Operating Systems

(Sistemas Operativos)

Guide 2: Fork

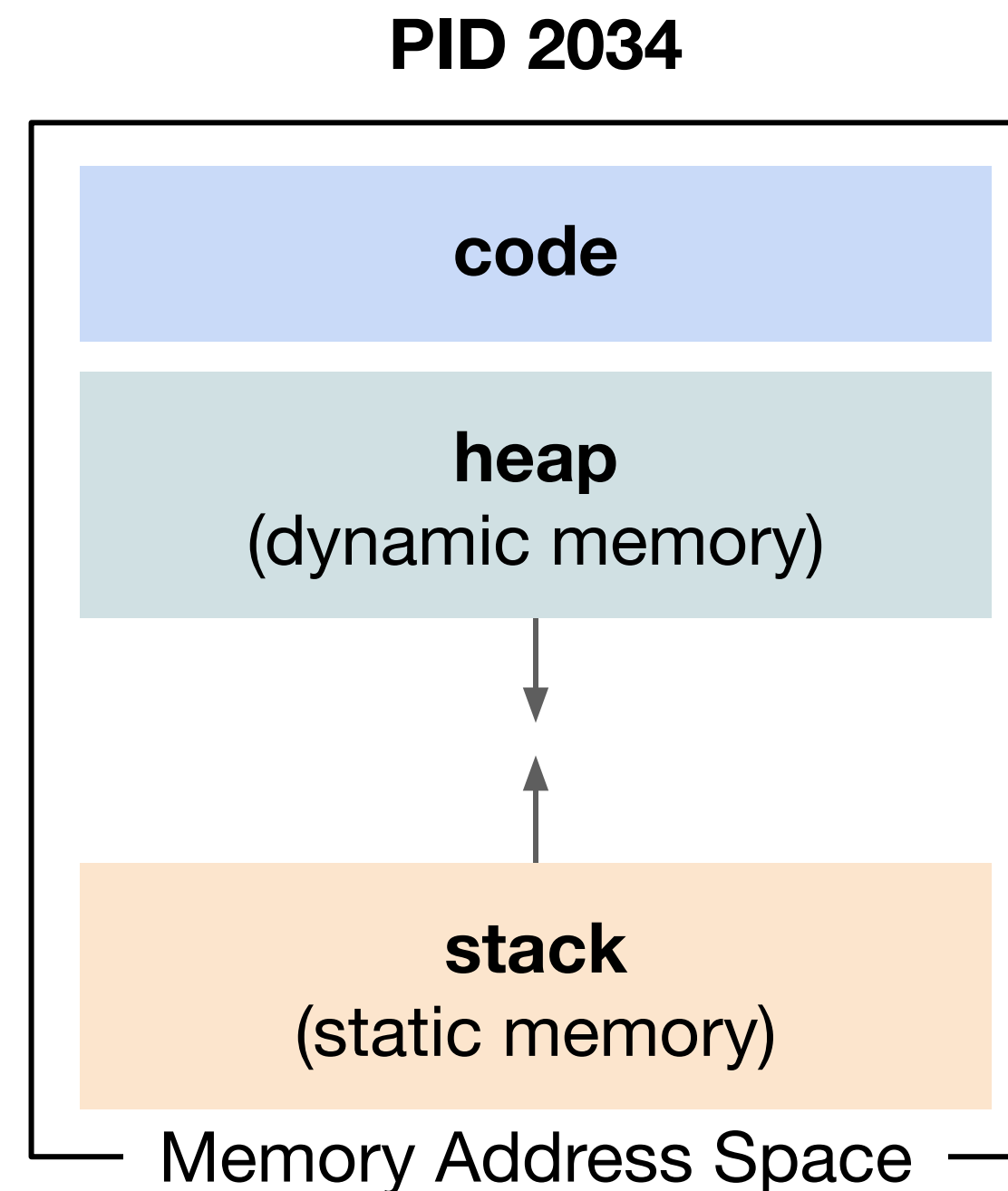University of Minho

2024 - 2025
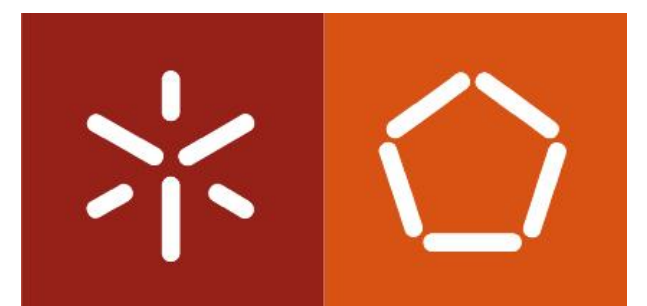
Process API
Memory address space

A **process**, identified by a **process identifier (PID)**, has access to its own **memory address space**

PID 2034

code
heap (dynamic memory)
stack (static memory)

Memory Address Space

*simplified representation of an address space (*e.g.*, not including the static data segment)

Process API
Creating a process
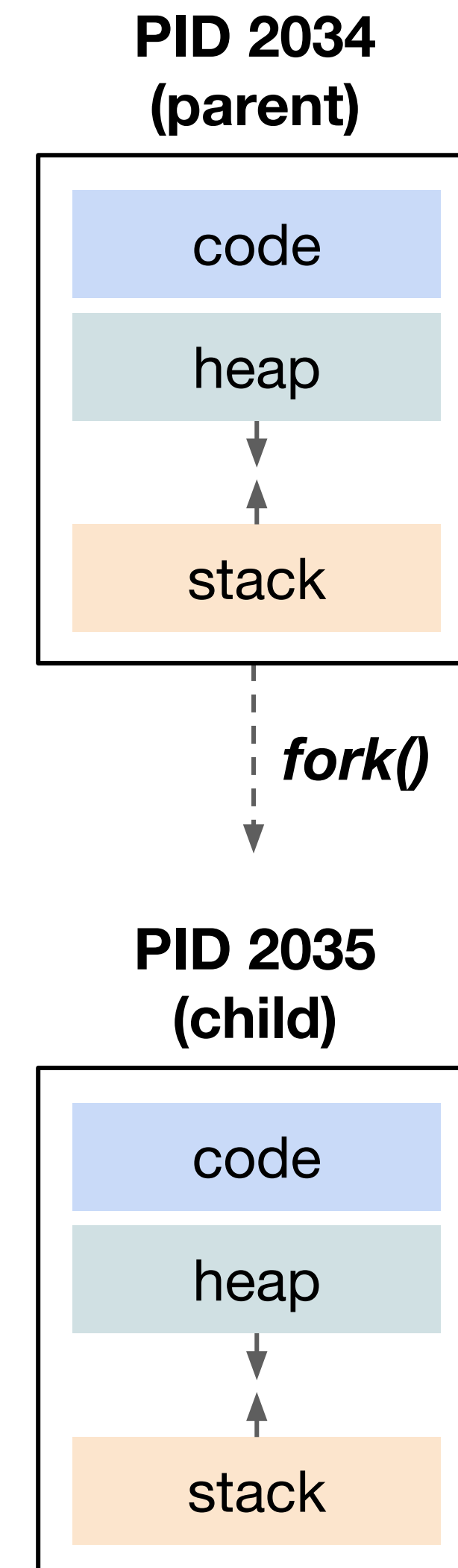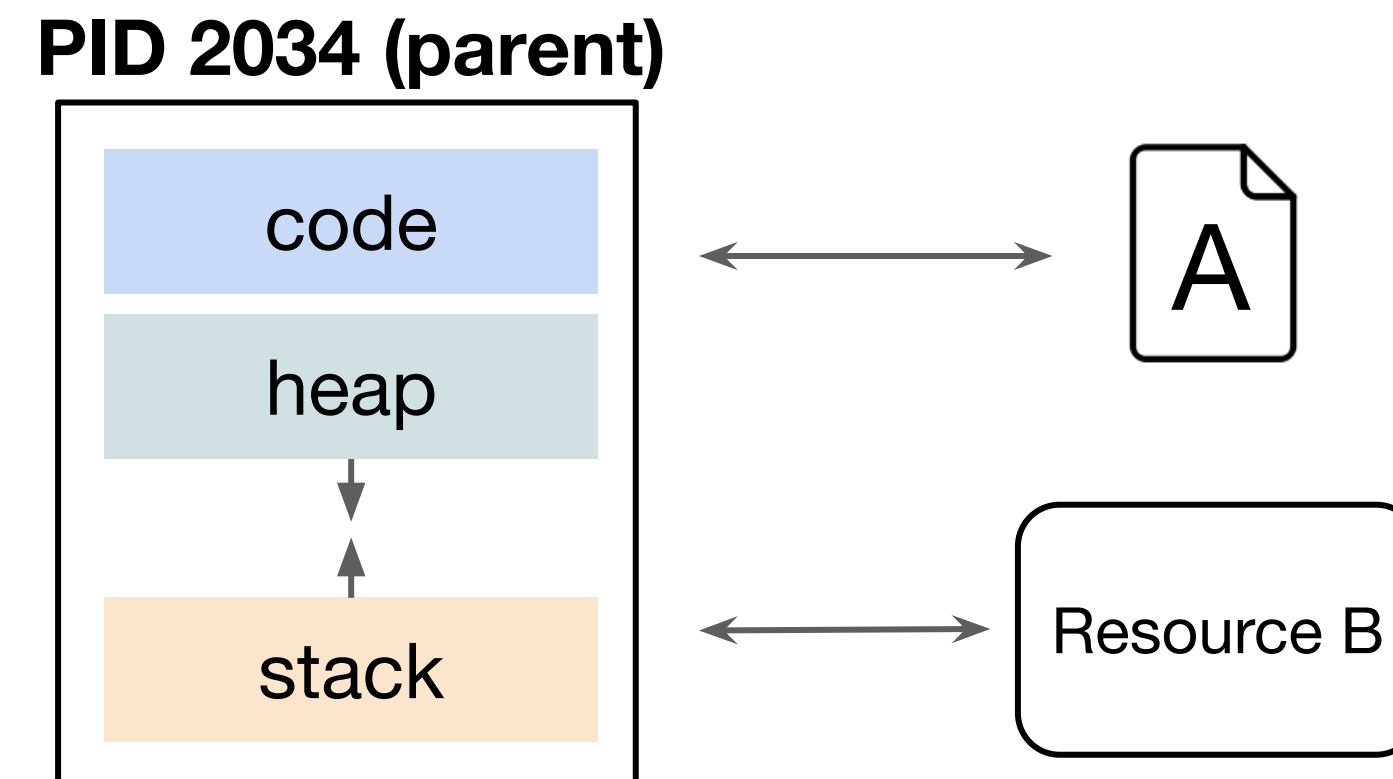
**PID 2034
(parent)**

code
heap
stack

fork()

**PID 2035
(child)**

code
heap
stack

#include <unistd.h>

- *pid_t* **fork***(void)*
  - Returns:
    - the **PID** of the **child-process** <u>to the parent process</u>
    - **0** <u>**to the child-process**</u>
    - **-1 on error**

For more information: *$ man 2 fork*
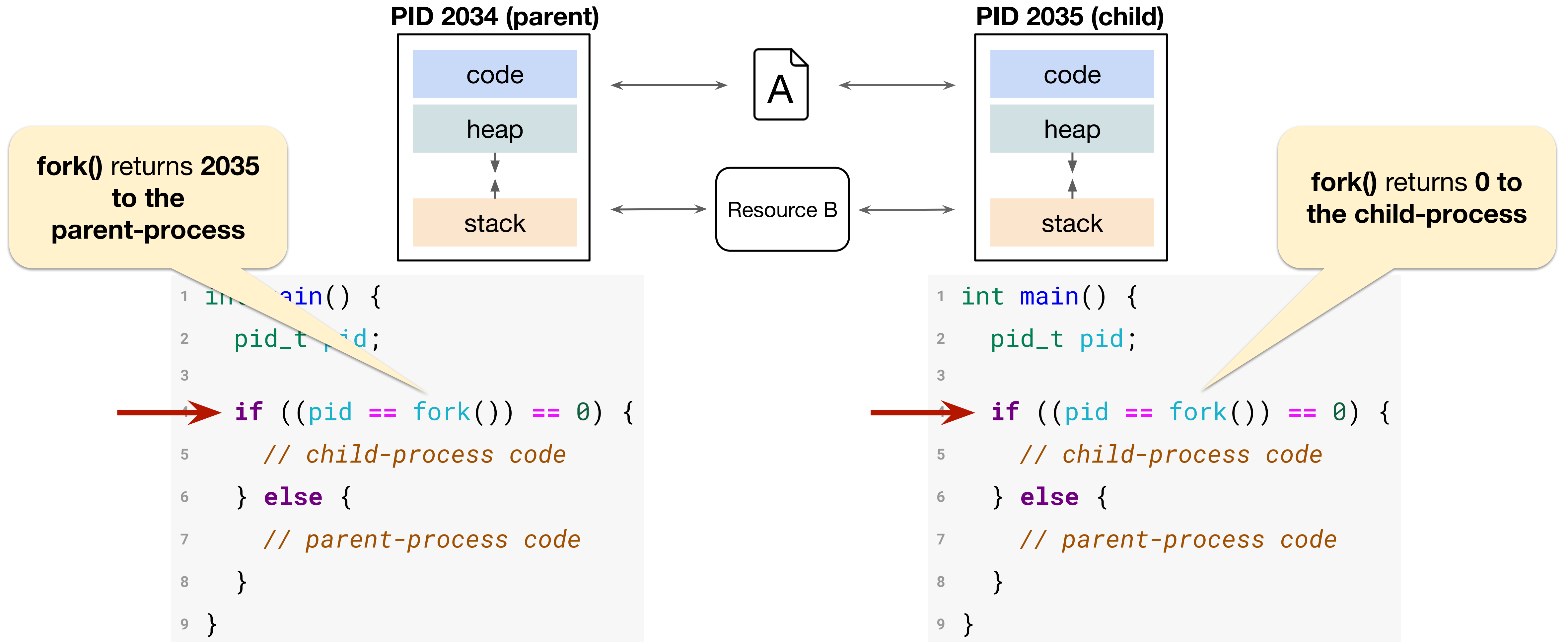
Process API
Example: creating a process

PID 2034 (parent)


```
1  int main() {
2    pid_t pid;
3
4    if ((pid == fork()) == 0) {
5      // child-process code
6    } else {
7      // parent-process code
8    }
9  }
```

Process API
Example: creating a process

PID 2034 (parent)

| code |
| heap |
| stack |

A

Resource B

PID 2035 (child)

| code |
| heap |
| stack |

fork() returns **2035** to the parent-process

fork() returns **0 to the child-process**

```
1  int main() {
2    pid_t pid;
3
4    if ((pid == fork()) == 0) {
5      // child-process code
6    } else {
7      // parent-process code
8    }
9  }
```

```
1  int main() {
2    pid_t pid;
3
4    if ((pid == fork()) == 0) {
5      // child-process code
6    } else {
7      // parent-process code
8    }
9  }
```

Process API

Example: creating a process

PID 2034 (parent)

| code |
| heap |
| stack |

A

Resource B

PID 2035 (child)

| code |
| heap |
| stack |

```
1  int main() {
2    pid_t pid;
3
4    if ((pid == fork()) == 0) {
5      // child-process code
6    } else {
7      // parent-process code
8    }
9  }
```
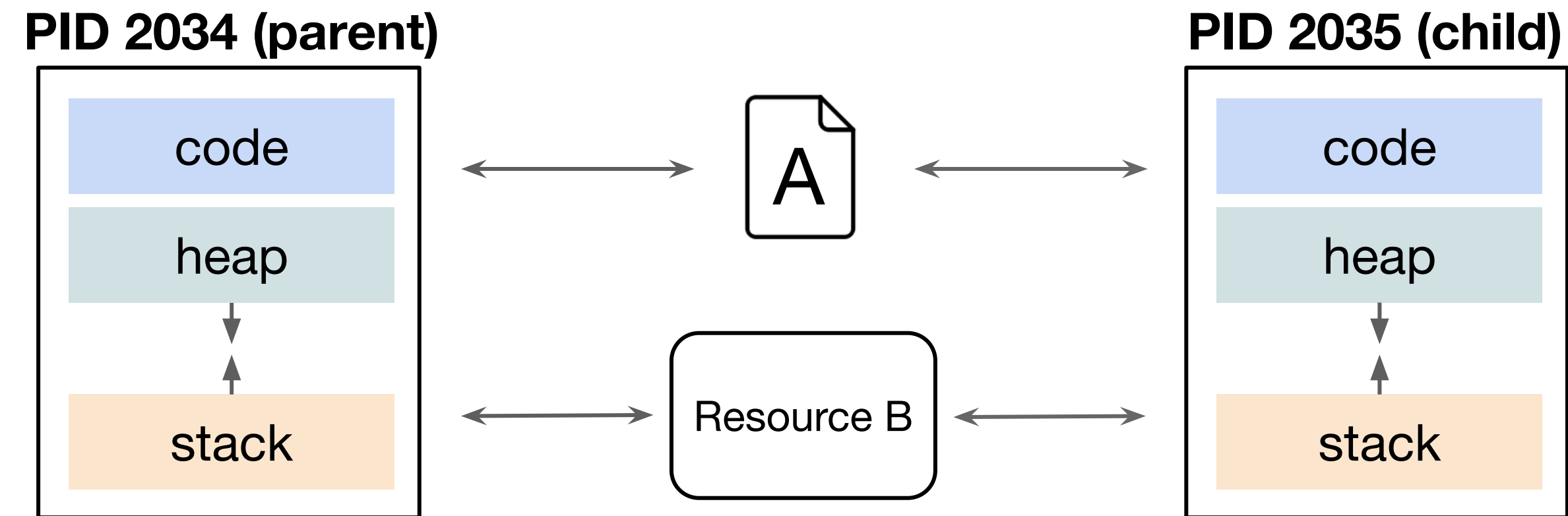
parent enters here →

```
1  int main() {
2    pid_t pid;
3
4    if ((pid == fork()) == 0) {
5      // child-process code
6    } else {
7      // parent-process code
8    }
9  }
```

child enters here →

Process API
Example: creating a process

PID 2034 (parent)

code

heap

stack

A

Resource B

PID 2035 (child)

code

heap

stack

Open resources of the parent-process are **inherited by the child-process**

```
1  int main() {
2    pid_t pid;
3
4    if ((pid == fork()) == 0) {
5      // child-process code
6    } else {
7      // parent-process code
8    }
9  }
```
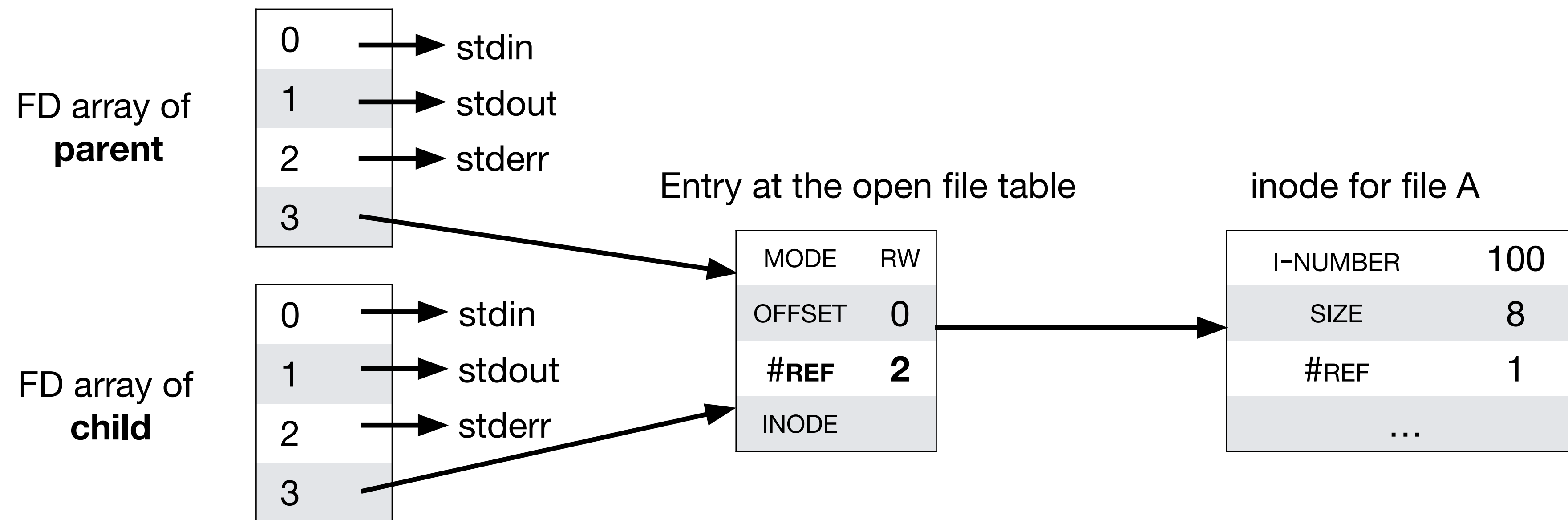
```
1  int main() {
2    pid_t pid;
3
4    if ((pid == fork()) == 0) {
5      // child-process code
6    } else {
7      // parent-process code
8    }
9  }
```
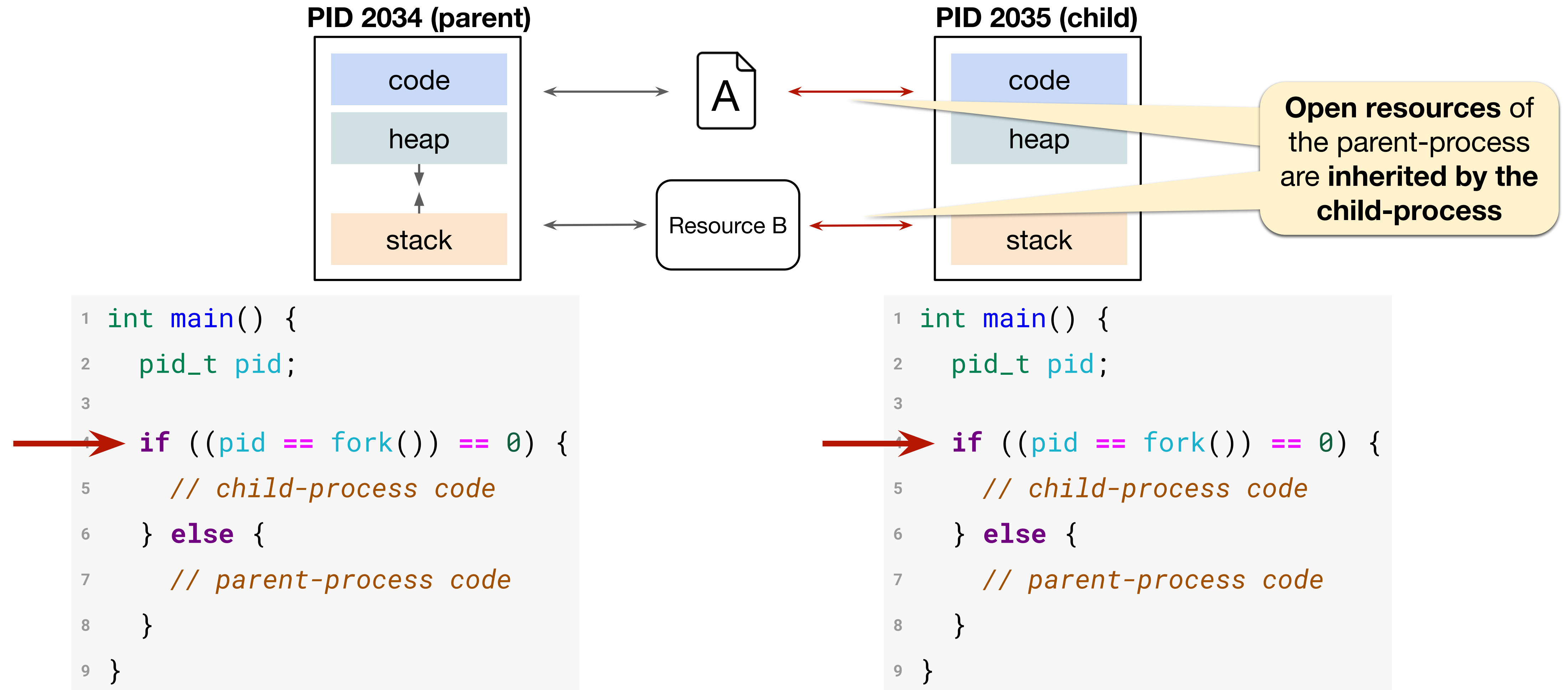
File System Interface
Shared open file table entries with **fork**

- Parent and child **share the open file table entry**

- **Be careful:** reads, writes, and seeks may update the **offset field concurrently**!

FD array of **parent**

0	→ stdin
1	→ stdout
2	→ stderr
3	

FD array of **child**

0	→ stdin
1	→ stdout
2	→ stderr
3	

Entry at the open file table

MODE	RW
OFFSET	0
#REF	**2**
INODE	

inode for file A

I-NUMBER	100
SIZE	8
#REF	1
…	

Process API
Terminating processes: **child's perspective**

#include <unistd.h>

- *void **_exit**(int status)*
  - ○ **status:** status of the current process when exiting
    - ■ 0: the process exited normally

For more information: *$ man 2 exit*

Process API
Terminating processes: **parent's perspective**

#include <sys/wait.h>

- *pid_t **wait**(int *status)*

    - **status:** memory address where termination information of the child-process is written to

    - Returns: the **PID** of the **terminated child-process**

For more information: *$ man 2 wait*

#include <sys/wait.h>

- **WIFEXITED**(*status*)
    - Returns: 1 if the child-process exited normally

- **WEXITSTATUS**(*status*)
    - employed **only if WIFEXITED returned 1**

    - Returns: the **exit status of the child –** the least significant 8 bits of *status* **specified when the child exited**

Process API
Terminating processes: **parent's perspective**

#include <sys/wait.h>

- *pid_t **waitpid**(pid_t pid, int * wstatus, int options)*
  - **pid:**
    - **> 0:** wait for the **child process whose PID is *pid***
    - check wait's man page for other wait behaviours that one can specify with ***pid***

  - **wstatus:** memory address where termination information of the child-process is written to

  - **options:** extra arguments that change waitpid's default behavior

For more information: *$ man 2 wait*

Process API
Example: Terminating a Process

PID 2034 (parent)

```c
1  int main() {
2    pid_t pid;
3    int status;
4
5    if ((pid == fork()) == 0) {
6      // child-process code
7      _exit(0);
8    } else {
9      // parent-process code
10     pid_t child = wait(&status);
11     if (WIFEXITED(status))
12       printf("Exit %d\n", WEXITSTATUS(status));
13     else
14       printf("Child exited with error\n");
15   }
16 }
```

PID 2035 (child)

```c
1  int main() {
2    pid_t pid;
3    int status;
4
5    if ((pid == fork()) == 0) {
6      // child-process code
7      _exit(0);
8    } else {
9      // parent-process code
10     pid_t child = wait(&status);
11     if (WIFEXITED(status))
12       printf("Exit %d\n", WEXITSTATUS(status));
13     else
14       printf("Child exited with error\n");
15   }
16 }
```

Process API
Example: Terminating a Process

PID 2034 (parent)

```
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork()) == 0) {
6       // child-process code
7       _exit(0);
8     } else {
9       // parent-process code
10      pid_t child = wait(&status);
11      if (WIFEXITED(status))
12        printf("Exit %d\n", WEXITSTATUS(status));
13      else
14        printf("Child exited with error\n");
15    }
16  }
```

PID 2035 (child)

```
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork()) == 0) {
6       // child-process code
7       _exit(0);
8     } else {
9       // parent-process code
10      pid_t child = wait(&status);
11      if (WIFEXITED(status))
12        printf("Exit %d\n", WEXITSTATUS(status));
13      else
14        printf("Child exited with error\n");
15    }
16  }
```

Process API
Example: Terminating a Process

PID 2034 (parent)

```
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork
6       // child-proces
7       _exit(0);
8     } else {
9       // parent-process  ode
10      pid_t child = wait(&status);
11      if (WIFEXITED(status))
12        printf("Exit %d\n", WEXITSTATUS(status));
13      else
14        printf("Child exited with error\n");
15    }
16  }
```

wait() blocks the parent until the child exits

PID 2035 (child)

```
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork()) == 0) {
6       // child-process code
7       _exit(0);
8     } else {
9       // parent-process code
10      pid_t child = wait(&status);
11      if (WIFEXITED(status))
12        printf("Exit %d\n", WEXITSTATUS(status));
13      else
14        printf("Child exited with error\n");
15    }
16  }
```

Process API
Example: Terminating a Process

PID 2034 (parent)

```
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork()) == 0) {
6       // child-process code
7       _exit(0);
8     } else {
9       // parent-process code
10 →    pid_t child = wait(&status);
11      if (WIFEXITED(status))
12        printf("Exit %d\n", WEXITSTATUS(status));
13      else
14        printf("Child exited with error\n");
15    }
16  }
```

PID 2035 (child)

```
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork()) == 0) {
6       // child-process code
7  →    _exit(0);
8     } else {
9       // parent-process code
10      pid_t child = wait(&status);
11      if (WIFEXITED(status))
12        printf("Exit %d\n", WEXITSTATUS(status));
13      else
14        printf("Child exited with error\n");
15    }
16  }
```

Process API
Example: Terminating a Process

PID 2034 (parent)

```c
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork()) == 0) {
6       // child-process code
7       _exit(0);
8     } else {
9       // parent-process code
10      pid_t child = wait(&status);
11      if (WIFEXITED(status))
12        printf("Exit %d\n", WEXITSTATUS(status));
13      else
14        printf("Child exited with error\n");
15    }
16  }
```

PID 2035 (child)

```c
1   int main() {
2     pid
3     int
4
5     if ((p        fork()) == 0) {
6       // c  ld-process code
7       _exit(0);
8     } else {
9       // parent-process code
10      pid_t child = wait(&status);
11      if (WIFEXITED(status))
12        printf("Exit %d\n", WEXITSTATUS(status));
13      else
14        printf("Child exited with error\n");
15    }
16  }
```

_exit() terminates the current process

Process API
Example: Terminating a Process

PID 2034 (parent)

```
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork()) == 0
6       // child-process code
7       _exit(0);
8     } else {
9       // parent-process code
10 →   pid_t child = wait(&status);
11     if (WIFEXITED(status))
12       printf("Exit %d\n", WEXITSTATUS(status));
13     else
14       printf("Child exited with error\n");
15   }
16 }
```

> The **status** passed by the child's *_exit() is* **forward to the parent**

Process API
Example: Terminating a Process

```
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork()) == 0) {
                        code
```

The **parent continues** its execution **after the child has terminated**

```
9         parent-process code
10        pid_t child = wait(&status);
11        if (WIFEXITED(status))
12          printf("Exit %d\n", WEXITSTATUS(status));
13        else
14          printf("Child exited with error\n");
15      }
16    }
```
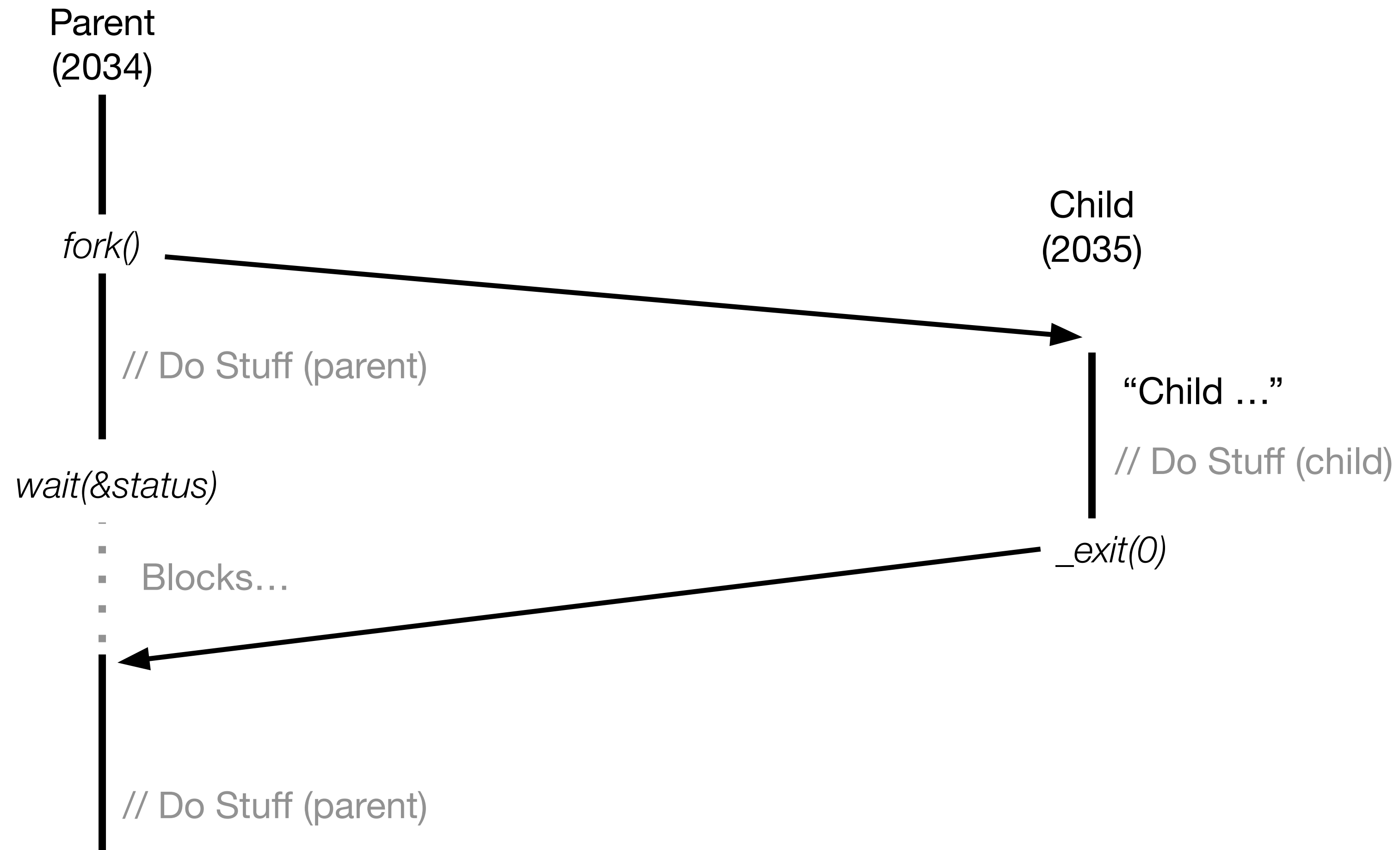
```
1   int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid == fork()) == 0) {
6       // child-process code
7       _exit(0);
8     } else {
9       // parent-process code
10      pid_t child = wait(&status);
11      if (WIFEXITED(status))
12        printf("Exit %d\n", WEXITSTATUS(status));
13      else
14        printf("Child exited with error\n");
15    }
16  }
```

Process API
Example: Terminating a Process

Parent
(2034)

fork()

// Do Stuff (parent)

Child
(2035)

"Child …"

// Do Stuff (child)

wait(&status)

Blocks…

_exit(0)

// Do Stuff (parent)

More Information

- **Chapter 5** - Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau. **Operating Systems: Three Easy Pieces.** Arpaci-Dusseau Books, 2018.

- Avi Silberschatz, Peter Baer Galvin, Greg Gagne. **Operating System Concepts (10. ed)**. John Wiley & Sons, 2018.