## CSC 252: Computer Organization Spring 2021: Lecture 18

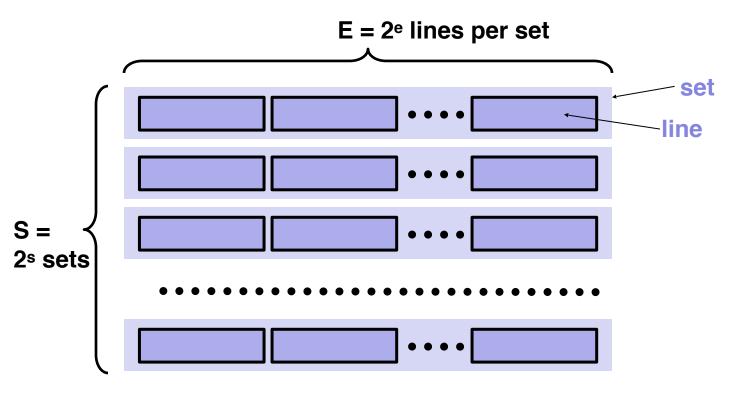
Instructor: Yuhao Zhu

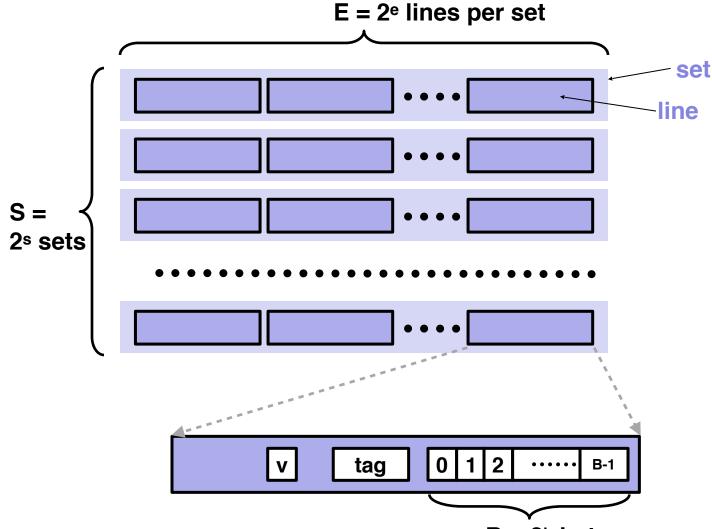
Department of Computer Science
University of Rochester

#### **Announcements**

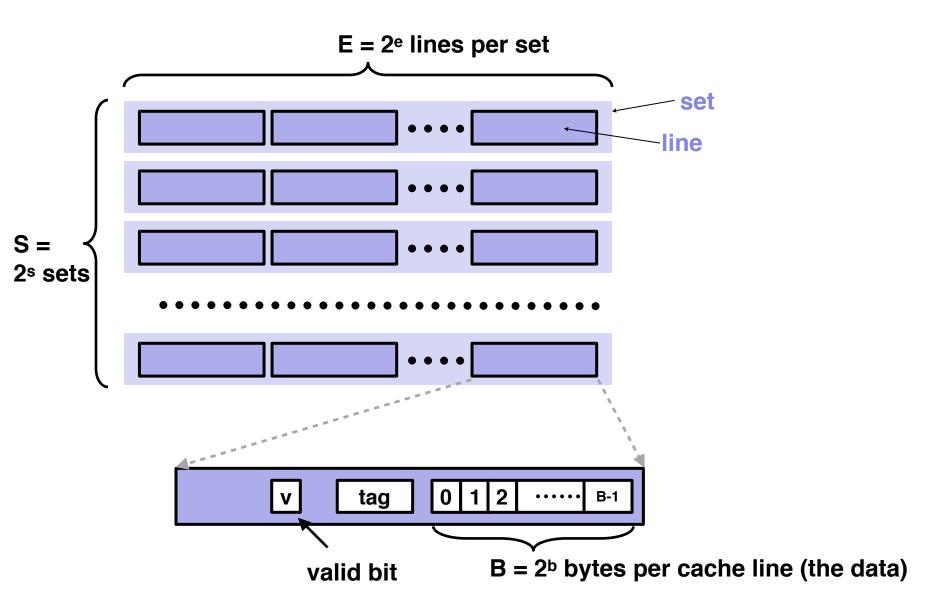
- Cache problem set: <a href="https://www.cs.rochester.edu/courses/252/spring2021/handouts.html">https://www.cs.rochester.edu/courses/252/spring2021/handouts.html</a>
- Not to be turned in. Won't be graded.
- Assignment 4 soon to be released later today.
- Final exam: Wednesday, May 12, 19:15 PM. 3 hours.

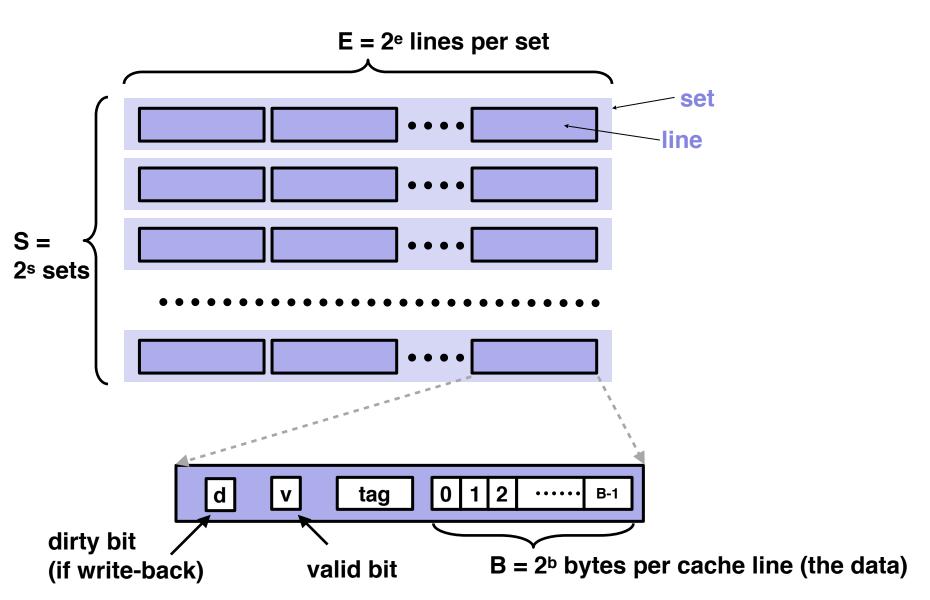
4	5	6	7	3	9	10
				Today		
11	12	13	14	15	16	17
18	19	20	21	22	Due	24

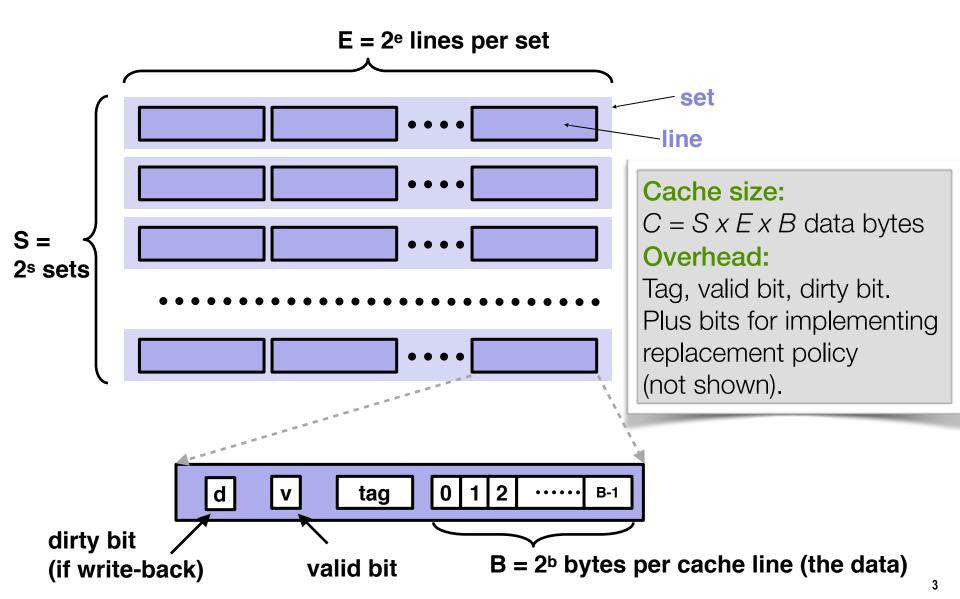




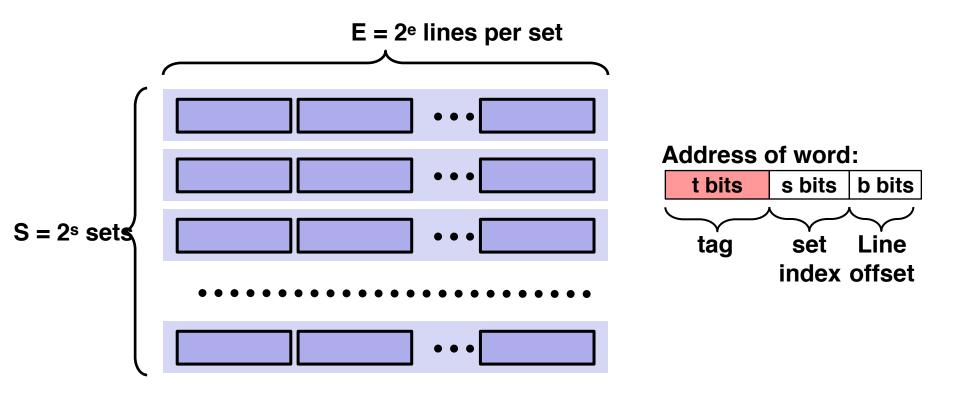
B = 2<sup>b</sup> bytes per cache line (the data)







#### **Cache Access**



# Cache Access E = 2e lines per set Address of word: t bits s bits b bits

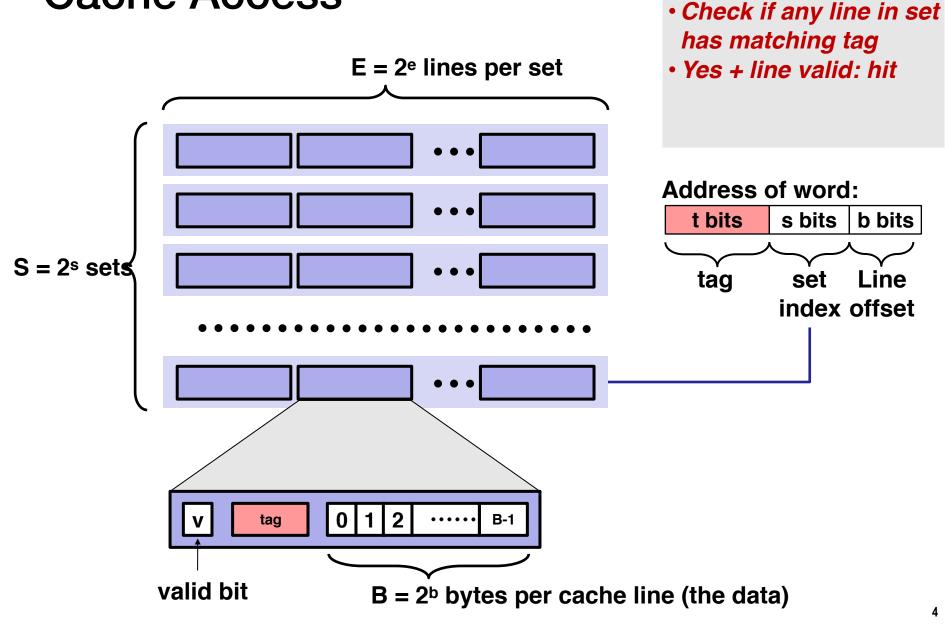
tag

set

Line

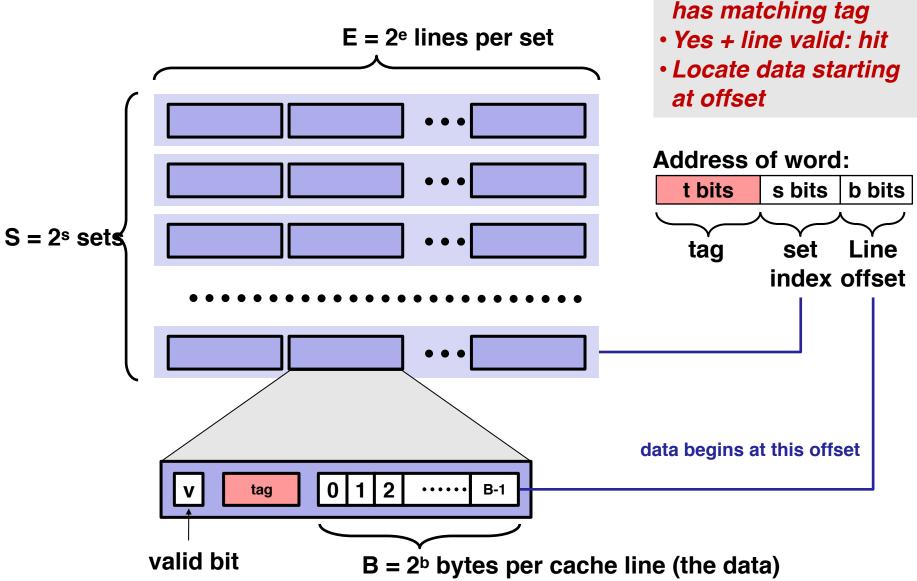
index offset

#### Cache Access



Locate set

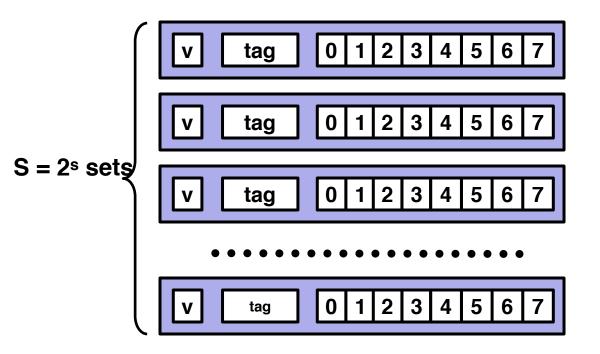
## Cache Access



Locate set

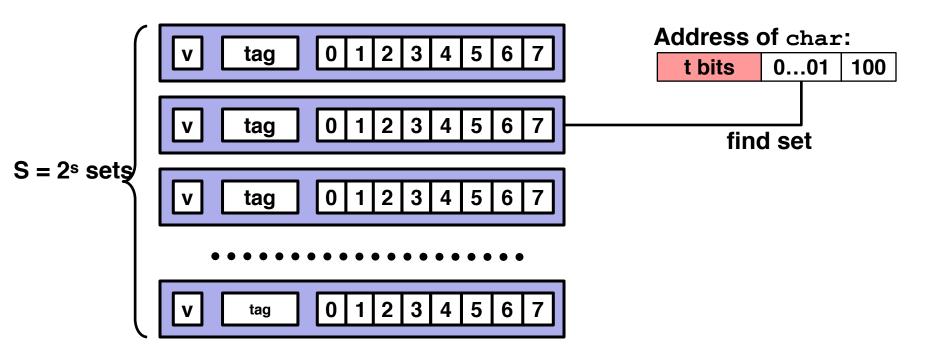
• Check if any line in set

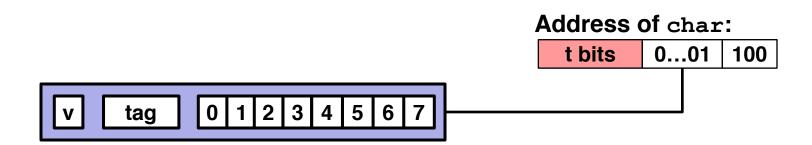
Direct mapped: One line per set Assume: cache line size 8 bytes

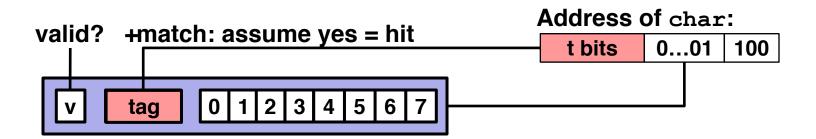


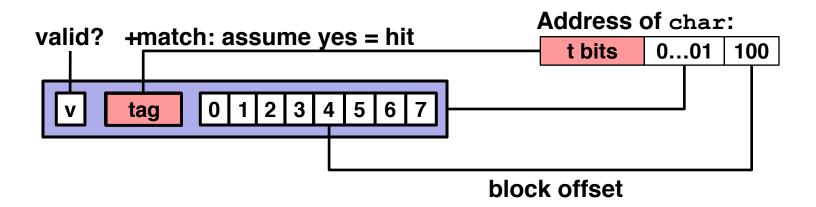
#### Address of char:

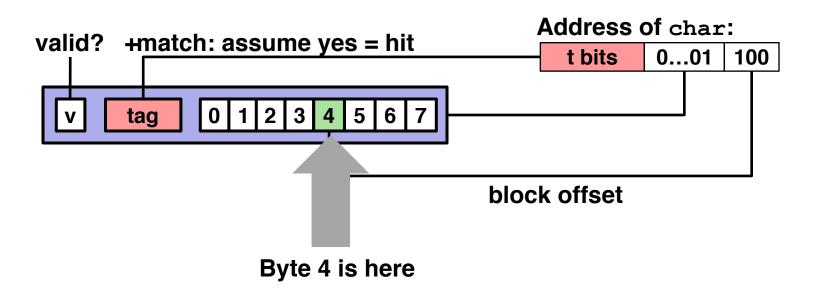
t bits 0...01 100



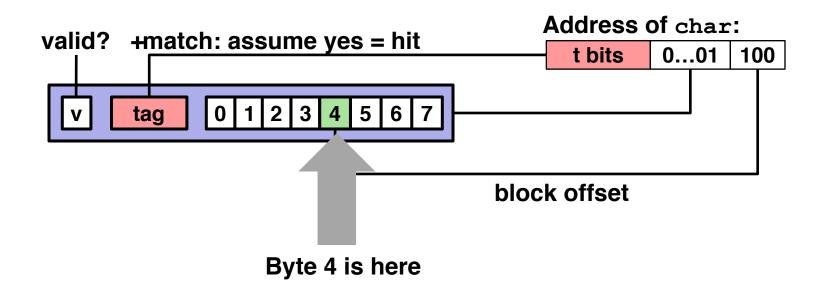








Direct mapped: One line per set Assume: cache line size 8 bytes



If tag doesn't match: old line is evicted and replaced

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

Address trace (reads, one byte per read):

0	[0 <u>00</u> 0 <sub>2</sub> ],
1	[0 <u>00</u> 1 <sub>2</sub> ],
7	[0 <u>11</u> 1 <sub>2</sub> ],
8	[1 <u>00</u> 0 <sub>2</sub> ],

0<u>00</u>0<sub>2</sub>]

	V	Tag	Line
Set 0	0	?	?
Set 1			
Set 2			
Set 3			

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

Address trace (reads, one byte per read):

miss

	- ( · · · · · · · · · · · · · · · · · ·
0	[0 <u>00</u> 0 <sub>2</sub> ],
1	[0 <u>00</u> 1 <sub>2</sub> ],
7	[0 <u>11</u> 1 <sub>2</sub> ],
8	[1 <u>00</u> 0 <sub>2</sub> ],
0	[0 <u>00</u> 0 <sub>2</sub> ]

	V	Tag	Line
Set 0	0	?	?
Set 1			
Set 2			
Set 3			

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

Address trace (reads, one byte per read):

miss

0	[0 <u>00</u> 0 <sub>2</sub> ],
1	[0 <u>00</u> 1 <sub>2</sub> ],
7	[0 <u>11</u> 1 <sub>2</sub> ],
8	[1 <u>00</u> 0 <sub>2</sub> ],
0	[0000_]

	V	Tag	Line
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3			

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

Address trace (reads, one byte per read):

0	[0 <u>00</u> 0 <sub>2</sub> ],	miss
1	[0 <u>00</u> 1 <sub>2</sub> ],	
7	[0 <u>11</u> 1 <sub>2</sub> ],	
8	[1 <u>00</u> 0 <sub>2</sub> ],	
0	$[0000_{2}]$	

	V	Tag	Line	
Set 0	1	0	M[0-1]	<b> ←</b> Tł
Set 1				
Set 2				
Set 3		_		

← The two bytes at memory address 0 and 1

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

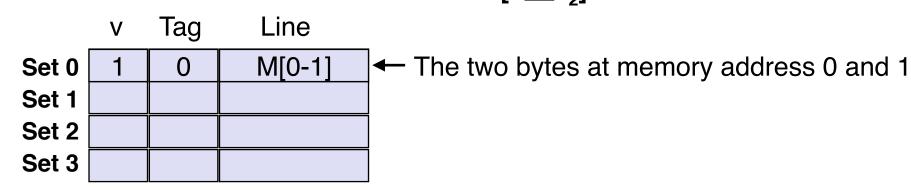
0	[0 <u>00</u> 0 <sub>2</sub> ],	miss
1	[0 <u>00</u> 1 <sub>2</sub> ],	hit
7	[0 <u>11</u> 1 <sub>2</sub> ],	
8	[1 <u>00</u> 0 <sub>2</sub> ],	
0	[.0000]	

	V	Tag	Line	
Set 0	1	0	M[0-1]	The two bytes at memory address 0 and 1
Set 1				
Set 2				
Set 3	_			

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

	, ,	· <i>J</i>
0	[0 <u>00</u> 0 <sub>2</sub> ],	miss
1	[0 <u>00</u> 1 <sub>2</sub> ],	hit
7	[0 <u>11</u> 1 <sub>2</sub> ],	miss
8	[1 <u>00</u> 0 <sub>2</sub> ],	
0	[0 <u>00</u> 0 <sub>2</sub> ]	



t=1	s=2	b=1
X	XX	X

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

	,	<i>.</i>
0	[0 <u>00</u> 0 <sub>2</sub> ],	miss
1	[0 <u>00</u> 1 <sub>2</sub> ],	hit
7	[0 <u>11</u> 1 <sub>2</sub> ],	miss
8	[1 <u>00</u> 0 <sub>2</sub> ],	
0	$[0000_{2}]$	

	V	Tag	Line	
Set 0	1	0	M[0-1]	← The two bytes at memory address 0 and 1
Set 1				
Set 2				
Set 3	1	0	M[6-7]	

_t=1	s=2	b=1
X	XX	X

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

	\. \. \. \. \. \. \. \. \. \. \. \. \. \	- <b>,</b>
0	[0 <u>00</u> 0 <sub>2</sub> ],	miss
1	[0 <u>00</u> 1 <sub>2</sub> ],	hit
7	[0 <u>11</u> 1 <sub>2</sub> ],	miss
8	[1 <u>00</u> 0 <sub>2</sub> ],	
0	[ <sub>0</sub> 0000]	

	V	Tag	Line	
Set 0	1	0	M[0-1]	The two bytes at memory address 0 and 1
Set 1				
Set 2				
Set 3	1	0	M[6-7]	The two bytes at memory address 6 and 7

t=1	s=2	b=1
X	XX	X

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

taarooo traoo	(ioaao, oiio	by to poi load,
0	[0 <u>00</u> 0 <sub>2</sub> ],	miss
1	[0 <u>00</u> 1 <sub>2</sub> ],	hit
7	[0 <u>11</u> 1 <sub>2</sub> ],	miss
8	[1 <u>00</u> 0 <sub>2</sub> ],	miss
0	[.0000]	

				- 2-
	V	Tag	Line	
Set 0	1	0	M[0-1]	The two bytes at memory address 0 and 1
Set 1				
Set 2				
Set 3	1	0	M[6-7]	The two bytes at memory address 6 and 7

t=1	s=2	b=1
X	XX	X

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

	(. 5 % 6. 5	, 10 p c cate
0	[0 <u>00</u> 0 <sub>2</sub> ],	miss
1	[0 <u>00</u> 1 <sub>2</sub> ],	hit
7	[0 <u>11</u> 1 <sub>2</sub> ],	miss
8	[1 <u>00</u> 0 <sub>2</sub> ],	miss
0	[0000]	

				<u> </u>
	V	Tag	Line	
Set 0	1	1	M[8-9]	The two bytes at memory address 8 and 9
Set 1				
Set 2				
Set 3	1	0	M[6-7]	The two bytes at memory address 6 and 7

_t=1	s=2	b=1
X	XX	X

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

aarooo traoo	(I dadd, dild	by to poi loa
0	[0 <u>00</u> 0 <sub>2</sub> ],	miss
1	[0 <u>00</u> 1 <sub>2</sub> ],	hit
7	[0 <u>11</u> 1 <sub>2</sub> ],	miss
8	[1 <u>00</u> 0 <sub>2</sub> ],	miss
0	[.0000]	miss

	V	Tag	Line	
Set 0	1	1	M[8-9]	The two bytes at memory address 8 and 9
Set 1				
Set 2				
Set 3	1	0	M[6-7]	The two bytes at memory address 6 and 7

t=1	s=2	b=1	
X	XX	X	

4-bit address space, i.e., Memory = 16 bytes

B=2 bytes/line, S=4 sets, E=1 line/set

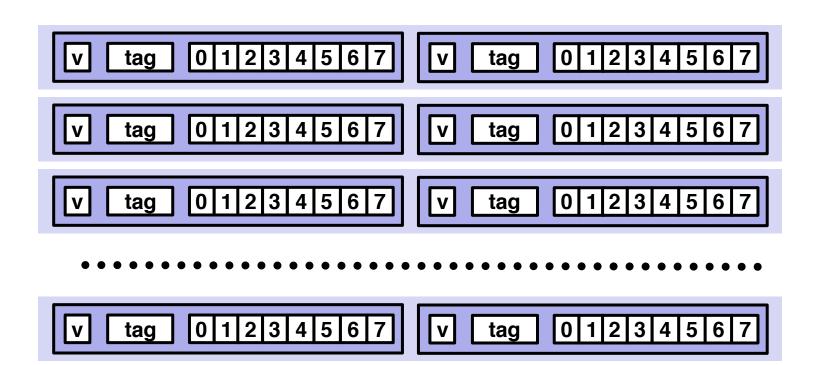
	(. 5 % %	
0	[0 <u>00</u> 0 <sub>2</sub> ],	miss
1	[0 <u>00</u> 1 <sub>2</sub> ],	hit
7	[0 <u>11</u> 1 <sub>2</sub> ],	miss
8	[1 <u>00</u> 0 <sub>2</sub> ],	miss
0	[c0000]	miss

	V	Tag	Line	
Set 0	1	0	M[0-1]	The two bytes at memory address 0 and 1
Set 1				
Set 2				
Set 3	1	0	M[6-7]	The two bytes at memory address 6 and 7

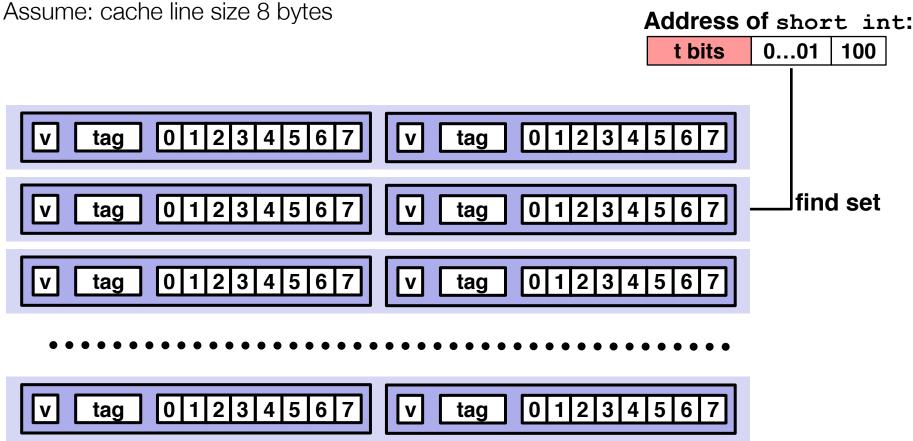
E = 2: Two lines per set

Assume: cache line size 8 bytes

Address of short int:
t bits 0...01 100

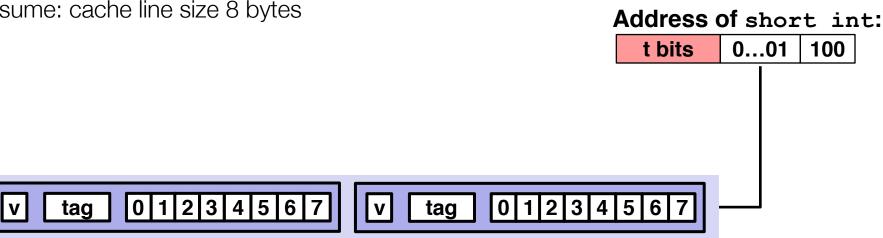


E = 2: Two lines per set



E = 2: Two lines per set

Assume: cache line size 8 bytes



E = 2: Two lines per set

Assume: cache line size 8 bytes

Compare both

t bits

0...01

100

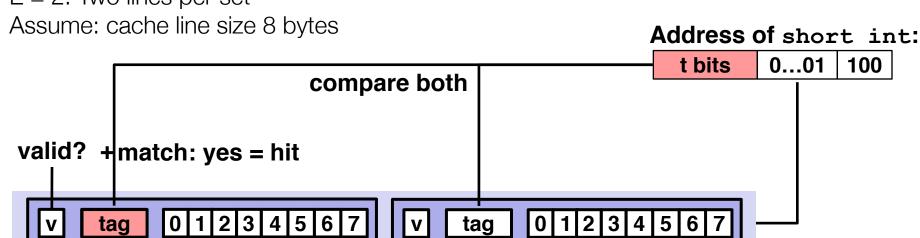
v tag

0 1 2 3 4 5 6 7

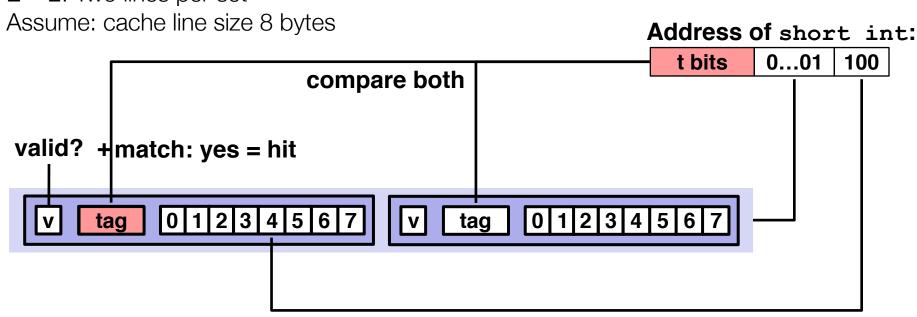
v tag

0 1 2 3 4 5 6 7

E = 2: Two lines per set



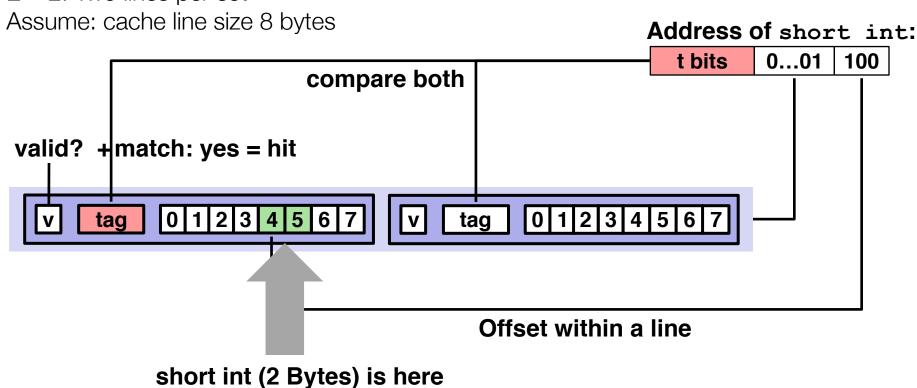
E = 2: Two lines per set



Offset within a line

## E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

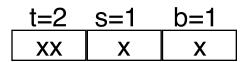


## E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set Assume: cache line size 8 bytes Address of short int: 0...01 t bits 100 compare both valid? +|match: yes = hit tag |0|1|2|3|4|5|6|7 Offset within a line short int (2 Bytes) is here

#### No match:

- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU),

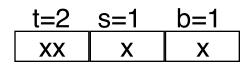


4-bit address space, i.e., Memory = 16 bytes S=2 sets, E=2 cache lines/set

- $0 \qquad [00\underline{0}0_2],$
- 1  $[0001_2]$ ,
- 7  $[01\underline{1}1_2],$
- 8  $[10\underline{0}0_2],$
- $0 \qquad [00\underline{0}0_2]$

	V	Tag	Line		
Set 0	0	?	?	0	



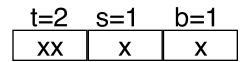


4-bit address space, i.e., Memory = 16 bytes S=2 sets, E=2 cache lines/set

0	[00 <u>0</u> 0 <sub>2</sub> ],	miss
1	[00 <u>0</u> 1 <sub>2</sub> ],	
7	[01 <u>1</u> 1 <sub>2</sub> ],	
8	[10 <u>0</u> 0 <sub>2</sub> ],	
0	[c0000]	

	V	Tag	Line		
Set 0	0	?	?	0	



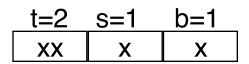


4-bit address space, i.e., Memory = 16 bytes S=2 sets, E=2 cache lines/set

0	[00 <u>0</u> 0 <sub>2</sub> ],	miss
1	[00 <u>0</u> 1 <sub>2</sub> ],	
7	[01 <u>1</u> 1 <sub>2</sub> ],	
8	[10 <u>0</u> 0 <sub>2</sub> ],	
0	[0000]	

	V	Tag	Line		
Set 0	1	00	M[0-1]	0	



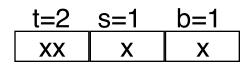


4-bit address space, i.e., Memory = 16 bytes S=2 sets, E=2 cache lines/set

0	[00 <u>0</u> 0 <sub>2</sub> ],	miss
1	[00 <u>0</u> 1 <sub>2</sub> ],	hit
7	[01 <u>1</u> 1 <sub>2</sub> ],	
8	[10 <u>0</u> 0 <sub>2</sub> ],	
0	[00 <u>0</u> 0 <sub>2</sub> ]	

	V	Tag	Line			
Set 0	1	00	M[0-1]	(	C	



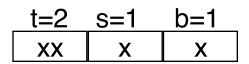


4-bit address space, i.e., Memory = 16 bytes S=2 sets, E=2 cache lines/set

0	[00 <u>0</u> 0 <sub>2</sub> ],	miss
1	[00 <u>0</u> 1 <sub>2</sub> ],	hit
7	[01 <u>1</u> 1 <sub>2</sub> ],	miss
8	[10 <u>0</u> 0 <sub>2</sub> ],	
0	[00 <u>0</u> 0 <sub>2</sub> ]	

	V	Tag	Line			
Set 0	1	00	M[0-1]	0		



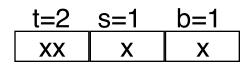


4-bit address space, i.e., Memory = 16 bytes S=2 sets, E=2 cache lines/set

0	[00 <u>0</u> 0 <sub>2</sub> ],	miss
1	[00 <u>0</u> 1 <sub>2</sub> ],	hit
7	[01 <u>1</u> 1 <sub>2</sub> ],	miss
8	[10 <u>0</u> 0 <sub>2</sub> ],	
0	[00 <u>0</u> 0 <sub>2</sub> ]	

	V	Tag	Line			
Set 0	1	00	M[0-1]	(	0	



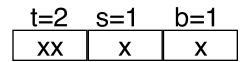


4-bit address space, i.e., Memory = 16 bytes S=2 sets, E=2 cache lines/set

0	[00 <u>0</u> 0 <sub>2</sub> ],	,	miss
1	[00 <u>0</u> 1 <sub>2</sub> ],		hit
7	[01 <u>1</u> 1 <sub>2</sub> ],		miss
8	[10 <u>0</u> 0 <sub>2</sub> ],		miss
0	[0000]		

	V	Tag	Line		
Set 0	1	00	M[0-1]	0	



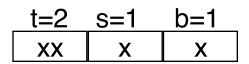


4-bit address space, i.e., Memory = 16 bytes S=2 sets, E=2 cache lines/set

0	[00 <u>0</u> 0 <sub>2</sub> ],	miss
1	[00 <u>0</u> 1 <sub>2</sub> ],	hit
7	[01 <u>1</u> 1 <sub>2</sub> ],	miss
8	[10 <u>0</u> 0 <sub>2</sub> ],	miss
0	[00 <u>0</u> 0 <sub>2</sub> ]	

	V	Tag	Line				
Set 0	1	00	M[0-1]	1	10	M[8-9]	





4-bit address space, i.e., Memory = 16 bytes S=2 sets, E=2 cache lines/set

0	[00 <u>0</u> 0 <sub>2</sub> ],	miss
1	[00 <u>0</u> 1 <sub>2</sub> ],	hit
7	[01 <u>1</u> 1 <sub>2</sub> ],	miss
8	[10 <u>0</u> 0 <sub>2</sub> ],	miss
0	[0000]	hit

	V	Tag	Line			
Set 0	1	00	M[0-1]	1	10	M[8-9]



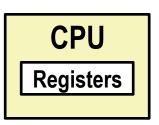
# **Today**

- Processes and Signals: running multiple programs concurrently
  - Processes
  - Process Control
  - Signals

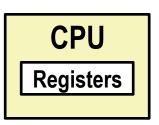
- Definition: A *process* is an instance of a running program.
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"

- Definition: A *process* is an instance of a running program.
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:

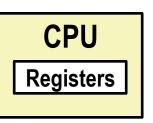
- Definition: A process is an instance of a running program.
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - "Owns" the CPU



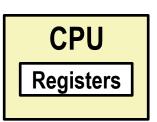
- Definition: A process is an instance of a running program.
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - "Owns" the CPU
    - Each program seems to have exclusive use of the CPU



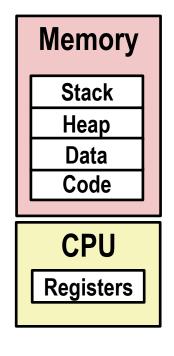
- Definition: A process is an instance of a running program.
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - "Owns" the CPU
    - Each program seems to have exclusive use of the CPU
    - Done by the OS kernel through "context switching"



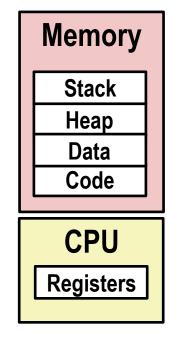
- Definition: A process is an instance of a running program.
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - "Owns" the CPU
    - Each program seems to have exclusive use of the CPU
    - Done by the OS kernel through "context switching"
  - Private address space



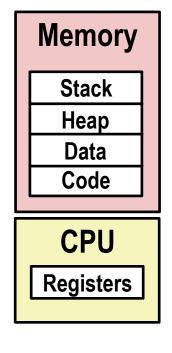
- Definition: A process is an instance of a running program.
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - "Owns" the CPU
    - Each program seems to have exclusive use of the CPU
    - Done by the OS kernel through "context switching"
  - Private address space



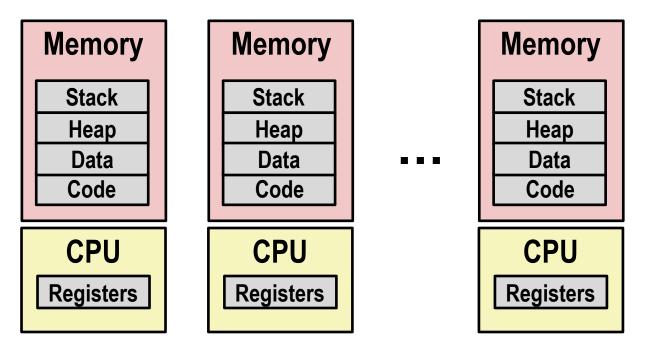
- Definition: A process is an instance of a running program.
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - "Owns" the CPU
    - Each program seems to have exclusive use of the CPU
    - Done by the OS kernel through "context switching"
  - Private address space
    - Each program seems to have exclusive use of main memory.



- Definition: A process is an instance of a running program.
  - One of the most profound ideas in computer science
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - "Owns" the CPU
    - Each program seems to have exclusive use of the CPU
    - Done by the OS kernel through "context switching"
  - Private address space
    - Each program seems to have exclusive use of main memory.
    - Provided by OS through "virtual memory"



# Multiprocessing: The Illusion

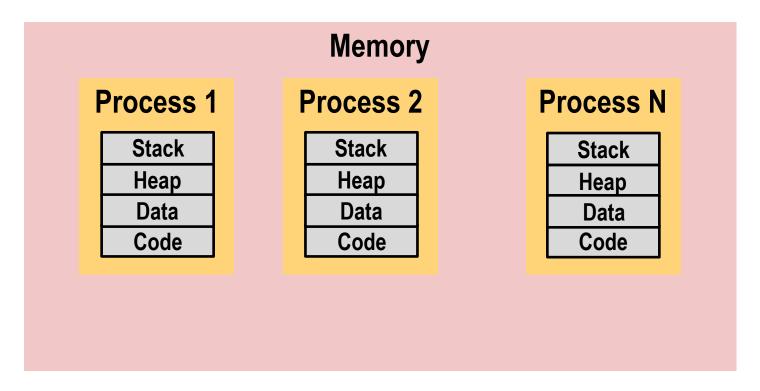


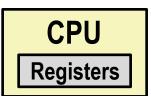
- Computer runs many processes simultaneously
  - Applications for one or more users
    - Web browsers, email clients, editors, ...
  - Background tasks
    - Monitoring network & I/O devices

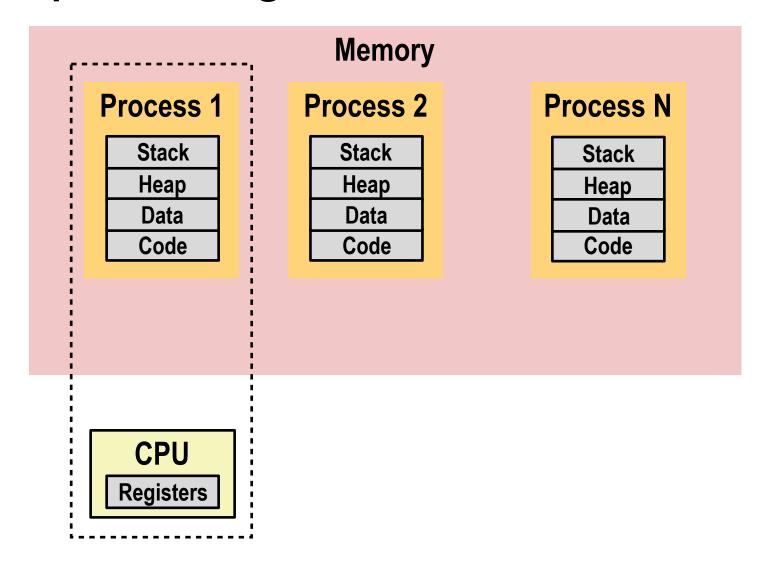
### Multiprocessing Example

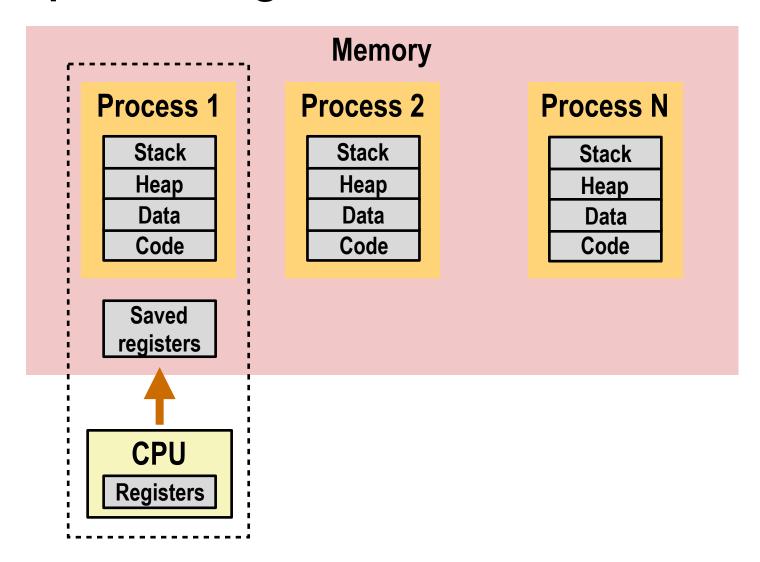
```
000
                                          X xterm
Processes: 123 total, 5 running, 9 stuck, 109 sleeping, 611 threads
                                                                                      11:47:07
Load Avg: 1.03, 1.13, 1.14 CPU usage: 3.27% user, 5.15% sys, 91.56% idle
SharedLibs: 576K resident, OB data, OB linkedit.
MemRegions: 27958 total, 1127M resident, 35M private, 494M shared.
PhysMem: 1039M wired, 1974M active, 1062M inactive, 4076M used, 18M free.
VM: 280G vsize, 1091M framework vsize, 23075213(1) pageins, 5843367(0) pageouts.
Networks: packets: 41046228/11G in, 66083096/77G out.
Disks: 17874391/349G read, 12847373/594G written.
PID
      COMMAND
                                  #TH
                                             #PORT #MREG RPRVT
                                                                 RSHRD
                                                                        RSIZE
                    2CPU TIME
                                        #WO
                                                                               VPRVT
                                                                                      VSIZE
99217- Microsoft Of 0.0 02:28.34 4
                                             202
                                                   418
                                                         21M
                                                                 24M
                                                                        21M
                                                                               66M
                                                                                      763M
99051
      usbmuxd
                    0.0 00:04.10 3
                                             47
                                                   66
                                                         436K
                                                                 216K
                                                                        480K
                                                                               60M
                                                                                      2422M
99006
      iTunesHelper 0.0 00:01.23 2
                                                         728K
                                                                 3124K
                                                                        1124K
                                                                               43M
                                                                                      2429M
                                                   24
                                                         224K
84286
                    0.0 00:00.11 1
                                                                 732K
                                                                        484K
                                                                               17M
                                                                                      2378M
      bash
                                             32
84285
      xterm
                    0.0 00:00.83 1
                                                   73
                                                         656K
                                                                872K
                                                                        692K
                                                                               9728K
                                                                                      2382M
55939- Microsoft Ex 0.3 21:58.97 10
                                             360
                                                   954
                                                                 65M
                                                                               114M
                                                         16M
                                                                        46M
                                                                                      1057M
54751
      sleep
                    0.0 00:00.00 1
                                             17
                                                   20
                                                         92K
                                                                 212K
                                                                        360K
                                                                               9632K
                                                                                      2370M
                                             33
                                                   50
                                                                        1736K
54739
       launchdadd
                    0.0 00:00.00 2
                                                         488K
                                                                 220K
                                                                               48M
                                                                                      2409M
                                             30
                    6.5 00:02.53 1/1
                                                         1416K
                                                                 216K
                                                                        2124K
                                                                               17M
54737
      top
                                                                                      2378M
                    0.0 00:00.02 7
                                             53
54719
      automountd
                                                   64
                                                         860K
                                                                 216K
                                                                        2184K
                                                                               53M
                                                                                      2413M
54701
                    0.0 00:00.05 4
                                             61
                                                         1268K
                                                                2644K
                                                                        3132K
                                                                               50M
                                                                                      2426M
      ocspd
                                                   389+
                                                         15M+
54661
      Grab
                    0.6
                         00:02.75 6
                                                                 26M+
                                                                        40M+
                                                                               75M+
                                                                                      2556M+
                                                                224K
54659
                         00:00.15 2
                                             40
                                                   61
                                                          3316K
                                                                        4088K
                                                                               42M
                                                                                      2411M
      cookied
                    0.0
```

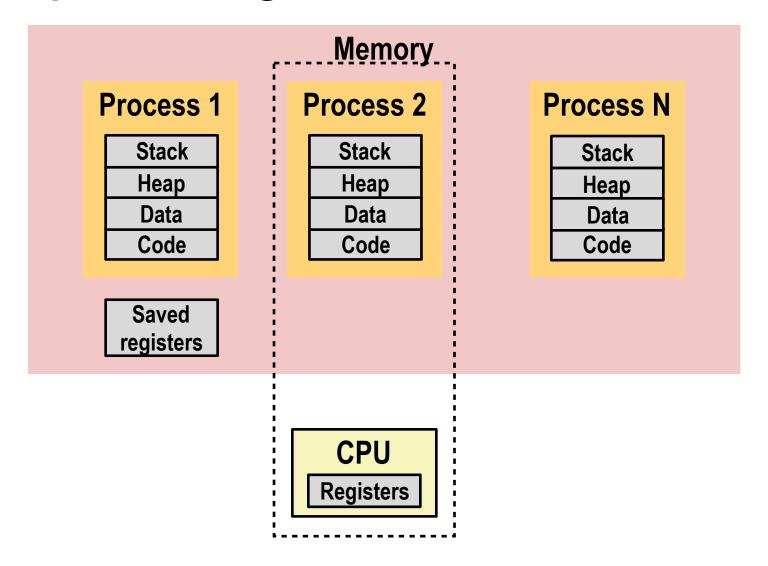
- Running program "top" on Unit/Linux
  - System has 123 processes, 5 of which are active
  - Identified by Process ID (PID)

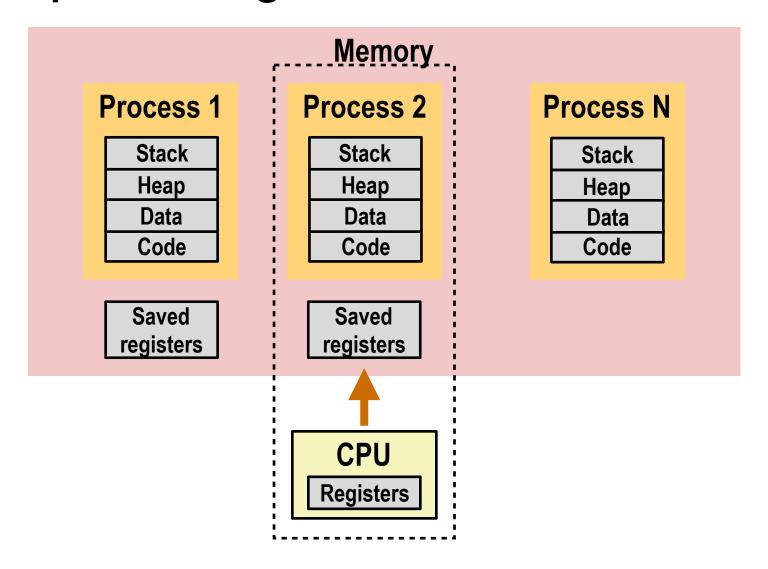


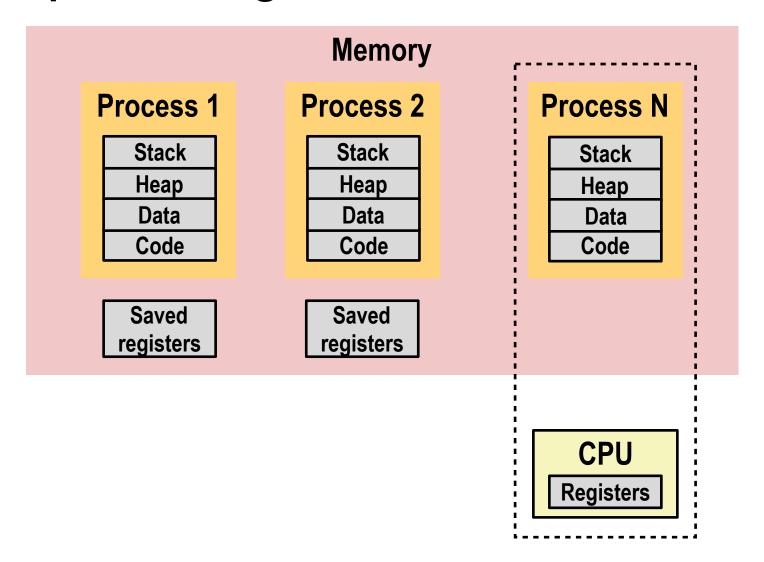


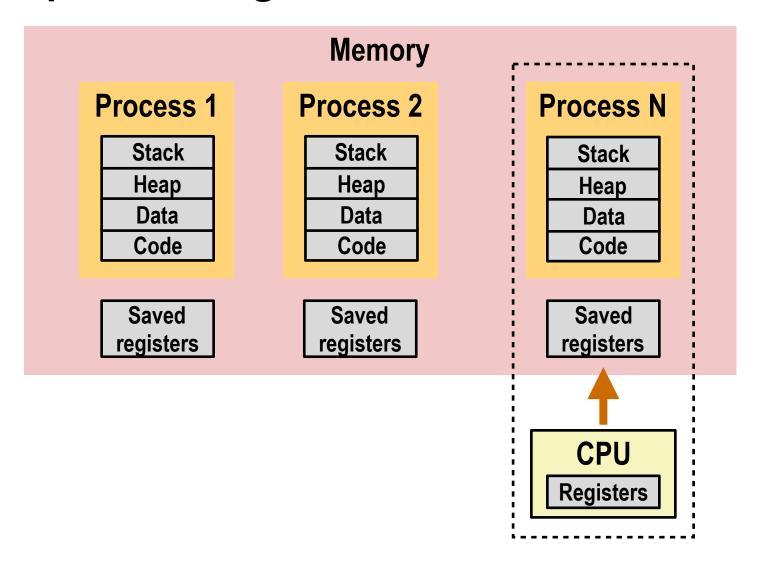


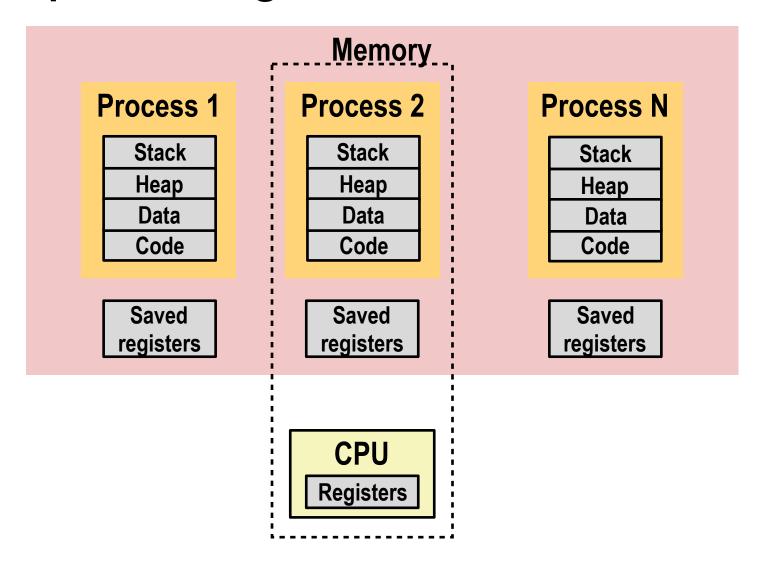


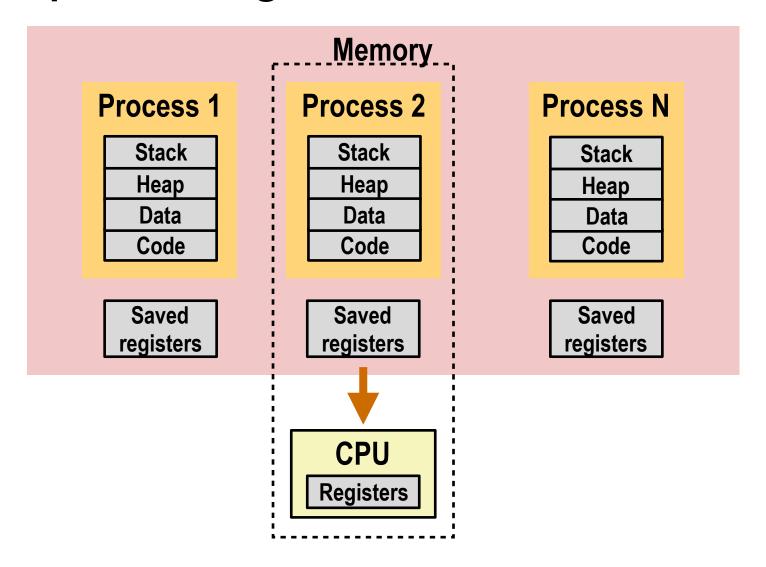


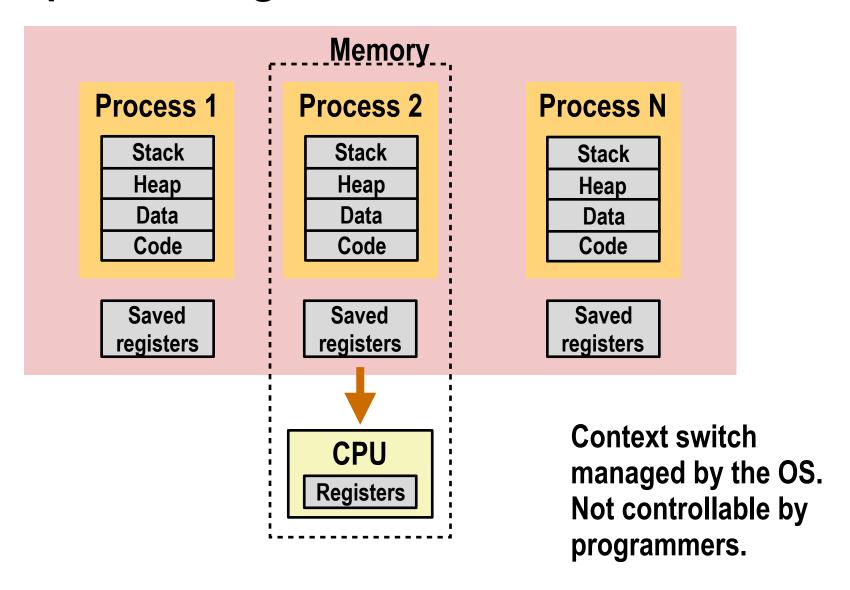




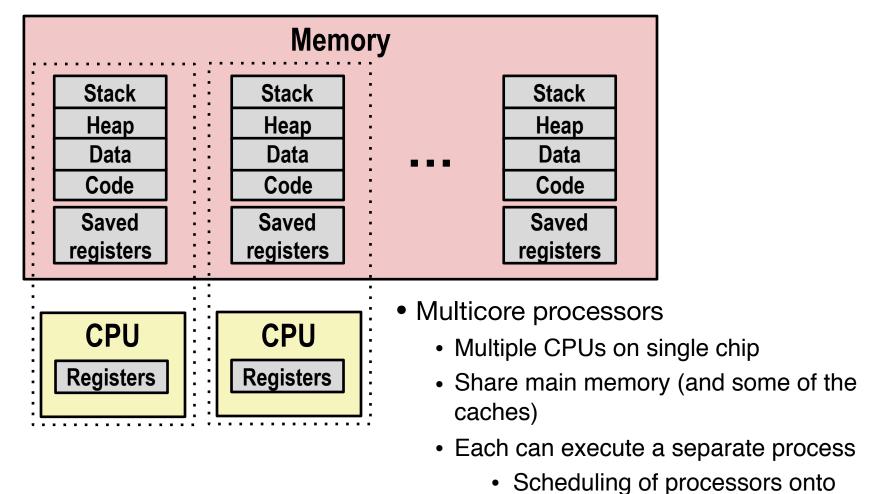








## Multiprocessing: The Multi-Core Case



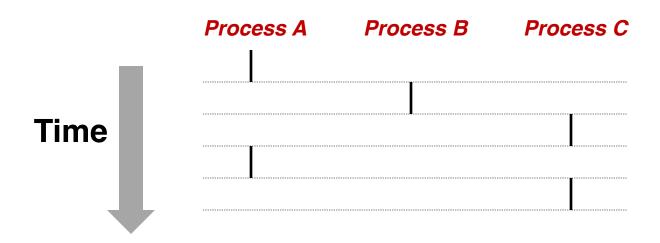
cores done by kernel

#### **Concurrent Processes**

- Each process is a logical control flow.
- Two processes *run concurrently* (are concurrent) if their flows overlap in time
- Otherwise, they are sequential

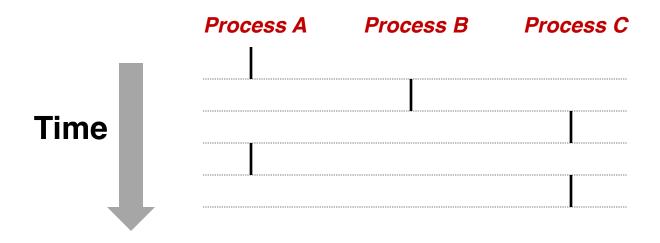
### **Concurrent Processes**

- Each process is a logical control flow.
- Two processes run concurrently (are concurrent) if their flows overlap in time
- Otherwise, they are sequential
- Examples (running on single core):



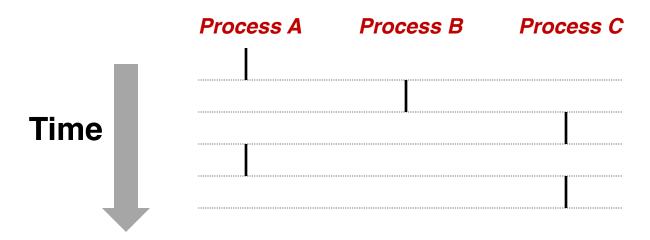
#### **Concurrent Processes**

- Each process is a logical control flow.
- Two processes run concurrently (are concurrent) if their flows overlap in time
- Otherwise, they are sequential
- Examples (running on single core):
  - Concurrent: A & B, A & C



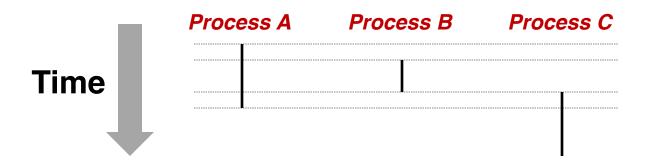
#### **Concurrent Processes**

- Each process is a logical control flow.
- Two processes run concurrently (are concurrent) if their flows overlap in time
- Otherwise, they are sequential
- Examples (running on single core):
  - Concurrent: A & B, A & C
  - Sequential: B & C



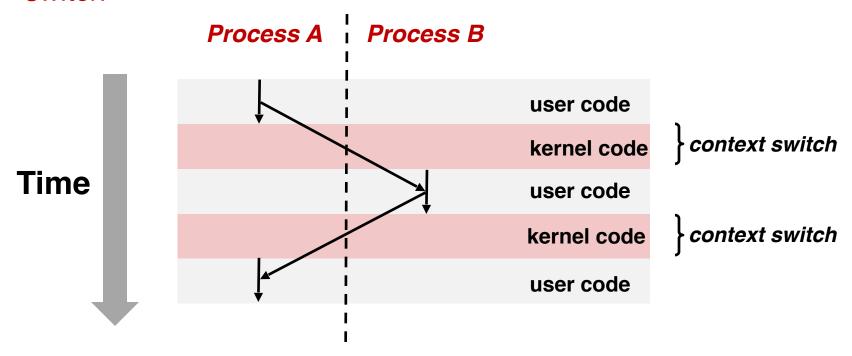
#### **User View of Concurrent Processes**

- Control flows for concurrent processes are physically disjoint in time
- However, we can think of concurrent processes as running in parallel with each other



# **Context Switching**

- Processes are managed by a shared chunk of memory-resident
   OS code called the kernel
  - Important: the kernel is not a separate process, but rather runs as part of some existing process.
- Control flow passes from one process to another via a context switch



# **Today**

- Exceptions/Interrupts
- Processes and Signals: Special kinds of exception
  - Processes
  - Process Control
  - Signals

# **Obtaining Process IDs**

- pid\_t getpid(void)
  - Returns PID of current process
- pid\_t getppid(void)
  - Returns PID of parent process

# **Creating and Terminating Processes**

From a programmer's perspective, we can think of a process as being in one of three states

#### Running

• Process is either executing, or waiting to be executed and will eventually be *scheduled* (i.e., chosen to execute) by the kernel

#### Stopped

 Process execution is suspended and will not be scheduled until further notice (through something call **signals**)

#### Terminated

Process is stopped permanently

#### **Terminating Processes**

- Process becomes terminated for one of three reasons:
  - Receiving a signal whose default action is to terminate
  - Returning from the main routine
  - Calling the exit function
- void exit(int status)
  - Terminates with an exit status of status
  - Convention: normal return status is 0, nonzero on error
  - Another way to explicitly set the exit status is to return an integer value from the main routine
- exit is called once but never returns.

## **Creating Processes**

- Parent process creates a new running child process by calling fork
- int fork(void)
  - Returns 0 to the child process, child's PID to parent process
  - Child is almost identical to parent:
    - Child get an identical (but separate) copy of the parent's (virtual) address space (i.e., same stack copies, code, etc.)
    - Child gets identical copies of the parent's open file descriptors
    - Child has a different PID than the parent
- fork is interesting (and often confusing) because it is called once but returns twice

```
int main()
{
    pid_t pid;
    int x = 1;
    pid = Fork();
    if (pid == 0) { /* Child */
        printf("child : x=%d\n", ++x);
       exit(0);
    /* Parent */
    printf("parent: x=%d\n", --x);
    exit(0);
                                fork.c
```

```
linux> ./fork
parent: x=0
child : x=2
```

```
int main()
{
    pid_t pid;
    int x = 1;
    pid = Fork();
    if (pid == 0) { /* Child */
        printf("child : x=%d\n", ++x);
       exit(0);
    /* Parent */
    printf("parent: x=%d\n", --x);
    exit(0);
                                 fork.c
```

• Call once, return twice

linux> ./fork
parent: x=0
child : x=2

```
int main()
{
    pid_t pid;
    int x = 1;
    pid = Fork();
    if (pid == 0) { /* Child */
        printf("child : x=%d\n", ++x);
       exit(0);
    /* Parent */
    printf("parent: x=%d\n", --x);
    exit(0);
                                 fork.c
```

- Call once, return twice
- Concurrent execution
  - Can't predict execution order of parent and child

```
linux> ./fork
parent: x=0
child : x=2
```

```
int main()
    pid t pid;
    int x = 1;
    pid = Fork();
    if (pid == 0) { /* Child */
        printf("child : x=%d\n", ++x);
       exit(0);
    /* Parent */
    printf("parent: x=%d\n", --x);
    exit(0);
                                 fork.c
```

- Call once, return twice
- Concurrent execution
  - Can't predict execution order of parent and child
- Duplicate but separate address space
  - x has a value of 1 when fork returns in parent and child
  - Subsequent changes to x are independent

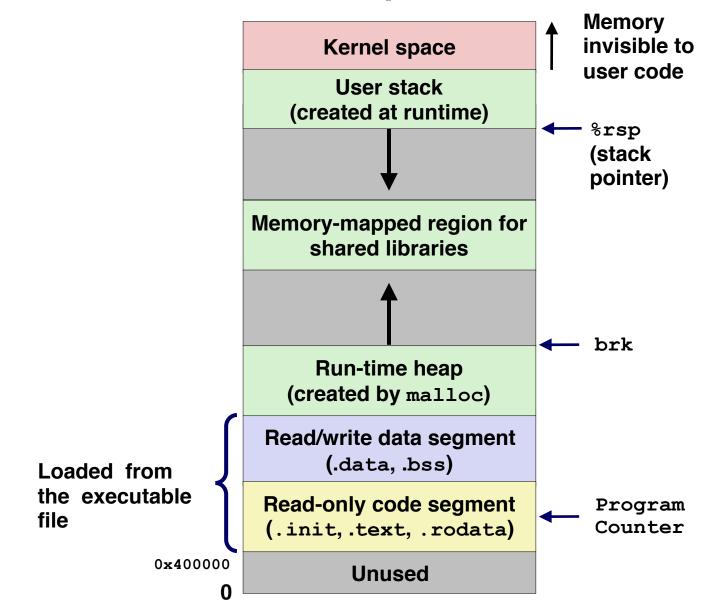
```
linux> ./fork
parent: x=0
child : x=2
```

```
int main()
    pid t pid;
    int x = 1;
    pid = Fork();
    if (pid == 0) { /* Child */
        printf("child : x=%d\n", ++x);
       exit(0);
    /* Parent */
    printf("parent: x=%d\n", --x);
    exit(0);
                                 fork.c
```

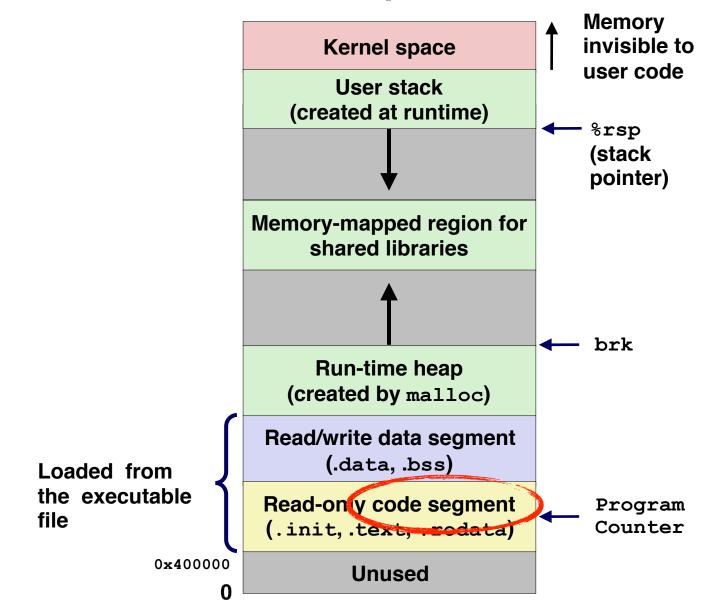
```
linux> ./fork
parent: x=0
child : x=2
```

- Call once, return twice
- Concurrent execution
  - Can't predict execution order of parent and child
- Duplicate but separate address space
  - x has a value of 1 when fork returns in parent and child
  - Subsequent changes to x are independent
- Shared open files
  - stdout is the same in both parent and child

# **Process Address Space**



# **Process Address Space**

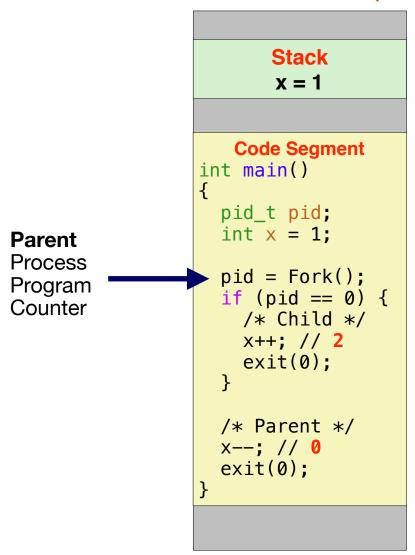


```
Code Segment
int main()
  pid_t pid;
  int x = 1;
  pid = Fork();
  if (pid == 0) {
   /* Child */
   x++; // 2
    exit(0);
 /* Parent */
 x--; // 0
 exit(0);
```

#### **Parent** Address Space

```
Stack
       x = 1
   Code Segment
int main()
  pid_t pid;
  int x = 1;
  pid = Fork();
  if (pid == 0) {
   /* Child */
    x++; // 2
    exit(0);
  /* Parent */
 x--; // 0
 exit(0);
```

#### **Parent** Address Space



#### Parent Address Space Child Address Space

#### Stack x = 1**Code Segment** int main() pid\_t pid; int x = 1: pid = Fork(); if (pid == 0) { /\* Child \*/ X++; // 2exit(0);

/\* Parent \*/

x--; // 0

exit(0);

**Parent** Process

Program

Counter

```
Stack
       x = 1
   Code Segment
int main()
 pid_t pid;
 int x = 1;
 pid = Fork();
 if (pid == 0) {
   /* Child */
    X++; // 2
    exit(0);
 /* Parent */
 x--; // 0
 exit(0);
```

#### Parent Address Space

#### Stack x = 1**Code Segment** int main() pid\_t pid; int x = 1; **Parent** Process pid = Fork(); Program if (pid == 0) { Counter /\* Child \*/ X++; // 2exit(0); /\* Parent \*/ x--; // 0 exit(0);

#### **Child** Address Space

```
Stack
       x = 1
   Code Segment
int main()
  pid_t pid;
 int x = 1;
                            Child
                            Process
  pid = Fork();
                            Program
  if (pid == 0) {
                            Counter
    /* Child */
    X++; // 2
    exit(0);
  /* Parent */
 x--; // 0
 exit(0);
```

## **Creating Processes**

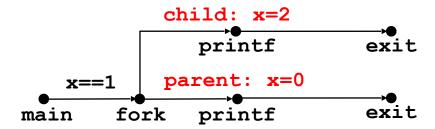
- Parent process creates a new child process by calling fork
- Child get an identical (but separate) copy of the parent's (virtual) address space (i.e., same stack copies, code, etc.)
- int fork(void)
  - Returns 0 to the child process
  - Returns **child's PID** to the parent process

# **Process Graph Example**

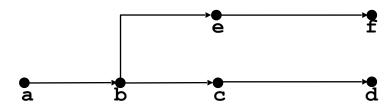
```
int main()
{
    pid_t pid;
    int x = 1;
                                                      child: x=2
                                                                        Child
    pid = Fork();
                                                         printf
                                                                  exit
    if (pid == 0) { /* Child */
                                                      parent: x=0
        printf("child : x=%d\n", ++x);
                                              x==1
                                                                       Parent
       exit(0);
                                                  fork
                                                         printf
                                                                  exit
                                          main
    }
    /* Parent */
    printf("parent: x=%d\n", --x);
    exit(0);
                                 fork.c
```

## Interpreting Process Graphs

Original graph:

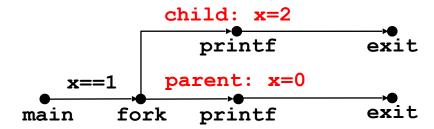


Abstracted graph:

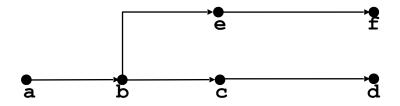


## Interpreting Process Graphs

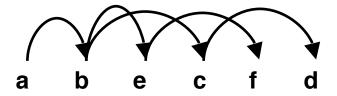
• Original graph:



Abstracted graph:

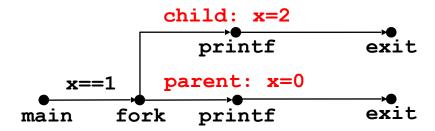


Feasible execution ordering:

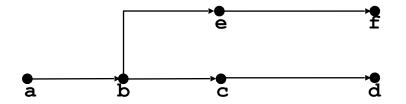


## Interpreting Process Graphs

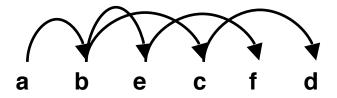
Original graph:



Abstracted graph:



Feasible execution ordering:



Infeasible execution ordering:

