Don't Trust the PID!

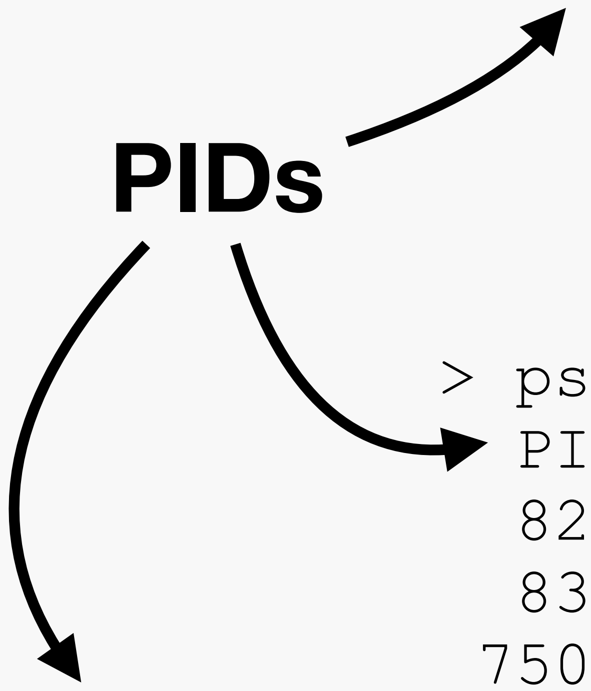
Stories of a simple logic bug and where to find it

Samuel Groß (@5aelo)

The PID (Process IDentifier)

- Used to identify a running process
- Incremented when spawning new process
- For historical reasons limited to < 100k*

```
/usr/bin/whoami &
# root
echo $!
# 52892
```



PIDs

```
int pid = fork();
if (pid == 0) {
    return do_child();
} else if (pid < 0) {
    return -1;
}
printf("Child PID: %d\n", pid);
```

```
> ps
  PID TTY           TIM
  828 ttys000    0:00.2
  830 ttys000    0:01.8
 7508 ttys001    0:00.0
15820 ttys001    0:00.2
15822 ttys001    0:00.8
```

*on XNU at least, presumably it was originally stored in a 16-bit int

PID Wraparound

- What happens after 100k processes have been spawned?
- PID wraps around, next free PID is reused
- Try this at home:

```
while (1) {
    int pid = fork();
    if (pid <= 0) {
        break;
    } else {
        printf("pid: %d\n", pid);
        wait(NULL);
    }
}
```

😮*

```
...
pid: 99994
pid: 99995
pid: 99996
pid: 99997
pid: 99998
pid: 103
pid: 104
pid: 106
pid: 109
...
```

* not actually that surprising

A Vulnerability Pattern

```
int pid = client->pid;
if (security_check(action, pid)) {
    perform_action(client);
}
```

Some local IPC service

A Vulnerability Pattern

```
int pid = client->pid;
if (security_check(action, pid)) {
    perform_action(client);
}
```

Problem: no guarantee this is still the requesting process

A Vulnerability Pattern

```
int pid = client->pid;
if (security_check(action, pid)) {
    perform_action(client);
}
```

- Race condition: client process terminates and somehow a new, more privileged process is spawned into its PID
- Vulnerability comes in different "flavours"
- Sometimes conveniently exploitable if PID is cached

Example

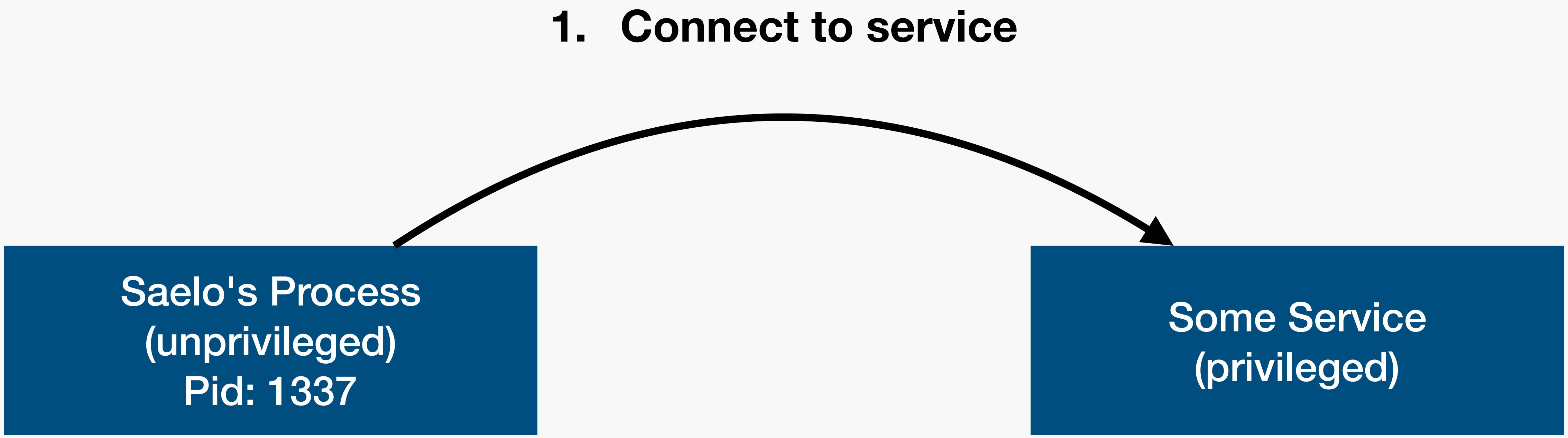
Saelo's Process
(unprivileged)
Pid: 1337

Some Service
(privileged)

Goal: get here

```
void Service::handleMessage(...)
{
    int pid = client->pid;
    if (security_check(action, pid)) {
        perform_action(client);
    }
}
```

Example Attack



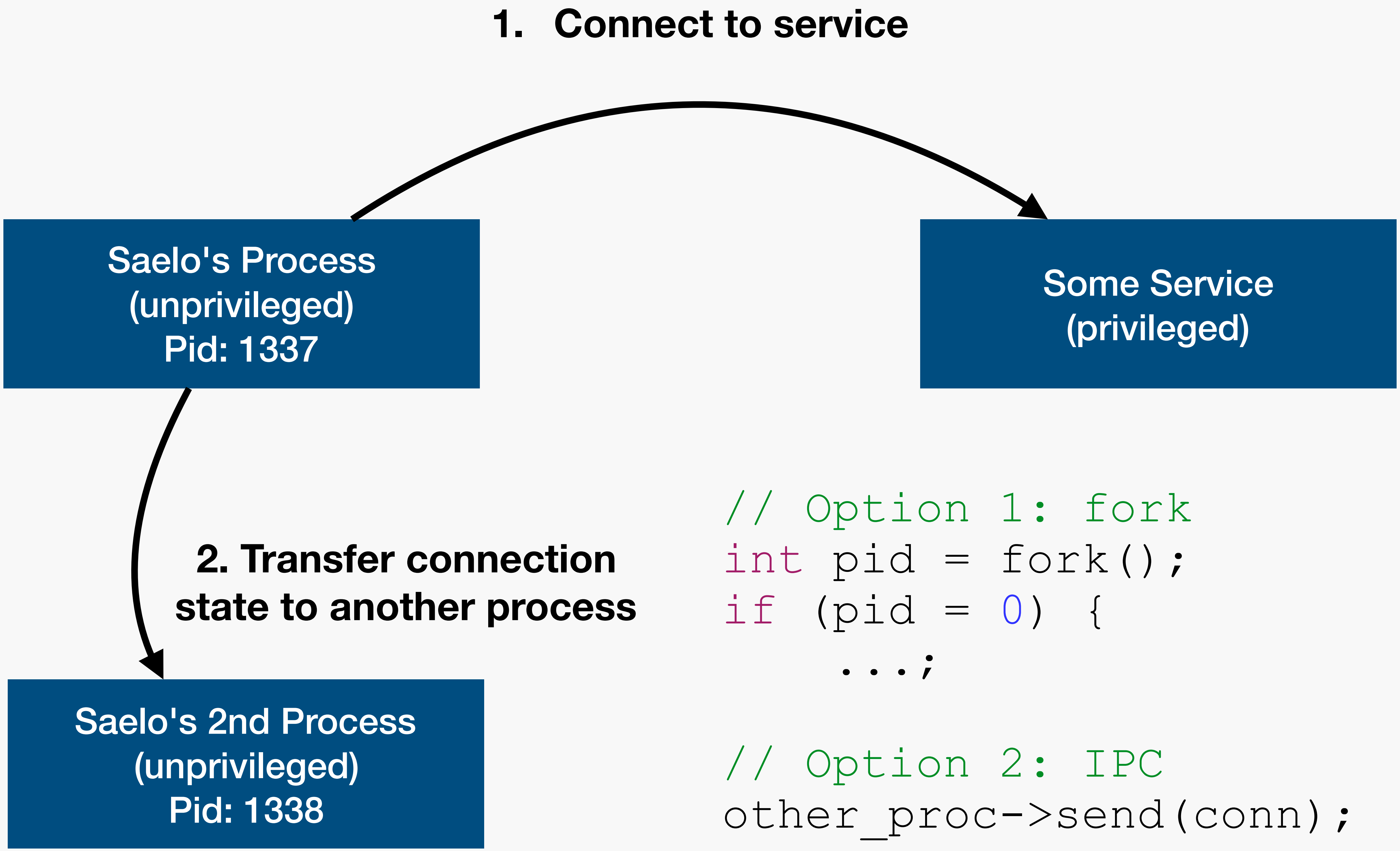
1. Connect to service

Saelo's Process
(unprivileged)
Pid: 1337

Some Service
(privileged)

```
void Service::acceptConnection(...)
{
      ...;
      Client* = new Client;
      client->pid = getRemotePid()
      ...;
}
```

Example Attack

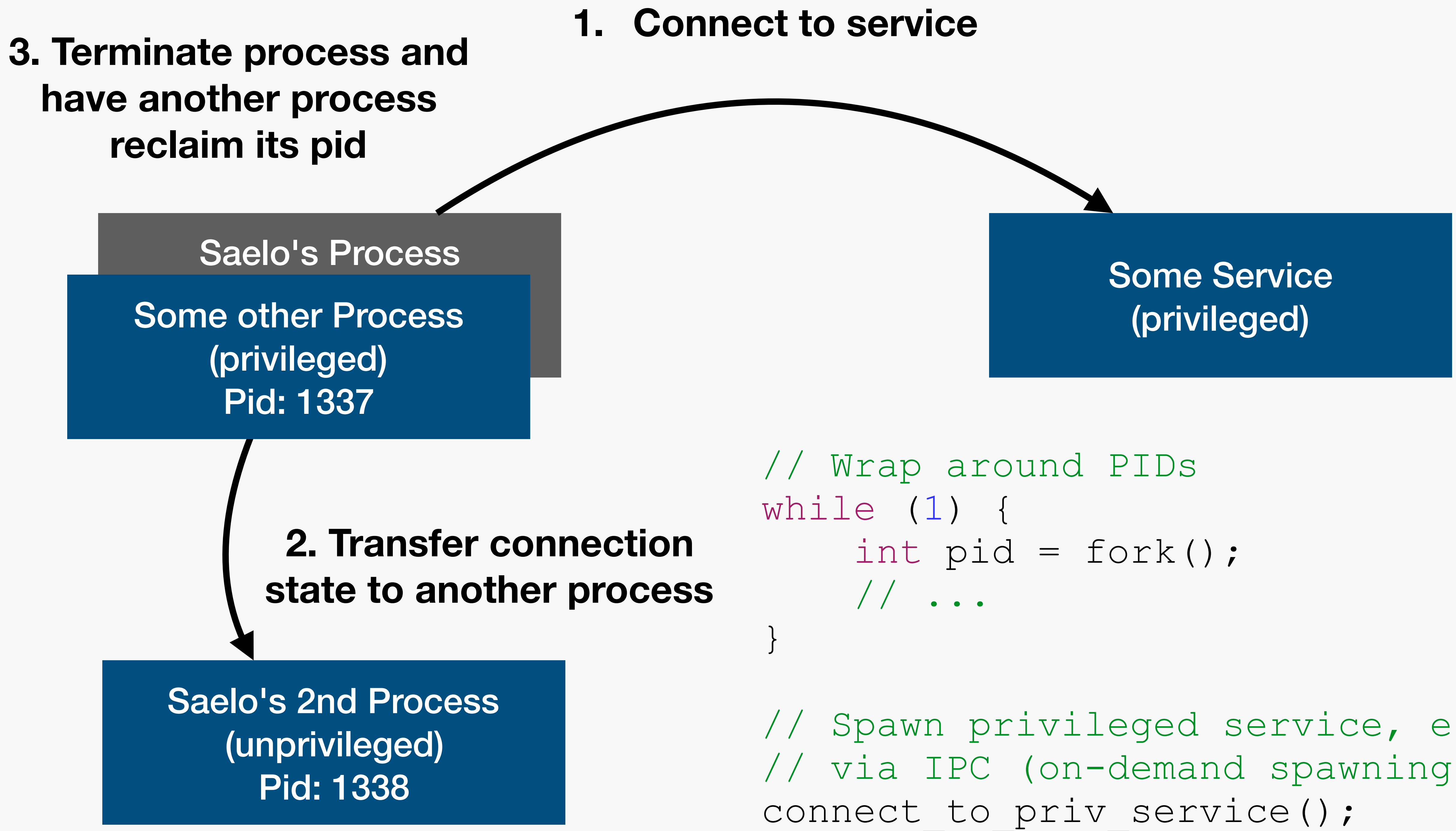


1. Connect to service

2. Transfer connection state to another process

```
// Option 1: fork
int pid = fork();
if (pid = 0) {
    ...;

// Option 2: IPC
other_proc->send(conn);
```

Example Attack



1. Connect to service

3. Terminate process and have another process reclaim its pid

Saelo's Process

Some other Process (privileged) Pid: 1337

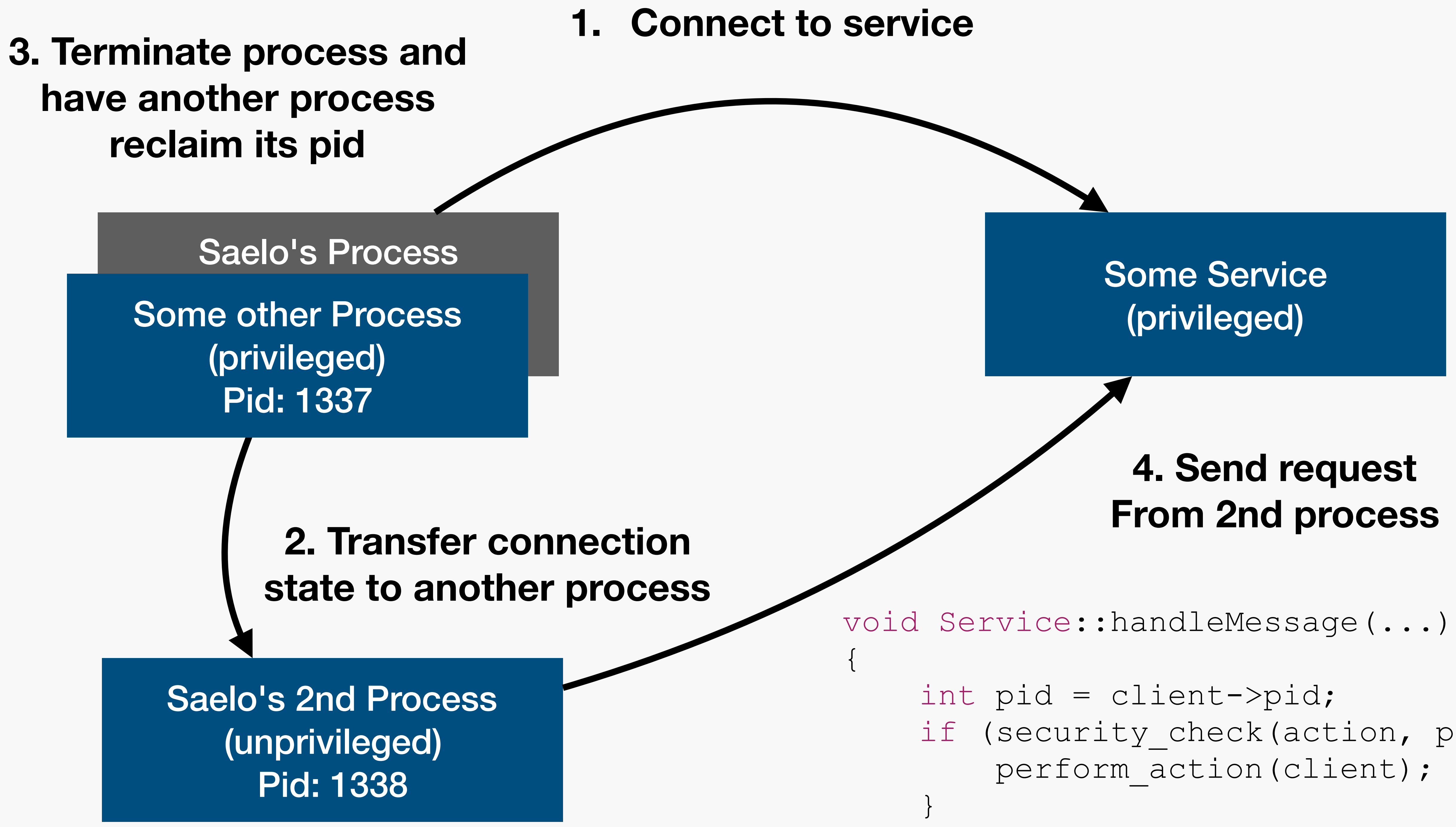
Some Service (privileged)

2. Transfer connection state to another process

Saelo's 2nd Process (unprivileged) Pid: 1338

```
// Wrap around PIDs
while (1) {
    int pid = fork();
    // ...
}

// Spawn privileged service, e.g.
// via IPC (on-demand spawning)
connect_to_priv_service();
```

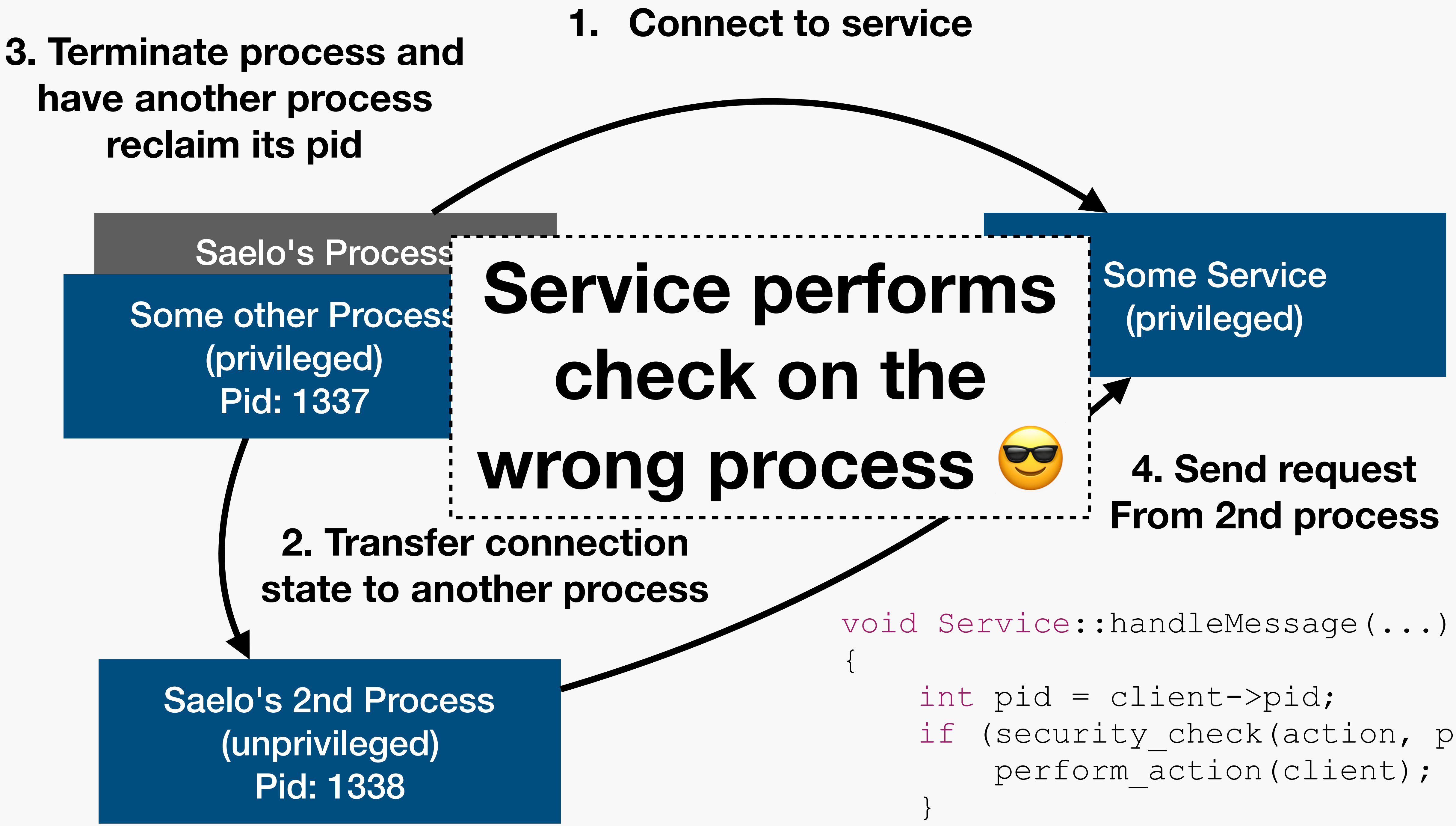
Example Attack



1. Connect to service
3. Terminate process and have another process reclaim its pid
Saelo's Process
Some other Process (privileged) Pid: 1337
Some Service (privileged)
2. Transfer connection state to another process
4. Send request From 2nd process
Saelo's 2nd Process (unprivileged) Pid: 1338

```
void Service::handleMessage(...)
{
    int pid = client->pid;
    if (security_check(action, pid)) {
        perform_action(client);
    }
}
```

Example Attack



1. Connect to service

2. Transfer connection state to another process

3. Terminate process and have another process reclaim its pid

4. Send request From 2nd process

Saelo's Process
Some other Process (privileged) Pid: 1337
Some Service (privileged)
Saelo's 2nd Process (unprivileged) Pid: 1338

Service performs check on the wrong process 😎

```
void Service::handleMessage(...)
{
    int pid = client->pid;
    if (security_check(action, pid)) {
        perform_action(client);
    }
}
```

Preconditions

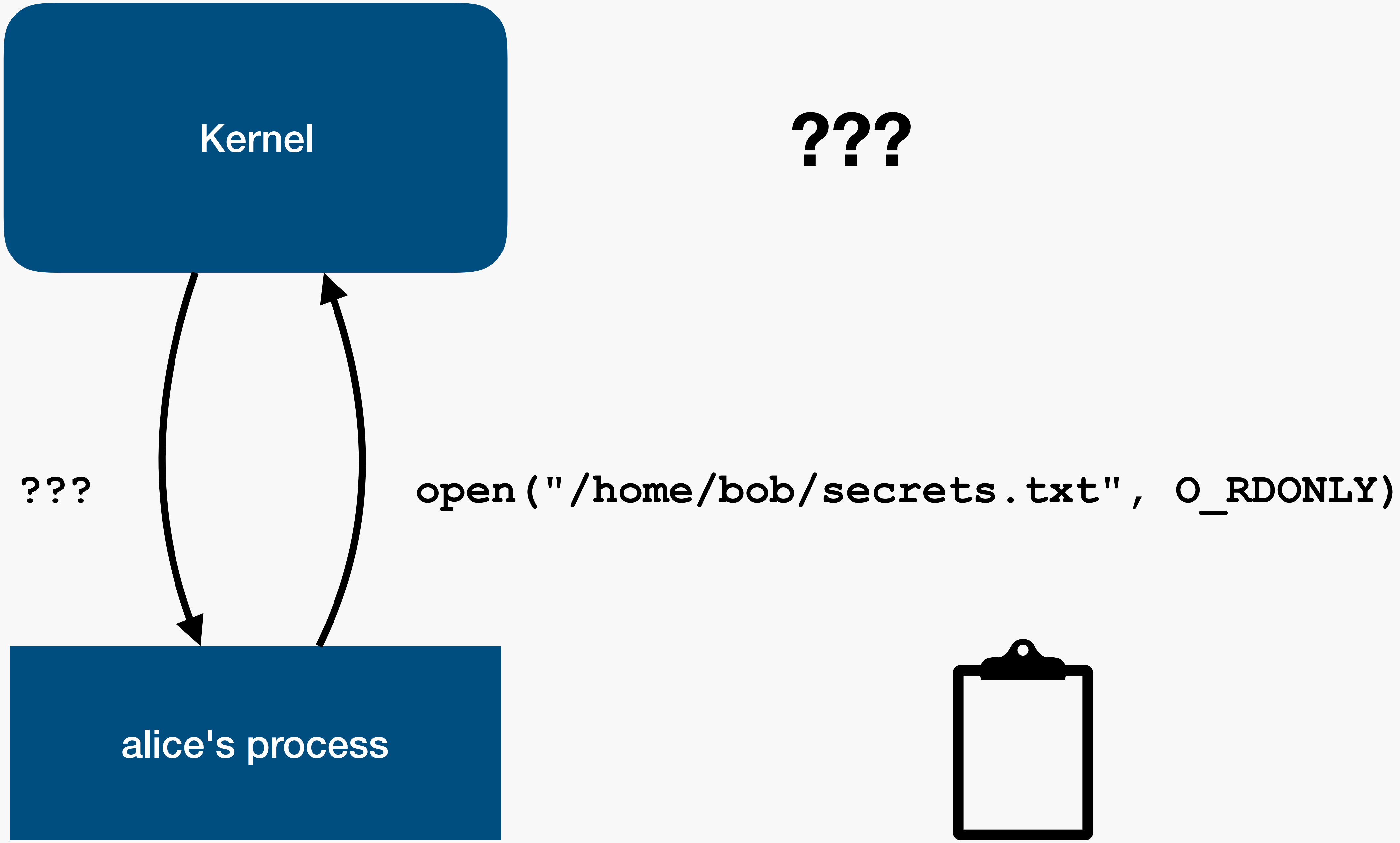
- Usually need at least two controlled processes that can communicate with each other
- Ability to spawn many (unprivileged) processes to wrap around PIDs
- Ability to spawn at least one privileged process

Agenda

1. Why does this happen?
   - Overview: macOS userland security and sandboxing
2. How to do it correctly?
   - The audit token
3. Where has this happened?
   - authd and pwn2own 2017
   - `sandbox_check` fundamentally broken

"Classic" OS Design

Kernel manages all ressources



Kernel

???

???

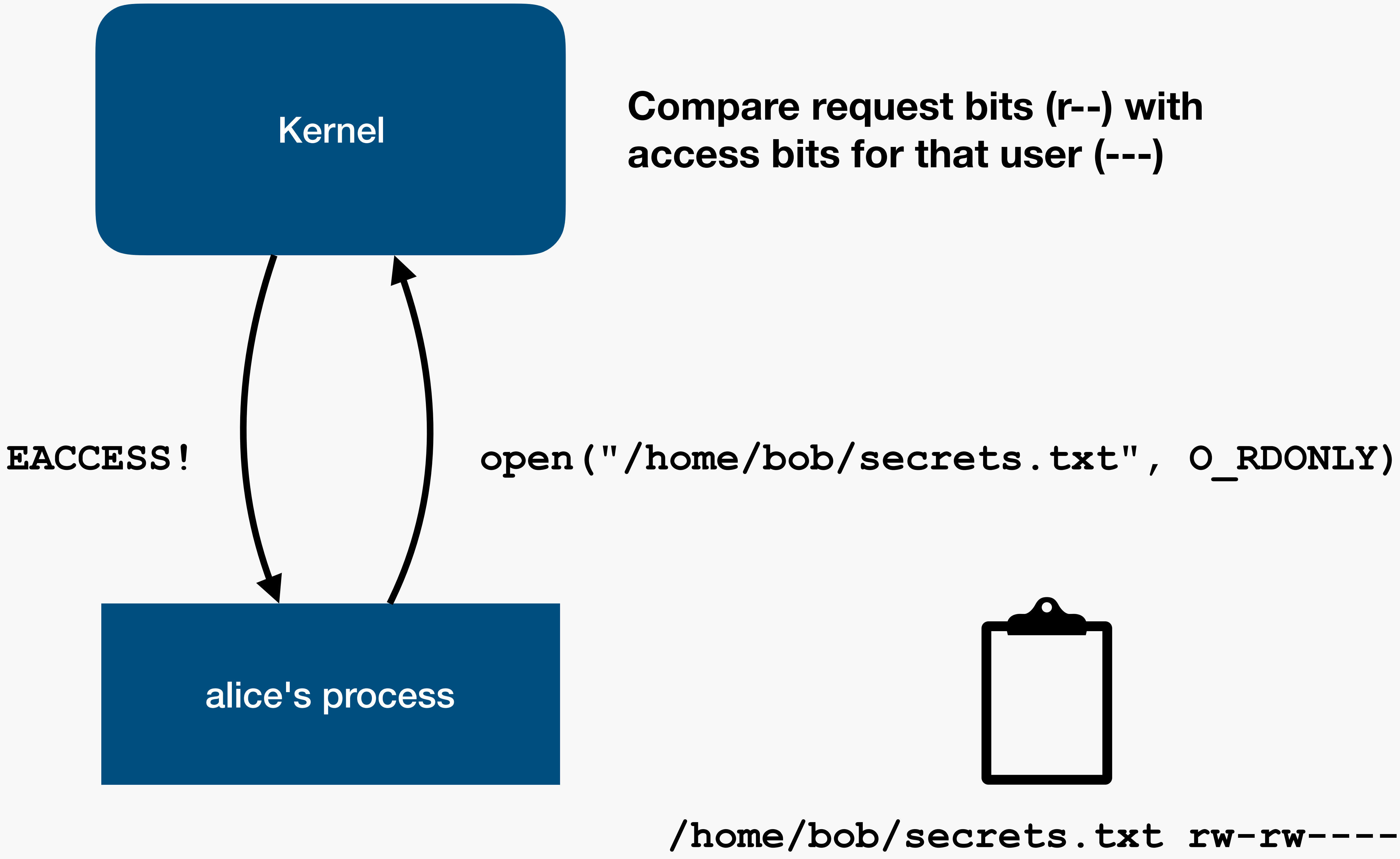
`open("/home/bob/secrets.txt", O_RDONLY)`

alice's process

`/home/bob/secrets.txt rw-rw----`

"Classic" OS Design

Kernel manages all ressources


Kernel
EACCESS!
open("/home/bob/secrets.txt", O_RDONLY)
alice's process
/home/bob/secrets.txt rw-rw----

Compare request bits (r--) with access bits for that user (---)

Userspace Resources?

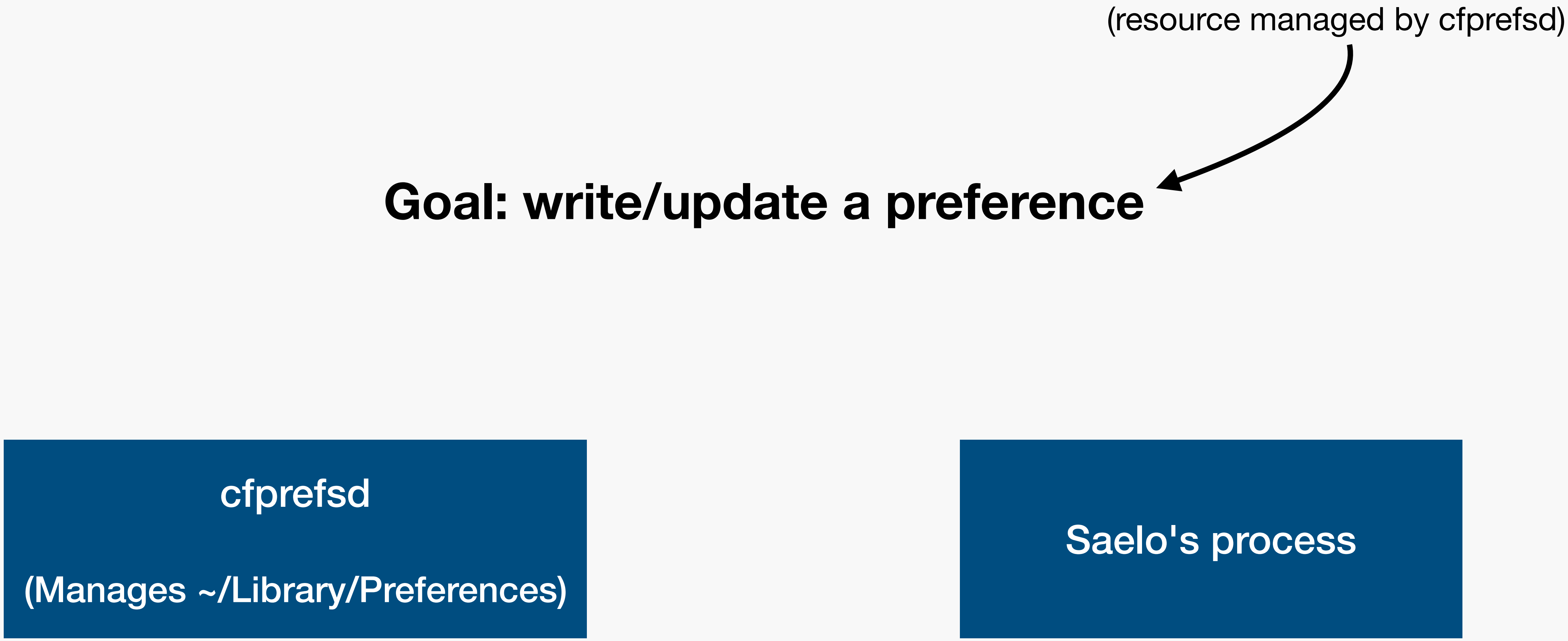
Wanted: resource management in userspace

- Cloud documents, contacts, UI events, clipboard, preferences, keychain, ... are all userspace "resources"

Benefits of managing things in userspace:

- Userspace code probably easier to write than kernel code
- Access to memory safe languages (e.g. Swift on macOS)
- Small, restricted services that can be sandboxed to only have access to the resources they need

Example: cfprefsd



(resource managed by cfprefsd)

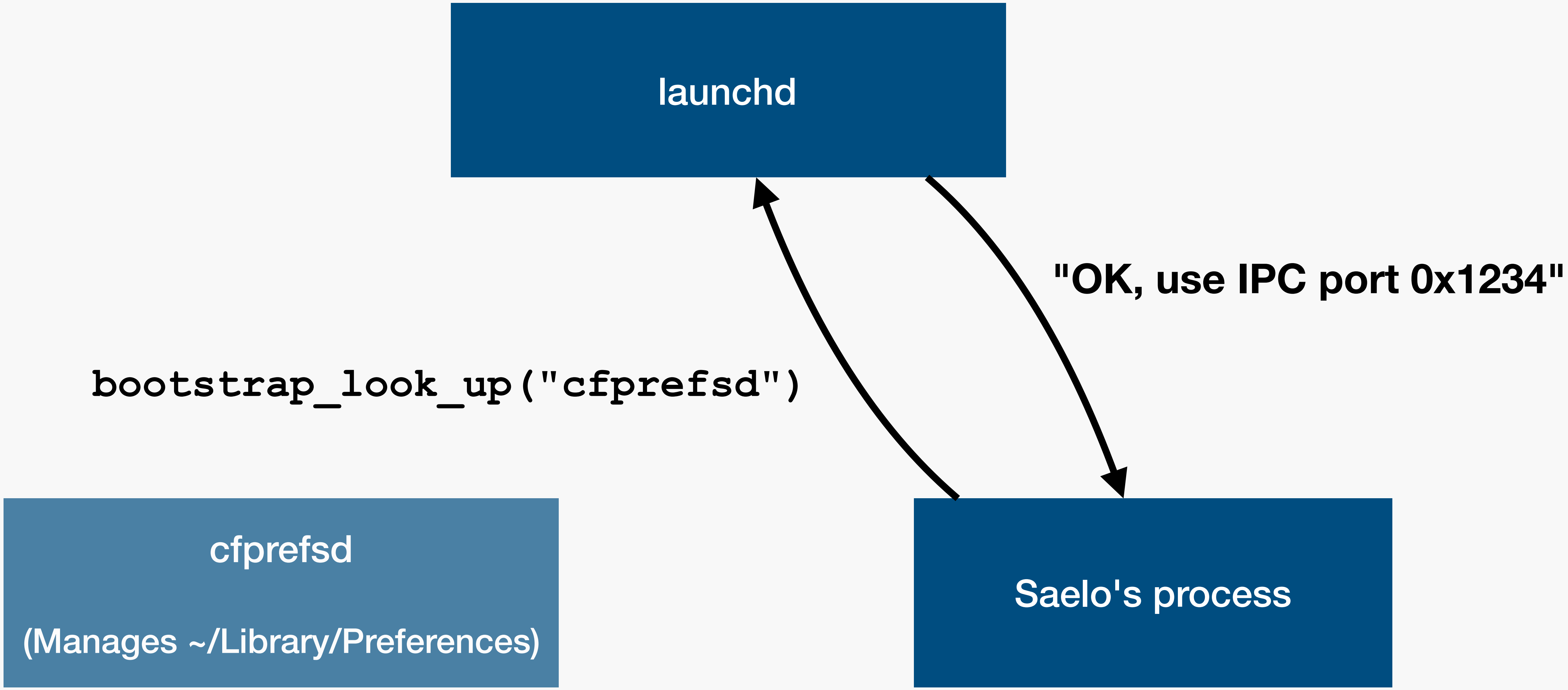
Goal: write/update a preference

cfprefsd

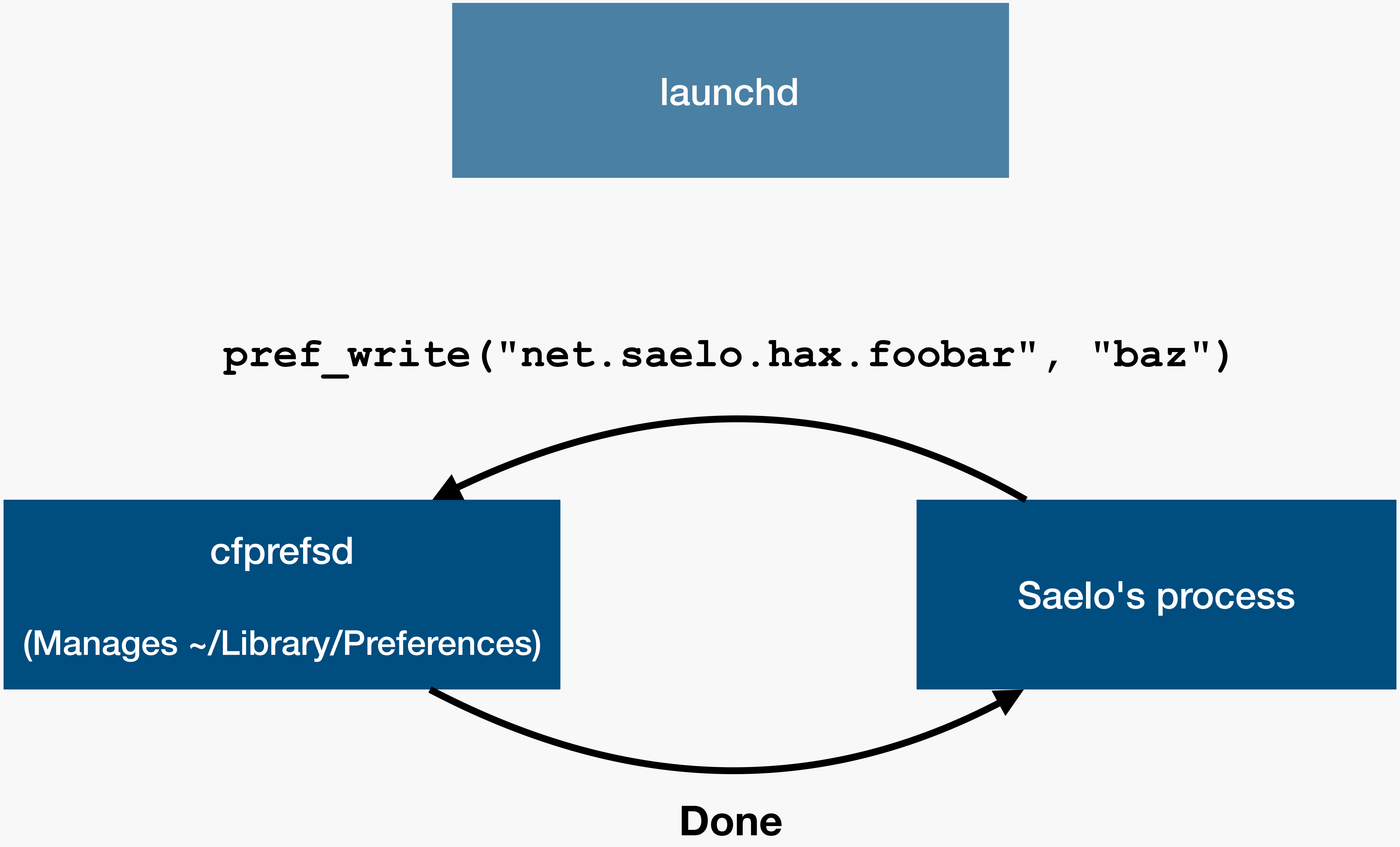
(Manages ~/Library/Preferences)

Saelo's process

Example: cfprefsd


launchd
"OK, use IPC port 0x1234"
bootstrap_look_up("cfprefsd")
cfprefsd
(Manages ~/Library/Preferences)
Saelo's process

Example: cfprefsd


launchd
pref_write("net.saelo.hax.foobar", "baz")
cfprefsd
(Manages ~/Library/Preferences)
Saelo's process
Done

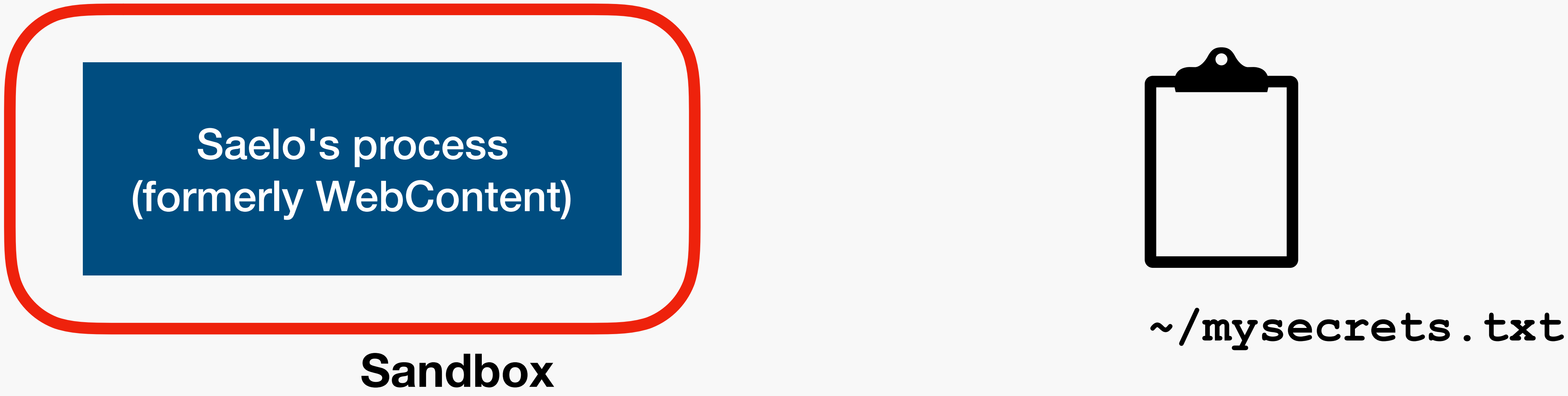
Userspace Security, 1

- Services eventually need to do access checks
  - cfprefsd shouldn't allow reading/writing other user's preferences
- So far simple: kernel can attach UID/GID etc. to IPC messages and services can use those

Adding Flexibility

Classic security model not flexible enough, might also want:

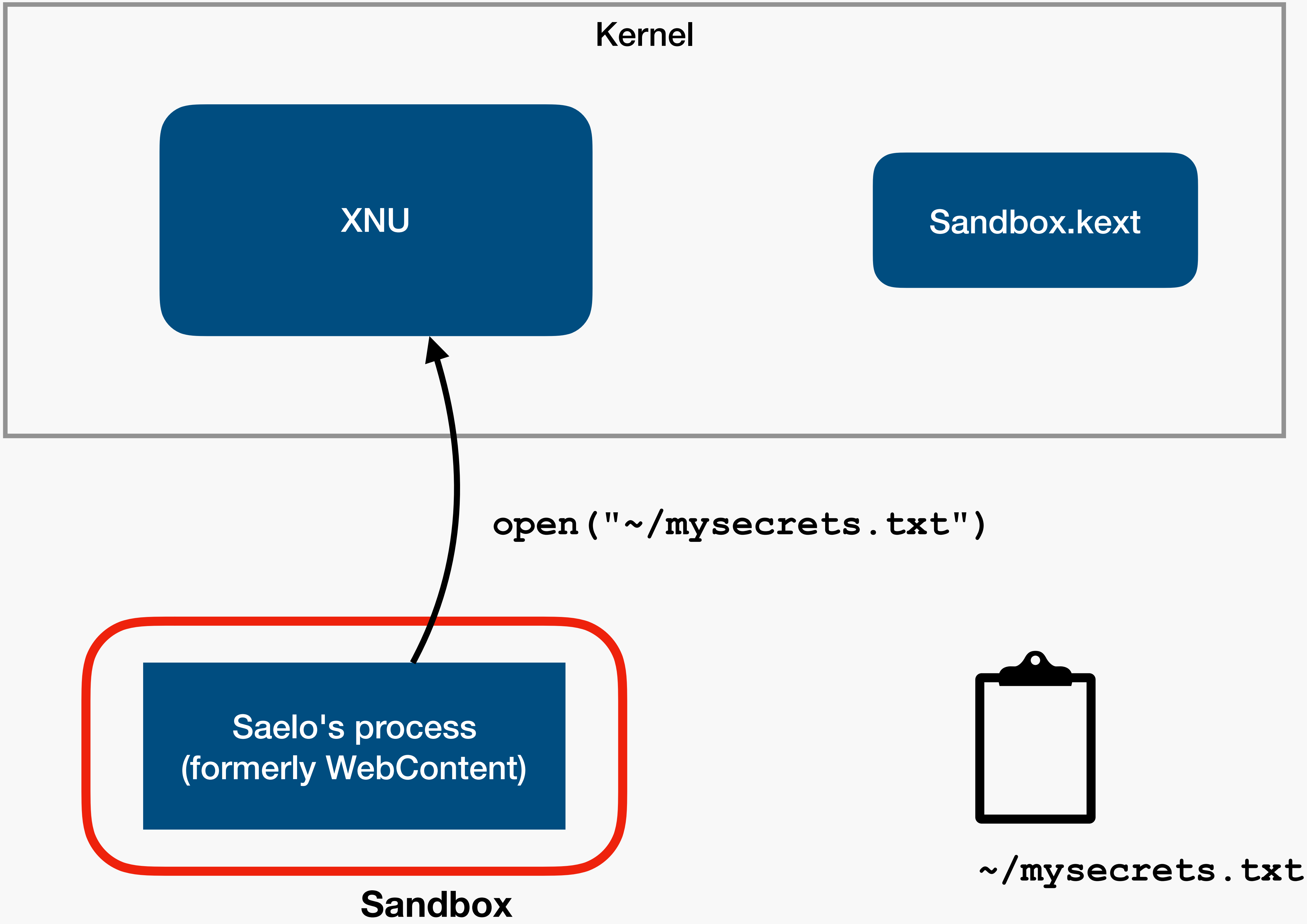
- Sandboxing, i.e. mechanism to restrict selected processes
- Entitlements, i.e. mechanism to empower selected processes



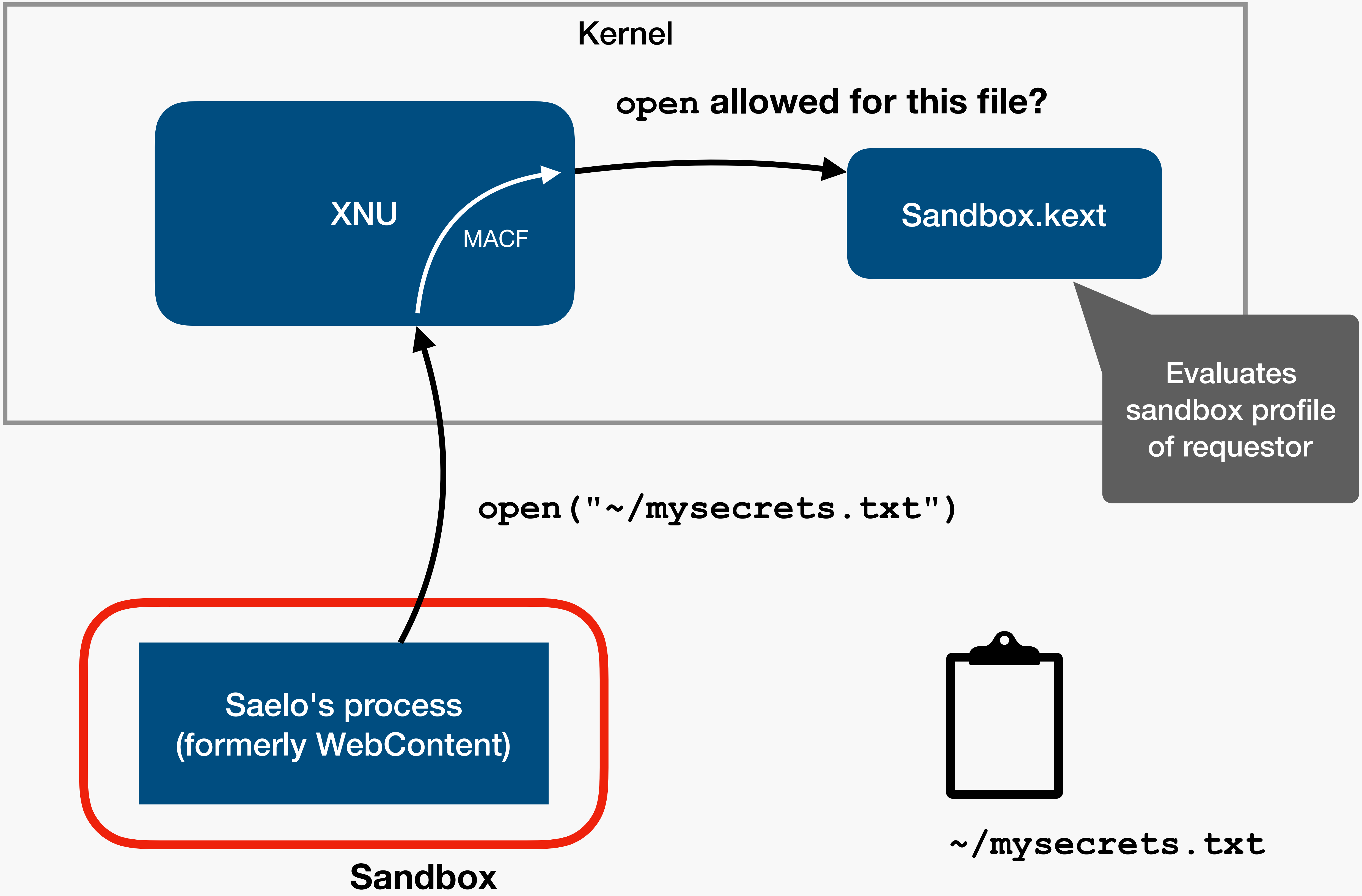
Saelo's process
(formerly WebContent)

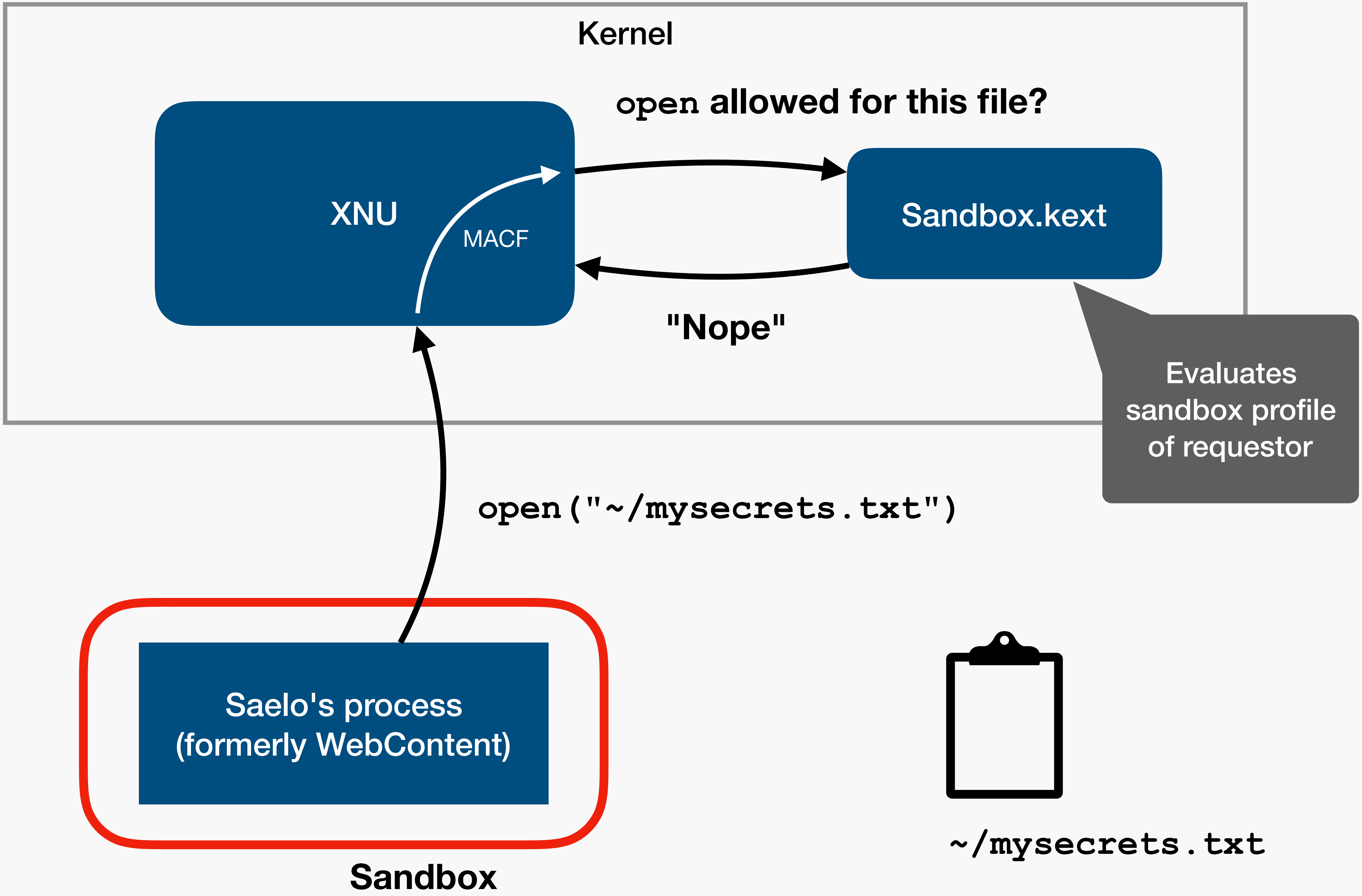
Sandbox

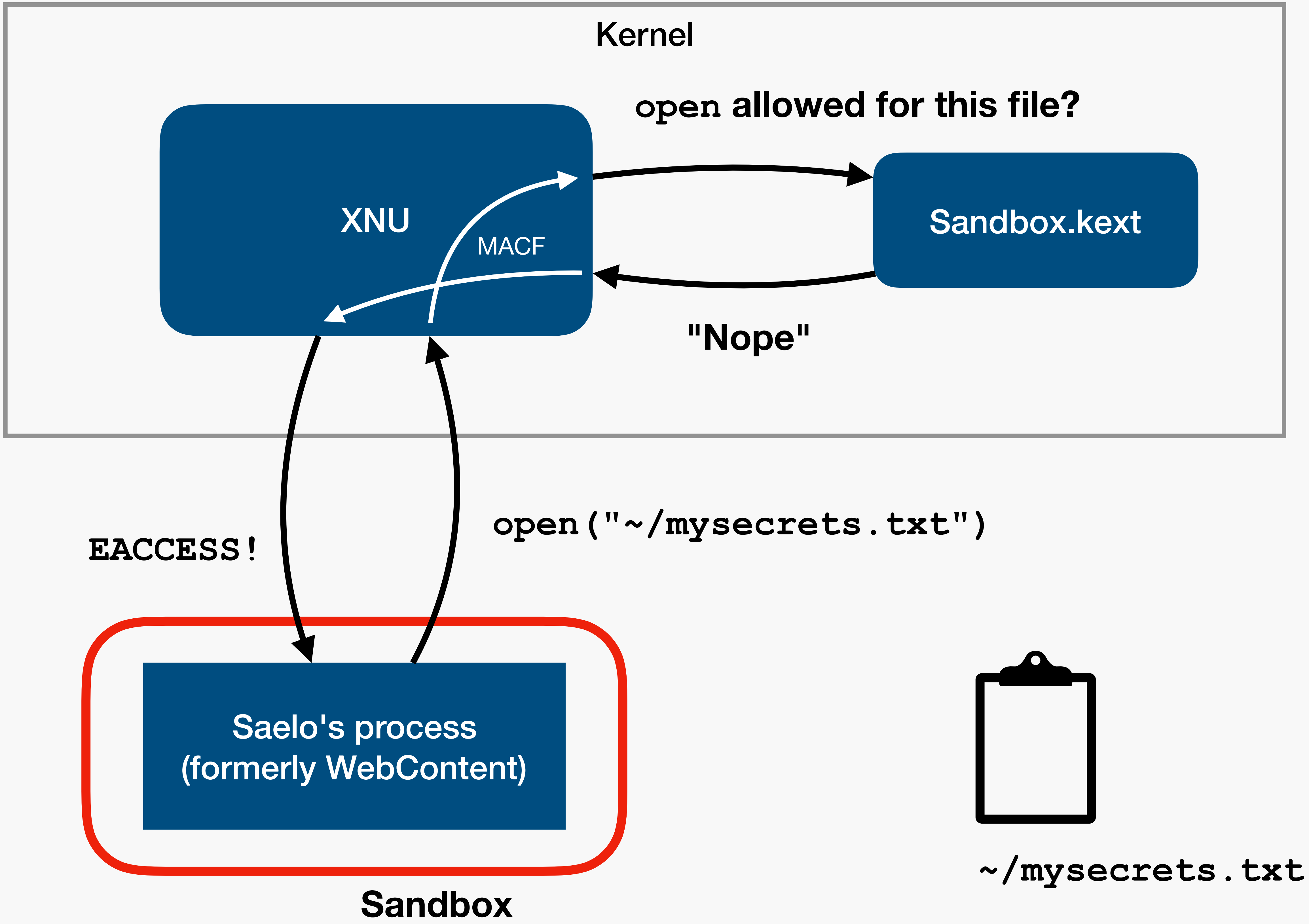
`~/mysecrets.txt`



Kernel
XNU
Sandbox.kext
open("~/mysecrets.txt")
Saelo's process
(formerly WebContent)
Sandbox
~/mysecrets.txt


Kernel
open allowed for this file?
XNU
MACF
Sandbox.kext
Evaluates sandbox profile of requestor
open("~/mysecrets.txt")
Saelo's process (formerly WebContent)
Sandbox
~/mysecrets.txt


Kernel
open allowed for this file?
XNU
MACF
Sandbox.kext
"Nope"
Evaluates sandbox profile of requestor
open("~/mysecrets.txt")
Saelo's process (formerly WebContent)
Sandbox
~/mysecrets.txt



Kernel

XNU

MACF

Sandbox.kext

`open` allowed for this file?

"Nope"

`EACCESS!`

`open("~/mysecrets.txt")`

Saelo's process (formerly WebContent)

Sandbox

`~/mysecrets.txt`

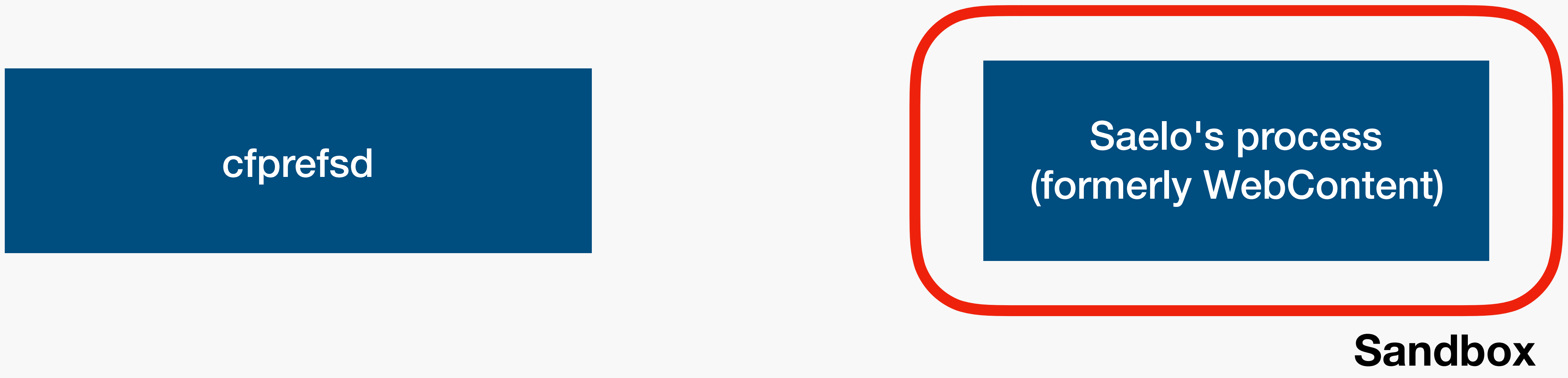
Userspace Security, 2

- Sandbox profile and entitlement information are required by some userspace services to perform access checks
  - cfprefsd shouldn't allow sandboxed processes to write preferences
- This data is associated with each process in the kernel

=> Must have API to obtain this information for a process

=> Intuitive (but bad) choice: query this data by PID

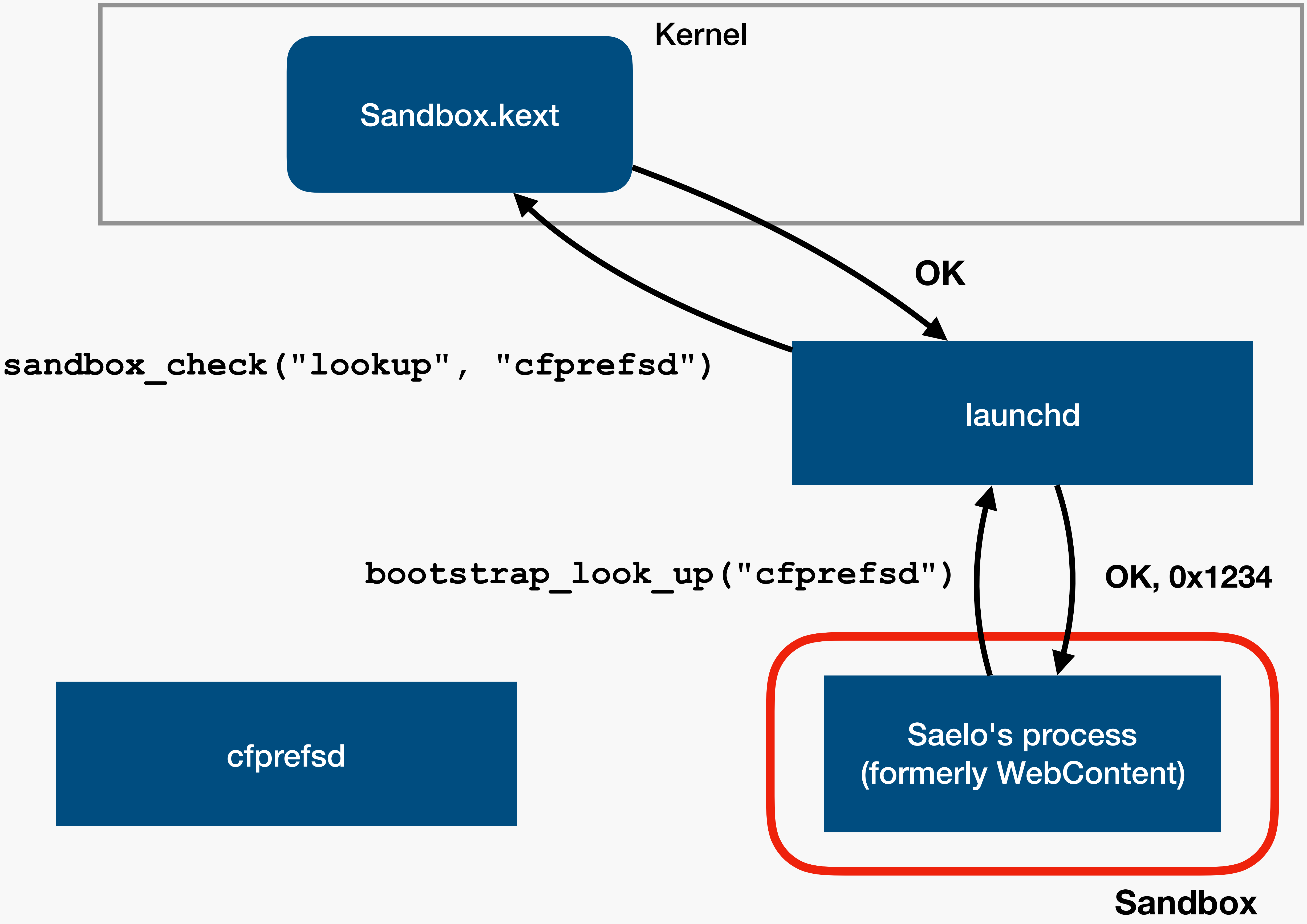
Goal: write/update a preference

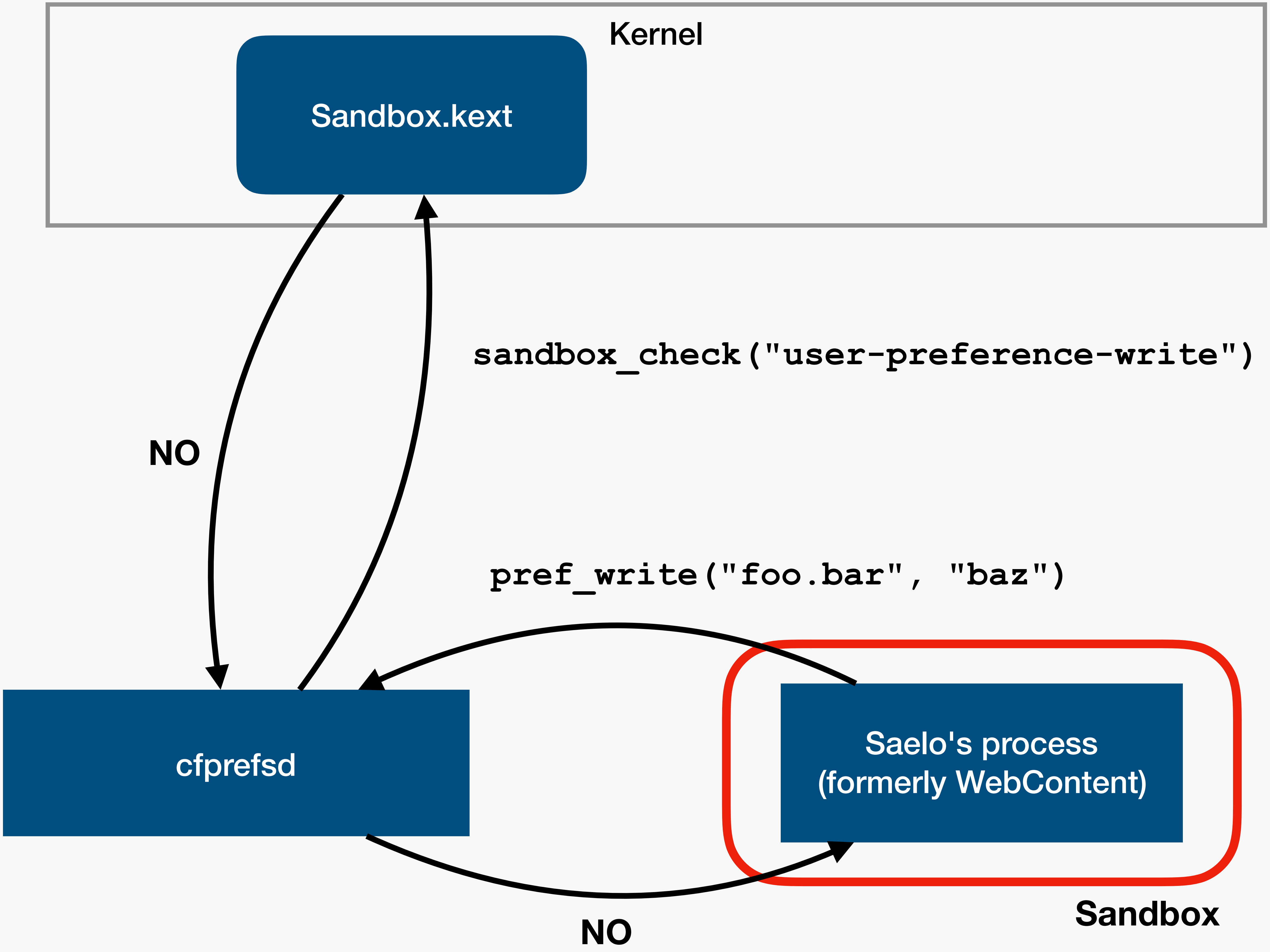


cfprefsd

Saelo's process (formerly WebContent)

Sandbox


Kernel
Sandbox.kext
OK
sandbox_check("lookup", "cfprefsd")
launchd
bootstrap_look_up("cfprefsd")
OK, 0x1234
cfprefsd
Saelo's process
(formerly WebContent)
Sandbox



Kernel

Sandbox.kext

```
sandbox_check("user-preference-write")
```

NO

```
pref_write("foo.bar", "baz")
```

cfprefsd

Saelo's process
(formerly WebContent)

NO

Sandbox

Userspace Security, 3

Potential problem now:

Access-control data can be obtained via a PID

=> Can lead to PID reuse issues and unsafe checks

How to do it correctly

The audit_token_t in XNU

The Basic Fix

- Simple: use a bigger, ideally unique PID instead
- In XNU: **`audit_token_t`**
  - Structure attached to IPC messages (mach messages)
  - Obtain via e.g. **`xpc_dictionary_get_audit_token`**
- Usual fix for PID related issues: use audit token instead

From apple's dev forum: "The OS's process ID space is relatively small, which means that process IDs are commonly reused. Thus, it's a bad idea to use a process ID in security-related work. There is a recommended alternative to process IDs, namely audit tokens, ..."

```
typedef struct {
  unsigned int        val[8];
} audit_token_t;
```

- Opaque structure
- Contains `p_idversion`, essentially a 32-bit PID
- Initialized in `set_security_token_task_internal`:

```
audit_token.val[0] = my_cred->cr_audit.as_aia_p->ai_auid;
audit_token.val[1] = my_pcred->cr_uid;
audit_token.val[2] = my_pcred->cr_gid;
audit_token.val[3] = my_pcred->cr_ruid;
audit_token.val[4] = my_pcred->cr_rgid;
audit_token.val[5] = p->p_pid;
audit_token.val[6] = my_cred->cr_audit.as_aia_p->ai_asid;
audit_token.val[7] = p->p_idversion;
```

Pwn2Own '17, authd

Pwn2Own 2017

Participated together with @_niklasb

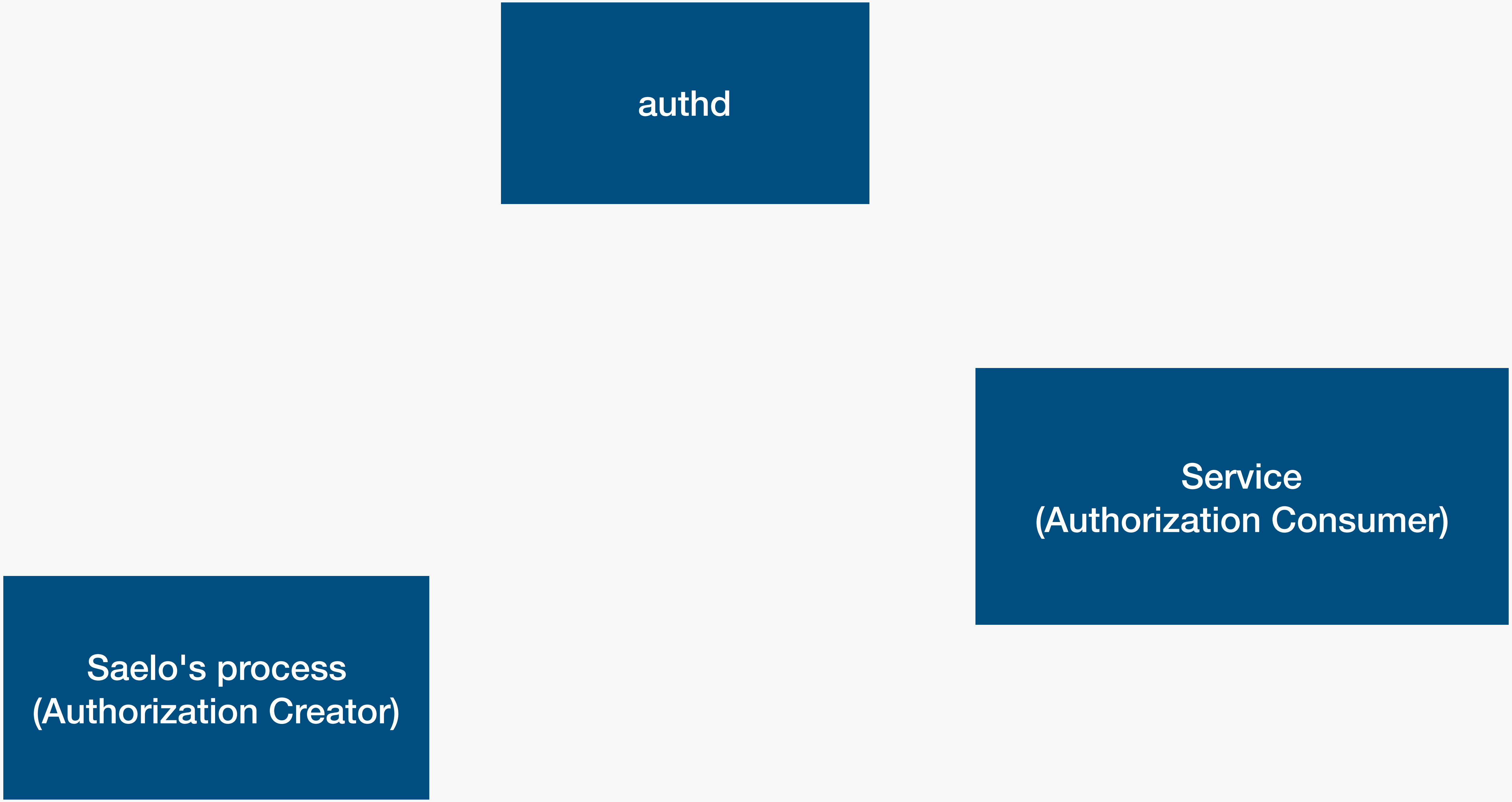
Context:

- Had Safari renderer bugs
- Niklas had a TOCTOU user -> root escalation in diskarbitrationd (CVE-2017-2533)
- But: couldn't reach it from the sandbox as it required the "system.volume.internal.mount" authorization

=> I started looking into authd for vulnerabilities

authd

- Authorization system for userspace policy enforcement
- Predates entitlement system and seems somewhat deprecated now (?)
- Service responsible for issuing "authorizations"
- Idea: rule system to determine whether process could obtain an authorization



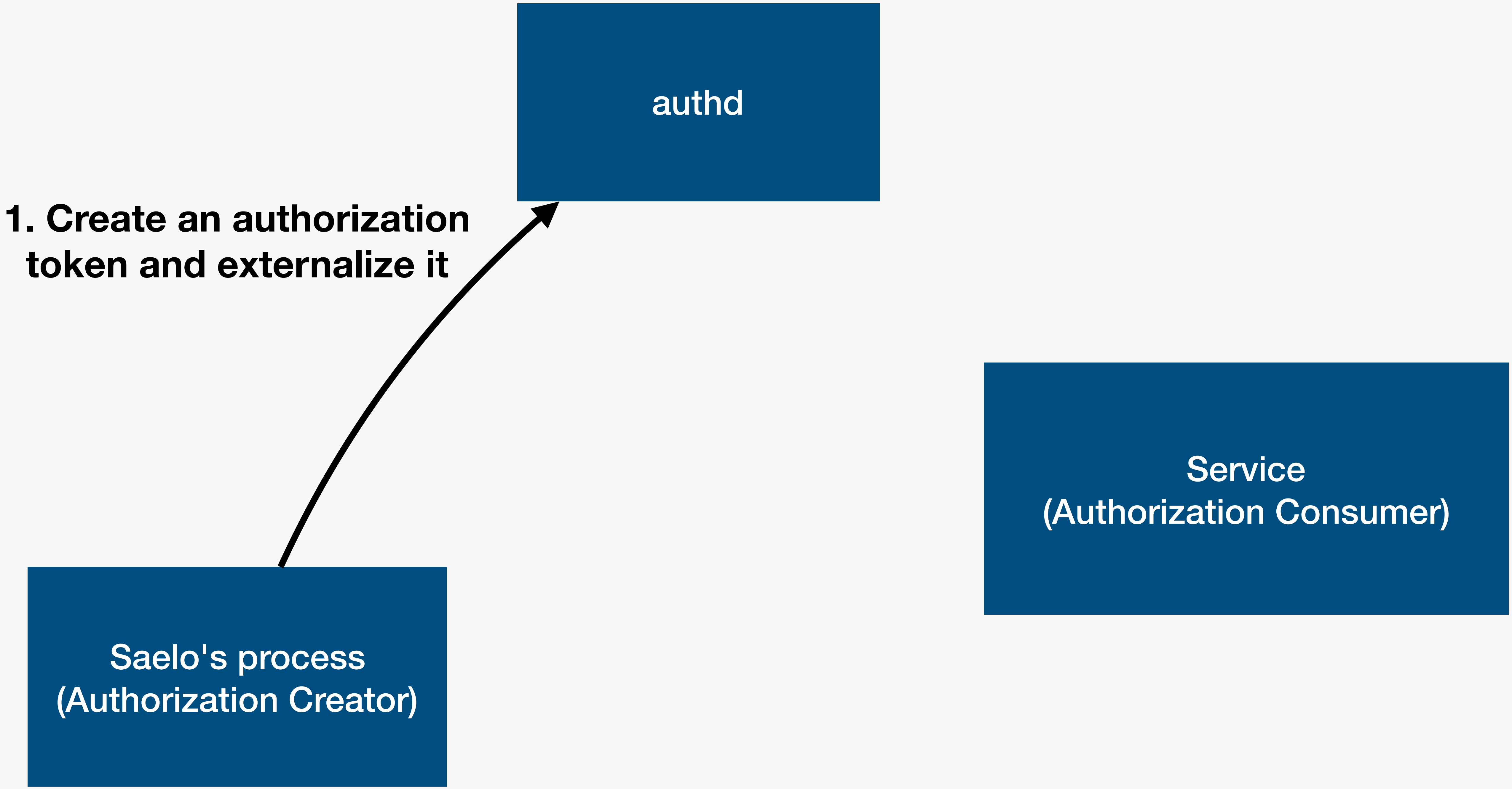
authd

Service
(Authorization Consumer)

Saelo's process
(Authorization Creator)

Token Database

External Form	Creator PID	Creator UID



authd

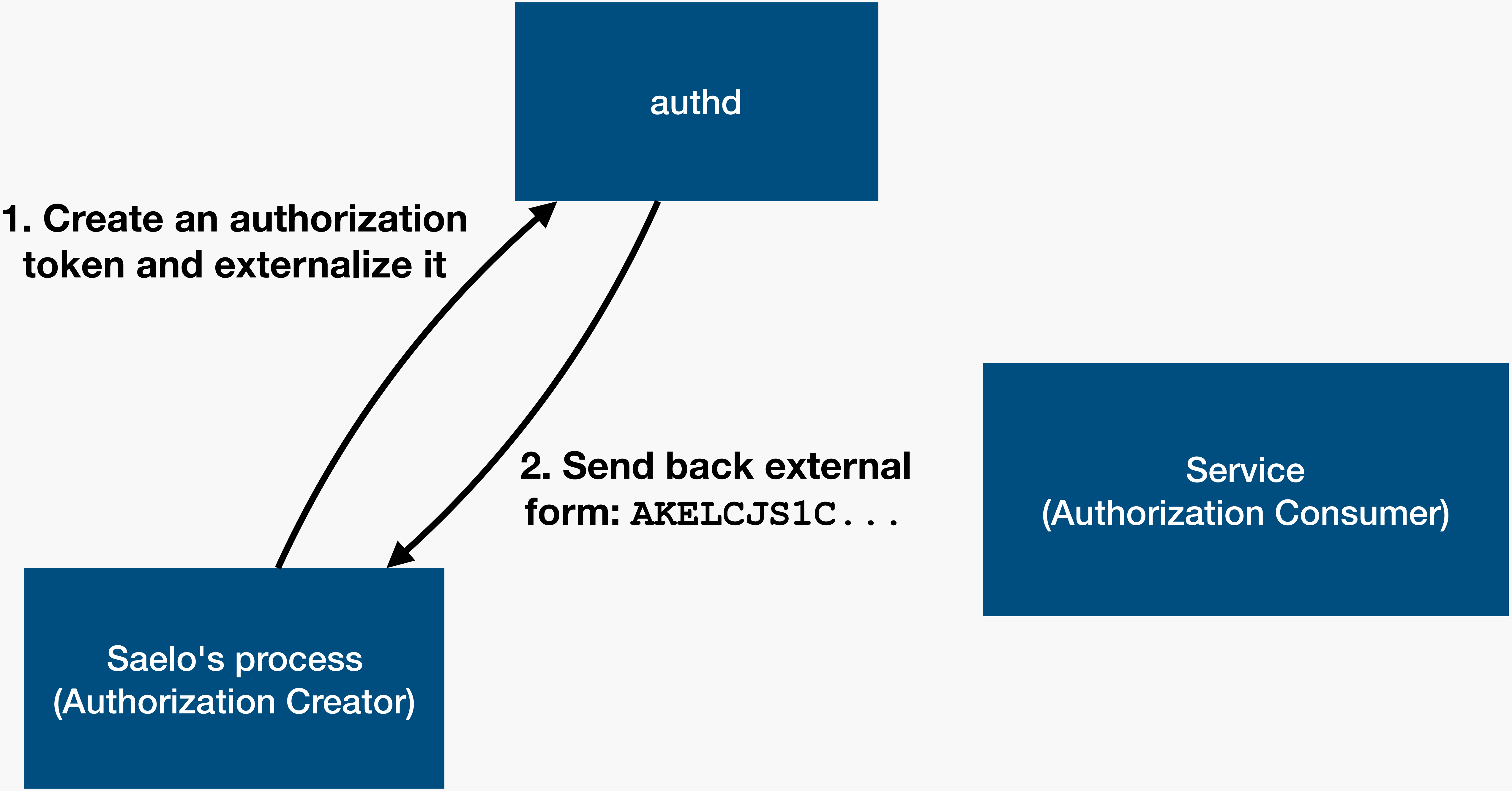
1. Create an authorization token and externalize it

Service (Authorization Consumer)

Saelo's process (Authorization Creator)

Token Database

External Form	Creator PID	Creator UID
AKELCJS1C...	1337	501



authd

1. Create an authorization token and externalize it

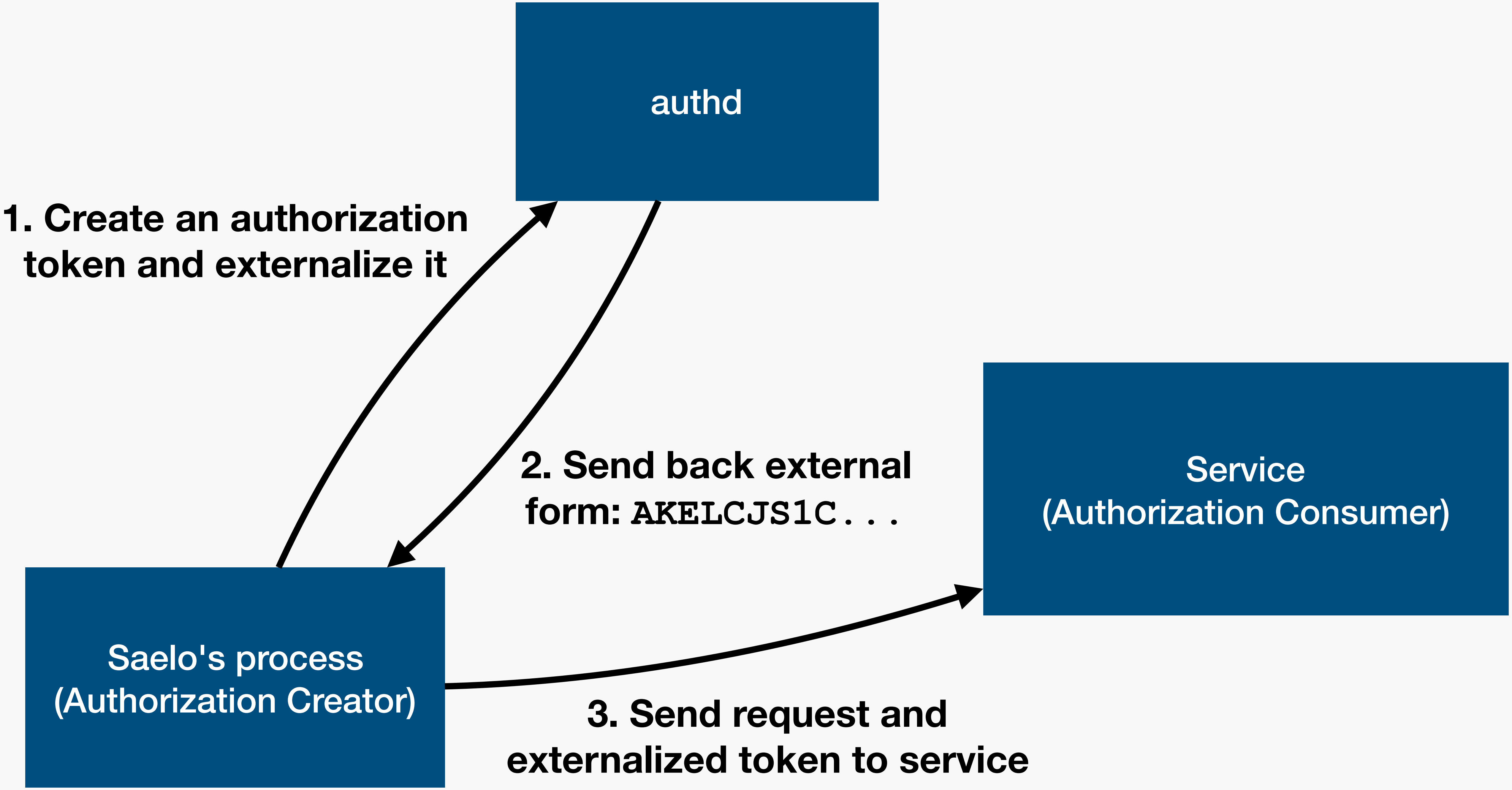
2. Send back external form: `AKELCJS1C...`

Service
(Authorization Consumer)

Saelo's process
(Authorization Creator)

Token Database

External Form	Creator PID	Creator UID
AKELCJS1C...	1337	501



authd

1. Create an authorization token and externalize it

2. Send back external form: `AKELCJS1C...`

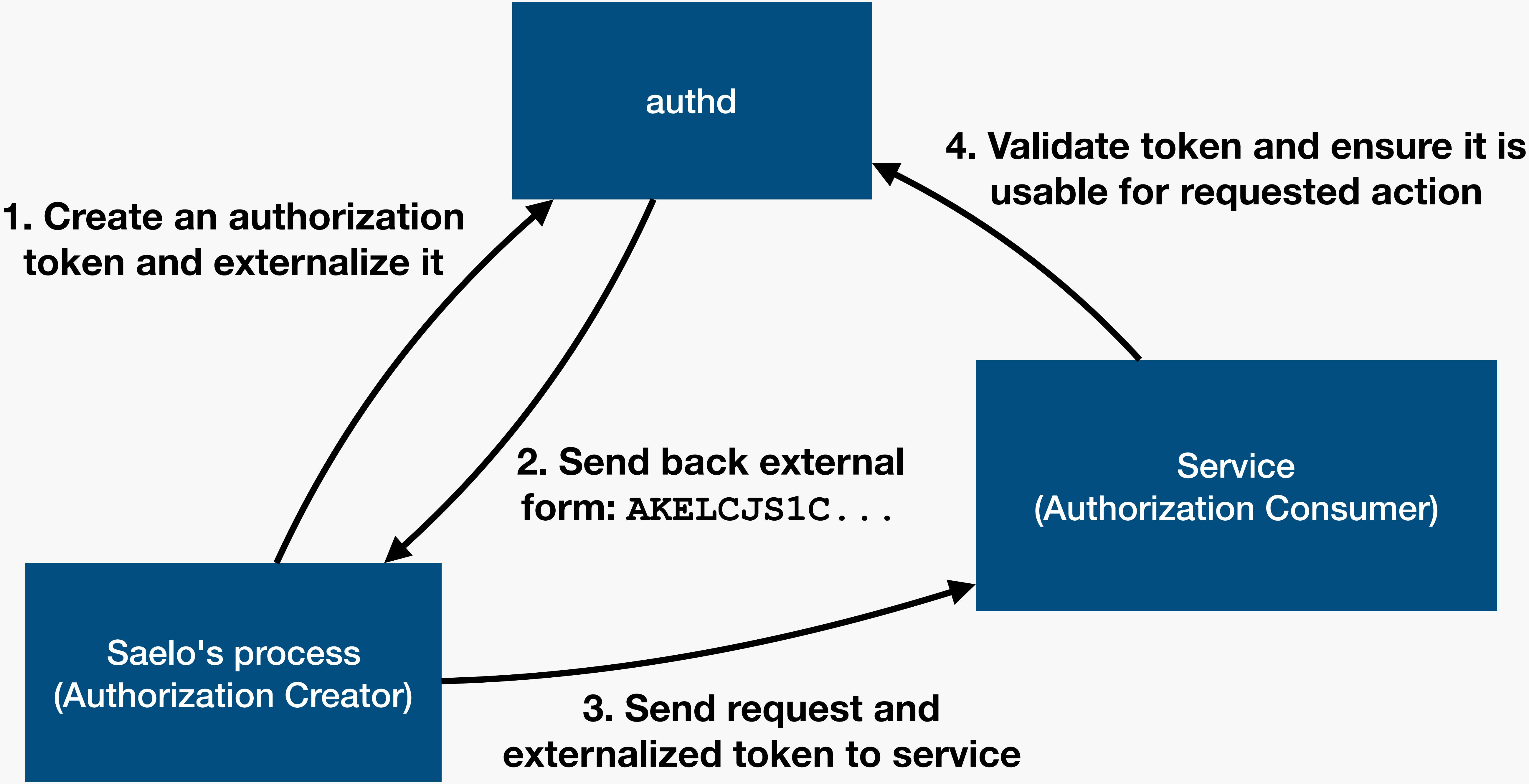
Service (Authorization Consumer)

Saelo's process (Authorization Creator)

3. Send request and externalized token to service

Token Database

External Form	Creator PID	Creator UID
AKELCJS1C...	1337	501



authd

1. Create an authorization token and externalize it

2. Send back external form: `AKELCJS1C...`

3. Send request and externalized token to service

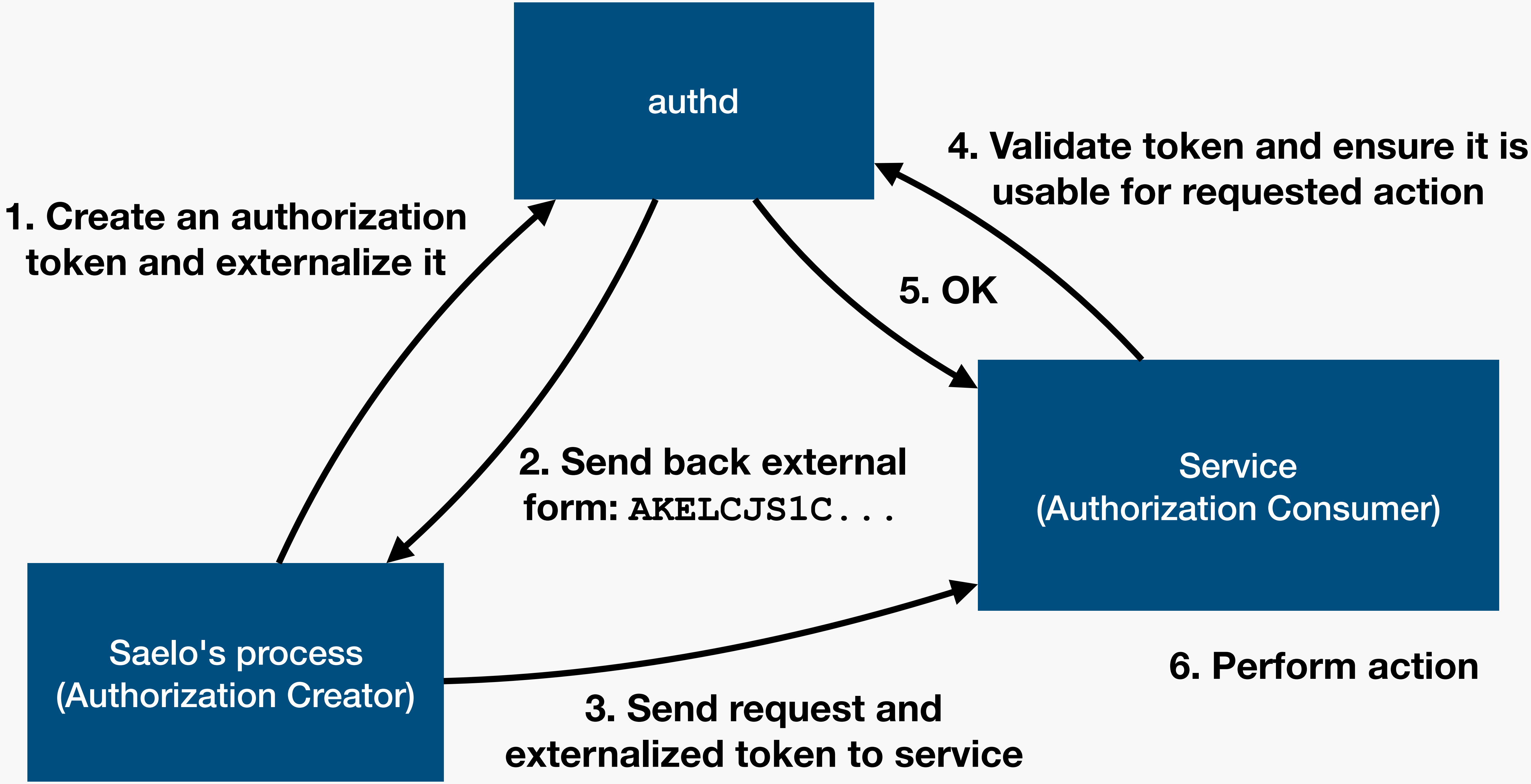
4. Validate token and ensure it is usable for requested action

Service (Authorization Consumer)

Saelo's process (Authorization Creator)

Token Database

External Form	Creator PID	Creator UID
AKELCJS1C...	1337	501



authd

Saelo's process
(Authorization Creator)

Service
(Authorization Consumer)

1. Create an authorization token and externalize it

2. Send back external form: `AKELCJS1C...`

3. Send request and externalized token to service

4. Validate token and ensure it is usable for requested action

5. OK

6. Perform action

```
> security authorizationdb read system.volume.
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>class</key>
        <string>rule</string>
        <key>comment</key>
        <string>system.volume.(external|internal|removable).
(adopt|encode|mount|rename|unmount)</string>
        <key>created</key>
        <real>455638795.69457</real>
        <key>k-of-n</key>
        <integer>1</integer>
        <key>modified</key>
        <real>455638795.69457</real>
        <key>rule</key>
        <array>
                <string>is-root</string>
                <string>is-admin</string>
                <string>authenticate-admin-30</string>
        </array>
        <key>version</key>
        <integer>0</integer>
</dict>
</plist>
```

From WebContent

Safari renderer runs as current user

=> **`is-admin`** rule is fulfilled

But, trying to obtain the right from within the renderer fails

🤔

```
static bool _verify_sandbox(engine_t engine, const char * right)
{
    pid_t pid = process_get_pid(engine->proc);
    if (sandbox_check(pid, "authorization-right-obtain", right))
{
        LOGE("Sandbox denied authorizing right, ...");
        return false;
    }

    pid = auth_token_get_pid(engine->auth);
    if (auth_token_get_sandboxed(engine->auth) &&
        sandbox_check(pid, "authorization-right-obtain", right))
{
        LOGE("Sandbox denied authorizing right, ...");
        return false;
    }

    return true;
}
```

authd source code before march 2017

Sandbox!

- Problem: authd only grants authorizations to non-sandboxed processes

- Authorization issuer as well as consumer must not be sandboxed

    - Or have the following in their sandbox profile:
      `(allow authorization-right-obtain (right-name "system.volume.internal.mount"))`

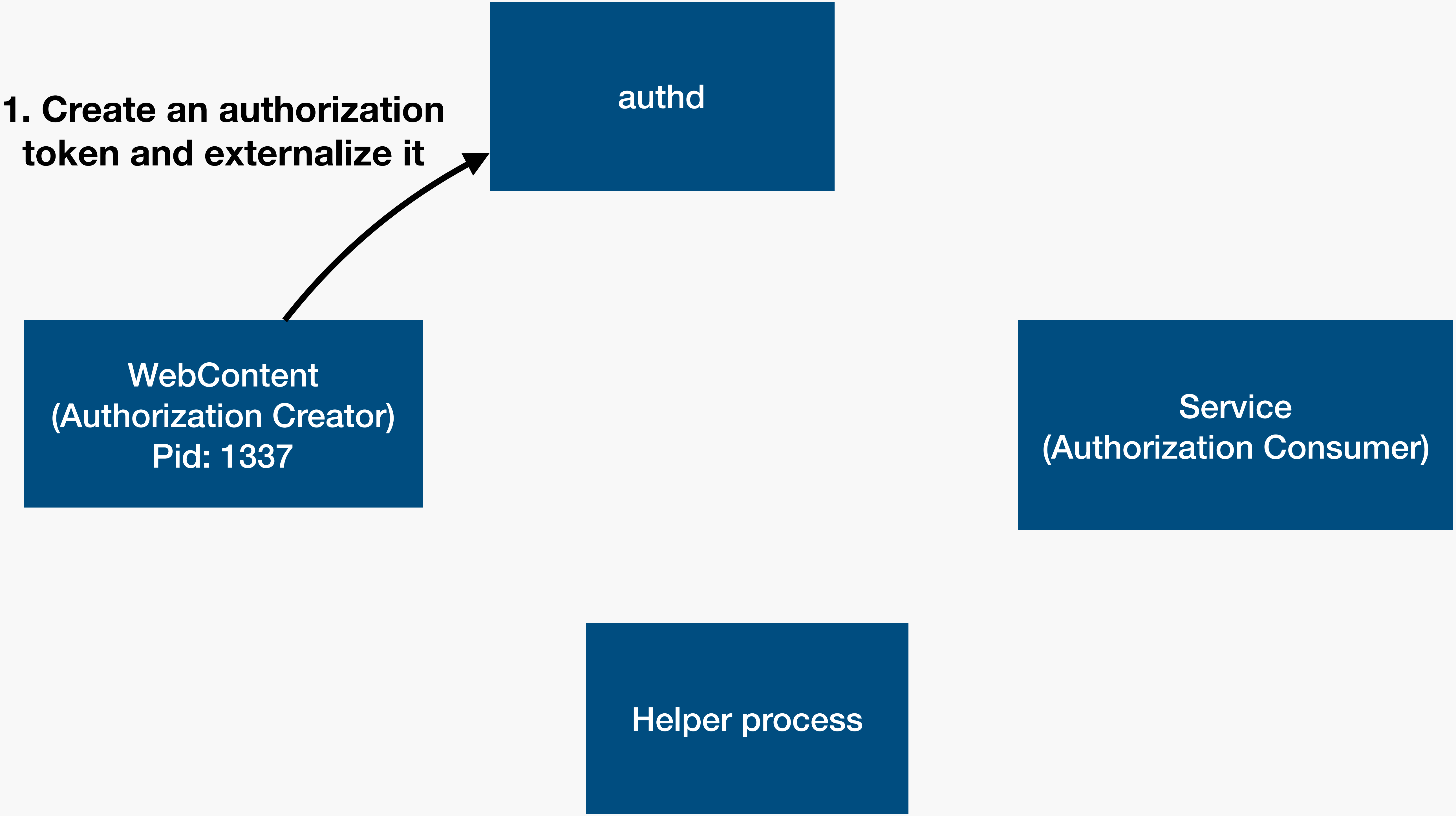
```
static bool _verify_sandbox(engine_t engine, const char * right)
{
    pid_t pid = process_get_pid(engine->proc);
    if (sandbox_check(pid, "authorization-right-obtain", right))
{
        LOGE("Sandbox denied authorizing right, ...");
        return false;
    }

    pid = auth_token_get_pid(engine->auth);
    if (auth_token_get_sandboxed(engine->auth) &&
        sandbox_check(pid, "authorization-right-obtain", right))
{
        LOGE("Sandbox denied authorizing right, ...");
        return false;
    }

    return true;
}
```

```
static bool _verify_sandbox(engine_t engine, const char * right)
{
    pid_t pid = process_get_pid(engine->proc);
    if (sandbox_check(pid, "authorization-right-obtain", right))
{
        LOGE("Sandbox denied authorizing right, ...");
        return false;
    }

    pid = auth_token_get_pid(engine->auth);
    if (auth_token_get_sandboxed(engine->auth) &&
        sandbox_check(pid, "authorization-right-obtain", right))
{
        LOGE("Sandbox denied authorizing right, ...");
        return false;
    }

    return true;
}
```

Problem: pid is taken from datastructure created when client first connected
=> can reuse PID! (CVE-2017-2535)

Token Database

External Form	Creator Pid	Creator UID



1. Create an authorization token and externalize it

authd

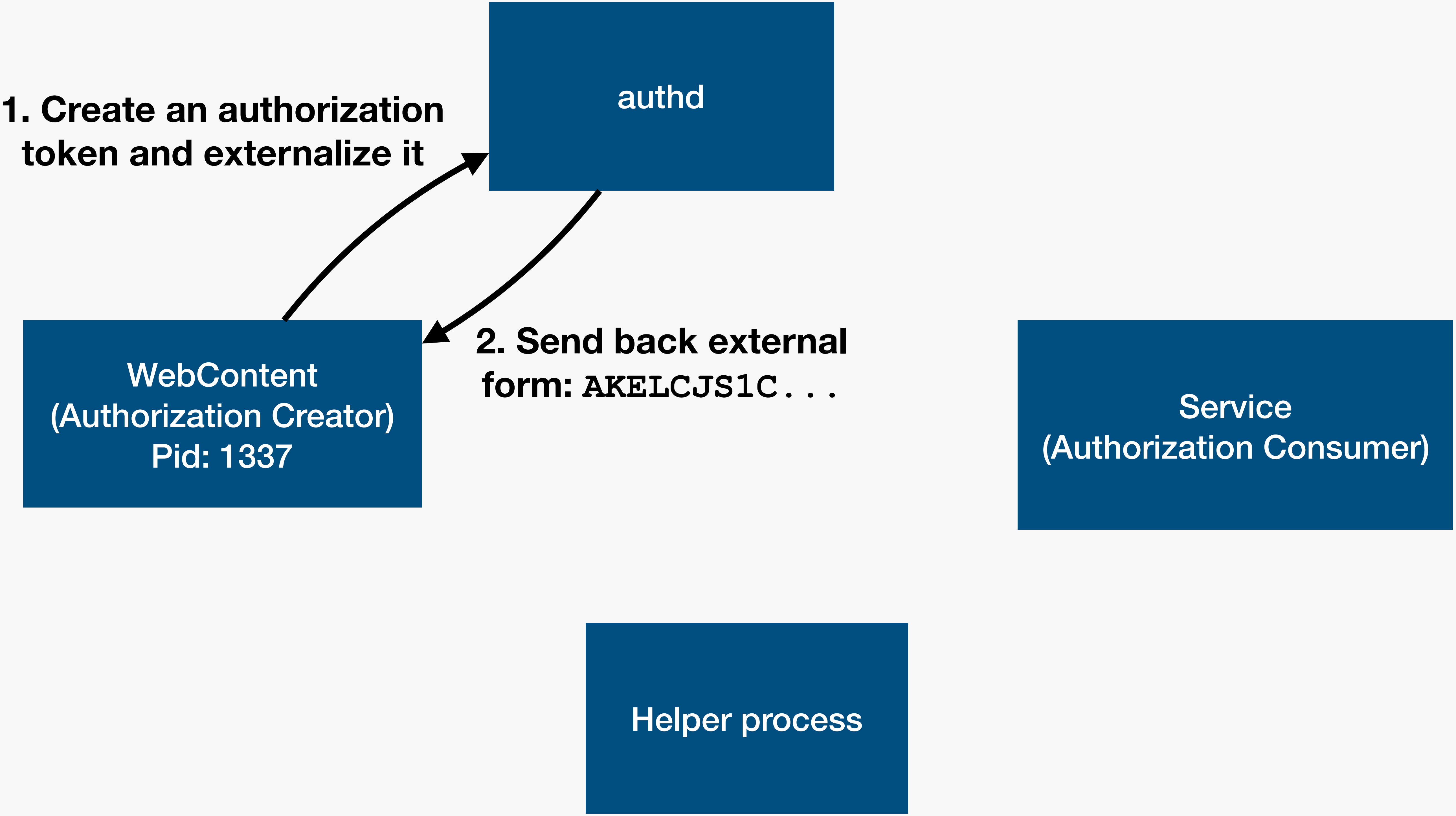
WebContent
(Authorization Creator)
Pid: 1337

Service
(Authorization Consumer)

Helper process

Token Database

External Form	Creator Pid	Creator UID
AKELCJS1C...	1337	501



authd

1. Create an authorization token and externalize it

WebContent
(Authorization Creator)
Pid: 1337

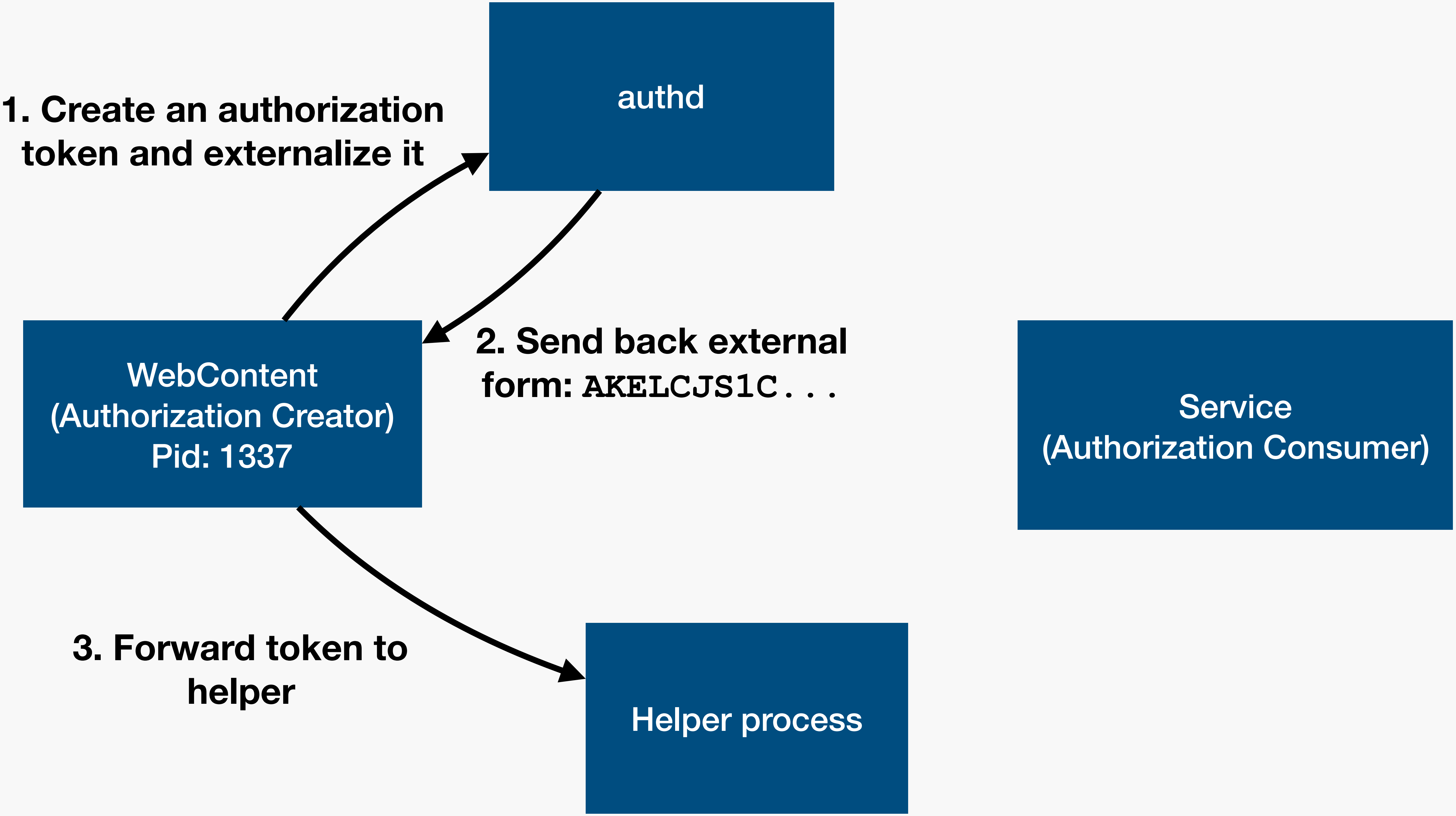
2. Send back external form: `AKELCJS1C...`

Service
(Authorization Consumer)

Helper process

Token Database

External Form	Creator Pid	Creator UID
AKELCJS1C...	1337	501



authd

1. Create an authorization token and externalize it

2. Send back external form: `AKELCJS1C...`

WebContent
(Authorization Creator)
Pid: 1337

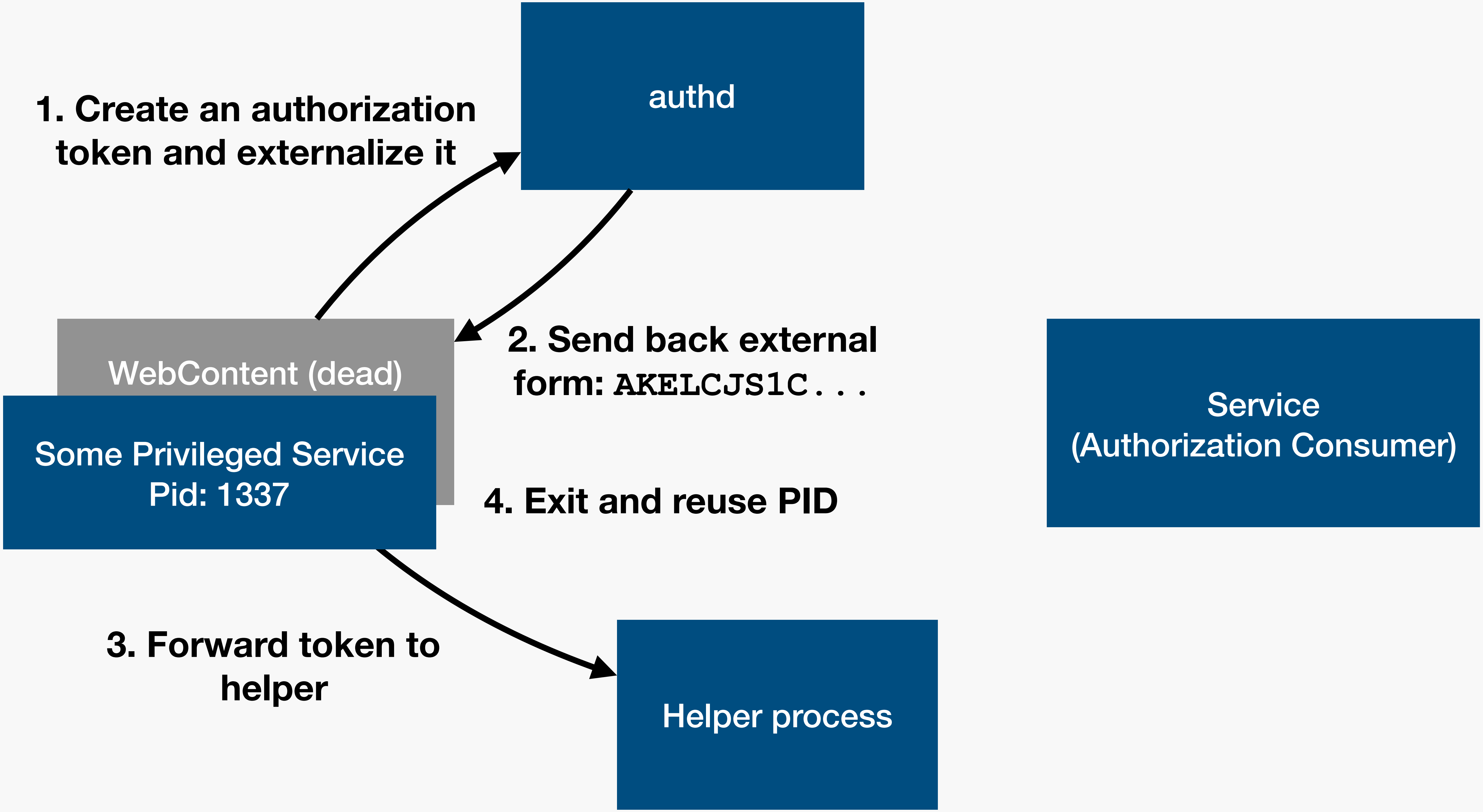
Service
(Authorization Consumer)

3. Forward token to helper

Helper process

Token Database

External Form	Creator Pid	Creator UID
AKELCJS1C...	1337	501



authd

1. **Create an authorization token and externalize it**

2. **Send back external form: `AKELCJS1C...`**

WebContent (dead)

Some Privileged Service
Pid: 1337

4. **Exit and reuse PID**

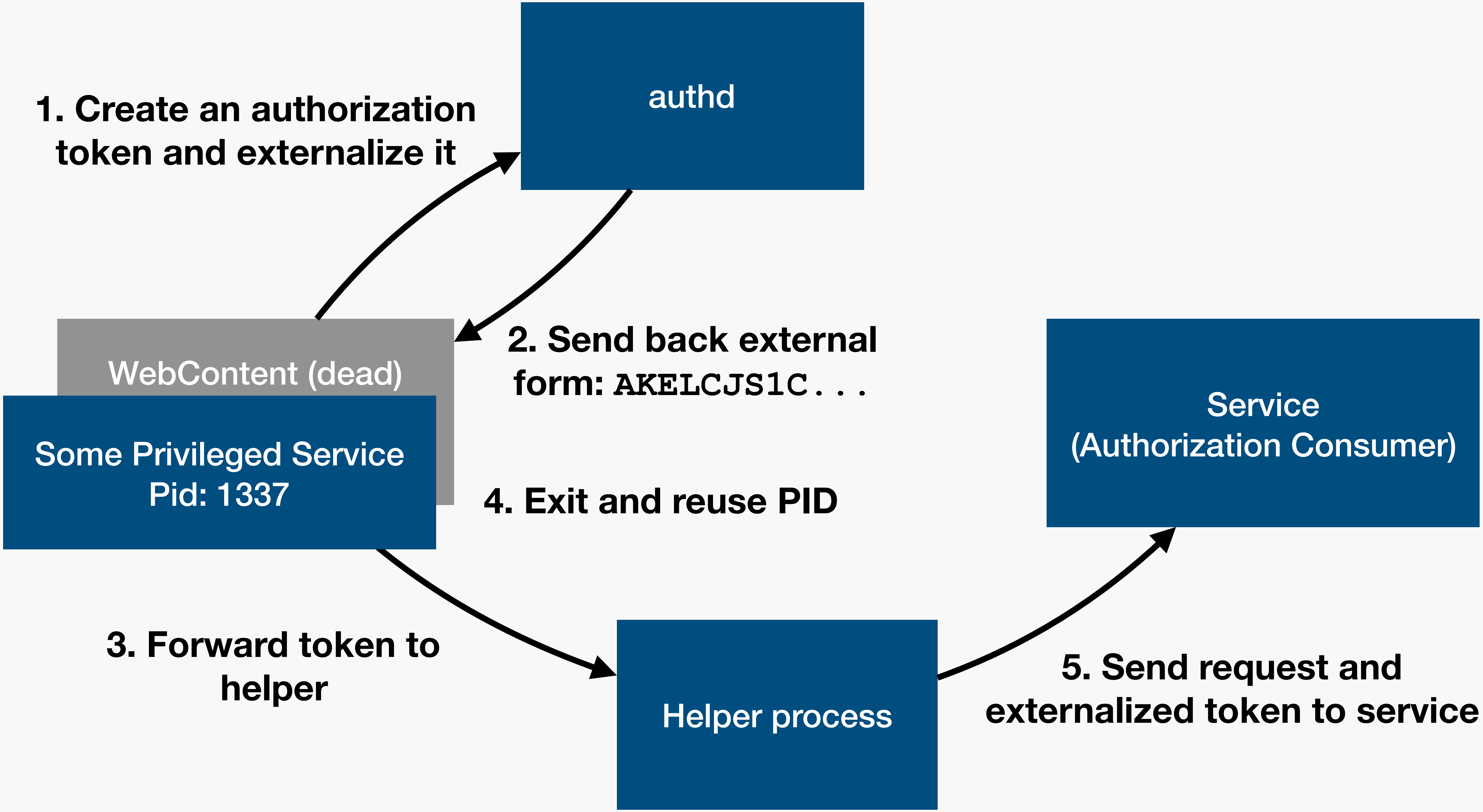
Service
(Authorization Consumer)

3. **Forward token to helper**

Helper process

Token Database

External Form	Creator Pid	Creator UID
AKELCJS1C...	1337	501



authd

1. Create an authorization token and externalize it

2. Send back external form: `AKELCJS1C...`

WebContent (dead)

Some Privileged Service
Pid: 1337

4. Exit and reuse PID

Service (Authorization Consumer)

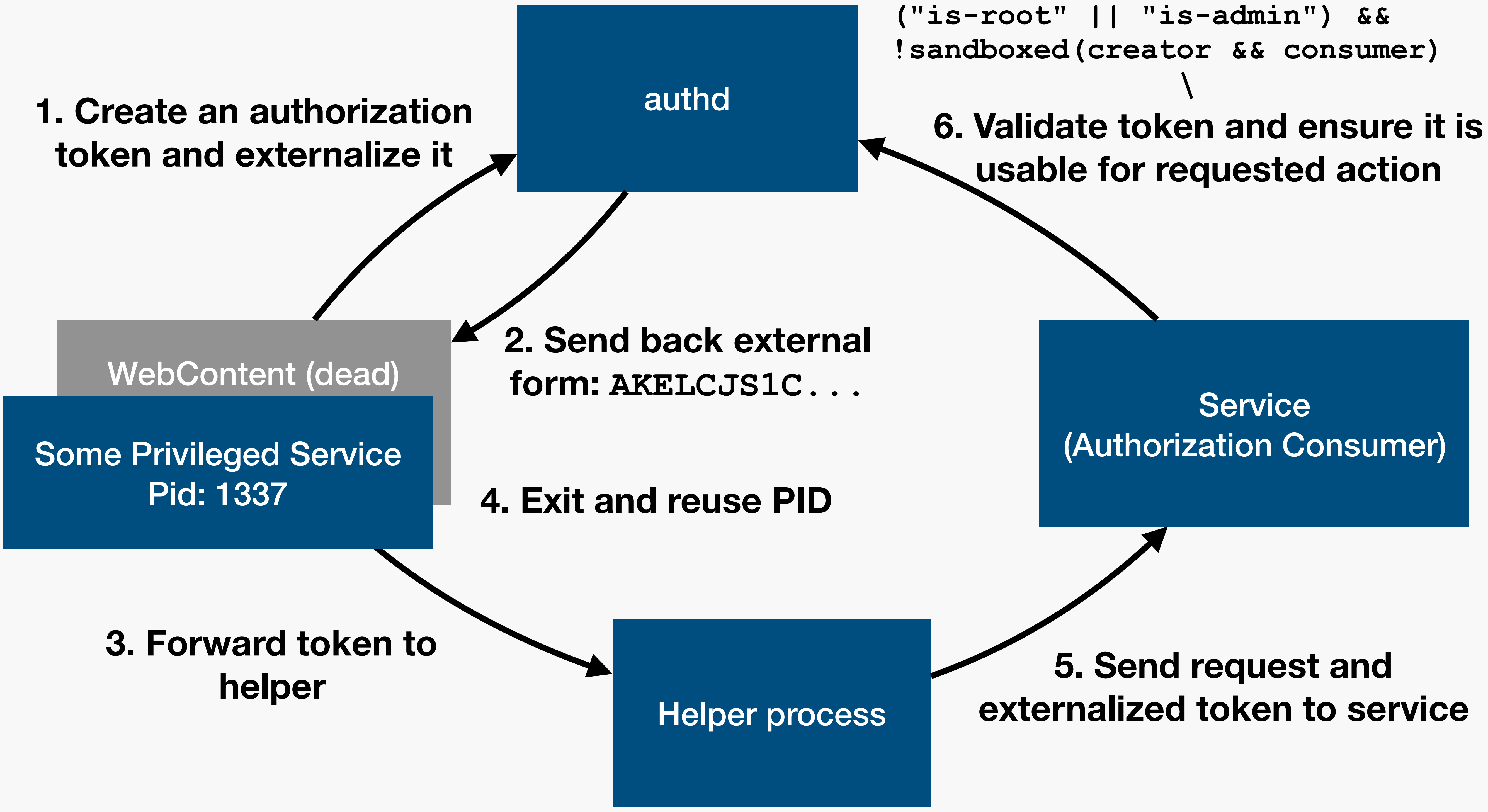
3. Forward token to helper

Helper process

5. Send request and externalized token to service

Token Database

External Form	Creator Pid	Creator UID
AKELCJS1C...	1337	501



```
("is-root" || "is-admin") &&
!sandboxed(creator && consumer)
```

authd

WebContent (dead)

Some Privileged Service
Pid: 1337

Helper process

Service
(Authorization Consumer)

1. Create an authorization token and externalize it

2. Send back external form: **`AKELCJS1C...`**

3. Forward token to helper

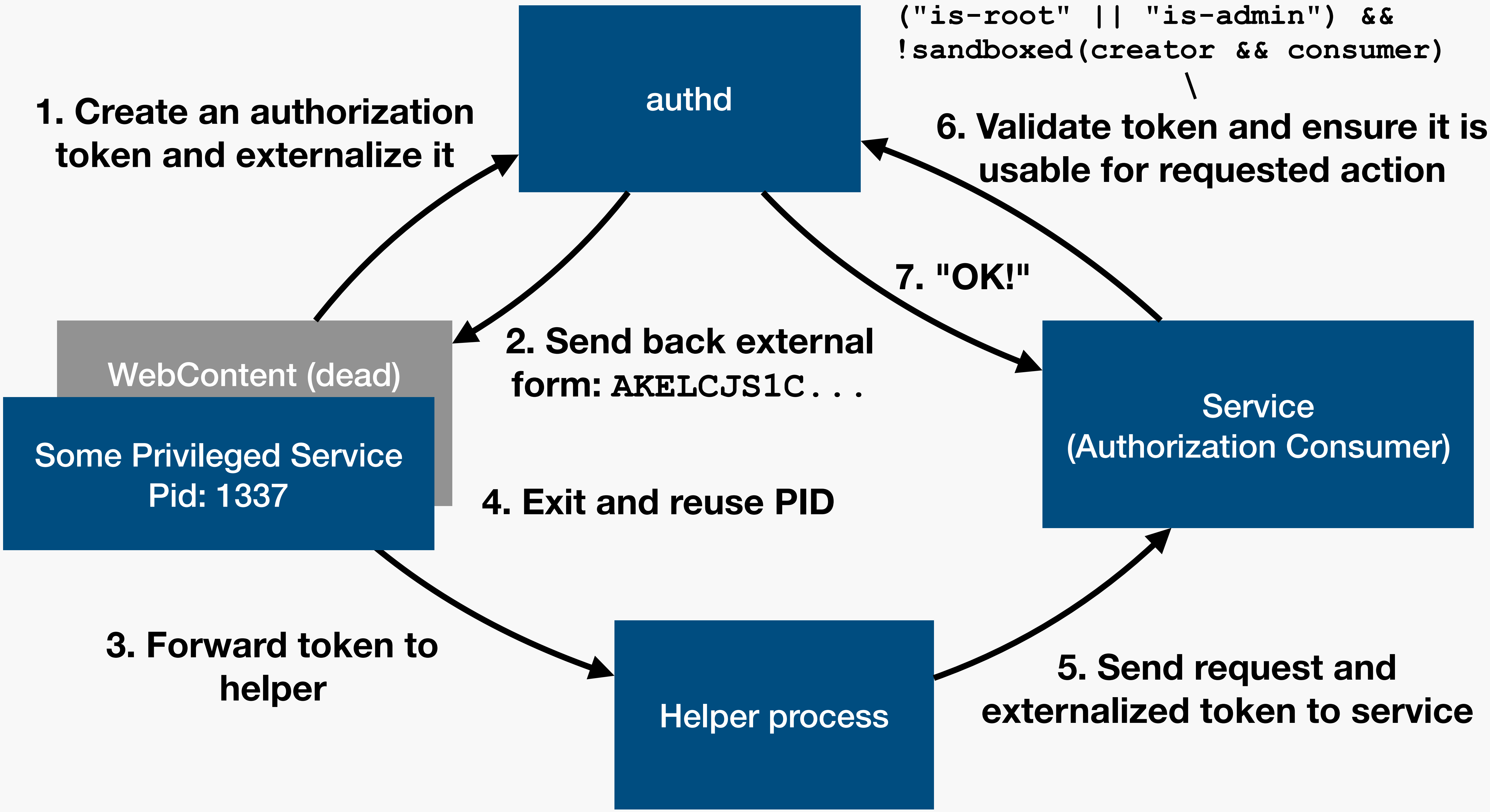
4. Exit and reuse PID

5. Send request and externalized token to service

6. Validate token and ensure it is usable for requested action

Token Database

External Form	Creator Pid	Creator UID
AKELCJS1C...	1337	501



authd

1. Create an authorization token and externalize it

2. Send back external form: `AKELCJS1C...`

WebContent (dead)

Some Privileged Service
Pid: 1337

3. Forward token to helper

4. Exit and reuse PID

Helper process

5. Send request and externalized token to service

Service
(Authorization Consumer)

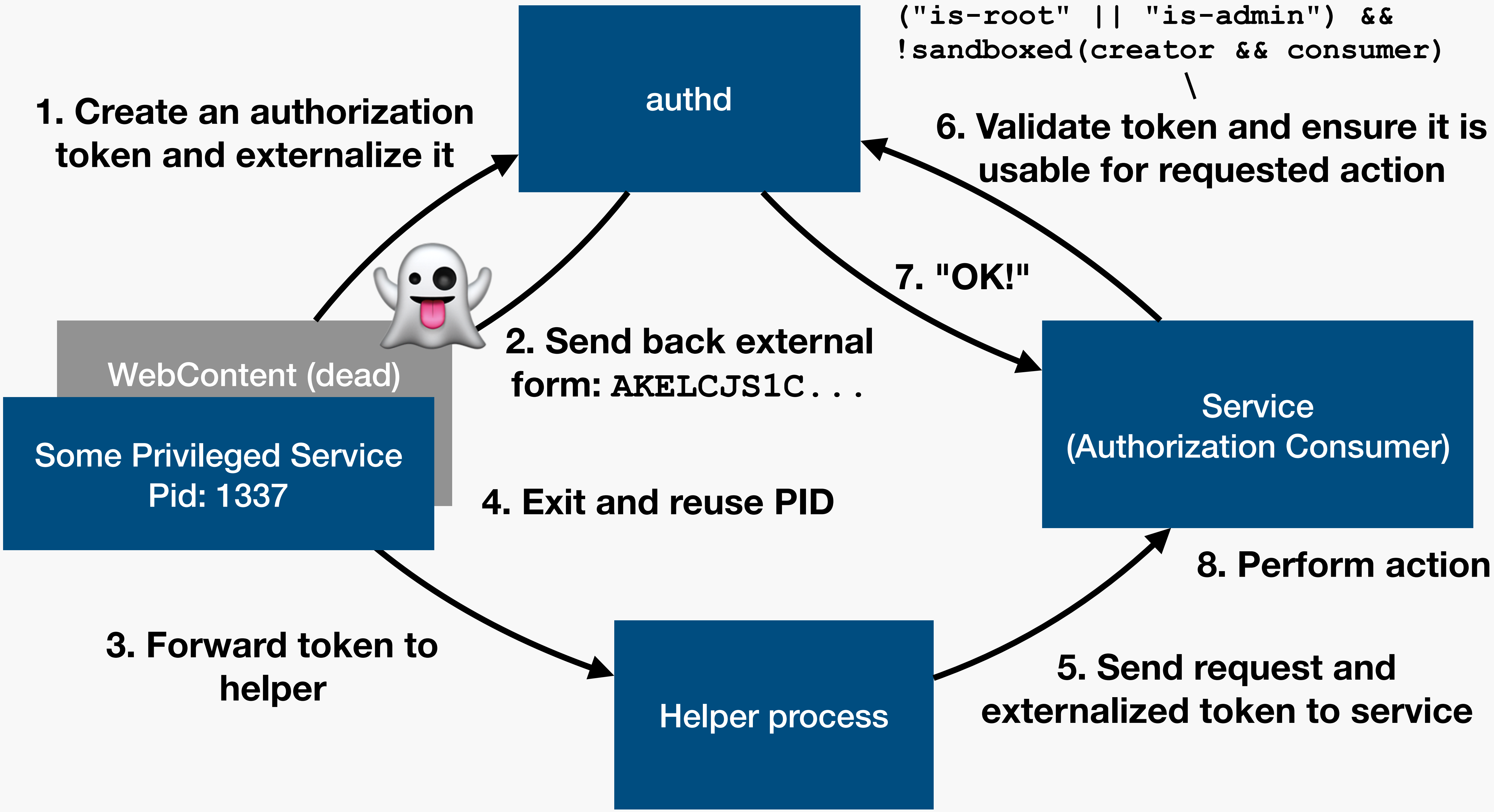
6. Validate token and ensure it is usable for requested action

```
("is-root" || "is-admin") &&
!sandboxed(creator && consumer)
```

7. "OK!"

Token Database

External Form	Creator Pid	Creator UID
AKELCJS1C...	1337	501



```
("is-root" || "is-admin") &&
!sandboxed(creator && consumer)
```

1. Create an authorization token and externalize it

2. Send back external form: `AKELCJS1C...`

3. Forward token to helper

4. Exit and reuse PID

5. Send request and externalized token to service

6. Validate token and ensure it is usable for requested action

7. "OK!"

8. Perform action

Final Exploit

- In our chain: helper process was speechsynthesisd which was allowed to fork and would load arbitrary .dylibs from a WebContent writable dir (CVE-2017-2534 by Niklas)

- Needed to crash a privileged service so it restarts and reclaims the PID => simple nullptr deref in nsurlstoraged

- Exploit implementation by Niklas: https://github.com/phoenhex/files/tree/master/exploits/safari-sbx

The Generic Issue

`sandbox_check` fundamentally broken

Race Conditions!

- Even if PID is not cached by the server, any security check that only uses the PID will likely still be insecure!
- Reason: there is a time window between sending the request in the client and handling the request in the server

  => Client can exit and another process can reclaim its PID

- Example: **`sandbox_check`** on macOS/iOS

sandbox_check

Darwin userland sandbox checking comes in two flavours:

- **`sandbox_check_by_audit_token`**
- **`sandbox_check(pid, ACTION)`**

This can't be safe...

CVEs ...

- Thought about presenting the Pwn2Own bug sometime
- Knew about `sandbox_check` weakness, figured I'd report it before talking about the Pwn2Own bug
- Not crazy serious, e.g. launchd always uses audit token

  => Wrote a half-hearted report in late 2017

  ...

About the security content of macOS 10.13.4

About the security content of iOS 11.3

```
curl -s  https://support.apple.com/en-us/HT208692 \
         https://support.apple.com/en-us/HT208693 \
    | grep 5aelo | sort -u | wc -l
```

>>> **9** <<<

😳 => 🤩

* Essentially apple assigned a CVE for every vulnerable service they found

Easy Exploit?

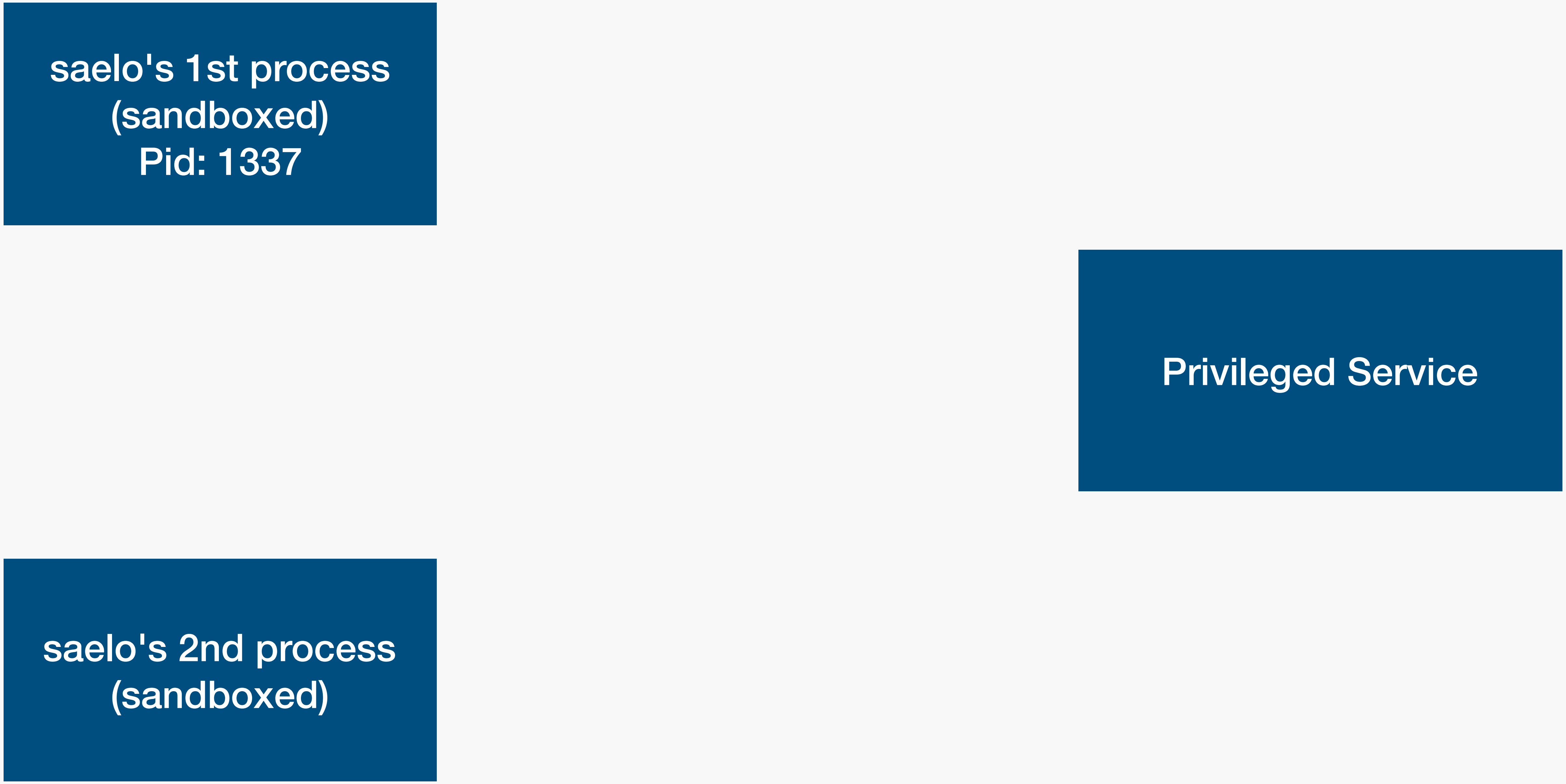
Problem: if the client dies, how can we receive a reply?

Solution: transfer mach IPC endpoint to other process!

Mach Messages

- Mach is the microkernel inside XNU
- Mach messages are the core IPC mechanism in Darwin
    - Many other IPC mechanisms built on top, notably XPC
    - Topic of many presentations, blog posts, etc.
- Unidirectional, relies on mach ports as endpoints
- Cool feature: ports can be transferred to other processes!

The Final Attack



saelo's 1st process
(sandboxed)
Pid: 1337

Privileged Service

saelo's 2nd process
(sandboxed)

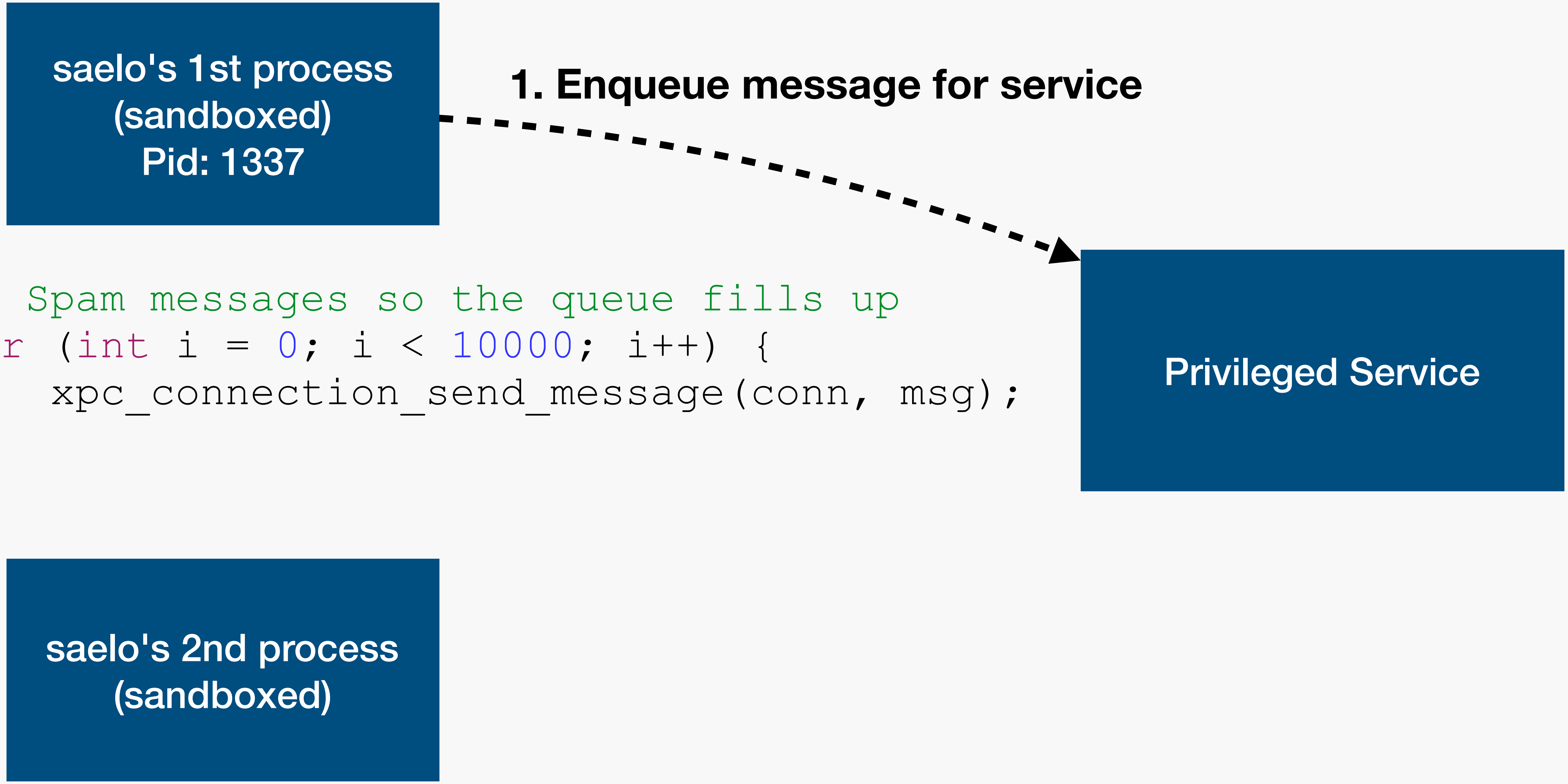
saelo's 1st process
(sandboxed)
Pid: 1337

Needs either `(allow process-fork)` or some patience while crashing and respawning IPC services ;)

Preparation:
PIDs are wrapped around so next free PID is just before 1337

Privileged Service

saelo's 2nd process
(sandboxed)



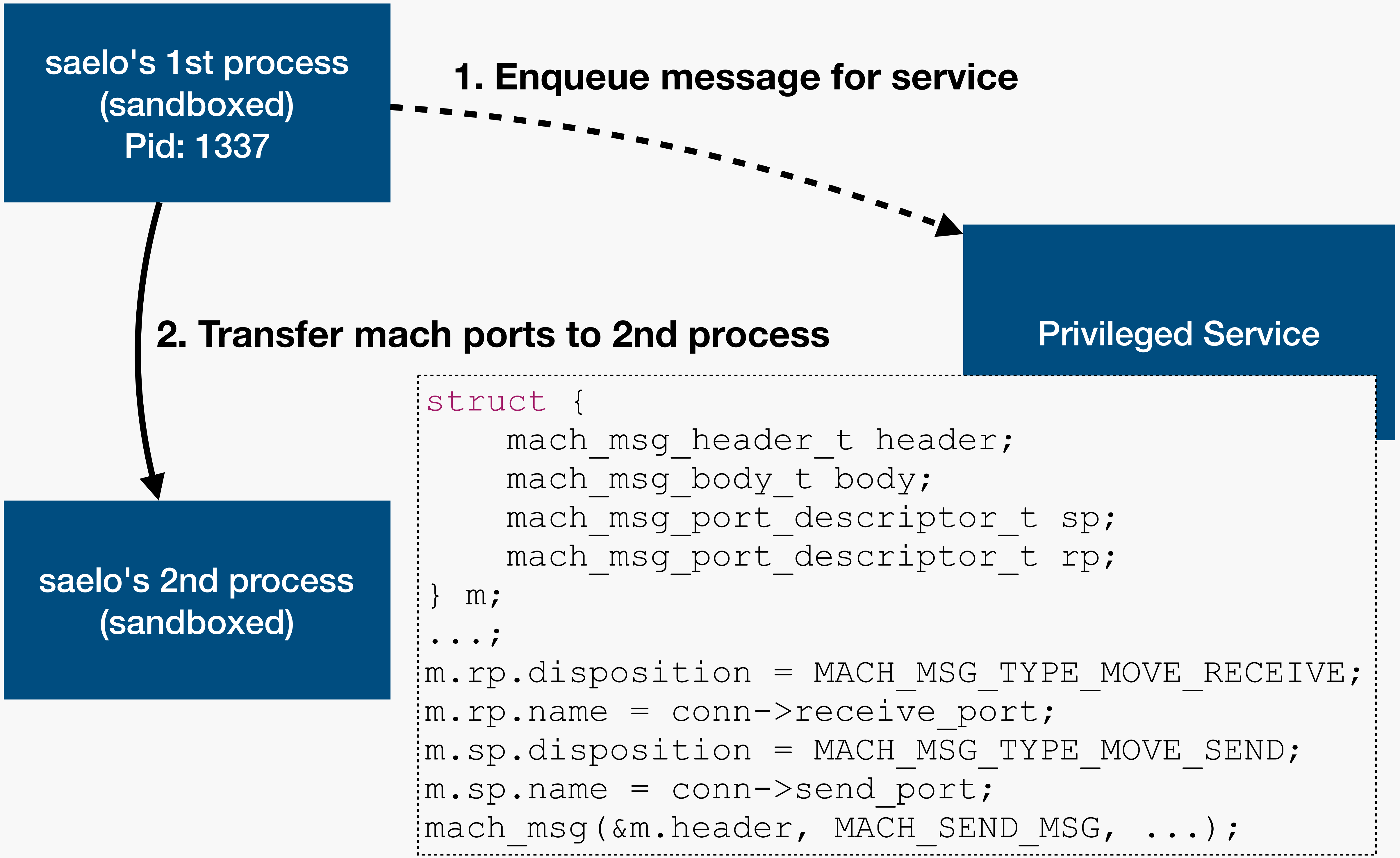
saelo's 1st process
(sandboxed)
Pid: 1337

1. Enqueue message for service

Privileged Service

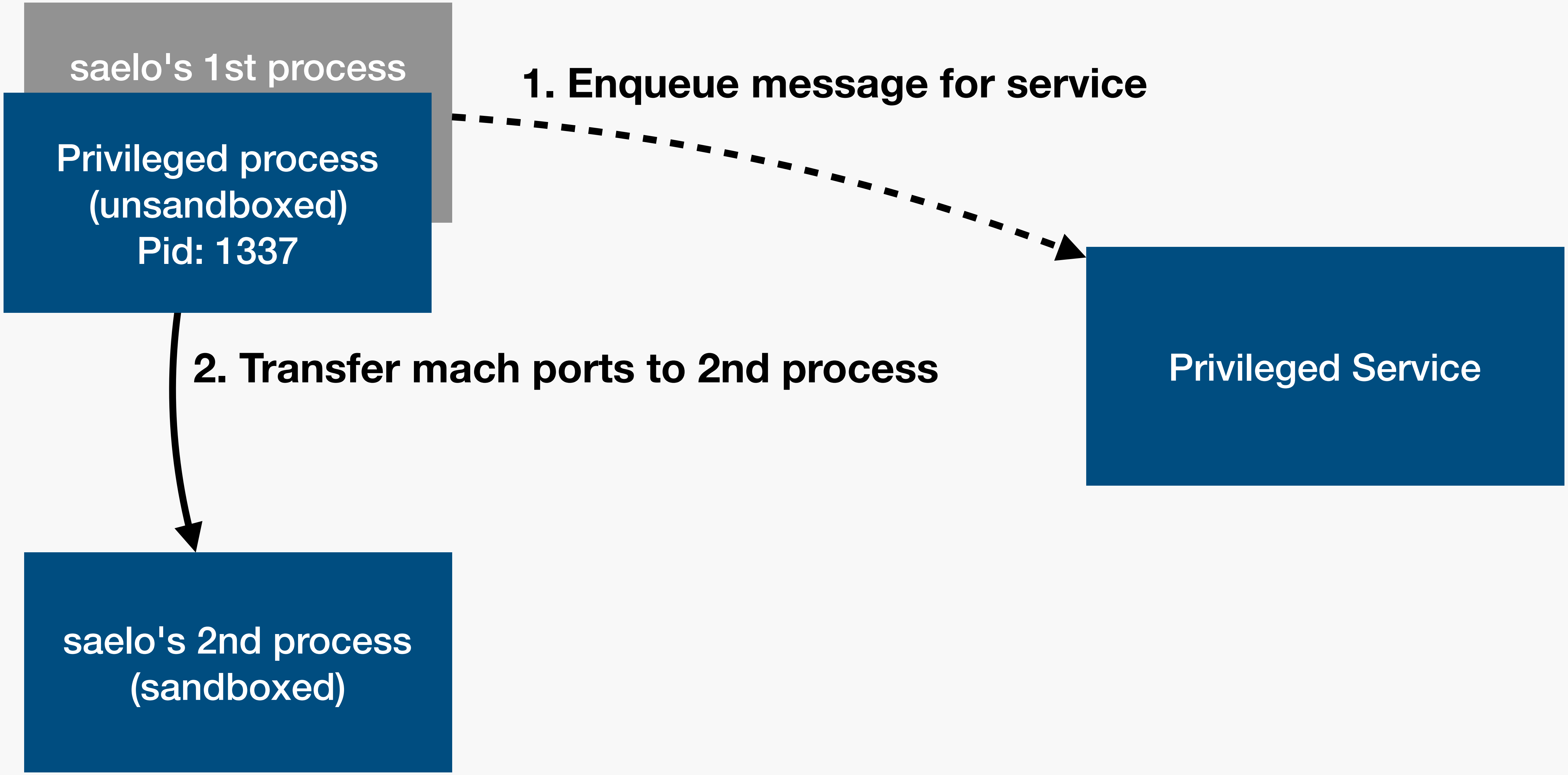
```
// Spam messages so the queue fills up
for (int i = 0; i < 10000; i++) {
    xpc_connection_send_message(conn, msg);
}
```

saelo's 2nd process
(sandboxed)



```
struct {
    mach_msg_header_t header;
    mach_msg_body_t body;
    mach_msg_port_descriptor_t sp;
    mach_msg_port_descriptor_t rp;
} m;
...;
m.rp.disposition = MACH_MSG_TYPE_MOVE_RECEIVE;
m.rp.name = conn->receive_port;
m.sp.disposition = MACH_MSG_TYPE_MOVE_SEND;
m.sp.name = conn->send_port;
mach_msg(&m.header, MACH_SEND_MSG, ...);
```

3. First process dies and some unsandboxed process (spawned by the other process) reclaims its PID



saelo's 1st process

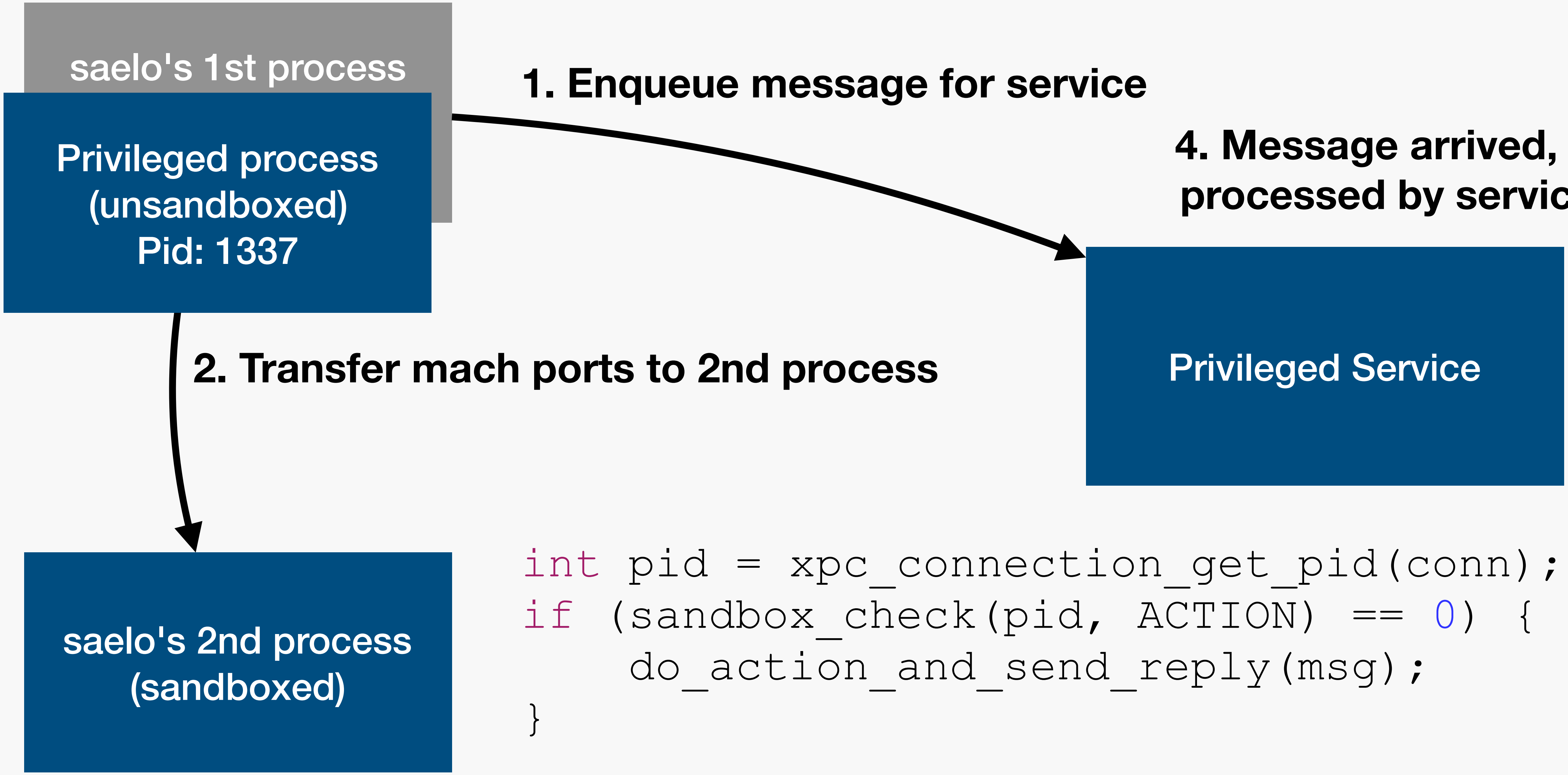
Privileged process
(unsandboxed)
Pid: 1337

1. Enqueue message for service

2. Transfer mach ports to 2nd process

Privileged Service

saelo's 2nd process
(sandboxed)



3. First process dies and some unsandboxed process (spawned by the other process) reclaims its PID

saelo's 1st process

Privileged process
(unsandboxed)
Pid: 1337

1. Enqueue message for service

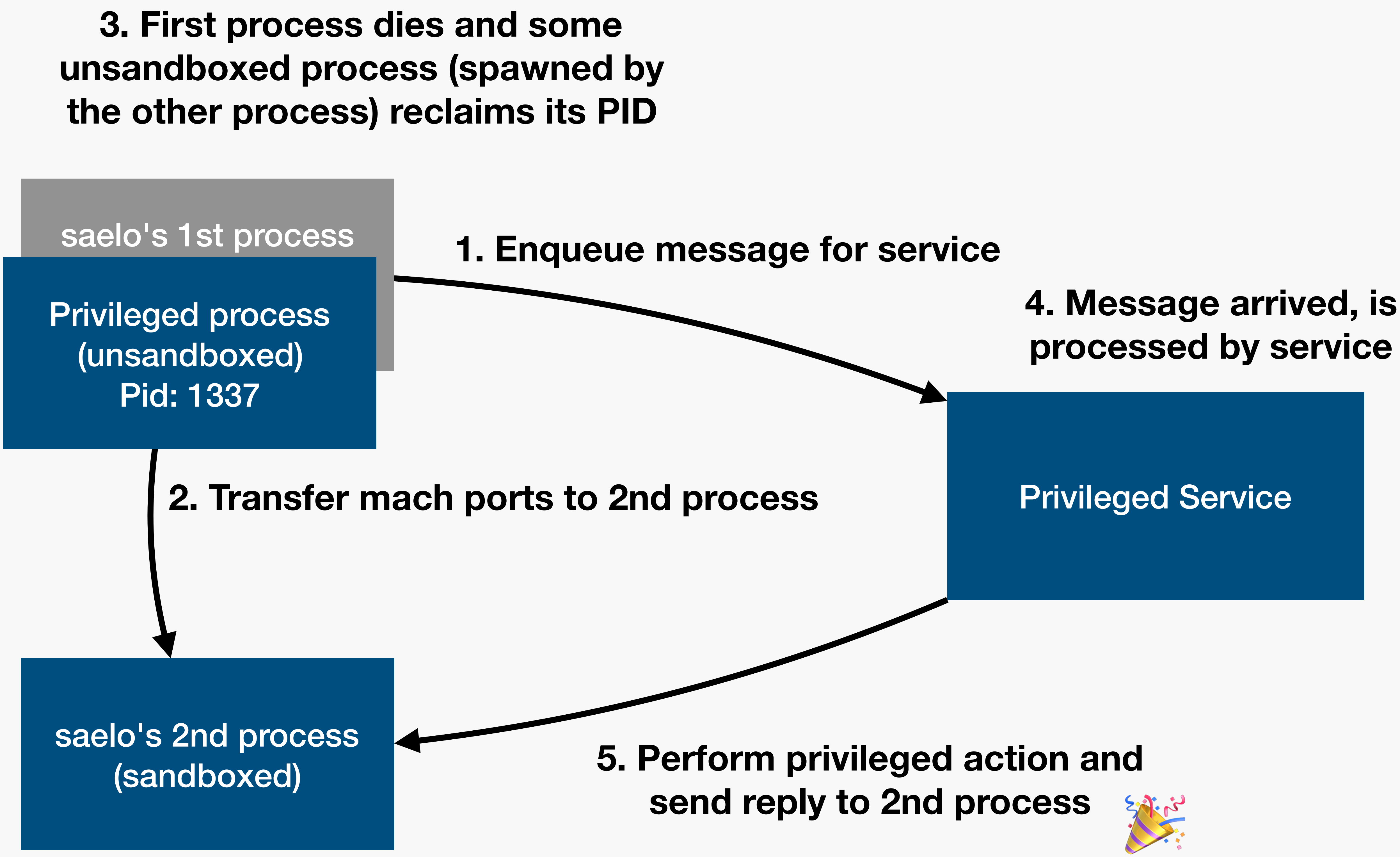
4. Message arrived, is processed by service

Privileged Service

2. Transfer mach ports to 2nd process

saelo's 2nd process
(sandboxed)

```
int pid = xpc_connection_get_pid(conn);
if (sandbox_check(pid, ACTION) == 0) {
    do_action_and_send_reply(msg);
}
```



3. First process dies and some unsandboxed process (spawned by the other process) reclaims its PID

saelo's 1st process

Privileged process
(unsandboxed)
Pid: 1337

1. Enqueue message for service

4. Message arrived, is processed by service

2. Transfer mach ports to 2nd process

Privileged Service

saelo's 2nd process
(sandboxed)

5. Perform privileged action and send reply to 2nd process 🎉

Summary

Don't use the PID for security checks :)

References

Our writeup for the Pwn2Own '17 chain:

- https://phoenhex.re/2017-07-06/pwn2own-sandbox-escape#performing-the-right-check-on-the-wrong-process

Similar bugs discovered by Project Zero in 2017:

- macOS userland entitlement checks: https://bugs.chromium.org/p/project-zero/issues/detail?id=1223
- Android KeyStore: https://bugs.chromium.org/p/project-zero/issues/detail?id=1406

Probably more... ?