Параллельные компьютерные архитектуры

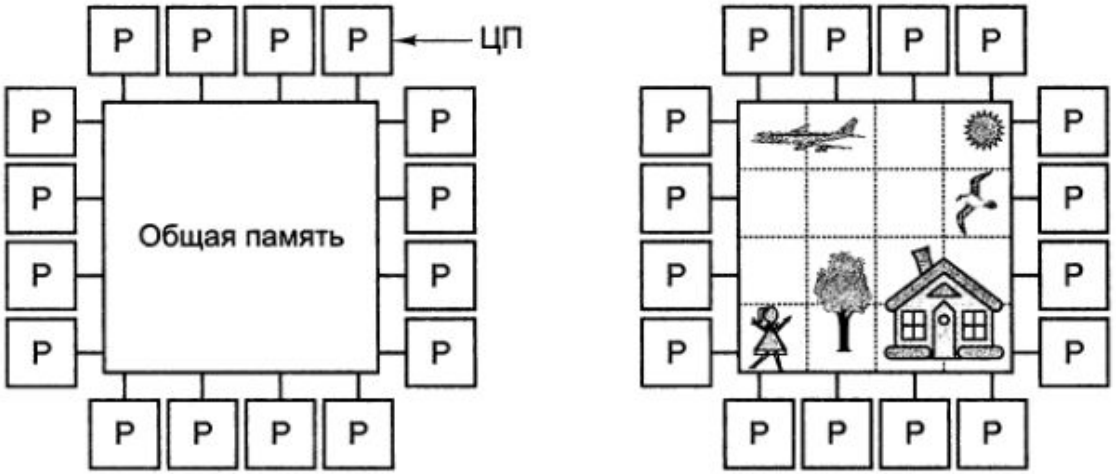
Ч.2

Мультипроцессоры и мультикомпьютеры

- В любой параллельной компьютерной системе процессоры, выполняющие разные части единого задания, должны как-то взаимодействовать друг с другом, чтобы обмениваться информацией
- Для обмена информацией предложено и реализовано две стратегии: мультипроцессоры и мультикомпьютеры.
- Ключевое различие между стратегиями состоит в наличии или отсутствии общей памяти
- Это различие сказывается как на конструкции, устройстве и программировании таких систем, так и на их стоимости и размерах

Мультипроцессоры

- Параллельный компьютер, в котором все процессоры совместно используют общую физическую память, называется мультипроцессором, или системой с об-общей памятью



- Два процесса имеют возможность легко обмениваться информацией — для этого один из них просто записывает данные в общую память, а другой их считывает

- Все процессоры в мультипроцессоре используют единое адресное пространство → функционирует только одна копия операционной системы
- Организация, в основе которой лежит единая система, и отличает мультипроцессор от мультикомпьютера
- В одних мультипроцессорных системах только определенные процессоры получают доступ к устройствам ввода-вывода, в других - каждый процессор может получить доступ к любому устройству ввода-вывода
- Если все процессоры имеют равный доступ ко всем модулям памяти и всем устройствам ввода-вывода, и между процессорами возможна полная взаимозаменяемость, такой мультипроцессор называется симметричным (Symmetric Multiprocessor, SMP)

Мультикомпьютеры

- Во втором варианте параллельной архитектуры каждый процессор имеет собственную память, доступную только этому процессору
- Такая схема называется мультикомпьютером, или системой с распределенной памятью
- Каждый процессор в мультикомпьютере имеет собственную локальную память, к которой этот процессор может обращаться, но никакой другой процессор не может получить доступ к локальной памяти данного процессора
- Мультипроцессоры имеют одно физическое адресное пространство, разделяемое всеми процессорами, а мультикомпьютеры содержат отдельные физические адресные пространства для каждого процессора

- Поскольку процессоры в мультикомпьютере не могут взаимодействовать друг с другом простыми обращениями к общей памяти, процессоры обмениваются сообщениями через связывающую их коммуникационную сеть
- Примеры мультикомпьютеров - IBM BlueGene/L, Red Storm, кластер Google

Обмен данными

- При отсутствии общей памяти, реализованной аппаратно, предполагается определенная программная структура
- Пример: *Сначала процессору 0 нужно как-то выяснить, какой процессор содержит необходимые ему данные, и послать этому процессору сообщение с запросом копии данных→ затем процессор 0 блокируется до получения ответа → когда процессор 1 получает сообщение, оно программно анализируется, после чего затребованные данные передаются обратно → когда процессор 0 получает ответное сообщение, блокировка программно снимается, и процессор продолжает работу*
- В мультикомпьютере для взаимодействия между процессорами часто используются примитивы *send* и *receive*.
- ПО мультикомпьютера имеет более сложную структуру, чем ПО мультипроцессора

Вопрос?

- Зачем вообще создавать мультикомпьютеры, если мультипроцессоры гораздо проще программировать?

Вопрос?

- Зачем вообще создавать мультикомпьютеры, если мультипроцессоры гораздо проще программировать?

Ответ

- Создать большой мультикомпьютер проще и дешевле, чем мультипроцессор с таким же количеством процессоров.
- Реализация общей памяти, совместно используемой несколькими сотнями процессоров, — это сложная задача, а разработать мультикомпьютер, содержащий 10 000 процессоров и более, довольно легко

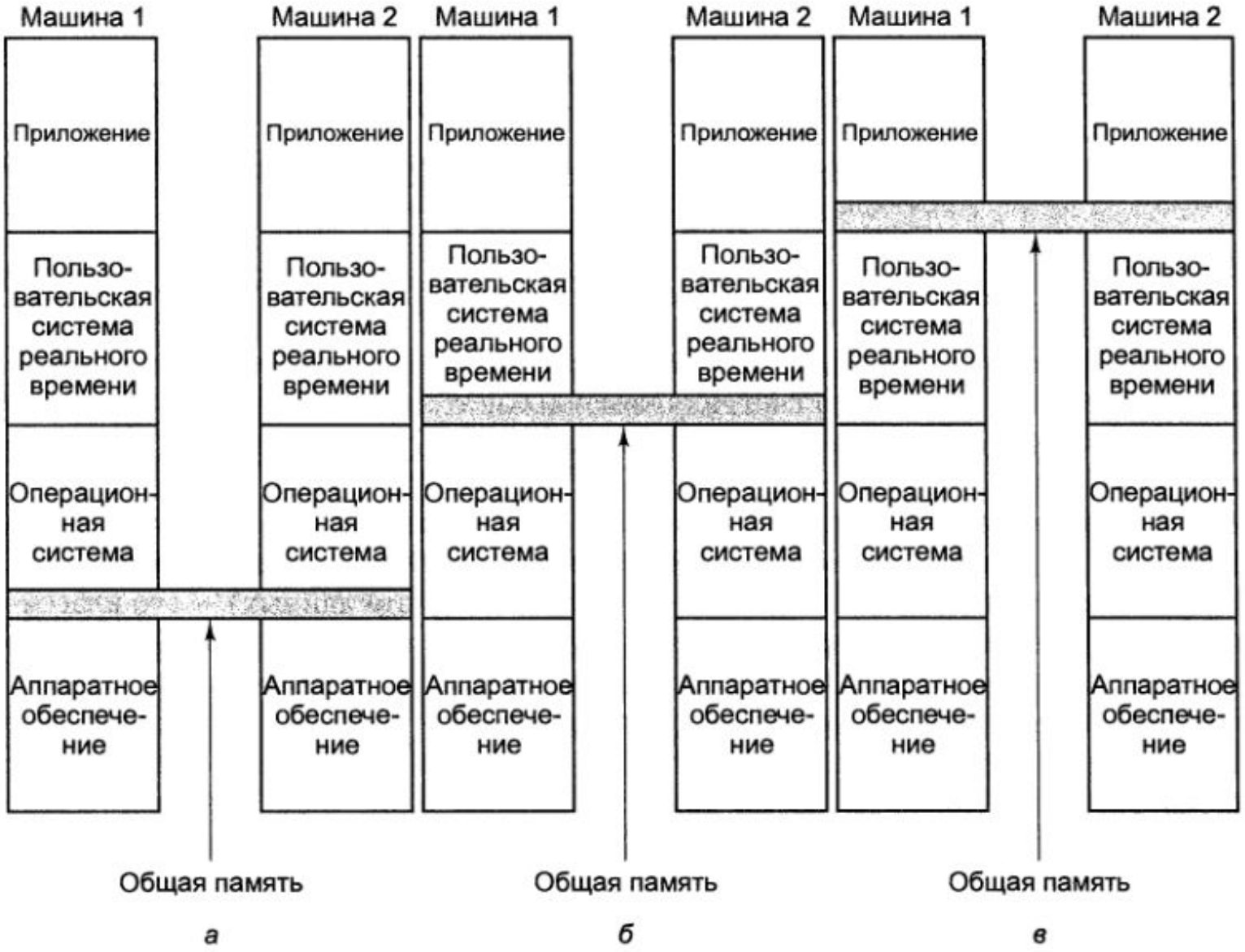
Дилемма

- Мультипроцессоры сложно разрабатывать, но легко программировать, а мультикомпьютеры легко строить, но трудно программировать

- Гибридные системы
- Различные реализации совместной память можно реализовывать по-разному, причем каждый вариант будет иметь достоинства и недостатки
- Масштабируемость - система, которая будет продолжать исправно работать при добавлении все новых и новых процессоров

Уровни реализации общей памяти

- Компьютерные системы не монолитны, а имеют многоуровневую структуру
- а) общая память, реализованная аппаратно, как в «настоящем» мультипроцессоре
- б) используется аппаратное обеспечение мультикомпьютера и операционная систем, которая моделирует общую память, предоставляя единое виртуальное адресное пространство - распределенная общая память (Distributed Shared Memory,DSM)
- в) реализация общей память программно пользовательской системой реального времени. При таком подходе абстракцию общей памяти создает язык программирования, и эта абстракция реализуется компилятором (модель общей памяти может зависеть от используемого языка программирования).



- Уровни, на которых можно реализовать общую память: аппаратная реализация (а); ОС (б); программная реализация (в)

Классификация параллельных компьютерных систем

- В основе классификации лежат понятия потоков команд и потоков данных. Поток команд соответствует счетчику команд. Система с ***n*** процессорами имеет ***n*** счетчиков команд и, следовательно, ***n*** потоков команд. Поток данных состоит из набора операндов.

Потоки команд	Потоки данных	Категория	Примеры
1	1	SISD	Классическая машина фон-Неймана
1	Много	SIMD	Векторный суперкомпьютер, матричный процессор
Много	1	MISD	Не существует
Много	Много	MIMD	Мультипроцессор, мультикомпьютер

SISD (Single Instruction stream Single Data stream)

- SISD - один поток команд с одним потоком данных) — это классическая последовательная компьютерная архитектура фон Неймана
- Компьютер фон Неймана имеет один поток команд и один поток данных и в каждый момент времени может выполнять только одно действие

SIMD (Single Instruction-stream Multiple Data-stream)

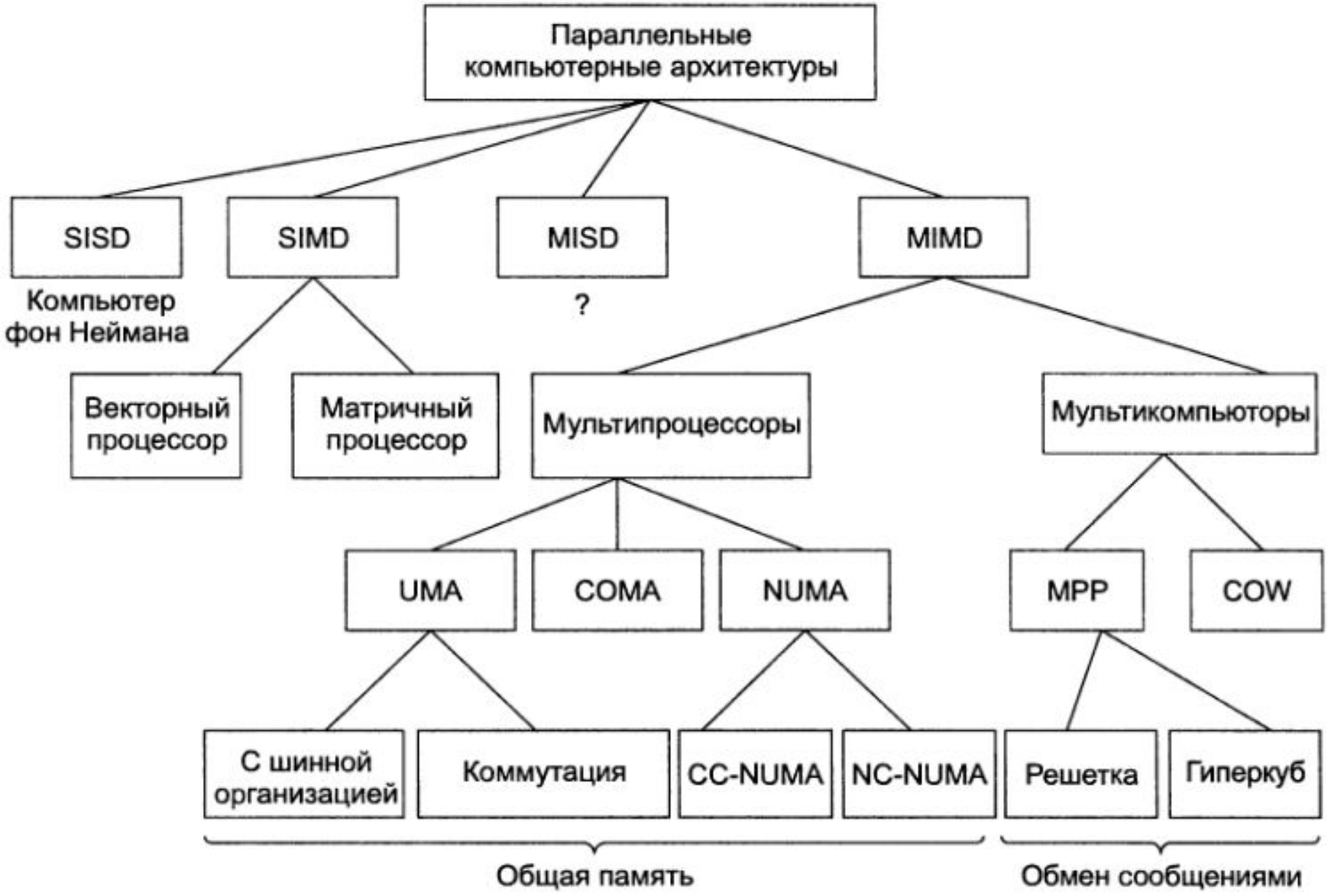
- SIMD - один поток команд с несколькими потоками данных, имеется один блок управления, выдающий по одной команде, но при этом есть несколько АЛУ, которые могут обрабатывать несколько наборов данных одновременно
- SSE-команды в процессорах Pentium относятся к категории SIMD-команд
- Потоковые процессоры специально разработаны для обработки мультимедийных данных используют идеи SIMD

MISD (Multiple Instruction-stream Single Data-stream)

- MISD— несколько потоков команд с одним потоком данных) - несколько команд оперируют одним набором данных
- К категории MISD относят машины с конвейерами

MIMD (Multiple Instruction-stream Multiple Data- stream)

- MIMD— несколько потоков команд с несколькими потоками данных - несколько независимых процессоров работают как часть большой системы
- В эту категорию попадают большинство параллельных процессоров
- И мультипроцессоры, и мультикомпьютеры — это MIMD-машины


Параллельные компьютерные архитектуры
SISD
SIMD
MISD
MIMD
Компьютер фон Неймана
?
Векторный процессор
Матричный процессор
Мультипроцессоры
Мультикомпьюторы
UMA
COMA
NUMA
MPP
COW
С шинной организацией
Коммутация
CC-NUMA
NC-NUMA
Решетка
Гиперкуб
Общая память
Обмен сообщениями

Семантика памяти

- Семантику памяти можно рассматривать как контракт между программным и аппаратным обеспечением памяти. Если программное обеспечение соглашается следовать определенным правилам, то память соглашается выдавать определенные результаты
- Основная проблема - сами правила, которые называются моделями состоятельности

Строгая состоятельность

- При любом считывании из адреса x всегда возвращается значение самой последней записи в x.
- Должен быть единственный модуль памяти, просто обслуживающий все запросы по мере их поступления (первым поступил — первым обработан), кэширование и дублирование данных не допускаются
- Значительное торможение работы памяти

Секвенциальная состоятельность

- В соответствии с этой моделью при наличии нескольких запросов на чтение и запись порядок обработки запросов определяется аппаратно, но при этом все процессоры воспринимают один и тот же порядок

Пример

- Предположим, процессор 1 записывает значение 100 в слово x, а через 1 нс процессор 2 записывает туда же значение 200. А теперь предположим, что через 1 нс после начала второй операции записи (процесс записи еще не закончен) два других процессора, 3 и 4, считывают слово x по два раза.

- Два процессора записывают, а другие два процессора считывают одно и то же слово из общей памяти

- Возможные варианты очередности событий

Вариант 1	Вариант 2	Вариант 3
Запись значения 100	Запись значения 100	Запись значения 200
Запись значения 200	Чтение значения 100 процессором 3	Чтение значения 200 процессором 4
Чтение значения 200 процессором 3	Запись значения 200	Запись значения 100
Чтение значения 200 процессором 3	Чтение значения 200 процессором 4	Чтение значения 100 процессором 3
Чтение значения 200 процессором 4	Чтение значения 200 процессором 3	Чтение значения 100 процессором 4
Чтение значения 200 процессором 4	Чтение значения 200 процессором 4	Чтение значения 100 процессором 3

- В первом варианте оба процессора получают значение 200 в каждой из двух операций считывания.
- Во втором варианте процессор 3 получает значения 100 и 200, а процессор 4 — оба раза но 200.
- В третьем варианте процессор 3 получает два раза по 100, а процессор 4 — значения 200 и 100.

- Правила секвенциальной состоятельности не выглядят столь «жестокими», как правила строгой состоятельности
- Даже если несколько событий совершаются одновременно, считается, что на самом деле они происходят в определенном порядке (который может выбираться произвольно), и все процессоры воспринимают именно этот порядок

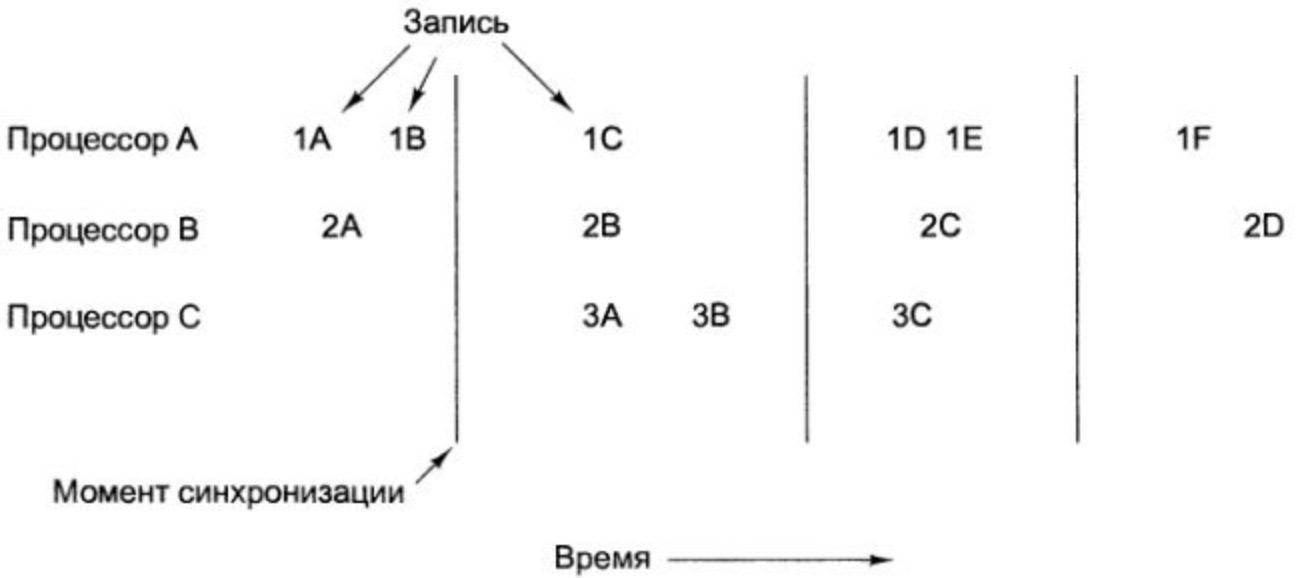
Процессорная состоятельность

- Не слишком строгая модель, но зато ее легче реализовать на больших мультипроцессорах
- Свойства:
  - 1. Все процессоры видят операции записи любого процессора в том порядке, в котором эти операции выполняются
  - 2. Все процессоры видят все операции записи в любое слово памяти в одном и том же порядке

- Два процессора (1 и 2) начинают три операции записи значений 1A, 1B, 1C и 2A, 2B, 2C одновременно
- Другие процессоры, которые заняты считыванием слов из памяти, увидят какую-либо последовательность из шести операций записи, например, 1A, 1B, 2A, 2B, 1C, 2C или 2A, 1A, 2B, 2C, 1B, 1C; и т. п.
- При процессорной состоятельности не гарантируется, что каждый процессор видит один и тот же порядок (в отличие от секвенциальной состоятельности)
- Гарантируется абсолютно точно — ни один процессор не увидит последовательность, в которой сначала выполняется операция 1B, а затем — 1A
- Порядок, в котором выполняются обращения одного и того же процессора, остается одинаковым для всех наблюдателей

Слабая состоятельность

- В модели слабой состоятельности не гарантируется, что операции записи, произведенные одним процессором, будут восприниматься другими в том же порядке
- Один процессор может увидеть сначала операцию 1A, а п



В слабо состоятельной памяти периодически выполняются операции синхронизации

Свободная состоятельность

- Используется нечто похожее на критические секции программы - если процесс выходит за пределы критической области, это не значит, что все записи должны немедленно завершиться. Требуется только, чтобы все записи были завершены до того, как какой-нибудь процесс снова войдет в эту критическую область
- Операция синхронизации разделяется на две разные операции: *acquire* и *release*

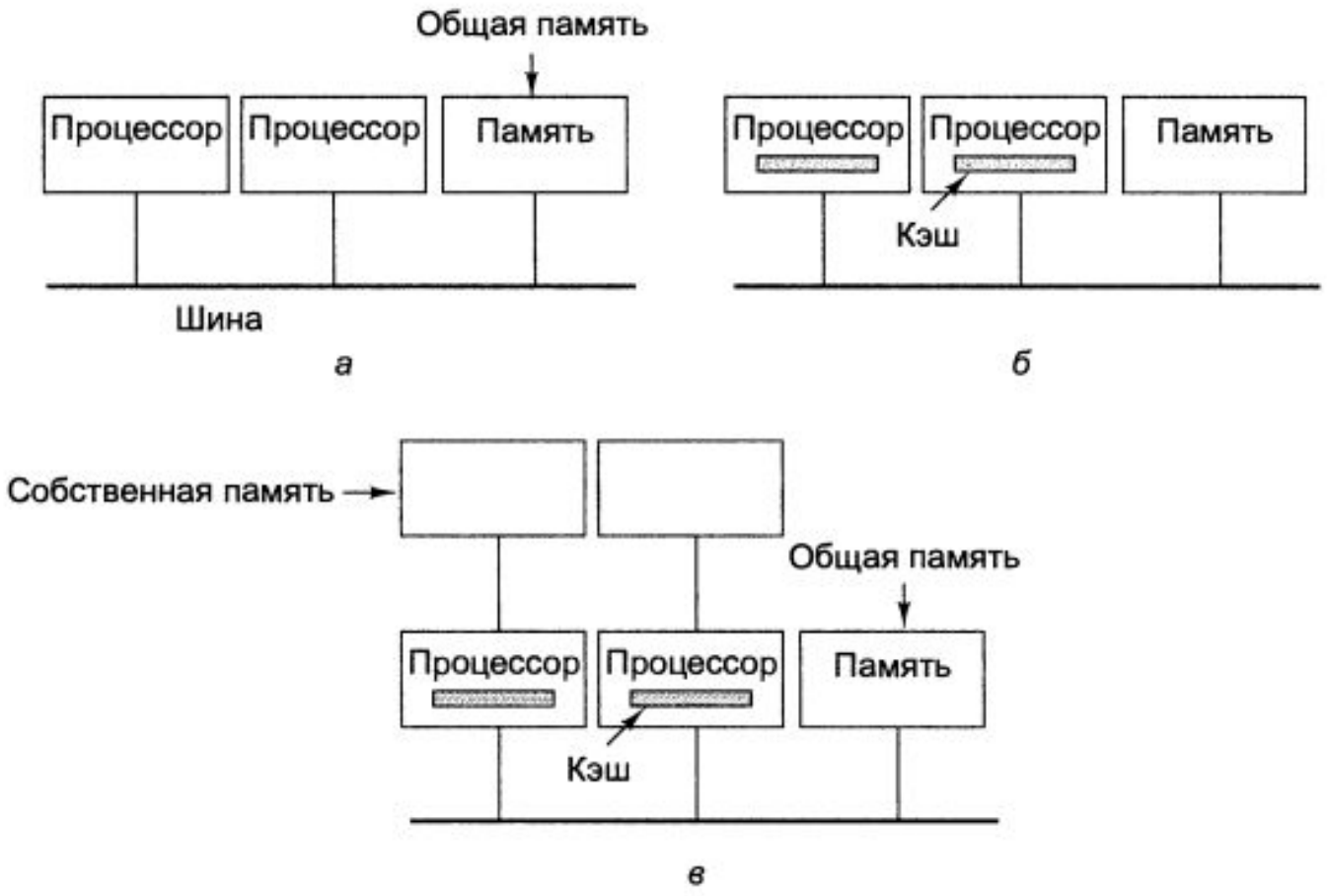
- Чтобы считать или записать совместно используемую переменную, процессор (то есть его программное обеспечение) сначала должен выполнить операцию *acquire* с переменной синхронизации, что позволит ему получить монопольный доступ к общим данным
- Далее процессор может делать с этими данными все, что ему требуется (считывать или записывать), а по завершении он должен выполнить операцию *release* с переменной синхронизации, чтобы показать, что он завершил работу
- Операция *release* не требует завершения незаконченных записей, но сама она не может завершиться, пока не закончатся все ранее начатые операции записи. Более того, новые операции с памятью могут начинаться сразу же

- Когда начинается следующая операция *acquire*, производится проверка, все ли предыдущие операции *release* завершены
- Если нет, то операция *acquire* задерживается до тех пор, пока это не будет сделано (а перед тем, как завершатся все операции *release*, должны быть завершены все операции записи)
- Если следующая операция *acquire* выполняется через достаточно длительный промежуток времени после последней операции *release*, ей не нужно ждать, и она может войти в критическую область без задержки
- Если операция *acquire* выполняется через небольшой промежуток времени после операции *release*, она (и все команды, которые должны выполняться следом) ожидает завершения всех операций *release*
- Это гарантирует, что все переменные в критической области будут обновлены

UMA-мультипроцессоры в симметричных мультипроцессорных архитектурах

- UMA (Uniform Memory Access) — однородный доступ к памяти
- В UMA-машинах каждый процессор имеет одно и то же время доступа к любому модулю памяти - каждое слово может быть считано из памяти с той же скоростью, что и любое другое слово. Если это технически невозможно, самые быстрые обращения замедляются, чтобы соответствовать самым медленным. Это и значит «однородный» доступ
- Такая однородность делает производительность предсказуемой
- Количество процессоров в UMA-мультипроцессорах обычно ограничивается несколькими десятками
- *Пример: IBM eServer, Sun StarFire, HP Superdome, SGI Origin*

Варианты мультипроцессора на одной шине:



- без кэш-памяти (а);
- с кэш-памятью (б);
- с кэш-памятью и отдельными модулями локальной памяти (в)

Согласованность кэш-памяти

- Проблема согласованности кэшей
- Протоколы согласования кэшей
- Следящий кэш- контроллер кэш-памяти, мониторит запросы, идущие по шине от других процессоров
- Сквозная запись

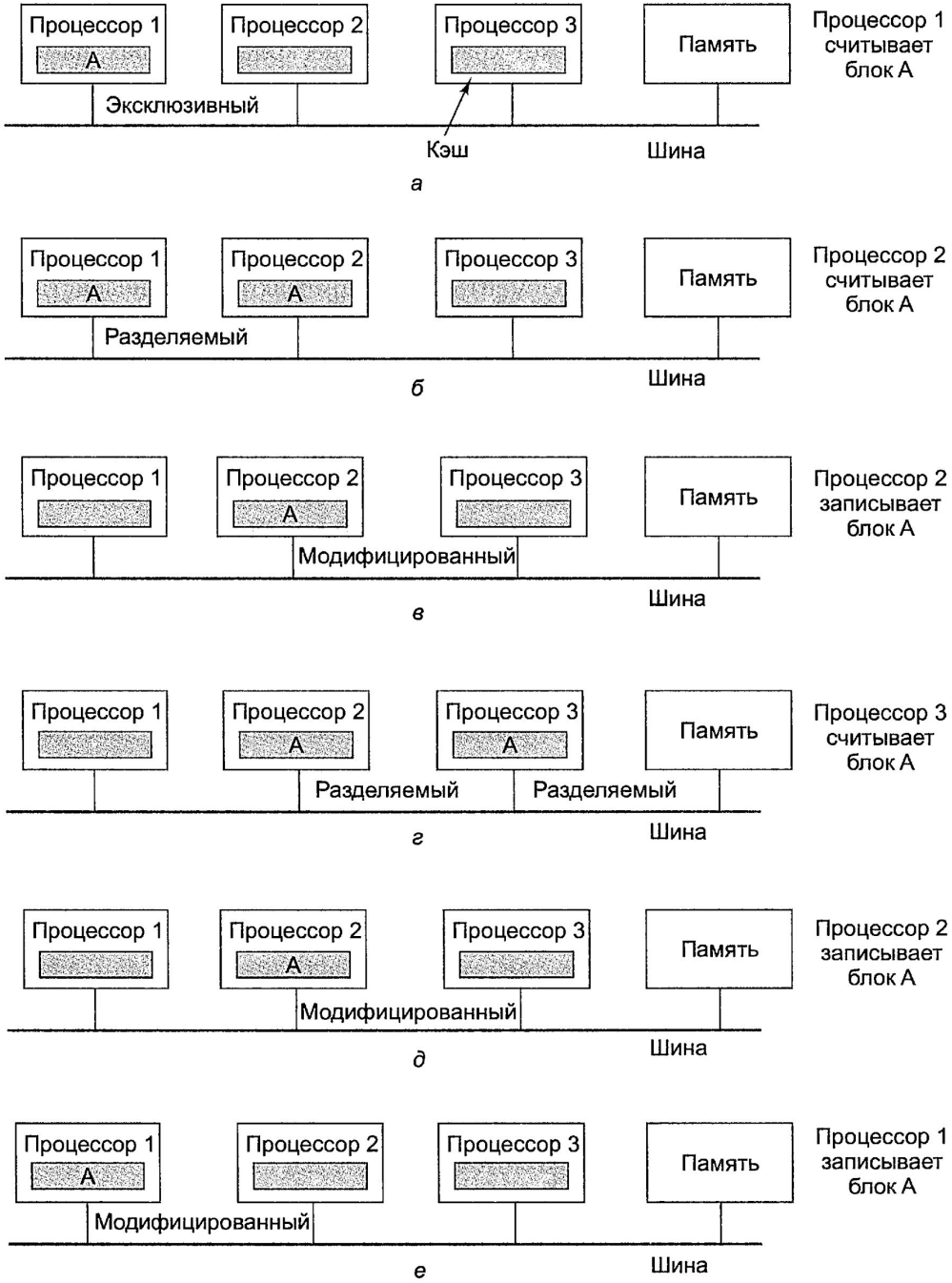
Сквозная запись

Действие	Локальный запрос	Удаленный запрос
Кэш-промах чтения	Вызов данных из памяти	
Кэш-попадание чтения	Использование данных из локального кэша	
Кэш-промах записи	Обновление данных в памяти	
Кэш-попадание записи	Обновление кэша и памяти	Объявление элемента в кэше недействительным

- Стратегия обновления
- Стратегия объявления данных недействительными

Протокол отложенной записи

- MESI (Invalid, Shared, Exclusive, Modified — недействительный, разделяемый, эксклюзивный, модифицированный)
- недействительный — элемент кэша содержит недействительные данные
- разделяемый — элемент может храниться в нескольких кэшах, память обновлена
- эксклюзивный — элемент находится только в данном кэше (ни в каких других кэшах его нет), память обновлена
- модифицированный — элемент действителен, основная память недействительна, копий элемента не существует


Процессор 1
A
Процессор 2
Процессор 3
Память
Процессор 1 считывает блок A
Эксклюзивный
Кэш
Шина
а
Процессор 1
A
Процессор 2
A
Процессор 3
Память
Процессор 2 считывает блок A
Разделяемый
Шина
б
Процессор 1
Процессор 2
A
Процессор 3
Память
Процессор 2 записывает блок A
Модифицированный
Шина
в
Процессор 1
Процессор 2
A
Процессор 3
A
Память
Процессор 3 считывает блок A
Разделяемый
Разделяемый
Шина
г
Процессор 1
Процессор 2
A
Процессор 3
Память
Процессор 2 записывает блок A
Модифицированный
Шина
д
Процессор 1
A
Процессор 2
Процессор 3
Память
Процессор 1 записывает блок A
Модифицированный
Шина
е

NUMA-мультипроцессоры

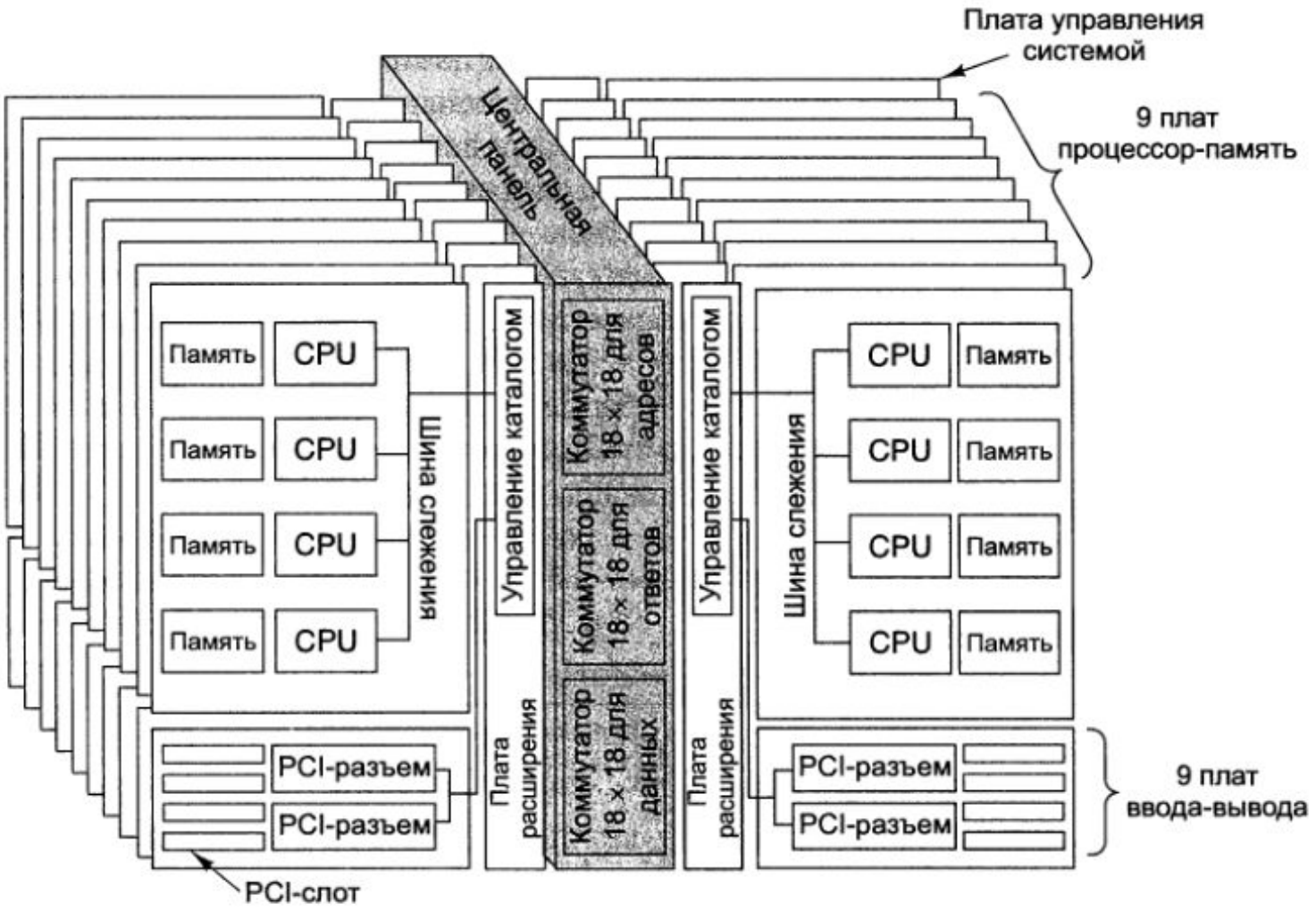
- NUMA (NonUniform Memory Access) - неоднородный доступ к памяти
- NUMA-машины имеют три ключевые характеристики, которые в совокупности отличают их от других мультипроцессоров:
  - существует единое адресное пространство, видимое всеми процессорами;
  - доступ к удаленной памяти производится командами LOAD и STORE;
  - доступ к удаленной памяти выполняется медленнее, чем доступ к локальной.

- Если время доступа к удаленной памяти не замаскировано кэшированием (кэш отсутствует), такая система называется NC-NUMA (No Caching NUMA — NUMA без кэширования).
- Если присутствуют согласованные кэши, то система называется CC-NUMA (Coherent Cache NUMA — NUMA с согласованными кэшами).
- Система аппаратной распределенной общей памятью, - аналогична распределенной общей памяти (DSM), реализованной программно, однако поддерживается аппаратно с использованием страниц маленького размера

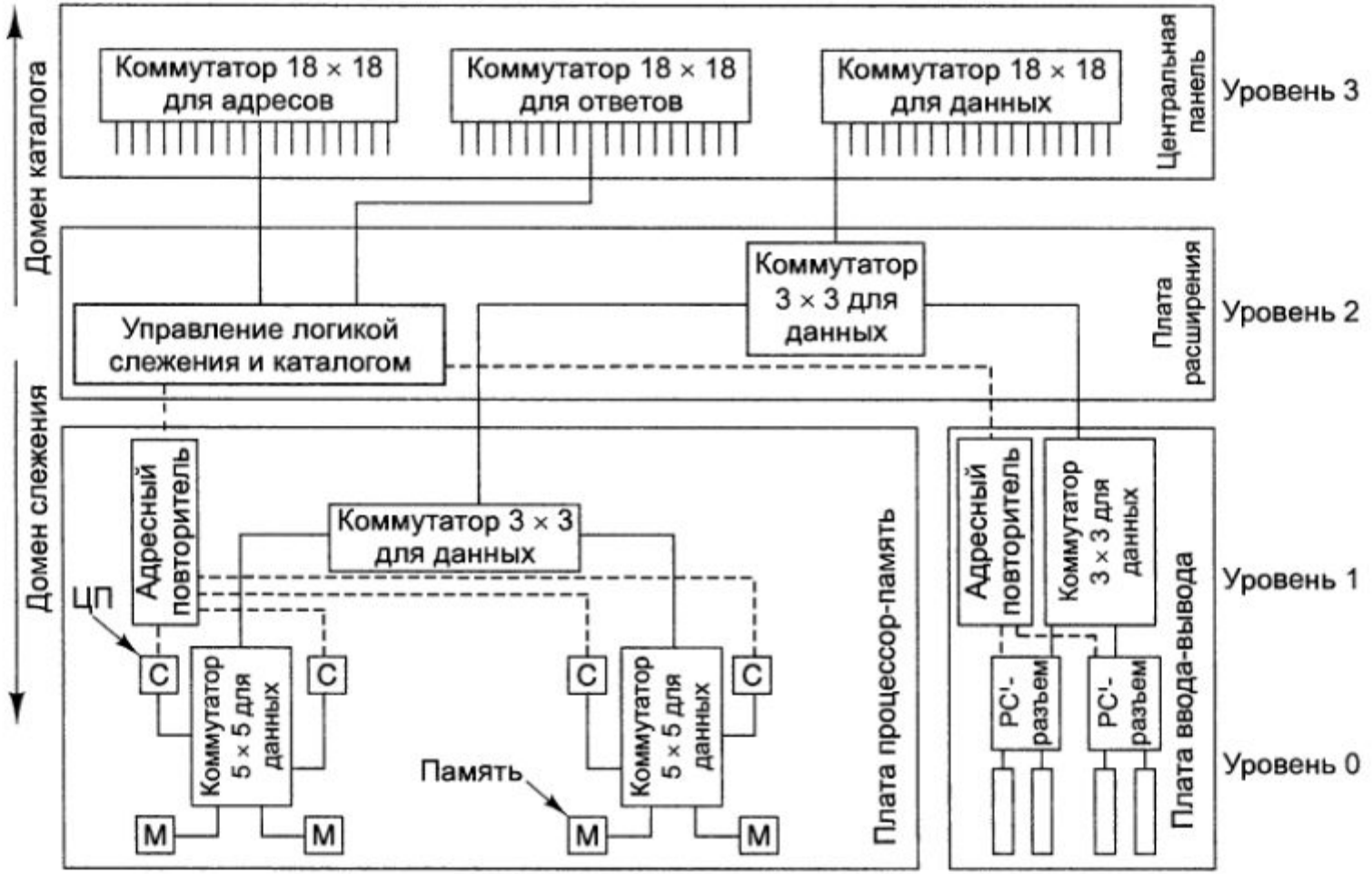
- *Пример: SGI Origin 2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000, Cray T3E*

NUMA-мультипроцессор Sun Fire E25K

- Система E25K содержит 18 наборов плат, каждый набор состоит из платы процессор-память, платы ввода-вывода с четырьмя PCI-слотами и платы расширения
- Плата расширения попарно объединяет платы процессор-память и ввода-вывода, связывая эти пары с центральной панелью, которая несет остальные платы и обеспечивает их коммутацию
- На каждой плате процессор-память находится 4 процессора и 4 модуля ОЗУ по 8 Гбайт
- Таким образом, на каждой плате процессор-память имеется 8 процессоров и 32-гигабайтное ОЗУ
- В целом в системе E25K имеются 144 процессора, 576 Гбайт памяти и 72 PCI-слота



- Мультипроцессор E25K компании Sun Microsystems



- Четырехуровневое соединение блоков в Sun Fire E25K.
- Пунктирные линии означают передачу адресов, сплошные — передачу данных

Общая память

- На самом нижнем уровне адресное пространство объемом 576 Гбайт разбивается на 2^29 блоков по 64 байта
- Это — неделимые элементы памяти. У каждого из них есть своя «родная» плата, где блок «живет», пока он не потребуется где-то еще
- Большинство блоков большую часть времени находятся на своих платах. Когда процессору требуется блок, будь то с собственной платы или с любой другой из 17 оставшихся, он сначала запрашивает копию в собственном кэше, после чего работает с кэшированной копией
- Хотя на каждой микросхеме в системе E25K находятся два процессора, у них общее адресное пространство, а, значит, и общий кэш блоков

- Каждый блок памяти (и каждая строка кэша всех микросхем) может находиться в одном из трех состояний:
  - эксклюзивный доступ (для записи);
  - совместный доступ (для чтения);
  - недействителен (то есть пуст).

- На уровне наборов плат логика слежения обеспечивает каждому процессору возможность сверять поступающие запросы со списком блоков в его локальном кэше
- Когда процессор обращается к слову памяти, он сначала преобразует виртуальный адрес в физический и проверяет, есть ли нужный блок в кэше
- Если нужный блок обнаруживается в собственном кэше, затребованное слово возвращается. В противном случае логика слежения проверяет, есть ли нужный блок в пределах того же набора плат. Если есть, то запрос выполняется. Иначе запрос посылается через схему перекрестной коммутации адресных линий.
- Логика слежения способна обслуживать по одному запросу за такт - тактовая частота системы составляет 150 МГц, можно обработать 150 миллионов запросов в секунду, или 2,7 миллиардов запросов для всех 18

- За счет распределения нагрузки между разными устройствами на разных платах Sun Fire E25K может работать с очень высокой производительностью
- Центральная панель способна поддерживать девять одновременных обменов данными с девятью платами-отправителями и девятью платами-получателями
- Так как схема перекрестной коммутации данных имеет ширину 32 байта, за каждый такт может передаваться 288 байт данных
- На тактовой частоте 150 МГц это дает пиковую пропускную способность 40 Гбайт/сек, когда все обращения направлены удаленным платам

COMA-мультипроцессоры

- COMA (Cache Only Memory Access) — доступ только к кэш-памяти
- Использования основной памяти каждого процессора в качестве кэш-памяти
- Физическое адресное пространство делится на строки кэша, которые по запросу свободно перемещаются в системе. Блоки памяти не имеют собственных машин. У них, как у кочевников в некоторых странах третьего мира, дом там, где они оказались
- Использование основной памяти в качестве большого кэша увеличивает процент кэш-попаданий → производительность
- Пример: *Kendall Square Research (KSR) 1*

Вопросы?