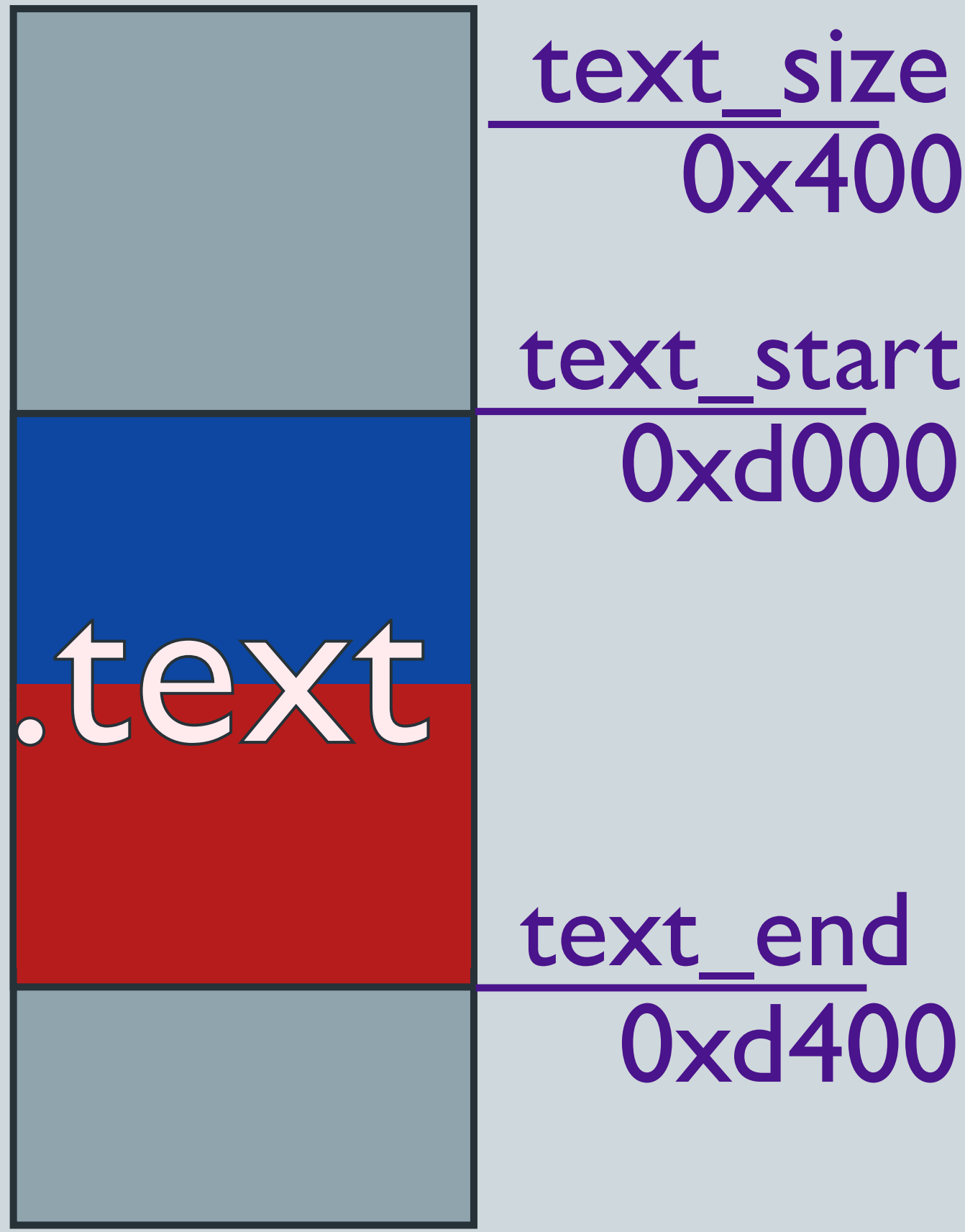


```
extern char
  text_size[],
  text_start[],
  text_end[];

int main() {

  assert(&text_size
    == (char*)0x400);

  assert(&text_start
    == (char*)0xd000);
}
```

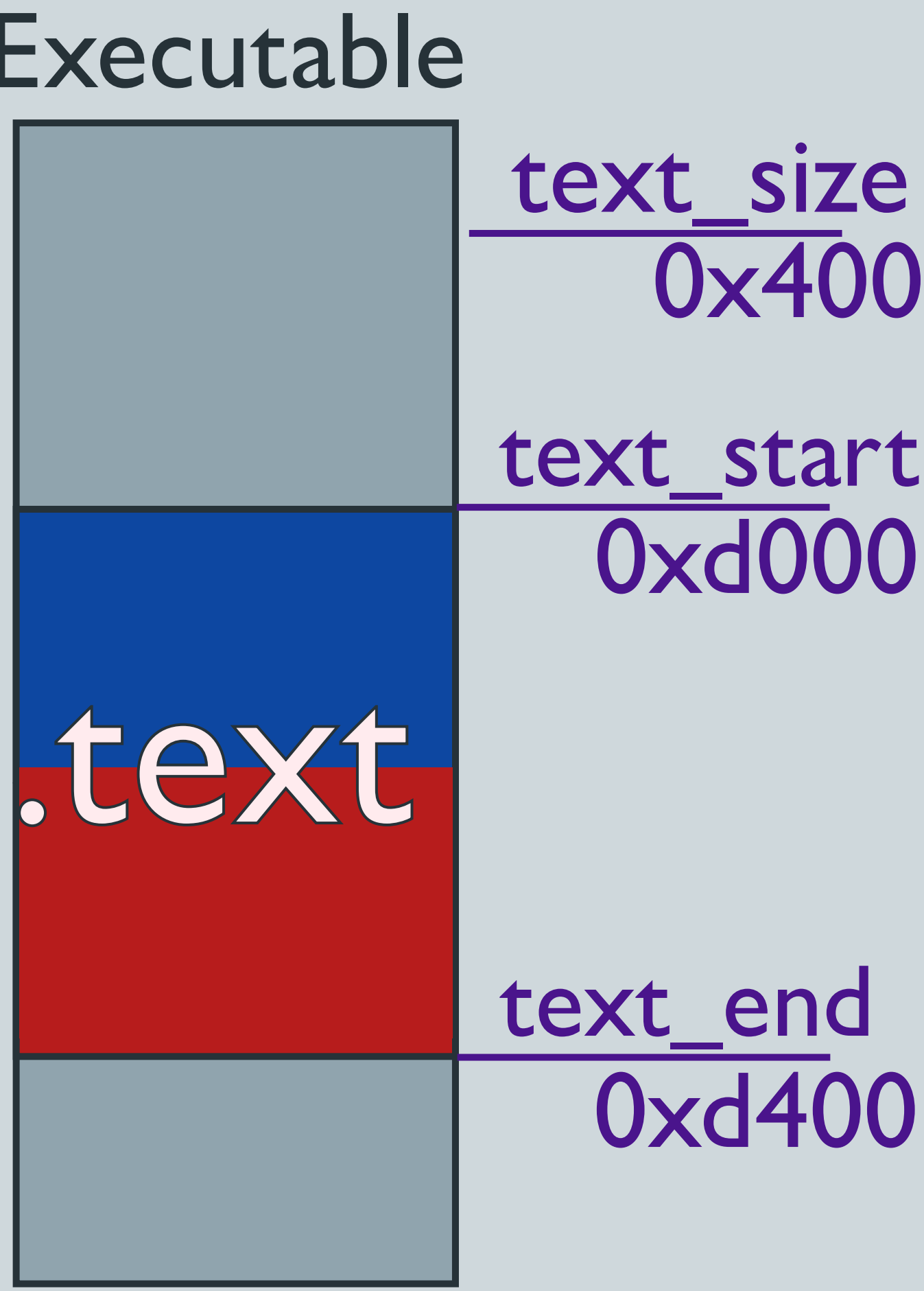


```
extern char
  text_size[],
  text_start[],
  text_end[];

int main() {

 memcpy(
    buf,
    (void *)&text_start,
    (size_t)&text_size);

}
```



```
extern char
  text_size[],
  text_start[],
  text_end[];

int main() {

 memcpy(
    buf,
    (void *)&text_start,
    (size_t)&text_size);

}
```

```
extern char
  text_size[],
  text_start[],
  text_end[];

int main() {

 memcpy(
    buf,
    (void *)&text_start,
    (size_t)&text_size);

}
```

```
extern char
  text_size[],
  text_start[],
  text_end[];

int main() {

 memcpy(
    buf,
    (void *)&text_start,
    (size_t)&text_size);

}
```

Symbol Table

foo	12
bar	0
...	...
text_start	???
text_size	???

```
extern char
  text_size[],
  text_start[],
  text_end[];

int main() {

 memcpy(
    buf,
    (void *)&text_start,
    (size_t)&text_size);

}
```

linker_syms.h

```
char text_start[];
&text_start
    = 0xd000;

char text_size[];
&text_size
    = 0x400;
```

```
extern char
  text_size[],
  text_start[],
  text_end[];

int main() {

 memcpy(
    buf,
    (void *)&text_start,
    (size_t)&text_size);

}
```

linker_syms.h

```
char text_start[];
&text_start
    = 0xd000;

char text_size[];
&text_size
    = 0x400;
```

```
extern char
  text_size[],
  text_start[],
  text_end[];

int main() {

 memcpy(
    buf,
    (void *)  text_start,
    (size_t)   text_size);

}
```

linker_syms.h

```
size_t text_start;
text_start
    = 0xd000;

size_t text_size;
text_size
    = 0x400;
```

```
extern char
  text_size[],
  text_start[],
  text_end[];

int main() {

 memcpy(
    buf,
    (void *)  text_start,
    (size_t)  text_size);

}
```

linker_syms.h

```
size_t text_start;
text_start
    = 0xd000;

size_t text_size;
text_size
    = 0x400;
```

.ld

exe

Symbol Table	
foo	12
bar	0
...	...
text_start	
text_size	

The Missing Link: Explaining ELF Static Linking, Semantically

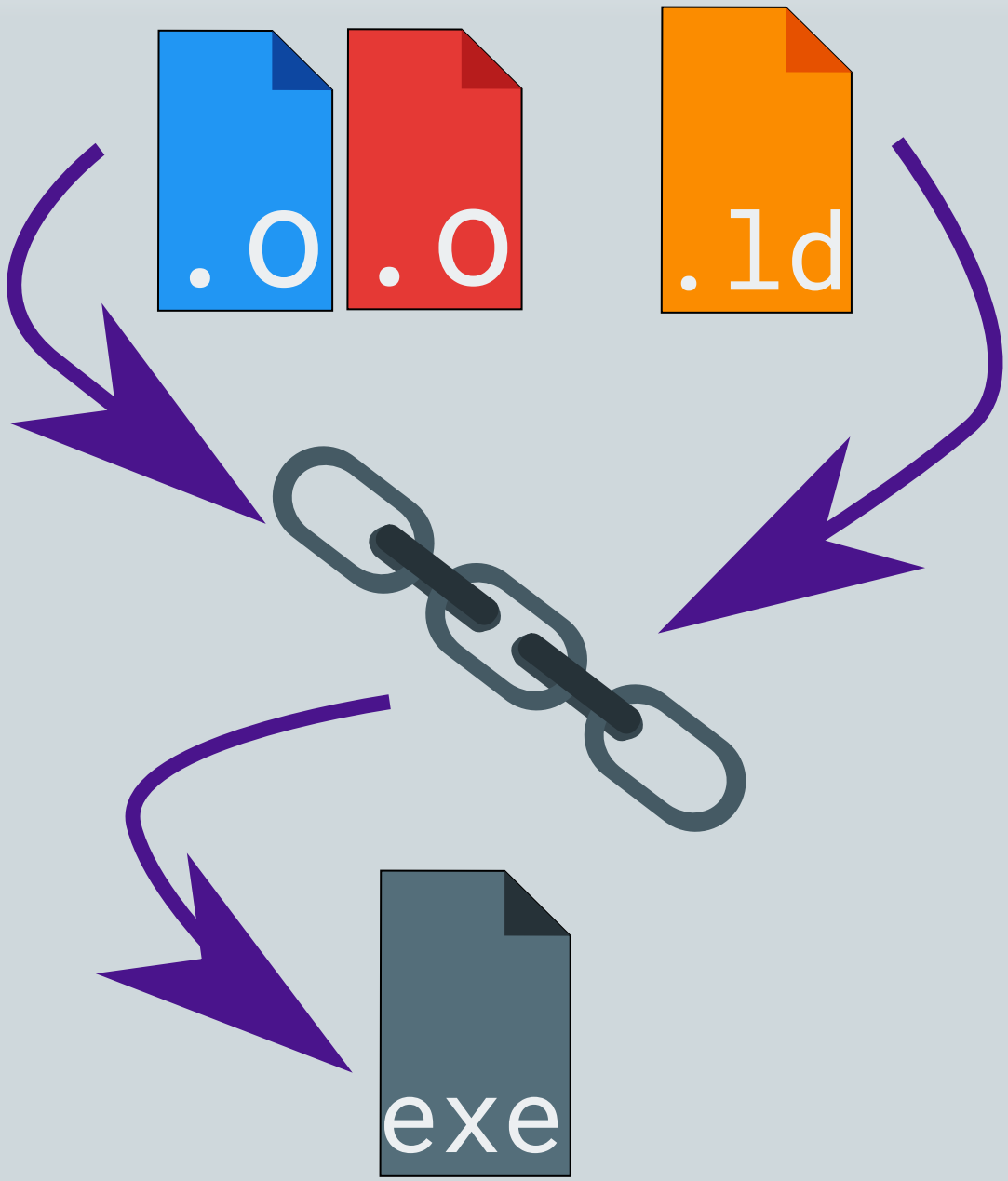
OOPSLA'16

Stephen Kell Dominic P. Mulligan Peter Sewell
Computer Laboratory, University of Cambridge, United Kingdom
firstname.lastname@cl.cam.ac.uk

The Missing Link: Explaining ELF Static Linking, Semantically

OOPSLA'16

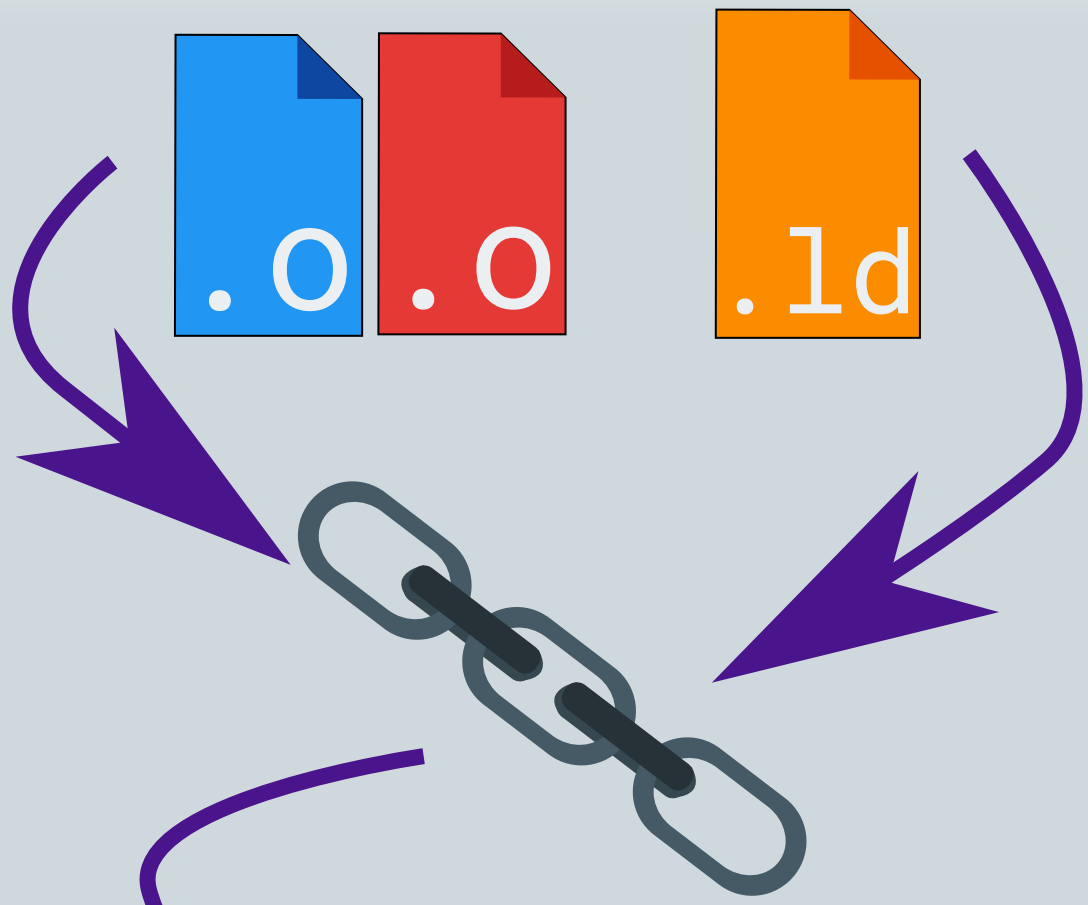
Stephen Kell Dominic P. Mulligan Peter Sewell
Computer Laboratory, University of Cambridge, United Kingdom
firstname.lastname@cl.cam.ac.uk

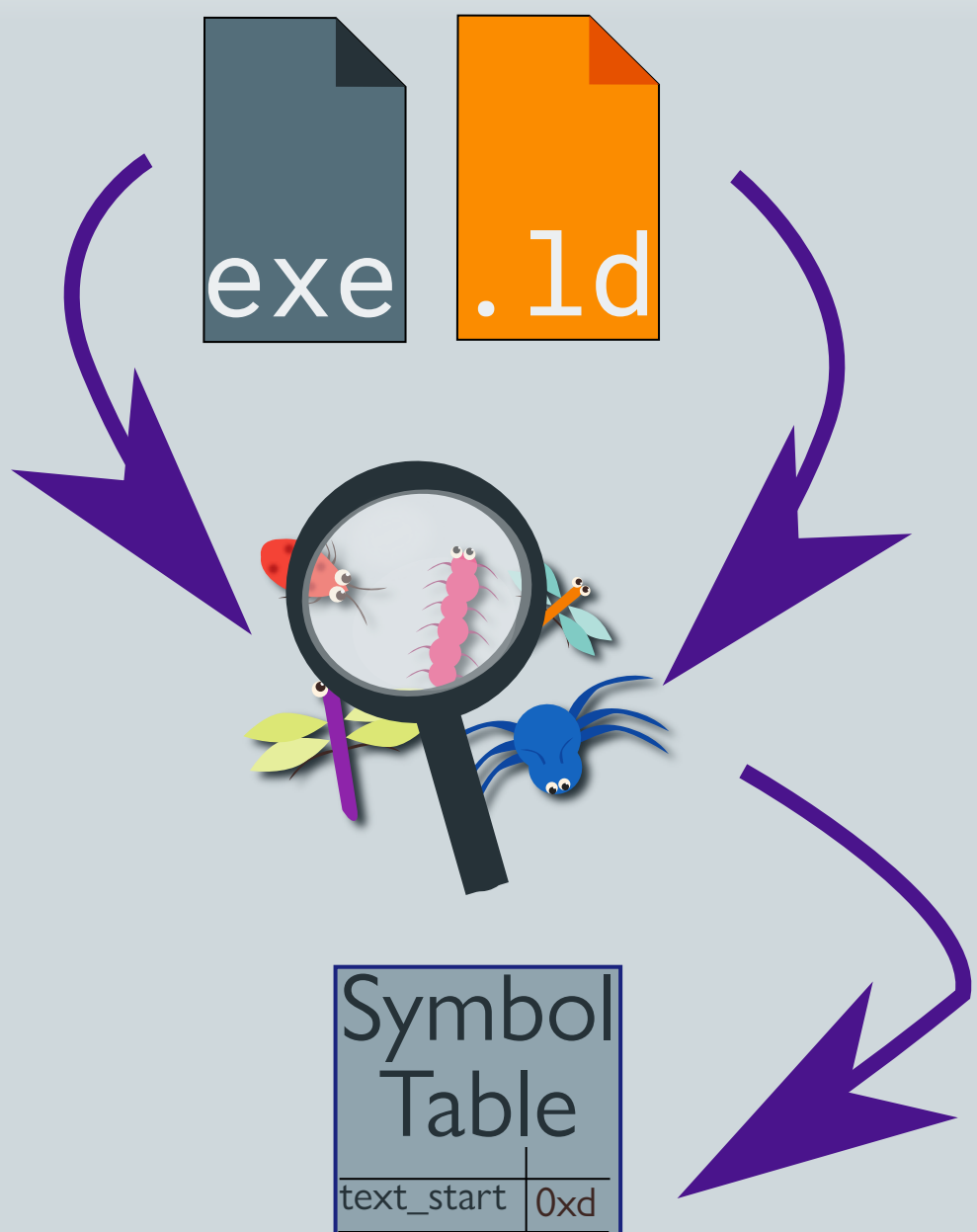


The Missing Link: Explaining ELF Static Linking, Semantically

OOPSLA'16

Stephen Kell Dominic P. Mulligan Peter Sewell
Computer Laboratory, University of Cambridge, United Kingdom
firstname.lastname@cl.cam.ac.uk





Four Functions

Four Functions

$addr_S \quad addr_E$

$: ident \rightharpoonup \mathbb{N}$

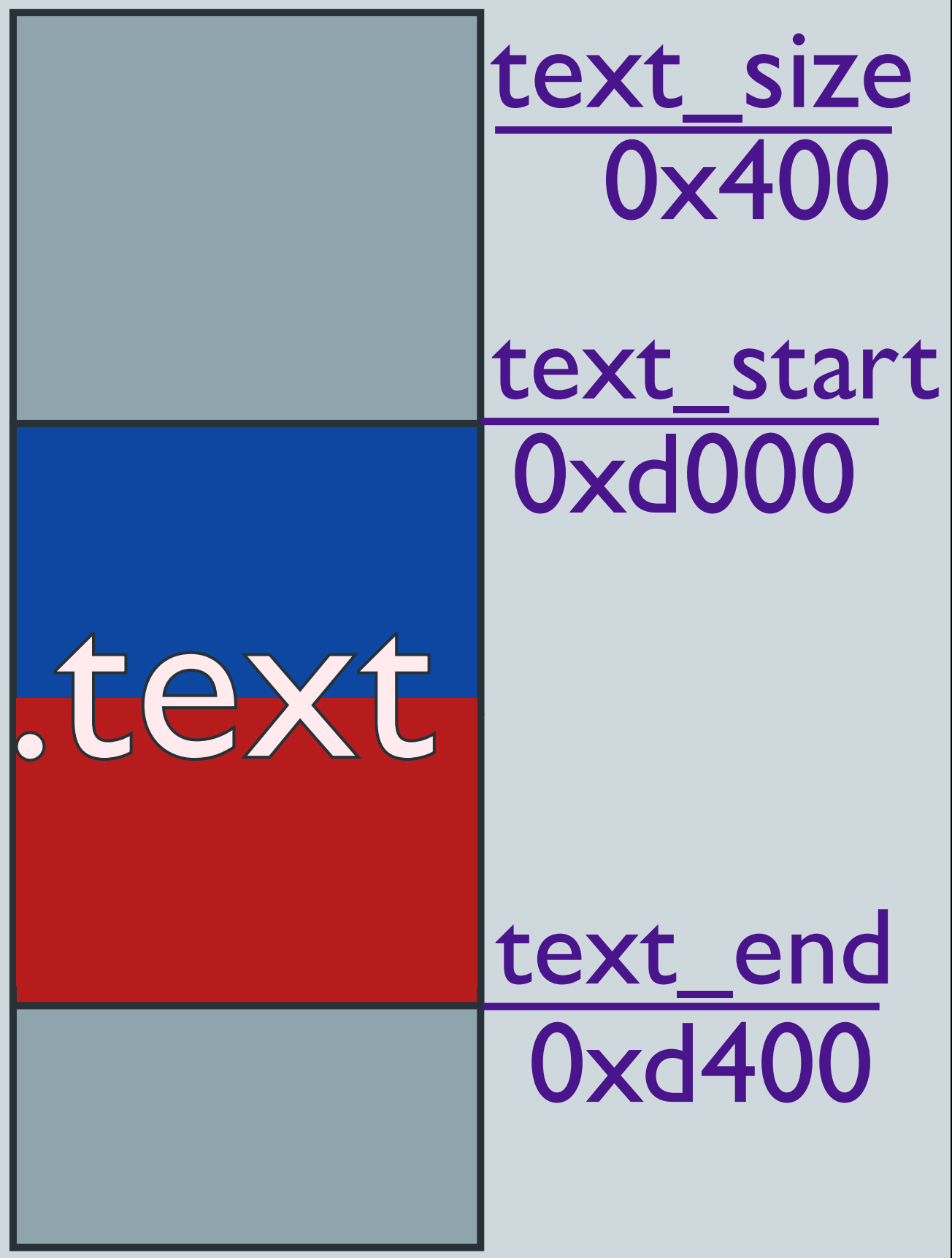
$symb_S \quad symb_E$

$: ident \rightharpoonup ident$

Four Functions

$addr_S \quad addr_E$
$: ident \rightarrow \mathbb{N}$

$symb_S \quad symb_E$
$: ident \rightarrow ident$



Four Functions

$addr_S \quad addr_E$

$: ident \rightharpoonup \mathbb{N}$

$symb_S \quad symb_E$

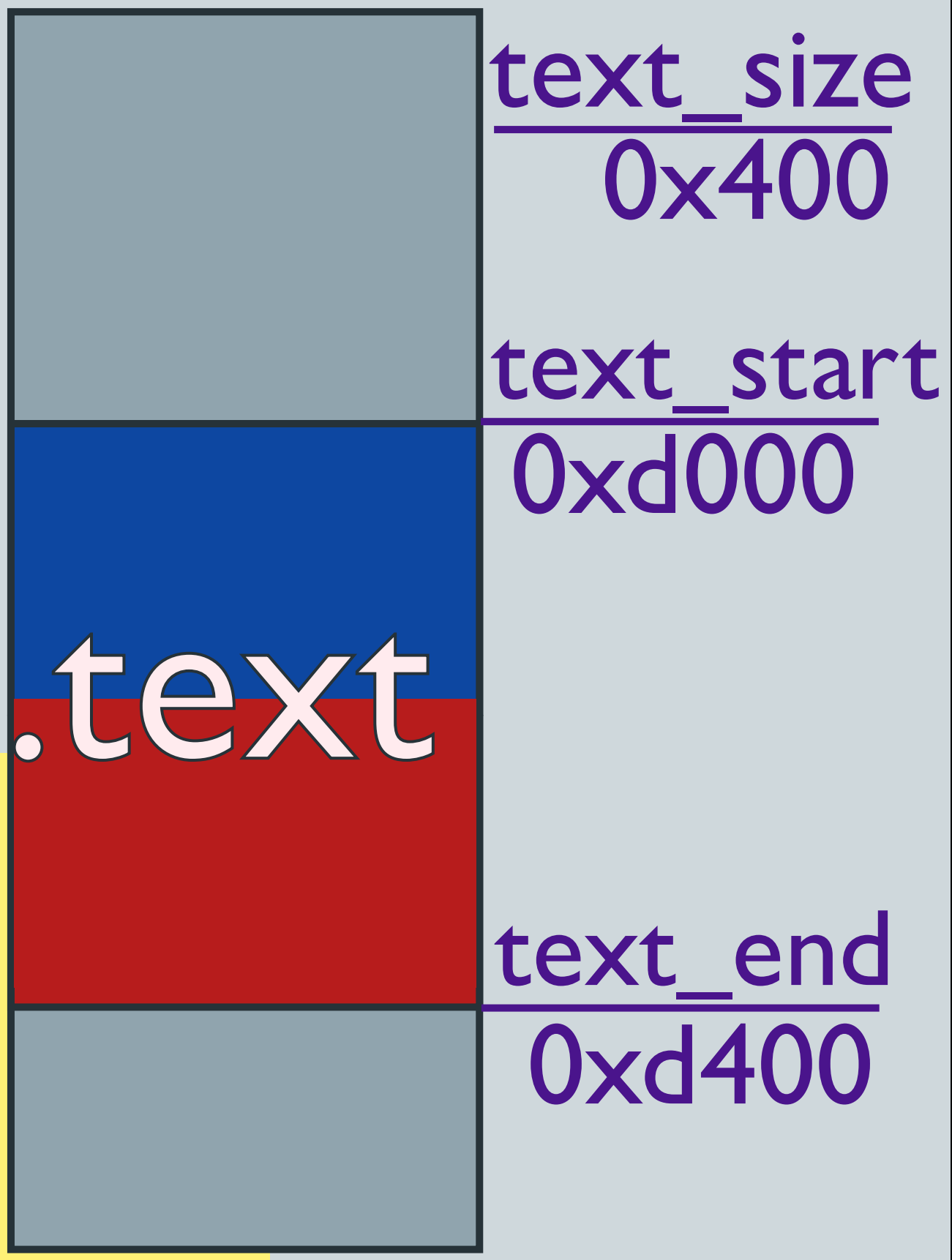
$: ident \rightharpoonup ident$

$symb_S =$

$.text \mapsto text\_start$

$symb_E =$

$.text \mapsto text\_end$



Four Functions

$addr_S \quad addr_E$

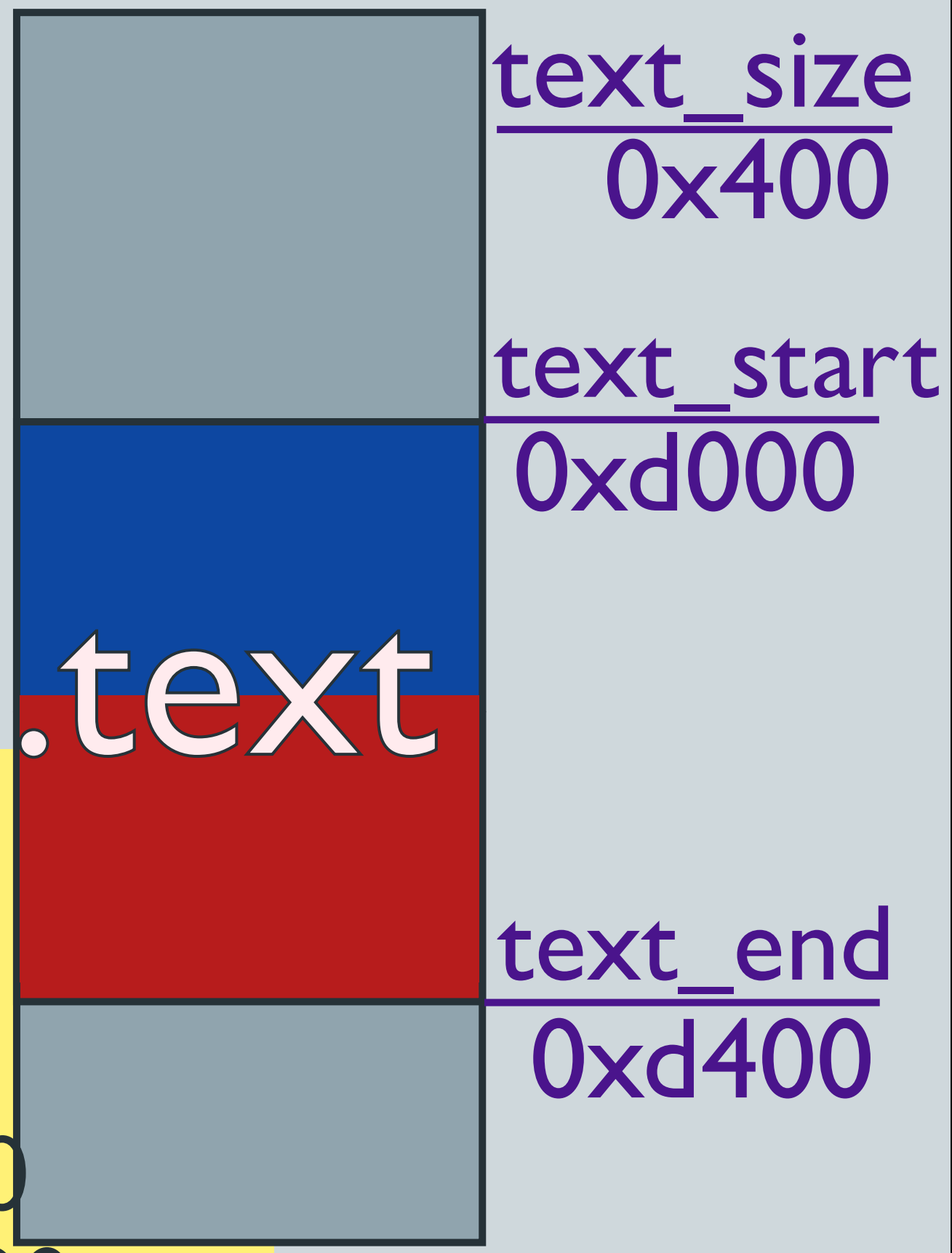
$: ident \rightarrow \mathbb{N}$

$symb_S \quad symb_E$

$: ident \rightarrow ident$

$addr_S =$

.text ↦ 0xd000
text_size ↦ 0x400
text_end ↦ 0xd400
text_start ↦ 0xd000



Four Functions

$addr_S \quad addr_E$

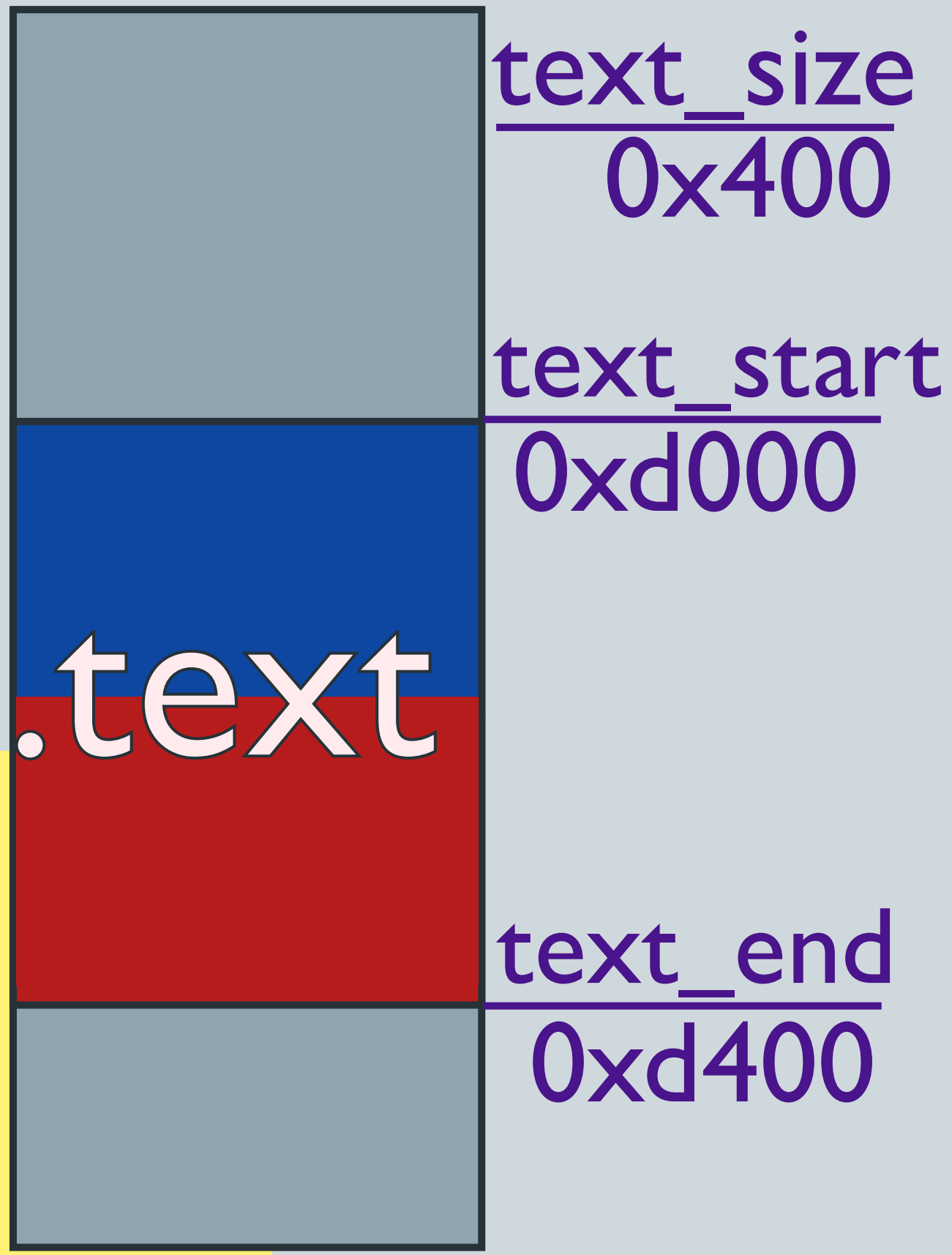
$: ident \rightharpoonup \mathbb{N}$

$symb_S \quad symb_E$

$: ident \rightharpoonup ident$

$addr_E =$

$.text \mapsto 0xd400$



Grammar

Grammar

MEMORY {

    ———————— Memory
    ———————— Directives
    ————————

}

Grammar

MEMORY {

    ——————— Memory
    ——————— Directives

}

SECTIONS {

    ——————— Section
    ——————— Directives

}

Grammar

MEMORY {

    ——————— Memory
    ——————— Directives

}

SECTIONS {

    ——————— Section
    ——————— Directives

}

——————— Global
——————— Assignments

Grammar

```
MEMORY {
        ____________
        ____________
        ____________
}

SECTIONS {
        ____________
        ____________
        ____________
}

____________
____________
____________
```

```
.text : {
  text_start = .;
  *(.text*)
  text_end = .;
}
```

Grammar

```
MEMORY {
        ____________________
        ____________________
        ____________________
}

SECTIONS {
        ____________________
        ____________________
        ____________________
}

____________________
____________________
____________________
```

```
text_size =
  SIZEOF(.text);
```

Grammar

MEMORY {

}

SECTIONS {

}

rom :
ORIGIN =
0x2000,
LENTGH =
0x400;

Expressions

$$\frac{number(\mathrm{D}) = n}{\langle \text{Decimal D} \rangle \Downarrow n}$$

$$\frac{number(H)_{16} = n_{10}}{\langle \text{Hex H} \rangle \Downarrow n}$$

$$\frac{number(\mathrm{D}) \times 1024 = n}{\langle \text{Decimal D} \rangle \ \mathrm{k} \ \Downarrow n}$$

$$\frac{number(\mathrm{D}) \times 1024^2 = n}{\langle \text{Decimal D} \rangle \ \mathrm{M} \ \Downarrow n}$$

$$\frac{\mathrm{E}_1 \Downarrow n_1 \qquad \mathrm{E}_2 \Downarrow n_2 \qquad operator(\mathrm{O}) = \odot \qquad n_1 \odot n_2 = n}{\langle \text{ConstExpr } \mathrm{E}_1 \rangle \langle \text{Operator O} \rangle \langle \text{ConstExpr } \mathrm{E}_2 \rangle \Downarrow n}$$

Expressions

$$\frac{\mathrm{add}_s(S) = a_1 \qquad \mathrm{add}_E(S) = a_2 \qquad a_2 - a_1 = n}{\texttt{SIZEOF}\,(\ \langle \text{SectionName } S \rangle\ )\ \Downarrow n}$$

Expressions

$$\frac{\mathsf{add}_s(\mathsf{M}) = a_1 \qquad \mathsf{add}_e(\mathsf{M}) = a_2 \qquad a_2 - a_1 = n}{\mathsf{LENGTH}(\ \langle \mathsf{MemoryName\ M} \rangle\ ) \Downarrow n}$$

Expressions

$$\mathrm{add}_s(\mathrm{M}) = n$$

$$\texttt{ORIGIN(}\ \langle \mathrm{MemoryName\ M} \rangle\ \texttt{)} \Downarrow n$$

Interpretation

MEMORY {

_____

_____

_____

}

SECTIONS {

_____

_____

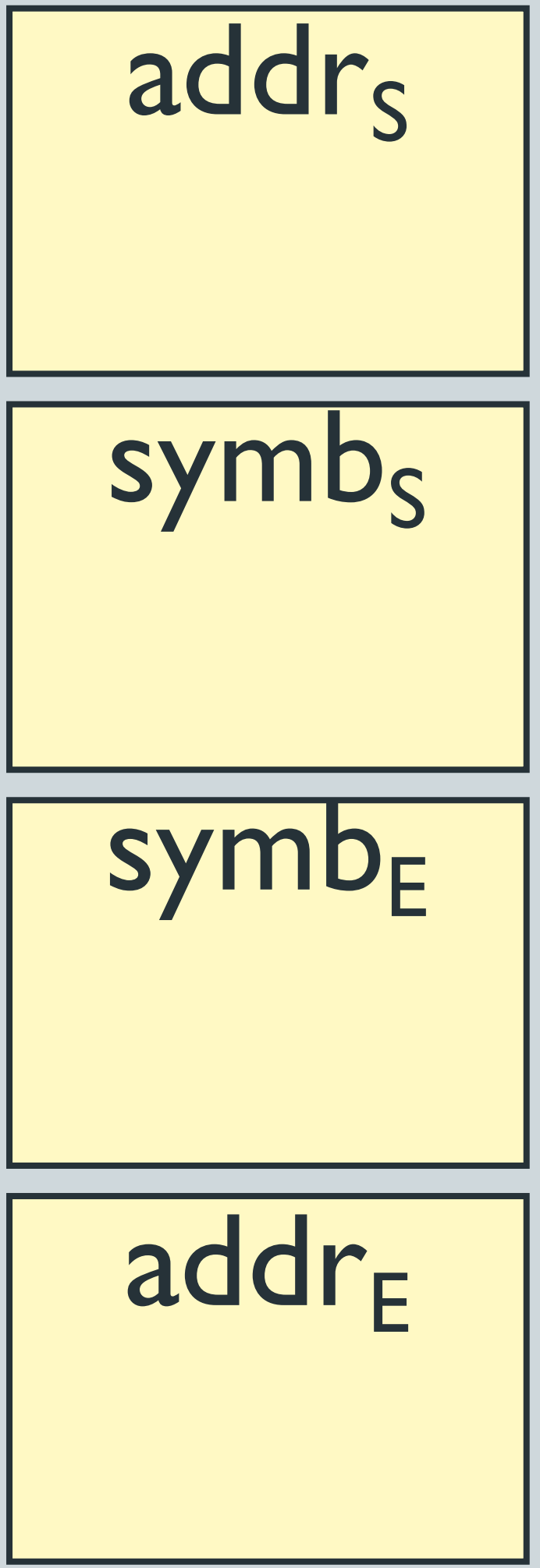
_____

}

_____

_____

_____



Interpretation

MEMORY {

}

SECTIONS {

}

$addr_S$

$symb_S$

$symb_E$

$addr_E$

Interpretation

MEMORY {

}

SECTIONS {

}

$addr_S$

$symb_S$

$symb_E$

$addr_E$

Interpretation

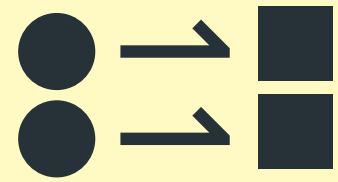
MEMORY {

}

SECTIONS {

}

$addr_S$



$symb_S$

$symb_E$

$addr_E$

Interpretation

MEMORY {

}

SECTIONS {

}

$addr_S$

$symb_S$

$symb_E$

$addr_E$

Interpretation

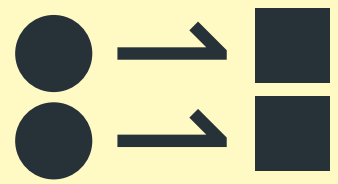
MEMORY {

}

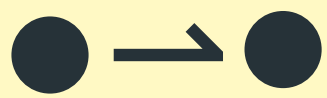
SECTIONS {

}

$addr_S$

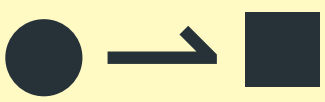


$symb_S$



$symb_E$

$addr_E$



Interpretation

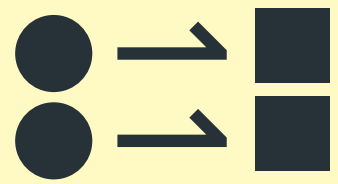
MEMORY {

}

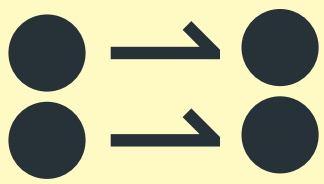
SECTIONS {

}

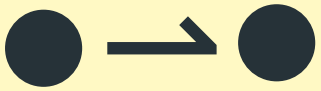
$addr_S$



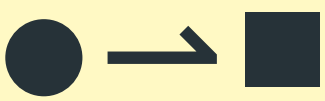
$symb_S$



$symb_E$



$addr_E$



Interpretation

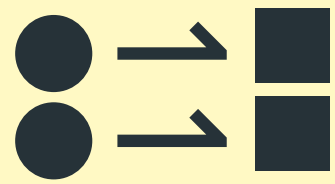
MEMORY {

}

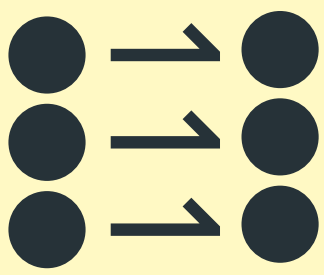
SECTIONS {

}

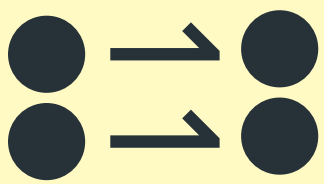
$addr_S$



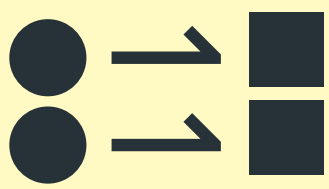
$symb_S$



$symb_E$



$addr_E$



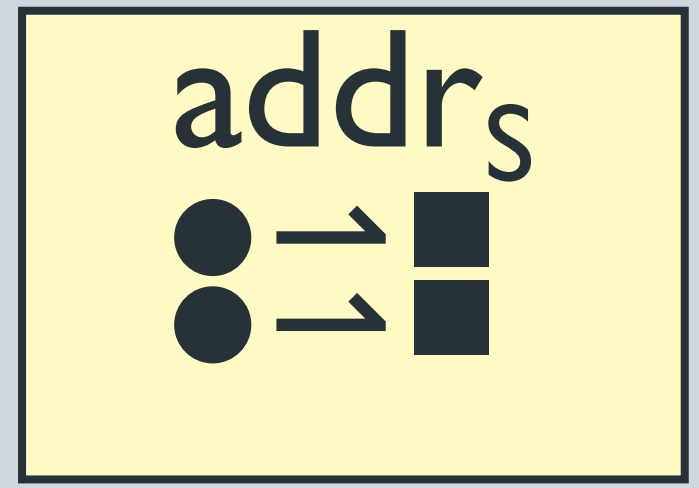
Interpretation

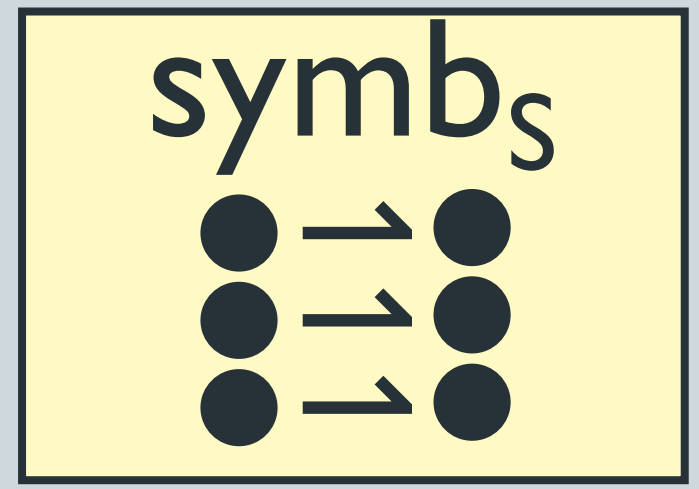
MEMORY {

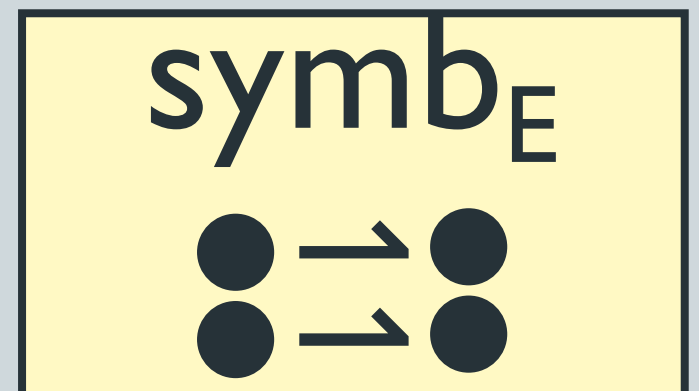
}

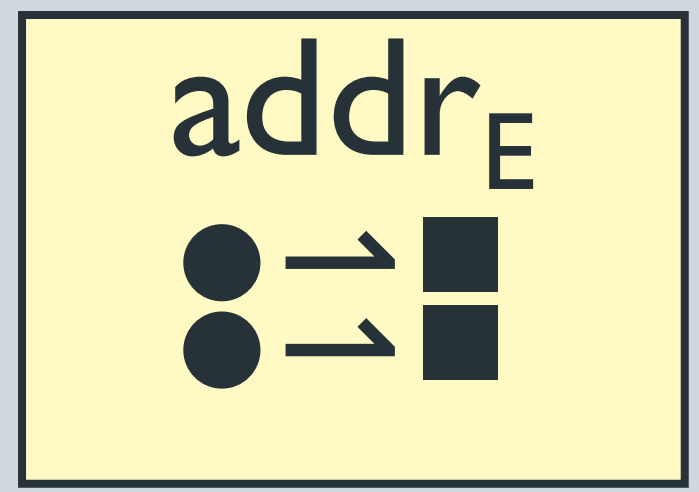
SECTIONS {

}









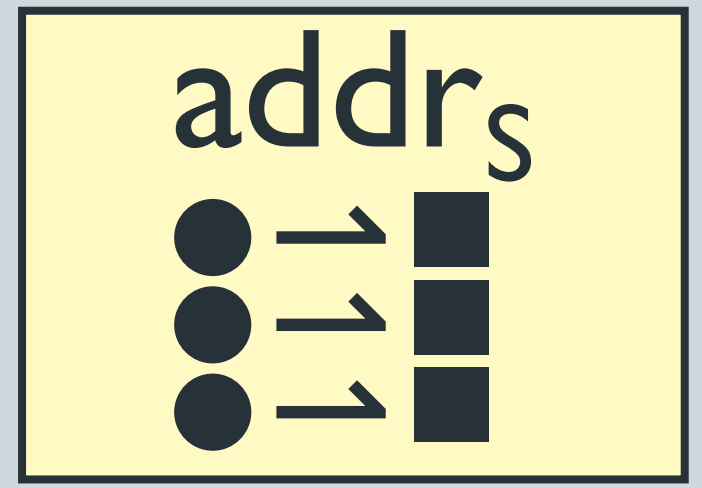
Interpretation

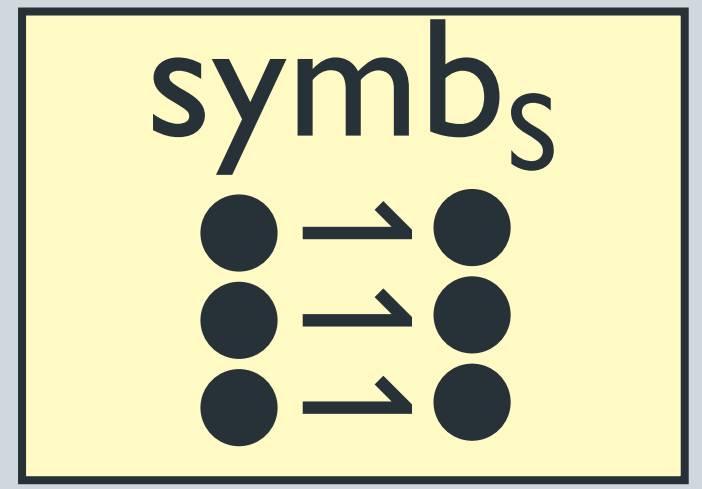
MEMORY {

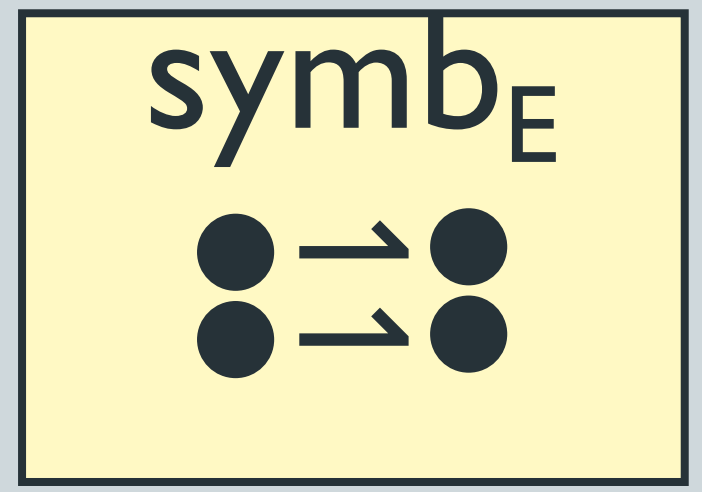
}

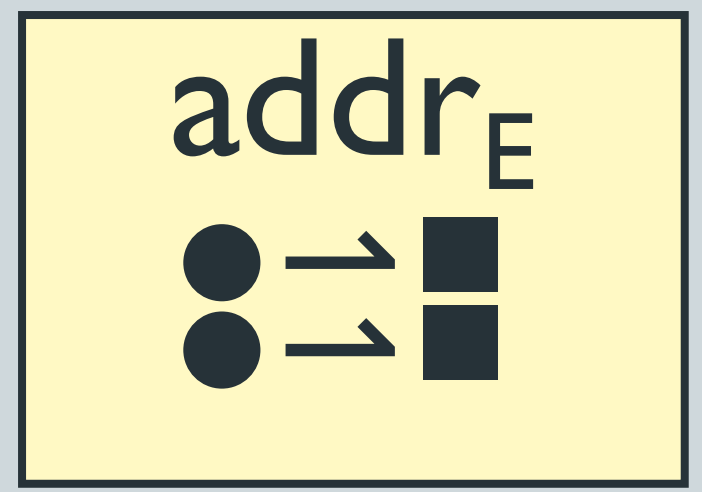
SECTIONS {

}









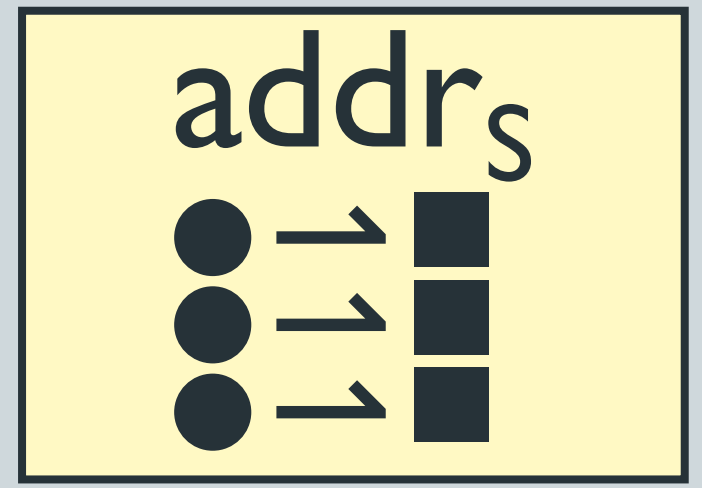
Interpretation

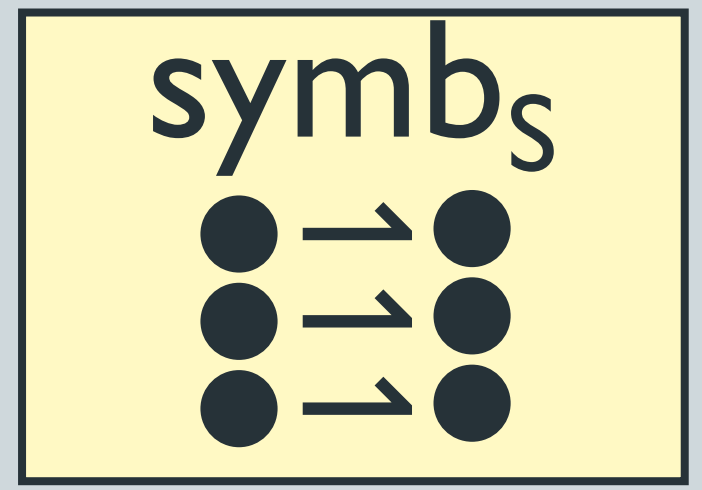
MEMORY {

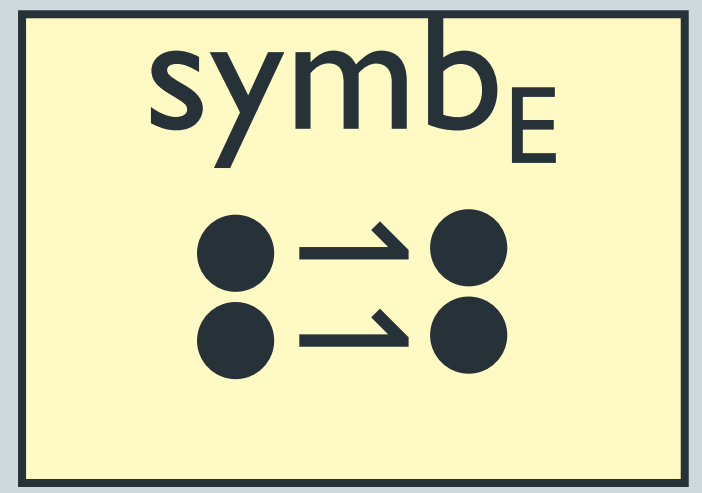
}

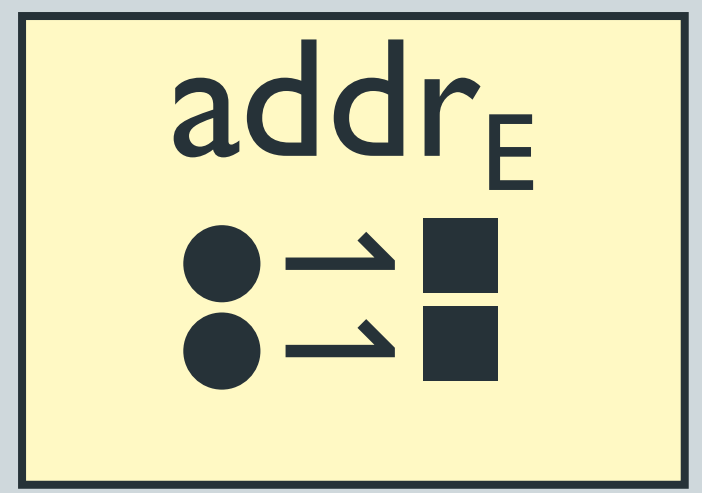
SECTIONS {

}







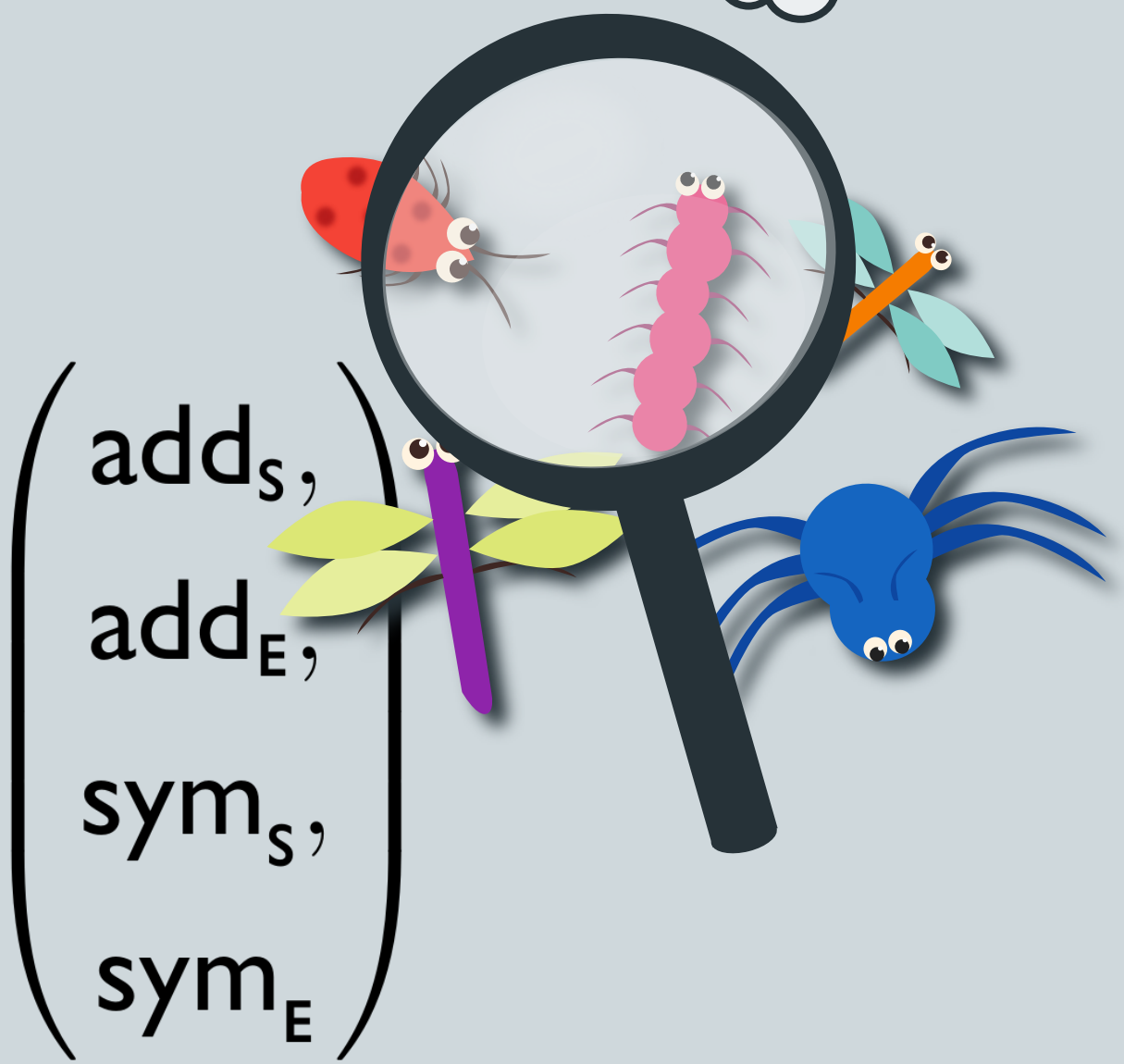


$$\left(\begin{array}{c} \left[\langle \text{MemoryDirective} \rangle\right], \\ \langle \text{Identifier} \rangle \rightharpoonup \mathbb{N}, \\ \langle \text{Identifier} \rangle \rightharpoonup \mathbb{N}, \\ \langle \text{Identifier} \rangle \rightharpoonup \langle \text{Identifier} \rangle, \\ \langle \text{Identifier} \rangle \rightharpoonup \langle \text{Identifier} \rangle \end{array}\right) \overset{\text{mem}}{\rightsquigarrow} \left(\begin{array}{c} \left[\langle \text{MemoryDirective} \rangle\right], \\ \langle \text{Identifier} \rangle \rightharpoonup \mathbb{N}, \\ \langle \text{Identifier} \rangle \rightharpoonup \mathbb{N}, \\ \langle \text{Identifier} \rangle \rightharpoonup \langle \text{Identifier} \rangle, \\ \langle \text{Identifier} \rangle \rightharpoonup \langle \text{Identifier} \rangle \end{array}\right)$$

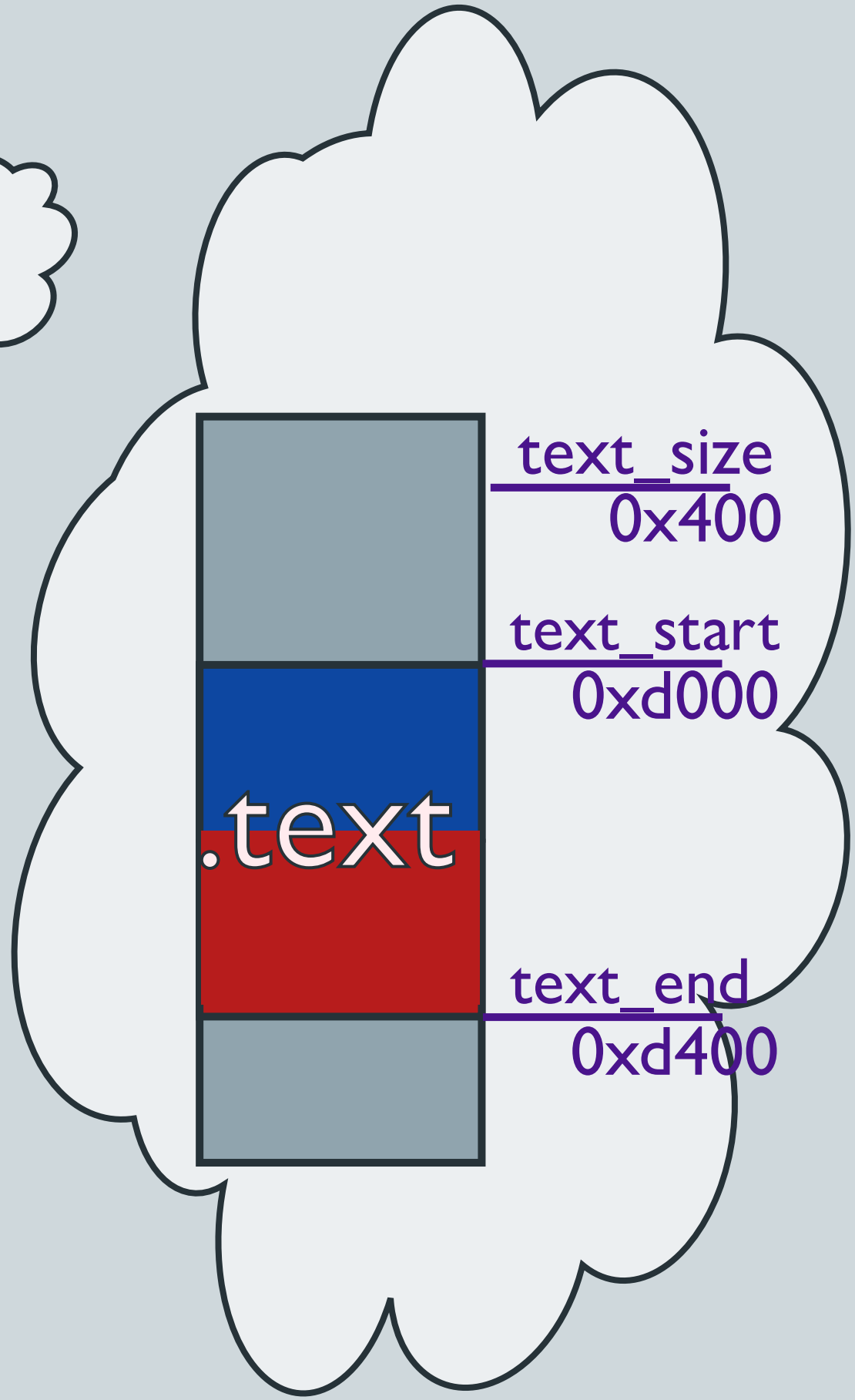
$$\left(\begin{array}{c} \left[\langle \text{SectionsDirective} \rangle\right], \\ \langle \text{Identifier} \rangle \rightharpoonup \mathbb{N}, \\ \langle \text{Identifier} \rangle \rightharpoonup \mathbb{N}, \\ \langle \text{Identifier} \rangle \rightharpoonup \langle \text{Identifier} \rangle, \\ \langle \text{Identifier} \rangle \rightharpoonup \langle \text{Identifier} \rangle \end{array}\right) \overset{\text{sec}}{\rightsquigarrow} \left(\begin{array}{c} \left[\langle \text{SectionsDirective} \rangle\right], \\ \langle \text{Identifier} \rangle \rightharpoonup \mathbb{N}, \\ \langle \text{Identifier} \rangle \rightharpoonup \mathbb{N}, \\ \langle \text{Identifier} \rangle \rightharpoonup \langle \text{Identifier} \rangle, \\ \langle \text{Identifier} \rangle \rightharpoonup \langle \text{Identifier} \rangle \end{array}\right)$$

$$\begin{pmatrix} \mathsf{M}, \\ \varnothing, \\ \varnothing, \\ \varnothing, \\ \varnothing \end{pmatrix} \overset{\mathsf{mem}*}{\rightsquigarrow} \begin{pmatrix} \varepsilon, \\ \mathsf{add}_{\mathsf{s}}^{\mathsf{M}}, \\ \mathsf{add}_{\mathsf{E}}^{\mathsf{M}}, \\ \mathsf{sym}_{\mathsf{s}}^{\mathsf{M}}, \\ \mathsf{sym}_{\mathsf{E}}^{\mathsf{M}} \end{pmatrix} \quad \begin{pmatrix} \mathsf{S}, \\ \mathsf{add}_{\mathsf{s}}^{\mathsf{M}}, \\ \mathsf{add}_{\mathsf{E}}^{\mathsf{M}}, \\ \mathsf{sym}_{\mathsf{s}}^{\mathsf{M}}, \\ \mathsf{sym}_{\mathsf{E}}^{\mathsf{M}} \end{pmatrix} \overset{\mathsf{sec}*}{\rightsquigarrow} \begin{pmatrix} \varepsilon, \\ \mathsf{add}_{\mathsf{s}}^{\mathsf{S}}, \\ \mathsf{add}_{\mathsf{E}}^{\mathsf{S}}, \\ \mathsf{sym}_{\mathsf{s}}^{\mathsf{S}}, \\ \mathsf{sym}_{\mathsf{E}}^{\mathsf{S}} \end{pmatrix} \quad \begin{pmatrix} \mathsf{G}, \\ \mathsf{add}_{\mathsf{s}}^{\mathsf{S}}, \\ \mathsf{add}_{\mathsf{E}}^{\mathsf{S}}, \\ \mathsf{sym}_{\mathsf{s}}^{\mathsf{S}}, \\ \mathsf{sym}_{\mathsf{E}}^{\mathsf{S}} \end{pmatrix} \overset{\mathsf{glo}*}{\rightsquigarrow} \begin{pmatrix} \varepsilon, \\ \mathsf{add}_{\mathsf{s}}, \\ \mathsf{add}_{\mathsf{E}}, \\ \mathsf{sym}_{\mathsf{s}}, \\ \mathsf{sym}_{\mathsf{E}} \end{pmatrix}$$

$$\left(\begin{bmatrix} \texttt{MEMORY } \{ \\ \langle \mathsf{MemoryDirectives\ M} \rangle \\ \} \\ \texttt{SECTIONS } \{ \\ \langle \mathsf{SectionsDirectives\ S} \rangle \\ \} \\ \langle \mathsf{GlobalAssignments\ G} \rangle \end{bmatrix} \right) \rightsquigarrow \begin{pmatrix} \mathsf{add}_{\mathsf{s}}, \\ \mathsf{add}_{\mathsf{E}}, \\ \mathsf{sym}_{\mathsf{s}}, \\ \mathsf{sym}_{\mathsf{E}} \end{pmatrix}$$



$$\begin{pmatrix} add_S, \\ add_E, \\ sym_S, \\ sym_E \end{pmatrix}$$



Model Checking Boot Code from AWS Data Centers

CAV'18

Byron Cook[1,2], Kareem Khazem[1,2], Daniel Kroening[3], Serdar Tasiran[1], Michael Tautschnig[1,4], and Mark R. Tuttle[1]

Implemented in CBMC

Semantics of Linker Scripts
(a static analysis perspective)

KAREEM KHAZEM

- Static analysers need to understand program behaviour

- Linker scripts introduce information unavailable to static analyser

- Solution: parse linker scripts, integrate into static analysis