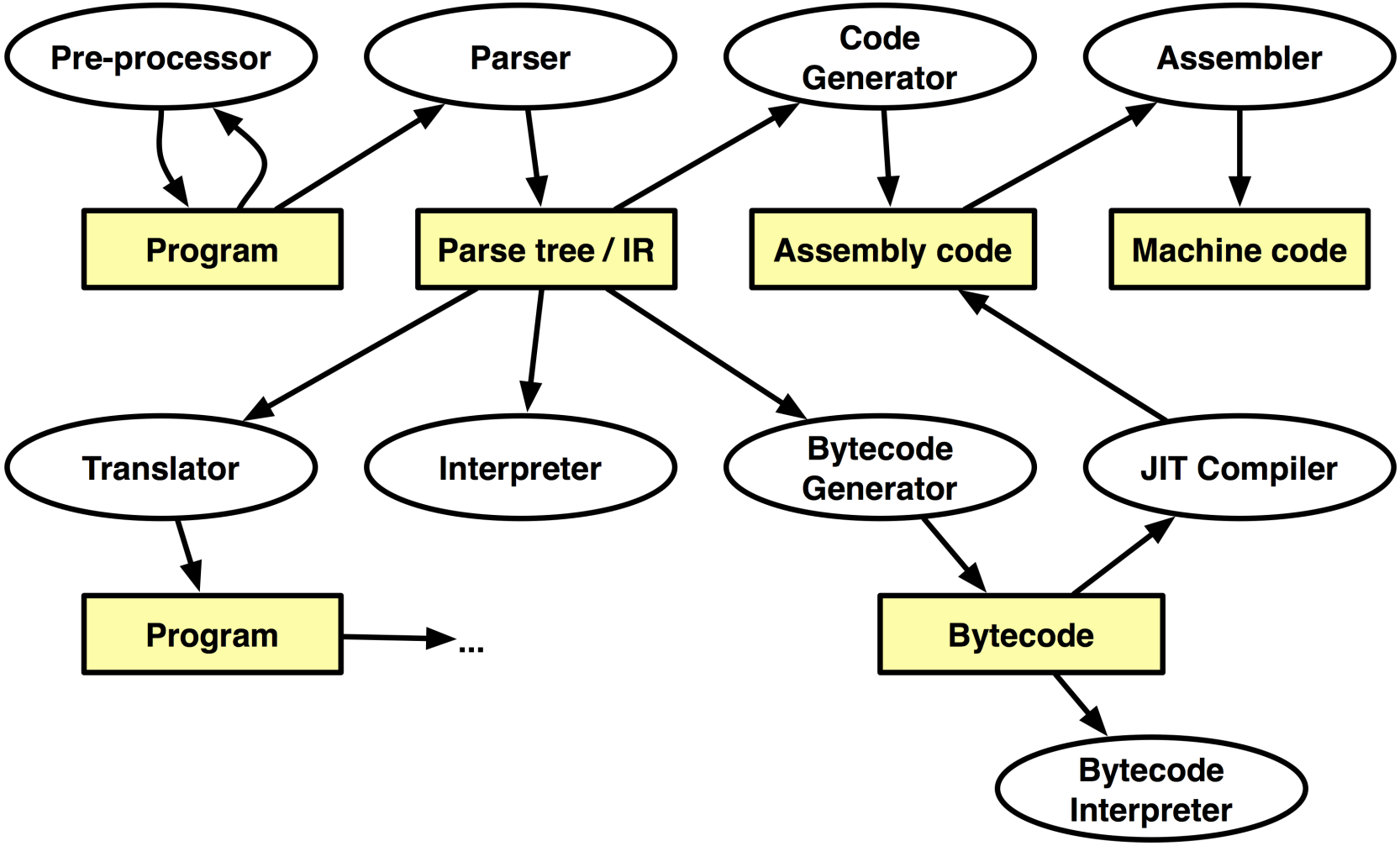
Construção de Compiladores

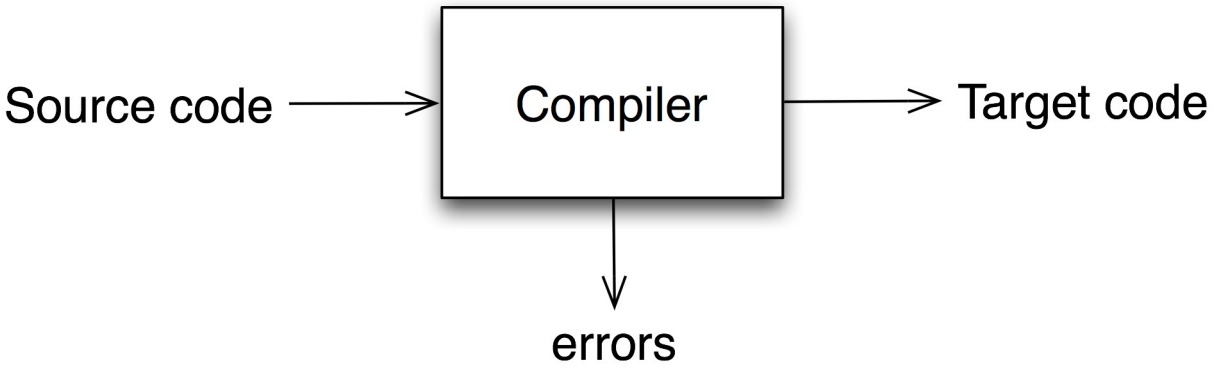
Introdução

Compiladores, Interpretadores …



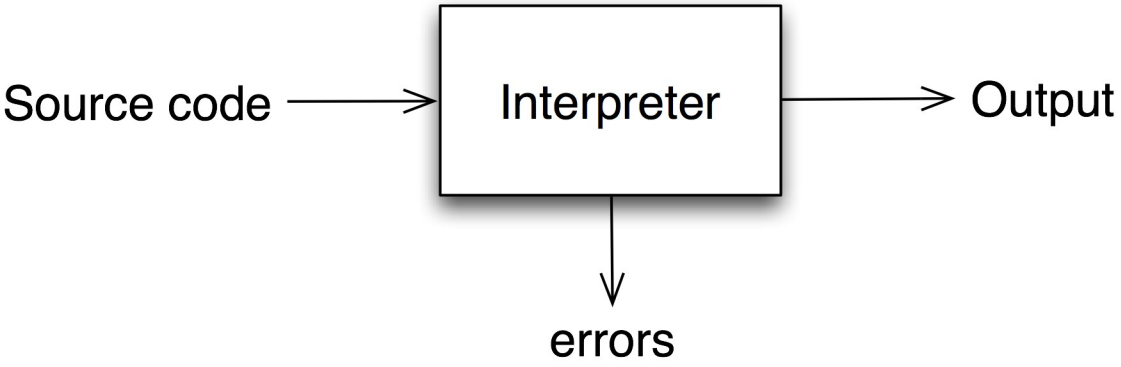
O que é um compilador?

Um programa que traduz um programa *executável* em uma linguagem em um programa *executável* em uma linguagem



O que é um interpretador?

Um programa que lê um programa *executável* e produz os resultados da execução do programa



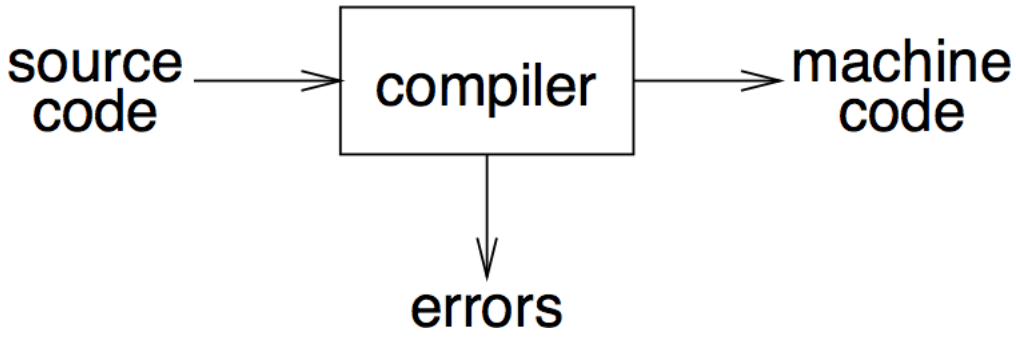
Quais qualidades são importantes em um compilador?

1. Código correto
2. A saída deve executar rapidamente
3. O compilador deve executar rapidamente
4. O tempo de compilação deve ser proporcional ao tamanho do código
5. Suporte para compilação em separado
6. Bom diagnóstico para erros sintáticos
7. Funcionar bem com um depurador
8. Bom diagnóstico para anomalias de fluxo
9. Chamadas entre linguagens
10. Otimização consistente e previsível

Um pouco de história

- **1952:** primeiro compilador (linker/loader) escrito por Grace Hopper para a linguagem de programação **A-0**
- **1957:** primeiro compilador completo para **FORTRAN** escrito por John Backus e sua equipe
- **1960:** compiladores **COBOL** para múltiplas arquiteturas
- **1962:** primeiro compilador autocontido para **LISP**

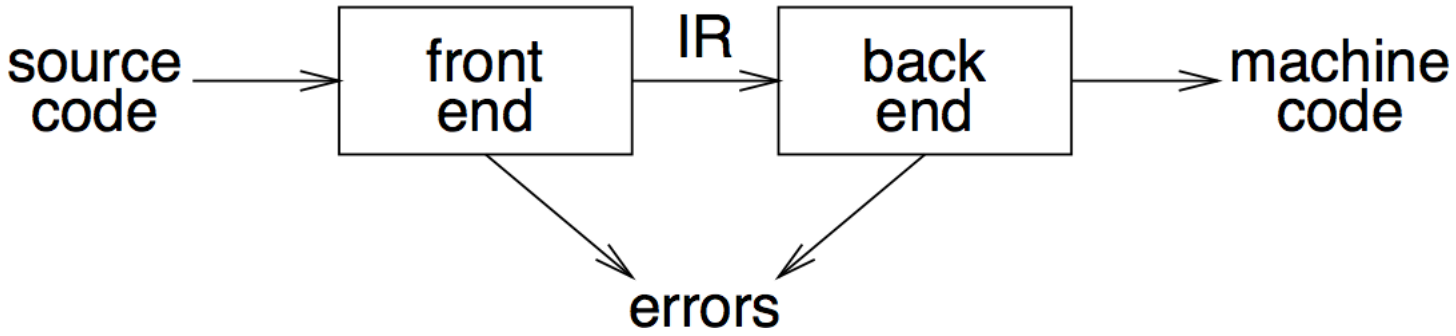
Visão abstrata



- Reconhecimento de programas legais (e illegais)
- Geração de código correto
- Gerenciamento da armazenagem de todas as variáveis e código
- Acordo sobre o formato do código objeto (ou assembly)

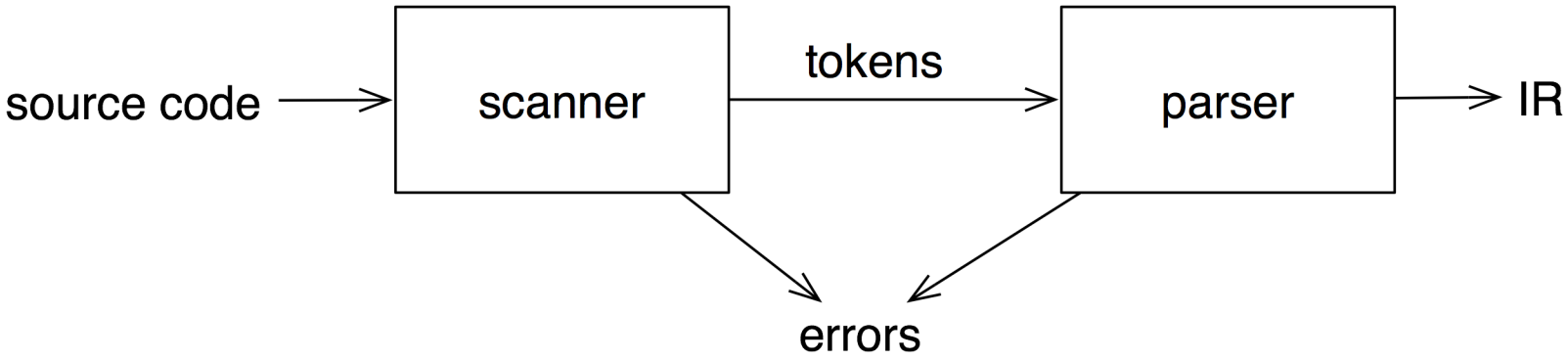
Um grande passo do assembler — notações de alto nível

Compilador tradicional de dois passos



- Representação intermediária (RI)
- Vanguarda mapeia código legal em RI
- Retaguarda mapeia RI para a máquina alvo
- Simplifica o redirecionamento
- Permite múltiplas vanguardas
- Múltiplos passos ⇒ melhor código

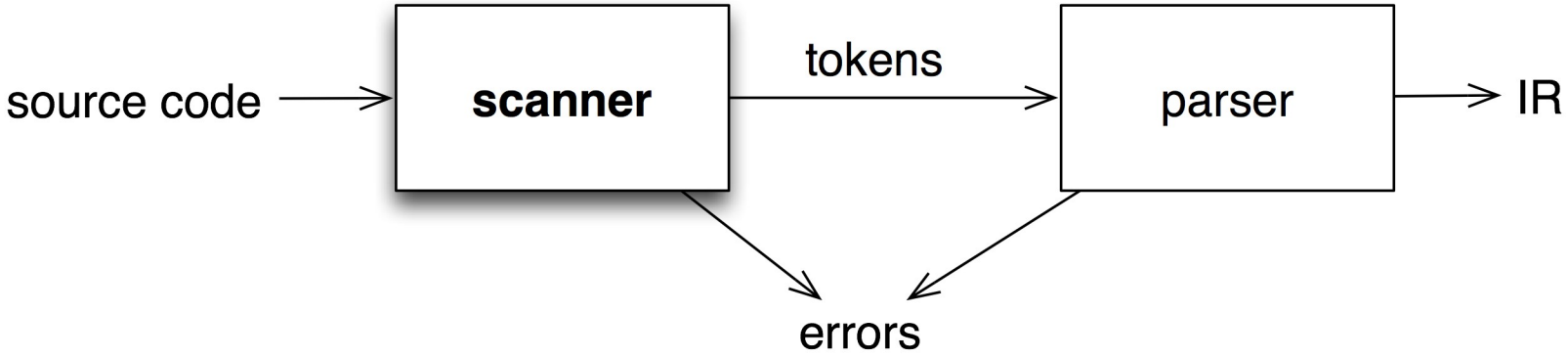
Vanguarda



- Reconhece código legal
- Relata erros
- Produz RI
- Mapa preliminar de armazenagem
- Formata o código para a retaguarda

Muitas tarefas da construção da vanguarda podem ser automatizadas

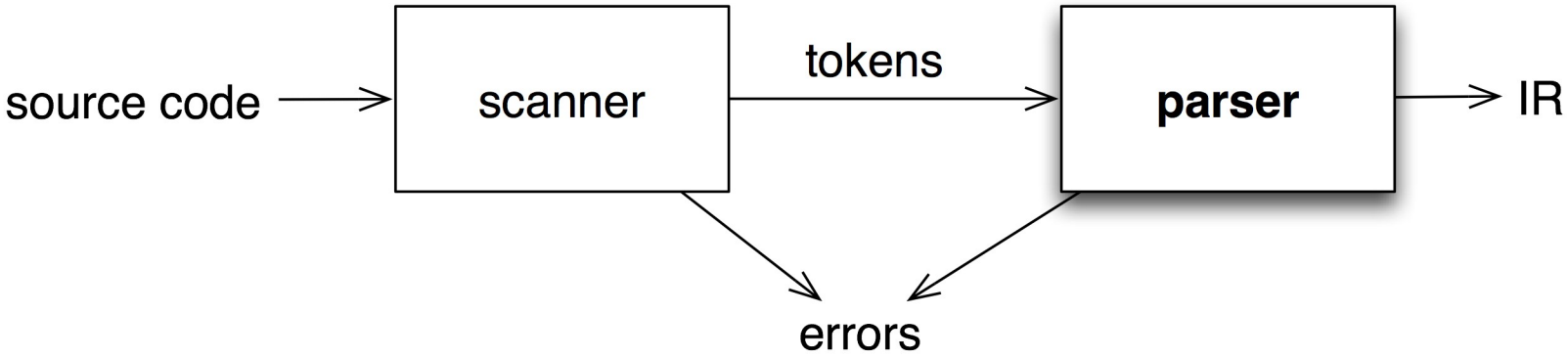
Analisador léxico



- Mapeia caracteres em *tokens*
- O valor em forma de uma string de caracteres de um token é chamado de *lexema*
- elimina espaços em branco

```
x = x + y  →  <id,x> = <id,x> + <id,y>
```

Analisador sintático



- Reconhece sintaxe livre de contexto
- Guia a análise sensível a contexto
- constrói RI(s)
- Produz mensagens de erros significativas
- Tenta correção de erro

Os geradores de parsers mecanizam muito do trabalho

Gramáticas livres de contexto

A *sintaxe livre de contexto* é especificada com uma gramática, normalmente na forma de *Backus-Naur* (BNF)

```
1. <objetivo> := <expr>
2. <expr>  := <expr> <op> <term>
3.         |  <term>
4. <term>  := numero
5.         |  id
6. <op> := +
7.         |  -
```

Uma gramática $G = (S,N,T,P)$

- S é o *símbolo inicial*
- N é um conjunto de *símbolos não-terminais*
- T é um conjunto de símbolos terminais
- *P é um conjunto de produções* — $P: N \rightarrow (N \cup T)^*$

Derivando sentenças válidas

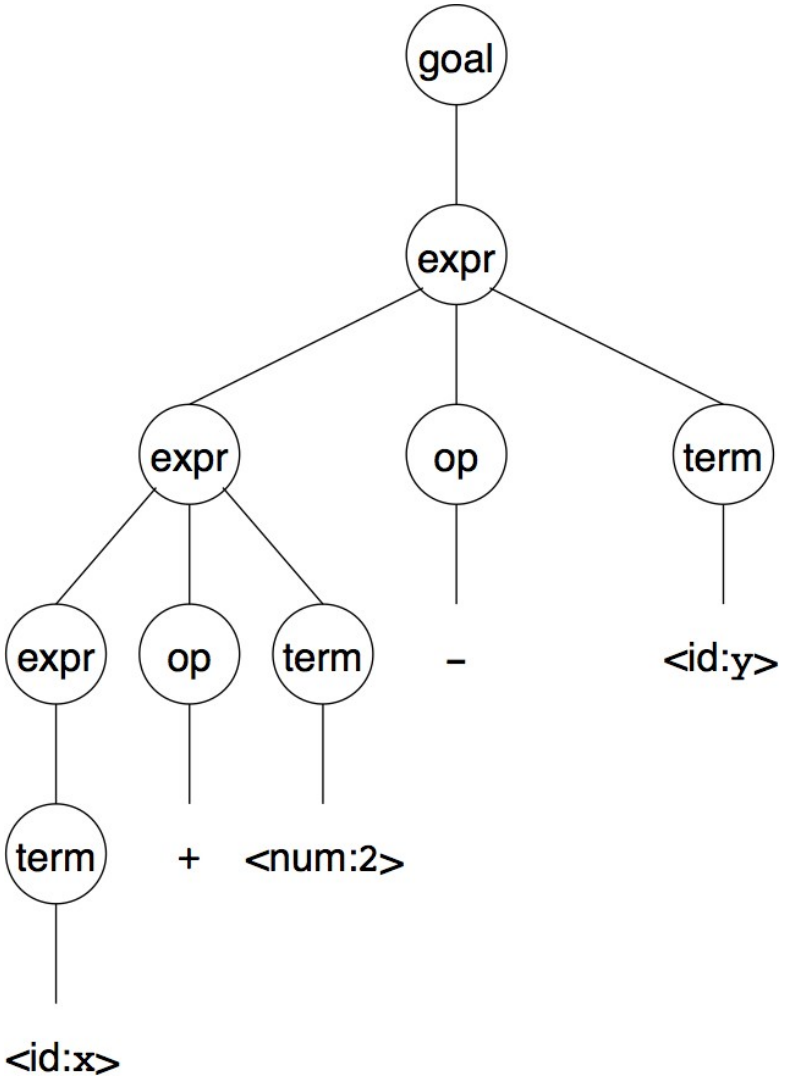
Produção	Resultado
	<objetivo>
1	<expr>
2	<expr> <op> <term>
5	<expr> <op> y
7	<expr> - y
2	<expr> <op> <term> - y
4	<expr> <op> 2 - y
6	<expr> + 2 - y
3	<term> + 2 - y
5	x + 2 - y

Dada uma gramática, sentenças válidas podem ser derivadas por repetidas substituições.

Para reconhecer uma sentença válida em alguma GLC, nós invertemos este processo e construimos uma análise sintática.

Árvores sintáticas

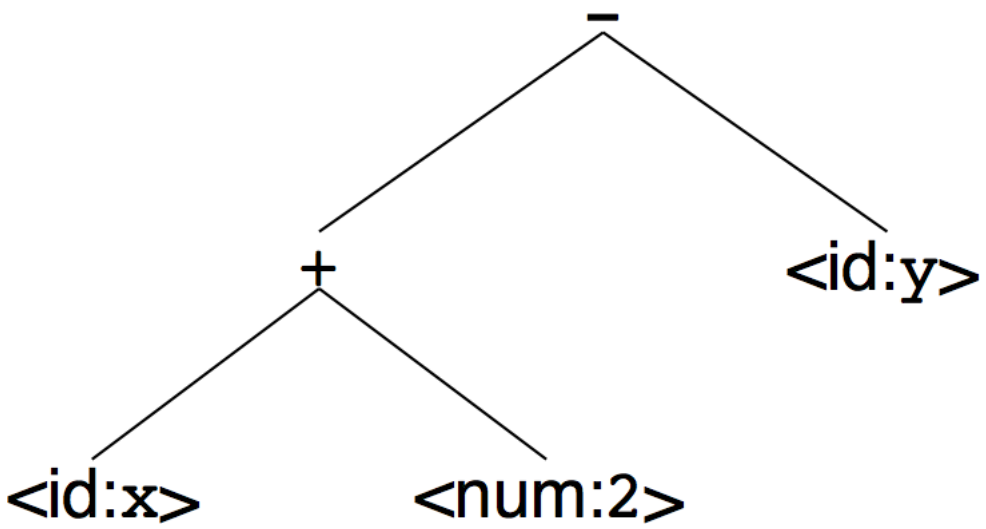
Uma análise sintática pode ser representada por uma *árvore sintática*.



Obviamente, isto contém um monte de informações desnecessárias

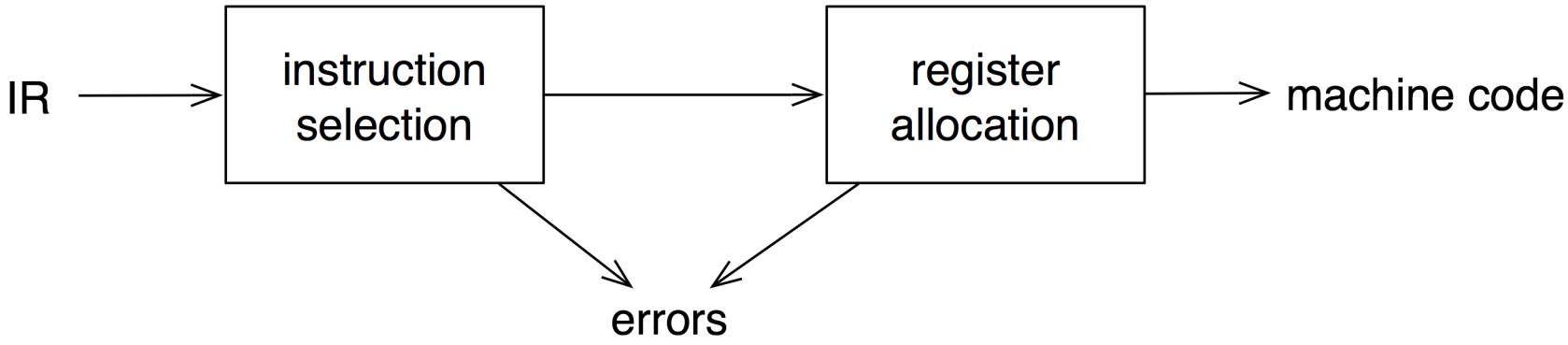
Árvores sintáticas abstratas

Assim, compiladores frequentemente usam uma ár*vore sintática* abstrata (AST)



ASTs são muitas vezes usadas como RI.

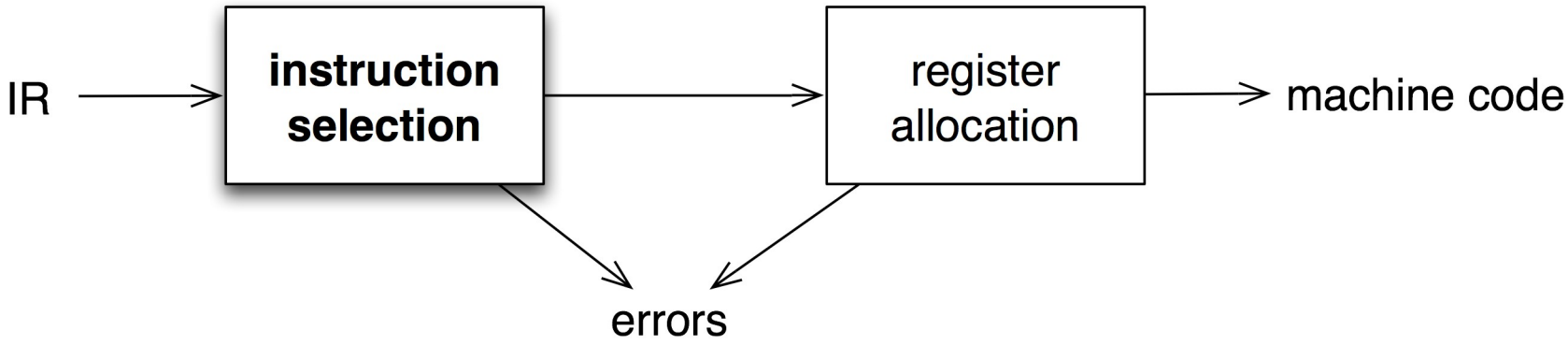
Retaguarda



- Traduz a RI no código da máquina alvo
- Escolhe instruções para cada operação da RI
- Decide o que manter em registradores em cada ponto
- Assegura conformidade com as interfaces do sistema

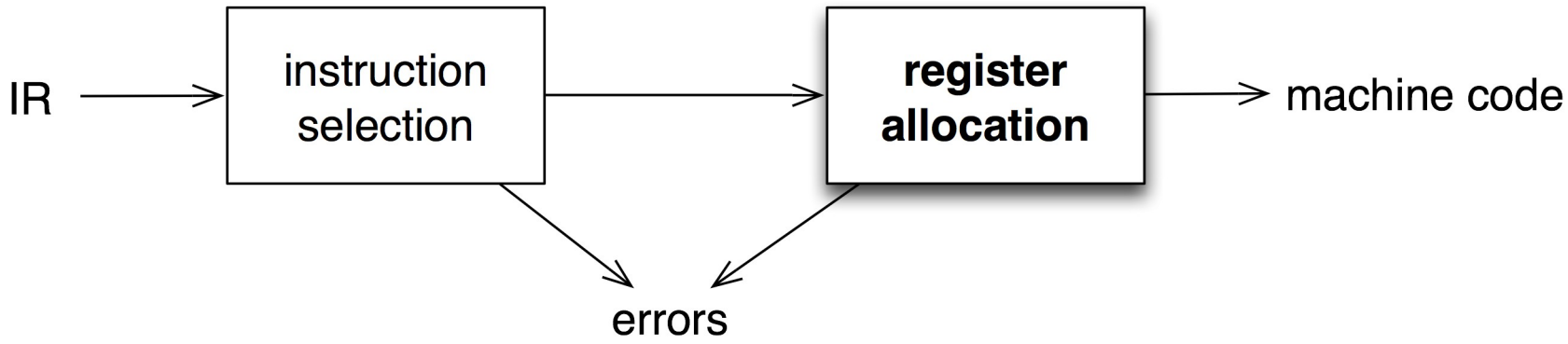
Automação tem pouco sucesso aqui

Seleção de instrução



- Produz código rápido e compacto
- Usa os modos de endereçamento disponíveis
- Um problema de casamento de padrões
  - *Técnicas ad hoc*
  - *Casamento de padrões em árvores*
  - *Casamento de padrões em strings*
  - *Programação dinâmica*

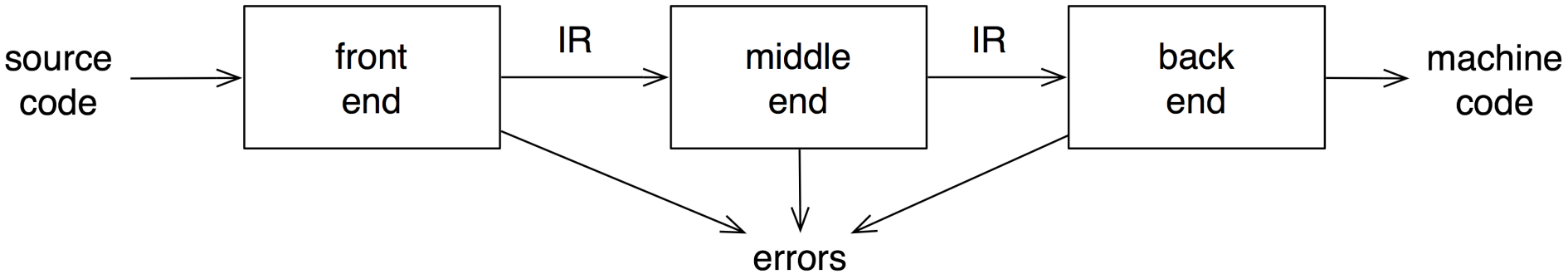
Alocação de registradores



- Ter um valor em um registrador quando necessário
- Recusos limitados
- Altera a escolha das instruções
- Pode mover cargas e armazenagens
- Alocação ótima é difícil

Muitos alocadores modernos usam uma analogia com coloração de grafos

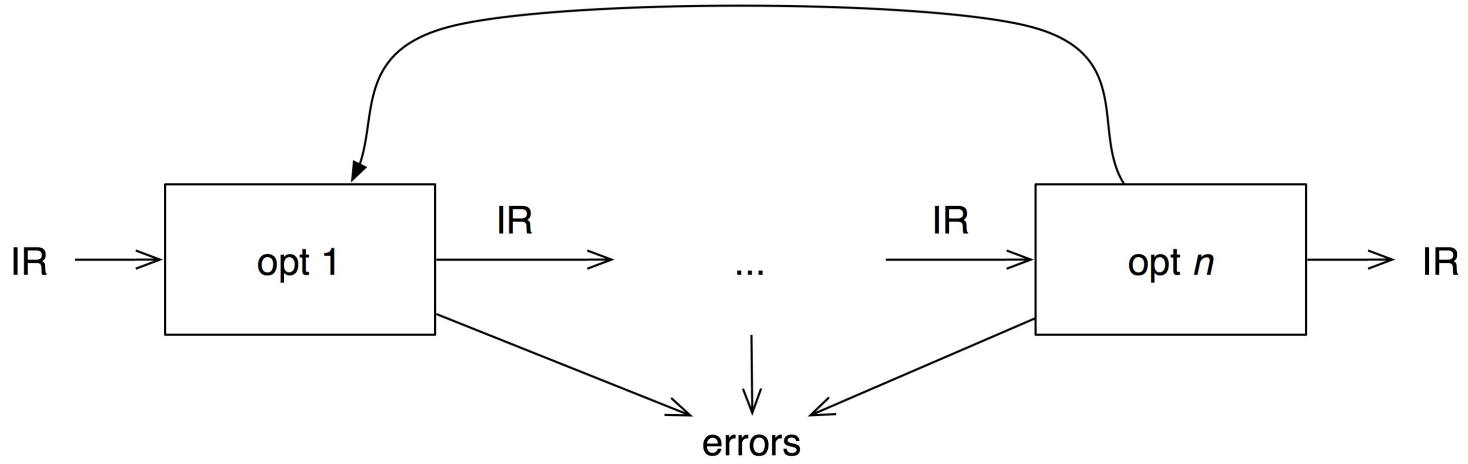
Compilador tradicional em três passos



- Analisa e altera a RI
- O objetivo é reduzir tempo de execução
- Deve preservar valores

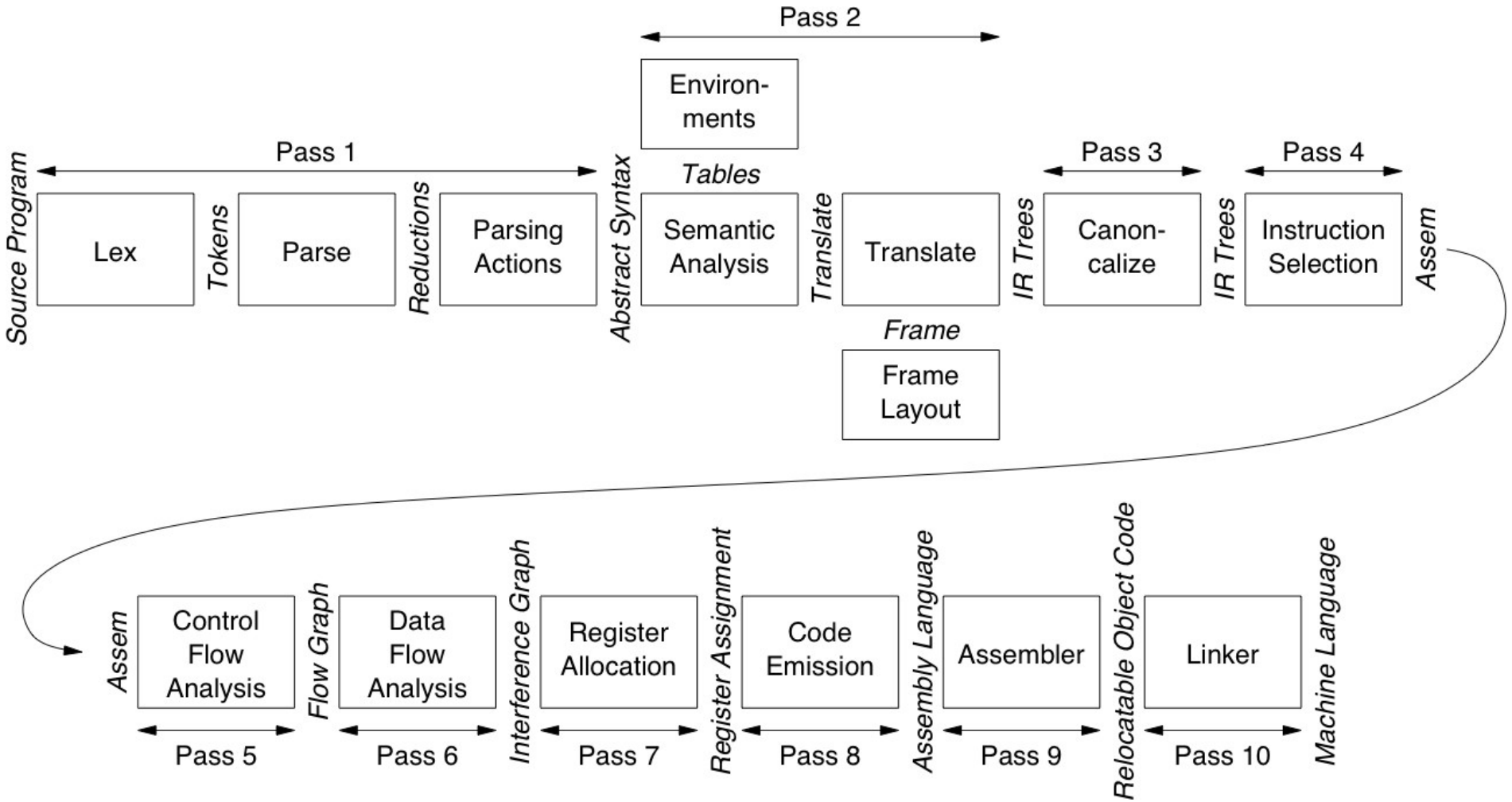
Optimizador (middle end)

Otimizadores modernos normalmente são construídos como um conjunto de passos



- Propagação de constantes e folding
- Movimentação de código
- Redução da força do operador
- Eliminação de sub-expressões comuns
- Eliminação de armazenagem redundante
- Eliminação de código morto

Uma arquitetura geral de um compilador



Fases do compilador

Análise léxica	Particiona o arquivo fonte em palavras individuais ou *tokens*
Análise sintática	Analisa a estrutura sintática do programa
Ações sintáticas	Constrói uma parte da *árvore sintática abstrata* para cada sentença
Análise semântica	Determina o que cada sentença significa, relaciona uso de variáveis às suas definições, verifica os tipos das expressões, requisita a traduação de cada sentença
Leiaute de quadros	Posiciona variáveis, parâmetros de funções, etc. em registros de ativação (quadros de pilha) de forma dependente de máquina
Tradução	Produz *árvores de representação intermediárias* (árvores IR), uma notação que não está ligada a uma linguagem fonte particular ou máquina alvo
Canonização	Realça os efeitos colateriais de expressões e organiza os ramos condicionais para conveniência de fases posteriores
Seleção de instruções	Agrupa os nós da árvore RI em partes que correspondem a ações das instruções da máquina alvo
Análise do fluxo de controle	Analisa sequências de instruções em um *grafo de fluxo de controle* mostrando todos os fluxos de controle que um programa poderia seguir se executado
Análise do fluxo de dados	Reúne informação sobre o fluxo de dados através das variáveis do programa; Ex: a análise do tempo de vida calcula os trechos onde cada variável mantém um valor que ainda será necessário (vivo)
Alocação de registradores	Escolhe registradores para variáveis e valores temporários; variáveis que não estão vivas simultaneamente podem compartilhar o mesmo registrador
Emissão de código	Substitui nomes temporárrios em cada instrução de máquina por registradores