OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Подробнее о контейнерах и вводе-выводе

Андрей Рыжиков

@ryzhikovas

Контейнеры - решение рутинных задач

Ежедневно мы решаем однотипные задачи.

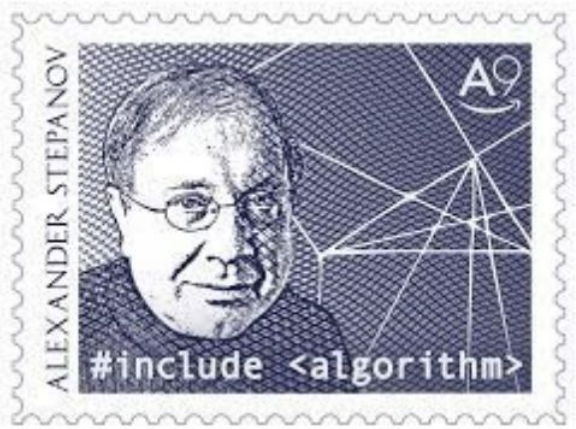
- Картотека сотрудников. Библиотека.
- Доставка еды. Маршруты автобусов.
- Бронирование отелей…

- **Не база данных!**

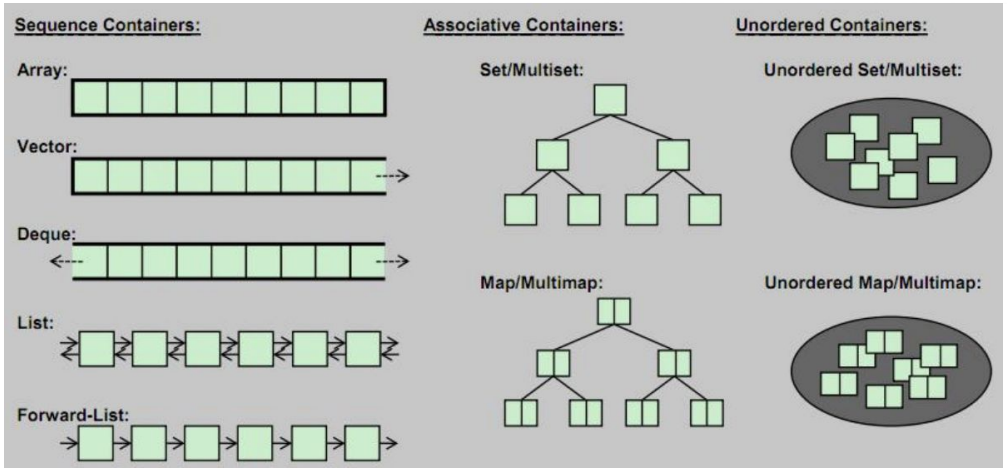
- Велосипедим что-то свое.
- Обобщаем.
- Поддерживаем.
- Устаём. Понимаем, что неэффективно.
- Переходим на std и boost.

Стандартная библиотека шаблонов

- Идеи реализованы во время работы А. Степанова в компании HP, затем в SGI (совместно с Менг Ли).
- 1994 — включена в официальный стандарт C++

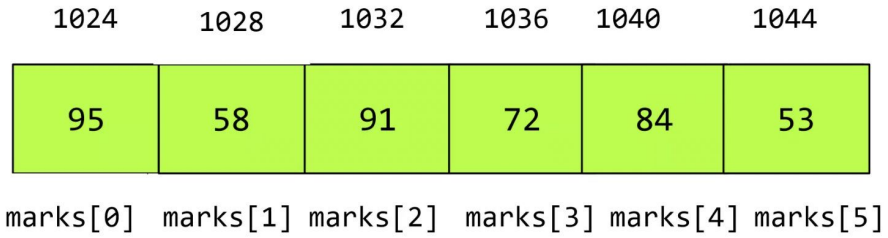


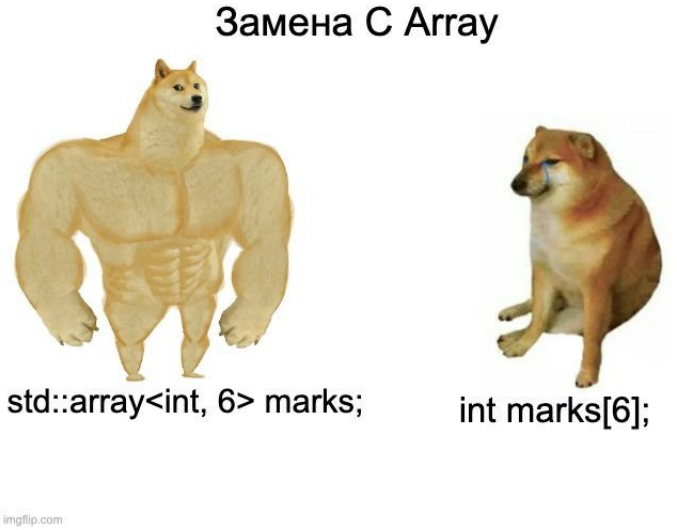
- Контейнеры
- Алгоритмы
- Итераторы



std::array<class, std::size_t>

- **Статический** массив **фиксированной** длины
- Расположен в **непрерывном** сегменте памяти





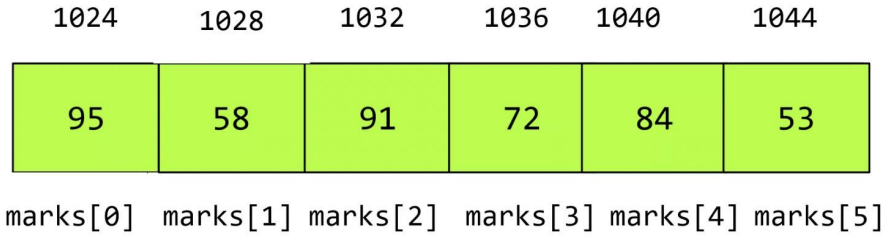
- **Эффективность**:

  линейность, прямой доступ, скорость доступа, кэш процессора

- **Применение**:
  - временный буфер
  - основа для более сложных структур

std::vector<class, …>

- **Динамический** массив. Количество элементов можно менять.
- Расположен в **непрерывном** сегменте **кучи**



- **Эффективность**:

  линейность, прямой доступ, скорость доступа, кэш процессора, вставка/удаление

- **Применение**:
  - нужен resize и быстрый доступ по индексу
  - не критична скорость вставки в произвольное место

std::vector<class, …>

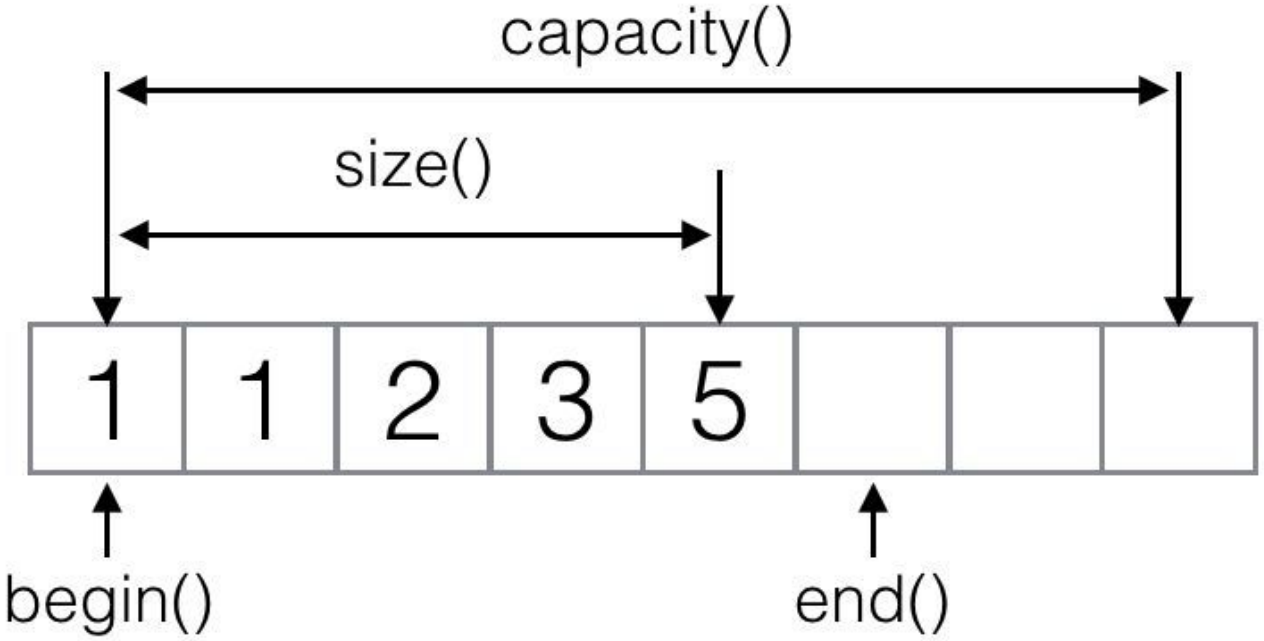
- **Функциональнее** std::array:
  - динамический размер
  - может быть очень большим

- **Но какова цена?**
  - дороже в рантайме

- Ключ к правильному выбору контейнера для конкретной задачи - понимание **внутреннего устройства** контейнера

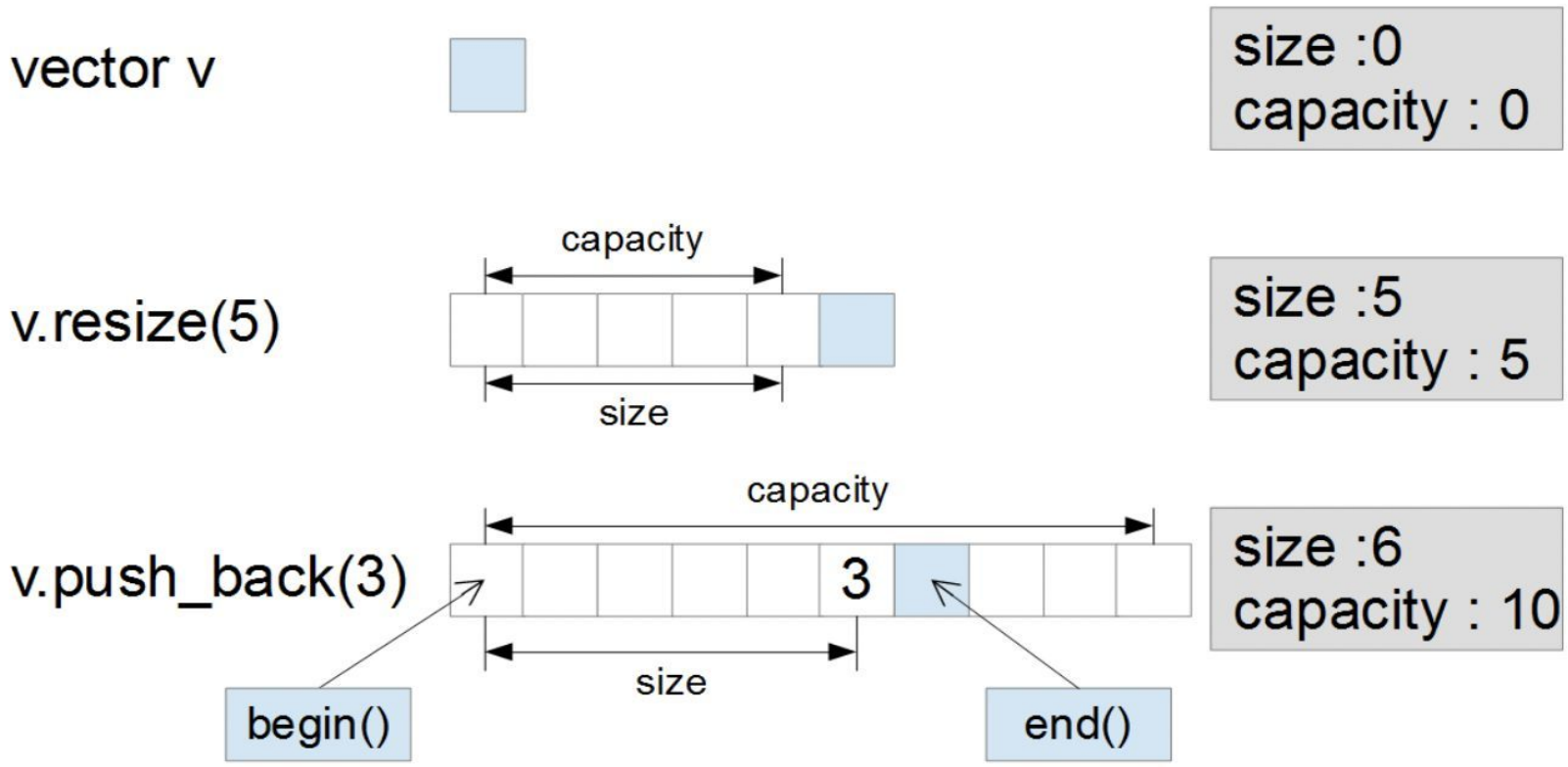
std::vector<class, …>

Структура памяти



std::vector<class, ...>

Структура памяти



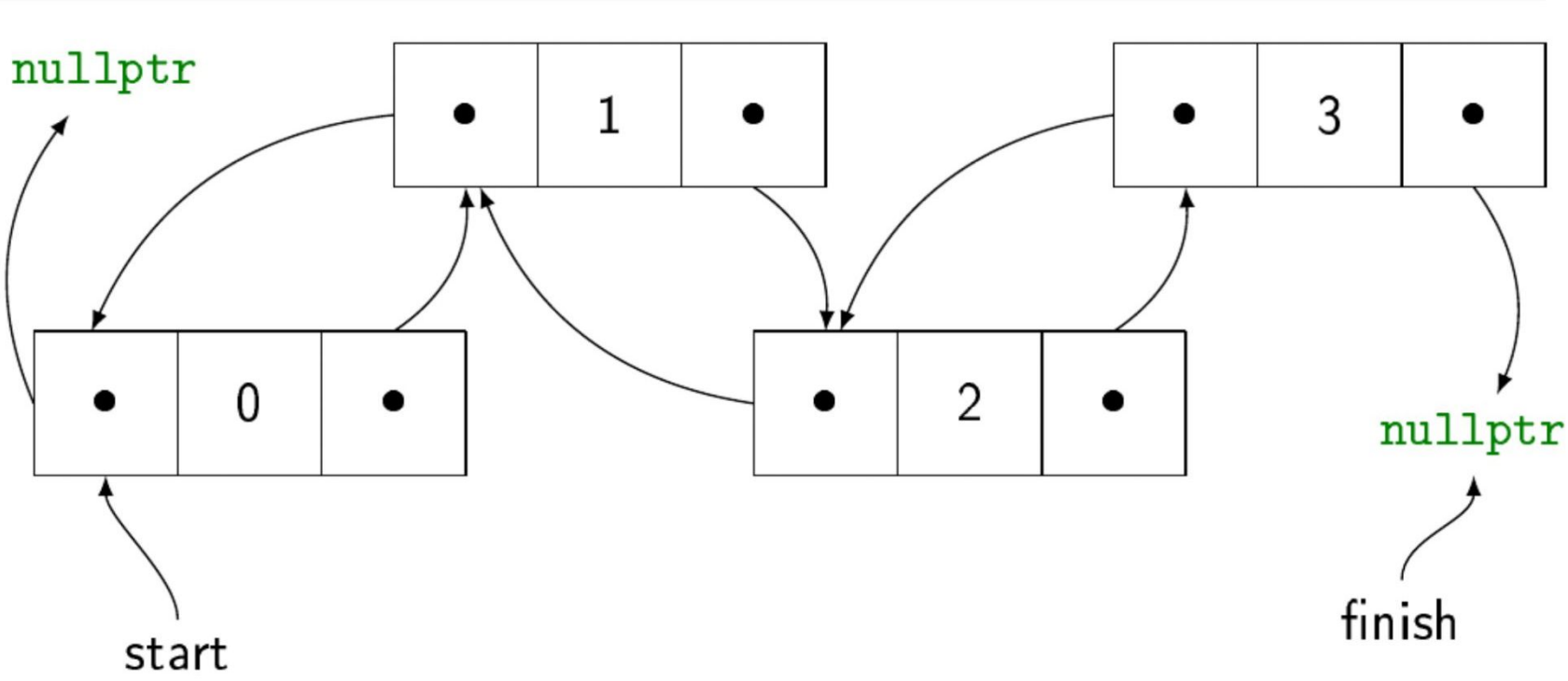
std::vector<class, ...>

- **Динамический** массив. **Эмулирует** расширяемость
- Расположен в **куче**
- **Непрерывен**: доступ по индексу **O(1)**, кэш-friendly
- **Непрерывен**: линеен, прямой доступ
- **Вставка**:
  - в конец **O(1)+**
  - иначе **O(N)**
- **Удаление**:
  - в конце **O(1)** (не освобождает память, см. shrink_to_fit)
  - иначе **O(N)**
- **Инвалидирует итераторы** (в т.ч. указатели) при перевыделении памяти

std::list<class, ...>

Двусвязный список

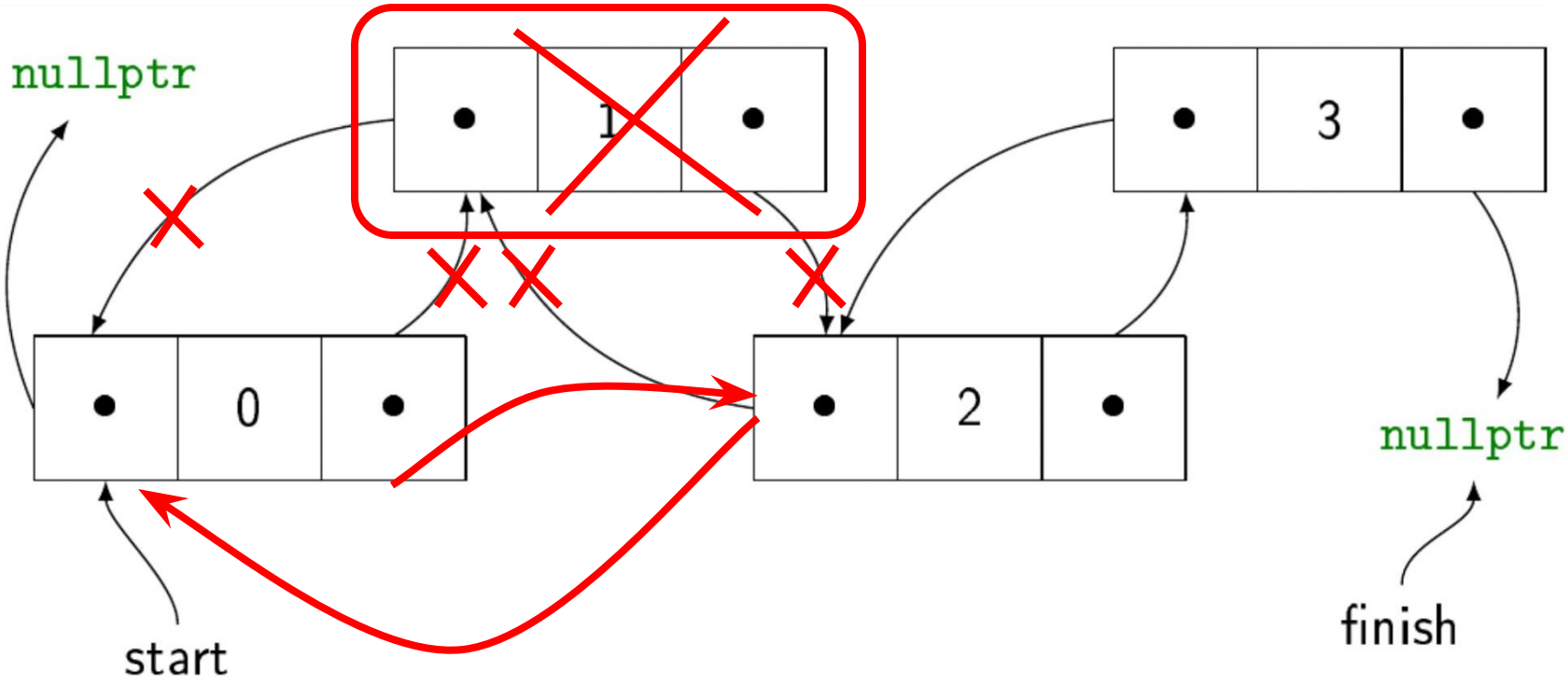
- К каждой записи прикреплено две ссылки: предыдущая и следующая записи



std::list<class, ...>

Вставка и удаление

- При удалении/вставке обновляем ссылки у соседей без переаллокации прочих элементов



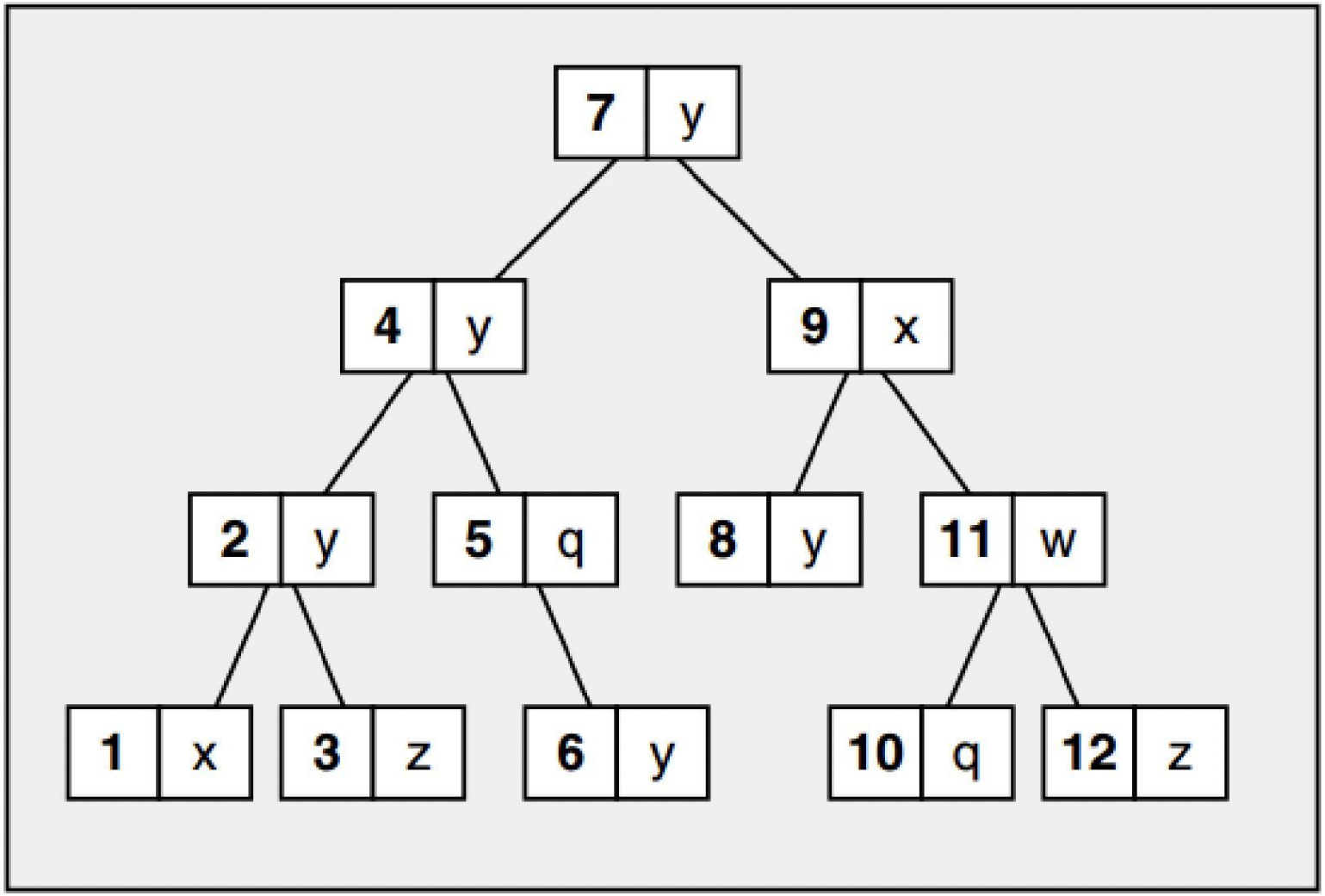
std::list<class, …>

- Упорядоченный
- Расположен в **куче, нелинейно**
- **Нет непрерывности данных**: произвольный доступ **O(N),** медленный обход
- **Вставка O(1)**
- **Удаление O(1)**
- **Инвалидирует итераторы** только удаленных элементов

std::map<class K, class Val, class Cmp, ...>

- **Ассоциативный** контейнер - хранит ключ-значение
- Данные **отсортированы** по ключу
- Ключи **уникальны**
- Расположен в **куче, нелинейно**
- Произвольный доступ по “индексу” **O(N)**
- Вставка/поиск/удаление **O(log N)**
- Обязателен **comparator** для ключа

std::map<class K, class Val, class Cmp, ...>



Что выбрать?

Единый интерфейс может спровоцировать неправильный выбор контейнера

- Если нужен специфичный интерфейс (**уверены?**) - выбор очевиден
- Быстрый маленький буфер известного размера - **std::array**
- Иначе, если нужен произвольный доступ
- к элементам - std::vector
- Не нужно искать и вставлять в начало/середину - std::vector
- Много вставок, не нужно искать по ключу - std::list
- Иначе std::map или …
- We need to go deeper

А есть что-то ещё?

А есть что-то ещё?

- **std::set** - как std::map, но только ключи, без значений

  … std::map привязывает к ключу значение

- **std::multiset и std::multimap** - как std::set / std::map, но ключи не уникальны

  … в std::set / std::map невозможно поместить два одинаковых ключа

- **std::unordered_set и std::unordered_map** (multi) - под капотом хеш-таблицы вместо дерева

  … вместо компаратора нужна хэш-функция

  *… иная асимптотика. Доступ по ключу - константное время **в среднем***

  … часто работает правило - если не нужна сортировка в map/set - замени на unordered

- **std::deque** ~ list<fixed-size-vector> + header

  … доступ по индексу **O(1),** но **медленнее std::vector**

  … вставка в конец и **начало O(1)**

- **А еще адаптеры:** stack, queue, priority_queue, flat_map…

Заключение

Вопросы?

Заполните, пожалуйста,
опрос о занятии
по ссылке в чате