



Université  
de Lille



**FACULTÉ  
DES SCIENCES ET  
TECHNOLOGIES**  
Département Informatique

# UE - Projet – L3 – S2

Chaabane Djeraba

2023-2024



2 options possibles



Arbre-B



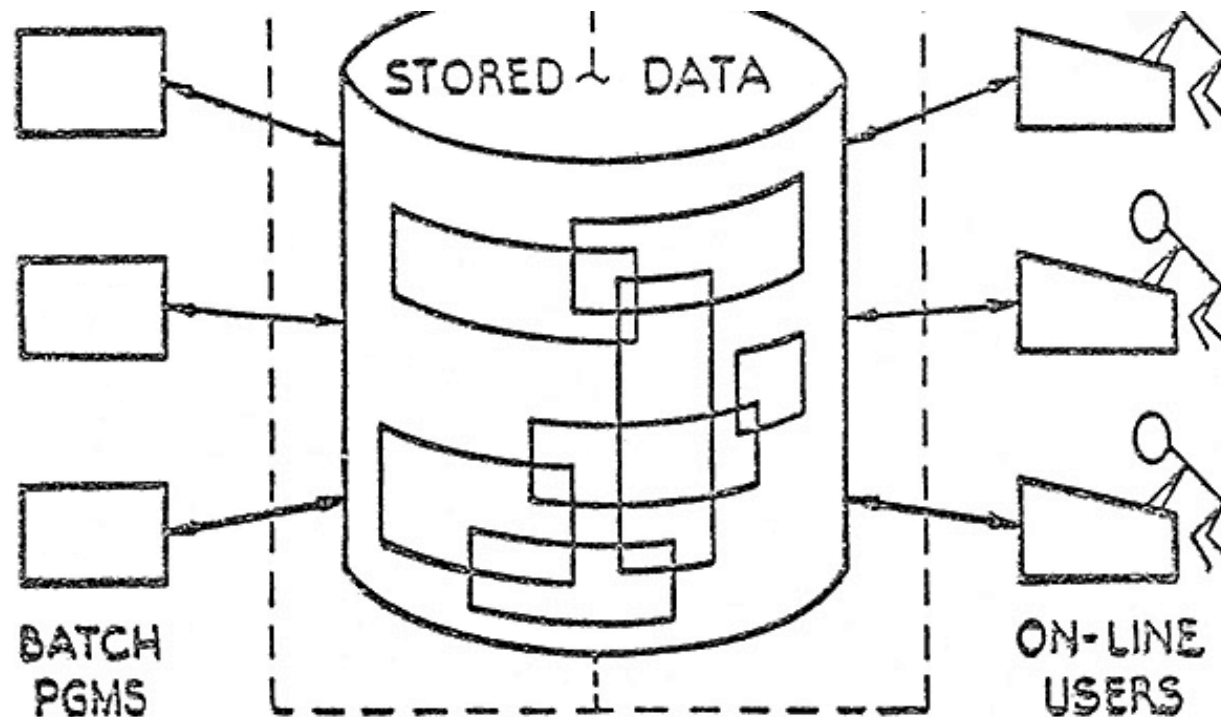
Projet Libre





# Structure de données : Arbre-B

- Stockage d'un volume important de données
- Versions ; B-Tree, B+Tree, B\*Tree



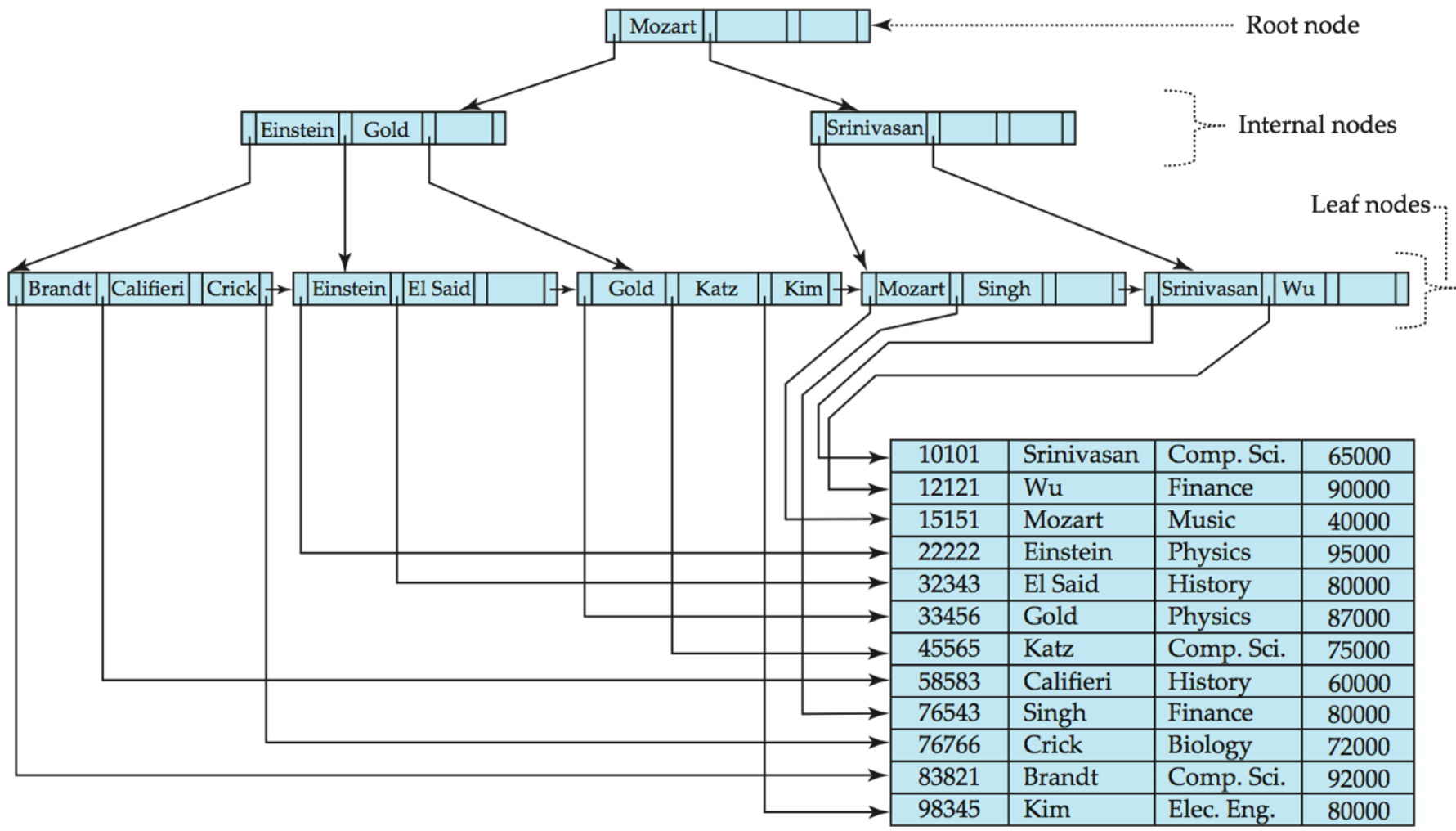
# Structure de données : Arbre-B



- Arbre n-aire équilibré. Majorité des arbres balancés (AVL et Red-Black) sont en MC. Arbre-B stocke les données en MS. Populaire dans les bases de données. Beaucoup de variantes
- Inventé par Rudolf Bayer et Edward Meyers McCreight, 1970, Mathematical and Information Sciences Laboratory BOEING SCIENTIFIC RESEARCH LABORATORIES July 1970
- Bayer, R.; McCreight, E.M. (1972), "[Organization and maintenance of large ordered indexes](#)" (PDF), Acta Informatica, 1 (3): 173 189, doi: [10.1007/bf00288683](#), S2CID 29859053, retrieved 2010-09-02



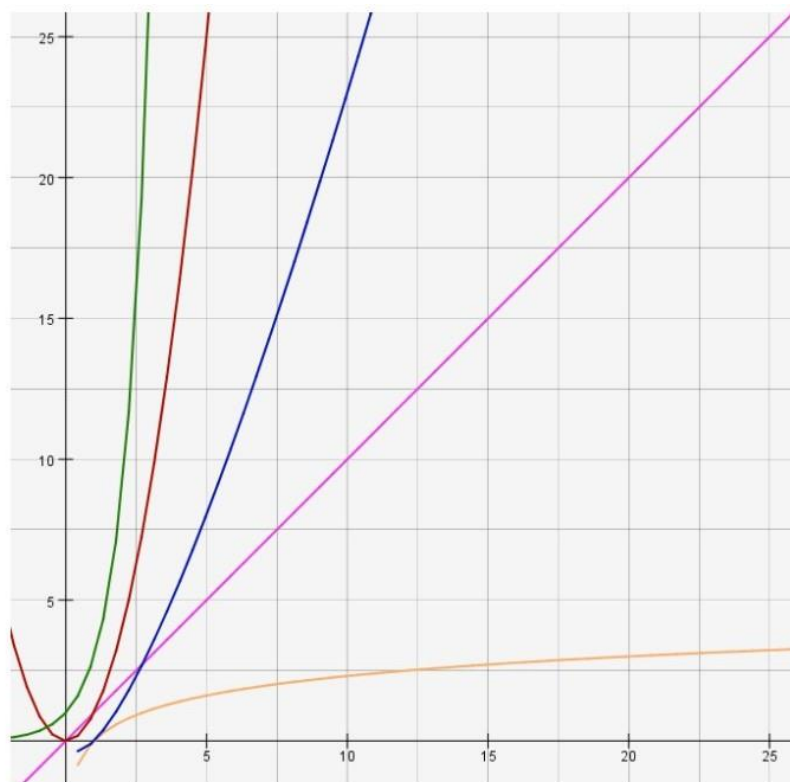
# Structure de données : Arbre-B





# Structure de données : Arbre-B

- Complexité temporelle : Recherche, insertion et suppression en  $\log(n)$ .  $n$  : nombre de clés.



Complexités classiques  
(ordre de grandeur)

$$\left\{ \begin{array}{l} O(e^n) \\ O(n^2) \\ O(n \log(n)) \\ O(n) \\ O(\log(n)) \end{array} \right.$$



# Projet Libre

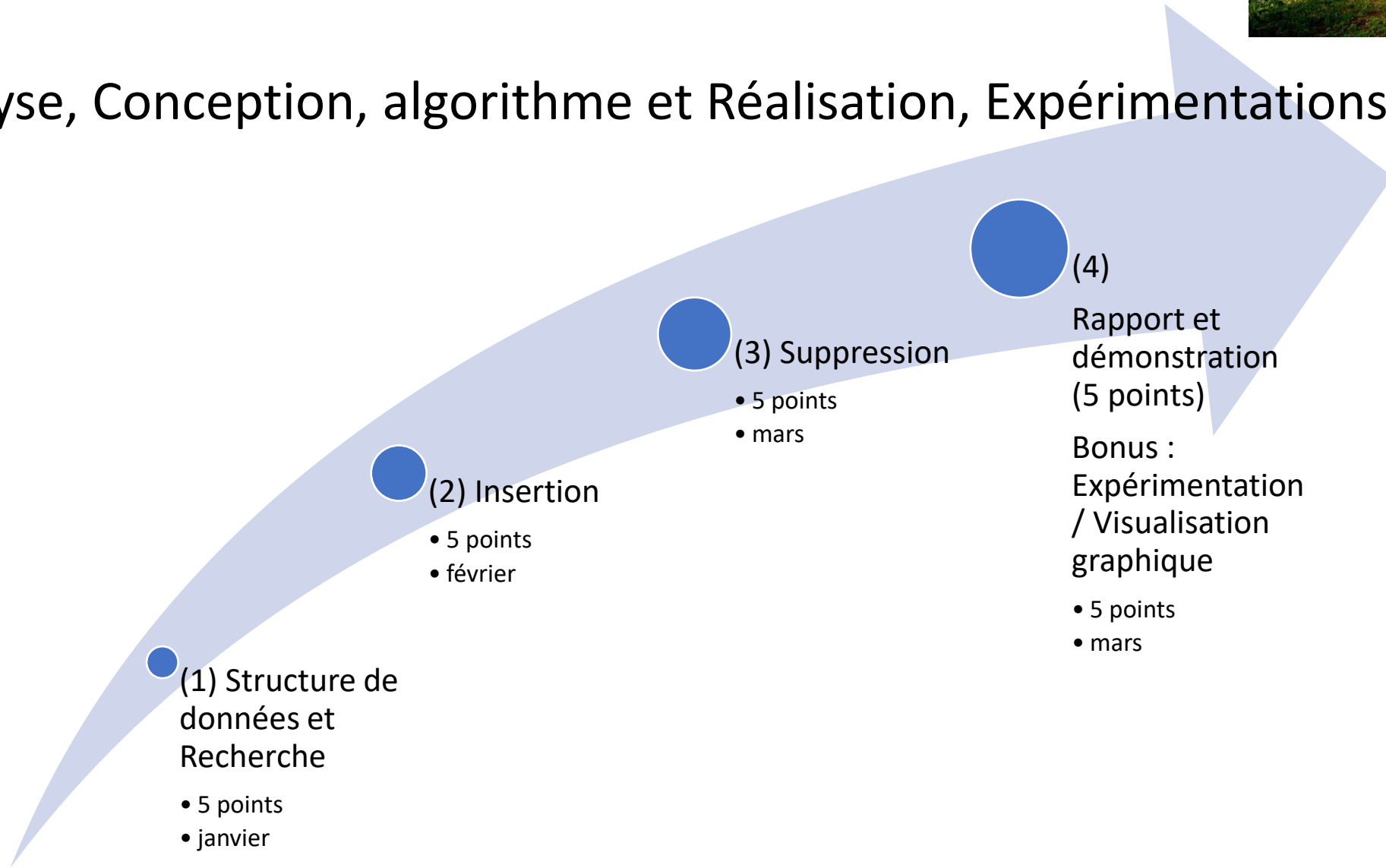


- Il est possible de choisir un projet différent. Le sujet en question doit être proposé et validé par l'enseignant responsable du groupe, avant le 11 janvier 2024. *1/3 des projets d'un groupe sont libres.*
- Contraintes : Concepts à objets, Python, algorithmie, structure de données, tests/validation
- Exemples : Jeux Bionus en ligne, Pseudo Fight, Jeux interface web, Jeux du Donjon, Puissance 4, Générateur de termes (mot, phrases), Quizz musical, Détection de faces avec IA, Classification d'images, Jeux de réflexes.

# Travail à réaliser



- Analyse, Conception, algorithme et Réalisation, Expérimentations





# Travail à réaliser



- Conception du schéma en UML
  - Structures de données, selon la technologie objet
- Structures de données et des algorithmes
- Codage en python
- Expérimentations
  - Réalisation des protocoles de tests
  - Optimisation de la valeur de k (nombre de clés maximum) (Bonus)
  - Mesure des performances selon plusieurs scénarios (Bonus)

# Méthodologie



- Les travail réalisé doit être téléversé sur Gitlab, chaque semaine.
- Jalons : semaine du 22 janvier (recherche et STD), 26 février (insertion), 25 mars (suppression).
- Travail en monôme ou en binôme.
- Avant de se lancer dans la réalisation et le développement des algorithmes du projet, il est nécessaire de prendre le temps de le comprendre et de le dérouler sur des données tests. Ensuite, organiser et développer l'architecture en technologie objet du projet (classes et liens entre les classes).

# Livrable



- Livrable finale = Démonstration + Code (Gitlab) + Documentation (Gitlab), à la fin de chaque séance aux cours des deux dernières séances du projet.
- La démonstration est réalisée sur le poste de travail, sur une durée de 30 minutes environ, avec les expérimentations.
- La documentation ne doit pas dépasser 10 pages : architecture générale, algorithmes, complexité temporelle, problèmes rencontrés, points à améliorer.

# Organisation



- <https://www.fil.univ-lille.fr/~aubert/l3/intervenantL3S6/>
- Cours (1h30/séance – 4,5 h) auront lieu :
  - mardi 9 janvier, 13 h - 14 h 30, amphi BACCHUS, M5
  - lundi 15 janvier, amphi BACCHUS, M5
  - lundi 22 janvier 16 h 30 – 18 h, amphi BACCHUS, M5
- TP (3,5 h / séance, 42 h TP / 12 séances)
  - Chaabane Djeraba : G2 (mardi après-midi, A11-M5) et G3 (mercredi matin, A11-M5)
  - Damien Pollet : G4 (mercredi matin A12-M5)
  - Pierre Allegrau : G5 (jeudi matin, A11-M5)
  - Jean-Luc Intumwayase : G6 (mercredi matin, A13 - M5)
  - Salman Farhat : G1 (jeudi matin, M4-A12)
  - jeudi ou mercredi matin : 8H-11h30 et mardi après-midi : 14:45 - 18:15



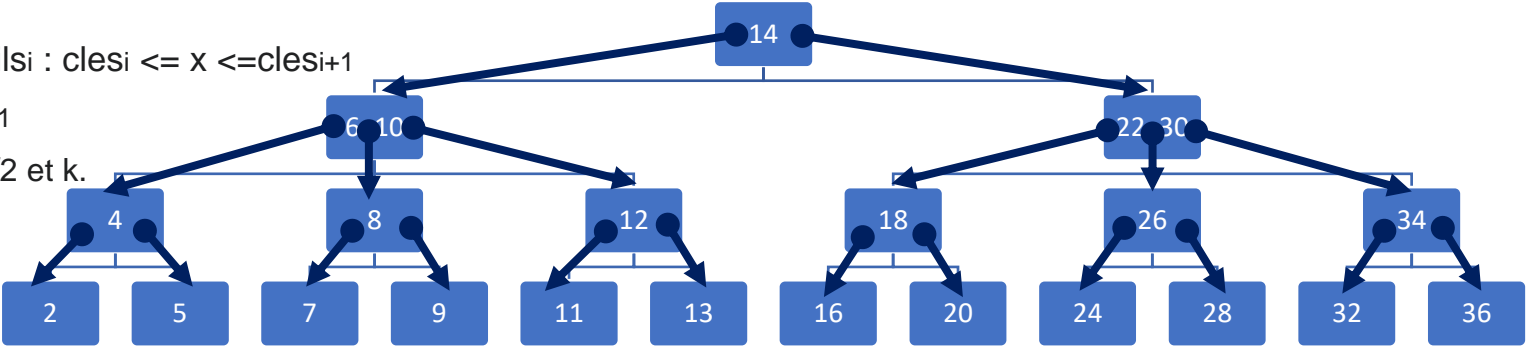


# Définition

- Arbre-B (k) ou Arbre-B(L, U) :
  - k = nombre de clés max par nœud. Nombre de clés min =  $k/2$ .
  - U est le nombre de nœuds fils max et L le nombre de nœuds fils min.  $L=k/2+1$  et  $U=k+1$



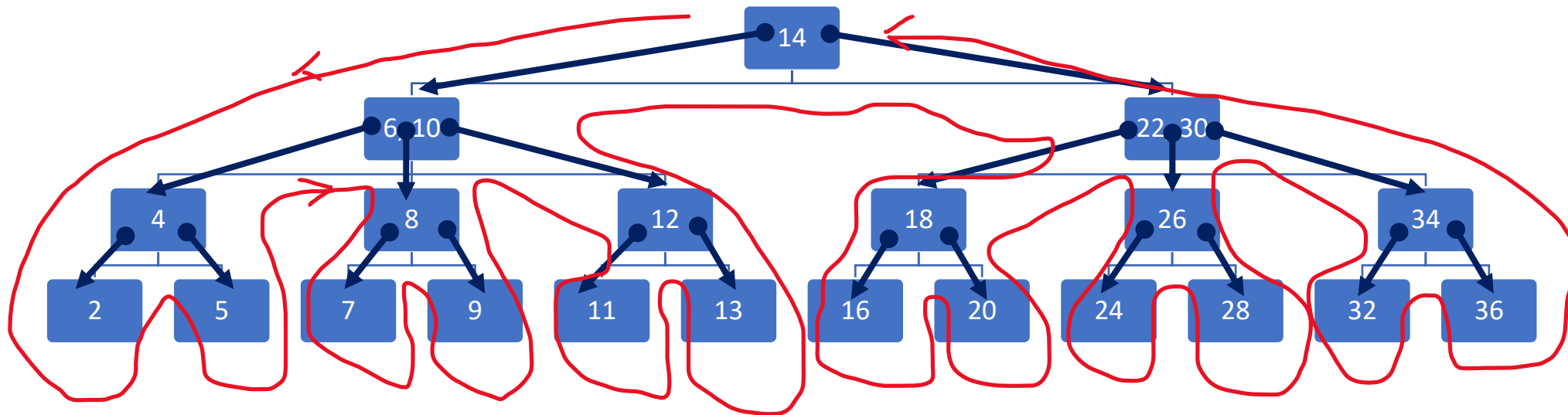
- Propriétés :
  - Toutes les feuilles ont la même profondeur, à savoir la hauteur h de l'arbre.
  - $k/2 \leq n \leq k$ . Taux de remplissage min = 50%, et moyen 75%.
  - n = nombre de clés contenus dans le nœud x
  - Si x n'est pas une feuille :
    - pour  $2 \leq i \leq n$ , pour toute clef x du fils<sub>i</sub> :  $cles_i \leq x \leq cles_{i+1}$
    - Pour toute clef x du fils<sub>1</sub> :  $x \leq cles_1$
  - Si x n'est pas la racine, n est compris entre  $k/2$  et k.



$K=2 \Rightarrow L=2, U=3$ , Arbre-B (2) ou Arbre-B(2,3)



## Arbre n-aire de recherche



Linéarisation

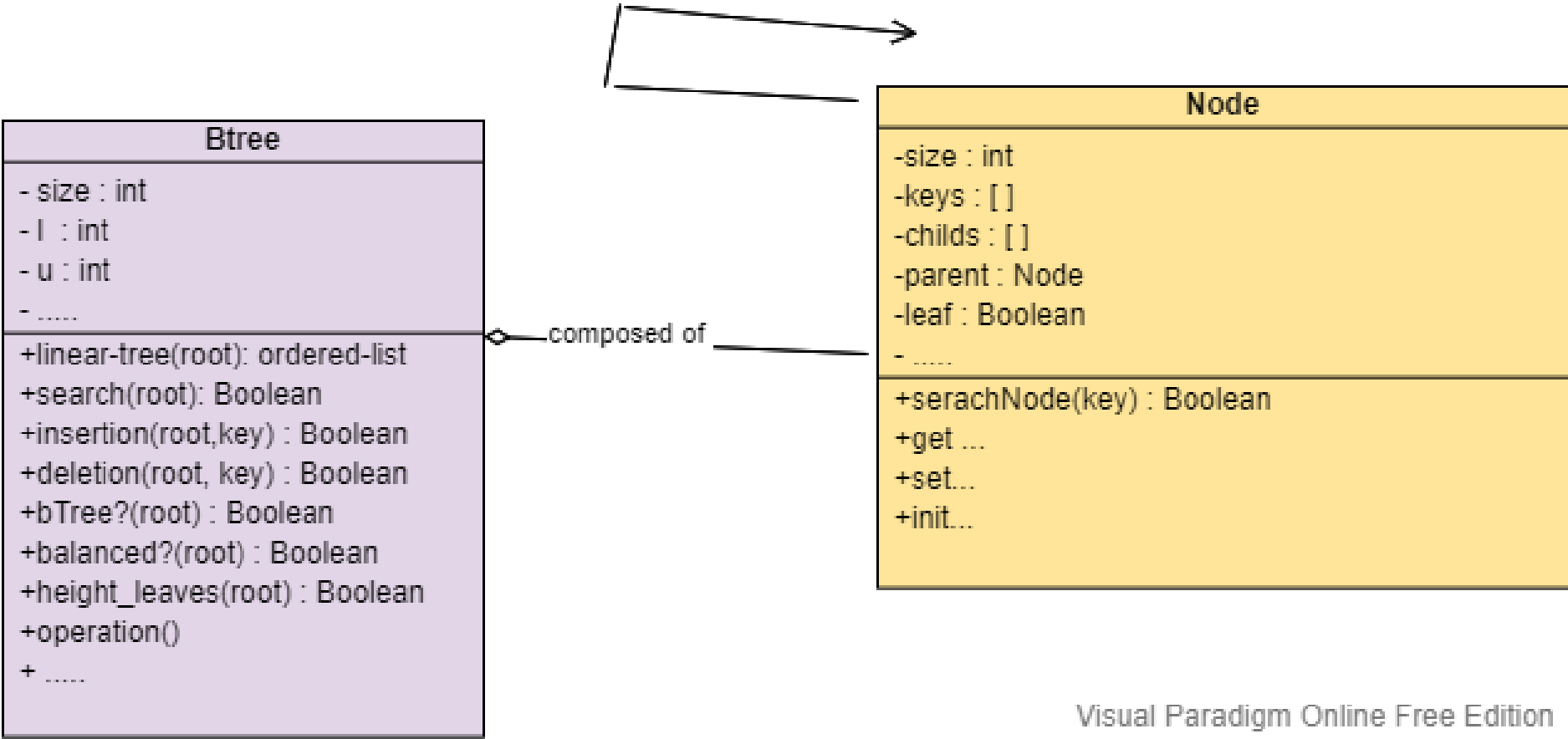
2	4	5	6	7	8	9	10	11	12
13	14	16	18	20	24	26	28	30	32
24	36								





# Architecture générale

Visual Paradigm Online Free Edition

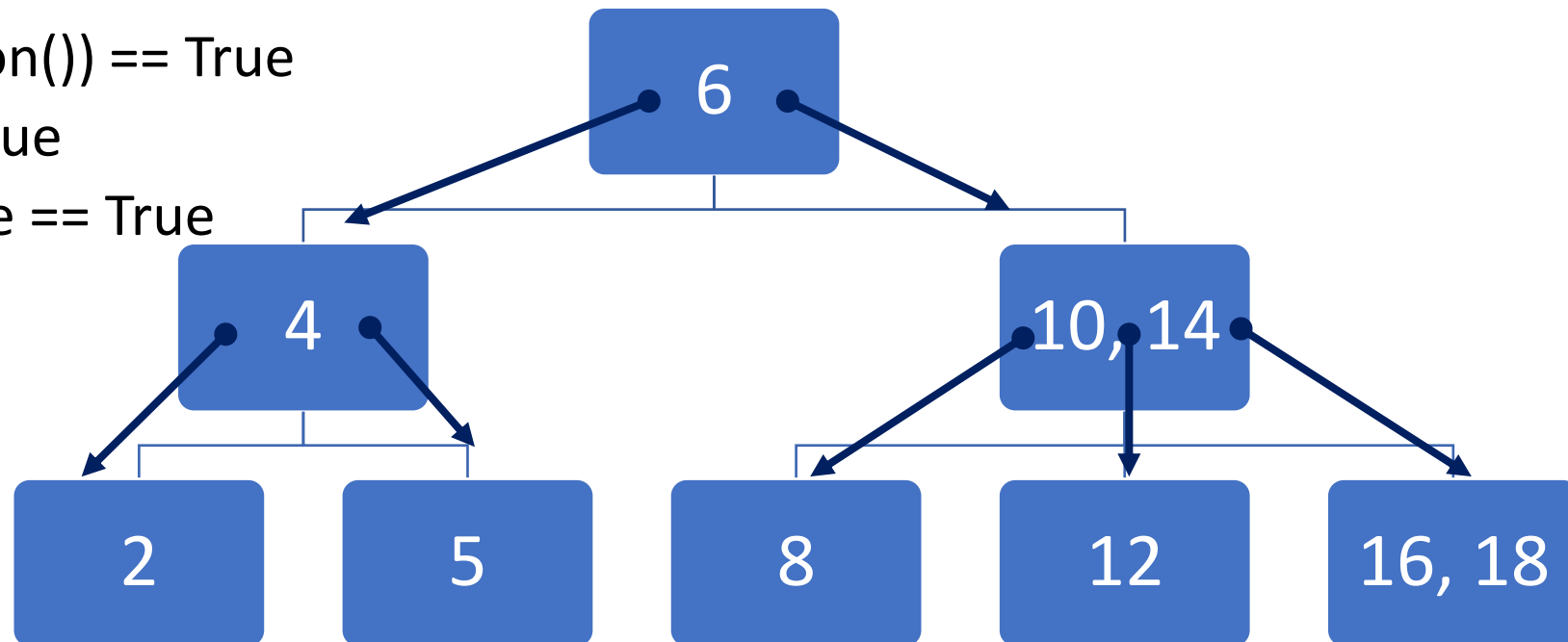


Visual Paradigm Online Free Edition

# Arbre-B? ()



- Exemples :
  - `b.Arbre-B?()` -> True
- Tests unitaires / post-conditions
  - `&& trier(b.linearisation()) == True`
  - `&& b.equilibre() == True`
  - `&& b.taux_couverture == True`



# Algorithme de vérification <à écrire>





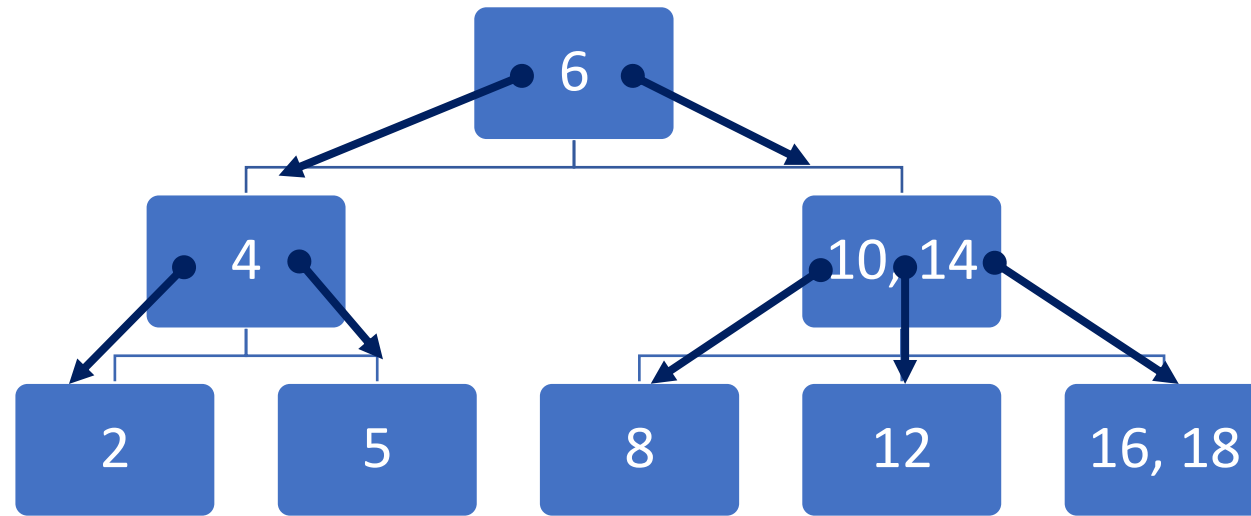
# Recherche (clé)

- Exemples :

- `b.recherche(clé) -> True/False`
- `b.recherche(6) -> True`
- `b.recherche(16) -> True`
- `b.recherche(13, root) -> False`
- `b.recherche(1) -> False`

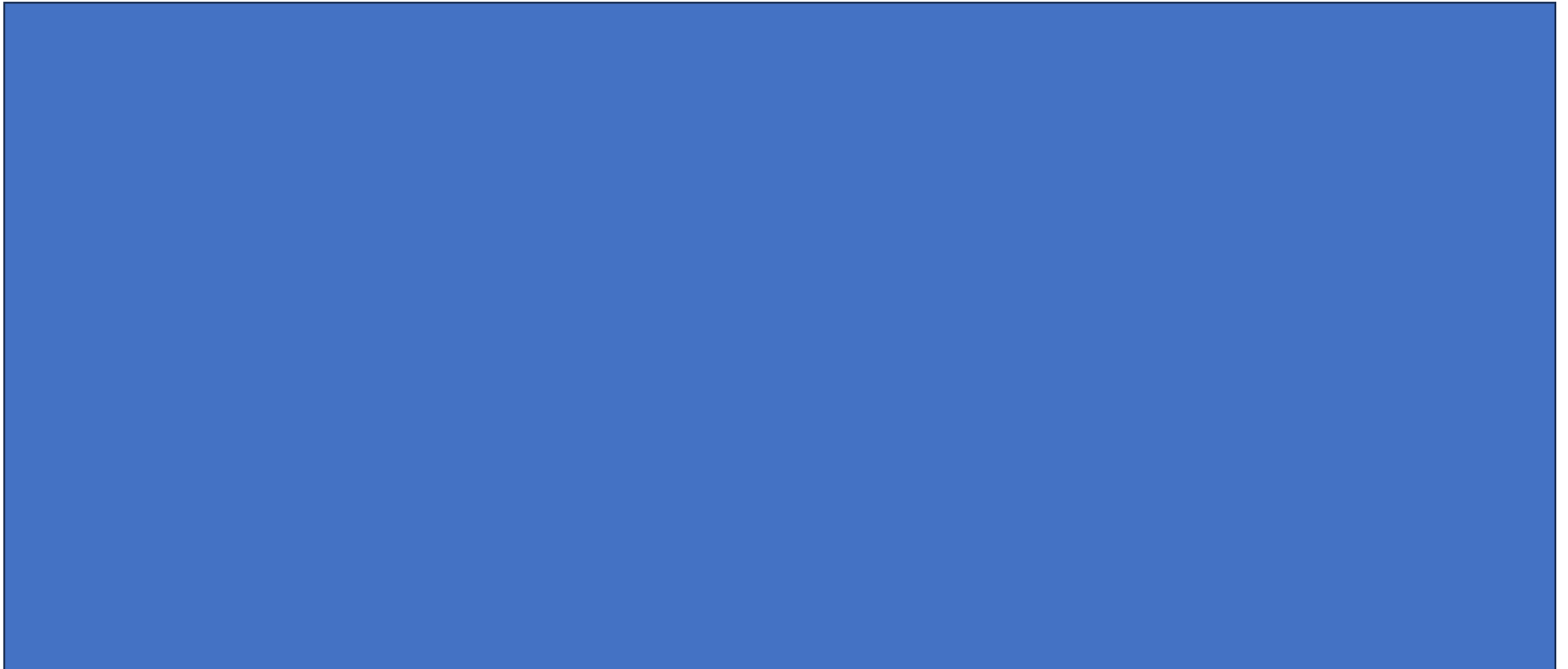
- Tests unitaires / post-conditions

- `clé in b.linearisation () == b.recherche(clé)`
- Ex. `6 in b.linearisation() == b.recherche(6) == True`
- $O(\log(n))$



2	4	5	6	8	10	12	14	16	18
---	---	---	---	---	----	----	----	----	----

# Algorithme de recherche <à écrire>



# Insertion (clé)

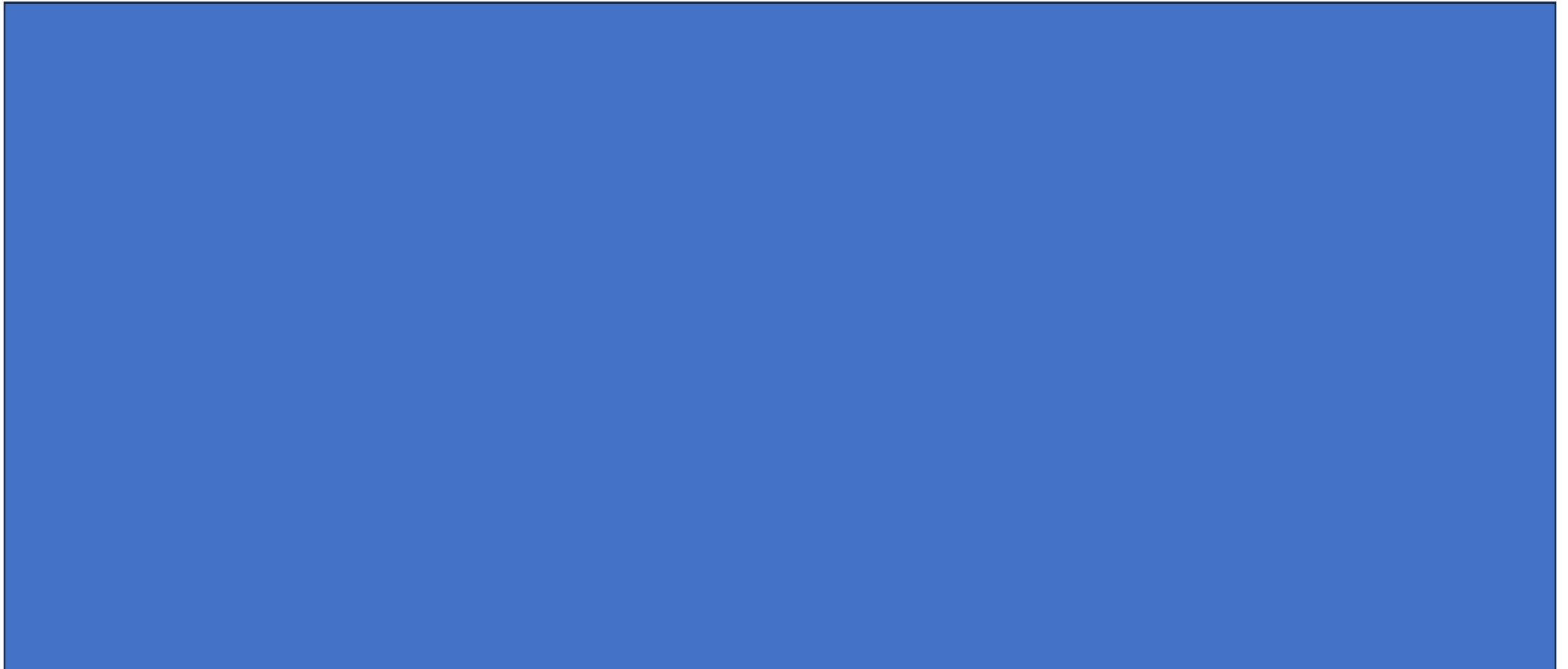


- Insertion (clé) -> True
- Insertion (listeclé) -> True
- Exemples :
  - b=Arbre-B (k=2)
  - b.insertion(clé) -> True
  - b.insertion(2) -> True
  - b.insertion(l) -> True
  - b.insertion(listeclé=[2, 4, 5, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 7, 9, 11, 13])
- Tests unitaires / post-conditions
  - b.recherche(clé) == True
  - b\_avant\_inertion.taille() == b\_après\_insertion.taille () - 1
  - b.arbre-B?()==True

<https://www.cs.usfca.edu/~galles/visualization/BTree.html>



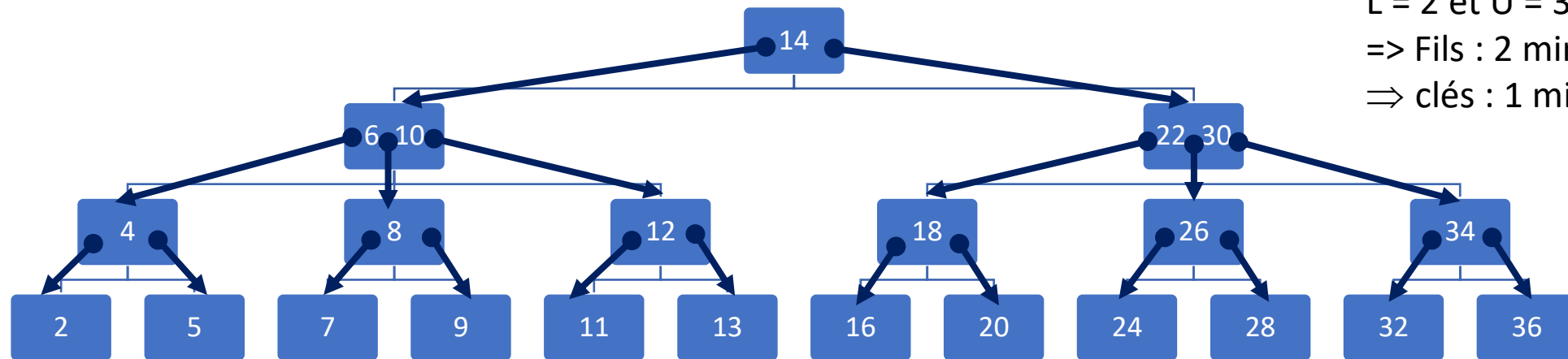
# Algorithme d'insertion <à écrire>





# Insertion (clé)

- $O(\log(n))$
- `b.insertion(l=[20, 22, 24, 26, 28, 30, 32, 34, 36, 7, 9, 11, 13]) -> True`



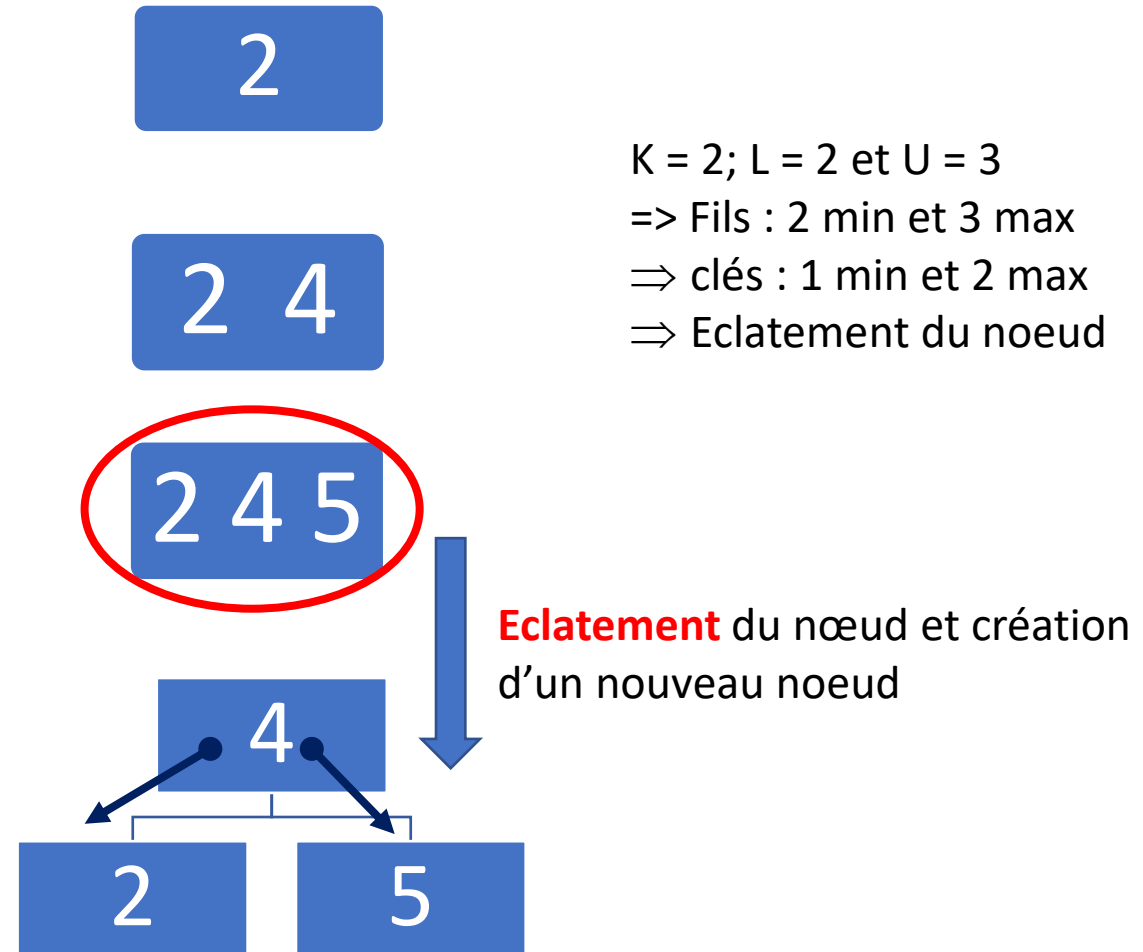
$L = 2$  et  $U = 3$   
 $\Rightarrow$  Fils : 2 min et 3 max  
 $\Rightarrow$  clés : 1 min et 2 max

- [www.youtube.com/watch?v=coRJrcIYbF4](https://www.youtube.com/watch?v=coRJrcIYbF4)



# Insertion (clé)

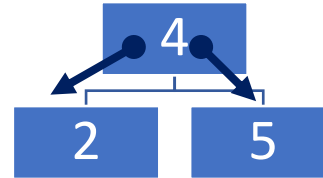
- b.insertion(2)
- b.insertion(4)
- b.insertion(5)



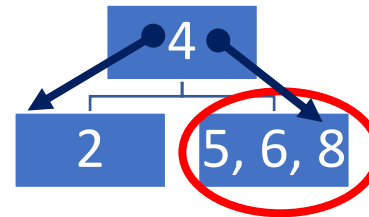
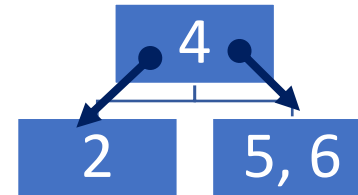


# Insertion (clé)

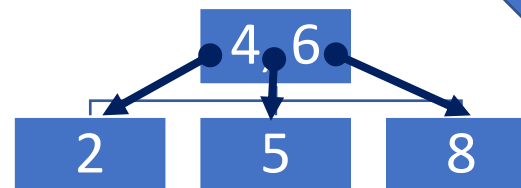
- B.insertion(6)



- b.insertion(8)



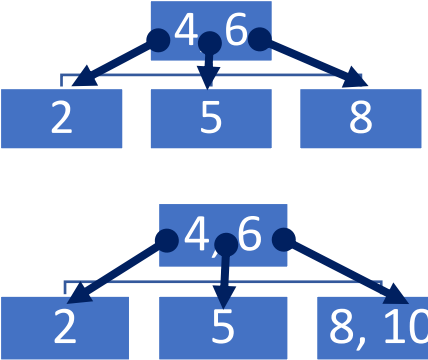
**Eclatement du nœud et  
remonter une clé** au parent



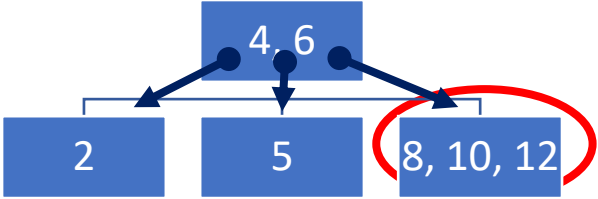


# Insertion (clé)

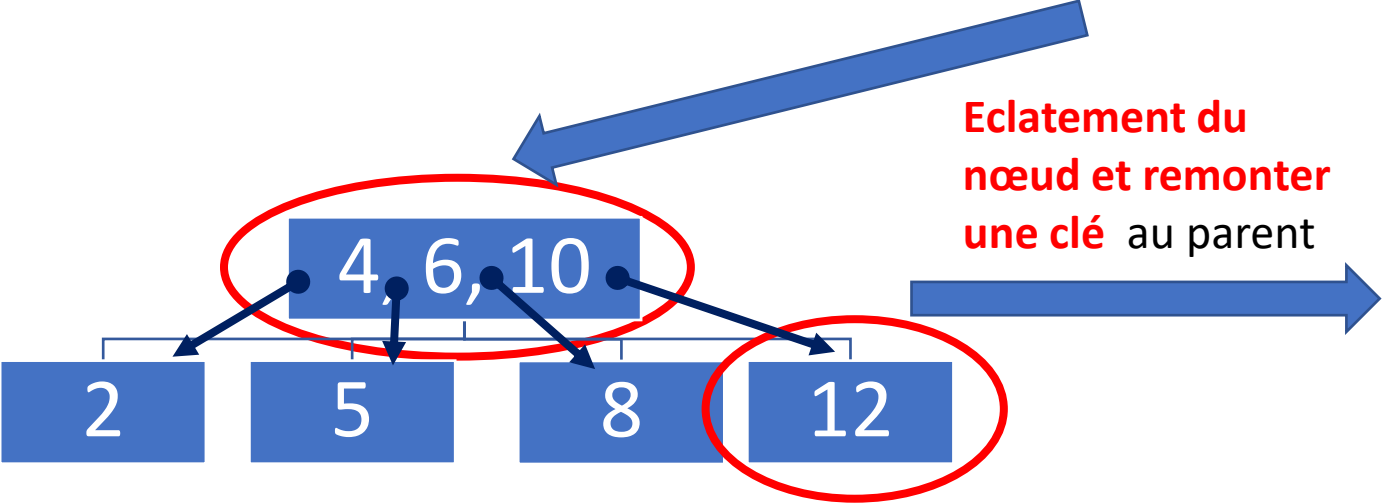
- b.insertion(10)



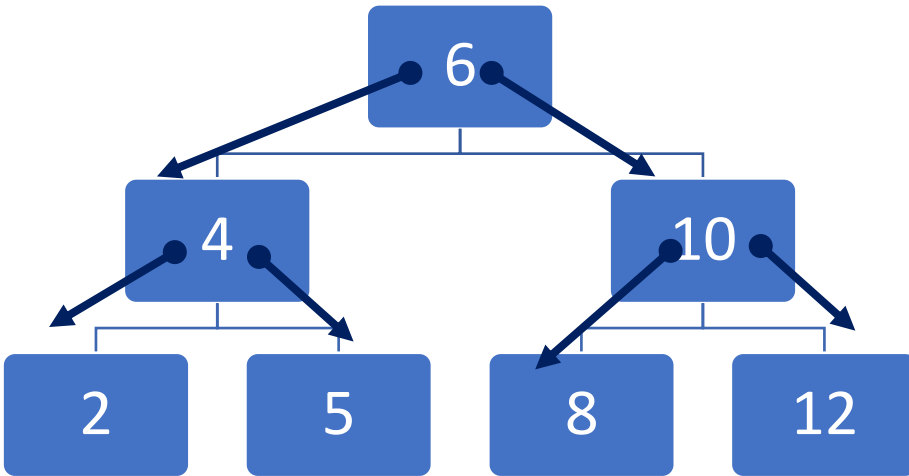
- b.insertion(12)



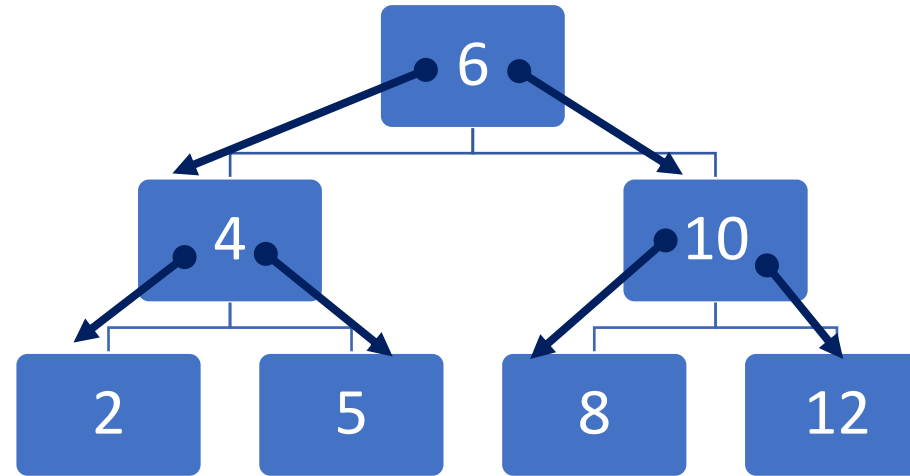
Eclatement du nœud et remonter  
une clé au parent



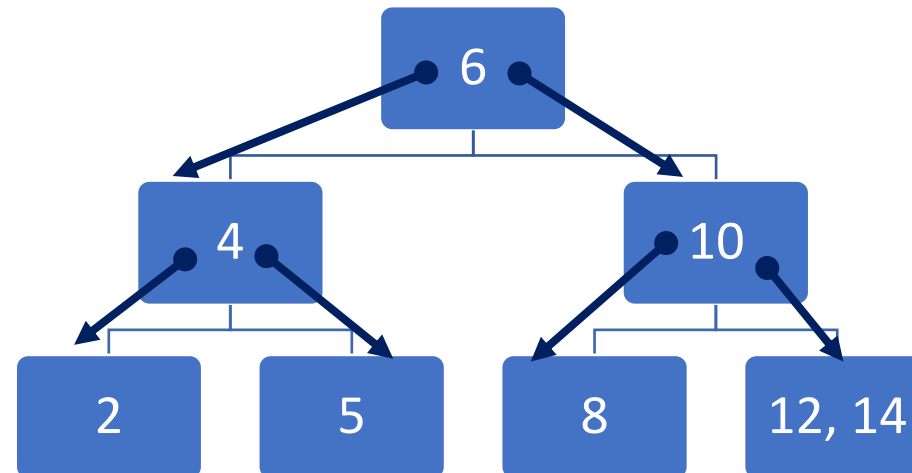
Eclatement du  
nœud et remonter  
une clé au parent



# Insertion (clé)



- `b.insertion(14)`

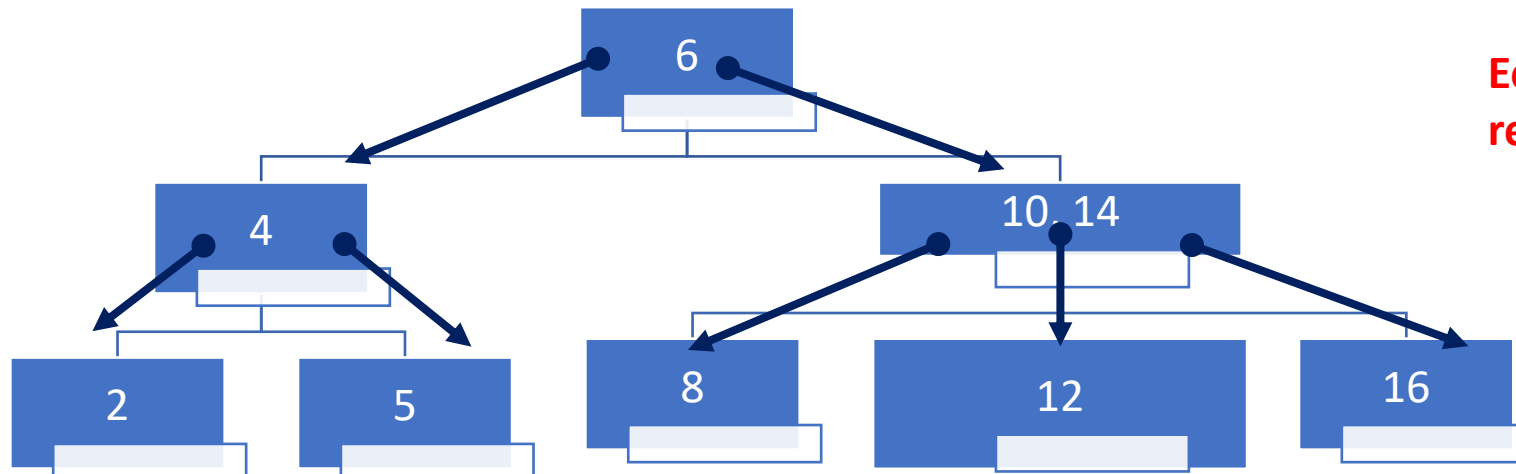
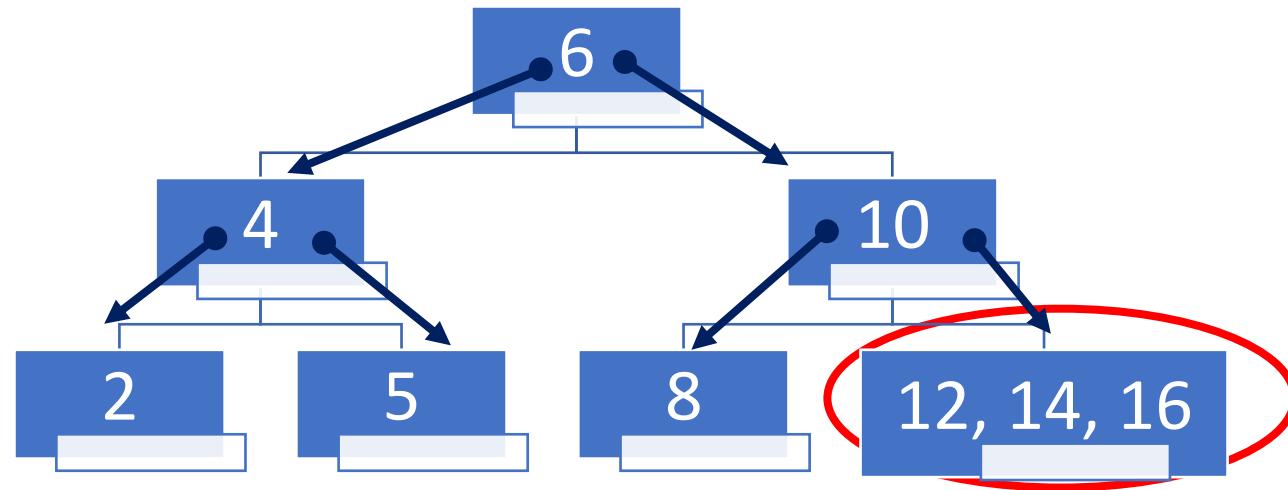






# Insertion (clé)

- b.insertion(16)

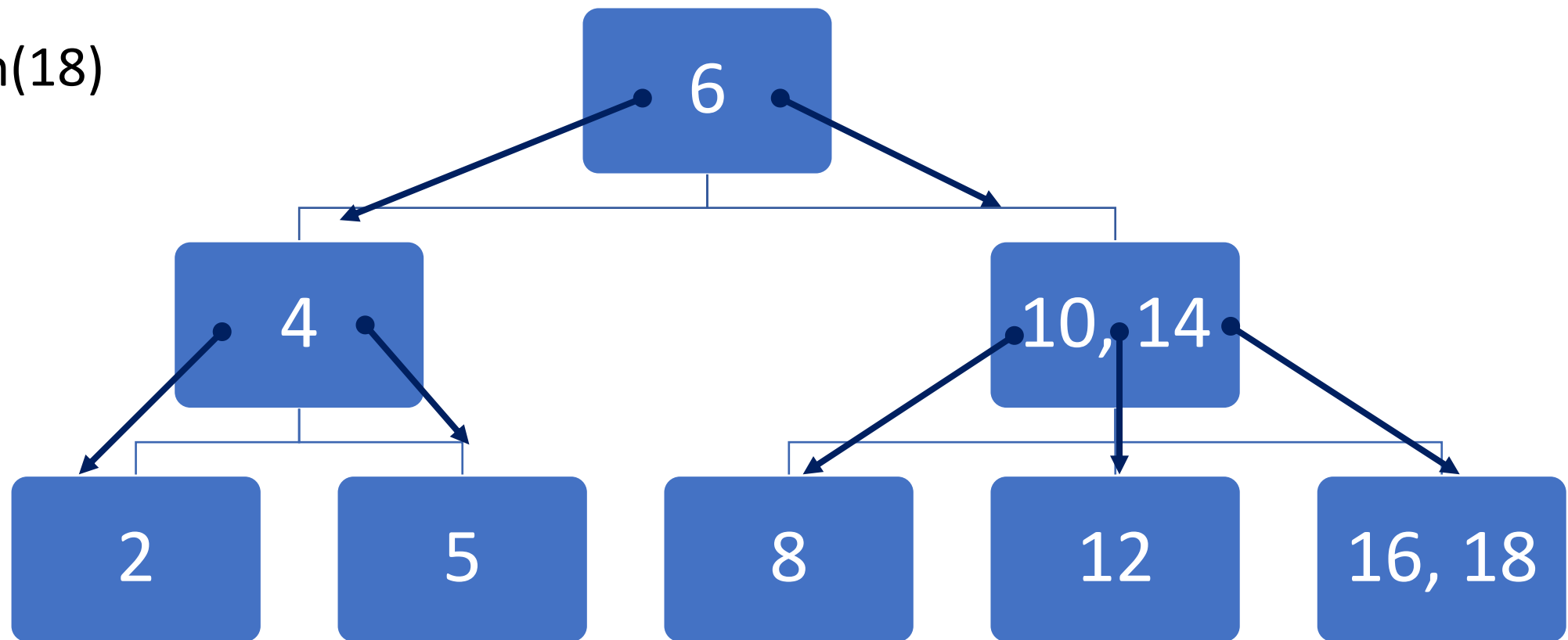


**Eclatement du nœud et  
remonter une clé au parent**

# Insertion (clé)



- b.insertion(18)



# Suppression (clé)



- $O(\log(n))$
- `suppression(listeclé) -> True/False`
- `suppression(clé) -> True/False`
- Exemples :
  - `b=arbre-B (k=2)`
  - `b.suppression(l) -> True/False`
  - `b.suppression(key) -> True/False`
  - `b.suppression(l=[14, 10, 20, 18, 16, 24, 6]) -> True`
  - `b.suppression(root, 14) .... b.suppression(root, 6)`
- Tests unitaires / post-conditions
  - `b.recherche(clé) == False`
  - `&& b_avant_insertion.taille() == b_apres_insertion.taille() + 1`
  - `b.arbre-B?()==True`

# deletion (clé)

$L = 2$  et  $U = 3$

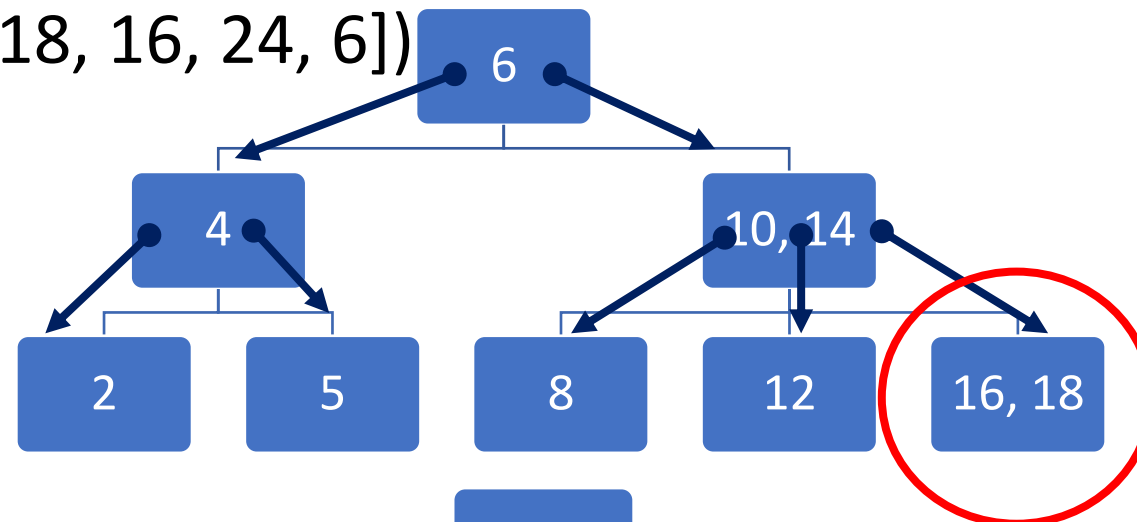
$\Rightarrow$  Fils : 2 min et 3 max

$\Rightarrow$  clés : 1 min et 2 max

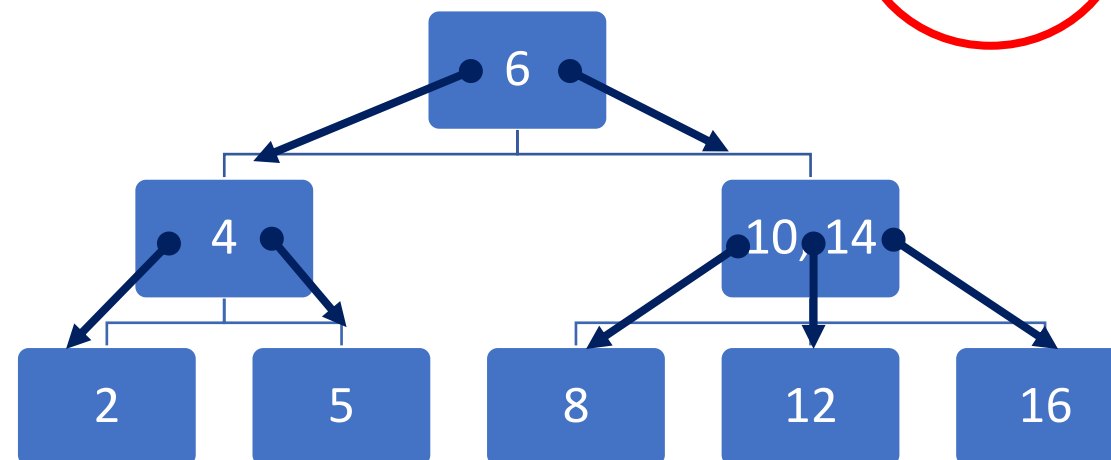
$\Rightarrow$  Eclatement du noeud



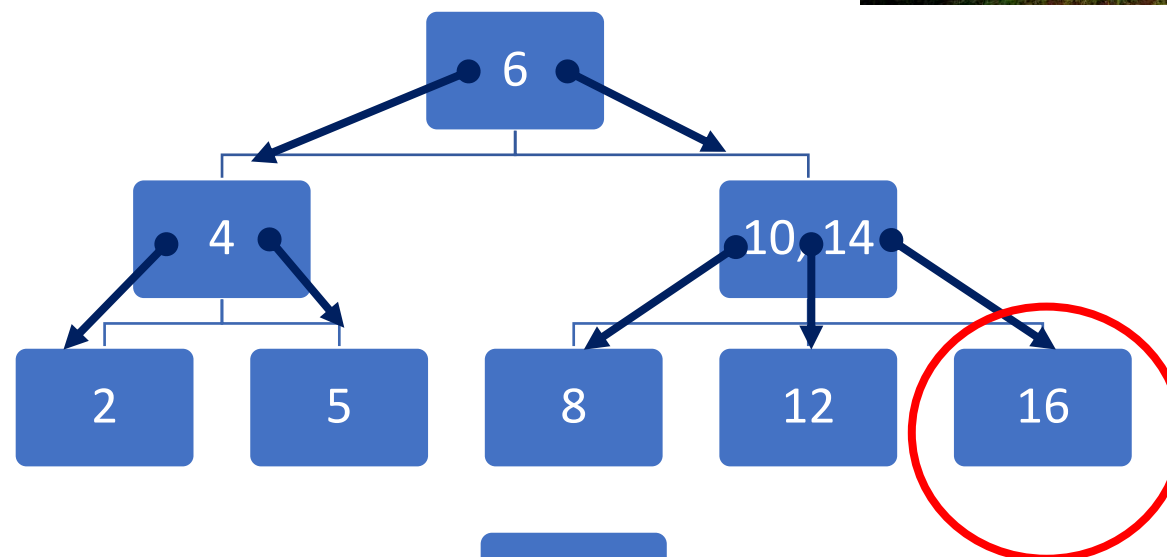
- `b.deletionL(root, , l=[18, 10, 20, 18, 16, 24, 6])`



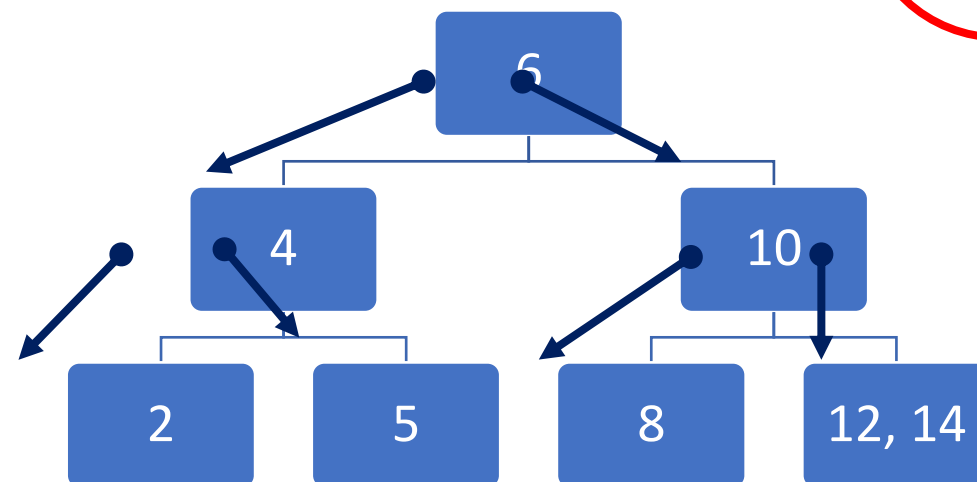
- `b.deletion(root, 18)`



# deletion (clé)



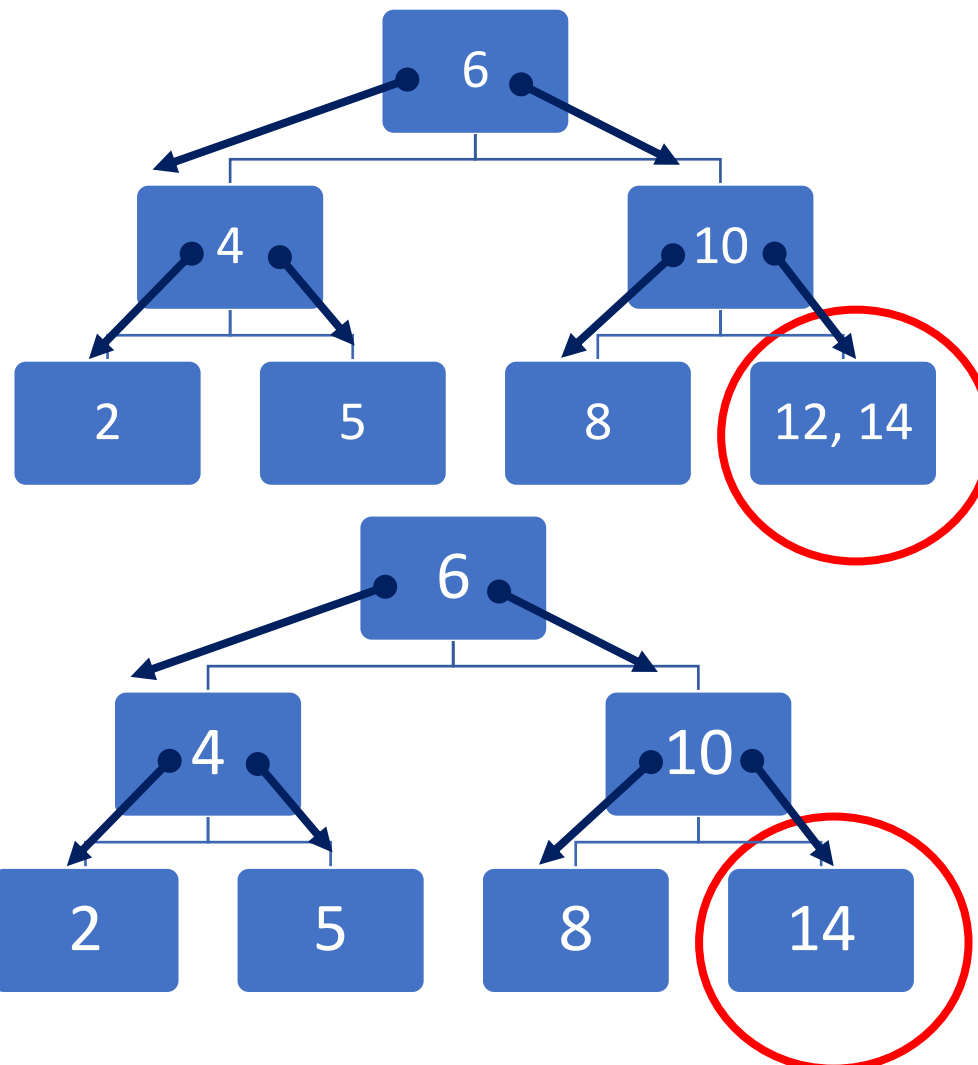
- `b.deletion(root, 16)`



# deletion (clé)



- `b.deletion(root, 12)`

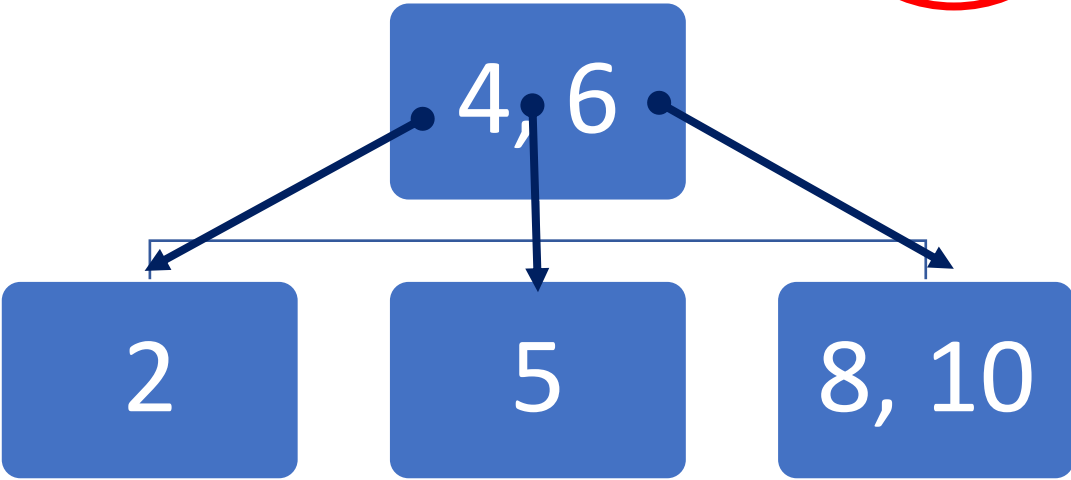
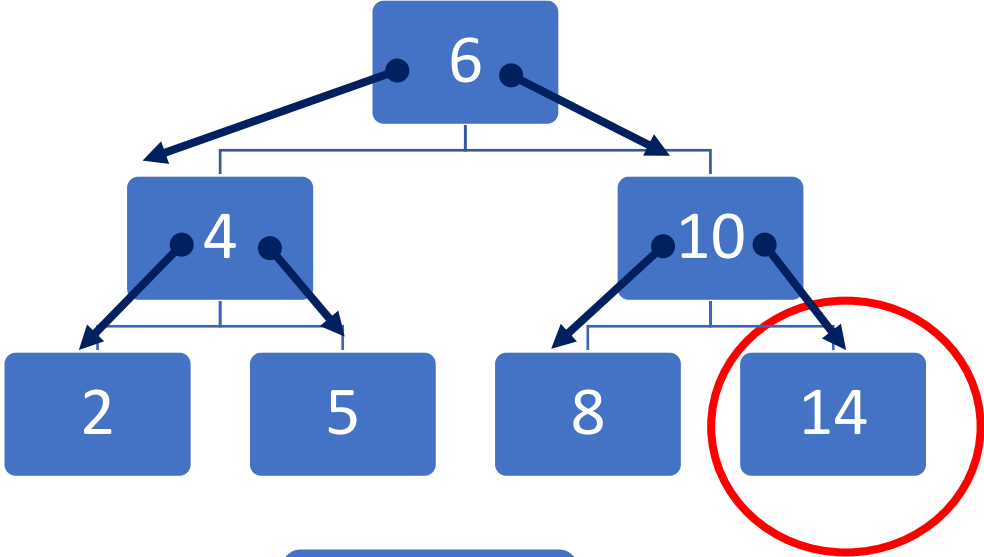
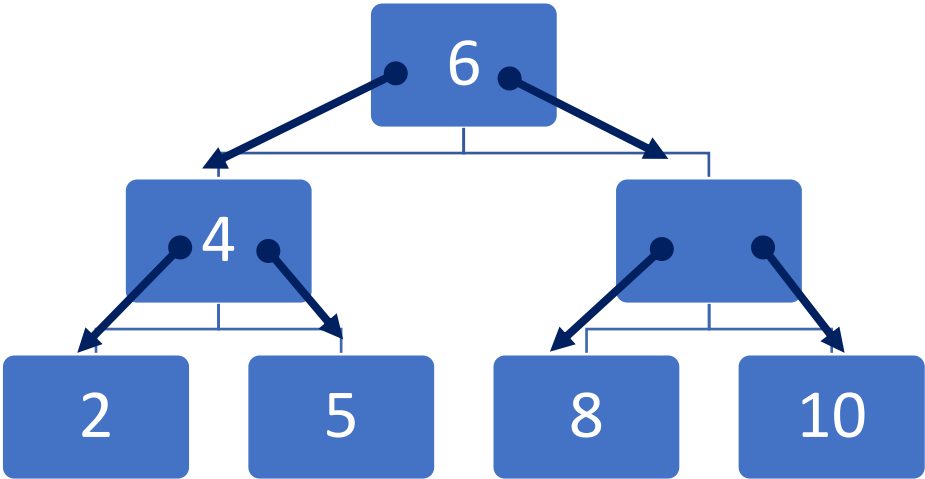






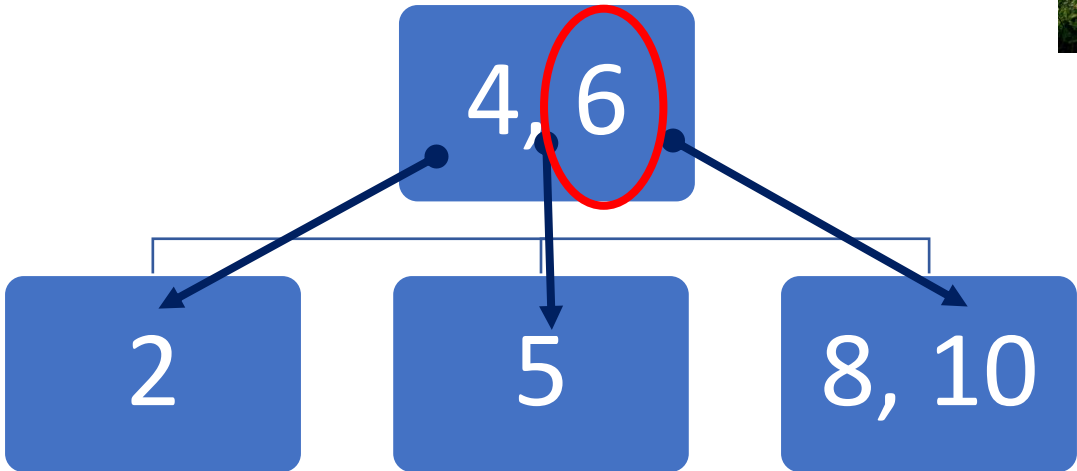
# deletion (clé)

- `b.deletion(root, 14)`

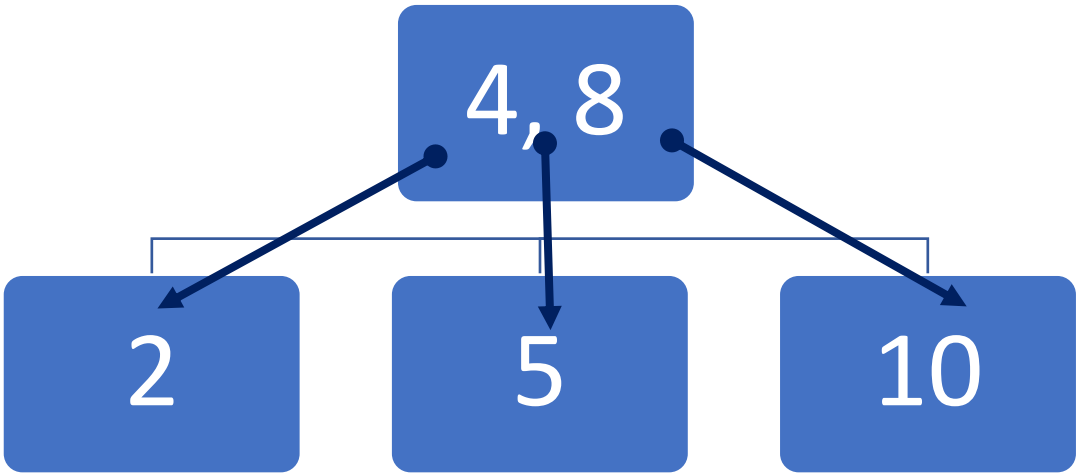




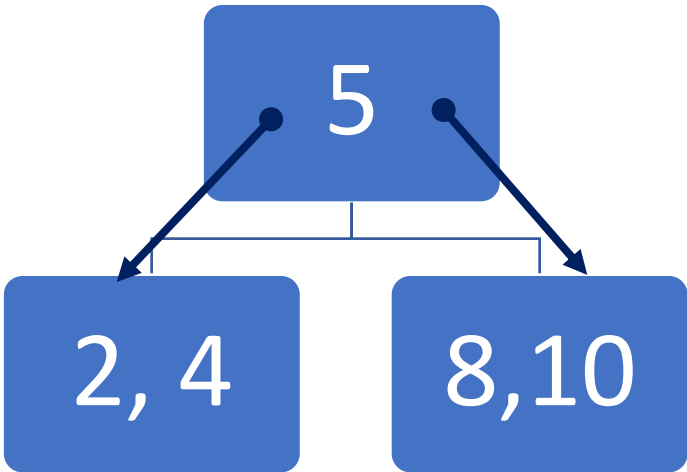
# deletion (clé)



- `b.deletion(root, 6)`



ou



# Expérimentations : tests



- $K = 2$ . Le noeud contient 2 clés au maximum et 1 clé au minimum.
- Insertion dans l'ordre : 2, 4, 5, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 7, 9, 11, 13.
- Suppression dans l'ordre : 14, 10, 20, 18, 16, 24, 6
- La vidéo ci-dessous visualise l'état de l'arbre à chaque insertions ou suppression.
- <https://www.youtube.com/watch?v=coRJrcIYbF4>

# Expérimentations : tests



- Suite à cette première batterie de tests, il convient de réaliser d'autres tests.
- Le noeud contient  $k = 10$  clés au maximum et 5 clés au minimum.
- L'arbre contient 100 000 clés.
- Suppression dans l'ordre : 10, 20, 30, ... 5000, 5, 15, 25, 35, ..4995



# BONUS - Expérimentations : val. opt. k

- La performance du schéma dépend de la valeur de k. accès disque = 1 ms et l'accès à une page en mémoire centrale = 1 nano seconde. Normalement, nous devrions trouver une valeur optimale de  $64 < k < 128$ .  $v$  = taux d'occupation.  $1 < v < 2$ .  $f(k,v)$  = temps de parcours de l'arbre.

K	F(k,1)	F(k,5)	F(k,2)
2			
4			
$2^3$			
...			
$2^{16}$			

Hauteur de l'arbre	Taille minimale de l'index	Taille maximale de l'index
2		
4		
$2^3$		
...		
$2^{16}$		



# Expérimentations : Performance

- % stockage utilisé
- VR / T : nombre moyen de lectures virtuelles sur disque / transaction
- PR / T : nombre moyen de lectures physiques / transaction
- VW / I : nombre moyen d'écritures virtuelles / insertion ou suppression
- PW / I : nombre moyen d'écritures physiques / insertion ou suppression
- T/s : nombre moyen de transactions / seconde

	% stockage utilisé	VR/T*	PR/T	VW/I	PW/I	T/sec.
E1(1)						
....						
E10(4)						



# Expérimentations : Scénarios



- E1: 25 éléments/page
  - (1) 10000 insertions sequential by key,
  - (2) 50 insertions, 50 retrievals, and 100 deletions uniformly random in the key space.
- E2: 120 elements/page; otherwise identical to E1.
- E3: 250 elements/page; otherwise identical to E1.
- E4: 120 elements/page, overflow permitted.
  - (1) 10000 insertions sequential by key,
  - (2) 1000 retrievals uniformly random in key space,
  - (3) 10000 sequential deletions.
- E5: 120 elements/page.
  - (1) 5000 insertions uniformly random in key space,
  - (2) 1000 retrievals uniformly random in key space,
  - (3) 5000 deletions uniformly random in key space.

# Expérimentations : Scénarios



- E6: Overflow permitted; otherwise identical to E5.
- E7: 120 elements/page, overflow permitted.
  - (1) 5000 insertions sequential by key,
  - (2) 6000 each insertions, retrievals, and deletions uniformly random in key space.
- E9: 250 elements/page; otherwise identical to E8.
- E10: 120 elements/page, overflow permitted.
  - (1) 100 000 insertions sequential by key,
  - (2) 1000 each insertions, deletions, and retrievals uniformly random in key space,
  - (3) 100 group retrievals uniformly random in key space, where a group is a sequence of 100 consecutive keys (statistics on the basis of 10000 transactions),
  - (4) 10000 insertions sequential by key, to merge uniformly with the elements inserted in phase (1).



# Références

- R. Bayer et E. McCreight, « Organization and maintenance of large ordered indices », SIGFIDET '70: Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control, ACM, novembre 1970, p. 107-141 (ISBN 978-1-4503-7941-0, DOI 10.1145/1734663.1734671)
- [www.youtube.com/watch?v=coRJrcIYbF4](http://www.youtube.com/watch?v=coRJrcIYbF4)
- <https://www.cs.usfca.edu/~galles/visualization/BTree.html>

# Python - Objets

- Classe
- Objet
- Communication avec envoi de messages
- Héritage
- Agrégation
- Polymorphisme