

# C++ むかしばなし

わんくま同盟

Microsoft MVP for

Visual Developer - Visual C++

*επιστημη*

episteme@wankuma.com

## 二十数年前のある日...

- AT&T , Bjarne Stroustrup
- シミュレーションの研究にSimulaを利用
- めさめさ重たい! → えれー迷惑
- 新たに言語を作っちゃまえ!
- Cで書けば速いのが手っ取り早く  
作れんでね?
- 前提: 今あるコンパイラ/リンクを**そのまま**使う!

インタプリタ と トランスレータ

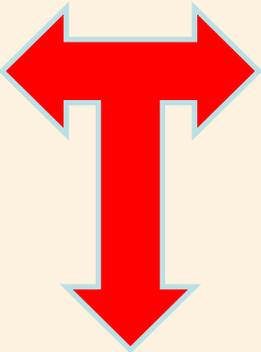
S



L

LによるSのインタプリタ

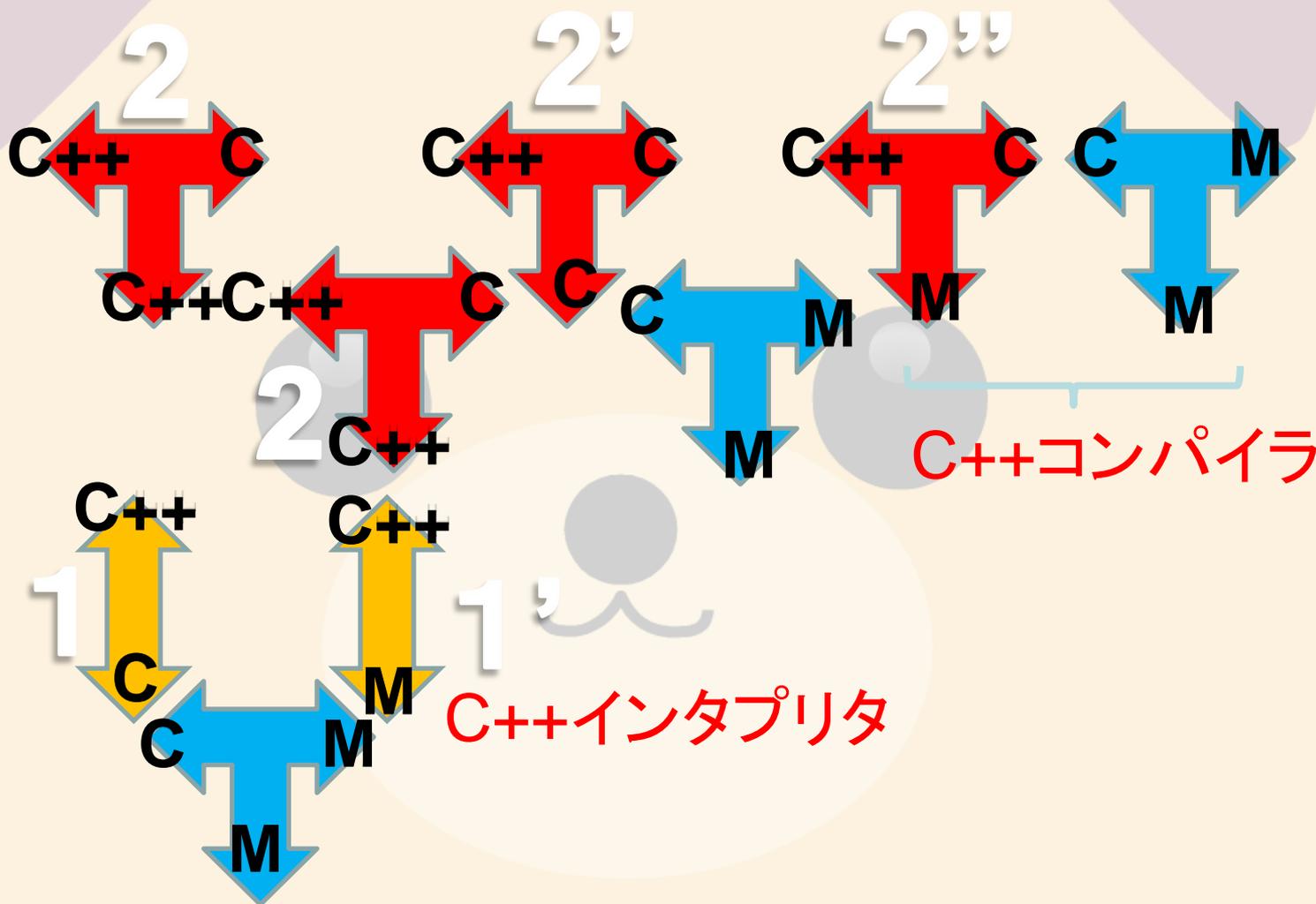
S



D

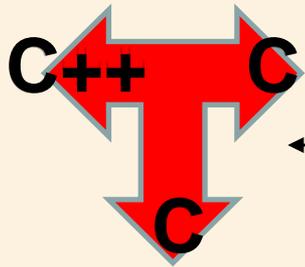
LによるS→Dトランスレータ

# C++コンパイラのつくりかた



ここで疑問

なぜ Cで書かなかったのでしょうか？



← これコンパイルすればあっちゅーまにできるやん!

## Munch(Bunch)と呼ばれるツール

- グローバルインスタンスを何時コンストラクト?
- グローバルインスタンスを何時デストラクト?
- コンストラクタ/デストラクタ・チェーンを作り
- mainの直前/直後にチェーンをたどる
- グローバルインスタンスはリンクするまで未定
- リンカは従来の(C用)を使うので...
- リンク後にパッチをあてる

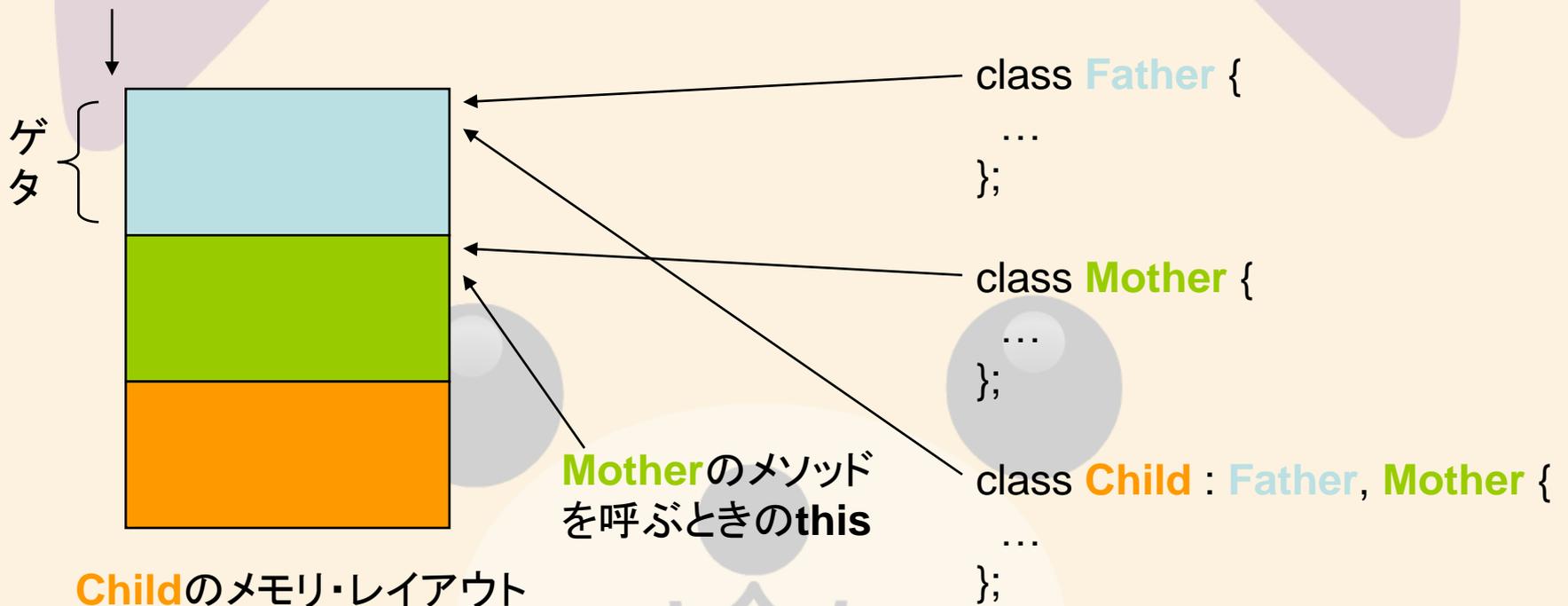
↑これがMunch

## 仮想関数テーブルの在り処

- 昔のコンパイラ(fcfront1.2あたり)  
仮想関数テーブルはヘッダを#includeしたすべてのコンパイル・ユニットにstaticで展開  
→ **でけー!**唯一ひとつにできんもんかね。
- Cfront2.0の頃改良  
仮想関数のうち、最初に宣言されてるものを実装しているコンパイル・ユニットに展開

# FatherとChildのメソッド を呼ぶときのthis

## 多重継承はメンドクセー



**Child\* を Mother\* にキャストするとズれる!**

→ テキトーに(Father分だけ)ゲタを履かさにゃなんののです

# マクロによるtemplateもどき

Generic.h (抜粋)

```
#define name2(a,b) a¥ ← 継続行  
b  
  
#define declare(a,t) name2(a,declare)(t)  
#define implement(a,t) name2(a,implement)(t)
```

declare(Stack,int)

→ name2(Stack,declare)(int)

→ Stackdeclare(int)

implement(Stack,int)

→ names(Stack,implement)(int)

→ Stackimplement(int)

# マクロによるtemplateもどき

Stack(T)のヘッダ :  
stack.h

```
#define Stack(type) name2(Stack, type)

#define Stackdeclare(T) ¥
class Stack(T) {¥
    T data[16];¥
    int top; ¥
public: ¥
    Stack(T) () { ¥
        top = 0; ¥
    } ¥
    ... ¥
};

#define Stackimplement(T) ¥
Stack(T)::Stack(T) () { ¥
    top = 0; ¥
} ¥
...
```

# マクロによるtemplateもどき

```
#include "stack.h"  
  
declare(Stack, int)  
implement(Stack, int)  
  
int main() {  
    Stack(int) s;  
    s.push(1);  
    s.push(2);  
    ...  
}
```

使うときのオマジナイ

どこかに一箇所書くべし

## templateの実現方法(1)

- Inclusion-model

実装をヘッダに書く。

現実装系の多くが採用

コンパイル・ユニットそれぞれに展開される

なので生成コードがデカい

リンカが重複コードをまとめる

## templateの実現方法(2)

- separation-model

### 実装をヘッダと分離する

ヘッダと同じディレクトリに同一basenameで  
実装を用意しておき、

1. コンパイル(実装コードは無い)
2. 仮リンク → 未定義関数一覧が手に入る
3. それを手がかりに実装をコンパイル
4. 改めてリンクして完成