SUSE

11 DECEMBER 2022

Introduction to system/backend programming

System programming

System programming?

- System programming is about **mechanical sympathy**
- It’s about knowing **how and why** stuff work the way they do
- And most importantly – being able to critically think about your program or **system**

Array copy

```
void copyij(long int src[2048][2048], long int dst[2048][2048])
{
  long int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}

void copyji(long int src[2048][2048], long int dst[2048][2048])
{
  long int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

Array copy

```
void copyij(long int src[2048][2048], long int dst[2048][2048])
{
  long int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}

void copyji(long int src[2048][2048], long int dst[2048][2048])
{
  long int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

copyij time spent: 0.016467 s (16ms)
copyji time spent: 0.039325 s (39ms) 2x **slowdown**

Array copy

```
void copyij(long int src[2048][2048], long int dst[2048][2048])
{
  long int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}

void copyji(long int src[2048][2048], long int dst[2048][2048])
{
  long int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

copyij time spent: 0.016467 s (16ms)
copyji time spent: 0.039325 s (39ms) x**2**
slowdown

Row-major

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Column-major

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

0-3	Node 1
4-7	Node 2
8-11	Node 3
12-15	Node 4

System calls

- Computer programs (usually) run within the confines of the OS/kernel

System calls

- Computer programs (usually) run within the confines of the OS
- In order for them to interface with the external world (R/W files, R/W on the network) they need the help of the OS

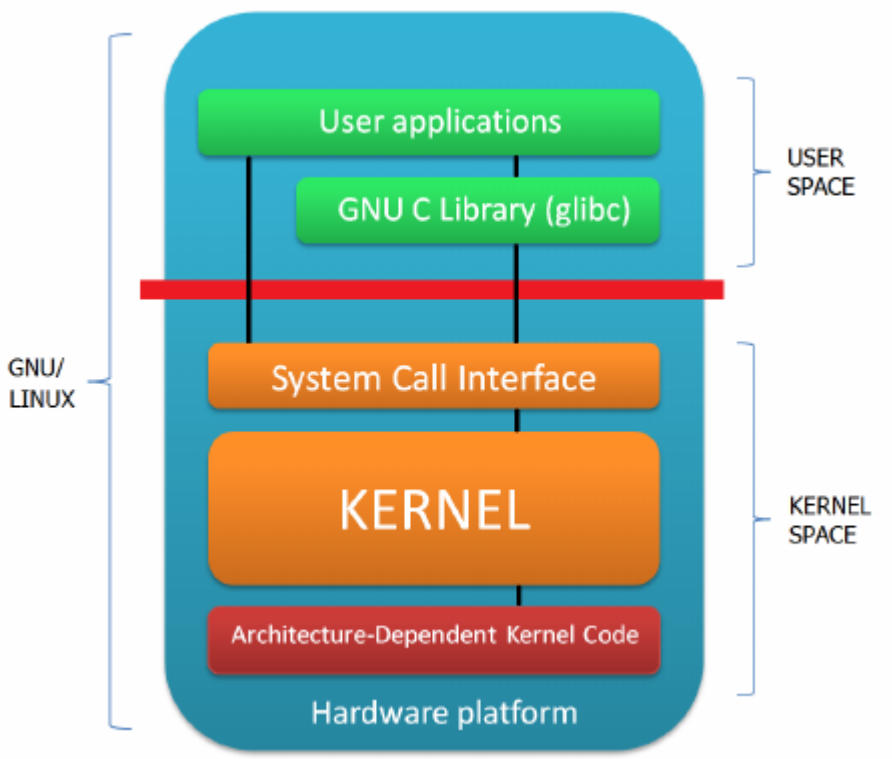
System calls

- Computer programs (usually) run within the confines of the OS
- In order for them to interface with the external world (R/W files, R/W on the network) they need the help of the OS
- This is achieved via System Calls (or syscalls)
  - That’s the interface the operating system (the kernel) gives to programs
  - This allows to give controlled access to the hardware (hdd, network card).

System calls

- Computer programs (usually) run within the confines of the OS
- In order for them to interface with the external world (R/W files, R/W on the network) they need the help of the OS
- This is achieved via System Calls (or syscalls)
  - That’s the interface the operating system (the kernel) gives to programs
  - This allows to give controlled access to the hardware (hdd, network card).
- System calls are generally implemented as an **exception handler** i.e they cause the current **execution context** to jump to the **kernel** and then get routed to the correct function
  - When this happens the kernel performs some action on behalf of the user

System calls



System calls (2)

```
with open("test.txt", "w+") as f:
    f.write("Hello world")
    f.seek(0)
    print(f.read())
```

```
# strace python3 filewrite.py
openat(AT_FDCWD, "test.txt", O_RDWR|O_CREAT|O_TRUNC|
O_CLOEXEC, 0666) = 3
<ommitted for brevity>
write(3, "Hello world", 11)             = 11
lseek(3, 0, SEEK_SET)                   = 0
read(3, "Hello world", 12)              = 11
write(1, "Hello world\n", 12Hello world)   = 12
```

https://man7.org/linux/man-pages/man2/open.2.html
https://man7.org/linux/man-pages/man2/write.2.html
https://man7.org/linux/man-pages/man2/lseek.2.html
https://man7.org/linux/man-pages/man2/read.2.html

Operating System

- We saw the term OS and you've probably heard of Linux/Windows/Mac but...

Operating System

- We saw the term OS and you've probably heard of Linux/Windows/Mac but…
  - Have you asked yourself **what exactly IS the OS**?

Operating System

- We saw the term OS and you've probably heard of Linux/Windows/Mac but…
  - Have you asked yourself **what exactly IS the OS**?
  - Is it the taskbar or the task manager on Windows?
  - Is it the root shell on a Linux system?

Operating System

- We saw the term OS and you've probably heard of Linux/Windows/Mac but…
  - Have you asked yourself **what exactly IS the OS**?
  - Is it the taskbar or the task manager on Windows?
  - Is it the root shell on a Linux system?
- The answer is all **NO**

Operating System

- We saw the term OS and you've probably heard of Linux/Windows/Mac but…
  - Have you asked yourself **what exactly IS the OS**?
  - Is it the taskbar or the task manager on Windows?
  - Is it the root shell on a Linux system?
- The answer is all **NO**
- The OS is really a bunch of **exception handlers** (like the syscall handler) + accompanying code like:
  - Architecture support code
  - The process scheduler
  - The virtual memory subsystem
  - Hardware device drivers
  - Filesystems

Operating System

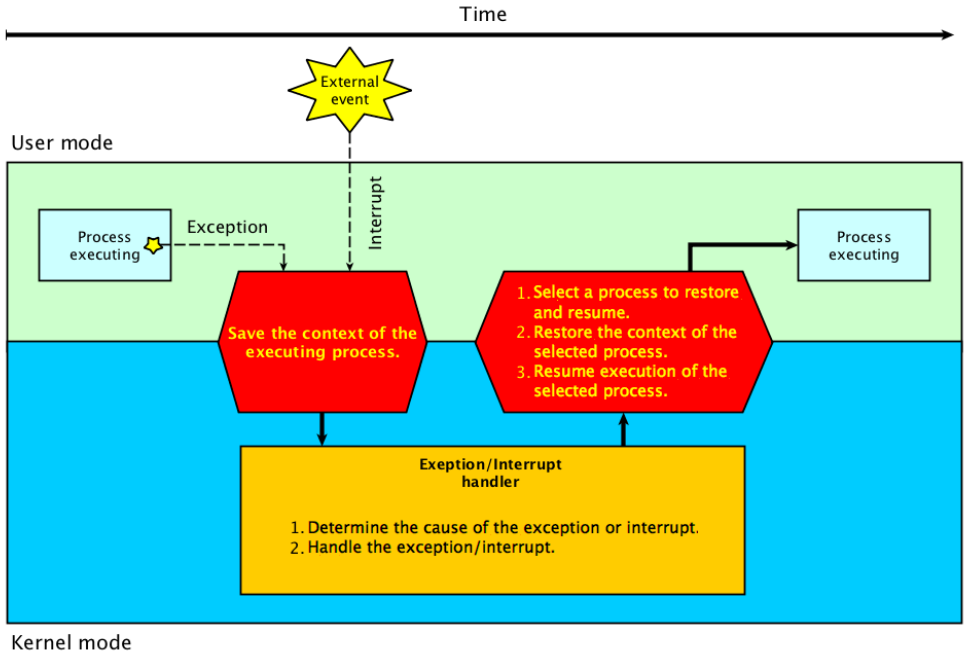
- The OS is always running as part of every ordinary process
  - Thus we conclude each process has a user space portion and kernel (privileged) portion
  - Kernel is not copied, the state is shared I.e currently running process or misc. accounting
  - There’s a constant jump between user space/kernel space via (traps) syscalls and asynchronous exceptions (interrupts)

Operating System (2)

- The OS is always running as part of every ordinary process
  - Thus we conclude each process has a user space portion and kernel (privileged) portion
  - Kernel is not copied, the state is shared I.e currently running process or misc. accounting
  - There’s a constant jump between user space/kernel space via (traps) syscalls and asynchronous exceptions (interrupts)
- Asynchronous Exception (interrupts)
  - Causes current execution to suspend and jump to respective handler, handle the event and return back to the user process.

Operating System (2)

- The OS is always running as part of every ordinary process
  - Thus we conclude each process has a user space portion and kernel (privileged) portion
  - Kernel is not copied, the state is shared I.e currently running process or misc. accounting
  - There’s a constant jump between user space/kernel space via (traps) syscalls and asynchronous exceptions (interrupts)



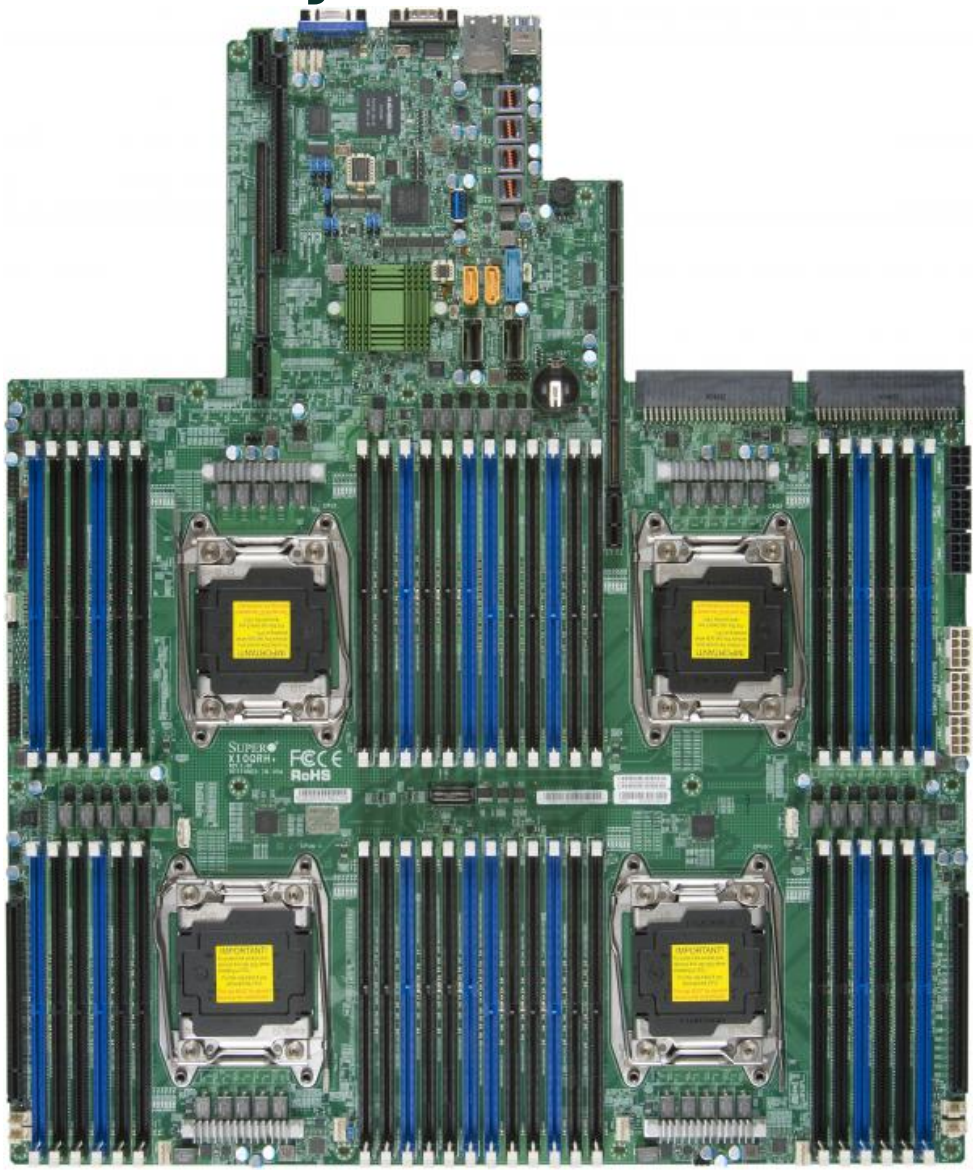
Operating System (2)

- The OS is always running as part of every ordinary process
  - Thus we conclude each process has a user space portion and kernel (privileged) portion
  - Kernel is not copied, the state is shared I.e currently running process or misc. accounting
  - There’s a constant jump between user space/kernel space via (traps) syscalls and asynchronous exceptions (interrupts)
- Asynchronous Exception (interrupts)
  - Causes current execution to suspend and jump to respective handler, handle the event and return back to the user process.
  - Type of interrupts
    - Keyboard key stroke
    - Network packet arrival
    - hdd/ssd byte transferred
    - Time interrupt

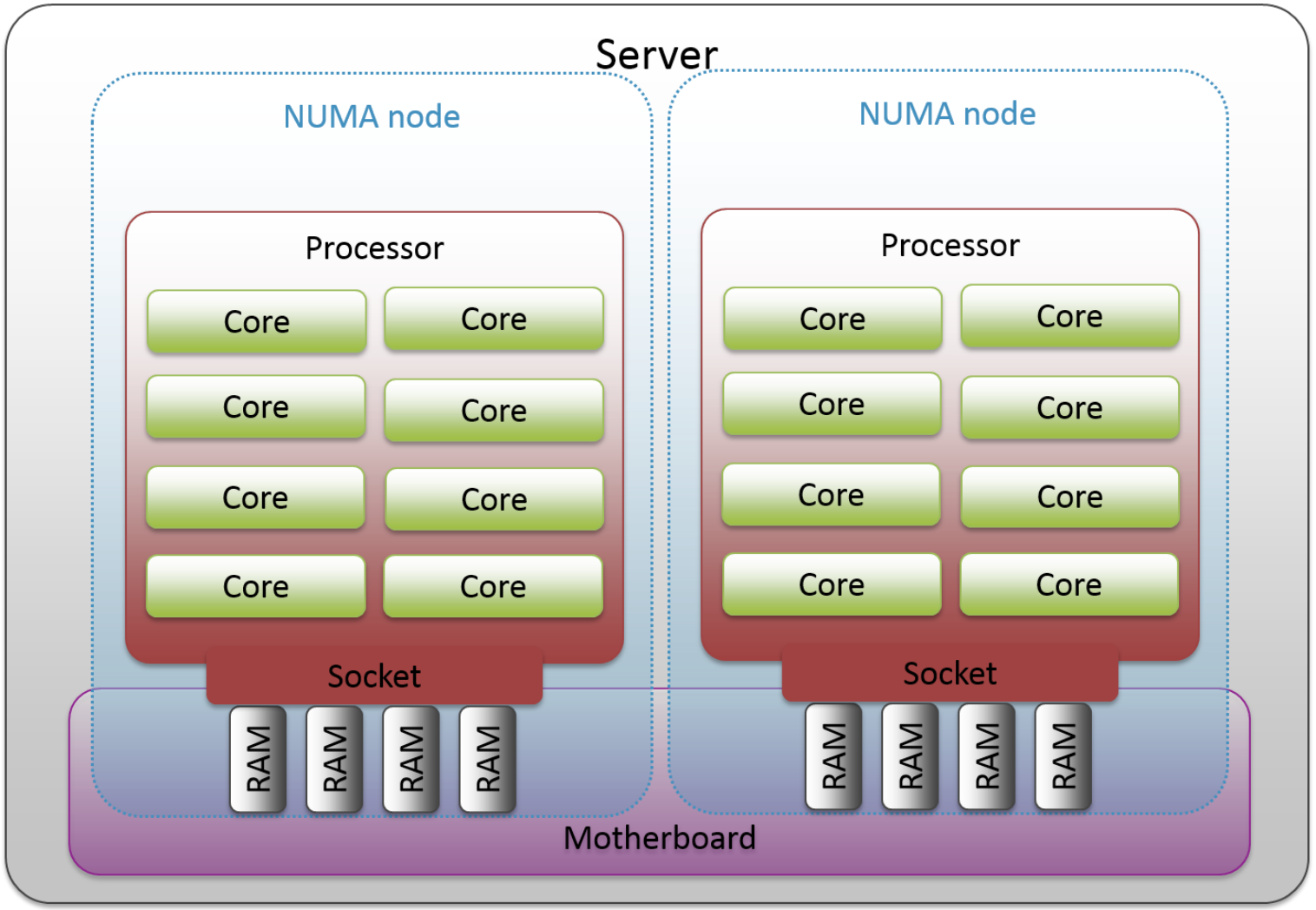
Operating System (2)

- The OS is always running as part of every ordinary process
  - Thus we conclude each process has a user space portion and kernel (privileged) portion
  - Kernel is not copied, the state is shared I.e currently running process or misc. accounting
  - There’s a constant jump between user space/kernel space via (traps) syscalls and asynchronous exceptions (interrupts)
- Asynchronous Exception (interrupts)
  - Causes current execution to suspend and jump to respective handler, handle the event and return back to the user process.
  - Type of interrupts
    - Keyboard key stroke
    - Network packet arrival
    - hdd/ssd byte transferred
    - Time interrupt
- There are 2 other types of exceptions – faults and abort
  - Faults – Unintentional but recoverable e.g. page faults (recoverable), protection fault (unrecoverable).
  - Aborts – similar to unrecoverable faults, causes program to terminate ii.e divide-by-zero, parity error, generally signals some hardware problem in the CPU.

Non-uniform memory access



Non-uniform memory access

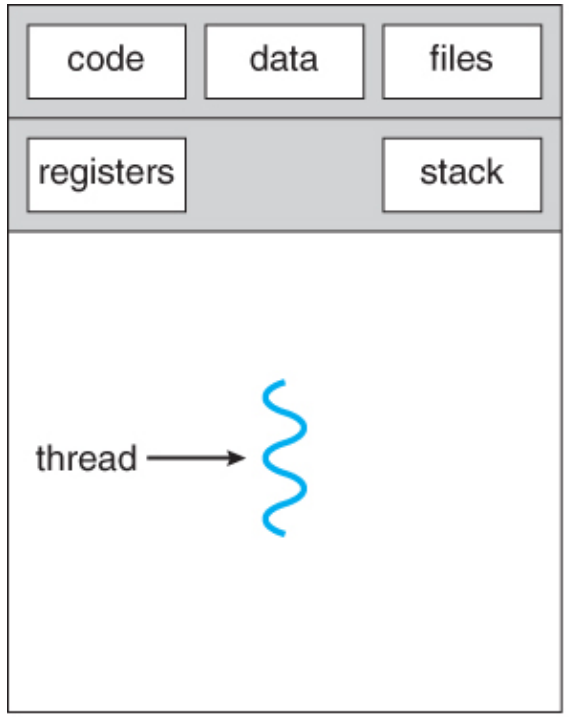


Process execution

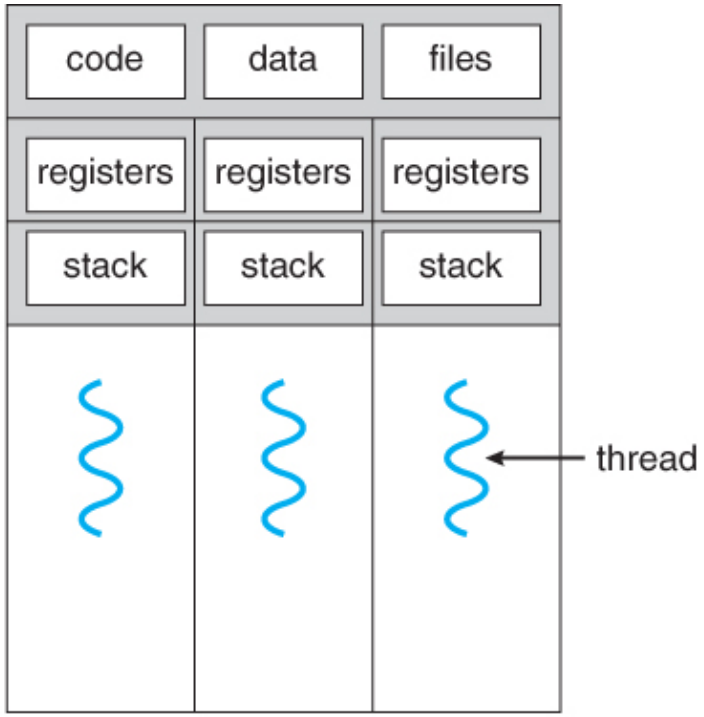
- Process is an instance of a running program
  - It contains all the state necessary for the process to run – opened files, code, data, CPU state (program counter, stack, registers)
  - At least one thread of execution

Process execution

- Process is an instance of a running program
  - It contains all the state necessary for the process to run – opened files, code, data, CPU state (program counter, stack, registers), memory address space
  - At least one thread of execution

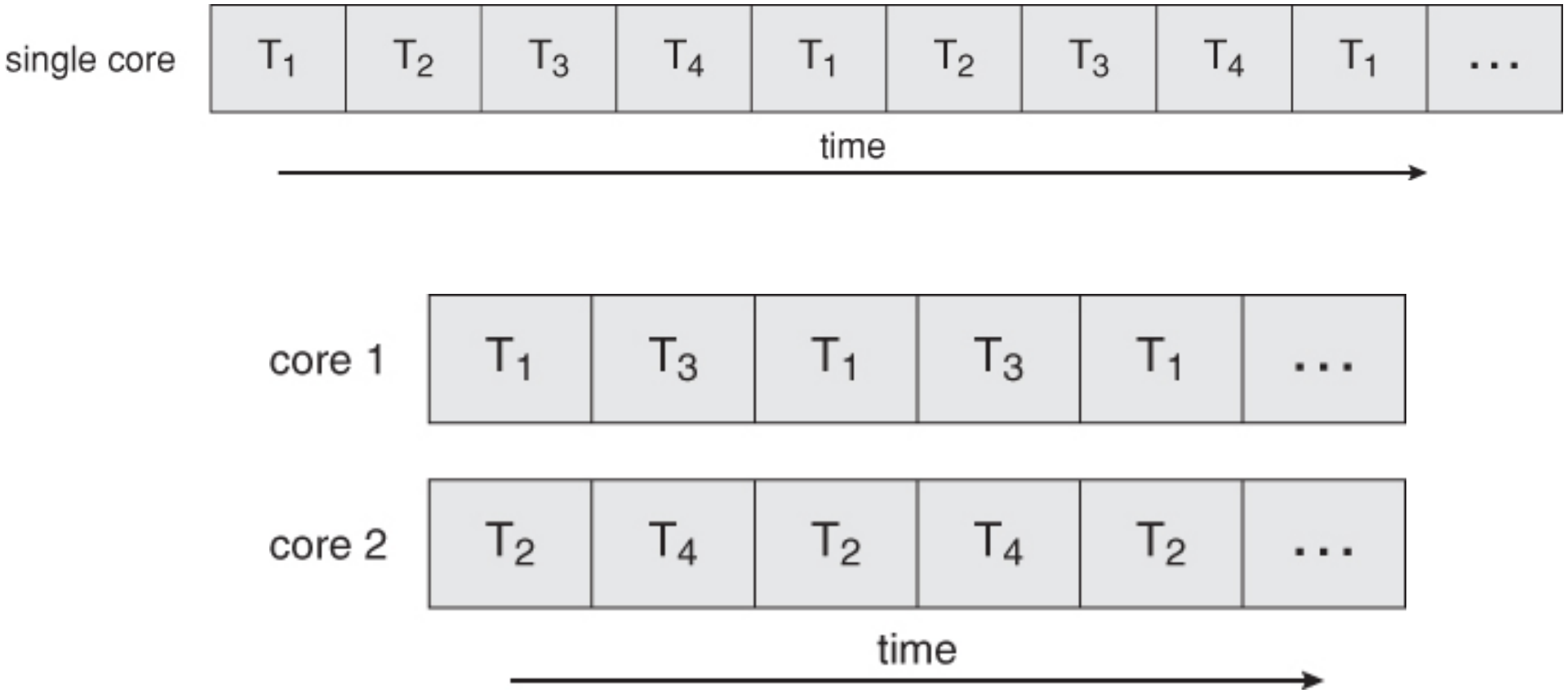


single-threaded process



multithreaded process

Concurrency vs Parallelism



Thank you

For more information, contact SUSE at:

+1 800 796 3700 (U.S./Canada)

Maxfeldstrasse 5

90409 Nuremberg

www.suse.com