




# 基本データ型の話

プログラマのためのC言語 第7回



```
int · a · = · 100;
```

↑  
ここの話

変数の型はなんのためにあるのか

メモリの使用量と使い方を決める

実際にメモリでも見てみる

Visual Studioのメモリデバッガ

型の指定方法多すぎる問題

typedefで別名定義

## 基本データ型の種類と範囲

char

short

float

long

int

double

signed

unsigned

## 処理系によってなに？

そもそも処理系とは？

## その他

基本型って？

C言語にbool型はないのか？

signed, unsignedは型修飾子？

“

変数の型はなんのためにあるのか？

- 変数の型はなんのためにあるのか？

○ メモリの使用量と使い方を決めるため



- 変数の型はなんのためにあるのか？

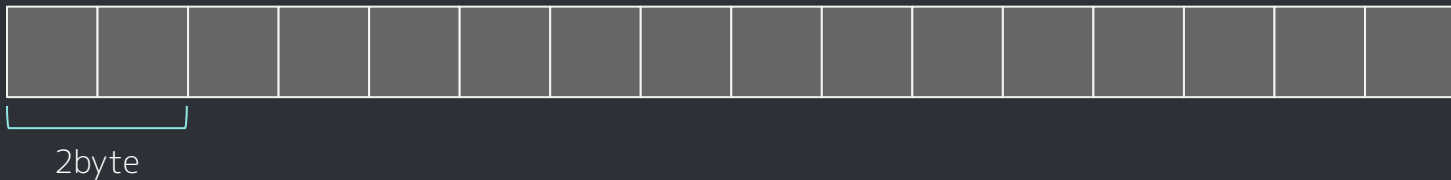
## ○ メモリの使用量と使い方を決めるため



1byte

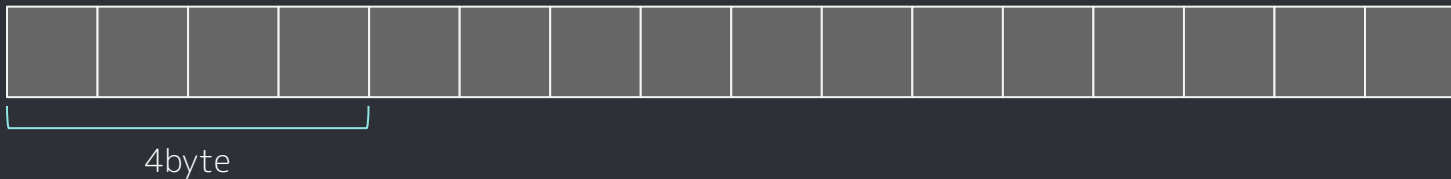
- 変数の型はなんのためにあるのか？

## ○ メモリの使用量と使い方を決めるため



- 変数の型はなんのためにあるのか？

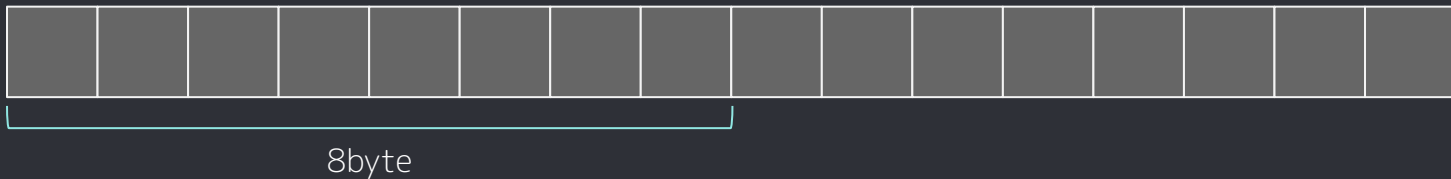
## ○ メモリの使用量と使い方を決めるため





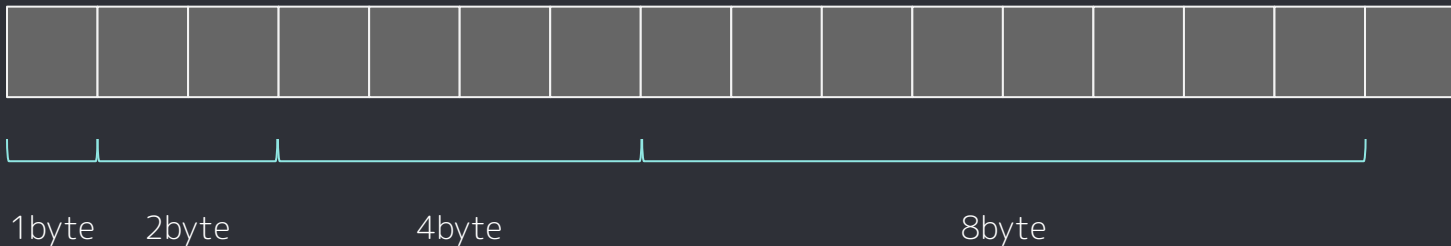
- 変数の型はなんのためにあるのか？

## ○ メモリの使用量と使い方を決めるため



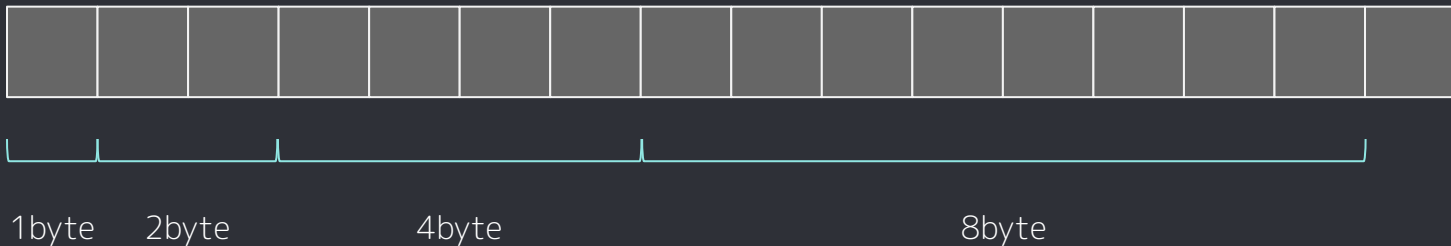
- 変数の型はなんのためにあるのか？

## ○ メモリの使用量と使い方を決めるため



- 変数の型はなんのためにあるのか？

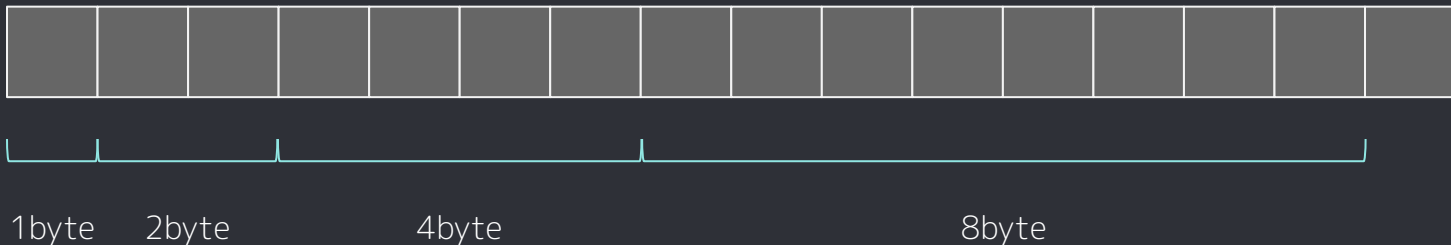
- メモリの使用量と使い方を決めるため



使い方は？

- 変数の型はなんのためにあるのか？

- メモリの使用量と使い方を決めるため



使い方は？

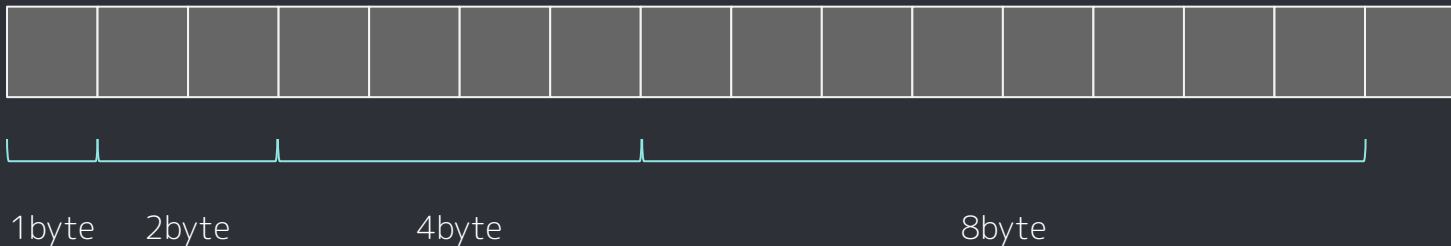
整数

or

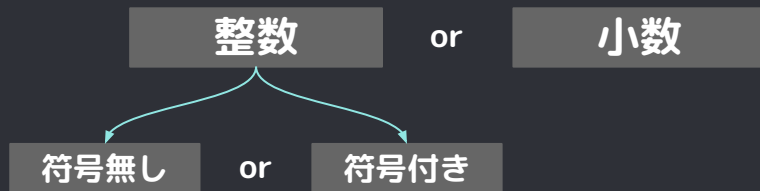
小数

- 変数の型はなんのためにあるのか？

- メモリの使用量と使い方を決めるため



### 使い方は？



“

基本データ型の種類と範囲



使い方

整数

小数

※型のサイズは処理系により異なる場合があります

使い方

符号

整数

符号付き

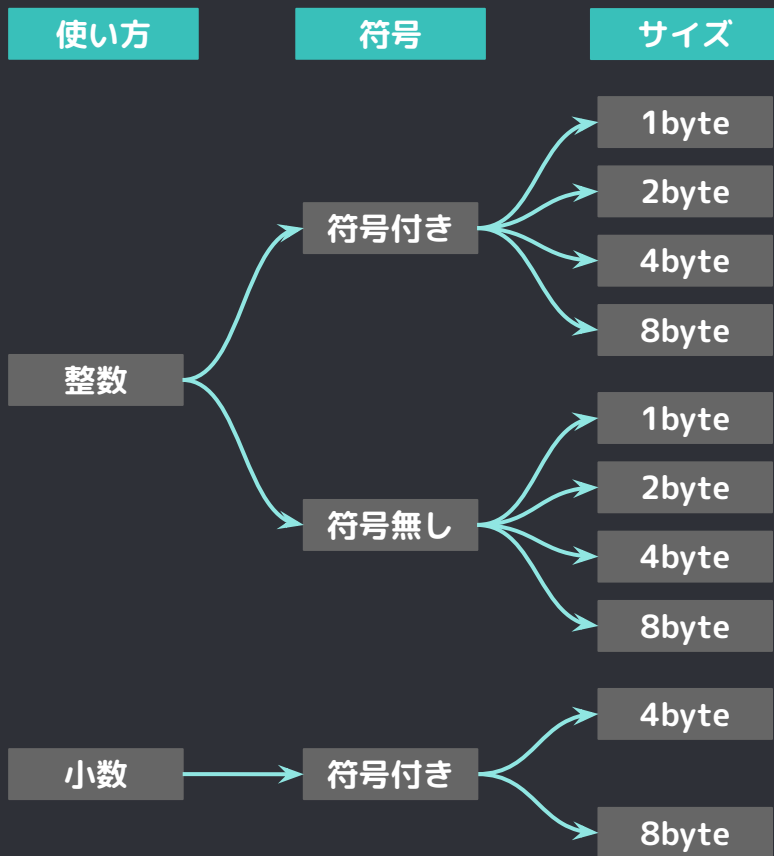
符号無し

小数

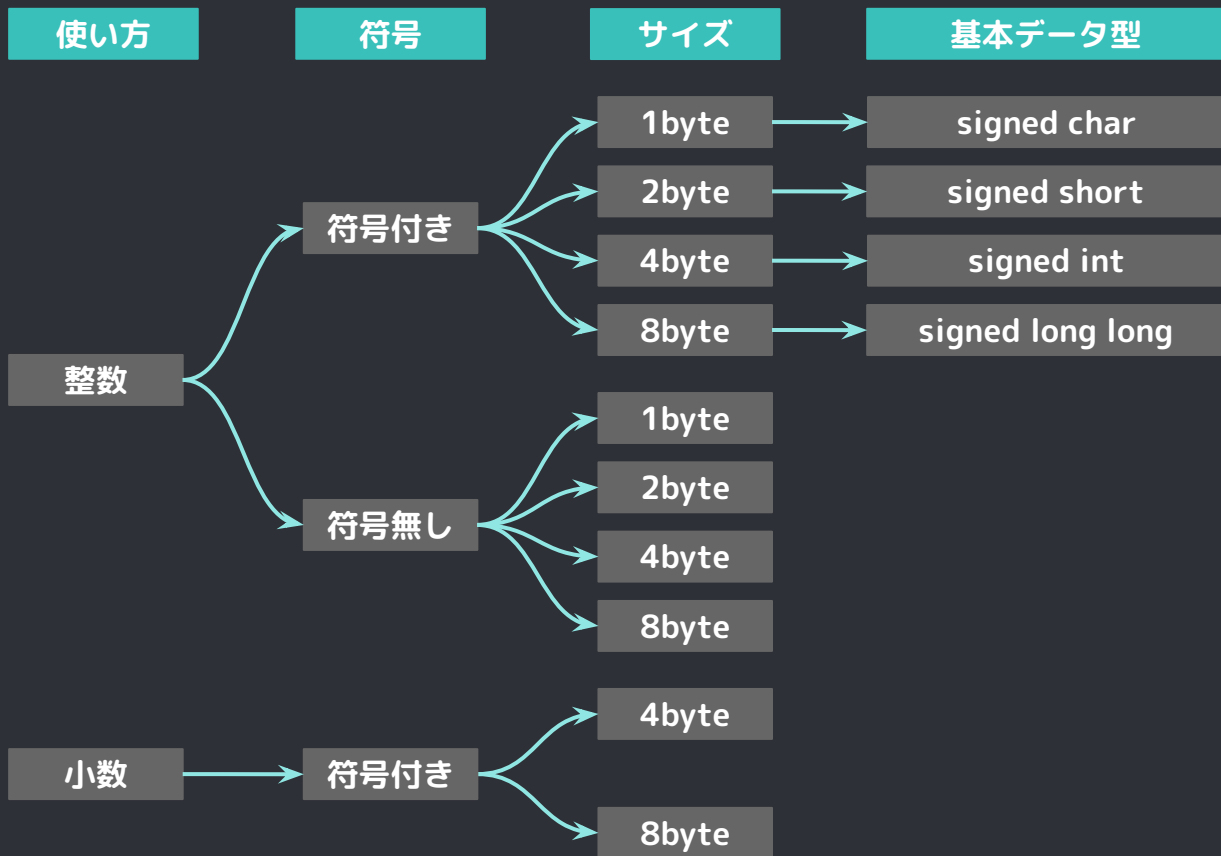
符号付き

※型のサイズは処理系により異なる場合があります

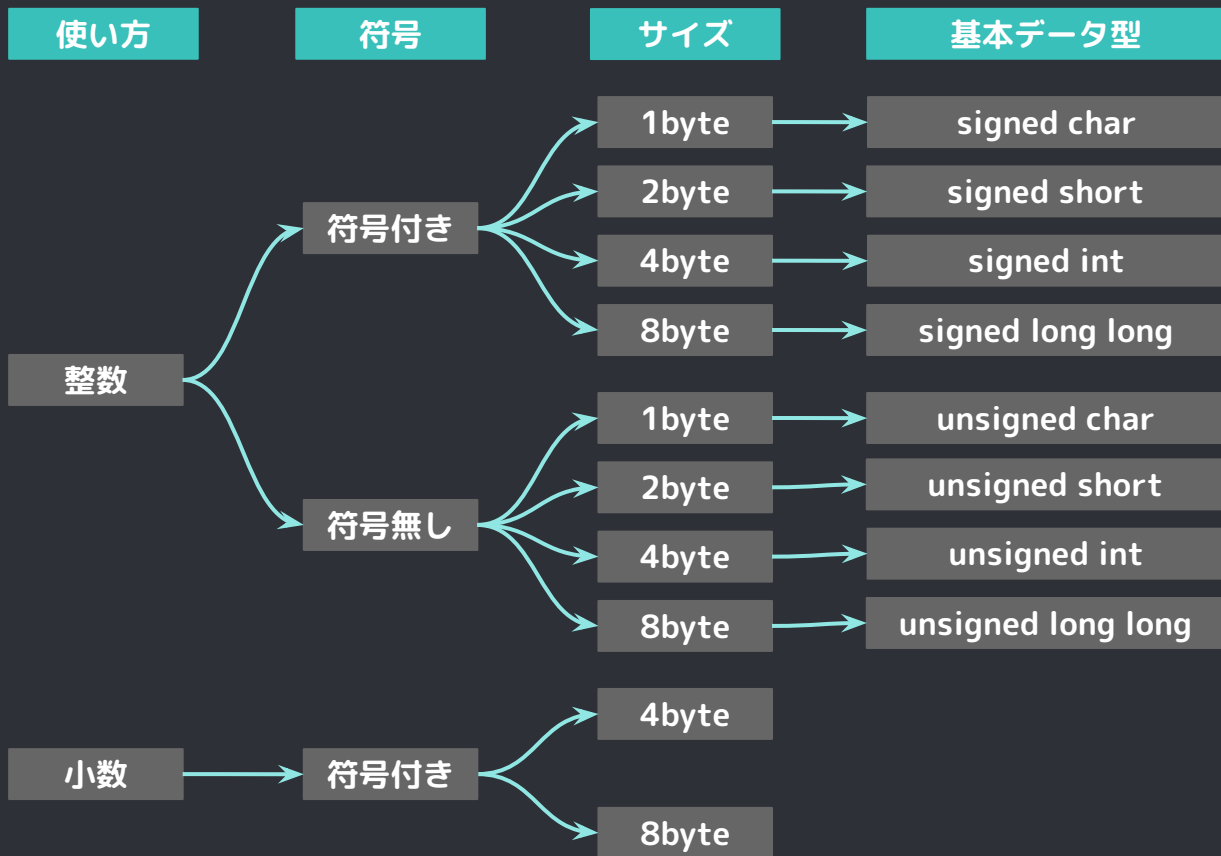




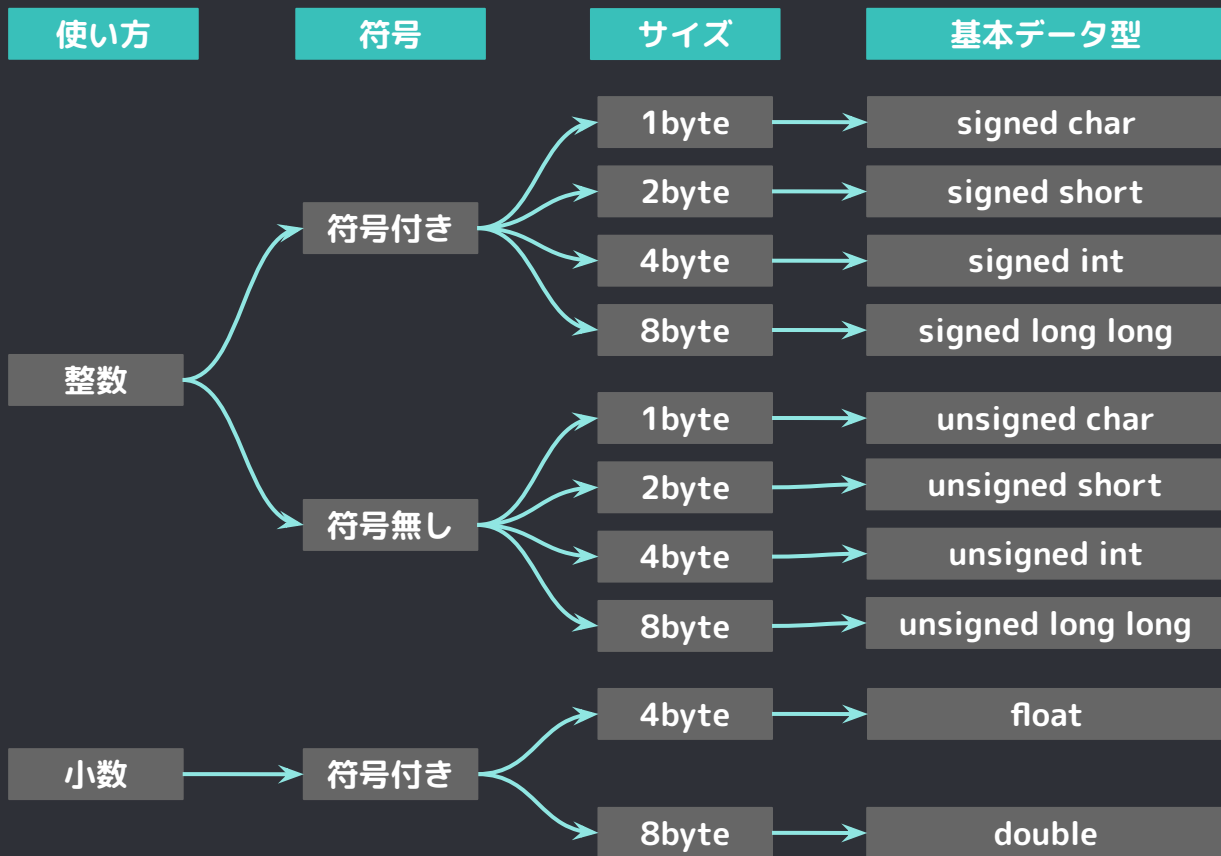
※型のサイズは処理系により異なる場合があります



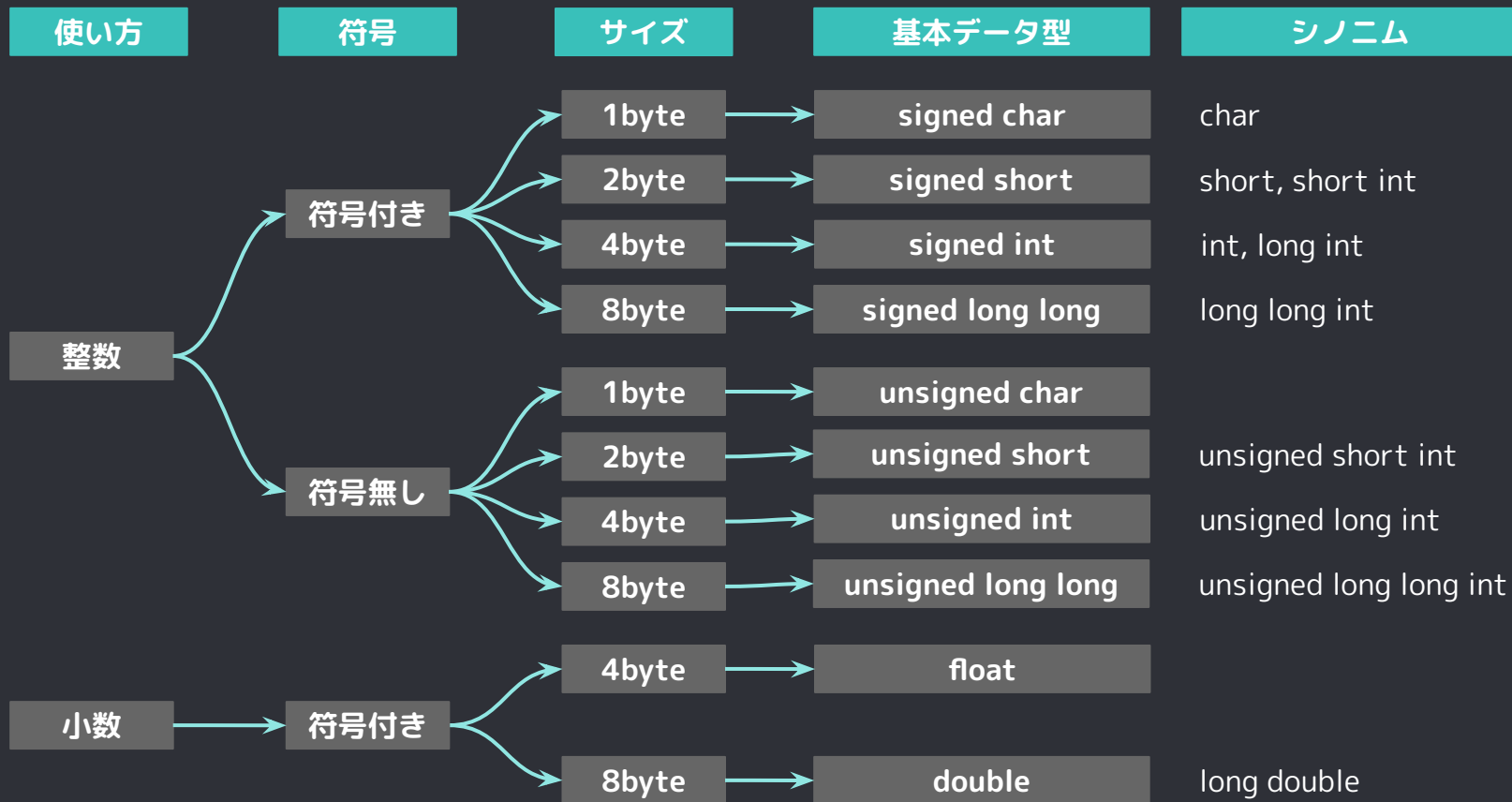
※型のサイズは処理系により異なる場合があります



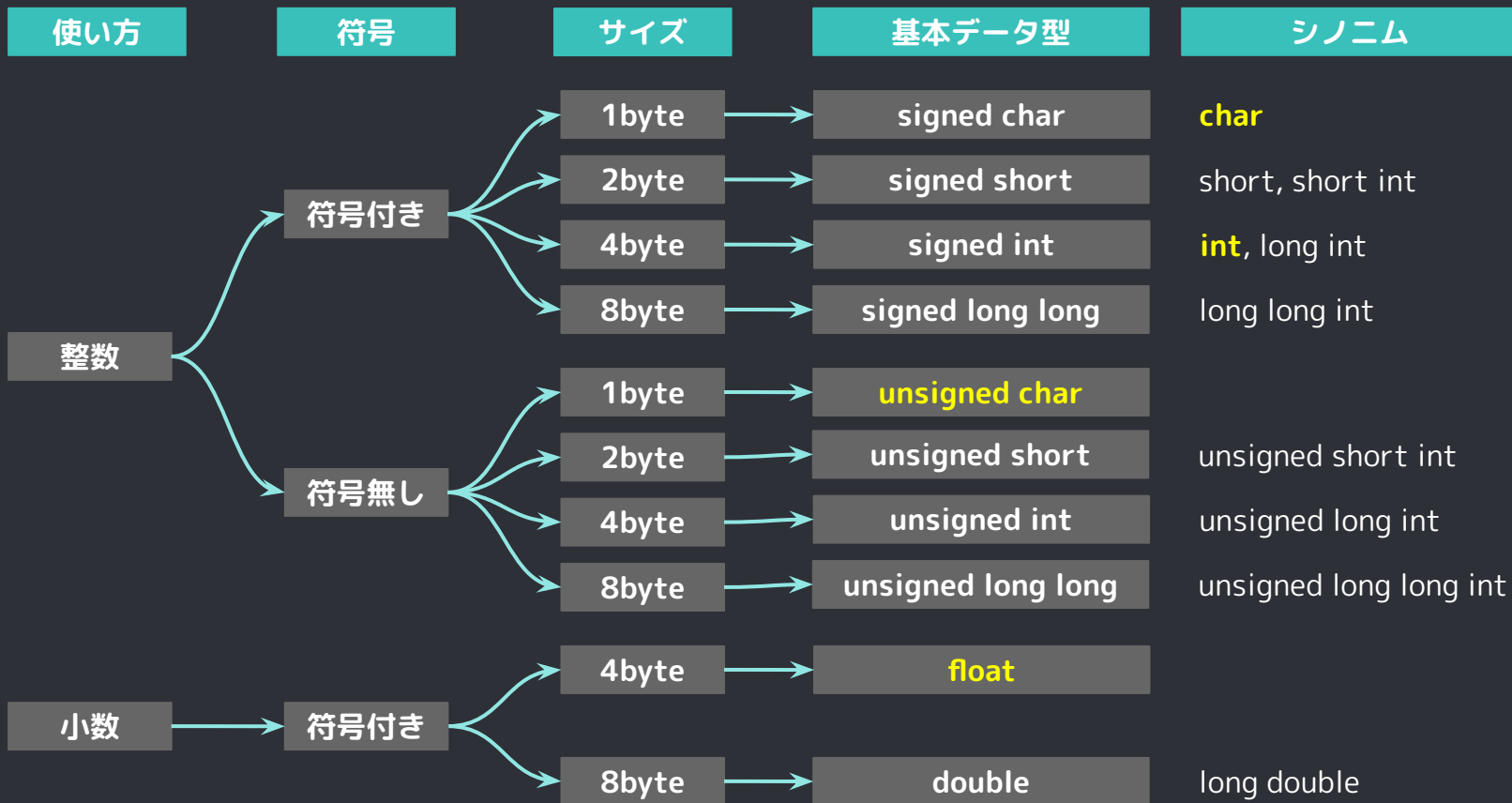
※型のサイズは処理系により異なる場合があります



※型のサイズは処理系により異なる場合があります



※型のサイズは処理系により異なる場合があります



※型のサイズは処理系により異なる場合があります

- 変数のサイズと範囲

※型のサイズは処理系により異なる場合があります

## ● 変数のサイズと範囲

型	サイズ	最小値	最大値
unsigned char	1byte	0	255
signed char	1byte	-128	127

1byte = 8bit = 2の8乗 = 256パターン

※型のサイズは処理系により異なる場合があります



## 変数のサイズと範囲

型	サイズ	最小値	最大値
unsigned char	1byte	0	255
signed char	1byte	-128	127
unsigned short	2byte	0	65,535
signed short	2byte	-32,768	32767

2byte = 16bit = 2の16乗 = 65536パターン

※型のサイズは処理系により異なる場合があります

## 変数のサイズと範囲

型	サイズ	最小値	最大値
unsigned char	1byte	0	255
signed char	1byte	-128	127
unsigned short	2byte	0	65,535
signed short	2byte	-32,768	32767
unsigned int	4byte	0	4,294,967,295
signed int	4byte	-2,147,483,648	2,147,483,647

4byte = 32bit = 2の32乗 = 約43億パターン

※型のサイズは処理系により異なる場合があります

## 変数のサイズと範囲

型	サイズ	最小値	最大値
unsigned char	1byte	0	255
signed char	1byte	-128	127
unsigned short	2byte	0	65,535
signed short	2byte	-32,768	32767
unsigned int	4byte	0	4,294,967,295
signed int	4byte	-2,147,483,648	2,147,483,647
unsigned long long	8byte	0	18,446,744,073,709,551,615
signed long long	8byte	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

8byte = 64bit = 2の64乗 = 約1844京パターン

※型のサイズは処理系により異なる場合があります

## 変数のサイズと範囲

型	サイズ	最小値	最大値
unsigned char	1byte	0	255
signed char	1byte	-128	127
unsigned short	2byte	0	65,535
signed short	2byte	-32,768	32767
unsigned int	4byte	0	4,294,967,295
signed int	4byte	-2,147,483,648	2,147,483,647
unsigned long long	8byte	0	18,446,744,073,709,551,615
signed long long	8byte	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
float	4byte	-3.40282e+38	3.40282e+38
double	8byte	-1.79769e+308	1.79769e+308

e+38 という表記は 10の38乗を表す

※型のサイズは処理系により異なる場合があります

## 変数のサイズと範囲

型	サイズ	最小値	最大値	
unsigned char	1byte	0	255	文字型
signed char	1byte	-128	127	
unsigned short	2byte	0	65,535	整数型
signed short	2byte	-32,768	32767	
unsigned int	4byte	0	4,294,967,295	
signed int	4byte	-2,147,483,648	2,147,483,647	
unsigned long long	8byte	0	18,446,744,073,709,551,615	
signed long long	8byte	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	
float	4byte	-3.40282e+38	3.40282e+38	浮動小数点型
double	8byte	-1.79769e+308	1.79769e+308	

charは1文字分の文字コードを格納するの型だったのでchar型とか文字型と言われるが、内部的には **ただの1byte整数型** であり、文字以外に使っても良い。

※型のサイズは処理系により異なる場合があります

“

実際にメモリでも見てみる

```
int a;
```



a

符号付き整数

```
int a;
```



a

符号付き整数

```
int a;  
unsigned int b;
```



a

b

符号付き整数

符号無し整数



```
int a;
```



a

符号付き整数

```
int a;  
unsigned int b;
```



a

b

符号付き整数

符号無し整数

```
int a;  
short b;  
char c;
```



a

b

c

符号付き整数

```
int a;  
signed int b;  
long c;
```



```
int a;  
signed int b;  
long c;
```



```
float a;  
double b;
```

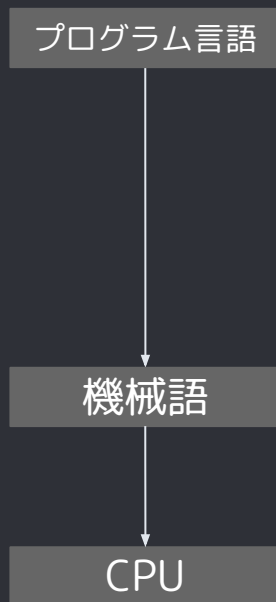


“

処理系によってなに？

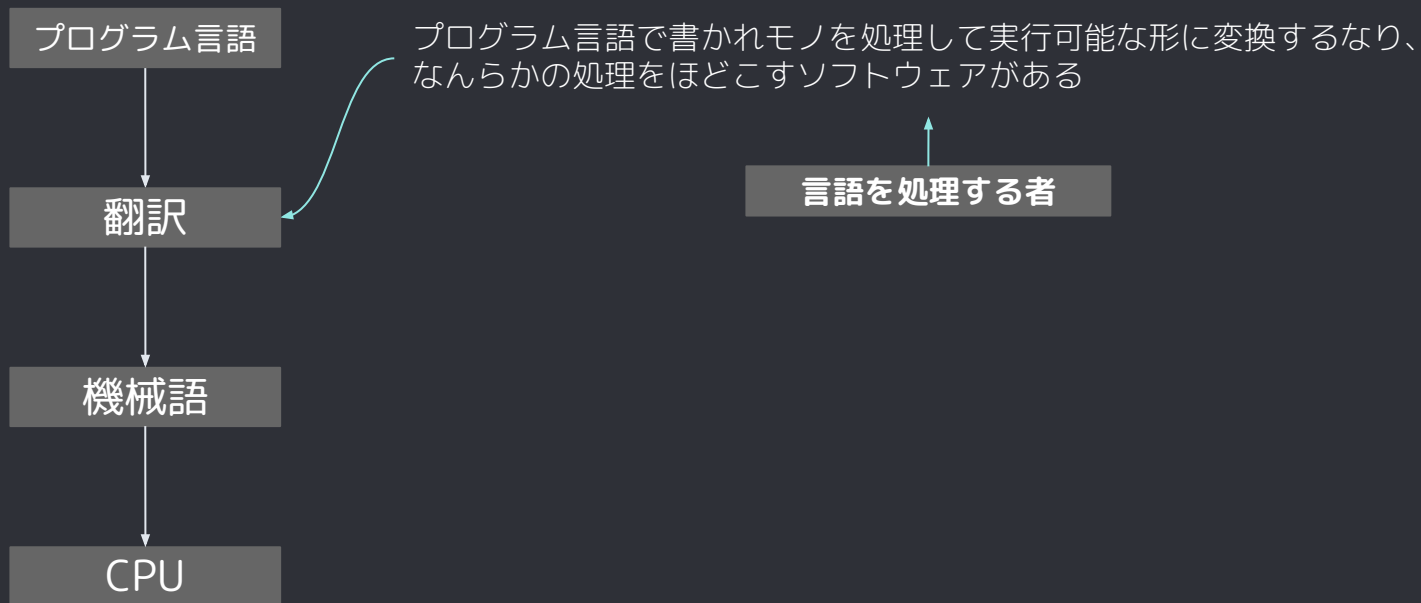
- 処理系によってなに？

○ そもそも処理系ってなに？



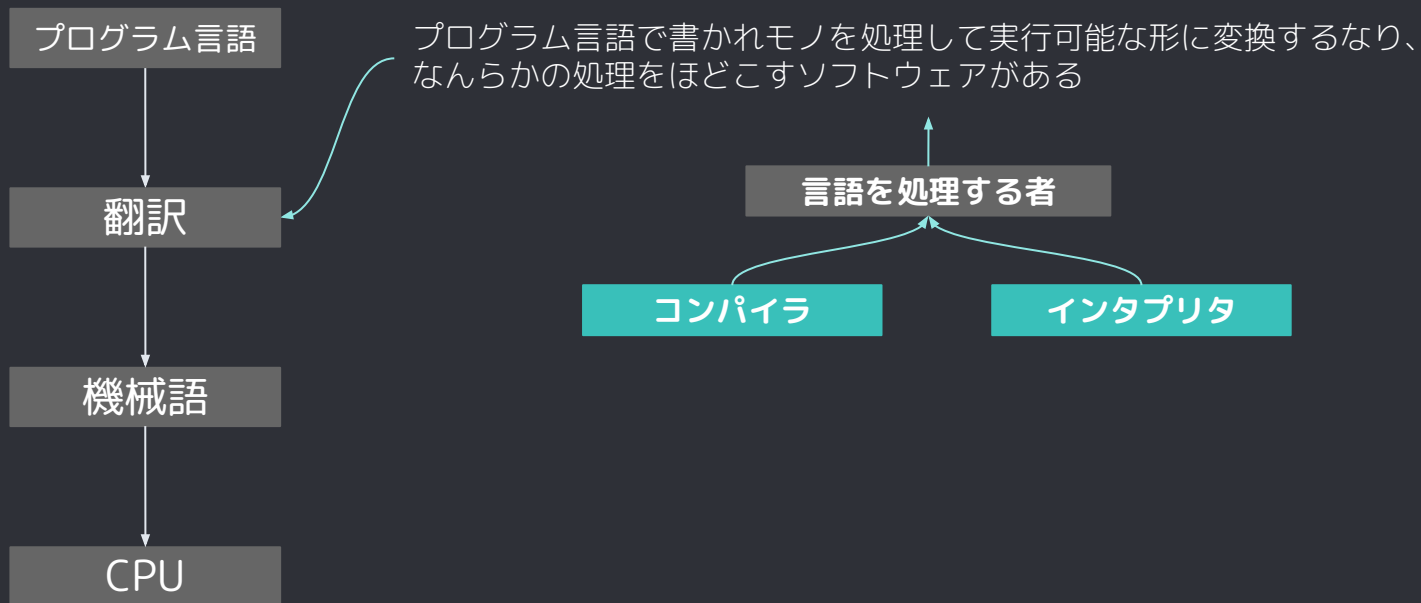
- 処理系によってなに？

- そもそも処理系ってなに？



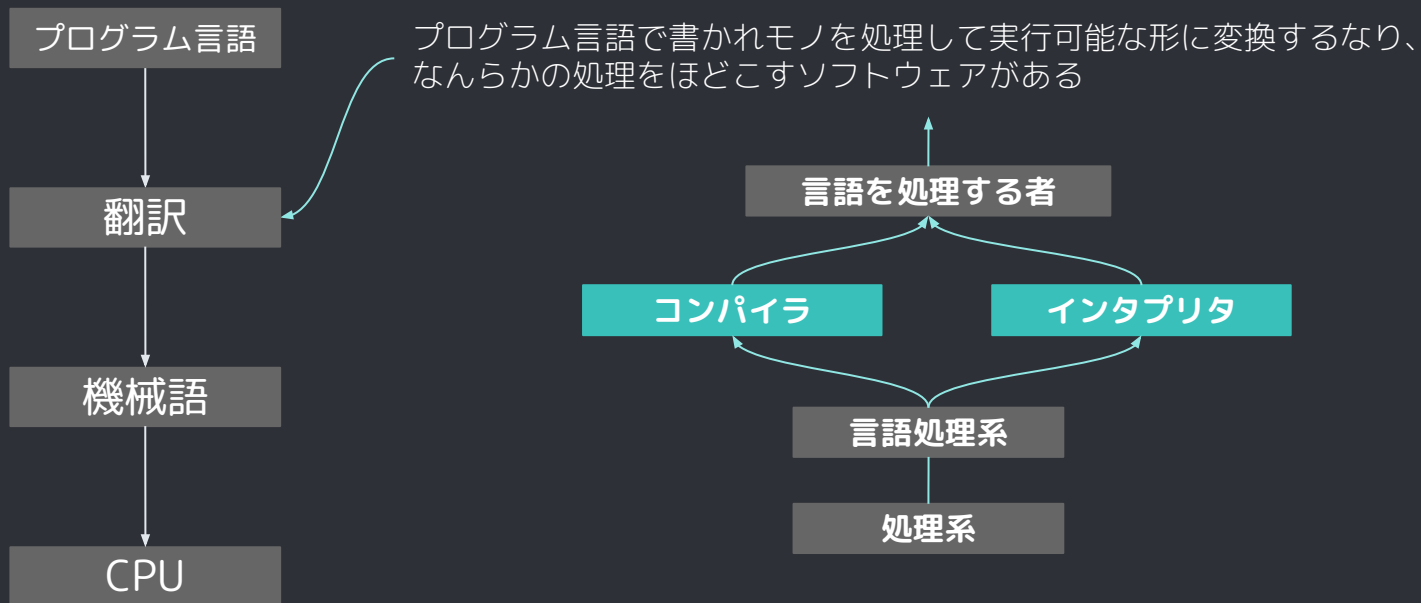
## ● 処理系によってなに？

## ○ そもそも処理系ってなに？



## ● 処理系によってなに？

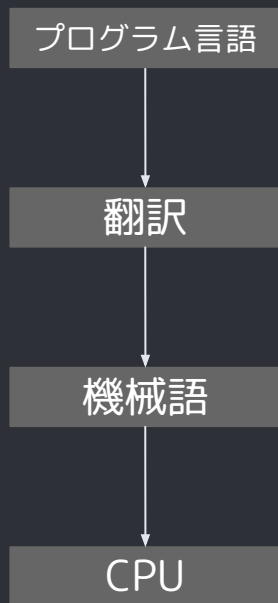
## ○ そもそも処理系ってなに？



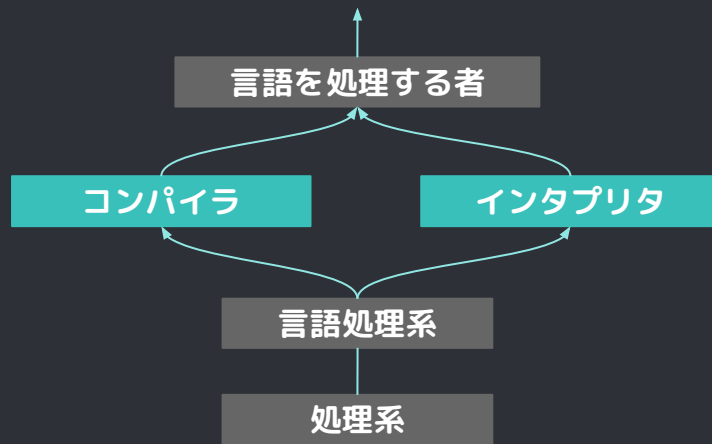


## ● 処理系によってなに？

### ○ そもそも処理系ってなに？



プログラム言語で書かれモノを処理して実行可能な形に変換するなり、  
なんらかの処理をほどこすソフトウェアがある



またOSなどプログラムの動作環境(実行環境)全般の事を処理系といったりもする

- 処理系によってなに？

○ 一言にC言語といっても

C言語

- 処理系によってなに？

## 言語処理系(コンパイラ)は沢山ある

C言語

主な処理系

Visual C++

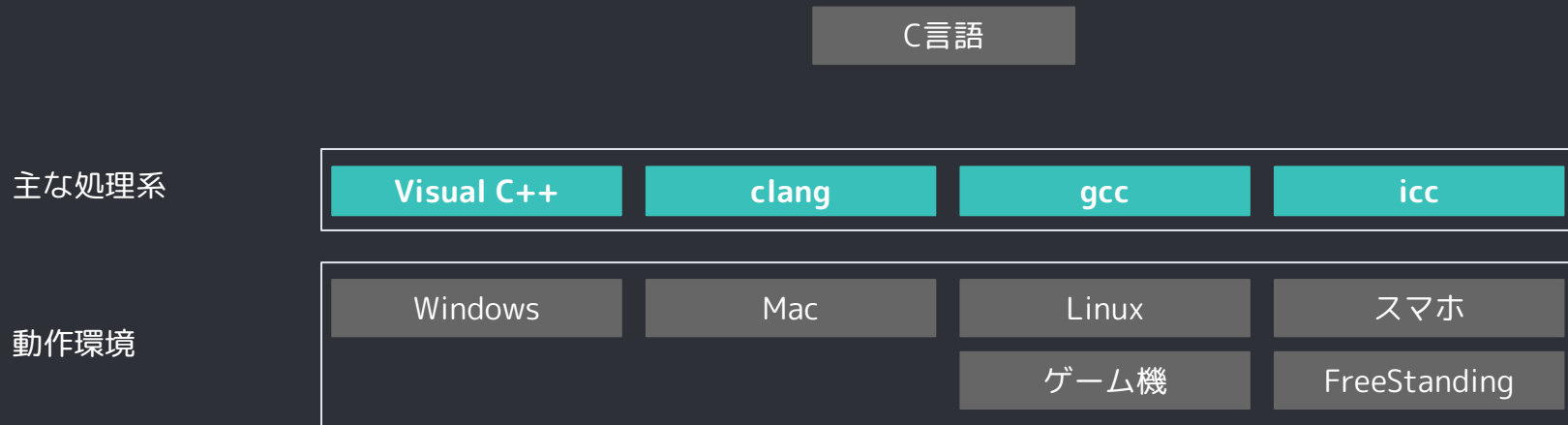
clang

gcc

icc

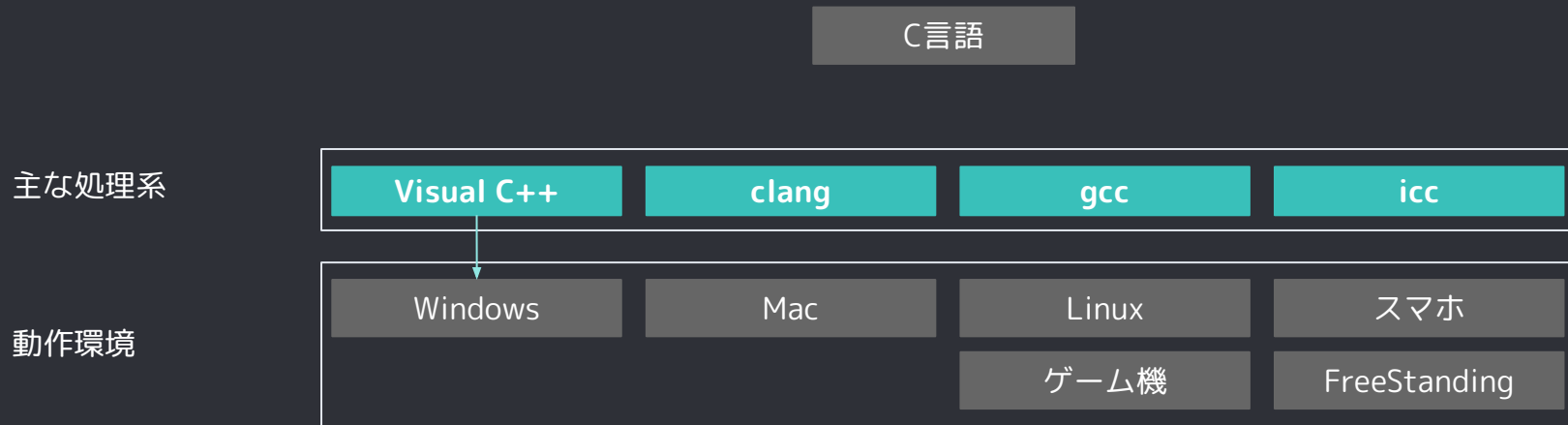
- 処理系によってなに？

## ○ 動作環境も沢山ある



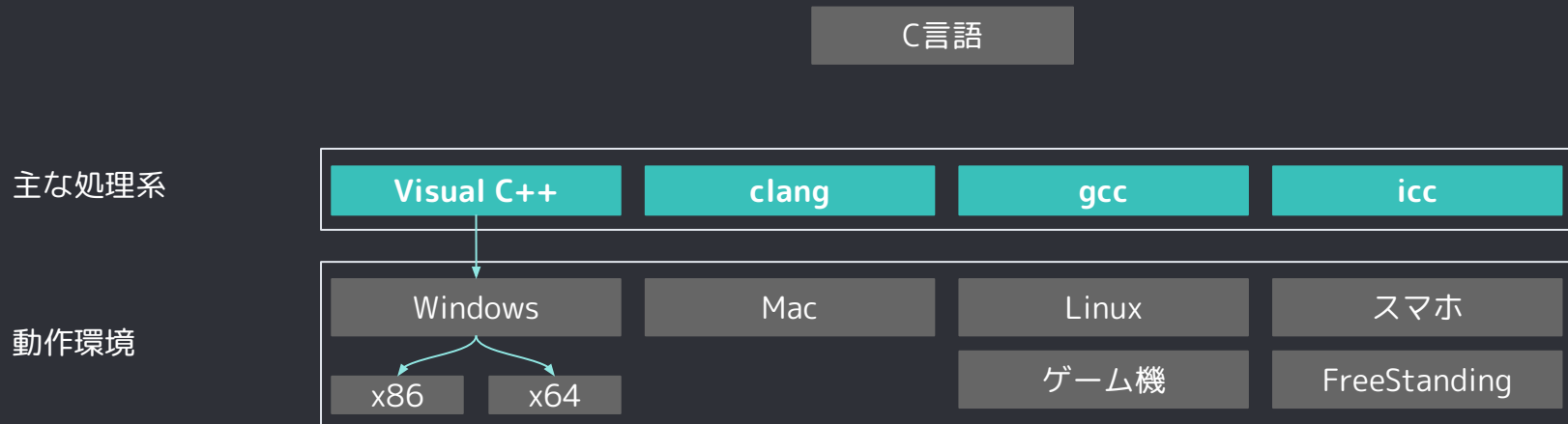
- 処理系によってなに？

- 一言に処理系といっても複雑



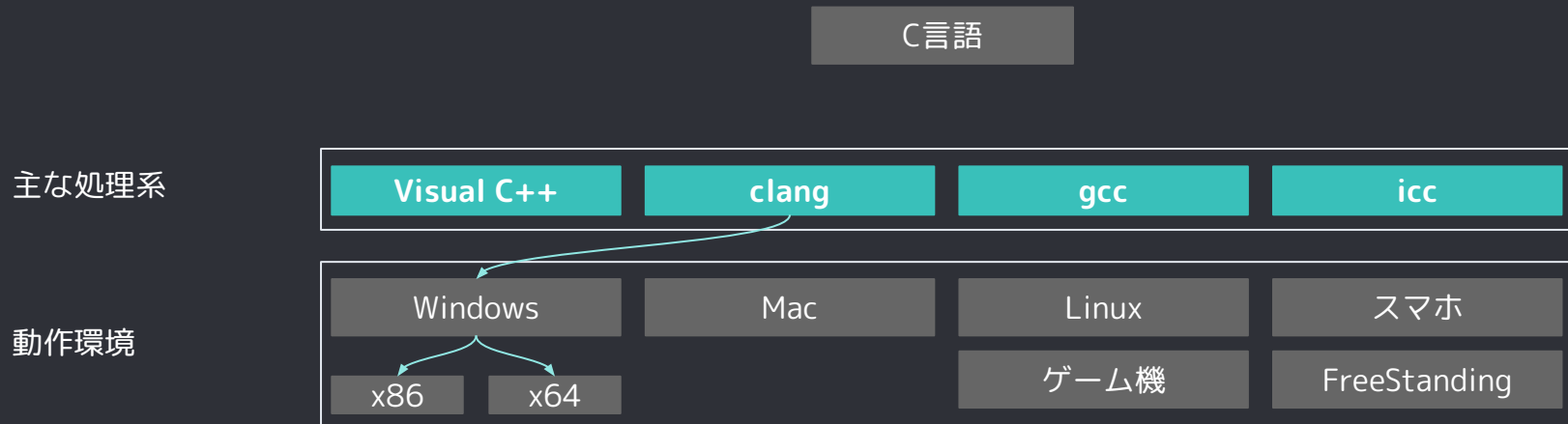
- 処理系によってなに？

- 一言に処理系といっても複雑



- 処理系によってなに？

- 一言に処理系といっても複雑



- 処理系によってなに？

- 処理系で何が変わる？



- 処理系によってなに？

- 処理系で何が変わる？

- ✓ いろいろ

- 処理系によってなに？

- 処理系で何が変わる？

- ✓ いろいろ
- ✓ C言語の標準規格で未規定の動作でコンパイラ(処理系)が動作を定義しているもの
  - ・ 「**処理系定義**」という

## ● 処理系によってなに？

### ○ 処理系で何が変わる？

- ✓ いろいろ
- ✓ C言語の標準規格で未規定の動作でコンパイラ(処理系)が動作を定義しているもの
  - ・ 「**処理系定義**」という
- ✓ 整数型のサイズや表現
  - ・ int型は2byteになる処理系もある
  - ・ signedを省略した場合、符号付きになるか符号無しになるかは処理系による
  - ・ 符号付き整数型のマイナス表現が必ず2の補数表現とは限らない
  - ・ etc...

“

型の指定方法多すぎる問題

## ● 型の指定方法多すぎる問題

### ○ 同じ型でも指定方法が複数ある

```
int.....a=0; // 4byte 符号付き整数
signed.....b=0; // 4byte 符号付き整数
long.....c=0; // 4byte 符号付き整数
signed int..d=0; // 4byte 符号付き整数
signed long.e=0; // 4byte 符号付き整数
```

Visual C++(32bit環境)だとこれらは全部同じ

## ● 型の指定方法多すぎる問題

### ○ 同じ型でも指定方法が複数ある

```
int.....a=0; // 4byte 符号付き整数  
signed.....b=0; // 4byte 符号付き整数  
long.....c=0; // 4byte 符号付き整数  
signed int...d=0; // 4byte 符号付き整数  
signed long..e=0; // 4byte 符号付き整数
```

Visual C++(32bit環境)だとこれらは全部同じ

### 別の処理系に移植したら変数の扱いが変わるかも

```
int.....a=0; // 2byte 符号無し整数  
signed.....b=0; // 4byte 符号付き整数  
long.....c=0; // 8byte 符号無し整数  
signed int...d=0; // 2byte 符号付き整数  
signed long..e=0; // 8byte 符号付き整数
```

## ● 型の指定方法多すぎる問題

### ○ typedef で別名を定義

```
typedef char .....c8; // 文字用 8bit
typedef char .....i8; // 符号付き整数 8bit
typedef short .....i16; // 符号付き整数 16bit
typedef int .....i32; // 符号付き整数 32bit
typedef long long .....i64; // 符号付き整数 64bit
typedef unsigned char .....u8; // 符号無し整数 8bit
typedef unsigned short .....u16; // 符号無し整数 16bit
typedef unsigned int .....u32; // 符号無し整数 32bit
typedef unsigned long long .....u64; // 符号無し整数 64bit
typedef float .....f32; // 単精度浮動小数
typedef double .....f64; // 倍精度浮動小数
```

変数の用途とサイズで別名定義して、定義した型を使う  
処理系が変わった場合、この定義だけ修正すればOK

“

その他



- その他

- 基本型って？

基本型

char

int

float

double

etc...

プリミティブ型とか、組み込み型っていたりもする

- その他

- 基本型って？

### 基本型

char

int

float

double

etc...

void

プリミティブ型とか、組み込み型っていたりもする

## ● その他

### ○ 基本型って？

#### 基本型

char

int

float

double

etc...

void

プリミティブ型とか、組み込み型っていったりもする

#### 複合型

構造体

列挙型

共用体

ユーザー定義型といったりもする

## ● その他

### ○ 基本型って？

#### 基本型

char

int

float

double

etc...

void

プリミティブ型とか、組み込み型っていったりもする

#### 複合型

##### 構造体

char

int

float

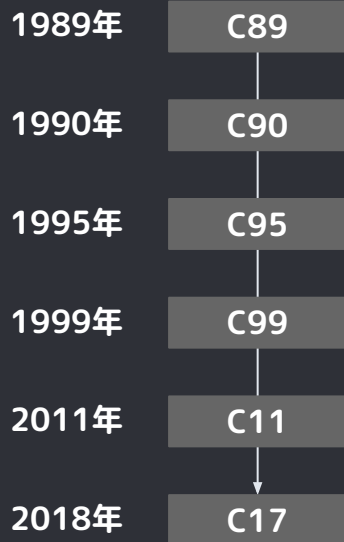
##### 列挙型

##### 共用体

ユーザー定義型といったりもする

- その他

- C言語にbool型はない？



- その他

- C言語にbool型はない？

1989年

C89

1990年

C90

1995年

C95

1999年

C99

Bool という型が追加された

2011年

C11

2018年

C17

- その他

- signed や unsigned は型修飾子？

- その他

- signed や unsigned は型修飾子？

- ✓ 型修飾子 ではなく **型指定子** に分類される
  - ・ メモリをどれくらい、どのように扱うのかは 型 である



## ● その他

### ○ signed や unsigned は型修飾子？

- ✓ 型修飾子 ではなく **型指定子** に分類される
  - ・ メモリをどれくらい、どのように扱うのかは 型 である
- ✓ C言語の場合 **型修飾子** は **const** や **volatile** がある
  - ・ 変数への再代入を禁止したり
  - ・ 最適化を抑止したり

## ● その他

### ○ signed や unsigned は型修飾子？

- ✓ 型修飾子 ではなく **型指定子** に分類される
  - ・ メモリをどれくらい、どのように扱うのかは 型 である
- ✓ C言語の場合 **型修飾子** は **const** や **volatile** がある
  - ・ 変数への再代入を禁止したり
  - ・ 最適化を抑止したり
- ✓ 他の言語
  - ・ public, protected, private といったアクセス修飾子や static, final などがある



おしまい