

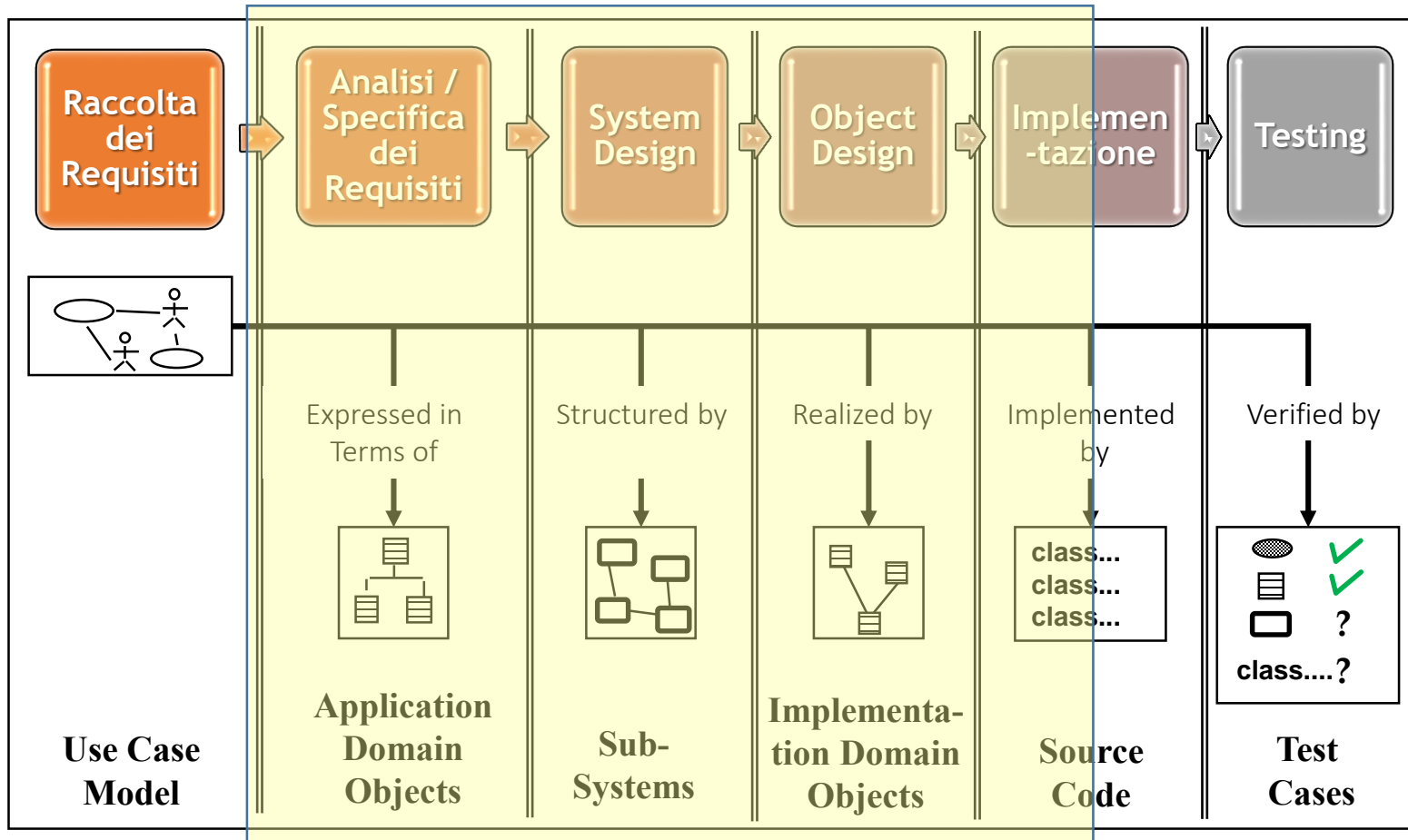


UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Ingegneria del Software – Gli State Charts

Prof. Sergio Di Martino

Ciclo di Vita del Software



Diagrammi di stato

- Un Diagramma di Stato è un formalismo grafico largamente utilizzato in computer science per rappresentare il comportamento di un artefatto (oggetto, modulo, intero sistema, etc...) composto da un numero finito di stati.
- Ne esistono svariate versioni
 - deterministic finite state machine (DFA),
 - nondeterministic finite state machine (NFA),
 - Moore machine
 - Harel's StateCharts
 - ...

Stati e transizioni

- Paradigma ben noto e molto comprensibile per descrivere il comportamento di entità dinamiche
 - stati rilevanti dell'entità
 - transizioni = possibili passaggi di stati, magari con annotazioni riguardo a cosa ha causato la transizione, o che cosa viene rilevato sulla transizione
- Vantaggio principale: grande impatto visuale, molto leggibile, per modellare aspetti dinamici e transienti



Statechart

- Notazione visuale e formale, sviluppata da D. Harel, fine anni 80, per descrivere il comportamento di sistemi reattivi
 - Le transizioni descrivono come l'entità modellata "reagisce" a eventi (generati dall'esterno o da se stesso)
 - Le transizioni sono (in genere) lanciate (triggered) dagli eventi
- Statechart diagram di UML: adattamento delle statechart ad un mondo O-O
- Usati principalmente per descrivere il comportamento di:
 - Use Cases
 - Classi

Statechart diagram

- Da UML specification

“A statechart diagram can be used to describe the behavior of a model element such as an object or an interaction. Specifically, it describes possible sequences of states and actions through which the element can proceed during its lifetime as a result of reacting to discrete events (e.g., signals, operation invocations).”

“Statechart diagrams represent the behavior of entities capable of dynamic behavior by specifying its response to the receipt of event instances. Typically, it is used for describing the behavior of classes, but statecharts may also describe the behavior of other model entities such as use-cases, actors, subsystems, operations, or methods.”

A cosa servono gli statechart diagram

- Ogni oggetto in un sistema ha un certo ciclo di vita. Tra la sua creazione e la sua distruzione esso interagisce con gli oggetti vicini.
- L'oggetto quindi dovrà rispondere a stimoli **asincroni** che gli arrivano dal suo ambiente
 - Es.: un cellulare deve essere sempre pronto a rispondere a chiamate in entrata
- Esistono oggetti il cui comportamento dipende dallo stato dei suoi attributi
 - Es.: in un cellulare, all'arrivo di una chiamata l'attivarsi o meno della suoneria dipende dalla modalità (normale o "silenziosa") in cui si trova l'apparecchio.

A cosa servono gli statechart diagram?

- Il ciclo di vita di questi artefatti (che spesso sono classi, ma possono essere anche interi sottosistemi) viene modellato attraverso Statecharts
- Uno Statechart di design è quindi un diagramma chiave per guidare i programmatori nell'implementazione del sistema
 - Tutto ciò che c'è nello statechart di design è un comportamento da implementare

Il Formalismo

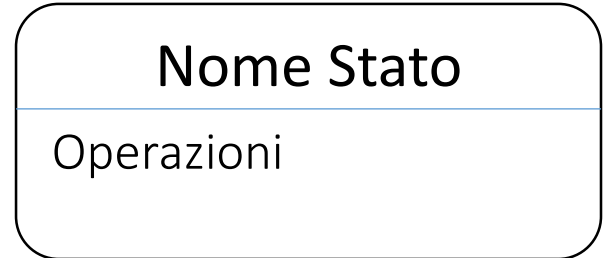
Eventi, Stati e Transizioni

- Un **Evento** è un avvenimento significativo per un artefatto
- Uno **Stato** è una condizione di un oggetto in un intervallo di tempo compreso tra due eventi
 - Uno **stato** è una situazione nella vita dell'oggetto in questione in cui esso soddisfa una qualche condizione, esegue una qualche attività o è in attesa di un qualche evento.
- Una **Transizione** è una relazione che lega uno stato di partenza ed uno stato di arrivo (non necessariamente distinti).
 - Le transizioni che riportano nello stesso stato sono dette *self-transition*

Rappresentazioni in UML

- Stato:

- Un rettangolo arrotondato
- Un nome
- Un elenco opzionale di operazioni da svolgere al suo interno



- Ogni transizione è etichettata con tre elementi, tutti opzionali

Evento [Condizione] / Azione

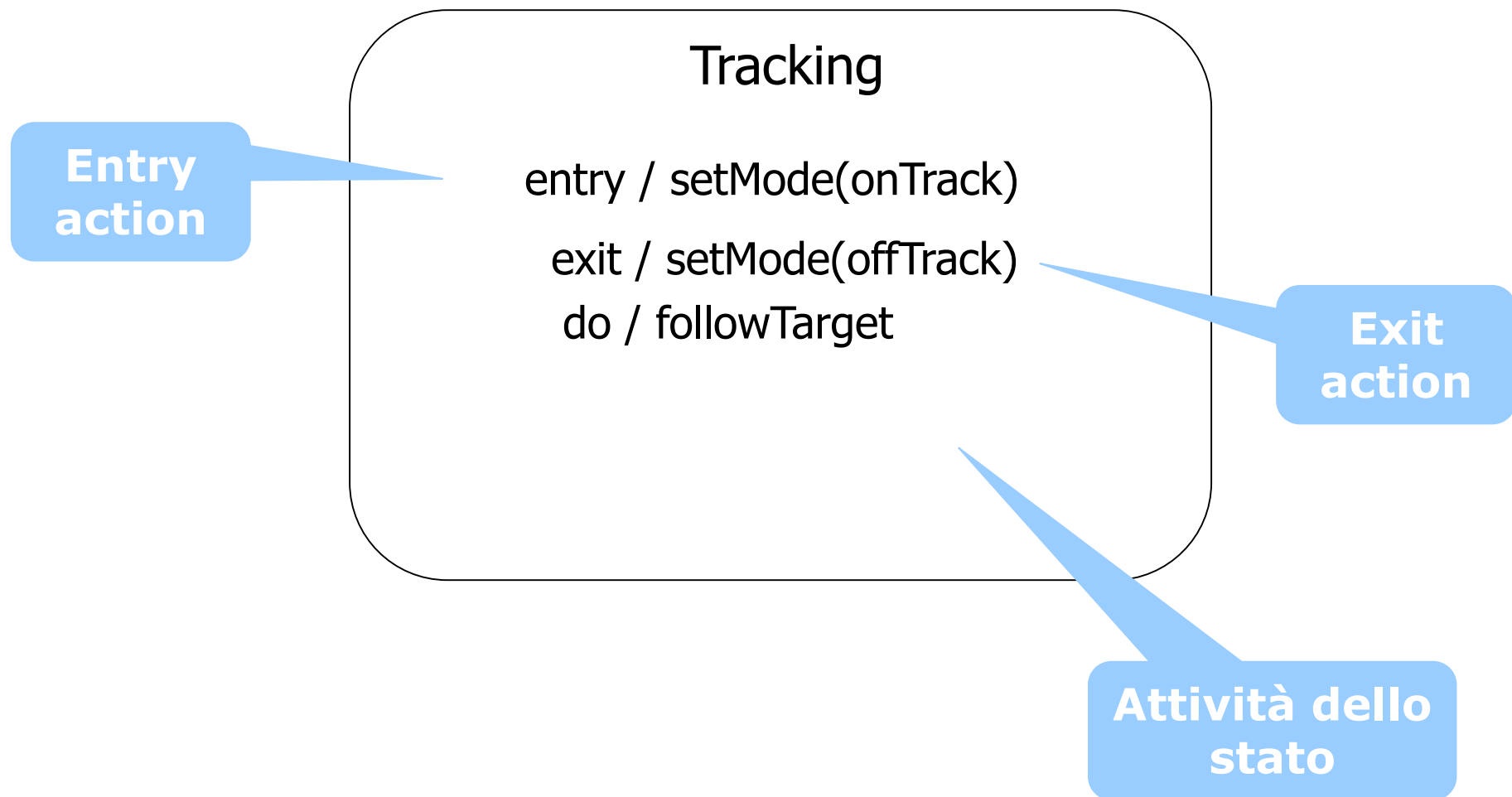
Dettagli sulle Transizioni

- **Evento** (trigger): è l'occorrenza di uno stimolo che può innescare una transizione fra stati
 - Es: NextTrackButtonPressed()
- **Condizione** (guard): espressione booleana.
 - Se da uno stato escono più transizioni, le loro condizioni devono essere mutuamente esclusive
 - Può essere espressa usando OCL, linguaggi di programmazione, pseudocodice....
 - Es: [This.CurrentTrack < This.TotalTracks]
- **Azione**: è associata alla transizione, è considerata un processo rapido, non interrompibile da un evento (atomica)
 - Es: This.CurrentTrack++



Stato: notazione

- Uno stato può opzionalmente avere le seguenti azioni:
 - **Entry**: azione eseguita ogni volta che si entra nello stato (cioè in risposta all'evento entry)
 - **Exit**: azione eseguita ogni volta che si lascia lo stato (cioè in risposta all'evento exit)
 - **Do**: è associata allo stato, può prevedere un lasso di tempo considerevole, e può essere interrotta da un evento (non-atomica).

Stato (es.)

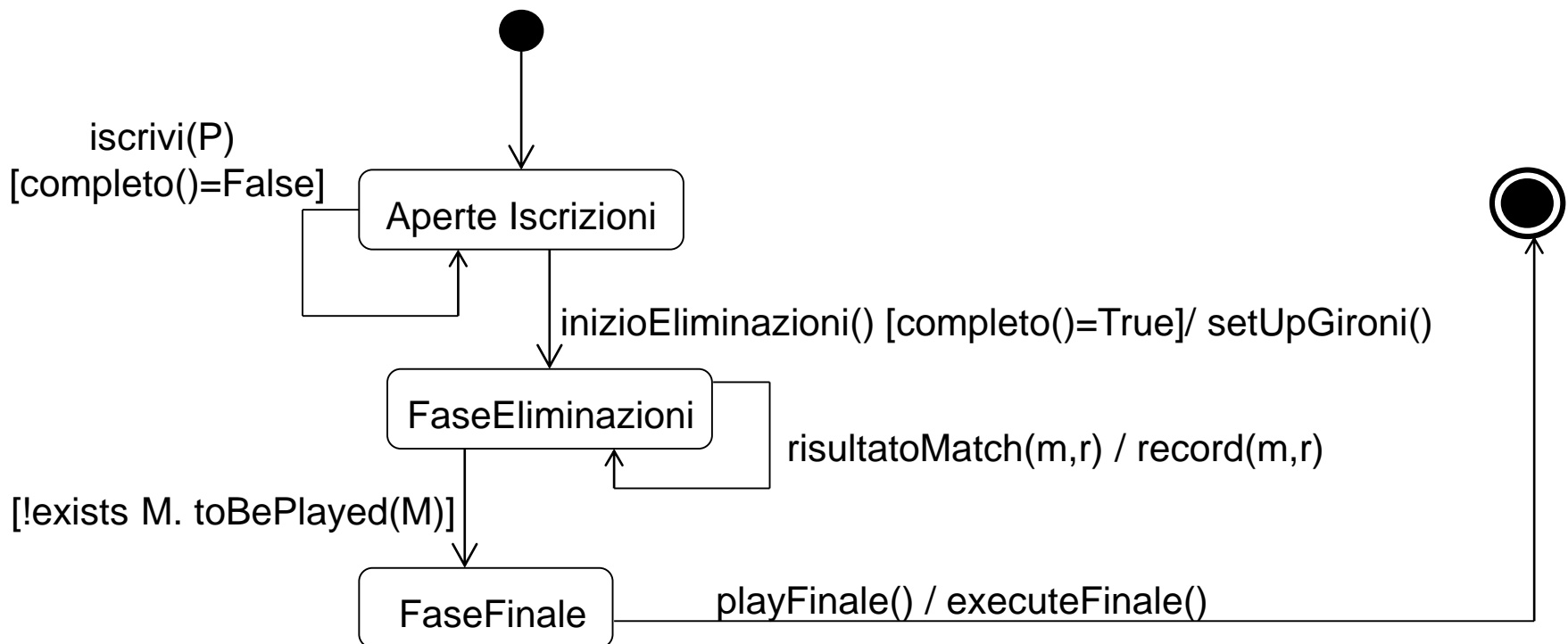


Stato iniziale e stato finale

- Sono due stati particolari: il primo è il punto di partenza del diagramma ed il secondo è il punto di arrivo.
- Sono indicati rispettivamente con e  
- Negli statechart diagram che modellano il comportamento di sistemi destinati a girare di continuo (per esempio sistemi embedded) lo stato finale può non essere presente

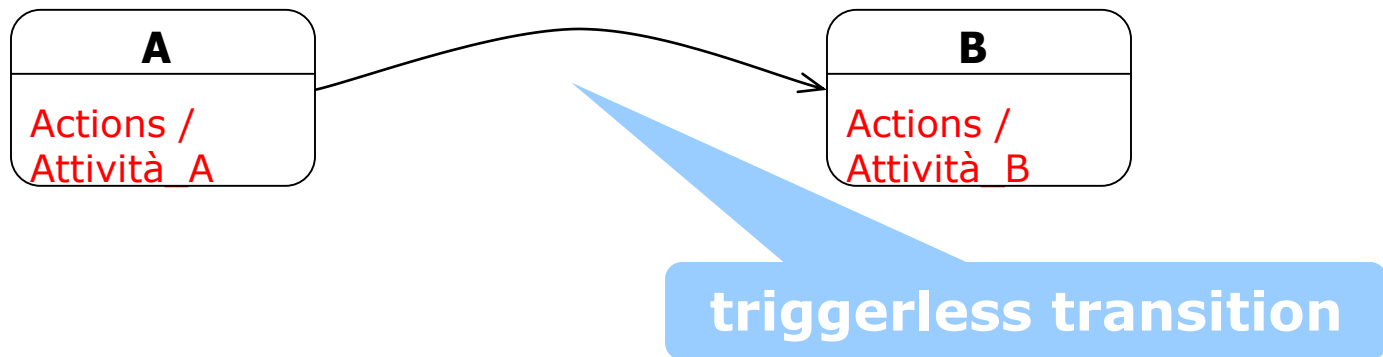
Esempio: behaviour dei tornei

- Es.1) Definire la classe Torneo, ed eventuali altre, in modo che gli eventi, le espressioni e le azioni che appaiono nella statechart sopra siano ben definite.



Transizioni triggerless

- Una transizione che non abbia nessun evento ad innescarla è detta triggerless



- Al termine delle Actions di A si passa automaticamente nello stato B
- E' il termine dell'attività ad innescare la transizione fra stati

Definizione di sottostati

Sottostati

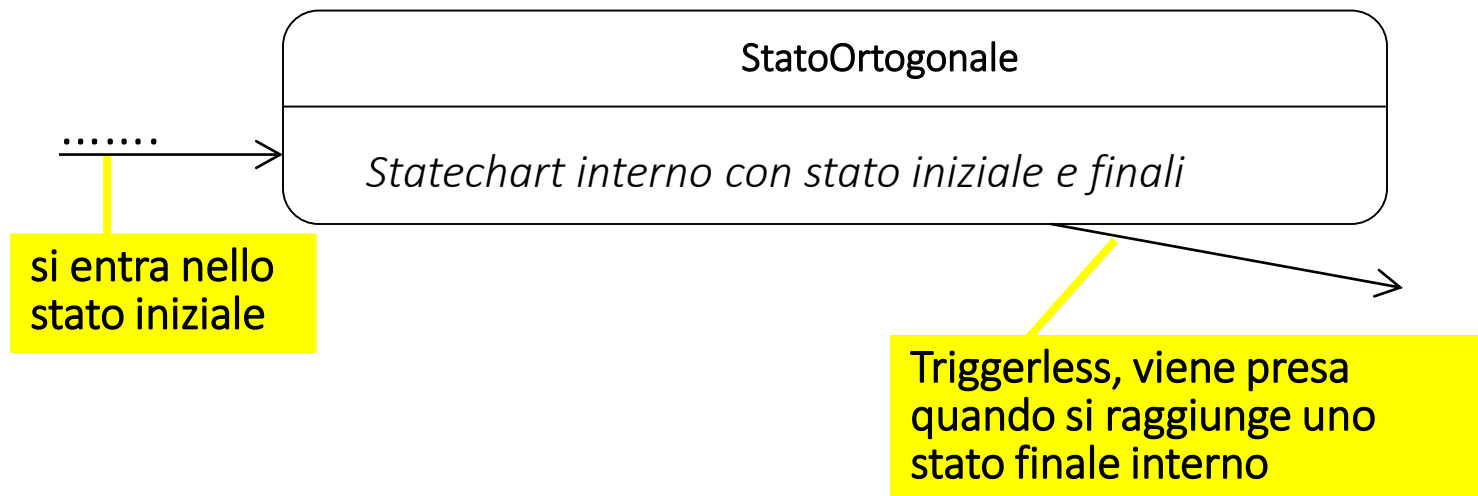
- Il comportamento di uno stato può essere modellato anch'esso con una macchina a stati e questo a qualsiasi livello di profondità
 - Può migliorare molto la leggibilità del diagramma, ma...
 - Troppi livelli penalizzano la leggibilità del diagramma!
- Uno stato composito (cioè che contiene sottostati) può possedere entry, do ed exit action, con lo stesso significato che essi hanno negli stati semplici
 - Una transizione che parte da un superstato rappresenta una transizione da ogni sottostato

Stati composti

- Uno stato può essere decomposto
 - ortogonalmente (in sottostati mutuamente esclusivi)
 - concorrentemente (in sottostati concorrenti)

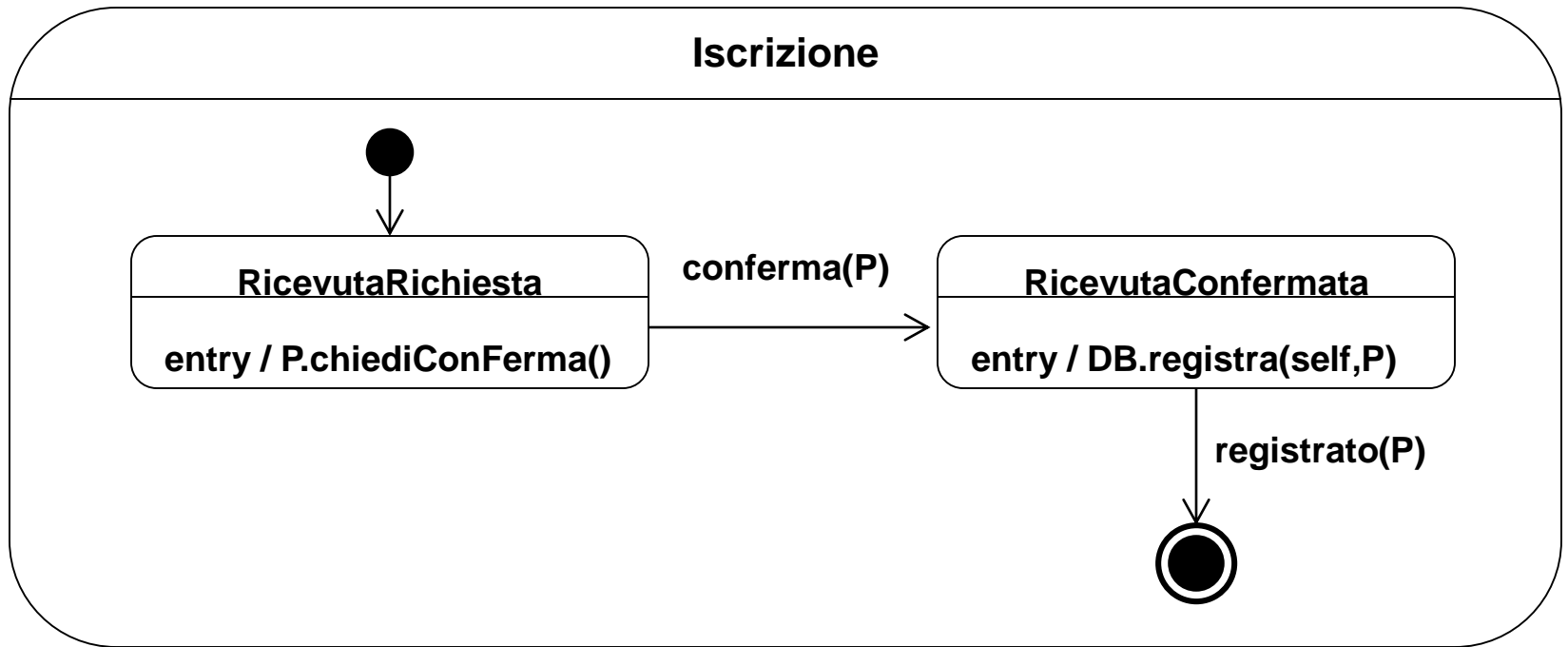
SottoStati Ortogonali

- uno stato può essere decomposto (strutturato) dettagliando cosa fa l'entità modellata quando è in quello stato
- la decomposizione di uno stato si può riportare a parte [migliora la leggibilità]



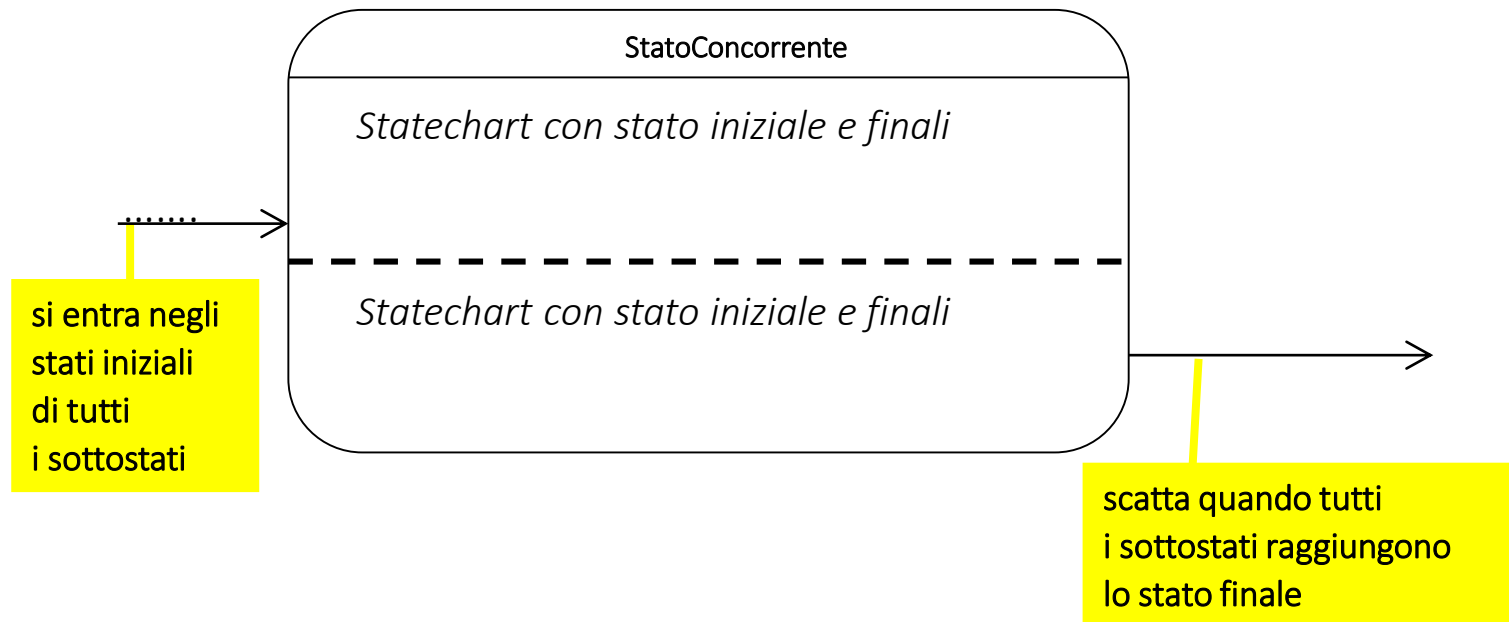
- sono ammesse anche transizioni che entrano direttamente in uno stato interno

Esempio: stato ortogonale

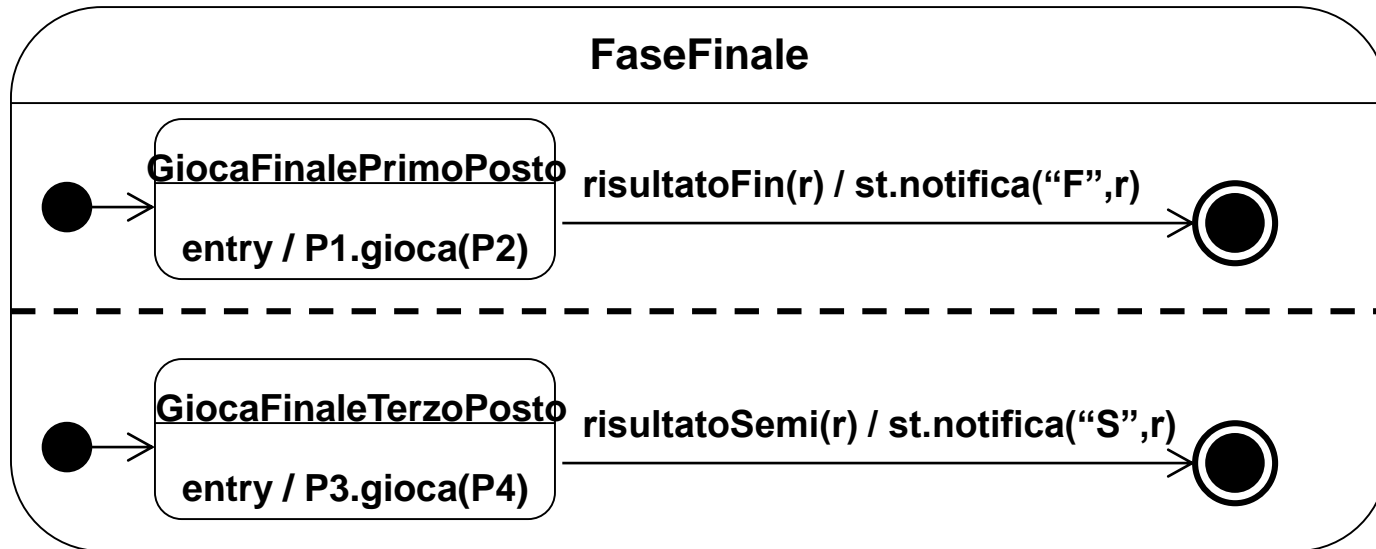


SottoStati Concorrenti

- uno stato può essere decomposto concorrentemente (in sottostati paralleli)
 - OK in fase di analisi (può semplificare le cose)
 - In fase di design richiede molta attenzione (sistema multiprocessore?)



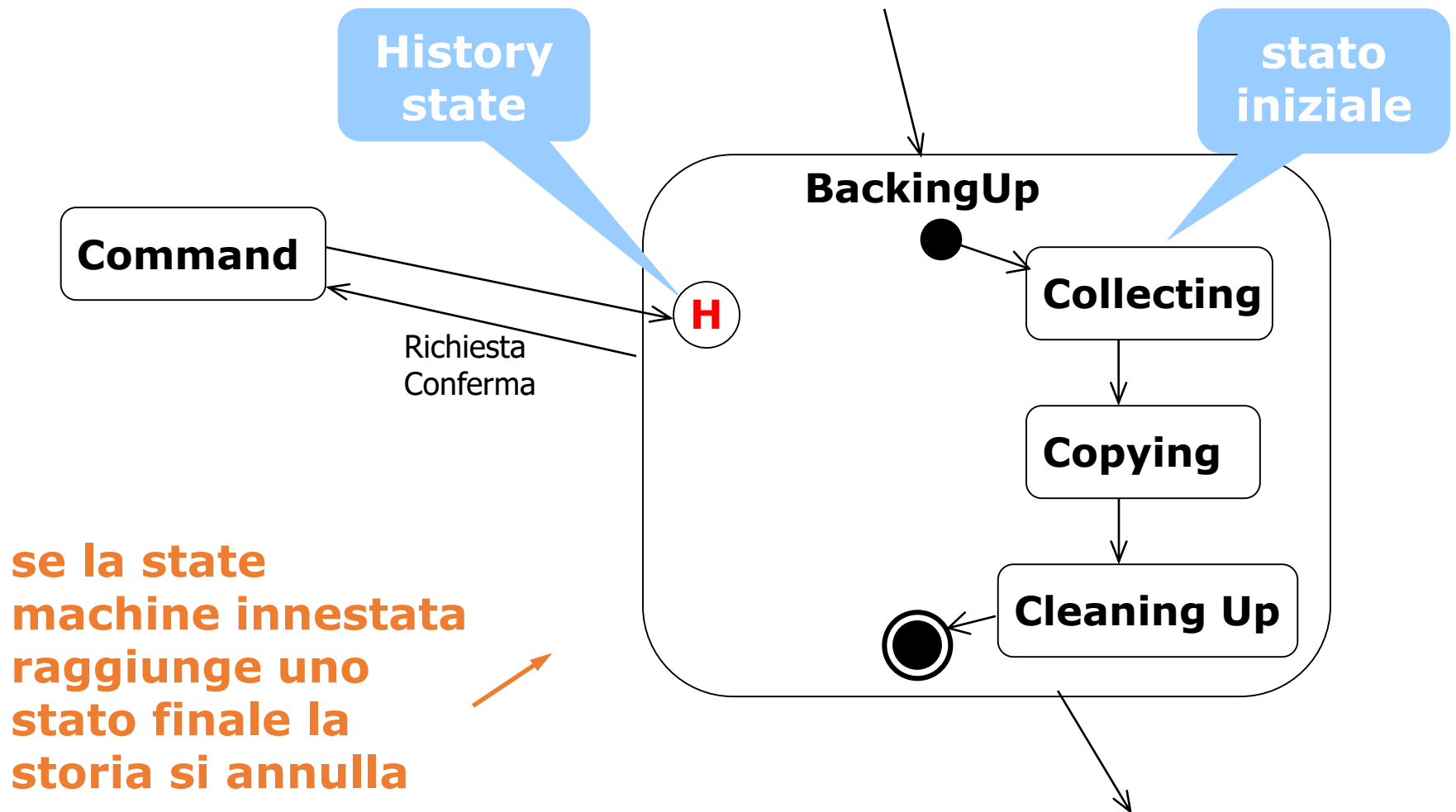
Esempio: stato concorrente



History states

- In generale quando, per effetto di una transizione, si entra in uno stato composito, la state machine innestata comincia dal suo stato iniziale (a meno che la transizione non specifichi come target uno specifico sottostato)
- Esistono però situazioni in cui vorremmo poter memorizzare quale sottostato era attivo nel momento in cui si abbandona uno stato composito
- Lo scopo è quello di riprendere da quel punto le attività una volta che si rientri nello stato composito

History states (es.)



H e H*

- Lo stato H è detto history state superficiale. Infatti in caso di state machine innestate a più di un livello ricorda soltanto lo stato del primo livello (cioè quello in cui si trova) mentre gli altri livelli ripartono dallo stato iniziale.
- Se si vuole ricordare lo stato di ogni livello, esiste uno history state profondo, cioè H*.
 - Con l'uso di H* ci si assicura che tutte le state machine ad ogni livello sottostante a quello con H* ricordino lo stato in cui si trovavano all'abbandono dello stato composito complessivo
- Ovviamente in caso di macchine innestate ad un solo livello H e H* sono equivalenti

Linee guida per sviluppo di statecharts (definizione stati)

- Il nome dello stato deve riflettere un intervallo di tempo reale
- Dare nomi significativi agli stati
 - Se questo non è possibile significa che c'è qualche problema nel progetto
 - Pensare alle successive necessità di manutenzione!
- Il nome dello stato deve essere unico
- Deve essere possibile uscire da ogni stato
- Attivazione dei sottostati in base al tipo di scomposizione

Linee guida per sviluppo di statecharts (azioni ed eventi)

- Non confondere evento (causa) con azione (effetto)
- Nomi di eventi e azioni non ambigui
- Condizioni sugli eventi hanno valore booleano
- Azioni e condizioni sono opzionali: da usare solo se necessario
- Check finale: abbiamo descritto TUTTI i comportamenti possibili del sistema?