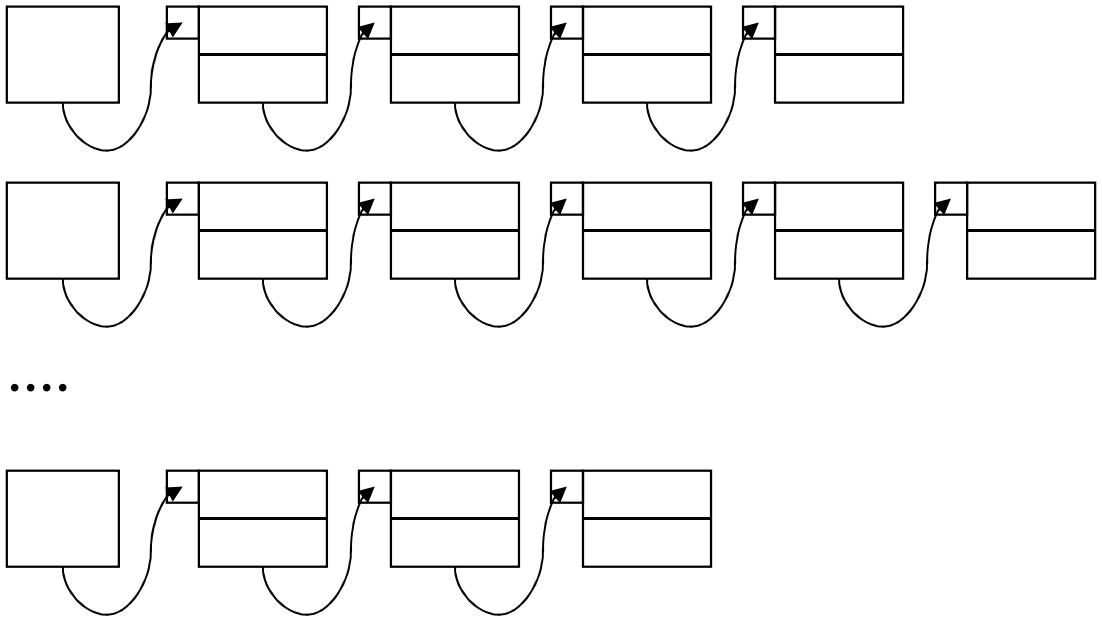
Tablice z haszowaniem

- W przypadku PAO macierz kubełków jest indeksowana i zawiera nagłówki list powiązanych.
- Funkcja mieszająca (haszująca) traktuje klucz wyszukiwania jako argument i wylicza dla niego wartość (np. liczbę z zakresu 0... k-1, gdzie k jest liczbą kubełków)



Tablice z haszowaniem

- W modelu dla pamięci pomocniczej tablice haszujące mają nieco inną postać:
  - Macierz kubełków składa się z bloków, a nie ze wskaźników do nagłówków list
  - Dane dla odpowiedniego kubełka są umieszczane w bloku aż do zapełnienia, a następnie tworzony jest łańcuch bloków nadmiarowych.

Przykład:

- Niech wartości klucza będą literami (w początkowym stanie wartości A ... F)
- Funkcja haszująca h zwraca tylko cztery wartości (czyli $k = 4$):
  - h(D)=0, h(C)=h(E)=1, h(B)=2, h(A)=h(F)=3.

0	D

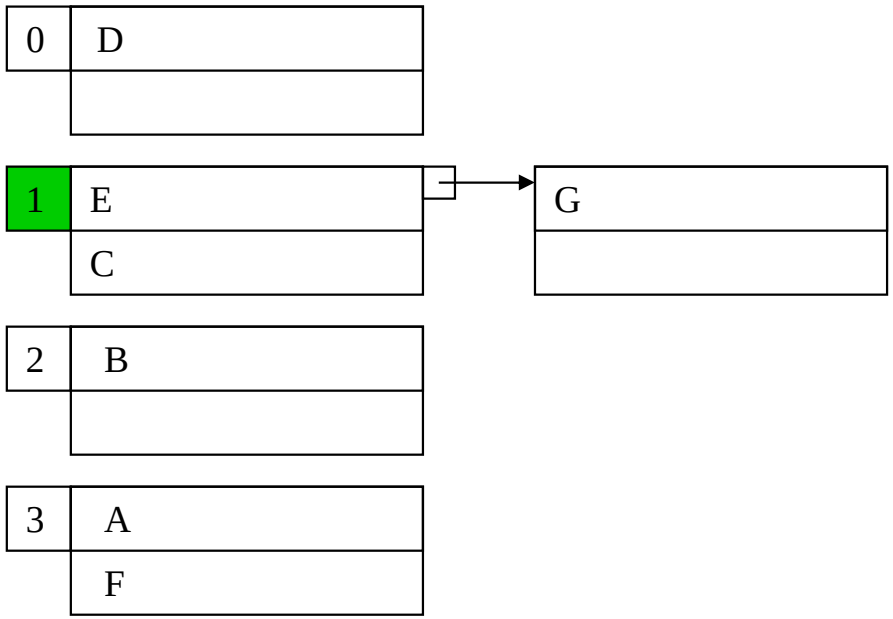
2	B

1	E
	C

3	A
	F

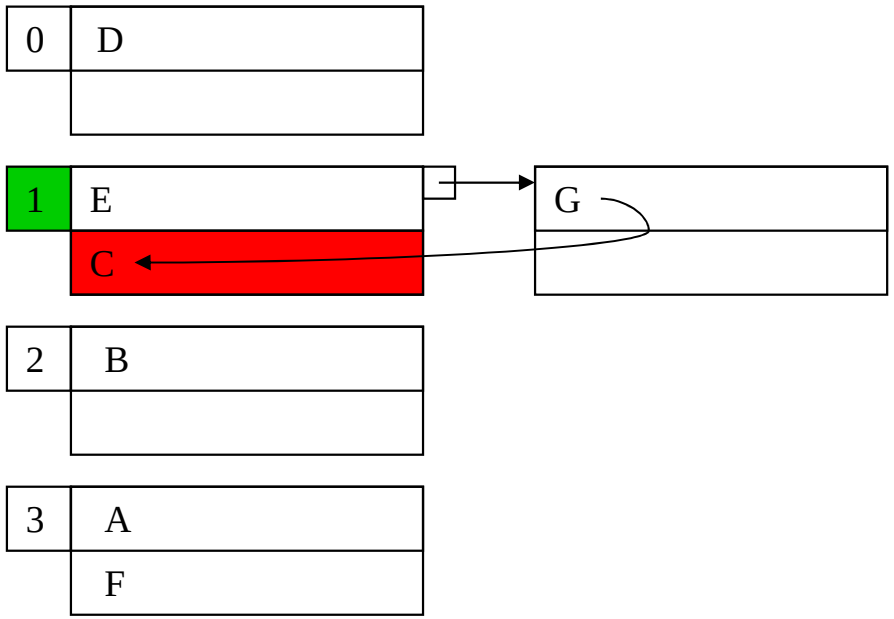
Operacje na tablicach z haszowaniem

- Wstawiamy nowy rekord (z wartością G). Jeśli wartością funkcji haszującej h(G) jest 1, to nowy rekord dołączamy do kubełka 1.
- Ponieważ w bloku tego kubełka nie ma już miejsca, tworzymy blok nadmiarowy.



Operacje na tablicach z haszowaniem

- Podczas usuwania (np. rekordu z wartością C) ustalamy kubłek (wg funkcji haszującej h(C)=1 więc ponownie ten sam kubełek).
- Przeszukujemy listę bloków związanych z kubełkiem 1.
- Powstaje pytanie, czy konsolidować miejsce (czyli czy warto przesuwać rekordy między blokami?).
  - Jeśli będziemy często trafiać w ten sam kubełek, to dodając i usuwając naprzemiennie trzeba będzie usuwać i tworzyć bloki nadmiarowe



Wybór funkcji haszującej

- Co powinno charakteryzować funkcję haszującą?
  - Powinna być łatwa do obliczenia (bo będzie obliczana często)
  - Powinna tak „mieszać”, żeby rekordy rozkładały się w miarę równomiernie, czyli do jednego kubełka przypisywane były zbliżone ilości rekordów.

- Dla liczb całkowitych może to być na przykład reszta z dzielenia przez k.
- Dla ciągu znaków też można wykorzystać resztę z dzielenia, ale np. z sumy wartości im przypisanych (kody ASCII ?)

- Optymalnie byłoby, gdyby kubełków było tyle, by każdy miał tylko jeden blok (bez dowiązaywania). Wtedy typowe wyszukiwanie miałoby tylko jeden dostęp do dysku. Byłoby to lepsze od indeksów rzadkich, gęstych a nawet B-drzew
- Naturalne jednak jest, że gdy plik się zwiększa i przybywa rekordów, to prędzej czy później powstają bloki dowiązane. Warto więc zadbać przynajmniej o to, by było ich niewiele.

Haszowanie dynamiczne

- Do tej pory przykłady dotyczyły **haszowania statycznego**, w którym nie zmieniała się ilość kubełków k.
- Pierwszą z technik **haszowania dynamicznego** jest **haszowanie rozszerzalne:**
  - Wprowadza się poziom pośredni. Kubełki reprezentuje się jako matrycę wskaźników do bloków, a nie matrycę złożoną z bloków.
  - Taka matryca wskaźników może się zwiększać. Długość jest zawsze potęgą 2 (ilość kubełków zawsze rośnie dwukrotnie)
  - Dla każdego klucza K funkcja haszująca h generuje ciąg n bitów (przy jakimś dużym n – powiedzmy n=32) Numery kubełków są jednak mniejsze i do ich ustalenia wykorzystywana jest tylko część bitów.
  - Macierz kubełków ma więc 2^i pozycji, gdzie i jest właśnie ***aktualnie używaną do numeracji kubełków ilością bitów.***

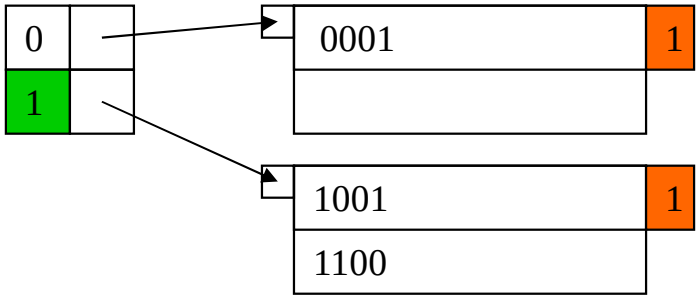
Przykład:

- Funkcja haszująca tworzy w wyniku obliczenia wartości dla klucza K ciąg czterech bitów (n=4)
- Na tym etapie używany jest tylko jeden z tych bitów, czyli i=1
- w macierzy kubełków mamy zatem tylko dwie pozycje $2^i = 2^1$

Haszowanie rozszerzalne

- Wartości w macierzy kubełków wskazują na dwa bloki:
  - pierwszy z nich zawiera wartości z takimi kluczami, dla których funkcja haszująca wyliczyła ciąg bitów **zaczynający się na 0.**
  - Drugi z nich zawiera wartości z takimi kluczami, dla których funkcja haszująca wyliczyła ciąg bitów **zaczynający się na 1.**

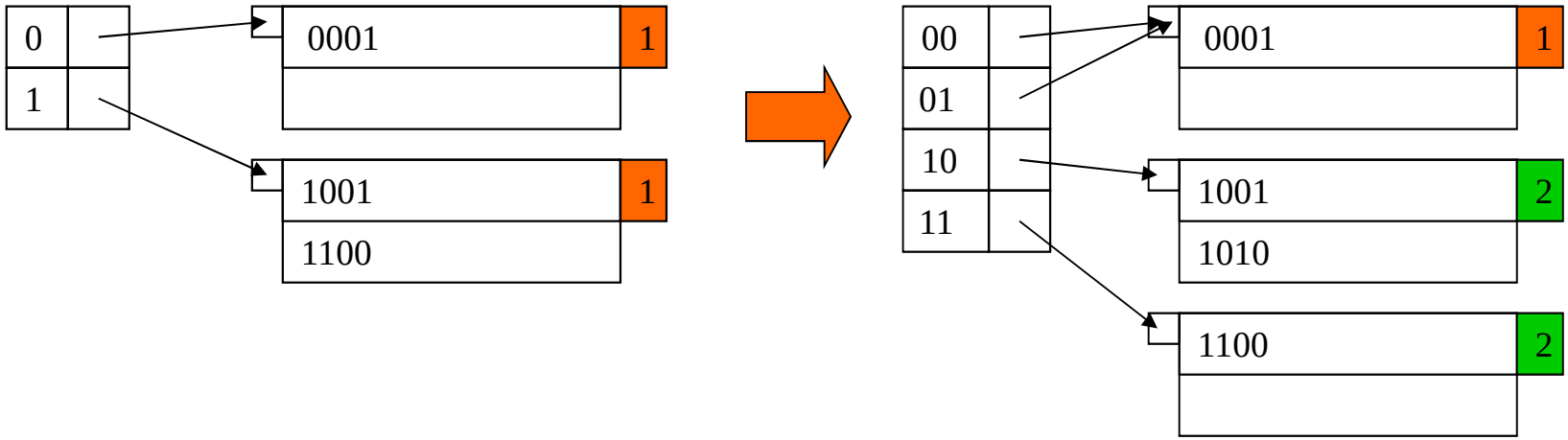
- W bloku zapisana jest informacja o aktualnie używanej ilości bitów służących do reprezentowania kubełka.



Haszowanie rozszerzalne

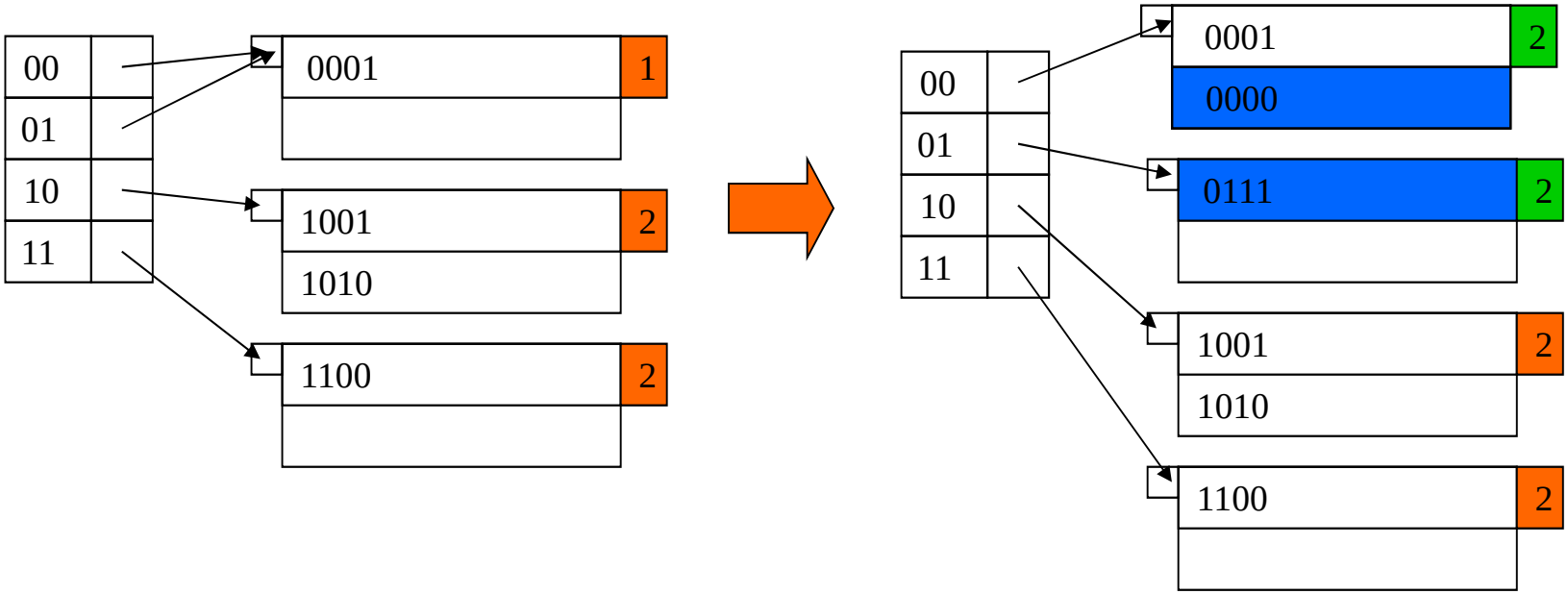
- Co stanie się przy wstawianiu kolejnego rekordu?
  - Jeśli wstawiany rekord po wyliczeniu funkcji haszującej da wartość z jedynką na początku, to nie zmieści się do bloku, bo istnieją tam już obecnie dwa rekordy.
  - W haszowaniu statycznym konieczne byłoby dowiązanie nowego bloku, a w rozszerzalnym zmieni się postać macierzy kubełków.

- Jeśli wstawiany rekord po wyliczeniu funkcji haszującej dał wartość np. 1010 i nie mieści się w bloku, zwiększamy ilość bitów reprezentujących numer kubełka z i=1 na i=2
- z dwóch kubełków tworzy się cztery:



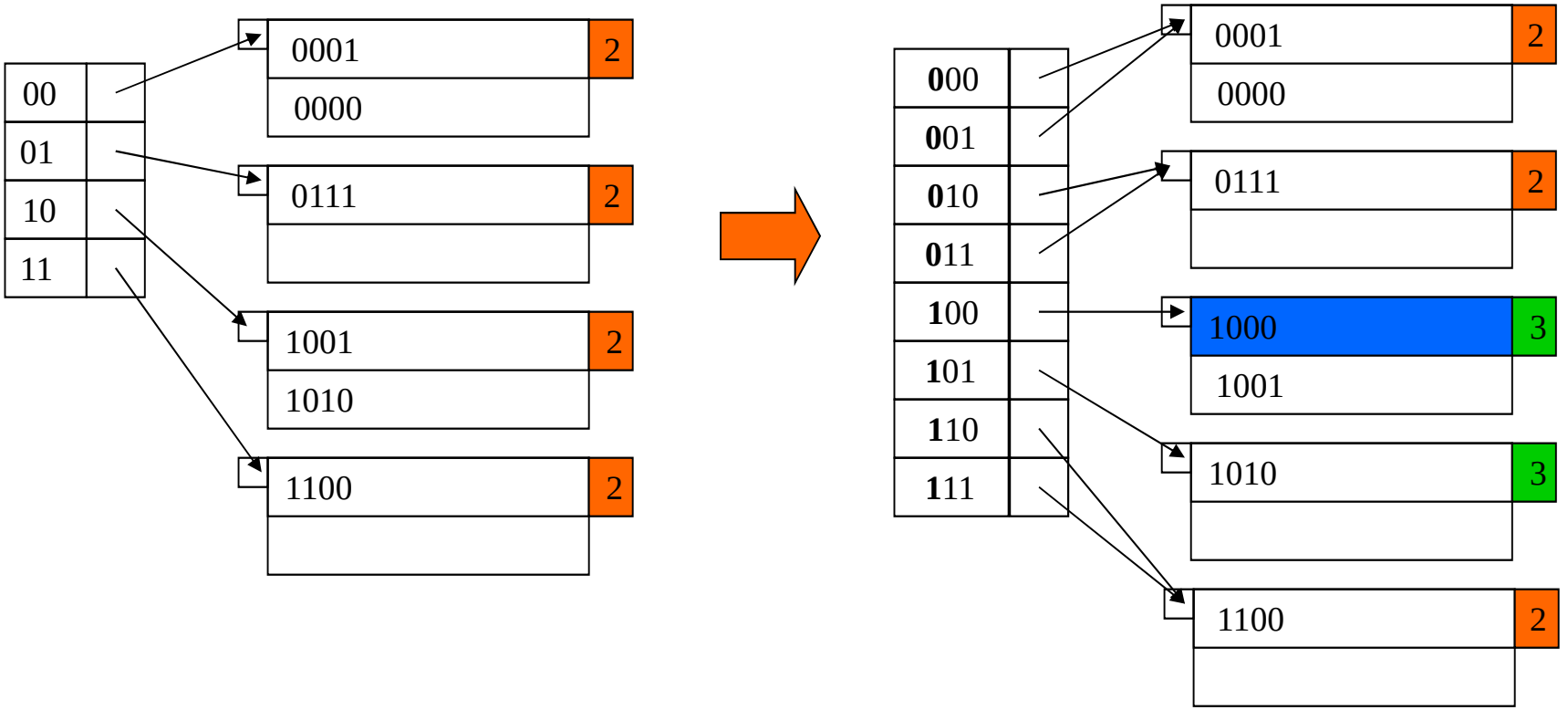
Haszowanie rozszerzalne

- Zaktualizowane zostały w nagłówkach bloków wartości mówiące o ilości używanych bitów (ale tylko tam, gdzie było to potrzebne)
- Gdyby dodać kolejne dwa rekordy z wartościami funkcji haszującej 0000 i 0111 oba powinny trafić do kubełka 0. Tym razem, mimo że nie mieszczą się w bloku, nie ma konieczności zmiany rozmiaru kubełka (i=2 jest wystarczające)
- Należy tylko zaktualizować nagłówek bloku.



Haszowanie rozszerzalne

- Dodanie w tej sytuacji rekordu z wartością funkcji haszującej 1000 spowoduje konieczność zmiany i=2 na i=3



Haszowanie rozszerzalne

- **Zaleta:**
  - Udaje się utrzymać jeden blok z danymi dla kubełka.
- **Wady:**
  - Dużo kosztuje reorganizowanie rekordów w blokach w momencie podwajania kubełków.
  - W pewnym momencie macierz kubełków też może przestać się mieścić w pamięci i ilość dostępów do dysku zacznie się zwiększy.
  - Może się „zwyrodnić” lokalizowanie rekordów. W tym „modelowym przypadku” są tylko dwa rekordy w bloku, ale może się tak zdarzyć, że rekordów będzie mało, a mimo to **ułożą się tak niekorzystnie,** że trzeba będzie je adresować dużą ilością bitów mimo, że do pozostałych wystarczyłoby ich o wiele mniej.

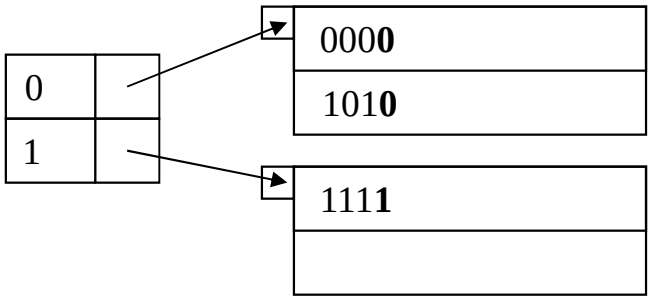
Haszowanie liniowe

- Alternatywną metodą haszowania rozszerzalnego jest haszowanie liniowe, w którym ilość kubełków rośnie znacznie wolniej.
- **Zasady**
  - Liczbę kubełków wyznacza się tak, żeby **średnia liczba rekordów na kubełek stanowiła stałą część liczby rekordów mieszczących się w bloku** (np. 80%)
  - Dopuszcza się bloki przepełnione, ale średnia liczba przepełnienia bloku nie przekracza jednego przepełnionego na kubełek
  - Liczba bitów do numerowania pozycji macierzy kubełka wynosi $\lceil \log_2 n \rceil$ (gdzie n – bieżąca liczba kubełków)
  - Do numerowania pozycji tablicy używa się i-bitów funkcji haszującej.
  - Rekord z kluczem K należy więc do kubełka $a_1,\ldots,a_i$ gdzie $a_1,\ldots,a_i$ są **ostatnimi** bitami h(K).
  - Niech m będzie równe liczbie zapisanej binarnie jako $a_1,\ldots,a_i$
    - Jeśli $m<n$ to istnieje kubełek o numerze m i umieszczamy w nim rekord
    - Jeśli $n \le m < 2^i$ to kubełek m jeszcze nie istnieje i wówczas umieszczamy rekord w kubełku $m-2^{i-1}$

    (czyli w tym, do którego trafilibyśmy, gdyby zamienić a_1 (które jest równe 1) na 0

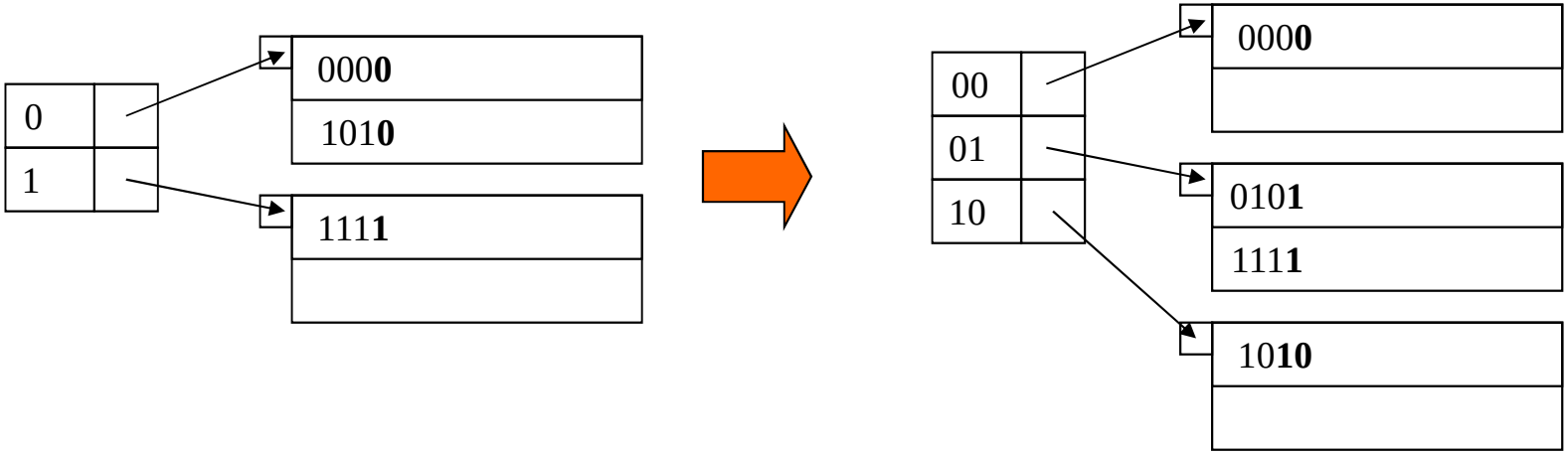
Haszowanie liniowe

- **Przykład:**
  - Niech n=2 (dwa kubełki), zatem i=1 (jeden bit wystarczy).
  - Funkcja haszująca nadal zwraca 4 bity
  - r – aktualna liczba rekordów w tablicy z haszowaniem (tutaj 3).
  - r/n – średnia ilość rekordów na kubełek.
  - Tym razem patrzmy przy przypisywaniu do kubełka na **OSTATNI bit**



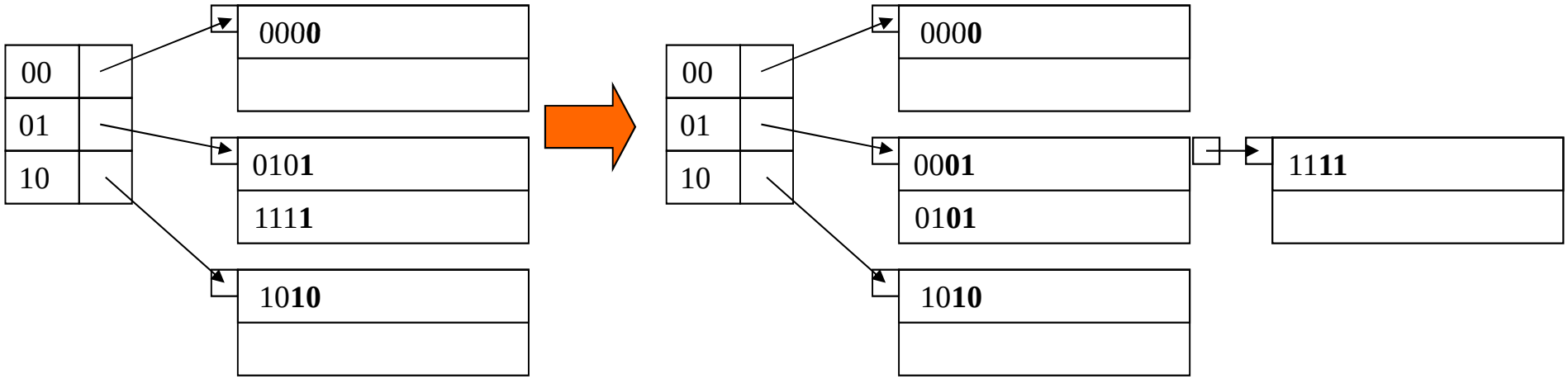
Haszowanie liniowe

- Ustalmy, że próg r/n ma jakąś wygodną w liczeniu wartość (np. 85%)
  - w bloku mieszczą się dwa rekordy, więc 2 * 85% = 1.7 Zatem r <= 1.7 n
- Wstawiając klucz o wartości po haszowaniu 0101
  - rekord trafia do drugiego kubełka, w którym jest miejsce, więc nie ma potrzeby rezerwowania nowego bloku.
  - Jednak mamy obecnie 4 rekordy (r=4, n=2), r/n > 1.7
  - Współczynnik r/n przekroczył ustaloną wartość 1.7, konieczne jest więc zwiększenie ilości kubełków (do n=3)
  - Trzeba zatem zwiększyć ilość bitów do numerowania kubełków (teraz potrzebne jest i=2)
  - Dotychczasowe kubełki 0 i 1 są więc numerowane obecnie jako 00 i 01, natomiast nowo utworzony ma numer 10.
  - **Nie jest tworzony kubełek 11, a rekord 1111 pozostaje w kubełki 01**



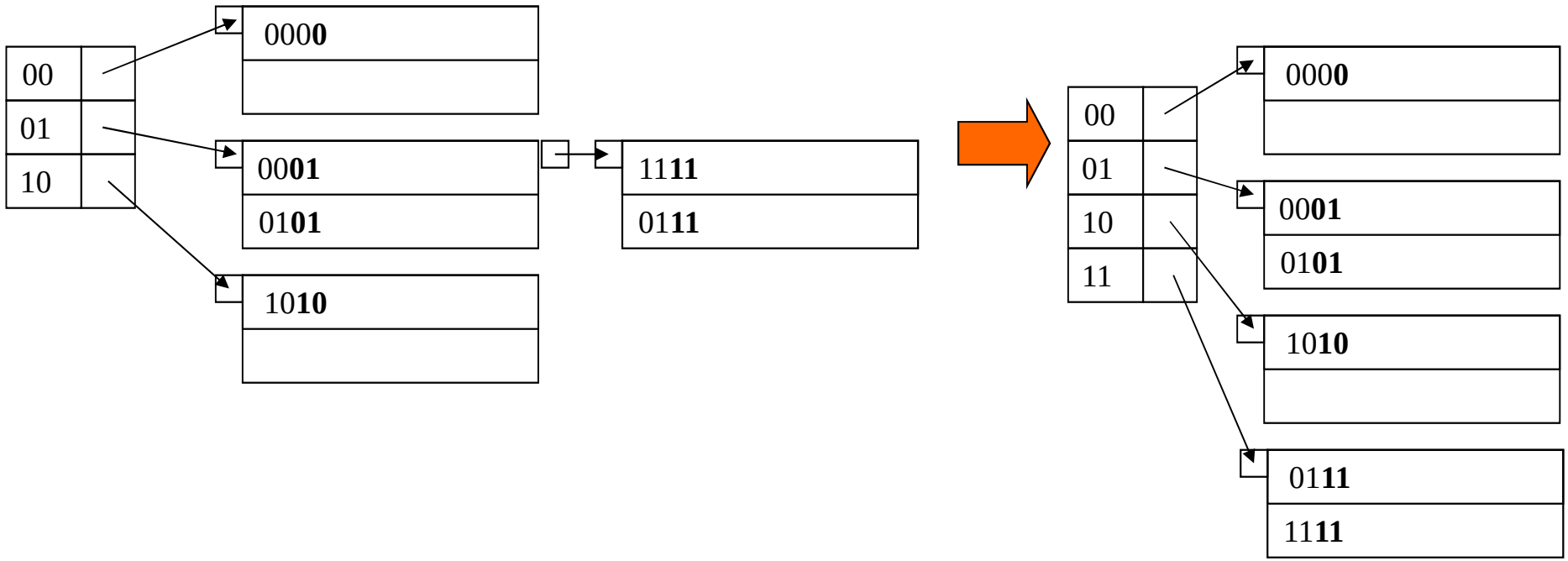
Haszowanie liniowe

- Dodajmy 0001
  - mamy obecnie 5 rekordów (r=5, n=3), r/n < 1.7. Nie powstaje nowy kubełek
  - Dane nie mieszczą się w bloku przypiętym do kubełka 0, ale nie możemy dodać nowego kubełka, bo współczynnik r/n nie przekroczył ustalonego progu.
  - Chwilowo dowiązujemy blok nadmiarowy.



Haszowanie liniowe

- Dodajmy 0111
  - ostatnie dwa bity to 11, ale taki kubełek nie istnieje.
  - Zgodnie z zasadą rekord zostaje umieszczony przy tym kubełku, który różni się tylko na pierwszym bicie (czyli w kubełku 01)
  - W tym bloku nie ma miejsca, więc rekord trafia do bloku nadmiarowego (który już istnieje)
  - mamy obecnie 6 rekordów (r=6, n=3), r/n > 1.7. Powstaje nowy kubełek (11) i rekordy zostają przeorganizowane (a zbędne bloki usunięte)



Haszowanie liniowe

- **Wyszukiwanie:**
  - gdy szukamy 1010 przy i=2, to 10 jest zapisem dwójkowym m=2
  - m<n więc kubełek musi istnieć

  - gdy szukamy 1011, to 11 jest zapisem dwójkowym m=3.
  - m>=n, więc nie istnieje kubełek 11. **Szukamy rekordu w kubełku różniącym się na pierwszym bicie** (czyli 01).
  - Jeśli tu nie zostanie znaleziony, to można stwierdzić, że taki rekord nie istnieje.