MODERN C++ DESIGN PATTERNS

Modern C++ Alignment

by Prasanna Ghali

Data Alignment in Structures

- For scalar data types, compilers assign addresses that are divisible by size of data type in bytes
  - Variables of type `int` are assigned storage at addresses divisible by 4 i.e., these addresses have least significant 2 bits cleared to 0
  - Variables of type `double` are assigned storage at addresses divisible by 8
- Compiler must therefore *pad* structures, classes, and unions so that each structure element is naturally aligned

Sizes

92 bytes

88 bytes

- What is result of evaluation of `sizeof(Weapon)`, `sizeof(Armor)`, and `sizeof(Player)`?

268 bytes

```
struct Weapon {
  char     name[81];
  int32_t  damage;
  float    range;
};
```

```
struct Armor {
  char     name[81];
  int32_t  protection;
};
```

```
struct Player {
  char     name[81];
  Weapon   weapon;
  Armor    armor;
  int32_t  health;
};
```

Data Padding: struct Player

- Memory layout of hero can only be determined by looking at individual data members of Player

```
struct Player {
  char    name[81];
  Weapon  weapon;
  Armor   armor;
  int32_t health;
};

Player hero;
```

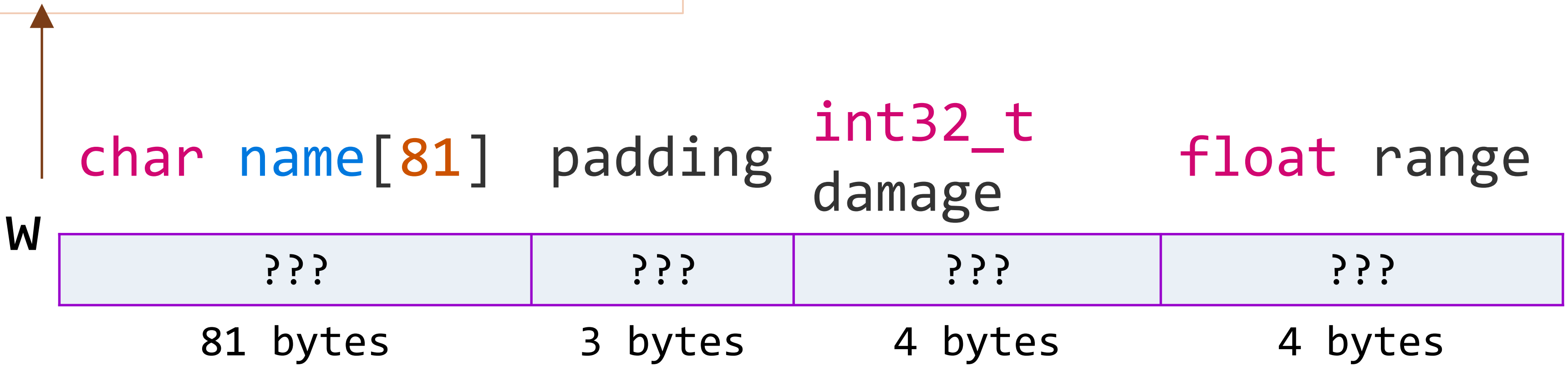
Data Alignment: `struct Weapon` (1/2)

- □ Compilers assign storage for variables of scalar data types at addresses that are multiples of data type’s size in bytes
- □ To ensure each structure element is naturally aligned, compilers *must* align structure objects based on *largest* data member type

Data Alignment: struct Weapon (2/2)

Objects of type Weapon are given storage at addresses divisible by 4 since they contain int32_t and float data members

```
struct Weapon {
   char     name[81];
   int32_t  damage;
   float    range;
};
Weapon w;
```



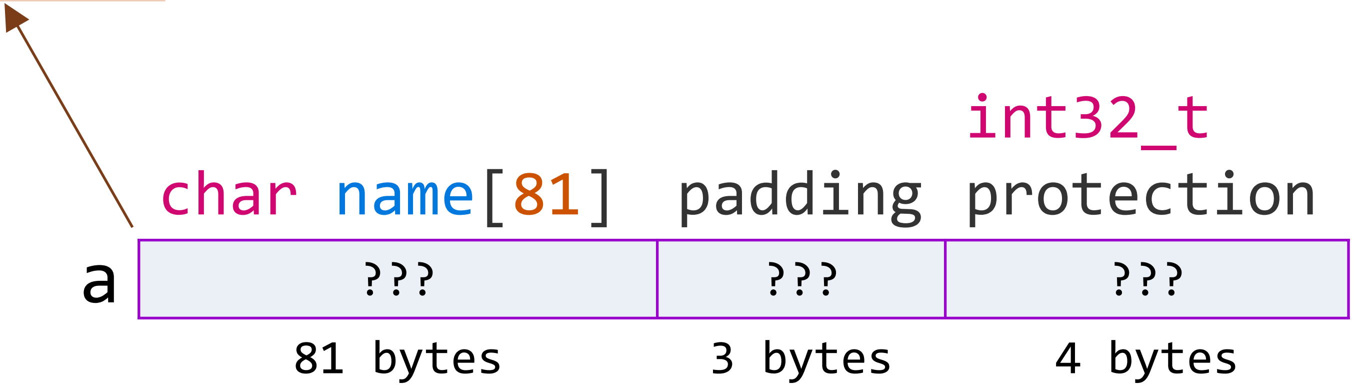
w

char name[81]	padding	int32_t damage	float range
???	???	???	???
81 bytes	3 bytes	4 bytes	4 bytes

Data Alignment: struct Armor

```
struct Armor {
   char     name[81];
   int32_t protection;
};
Armor a;
```

Objects of type Armor are given storage at addresses divisible by 4



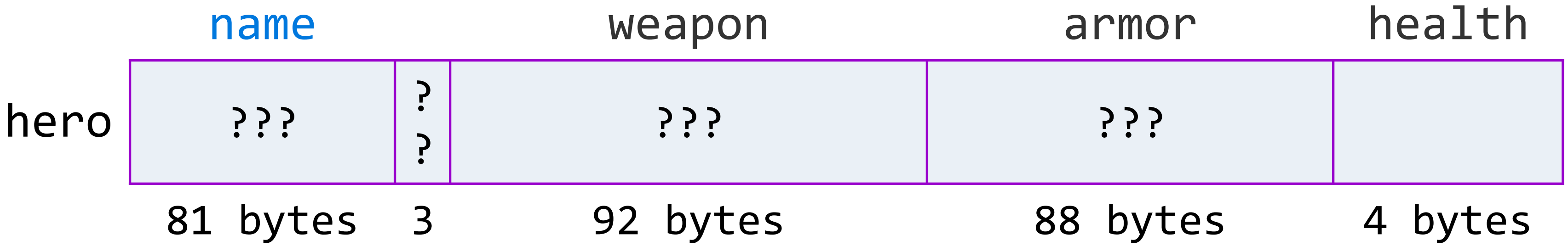
Data Padding: struct Player (1/2)

```
struct Weapon {
  char    name[81];
  int32_t damage;
  float   range;
};

struct Armor {
  char    name[81];
  int32_t protection;
};
```

```
struct Player {
  char    name[81];
  Weapon  weapon;
  Armor   armor;
  int32_t health;
};

Player hero;
```

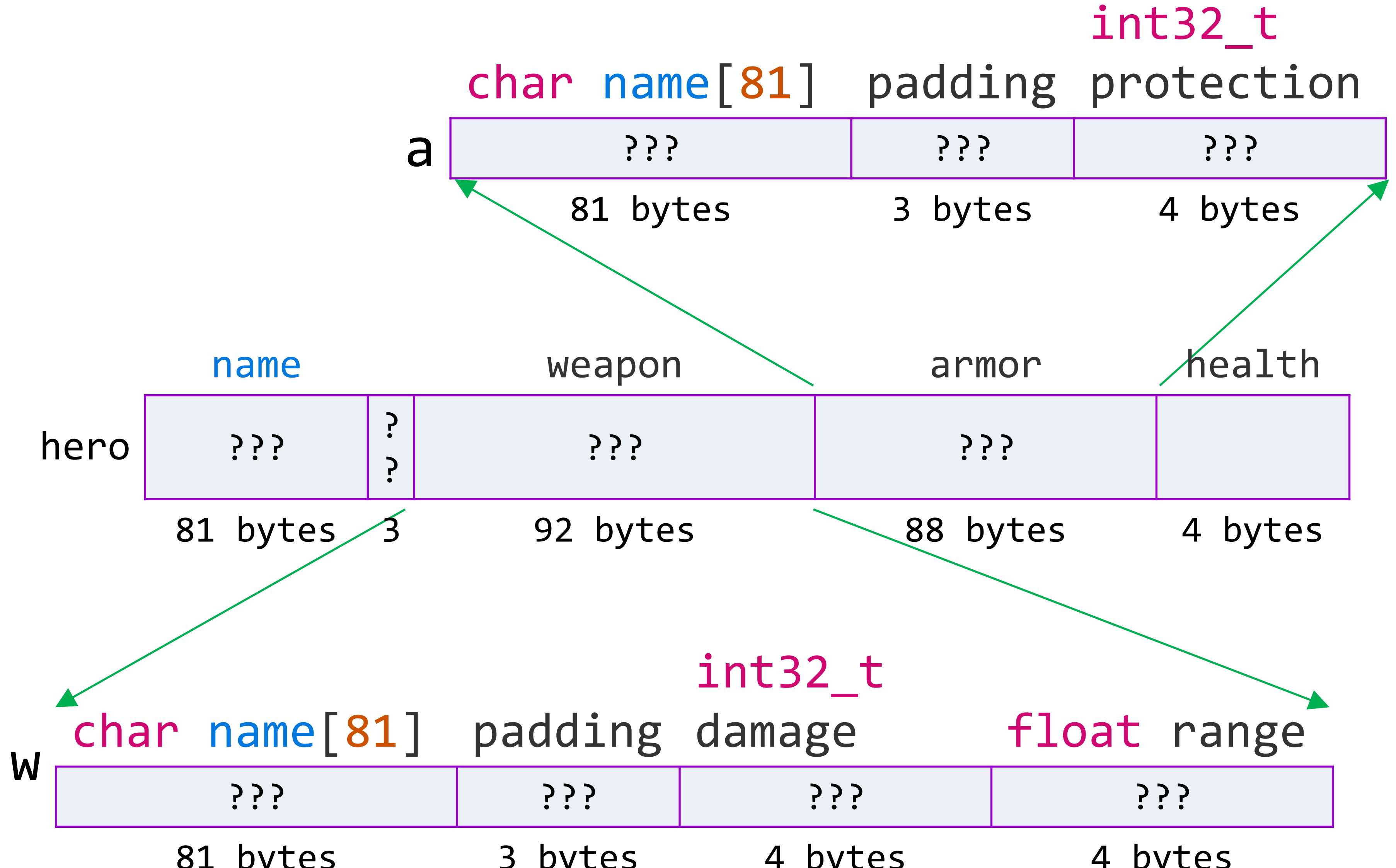


name weapon armor health

hero ??? ?? ??? ???

81 bytes 3 92 bytes 88 bytes 4 bytes

Data Padding: struct Player (2/2)



int32_t

char name[81] padding protection

a

??? ??? ???

81 bytes 3 bytes 4 bytes

name weapon armor health

hero

??? ?? ??? ???

81 bytes 3 92 bytes 88 bytes 4 bytes

int32_t

char name[81] padding damage float range

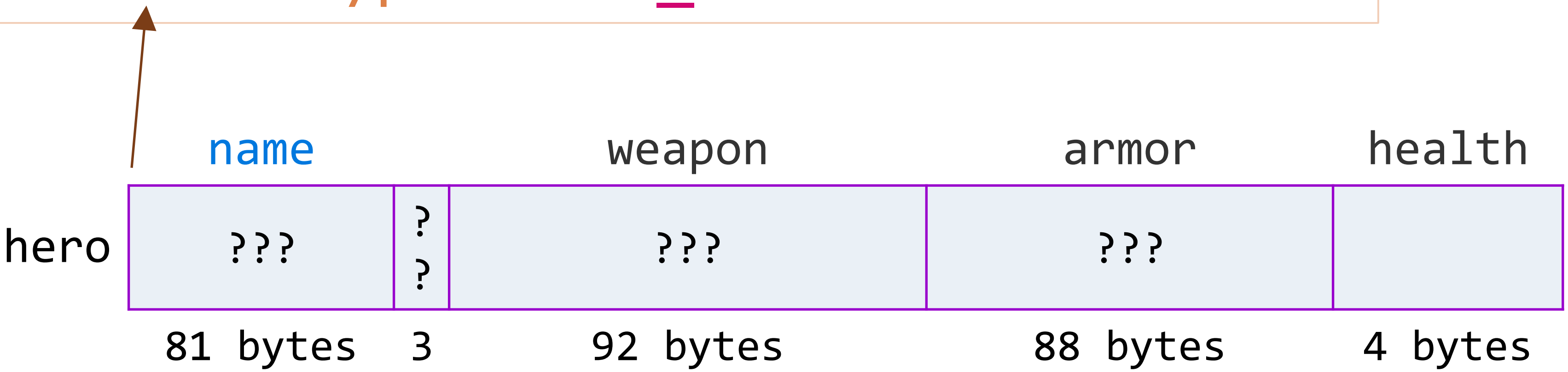
w

??? ??? ??? ???

81 bytes 3 bytes 4 bytes 4 bytes

Data Alignment: struct Player

Objects of type Player are given storage at addresses divisible by 4 since it contains data members of type int32_t and float



	name		weapon	armor	health
hero	???	? ?	???	???	
	81 bytes	3	92 bytes	88 bytes	4 bytes

Custom Alignment (1/2)

- Now, we know how compilers align class and struct objects in memory
- But, how to align such objects on a specific byte boundary?

```
// how to align member i at 16 byte boundaries
struct PreCpp11 {
  int32_i i;
};
```

Custom Alignment (2/2)

- Inject padding …

```
struct PreCpp11 {
  int32_t i;
  uint8_t pad[16-sizeof(int32_t)];
};

PreCpp11 p1;
bool flag = reinterpret_cast<size_t>(&p1) % 16;
std::cout << "p1 is " << (flag ? "not" : "")
          << "aligned at 16 byte boundary\n";
```

Custom Alignment in Modern C++

- `alignas` is C++11 way of specifying custom alignment on class, struct, or union, or on individual members
- `alignof` is C++11 way of querying alignment requirements of a type

```
struct Empty {};
struct alignas(32) Empty32 {};
struct SAC { char ch[8]; };
struct alignas(alignof(long double)) SALD {
  char ch[8];
};
```