CMSC 430 - 8 Sept 2026

Our first compilers

- Quick Racket and a86 review
- Abscond - integer literals
- Blackmail - unary primitives
- Con - conditional expressions
- Dupe - booleans, revised conditional

CMSC 430 - 8 Sept 2026

Announcements

- Assignment 1 due Thurs, midnight
- New quiz released today, due by start of class Thursday
- TA Office Hours published on course web page

Abscond

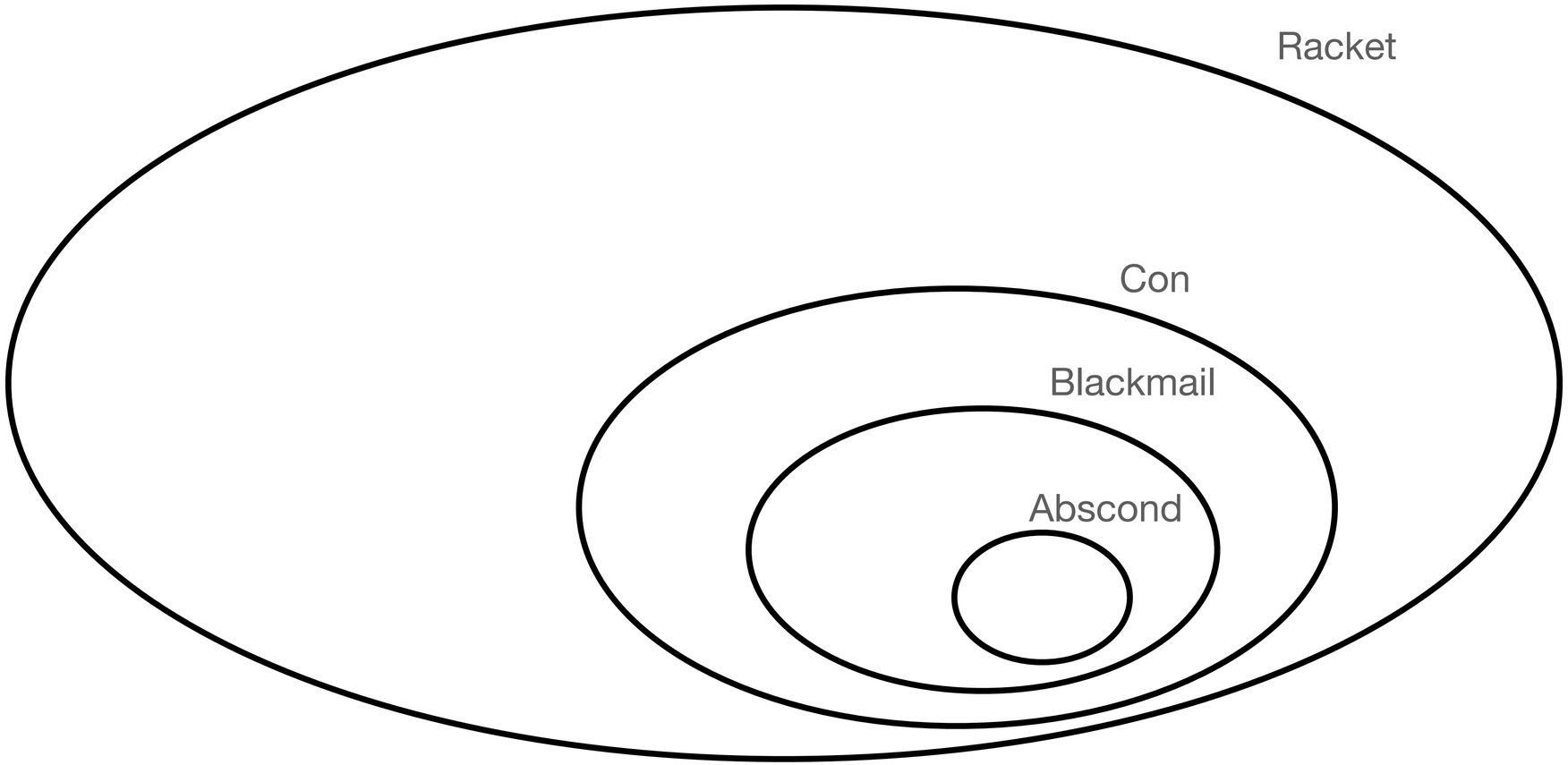
Learning objectives

Abscond

You should be able to answer:

- What is the structure of the Abscond compiler?
- How are Abscond programs represented abstractly?
- What invariant relates source programs and compiled code?

Language subsets



Example Abscond program

Concrete syntax

```
#lang racket
42
```

Example Abscond program

Abstract syntax

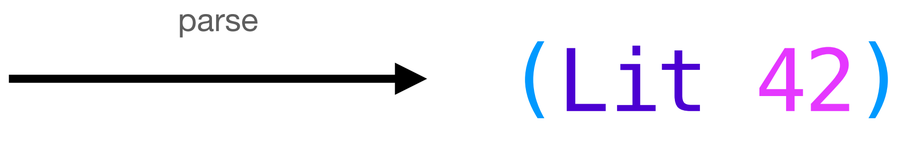
```
(Lit 42)
```

```
;; type Expr =
;; | (Lit Integer)
(struct Lit (n))
```

Example Abscond program

Parser constructs AST from string input

```
#lang racket
42
```



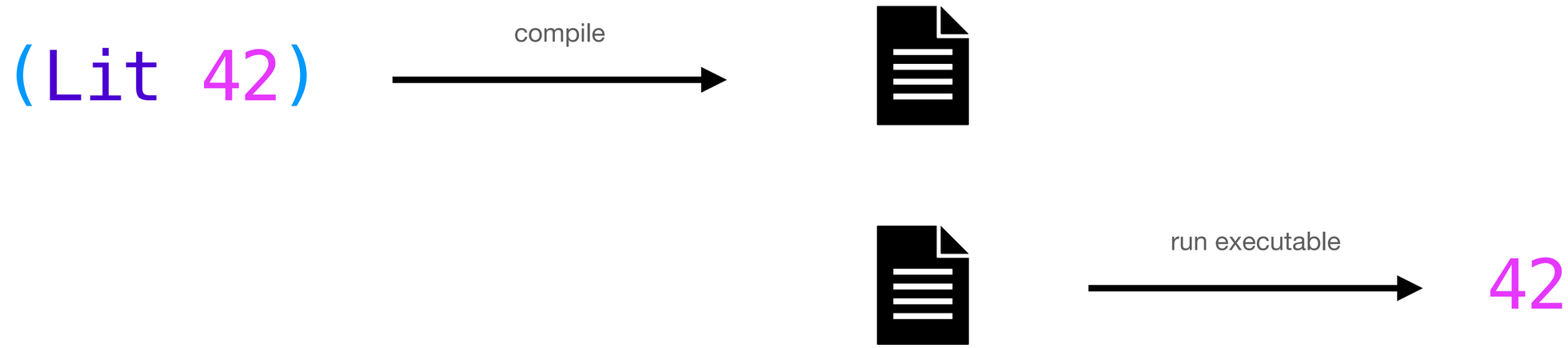

```
(Lit 42)
```

Example Abscond program

Interpreter computes meaning from AST

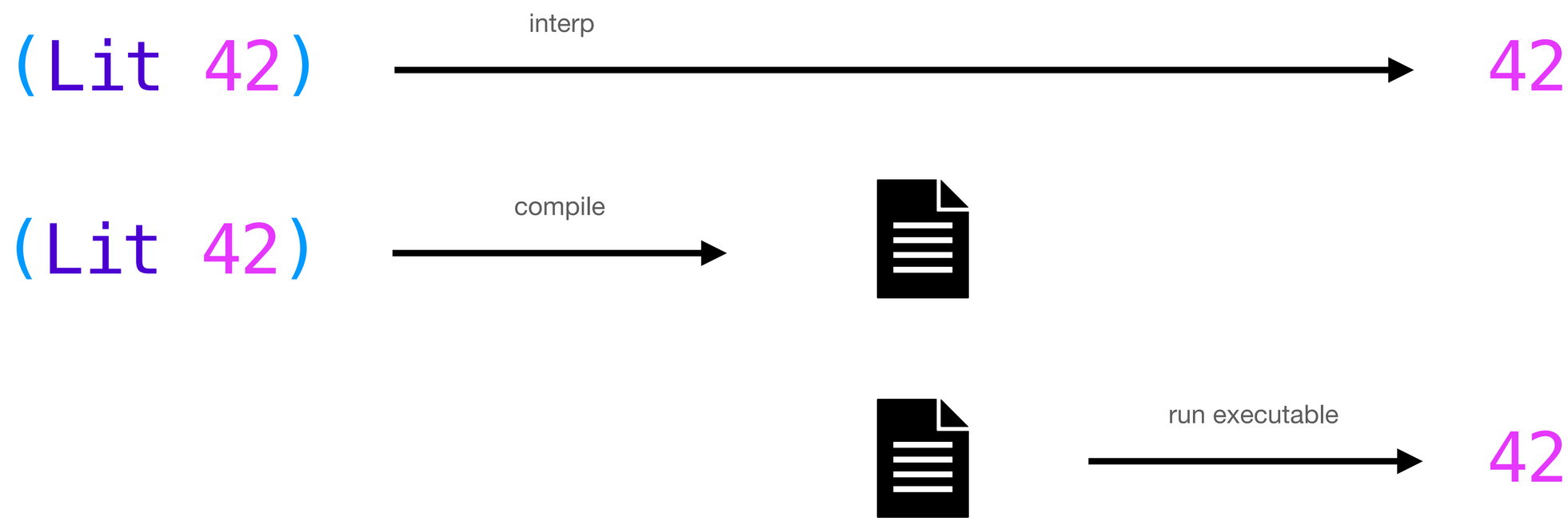
Example Abscond program

Compiler computes a program that computes meaning



Example Abscond program

Compiler specification



Example Abscond program

Our approach to parsing: reader + parser

```
#lang racket
42
```


```
42
```


```
(Lit 42)
```

String

Semi-structured data;
ensures parens balanced, etc.

S-Expr

Expr

Structured data; ensures
grammatically well-formed

Example Abscond program

Our approach to compiling: compiler + stand-alone runtime

```
(Lit 42)
```

compile →

```
(Global 'entry)
(Label 'entry)
(Mov rax 42)
(Ret)
```

asm-display →

```
        .intel_syntax noprefix
        .text
        .global "_entry"
"_entry":
        mov rax, 42
        ret
```

Expr

Construct AST of assembly program

Asm

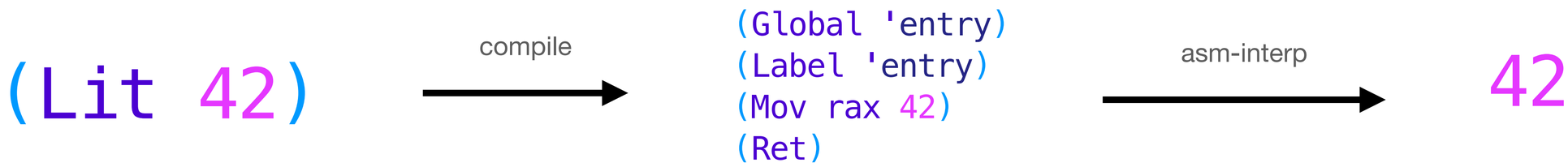
.s

Print to standard assembly format; call assembler

Link w/ stand-alone runtime system

Example Abscond program

Our approach to testing: compiler + asm-interp



Expr

Asm

Value

Construct AST of assembly program

Run compiled program to compute value

Abscond

At the heart of it

```
;; Expr -> Asm
(define (compile e) ...)
;; Asm -> Value
(define (run a) ...)

;; Expr -> Value
(define (interp e) ...)

;; compile ∘ run ≡ interp
```

Abscond interpreter

At the heart of it

```
;; Expr -> Integer
(define (interp e)
  (match e
    [(Lit i) i]))
```

Abscond compiler

At the heart of it

```
;; Expr -> Asm
(define (compile e)
  (list (Global 'entry)
        (Label 'entry)
        (match e
          [(Lit i) (Mov rax i)])
        (Ret)))
```

Abscond correctness

At the heart of it

```
;; Expr -> Void
(define (check-compiler e)
  (check-equal? (interp e)
                (asm-interp (compile e))))
```

Hosted vs Standalone Runtime System

The runtime system is code that is needed at runtime to assist execution of compiled code. We will use two different runtime systems:

- Hosted: written in Racket w/ asm-interp; requires Racket at runtime
- Stand-alone: written in C; doesn't require Racket at runtime

Mostly we used hosted for development and testing, but the standalone is useful for when you want to make standalone executables.

Abscond directories

- `syntax/` - AST definition, parser
- `interpreter/` - interpreter, command line interpreter
- `compiler/` - compiler, command line compiler
- `executor/` - hosted runtime, command line executor
- `runtime/` - standalone runtime
- `test/` - unit tests

Blackmail

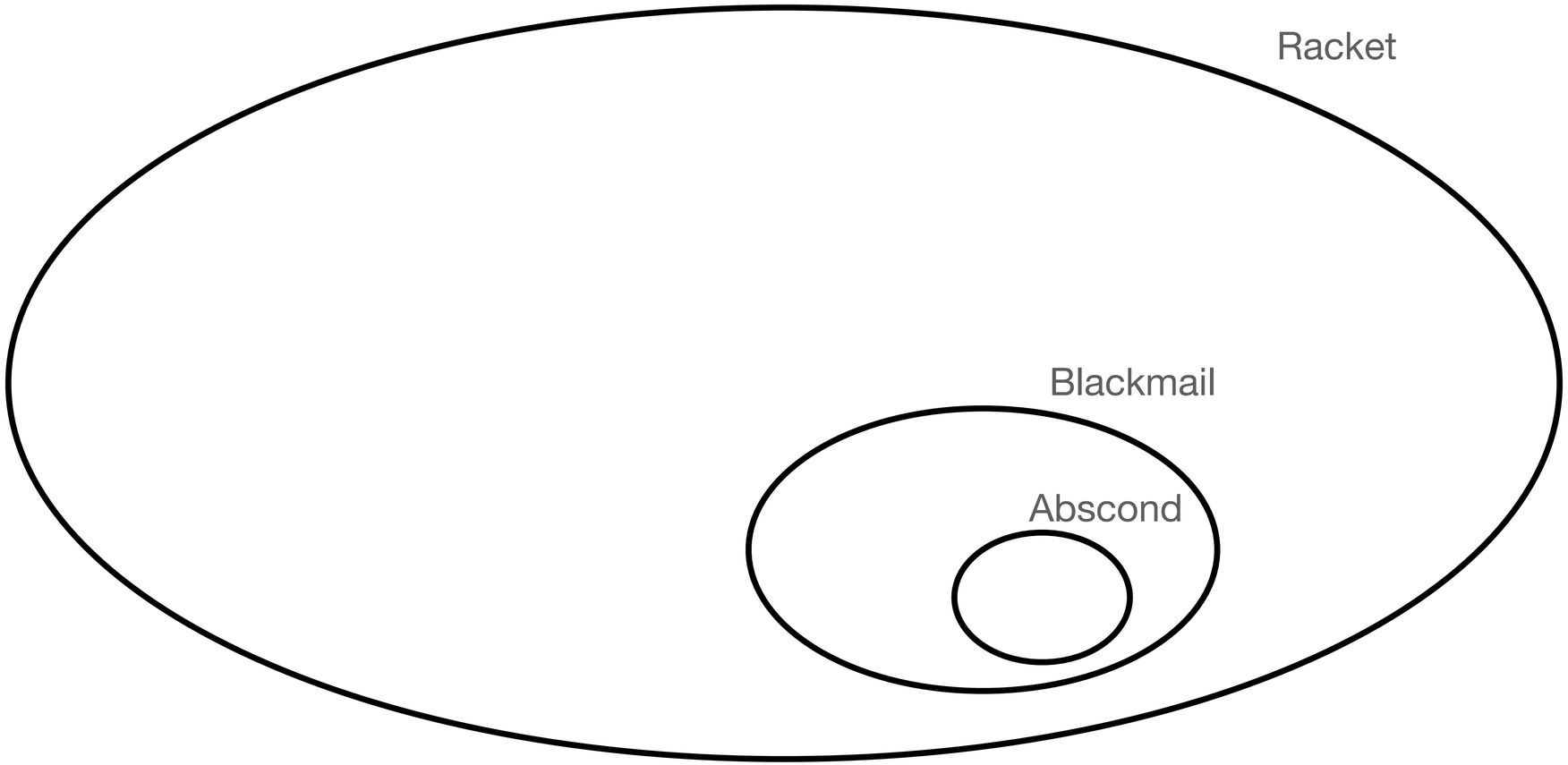
Learning objectives

Blackmail

You should be able to answer:

- How are Blackmail programs represented abstractly?
- What's the process for systematically growing our language?
- How do we structure an interpreter and compiler for an inductively defined AST data type?
- How do we both use and ensure the compiler invariants?

Language subsets



Recipe for growing a language

- Write examples
- Extend concrete syntax
- Extend abstract syntax
- Extend parser
- Revise interpreter to specify semantics
- Revise compiler & run-time system to implement semantics
- Test against examples
- *Rinse and repeat*

Example Blackmail program

Concrete syntax

```
#lang racket
(add1 (add1 40))
```

Example Blackmail program

Abstract syntax

```
(Prim1 'add1 (Prim1 'add1 (Lit 40)))
```

```
;; type Expr = (Lit Integer)
;;           | (Prim1 Op1 Expr)
;; type Op1 = 'add1 | 'sub1
(struct Lit (i))
(struct Prim1 (p e))
```

Example Blackmail program

Interpreter computes meaning from AST

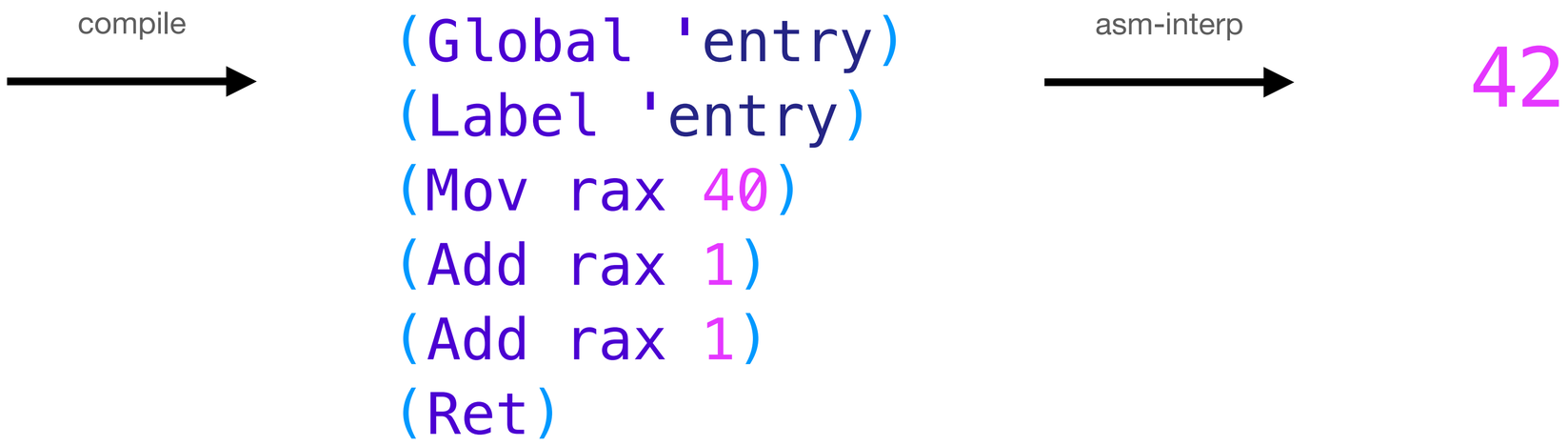
```
(Prim1 'add1 (Prim1 'add1 (Lit 40)))
```


```
42
```

Example Blackmail program

Interpreter computes meaning from AST

```
(Prim1 'add1 (Prim1 'add1 (Lit 40)))
```



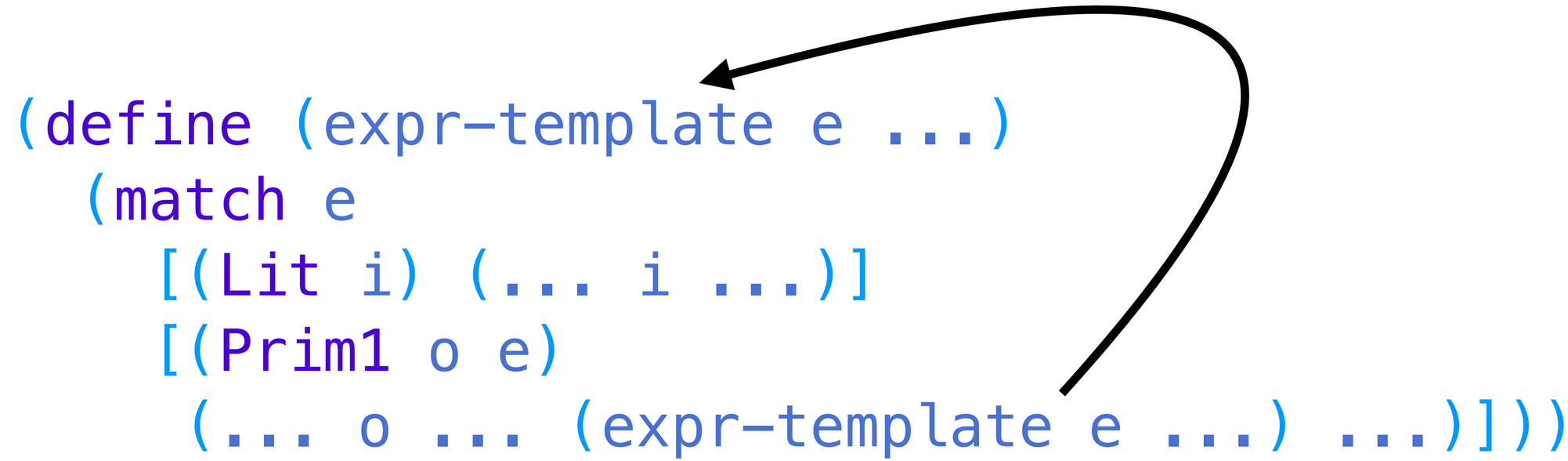
Expr is an inductive data type

Abstract syntax

```
;; type Expr = (Lit Integer)
;;           | (Prim1 Op1 Expr)
;; type Op1 = 'add1 | 'sub1
(struct Lit (i))
(struct Prim1 (p e))
```

Expr is an inductive data type

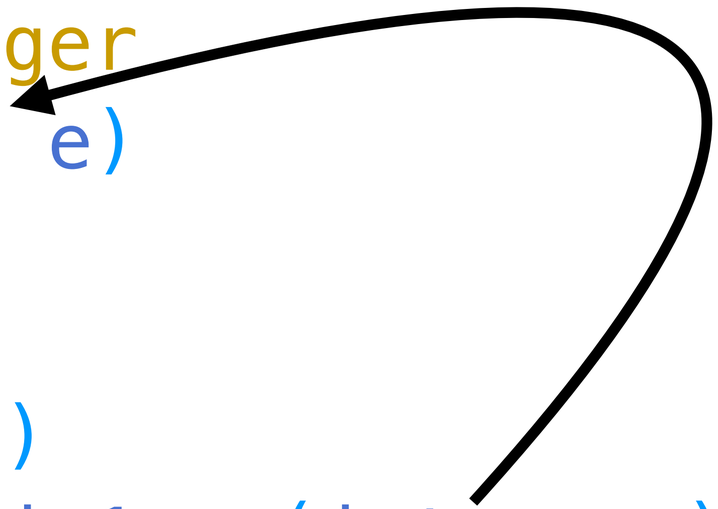
Abstract syntax



```
(define (expr-template e ...)
  (match e
    [(Lit i) (... i ...)]
    [(Prim1 o e)
     (... o ... (expr-template e ...) ...)]))
```

Blackmail interpreter

```
;; Expr -> Integer
(define (interp e)
  (match e
    [(Lit i) i]
    [(Prim1 p e)
     (interp-prim1 p (interp e))]))
```



Blackmail interpreter

```
;; Op1 Integer -> Integer
(define (interp-prim1 op i)
  (match op
    ['add1 (add1 i)]
    ['sub1 (sub1 i)]))
```

Blackmail compiler

Compiling programs

```
;; Expr -> Asm
(define (compile e)
  (prog (Global 'entry)
        (Label 'entry)
        (compile-e e)
        (Ret)))
```

Blackmail compiler

Key invariant

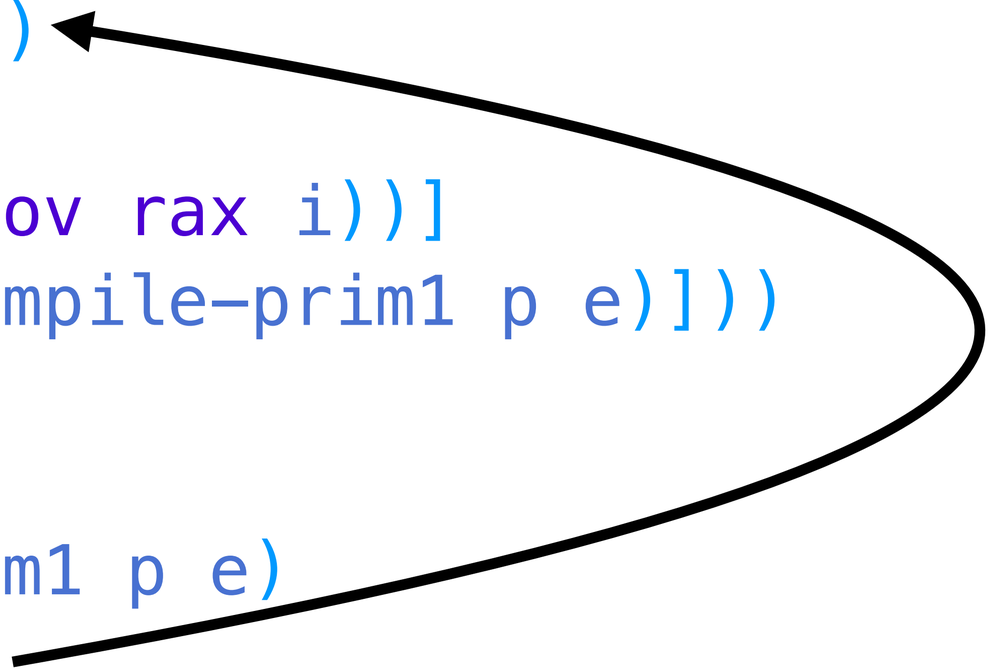
```
;; Invariant:
;; Produces instructions that when run,
;; leave e's value in rax
(compile-e e)
```

Blackmail compiler

Compiling expressions

```
;; Expr -> Asm
(define (compile-e e)
  (match e
    [(Lit i) (seq (Mov rax i))]
    [(Prim1 p e) (compile-prim1 p e)]))

;; Op1 Expr -> Asm
(define (compile-prim1 p e)
  (seq (compile-e e)
       (compile-op1 p)))
```



Blackmail compiler

Compiling primitives

```
;; Op1 -> Asm
(define (compile-op1 p)
  (match p
    ['add1 (Add rax 1)]
    ['sub1 (Sub rax 1)]))
```

Blackmail compiler

Example

```
> (asm-interp (compile (parse '(add1 (add1 40)))))
42
```

For next time

- Read Con, Dupe notes
- Study Abscond, Blackmail notes notes
- Take ELMS quiz

CMSC 430 - 10 Sept 2026

Our first compilers

- Quick Blackmail review
- Con - conditional expressions
- Dupe - booleans, revised conditional