UNIVERSITÀ DEGLI STUDI DI TRIESTE

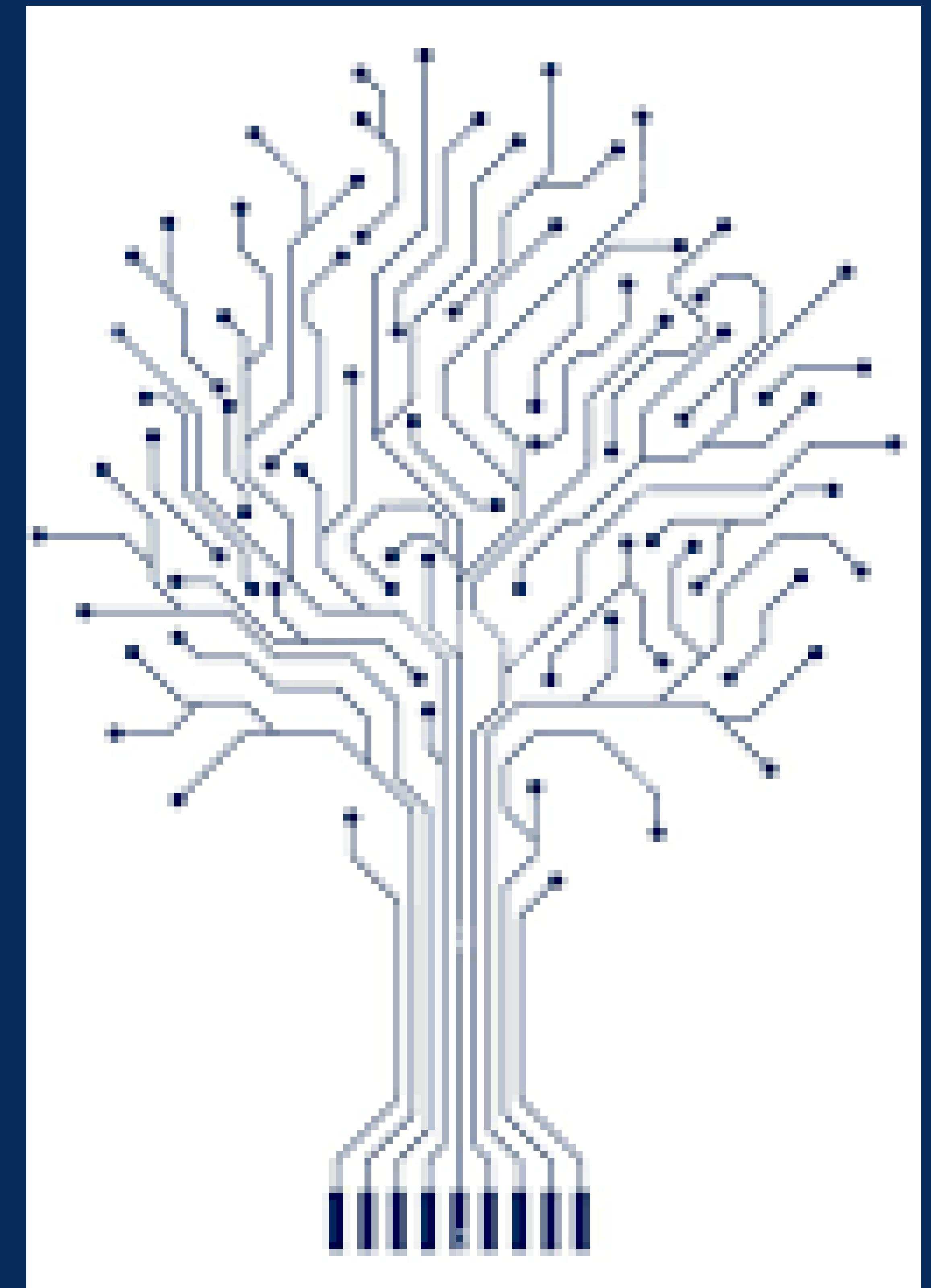
MODULO 2: Alberi binari e BST

Prof.ssa Giulia Cisotto

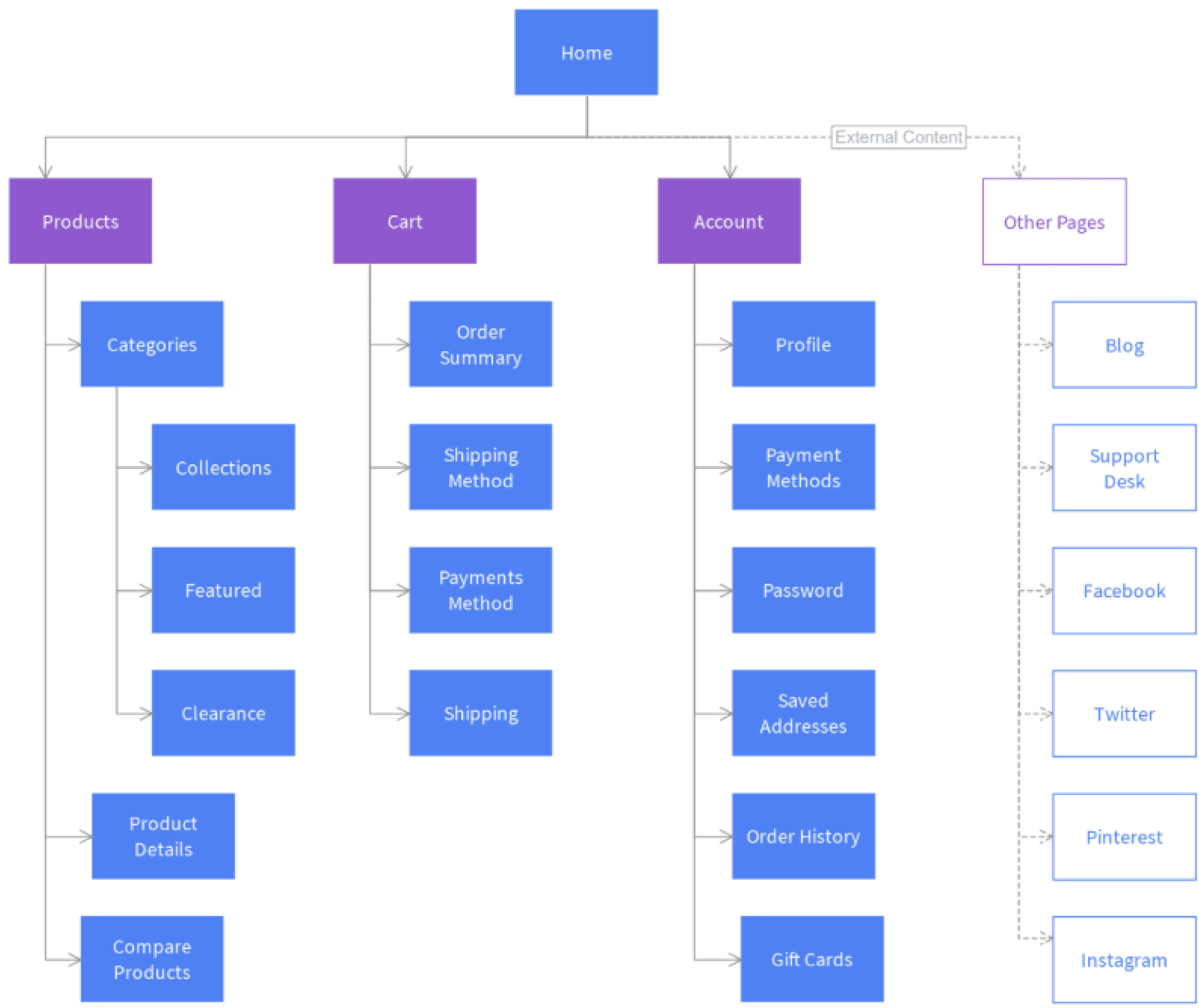
giulia.cisotto@units.it

Trieste, 28 aprile 2026

AGENDA DI OGGI

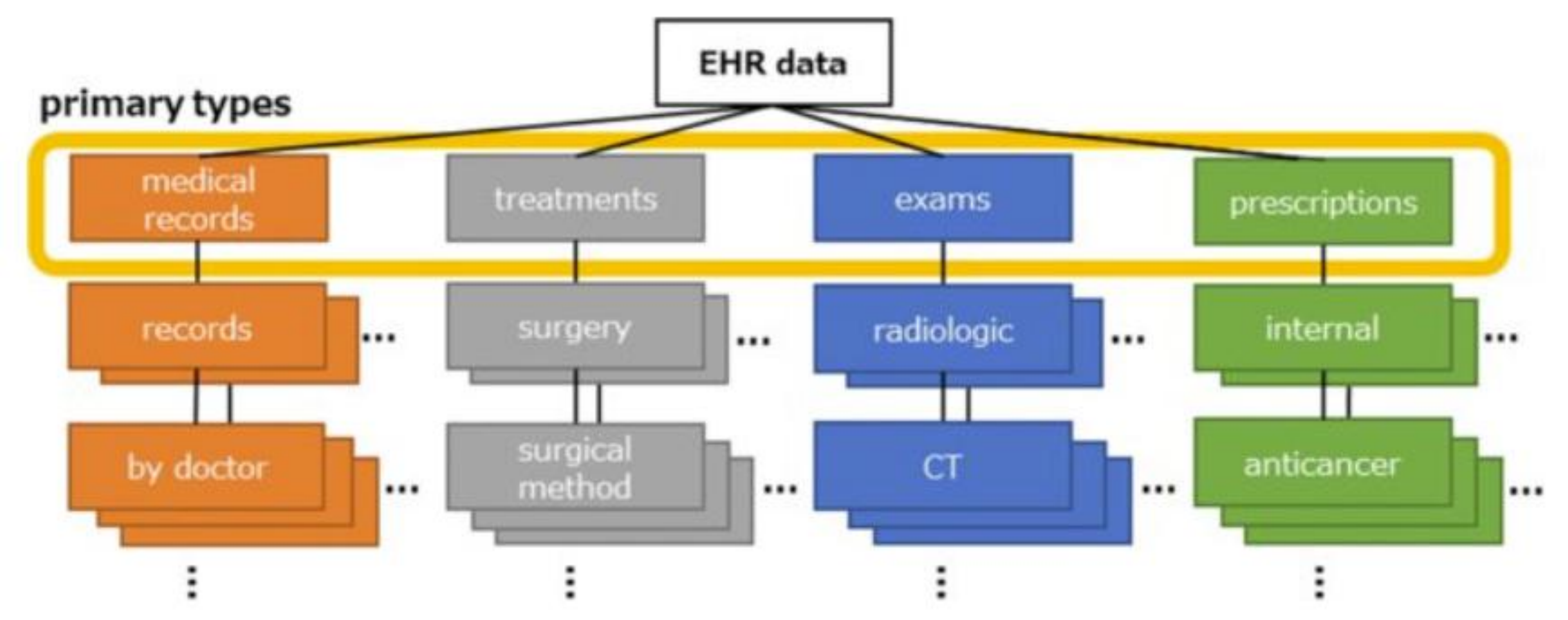


- Alberi binari e BST: definizione
- Operazioni su BST:
  - visita
  - inserimento
  - ricerca
  - cancellazione

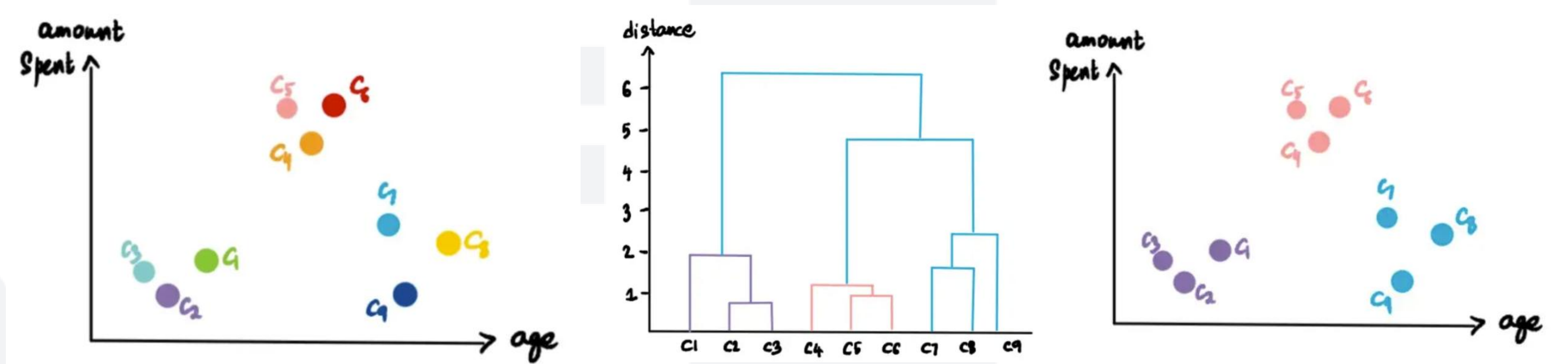


Sito web di e-commerce (link)

[The example tree structure of EHR data types and primary types](#)



Dendrogramma in clustering gerarchico (unsupervised **machine learning**)



UNIVERSITÀ DEGLI STUDI DI TRIESTE

ALBERI

Sono strutture dati ASTRATTE e ORDINATE dove ogni oggetto della struttura è definito dai seguenti elementi:

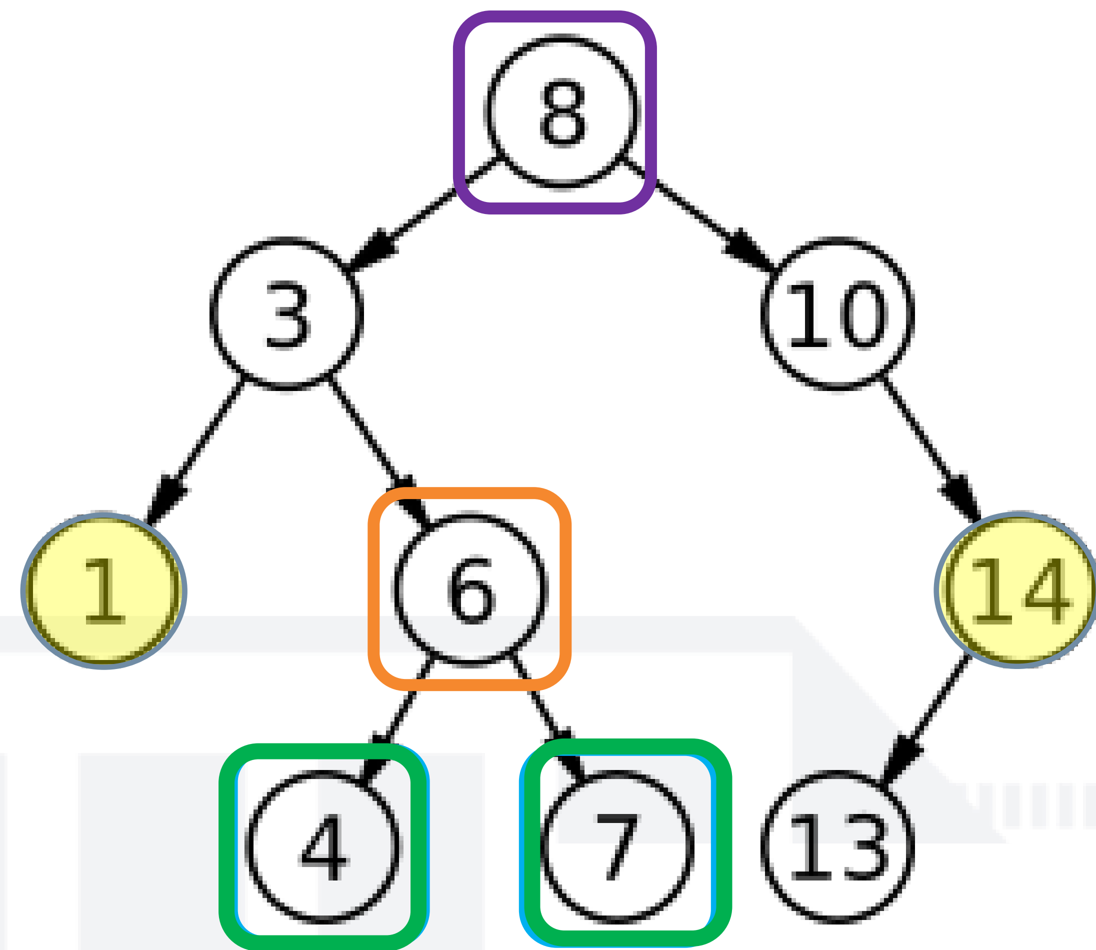
- **Chiave** (key) è un identificativo dell'oggetto
- **Contenuto** (dato che può essere un numero, una stringa, un'ulteriore struttura dati..)
- **Riferimenti** (o puntatori) all'oggetto precedente e agli oggetti successivi

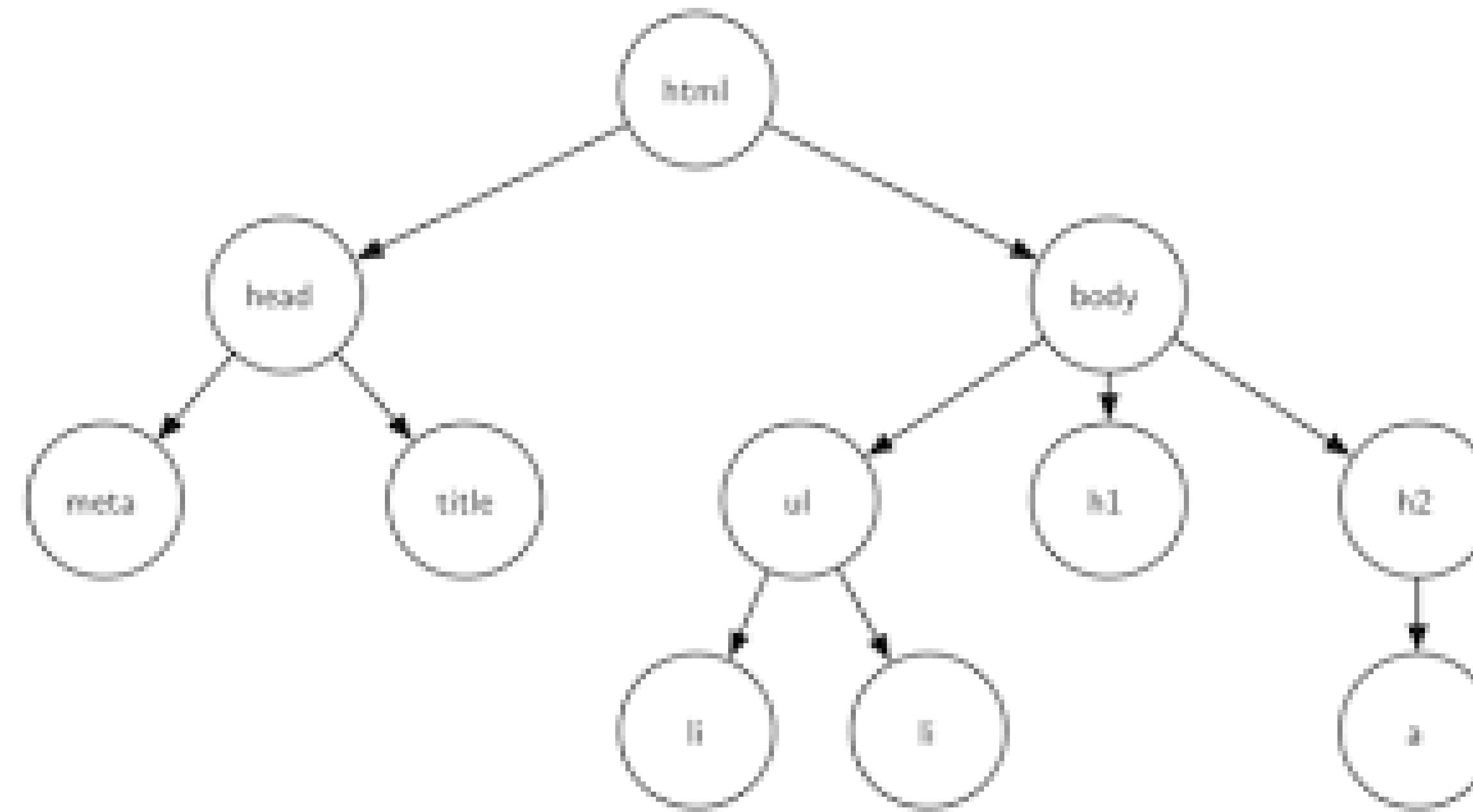
Vincoli (proprietà per poter definire la struttura proprio «un albero»):

1) Per ogni oggetto c'è un solo oggetto precedente
2) Non ci sono cicli, ovvero non posso ritornare ciclicamente su un oggetto già visitato soltanto andando avanti

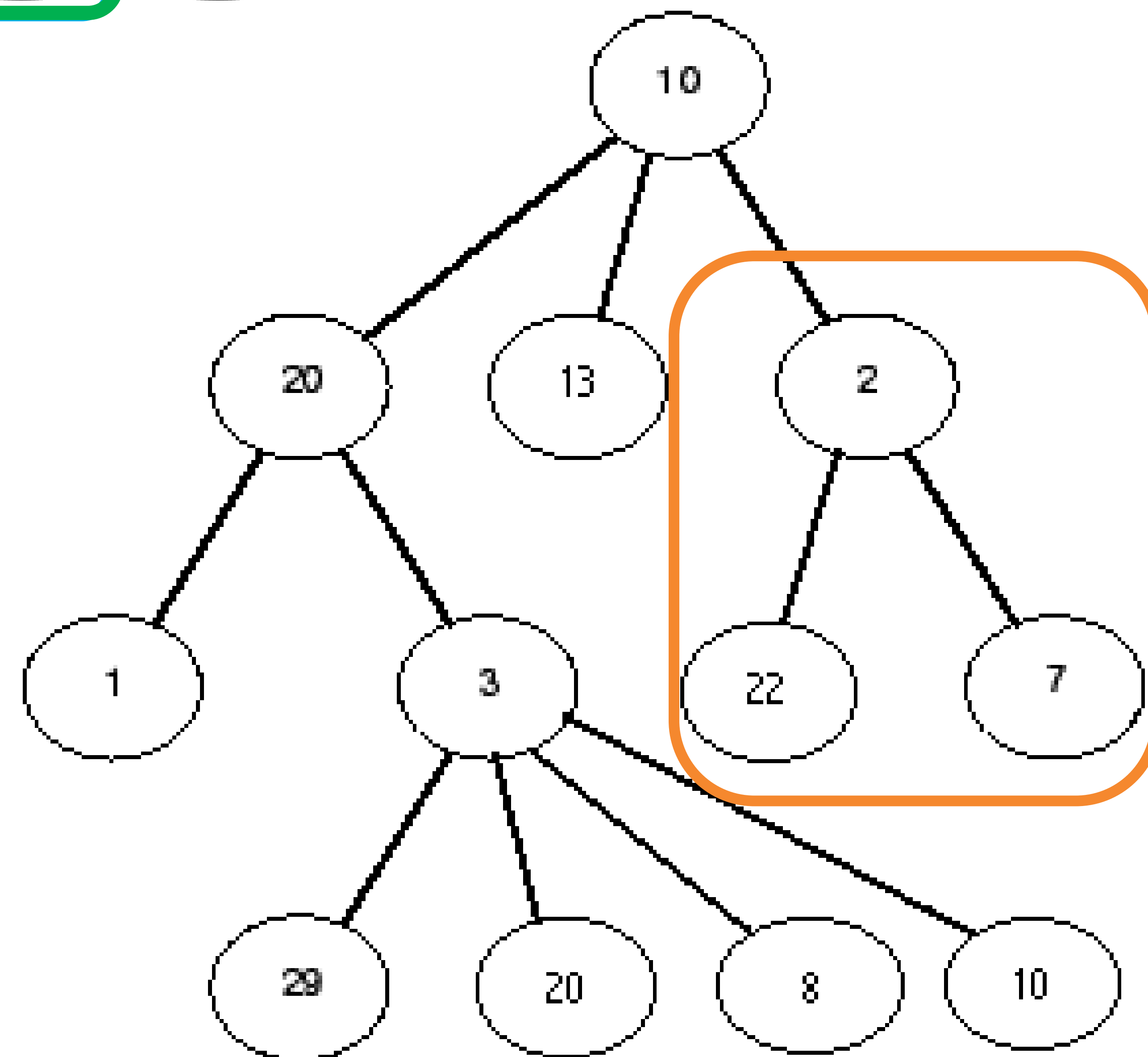
UNIVERSITÀ DEGLI STUDI DI TRIESTE

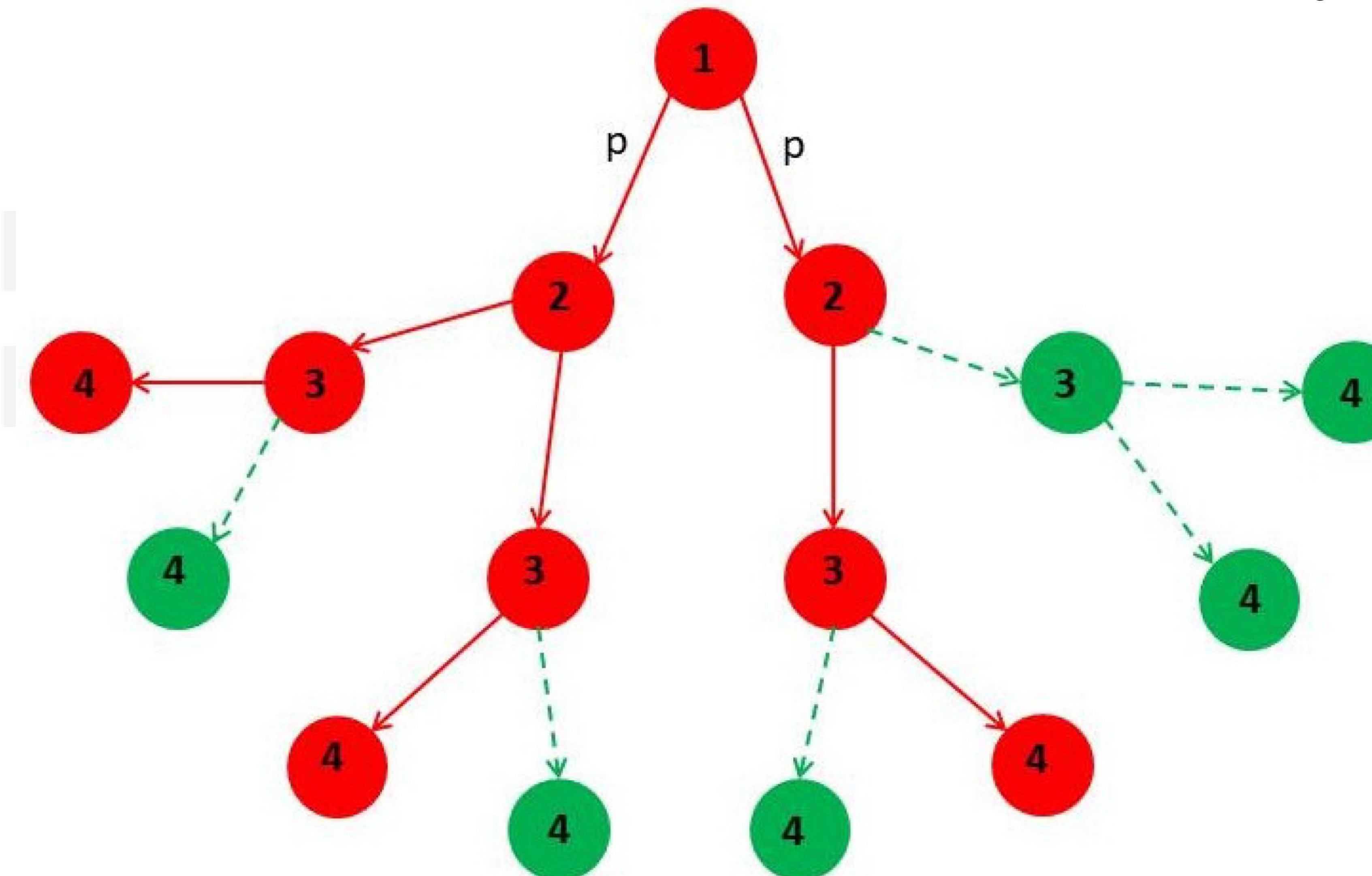
ALBERI: ESEMPI E DEFINIZIONI AGGIUNTIVE





- **Genitore (parent)**
- **Figli (children)**
- **Radice (root)**
- **Nodi foglia (leaf nodes)**
- **Minimo e massimo**
- **Sottoalbero**





UNIVERSITÀ DEGLI STUDI DI TRIESTE

VINCOLI E VANTAGGI

A seconda dell’applicazione posso imporre nuovi vincoli e «costringere» l’inserimento di nuovi elementi in determinate posizioni e «vietarne» l’inserimento in altre.

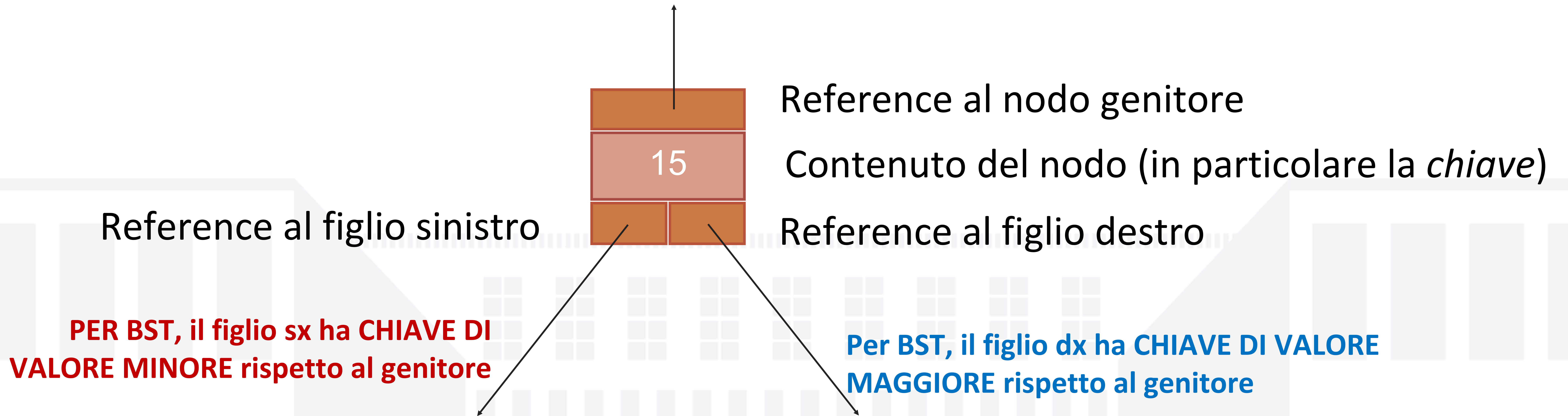
Intuitivamente: inserimento richiederà più «attenzione» (e tempo), ma la ricerca sarà avvantaggiata, sapendo già dove sarà – con maggior probabilità – l’elemento cercato.

Una delle tipologie di alberi maggiormente utilizzata sono gli:

ALBERI BINARI DI RICERCA O BINARY SEARCH TREE (BST)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

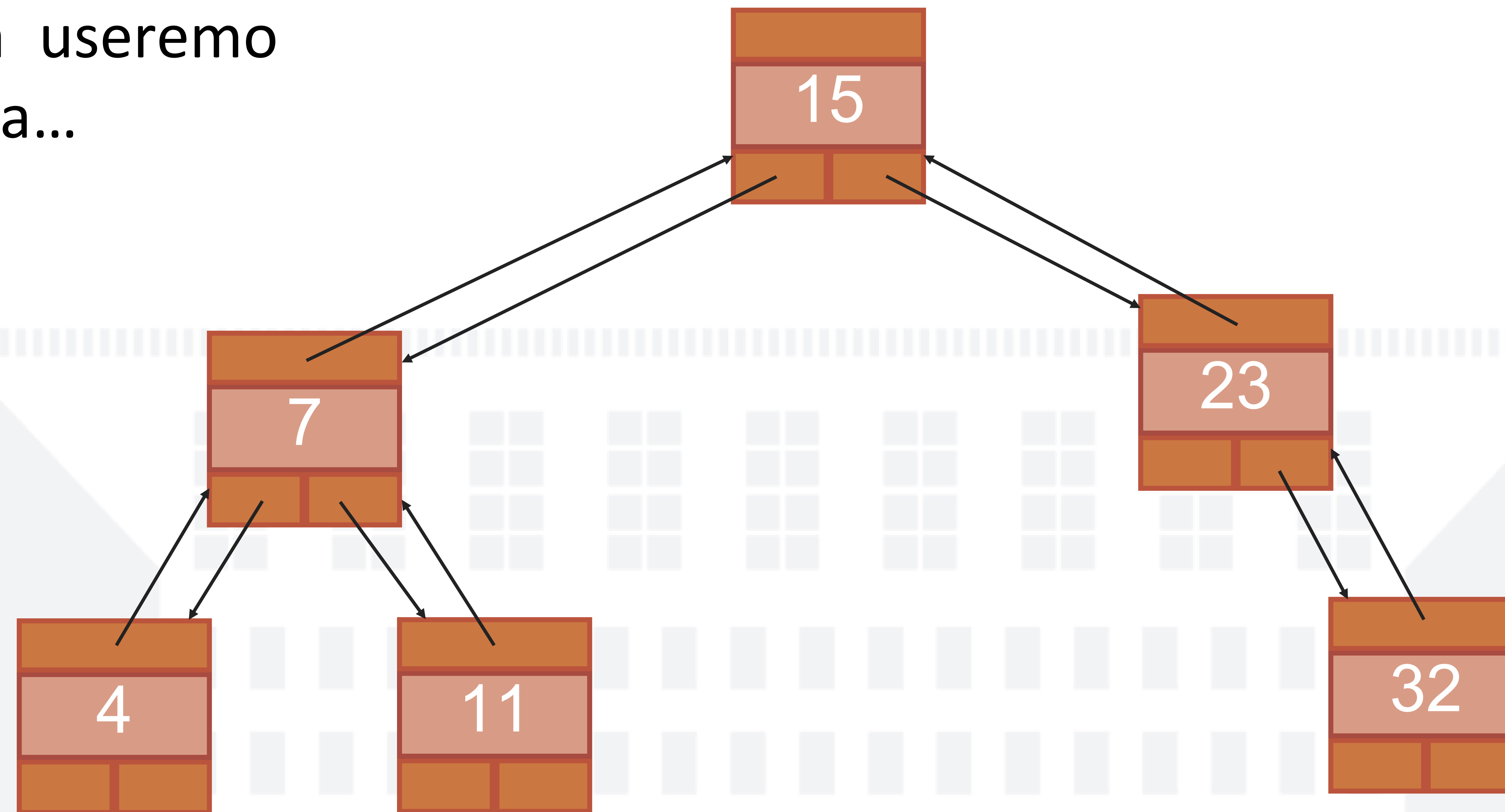
NODO DI UN ALBERO BINARIO DI RICERCA E DI UN BST



A seconda dell'implementazione e delle operazioni da svolgere potremmo non avere il riferimento al nodo genitore, avere un riferimento al nodo "fratello", etc.

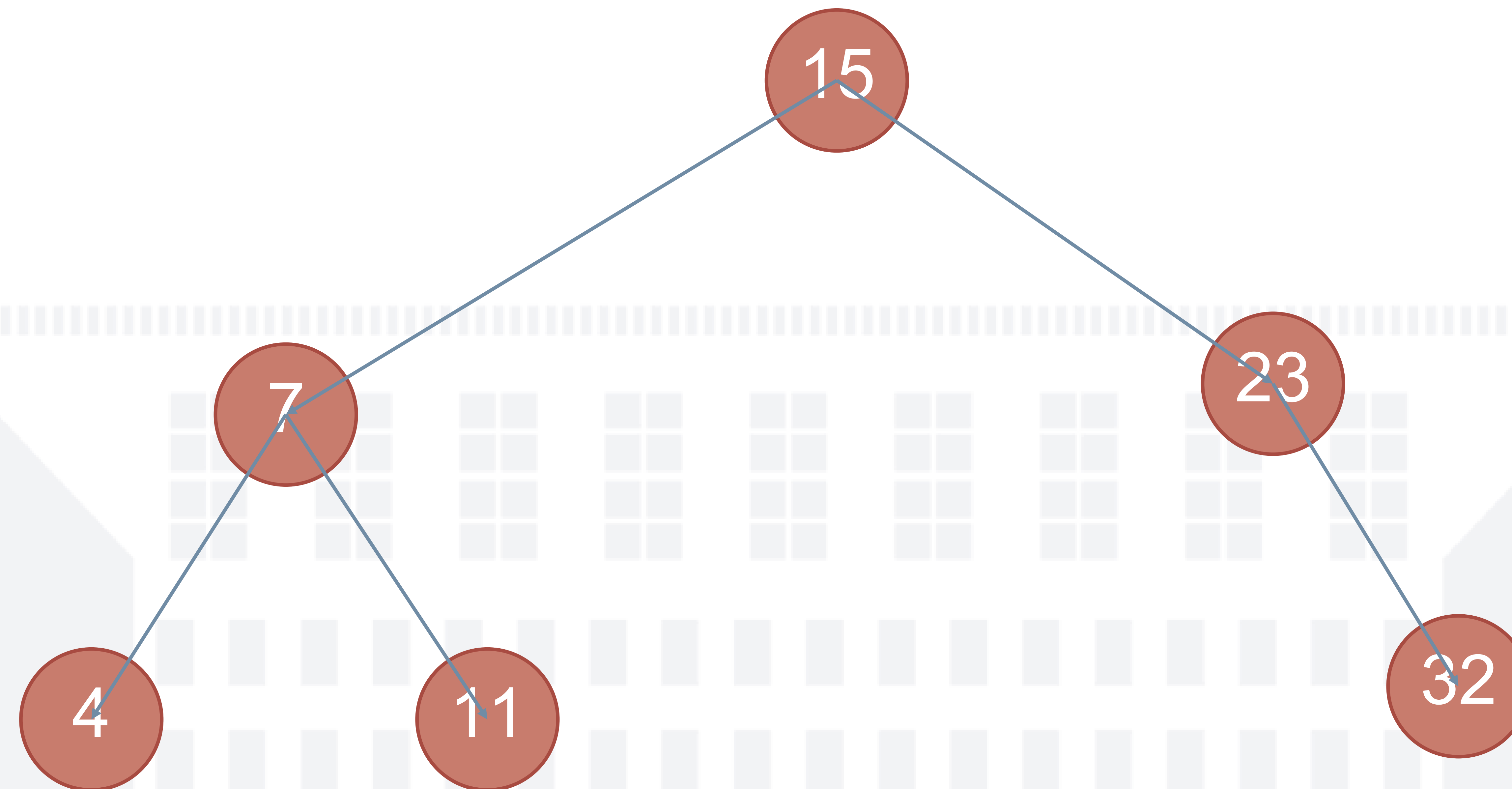
BST: RAPPRESENTAZIONE GRAFICA

Qui visualizziamo tutti i riferimenti, ma spesso per chiarezza useremo una notazione più stilizzata…



… ricordando che in ogni caso tutti questi reference continuano ad esistere

BST: RAPPRESENTAZIONE GRAFICA

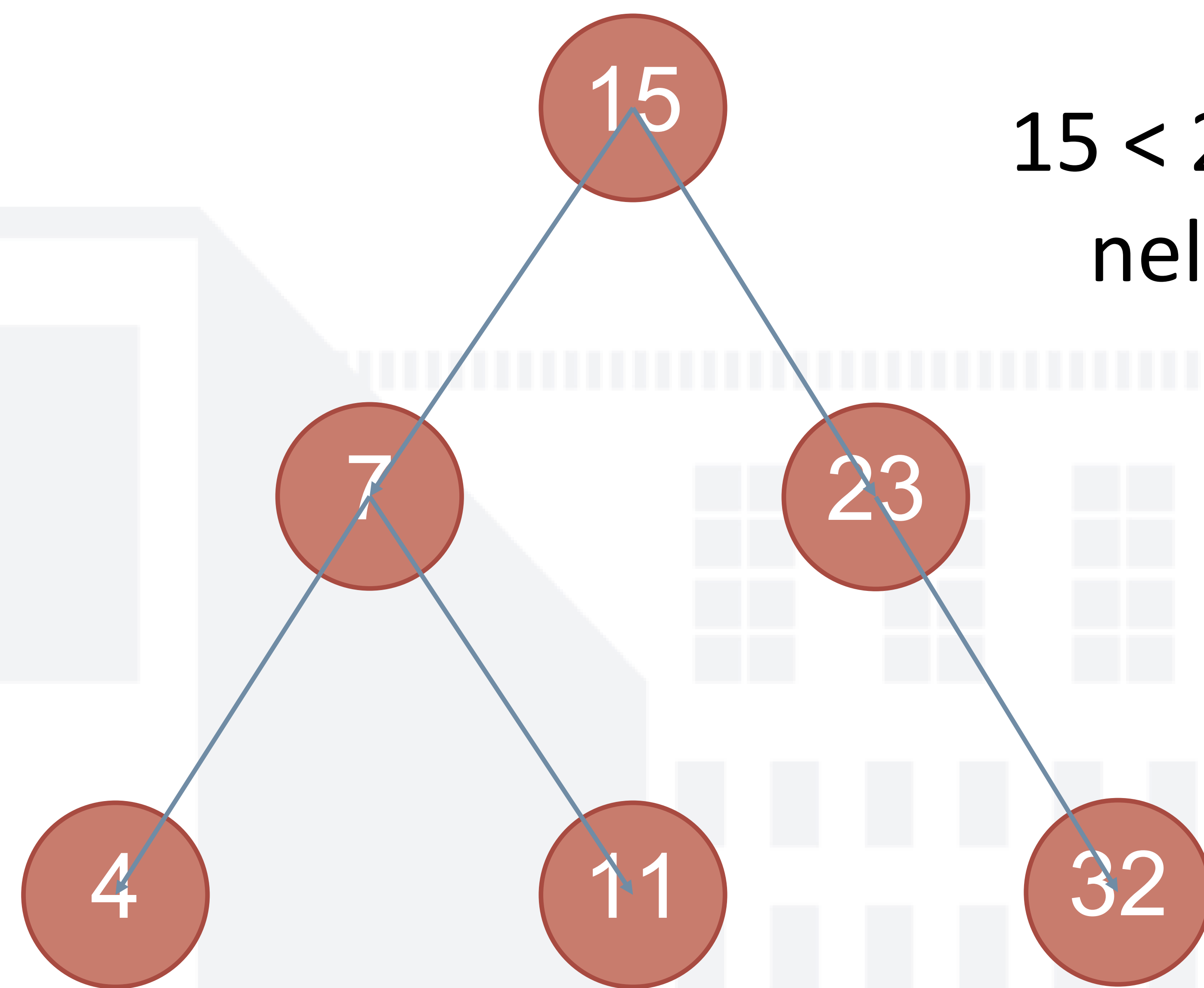


BST: VINCOLI

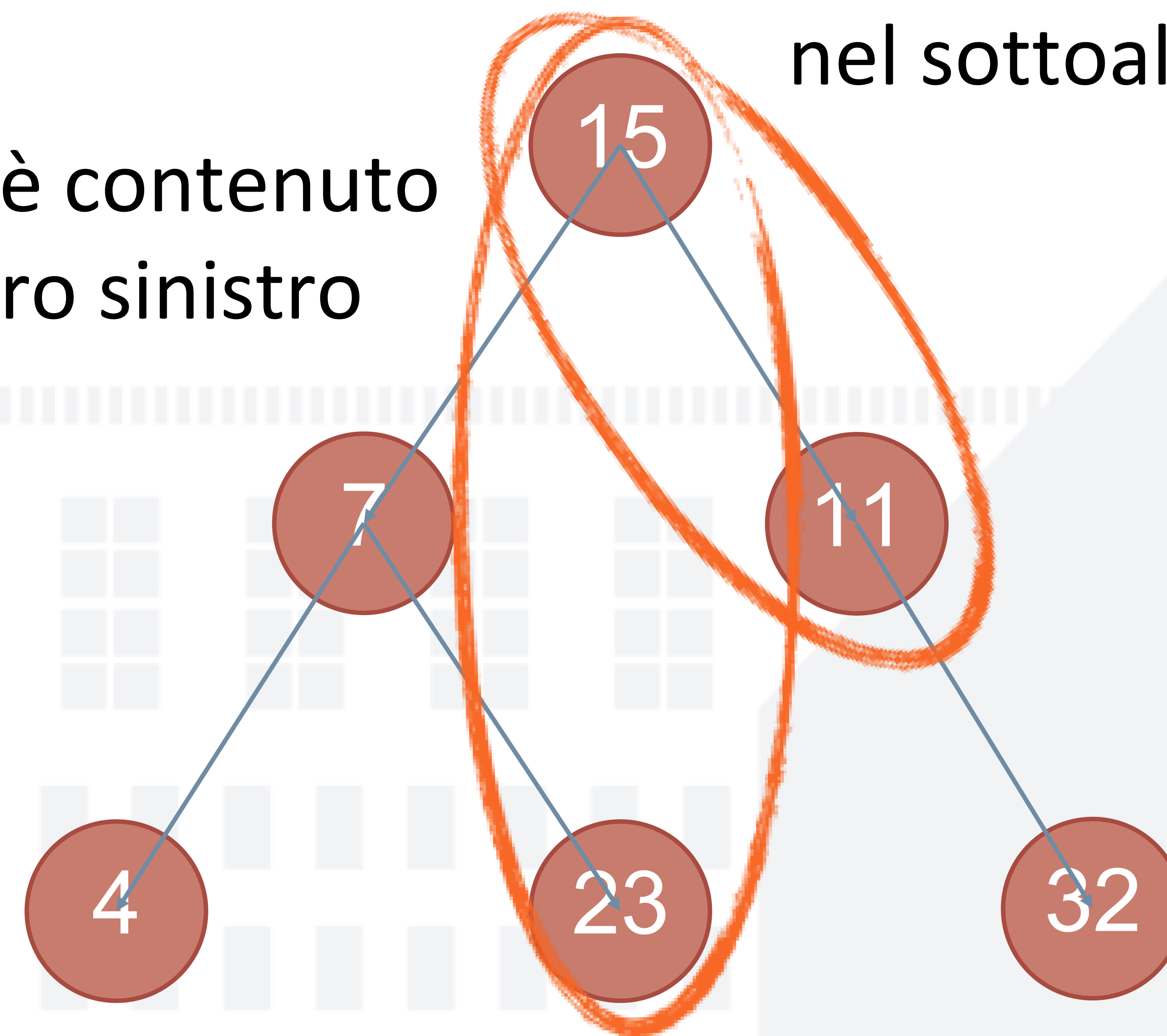
- Richiede **chiavi totalmente ordinate** (es. interi)
- Ogni nodo dell'albero contiene una chiave
- Ogni nodo dell'albero ha la seguente **proprietà**:
  - Tutti i nodi nel sottoalbero sinistro hanno chiave *minore* della chiave nel nodo
  - Tutti i nodi del sottoalbero destro hanno chiave *maggiore* della chiave nel nodo

- **Ogni sottoinsieme dell'albero è, a sua volta, un BST** che gode delle stesse proprietà.

ALBERO BINARIO DI RICERCA VS ALBERO BINARIO



15 < 23, ma 11 è contenuto
nel sottoalbero sinistro



15 > 11, ma 11 è contenuto
nel sottoalbero destro

E’ un albero binario di ricerca

NON è un albero binario di ricerca

UNIVERSITÀ DEGLI STUDI DI TRIESTE

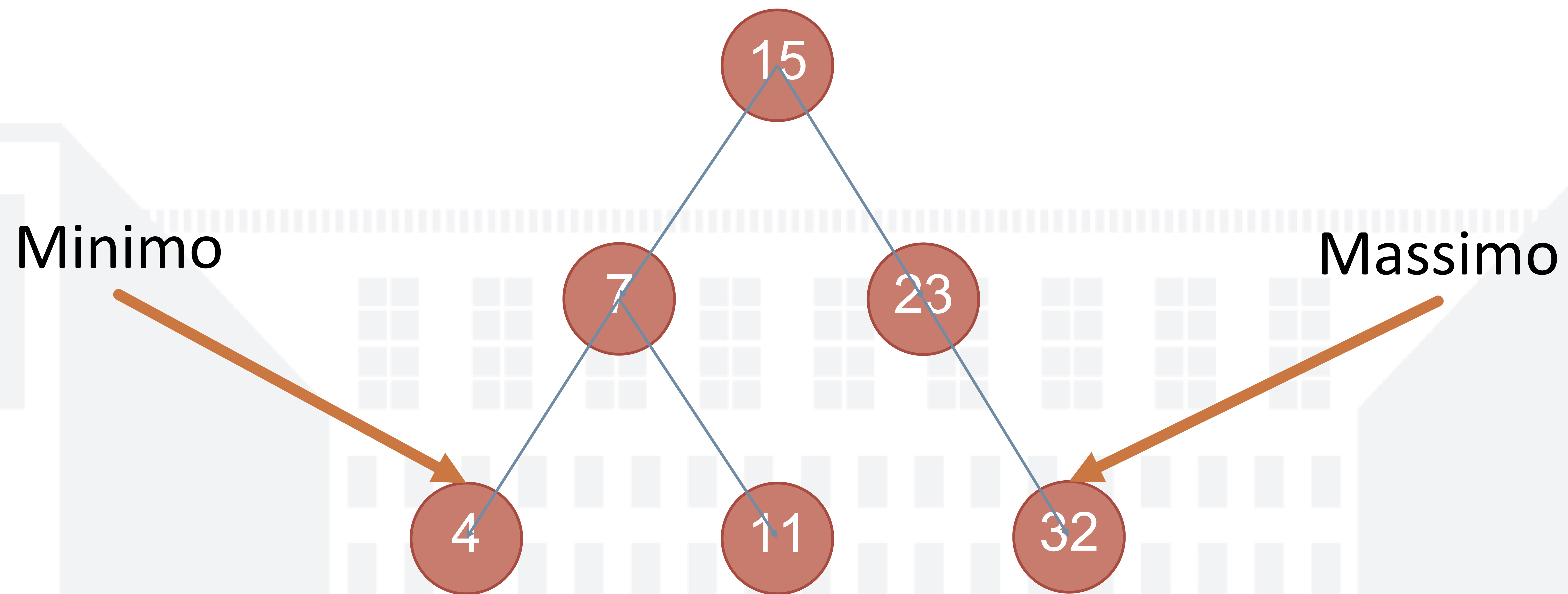
ALBERI BINARI DI RICERCA: MASSIMO E MINIMO

Per definizione il **massimo** sarà il nodo "più a destra" nell'albero.

È sufficiente muoversi dalla radice verso destra fino a quando si incontra un nodo senza figlio destro: il valore che contiene è il massimo.

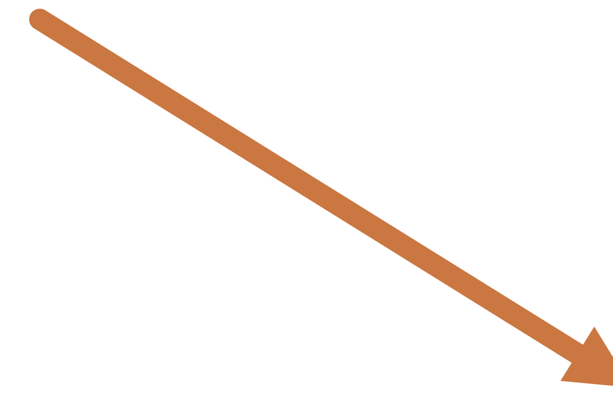
Simmetricamente per il **minimo**: è sufficiente continuare a spostarsi verso sinistra.

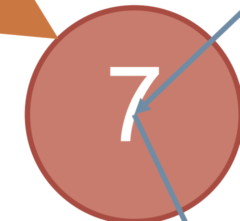
ALBERI BINARI DI RICERCA: MASSIMO E MINIMO

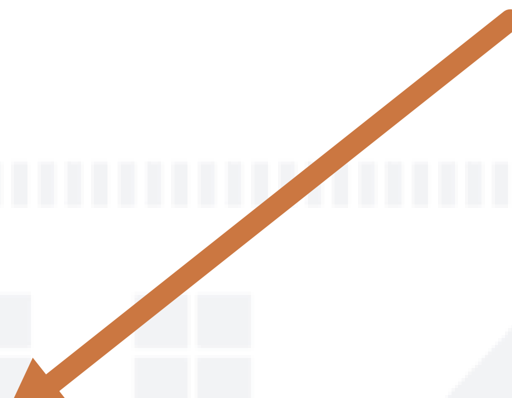


Dato che dobbiamo "scendere" fino alle foglie, nel caso peggiore siamo comunque limitati dall'altezza dell'albero per trovare il minimo o il massimo

ALBERI BINARI DI RICERCA: MASSIMO E MINIMO







15
7
23
11
17
32
24

Non è detto che minimo e massimo siano sempre delle foglie!

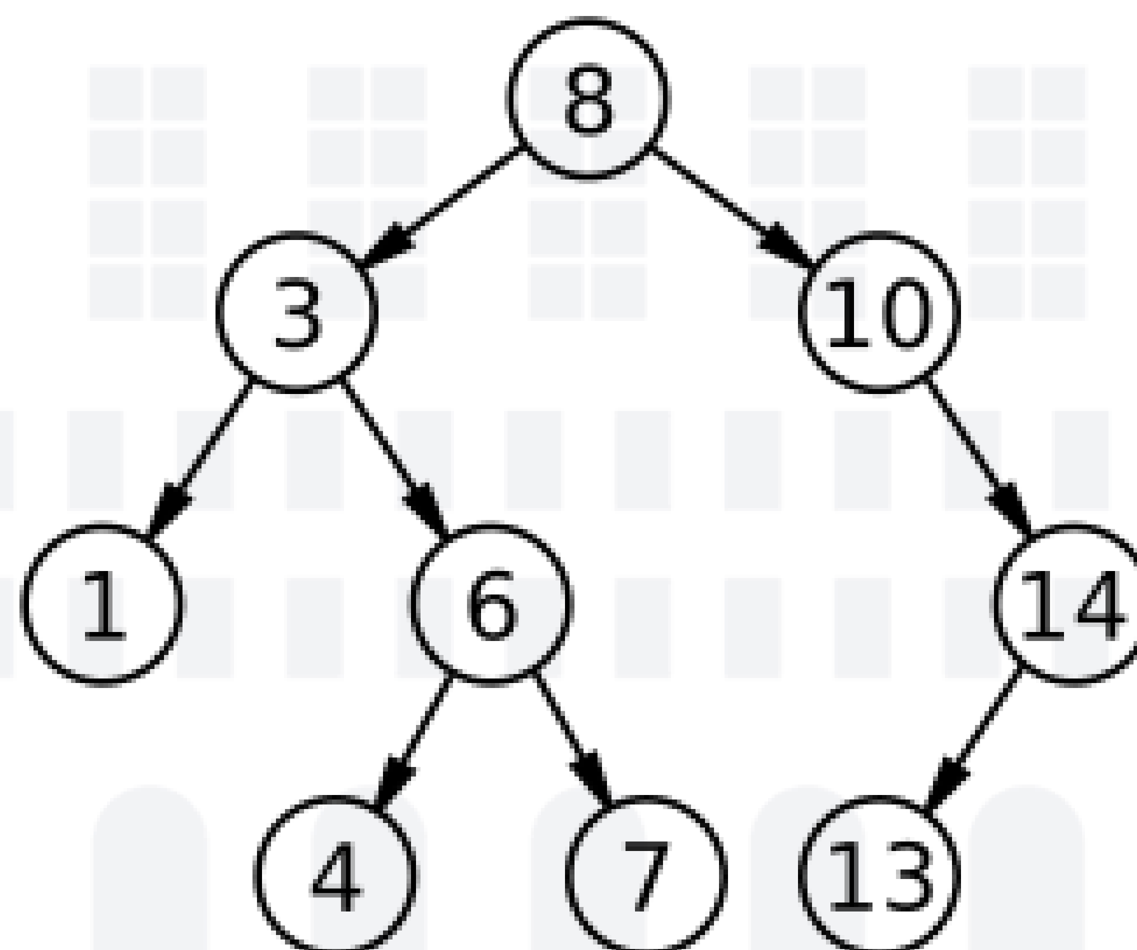
vale per tutti gli alberi binari (BST e non)

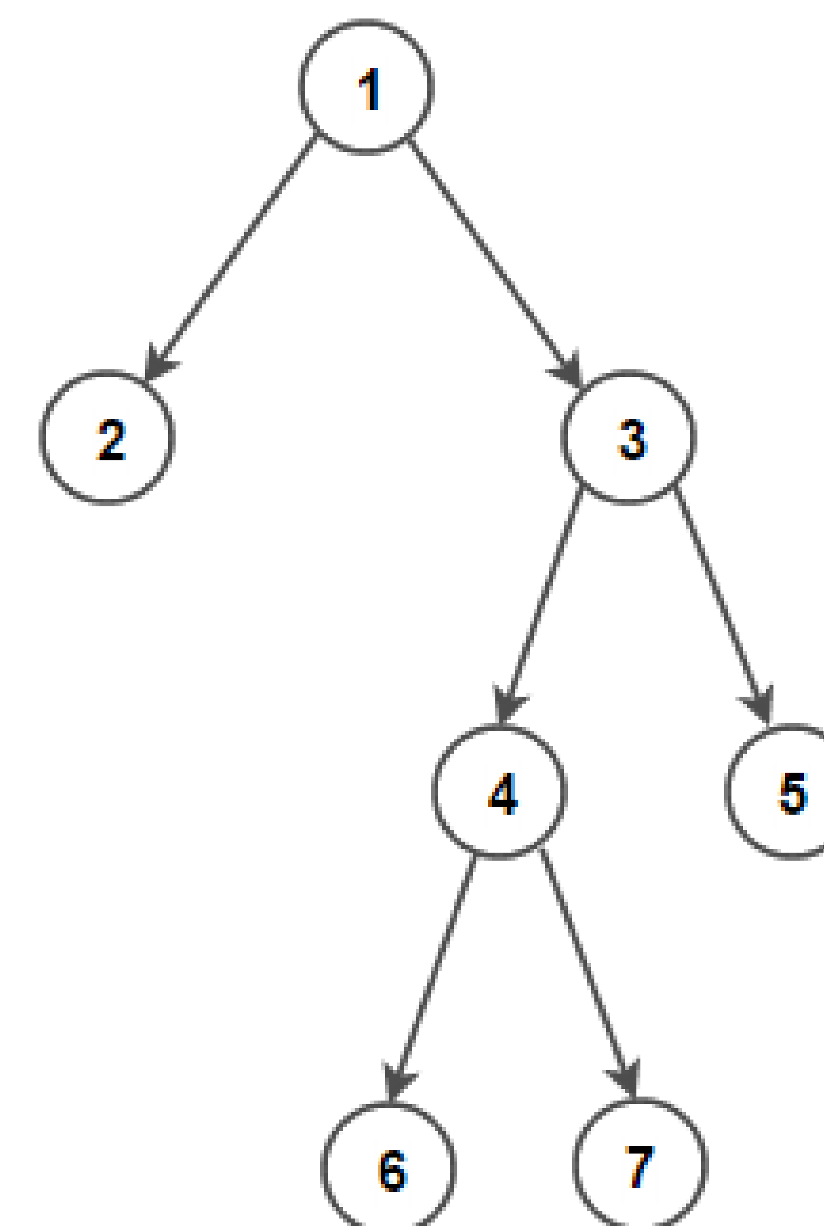
ALBERI BINARI: ULTERIORI DEFINIZIONI UTILI

ALTEZZA DI UN ALBERO (h)**:** distanza (in termini di *generazioni* tra un livello e l'altro) che porta dalla *root* al nodo foglia più basso possibile

PROFONDITA' DI UN NODO: distanza dalla *root* ad un nodo specifico.

ALBERO BILANCIATO: è un albero in cui la differenza assoluta tra l'altezza del sottoalbero sinistro e destro *per ogni nodo* è 0 o 1.





NOTA: Per un albero binario e bilanciato, h ≈ $log_2 n$ *dove n=#nodi*

UNIVERSITÀ DEGLI STUDI DI TRIESTE

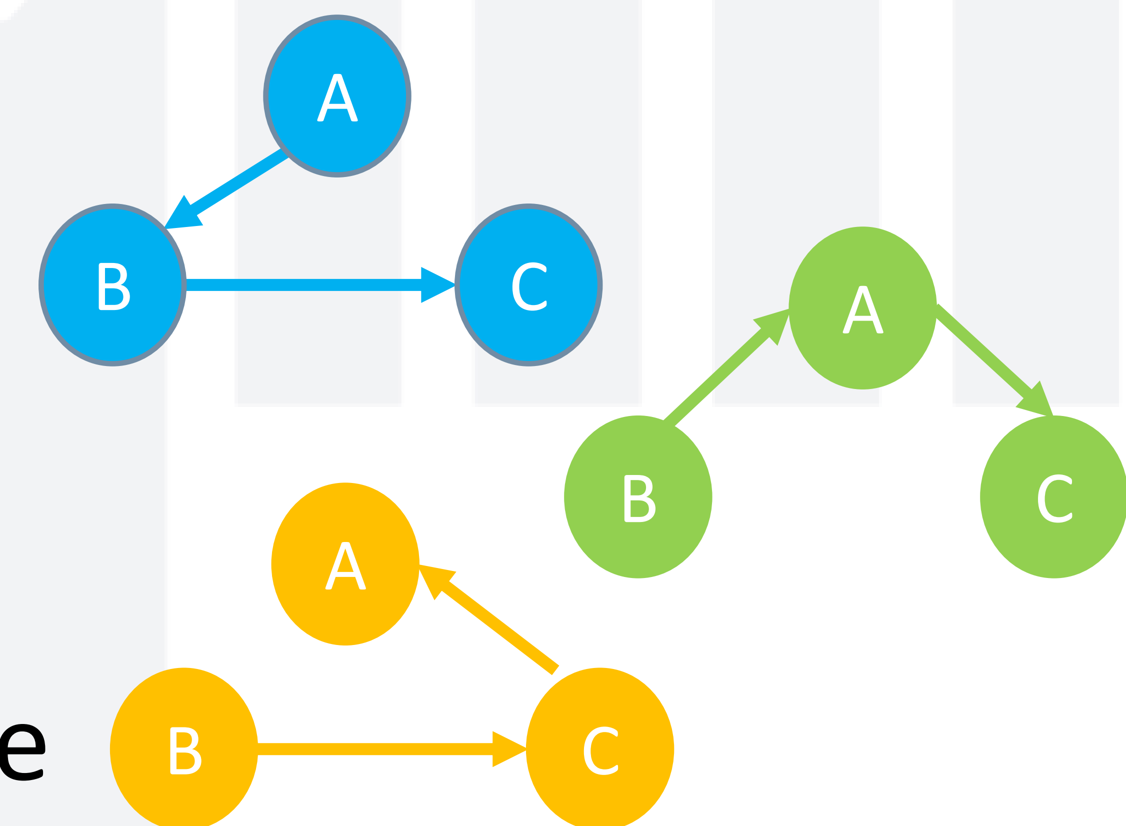
VISITA DEI NODI IN PRE-ORDINE, IN-ORDINE E POST-ORDINE

- Gli **alberi binari generalmente** hanno diversi modi in cui possono enumerare gli elementi (visitandoli tutti)
- I tre modi principali differiscono solo per il momento in cui viene visitato il nodo corrente:

vale per tutti gli alberi
- **Pre-ordine**: nodo corrente, sottoalbero *sinistro*, sottoalbero destro

vale solo per alberi binari
- **In-ordine**: sottoalbero *sinistro*, nodo corrente, sottoalbero destro

vale per tutti gli alberi
- **Post-ordine**: sottoalbero *sinistro*, sottoalbero destro, nodo corrente



Definizioni di PREDECESSORE e SUCCESSORE di un nodo: in un BST sono i nodi che contengono il valore immediatamente successivo o precedente rispetto a un nodo dato, seguendo l'ordine delle chiavi. In un **albero binario generico** sono legati alla **sequenza di attraversamento →** senza specificare il tipo di visita, i termini sono ambigui

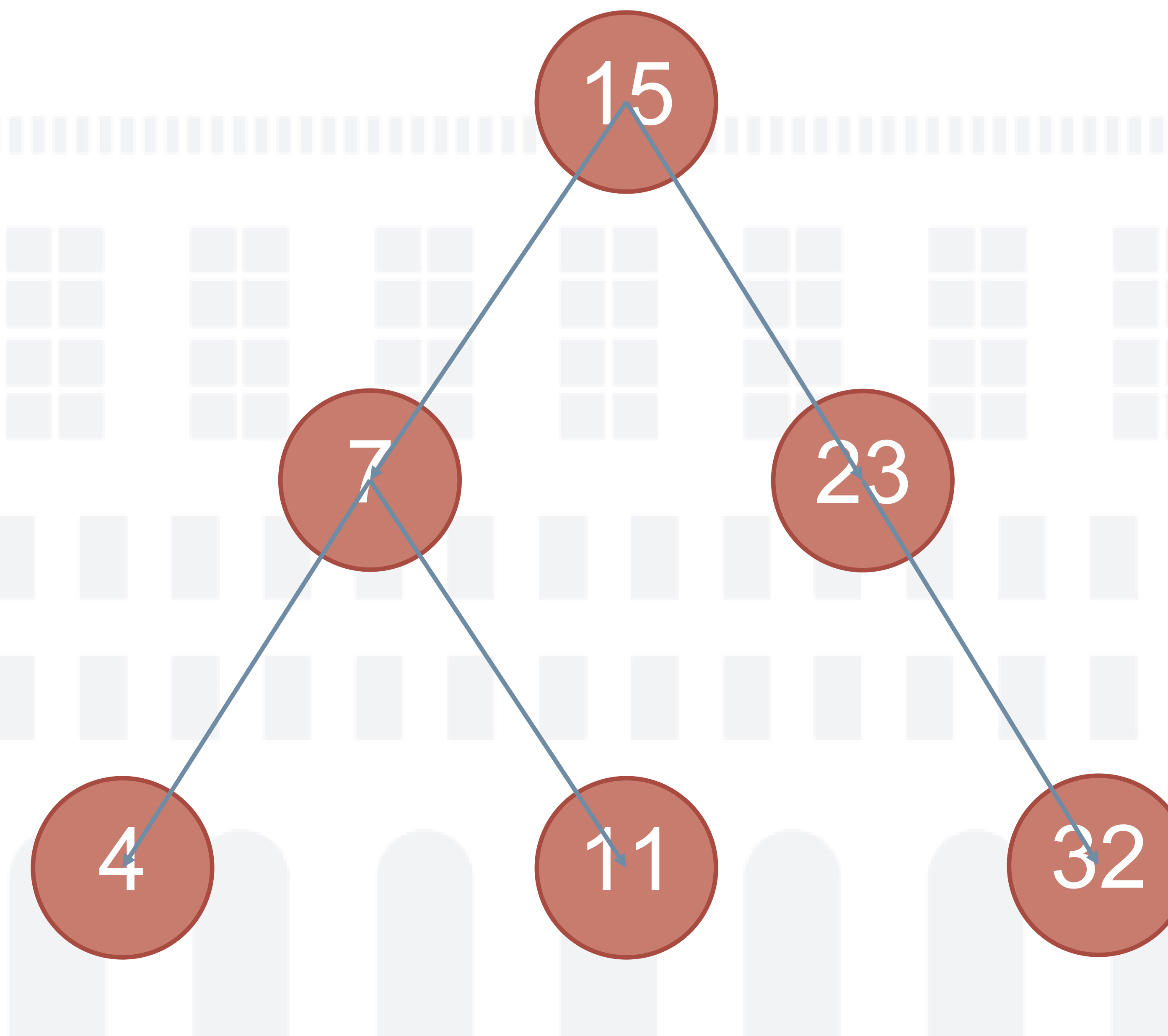
VISITA DEI NODI IN UN BST: ESEMPIO

Visita in pre-ordine: 15 7 4 11 23 32

Visita in ordine: 4 7 11 15 23 32

Visita in post-ordine: 4 11 7 32 23 15

La visita in ordine visita sempre gli elementi **in ordine del valore della chiave** (partendo dalla foglia più a sinistra e finendo alla foglia più a destra)



Visita in pre-ordine: parto dalla radice, entro nel sottoalbero sinistro. Il «parent» è il nodo corrente, quindi visito il suo sottoalbero sinistro … Una volta esaurito il sottoalbero sinistro, allora riparto dalla radice e ripeto per il sottoalbero di destra.

Visita in post-ordine: parto dal basso a sinistra, visito la prima foglia a sinistra, poi l'eventuale «fratello», poi il parent, così esaurisco il sottoalbero sinistro. Poi riparto dalla foglia più a sinistra del sottoalbero di destra e ripeto…

RICERCA SU BST: ALGORITMO

- Dato un BST e una chiave, la ricerca può essere suddivisa in ***quattro casi***:
- **Albero vuoto**: l'elemento non è contenuto
- **La chiave della radice** è quella che stiamo cercando: abbiamo trovato l'elemento
- La chiave che stiamo cercando è **minore** della chiave nella radice: cerchiamo ricorsivamente nel sottoalbero di **sinistra**
- La chiave che stiamo cercando è **maggiore** della chiave nella radice: cerchiamo ricorsivamente nel sottoalbero di **destra**

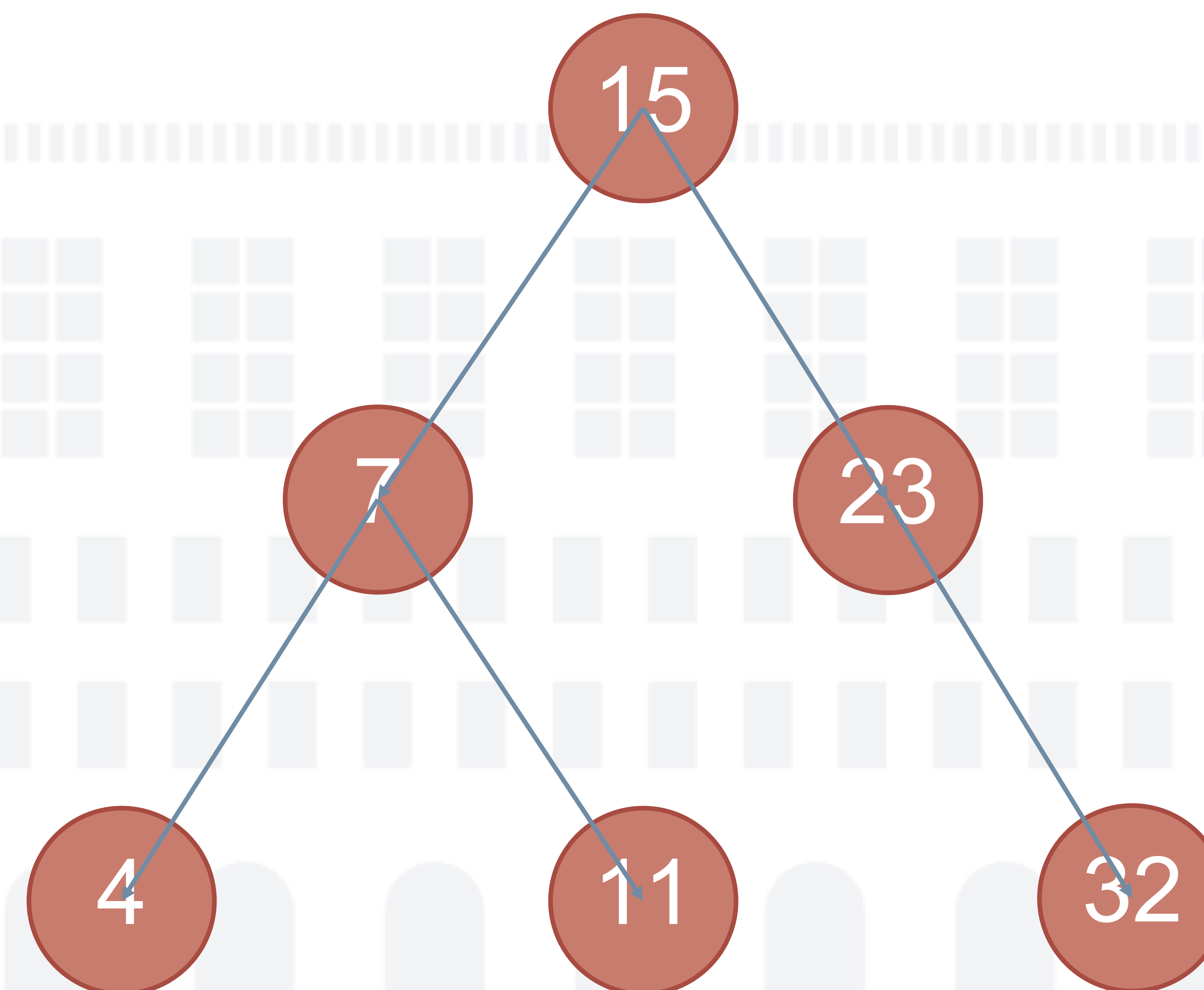
RICERCA SU BST: PSEUDOCODICE

```
Parametri: Nodo radice, key
if radice is None
  return None # l'albero è vuoto, quindi sicuramente la chiave non esiste
if radice.key == key
  return radice # abbiamo trovato un nodo con la chiave che cercavamo
if key < radice.key
  return ricerca(radice.left, key) # ricerca nel sottoalbero sinistro
else # ovvero key > radice.key
  return ricerca(radice.right, key) # ricerca nel sottoalbero destro
```

Ricorsività!!

RICERCA SU BST: ESEMPIO

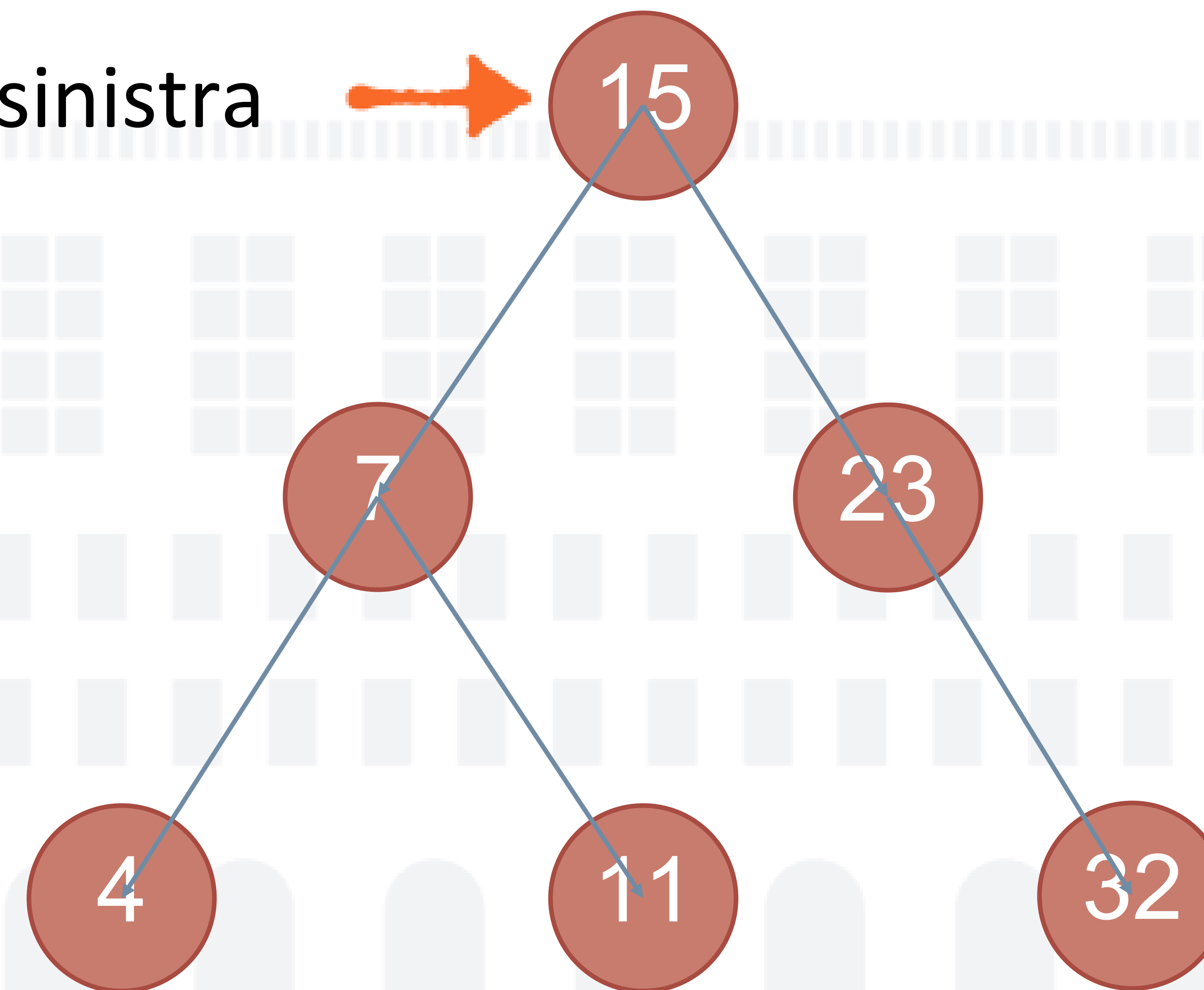
Supponiamo di voler trovare l’elemento di chiave 11



RICERCA SU BST: ESEMPIO

Supponiamo di voler trovare l'elemento di chiave 11

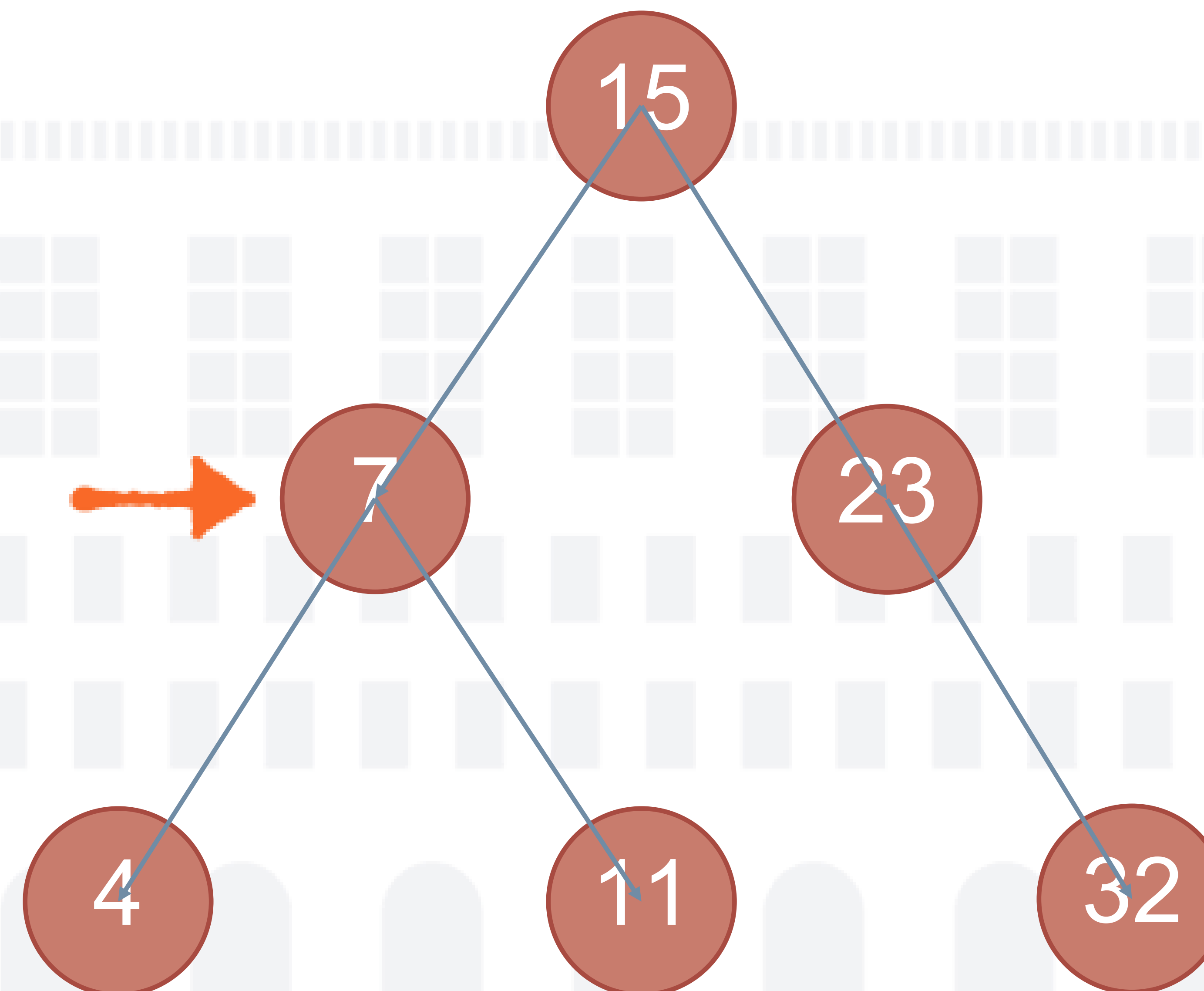
11 < 15, quindi ci muoviamo a sinistra



RICERCA SU BST: ESEMPIO

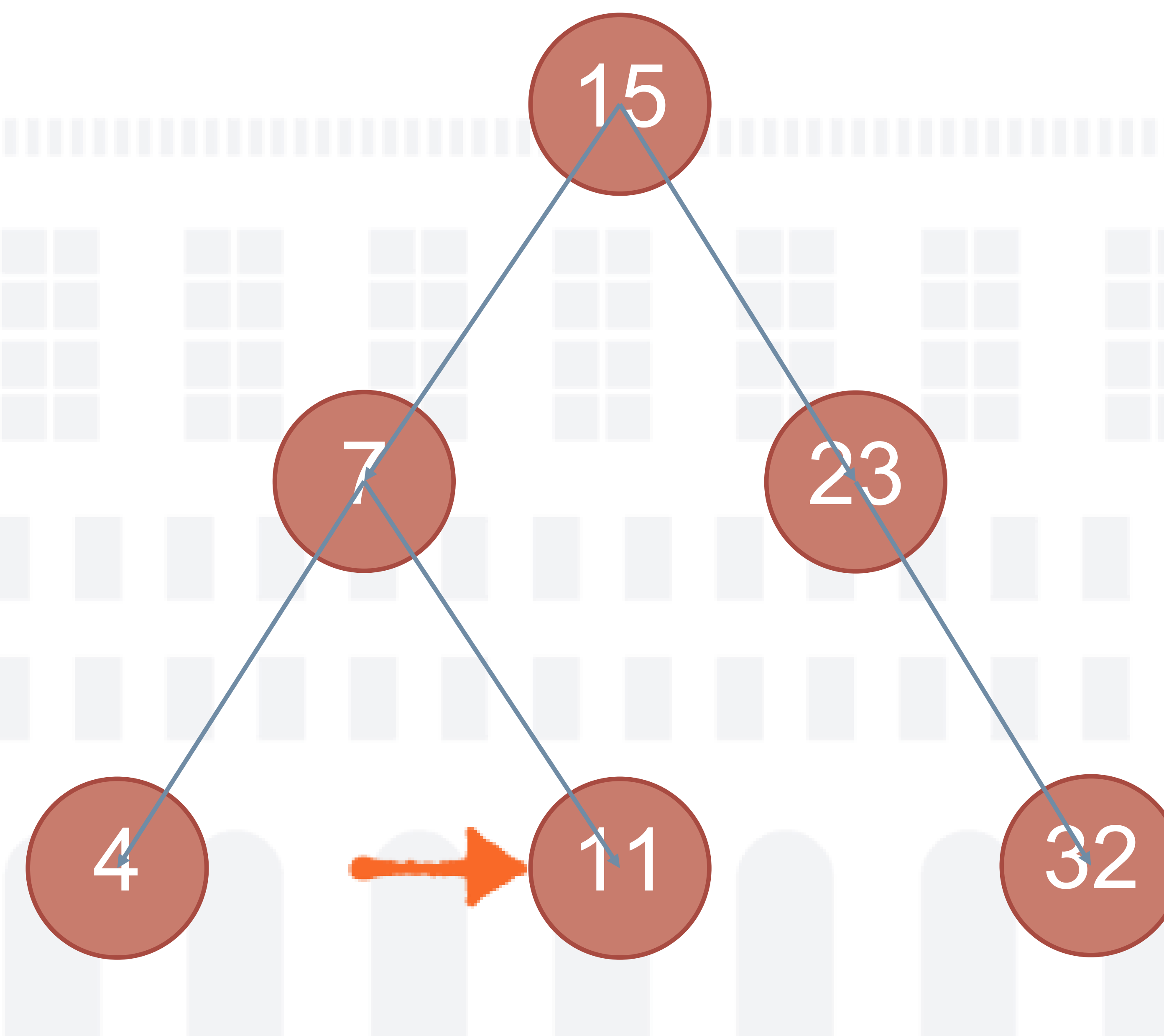
Supponiamo di voler trovare l'elemento di chiave 11

11 > 7, quindi ci muoviamo a destra



RICERCA SU BST: ESEMPIO

Supponiamo di voler trovare l'elemento di chiave 11



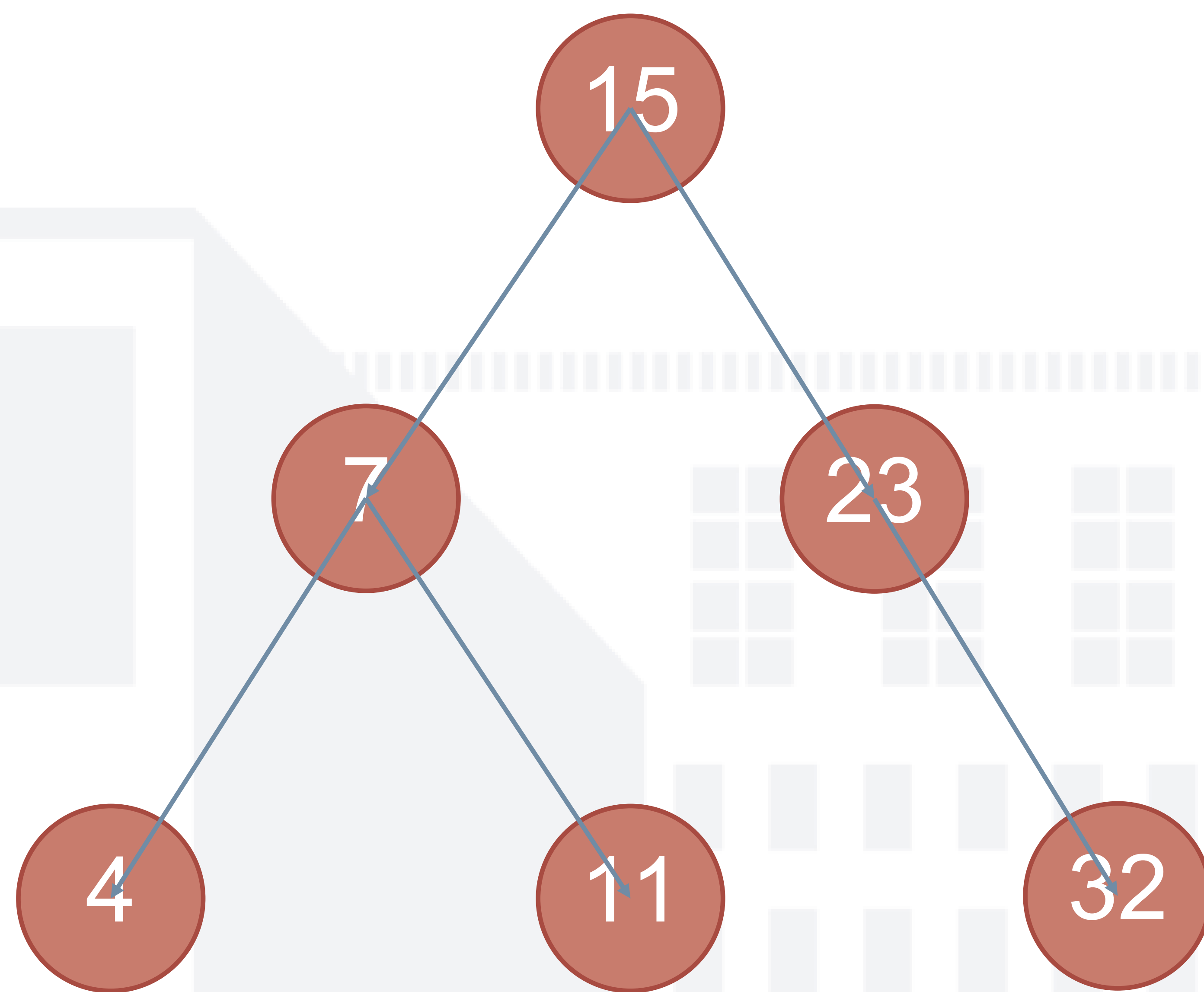
Trovato!

RICERCA SU BST: COMPLESSITA’

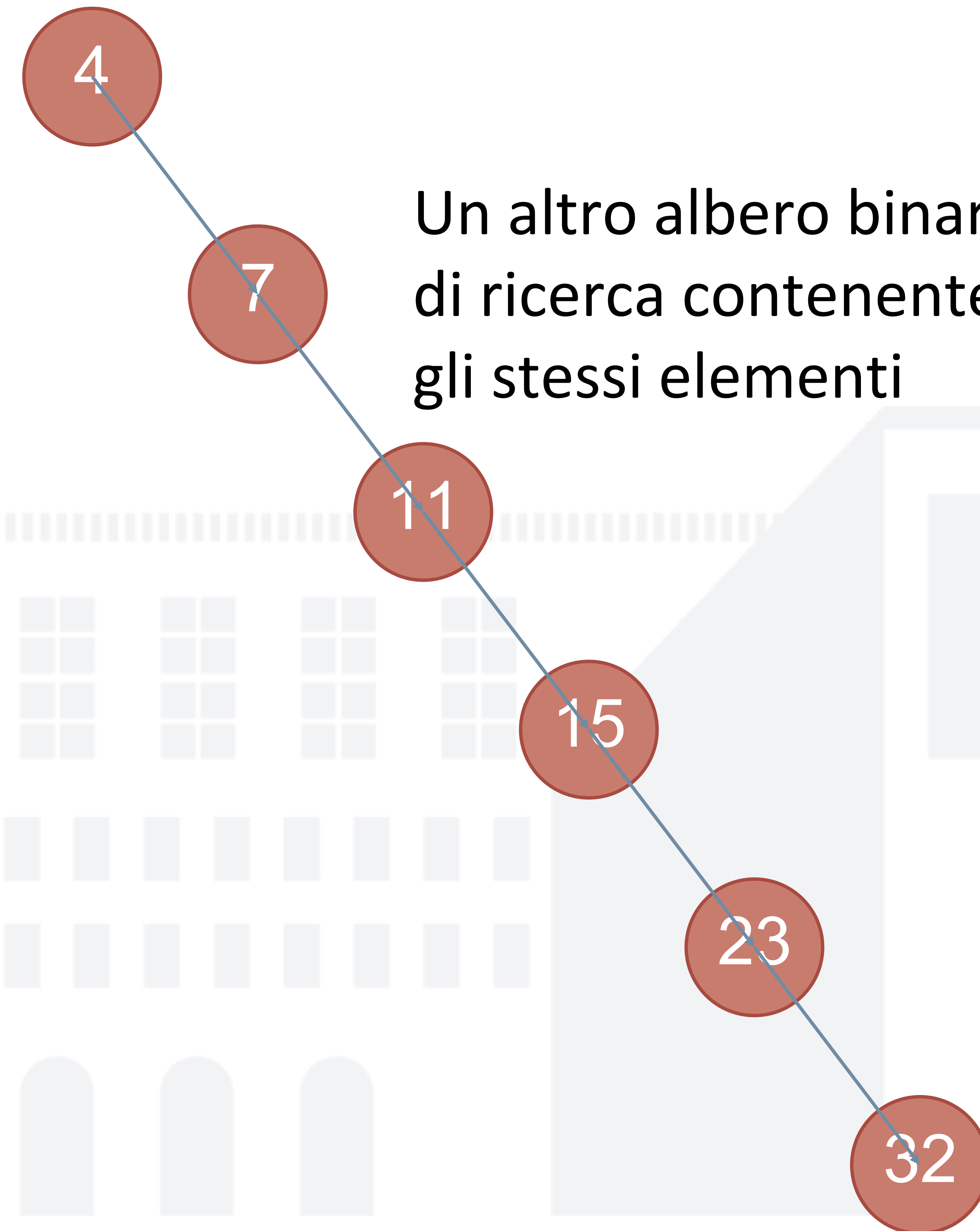
- Facciamo un ***numero costante di passi prima*** di ogni chiamata ricorsiva…
- … E facciamo un ***numero di chiamate ricorsive*** che è limitato dall’**altezza dell’albero** (ad ogni chiamata ricorsiva scendiamo di un livello)
- Quindi $O(h)$ dove h è l’altezza dell’albero.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

BST BILANCIATI E SBILANCIATI:



Albero binario di ricerca



Un altro albero binario di ricerca contenente gli stessi elementi

CONSEGUENZE

- **Nel caso migliore** abbiamo un albero con la profondità minima necessaria a contenere tutti gli elementi, un BST bilanciato, quindi la ricerca è particolarmente efficiente: $\boldsymbol{O(\log n)}$

- **Nel caso peggiore** abbiamo qualcosa di *simile ad una lista concatenata*, la profondità dell'albero è lineare rispetto al numero di elementi e **la ricerca richiede tempo** $\boldsymbol{O(n)}$

UNIVERSITÀ DEGLI STUDI DI TRIESTE

INSERIMENTO DI UN NUOVO NODO SU BST (1/2)

L'inserimento avviene in modo simile alla ricerca

Data una chiave k dobbiamo trovare un posto libero per inserire quella chiave **rispettando la proprietà dell'albero binario di ricerca**

Confrontiamo k con la chiave nella radice:

- Se k è maggiore, andrà inserita a destra della radice
- Se k è minore, andrà inserita a sinistra della radice

INSERIMENTO DI UN NUOVO NODO SU BST (2/2)

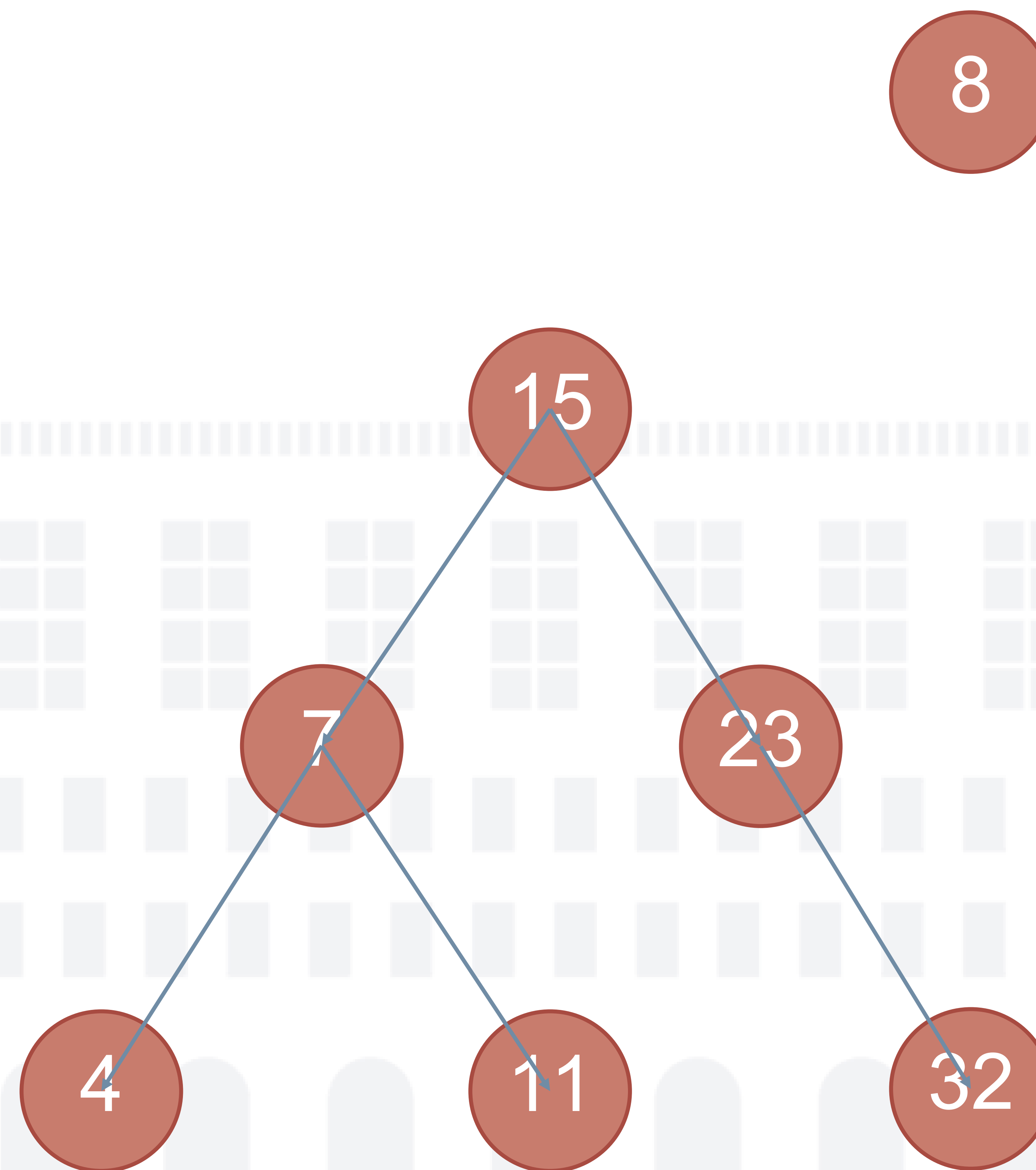
Supponiamo di dover proseguire a sinistra (k minore del valore nella radice). Abbiamo *due casi*:

- **La radice non ha un figlio sinistro**: possiamo direttamente inserire k come figlio sinistro della radice
- **La radice ha un figlio sinistro**: chiamiamo ricorsivamente l'algoritmo di inserimento di un nodo nell'albero considerando l'albero che ha per radice il figlio sinistro

Simmetricamente se si prosegue a destra.

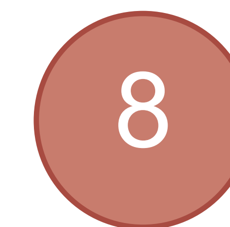
INSERIMENTO DI UN NUOVO NODO SU BST: ESEMPIO

Supponiamo di voler inserire un elemento di chiave 8

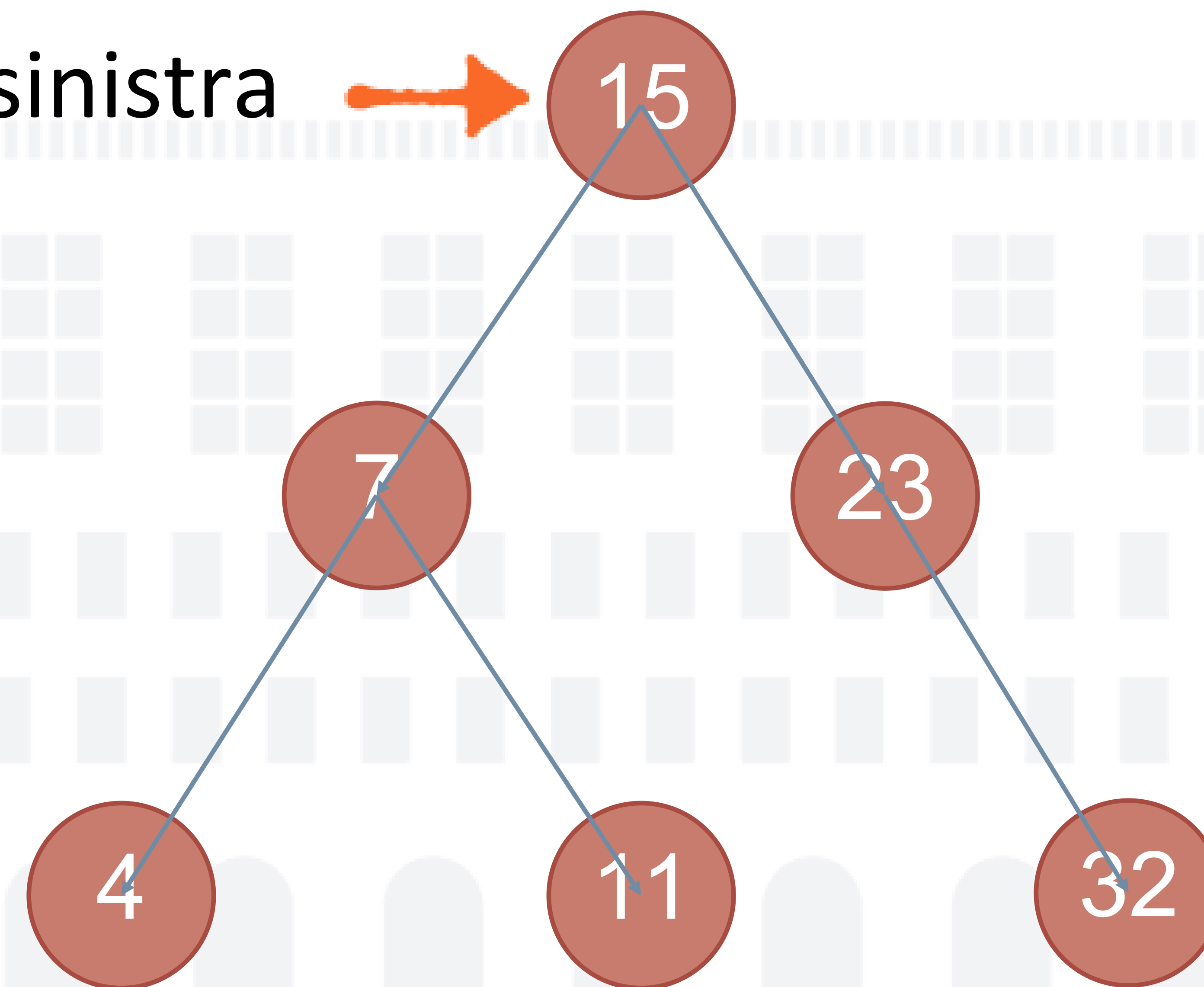


INSERIMENTO DI UN NUOVO NODO SU BST: ESEMPIO

Supponiamo di voler inserire un elemento di chiave 8



8 < 15, quindi ci muoviamo a sinistra



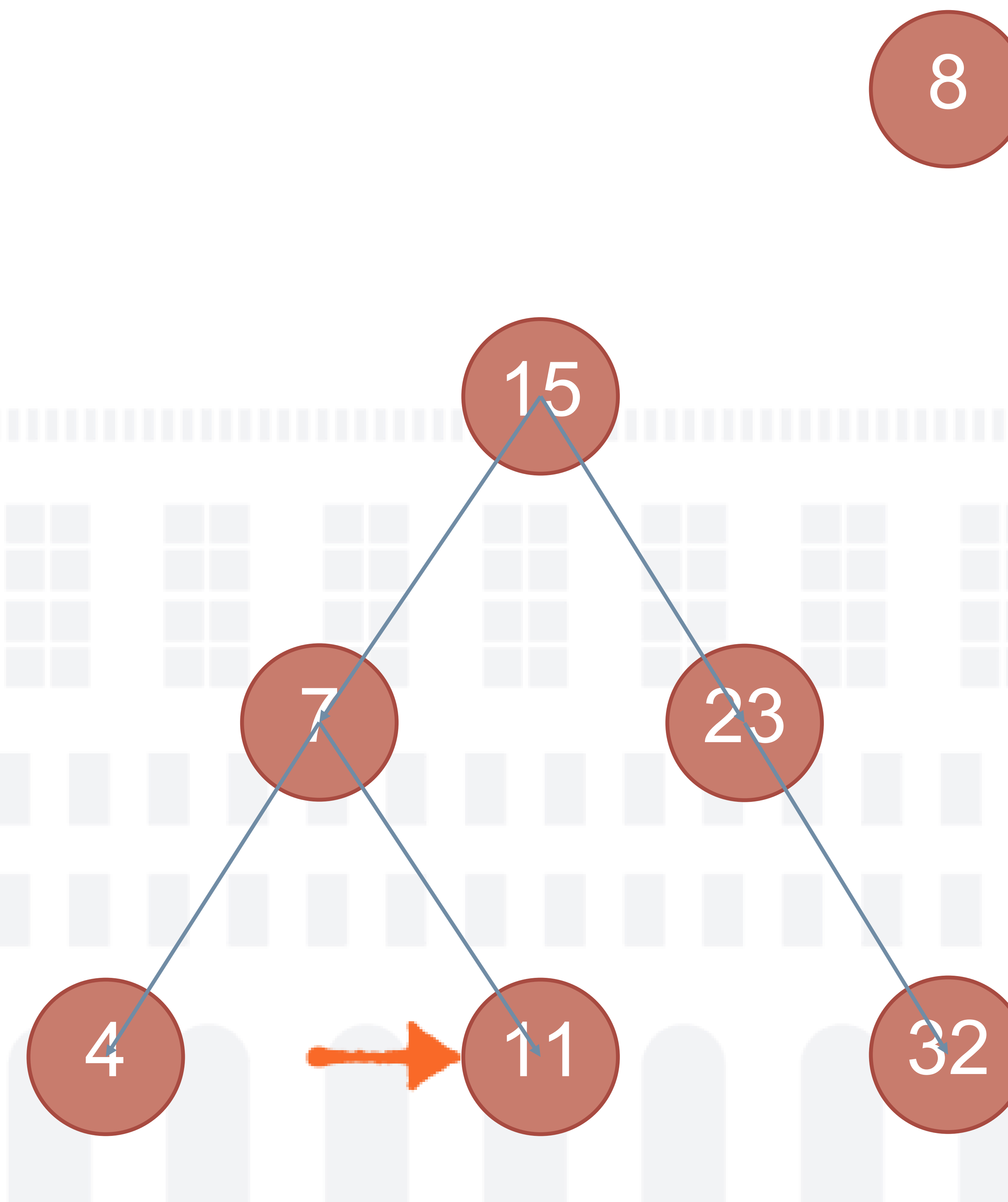
INSERIMENTO DI UN NUOVO NODO SU BST: ESEMPIO

Supponiamo di voler inserire un elemento di chiave 8

8 > 7, quindi ci muoviamo a destra

INSERIMENTO DI UN NUOVO NODO SU BST: ESEMPIO

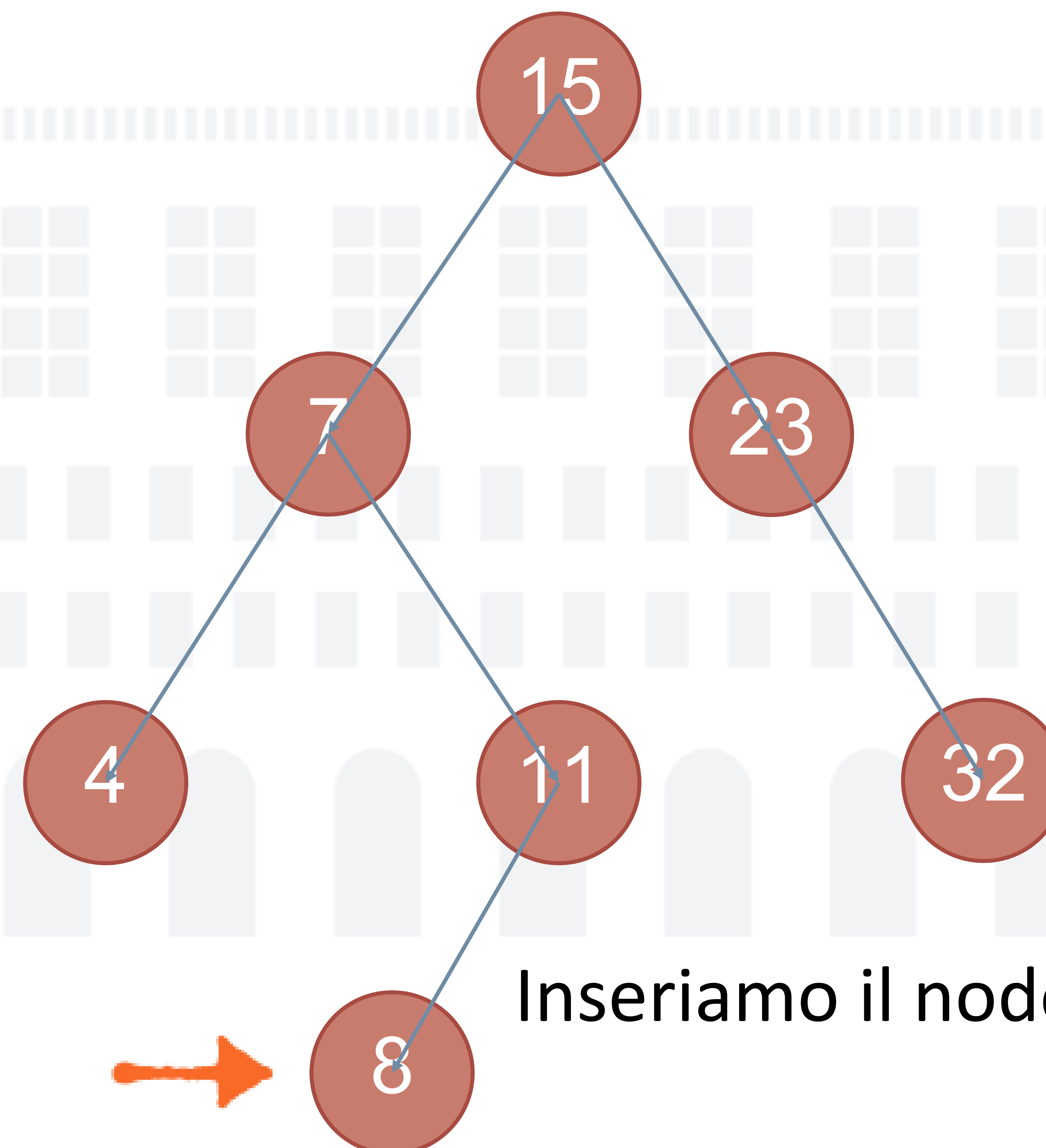
Supponiamo di voler inserire un elemento di chiave 8



8 < 11, ci dovremmo muovere a sinistra,
ma il nodo non ha un figlio sinistro

INSERIMENTO DI UN NUOVO NODO SU BST: ESEMPIO

Supponiamo di voler inserire un elemento di chiave 8



Inseriamo il nodo “8” come figlio sinistro di “11”

INSERIMENTO DI UN NUOVO NODO SU BST: PSEUDOCODICE

```
Parametri: Nodo radice, key
if radice is None
   return Nodo(key) # l'albero è vuoto, ritorniamo un nuovo albero
if key < radice.key
   if radice.left is None # figlio sinistro libero per l'inserimento
      radice.left = nuovo_nodo(key)
   else # chiamata ricorsiva sul sottoalbero sinistro
      inserisci(radice.left, key)
else # ovvero key ≥ radice.key
   if radice.right is None # figlio destro libero per l'inserimento
      radice.right = nuovo_nodo(key)
   else # chiamata ricorsiva sul sottoalbero sinistro
      inserisci(radice.right, key)
```

UNIVERSITÀ DEGLI STUDI DI TRIESTE

INSERIMENTO DI UN NUOVO NODO SU BST: COMPLESSITA’

Prima di effettuare una chiamata ricorsiva effettuiamo **un numero costante di passi**.

Ogni chiamata ricorsiva ci fa scendere di un livello nell’albero.

Quindi la complessità dell’inserimento dipende, *come per la ricerca*, dalla profondità dell’albero: $\boldsymbol{O}(\boldsymbol{h})$

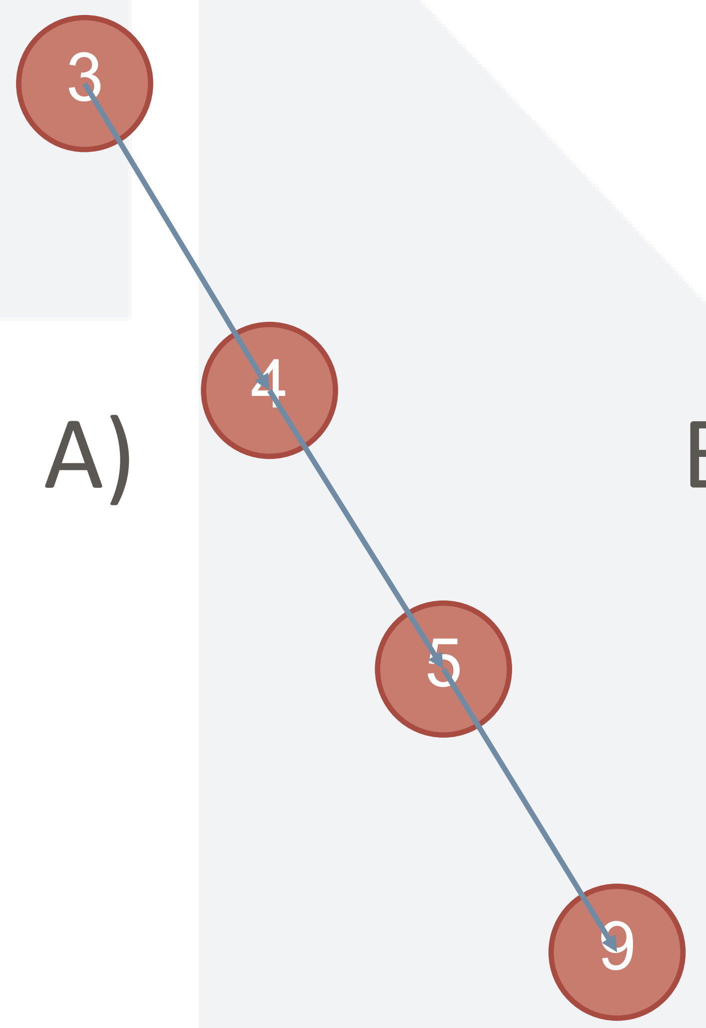
QUIZ TIME!

Supponiamo di inserire in un albero binario di ricerca inizialmente vuoto i seguenti valori nell'ordine in cui appaiono:

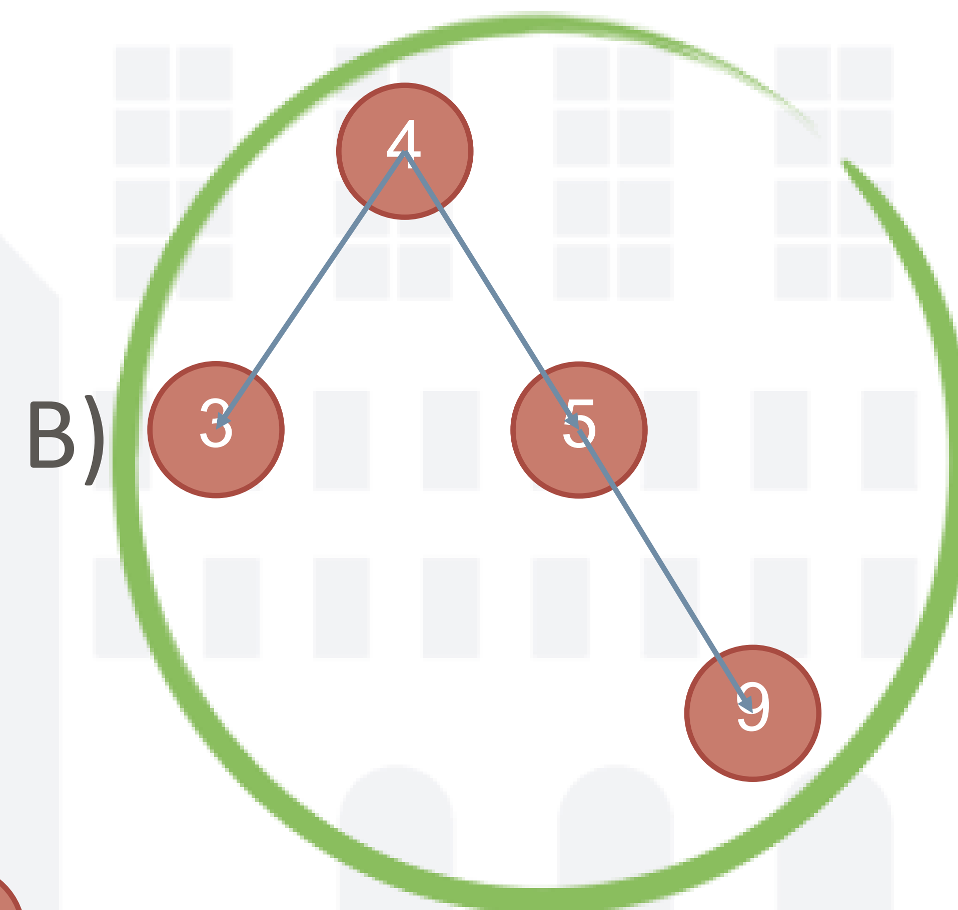
4 5 9 3

Quale di questi è l'albero risultante?

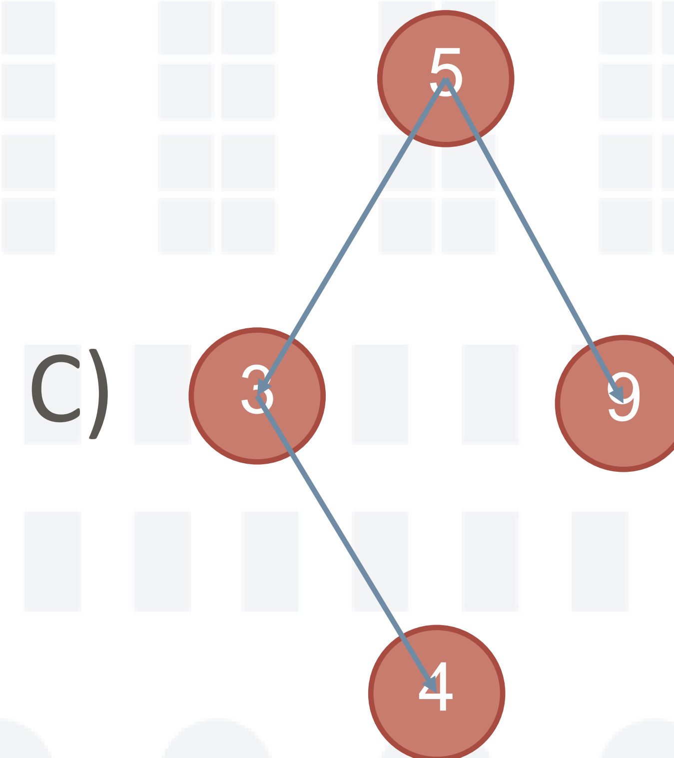
A)



B)



C)



D)

IDENTIFICAZIONE DEL SUCCESSORE/PREDECESSORE DI UN NODO IN BST

Ricerca del **successore** di un nodo identificato con la sua chiave k. La ricerca del predecessore è equivalente e simmetrica.

Caso semplice: il nodo ha un **figlio destro**

- Allora la chiave più piccola strettamente più grande di k è il minimo del sottoalbero avente radice il figlio destro e quindi sarà il successore di k.

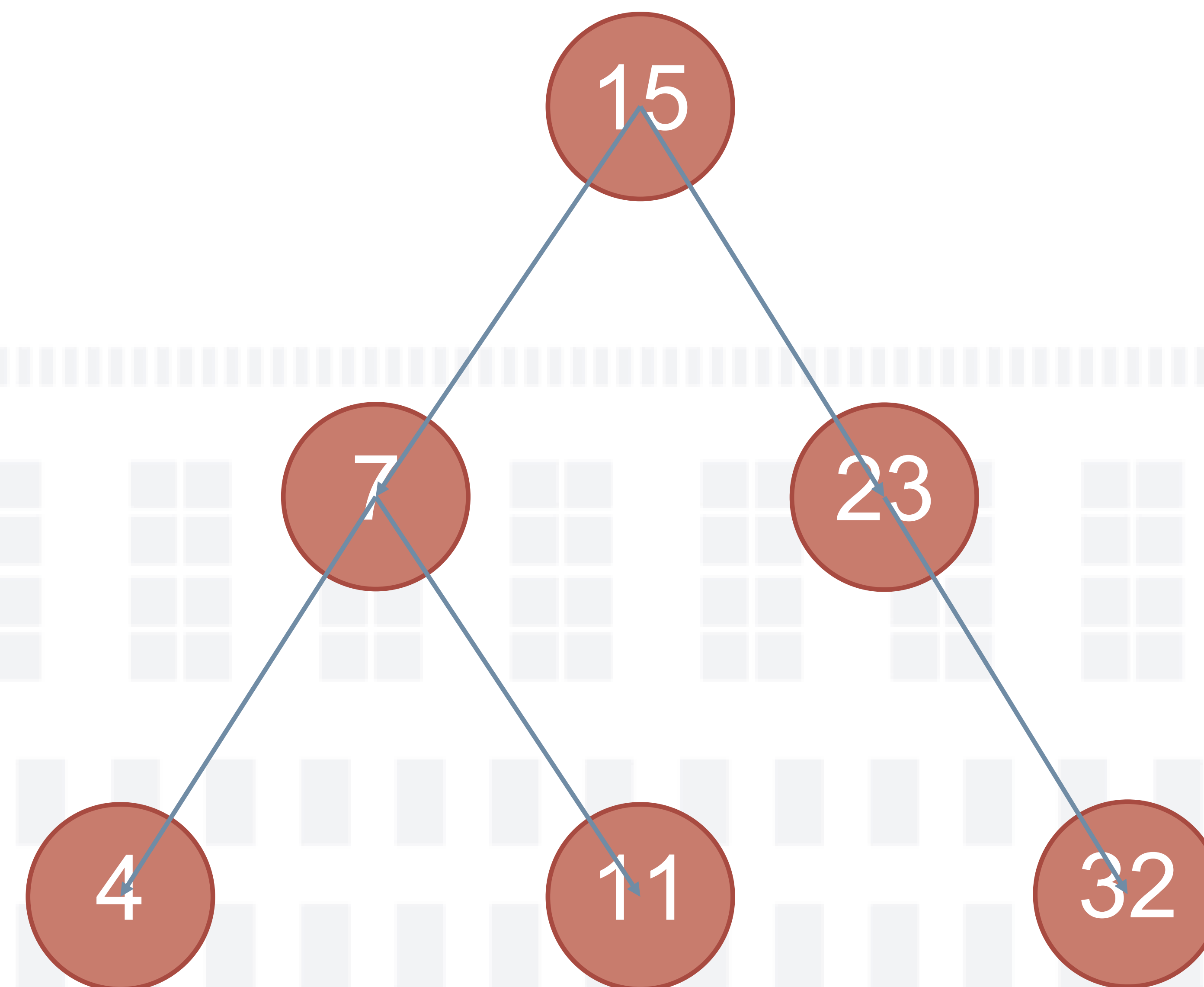
Caso difficile: il nodo non ha figlio destro

- Allora dobbiamo risalire nella catena di genitori

ATTENZIONE: ipotizziamo non ci siano chiavi doppie.

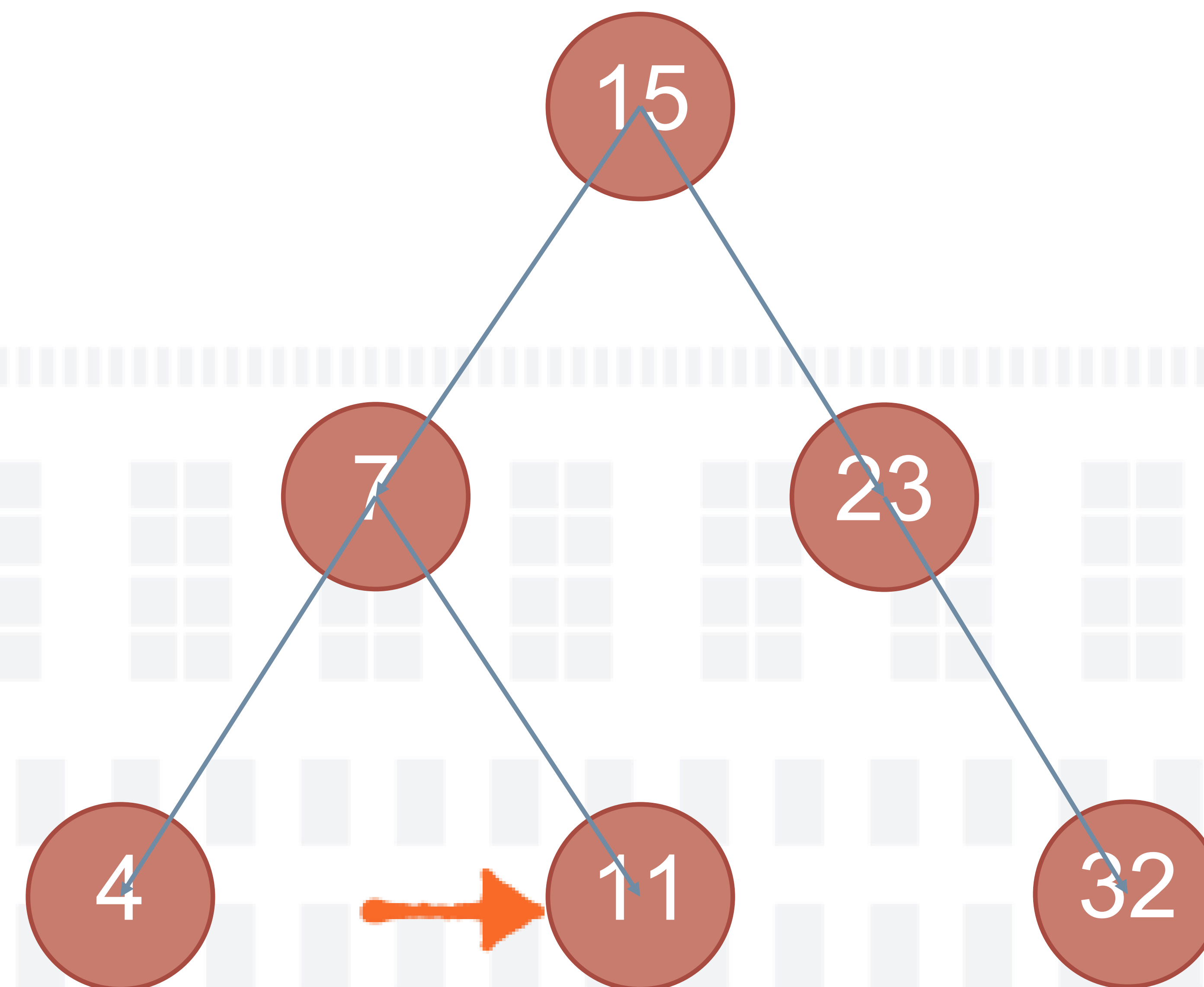
IDENTIFICAZIONE SUCCESSORE IN BST: ESEMPIO

Supponiamo di voler trovare il successore di “11”



IDENTIFICAZIONE SUCCESSORE IN BST: ESEMPIO

Supponiamo di voler trovare il successore di “11”

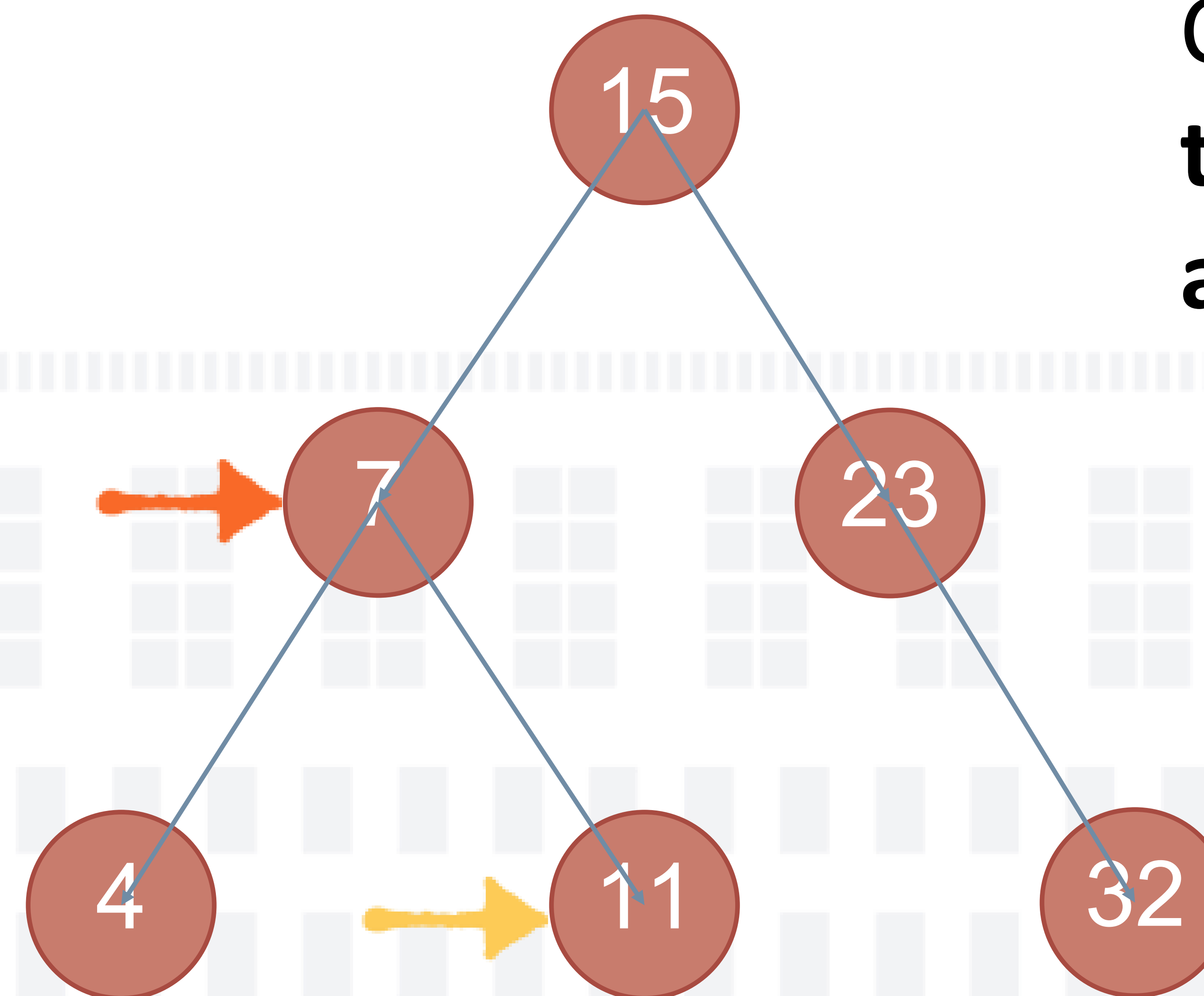


“11” non ha un figlio destro, quindi cerchiamo tra i genitori

IDENTIFICAZIONE SUCCESSORE IN BST: ESEMPIO

Supponiamo di voler trovare il successore di “11”

“7” è il genitore di “11”, ma è minore, dato che proveniamo dal figlio destro

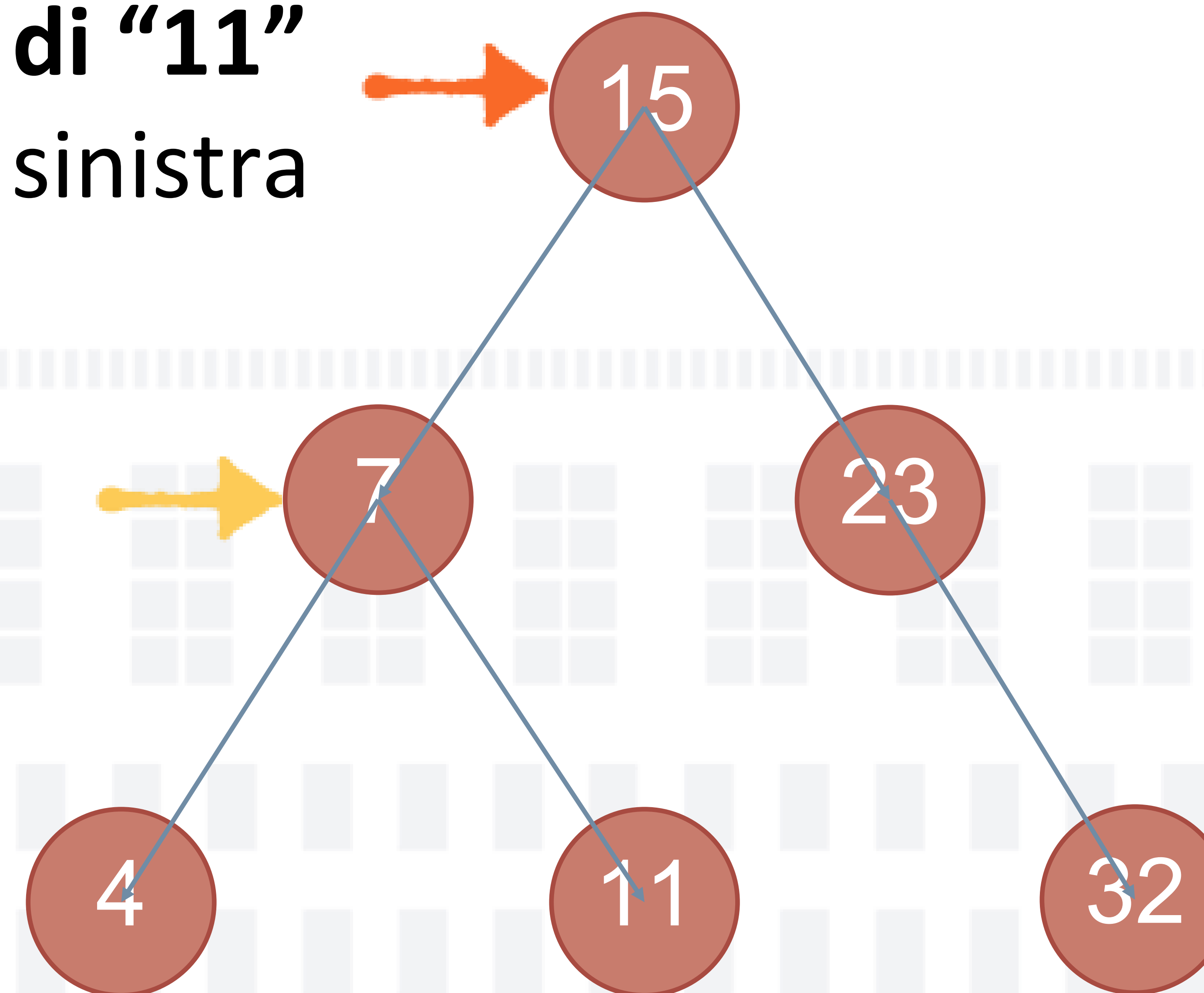


Continuiamo a risalire **finché non troviamo un genitore al quale arriviamo dal figlio di sinistra**

IDENTIFICAZIONE SUCCESSORE IN BST: ESEMPIO

Supponiamo di voler trovare il successore di “11”

Abbiamo trovato **il successore di “11”** dato che arriviamo dal figlio di sinistra



IDENTIFICAZIONE SUCCESSORE IN BST: PSEUDOCODICE

```
Parametri: Nodo
if nodo.right is not None:  # Se il nodo ha un figlio destro
   return minimo(nodo.right) # successore= minimo del sottoalbero destro
                             # cioè nodo più a sinistra tra quelli maggiori di nodo
x = nodo
y = nodo.parent
while y is not None and y.right is x  # finché esiste un genitore
                                      # e proveniamo dal figlio di destra
   x = x.parent                       # saliamo di un livello
   y = y.parent
return y
```

Se il nodo non ha figlio destro, bisogna risalire verso i genitori finché si trova un genitore y per il quale il nodo x è nel suo sottoalbero sinistro. Quel genitore è il successore. Se y diventa None, significa che il nodo era il massimo dell'albero, quindi non ha successore.

IDENTIFICAZIONE SUCCESSORE IN BST: COMPLESSITA'

La **complessità** di trovare il successore può essere divisa in *due casi*:

- Uguale alla complessità di trovare il minimo, se esiste un figlio destro
- Se il figlio destro non esiste, dobbiamo risalire l'albero… ma il numero di volte che risaliamo è comunque limitato dall'altezza dell'albero.

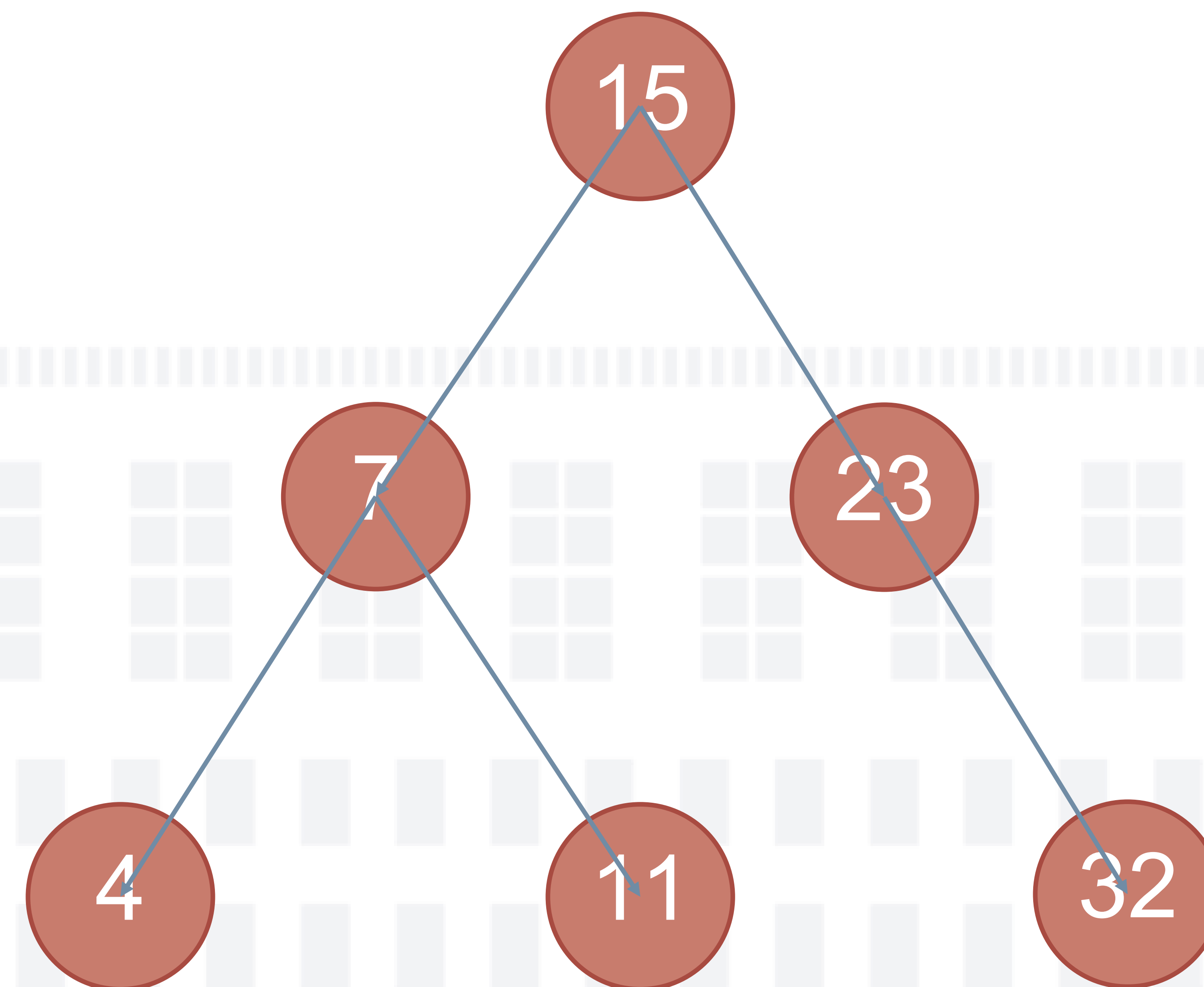
Quindi trovare il successore richiede al più un tempo $\boldsymbol{O}(\boldsymbol{h})$.

CANCELLAZIONE DI UN NODO SU BST: ALGORITMO (1/2)

- La rimozione di un elemento è l’operazione più complessa
- Abbiamo tre casi a seconda del numero di figli che ha un nodo:
  - Il nodo non ha figli: la rimozione è facile
  - Il nodo ha un solo figlio: facciamo prendere al figlio il posto del genitore
  - Il nodo ha due figli… questo è più complesso

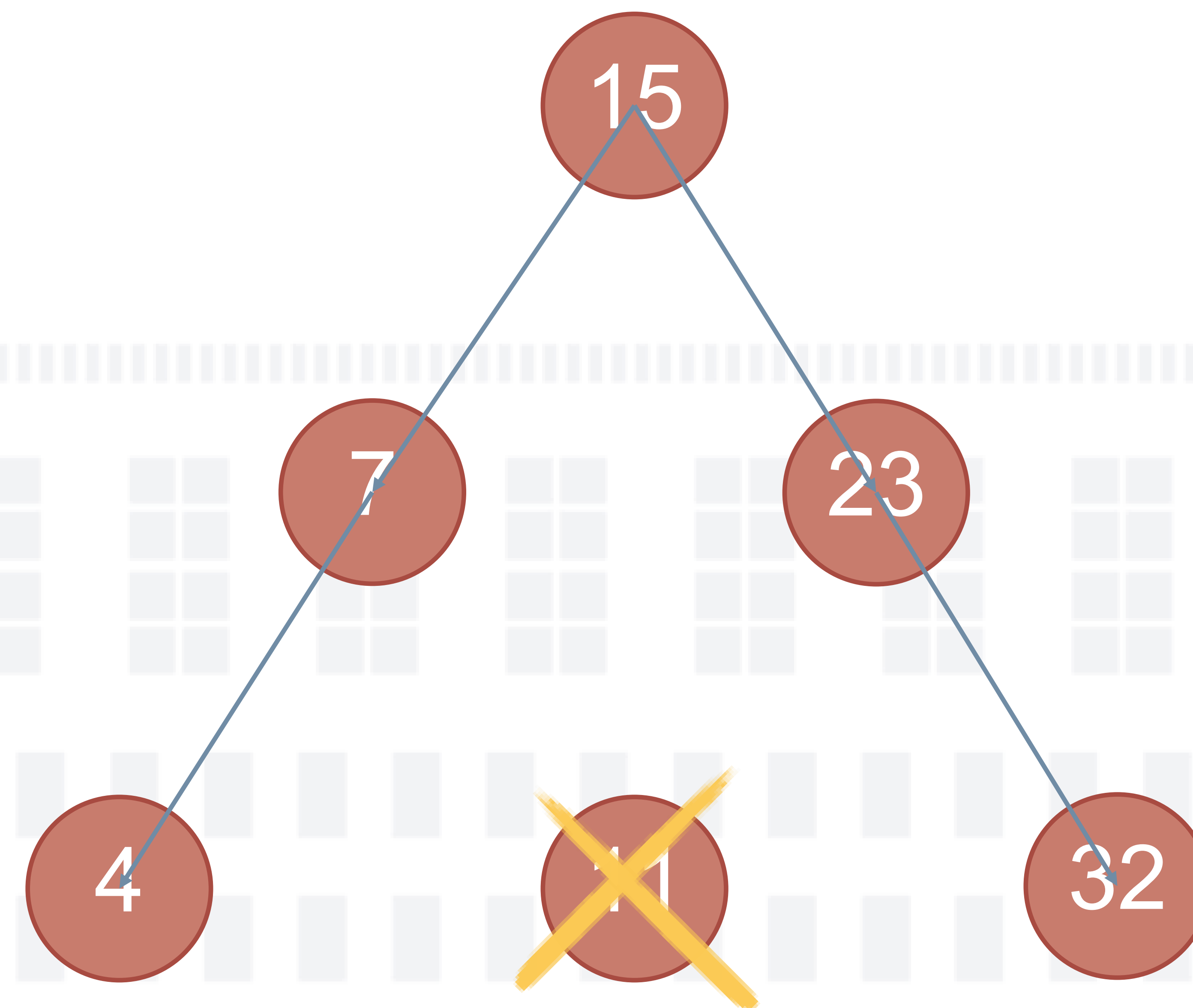
CANCELLAZIONE DI UN NODO SU BST: ESEMPIO

Supponiamo di voler cancellare “11”



CANCELLAZIONE DI UN NODO SU BST: ESEMPIO

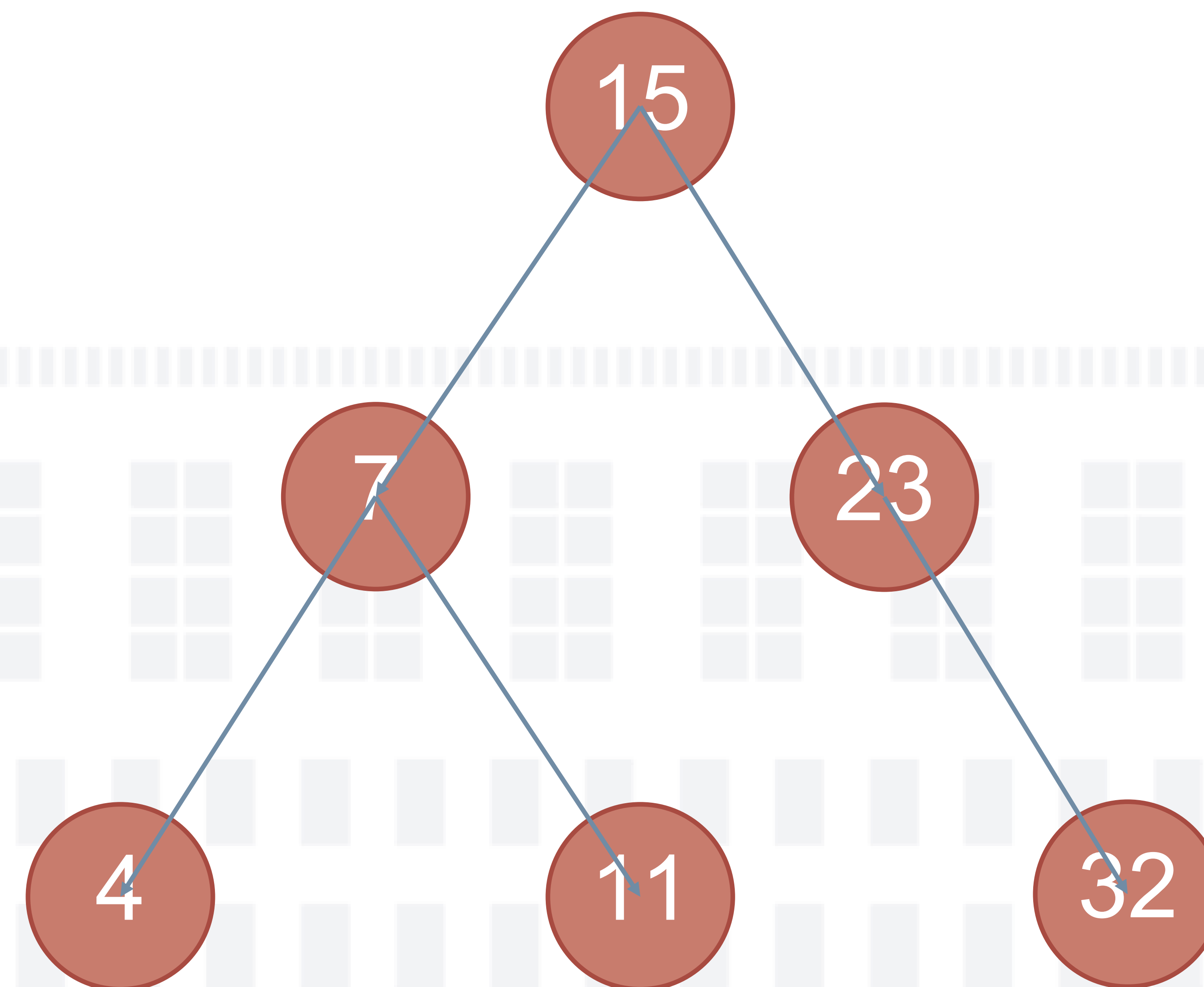
Supponiamo di voler cancellare “11”



“11” Non ha figli, quindi possiamo eliminarlo senza problemi

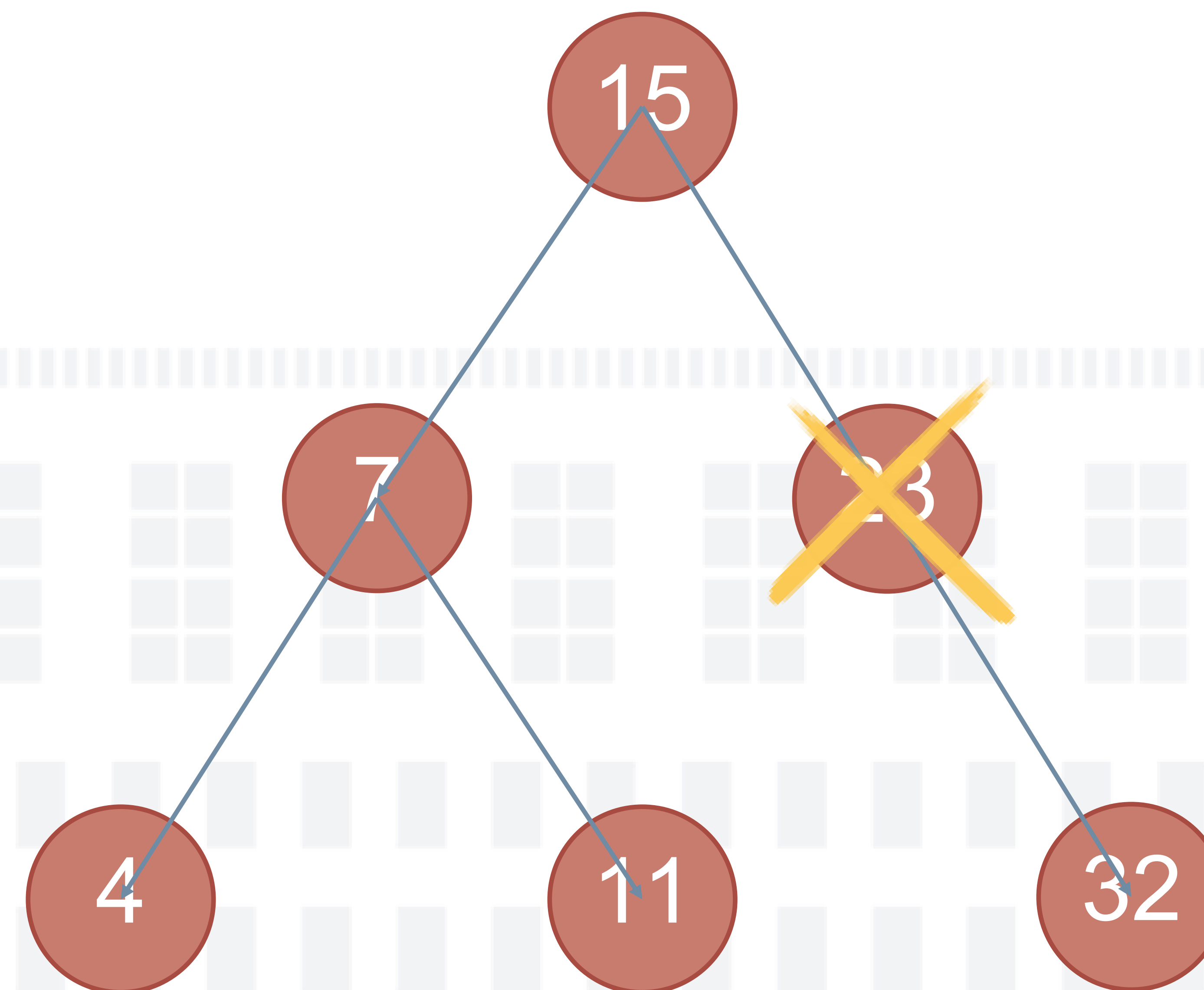
CANCELLAZIONE DI UN NODO SU BST: ESEMPIO

Supponiamo di voler cancellare “23”



CANCELLAZIONE DI UN NODO SU BST: ESEMPIO

Supponiamo di voler cancellare “23”



Avendo un solo figlio, possiamo rimpiazzare “23” col suo sottoalbero destro

UNIVERSITÀ DEGLI STUDI DI TRIESTE

CANCELLAZIONE DI UN NODO SU BST: ALGORITMO (2/2)

Per cancellare **nel caso di due figli** possiamo dividere la procedura in *due passi*:

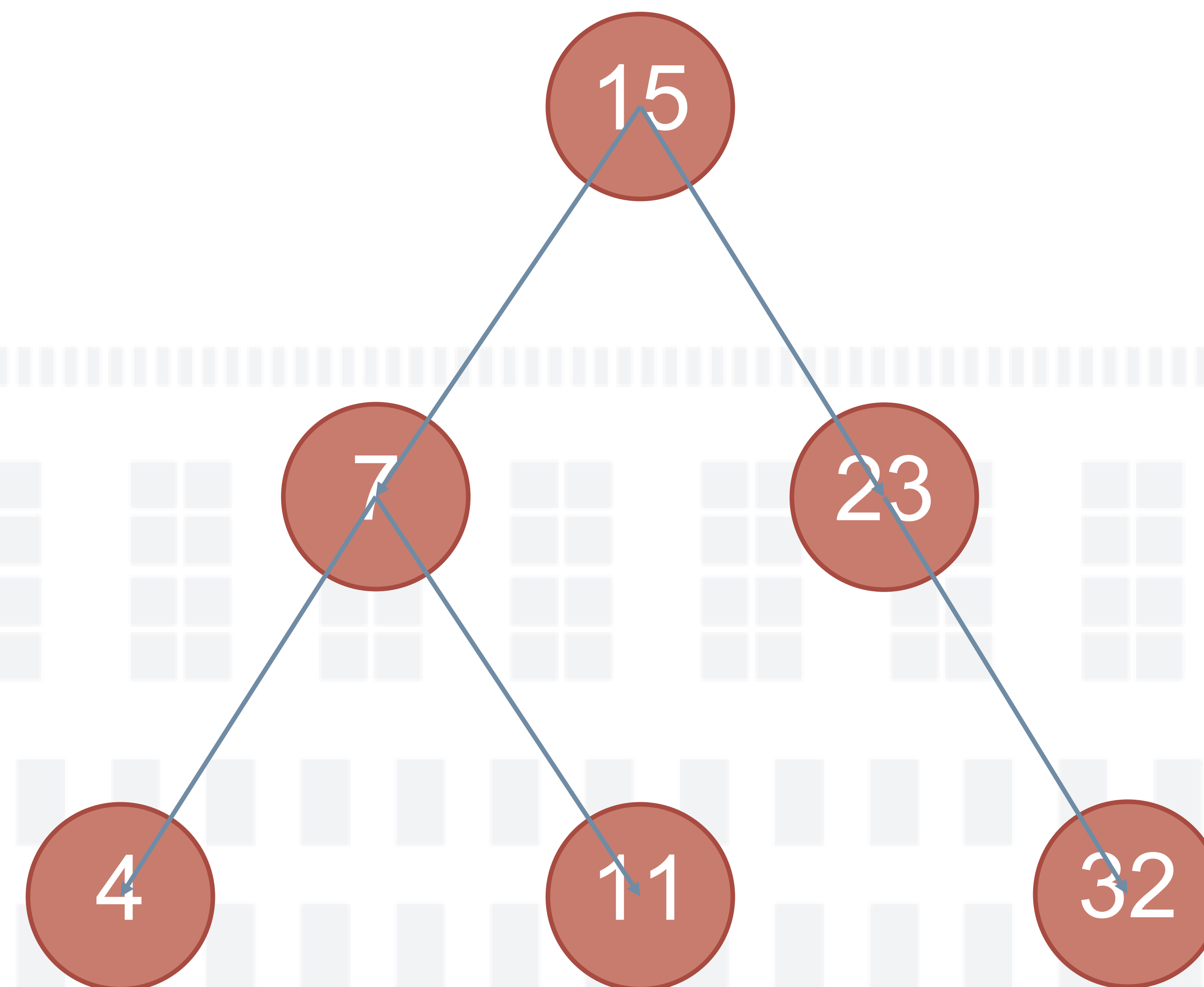
1. Mettiamo il predecessore del nodo da cancellare nel posto del nodo cancellato (*il predecessore è necessariamente nel sottoalbero sinistro**)
2. Dato che *il predecessore* è il massimo del sottoalbero di sinistra, *non può avere figlio destro*, quindi possiamo eliminarlo senza problemi

Equivalentemente si potrebbe usare il successore (e il sottoalbero di destra)

**Nota. In un BST tutti i nodi del sottoalbero sinistro hanno chiave minore del nodo corrente. Il predecessore è, per definizione, il nodo con la chiave più grande tra quelle minori del nodo corrente.*

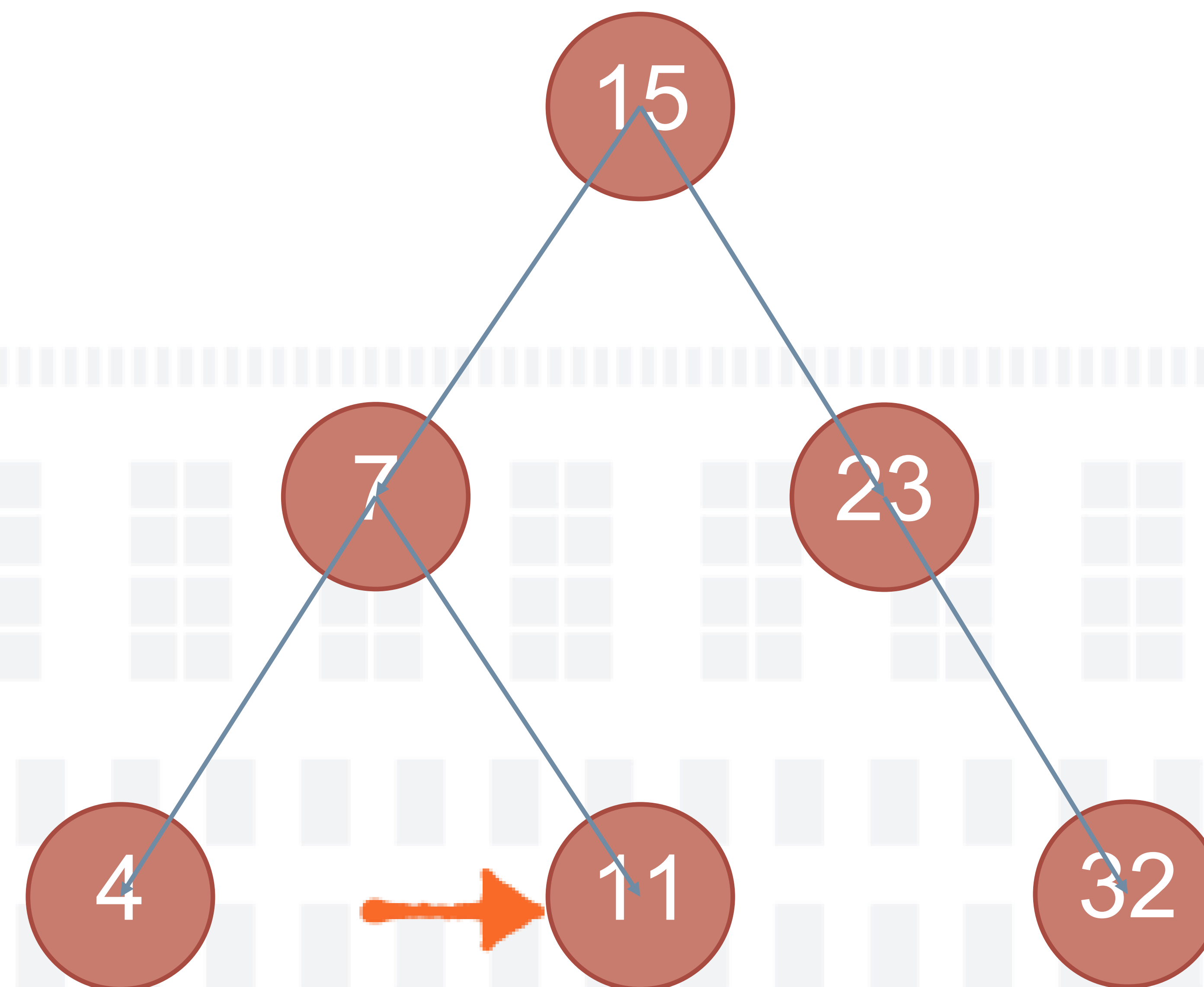
CANCELLAZIONE DI UN NODO SU BST: ESEMPIO

Supponiamo di voler cancellare “15”



CANCELLAZIONE DI UN NODO SU BST: ESEMPIO

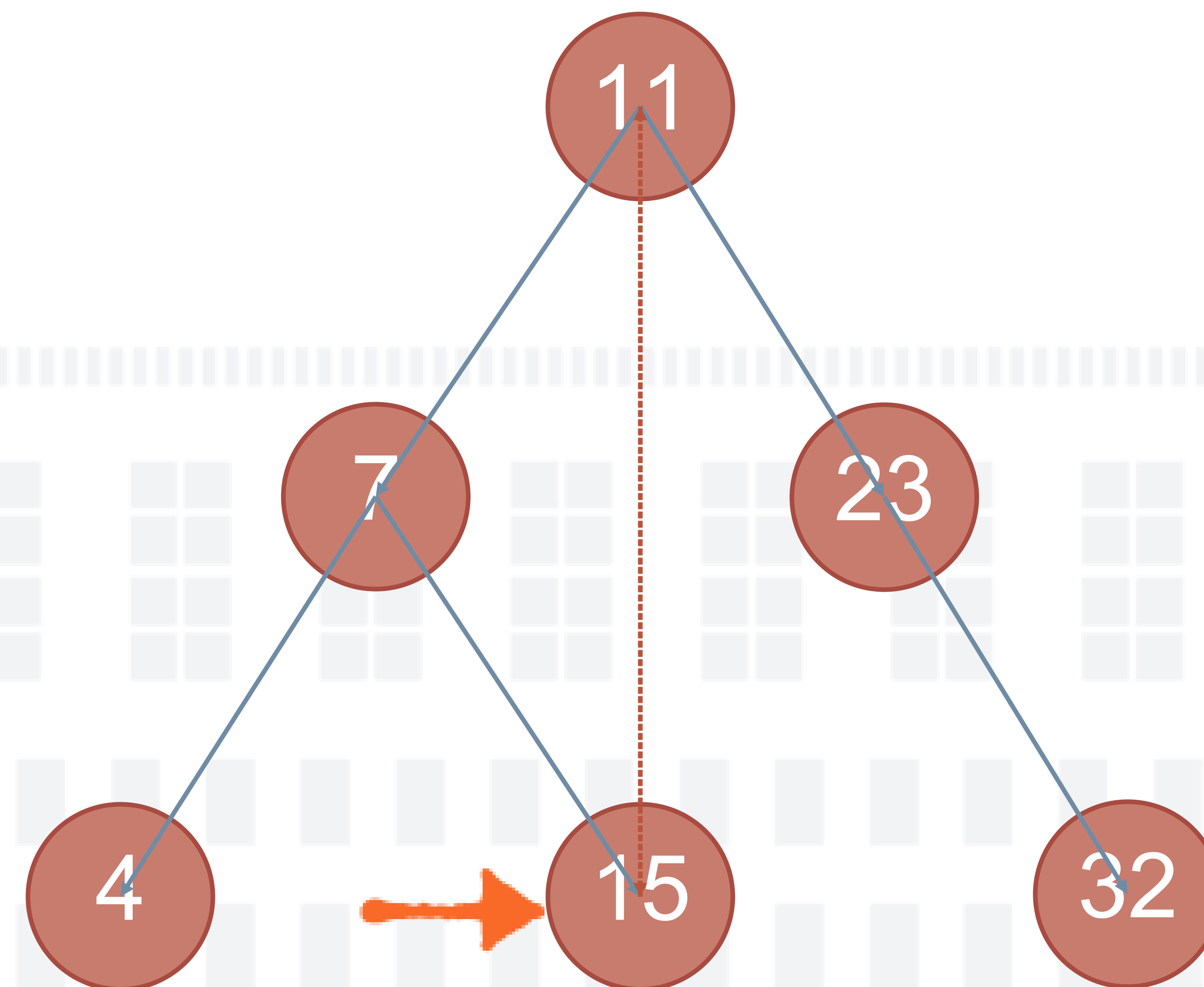
Supponiamo di voler cancellare “15”



Individuiamo **il predecessore di “15”**

CANCELLAZIONE DI UN NODO SU BST: ESEMPIO

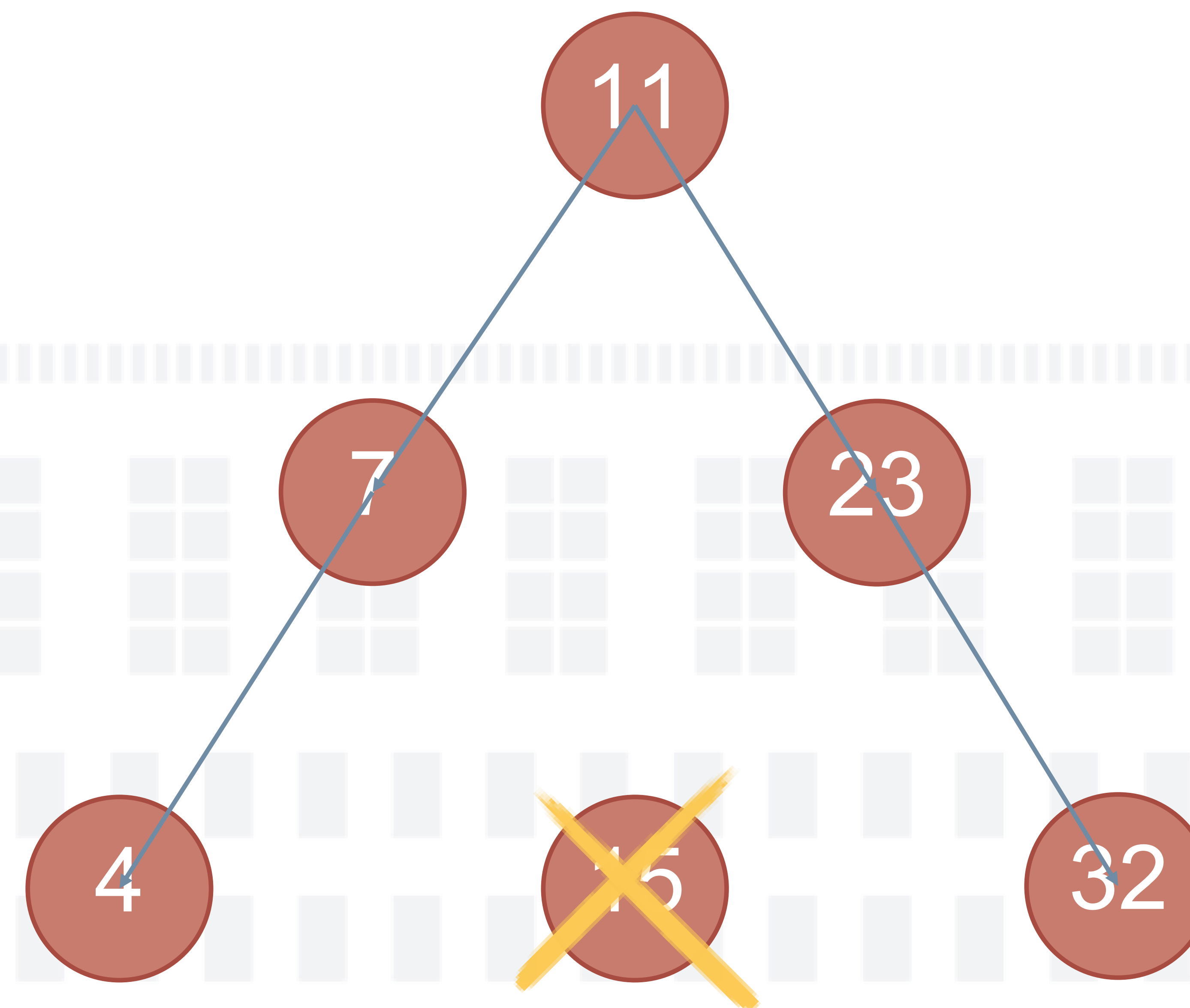
Supponiamo di voler cancellare “15”



Scambiamo di posto il nodo da cancellare ed il predecessore

CANCELLAZIONE DI UN NODO SU BST: ESEMPIO

Supponiamo di voler cancellare "15"



Eliminiamo il nodo

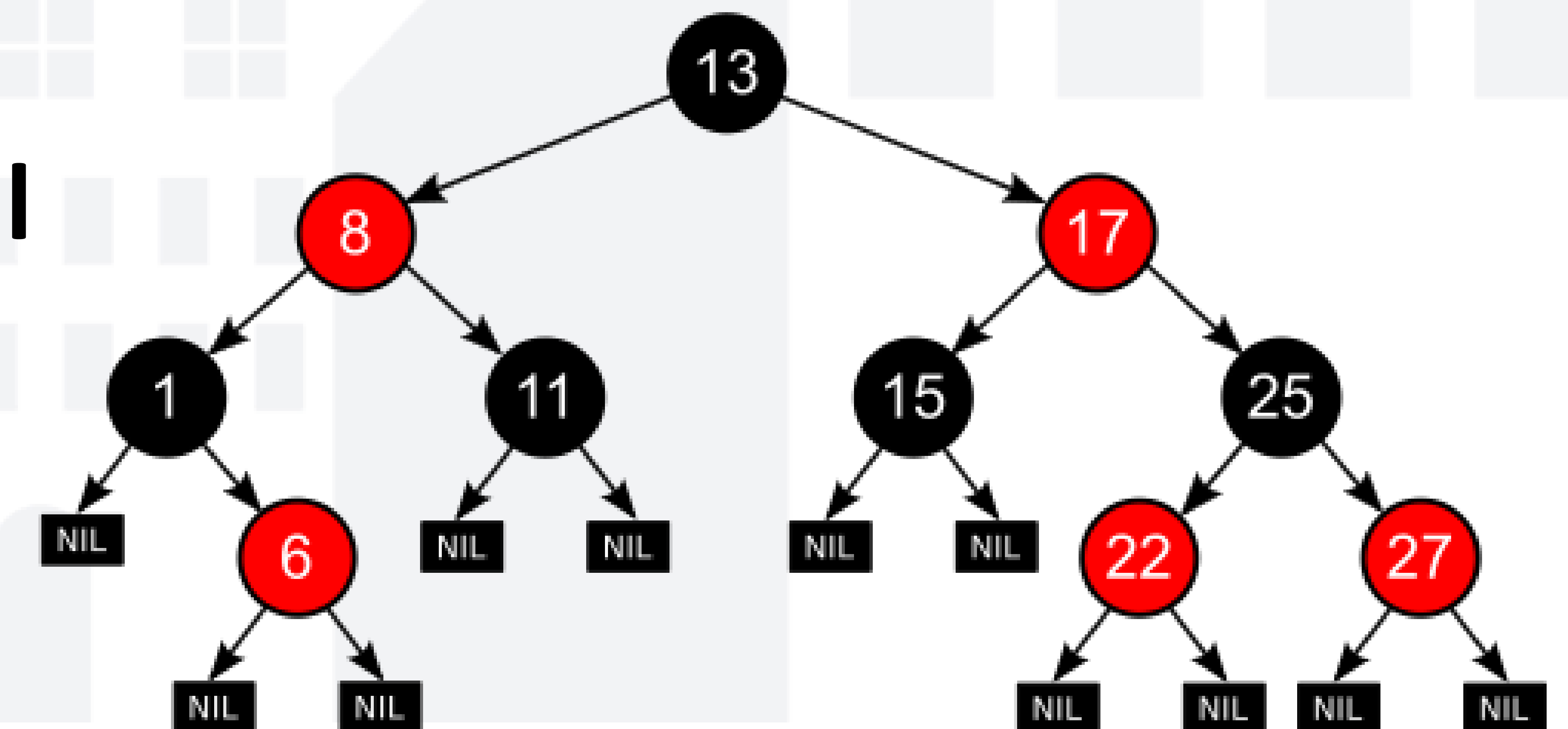
se avesse un figlio sinistro,
rientreremmo facilmente nel caso
«rimozione con 1 figlio»

CANCELLAZIONE DI UN NODO SU BST: COMPLESSITA'

- La **complessità** della cancellazione dipende dal tipo di rimozione che dobbiamo fare:
  - Rimuovere un nodo con zero o un figlio richiede tempo costante
  - Rimuovere un nodo con due figli richiede di trovare il nodo successore, che richiede tempo $O(h)$
- Anche in questo caso la complessità dipende dall'altezza

ALBERI BINARI: DISCUSSIONE

- Tutte le operazioni viste dipendono dall'altezza dell'albero
- A seconda dell'ordine in cui vengono inseriti i dati è possibile ottenere grandi differenze in termini di altezza:
  - Alberi bilanciati a profondità $\boldsymbol{O(logn)}$
  - Ma nei casi più sbilanciati la profondità è $\boldsymbol{O(n)}$
- **Esistono varianti di alberi in grado di mantenere il bilanciamento** (es., alberi rosso-neri)



Lista concatenata doppia (variante)

Ogni nodo ha 4 **attributi**:

- Dato x (`nodo.value, nodo.key`)
- Puntatore al figlio sinistro (`nodo.left`)
- Puntatore al figlio destro (`nodo.right`)
- *Puntantore al genitore (`nodo.parent`). Questo è opzionale, ma utile.*

- I nodi sono collegati tramite puntatori, **non contigui** come un array.

Alternative (meno usate):

- **Array**: memorizzando i nodi in posizioni calcolate (es. heap-style: sinistro = 2i, destro = 2i+1).
- **Hash table**: per rappresentazioni non strutturate.

IN SINTESI: COMPLESSITA' PER BST

OPERAZIONE	CASO PEGGIORE	CASO MEDIO PER ALBERO BINARIO	*Caso medio con albero bilanciato*
ACCEDERE AD UN ELEMENTO	$O(n)$	$O(h)$	$O(logn)$
RICERCA	$O(n)$	$O(h)$	$O(logn)$
INSERIMENTO	$O(n)$	$O(h)$	$O(logn)$
RIMOZIONE	$O(n)$	$O(h)$	$O(logn)$
SPAZIO OCCUPATO	$O(n)$	$O(n)$	$O(n)$

Nota. Lo spazio occupato in realtà sarebbe 3 × n, dove n = spazio occupato dal dato da memorizzare nel nodo + 2 puntatori ai due figli + riferimento al parent (opzionale). Tuttavia, rimane comunque O(n). *Come mai?*

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Materiale per la lezione

- Cormen et al. **CAP. 3.12**

Cosa significa davvero apprendere, o comprendere?

Negli esseri umani, la comprensione nasce dall'intreccio tra esperienza, conoscenze condivise, contesto educativo, curiosità personale e una componente biologica, evolutiva e genetica che ci accompagna fin dall'inizio. Le macchine seguono un'altra strada: apprendono dai dati, cercano regolarità, ottimizzano, riducono l'errore. Ma quello che osserviamo è davvero comprensione, o una sua sofisticata simulazione?

Durante la serata partiremo dalle basi: come "impara" una macchina? Che cosa significa, per un algoritmo, riconoscere un'immagine, classificare un segnale, trascrivere una voce o generare una frase? Attraverso esempi concreti vedremo che le reti neurali possono raggiungere risultati impressionanti, ma spesso usando strategie molto diverse dalle nostre. Anche i moderni modelli linguistici, come ChatGPT, Gemini o Claude, sanno produrre risposte convincenti e sorprendenti, ma possono sbagliare in modi che rivelano la distanza tra generare linguaggio e comprenderlo come lo intendiamo noi.

A questo punto il focus della domanda si sposta: se le macchine riconoscono oggetti, parole e testi, possono anche riconoscere qualcosa di noi? Possono "capirci" quando non riusciamo più a comunicare nel modo abituale? Da qui entreremo nel territorio in cui neuroscienze, intelligenza artificiale e tecnologia si incontrano: quello delle Brain-Computer Interface, sistemi pensati per trasformare l'attività cerebrale in un possibile canale di comunicazione, ad esempio per persone con gravi difficoltà motorie o linguistiche.

Questo percorso ci porterà a osservare non solo le potenzialità dell'AI, ma anche i suoi limiti quando incontra la complessità del cervello umano. E forse ci aiuterà a riflettere su ciò che rende unico il nostro modo di apprendere, comprendere e comunicare.