

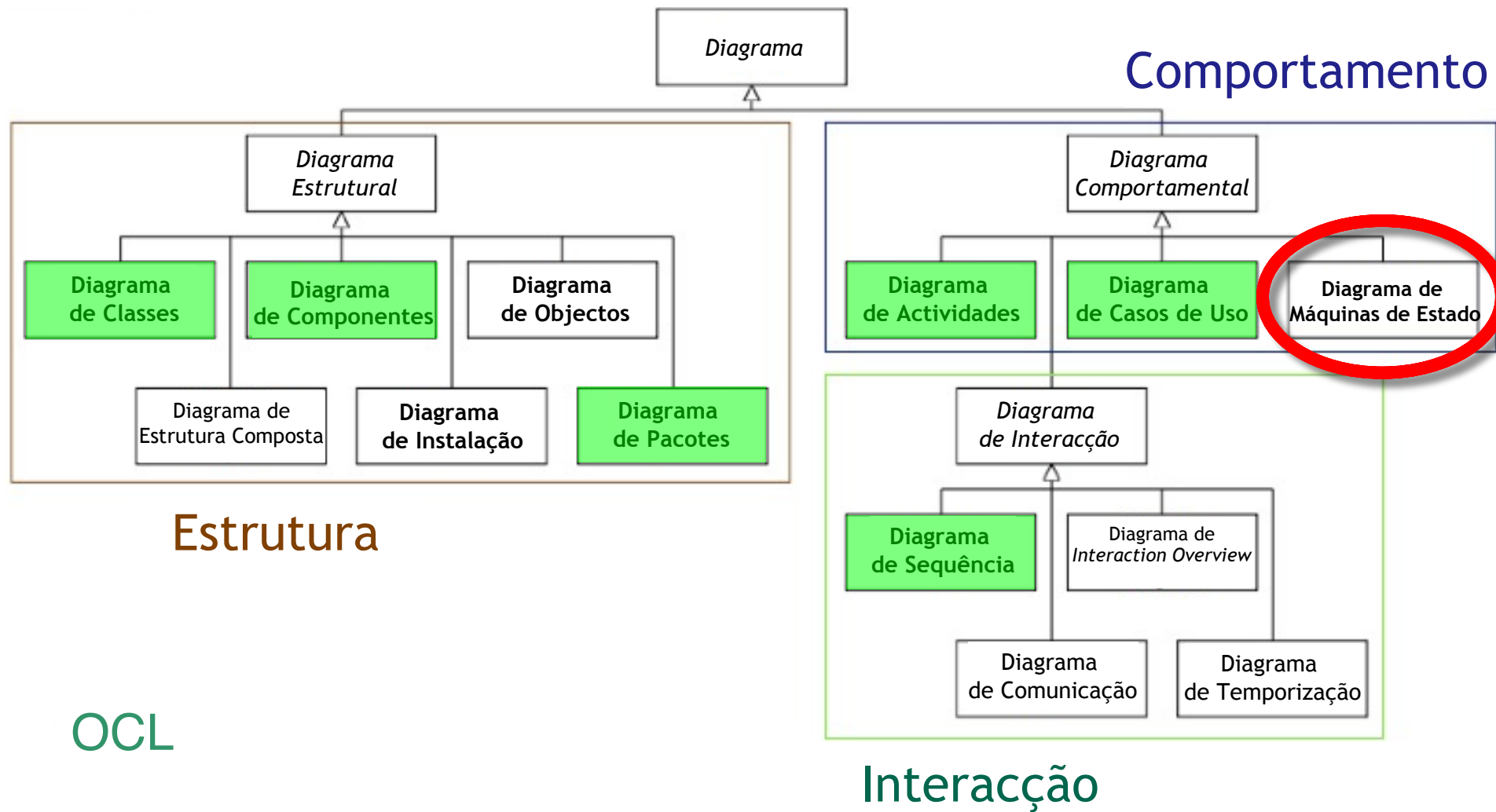


Desenvolvimento de Sistemas Software

Modelação Comportamental (Máquinas de Estado)



Diagramas da UML 2.x





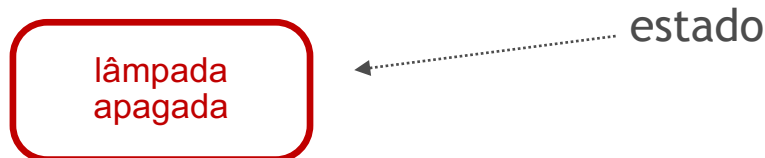
Introdução aos Diagramas de Estado – Aplicação

- Os Diagramas de Estado permitem modelar o comportamento de um dado objecto/sistema de forma global.
- A ênfase é colocada no estado do objecto/sistema — modelam-se todos os estados possíveis que o objecto/sistema atravessa em resposta aos eventos que podem ocorrer.
- Úteis para modelar:
 - O comportamento de um objecto de forma transversal aos use case do Sistema
 - O Sistema como um todo
- Devem utilizar-se para entidades/classes em que se torne necessário compreender o comportamento do objecto de forma global ao sistema.
 - Nem todas as entidades/classes vão necessitar de diagramas de estado.

Diagramas de Estado

Notação base

- Estado — define uma possível estado do objecto (normalmente traduz-se em valores específicos dos seus atributos)



- Pseudoestado inicial — estado do objecto quando é criado



- Estado final — destruição do objecto



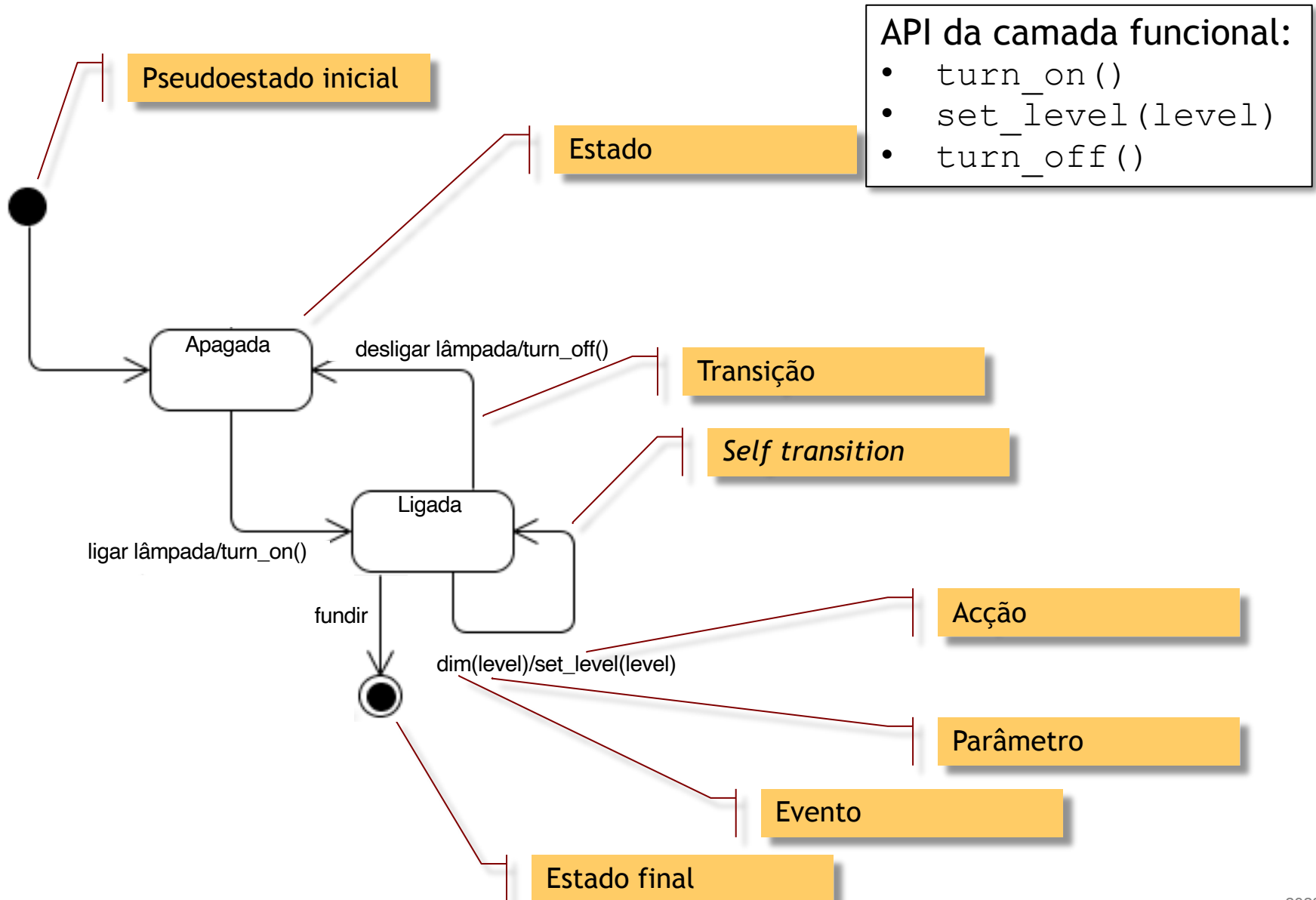
- Transições — evento[guarda]/acção (todos são opcionais!)



Sem evento, a transição acontece automaticamente



Maquina de Estados básica

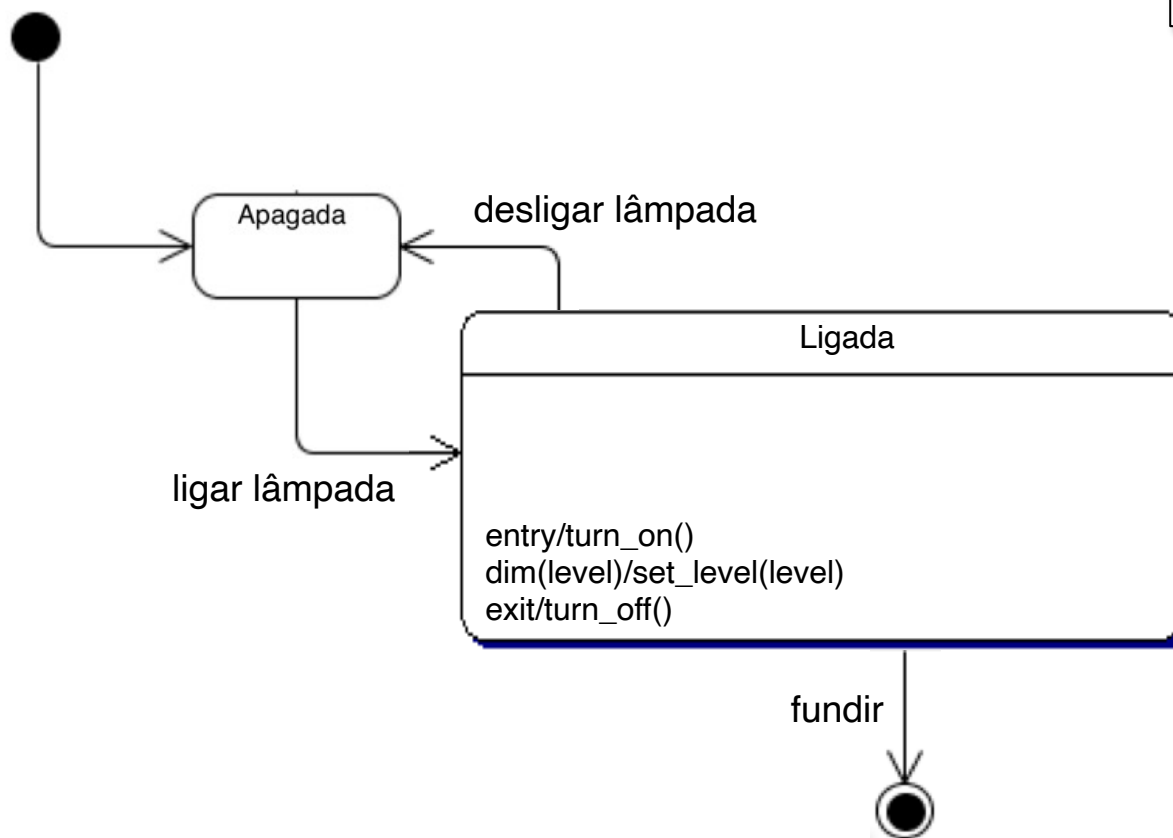




Atividades internas

API da camada funcional:

- `turn_on()`
- `set_level(level)`
- `turn_off()`





Actividades internas

- Actividades que não provocam transições de estado...

entry/acção

- “acção” é automaticamente executada quando o objecto entra no estado;

do/acção

- “acção” é continuamente executada enquanto o objecto estiver no estado;

exit/acção

- “acção” é automaticamente executada quando o objecto sai do estado;

evento/acção

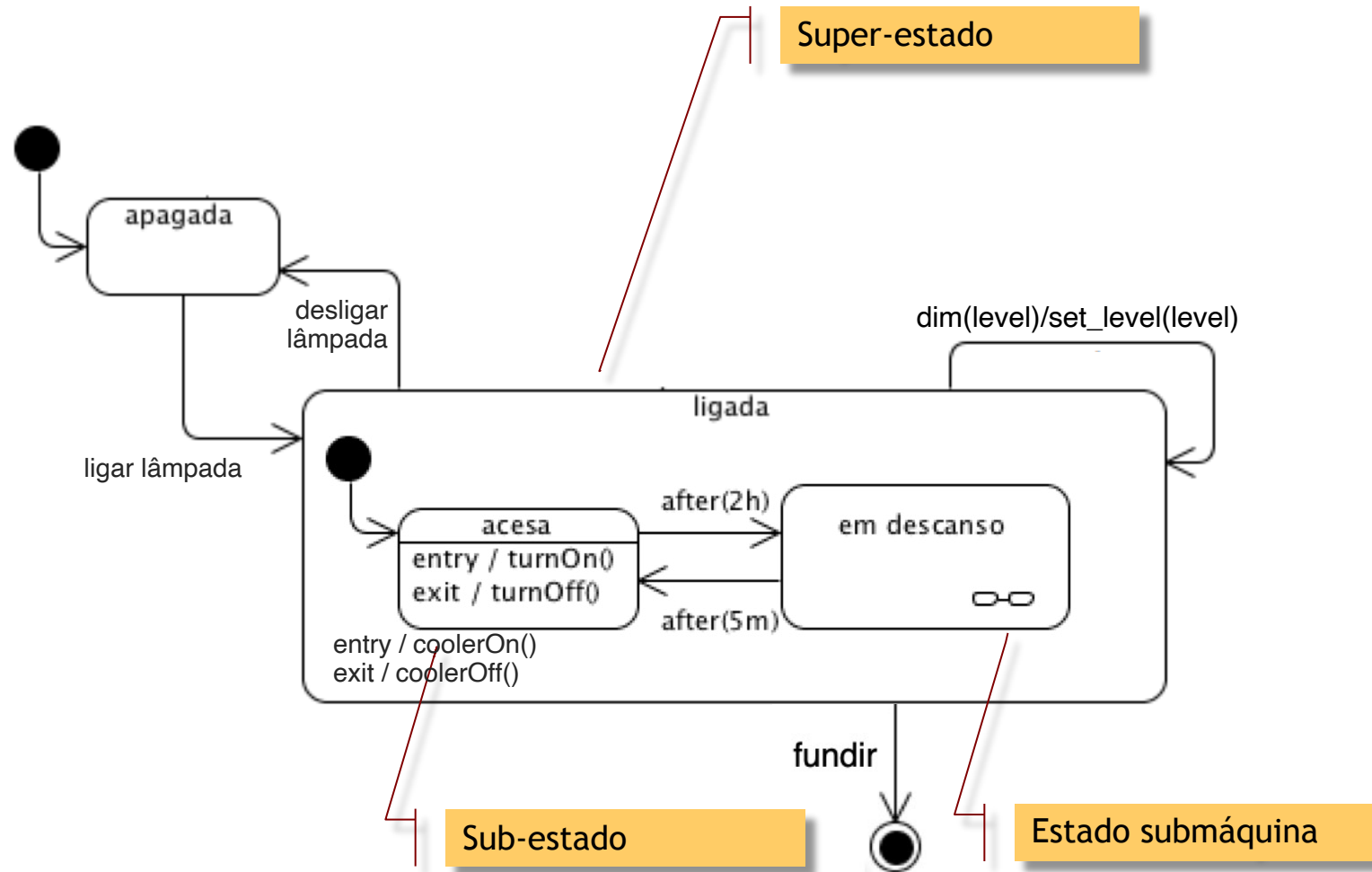
- “acção” é automaticamente executada se “evento” ocorrer (transição interna);

evento/defer

- “evento” é deferido até o estado actual ser abandonado.

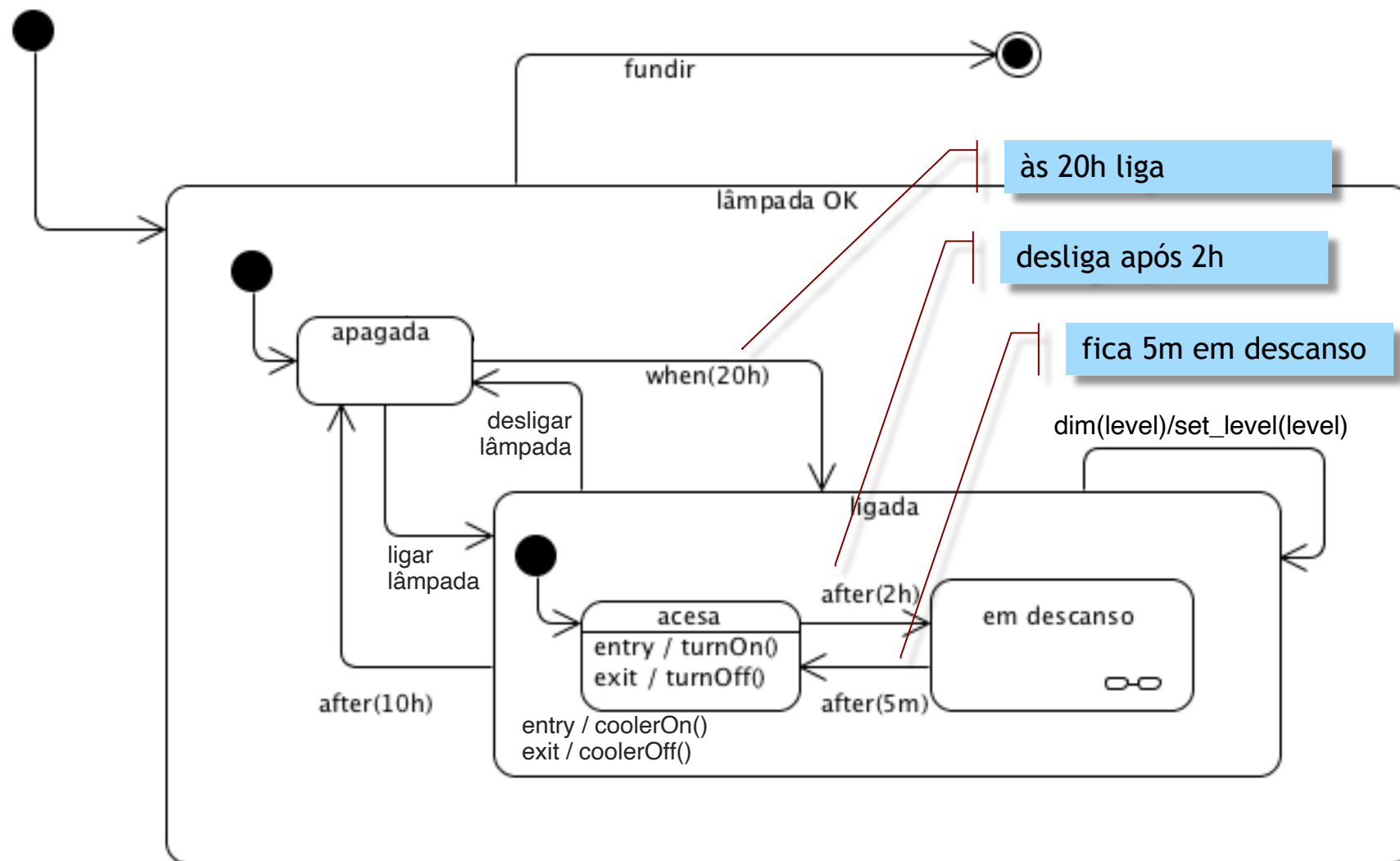


Estados e estados compostos (super-estados)



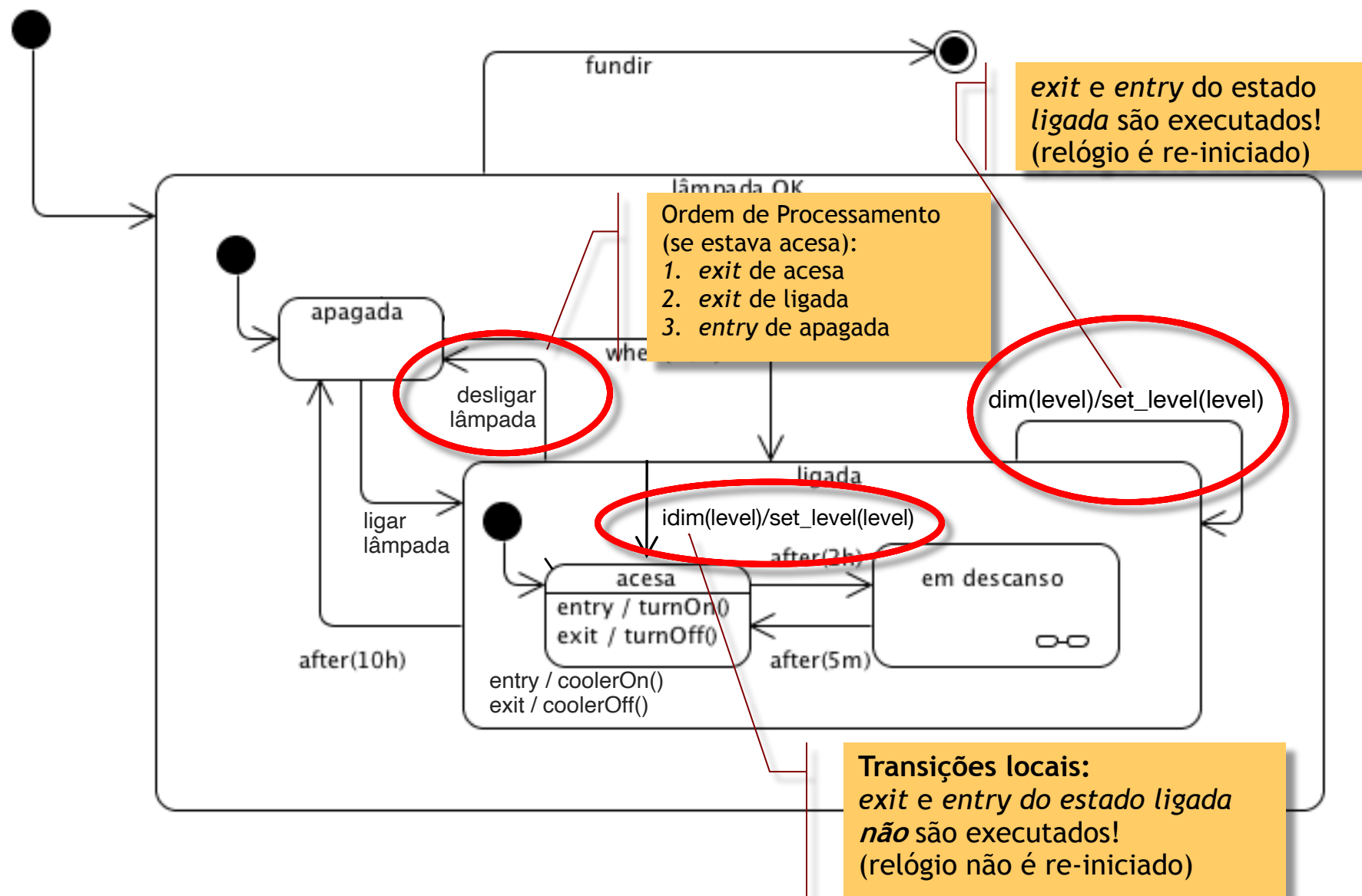


Eventos *when* / *after*





Transições vs. actividades internas

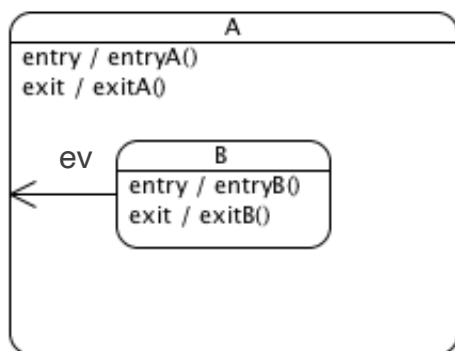




Transições locais vs. transições externas

Em resposta ao evento **ev**, o modelo...

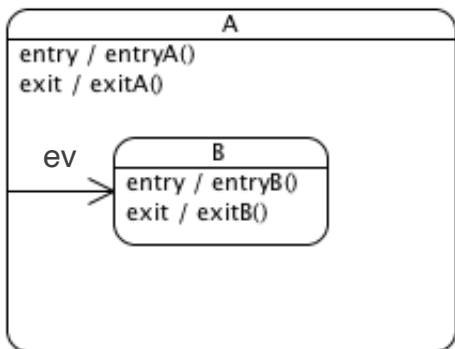
Transições locais



Executa:

- exitB()
- ...

(sub-estado para super-estado)

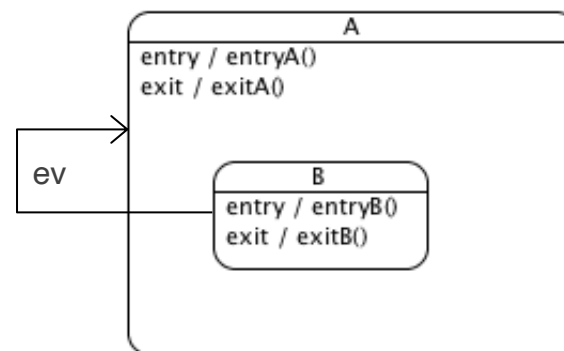


Executa:

- ...
- entryB()

(super-estado para sub-estado)

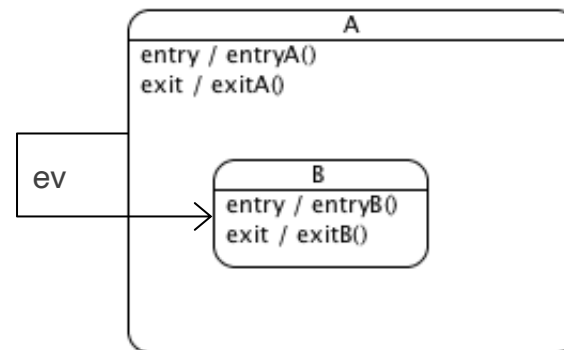
Transições externas



Executa:

1. exitB()
2. exitA()
3. entryA()
4. ...

(sub-estado para super-estado)



Executa:

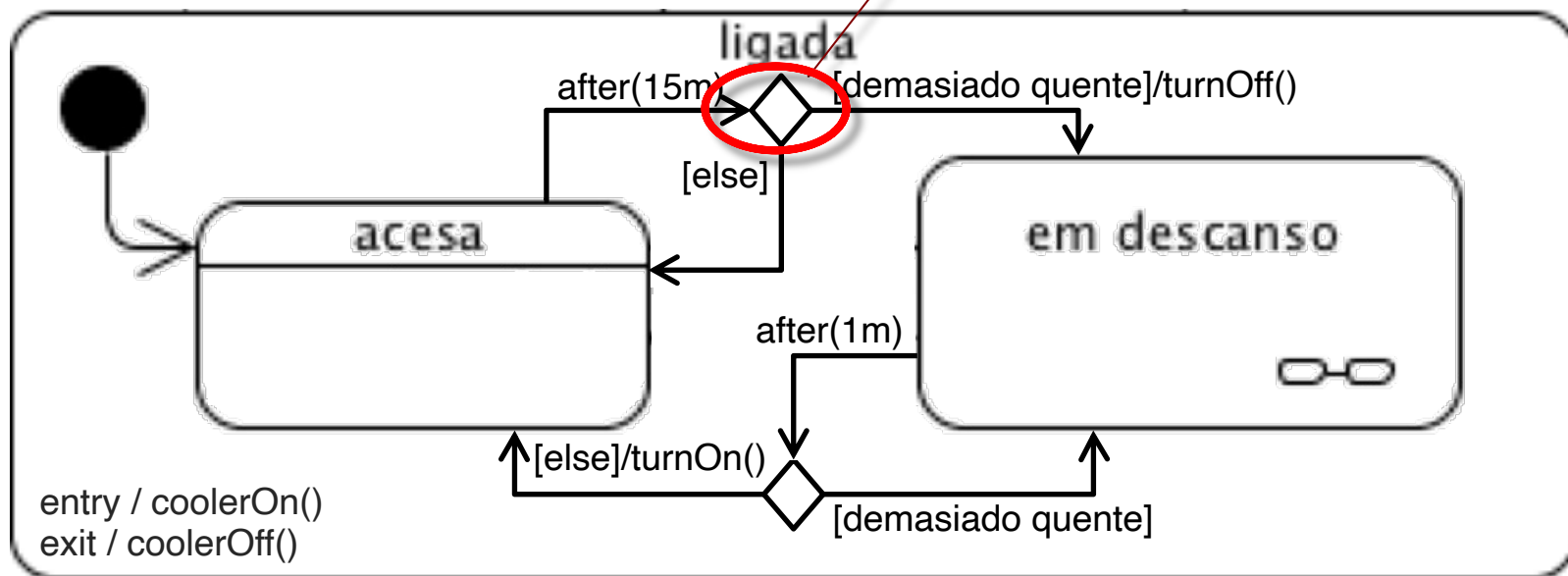
1. ...
2. exitA()
3. entryA()
4. entryB()

(super-estado para sub-estado)

Pseudoestado de Escolha

- Ramificação condicional (dinâmica!) em função do valor de uma expressão.
- Decisão pode ser uma função de acções anteriores.
- Caso mais que uma guarda verdadeira, a escolha é não determinística.
- Se nenhuma guarda for verdadeira, o modelo está mal formado ([else]!)

Pseudo-estado de Escolha



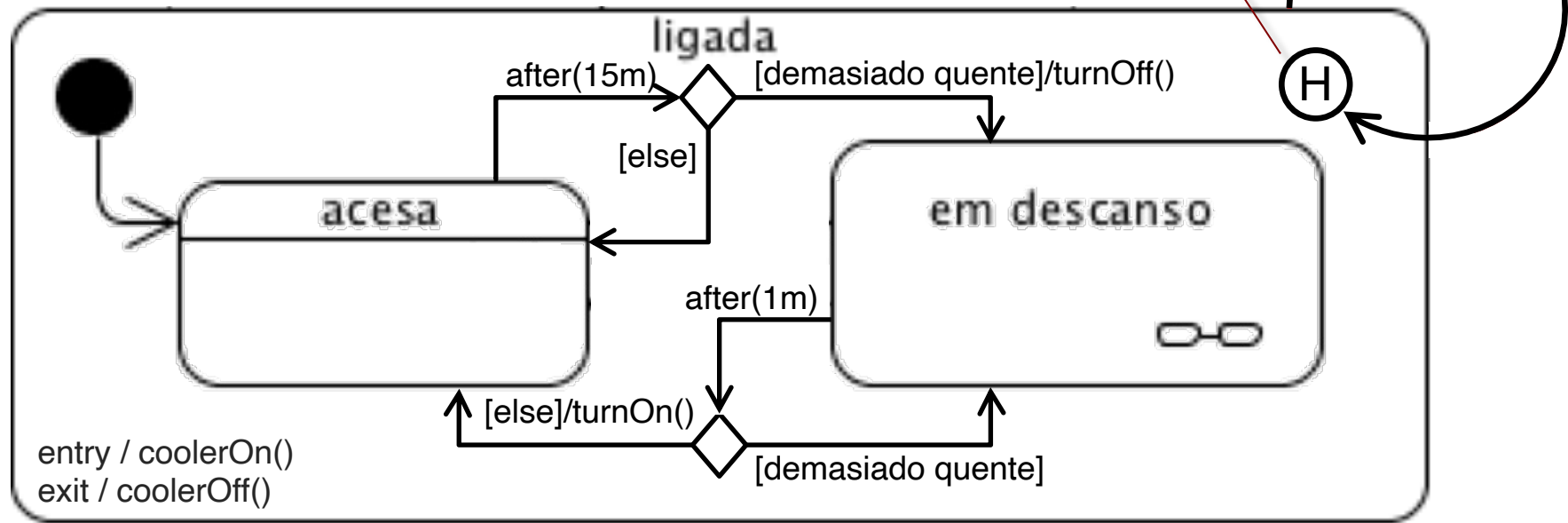
Pseudoestados de História

- Permitem modelar interrupções – actividade da máquina é retomada no estado em que se encontrava aquando da última saída

- (H) shallow history
- (H*) deep history

Shallow history
Regressa ao início do subestado (acesa/em descanso) em que estava

dim(level)/set_level(level)





Pseudo-estados de História

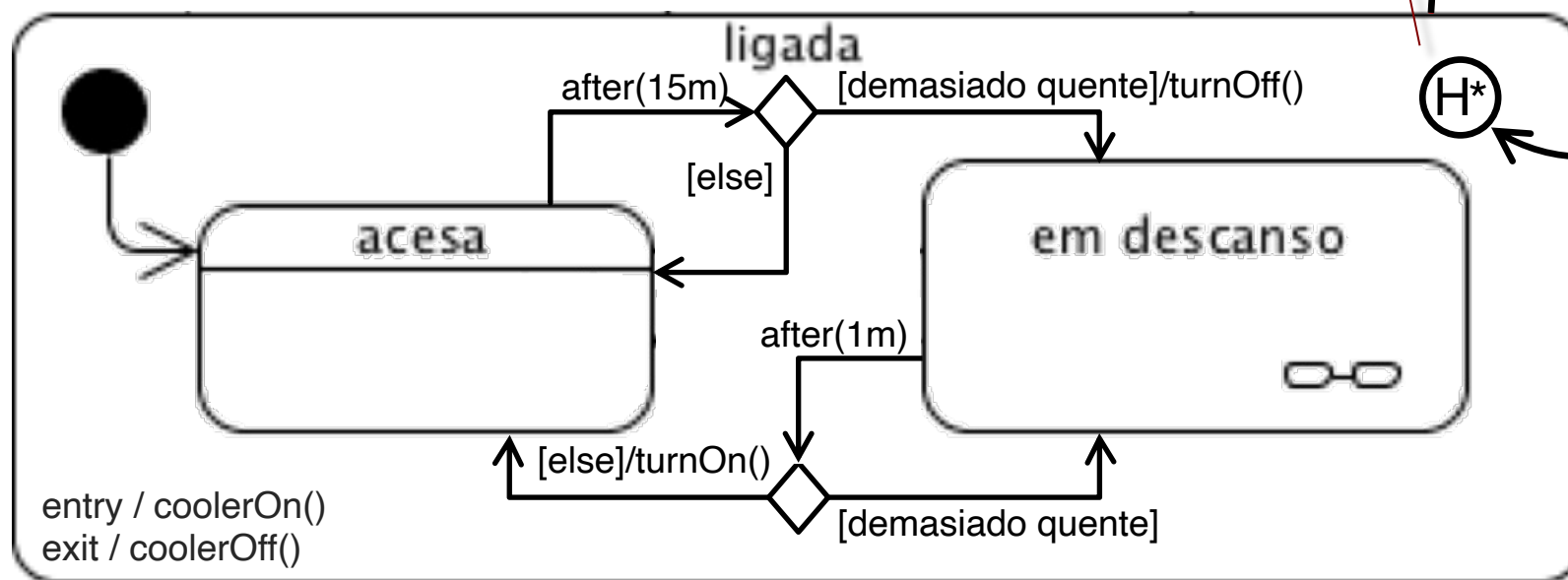
- Permitem modelar interrupções – actividade da máquina é retomada no estado em que se encontrava aquando da última saída

- \textcircled{H} shallow history
- $\textcircled{H^*}$ deep history

Deep history

Regressa subestado (acesa/em descanso) em que estava e, se ele tiver subestados, ao subestado em que estava dentro dele, e assim sucessivamente.

$\text{dim}(\text{level})/\text{set_level}(\text{level})$





Resumo da notação (até agora)

	Estado
	Estado composto
	Estado submáquina
	Pseudoestado inicial
	Estado final
	Transição (evento [condição] / acção) (entre estados vs. para o próprio estado vs. locais)
	Pseudoestado de escolha
	Pseudoestados de história (shallow/deep)

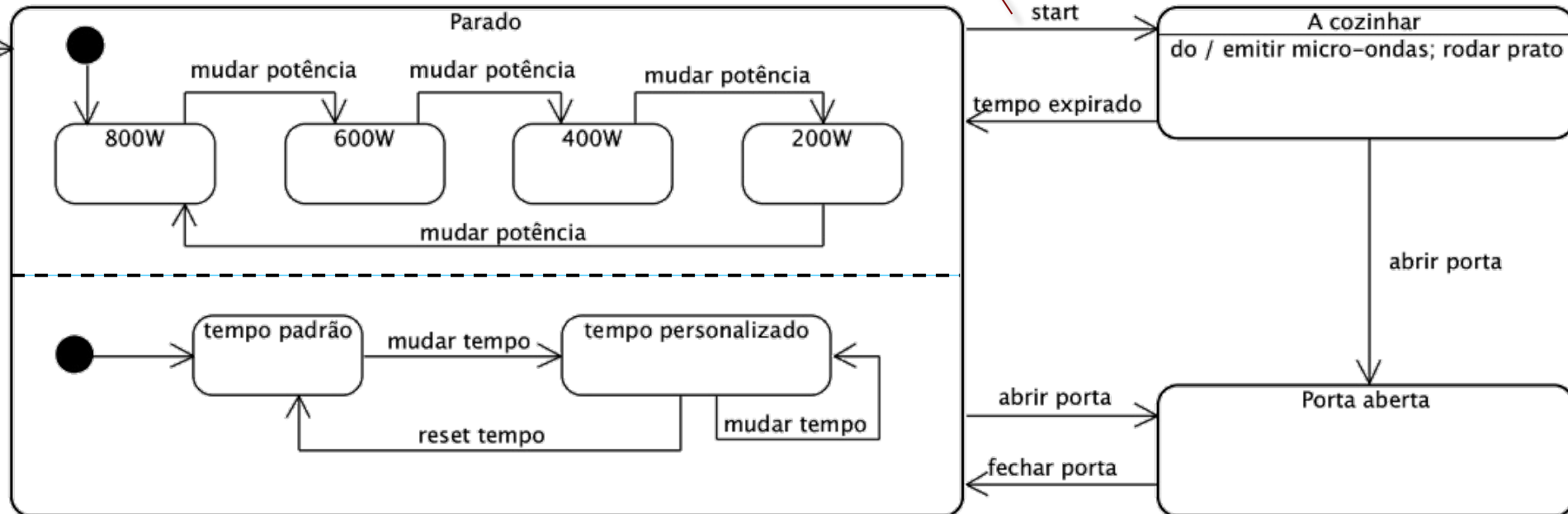


Estados com concorrência...

- Um estado pode ser dividido em “regiões” ortogonais
- Cada região contém um subdiagrama
- Os diagramas das regiões são executados de forma concorrente

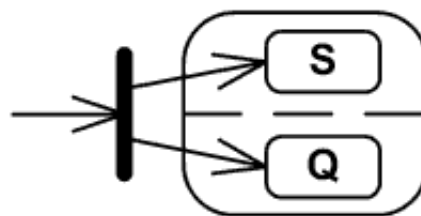
fork implícito

join implícito

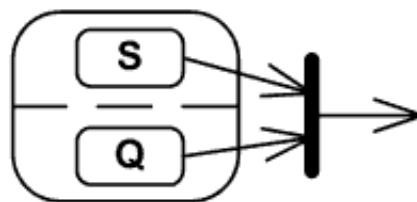


Pseudoestados *fork* e *join*

- Permitem gerir concorrência.
- Fork - divide uma transição de entrada em duas ou mais transições
 - Transições de saída têm que terminar em regiões ortogonais distintas

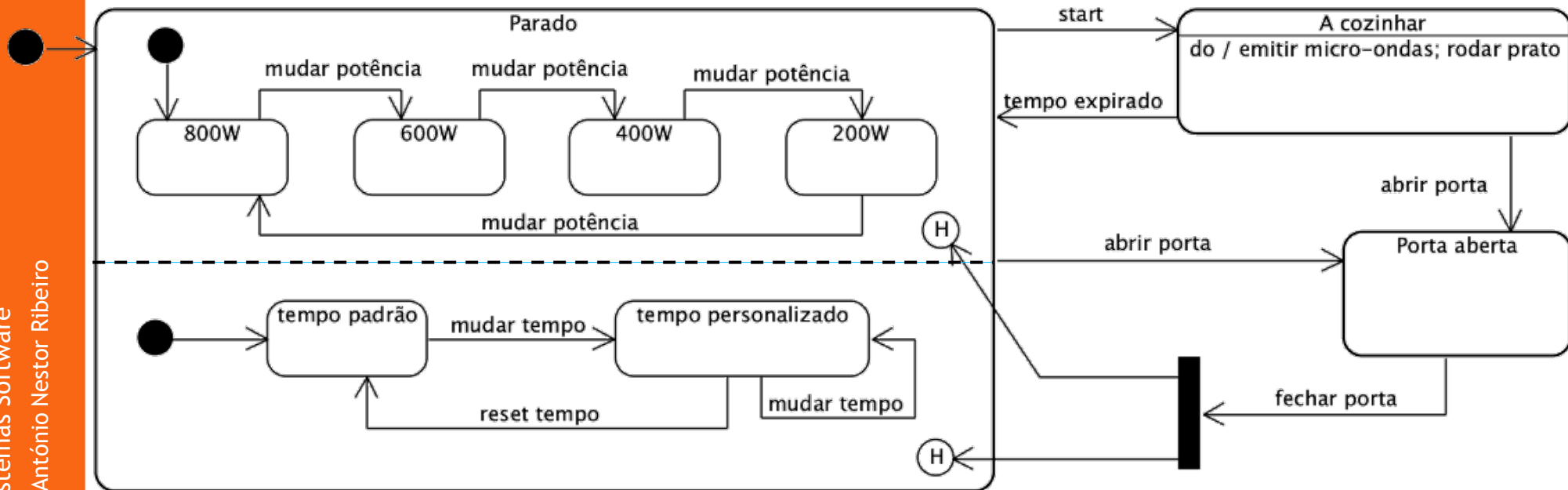


- Join - funde duas ou mais transições de entrada numa só transição de saída
 - Transições de entrada têm que originar em regiões ortogonais distinta



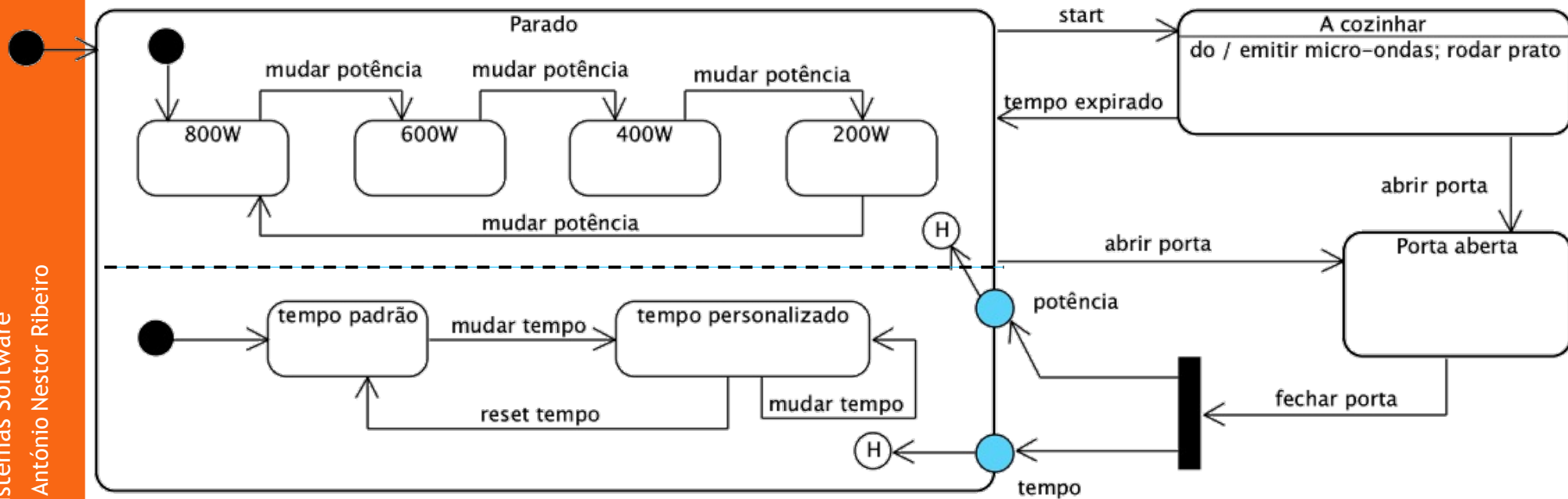


Estados com concorrência...



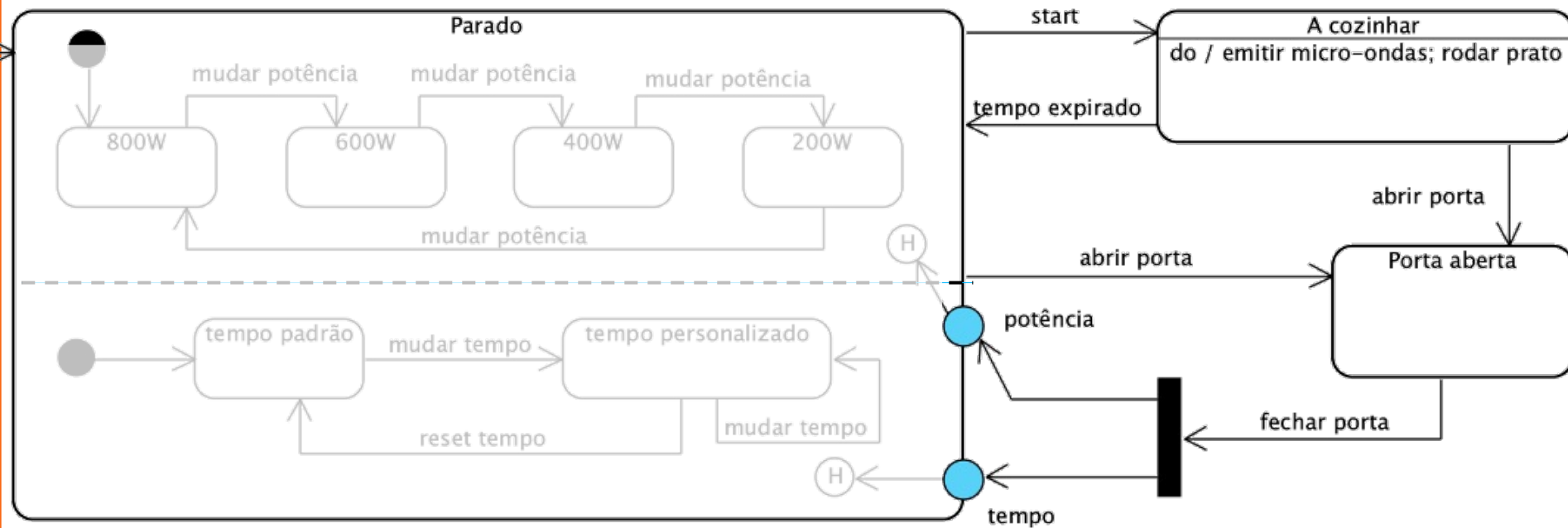


Estados com concorrência...







Estados com concorrência...





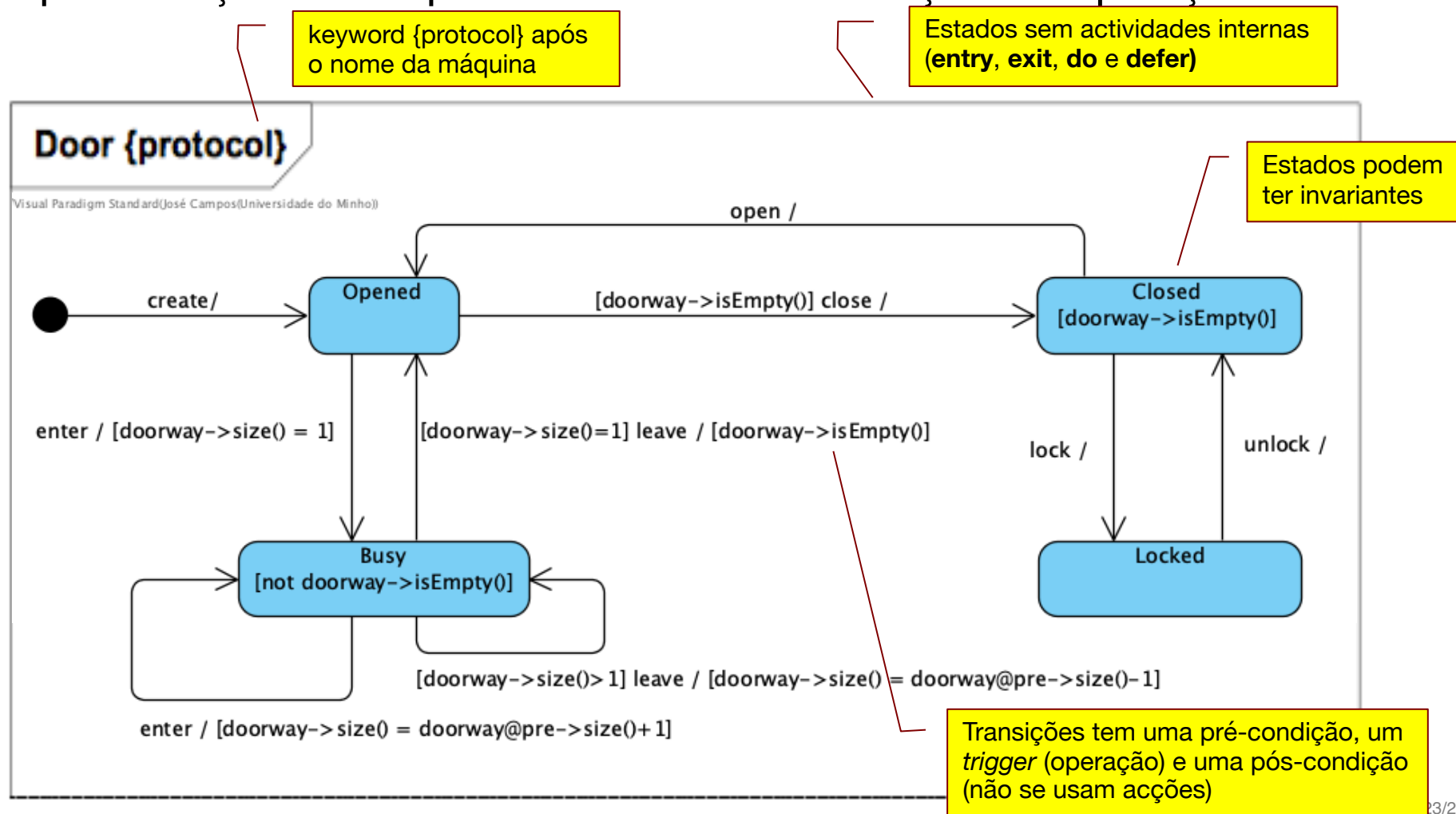
Pseudoestados Ponto de entrada e Ponto de saída

- Ponto de entrada 
 - Permite definir um ponto de entrada numa máquina de estados ou num estado composto
 - O ponto de entrada é identificado por nome
 - O ponto de entrada transita para um estado interno que poderá ser diferente do definido pelo estado inicial
- Ponto de saída 
 - Permite definir um ponto de saída alternativo ao estado final
 - O ponto de saída é identificado por nome



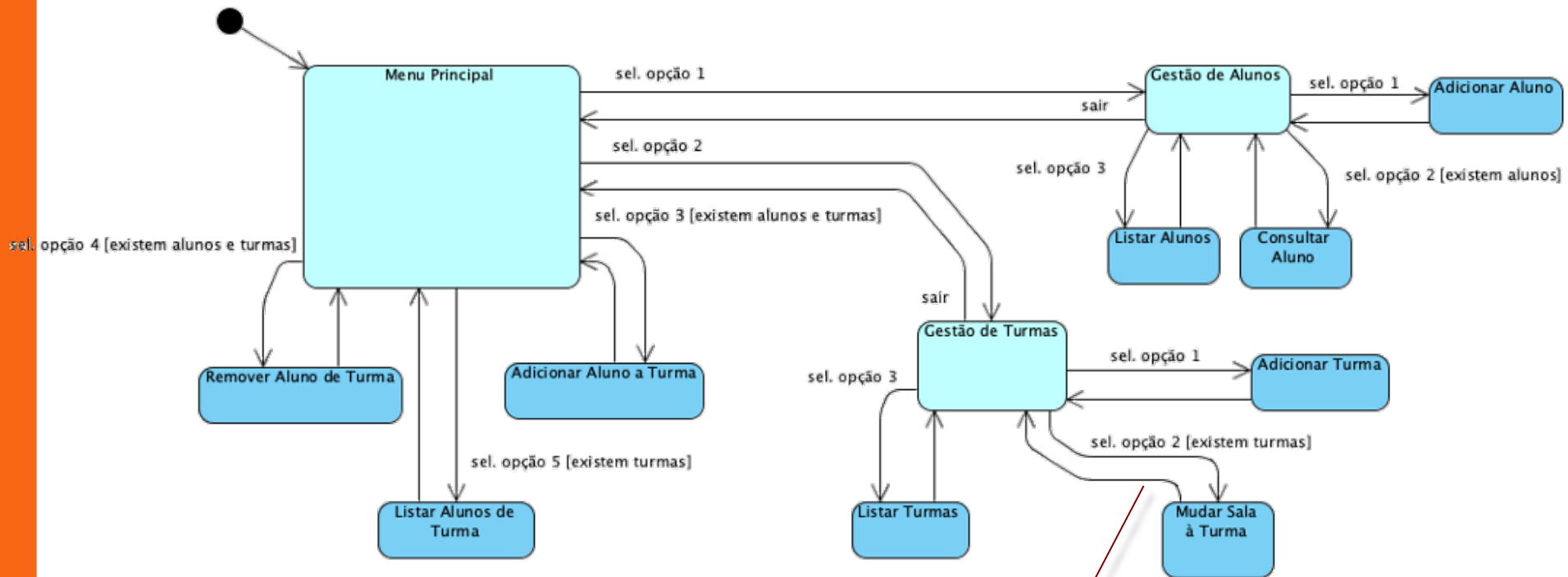
Protocol State Machines

- Especificam que operações podem ser invocadas em cada estado e em que condições - a sequências válidas de invocação das operações.





Modelação do controlo de diálogo da interface



```
public TextUI() {
    // Criar o menu
    this.menu = new Menu(new String[]{ "Adicionar Aluno", ...
    this.menu.setHandler(1, this::trataAdicionarAluno);
    this.menu.setHandler(2, this::trataConsultarAluno);
    this.menu.setHandler(3, this::trataListarAlunos);
    this.menu.setHandler(4, this::trataAdicionarTurma);
    this.menu.setHandler(5, this::trataMudarSalaTurma);
    this.menu.setHandler(6, this::trataListarTurmas);
```

Completion transition
Não tem evento, dispara automaticamente logo que possível.



Diagramas da UML 2.x

