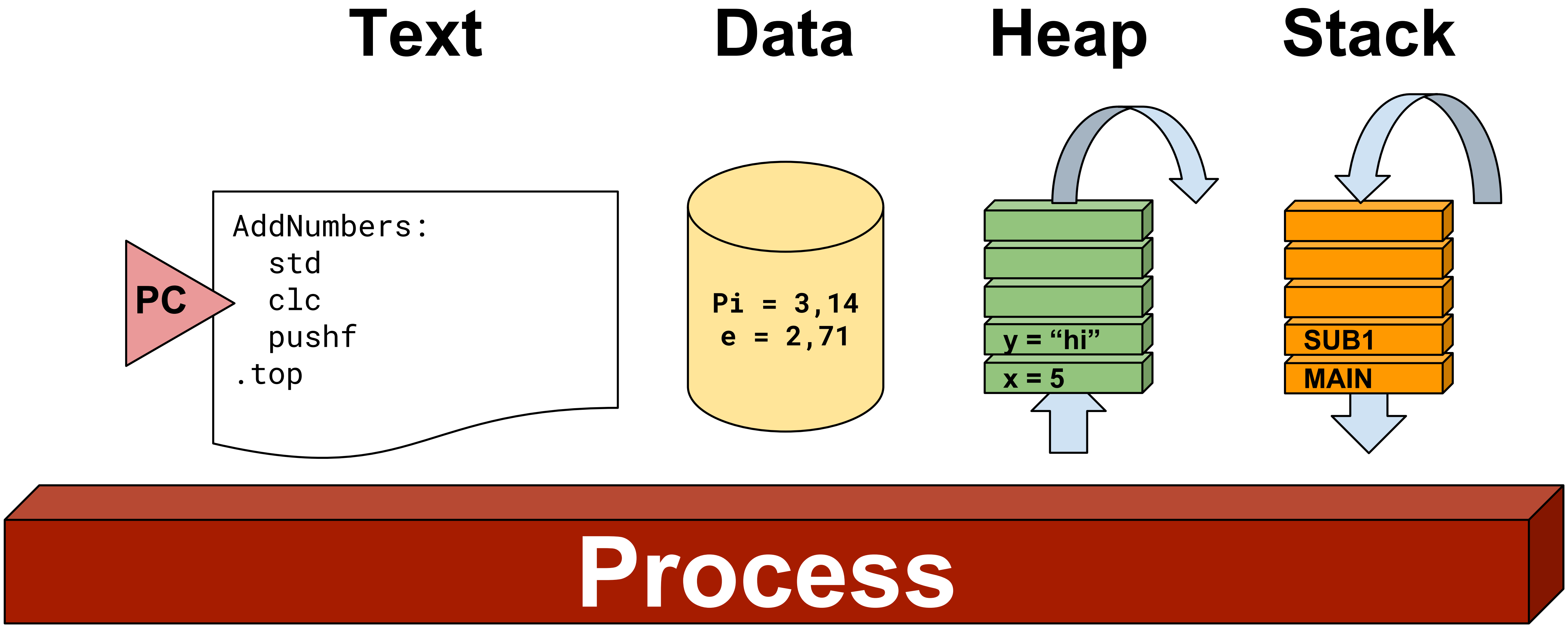
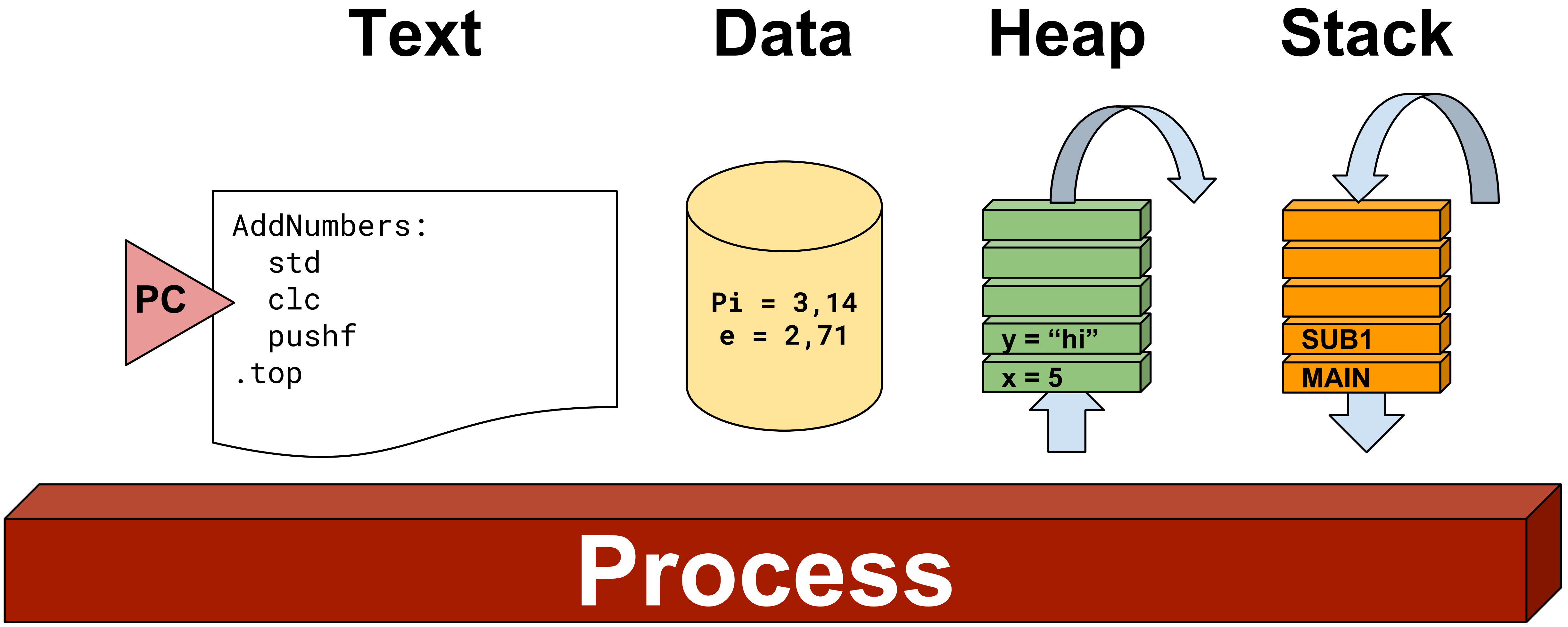
Introduction to Interprocess Communication

The Process


Text
Data
Heap
Stack
PC
AddNumbers:
std
clc
pushf
.top
Pi = 3,14
e = 2,71
y = "hi"
x = 5
SUB1
MAIN
Process

Process Control Block

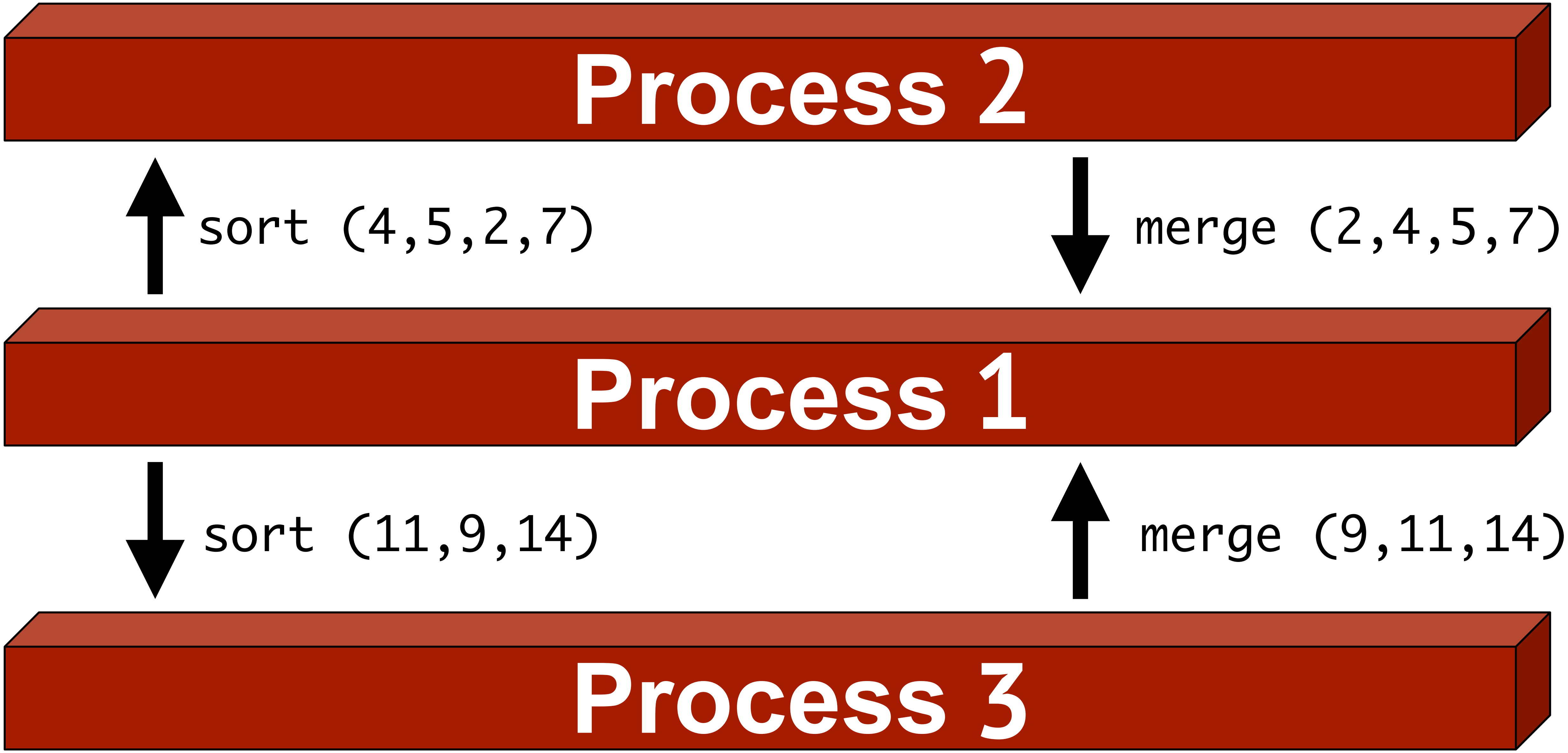


```
AddNumbers:
  std
  clc
  pushf
.top
```

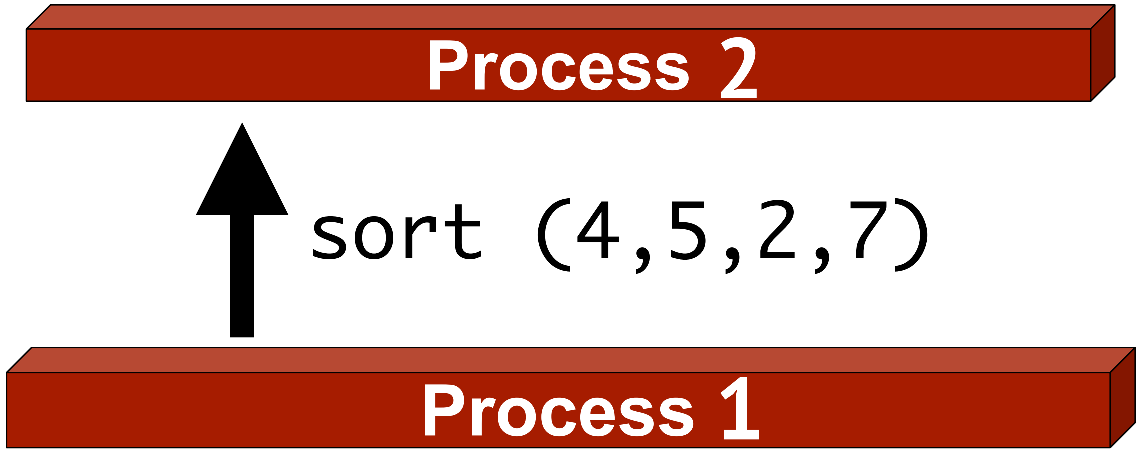
```
Pi = 3,14
 e = 2,71
```

process state	process number	process counter	REGISTERS	memory limits	pointers	…

Interprocess Communication

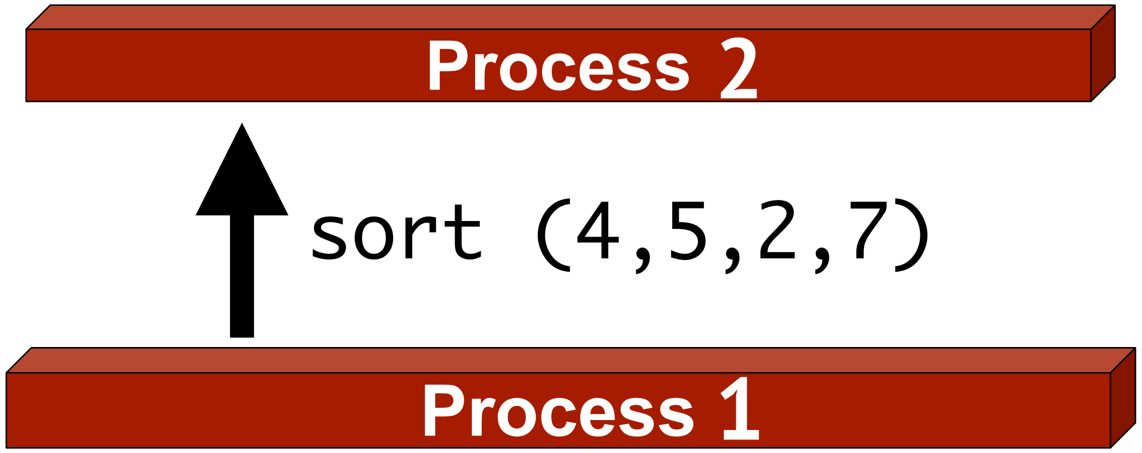


Interprocess Communication



Why do processes communicate?

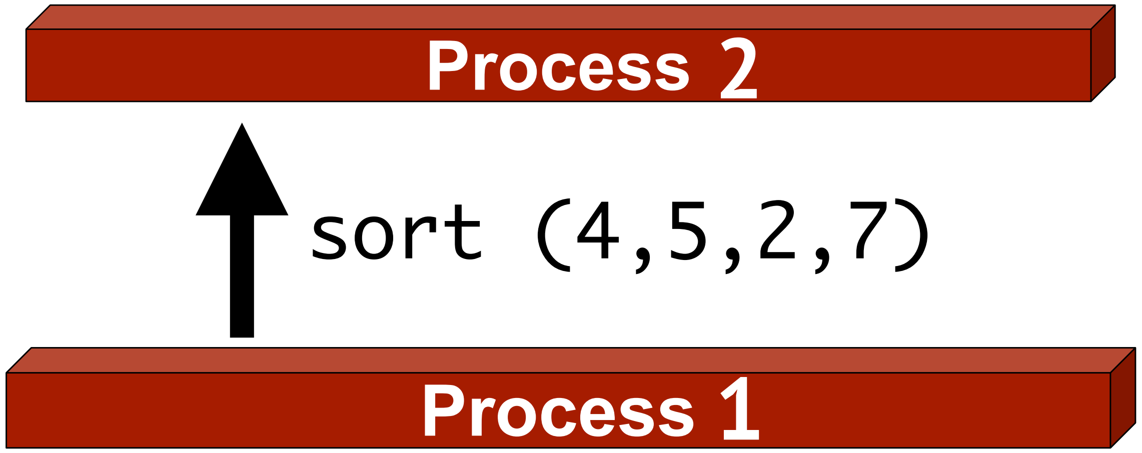
Interprocess Communication



Why do processes communicate?

- **Information sharing** - e.g., concurrent access to files;

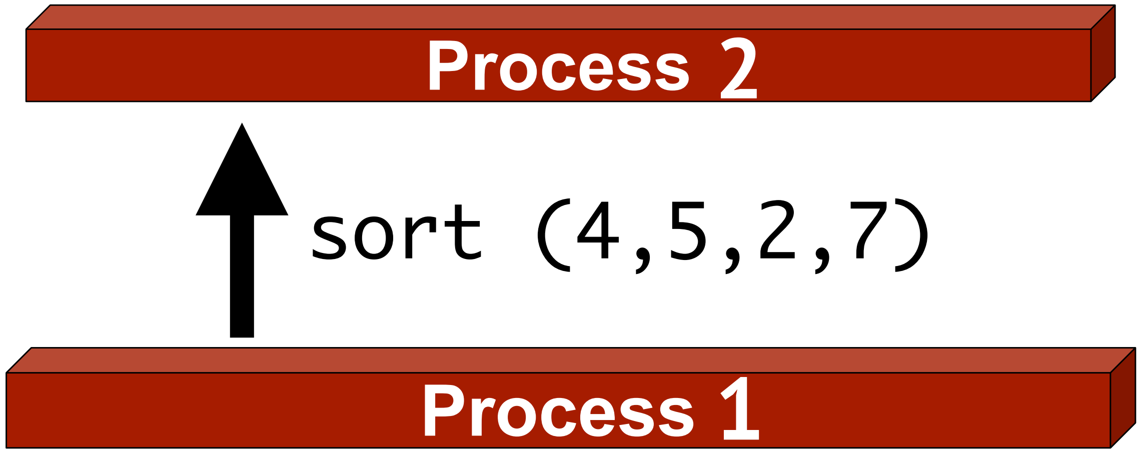
Interprocess Communication



Why do processes communicate?

- **Information sharing** - e.g., concurrent access to files;
- **Computation speed** - same aim divided in multiple tasks;

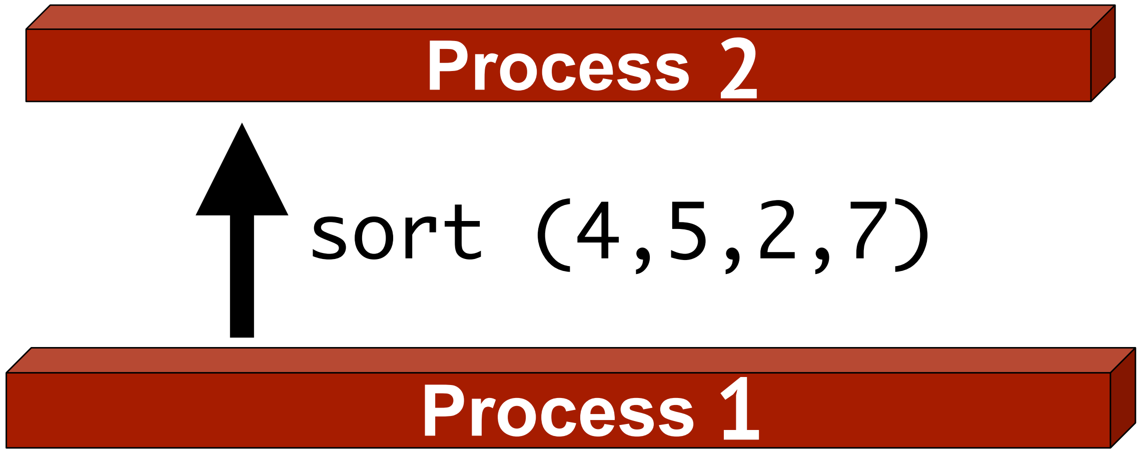
Interprocess Communication



Why do processes communicate?

- **Information sharing** - e.g., concurrent access to files;
- **Computation speed** - same aim divided in multiple tasks;
- **Modularity** - reuse processes;

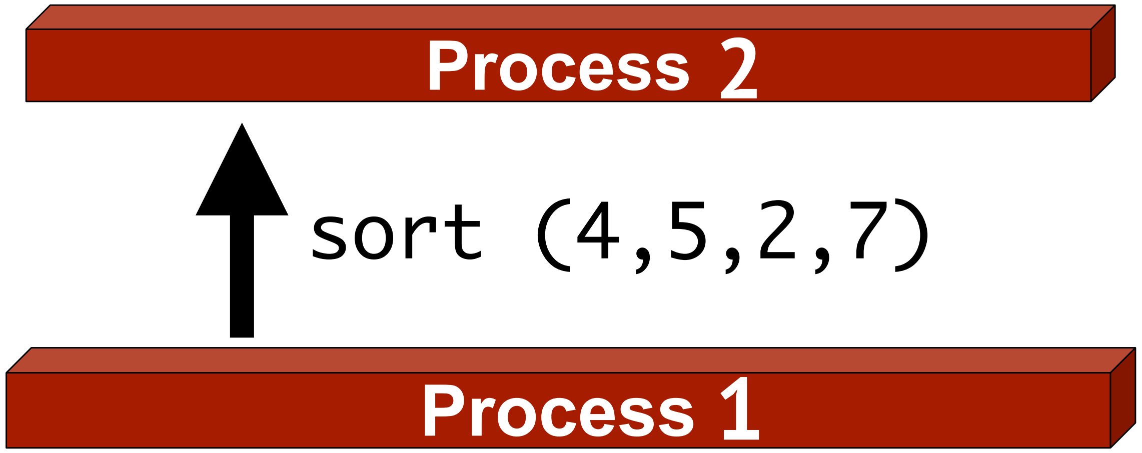
Interprocess Communication



Why do processes communicate?

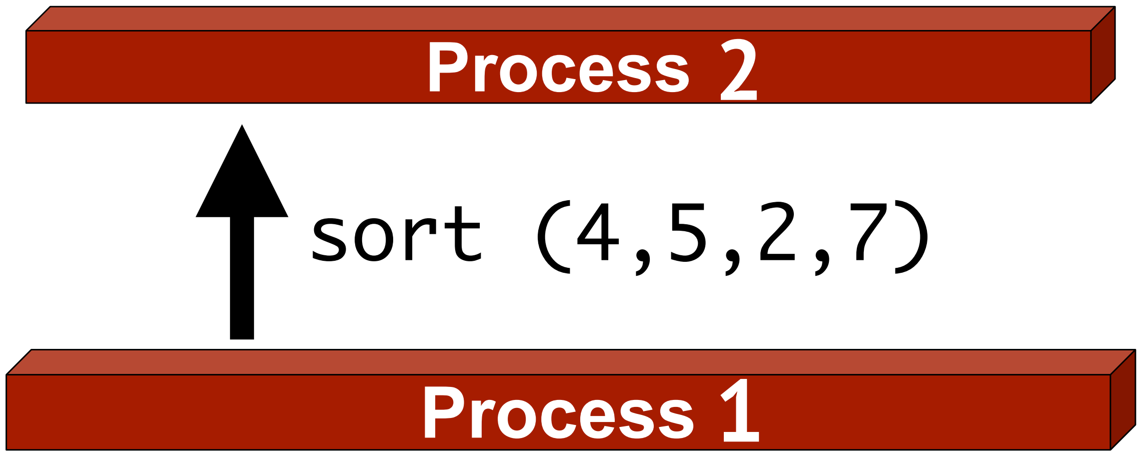
- **Information sharing** - e.g., concurrent access to files;
- **Computation speed** - same aim divided in multiple tasks;
- **Modularity** - reuse processes;
- **Convenience** - multitasking.

Interprocess Communication



How do processes communicate?

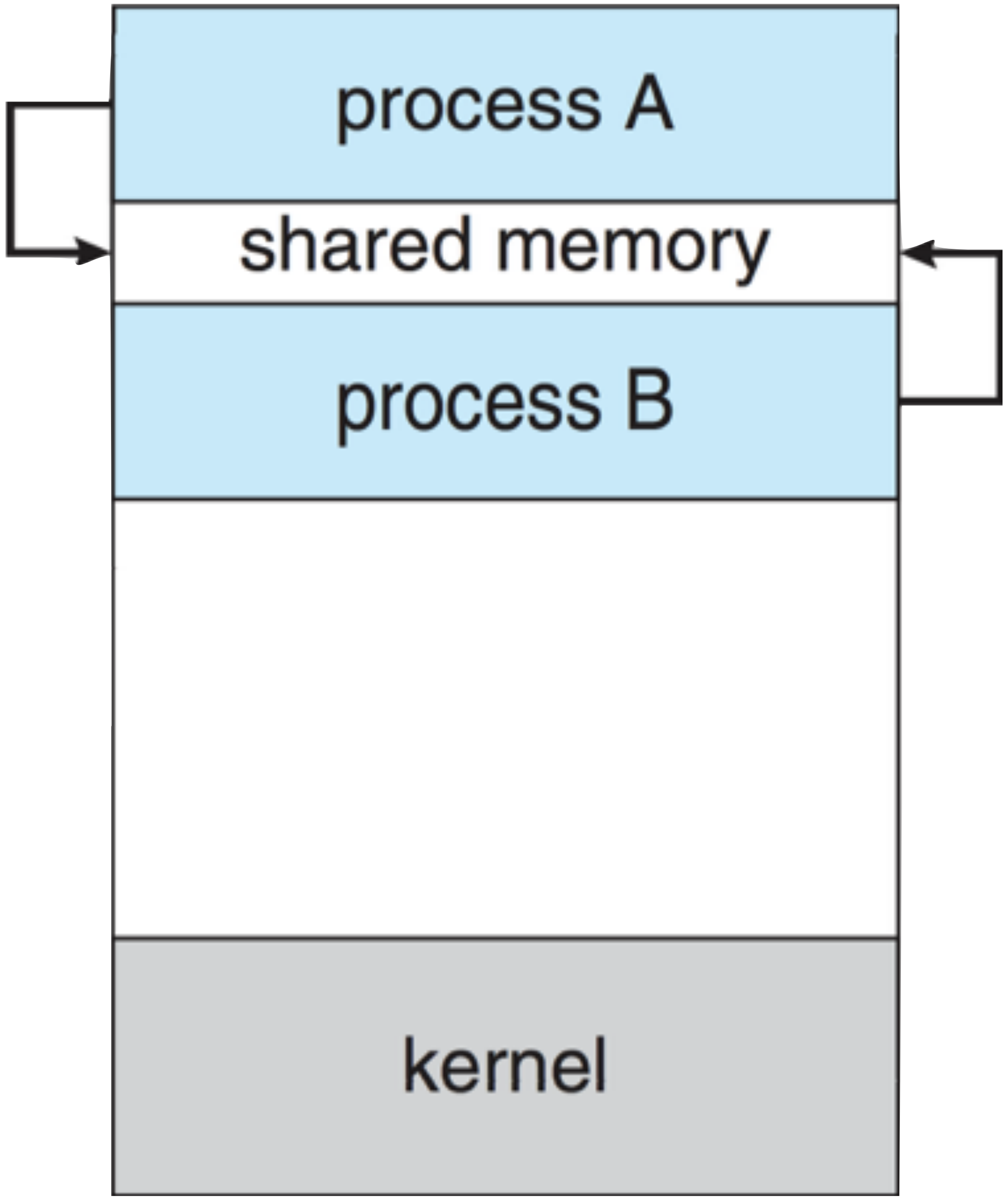
Interprocess Communication

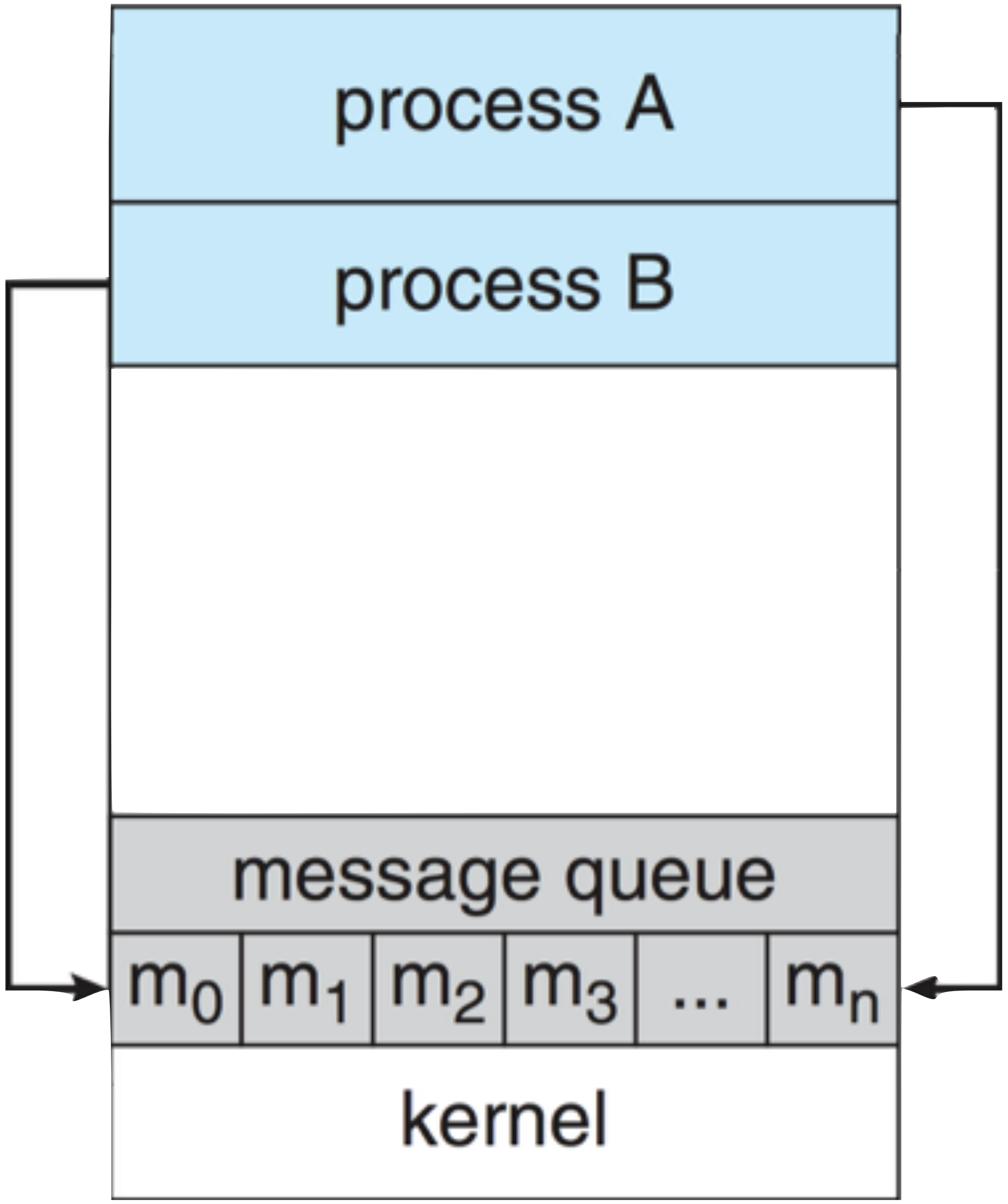


How do processes communicate?

Shared Memory V **Message Passing**

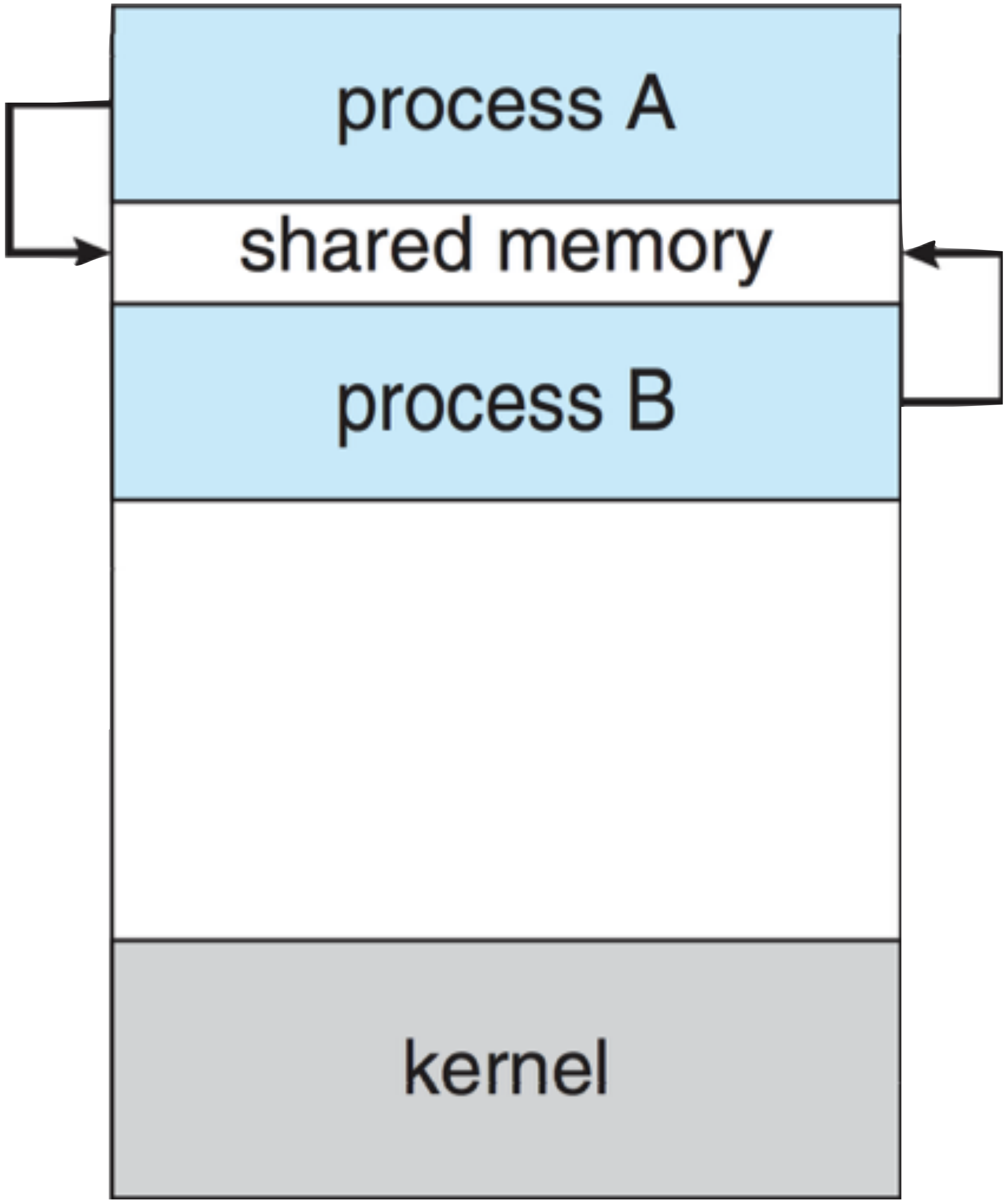
Shared Memory v Message Passing





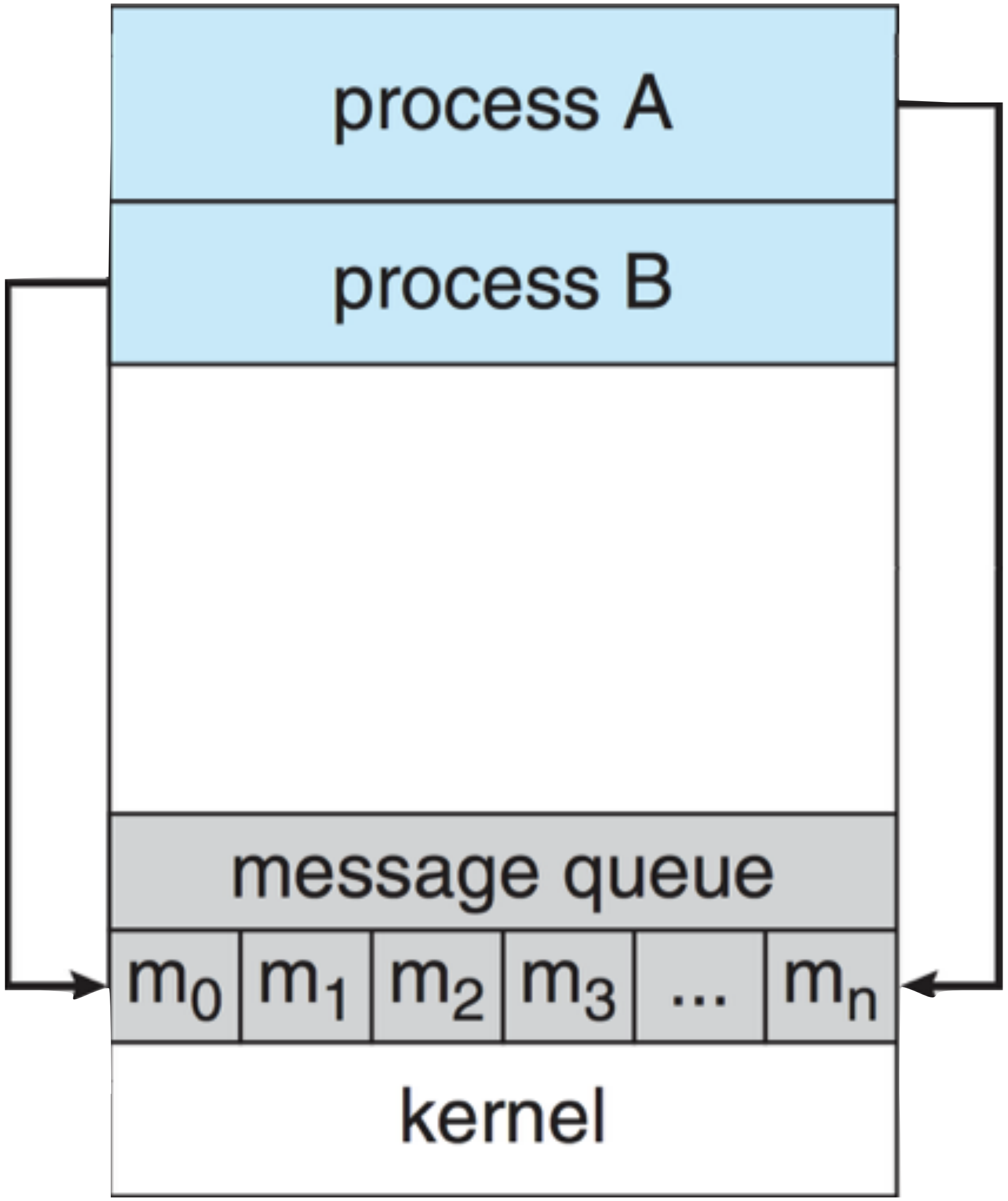
Shared Memory

- quick (and dirty);
- shared segment of memory;
- **hack-ish**, processes bypass memory protections of the OS.



Message Passing

- model scales from local to remote processes;
- needs a **communication link**

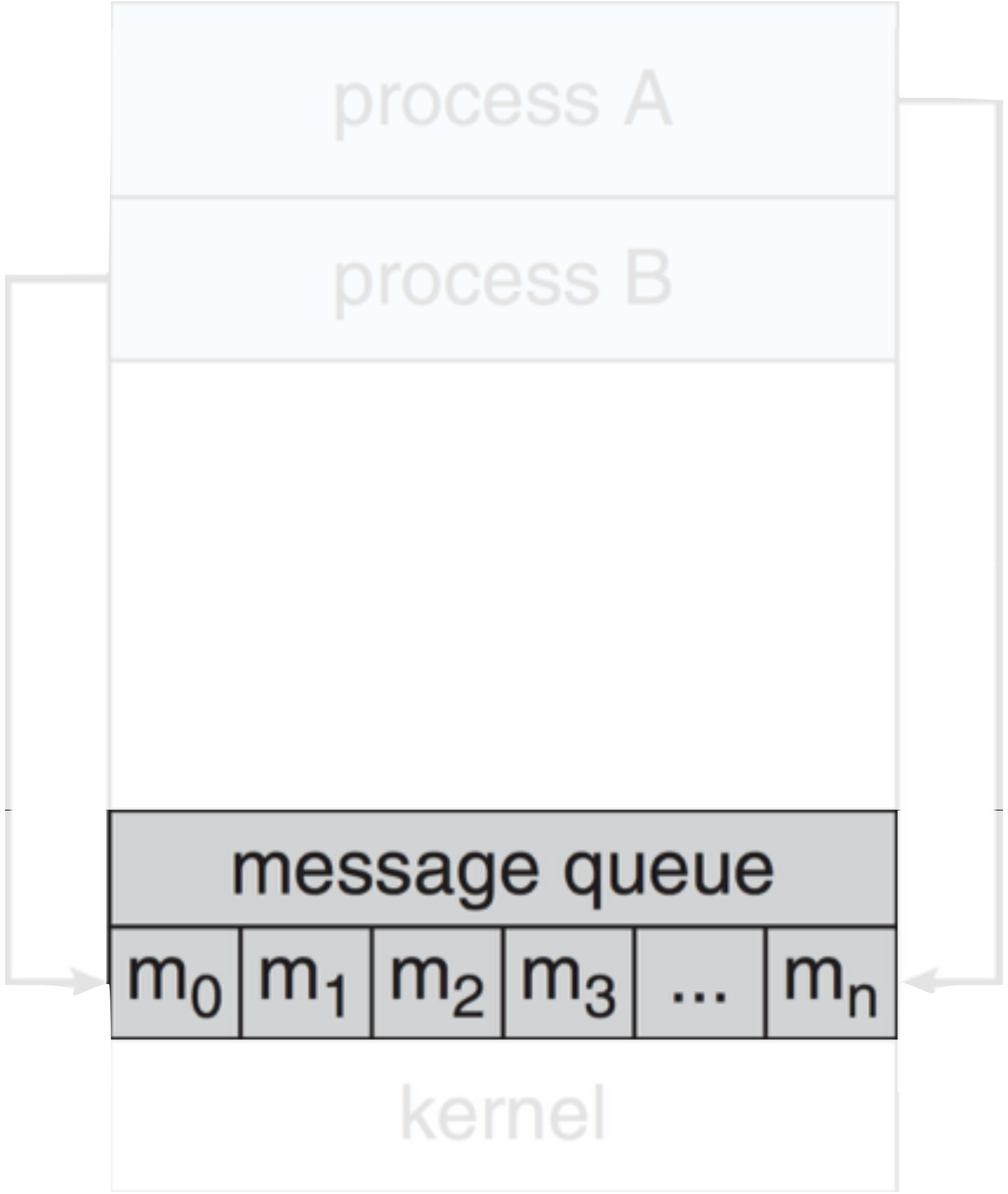


Message Passing: the communication link

Two concerns of implementation:

Physical

Logical

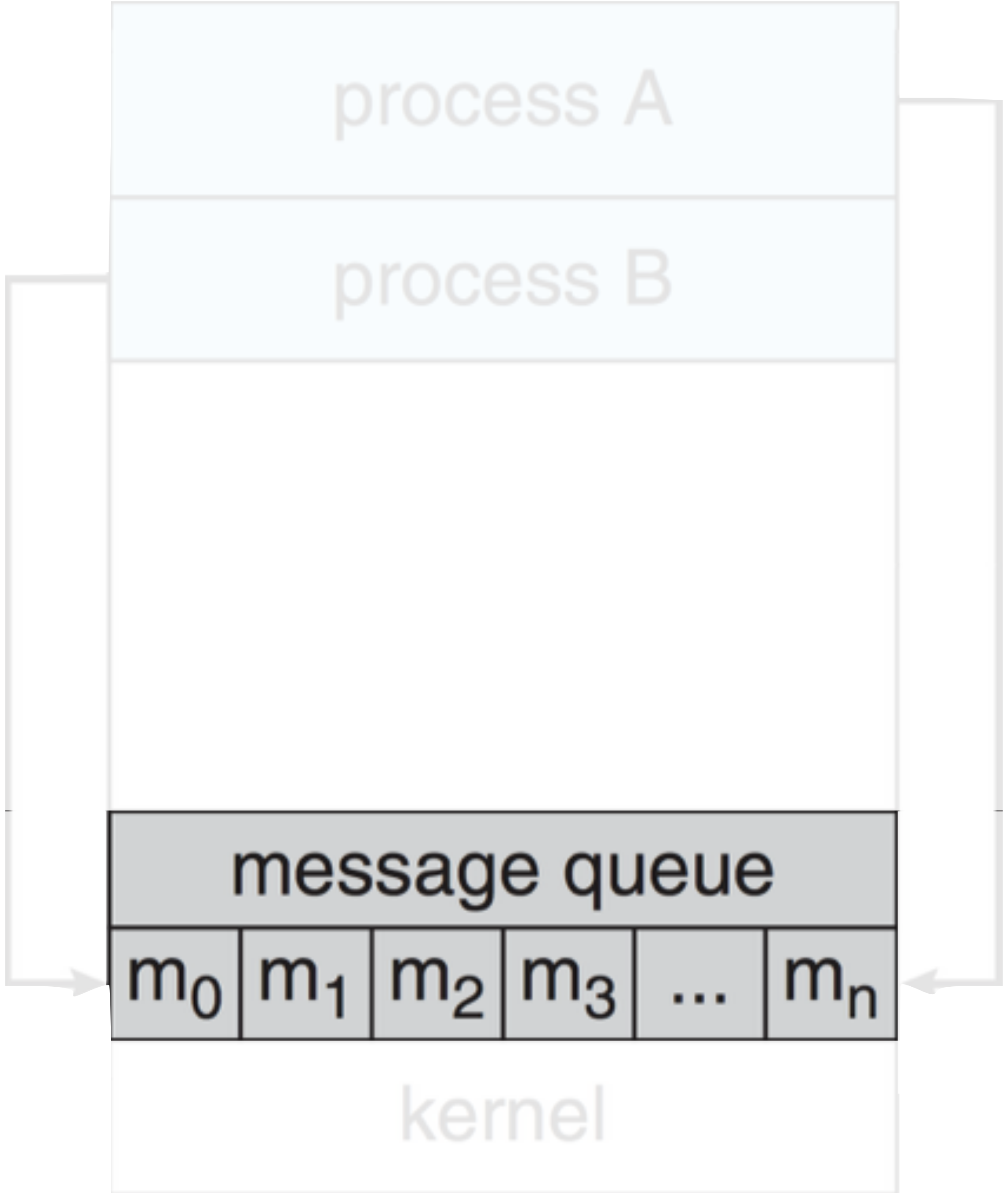


Message Passing: the communication link

Two concerns of implementation:

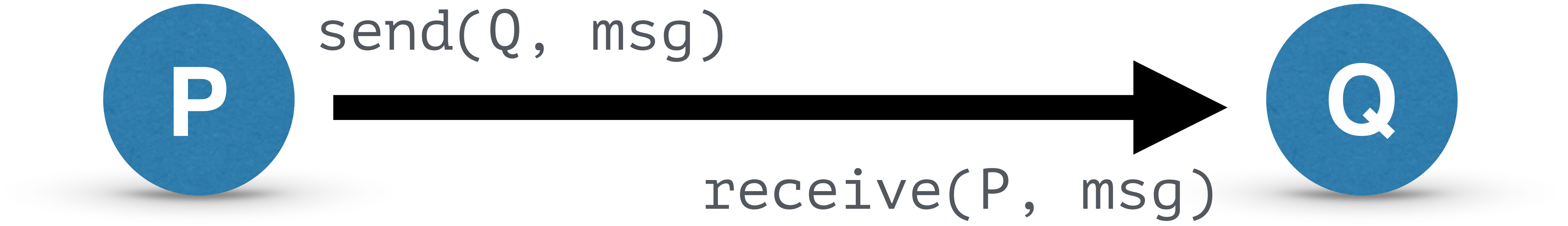
Physical

Logical



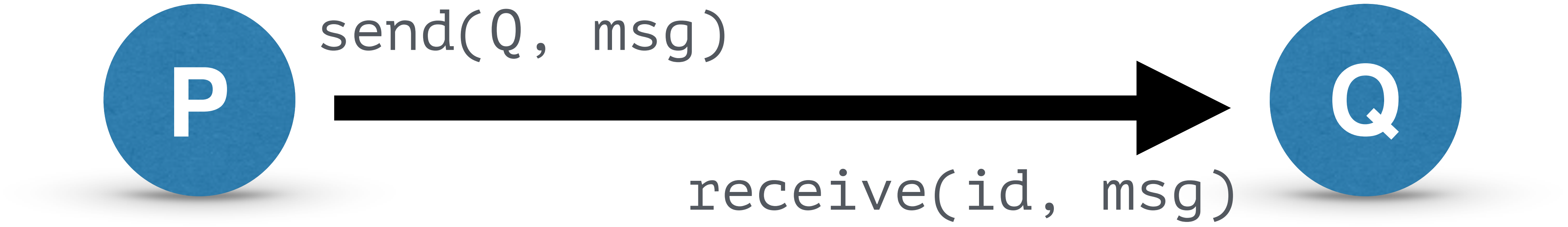
Logical Implementation

Direct communication



Logical Implementation

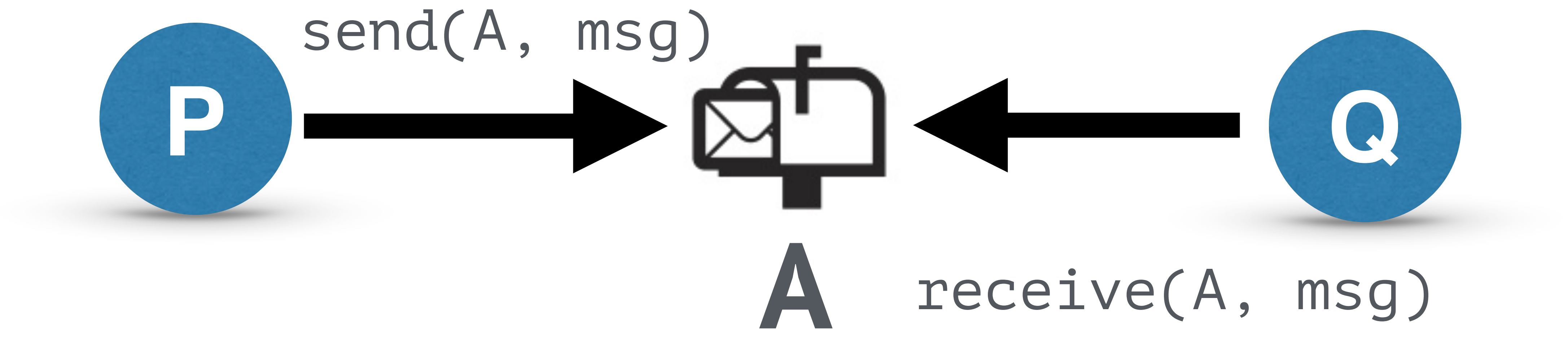
Direct communication



(asymmetric)

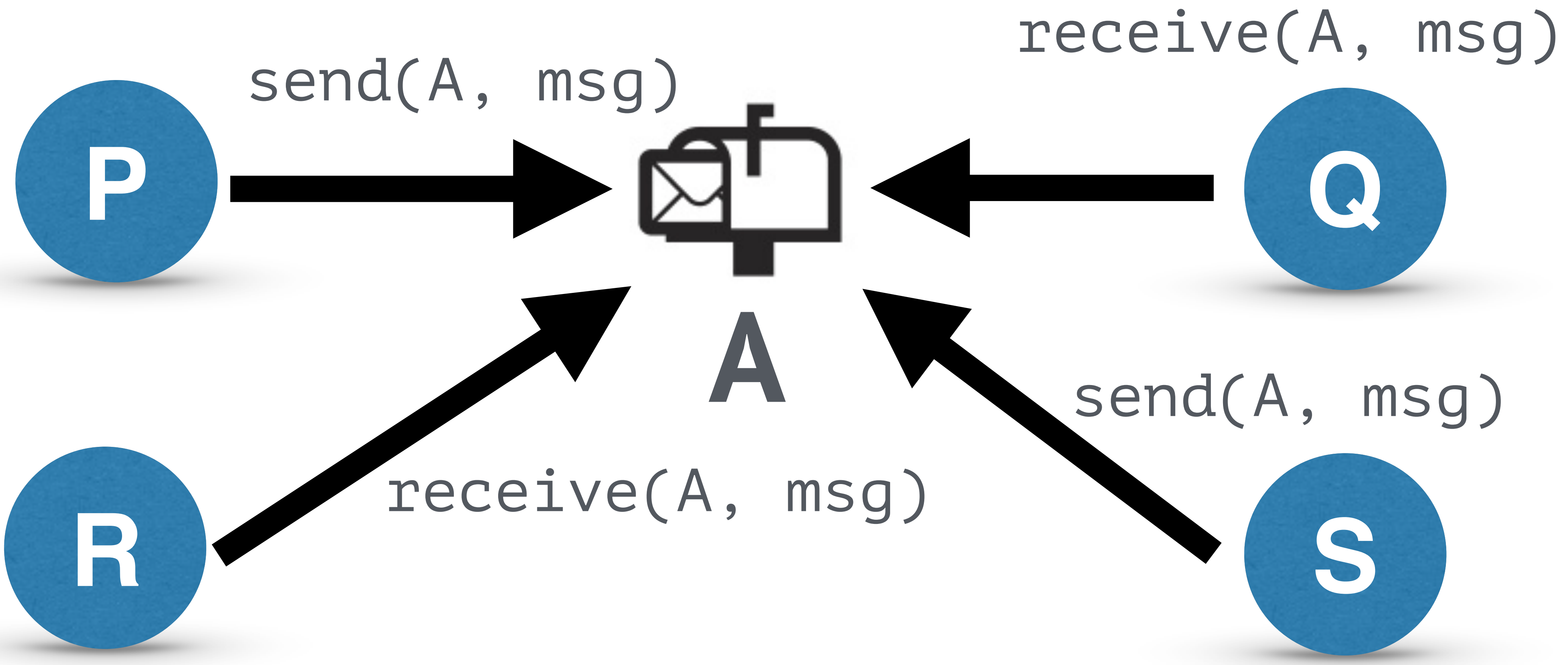
Logical Implementation

Indirect communication



Logical Implementation

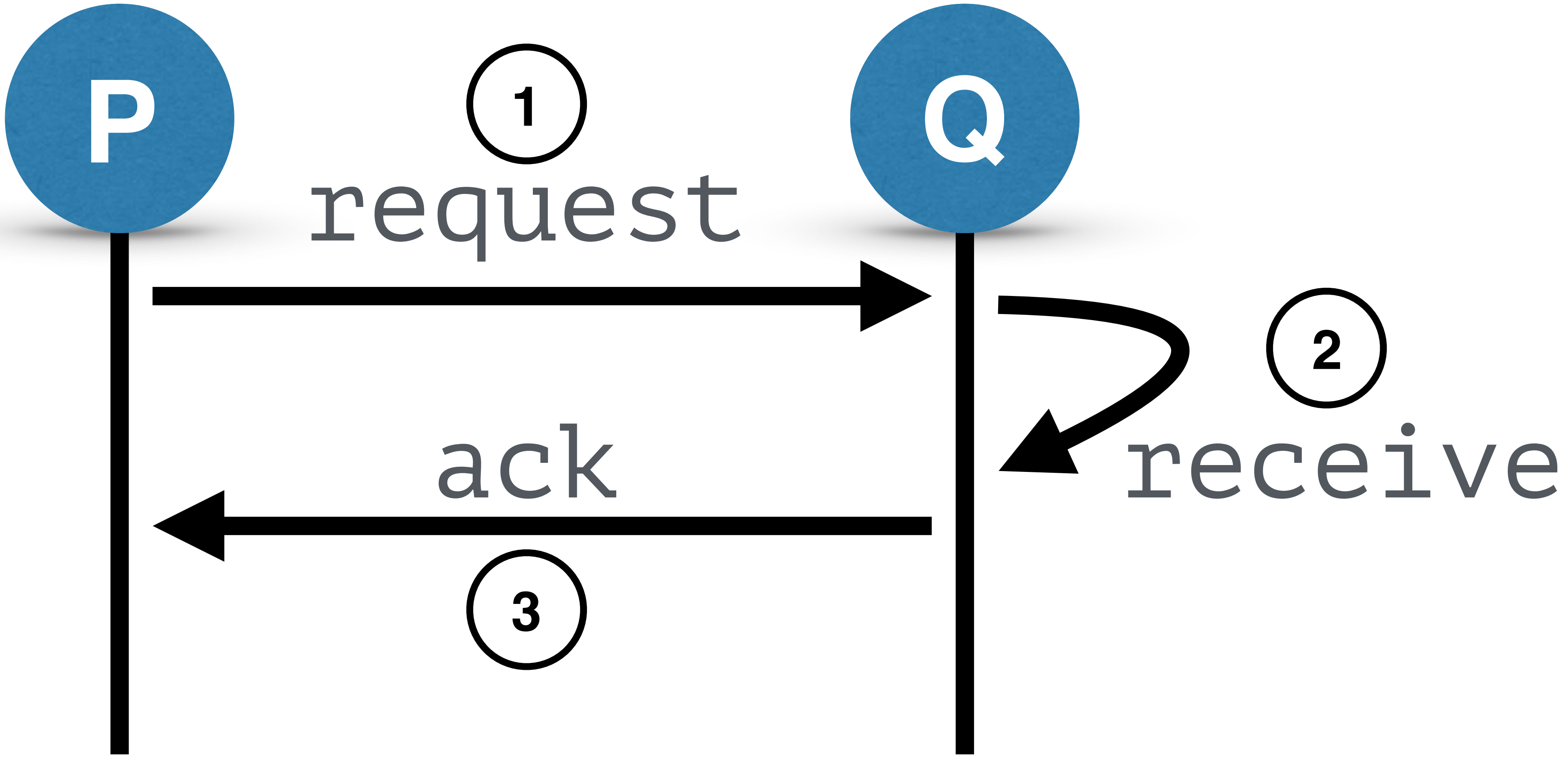
Indirect communication



Logical Implementation

Synchronous communication

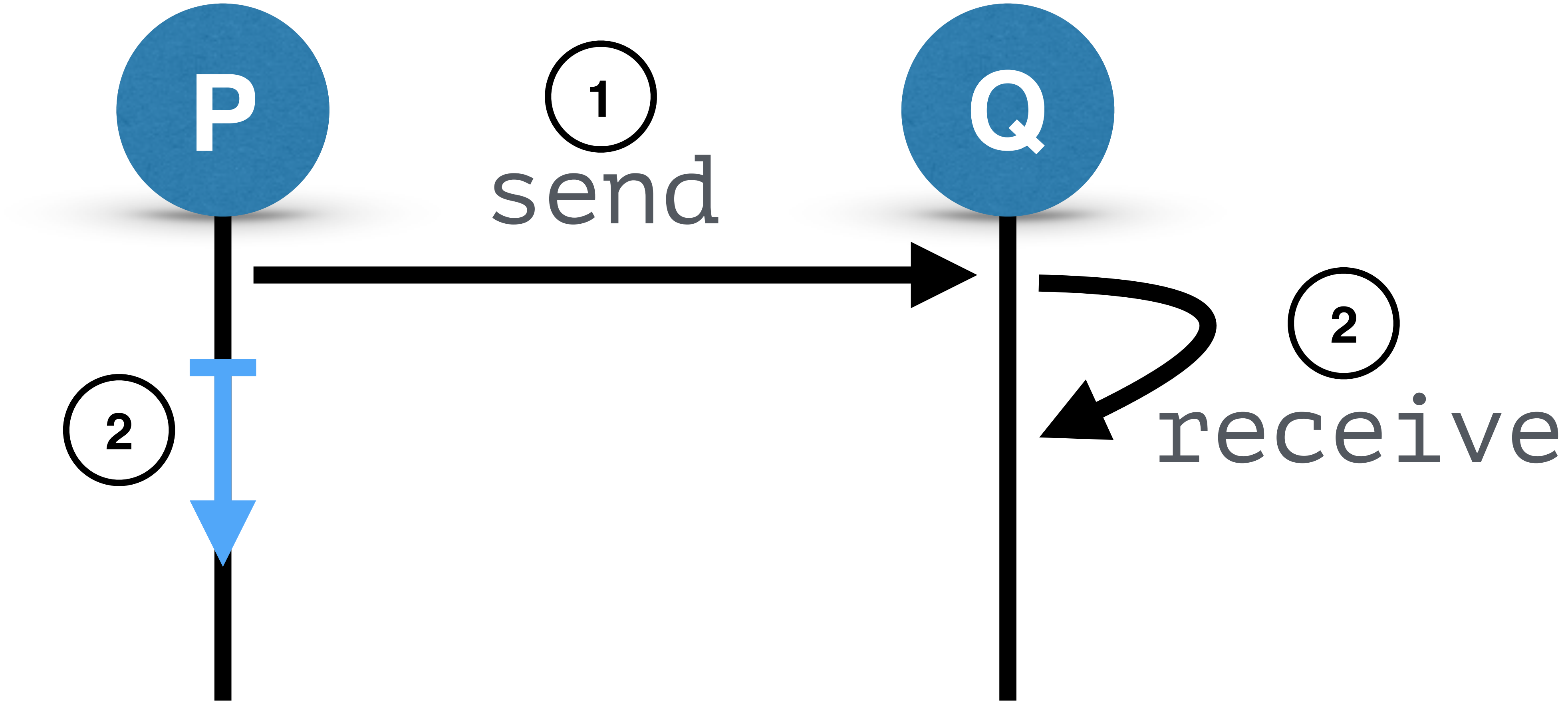
Blocking send



Logical Implementation

Synchronous communication

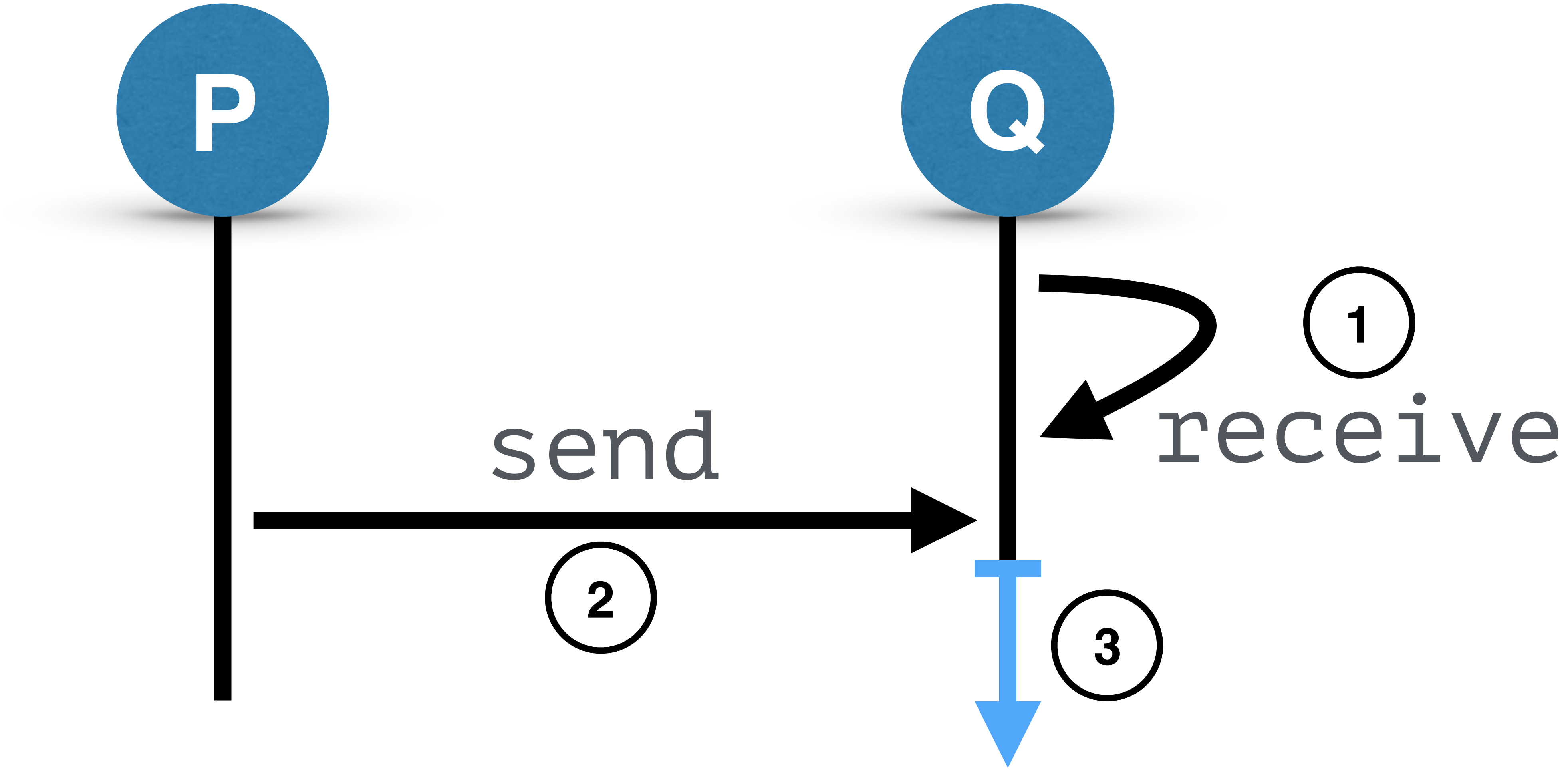
Nonblocking send



Logical Implementation

Synchronous communication

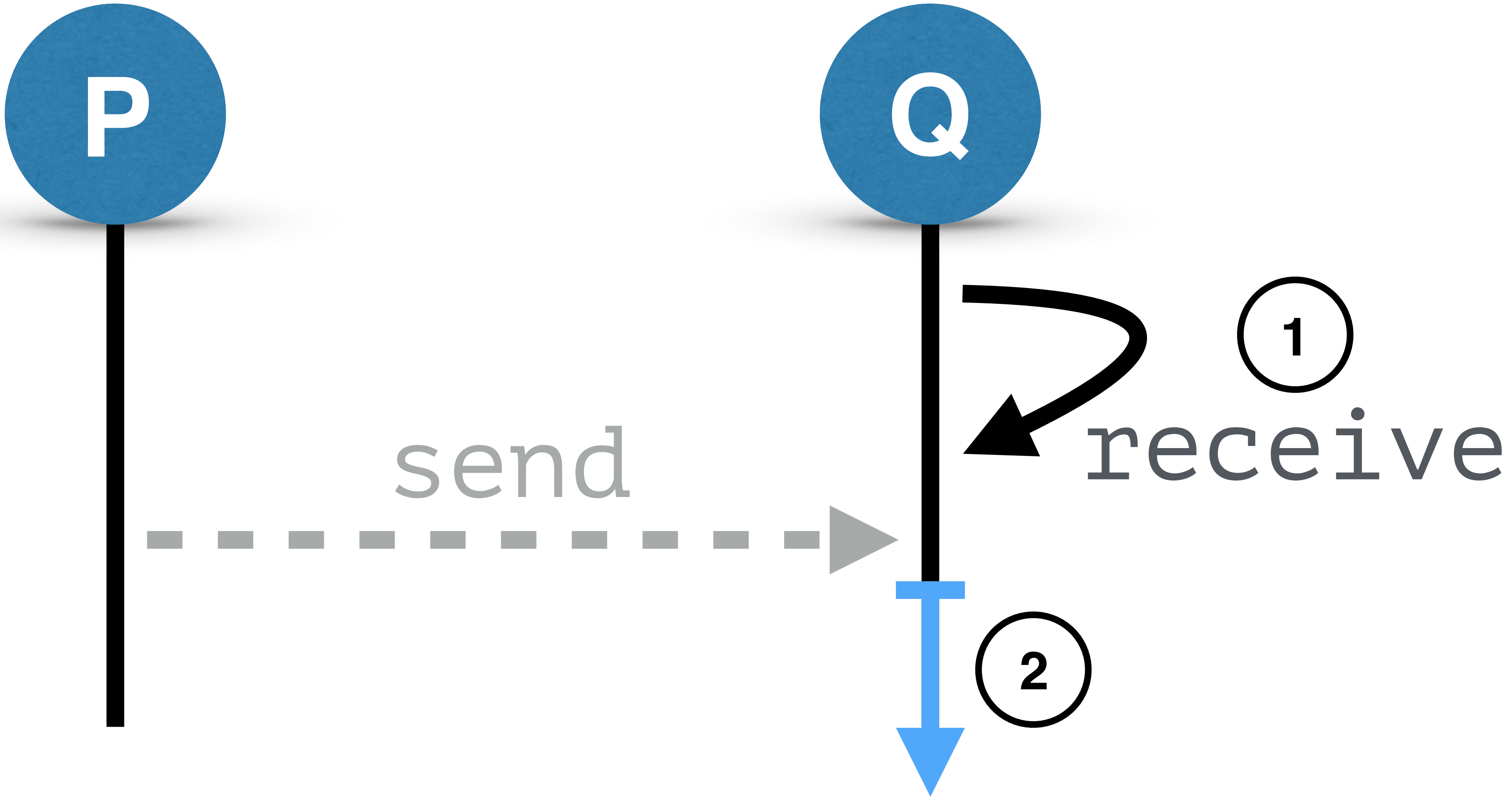
Blocking receive



Logical Implementation

Synchronous communication

Nonblocking receive



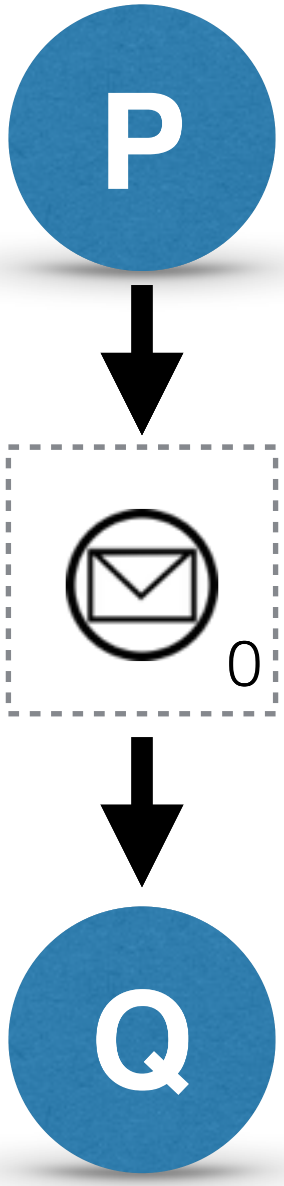
Logical Implementation

Buffering

Logical Implementation

Buffering

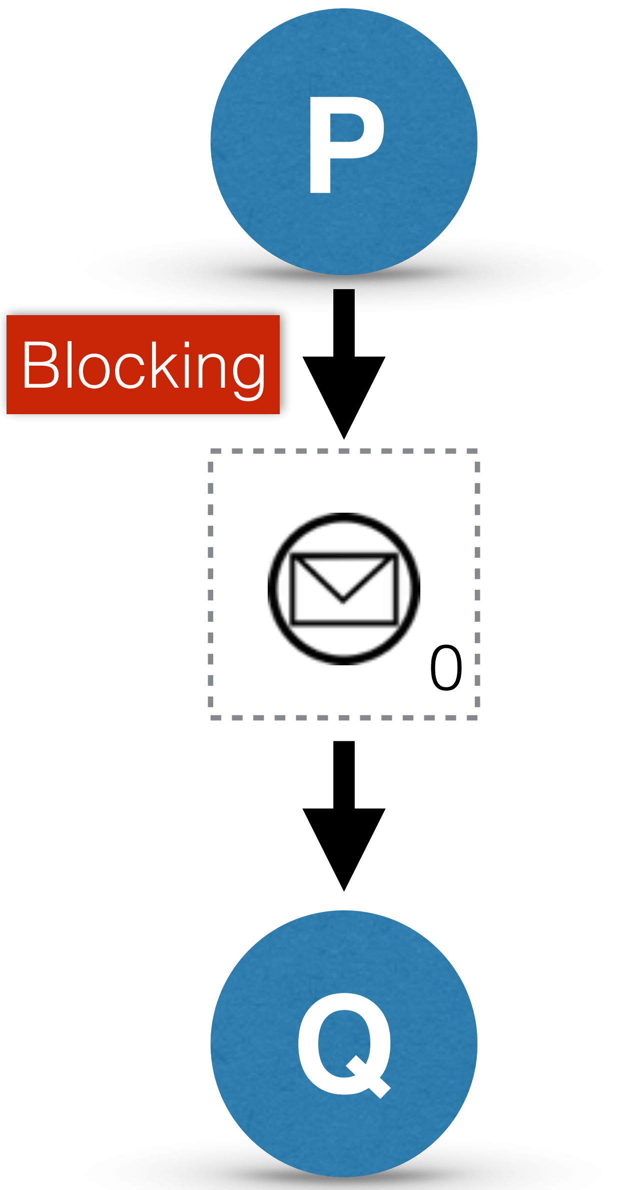
Zero Capacity



Logical Implementation

Buffering

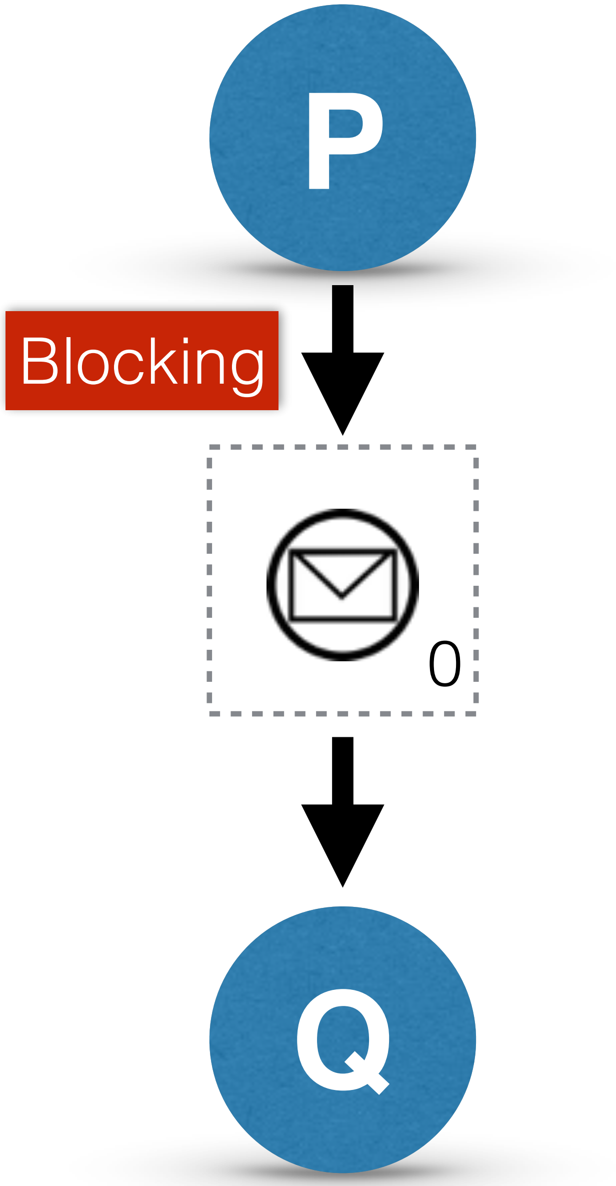
Zero Capacity



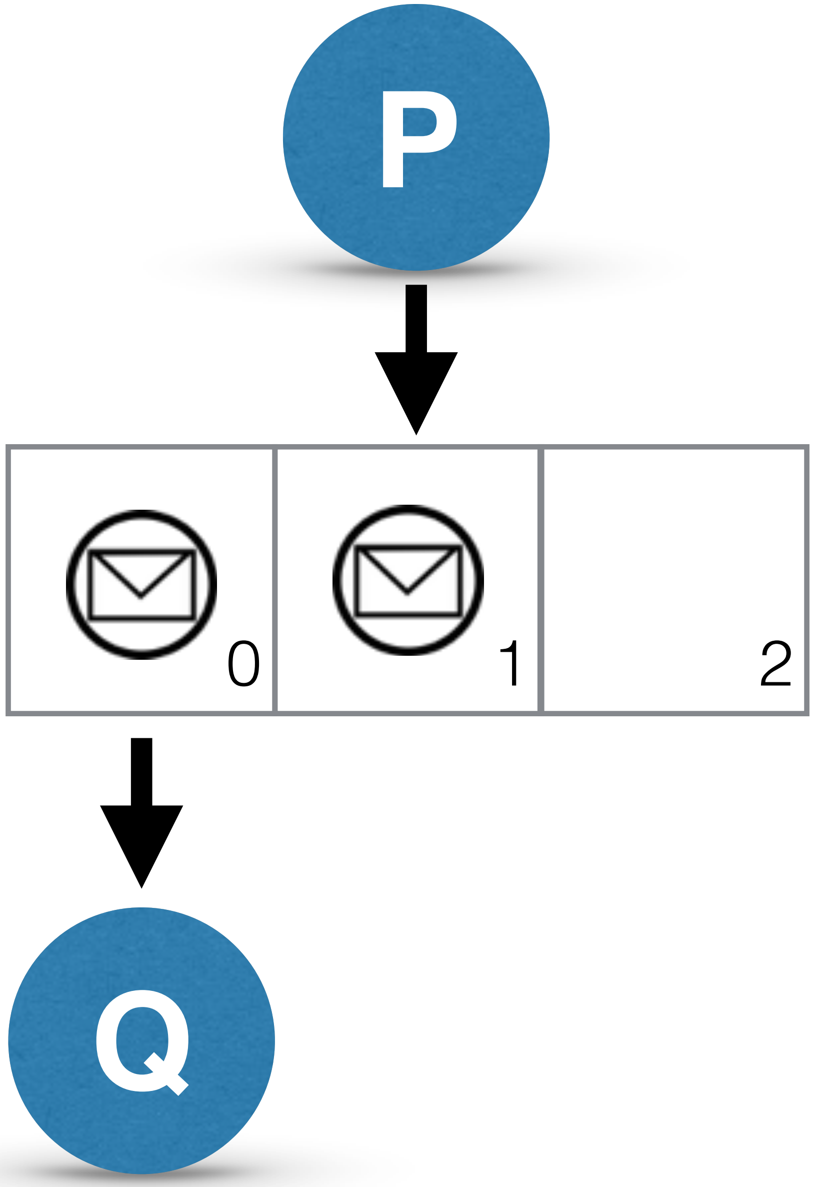
Logical Implementation

Buffering

Zero Capacity



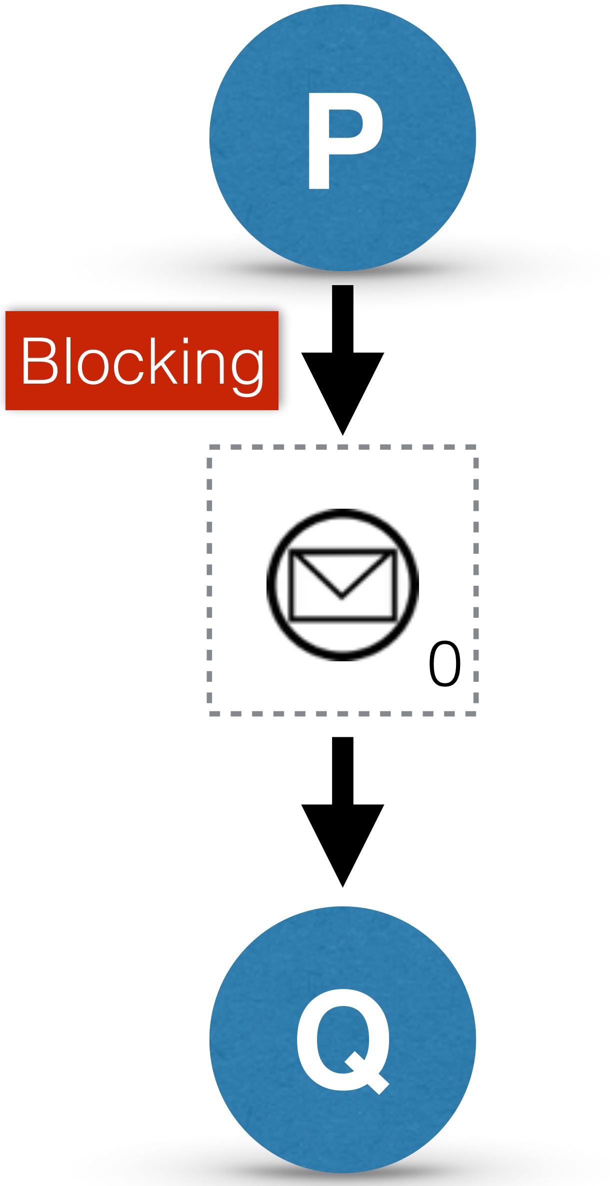
Bounded Capacity



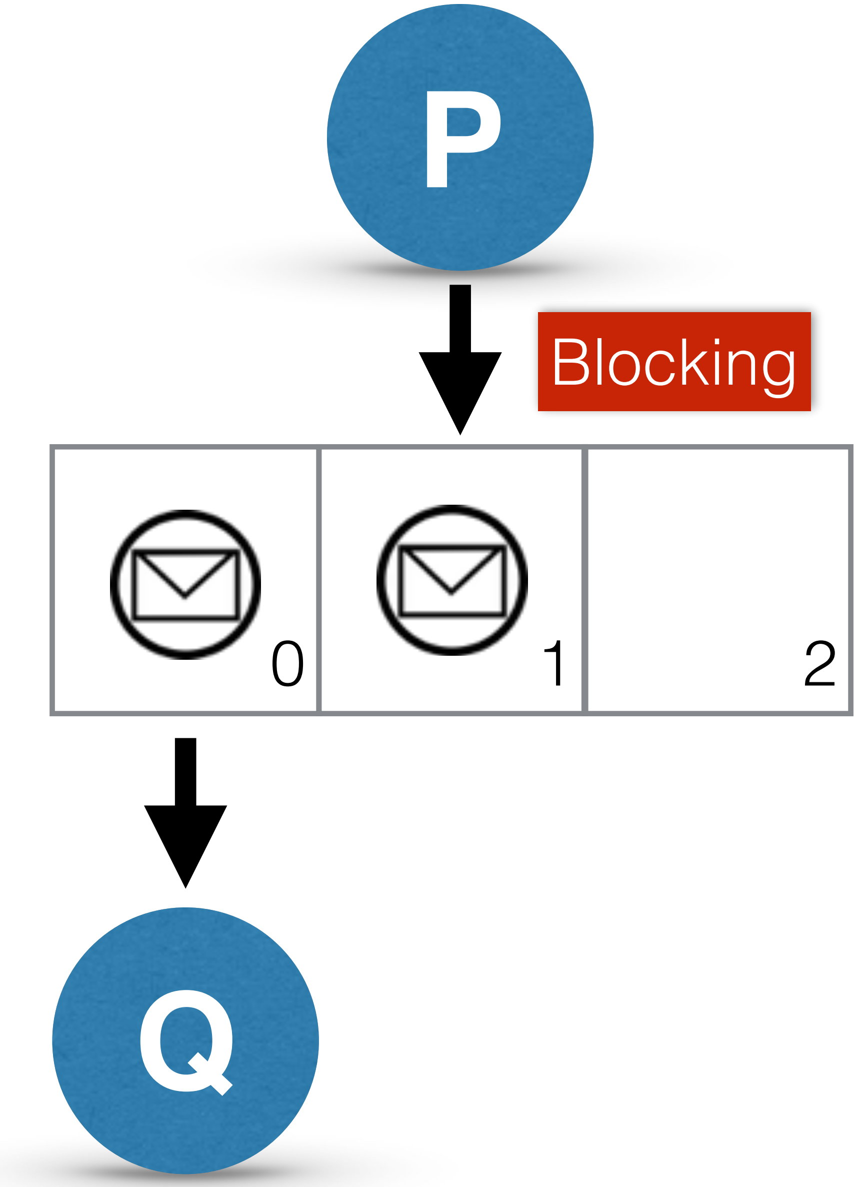
Logical Implementation

Buffering

Zero Capacity

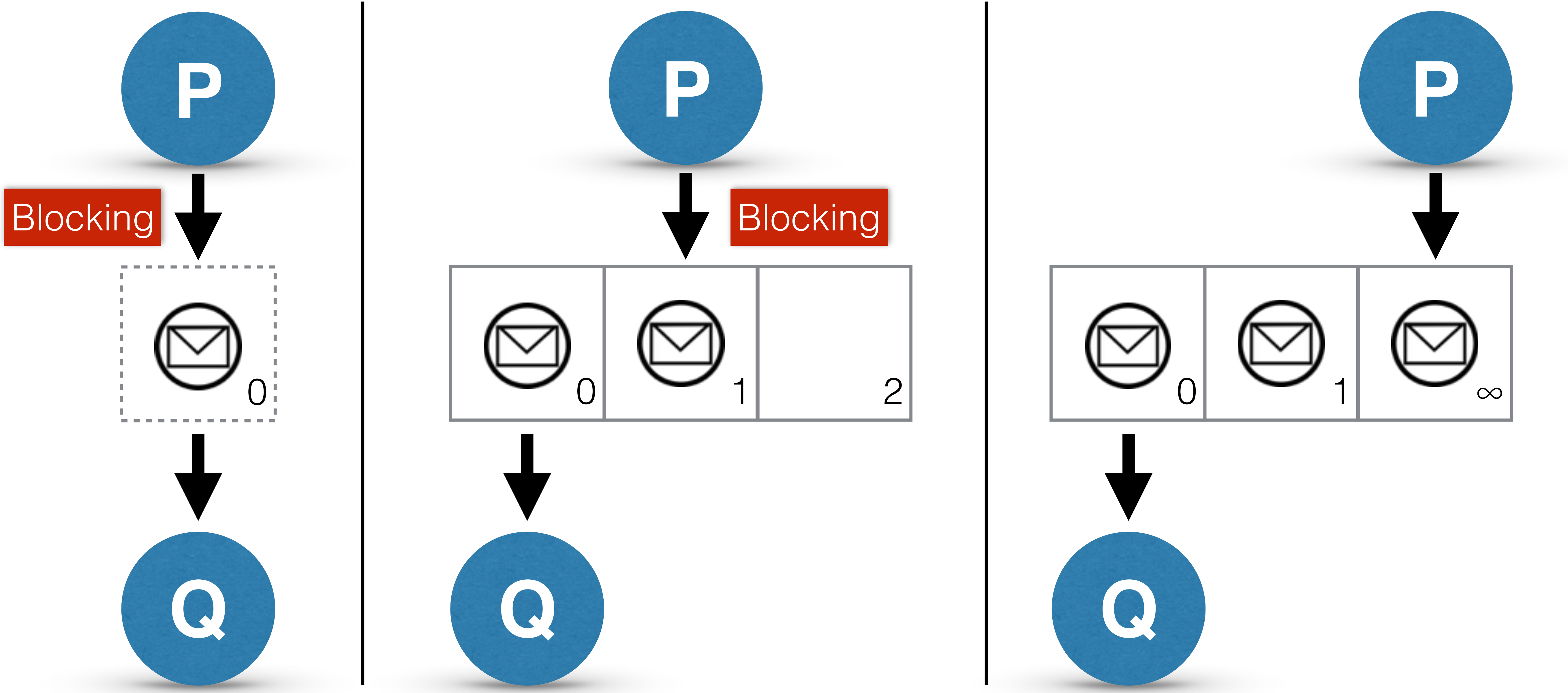


Bounded Capacity



Logical Implementation

Buffering



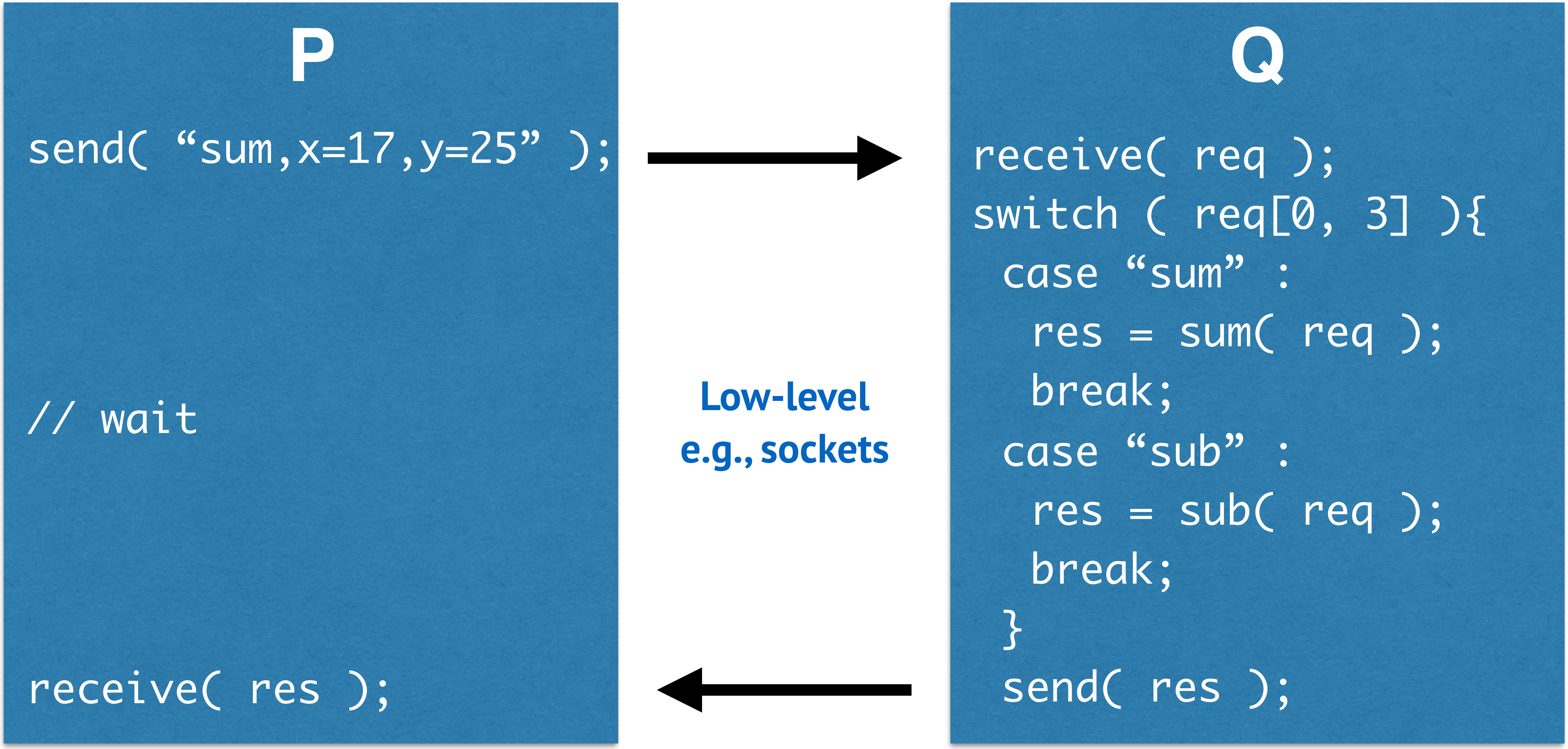
Zero Capacity

Bounded Capacity

Unbounded Capacity

Remote Invocation

Request-reply protocols



P

```
send( “sum,x=17,y=25” );

// wait

receive( res );
```

Q

```
receive( req );
switch ( req[0, 3] ){
  case “sum” :
    res = sum( req );
    break;
  case “sub” :
    res = sub( req );
    break;
  }
  send( res );
```

Low-level
e.g., sockets

Remote Invocation

Sockets

P

```
try {
  /* make connection to server socket */
  Socket toServer = new Socket( “127.0.0.1”, 6013 );
  PrintWriter pout = new PrintWriter( toServer.getOutputStream(), true );
  /* write the request to the server */
  pout.println( “sum,x=17,y=25” );
  toServer.close();
  /* accept response connection from server */
  toMe = new ServerSocket( 6012 );
  toMe.accept();
  InputStream in = toMe.getInputStream();
  BufferedReader bin = new BufferedReader( new InputStreamReader( in ) );
  /* read the data from the socket */
  String response = bin.readLine()
  /* close the socket connection */
  toMe.close();
} catch (IOException ioe) { System.err.println(ioe) };
```

Remote Invocation

Request-reply protocols

- low-level support for requesting the execution of a remote operation;
- support for RPC and RMI, discussed below

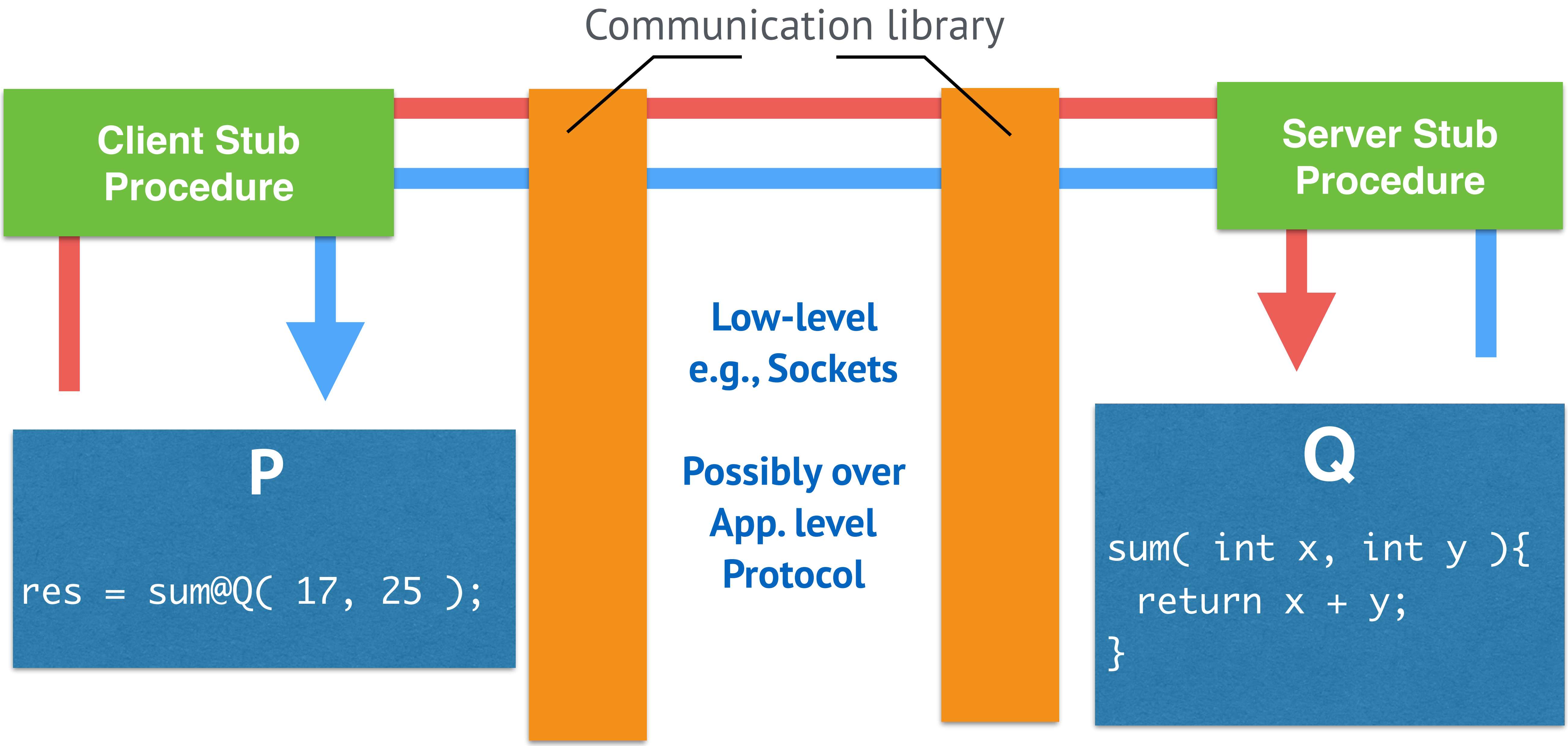
Remote Invocation

Request-reply protocols

- low-level support for requesting the execution of a remote operation (HTTP, FTP, etc. are Request-reply protocols);
- support for RPC and RMI (next);

Remote Invocation

Remote Procedure Calls



```
res = sum@Q( 17, 25 );
```

```
sum( int x, int y ){
  return x + y;
}
```

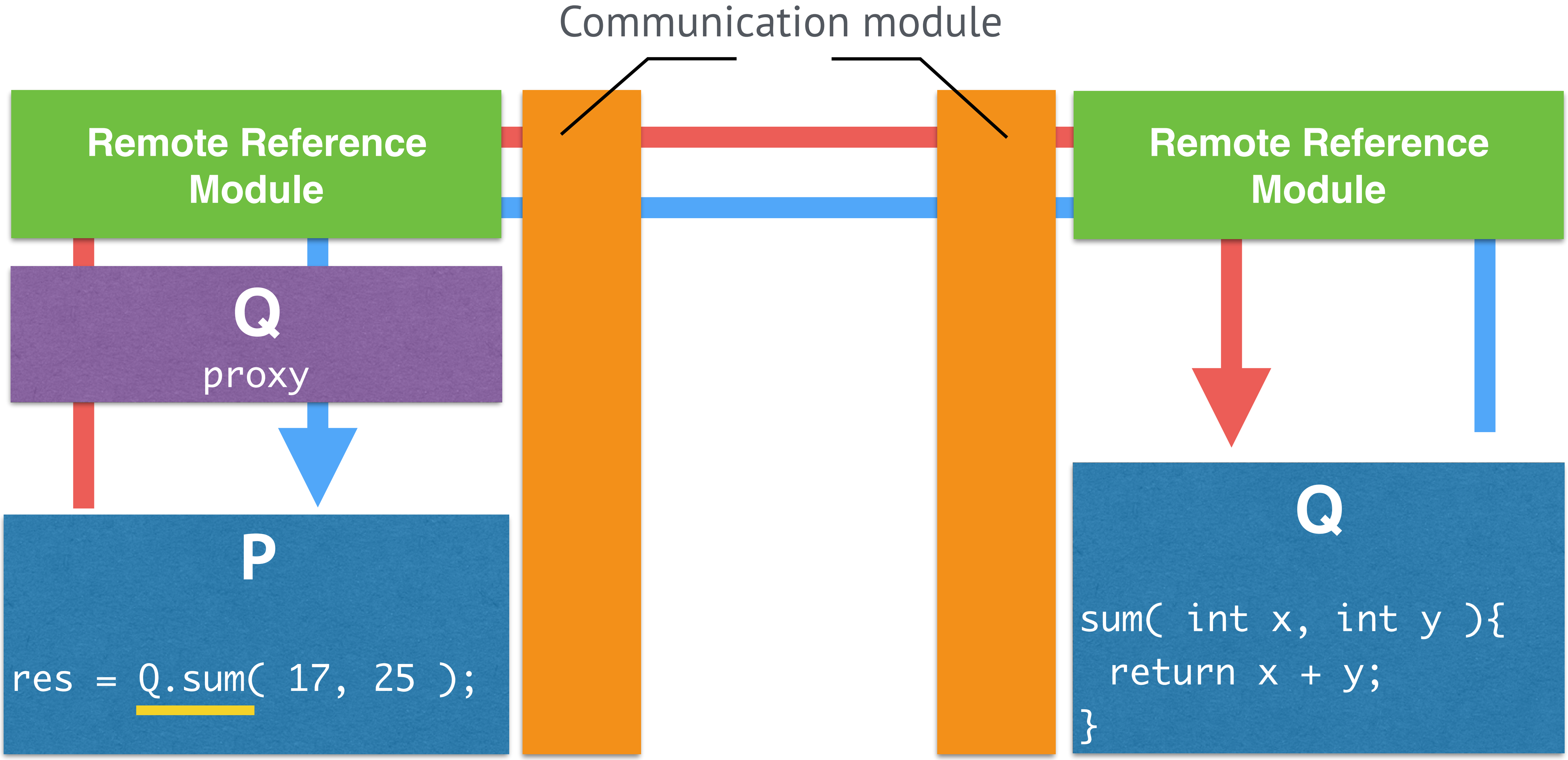
Remote Invocation

Remote Procedure Calls

- programming with **interfaces** (recall, an interface specifies the procedures and the variables available to others);
- Separation of concerns: interfaces remain the same but their implementation may change;
- High degree of heterogeneity.

Remote Invocation

Remote Method Invocation



```
res = Q.sum( 17, 25 );
```

```
sum( int x, int y ){
  return x + y;
}
```

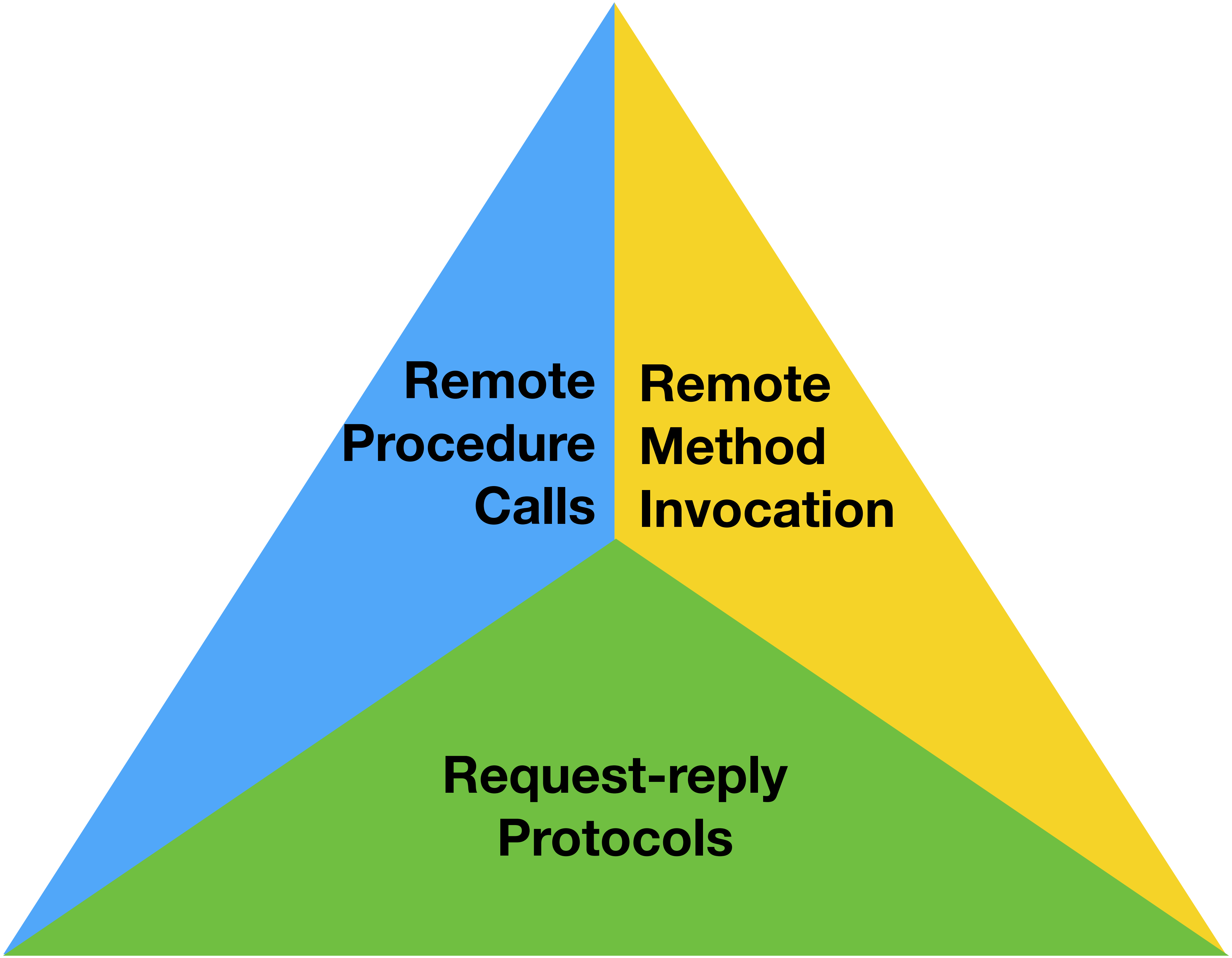
Remote Invocation

Remote Method Invocation

- Full object-oriented paradigm for programming distributed systems;

- Strictly Java.

Remote Invocation


Remote Procedure Calls
Remote Method Invocation
Request-reply Protocols