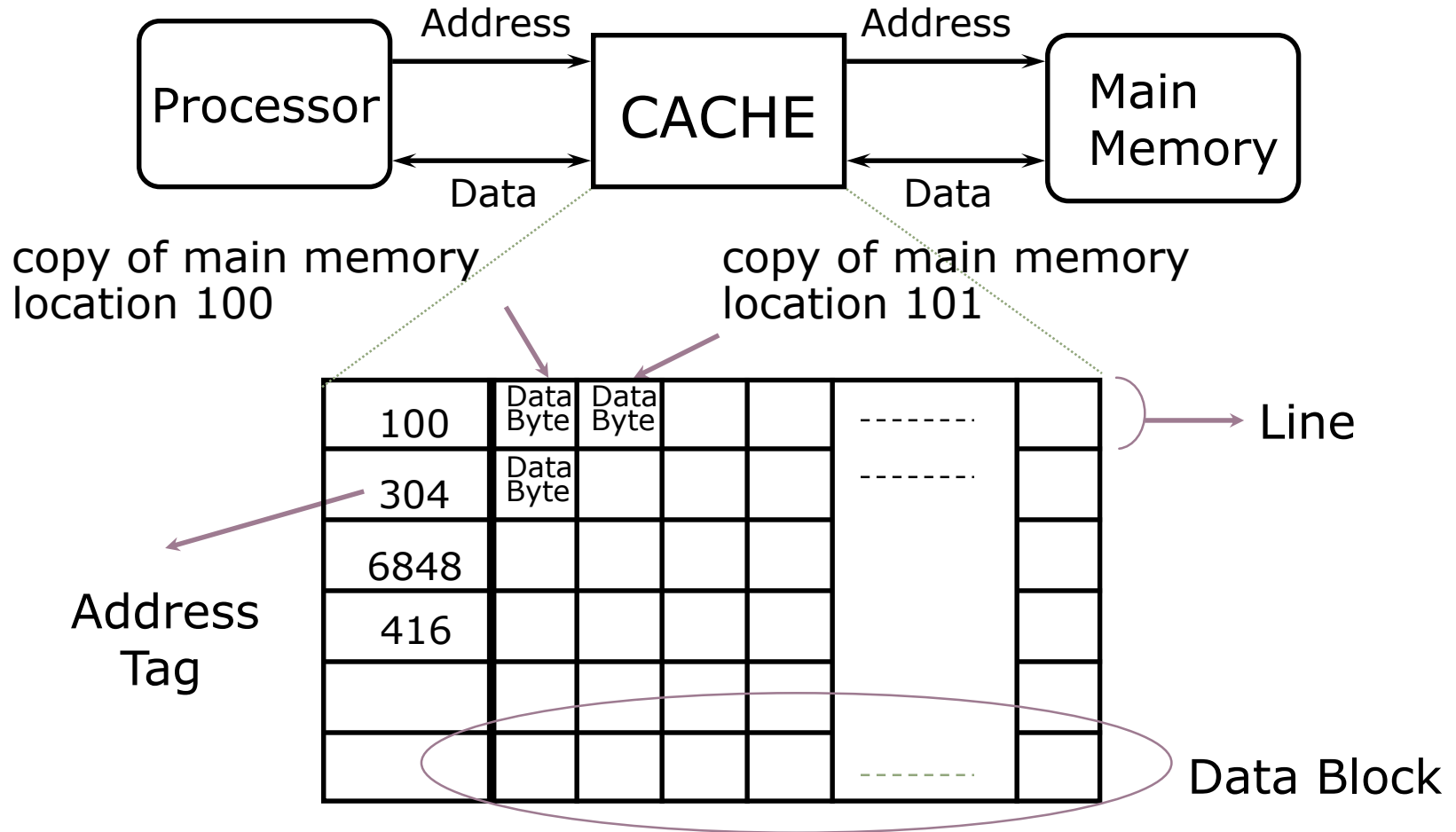# Caches (continued)

Daniel Sanchez
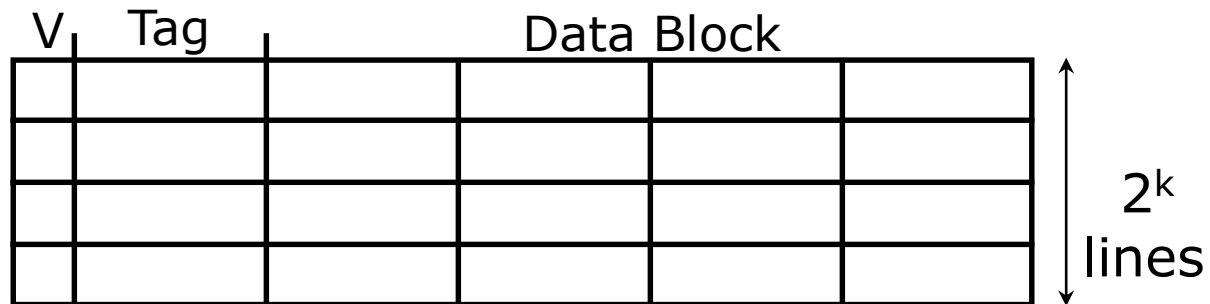Computer Science and Artificial Intelligence Laboratory
M.I.T.

Reminder: Inside a Cache
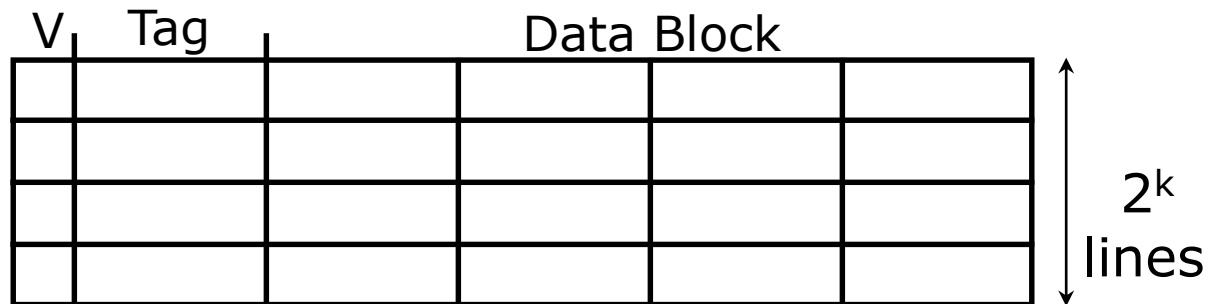
Processor — Address → CACHE — Address → Main Memory

Processor ← Data — CACHE ← Data — Main Memory

copy of main memory location 100

copy of main memory location 101

100	Data Byte	Data Byte			- - - - - -	
304	Data Byte				- - - - - .	
6848						
416						
					- - - - - - -	

Line

Address Tag

Data Block

Reminder: Direct-Mapped Cache

V | Tag   Data Block

2^k lines

Reminder: Direct-Mapped Cache

Block number   Block offset

Tag	Index	Offset

V   Tag   Data Block

2^k lines

Reminder: Direct-Mapped Cache

Block number ⎴ Block offset

Tag	Index	Offset

k

V | Tag | | Data Block

2^k lines

Reminder: Direct-Mapped Cache

Block number | Block offset

| Tag | Index | Offset |

t

k

V | Tag | | k | Data Block

2^k lines

t

HIT

Reminder: Direct-Mapped Cache

Block number   Block offset

Tag	Index	Offset

t

V   Tag   k   Data Block   b

2^k lines

t

HIT

Data Word or Byte

2-Way Set-Associative Cache

Tag	Index	Block Offset

V	Tag	Data Block
●	●	●

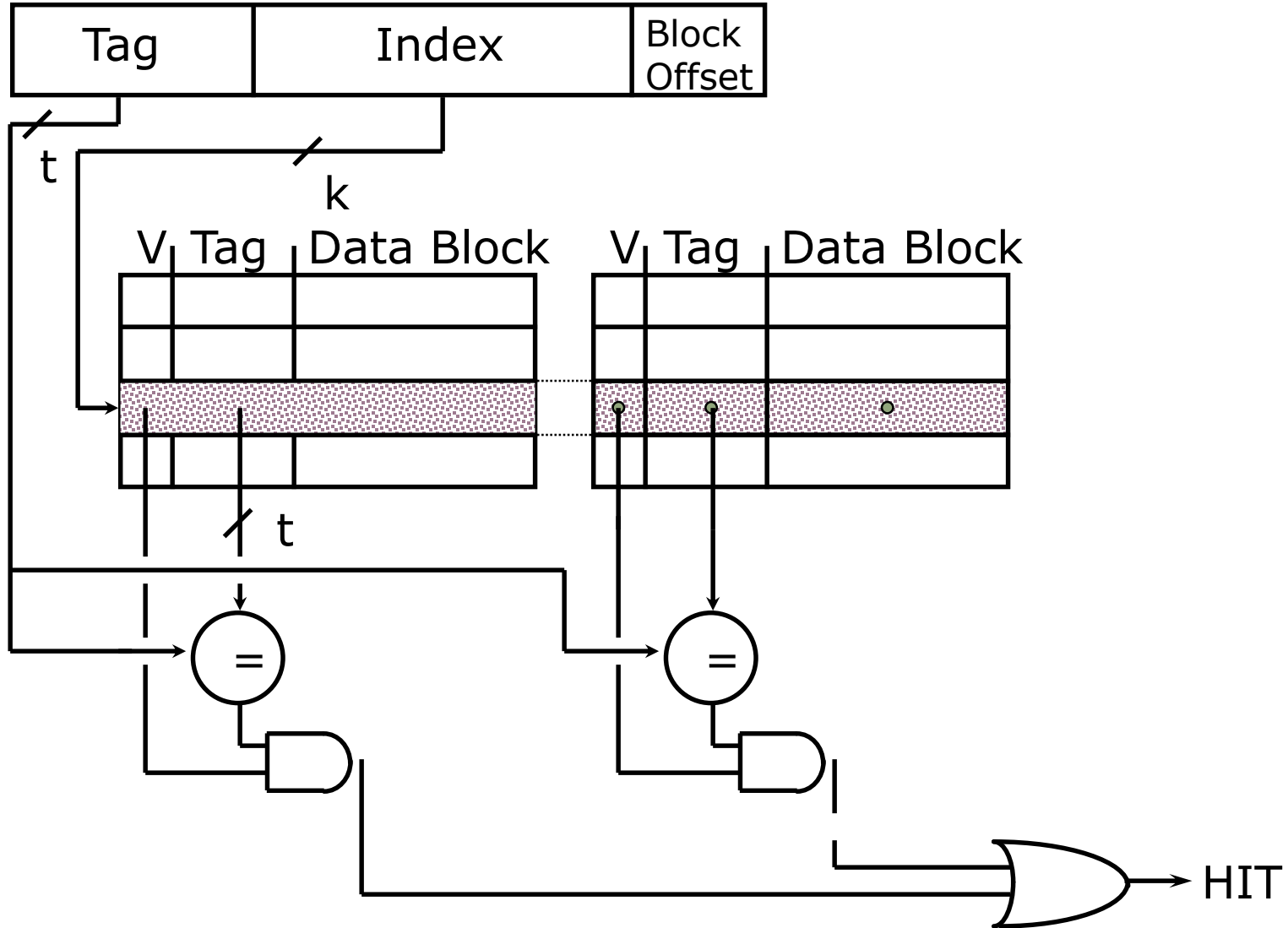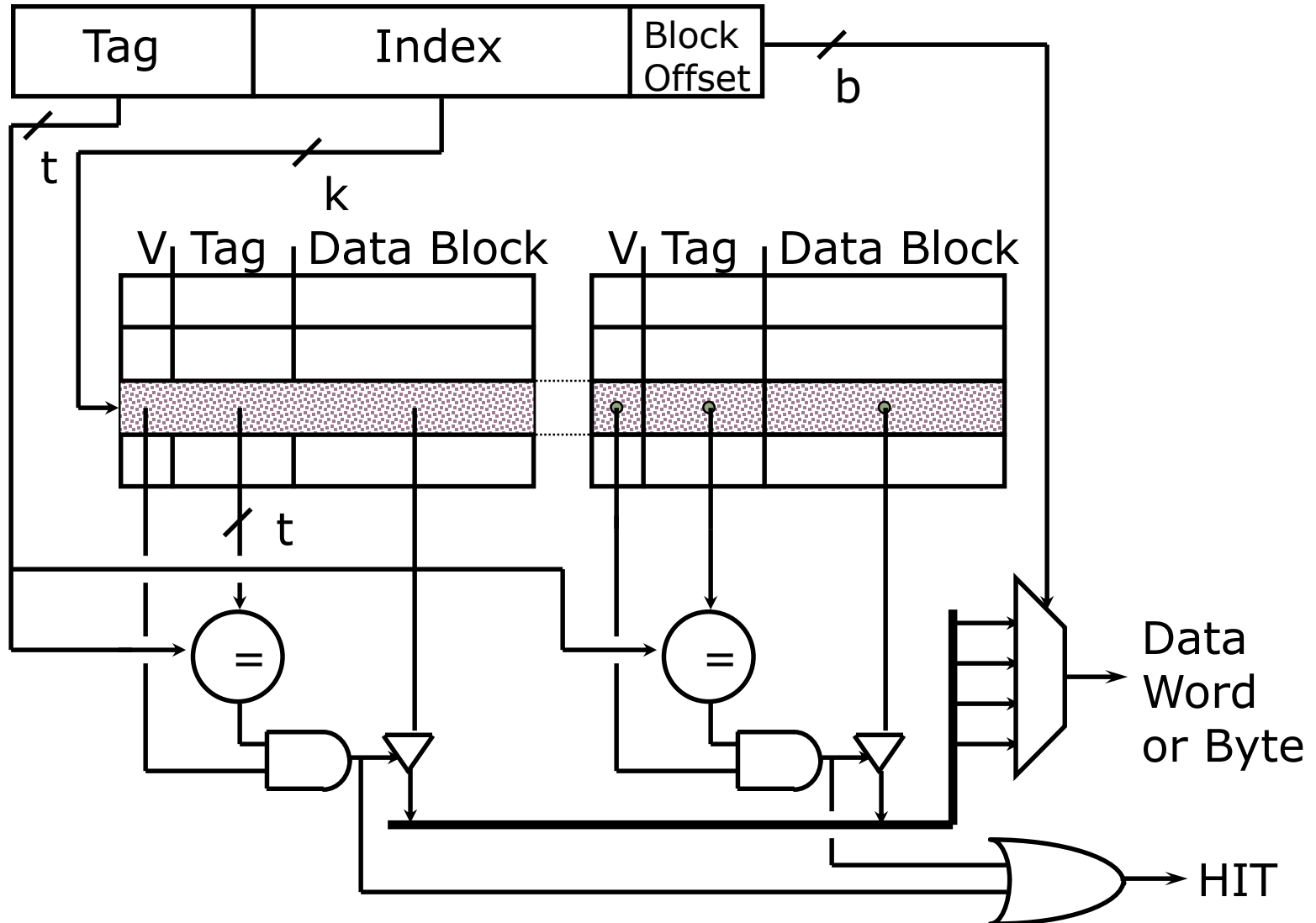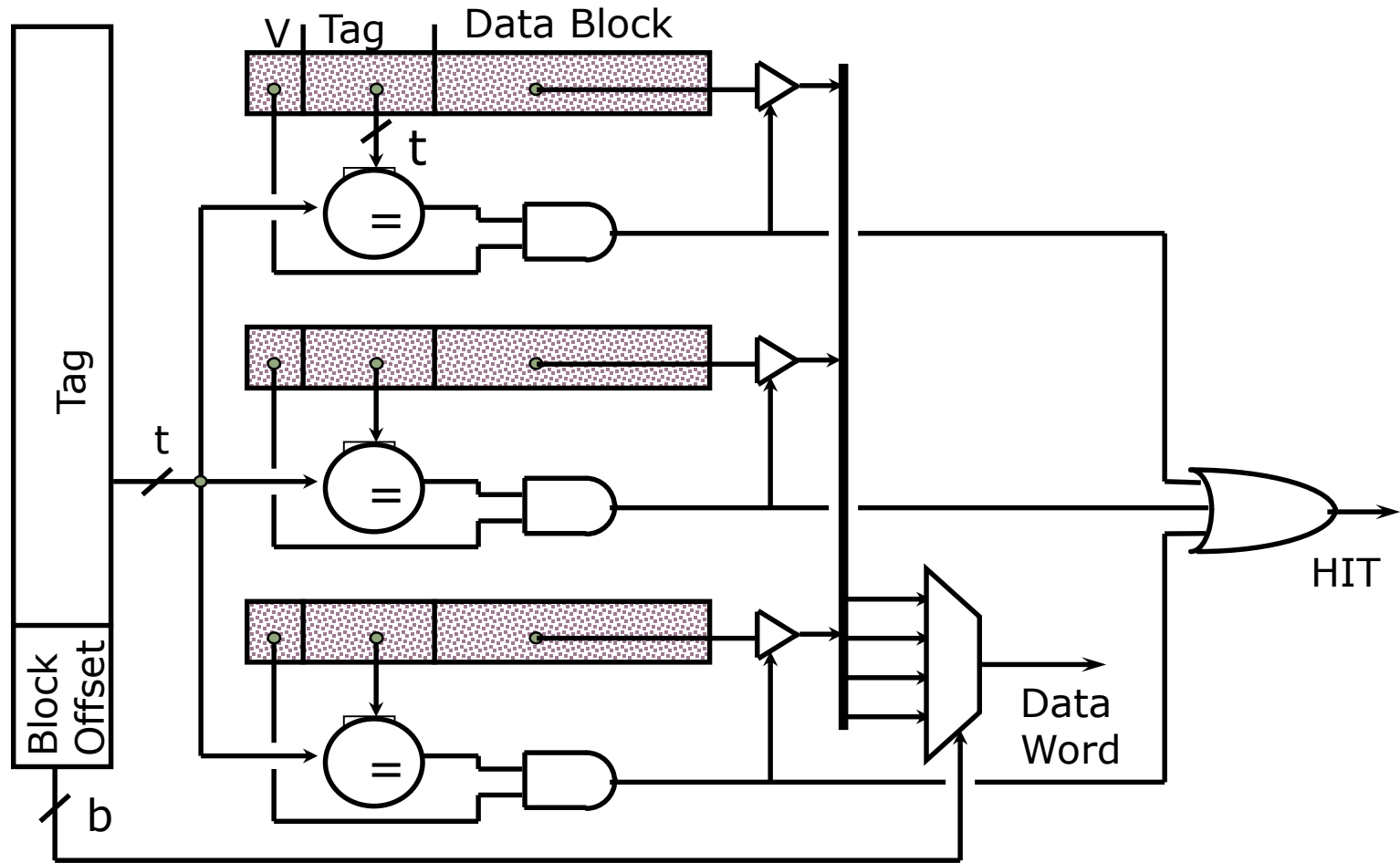| V	Tag	Data Block
		●
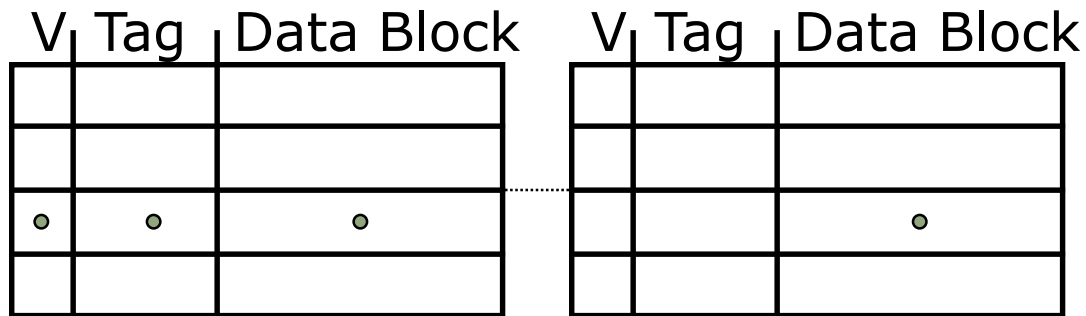
2-Way Set-Associative Cache
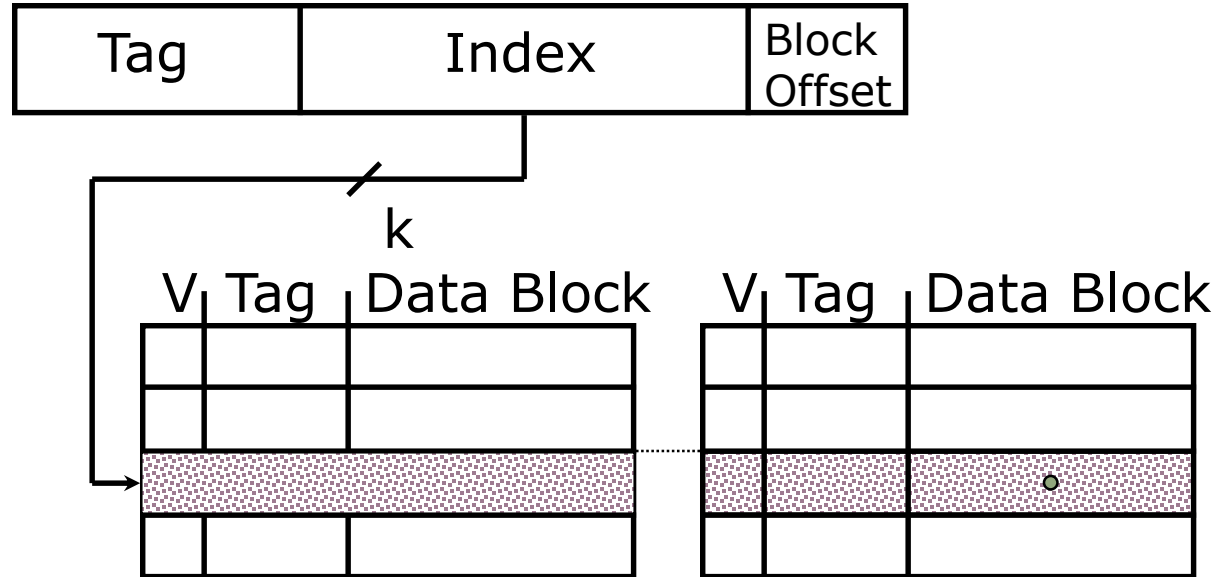
2-Way Set-Associative Cache
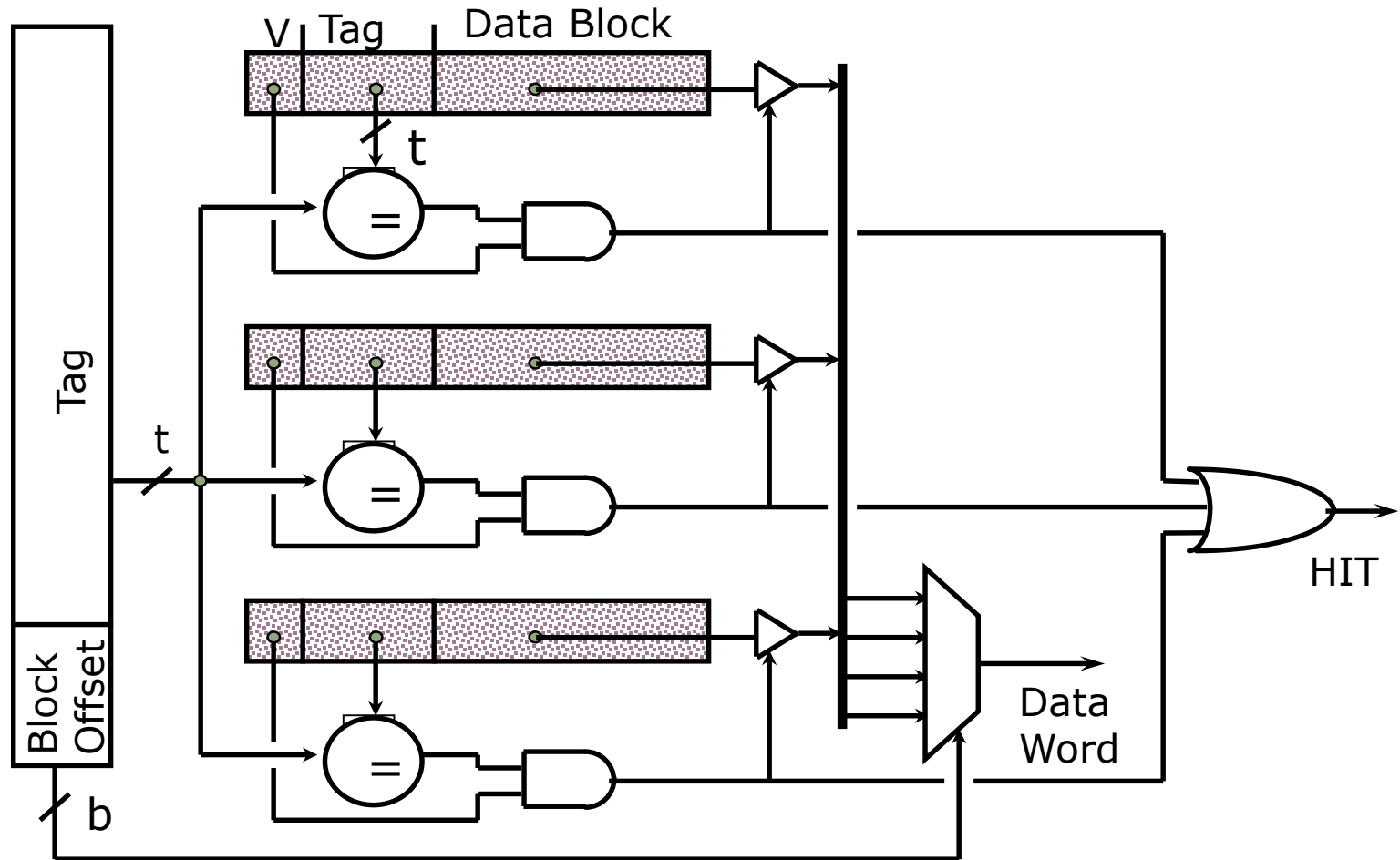
2-Way Set-Associative Cache

Fully Associative Cache

Q: Where are the index bits? _____

Fully Associative Cache

Q: Where are the index bits? _____ Not needed

Placement Policy

Block Number

| | | | | | | | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |

Memory

Set Number   0 1 2 3 4 5 6 7

Cache

Direct
Mapped
only into
block 4
(12 mod 8)

block 12
can be placed

Placement Policy

Block Number

```
                        1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Memory

Set Number   0 1 2 3 4 5 6 7   0   1   2   3

Cache

Direct
Mapped
only into
block 4
(12 mod 8)
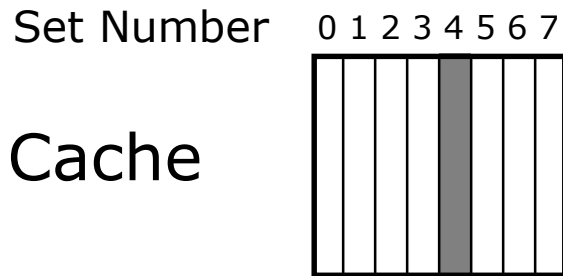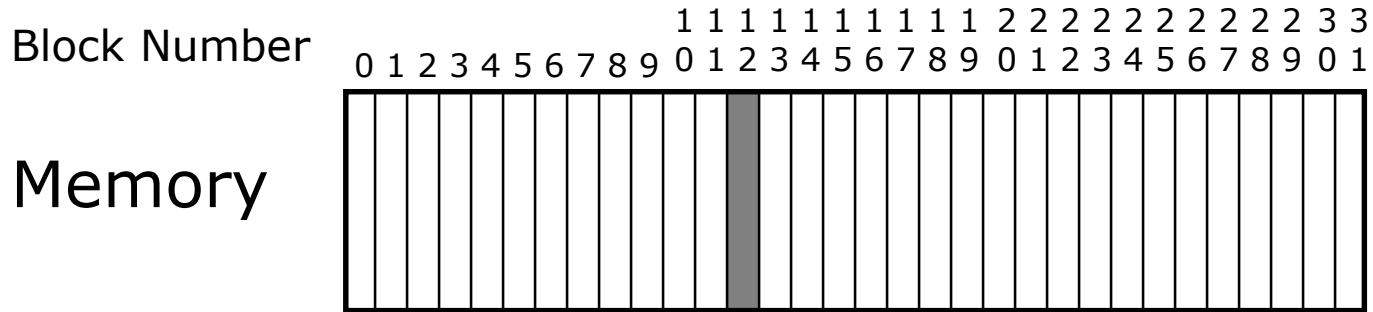
(2-way) Set
Associative
anywhere in
set 0
(12 mod 4)

block 12
can be placed

Placement Policy

Block Number

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Memory

Set Number   0 1 2 3 4 5 6 7   0   1   2   3
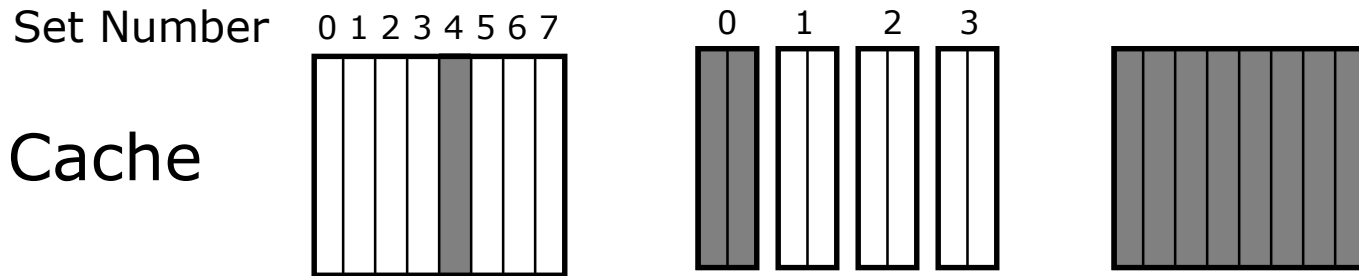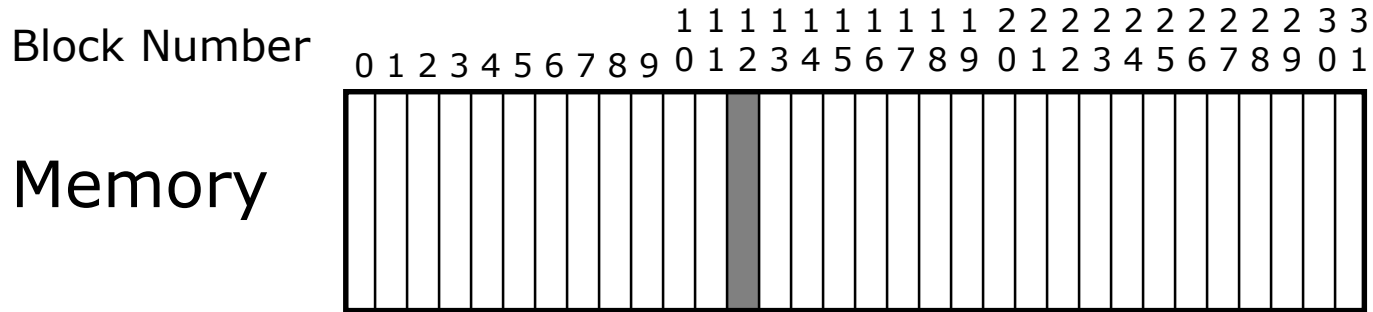
Cache

Direct Mapped
only into block 4
(12 mod 8)

(2-way) Set Associative
anywhere in set 0
(12 mod 4)

Fully Associative
anywhere

block 12 can be placed

Improving Cache Performance

Average memory access time (AMAT) =
            Hit time + Miss rate x Miss penalty

To improve performance:
- reduce the hit time
- reduce the miss rate (e.g., larger, better policy)
- reduce the miss penalty (e.g., L2 cache)

What is the simplest design strategy?

Improving Cache Performance

Average memory access time (AMAT) =
        Hit time + Miss rate x Miss penalty

To improve performance:
- reduce the hit time
- reduce the miss rate (e.g., larger, better policy)
- reduce the miss penalty (e.g., L2 cache)

What is the simplest design strategy?

Biggest cache that doesn't increase hit time past 1-2 cycles (approx. 16-64KB in modern technology)

[design issues more complex with out-of-order superscalar processors]

Causes for Cache Misses [Hill, 1989]

- *Compulsory:*
  First reference to a block *a.k.a.* cold start misses
  - misses that would occur even with infinite cache

- *Capacity:*
  cache is too small to hold all data the program needs
  - misses that would occur even under perfect placement & replacement policy

- *Conflict:*
  misses from collisions due to block-placement strategy
  - misses that would not occur with full associativity

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses			
Capacity misses			
Conflict misses			
Hit latency			
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=		
Capacity misses			
Conflict misses			
Hit latency			
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=		
Capacity misses	⬇		
Conflict misses			
Hit latency			
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=		
Capacity misses	⬇		
Conflict misses	⬇		
Hit latency			
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=		
Capacity misses	⬇		
Conflict misses	⬇		
Hit latency	⬆		
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=		
Capacity misses	↓		
Conflict misses	↓		
Hit latency	↑		
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	
Capacity misses	⬇		
Conflict misses	⬇		
Hit latency	⬆		
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	
Capacity misses	⬇	=	
Conflict misses	⬇		
Hit latency	⬆		
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	
Capacity misses	↓	=	
Conflict misses	↓	↓	
Hit latency	↑		
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	
Capacity misses	↓	=	
Conflict misses	↓	↓	
Hit latency	↑	↑	
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=	=	
Capacity misses	⬇	=	
Conflict misses	⬇	⬇	
Hit latency	⬆	⬆	
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	⬇
Capacity misses	⬇	=	
Conflict misses	⬇	⬇	
Hit latency	⬆	⬆	
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=	=	↓
Capacity misses	↓	=	↓
Conflict misses	↓	↓	
Hit latency	↑	↑	
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	↓
Capacity misses	↓	=	↓
Conflict misses	↓	↓	?
Hit latency	↑	↑	
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=	=	↓
Capacity misses	↓	=	↓
Conflict misses	↓	↓	?
Hit latency	↑	↑	=
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=	=	⬇
Capacity misses	⬇	=	⬇
Conflict misses	⬇	⬇	?
Hit latency	⬆	⬆	=
Miss latency	=	=	⬆⬇

* Assume substantial spatial locality

Block-level Optimizations

- Tags are too large, i.e., too much overhead
  - Simple solution: Larger blocks, but miss penalty could be large.

Block-level Optimizations

- Tags are too large, i.e., too much overhead
  - Simple solution: Larger blocks, but miss penalty could be large.

- Sub-block placement (aka sector cache)
  - A valid bit added to units smaller than the full block, called sub-blocks
  - Only read a sub-block on a miss
  - *If a tag matches, is the sub-block in the cache?*

100	*1*		*1*		*1*		*1*
300	*1*		*1*		*0*		*0*
204	*0*		*1*		*0*		*1*

Replacement Policy

Which block from a set should be evicted?

Replacement Policy

Which block from a set should be evicted?

- Random

Replacement Policy

Which block from a set should be evicted?

- Random

Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
    - LRU cache state must be updated on every access
    - true implementation only feasible for small sets (2-way)
    - pseudo-LRU binary tree was often used for 4-8 way

Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree was often used for 4-8 way

- First In, First Out (FIFO) a.k.a. Round-Robin
  - used in highly associative caches
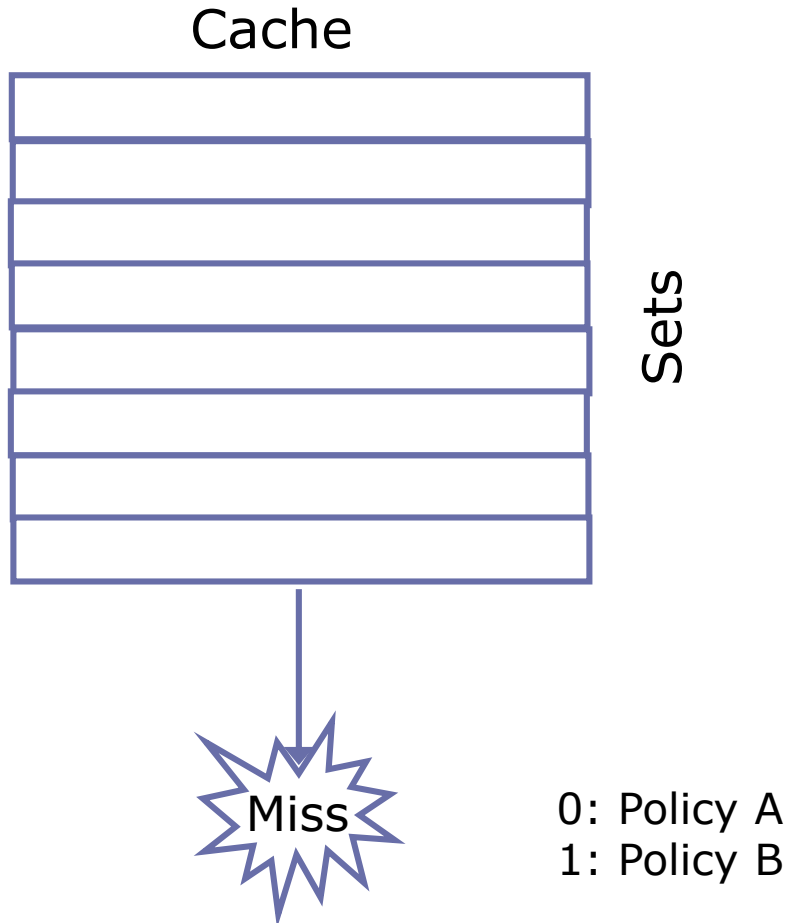
Replacement Policy

Which block from a set should be evicted?

• Random

• Least Recently Used (LRU)
  • LRU cache state must be updated on every access
  • true implementation only feasible for small sets (2-way)
  • pseudo-LRU binary tree was often used for 4-8 way

• First In, First Out (FIFO) a.k.a. Round-Robin
  • used in highly associative caches

• Not Least Recently Used (NLRU)
  • FIFO with exception for most recently used block or blocks

Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree was often used for 4-8 way

- First In, First Out (FIFO) a.k.a. Round-Robin
  - used in highly associative caches

- Not Least Recently Used (NLRU)
  - FIFO with exception for most recently used block or blocks

- One-bit LRU
  - Each way represented by a bit. Set on use, replace first unused.
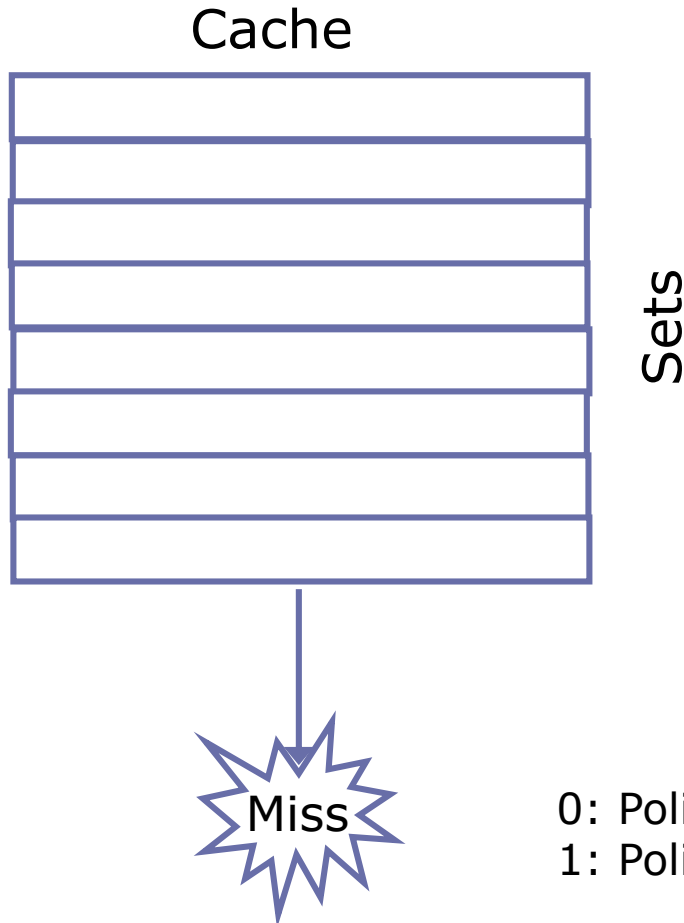
Multiple replacement policies
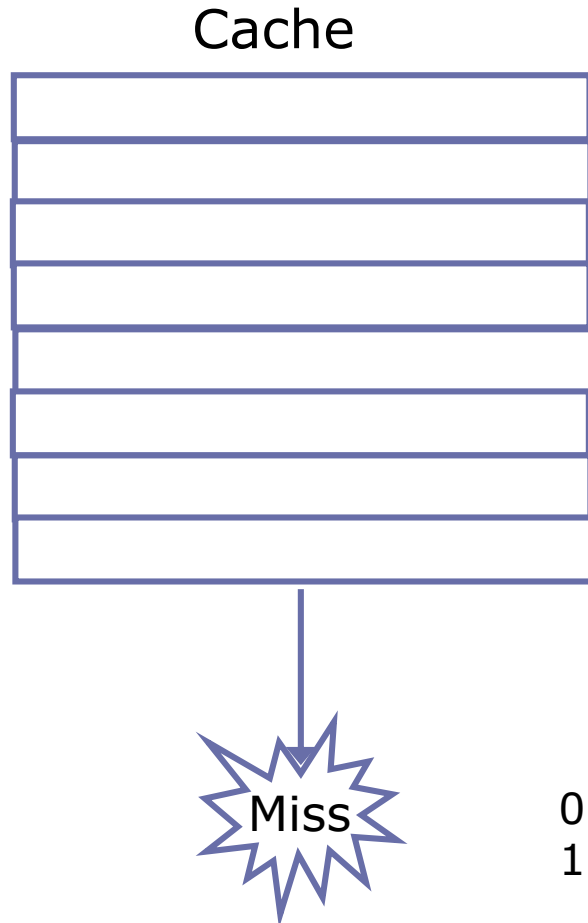
Use the best replacement policy for a program

Cache

Sets

Miss

0: Policy A
1: Policy B

Multiple replacement policies

Use the best replacement policy for a program
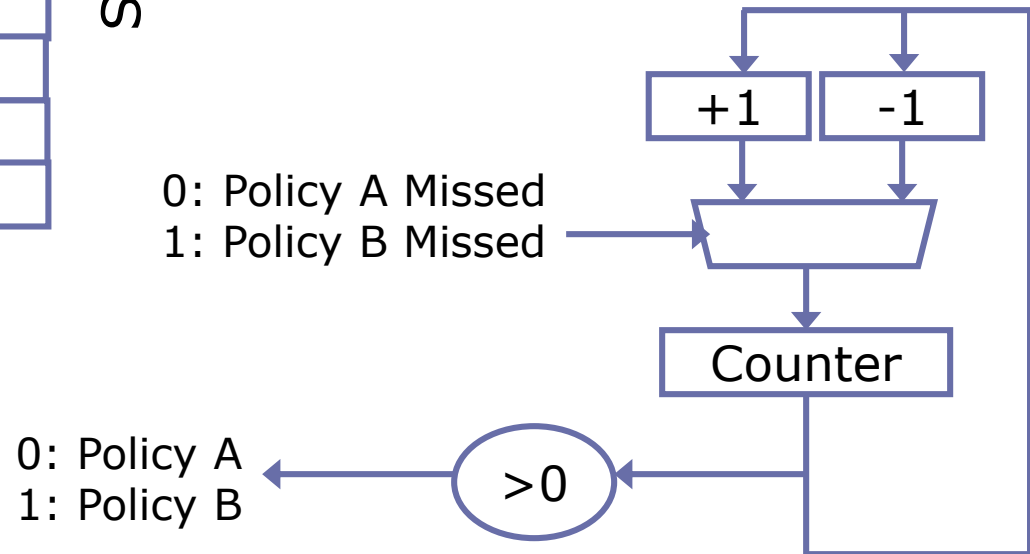
Cache

Sets

How do we decide which policy to use?

Miss

0: Policy A
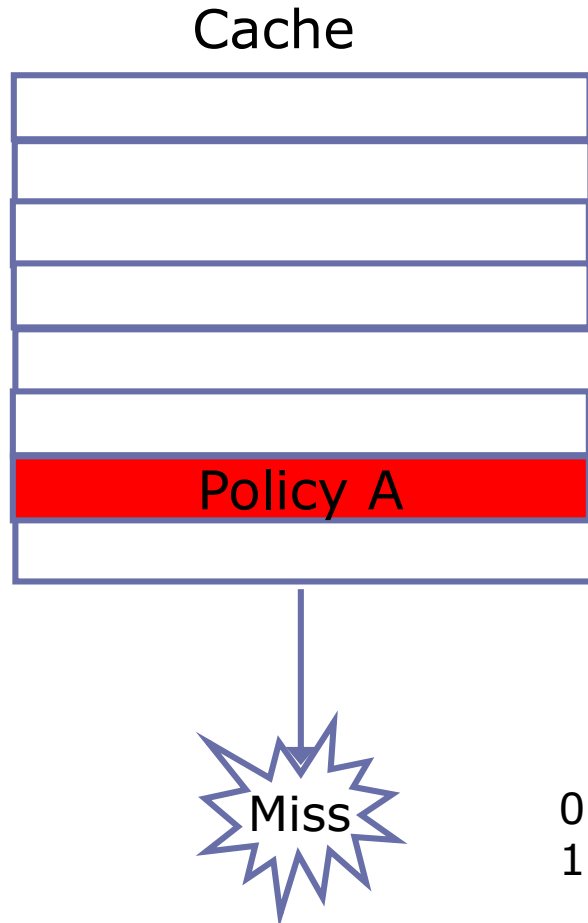1: Policy B

Multiple replacement policies

Use the best replacement policy for a program

Cache

How do we decide
which policy to use?

Sets

+1   -1

0: Policy A Missed
1: Policy B Missed

Counter

Miss

0: Policy A
1: Policy B   >0
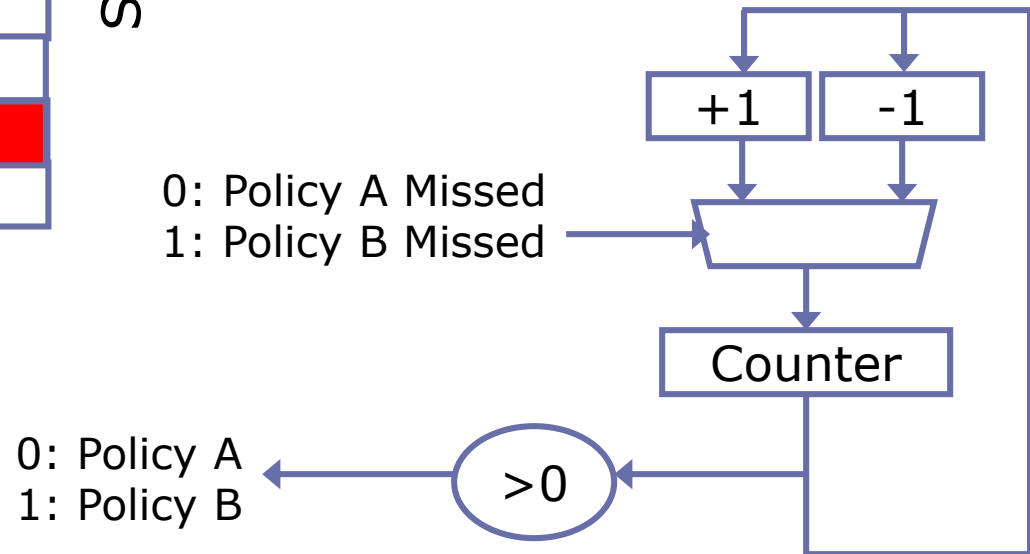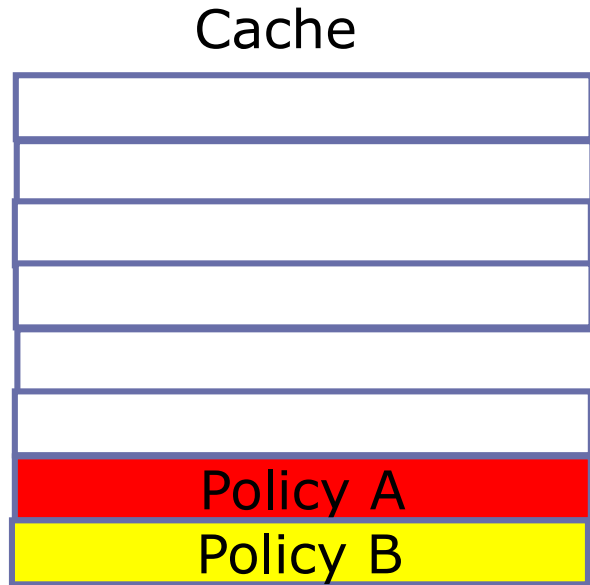
Multiple replacement policies

Use the best replacement policy for a program

Cache

How do we decide
which policy to use?

Sets

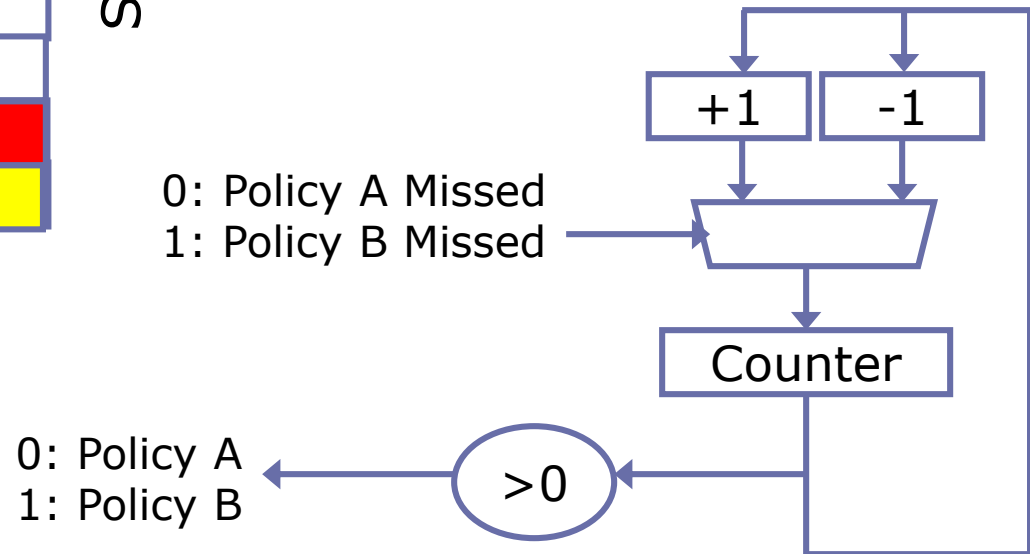Policy A

Miss

+1   -1

0: Policy A Missed
1: Policy B Missed

Counter

0: Policy A
1: Policy B

>0

Multiple replacement policies

Use the best replacement policy for a program

Cache

Sets

Policy A

Policy B

Miss

How do we decide
which policy to use?
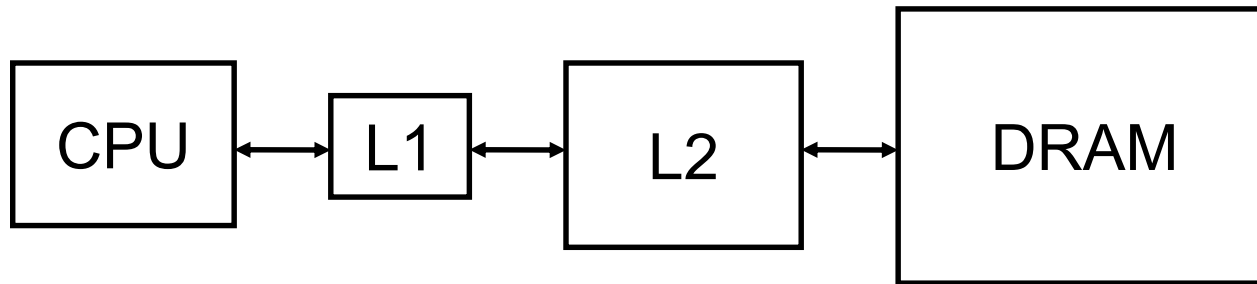
+1   -1

0: Policy A Missed
1: Policy B Missed

Counter

0: Policy A
1: Policy B

>0

Multilevel Caches

- A memory cannot be large and fast
- Add level of cache to reduce miss penalty
  - Each level can have longer latency than level above
  - So, increase sizes of cache at each level

```
CPU <--> L1 <--> L2 <--> DRAM
```

Metrics:

Local miss rate = misses in cache/ accesses to cache

Global miss rate = misses in cache / CPU memory accesses

Misses per instruction (MPI) = misses in cache / number of instructions

Inclusion Policy

- Inclusive multilevel cache:
  - Inner cache holds copies of data in outer cache
  - On miss, line inserted in inner and outer cache; replacement in outer cache invalidates line in inner cache
  - External accesses need only check outer cache
  - Commonly used (e.g., Intel CPUs up to Broadwell)

- Non-inclusive multilevel caches:
  - Inner cache may hold data not in outer cache
  - Replacement in outer cache doesn't invalidate line in inner cache
  - Used in Intel Skylake, ARM

- Exclusive multilevel caches:
  - Inner cache and outer cache hold different data
 - Swap lines between inner/outer caches on miss
  - Used in AMD processors

Inclusion Policy

- Inclusive multilevel cache:
  - Inner cache holds copies of data in outer cache
  - On miss, line inserted in inner and outer cache; replacement in outer cache invalidates line in inner cache
  - External accesses need only check outer cache
  - Commonly used (e.g., Intel CPUs up to Broadwell)

- Non-inclusive multilevel caches:
  - Inner cache may hold data not in outer cache
  - Replacement in outer cache doesn't invalidate line in inner cache
  - Used in Intel Skylake, ARM

- Exclusive multilevel caches:
  - Inner cache and outer cache hold different data
  - Swap lines between inner/outer caches on miss
  - Used in AMD processors

Why choose one type or the other?
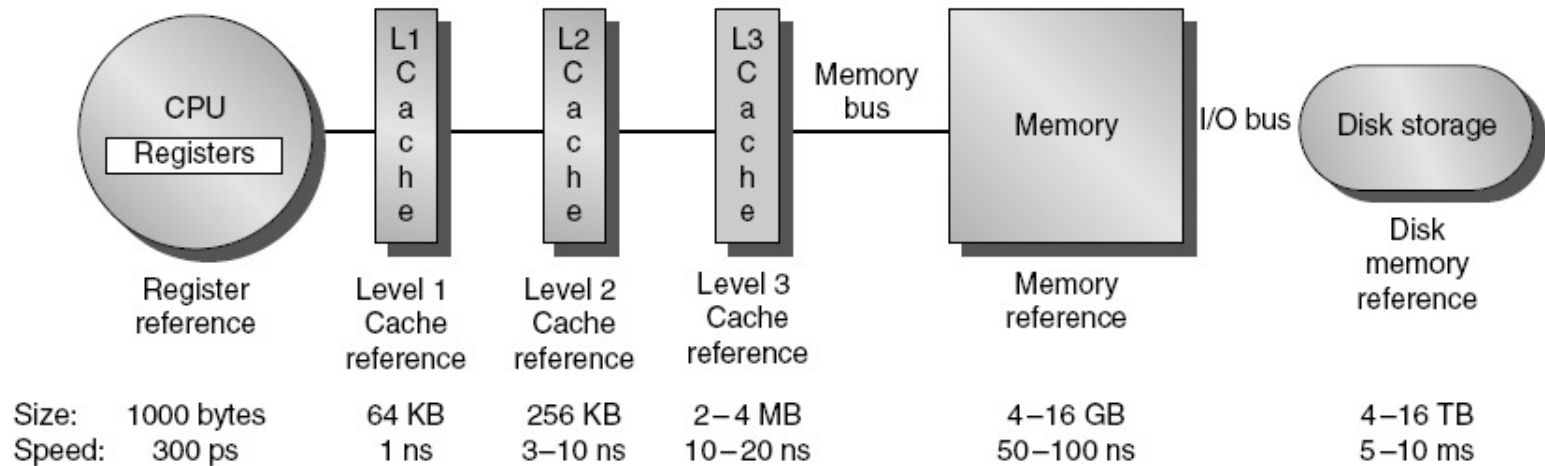
Victim Caches (HP 7200)
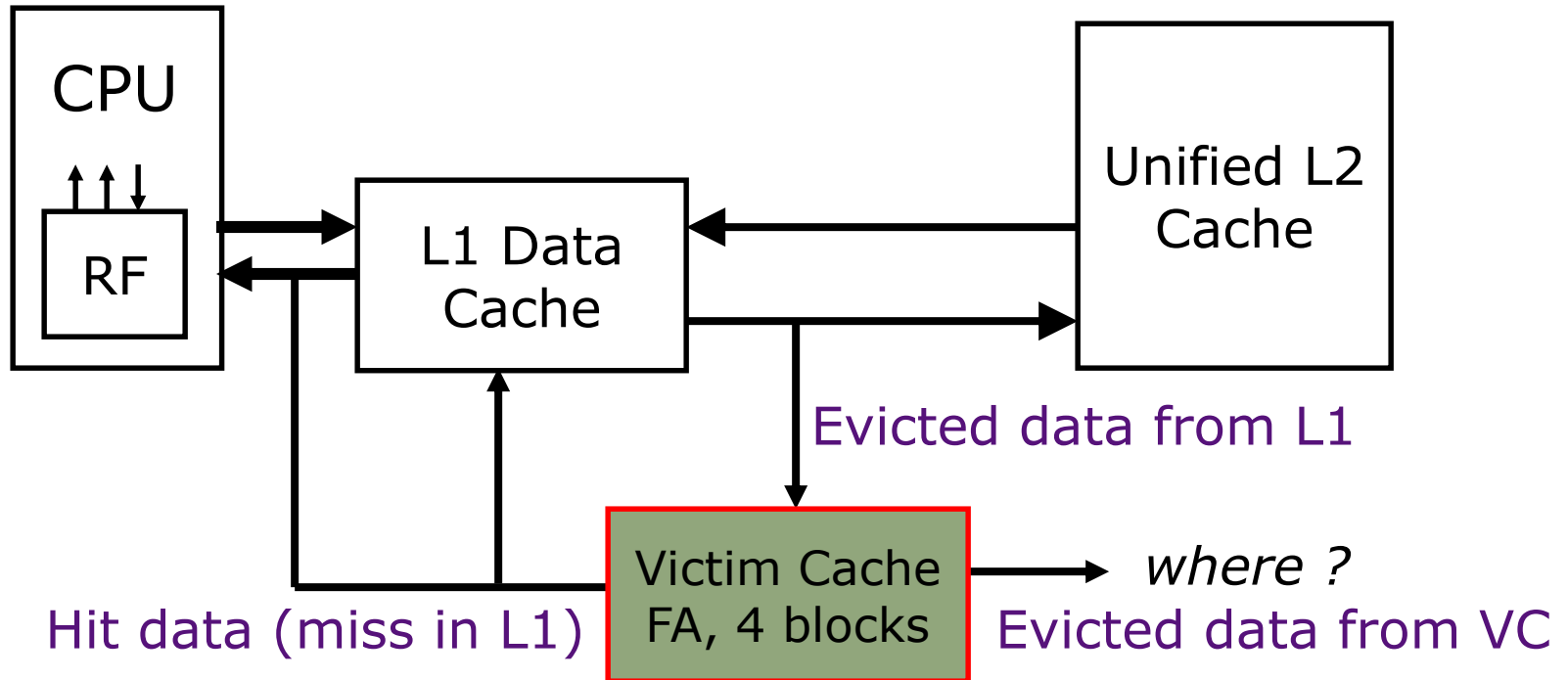
Victim cache is a small associative back up cache, added to a direct mapped cache, which holds recently evicted lines
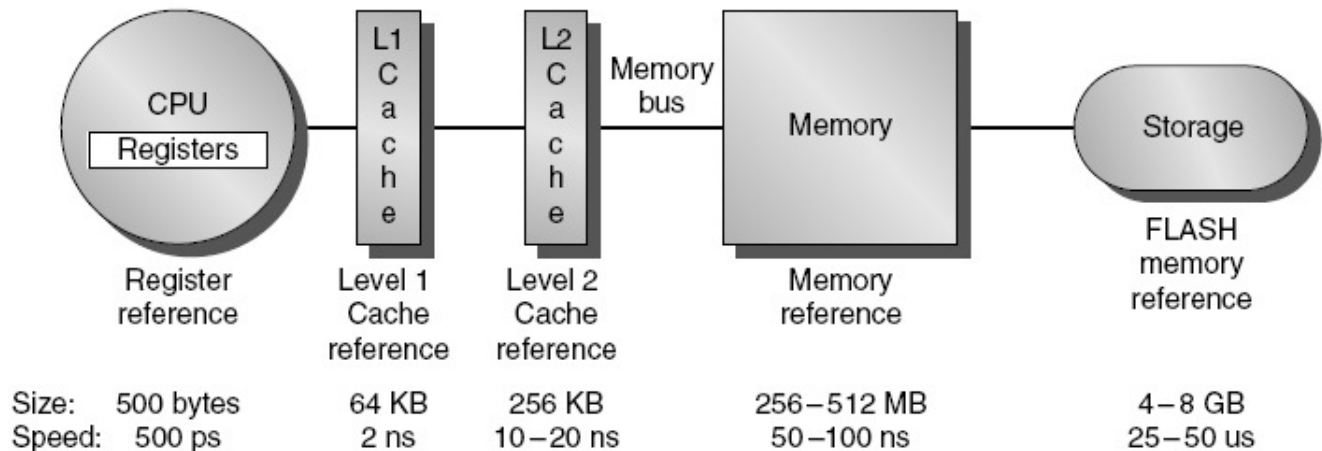- First look up in direct mapped cache
- If miss, look in victim cache
- If hit in victim cache, swap hit line with line now evicted from L1
- If miss in victim cache, L1 victim -> VC, VC victim->?

Fast hit time of direct mapped but with reduced conflict misses

Typical memory hierarchies

(a) Memory hierarchy for server

| Size: | 1000 bytes | 64 KB | 256 KB | 2–4 MB | 4–16 GB | 4–16 TB |
| Speed: | 300 ps | 1 ns | 3–10 ns | 10–20 ns | 50–100 ns | 5–10 ms |

(b) Memory hierarchy for a personal mobile device

| Size: | 500 bytes | 64 KB | 256 KB | 256–512 MB | 4–8 GB |
| Speed: | 500 ps | 2 ns | 10–20 ns | 50–100 ns | 25–50 us |

Memory Management:
From Absolute Addresses to Demand Paging

Daniel Sanchez
Computer Science and Artificial Intelligence Laboratory
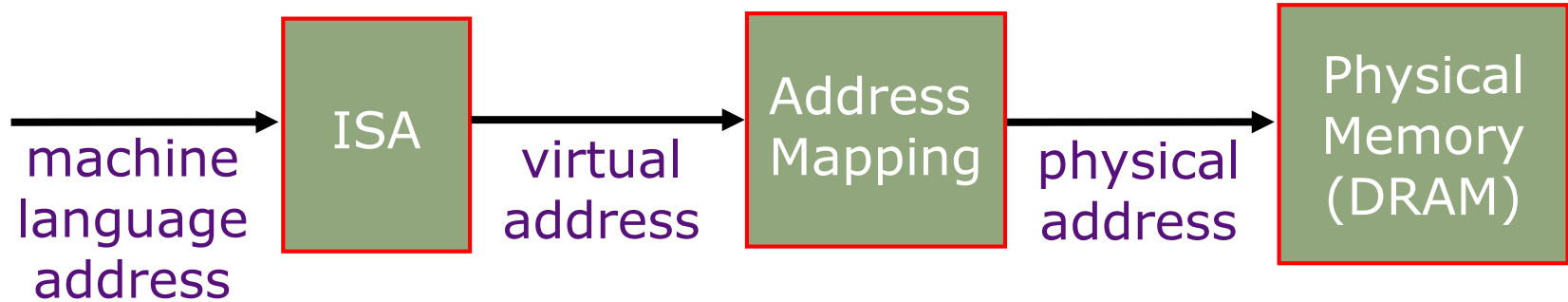M.I.T.

Memory Management

- The Fifties
  - Absolute Addresses
  - Dynamic address translation

- The Sixties
  - Atlas and Demand Paging
  - Paged memory systems and TLBs

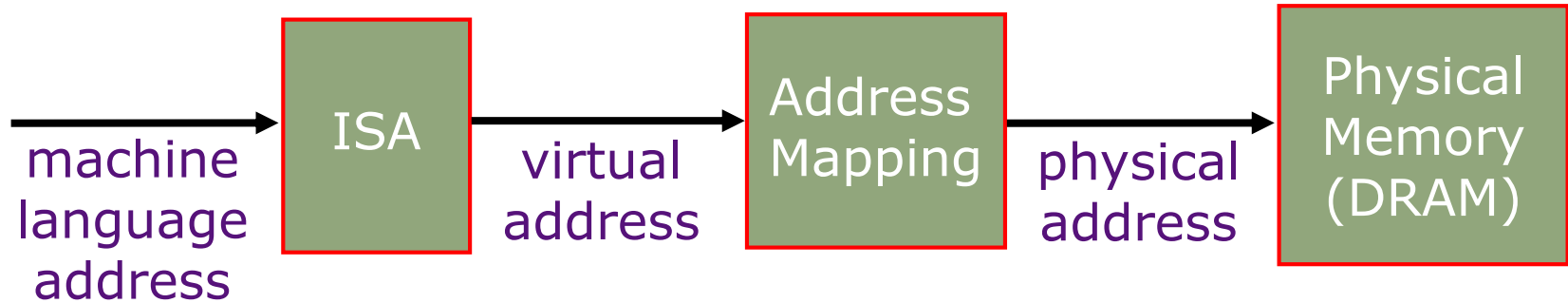- Modern Virtual Memory Systems

Names for Memory Locations

machine language address → **ISA** → virtual address → **Address Mapping** → physical address → **Physical Memory (DRAM)**

Names for Memory Locations

machine
language
address → **ISA** → virtual
address → **Address Mapping** → physical
address → **Physical Memory (DRAM)**

- ## Machine language address
  - – as specified in machine code

- ## Virtual address
  - – ISA specifies translation of machine code address into virtual address of program variable (sometimes called *effective* address)

- ## Physical address
  - ⇒ operating system specifies mapping of virtual address into name for a physical memory location

Absolute Addresses

EDSAC, early 50's

virtual address = physical memory address

Absolute Addresses

EDSAC, early 50's

virtual address = physical memory address

- Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)

Absolute Addresses

EDSAC, early 50's

virtual address = physical memory address

- Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)
- Addresses in a program depended upon where the program was to be loaded in memory

Absolute Addresses

EDSAC, early 50's

virtual address = physical memory address

- Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)
- Addresses in a program depended upon where the program was to be loaded in memory
- *But* it was more convenient for programmers to write location-independent subroutines

Absolute Addresses

EDSAC, early 50's

virtual address = physical memory address

- Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)
- Addresses in a program depended upon where the program was to be loaded in memory
- *But* it was more convenient for programmers to write location-independent subroutines

How could location independence be achieved?

Absolute Addresses

EDSAC, early 50's

> virtual address = physical memory address

- Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)
- Addresses in a program depended upon where the program was to be loaded in memory
- *But* it was more convenient for programmers to write location-independent subroutines

How could location independence be achieved?

Linker and/or loader modify addresses of subroutines and callers when building a program memory image

Multiprogramming

Motivation

In the early machines, I/O operations were slow and each word transferred involved the CPU

Higher throughput if CPU and I/O of 2 or more programs were overlapped. *How?*
  ⇒ *multiprogramming*

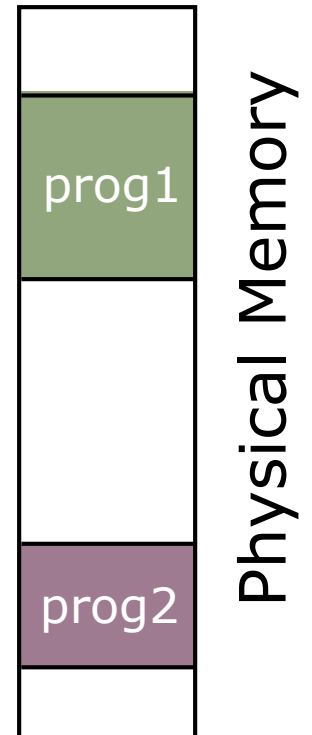## Location-independent programs

Programming and storage management ease
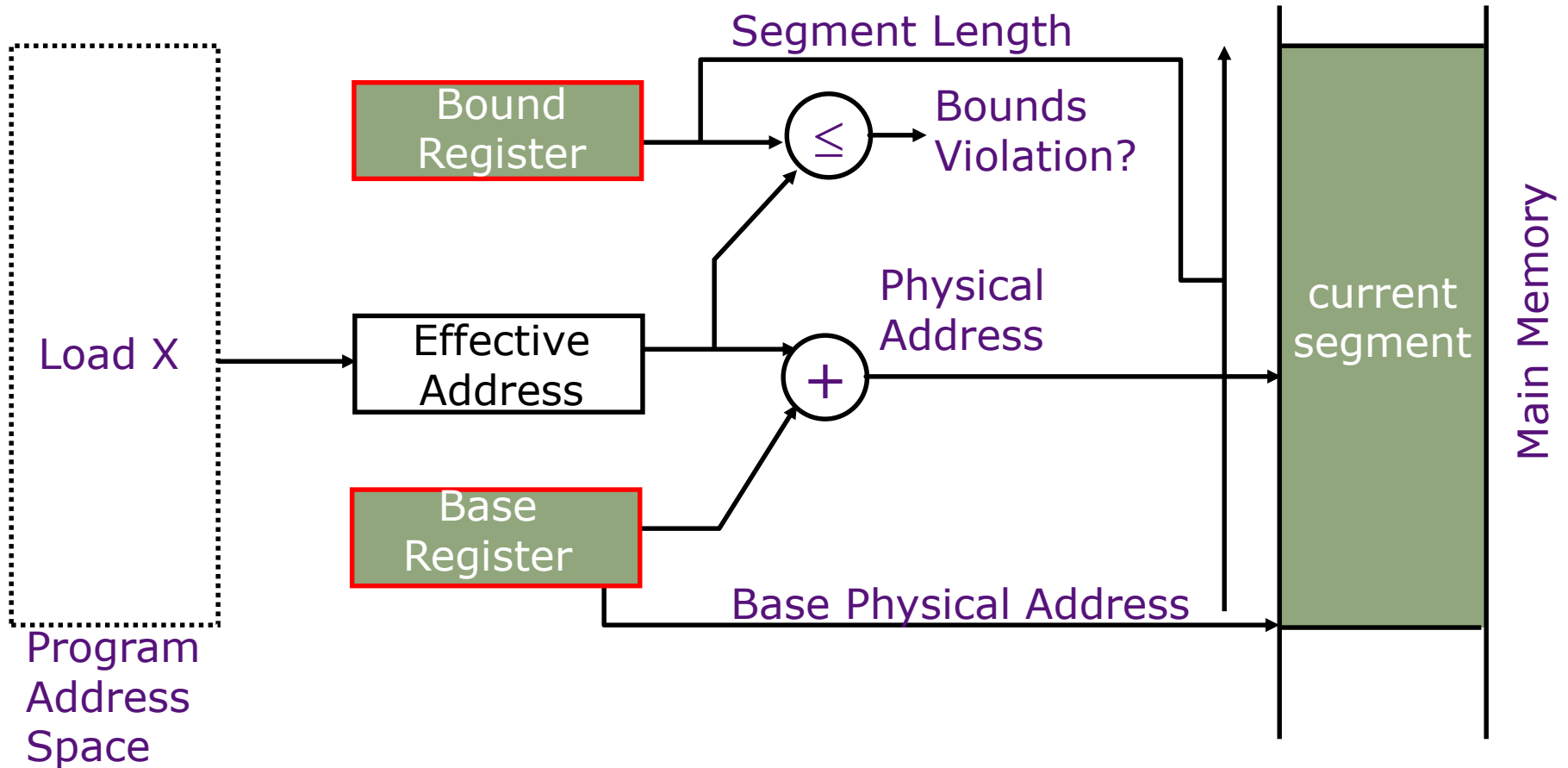 ⇒ need for a *base register*

Protection

Independent programs should not affect each other inadvertently

  ⇒ need for a *bound register*

Physical Memory

prog1

prog2

Simple Base and Bound Translation

Segment Length

Bound Register → ≤ → Bounds Violation?

Load X → Effective Address

Physical Address

+

Base Register

Base Physical Address

current segment

Main Memory

Program Address Space

Base and bounds registers are visible/accessible only when processor is running in *supervisor mode*
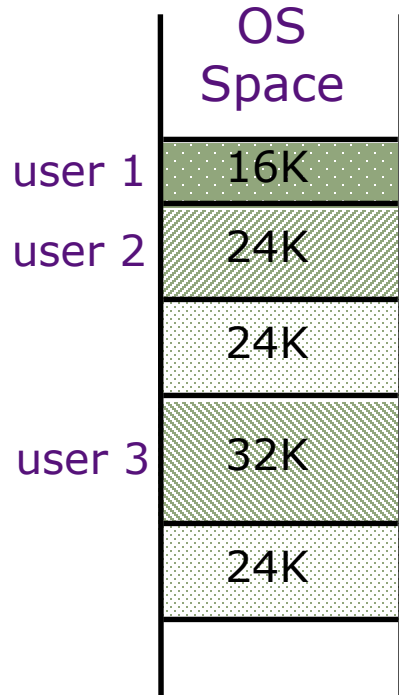
Separate Areas for Code and Data
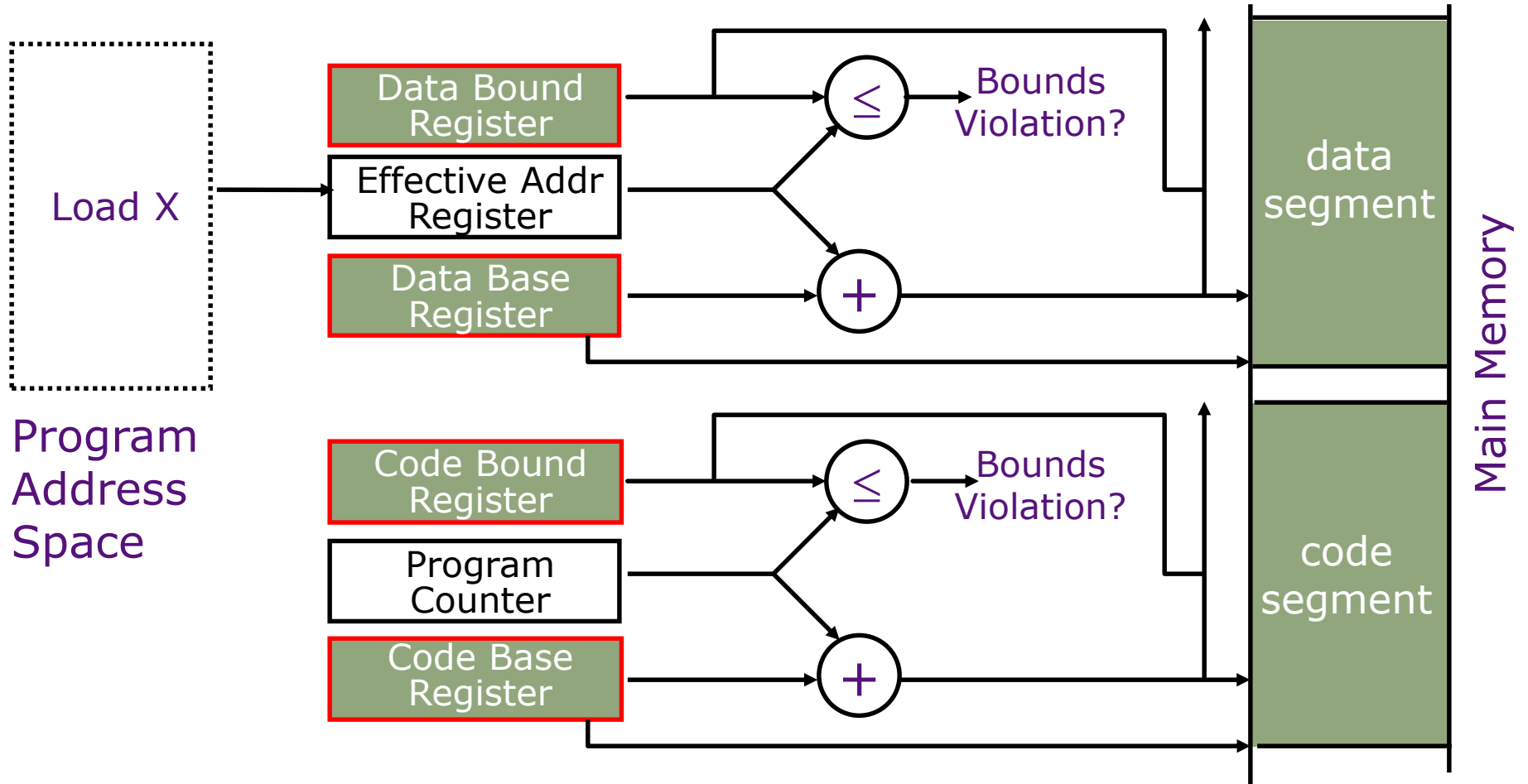
What is an advantage of this separation?
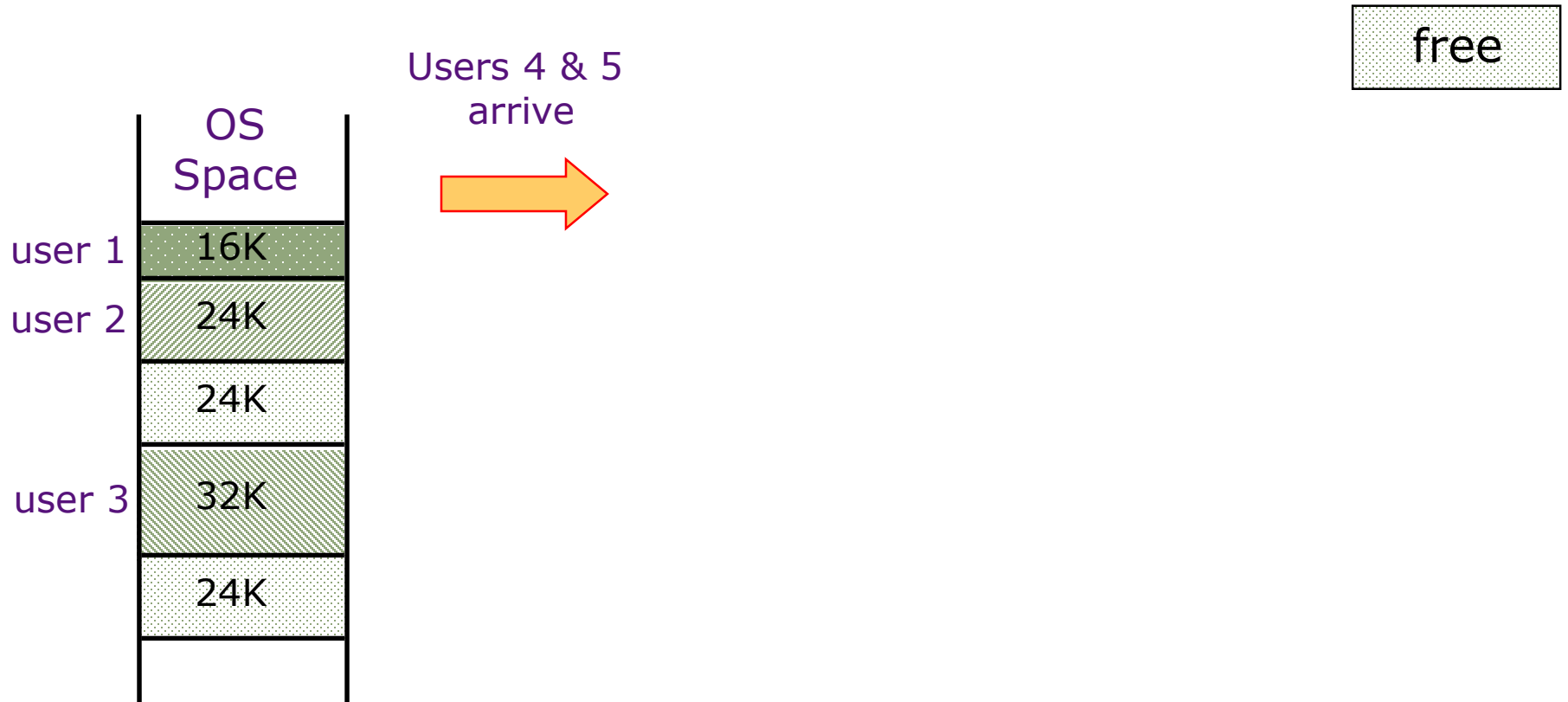(Scheme used on all Cray vector supercomputers prior to X1, 2002)

Memory Fragmentation

free

OS Space

user 1 — 16K
user 2 — 24K
24K
user 3 — 32K
24K

Memory Fragmentation

free

Users 4 & 5
arrive

➡

OS
Space

user 1 | 16K

user 2 | 24K

24K

user 3 | 32K

24K

Memory Fragmentation

free

Users 4 & 5
arrive

OS Space
user 1 — 16K
user 2 — 24K
24K
user 3 — 32K
24K

➡

OS Space
user 1 — 16K
user 2 — 24K
user 4 — 16K
8K
user 3 — 32K
user 5 — 24K

Memory Fragmentation

Users 4 & 5
arrive

Users 2 & 5
leave

free

OS Space
user 1 — 16K
user 2 — 24K
24K
user 3 — 32K
24K

OS Space
user 1 — 16K
user 2 — 24K
user 4 — 16K
8K
user 3 — 32K
user 5 — 24K

Memory Fragmentation

Users 4 & 5 arrive

Users 2 & 5 leave

free

OS Space

user 1	16K
user 2	24K
	24K
user 3	32K
	24K

OS Space

user 1	16K
user 2	24K
user 4	16K
	8K
user 3	32K
user 5	24K

OS Space

user 1	16K
	24K
user 4	16K
	8K
user 3	32K
	24K

Memory Fragmentation

Users 4 & 5
arrive

Users 2 & 5
leave

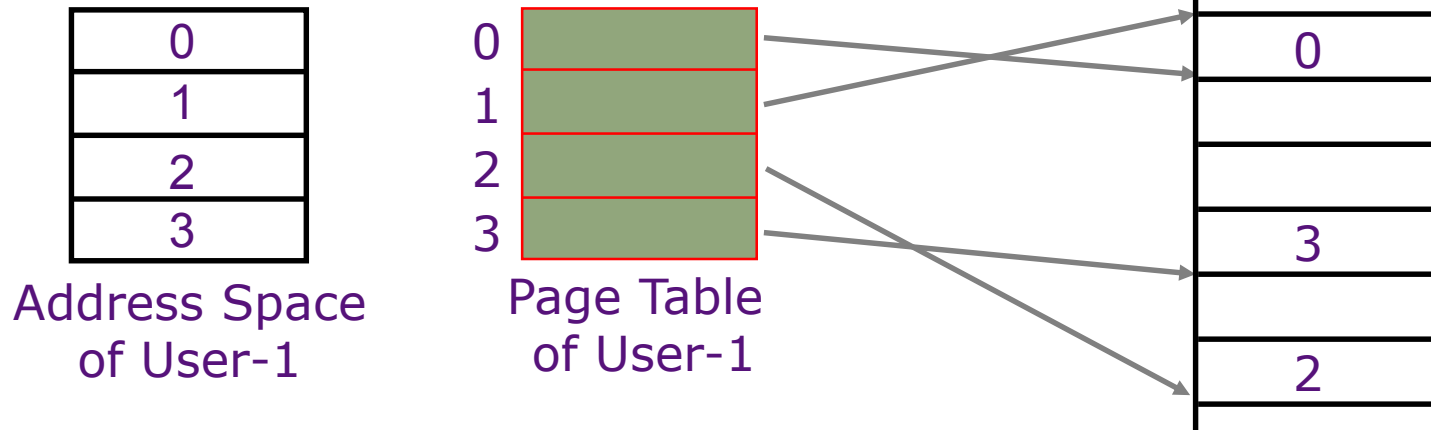| OS Space	OS Space	OS Space
user 1 — 16K	user 1 — 16K	user 1 — 16K
user 2 — 24K	user 2 — 24K	24K
24K	user 4 — 16K	user 4 — 16K
	8K	8K
user 3 — 32K	user 3 — 32K	user 3 — 32K
24K	user 5 — 24K	24K

As users come and go, the storage is "fragmented". Therefore, at some stage programs have to be moved around to compact the storage.

Paged Memory Systems

- Processor-generated address can be interpreted as a pair <page number, offset>

page number	offset

- A page table contains the physical address of the base of each page

0		0		1
1		1		0
2		2		
3		3		

Address Space of User-1   Page Table of User-1
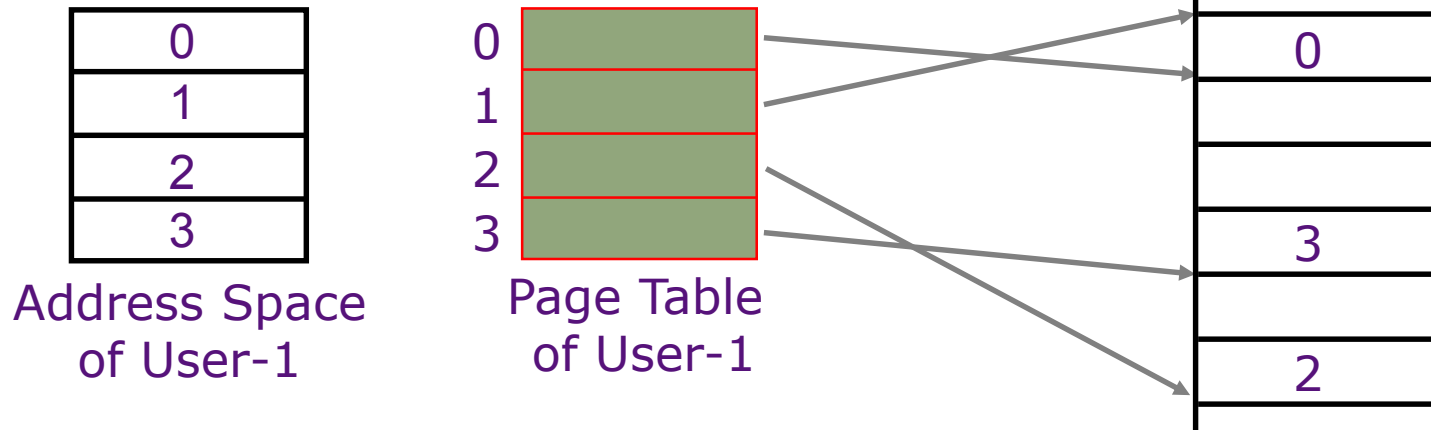
Paged Memory Systems

- Processor-generated address can be interpreted as a pair <page number, offset>

page number	offset

- A page table contains the physical address of the base of each page

Address Space of User-1

Page Table of User-1

Page tables make it possible to store the pages of a program non-contiguously.

Private Address Space per User
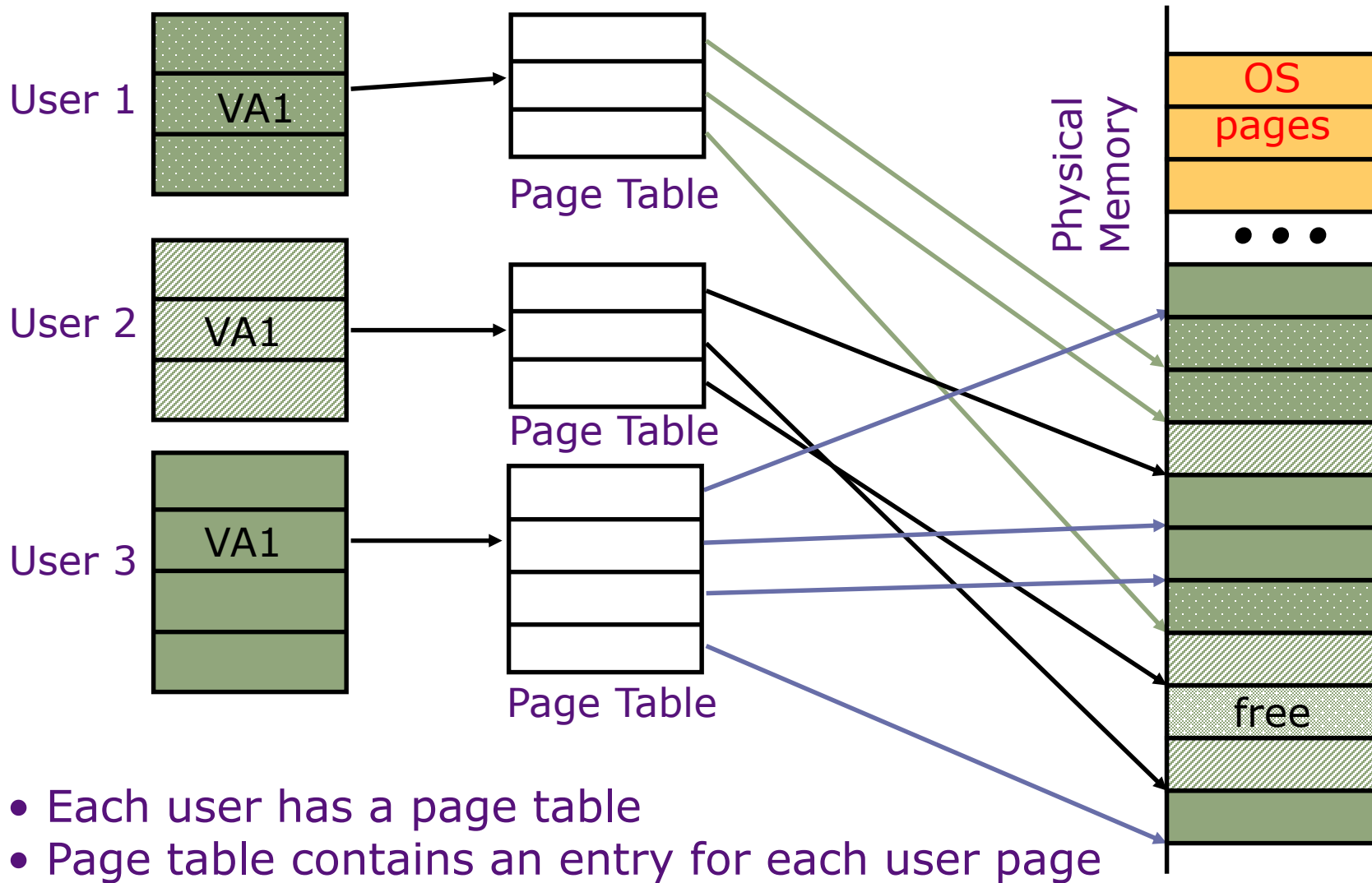
- Each user has a page table
- Page table contains an entry for each user page

Where Should Page Tables Reside?

- Space required by the page tables (PT) is proportional to the address space, number of users, …

    $\Rightarrow$ Space requirement is large

    $\Rightarrow$ Too expensive to keep in registers

Where Should Page Tables Reside?

- Space required by the page tables (PT) is proportional to the address space, number of users, ...

  $\Rightarrow$ Space requirement is large

  $\Rightarrow$ Too expensive to keep in registers

- Idea: Keep PT of the current user in special registers

  – may not be feasible for large page tables

  – Increases the cost of context swap

Where Should Page Tables Reside?

- Space required by the page tables (PT) is proportional to the address space, number of users, ...

  ⇒ Space requirement is large
  ⇒ Too expensive to keep in registers

- Idea: Keep PT of the current user in special registers

  – may not be feasible for large page tables
  – Increases the cost of context swap

- Idea: Keep PTs in the main memory

  – needs one reference to retrieve the page base address and another to access the data word
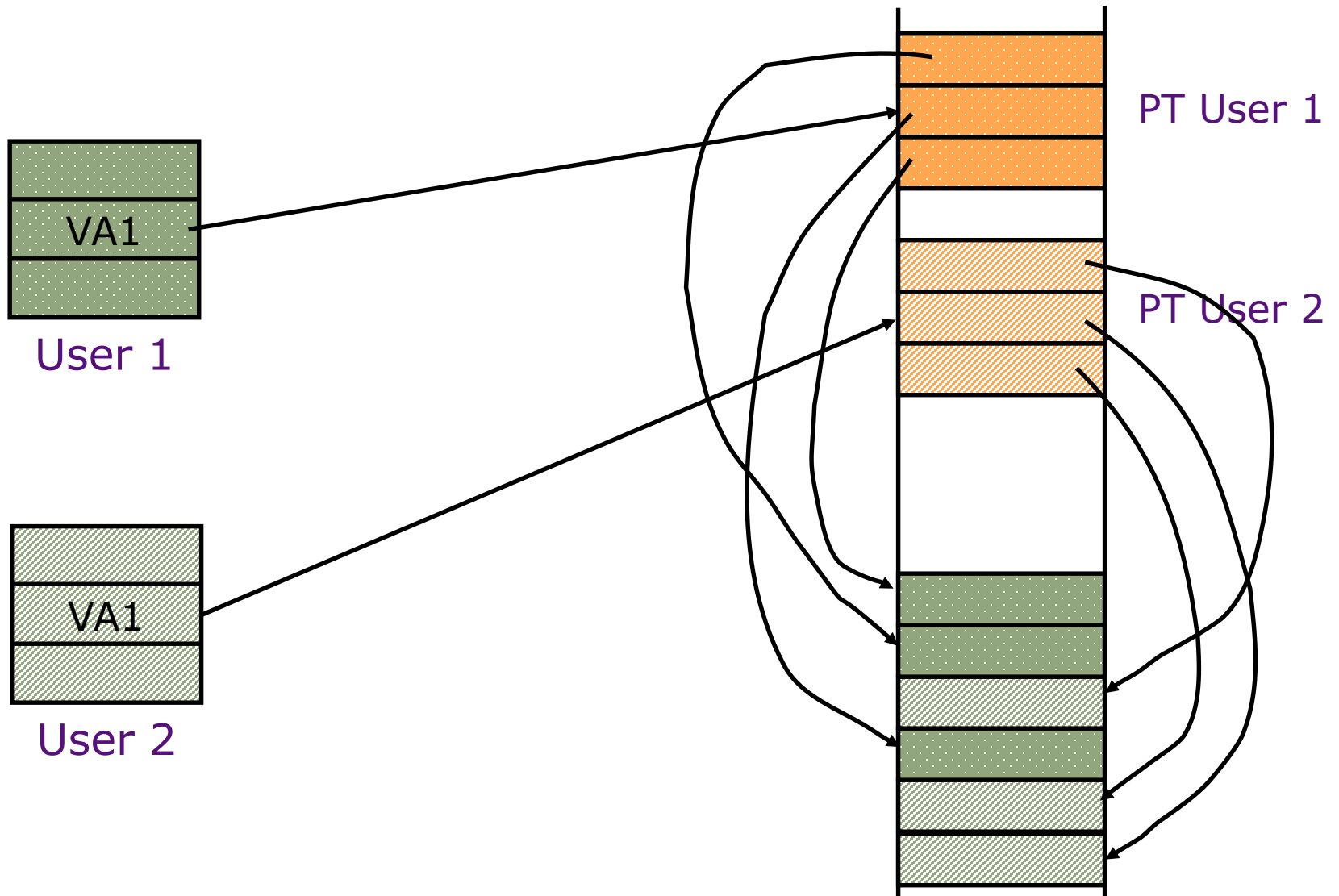
Where Should Page Tables Reside?

- Space required by the page tables (PT) is proportional to the address space, number of users, ...

  $\Rightarrow$ Space requirement is large
  $\Rightarrow$ Too expensive to keep in registers

- Idea: Keep PT of the current user in special registers

  – may not be feasible for large page tables

  – Increases the cost of context swap

- Idea: Keep PTs in the main memory

  – needs one reference to retrieve the page base address and another to access the data word
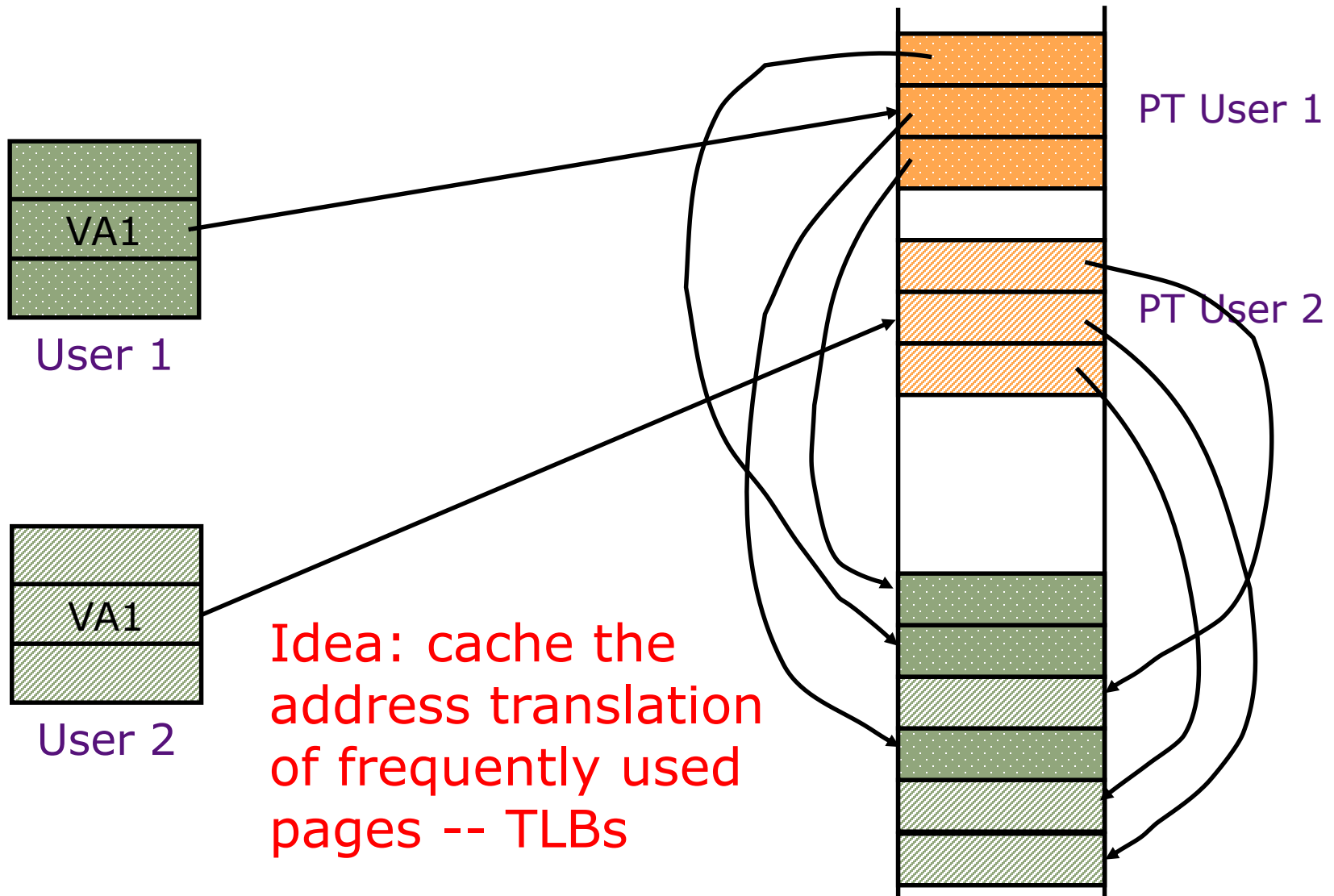
  $\Rightarrow$ *doubles the number of memory references!*

Page Tables in Physical Memory

VA1

User 1

VA1

User 2

PT User 1

PT User 2

Page Tables in Physical Memory

VA1

User 1

VA1

User 2

PT User 1

PT User 2

Idea: cache the address translation of frequently used pages -- TLBs
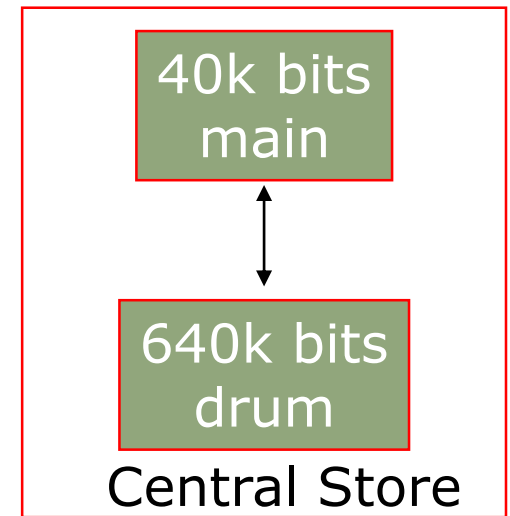
A Problem in Early Sixties

- There were many applications whose data could not fit in the main memory, e.g., payroll
  - *Paged memory system reduced fragmentation but still required the whole program to be resident in the main memory*

- Programmers moved the data back and forth from the secondary store by *overlaying* it repeatedly on the primary store
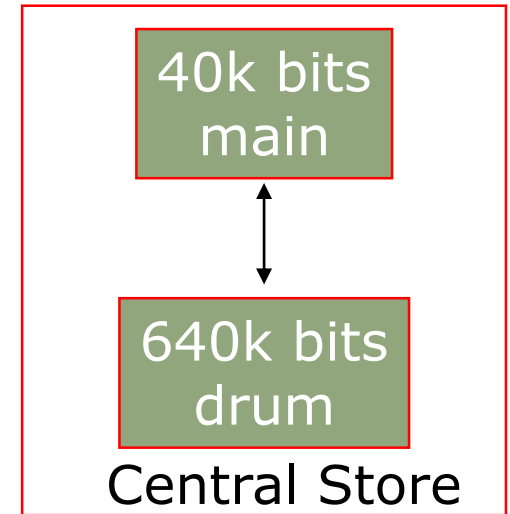
tricky programming!

Manual Overlays

40k bits main

640k bits drum

Central Store

Ferranti Mercury
1956

Manual Overlays

- Assume an instruction can address all the storage on the drum
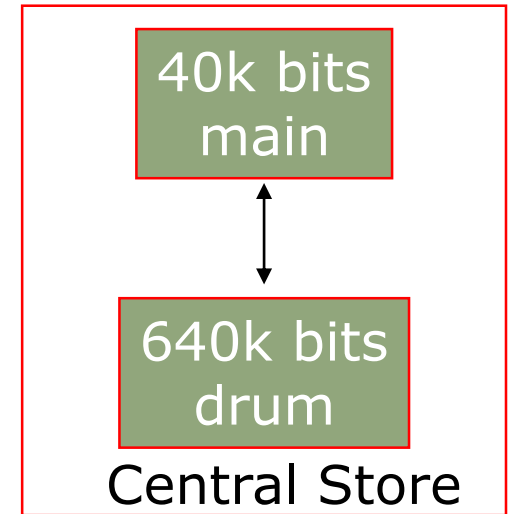

40k bits main

640k bits drum

Central Store

Ferranti Mercury 1956

Manual Overlays

- Assume an instruction can address all the storage on the drum

- *Method 1:* programmer keeps track of addresses in the main memory and initiates an I/O transfer when required

40k bits main

640k bits drum

Central Store

Ferranti Mercury 1956

Manual Overlays

- Assume an instruction can address all the storage on the drum

- *Method 1:* programmer keeps track of addresses in the main memory and initiates an I/O transfer when required
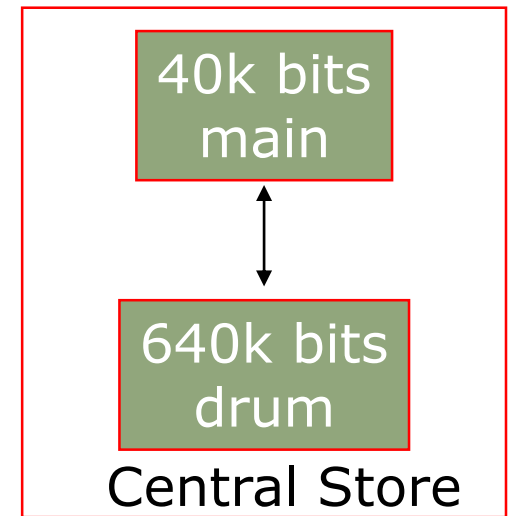
- *Method 2:* automatic initiation of I/O transfers by software address translation

40k bits main

640k bits drum

Central Store

Ferranti Mercury 1956

Manual Overlays

- Assume an instruction can address all the storage on the drum

- *Method 1:* programmer keeps track of addresses in the main memory and initiates an I/O transfer when required

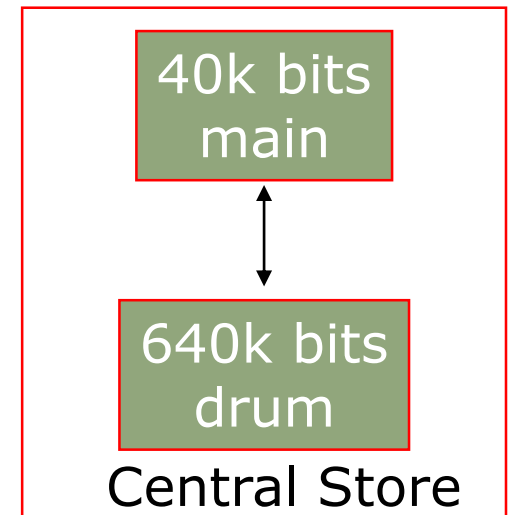- *Method 2:* automatic initiation of I/O transfers by software address translation

  Brooker's interpretive coding, 1960

40k bits main

↕

640k bits drum

Central Store

Ferranti Mercury 1956

Manual Overlays

- Assume an instruction can address all the storage on the drum

- *Method 1:* programmer keeps track of addresses in the main memory and initiates an I/O transfer when required

- *Method 2:* automatic initiation of I/O transfers by software address translation

  Brooker's interpretive coding, 1960

40k bits
main

↕

640k bits
drum

Central Store

Ferranti Mercury
1956

Problems?
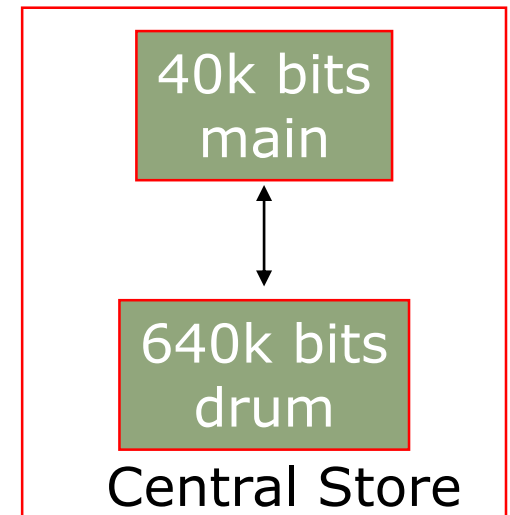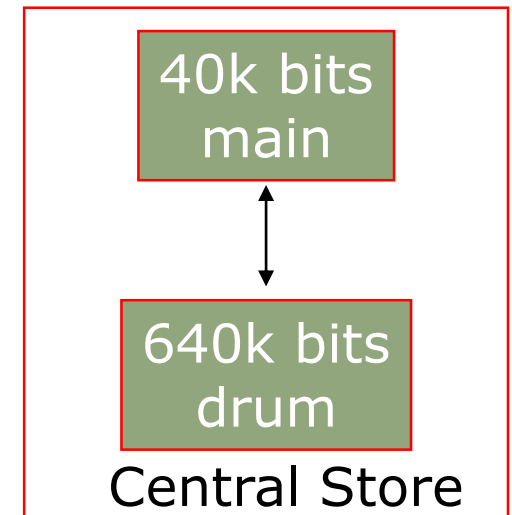
Manual Overlays

- Assume an instruction can address all the storage on the drum

- *Method 1:* programmer keeps track of addresses in the main memory and initiates an I/O transfer when required

- *Method 2:* automatic initiation of I/O transfers by software address translation

  Brooker's interpretive coding, 1960

Problems?

40k bits main

640k bits drum

Central Store

Ferranti Mercury 1956

Manual Overlays

- Assume an instruction can address all the storage on the drum

- *Method 1:* programmer keeps track of addresses in the main memory and initiates an I/O transfer when required

- *Method 2:* automatic initiation of I/O transfers by software address translation

  Brooker's interpretive coding, 1960

40k bits main

640k bits drum

Central Store

Ferranti Mercury 1956

Problems?   Method1: Difficult, error prone
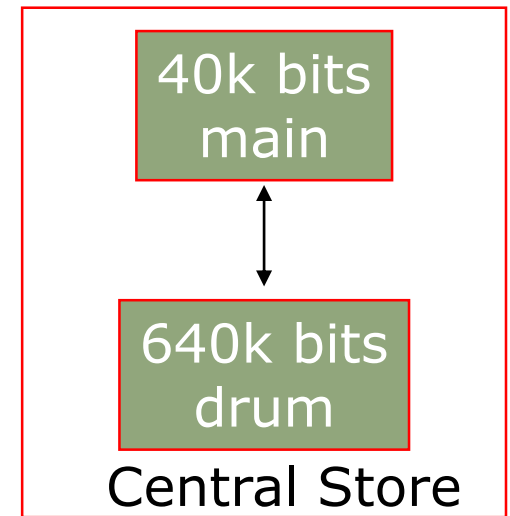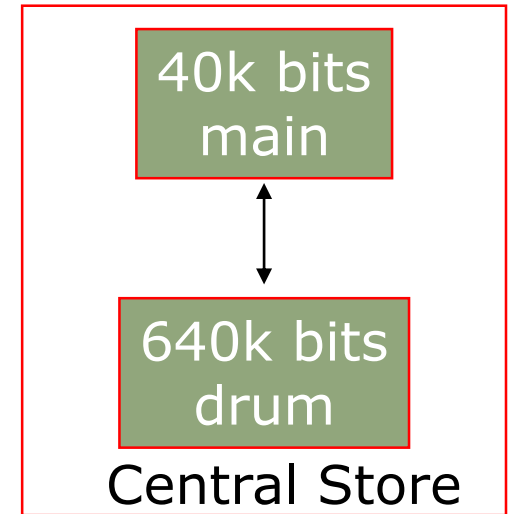
Manual Overlays

- Assume an instruction can address all the storage on the drum

- *Method 1:* programmer keeps track of addresses in the main memory and initiates an I/O transfer when required

- *Method 2:* automatic initiation of I/O transfers by software address translation

  Brooker's interpretive coding, 1960

40k bits main

640k bits drum

Central Store

Ferranti Mercury 1956

Problems?

Method1: Difficult, error prone
Method2: Inefficient

Demand Paging in Atlas (1962)

"A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor."

Tom Kilburn

Primary
32 Pages
512 words/page

Central
Memory

Secondary
(Drum)
32x6 pages

Demand Paging in Atlas (1962)

"A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor."
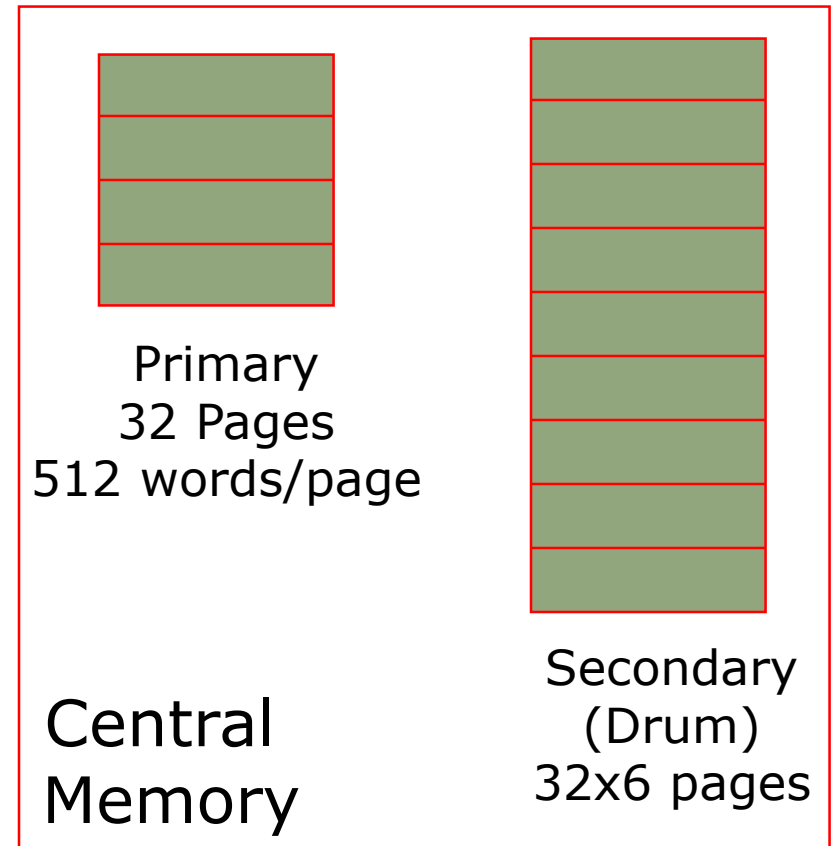
Tom Kilburn

Primary memory as a *cache* for secondary memory

Primary
32 Pages
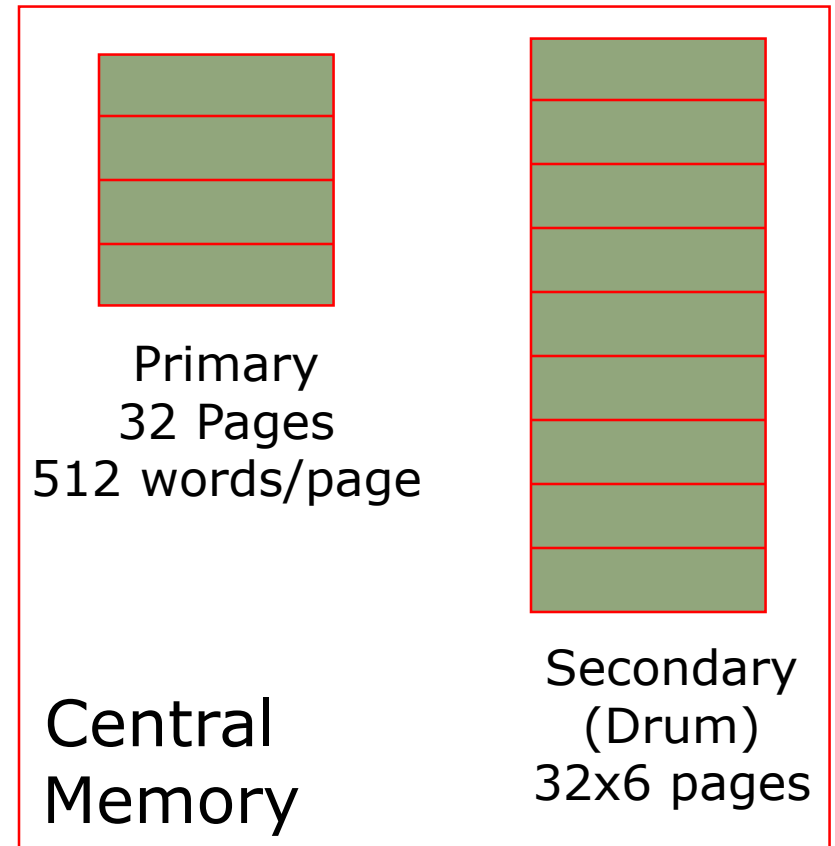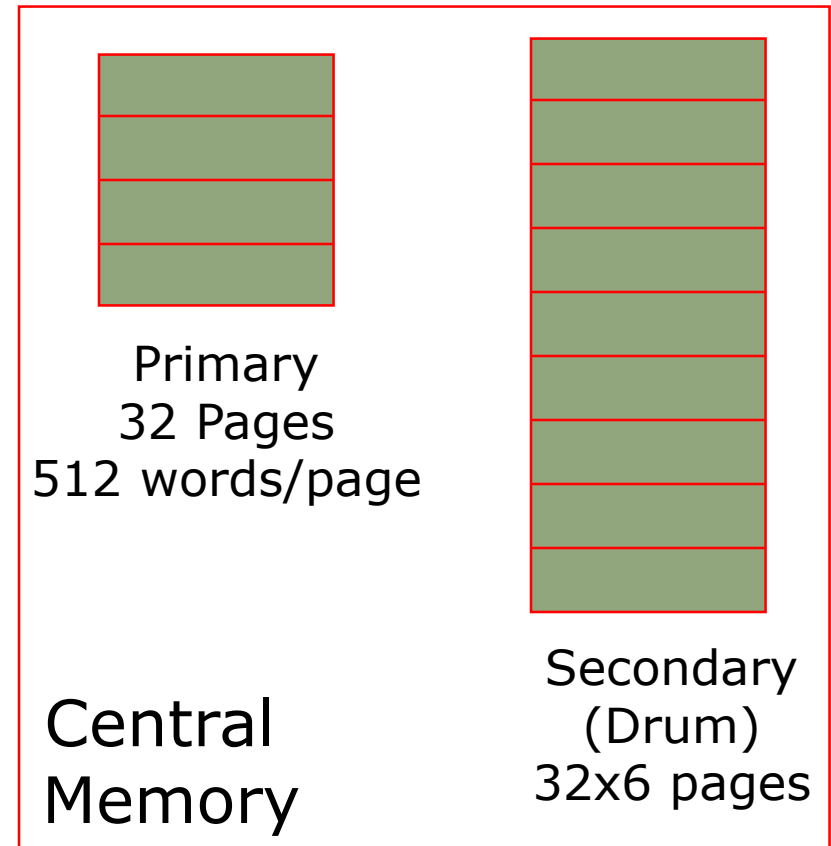512 words/page

Secondary
(Drum)
32x6 pages

Central
Memory

Demand Paging in Atlas (1962)

"A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor."

Tom Kilburn

Primary memory as a *cache* for secondary memory

User sees 32 x 6 x 512 words of storage

Primary
32 Pages
512 words/page

Secondary
(Drum)
32x6 pages

Central
Memory

Hardware Organization of Atlas

Effective Address → **Initial Address Decode** →

16 ROM pages
0.4 ~1 μsec — system code (not swapped)

2 subsidiary pages
1.4 μsec — system data (not swapped)

48-bit words
512-word pages

1 Page Address Register (PAR) per page frame

PARs

0
⋮
31

<effective PN , status>

Main
32 pages
1.4 μsec

Drum (4)
192 pages

8 Tape decks
88 sec/word

Hardware Organization of Atlas

Effective Address → Initial Address Decode

16 ROM pages
0.4 ~1 μsec
system code
(not swapped)

2 subsidiary pages
1.4 μsec
system data
(not swapped)

PARs
0
⋮
31

Main
32 pages
1.4 μsec

Drum (4)
192 pages

8 Tape decks
88 sec/word

48-bit words
512-word pages

1 Page Address Register (PAR) per page frame

<effective PN , status>

Compare the effective page address against all 32 PARs
match   ⇒ normal access
no match   ⇒ *page fault*
   save the state of the partially executed instruction

Atlas Demand Paging Scheme

- ## On a page fault:
  - Input transfer into a free page is initiated

  - The Page Address Register (PAR) is updated

  - If no free page is left, a *page is selected to be replaced* (based on usage)

  - The replaced page is written on the drum
    - to minimize the drum latency effect, the first empty page on the drum was selected

  - The *page table is updated* to point to the new location of the page on the drum

Caching vs. Demand Paging
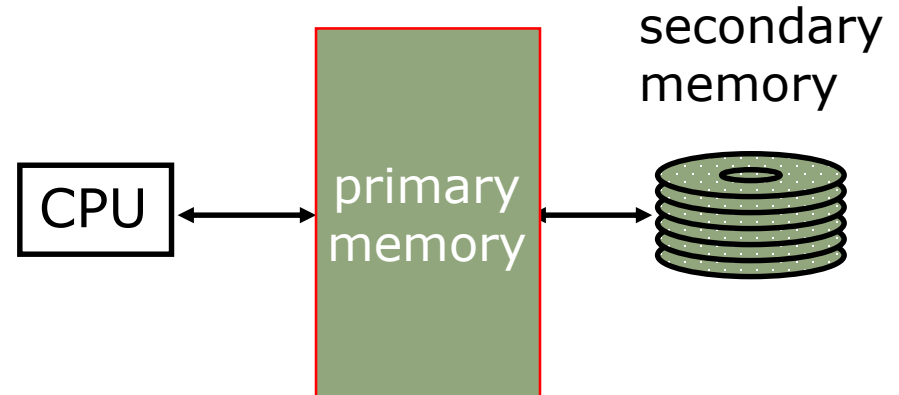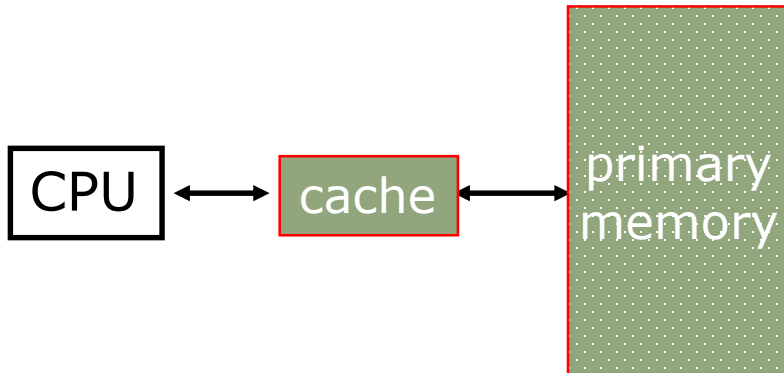
Caching

  cache entry
  cache block (~32 bytes)
  cache miss rate (1% to 20%)
  cache hit (~1 cycle)
  cache miss (~100 cycles)
  a miss is handled
      in *hardware*

Demand paging

  page frame
  page (~4K bytes)
  page miss rate (<0.001%)
  page hit (~100 cycles)
  page miss (~5M cycles)
  a miss is handled
      mostly in *software*

Modern Virtual Memory Systems
Illusion of a large, private, uniform store

Protection & Privacy
> several users, each with their private
> address space and one or more
> shared address spaces
> > page table ≡ name space

OS

user$_i$
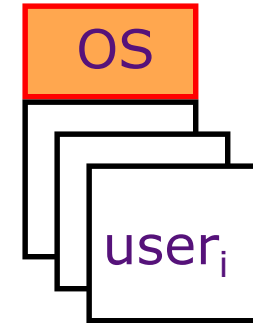
Demand Paging
> Provides the ability to run programs
> larger than the primary memory
>
> Hides differences in machine
> configurations

Swapping Store

Primary Memory

The price is address translation on each memory reference

VA → mapping [TLB] → PA
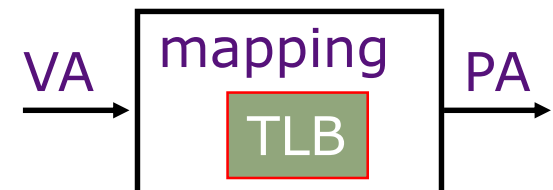
Linear Page Table

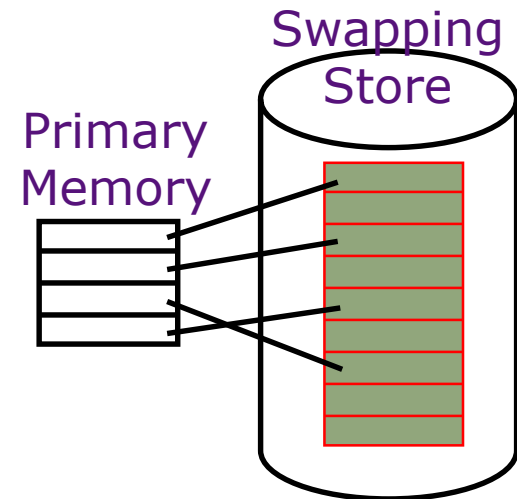- **Page Table Entry (PTE) contains:**
  - A bit to indicate if a page exists
  - PPN (physical page number) for a memory-resident page
  - DPN (disk page number) for a page on the disk
  - Status bits for protection and usage

- **OS sets the Page Table Base Register whenever active user process changes**
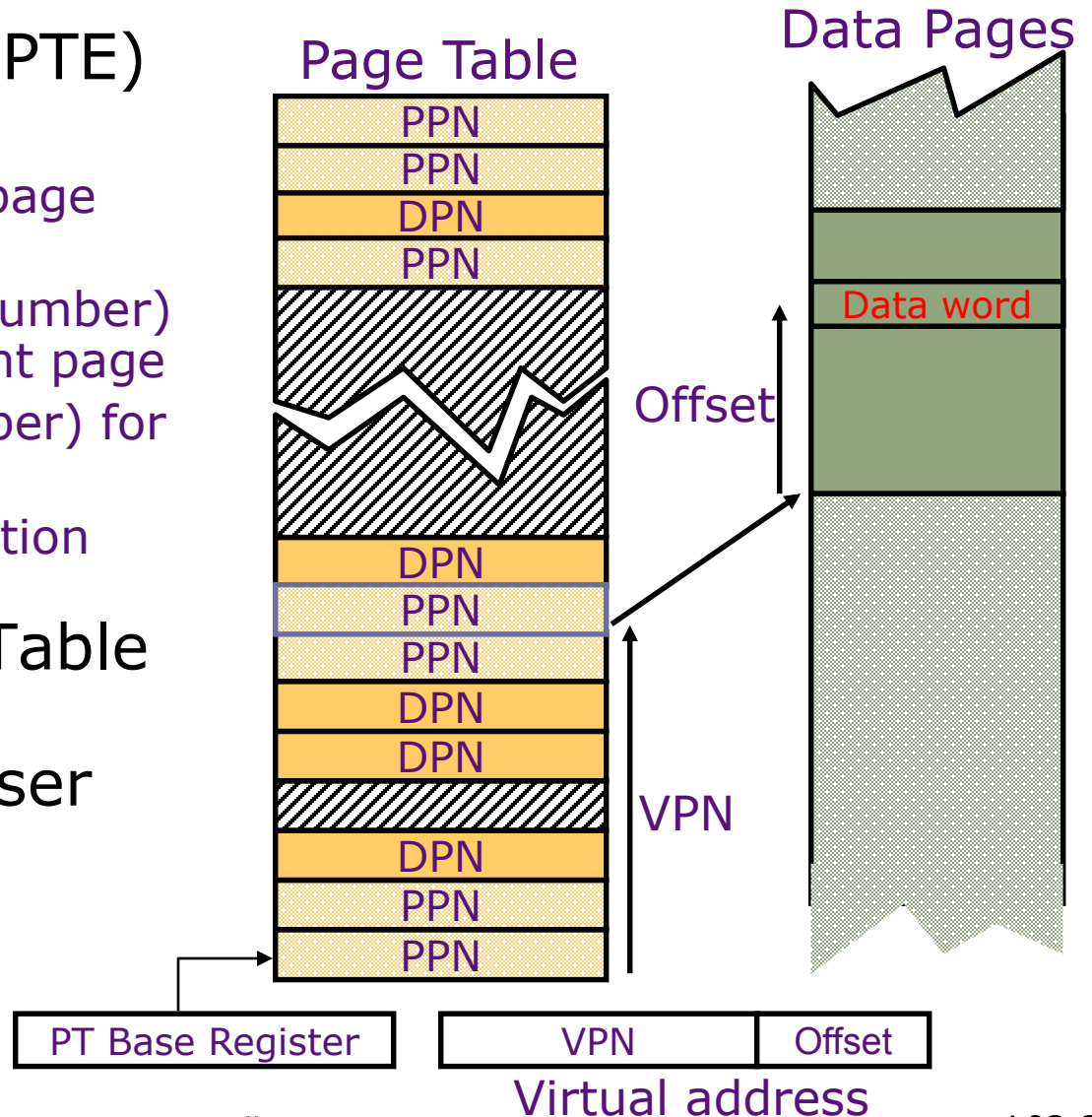
Page Table

Data Pages

PPN
PPN
DPN
PPN

Data word

Offset

DPN
PPN
PPN
DPN
DPN

VPN

DPN
PPN
PPN

PT Base Register

VPN | Offset

Virtual address

Size of Linear Page Table

With 32-bit addresses, 4 KB pages & 4-byte PTEs:

$\Rightarrow$ 2^{20} PTEs, i.e, 4 MB page table per user

$\Rightarrow$ 4 GB of swap space needed to back up the full virtual address space

Size of Linear Page Table

With 32-bit addresses, 4 KB pages & 4-byte PTEs:

$\Rightarrow$ 2^{20} PTEs, i.e, 4 MB page table per user

$\Rightarrow$ 4 GB of swap space needed to back up the full virtual address space

Larger pages?

- Internal fragmentation (Not all memory in a page is used)
- Larger page fault penalty (more time to read from disk)

Size of Linear Page Table

With 32-bit addresses, 4 KB pages & 4-byte PTEs:

$\Rightarrow$ 2^{20} PTEs, i.e, 4 MB page table per user

$\Rightarrow$ 4 GB of swap space needed to back up the full virtual address space

Larger pages?

- Internal fragmentation (Not all memory in a page is used)
- Larger page fault penalty (more time to read from disk)

What about 64-bit virtual address space???

- Even 1MB pages would require 2^{44} 8-byte PTEs (35 TB!)

Size of Linear Page Table

With 32-bit addresses, 4 KB pages & 4-byte PTEs:

$\Rightarrow$ 2^{20} PTEs, i.e, 4 MB page table per user

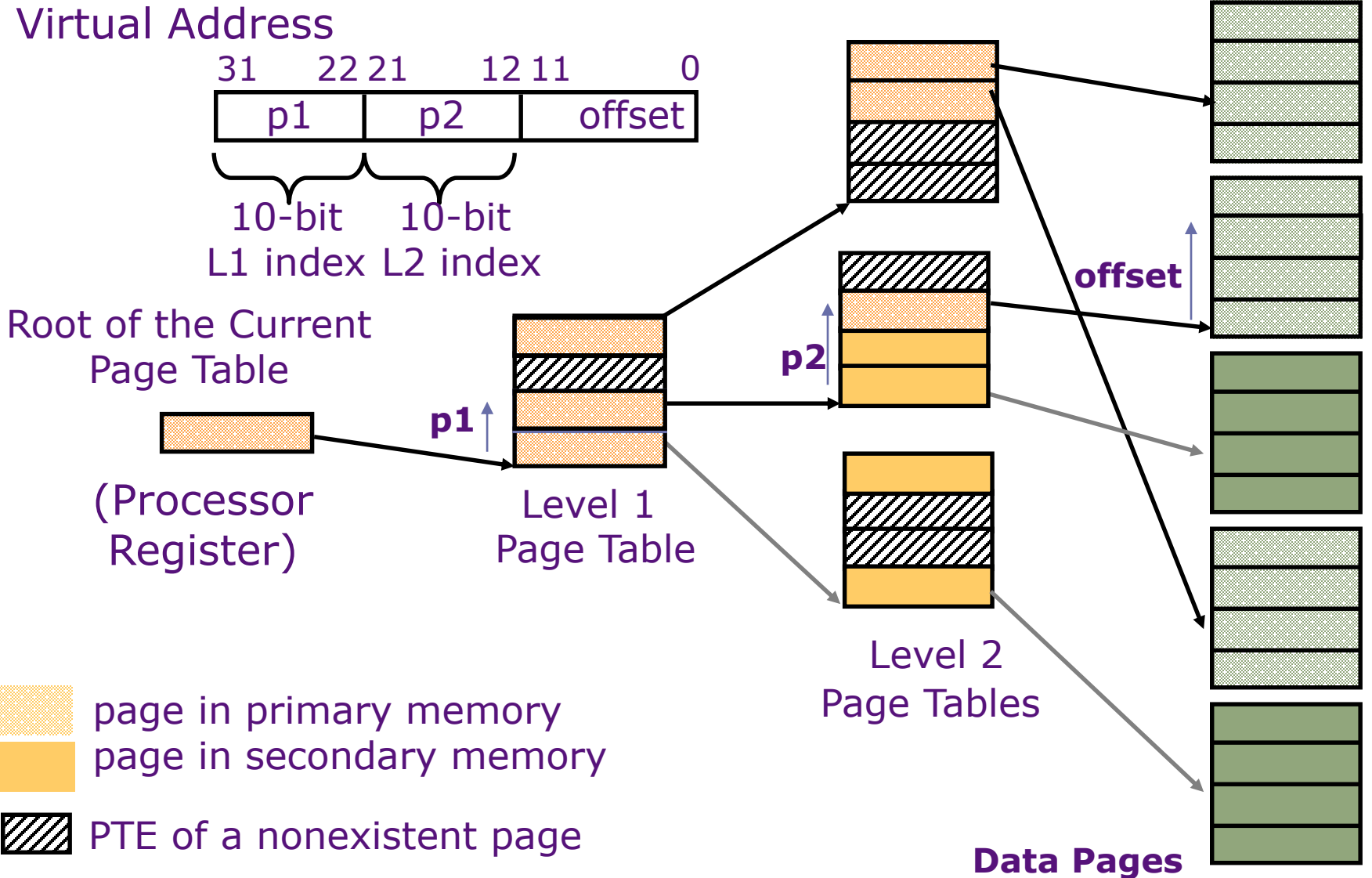$\Rightarrow$ 4 GB of swap space needed to back up the full virtual address space

Larger pages?

- Internal fragmentation (Not all memory in a page is used)
- Larger page fault penalty (more time to read from disk)

What about 64-bit virtual address space???

- Even 1MB pages would require 2^{44} 8-byte PTEs (35 TB!)

What is the "saving grace"?

Hierarchical Page Table

Virtual Address

```
 31      22 21      12 11          0
┌────────┬──────────┬──────────────┐
│   p1   │    p2    │    offset    │
└────────┴──────────┴──────────────┘
```

10-bit 10-bit
L1 index L2 index

Root of the Current
Page Table

(Processor
Register)

p1

Level 1
Page Table

p2

Level 2
Page Tables

offset

Data Pages

page in primary memory
page in secondary memory

PTE of a nonexistent page

Address Translation & Protection

Virtual Address	Virtual Page No. (VPN)	offset

Kernel/User Mode

Read/Write

Protection Check

Address Translation

Exception?

Physical Address	Physical Page No. (PPN)	offset
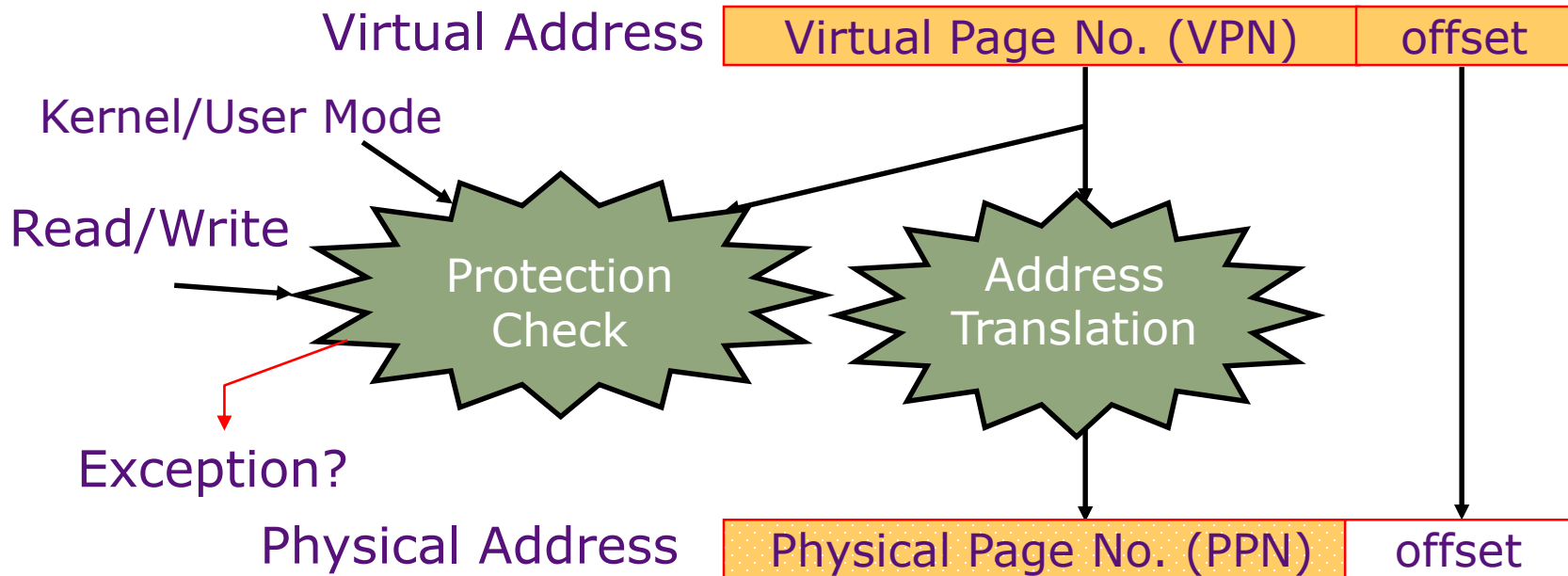
• Every instruction and data access needs address translation and protection checks

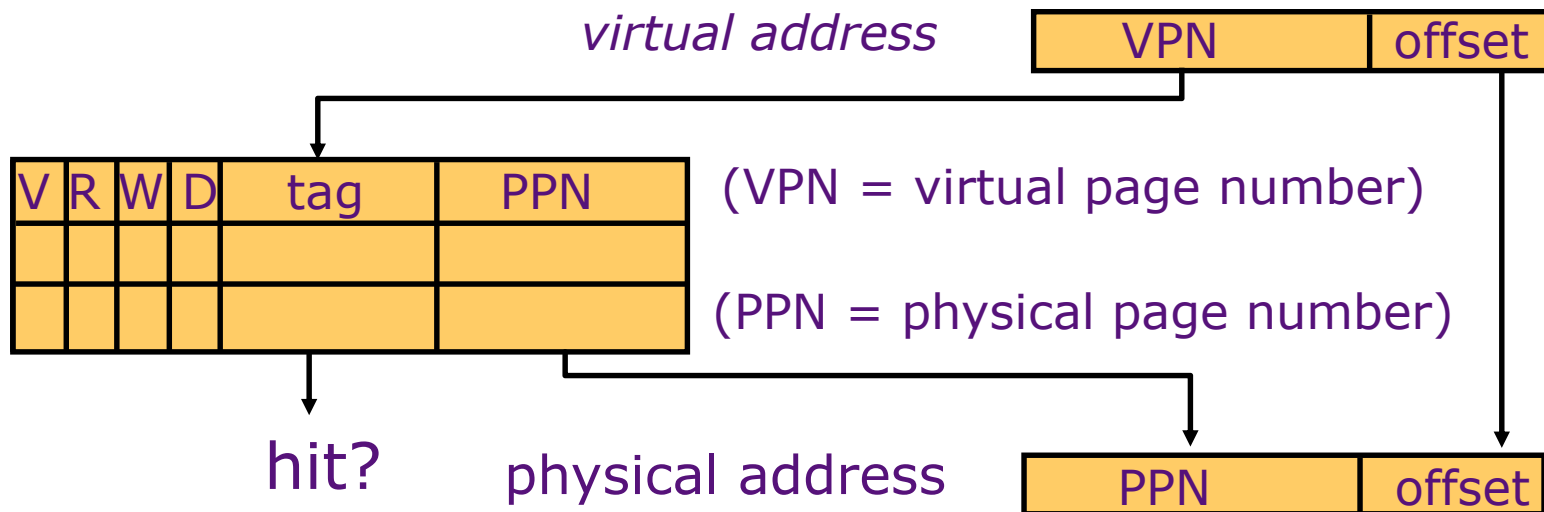*A good VM design needs to be fast (~ one cycle) and space-efficient*

Translation Lookaside Buffers

Address translation is very expensive!
    In a two-level page table, each reference
    becomes several memory accesses

Solution: *Cache translations in TLB*

      TLB hit $\Rightarrow$ *Single-cycle Translation*
      TLB miss $\Rightarrow$ *Page Table Walk to refill*

virtual address | VPN | offset |

V	R	W	D	tag	PPN

(VPN = virtual page number)

(PPN = physical page number)

hit?   physical address | PPN | offset |

TLB Designs

- Typically 32-128 entries, usually highly associative
  - Each entry maps a large page, hence less spatial locality across pages ➜ more likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-8 way set-associative

- Random or FIFO replacement policy

- No process information in TLB?

- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB

Example: 64 TLB entries, 4KB pages, one page per entry

TLB Reach = _____?

TLB Designs

- Typically 32-128 entries, usually highly associative
  - Each entry maps a large page, hence less spatial locality across pages ➜ more likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-8 way set-associative

- Random or FIFO replacement policy

- No process information in TLB?

- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB

Example: 64 TLB entries, 4KB pages, one page per entry

TLB Reach = <u>*64 entries * 4 KB = 256 KB (if contiguous)*</u> *?*

Next lecture:

Modern Virtual Memory Systems