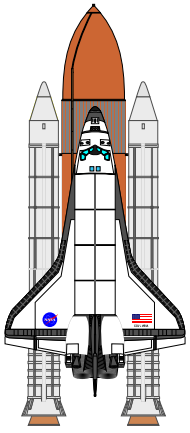
SuperGlue: IDL-Based, System-Level Fault Tolerance for Embedded Systems

June 20, 2016

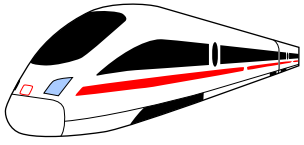
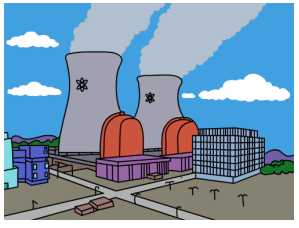
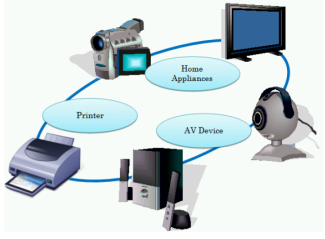
Computer Science Department
The George Washington University

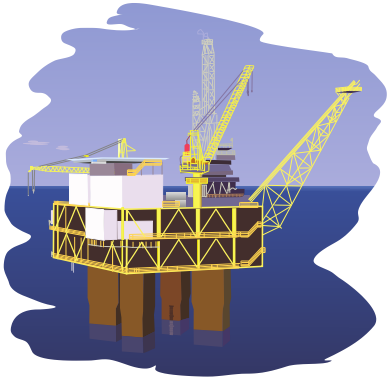
Outlines

Real-Time and Embedded Systems



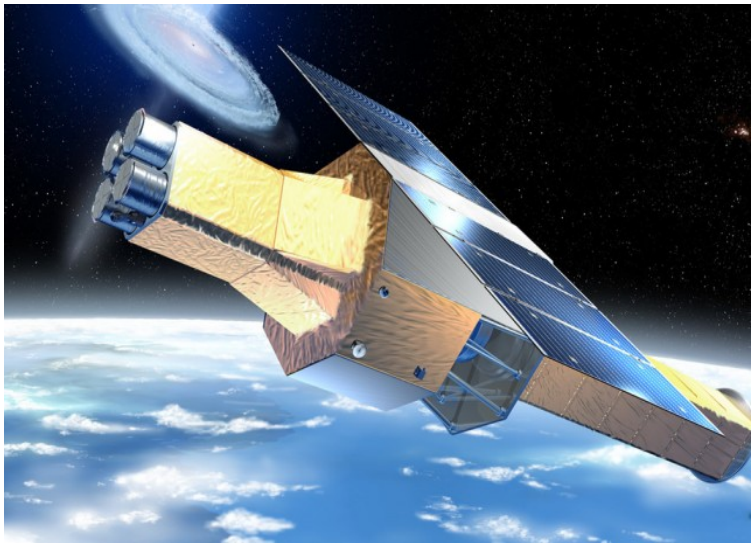




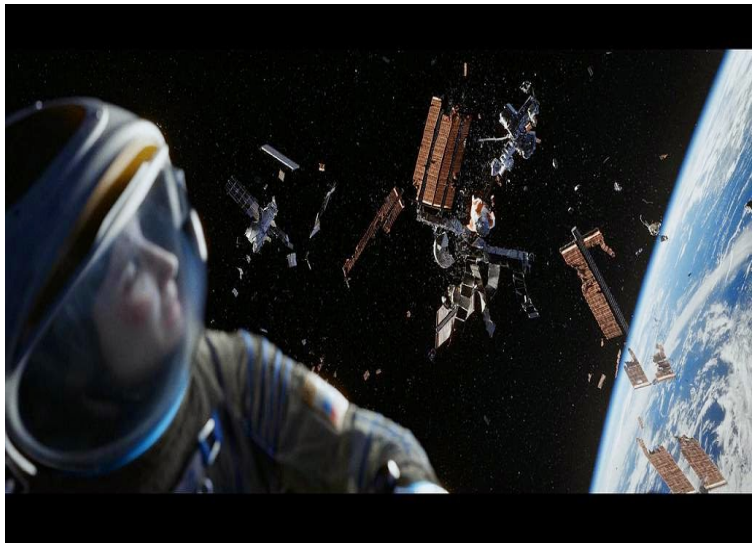

http://www.pcmag.com/encyclopedia/term/42554/embedded-system

Consequences of Embedded System Faults

Japanese X-ray astronomy satellite Hitomi lost in 2016



Fault

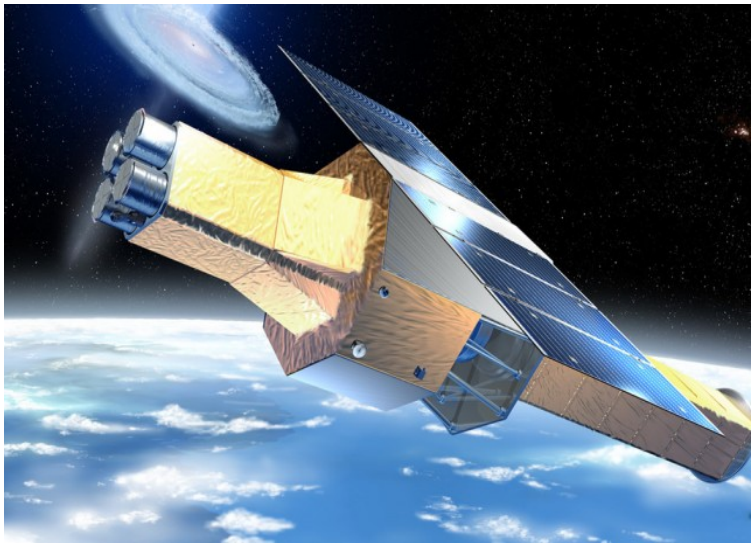


- Financial losses of $286 million USD
- "*It's a scientific tragedy*" - Richard Mushotzky, UMD

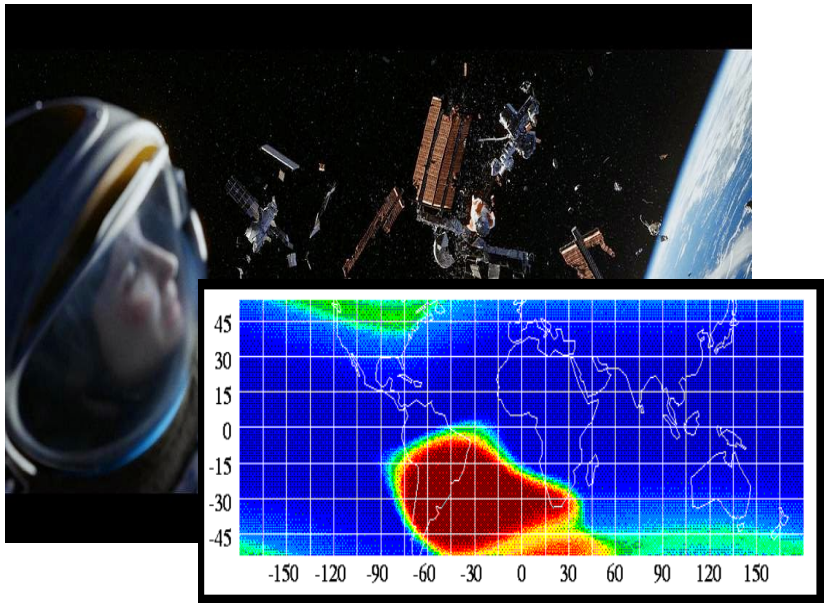
`http://www.nature.com/news/software-error-doomed-japanese-hitomi-spacecraft-1.19835`

Consequences of Embedded System Faults

Japanese X-ray astronomy satellite Hitomi lost in 2016



Fault



The South Atlantic Anomaly

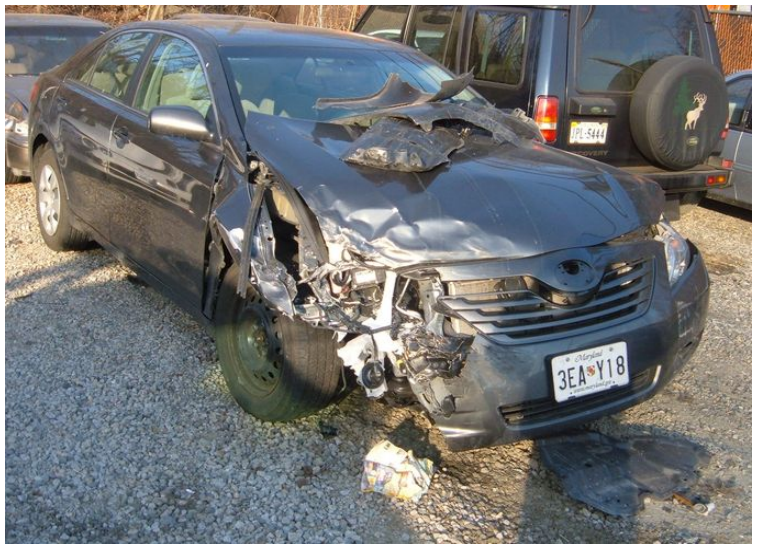
- Financial losses of $286 million USD
- "*It's a scientific tragedy*" - Richard Mushotzky, UMD

`http://www.nature.com/news/software-error-doomed-japanese-hitomi-spacecraft-1.19835`

Consequences of embedded system faults

Toyota Sudden Unintended Acceleration (SUA) in 2004 – 2010

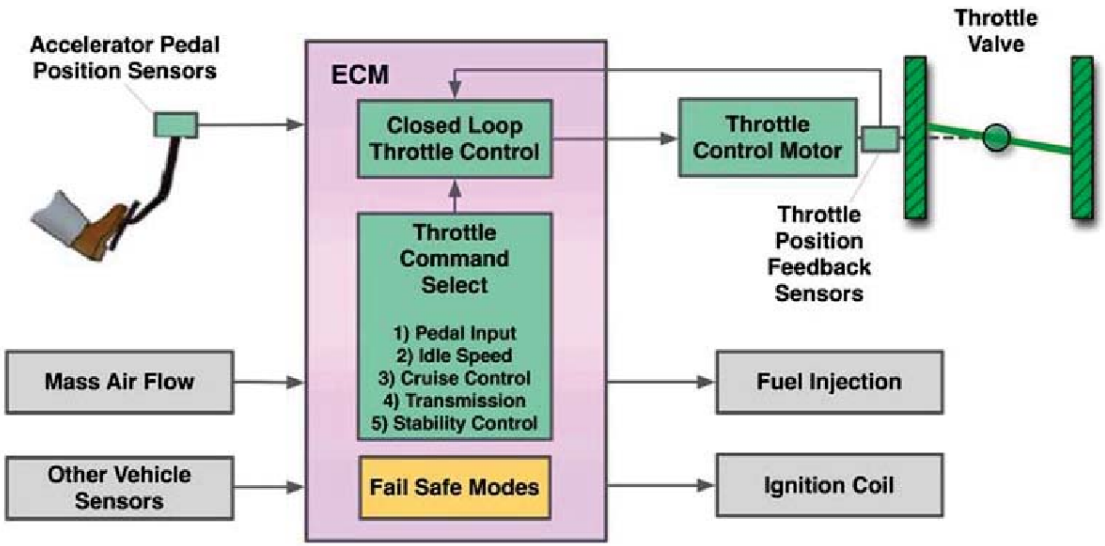
Fault

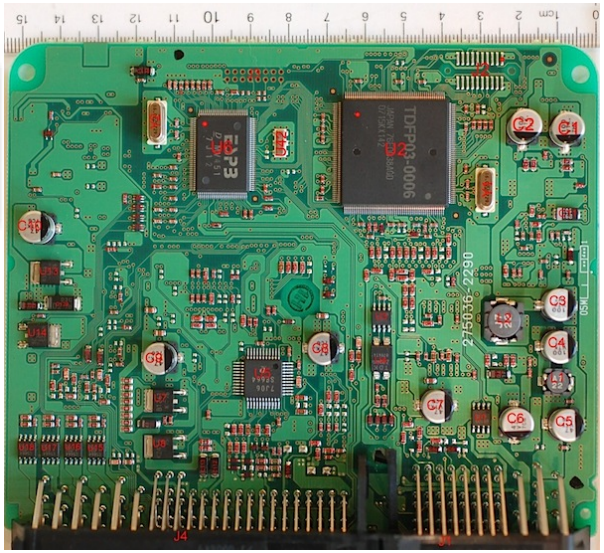


- 89 deaths as of May 2010 and nearly 400 U.S. lawsuits
- Recall 10+ million vehicles and pay $1.2 Billion USD fine

`http://www.cbsnews.com/news/toyota-unintended-acceleration-has-killed-89/`

Sudden Unintended Acceleration



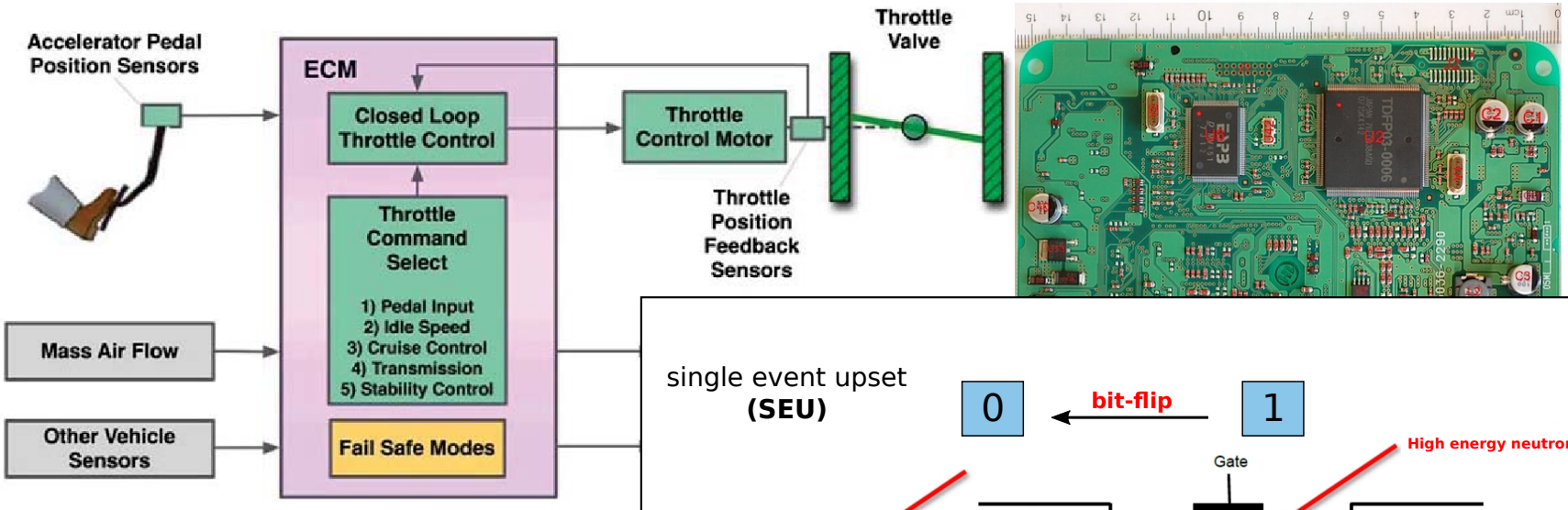


Uncontrolled acceleration in Toyota Camrys

- Electronic Throttle Control System (ETCS)
- OSEK OS, 24 tasks, 280K LOC of C
- bit flip in scheduler data-structures
  $\rightarrow$ reproducible 30-sec uncontrolled acceleration

"*a bit-flip there, will have the effect of killing one of the tasks*"
– Bookout v Toyota

Sudden Unintended Acceleration



Radiation strike causing transistor disruption [Gorini 2012]

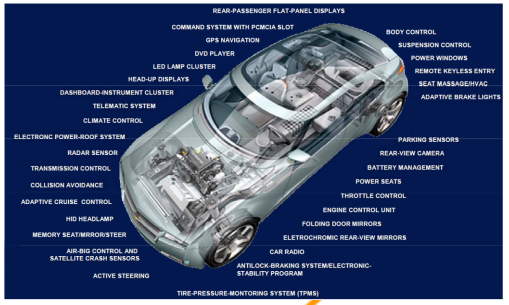
Uncontrolled acceleration in

- Electronic Throttle Con
- OSEK OS, 24 tasks, 28
- bit flip in scheduler data-structures
  $\rightarrow$ reproducible 30-sec uncontrolled acceleration

"*a bit-flip there, will have the effect of killing one of the tasks*"
– Bookout v Toyota

Embedded faults: Bad Now, Worse Tomorrow

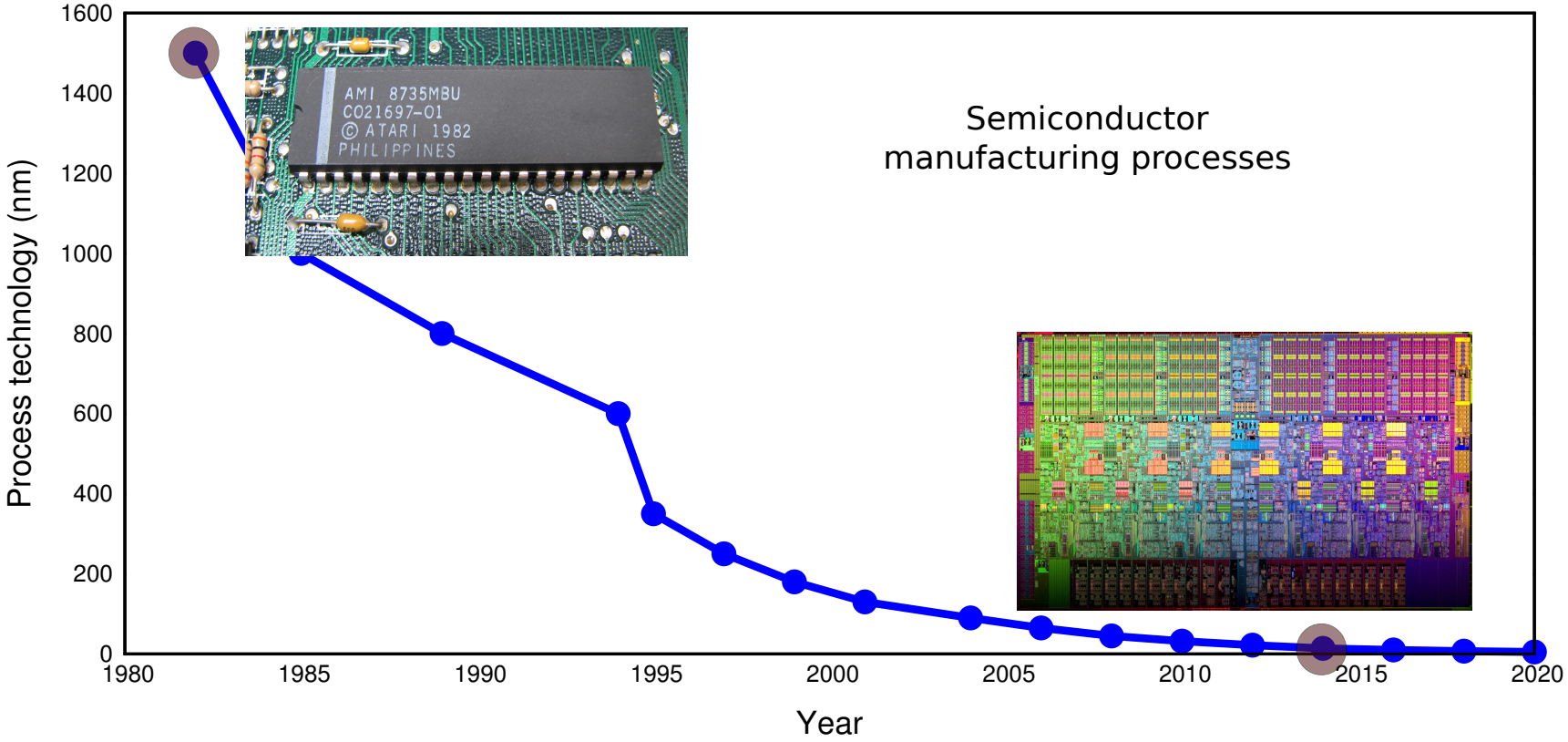
ABS: Antilock Brake System
ACC: Adaptive Cruise Control
BCM: Body Control Module
Cyl Deact: Cylinder Deactivation
ECS: Electronics, Controls, and Software
EGR: Exhaust Gas Recirculation.
GDI: Gas Direct Injection
OBD: Onboard Diagnostics
TCC: Torque Converter Clutch
PT: Powertrain
ESC: Electronic Stability Control



\+ more functionality
\− more complexity → dependability more challenging

https://www.src.org/calendar/e003506/03-sessionii-rajkumar.pdf

Embedded faults: Bad Now, Worse Tomorrow



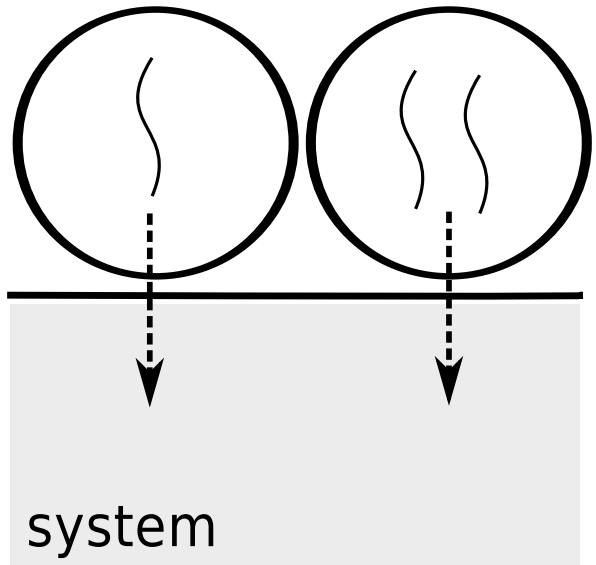
Decreasing process sizes $\rightarrow$ 5nm

- \+ faster
- \+ less power
- \+ smaller
- – increased vulnerability to HW transient faults

Application-Level Fault Tolerance

Application fault tolerance

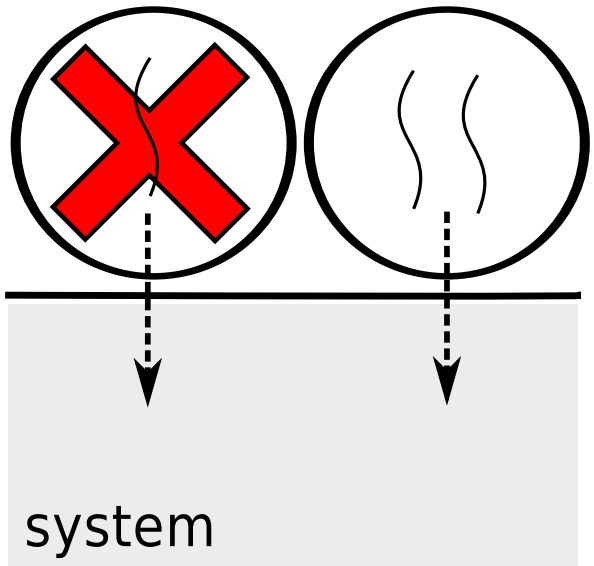
- example recovery techniques
  - recovery blocks
  - checkpointing
  - re-fork
- temporal redundancy
  - detect fault by job completion
  - replay execution from saved state



Application-Level Fault Tolerance

Application fault tolerance

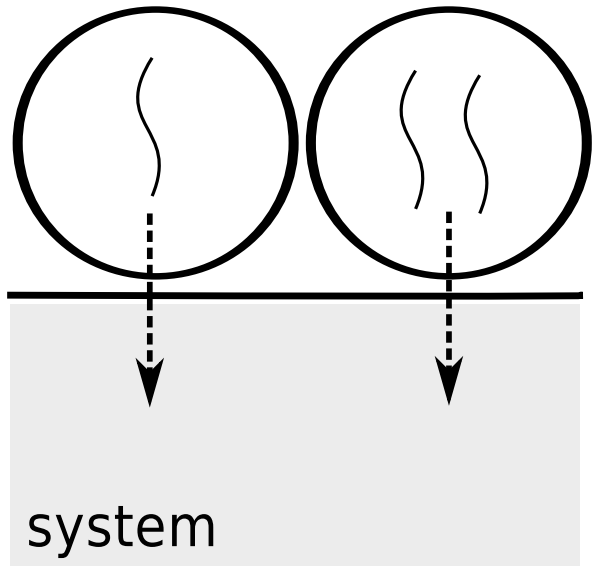
- example recovery techniques
  - recovery blocks
  - checkpointing
  - re-fork
- temporal redundancy
  - detect fault by job completion
  - replay execution from saved state



Application-Level Fault Tolerance

Application fault tolerance

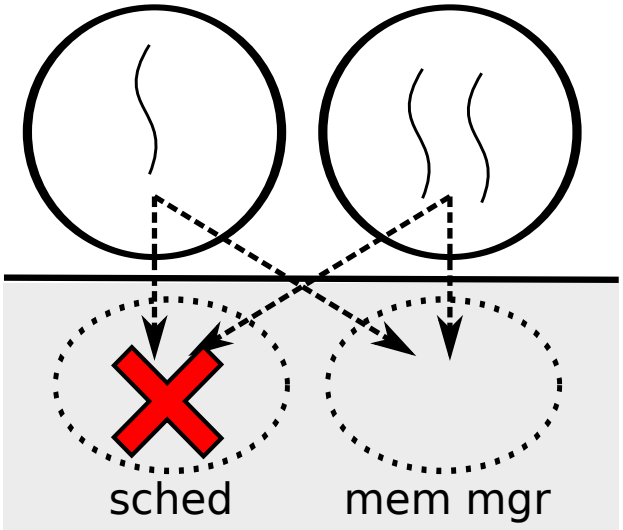
- example recovery techniques
  - recovery blocks
  - checkpointing
  - re-fork
- temporal redundancy
  - detect fault by job completion
  - replay execution from saved state



System-Level Fault Tolerance

System-level fault tolerance

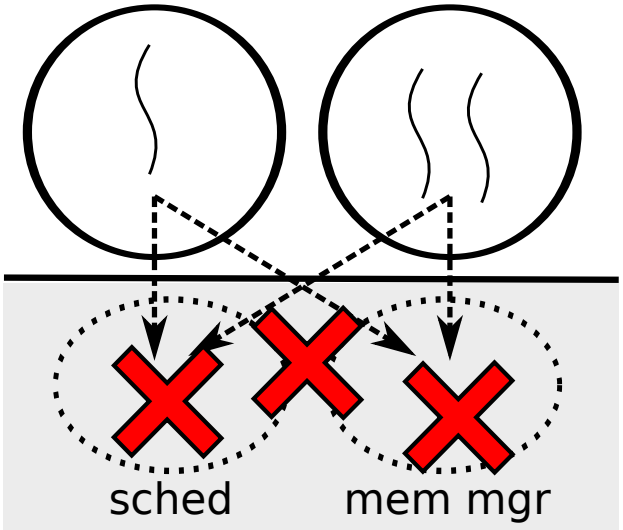
- failures in
  - scheduler
  - memory mapping manager
  - file-systems
  - synchronization manager
  - ...



System-Level Fault Tolerance

System-level fault tolerance

- failures in
  - scheduler
  - memory mapping manager
  - file-systems
  - synchronization manager
  - ...



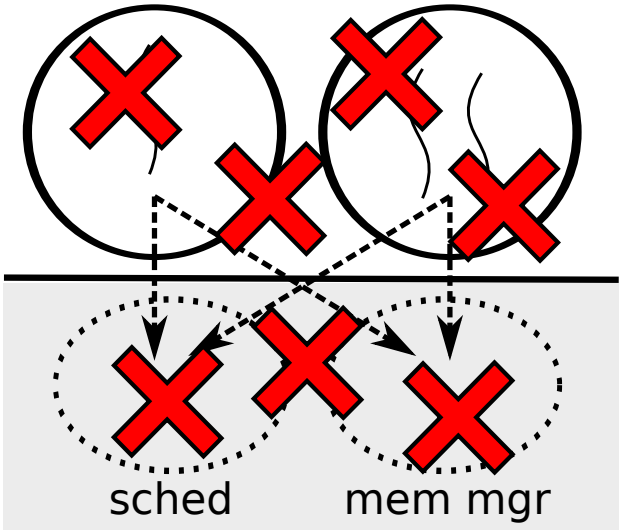
System components contain state for all tasks

- failure impacts memory of *all* tasks

System-Level Fault Tolerance

System-level fault tolerance

- failures in
  - scheduler
  - memory mapping manager
  - file-systems
  - synchronization manager
  - ...



System components contain state for all tasks

- failure impacts memory of *all* tasks

Recovery requires resources

- processing time...to recovery scheduler
- memory...to recover memory mapper

System-Level Fault Tolerance

System-level fault tolerance

- failures in
  - scheduler
  - memory mapping manager
  - file-systems
  - synchronization manager
  - ...

System components contain state for all tasks

- failure impacts memory of *all* tasks

Recovery requires resources

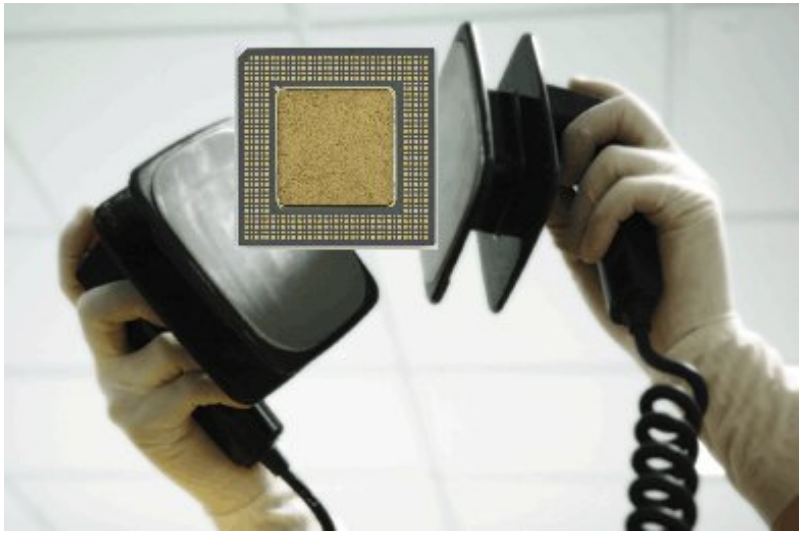
- processing time...to recovery scheduler
- memory...to recover memory mapper

Outlines

C^3: The Computational Crash Cart

C^3: Computational Crash Cart

- resuscitate system
- from system-level faults
- predictably



Main ideas

- pervasive fault isolation $\rightarrow$ restrict propagation
- efficient μ-reboot of individual components
- interface-driven, application-oblivious recovery

C^3 – an interface-driven, predictable system-level fault recovery mechanism

J. Song, J. Wittrock, and G. Parmer, "Predictable, efficient system-level fault tolerance in C^3" in RTSS, 2013

Composite: A Component-Based OS

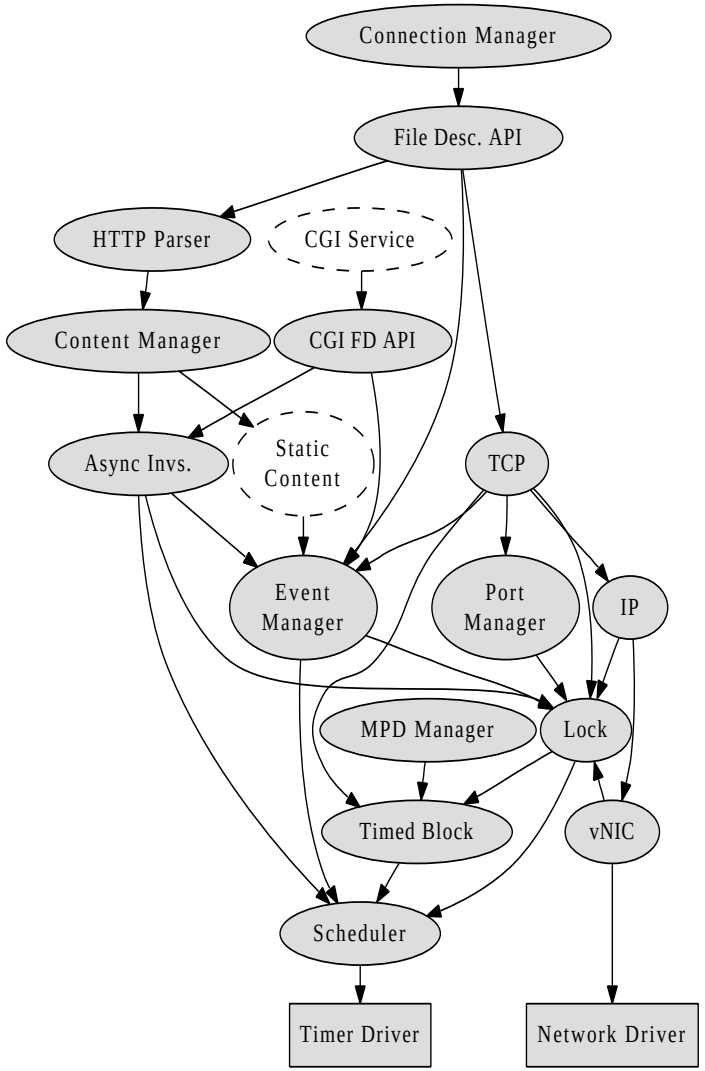
System functionality as *components*

- user-level, protection domains
- fine-grained fault isolation

Low-level services are components

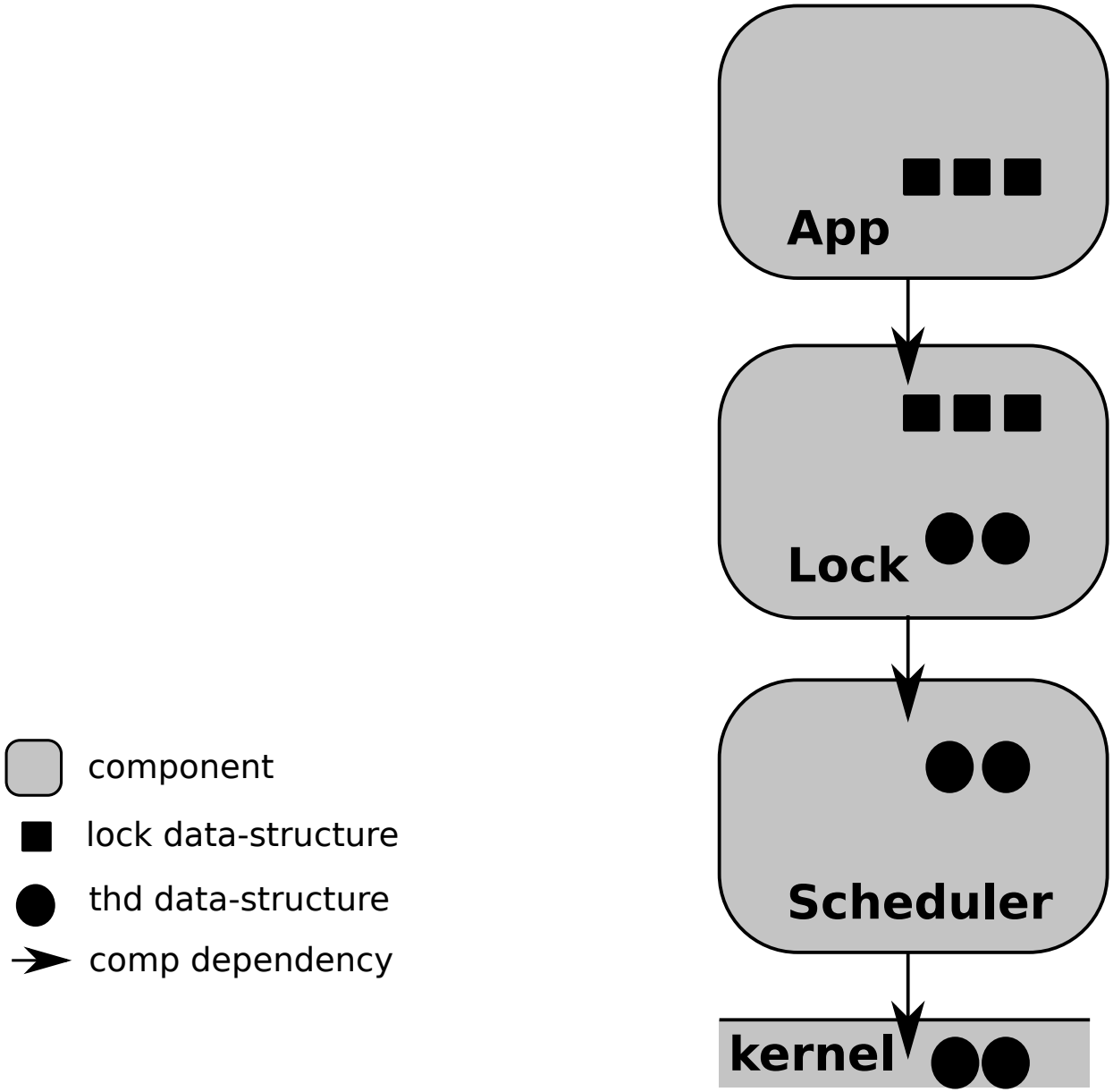
- kernel is small and policy-less
- scheduling, memory mapping, synchronization...are in user level

Components interact via *invocation* of interface functions

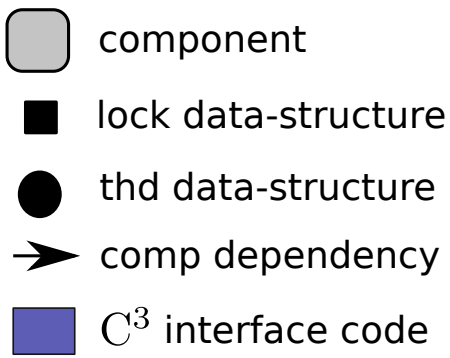


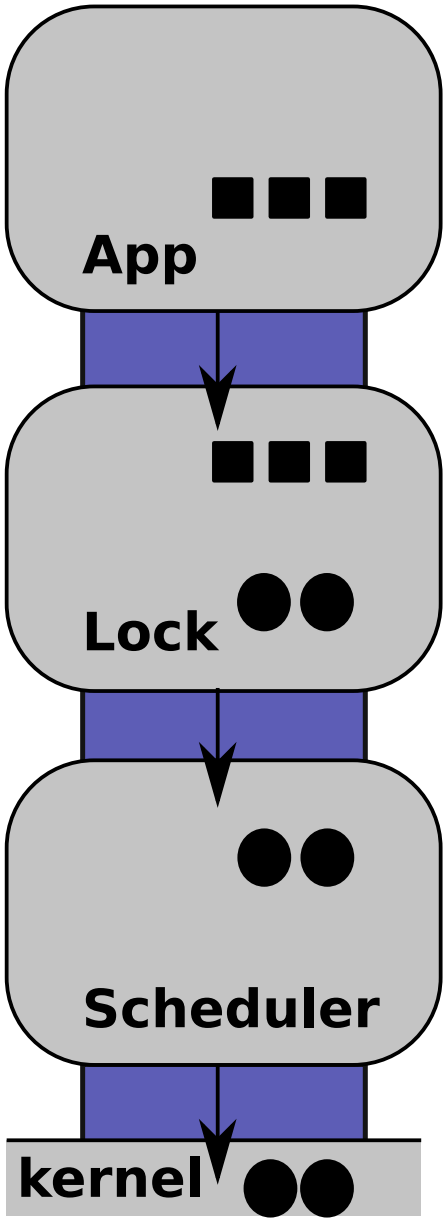
Example component graph

C^3 System-Level Fault Recovery



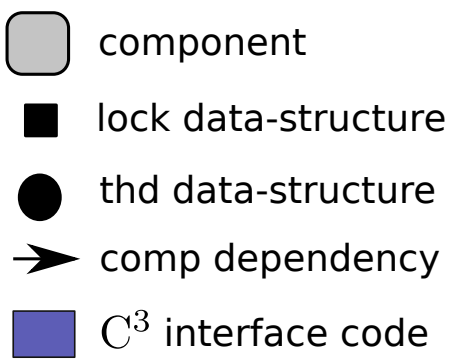
Recovery sequence in C^3

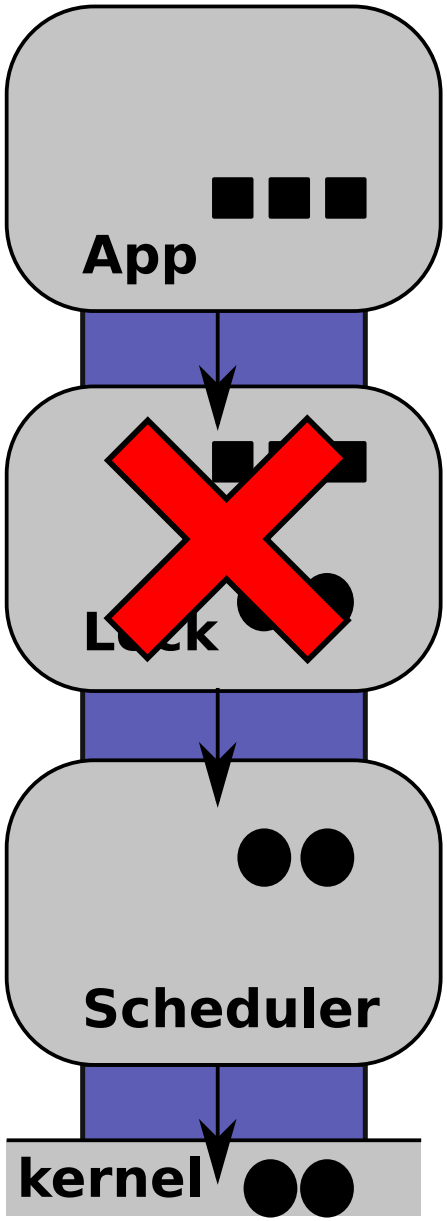




Recovery sequence in C³

1. fault detection

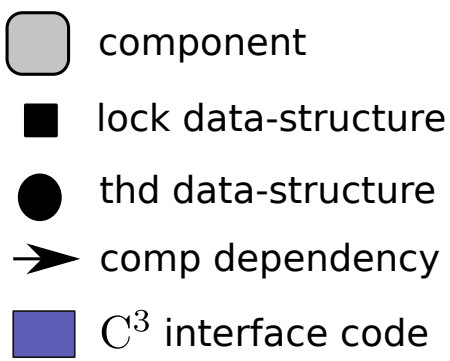


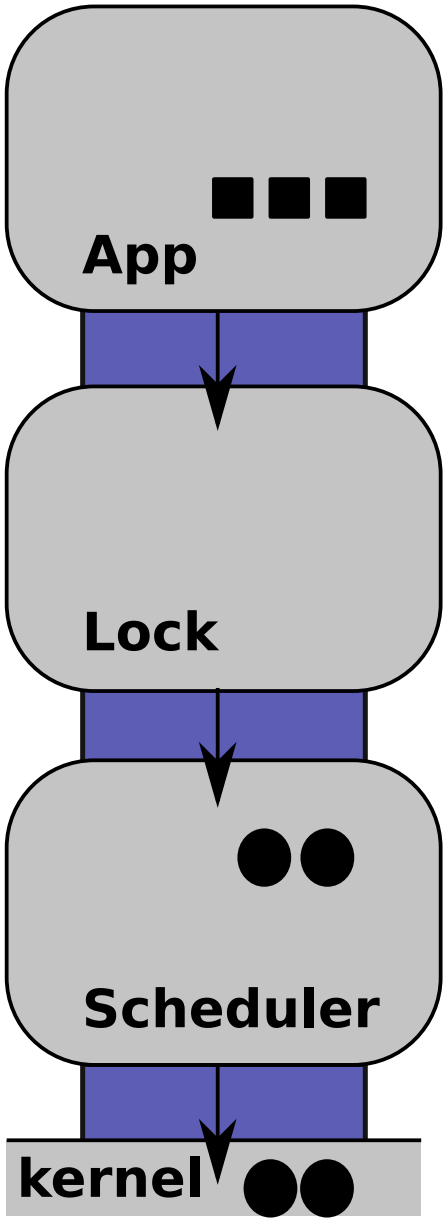


C^3 System-Level Fault Recovery

Recovery sequence in C^3

1. fault detection
2. safe-state recovery
   - μ-reboot

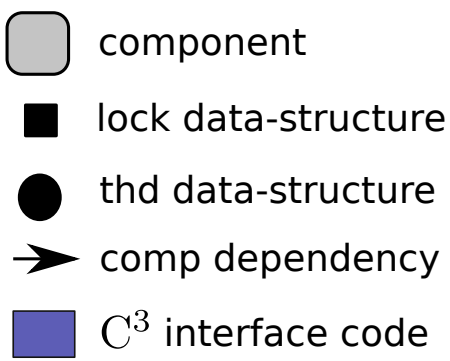


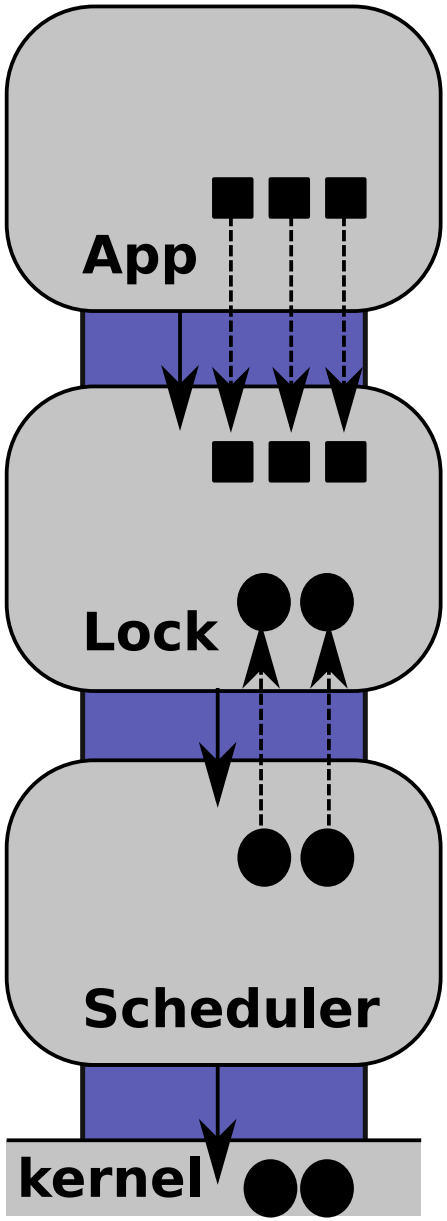


C^3 System-Level Fault Recovery

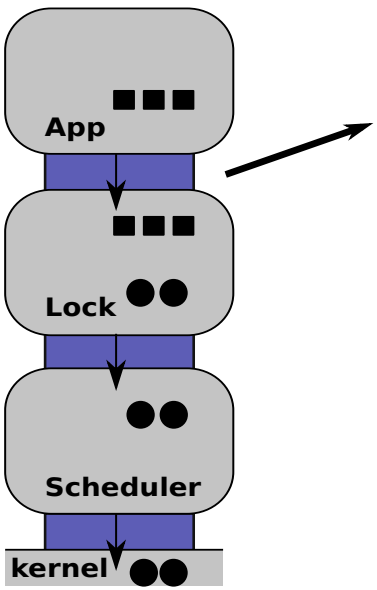
Recovery sequence in C^3

1. fault detection
2. safe-state recovery
   - μ-reboot
3. consistent-state recovery
   - object state recovery



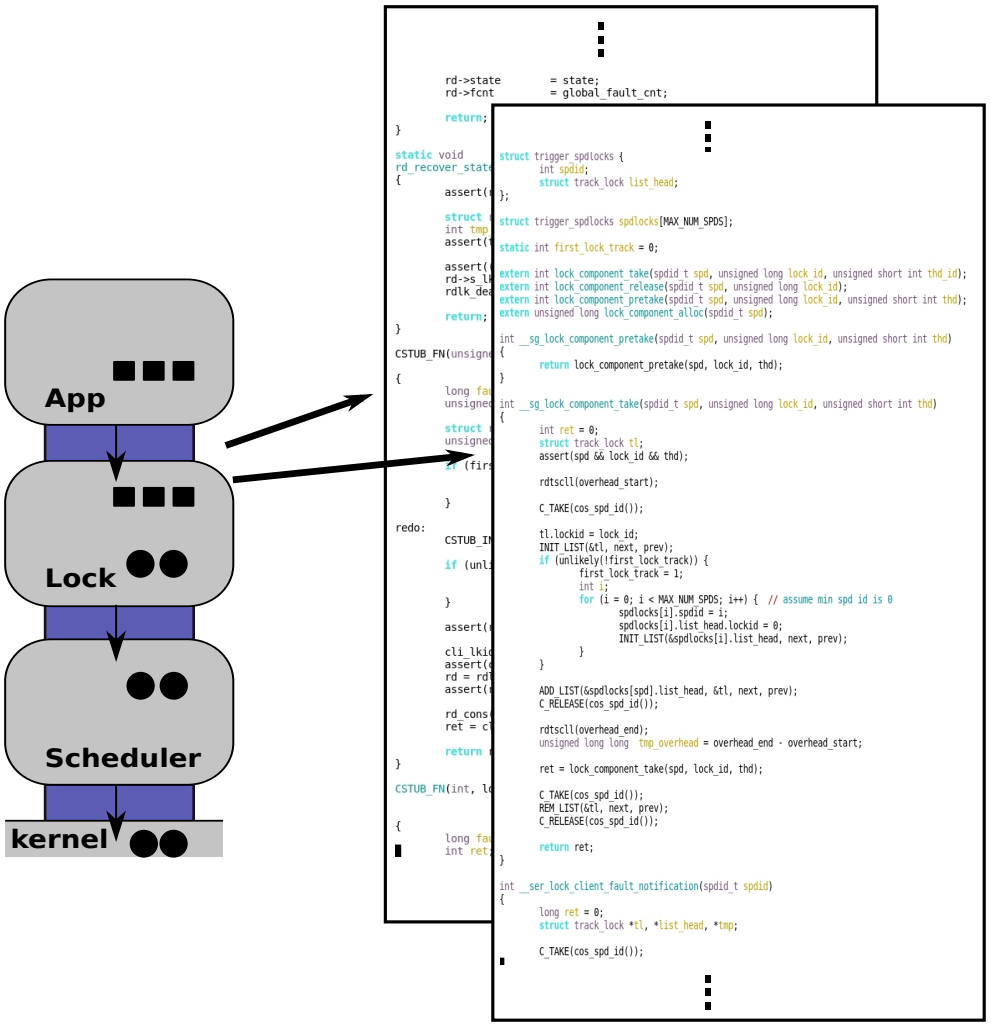


C^3 Implementation – Writing Code Manually

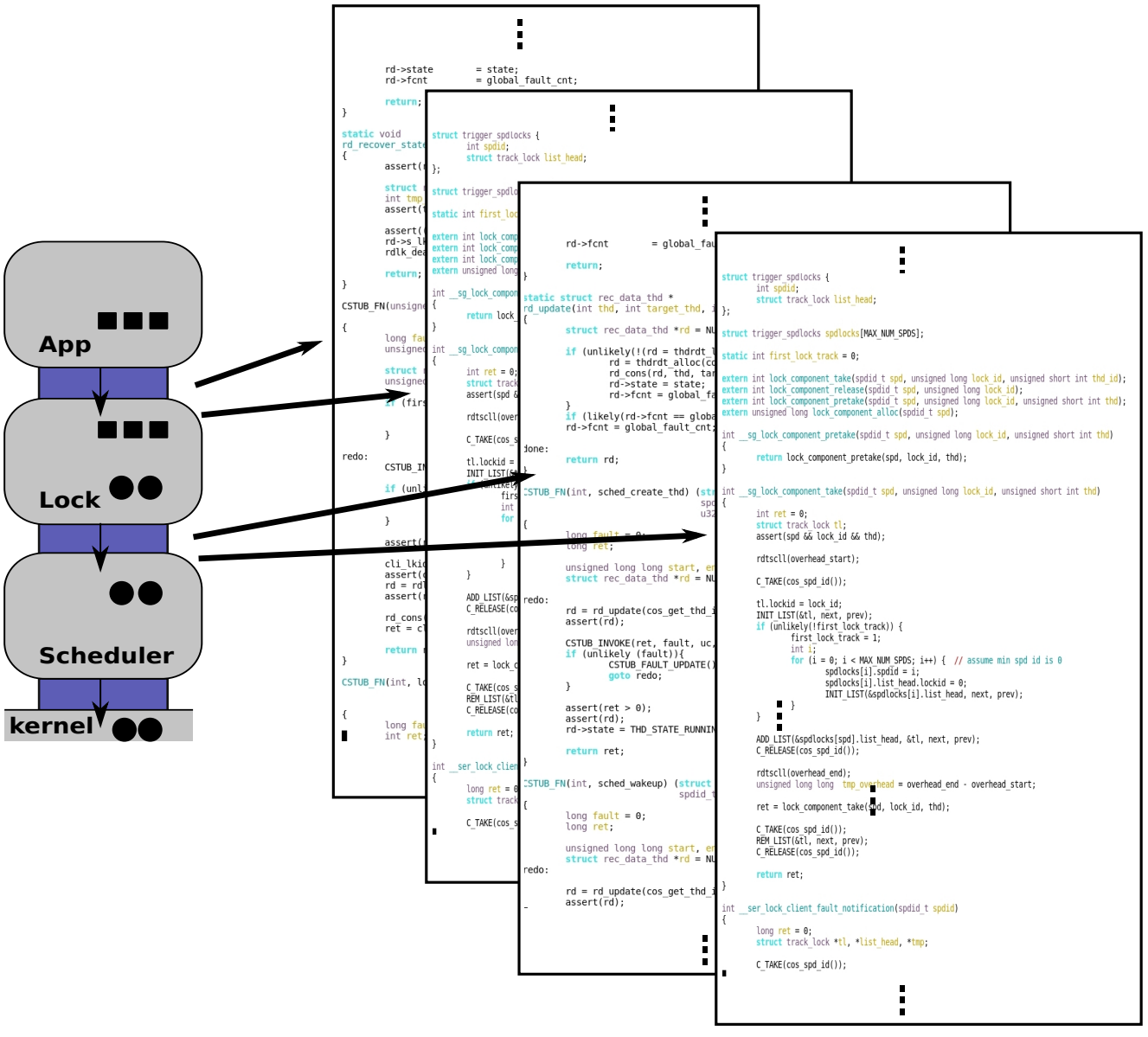



```
        rd->state        = state;
        rd->fcnt         = global_fault_cnt;

        return;
}

static void
rd_recover_state(struct rec_data_lk *rd)
{
        assert(rd && rd->c_lkid);

        struct rec_data_lk *tmp;
        int tmp_lkid = lock_component_alloc(cos_spd_id());
        assert(tmp_lkid);

        assert((tmp = rdlk_lookup(tmp_lkid)));
        rd->s_lkid = tmp->s_lkid;
        rdlk_dealloc(tmp_lkid);

        return;
}

CSTUB_FN(unsigned long, lock_component_alloc) (struct usr_inv_cap *uc,
                                               spdid_t spdid)
{
        long fault;
        unsigned long ret;

        struct rec_data_lk *rd = NULL;
        unsigned long ser_lkid, cli_lkid;

        if (first == 0) {
                cos_map_init_static(&uniq_lkids);
                first = 1;
        }

redo:
        CSTUB_INVOKE(ret, fault, uc, 1, spdid);

        if (unlikely (fault)){
                CSTUB_FAULT_UPDATE();
                goto redo;
        }

        assert(ret > 0);

        cli_lkid = rdlk_alloc();
        assert(cli_lkid >= 1);
        rd = rdlk_lookup(cli_lkid);
        assert(rd);

        rd_cons(rd, cos_spd_id(), cli_lkid, ret, LOCK_ALLOC);
        ret = cli_lkid;

        return ret;
}

CSTUB_FN(int, lock_component_pretake) (struct usr_inv_cap *uc,
                                       spdid_t spdid, unsigned long lock_id,
                                       unsigned short int thd)
{
        long fault = 0;
        int ret;
```

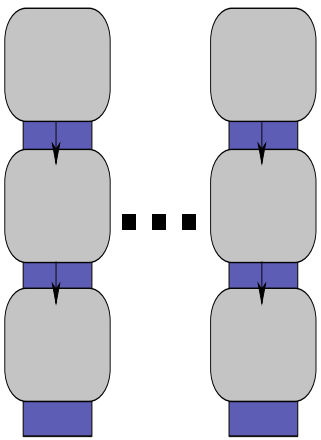
C^3 Implementation – Writing Code Manually



C³ Implementation – Writing Code Manually

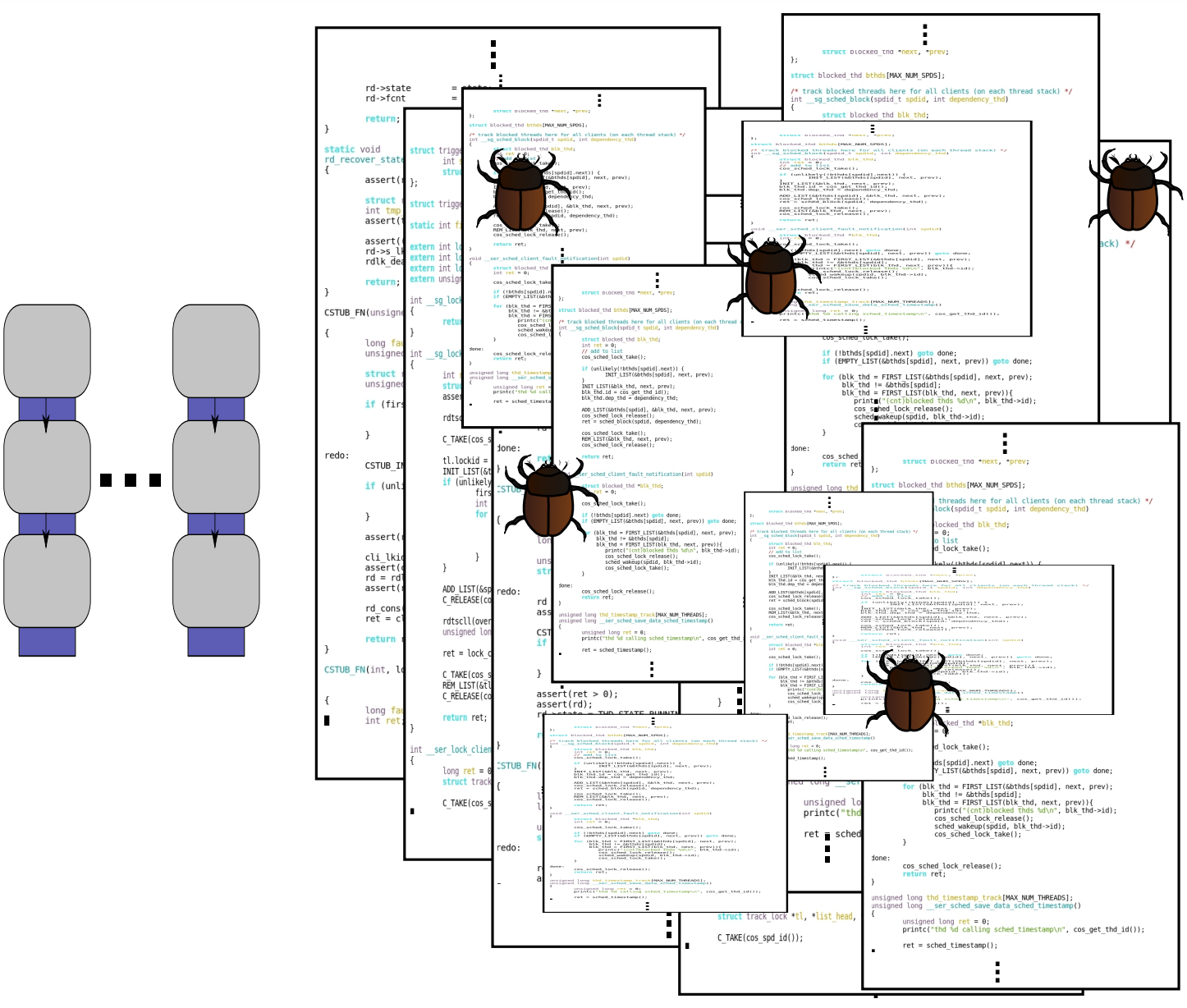


C^3 Implementation – Writing Code Manually

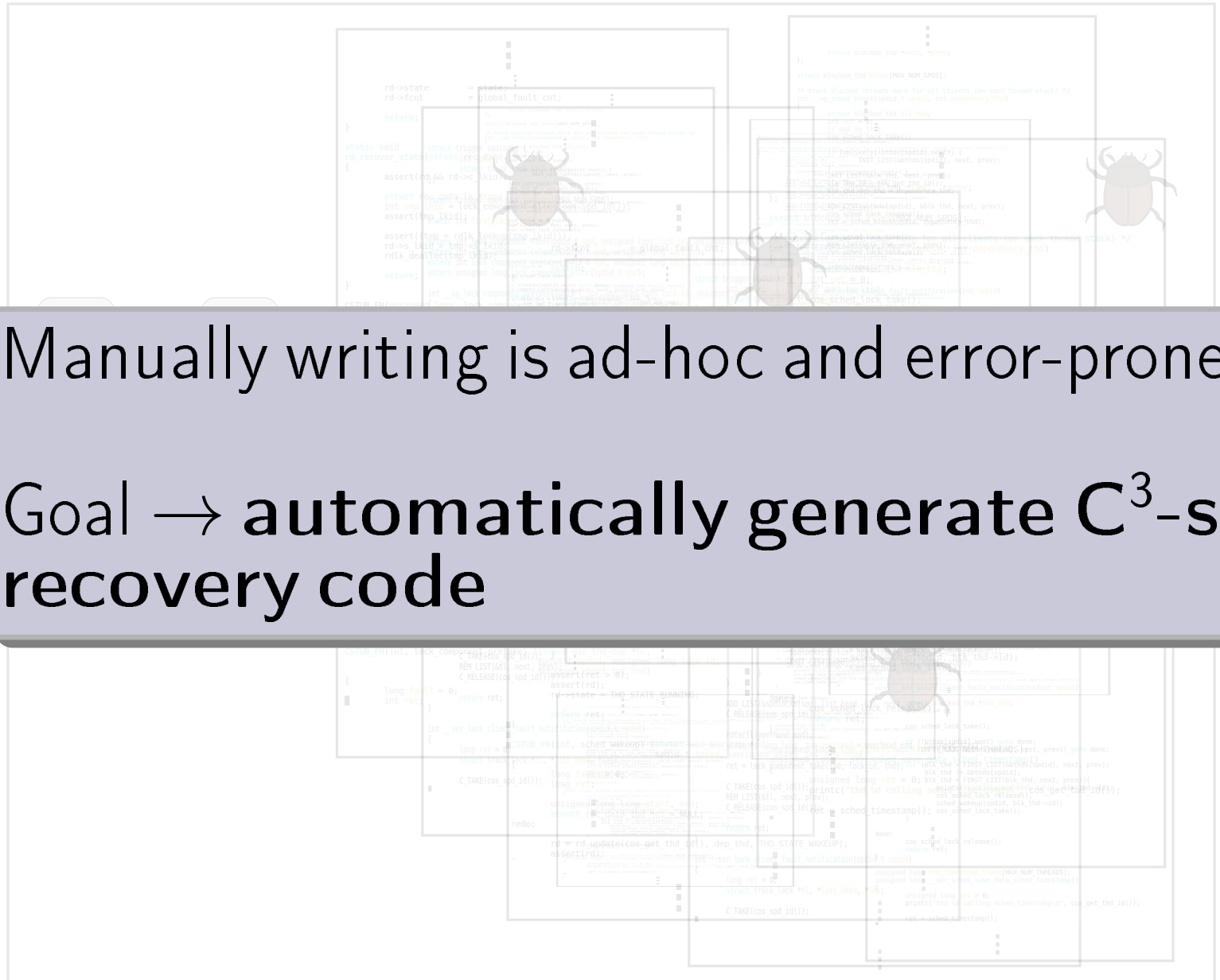


lock
scheduler
memory manager
file system
event manager
timer manager
...

C³ Implementation – Writing Code Manually



C^3 Implementation – Writing Code Manually



Manually writing is ad-hoc and error-prone

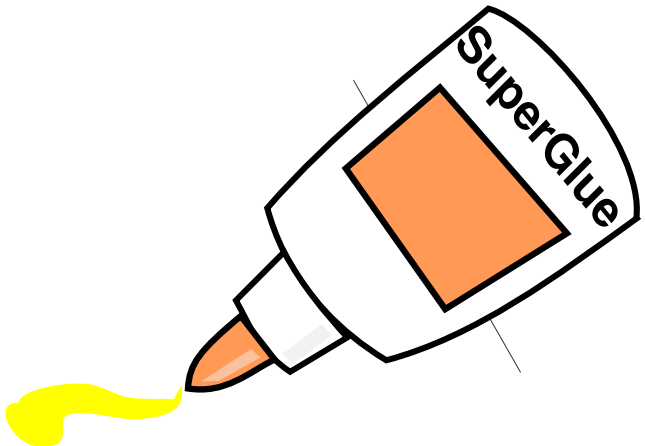
Goal $\rightarrow$ **automatically generate C^3-style recovery code**

Outlines

SuperGlue – IDL-based System-Level Fault Tolerance

SuperGlue

- automatically generate C^3 fault recovery code
- for all system-level services



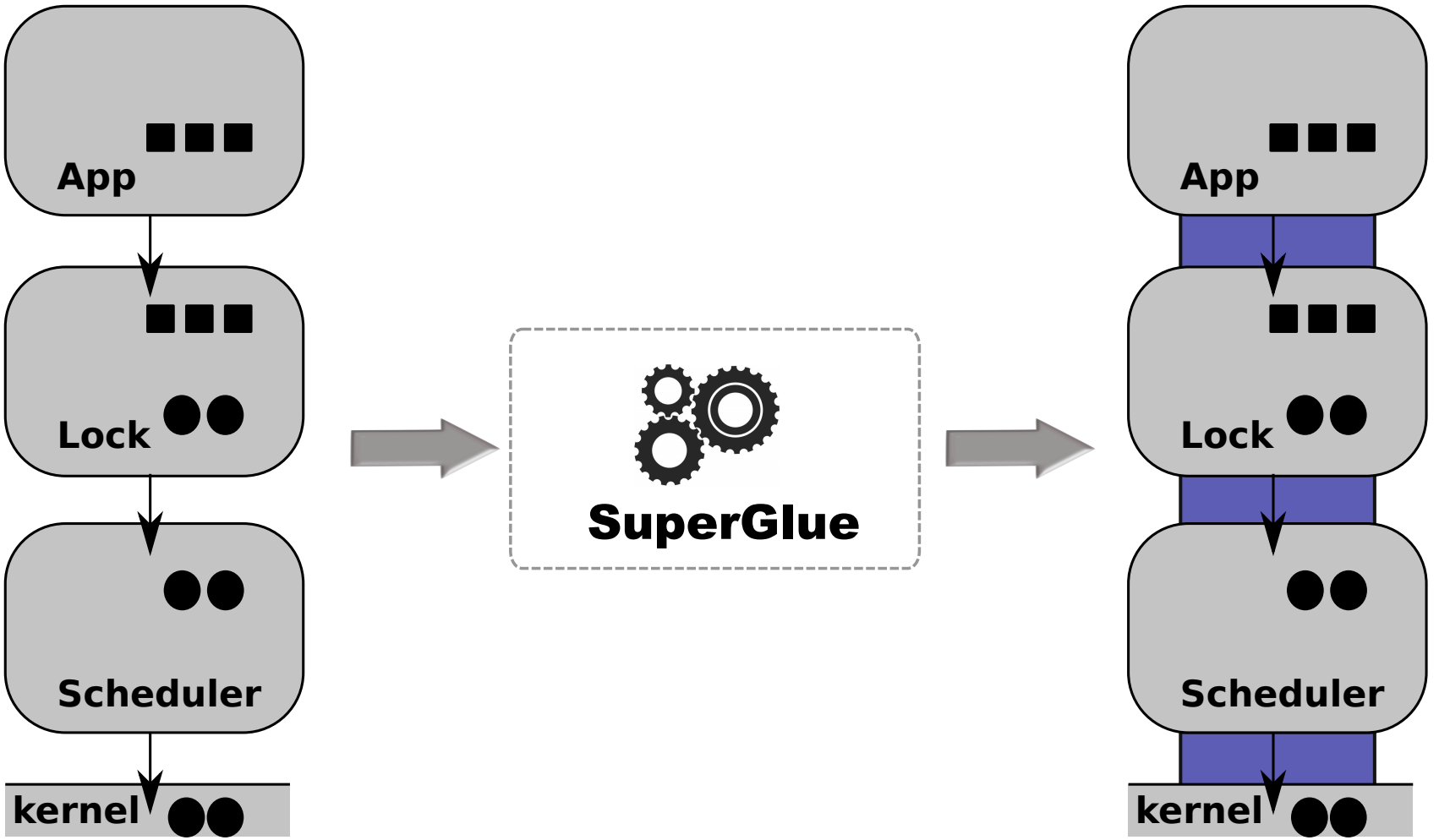
Main ideas

- a system model and interface description language (IDL)
- an IDL compiler synthesizes recovery code from the model

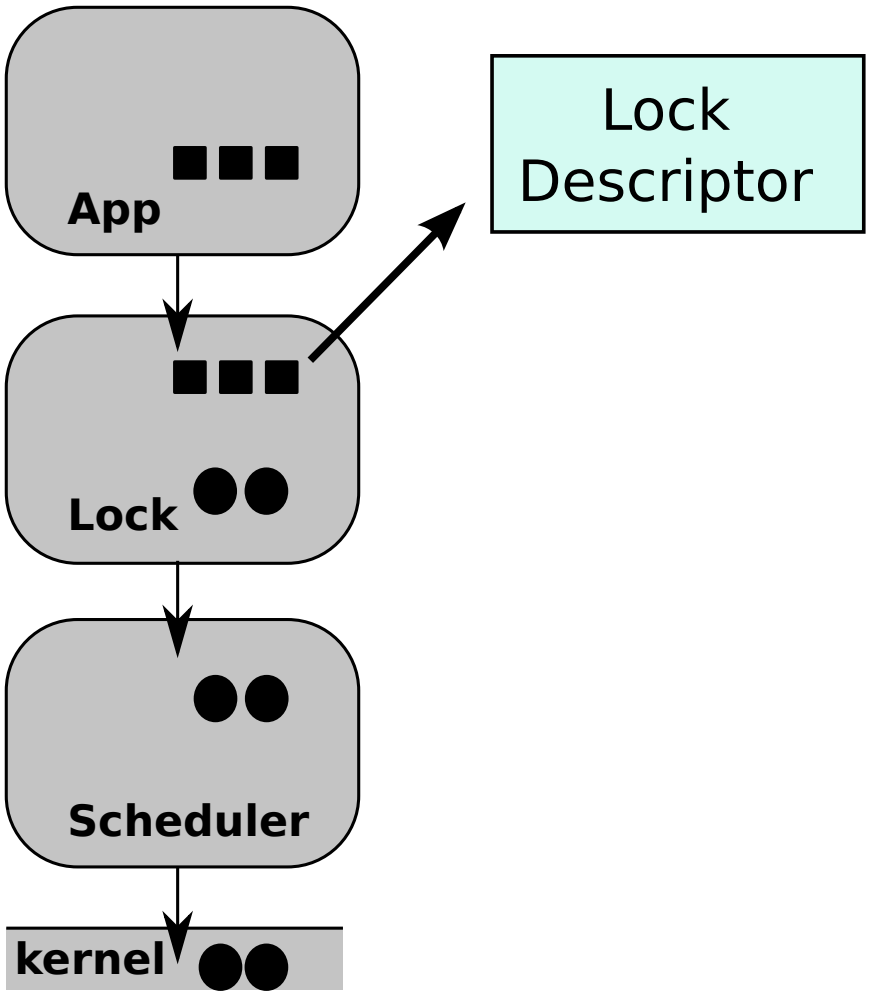
Fault Model (same as C^3):

- processor SEUs and fault is detected immediately (*fail-stop*)

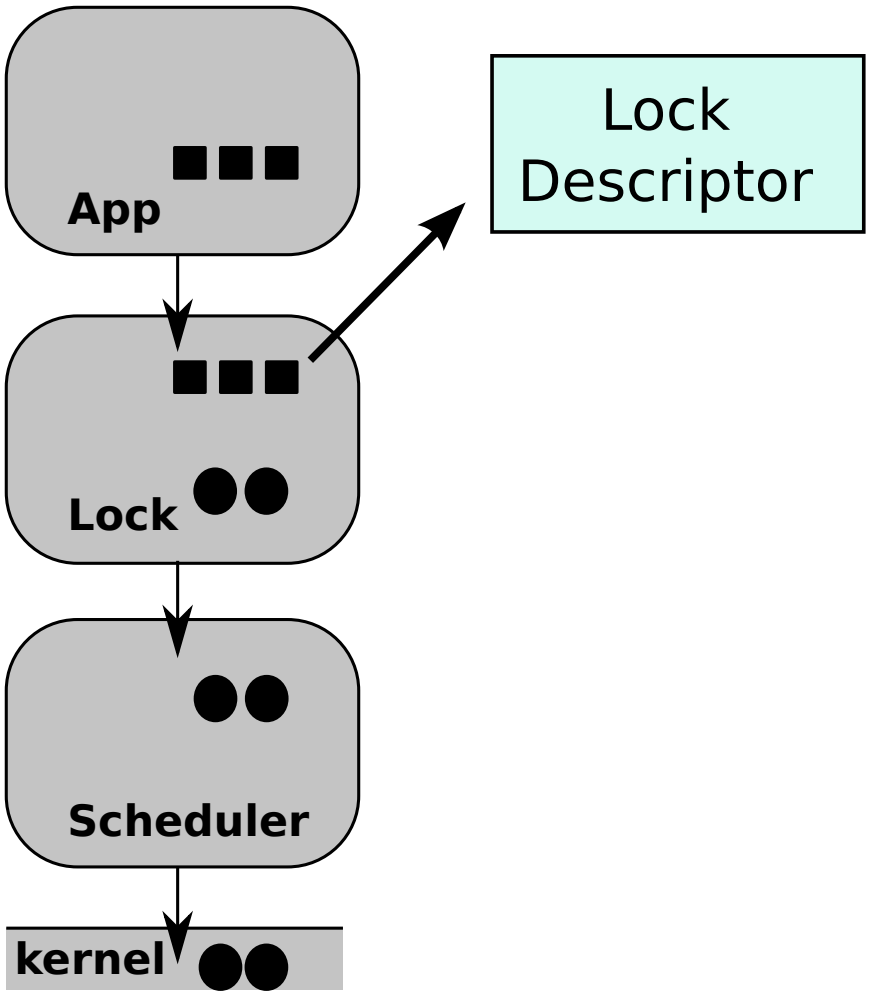
SuperGlue – IDL-based System-Level Fault Tolerance



System Model – Descriptor and Resource



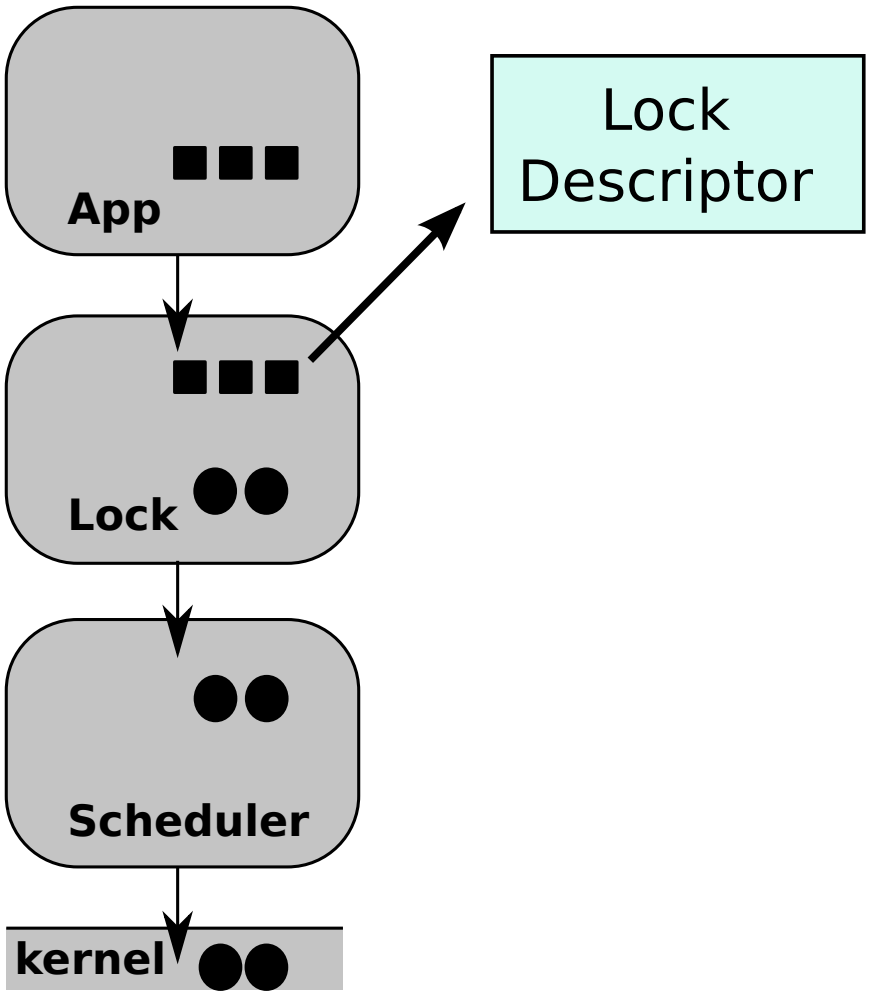
System Model – Descriptor and Resource



Descriptor

- accessed locally or globally

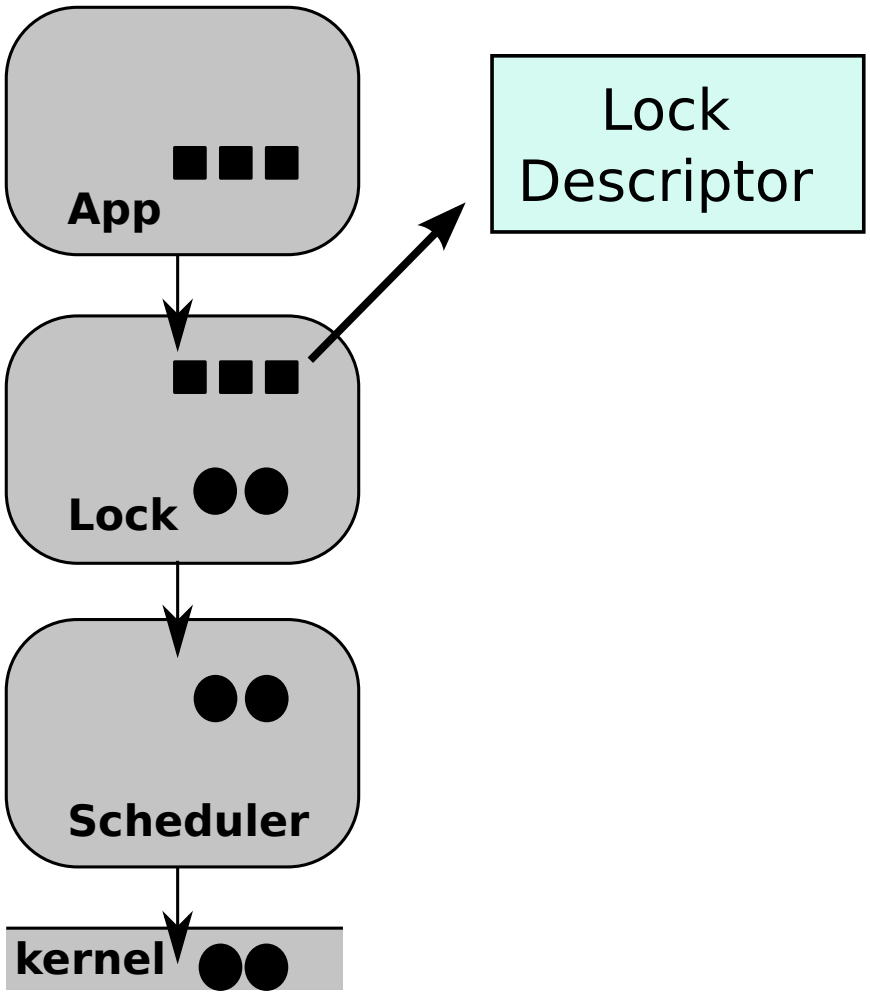
System Model – Descriptor and Resource



Descriptor

- accessed locally or globally
- meta data

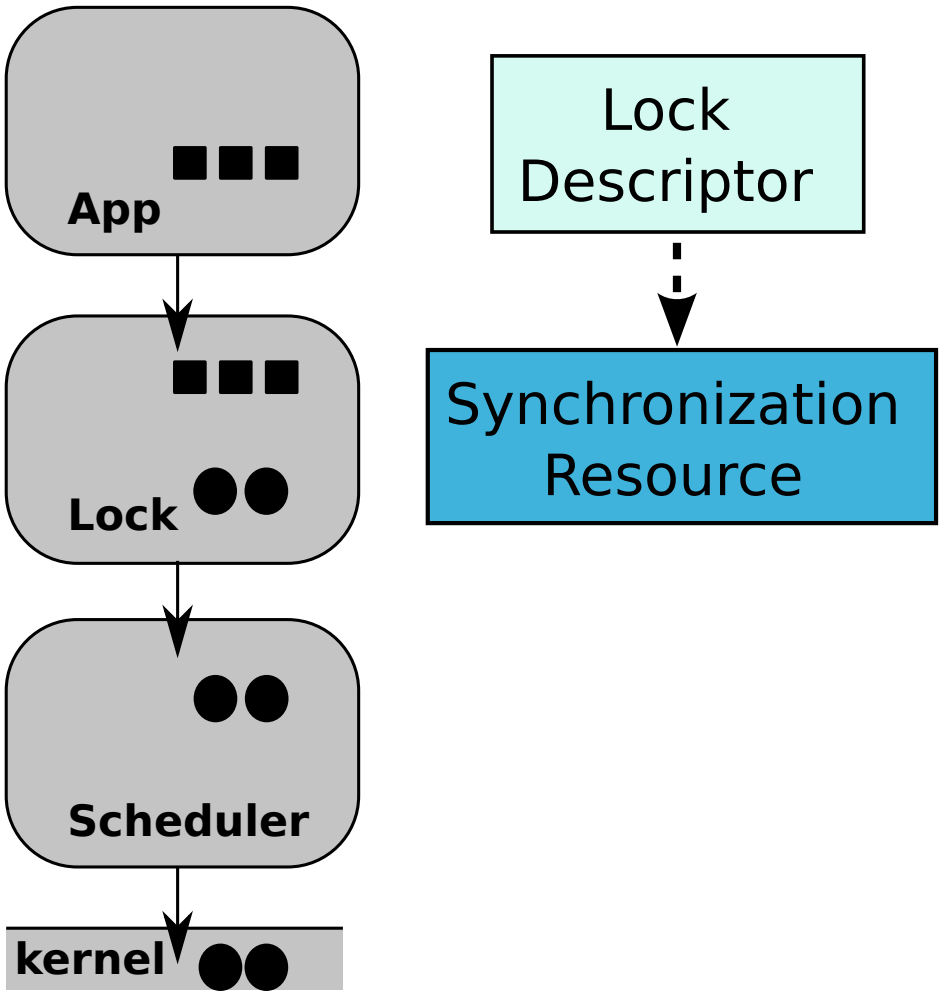
System Model – Descriptor and Resource



Descriptor

- accessed locally or globally
- meta data
- dependent descriptor

System Model – Descriptor and Resource

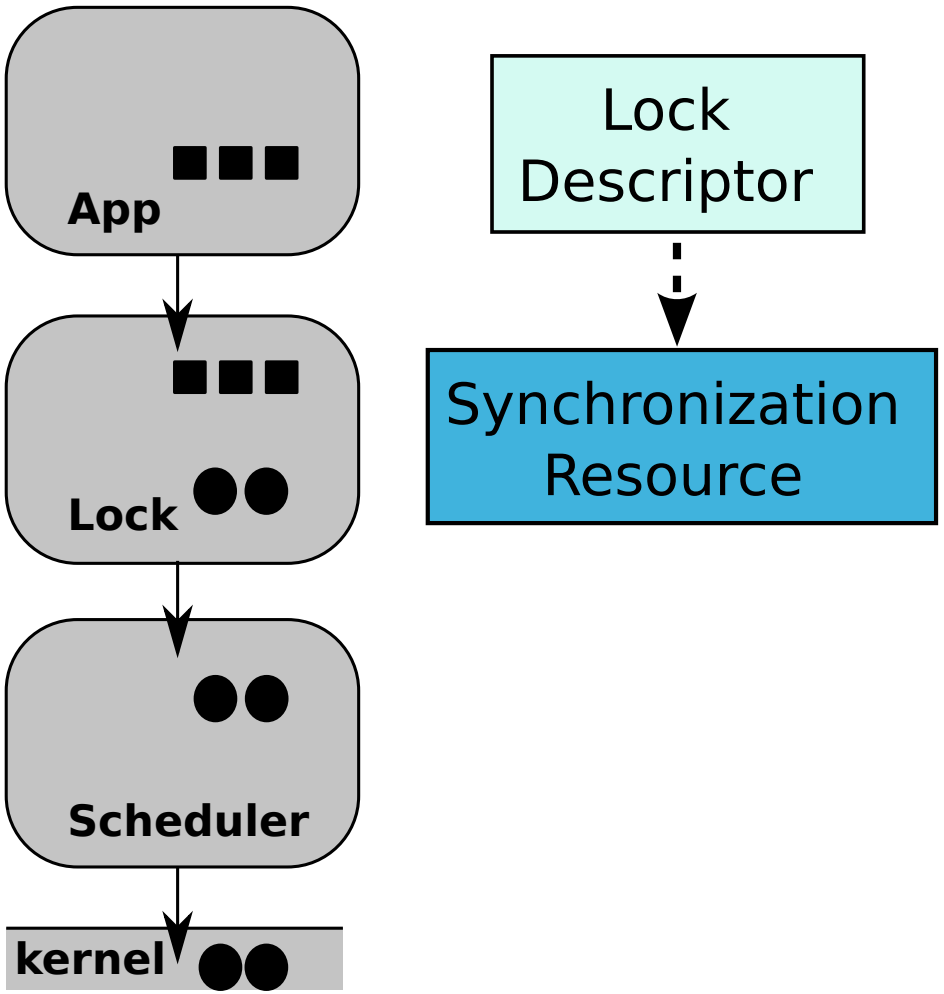


Descriptor

- accessed locally or globally
- meta data
- dependent descriptor

Resource

System Model – Descriptor and Resource



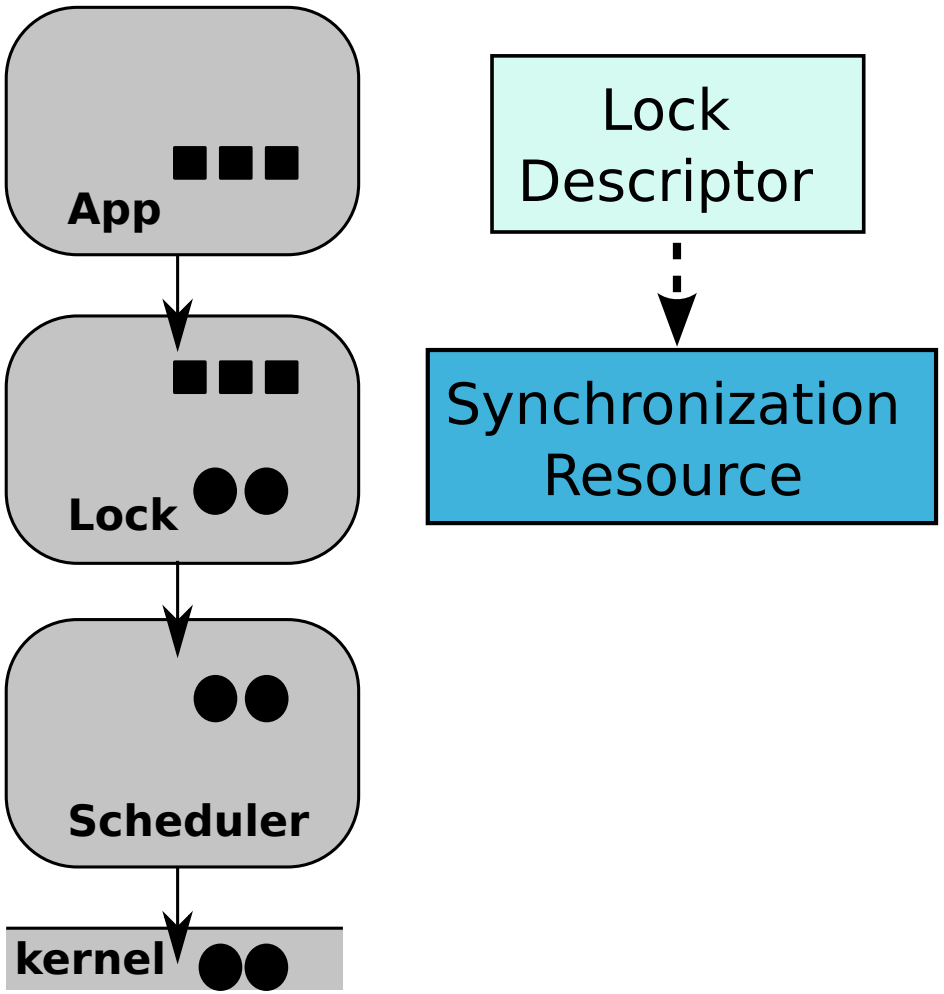
Descriptor

- accessed locally or globally
- meta data
- dependent descriptor

Resource

- thread can block

System Model – Descriptor and Resource



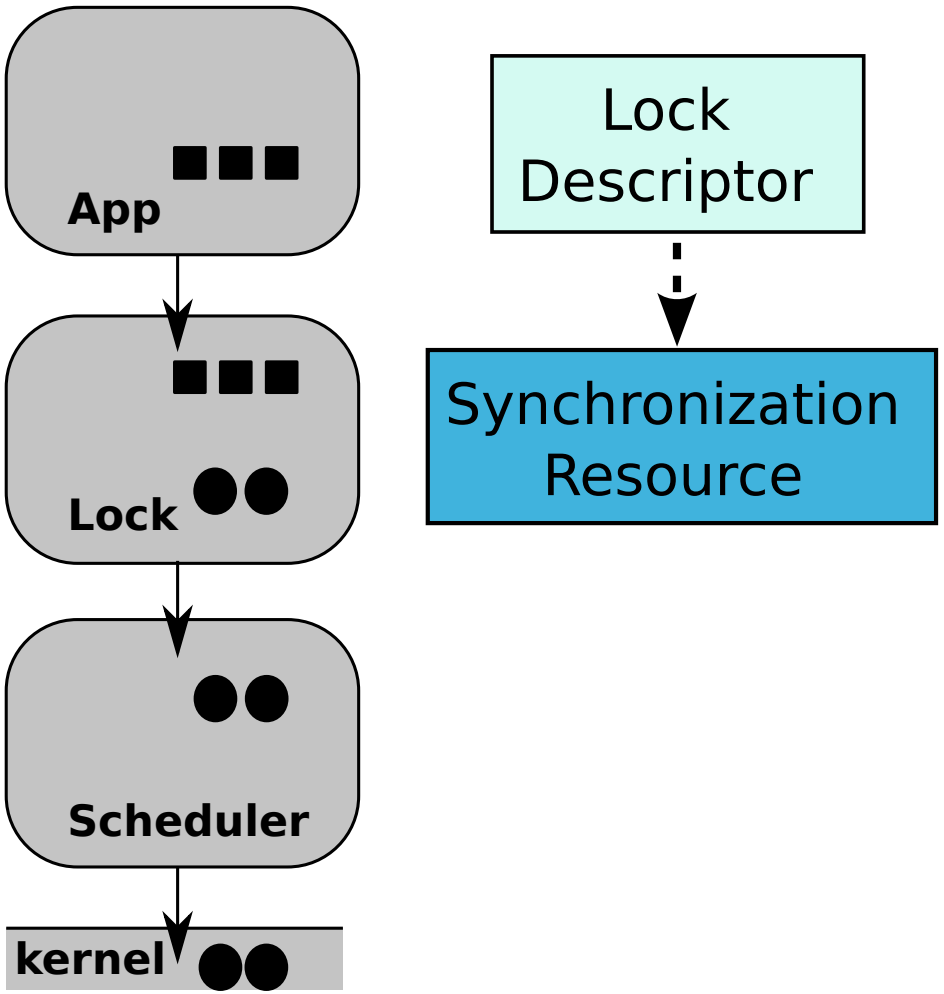
Descriptor

- accessed locally or globally
- meta data
- dependent descriptor

Resource

- thread can block
- meta data

System Model – Descriptor and Resource



Descriptor

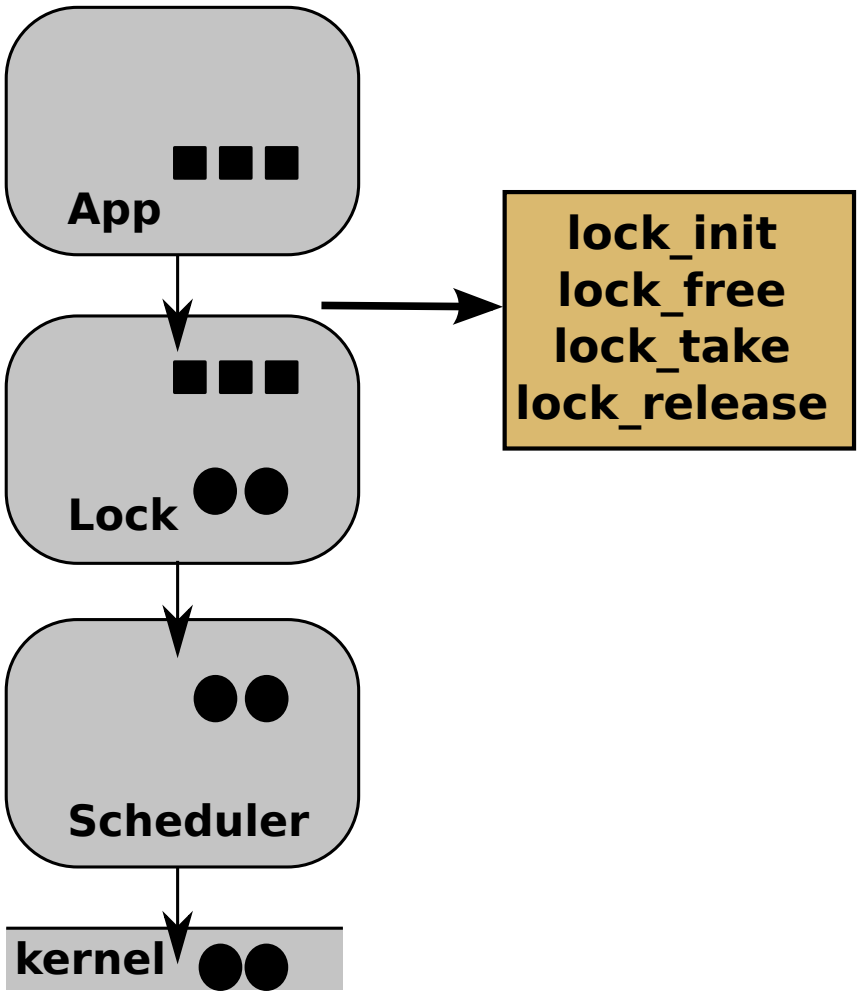
- accessed locally or globally
- meta data
- dependent descriptor

Resource

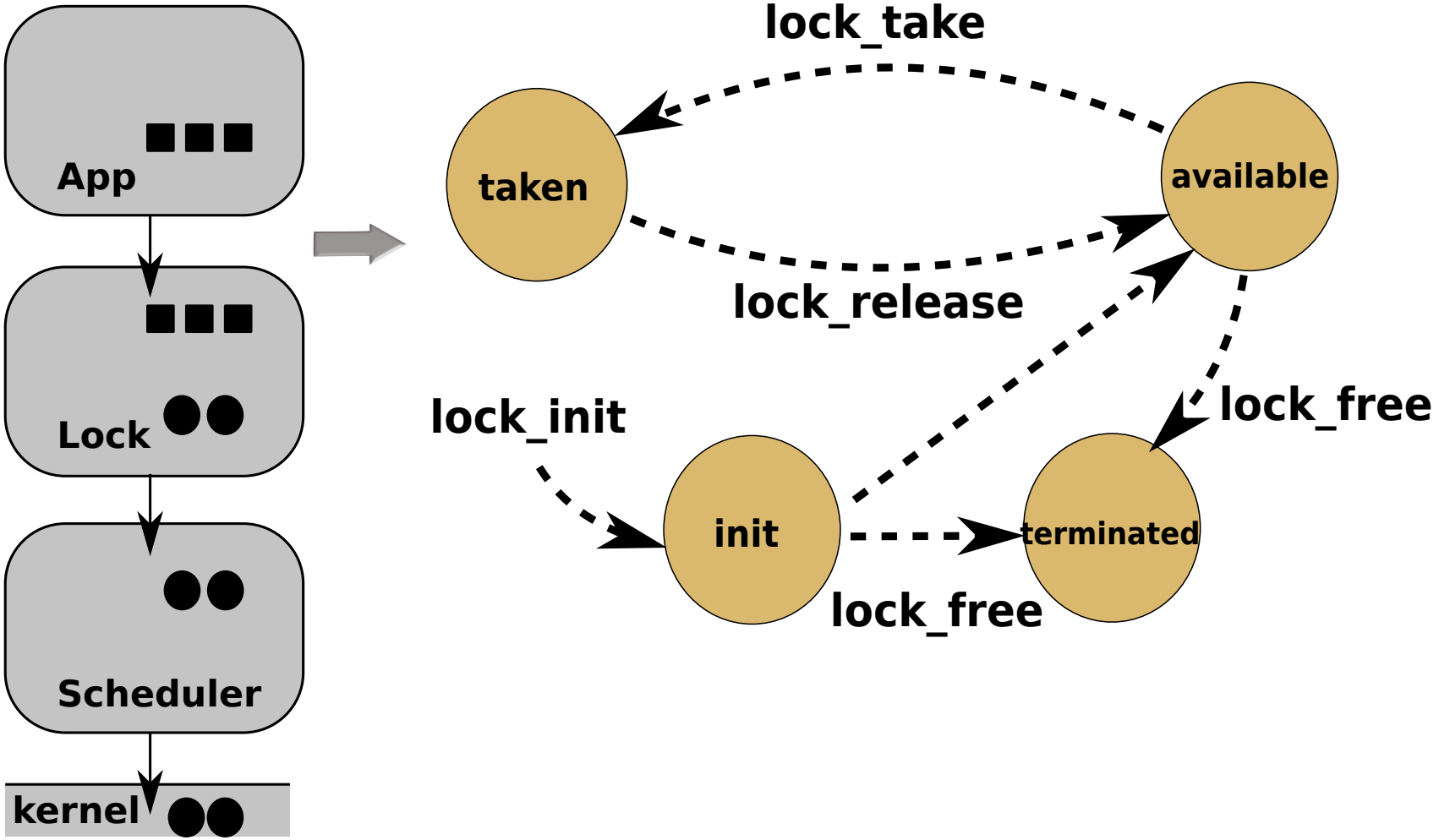
- thread can block
- meta data

lock, scheduler, memory manager, file system, event...

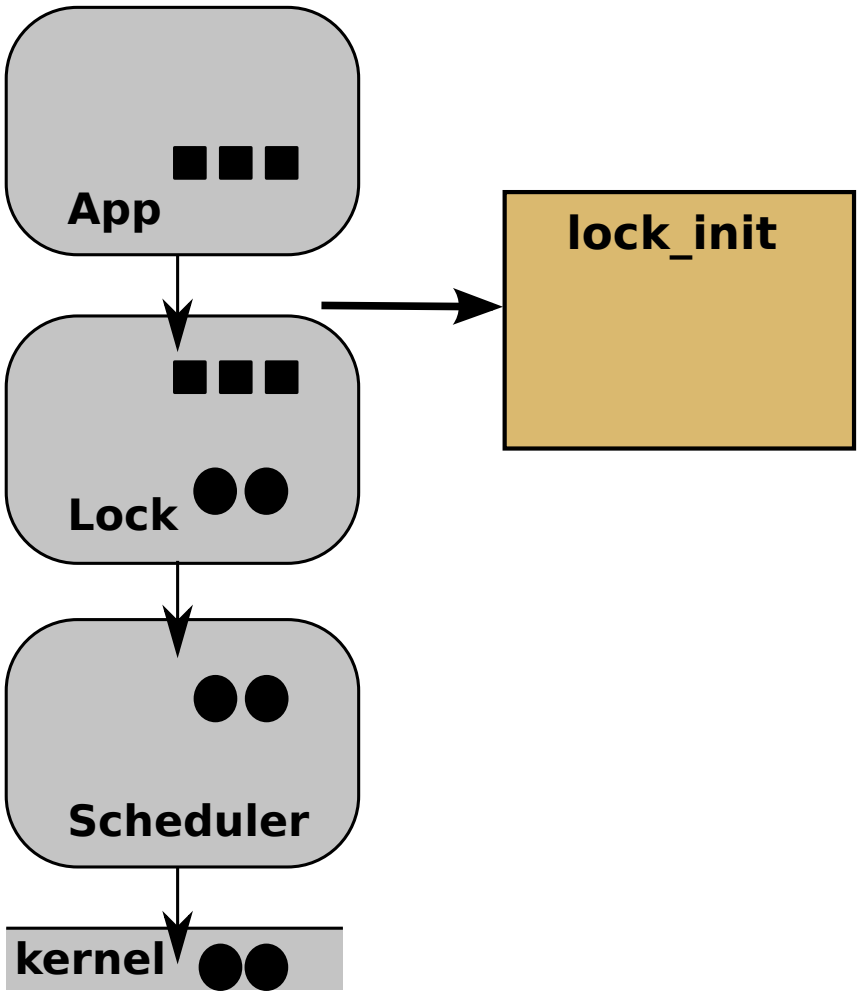
System Model – Component Interface Functions



System Model – Component Interface Functions



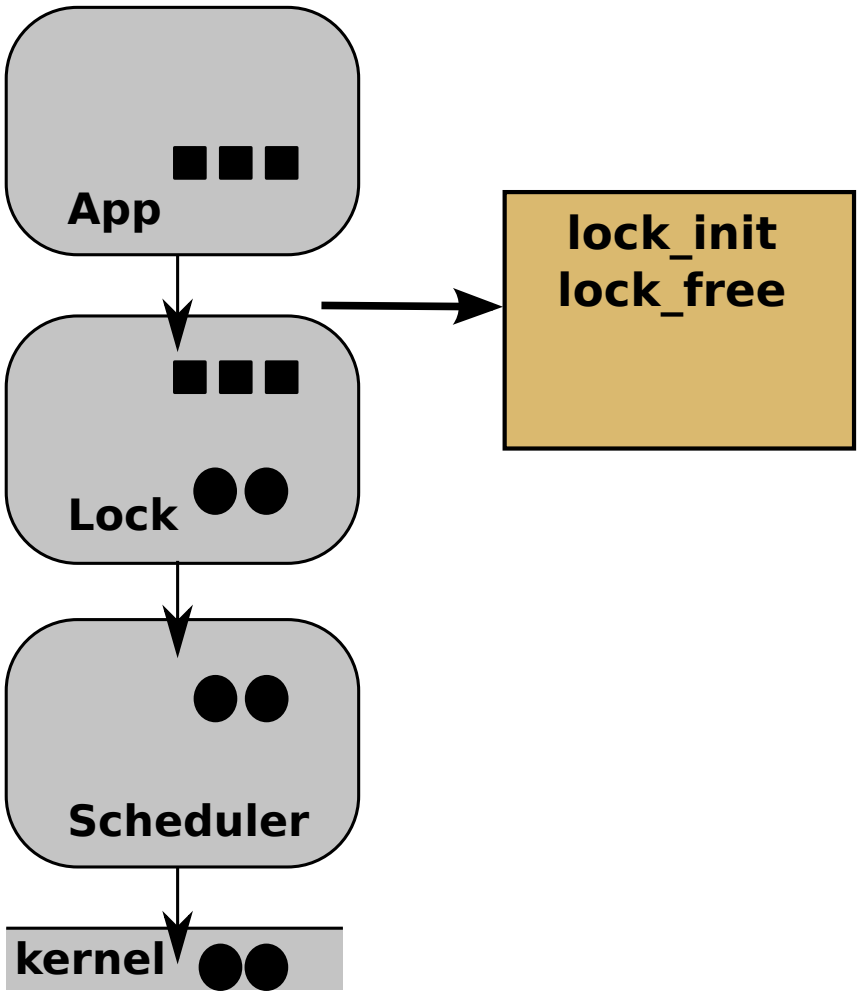
System Model – Component Interface Functions



Interface Function Types

- create descriptor

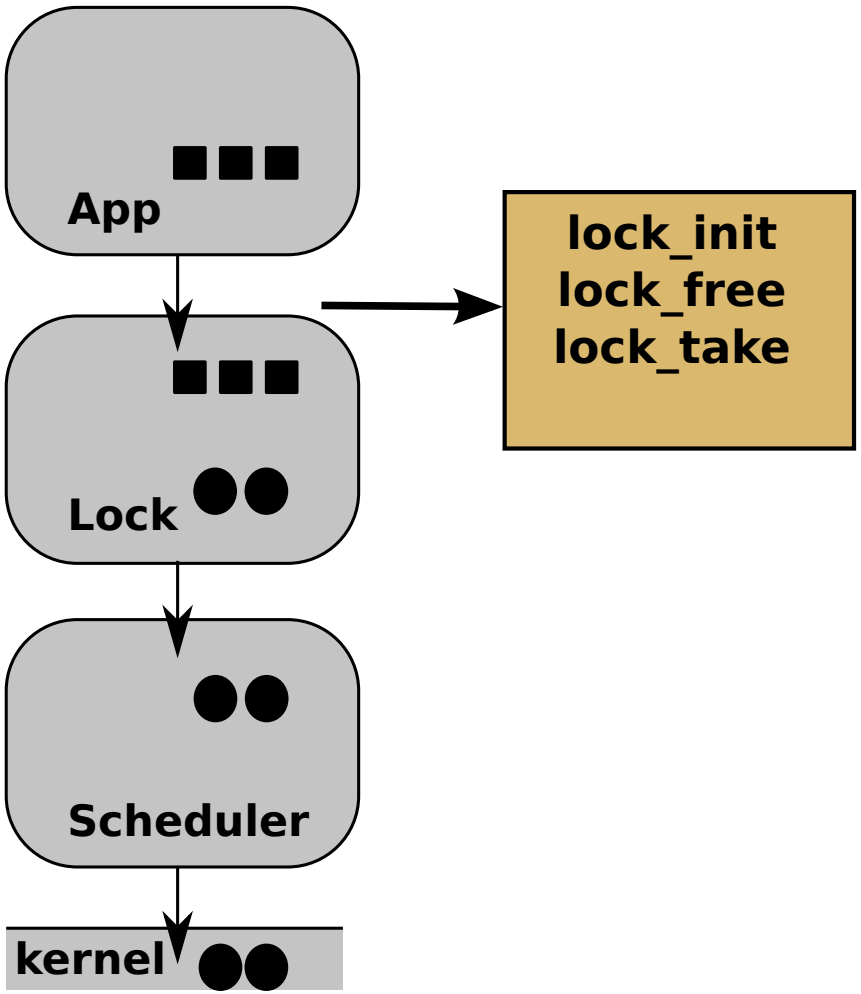
System Model – Component Interface Functions



Interface Function Types

- create descriptor
- terminate descriptor

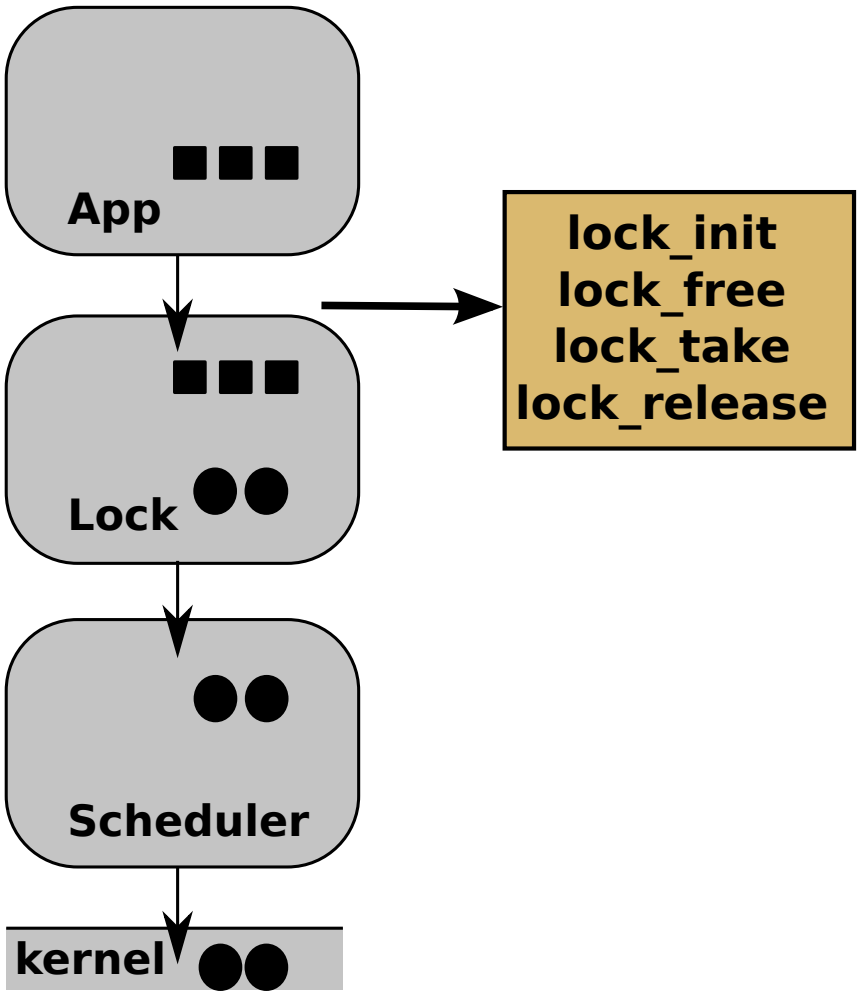
System Model – Component Interface Functions



Interface Function Types

- create descriptor
- terminate descriptor
- block invoking thread

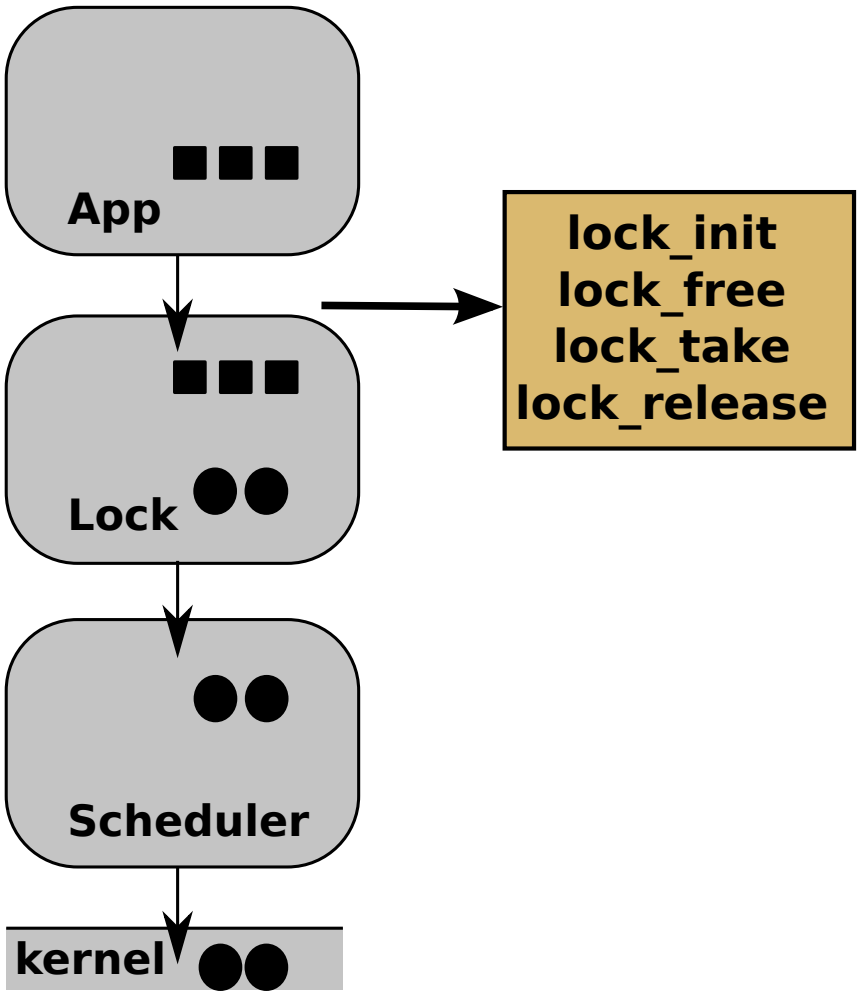
System Model – Component Interface Functions



Interface Function Types

- create descriptor
- terminate descriptor
- block invoking thread
- wake up thread

System Model – Component Interface Functions

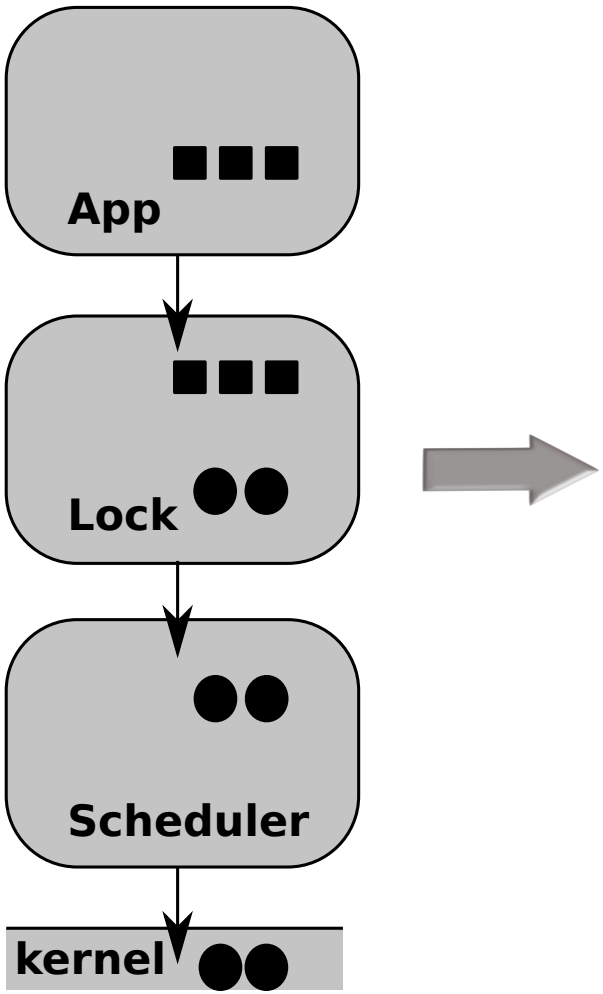


Interface Function Types

- create descriptor
- terminate descriptor
- block invoking thread
- wake up thread

lock, scheduler, memory manager, file system, event...

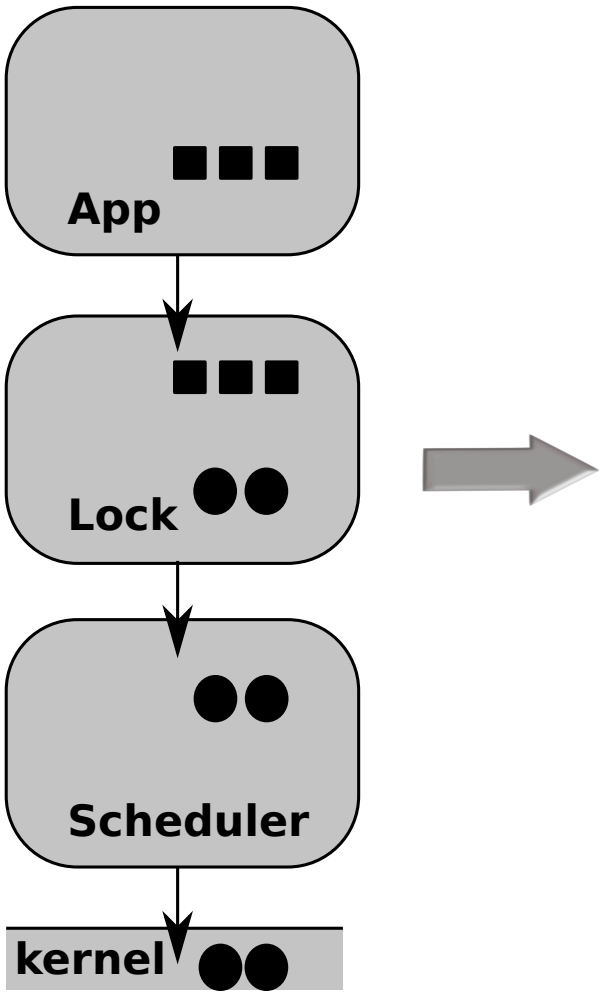
System Model in IDL-based Specification




```
service_global_info = {
        service                 = lock,

        desc_close_self_only    = true,
        desc_dep_create_none    = true,
        desc_remove_tracking    = true,
        desc_global             = false,
        desc_block              = true,
        desc_has_data           = false,
        resc_has_data           = false,
};

sm_creation(lock_component_alloc);
sm_transition(lock_component_alloc, lock_component_free);
sm_transition(lock_component_alloc, lock_component_pretake);
sm_transition(lock_component_pretake, lock_component_take);
sm_transition(lock_component_take, lock_component_release);
sm_transition(lock_component_release, lock_component_pretake);
sm_transition(lock_component_release, lock_component_free);
sm_terminal(lock_component_free);
sm_block(lock_component_take);
sm_wakeup(lock_component_release);

desc_data_retval(ul_t, lock_id)
lock_component_alloc(spdid_t desc_data(spdid));
int lock_component_take(spdid_t spdid, ul_t desc(lock_id),
                        u32_t desc_data(thd_id));
int lock_component_pretake(spdid_t spdid, ul_t desc(lock_id),
                           u32_t thd_id);
int lock_component_release(spdid_t spdid, ul_t desc(lock_id));
int lock_component_free(spdid_t spdid, ul_t desc_terminate(lock_id));
```

“descriptor and resource” + “interface functions”
$\rightarrow$ described using IDL

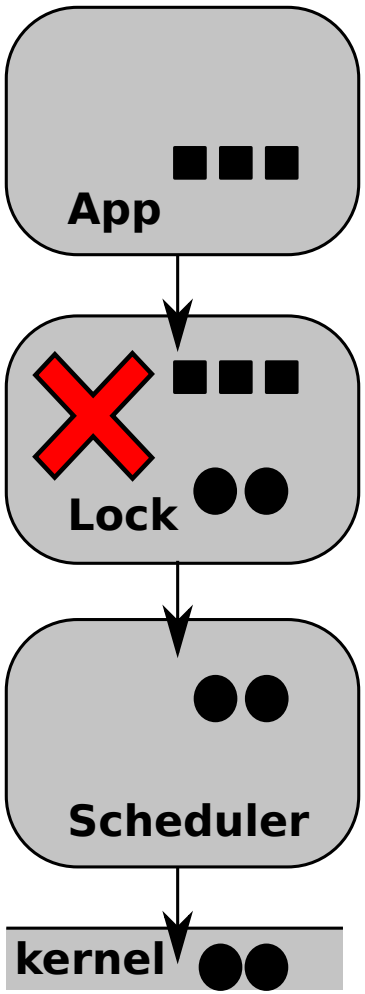
System Model in IDL-based Specification



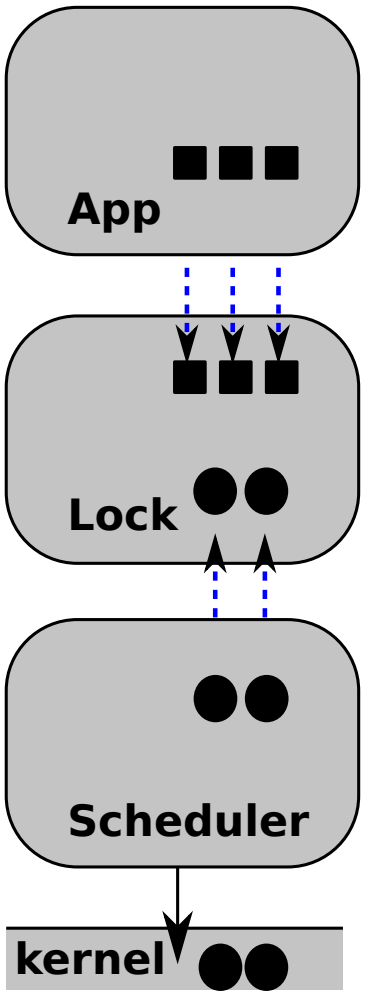

```
service_global_info = {
        service                 = lock,

        desc_close_self_only    = true,
        desc_dep_create_none    = true,
        desc_remove_tracking    = true,
        desc_global             = false,
        desc_block              = true,
        desc_has_data           = false,
        resc_has_data           = false,
};

sm_creation(lock_component_alloc);
sm_transition(lock_component_alloc, lock_component_free);
sm_transition(lock_component_alloc, lock_component_pretake);
sm_transition(lock_component_pretake, lock_component_take);
sm_transition(lock_component_take, lock_component_release);
sm_transition(lock_component_release, lock_component_pretake);
sm_transition(lock_component_release, lock_component_free);
sm_terminal(lock_component_free);
sm_block(lock_component_take);
sm_wakeup(lock_component_release);

desc_data_retval(ul_t, lock_id)
lock_component_alloc(spdid_t desc_data(spdid));
int lock_component_take(spdid_t spdid, ul_t desc(lock_id),
                        u32_t desc_data(thd_id));
int lock_component_pretake(spdid_t spdid, ul_t desc(lock_id),
                           u32_t thd_id);
int lock_component_release(spdid_t spdid, ul_t desc(lock_id));
int lock_component_free(spdid_t spdid, ul_t desc_terminate(lock_id));
```

"descriptor and resource" + "interface functions"
$\rightarrow$ described using IDL in average 37 LOC

Set of Recovery Mechanisms in C^3



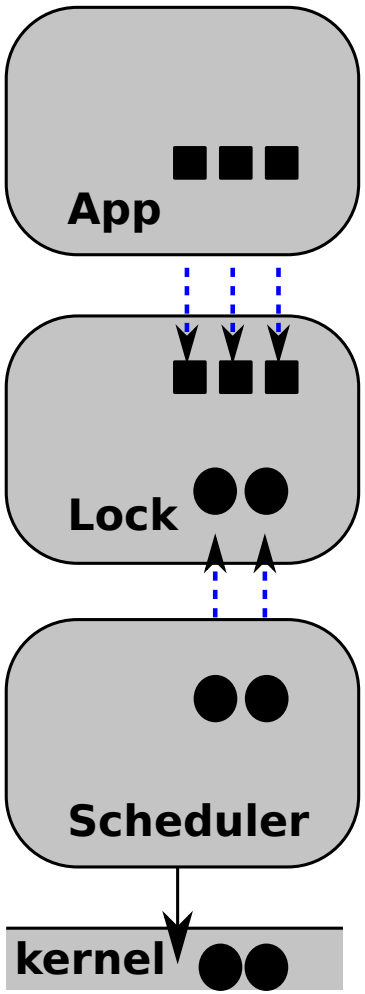
Set of Recovery Mechanisms in C^3



Basic Recovery

- always through component operation

Set of Recovery Mechanisms in C^3



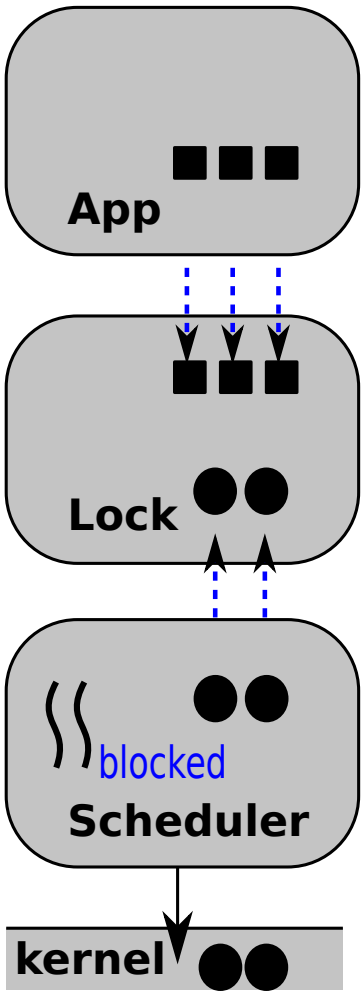
Basic Recovery

- always through component operation

Timing of Recovery

- on-demand: recover only when access

Set of Recovery Mechanisms in C^3



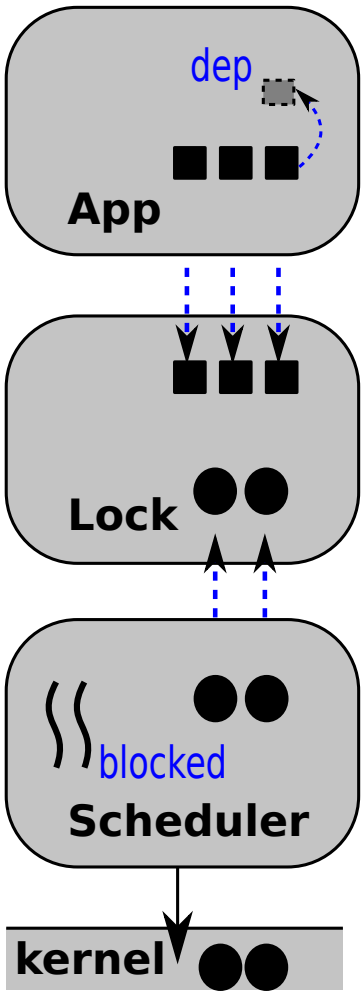
Basic Recovery

- always through component operation

Timing of Recovery

- on-demand: recover only when access
- eager: wake up all blocking threads

Set of Recovery Mechanisms in C^3



Basic Recovery

- always through component operation

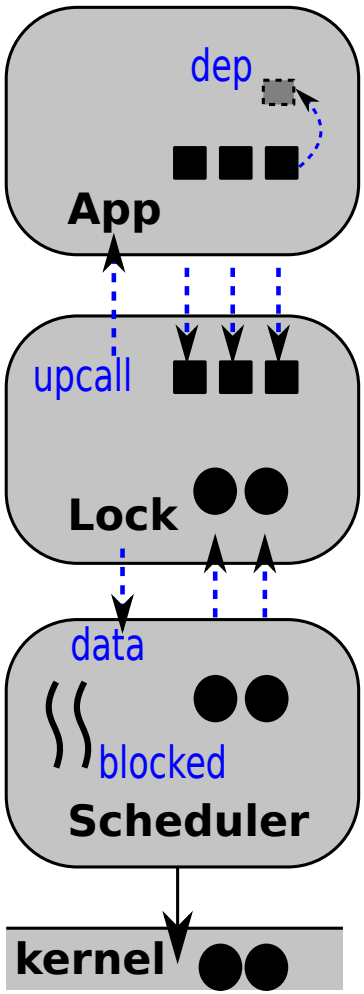
Timing of Recovery

- on-demand: recover only when access
- eager: wake up all blocking threads

Recovery with Dependency

- recover parent or children descriptor

Set of Recovery Mechanisms in C^3



Basic Recovery

- always through component operation

Timing of Recovery

- on-demand: recover only when access
- eager: wake up all blocking threads

Recovery with Dependency

- recover parent or children descriptor

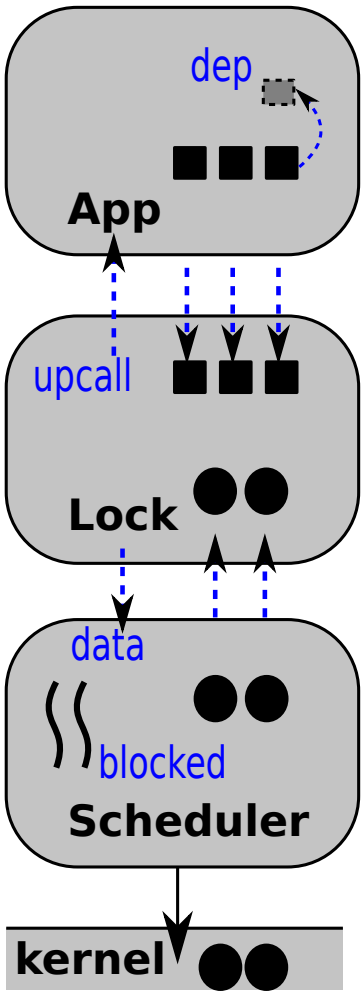
Recovery with Storage

- recover data

Recovery with Upcall

- upcall into client component for recovery

Set of Recovery Mechanisms in C^3



Basic Recovery

- always through component operation

Timing of Recovery

- on-demand: recover only when access
- eager: wake up all blocking threads

Recovery with Dependency

- recover parent or children descriptor

Recovery with Storage

- recover data

Recovery with Upcall

- upcall into client component for recovery

lock, scheduler, memory manager, file system, event...

System Model → Recovery Mechanisms

System Model in IDL

```
service_global_info = {
        service                 = lock,

        desc_close_self_only    = true,
        desc_dep_create_none    = true,
        desc_remove_tracking    = true,
        desc_global             = false,
        desc_block              = true,
        desc_has_data           = false,
        resc_has_data           = false,
};

sm_creation(lock_component_alloc);
sm_transition(lock_component_alloc, lock_component_free);
sm_transition(lock_component_alloc, lock_component_pretake);
sm_transition(lock_component_pretake, lock_component_take);
sm_transition(lock_component_take, lock_component_release);
sm_transition(lock_component_release, lock_component_pretake);
sm_transition(lock_component_release, lock_component_free);
sm_terminal(lock_component_free);
sm_block(lock_component_take);
sm_wakeup(lock_component_release);

desc_data_retval(ul_t, lock_id)
lock_component_alloc(spdid_t desc_data(spdid));
int lock_component_take(spdid_t spdid, ul_t desc(lock_id),
                        u32_t desc_data(thd_id));
int lock_component_pretake(spdid_t spdid, ul_t desc(lock_id),
                           u32_t thd_id);
int lock_component_release(spdid_t spdid, ul_t desc(lock_id));
int lock_component_free(spdid_t spdid, ul_t desc_terminate(lock_id));
```

Recovery Mechanisms

Basic Recovery

Timing of Recovery

Recovery with Dependency

Recovery with Storage

Recovery with Upcall

System Model → Recovery Mechanisms

System Model in IDL

```
service_global_info = {
        service                 = lock,

        desc_close_self_only    = true,

};

sm_c
sm_t
sm_transition(lock_component_alloc, lock_component_pretake);
sm_transition(lock_component_pretake, lock_component_take);
sm_transition(lock_component_take, lock_component_release);
sm_transition(lock_component_release, lock_component_pretake);
sm_transition(lock_component_release, lock_component_free);
sm_terminal(lock_component_free);
sm_block(lock_component_take);
sm_wakeup(lock_component_release);

desc_data_retval(ul_t, lock_id)
lock_component_alloc(spdid_t desc_data(spdid));
int lock_component_take(spdid_t spdid, ul_t desc(lock_id),
                        u32_t desc_data(thd_id));
int lock_component_pretake(spdid_t spdid, ul_t desc(lock_id),
                           u32_t thd_id);
int lock_component_release(spdid_t spdid, ul_t desc(lock_id));
int lock_component_free(spdid_t spdid, ul_t desc_terminate(lock_id));
```

Recovery Mechanisms

Basic Recovery

which recovery mechanisms to use? how to generate the recovery code?

Recovery with Dependency

Recovery with Storage

Recovery with Upcall

System Model in IDL

```
service_global_info = {
        service                 = lock,

        desc_close_self_only    = true,

};

sm_c
sm_t
sm_t
sm_t
sm_t
sm_t
sm_t
sm_t
sm_t
sm_b
sm_w

desc_d
lock_component_alloc(spdid_t desc_data(spdid));
int lock_component_take(spdid_t spdid, ul_t desc(lock_id),
                        u32_t desc_data(thd_id));
int lock_component_pretake(spdid_t spdid, ul_t desc(lock_id),
                           u32_t thd_id);
int lock_component_release(spdid_t spdid, ul_t desc(lock_id));
int lock_component_free(spdid_t spdid, ul_t desc_terminate(lock_id));
```

Recovery Mechanisms

Basic Recovery

which recovery mechanisms to use? how to generate the recovery code?

Idea: evaluate predicates and generate the code from a set of chosen code templates

ncy

Recovery with Upcall

Example – Client Invocation Stub

- Predicate ⇒ *"true"*
- Code Template ⇒

```
/* predicate: true */
CSTUB_FN(IDL_fntype, IDL_fname) (IDL_parsdecl) {
  long fault = 0;
  int  ret   = 0;
redo:
  cli_if_desc_update_IDL_fname(IDL_params);

  ret = cli_if_invoke_IDL_fname(IDL_params);
  if (fault){
    CSTUB_FAULT_UPDATE();
    if (cli_if_desc_update_post_fault_IDL_fname()) goto
        redo;
  }
  ret = cli_if_track_IDL_fname(ret, IDL_params);
  return ret;
}
```

Example – Client Invocation Stub

- Predicate ⇒ *"true"*
- Code Template ⇒

```
/* predicate: true */
CSTUB_FN(IDL_fntype, IDL_fname) (IDL_parsdecl) {
  long fault = 0;
  int  ret   = 0;
redo:
  cli_if_desc_update_IDL_fname(IDL_params);

  ret = cli_if_invoke_IDL_fname(IDL_params);
  if (fault){
    CSTUB_FAULT_UPDATE();
    if (cli_if_desc_update_post_fault_IDL_fname()) goto
        redo;
  }
  ret = cli_if_track_IDL_fname(ret, IDL_params);
  return ret;
}
```

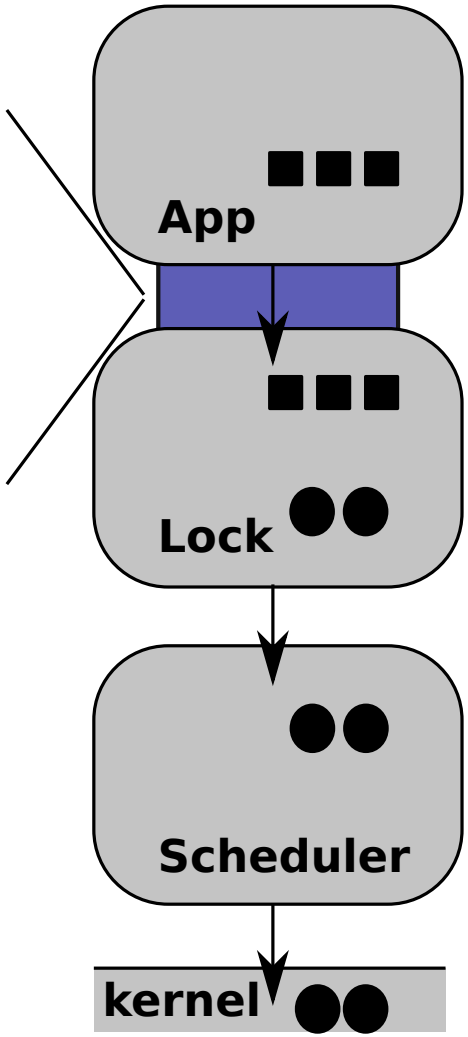
Example – Client Stub Descriptor State Tracking

- Predicate $\Rightarrow$ *"the interface function can create descriptor and the descriptor is not accessed between components"*
- Code Template $\Rightarrow$

```
/* predicate: f ∈ I_{d_r}^{create} ∧ ¬G_{d_r} */
static inline int cli_if_track_IDL_fname(int ret,
                                         IDL_parsdecl) {
  if (ret == -EINVAL) return ret;

  struct desc_track *desc = call_desc_alloc();
  if (!desc) return -ENOMEM;
  call_desc_track(desc, ret, IDL_params);

  return desc->IDL_id;
}
```

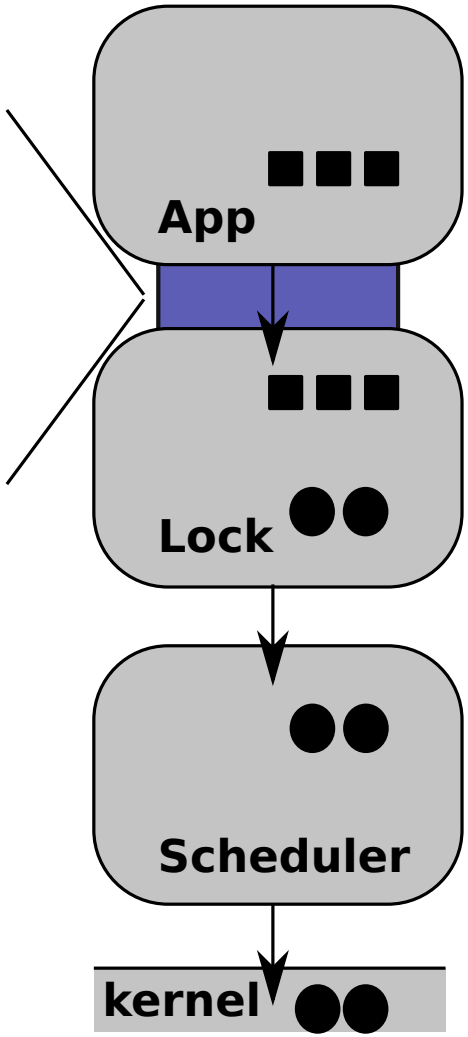
Example – Generated Recovery Code Snippet

```
        desc->server_lock_id = new_desc->server_lock_id;
        desc->state = state_lock_component_alloc;
        desc->fault_cnt = global_fault_cnt;

        call_desc_dealloc(new_desc);
        block_cli_if_recover_data(desc);
}

static inline struct desc_track *call_desc_update(ul_t id, int next_state)
{
        struct desc_track *desc = NULL;
        unsigned int from_state = 0;
        unsigned int to_state = 0;

        if (id == 0)
                return NULL;

        desc = call_desc_lookup(id);
        if (unlikely(!desc))
                goto done;

        desc->next_state = next_state;

        if (likely(desc->fault_cnt == global_fault_cnt))
                goto done;
        desc->fault_cnt = global_fault_cnt;

 done:
        return desc;
}

CSTUB_FN(int, lock_component_take)(struct usr_inv_cap * uc, spdid_t spdid,
                                   ul_t lock_id, u32_t thd_id) {
        long fault = 0;
        int ret = 0;

        call_map_init();

 redo:
        block_cli_if_desc_update_lock_component_take(spdid, lock_id, thd_id);

        ret =
            block_cli_if_invoke_lock_component_take(spdid, lock_id, thd_id, ret,
                                                    &fault, uc);
        if (unlikely(fault)) {

                CSTUB_FAULT_UPDATE();
                if (block_cli_if_desc_update_post_fault_lock_component_take()) {
                        goto redo;
                }
```



Example – Generated Recovery Code Snippet

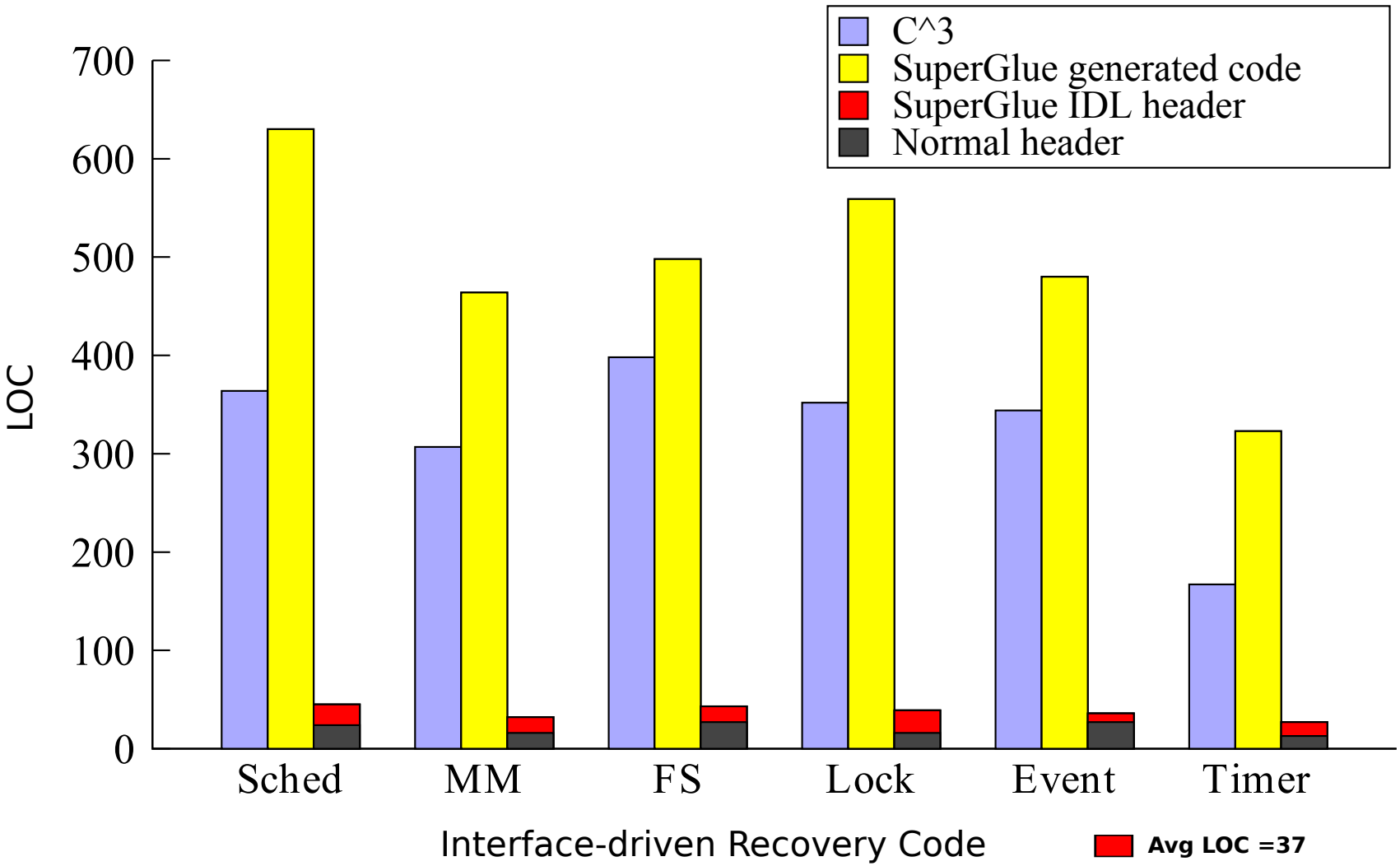
```
        desc->server_lock_id = new_desc->server_lock_id;
        desc->state = state_lock_component_alloc;
        desc->fault_cnt = global_fault_cnt;

        call_desc_dealloc(new_desc);
        block_cli_if_recover_data(desc);
}

static inline struct desc_track *call_desc_update(ul_t id, int next_state)
{
        struct desc_track *desc = NULL;
        unsigned int from_state = 0;
        unsigned int to_state = 0;

        if (id == 0)
                return NULL;

        desc = call_desc_lookup(id);
        if (unlikely(!desc))
                goto done;

        desc->next_state = next_state;

        if (likely(desc->fault_cnt == global_fault_cnt))
                goto done;
        desc->fault_cnt = global_fault_cnt;

 done:
        return desc;
}

CSTUB_FN(int, lock_component_take)(struct usr_inv_cap * uc, spdid_t spdid,
                                   ul_t lock_id, u32_t thd_id) {
        long fault = 0;
        int ret = 0;

        call_map_init();

 redo:
        block_cli_if_desc_update_lock_component_take(spdid, lock_id, thd_id);

        ret =
            block_cli_if_invoke_lock_component_take(spdid, lock_id, thd_id, ret,
                                                    &fault, uc);
        if (unlikely(fault)) {

                CSTUB_FAULT_UPDATE();
                if (block_cli_if_desc_update_post_fault_lock_component_take()) {
                        goto redo;
                }
```



lock, scheduler, memory manager, file system, event...

Outlines

Code Generation Result



Fault Injection Campaign Results

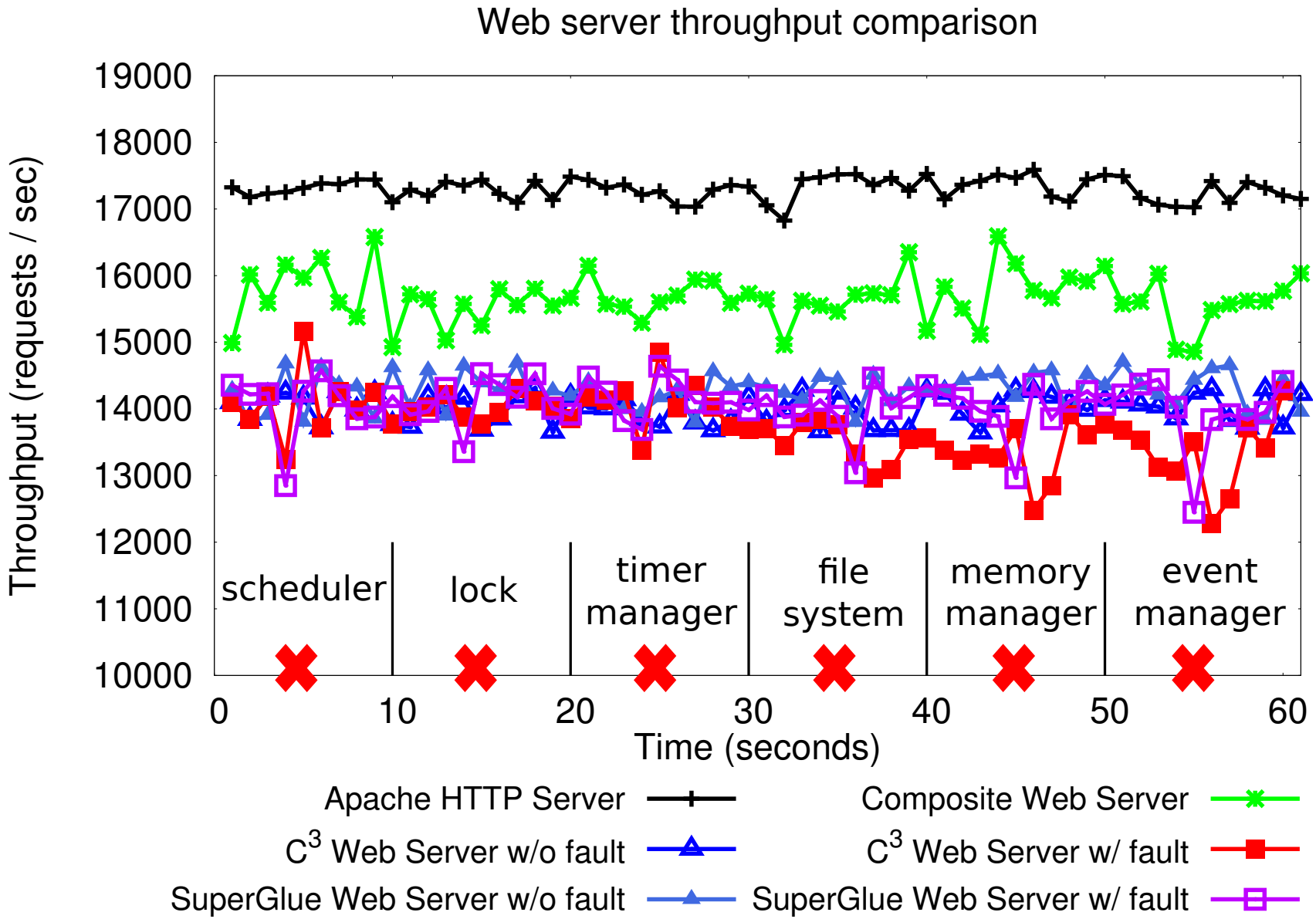
- SuperGlue implemented in *Composite* component-based OS
  - Intel i7-2760QM, running at 2.4 Ghz
- Transient faults modeled as random register bit-flips (SWIFI)

System Component	Injected	Fault Activation Ratio	Recovery Success Rate
Sched	500	98.36%	88.58%
MM	500	94.26%	91.48%
FS	500	94.7%	96.14%
Lock	500	93.82%	92.35%
Event	500	93.83%	96%
Timer	500	97.23%	94.62%

∗ fault activation ratio = activated faults/injected faults

∗ recovery success ratio = recovered faults/activated faults

Web Server Evaluation with Injected Faults



Outlines

Conclusion

SuperGlue – system-level fault tolerance code generation

- synthesizes recovery code from the high-level system model
- effectively eases the programmer burden
- efficient and predictable system-level fault recovery

Thanks

? || /* */

composite.seas.gwu.edu

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC