Concurrency – Multithreading

Jinyang Li

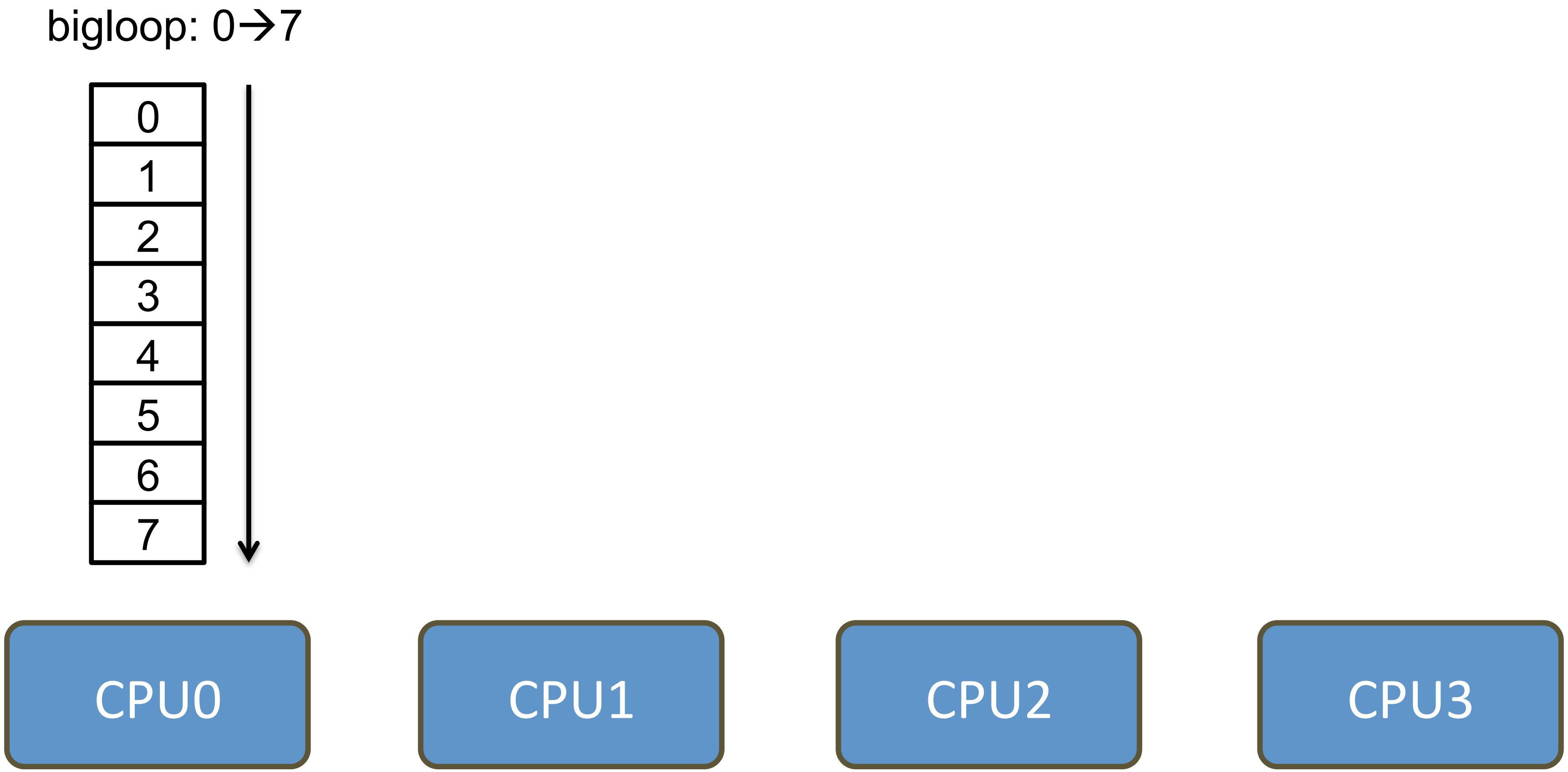
based on slides by Tiger Wang

Example

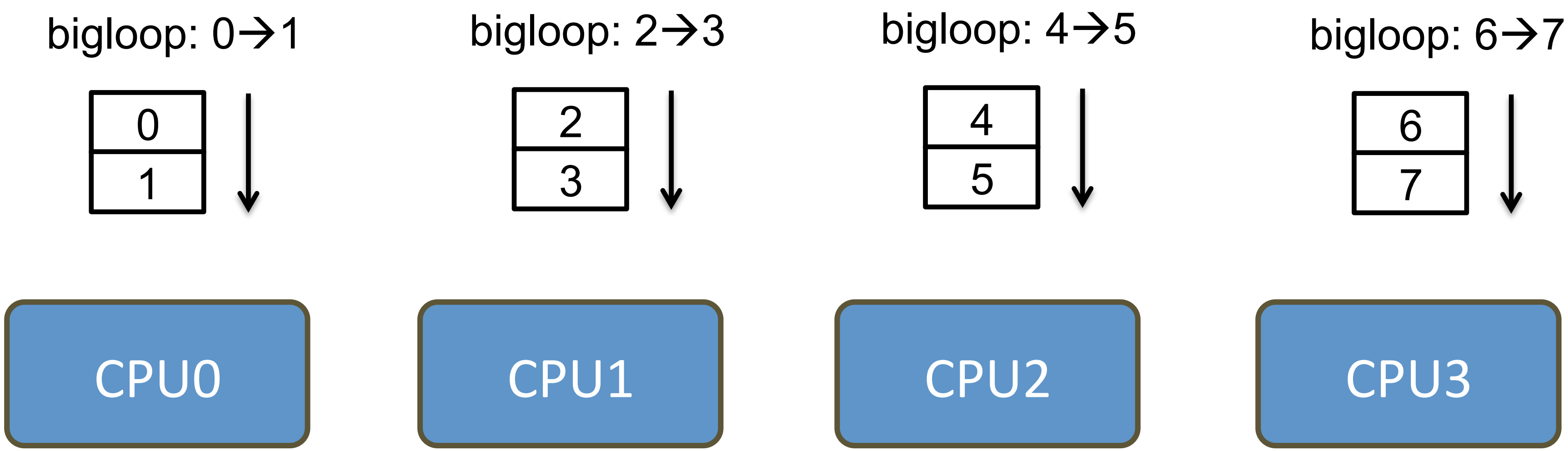
```
long bigloop(int *arr, int sz) {
  long r = 0;
  for(int i = 0; i < sz; i++)
    r += arr[i];
  return r;
}

int main() {
  ...
  long r = bigloop(arr, 1000000);
  ...
}
```

How to improve the performance with multicore?

Parallelization



Parallelization



bigloop: 0→1

0

1

CPU0

bigloop: 2→3

2

3

CPU1

bigloop: 4→5

4

5

CPU2

bigloop: 6→7

6

7

CPU3

Performance can be improved by 4X

Concurrency

What's concurrency?

- things happening "simultaneously”
  1. multiple CPU cores concurrently executing instructions
  2. CPU and I/O devices concurrently doing processing

Why write concurrent programs?

- speed up programs using multiple CPUs
- speed up programs by concurrently doing CPU processing and I/O.

How to write concurrent programs?

Use multiple processes

- Each process uses a different CPU
- Different processes runs different tasks
  - They have separate address spaces
  - Elaborate to communicate with each other

Use multiple threads

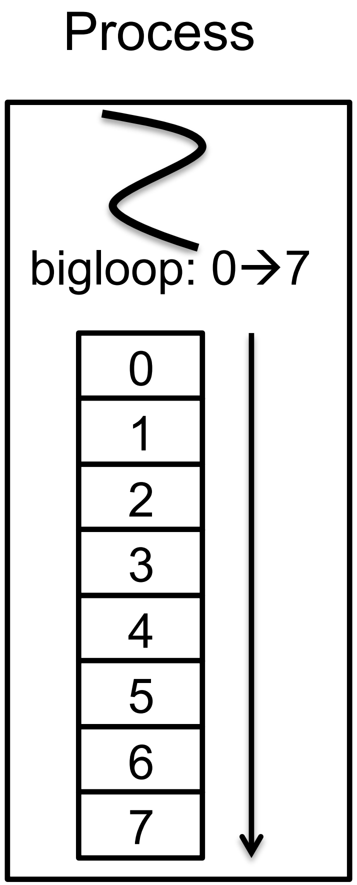
In this lecture

Use multiple processes

- Each process uses a different CPU
- Different processes runs different tasks
  - They have separated address space
  - Elaborate to communicate with each other

Use multiple threads

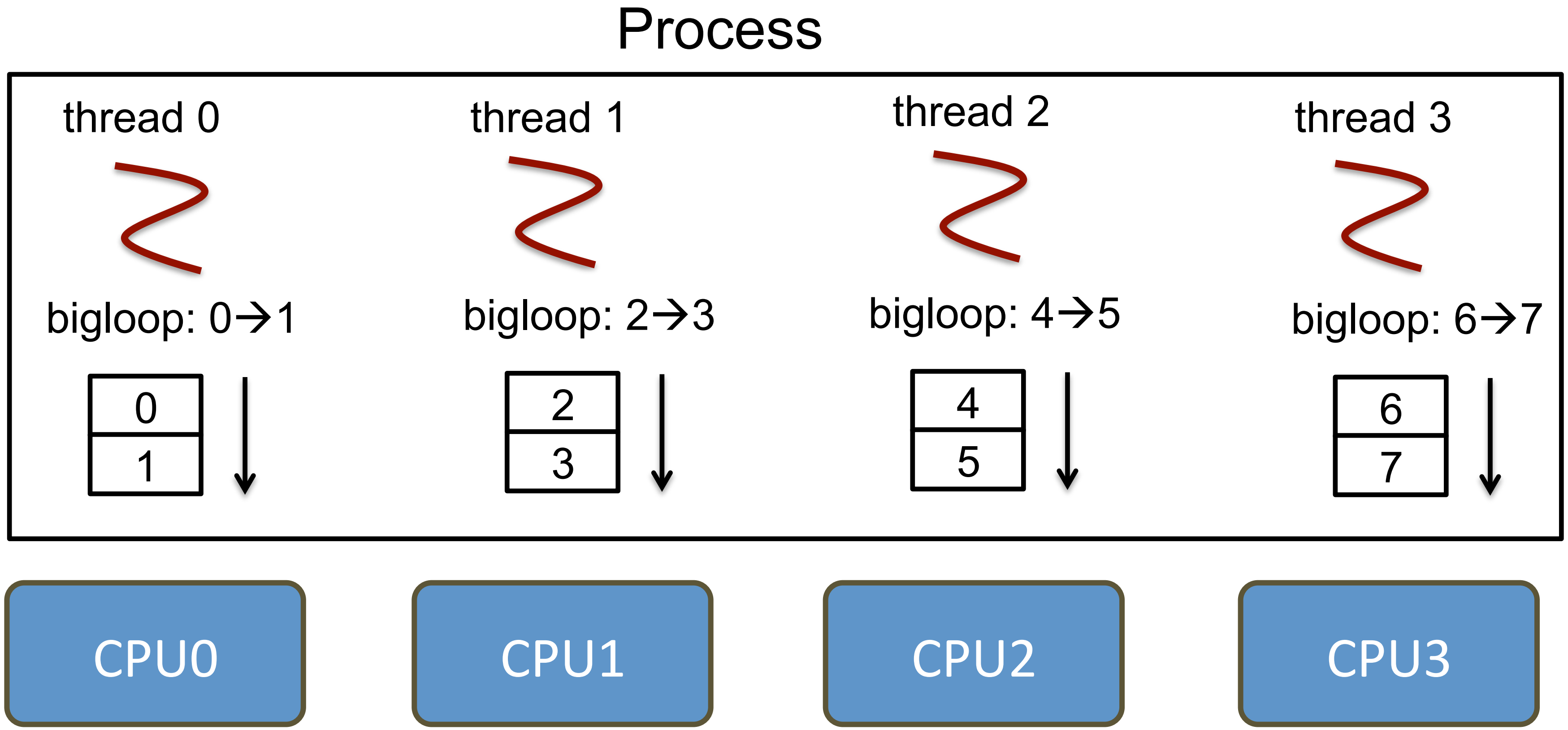
Multiple threads (Multithreading)



```
long bigloop(int *arr, int sz) {
  long r = 0;
  for(int i = 0; i < sz; i++)
    r += arr[i];
  return r;
}

int main() {
  ...
  long r = bigloop(arr, 8);
  ...
}
```

CPU0 CPU1 CPU2 CPU3

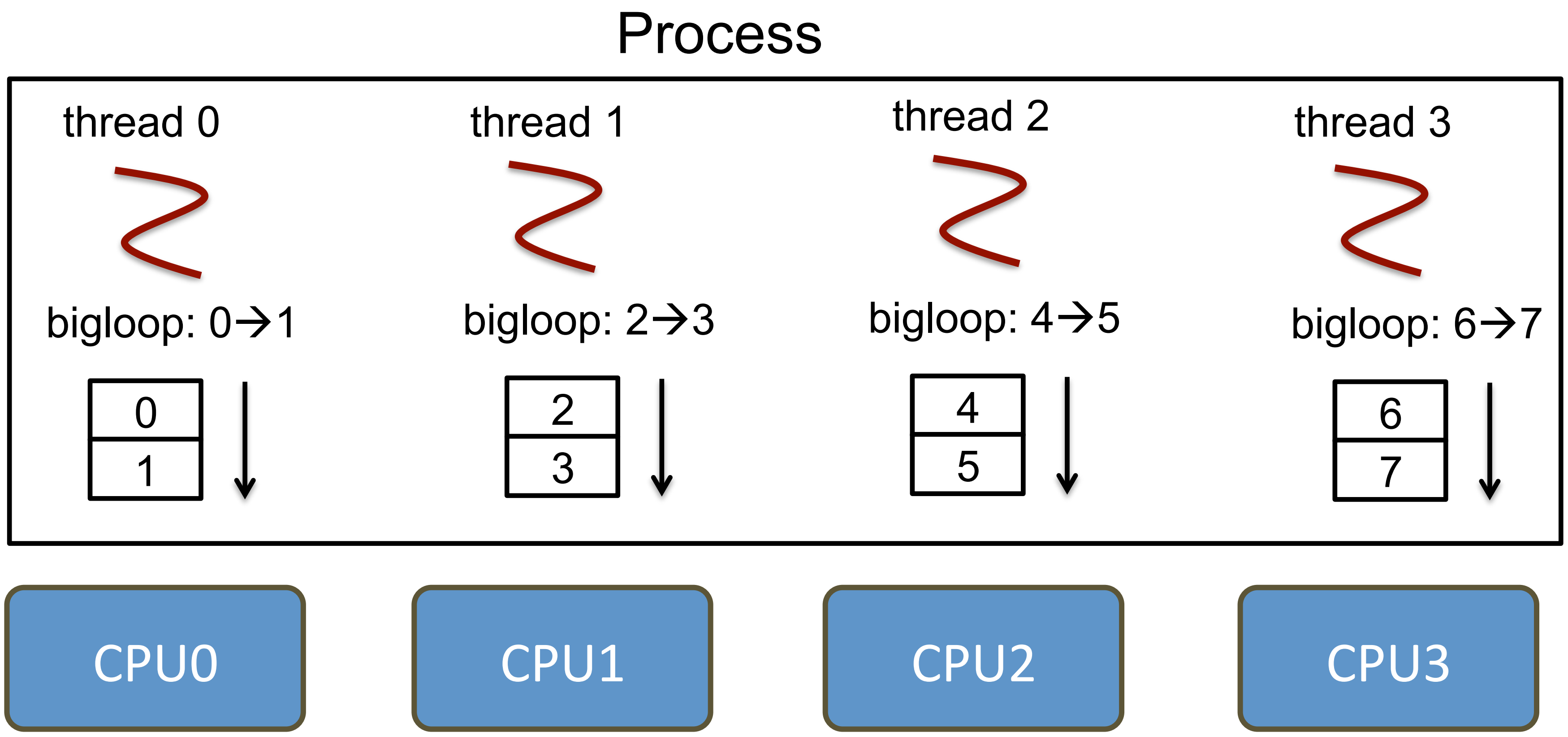
Multiple threads (Multithreading)



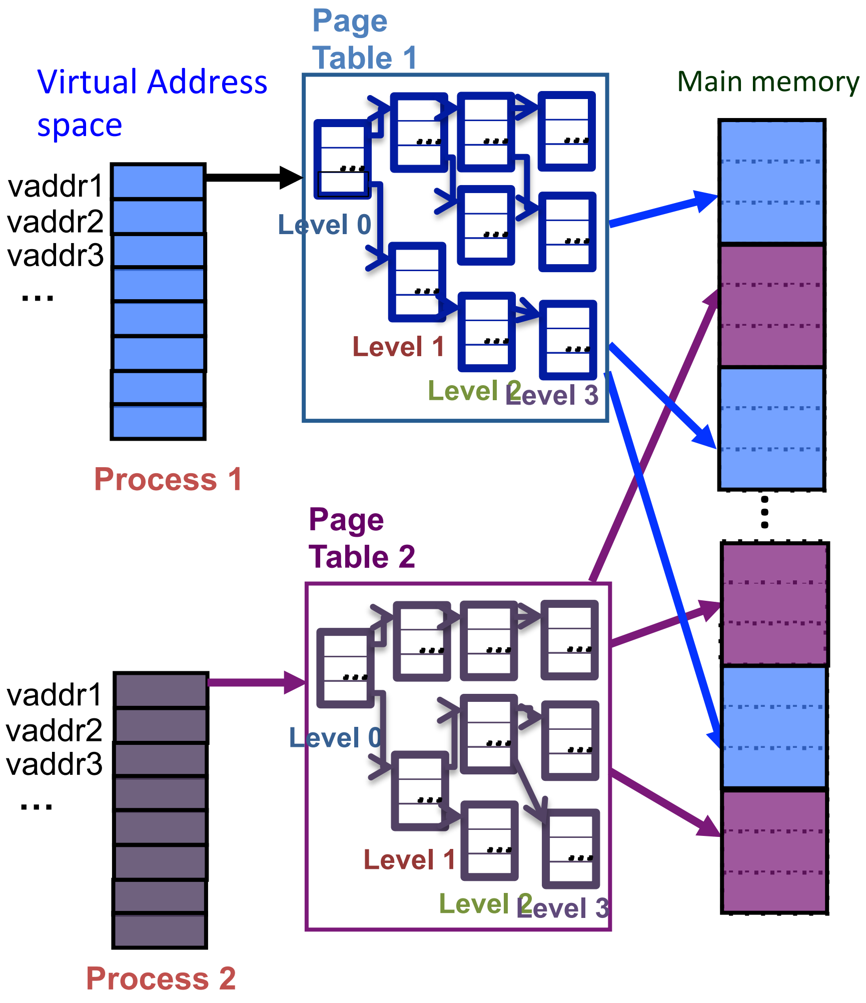
Multiple threads (Multithreading)

Single process, multiple threads

- Share the same memory space
- Has its own stack
- Has its own control flow

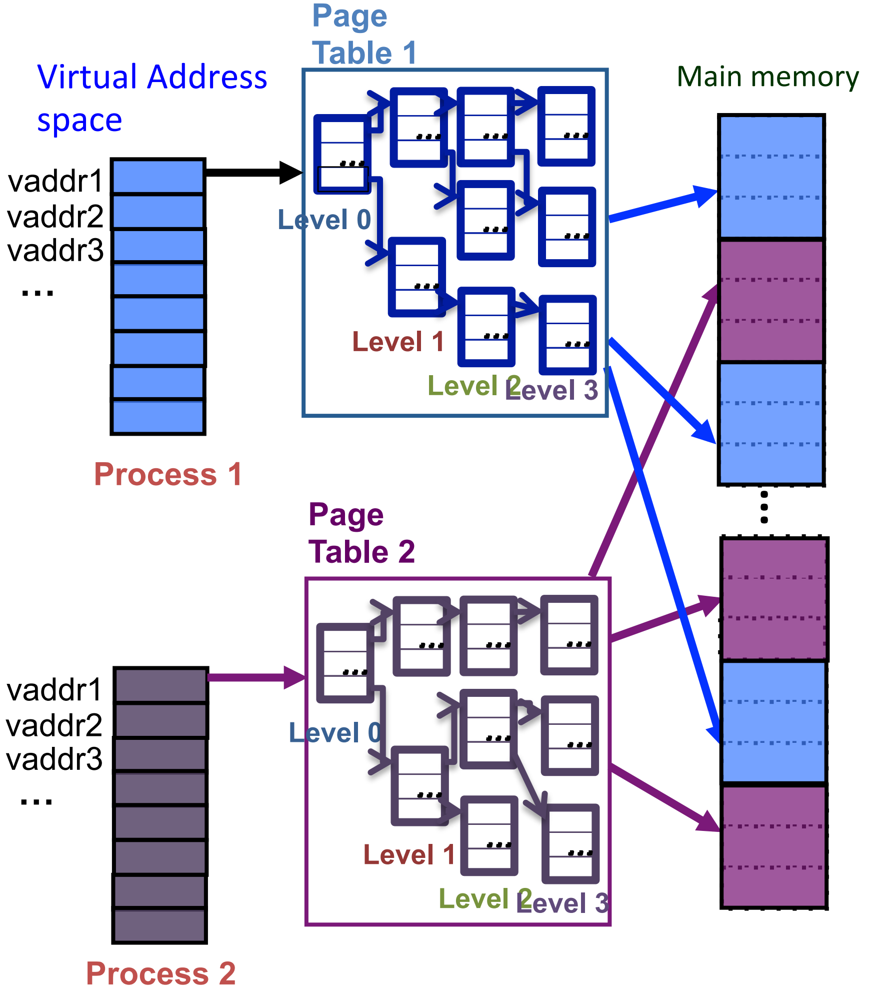


Share the memory space

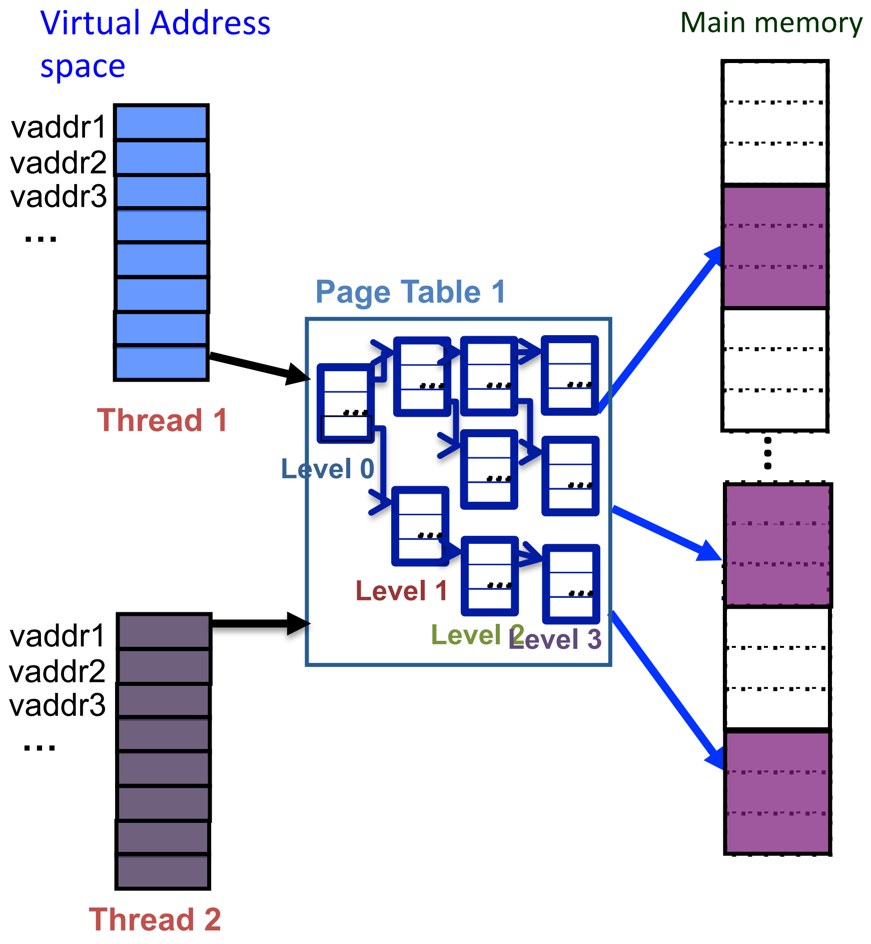


Different processes have different page tables

Share the memory space

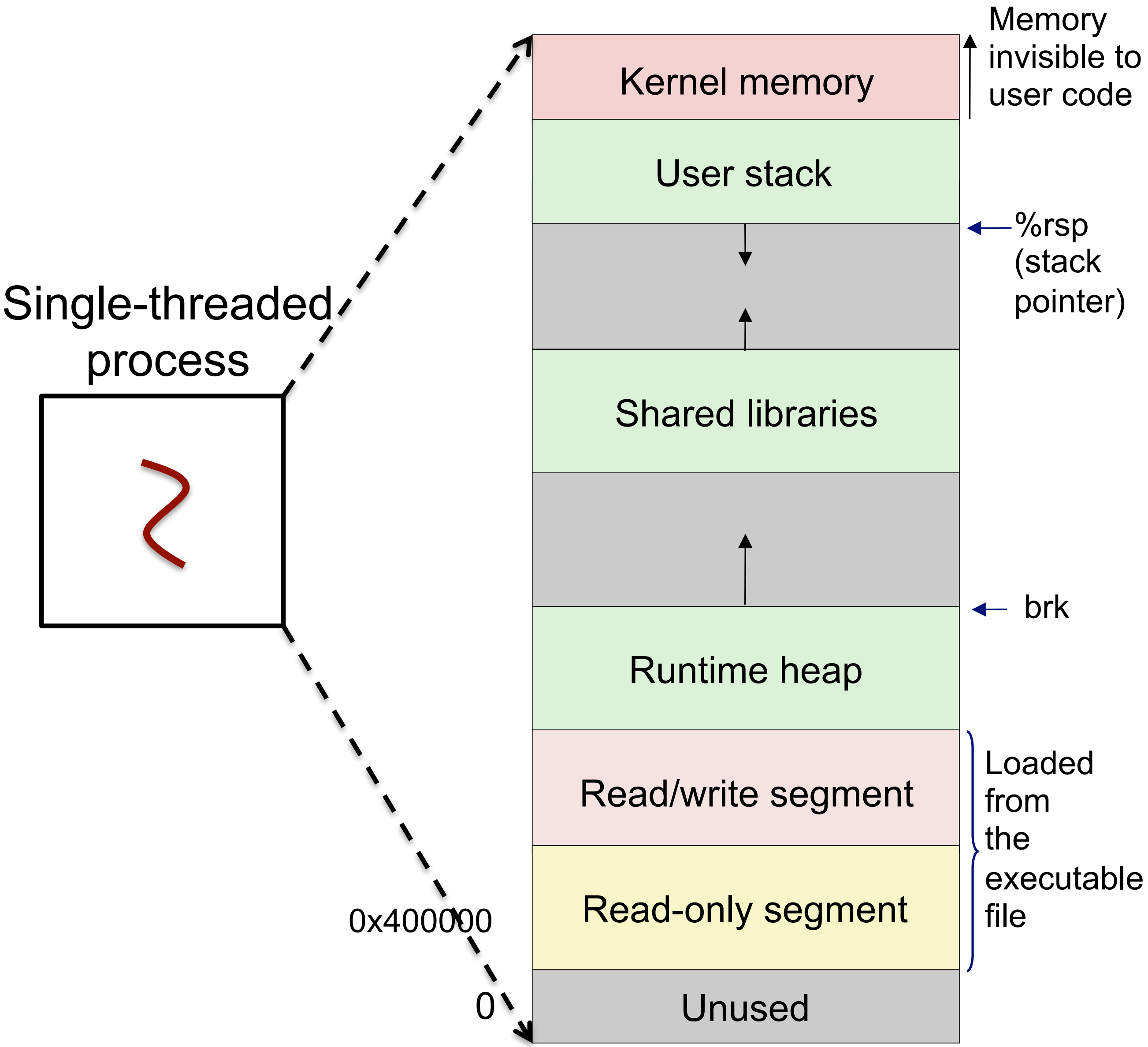


Different processes have different page tables



Different threads of the same process share the same page table

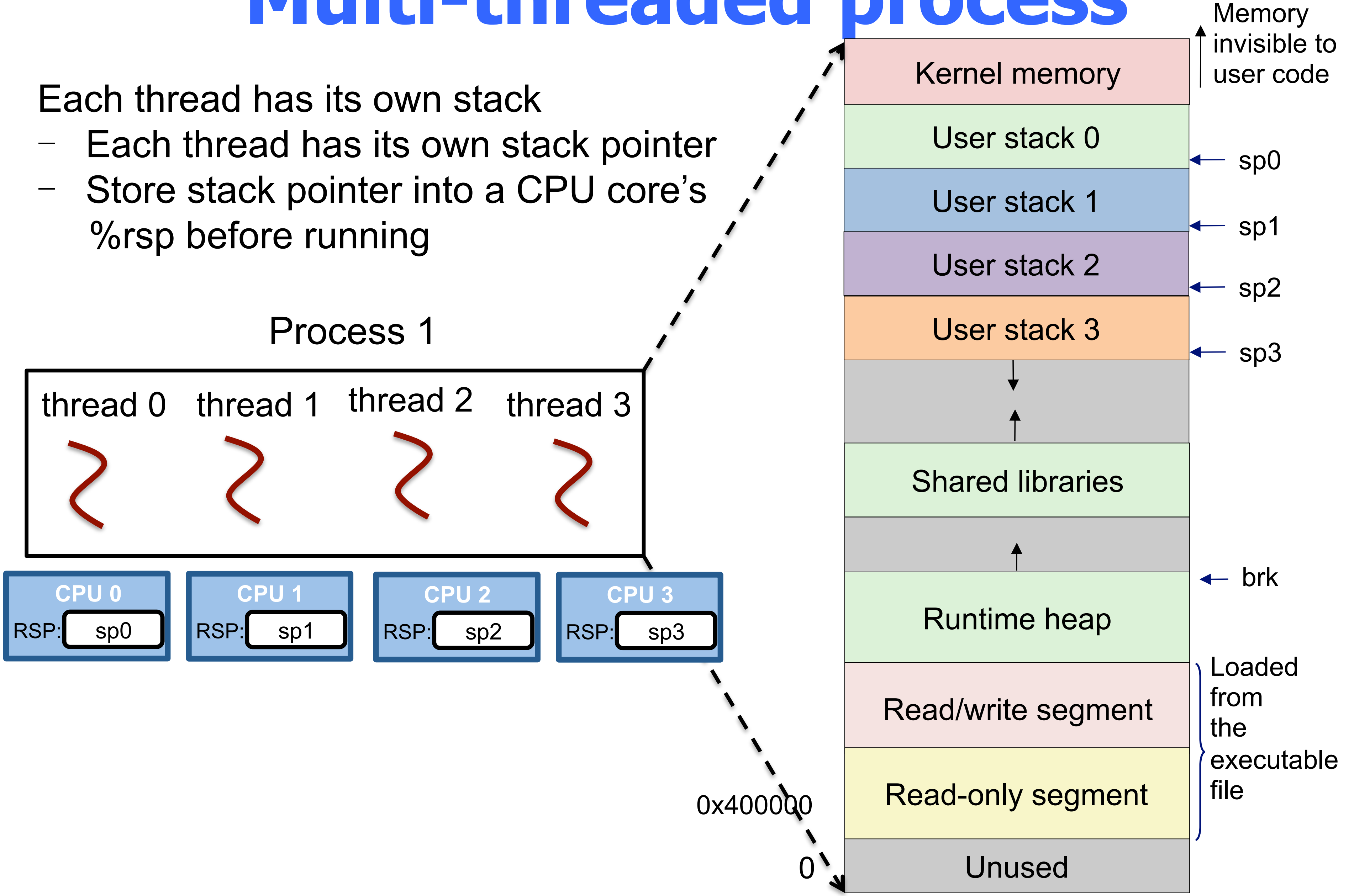
Single threaded process



Multi-threaded process

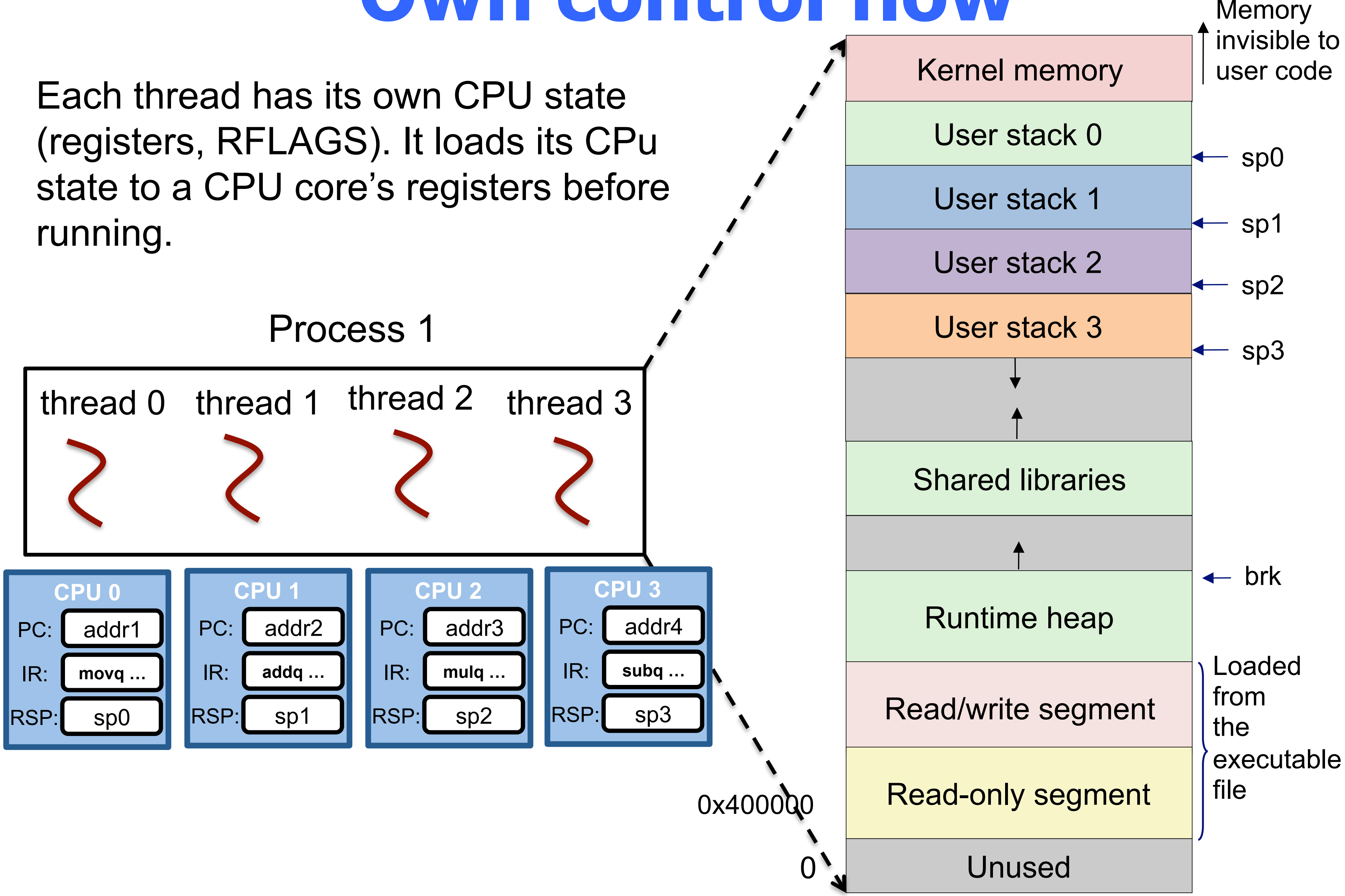
Each thread has its own stack

- Each thread has its own stack pointer
- Store stack pointer into a CPU core's %rsp before running



Own control flow

Each thread has its own CPU state (registers, RFLAGS). It loads its CPu state to a CPU core's registers before running.



POSIX thread interface

POSIX: Portable Operating System Interface

- POSIX defines the API for variants of Unix

Thread interface defined by POSIX

- pthread_create: create a new thread
- pthread_join: wait for the target thread terminated

pthread_create

```
#include <pthread.h>
int pthread_create(pthread_t *thread_id,
           const pthread_attr_t *attr,
           void *(*start_routine)(void*),
           void *arg);
```

Create a new thread

- It executes `start_routine` with `arg` as its sole argument.
- Its attribute is specified by `attr`
- Upon successful completion, it will store the ID of the created thread in the location referenced by `thread_id`.

Return value

- zero: success
- non-zero (error number): fail

Example 1 – Create

```c
void* func(void* arg) {
  printf("This is the created thread\n");
  return NULL;
}

int main(int argc, char* argv[]) {

  pthread_t tid;
  int r = pthread_create(&tid, NULL, &func, NULL);
  if(r != 0) {
    printf(“create thread failed”);
    return 1;
  }

  return 0;
}
```

```
gcc create.c -lpthread
```

Example 1 – Create

```c
void* func(void* arg) {
  printf("This is the created thread\n");
  return NULL;
}

int main(int argc, char* argv[]) {
  pthread_t tid;
  pthread_create(&tid, NULL, &func, NULL);
  return 0;
}
```

```
gcc create.c -lpthread
```

Process finishes when its main thread exits.

- All created threads are terminated

pthread_join

```
#include <pthread.h>
int pthread_join(pthread_t thread_id, void **ret_ptr);
```

Wait for the target thread to finish

- Upon success, the return value of the target thread is stored at the location pointed to by `ret_ptr`.

Return value

- zero: success
- non-zero (error number): fail

Example 2 – Join

```c
void* func(void* arg) {
  printf("This is the created thread\n");
  return NULL;
}

int main(int argc, char* argv[]) {

  pthread_t tid;
  pthread_create(&tid, NULL, &func, NULL);
  pthread_join(tid, NULL);
  return 0;
}
```

Example 3 – Parameter

Question – what is expected result ?

```
void* func(void* arg) {
  int p = *(int *)arg;
  p = p + 1;
  return &p;
}

int main(int argc, char* argv[]) {

  int param = 100;

  pthread_t tid;
  pthread_create(&tid, NULL, &func, (void *)&param);
  ...

  int *res = NULL;
  pthread_join(tid, &res);
  ...

  printf("result: addr %lx val %d\n", res, *res);
  return 0;
}
```

Example 3 – Parameter

```
void* func(void* arg) {
  int p = *(int *)arg;
  p = p + 1;
  return &p;
}
```

p is on the stack of the created thread
-- it is destroyed when the thread terminates

```
int main(int argc, char* argv[]) {

  int param = 100;

  pthread_t tid;
  pthread_create(&tid, NULL, &func, (void *)&param);
  ...

  int *res = NULL;
  pthread_join(tid, &res);
  ...

  printf("result: addr %lx val %d\n", res, *res);
  return 0;
}
```

Example 3 – Parameter

```
void* func(void* arg) {
  int p = *(int *)arg;
  p = p + 1;
  int *r = malloc(sizeof(int));
  *r = p;
  return (void *)r;
}

int main(int argc, char* argv[]) {

  int param = 100;

  pthread_t tid;
  pthread_create(&tid, NULL, &func, (void *)&param);
  ...

  int *res = NULL;
  pthread_join(tid, &res);
  ...

  printf("result: addr %lx val %d\n", res, *res);
  return 0;
}
```

Example 3 – Parameter

```
void* func(void* arg) {
  int p = *(int *)arg;
  p = p + 1;
  int *r = malloc(sizeof(int));
  *r = p;
  return (void *)r;
}

int main(int argc, char* argv[]) {

  int param = 100;

  pthread_t tid;
  pthread_create(&tid, NULL, &func, (void *)&param);
  ...

  int *res = NULL;
  pthread_join(tid, &res);
  ...

  printf("result: addr %lx val %d\n", res, *res);
  free(res)
  return 0;
}
```

Example 4 – Interleave

Question – what is the expected result ?

```
void* func(void* arg) {
  printf(“1”);
}

int main(int argc, char* argv[]) {

  printf(“0”);

  pthread_t tid;
  int r = pthread_create(&tid, NULL, &func, NULL);
  ...
  printf(“2”);

  ...
  return 0;
}
```

Example 4 – Interleave

Question – what is the expected result ?

Answer: 012 or 021

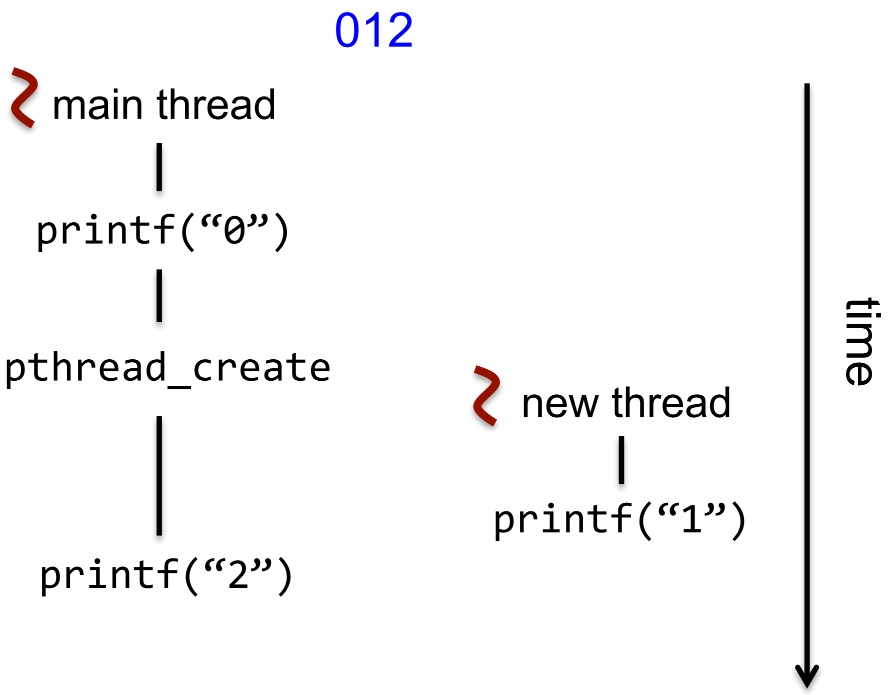
```
void* func(void* arg) {
  printf(“1”);
}

int main(int argc, char* argv[]) {

  printf(“0”);

  pthread_t tid;
  int r = pthread_create(&tid, NULL, &func, NULL);
  ...
  printf(“2”);

  ...
  return 0;
}
```

Example 4 – Interleave

```
void* func(void* arg) {
  printf(“1”);
}

int main(int argc, char* argv[]) {

  printf(“0”);

  pthread_t tid;
  int r = pthread_create(
            &tid, NULL, &func, NULL);
  ...
  printf(“2”);

  ...
  return 0;
}
```

Question – what is the expected result ?

Answer: 012 or 021

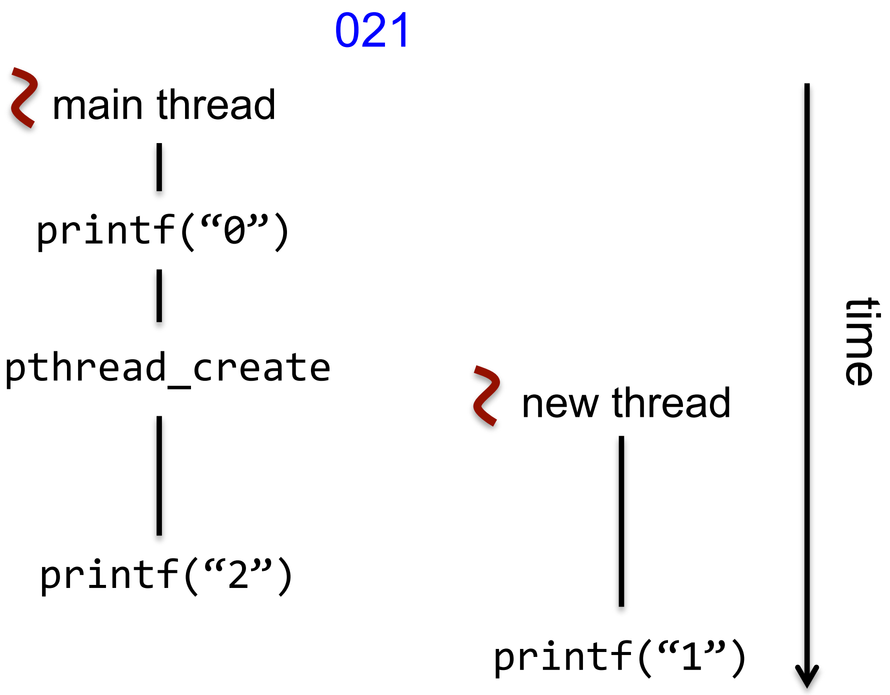


Example 4 – Interleave

```
void* func(void* arg) {
  printf(“1”);
}

int main(int argc, char* argv[]) {

  printf(“0”);

  pthread_t tid;
  int r = pthread_create(
            &tid, NULL, &func, NULL);
  ...
  printf(“2”);

  ...
  return 0;
}
```

Question – what is the expected result ?

Answer: 012 or 021



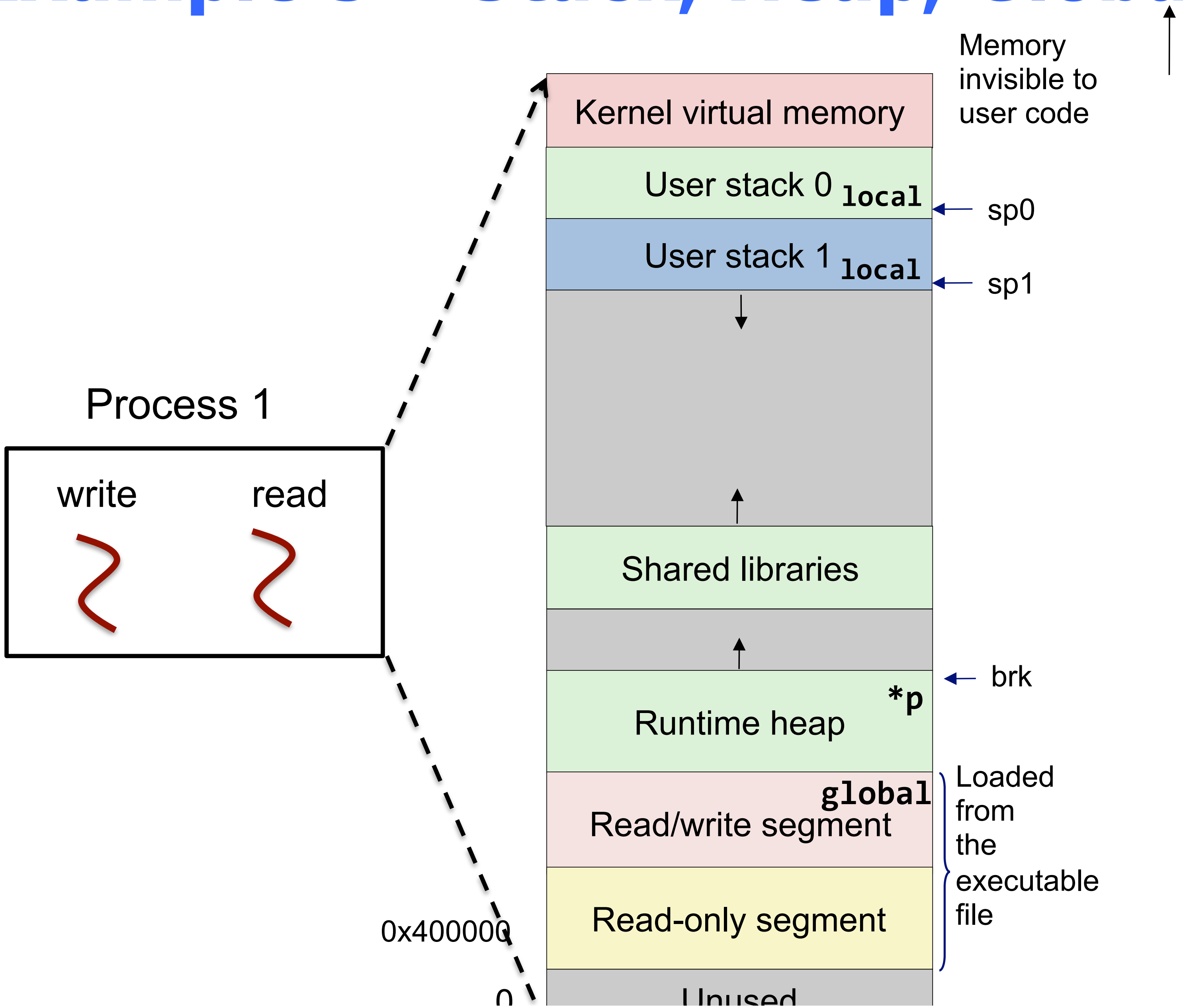
Example 5 – Stack, Heap, Global

```
int global = 0;

void* write(void* arg) {
  int local = 100;
  global = 100;
  *(int *)arg = 100;
}
```

```
void* read(void* arg) {
  int local = 0;
  printf("local %d global %d heap %d\n",
                  local, global, *(int *)arg);
  return NULL;
}
```

```
int main(int argc, char* argv[]) {
  int *p = (int *)malloc(sizeof(int));
  pthread_t tid1, tid2;
  pthread_create(&tid1, NULL, &write, (void *)p);
  ...
  pthread_join(tid1, NULL);
  pthread_create(&tid2, NULL, &read, (void *)p);
  ...
  return 0;
}
```

Example 5 – Stack, Heap, Global



Example 5 – Stack, Heap, Global

```
int global = 0;

void* write(void* arg) {
  int local = 0;
  local = 100;
  global = 100;
  int *ptr = (int *)arg;
  (*ptr) = 100;
}
```

```
void* read(void* arg) {
  int local = 0;
  printf("local %d global %d heap %d\n",
                  local, global, *(int *)arg);
  return NULL;
}
```

```
int main(int argc, char* argv[]) {
  int *p = (int *)malloc(sizeof(int));
  pthread_t tid1, tid2;
  pthread_create(&tid1, NULL, &write, (void *)p);
  ...
  pthread_join(tid1, NULL);
  pthread_create(&tid2, NULL, &read, (void *)p);
  ...
  return 0;
}
```

What are the output?

local 0 global 100 heap 100

Example 5 – Stack, Heap, Global

```
int global = 0;

void* write(void* arg) {
  int local = 0;
  local = 100;
  global = 100;
  int *ptr = (int *)arg;
  (*ptr) = 100;
}
```

```
void* read(void* arg) {
  int local = 0;
  printf("local %d global %d heap %d\n",
                  local, global, *(int *)arg);
  return NULL;
}
```

```
int main(int argc, char* argv[]) {
  int *p = (int *)malloc(sizeof(int));
  pthread_t tid1, tid2;
  pthread_create(&tid1, NULL, &write, (void *)p);
  ...
  ~~pthread_join(tid1, NULL);~~
  pthread_create(&tid2, NULL, &read, (void *)p);
  ...
  return 0;
}
```

What are the output?

local 0 global 0 heap 0

local 0 global 100 heap 0

local 0 global 100 heap 100

Example 6 – bigloop

Parallelize bigloop into two threads

```
#define LEN 1000000000

long bigloop(int *arr) {
  long r = 0;
  for(int i = 0; i < LEN; i++)
    r += arr[i];
  return r;
}

int main() {
  int *arr = malloc(LEN * sizeof(int));
  ...
  long r = bigloop(arr);
  ...
}
```

Example 6 – bigloop

```
#define LEN 1000000000

void* loop_thr1(void *arg){
  long *r = malloc(sizeof(long));
  int *arr = (int *)arg;

  for(int i = 0; i < LEN/2; i++)
    (*r) += arr[i];
  return (void *)r;
}
```

```
void* loop_thr2(void *arg){
  long *r = malloc(sizeof(long));
  int *arr = (int *)arg;

  for(int i = LEN/2; i < LEN; i++)
    (*r) += arr[i];
  return (void *)r;
}
```

Can we merge loop_thr1 with loop_thr2?

```
int main() {
  int *arr = malloc(LEN * sizeof(int));
  ...
  pthread_t tid1, tid2;
  pthread_create(&tid, NULL, &loop_thr1, (void *)arr);
  pthread_create(&tid, NULL, &loop_thr2, (void *)arr);
  long *res1, *res2;
  pthread_join(tid, &res1);
  pthread_join(tid, &res2);
  printf(“result is %ld\n”, (*res1) + (*res2));
 }
```

Example 6 – bigloop

```
#define LEN 1000000000

typedef struct {
 int *arr;
 int len;
} loop_info;

int main() {
  int *arr = malloc(LEN * sizeof(int));
  ...
  pthread_t tids[2];
  for (int i = 0; i < 2; i++) {
     loop_info *info = (loop_info *)malloc(sizeof(loop_info));
     info->arr = arr + i * LEN/2;
     info->len = LEN/2;
     pthread_create(&tids[i], NULL, &loop, (void *)info);
  }
  for (int i = 0; i < 2; i++) {
     long *res;
     pthread_join(tids[i], &res);
     result += (*res);
 }
```

```
void* loop(void *arg){
  loop_info *info = (loop_info *)arg;
  long *r = malloc(sizeof(long));
  for(int i = 0; i < info->len; i++)
    (*r) += info->arr[i];
  return (void *)r;
}
```