

PCS-3566 / PCS-3866

Linguagens e Compiladores

Prof. João José Neto

Aula 01 – Introdução

Linguagens, Compiladores e Interpretadores

- Esta disciplina estuda a implementação de programas que preparam para a execução textos escritos em linguagens de programação de alto nível.
 - **Linguagem** – este é o objetivo último do nosso estudo: conhecer formas de disponibilizar no computador uma linguagem de programação, de modo que possa ser usada para construir programas executáveis. Nesta disciplina, será priorizado o estudo de linguagens de alto nível imperativas.
 - **Montadores** – as linguagens de programação mais primitivas denotam, em notação simbólica, linguagens próximas das de máquina. Alvos de interesse de outras disciplinas, os montadores são os programas de sistema responsáveis pela preparação para a execução dessas linguagens simbólicas de baixo nível.
 - **Compiladores** – uma das maneiras de implementar uma linguagem de programação de alto nível é através de programas compiladores, que efetuam a tradução da linguagem de entrada para uma forma que seja executável no computador, como por exemplo uma linguagem de máquina. Nesta disciplina, os compiladores serão priorizados em relação aos interpretadores como forma de implementação de linguagens de programação.
 - **Interpretadores** – outra alternativa evita fazer traduções, preferindo executar procedimentos que realizam aos poucos, na sequência adequada, operações equivalentes a cada uma as diversas partes do programa-fonte de entrada. Aqui, técnicas usais da construção de interpretadores poderão ser empregadas em conjunto com a simulação de sistemas reativos, orientada a eventos, e na implementação de ambientes de execução e de linguagens de script.

Linguagens, Compiladores e Interpretadores

- Esta disciplina estuda a implementação de programas que preparam para a execução textos escritos em linguagens de programação de alto nível.
 - **Linguagem** – este é o objetivo último do nosso estudo: conhecer formas de disponibilizar no computador uma linguagem de programação, de modo que possa ser usada. Nesta disciplina, será priorizado o estudo de linguagens de alto nível imperativas.
 - **Montagem** – estudo da notação e das outras disciplinas relacionadas pela preparação de programas executáveis.
 - **Compiladores** – estudo da programação e da tradução de programas de alto nível para o computador. Nesta disciplina, os compiladores de implementação de linguagens de alto nível imperativas serão estudados.
 - **Interpretadores** – estudo dos procedimentos e da implementação de programas de alto nível imperativos. Nesta disciplina, será priorizado o estudo de linguagens de alto nível imperativas.

Linguagens, Compiladores e Interpretadores

- Esta disciplina estuda a implementação de programas que preparam para a execução textos escritos em linguagens de programação de alto nível
 - **Linguagem** – este é o primeiro passo para disponibilizar no computador uma linguagem que possa ser usada diretamente pelo usuário. É priorizado o estilo de programação.
 - **Montadores** – a notação simbólica das linguagens de outras disciplinas é traduzida para a notação de máquina pela preparação de programas de sistema.
 - **Compiladores** – a programação de alto nível é traduzida para a linguagem de máquina a tradução da linguagem de alto nível para a linguagem de máquina, com o auxílio de programas de sistema, os compiladores são os programas de sistema responsáveis pela preparação para a execução dessas linguagens simbólicas de baixo nível.
 - **Interpretadores** – os procedimentos de execução equivalentes a compilação são executados diretamente. Aqui, técnicas usadas para a implementação de compiladores são usadas para a implementação de interpretadores.

Linguagens, Compiladores e Interpretadores

- Esta disciplina estuda a implementação de linguagens de programação para a execução de textos escritos em uma linguagem de programação.
 - **Linguagem** – estuda a forma de disponibilizar no computador uma linguagem que possa ser usada para a implementação de programas, sendo priorizado o estudo da sintaxe e da semântica.
 - **Montadores** – estuda a notação simbólica e a tradução da linguagem de montagem para a linguagem de máquina, sendo priorizada a implementação da notação simbólica e a tradução da linguagem de montagem para a linguagem de máquina.
 - **Compiladores** – estuda a programação de computadores e a tradução da linguagem de programação de alto nível para a linguagem de máquina, com os compiladores sendo priorizados em relação aos interpretadores como forma de implementação de linguagens de programação.
 - **Interpretadores** – estuda os procedimentos de implementação de linguagens de programação, sendo priorizados os procedimentos equivalentes a compiladores e interpretadores. Aqui, técnicas usadas para a implementação de linguagens de programação são priorizadas.

Interpretadores – outra alternativa evita fazer traduções, preferindo executar procedimentos que realizam aos poucos, na sequência adequada, operações equivalentes a cada uma as diversas partes do programa-fonte de entrada. Aqui, técnicas usais da construção de interpretadores poderão ser empregadas em conjunto com a simulação de sistemas reativos, orientada a eventos, e na implementação de ambientes de execução e de linguagens de script.

os compiladores são priorizados em relação aos interpretadores como forma de implementação de linguagens de programação.

- **Interpretadores** – outra alternativa evita fazer traduções, preferindo executar procedimentos que realizam aos poucos, na sequência adequada, operações equivalentes a cada uma as diversas partes do programa-fonte de entrada. Aqui, técnicas usais da construção de interpretadores poderão ser empregadas em conjunto com a simulação de sistemas reativos, orientada a eventos, e na implementação de ambientes de execução e de linguagens de script.

Paradigmas de Linguagens de Programação

- Há muitos tipos (paradigmas) de linguagens de programação, cada qual apresentando características adequadas a determinadas aplicações:
 - **Imperativo** – os programas assumem a forma de textos contendo comandos, cada qual ordena ao computador a execução de tarefas a eles associadas, na sequência especificada pelo programador. Ex.: Linguagem C.
 - **Funcional** – os programas são reduzidos a declarações de funções mutuamente dependentes, paramétricas e recursivas, e sua execução se resume à avaliação da função que representa o programa como um todo. Ex.: Linguagem LISP.
 - **Lógico** – o programador fornece a um ambiente de execução uma série de regras e uma série de fatos, cabendo ao computador, dada uma questão introduzida pelo usuário, buscar e aplicar uma combinação adequada das regras e dos fatos para obter a resposta desejada. Ex.: Linguagem PROLOG.
 - **Visual** – este paradigma se apoia fortemente no conceito de sistemas reativos, ou da simulação orientada a eventos: um ambiente de programação permite ao programador dispor ícones sobre uma tela e associá-los a procedimentos de tratamento, a serem executados sempre que o ícone receber um estímulo do usuário ou do próprio programa em execução. Ex.: Linguagem Visual Basic.
 - **Orientado a objetos** – com raras implementações puras, este paradigma é encontrado geralmente combinado ao imperativo. Ex. Linguagem Smalltalk.
 - **Outras** – a maior parte das demais linguagens de programação usuais costuma apresentar-se na forma de combinações de dois ou mais paradigmas.

Paradigmas de Linguagens de Programação

- Há muitos tipos (paradigmas) de linguagens de programação, cada qual apresentando características adequadas a determinadas aplicações:
 - **Imperativo** – os programas assumem a forma de textos contendo comandos, cada qual ordena ao computador a execução de tarefas a eles associadas, na sequência especificada pelo programador. Ex.: Linguagem C.
 - **Funcional** – os programas são reduzidos a declarações de funções mutuamente dependentes, onde a ordem de execução é determinada pela dependência da função.
 - **Lógico** – os programas são reduzidos a regras e fatos, onde a ordem de execução é determinada pelas regras de inferência.
 - **Visual** – os programas são reduzidos a objetos gráficos, onde a ordem de execução é determinada pelo usuário ou do próprio programa em execução. Ex.: Linguagem visual Basic.
 - **Orientado a objetos** – com raras implementações puras, este paradigma é encontrado geralmente combinado ao imperativo. Ex. Linguagem Smalltalk.
 - **Outras** – a maior parte das demais linguagens de programação usuais costuma apresentar-se na forma de combinações de dois ou mais paradigmas.

Imperativo – os programas assumem a forma de textos contendo comandos, cada qual ordena ao computador a execução de tarefas a eles associadas, na sequência especificada pelo programador. Ex.: Linguagem C.

Paradigmas de Linguagens de Programação

- Há muitos tipos (paradigmas) de linguagens de programação, cada qual apresentando características adequadas a determinadas aplicações:
 - **Imperativo** – os programas assumem a forma de textos contendo comandos, cada qual ordena ao computador a execução de tarefas a eles associadas, na sequência especificada pelo programador. Ex.: Linguagem C.
 - **Funcional** – os programas são reduzidos a declarações de funções mutuamente dependentes, paramétricas e recursivas, e sua execução se resume à avaliação da função que representa o programa como um todo. Ex.: Linguagem LISP.
 - **Lógico** – o programa apresenta-se a um ambiente de execução uma série de regras introduzidas pelo usuário.
 - **Visual** – os programas são reduzidos a declarações de funções mutuamente dependentes, paramétricas e recursivas, e sua execução se resume à avaliação da função que representa o programa como um todo. Ex.: Linguagem LISP.
 - **Orientado a objetos** – os programas são reduzidos a declarações de funções mutuamente dependentes, paramétricas e recursivas, e sua execução se resume à avaliação da função que representa o programa como um todo. Ex.: Linguagem LISP.
 - **Outras** – a maior parte das demais linguagens de programação usuais costuma apresentar-se na forma de combinações de dois ou mais paradigmas.

Paradigmas de Linguagens de Programação

- Há muitos tipos (paradigmas) de linguagens de programação, cada qual apresentando características próprias:
 - **Imperativo** – os programas são escritos de tal forma que cada qual ordena ao computador a execução de uma sequência especificada por ele.
 - **Funcional** – os programas são escritos de tal forma que as operações são dependentes, passando de uma função para outra.
 - **Lógico** – o programador fornece a um ambiente de execução uma série de regras e uma série de fatos, cabendo ao computador, dada uma questão introduzida pelo usuário, buscar e aplicar uma combinação adequada das regras e dos fatos para obter a resposta desejada. Ex.: Linguagem PROLOG.
 - **Visual** – este paradigma surgiu com o advento da simulação orientada a objetos, onde ao programador dispõem-se ícones e menus para serem executados pelo usuário ou do próprio programa.
 - **Orientado a objetos** – com o advento da programação orientada a objetos, encontrado geralmente com a linguagem C++.
 - **Outras** – a maior parte das linguagens de programação apresentam-se na forma de compiladores ou interpretadores.

Paradigmas de Linguagens de Programação

- Há muitos tipos (paradigmas) de linguagens de programação, cada qual apresentando características adequadas a determinadas aplicações:

- **Imperativo** – os dados, cada qual ordenados, sequência e...
- **Funcional** – dependente da função e...
- **Lógico** – o regras e um introduzida regras e do...
- **Visual** – este paradigma se apoia fortemente no conceito de sistemas reativos, ou da simulação orientada a eventos: um ambiente de programação permite ao programador dispor ícones sobre uma tela e associá-los a procedimentos de tratamento, a serem executados sempre que o ícone receber um estímulo do usuário ou do próprio programa em execução. Ex.: Linguagem Visual Basic.
- **Orientado** – encontrado...
- **Outras** – a apresentar...

Paradigmas de Linguagens de Programação

- Há muitos tipos (paradigmas) de linguagens de programação, cada qual apresentando características adequadas a determinadas aplicações:
 - **Imperativo** – os programas assumem a forma de textos contendo comandos, cada qual ordena ao computador a execução de tarefas a eles associadas, na sequência especificada pelo programador. Ex.: Linguagem C.
 - **Funcional** – os programas são reduzidos a declarações de funções mutuamente dependentes, paramétricas e recursivas, e sua execução se resume à avaliação da função. Ex.: Linguagem P.
 - **Lógico** – os programas são reduzidos a regras de inferência, e sua execução se resume à introdução de regras. Ex.: Linguagem G.
 - **Visual** – os programas são reduzidos a ícones e procedimentos, e sua execução se resume à interação entre o usuário e o programa. Ex.: Linguagem Smalltalk.
 - **Orientado a objetos** – com raras implementações puras, este paradigma é encontrado geralmente combinado ao imperativo. Ex.: Linguagem Smalltalk.
 - **Outras** – a maior parte das demais linguagens de programação usuais costuma apresentar-se na forma de combinações de dois ou mais paradigmas.

Orientado a objetos – com raras implementações puras, este paradigma é encontrado geralmente combinado ao imperativo. Ex. Linguagem Smalltalk.

Paradigmas de Linguagens de Programação

- Há muitos tipos (paradigmas) de linguagens de programação, cada qual apresentando características adequadas a determinadas aplicações:
 - **Imperativo** – os programas assumem a forma de textos contendo comandos, cada qual ordena ao computador a execução de tarefas a eles associadas, na sequência especificada pelo programador. Ex.: Linguagem C.
 - **Funcional** – os programas são compostos por funções que são chamadas sequencialmente, cada uma dependendo da função que a precedeu. Ex.: Linguagem Lisp.
 - **Lógico** – o programa é composto por regras e uma base de dados, sendo a execução introduzida por uma consulta às regras e dos fatos. Ex.: Linguagem Prolog.
 - **Visual** – este paradigma é baseado na simulação de um processo, onde o programador dispõe de uma tela e associa-a procedimentos de tratamento, a serem executados sempre que o ícone receber um estímulo do usuário ou do próprio programa em execução. Ex.: Linguagem Visual Basic.
 - **Orientado a objetos** – com raras implementações puras, este paradigma é encontrado geralmente combinado ao imperativo. Ex. Linguagem Smalltalk.
 - **Outras** – a maior parte das demais linguagens de programação usuais costuma apresentar-se na forma de combinações de dois ou mais paradigmas.

Componentes das Linguagens de Programação

- As linguagens usuais de programação se apresentam em geral na forma de textos nos quais se podem identificar estruturas com diversas granularidades, entre as quais:
 - **Elementos básicos de representação** – tipicamente caracteres ASCII em arquivos de texto.
 - **Comentários** – sem valor para o computador, mas muito úteis para o programador.
 - **Abreviaturas** – na forma de funções, subrotinas, macros, etc.
 - **Elementos léxicos** – tipicamente identificadores, numerais inteiros em diversas bases, números de ponto flutuante, palavras reservadas, sinais de formatação: pontuação, sinais de agrupamento, de espaçamento, etc.
 - **Construções sintáticas gerais** – listas de identificadores, constantes, parâmetros e argumentos, sequências, expressões, seletores, etc.
 - **Construções sintáticas específicas** – declarações, comandos, blocos, estruturas de dados, estruturas de controle, etc.

Componentes das Linguagens de Programação

- As linguagens usuais de programação se apresentam em geral na forma de textos nos quais se podem identificar estruturas com diversas granularidades, entre as quais:
 - **Elementos básicos de representação** – tipicamente caracteres ASCII em arquivos de texto.
 - **Comentários** – sem valor para o computador, mas muito úteis para o programador.
 - **Abreviaturas** – na forma de funções, subrotinas, macros, etc.
 - **Elementos léxicos** – tipicamente identificadores, numerais inteiros em diversas bases, números de ponto flutuante, palavras reservadas, sinais de formatação: pontuação, sinais de agrupamento, de espaçamento, etc.
 - **Construções sintáticas gerais** – listas de identificadores, constantes, parâmetros e argumentos, sequências, expressões, seletores, etc.
 - **Construções sintáticas específicas** – declarações, comandos, blocos, estruturas de dados, estruturas de controle, etc.

Componentes das Linguagens de Programação

- As linguagens usuais de programação se apresentam em geral na forma de textos nos quais se podem identificar estruturas com diversas granularidades, entre as quais:
 - **Elementos básicos de representação** – tipicamente caracteres ASCII em arquivos de texto.
 - **Comentários** – sem valor para o computador, mas muito úteis para o programador.
 - **Abreviaturas** – na forma de funções, subrotinas, macros, etc.
 - **Elementos léxicos** – tipicamente identificadores, numerais inteiros em diversas bases, números de ponto flutuante, palavras reservadas, sinais de formatação: pontuação, sinais de agrupamento, de espaçamento, etc.
 - **Construções sintáticas gerais** – listas de identificadores, constantes, parâmetros e argumentos, sequências, expressões, seletores, etc.
 - **Construções sintáticas específicas** – declarações, comandos, blocos, estruturas de dados, estruturas de controle, etc.

Componentes das Linguagens de Programação

- As linguagens usuais de programação se apresentam em geral na forma de textos nos quais se podem identificar estruturas com diversas granularidades, entre as quais:
 - **Elementos básicos de representação** – tipicamente caracteres ASCII em arquivos de texto.
 - **Comentários** – sem valor para o computador, mas muito úteis para o programador.
 - **Abreviaturas** – na forma de funções, subrotinas, macros, etc.
 - **Elementos léxicos** – tipicamente identificadores, numerais inteiros em diversas bases, números de ponto flutuante, palavras reservadas, sinais de formatação: pontuação, sinais de agrupamento, de espaçamento, etc.
 - **Construções sintáticas gerais** – listas de identificadores, constantes, parâmetros e argumentos, sequências, expressões, seletores, etc.
 - **Construções sintáticas específicas** – declarações, comandos, blocos, estruturas de dados, estruturas de controle, etc.

Componentes das Linguagens de Programação

- As linguagens usuais de programação se apresentam em geral na forma de textos nos quais se podem identificar estruturas com diversas granularidades, entre as quais:
 - **Elementos básicos de representação** – tipicamente caracteres ASCII em arquivos de texto.
 - **Comentários** – sem valor para o computador, mas muito úteis para o programador.
 - **Abreviaturas** – na forma de funções, subrotinas, macros, etc.
 - **Elementos léxicos** – tipicamente identificadores, numerais inteiros em diversas bases, números de ponto flutuante, palavras reservadas, sinais de formatação: pontuação, sinais de agrupamento, de espaçamento, etc.
 - **Construções sintáticas gerais** – listas de identificadores, constantes, parâmetros e argumentos, sequências, expressões, seletores, etc.
 - **Construções sintáticas específicas** – declarações, comandos, blocos, estruturas de dados, estruturas de controle, etc.

Componentes das Linguagens de Programação

- As linguagens usuais de programação se apresentam em geral na forma de textos nos quais se podem identificar estruturas com diversas granularidades, entre as quais:
 - **Elementos básicos de representação** – tipicamente caracteres ASCII em arquivos de texto.
 - **Comentários** – sem valor para o computador, mas muito úteis para o programador.
 - **Abreviaturas** – na forma de funções, subrotinas, macros, etc.
 - **Elementos léxicos** – tipicamente identificadores, numerais inteiros em diversas bases, números de ponto flutuante, palavras reservadas, sinais de formatação: pontuação, sinais de agrupamento, de espaçamento, etc.
 - **Construções sintáticas gerais** – listas de identificadores, constantes, parâmetros e argumentos, sequências, expressões, seletores, etc.
 - **Construções sintáticas específicas** – declarações, comandos, blocos, estruturas de dados, estruturas de controle, etc.

Componentes das Linguagens de Programação

- As linguagens usuais de programação se apresentam em geral na forma de textos nos quais se podem identificar estruturas com diversas granularidades, entre as quais:
 - **Elementos básicos de representação** – tipicamente caracteres ASCII em arquivos de texto.
 - **Comentários** – sem valor para o computador, mas muito úteis para o programador.
 - **Abreviaturas** – na forma de funções, subrotinas, macros, etc.
 - **Elementos léxicos** – tipicamente identificadores, numerais inteiros em diversas bases, números de ponto flutuante, palavras reservadas, sinais de formatação: pontuação, sinais de agrupamento, de espaçamento, etc.
 - **Construções sintáticas gerais** – listas de identificadores, constantes, parâmetros e argumentos, sequências, expressões, seletores, etc.
 - **Construções sintáticas específicas** – declarações, comandos, blocos, estruturas de dados, estruturas de controle, etc.

Compiladores

- A compilação consiste na obtenção de um programa em formato executável a partir do programa-fonte fornecido pelo usuário.
- Um compilador típico executa as seguintes operações:
 - **Análise Léxica** – Decompor o programa-fonte, dele extraíndo os átomos (partículas elementares), e converter a sequência assim obtida em outra, formada apenas desses elementos léxicos da linguagem.
 - **Análise sintática** – Aplicando-se sucessivamente o mesmo processo às sequências resultantes, até que todo o programa seja devidamente esgotado, permite nelas reconhecer construções sintáticas cada vez mais gerais, até que uma única estrutura global seja enfim identificada.
 - **Geração de código** – As estruturas sintáticas assim levantadas são então interpretadas, com o objetivo de se construir, a partir das informações nelas contidas, um código-objeto executável equivalente.
 - **Atividades adicionais** – Além dessas atividades básicas, compiladores dedicam-se a diversas importantes atividades auxiliares, entre as quais: a produção de diversos tipos de relatórios e listagens, a identificação, o registro e a recuperação de erros, a otimização de código: global e local, independente e dependente de máquina, e outros.

Compiladores

- A compilação consiste na obtenção de um programa em formato executável a partir do programa-fonte fornecido pelo usuário.
- Um compilador típico executa as seguintes operações:
 - **Análise Léxica** – Decompor o programa-fonte, dele extraíndo os átomos (partículas elementares), e converter a sequência assim obtida em outra, formada apenas desses elementos léxicos da linguagem.
 - **Análise sintática** – Aplicando-se sucessivamente o mesmo processo às sequências resultantes, até que todo o programa seja devidamente esgotado, permite nelas reconhecer construções sintáticas cada vez mais gerais, até que uma única estrutura global seja enfim identificada.
 - **Geração de código** – As estruturas sintáticas assim levantadas são então interpretadas, com o objetivo de se construir, a partir das informações nelas contidas, um código-objeto executável equivalente.
 - **Atividades adicionais** – Além dessas atividades básicas, compiladores dedicam-se a diversas importantes atividades auxiliares, entre as quais: a produção de diversos tipos de relatórios e listagens, a identificação, o registro e a recuperação de erros, a otimização de código: global e local, independente e dependente de máquina, e outros.

Compiladores

- A compilação consiste na obtenção de um programa em formato executável a partir do programa-fonte fornecido pelo usuário.
- Um compilador típico executa as seguintes operações:
 - **Análise Léxica** – Decompor o programa-fonte, dele extraíndo os átomos (partículas elementares), e converter a sequência assim obtida em outra, formada apenas desses elementos léxicos da linguagem.
 - **Análise sintática** – Aplicando-se sucessivamente o mesmo processo às sequências resultantes, até que todo o programa seja devidamente esgotado, permite nelas reconhecer construções sintáticas cada vez mais gerais, até que uma única estrutura global seja enfim identificada.
 - **Geração de código** – As estruturas sintáticas assim levantadas são então interpretadas, com o objetivo de se construir, a partir das informações nelas contidas, um código-objeto executável equivalente.
 - **Atividades adicionais** – Além dessas atividades básicas, compiladores dedicam-se a diversas importantes atividades auxiliares, entre as quais: a produção de diversos tipos de relatórios e listagens, a identificação, o registro e a recuperação de erros, a otimização de código: global e local, independente e dependente de máquina, e outros.

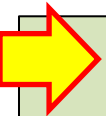
Compiladores

- A compilação consiste na obtenção de um programa em formato executável a partir do programa-fonte fornecido pelo usuário.
- Um compilador típico executa as seguintes operações:
 - **Análise Léxica** – Decompor o programa-fonte, dele extraíndo os átomos (partículas elementares), e converter a sequência assim obtida em outra, formada apenas desses elementos léxicos da linguagem.
 - **Análise sintática** – Aplicando-se sucessivamente o mesmo processo às sequências resultantes, até que todo o programa seja devidamente esgotado, permite nelas reconhecer construções sintáticas cada vez mais gerais, até que uma única estrutura global seja enfim identificada.
 - **Geração de código** – As estruturas sintáticas assim levantadas são então interpretadas, com o objetivo de se construir, a partir das informações nelas contidas, um código-objeto executável equivalente.
 - **Atividades adicionais** – Além dessas atividades básicas, compiladores dedicam-se a diversas importantes atividades auxiliares, entre as quais: a produção de diversos tipos de relatórios e listagens, a identificação, o registro e a recuperação de erros, a otimização de código: global e local, independente e dependente de máquina, e outros.

Compiladores

- A compilação consiste na obtenção de um programa em formato executável a partir do programa-fonte fornecido pelo usuário.
- Um compilador típico executa as seguintes operações:
 - **Análise Léxica** – Decompor o programa-fonte, dele extraíndo os átomos (partículas elementares), e converter a sequência assim obtida em outra, formada apenas desses elementos léxicos da linguagem.
 - **Análise sintática** – Aplicando-se sucessivamente o mesmo processo às sequências resultantes, até que todo o programa seja devidamente esgotado, permite nelas reconhecer construções sintáticas cada vez mais gerais, até que uma única estrutura global seja enfim identificada.
 - **Geração de código** – As estruturas sintáticas assim levantadas são então interpretadas, com o objetivo de se construir, a partir das informações nelas contidas, um código-objeto executável equivalente.
 - **Atividades adicionais** – Além dessas atividades básicas, compiladores dedicam-se a diversas importantes atividades auxiliares, entre as quais: a produção de diversos tipos de relatórios e listagens, a identificação, o registro e a recuperação de erros, a otimização de código: global e local, independente e dependente de máquina, e outros.

Interpretadores

- 
- A interpretação consiste em, após a identificação dos diversos componentes léxicos e sintáticos do texto-fonte de um programa, atribuir significado a eles mediante a execução de procedimentos de tratamento, que levam em consideração o padrão sintático encontrado e a situação da execução no instante em questão.
- Trata-se de mais uma das inúmeras instâncias de sistemas reativos que temos estudado, e sua realização pode ser efetuada de forma idêntica, através de um simulador guiado por eventos.
 - Assim, cada um dos elementos identificados ao longo do texto-fonte pode ser associado a um evento independente, e a execução de procedimentos específicos efetua o tratamento de cada “interrupção” associada ao evento.
 - As rotinas de tratamento podem por sua vez gerar ocorrências de eventos dependentes, cujas rotinas de tratamento podem ser dedicadas a atividades de geração de código (em compiladores) ou de interpretação direta (em interpretadores, em emuladores de conjuntos de instruções ou em ambientes de execução)
 - A seguir, mostra-se um resumo do processo de implementação de linguagens de programação.

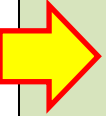
Interpretadores

- A interpretação consiste em, após a identificação dos diversos componentes léxicos e sintáticos do texto-fonte de um programa, atribuir significado a eles mediante a execução de procedimentos de tratamento, que levam em consideração o padrão sintático encontrado e a situação da execução no instante em questão.
- Trata-se de mais uma das inúmeras instâncias de sistemas reativos que temos estudado, e sua realização pode ser efetuada de forma idêntica, através de um simulador guiado por eventos.
- Assim, cada um dos elementos identificados ao longo do texto-fonte pode ser associado a um evento independente, e a execução de procedimentos específicos efetua o tratamento de cada “interrupção” associada ao evento.
- As rotinas de tratamento podem por sua vez gerar ocorrências de eventos dependentes, cujas rotinas de tratamento podem ser dedicadas a atividades de geração de código (em compiladores) ou de interpretação direta (em interpretadores, em emuladores de conjuntos de instruções ou em ambientes de execução)
- A seguir, mostra-se um resumo do processo de implementação de linguagens de programação.

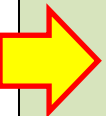
Interpretadores

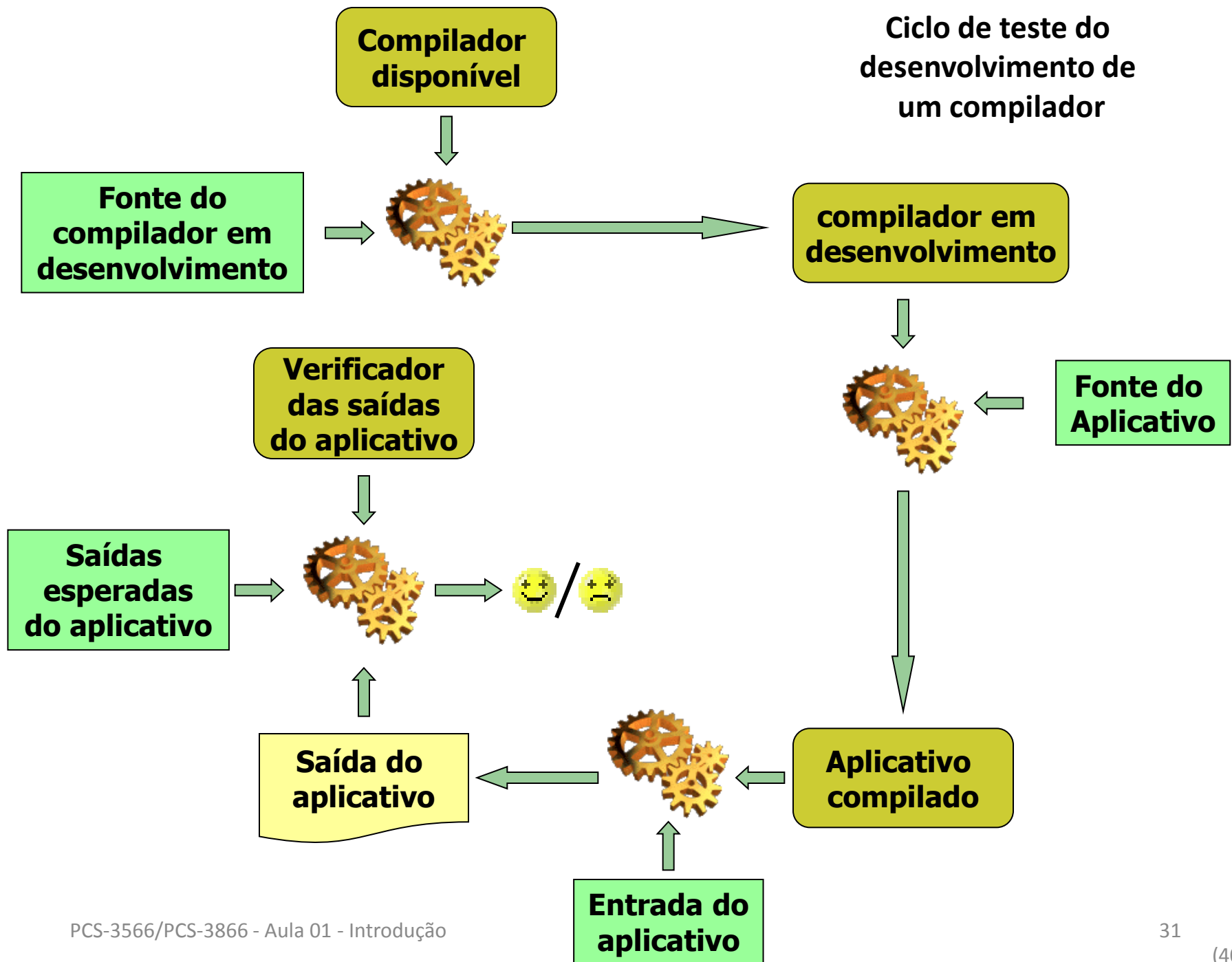
- A interpretação consiste em, após a identificação dos diversos componentes léxicos e sintáticos do texto-fonte de um programa, atribuir significado a eles mediante a execução de procedimentos de tratamento, que levam em consideração o padrão sintático encontrado e a situação da execução no instante em questão.
- Trata-se de mais uma das inúmeras instâncias de sistemas reativos que temos estudado, e sua realização pode ser efetuada de forma idêntica, através de um simulador guiado por eventos. Assim, cada um dos elementos identificados ao longo do texto-fonte pode ser associado a um evento independente, e a execução de procedimentos específicos efetua o tratamento de cada “interrupção” associada ao evento.
- As rotinas de tratamento podem por sua vez gerar ocorrências de eventos dependentes, cujas rotinas de tratamento podem ser dedicadas a atividades de geração de código (em compiladores) ou de interpretação direta (em interpretadores, em emuladores de conjuntos de instruções ou em ambientes de execução)
- A seguir, mostra-se um resumo do processo de implementação de linguagens de programação.

Interpretadores

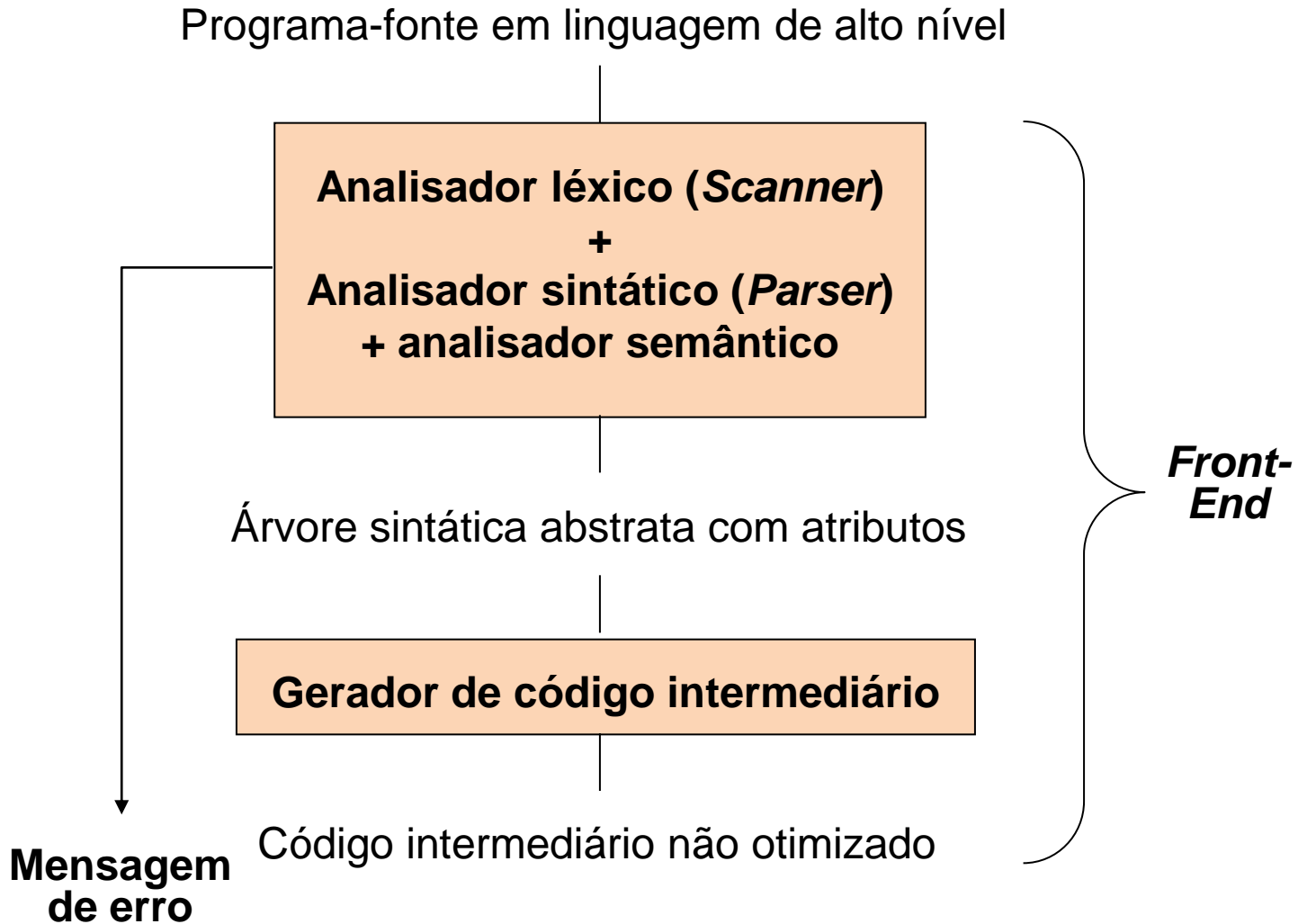
- A interpretação consiste em, após a identificação dos diversos componentes léxicos e sintáticos do texto-fonte de um programa, atribuir significado a eles mediante a execução de procedimentos de tratamento, que levam em consideração o padrão sintático encontrado e a situação da execução no instante em questão.
 - Trata-se de mais uma das inúmeras instâncias de sistemas reativos que temos estudado, e sua realização pode ser efetuada de forma idêntica, através de um simulador guiado por eventos.
 - Assim, cada um dos elementos identificados ao longo do texto-fonte pode ser associado a um evento independente, e a execução de procedimentos específicos efetua o tratamento de cada “interrupção” associada ao evento.
- 
- As rotinas de tratamento podem por sua vez gerar ocorrências de eventos dependentes, cujas rotinas de tratamento podem ser dedicadas a atividades de geração de código (em compiladores) ou de interpretação direta (em interpretadores, em emuladores de conjuntos de instruções ou em ambientes de execução)
- A seguir, mostra-se um resumo do processo de implementação de linguagens de programação.

Interpretadores

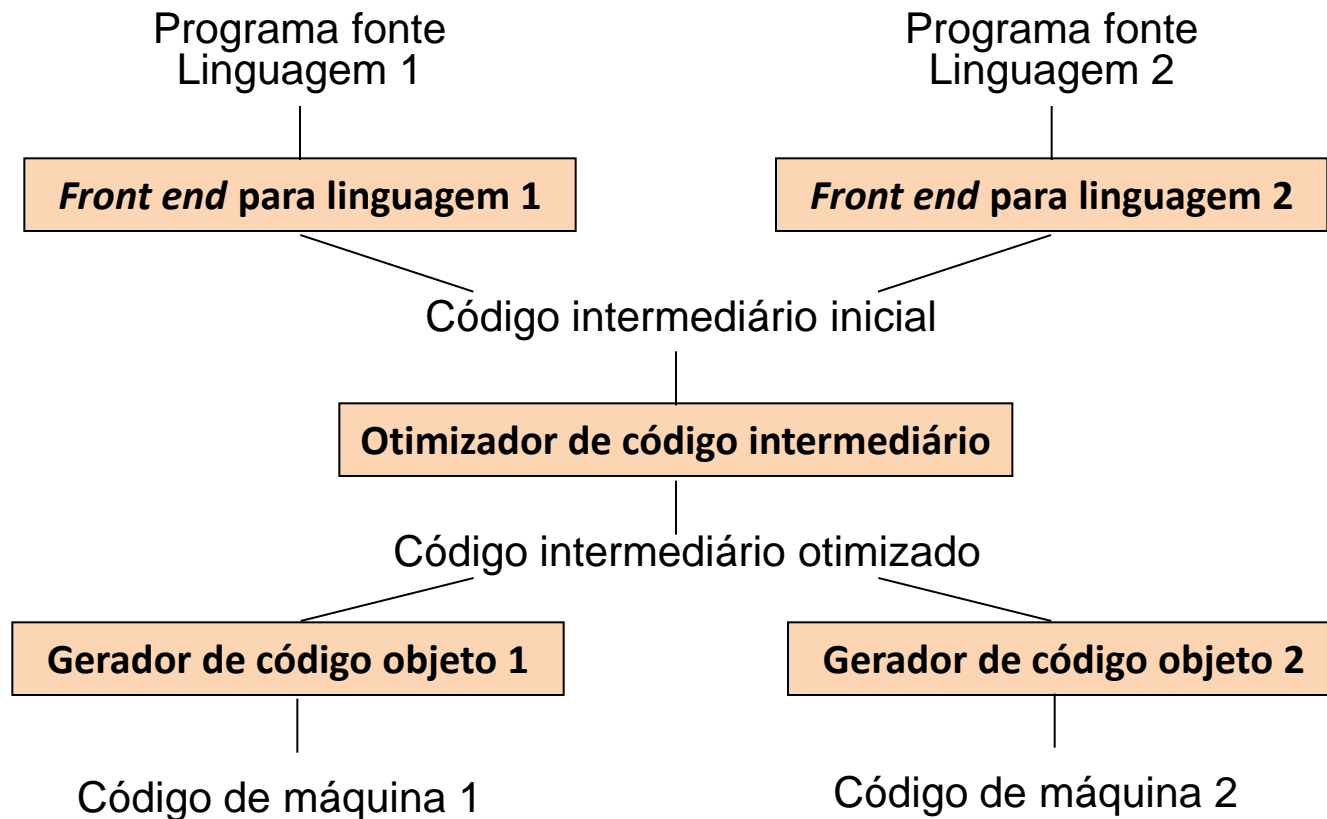
- A interpretação consiste em, após a identificação dos diversos componentes léxicos e sintáticos do texto-fonte de um programa, atribuir significado a eles mediante a execução de procedimentos de tratamento, que levam em consideração o padrão sintático encontrado e a situação da execução no instante em questão.
 - Trata-se de mais uma das inúmeras instâncias de sistemas reativos que temos estudado, e sua realização pode ser efetuada de forma idêntica, através de um simulador guiado por eventos.
 - Assim, cada um dos elementos identificados ao longo do texto-fonte pode ser associado a um evento independente, e a execução de procedimentos específicos efetua o tratamento de cada “interrupção” associada ao evento.
 - As rotinas de tratamento podem por sua vez gerar ocorrências de eventos dependentes, cujas rotinas de tratamento podem ser dedicadas a atividades de geração de código (em compiladores) ou de interpretação direta (em interpretadores, em emuladores de conjuntos de instruções ou em ambientes de execução)
-  A seguir, mostra-se um resumo do processo de implementação de linguagens de programação.



Front-End de um compilador

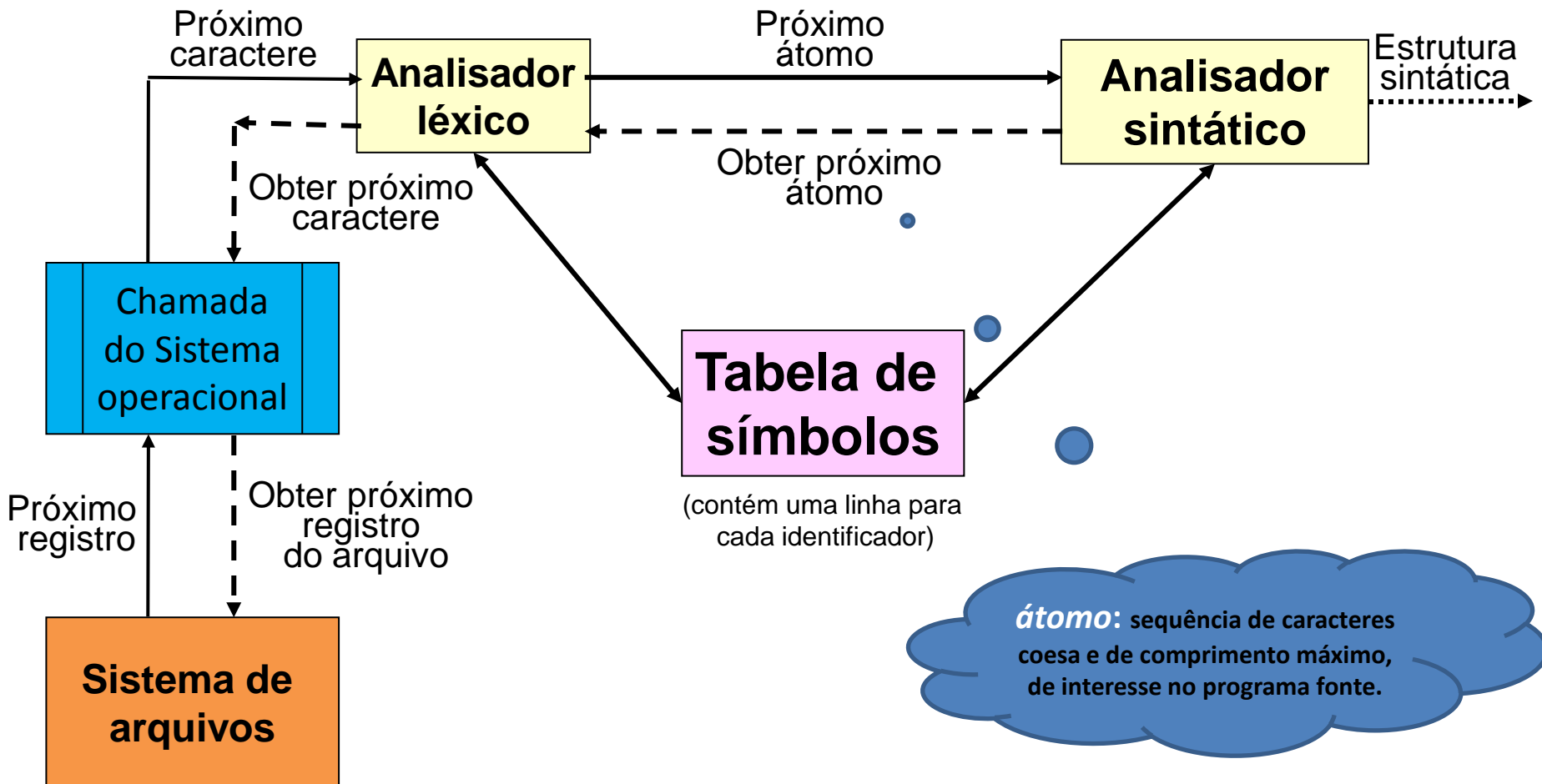


Esquema da construção de compilador, baseada em seus componentes usuais

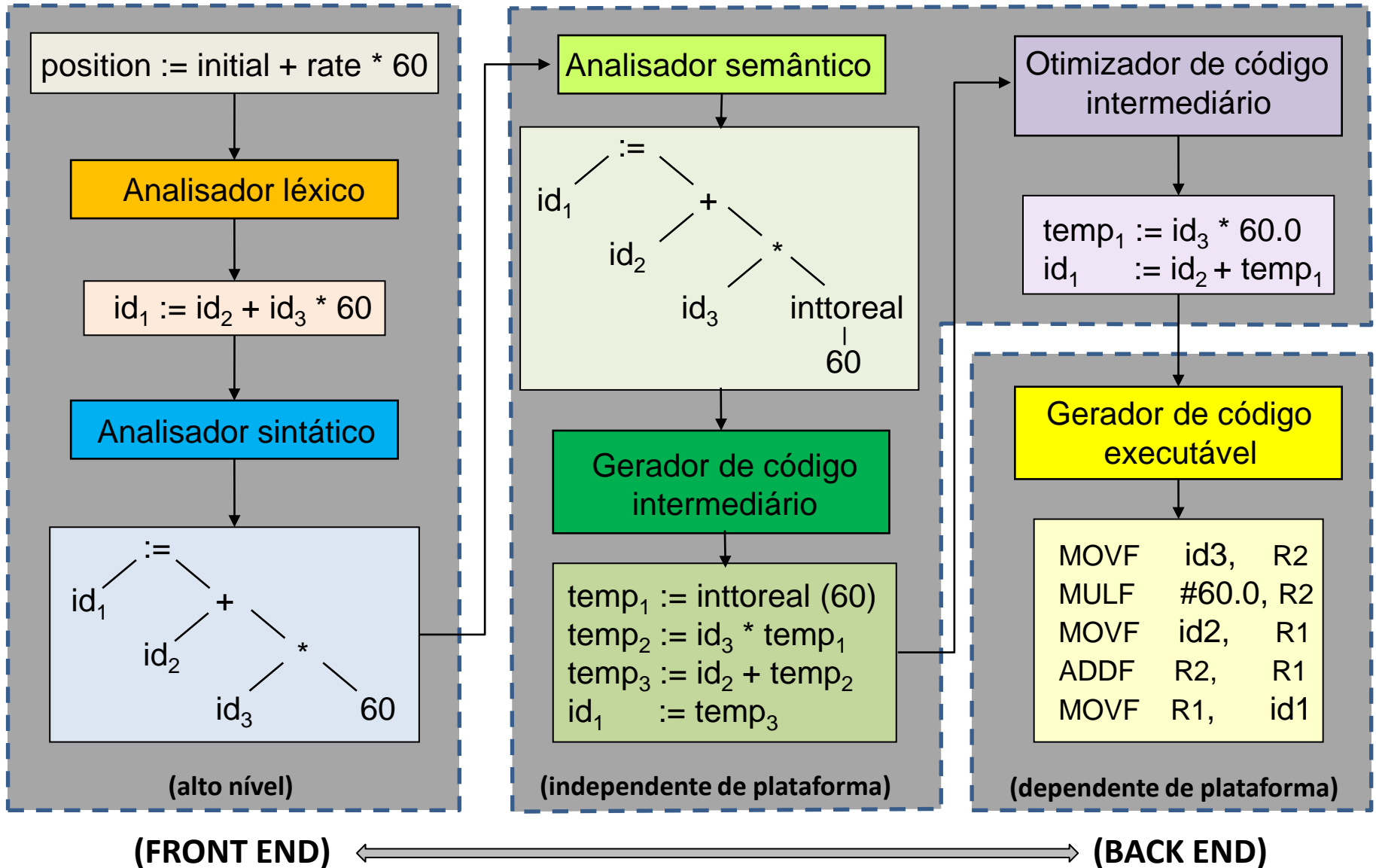


Front end – análise léxica e sintática

(adaptado de Aho, Sethi, Ullman, p. 160)



Etapas de um processo de compilação típico



Aspectos de Implementação

Diretrizes

- Da observação do rápido e superficial panorama apresentado nos slides anteriores, já é possível extrair algumas orientações para os estudos a serem desenvolvidos nesta disciplina.
- A técnica guiada por eventos tem sido adotada para simulações de autômatos e outros formalismos abstratos, bem como para a implementação de simulações de diversas partes de sistemas operacionais e outros programas concorrentes e paralelos.
- Disciplinas tais como Lógica Computacional, Linguagens e Compiladores, Sistemas de Programação e Sistemas Operacionais têm explorado extensivamente essa via intuitiva e prática de implementação, de ampla aplicabilidade acadêmica e profissional.
- Assim, o uso da técnica guiada por eventos se nos apresenta como uma proposta muito atraente também para a construção de processadores de linguagens de programação, dadas as afinidades apresentadas entre os programas que nos propomos a construir e os clássicos sistemas reativos, cujos modelos costumam criados de forma própria ao uso em simulações guiadas por eventos.

Diretrizes

- Da observação do rápido e superficial panorama apresentado nos slides anteriores, já é possível extrair algumas orientações para os estudos a serem desenvolvidos nesta disciplina.
- A técnica guiada por eventos tem sido adotada para simulações de autômatos e outros formalismos abstratos, bem como para a implementação de simulações de diversas partes de sistemas operacionais e outros programas concorrentes e paralelos.
- Disciplinas tais como Lógica Computacional, Linguagens e Compiladores, Sistemas de Programação e Sistemas Operacionais têm explorado extensivamente essa via intuitiva e prática de implementação, de ampla aplicabilidade acadêmica e profissional.
- Assim, o uso da técnica guiada por eventos se nos apresenta como uma proposta muito atraente também para a construção de processadores de linguagens de programação, dadas as afinidades apresentadas entre os programas que nos propomos a construir e os clássicos sistemas reativos, cujos modelos costumam criados de forma própria ao uso em simulações guiadas por eventos.

Diretrizes

- Da observação do rápido e superficial panorama apresentado nos slides anteriores, já é possível extrair algumas orientações para os estudos a serem desenvolvidos nesta disciplina.
- A técnica guiada por eventos tem sido adotada para simulações de autômatos e outros formalismos abstratos, bem como para a implementação de simulações de diversas partes de sistemas operacionais e outros programas concorrentes e paralelos.
- Disciplinas tais como Lógica Computacional, Linguagens e Compiladores, Sistemas de Programação e Sistemas Operacionais têm explorado extensivamente essa via intuitiva e prática de implementação, de ampla aplicabilidade acadêmica e profissional.
- Assim, o uso da técnica guiada por eventos se nos apresenta como uma proposta muito atraente também para a construção de processadores de linguagens de programação, dadas as afinidades apresentadas entre os programas que nos propomos a construir e os clássicos sistemas reativos, cujos modelos costumam criados de forma própria ao uso em simulações guiadas por eventos.

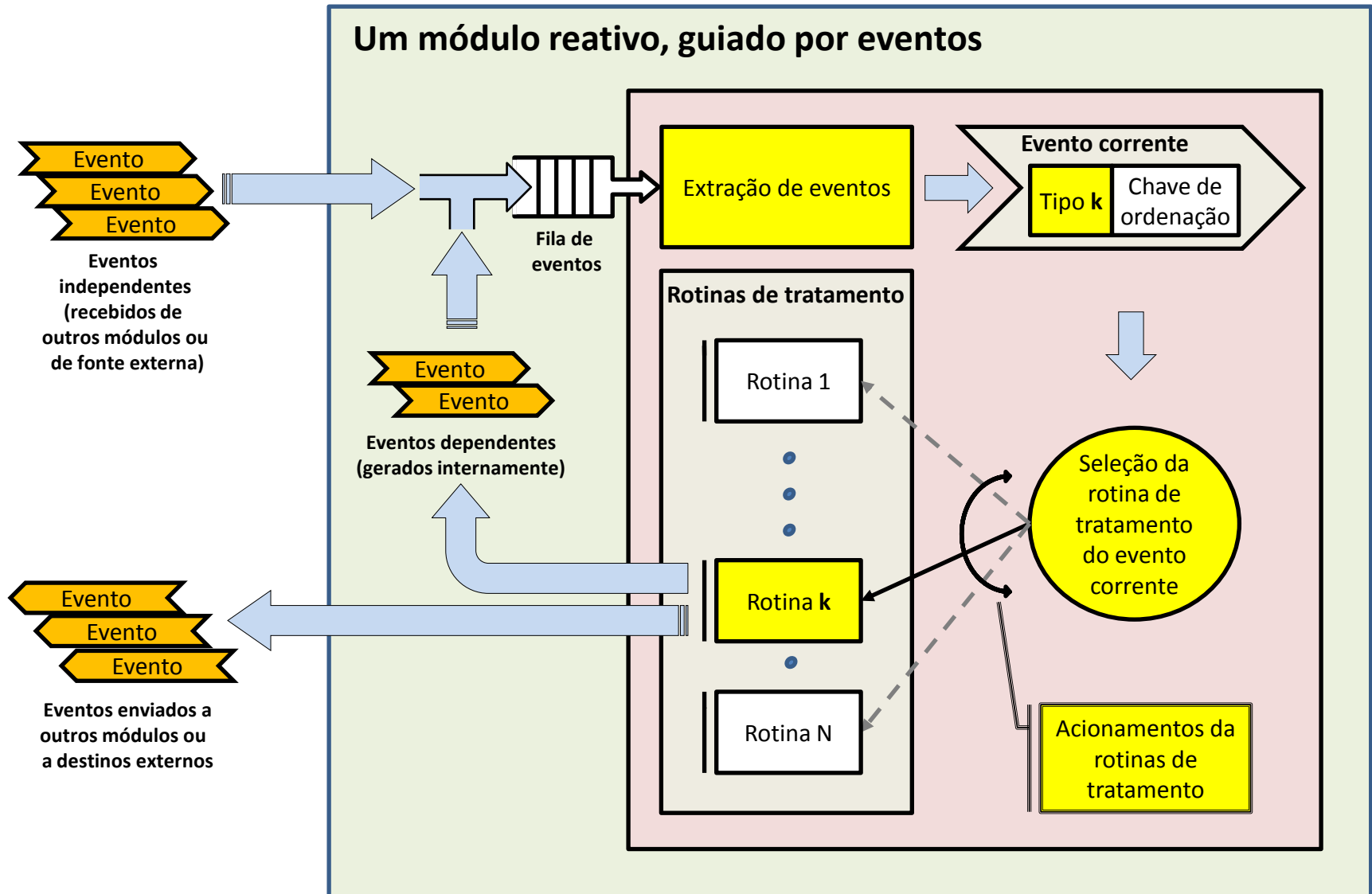
Diretrizes

- Da observação do rápido e superficial panorama apresentado nos slides anteriores, já é possível extrair algumas orientações para os estudos a serem desenvolvidos nesta disciplina.
- A técnica guiada por eventos tem sido adotada para simulações de autômatos e outros formalismos abstratos, bem como para a implementação de simulações de diversas partes de sistemas operacionais e outros programas concorrentes e paralelos.
- Disciplinas tais como Lógica Computacional, Linguagens e Compiladores, Sistemas de Programação e Sistemas Operacionais têm explorado extensivamente essa via intuitiva e prática de implementação, de ampla aplicabilidade acadêmica e profissional.
- Assim, o uso da técnica guiada por eventos se nos apresenta como uma proposta muito atraente também para a construção de processadores de linguagens de programação, dadas as afinidades apresentadas entre os programas que nos propomos a construir e os clássicos sistemas reativos, cujos modelos costumam criados de forma própria ao uso em simulações guiadas por eventos.

Projeto

- A seguir, é apresentada uma coleção de slides cujo conteúdo pode auxiliar na elaboração do software a ser desenvolvido nesta disciplina.
- Como subsídios, apresentam-se:
 - Uma visão panorâmica da estrutura geral do programa, a ser desenvolvido como software guiado por eventos.
 - A contextualização da primeira etapa do software em questão, no projeto de um compilador completo.
 - uma síntese das técnicas dirigidas por eventos, com ênfase na sua potencial multiplicidade e estrutura hierárquica.

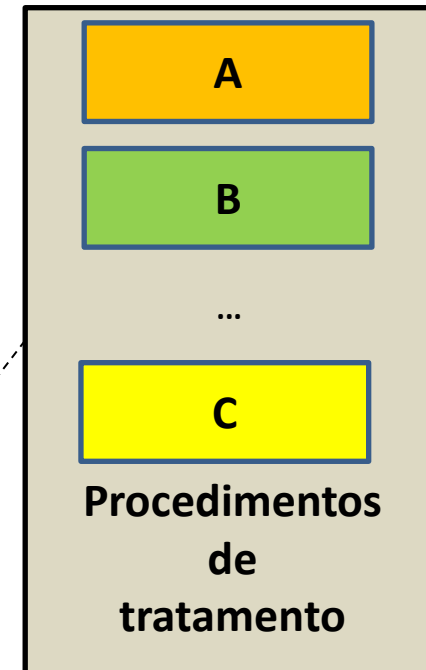
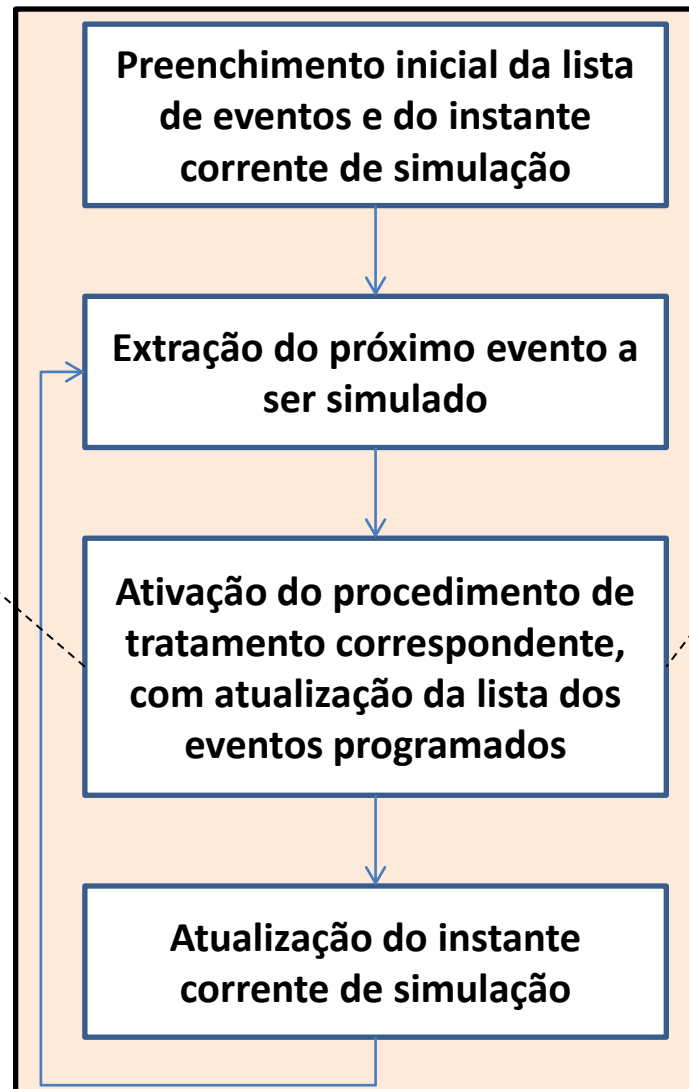
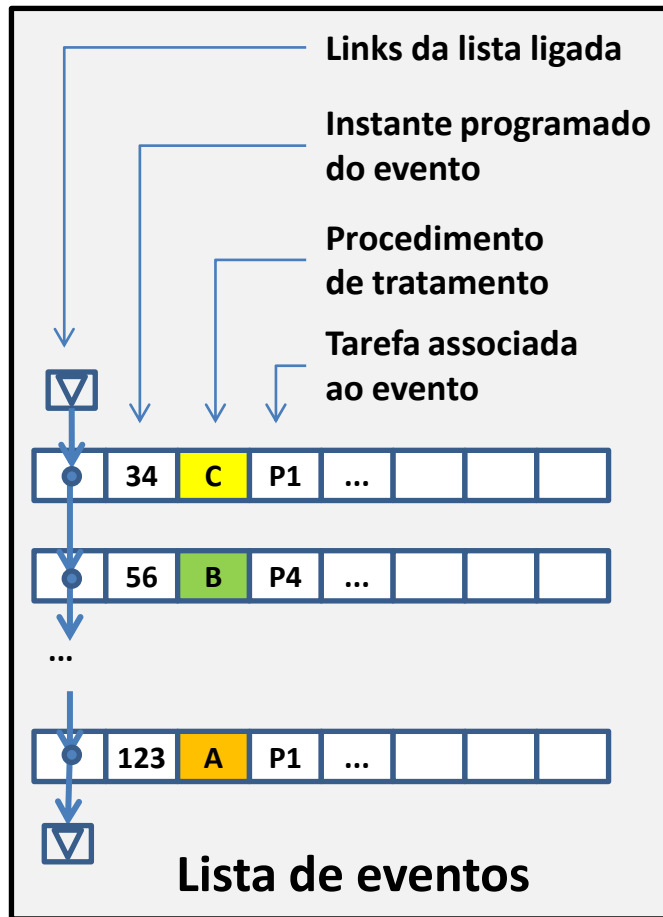
Módulo reativo, guiado por eventos



Motor de eventos, de uso geral

- O motor aqui proposto pode ser utilizado como ponto de partida na implementação de aplicações baseadas na simulação dirigida por eventos
- Destacam-se, como principais elementos desse motor:
 - Um procedimento de **carga das informações iniciais** de simulação
 - Uma estrutura de dados representando, cronologicamente ordenada, a **sequência dos eventos a serem tratados**.
 - Um conjunto de **procedimentos de tratamento**, cada qual associado ao tipo correspondente de evento.
 - Um **loop de extração do evento** a simular, com a **execução do procedimento de tratamento** correspondente .
 - Um **procedimento de finalização**, para a emissão de relatórios.

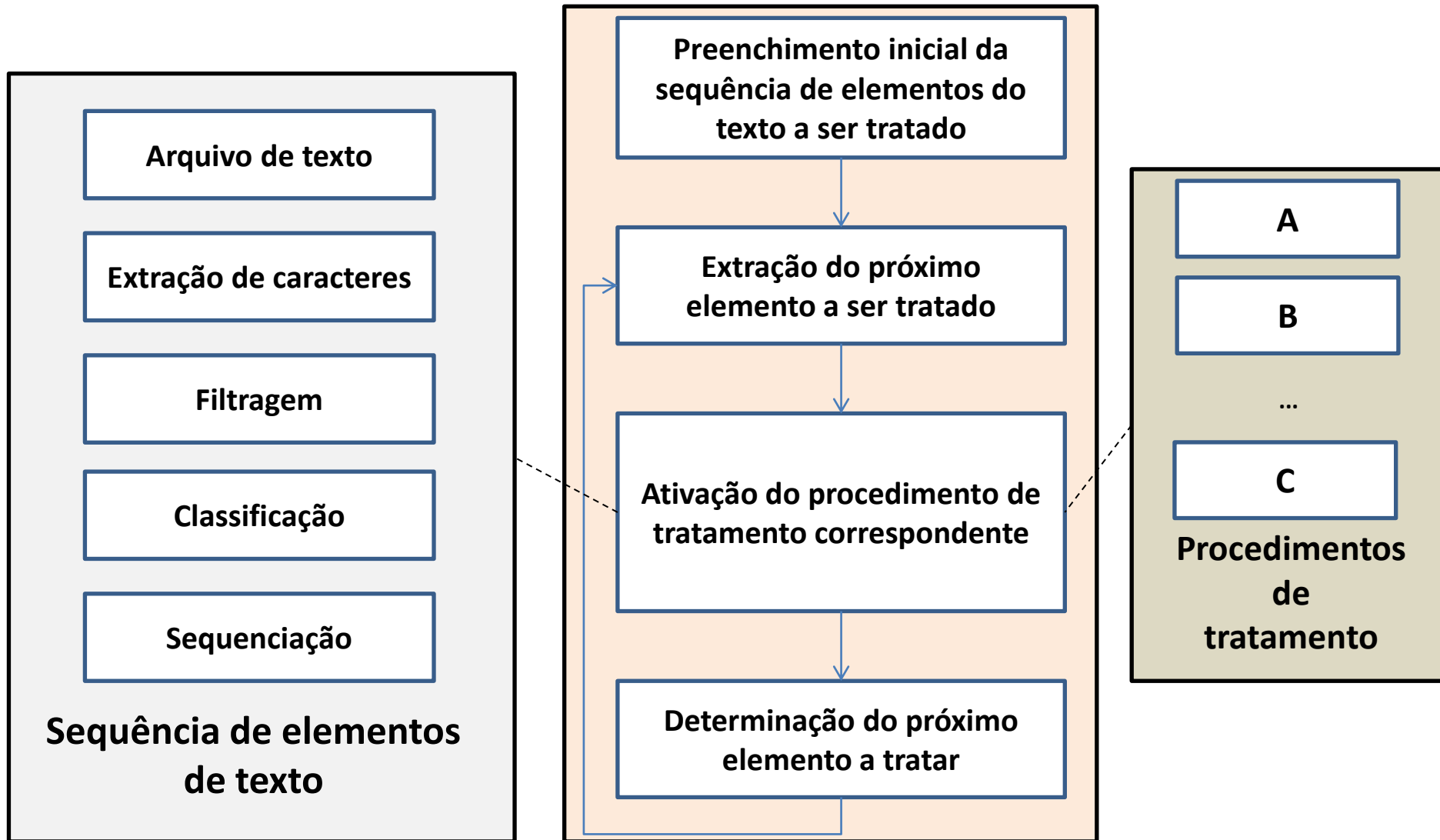
Esquema de um motor de simulação



Aplicações em Processamento de Texto

- Transcrição simples e filtrada
- Transcrição, filtrada e classificada (análise léxica)
- Transcrição, filtrada e classificada, multi-nível
- Verif. sintática livre de contexto (apoio à codificação)
- Extração de componentes de palavras e flexão
- Verif. e correção ortográfica p/ apoio à digitação
- Transcrição fonética p/ apoio a deficiente visual
- Processamento de macros léxicas
- Verificação e correção gramatical
- Preenchimento de formulários

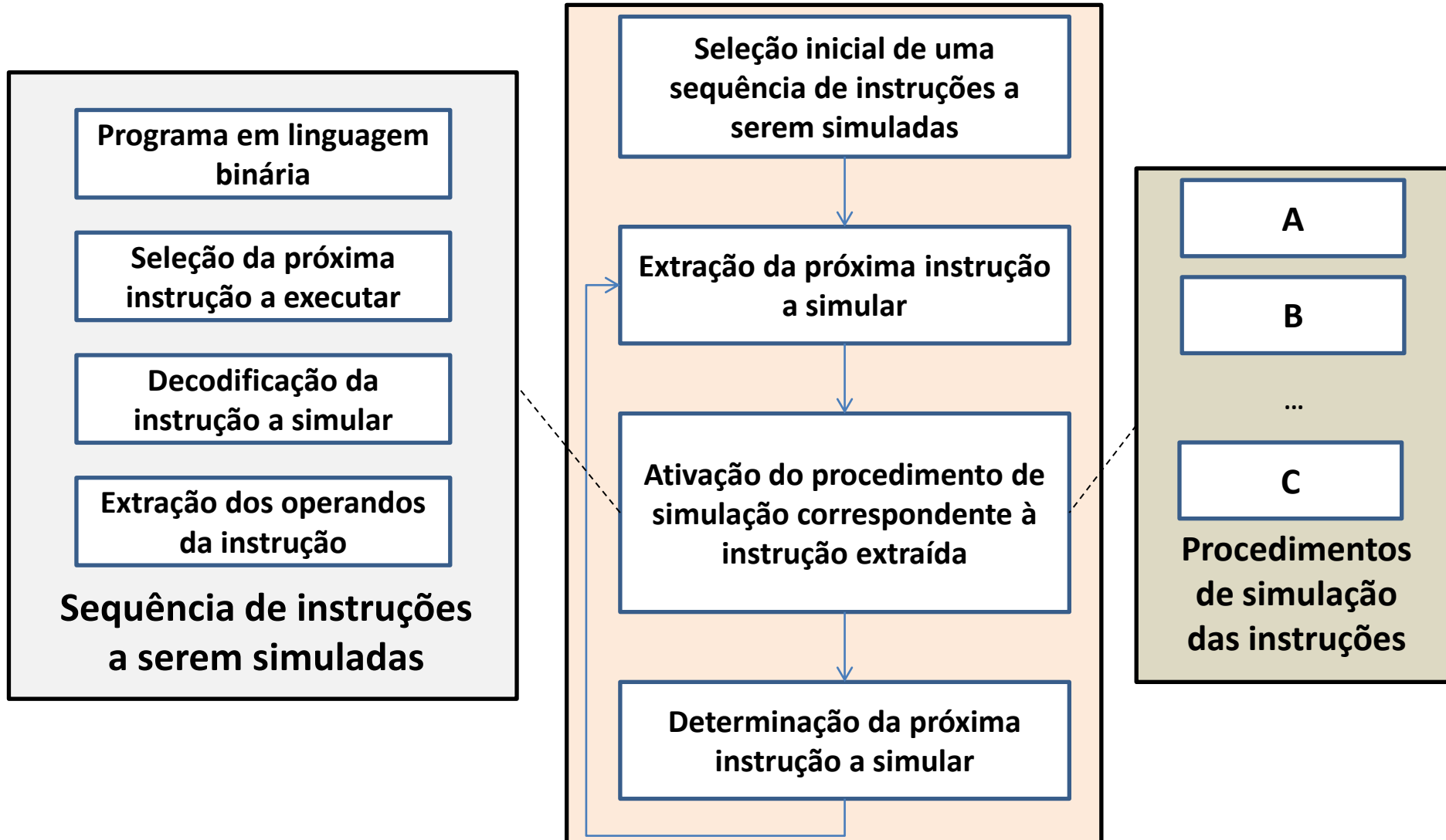
Motor dirigido por eventos, para processamento de textos



Aplicações em simulação de processador

- Máquinas abstratas formais (autômatos e similares)
- Máquinas de calcular, simples e programáveis
- Processador de um conj. de instruções (máq. virtual)
- Compilação e interpretação de ling. de programação
- Processador microprogramável (simul. em 2 níveis)
- Interfaces interativas (interação humano-computador)
- Painéis de supervisão/controle (botões, chaves, indicadores, alarmes, etc)

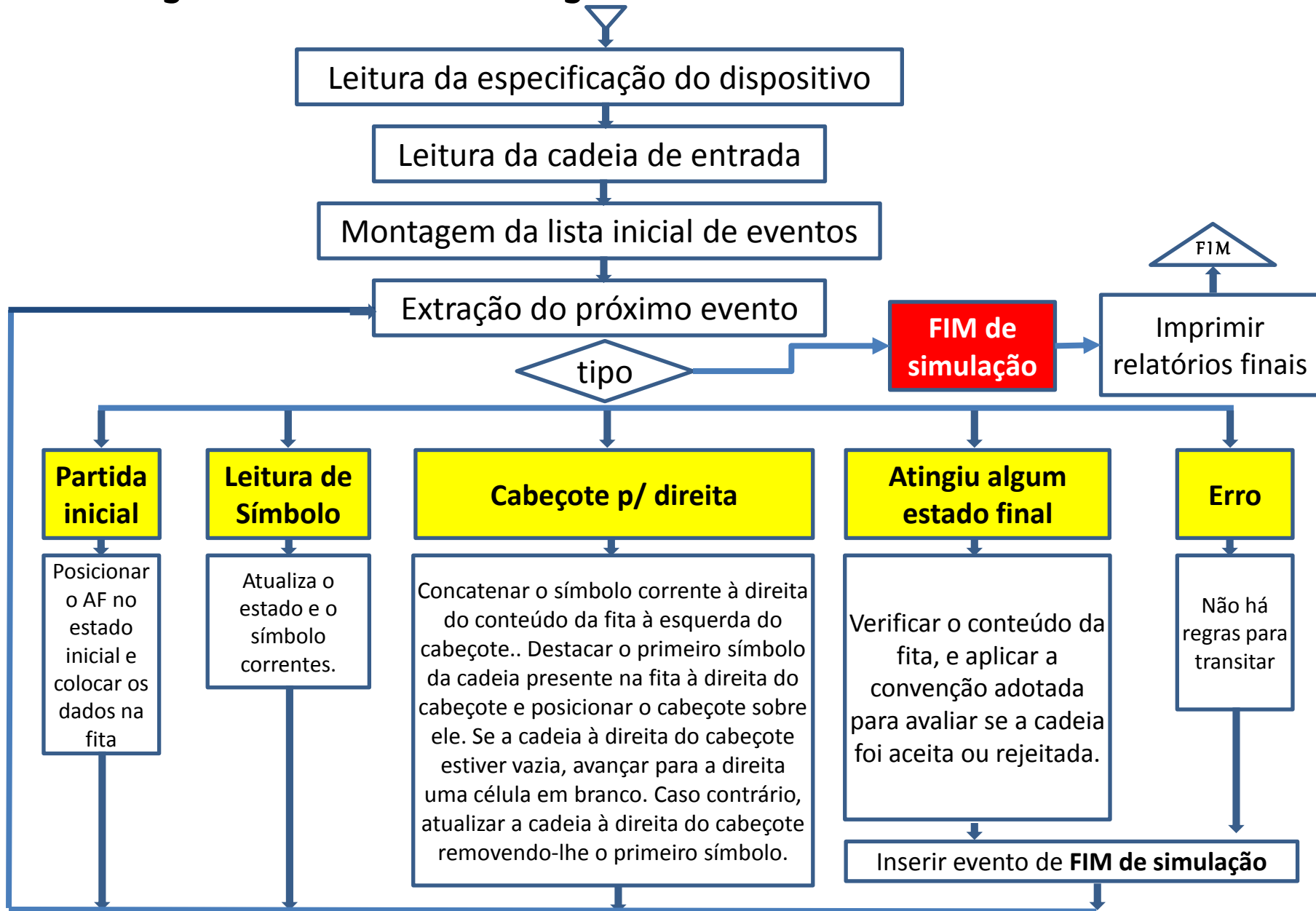
Motor dirigido por eventos para simulação de processadores



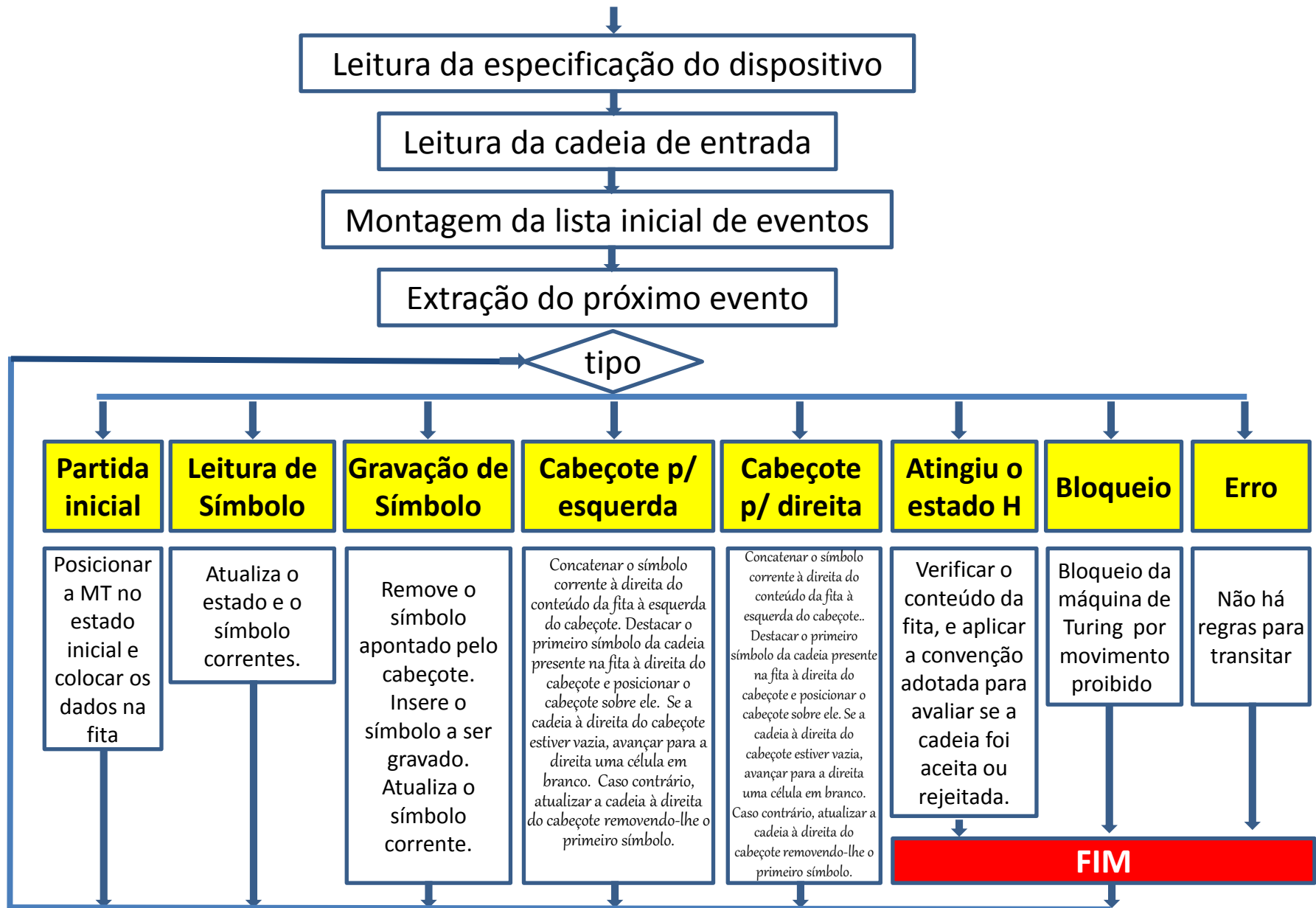
Simulação de aplicações em tempo real

- Sistemas operacionais
 - Sequenciamento de tarefas assíncronas (scheduling)
 - Multiprogramação
 - Protocolos
- Aplicações web
 - Programas com múltiplas interfaces interativas
 - Comércio eletrônico
 - Agendas eletrônicas
- Sistemas reativos
 - Jogos eletrônicos
 - Monitoramento, supervisão e controle de processos
 - Treinamento em simuladores para pilotos de sistemas
 - Ambientes de ensino com o apoio do computador

Esboço da simulação de autômato finito



Esboço de simulação de Máquina de Turing



Estratégia adotada

- Para o projeto da simulação, decidiu-se escolher a **modelagem discreta** dirigida por **eventos**.
- Para organizar o simulador, foi selecionada a técnica clássica da **simulação dirigida por eventos discretos**.
- Para implementar o simulador, optou-se pela construção de um **núcleo (motor de eventos), de uso e aplicação geral**, a desempenhar o papel da componente dinâmica dos mecanismos de simulação.
- Essas **escolhas baseiam-se** na evidência histórica de inúmeros **casos similares bem sucedidos**.
- Assim, um mesmo **motor de simulação** pode ser reutilizado em **variadas aplicações** de simulações desse tipo.

Operação

- Um simulador dirigido por eventos deve gerar respostas a estímulos recebidos do meio externo na forma de uma lista inicial, a qual deve especificar todos os eventos independentes que deverão alimentá-lo.
- Antes de iniciar sua operação, esse programa deverá efetuar uma entrada de dados, através da qual obterá informação acerca do conjunto de eventos independentes que irão estimulá-lo, bem como de seus instantes de ocorrência, os quais são organizados em uma lista cronologicamente ordenada – a lista de eventos.
- Sendo reativo, o sistema simulado responde, na ordem cronológica da lista, a cada um desses eventos, executando, na devida sequência, as correspondentes rotinas de tratamento.
- Como resultado da execução das rotinas de tratamento, são geradas as saídas simuladas do sistema em estudo.

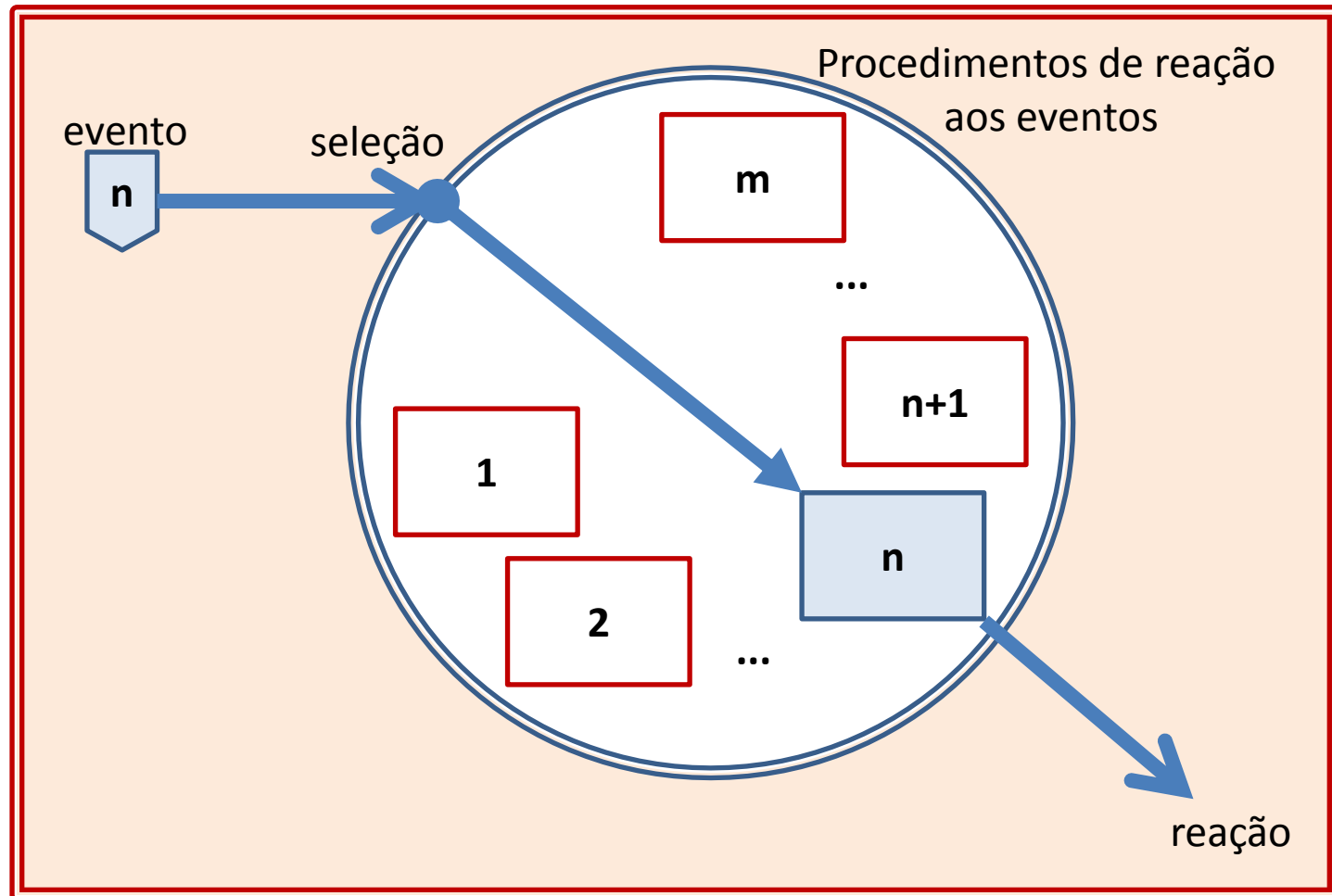
Componentes principais do simulador

- Um **conjunto de tarefas** de simulação, referentes a **fenômenos simultâneos**, a serem executadas concorrentemente
- Um procedimento de **carga das informações iniciais** referentes ao particular alvo de cada uma das tarefas desejadas de simulação
- Uma **lista de eventos**, cronologicamente ordenada, representando a **sequência dos eventos a serem tratados**.
- Um conjunto de **procedimentos de reação aos eventos**, cada qual associado a um dos diversos possíveis tipos de eventos.
- Um **loop de simulação**, composto da **extração do evento** a simular, acompanhado da **seleção do procedimento de reação** correspondente e sua **ativação** .
- Um **procedimento de finalização**, com emissão de resultados.

Lista de eventos

- Os dados assim lidos compõem uma versão inicial de uma lista de eventos, a serem tratados pelo simulador, lista essa que poderá sofrer, durante a simulação, alterações impostas pela dinâmica do processo simulado.
- A lista de eventos geralmente é implementada na forma de uma lista ligada, que se deve manter organizada em ordem crescente, ordenada conforme o instante de ocorrência previsto para os seus diversos eventos.
- Em caso de conflito (quando houver dois ou mais eventos programados para ocorrerem simultaneamente), adota-se algum critério arbitrário de desempate, por exemplo, fazendo prevalecer a ordem de inserção dos eventos na lista, ou seja, tratando em primeiro lugar o evento que tiver sido inserido antes.
- Sendo necessária a inserção, na lista, de eventos a serem programados para ocorrerem em momento anterior ao instante corrente de simulação, isso exigiria artifícios desprovidos de interpretação física. Uma técnica elegante para tratar casos anômalos como este é inserir o evento em uma lista emergencial auxiliar, reservada apenas para tais casos, a qual terá prioridade sobre a fila usual de eventos.

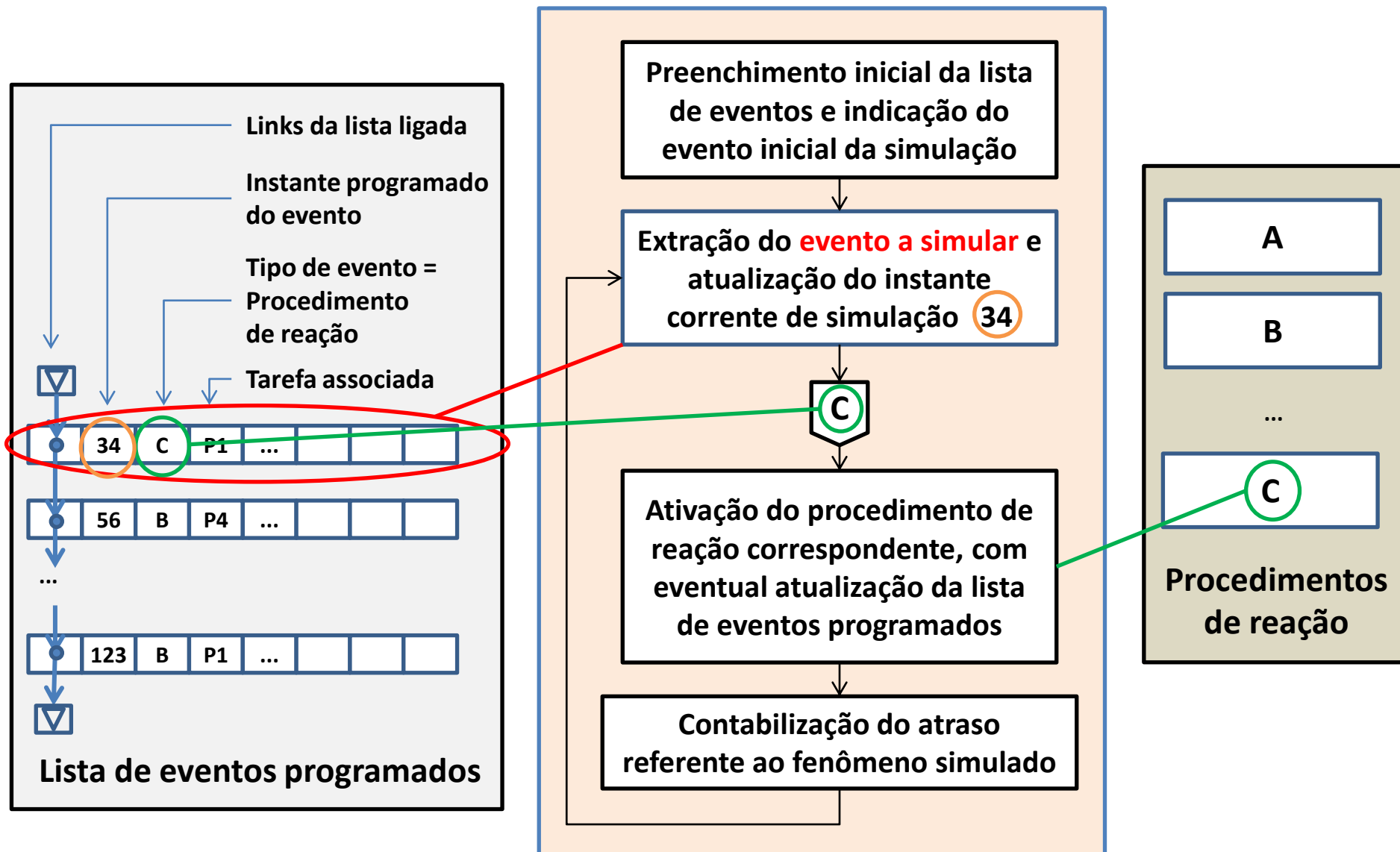
Reação a um evento



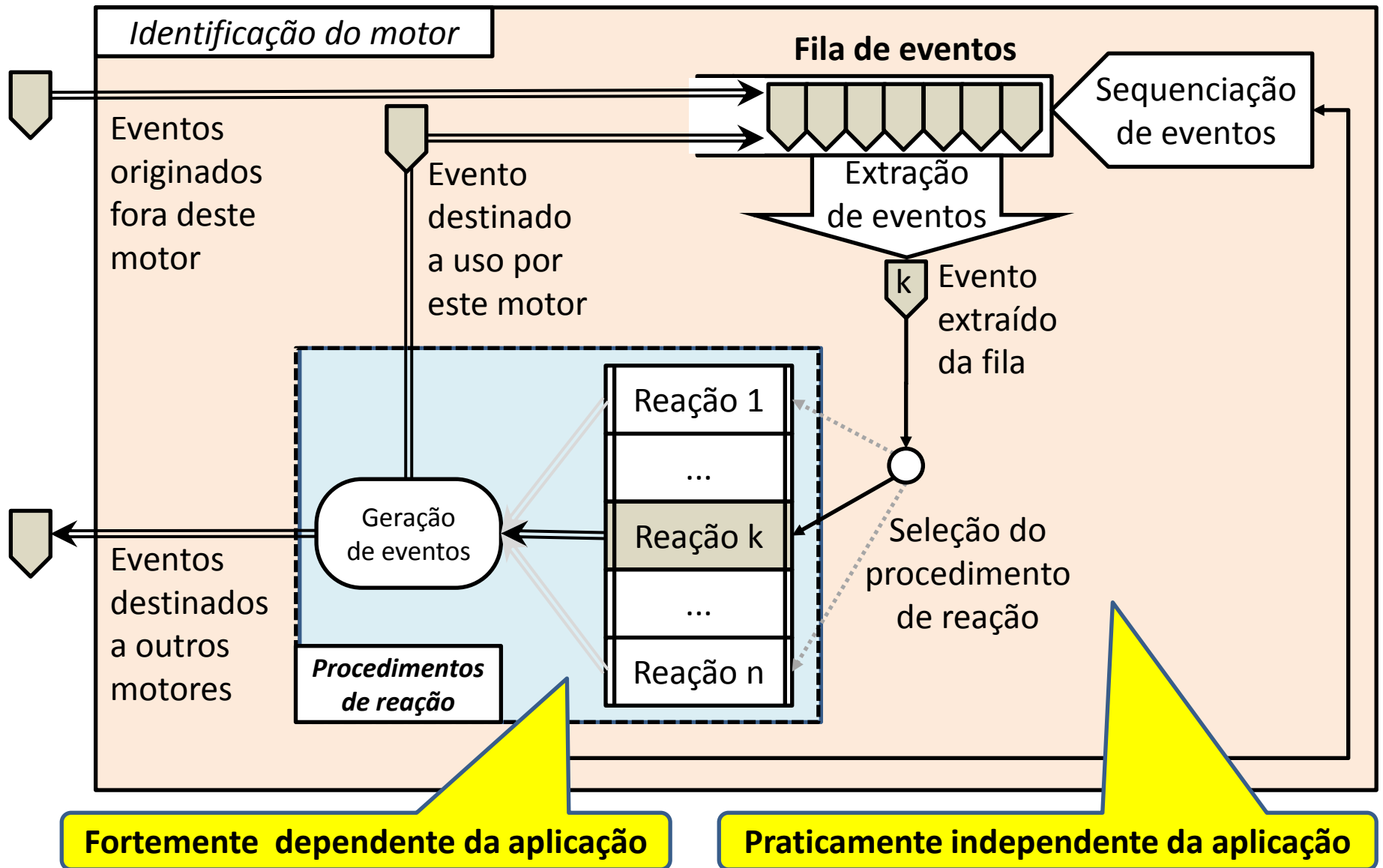
O loop de simulação (temporal)

- Construída a lista inicial de eventos (compreendendo apenas eventos independentes), a simulação propriamente dita pode ter início, e para isso o simulador entra em um ciclo de execução no qual:
 - Um evento é extraído, do início da lista de eventos, para ser tratado pelo simulador
 - Se o instante corrente de simulação for posterior àquele previsto para a ocorrência do evento, este deverá ser simulado como se tivesse ocorrido no instante originalmente previsto para ele
 - Caso contrário, sua ocorrência deverá ser simulada no instante corrente de simulação.
 - Identifica-se o tipo de evento, acionando-se a correspondente rotina de tratamento, de forma análoga ao que é executado pelo hardware quando da interpretação de uma instrução de máquina, ou da ocorrência de um pedido de interrupção.
 - Executada a rotina de tratamento do evento, retorna-se ao primeiro passo, enquanto houver eventos a serem tratados e não for atingido o instante de simulação final especificado.

Esquema de um motor de eventos



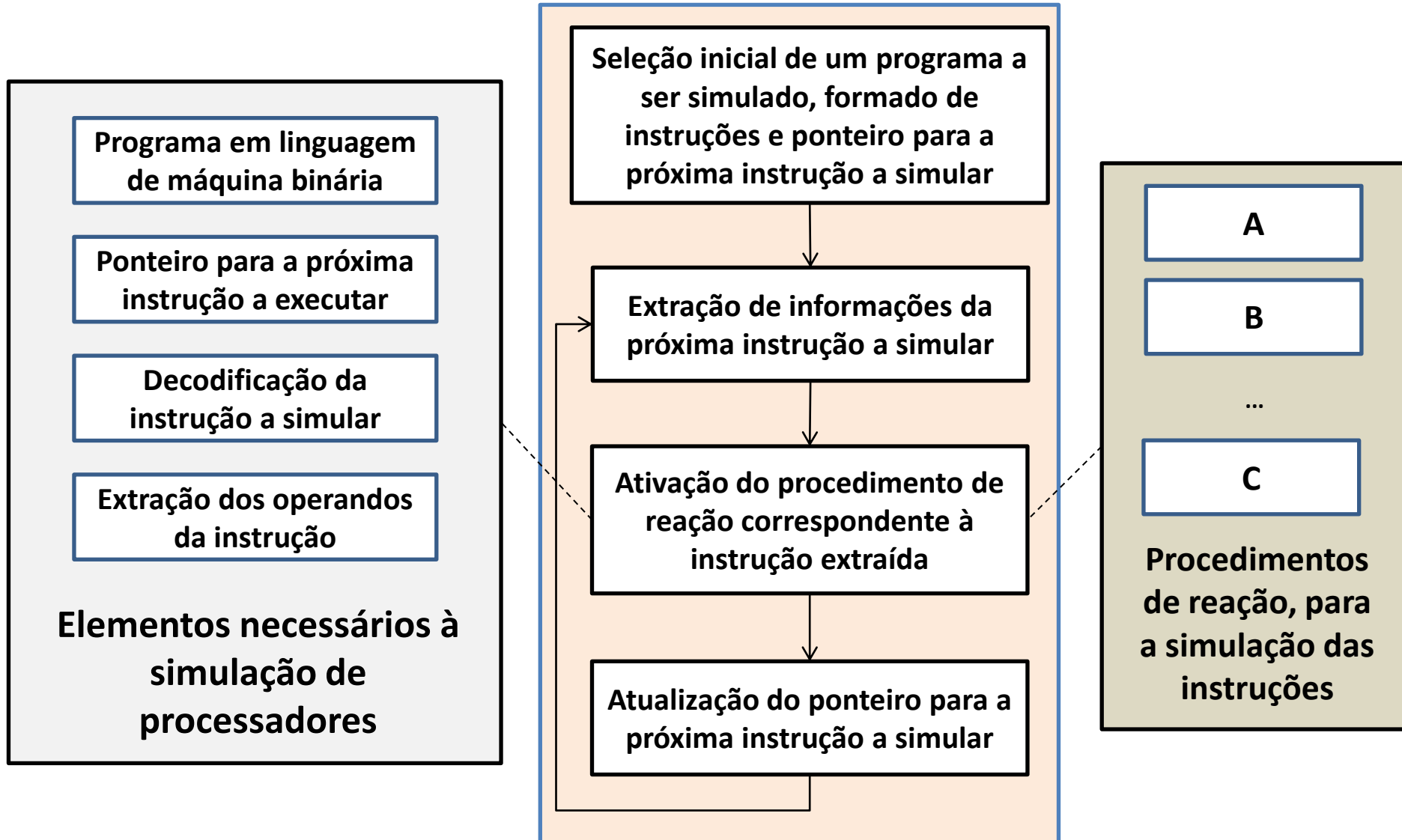
Simulação usando motor de eventos geral



Exemplos de usos do motor de eventos na simulação de computações

- **Simulação de computações**
 - **Máquinas abstratas** formais (autômatos e similares)
 - **Máquinas de calcular**, simples ou programáveis
 - Processador de um **conjunto de instruções** (máq.virtual)
 - **Compilação/interpretação** de linguagem de programação
 - **Processador microprogramado** (simulação em 2 níveis)
 - **Interfaces** interativas (interação humano-computador)
 - Painéis de **supervisão/controle** (botões, chaves, indicadores, alarmes, etc.)

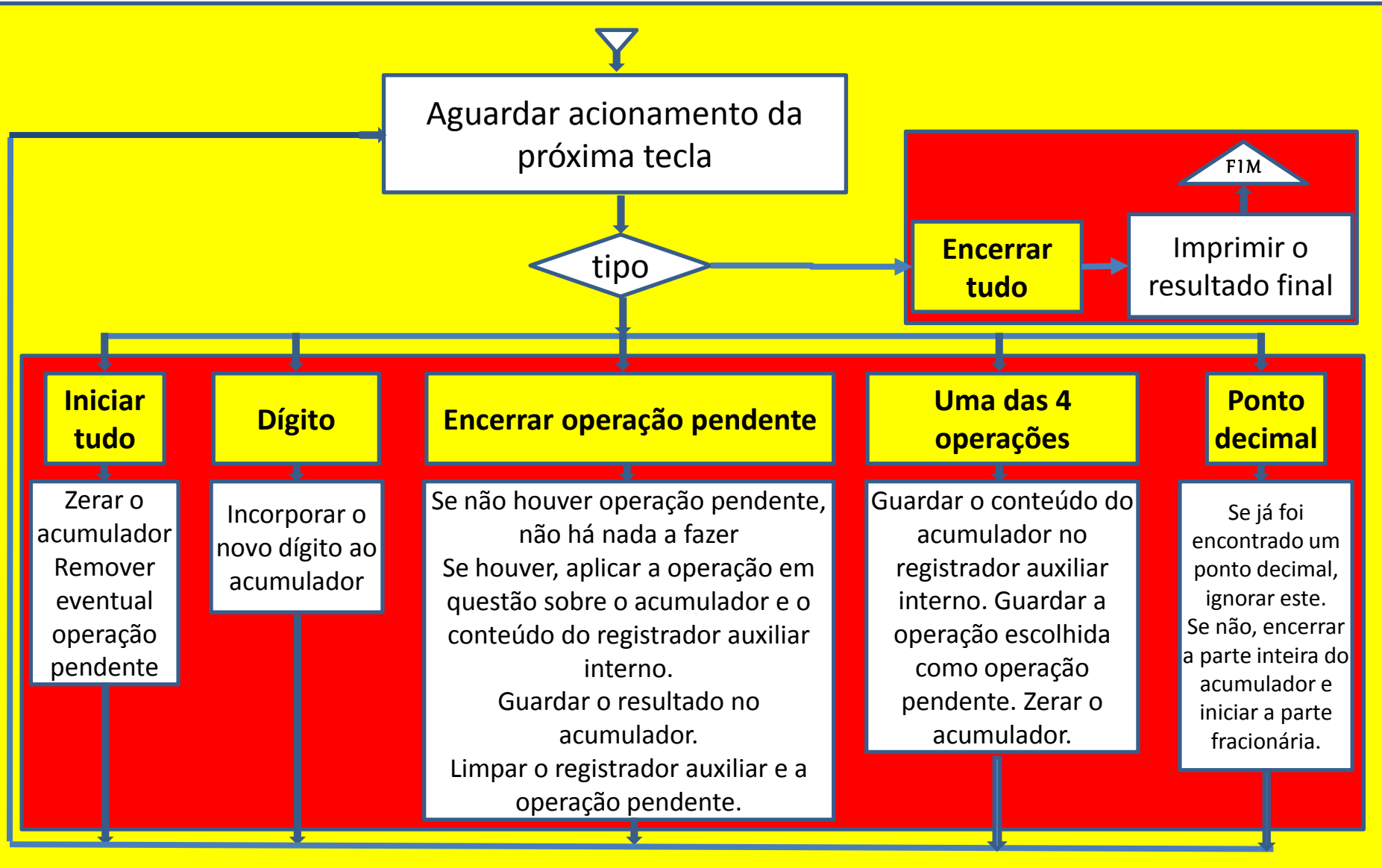
Aplicação de um motor de eventos à simulação de processadores



Calculadora simples - teclas e elementos auxiliares

Tecla	Função associada
Clr	Iniciar tudo
0	dígito 0 (entrada de dados)
1	dígito 1 (entrada de dados)
2	dígito 2 (entrada de dados)
3	dígito 3 (entrada de dados)
4	dígito 4 (entrada de dados)
5	dígito 5 (entrada de dados)
6	dígito 6 (entrada de dados)
7	dígito 7 (entrada de dados)
8	dígito 8 (entrada de dados)
9	dígito 9 (entrada de dados)
+	operador de soma (quatro operações)
-	operador de subtração (quatro operações)
*	operador de multiplicação (quatro operações)
/	operador de divisão (quatro operações)
=	Executar a operação pendente
x	Encerrar tudo
acumulador	visor de trabalho
Registrador auxiliar	para operações diádicas com o acumulador
Operação pendente	registra operação anteriormente acionada

Esboço da lógica do simulador da calculadora



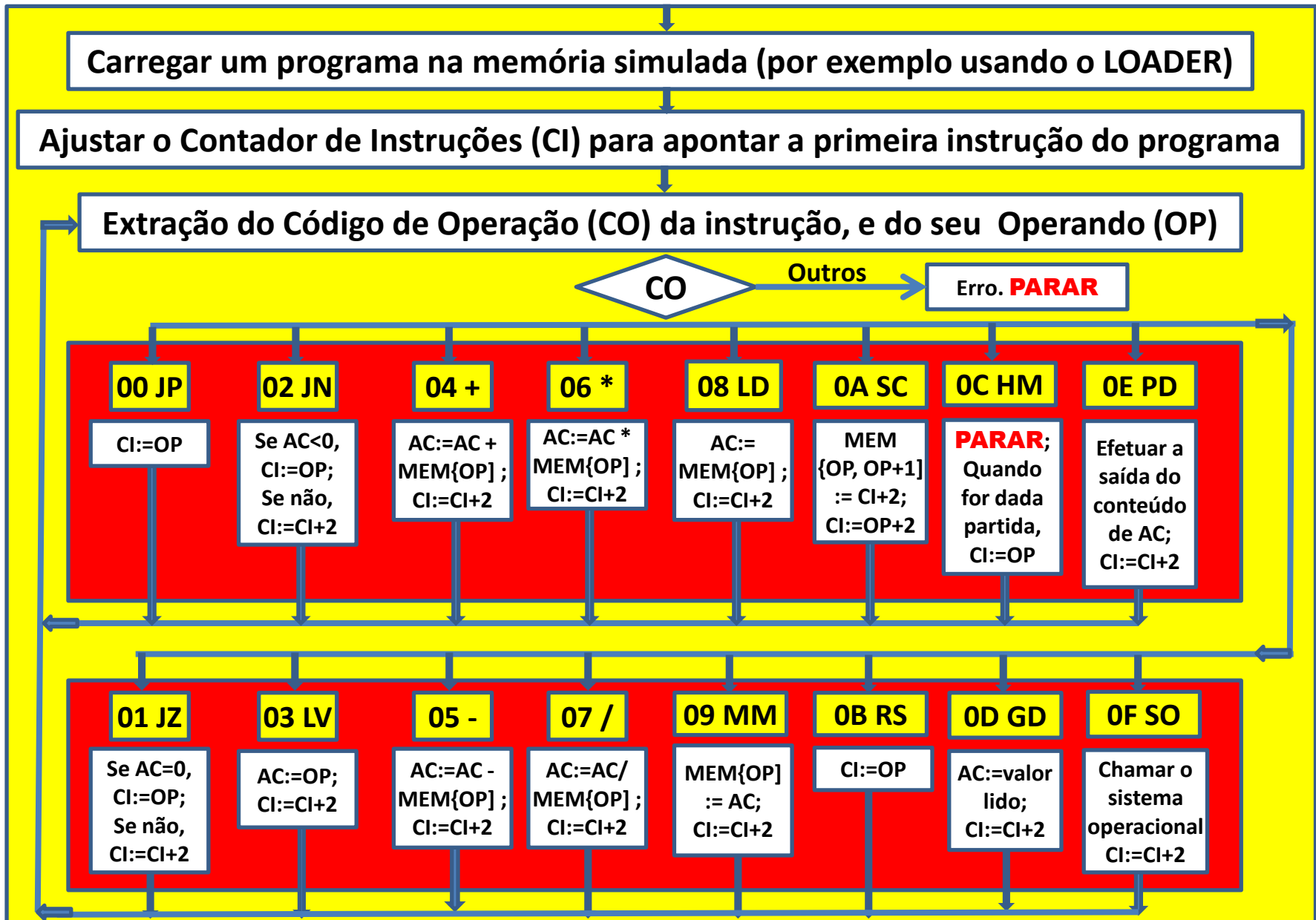
Simulação de instruções de uma MVN

- **Representação essencial da máquina**
 - Memória principal
 - Registradores:
 - Contador de instruções
 - Acumulador
 - Registrador de endereçamento da memória
 - Registrador de dados da memória
 - Principais operações
 - Aritméticas e lógicas
 - Leitura e gravação na memória
 - Entrada e saída de dados
 - Parada de processamento

Tabela de mnemônicos da MVN

; MNEMÔNICOS	CÓDIGO	INSTRUÇÃO / PSEUDO-INSTRUÇÃO
;		
; JP	/0xxx	JUMP INCONDICIONAL
; JZ	/1xxx	JUMP IF ZERO
; JN	/2xxx	JUMP IF NEGATIVE
; LV	/3xxx	LOAD VALUE
; +	/4xxx	ADD
; -	/5xxx	SUBTRACT
; *	/6xxx	MULTIPLY
; /	/7xxx	DIVIDE
; LD	/8xxx	LOAD FROM MEMORY
; MM	/9xxx	MOVE TO MEMORY
; SC	/Axxx	SUBROUTINE CALL
; RS	/Bxxx	RETURN FROM SUBROUTINE
; HM	/Cxxx	HALT MACHINE
; GD	/Dxxx	GET DATA
; PD	/Exxx	PUT DATA
; OS	/Fxxx	OPERATING SYSTEM CALL
;		
; @		ORIGIN
; #		END
; K		CONSTANT

Esboço da lógica do simulador da MVN



FIM