SANDER EVERS

ASYNCHROON PROGRAMMEREN

VAN AJAX TOT AWAIT

ASYNCHROON PROGRAMMEREN:

coöperatief multitasken binnen één thread

GESCHIEDENIS

1999	IE5: XMLHTTP ActiveX control
2005	XmlHttpRequest W3C standard, “AJAX”
2012	C# 5.0: async/await
2015	Python 3.5: async/await
	ES2015: Promise API
2017	ES2017: async/await

ONE MACHINE, ONE THREAD

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
0    0        let pos_x = 0, pos_y = 0;
              let v_x = 100, v_y = 100;
              let a_x =   0, a_y =  -9.81;

              while (pos_y >= 0) {
                  v_x += a_x;
                  v_y += a_y;
                  pos_x += v_x;
                  pos_y += v_y;
              }

              console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
  0    0
100  100
```

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
  0    0
100  100
  0  -9.81
```

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
  0    0
100  100
  0   -9.81
```

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
  0    0
100  100
  0   -9.81
```

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
  0    0
100  90.19
  0  -9.81
```

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
100    0
100  90.19
  0  -9.81
```

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
100   90.19   let pos_x = 0, pos_y = 0;
100   90.19   let v_x = 100, v_y = 100;
  0   -9.81   let a_x =   0, a_y =  -9.81;

              while (pos_y >= 0) {
                  v_x += a_x;
                  v_y += a_y;
                  pos_x += v_x;
                  pos_y += v_y;
              }

              console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
100   90.19    let pos_x = 0, pos_y = 0;
100   90.19    let v_x = 100, v_y = 100;
  0   -9.81    let a_x =   0, a_y =  -9.81;

               while (pos_y >= 0) {
                   v_x += a_x;
                   v_y += a_y;
                   pos_x += v_x;
                   pos_y += v_y;
               }

               console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
100   90.19   let pos_x = 0, pos_y = 0;
100   90.19   let v_x = 100, v_y = 100;
  0   -9.81   let a_x =   0, a_y =  -9.81;

              while (pos_y >= 0) {
                  v_x += a_x;
                  v_y += a_y;
                  pos_x += v_x;
                  pos_y += v_y;
              }

              console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
100   90.19    let pos_x = 0, pos_y = 0;
100   80.38    let v_x = 100, v_y = 100;
  0   -9.81    let a_x =   0, a_y =  -9.81;

               while (pos_y >= 0) {
                   v_x += a_x;
                   v_y += a_y;
                   pos_x += v_x;
                   pos_y += v_y;
               }

               console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
200   90.19   let pos_x = 0, pos_y = 0;
100   80.38   let v_x = 100, v_y = 100;
  0   -9.81   let a_x =   0, a_y =  -9.81;

              while (pos_y >= 0) {
                  v_x += a_x;
                  v_y += a_y;
                  pos_x += v_x;
                  pos_y += v_y;
              }

              console.log(pos_x);
```

ONE MACHINE, ONE THREAD

```
200 170.57    let pos_x = 0, pos_y = 0;
100  80.38    let v_x = 100, v_y = 100;
  0  -9.81    let a_x =   0, a_y =  -9.81;

              while (pos_y >= 0) {
                  v_x += a_x;
                  v_y += a_y;
                  pos_x += v_x;
                  pos_y += v_y;
              }

              console.log(pos_x);
```

MULTI-PROCESSING

```
600 393.99
100  31.33
  0  -9.81
```

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

```
384 121.14
128  30.57
  0  -9.81
```

```
let pos_x = 0, pos_y = 0;
let v_x = 100, v_y = 100;
let a_x =   0, a_y =  -9.81;

while (pos_y >= 0) {
    v_x += a_x;
    v_y += a_y;
    pos_x += v_x;
    pos_y += v_y;
}

console.log(pos_x);
```

PROCESS = { STACK, INSTRUCTION_POINTER }

MULTI-THREADING

```
600 393.99
384 121.14
```

```
            pos_x1 = 0, pos_y1 = 0;
100  31.33  let v_x = 100, v_y = 100;
  0  -9.81  let a_x =   0, a_y =  -9.81;

            while (pos_y1 >= 0) {
                v_x += a_x;
                v_y += a_y;
                pos_x1 += v_x;
                pos_y1 += v_y;
                if (pos_x1 == pos_x2
                        && pos_y1 == pos_y2)
                    console.log('💥');
            }

            console.log(pos_x1);
```

```
            pos_x2 = 0, pos_y2 = 0;
128  30.57  let v_x = 100, v_y = 100;
  0  -9.81  let a_x =   0, a_y =  -9.81;

            while (pos_y2 >= 0) {
                v_x += a_x;
                v_y += a_y;
                pos_x2 += v_x;
                pos_y2 += v_y;
                if (pos_x1 == pos_x2
                        && pos_y1 == pos_y2)
                    console.log('💥');
            }

            console.log(pos_x2);
```

PROCESS = { HEAP, THREADS: [{STACK, INSTRUCTION_POINTER}] }

TWEE THREADS OP 1 PROCESSOR

```
while ...
v_x += a_x;
v_y += a_y;
pos_x1 += v_x;
pos_y1 += v_y;
if ...
                <context switch>
                                        while ...
                                        v_x += a_x;
                                        v_y += a_y;
                                        pos_x2 += v_x;
                                        pos_y2 += v_y;
                                        if ...
                <context switch>
while ...
v_x += a_x;
v_y += a_y;
pos_x1 += v_x;
pos_y1 += v_y;
if ...
```

PRE-EMPTIVE MULTITHREADING/-PROCESSING

```
while ...
v_x += a_x;
v_y += a_y;
pos_x1 += v_x;
            <context switch>
                              while ...
                              v_x += a_x;
            <context switch>
pos_y1 += v_y;
if ...
while ...
v_x += a_x;
            <context switch>
                              v_y += a_y;
                              pos_x2 += v_x;
                              pos_y2 += v_y;
            <context switch>
v_y += a_y;
pos_x1 += v_x;
pos_y1 += v_y;
if ...
```

TERUG NAAR 2018

NETWORK THREAD   APP THREAD   DB THREAD

```
...
```

```
function handlePOST(req,res) {
    let text = req.body_sync();

}
```

NETWORK THREAD

```
...

buf += get_chunk();
buf += get_chunk();
buf += get_chunk();
```

<context switch>

APP THREAD

```
function handlePOST(req,res) {
    let text = req.body_sync();







}
```

DB THREAD

NETWORK THREAD APP THREAD DB THREAD

```
...                           function handlePOST(req,res) {
                                  let text = req.body_sync();
                  <context switch>
buf += get_chunk();
buf += get_chunk();
buf += get_chunk();
                  <context switch>
                                  let entity = parse(text);
...                               let id = db.insert_sync(entity);








                              }
```

NETWORK THREAD

APP THREAD

DB THREAD

```
...                     function handlePOST(req,res) {
                            let text = req.body_sync();
             <context switch>
buf += get_chunk();
buf += get_chunk();
buf += get_chunk();
             <context switch>
                            let entity = parse(text);
...                         let id = db.insert_sync(entity);
                                                   <context switch>
                                                                 ...
                                                                 acquire_table_lock();

                                                                 write_record(...);
                                                                 release_table_lock();







                        }
```

NETWORK THREAD APP THREAD DB THREAD

```
...           function handlePOST(req,res) {
                  let text = req.body_sync();
       <context switch>
buf += get_chunk();
buf += get_chunk();
buf += get_chunk();
       <context switch>
                  let entity = parse(text);
...               let id = db.insert_sync(entity);
                                    <context switch>
                                          ...
                                          acquire_table_lock();

                                          write_record(...);
                                          release_table_lock();
                                    <context switch>
                  entity.id = id;
                  res.text_sync(entity.json());


              }
```

NETWORK THREAD

APP THREAD

DB THREAD

```
...                     function handlePOST(req,res) {
                            let text = req.body_sync();
             <context switch>
buf += get_chunk();
buf += get_chunk();
buf += get_chunk();
             <context switch>
                            let entity = parse(text);
...                         let id = db.insert_sync(entity);
                                                   <context switch>
                                                                 ...
                                                                 acquire_table_lock();

                                                                 write_record(...);
                                                                 release_table_lock();
                                                   <context switch>
                            entity.id = id;
                            res.text_sync(entity.json());
             <context switch>
                            res.close();
                        }
```

NETWORK THREAD

```
buf += get_chunk();
buf += get_chunk();
buf += get_chunk();




write_chunk();
write_chunk();


write_end();
```

APP THREAD

```
function handlePOST() {
    req.body_async(handleBody);
    console.log('Started reading');
}

function handleBody(text) {
    entity = parse(text);
    db.insert_async(entity, handleId);
    console.log('Started insert');
}

function handleId(id) {
    entity.id = id;
    let json = entity.json();
    res.text_async(json, handleDone);
}

function handleDone()
    res.close();
}
```

DB THREAD

```
...
acquire_table_lock();
write_record(...);
release_table_lock();
```

FRAMEWORK CODE

```
class Request {
    function body_async(callback) {
        ...<magic return>...
        callback(text);
    }
}

class DatabaseConnection {
    function insert_async(entity, callback) {
        ...<magic return>...
        callback(id);
    }
}

class Response {
    function text_async(text, callback) {
        ...<magic return>...
        callback();
    }

    function close_async(callback) {...}
}
```

APP CODE

```
var req, res, entity, id;

function handlePOST() {
    req.body_async(handleBody);
}

function handleBody(text) {
    entity = parse(text);
    db.insert_async(entity, handleId);
}

function handleId(id) {
    entity.id = id;
    let json = entity.json();
    res.text_async(json, handleDone);
}

function handleDone()
    res.close_async();
}
```

FRAMEWORK CODE

```
class Request {
    function body_async(callback) {
        ...<magic return>...
        callback(text);
    }
}

class DatabaseConnection {
    function insert_async(entity, callback) {
        ...<magic return>...
        callback(id);
    }
}

class Response {
    function text_async(text, callback) {
        ...<magic return>...
        callback();
    }

    function close_async(callback) {...}
}
```

APP CODE

```
var req, res, entity, id;

function handlePOST() {
    req.body_async(handleBody);
}

function handleBody(text) {
    entity = parse(text);
    db.insert_async(entity, handleId);
}

function handleId(id) {
    entity.id = id;
    let json = entity.json();
    res.text_async(json, handleDone);
}

function handleDone()
    res.close_async();
}
```

FRAMEWORK CODE

```
class Request {
    function body_async(callback, ...args) {
        ...<magic return>...
        callback(text, ...args);
    }
}

class DatabaseConnection {
    function insert_async(entity, callback, ...args) {
        ...<magic return>...
        callback(id, ...args);
    }
}

class Response {
    function text_async(text, callback, ...args) {
        ...<magic return>...
        callback(...args);
    }

    function close_async(callback) {...}
}
```

APP CODE

```
// var req, res, entity, id;

function handlePOST(req, res) {
    req.body_async(handleBody, res);
}

function handleBody(text, res) {
    let entity = parse(text);
    db.insert_async(entity, handleId, res, entity);
}

function handleId(id, res, entity) {
    entity.id = id;
    let json = entity.json();
    res.text_async(json, handleDone, res);
}

function handleDone(res)
    res.close_async();
}
```

FRAMEWORK CODE

```
class Request {
    function body_async(callback) {
        ...<magic return>...
        callback(text);
    }
}

class DatabaseConnection {
    function insert_async(entity, callback) {
        ...<magic return>...
        callback(id);
    }
}

class Response {
    function text_async(text, callback) {
        ...<magic return>...
        callback();
    }

    function close_async(callback) {...}
}
```

APP CODE

```
// var req, res, entity, id;

function handlePOST(req, res) {
    req.body_async(function(text) {
        let entity = parse(text);
        db.insert_async(entity, function(id) {
            entity.id = id;
            let json = entity.json();
            res.text_async(json, function() {
                res.close_async();
            });
        });
    });
}
```

FRAMEWORK CODE

```
class Request {
    function body_async(callback) {
        ...<magic return>...
        callback(text);
    }
}

class DatabaseConnection {
    function insert_async(entity, callback) {
        ...<magic return>...
        callback(id);
    }
}

class Response {
    function text_async(text, callback) {
        ...<magic return>...
        callback();
    }

    function close_async(callback) {...}
}
```

APP CODE

```
// var req, res, entity, id;

function handlePOST(req, res, callback) {
    req.body_async(function(text) {
        let entity = parse(text);
        db.insert_async(entity, function(id) {
            entity.id = id;
            let json = entity.json();
            res.text_async(json, function() {
                res.close_async(callback);
            });
        });
    });
}
```

👷👍

FRAMEWORK CODE

```
class Request {
    function body_async() {
        return new Promise(function(resolve) {
           ... <async stuff> resolve(text);
        });
    }
}

class DatabaseConnection {
    function insert_async(entity) {
        return new Promise(function(resolve) {
           ... <async stuff> resolve(id);
        });
    }
}

class Response {
    function text_async(text) {
        return new Promise(function(resolve) {
           ... <async stuff> resolve();
        });
    }
}
```

APP CODE

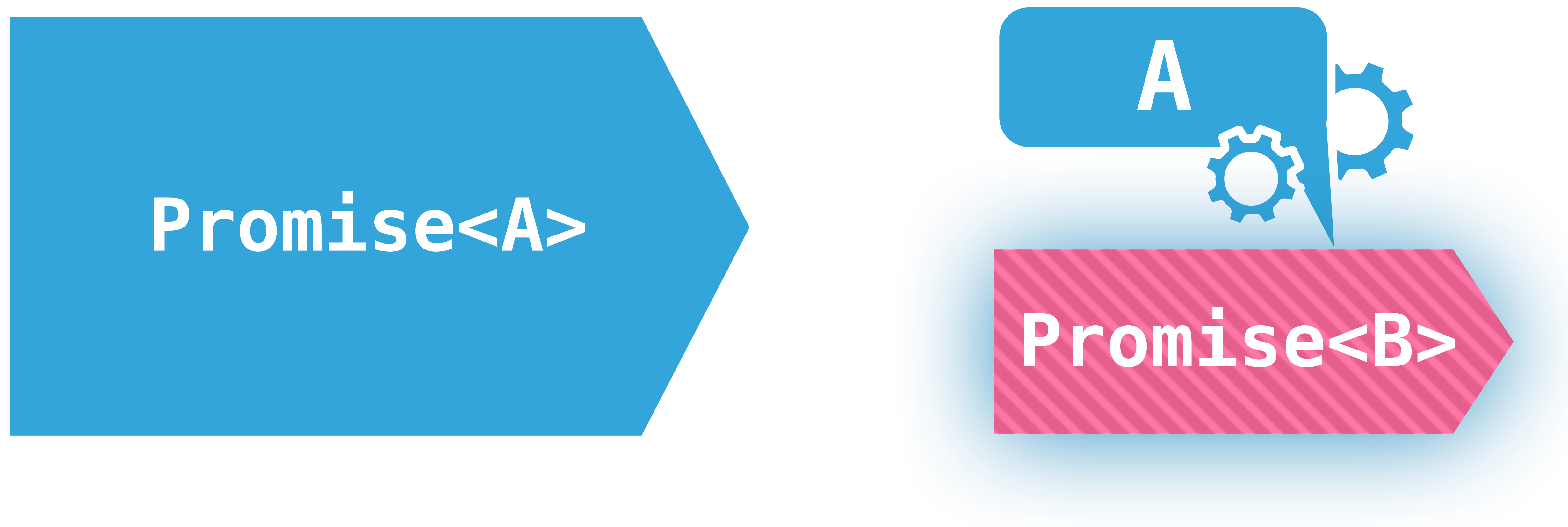
```
function handlePOST(req, res) {
    let entity;
    req.body_async().then(function(text) {
        entity = parse(text);
        return db.insert_async(entity);
    }).then(function(id) {
        entity.id = id;
        let json = entity.json();
        return res.text_async(json);
    }).then(function() {
        return res.close_async();
    });
}
```

PROMISE API

```
class Promise<A> {
    subscribers: Array<A=>any>

    function then(handler: A=>Promise<B>): Promise<B> {
        subscribe handler;
        return a new Promise<B>;
    }
}
```

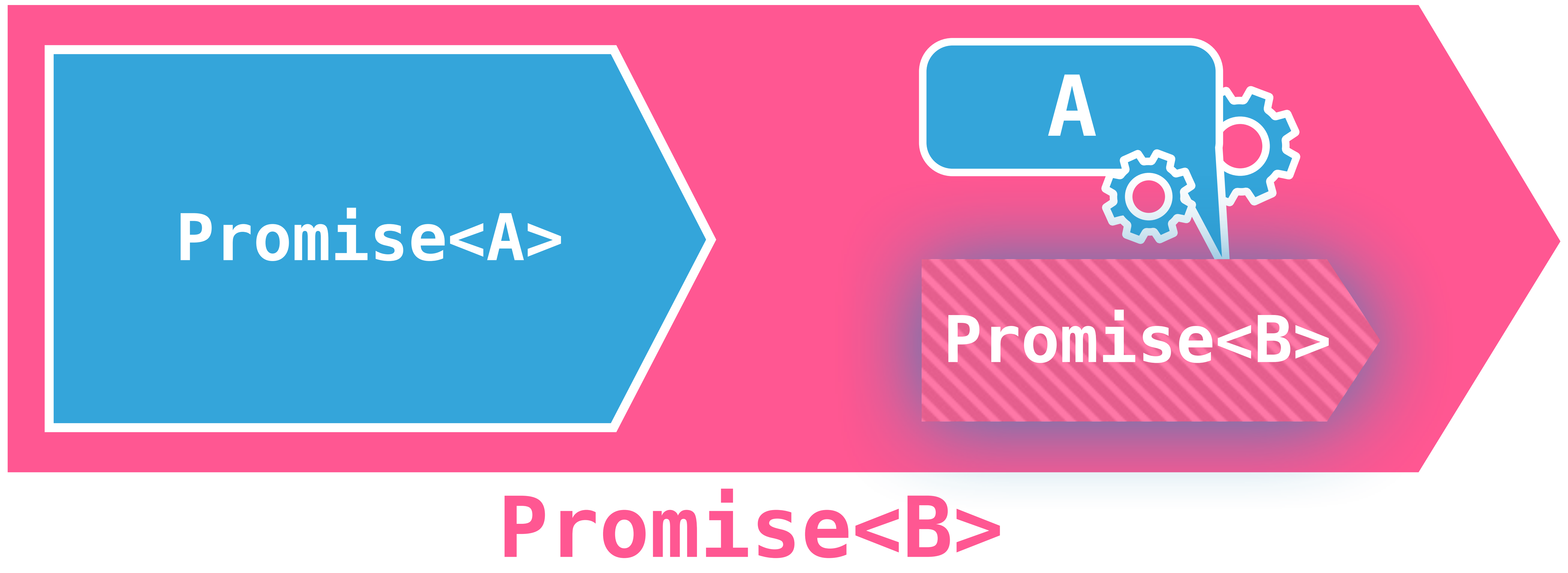
PROMISE API

```
class Promise<A> {
    subscribers: Array<A=>any>

    function then(handler: A=>Promise<B>): Promise<B> {
        subscribe handler;
        return a new Promise<B>;
    }
}
```



PROMISE API

```
class Promise<A> {
    subscribers: Array<A=>any>

    function then(handler: A=>Promise<B>): Promise<B> {
        subscribe handler;
        return a new Promise<B>;
    }
}
```



FRAMEWORK CODE

```
class Request {
    function body_async() { return textP }
}

class DatabaseConnection {
    function insert_async(entity) { return idP }
}

class Response {
    function text_async(text) { return doneP }

    function close_async() { return doneP }
}
```

APP CODE

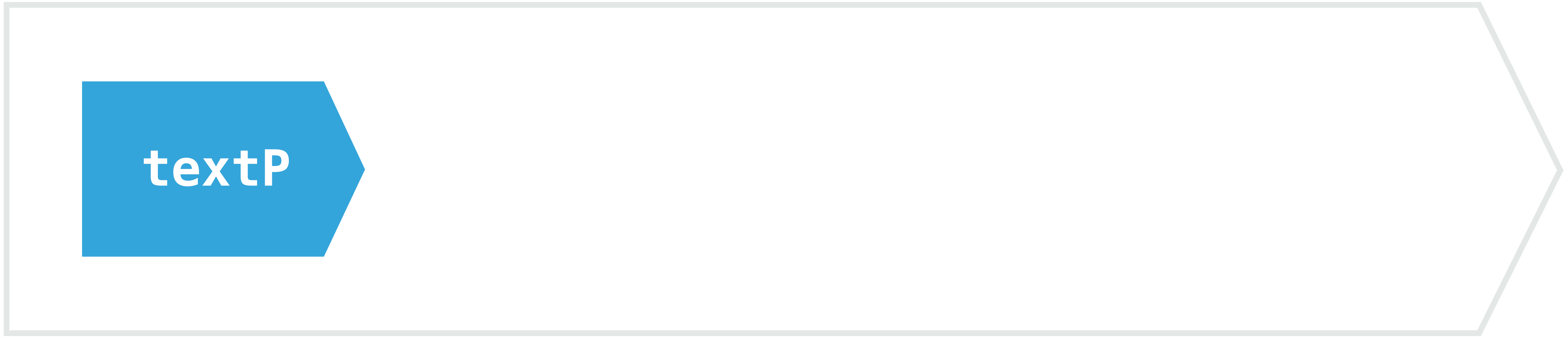
```
function handlePOST(req, res) {







}
```

FRAMEWORK CODE

```
class Request {
    function body_async() { return textP }
}

class DatabaseConnection {
    function insert_async(entity) { return idP }
}

class Response {
    function text_async(text) { return doneP }

    function close_async() { return doneP }
}
```

APP CODE

```
function handlePOST(req, res) {
    let entity;
    req.body_async();







}
```

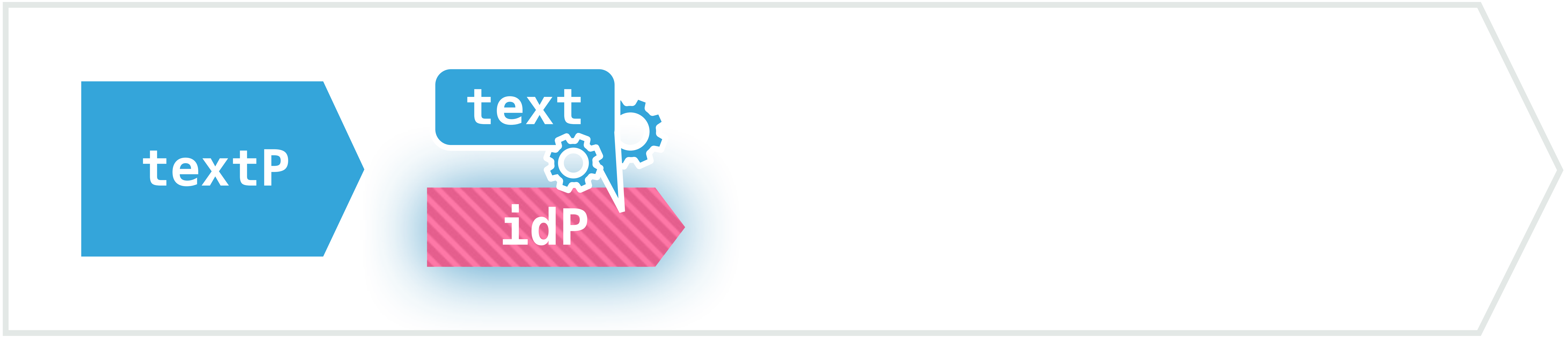


FRAMEWORK CODE

```
class Request {
    function body_async() { return textP }
}

class DatabaseConnection {
    function insert_async(entity) { return idP }
}

class Response {
    function text_async(text) { return doneP }

    function close_async() { return doneP }
}
```

APP CODE

```
function handlePOST(req, res) {
    let entity;
    req.body_async().then(function(text) {
        entity = parse(text);
        return db.insert_async(entity);
    });




}
```

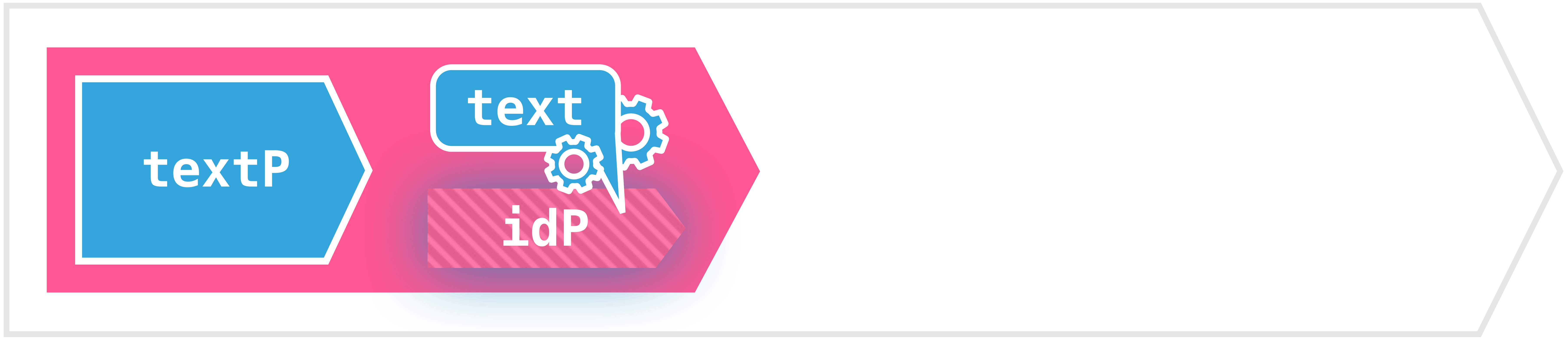


FRAMEWORK CODE

```
class Request {
    function body_async() { return textP }
}

class DatabaseConnection {
    function insert_async(entity) { return idP }
}

class Response {
    function text_async(text) { return doneP }

    function close_async() { return doneP }
}
```

APP CODE

```
function handlePOST(req, res) {
    let entity;
    req.body_async().then(function(text) {
        entity = parse(text);
        return db.insert_async(entity);
    });




}
```

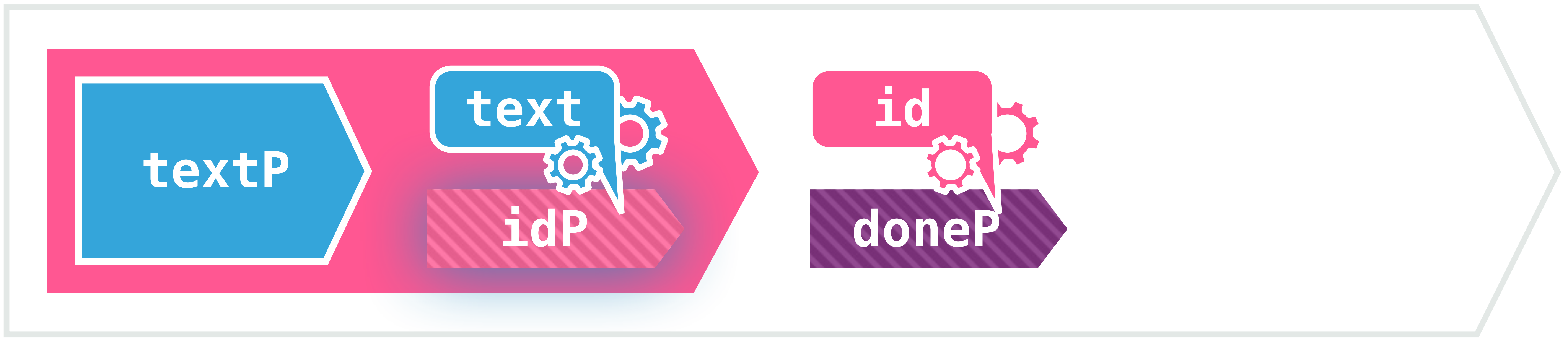


FRAMEWORK CODE

```
class Request {
    function body_async() { return textP }
}

class DatabaseConnection {
    function insert_async(entity) { return idP }
}

class Response {
    function text_async(text) { return doneP }

    function close_async() { return doneP }
}
```

APP CODE

```
function handlePOST(req, res) {
    let entity;
    req.body_async().then(function(text) {
        entity = parse(text);
        return db.insert_async(entity);
    }).then(function(id) {
        entity.id = id;
        let json = entity.json();
        return res.text_async(json);
    });


}
```

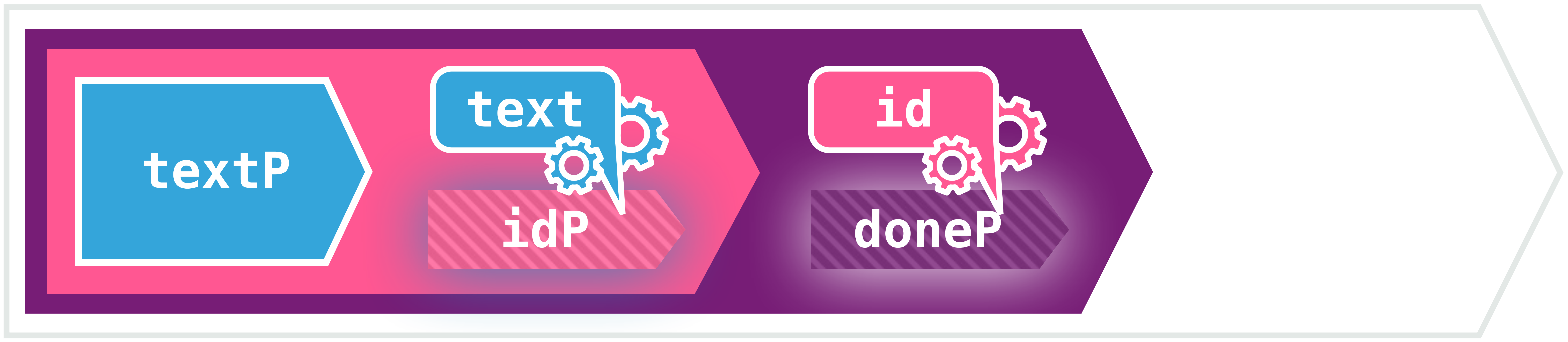


FRAMEWORK CODE

```
class Request {
    function body_async() { return textP }
}

class DatabaseConnection {
    function insert_async(entity) { return idP }
}

class Response {
    function text_async(text) { return doneP }

    function close_async() { return doneP }
}
```

APP CODE

```
function handlePOST(req, res) {
    let entity;
    req.body_async().then(function(text) {
        entity = parse(text);
        return db.insert_async(entity);
    }).then(function(id) {
        entity.id = id;
        let json = entity.json();
        return res.text_async(json);
    });


}
```

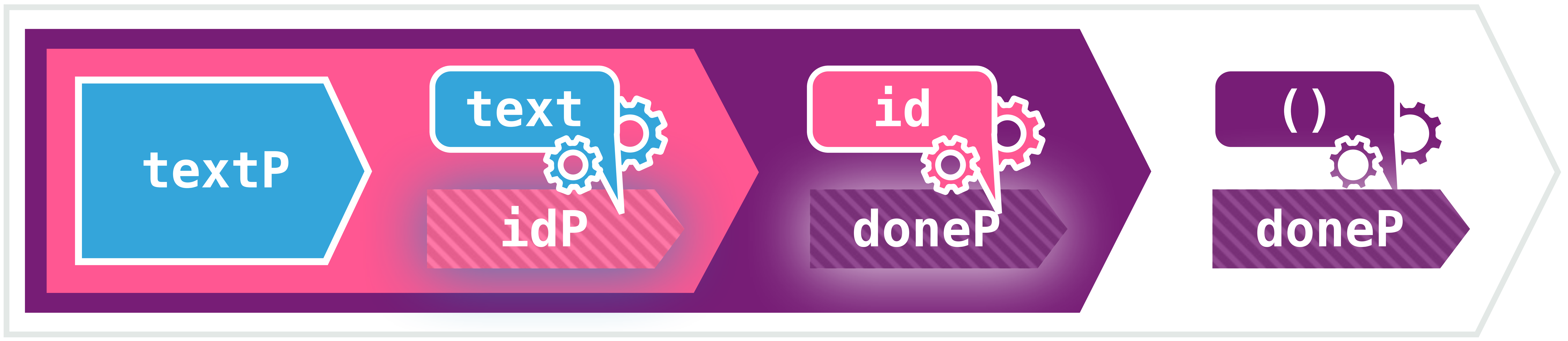


FRAMEWORK CODE

```
class Request {
    function body_async() { return textP }
}

class DatabaseConnection {
    function insert_async(entity) { return idP }
}

class Response {
    function text_async(text) { return doneP }

    function close_async() { return doneP }
}
```

APP CODE

```
function handlePOST(req, res) {
    let entity;
    req.body_async().then(function(text) {
        entity = parse(text);
        return db.insert_async(entity);
    }).then(function(id) {
        entity.id = id;
        let json = entity.json();
        return res.text_async(json);
    }).then(function() {
        return res.close_async();
    });
}
```

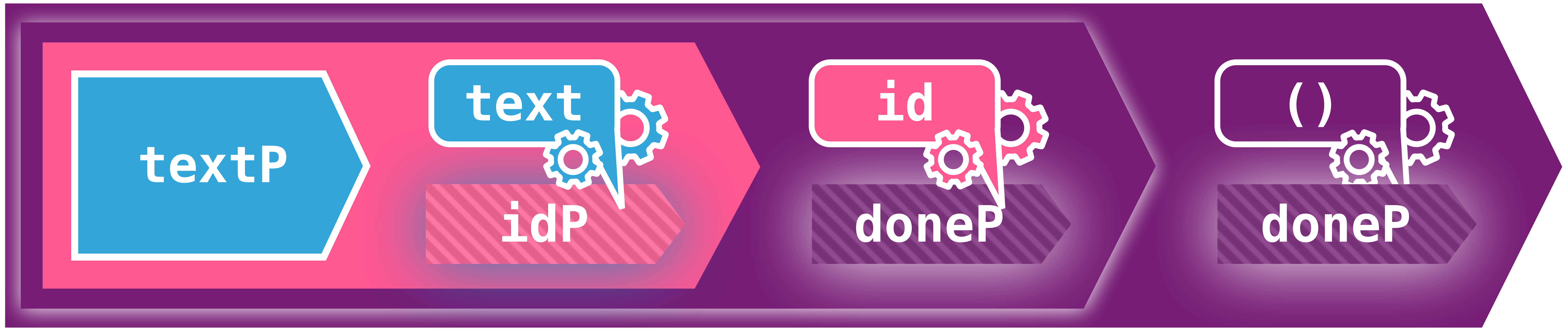


FRAMEWORK CODE

```
class Request {
    function body_async() { return textP }
}

class DatabaseConnection {
    function insert_async(entity) { return idP }
}

class Response {
    function text_async(text) { return doneP }

    function close_async() { return doneP }
}
```

APP CODE

```
function handlePOST(req, res) {
    let entity;
    req.body_async().then(function(text) {
        entity = parse(text);
        return db.insert_async(entity);
    }).then(function(id) {
        entity.id = id;
        let json = entity.json();
        return res.text_async(json);
    }).then(function() {
        return res.close_async();
    });
}
```

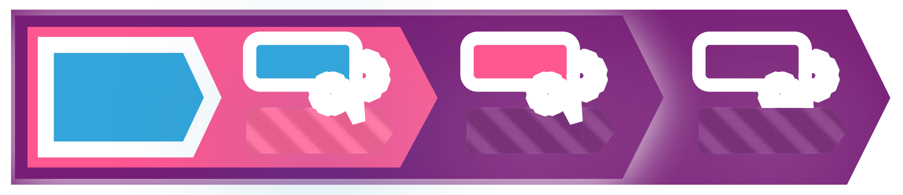


FRAMEWORK CODE

```
class Request {
    function body_async() { return textP }
}

class DatabaseConnection {
    function insert_async(entity) { return idP }
}

class Response {
    function text_async(text) { return doneP }

    function close_async() { return doneP }
}
```

APP CODE

```
function handlePOST(req, res) {
    let entity;
    let putP =
      req.body_async().then(function(text) {
          entity = parse(text);
          return db.insert_async(entity);
      }).then(function(id) {
          entity.id = id;
          let json = entity.json();
          return res.text_async(json);
      }).then(function() {
          return res.close_async();
      });

    return putP
}
```



👷‍♀️ 👍

PROMISE API

```
class Promise<A> {
    subscribers: Array<A=>any>
    rejection_subscribers: Array<E=>any>

    function then(handler: A=>Promise<B>): Promise<B> {
        subscribe handler;
        return a new Promise<B>;
    }

    function catch(handler: E=>Promise<A>): Promise<A> {
        subscribe handler to rejection;
        return a new Promise<A>;
    }
}
```

APP CODE

```
function handlePOST(req, res) {
    let entity;
    req.body_async().then(function(text) {
        entity = parse(text);
        return db.insert_async(entity);
    }).then(function(id) {
        entity.id = id;
        let json = entity.json();
        return res.text_async(json);
    }).catch(function(err) {
        res.set_status(500);
        return Promise.resolve();
    }).then(function() {
        return res.close_async();
    });
}
```

ASYNC / AWAIT

FRAMEWORK CODE

```
class Request {
    async function body_async() {
        text = await ...
        return text;
    }
}

class DatabaseConnection {
    async function insert_async(entity) {
        text = await ...
        return id;
    }
}

class Response {
    async function text_async(text) {
        await ...
    }
    async function close_async() {
        await ...
    }
}
```

APP CODE

```
async function handlePOST(req, res) {
    let text = await req.body_async();
    let entity = parse(text);
    let id = await db.insert_async(entity);
    entity.id = id;
    let json = entity.json();
    await res.text_async(json);
    await res.close_async();
}
```

ASYNC

```
async function foo() {
   return “foo”;
}
```

PROMISE

```
function foo() {
    return Promise.resolve(“foo”);
}
```

ASYNC

```
async function foo() {
   return “foo”;
}
```

```
async function bar() {
   let x = await foo();
   return x + “bar”;
}
```

PROMISE

```
function foo() {
    return Promise.resolve(“foo”);
}
```

```
function bar() {
    return foo().then(function(x) {
        return Promise.resolve(x + “bar”);
    }
}
```

ASYNC

✓

```
function parallel() {
   foo();
   foo();
   return “kicked off two tasks”;
}
```

PROMISE

```
function parallel() {
    foo();
    foo();
    return “kicked off two tasks”;
}
```

ASYNC

PROMISE

✓

```
function parallel() {
   foo();
   foo();
   return “kicked off two tasks”;
}
```

```
function parallel() {
    foo();
    foo();
    return “kicked off two tasks”;
}
```

✗

```
function forbidden() {
   let x = await foo();
   return x + “bar”;
}
```

```
function forbidden() {
    let promise = foo();
    let x = promise.wait_sync();
    return x + “bar”;
}
```

ASYNC

PROMISE

✓

```
function parallel() {
   foo();
   foo();
   return “kicked off two tasks”;
}
```

```
function parallel() {
    foo();
    foo();
    return “kicked off two tasks”;
}
```

✗

```
function forbidden() {
   let x = await foo();
   return x + “bar”;
}
```

```
function forbidden() {
    let promise = foo();
    let x = promise.wait_sync();
    return x + “bar”;
}
```

☠

```
async function evil() {
   let res = performLongCalculation();
   return res;
}
```

```
function evil() {
    return new Promise(function(resolve,reject) {
        resolve(performLongCalculation())
    });
}
```

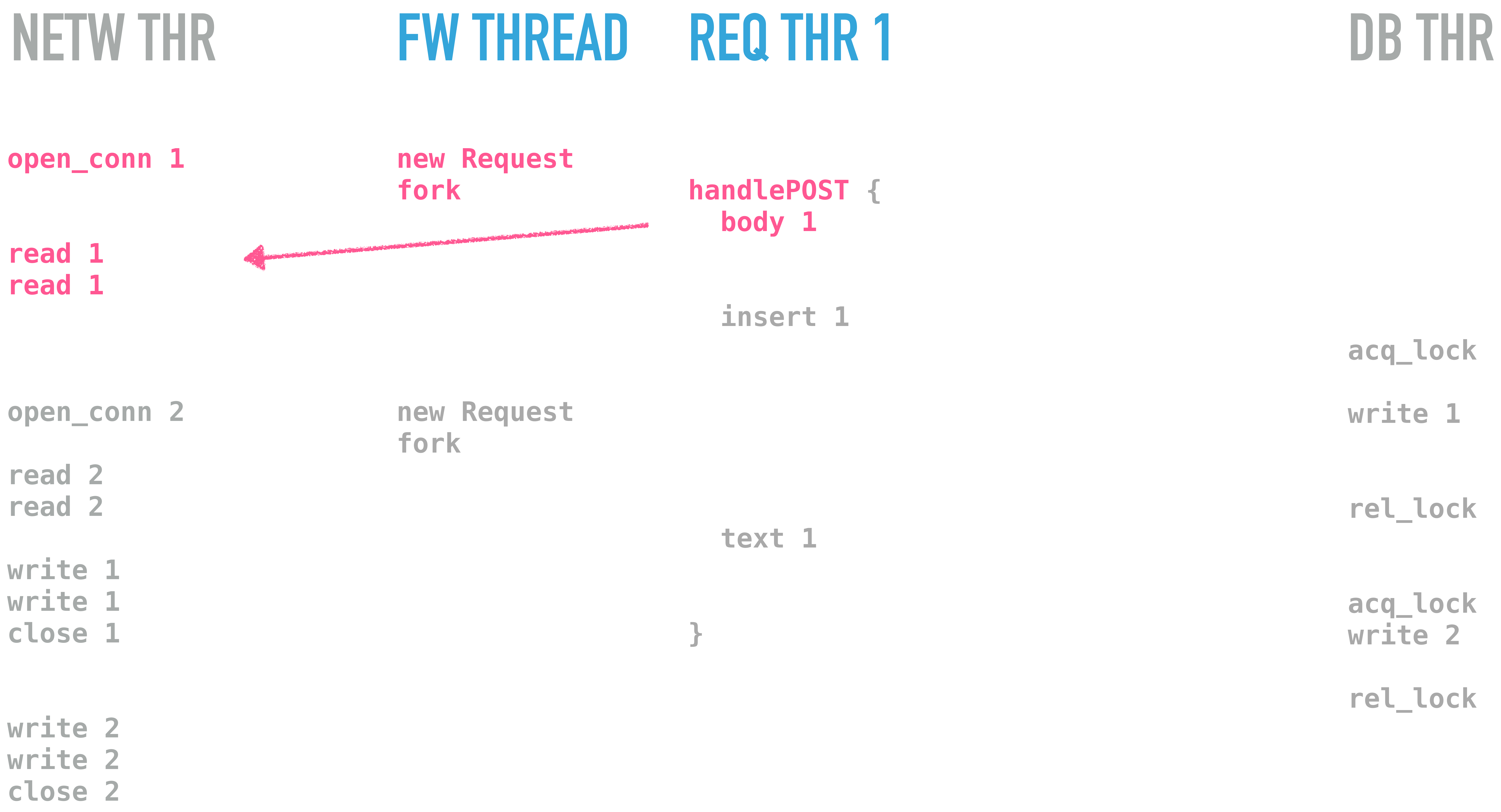
WAAROM??

NETW THR	FW THREAD	DB THR
open_conn 1	new Request	
	fork	
read 1		
read 1		
		acq_lock
open_conn 2	new Request	write 1
	fork	
read 2		
read 2		rel_lock
write 1		
write 1		acq_lock
close 1		write 2
		rel_lock
write 2		
write 2		
close 2		

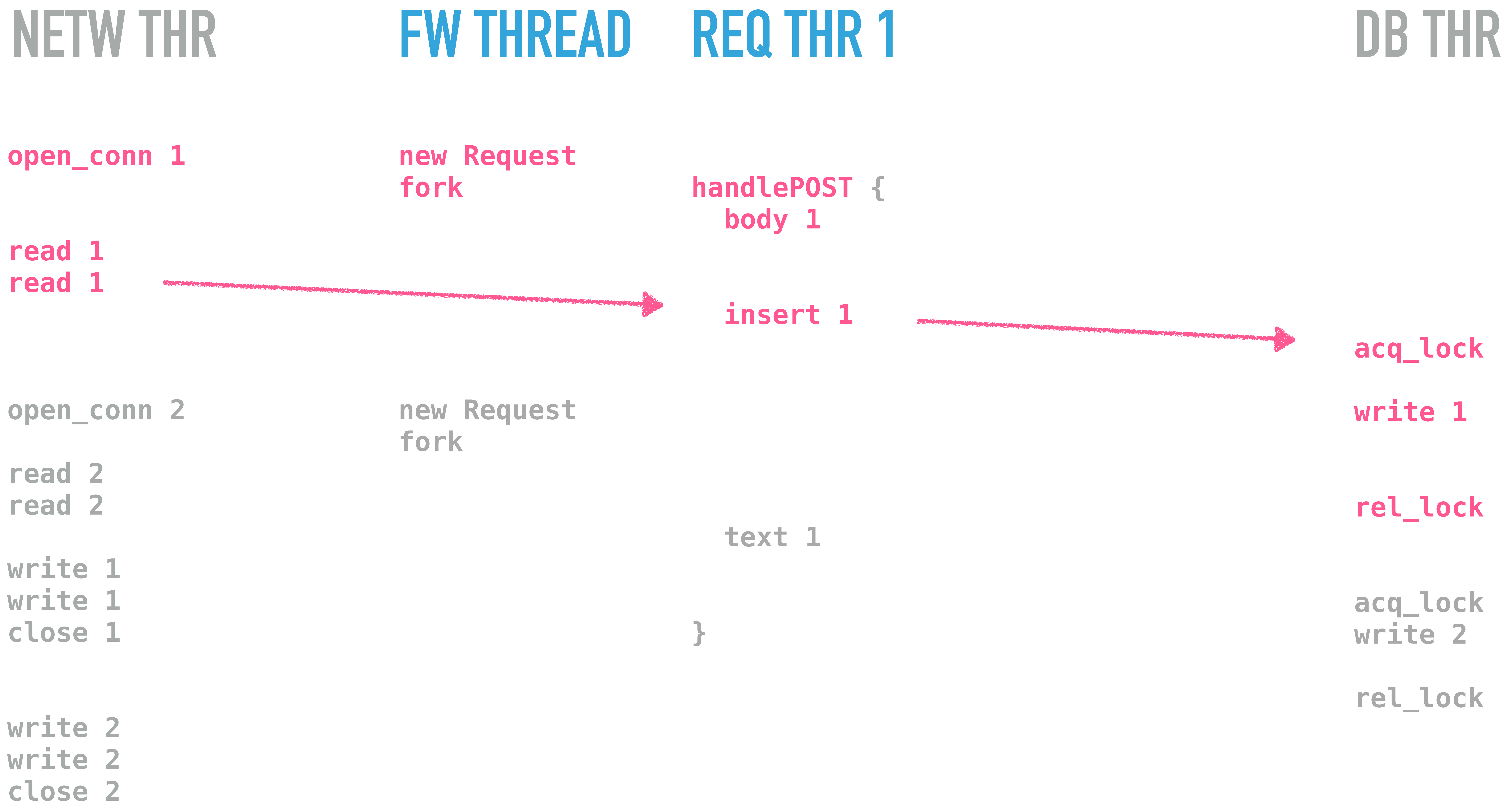
NETW THR	FW THREAD	DB THR
open_conn 1	new Request	
	fork	
read 1		
read 1		
		acq_lock
open_conn 2	new Request	write 1
	fork	
read 2		
read 2		rel_lock
write 1		
write 1		acq_lock
close 1		write 2
		rel_lock
write 2		
write 2		
close 2		

NETW THR	FW THREAD	REQ THR 1	DB THR
open_conn 1	new Request		
	fork	handlePOST {	
		body 1	
read 1			
read 1			
		insert 1	
			acq_lock
open_conn 2	new Request		write 1
	fork		
read 2			
read 2			rel_lock
		text 1	
write 1			
write 1			acq_lock
close 1		}	write 2
			rel_lock
write 2			
write 2			
close 2			

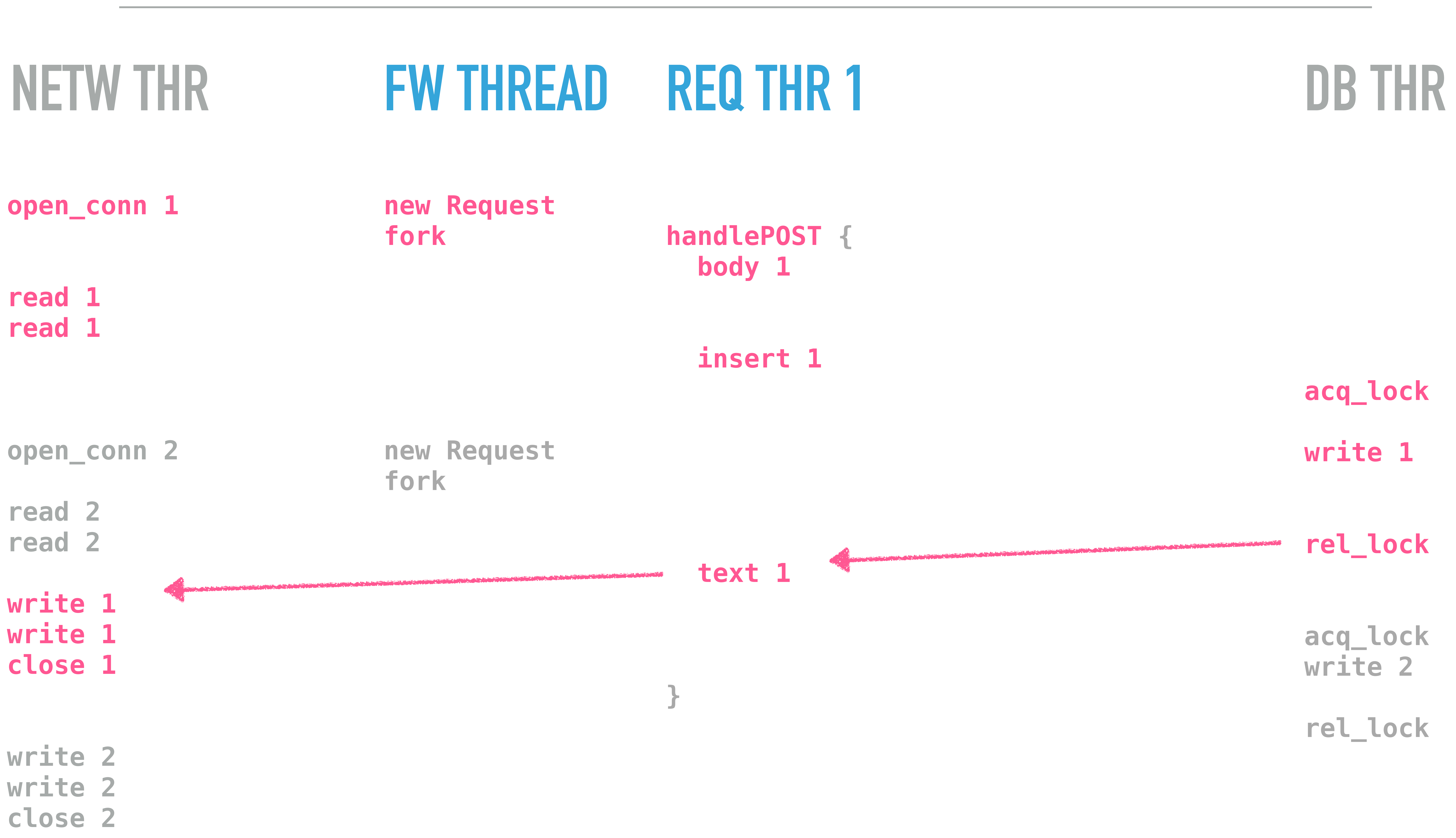
NETW THR	FW THREAD	REQ THR 1	DB THR
open_conn 1	new Request		
	fork	handlePOST {	
		body 1	
read 1			
read 1			
		insert 1	
			acq_lock
open_conn 2	new Request		write 1
	fork		
read 2			
read 2			rel_lock
		text 1	
write 1			
write 1			acq_lock
close 1		}	write 2
			rel_lock
write 2			
write 2			
close 2			



```
NETW THR            FW THREAD         REQ THR 1             DB THR

open_conn 1         new Request
                    fork              handlePOST {
                                        body 1
read 1
read 1
                                        insert 1
                                                            acq_lock

open_conn 2         new Request                             write 1
                    fork

read 2
read 2                                                      rel_lock
                                        text 1

write 1
write 1                                                     acq_lock
close 1                               }                     write 2

                                                            rel_lock

write 2
write 2
close 2
```



```
NETW THR          FW THREAD         REQ THR 1                              DB THR

open_conn 1       new Request
                  fork              handlePOST {
                                      body 1
read 1
read 1
                                      insert 1
                                                                           acq_lock

open_conn 2       new Request                                              write 1
                  fork
read 2
read 2                                                                     rel_lock
                                      text 1
write 1
write 1                                                                    acq_lock
close 1                             }                                      write 2

                                                                           rel_lock
write 2
write 2
close 2
```


NETW THR
FW THREAD
REQ THR 1
DB THR
open_conn 1
new Request
fork
handlePOST {
body 1
read 1
read 1
insert 1
acq_lock
open_conn 2
new Request
fork
write 1
read 2
read 2
rel_lock
text 1
write 1
write 1
close 1
acq_lock
write 2
}
rel_lock
write 2
write 2
close 2

NETW THR	FW THREAD	REQ THR 1	REQ THR 2	DB THR
open_conn 1	new Request			
	fork	handlePOST {		
		body 1		
read 1				
read 1				
		insert 1		
				acq_lock
open_conn 2	new Request			write 1
	fork		handlePUT {	
read 2			body 2	
read 2				rel_lock
		text 1		
write 1			insert 2	
write 1				acq_lock
close 1		}		write 2
				rel_lock
write 2			text 2	
write 2				
close 2			}	

```
NETW THR      APPLICATION THREAD                            DB THR

open_conn 1   new Request
              async call  a_handlePOST {
                            a_body 1

read 1
read 1
                            a_insert 1
                                                            acq_lock

open_conn 2   new Request                                   write 1
              async call                  a_handlePOST {
read 2                                      a_body 2
read 2                                                      rel_lock
                            a_text 1
write 1                                     a_insert 2
write 1                                                     acq_lock
close 1                   }                                 write 2

                                                            rel_lock
write 2                                     a_text 2
write 2
close 2                                   }
```

BANKREKENING

```
var account;

// withdraw endpoint
function handlePOST(req,res) {

    let euros = req.query.euros;
    if (account >= euros) {
        account = account - euros;
        res.set_status(200);
        res.close();
    } else {
        res.set_status(401);
        res.close();
    }

}
```

REQUEST 1 – WITHDRAW 250

```
var account;

function handlePOST(req,res) {

    let euros = req.query.euros;
    if (account >= euros) {
        account = account - euros;
        res.set_status(200);
        res.close();


    } else {
        res.set_status(401);
        res.close();
    }
}
```

```
<account starts at 300>

<context switch>
<account has 50 euros>

<account has 50 euros>
```

✓

REQUEST 2 – WITHDRAW 250

```
var account;

function handlePOST(req,res) {

    let euros = req.query.euros;






    if (account >= euros) {
        account = account - euros;
        res.set_status(200);
        res.close();
    } else {
        res.set_status(401);
        res.close();
    }

}
```

REQUEST 1 – WITHDRAW 250

```
var account;                               <account starts at 300>

function handlePOST(req,res) {

    let euros = req.query.euros;
    if (account >= euros) {
                                              <context switch>
                                           <account has 300 euros>




                                              <context switch>
                                            <account has 50 euros>
        account = account – euros;
        res.set_status(200);
        res.close();

    } else {
        res.set_status(401);              <account has -200 euros>
        res.close();                                ✗
    }
}
```

REQUEST 2 – WITHDRAW 250

```
var account;

function handlePOST(req,res) {

    let euros = req.query.euros;


    if (account >= euros) {
        account = account – euros;
        res.set_status(200);
        res.close();




    } else {
        res.set_status(401);
        res.close();
    }

}
```

REQUEST 1 – WITHDRAW 250

```
var account;

async function handlePOST(req,res) {
    let euros = req.query.euros;
    if (account >= euros) {
        account = account – euros;
        res.set_status(200);
        await res.close();


    } else {
        res.set_status(401);
        await res.close();
    }
}
```

<account starts at 300>

<task switch>
<account has 50 euros>

REQUEST 2 – WITHDRAW 250

```
var account;

async function handlePOST(req,res) {
    let euros = req.query.euros;
    if (account >= euros) {
        account = account – euros;
        res.set_status(200);
        await res.close();
    } else {
        res.set_status(401);
        await res.close();
    }
}
```

✓

BANK ACCOUNT ACTOR

```
var account;

// withdraw endpoint
async function handlePOST(req,res) {

    let euros = req.query.euros;
    let ok = await account.withdraw(euros);
    if (ok) {
       res.set_status(200);
    else
       res.set_status(401);
    res.close();

}
```

```
class Account {
    var amount = 0;
    var q = new MessageQueue();

    async function withdraw(euros) {
        let msg = new Msg('w',euros);
        this.q.put(msg);
        let ok = await msg.response.get();
        return ok;
    }









}
```

BANK ACCOUNT ACTOR

```
var account;

// withdraw endpoint
async function handlePOST(req,res) {

    let euros = req.query.euros;
    let ok = await account.withdraw(euros);
    if (ok) {
       res.set_status(200);
    else
       res.set_status(401);
    res.close();

}
```

```
class Account {
    var amount = 0;
    var q = new MessageQueue();

    async function withdraw(euros) {
        let msg = new Msg('w',euros);
        this.q.put(msg);
        let ok = await msg.response.get();
        return ok;
    }

    async function handleQ() {
        while (true) {
            let msg = await q.get();
            if (msg.cmd == 'w') {
                if (amount >= msg.euros) {
                    amount -= euros;
                    msg.response.put(true);
                } else {
                    msg.response.put(false);
                }
            }
        }
    }
}
```

async programmeren =
coöperatief multitasken
binnen één thread

- ☠ task mag NOOIT blocken
- ❤ geen context switches
- ❤ minder locking nodig