



Real-time Systems SMD138

Lecture 4:

The inner workings of a kernel
(Burns/Wellings ch 7.3.1 only)

Scope

- To demystify the concept of "concurrent" execution on a sequential processor
- To provide initial help with lab assignment 2
- To set the scene for our forthcoming model of reactive concurrent programming

The infamous goto

```
A;  
while (1) {  
    B;  
    if (e) goto label;  
    C;  
}  
label:  
D;
```

Classic code

```
A;  
while (1) {  
    B;  
    if (e) break;  
    C;  
}  
D;
```

Equivalent code,
structured programming

Non-local goto?

```
A;  
while (1) {  
    fun();  
}  
label:  
D;  
...  
  
void fun() {  
    int x[1000];  
    B;  
    if (e) goto label;  
    C;  
}
```

Can't jump to invisible label

```
A;  
while (1) {  
    try fun();  
    catch(E) break;  
}  
D;  
...  
  
void fun() {  
    int x[1000];  
    B;  
    if (e) throw(E);  
    C;  
}
```

Can't just leave x on stack

Note: Java/C++ code - not valid in C

A non-local goto!

```
#include <setjmp.h>
```

```
jmp_buf bf;
```

```
A;  
while (1) {  
    if (setjmp(bf)) break;  
    fun();    Defines label "bf"  
}  
D;
```

```
void fun() {  
    int x[1000];  
    B;  
    if (e) longjmp(bf, 1);  
    C;    Jumps to label "bf"  
}
```

Setjmp/longjmp

- Save context (code-pointer and stack-pointer) in `buf`, then return 0
`setjmp(buf);`
- Load context from `buf`, then return `n`
`longjmp(buf, n);`
- Note: `longjmp()` returns its value after the registers have been restored; it will thus look as if it is `setjmp()` that returns `n`
- Note 2: a `jmp_buf` is known to be an array - no `&` operator needed

Ordinary function calls

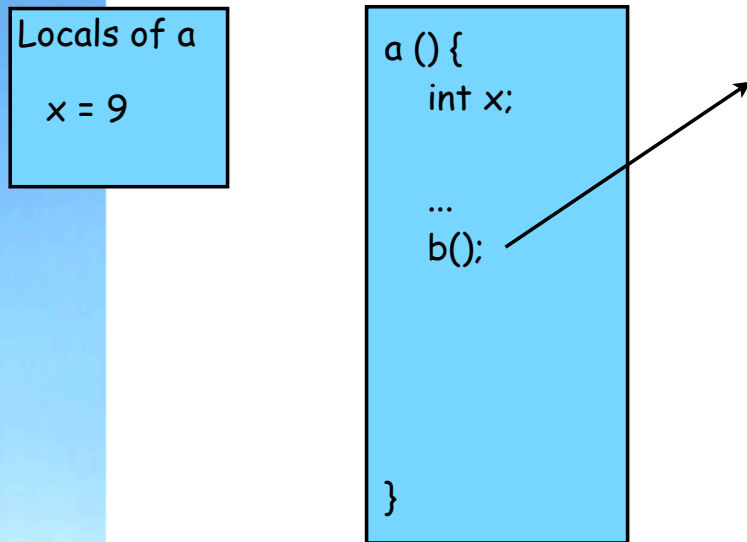
Locals of a
x = 9

```
a(){  
  int x;  
  
  ...  
  
}
```

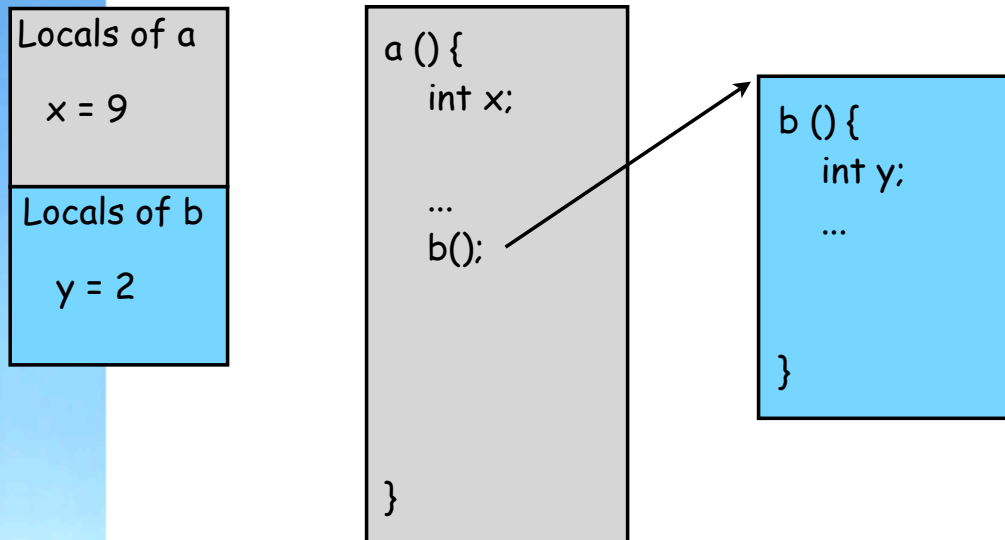
Ordinary function calls

Locals of a
x = 9

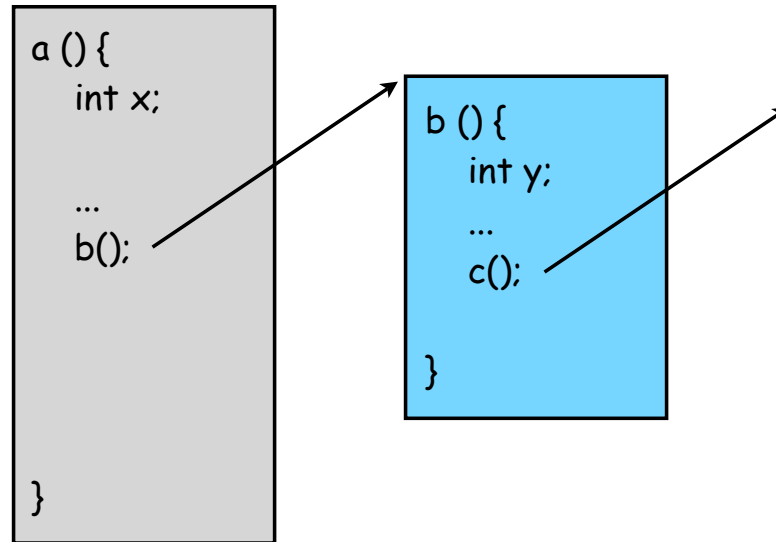
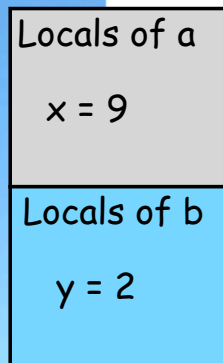
```
a() {  
  int x;  
  
  ...  
  b();  
  
}
```



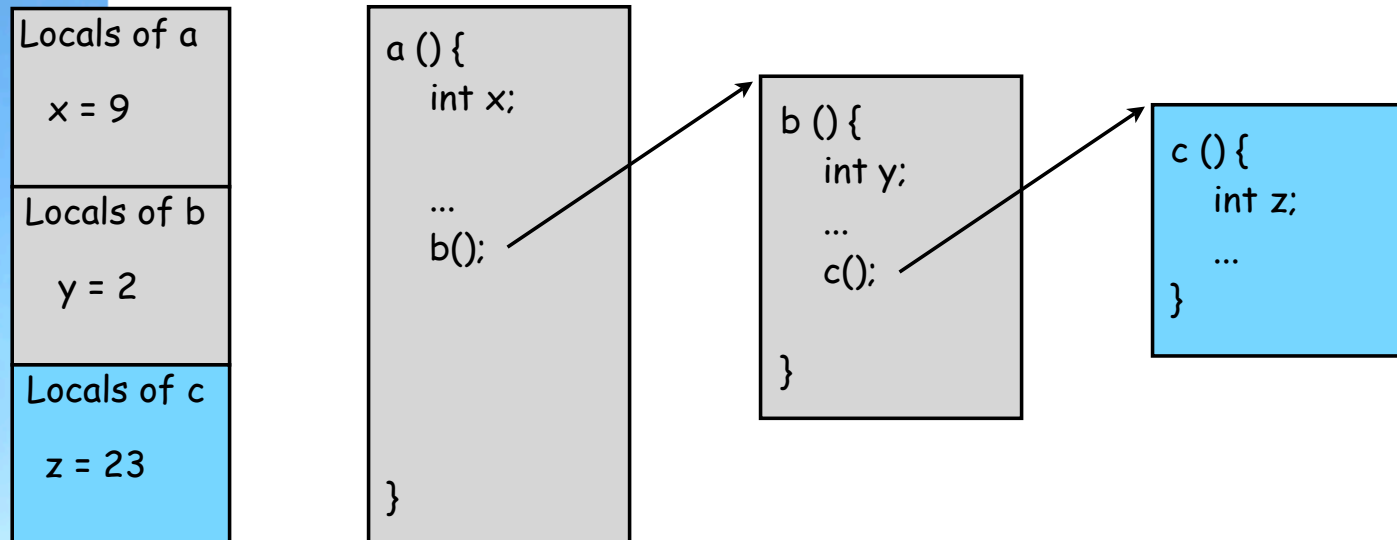
Ordinary function calls



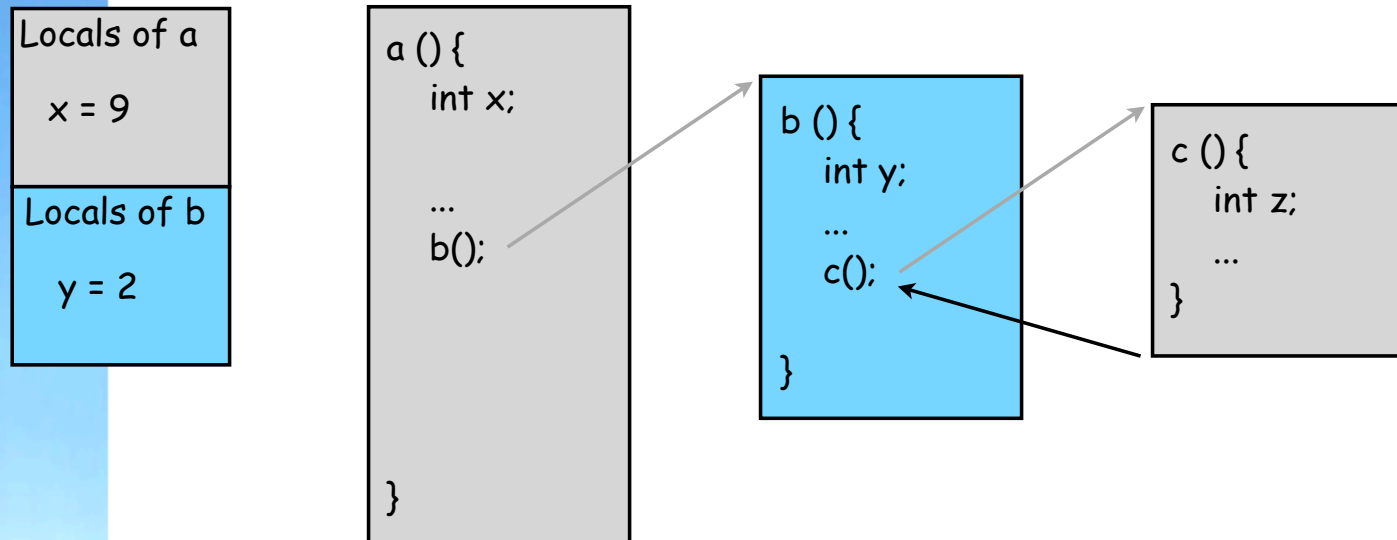
Ordinary function calls



Ordinary function calls

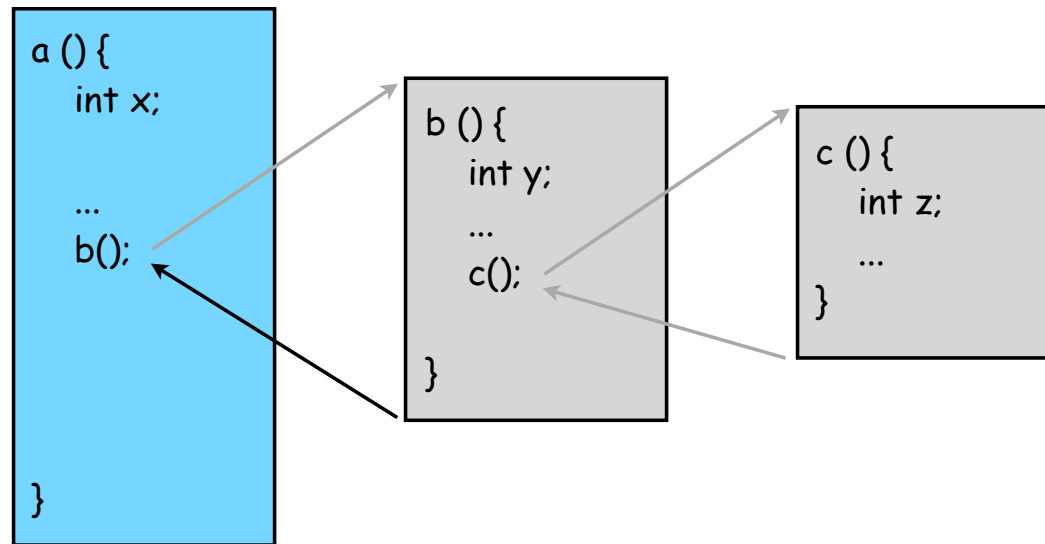


Ordinary function calls



Ordinary function calls

Locals of a
x = 9



Using setjmp/longjmp

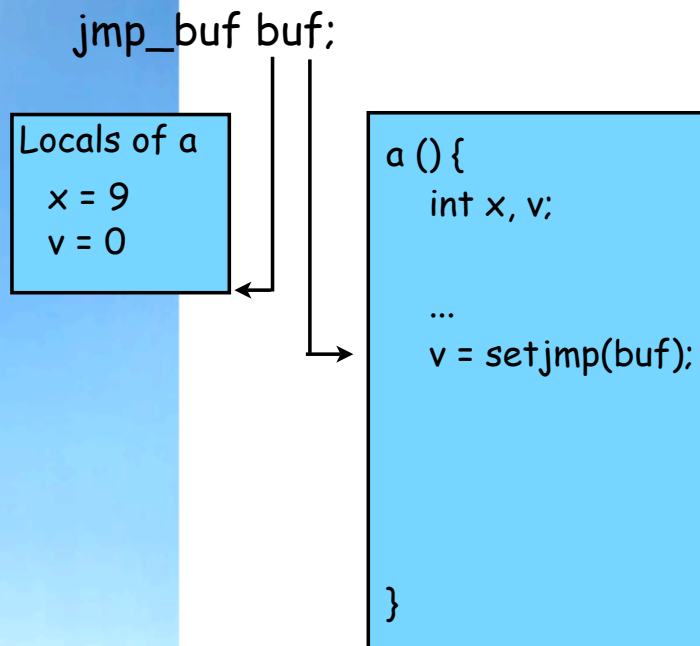
```
jmp_buf buf;
```

Locals of a

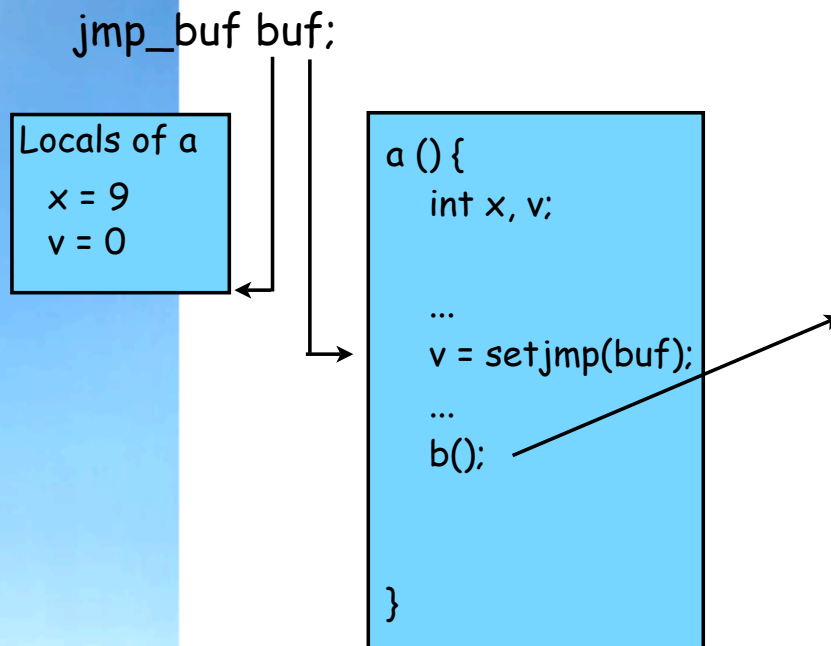
```
x = 9  
v =
```

```
a(){  
  int x, v;  
  
  ...  
  
}
```

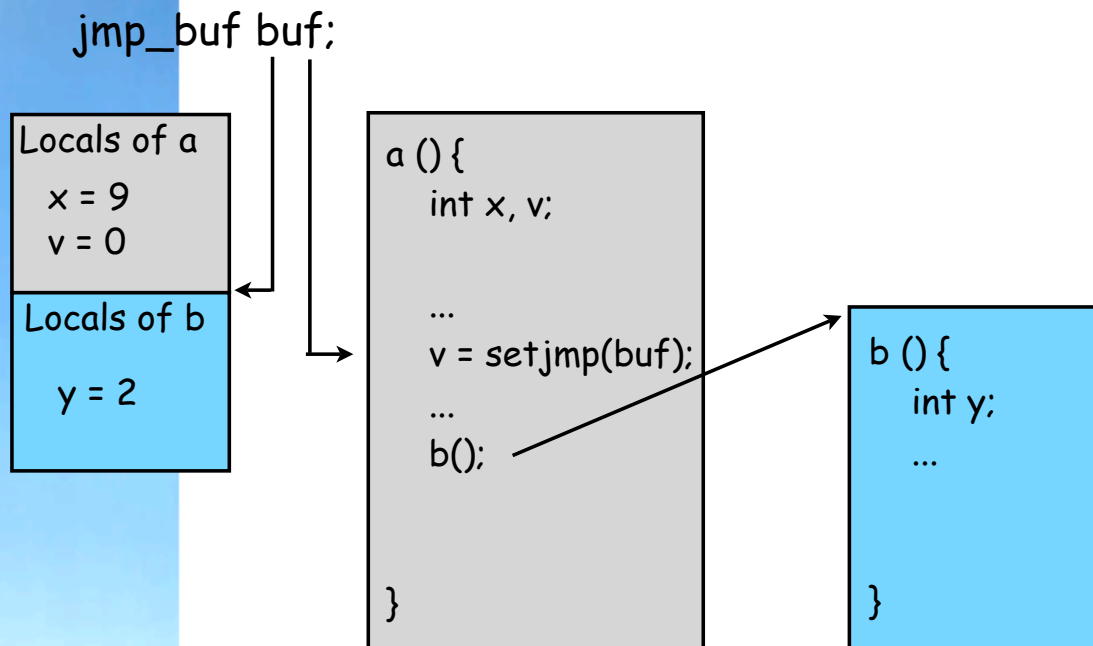
Using setjmp/longjmp



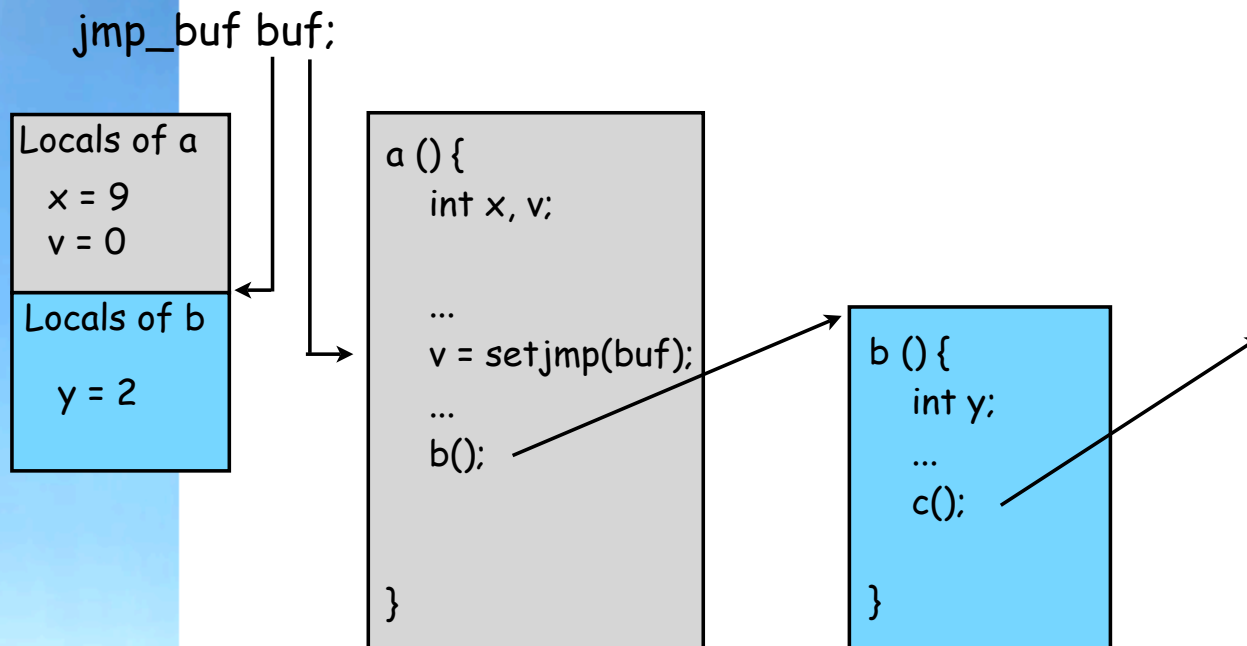
Using setjmp/longjmp



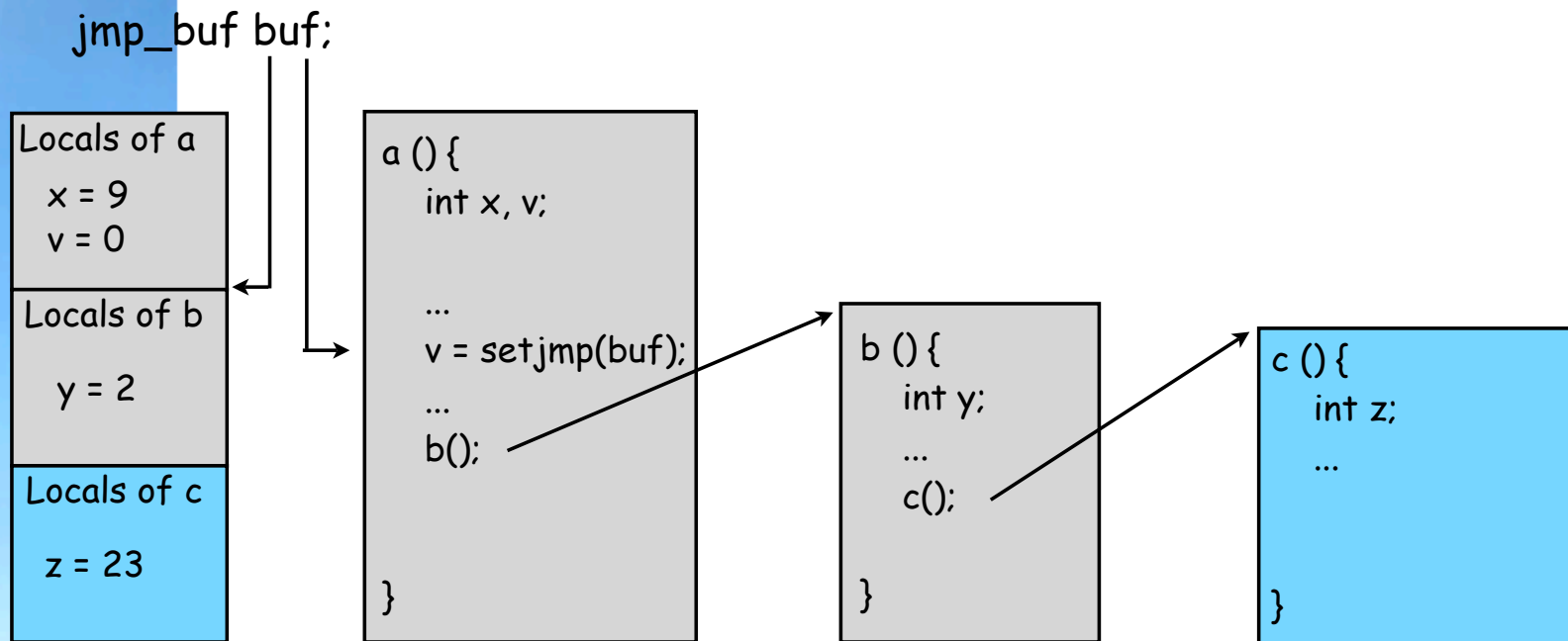
Using setjmp/longjmp



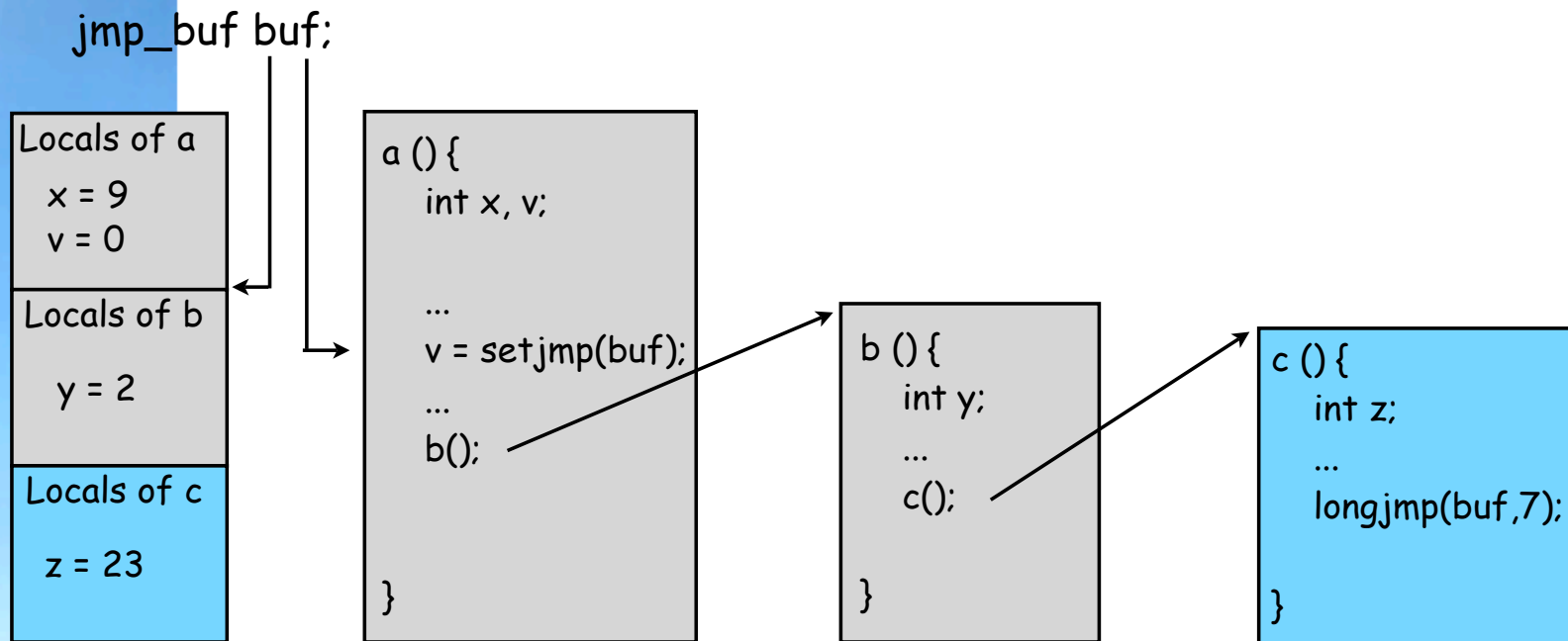
Using setjmp/longjmp



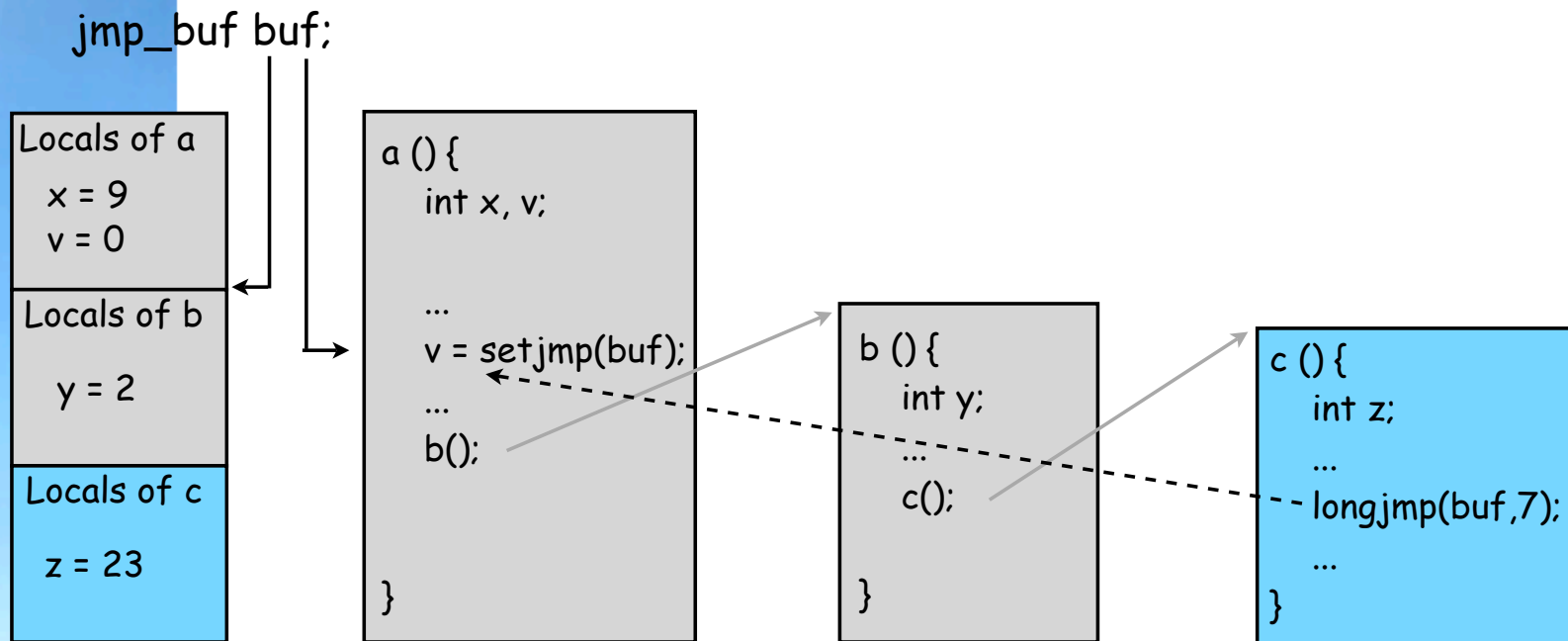
Using setjmp/longjmp



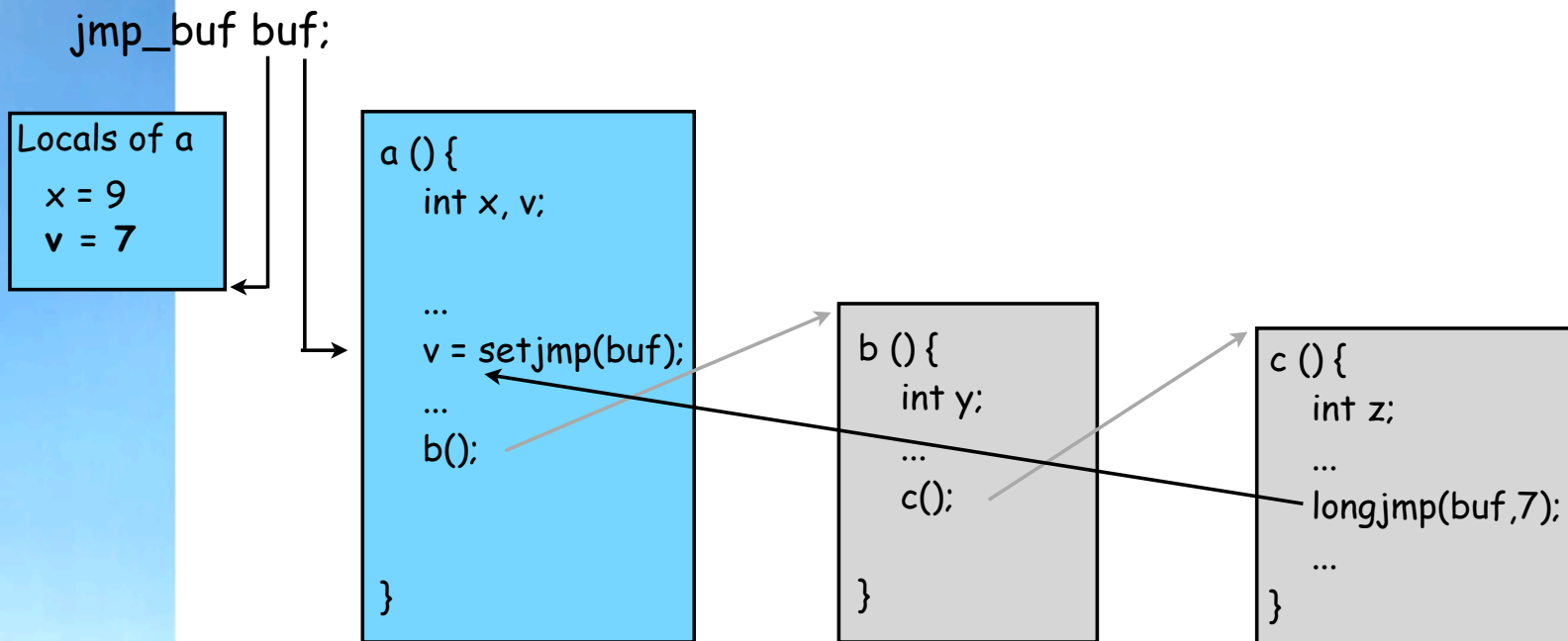
Using setjmp/longjmp



Using setjmp/longjmp



Using setjmp/longjmp



Misusing setjmp/longjmp

```
jmp_buf buf;
```

```
Locals of a  
x = 9
```

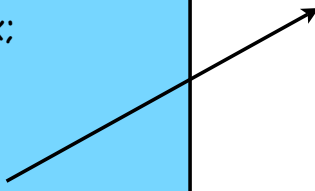
```
a(){  
  int x;  
  
  ...  
  
}
```

Misusing setjmp/longjmp

```
jmp_buf buf;
```

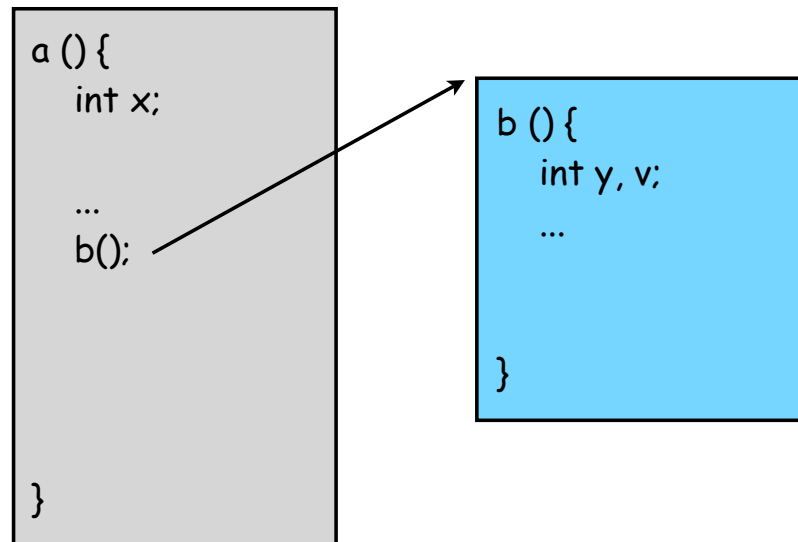
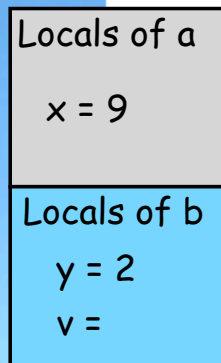
```
Locals of a  
x = 9
```

```
a() {  
    int x;  
  
    ...  
    b();  
  
}
```

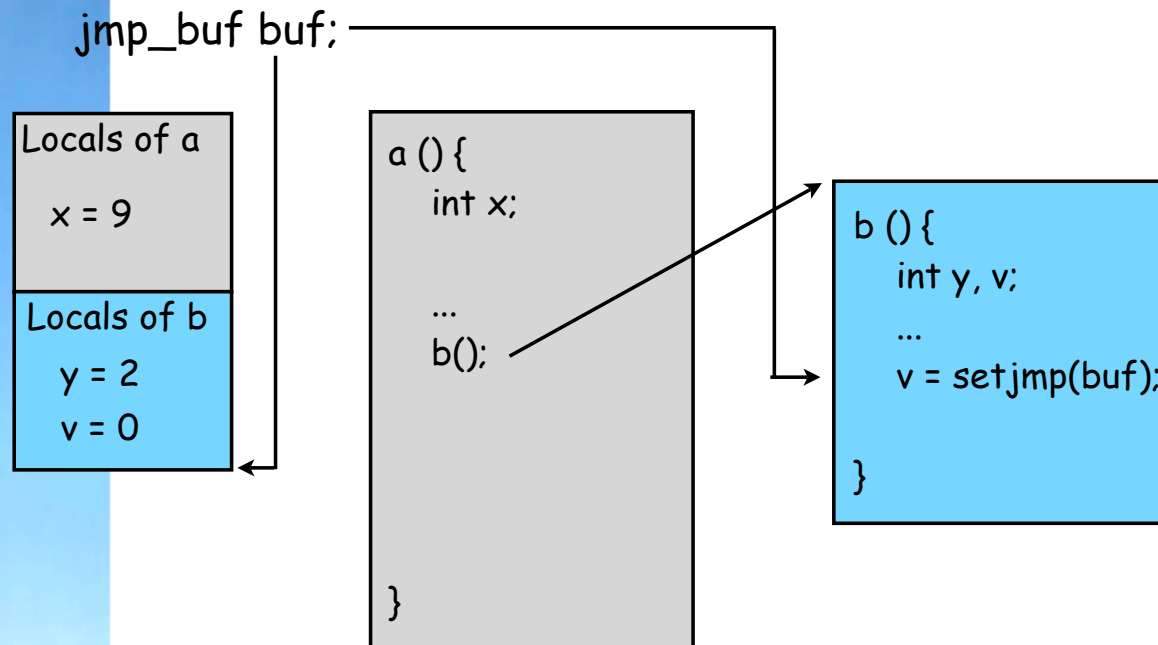


Misusing setjmp/longjmp

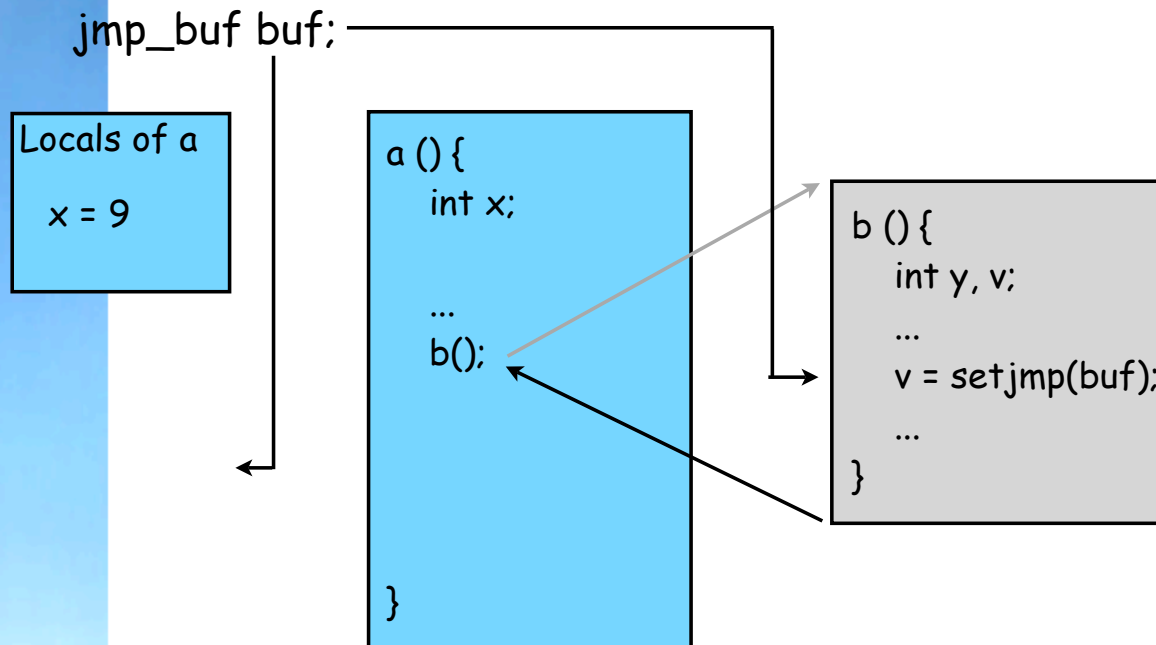
jmp_buf buf;



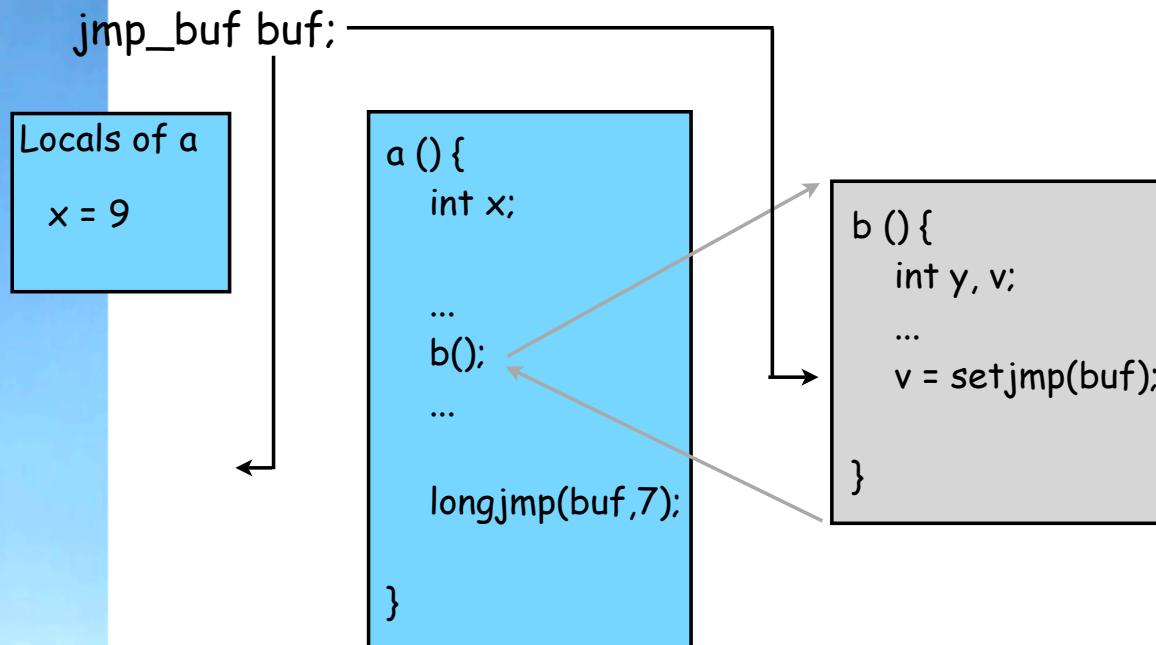
Misusing setjmp/longjmp



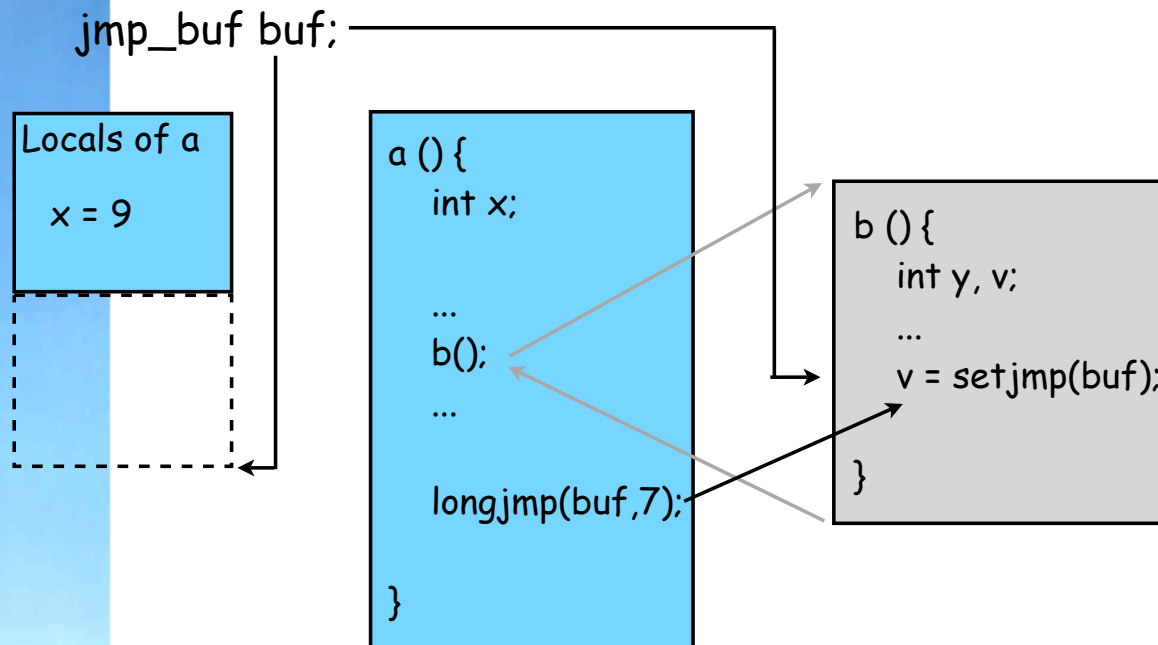
Misusing setjmp/longjmp



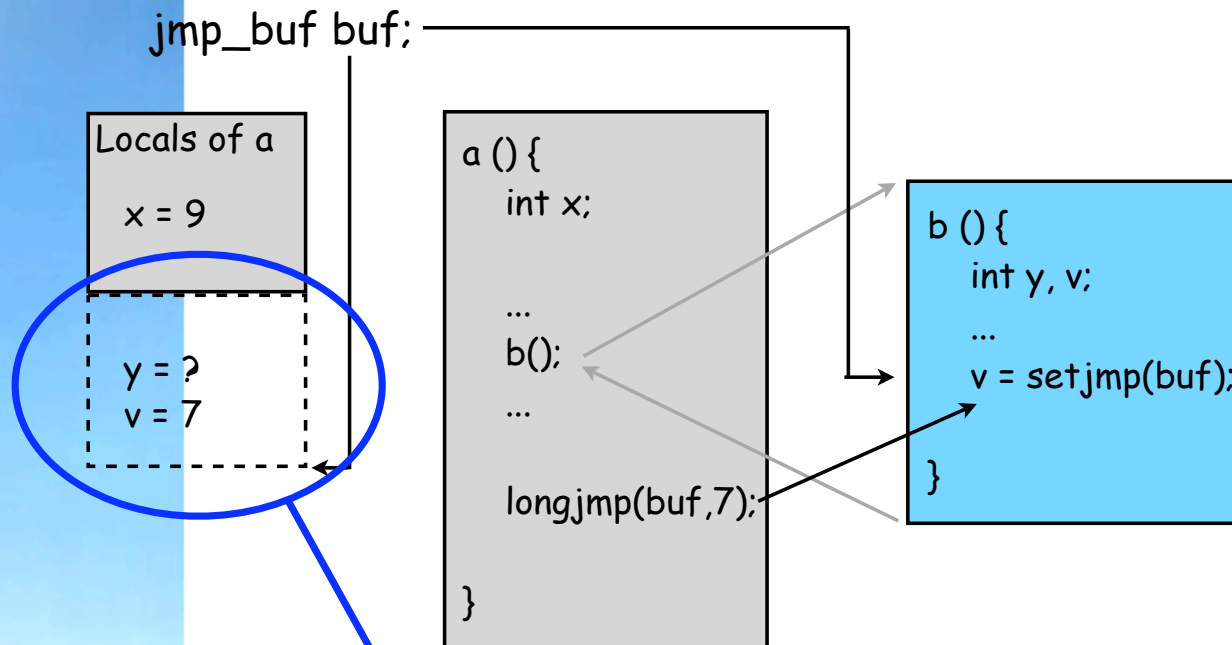
Misusing setjmp/longjmp



Misusing setjmp/longjmp



Misusing setjmp/longjmp



Non-existing stack-frame!

Hacking setjmp/longjmp!

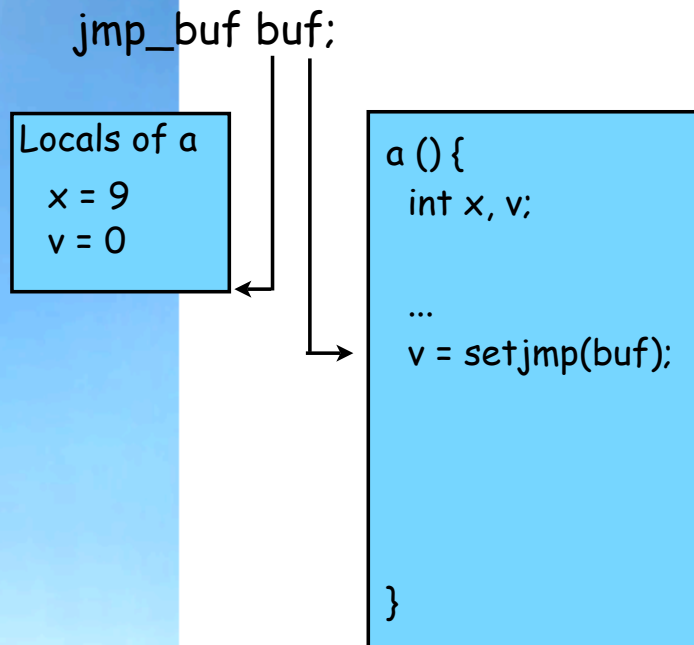
```
jmp_buf buf;
```

Locals of a

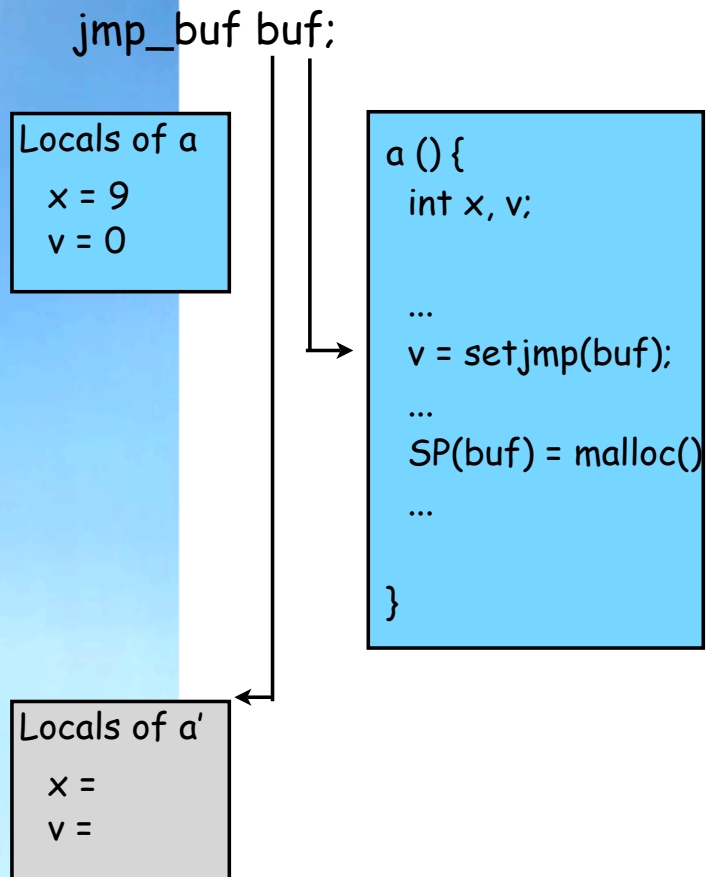
```
x = 9  
v =
```

```
a(){  
  int x, v;  
  
  ...  
  
}
```

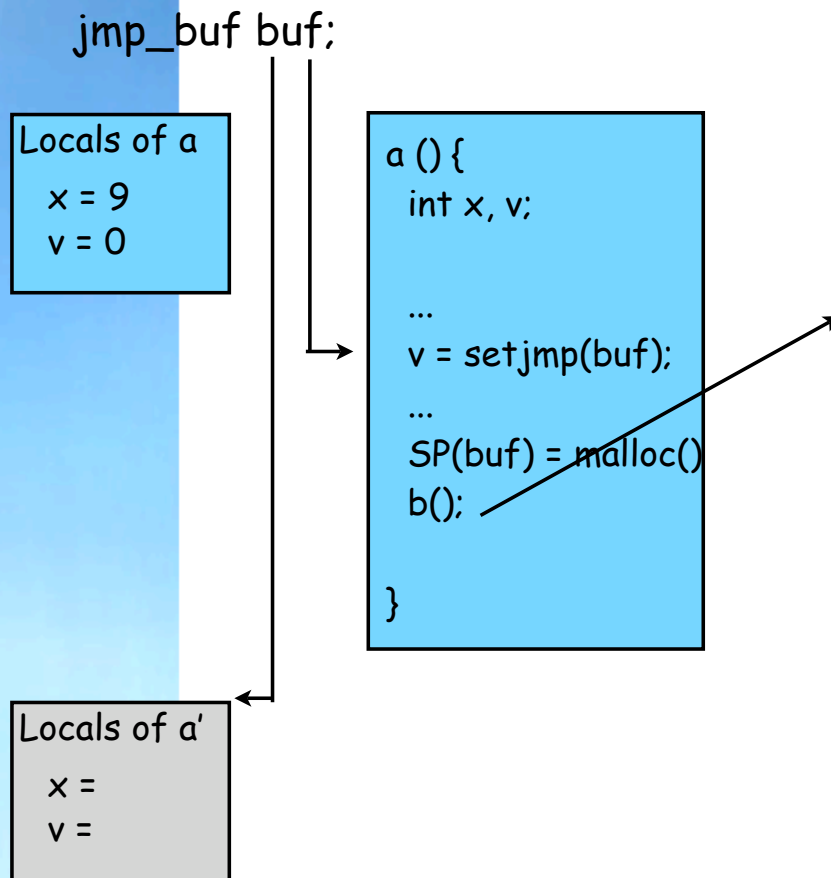
Hacking setjmp/longjmp!



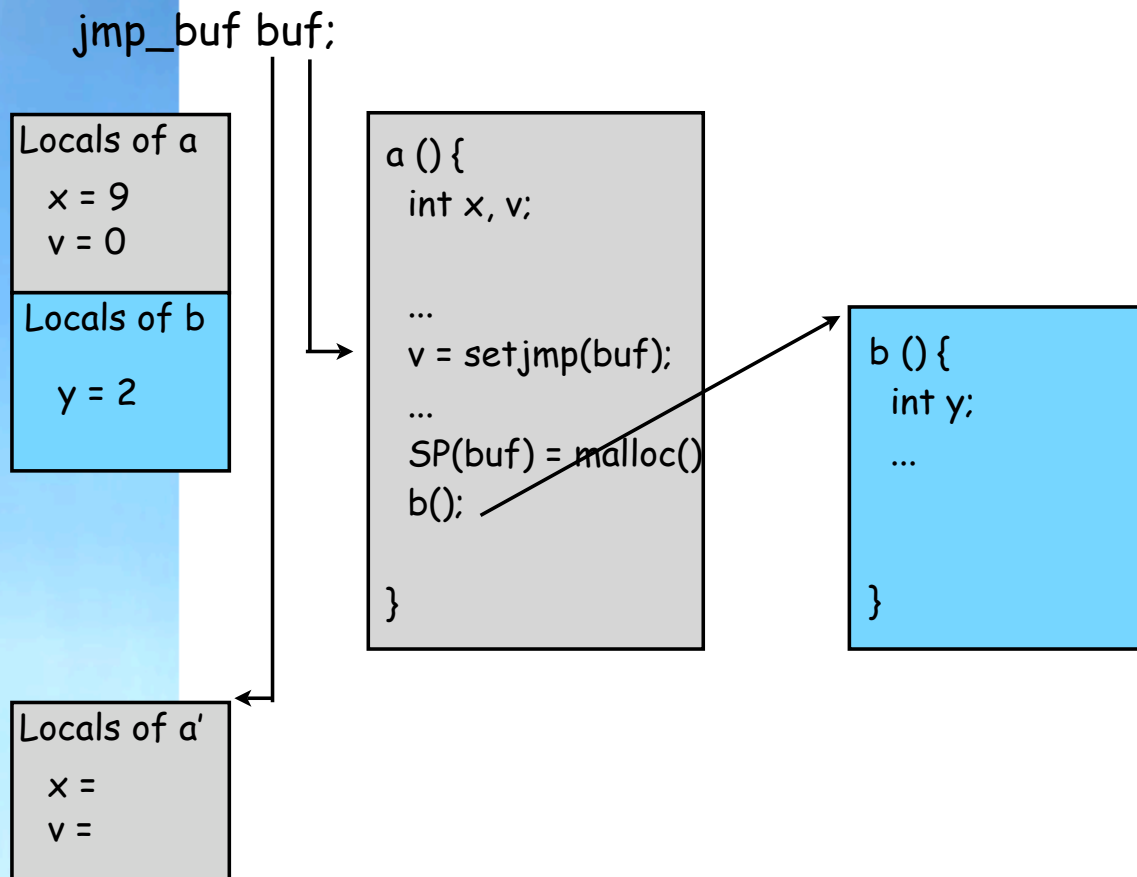
Hacking setjmp/longjmp!



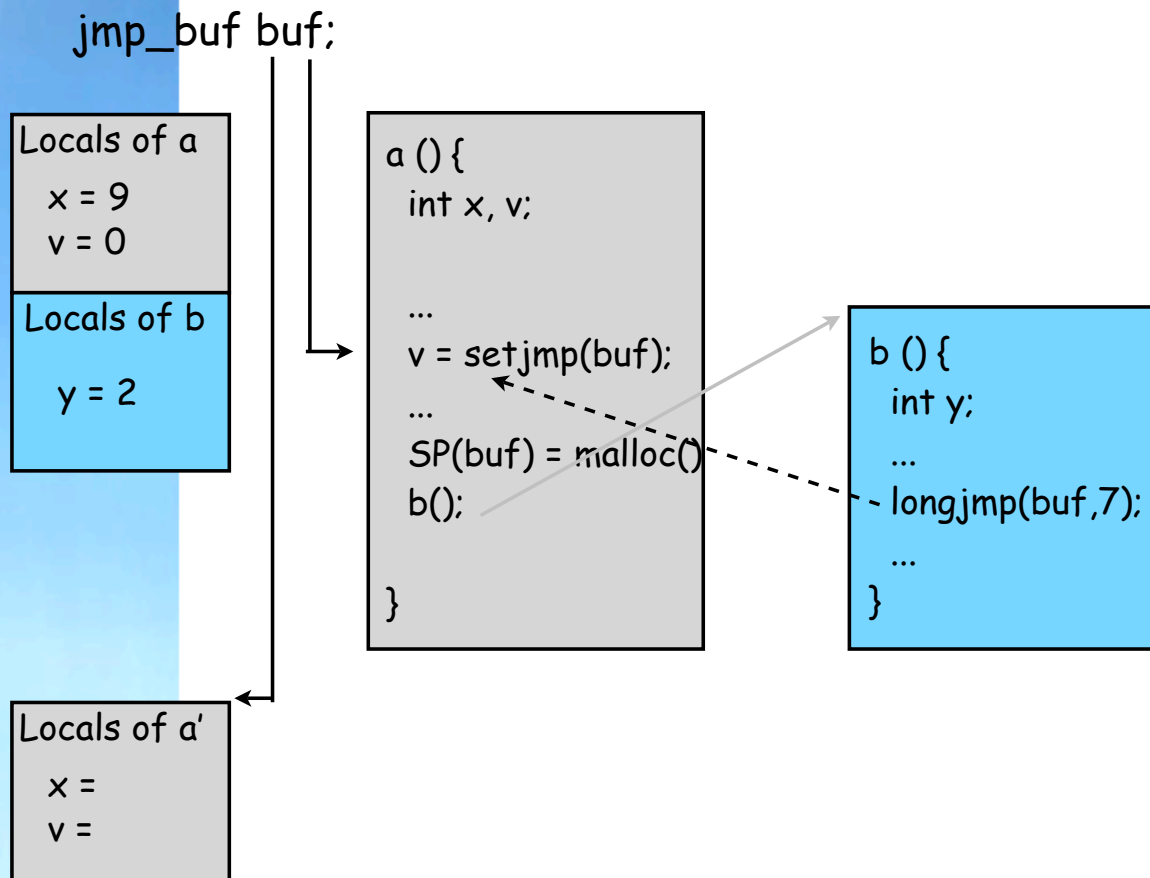
Hacking setjmp/longjmp!



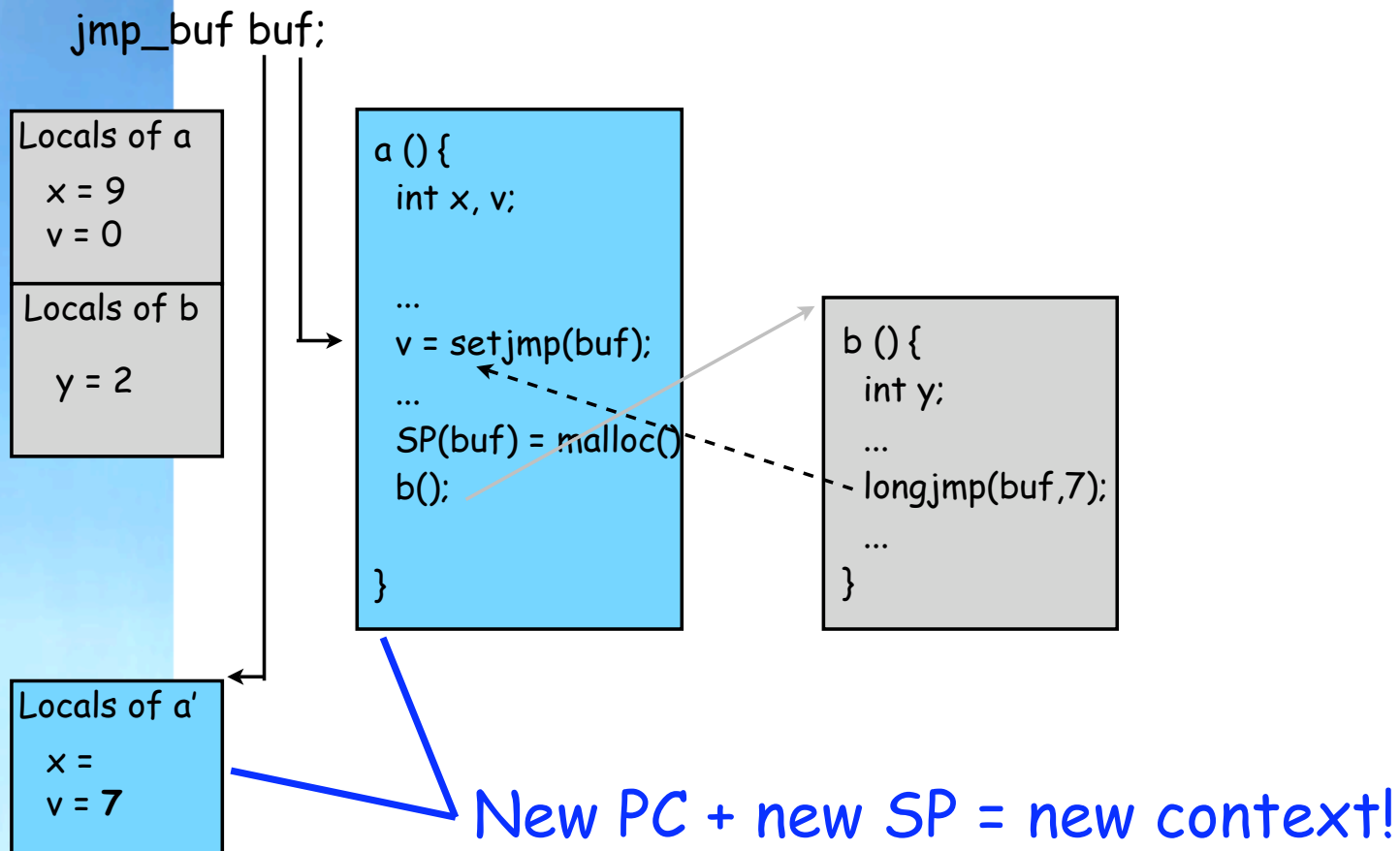
Hacking setjmp/longjmp!



Hacking setjmp/longjmp!



Hacking setjmp/longjmp!



Basic threading idea

- If one has (say) three `jmp_bufs` A, B, and C, cyclic context-switching can be implemented by running `setjmp(A); longjmp(B);` followed by `setjmp(B); longjmp(C);` followed by `setjmp(C); longjmp(A);`
- However, we can't just let each `longjmp` chop off a bit of the stack...
- Solution: modify each `jmp_buf` to point to a unique stack area!
- Warning: the hack required is **platform dependent** (but it's the only hack we'll need!)

Creating new contexts

Creating a thread context thus means creating a `jmp_buf` pointing to a fresh stack. Schematically:

```
spawn(void (*fun)(void)) {  
    jmp_buf *t = malloc(...);  
    if (setjmp(*t) != 0)  
        fun();  
    STACKPTR(t) = malloc(...);  
    // remember t ...  
}
```

Caveat: can't really write `fun()`,
because original stack is gone!

Note: macro `STACKPTR` is highly platform dependent!

The thread block

Associate with each thread a thread block, instead of just a jmp_buf:

```
typedef struct Thread {  
    void (*fun)(int);    // code to run  
    int arg;            // argument to the above  
    jmp_buf context;    // machine state  
    ...                // more...  
} Thread;
```

Let the global variable `current` point to the running thread block

A second attempt

Now we can write a better `spawn`:

```
spawn(void (*fun)(int), int arg) {  
    Thread *t = malloc(...);  
    t->fun = fun;  
    t->arg = arg;  
    if (setjmp(t->context) != 0)  
        current->fun(current->arg);  
    STACKPTR(t) = malloc(...);  
    // remember t ...  
}
```

Q 1: who sets `current`?

Q 2: what happens when `fun` returns?

The ready queue

- If all runnable threads are kept in a queue, we can implement a function `yield()` that switches execution to another thread
- `yield()` must
 - enqueue the current thread at the ready queue
 - pick a new thread from the ready queue
 - update `current`
 - perform the `setjmp(A); longjmp(B)` trick to switch contexts - a dispatch
- Later on we can try to make `yield()` happen automatically; i.e., by means of an `interrupt!`

The dispatch function

Assuming that `current` points to the running thread block, a `dispatch` can be implemented as follows:

```
void dispatch(Thread *next) {  
    if (setjmp(current->context) == 0) {  
        current = next;  
        longjmp(next->context,1);  
    }  
}
```

Note that the function returns directly when control later enters via the "fake return" from `setjmp`

Putting it all together

- We will run through a complete stepwise execution of a program that spawns a thread that computes prime numbers, and then goes on to compute prime numbers itself too
- The current code and stack pointers of the cpu are denoted by thick arrows
- Other pointers, as well as the ready queue, will be represented informally
- The code will be schematic, but full details can be found in lab assignment 2
- Global variables are in the middle, stacks to the right

```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



A

```

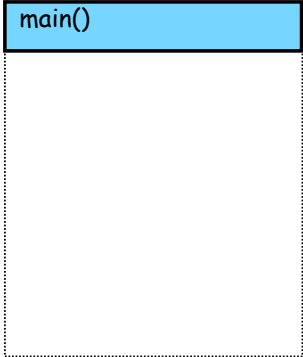
fun = main
arg = 0
context =

```

```

current = A
readyQ = []

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...);
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...);
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

A
fun = main
arg = 0
context =

```

```

current = A
readyQ = []

```

```

main()
spawn()
fun = primes
arg = 101
t =

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



A

```

fun = main
arg = 0
context =

```

```

current = A
readyQ = []

```

B

```

fun =
arg =
context =

```

```

main()
spawn()
fun = primes
arg = 101
t = B

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



A

```

fun = main
arg = 0
context =

```

```

current = A
readyQ = []

```

B

```

fun = primes
arg =
context =

```

```

main()
spawn()
fun = primes
arg = 101
t = B

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



A

```

fun = main
arg = 0
context =

```

```

current = A
readyQ = []

```

B

```

fun = primes
arg = 101
context =

```

```

main()
spawn()
fun = primes
arg = 101
t = B

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

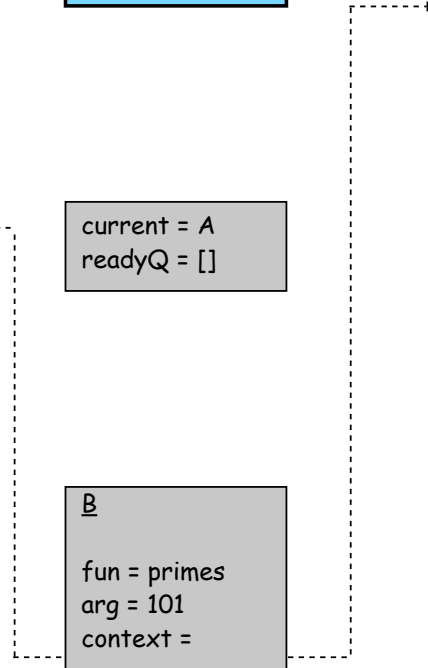
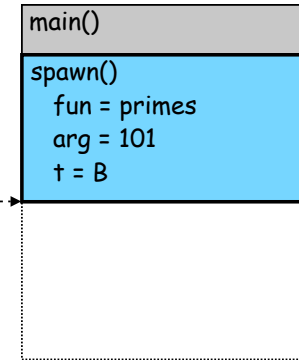
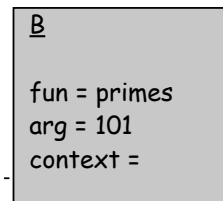
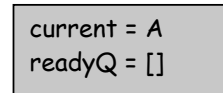
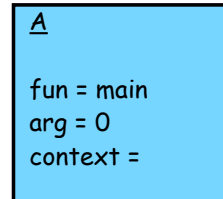
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

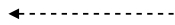
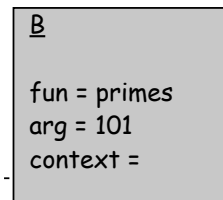
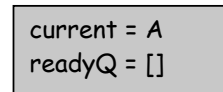
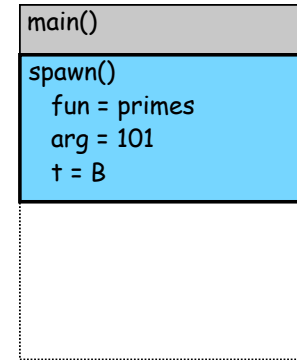
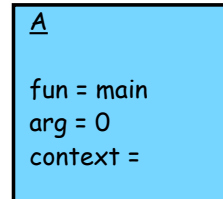
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

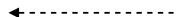
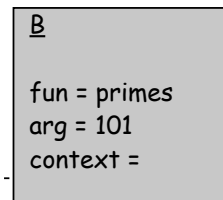
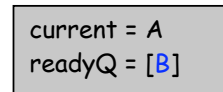
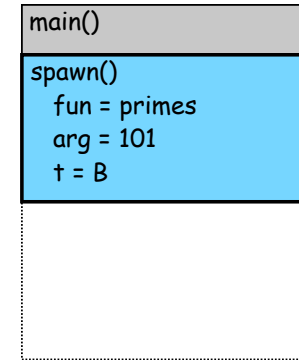
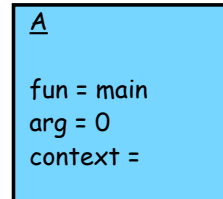
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



A

```

fun = main
arg = 0
context =

```

```

current = A
readyQ = [B]

```

B

```

fun = primes
arg = 101
context =

```

main()



main()

```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

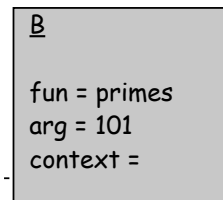
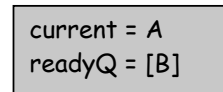
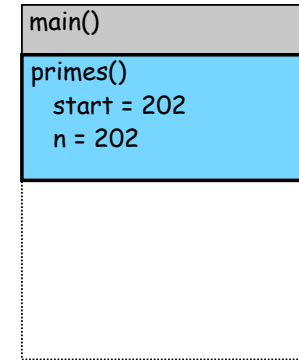
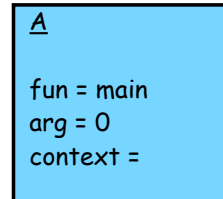
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

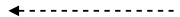
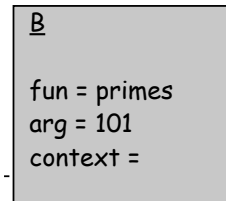
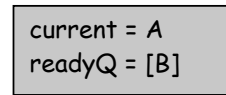
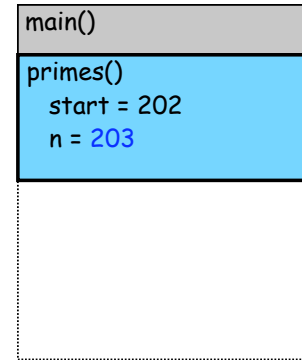
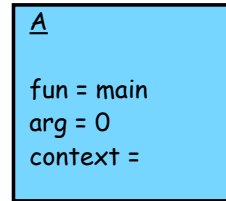
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

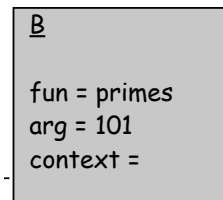
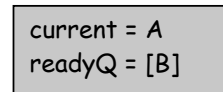
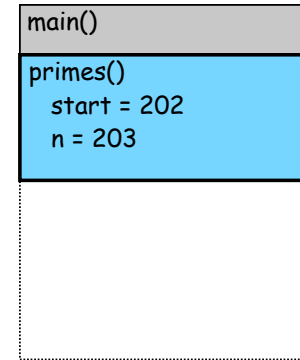
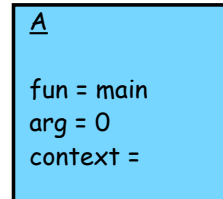
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

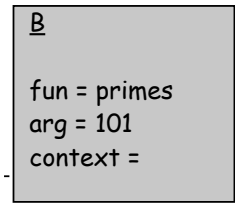
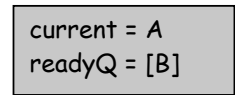
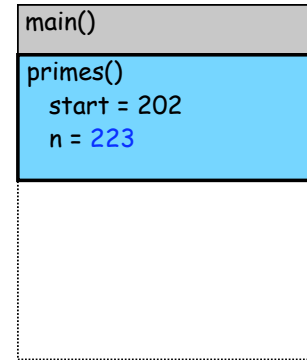
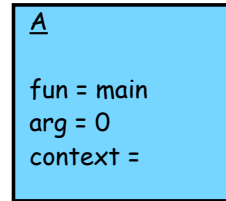
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

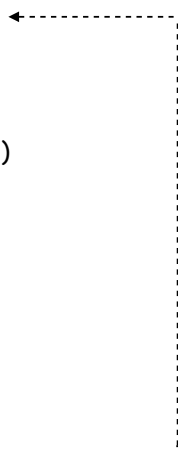
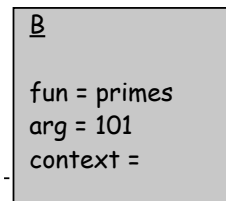
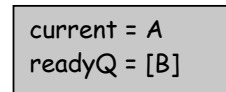
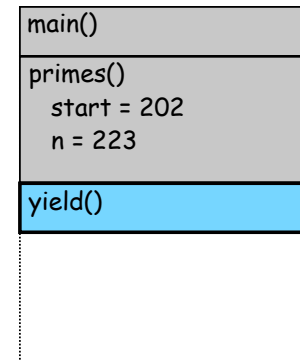
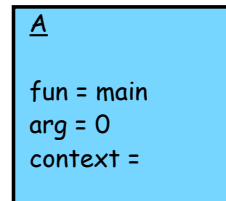
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

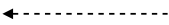
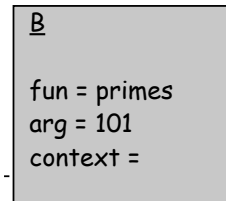
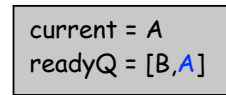
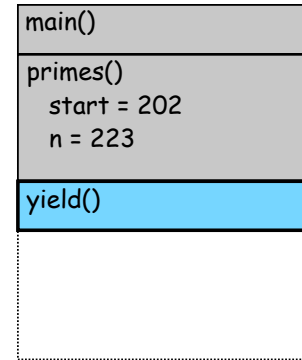
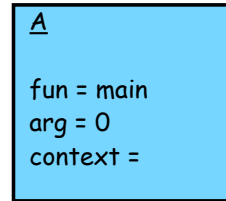
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

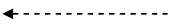
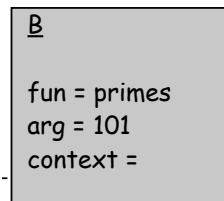
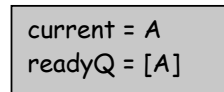
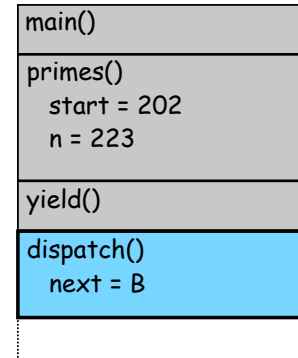
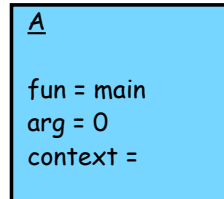
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



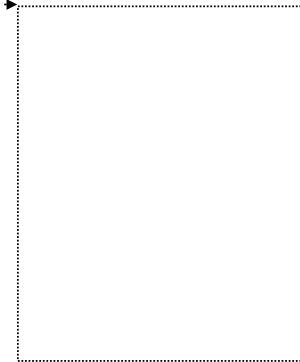
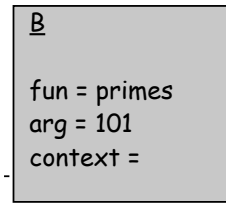
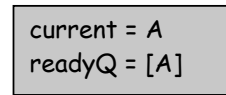
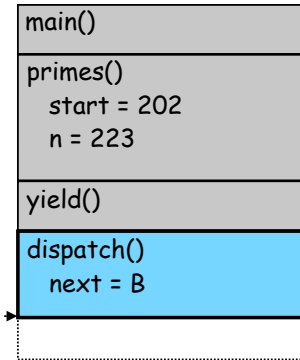
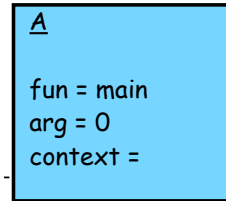
```
void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}
```

```
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}
```

```
void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}
```

```
primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}
```

```
main() {
    spawn(primes, 101);
    primes(202);
}
```



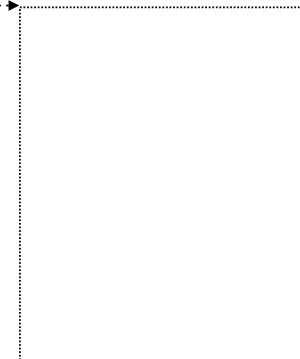
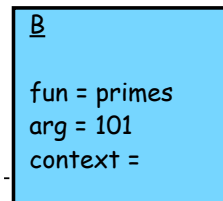
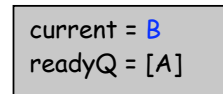
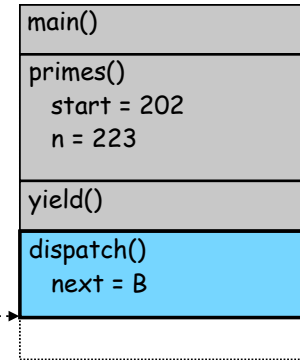
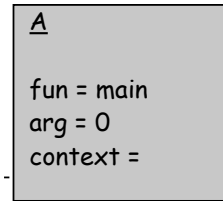
```
void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}
```

```
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}
```

```
void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}
```

```
primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}
```

```
main() {
    spawn(primes, 101);
    primes(202);
}
```



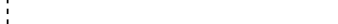
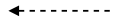
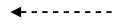
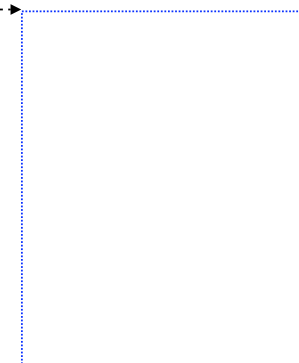
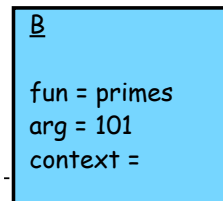
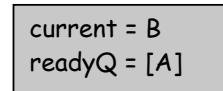
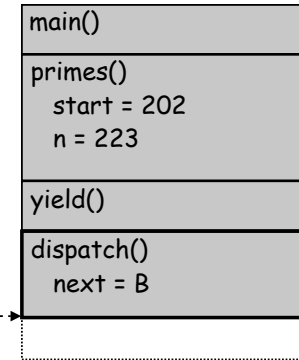
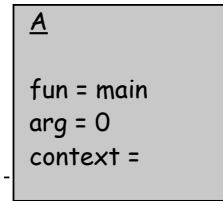
```
void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}
```

```
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}
```

```
void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}
```

```
primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}
```

```
main() {
    spawn(primes, 101);
    primes(202);
}
```



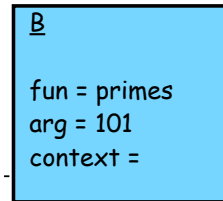
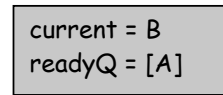
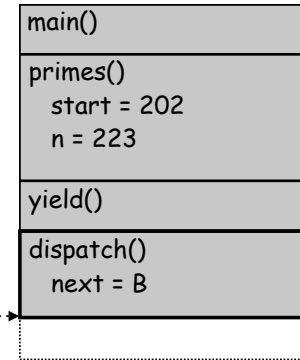
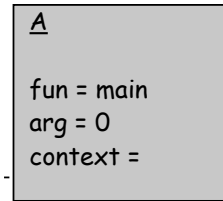
```
void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}
```

```
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}
```

```
void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}
```

```
primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}
```

```
main() {
    spawn(primes, 101);
    primes(202);
}
```



Note that fun and arg can't be found on the stack anymore!

```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

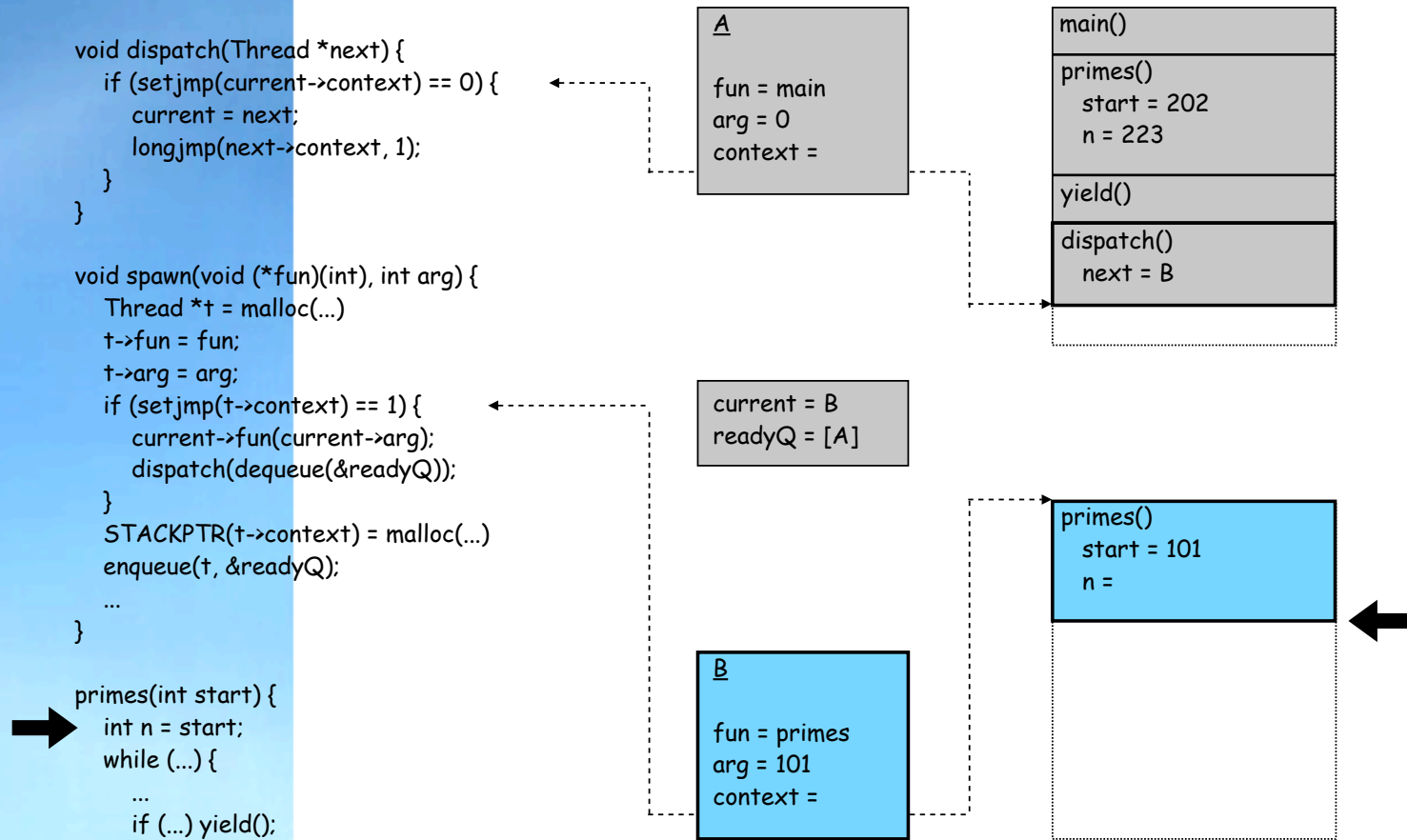
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

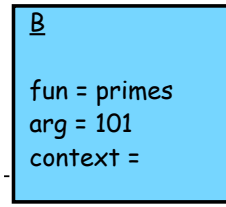
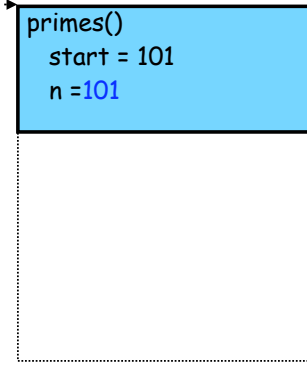
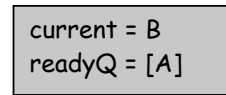
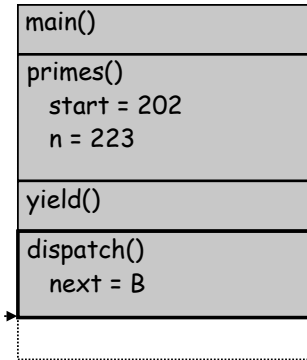
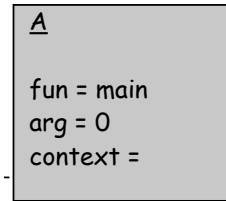
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

```

```

void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

```

```

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

```

```

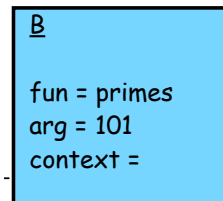
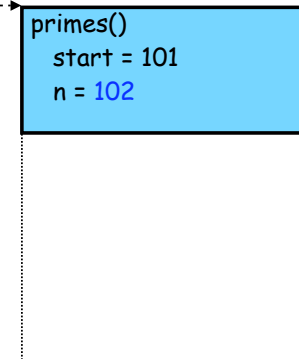
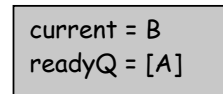
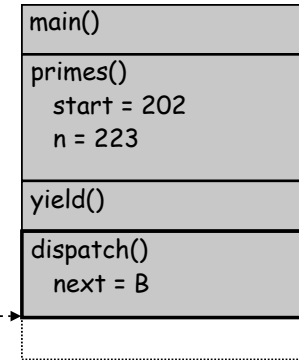
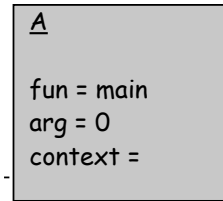
primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

```

```

main() {
    spawn(primes, 101);
    primes(202);
}

```



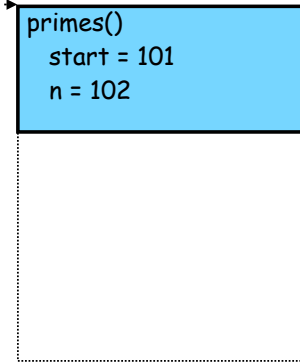
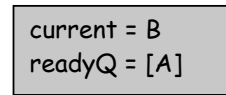
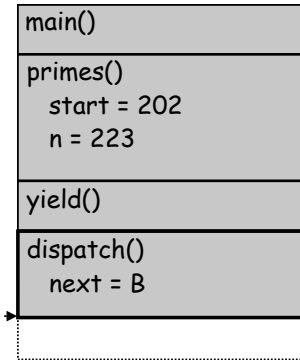
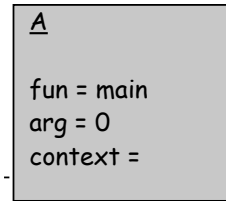
```
void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}
```

```
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}
```

```
void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}
```

```
primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}
```

```
main() {
    spawn(primes, 101);
    primes(202);
}
```



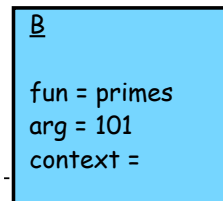
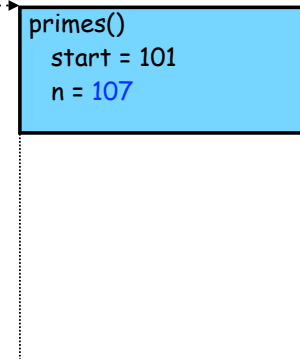
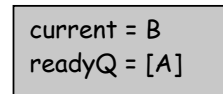
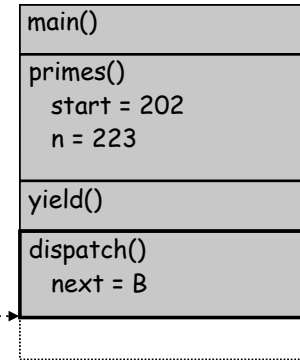
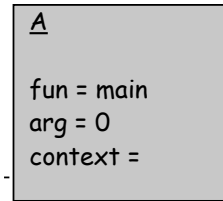
```
void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}
```

```
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}
```

```
void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}
```

```
primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}
```

```
main() {
    spawn(primes, 101);
    primes(202);
}
```



```

void yield() {
  → enqueue(current, &readyQ);
  dispatch(dequeue(&readyQ));
}

```

```

void dispatch(Thread *next) {
  if (setjmp(current->context) == 0) {
    current = next;
    longjmp(next->context, 1);
  }
}

```

```

void spawn(void (*fun)(int), int arg) {
  Thread *t = malloc(...)
  t->fun = fun;
  t->arg = arg;
  if (setjmp(t->context) == 1) {
    current->fun(current->arg);
    dispatch(dequeue(&readyQ));
  }
  STACKPTR(t->context) = malloc(...)
  enqueue(t, &readyQ);
  ...
}

```

```

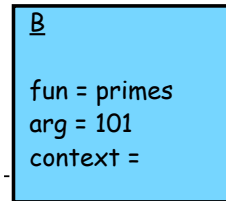
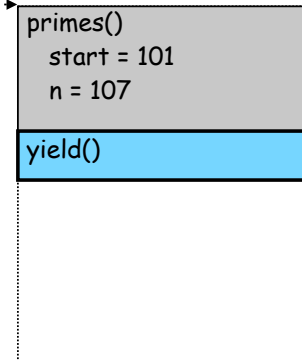
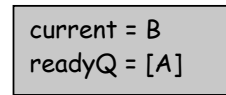
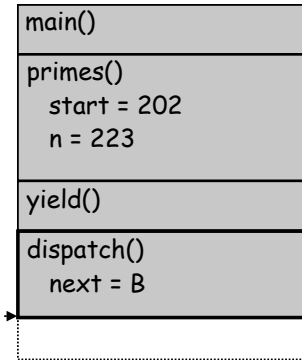
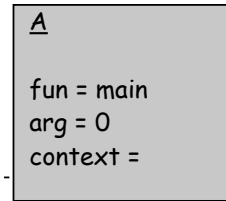
primes(int start) {
  int n = start;
  while (...) {
    ...
    if (...) yield();
  }
}

```

```

main() {
  spawn(primes, 101);
  primes(202);
}

```



```

void yield() {
  enqueue(current, &readyQ);
  dispatch(dequeue(&readyQ));
}

```

```

void dispatch(Thread *next) {
  if (setjmp(current->context) == 0) {
    current = next;
    longjmp(next->context, 1);
  }
}

```

```

void spawn(void (*fun)(int), int arg) {
  Thread *t = malloc(...)
  t->fun = fun;
  t->arg = arg;
  if (setjmp(t->context) == 1) {
    current->fun(current->arg);
    dispatch(dequeue(&readyQ));
  }
  STACKPTR(t->context) = malloc(...)
  enqueue(t, &readyQ);
  ...
}

```

```

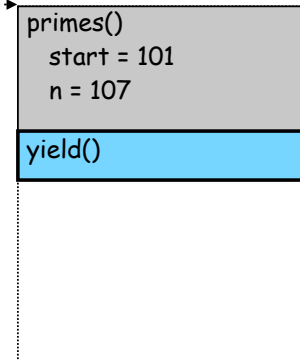
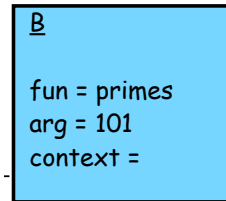
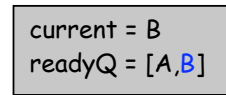
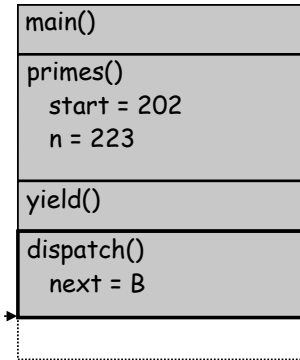
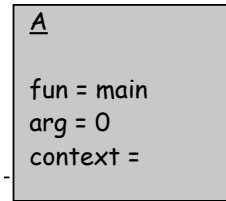
primes(int start) {
  int n = start;
  while (...) {
    ...
    if (...) yield();
  }
}

```

```

main() {
  spawn(primes, 101);
  primes(202);
}

```



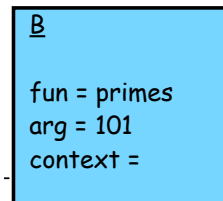
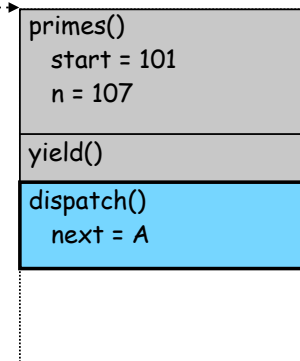
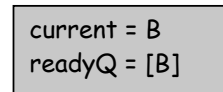
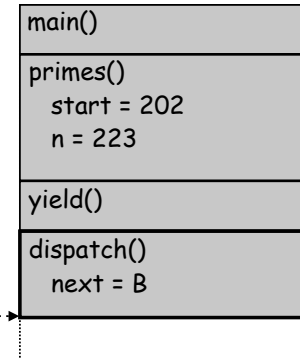
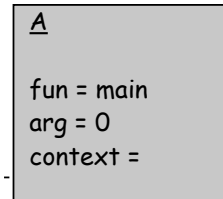
```
void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}
```

```
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}
```

```
void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}
```

```
primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}
```

```
main() {
    spawn(primes, 101);
    primes(202);
}
```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

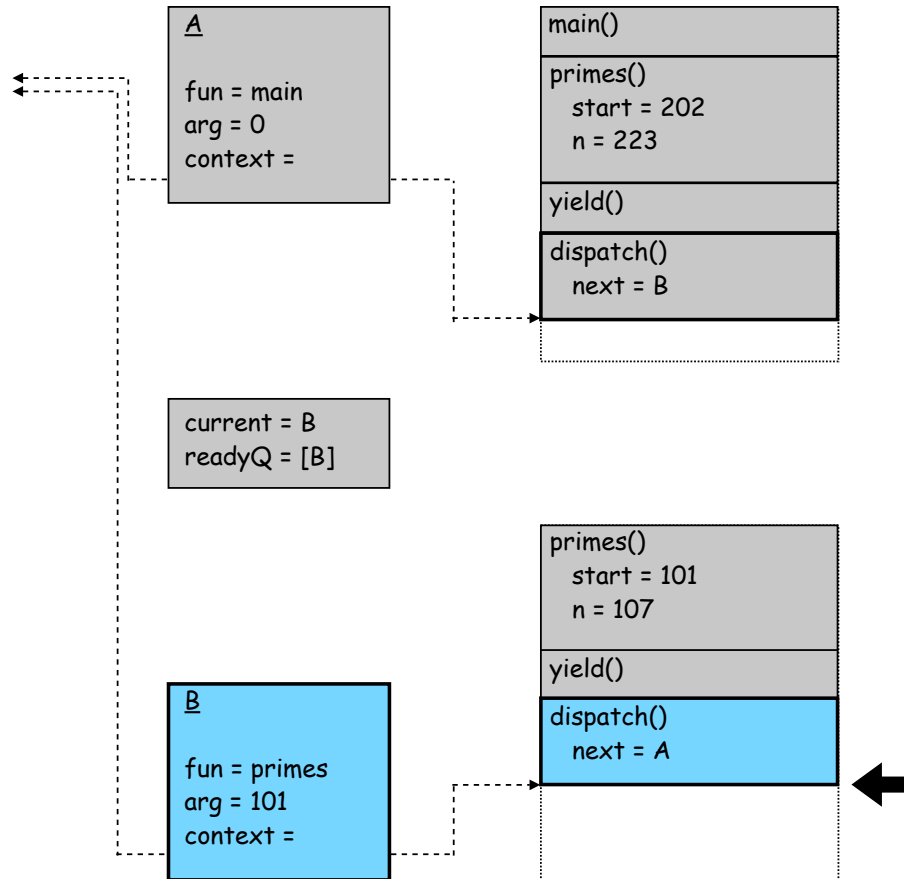
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

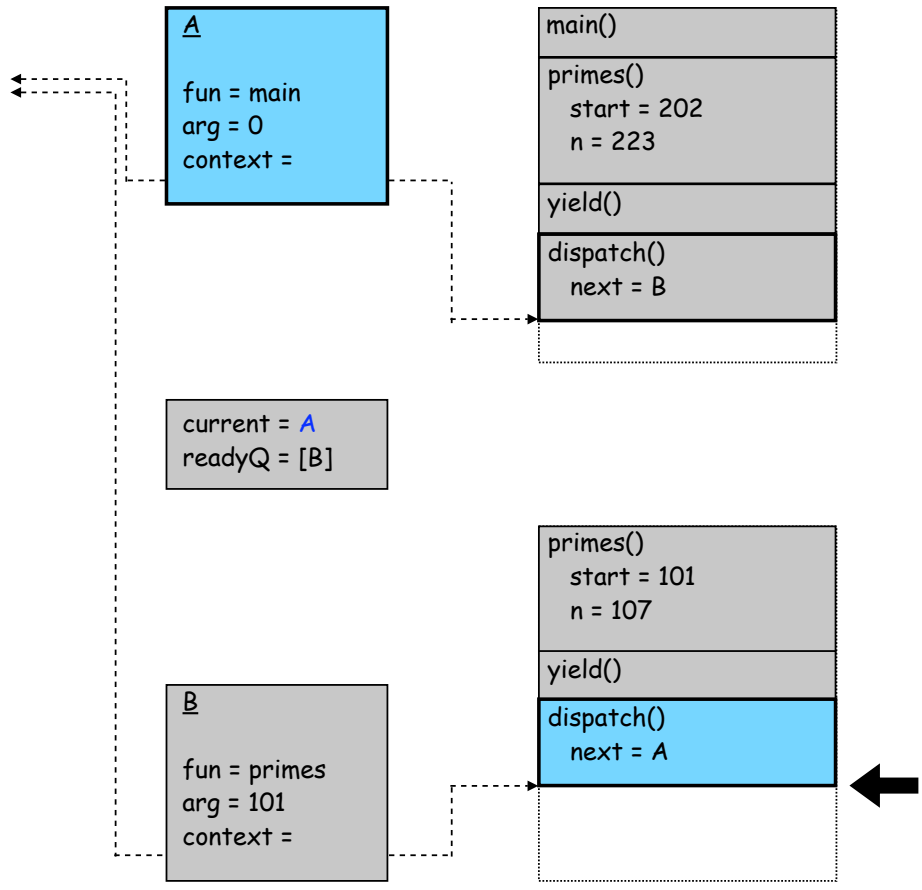
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

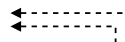
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



A

```

fun = main
arg = 0
context =

```

```

current = A
readyQ = [B]

```

B

```

fun = primes
arg = 101
context =

```

```

main()
primes()
  start = 202
  n = 223
yield()
dispatch()
  next = B

```



```

primes()
  start = 101
  n = 107
yield()
dispatch()
  next = A

```

```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

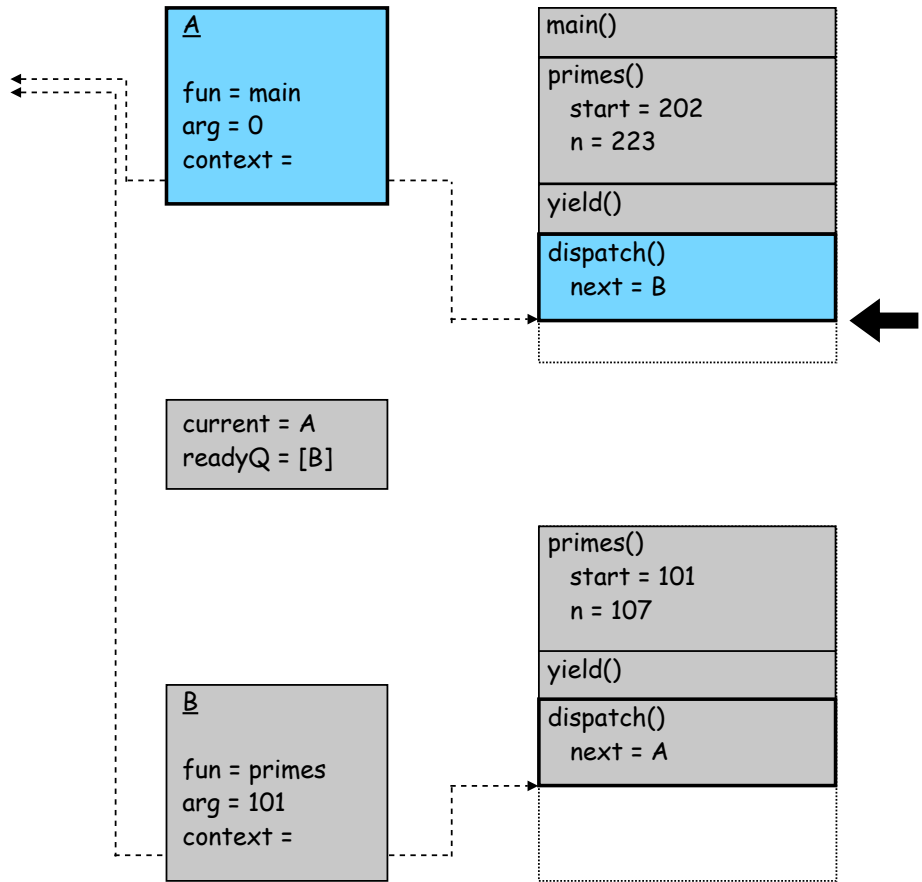
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

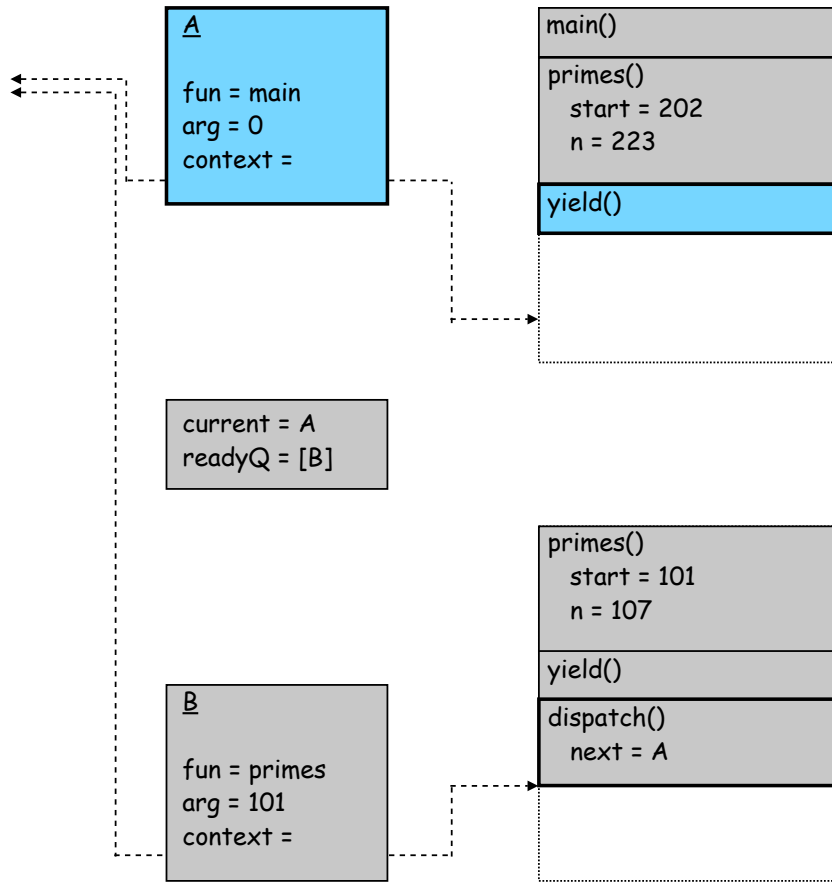
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

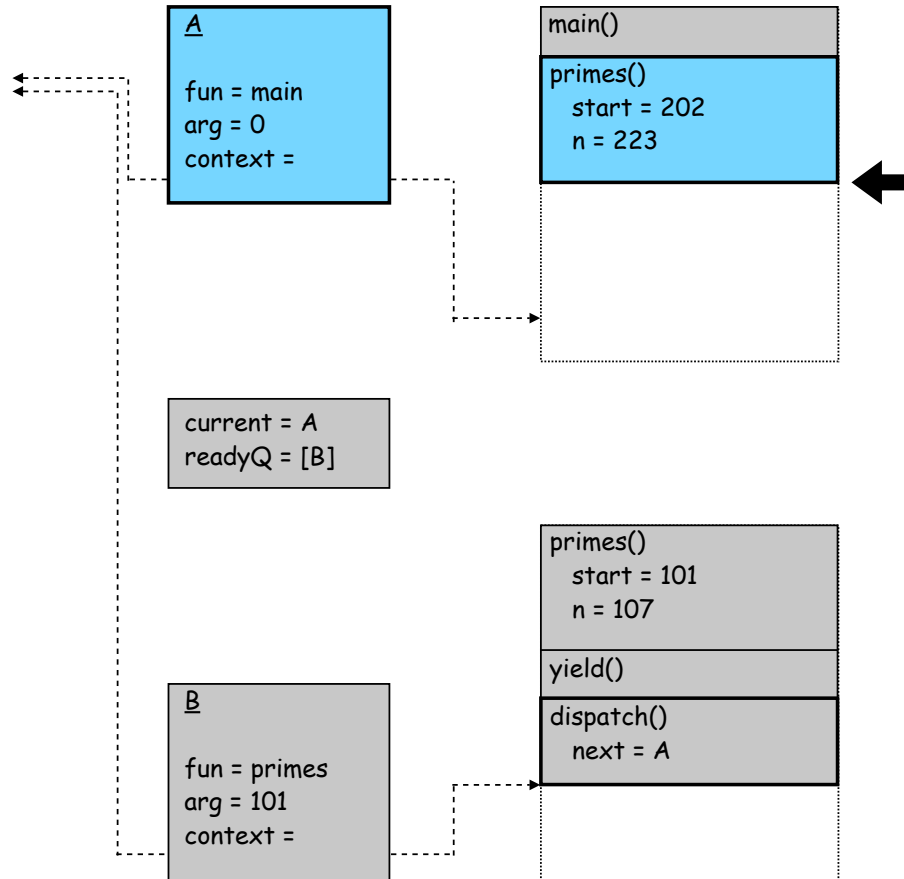
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

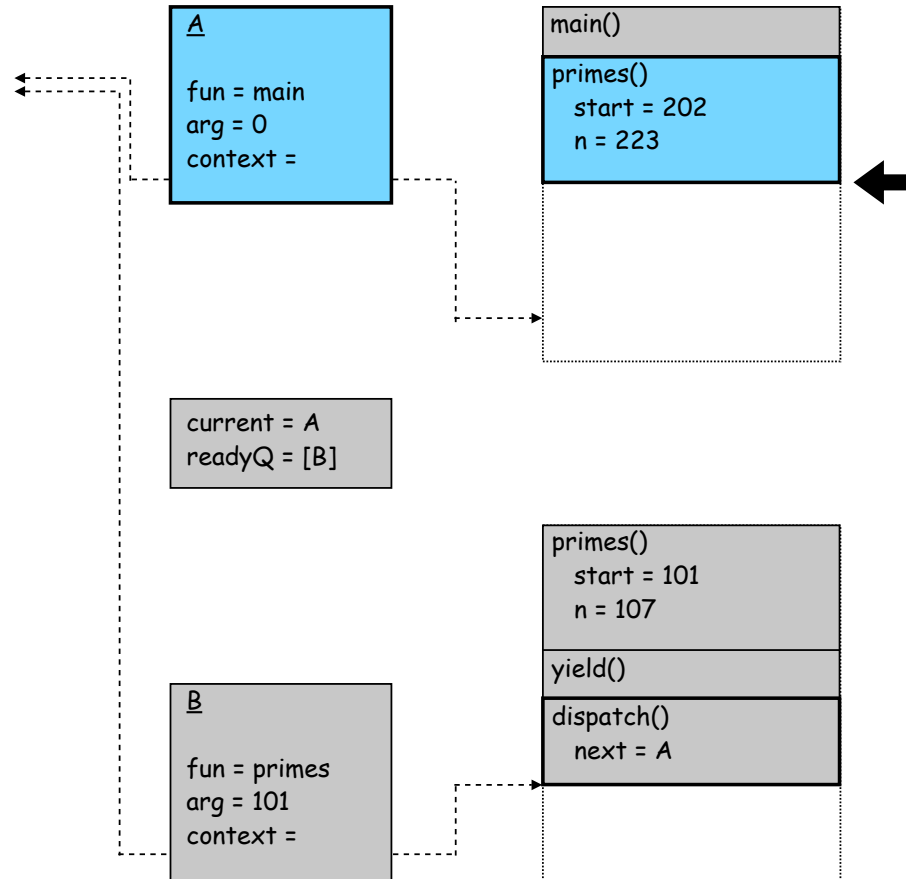
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

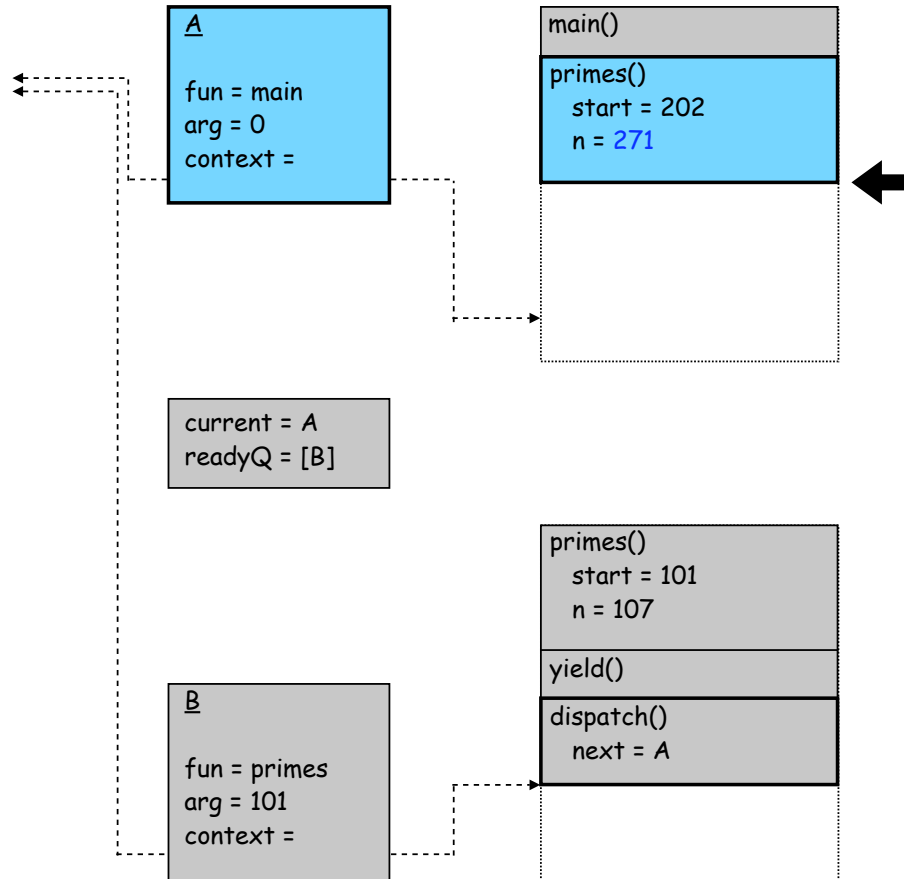
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

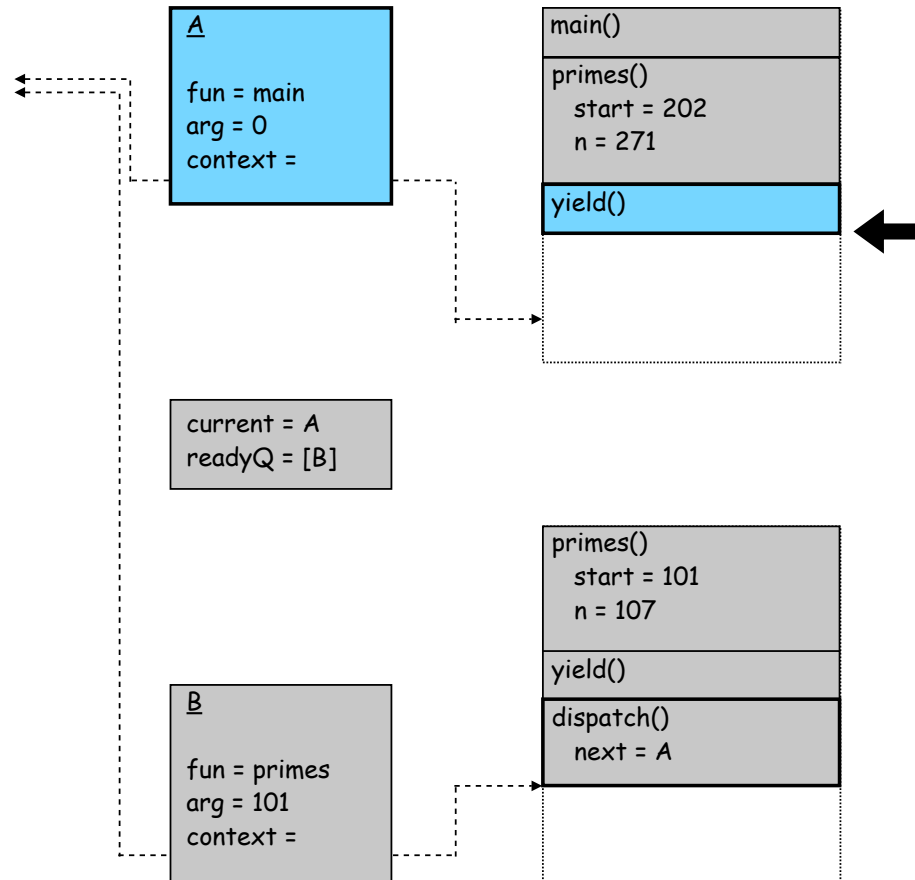
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

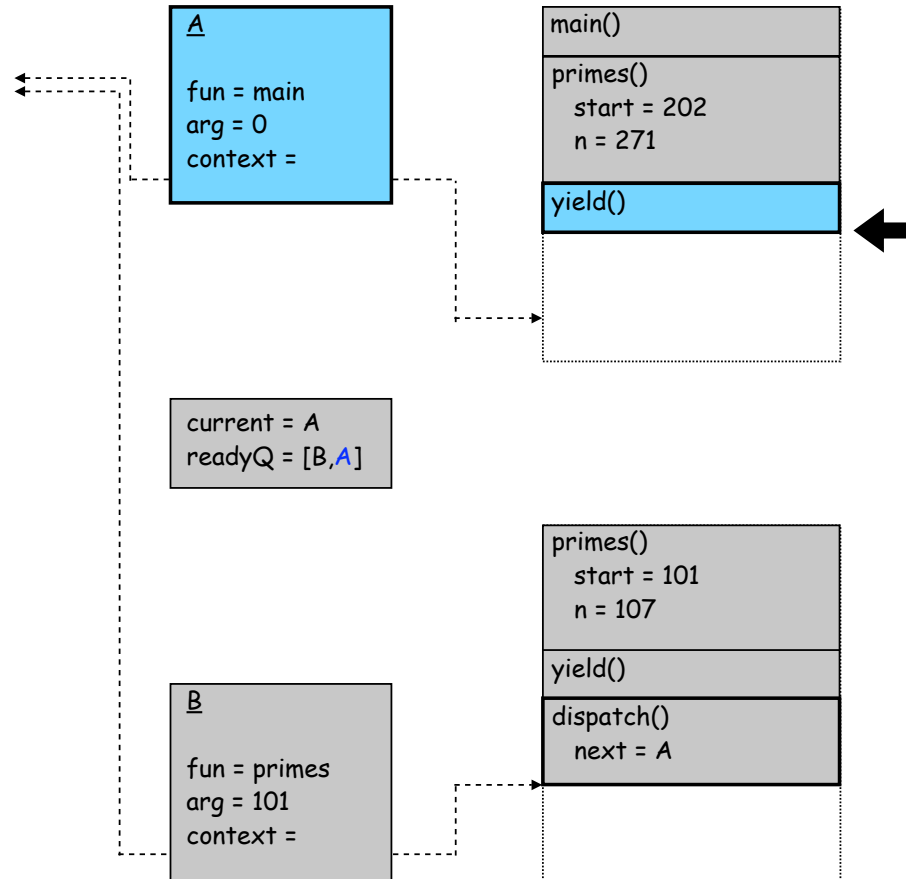
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

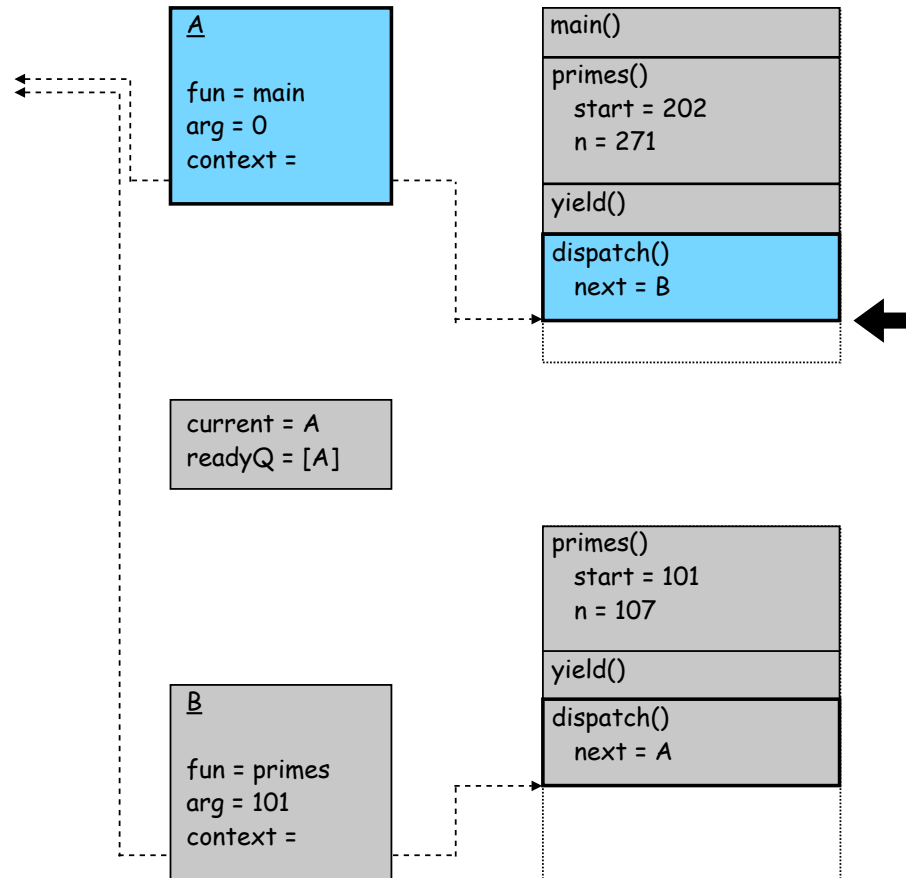
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

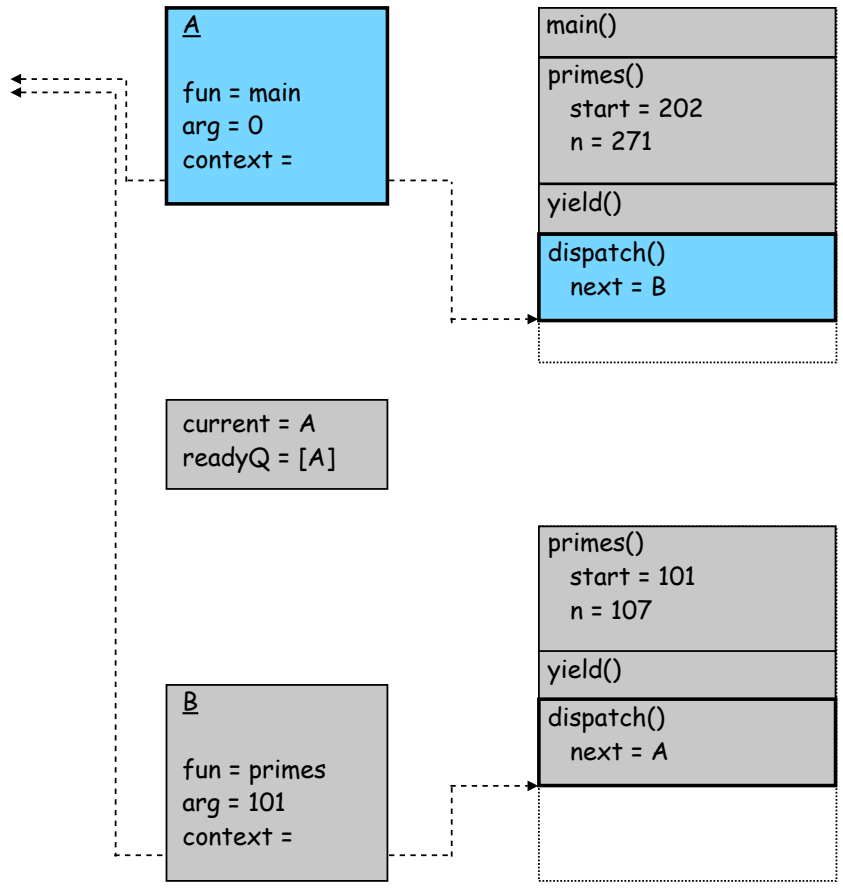
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

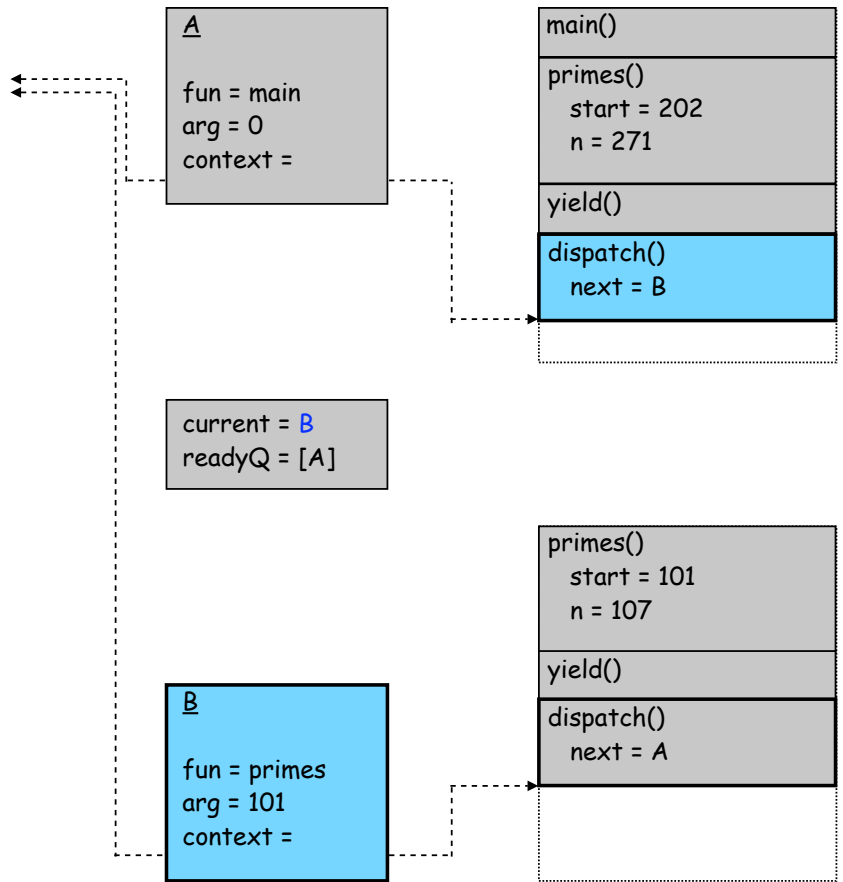
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

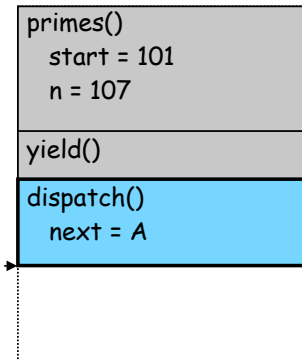
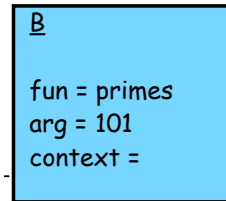
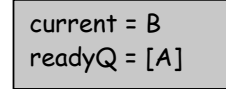
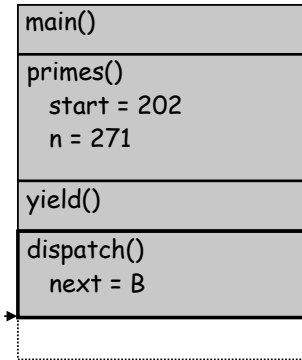
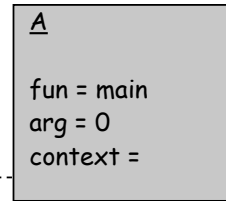
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

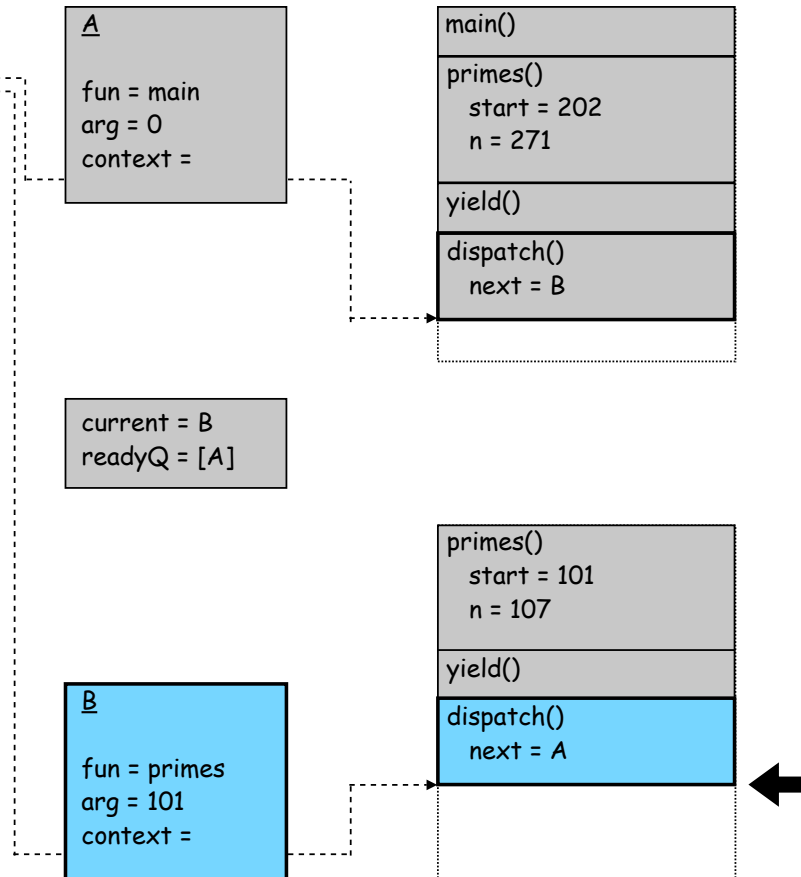
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

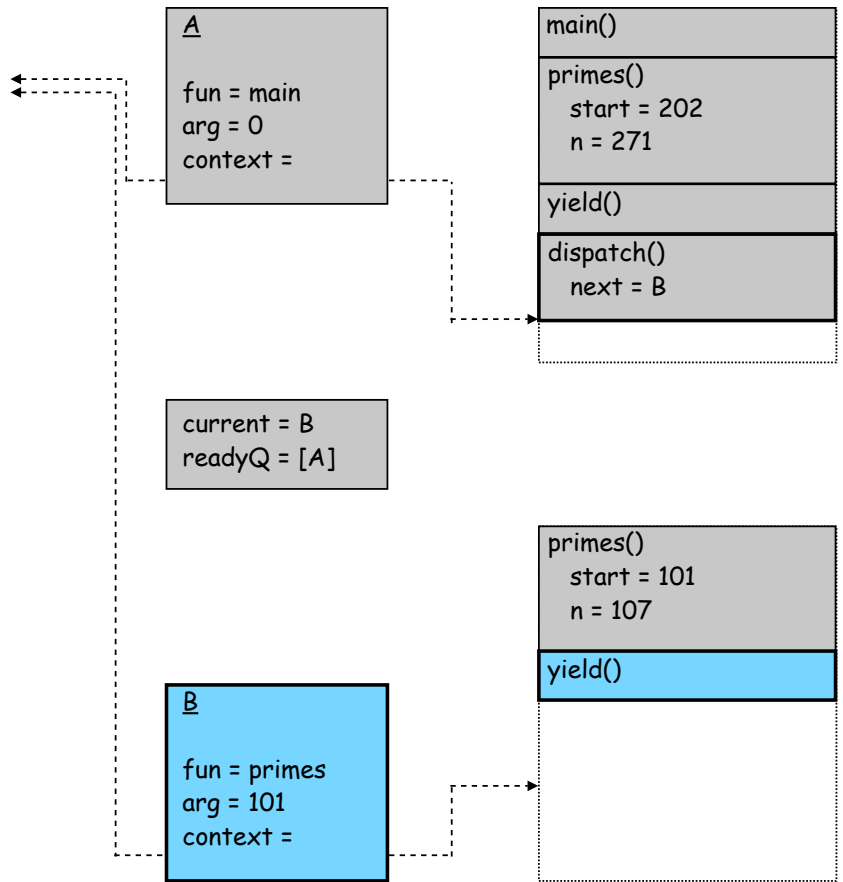
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

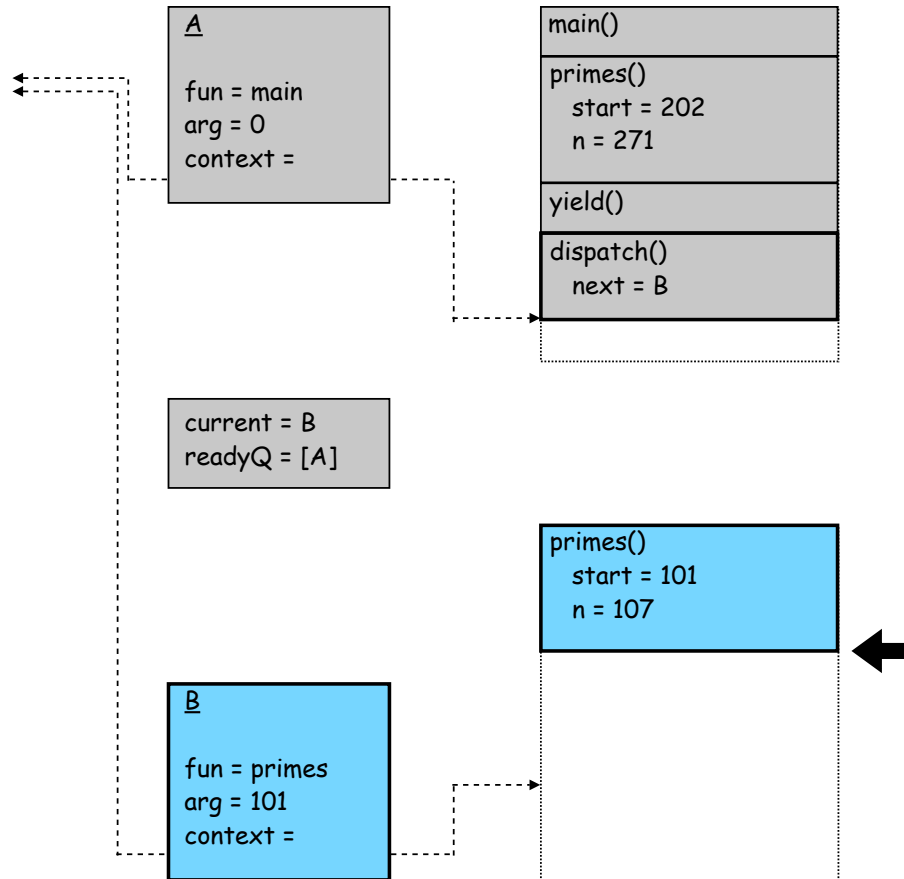
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

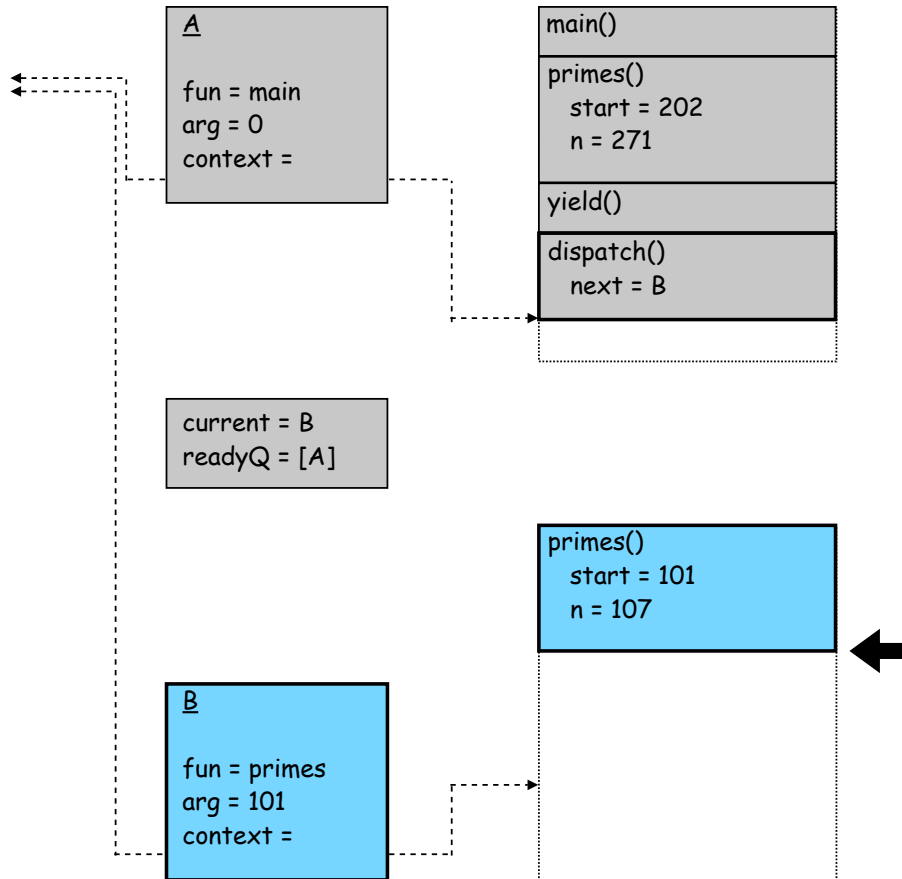
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

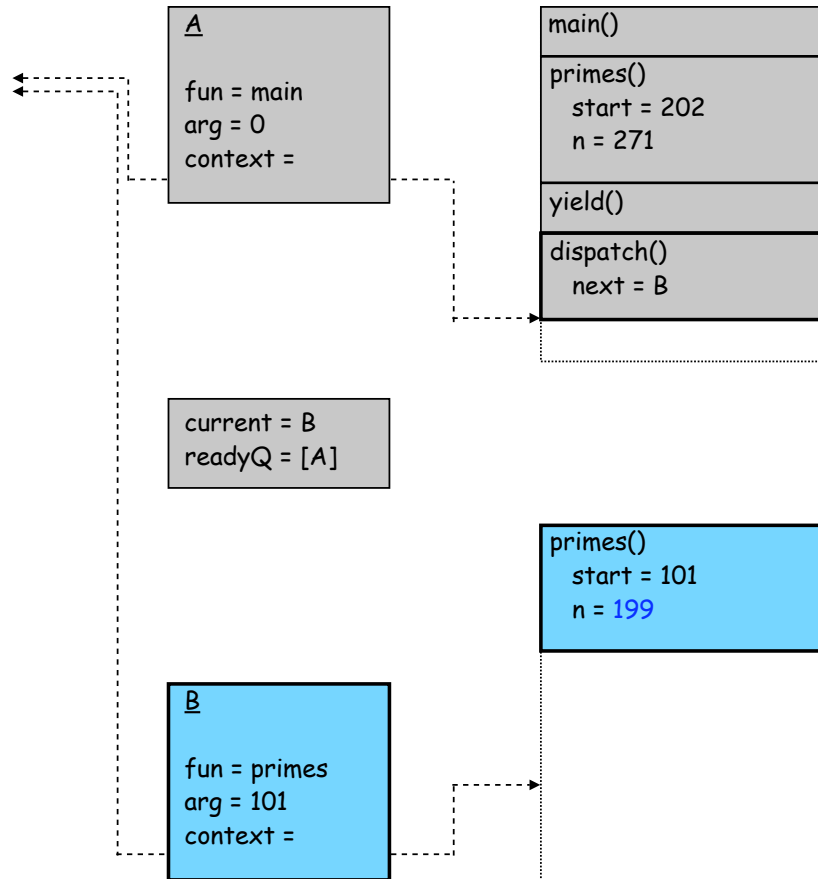
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

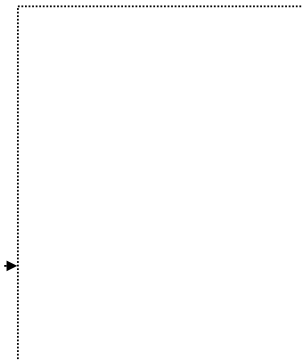
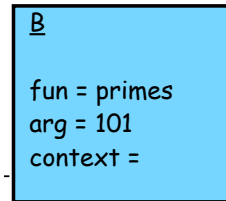
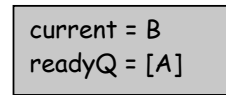
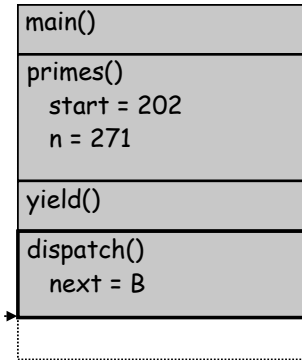
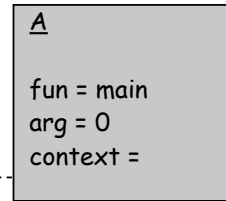
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

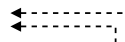
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



A

```

fun = main
arg = 0
context =

```

```

main()
primes()
  start = 202
  n = 271
yield()
dispatch()
  next = B

```

```

current = B
readyQ = []

```

```

dispatch()
  next = A

```



B

```

fun = primes
arg = 101
context =

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

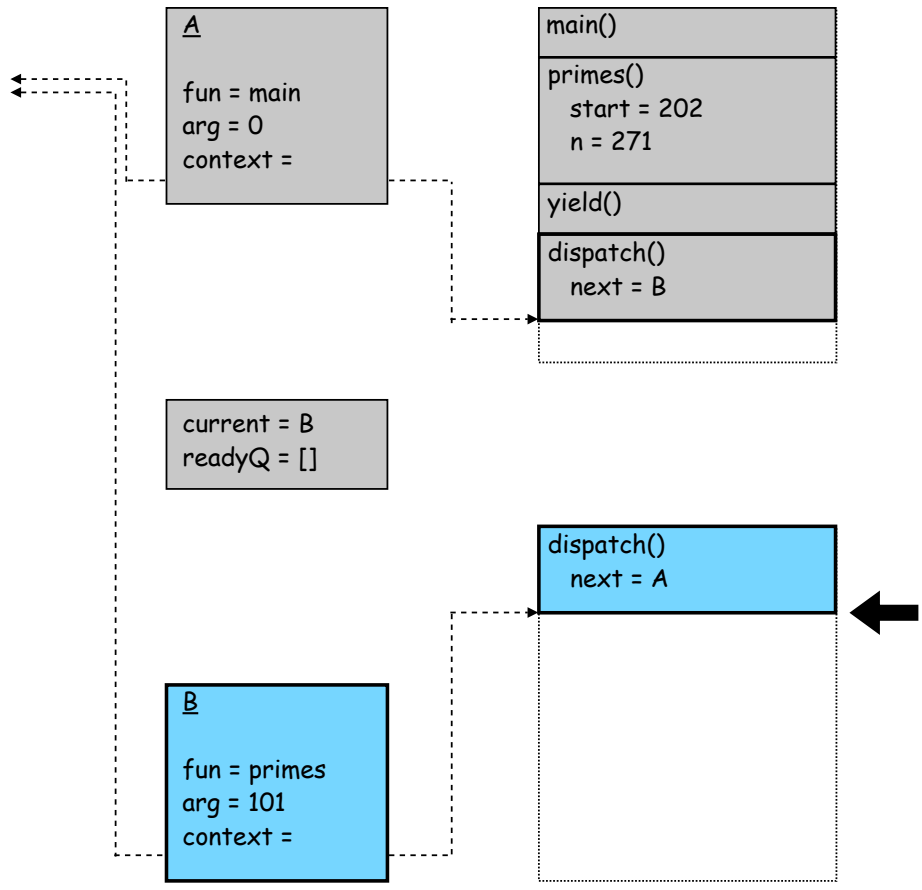
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

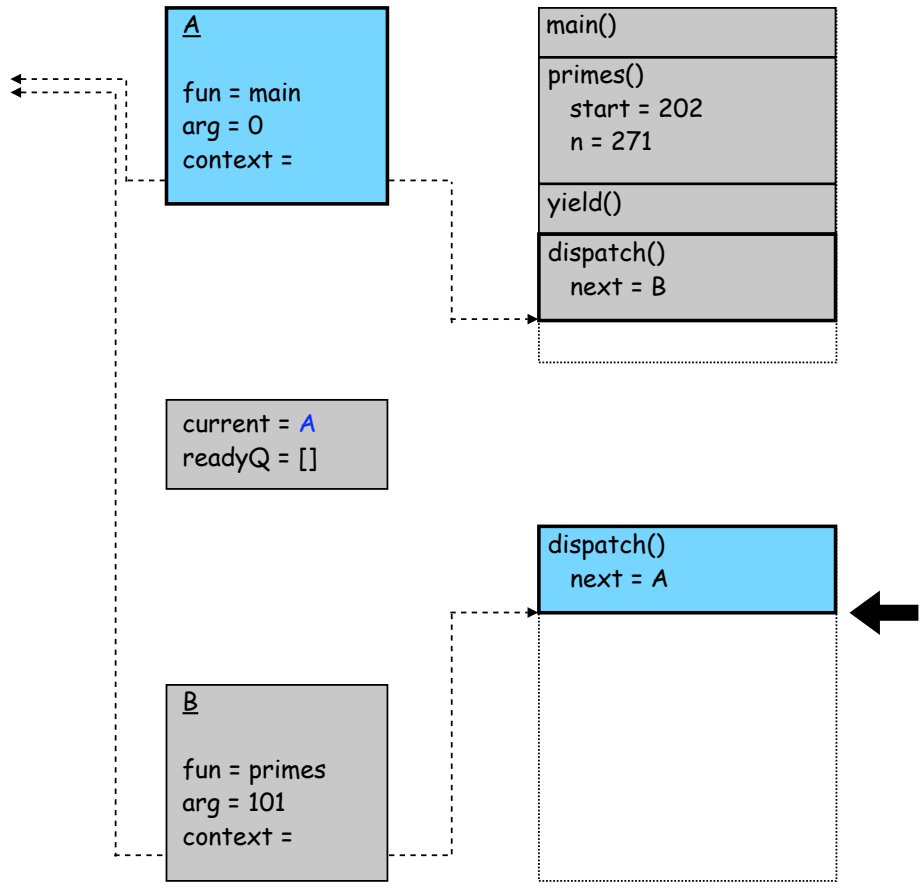
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



A

```

fun = main
arg = 0
context =

```

```

current = A
readyQ = []

```

B

```

fun = primes
arg = 101
context =

```

```

main()
primes()
  start = 202
  n = 271
yield()
dispatch()
  next = B

```



```

dispatch()
  next = A

```

```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

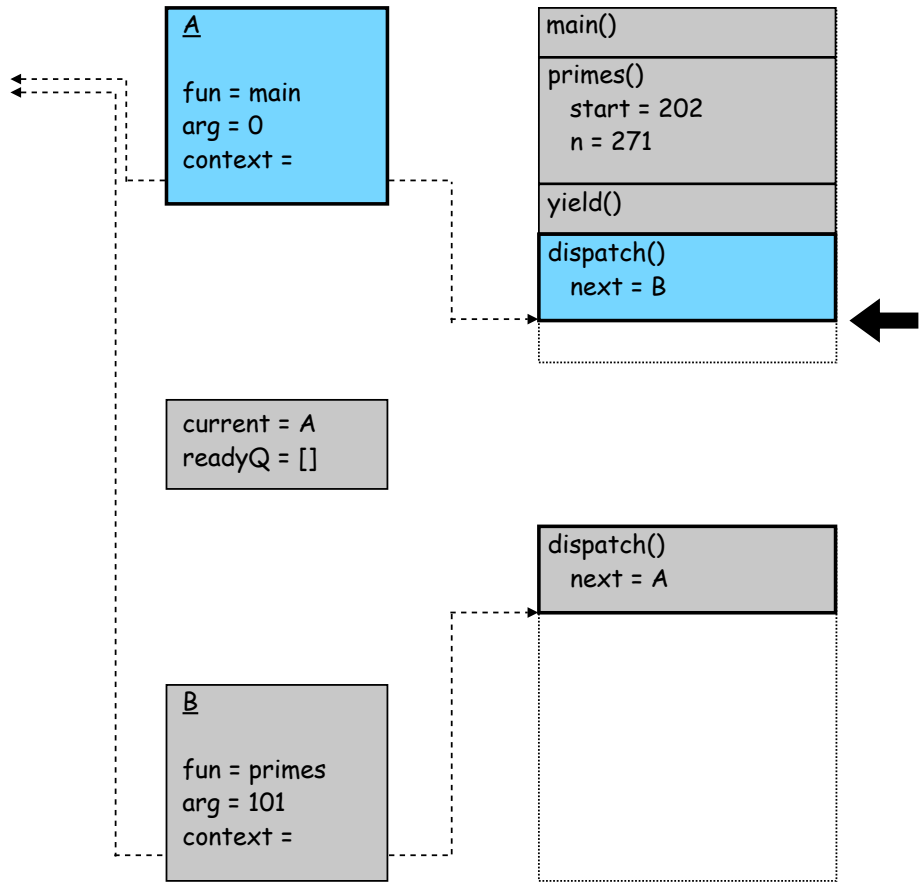
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

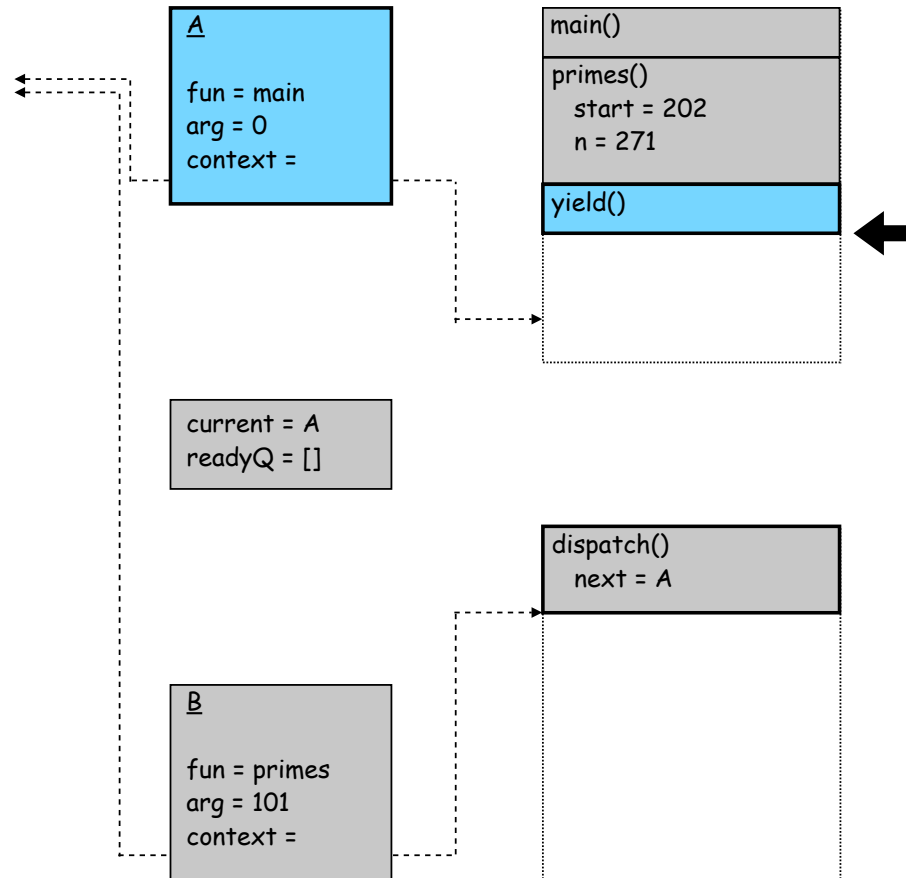
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

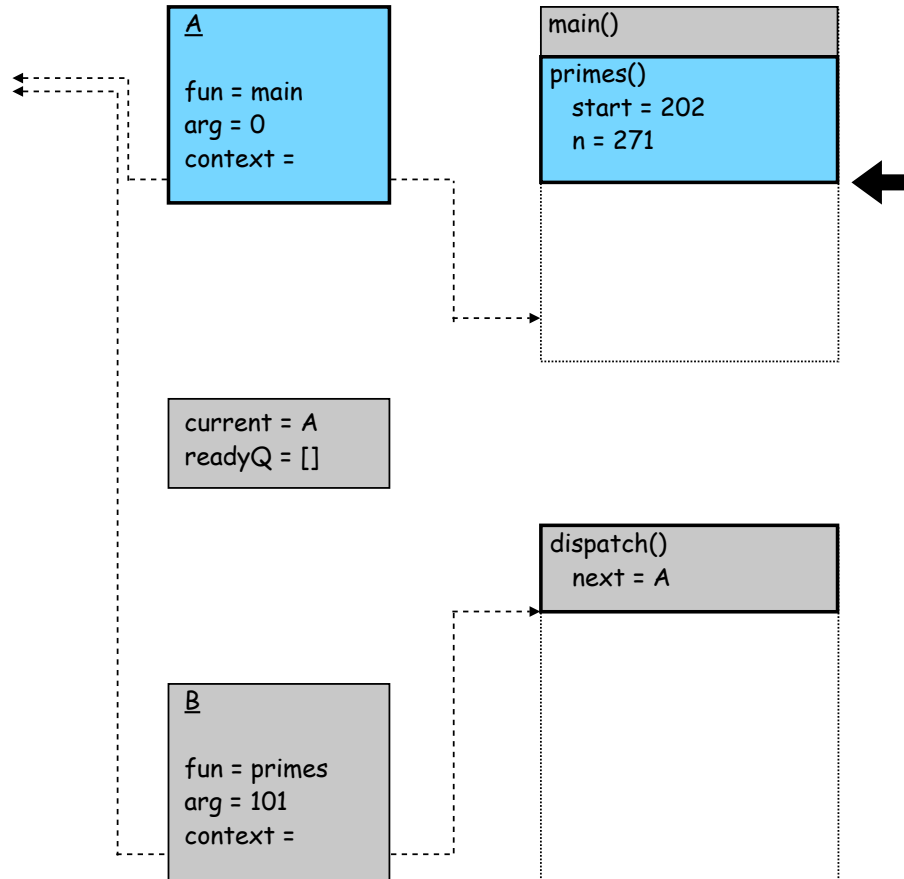
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

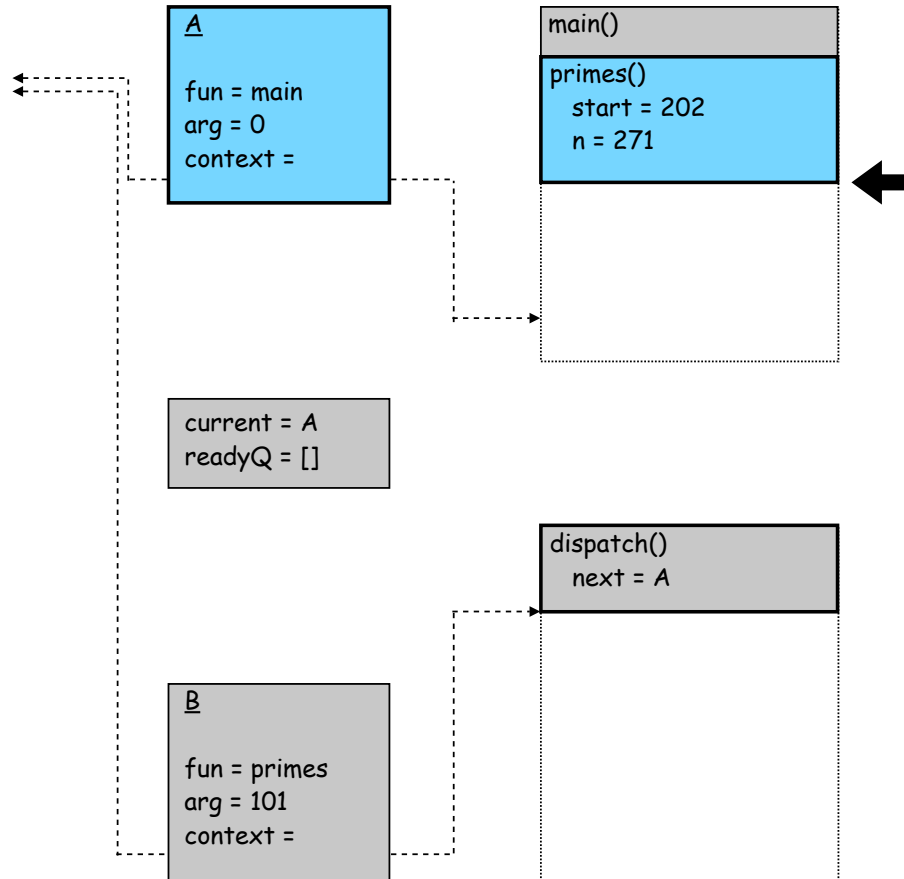
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



```

void yield() {
    enqueue(current, &readyQ);
    dispatch(dequeue(&readyQ));
}

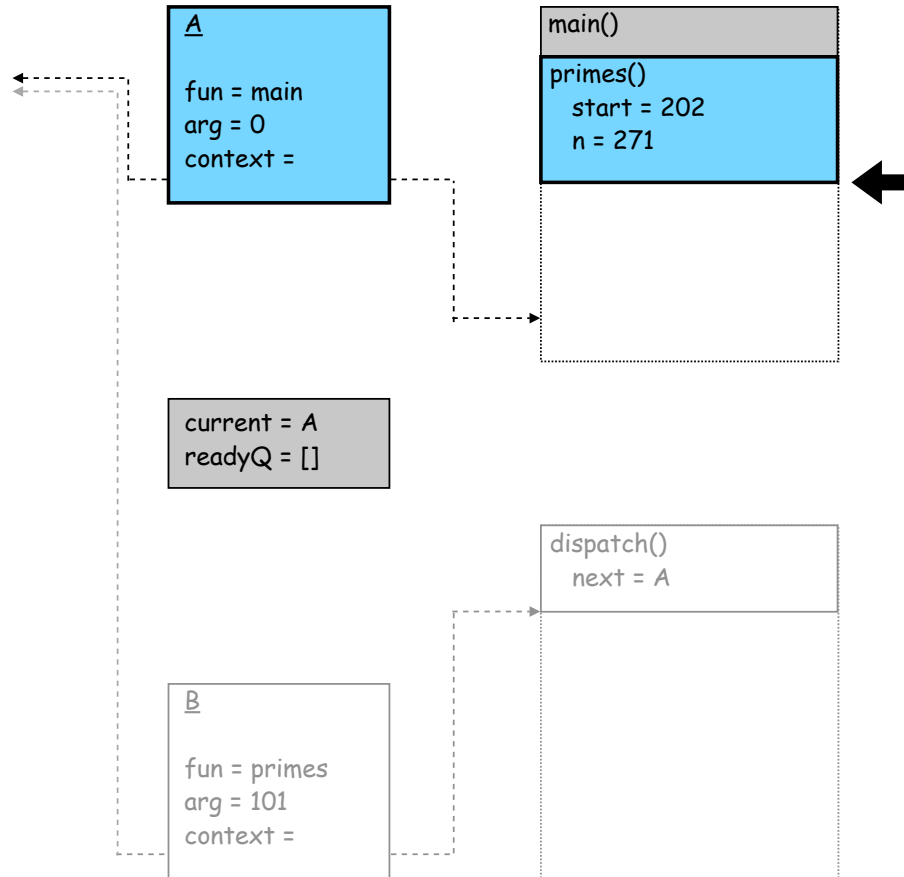
void dispatch(Thread *next) {
    if (setjmp(current->context) == 0) {
        current = next;
        longjmp(next->context, 1);
    }
}

void spawn(void (*fun)(int), int arg) {
    Thread *t = malloc(...)
    t->fun = fun;
    t->arg = arg;
    if (setjmp(t->context) == 1) {
        current->fun(current->arg);
        dispatch(dequeue(&readyQ));
    }
    STACKPTR(t->context) = malloc(...)
    enqueue(t, &readyQ);
    ...
}

primes(int start) {
    int n = start;
    while (...) {
        ...
        if (...) yield();
    }
}

main() {
    spawn(primes, 101);
    primes(202);
}

```



Calling yield()

Explicitly

```
ld a,r1
ld b,r2
add r1,r2
st r2,c
jsr yield
ld c,r0
cmp #37,r0
ble label34
...
```

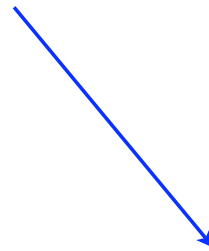
```
yield:
sub #2,sp
...
mov #0,r0
rts
```

Implicitly

```
ld a,r1
ld b,r2
add r1,r2
st r2,c
ld c,r0
cmp #37,r0
ble label34
...
```

```
vector_3:
push r0-r2
jsr yield
pop r0-r2
rti
```

Interrupt on pin 3



Installing ISRs

```
#include <avr/interrupt.h>
...
ISR (interrupt_name) {
    ... code ...
}
```

avr-gcc specific, may look very different on other platforms (if supported at all)

expands to

```
void _vector_x (void) __attribute__((signal)) {
    ... code ...
}
```

Preventing interrupts

`cli();`

... code that must not be interrupted ...

`sei();`

avr-gcc specific - may look very different on other platforms
(if supported at all)

What's next?

- Timer-based yields (by means of an interrupt handler)
- Mutual exclusion (implementing lock and unlock)
- Generic interrupt- and event-handling (lab 3)
- ... all the bells and whistles of a real os! (optional :-)
- Time to jump into lab 2!