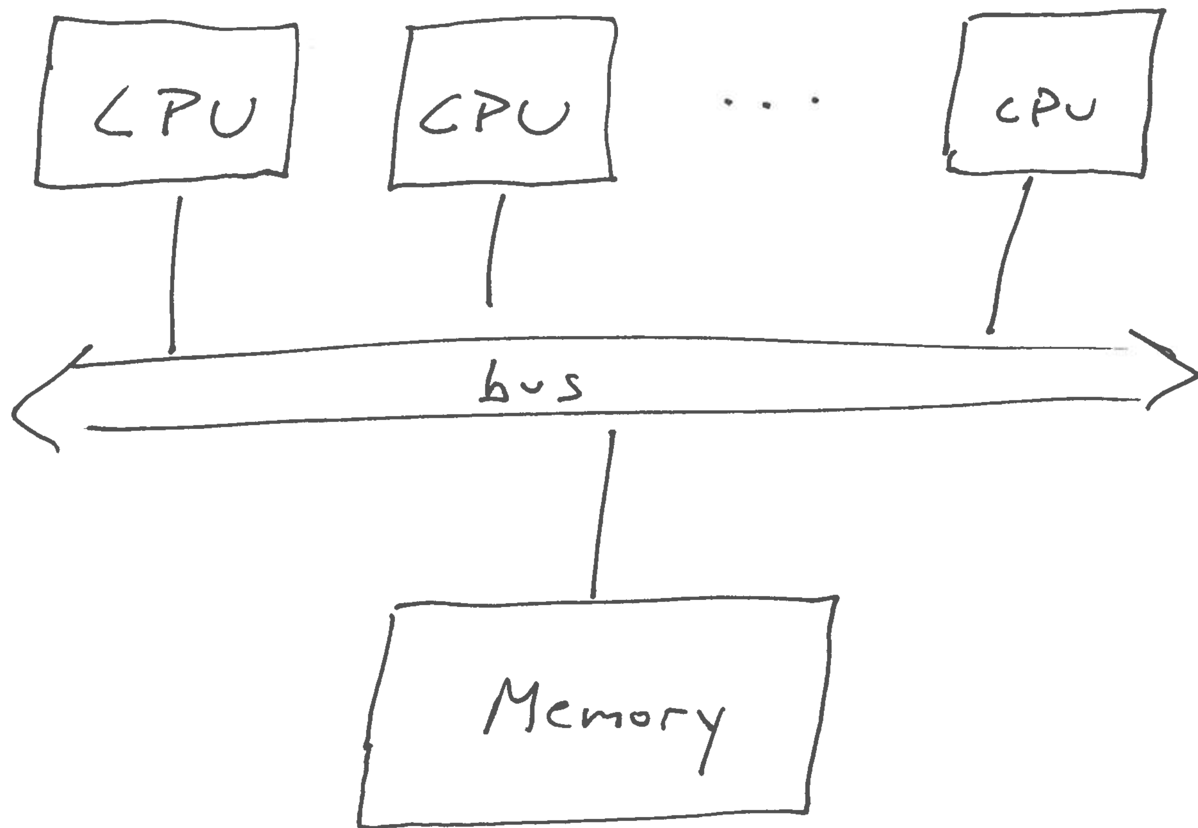
Threads

CS520

Dept. of Computer Science
Univ. of New Hampshire


CPU
CPU
· · ·
CPU
bus
Memory

thread: a virtual CPU

executes instructions

has registers

uses stack

useful for performance – exploit multiple CPUs

useful for design – e.g. assign thread to wait for GUI events while another thread does main computation

But

memory is shared

threads need to contend for it

threads can interfere with each other

Consider two threads incrementing the same variable.

$i = \emptyset$

T_0	T_1
i++	i++

$i = ?$

First, what is i++ actually? — not atomic

load i

add 1

store i

Need to consider interleavings of the low-level instructions:

$i = \emptyset$

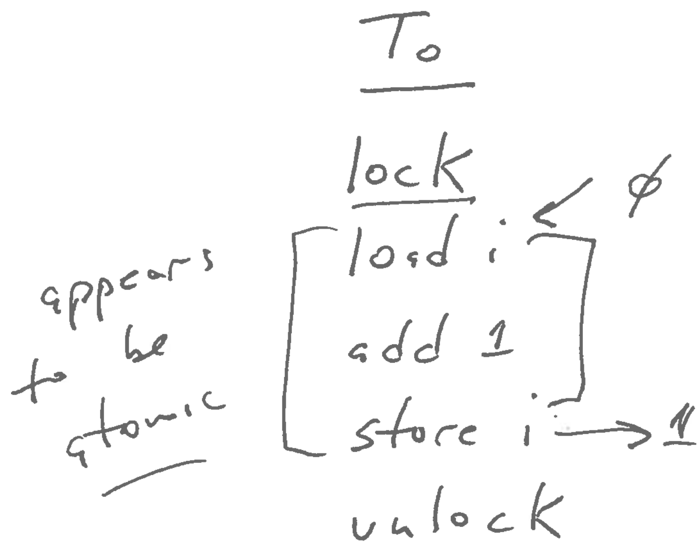
T_0	T_1
$\emptyset \leftarrow$ load i	
add 1	
store i → 1	
	load i → $\emptyset$
	add 1
	store i → 1

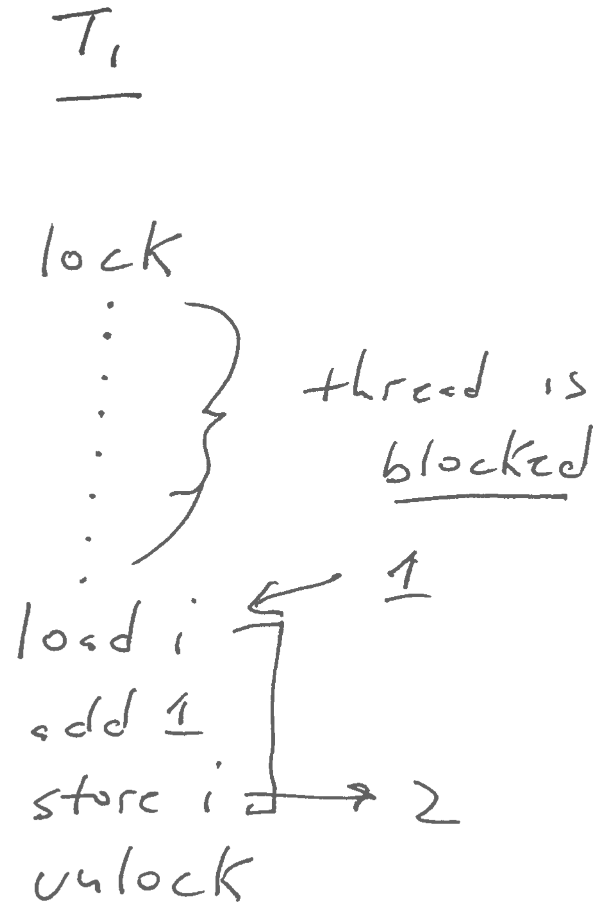
$i = \cancel{2}\ 1$

race condition: possibility of incorrect results due to timing

need mechanism to ensure only one thread at a time is in the <u>critical</u> <u>section</u>:

$i = 0$





$i = \not{1}\ 2$

Also need mechanism to control interaction of threads.

For example, consider a thread producing a series of values to be consumed by another thread.

consumer thread can't consume until producer produces value

producer thread can't produce until last value consumed

assuming a buffer of length 1

<u>shared</u> <u>data</u>

valuePresent (boolean)
value } initialized to false

<u>also</u> <u>shared</u>

mu - a lock (mutex)
cv - a condition variable

<u>producer</u>

```
lock(mu)
while (if) (valuePresent) {
    wait(cv, mu)
}
value = newValue;
valuePresent = true;
signal(cv)
unlock(mu)
```

<u>consumer</u>

```
lock(mu)
while (if) (!valuePresent) {
    wait(cv, mu)
}
newValue = value;
valuePresent = false;
signal(cv)
unlock(mu)
```

<u>note</u>: when wait is called, lock is released (caller must own lock)
when signal is called, ~~first~~ a single waiter is released
<u>but</u> <u>newly</u> <u>released</u> <u>waiter</u> <u>must</u> <u>wait</u> <u>to</u> <u>re-acquire</u> <u>lock</u>

bad things can happen:

<u>race condition</u>

<u>deadlock</u> – all threads are blocked

<u>livelock</u> – threads spend so much time responding to each other's actions that they cannot make progress on their work

<u>starvation</u> – some threads dominate access to shared resource so that other threads cannot access it