Operators and Conversions Between Data Types

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- Arithmetic and Assignment Operators
- Data Types and Literal Assignment
- Implicit Conversions
- Explicit Conversions

More Variable Declaration

- Can declare multiple variables on one line:

```
double length, depth, height;
```

comma

equivalent

```
double length;
double depth;
double height;
```

- Can combine declarations and initializations:

```
int age = 30, weight, votes;
```

equivalent

```
int age = 30;
int weight;
int votes;
```

Doing Arithmetic

- C# has math operators for numeric data

Operation	C# Operator	Algebraic Expression	C# Expression
Addition	+	$x + 7$	`myVar + 7`
Subtraction	–	$x - 7$	`myVar - 7`
Multiplication	*	$x \cdot 7$	`myVar * 7`
Division	/	$\frac{x}{7}, x/7, x \div 7$	`myVar / 7`
Remainder	%	$x \bmod 7$	`myVar % 7`

Remainder after integer division:
$44 \bmod 7 = 2$
because $44 \div 7 = 6$
with remainder 2

Arithmetic and Assignment

- Result of an arithmetic **expression** is a numeric value
- Numeric values can be assigned to variables

```
int myVar = 3 * 4;
```

The value 12 is stored in myVar

This expression evaluates to 12

- Type of variable must match type of expression

```
double goodSum = 4.5 + 6.4;
```

This expression evaluates to 10.9

```
float badSum = 4.5 + 6.4;
```

```
float floatSum = 4.5f + 6.4f;
```

Arithmetic and Variables

- Variable in arithmetic expression = read its current value

```
int a = 4;
int b = a + 5;
a = b * 2;
b = a - 20;
```

a has value 4, so b gets the resulting value 9

b has value 9, now a is assigned new value 18

a has value 18, now b is assigned new value -2

```
float x = 30.0f;
float y = x / 8f;
x = 2f * y + 0.5f;
float z = y * 1.5f;
```

x has value 30, so y gets the value 3.75

y has value 3.75, x is assigned new value 8

y has value 3.75, z is assigned 5.625

Self-Assignment

- A variable can appear on both sides of the = operator

What does this do?

```
int myVar = 4;
myVar = myVar * 2;
```

Store result of 4 * 2 into myVar

Read myVar's current value, 4

- Entire right side is evaluated before doing assignment

```
float x = 30.0f;
float y = x / 8f;
x = 2f * x + 0.5f;
y = y * 1.5f + y;
```

y gets the value 3.75, as before

x is assigned new value 60.5

y is assigned new value 9.375

y has value 3.75 both times

Self-Assignment Shortcuts

- Compound assignment operators: combine arithmetic and assignment
- Entire right side is evaluated before compound operator

Statement	Equivalent
`x += 2;`	`x = x + 2;`
`x -= 2;`	`x = x – 2;`
`x *= 2;`	`x = x * 2;`
`x /= 2;`	`x = x / 2;`

```
int myVar = 4;
myVar *= 2;
myVar += 6 * -3;
```

myVar changes to 8 like before

myVar gets new value -10

Result of expression is -18

- Compound operator can only replace one arithmetic operator

```
x = 2f * x + 0.5f;
```



```
x *= 2f;
x += 0.5f;
```

Outline

- Arithmetic and Assignment Operators
- **Data Types and Literal Assignment**
- Implicit Conversions
- Explicit Conversions

Review: Numeric Data Types

- Integers

Size & range ↓

Type	Size	Range of Values		Type	Size	Range of Values
`short`	2 bytes	$-2^{15} \dots 2^{15}-1$	↔	`ushort`	2 bytes	$0 \dots 2^{16}-1$
`int`	4 bytes	$-2^{31} \dots 2^{31}-1$	↔	`uint`	4 bytes	$0 \dots 2^{32}-1$
`long`	8 bytes	$-2^{63} \dots 2^{63}-1$	↔	`ulong`	8 bytes	$0 \dots 2^{64}-1$

- Floating-point

Type	Size	Range of Values	Digits of Precision
`float`	4 bytes	$\pm 1.5 \cdot 10^{-45} \dots \pm 3.4 \cdot 10^{38}$	7
`double`	8 bytes	$\pm 5.0 \cdot 10^{-324} \dots \pm 1.7 \cdot 10^{308}$	15-16
`decimal`	16 bytes	$\pm 1.0 \cdot 10^{-28} \dots \pm 7.9 \cdot 10^{28}$	28-29

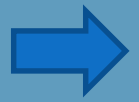
Assignment From Literals

- If literal type matches variable type, assignment always works:

```
int myAge = 29;
double myHeight = 1.77;
float radius = 2.3f;
```

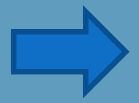
- What if literal type is different?

```
float radius = 2.3;
```



Error! Can't convert `double` to `float`

```
float radius = 2;
```

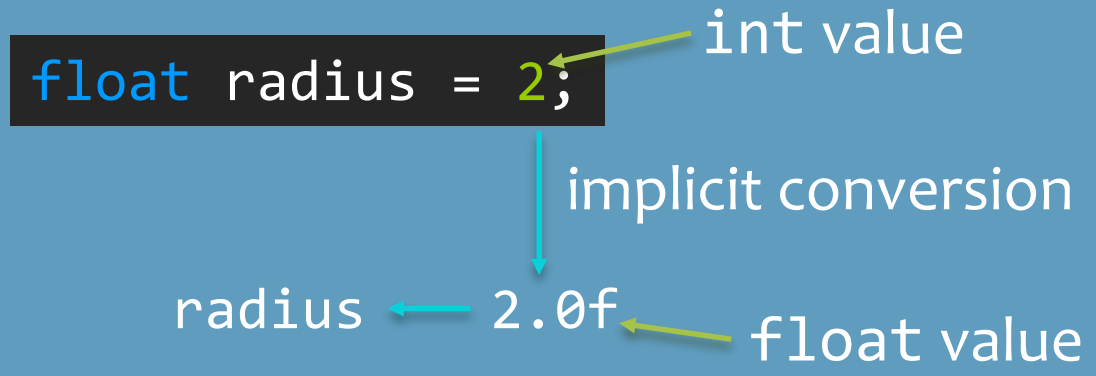


No error, even though 2 is an `int` literal

Why does this work?

Implicit Conversions

- Value type must still match variable type
- Some types can be **implicitly converted** to others:



- Also applies to assignment from variables:

```
int length = 2;
float radius = length;
```

value 2 implicitly converted to `2.0f`

gets value `2.0f`

Implicit Conversions

Type	Possible Implicit Conversions
short	int, long, float, double, decimal
int	long, float, double, decimal
long	float, double, decimal
ushort	uint, int, ulong, long, decimal, float, double
uint	ulong, long, float, double, decimal
ulong	float, double, decimal
float	double

What's the pattern here? Given a type, what can you implicitly convert it to?

Implicit Conversions are "Safe"

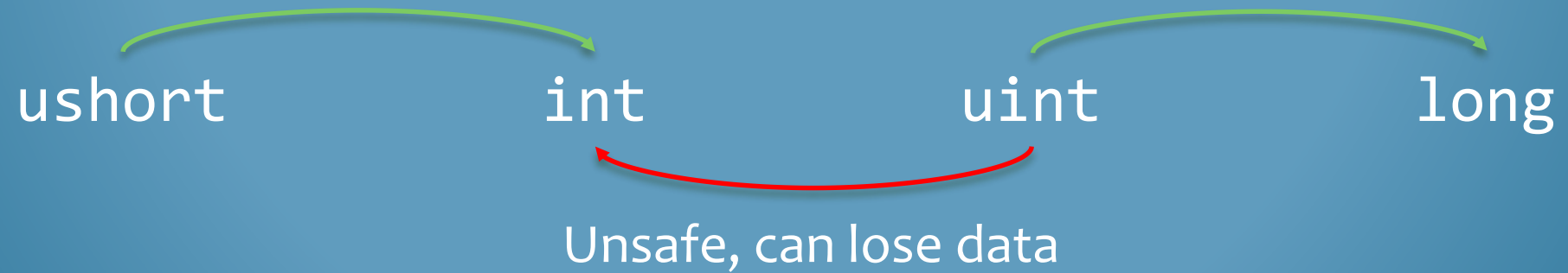
- `int` range: $-2^{31} \ldots 2^{31} - 1$; `float` integer range: $\pm 3.4 \cdot 10^{38}$
- Any `int` can be stored in a `float` **without losing data**
- Reverse is not safe: Storing `4.7f` in an `int` will lose the fraction

- All integer types are safe to convert to `float` or `double`

Other Safe Conversions

- Smaller integer to larger integer; `float` to `double`

- Unsigned to *larger* signed integer (why larger?)

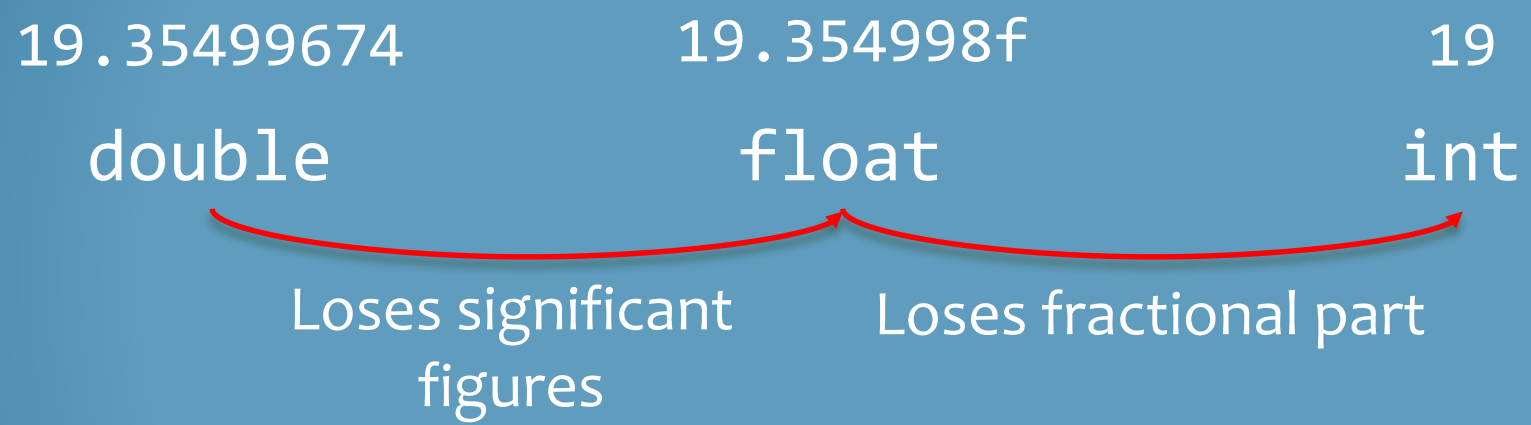


Outline

- Arithmetic and Assignment Operators
- Data Types and Literal Assignment
- Implicit Conversions
- **Explicit Conversions**

Data-Losing Conversions

- Unsafe conversions: Potential to lose data



- Will not happen automatically; compile error

```
double length = 2.886;
float radius = length;
```

Error! Can't convert `double` to `float`

Explicit Conversion with Casts

- Cast operator: Force the compiler to allow an unsafe conversion

```
double length = 2.886;
float radius = (float) length;
```

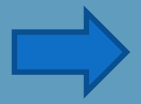
gets value `2.886f`

Target type name in parentheses

`double` value

- Explicit conversion from original to target type must exist
  - Most built-in C# types have explicit conversions defined

```
string strAge = "29";
int myAge = (int) strAge;
```



Error! Can't convert `string` to `int`

Casting Side-Effects

- Casting from floating-point to integer: fraction is *truncated*

```
int intLength = (int) length;
```

gets value 2

cast to int

value: 2.886

- Casting to less precise floating-point: fraction is *rounded*

```
decimal myDecimal = 123456789.999999918m;
double myDouble = (double) myDecimal;
float myFloat = (float) myDouble;
```

myDouble: 123456789.99999993

myFloat: 123456790.0f

Floating-Point Casts and Range

- `float` is less precise than `double`, but also has a smaller range

Type	Size	Range of Values	Digits of Precision
`float`	4 bytes	$\pm 1.5 \cdot 10^{-45} \ldots \pm 3.4 \cdot 10^{38}$	7
`double`	8 bytes	$\pm 5.0 \cdot 10^{-324} \ldots \pm 1.7 \cdot 10^{308}$	15-16

- What if the `double` value is outside the range?

```
double myDouble = 3.5e-300;
float myFloat = (float) myDouble;
```

myFloat: `0f`

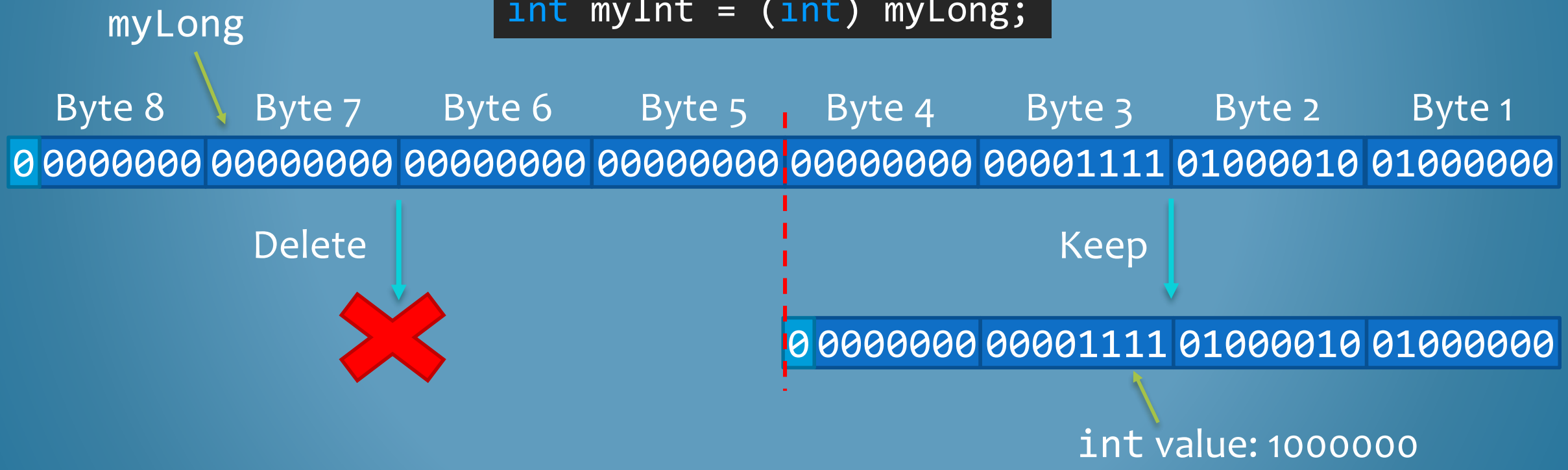
```
double myDouble = 6.2e57;
float myFloat = (float) myDouble;
```

myFloat: `Infinity`

Casting Side-Effects

- Casting to smaller integer: Most significant *bits* are truncated

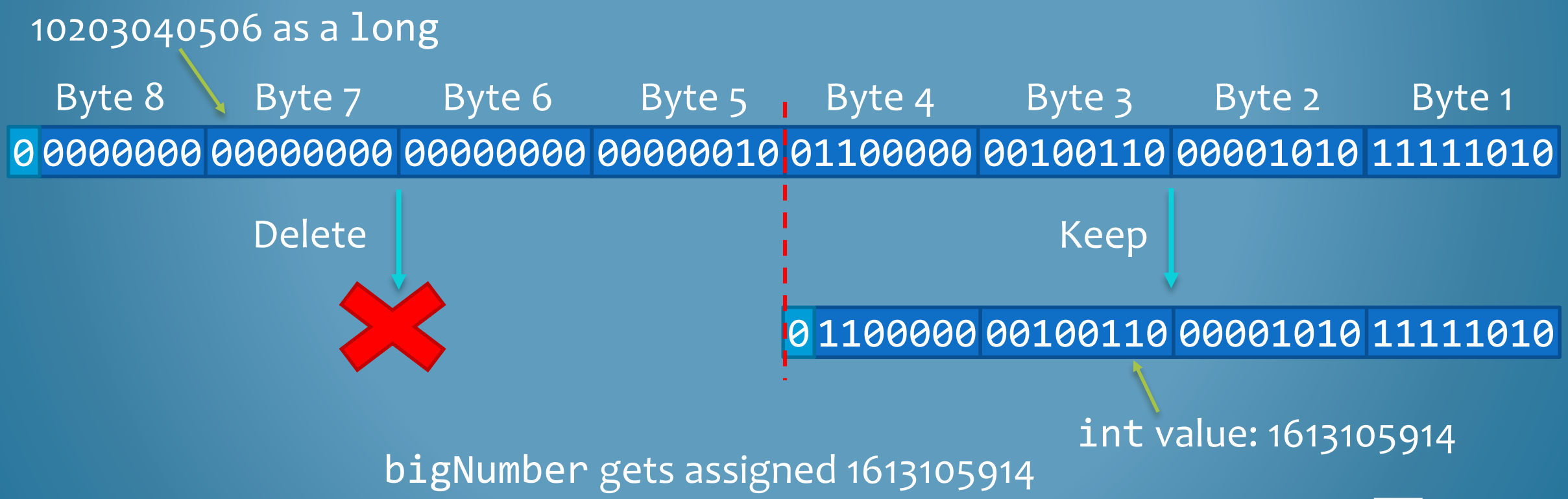
```
long myLong = 1000000;
int myInt = (int) myLong;
```



Casting Side-Effects

- Caution: Truncating **bits** will not "round down" in decimal

```
int bigNumber = (int) 10203040506;
```

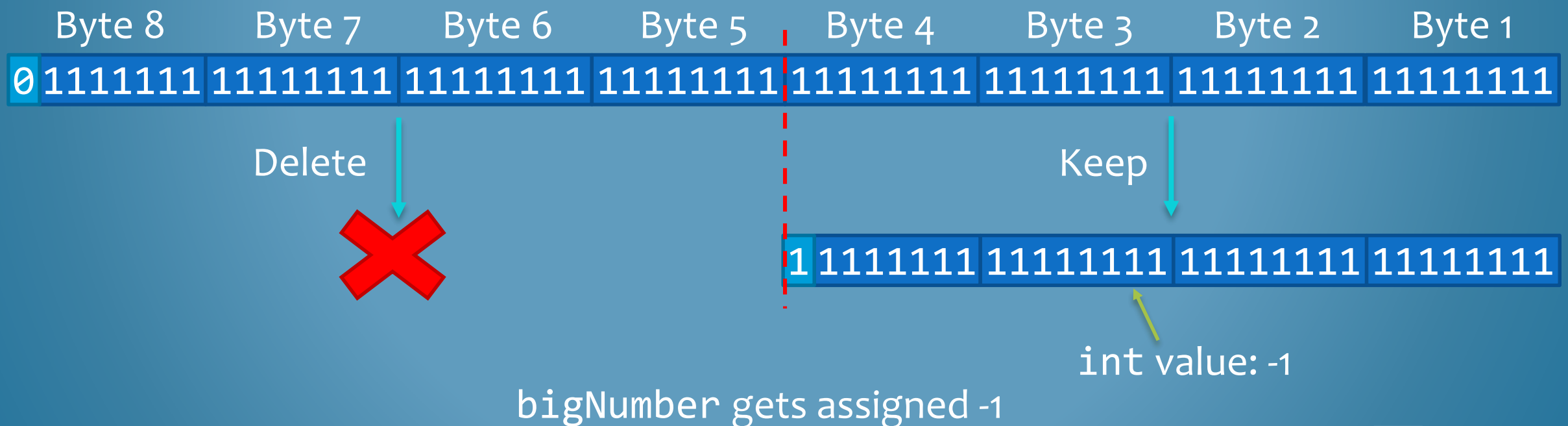


Casting Side-Effects

- Caution: Highest bit of truncated number becomes the sign bit

```
int bigNumber = (int) 9223372036854775807;
```

long maximum: $2^{63} - 1$



Casting Side-Effects

- Casting to `decimal`: Stored precisely, *unless* it is out of range – crashes with a `System.OverflowException`

```
decimal fromSmall = (decimal) 42.76875;
double bigDouble = 2.65e35;
decimal fromBig = (decimal) bigDouble;
```

→ fromSmall: `42.76875m`

→ `System.OverflowException!`

- Only time C# will "check" for overflow by default, instead of letting weird behavior happen

Common Conversions Summary