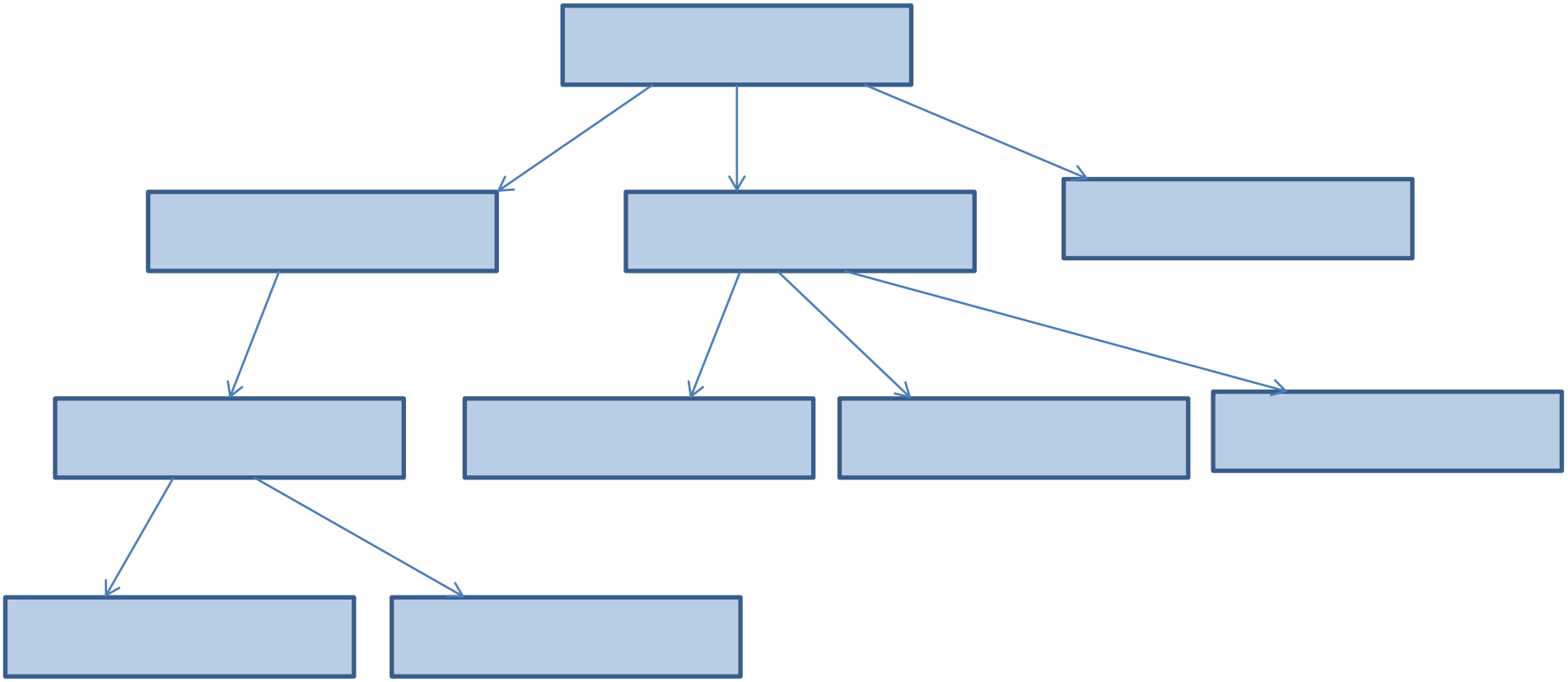
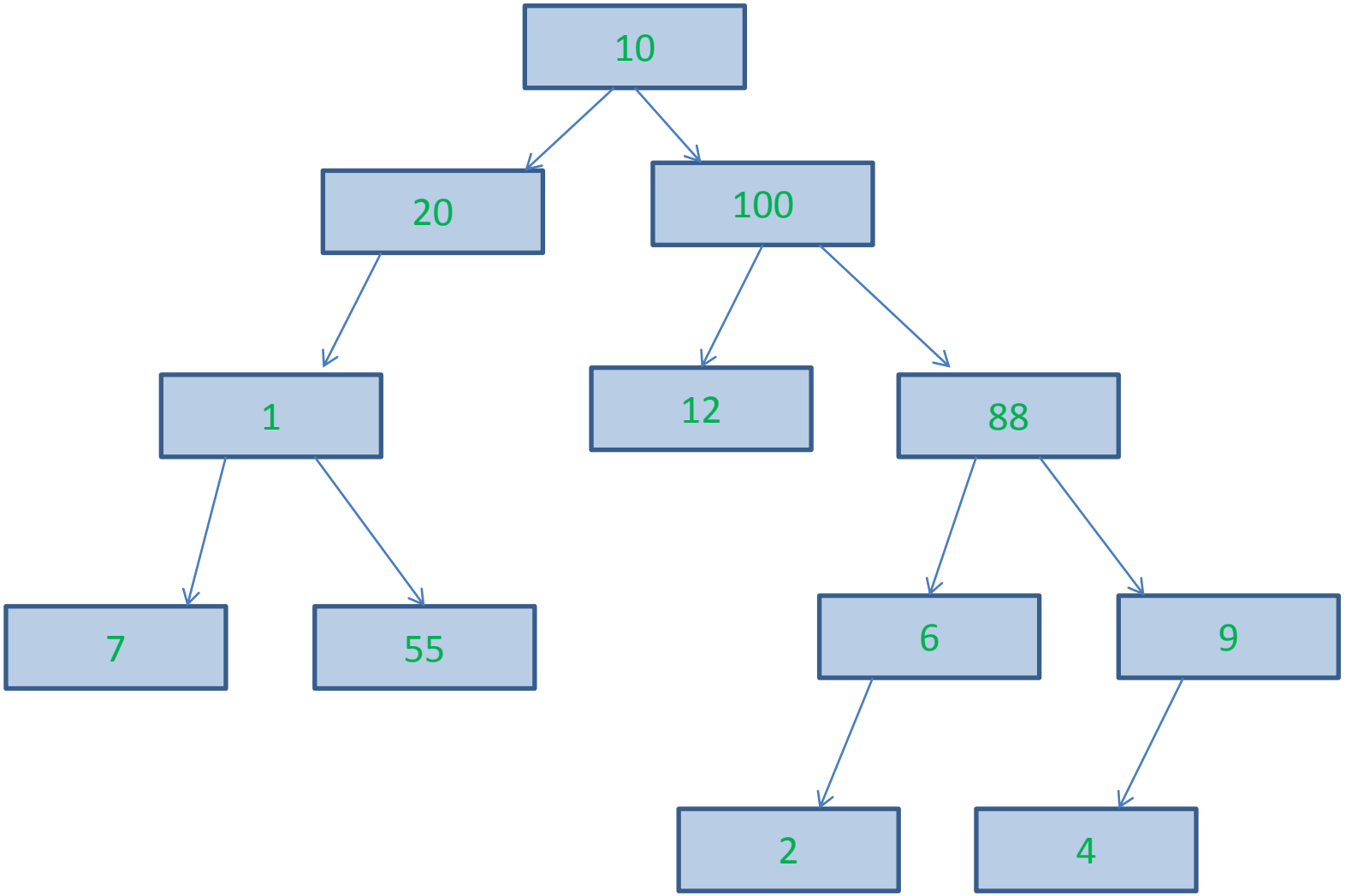
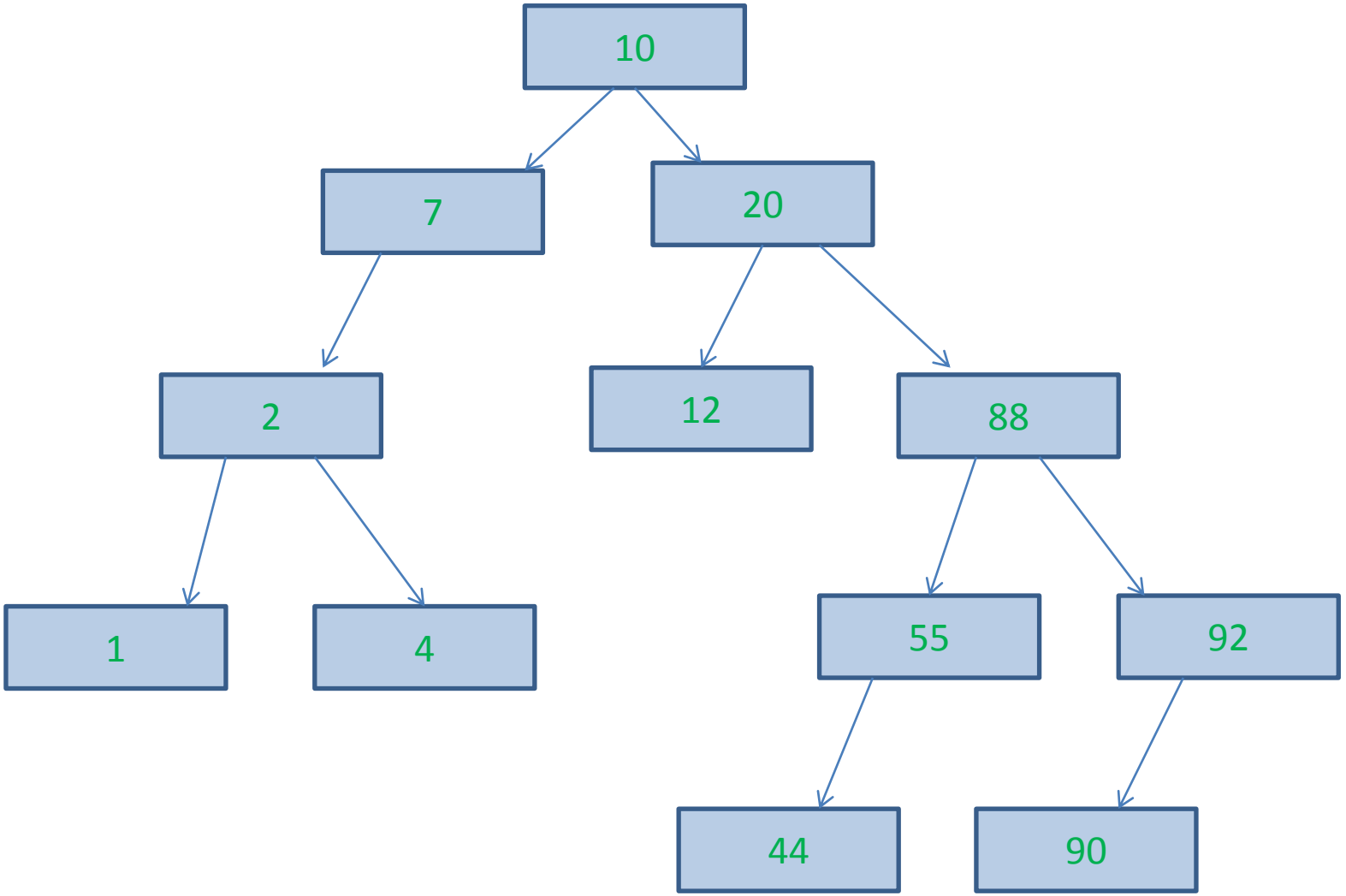


В общем смысле дерево - это иерархическая структура, хранящая коллекцию объектов.

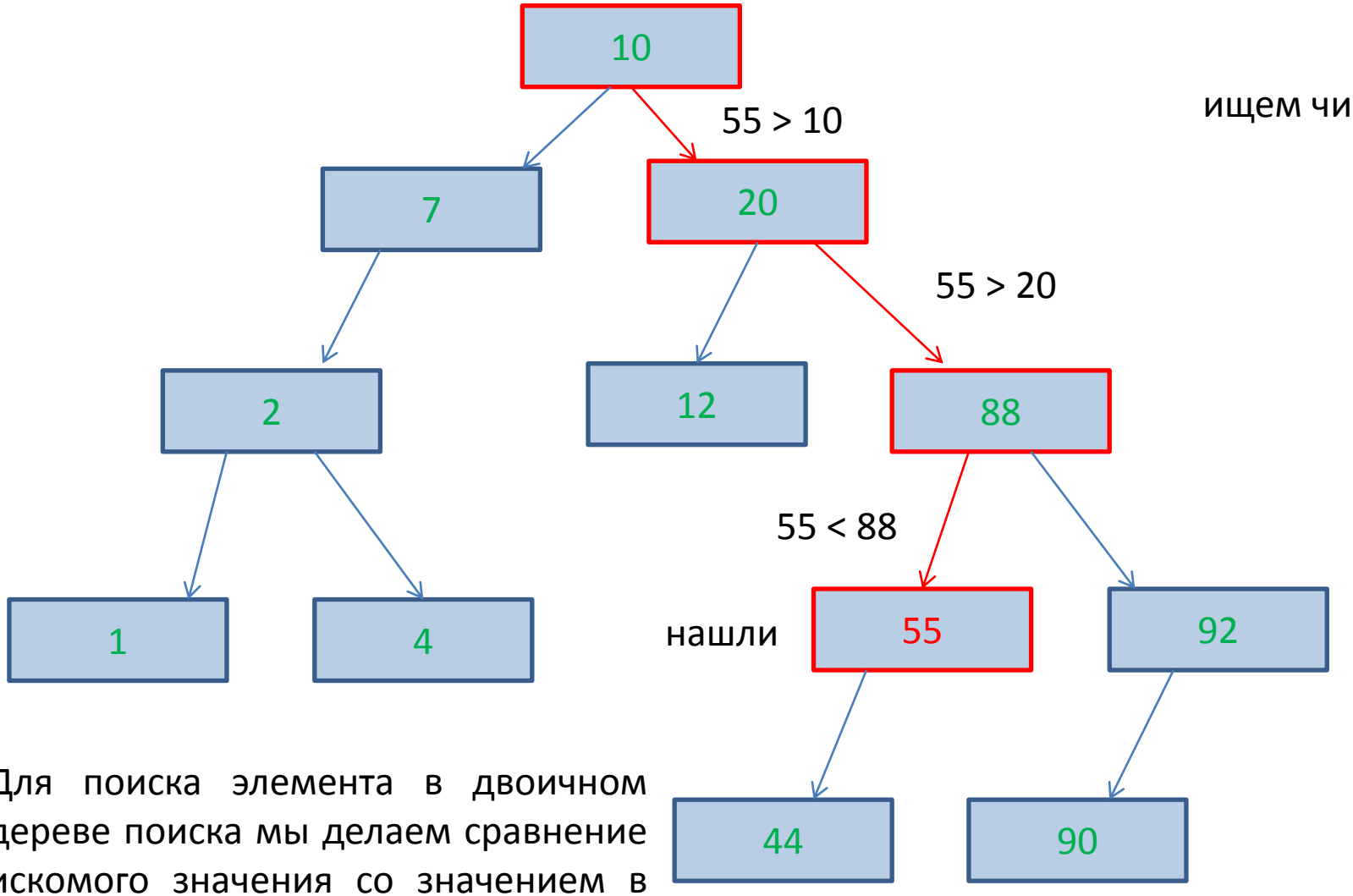


Среди деревьев выделяют **двоичные деревья**, в которых у каждого узла может быть не более двух сыновей.

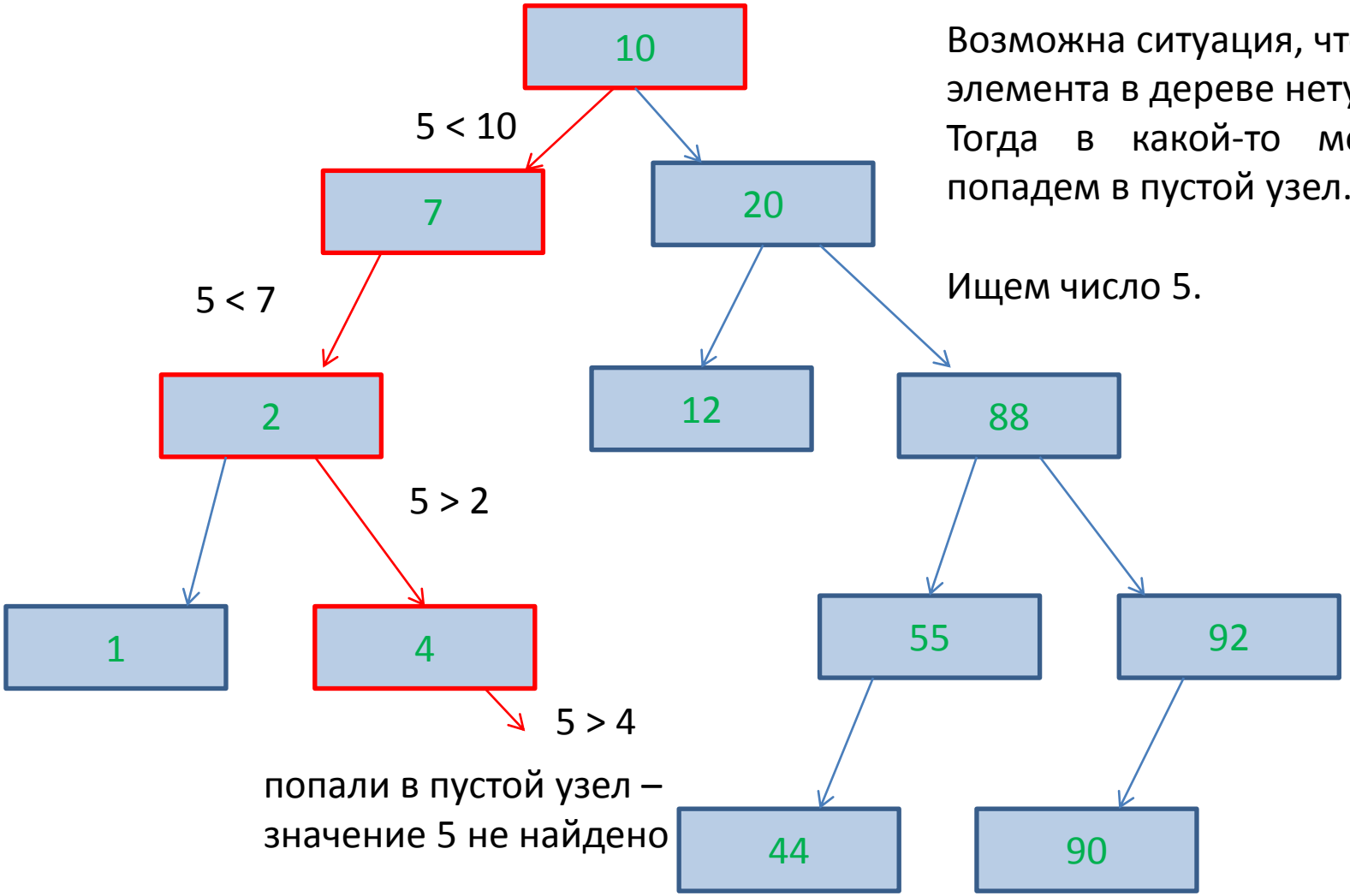
Для простоты стрелки к несуществующим сыновьям будут опускаться.
Также как информационная часть узла будут рассматриваться целые числа.



Среди двоичных деревьев выделяют **двоичные деревья поиска**, в которых у каждого узла в левом поддереве содержатся элементы с меньшими чем в данном узле значениями, а в правом поддереве – с большими.



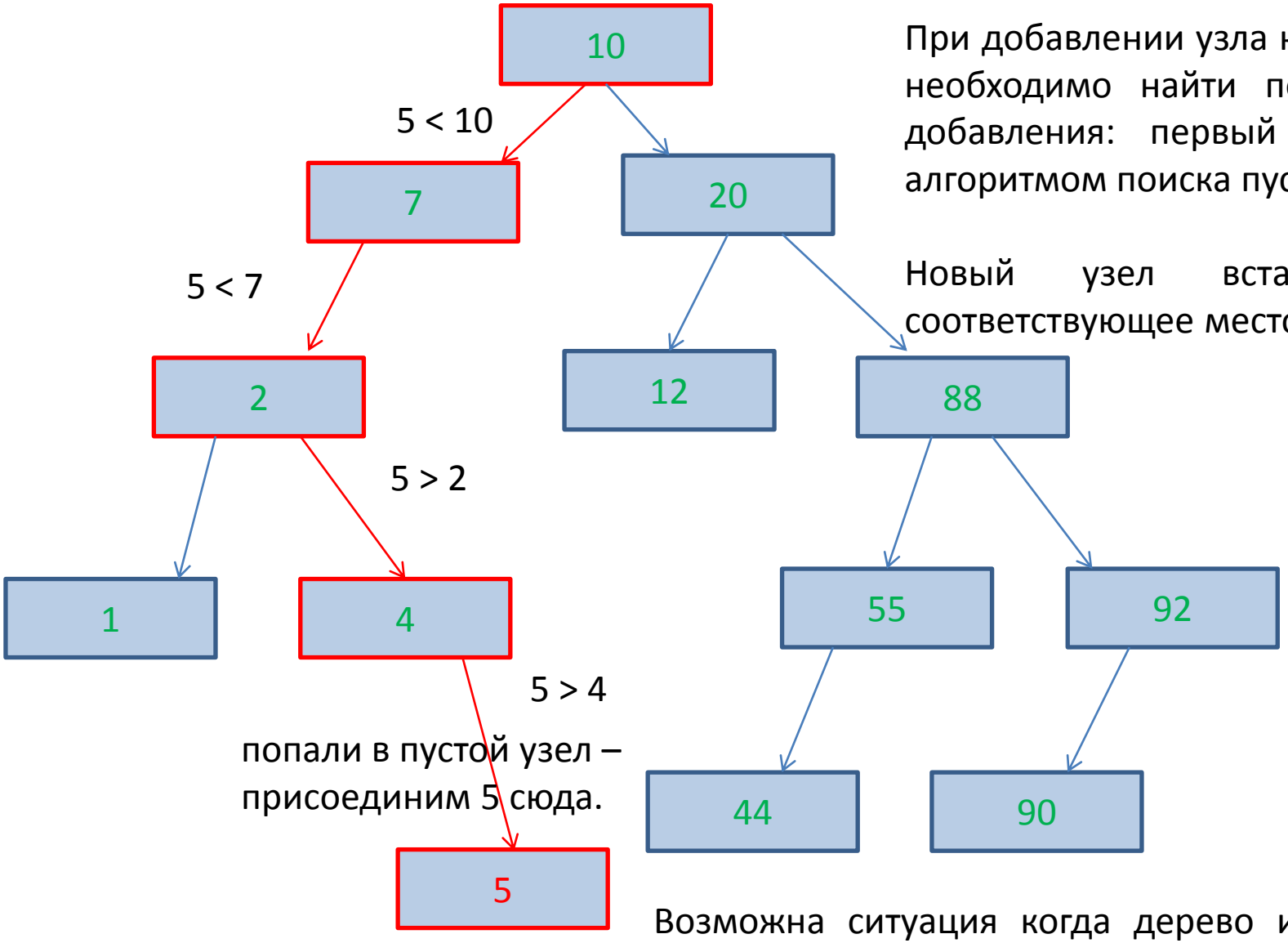
Для поиска элемента в двоичном дереве поиска мы делаем сравнение искомого значения со значением в текущем узле. Если совпадают – элемент найден. Если искомое значение меньше – повторяем операцию в левом поддереве. Если больше – в правом.



Возможна ситуация, что искомого элемента в дереве нету.
Тогда в какой-то момент мы попадем в пустой узел.

Ищем число 5.

попали в пустой узел –
значение 5 не найдено



При добавлении узла к дереву нам необходимо найти позицию для добавления: первый найденный алгоритмом поиска пустой узел.

Новый узел вставляется в соответствующее место.

Возможна ситуация когда дерево изначально пустое – тогда новый узел станет корнем.
Также узел с требуемым значением уже может существовать – тогда нам нету необходимости делать добавление.

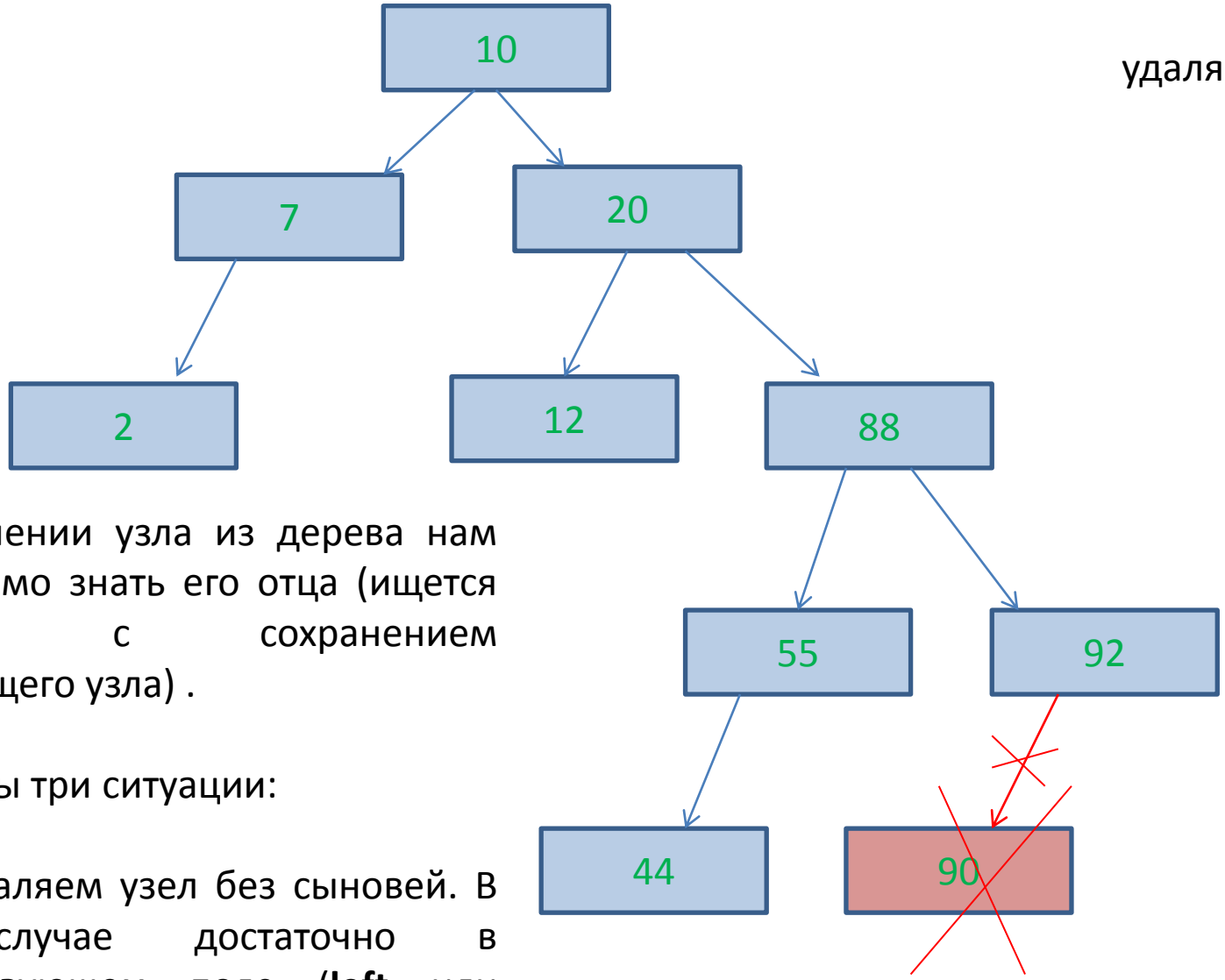
Для представления узла двоичного дерева обычно используется следующая структура:

```
struct TreeItem {
   struct TreeItem* left;   /*указатель на левого сына*/
   struct TreeItem* right; /*указатель на правого сына*/

   int info;                    /*информационная часть*/
};
```

А само дерево представляется своим корнем:

```
struct TreeItem *root = NULL;
```



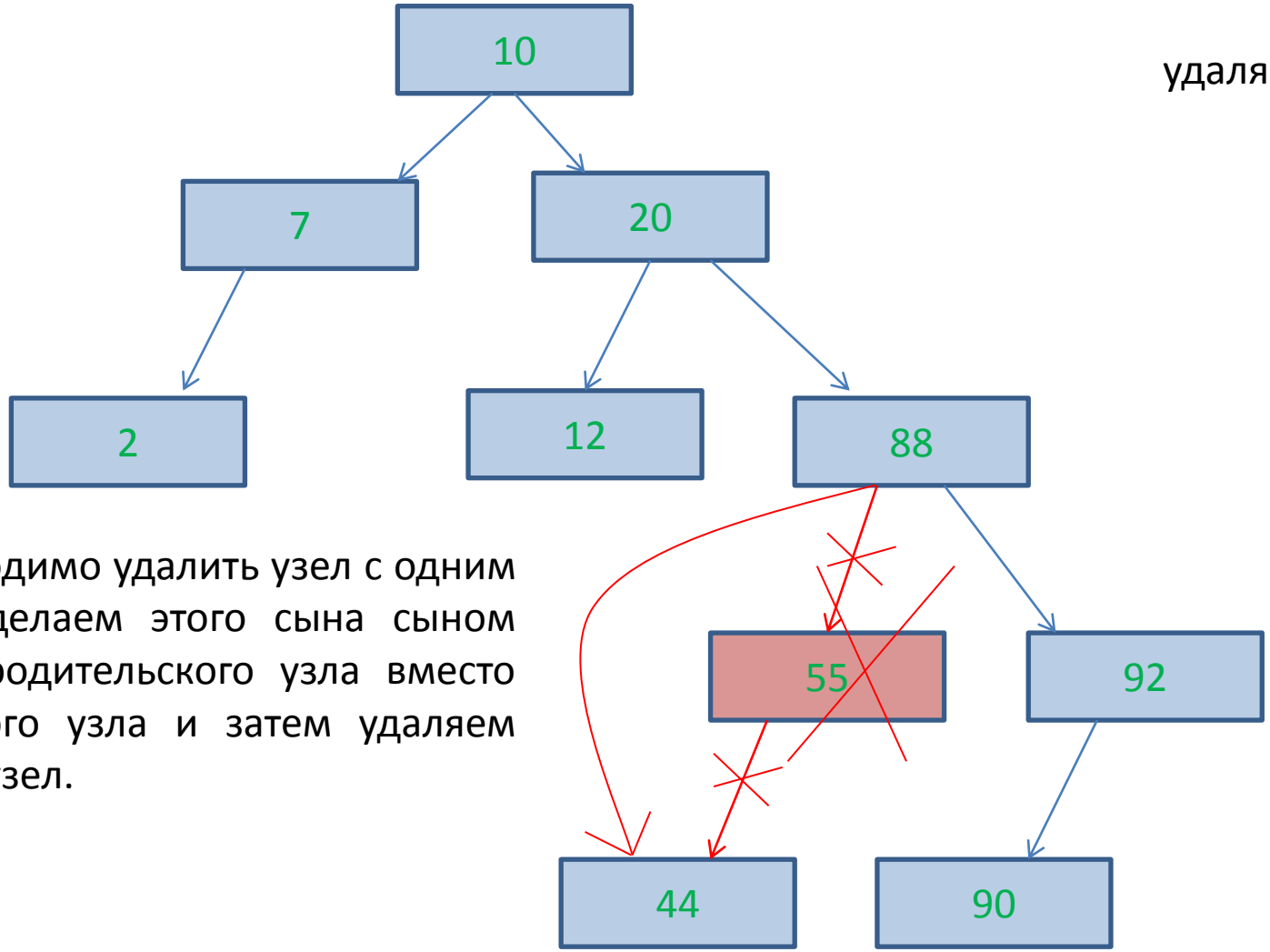
удаляем узел 90

При удалении узла из дерева нам необходимо знать его отца (ищется поиском с сохранением предыдущего узла) .

Возможны три ситуации:

а) мы удаляем узел без сыновей. В этом случае достаточно в соответствующем поле (**left** или **right**) родительского узла прописать **NULL** и удалить требуемый узел.

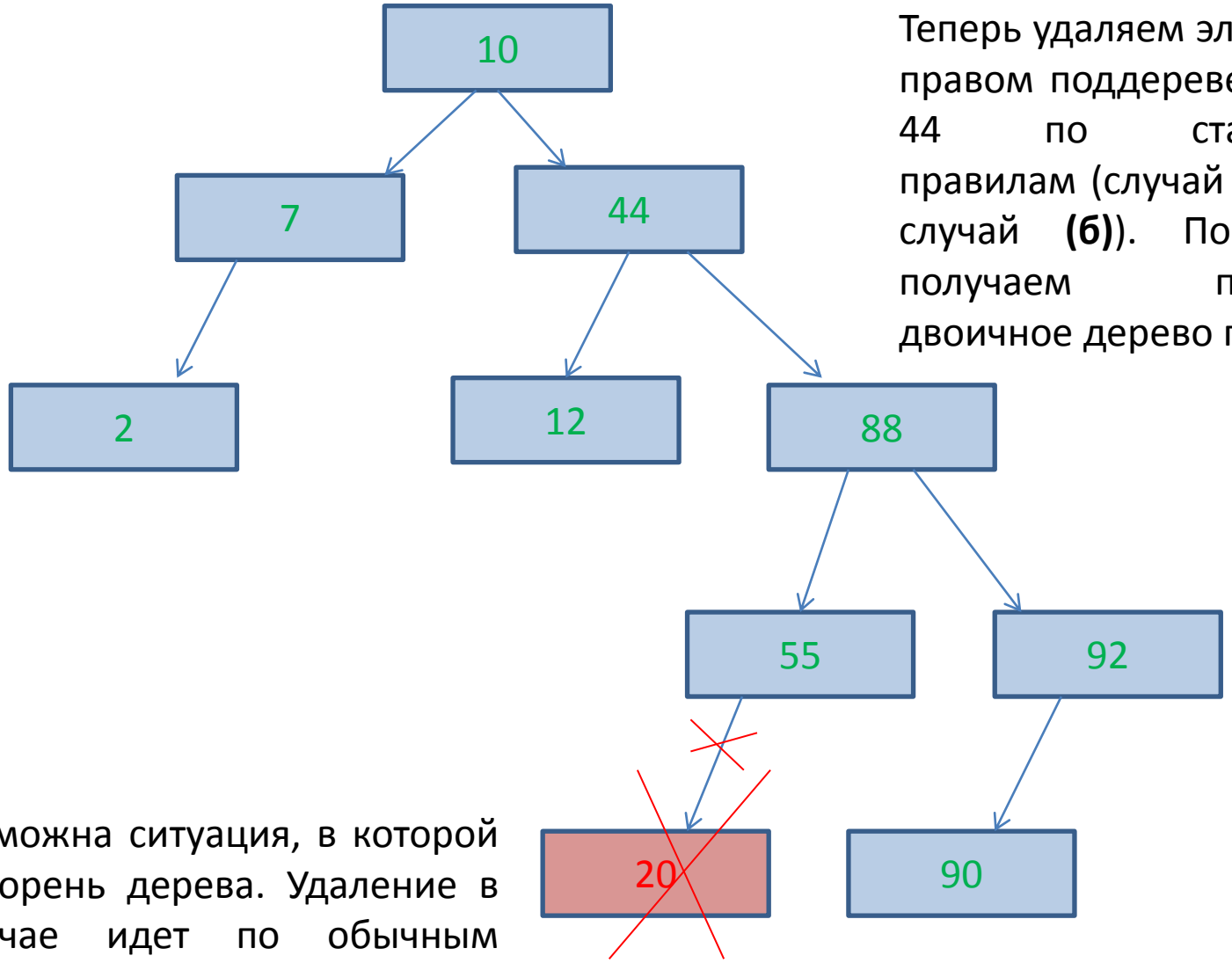
удаляем узел 55



б) необходимо удалить узел с одним сыном: делаем этого сына сыном нашего родительского узла вместо удаляемого узла и затем удаляем нужный узел.

Удаляем элемент 20, у которого два сына. Находим в правом поддереве самый маленький элемент (44) и меняем значения местами.
Для поиска такого элемента мы всегда двигались в левую ветку в правом поддереве.

в) при удалении узла с двумя сыновьями нам необходимо: найти в его правом поддереве самый маленький элемент (или в левом самый большой) и **поменять местами значения** в найденном узле и в удаляемом узле. После этого удалить в его правом поддереве (левом поддереве) нужный элемент по стандартному алгоритму. У него после обмена будет не более чем один сын.



Теперь удаляем элемент 20 в правом поддереве элемента 44 по стандартным правилам (случай **(а)** или же случай **(б)**). После этого получаем правильное двоичное дерево поиска.

Также возможна ситуация, в которой удаляют корень дерева. Удаление в этом случае идет по обычным правилам, но после удаления у дерева может поменяться корневой узел. Или же дерево может стать пустым после удаления из него единственного узла-корня.

При работе с деревьями часто необходимо что-то сделать с каждым из его узлов (вывод на экран, какой-либо подсчет).

Для этого пишется рекурсивная функция для обхода дерева, которую изначально вызывают с его корнем.

Для каждого из узлов она в определенном порядке выполняет три действия:

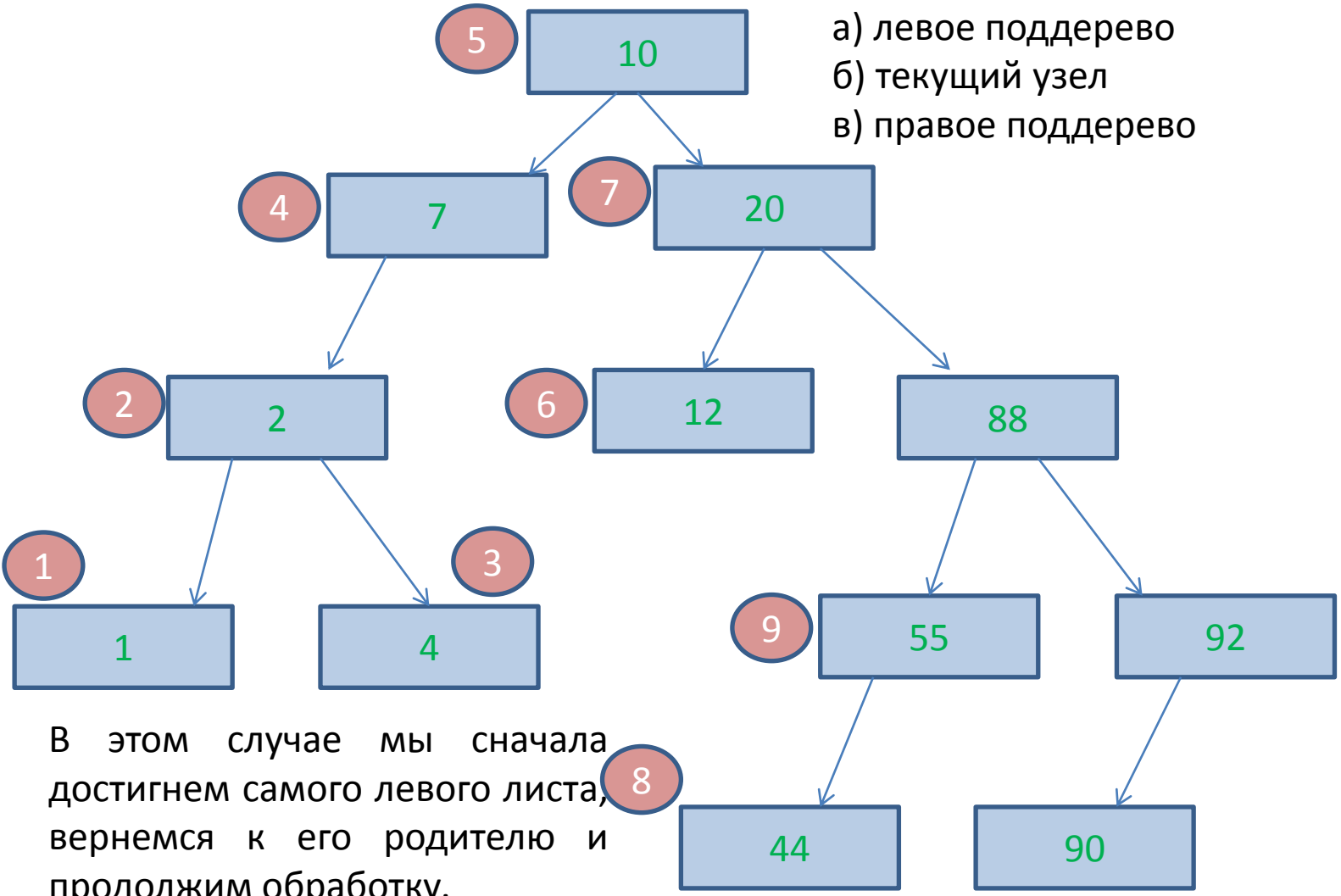
а) обрабатывает текущий узел

б) вызывает сама себя для левого поддерева текущего узла

в) вызывает сама себя для правого поддерева текущего узла

В зависимости от порядка этих действий (например, **абв, бва, вба**) мы по-разному обрабатываем наше дерево.

Симметричный обход:
а) левое поддерево
б) текущий узел
в) правое поддерево



В этом случае мы сначала достигнем самого левого листа, вернемся к его родителю и продолжим обработку.

Дерево будет обработано в порядке возрастания значений узлов.

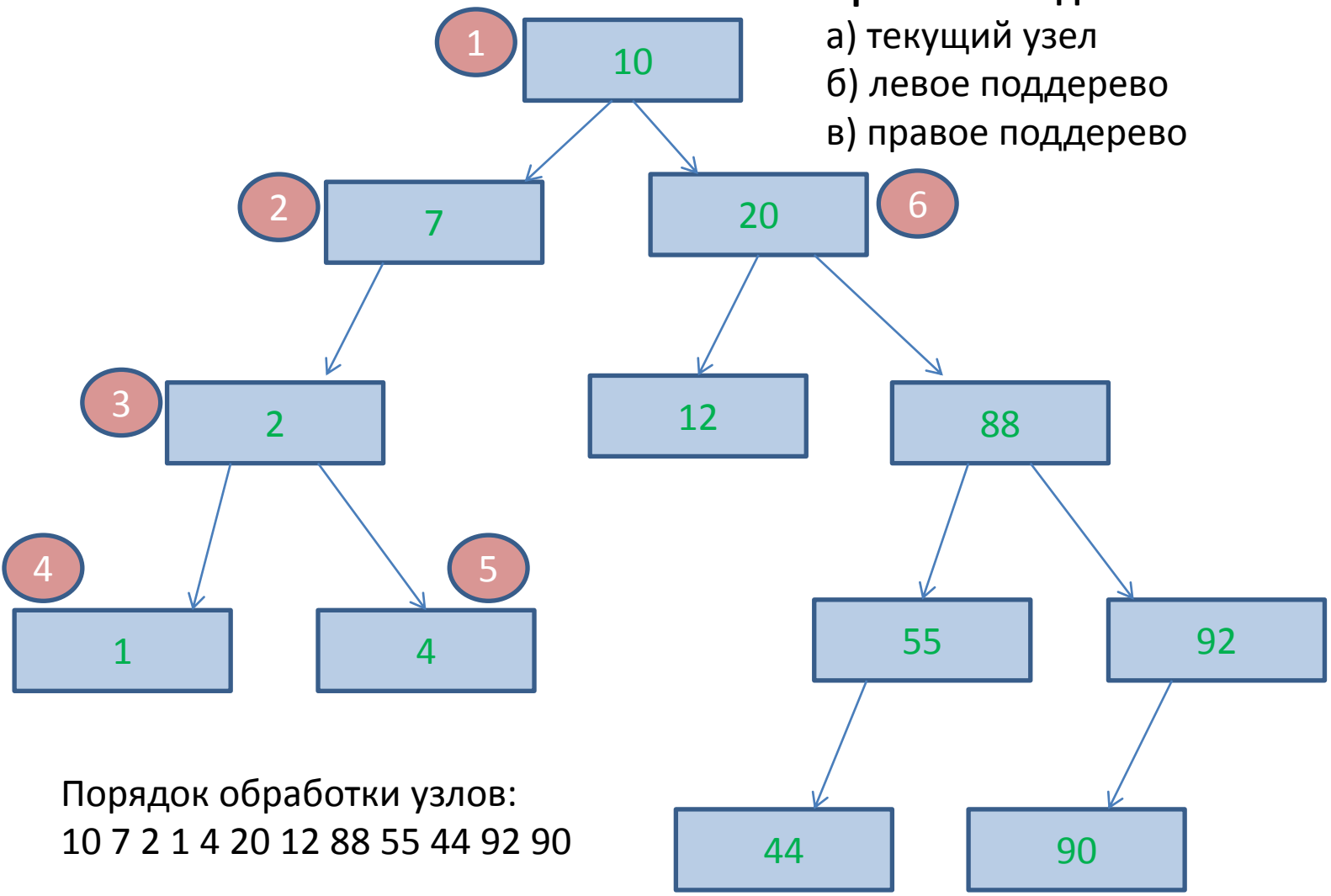
Порядок обработки узлов:
1 2 4 7 10 12 20 44 55 88 90 92

Прямой обход:

а) текущий узел

б) левое поддерево

в) правое поддерево



Порядок обработки узлов:

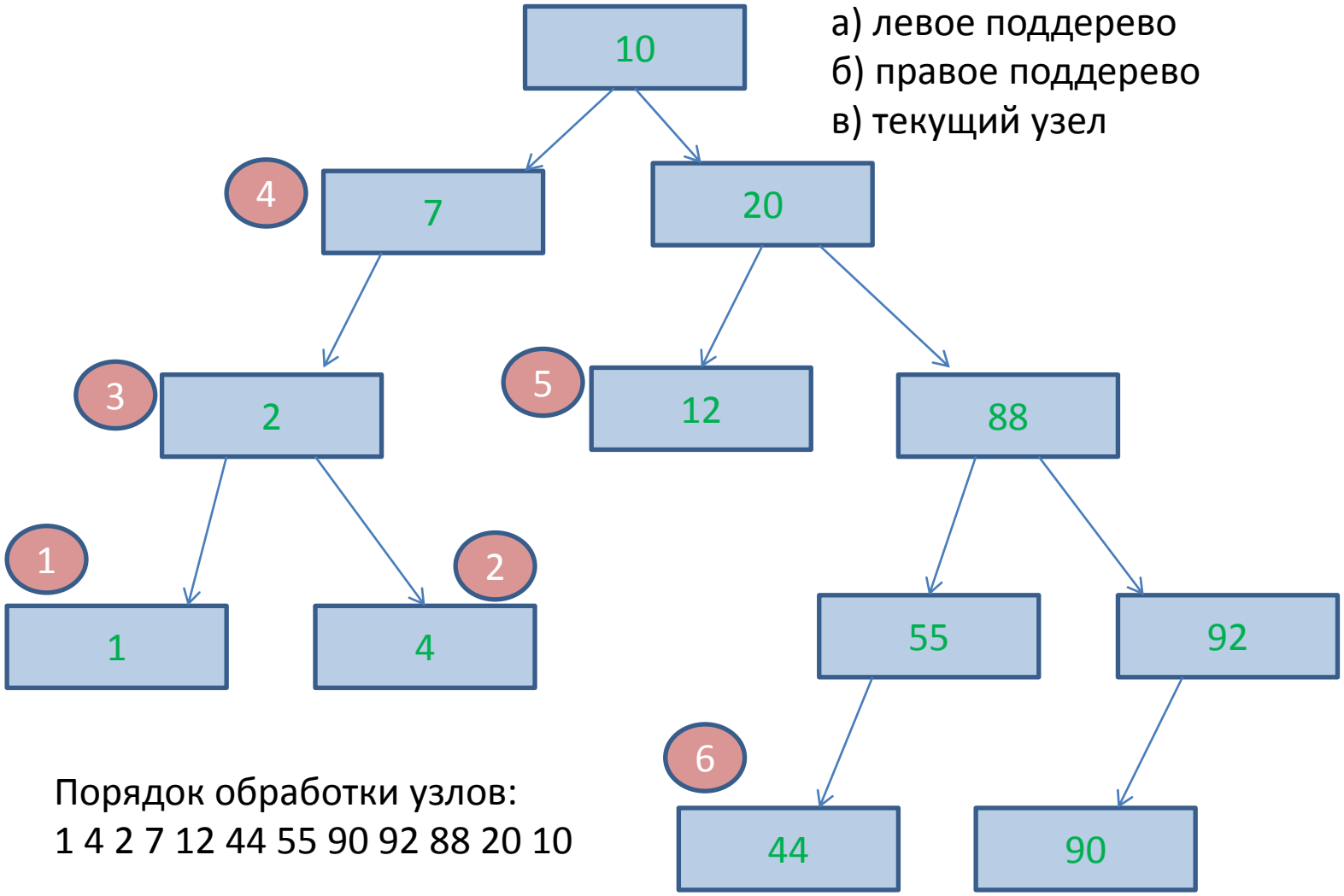
10 7 2 1 4 20 12 88 55 44 92 90

Обратный обход:

а) левое поддерево

б) правое поддерево

в) текущий узел



Порядок обработки узлов:

1 4 2 7 12 44 55 90 92 88 20 10

Этот обход используется, например, при очистке памяти, занимаемой деревом.

Возможны и другие варианты обхода.

Например, для обработки узлов в порядке убывания значений можно воспользоваться вот таким обходом:
а) правое поддерево
б) текущий узел
в) левое поддерево