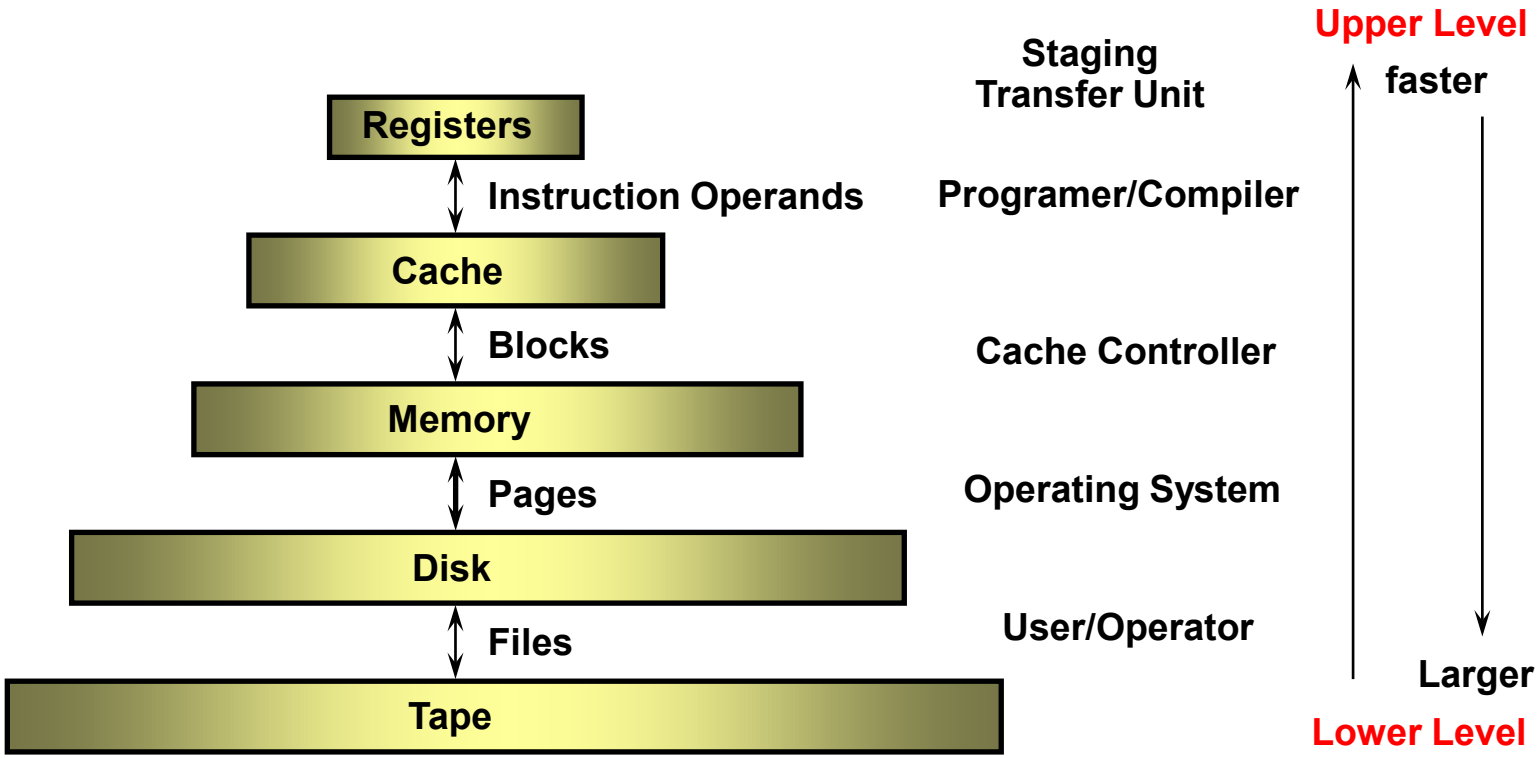
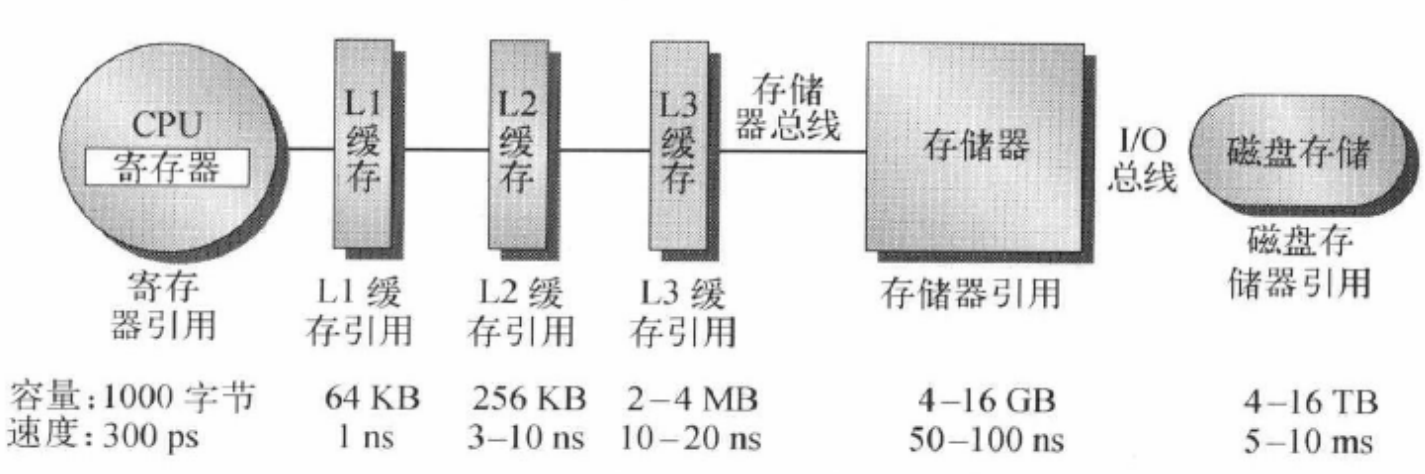
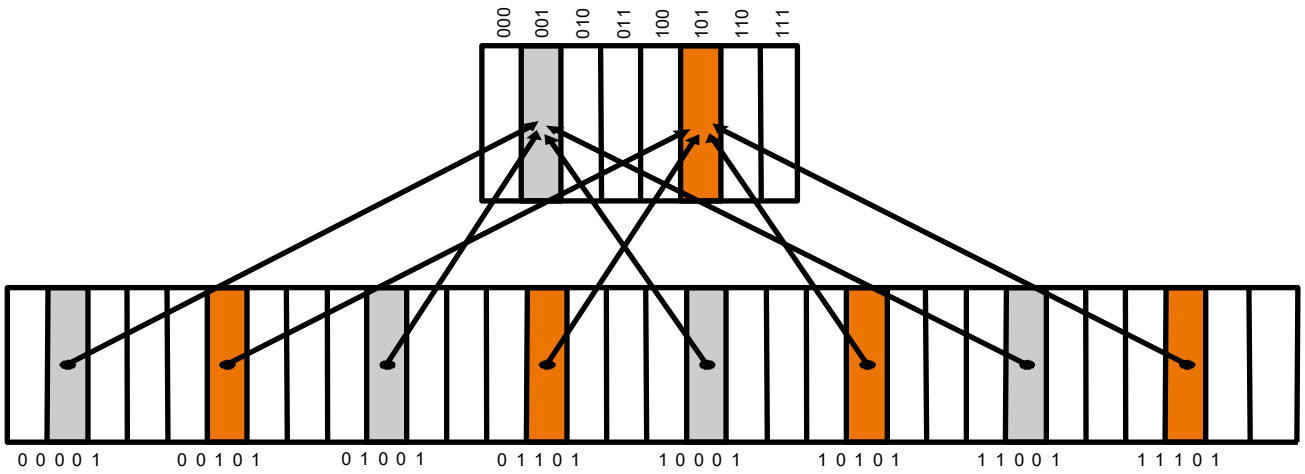
cache optimization

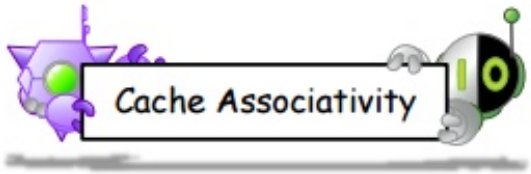
speaker: Gin
Hotine


Registers
Instruction Operands
Cache
Blocks
Memory
Pages
Disk
Files
Tape
Staging
Transfer Unit
Programer/Compiler
Cache Controller
Operating System
User/Operator
Upper Level
faster
Larger
Lower Level

Memory access

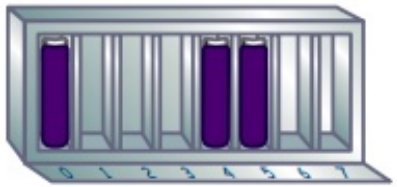



000
001
010
011
100
101
110
111
0 0 0 0 1
0 0 1 0 1
0 1 0 0 1
0 1 1 0 1
1 0 0 0 1
1 0 1 0 1
1 1 0 0 1
1 1 1 0 1



Just as bookshelves come in different shapes and sizes, caches can also take on a variety of forms and capacities. But no matter how large or small they are, caches fall into one of three categories: direct mapped, n-way set associative, and fully associative.

Direct Mapped



Tag	Index	Offset

A cache block can only go in one spot in the cache. It makes a cache block very easy to find, but it's not very flexible about where to put the blocks.

2-Way Set Associative

Tag	Index	Offset

This cache is made up of sets that can fit two blocks each. The index is now used to find the set, and the tag helps find the block within the set.

4-Way Set Associative

Tag	Index	Offset

Each set here fits four blocks, so there are fewer sets. As such, fewer index bits are needed.

Fully Associative

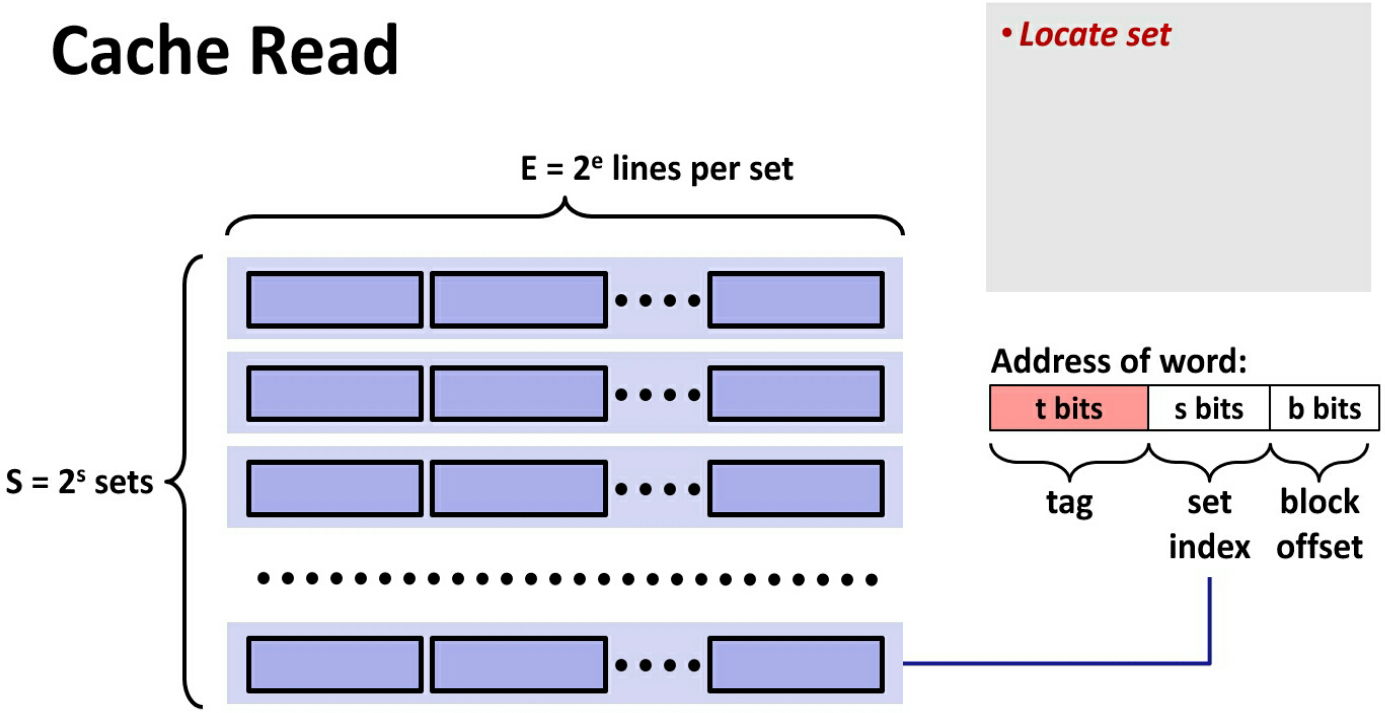
Tag	Offset

No index is needed, since a cache block can go anywhere in the cache. Every tag must be compared when finding a block in the cache, but block placement is very flexible!

time

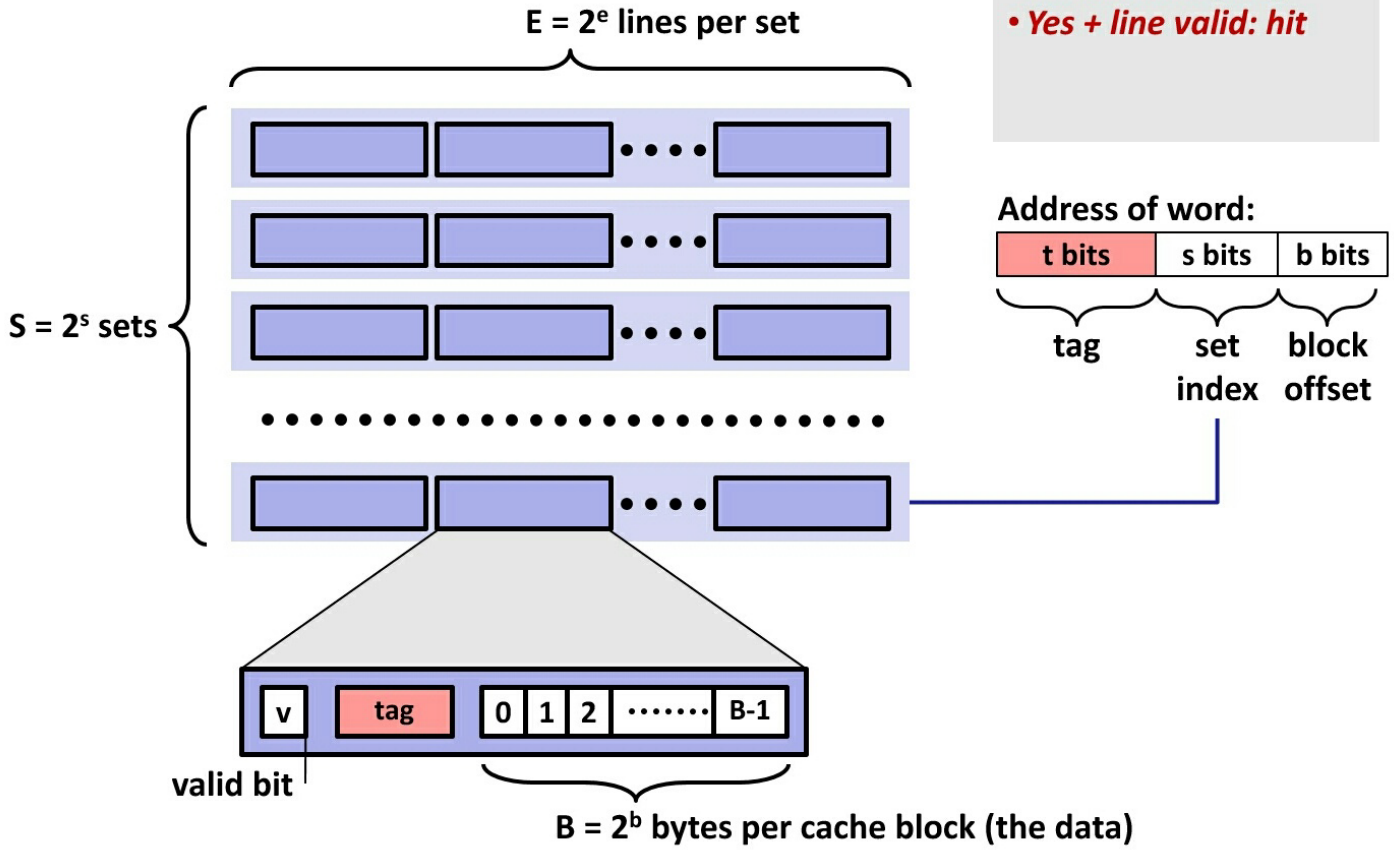
Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	

Cache Read


E = 2^e lines per set
S = 2^s sets
• Locate set
Address of word:
t bits
s bits
b bits
tag
set index
block offset

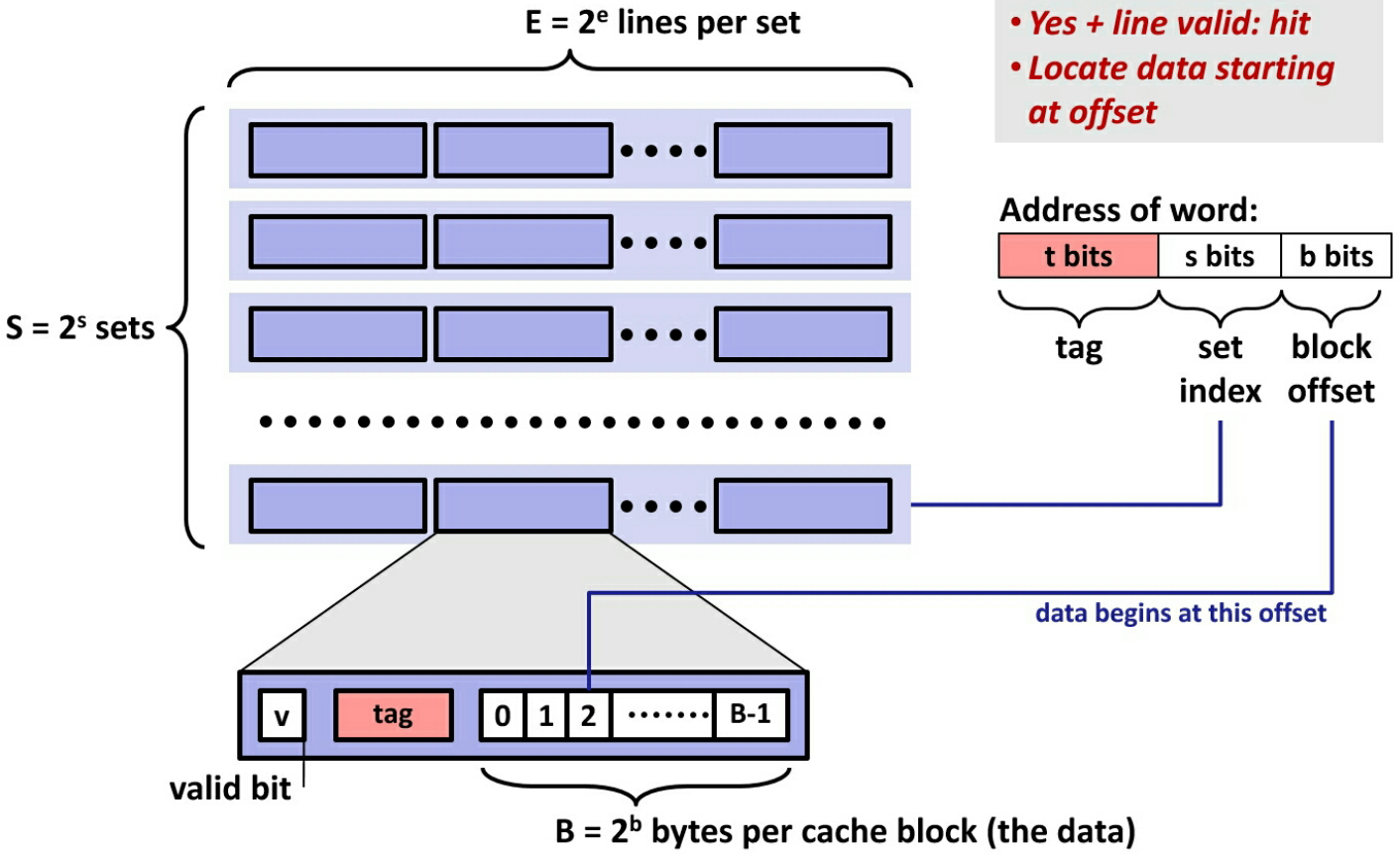
Cache Read

- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*



Cache Read

- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*



2-way set associative (0, 8, 0, 6, 8)

time

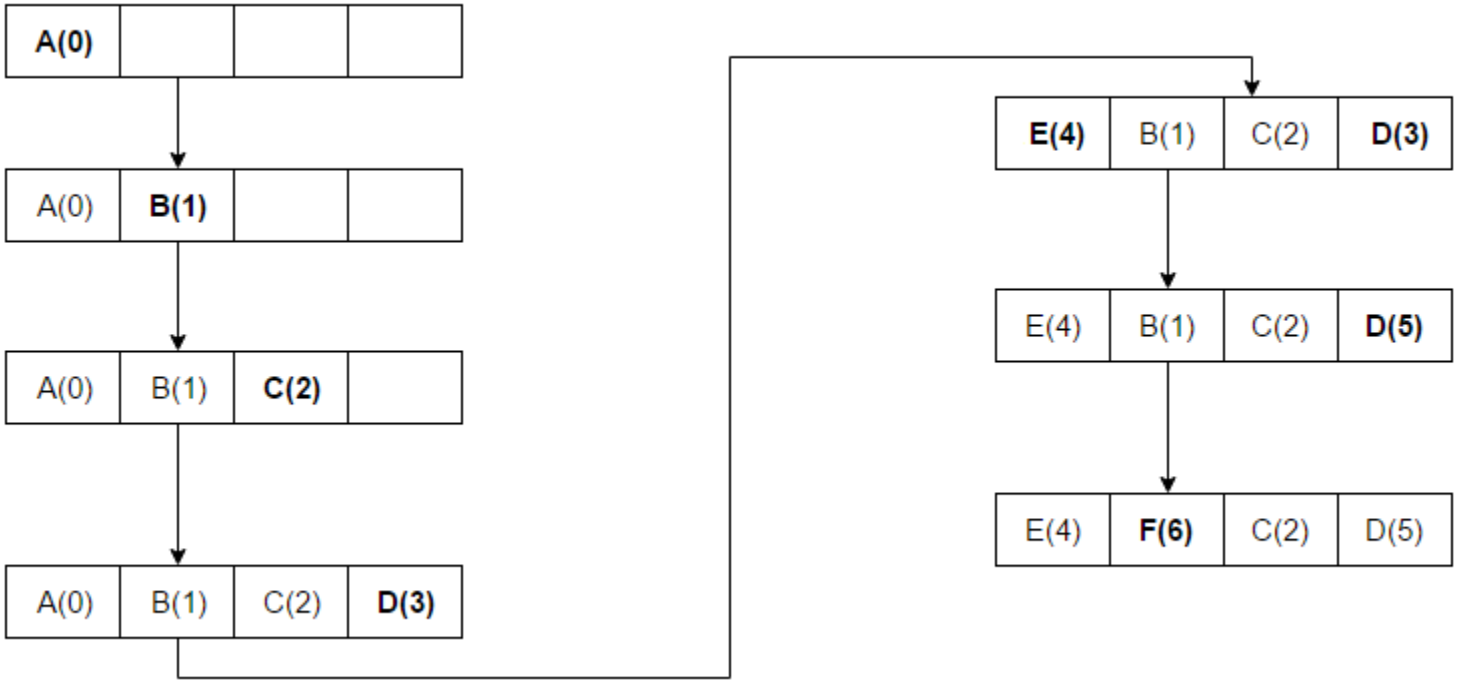
Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

Fully associative (0, 8, 0, 6, 8)

time

Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	M[em8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[8]	Mem[6]	

Cache Block Replacement



- Compulsory miss
- Conflict miss
- Cache coherency miss

time

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

Branch prediction

- icache optimization
- prediction failure's price is really high
- data conditional transportation

```
compare:
.LFB21:
    .cfi_startproc
    cmpq    %rsi, %rdi
    jl  .L4
    movq    %rsi, %rax
    cqto
    idivq   %rdi
    ret
.L4:
    movq    %rdi, %rax
    cqto
    idivq   %rsi
    ret
```

```
ccompare:
.LFB22:
    .cfi_startproc
    movq    %rdi, %rax
    cqto
    idivq   %rsi
    movq    %rax, %rcx
    movq    %rsi, %rax
    cqto
    idivq   %rdi
    cmpq    %rcx, %rax
    cmovg   %rcx, %rax
    ret
```

cache reordering

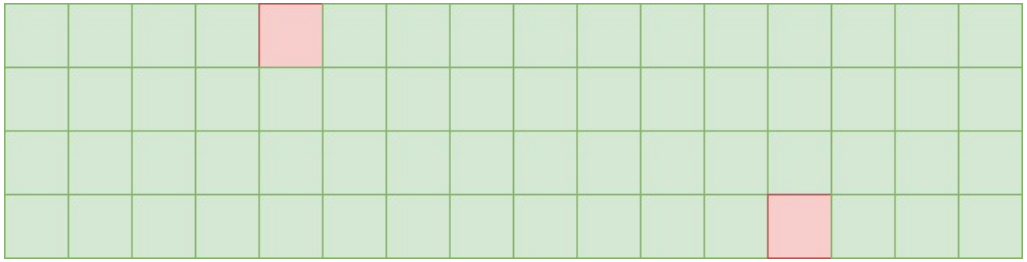
0	1	2	3												15
16															
32															

0	1	2	3	16			20								
4	5	6	7												
8	9	10	11												
12	13	14	15												

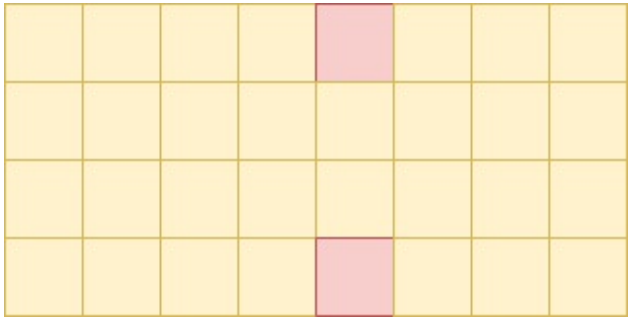
cache reordering

warm data: data which will be refered most times

cold data: data which will be refered may be just once



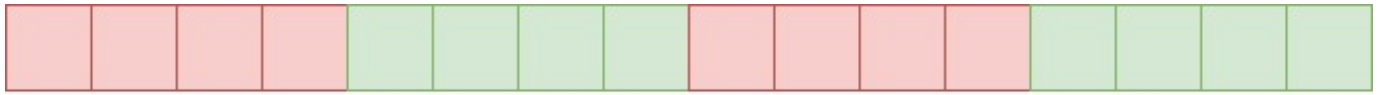
In cache:

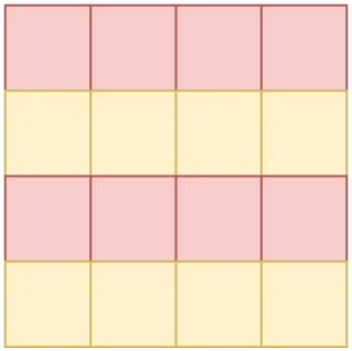


If more warm data?

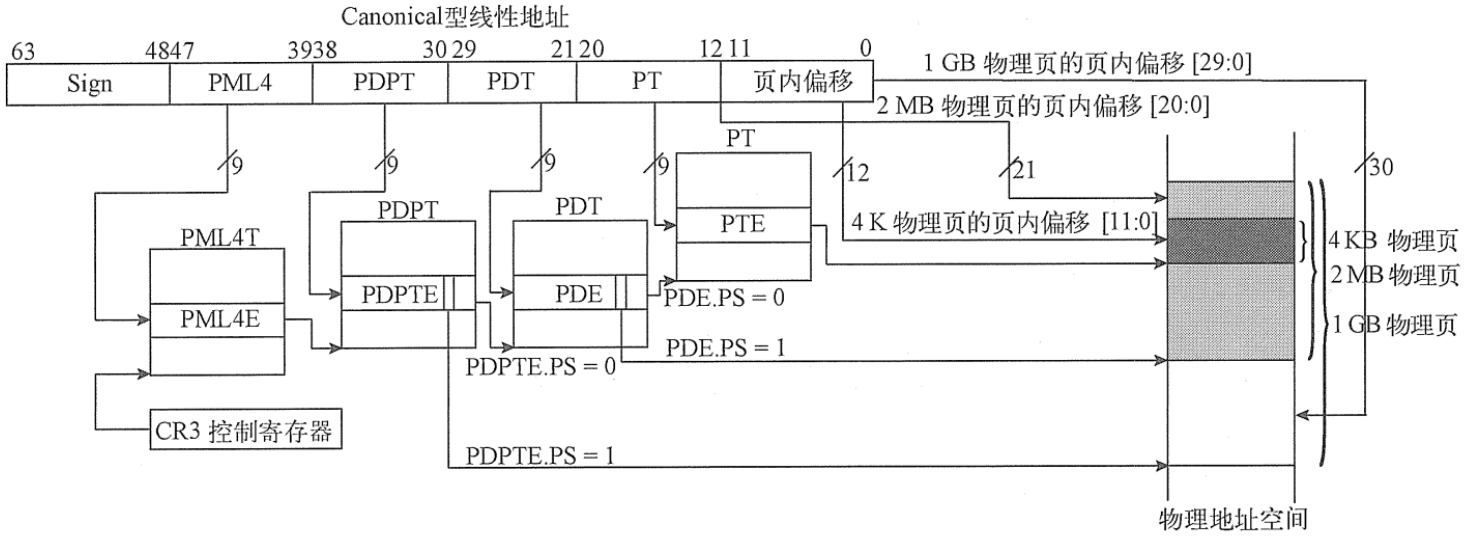
cache locality

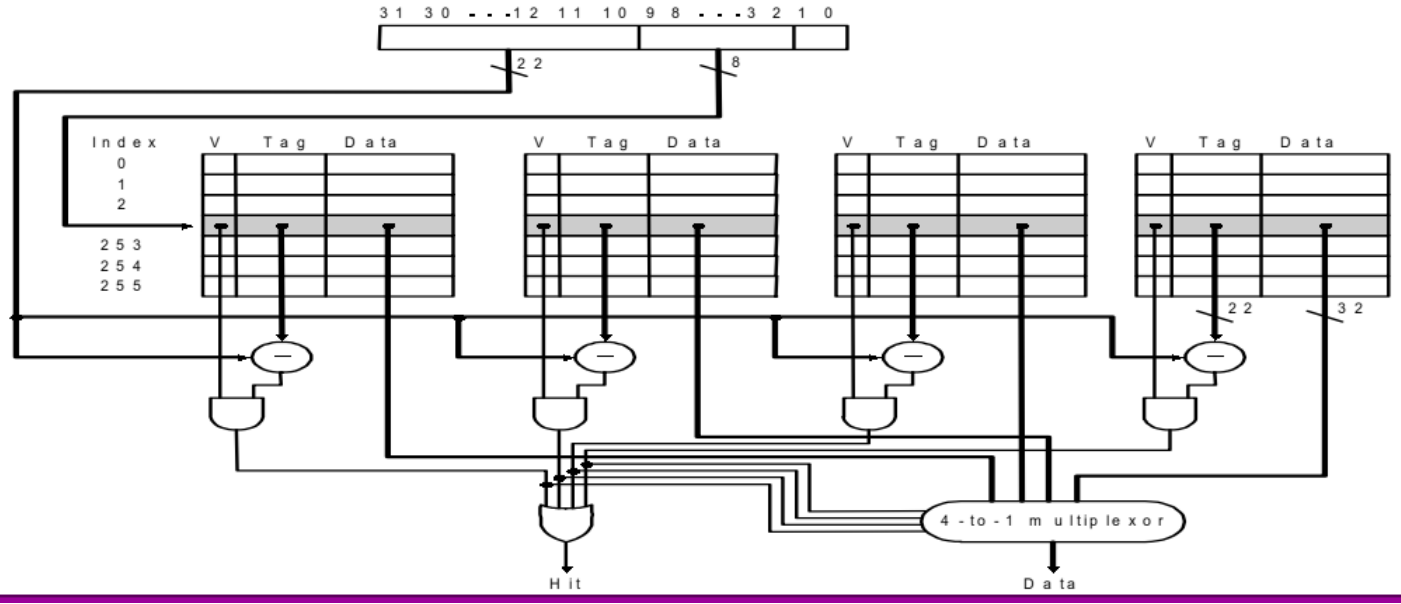
- what about multi-process?



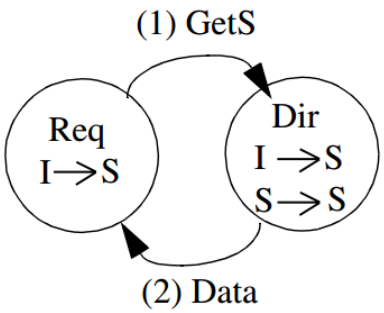


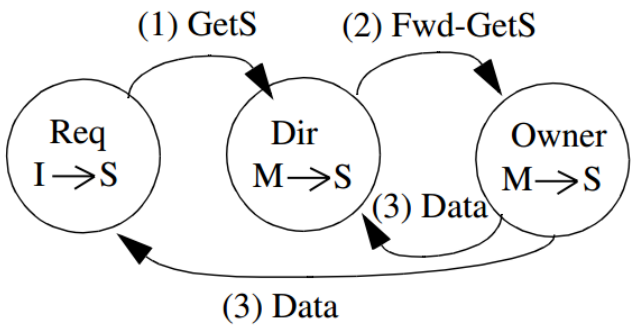
VIVP, VIPT, PIPT



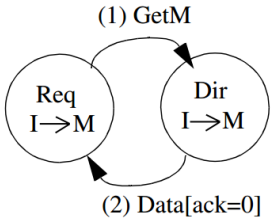


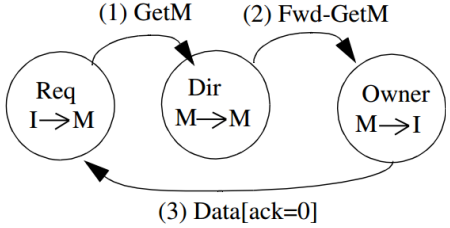
cache coherency: MSI

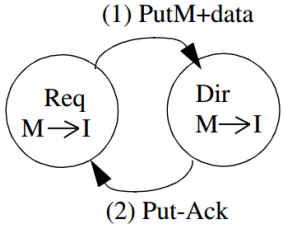


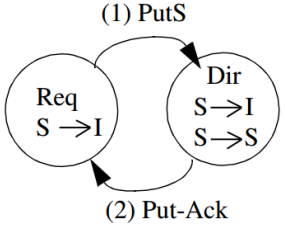


Transitions from I to S.

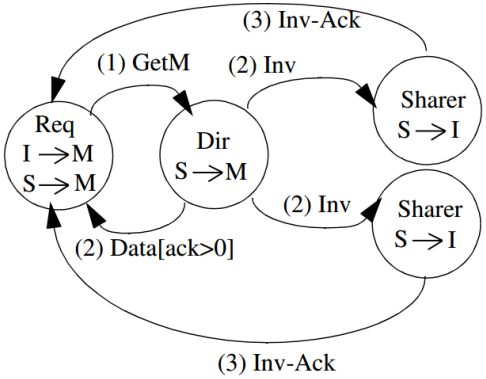








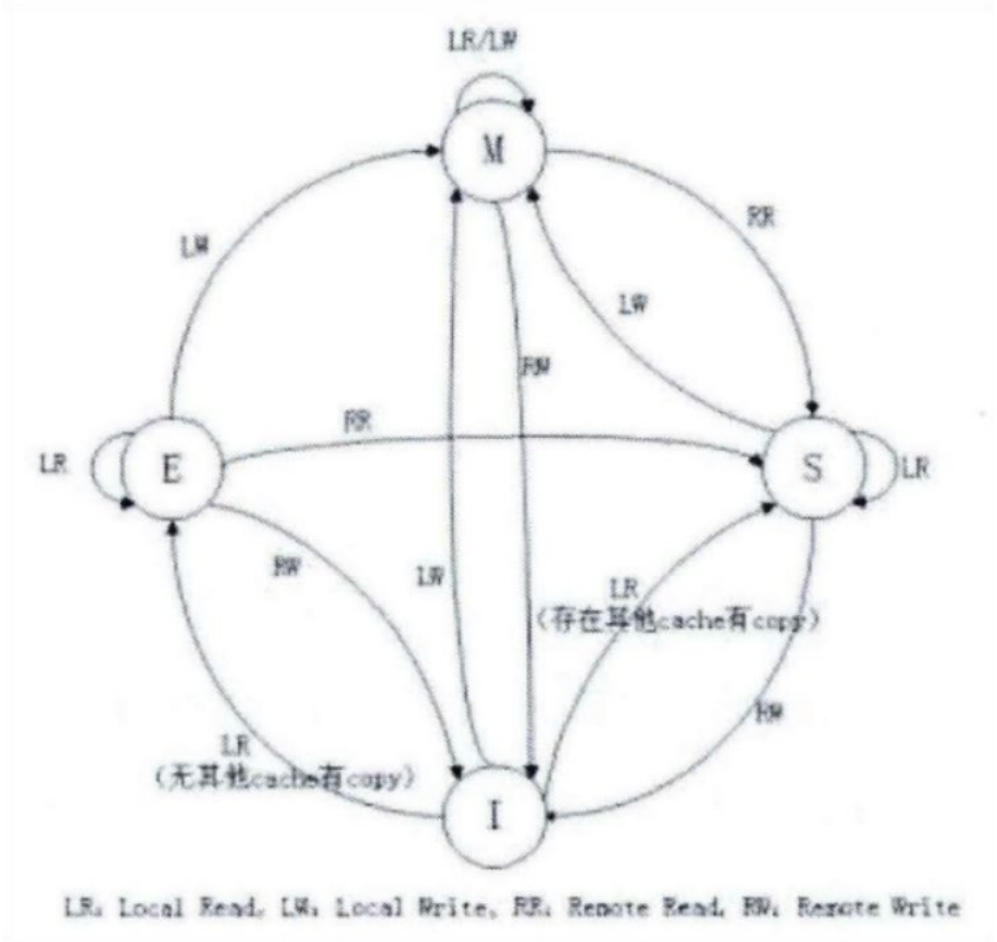
Transition from M or S to I



The only sharer might be the requestor, in which case no Invalidation messages are sent and the Data message from the Dir to Req has an AckCount of zero.

Transitions from I or S to M

cache coherency: MESI



cache coherency: MOESI