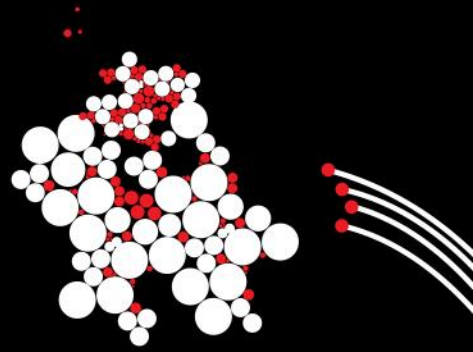
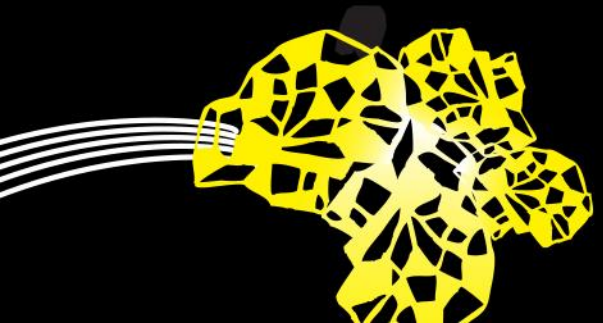
UNIVERSITY OF TWENTE.



OPERATING SYSTEMS

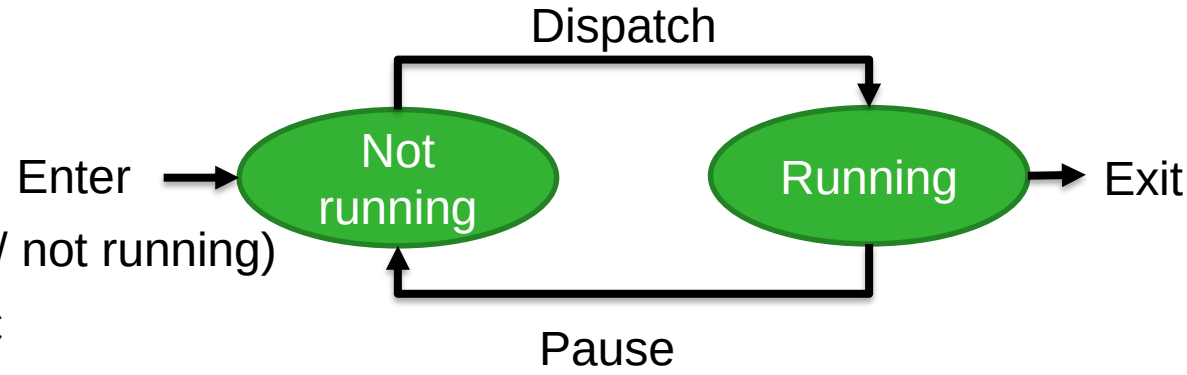
PROCESSES

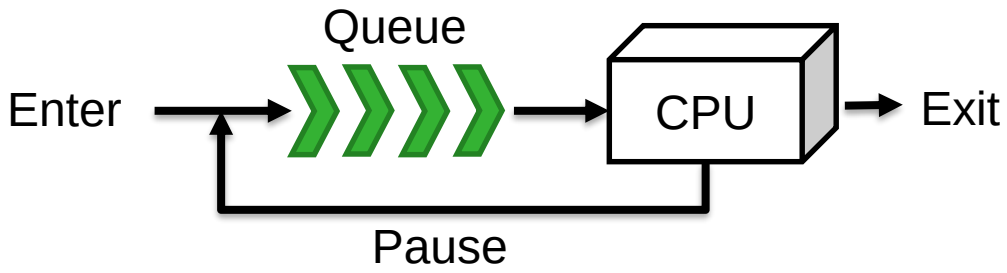
ERIK TEWS <e.tews@utwente.nl>



PRINCIPLE OF CONCURRENCY

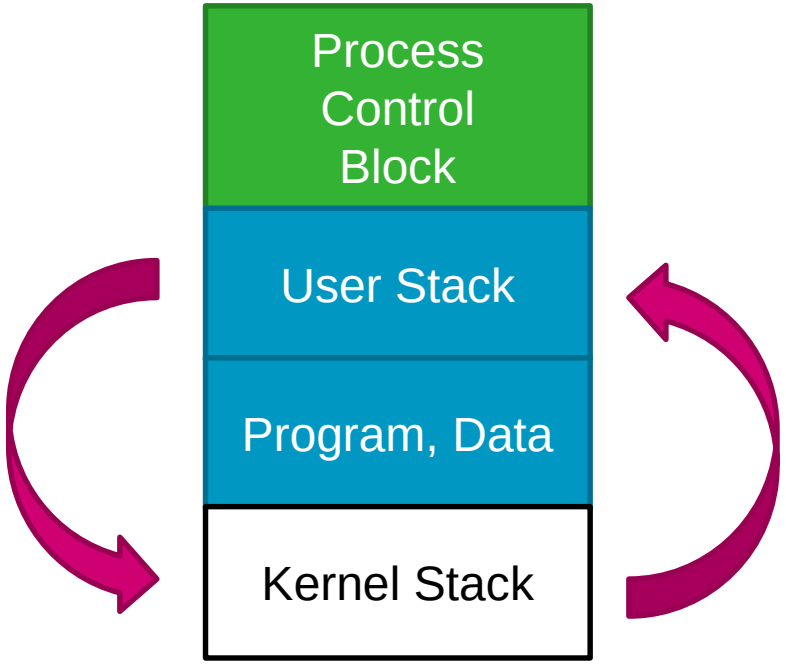
- A process has:
  - Identifier
  - State (running / not running)
  - Registers + PC
  - Memory (+ Program)
  - I/O, accounting, other ressources





PROCESS SWITCH

- User mode ⟷ Kernel mode
  - Synchronous: System call
  - Asynchronous: Interrupt



PROCESS CREATION EXAMPLE

- Output?
- gcc Fork.c
- strace ./a.out
- strace -f ./a.out

- Which system call?

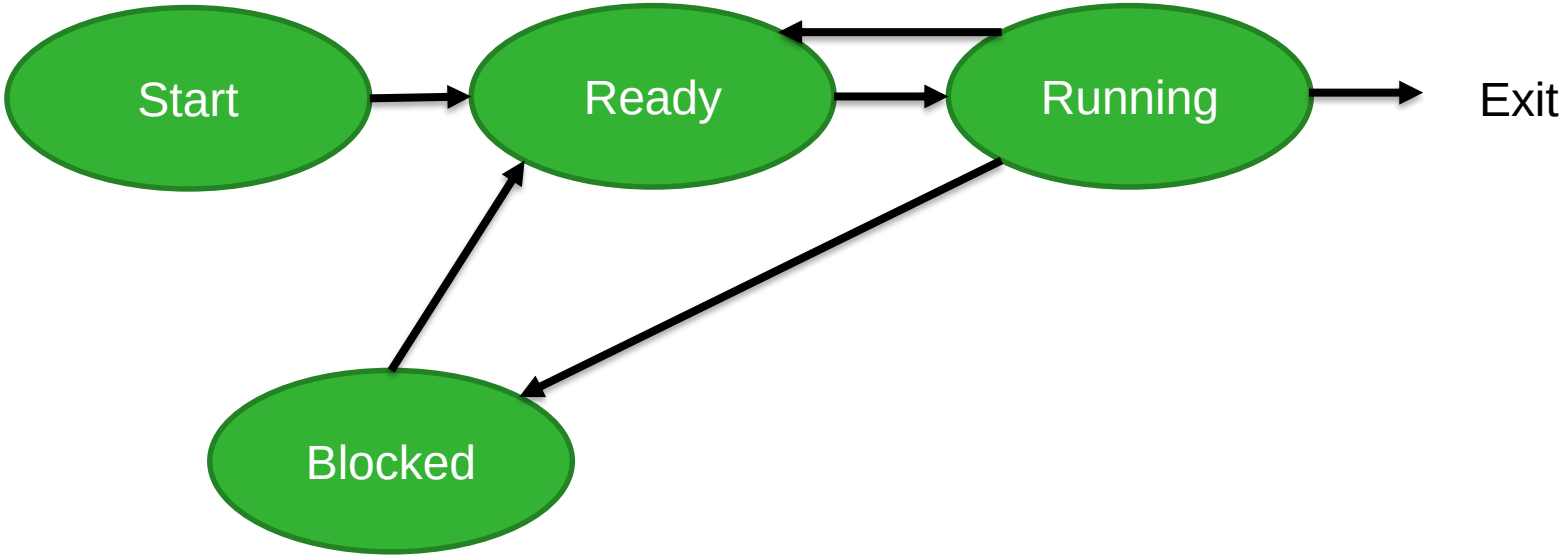
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <wait.h>
int main(int argc, char *argv[]) {
 pid_t pid=fork();
 printf("%s\n", argv[0]);
 if (pid==0) { /* child process */
  static char *argv[]={"echo","Foo",NULL};
  execv("/bin/echo",argv);
  exit(127); /* only if execv fails */
 } else { /* pid!=0; parent process */
  waitpid(pid,0,0); /* wait for child exit */
 }
 return 0;
}
```

UNIX SIGNAL EXAMPLE

- Output?
- gcc Signal.c
- ./a.out

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
void sigsegv_handler(int sig) {
 printf("SIGSEGV received.\n");
 exit(0);
}
int main() {
 int *pointer=(int *)NULL;
 signal(SIGSEGV,sigsegv_handler);
 printf("About to segfault:\n");
 *pointer=0;
 printf("Why didn't we crash?\n");
 return 1;
}
```

QUEUING



SUMMARY

- Principle of concurrency
- A process is code+data+state
- Management involves creation, switching and termination
- Signals can be used between processes