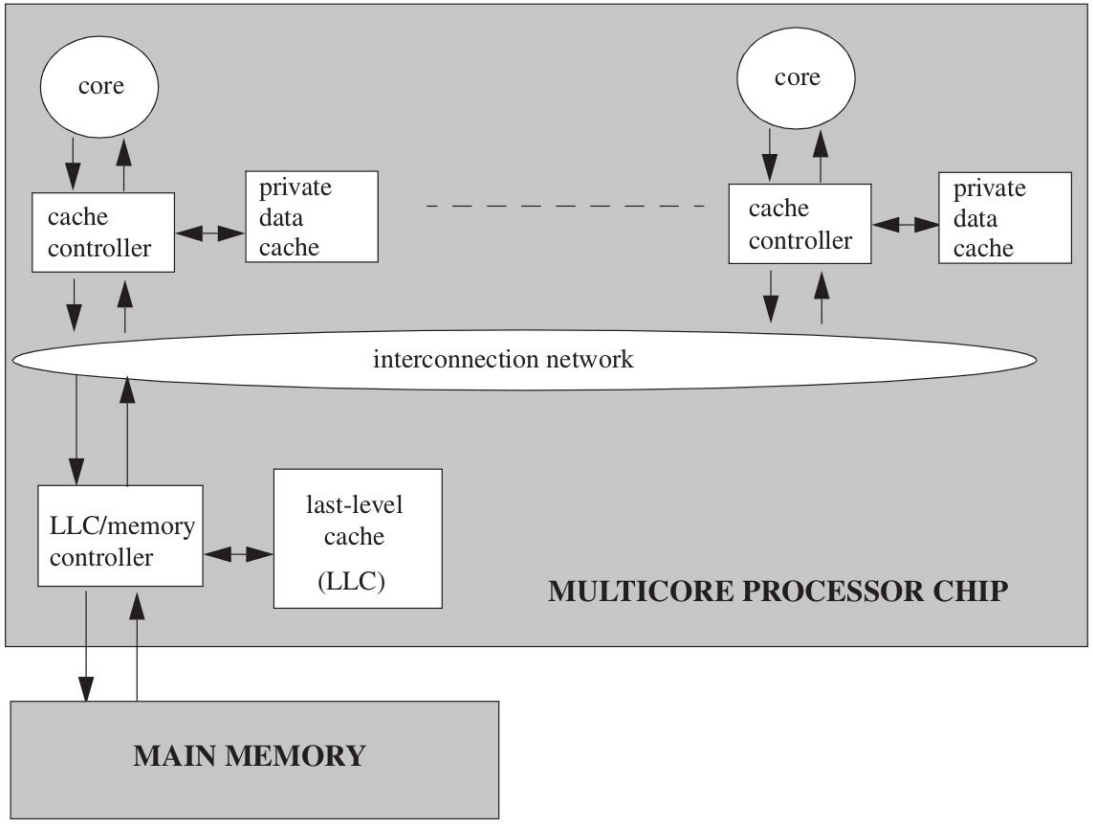
Protokoły spójności pamięci podręcznej

Seminarium ASK 2017

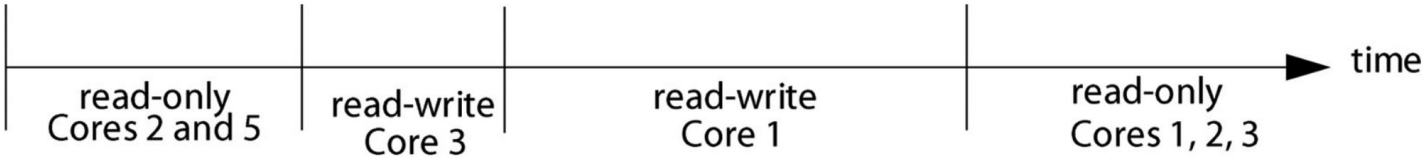
Jakub Piecuch

Protokoły spójności pamięci podręcznej



Niezmienniki spójności cache

- Single-Writer-Multiple-Reader (SWMR):
  - Dla każdej komórki pamięci, w dowolnym momencie istnieje
    - Jeden procesor mogący czytać i modyfikować tę komórkę, albo
    - Dowolna liczba procesorów mogących ją tylko odczytać
- “Życie” jednej komórki pamięci możemy podzielić na epoki odczytu i epoki odczytu-zapisu



- Wartość komórki pamięci na początku dowolnej epoki jest równa wartości tej komórki pod koniec ostatniej epoki odczytu-zapisu

Jak działa protokół?

- Do każdego prywatnego cache i do pamięci dodajemy **kontrolery spójności** *(coherence controllers)*
- Kontrolery wymieniają się wiadomościami w celu utrzymania niezmienników
- Interakcje pomiędzy kontrolerami są wyspecyfikowane przez protokół
- W naszym przykładowym systemie wyróżniamy 2 rodzaje kontrolerów:
  - **Kontrolery cache**
  - **Kontroler pamięci/LLC** (LLC - Last Level Cache)

Zadania kontrolerów

- Kontroler cache obsługuje żądania z dwóch źródeł:
  - Bezpośrednio od procesora skojarzonego z kontrolerem
    - Otrzymuje żądania zapisów i odczytów, zwraca żądane wartości
    - Cache miss powoduje rozpoczęcie transakcji poprzez wysłanie żądania dotyczącego żądanego bloku (np. uzyskanie kopii do odczytu)
  - Od innych kontrolerów w sieci
    - Otrzymuje odpowiedzi na wcześniejsze własne żądania
      - Zarówno od innych kontrolerów cache jak i od kontrolera pamięci/LLC
    - Otrzymuje żądania od innych kontrolerów cache
- Kontroler pamięci/LLC obsługuje wyłącznie żądania innych kontrolerów, nie wysyła własnych żądań

Co specyfikuje protokół?

- Kontrolery dla każdej linii implementują identyczny automat skończony
  - Każda linia ma przypisany stan
  - Odpowiedź na jakieś zdarzenie E (np. żądanie wartości do odczytu od innego kontrolera) dotyczące linii w stanie S jest funkcją E i S
  - Automat implementowany przez kontrolery cache zazwyczaj różni się od tego implementowanego przez kontroler pamięci/LLC
- Protokół specyfikuje:
  - Możliwe stany każdej linii
    - Stany stabilne
    - Stany przechodnie wynikające z faktu że transakcje nie są natychmiastowe (wymagają wymiany wiadomości pomiędzy kontrolerami)
  - Możliwe transakcje i wiadomości
  - Odpowiedzi kontrolerów cache i pamięci na wiadomości (funkcja stanu i wiadomości)

Rodzaje protokołów: snooping vs directory

Snooping protocols:

- Kontroler rozpoczyna żądanie poprzez rozgłoszenie wiadomości do wszystkich pozostałych. Kontrolery odpowiednio reagują na wiadomość (np. wysyła dane jeśli jest właścicielem)
- Większość takich protokołów zakłada że żądania docierają w globalnym liniowym porządku, co sugeruje implementację korzystającą ze wspólnej szyny do komunikacji
- Proste, ale źle się skalują wraz ze wzrostem liczby procesorów
  - Przepustowość szyny stanowi duże ograniczenie
  - W większych systemach tracimy gwarancję atomiczności rozgłoszenia wiadomości

Rodzaje protokołów: snooping vs directory

Directory protocols:

- Kontroler cache rozpoczyna żądanie linii cache poprzez wysłanie żądania do domowego kontrolera dla tej linii
- Kontroler domowy utrzymuje katalog *(directory)*, w którym przechowywany jest stan wszystkich linii (np. właściciel, kto posiada kopię tylko do odczytu, itp.)
- Kontroler domowy wysyła odpowiednią wiadomość do odpowiednich kontrolerów (np. żądanie unieważnienia starej wartości do wszystkich kontrolerów posiadających kopię linii poza tym, który wysłał początkowe żądanie)
- Nie wymaga topologii szyny, daje większą swobodę implementacji
- Nie rozgłasza wiadomości, więc ogranicza obciążenie sieci, lepiej się skaluje
- Większość transakcji wymaga jednego lub więcej dodatkowych kroków

Rodzaje protokołów: invalidate vs update

Invalidate protocols:

- Gdy procesor chce zmodyfikować linię cache, musi mieć tę linię na wyłączność (nikt inny nie może jej u siebie przechowywać, nawet w stanie tylko do odczytu)
  - To wymaga aby wszyscy inni posiadacze tej linii unieważnili starą wartość

Update protocols:

- Przy modyfikacji linii cache, nowa kopia jest rozgłaszana do wszystkich posiadaczy starej kopii
  - Nie wymaga posiadania linii na wyłączność i unieważniania
  - Potrzeba wysłania zawartości całej linii zwiększa obciążenie sieci

Prosty protokół: Valid/Invalid

- Zakładamy że wszystkie kontrolery są podłączone do wspólnej szyny
- Zakładamy że cache jest write-back, czyli zmodyfikowaną lokalną kopię danych wysyłamy do pamięci dopiero wtedy gdy musimy zastąpić ją inną linią
- 2 stabilne stany: **V**alid i **I**nvalid
- W przypadku kontrolerów cache:
  - Invalid = linii nie ma w cache
  - Valid = mamy linię na wyłączność
- W przypadku kontrolera pamięci:
  - Invalid = żaden prywatny cache nie ma linii w stanie Valid
  - Valid = dokładnie jeden prywatny cache ma linię w stanie Valid
- 1 stan pośredni: IV^D
  - Stan przejściowy przy zmianie z I na V

Prosty protokół: Valid/Invalid

- Istnieją 2 rodzaje transakcji i 3 rodzaje wiadomości
- Transakcje:
  - Get - składa się z wiadomości Get (żądanie linii) i DataResp (odpowiedź zawierająca dane)
  - Put - przekazuje linię do kontrolera pamięci/LLC w celu zapisania do pamięci
- Transakcja Get jest atomiczna (żadne inne wiadomości nie mogą zostać wysłane pomiędzy wiadomością Get i odpowiadającą jej wiadomością DataResp)

Protokół Valid/Invalid: specyfikacja kontrolera cache

TABLE 6.2: Cache Controller Specification. Shaded Entries are Impossible and Blank Entries Denote Events That are Ignored.

States	Core Events		Bus Events					
			Messages for Own Transactions			Messages for Other Cores' Transactions		
	Load or Store	**Evict Block**	**Own-Get**	**DataResp for Own-Get**	**Own-Put**	**Other-Get**	**DataResp for Other-Get**	**Other-Put**
I	issue Get /IV^D							
IV^D	stall Load or Store	stall Evict		copy data into cache, perform Load or Store /V				
V	perform Load or Store	Issue Put (with data) /I				Send DataResp /I		

Protokół Valid/Invalid: specyfikacja kontrolera pamięci/LLC

TABLE 6.3: Memory Controller Specification

State	Bus Events	
	Get	**Put**
I	send data block in DataResp message to requestor/V	
V		Update data block in memory/I

Problemy z protokołem Valid/Invalid

- Wiele procesorów nie może w tym samym czasie przechowywać w cache tej samej linii, nawet jeśli chcą z niej tylko czytać
- Przy usuwaniu linii z cache żeby zrobić miejsce dla nowych danych wykonujemy zapis do pamięci nawet jeśli dane w pamięci są aktualne
- Widać że warto rozróżnić więcej stanów niż tylko Valid/Invalid, tylko jakie?

Główne cechy charakteryzujące daną linię cache

- **Valid**:
  - Kopia w cache jest aktualna
  - Możemy z niej czytać
  - Nie możemy pisać, chyba że mamy ją na wyłączność
- **Dirty**:
  - Kopia w cache jest aktualna
  - Różni się od wartości w LLC/pamięci
  - Kontroler cache jest odpowiedzialny za zapisanie aktualnej wartości w pamięci
- **Exclusive**:
  - Nikt inny nie ma kopii tej linii w swoim prywatnym cache
- **Owned**:
  - Kontroler cache posiadający linie w tym stanie jest odpowiedzialny za odpowiadanie na żądania innych kontrolerów dotyczące tej linii
  - Nie można takiej linii wyrzucić bez przekazania praw własności do innego kontrolera

Stany MOESI

- **M**odified - valid, exclusive, owned, potentially dirty
  - Z linii można czytać i pisać
  - Cache ma jedyną aktualną kopię tej linii i musi odpowiadać na żądania dotyczące tej samej linii
  - Kopia w pamięci/LLC może być nieaktualna
- **O**wned - valid, owned, potentially dirty, not exclusive
  - Cache posiada kopię tylko do odczytu i musi odpowiadać na żądania dotyczące tej linii
  - Inni mogą mieć kopię tylko do odczytu, ale nie są jej posiadaczami
  - Kopia linii w LLC/pamięci może być nieaktualna
- **E**xclusive - valid, exclusive, not dirty
  - Nikt inny nie ma swojej prywatnej kopii
  - Kopia w pamięci/LLC jest aktualna
- **S**hared - valid, not exclusive, not owned
  - Kopia tylko do odczytu
- **I**nvalid - not valid
  - Cache nie zawiera linii lub zawiera potencjalnie nieaktualną kopię
  - Nie można czytać ani pisać

Przykład: Intel QuickPath Interconnect (QPI)

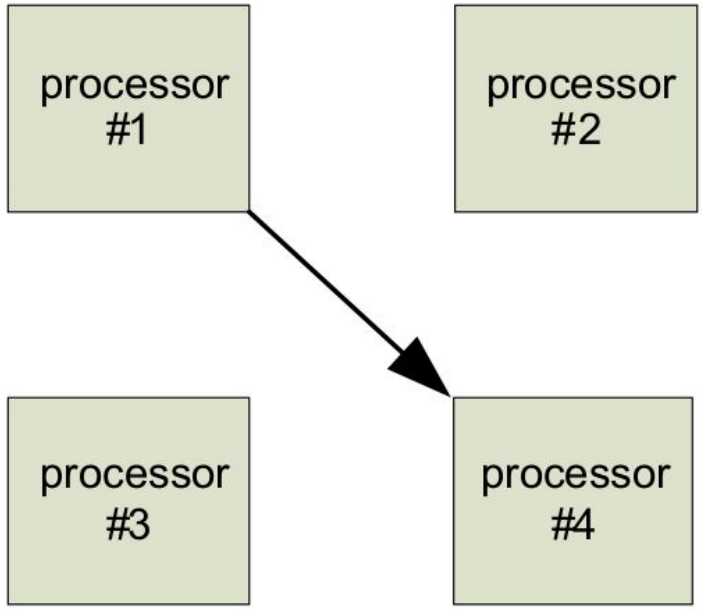
- Directory protocol
- Wyróżnia się dwa rodzaje “agentów” *(agents)*:
  - *Home agent*
  - *Caching agent*
- Struktura katalogu rozproszona pomiędzy *home agents*
  - Pośredniczą w transakcjach
- *Caching agents* mogą rozpoczynać transakcję i trzymać kopie danych w swoim prywatnym cache
  - Może też udostępniać swoje dane innym agentom
- Stany MESIF
  - **F**orward - valid, not dirty, not exclusive, owned
- Dwa rodzaje: *home snoop* i *source snoop*

Przykład transakcji: *home snoop*

- P1 - *caching agent* żądający linii cache
- P4 - *home agent* dla żądanej linii (ma informacje na temat stanu żądanej linii)
- P3 ma kopię żądanej linii w stanie M, E lub F

Krok 1

P1 wysyła żądanie do *home agent* zarządzającego żądanymi danymi.

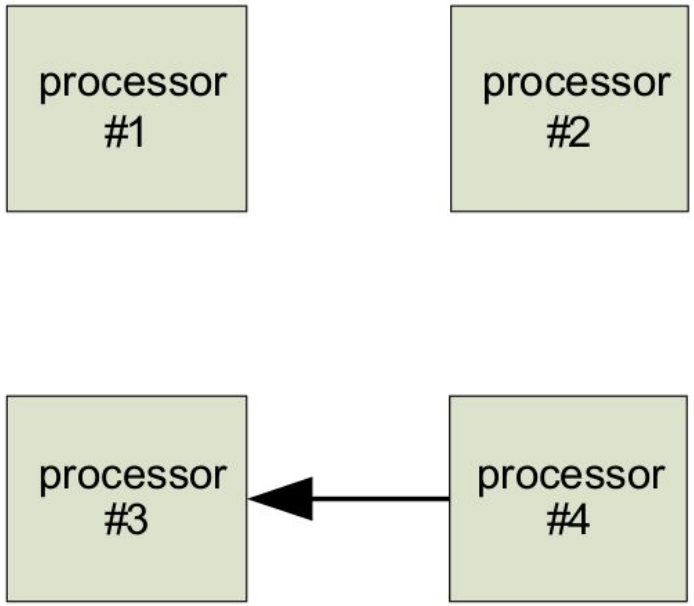


Przykład transakcji: *home snoop*

- P1 - *caching agent* żądający linii cache
- P4 - *home agent* dla żądanej linii (ma informacje na temat stanu żądanej linii)
- P3 ma kopię żądanej linii w stanie M, E lub F

Krok 2

P4 *(home agent)* wysyła *snoop request* do P3 (jeśli P3 ma dane to je wyśle do P1).

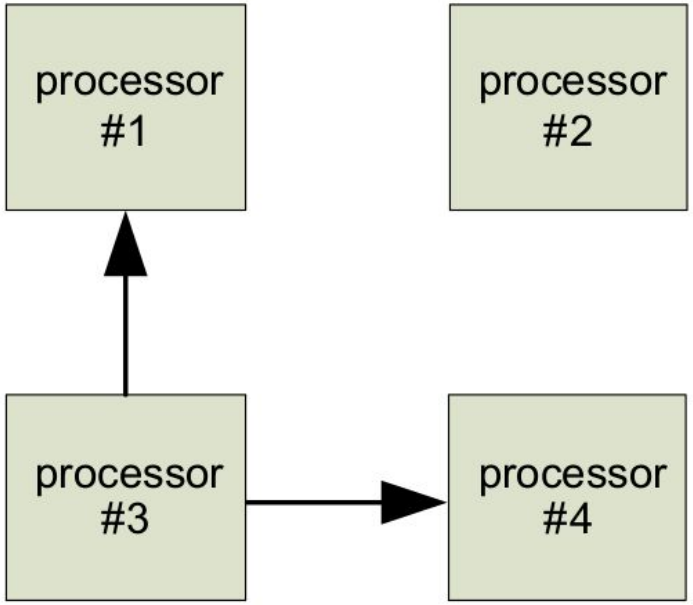


Przykład transakcji: *home snoop*

- P1 - *caching agent* żądający linii cache
- P4 - *home agent* dla żądanej linii (ma informacje na temat stanu żądanej linii)
- P3 ma kopię żądanej linii w stanie M, E lub F

Krok 3

P3 odpowiada na *snoop request* wysyłając dane do P1 i powiadamiając o tym P4.

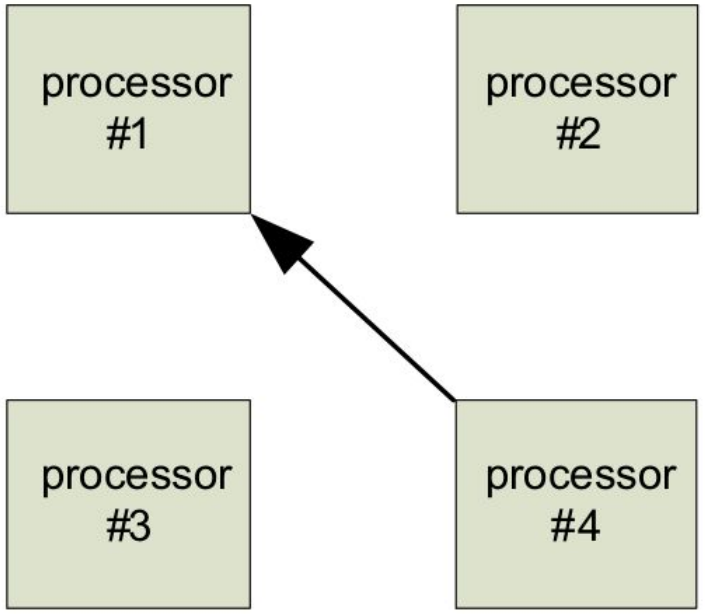


Przykład transakcji: *home snoop*

- P1 - *caching agent* żądający linii cache
- P4 - *home agent* dla żądanej linii (ma informacje na temat stanu żądanej linii)
- P3 ma kopię żądanej linii w stanie M, E lub F

Krok 4

P4 powiadamia P1 o zakończeniu transakcji.

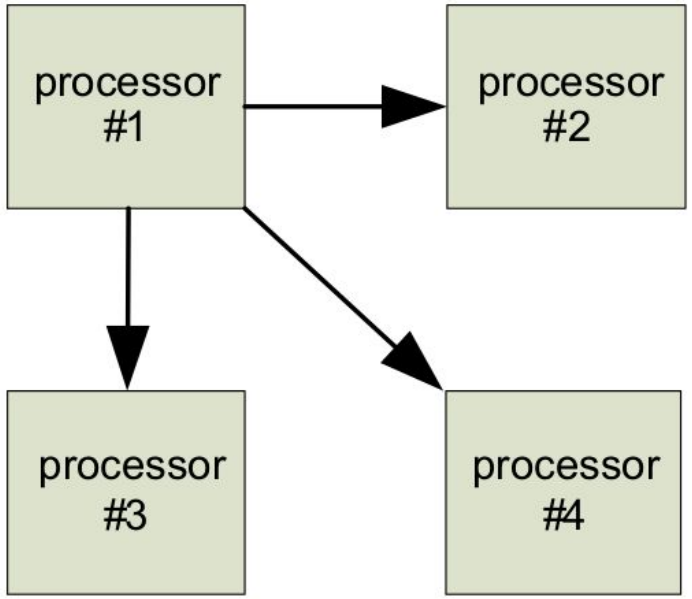


Przykład transakcji: *source snoop*

- P1 - *caching agent* żądający linii cache
- P4 - *home agent* dla żądanej linii (ma informacje na temat stanu żądanej linii)
- P3 ma kopię żądanej linii w stanie M, E lub F

Krok 1

P1 rozgłasza wysyła *snoop request* do wszystkich *caching agents.*

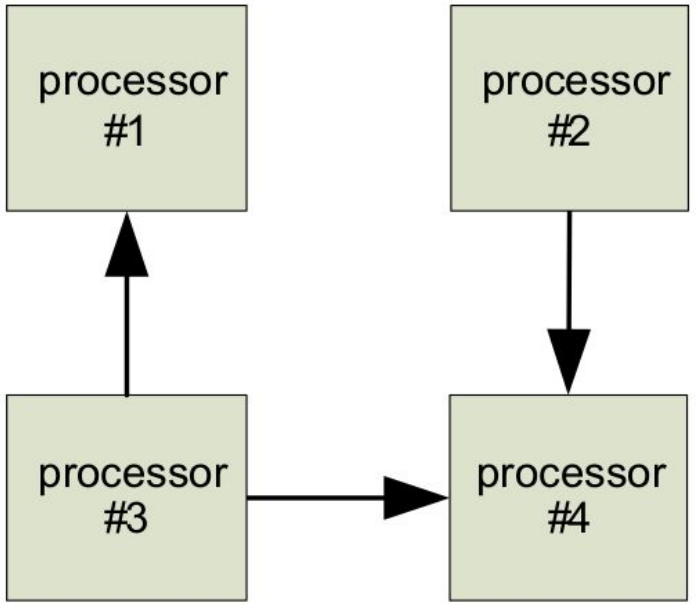


Przykład transakcji: *source snoop*

- P1 - *caching agent* żądający linii cache
- P4 - *home agent* dla żądanej linii (ma informacje na temat stanu żądanej linii)
- P3 ma kopię żądanej linii w stanie M, E lub F

Krok 2

P2 i P3 wysyłają odpowiedź na *snoop request* do P4 (*home agent*).
P3 wysyła dane do P1.

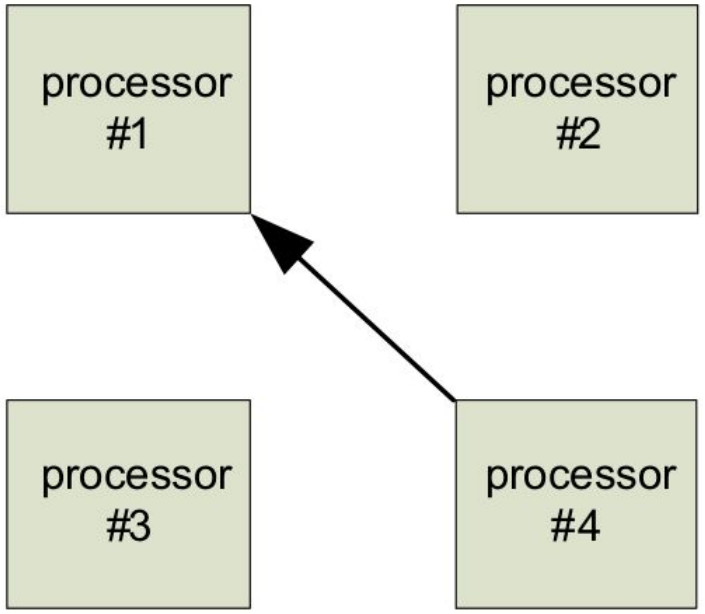


Przykład transakcji: *home snoop*

- P1 - *caching agent* żądający linii cache
- P4 - *home agent* dla żądanej linii (ma informacje na temat stanu żądanej linii)
- P3 ma kopię żądanej linii w stanie M, E lub F

Krok 3

P4 powiadamia P1 o zakończeniu transakcji.



Literatura

1. Daniel J. Sorin, Mark D. Hill, David A. Wood, *A Primer on Memory Consistency and Cache Coherence,* Morgan & Claypool Publishers, 2011
2. An Introduction to the Intel® QuickPath Interconnect, Jan. 2009