Building Typed Languages with Turnstile

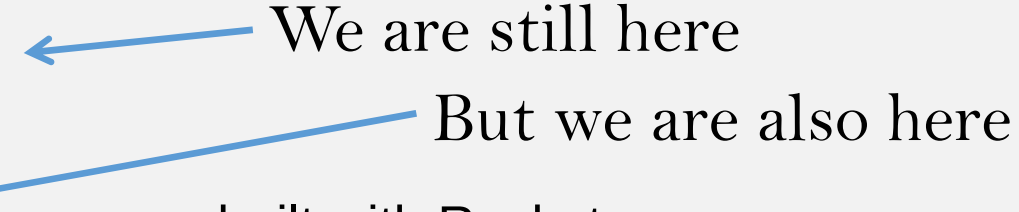
Stephen Chang

Racket Summer School 2018

(Thursday afternoon)

Where are we in the schedule?

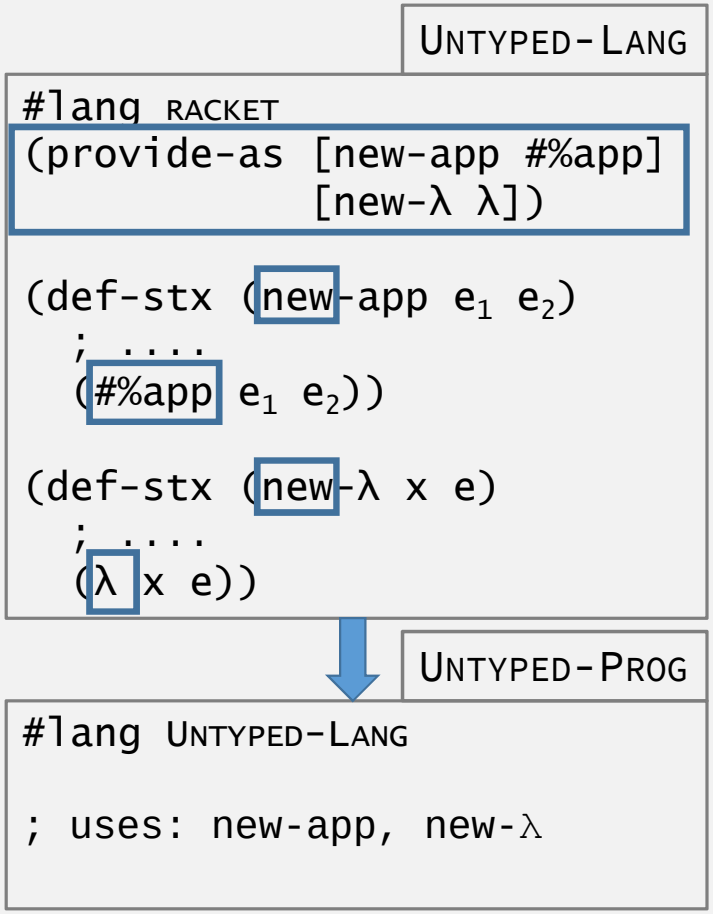
- Monday, Tuesday, Wednesday:
  - Tools for building languages with Racket
- Thursday:
  - How to build *typed* languages
- Thursday and Friday:
  - Let's look at some *example* languages built with Racket



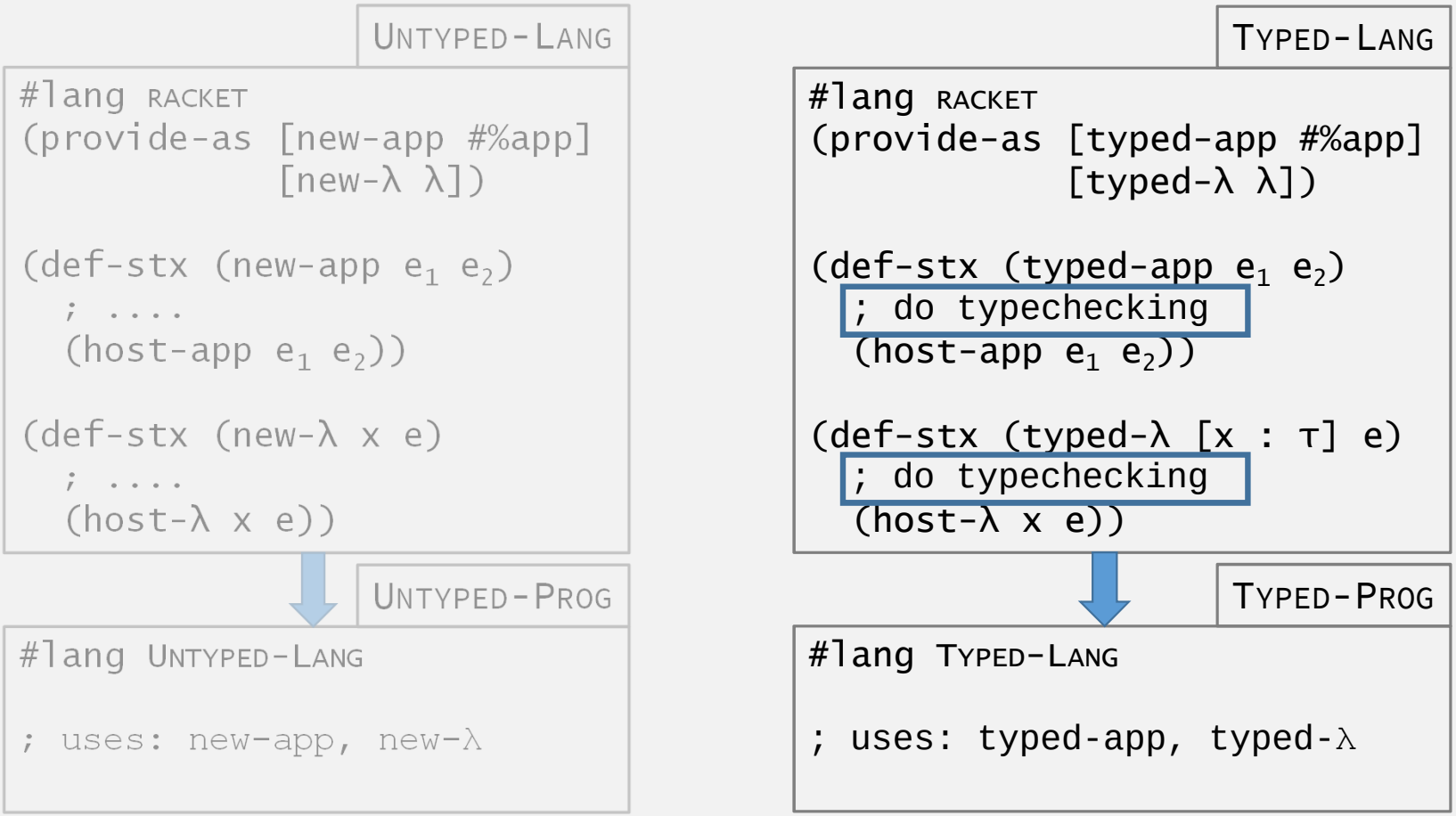
The language of type rules … is a language

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_{in} \rightarrow \tau_{out} \qquad \Gamma \vdash e_2 \Leftarrow \tau_{in}}{\Gamma \vdash e_1 e_2 \Rightarrow \tau_{out}}$$

A Macro-based Language in Racket



IDEA: A Typed Macro-based Language



“do typechecking” = expand + stx props

```
(define-syntax (typed-app stx)
 (syntax-parse stx
  [(_ e1 e2)
   #:with e3 (local-expand #'e1)
   #:with (→ τin τ) (stx-prop #'e3'ty)
   #:with e4 (local-expand #'e2)
   #:with τarg (stx-prop #'e4 'ty)
   #:fail-unless (stx= #'τarg #'τin)
   (stx-prop #'(#%app e3 e4) 'ty #'τ)]))
```

Compute type

Check type

Assign type

Types checked when macro expands

Leverage Domain-specific Syntax

$$\text{[T-App]}\ \frac{\Gamma \vdash e_1 \Rightarrow \tau_{in} \rightarrow \tau_{out} \quad \Gamma \vdash e_2 \Leftarrow \tau_{in}}{\Gamma \vdash e_1 e_2 \Rightarrow \tau_{out}}$$

Compute type

Check type

Assign type

TURNSTILE: Type and Rewrite Rules

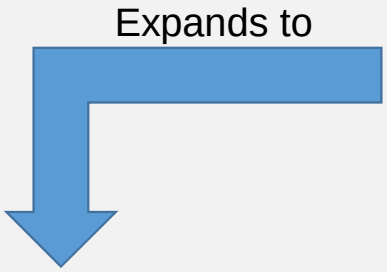
$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_{in} \rightarrow \tau_{out} \qquad \Gamma \vdash e_2 \Leftarrow \tau_{in}}{\Gamma \vdash e_1\, e_2 \Rightarrow \tau_{out}}$$

```
(def-typed-stx (typed-app e1 e2) »
  [⊢ e1 » e1' ⇒ (→ τin τout)]
  [⊢ e2 » e2' ⇐ τin]
  ----------------------------
  [⊢ (#%app e1' e2') ⇒ τout])
```

Conclusion: inputs

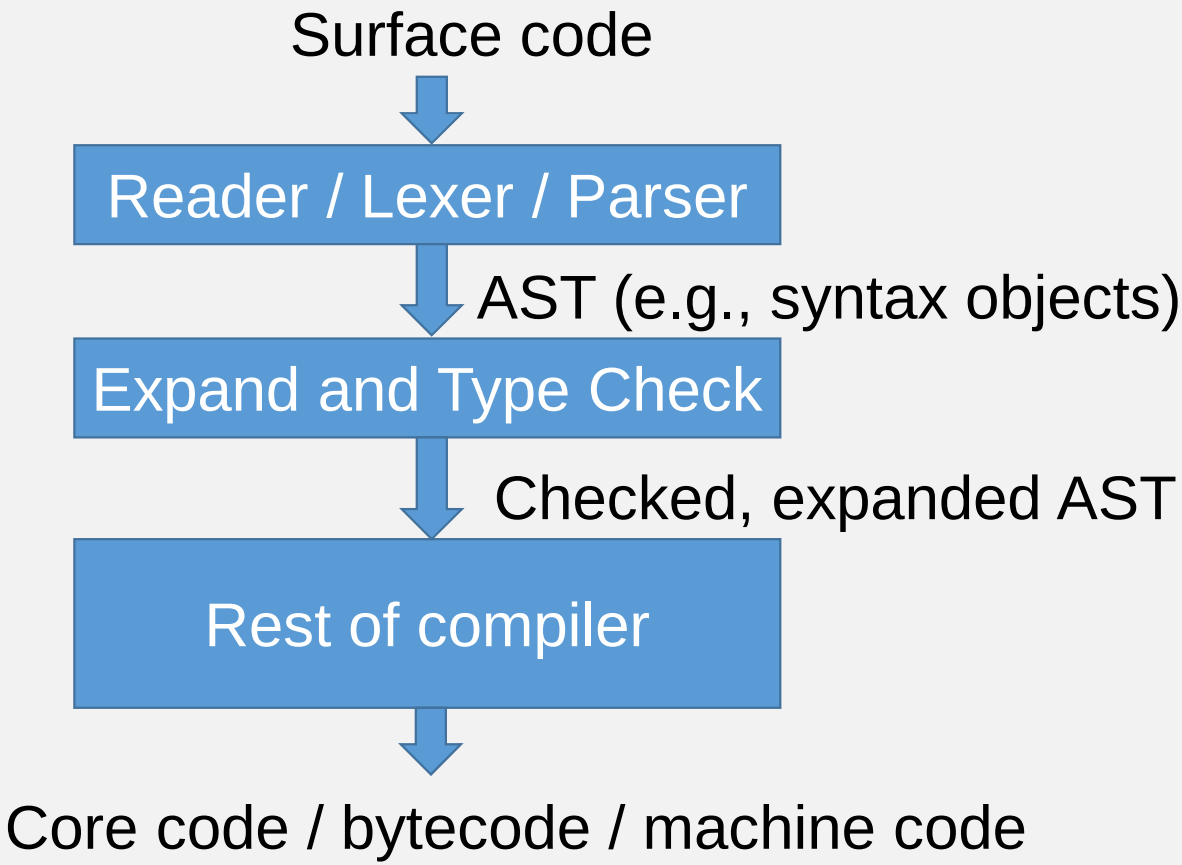
Conclusion: outputs

TURNSTILE: Type and Rewrite Rules




```
(def-typed-stx (typed-app e₁ e₂) »
  [⊢ e₁ » e₁' ⇒ (→ τ_in τ_out)]
  [⊢ e₂ » e₂' ⇐ τ_in]
  ---------------------------
  [⊢ (#%app e₁' e₂') ⇒ τ_out])
```

```
(def-stx (typed-app e₁ e₂)
  #:with e₁' (expand e₁)
  #:with (→ τ_in τ_out) (detach e₁')
  #:with e₂' (expand e₂)
  #:with τ_arg (detach e₂')
  #:fail-unless (stx= τ_arg τ_in)
  (attach (#%app e₁' e₂') τ_out)
```


Surface code
Reader / Lexer / Parser
AST (e.g., syntax objects)
Expand and Type Check
Checked, expanded AST
Rest of compiler
Core code / bytecode / machine code