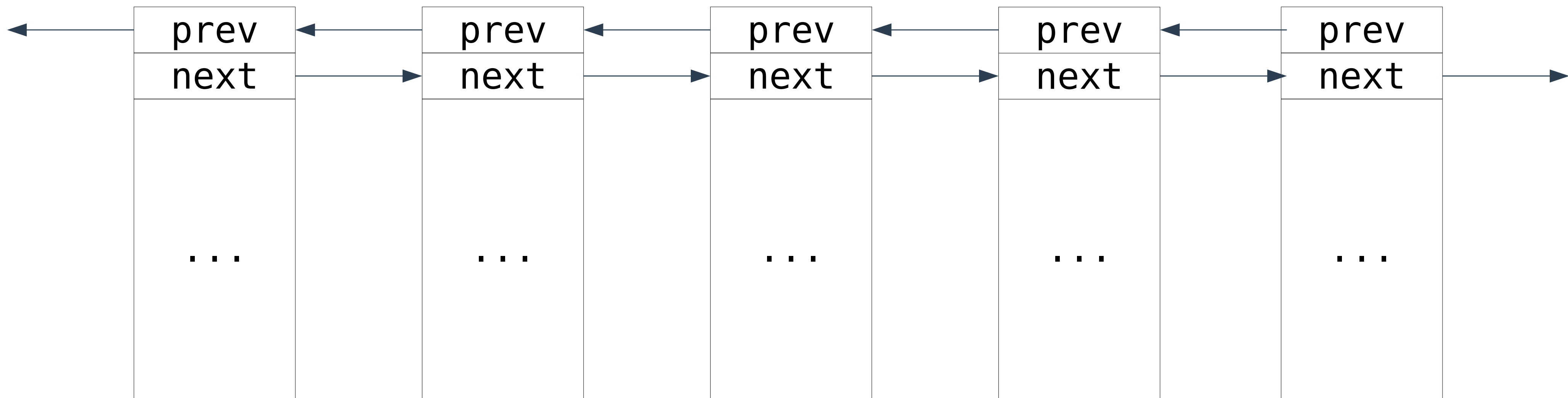
Intrusive Containers

План

- Наблюдение про двусвязные списки
- Примеры
- Попытка вынести общий код (сделать контейнер)
- Обсуждение результата

unit



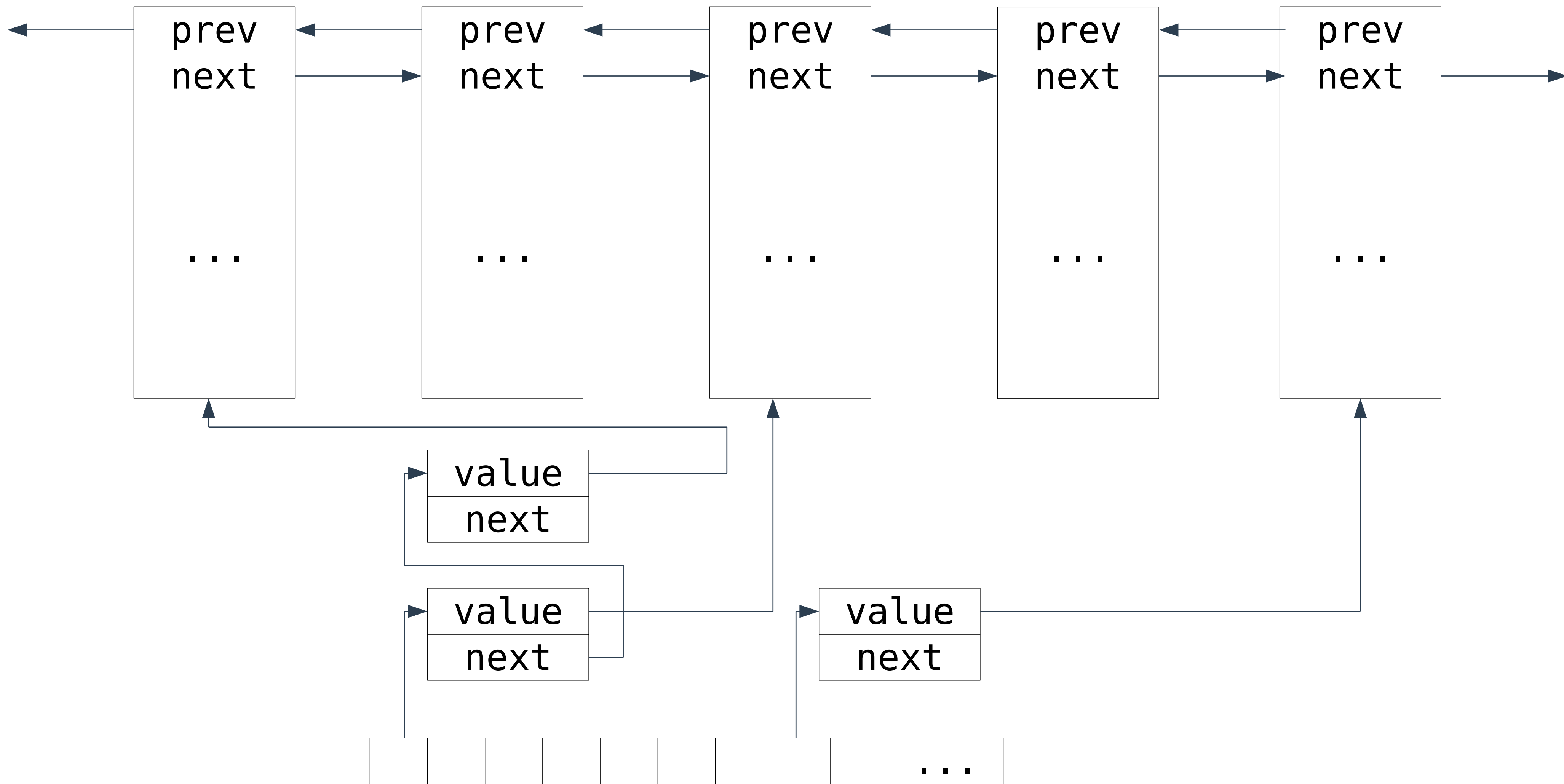

```
struct unit
{
    unit* prev;
    unit* next;
    ...
};
```

Подмножество элементов

```
struct unit
{
    unit* prev;
    unit* next;
    ...
};

std::unordered_set<unit*> selected;
```

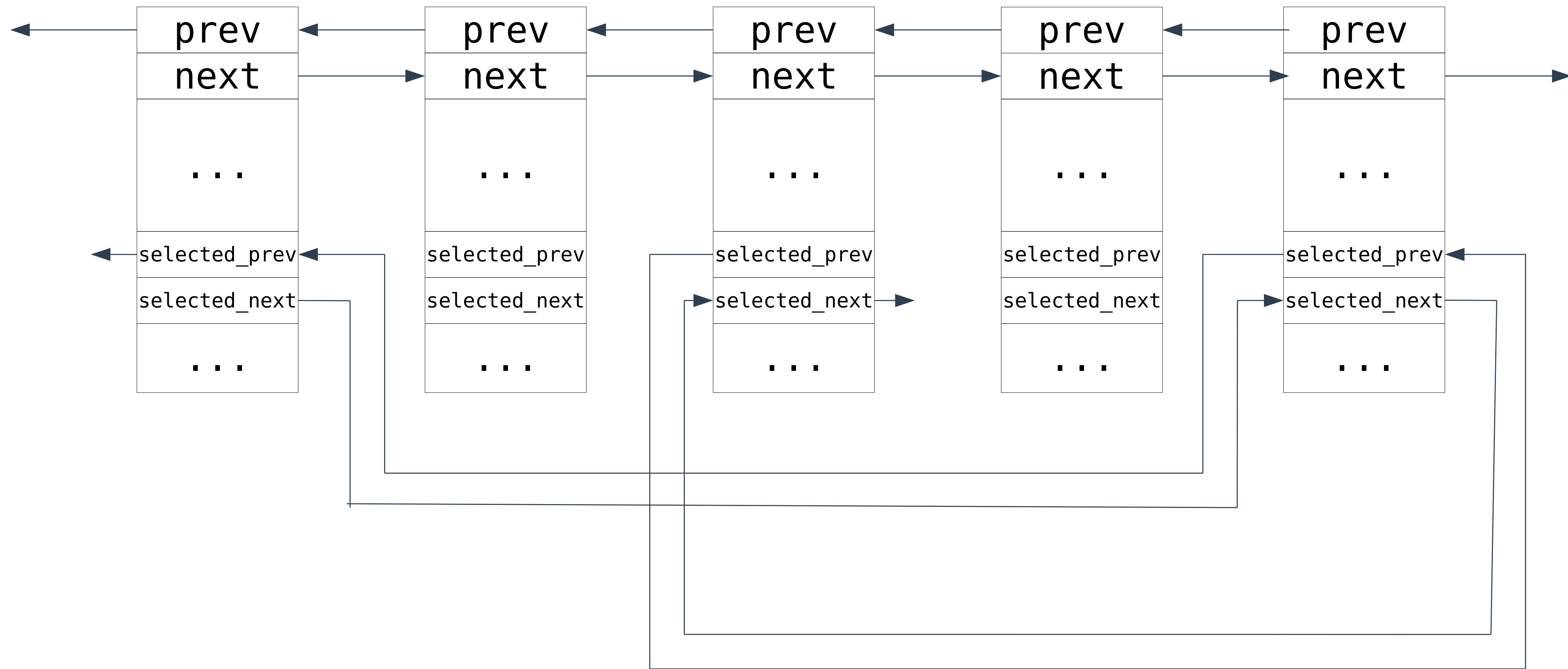
Подмножество элементов (2)



Подмножество элементов (3)

```
struct unit
{
    unit* prev;
    unit* next;
    ...
    unit* selected_prev;
    unit* selected_next;
    ...
};
```

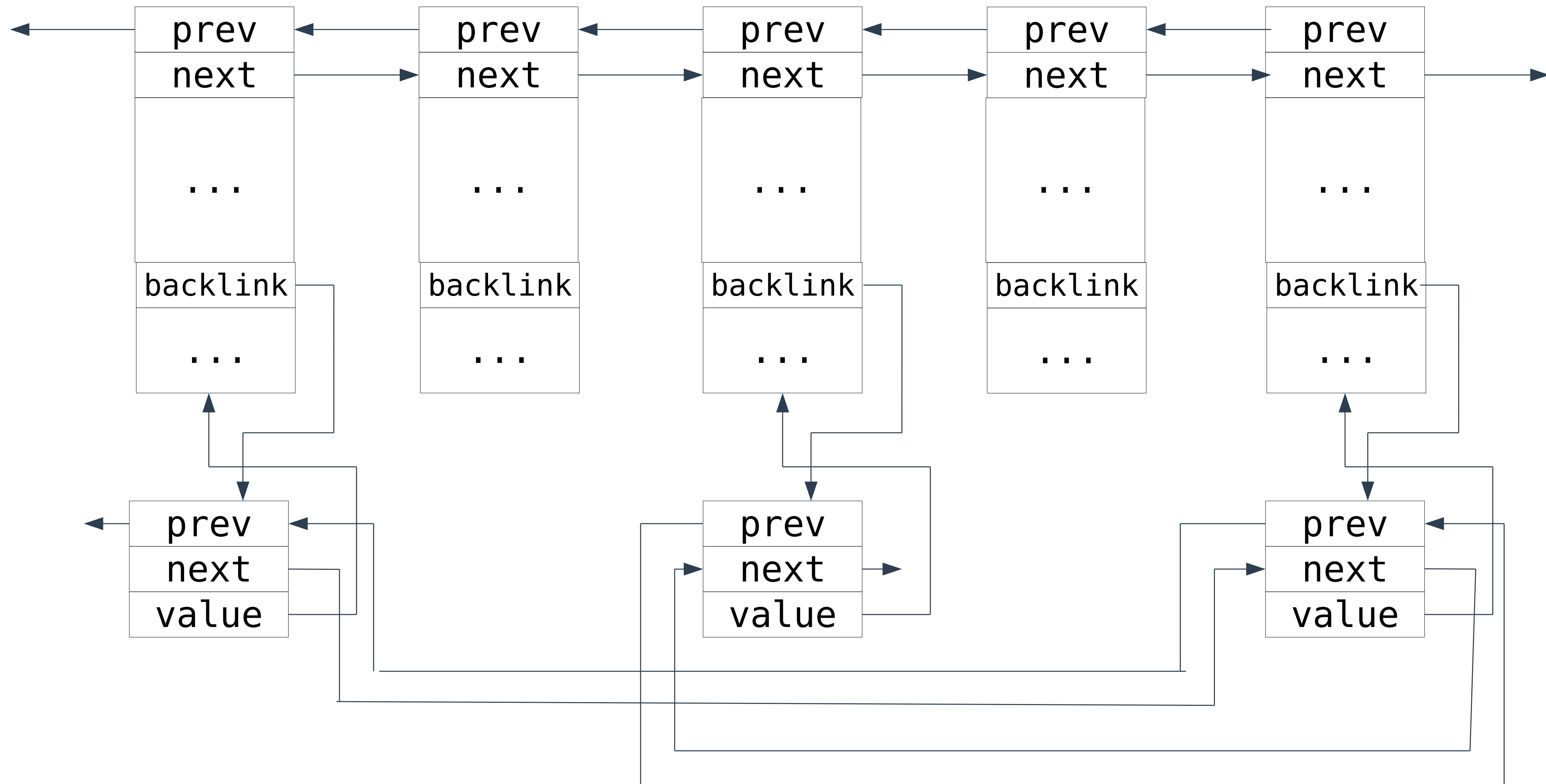
Подмножество элементов (4)



Бонусы

- Нет аллокаций/освобождений памяти при вставке/удалении
- Возможно использовать двусвязный список вместо хеш-таблицы
- Лучшая локальность ссылок при итерации по элементам

Обычные списки

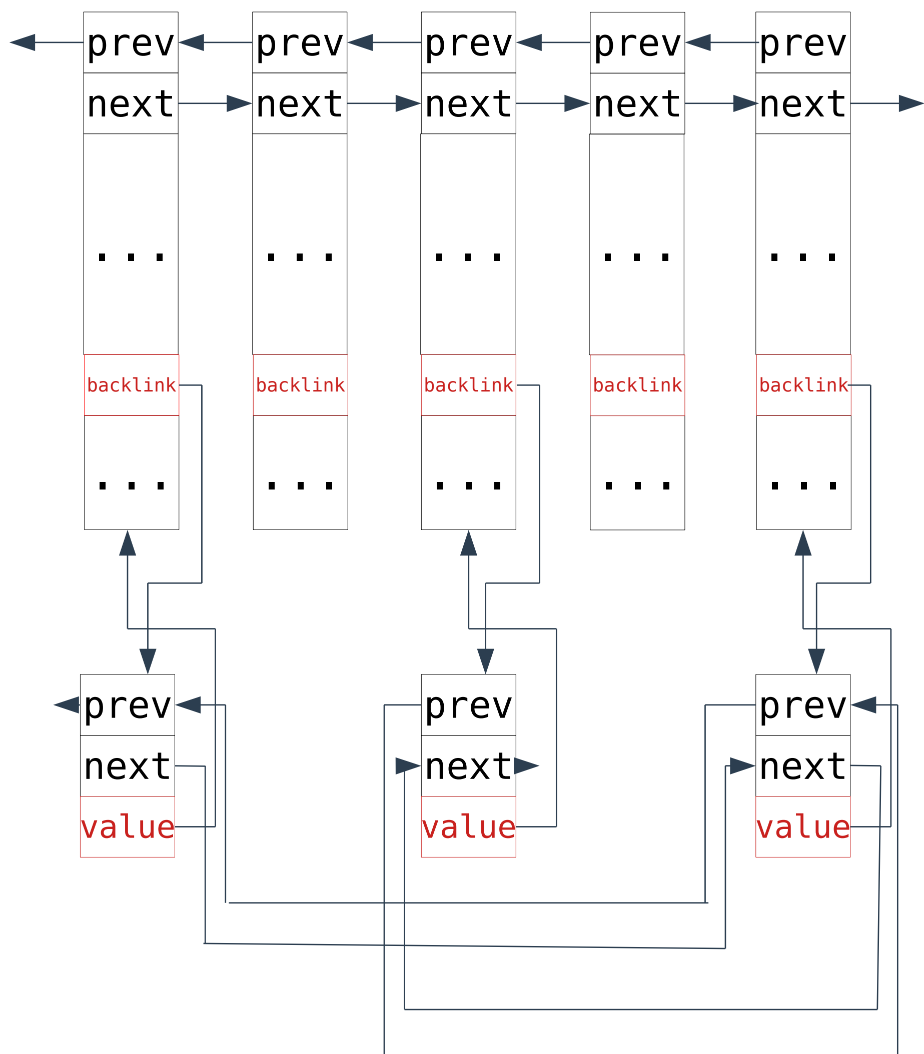


Обычные списки (2)

```
struct unit
{
    ...
    std::list<unit*>::iterator backlink;
    ...
};

std::list<unit*> selected;
```

Сравнение



Использование памяти

- Меньшее использование памяти
  - Если много элементов добавлено в список
- Большее использование памяти
  - Если мало элементов добавлено в список

- Всё равно может быть выгодно за счет накладных расходов аллокатора памяти

Замечание

Всё сказанное обобщается на другие структуры данных:

- Двусвязные списки
- Деревья
- Хеш-таблицы использующие списки для разрешения коллизий

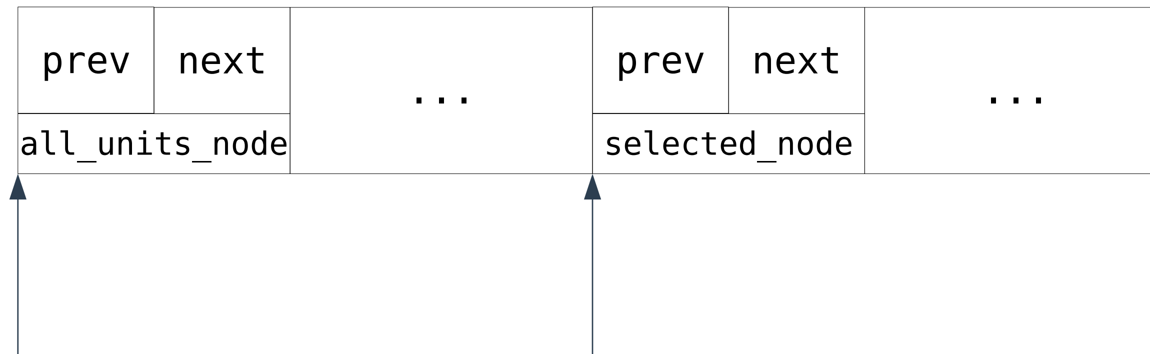
Пример: LRU-кеш

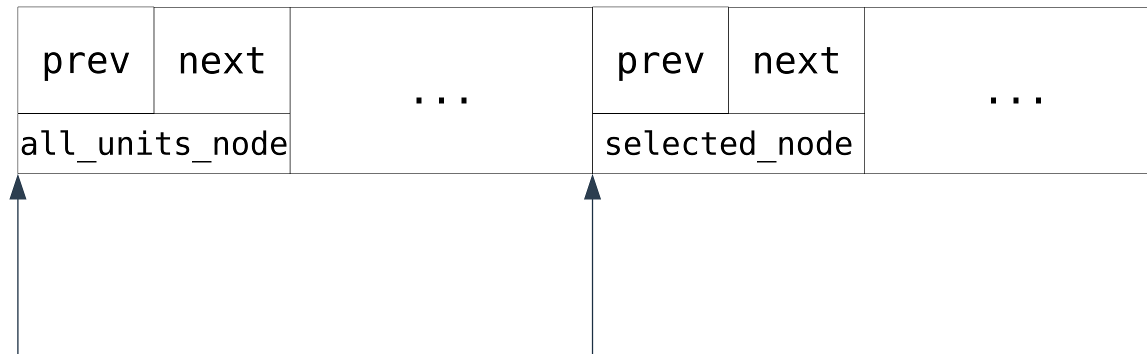
Пример узел LRU-кеша

C-style подход

```
struct list_element
{
    list_element* prev;
    list_element* next;
};

struct unit
{
    list_element all_units_node;
    ...
    list_element selected_node;
    ...
};
```


unit
prev
next
...
all_units_node
prev
next
...
selected_node




```
#define container_of(ptr, type, member)                          \
    ((type*)((char*)(ptr) - offsetof(type, member)))
```