Hello

Virtualization ←
⇒ Concurrency ←
Persistence

Bugs

⇒ Atomicity violation
(forgot to use locks)

⇒ Ordering violations
(forgot to use cond. variable)
e.g. use before init.

Deadlock

→ what is it?
→ How it arises?
→ How to avoid?

two locks: m_1, m_2

simple example:

T_1	T_2
(held)	(held)
— acquire(m_1)] hold	— acquire(m_2) hold
acquire(m_3)	
try — acquire(m_2) wait	try — acquire(m_1) wait

state:

m_1	held
m_2	held

$$\begin{tikzcd} & T_1 \arrow[dl, "holds"'] & \\ m_1 \arrow[dr, "wait"'] & & m_2 \arrow[ul, "wait"'] \\ & T_2 \arrow[ur, "holds"'] & \end{tikzcd}$$

Conditions:

=> 'Mutual exclusion'

=> Hold + wait

=> No preemption (of lock) (unlock)
once held, held until release is called

=> Circular wait

Use avoidance to prevent deadlock
(contrast: detect + recover)

Best solution:
=> Lock Ordering

T_1	T_2
m_1	m_1
m_2	m_2

Hold + wait:
acquire one lock, ✓
then try to acquire another one ✗

acquire(lock)
acquire(m_1)]
acquire(m_2)
release(lock)

acquire(lock)
acquire(m_2)
acquire(m_1)
release(lock)

surround lock acquisition
w/ global lock/unlock

Test (tomorrow)
⇒ (opposite room)
⇒ closed book/notes, helper sheet
Review: (7pm in 1240)

old: acquire lock
(may) either stuck waiting
OR
return w/ lock held

new: "try" lock

try to acquire lock

try to acquire lock
if can't, return (w/ error code)
(immediately)
if can, acquire + return

Approach:
use try locks

Problems:
complex (undo ...)
inefficient
livelock: might get stuck "trying" to get locks

"lock free" approach.

```
list_insert(int value) {
    node_t *n = malloc(sizeof(node_t));
    assert(n != NULL);
    n -> value = value;
    n -> next = head;   // insert at head
    head = n;
}
```