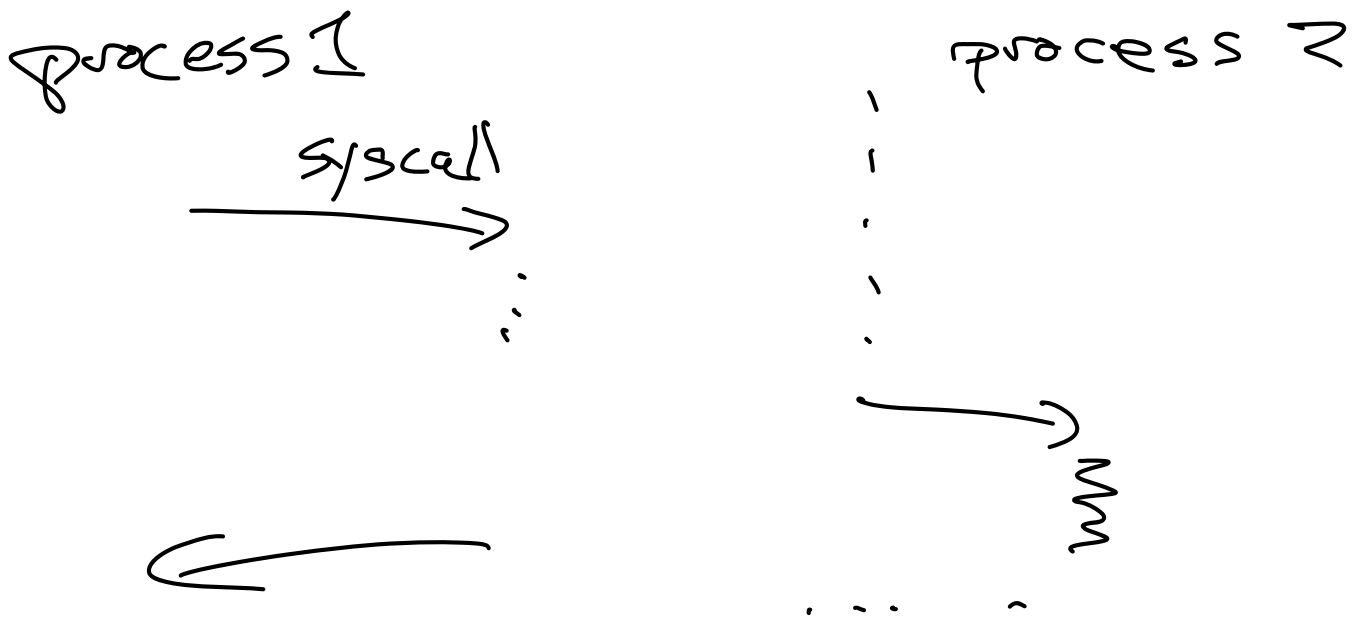
CS 3650 – Computer Systems
Spring 2024
Peter Desnoyers

Lecture ~~11~~ 12, Tue ~~Feb 13~~ Thur 15 2024

Threads & concurrency

process = address space
OS state (open files, current dir, ...)
thread of control (stack)



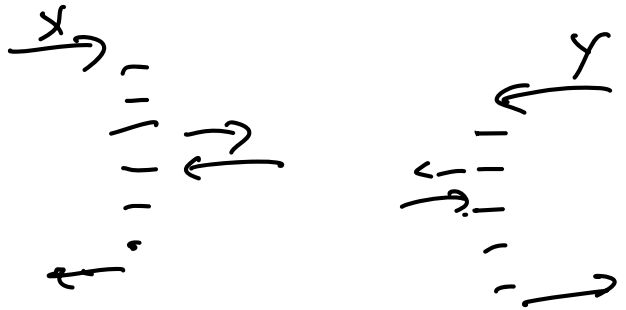
Threads:

same address space
same OS ~~state~~
different stack

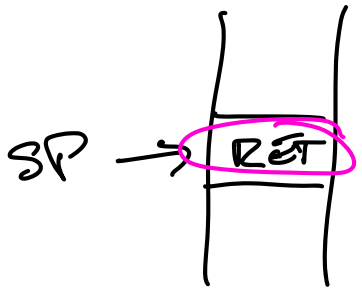
why threads?

multi-core: use >1 CPU
same addr space
"work while blocking"

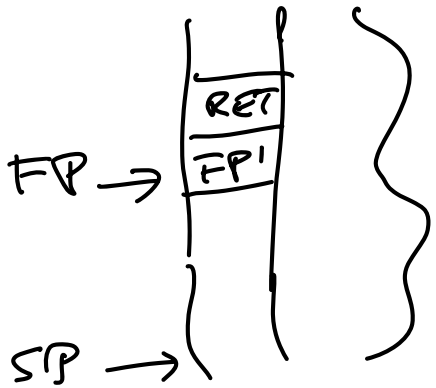
→ manage multiple asynchronous things



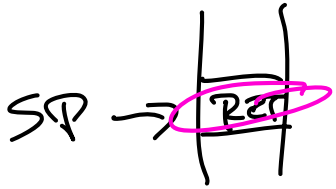
How? (single-core version)



entering function



middle



RET

```
{
f();
}
```

⇒

```
instr
instr
CALL f
instr
instr
⋮
```

RET addr

```
f(): PUSH FP
     FP=SP
     ...
     instrs that do f
     SP=FP
     POP FP
     RET
```

Thread 1

Thread 2

{

yield 1→2 ⟶

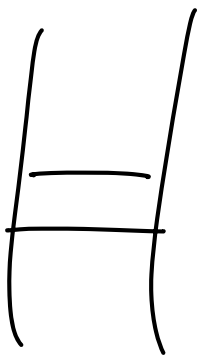
STORE SP → saved_SP_1

LOAD SP ← savedSP_2

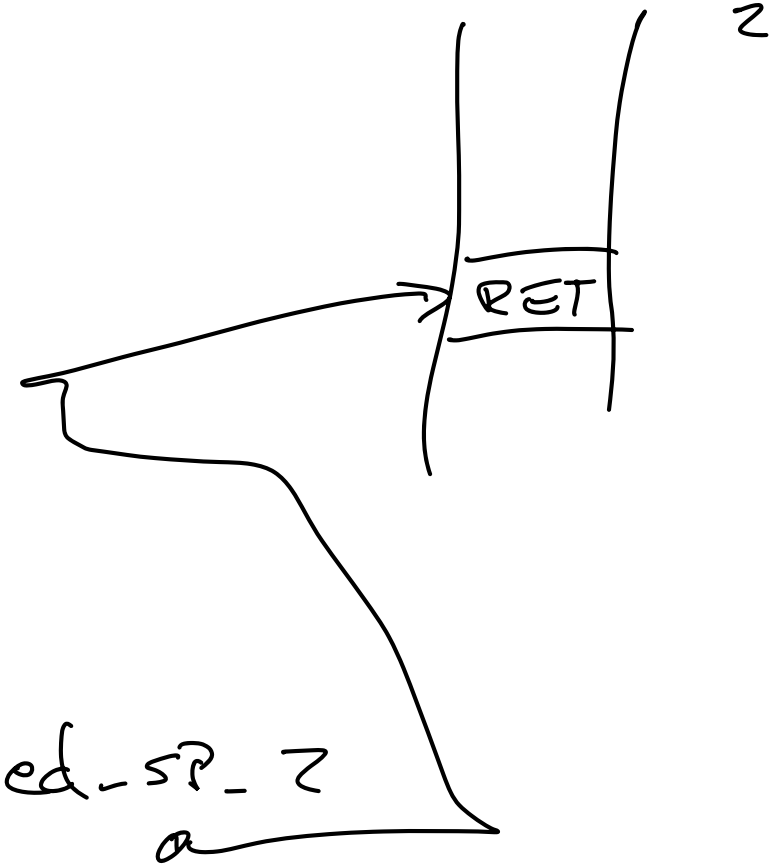
RET

stack 1

SP ⟶



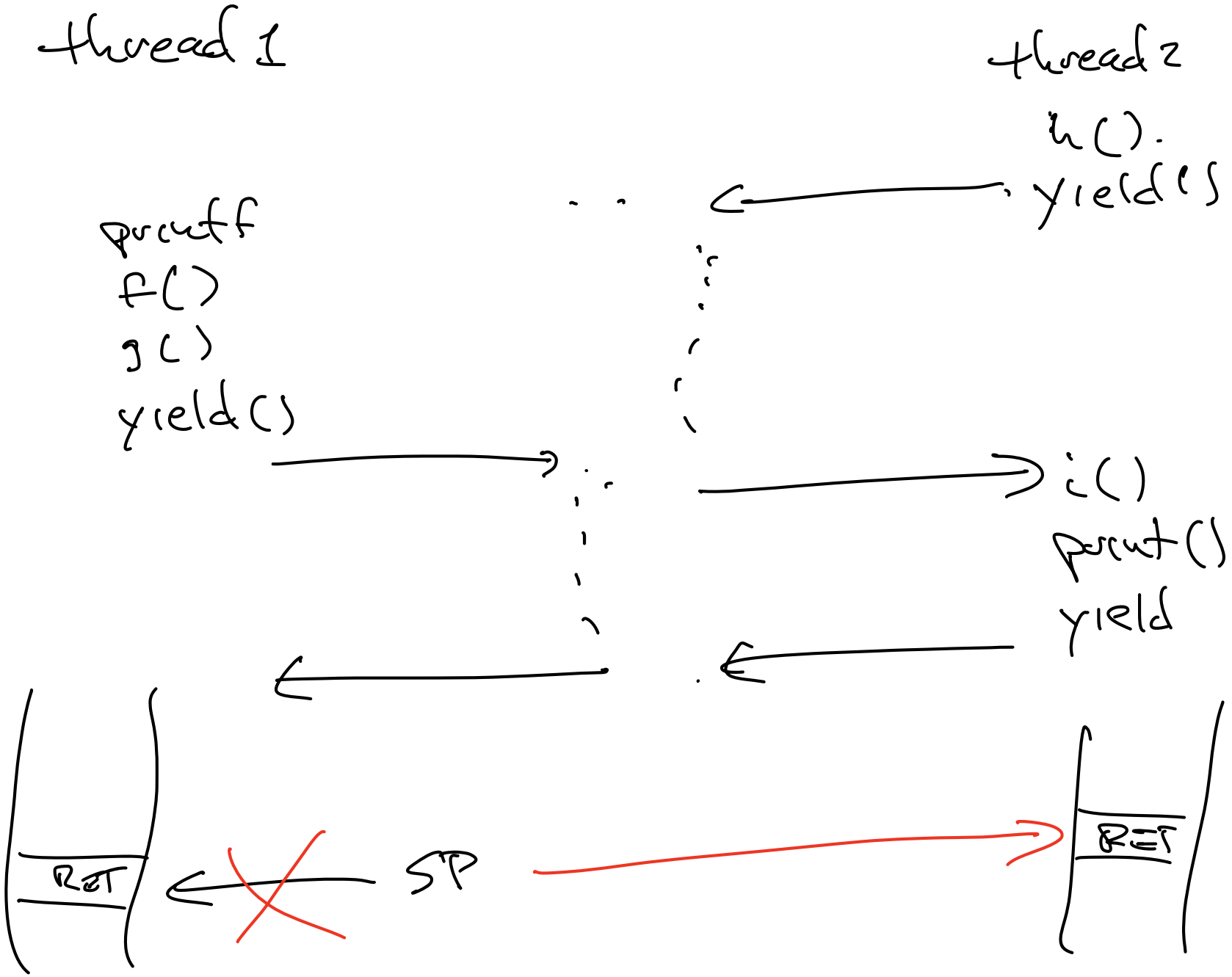
stack 2

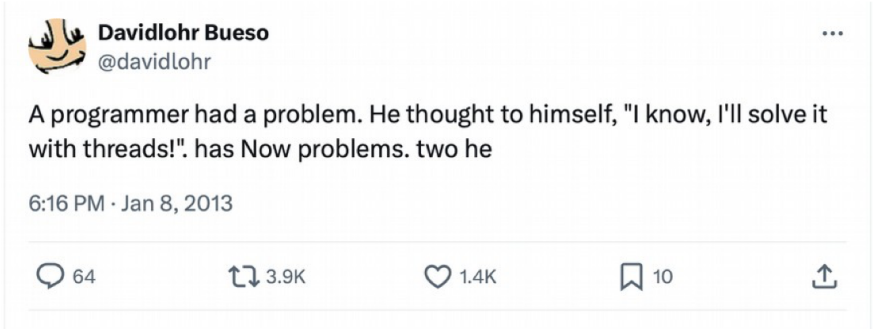


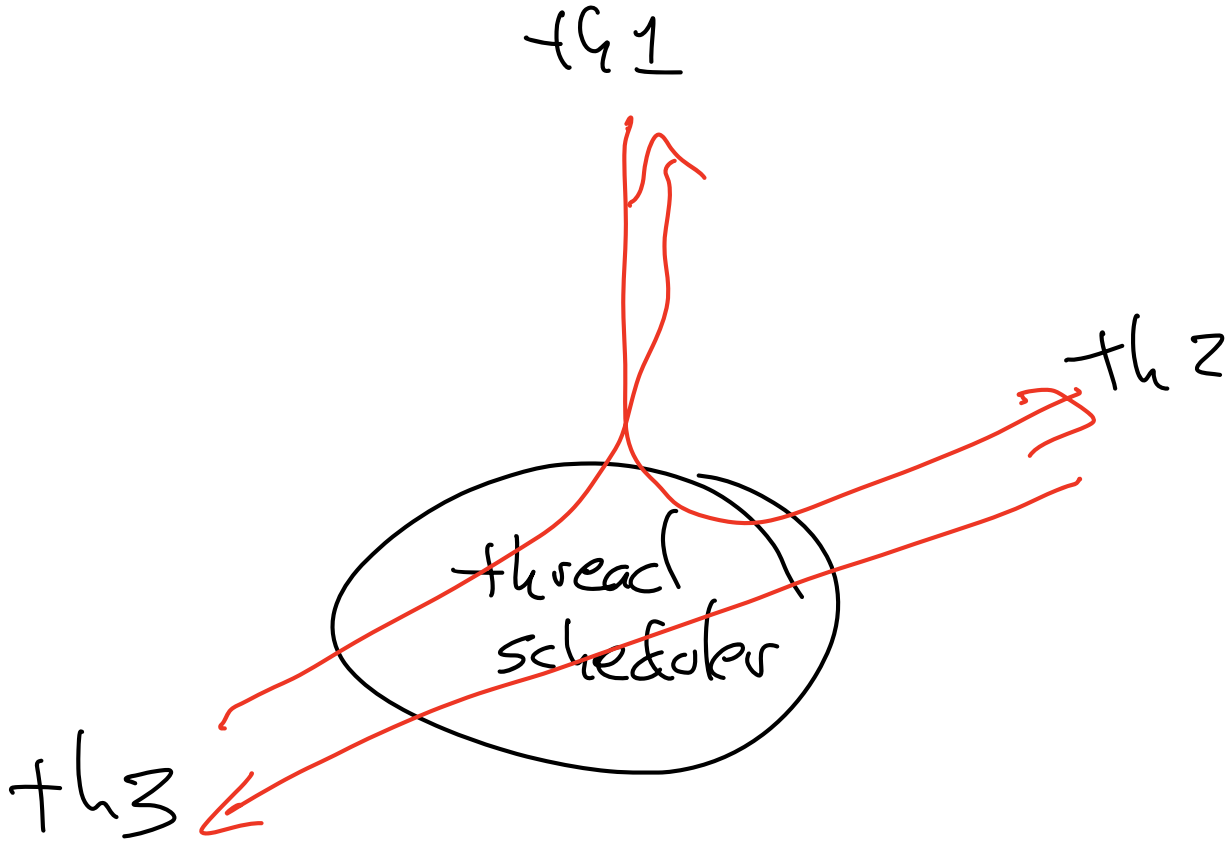
variable :

saved_SP_1

saved_SP_2


thread 1
thread 2
h()
yield()
printf
f()
g()
yield()
i()
print()
yield
RET
SP
RET

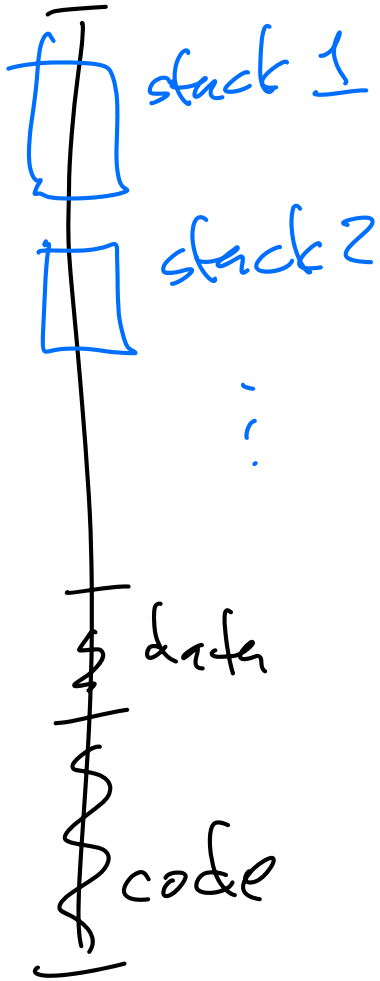

Davidlohr Bueso
@davidlohr
A programmer had a problem. He thought to himself, "I know, I'll solve it with threads!". has Now problems. two he
6:16 PM · Jan 8, 2013
64
3.9K
1.4K
10

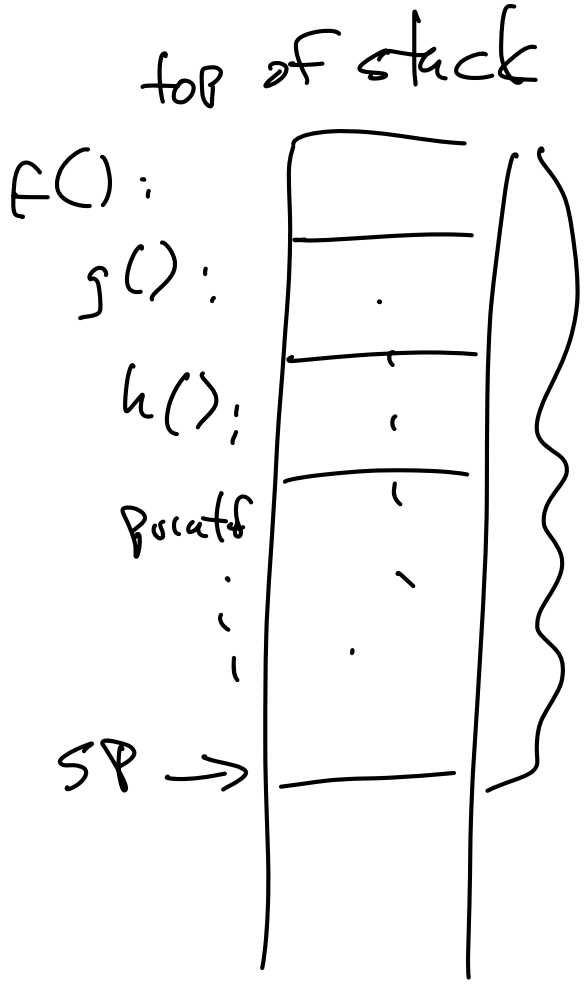


th 1 :
- send DB req
- wait for result
- send xyz req
- wait for result

done

th 2 :
- send network lookup
- wait result
- send DB req
- wait result
- :


stack 1
stack 2
data
code


top of stack
f():
g():
h();
printf
SP

How do we do this, really?

POSIX threads

pthread_t ← "thread handle"

pthread API weirdnesses:

int operation (type *return_val, attr *attrs)

← success/failure

NULL

pthread_t th;

attr ↓

pthread_create(&th, NULL, thread_fn, thread_arg)

↑ void*

void *thread_fn(void *arg)

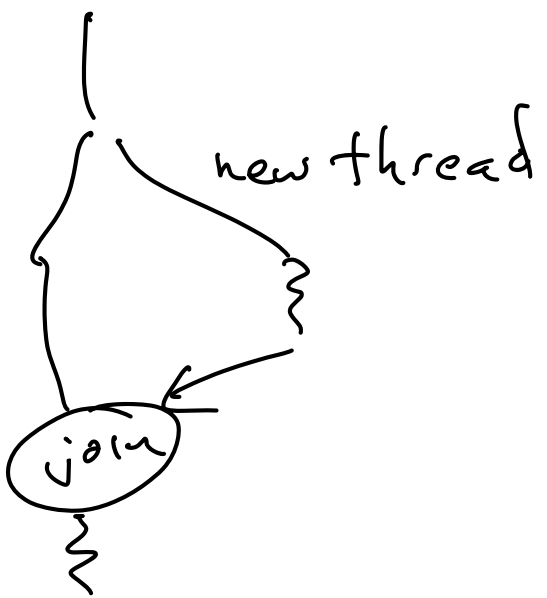
```
void * simple_thread (void * iguove) {
    printf("thread\n");
    return NULL;
}
```

```
main() {
    pthread_create(&th)
    void *retval
    pthread_join(th, &retval)
}
```

thread fn:

→ return value

→ pass value to pthread_exit()



Race conditions

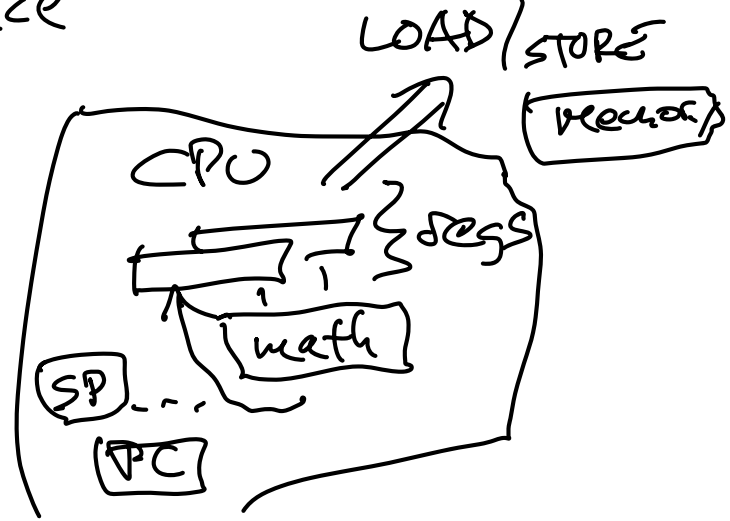
simple bank account

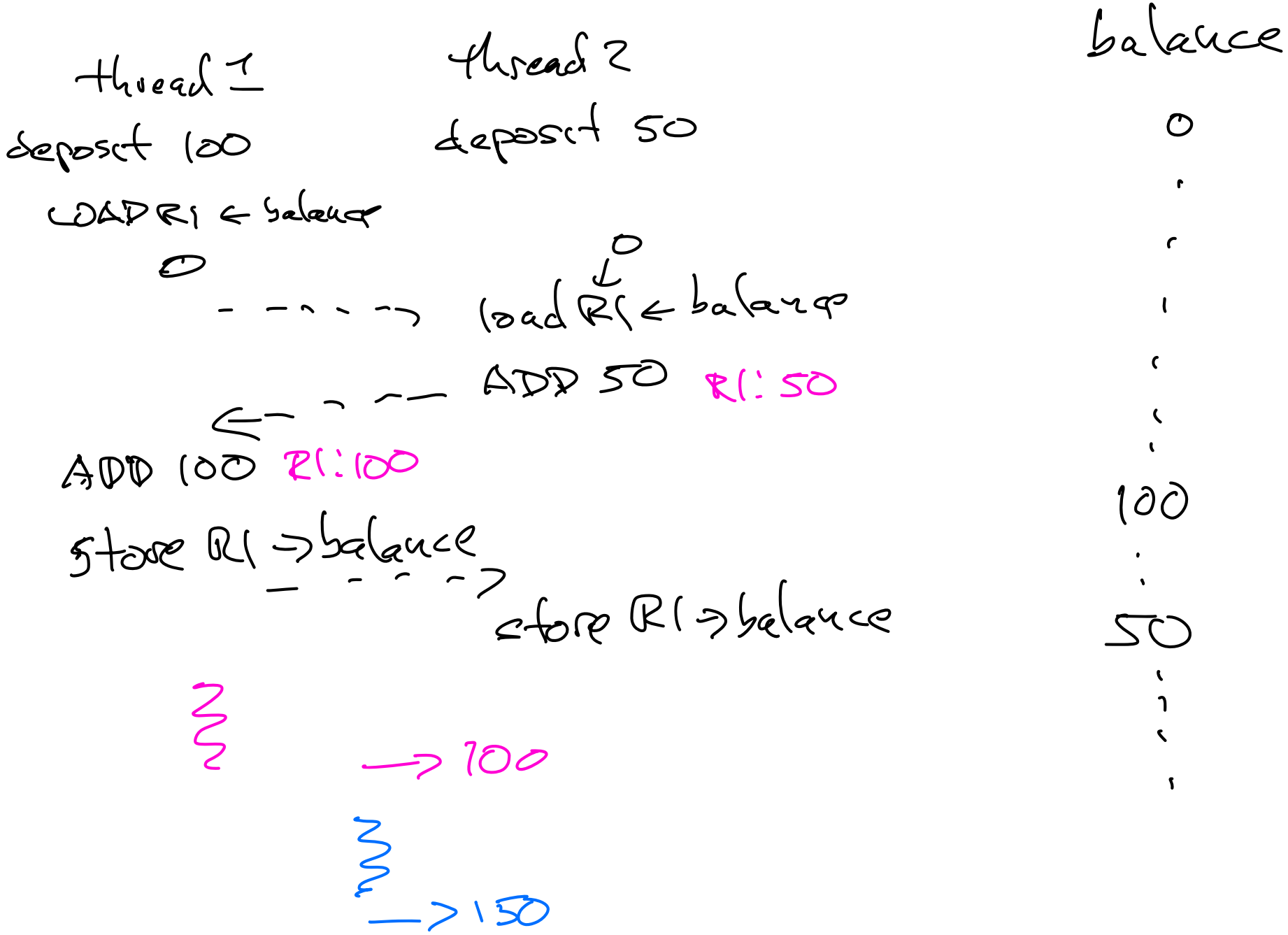
thread 1
deposit 100
LOAD R1 ← balance
ADD 100 R1: 100
store R1 → balance

thread 2
deposit 50
load R1 ← balance
ADD 50 R1: 50

class account
int balance
method deposit (sum)
{ balance = balance + sum

LOAD R1 ← balance
LOAD R2 ← sum
ADD R1+R2 → R3
STORE R3 → balance




thread 1
thread 2
balance
deposit 100
deposit 50
LOAD R1 ← balance
0
load R1 ← balance
ADD 50
R1: 50
ADD 100
R1: 100
store R1 → balance
store R1 → balance
100
50
→ 100
→ 150

function invariants

invariant: property true of

class
< complicated stuff >
method A
method B
...

bank_account_with_pennies

int dollars
int cents

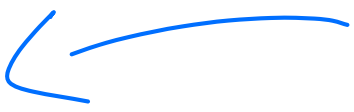
invariant:
$0 \leq cents < 100$

cents = 371

:

deposit (d, c)

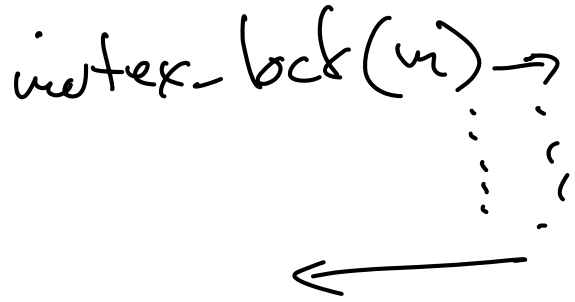
dollars += d
cents += c
if cents >= 100
    dollars +=
    cents = cents - 100



solution : mutex

mutex_lock(m)

mutex_unlock(m)



th1

th2