Byte-oriented Programming

Robert Smallshire
COFOUNDER – SIXTY NORTH
@robsmallshire rob@sixty-north.com

Summary

Summary

Summary

Bitwise operators

Summary

Bitwise operators

Binary integer representation

Summary

Bitwise operators

Binary integer representation

The `bytes` type in detail

Summary

Bitwise operators

Binary integer representation

The `bytes` type in detail

The `bytearray` type

Summary

Bitwise operators

Binary integer representation

The `bytes` type in detail

The `bytearray` type

Packing and unpacking binary data

Summary

Bitwise operators

Binary integer representation

The `bytes` type in detail

The `bytearray` type

Packing and unpacking binary data

Sharing data with `memoryview`

Bitwise Operations on Integers

Bitwise Operators

&	|	^
And	Or	Exclusive-Or (XOR)
~	<<	>>
Not	Left shift	Right shift

Bitwise Operators

&	|	^
And	Or	Exclusive-Or (XOR)
~	<<	>>
Not	Left shift	Right shift

Bitwise Operators

&	|	^
And	Or	Exclusive-Or (XOR)
~	<<	>>
Not	Left shift	Right shift

Bitwise Operators

&	|	^
And	Or	Exclusive-Or (XOR)
~	<<	>>
Not	Left shift	Right shift

Bitwise Operators

&	|	^
And	Or	Exclusive-Or (XOR)
~	<<	>>
Not	Left shift	Right shift

Bitwise Operators

&	|	^
And	Or	Exclusive-Or (XOR)
~	<<	>>
Not	Left shift	Right shift

Bitwise Operators

& And	| Or	^ Exclusive-Or (XOR)
~ Not	<< Left shift	>> Right shift

1	1	1	0	0	1	0	0

XOR

0	0	1	0	0	1	1	1

1	1	1	0	0	1	0	0

XOR

0	0	1	0	0	1	1	1

=

1	1	0	0	0	0	1	1

1	1	1	1	0	0	0	0	0b11110000

NOT

0	0	0	0	1	1	1	1	0b1111

1	1	1	1	0	0	0	0	0b11110000

NOT

0	0	0	0	1	1	1	1	0b1111

–	1	1	1	1	0	0	0	1	-0b11110001

Representing Negative Integers with Two's Complement

Representing Negative Integers with Two's Complement

Signed
decimal

$-127 \leq x \leq 128$

Representing Negative Integers with Two's Complement

Signed decimal

$-127 \leq x \leq 128$

-58

Representing Negative Integers with Two's Complement

Signed
decimal
$-127 \leq x \leq 128$

58

abs

−58

Representing Negative Integers with Two's Complement

Signed decimal

$-127 \leq x \leq 128$

Fixed-precision 8-bit two's complement

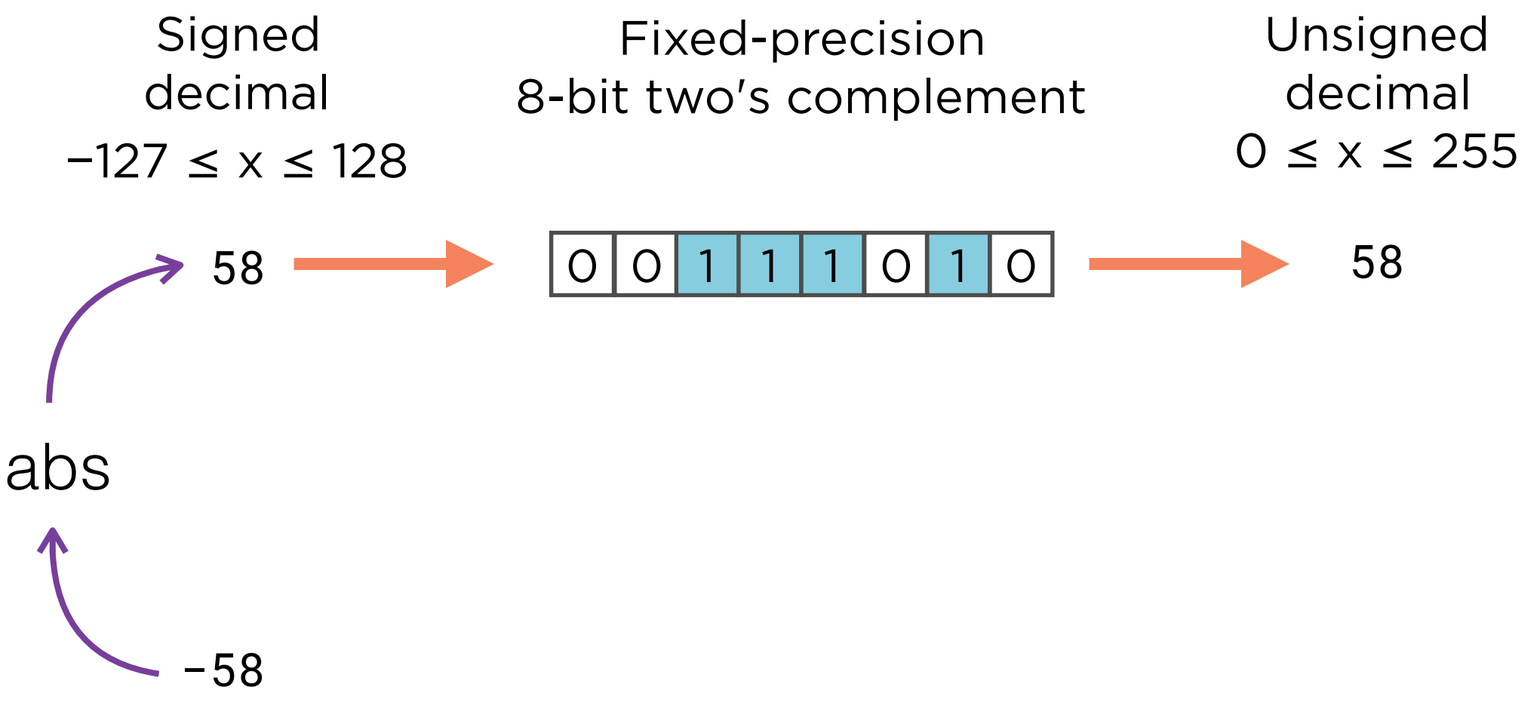
58

0	0	1	1	1	0	1	0

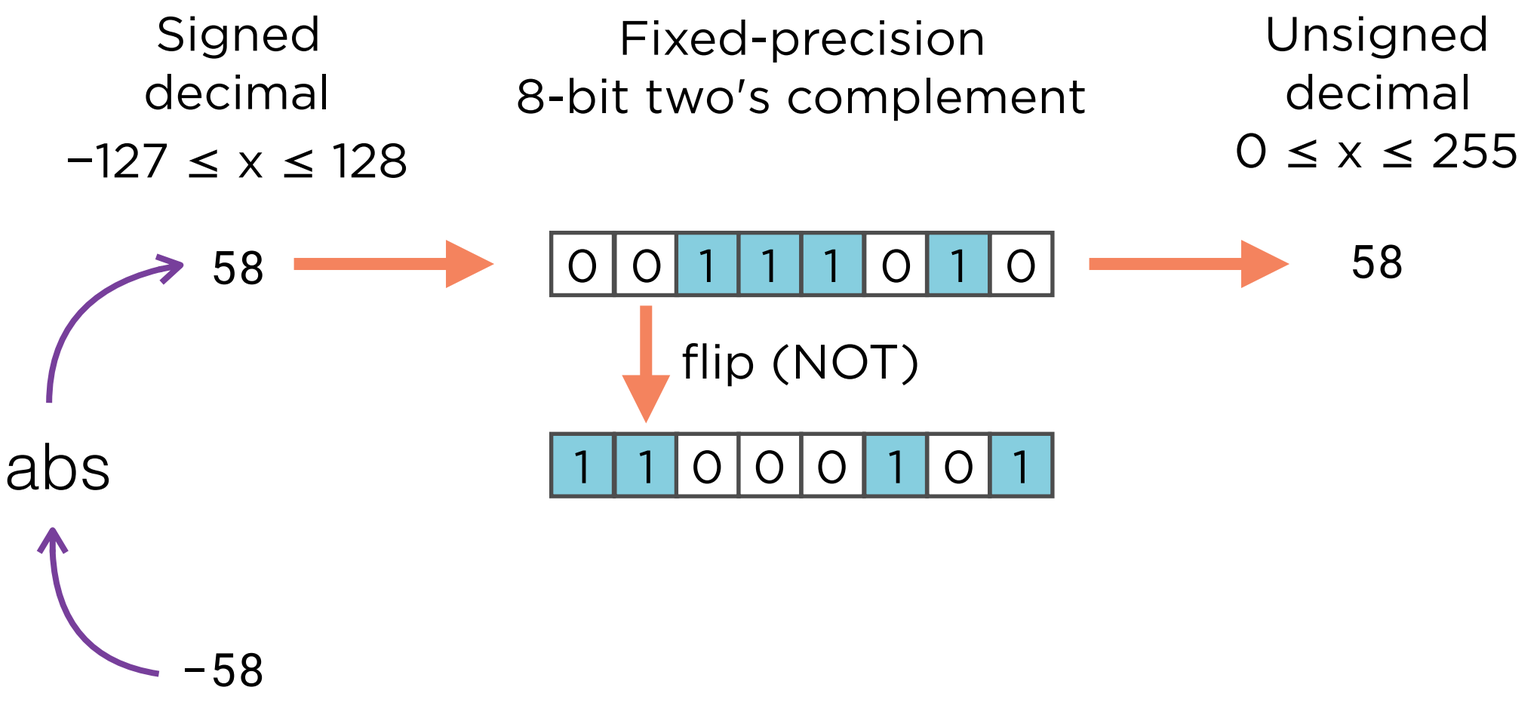
abs

−58

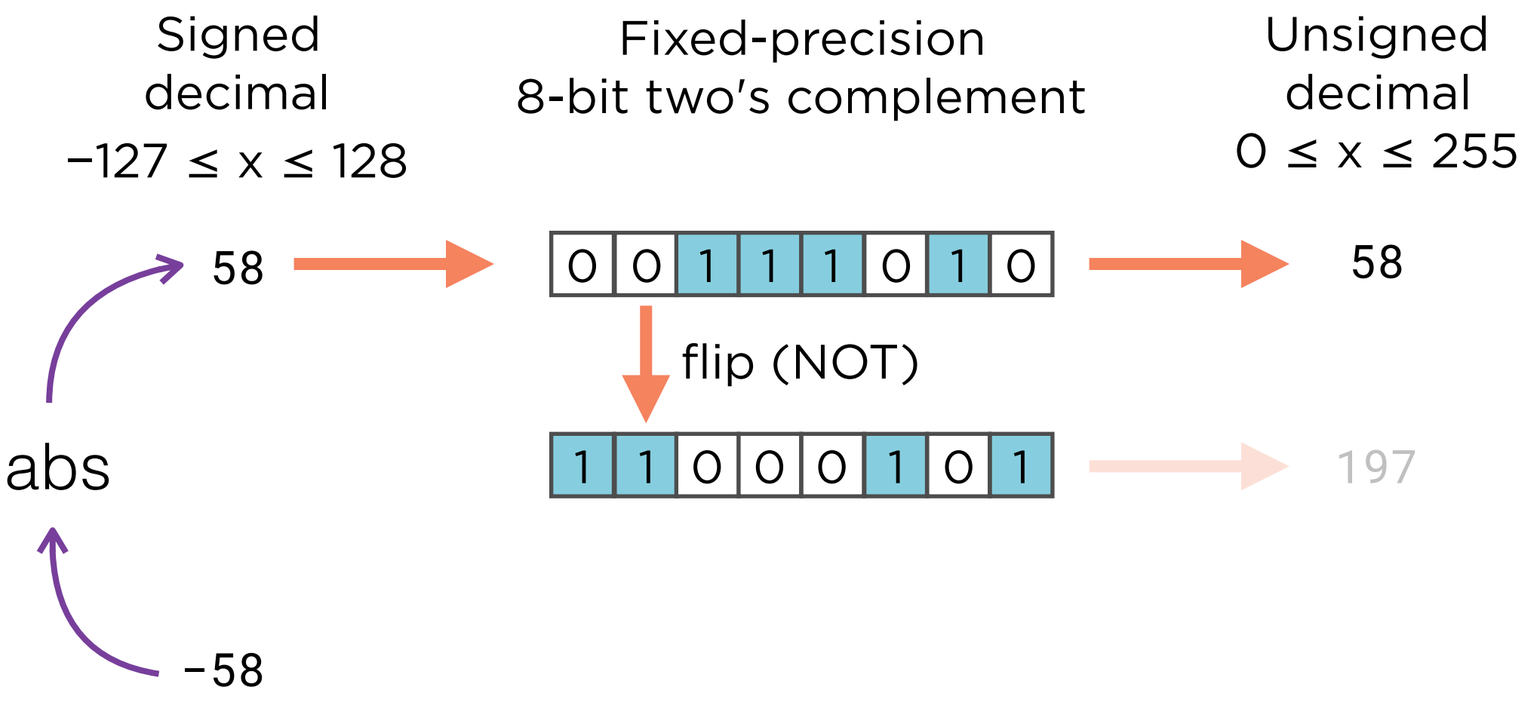
Representing Negative Integers with Two's Complement



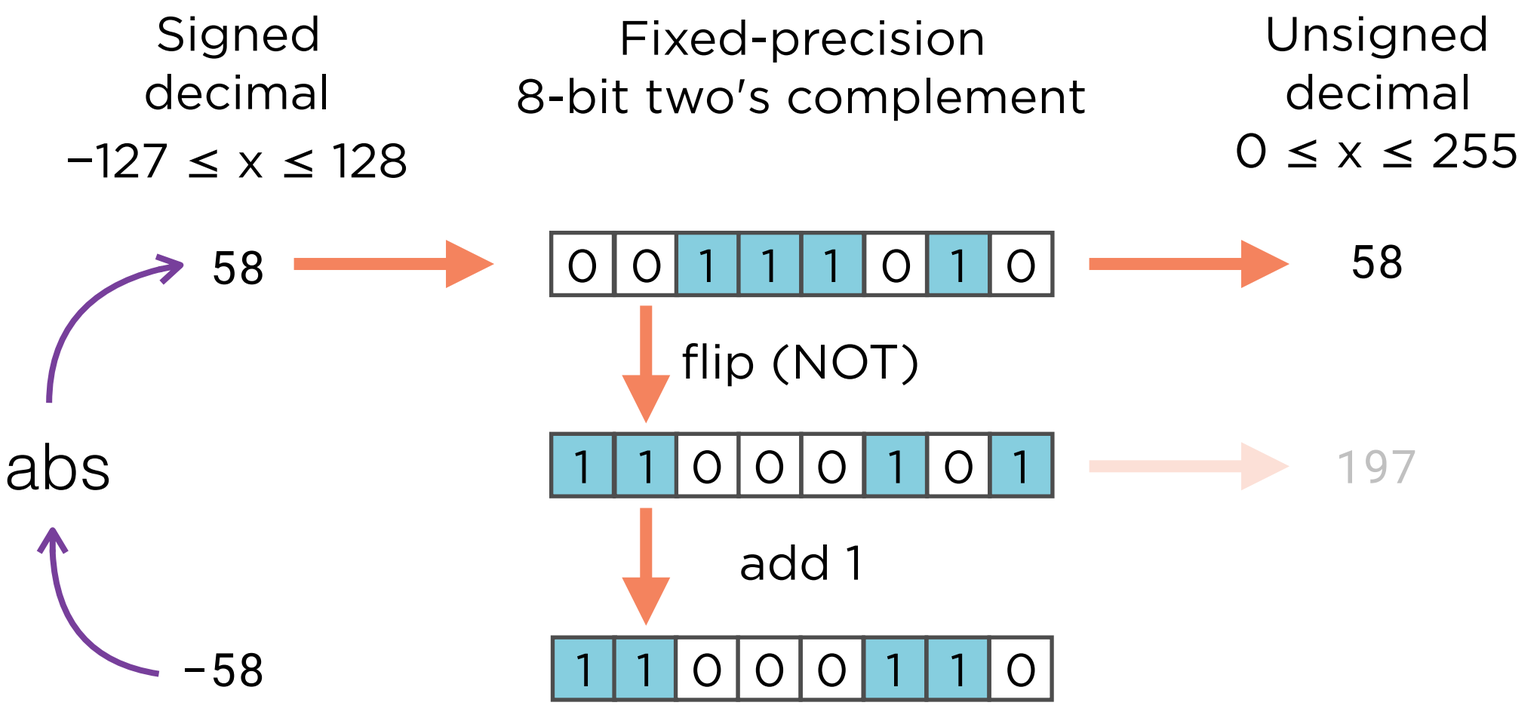
Representing Negative Integers with Two's Complement



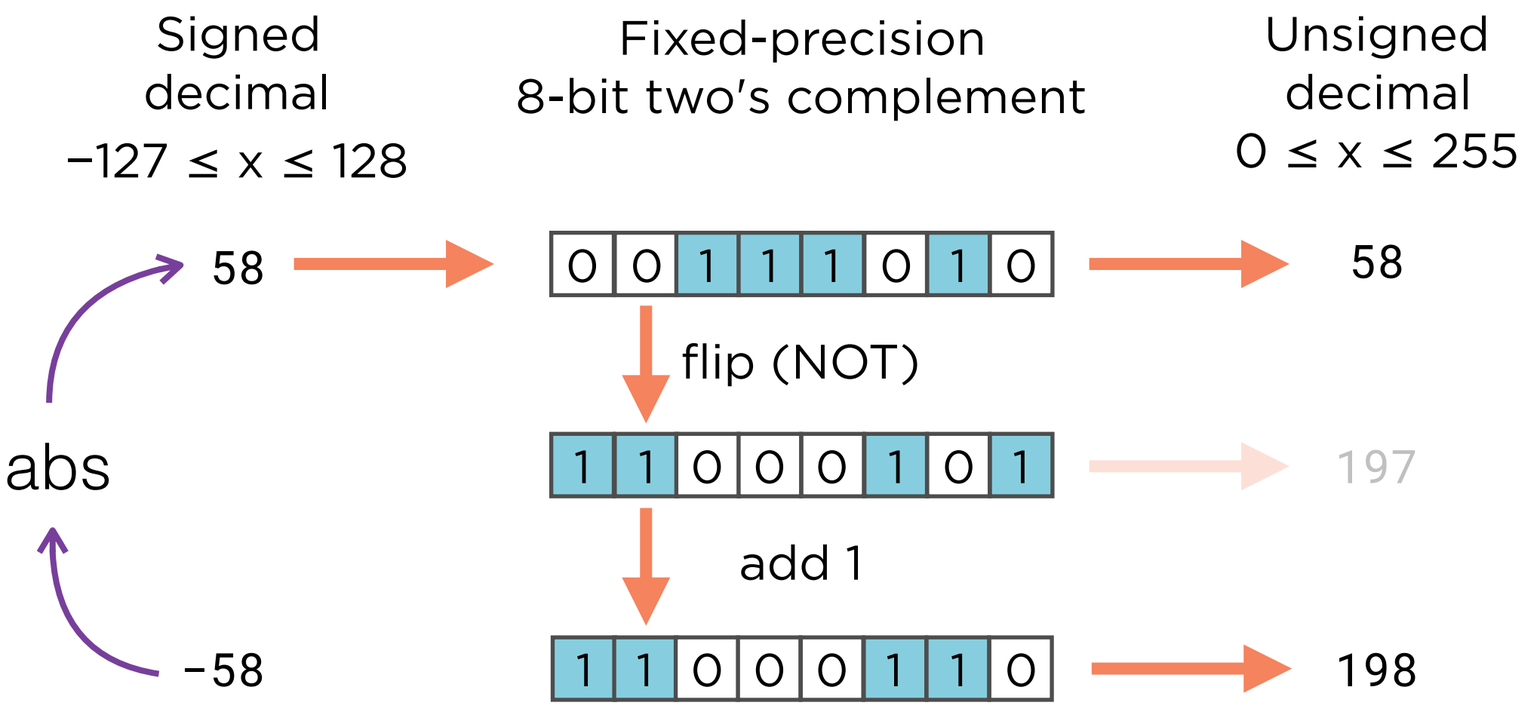
Representing Negative Integers with Two's Complement



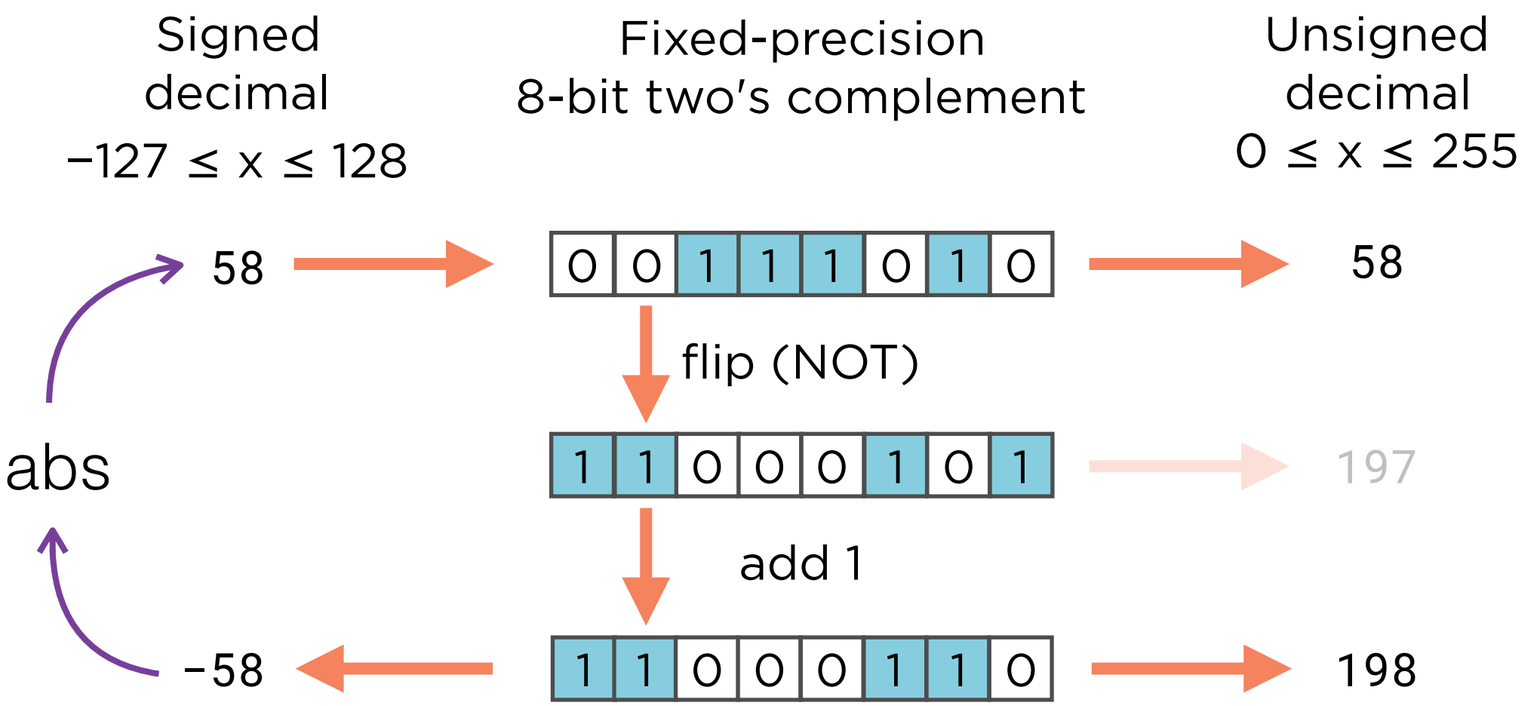
Representing Negative Integers with Two's Complement



Representing Negative Integers with Two's Complement



Representing Negative Integers with Two's Complement



Representing Negative Integers with Two's Complement

Representing Negative Integers with Two's Complement

Superior to sign-bit and magnitude

Single representation of zero

Natural arithmetic

Representing Negative Integers with Two's Complement

Superior to sign-bit and magnitude

Single representation of zero

Natural arithmetic

Python 3 integers

Unlimited precision

No fixed-width representation

Representing Negative Integers with Two's Complement

Superior to sign-bit and magnitude

Single representation of zero

Natural arithmetic

Python 3 integers

Unlimited precision

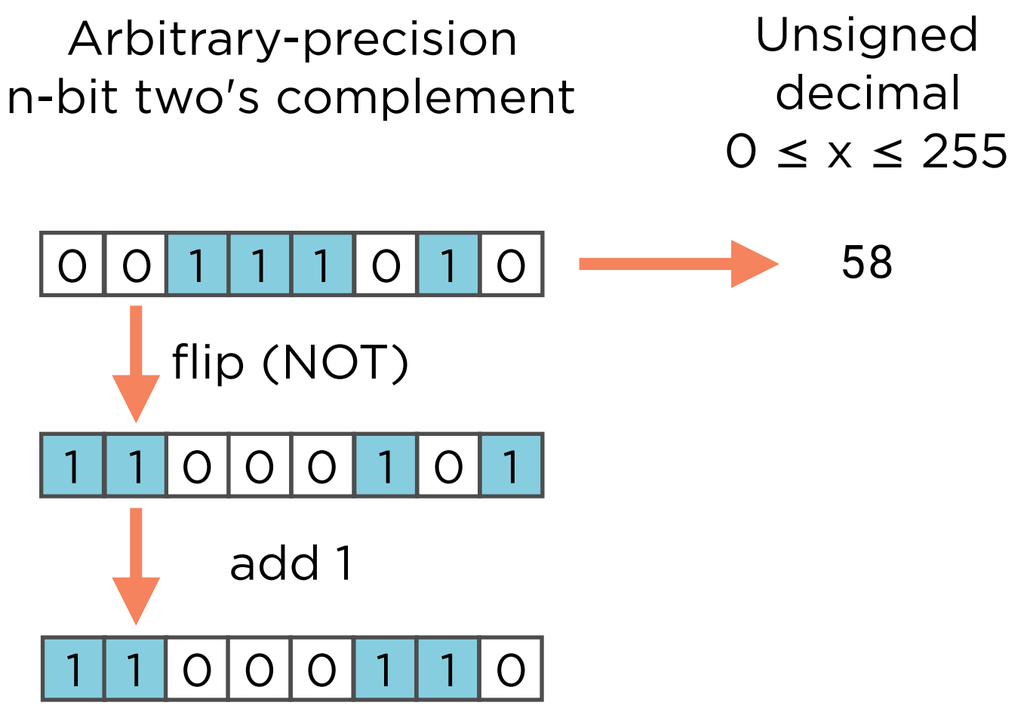
No fixed-width representation

Two's complement

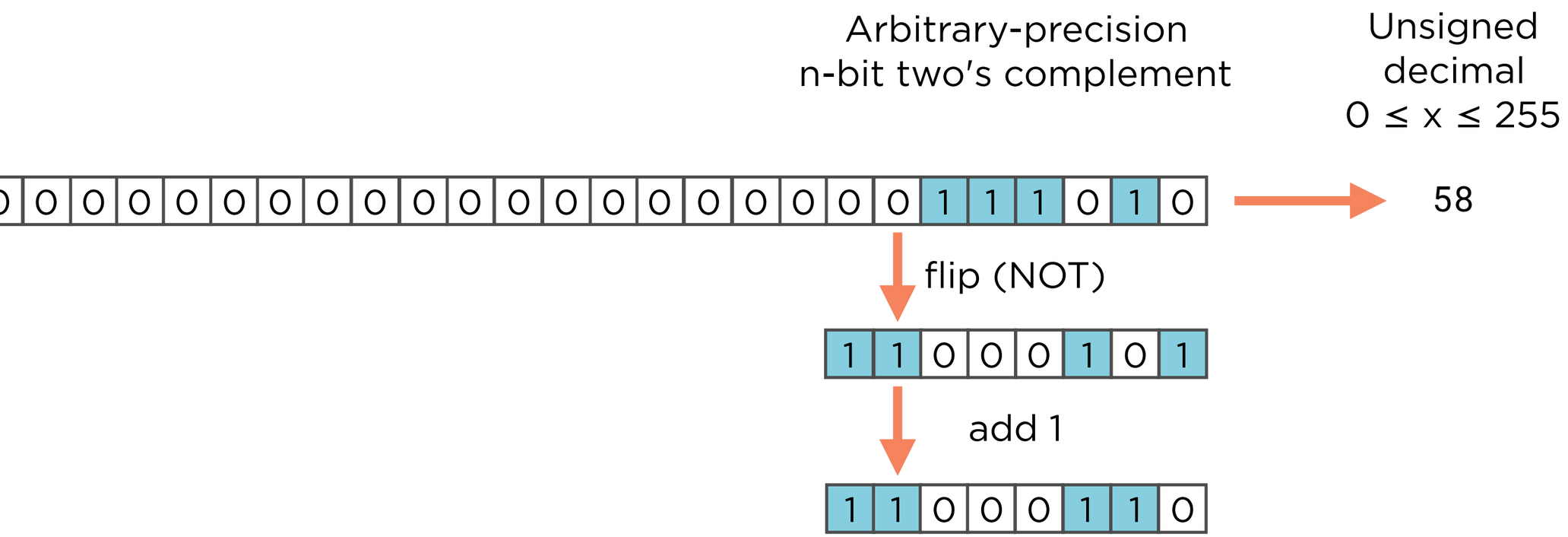
Unlimited precision

?

Representing Negative Integers with Two's Complement



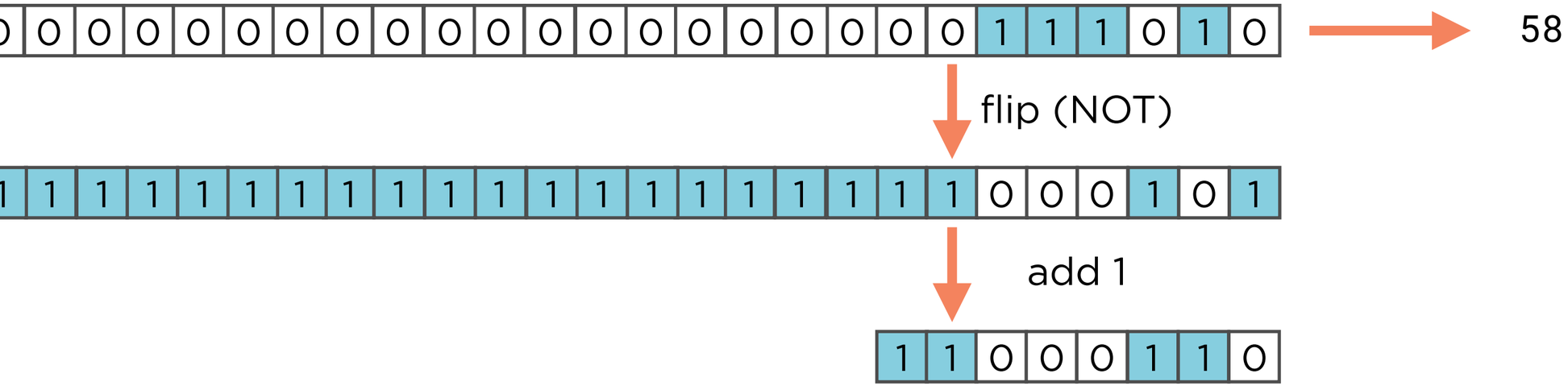
Representing Negative Integers with Two's Complement



Representing Negative Integers with Two's Complement

Arbitrary-precision
n-bit two's complement

Unsigned
decimal
$0 \leq x \leq 255$

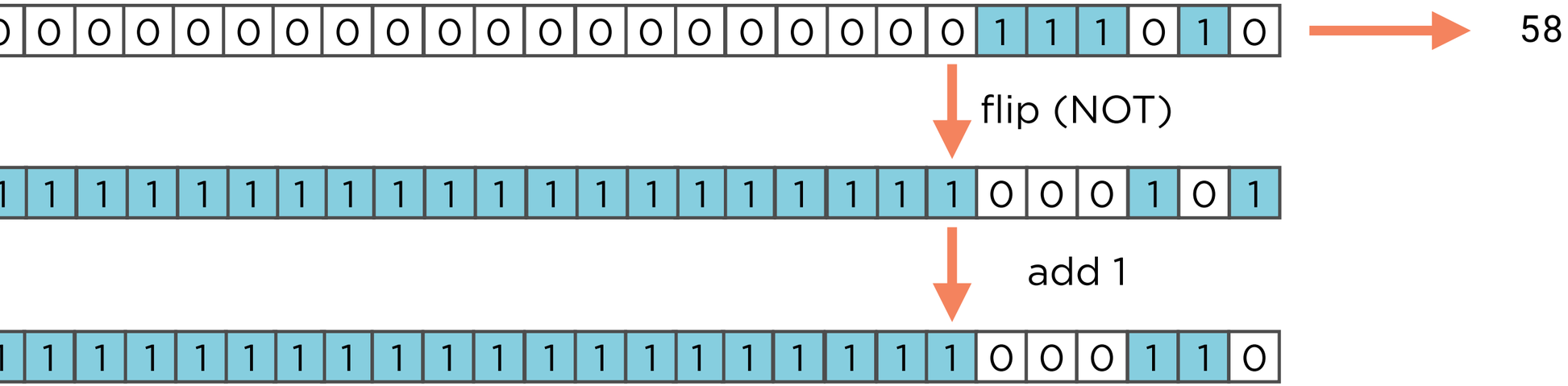


58

Representing Negative Integers with Two's Complement

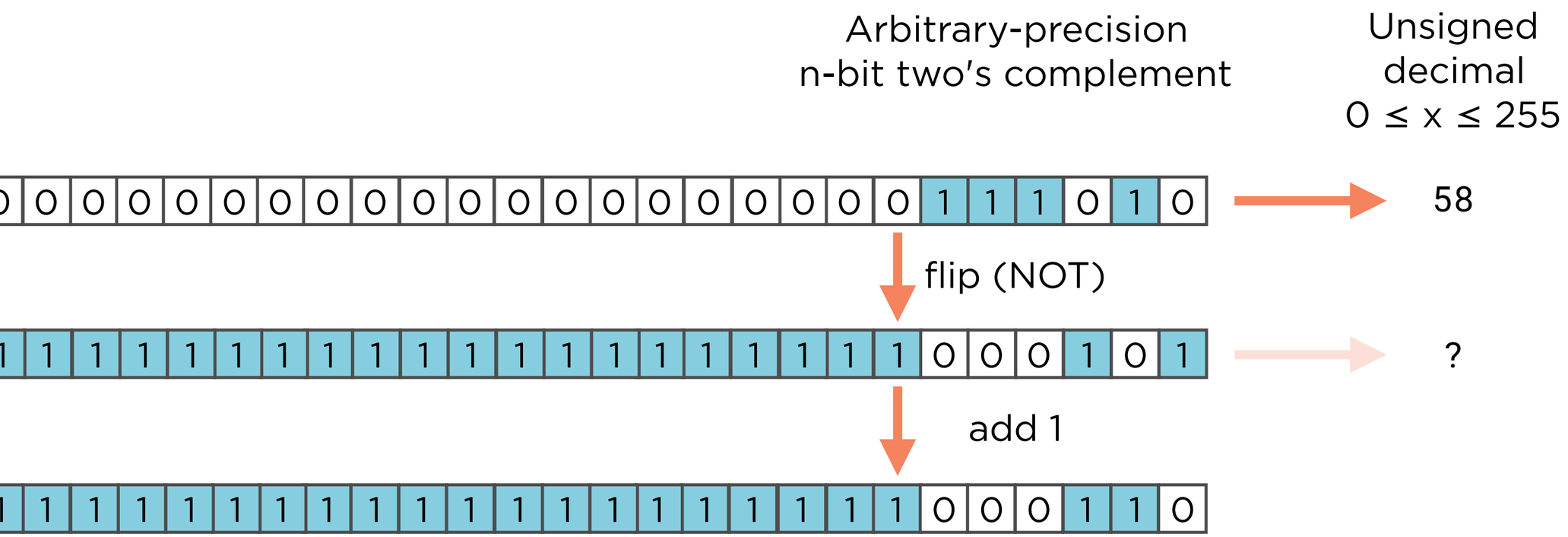
Arbitrary-precision
n-bit two's complement

Unsigned
decimal
$0 \leq x \leq 255$

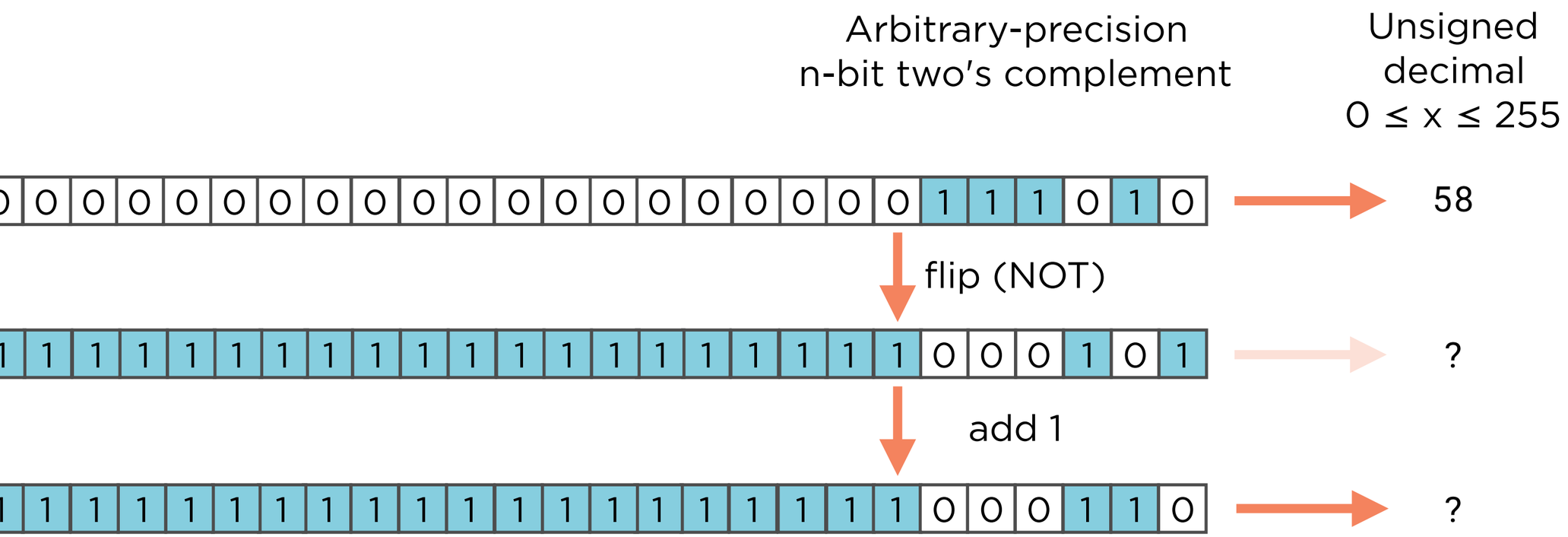


58

Representing Negative Integers with Two's Complement

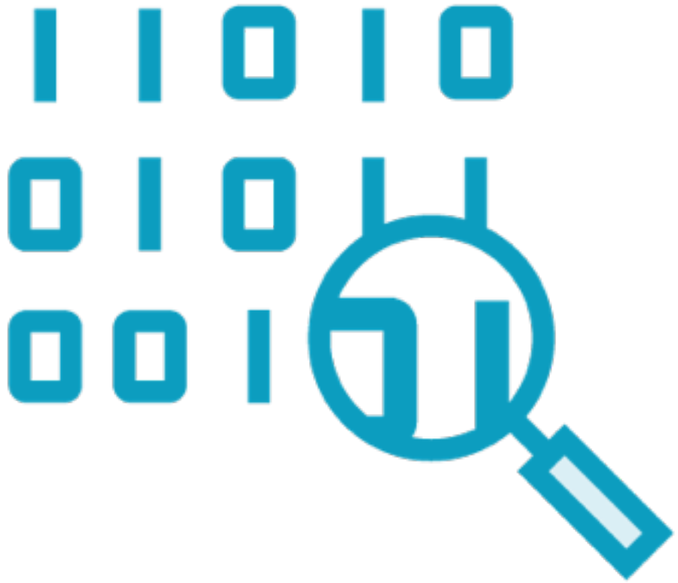


Representing Negative Integers with Two's Complement



If bitwise operations result in negative numbers...

If bitwise operations result in negative numbers...



Bitwise not

Bitwise not

```
>>>
```

Bitwise not

```
>>> v = 0b11110000
```

Bitwise not

v =

```
>>> v = 0b11110000
```

Bitwise not

Binary sign and magnitude

v =

```
>>> v = 0b11110000
```

Bitwise not

Binary sign and magnitude

v = +0b11110000

```
>>> v = 0b11110000
```

Bitwise not

Binary sign and magnitude

v = +0b11110000

```
>>> v = 0b11110000
>>>
```

Bitwise not

Binary sign and magnitude

v = +0b11110000

```
>>> v = 0b11110000
>>> v
```

Bitwise not

	Signed decimal		Binary sign and magnitude
v =	240	←	+0b11110000

```
>>> v = 0b11110000
>>> v
```

Bitwise not

	Signed decimal		Binary sign and magnitude
v =	240	←	+0b11110000

```
>>> v = 0b11110000
>>> v
240
>>>
```

Bitwise not

	Signed decimal		Binary sign and magnitude
v =	240	←	+0b11110000

```
>>> v = 0b11110000
>>> v
240
>>> ~v
```

Bitwise not

	Binary two's complement 8-bit		Signed decimal		Binary sign and magnitude
v =	1 1 1 1 0 0 0 0	←	240	←	+0b11110000

```
>>> v = 0b11110000
>>> v
240
>>> ~v
```

Bitwise not

Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v = 0 1 1 1 1 0 0 0 0	240	+0b11110000

```
>>> v = 0b11110000
>>> v
240
>>> ~v
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =			

```
>>> v = 0b11110000
>>> v
240
>>> ~v
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1		

```
>>> v = 0b11110000
>>> v
240
>>> ~v
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	

```
>>> v = 0b11110000
>>> v
240
>>> ~v
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>>
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	-241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	-241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Take two's complement of magnitude

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	~~-~~0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Take two's complement of magnitude

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

abs

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Take two's complement of magnitude

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

abs

1	1	1	1	0	0	0	1

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Take two's complement of magnitude

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

abs

1 1 1 1 0 0 0 1

flip (NOT)

Take two's complement of magnitude

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

abs

1	1	1	1	0	0	0	1

flip (NOT)

0	0	0	0	1	1	1	0

Take two's complement of magnitude

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

abs

1	1	1	1	0	0	0	1

flip (NOT)

0	0	0	0	1	1	1	0

add 1

Take two's complement of magnitude

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Take two's complement of magnitude

abs

1 1 1 1 0 0 0 1

flip (NOT)

0 0 0 0 1 1 1 0

add 1

0 0 0 0 1 1 1 1

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Take two's complement of magnitude

abs

1 1 1 1 0 0 0 1

flip (NOT)

0 0 0 0 1 1 1 0

add 1

0 0 0 0 1 1 1 1

✓

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

+

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

+

$2^{\text{num\_bits}} =$

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

+

2^{num_bits} = 256

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

+

$2^{\text{num\_bits}} =$ 256

=

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

+

2^{num_bits} = 256

=

15

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

2^{num_bits} =

+

256

=

15

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

2^{num_bits} =

+

256

=

15

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

2^{num_bits} =

+

256

=

15 → 0 0 0 0 1 1 1 1

Bitwise not

	Binary two's complement ~~8-bit~~ 9-bit	Signed decimal	Binary sign and magnitude
v =	0 1 1 1 1 0 0 0 0	240	+0b11110000
~v =	1 0 0 0 0 1 1 1 1	−241	-0b11110001

```
>>> v = 0b11110000
>>> v
240
>>> ~v
-241
>>> bin(~v)
'-0b11110001'
```

2^{num_bits} =

+

256

=

15 → 0 0 0 0 1 1 1 1 ✓

Two's complement with bitwise shift

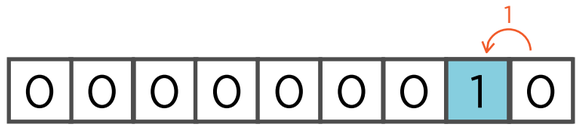
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
```

0	0	0	0	0	0	0	0	1

1 << 0 = 1

Two's complement with bitwise shift

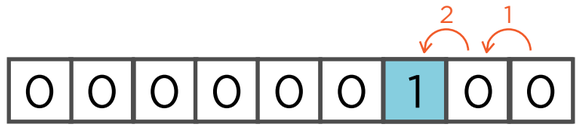
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
```



1 << 1 = 2

Two's complement with bitwise shift

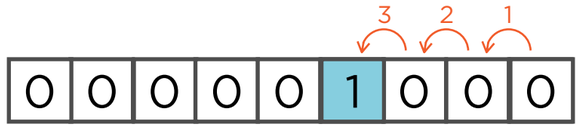
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
```



1 << 2 = 4

Two's complement with bitwise shift

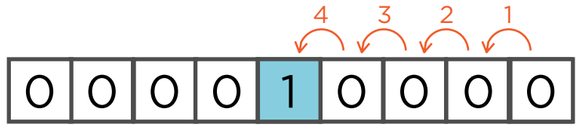
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
```



1 << 3 = 8

Two's complement with bitwise shift

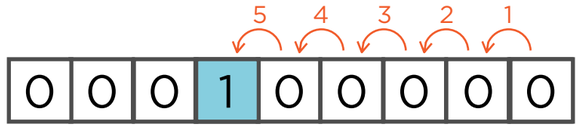
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
```



1 << 4 = 16

Two's complement with bitwise shift

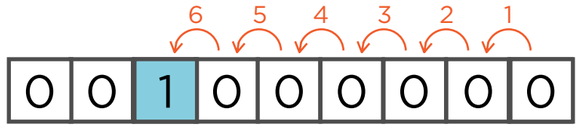
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
```



1 << 5 = 32

Two's complement with bitwise shift

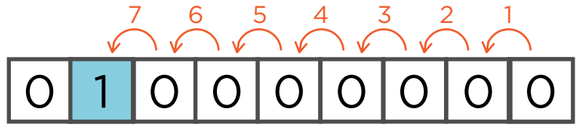
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
```



1 << 6 = 64

Two's complement with bitwise shift

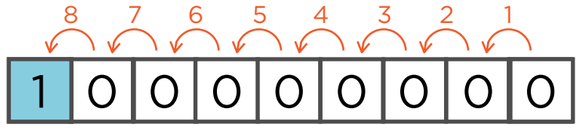
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
```



1 << 7 = 128

Two's complement with bitwise shift

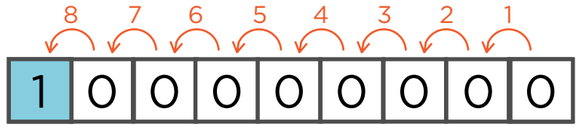
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
```



1 << 8 = 256

Two's complement with bitwise shift

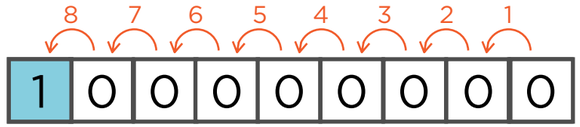
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
>>>
```



1 << 8 = 256

Two's complement with bitwise shift

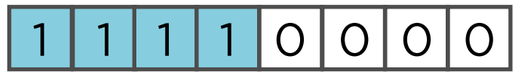
```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
>>> v
```

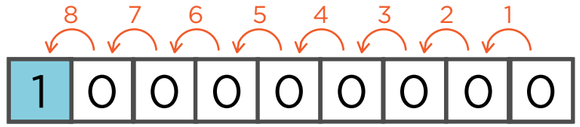


1 << 8 = 256

Two's complement with bitwise shift

```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
>>> v
240
>>>
```

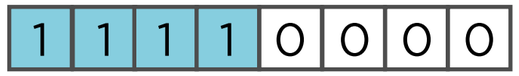


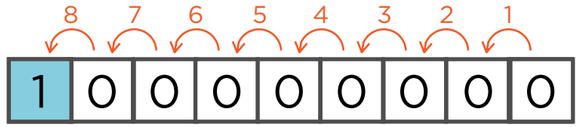


1 << 8 = 256

Two's complement with bitwise shift

```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
>>> v
240
>>> ~v
```

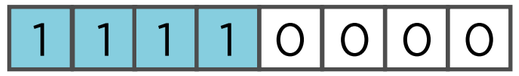


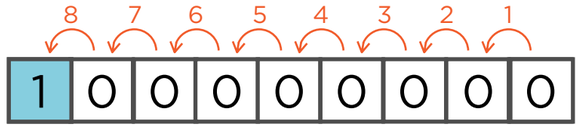


1 << 8 = 256

Two's complement with bitwise shift

```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
>>> v
240
>>> ~v
-241
>>>
```

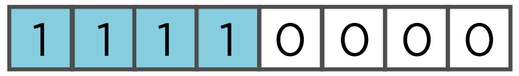


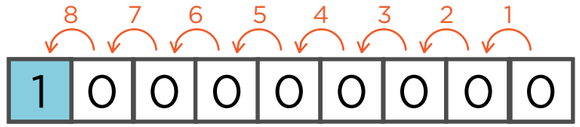


1 << 8 = 256

Two's complement with bitwise shift

```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
>>> v
240
>>> ~v
-241
>>> twos_complement(~v, 8)
```

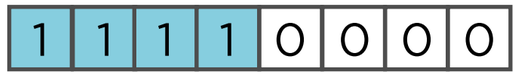


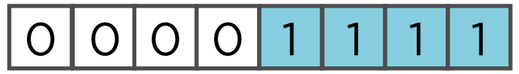


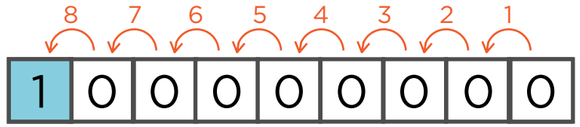
1 << 8 = 256

Two's complement with bitwise shift

```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
>>> v
240
>>> ~v
-241
>>> twos_complement(~v, 8)
15
>>>
```







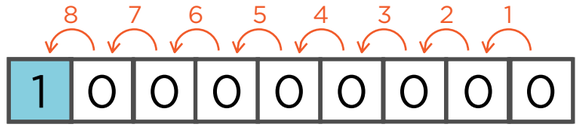
1 << 8 = 256

Two's complement with bitwise shift

```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
>>> v
240
>>> ~v
-241
>>> twos_complement(~v, 8)
15
>>> bin(_)
```

1	1	1	1	0	0	0	0

0	0	0	0	1	1	1	1



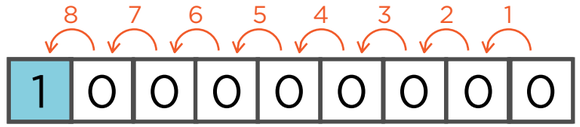
1 << 8 = 256

Two's complement with bitwise shift

```
>>> def twos_complement(x, num_bits):
...     if x < 0:
...         return x + (1 << num_bits)
...     return x
>>> v
240
>>> ~v
-241
>>> twos_complement(~v, 8)
15
>>> bin(_)
'0b1111'
>>>
```

1	1	1	1	0	0	0	0

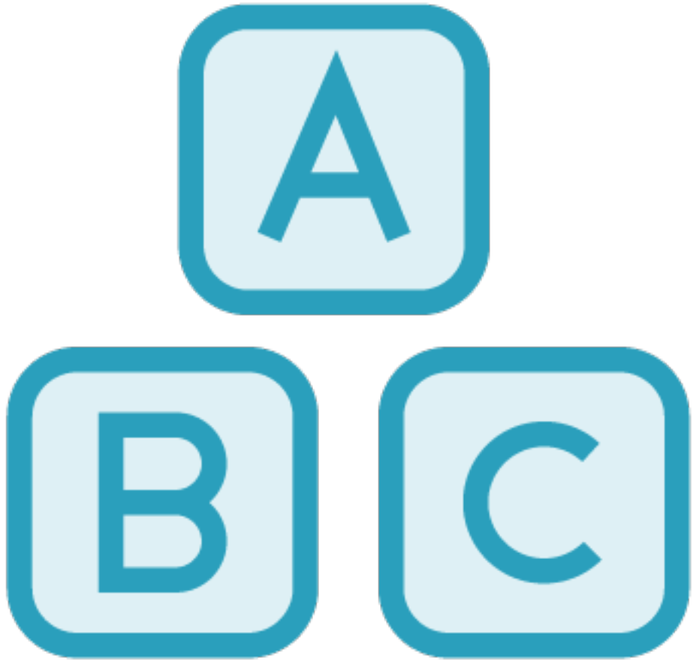
0	0	0	0	1	1	1	1



1 << 8 = 256

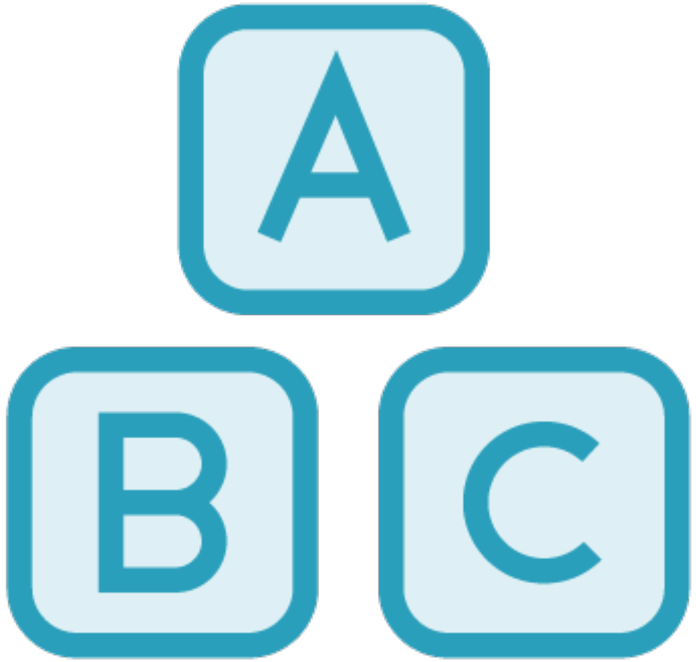
Byte-wise Operations with Integers

The `bytes` type in-depth



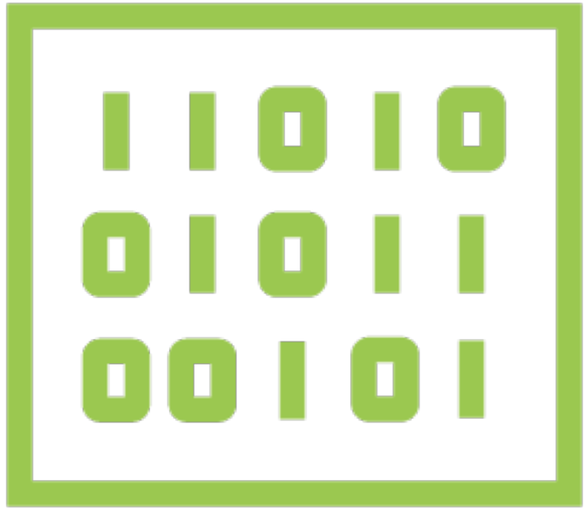
`str`

Immutable sequence of
Unicode code points



str

Immutable sequence of Unicode code points



bytes

Immutable sequence of bytes

```
b'A literal byte string'
```

```
b'A literal byte string'
```

bytes

Immutable sequence of bytes

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	NUL	DLE		0	@	P	`	p								
1	SOH	DC1	!	1	A	Q	a	q								
2	STX	DC2	"	2	B	R	b	r								
3	ETX	DC3	#	3	C	S	c	s								
4	EOT	DC4	$	4	D	T	d	t								
5	ENQ	NAK	%	5	E	U	e	u								
6	ACK	SYN	&	6	F	V	f	v								
7	BEL	ETB	'	7	G	W	g	w								
8	BS	CAN	(	8	H	X	h	x								
9	HT	EM	)	9	I	Y	i	y								
a	LF	SUB	*	:	J	Z	j	z								
b	VT	ESC	+	;	K	[	k	{								
c	FF	FS	,	<	L	\	l	|								
d	CR	GS	-	=	M	]	m	}								
e	SO	RS	.	>	N	^	n	~								
f	SI	US	/	?	O	_	o	DEL								

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	NUL	DLE		0	@	P	`	p								
1	SOH	DC1	!	1	A	Q	a	q								
2	STX	DC2	"	2	B	R	b	r								
3	ETX	DC3	#	3	C	S	c	s								
4	EOT	DC4	$	4	D	T	d	t								
5	ENQ	NAK	%	5	E	U	e	u								
6	ACK	SYN	&	6	F	V	f	v								
7	BEL	ETB	'	7	G	W	g	w								
8	BS	CAN	(	8	H	X	h	x								
9	HT	EM	)	9	I	Y	i	y								
a	LF	SUB	*	:	J	Z	j	z								
b	VT	ESC	+	;	K	[	k	{								
c	FF	FS	,	<	L	\	l	|								
d	CR	GS	-	=	M	]	m	}								
e	SO	RS	.	>	N	^	n	~								
f	SI	US	/	?	O	_	o	DEL								

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	NUL	DLE		0	@	P	`	p								
1	SOH	DC1	!	1	A	Q	a	q								
2	STX	DC2	"	2	B	R	b	r								
3	ETX	DC3	#	3	C	S	c	s								
4	EOT	DC4	$	4	D	T	d	t								
5	ENQ	NAK	%	5	E	U	e	u								
6	ACK	SYN	&	6	F	V	f	v								
7	BEL	ETB	'	7	G	W	g	w								
8	BS	CAN	(	8	H	X	h	x								
9	HT	EM	)	9	I	Y	i	y								
a	LF	SUB	*	:	J	Z	j	z								
b	VT	ESC	+	;	K	[	k	{								
c	FF	FS	,	<	L	\	l	|								
d	CR	GS	-	=	M	]	m	}								
e	SO	RS	.	>	N	^	n	~								
f	SI	US	/	?	O	_	o	DEL								

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	NUL	DLE		0	@	P	`	p								
1	SOH	DC1	!	1	A	Q	a	q								
2	STX	DC2	"	2	B	R	b	r								
3	ETX	DC3	#	3	C	S	c	s								
4	EOT	DC4	$	4	D	T	d	t								
5	ENQ	NAK	%	5	E	U	e	u								
6	ACK	SYN	&	6	F	V	f	v								
7	BEL	ETB	'	7	G	W	g	w								
8	BS	CAN	(	8	H	X	h	x								
9	HT	EM	)	9	I	Y	i	y								
a	LF	SUB	*	:	J	Z	j	z								
b	VT	ESC	+	;	K	[	k	{								
c	FF	FS	,	<	L	\	l	|								
d	CR	GS	-	=	M	]	m	}								
e	SO	RS	.	>	N	^	n	~								
f	SI	US	/	?	O	_	o	DEL								

The `bytearray` type

`bytearray`

Elements are bytes (integers 0–255)

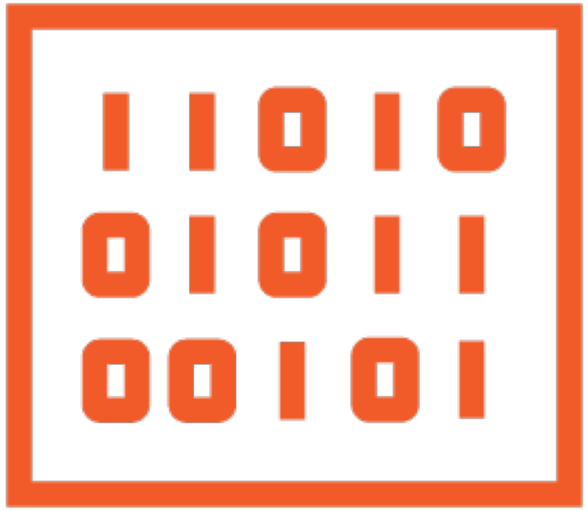
Sequence

Mutable

Similar methods to `list` and `bytes`

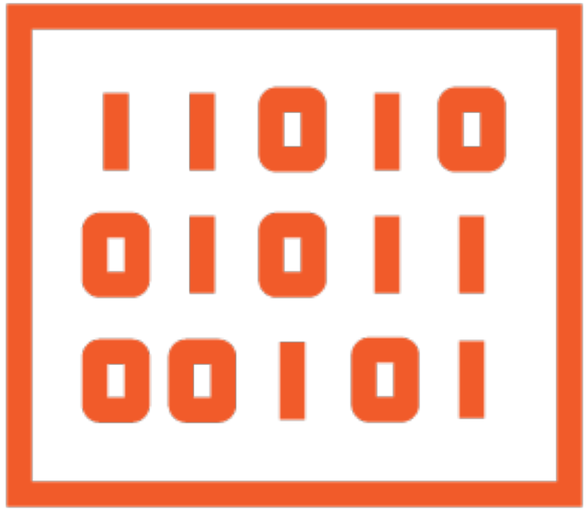
Interpreting binary structures with the `struct` module

C Structures for binary object interchange



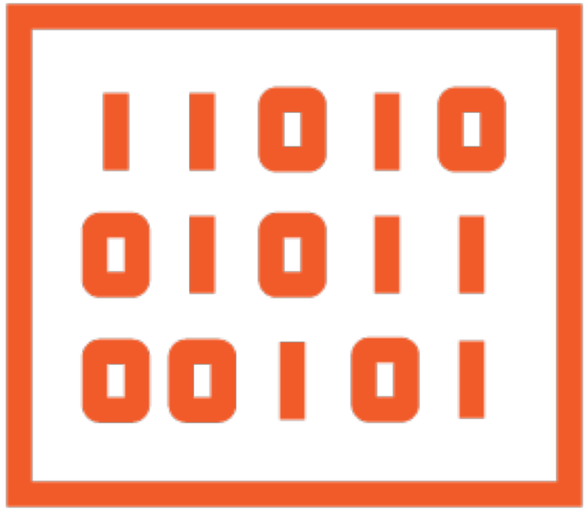
C structures

C Structures for binary object interchange

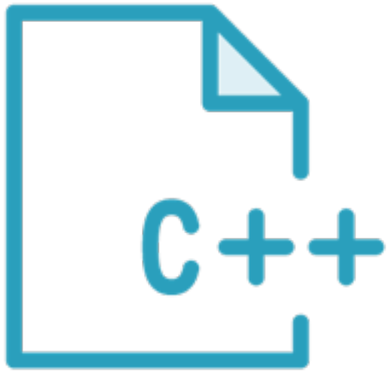


C structures

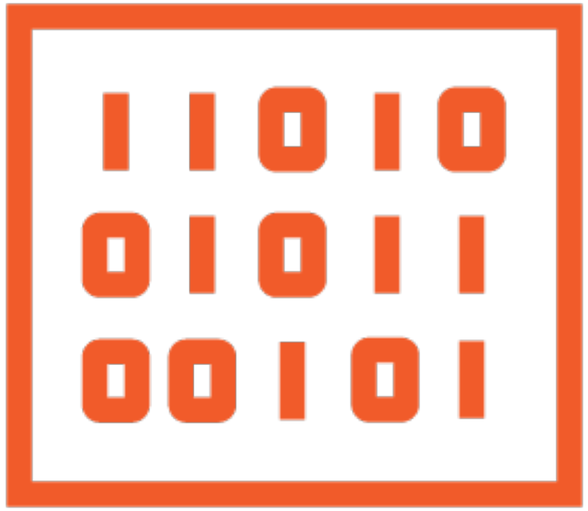
C Structures for binary object interchange



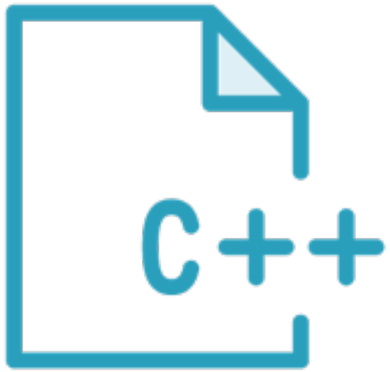
C structures



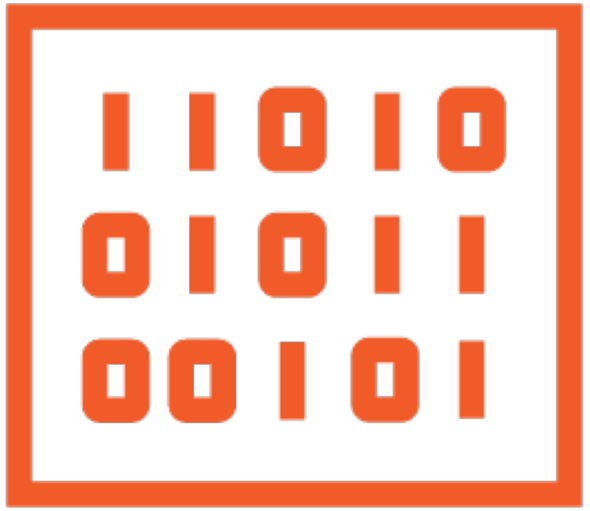
C Structures for binary object interchange



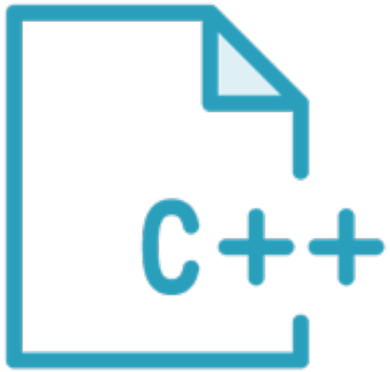
C structures



C Structures for binary object interchange



C structures



Demo

Demo

Demo

Using the `struct` module

Demo

Using the `struct` module

- Write binary data from C structs

Demo

Using the `struct` module

- Write binary data from C structs
- Read binary data in Python

Demo

Using the `struct` module

- Write binary data from C structs
- Read binary data in Python
- Create data object classes

```c
#include <stdio.h>

struct Vector {
    float x;
    float y;
    float z;
};

struct Color {
    unsigned short int red;
    unsigned short int green;
    unsigned short int blue;
};

struct Vertex {
    struct Vector position;
    struct Color color;
};
```

```c
int main(int argc, char** argv) {
    struct Vertex vertices[] = {
        { .position = { 3323.176, 6562.231, 9351.231 },
          .color = { 3040, 34423, 54321 } },

        { .position = { 7623.982, 2542.231, 9823.121 },
          .color = { 32736, 5342, 2321 } },

        { .position = { 6729.862, 2347.212, 3421.322 },
          .color = { 45263, 36291, 36701 } },

        { .position = { 6352.121, 3432.111, 9763.232 },
          .color = { 56222, 36612, 11214 } } };

    FILE* file = fopen("colors.bin", "wb");

    if (file == NULL) {
        return -1;
    }

    fwrite(vertices, sizeof(struct Vertex), 4, file);
    fclose(file);

    return 0;
}
```

Format Characters for the `struct` Module

Format	C Type	Python type	Standard size	Notes
`x`	pad byte	no value		
`c`	`char`	bytes of length 1	1	
`b`	`signed char`	integer	1	(1),(3)
`B`	`unsigned char`	integer	1	(3)
`?`	`_Bool`	bool	1	(1)
`h`	`short`	integer	2	(3)
`H`	`unsigned short`	integer	2	(3)
`i`	`int`	integer	4	(3)
`I`	`unsigned int`	integer	4	(3)
`l`	`long`	integer	4	(3)
`L`	`unsigned long`	integer	4	(3)
`q`	`long long`	integer	8	(2), (3)
`Q`	`unsigned long long`	integer	8	(2), (3)
`n`	`ssize_t`	integer		(4)
`N`	`size_t`	integer		(4)
`f`	`float`	float	4	(5)
`d`	`double`	float	8	(5)
`s`	`char[]`	bytes		
`p`	`char[]`	bytes		
`P`	`void *`	integer		(6)

https://docs.python.org/3/library/struct.html

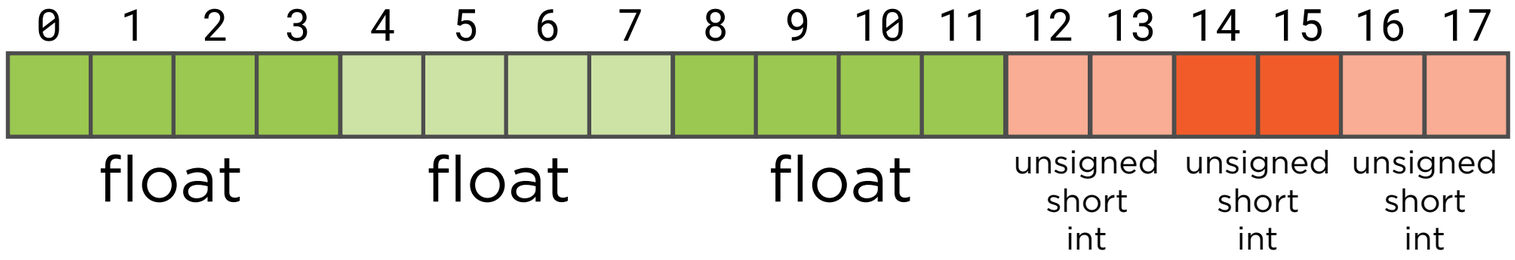
Byte Order, Size and Alignment

Character	Byte order	Size	Alignment
`@`	native	native	native
`=`	native	standard	none
`<`	little-endian	standard	none
`>`	big-endian	standard	none
`!`	network (= big-endian)	standard	none

If the first character is not one of these, `'@'` is assumed.

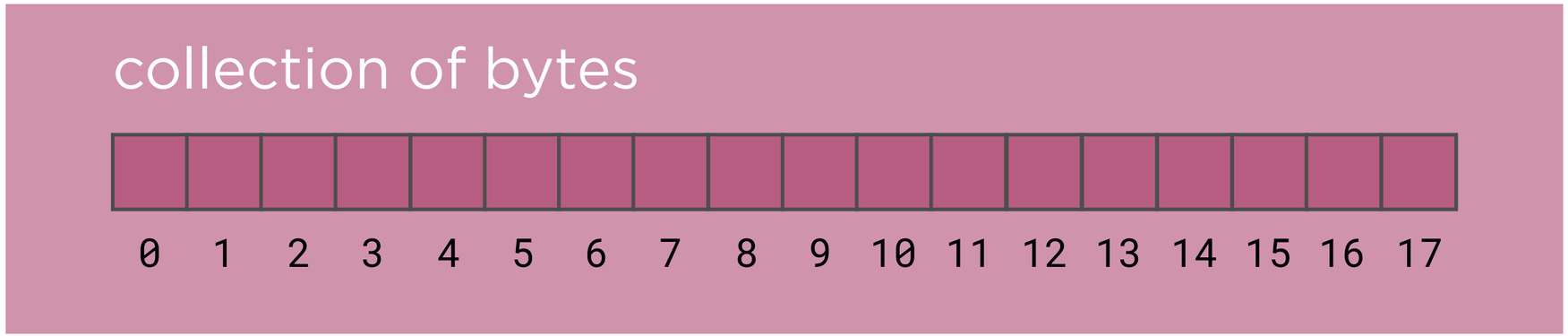
https://docs.python.org/3/library/struct.html

Binary Vertex Layout

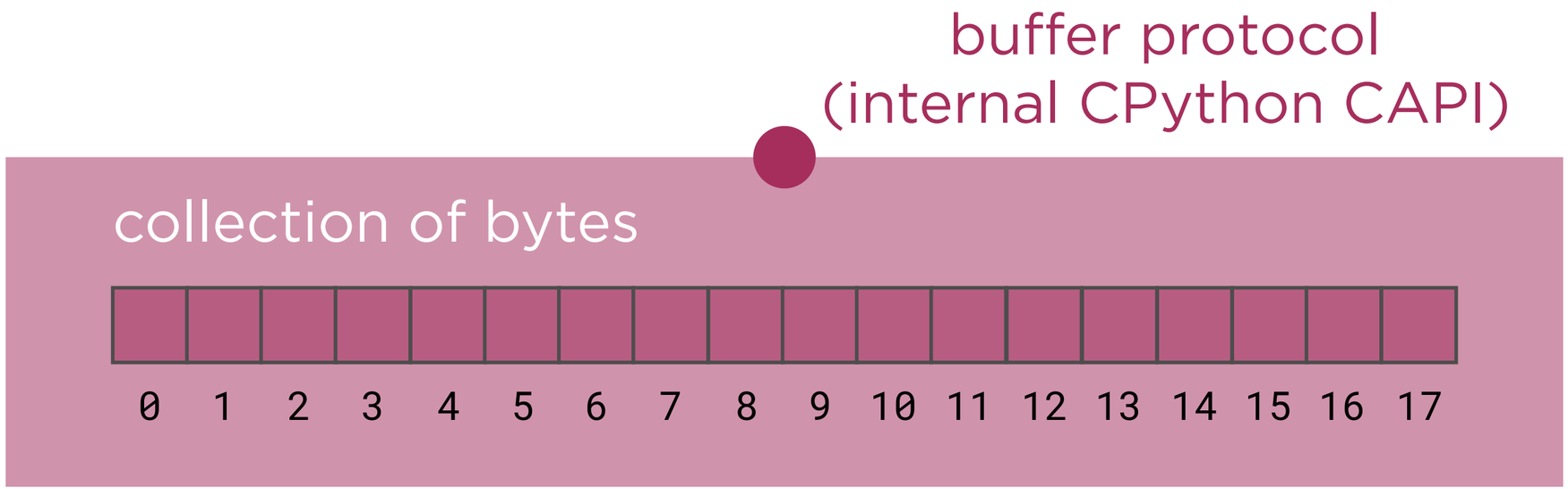


The `memoryview` type

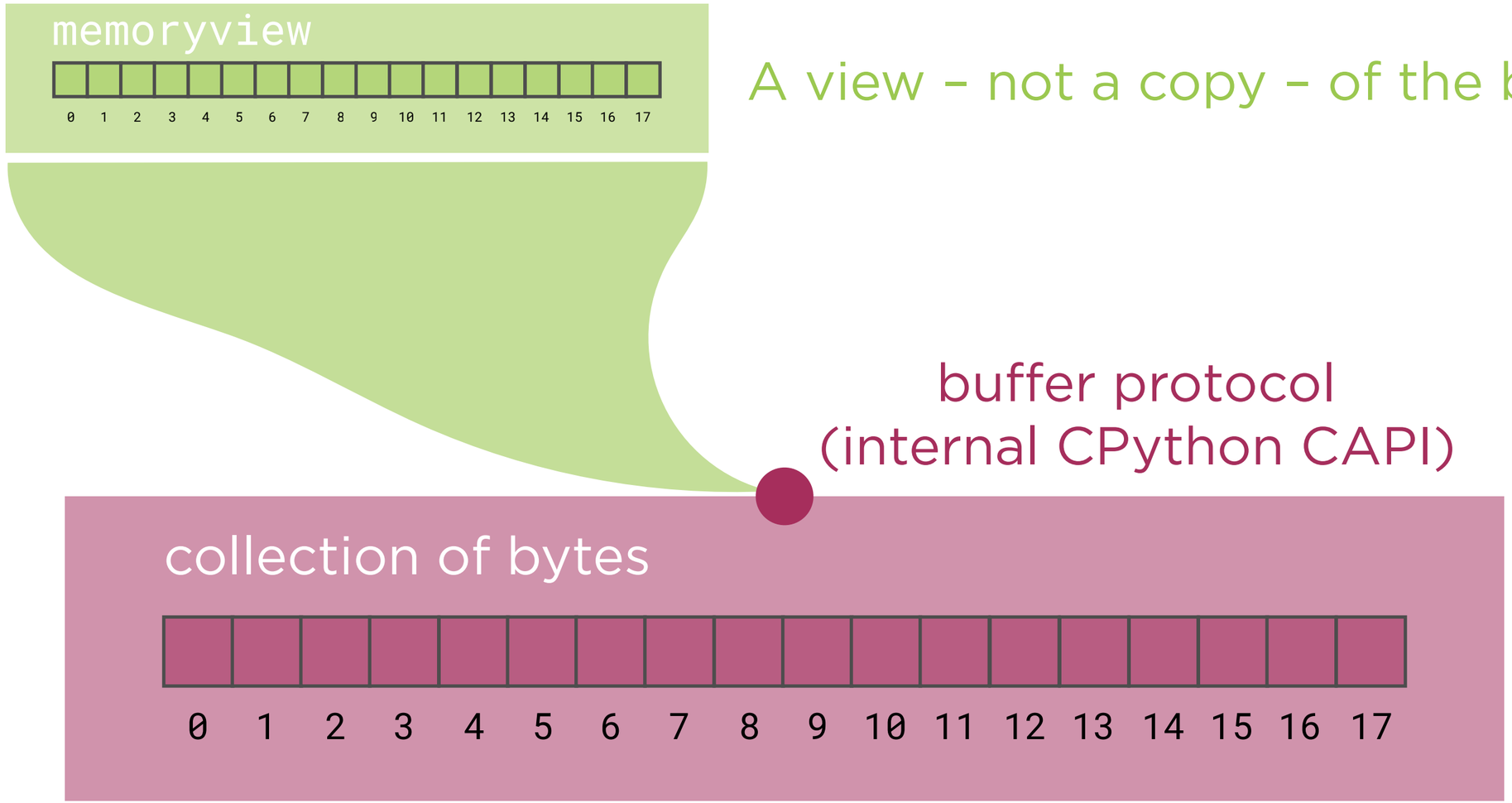
Memory views

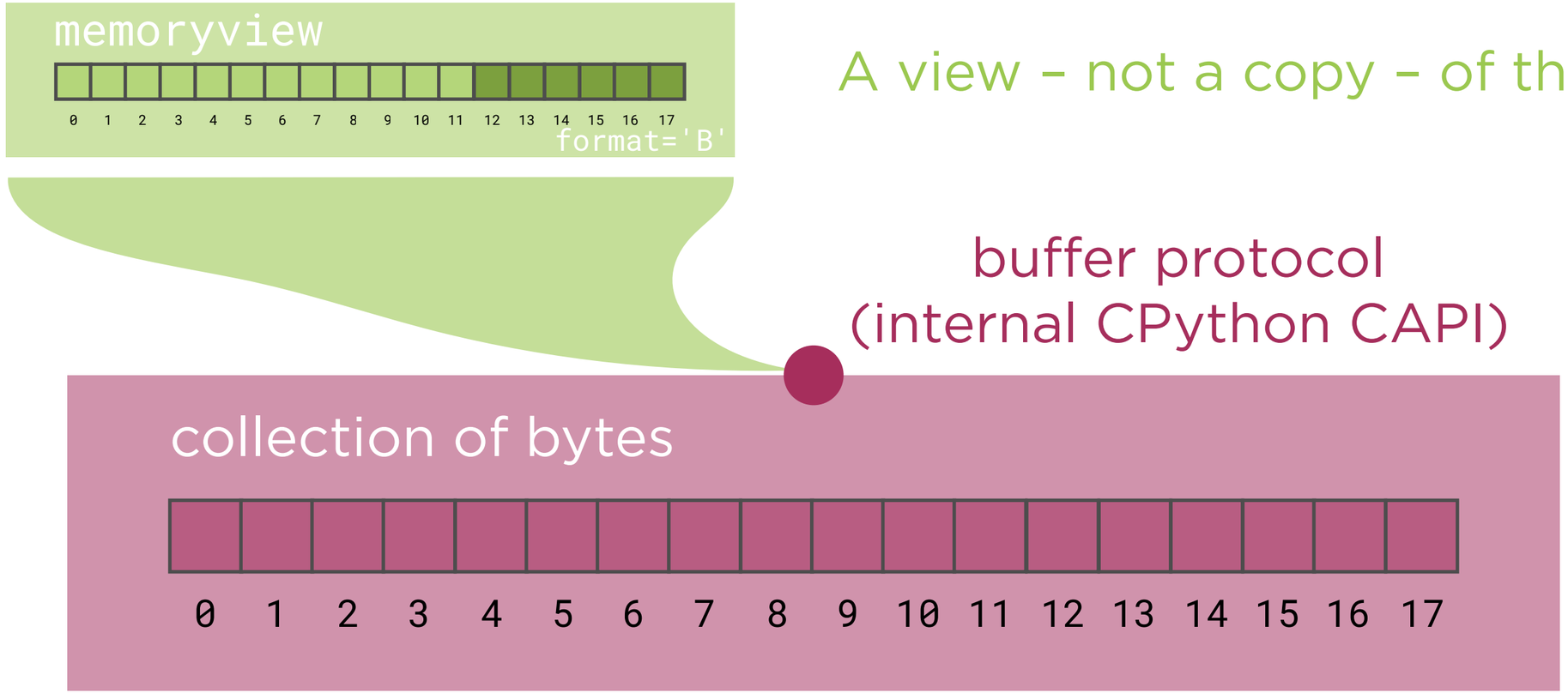


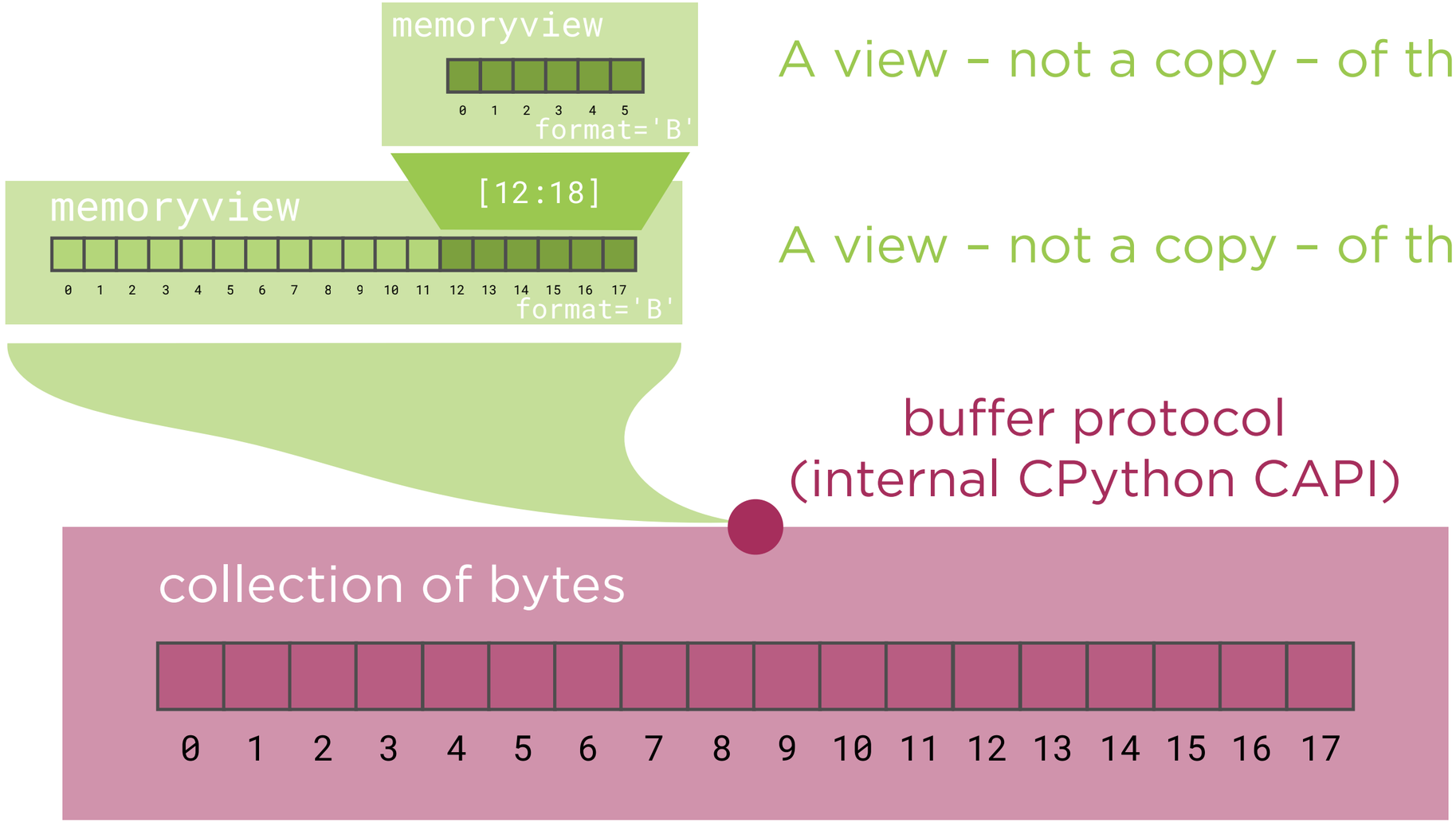
Memory views



Memory views



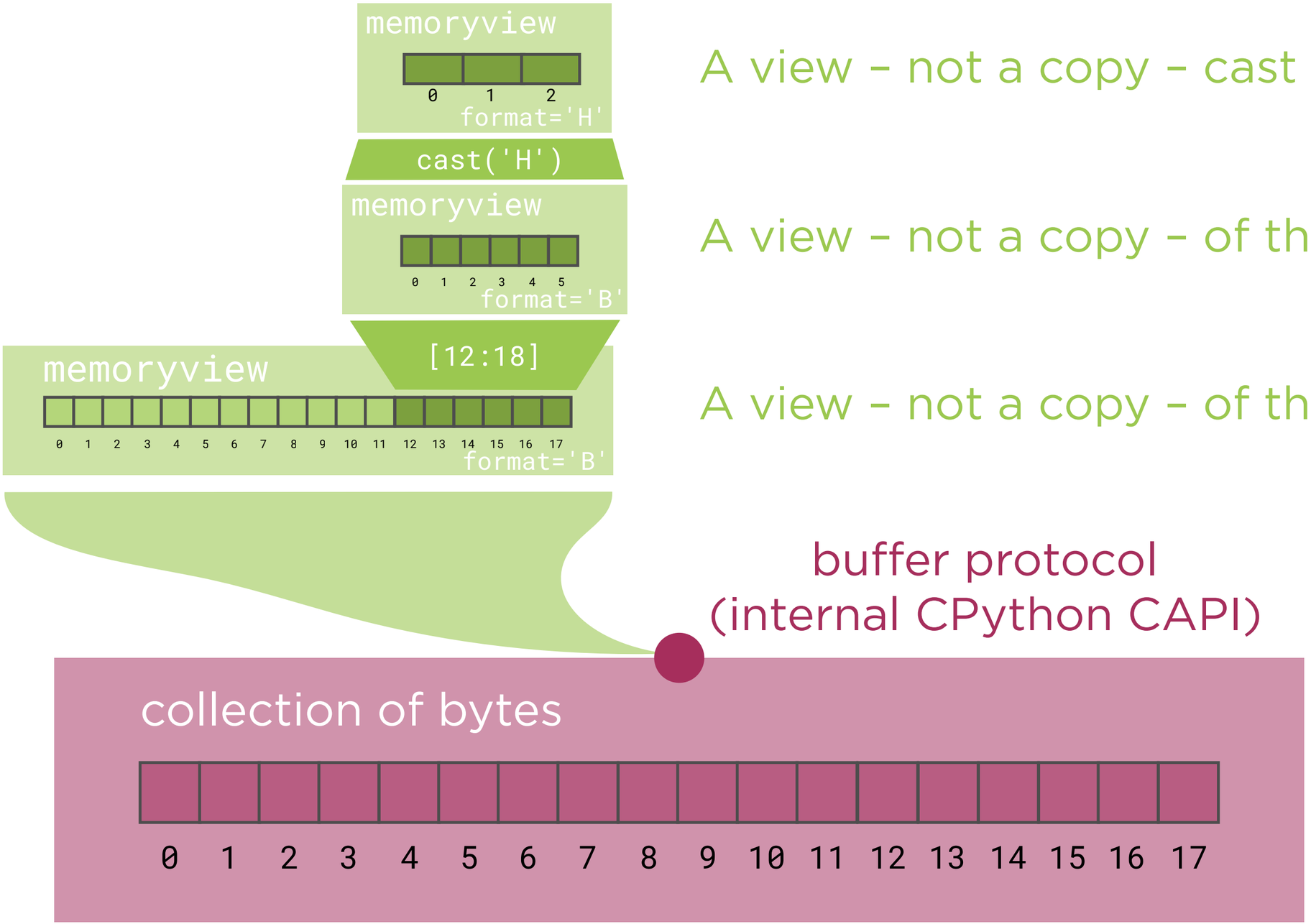

memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
format='B'
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
collection of bytes
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17



A view – not a copy – of the slice

A view – not a copy – of the byte buffer

buffer protocol
(internal CPython CAPI)



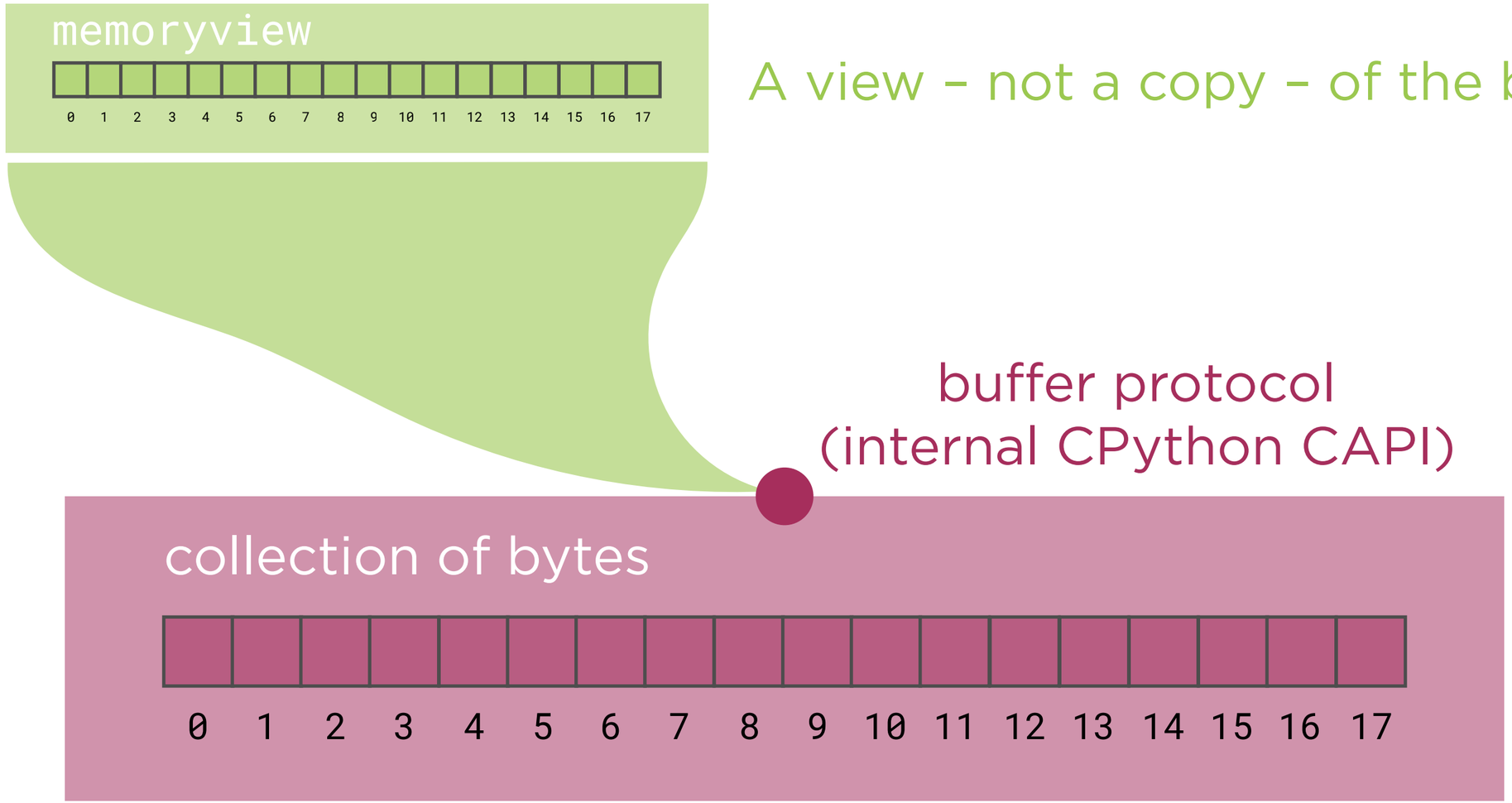
A view – not a copy – cast to integers

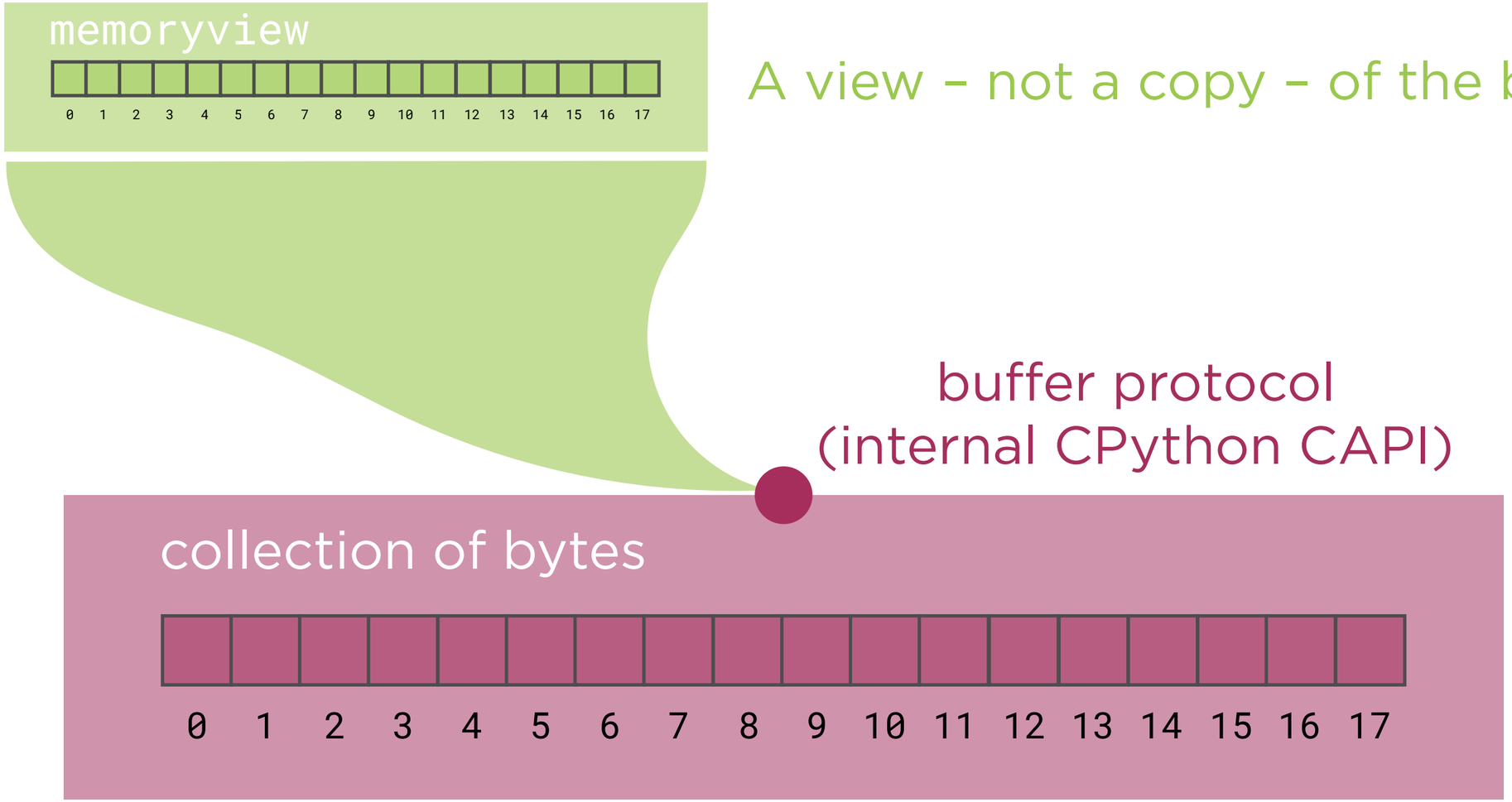
A view – not a copy – of the slice

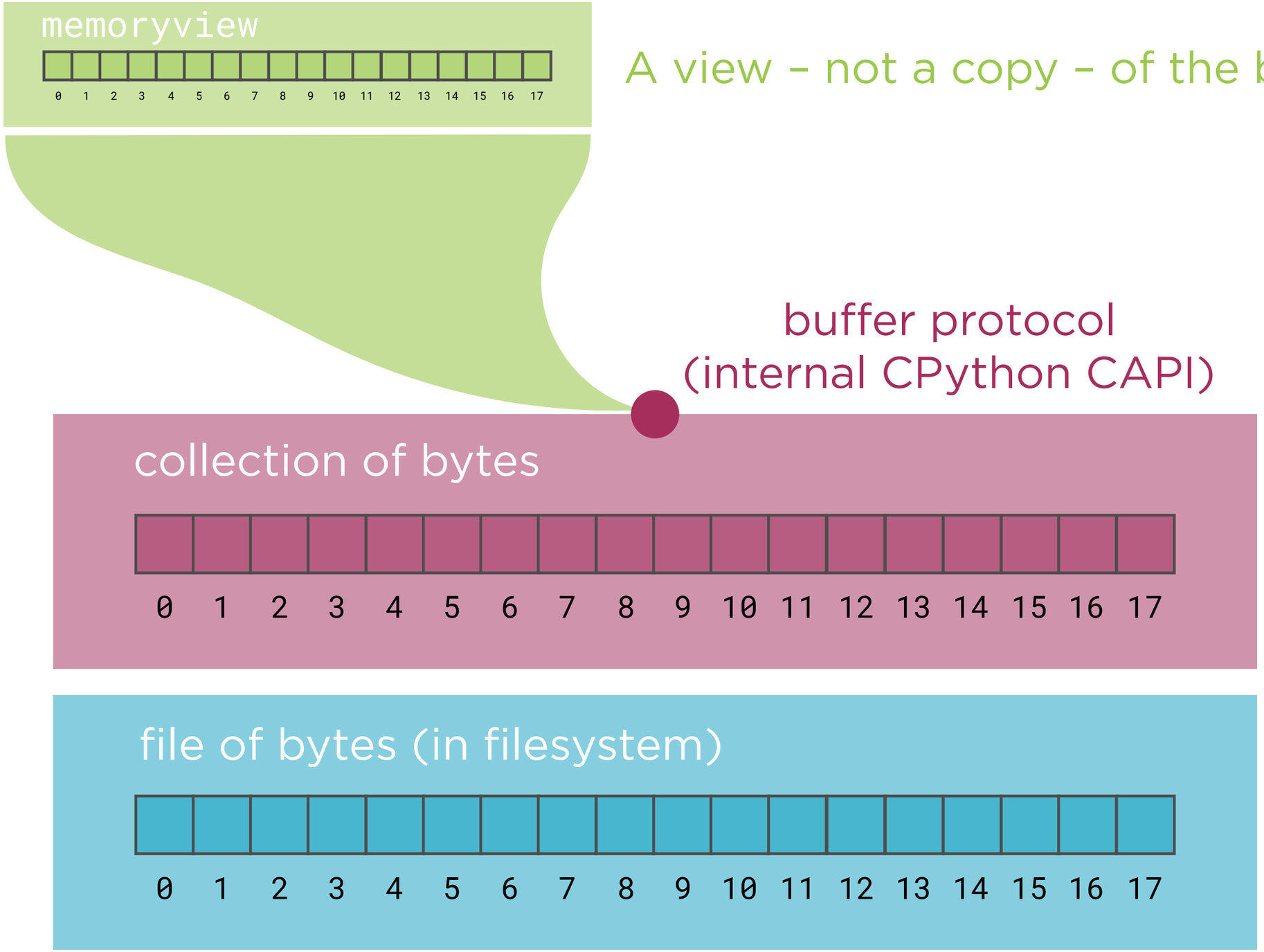
A view – not a copy – of the byte buffer

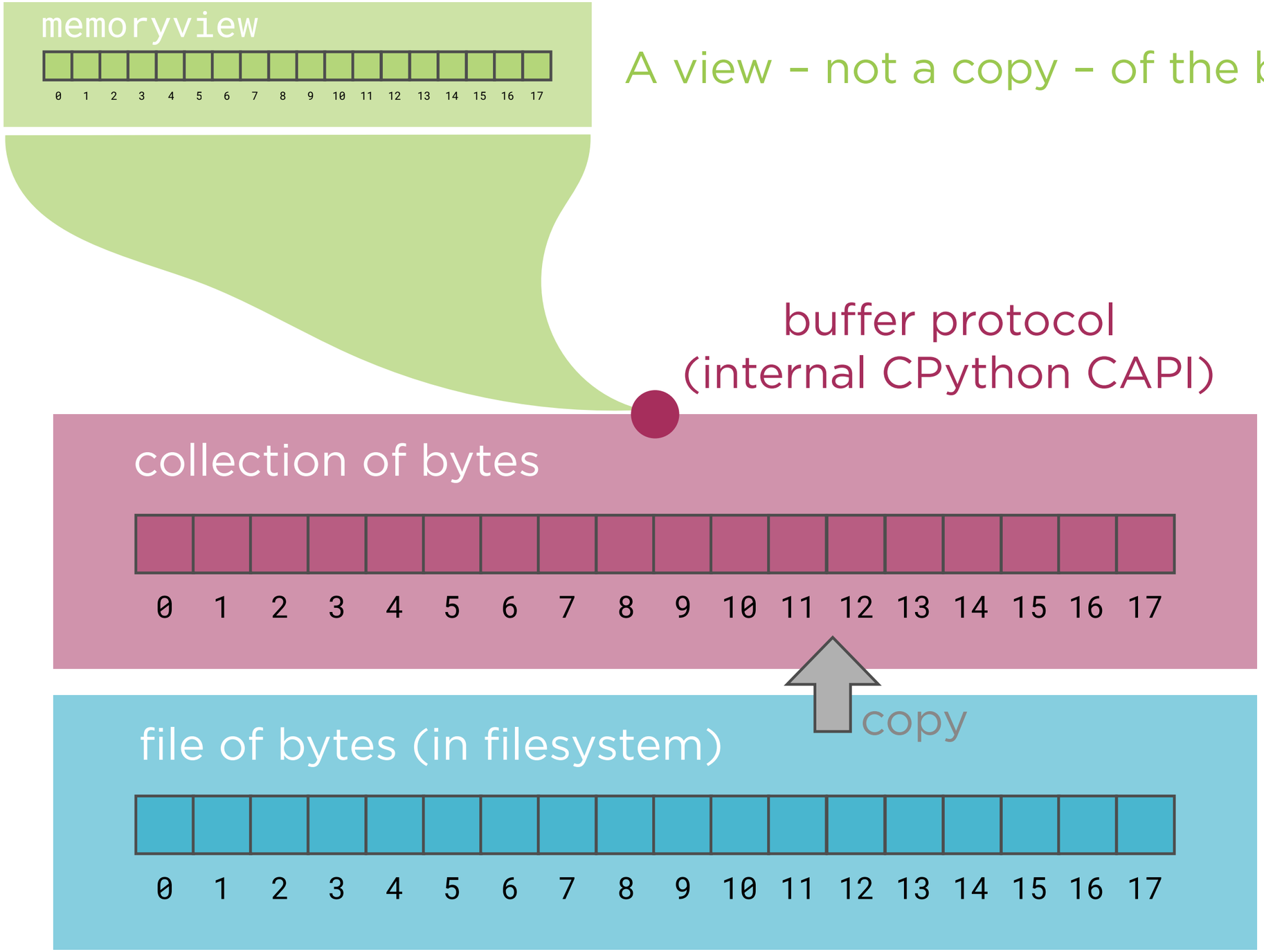
Memory-Mapped Files

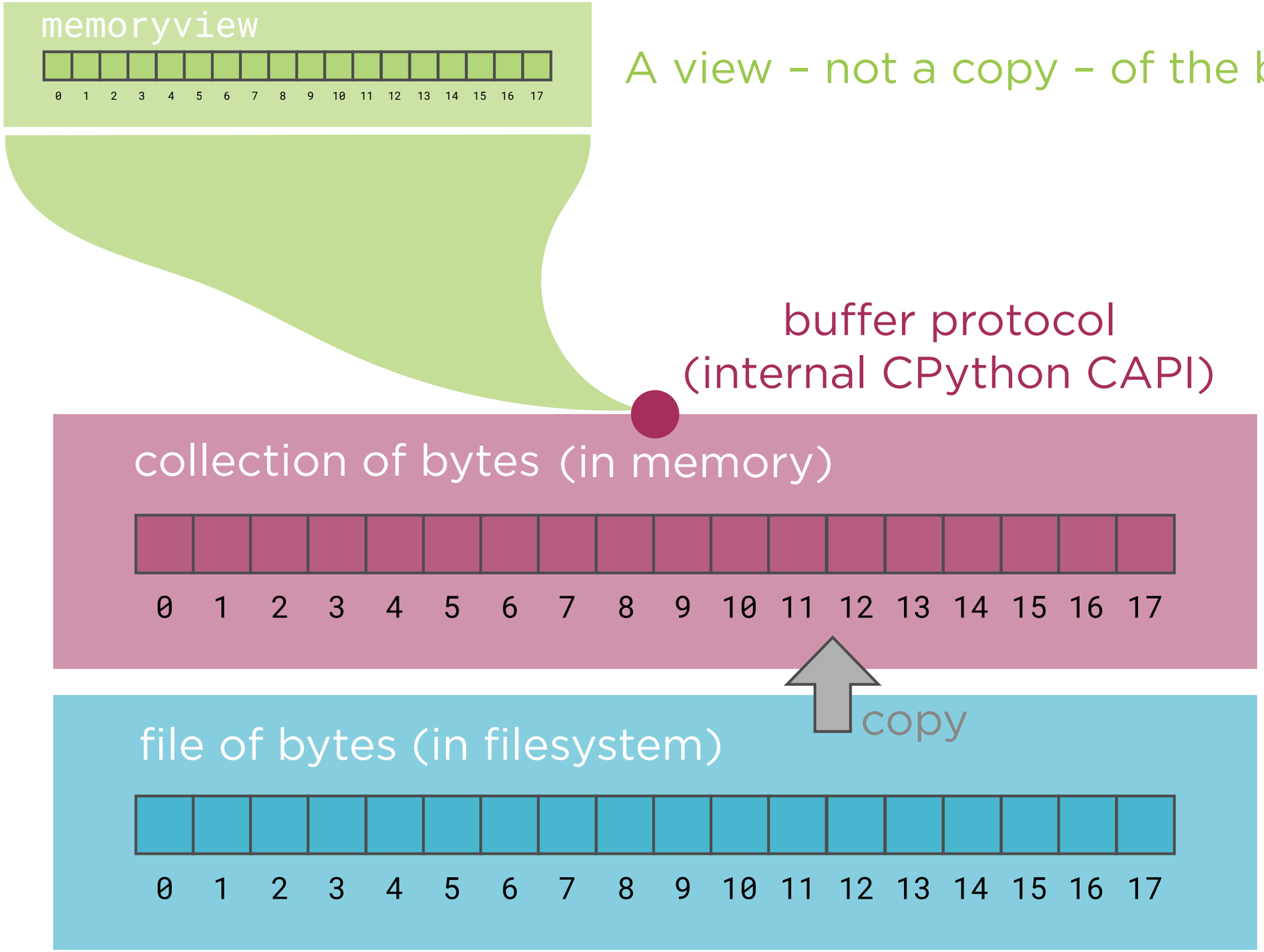
Memory views

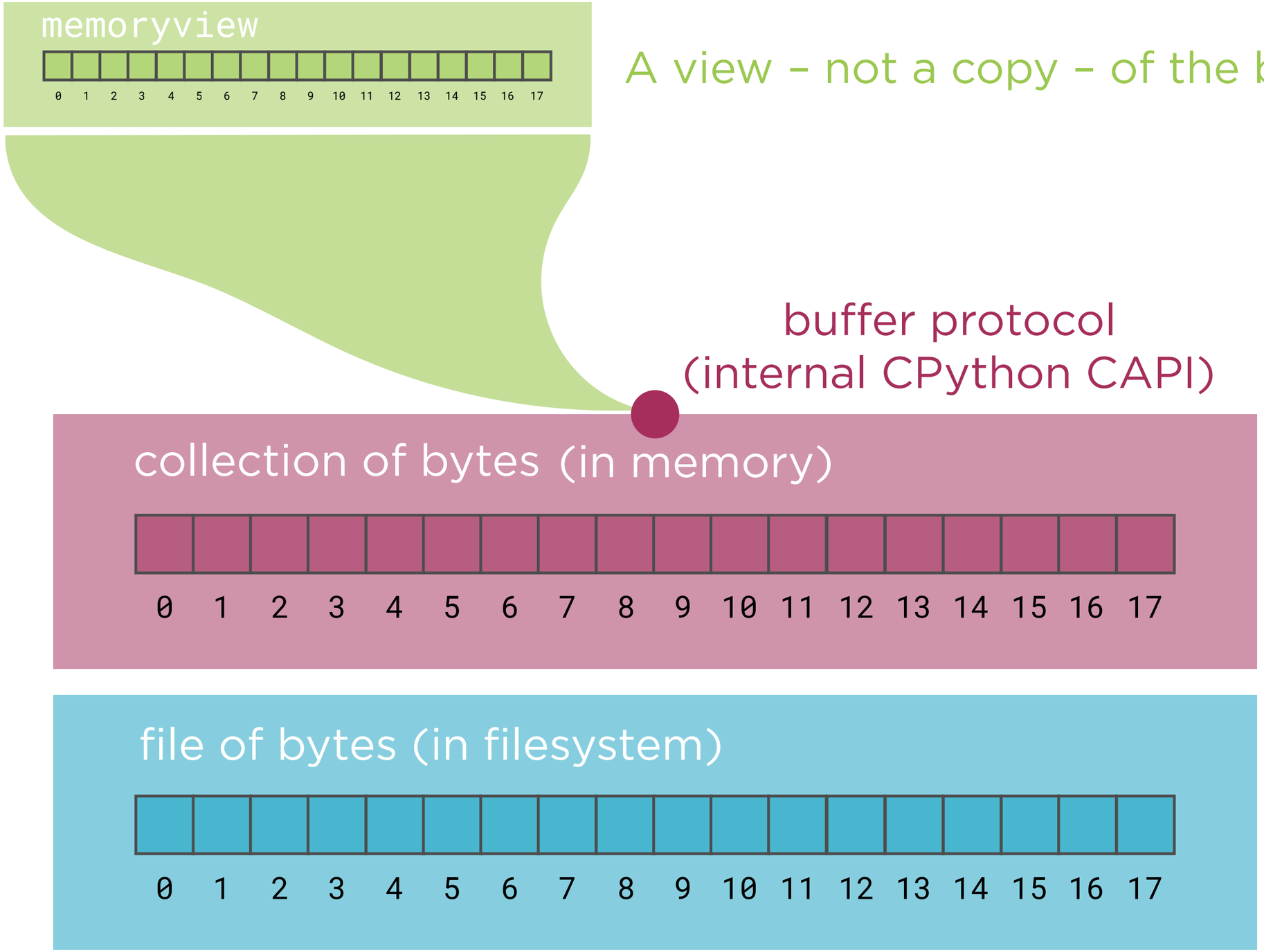


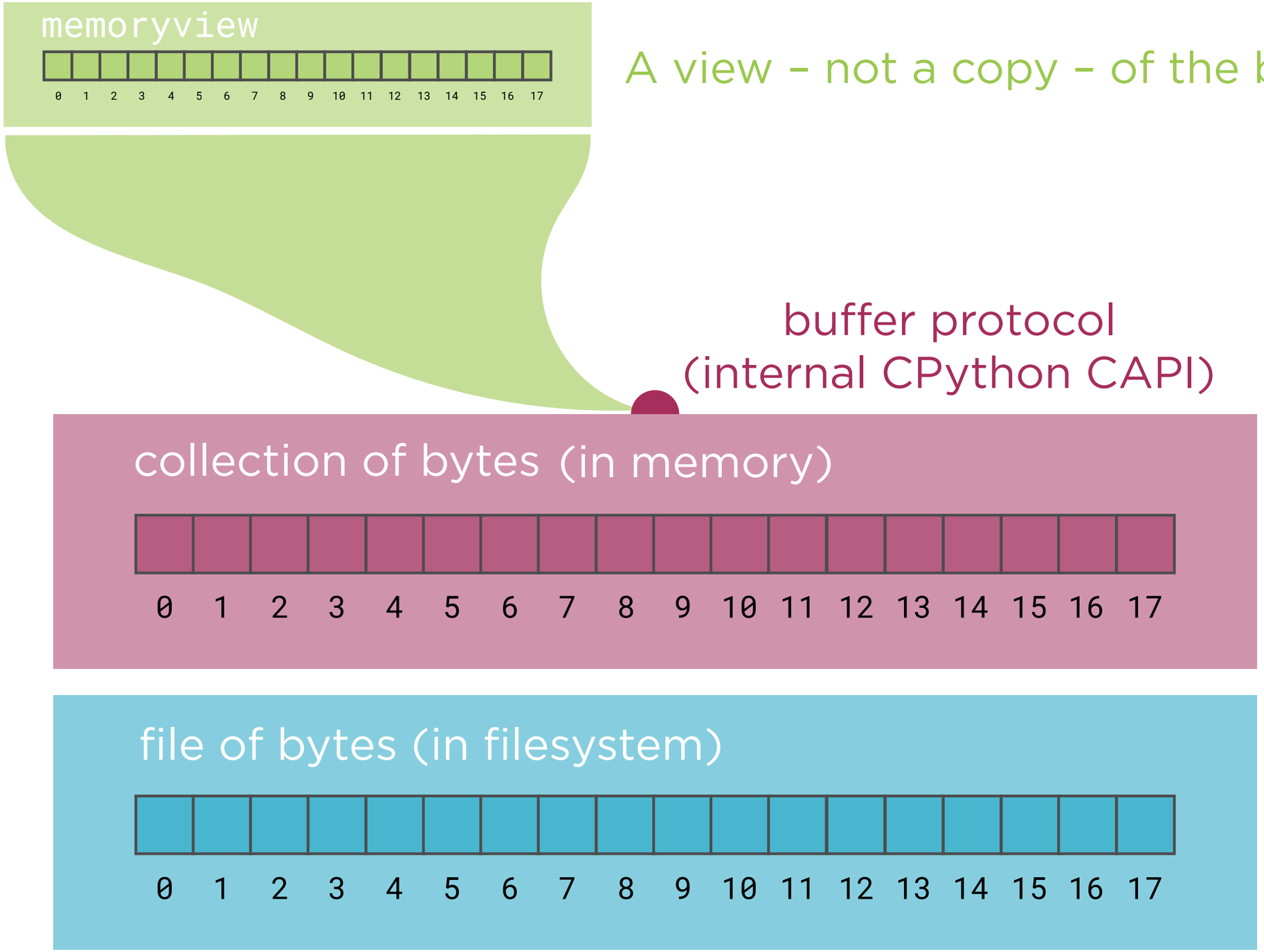

memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
collection of bytes
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

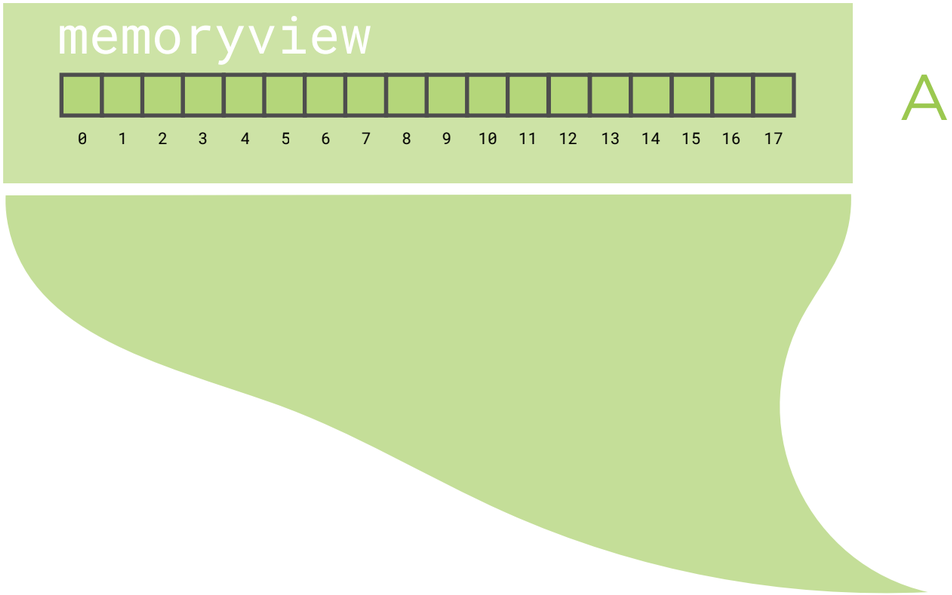

memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
collection of bytes
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
file of bytes (in filesystem)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17


memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
collection of bytes
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
copy
file of bytes (in filesystem)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

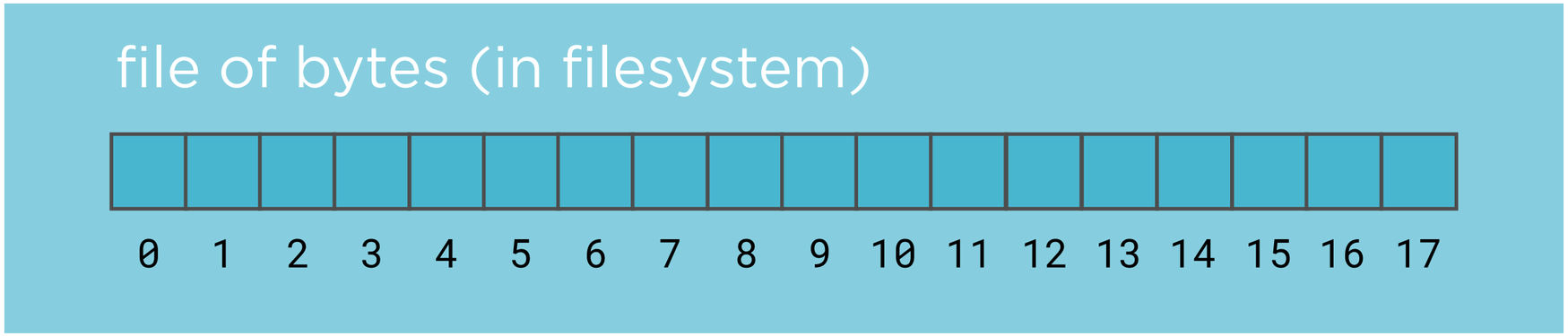

memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
collection of bytes (in memory)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
copy
file of bytes (in filesystem)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

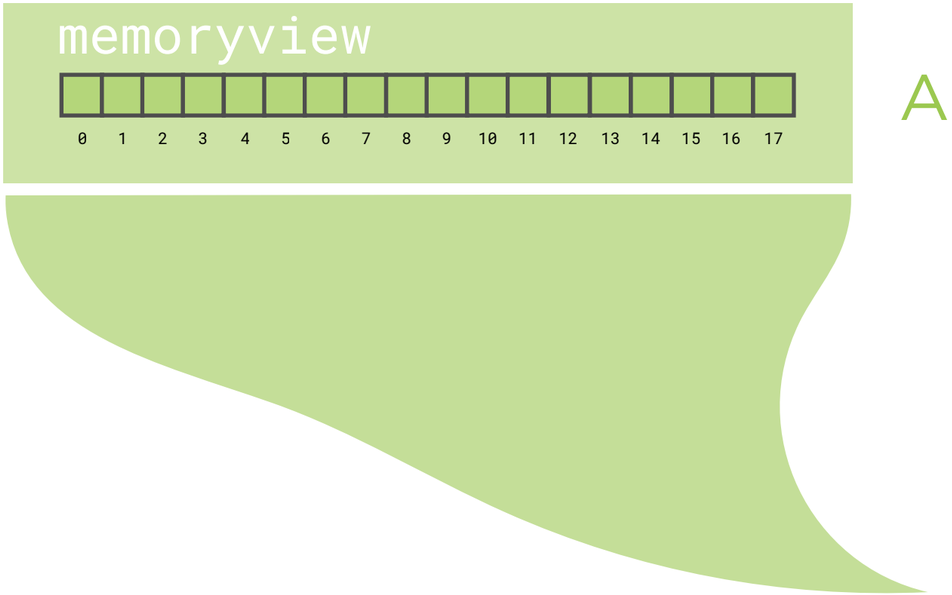

memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
collection of bytes (in memory)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
file of bytes (in filesystem)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17


memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
collection of bytes (in memory)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
file of bytes (in filesystem)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

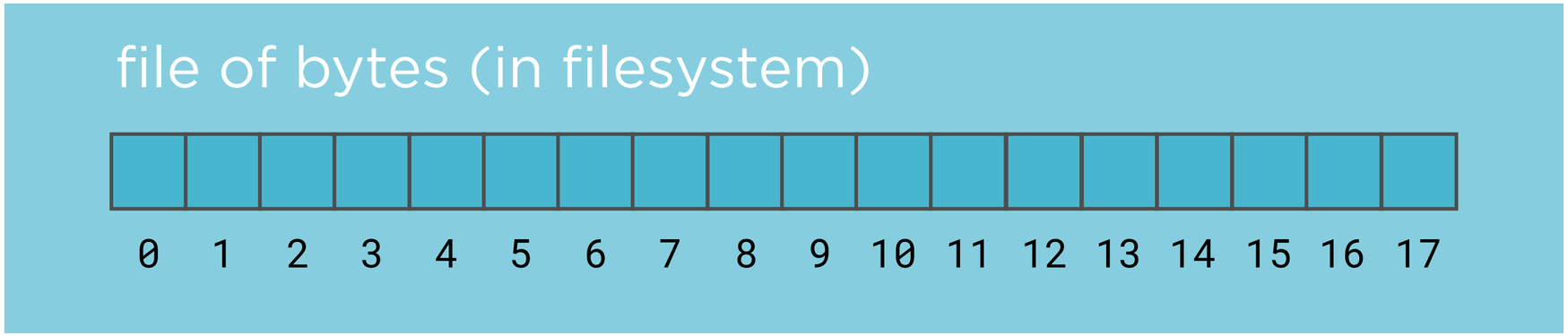


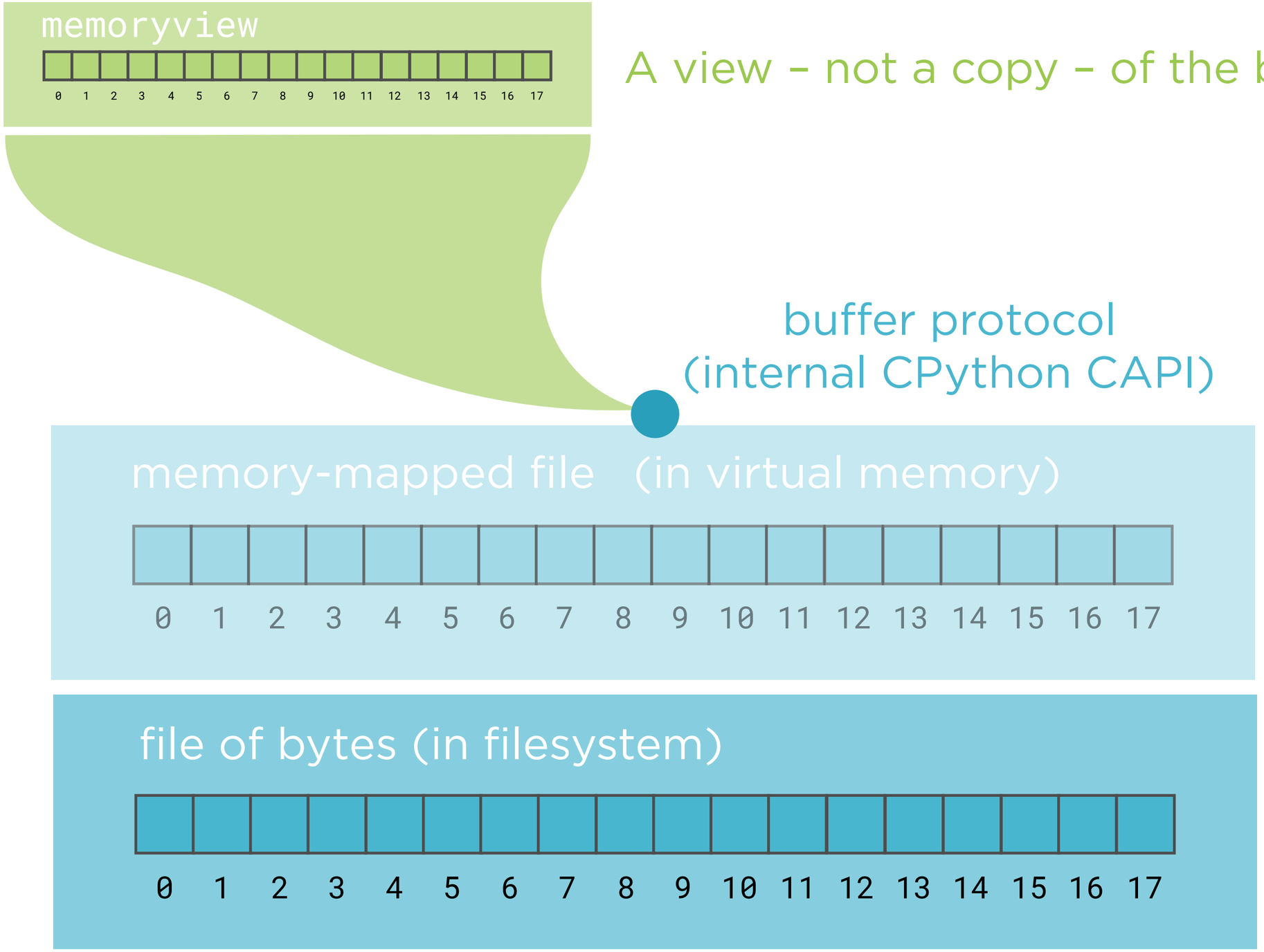
A view – not a copy – of the byte buffer

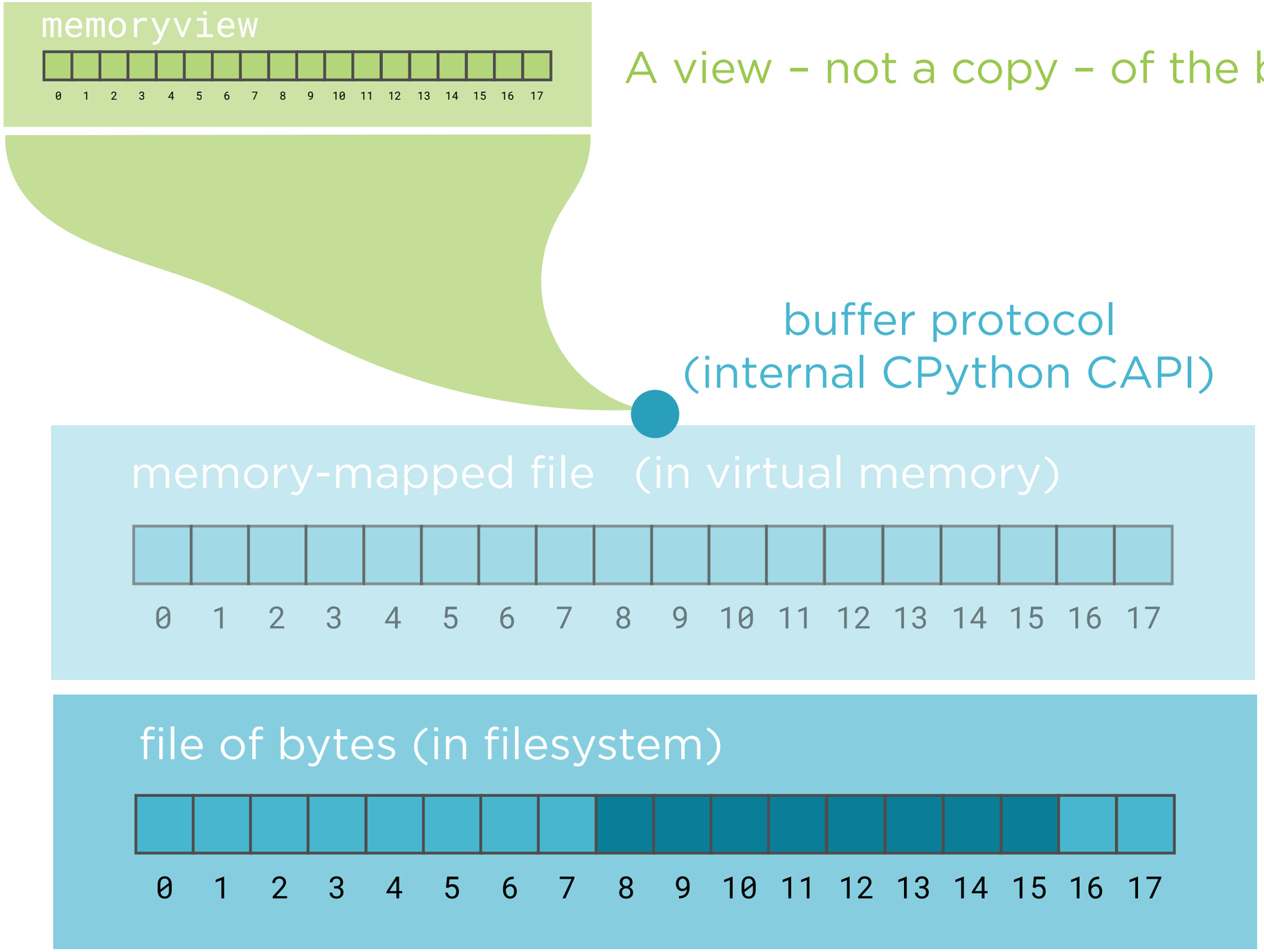


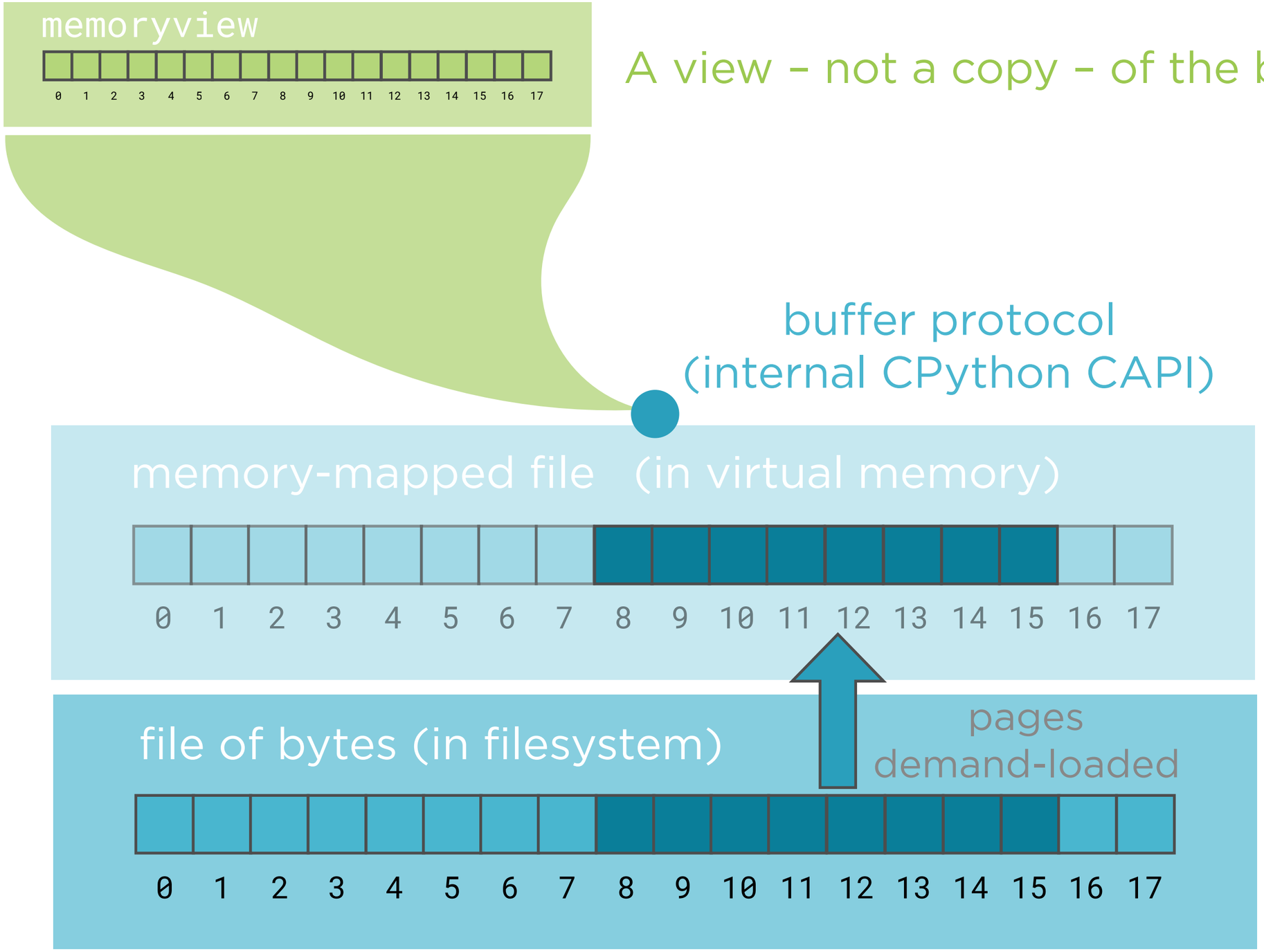


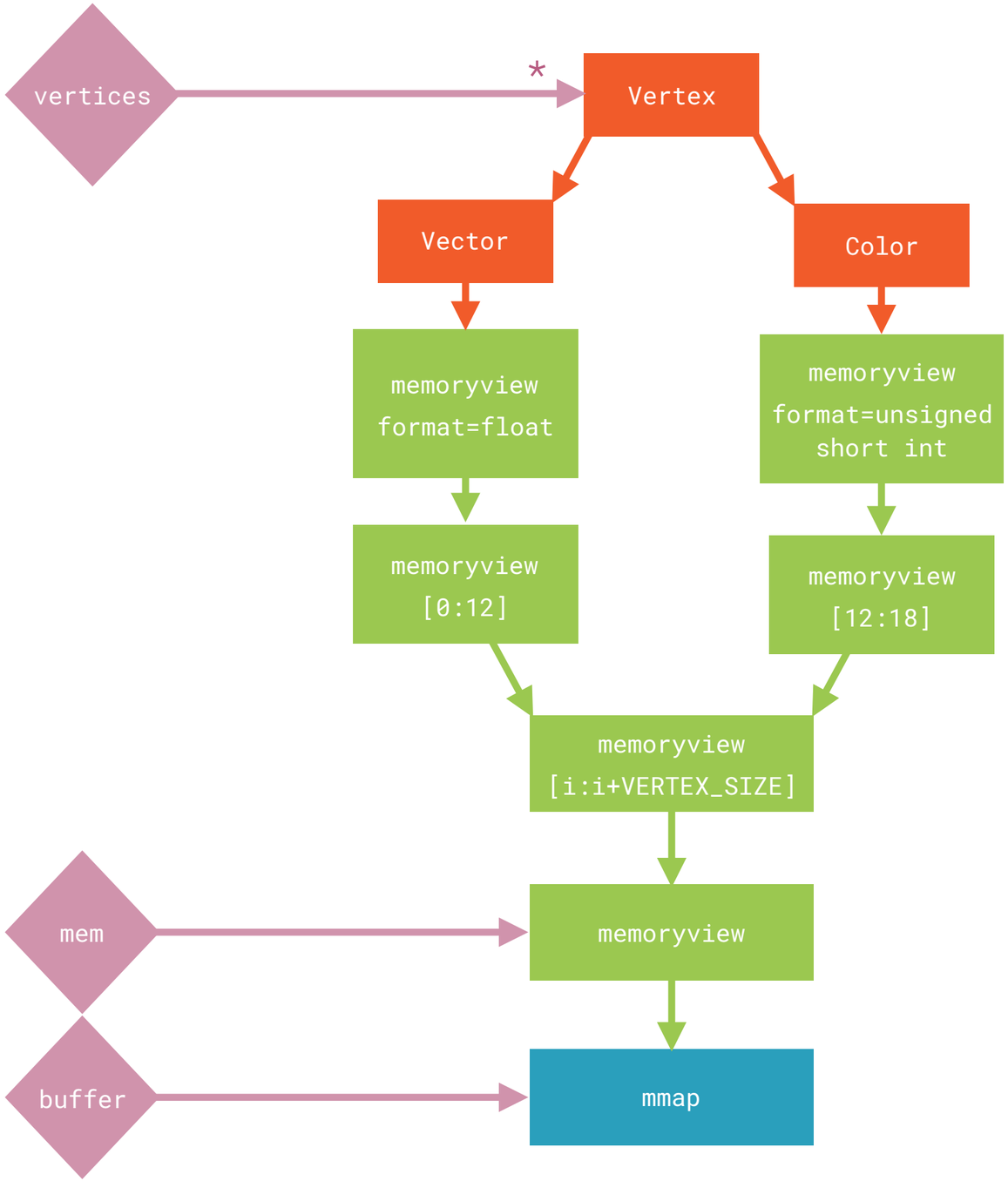
A view – not a copy – of the byte buffer

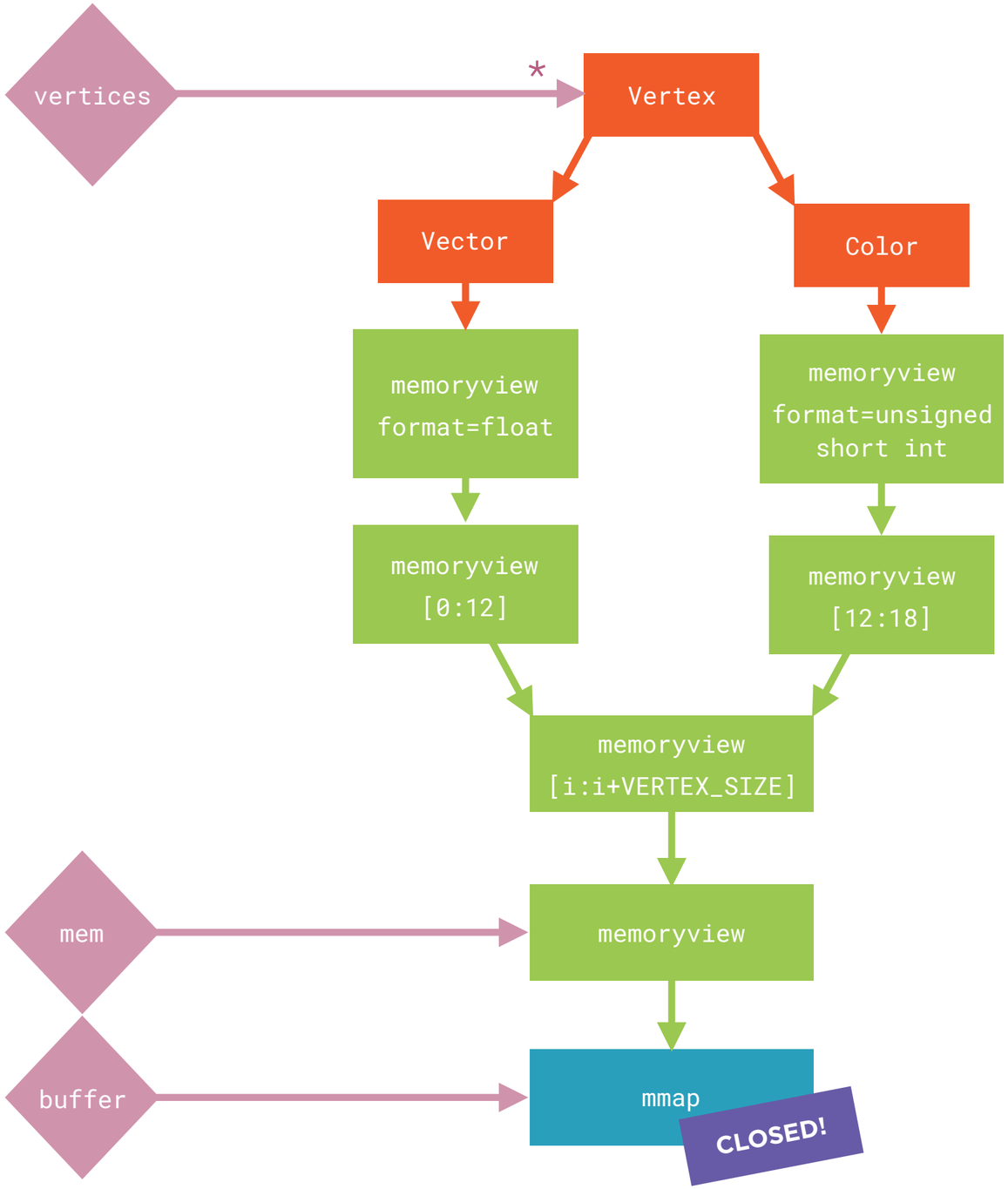


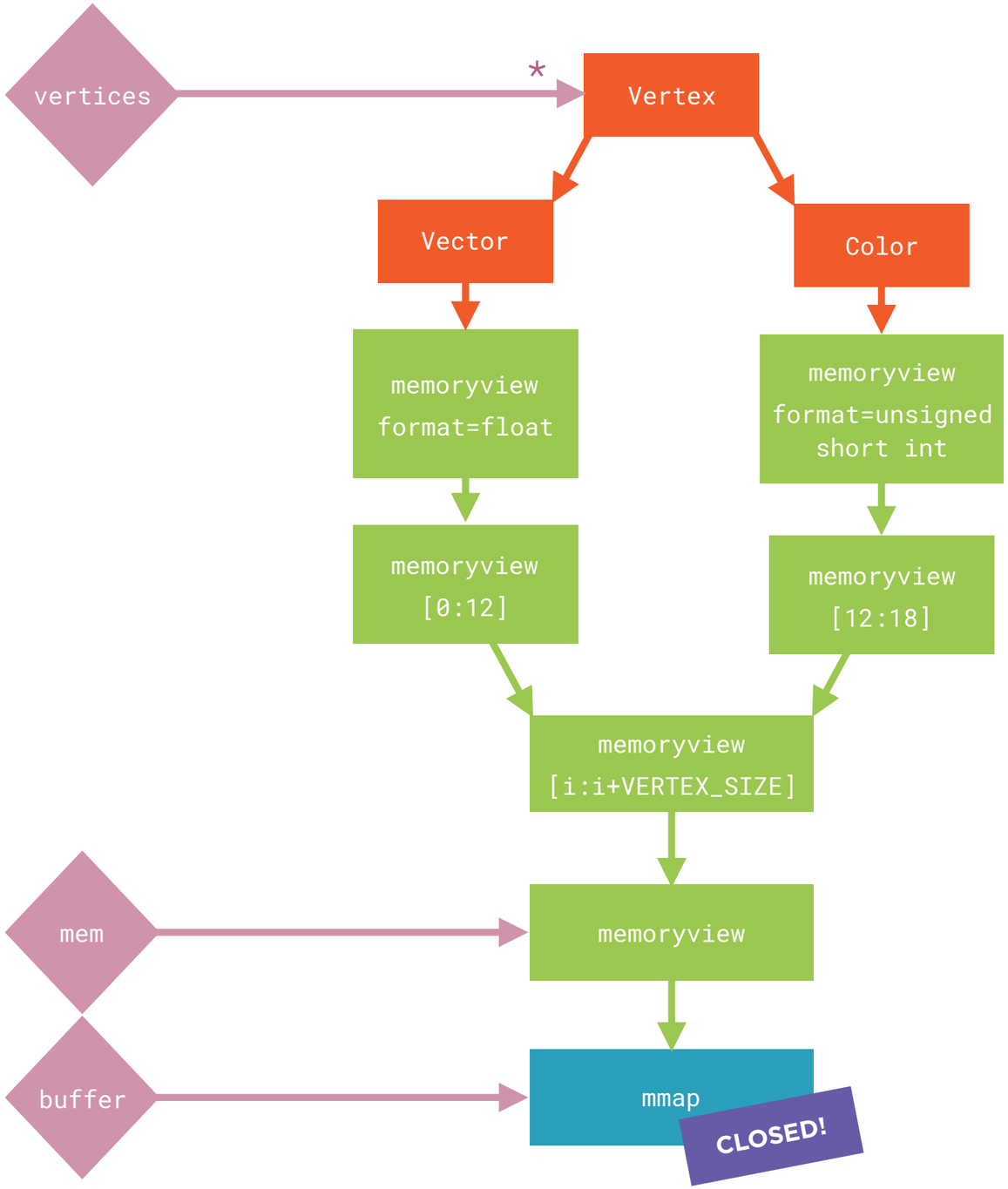

memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
memory-mapped file (in virtual memory)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
file of bytes (in filesystem)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

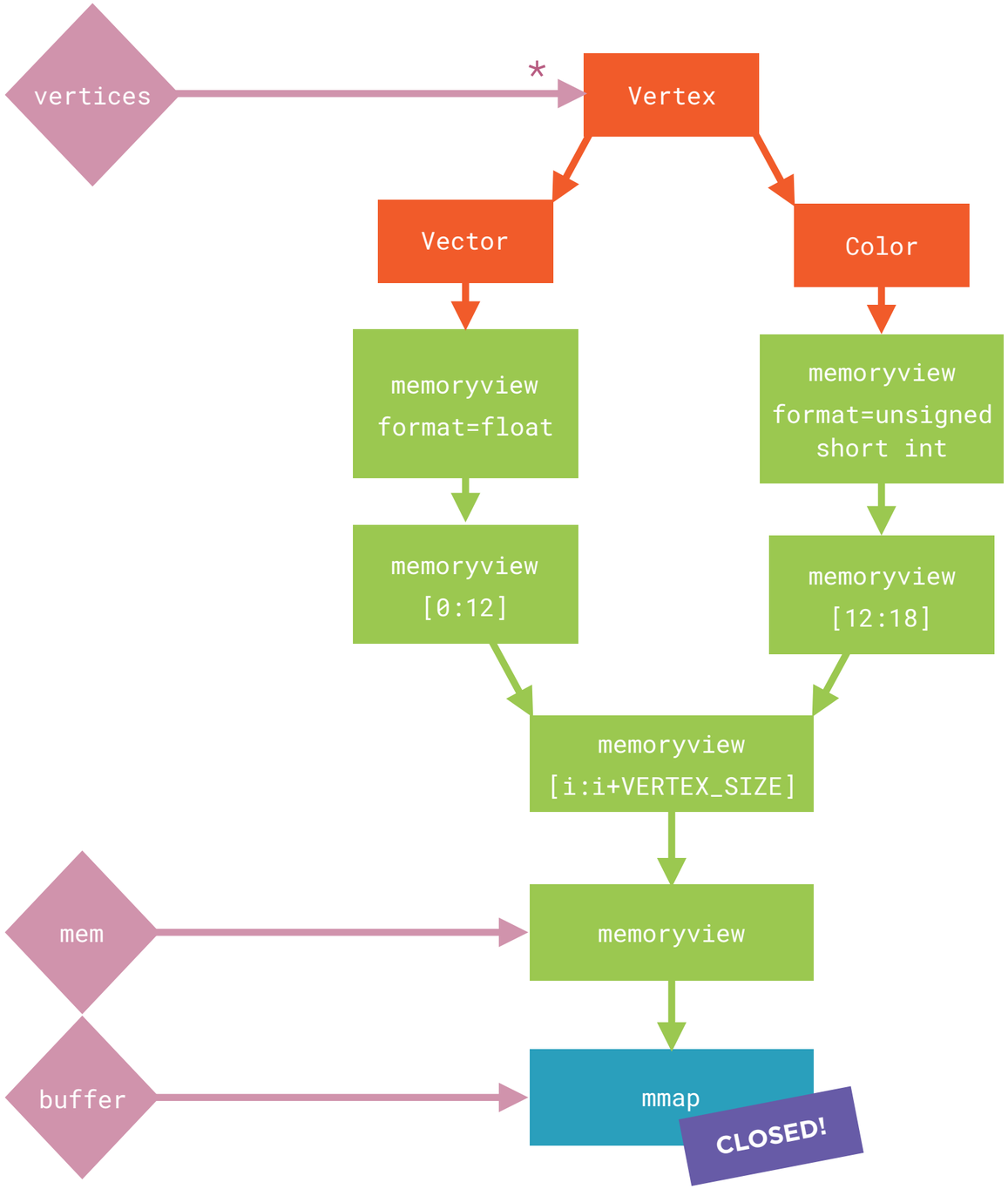

memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
memory-mapped file (in virtual memory)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
file of bytes (in filesystem)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

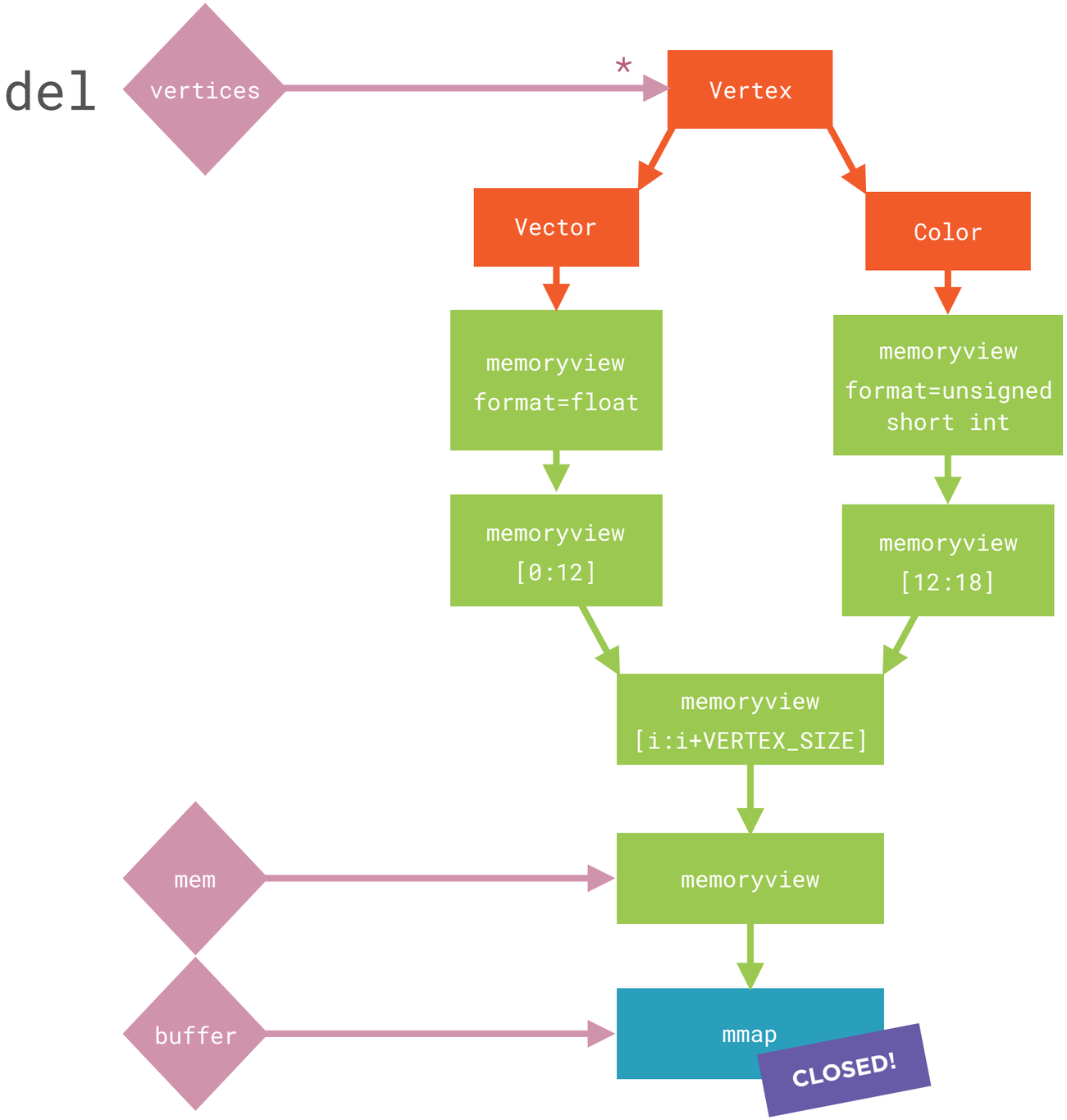

memoryview
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A view – not a copy – of the byte buffer
buffer protocol
(internal CPython CAPI)
memory-mapped file (in virtual memory)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
file of bytes (in filesystem)
pages
demand-loaded
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

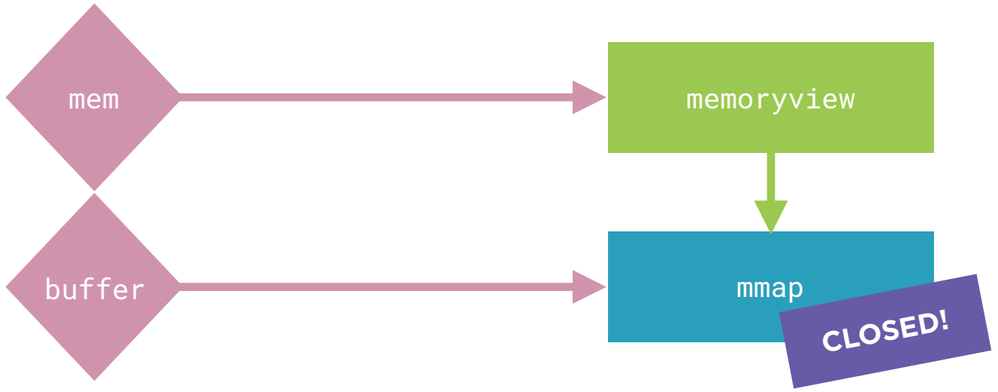

vertices
*
Vertex
Vector
Color
memoryview
format=float
memoryview
format=unsigned
short int
memoryview
[0:12]
memoryview
[12:18]
memoryview
[i:i+VERTEX_SIZE]
mem
memoryview
buffer
mmap

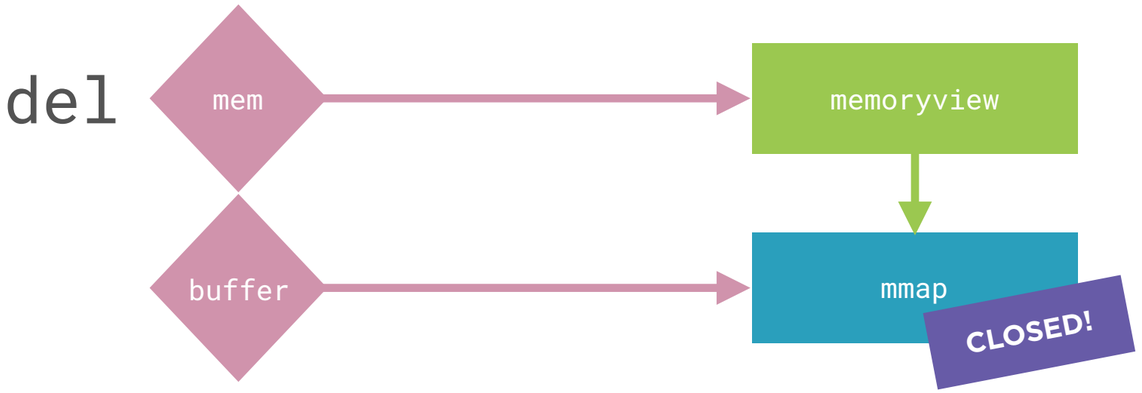

vertices
*
Vertex
Vector
Color
memoryview
format=float
memoryview
format=unsigned
short int
memoryview
[0:12]
memoryview
[12:18]
memoryview
[i:i+VERTEX_SIZE]
mem
memoryview
buffer
mmap
CLOSED!

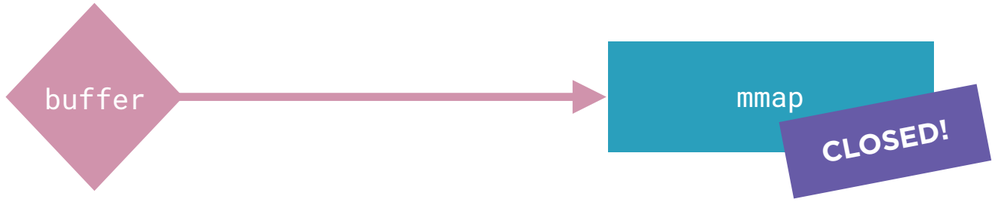

vertices
*
Vertex
Vector
Color
memoryview
format=float
memoryview
format=unsigned
short int
memoryview
[0:12]
memoryview
[12:18]
memoryview
[i:i+VERTEX_SIZE]
mem
memoryview
buffer
mmap
CLOSED!


vertices
*
Vertex
Vector
Color
memoryview
format=float
memoryview
format=unsigned
short int
memoryview
[0:12]
memoryview
[12:18]
memoryview
[i:i+VERTEX_SIZE]
mem
memoryview
buffer
mmap
CLOSED!


del
vertices
*
Vertex
Vector
Color
memoryview
format=float
memoryview
format=unsigned
short int
memoryview
[0:12]
memoryview
[12:18]
memoryview
[i:i+VERTEX_SIZE]
mem
memoryview
buffer
mmap
CLOSED!


mem
buffer
memoryview
mmap
CLOSED!


del
mem
buffer
memoryview
mmap
CLOSED!


buffer
mmap
CLOSED!

Summary

Summary

Summary

Bitwise XOR ^

Summary

Bitwise XOR ^

Bitwise NOT ~

Summary

Bitwise XOR ^

Bitwise NOT ~

Two's complement integers

Summary

Bitwise XOR ^

Bitwise NOT ~

Two's complement integers

bytes – immutable binary sequence

Summary

Summary

Summary

Interpret bytes using struct unpacking

Summary

Interpret bytes using struct unpacking

Display bytes using hexlify

Summary

Interpret bytes using struct unpacking

Display bytes using hexlify

C alignment awareness

Summary

Interpret bytes using struct unpacking

Display bytes using hexlify

C alignment awareness

memoryview – views, slices, casts

Summary

Interpret bytes using struct unpacking

Display bytes using hexlify

C alignment awareness

memoryview – views, slices, casts

code.interact() – drop to REPL