Principper for Samtidighed og Styresystemer

Threads and Processes

René Rydhof Hansen

18 FEB 2019

Learning Goals

After today's lecture you

- ... can define and explain the concept of a process
- ... can explain what a process image is
- ... can explain what a process control block, what it is used for and why it is needed
- ... can explain, in general terms, how process creation, switching and termination works
- ... can define and discuss process states
- ... can define and explain the concept of a thread
- ... can discuss when, where, and how (multi-)threads are useful
- ... can explain what a thread control block, what it is used for and why it is needed
- ... can discuss implementation strategies for thread support and explain the associated trade-offs

Processes

What is a process?

Definition (Process)

- The environment a program is executed in
- A running program (with all concomitant data)
- A "virtual machine"
  - ... but with limited access to e.g. the processor and I/O devices
  - Isolated from other processes

Why processes?

- ...

What is an Operating System?

An application manager

- Primary task: to run programs
- Support execution of several programs (sequentially/concurrently)
- Maximising performance

A resource manager

- Primary resource: execution time (CPU)
- External devices: disks, printers, networks, ...
- Support development: provide libraries for common resources
- Support advanced features: sharing, IPC

An abstraction

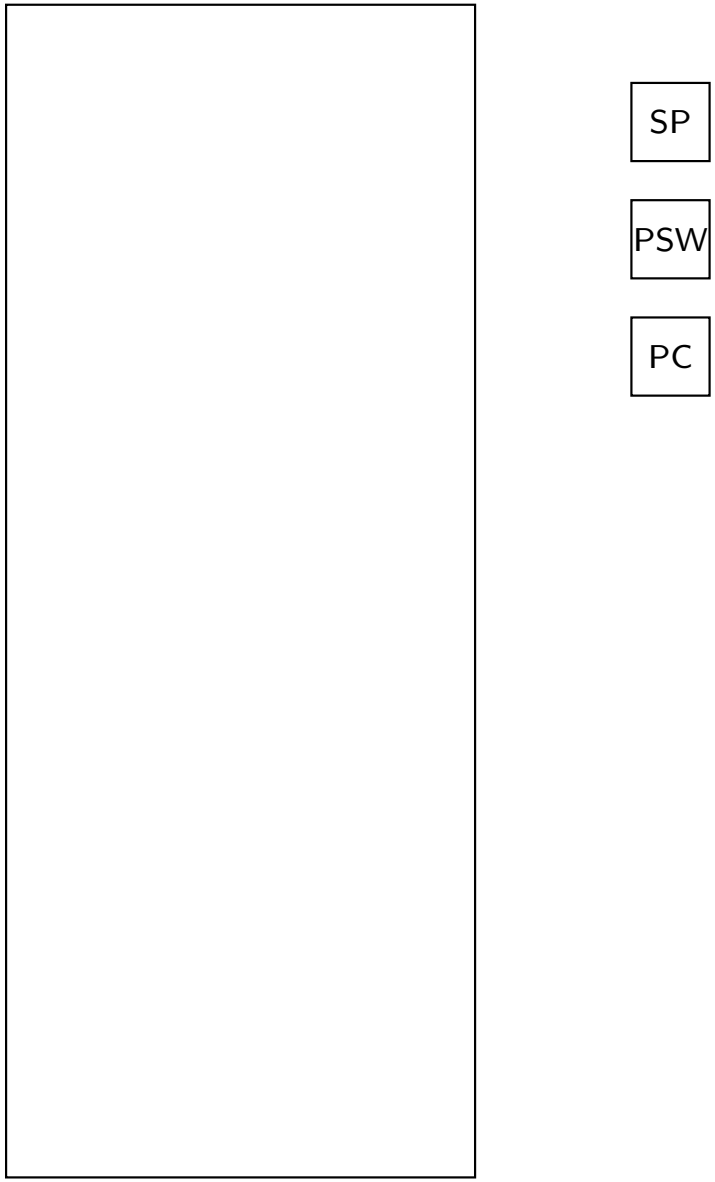
- Every process has its own machine
- Support development: high-level APIs/libraries
- Uniform representation of resources

Running a Program

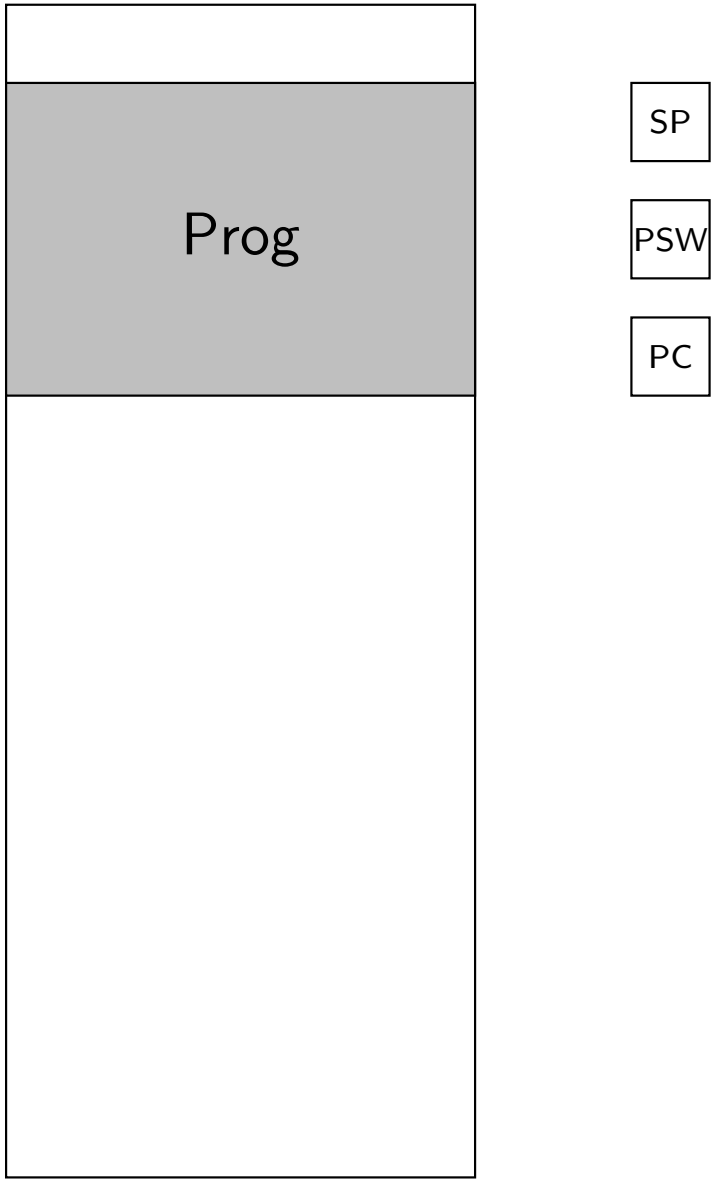
Running a Program (v1)

1. Load code from secondary memory
   - How? ... later lecture!
2. Allocate memory for the process
   - Assume simple fixed memory model (virtual... later!)
   - Code
   - Application memory
     - Heap
     - Stack (SP)
3. Start executing
   - PC points to next instruction
4. Continue until program terminates

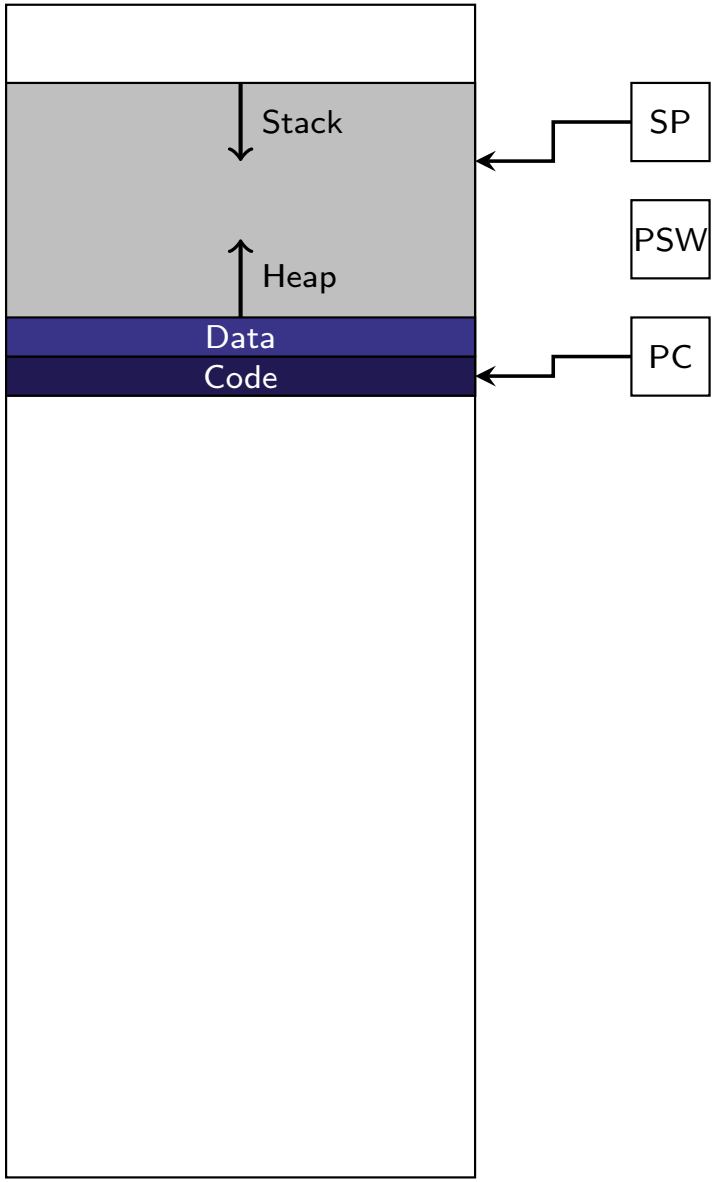
A Simple OS



A Simple OS



A Simple OS



A small problem

What happens at I/O?

- Wait for I/O to finish

A small problem

What happens at I/O?

- Wait a loooong time for I/O to finish
- Waste of time: CPU mostly idle
- Solution?

A small problem

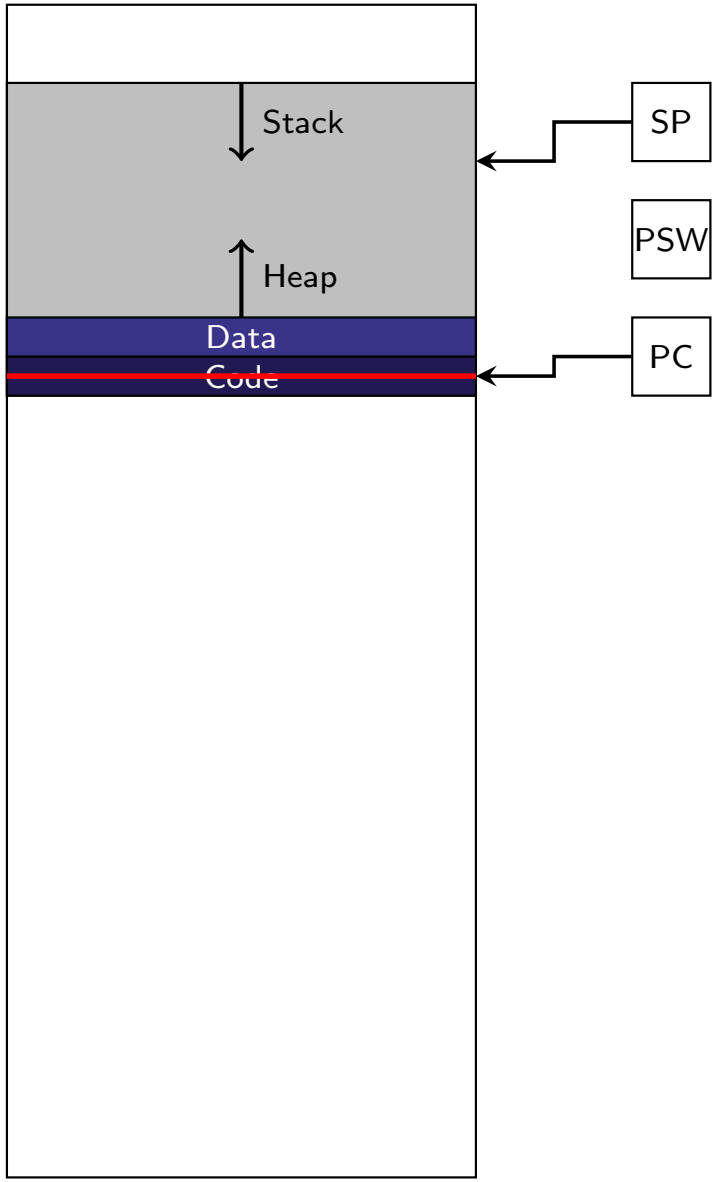
What happens at I/O?

- Wait a loooong time for I/O to finish
- Waste of time: CPU mostly idle
- Solution... switch to another program!

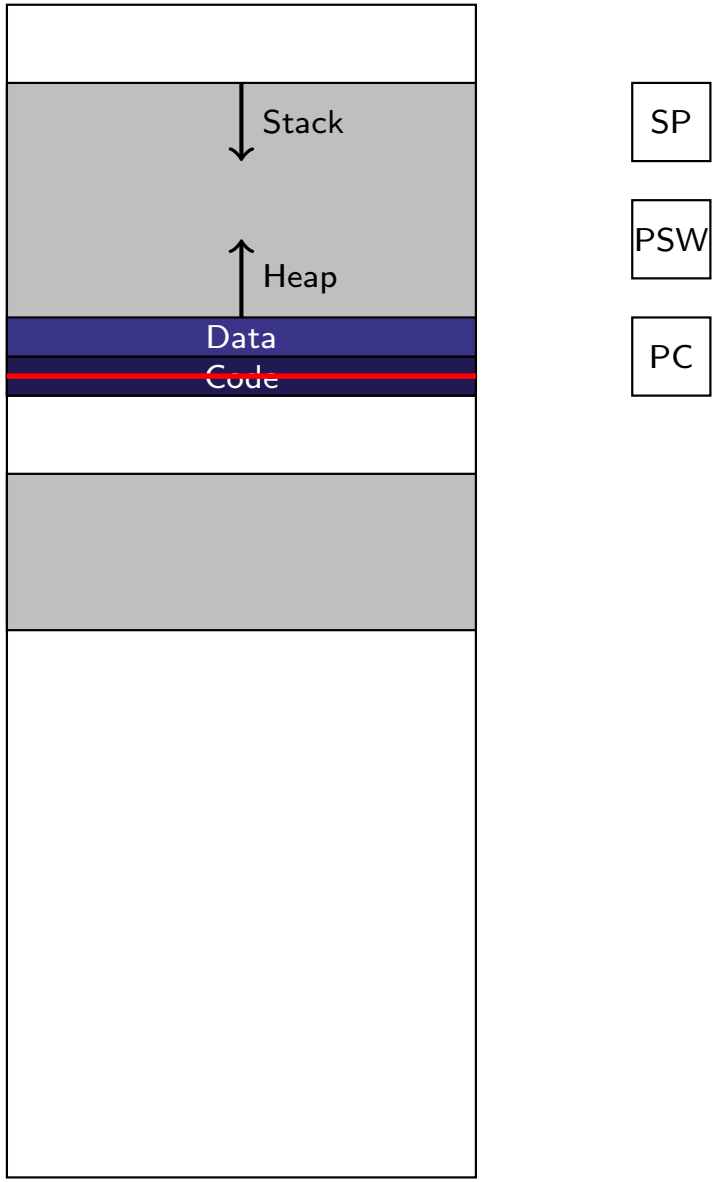
Running a Program (v2)

1. Load code from secondary memory
2. Allocate memory for the process
3. Start executing
4. Continue until program hits an I/O operation
5. Switch to another program

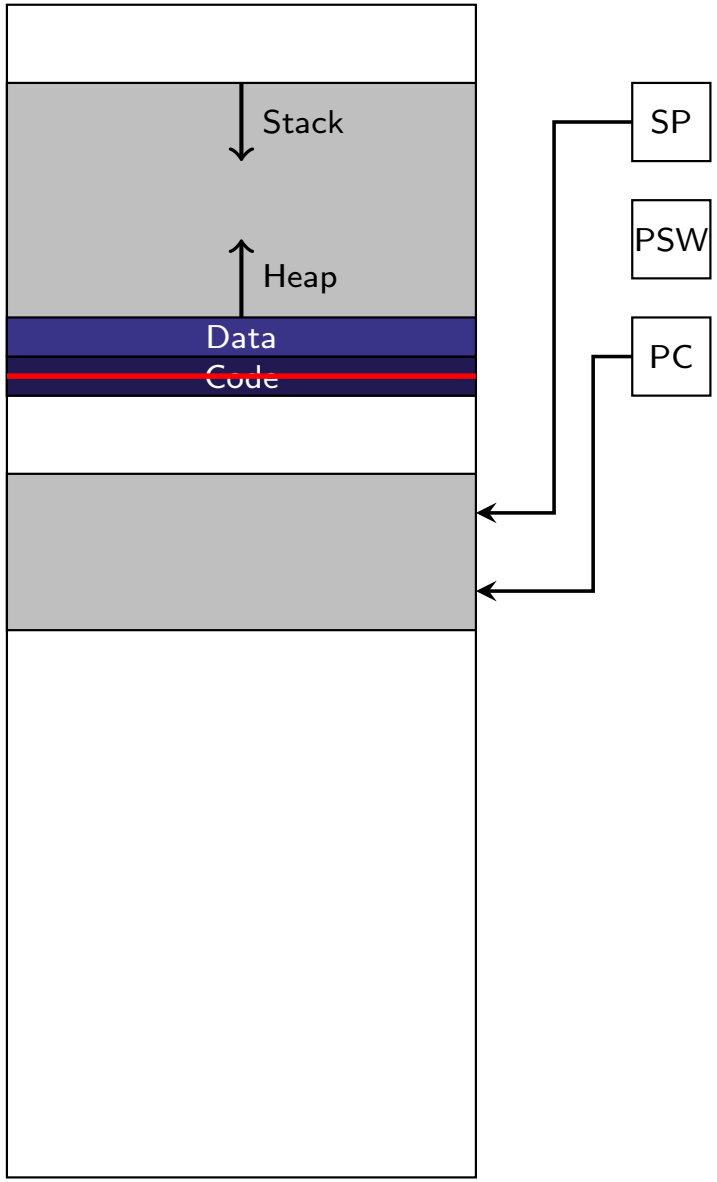
A Simple OS



A Simple OS



A Simple OS



The Lifecycle of a Process

When are processes created?

- Upon login
- Spawned by existing process
- New job started (by OS, user, ...)

When are processes terminated?

- Program completed
- Lack of resources
  - time (hard maximum limit)
  - memory
  - ...
- Error condition
  - invalid instruction
  - arithmetic error
  - ...
- Parent request/parent termination

How to switch back?

Easy!

- Save the relevant data for current process
- Re-instate relevant data for old process
- Problem?

Where to save this data?

- Reserved memory for each process: Process Control Block
- OS maintains process table linking PCBs

How to switch back?

Easy!

- Save the relevant data for current process
- Re-instate relevant data for old process
- Problem... relevant data?

Where to save this data?

- Reserved memory for each process: Process Control Block
- OS maintains process table linking PCBs

How to switch back?

Easy!

- Save the relevant data for current process
- Re-instate relevant data for old process
- Problem... relevant data?

Relevant Data

- Status registers: PC, SP, PSW, ...
- Data registers: EAX, ...
- Memory pointers (code, data, ...)
- I/O status

Where to save this data?

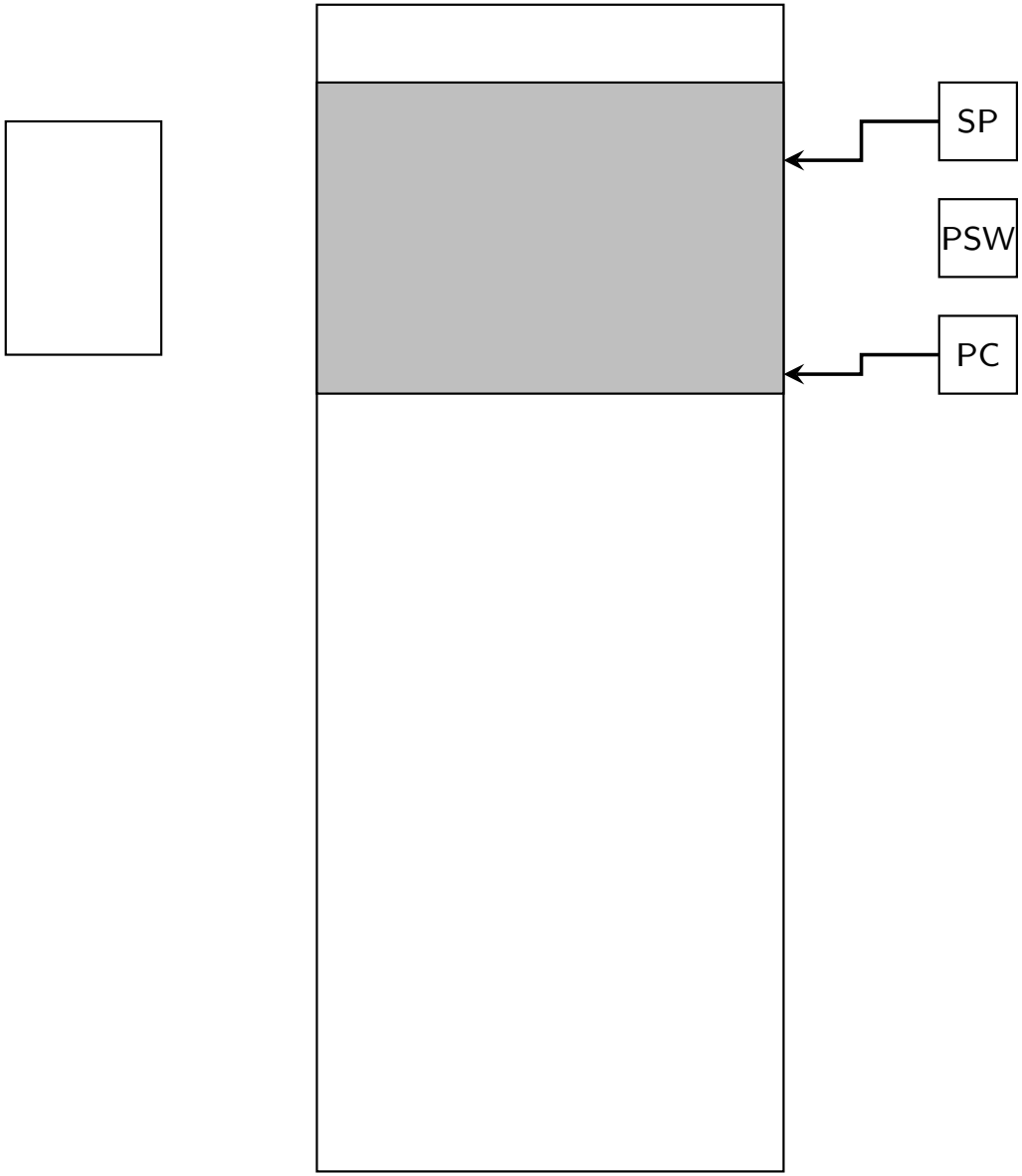
- Reserved memory for each process: Process Control Block
- OS maintains process table linking PCBs

Managing processes

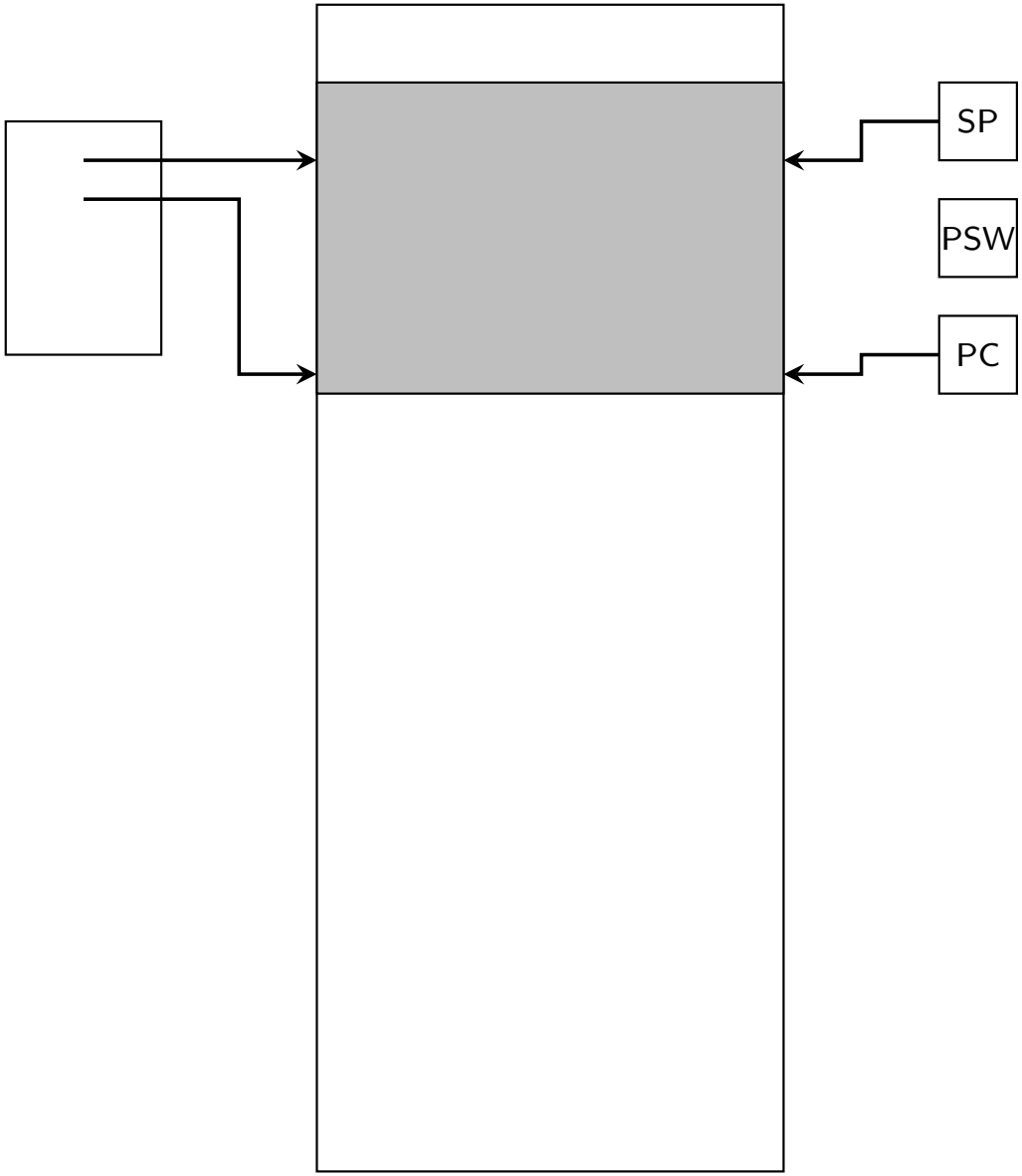
Process Control Block (v1)

- Program counter
- Memory pointers
- Context data (registers)
- I/O status

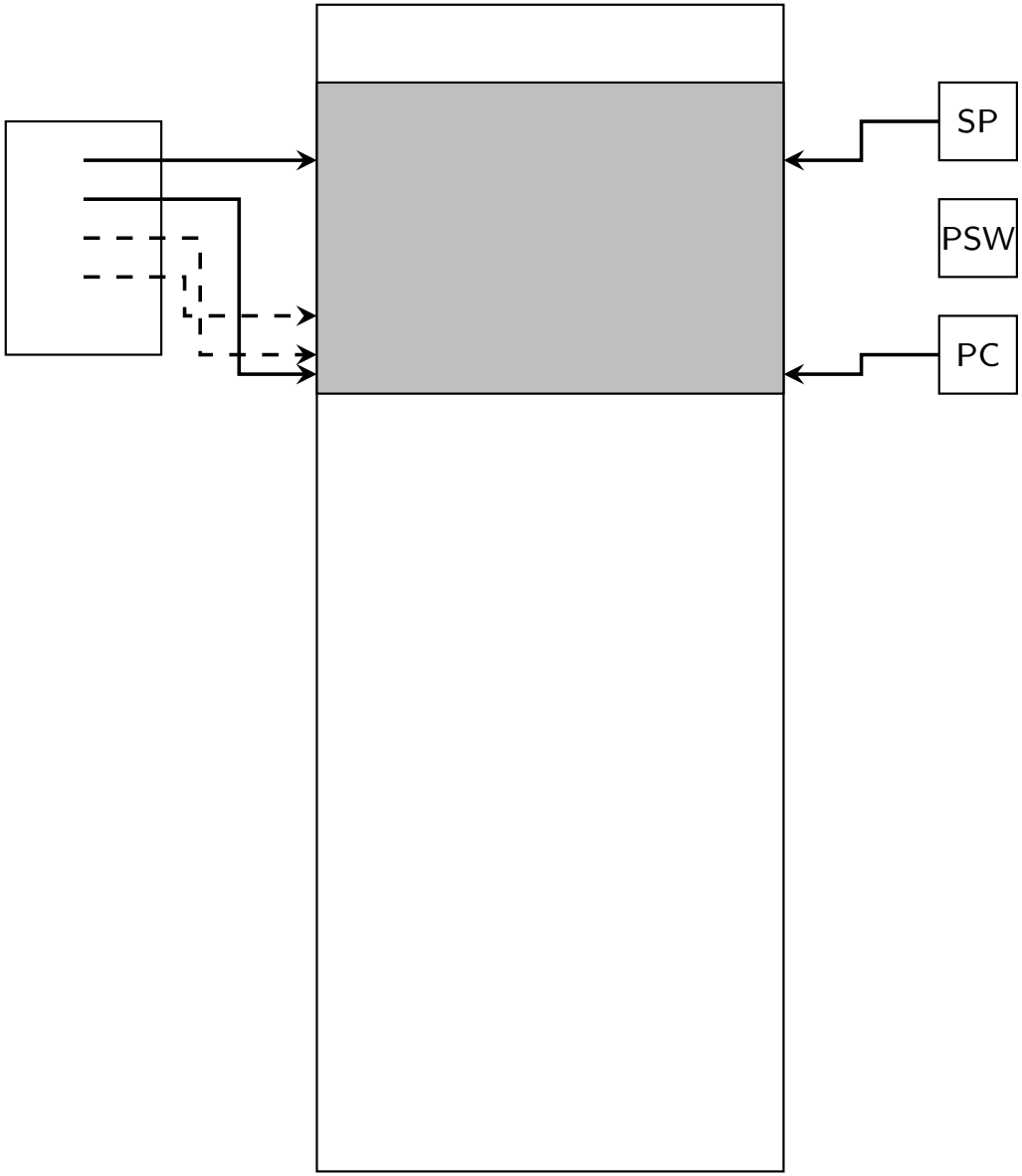
A Simple OS



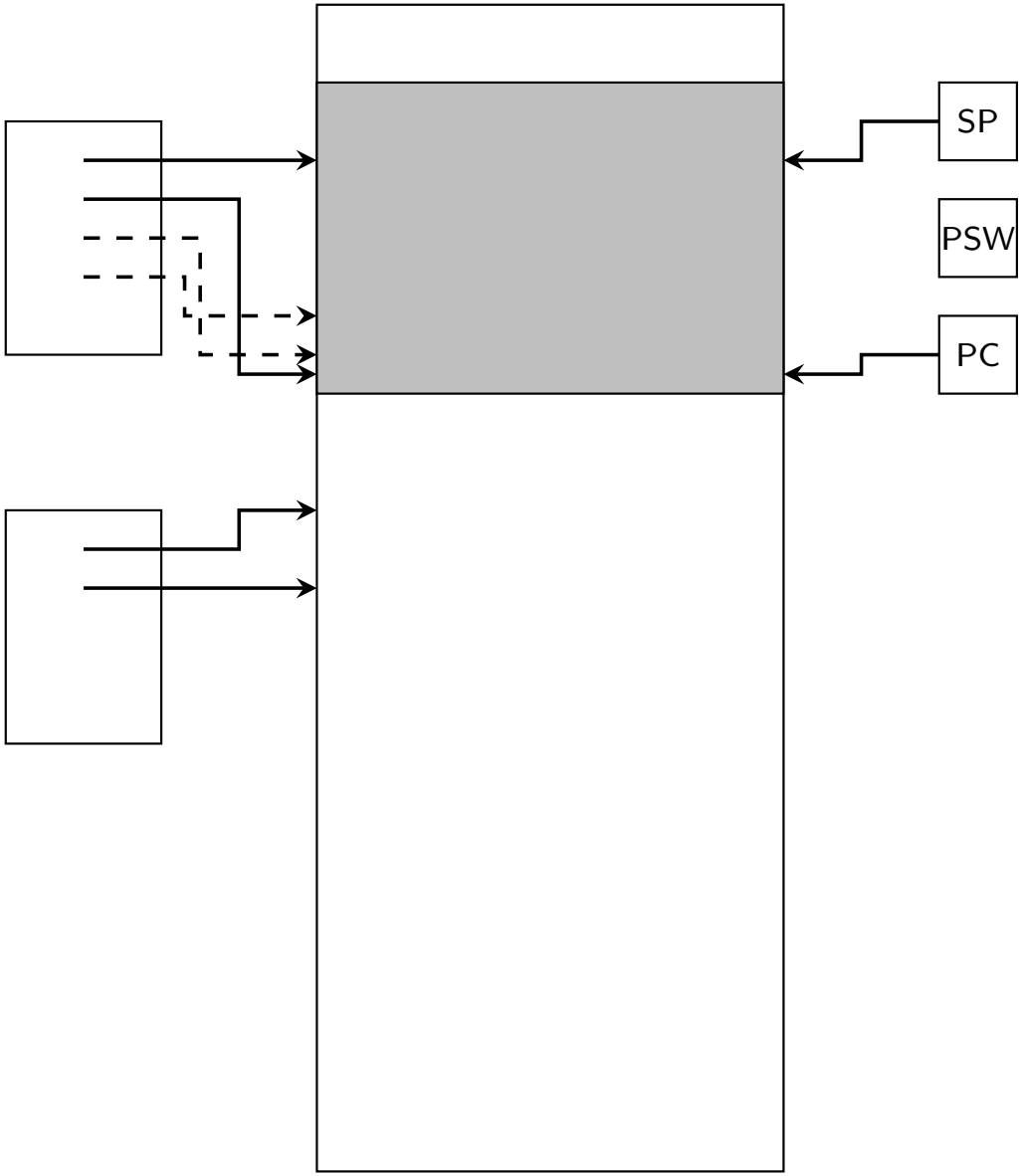
A Simple OS



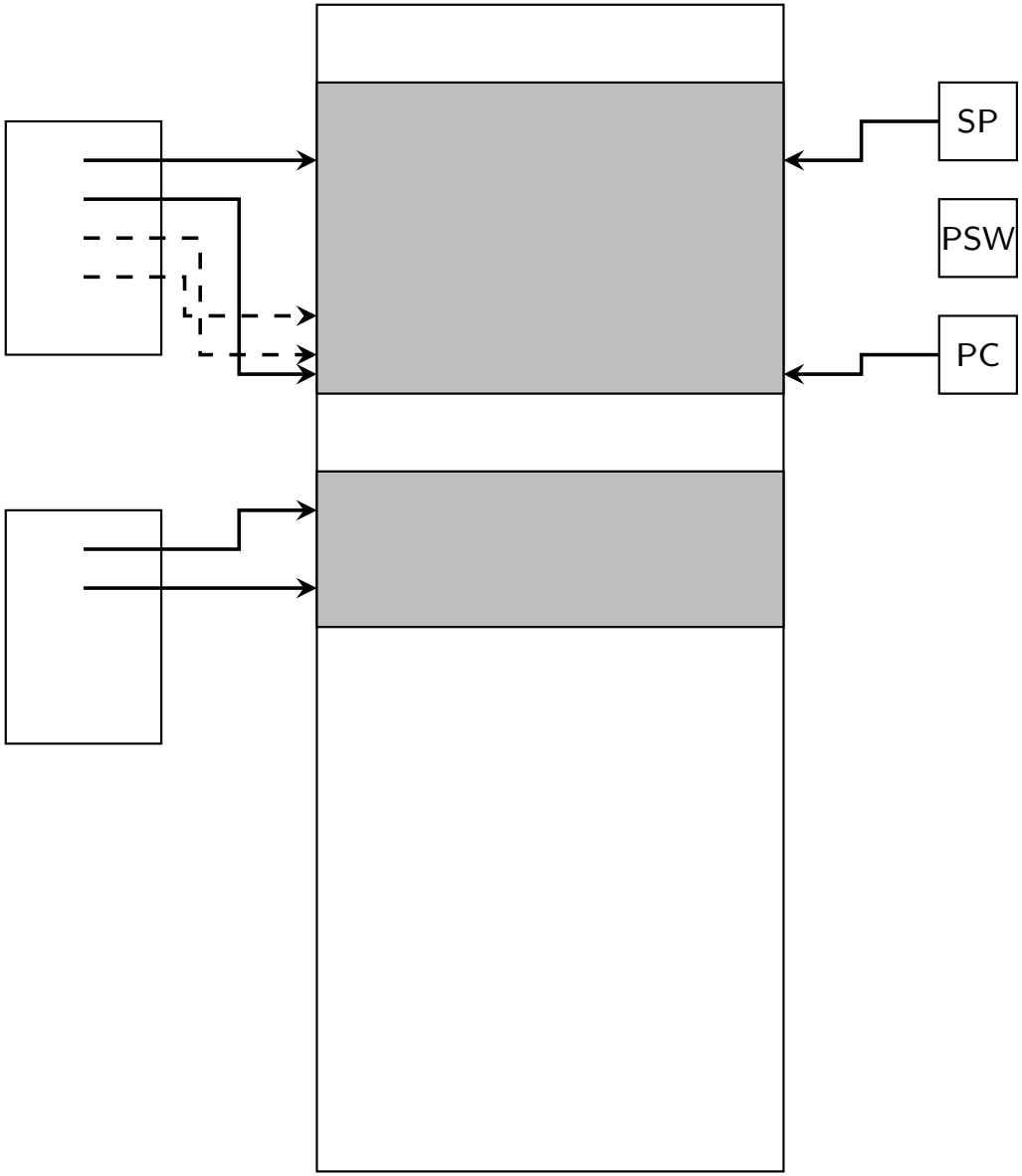
A Simple OS



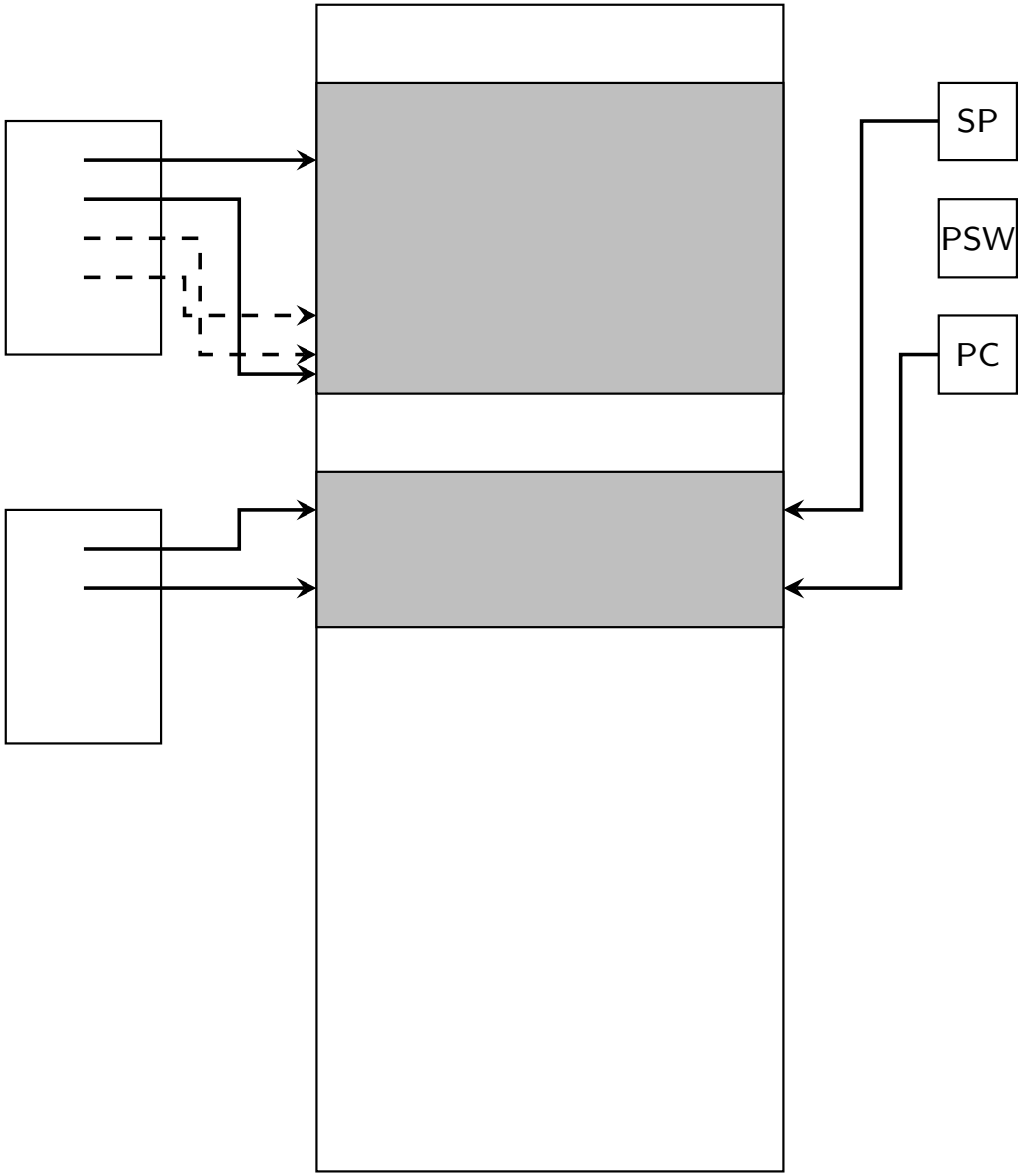
A Simple OS



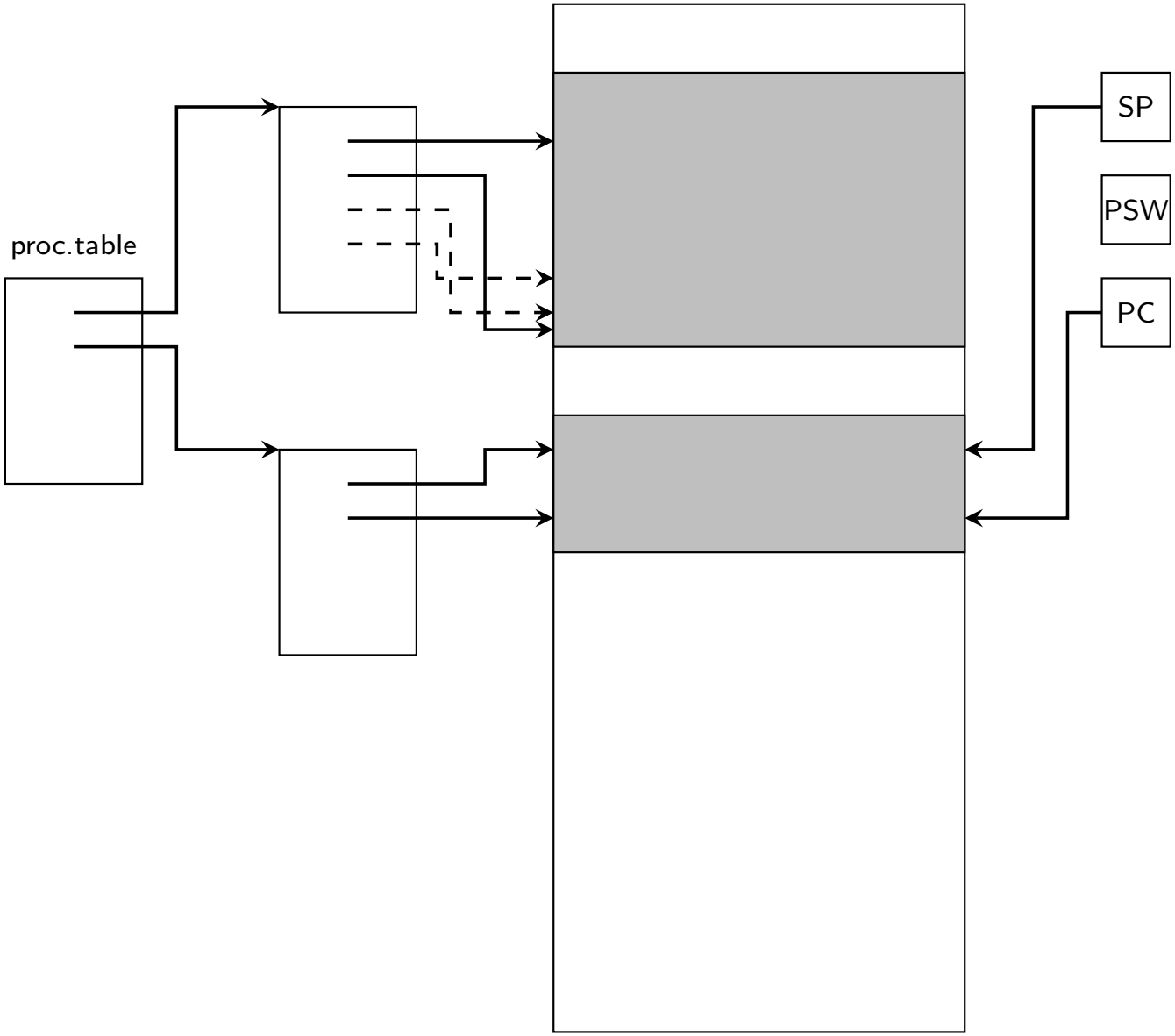
A Simple OS



A Simple OS



A Simple OS



Running a Program (v3)

1. Load code from secondary memory
2. Allocate memory for the process
3. Start executing
4. Continue until program hits an I/O operation
5. Switch to another program
   - Save the relevant data for current program
   - Pick another program
   - Re-instate new program's data
   - Execute

When to switch back?

- When the I/O operation has finished

When to switch back?

- When the I/O operation has finished
- ... and "the new" process has hit an I/O operation!
- In general: when OS regains control

When does the OS regain control?

- Interrupts
  - Clock interrupt
  - I/O interrupt
  - Memory fault
- Errors (traps)
- System calls

Important questions

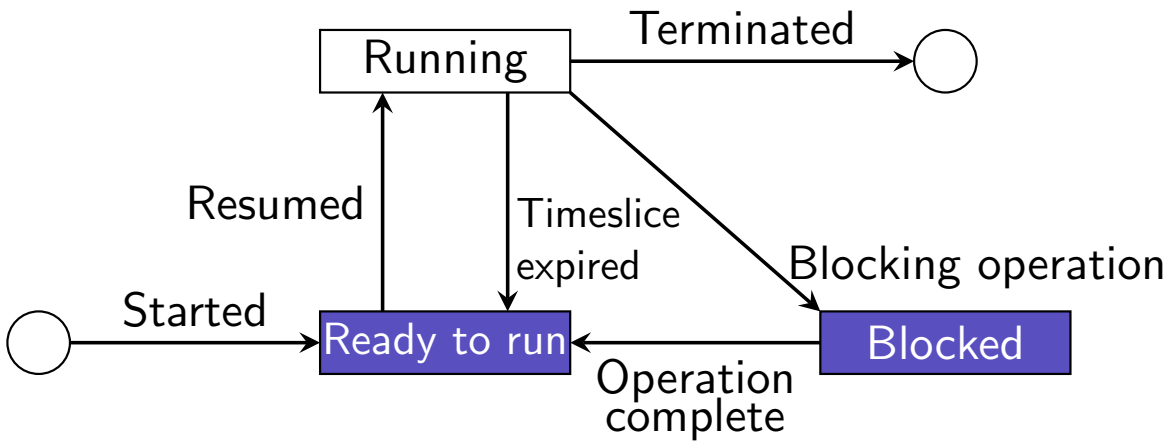
- When does the OS execute?
- Is there an OS process?

How to choose the next process?

- Need to identify processes that are not waiting
  - Need to track the state of a program
  - Possible states: waiting, ready?

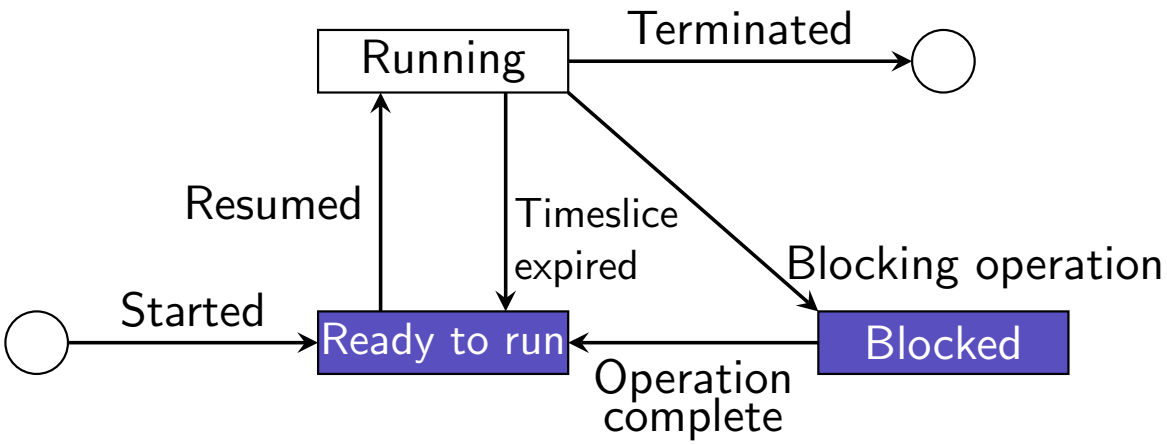
How to choose the next process?

- Need to identify processes that are not waiting
  - Need to track the state of a program
  - Possible states: waiting, ready?



How to choose the next process?

- Need to identify processes that are not waiting
  - Need to track the state of a program
  - Possible states: waiting, ready?
- Pick program from those that are ready
  - Which one?
  - Are some programs more important than others?
  - Need to track priority of a program
  - Enter PCBs into priority queue
  - Scheduling... more later!



Managing processes

Process Control Block (v2)

- Identifier
- State
- Priority
- Program counter
- Memory pointers
- Context data (registers)
- I/O status

Running a Program (v4)

1. Load code from secondary memory
2. Allocate memory for the process
3. Start executing
4. Continue until
    1. Program executes blocking operation
    2. Timeslice expires
5. Switch to another program
    1. Save the relevant data for current program
    2. Pick highest-priority program, ready to run
    3. Re-instate new program's data
    4. Execute

What is a process?

Definition (Process)

- The environment a program is executed in
- A running program (with all concomitant data)
- A "virtual machine"
  - ... but with limited access to e.g. the processor and I/O devices
  - Isolated from other processes

Process Image

Collection of process related data:

- Process Control Block
- Program (code)
- Stack
- Heap (user data)

What is a process?

Definition (Process (PSS))

- A running program (with all concomitant data)

Process Image

Collection of process related data:

- Process Control Block
- Program (code)
- Stack
- Heap (user data)

Threads

What is a process?

Definition (Process)

A running program (with all concomitant data)

Why processes?

- Unit of (resource) control
  - Devices (I/O, CPU), files (I/O), memory, locks, IPC, ...
- Unit of execution
  - Scheduling

Independent concerns?

What is a process?

Definition (Process)

A running program (with all concomitant data)

Why processes?

- Unit of (resource) control
  - Devices (I/O, CPU), files (I/O), memory, locks, IPC, ...
- Unit of execution
  - Scheduling

Independent concerns? ... to some degree: process as context for a thread

Definition (Thread)

The executable part of a process

Multithreading and Multiprogramming

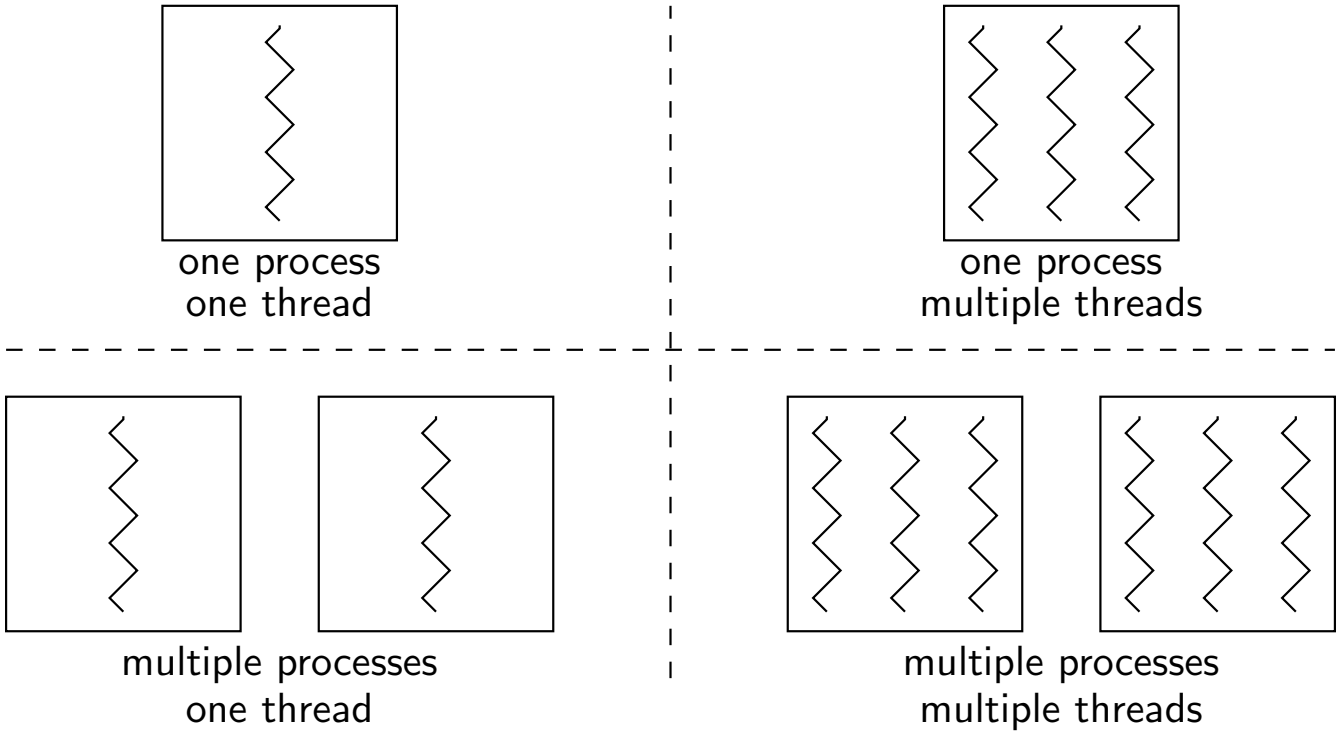
Definition (Multiprogramming)

- Split of tasks that can be executed concurrently (in parallel?) on shared processors
- Implemented by rapid switching between tasks
- Common in modern OSs

Definition (Multithreading)

- Split of processes into several threads
- Always at least one thread pr. process (even in non-multithreaded systems)

Multithreading and Multiprogramming



Why threads?

Performance (perceived)

- Low-cost thread creation, switching, and termination
- Efficient inter-thread communication (shared memory)
- Can utilise multi-processor/multi-core platforms

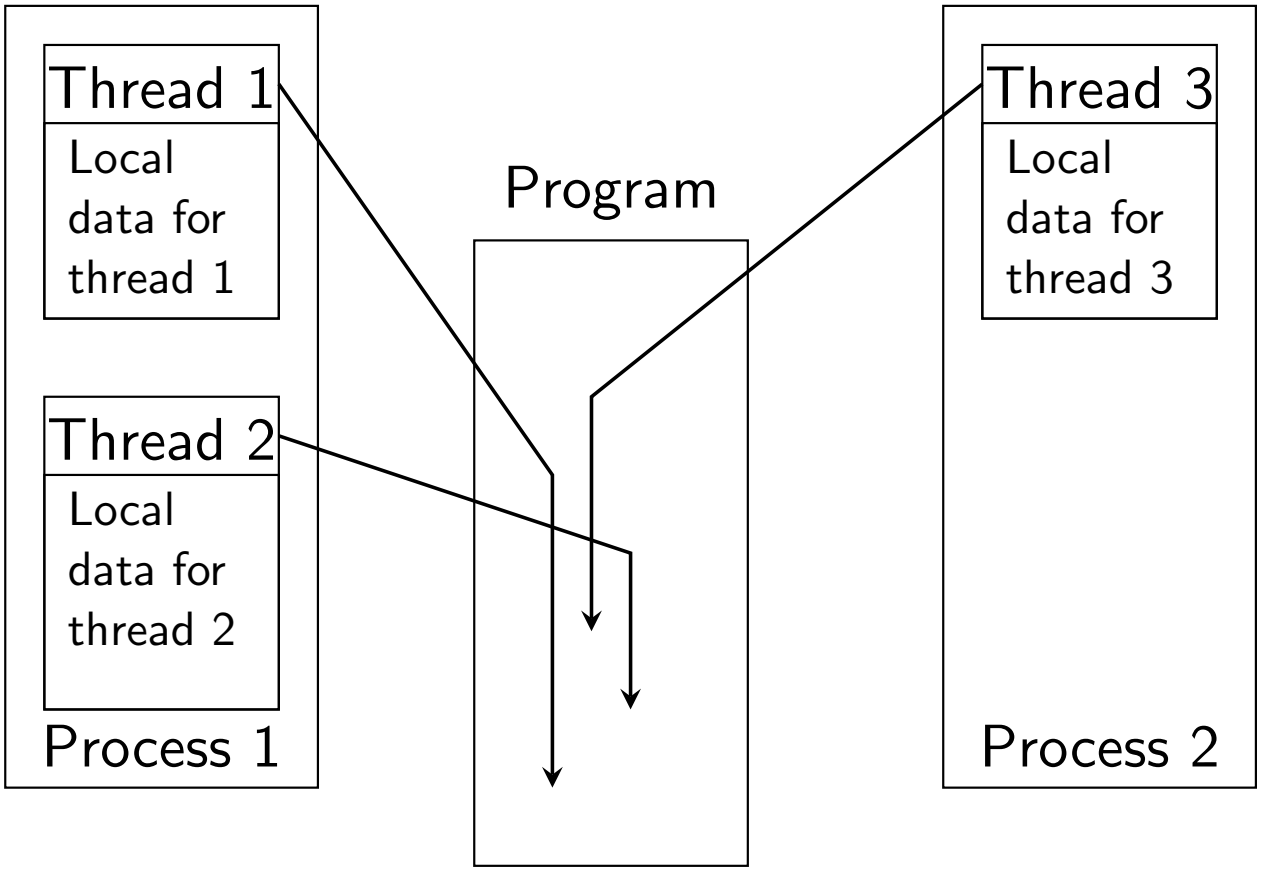
Better architecture/design

- Easy split into foreground and background work
- Asynchronous processing
- Modularity

Example (Responsive GUI)

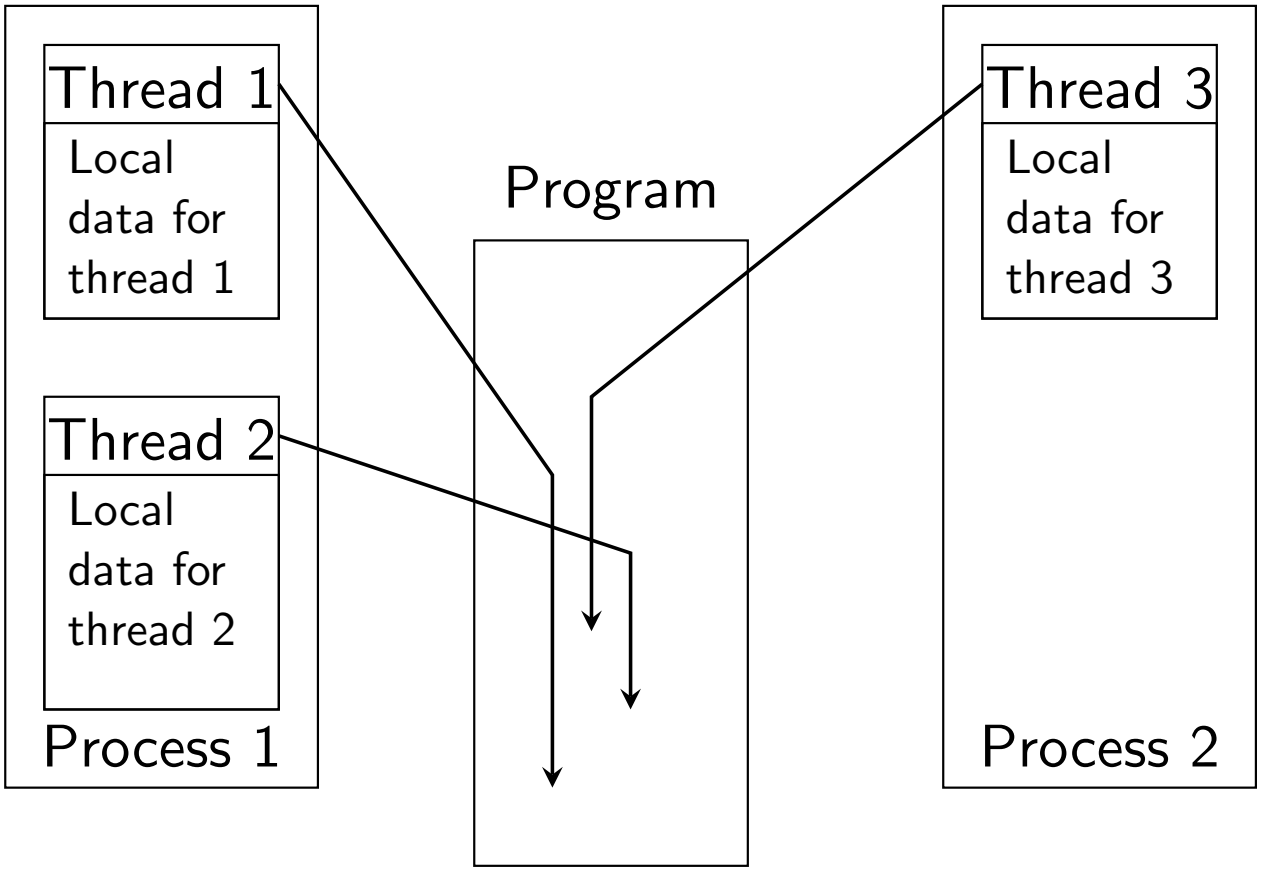
- Some threads tasked with "main" job
- Some threads tasked with updating the GUI; better (perceived) responsiveness
- Better/easier design and implementation

Thread execution



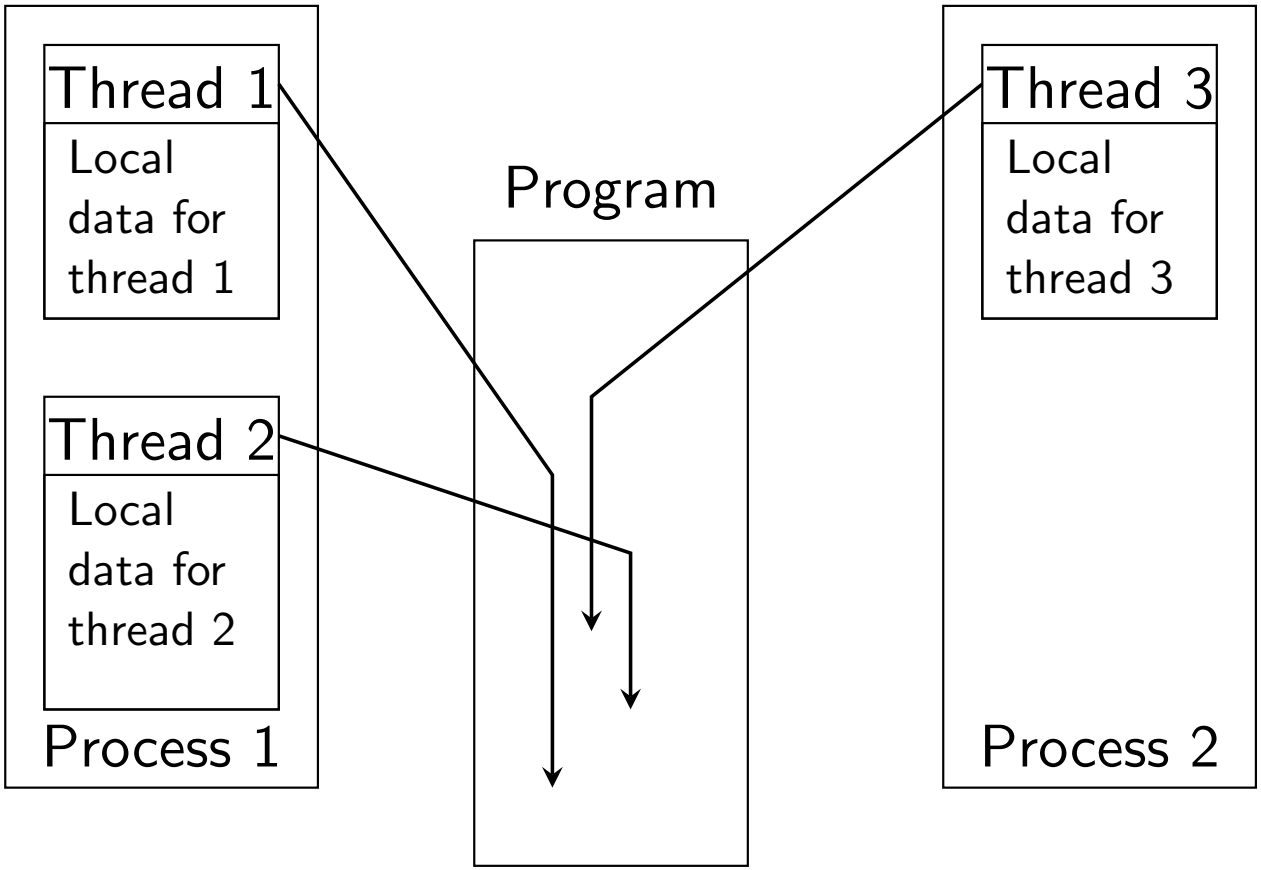
- Two processes executing the same program(!)
- Program-text in shared memory (memory mapping)

Thread execution



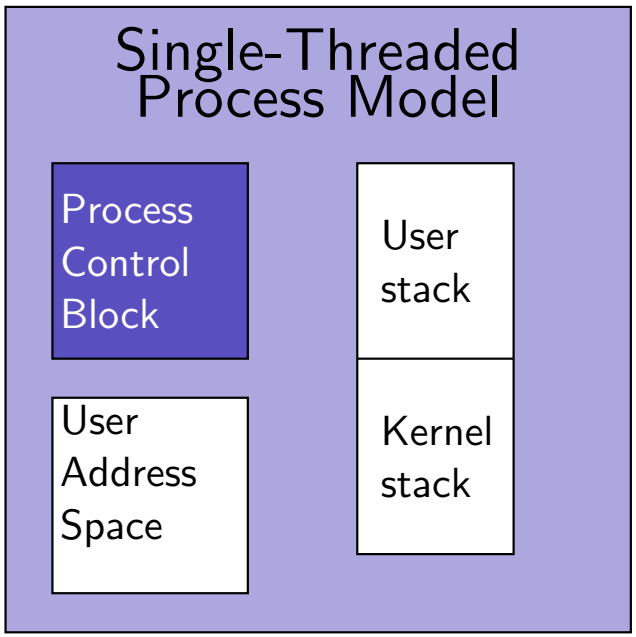
- Thread-local data?
- Process-local data?
- Shared data?

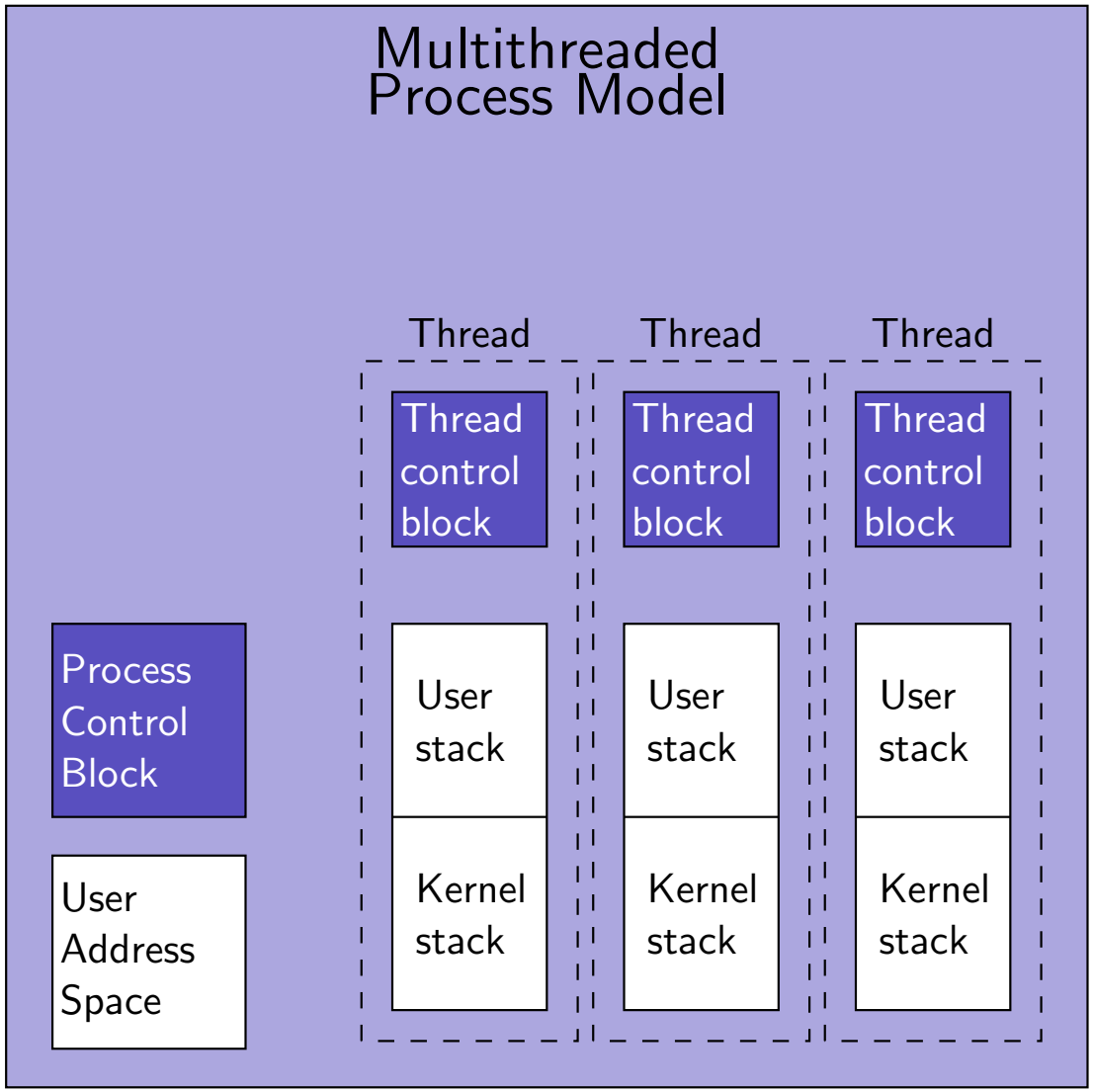
Thread execution



- Thread-local data? thread-ID, priority, stack
- Process-local data? address space, heap, open files, process-ID, parent, ownership, CPU reg. (copy)
- Shared data? Program-text

Process Control Block and Thread Control Block





Threads: Implementation Strategies

Implement in kernel-space (aka. obvious choice)

Implement in user-space

Hybrid: Kernel-support for user-space threads (aka. complex choice)

Threads: Implementation Strategies

Implement in kernel-space (aka. obvious choice)

- Modify/extend kernel to support threads
- Pro's: better handling of blocked threads

Implement in user-space

- User-space library for thread creation, switching, termination, ...
- Pro's: performance (kernel not involved), portable
- Con's: blocked threads

Hybrid: Kernel-support for user-space threads (aka. complex choice)

- Mainly in user-space
- Mapping user-space threads to (fewer) kernel-space threads
- Pro's: all of the above
- Con's: complex implementation, complex management

Sidetrack: Speedup of Multicore and Multithreading

Amdahl's Law: Exploiting Parallelism

Sidetrack: Speedup of Multicore and Multithreading

Amdahl's Law: Exploiting Parallelism

The potential performance speedup

$$\text{Speedup} = \frac{1}{(1-f) + \frac{f}{N}}$$

where

- f is the "parallelisable fraction"
- N is the number of available cores/CPUs

Example

Assuming no overhead:
Speedup for $N = 8$ at $f = 1.0$: 8.0
Speedup for $N = 8$ at $f = 0.9$: 4.7 (with overhead: < 2)

Learning Goals

After today's lecture you

- ... can define and explain the concept of a process
- ... can explain what a process image is
- ... can explain what a process control block, what it is used for and why it is needed
- ... can explain, in general terms, how process creation, switching and termination works
- ... can define and discuss process states
- ... can define and explain the concept of a thread
- ... can discuss when, where, and how (multi-)threads are useful
- ... can explain what a thread control block, what it is used for and why it is needed
- ... can discuss implementation strategies for thread support and explain the associated trade-offs

Appendix: Process Control Block

- Identifier
- State
- Priority
- Program counter
- Memory pointers
- Context data (registers)
- I/O status
- Accounting info.