Lista prosta – różne typy

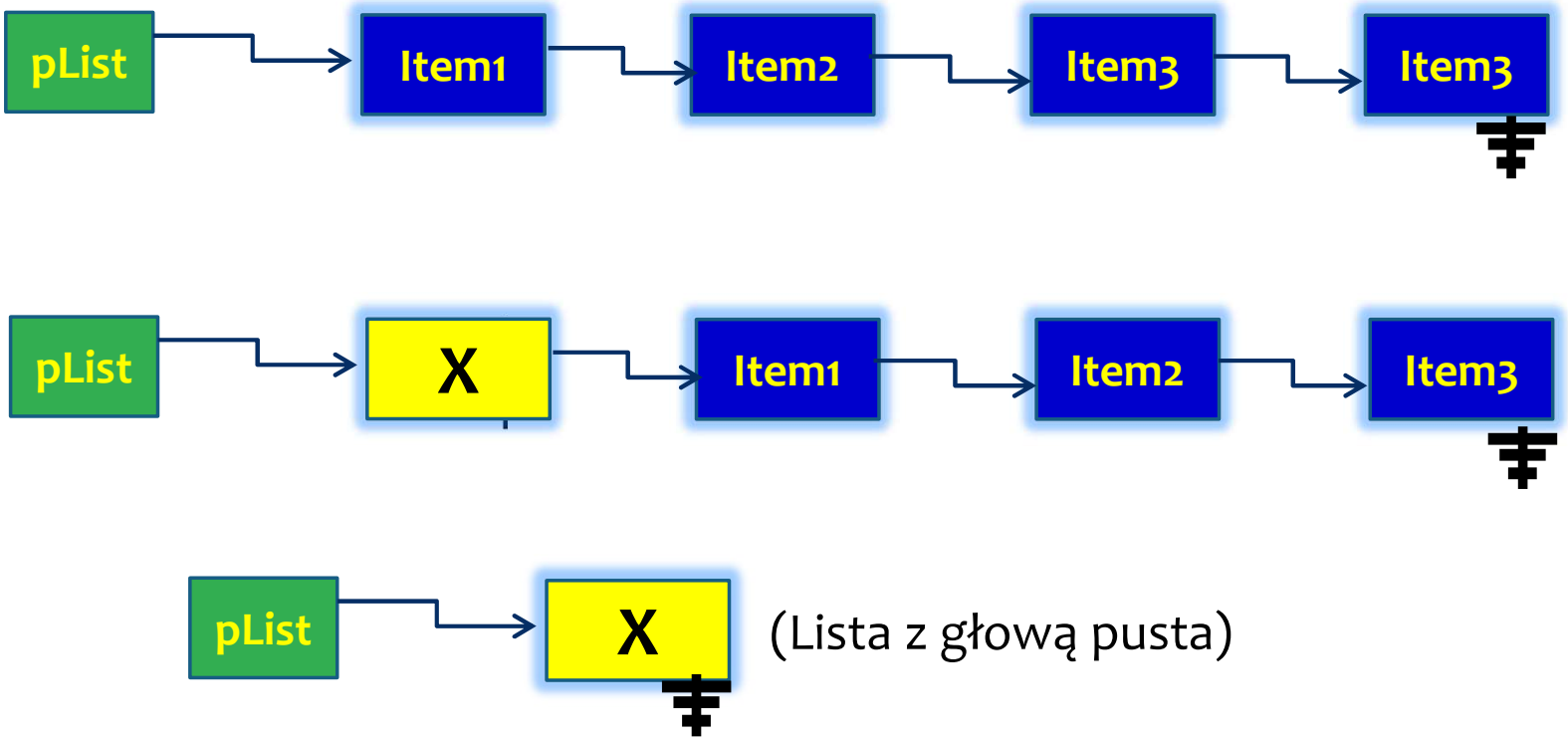
Listy Proste

* **Możliwe listy proste:**
  1. Lista, do której nowy element jest dodawany jako pierwszy i zdejmowany jest pierwszy z listy – jest to rodzaj kolejki LIFO (*Last IN First Out*) – stos
  2. Lista, do której dodawany element jest na końcu a zdejmowany jest z początku – jest to rodzaj kolejki FIFO (*First In First Out*). Dobrze robić jako strukturę dwuwskaźnikową (wskaźnik na pierwszy element i na ostatni element)
  3. Lista, do której dodawany może być element w dowolnym miejscu i usuwany z dowolnego miejsca – najlepiej gdy istnieje ZAWSZE wskaźnik do wcześniejszego elementu listy
  4. Lista cykliczna

LIFO – lista prosta (1)

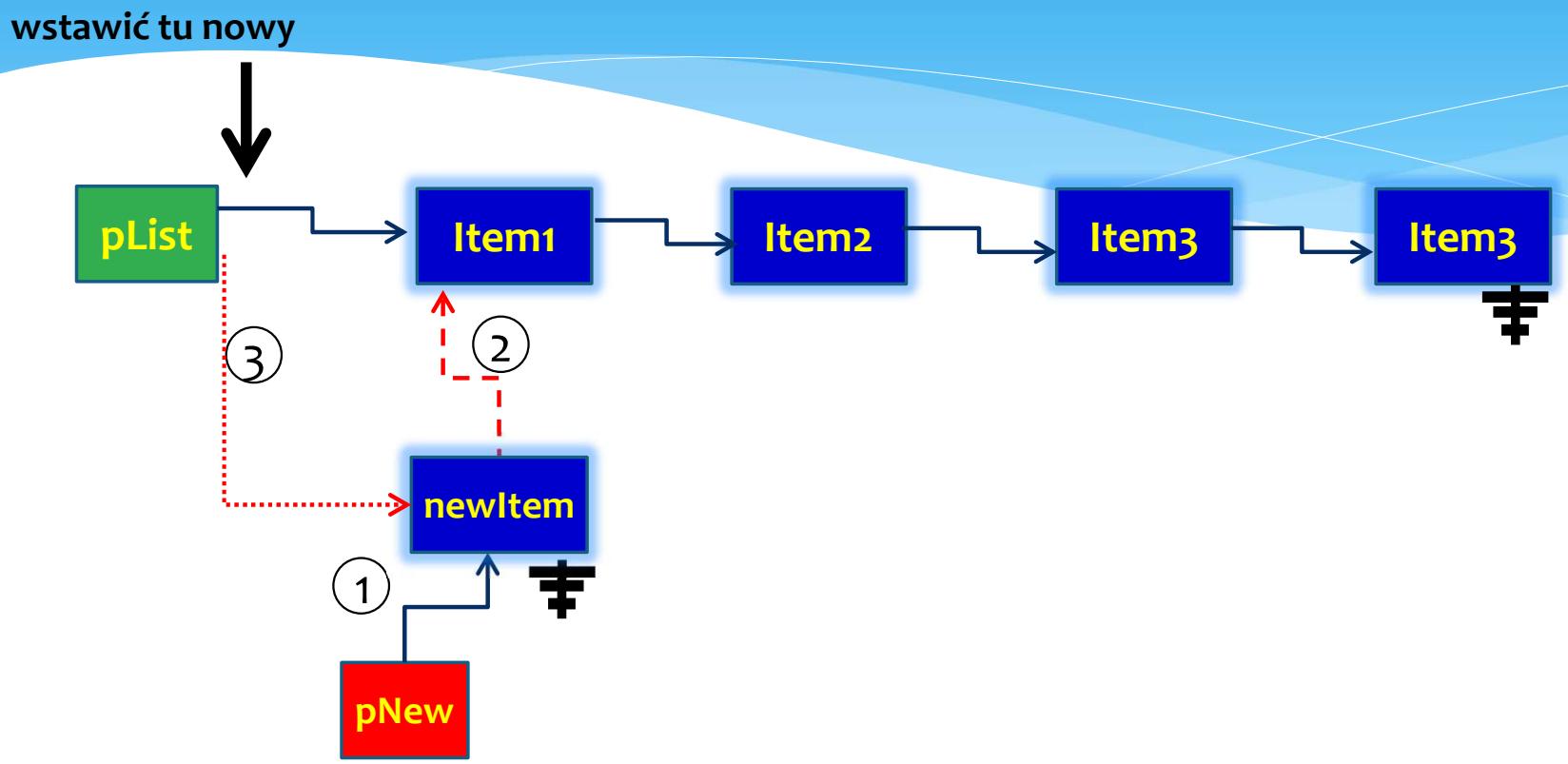
```
typedef struct tagListItem
{
  // jakaś informacja – dowolne pola struktry
  tagListItem*  pNext;
  // struct tagListItem*  pNext;
} ListItem;

// deklaracja listy
ListItem* pList = createList();
   createList() zwraca -  // nullptr (C++)
   Jeśli lista z głową to zwraca wsk na ListItem
```

LIFO – lista prosta (1)



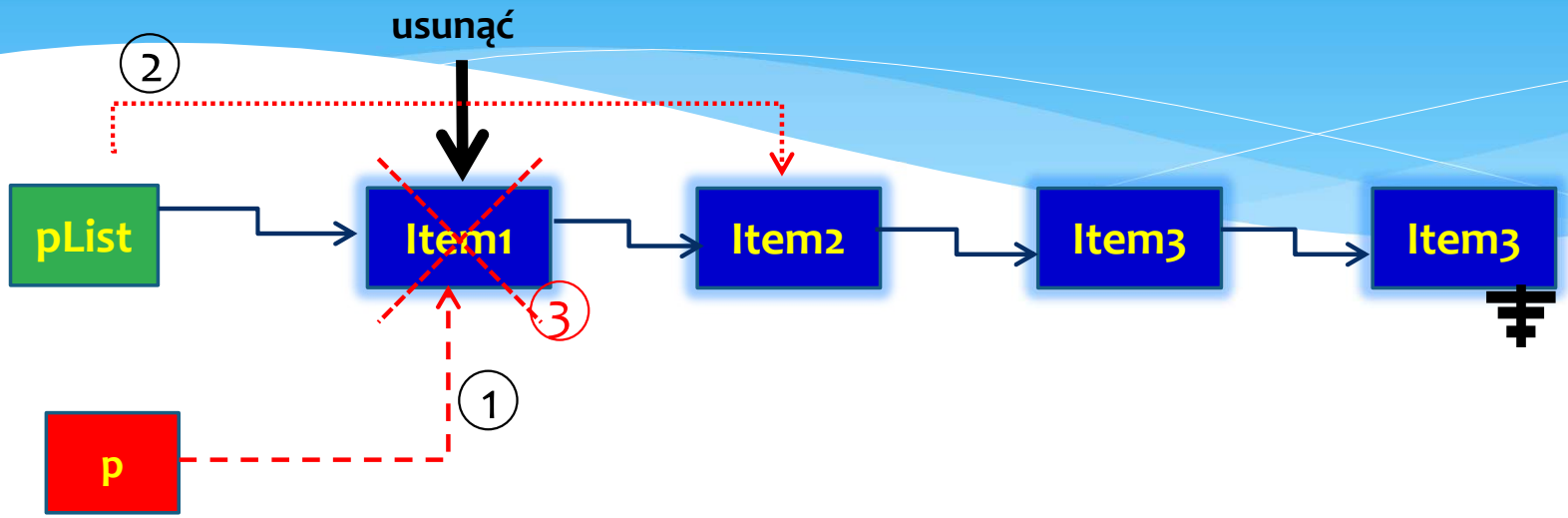
(Lista z głową pusta)

Dodanie elementu (przypadek 1)



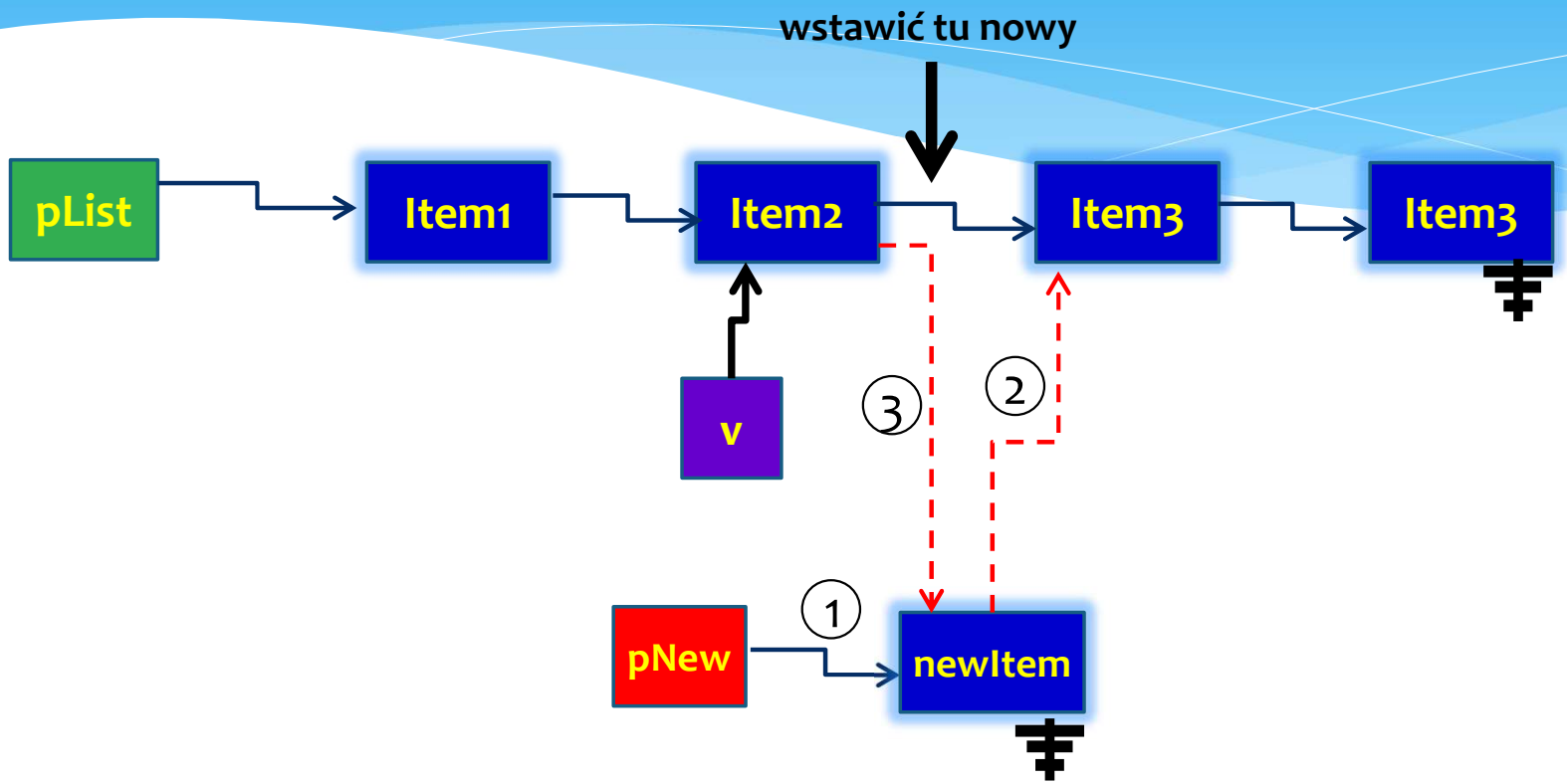

```
ListItem* pNew = //alokacja pamięci na nowy elemen listy (1)
// wstawić info do nowego elementu (ewentualnie wyzerować wcześniej)
// dowiązać do listy (2)
pNew->pNext = pList;   //  (*pNew).pNext;
// przewiązać wskaźnik listy na nowy element (3)
pList = pNew;
```

Usuwanie elementu (przypadek 1)



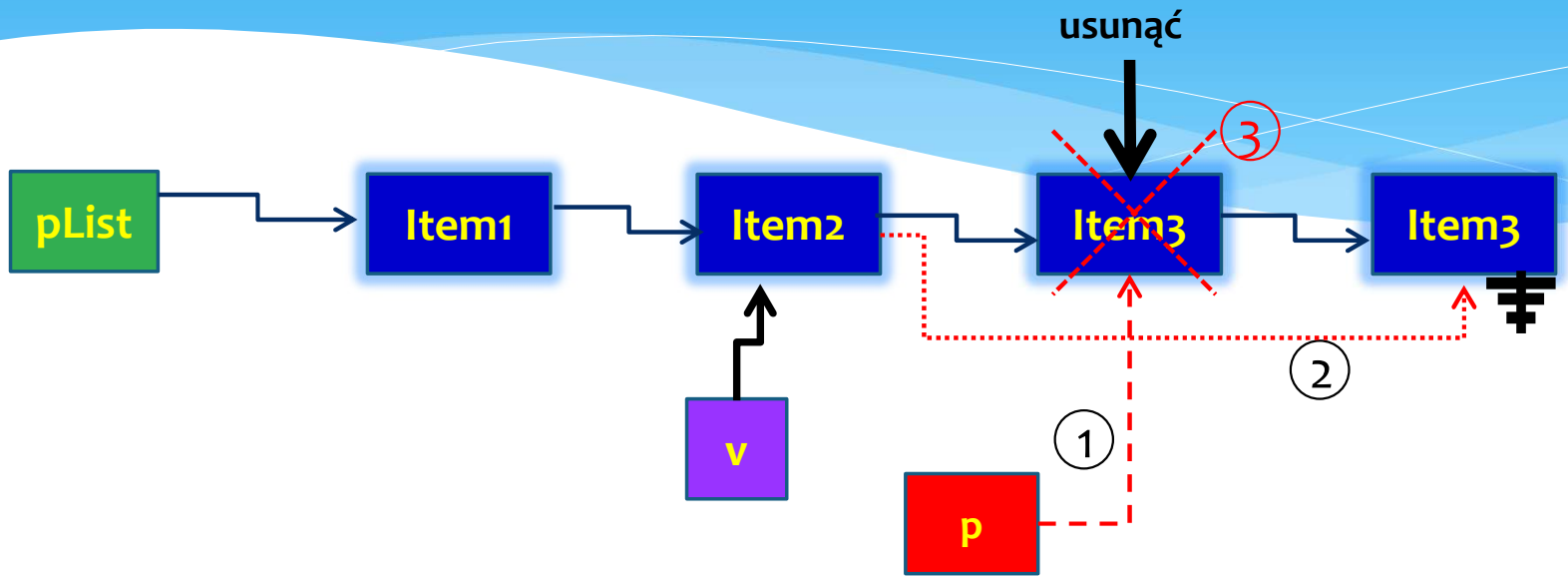

```
// "złapać" pierwszy element listy dodatkowym wskaźnikiem
ListItem* p = pList;  // (1)
// przewiązać wskaźnik listy na kolejny element
pList = p->pNext;  //(2)      pList = pList->pNext;
// zwolnić pamięć
free( p );  // (3)
```

Dodanie elementu (przypadek 3)



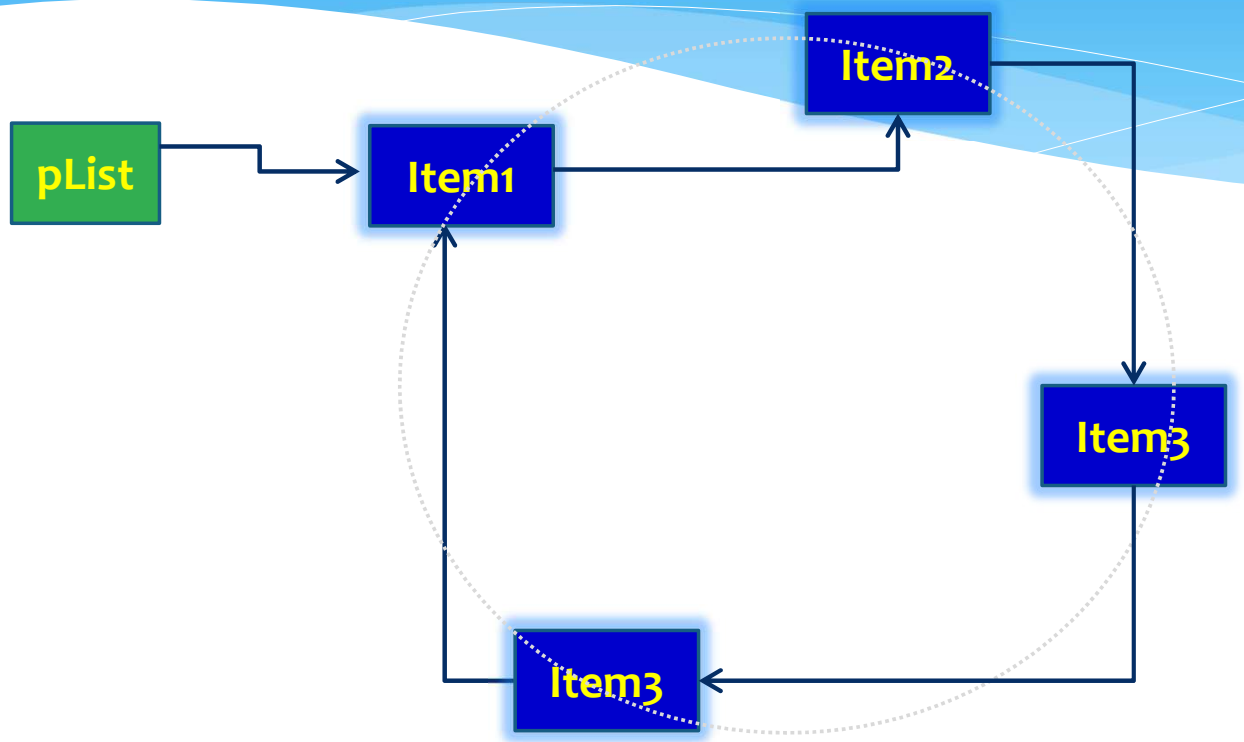

```
// wyszukać, po którym elemencie wstawić – ustawienie wsk. v
// problem gdy lista nie ma pustego elementu jako pierwszego (wartownik)
ListItem* pNew = //alokacja nowego   (1)
// wypełnienie informacji  np. pNew->nKey = x;
pNew->pNext = v->pNext;  // (2)
v->pNext = pNew;  // (3)
```

Usuwanie elementu (przypadek 3)




```
// wyszukać, do usunięcia - można użyć tylko v lub od razu v i p
ListItem* p = v->Next;  // (1)
v->next = p->pNext;  //(2)
free( p );  // (3)
```

Lista Cykliczna (przypadek 4)

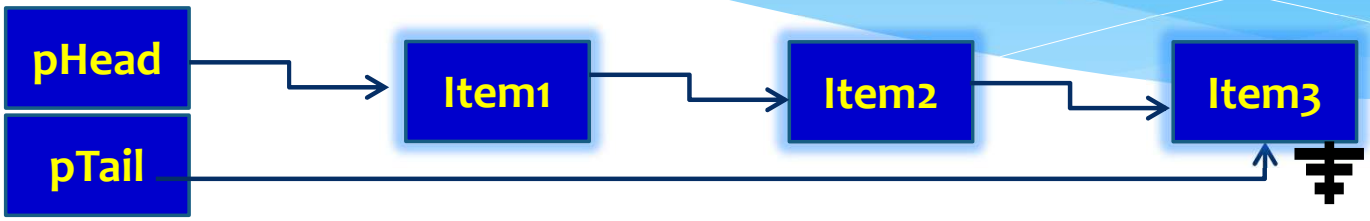


Lista może mieć pusty element jako pierwszy (lista z głową)

Wstawianie podobnie jak w przypadku listy prostej (wyjątek: lista pusta)

FIFO- wstawianie

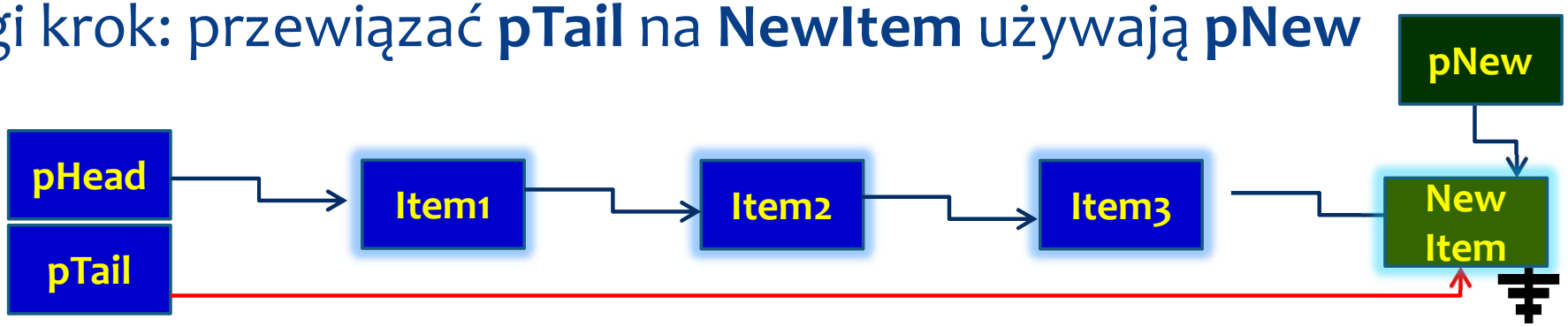
2. FIFO – ze strukturą dwuwskaźnikową



Wstawianie do kolejki:

* Pierwszy krok: przywiązać na końcu używając **pTail**

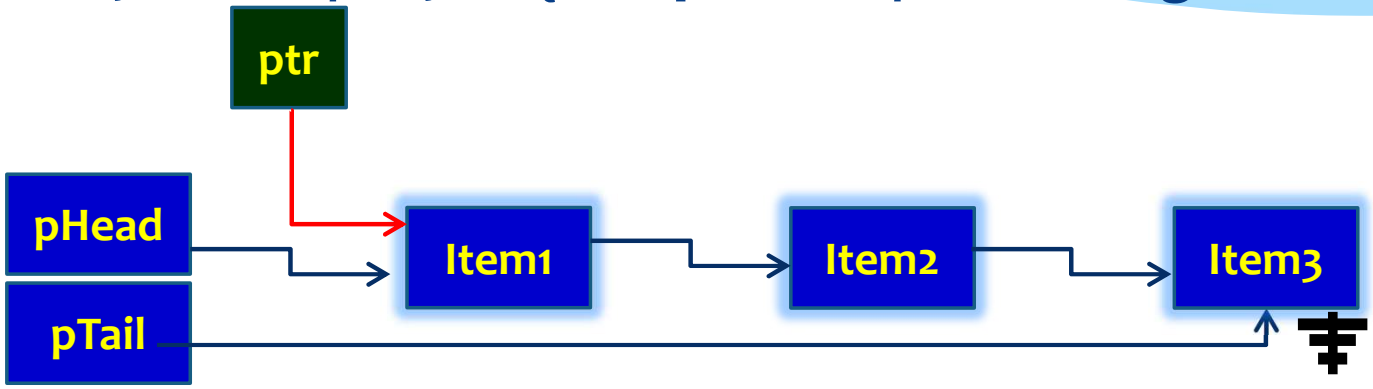
* Drugi krok: przewiązać **pTail** na **NewItem** używają **pNew**



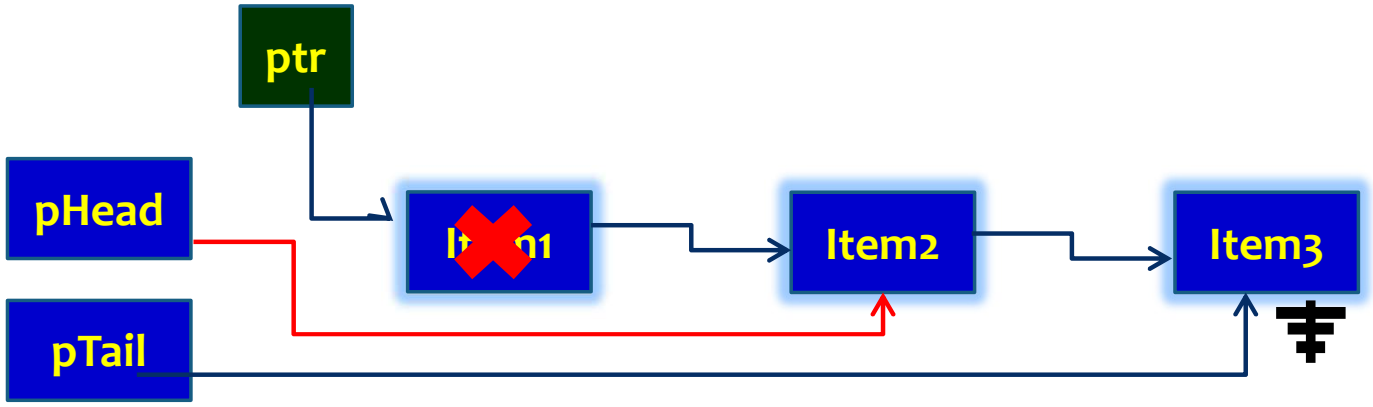
FIFO Usuwanie

Usuwanie z kolejki:

* Pierwszy krok: przywiązać **ptr** do pierwszego

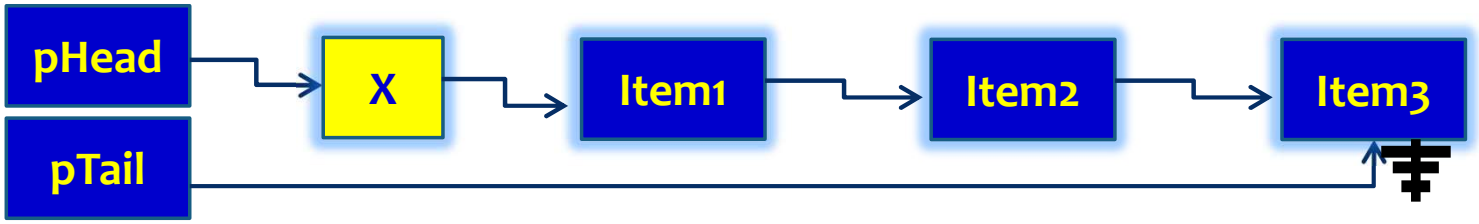


* Drugi krok: przewiązać **pHead** na następny i skasowanie



Listy Proste

* Każda lista może mieć tak zwaną głowę – pusty element (czasami nazywany wartownikiem). Wtedy pusta lista ma zawsze jeden pusty (nieistotny element). Ułatwia to wstawianie do kolejki i usuwanie z kolejki (bez dodatkowych warunków)

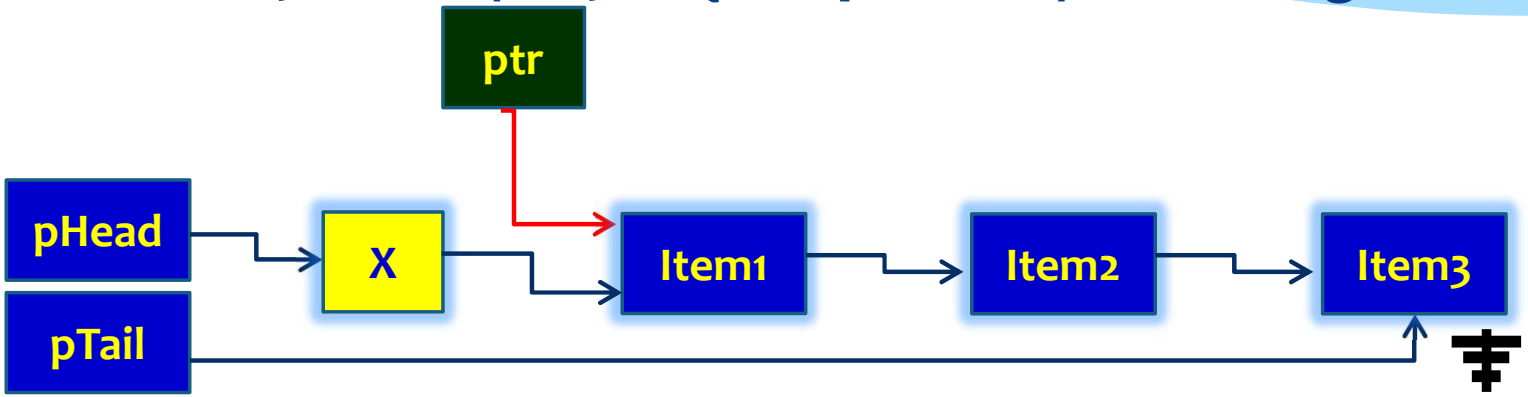


* W przypadku FIFO z głową wstawianie takie samo.

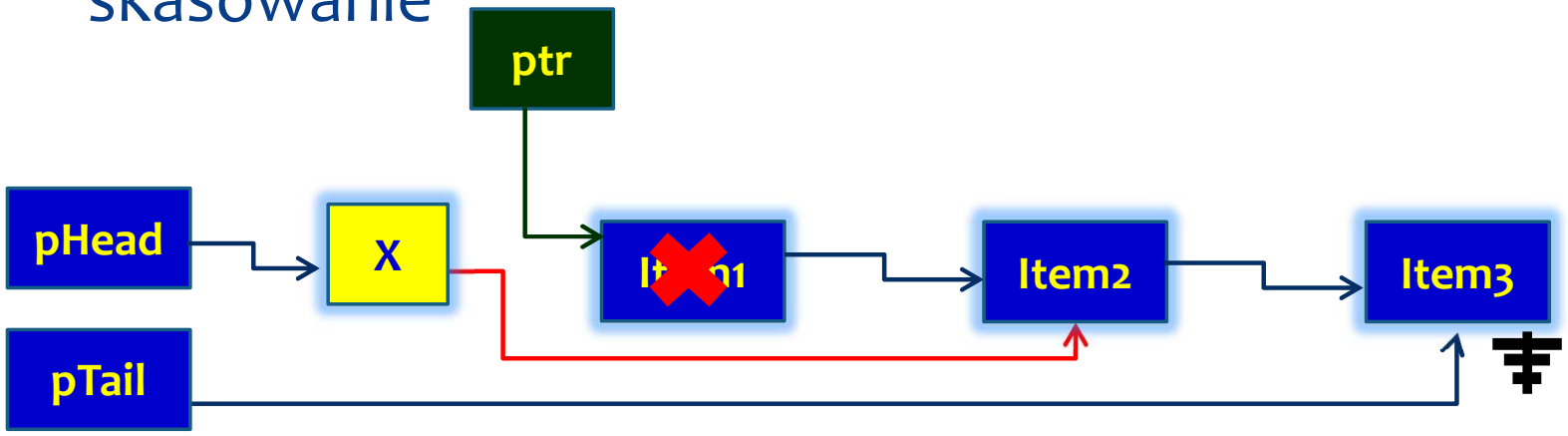
FIFO – z głową - Usuwanie

Usuwanie z kolejki:

* Pierwszy krok: przywiązać **ptr** do pierwszego



* Drugi krok: przewiązać **pHead->pNext** na następny i skasowanie



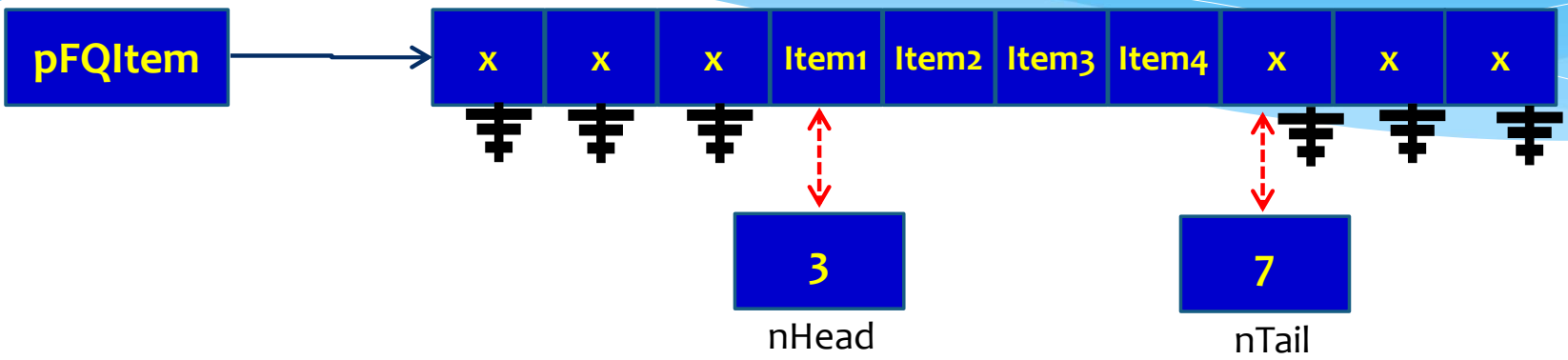
FIFO – lista prosta

```
typedef struct tagQFIFOItem
{
  // jakaś informacja
  tagQFIFOItem*  pNext;
} QFIFOItem;

typedef struct tagQFIFO
{
   QFIFOItem*  pHead;
   QFIFOItem*  pTail;
} QFIFO;
```

Po wykreowaniu dynamicznym struktury FIFO (kreowanie FIFO) należy ją wyzerować.

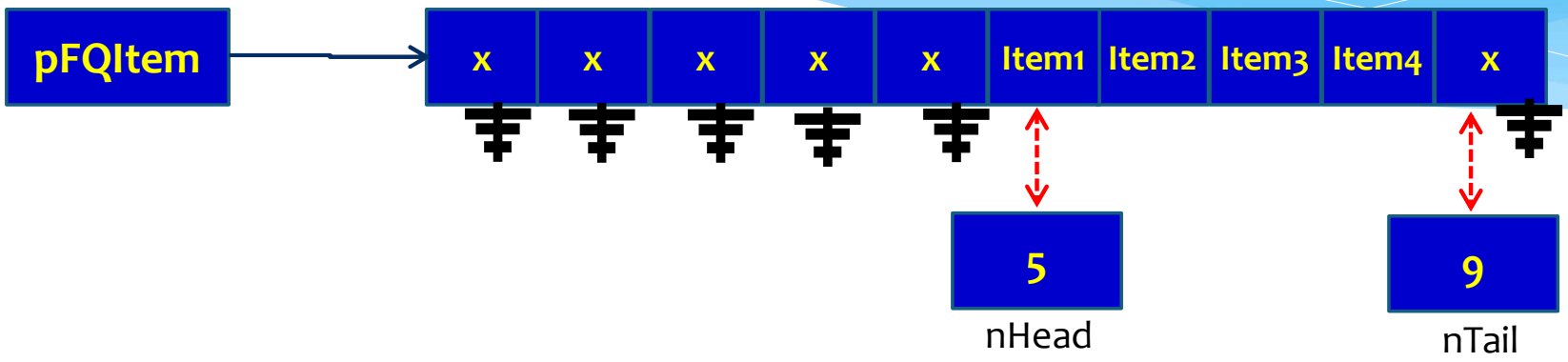
Jeśli FIFO z głową to kreowanie kolejki tworzy dodatkowo element typu `FIFOItem` (wyzerowany) oraz `pHead` i `pTail` wskazują na ten element) głowa

FIFO - tablicowe

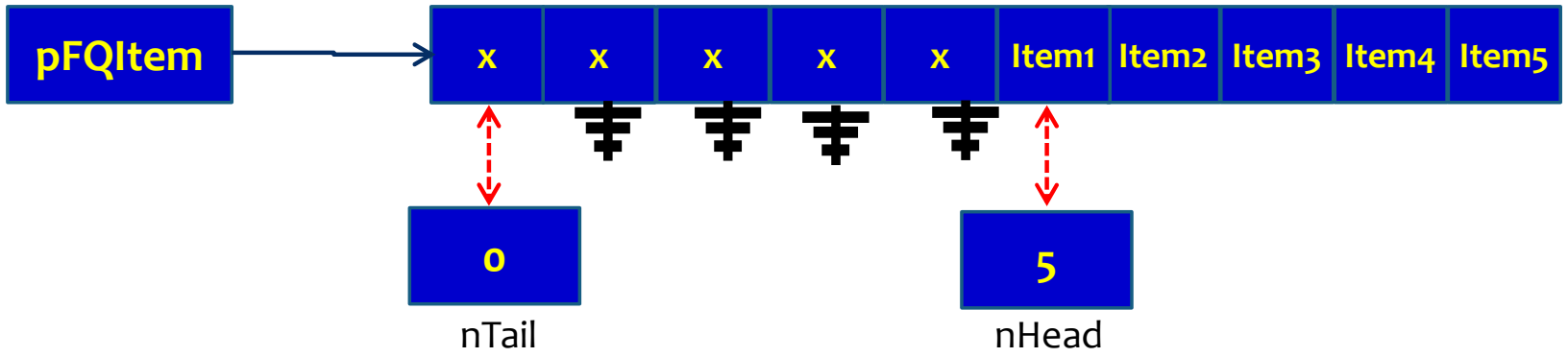


* Kolejka FIFO reprezentowana jest przez:
  * tablice wskaźników do elementów,
  * indeks pierwszego elementu w kolejce (cyklicznie zmieniany)
  * indeks pierwszego wolnego miejsca w kolejce (cyklicznie zmieniany)
  * ilość elementów w kolejce (musi być bo nHead może być większe od nTail), kolejka pusta jeśli ilość równa 0 i pełna jeśli ilość równa rozmiarowi tablicy.

Wstawienie



* Wstawianie w skrajnym przypadku (podobnie usuwanie)



FIFO - tablicowe

```
typedef struct
{
  // dynamic table of FQITEM poiters        - pFQItems // cyclic queue
  // index of the first item                            - nHead
  // index of the first empty item (one after last) - nTail
  // number of elements in the queue                    - nNoElem
  // queue size (the size of the dynamic table)     - nMaxElem
} FQueue;
```