

Paralelní a distribuované výpočty (B4B36PDV)

Branislav Bošanský, Michal Jakob

bosansky@fel.cvut.cz

Artificial Intelligence Center
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

Dnešní přednáška

Motivace



Dnešní přednáška

Konkurentní datové struktury

Vícero vláken chce přistupovat ke společné datové struktuře

- např. zásobník, fronta, spojový seznam
- binární vyhledávací strom, jiné vyhledávací stromy
- haldy
- ...

Dnešní přednáška

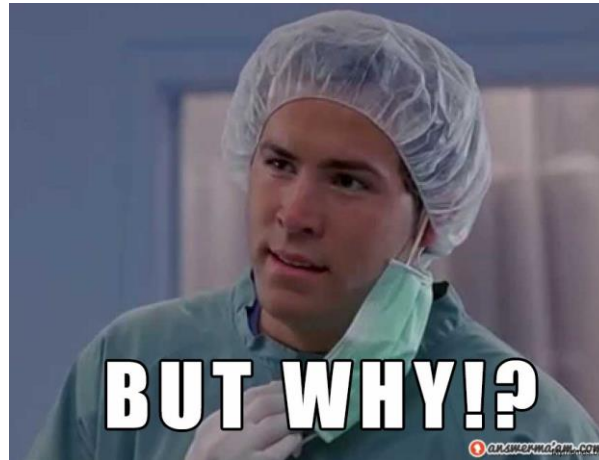
Konkurentní datové struktury

- Potřebujeme zabezpečit korektní operace s datovou strukturou
 - Vlákna chtějí současně vkládat, mazat, nebo vyhledávat

Dnešní přednáška

Konkurentní datové struktury

- Potřebujeme zabezpečit korektní operace s datovou strukturou
 - Vlákna chtějí současně vkládat, mazat, nebo vyhledávat



- Máme zámky, strukturu zamkneme a máme garantovanou konzistenci ...

Dnešní přednáška

Konkurentní datové struktury

- Potřebujeme zabezpečit korektní operace s datovou strukturou
 - Vlákna chtějí současně vkládat, mazat, nebo vyhledávat
- Máme zámky, strukturu zamkneme a máme garantovanou konzistenci ...
- Chceme navrhnout co nejefektivnější práci více vláken nad společnou datovou strukturou
- Dnes si ukážeme, jak můžeme konzistenci zajistit bez zámků (tzv. lock-free datové struktury)

Konkurentní datové struktury

Co chceme dosáhnout?

- Hlavní myšlenka

Vlákna optimisticky předpokládají, že vše bude v pořádku
(modifikace DS budou konzistentní)

- ... ale nemůžeme se na to spolehnout, takže

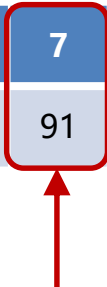
V případě detekce nekonzistence ji vlákno vyřeší/opraví

Konkurentní datové struktury

Příklad 1 – vyhledání maxima

- Vyhledání maxima v seznamu čísel
 - Chci najít maximální hodnotu a index na kterém se nachází
 - V případě rovnosti, chci co možná největší index

1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32



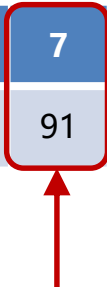
- Společná datová struktura
 - Dvě čísla – maximální hodnota & index

Konkurentní datové struktury

Příklad 1 – vyhledání maxima

- Vyhledání maxima v seznamu čísel
 - Chci najít maximální hodnotu a index na kterém se nachází
 - V případě rovnosti, chci co možná největší index

1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32



- Společná datová struktura
 - Dvě čísla – maximální hodnota & index
 - Jedno číslo – index aktuální maximální hodnoty
- Jak na to?

Konkurentní datové struktury

Příklad 1

- Vyhledání maxima v seznamu čísel
 - Chci najít maximální hodnotu a index na kterém se nachází
 - V případě rovnosti, chci co možná největší index
- První řešení - zámky

1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

max index

-1

Konkurentní datové struktury

Příklad 1

- Vyhledání maxima v seznamu čísel
 - Chci najít maximální hodnotu a index na kterém se nachází
 - V případě rovnosti, chci co možná největší index
- První řešení - zámky

Vlákno 1					Vlákno 2				
1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

max index

-1

Konkurentní datové struktury

Příklad 1

- Vyhledání maxima v seznamu čísel
 - Chci najít maximální hodnotu a index na kterém se nachází
 - V případě rovnosti, chci co možná největší index
- První řešení - zámky

Vlákno 1					Vlákno 2				
1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

max index

-1

Konkurentní datové struktury

Příklad 1

- Vyhledání maxima v seznamu čísel
 - Chci najít maximální hodnotu a index na kterém se nachází
 - V případě rovnosti, chci co možná největší index
- První řešení - zámky

Vlákno 1					Vlákno 2				
1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

- vlákno 1 porovná hodnotu
- pokud je aktuální hodnota větší
 - zamkne max index
 - provede úpravu
 - odemkne max index

max index
1

Konkurentní datové struktury

Příklad 1

- Vyhledání maxima v seznamu čísel
 - Chci najít maximální hodnotu a index na kterém se nachází
 - V případě rovnosti, chci co možná největší index
- První řešení - zámky

Vlákno 1					Vlákno 2				
1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

- vlákno 2 porovná hodnotu
- pokud je aktuální hodnota větší
 - zamkne max index
 - provede úpravu
 - odemkne max index

max index
6

Konkurentní datové struktury

Příklad 1

- Vyhledání maxima v seznamu čísel
 - Chci najít maximální hodnotu a index na kterém se nachází
 - V případě rovnosti, chci co možná největší index
- První řešení - zámky

Vlákno 1					Vlákno 2				
1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

- vlákno 2 porovná hodnotu
- pokud je aktuální hodnota větší
 - zamkne max index
 - provede úpravu
 - odemkne max index

max index
6

Jak to bude fungovat?

Konkurentní datové struktury

Příklad 1

- Může vzniknout nekonzistence

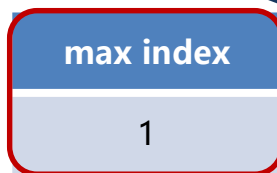
Vlákno 1					Vlákno 2				
1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

- vlákno 1 porovná hodnotu
- pokud je aktuální hodnota větší

- zamkne max index
- provede úpravu na hodnotu 1
- odemkne max index

- vlákno 2 porovná hodnotu
- pokud je aktuální hodnota větší

- zamkne max index
- provede úpravu na hodnotu 6
- odemkne max index



Konkurentní datové struktury

Příklad 1

- Může vzniknout nekonzistence

Vlákno 1					Vlákno 2				
1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

- vlákno 1 porovná hodnotu
- pokud je aktuální hodnota větší

- zamkne max index
- provede úpravu na hodnotu 1
- odemkne max index

- vlákno 2 porovná hodnotu
- pokud je aktuální hodnota větší

- zamkne max index
- provede úpravu na hodnotu 6
- odemkne max index

Výsledek může být nesprávný

max index
1

Konkurentní datové struktury

Příklad 1

- možné řešení:
 - vlákna budou zamykat max index před kontrolou – pomalé ☹
 - po získání zámku vlákno opět zkontroluje jestli je update aktuální

- vlákno 1 porovná hodnotu
- pokud je aktuální hodnota větší

- zamkne max index
- **zkontroluje, jestli je aktuální hodnota stále větší**
- provede úpravu na hodnotu 1
- odemkne max index

Vlákno 1					Vlákno 2				
1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

max index
1

Konkurentní datové struktury

Příklad 1

- Jde to i bez zámků?
- K nekonzistenci může dojít mezi kontrolou jestli je aktuální hodnota větší než v maximu a případnou výměnou
 - Necht' je aktuální hodnota **max index** == 2 a vlákno 2 testuje podmínku
 - Výměnu vykoná pouze tehdy, pokud je **max index** pořád 2

Vlákno 1					Vlákno 2				
1	2	3	4	5	6	7	8	9	10
3	12	42	91	74	24	91	66	19	32

Můžeme použít atomické proměnné

max index

2

Compare and Swap

- Atomická operace **compare and swap** (CAS)
 - Atomicky porovná jestli hodnota proměnné odpovídá očekávané hodnotě a pokud ano, provede změnu na novou hodnotu
- V C++, `compare_exchange_strong(expected, new)`

```
std::atomic_int atomic_max_index;

void find_max_cas(std::vector<int> & vector) {
#pragma omp parallel for num_threads(thread_count) shared(vector, atomic_max_index)
    for (int i=0; i<SIZE; ++i) {
        int tmp_index = atomic_max_index.load();
        while ((vector[i] > vector[tmp_index] || (vector[i] == vector[tmp_index] && i > tmp_index)) &&
            !atomic_max_index.compare_exchange_strong(tmp_index,i)) {
            tmp_index = atomic_max_index.load();
        }
    }
}
```

Compare and Swap

- Atomická operace **compare and swap** (CAS)
 - Atomicky porovná jestli hodnota proměnné odpovídá očekávané hodnotě a pokud ano, provede změnu na novou hodnotu
- V C++, `compare_exchange_strong(expected, new)`

podmínka pro maximum

```
std::atomic_int atomic_max_index;  
  
void find_max_cas(std::vector<int> & vector) {  
#pragma omp parallel for num_threads(thread_count) shared(vector, atomic_max_index)  
for (int i=0; i<SIZE; ++i) {  
    int tmp_index = atomic_max_index.load();  
    while ((vector[i] > vector[tmp_index] || (vector[i] == vector[tmp_index] && i > tmp_index)) &&  
           !atomic_max_index.compare_exchange_strong(tmp_index,i)) {  
        tmp_index = atomic_max_index.load();  
    }  
}  
}
```

hodnota indexu

Compare and Swap

Příklad 1

- Atomická operace **compare and swap** (CAS)
 - Atomicky porovná jestli hodnota proměnné odpovídá očekávané hodnotě a pokud ano, provede změnu na novou hodnotu
- V C++, `compare_exchange_strong(expected, new)`

```
std::atomic_int atomic_max_index;
```

```
void find_max_cas(std::vector<int> & vector) {
```

```
#pragma omp parallel for num_threads(thread_count) shared(vector, atomic_max_index)
```

```
for (int i=0; i<SIZE; ++i) {
```

```
int tmp_index = atomic_max_index.load();
```

```
while ((vector[i] > vector[tmp_index] || (vector[i] == vector[tmp_index] && i > tmp_index)) &&
```

```
!atomic_max_index.compare_exchange_strong(tmp_index,i)) {
```

```
tmp_index = atomic_max_index.load();
```

```
}
```

```
}
```

```
}
```

compare and swap – pokud je v proměnné **atomic_max_index** hodnota **tmp_index** (kterou jsme použili), tak provedeme update

Compare and Swap

Příklad 1

```
std::atomic_int atomic_max_index;

void find_max_cas(std::vector<int> & vector) {
#pragma omp parallel for num_threads(thread_count) shared(vector, atomic_max_index)
    for (int i=0; i<SIZE; ++i) {
        int tmp_index = atomic_max_index.load();
        while ((vector[i] > vector[tmp_index] || (vector[i] == vector[tmp_index] && i > tmp_index)) &&
!atomic_max_index.compare_exchange_strong(tmp_index,i)) {
            tmp_index = atomic_max_index.load();
        }
    }
}
```

výsledek je **true** pokud se operace povede, **false** v opačném případě

Compare and Swap

Příklad 1

```
std::atomic_int atomic_max_index;

void find_max_cas(std::vector<int> & vector) {
#pragma omp parallel for num_threads(thread_count) shared(vector, atomic_max_index)
  for (int i=0; i<SIZE; ++i) {
    int tmp_index = atomic_max_index.load();
    while ((vector[i] > vector[tmp_index] || (vector[i] == vector[tmp_index] && i > tmp_index)) &&
           !atomic_max_index.compare_exchange_strong(tmp_index,i)) {
      tmp_index = atomic_max_index.load();
    }
  }
}
```

Pokud je výsledek **false**, jiné vlákno mezitím změnilo **atomic_max_index**. Musíme aktualizovat hodnotu **tmp_index** a provést kontrolu znovu.

výsledek je **true** pokud se operace povede, **false** v opačném případě

Compare and Swap

Příklad 1

```
std::atomic_int atomic_max_index;

void find_max_cas(std::vector<int> & vector) {
#pragma omp parallel for num_threads(thread_count) shared(vector, atomic_max_index)
  for (int i=0; i<SIZE; ++i) {
    int tmp_index = atomic_max_index.load();
    while ((vector[i] > vector[tmp_index] || (vector[i] == vector[tmp_index] && i > tmp_index)) &&
           !atomic_max_index.compare_exchange_strong(tmp_index,i)) {
      tmp_index = atomic_max_index.load();
    }
  }
}
```

Pokud je výsledek **false**, jiné vlákno mezitím změnilo **atomic_max_index**. Musíme aktualizovat hodnotu **tmp_index** a provést kontrolu znovu.

výsledek je **true** pokud se operace povede, **false** v opačném případě

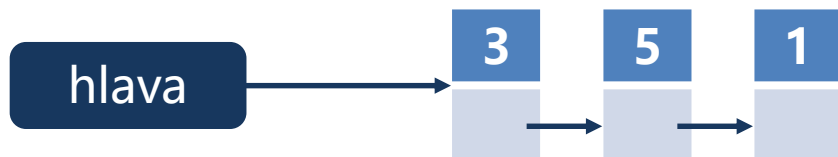
Kontrola je ve **while cyklu!**
(ne Konzistence se může vyskytnout opakovaně)

Konkurentní datové struktury

Příklad 2 – zásobník

Zásobník

```
struct Node {  
    int value = 0;  
    Node* successor = nullptr;  
  
    Node(int _value, Node* _successor) : value(_value), successor(_successor) {}  
};
```



Kritické místo je při vkládání a odebírání do/z vrcholu zásobníku

Konkurentní datové struktury

Příklad 2

Zásobník



Řešení pomocí zámků

```
void add_to_stack_locks(int new_value) {
    m.lock();
    head = new Node(new_value, head);
    m.unlock();
}

int pop_from_stack_locks() {
    m.lock();
    if (head == nullptr) {
        m.unlock();
        throw std::out_of_range("The stack is empty.");
        return -1;
    } else {
        Node* tmp = head;
        int val = head->value;
        head = head->successor;
        delete tmp;
        m.unlock();
        return val;
    }
}
```

Konkurentní datové struktury

Příklad 2

Řešení pomocí atomických proměnných



```
std::atomic<Node*> head2;

void add_to_stack_cas(int new_value) {
    Node* p = new Node(new_value, head2.load());
    while (!head2.compare_exchange_strong(p->successor, p)) {
        p->successor = head2.load();
    }
}

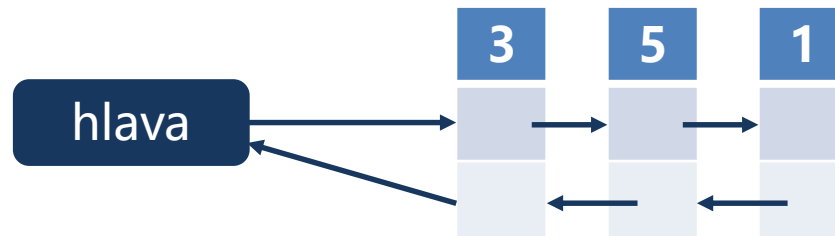
int pop_from_stack_cas() {
    if (head2.load() == nullptr) {
        throw std::out_of_range("The stack is empty.");
        return -1;
    } else {
        Node* h = head2.load();
        while (!head2.compare_exchange_strong(h, h->successor)) {
            h = head2.load();
        }
        int val = h->value;
        delete h;
        return val;
    }
}
```

Jak to bude fungovat?

Konkurentní datové struktury

Příklad 3

- Obousměrný spojový seznam



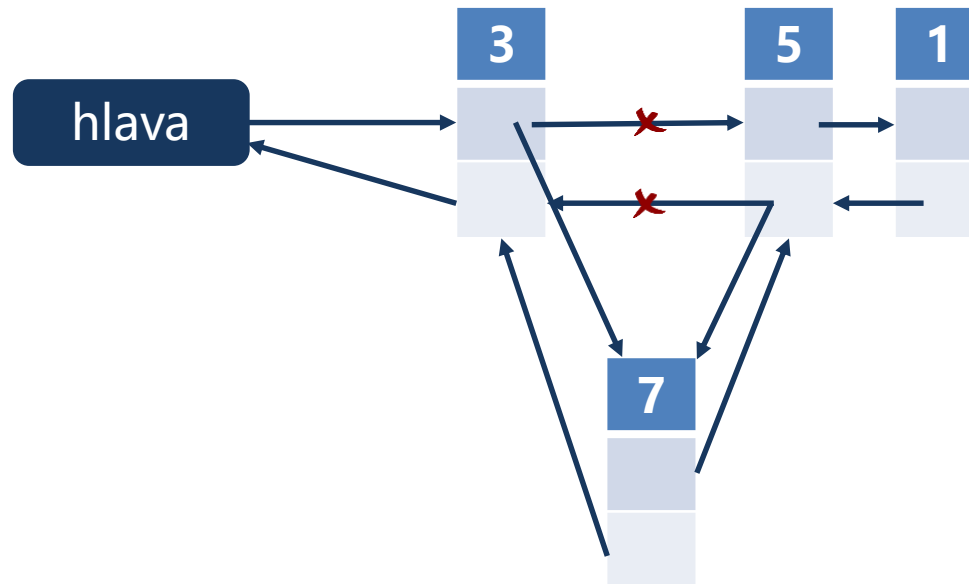
Operace přidání



Konkurentní datové struktury

Příklad 3

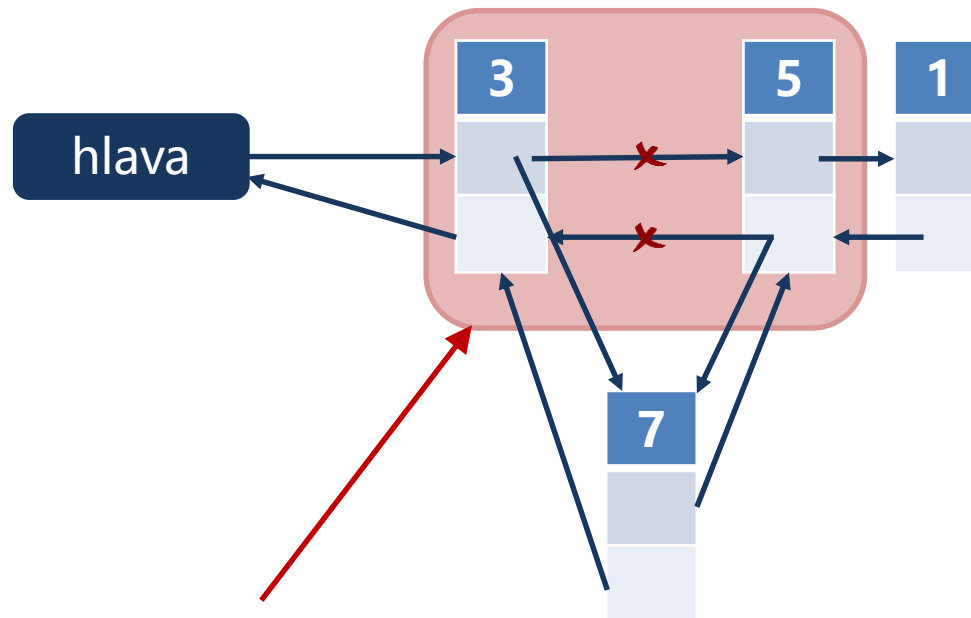
- Operace přidání pro obousměrný spojový seznam



Konkurentní datové struktury

Příklad 3

- Operace přidání pro obousměrný spojový seznam

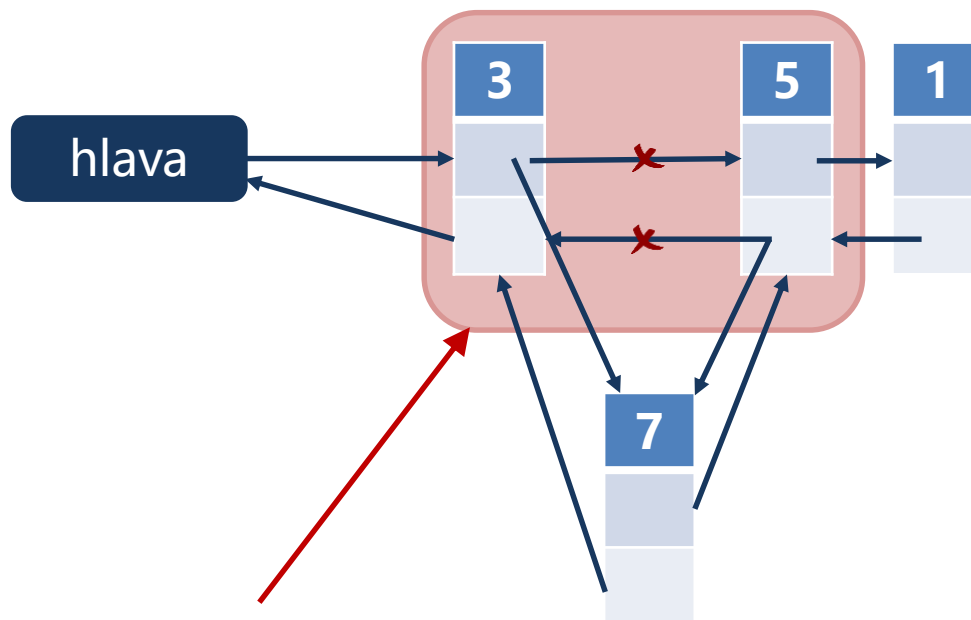


musíme zamknout oba prvky

Konkurentní datové struktury

Příklad 3

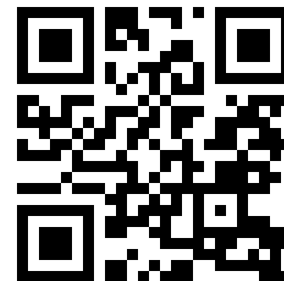
- Operace přidání pro obousměrný spojový seznam



musíme zamknout oba prvky

<https://goo.gl/a6BEMb>

Musíme zamykat oba současně nebo stačí vždy nejdřív prvek blíž k hlavě a pak jeho následovníka?



Konkurentní datové struktury

Příklad 3

- Řešení pomocí zámků

```
struct Node {  
    std::mutex m;  
    int value = 0;  
    Node* successor = nullptr;  
    Node* predecessor = nullptr;  
    Node(int _value, Node* _predecessor, Node* _successor) :  
        value(_value), predecessor(_predecessor), successor(_successor) {}  
};
```

Konkurentní datové struktury

Příklad 3

- Řešení pomocí zámků

```
Node* add_to_list_after(Node* _previous_node, int _new_value) {
    assert (_previous_node != nullptr);
    _previous_node->m.lock();

    Node* new_successor = _previous_node->successor;
    bool is_there_successor = new_successor != nullptr;

    if (is_there_successor) {
        new_successor->m.lock();
    }

    Node* new_node = new Node(_new_value, _previous_node, new_successor);

    if (is_there_successor)
        _previous_node->successor->predecessor = new_node;
    _previous_node->successor = new_node;

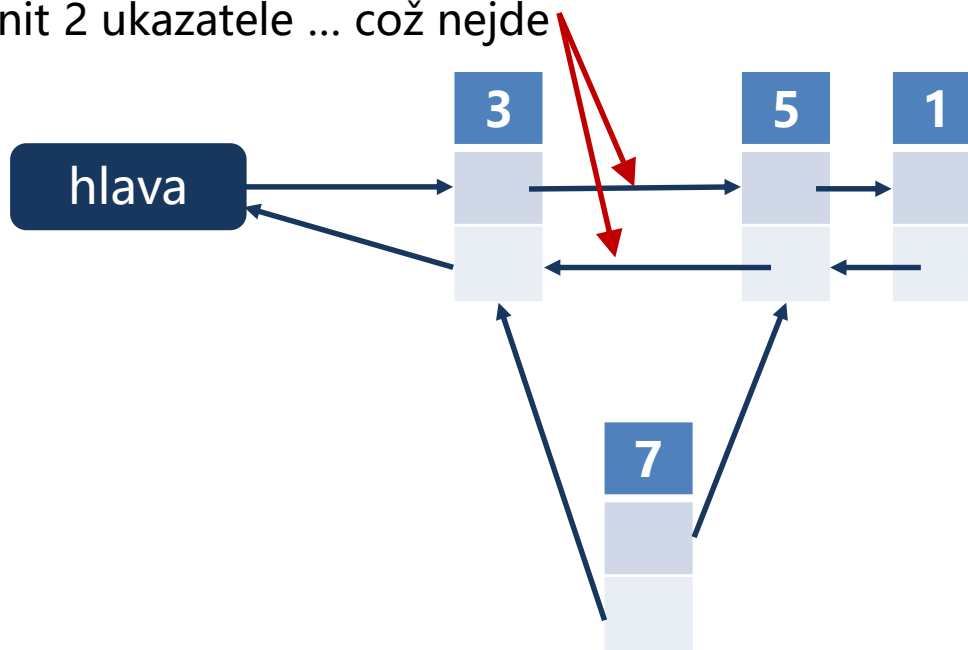
    if (is_there_successor)
        new_successor->m.unlock();
    _previous_node->m.unlock();
    return new_node;
}
```

Konkurentní datové struktury

Příklad 3

- Jak řešit pomocí atomických operací?

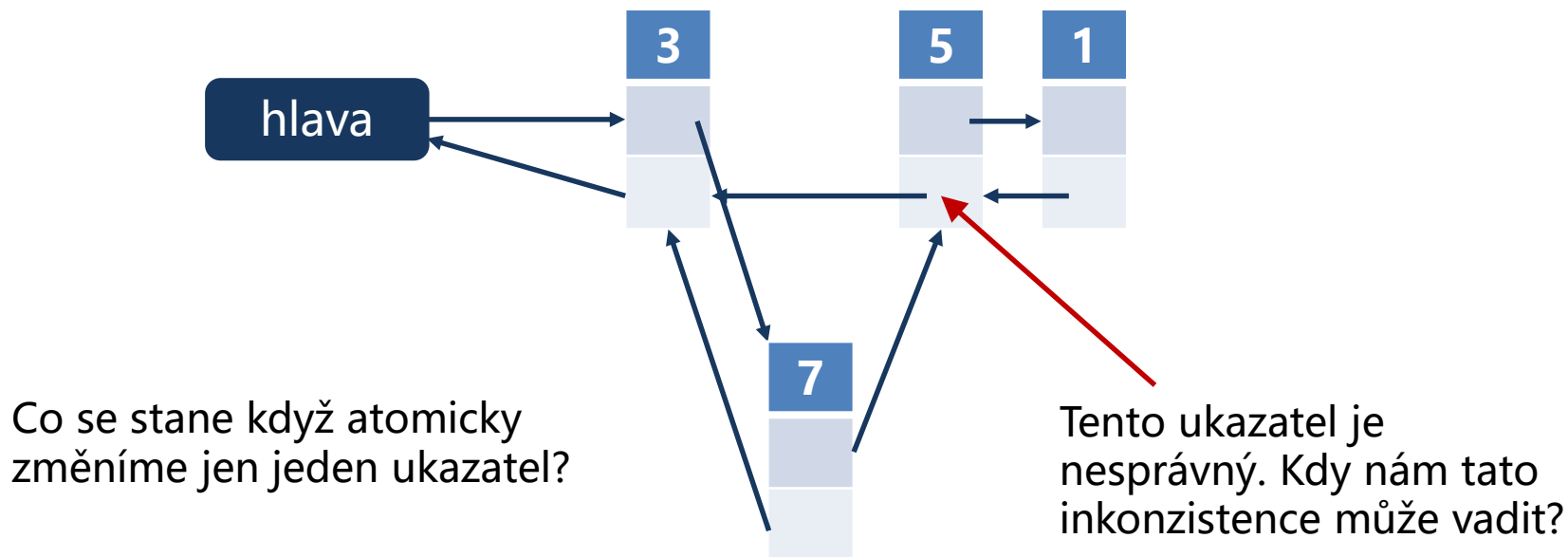
potřebovali bychom atomicky
změnit 2 ukazatele ... což nejde



Konkurentní datové struktury

Příklad 3

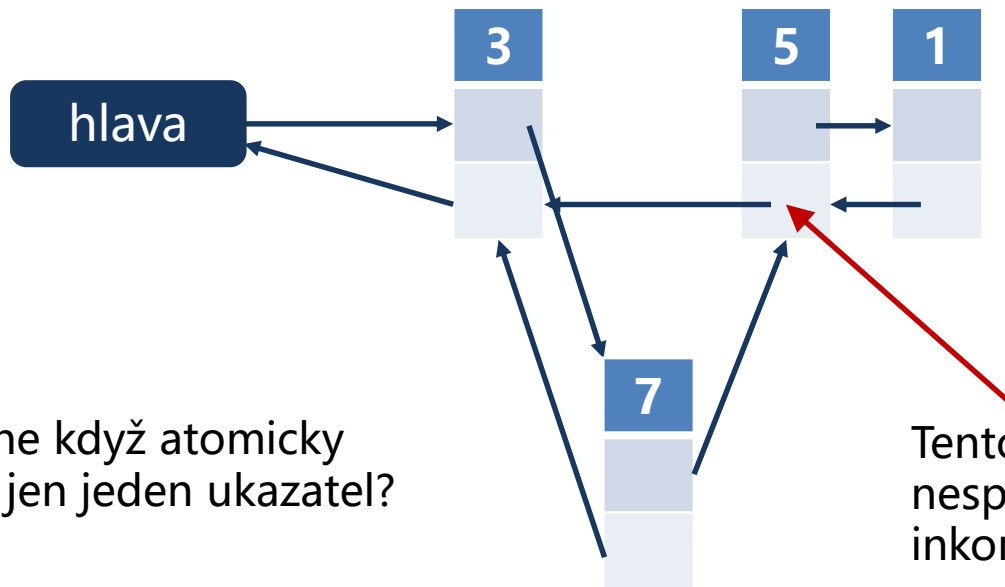
- Jak řešit pomocí atomických operací?



Konkurentní datové struktury

Příklad 3

- Jak řešit pomocí atomických operací?



Co se stane když atomicky změníme jen jeden ukazatel?

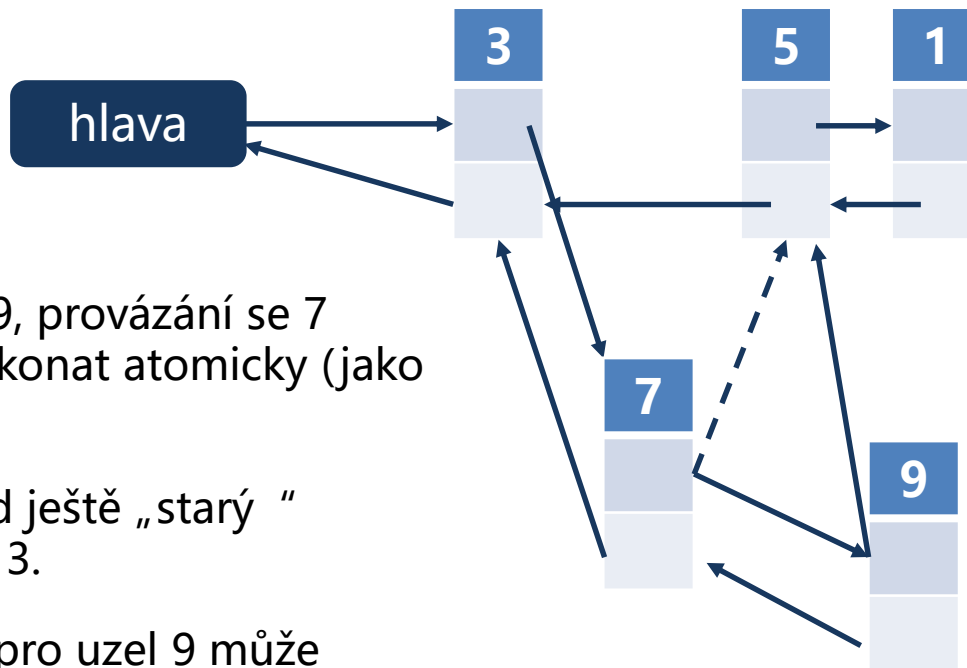
Tento ukazatel je nesprávný. Kdy nám tato inkonzistence může vadit?

Když chceme přidávat mezi 7 a 5.

Konkurentní datové struktury

Příklad 3

- Jak řešit pomocí atomických operací?



Přidáváme 9, provázání se 7
můžeme vykonat atomicky (jako
předtím).

U 5 je pořád ještě „starý“
ukazatel na 3.

Dokončení pro uzel 9 může
počkat, až bude v 5 správný
ukazatel (na 7).

Konkurentní datové struktury

Příklad 3

- Řešení pomocí CAS

```
struct AtomicNode {
    int value = 0;
    std::atomic<AtomicNode*> successor;
    std::atomic<AtomicNode*> predecessor;

    AtomicNode(int value) {
        successor.store(nullptr);
        predecessor.store(nullptr);
    }

    AtomicNode(int _value, AtomicNode* _predecessor, AtomicNode* _successor) : value(_value) {
        successor.store(_successor);
        predecessor.store(_predecessor);
    }
};
```

Konkurentní datové struktury

Příklad 3

- Řešení pomocí CAS

```
AtomicNode* atomic_add_to_list_after(AtomicNode* _previous_node, int _new_value) {
    assert (_previous_node != nullptr);

    AtomicNode* old_successor = _previous_node->successor;
    AtomicNode* new_node = new AtomicNode(_new_value, _previous_node, old_successor);

    while (!_previous_node->successor.compare_exchange_strong(old_successor, new_node)) {
        old_successor = _previous_node->successor;
        new_node->successor.store(old_successor);
    }

    if (old_successor != nullptr) {
        while (!old_successor->predecessor.compare_exchange_strong(_previous_node, new_node))
            ;
    }

    return new_node;
}
```


Konkurentní datové struktury

Příklad 3

- Řešení pomocí CAS

```
AtomicNode* atomic_add_to_list_after(AtomicNode* _previous_node, int _new_value) {
    assert (_previous_node != nullptr);

    AtomicNode* old_successor = _previous_node->successor;
    AtomicNode* new_node = new AtomicNode(_new_value, _previous_node, old_successor);

    while (!_previous_node->successor.compare_exchange_strong(old_successor, new_node)) {
        old_successor = _previous_node->successor;
        new_node->successor.store(old_successor);
    }

    if (old_successor != nullptr) {
        while (!old_successor->predecessor.compare_exchange_strong(_previous_node, new_node))
            ;
    }

    return new_node;
}
```

Změna ukazatele v
prvním prvku (před
vkládaným uzlem).

Konkurentní datové struktury

Příklad 3

- Řešení pomocí CAS

```
AtomicNode* atomic_add_to_list_after(AtomicNode* _previous_node, int _new_value) {
    assert (_previous_node != nullptr);

    AtomicNode* old_successor = _previous_node->successor;
    AtomicNode* new_node = new AtomicNode(_new_value, _previous_node, old_successor);

    while (!_previous_node->successor.compare_exchange_strong(old_successor, new_node)) {
        old_successor = _previous_node->successor;
        new_node->successor.store(old_successor);
    }

    if (old_successor != nullptr) {
        while (!old_successor->predecessor.compare_exchange_strong(_previous_node, new_node))
            ;
    }

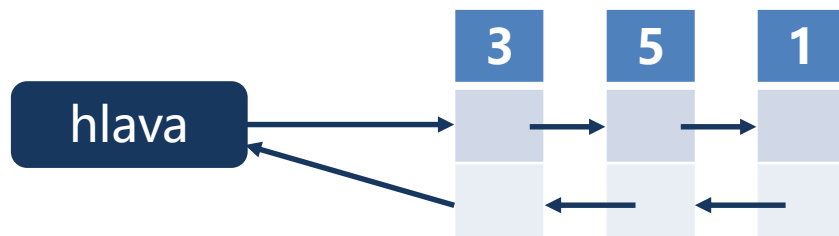
    return new_node;
}
```

Změna ukazatele v
druhého prvku (za
vkládaným uzlem).

Konkurentní datové struktury

Příklad 3

- Obousměrný spojový seznam



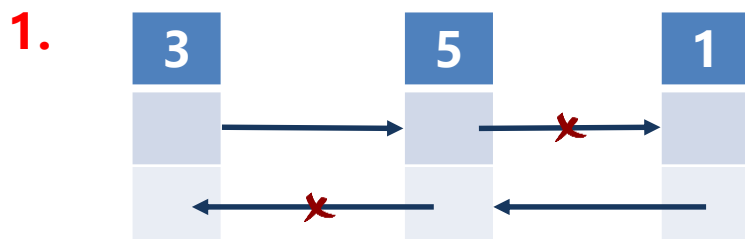
Operace mazání – komplexnější

Musíme označit které uzly (ukazatele) budou smazány

Konkurentní datové struktury

Příklad 3

- Mazání v obousměrném spojovém seznamu



2.

3.

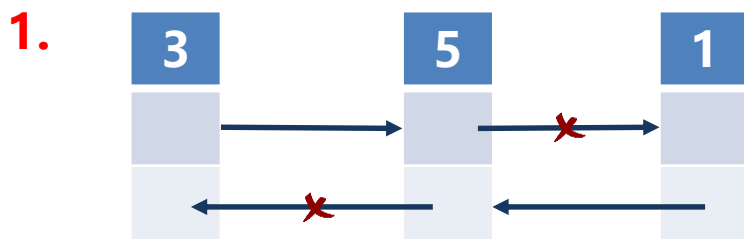
Provedeme následující atomické operace:

- Označíme ukazatele mazaného uzlu jako „ke smazání “

Konkurentní datové struktury

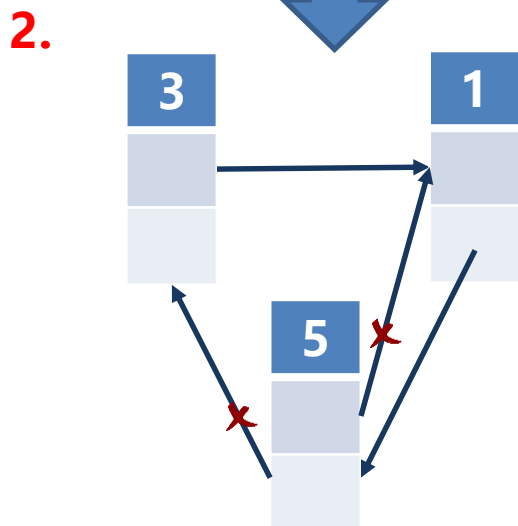
Příklad 3

- Mazání v obousměrném spojovém seznamu



Provedeme následující atomické operace:

- Označíme ukazatele mazaného uzlu jako „ke smazání“
- Převédeme ukazatel předchůdce

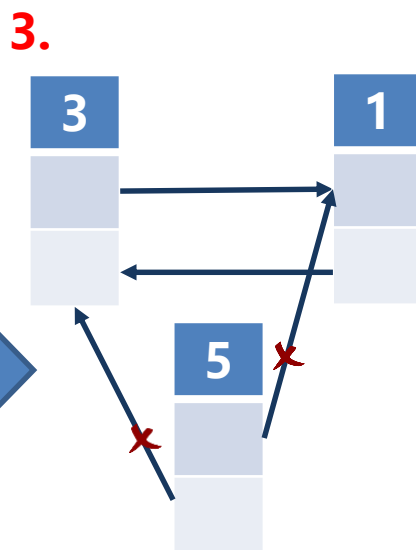
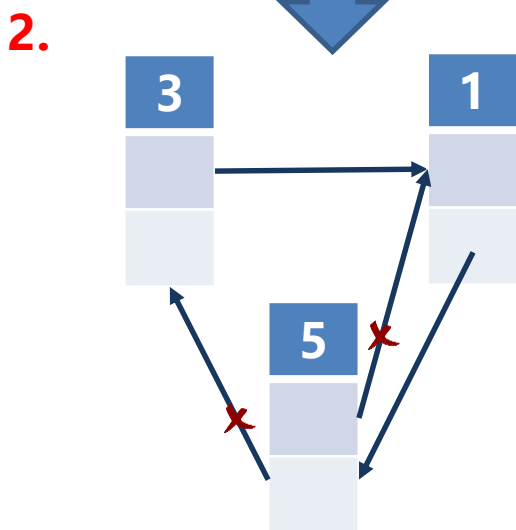
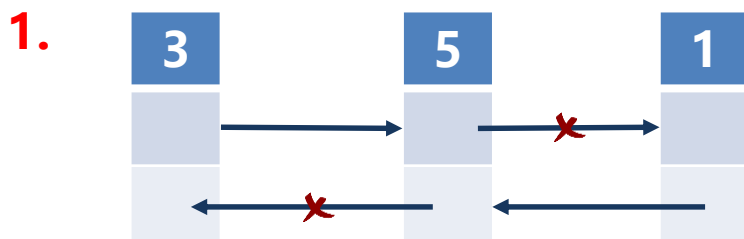


3.

Konkurentní datové struktury

Příklad 3

- Mazání v obousměrném spojovém seznamu



Provedeme následující atomické operace:

- Označíme ukazatele mazaného uzlu jako „ke smazání“
- Převédeme ukazatel předchůdce
- Převédeme ukazatel následovníka