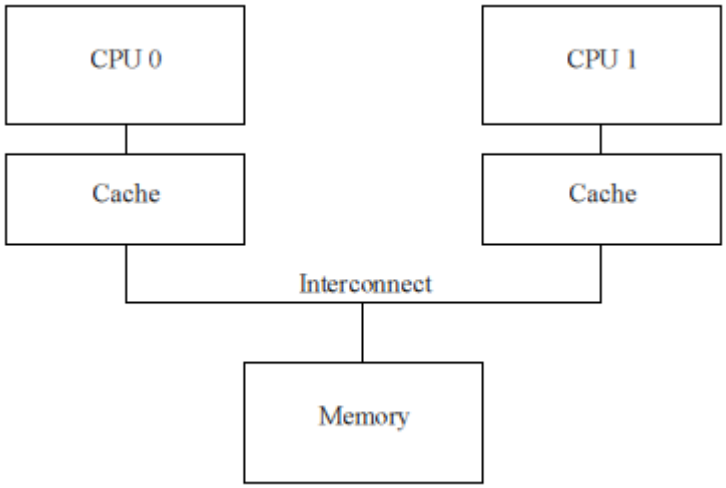
Кеши.

Кэш— промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше осуществляется быстрее, чем выборка исходных данных из более медленной памяти или удаленного источника, однако её объём существенно ограничен по сравнению с хранилищем исходных данных.

Структура кэша



Современные CPU намного быстрее подсистемы памяти. Процессор образца 2006 года мог выполнять 10 инструкций в наносекунду, но требовалось много десятков наносекунд для извлечения данных из основной памяти. Эта диспропорция в скорости (более чем на два порядка!) привела к многомегабайтным кэшам у современных процессоров. Кэши принадлежат процессорам и, как правило, время доступа к ним — несколько тактов.
CPU обменивается данными с кэшем блоками, называемыми *кэш-линиями*(cache line), размер которых обычно является степенью двойки – от 16 до 256 байт (зависит от CPU). Когда к ячейке памяти впервые обращается процессор, ячейка отсутствует в кэше, — эта ситуация называется *промахом* (cache miss или, более точно, “startup” или “warmup” cache miss). Промах означает, что CPU должен ждать (stalled) сотни тактов, пока данные не будут извлечены из памяти. Наконец данные будут загружены в кэш, и последующие обращения по данному адресу найдут данные в кэше, так что CPU будет работать на полной скорости. Спустя некоторое время кэш заполнится, и промахи будут приводить к вытеснению данных из кэша, чтобы дать место новым запрашиваемым данным. Такие промахи называются *промахами по переполнению* (capacity miss). Более того, вытеснение может произойти даже тогда, когда кэш не полон, так как он организован в железе в виде хеш-таблицы с фиксированным размером bucket’a(или *набора*, как его зовут разработчики CPU).

На рисунке справа представлен 2-ассоциативный (2-way) кэш с cache line размером 256 байт. Линия может быть пустой, что соответствует пустой ячейке таблицы. Числа слева – это адреса, данные которых может содержать ячейка. Так как линии выравнены по 256 байтам, младшие 8 бит адреса равны нулю, и хеш-функция выбирает следующие 4 бита в качестве индекса в хеш-таблице. Предположим, код программы расположен по адресам

	Way 0	Way 1
0x0	0x12345000	
0x1	0x12345100	
0x2	0x12345200	
0x3	0x12345300	
0x4	0x12345400	
0x5	0x12345500	
0x6	0x12345600	
0x7	0x12345700	
0x8	0x12345800	
0x9	0x12345900	
0xA	0x12345A00	
0xB	0x12345B00	
0xC	0x12345C00	
0xD	0x12345D00	
0xE	0x12345E00	0x43210E00
0xF		

0x43210E00 – 0x43210EFF, а сама программа последовательно обращается к данным по адресам 0x12345000 – 0x12345EFF. Пусть теперь она обращается к адресу 0x12345F00. Этот адрес хешируется на линию 0xF и обе ячейки (ways) этой линии пусты, так что 256 байт данных можно поместить в одну из них. Если программа обращается по адресу 0x1233000, хеш которой равен 0x0, то соответствующие 256-байтные данные могут быть помещены в ячейку 1 (way 1) линии 0x0. Если же программа обращается по адресу 0x1233E00 (хеш = 0xE), то одна из линий (ways) должна быть вытолкнута из кэша, чтобы освободить место под новые 256 байт данных. Если позже потребуется доступ к этим вытолкнутым данным, — получаем ситуацию промаха. Такой промах называется associativity miss.
Все это относится к чтению данных, а что будет при записи? Все CPU должны быть согласованы по данным, так что перед тем как записать, нужно удалить (invalidate) данные из кэшей других CPU. Только после того, как инвалидация завершится, процессор может безопасно записать данные. Если данные находятся в кэше CPU, но являются read-only, это называется *промахом записи* (write miss). Только после того, как CPU инвалидирует такие данные в кэшах других процессоров, CPU может повторно записать (и прочитать) эти данные. Далее, если какой-то CPU попытается обратиться к данным в то время, как другой CPU их инвалидировал для записи, — он получит промах, зовущийся *communication miss*, так как такая ситуация возникает, когда данные используются для взаимодействия (communication), как, например, мьютекс или spin-lock *[имеется в виду сам мьютекс, а не данные, которые он защищает; мьютекс – это какой-то флаг]*.
Как видно, большие усилия должны быть приложены для того, чтобы управлять *когерентностью* данных для всех CPU. Со всеми этими чтением/инвалидацией/записью легко вообразить, что данные будут потеряны или (что ещё хуже) у разных CPU будут разные данные в их кэшах. Эти проблемы решает *протокол когерентности данных* (cache-coherency protocol)

Уровни кэша

Кэш центрального процессора разделён на несколько уровней. Максимальное количество кэшей — четыре. В универсальном процессоре в настоящее время число уровней может достигать трёх. Кэш-память уровня N+1, как правило, больше по размеру и медленнее по скорости доступа и передаче данных, чем кэш-память уровня N.

•Самым быстрым является кэш первого уровня — L1 cache (level 1 cache). По сути, она является неотъемлемой частью процессора, поскольку расположена на одном с ним кристалле и входит в состав функциональных блоков. В современных процессорах обычно L1 разделен на два кэша — кэш команд (инструкций) и кэш данных (Гарвардская архитектура). Большинство процессоров без L1 не могут функционировать. L1 работает на частоте процессора, и, в общем случае, обращение к нему может производиться каждый такт. Зачастую является возможным выполнять несколько операций чтения/записи одновременно.

•Вторым по быстродействию является кэш второго уровня— L2 cache, который обычно, как и L1, расположен на одном кристалле с процессором. В ранних версиях процессоров L2 реализован в виде отдельного набора микросхем памяти на материнской плате. Объём L2 от 128 кбайт до 1−12 Мбайт. В современных многоядерных процессорах кэш второго уровня, находясь на том же кристалле, является памятью раздельного пользования— при общем объёме кэша в n Мбайт на каждое ядро приходится по n/c Мбайта, где c— количество ядер процессора.

•Кэш третьего уровня наименее быстродействующий, но он может быть очень большим— более 24 Мбайт. L3 медленнее предыдущих кэшей, но всё равно значительно быстрее, чем оперативная память. В многопроцессорных системах находится в общем пользовании и предназначен для синхронизации данных различных L2.

•Существует четвёртый уровень кэша, применение которого оправдано только для многопроцессорных высокопроизводительных серверов и мейнфреймов. Обычно он реализован отдельной микросхемой.

Когерентность кэша — свойство кэшей, означающее целостность данных, хранящихся в локальных кэшах для разделяемого ресурса. Когерентность кэшей — частный случайкогерентности памяти.

Когерентность памяти — свойство компьютерных систем, в которых два или более процессора или ядра имеют доступ к общей области памяти.
Воднопроцессорных системах (более строго — в одноядерных) лишь один процессорный узел выполняет всю работу, следовательно, только он один может выполнять чтение и запись определённой области памяти. В результате, когда содержимое ячейки меняется, все последующие операции чтения по данному адресу получат обновлённое значение даже при наличии кэширования.

С другой стороны, в многопроцессорных (многоядерных) системах несколько процессорных узлов работают одновременно, поэтому возможна ситуация параллельного доступа к одной ячейке памяти. При условии, что ни один из них не меняет значение данной ячейки, они могут сводобно пользоваться ей совместно, кэшируя по своему усмотрению. Но как только один из них обновляет значение ячейки, данные в локальных кэшах других узлов могут оказаться устаревшими. Следовательно, необходим механизм уведомления всех узлов об изменении значения в общей памяти; такой механизм называется протоколом когерентности. Если подобный протокол применён, то говорят, что система имеет «**когерентную память**».

Точная природа и смысл механизма когерентности определяется моделью консистентности, реализованной в протоколе. Чтобы писать правильные параллельные программы, программисты должны быть в курсе того, какая именно модель консистентности применена в их системах.

Когерентность определяет поведение чтений и записей в одно и то же место памяти. Кэш называется когерентным, если выполняются следующие условия:

1.Если процессор P записывает значение в переменную X, то при следующем считывании X он должен получить ранее записанное значение, если между записью и чтением X другой процессор не производил запись в X. Это условие связано с сохранением порядка выполнения программы, это должно выполняться и для однопоточной архитектуры.

2.Операция чтения X процессором P_1, следующая после того, как другой процессор P_2 существил запись в X, должна вернуть записанное значение, если другие процессоры не изменяли X между двумя операциями. Это условие определяет понятие когерентной видимости памяти.

3.Записи в одну и ту же ячейку памяти должны быть последовательными. Другими словами, если два процессора записывают в переменную X два значения: A, затем B — не должно случиться так, чтобы при считывании процессор сначала получал значение B, а затем A.

В этих условиях предполагается, что операции чтения и записи происходят мгновенно.

Однако этого не происходит на практике из-за задержек памяти и других особенностей архитектуры. Изменения, сделанные процессором P_1, могут быть не видны процессору P_2, если чтение произошло через очень маленький промежуток времени после записи. Модель консистентности памяти определяет, когда записанное значение будет видно при чтении из другого потока.

Механизмы когерентности кэшей

•Когерентность с использованием справочника (*directory*). Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы).

•Когерентность с использованием отслеживания (snooping). Каждый кэш, который содержит копию данных некоторого блока физической памяти, имеет также соответствующую копию служебной информации о его состоянии. Централизованная система записей отсутствует. Обычно кэши расположены на общей (разделяемой) шине и контроллеры всех кэшей наблюдают за шиной (просматривают ее) для определения того, не содержат ли они копию соответствующего блока.

•Перехват (snarfing). Когда из какого-либо одного кэша данные переписываются в оперативную память, контроллеры остальных получают сигнал об этом изменении ("перехватывают" информацию об изменении данных) и, если необходимо, изменяют соответствующие данные в своих кэшах.

Системы распределенной разделяемой памяти используют похожие механизмы для поддержания корректности между блоками памяти в слабосвязанных системах.

Протоколы поддержки когерентности отвечают за поддержание корректности данных между всеми кэшами в системе. Протокол поддерживает когерентность памяти согласно выбранной модели.

Протокол MSI.

В протоколе MSI каждый блок памяти может иметь одно из трех состояний:

MODIFIED – данные в блоке изменены и не согласуются с данными в основной памяти. Блок в этом состоянии должен будет перезаписать данные в основную память.

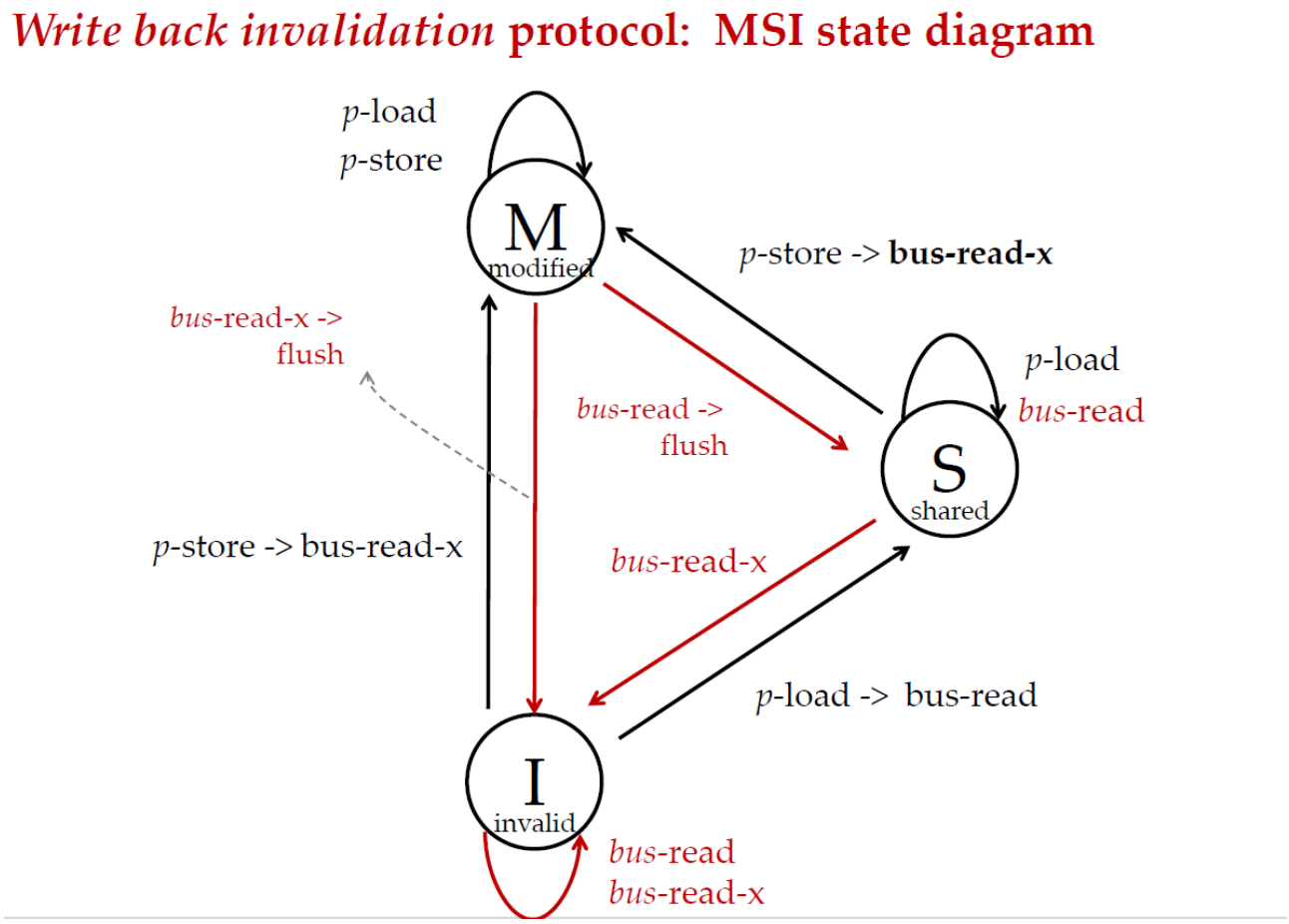
SHARED – данные в блоке не модифицированы. В этом состоянии блок предназначен только для чтения до тех пор, пока остается хотя бы один блок в таком состоянии.

INVALID – данные в этом блоке невалидны и должны быть записаны из других кэшей или основной памяти.

Когда запрос на чтение прибывает в кэш для блока в состоянии "M" или "S" , кэш отдает данные. Если блок в состоянии "I", он должен убедиться, что нет этого блока в состоянии "M" в любом другом кэше. Если другой кэш имеет блок в состоянии "M", то он должен

записать данные в основную память и перейти в состояние "S" или "I". После того, как любой блок "M" записывается обратно, кэш получает данные либо из резервного хранилища, или другого кэша с блоком в состоянии «S». Теперь кэш может поставлять данные запрашивающей стороне. После подачи данных, блок кэш-памяти находится в состоянии «S».

Когда запрос записи прибывает в кэш для блока в состоянии "M", кэш изменяет данные локально. Если ячейка находится в состоянии "S", кэш должен уведомить любые другие кэши, которые могут содержать блок в состоянии "S", что они должны инвалидировать ячейку. Затем данные могут быть изменены локально. Если блок находится в состоянии "I", кэш должен уведомить любые другие кэши, которые могут содержать блок в "S" или "M" говорится, что они должны инвалидировать блок. Если блок находится в другом кэше в состоянии "M", то кэш должен либо написать данные в основную память, либо поставить его запрашивающему кэшу. Если в этот момент кэш еще не имеют блок на уровне кэша, блок считывается из основной памяти, прежде чем изменять в памяти кэша. После модификации данных, блок кэш-памяти находится в состоянии "M".



Для любой пары кешей разрешены следующие состояния заданной кэш-строки:

	MODIFIED	*SHARED*	*INVALID*
MODIFIED	NO	NO	YES
SHARED	NO	YES	YES
INVALID	YES	YES	YES

Протокол MESI
Основная статья: http://habrahabr.ru/company/ifree/blog/196548/

Состояния MESI
MESI – это 4 возможных состояния кэш-линии: Modified-Exclusive-Shared-Invalid. Для поддержки этого протокола каждая линия, помимо собственно данных, должна иметь двухбитовый тег, хранящий её состояние.

Состояние *Modified* означает, что данные в кэш-линии только что записаны процессором-владельцем кэша, и гарантируется, что изменение ещё не появилось в кэшах других CPU. Можно сказать, что данными владеет один CPU. Так как данные в такой линии свежайшие, линия кэша готова к записи в память (или в кэш следующего уровня), и запись должна быть произведена прежде, чем линию можно будет заполнить другими данными.

Состояние *Exclusive* весьма похоже на Modified, за исключением того, что данные ещё не изменены процессором-владельцем кэша. Это, в свою очередь, означает, что линия содержит наисвежайшие данные, согласованные с памятью. CPU-владелец может писать в эту кэш-линию в любое время без оповещения других CPU, равно как и вытеснить её без какой-либо записи обратно в память.

Состояние *Shared* говорит, что линия продублирована по крайней мере в ещё одном кэше другого CPU. CPU-владелец такой линии не может писать в неё без предварительного согласования с другими CPU. Как и для состояния Exclusive, линия согласована с памятью, и может быть вытеснена без оповещения других CPU и без записи обратно в память.

Линия кэша в состоянии*Invalid* пуста, то есть не содержит никаких данных (или содержит мусор). Когда в кэш требуется подгрузить новые данные, они по возможности помещаются в Invalid-линию.

Так как все CPU должны поддерживать когерентность кэшей системы в целом, протокол описывает сообщения, с помощью которых координируются изменения состояния линий кэша всех процессоров.
Сообщения протокола MESI

Многие из переходов из одного состояния в другое требуют взаимодействия между CPU. Если все CPU подсоединены к единой разделяемой шине, достаточно следующих сообщений:

- **Read** (Чтение): это сообщение содержит физический адрес читаемой кэш-линии

•**Read Response** (Ответ на чтение): содержит данные, запрошенные предыдущим сообщением *Read*. *Read Response* может прийти от подсистемы памяти или от другого кэша. Например, если данные уже находятся в каком-либо кэше в состоянии “modified”, то такой кэш может ответить на *Read*

•**Invalidate** (Инвалидация): содержит физический адрес кэш-линии, которую требуется инвалидировать. *Все* прочие кэши должны удалить данные и ответить на это сообщение.

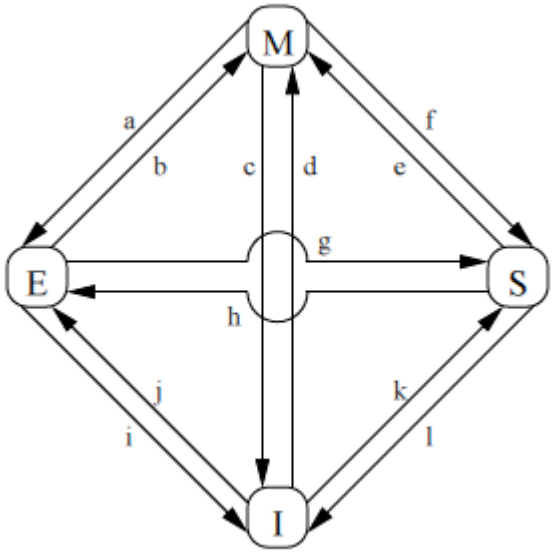
•**Invalidate Acknowledge** (Подтверждение инвалидации): CPU, получивший *Invalidate*, должен удалить требуемые данные и подтвердить это сообщением *Invalidate Acknowledge*

•**Read Invalidate** (Чтение с инвалидацией): это сообщение содержит физический адрес читаемой кэш-линии. В то же время оно диктует другим кэшам удалить эти данные. По сути, это комбинация сообщений *Read* и *Invalidate*. Это сообщение требует ответа*Read Response* и множества *Invalidate Acknowledge*

•**Writeback** (Запись в память): содержит адрес и собственно данные для записи в память. Разрешает кэшам очистить соответствующую линию в состоянии *Modified*

Диаграмма состояний MESI

Переходы на рисунке справа имеют следующее значение:



•**Переход a (M -> E)**: Кэш-линия записана обратно в память, но CPU оставил её в кэше и имеет право изменять её. Этот переход требует сообщения “writeback”.

•**Переход b (E -> M)**: CPU пишет в кэш-линию, к которой имеет эксклюзивный доступ. Этот переход не требует каких-либо сообщений.

•**Переход c (M -> I)**: CPU получает сообщение “read invalidate” кэш-линии в состоянии “Modified”. CPU должен удалить свою локальную копию и ответить сообщениями “read response” и “invalidate acknowledge”. Тем самым CPU отправляет данные и показывает, что более не имеет их копии у себя

•**Переход d (I -> M)**: CPU производит read-modify-write (RMW) операцию над данными, не находящимися в кэше. Он издает “read invalidate” сигнал, получая данные в “read response”. CPU завершает переход только когда получит все ответы “invalidate

acknowledge”

•**Переход e (S -> M)**: CPU производит read-modify-write (RMW) операцию над данными, которые были read-only. Он должен инициировать “invalidate” сигнал и ждать, пока не получит полный набор ответов “invalidate acknowledge”

•**Переход f (M -> S)**: Какой-то другой процессор читает данные, и эти данные находятся в кэше нашего CPU. В результате данные становятся read-only, что может привести к записи в память. Этот переход инициируется приемом сигнала “read”. CPU отвечает сообщением “read response”, содержащим запрашиваемые данные

•**Переход g (E -> S)**: Какой-то другой процессор читает данные, и эти данные находятся в кэше нашего CPU. Данные становятся разделяемыми и, следовательно, read-only. Переход инициируется приемом сигнала “read”. CPU отвечает сообщением “read response”, содержащим запрашиваемые данные

•**Переход h (S -> E)**: CPU решает, что ему надо записать данные в кэш-линию, и поэтому посылает “invalidate” сообщение. Переход не завершается, пока CPU не получит полный набор ответов “invalidate acknowledge”. Другие CPU выкидывают кэш-линию из своих кэшей сообщением “writeback”, так что наш CPU становится единственным, кэширующим эти данные

•**Переход i (E -> I)**: Другой CPU выполняет RMW-операцию с данными, которыми владеет наш CPU, так что наш процессор инвалидирует кэш-линию. Переход начинается с сообщения “read invalidate”, наш CPU отвечает сообщениями “read response” и “invalidate acknowledge”.

•**Переход j (I -> E)**: CPU сохраняет данные в новую кэш-линию и передает сообщение “read invalidate”. CPU не может закончить переход, пока не получит “read response” и полный набор “invalidate acknowledge”. Как только запись завершится, кэш-линия переходит в состояние “modified” через переход (b)

•**Переход k (I -> S)**: CPU загружает данные в новую кэш-линию. CPU посылает “read”-сообщение и завершает переход, получив “read response”

•**Переход l (S -> I)**: Другой CPU хочет сохранить данные в кэш-линию, которая имеет статус read-only, так как какой-то третий CPU (или, например, наш) разделяет данные с нами. Переход начинается приемом “invalidate”, и наш CPU отвечает “invalidate acknowledge”

Для любой пары кешей разрешены следующие состояния заданной кэш-строки:

	MODIFIED	*EXCLUSIVE*	*SHARED*	*INVALID*
MODIFIED	NO	NO	NO	YES
EXCLUSIVE	NO	NO	NO	YES
SHARED	NO	NO	YES	YES
INVALID	YES	YES	YES	YES

Пример протокола MESI

Давайте посмотрим, как работает MESI с точки зрения кэш-линии, на примере 4-процессороной системы с прямым отображением памяти в кэш. Данные находятся в памяти по адресу 0. Следующая таблица иллюстрирует изменение данных. Первая колонка – это последовательный номер операции, вторая – номер процессора, выполняющего операцию, третья – какая операция выполняется, следующие четыре колонки – состояние кэш-линии каждого CPU (в виде адрес памяти/состояние), заключительные две колонки – содержит ли память корректные данные (V) или нет (I).

			CPU Cache				Memory	
Sequence #	CPU #	Operation	0	1	2	3	0	8
0		Initial State	-/I	-/I	-/I	-/I	V	V
1	0	Load	0/S	-/I	-/I	-/I	V	V
2	3	Load	0/S	-/I	-/I	0/S	V	V
3	0	Invalidation	8/S	-/I	-/I	0/S	V	V
4	2	RMW	8/S	-/I	0/E	-/I	V	V
5	2	Store	8/S	-/I	0/M	-/I	I	V
6	1	Atomic Inc	8/S	0/M	-/I	-/I	I	V
7	1	Writeback	8/S	8/S	-/I	-/I	V	V

Сначала линии кэша CPU находятся в состоянии “invalid”, а память содержит корректные данные. Когда CPU 0 читает данные по адресу 0, кэш-линия CPU 0 переходит в состояние “shared” и согласована с памятью. CPU 3 также читает данные по адресу 0, так что кэш-линии переходят в состояние “shared” в кэшах обоих CPU, но по-прежнему согласованы с памятью. Далее CPU 0 загружает данные по адресу 8, что приводит к вытеснению кэш-линии и помещению в неё новых данных (прочитанных по адресу 8). Далее, CPU 2 читает данные по адресу 0, но затем понимает, что ему необходимо записать их (RMW-операция), так что он использует сигнал “read invalidate”, чтобы убедиться, что он имеет эксклюзивную копию данных. Сигнал “read invalidate” инвалидирует кэш-линию CPU 3 (хотя его данные пока остаются согласованными с памятью). Затем CPU 2 выполняет запись, являющуюся составной частью RMW-операции, что переводит кэш-линию в состояние “Modified”. Данные в памяти теперь устаревшие. CPU 1 выполняет атомарный инкремент и использует сигнал “read invalidate”, чтобы получить данные. Данные он получает из кэша CPU 2, а в самом CPU 2 данные инвалидируются. В результате данные находятся в кэш-линии CPU 1 в состоянии “modified” и по-прежнему не согласованы с памятью. Наконец, CPU 1 читает данные по адресу 8, что приводит к сбросу кэш-линии в память (используется сообщение “writeback”).

Протокол MOESI

MOESI — протокол поддержки когерентности кэшей микропроцессоров, включающий в себя все возможные состояния, используемые в других протоколах. В дополнение к состояниям часто используемого протокола MESI, добавлено пятое состояние «Owned», означающее что данные одновременно и модифицированы и разделяются (*modified* и *shared*). Оно позволяет избежать необходимости записи модифицированных данных обратно в основную память, прежде чем другие процессоры системы смогут ее прочесть. Данные все еще необходимо записать в память, но с данным протоколом эта запись (write-back) может быть отложена.

Каждая линия кэша находится в одном из пяти состояний:

Модифицирована (Modified)
Кэш-линия в модифицированном состоянии содержит наиболее свежие, корректные данные. Копия данных в основной памяти устарела и некорректна, и ни один другой процессор не имеет копии данных. Данные в кэш-линии могут быть повторно модифицированы без каких-либо запросов и изменений состояния. Состояние может поменяться на *Exclusive* при записи измененных данных в основную память.

Владелец (Owned)
Кэш-линия в состоянии owned содержит наиболее свежие, корректные данные. Состояние *Owned* похоже на состояние *Shared* тем, что другие процессоры могут иметь копию наиболее свежих и корректных данных. В отличие от состояния *Shared*, однако, копия в основной памяти может быть устаревшей и некорректной. Только один из процессоров может иметь данную кеш-линию в состоянии *Owned*, все остальные процессоры могут иметь эти данные только в состоянии *Shared*. Кеш-линия может перейти в состояние *Modified* после снятия актуальности (принудительного перевода в состояние *Invalid*) всех разделяемых копий в других процессорах, или в состояние *Shared* при записи измененных данных в основную память.

Эксклюзивное (Exclusive)
Кэш-линия в эксклюзивном состоянии содержит наиболее свежие, корректные данные. Копия в основной памяти также содержит наиболее свежую, корректную копию данных. Ни один другой процессор не имеет копии данных в своем кэше. Состояние может измениться на *Modified* в любой момент для модификации содержимого данной кэш-линии. Также состояние в любой момент может измениться на*Invalid*.

Разделяемое (Shared)
Кэш-линия в разделяемом состоянии содержит наиболее свежие, корректные данные. Другие процессоры в системе могут иметь копии данных в разделяемом состоянии. Копия в основной памяти также содержит наиболее свежую, корректную копию данных, если ни один другой процессор не имеет данной кэш-линии в состоянии owned. Запись в данную кэш-линию запрещенна, и требует перевода ее в эксклюзивное состояние с одновременным переводом всех остальных разделяемых копий в состояние *Invalid*. Также состояние может измениться на *Invalid* в любой момент.

Не актуальное (Invalid)
Кэш-линия в разделяемом состоянии не содержит корректных данных. Корректные копии данных могут быть либо в основной памяти, либо в кэше другого процессора.

Для любой пары кешей разрешены следующие состояния заданной кэш-строки:

	MODIFIED	*OWNED*	*EXCLUSIVE*	*SHARED*	*INVALID*
MODIFIED	NO	NO	NO	NO	YES
OWNED	NO	NO	NO	YES	YES
EXCLUSIVE	NO	NO	NO	NO	YES
SHARED	NO	YES	NO	YES	YES
INVALID	YES	YES	YES	YES	YES

Протокол MOSI

Почти как MSI, но добавлено состояние Владелец (Owned), которое указывает, что текущему процессору принадлежит этот блок, и он будет обслуживать запросы от других процессоров для этого блока.

Для любой пары кешей разрешены следующие состояния заданной кэш-строки:

	MODIFIED	*OWNED*	*SHARED*	*INVALID*
MODIFIED	NO	NO	NO	YES
OWNED	NO	NO	YES	YES
SHARED	NO	YES	YES	YES
INVALID	YES	YES	YES	YES

Протокол MESIF

MESIF — протокол поддержки когерентности кешей и памяти. Протокол основан на протоколеMESI, в который добавлено еще одно состояние. В новом протоколе 5 состояний: *Modified*(M), *Exclusive* (E), *Shared* (S), *Invalid* (I) и *Forward* (F). Дополнительное состояние F означает, что кэш является единственным ответчиком (designated responder) для любых запросов к данной кэш линии. Кэшстрока в состоянии S теперь не отвечает на снуп-запросы. При копировании F-строки в соседний кэш новая копия получает F состояние.

Для любой пары кэшей разрешены следующие комбинации состояний заданной кэш линии в разных процессорах:

	MODIFIED	*EXCLUSIVE*	*SHARED*	*INVALID*	*FORWARD*
MODIFIED	NO	NO	NO	YES	NO
EXCLUSIVE	NO	NO	NO	YES	NO
SHARED	NO	NO	YES	YES	YES
INVALID	YES	YES	YES	YES	YES
FORWARD	NO	NO	YES	YES	NO