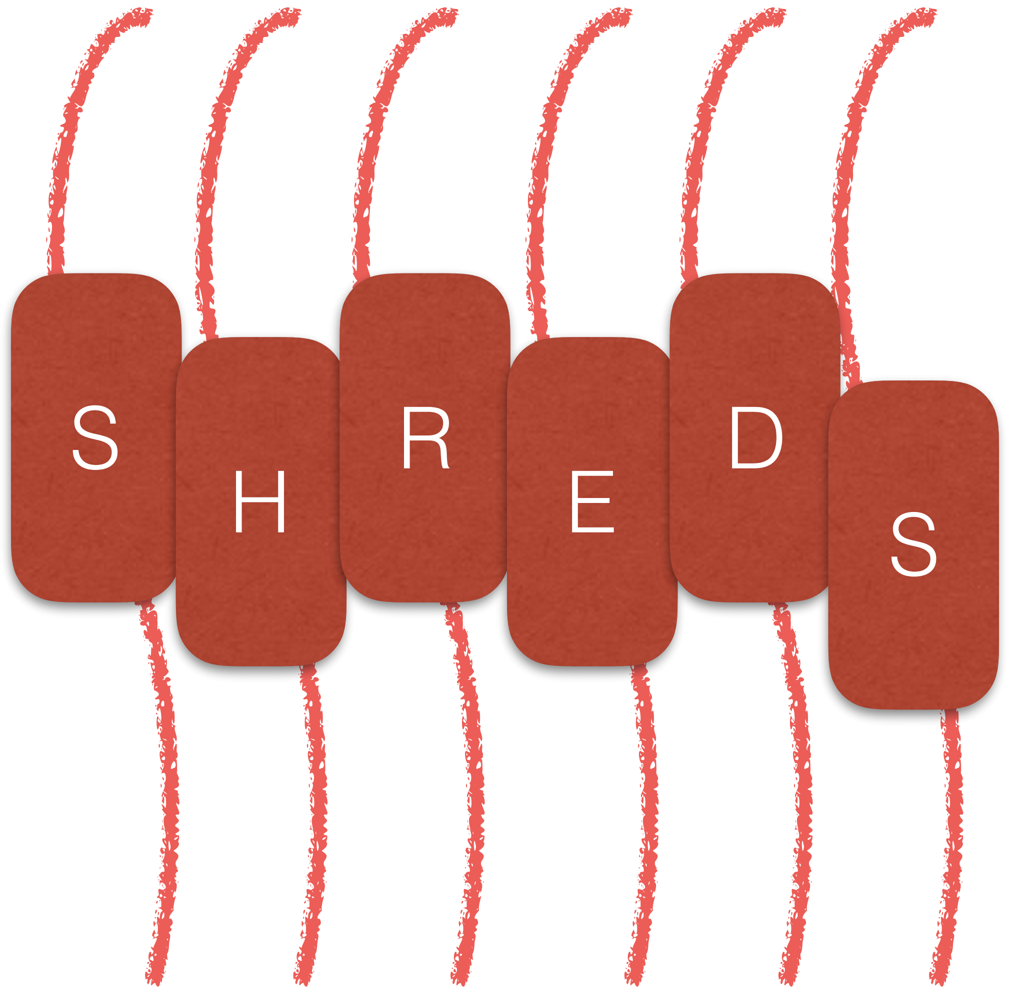
Stony Brook University



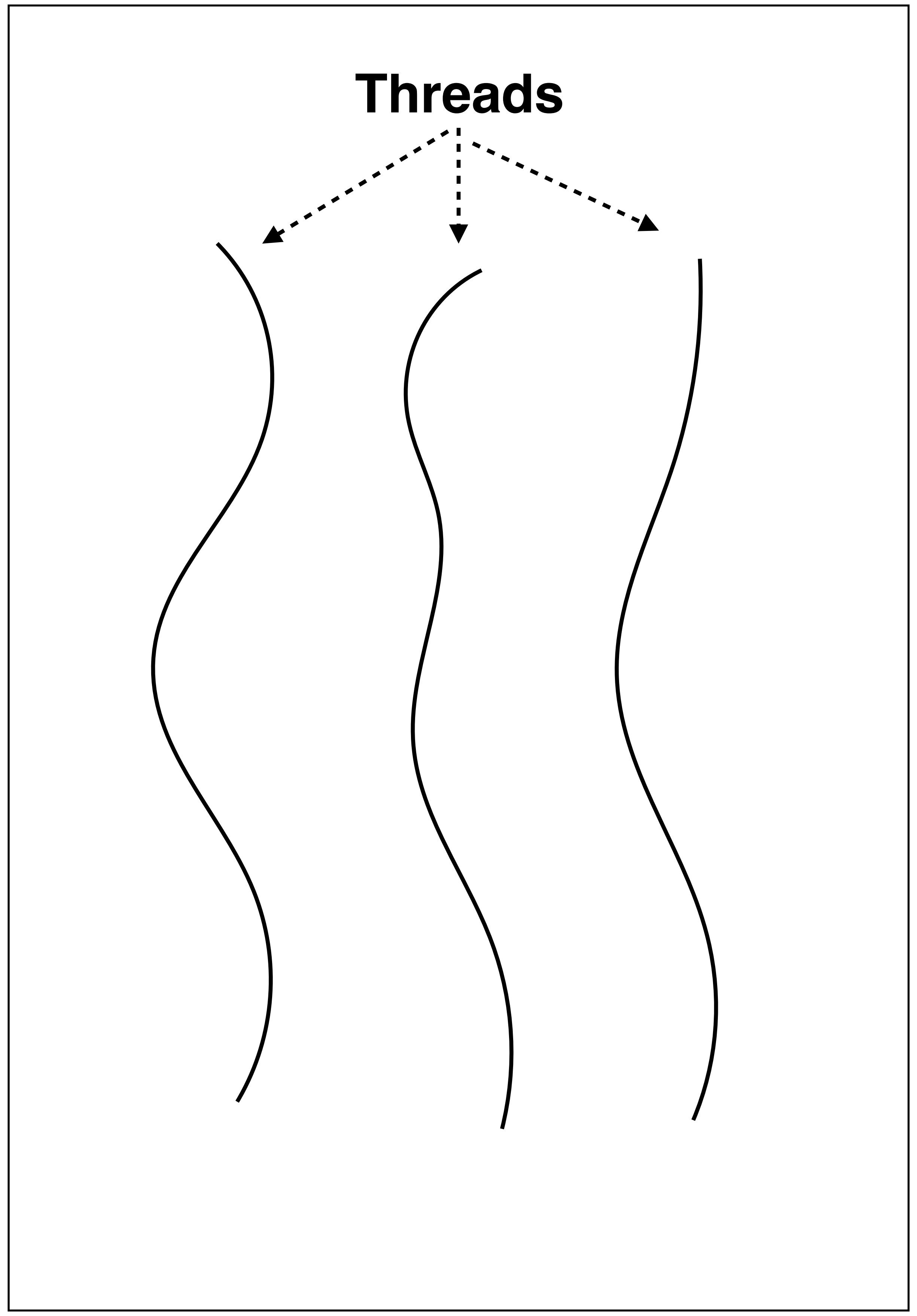
Shreds:
Fine-grained Execution Units with Private Memory

Yaohui Chen, Sebassujeen Reymondjohnson, Zhichuang Sun, Long Lu

RiS3 Lab / Computer Science / Stony Brook University

Execution Units

A Process



- **Traditional Execution Units**
  - Processes
    - Separate address spaces
  - Threads
    - Sharing one address space

In-process Memory Abuses

- **Definition:**

  Malicious or compromised components try to steal data or execute code of other components running in the same process.

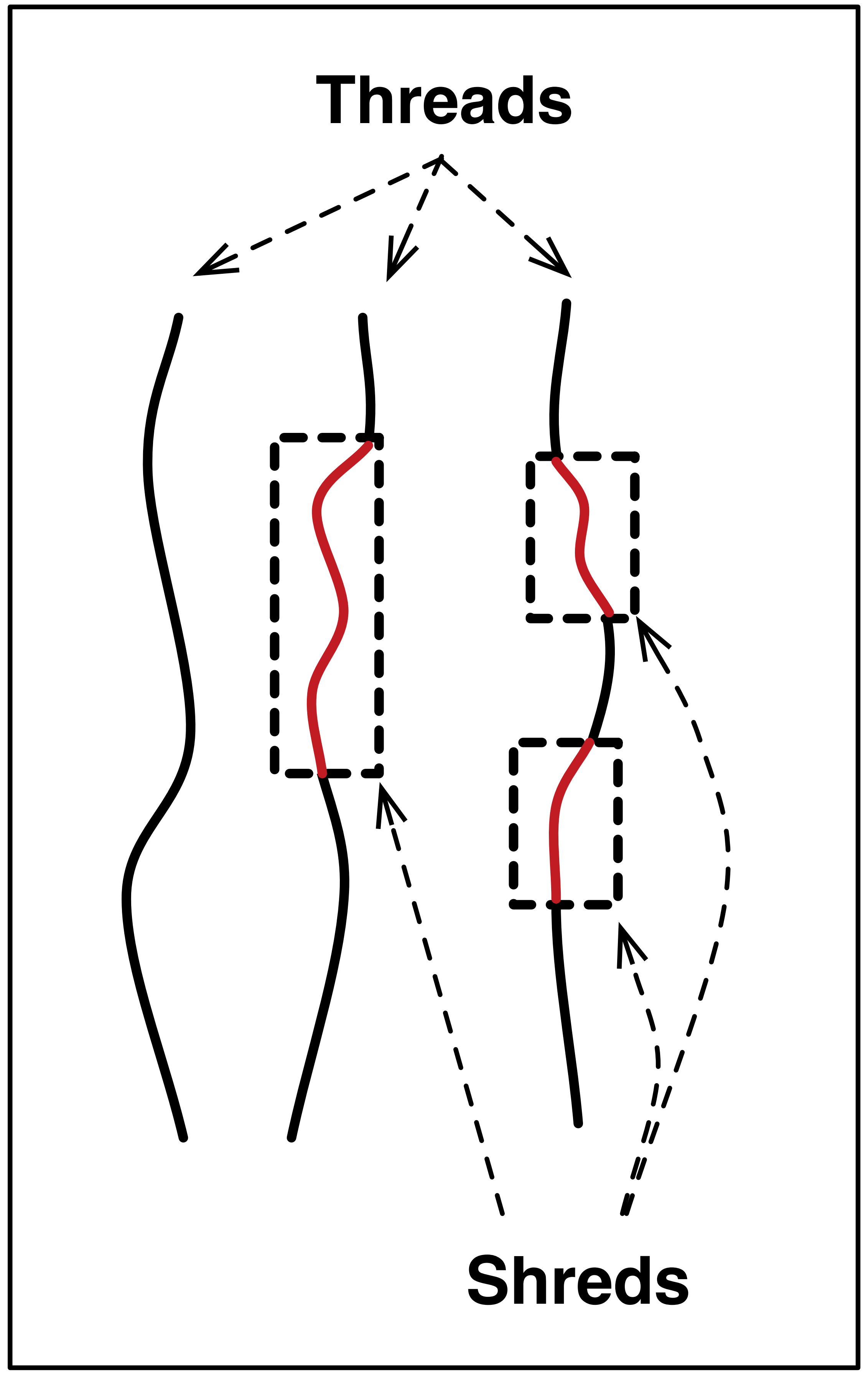
- **Two examples**
  - **Stealing secret data**
    - The Heartbleed bug
  - **Executing private code**
    - Private APIs in iOS Apps

Potential Mitigations of in-Process Abuse

Techniques	Why unsuitable
Process-level isolation *(OpenSSH, Chrome)*	• IPC is expensive • Adoption effort
Software fault isolation-like techniques *(Native Client)*	• Require instrumenting untrusted code • Ineffective on dynamic or external code
Hardware-assisted techniques *(SGX, Trustzone)*	• Overly restrictive execution environment • Semantic gap

Introducing Shred

A process



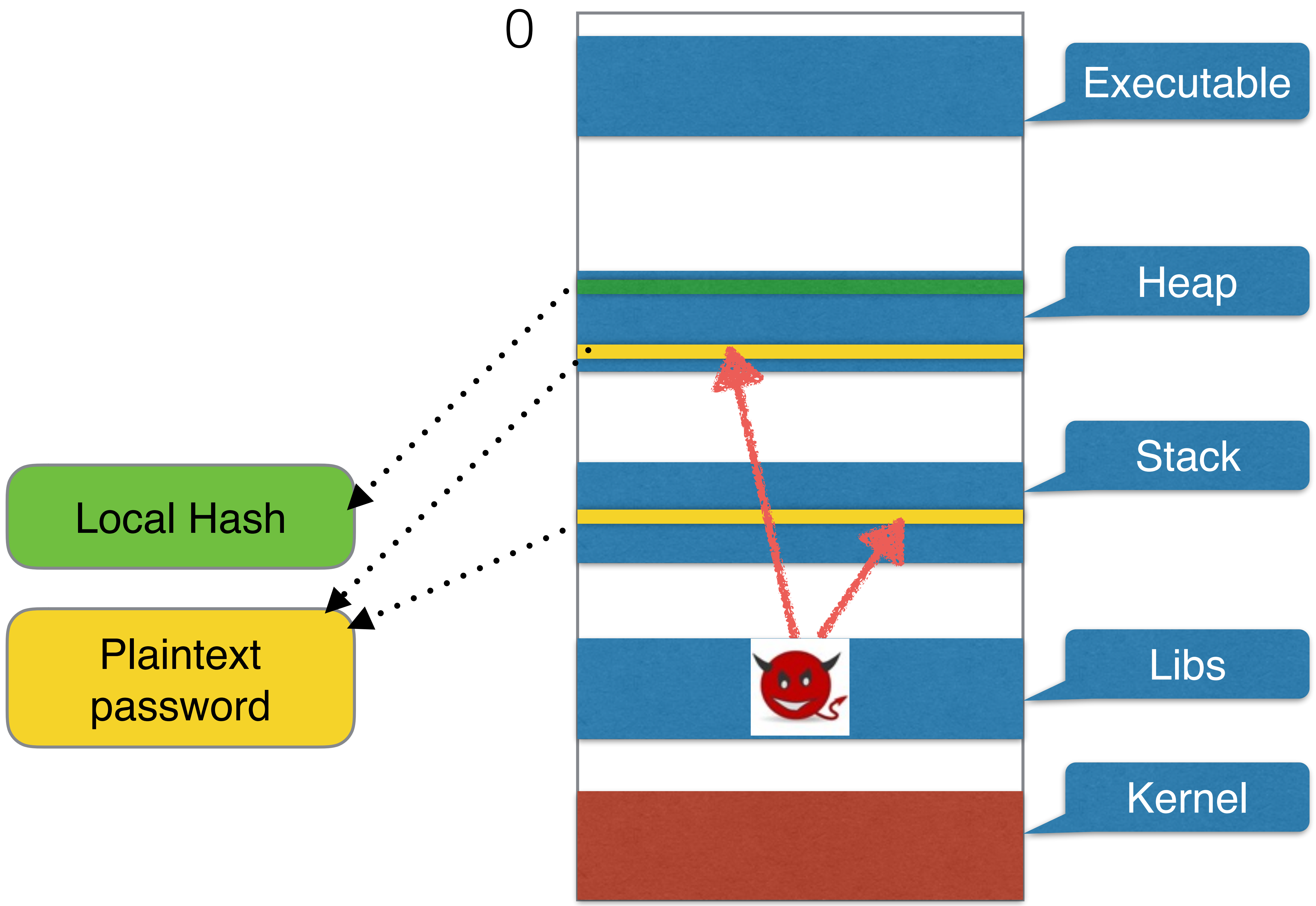
- **Shred**
  - **Arbitrarily** scoped segment of a thread execution
- **S-pool**
  - The private memory pool for each shred
- **Shred APIs & OS-level supports**

Threat Model

- Trusted OS
- Untrusted component

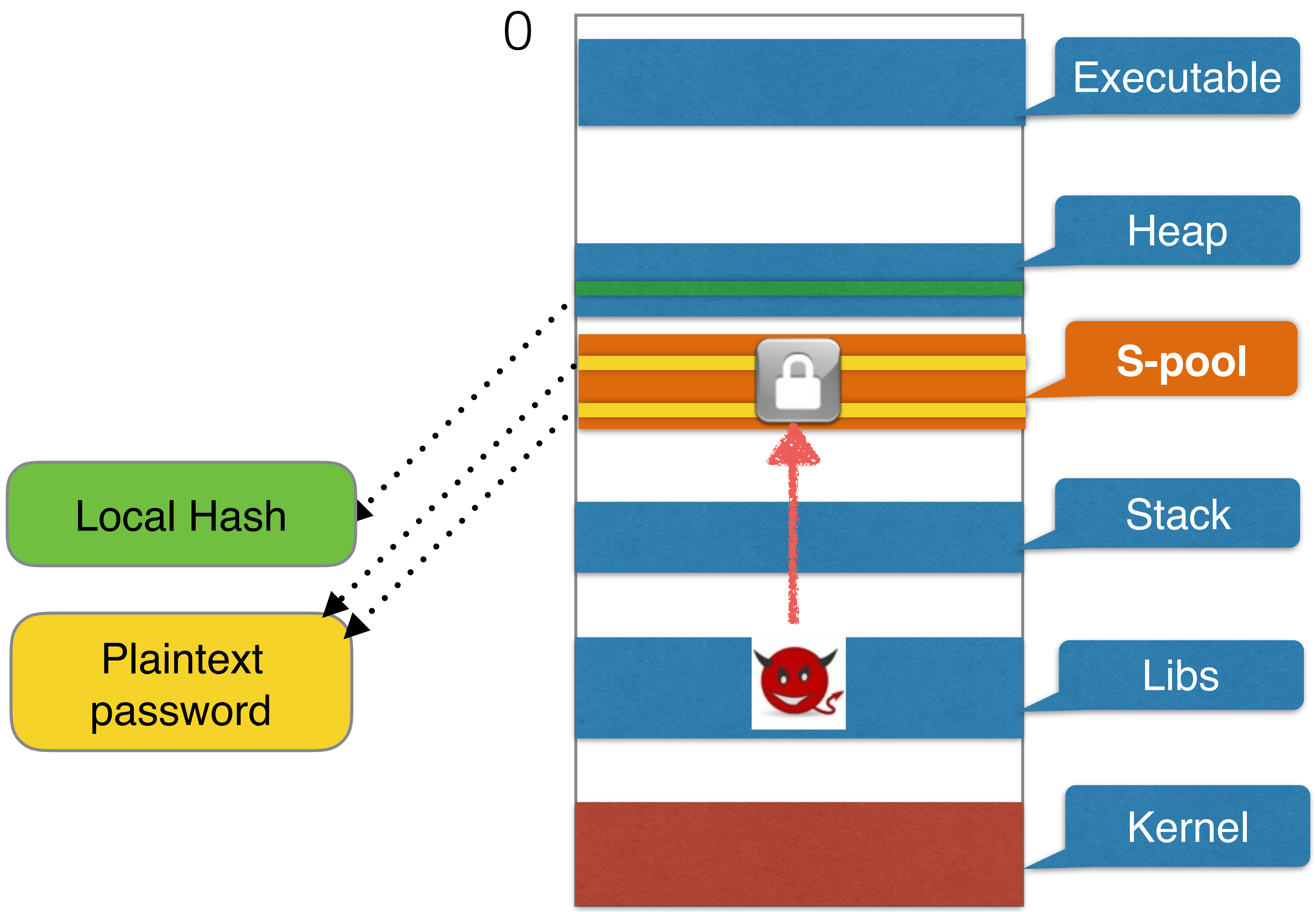
Example Use Case

- Password authentication on web server(**w/o shred**)



Example Use Case cont.

- Password authentication on web server(**w/ shred**)



Shred APIs

- err_t **shred_enter**(int *pool_desc*);
  - Start a shred execution on the current thread
  - Unlock s-pool
- err_t **shred_exit**();
  - Terminate a shred execution
  - lock down the s-pool

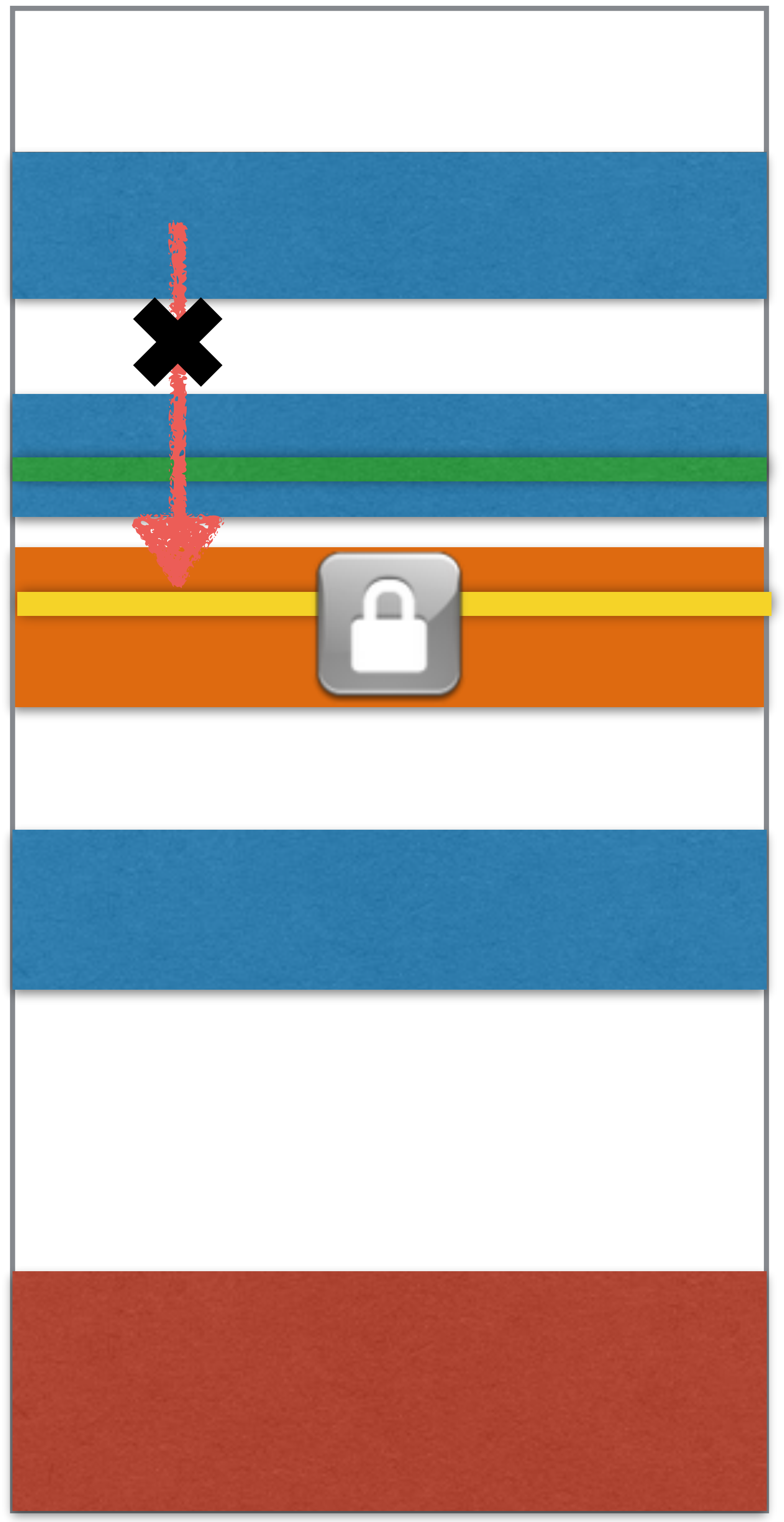
Shred creation APIs

- void * **spool_alloc**(size_t *size*);
  - Allocate memory inside S-pool
- err_t **spool_free**(void **ptr*);
  - Free memory inside S-pool

S-pool allocation APIs

Code Example—Lighttpd

```
int http_request_parse(server *srv, connection *con) {
   ...
   char *cur;  /* to receive password */
+  if (strncmp(cur, auth_str, auth_str_len)==0){
+    shred_enter(AUTH_PASSWD_POOL);
+    /* receive and save password */
+    data_string *ds = s_ds_init();
+    int pw_len = get_passwd_length(cur);
+    cur += auth_str_len + 1;
+    buffer_copy_string_len(ds->key, auth_str, auth_str_len);
+    buffer_copy_string_len(ds->value, cur, pw_len);
+    cur += pw_len;
+    shred_exit();
+  }
   ...
}
```



Listing 1: *lighttpd/src/request.c*

Code Example cont.

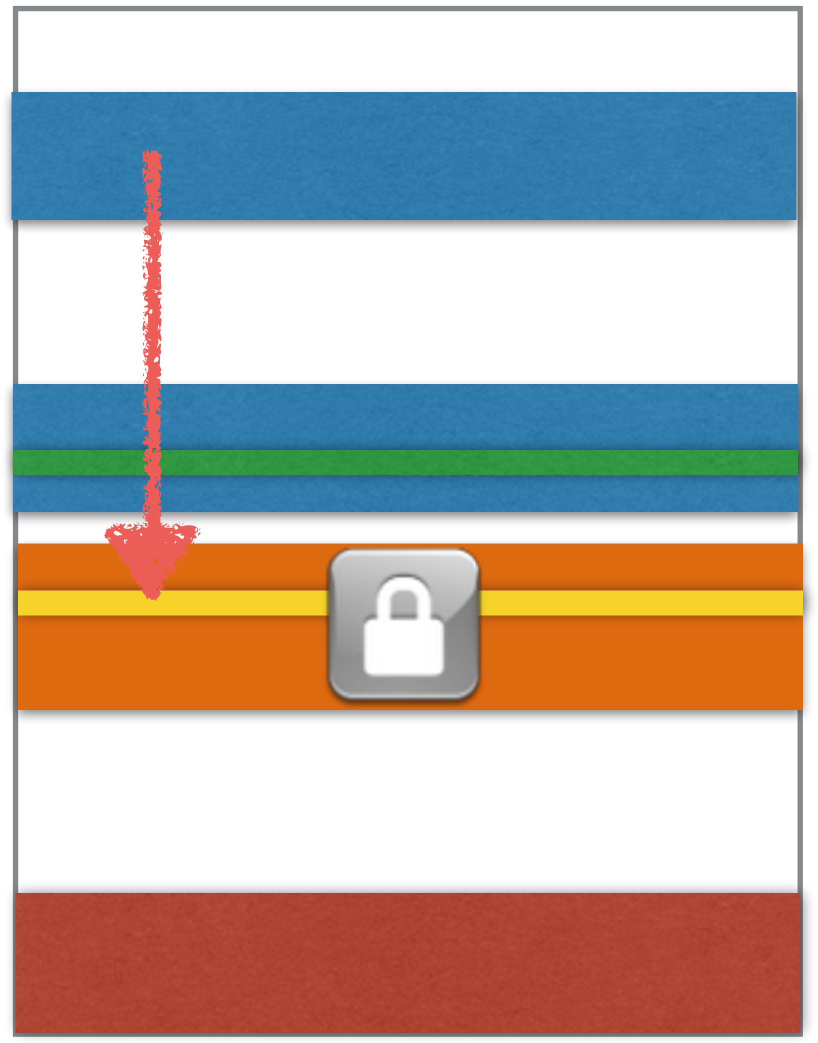
```
/* called inside a shred */
data_string *s_ds_init(void) {
    data_string *ds;
+   ds = spool_alloc(sizeof(*ds));
   ...
   return ds;
 }


/* called inside a shred */
void s_ds_free(data_string *ds) {
+    ...
+   spool_free(ds->key);
+    ...
    return;
}
```

S-pool allocation APIs wrapper

Listing 2: *lighttpd/src/data_string.c*

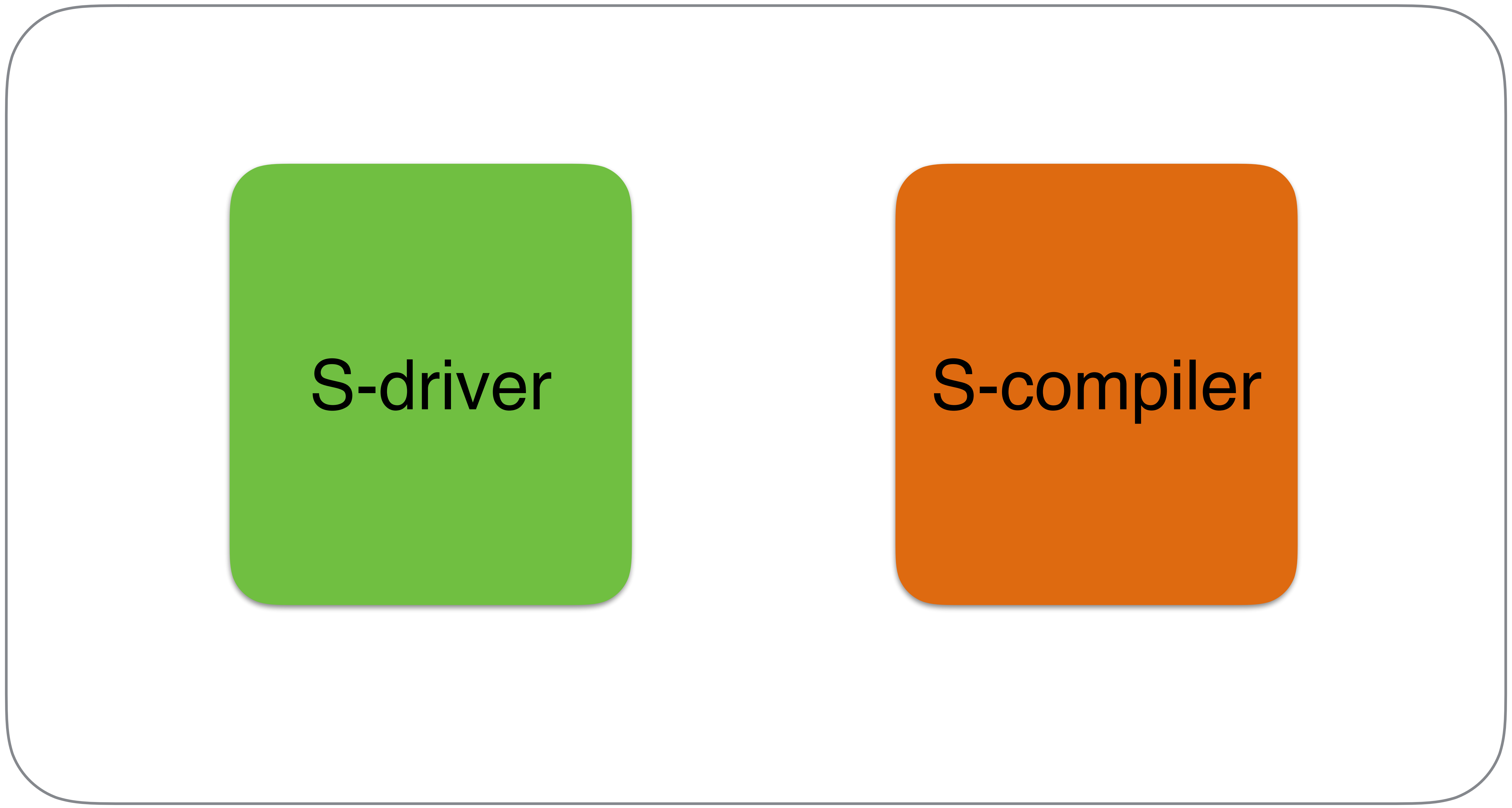
```
...
/* inside HTTP auth module */
+   shred_enter(AUTH_PASSWD_POOL);
/* ds points passwd obj in spool */
  http_authorization = ds->value->ptr;
/*hash passwd and compare with local copy*/
+   s_ds_free(ds);
+   shred_exit();
...
```



Listing 3: *lighttpd/src/mod_auth.c*

System overview

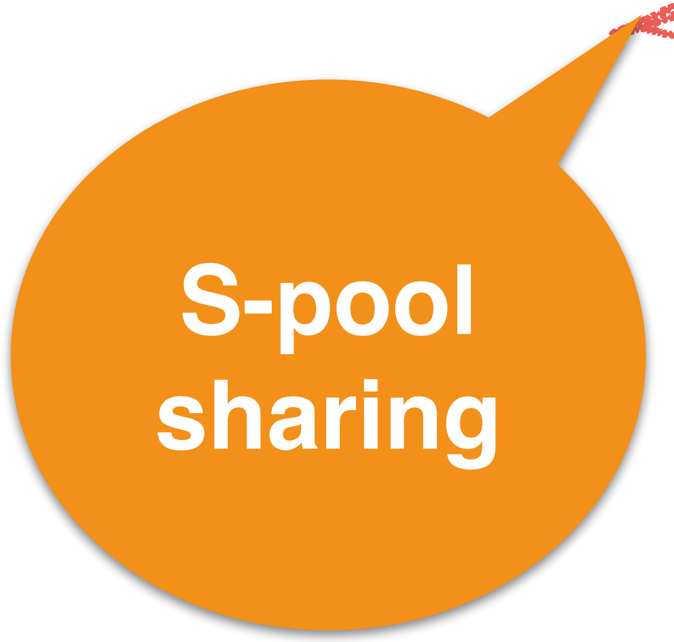
- **Two major components**

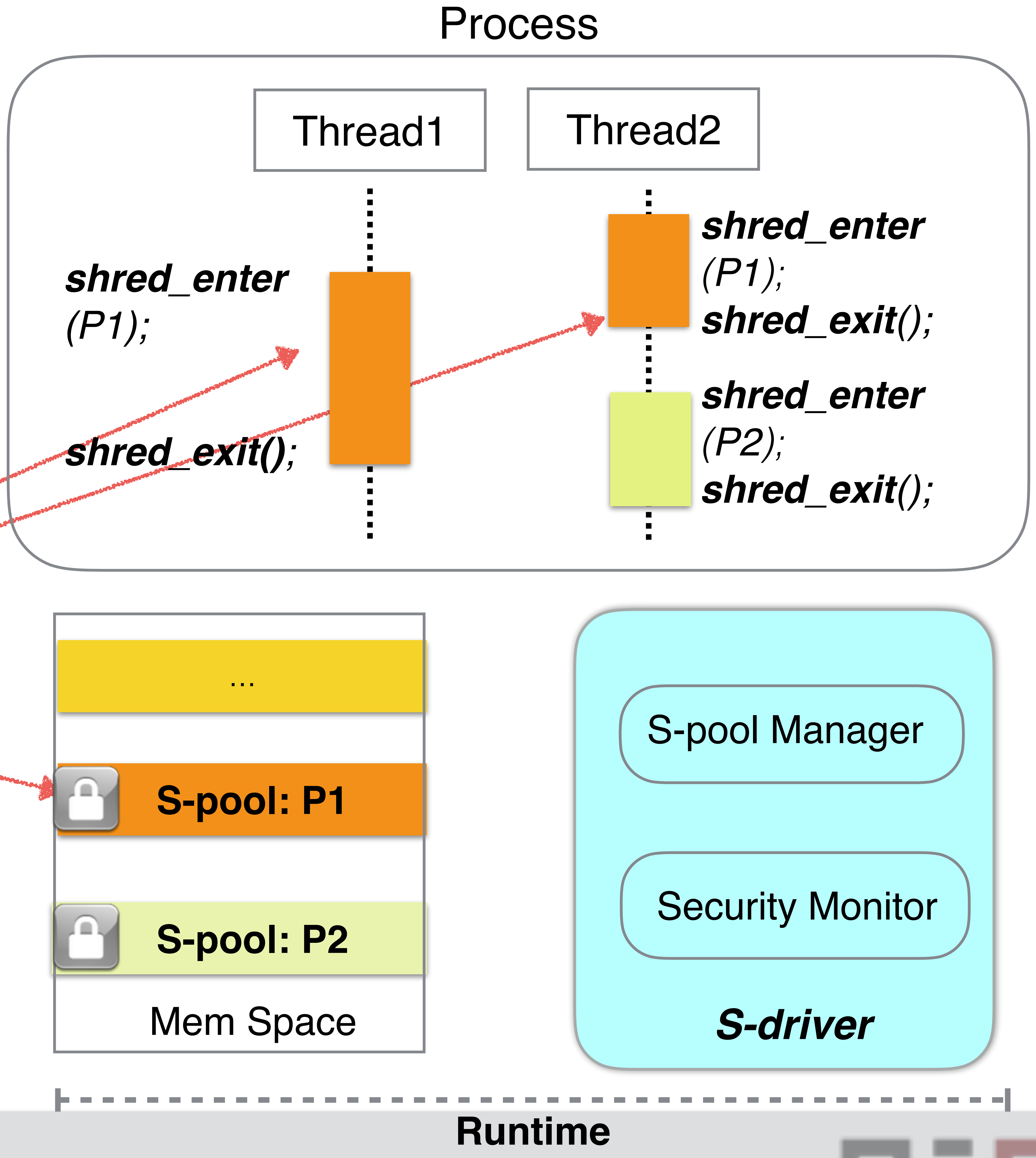


System Component: S-driver

S-driver

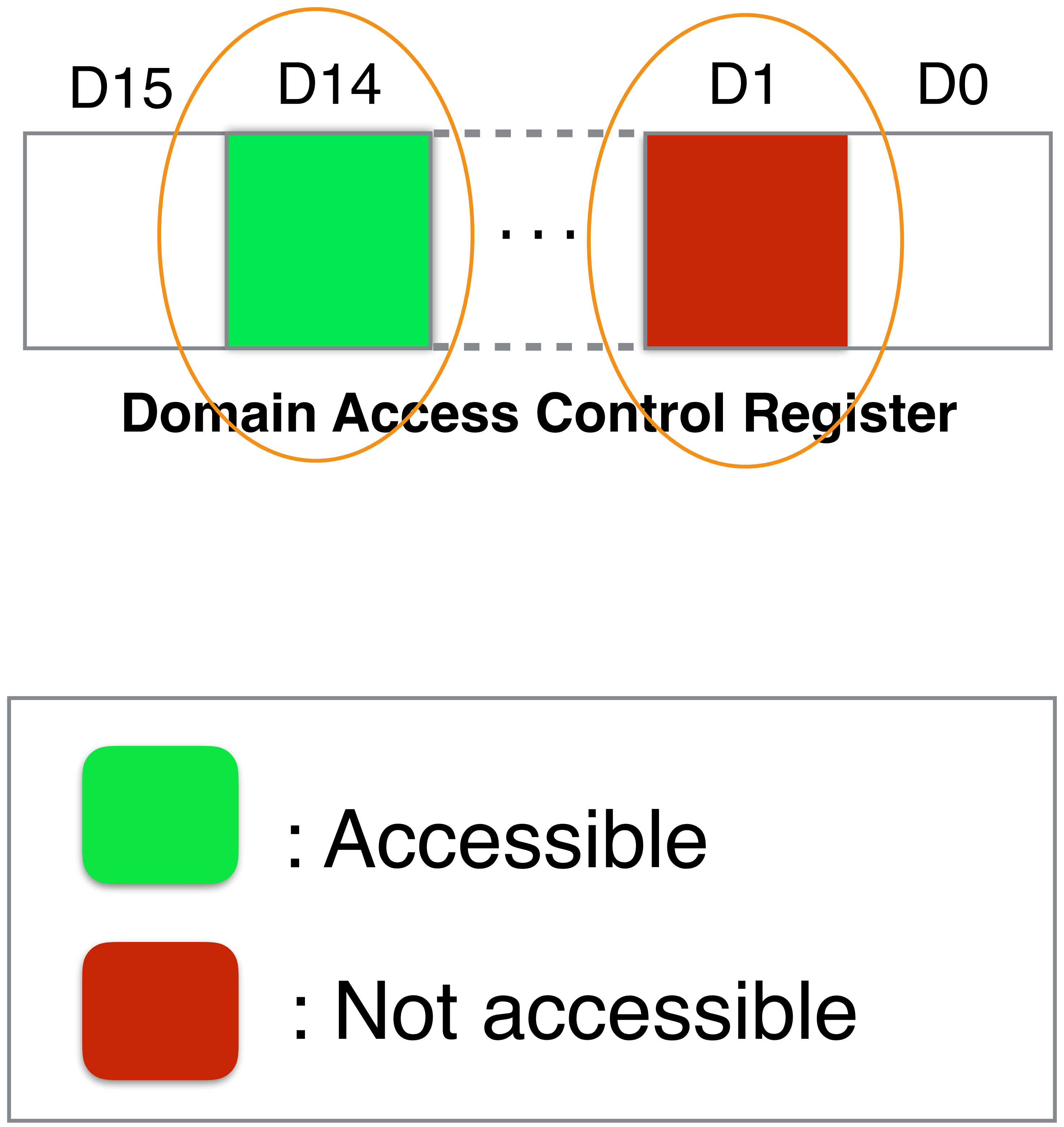
- Entry/exit of shreds
- S-pool (de)allocations
- Controls the access to S-pools

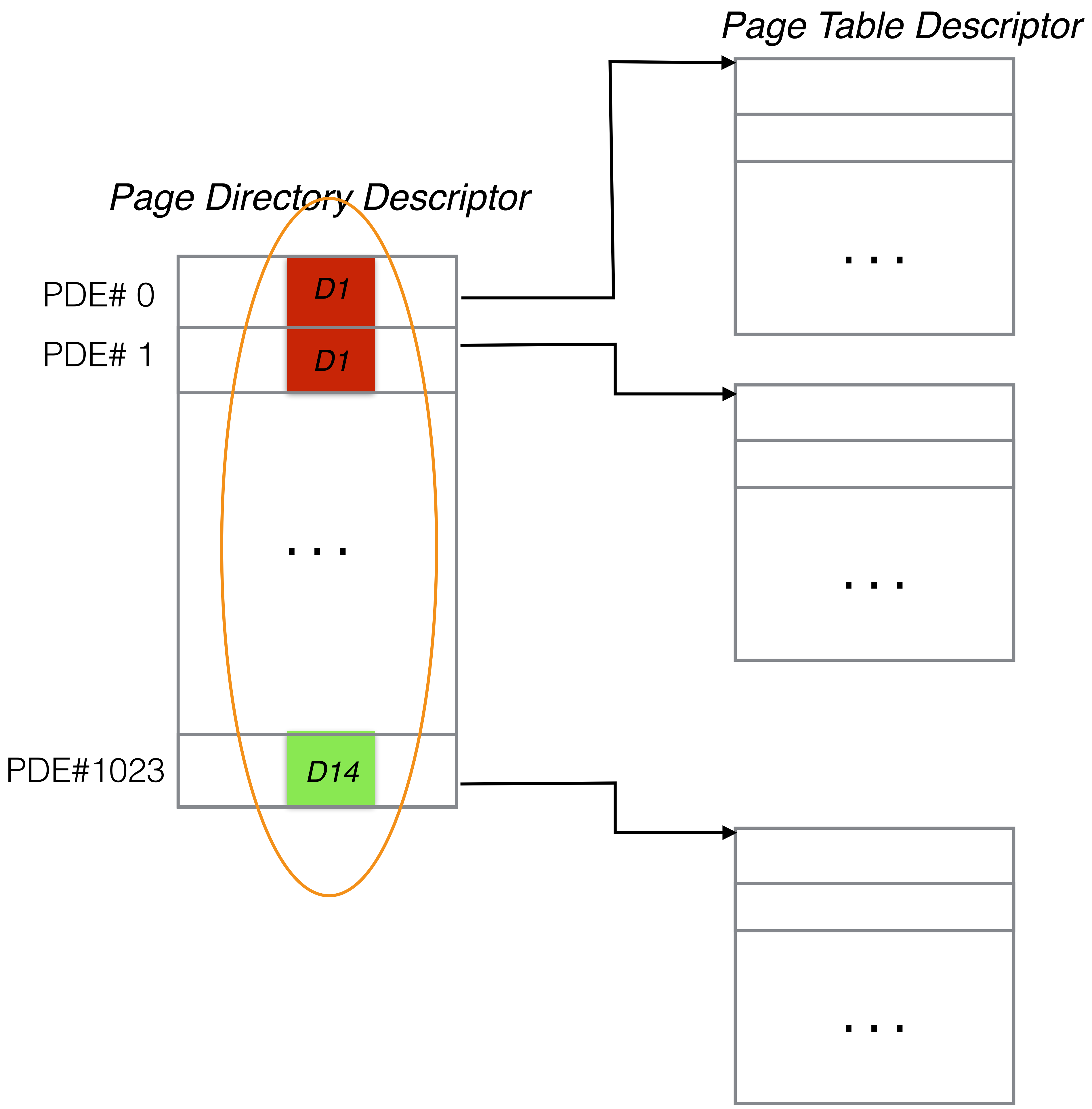




How S-pool is Built

- **Intel: Memory protection keys — The building block**





Challenges & Solutions

- ARM Memory Dom

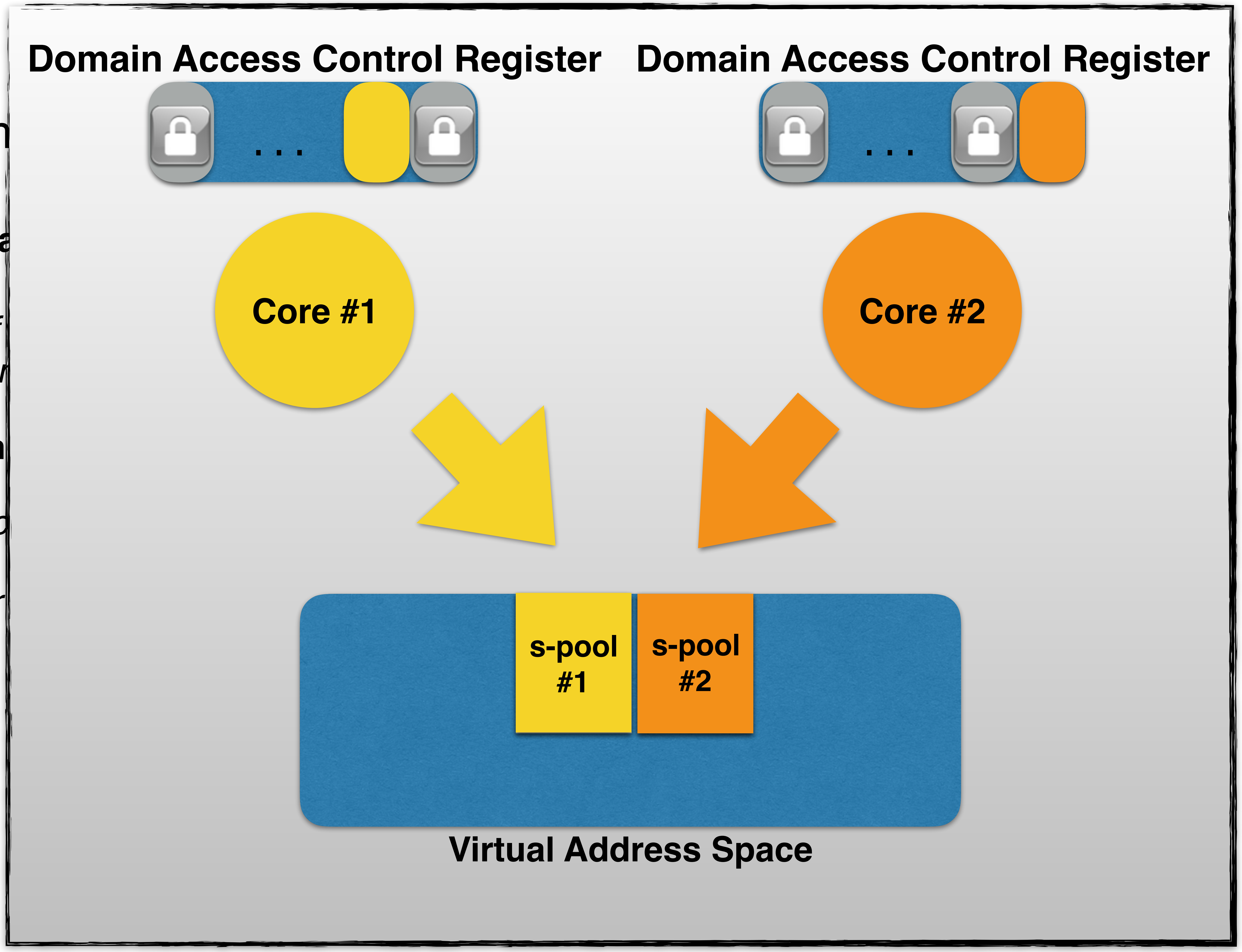
1)The granularity of the a

✓ *Create the notion of*
and use S-driver to r

2) Limited Domains: On

✓ *Statically bind an ad*

✓ *Reuse a domain for*



S-pool Managements

S-driver will,

- Lock s-pool when,
  - Shred exits
  - Context-switch Out
  - Asynchronous events: signal handling, etc

- Unlock s-pool when,
  - Shred enters
  - Context-switch in
  - Resuming from asynchronous events

Moving the Domain Adjustments Off the Critical Path

- Changing PDE is relatively cumbersome
  - Page table walking
  - TLB invalidation

- TWO knobs to control the accessibility of S-pool
  - Domain of the corresponding page table entry
  - Value of corresponding DACR entry

- Changing DACR value is much faster, only one instruction
  - ```
    MCR p15, 0, <Rd>, c3, c0, 0 ; Write DACR
    ```
  - Develop the **domain fault handler** to handle domain fault ***lazily***
    - Detecting attacks
    - Recover from legitimate domain faults
    ```

Runtime Protections

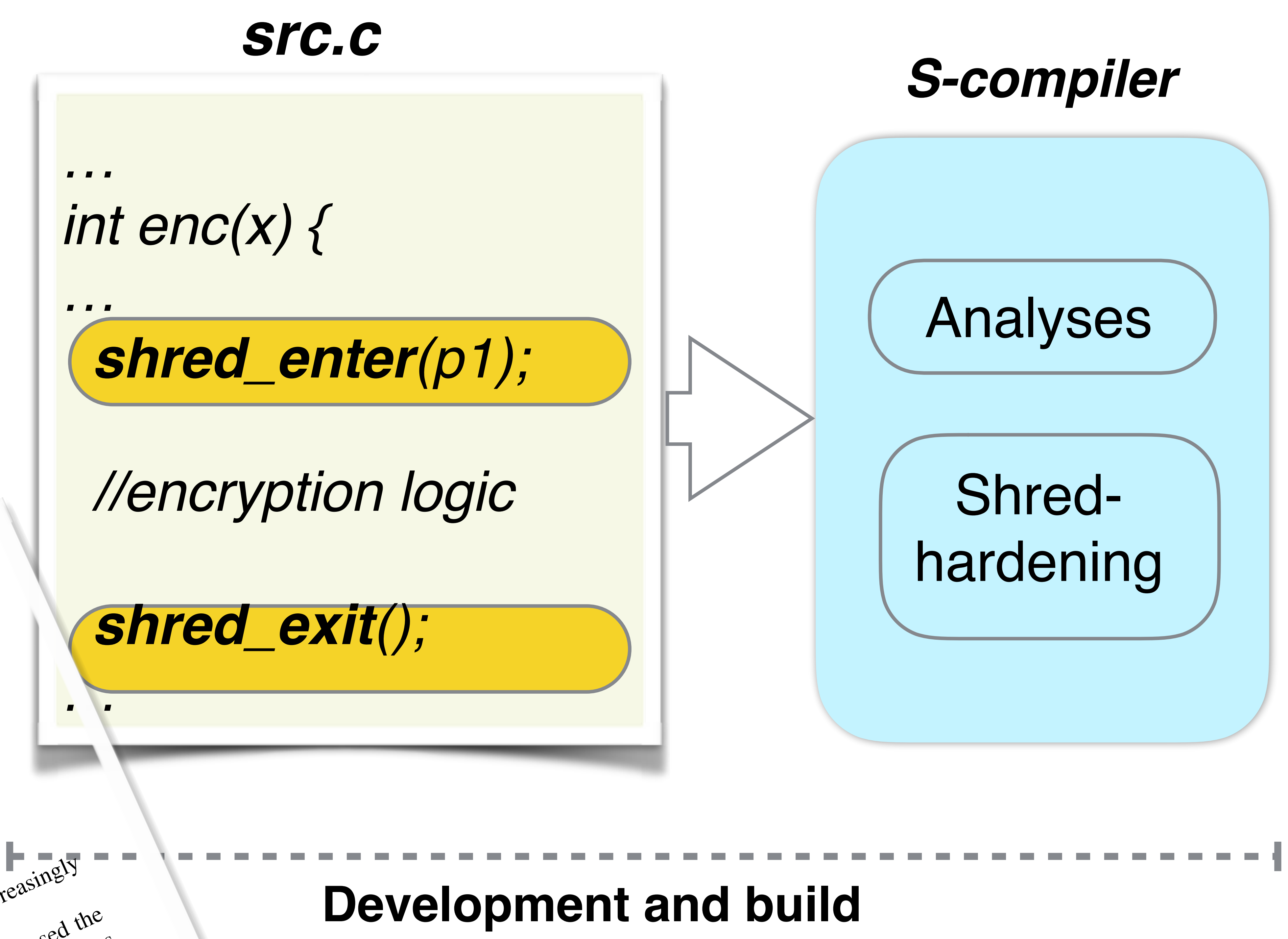
- Secure stacks
  - Each shred has a secure stack allocated from its s-pool

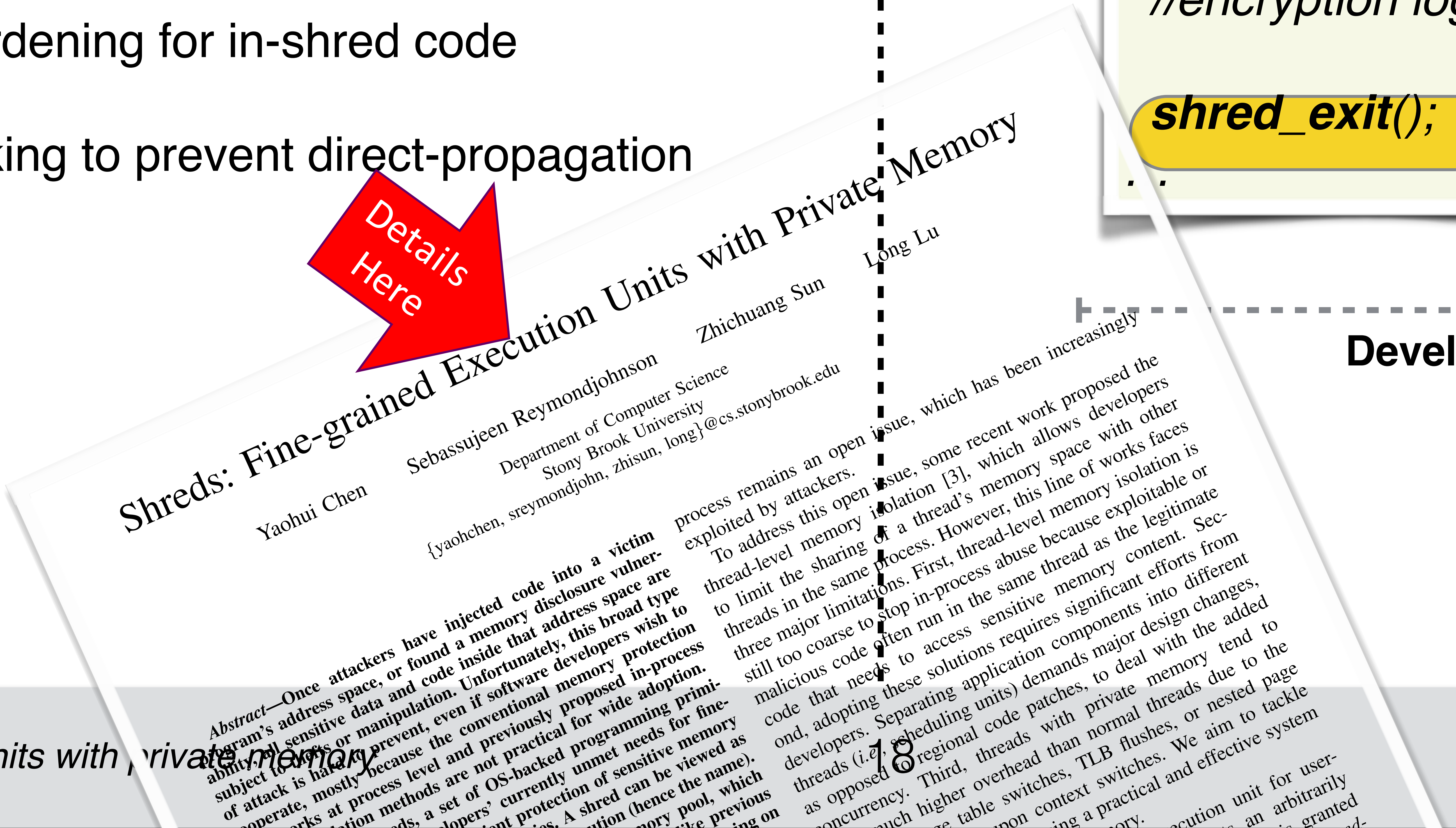
- System interface protection
  - ptrace()
  - /dev/mem
  - Directly read secret from file
  - etc

System Component: S-compiler

S-compiler

- Shred usage verification
- Associate each shred with its s-pool
- Control flow hardening for in-shred code
- Data flow checking to prevent direct-propagation





Evaluation

- Hardware spec: Raspberry Pi 2 Model B (Quad-core Cortex-A7 Processor with 1GB RAM)

Softwares

- **Curl**
- **Minizip**
- **OpenSSH**
- **OpenSSL**
- **Lighttpd**

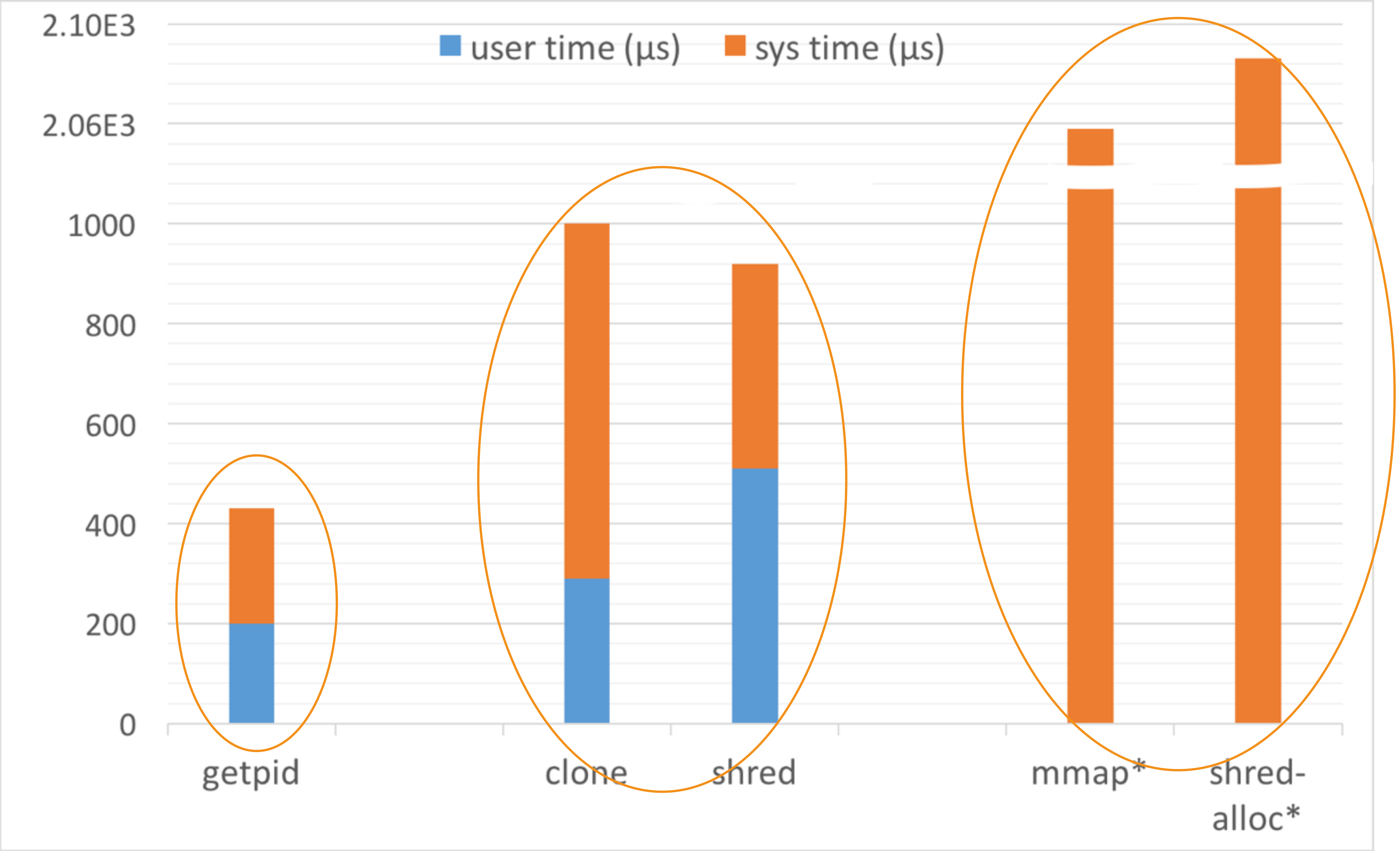
Easy adoption

- Avg. **21** SLOC change
- Avg. **32 min** adoption time

Low overhead

- Avg. **4.67%** slowdown
- Avg. **7.26%** RSS(resident set size) overhead

Evaluation cont.



Conclusion

- Goal— To help developers protect sensitive code/data from **in-process abuse**
- To achieve the goal we propose **shreds** with **private** memory
  - Fine-grained: Flexibly scoped segments of thread executions
  - Efficient and compatible : MMU based domain check
    - No multiple page tables
    - No nested paging
    - No heavy instrumentations
    - No hardware modifications
  - Robust:
    - Prevent out-shred attacks + intra-shred vulnerabilities