Concurrency – Multithreading

Zhaoguo Wang

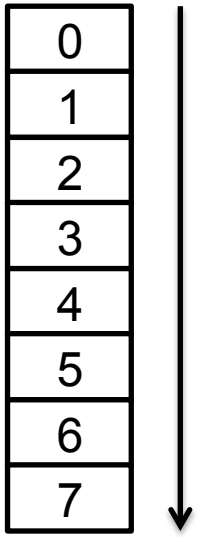
Example

```
long bigloop(int *arr) {
  long r = 0;
  for(int i = 0; i < 8; i++)
    r += arr[i];
  return r;
}

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

How to improve the performance with multicore?

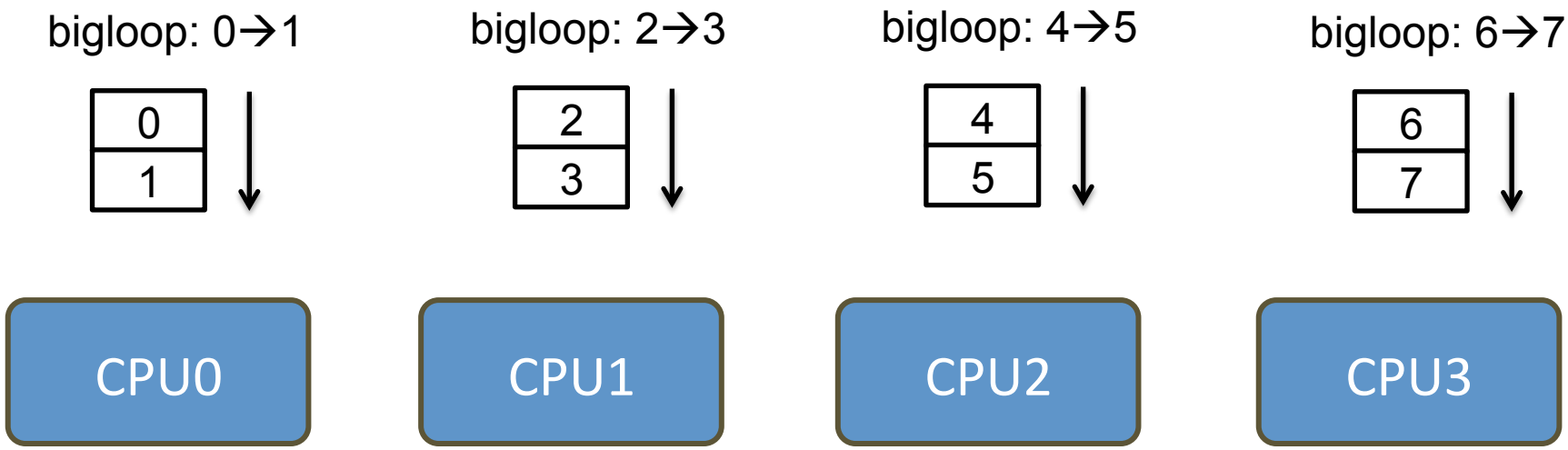
Parallelization

bigloop: 0→7



CPU0 CPU1 CPU2 CPU3

Parallelization

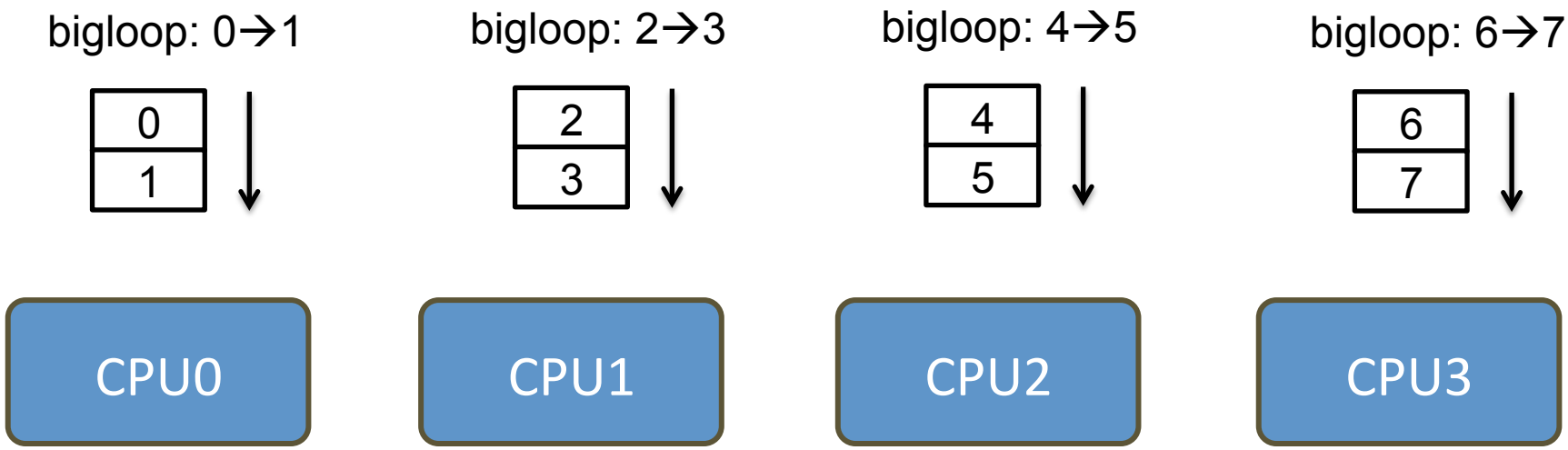


Performance can be improved by 4X

Parallelization

What's concurrency?

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



Performance can be improved by 4X

Concurrency

What's concurrency?

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

Why write concurrent programs?

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

In this lecture

What's concurrency?

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

Why write concurrent programs?

- speed up programs using multiple CPUs
- speed up programs by interleaving 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 separated address space
  - It is difficult to communicate with each other

Use multiple threads

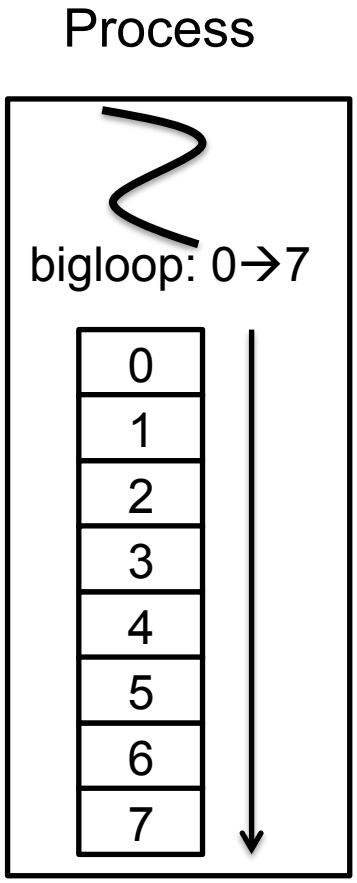
In this lecture

Use multiple processes

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

Use multiple threads

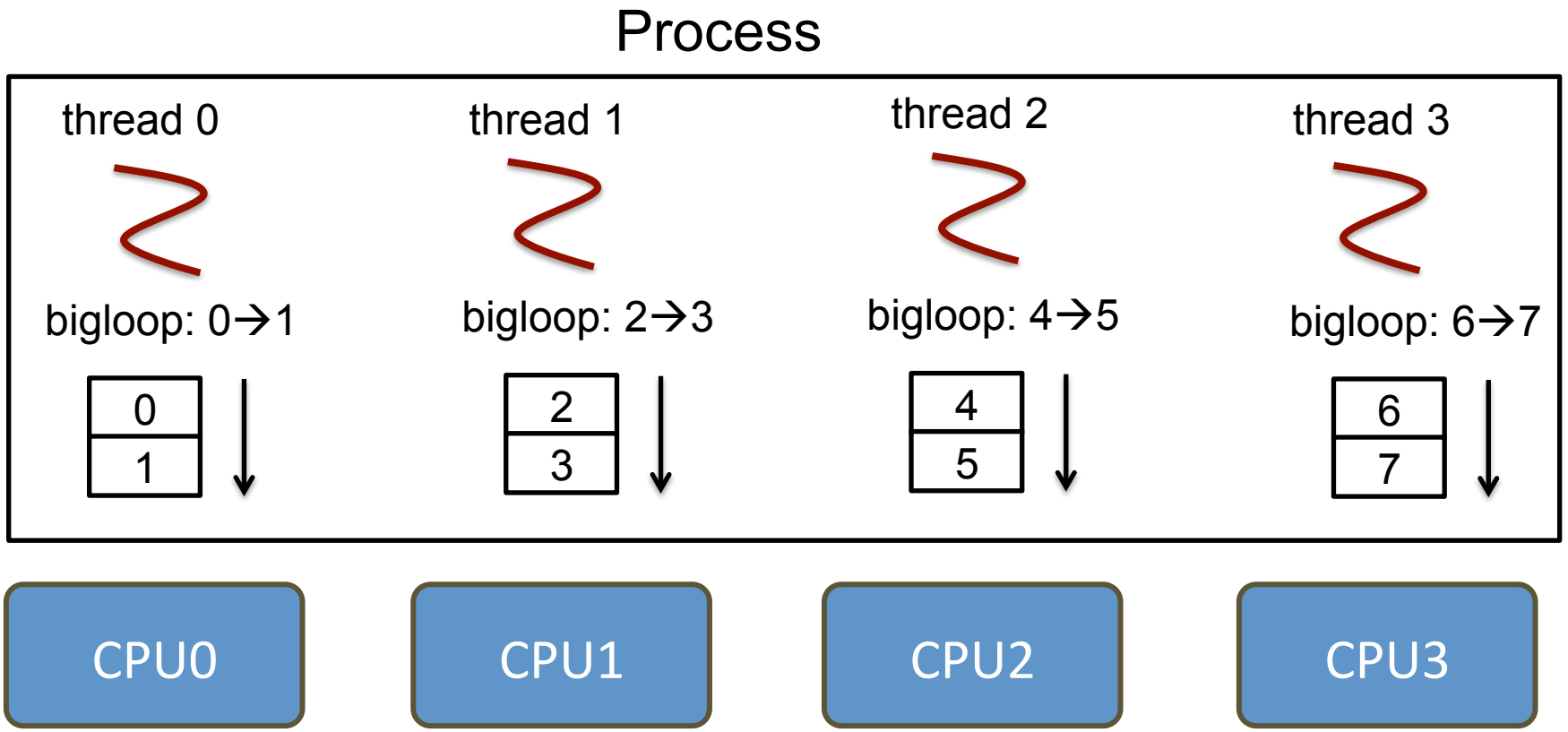
Multiple threads (Multithreading)




```
long bigloop(int *arr) {
  long r = 0;
  for(int i = 0; i < 8; i++)
    r += arr[i];
  return r;
}

int main() {
  int *arr = malloc(8 * sizeof(int));
  ...
  long r = bigloop(arr, 1);
  ...
}
```

CPU0 CPU1 CPU2 CPU3

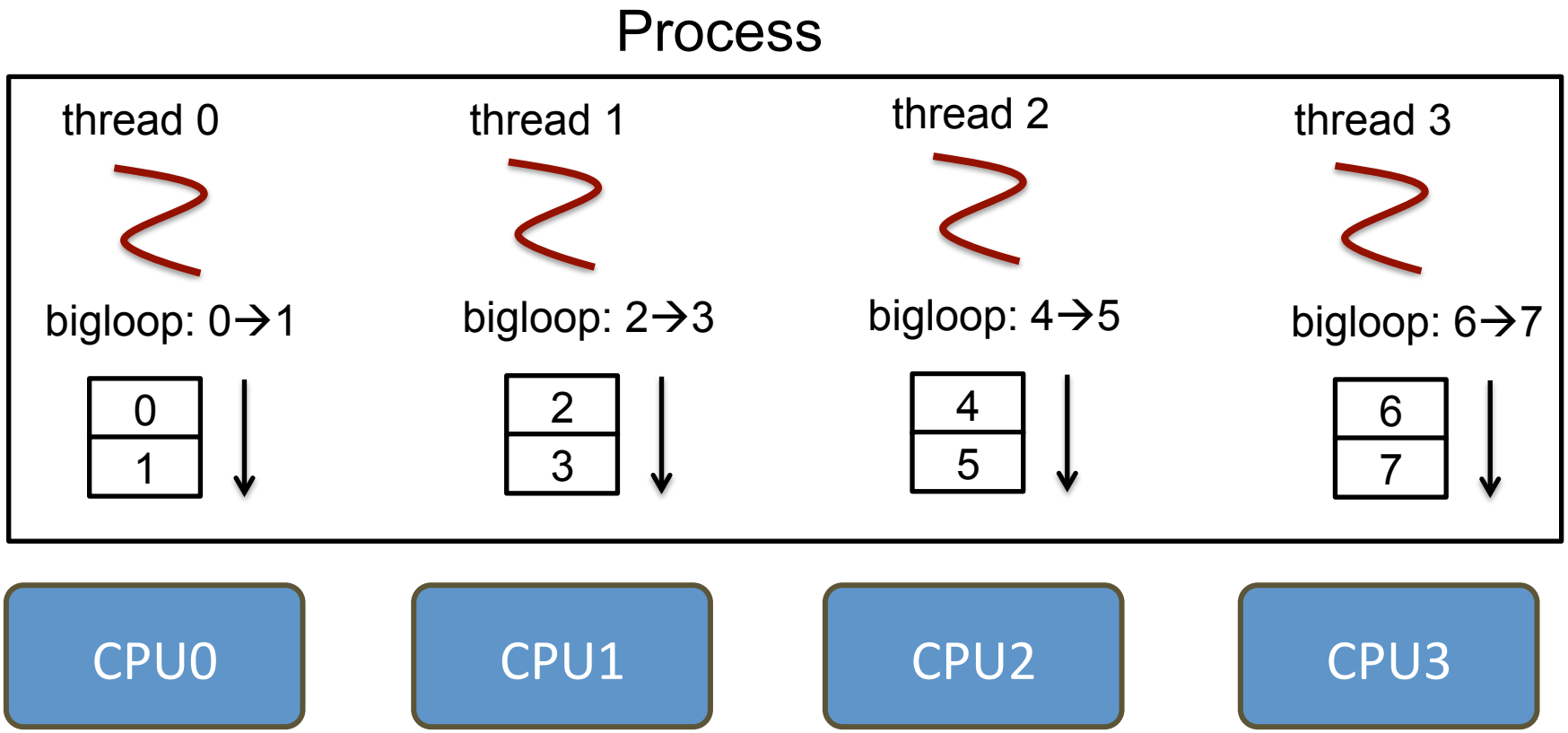
Multiple threads (Multithreading)



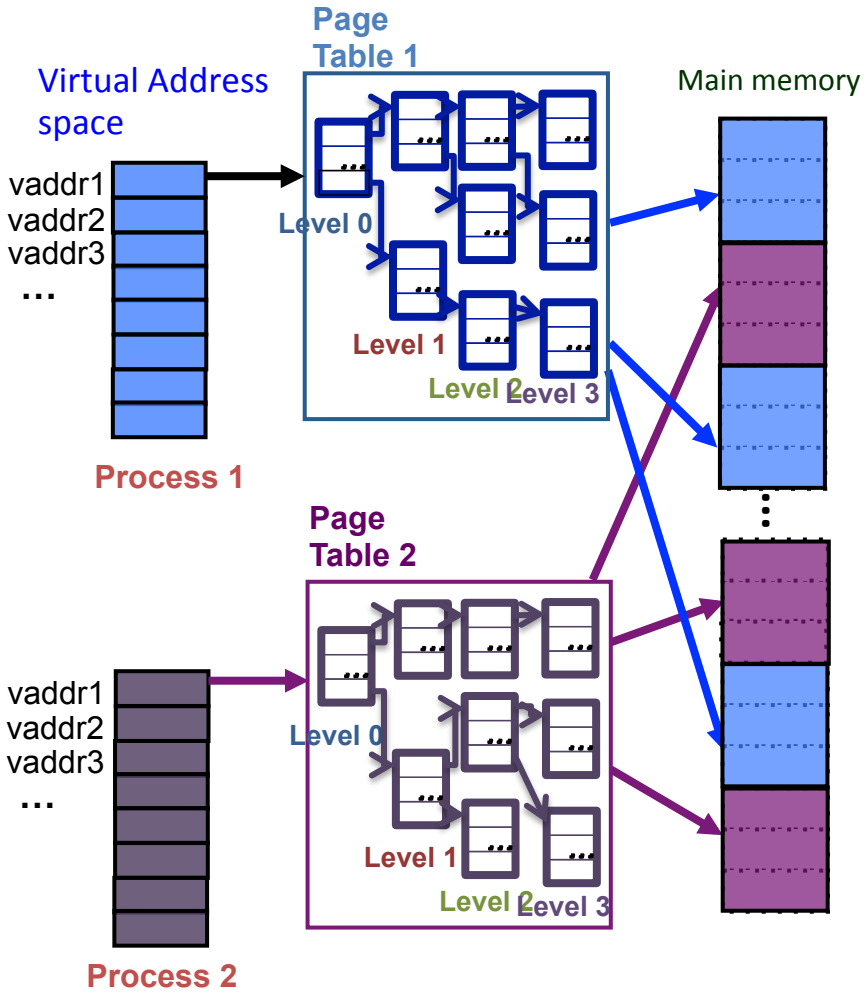
Multiple threads (Multithreading)

Single process, multiple threads

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

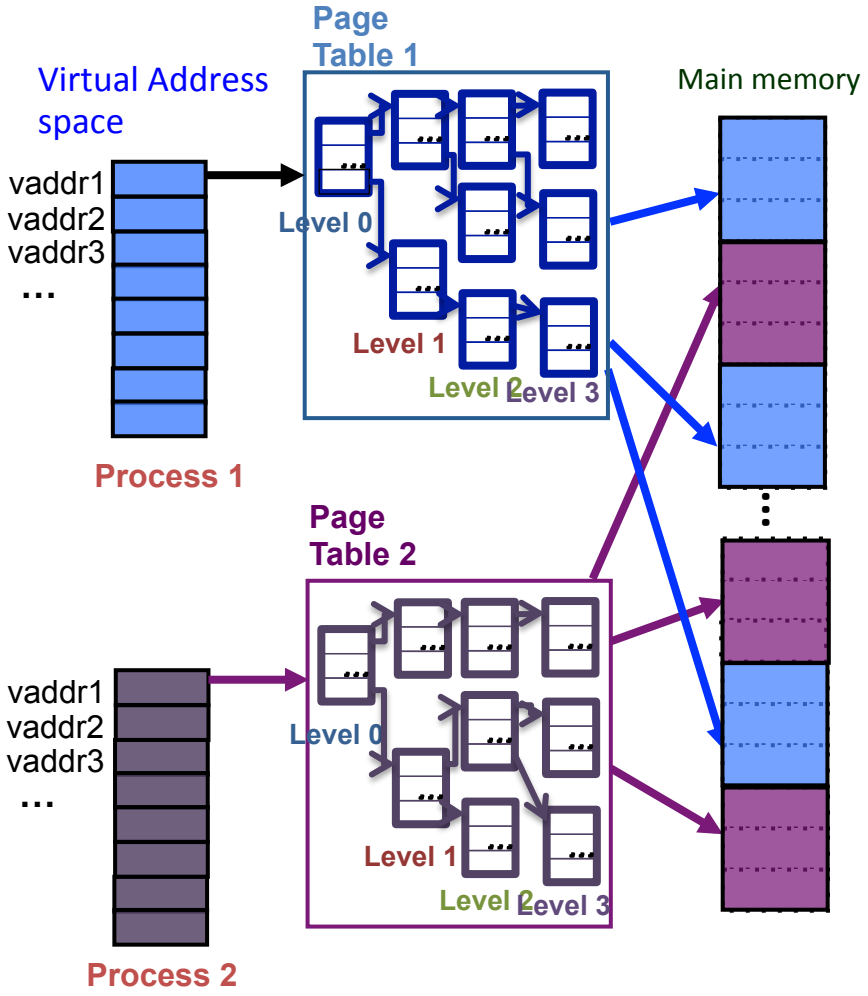


Share the memory space

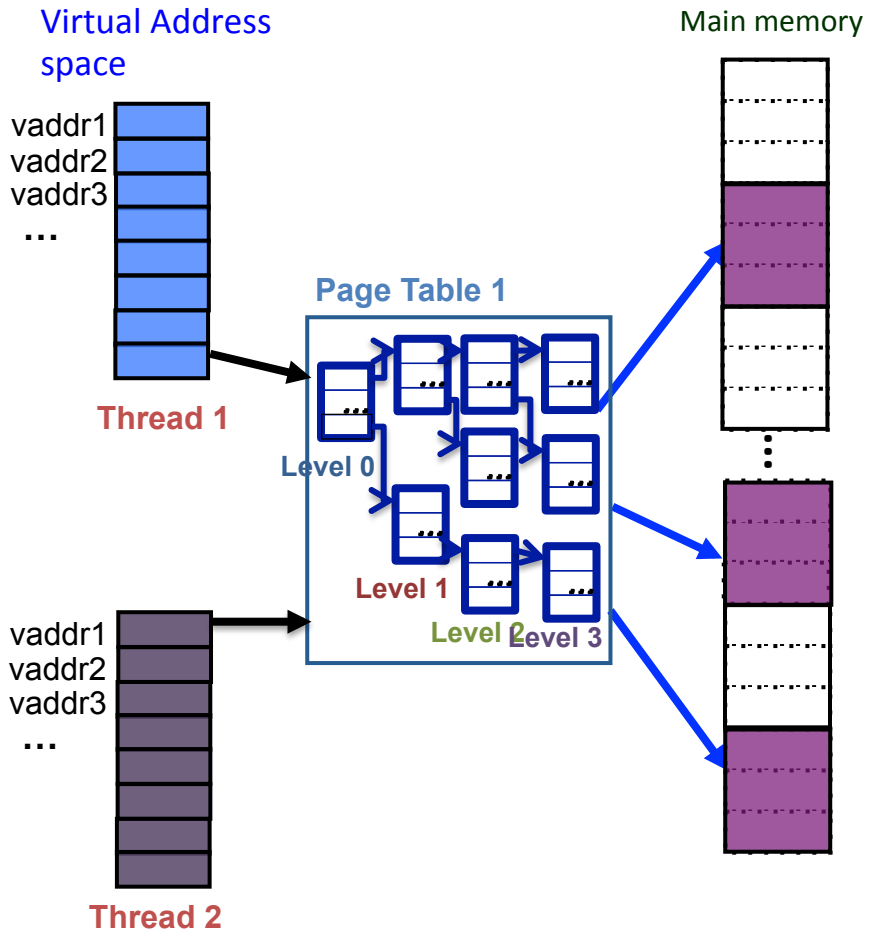


Different processes have different page tables

Share the memory space

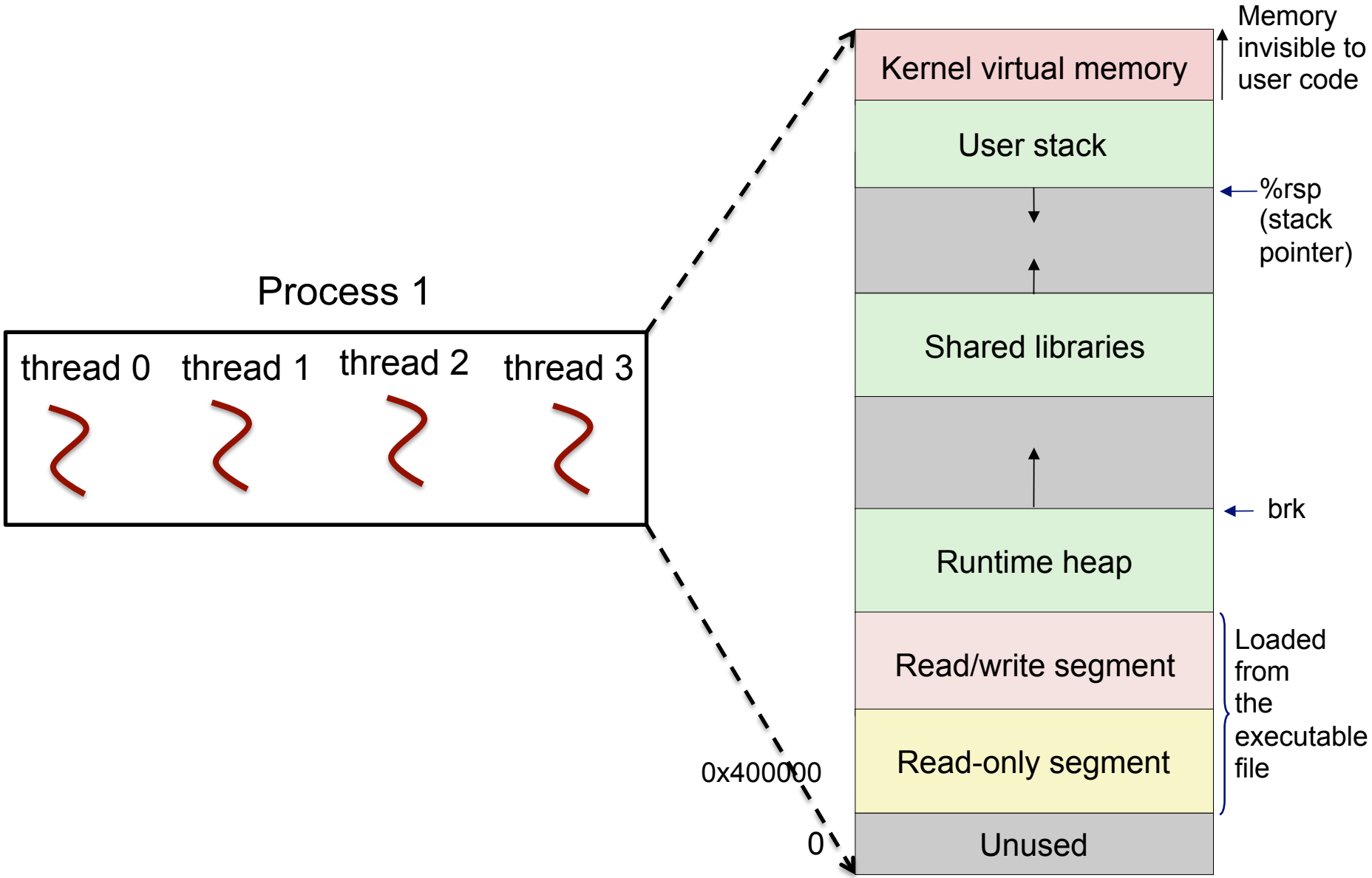


Different processes have different page tables



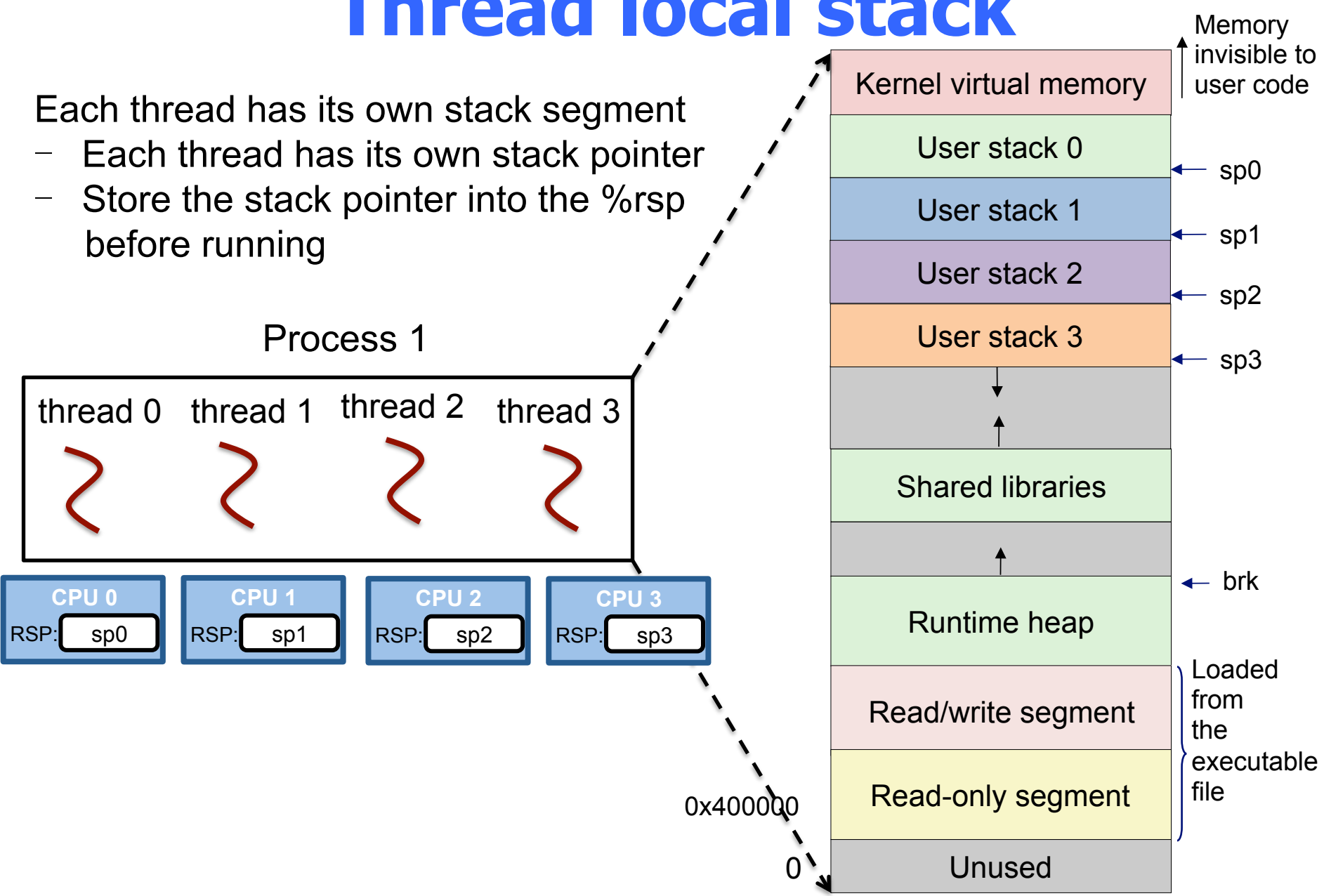
Different threads of the same process share the same page table

Thread local stack



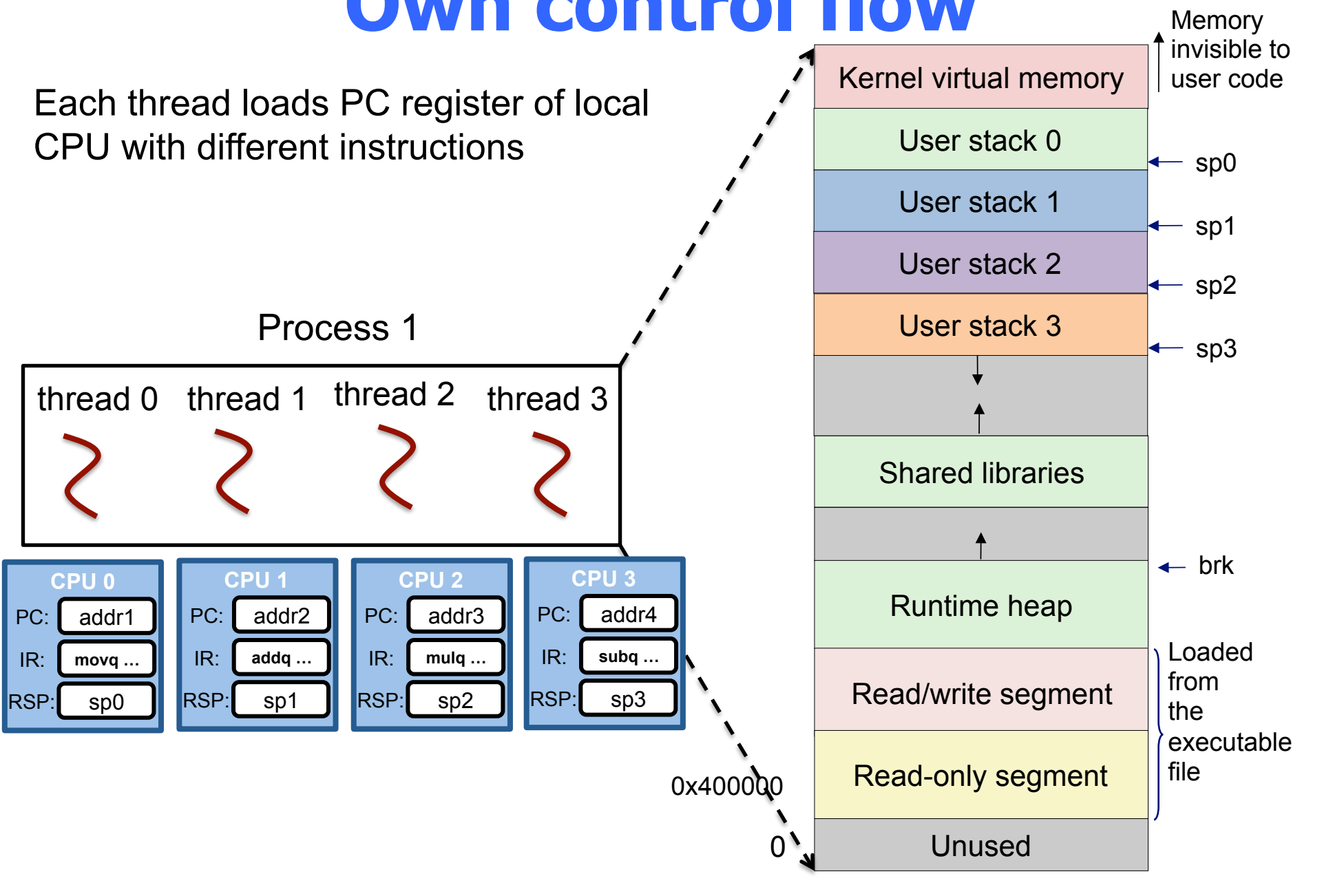
Thread local stack

Each thread has its own stack segment
- Each thread has its own stack pointer
- Store the stack pointer into the %rsp before running



Own control flow

Each thread loads PC register of local CPU with different instructions



POSIX thread interface

POSIX: Portable Operating System Interface

- POSIX defines the application programming interface (API) for software compatibility with variants of Unix and other operating systems

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

```
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

```
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

Main thread returns before the created thread finishes.

- Automatically terminate and reclaim the created thread.

pthread_join

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

Wait for the target thread to finish

- The target thread is specified by `thread`
- Upon success, the return value of the created thread will be available in the location referenced by `ret_ptr`.

Return value

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

Example 2 – Join

```
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)
    ...

  r = pthread_join(tid, NULL);
  if(r != 0)
    ...
  return 0;
}
```

Example 3 – Parameter

```
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;
  int r = pthread_create(&tid, NULL, &func, (void *)&param);
  ...

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

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

Question – what is expected result ?

Example 3 – Parameter

```
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;
  int r = pthread_create(&tid, NULL, &func, (void *)&param);
  ...

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

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

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

Example 3 – Parameter

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

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

  int param = 100;

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

  int *res = NULL;
  r = 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 = (void *)malloc(sizeof(int));
  *r = p;
  return (void *)r;
}

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

  int param = 100;

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

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

  printf("result: addr %lx val %d\n", res, *res);
  free(res)
  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 ?

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 ?

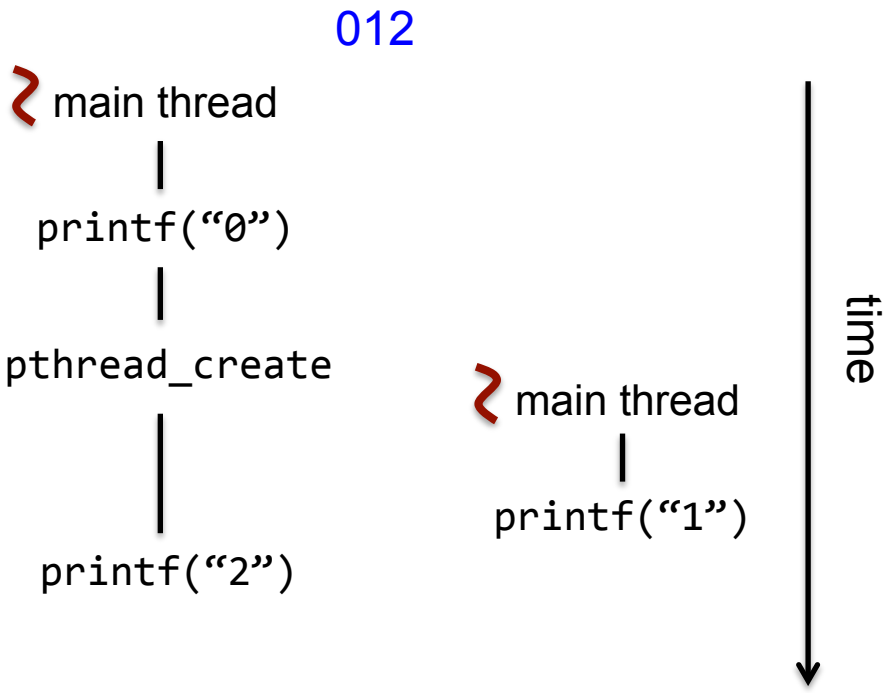
Answer: 012 or 021

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

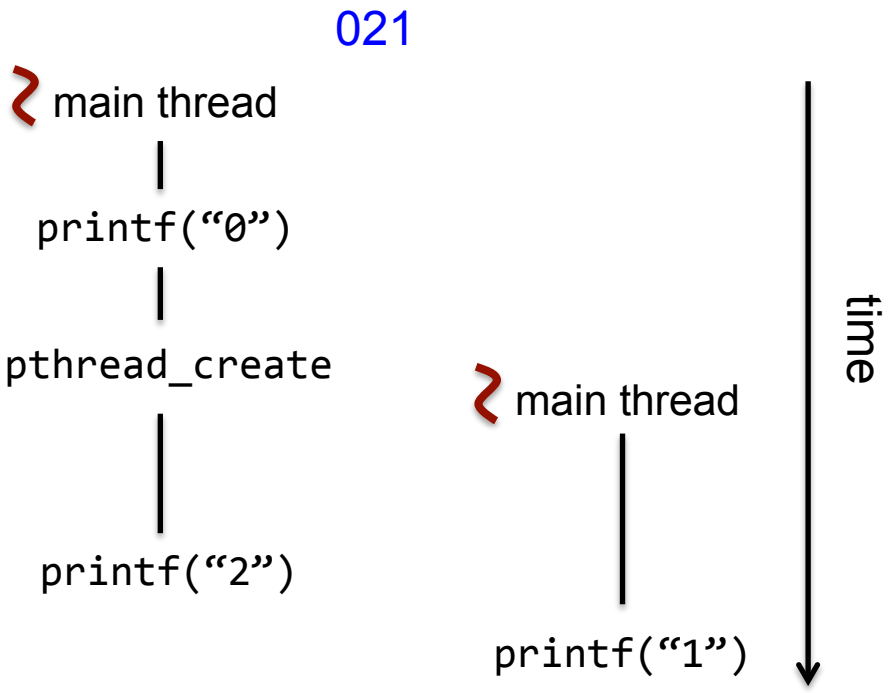


Example 4 – Interleave

```
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

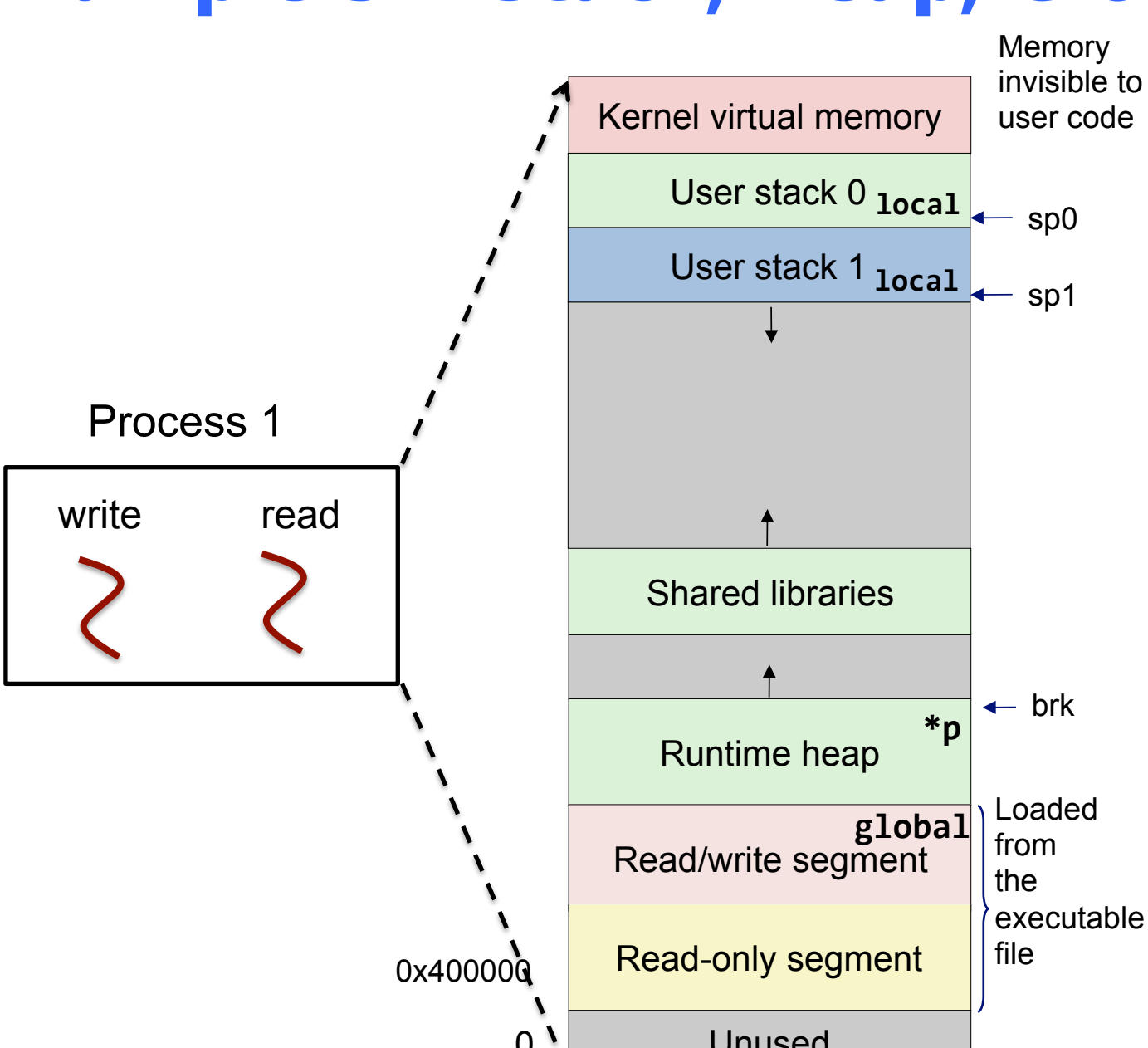


Example 5 – Stack, Heap, Global

```
int global = 0;

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

int main(int argc, char* argv[]) {
  int *p = (int *)malloc(sizeof(int));
  pthread_t tid;
  int r = pthread_create(&tid, NULL, &write, (void *)p);
  ...
  sleep(1)
  int r = pthread_create(&tid, NULL, &read, (void *)p);
  ...
  return 0;
}
```

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

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;
}

int main(int argc, char* argv[]) {
  int *p = (int *)malloc(sizeof(int));
  pthread_t tid;
  int r = pthread_create(&tid, NULL, &write, (void *)p);
  ...
  sleep(1)
  int r = pthread_create(&tid, NULL, &read, (void *)p);
  ...
  return 0;
}
```

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

How can read access `local` in write?

Example 5 – Stack, Heap, Global

```
int global = 0;
int *local_addr = 0;

void* write(void* arg) {
  int local = 0;
  local_addr = &local;
  local = 100;
  global = 100;
  int *ptr = (int *)arg;
  (*ptr) = 100;
  sleep(10);
}

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

int main(int argc, char* argv[]) {
  int *p = (int *)malloc(sizeof(int));
  pthread_t tid;
  int r = pthread_create(&tid, NULL, &write, (void *)p);
  ...
  sleep(1)
  int r = pthread_create(&tid, NULL, &read, (void *)p);
  ...
  return 0;
}
```

Example 3 – Review

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

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

  int param = 100;

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

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

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

Question – can we get rid of r in func?

Example 3 – Review

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

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

  int param = 100;

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

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

  printf("result: %d\n", param);
  return 0;
}
```

Question – can we get rid of r in func?

Example 6 – bigloop

```
#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);
  ...
}
```

Parallelize bigloop into two threads

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;
}

int main() {
  int *arr = malloc(LEN * sizeof(int));
  pthread_t tid1, tid2;
  int r = pthread_create(&tid, NULL, &loop_thr1, (void *)arr);
  ...
  r = pthread_create(&tid, NULL, &loop_thr2, (void *)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;
  int r = pthread_create(&tid, NULL, &loop_thr1, (void *)arr);
  ...
  r = pthread_create(&tid, NULL, &loop_thr2, (void *)arr);
  ...
 }
```

Example 6 – bigloop

```
#define LEN 1000000000

typedef struct {
 int *arr;
 int start;
 int end;
} loop_info;
```

```
void* loop(void *arg){
  long *r = malloc(sizeof(long));
  loop_info *info = (loop_info *)arg;

  for(int i = info->start; i < info->end; i++)
    (*r) += info->arr[i];
  return (void *)r;
}
```

```
int main() {
  int *arr = malloc(LEN * sizeof(int));
  pthread_t tid1, tid2;
  int r = pthread_create(&tid, NULL, &loop, (void *)p);
  ...
  r = pthread_create(&tid, NULL, &loop, (void *)p);
  ...
 }
```

Example 6 – bigloop

```
#define LEN 1000000000

typedef struct {
 int *arr;
 int start;
 int end;
} loop_info;
```

```
void* loop(void *arg){
  long *r = malloc(sizeof(long));
  loop_info *info = (loop_info *)arg;

  for(int i = info->start; i < info->end; i++)
    (*r) += info->arr[i];
  return (void *)r;
}
```

How to create N threads?

N is passed by user and LEN is multiple times of N.

```
int main() {
  int *arr = malloc(LEN * sizeof(int));
  pthread_t tid1, tid2;
  int r = pthread_create(&tid, NULL, &loop, (void *)p);
  ...
  r = pthread_create(&tid, NULL, &loop, (void *)p);
  ...
 }
```