System Programming in Rust

Yihao Sun yihao.sun@usu.edu

Fall 2026

Learning Target

- This is for those who suffered too much C++
- Rust 101: Functions, type, control flow….
- Struct, Enum and Trait in Rust
- Tooling for Rust
- Learn how to write an Expression Evaluator

Toolchain

Old School System Programming
(In minimal !!!!)













CONAN
C/C++ Package Manager

Now



Cargo is your, package manager + developer tool + build scripting + linter + documentation….

Cargo

C++

CMakeList.txt ——— 200+ line
cmake/
  gtest.cmake ——— 200+ line
  CPM.cmake ——— 200+ line
  …
vcpkg.conf

Rust

cargo.toml

Cargo

C++

```
vcpkg add port fmt
cmake -B build -D…
cmake --build build
ctest --test-dir build
./build/app
```

Rust

```
cargo run
```

Programming Paradigm

Imperative







Functional







Logic



SWI Prolog



Multi-Paradigm Programming

Imperative



Functional



Logic





After this course

Representing data: numbers

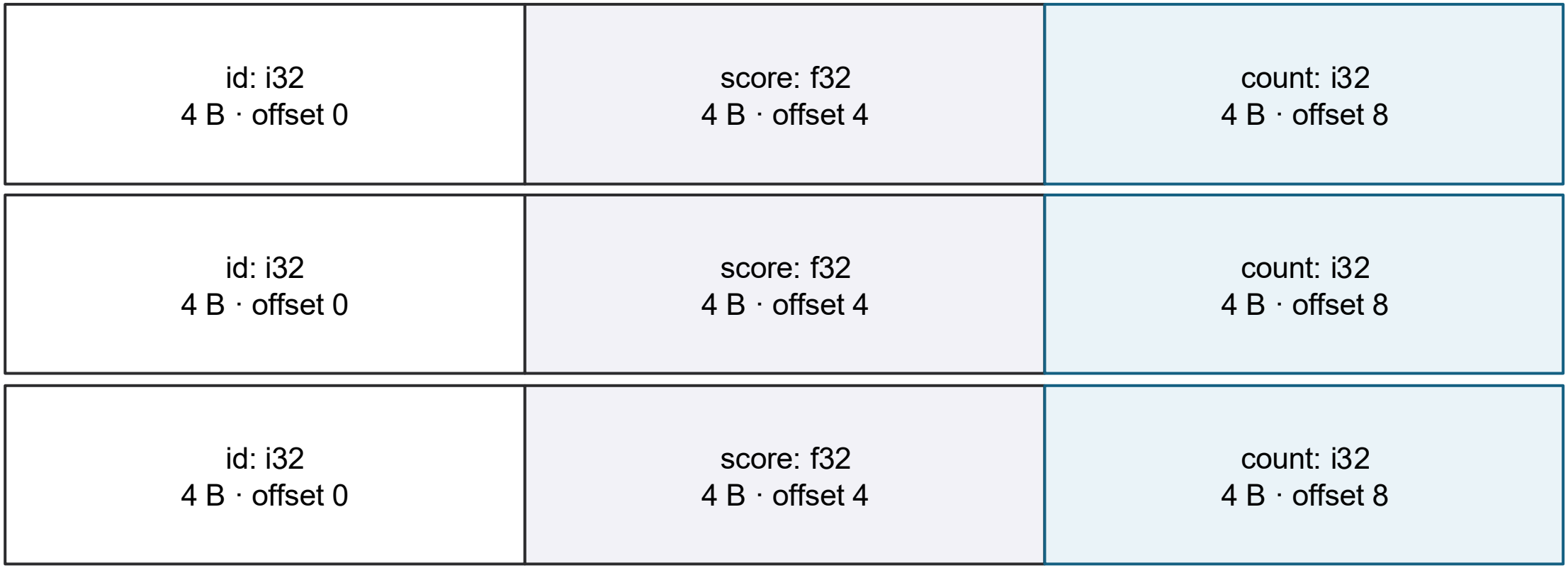



```
x = 10
x = 11.5
```

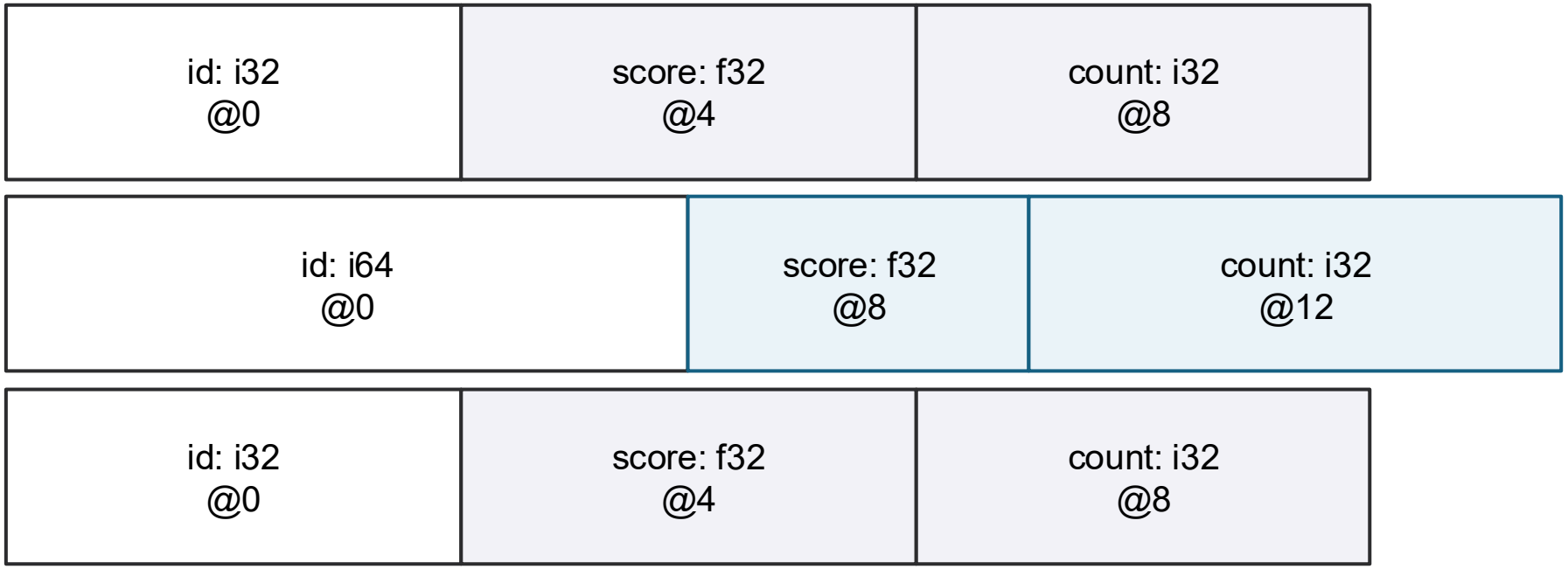
One name, two runtime types

A database stores bytes—in memory and in files.
Does this coercion cause use problem?

Stored rows are byte contracts



Update…


id: i32
@0
score: f32
@4
count: i32
@8
id: i64
@0
score: f32
@8
count: i32
@12
id: i32
@0
score: f32
@4
count: i32
@8

Debt paid …

C++

```
std::int32_t id = 10;
std::int64_t delta = 1;
auto total = id + delta;
```



i8 u8 f32
i16 u16 f64
i32 u32
i64 u64



int8_t uint8_t float32_t
int16_t uint16_t float64_t
int32_t uint32_t
int64_t uint64_t



Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
signed char	signed char	at least 8	8	8	8	8
unsigned char	unsigned char					
short	short int	at least 16	16	16	16	16
short int						
signed short						
signed short int						
unsigned short	unsigned short int					
unsigned short int						
int	int	at least 16	16	32	32	32
signed						
signed int						
unsigned	unsigned int					
unsigned int						
long	long int	at least 32	32	32	32	64
long int						
signed long						
signed long int						
unsigned long	unsigned long int					
unsigned long int						
long long	long long int (C++11)	at least 64	64	64	64	64
long long int						
signed long long						
signed long long int						
unsigned long long	unsigned long long int (C++11)					
unsigned long long int						

Rust: Static typing

Rust

```
let id: i32 = 10;
let delta: i64 = 1;
id + delta  // error
```

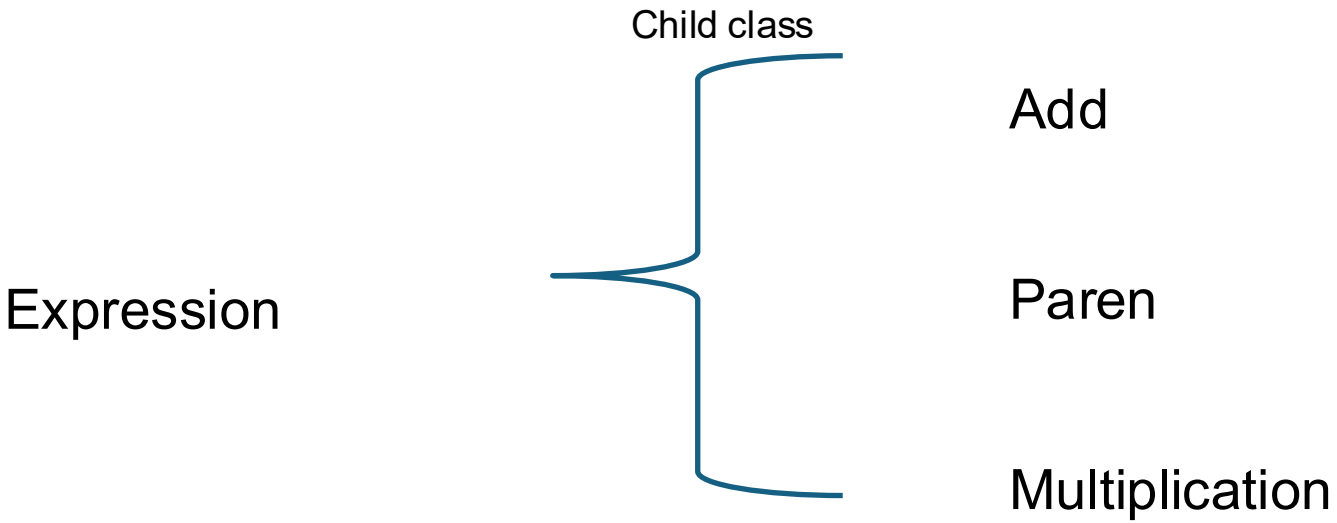
OOP: Class

```java
public class Car extends Person {
    public Car(double topSpeed, String driverName) {
        super(driverName);
        this.topSpeed = topSpeed;
    }
```



Representing Syntax Using Class?

(x + 2) * (x + 3)



Representing Syntax Using Class?

```
class Expr:
    def __add__(self, other):
        return Add(self, as_expr(other))

    def __radd__(self, other):
        return Add(as_expr(other), self)

    def __mul__(self, other):
        return Mul(self, as_expr(other))

    def __rmul__(self, other):
        return Mul(as_expr(other), self)


def as_expr(value):
    return value if isinstance(value, Expr) else Num(value)
```

```
@dataclass(frozen=True)
class Num(Expr):
    value: float


@dataclass(frozen=True)
class Var(Expr):
    name: str


@dataclass(frozen=True)
class Add(Expr):
    left: Expr
    right: Expr


@dataclass(frozen=True)
class Mul(Expr):
    left: Expr
    right: Expr
```

Traverse an Expression

```
def evaluate(expr, variables):
    # expr is type Expr, need know exact subtype
```

Use **isinstance** to check its subtype

Is subtyping a correct abstraction here?

“Industrial Quality OOP”

```
from abc import ABC, abstractmethod


class Expr(ABC):
    @abstractmethod
    def evaluate(self, env):
        pass
```

```
class Number(Expr):
    def evaluate(self, env):
        return self.value

class Add(Expr):
    pass

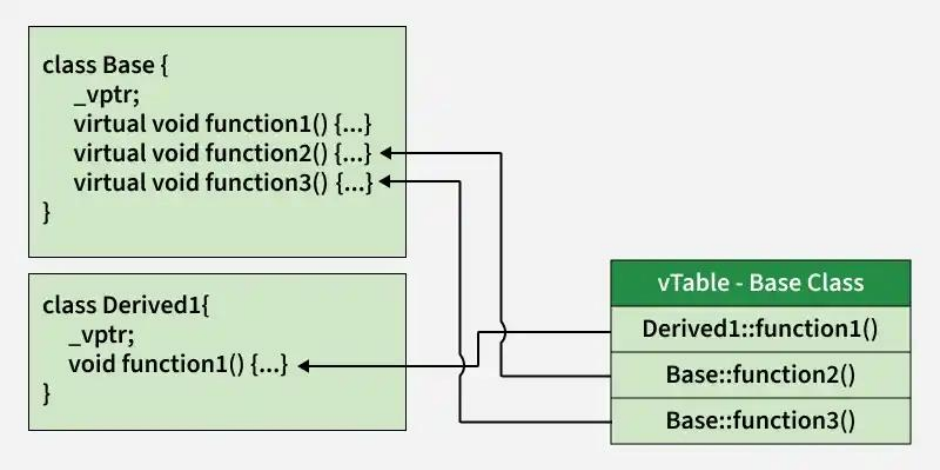
Add()  # TypeError
```

```
class Expr:
    def evaluate(self, env): ...
    def pretty_print(self): ...
    def type_check(self, context): ...
    def simplify(self): ...
    def compile(self, builder): ...
    def collect_variables(self): ...
    def serialize(self): ...
```



Performance Issue

- OOP with abstract class cause program relay on vtable, which add extra lookup every function call.
- System programmer will try to avoid it with …



Zero cost abstraction

Curiously Recurring Template Pattern

```
// The Curiously Recurring Template Pattern (CRTP)
template <typename T>
class Base {
    // methods within Base can use template to access members of Derived
};

class Derived : public Base<Derived> {
    // ...
};
```



Algebraic View: enum in rust

```
enum Expr {
    Number(i64),
    Add(Box<Expr>, Box<Expr>),
    Multiply(Box<Expr>, Box<Expr>),
}
```

- All expression only have one type `Expr`
- ADT in some FP lang
- Each sub case is called **type constructor**

Function + Pattern Match

```
fn evaluate(expr: &Expr) -> i64 {
    match expr {
        Expr::Number(value) => *value,
        Expr::Add(left, right) =>
            evaluate(left) + evaluate(right),
        Expr::Multiply(left, right) =>
            evaluate(left) * evaluate(right),
    }
}
```

- Clean way to inspect into ADT
- Compiler know if you are missing a matching arm!

Trait + Pattern Match

```
trait PrettyPrint {
    fn pretty_print(&self) -> String;
}
```

```
impl PrettyPrint for Expr {
    fn pretty_print(&self) -> String {
        match self {
            Expr::Number(n) => n.to_string(),

            Expr::Add(left, right) =>
                format!("({} + {})",
                    left.pretty_print(),
                    right.pretty_print()),

            Expr::Multiply(left, right) =>
                format!("({} * {})",
                    left.pretty_print(),
                    right.pretty_print()),
        }
    }
}
```

Generic and Template

We are not just going to evaluate single syntax in our approach, what if we want a general way to do evaluation with zero-cost abstraction?

```
template<
    typename T,
    typename = decltype(
        std::declval<const T&>().evaluate()
    )
>
Value run(const T& expression) {
    return expression.evaluate();
}
```



Generic and Trait

All you care is if **T** has some properties.

```
fn run<T: Evaluate>(expression: &T) {
    // specialized generic code
}
```

Next class

Formal Semantics of Conjunctive Queries

Review before next class

Open these two short course notes:

Database 01 — A Little Database, A Few Questions

Rust cheat sheet — A Little Rust, Side by Side

Reading before next class

Abiteboul, Hull, and Vianu · Foundations of Databases

Chapter 4: Conjunctive Queries · §§4.1–4.2

Relational-model review · §§3.1–3.3