ADTs

The BankAccount ADT

Implementation

≈

Bank System

Interface

≈

Bank Teller/
ATM

User

≈

Customer

BankAccount.h (Interface)

```c
#ifndef BANK_ACCOUNT_H
#define BANK_ACCOUNT_H

typedef struct bankAccount *BankAccount;

BankAccount newBankAccount(void);
void deposit(BankAccount acc, int amount);
void withdraw(BankAccount acc, int amount);
int getBalance(BankAccount acc);

#endif
```

Users can only see the interface

Implementation

≈

Bank System

Interface

≈

Bank Teller/
ATM

User

≈

Customer

accountUser.c (User)

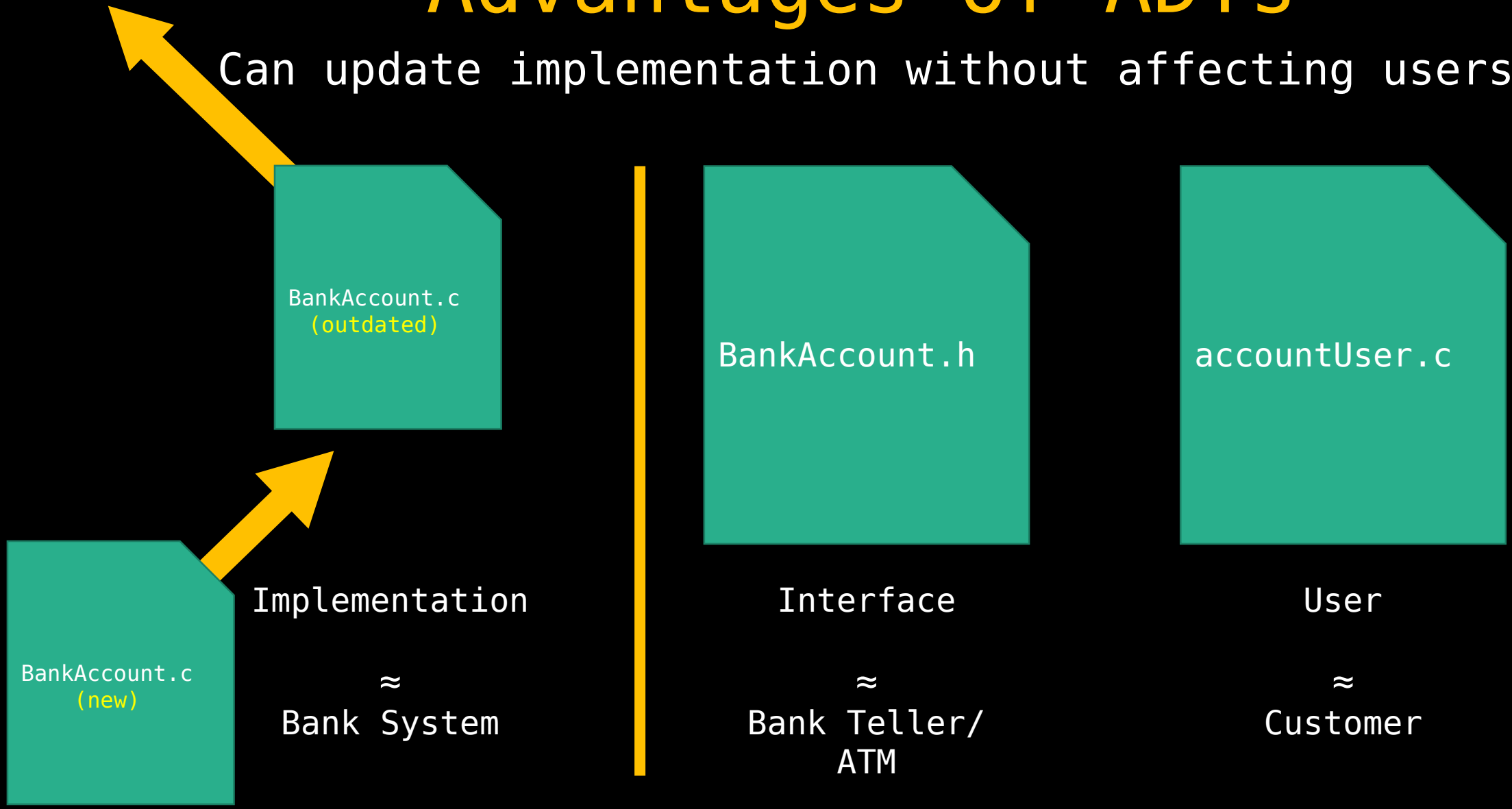
```c
#include “BankAccount.h”

int main(void) {
    BankAccount account = newBankAccount();

    // account->balance = 1000000;
    // ^ You can’t do this. Why?
    // You’ll get a compile error like:
    // error: dereferencing pointer to incomplete type
    // ’struct bankAccount’

    deposit(account, 150);
    // ^ You can do this, as deposit is in BankAccount.h

    ...
}
```

BankAccount.c (Implementation)

```c
#include “BankAccount.h”

struct bankAccount {
    int amount;
};

BankAccount newBankAccount(void) {
    BankAccount new = malloc(sizeof(*new));
    ...
    new->amount = 0;
    return new;
}

void deposit(BankAccount acc, int amount) {
    ...
}
...
```

Advantages of ADTs

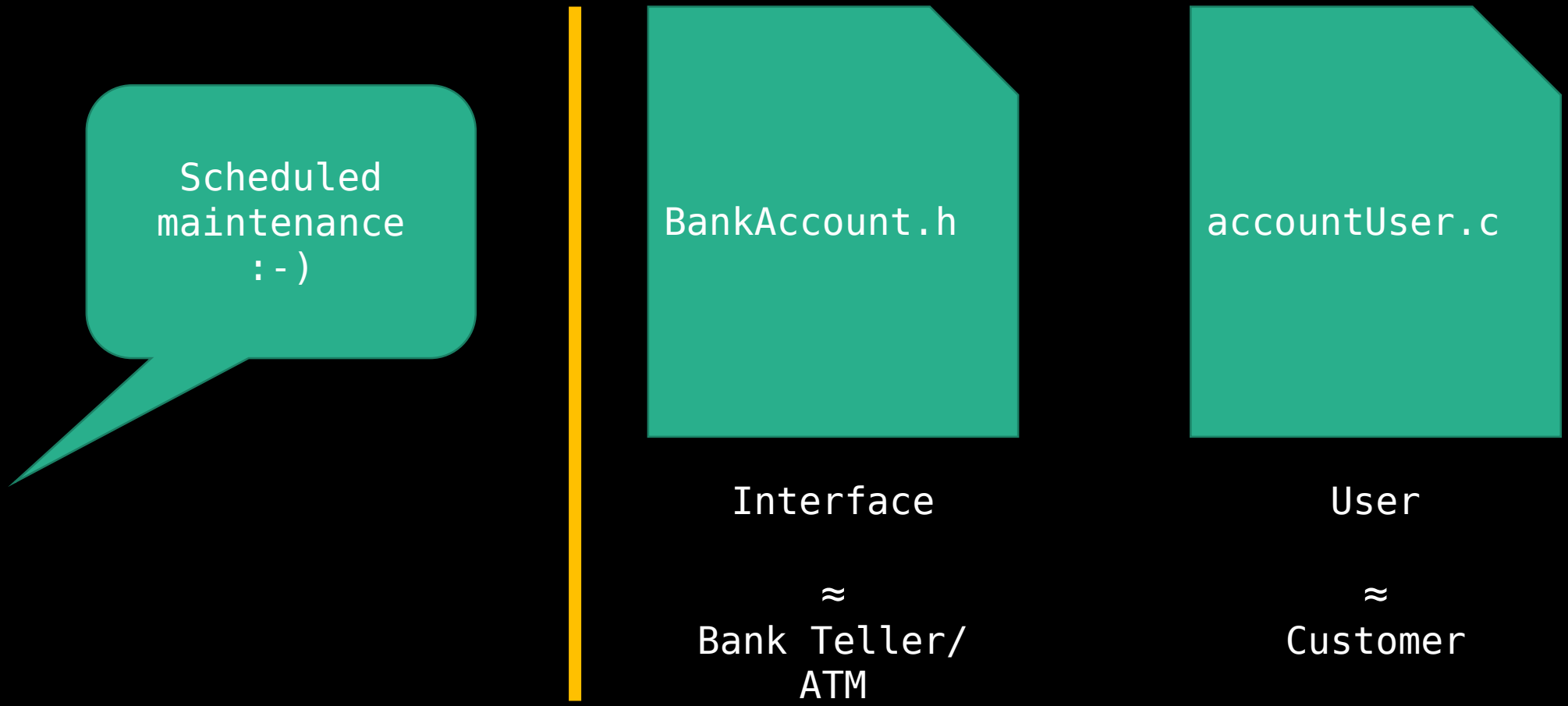
Can update implementation without affecting users



Advantages of ADTs

Can update implementation without affecting users

What the user sees...



Advantages of ADTs

Users don’t need to see the implementation*
*as long as interface has sufficient documentation

Implementation could be complete spaghetti...

BankAccount.h

Interface

≈

Bank Teller/ ATM

accountUser.c

User

≈

Customer

Advantages of ADTs

Separation of concerns
User does not need to care about the internal workings of the ADT.