



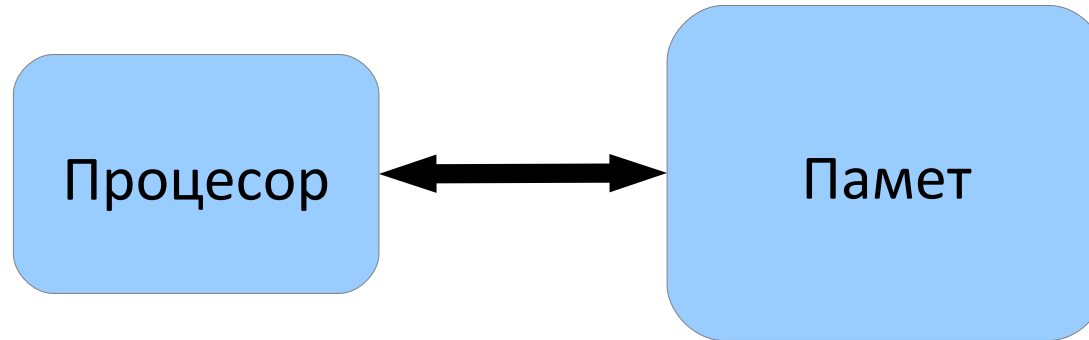
Паралелно Програмиране

Памет. Йерархия на паметта.
Архитектури на паметта на
паралелните компютри

доц. д-р Александър Пенев

Йерархия на Паметта

“Тясно гърло” при комуникацията



Производителността на високо-скоростните паралелни компютри обикновено е ограничена от *пропускателната способност и латентността*.

❖ Латентност

Времето за достъп до паметта >> Времето на цикъл на процесора

❖ Пропускателна способност

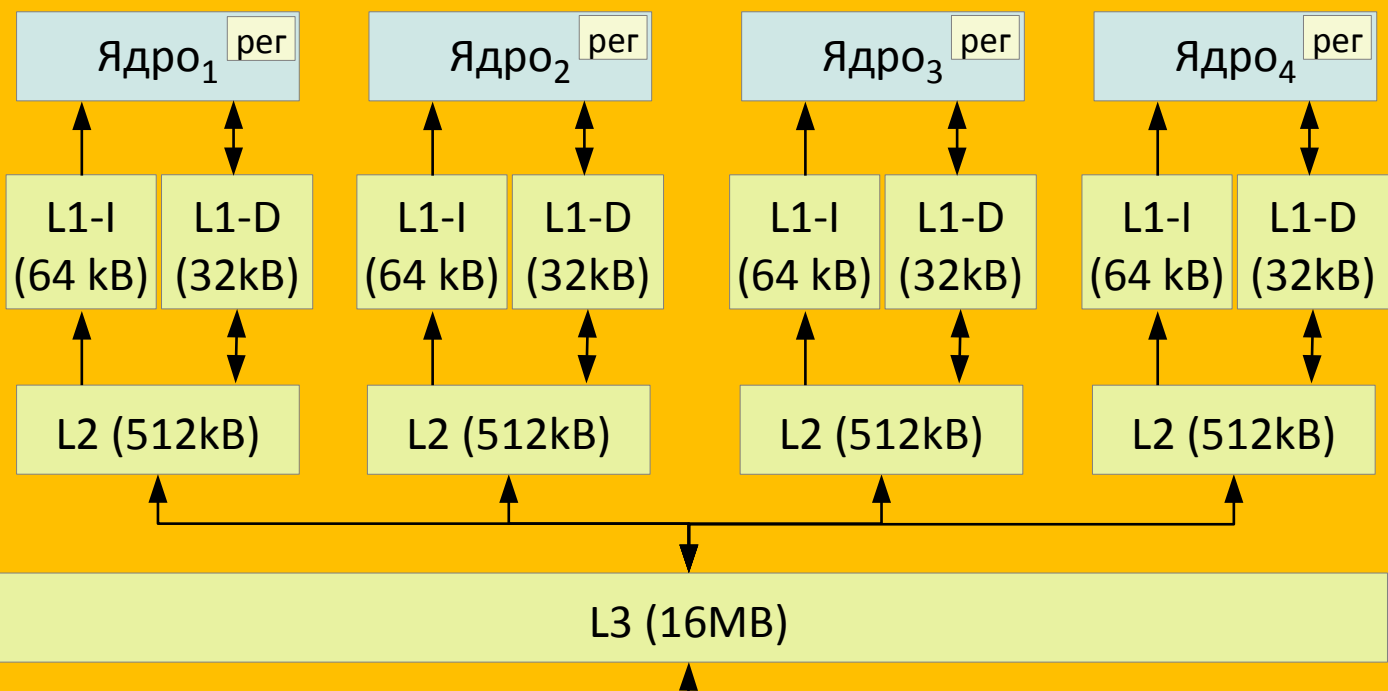
Максималният брой достъпи за единица време

Разбирането на работата на (достъпа до) паметта (и особено кеша) в съвременните (високо производителни и паралелни) системи е много важен фактор за разбирането и програмирането на ефективен паралелен софтуер.

Йерархия на паметта



Йерархия на паметта – Пример



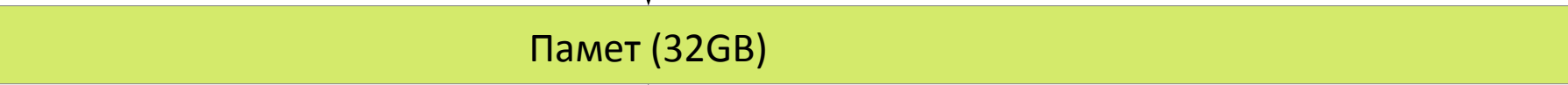
1 цикъл

3 цикъла

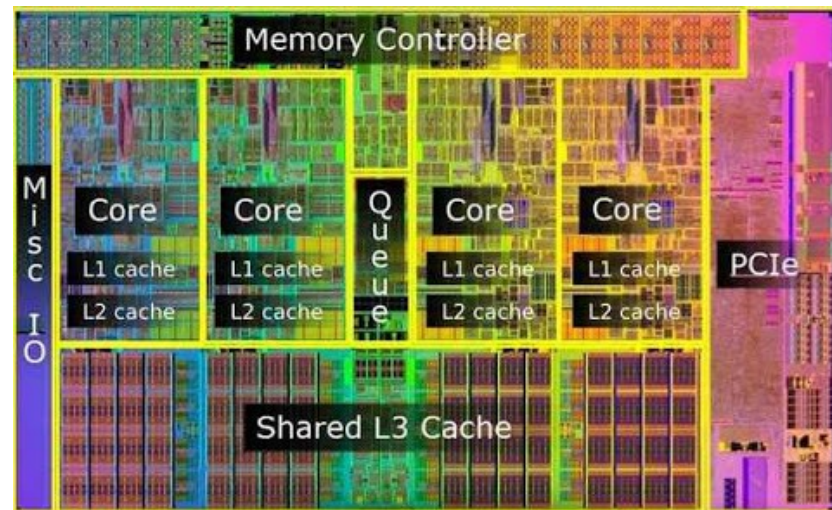
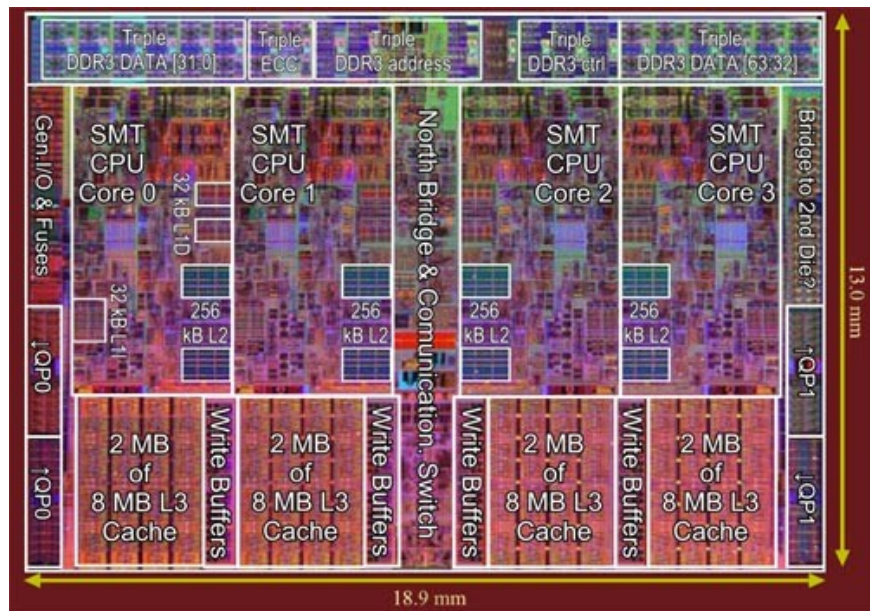
14 цикъла

100+ цикъла

>>100 цикъла



Йерархия на паметта – Пример



Памет в компютърната система

Основния проблем:

- ❖ Малък обем памет – бърз достъп;
- ❖ Голям обем памет – бавен достъп;
- ❖ Как програмата може да използва много памет и да има максимално бърз достъп до нея?

Йерархия на паметта

Задача на кеш системата:

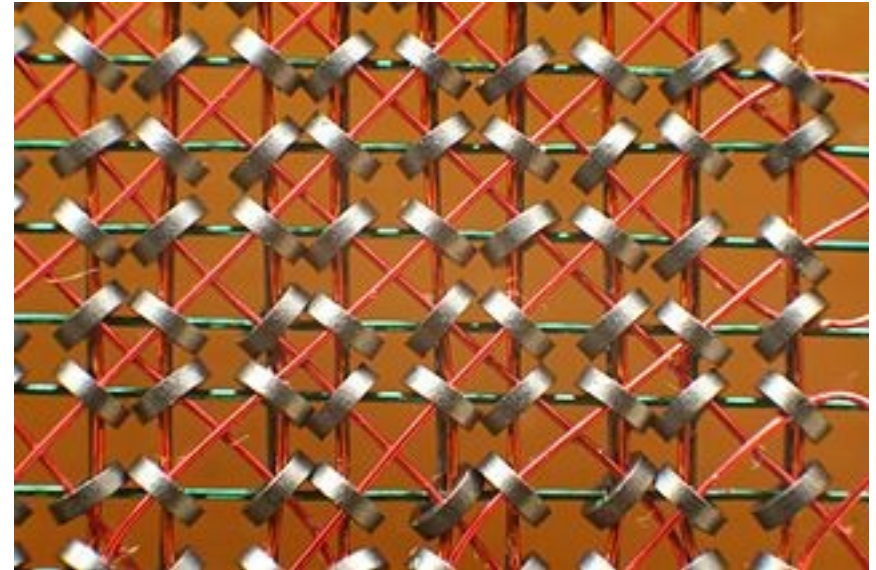
- ❖ Да съхранява най-често и най-вероятно използваните стойности в памет с много ниски времена на достъп;
- ❖ Хардуерни евристики определят какво и кога да бъде включено в кеш системата;

Какво става, ако начинът на достъп в програмата се различава от хардуерната евристика?

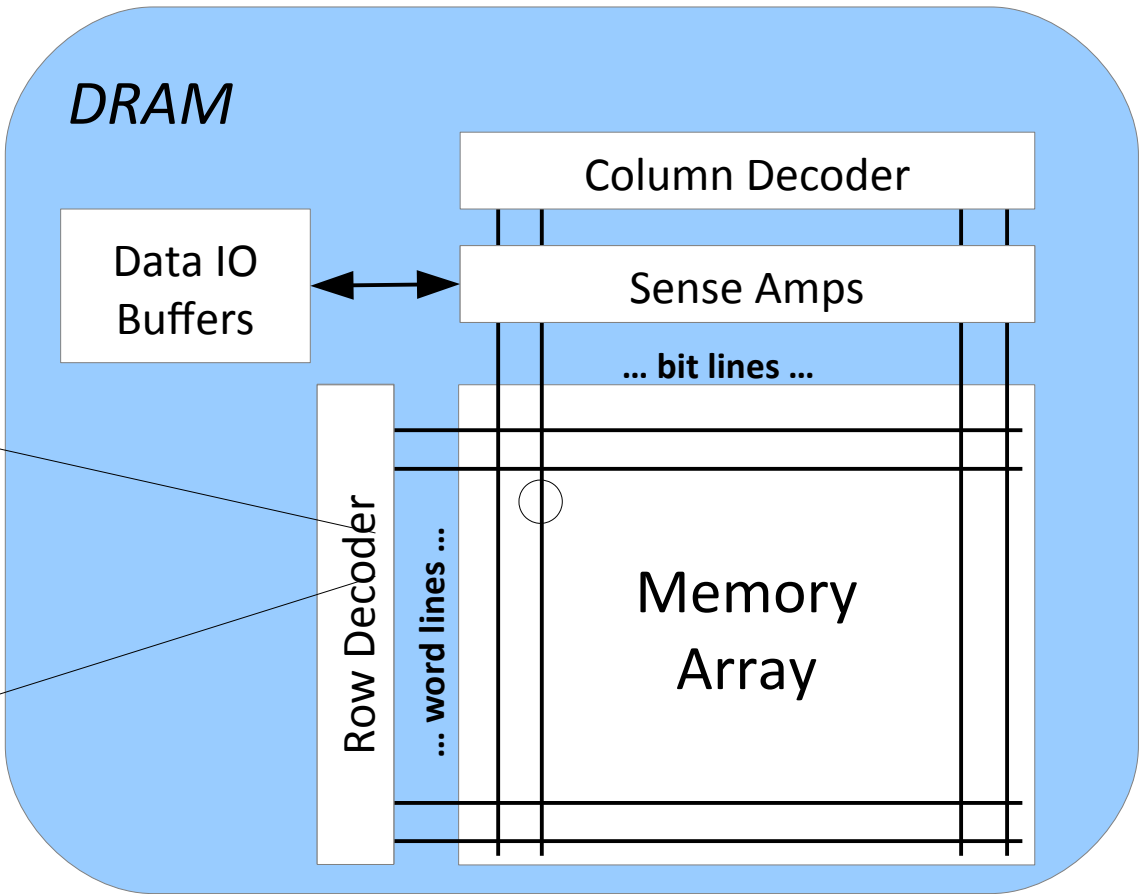
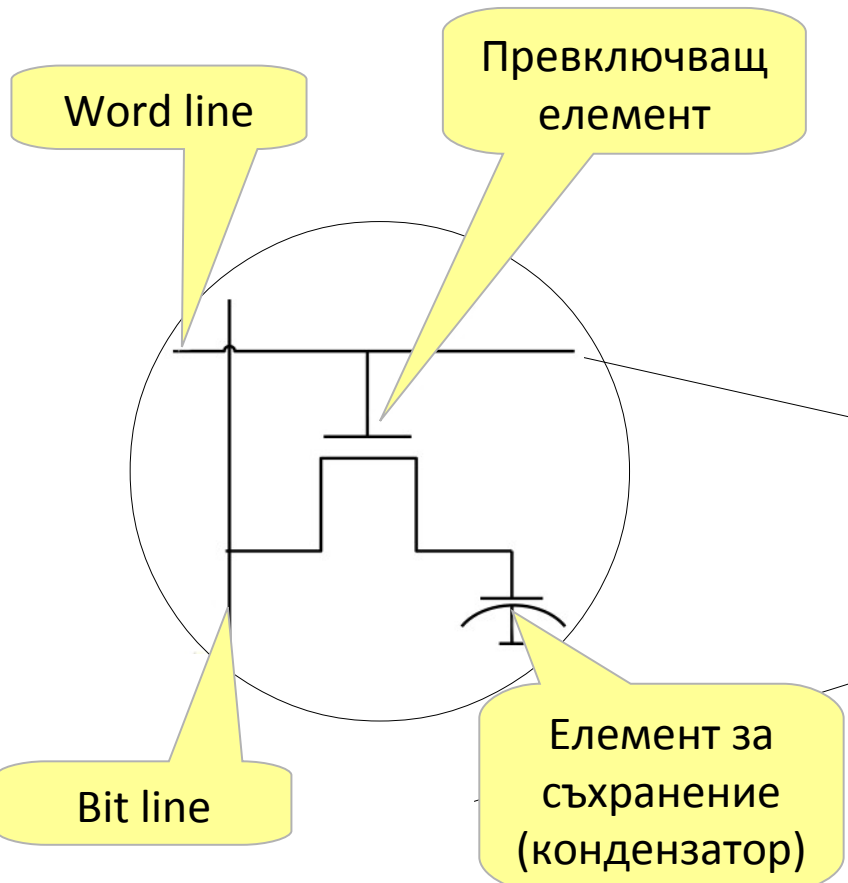
Памет

Core Memory

- ❖ Първата надеждна оперативна памет
- ❖ Предшественик на съвременната памет с произволен достъп (RAM)
- ❖ Битовете съхранени на магнетизирани ядра свързани в двумерна мрежа
- ❖ Използваше се до скоро в совалките на космическата програма на NASA

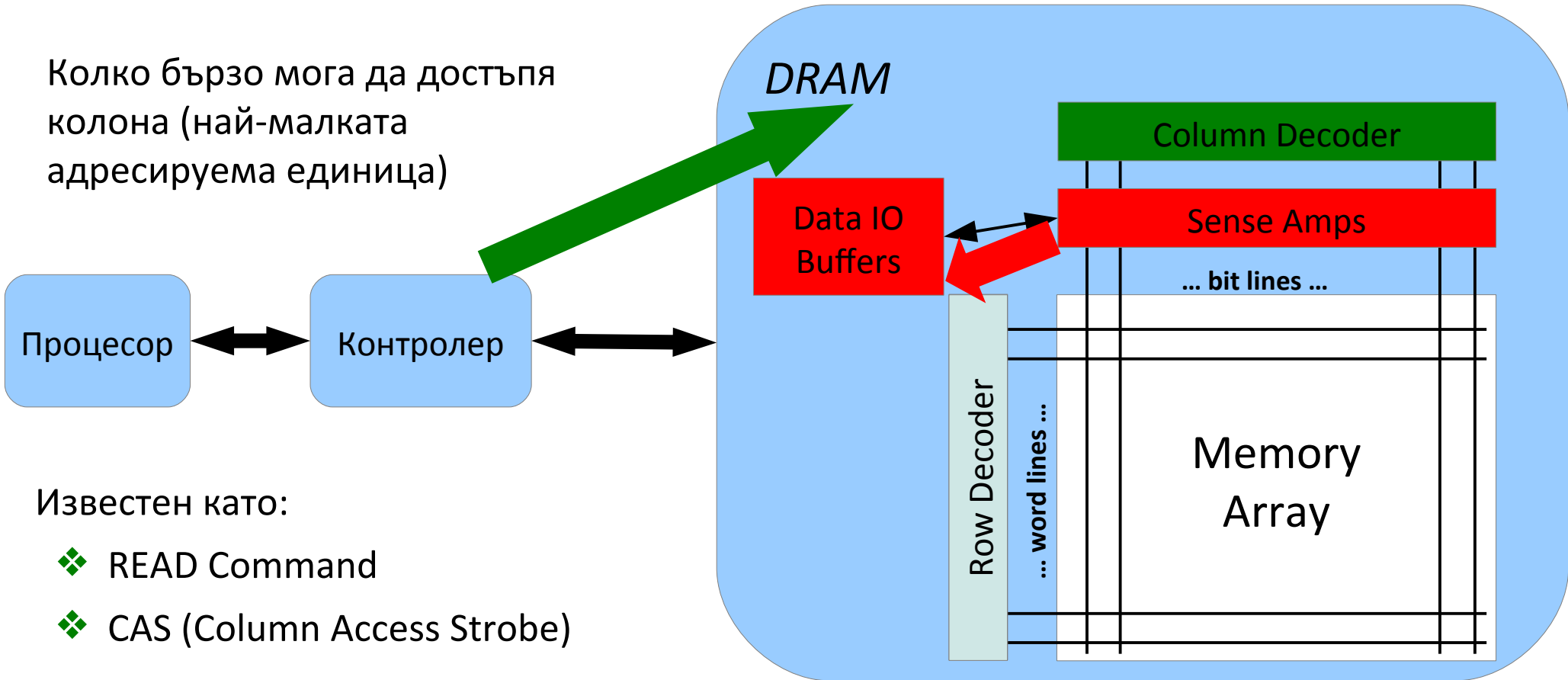


Памет от полупроводници. RAM



Протокол за достъп до DRAM

Колко бързо мога да достъпя колона (най-малката адресируема единица)

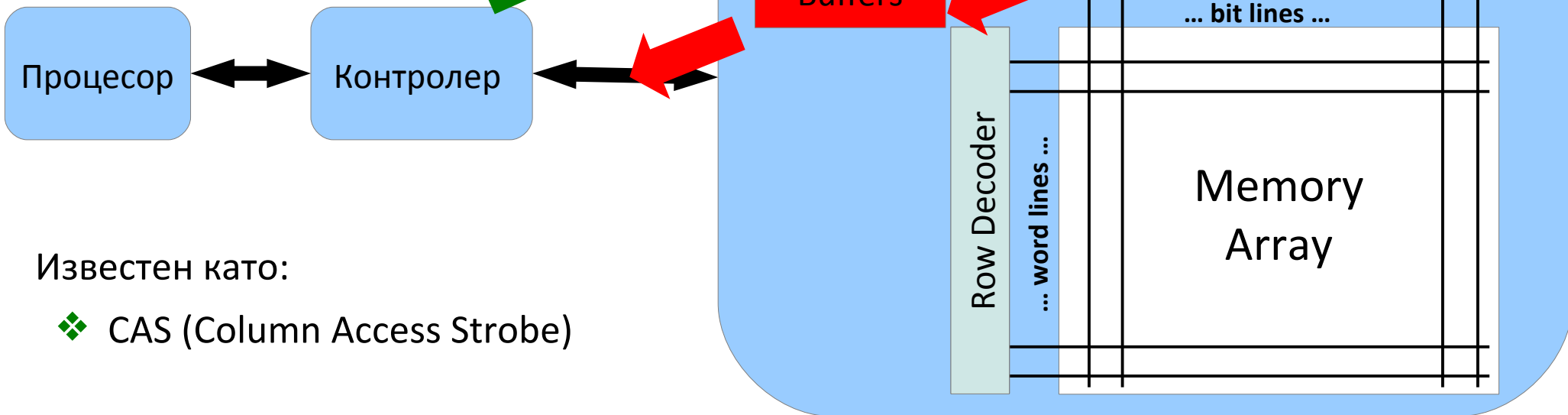


Известен като:

- ❖ READ Command
- ❖ CAS (Column Access Strobe)

Протокол за достъп до DRAM

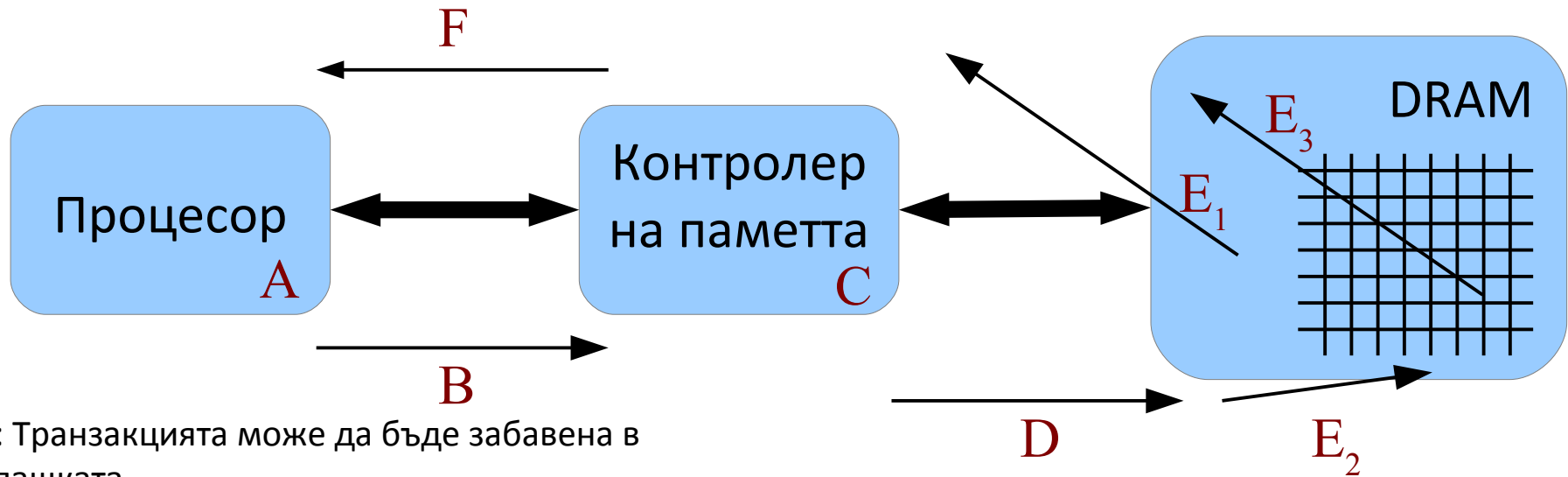
Колко бързо мога да достъпя колона (най-малката адресируема единица)



Известен като:

❖ CAS (Column Access Strobe)

Латентност при DRAM



- A: Транзакцията може да бъде забавена в опашката
- B: Транзакцията се изпраща до контролера
- C: Транзакцията се конвертира до последователност от команди (може да има опашка)
- D: Командите се изпращат до DRAM модула

- E₁: Има нужда от **CAS** или
- E₂: Има нужда от **RAS + CAS** или
- E₃: Има нужда от **PRE + RAS + CAS**
- F: Транзакцията се връща на процесора

$$\text{Латентност} = A + B + C + D + E + F$$

Действие на DRAM

Три стъпки при четене/запис от дадена банка (по около 20 ns)

❖ Достъп до ред (RAS)

- ❖ Декодира адреса на реда, активира адресирания ред (често множество kV на ред)
- ❖ Бит линиите споделят заряда с клетката за съхранение
- ❖ Малка промяна в напрежението засечена от усилвателя маркира (latch) целия ред от битове
- ❖ Усилвателят кара бит линиите да презаредят клетката

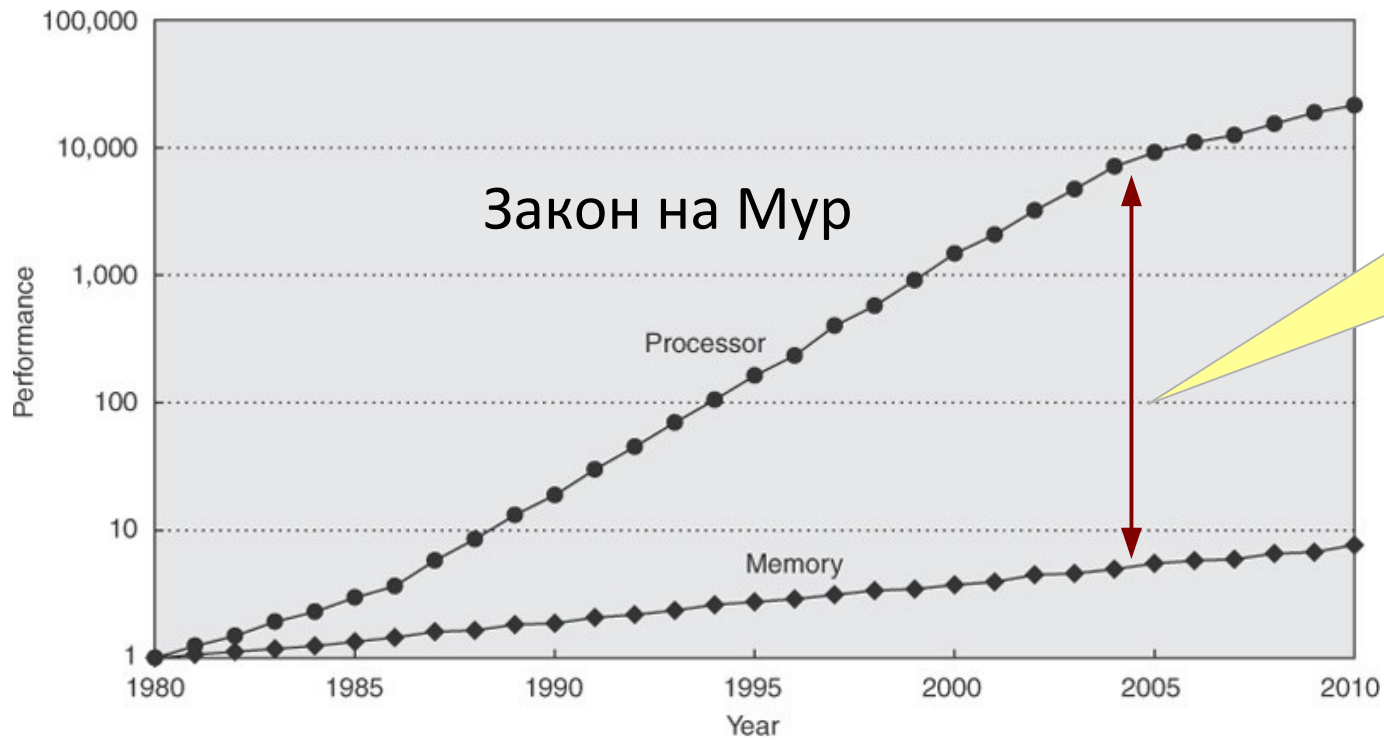
❖ Достъп до колона (CAS)

- ❖ Декодира се адреса на колоната, за да събере малък брой маркирани битове (4, 8, 16, 32, зависещ от DRAM пакета)
- ❖ При четене изпратените клетки се изпращат на изходните пинове
- ❖ При запис се презареждат маркираните битове с желаната стойност
- ❖ Може да има много достъпи до колони без друг достъп до ред (burst mode)

❖ Презареждане (Precharge)

- ❖ Зарежда бит линиите с междинна стойност преди следващ RAS

Производительность



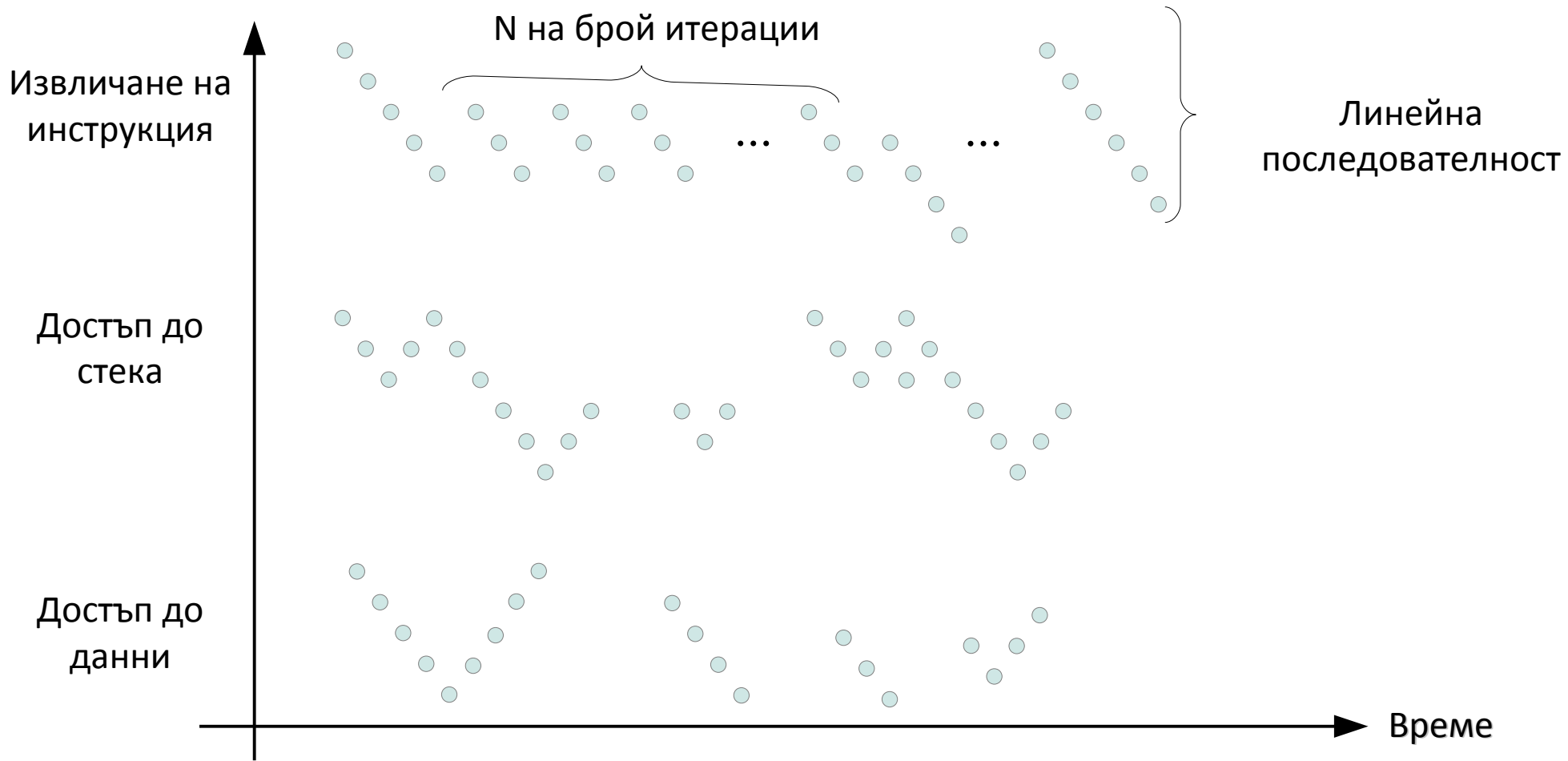
Нараства с
около 50%
годишно!

Памет на много нива

Идея: Прикриване на латентността чрез малки, бързи памети, наречени кешове

Кешовете са механизъм за прикриване на латентността, който се базира на емпиричното наблюдение, че начинът на достъп до паметта от процесора е добре предвидим.

Често срещани шаблони за достъп



Предвидими шаблони за достъп

Две предвидими свойства на достъпа до паметта:

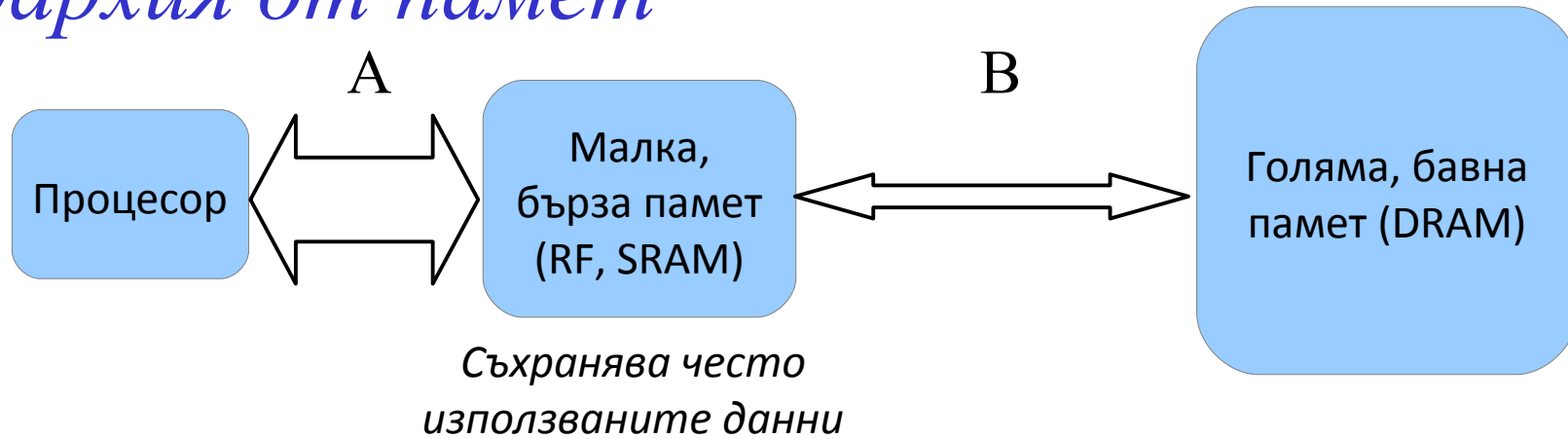
- ❖ **Времева локалност (Temporal Locality)**

Ако се достъпи даден адрес, то е много вероятно да се достъпи пак в близко бъдеще

- ❖ **Пространствена локалност (Spatial Locality)**

Ако се достъпи даден адрес, то е много вероятно околните адреси да се достъпват в близко бъдеще

Йерархия от памет



- ❖ Размер: Регистър \ll SRAM \ll DRAM;
- ❖ Латентност: Регистър \ll SRAM \ll DRAM;
- ❖ Пропускателна способност: на чипа \gg извън чипа;

При достъп до данни

- ❖ Попадение (hit) (данните са в бързата памет)
- ❖ Пропуск (miss) (данните не са в бързата памет)

Бързата памет е ефективна само когато проп. способност $B \ll A$



Управление на йерархията от памет

Малка бърза памет, например регистри

- ❖ Адресът се съдържа в инструкцията
- ❖ Реализирана като съвкупност от регистри (registry file)

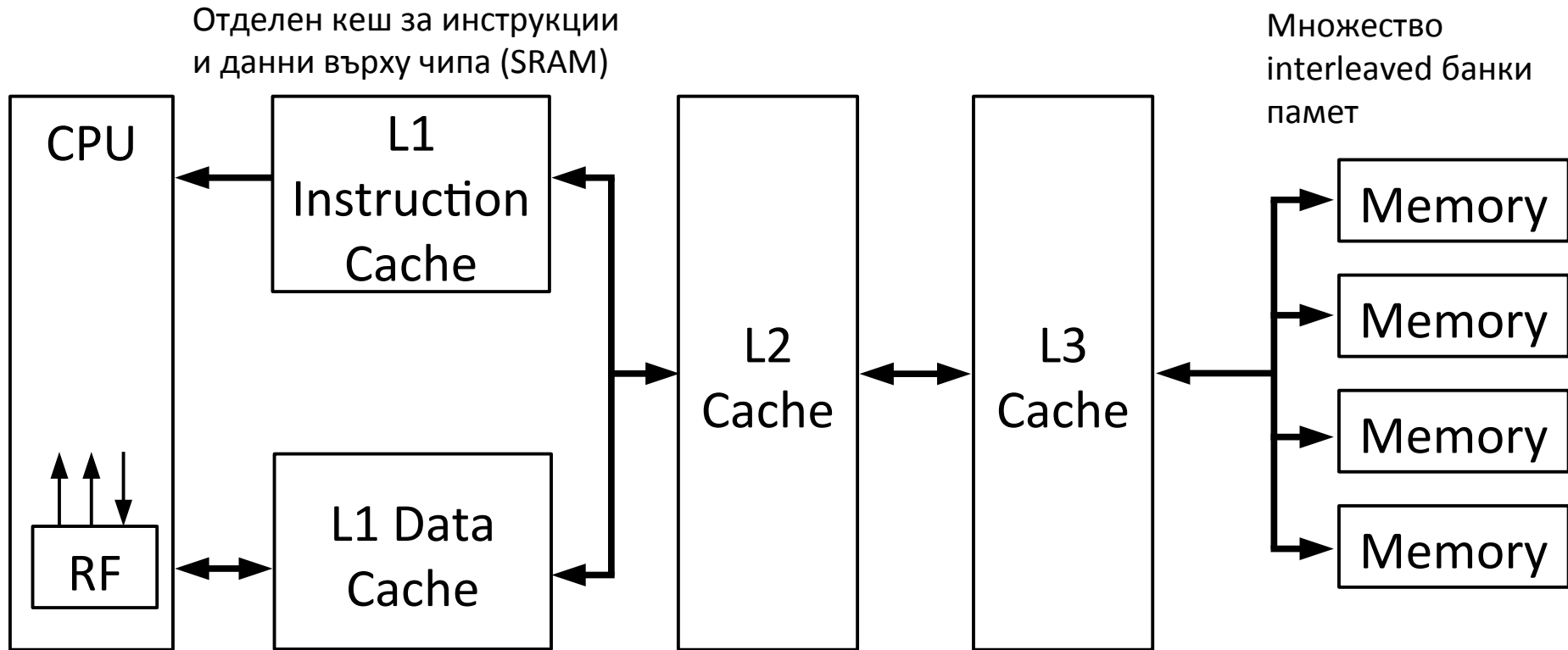
Хардуерът може да прави неща „зад гърба“ на софтуера, например управление на стека или преименуване на регистри

Голяма бавна памет

- ❖ Адресът се изчислява от стойностите в регистър
- ❖ Реализирана като йерархия от кешове

Хардуерът решава какво да съхрани в по-бързата памет. Софтуерът може да „подказва“, например не кеширай или извлечи предварително

Типична йерархия от памет (2012)



Набор много портови регистри

Голям кеш от второ и трето ниво близо до чипа



Терминология при йерархията от памет

❖ Попадение (hit)

Данните са в някой блок на бързата памет

❖ Процент попадения (hit rate)

Съотношението на достъпа до бързата памет и общия брой достъпи

❖ Време на попадение (hit time)

Времето, което отнема да се извлече заявката в бързата памет. Състои се от време за достъп до бързата памет и време за определяне дали е попадение или пропуск

❖ Пропуск (miss)

Данните трябва да се извлекат от по-бавната памет

❖ Процент пропуски (miss rate)

Процент пропуски = $1 - \text{процентът попадения}$

❖ Наказание при пропуск (miss penalty)

Времето, за което се замества блок в бързата памет с намерения блок и се доставя до проц

❖ Времето на попадение << Наказанието при пропуск

Проблеми при йерархията от памет

❖ Идентификация на блок

❖ Как се намира блокът, ако е в по-близката (по-бърза) памет?

Tag/Block

❖ Слагане на блок

❖ Къде може да се сложи блок в по-близката (по-бързата) памет?

Пълна асоциативност, Множествена асоциативност, Директно съпоставяне

❖ Заместване на блок

❖ Кой блок трябва да се замени при пропуск?

Произволен, LRU

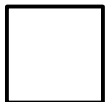
❖ Стратегия за запис

❖ Какво става при запис

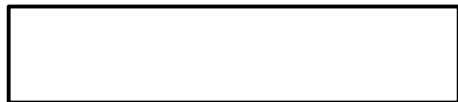
Write back, Write through (с Write buffer)

Кеш, използващ директно съпоставяне

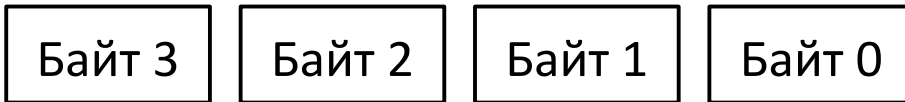
Бит за валидност



Кеш таг



Кеширани данни



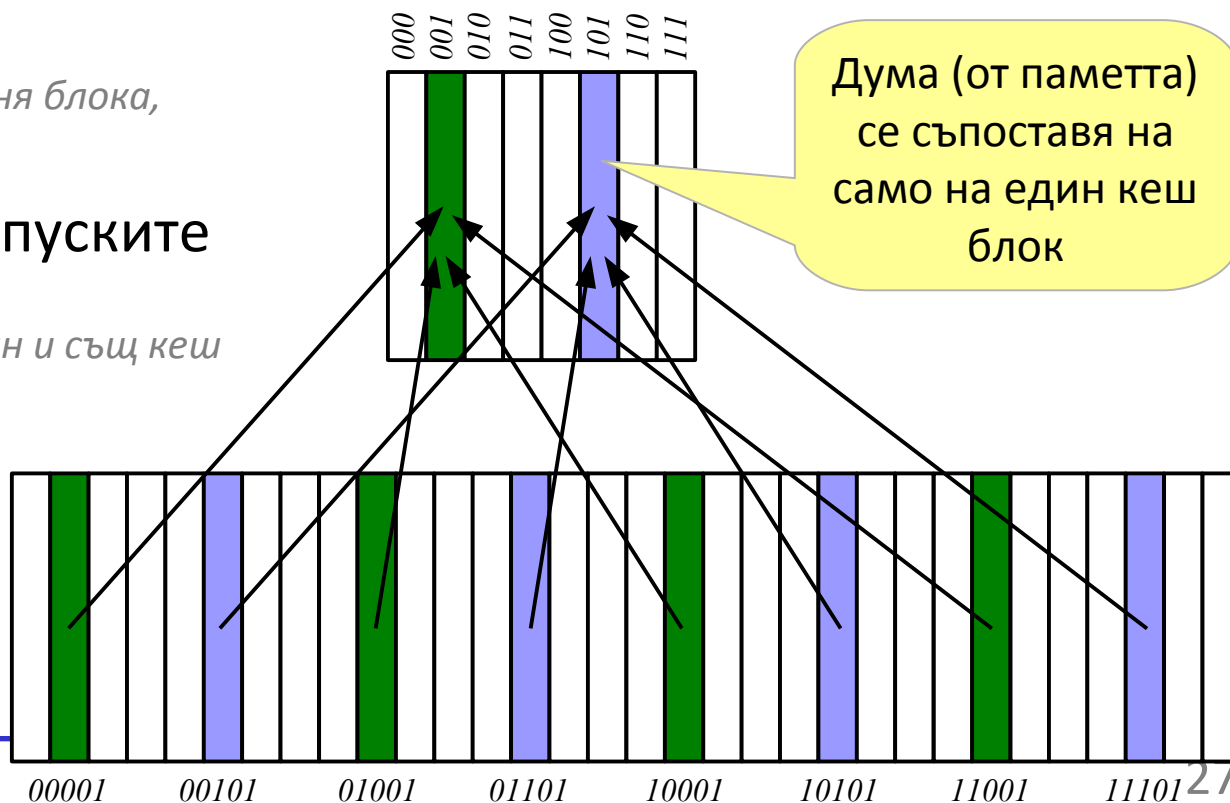
❖ Ефект Пинг Понг

Най-лошият случай е да се заменя блока, последван от пропуск

❖ За да се намалят пропуските

*Увеличаваме размера на кеша
Множество стойности за един и същ кеш индекс*

Адресът в кеша =
(Адресът на блока) mod
(Броят кеш блокове)



Достъп до кеша

Размерът на кеша зависи от

- ❖ Броя кеш блокове
- ❖ Броя битове в адреса
- ❖ Размера на думата

Например

- ❖ При n-битов адрес, 4-байтова дума и 1024 кеш блока
- ❖ Размерът на кеша = $1024[(n-10-2) + 1 + 32]$ бита

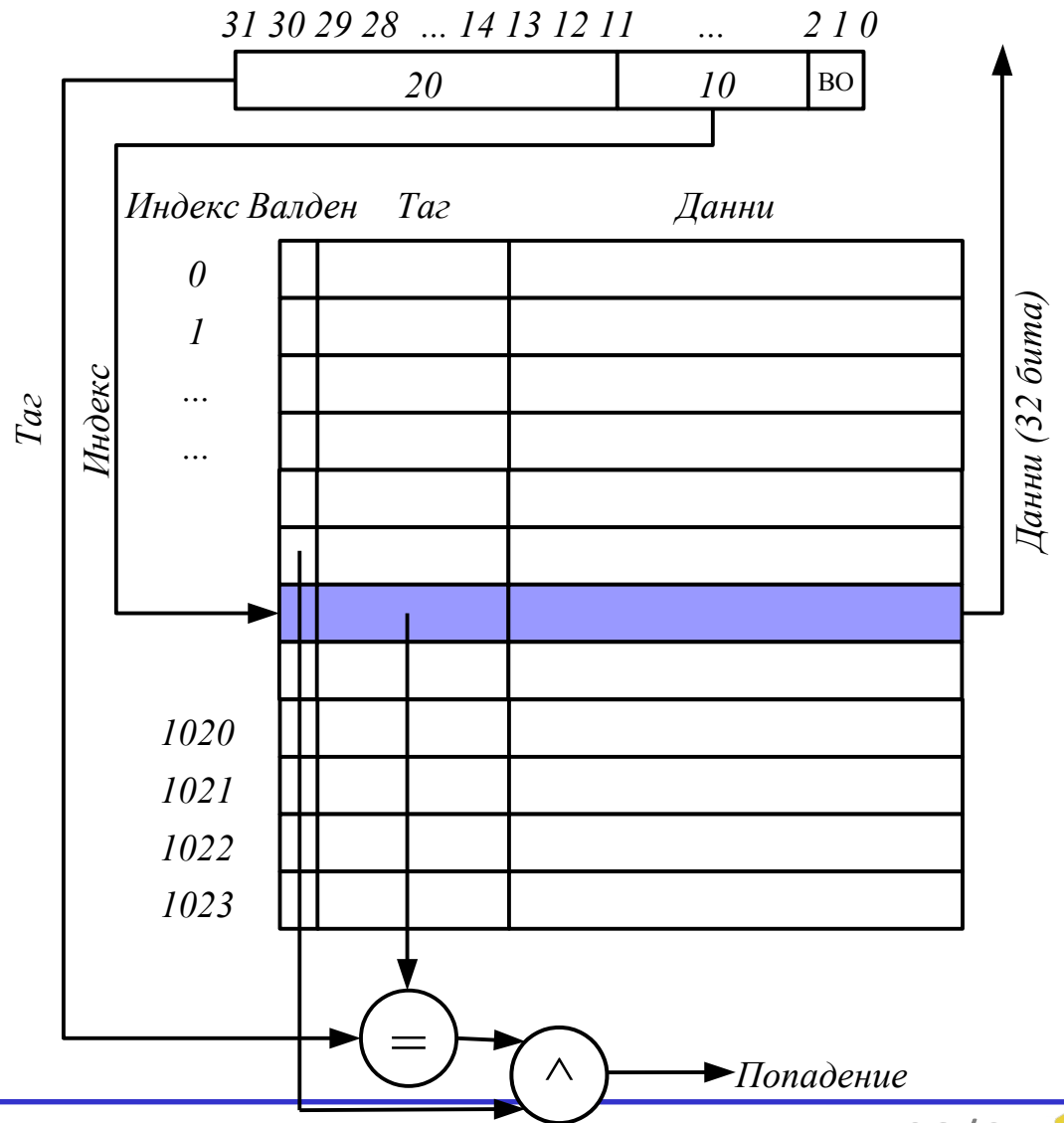
Брой кеш блокове

Таг

Размер на думата

Валидност

Памет

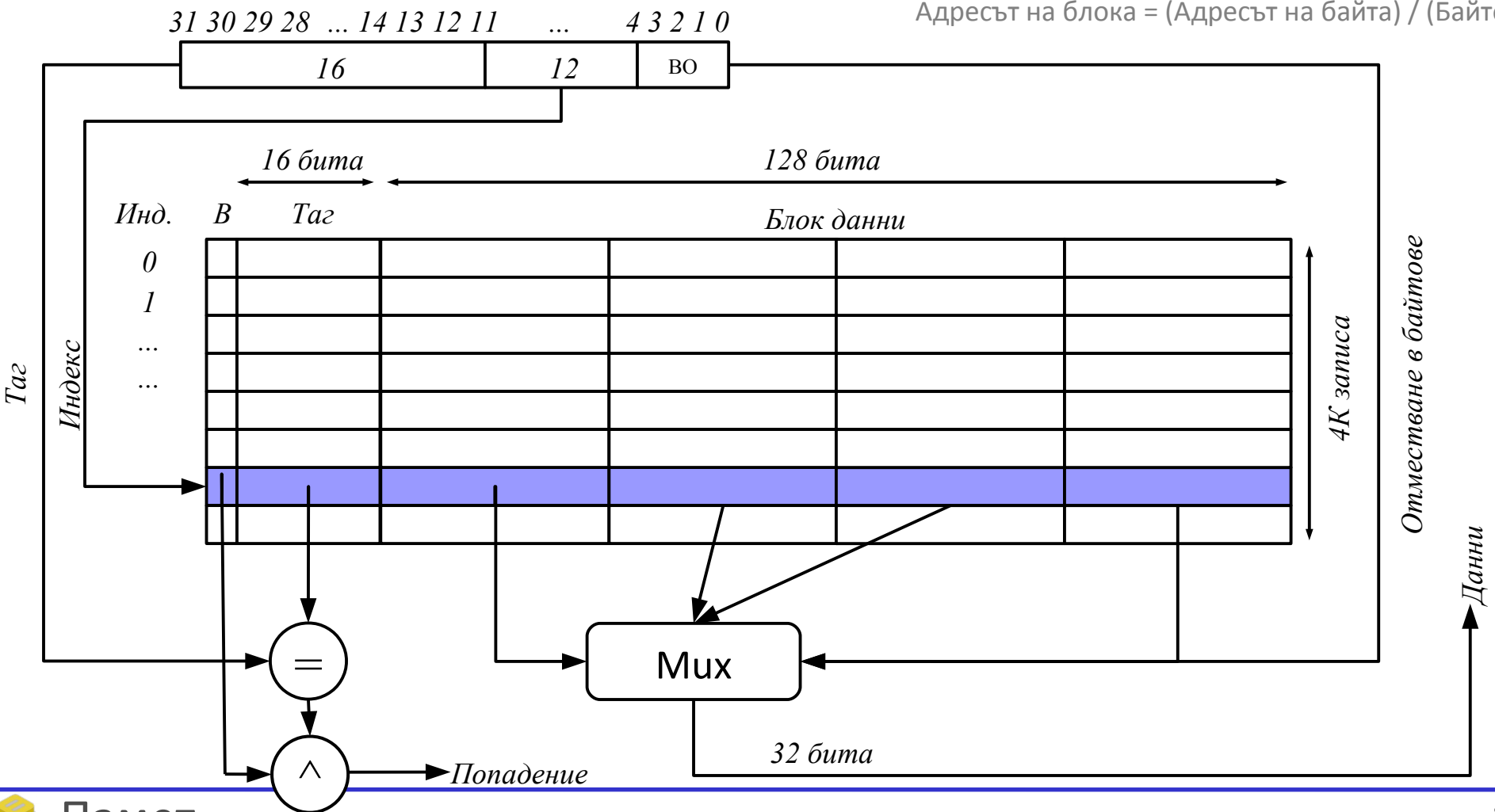


Достъп до кеша

Възползва се от пространствената локалност

Адресът в кеша = (Адресът на блока) mod (Броят кеш блокове)

Адресът на блока = (Адресът на байта) / (Байтове на блок)



Определяне на размера на блоковете

По-големи блокове данни се възползват по-добре от пространствената локалност, НО:

- ❖ По-големи блокове означава по-голямо наказание при пропуск

Повече време за запълване на блок

- ❖ Ако размерът на блока е прекалено голям в сравнение с размера на кеша, то процентът пропуски се повишава

Прекалено малко кеш блокове (блокове от данни в кеша)

Средното време за достъп =

Времето за попадение * (1 – Процентът пропуски) +

Наказанието за пропуск * Процентът пропуски

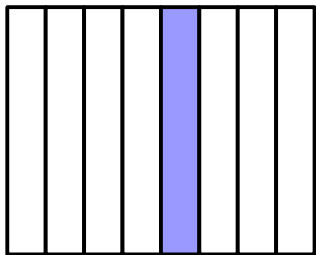
Местоположение на блоковете

Хардуерна сложност

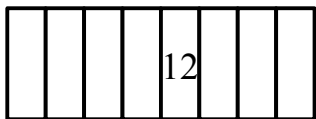
Оползотворяване на кеша

Кеш с директно съпоставяне

Номер блок: 0 1 2 3 4 5 6 7



Данни



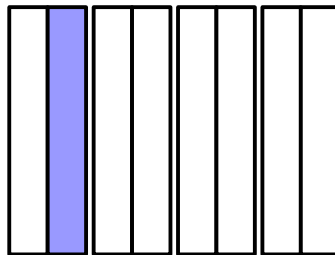
Търсене

Блок 12 може да бъде поставен:

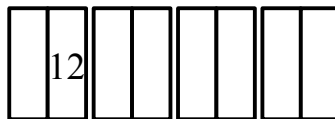
$$12 \bmod 8 = 4$$

Кеш с множествена асоциативност

Номер масив: 0 1 2 3



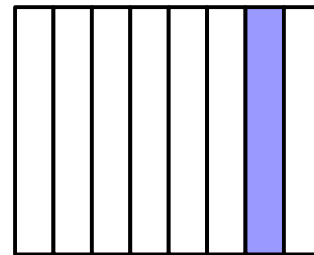
Данни



Търсене

$$12 \bmod 4 = 0$$

Кеш с пълна асоциативност



Данни



Търсене

- ❖ Номерът на масив = (Номерът на блок) mod (Броя на масиви в кеша)
- ❖ Повишената гъвкавост на поставянето на блокове намалява вероятността от пропуски



Управление на пропуските

- ❖ Пропуски при четене винаги извлича блокове от паметта – Политика на заместване;
- ❖ При запис трябва внимателна поддръжка на консистентността на кеша и оперативната памет (ОП) – Политики на запис;

Видове пропуски

- ❖ Студен пропуск

При първи достъп до данните

- ❖ Предварително извличане може да намали забавянето

- ❖ Пропуск породен от капацитета

*Предходният запис е бил премахнат, защото прекалено много други данни са били достъпвани.
Множеството работни данни е прекалено голямо*

- ❖ Реорганизиране на алгоритъма да преизползва данните преди да се премахнат от кеша

- ❖ В противен случай предварително извличане

Видове пропуски

❖ Пропуск при конфликти

Много данни се съпостават на едно и също място в кеша, което довежда до премахване дори и когато кеша не е пълен

❖ Реорганизиране на данните и/или изместване на масивите

❖ Пропуск True Sharing

Нишка в друг процесор е използвала данните и те са преместени в друг кеш

❖ Минимизиране на споделянето/заклучването

❖ Пропуск False Sharing

Другия процесор е използвал други данни в същия запис в кеша и записът е бил преместен

❖ Изместване (падинг) на данните и осигуряване на заключени структури да не попадат в кеша

Политики на заместване на ел. от кеша

При асоциативен кеш, кой блок памет трябва да се замени когато множеството се напълни?

- ❖ Произволен;
- ❖ Най-рядко използваният (Least Recently Used)
 - ❖ Състоянието трябва да бъде обновявано при всеки достъп
 - ❖ Реализацията има смисъл при кешове с малки множества на асоциативност (2, 4 мн.)
 - ❖ Псевдо LRU (като двоично дърво или чрез битове) (4, 8 мн.)
- ❖ First In First Out т.нар. Round-Robin (високо асоц. Кешове);
- ❖ Not Least Recently Used (NLRU)
 - ❖ FIFO с изключение за най-често използвания блок

Политики на заместване на ел. от кеша

❖ Тривиално при директното съпоставяне

Асоциативност	2 асоциативен		4 асоциативен		8 асоциативен	
	LRU	Произв.	LRU	Произв.	LRU	Произв.
Размер						
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Емпиричните резултати показват, че с увеличаване на размера на кеша се намалява значението на политиката на заместване

Политики за запис

❖ При попадение

❖ Write through: запис в кеша и ОП

По-голям трафик, но по-лесна синхронизация и кохерентност на кеша с паметта

❖ Write back: запис само в кеша

Записва се в паметта само когато блока се извади от кеша. Поддържа се „мръсен“ (dirty) бит, чрез който се намалява трафика

❖ При пропуск

❖ No write allocate: запис само в ОП

❖ Write allocate (ака извличане при запис)

❖ Хибридни

❖ Write through and no write allocate

❖ Write back with write allocate

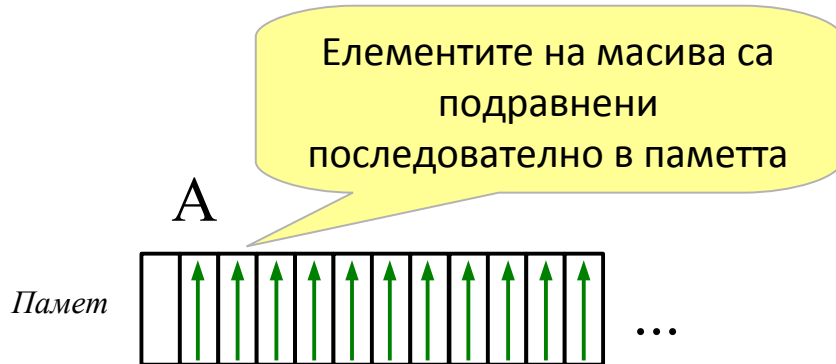
Прост кеш. Начални условия.

- ❖ 32Кб, директно съпоставяне, 64 байтови линии (512 записа);
- ❖ Достъп до кеша – един цикъл;
- ❖ Достъп до паметта – 100 цикъла;
- ❖ Думата е 1 байт;

Начини на достъп. Примери

```
#define S ((1<<20)*sizeof(int))
int A[S];

for(i=0; i<S; i++)
    Read A[i];
```



Достъп до данните:

- ❖ Четене на нов запис
- ❖ Четене на останалата част от записа
- ❖ Преместване на следващия запис

$\text{sizeof}(\text{int}) = 4$

16 елемента на запис(линия)

15 от всеки 16 са попадения

Общо време за достъп: $15*S/16+100*S/16$

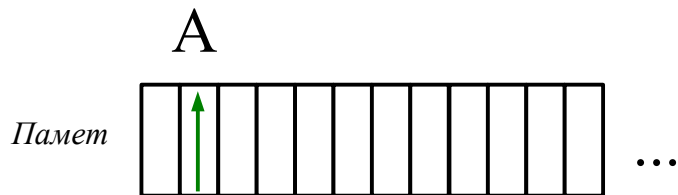
Какъв вид локалност? Пространствена

Какви видове пропуски? Студени

Начини на достъп. Примери

```
#define S ((1<<20)*sizeof(int))
int A[S];

for(i=0; i<S; i++)
    Read A[0];
```



Достъп до данните:

❖ Четене A[0] всеки път

sizeof(int) = 4

S броя четене

Всички достъпи (без първия) са
попадения

Общо време за достъп: 100 + S-1

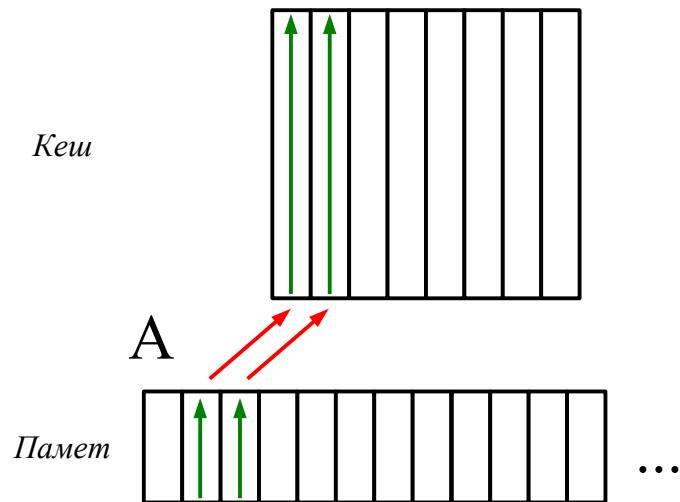
Какъв вид локалност? Времева

Какви видове пропуски? Студени

Начини на достъп. Примери

```
#define S ((1<<20)*sizeof(int))
int A[S];

for(i=0; i<S; i++)
    Read A[i % (1<<N)];
```



Достъп до данните:

- ❖ Четене на началния сегмент на A
МНОГО ПЪТИ

S броя четене

$4 \leq N \leq 13$

Един пропуск за всеки достъпен запис.
Останалите са попадения.

Достъпени записи: 2^{N-4}

Общо време за достъп: $2^{N-4} * 100 + S - 2^{N-4}$

Какъв вид локалност? Времева,
Пространствена

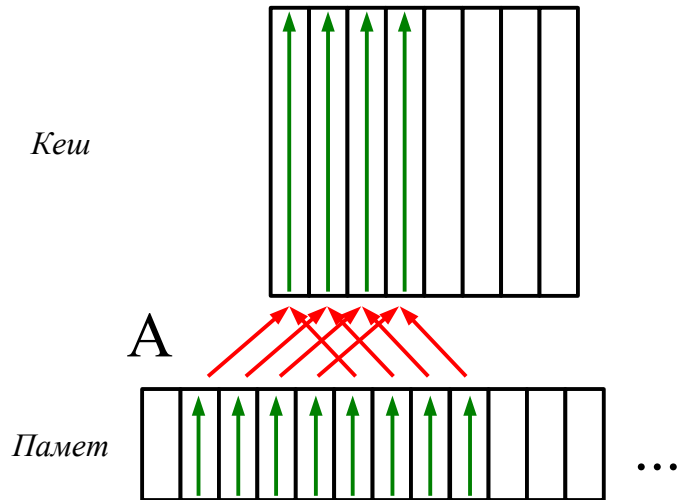
Какви видове пропуски? Студени



Начини на достъп. Примери

```
#define S ((1<<20)*sizeof(int))
int A[S];

for(i=0; i<S; i++)
    Read A[i % (1<<N)];
```



Достъп до данните:

- ❖ Четене на началния сегмент на A
МНОГО ПЪТИ

S броя четене

$N \geq 14$

Един пропуск за всеки достъпен запис.

Останалите са попадения.

Общо време за достъп: $15 * S / 16 + 100 * S / 16$

Какъв вид локалност?

Пространствена

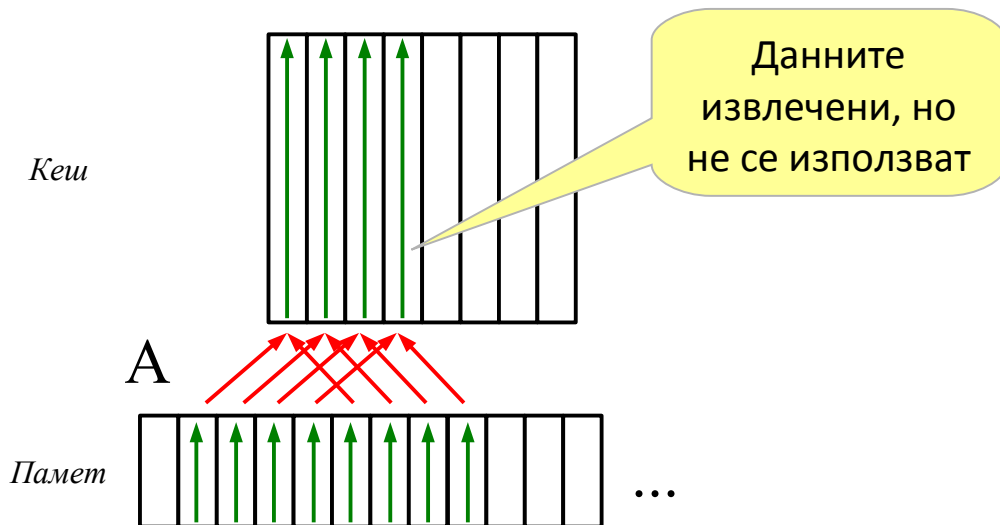
Какви видове пропуски? Студени, в

капацитета

Начини на достъп. Примери

```
#define S ((1<<20)*sizeof(int))
int A[S];

for(i=0; i<S; i++)
    Read A[(i*16) % (1<<N)];
```



Достъп до данните:

- ❖ Четене на всеки 16-ти елемент от началния сегмент на A много пъти

S броя четене

$N \geq 14$

Първи достъп до запис – пропуск.

Общо време за достъп: $100 * S$

Какъв вид локалност? Никаква

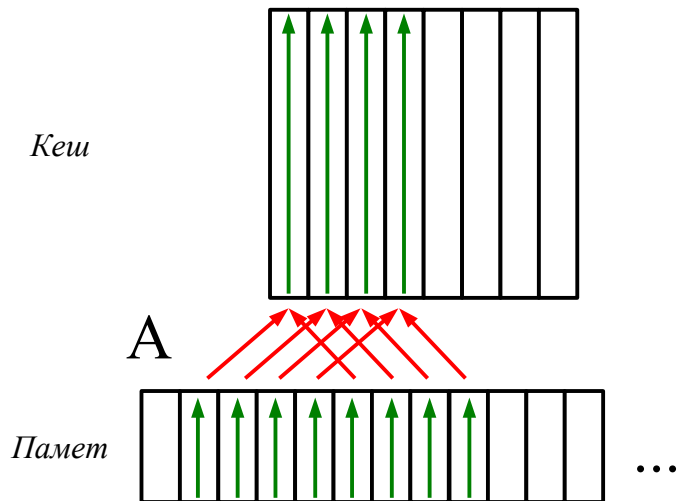
Какви видове пропуски?

Студени, в капацитета

Начини на достъп. Примери

```
#define S ((1<<20)*sizeof(int))
int A[S];

for(i=0; i<S; i++)
    Read A[random() %S];
```



Достъп до данните:

❖ Четене на произволни елементи от А

5 броя четене

Вероятност за попадение в кеша=8Кб/1Гб
= 1/256

Общо време за достъп:

$S * 255/256 * 100 + S * 1/256$

Какъв вид локалност?

Почти никаква

Какви видове пропуски?

Студени, в капацитета, конфликти

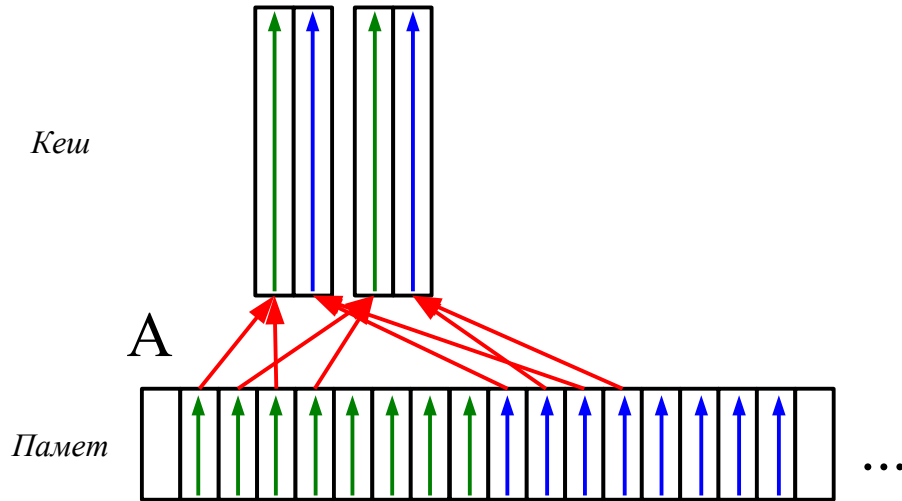
Множествено асоц. кеш. Начални условия.

- ❖ 32Кб, двуасоциативен, 64 байтови линии 256 множества;
- ❖ Много линии на множество, наричани пътища (way);
- ❖ Всяка дума от паметта се съпоставя на специфичен масив;
- ❖ Думата е 1 байт;

Начини на достъп. Примери

```
#define S ((1<<19)*sizeof(int))
int A[S];
int B[S];

for(i=0; i<S; i++)
    Read A[i], B[i];
```



Достъп до данните:

- ❖ Четене на елементи от А и В последователно

S броя четене

А и В се презастъпват в един и същи път, но има достатъчно линии

Общо време за достъп:

$$2 * (15/16 * S + 1/6 * S * 100)$$

Какъв вид локалност?

Пространствена локалност

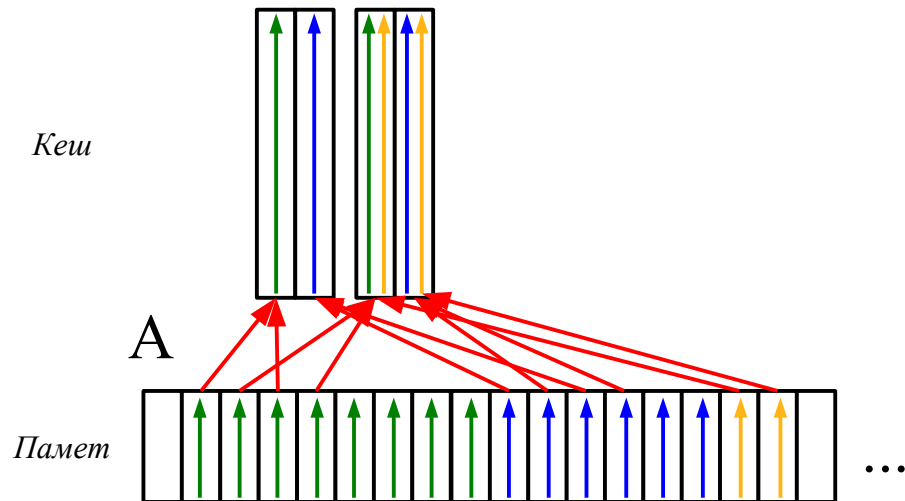
Какви видове пропуски?

Студени

Начини на достъп. Примери

```
#define S ((1<<19)*sizeof(int))
int A[S];
int B[S];
int C[S];

for(i=0; i<S; i++)
    Read A[i], B[i], C[i];
```



Достъп до данните:

- ❖ Четене на елементи от A и B последователно

S броя четене

A и B се презастъпват в един и същи път, но няма достатъчно линии

Общо време за достъп (с LRU):

$3 * S * 100$

Какъв вид локалност?

Пространствена локалност

Какви видове пропуски?

Студени, конфликти

Пример

Блоково умножение на Матрици

L1D кеш: 32 КБ

L2 кеш: 6 МБ

sizeof(double) = 8 байта

1. $3 \cdot 16 \cdot 16 \cdot 8 = 6$ КБ

L1 L2

2. $3 \cdot 32 \cdot 32 \cdot 8 = 24$ КБ

L1 L2

3. $3 \cdot 64 \cdot 64 \cdot 8 = 96$ КБ

L1 L2

4. $3 \cdot 128 \cdot 128 \cdot 8 = 384$ КБ

L1 L2

5. $2 \cdot 256 \cdot 256 \cdot 8 = 1.5$ МБ

L1 L2

6. $3 \cdot 512 \cdot 512 \cdot 8 = 6$ МБ

L1 L2

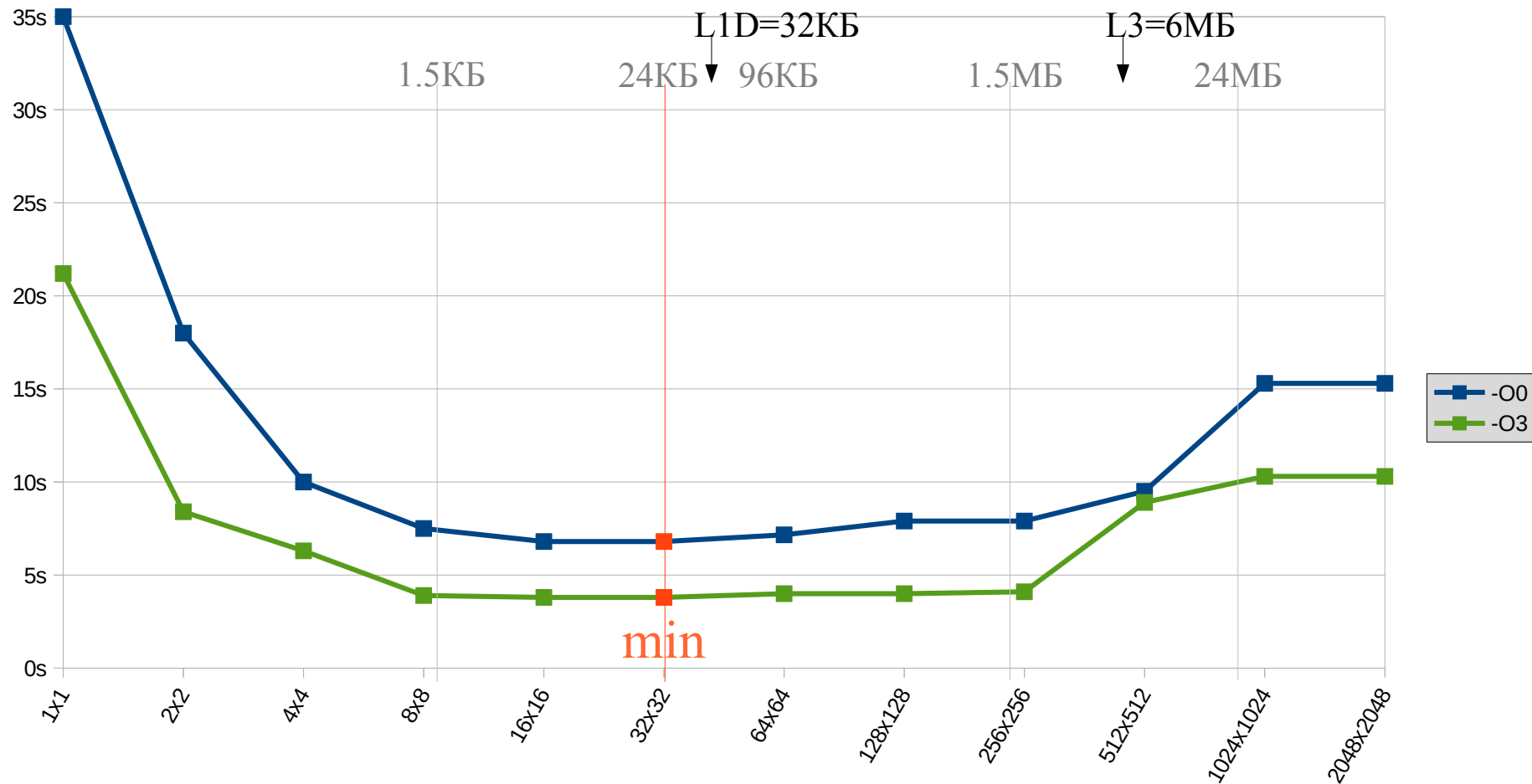
7. $3 \cdot 1024 \cdot 1024 \cdot 8 = 24$ МБ

L1 L2

8. $3 \cdot 2048 \cdot 2048 \cdot 8 = 32$ МБ

L1 L2

Матрици 1024x1024 на i7 (32КБ, 256КБ, 6МБ)



Архитектури на Паметта на Паралелните Компютри

Паралелни архитектури

❖ Споделена памет (Shared memory)

Много процесори могат да правят достъп до обща памет.

❖ Еднороден достъп до паметта (Uniform memory access – UMA)

Идентични процесори имат еднакъв достъп и еднакво време на достъп до паметта.

❖ Не-еднороден достъп до паметта (Non-uniform memory access – NUMA)

Не всички процесори имат еднакъв достъп и еднакво време на достъп до паметта.

❖ Cache only memory architecture (COMA)

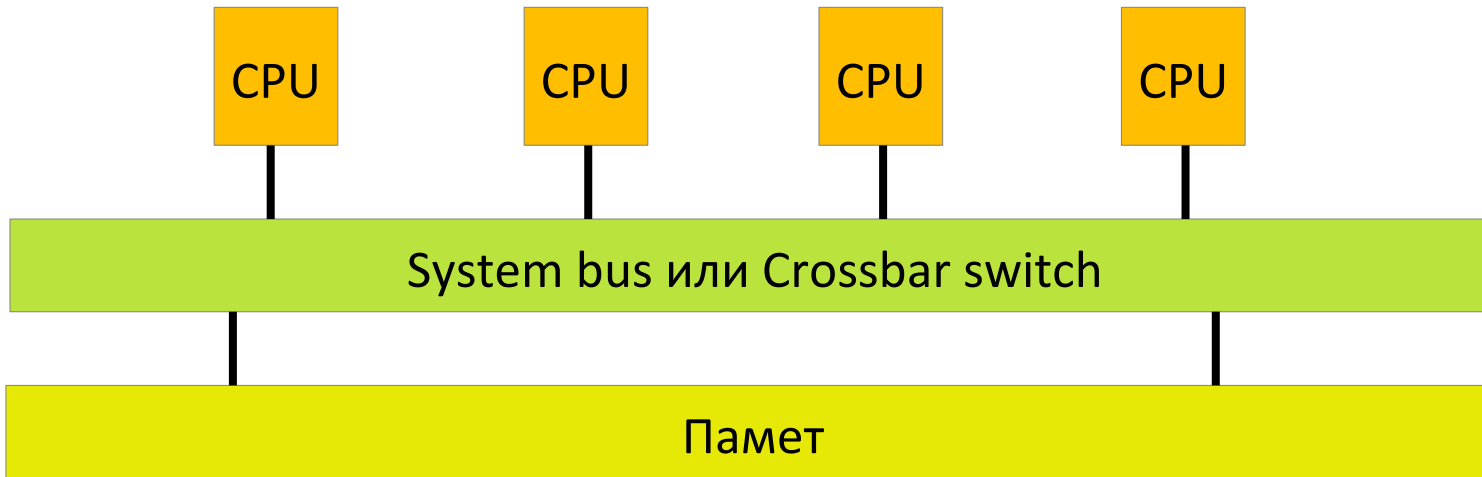
Цялата памет на възлите се използва само като кеш.

❖ Разпределена памет (Distributed memory)

Изисква комуникация по мрежата за достъп до между процесорната памет.

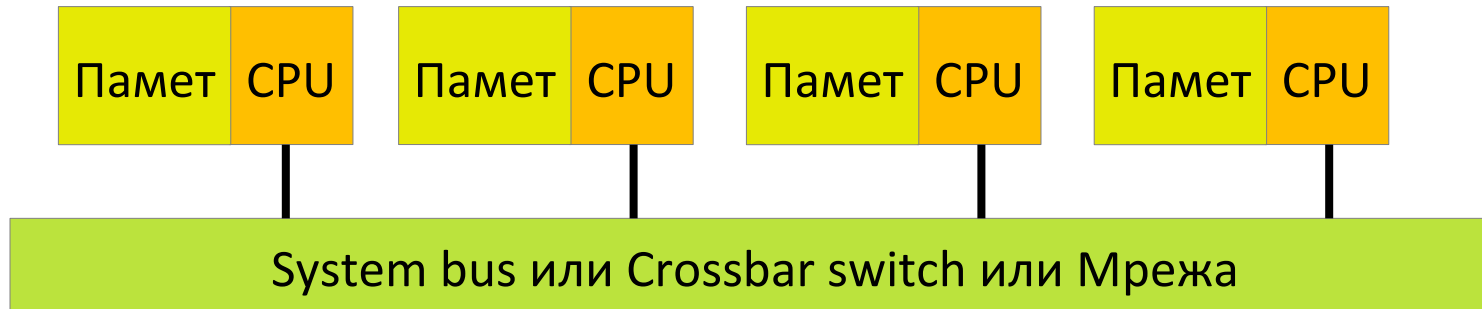
❖ Хибридна Разпределено-Споделена памет (Hybrid Distributed-Shared Memory)

Споделена памет (Shared memory)



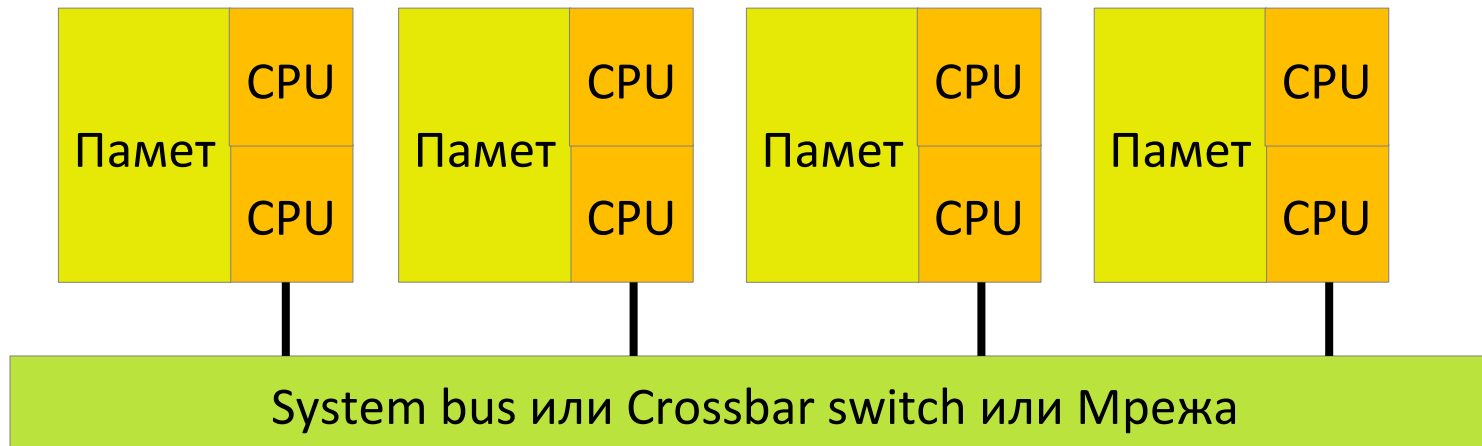
Всички процесори могат да правят достъп до обща памет.

Разпределена памет (*Distributed memory*)



Изисква комуникация (по мрежата) за достъп до разпределената памет.

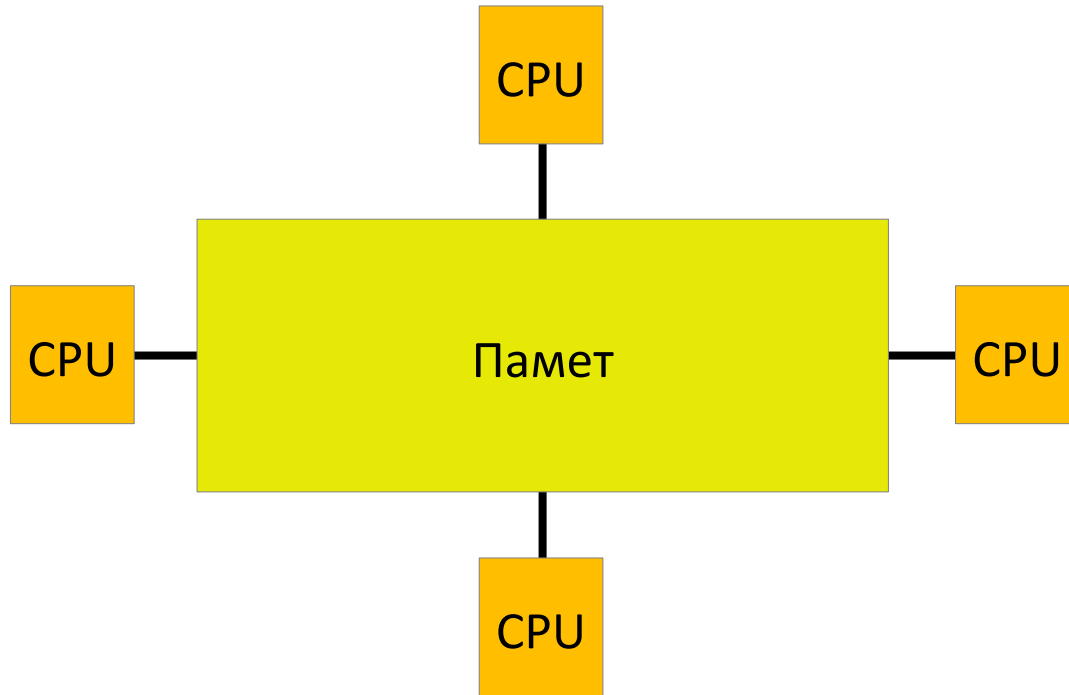
Хибридна Разпределено-Споделена памет (Hybrid Distributed-Shared Memory)



Притежава повечето от предимствата и недостатъците на предишните две.
Това е най-използваната в момента архитектура в големите компютри.

Еднороден достъп до паметта (Uniform memory access – UMA)

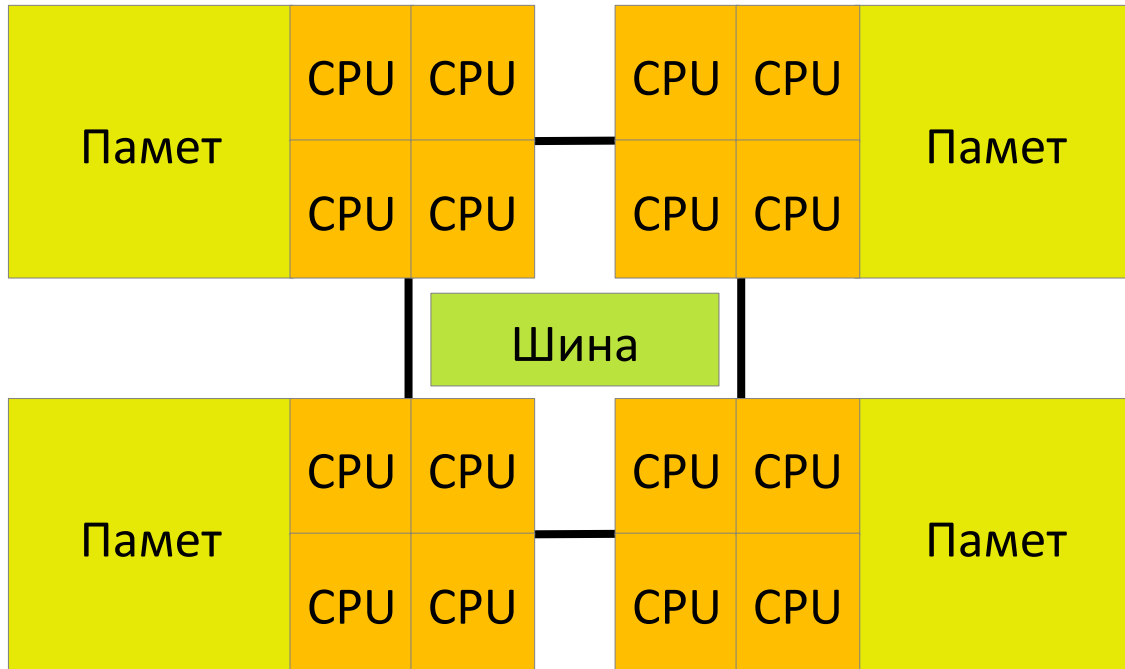
(SM)



- ❖ Много (идентични) процесори;
- ❖ Равноправен достъп;
- ❖ Еднакво време за достъп;

Не-еднороден достъп до паметта (Non-uniform memory access – NUMA)

(SM)

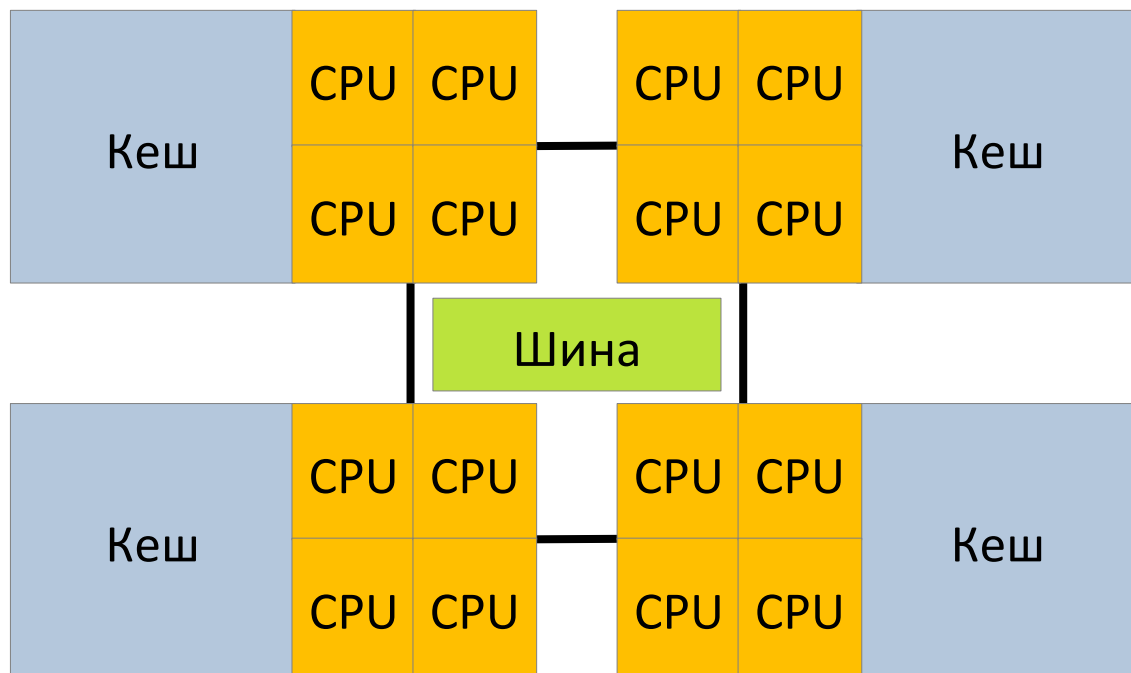


- ❖ Много процесори;
- ❖ Не всички процесори имат равноправен достъп до цялата памет;
- ❖ Достъпа до паметта през връзката е бавен;
- ❖ Програмиста гарантира за коректния достъп до глобалната памет;

Само кеш

(SM)

(Cache only memory architecture – COMA)



- ❖ Много процесори;
- ❖ Цялата памет се използва за кеш на данните;
- ❖ Може да се използва съвместно/хибридно с NUMA;

Shared-Nothing Архитектура

Shared-Nothing Архитектура

- ❖ Архитектурата Shared-Nothing (SN) е разпределена архитектура, в която всяка заявка за актуализация се изпълнява от един възел (процесор/памет/диск). Целта е да се премахне “съперничеството” между възлите; (*Web, DB, ...*)
- ❖ Възлите не споделят (едновременен достъп до) памет или диск;
- ❖ SN елиминира единични точки на повреда, което позволява на цялостната система да продължи да работи въпреки грешките в отделните възли;
- ❖ SN може да се мащабира чрез лесно добавяне на възли;
- ❖ SN обикновено разпределя данните между много възли, чрез репликация, което позволява да се изпълняват повече заявки;

Масивно Паралелни Компютри

GRID Системи,

Компютърни Клъстъри,

Силно паралелни процесорни масиви (MPPA)

Масивно Паралелни Компютри

Масивно паралелни компютри е използването на голям брой процесори (или отделни компютри) за извършване на набор от координирани изчисления паралелно (едновременно).

- ❖ GRID системи/изчисления;
- ❖ Компютърни клъстери;
- ❖ Силно паралелни процесорни масиви (MPPA);

GRID Системи

Grid системите са широко разпределени компютърни ресурси използвани заедно за постигането на обща цел.

- ❖ Използват middle-ware (например BOINC и др.);
- ❖ Те са форма на разпределен компютър (супер виртуален компютър);
- ❖ Изискват мрежа (Интернет) за връзка между системите;
- ❖ Слабо свързани системи;
- ❖ Обикновено силно хетерогенни;

Например, Worldwide LHC Computing Grid (WLCG) на ЦЕРН – над 170 компютърни центъра в 42 държави; Над 200000 ядра;

Компютърни Клъстери

Компютърен клъстер се нарича множество от компютри, което е (тясно) свързано за изпълнение на една задача и може да се разглежда като една цялостна система.

- ❖ Изискват (локална) мрежа, InfiniBand връзка и др.;
- ❖ Тясно свързани системи;
- ❖ Обикновено сравнително хомогенни;
- ❖ Специализиран софтуер оркестрира работата на клъстера;



Силно паралелни процесорни масиви (MPRA)

MPRA са вид интегрални схеми, които представляват масив от стотици или хиляди процесори и RAM памет. Тези процесори комуникират по между си чрез преконфигурируема взаимосвързаност на каналите.

