What is Linux Kernel keystore and why you should use it in your next application

Ignat Korchagin
@ignatkn

CLOUDFLARE

$ whoami

- Linux team at Cloudflare
- Systems security and performance
- Low-level programming

$ whoami

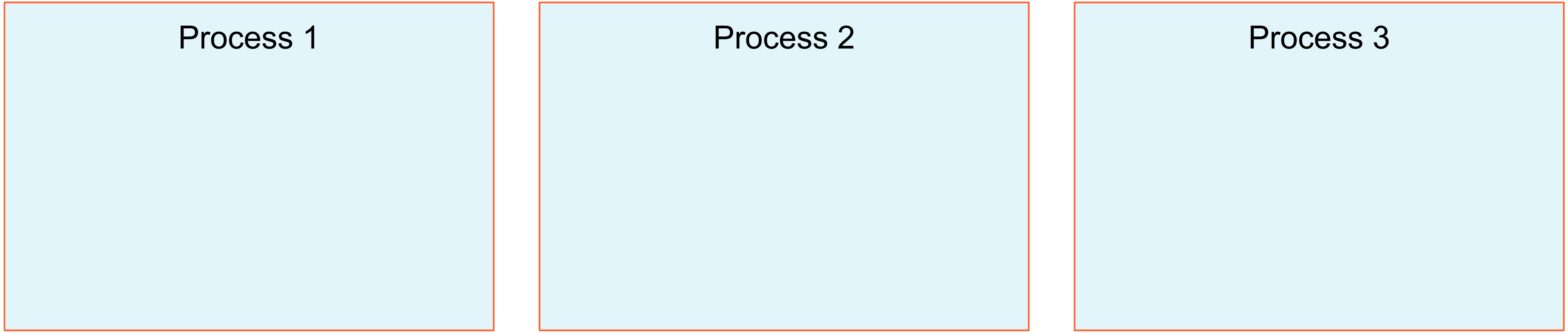
- Linux team at Cloudflare
- Systems security and performance
- Low-level programming
- *Fugitive programmer (US NSA banned C/C++)*

Application keys in memory

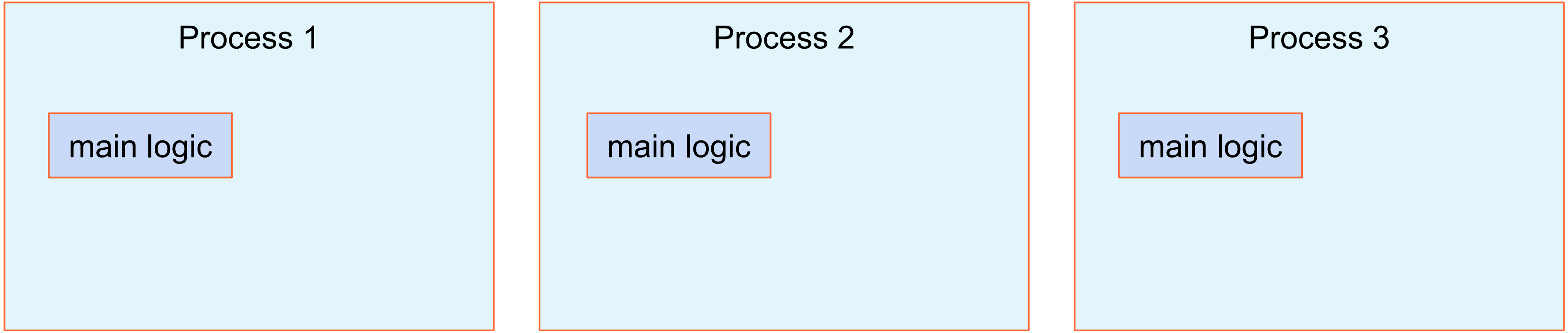
"NSA recommends that organizations use memory safe languages when possible and bolster protection through code-hardening defenses such as compiler options, tool options, and operating system configurations."

https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3215760/nsa-releases-guidance-on-how-to-protect-against-software-memory-safety-issues/

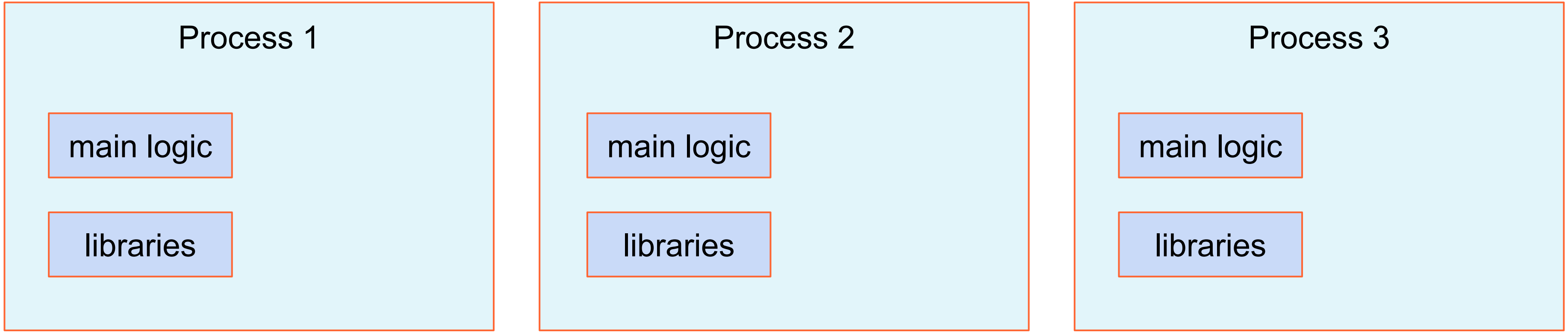
Linux address spaces



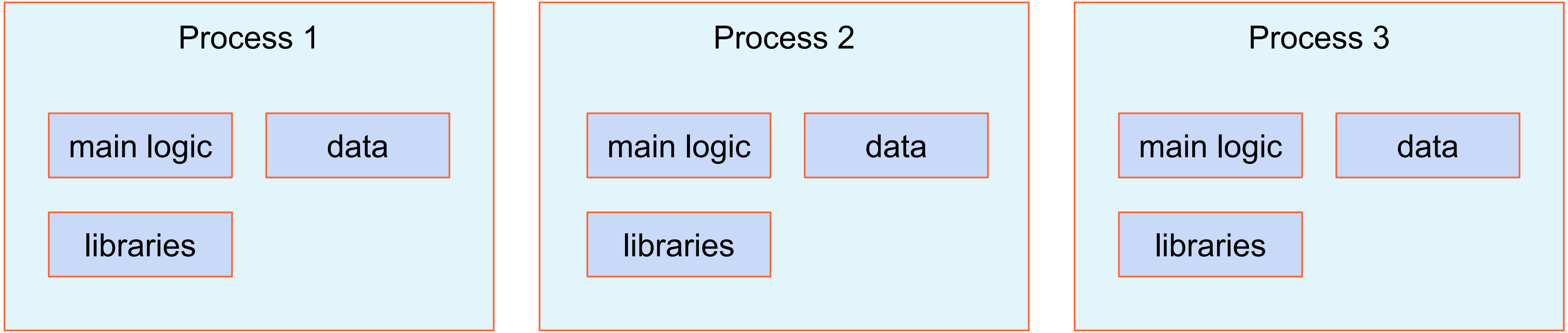
Linux address spaces



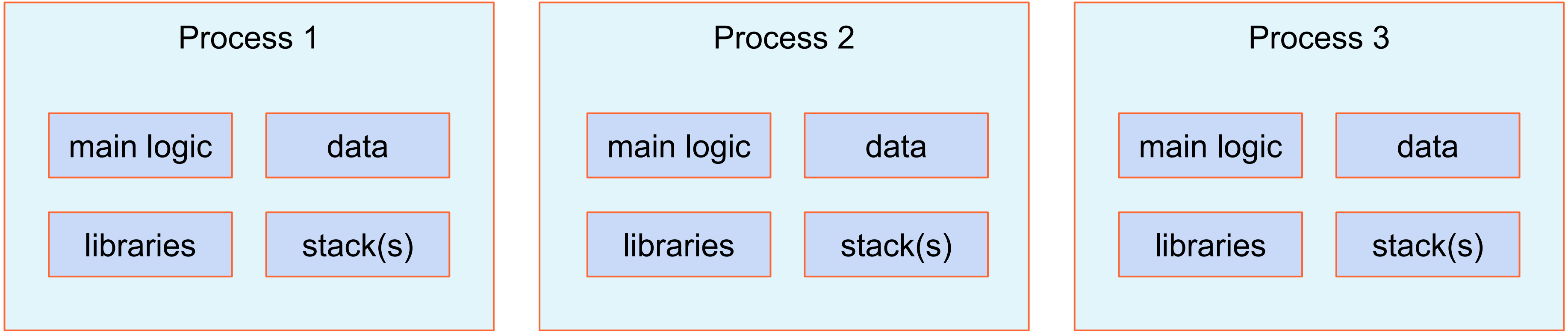
Linux address spaces



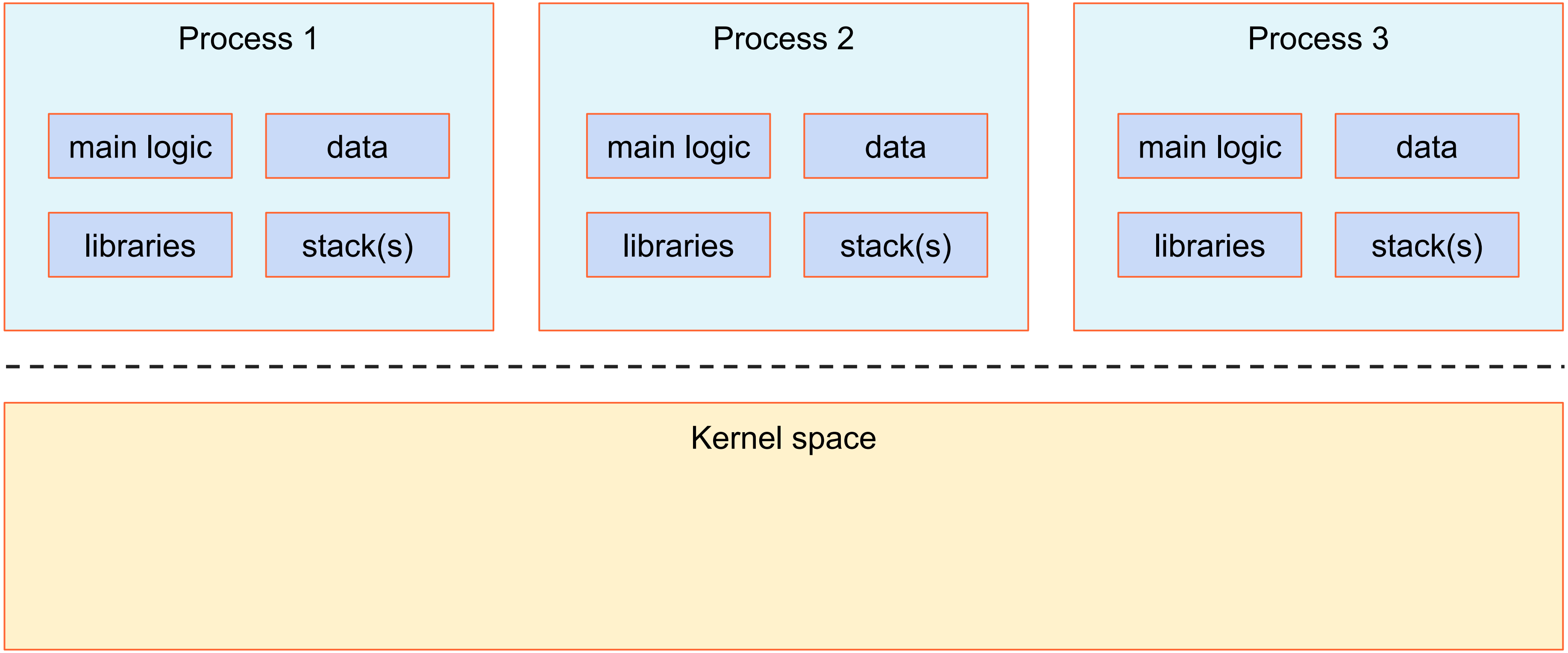
Linux address spaces



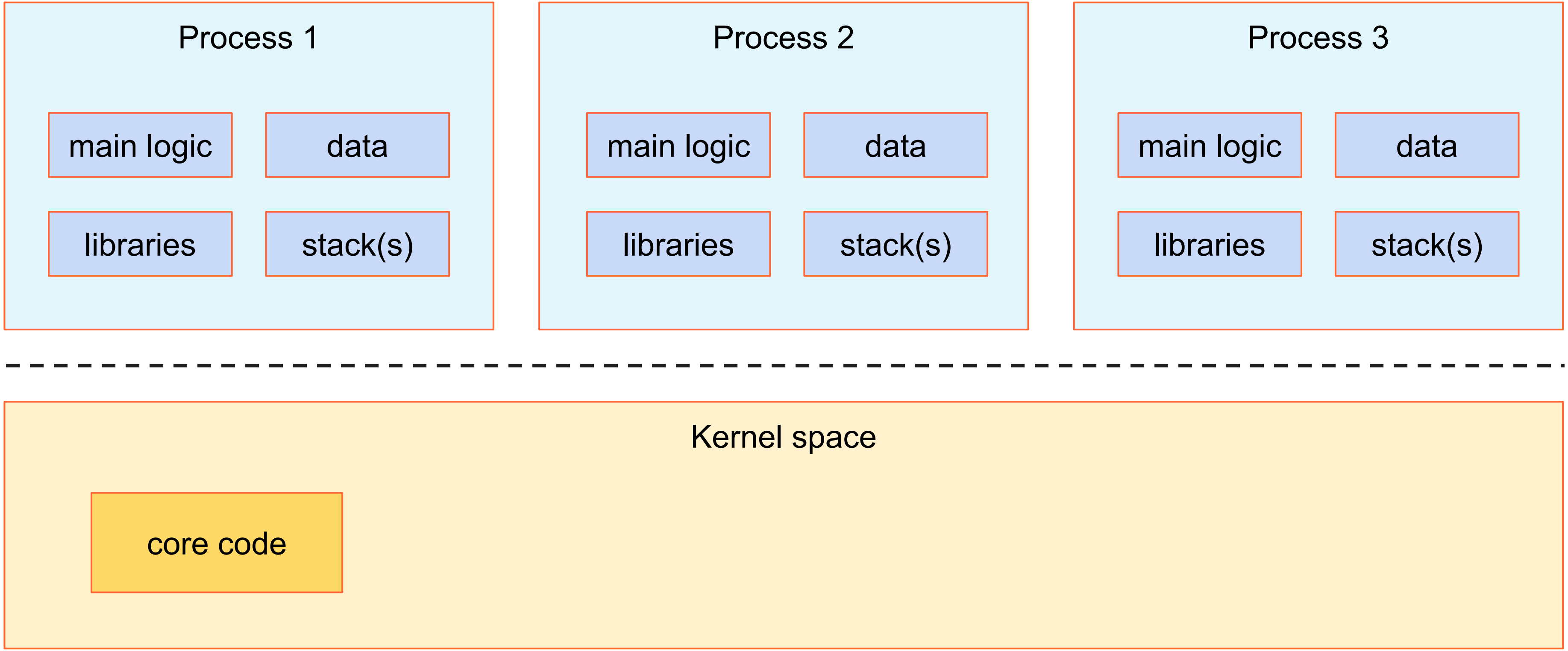
Linux address spaces



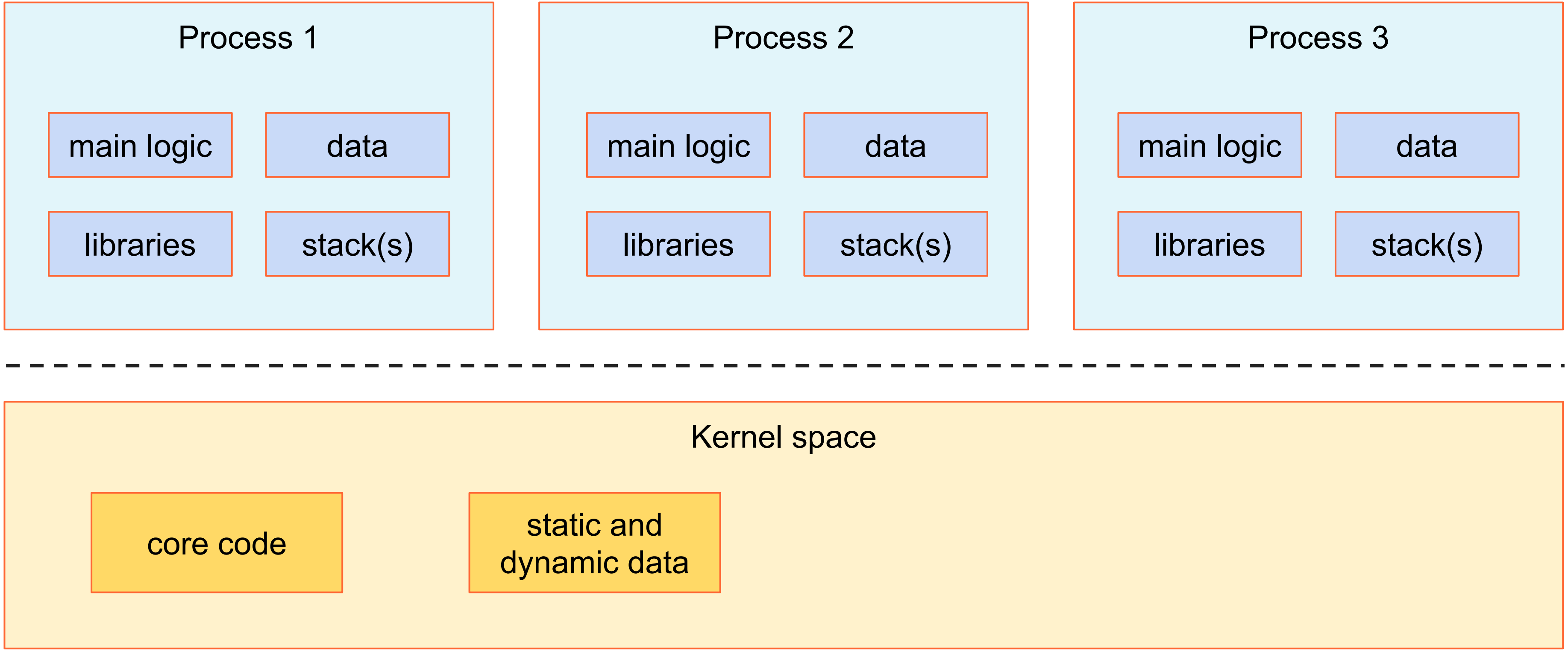
Linux address spaces



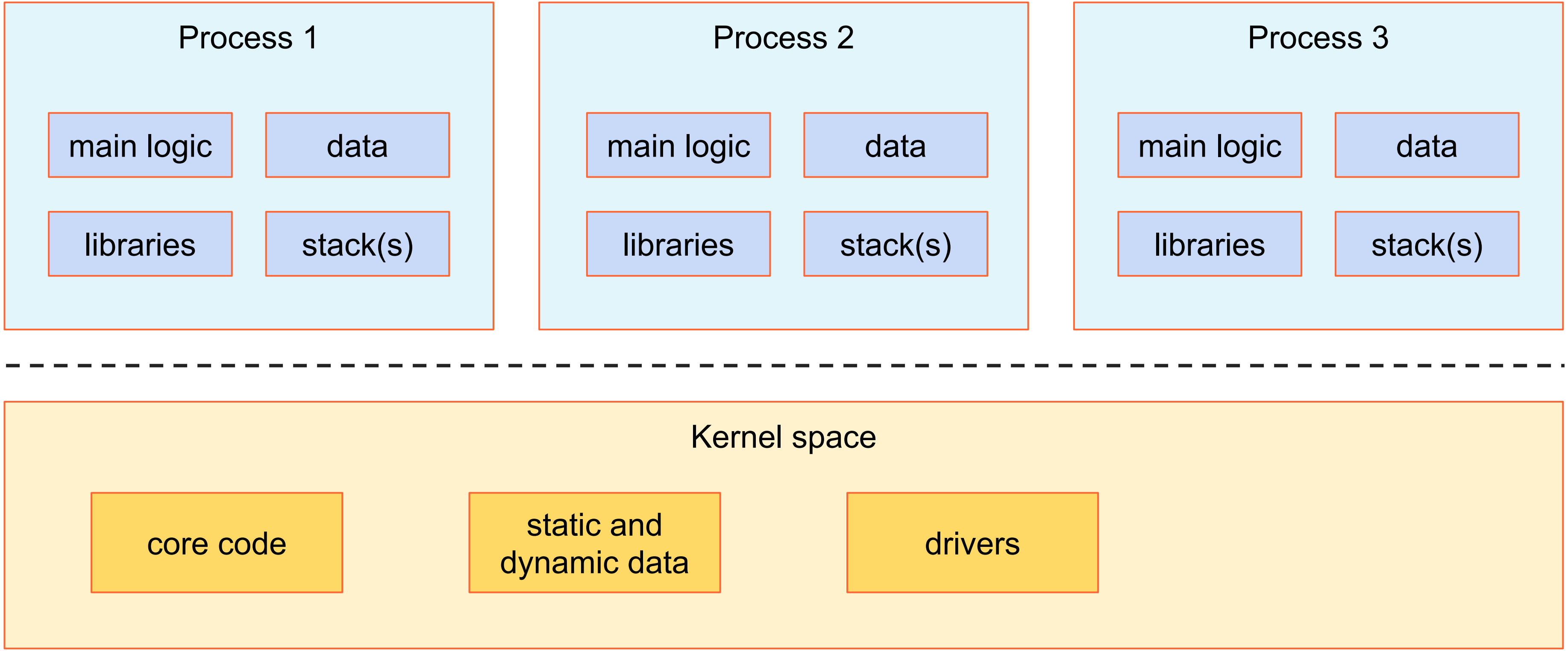
Linux address spaces



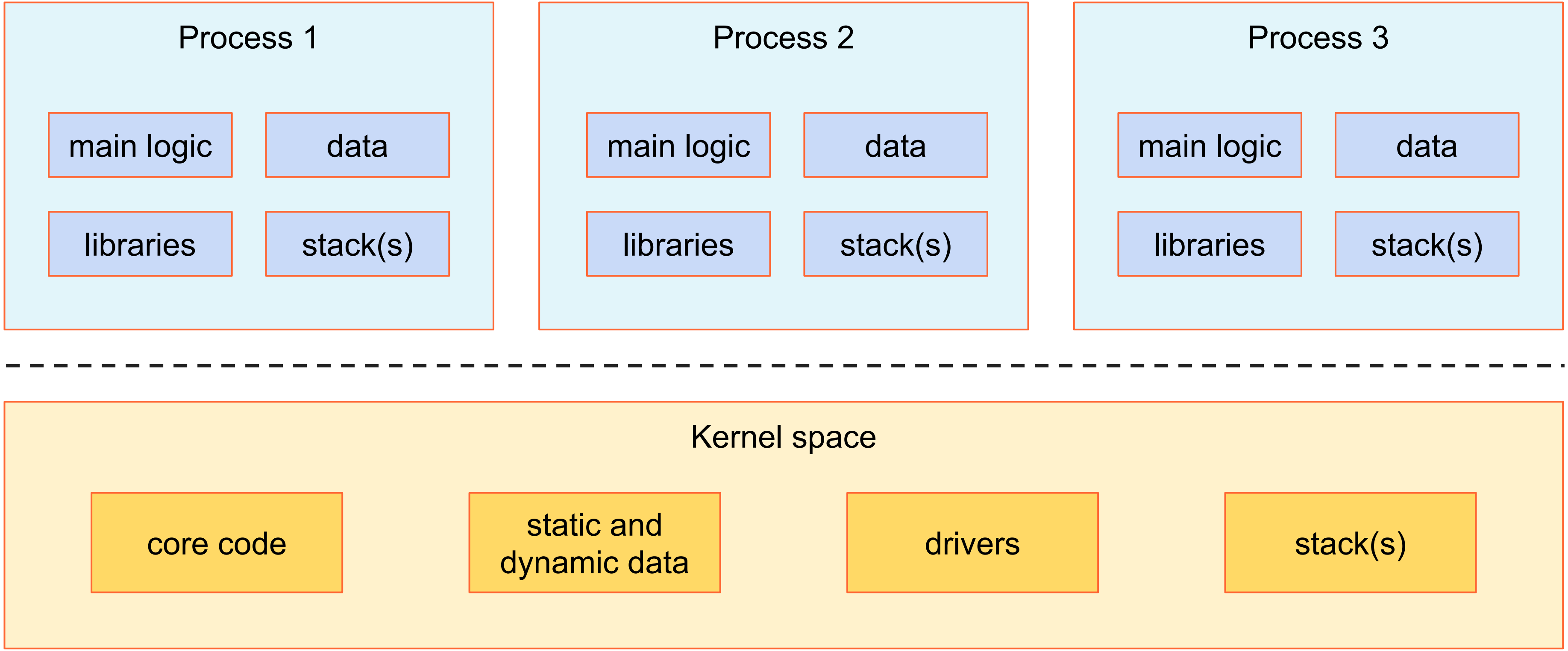
Linux address spaces



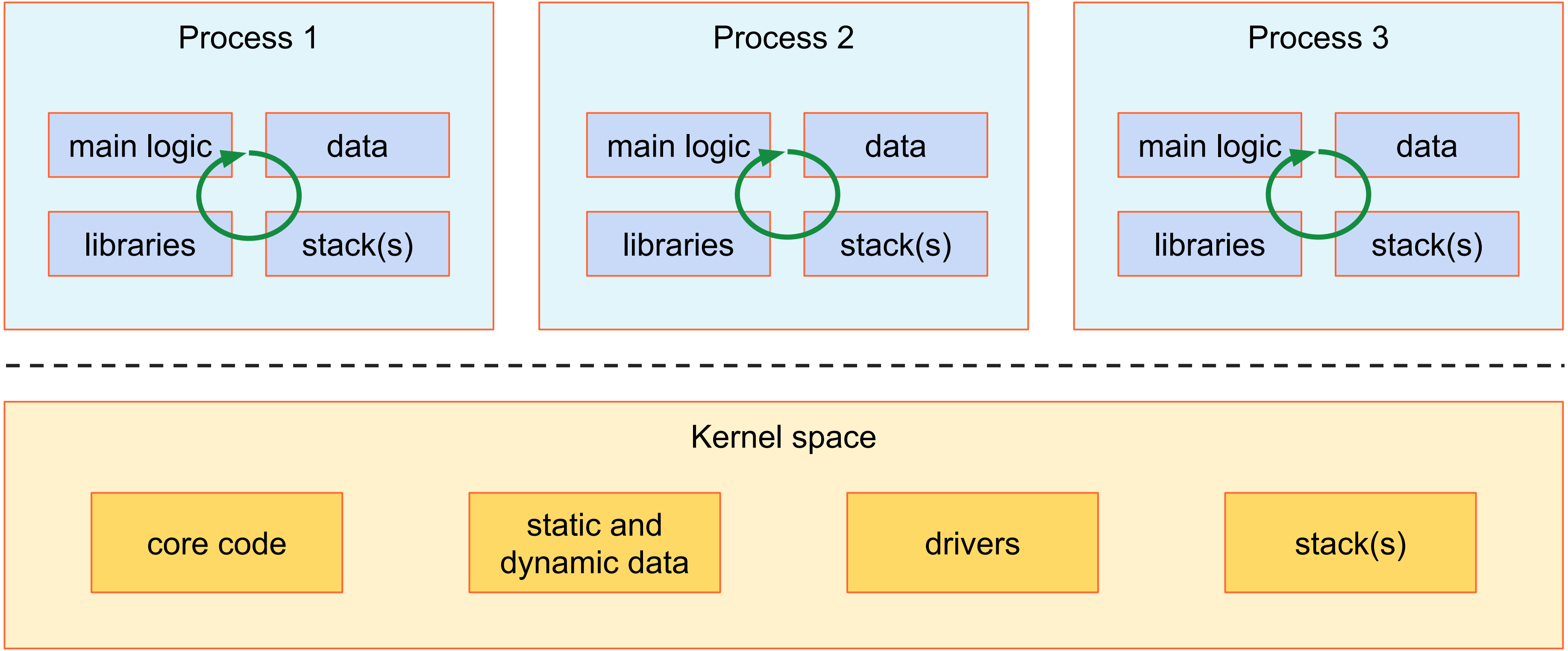
Linux address spaces



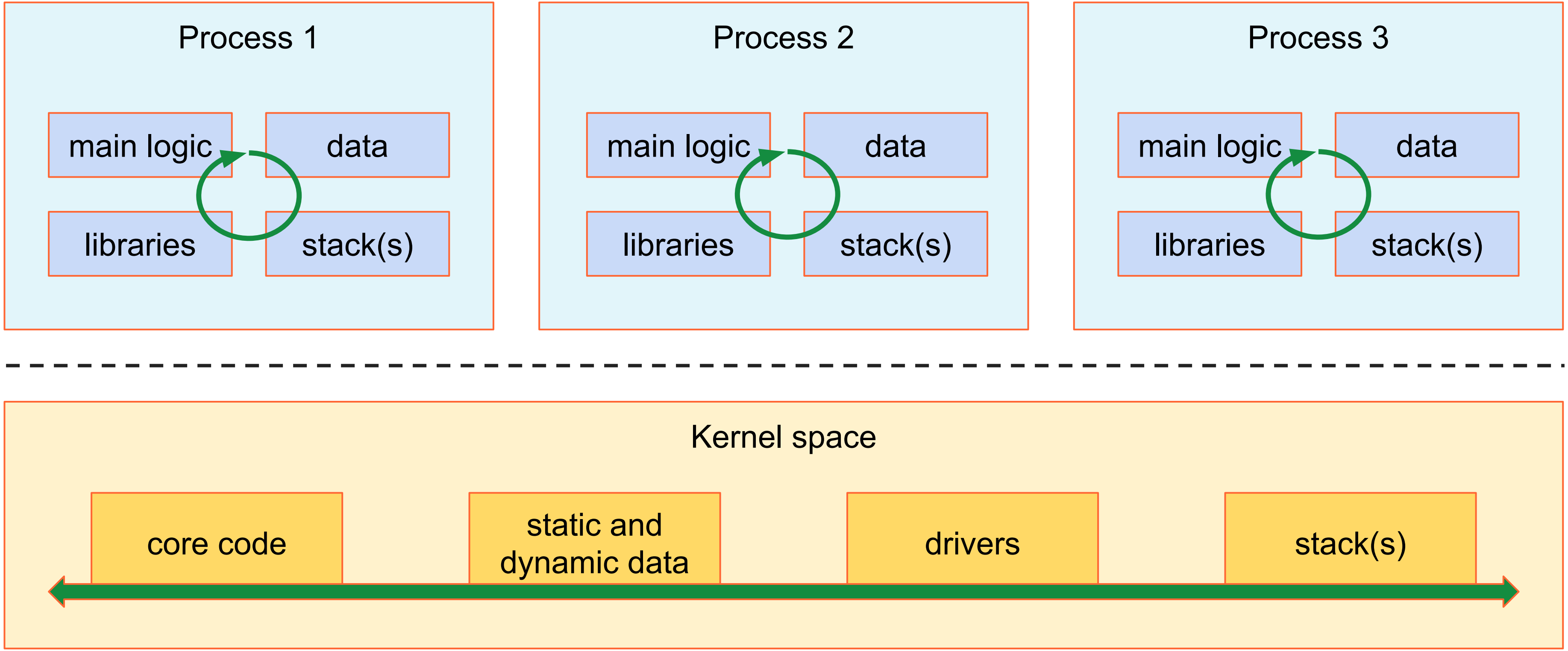
Linux address spaces



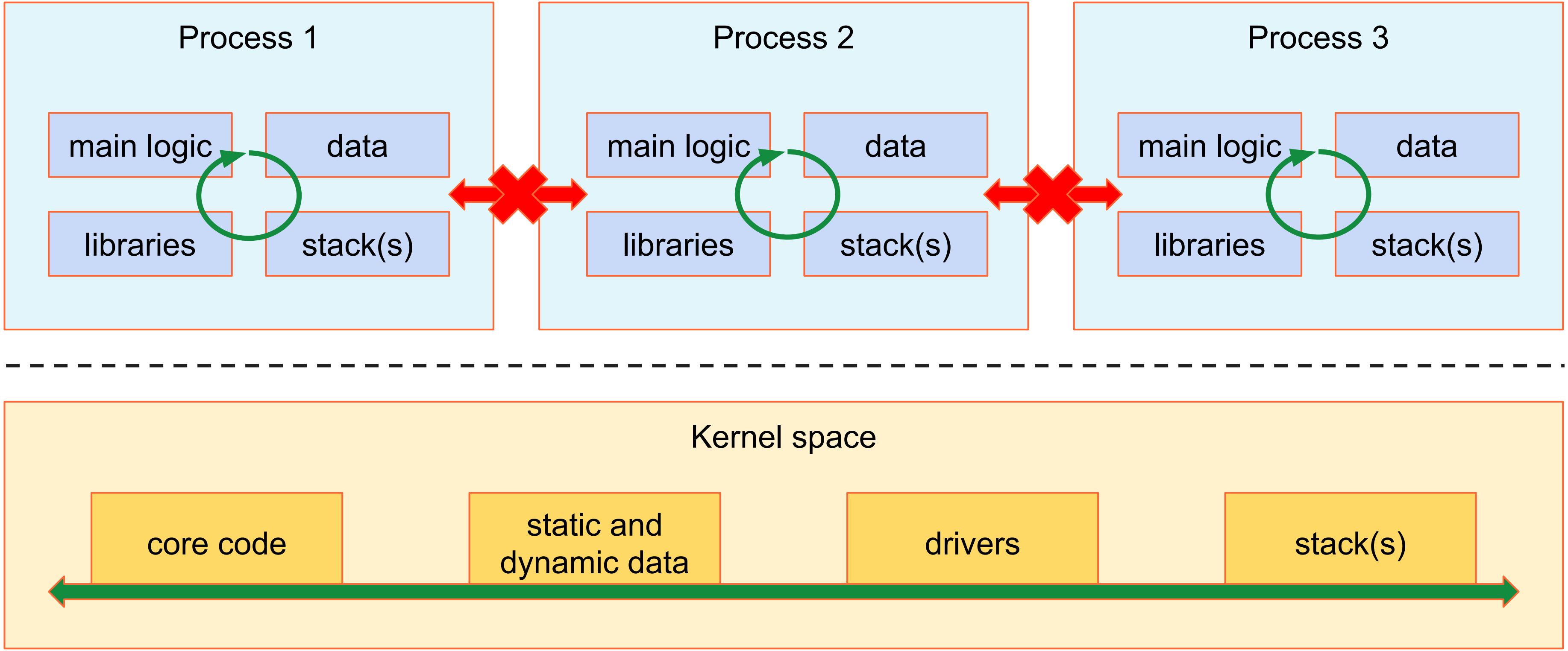
Linux address spaces



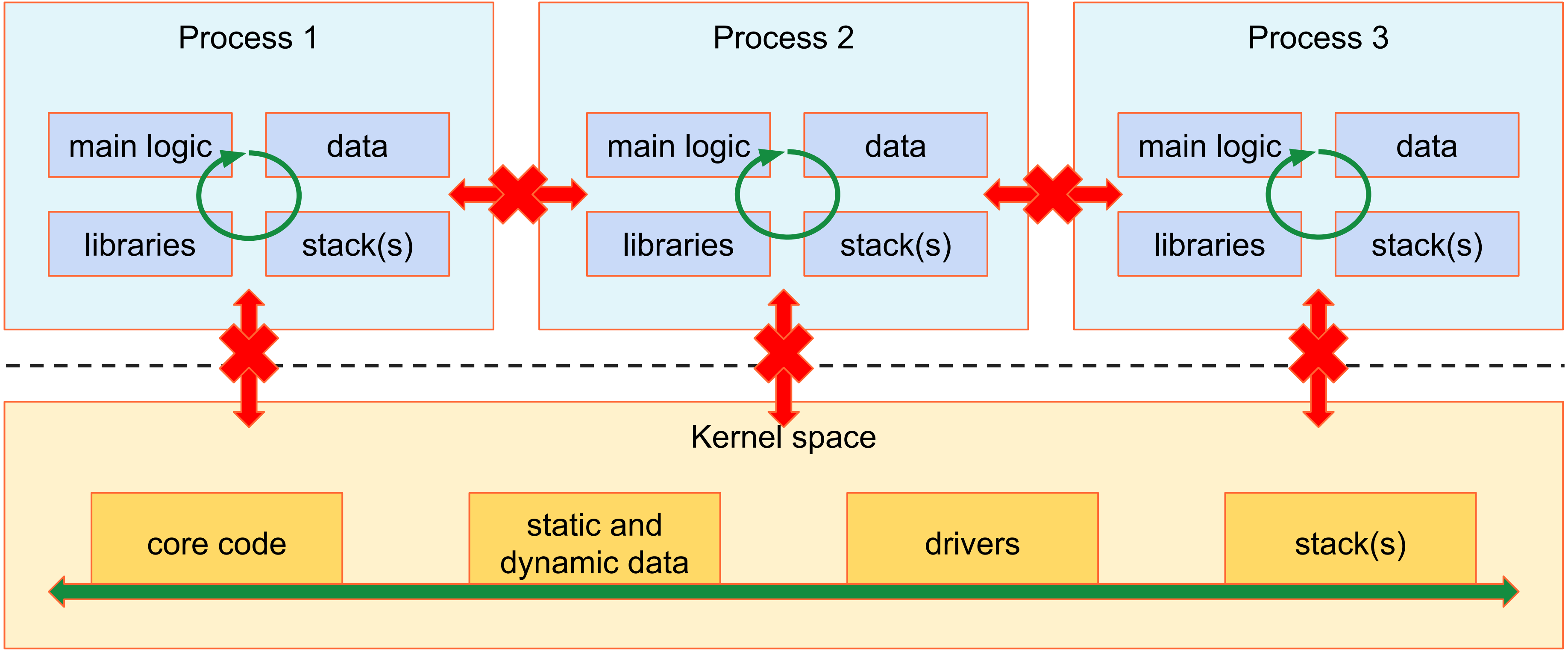
Linux address spaces



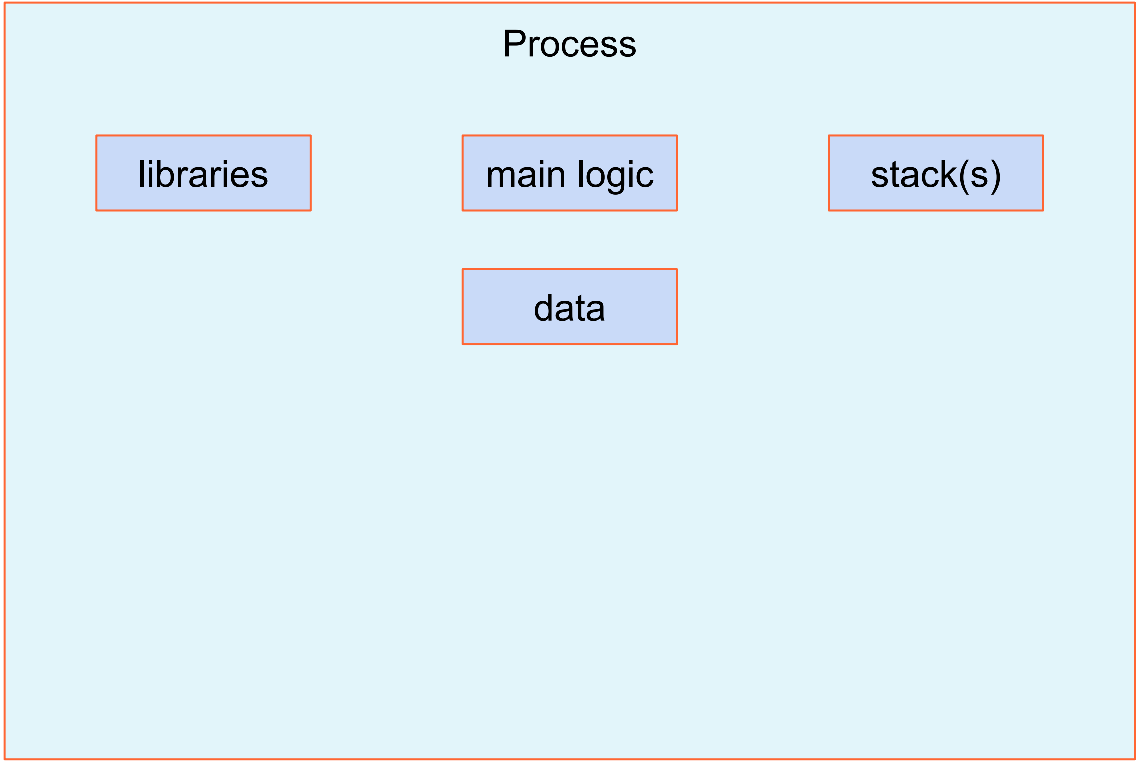
Linux address spaces



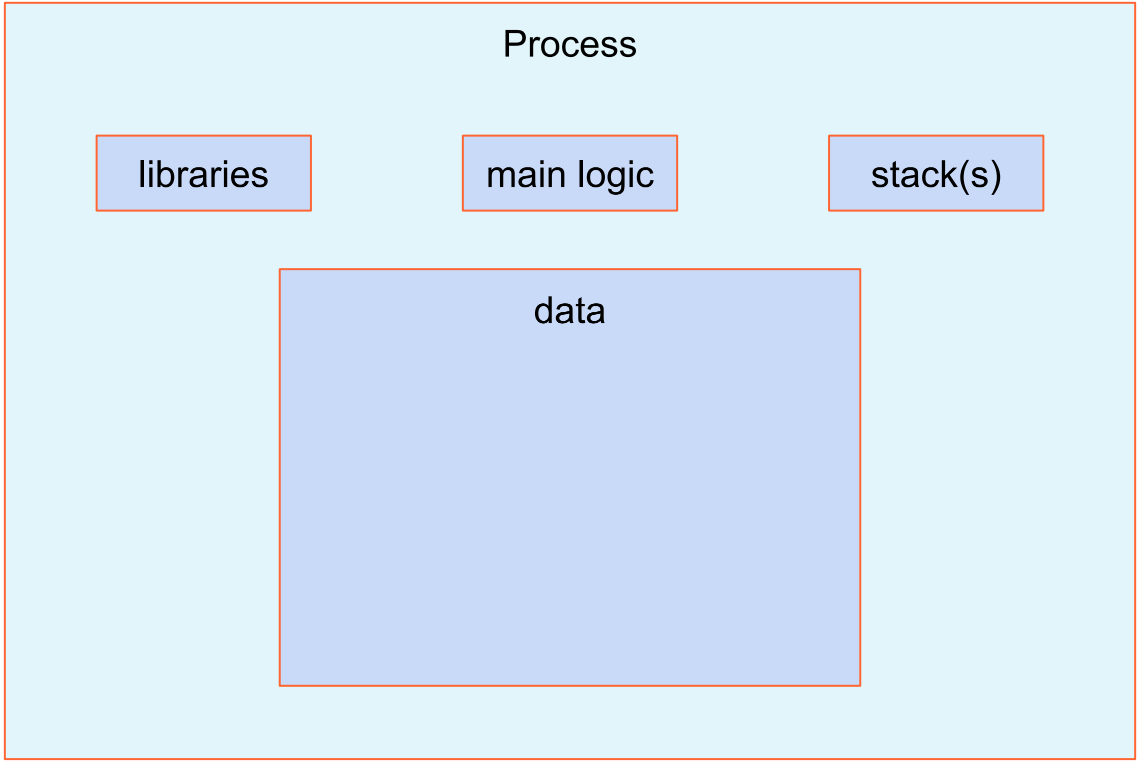
Linux address spaces



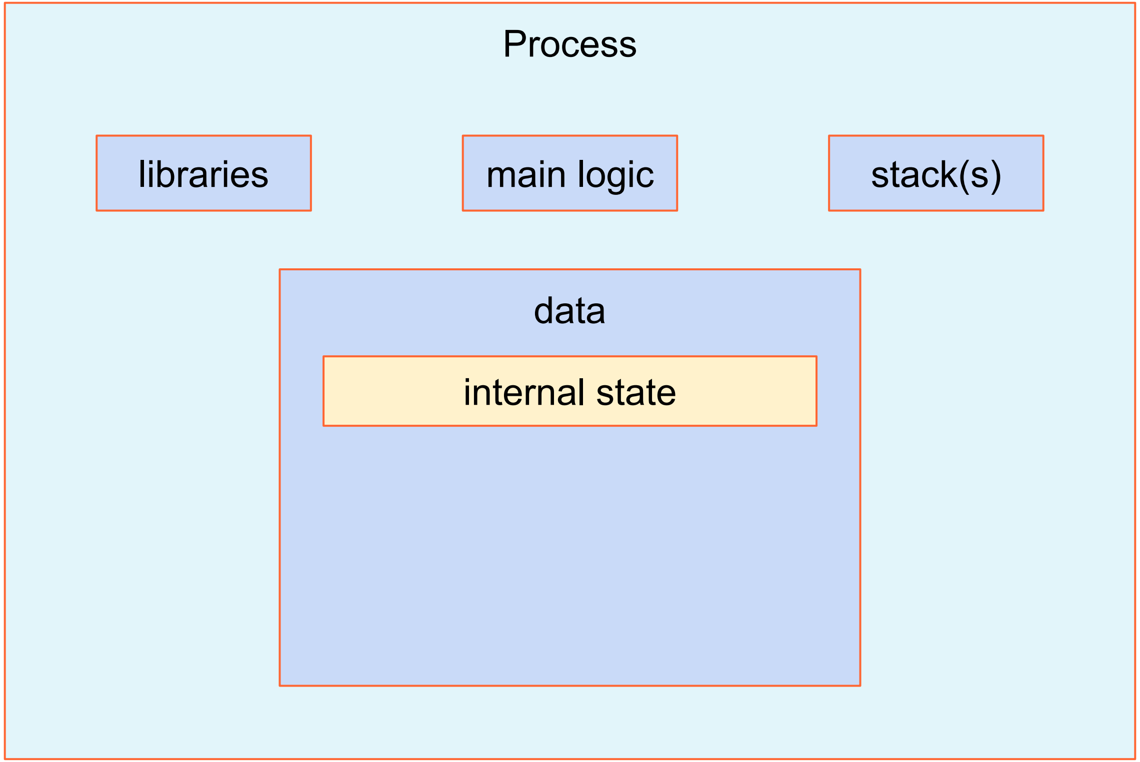
Linux address spaces



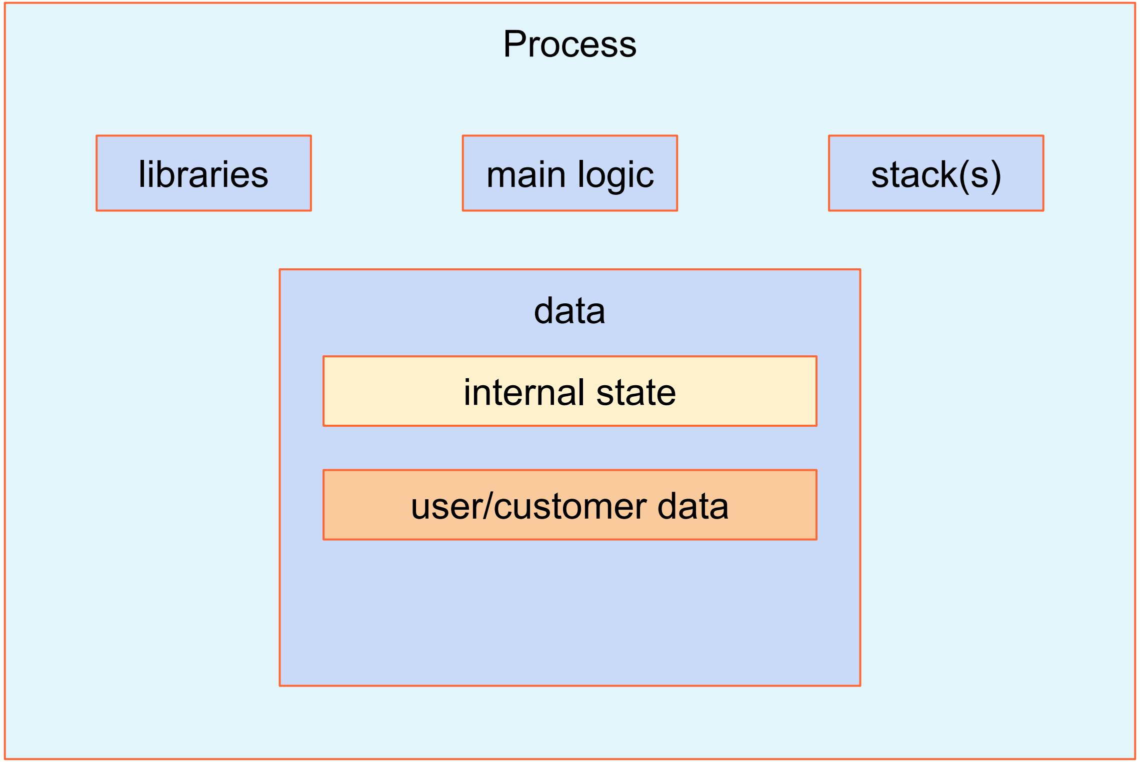
Linux address spaces



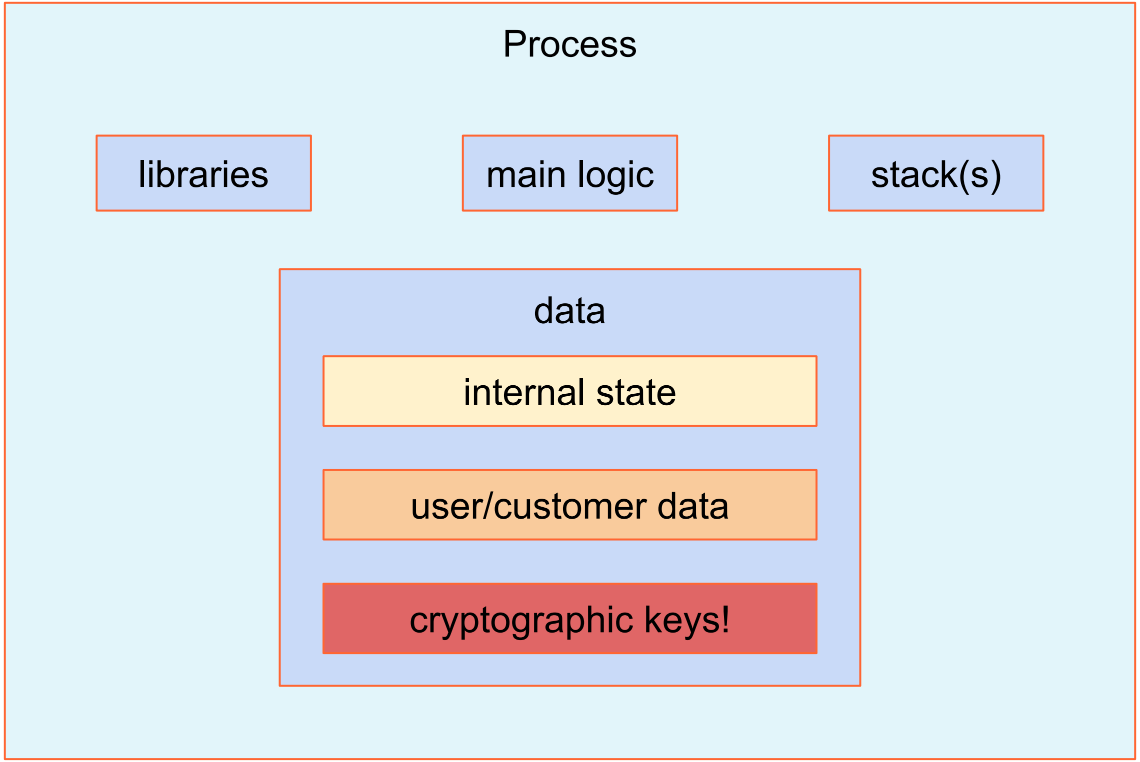
Linux address spaces



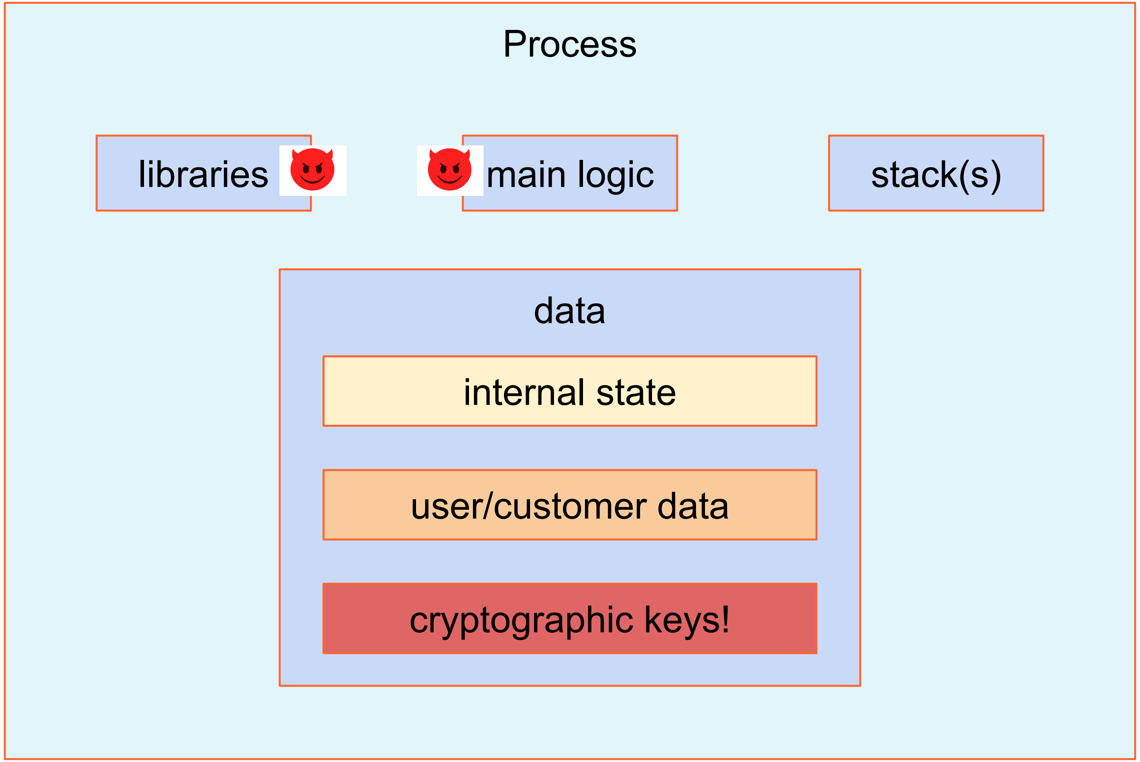
Linux address spaces



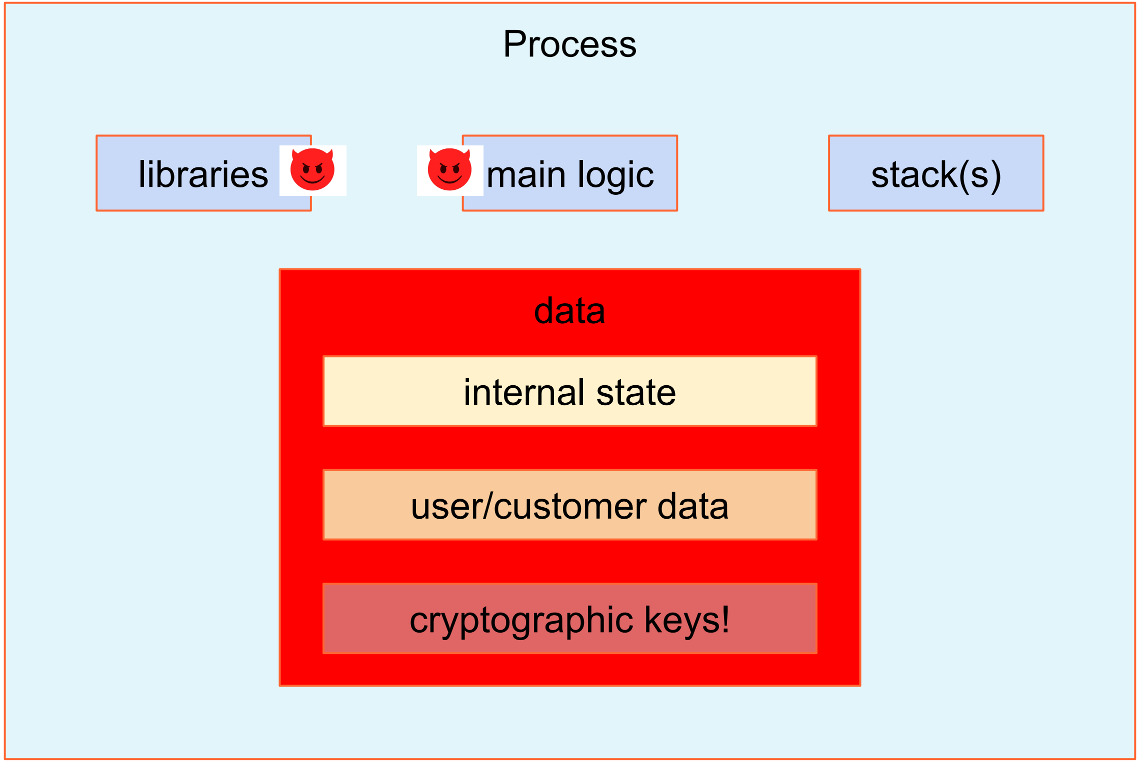
Linux address spaces



Linux address spaces

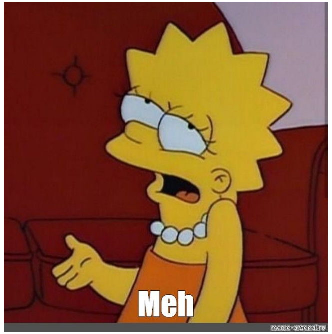


Linux address spaces



Not all process data is created equal

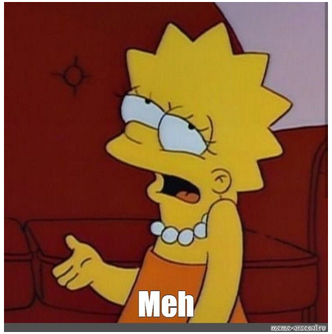
- Application internal state is compromised
  - Can be good or bad
  - Can lead to further compromise



Not all process data is created equal

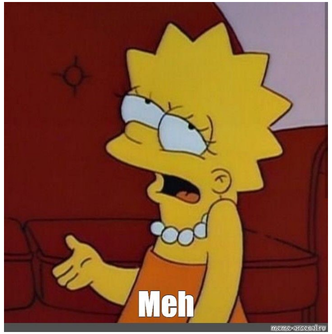
- Application internal state is compromised
  - Can be good or bad
  - Can lead to further compromise

- User/customer data is compromised
  - Privacy leaks

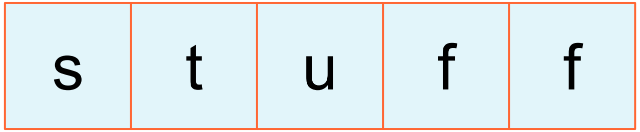


Not all process data is created equal

- Application internal state is compromised
  - Can be good or bad
  - Can lead to further compromise
- User/customer data is compromised
  - Privacy leaks
- Cryptographic key compromise
  - Data integrity compromise
  - Full security compromise
  - Total identity takeover



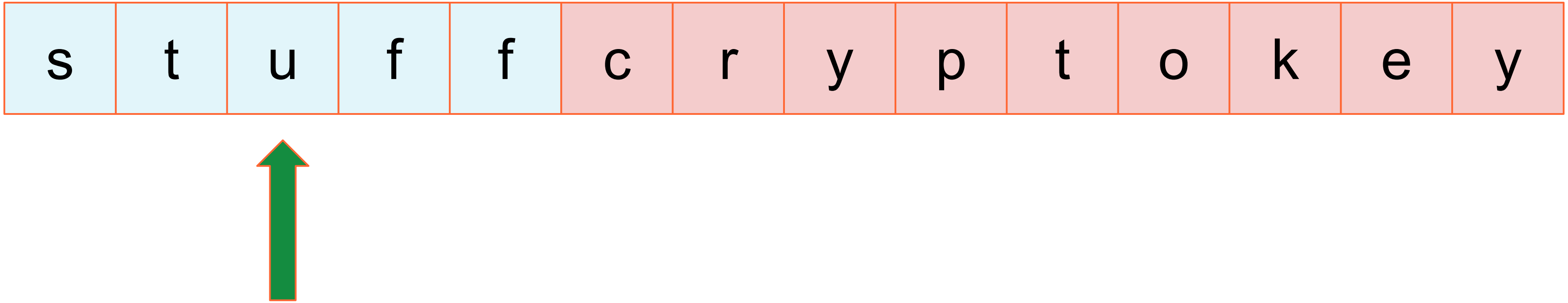
Untrusted inputs and out-of-bounds memory access



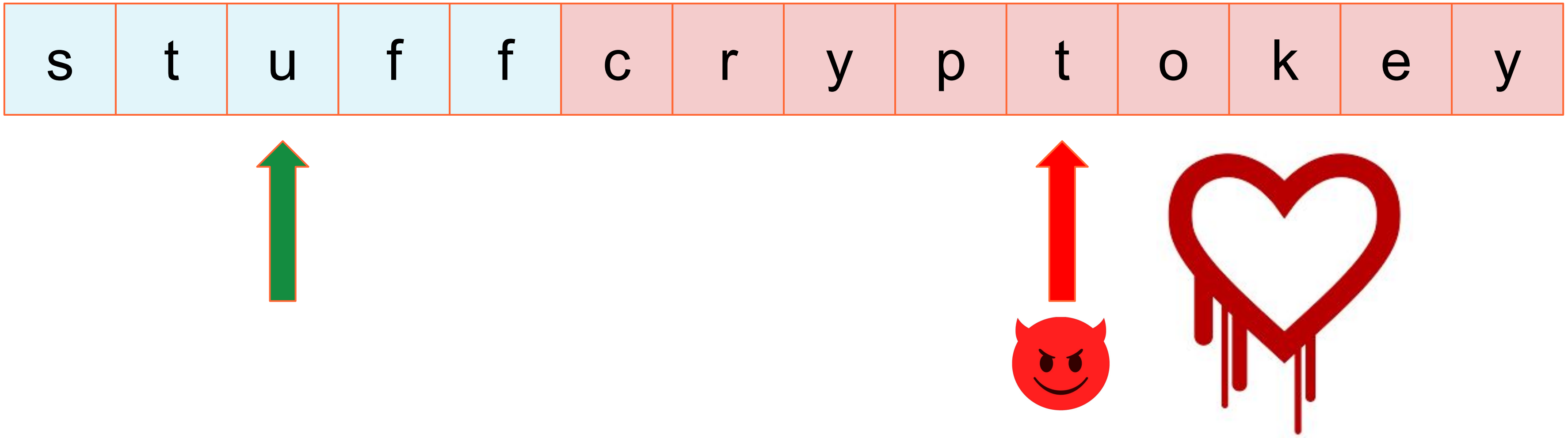
Untrusted inputs and out-of-bounds memory access

s	t	u	f	f	c	r	y	p	t	o	k	e	y

Untrusted inputs and out-of-bounds memory access



Untrusted inputs and out-of-bounds memory access



Arbitrary/remote code execution

Arbitrary/remote code execution

https://en.wikipedia.org/wiki/Log4Shell

Buffer reuse

```
#include <stdio.h>
#include <stdint.h>

static void encrypt(void)
{
    uint8_t key[] = "hunter2";
    printf("encrypting with super secret key: %s\n", key);
}

static void log_completion(void)
{
    /* oh no, we forgot to init the msg */
    char msg[8];
    printf("not important, just fyi: %s\n", msg);
}
```

```
int main(void)
{
    encrypt();
    /* notify that we're done */
    log_completion();
    return 0;
}
```

Buffer reuse

```c
#include <stdio.h>
#include <stdint.h>

static void encrypt(void)
{
    uint8_t key[] = "hunter2";
    printf("encrypting with super secret key: %s\n", key);
}

static void log_completion(void)
{
    /* oh no, we forgot to init the msg */
    char msg[8];
    printf("not important, just fyi: %s\n", msg);
}
```

```c
int main(void)
{
    encrypt();
    /* notify that we're done */
    log_completion();
    return 0;
}
```

Buffer reuse

```
#include <stdio.h>
#include <stdint.h>

static void encrypt(void)
{
    uint8_t key[] = "hunter2";
    printf("encrypting with super secret key: %s\n", key);
}

static void log_completion(void)
{
    /* oh no, we forgot to init the msg */
    char msg[8];
    printf("not important, just fyi: %s\n", msg);
}
```

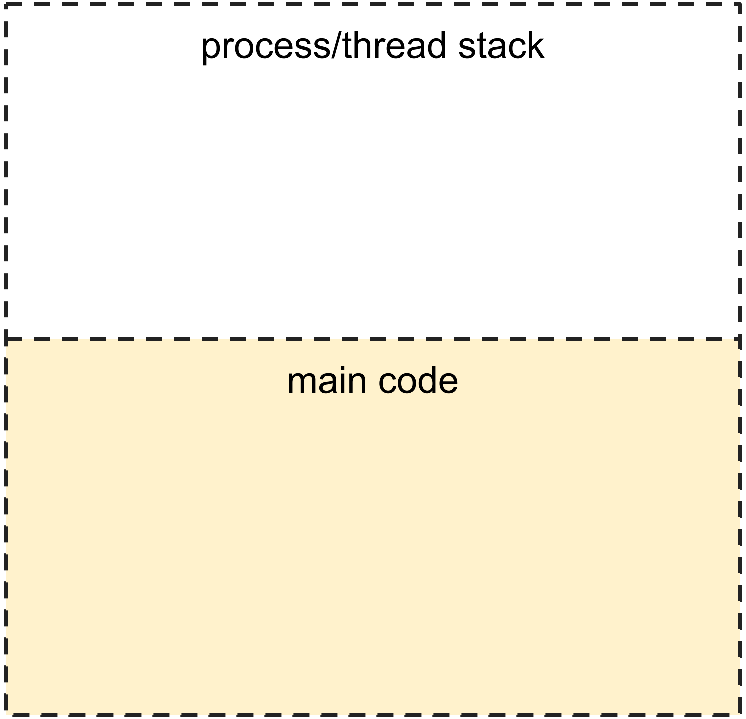
```
int main(void)
{
    encrypt();
    /* notify that we're done */
    log_completion();
    return 0;
}
```

```
$ gcc -o broken broken.c
$ ./broken
encrypting with super secret key: hunter2
not important, just fyi: hunter2
```

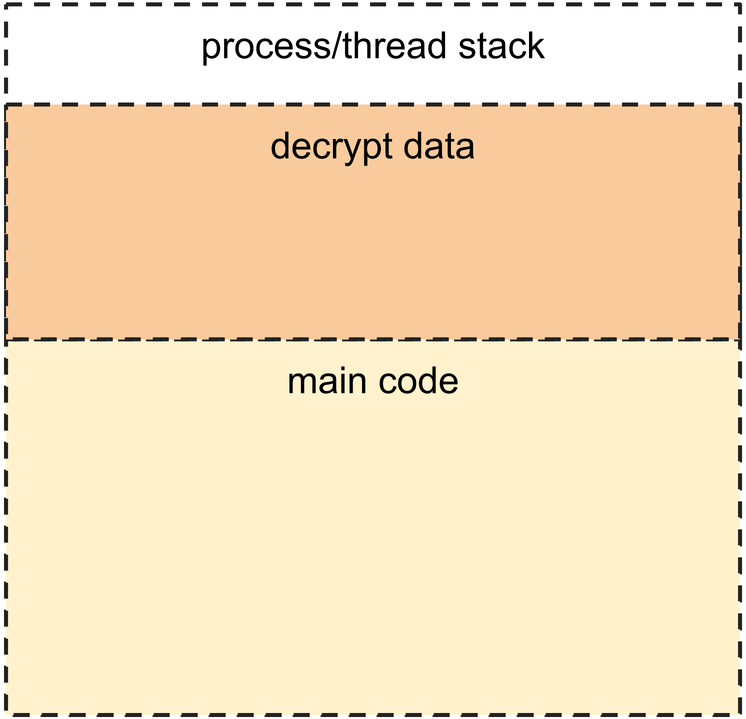
Buffer reuse

process/thread stack

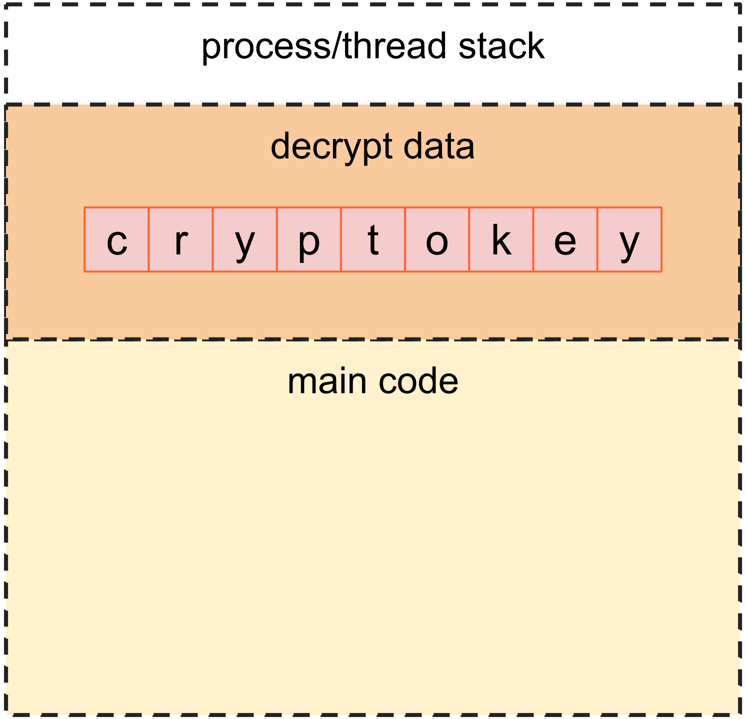
Buffer reuse



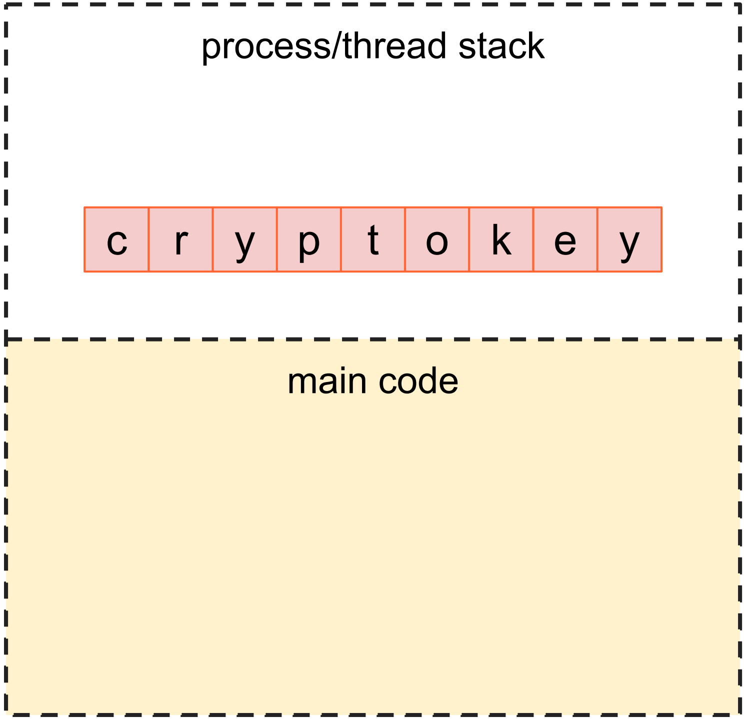
Buffer reuse



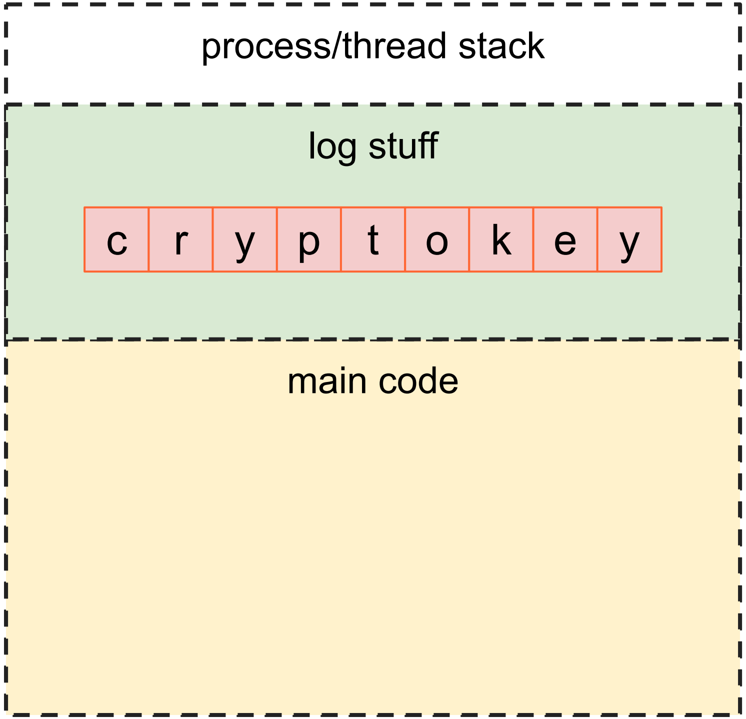
Buffer reuse



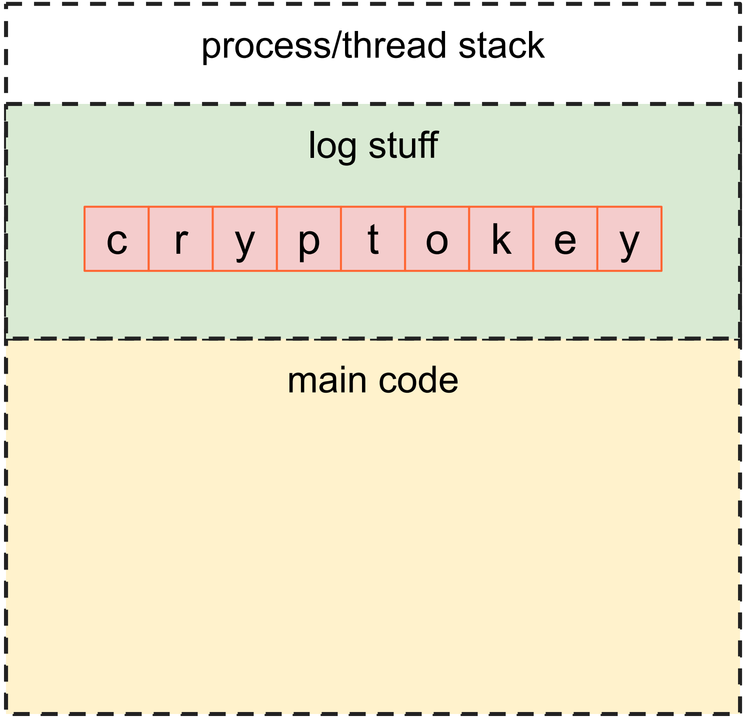
Buffer reuse



Buffer reuse



Buffer reuse



- Need to zero memory after key use
  - Both stack and heap
  - Challenging in garbage collected languages

Debugging info and tools

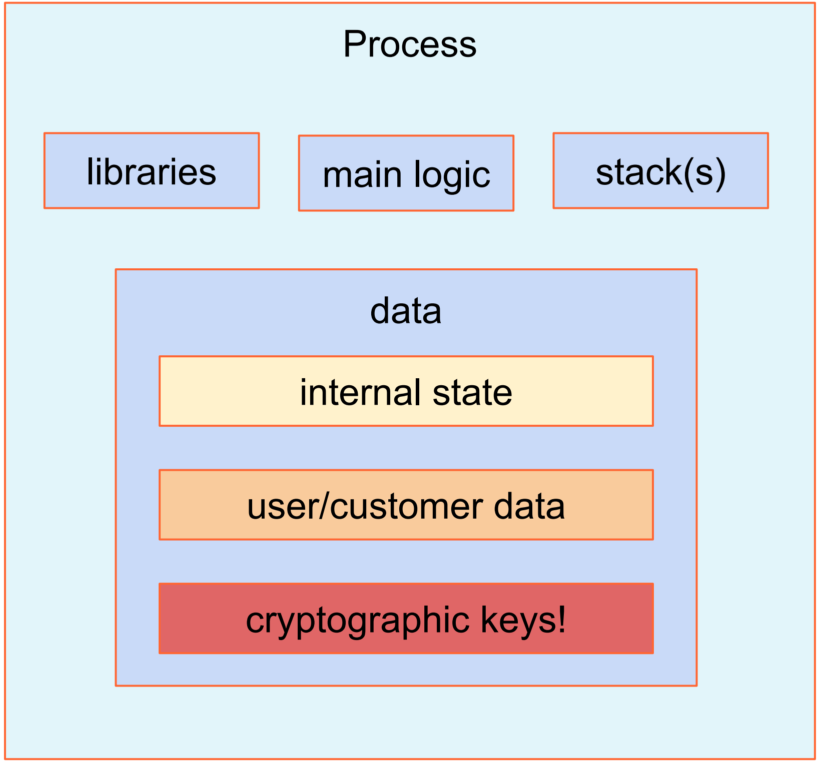
```
Segmentation fault
```

Debugging info and tools

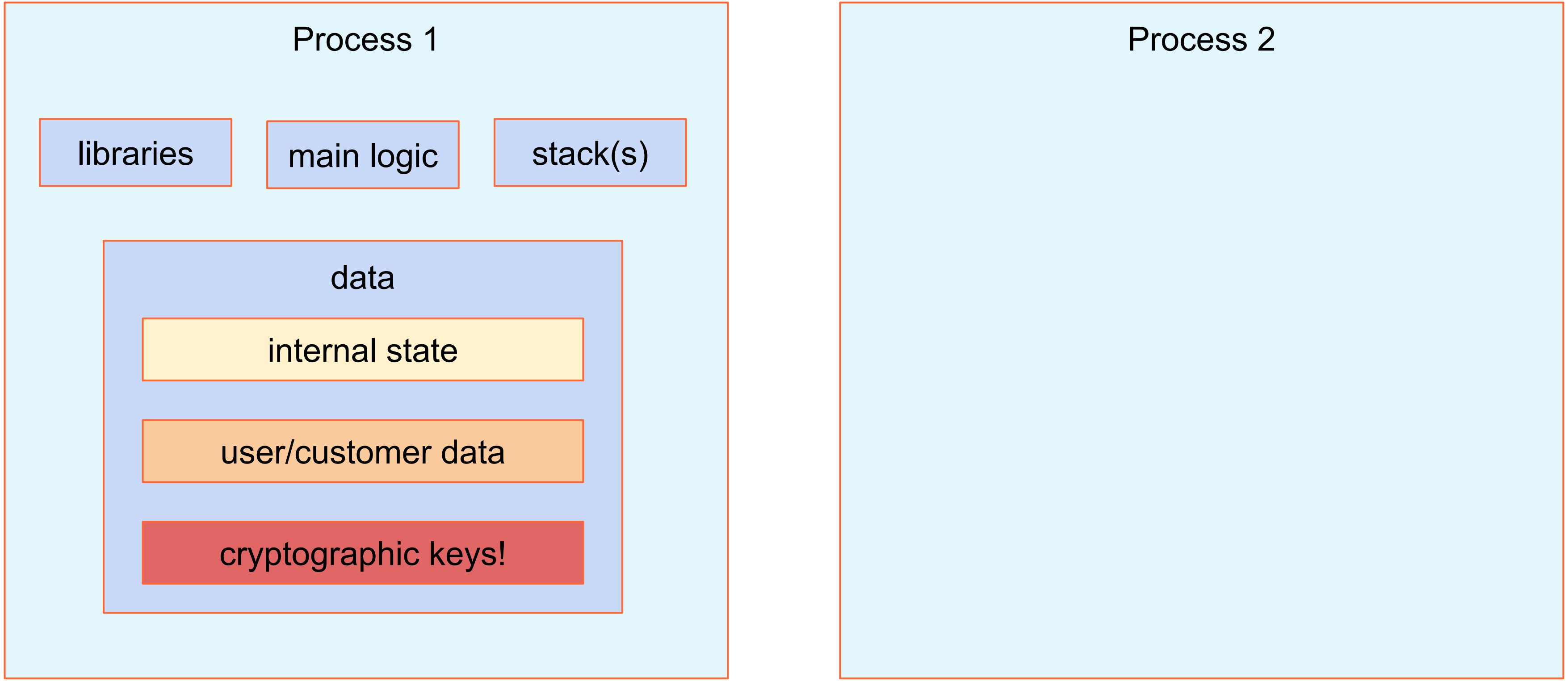
- logging
- coredumps
- gdb
- ptrace

Fix all the bugs?

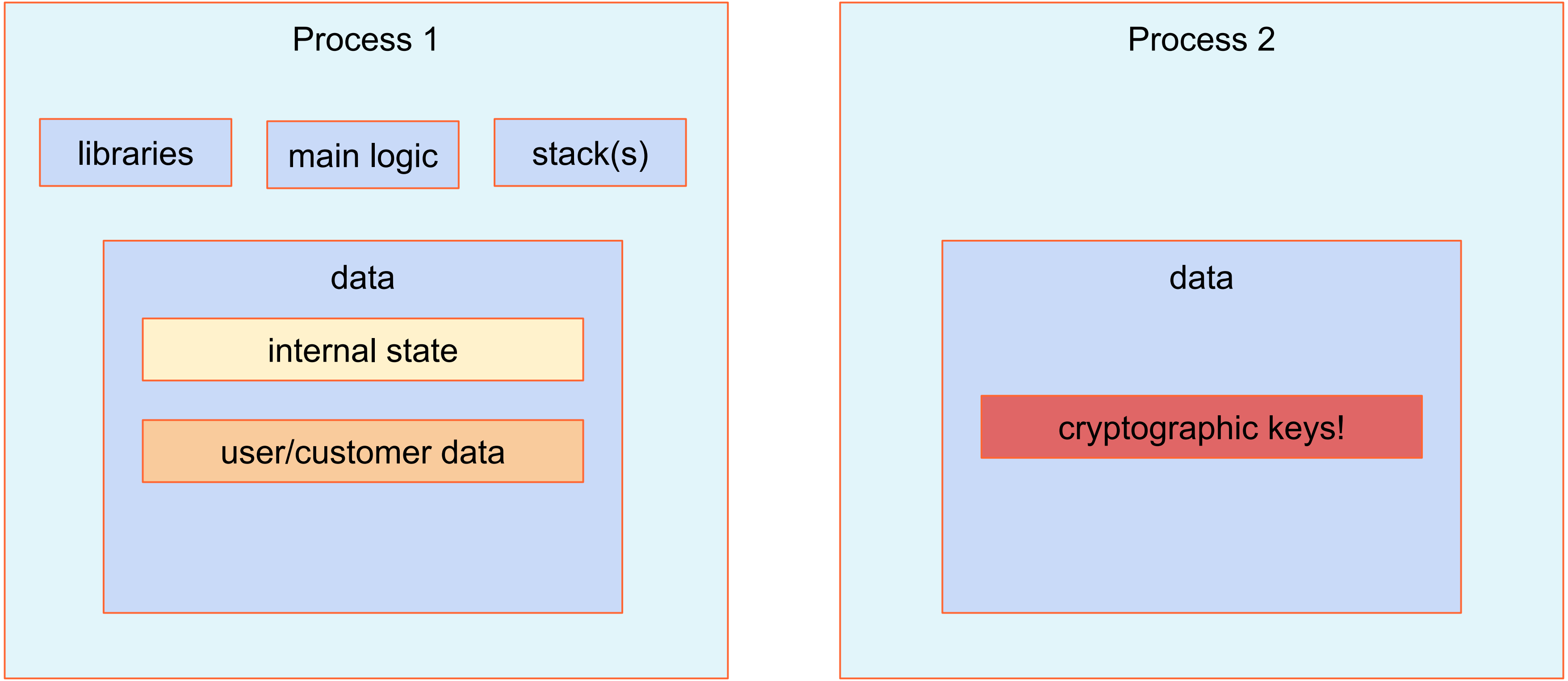
Linux address spaces



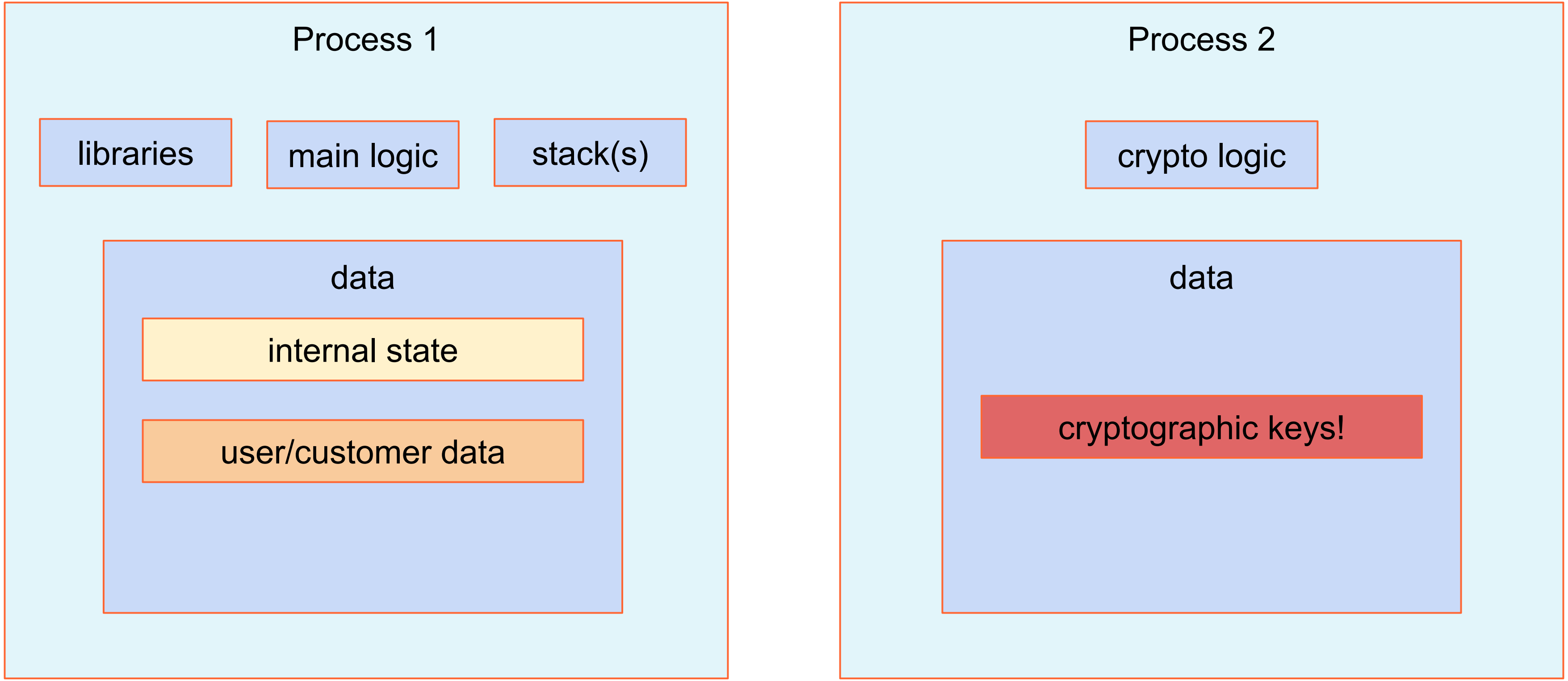
Linux address spaces



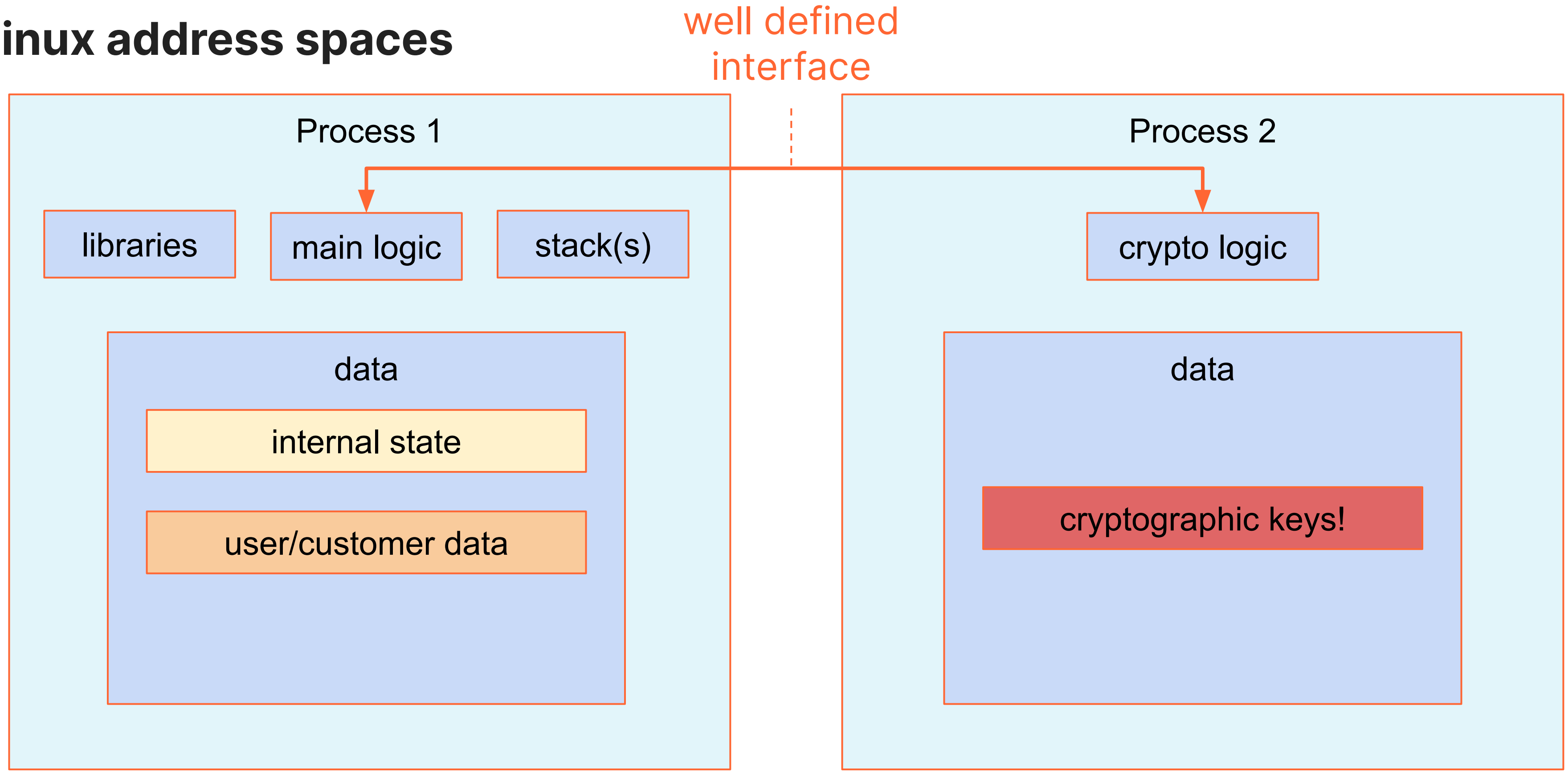
Linux address spaces



Linux address spaces



Linux address spaces



Key agent model

- Two processes: main and a helper “agent”:
  - main process does not have access to the cryptographic material (ensured by the OS address space isolation)
  - main communicates with the “agent” through a well-defined interface to perform cryptographic operations
  - main processes untrusted input and is usually network-facing
  - “agent” does not process untrusted input and is usually not network facing

Key agent model

- Two processes: main and a helper “agent”:
  - main process does not have access to the cryptographic material (ensured by the OS address space isolation)
  - main communicates with the “agent” through a well-defined interface to perform cryptographic operations
  - main processes untrusted input and is usually network-facing
  - “agent” does not process untrusted input and is usually not network facing
- Think of the “agent” as a software security key
  - ssh-agent
  - gpg-agent

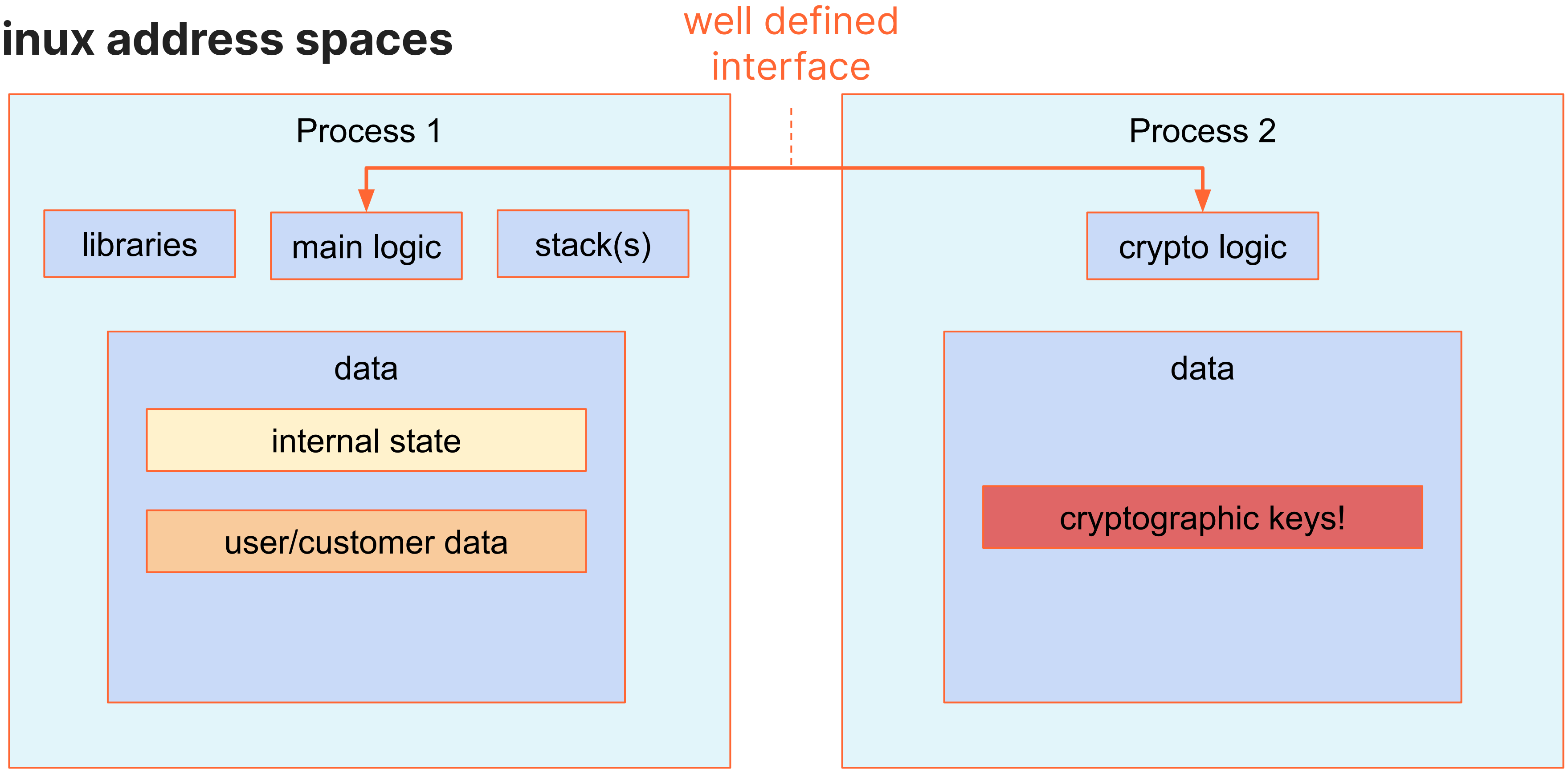
Key agent model

- Drawbacks
  - need to develop and maintain two programs/processes
  - need to design the “well-defined interface” between main and the agent
  - need to add communication support between the two processes (Unix sockets, shared memory etc)
  - need to somehow authenticate and enforce some ACLs of the main process in the agent

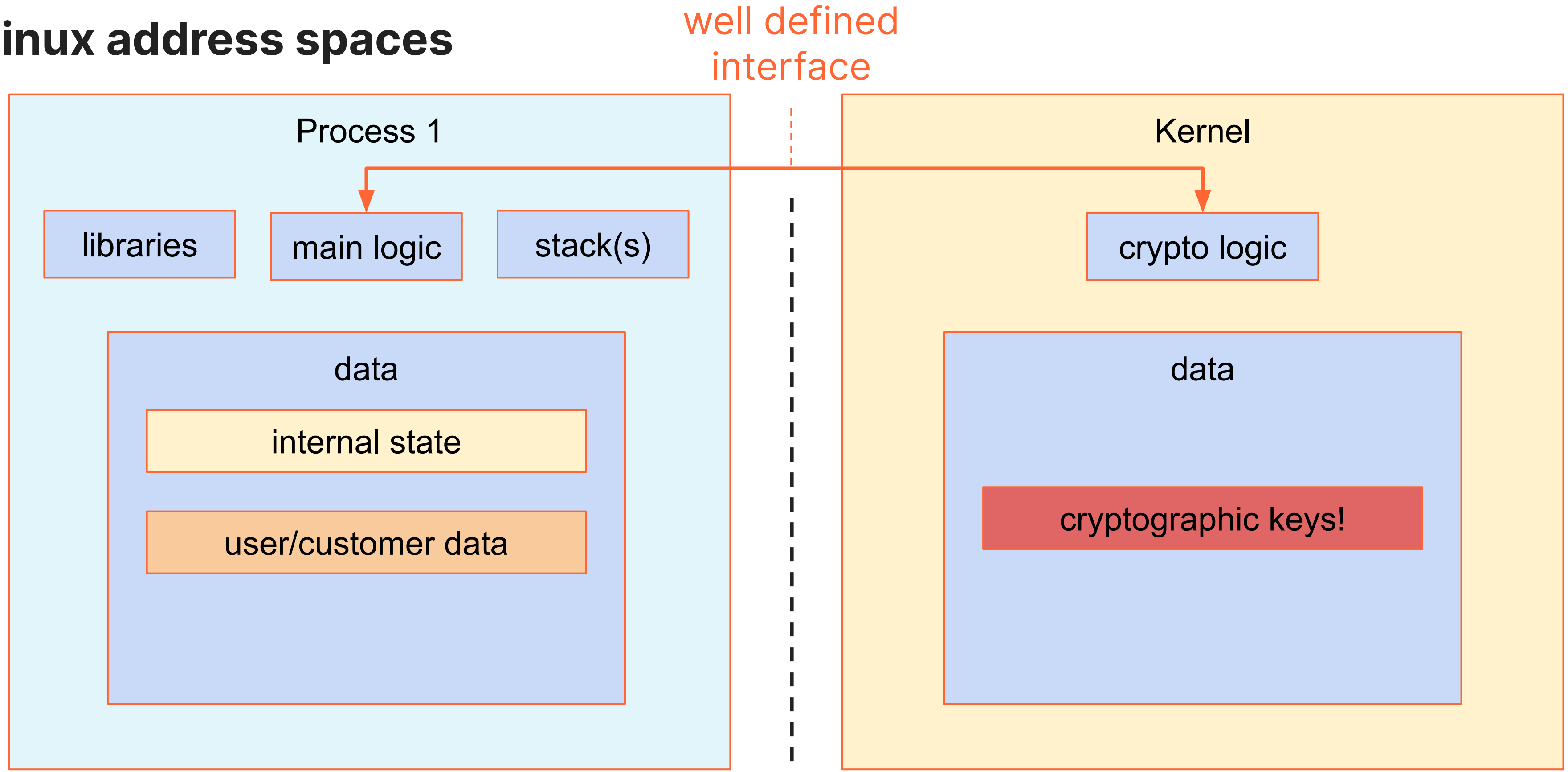
Linux Kernel key retention service

Or just Linux keystore

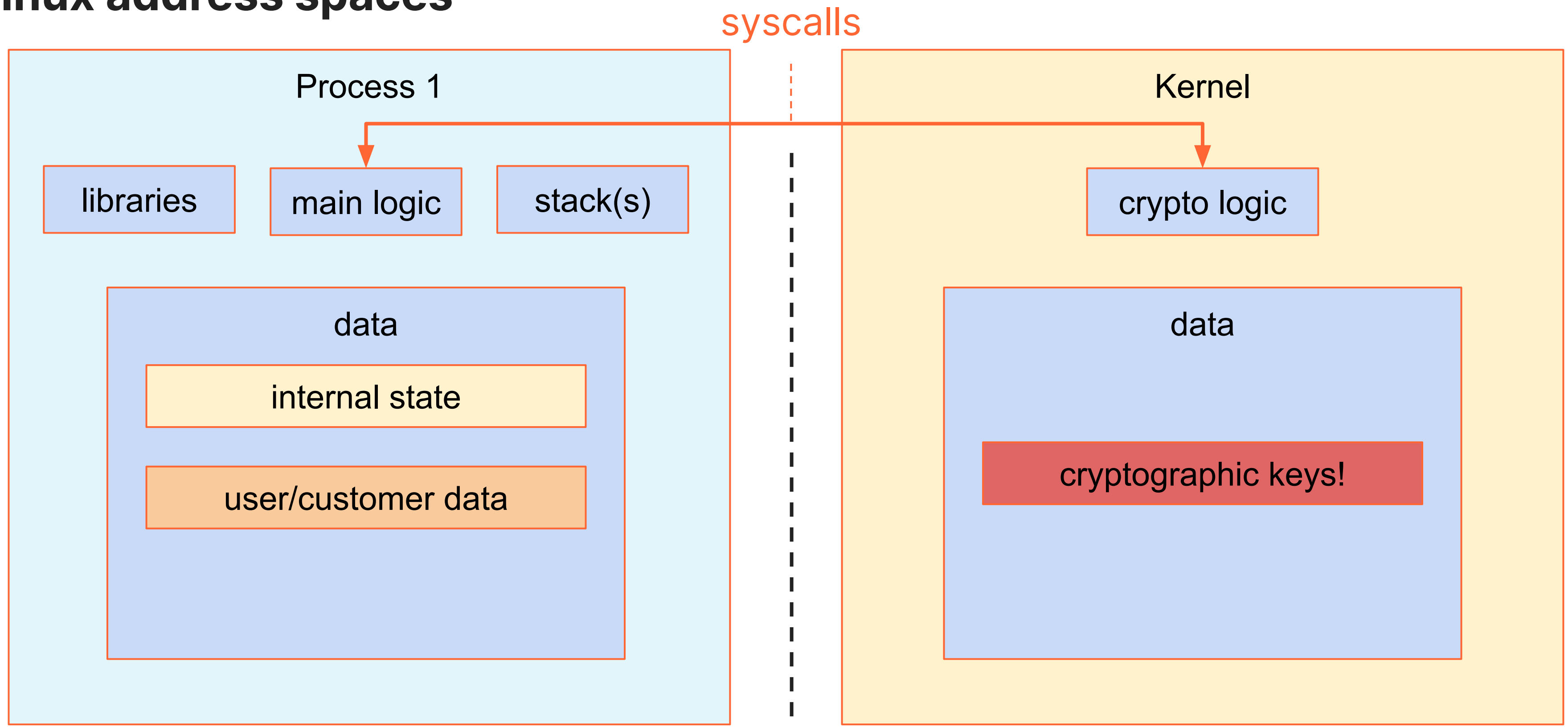
Linux address spaces



Linux address spaces



Linux address spaces



Linux Kernel key retention service

- Stores cryptograhic keys as kernel objects

Linux Kernel key retention service

- Stores cryptograhic keys as kernel objects
- Initially designed for sharing keys with kernel services
  - LUKS/dm-crypt
  - ecryptfs

CLOUDFLARE

Linux Kernel key retention service

- Stores cryptograhic keys as kernel objects
- Initially designed for sharing keys with kernel services
  - LUKS/dm-crypt
  - ecryptfs
- Can be used by userspace programs to manage their keys/secrets
  - keys are stored outside of the process address space
  - a well-defined system call interface to access and use the keys
  - kernel key objects have associated permissions and ACLs
    - including LSM hooks
  - key lifecycle can be implicitly bound to the code lifecycle
    - ex. key autodestruction, when a process terminates

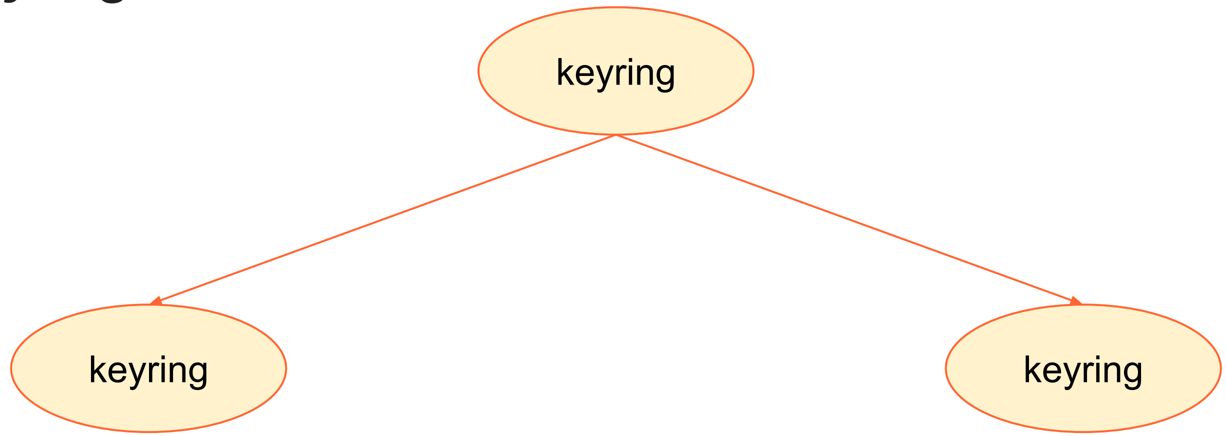
Linux Kernel key retention service

- Stores cryptograhic keys as kernel objects
- Initially designed for sharing keys with kernel services
  - LUKS/dm-crypt
  - ecryptfs
- Can be used by userspace programs to manage their keys/secrets
  - keys are stored outside of the process address space
  - a well-defined system call interface to access and use the keys
  - kernel key objects have associated permissions and ACLs
    - including LSM hooks
  - key lifecycle can be implicitly bound to the code lifecycle
    - ex. key autodestruction, when a process terminates

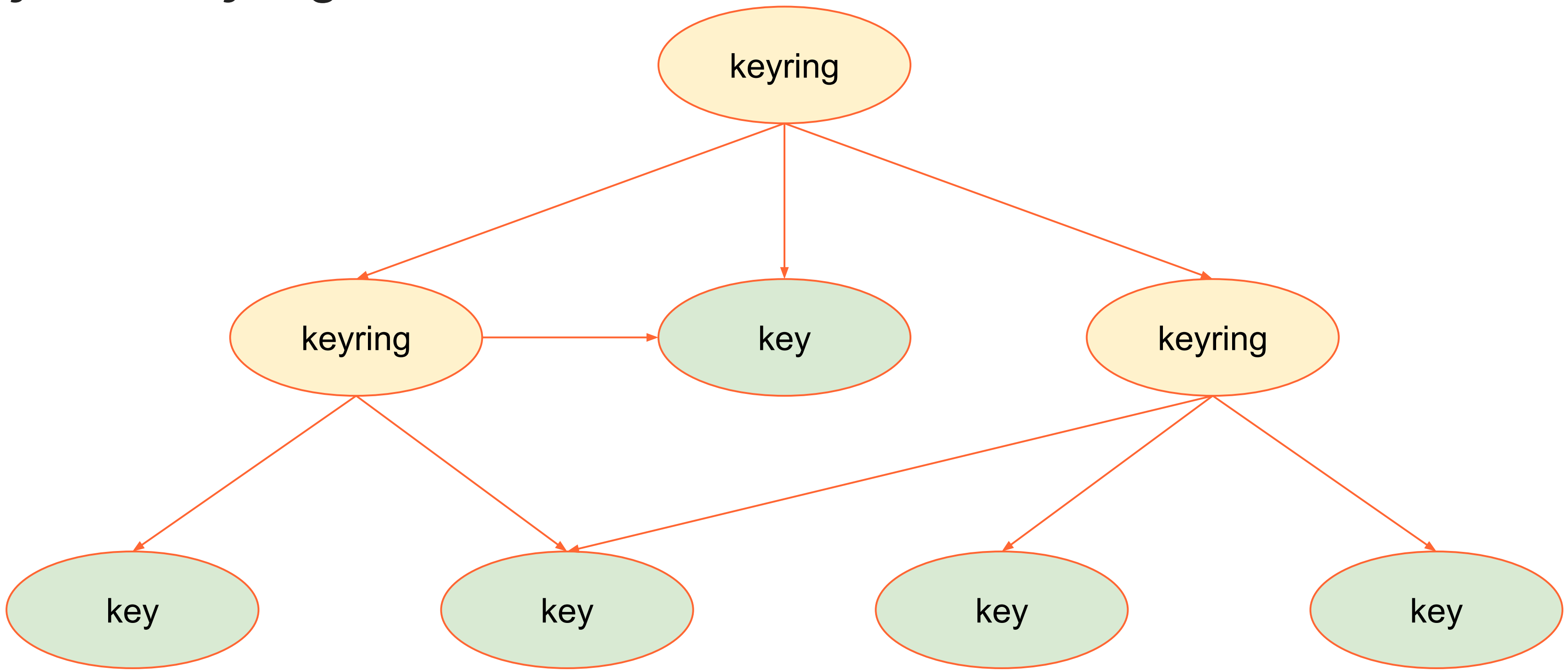
https://www.kernel.org/doc/html/latest/security/keys/core.html

Keys and keyrings

Keys and keyrings



Keys and keyrings



Keys and keyrings

Keys

- contain actual cryptographic material or a pointer to it
- can be read/written to and used to perform cryptographic transformations
- can be of different types:
  - user
  - logon
  - asymmetric
  - encrypted
  - trusted
- similar to a file on a filesystem
  - but can be linked to many keyrings in the same time

Keys and keyrings

Keys

- contain actual cryptographic material or a pointer to it
- can be read/written to and used to perform cryptographic transformations
- can be of different types:
  - user
  - logon
  - asymmetric
  - encrypted
  - trusted
- similar to a file on a filesystem
  - but can be linked to many keyrings in the same time

Keyrings

- contain links to keys and other keyrings
  - if a key is not linked to a single keyring, it is securely destroyed
- represent a collection of keys
- can be explicitly created or special:
  - thread
  - process
  - user
  - session
- may enforce key lifetime
- similar to a directory on a filesystem

Keys and keyrings

```
ignat@dev:~$ keyctl newring myring @u
850826109
```

CLOUDFLARE

Keys and keyrings

```
ignat@dev:~$ keyctl newring myring @u
850826109
ignat@dev:~$ keyctl add user mykey hunter2 %:myring
975891189
```

Keys and keyrings

```
ignat@dev:~$ keyctl newring myring @u
850826109
ignat@dev:~$ keyctl add user mykey hunter2 %:myring
975891189
ignat@dev:~$ keyctl show
Session Keyring
 346094565 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 850826109 --alswrv   1000  1000       \_ keyring: myring
 975891189 --alswrv   1000  1000           \_ user: mykey
```

Keys and keyrings

```
ignat@dev:~$ keyctl newring myring @u
850826109
ignat@dev:~$ keyctl add user mykey hunter2 %:myring
975891189
ignat@dev:~$ keyctl show
Session Keyring
 346094565 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 850826109 --alswrv   1000  1000       \_ keyring: myring
 975891189 --alswrv   1000  1000           \_ user: mykey
```

Keys and keyrings

```
ignat@dev:~$ keyctl newring myring @u
850826109
ignat@dev:~$ keyctl add user mykey hunter2 %:myring
975891189
ignat@dev:~$ keyctl show
Session Keyring
 346094565 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 850826109 --alswrv   1000  1000       \_ keyring: myring
 975891189 --alswrv   1000  1000           \_ user: mykey
ignat@dev:~$ keyctl print %user:mykey
hunter2
```

Example: secret sharing

```
alice@dev:~$ id
uid=1001(alice) gid=1001(alice)
groups=1001(alice)
```

```
bob@dev:~$ id
uid=1002(bob) gid=1002(bob)
groups=1002(bob)
```

Example: secret sharing

```
alice@dev:~$ id
uid=1001(alice) gid=1001(alice)
groups=1001(alice)
alice@dev:~$ keyctl add user secret
hunter2 @u
791615806
```

```
bob@dev:~$ id
uid=1002(bob) gid=1002(bob)
groups=1002(bob)
```

Example: secret sharing

```
alice@dev:~$ id
uid=1001(alice) gid=1001(alice)
groups=1001(alice)
alice@dev:~$ keyctl add user secret
hunter2 @u
791615806
```

```
bob@dev:~$ id
uid=1002(bob) gid=1002(bob)
groups=1002(bob)
bob@dev:~$ keyctl newring from-others @u
966722684
```

Example: secret sharing

```
alice@dev:~$ id
uid=1001(alice) gid=1001(alice)
groups=1001(alice)
alice@dev:~$ keyctl add user secret
hunter2 @u
791615806
```

```
bob@dev:~$ id
uid=1002(bob) gid=1002(bob)
groups=1002(bob)
bob@dev:~$ keyctl newring from-others @u
966722684
bob@dev:~$ keyctl setperm %:from-others
0x3f010004
```

Example: secret sharing

```
alice@dev:~$ id
uid=1001(alice) gid=1001(alice)
groups=1001(alice)
alice@dev:~$ keyctl add user secret
hunter2 @u
791615806
alice@dev:~$ keyctl move %user:secret
@u 966722684
```

```
bob@dev:~$ id
uid=1002(bob) gid=1002(bob)
groups=1002(bob)
bob@dev:~$ keyctl newring from-others @u
966722684
bob@dev:~$ keyctl setperm %:from-others
0x3f010004
```

Example: secret sharing

```
alice@dev:~$ id
uid=1001(alice) gid=1001(alice)
groups=1001(alice)
alice@dev:~$ keyctl add user secret
hunter2 @u
791615806
alice@dev:~$ keyctl move %user:secret
@u 966722684
alice@dev:~$ keyctl show
Session Keyring
 931561702 --alswrv   1001  1001
keyring: _ses
 107607516 --alswrv   1001 65534   \_
keyring: _uid.1001
```

```
bob@dev:~$ id
uid=1002(bob) gid=1002(bob)
groups=1002(bob)
bob@dev:~$ keyctl newring from-others @u
966722684
bob@dev:~$ keyctl setperm %:from-others
0x3f010004
```

Example: secret sharing

```
alice@dev:~$ id
uid=1001(alice) gid=1001(alice)
groups=1001(alice)
alice@dev:~$ keyctl add user secret
hunter2 @u
791615806
alice@dev:~$ keyctl move %user:secret
@u 966722684
alice@dev:~$ keyctl show
Session Keyring
 931561702 --alswrv   1001  1001
keyring: _ses
 107607516 --alswrv   1001 65534   \_
keyring: _uid.1001
```

```
bob@dev:~$ id
uid=1002(bob) gid=1002(bob)
groups=1002(bob)
bob@dev:~$ keyctl newring from-others @u
966722684
bob@dev:~$ keyctl setperm %:from-others
0x3f010004
bob@dev:~$ keyctl print %user:secret
hunter2
```

Example: secret sharing

```
alice@dev:~$ id
uid=1001(alice) gid=1001(alice)
groups=1001(alice)
alice@dev:~$ keyctl add user secret
hunter2 @u
791615806
alice@dev:~$ keyctl move %user:secret
@u 966722684
alice@dev:~$ keyctl show
Session Keyring
 931561702 --alswrv   1001  1001
keyring: _ses
 107607516 --alswrv   1001 65534   \_
keyring: _uid.1001
```

```
bob@dev:~$ id
uid=1002(bob) gid=1002(bob)
groups=1002(bob)
bob@dev:~$ keyctl newring from-others @u
966722684
bob@dev:~$ keyctl setperm %:from-others
0x3f010004
bob@dev:~$ keyctl print %user:secret
hunter2
bob@dev:~$ keyctl show @u
Keyring
 812825228 --alswrv   1002 65534
keyring: _uid.1002
 966722684 --alswrv   1002  1002   \_
keyring: from-others
 791615806 --alswrv   1001  1001
\_ user: secret
```

Special keyring types

- Process keyrings:
  - session keyring: current and all child processes
  - process keyring: private to the process
  - thread keyring: private to the thread

Special keyring types

- Process keyrings:
  - session keyring: current and all child processes
  - process keyring: private to the process
  - thread keyring: private to the thread
- User keyrings:
  - user keyring: shared between all processes with a UID
  - user session keyring: similar to user keyring

Special keyring types

- Process keyrings:
  - session keyring: current and all child processes
  - process keyring: private to the process
  - thread keyring: private to the thread
- User keyrings:
  - user keyring: shared between all processes with a UID
  - user session keyring: similar to user keyring
- Persistent keyrings:
  - shared between all processes with a UID
  - does not get destroyed, when last process with a UID exits
  - “expires” after a timeout, if not accessed before
    - for various non-interactive tasks, like cron jobs

Session keyring example

```
ignat@dev:~$ keyctl add user secret hunter2 @s
603482993
```

Session keyring example

```
ignat@dev:~$ keyctl add user secret hunter2 @s
603482993
ignat@dev:~$ keyctl show
Session Keyring
 464596277 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 603482993 --alswrv   1000  1000   \_ user: secret
```

Session keyring example

```
ignat@dev:~$ keyctl add user secret hunter2 @s
603482993
ignat@dev:~$ keyctl show
Session Keyring
 464596277 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 603482993 --alswrv   1000  1000   \_ user: secret
```

Session keyring example

```
ignat@dev:~$ keyctl add user secret hunter2 @s
603482993
ignat@dev:~$ keyctl show
Session Keyring
 464596277 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 603482993 --alswrv   1000  1000   \_ user: secret
```

```
ignat@dev:~$ sudo bpftrace -e 'kprobe:user_destroy { printf("destroying key %d\n", ((struct
key *)arg0)->serial) }'
Attaching 1 probe...
```

Session keyring example

```
ignat@dev:~$ keyctl add user secret hunter2 @s
603482993
ignat@dev:~$ keyctl show
Session Keyring
 464596277 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 603482993 --alswrv   1000  1000   \_ user: secret
ignat@dev:~$ exit
logout
Connection to dev closed.
```

```
ignat@dev:~$ sudo bpftrace -e 'kprobe:user_destroy { printf("destroying key %d\n", ((struct
key *)arg0)->serial) }'
Attaching 1 probe...
destroying key 603482993
```

Session keyring example

```
ignat@dev:~$ keyctl add user secret hunter2 @s
603482993
ignat@dev:~$ keyctl show
Session Keyring
 464596277 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 603482993 --alswrv   1000  1000   \_ user: secret
ignat@dev:~$ exit
logout
Connection to dev closed.
$ ssh dev
```

```
ignat@dev:~$ sudo bpftrace -e 'kprobe:user_destroy { printf("destroying key %d\n", ((struct key *)arg0)->serial) }'
Attaching 1 probe...
destroying key 603482993
```

Session keyring example

```
ignat@dev:~$ keyctl add user secret hunter2 @s
603482993
ignat@dev:~$ keyctl show
Session Keyring
 464596277 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 603482993 --alswrv   1000  1000   \_ user: secret
ignat@dev:~$ exit
logout
Connection to dev closed.
$ ssh dev
ignat@dev:~$ keyctl show
Session Keyring
 523682608 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
```

```
ignat@dev:~$ sudo bpftrace -e 'kprobe:user_destroy { printf("destroying key %d\n", ((struct
key *)arg0)->serial) }'
Attaching 1 probe...
destroying key 603482993
```

Session keyring example

```
ignat@dev:~$ keyctl add user secret hunter2 @s
603482993
ignat@dev:~$ keyctl show
Session Keyring
 464596277 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
 603482993 --alswrv   1000  1000   \_ user: secret
ignat@dev:~$ exit
logout
Connection to dev closed.
$ ssh dev
ignat@dev:~$ keyctl show
Session Keyring
 523682608 --alswrv   1000  1000  keyring: _ses
 517020096 --alswrv   1000 65534   \_ keyring: _uid.1000
```

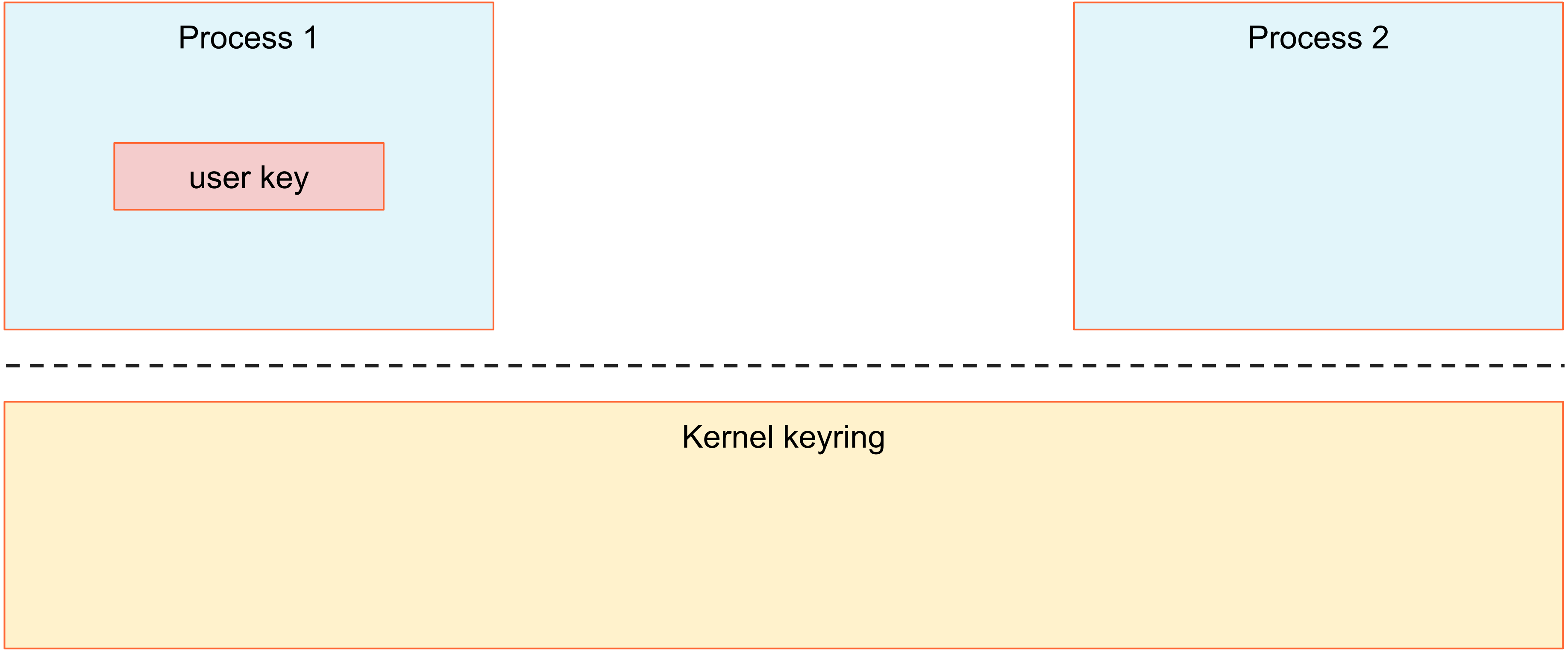
```
ignat@dev:~$ sudo bpftrace -e 'kprobe:user_destroy { printf("destroying key %d\n", ((struct
key *)arg0)->serial) }'
Attaching 1 probe...
destroying key 603482993
```

Special keyring types

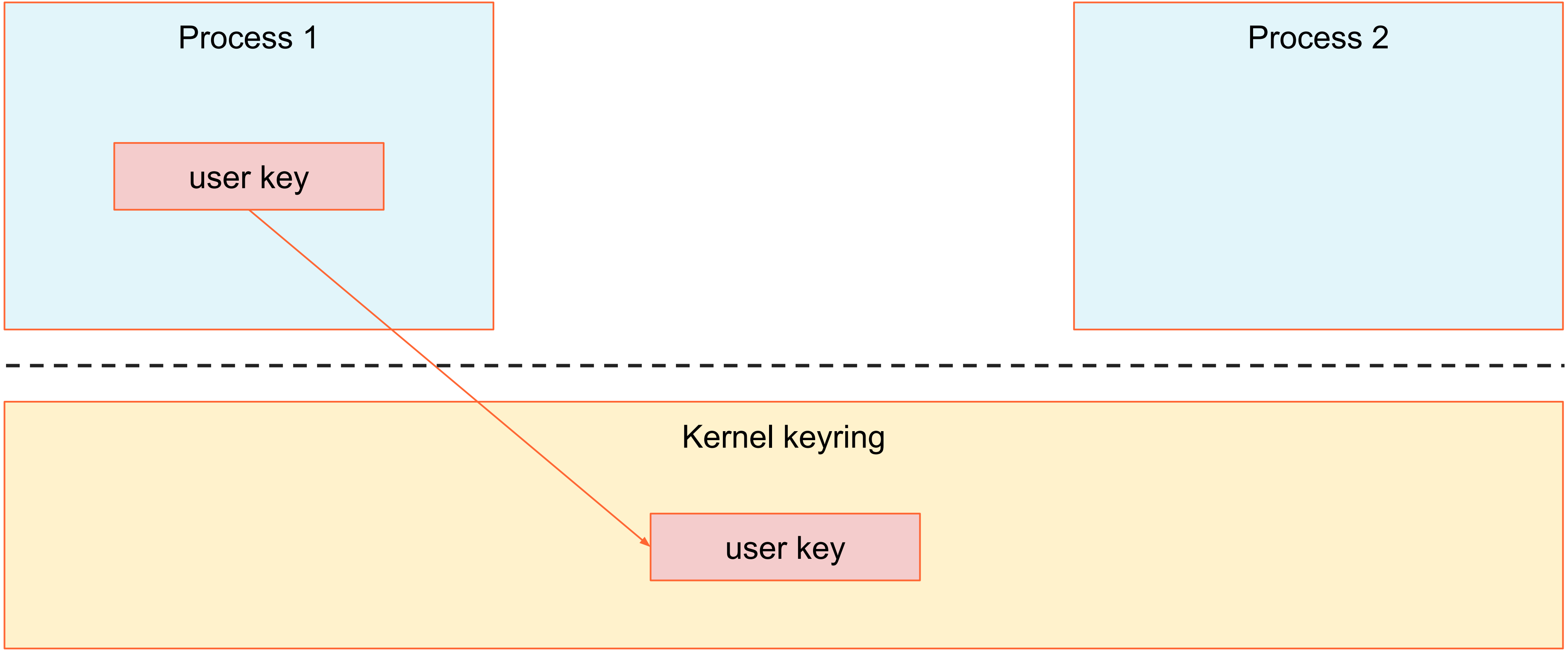
By selecting the appropriate keyring type you can ensure the keys will be securely destroyed, when not needed

Even if the application crashes!

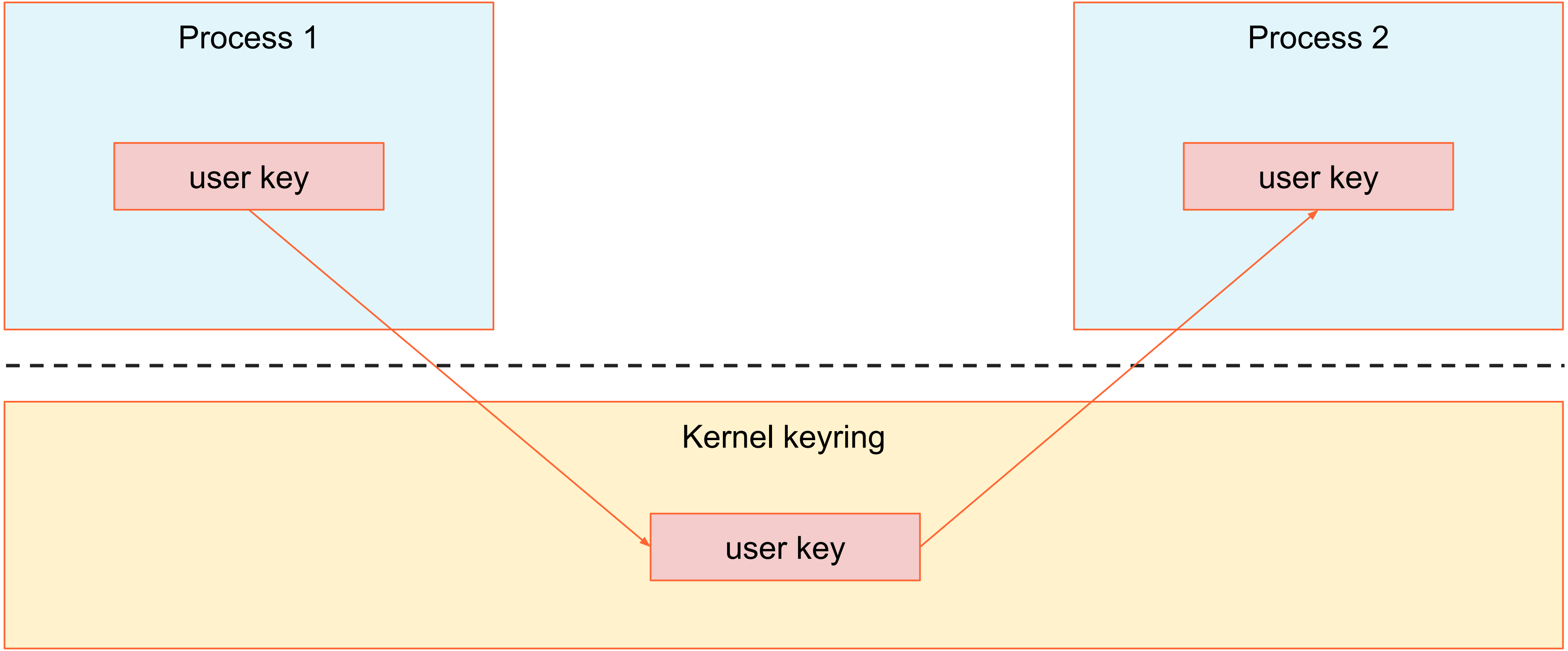
User keys



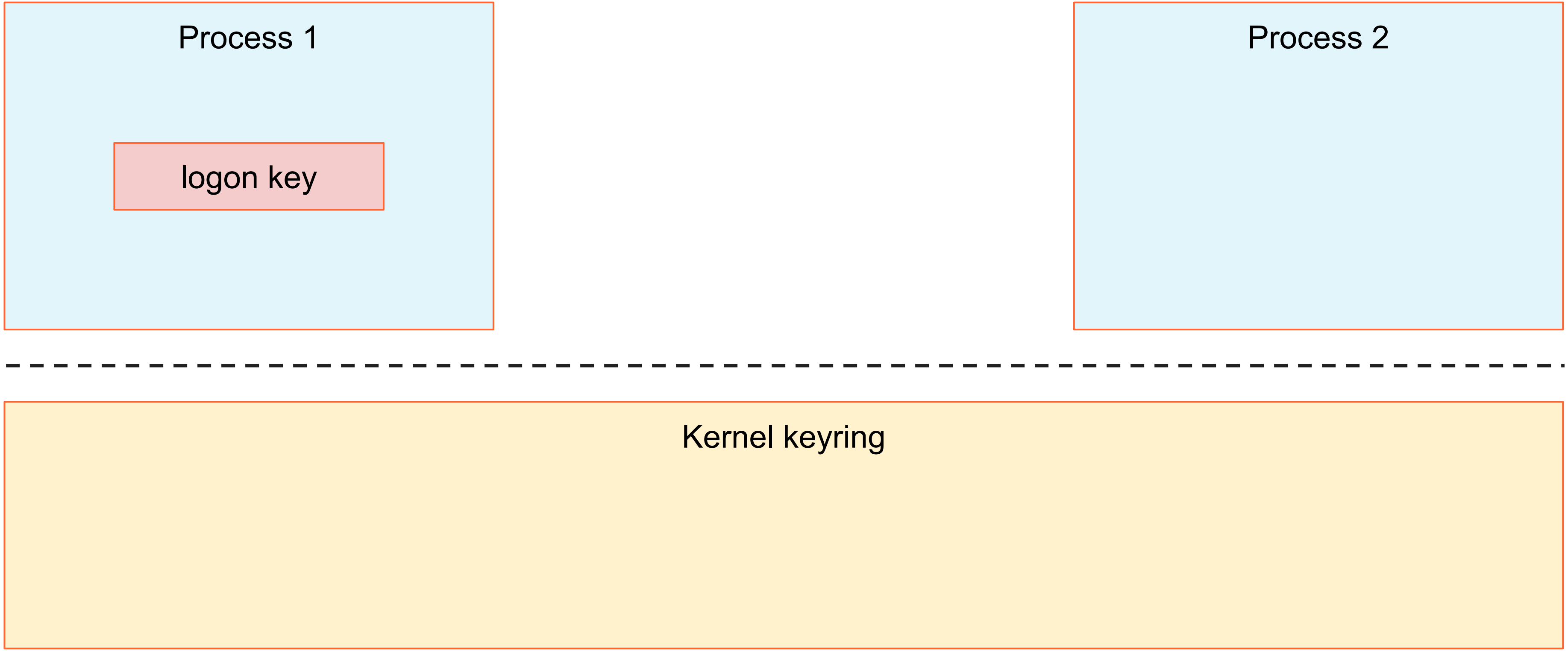
User keys



User keys

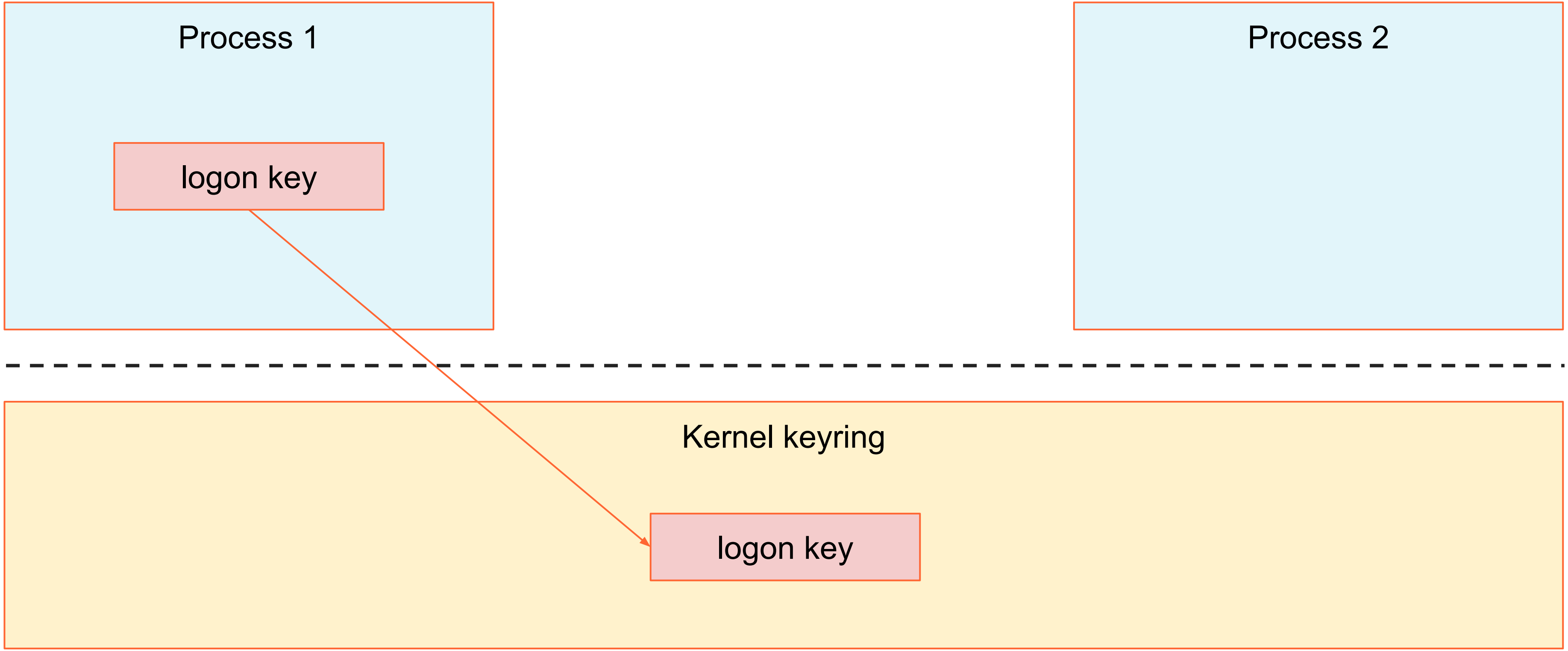

Process 1
user key
Process 2
user key
Kernel keyring
user key

Logon keys

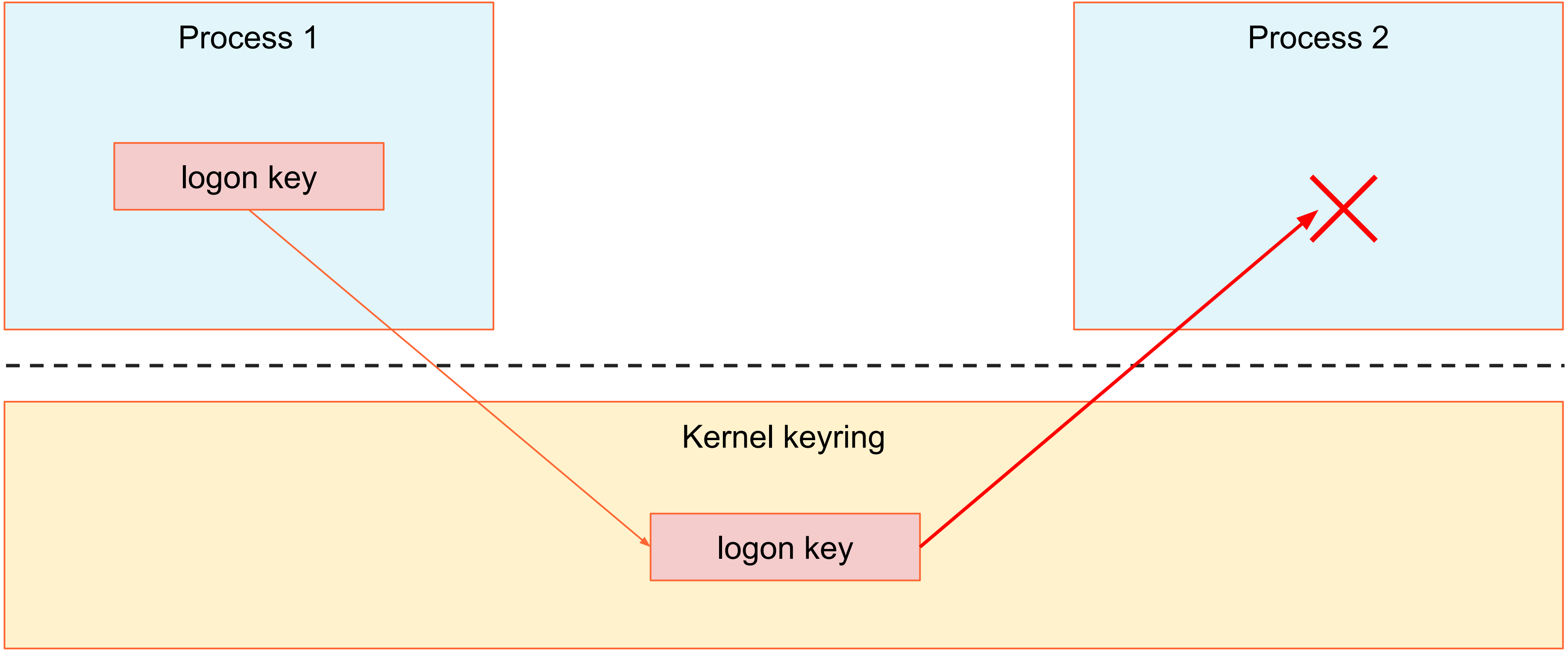


@ignatkn

Logon keys

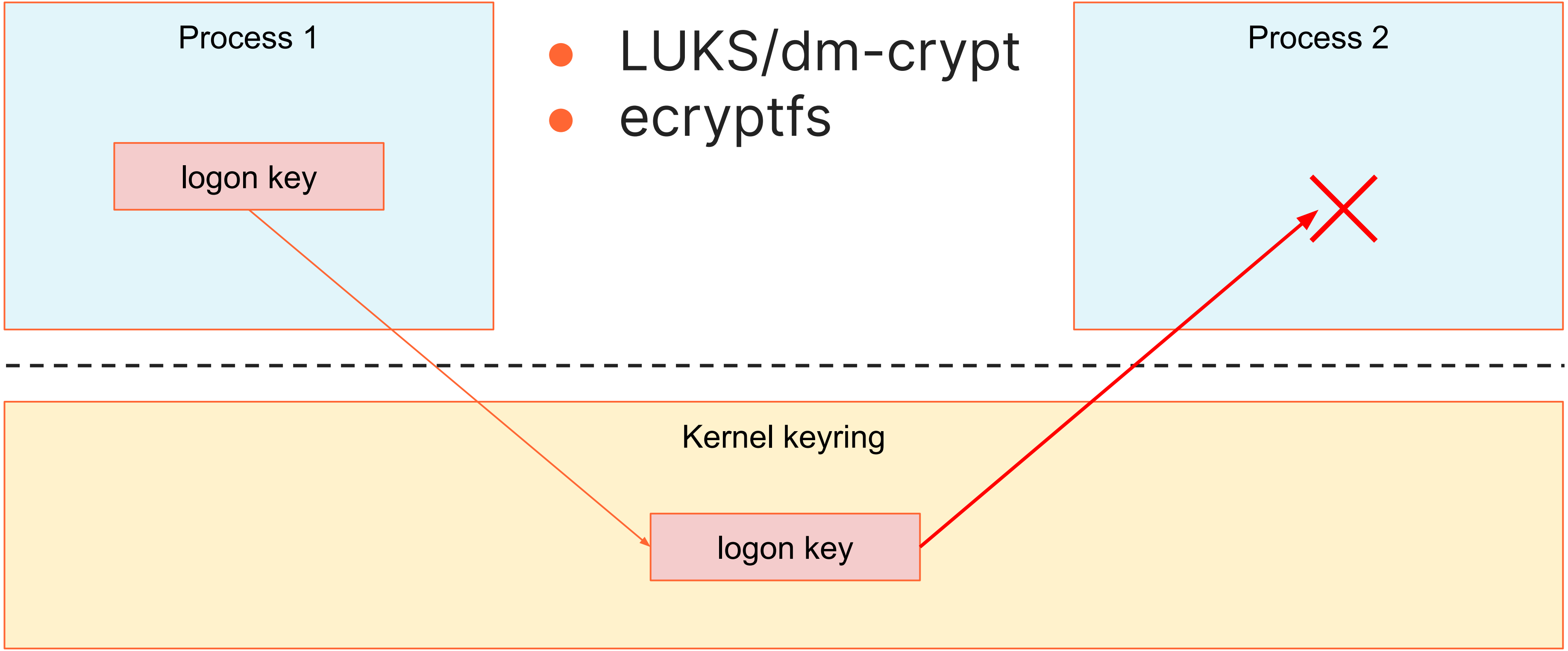


Logon keys



Logon keys

- LUKS/dm-crypt
- ecryptfs



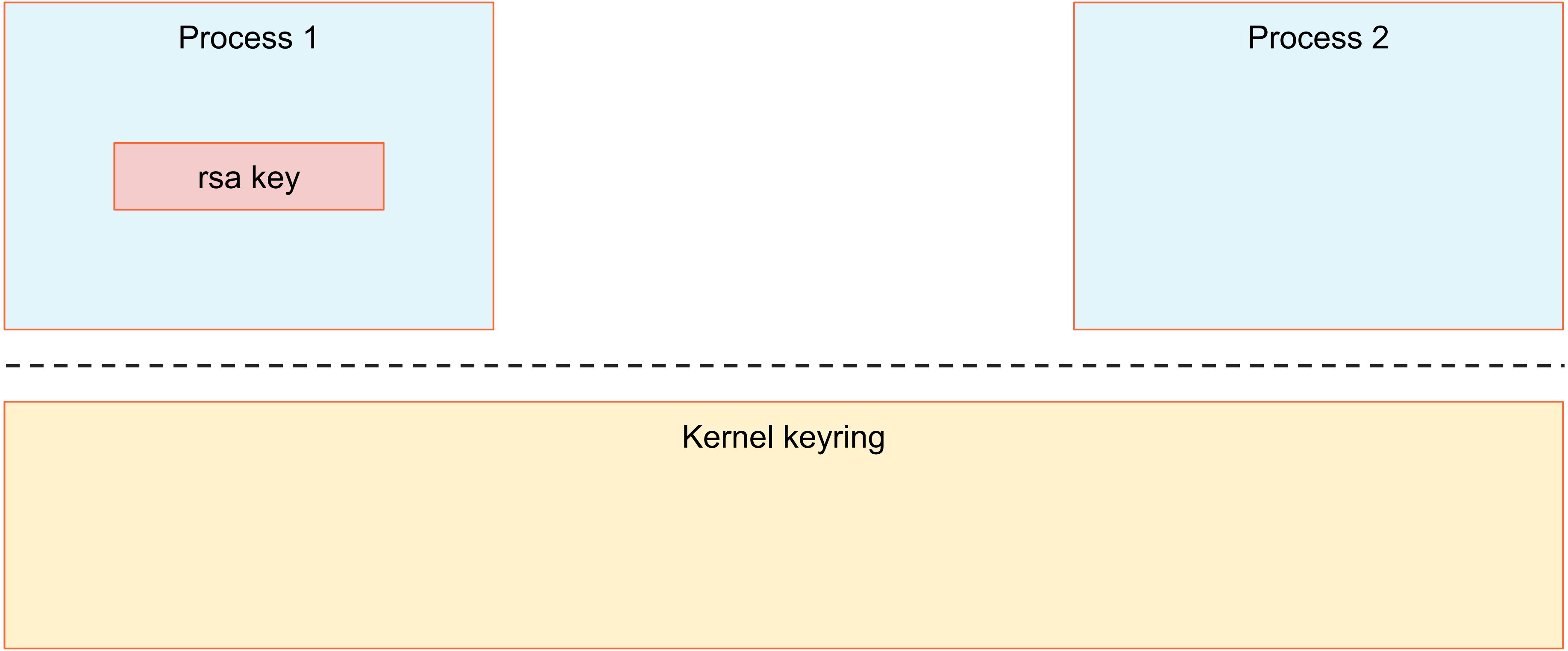
Logon keys in LUKS/dm-crypt

```
ignat@dev:~$ sudo dmsetup table
luks-sda: 0 937670320 crypt aes-xts-plain64
:64:logon:cryptsetup:8f5af694-c4ce-4ed0-89a8-386f67980f70-d0 0
8:0 32768
luks-sdb: 0 937670320 crypt aes-xts-plain64
:64:logon:cryptsetup:e76176e1-b819-40a8-b92a-618cce2cffe5-d0 0
8:16 32768
```

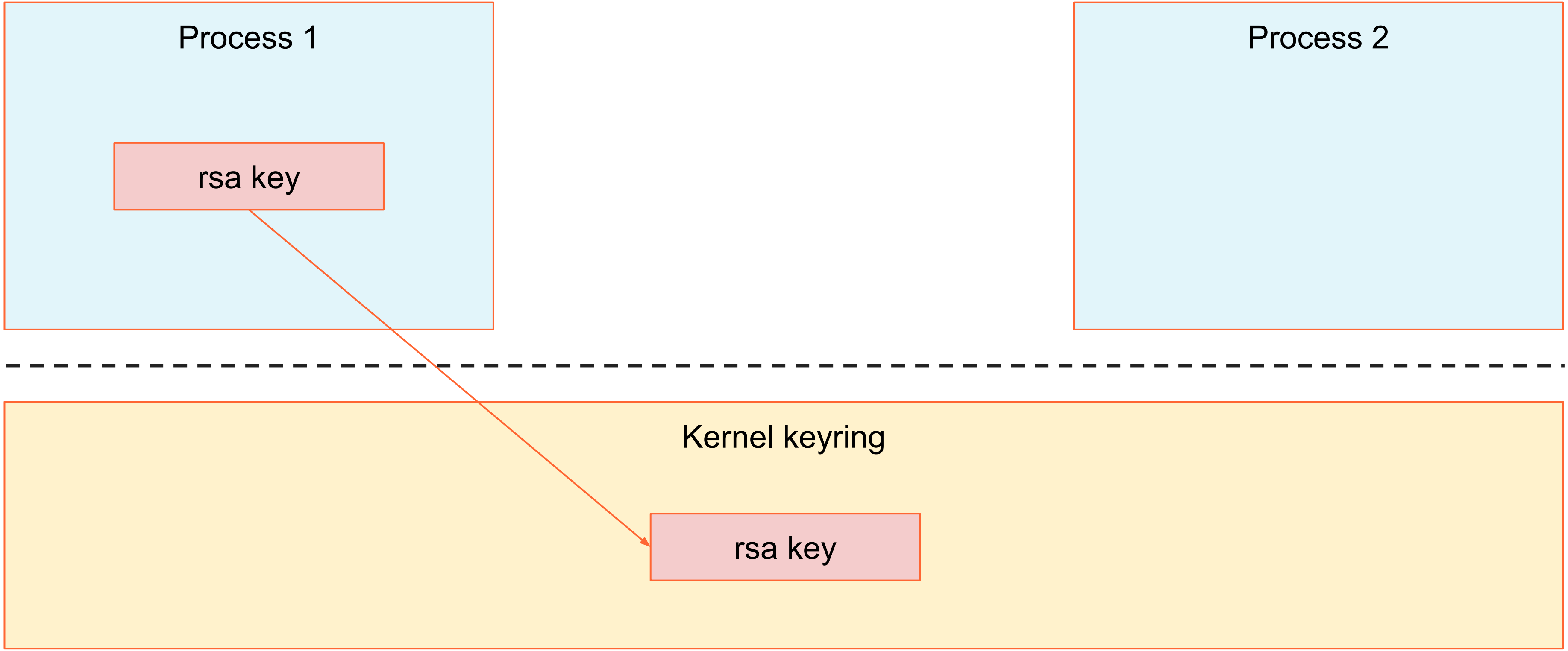
Logon keys in LUKS/dm-crypt

```
ignat@dev:~$ sudo dmsetup table
luks-sda: 0 937670320 crypt aes-xts-plain64
:64:logon:cryptsetup:8f5af694-c4ce-4ed0-89a8-386f67980f70-d0 0
8:0 32768
luks-sdb: 0 937670320 crypt aes-xts-plain64
:64:logon:cryptsetup:e76176e1-b819-40a8-b92a-618cce2cffe5-d0 0
8:16 32768
```

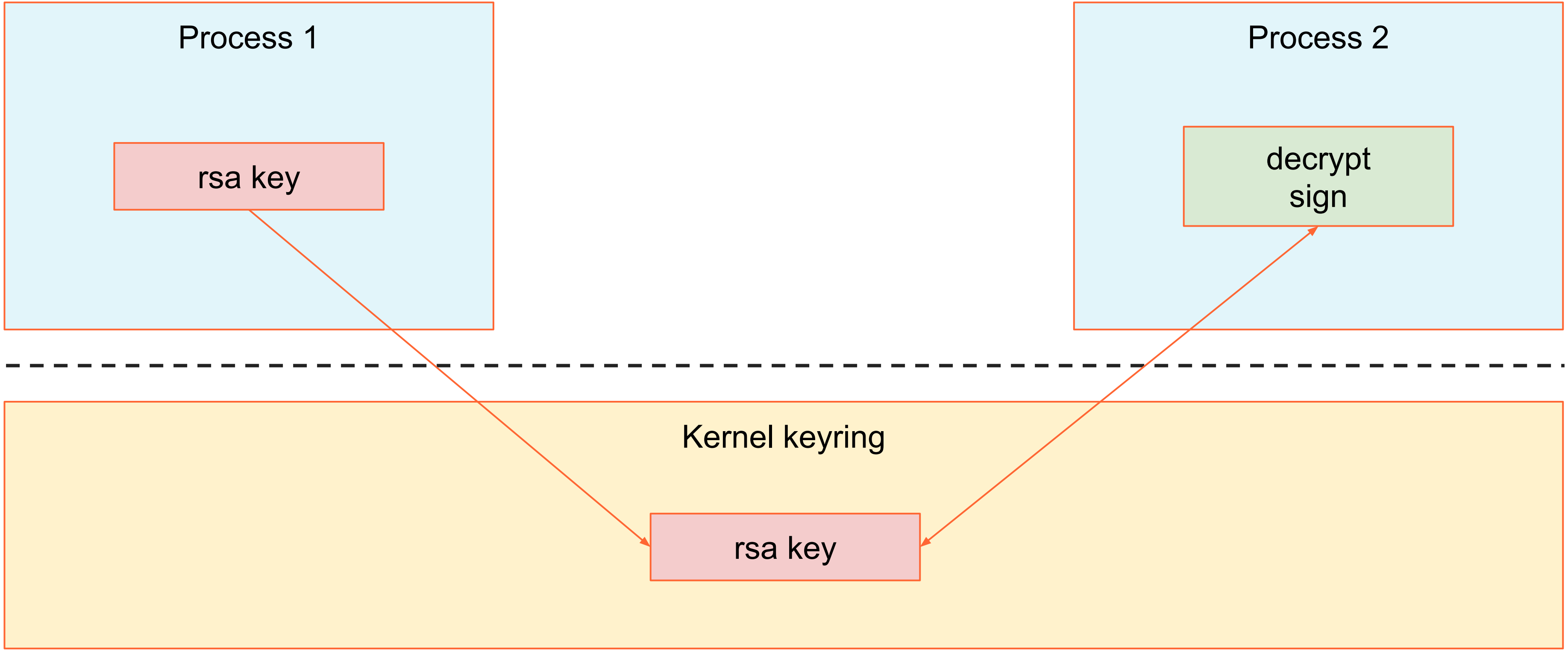
Asymmetric keys



Asymmetric keys



Asymmetric keys



Asymmetric key example (ssh-agent replacement)

```
ignat@dev:~$ openssl genrsa -out priv.pem
Generating RSA private key, 2048 bit long modulus (2 primes)
..........+++++
...........+++++
e is 65537 (0x010001)
ignat@dev:~$ openssl rsa -in priv.pem -pubout -out pub.pem
writing RSA key
```

Asymmetric key example (ssh-agent replacement)

```
ignat@dev:~$ openssl genrsa -out priv.pem
Generating RSA private key, 2048 bit long modulus (2 primes)
..........+++++
...........+++++
e is 65537 (0x010001)
ignat@dev:~$ openssl rsa -in priv.pem -pubout -out pub.pem
writing RSA key
ignat@dev:~$ openssl pkcs8 -in priv.pem -topk8 -outform DER -nocrypt -out
priv.p8
```

Asymmetric key example (ssh-agent replacement)

```
ignat@dev:~$ openssl genrsa -out priv.pem
Generating RSA private key, 2048 bit long modulus (2 primes)
..........+++++
...........+++++
e is 65537 (0x010001)
ignat@dev:~$ openssl rsa -in priv.pem -pubout -out pub.pem
writing RSA key
ignat@dev:~$ openssl pkcs8 -in priv.pem -topk8 -outform DER -nocrypt -out
priv.p8
ignat@dev:~$ cat priv.p8 | keyctl padd asymmetric "rsa-key" @s
717848853
```

CLOUDFLARE

Asymmetric key example (ssh-agent replacement)

```
ignat@dev:~$ openssl genrsa -out priv.pem
Generating RSA private key, 2048 bit long modulus (2 primes)
..........+++++
...........+++++
e is 65537 (0x010001)
ignat@dev:~$ openssl rsa -in priv.pem -pubout -out pub.pem
writing RSA key
ignat@dev:~$ openssl pkcs8 -in priv.pem -topk8 -outform DER -nocrypt -out
priv.p8
ignat@dev:~$ cat priv.p8 | keyctl padd asymmetric "rsa-key" @s
717848853
ignat@dev:~$ echo abc | openssl sha256 -binary > abc.sha256
ignat@dev:~$ keyctl pkey_sign %asymmetric:rsa-key 0 abc.sha256 enc=pkcs1
hash=sha256 >abc.sig
```

Asymmetric key example (ssh-agent replacement)

```
ignat@dev:~$ openssl genrsa -out priv.pem
Generating RSA private key, 2048 bit long modulus (2 primes)
..........+++++
...........+++++
e is 65537 (0x010001)
ignat@dev:~$ openssl rsa -in priv.pem -pubout -out pub.pem
writing RSA key
ignat@dev:~$ openssl pkcs8 -in priv.pem -topk8 -outform DER -nocrypt -out
priv.p8
ignat@dev:~$ cat priv.p8 | keyctl padd asymmetric "rsa-key" @s
717848853
ignat@dev:~$ echo abc | openssl sha256 -binary > abc.sha256
ignat@dev:~$ keyctl pkey_sign %asymmetric:rsa-key 0 abc.sha256 enc=pkcs1
hash=sha256 >abc.sig
ignat@dev:~$ echo abc | openssl sha256 -verify pub.pem -signature abc.sig
Verified OK
```

Asymmetric key example (ssh-agent replacement)

https://blog.cloudflare.com/the-linux-kernel-key-retention-service-and-why-you-should-use-it-in-your-next-application/

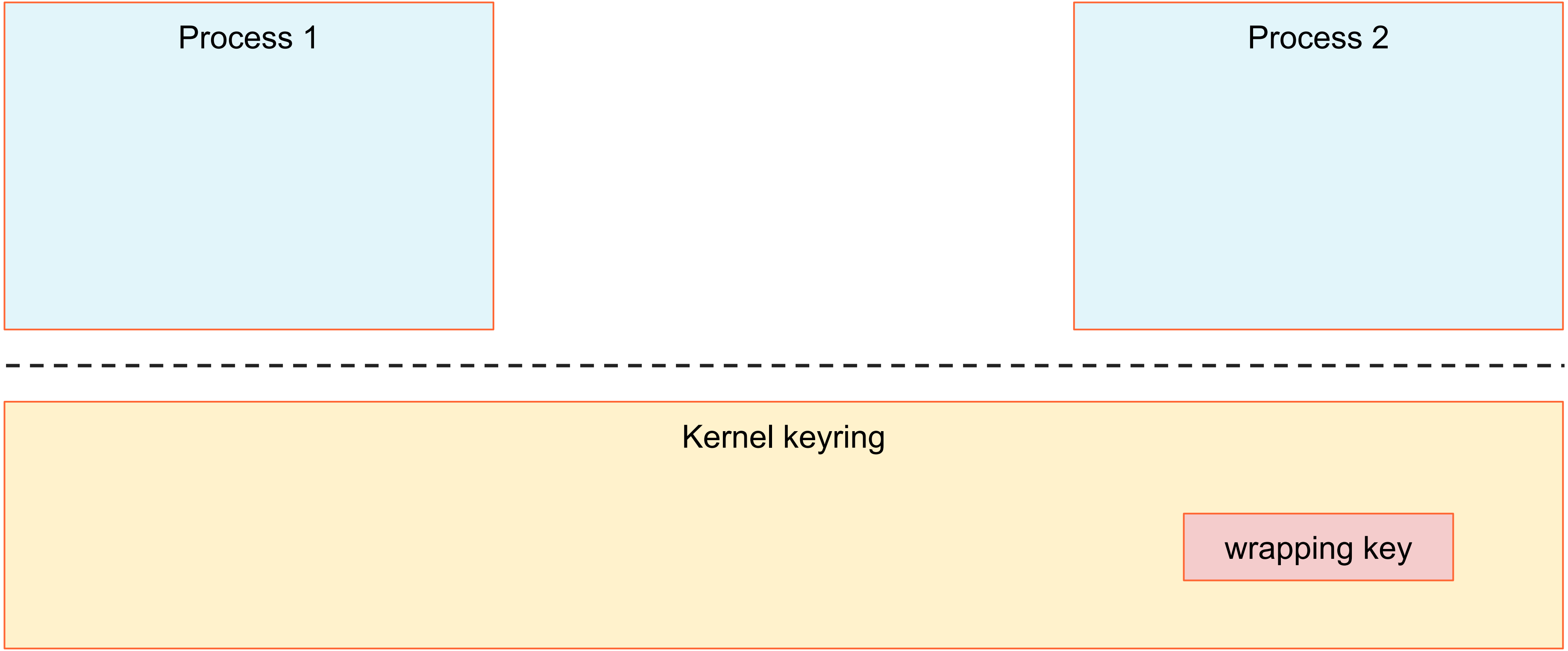
Keystore as a key management building block

Secure key distribution and provisioning

Minimizing cryptographic material exposure

How can we provision application keys without the cryptographic material ever being exposed to the userspace applications?

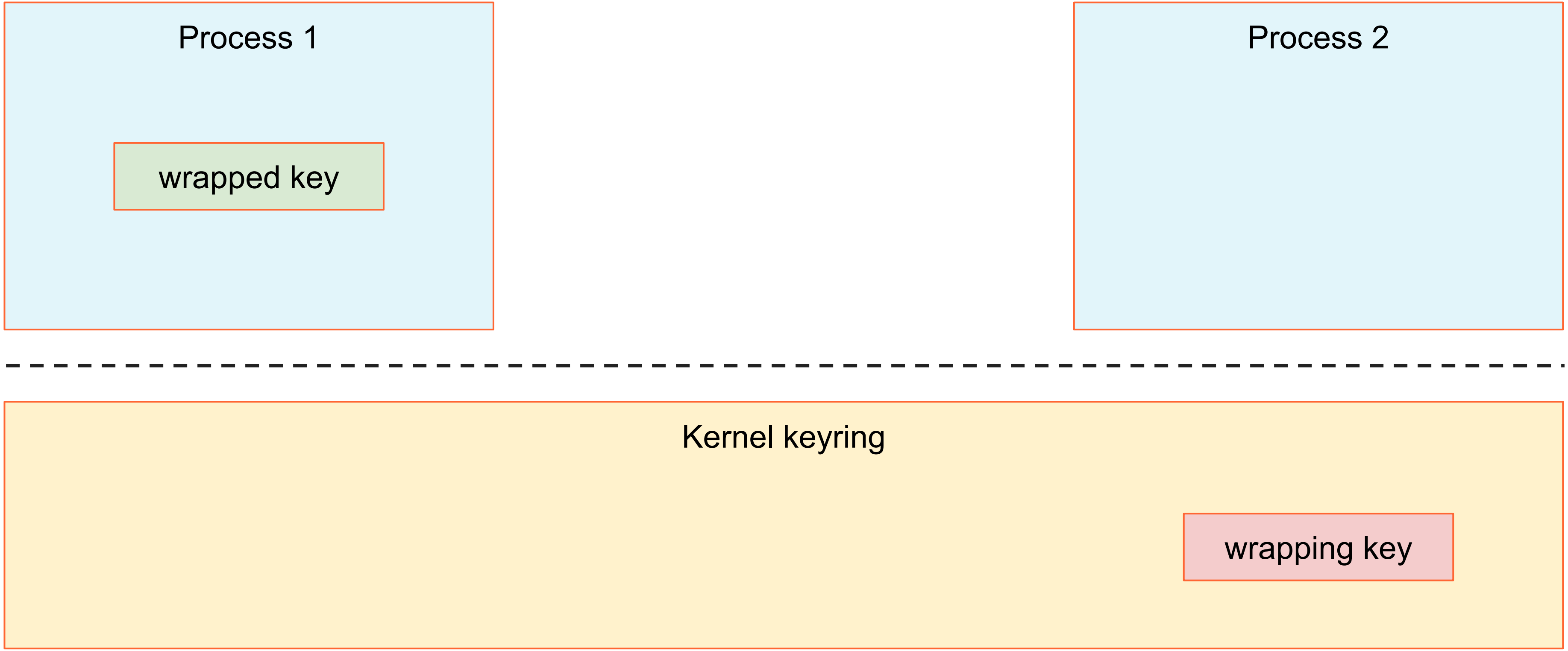
Encrypted keys



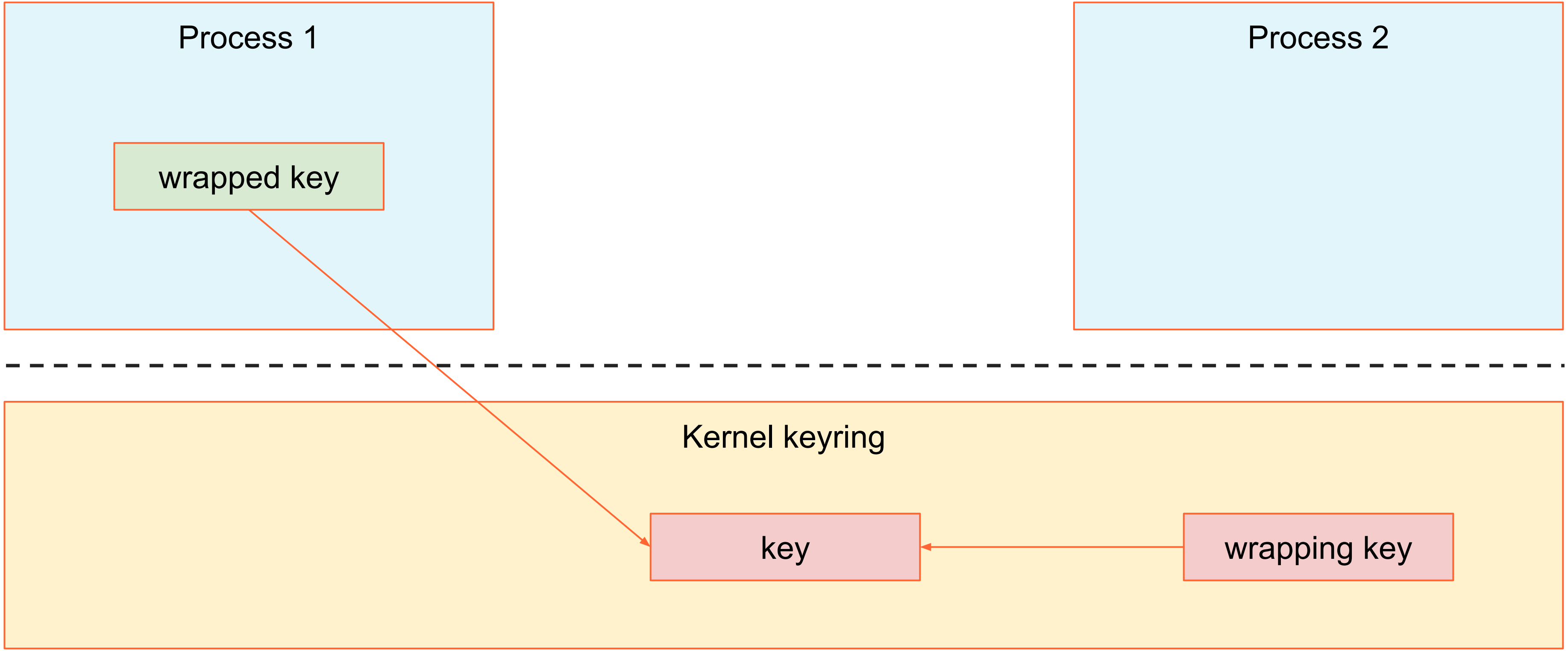
@ignatkn

CLOUDFLARE

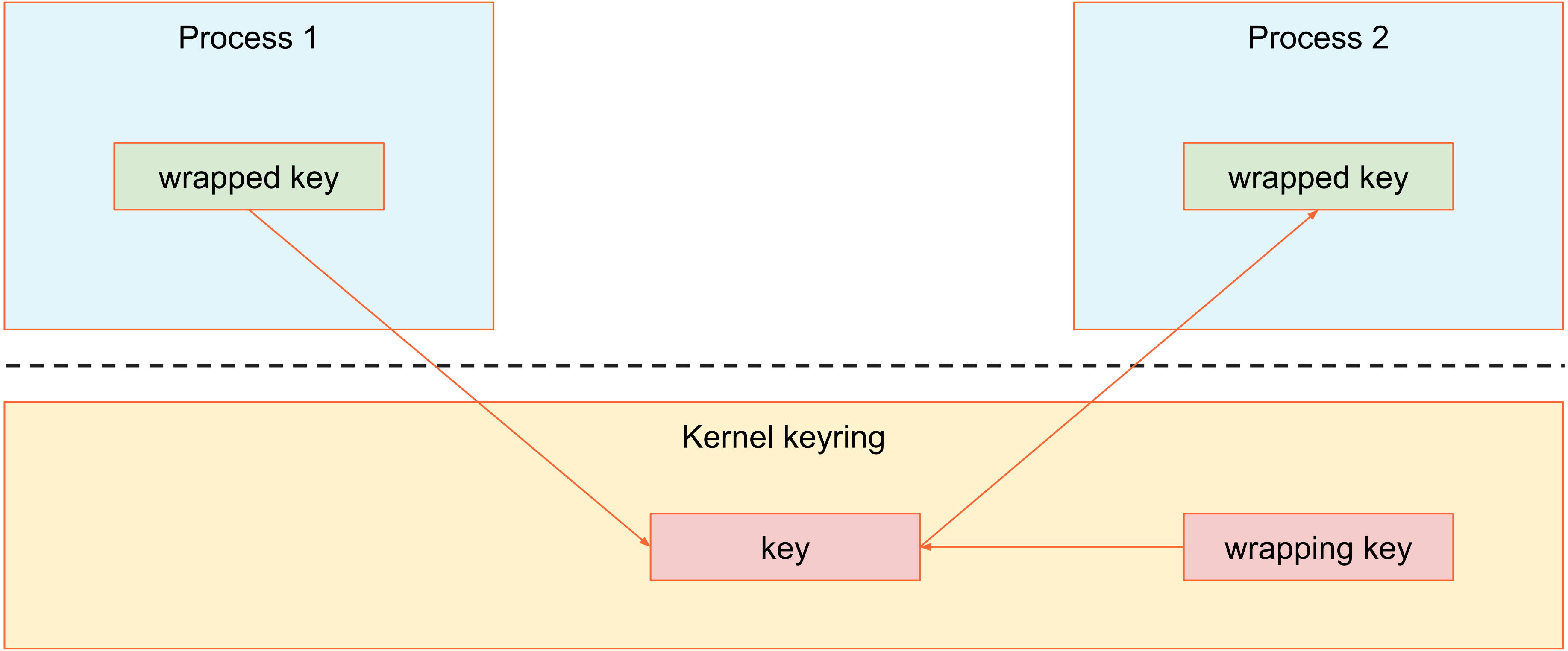
Encrypted keys



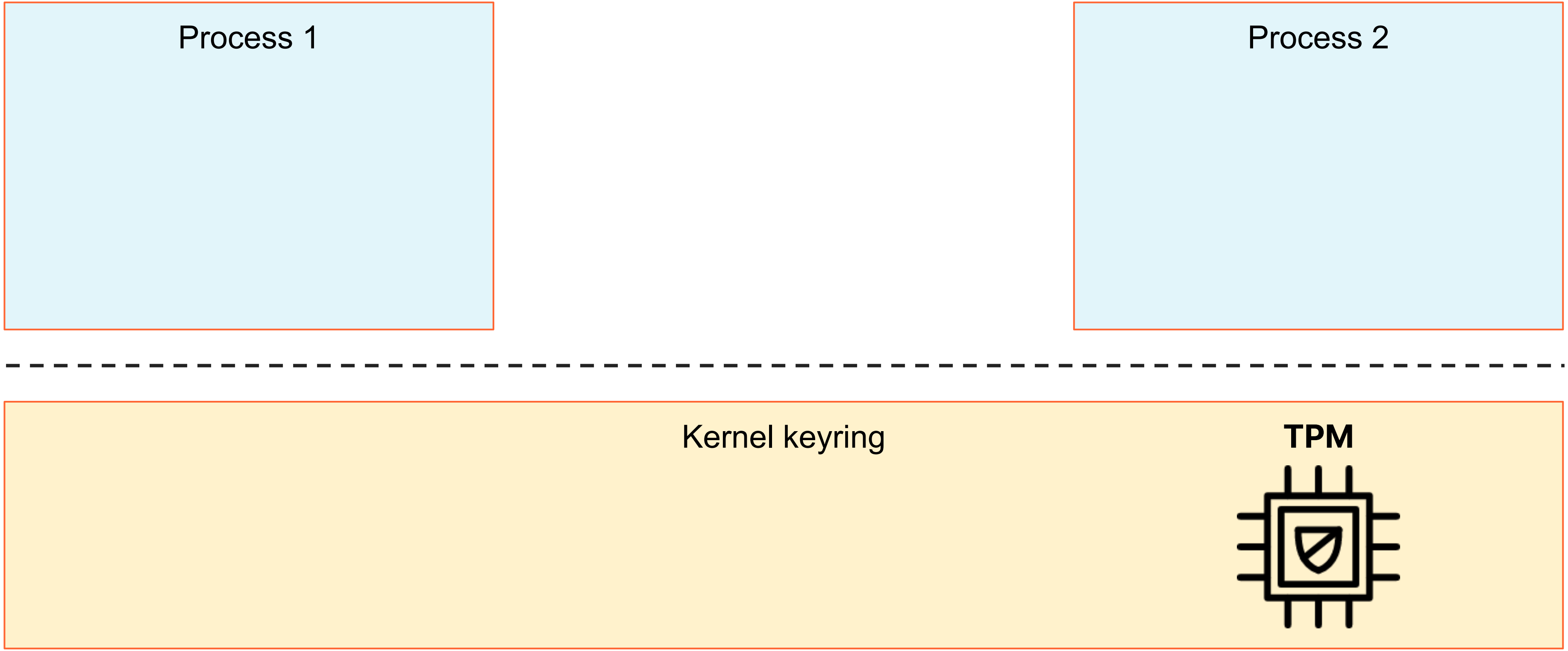
Encrypted keys



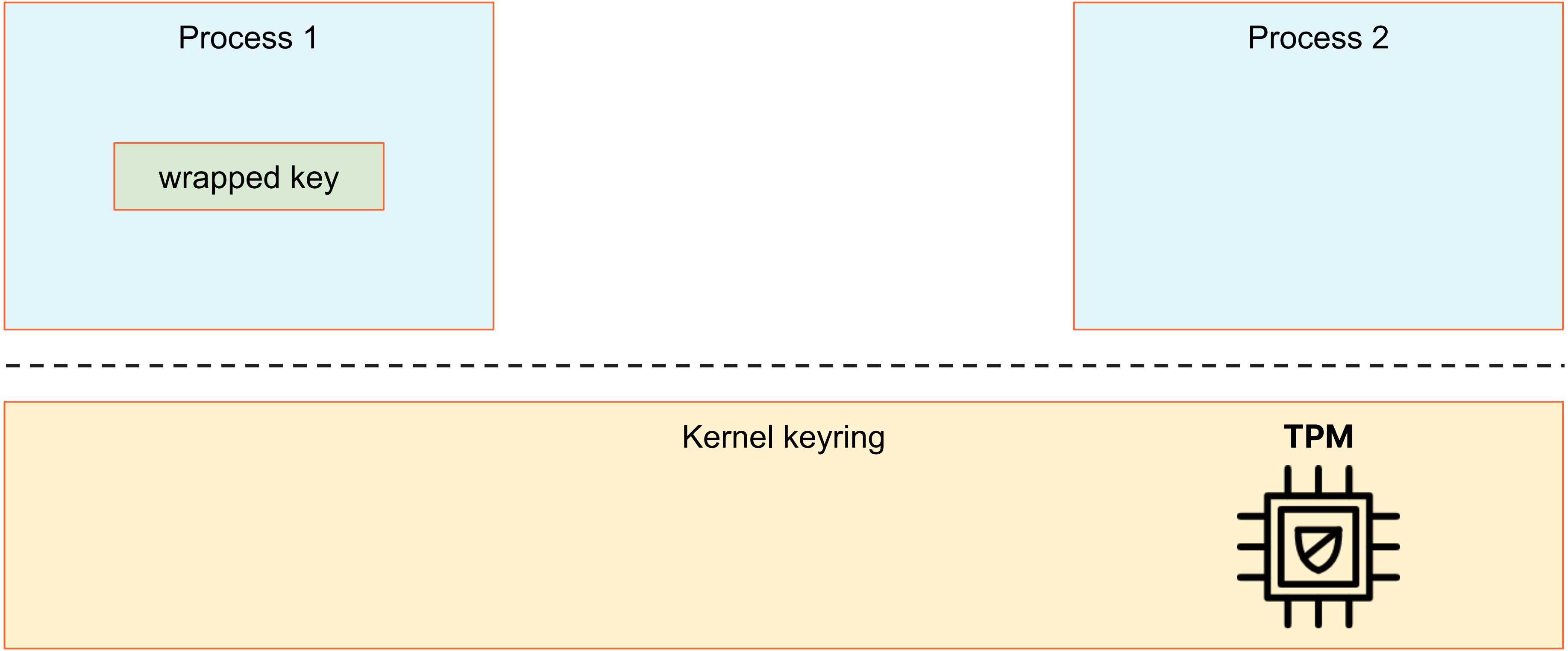
Encrypted keys



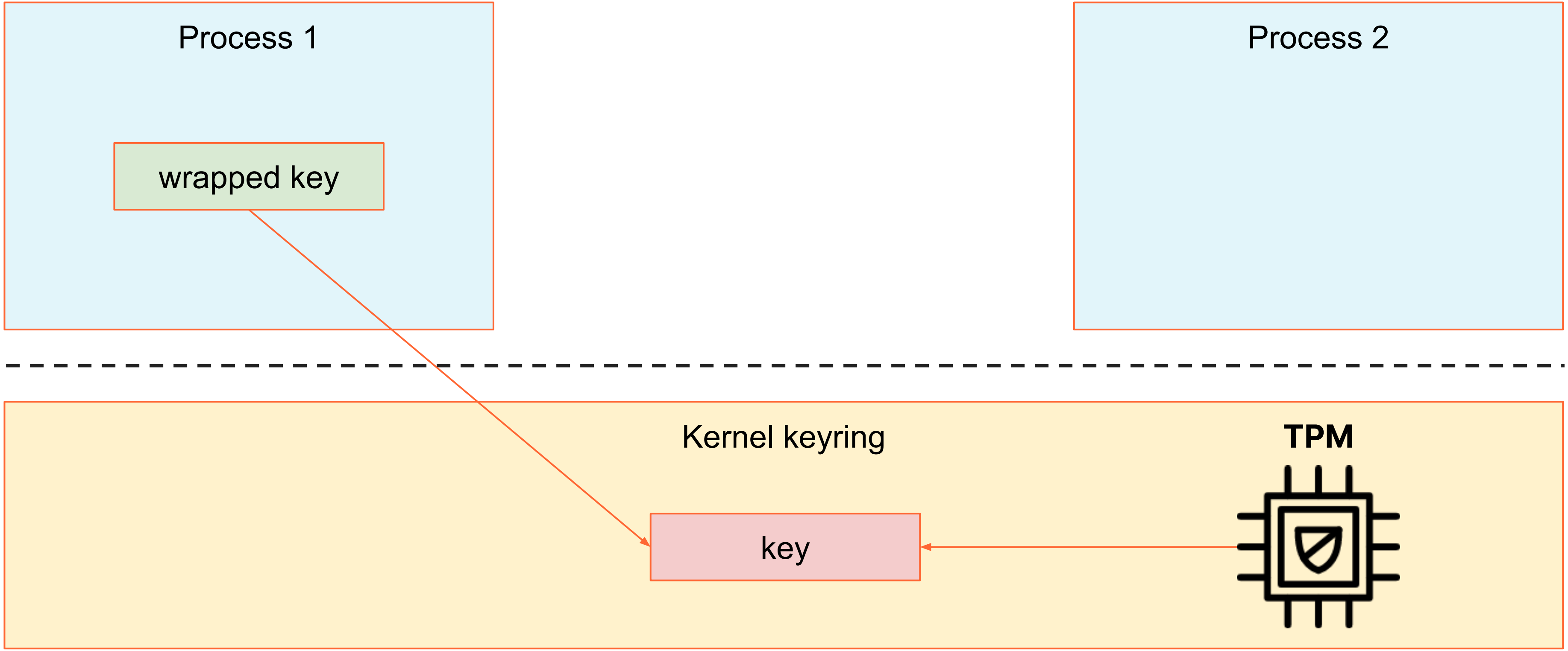
Trusted keys



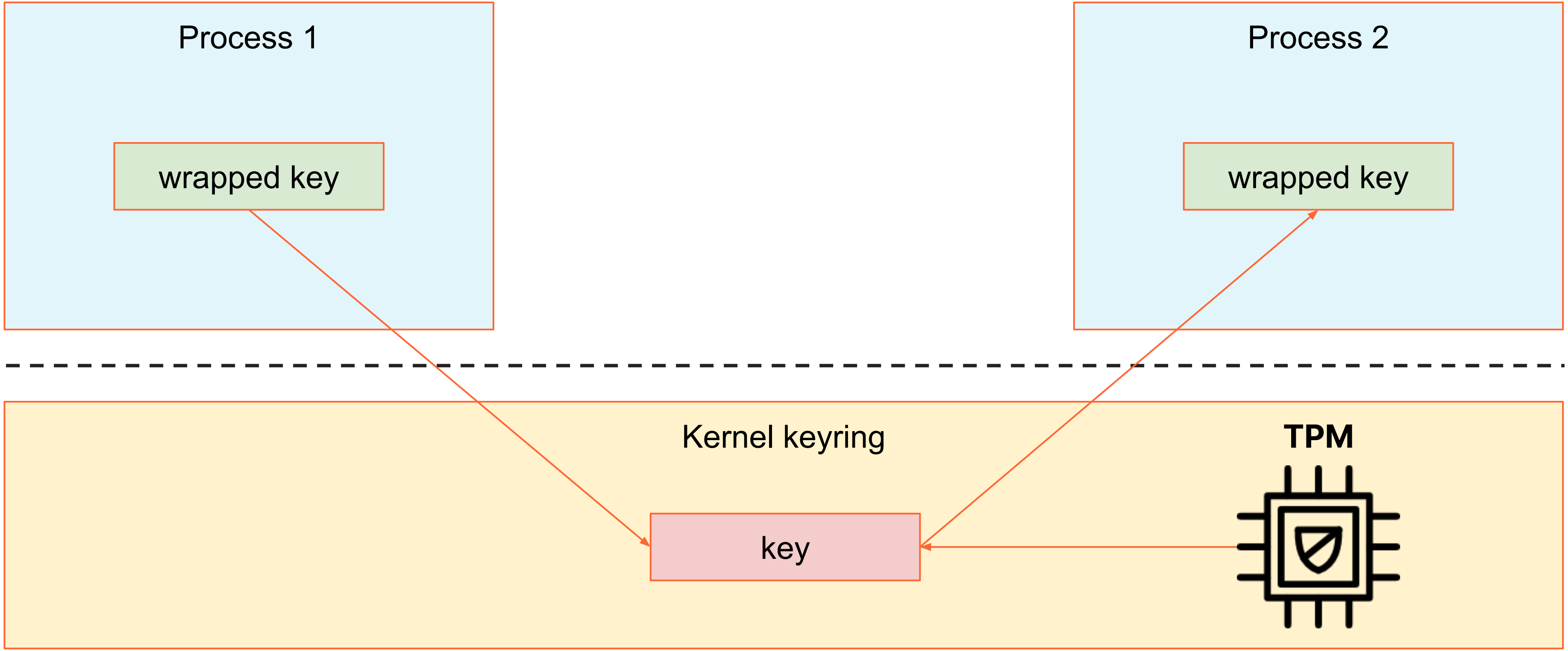
Trusted keys



Trusted keys



Trusted keys

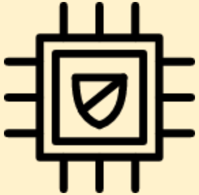


Combined schema

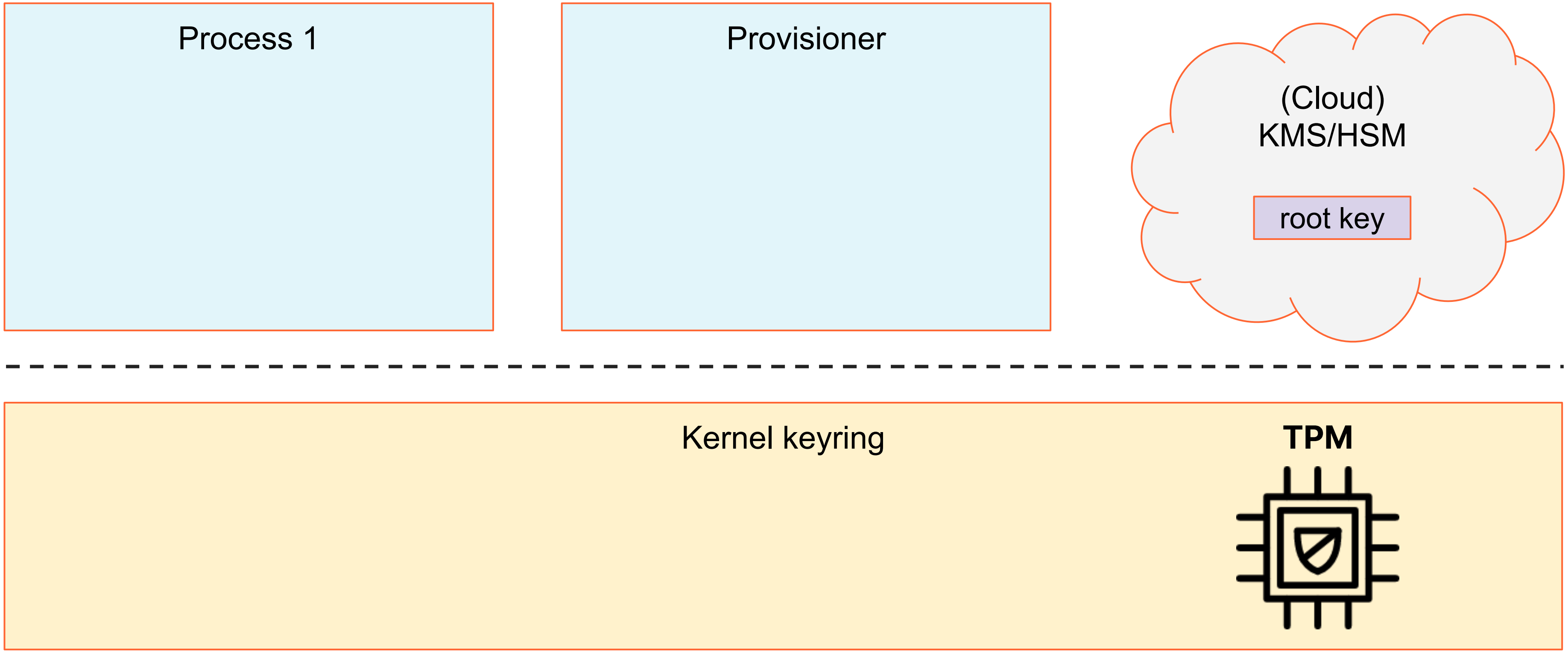
Process 1

Kernel keyring

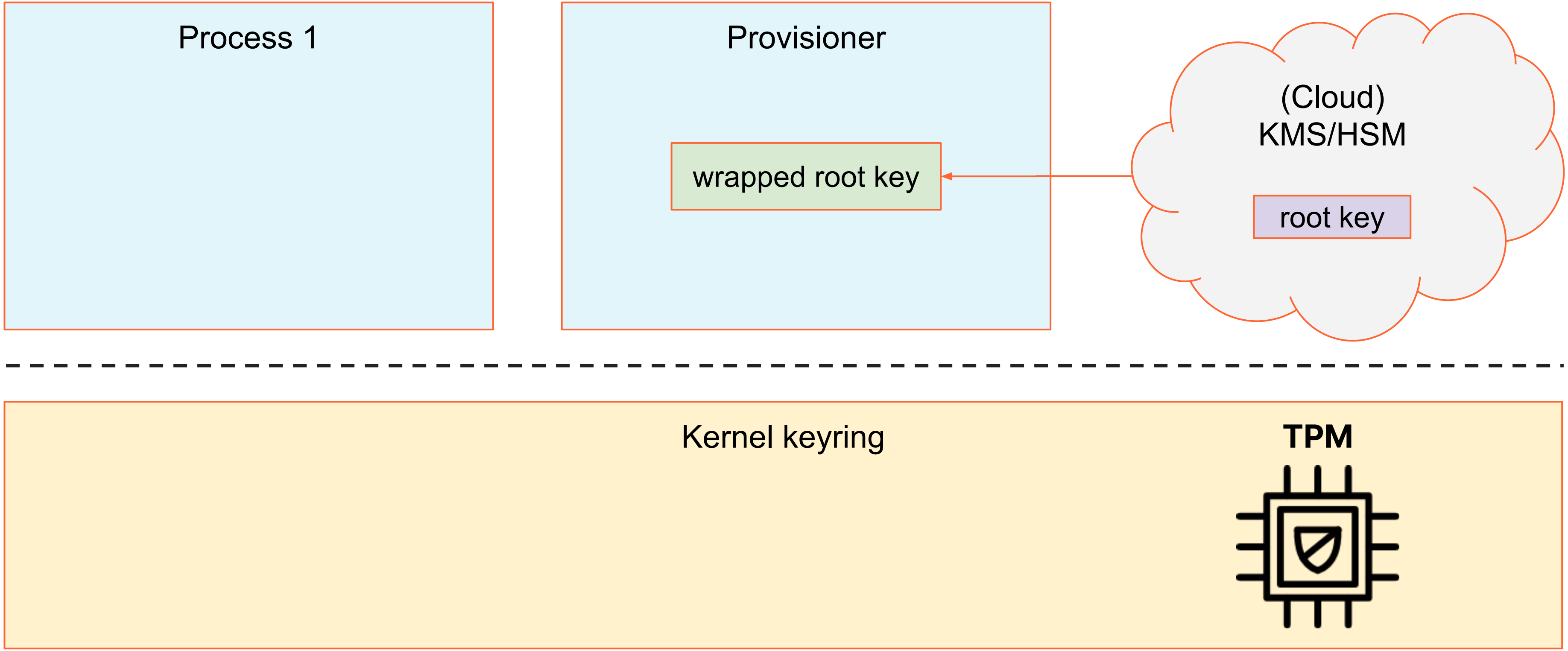
TPM



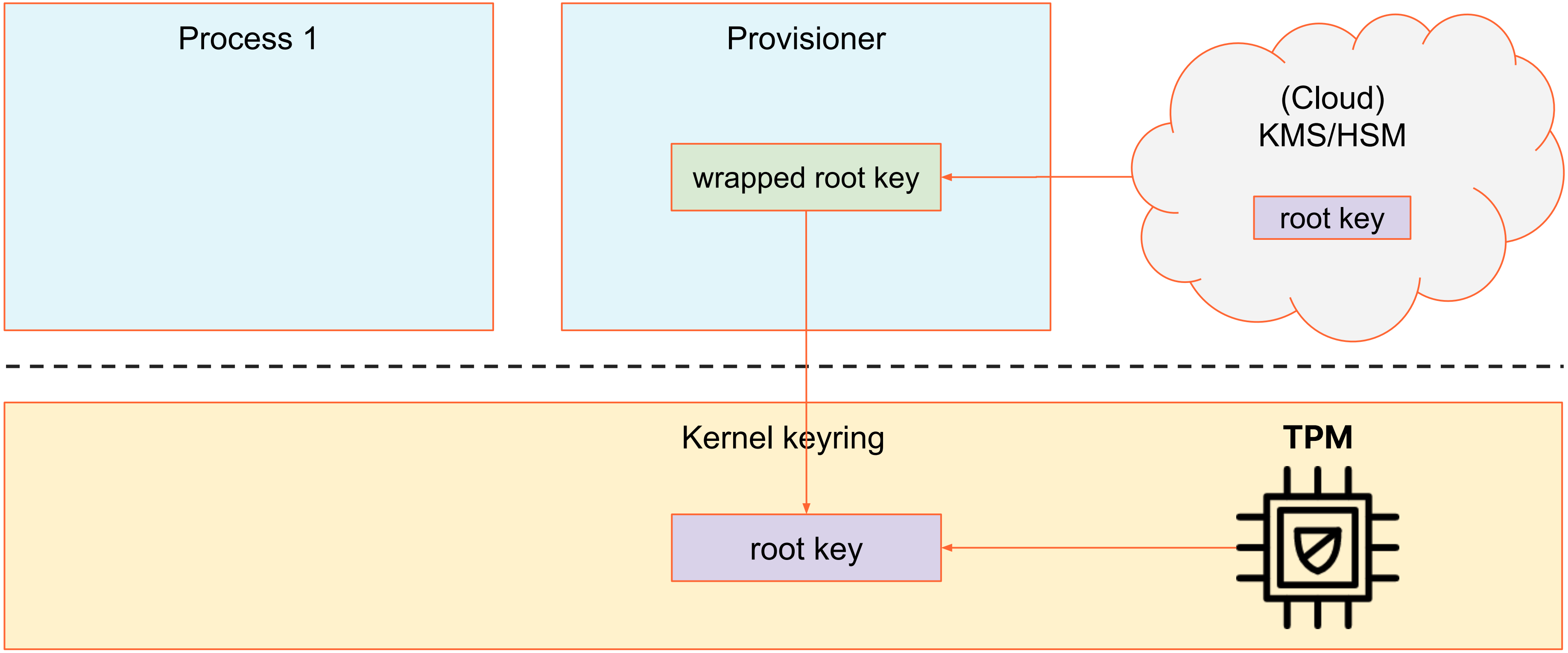
Combined schema



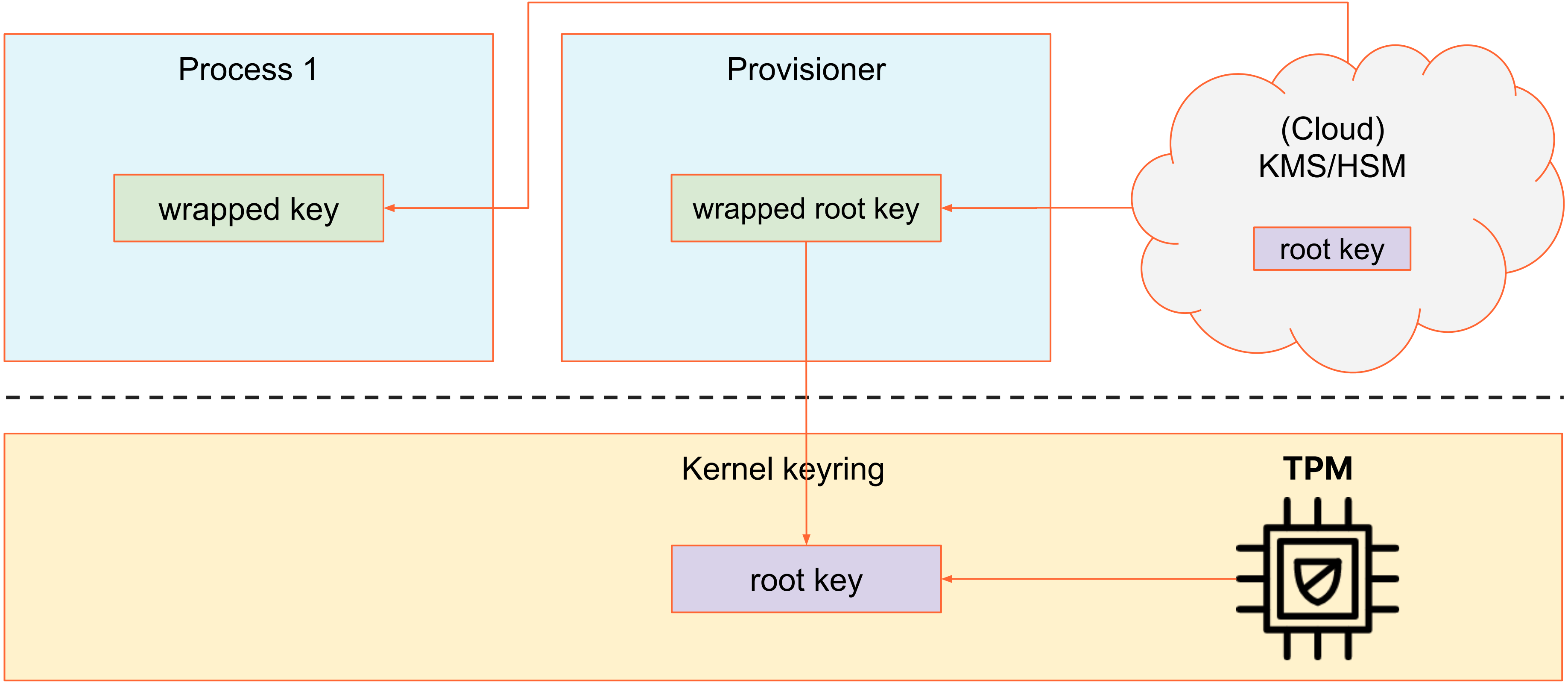
Combined schema



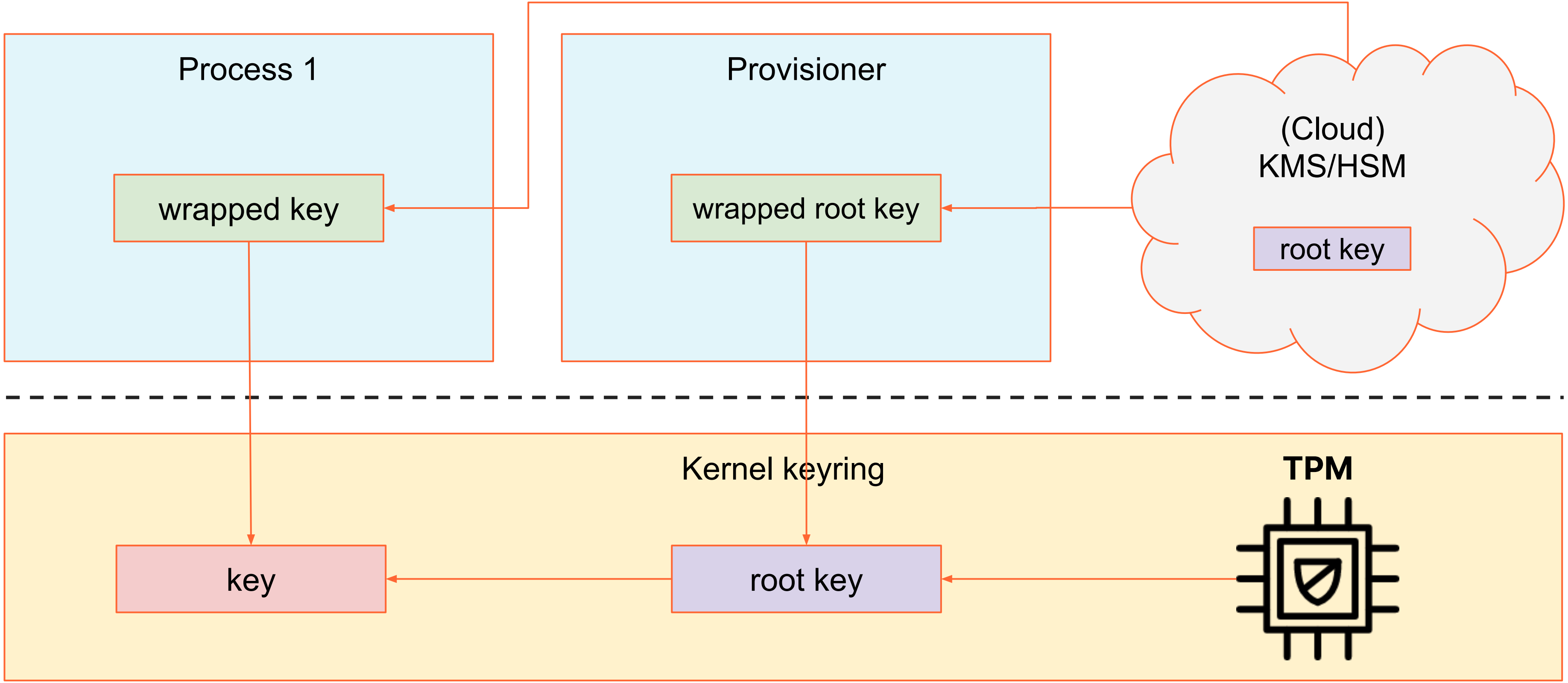
Combined schema



Combined schema



Combined schema



Combined schema problems

- Applications never see the plaintext cryptographic material in their process address space

Combined schema problems

- Applications never see the plaintext cryptographic material in their process address space
- But applications are responsible for contacting the centralised KMS/HSM to get their wrapped keys

Combined schema problems

- Applications never see the plaintext cryptographic material in their process address space
- But applications are responsible for contacting the centralised KMS/HSM to get their wrapped keys
  - need to know how to reach the centralised KMS/HSM
    - KMS/HSM URI endpoints in each application configuration
    - application code for client ↔ KMS/HSM communication protocol

Combined schema problems

- Applications never see the plaintext cryptographic material in their process address space
- But applications are responsible for contacting the centralised KMS/HSM to get their wrapped keys
  - need to know how to reach the centralised KMS/HSM
    - KMS/HSM URI endpoints in each application configuration
    - application code for client ↔ KMS/HSM communication protocol
  - little administrative control of the created Kernel key objects
    - invalid key permissions may even leak the key

Combined schema problems

- Applications never see the plaintext cryptographic material in their process address space
- But applications are responsible for contacting the centralised KMS/HSM to get their wrapped keys
  - need to know how to reach the centralised KMS/HSM
    - KMS/HSM URI endpoints in each application configuration
    - application code for client ↔ KMS/HSM communication protocol
  - little administrative control of the created Kernel key objects
    - invalid key permissions may even leak the key
  - KMS/HSM needs to somehow authenticate each requesting application

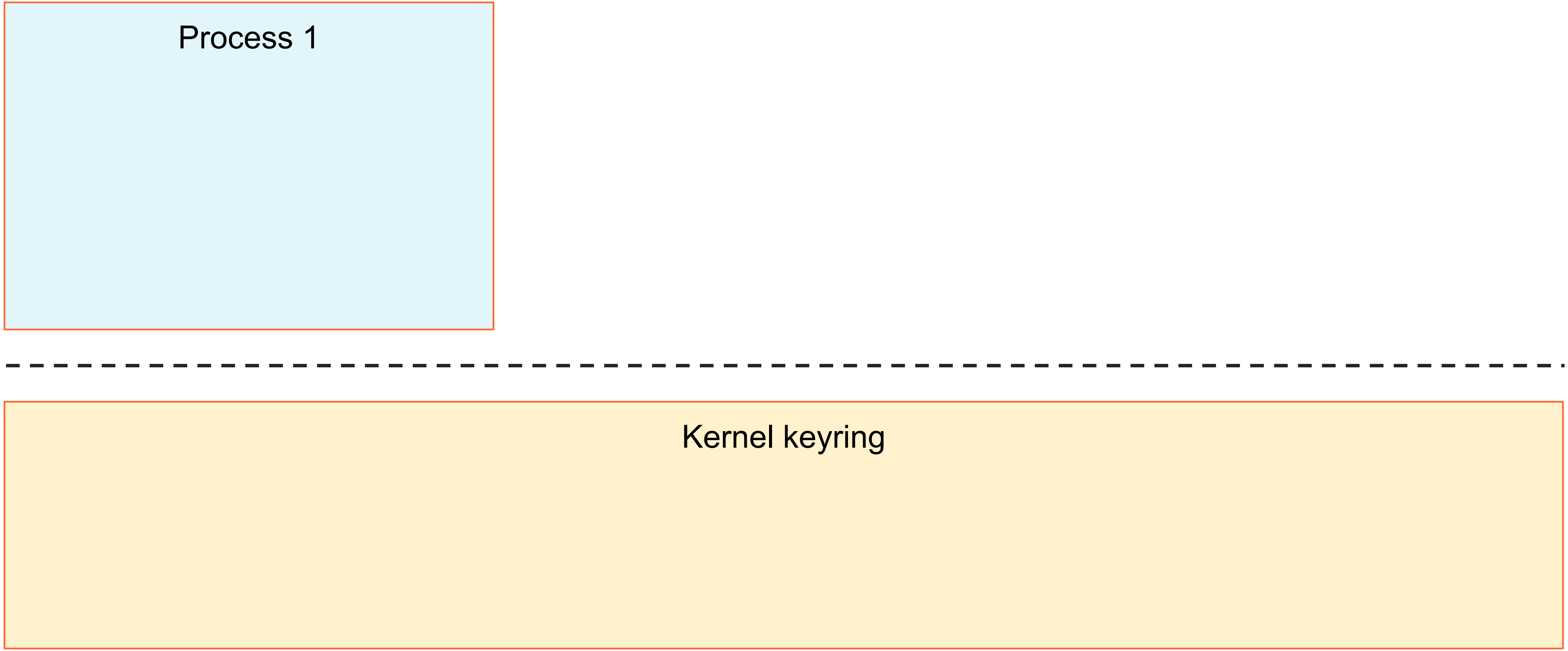
Linux Kernel key provisioning

- `add_key(2)`
  - adds the key to the specified keyring with the provided payload
  - payload is interpreted according to the key type
    - nothing for user/logon
    - private/public for asymmetric
    - wrapped for encrypted/trusted
  - https://man7.org/linux/man-pages/man2/add_key.2.html

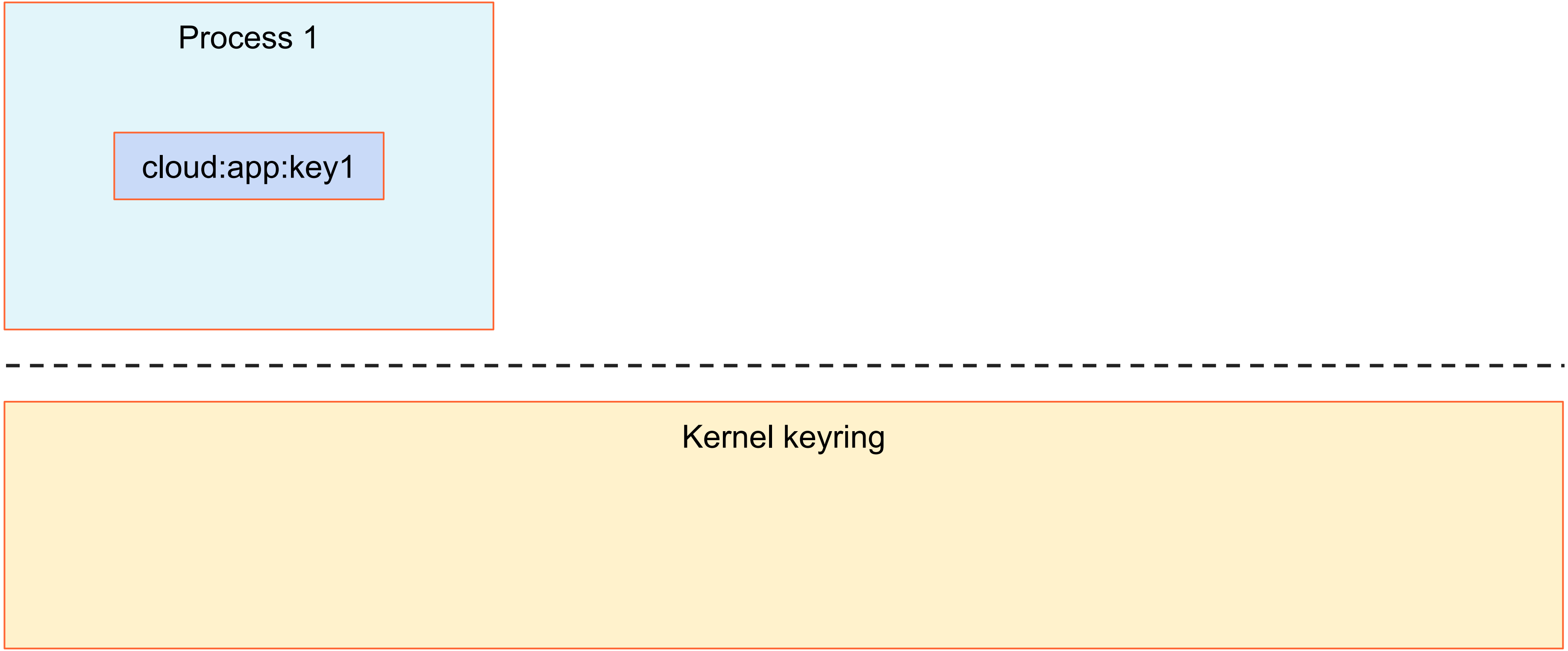
Linux Kernel key provisioning

- `add_key(2)`
  - adds the key to the specified keyring with the provided payload
  - payload is interpreted according to the key type
    - nothing for user/logon
    - private/public for asymmetric
    - wrapped for encrypted/trusted
  - https://man7.org/linux/man-pages/man2/add_key.2.html
- `request_key(2)`
  - a key is requested from the kernel based on a string id
    - the kernel is expected to provide the payload
  - if the kernel cannot satisfy the request, it calls a “helper” program
    - the helper program can hook into external KMS/HSM
    - the helper program can adjust key permissions
  - a more centralised and transparent API to add keys to the keyring
  - https://man7.org/linux/man-pages/man2/request_key.2.html

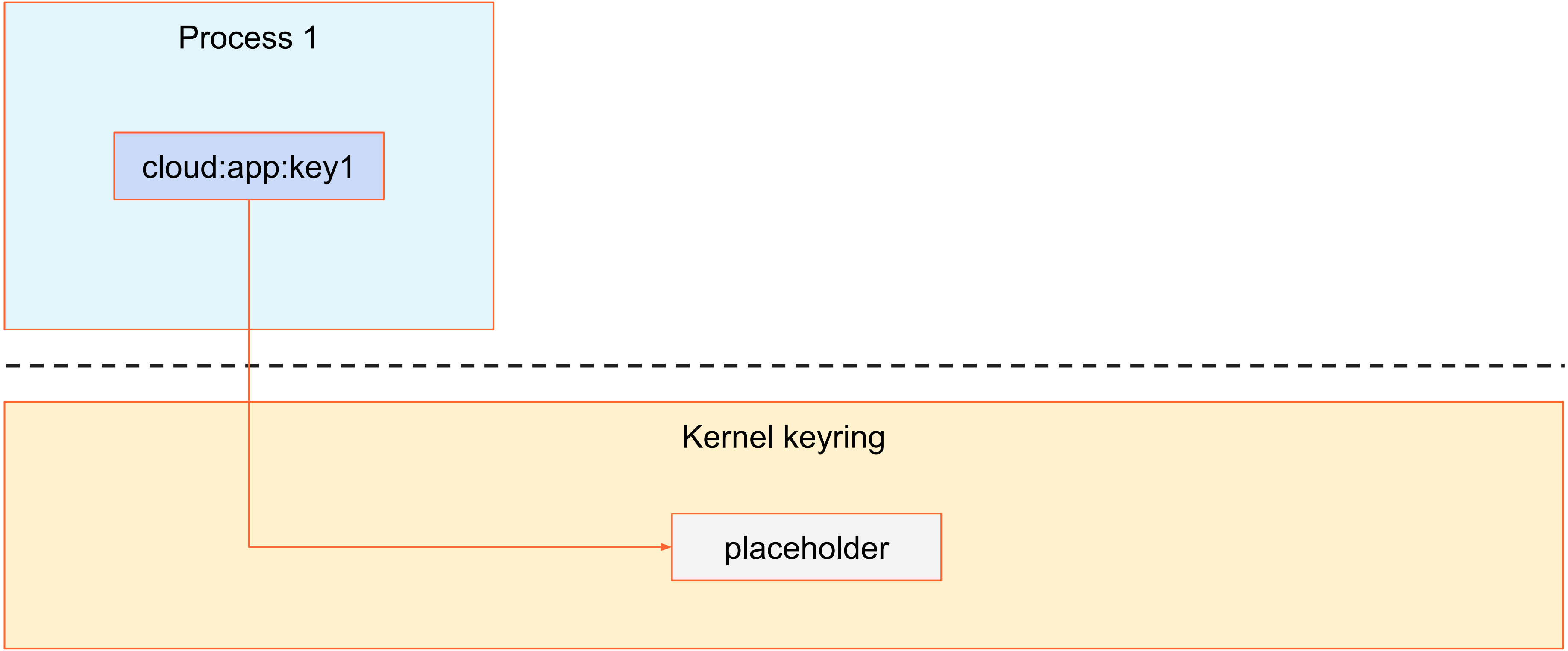
request_key(2) syscall



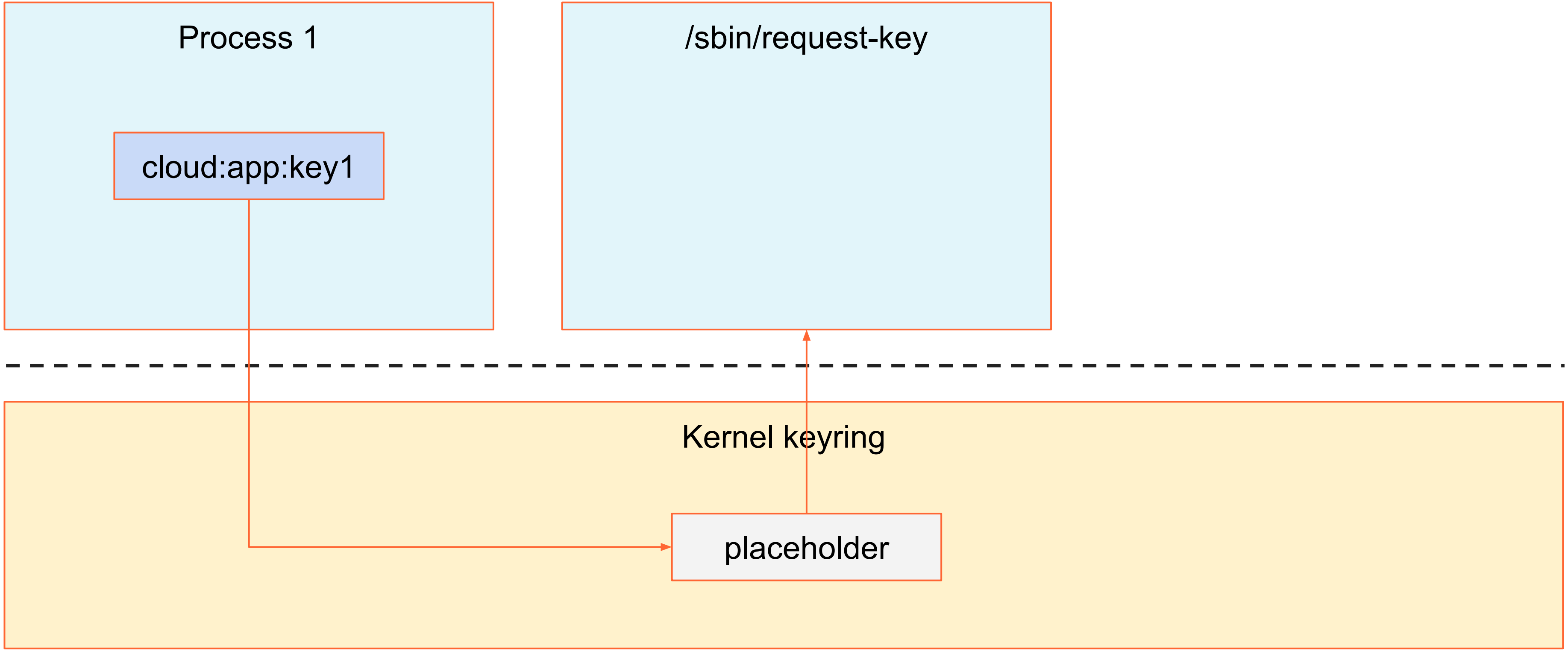
request_key(2) syscall



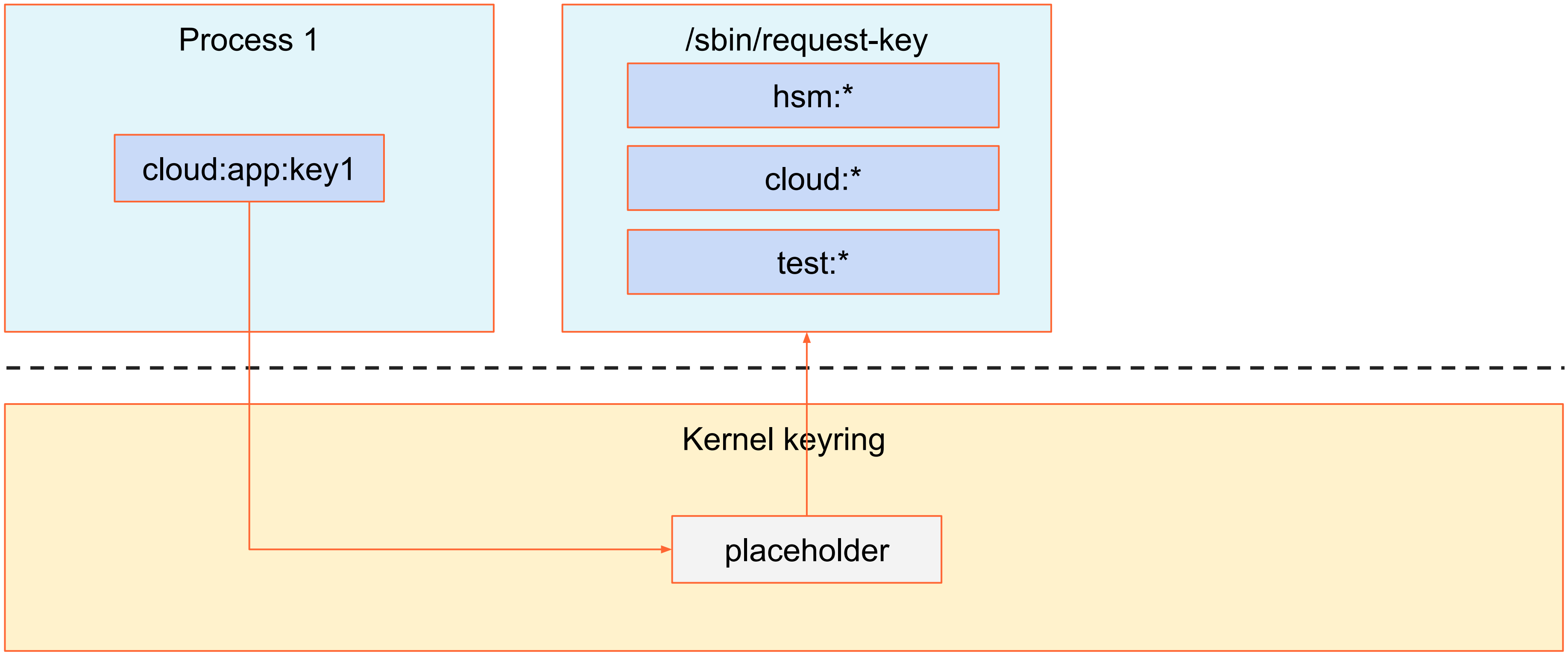
request_key(2) syscall



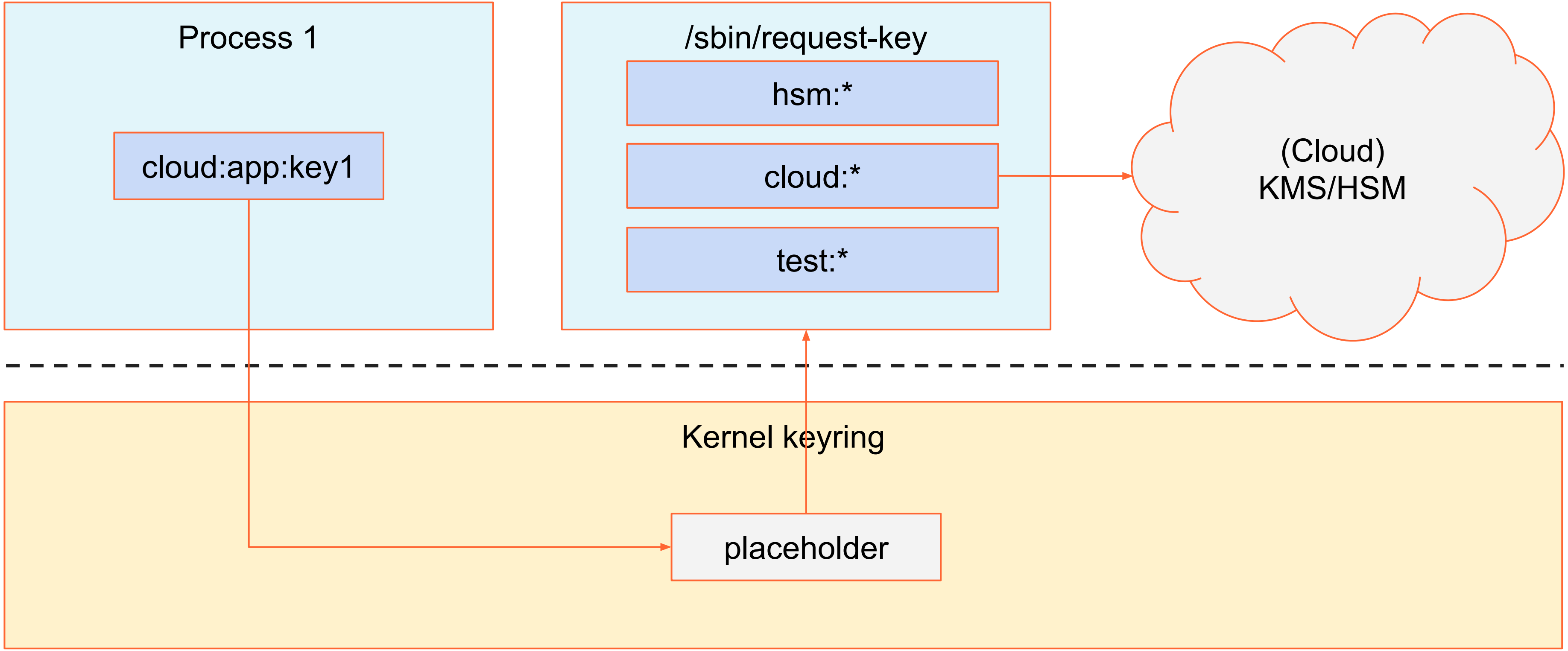
request_key(2) syscall



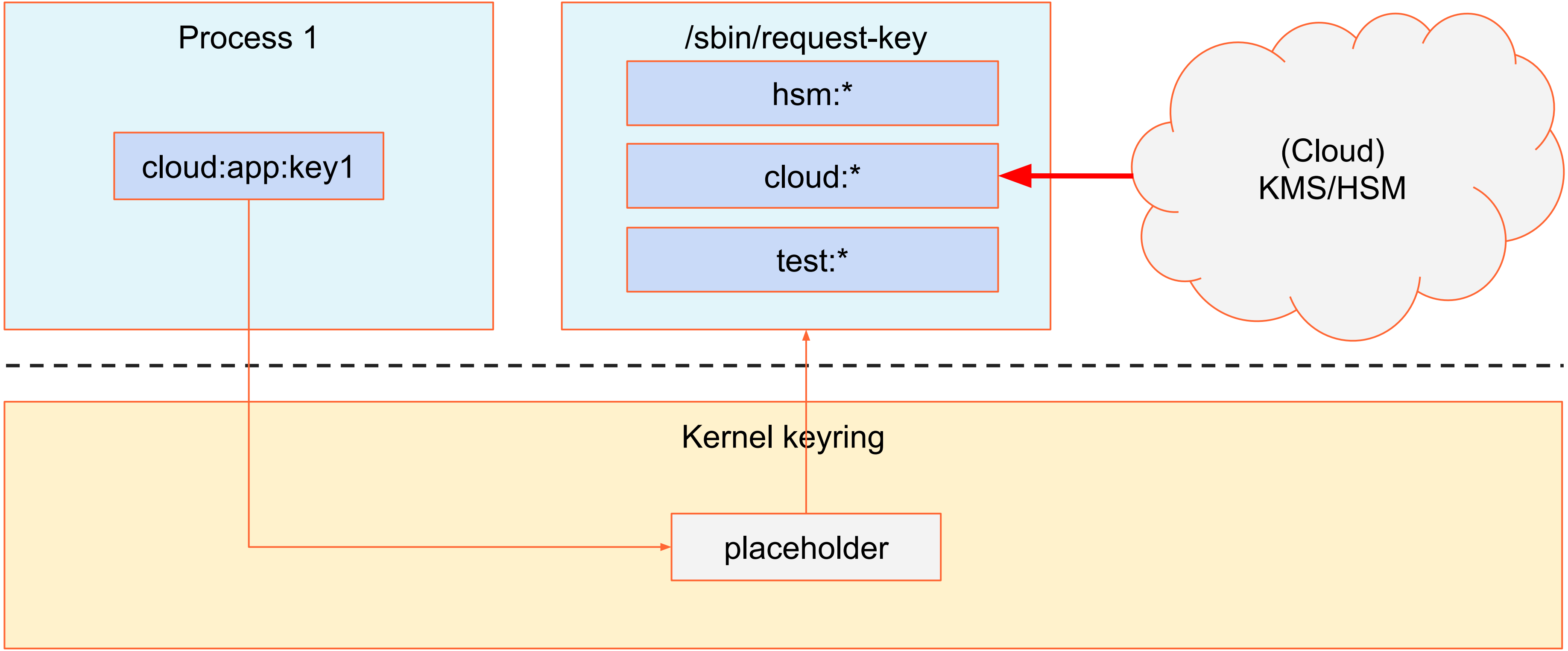
request_key(2) syscall



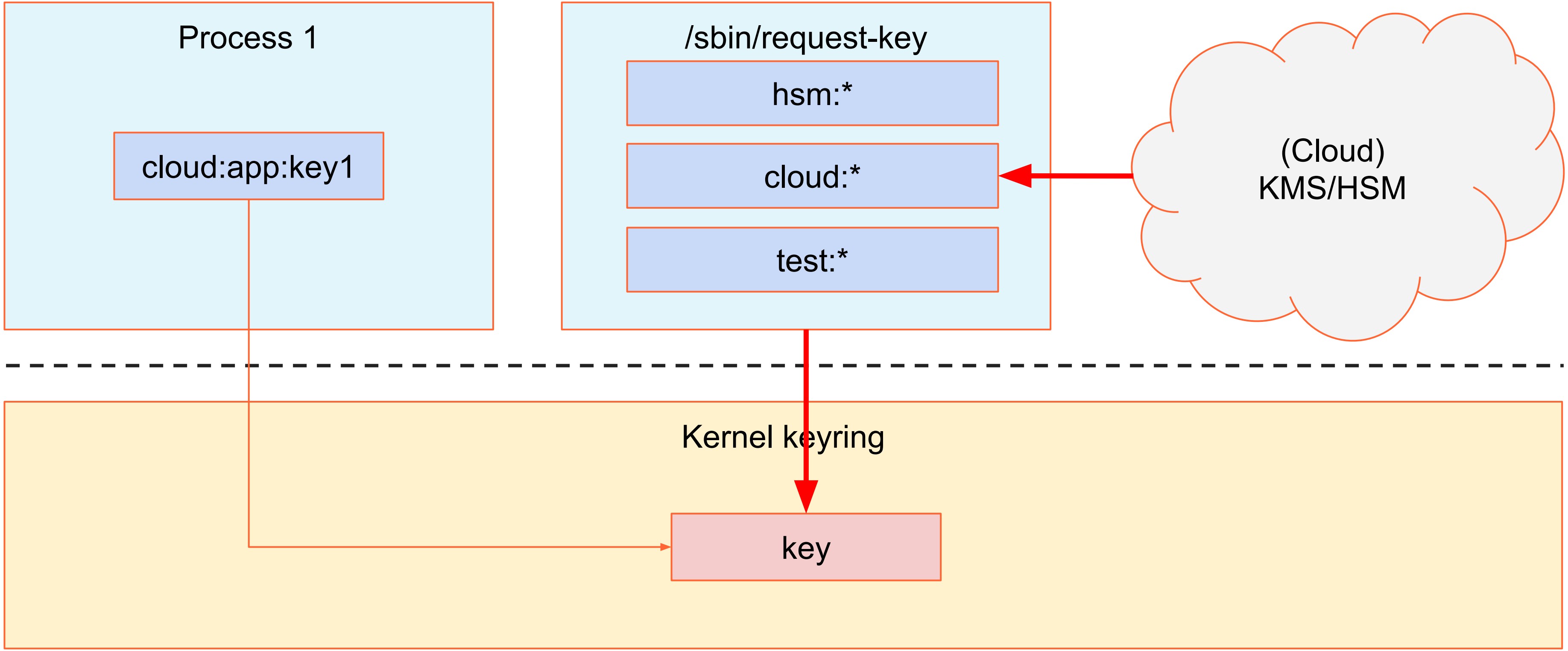
request_key(2) syscall



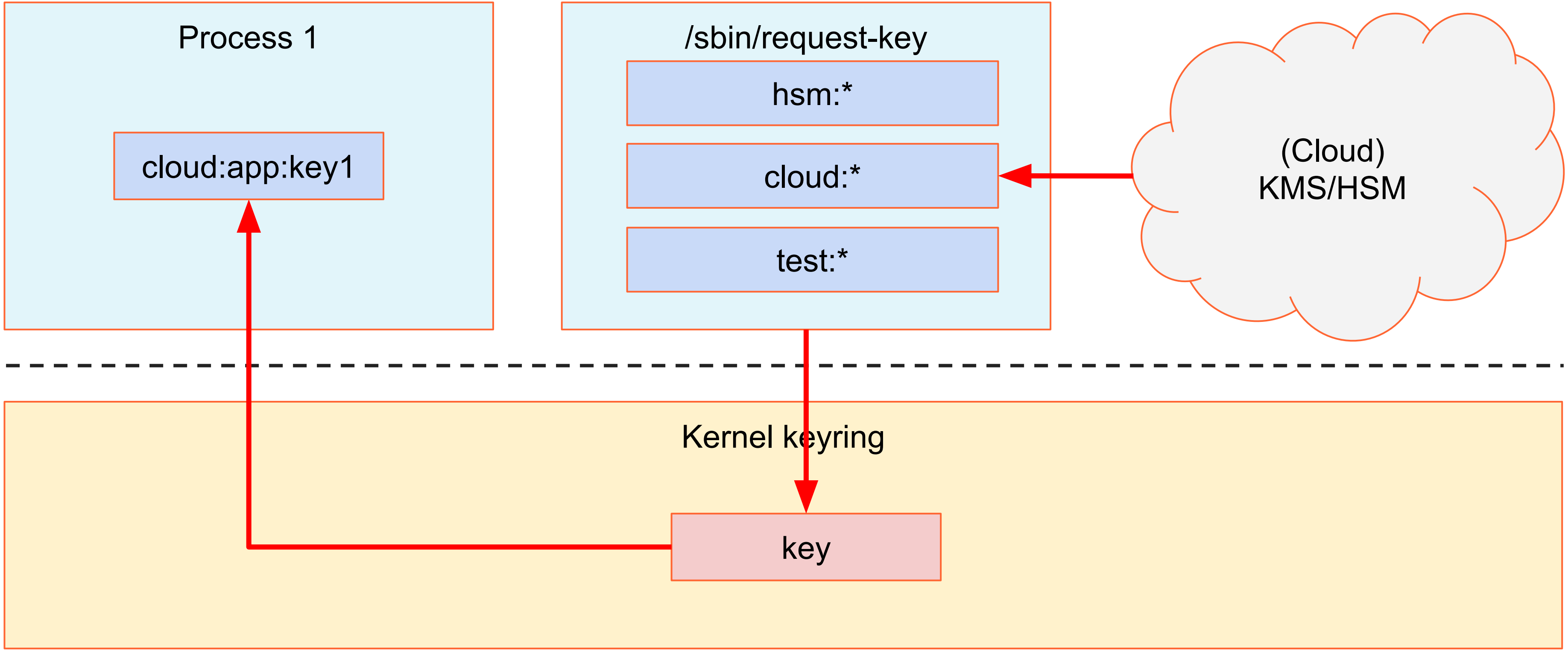
request_key(2) syscall



request_key(2) syscall



request_key(2) syscall



request_key(2) advantages

- A single centralised OS API to request keys for applications
  - no KMS/HSMs connection strings, URIs etc in the config
  - just a “free-form” string id
  - fully decoupled from key storage backends

request_key(2) advantages

- A single centralised OS API to request keys for applications
  - no KMS/HSMs connection strings, URIs etc in the config
  - just a “free-form” string id
  - fully decoupled from key storage backends
- A more secure way to instantiate keys in the Kernel
  - only the Kernel created process can instantiate the requested key
  - callout process can perform additional security checks
    - ex. requestor uid, gid, pid, executable path, package name etc.
  - can support multiple key storage backends
    - backends can be swapped transparently to the applications
  - only the callout process needs to be authenticated on the backend
  - backend connectors can be written in any language

CLOUDFLARE

Minimizing cryptographic material exposure

With `request_key(2)` support the key management and distribution becomes a core service of the operating system

Links

- https://www.kernel.org/doc/html/latest/security/keys/core.html
- https://www.kernel.org/doc/html/latest/security/keys/trusted-encrypted.html
- https://man7.org/linux/man-pages/man7/keyrings.7.html
- https://man7.org/linux/man-pages/man7/asymmetric.7.html
- https://man7.org/linux/man-pages/man1/keyctl.1.html
- https://blog.cloudflare.com/the-linux-kernel-key-retention-service-and-why-you-should-use-it-in-your-next-application/

Thank you!

Questions?