

08. Flussi di attività in UML

IS 2024-2025



Laura Semini, Jacopo Soldani

Corso di Laurea in Informatica

Dipartimento di Informatica, Università of Pisa

DIAGRAMMA DELLE ATTIVITÀ

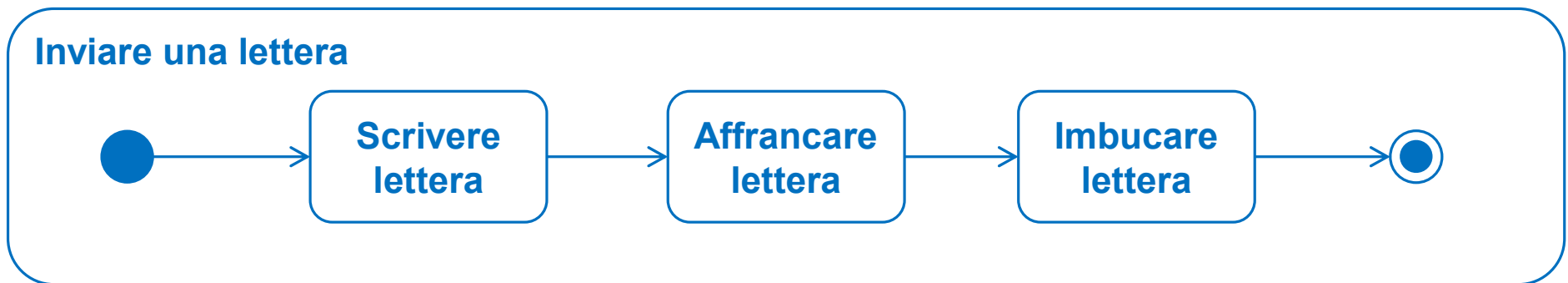
- Modella un **workflow** (aka. flusso di lavoro)
 - Ad esempio, un algoritmo o un processo
 - Antenati: flow chart e Petri net
- Descrive come **coordinare** un insieme di **azioni**
 - Sequenze
 - Scelte condizionali
 - Iterazioni
 - Concorrenza
- Modella un'attività relativa a **una o più entità**, ad esempio
 - Una o più classi che collaborano in un'attività comune
 - Uno o più attori che interagiscono con il sistema
 - Un'operazione offerta da una classe

ESEMPI DI UTILIZZO

- Modellare un processo aziendale (in fase di analisi)
- Modellare il flusso di un caso d'uso (in fase di analisi)
- Modellare il funzionamento di un'operazione di una classe (in fase di progettazione)
- Modellare un algoritmo (in fase di progettazione e/o testing)

ATTIVITÀ IN UML

Un'**attività** ha un nome ed è contenuta in un rettangolo con gli angoli smussati



Il contenuto di un'attività è un **grafo diretto**

- I **nodi** rappresentano le componenti dell'attività, ovvero **azioni** e nodi di controllo (come inizio e fine attività, ad esempio)
- Gli **archi** rappresentano il **control flow**, inteso come i possibili **path eseguibili**

INIZIO E FINE DI UN'ATTIVITÀ

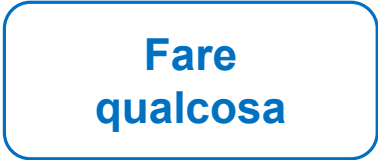
Nome attività



AZIONI

Le **azioni** sono rappresentate da rettangoli con angoli smussati

- Il nome deve descrivere un'azione \Rightarrow deve essere un **verbo**



Fare
qualcosa



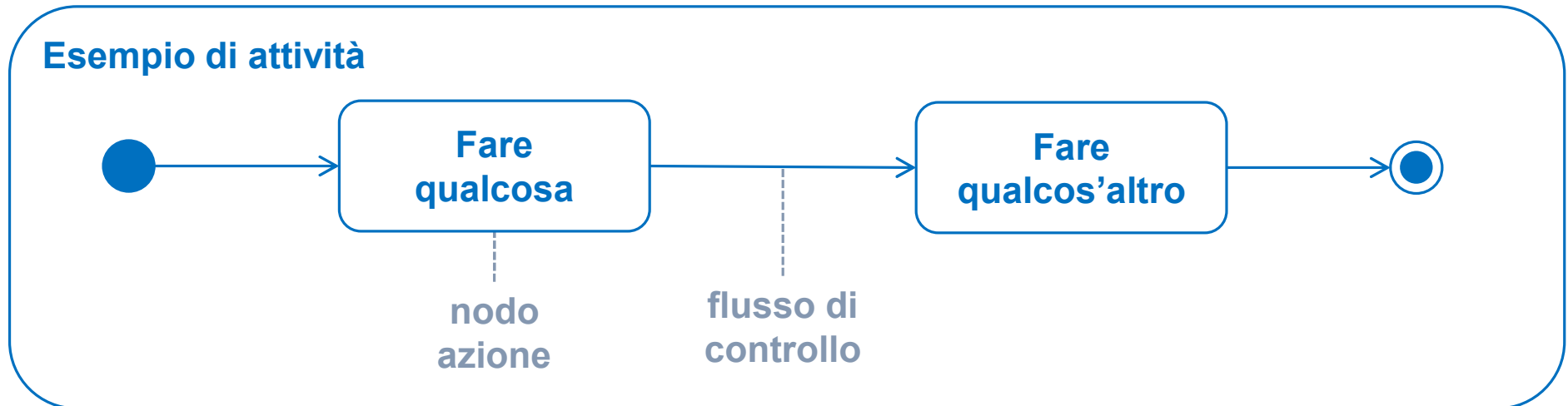
Fare
qualcos'altro

- Sono **atomiche** (e non interrompibili)

CONTROL FLOW

Ogni azione ha

- un solo **arco entrante**
- un solo **arco uscente**



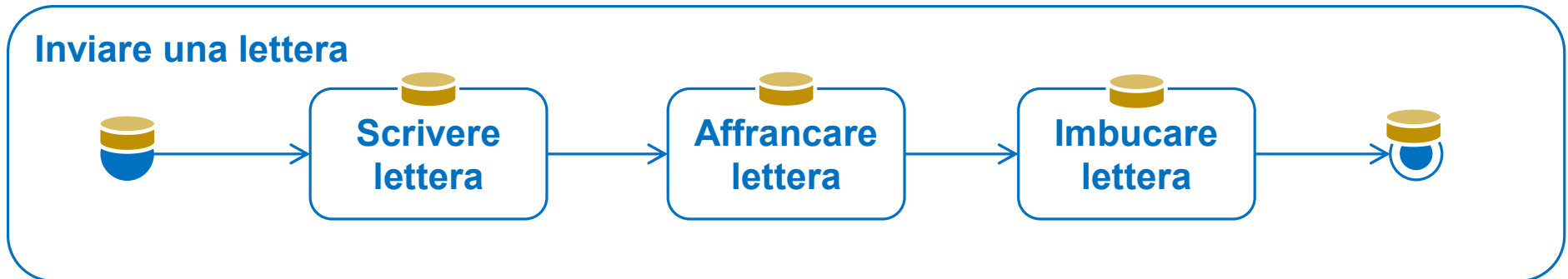
Un arco viene **attraversato** appena **termina l'azione** da cui esce

CONTROL FLOW (CONT.)

Quando un'azione è completa, scatta una **transizione automatica** all'azione che segue

La semantica può essere descritta con un **token game** 

- Un'azione viene eseguita quando riceve il token
- Quando è terminata, il token passa all'azione successiva



NODI DI CONTROLLO



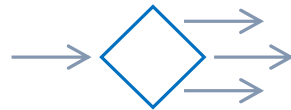
nodo **iniziale**



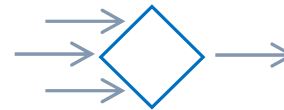
nodo **finale**



nodo di
fine flusso



nodo di **decisione** (con
guardie su archi uscenti)



nodo di
fusione



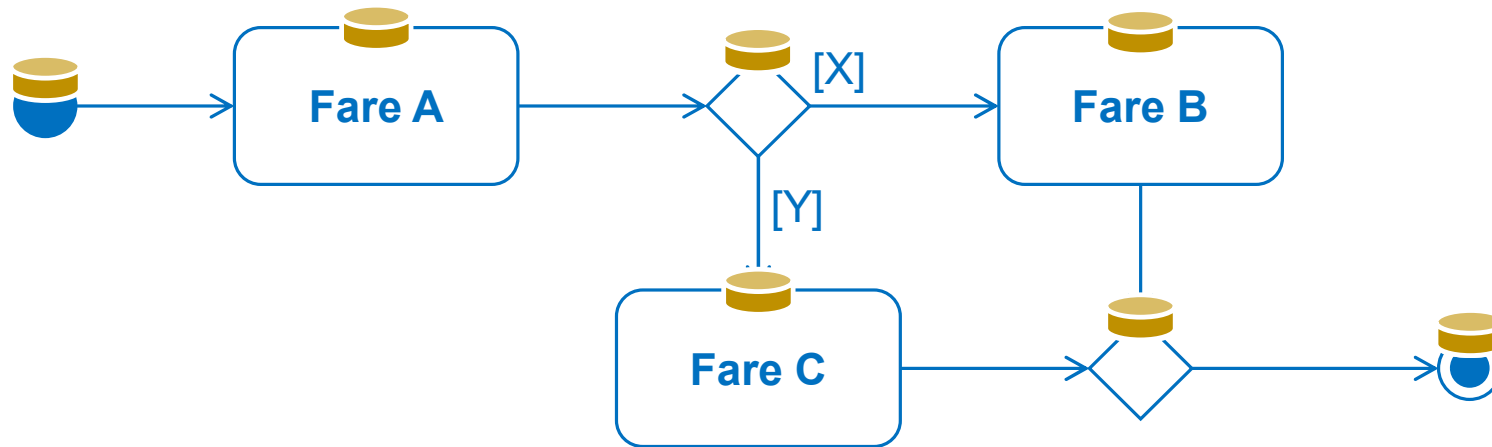
nodo di **fork**
(biforcazione)



nodo di **join**
(sincronizzazione)

Alterano il
control flow!

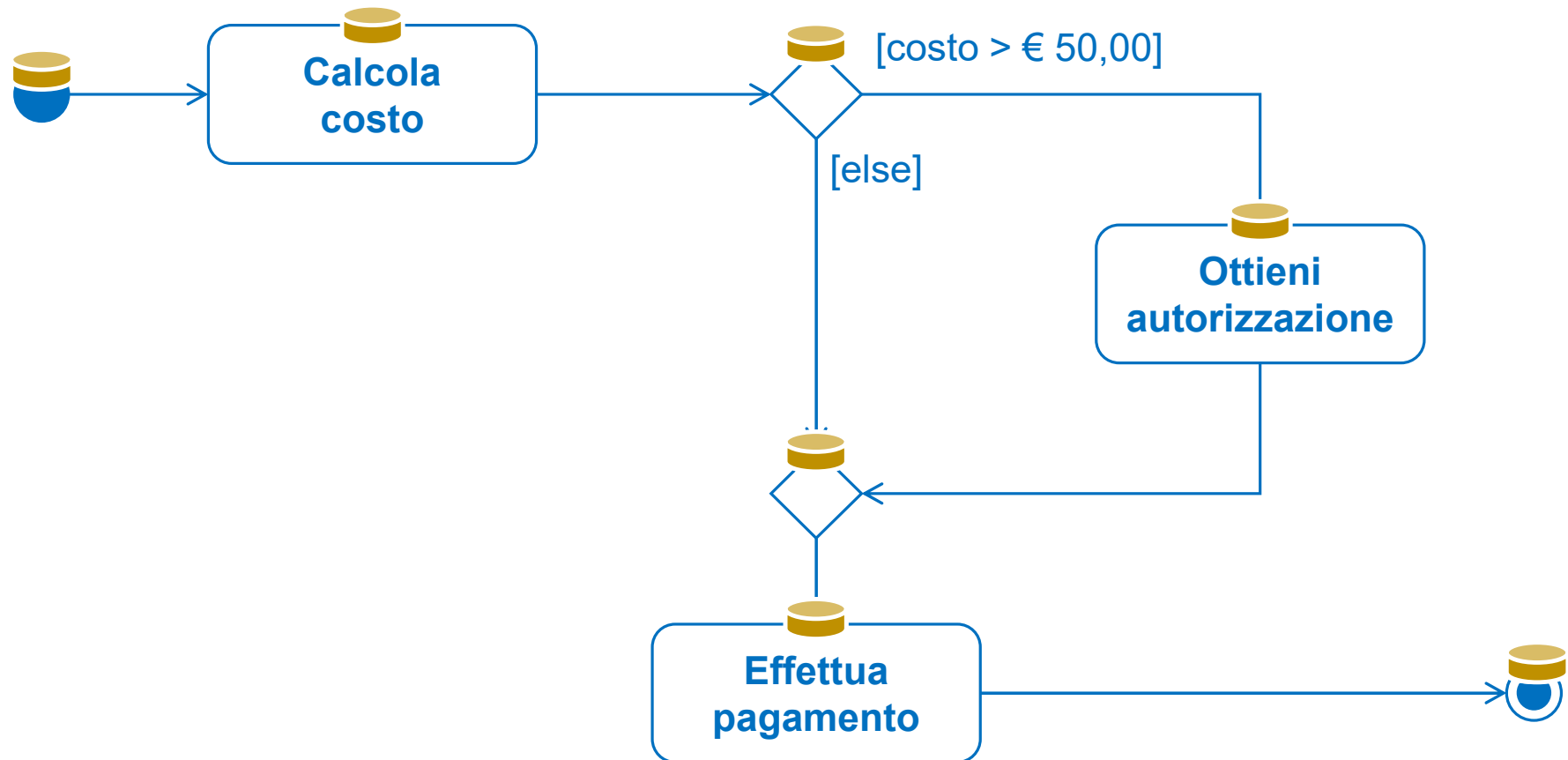
CONTROL FLOW: DECISIONE E FUSIONE



Il token deve prendere **sempre uno dei due cammini**, non può bloccarsi sul nodo di decisione

- Le **condizioni** X e Y devono coprire **tutti i possibili casi** (ovvero $X \vee Y = \text{true}$)
- In **una guardia** si può scrivere **[else]**

UN ALTRO ESEMPIO



CONTROL FLOW: DECISIONE E FUSIONE (CONT.)

Le condizioni di **guardia**

- Si esprimono tra parentesi quadre // [cond] in generale in UML
- Devono coprire **tutte le possibilità** // eventualmente usando [else]
- È bene che siano mutuamente esclusive // altrimenti comportamento non-deterministico

Dato un nodo di decisione

- Non è obbligatorio un nodo di fusione corrispondente
- Potrebbe esserci un nodo di fine flusso

WARNING: IMPRECISIONI NEL LIBRO DI TESTO

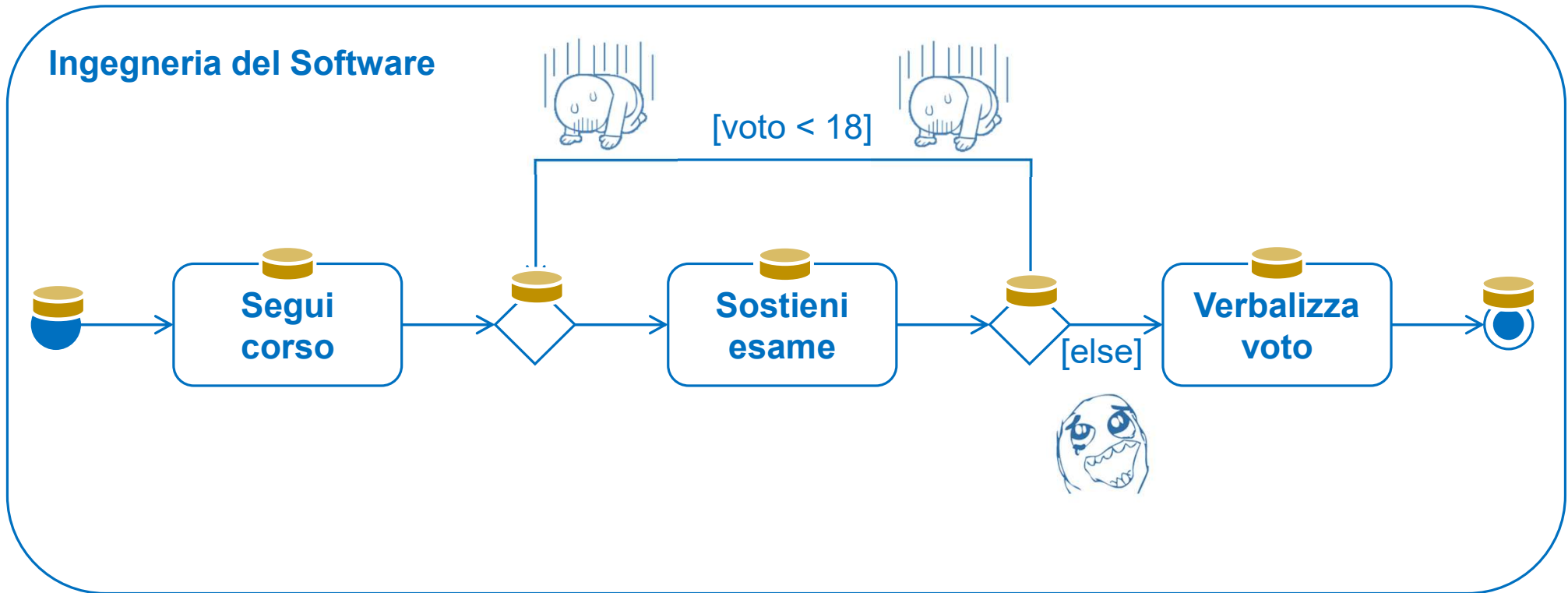
Nel libro di testo, il paragrafo su scelta e guardie ha due imprecisioni

- Afferma che è possibile avere due guardie g_1 e g_2 tali che $g_1 \vee g_2 = \text{false}$ (in generale, $\bigvee_i g_i = \text{false}$), violando la parte evidenziata in rosso dello standard
- Afferma che le guardie devono essere mutuamente esclusive ($g_1 \wedge g_2 = \text{false}$), in contrasto con la parte evidenziata in giallo dello standard

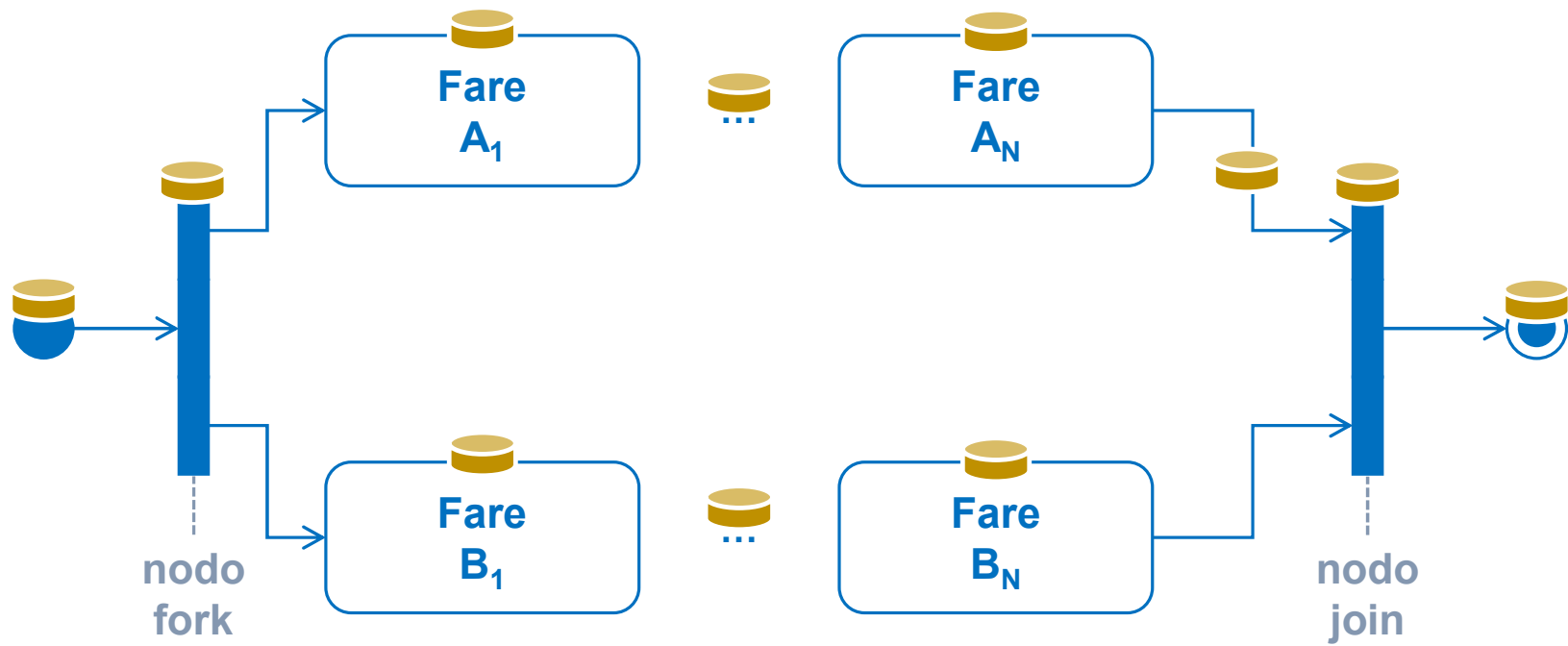
A decision node has one input and two or more outputs. The input value is used to evaluate guard conditions on each of the outputs. If a guard condition evaluates true, the corresponding output is eligible for selection. Exactly one eligible output is chosen to receive a copy of the input value. If more than a guard condition evaluates true, the choice of the output is nondeterministic.

If no guard condition is true, the model is ill formed.

MODELLARE ITERAZIONI



CONTROL FLOW: FORK E JOIN



CONTROL FLOW: FORK E JOIN (CONT.)

FAQ: Come funzionano fork e join nel token game?

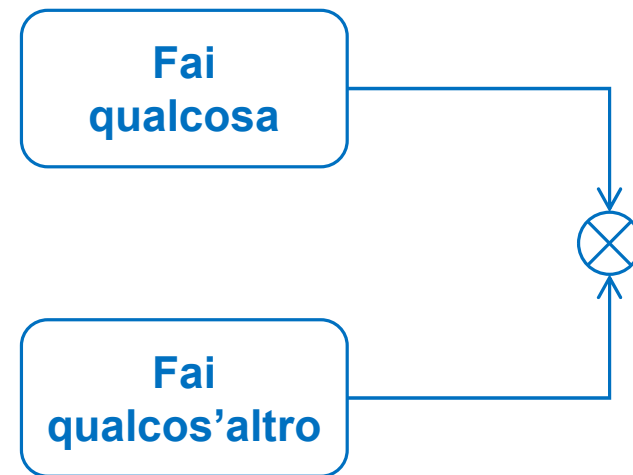
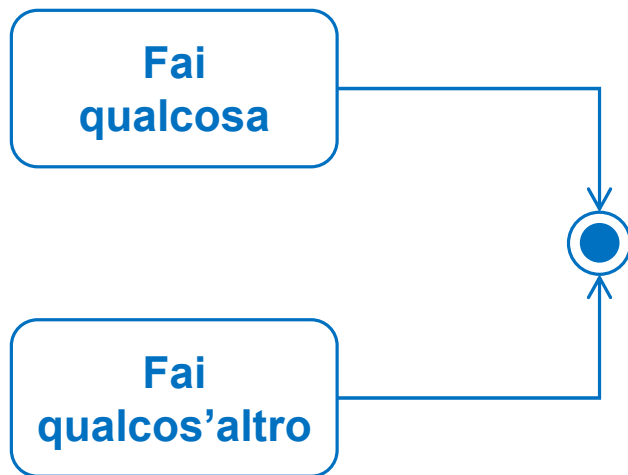
- La fork **moltiplica** i token
⇒ dato un token in ingresso, ne "produce" **uno per ogni freccia uscente**
- La join **consuma** i token
⇒ **attende** un token per ogni freccia entrante
⇒ **consuma tutti** i token e ne esce solo uno

NB: Non è necessaria una join per ogni fork

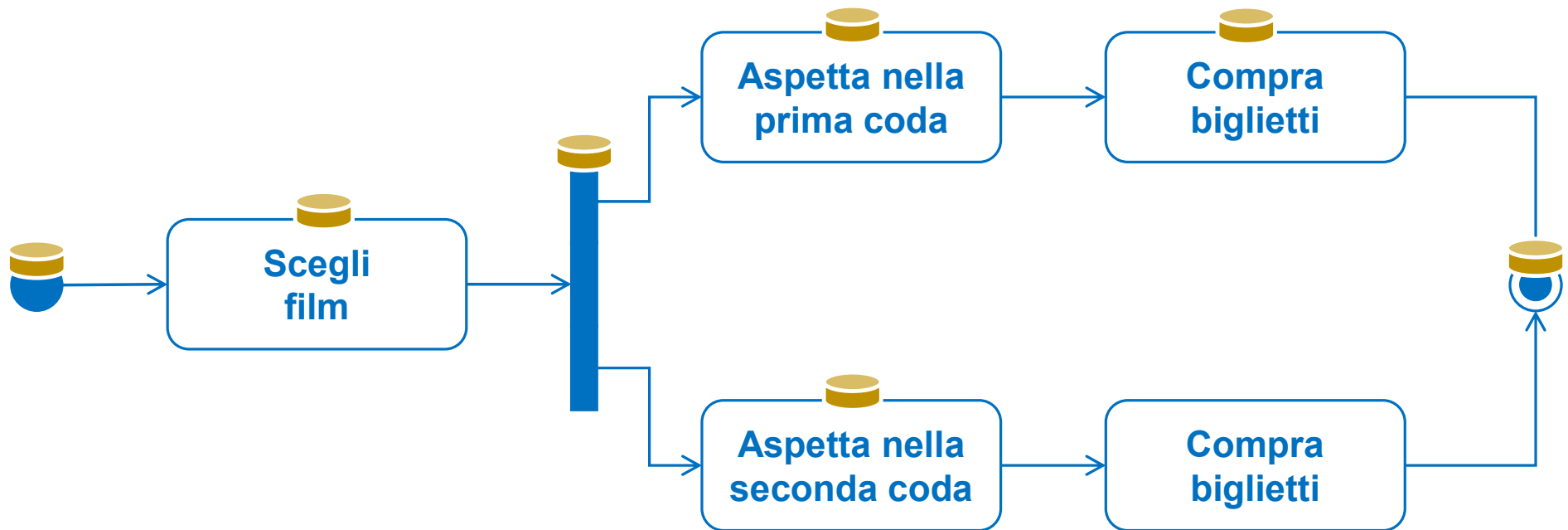
CONTROL FLOW: FINE ATTIVITÀ E FINE FLUSSO

I nodi di fine attività e fine flusso (e solo loro) possono avere **più archi** entranti

- **fine attività**: il primo token che raggiunge il nodo **termina l'intera attività**
- **fine flusso**: ciascun token termina **solo il flusso** corrispondente (non l'intera attività)

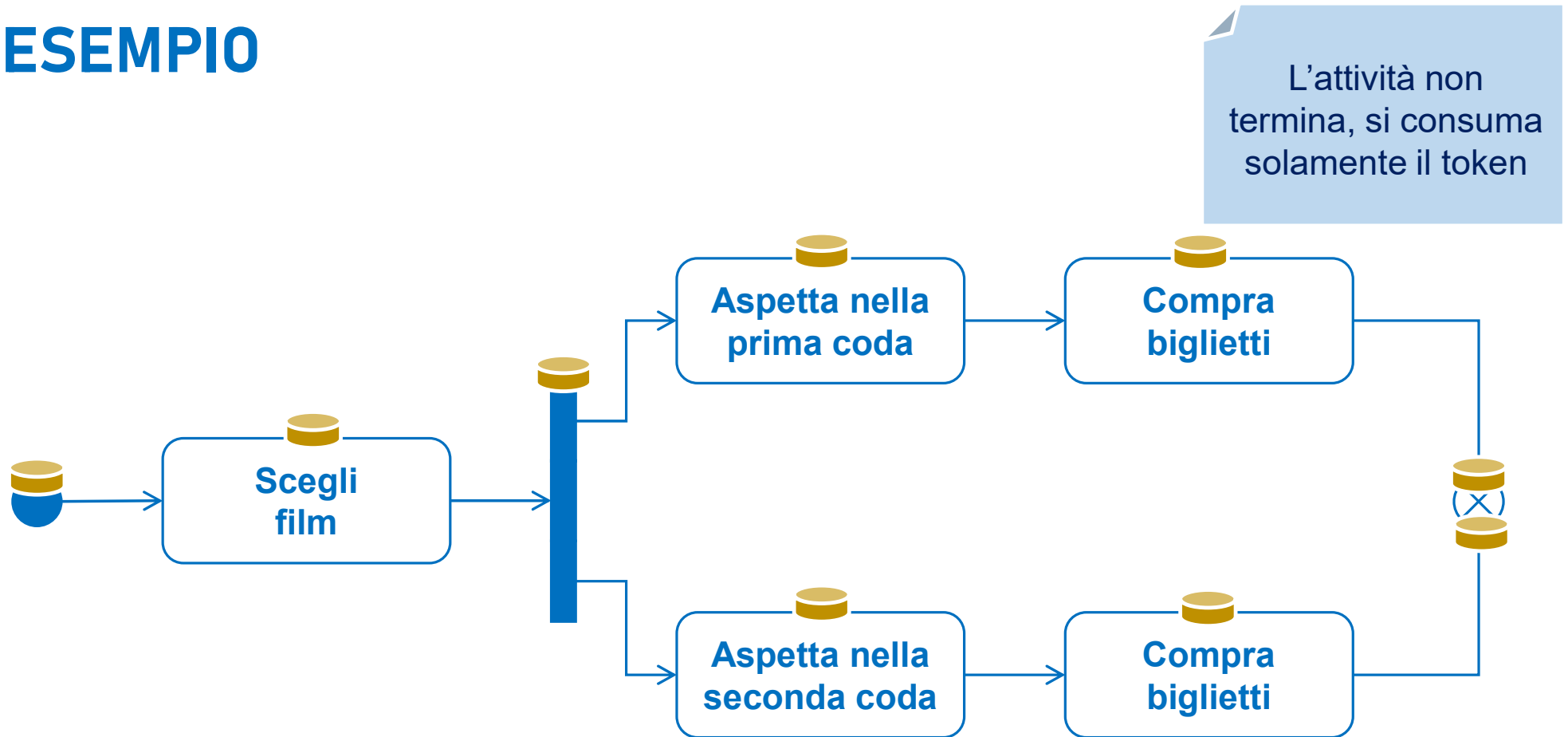


ESEMPIO



Il primo che compra i biglietti termina l'intera attività!

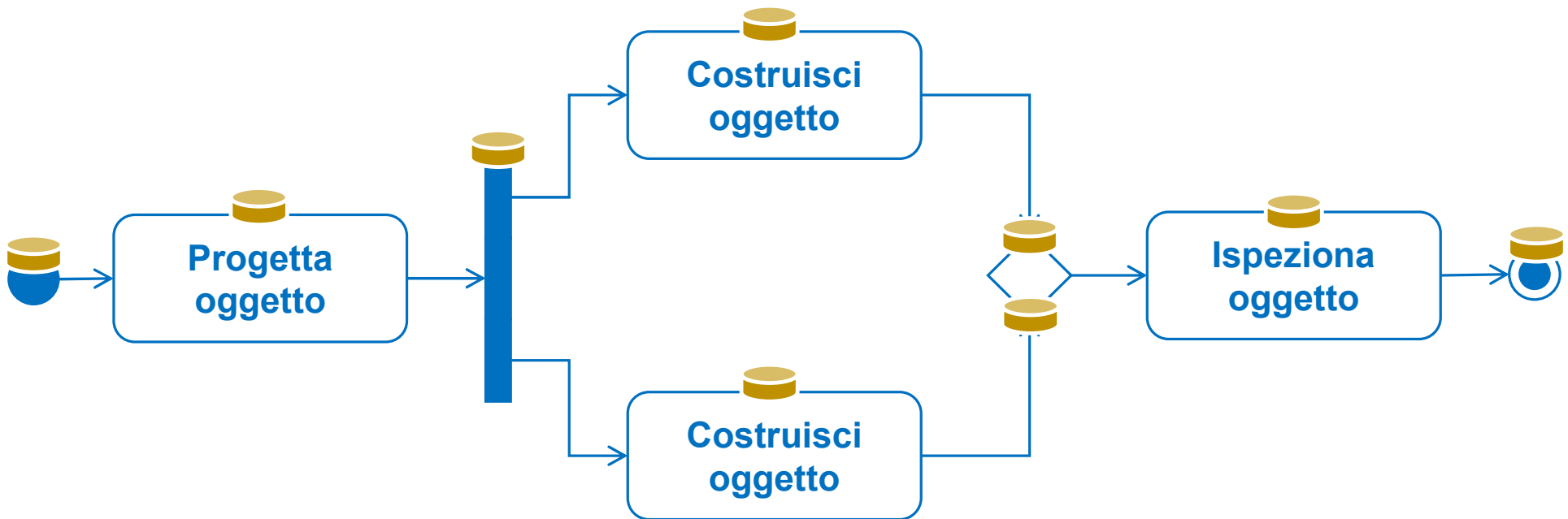
ESEMPIO



Vengono comprati i biglietti in entrambe le code

CONTROL FLOW: FORK E MERGE

Possibile, ma le azioni dopo la fork sono **eseguite più volte**



ESEMPIO, DAL WEB

Interessante, perché sbagliato 😊

Si possono specificare più archi entranti/uscenti in/da un nodo, ma

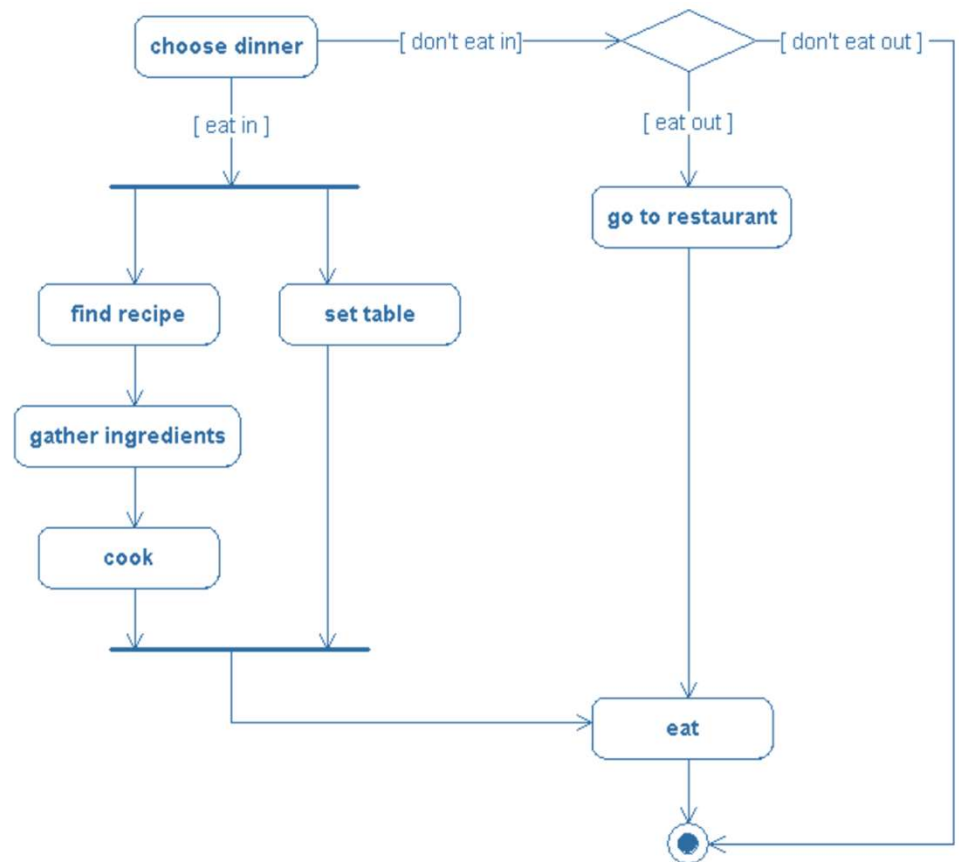
- se ne sconsiglia l'uso
- (ed è vietato in questo corso)

La semantica è quella di fork e join, ma

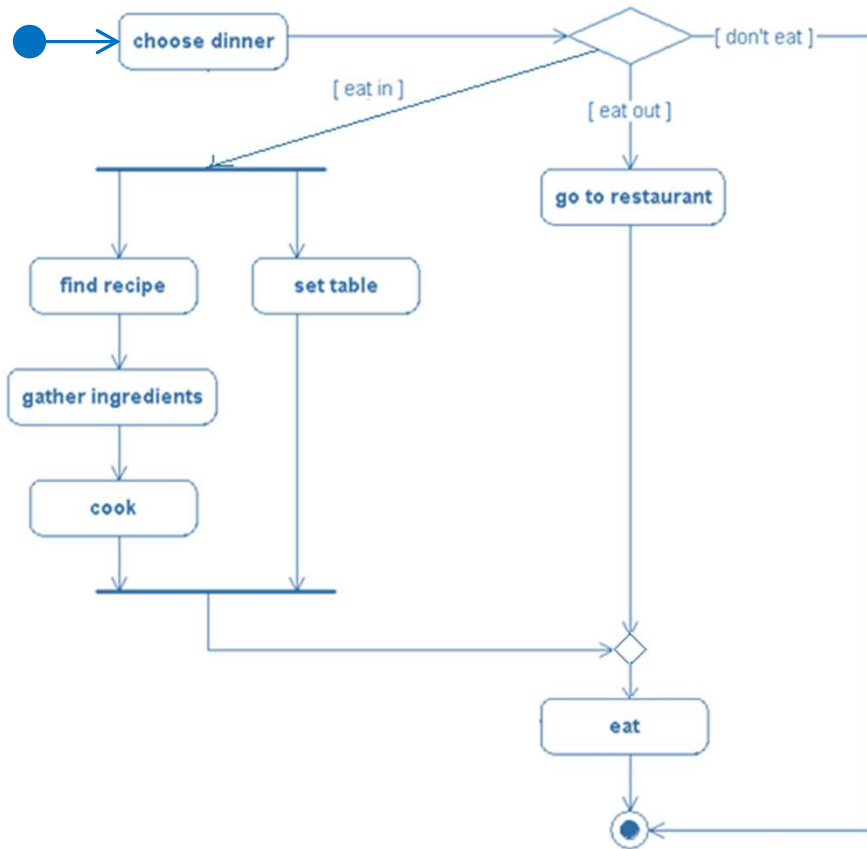
- è facile sbagliarsi e
- disegnare diagrammi che vanno in deadlock

nell'esempio

- eat attende due token
- che non possono mai arrivare



ESEMPIO, DAL WEB (CORRETTO)



Servono

- un nodo decisione prima di choose dinner
- un nodo fusione prima di eat

Tollerate due frecce entranti nello stato finale

SEGNALI ED EVENTI

Nodi specializzati consentono di gestire **invio** e **ricezione** di **segnali**

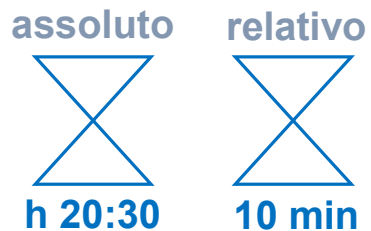
Invio di un **segnale** // è **asincrono** e **non blocca** l'attività



Accettazione di un **evento esterno**



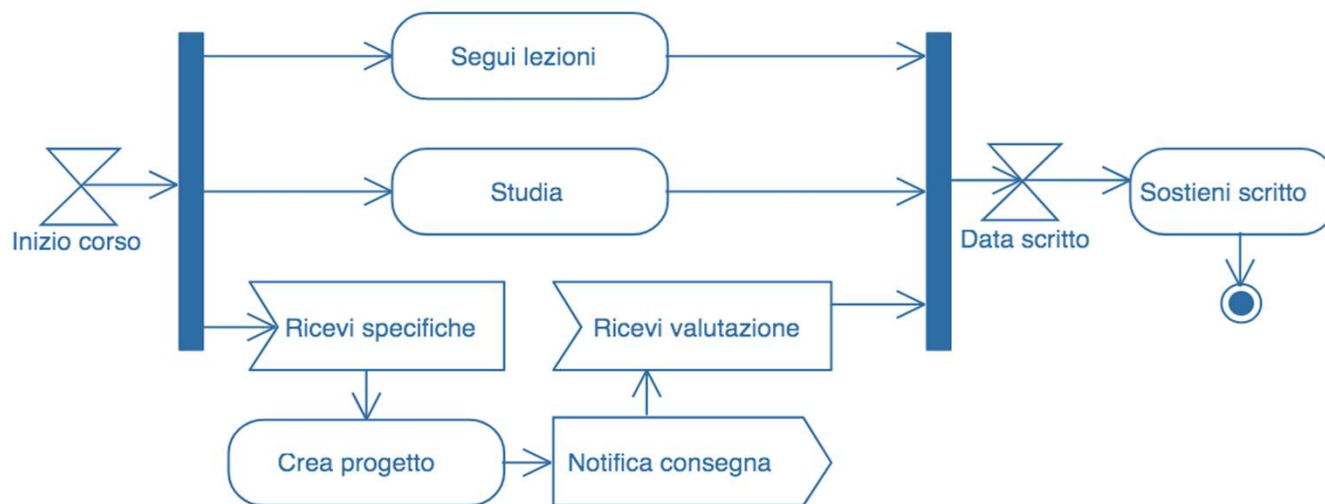
Accettazione di un **evento temporale**



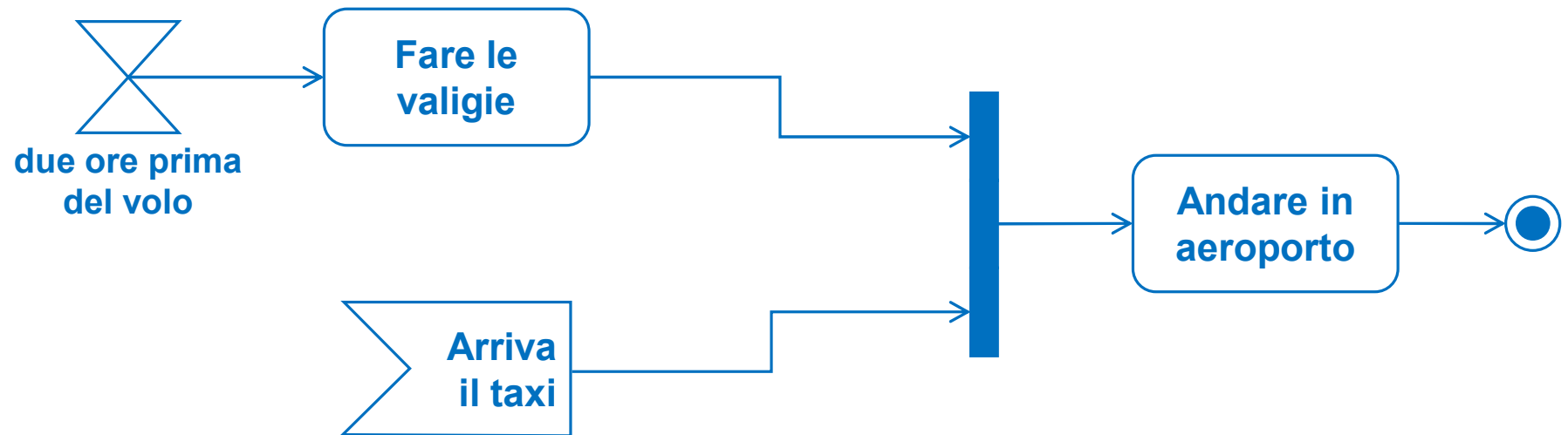
ACCETTAZIONE DI EVENTI

I nodi di accettazione eventi esterni/temporali **non necessitano di archi entranti**

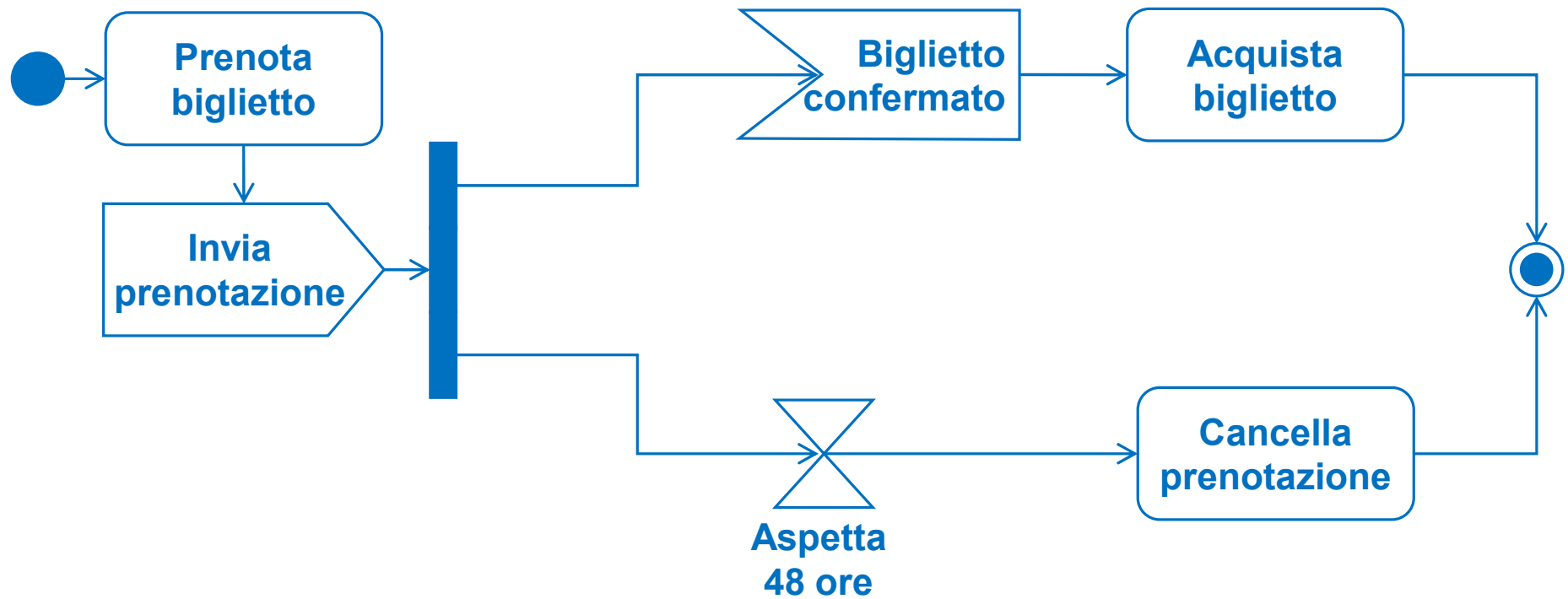
- se assente, quando si verifica l'evento, si **genera** un **token**
- se presente, l'azione è **abilitata** quando **arriva** il **token** e si **attende** l'evento per farlo transitare



ESEMPIO



ESEMPIO DI TIMEOUT



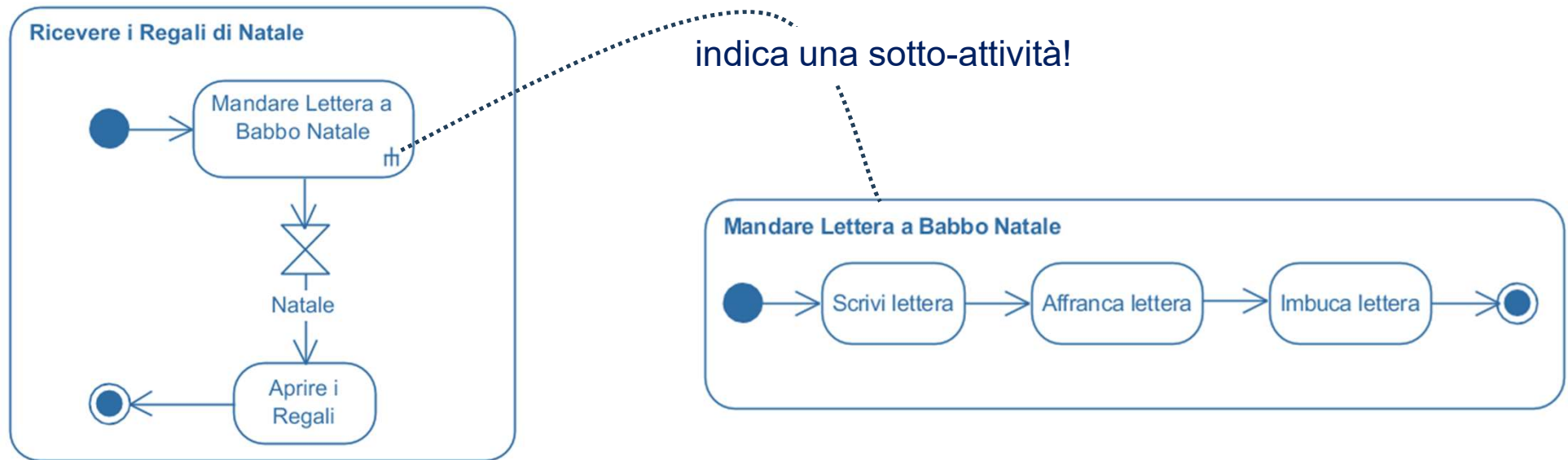
SEGNALI ED EVENTI VS. AZIONI

FAQ: Quando si usa un'azione? Quando si usano invece segnali ed eventi?

- **Azione:** quando effettuata dalla/e **entità** di cui si sta descrivendo il **comportamento**
- **Accettazione di eventi/invio segnali:** quando si comunica con un'**entità esterna**

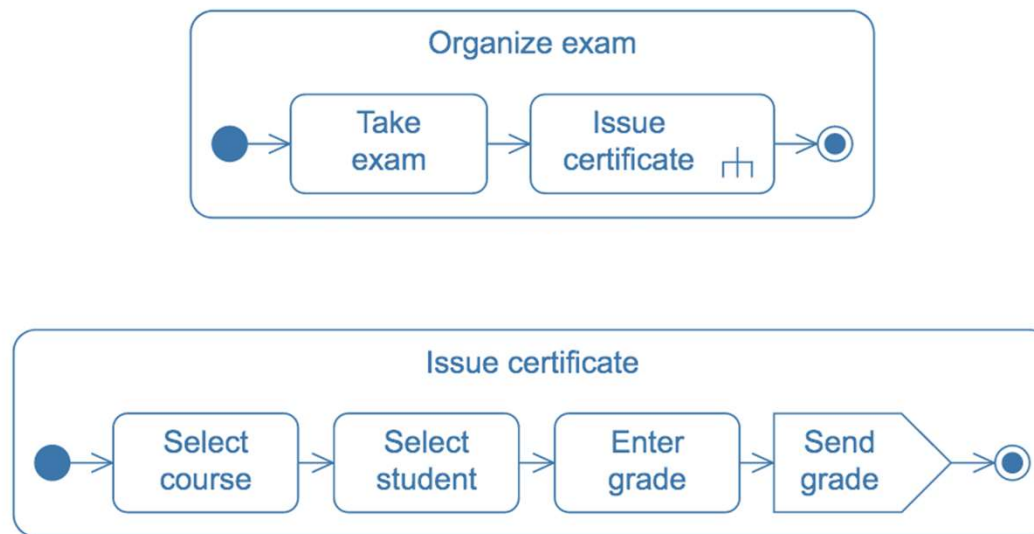
SOTTO-ATTIVITÀ

Un diagramma può contenere un riferimento ad un'attività secondaria (aka. **sotto-attività**)



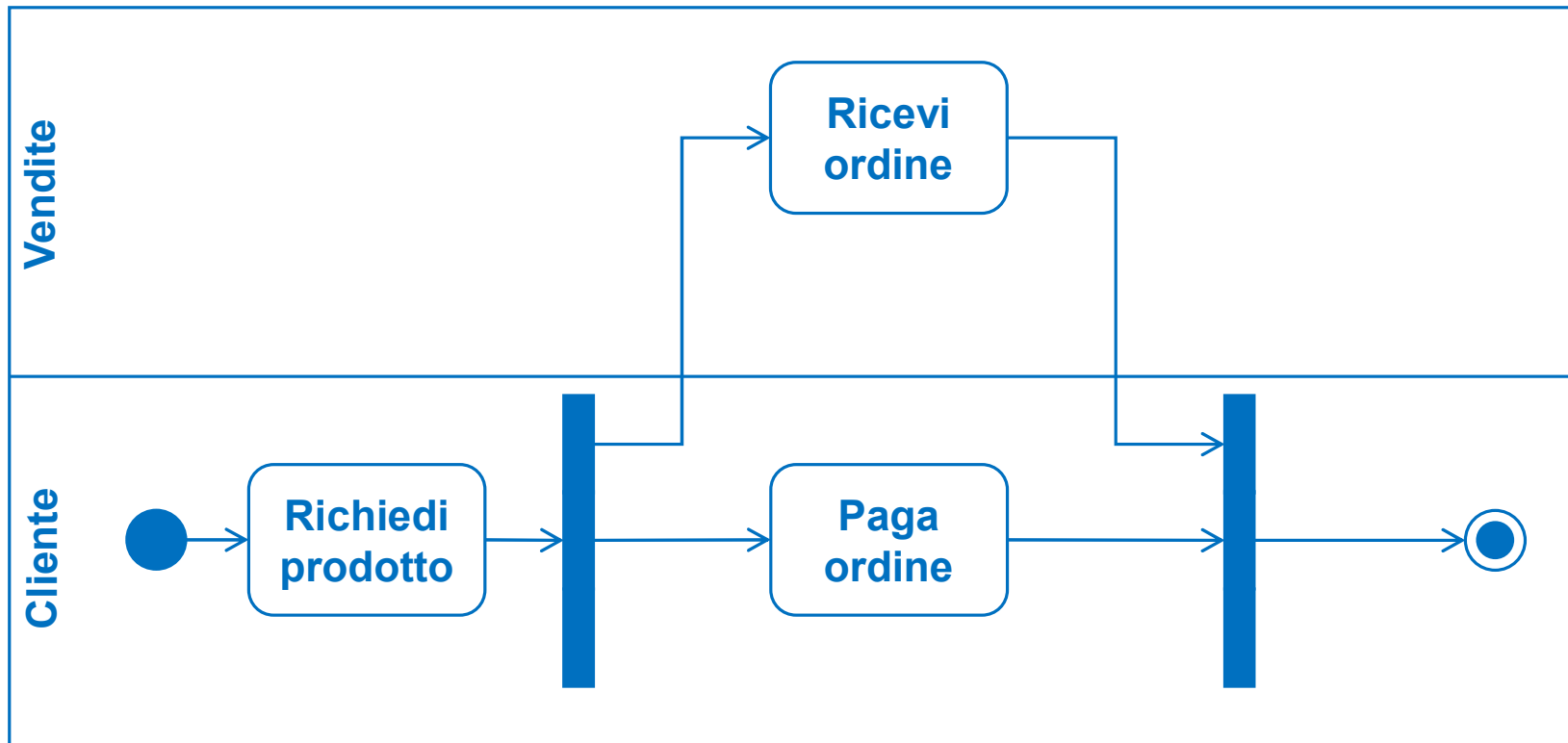
NB: Niente «rastrello» in Visual Paradigm, dove il nome viene messo in grassetto

ESEMPIO



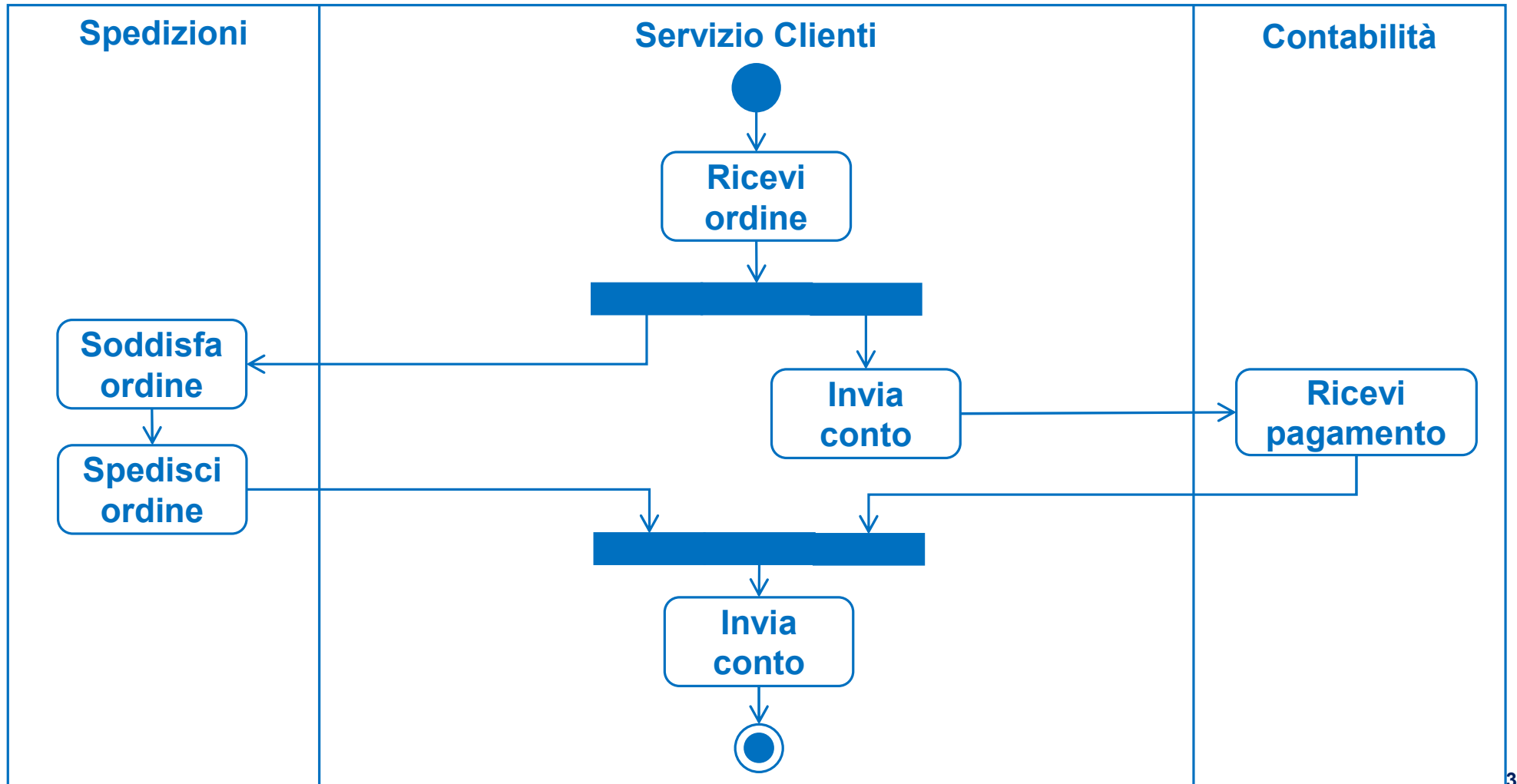
PARTIZIONI

Permettono di assegnare la responsabilità delle azioni



NB: Spesso corrispondono alla divisione in unità operative di un modello di business

ESEMPIO





HOMEWORK

Caso di riferimento: La Piscina

Descrivere con un diagramma di attività il processo che comprende:

- prenotazione vasca nuoto libero
- accesso alla piscina
- ... fino all'uscita

Trattare anche i casi di errore (ad esempio, arrivo fuori orario)

RIFERIMENTI

Contenuti

- **Sezione 7.1** di "UML@Classroom" (M. Seidl et al., 2015)
// escludendo parametri, pre- e post-condizioni
- **Sezione 7.2** di "UML@Classroom" (M. Seidl et al., 2015)
// escludendo *object flow wedge*
- **Sezione 7.3** di "UML@Classroom" (M. Seidl et al., 2015)
// escludendo guardie, *weight edge*, connettori, *decision behaviour* e diversa semantica scelte
- **Sezione 7.5** di "UML@Classroom" (M. Seidl et al., 2015)
- **Sezione 7.7** di "UML@Classroom" (M. Seidl et al., 2015)

Approfondimenti

- **Capitolo 14** di "Software Engineering" (G. C. Kung, 2023)