# RI301
# Strukture podataka

dr.sc. Edin Pjanić

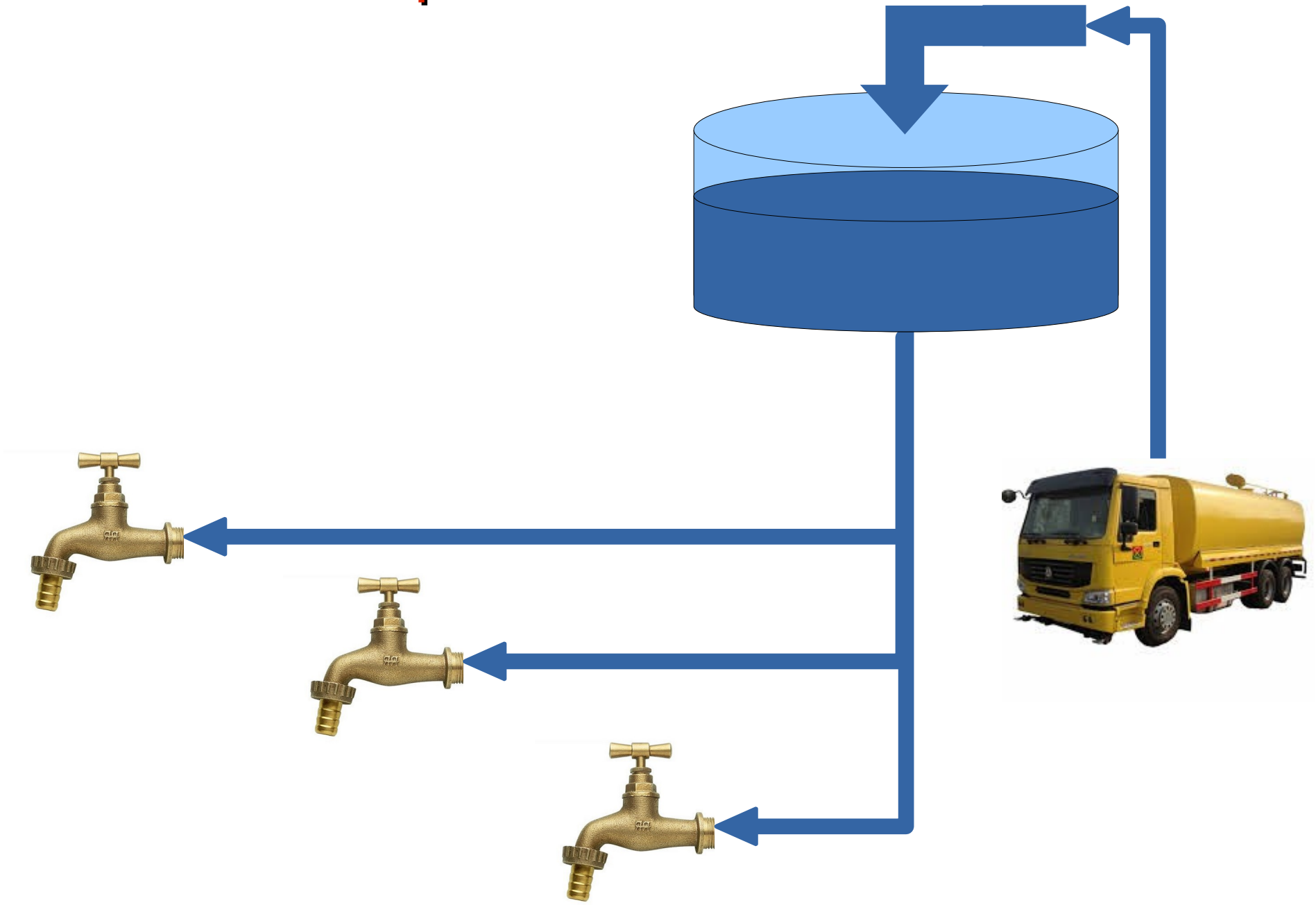
# Pregled predavanja

- Red (queue)
  - osnovne operacije i implementacija
  - cirkularni bafer

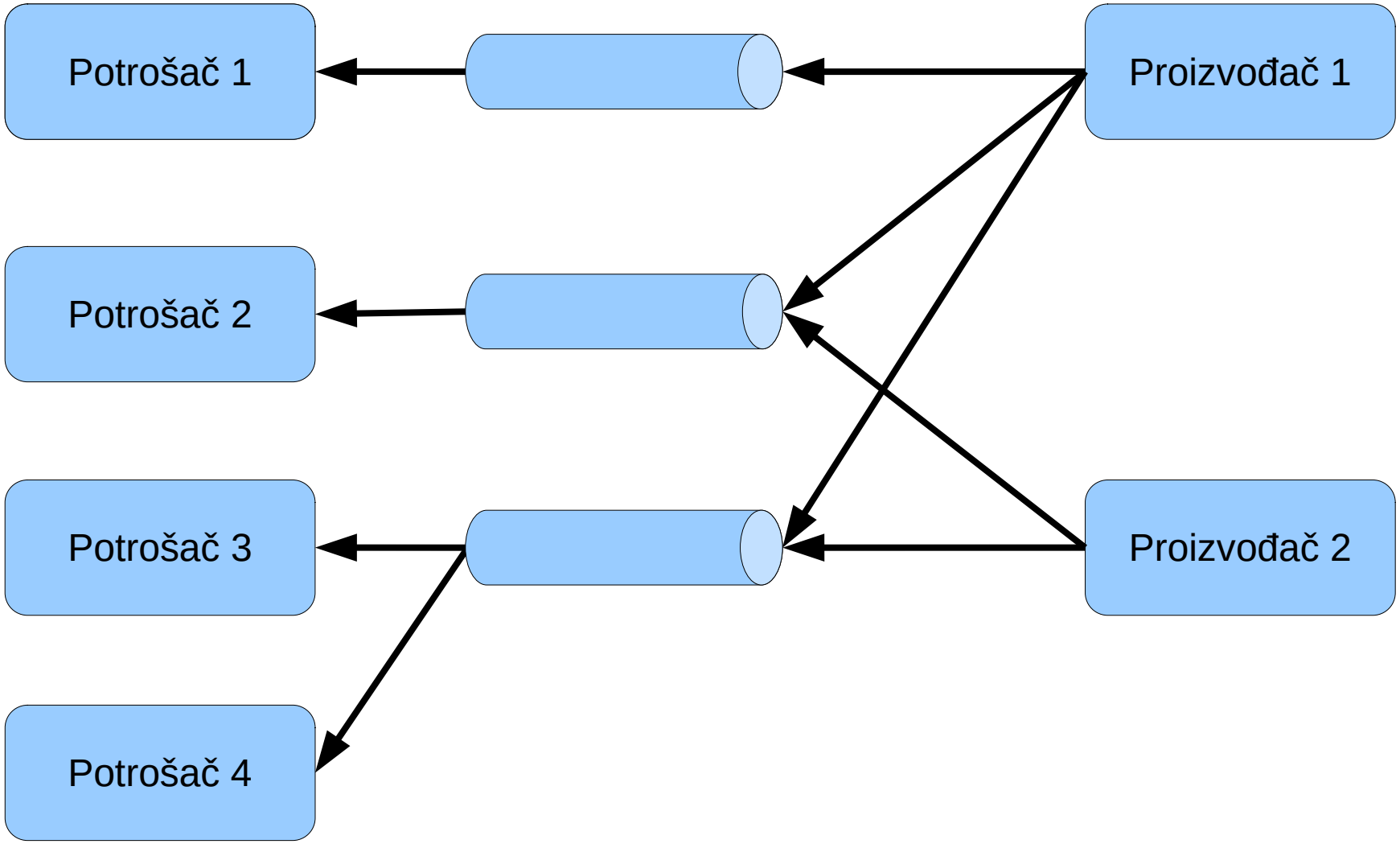
# Red (queue)

- Primjeri iz života za red (queue)


# Proizvođač - potrošač



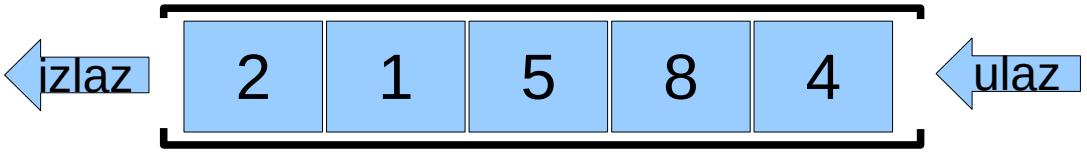
# Proizvođač - potrošač



# ATP: red (queue)

- Ideja je ista kao i u stvarnom životu: kolekcija elemenata u linearnom poretku (linearna struktura podataka), odnosno **lista**.
  - ulaz sa jedne a izlaz sa druge strane
  - moramo voditi računa o kapacitetu
  - sve operacije su $O(1)$

Red: <2, 1, 5, 8, 4>




Red (*Queue*):
FIFO – First In First Out

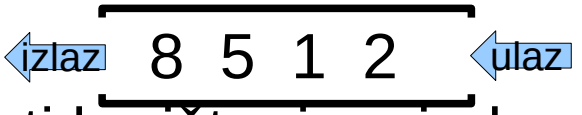
# Red (queue) – osnovne operacije

| Red (queue) | |
|---|---|
| `void enqueue(T x), void push(T x)`<br>`void ubaci(T x)` | dodavanje |
| `void dequeue() ili T dequeue()`<br>`void pop() ili T pop()`<br>`void izbaci() ili T izbaci()` | uklanjanje ili čitanje i uklanjanje |
| `T front()`<br>`T prednji()` | čitanje bez uklanjanja |
| `bool empty()`<br>`bool prazan()` | provjera da li je struktura prazna |

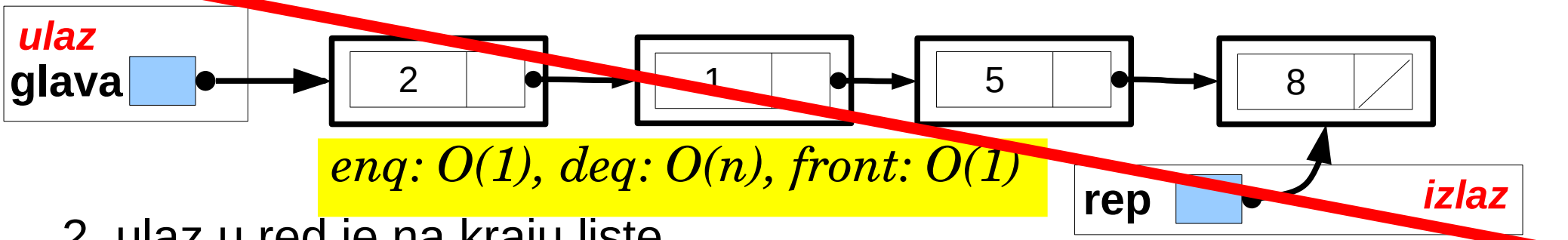
Moramo obratiti pažnju da li je struktura prazna pri operacijama uklanjanja ili čitanja.

- Moguće je definisati i ostale operacije (metode, operatore), prema potrebi.

# Red – implementacija povezanom listom

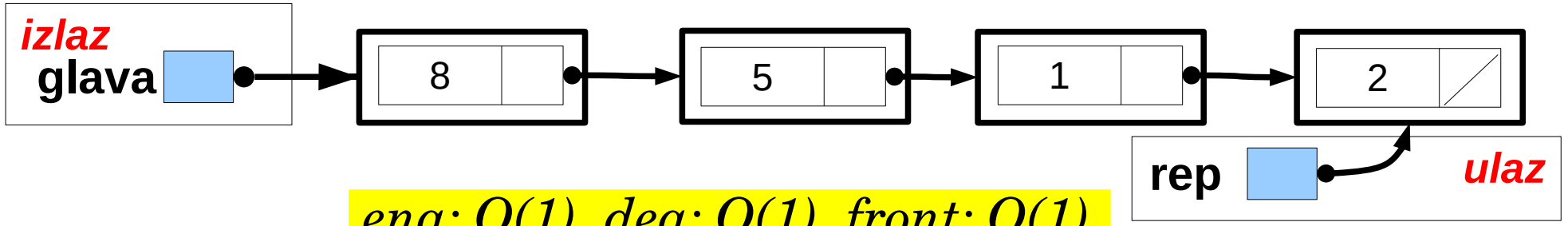



- Red pomoću povezane liste možemo realizovati korištenjem jedne od dvije strategije (koja je bolja?):
- 1. ulaz u red je na početku liste
  - dodajemo na početak i uklanjamo sa kraja liste




*enq: O(1), deq: O(n), front: O(1)*

2. ulaz u red je na kraju liste
  - dodajemo na kraj i uklanjamo sa početka liste




*enq: O(1), deq: O(1), front: O(1)*

# Red – ubacivanje elemenata

- Ubacivanje elementa u red (*enqueue, push*)

```
template<typename T>
template<typename U>
void Red<T>::ubaci(U && x)
{
  Cvor* novi = new Cvor( std:forward<U>(x) );

  if( empty() )
    glava = rep = novi;
  else {
    rep->naredni = novi;
    rep = rep->naredni;
  }
  ++vel_reda;
}
```

$O(1)$

# Red – izbacivanje elemenata – varijanta 1 (void)

- Izbacivanje elementa iz reda (*dequeue, pop*):

```
template<typename T>
void Red<T>::izbaci()
{
  // ako je red prazan uradi nešto…

  Cvor *temp = glava;
  glava = glava->naredni;

  delete temp;
  --vel_reda;
}
```

$O(1)$

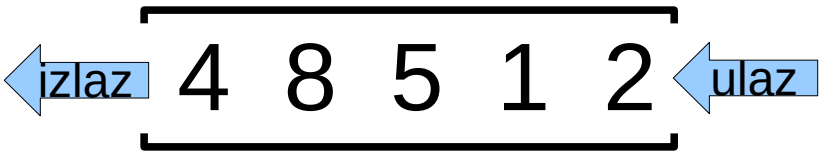
# Red – izbacivanje elemenata – varijanta 2 (povratna vrijednost)

- Izbacivanje elementa iz reda (*dequeue, pop*):

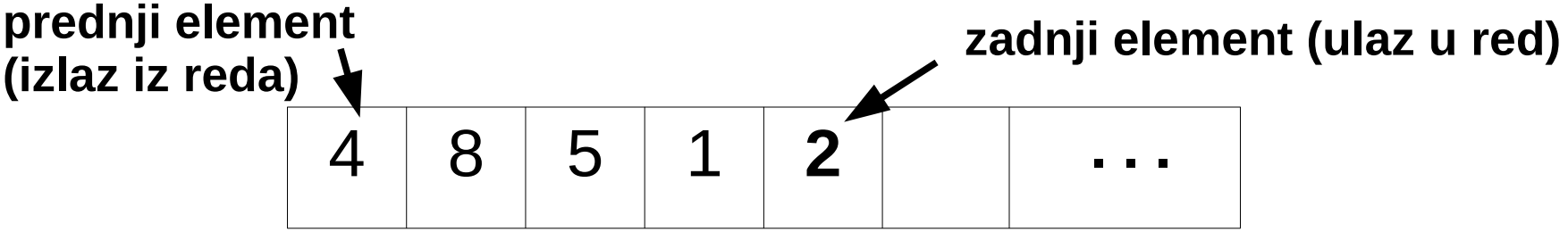
```
template<typename T>
T Red<T>::izbaci()
{
  // ako je red prazan uradi nešto…

  Cvor *temp = glava;
  glava = glava->naredni;
  T elem = std::move(temp->element);
  delete temp;
  --vel_reda;
  return elem;
}
```

$O(1)$

# Red (queue) – implementacija pomoću niza




- Red pomoću niza možemo realizovati tako što ćemo imati dva markera:
  - prednji – indeks prednjeg elementa u redu
  - zadnji – indeks zadnjeg elementa u redu

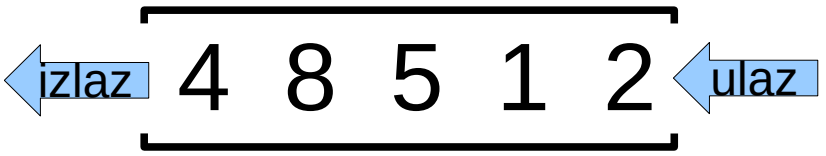



Dodaje se na jedan kraj a uklanja sa drugog – razmatranja kao kod liste implementirane nizom

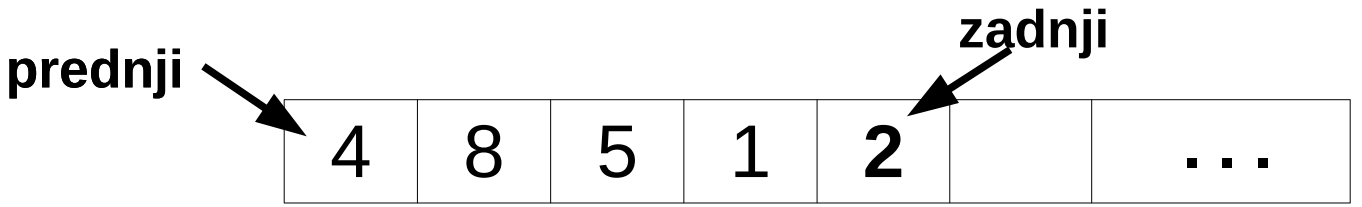
- pri uklanjanju prednjeg moramo sve pomjeriti ulijevo. Isto je ako obrnemo poredak.

$O(n)$

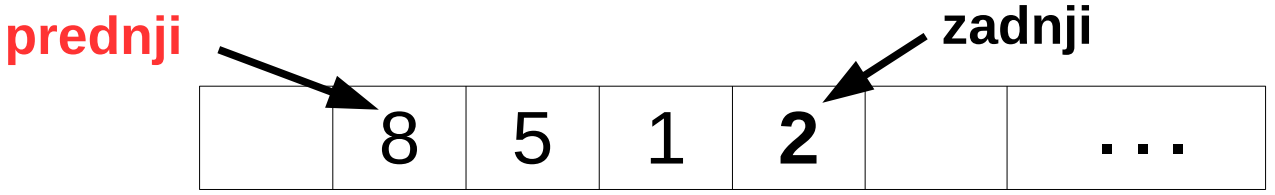
# Red (queue) – implementacija pomoću niza



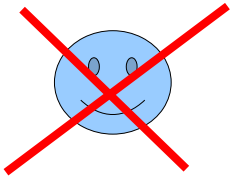

- Bolji pristup je da samo pomjeramo markere za prednji i zadnji element:



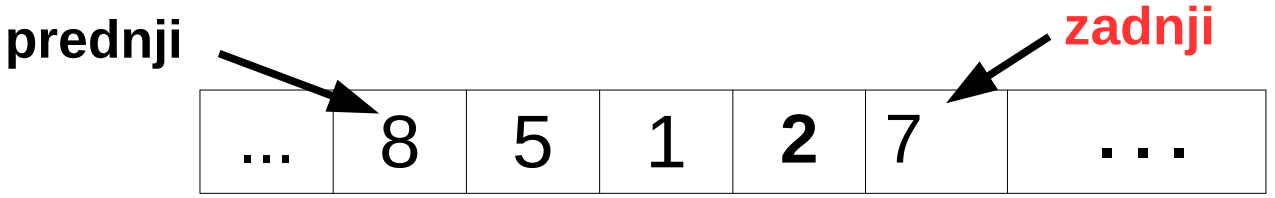

Izbacivanje:




$O(1)$



Ubacivanje:



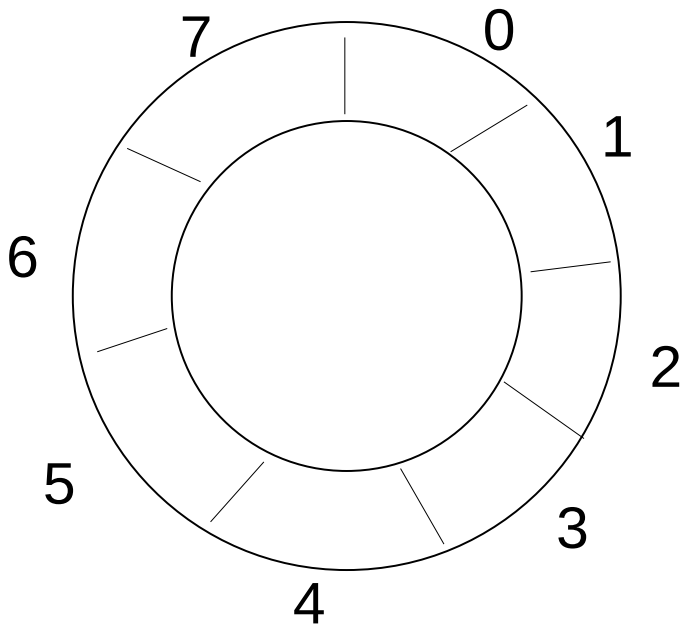

$O(1)$

Problem: šta kad dođemo do kraja niza? Realocirati niz ili sve prebaciti lijevo? Šta god od ovog da uradimo => $O(n)$

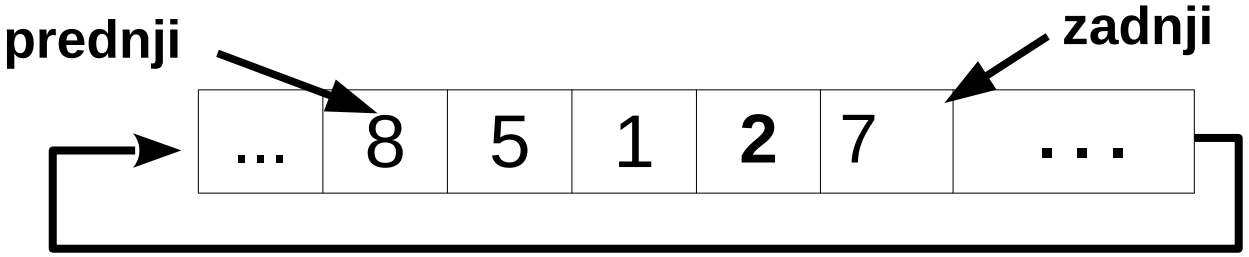
# Cirkularni bafer (circular buffer, ring buffer)

- Cirkularni bafer je niz kod kojeg su spojena dva kraja:

***IDEJA***




***REALIZACIJA***




Šta uraditi kad `zadnji` dođe do kraja niza:

```
++zadnji;
if(zadnji == kapacitet)
   zadnji = 0;
```

ili

```
zadnji = (zadnji+1) % kapacitet;
```

Slično važi i za `prednji`.

# Red – ubacivanje i izbacivanje elemenata

- Ubacivanje elementa u red (*enqueue, push*)

```
template<typename T>
template<typename U>
  Red<T>& Red<T>::ubaci(U && x)
  {
    if( !jePun() ){
      zadnji = (zadnji + 1) % kapacitet;
      elementi[zadnji] = std::forward<U>(x);
      ++br_elemenata;
      return *this;
    } else … realokacija
  }
```

$O(1)$

- Izbacivanje elementa iz reda (*dequeue, pop*):

```
template<typename T>
T Red<T>::izbaci()
{
  if(!jePrazan()){
    int indeks_za_pop = prednji;
    prednji = (prednji + 1) % kapacitet;
    --br_elemenata;
    return std::move(elementi[indeks_za_pop]);
  } else ...
}
```

$O(1)$

# Red – provjera da li je red prazan ili pun

- Prije ubacivanja elementa u red treba provjeriti je li red pun:

```
bool jePun() const
{
  return (br_elemenata == kapacitet);
}
```

$O(1)$

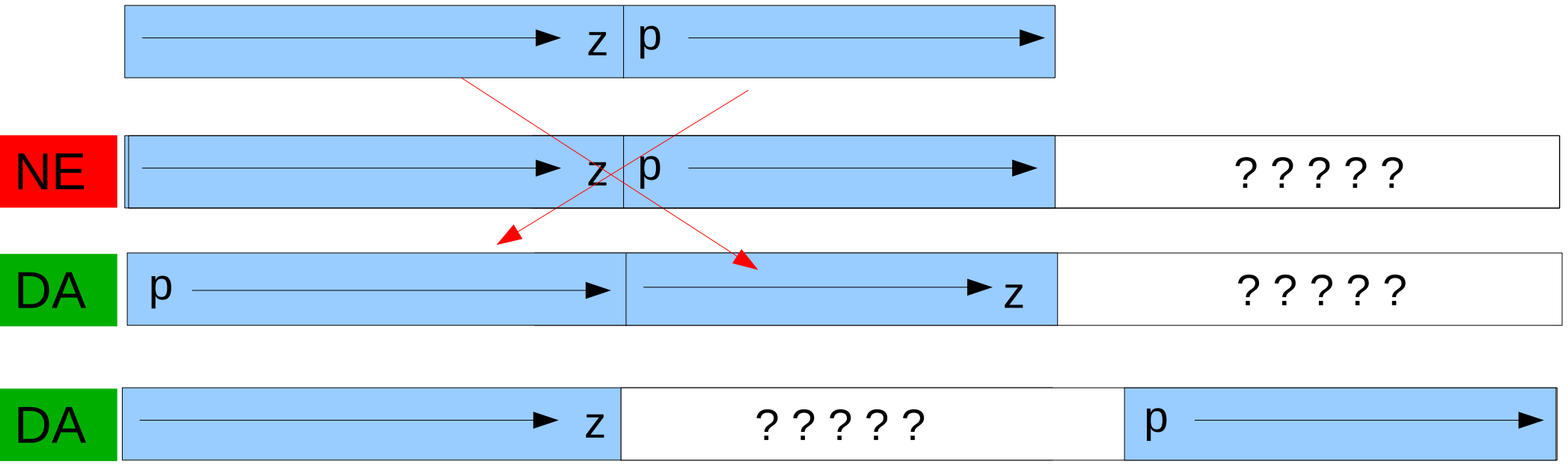
- Prije izbacivanja elementa iz reda treba provjeriti je li red prazan:

```
bool jePrazan() const
{
  return (br_elemenata == 0);
}
```

$O(1)$

# Cirkularni bafer – povećanje kapaciteta

- Povećanje kapaciteta cirkularnog bafera ima svoje specifičnosti u odnosu na povećanje običnog niza.
- Nije dovoljno samo kopirati elemente niza na iste indekse.
- Elementi moraju imati isti poredak **u redu** a ne u nizu.




$O(n)$

# Red (queue) – STL implementacija

- Header `<queue>`, klasa `queue`
- Metodi:
  - `size()` – broj elemenata u redu
  - `empty()` – vraća `true` ako je prazan, `false` ako nije
  - `push(x)` – ubacuje element u red (na kraj)
  - `pop()` – izbacuje element sa početka reda
  - `front()` – vraća referencu na prvi element u redu
  - `back()` – vraća referencu na zadnji element u redu
- Korištenje:

```
#include<queue>
  ...
  std::queue<int> intRed;
```

# Red i stog – usporedba implementacija

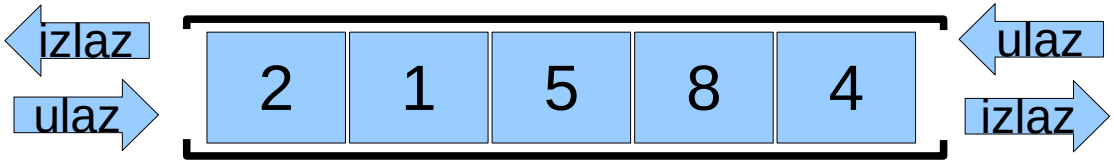
- Sve osnovne operacije imaju složenost O(1).
- Da li ove strukture implementirati pomoću niza ili povezane liste?

- Implementacija memorijom (nizom):
  - brzo dodavanje i uklanjanje
  - može zauzimati mnogo više memorije nego što je potrebno
- Implementacija povezanom listom
  - jedno dodavanje ili uklanjanje zahtijeva **malo** više vremena zbog dinamičke alokacije
  - bolja iskoristivost memorije, mada svaki čvor zauzima više memorije nego kod niza
- Ako je količina memorije kritičan faktor koristiti implementaciju nizom sa realokacijom pri svakoj promjeni

# ATP: dvostrani red (double ended queue)

- Red kod kojeg se elementi mogu ubacivati ili uklanjati sa oba kraja.
- Sve operacije su $O(1)$

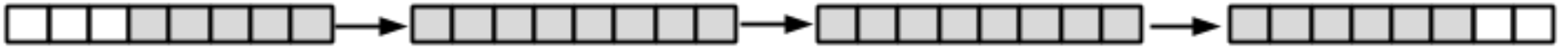
d.e. queue: <2, 1, 5, 8, 4>



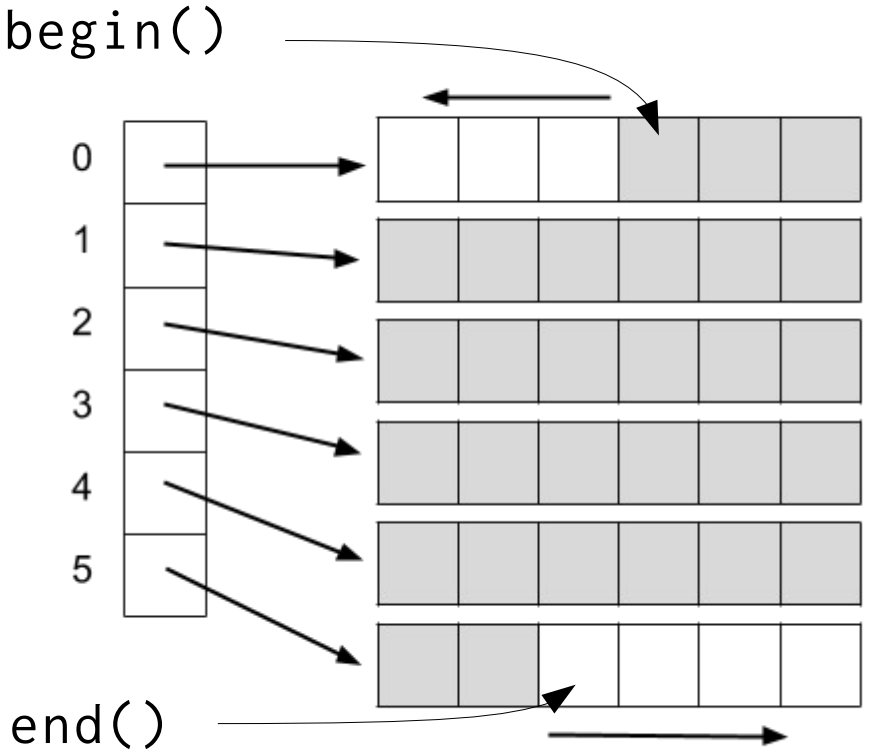
# Dvostrani red (deque) – STL implementacija

- Objedinjuje dobre osobine dvostruko povezane liste i vektora.
- Header `<deque>`, klasa `deque`
- Neki metodi:
  - `size` – broj elemenata u redu
  - `empty` – vraća `true` ako je prazan, `false` ako nije
  - `push_back(x) - ubacuje element na kraj`
  - `push_front(x) - ubacuje element na početak`
  - `pop_back - izbacuje element sa kraja`
  - `pop_front - izbacuje element sa početka`
  - `front` – vraća referencu na prvi element
  - `back` – vraća referencu na zadnji element
  - `operator[]` – pristup elementu na željenoj poziciji

# STL deque – logički prikaz interne organizacije



chunks...



# STL deque – uproštena interna organizacija