

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

Jiguo Song, Gedare Bloom, Gabriel Palmer  
Presented by **Teo Georgiev**

June 30, 2016

Computer Science Department  
The George Washington University

# Outlines

1 Motivation and Challenges

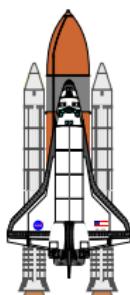
2 System-Level Fault Recovery

3 SuperGlue

4 Evaluation

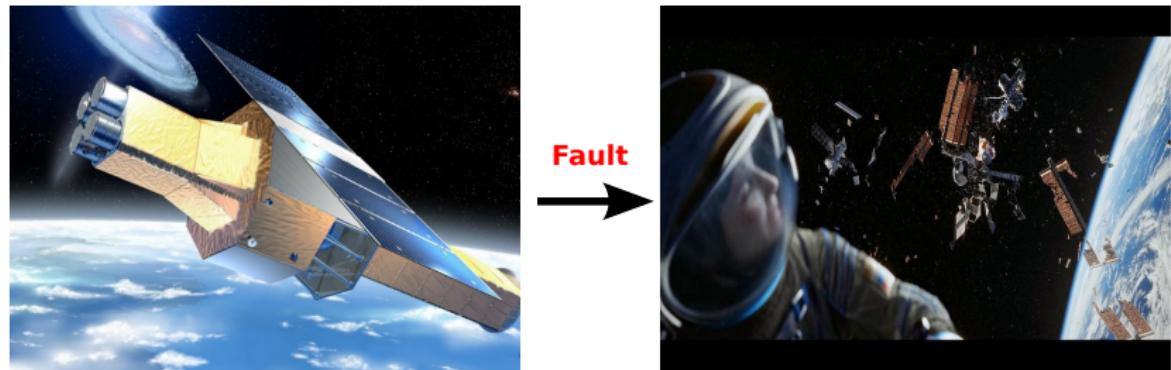
5 Conclusion

# Real-Time and Embedded Systems



# Consequences of Embedded System Faults

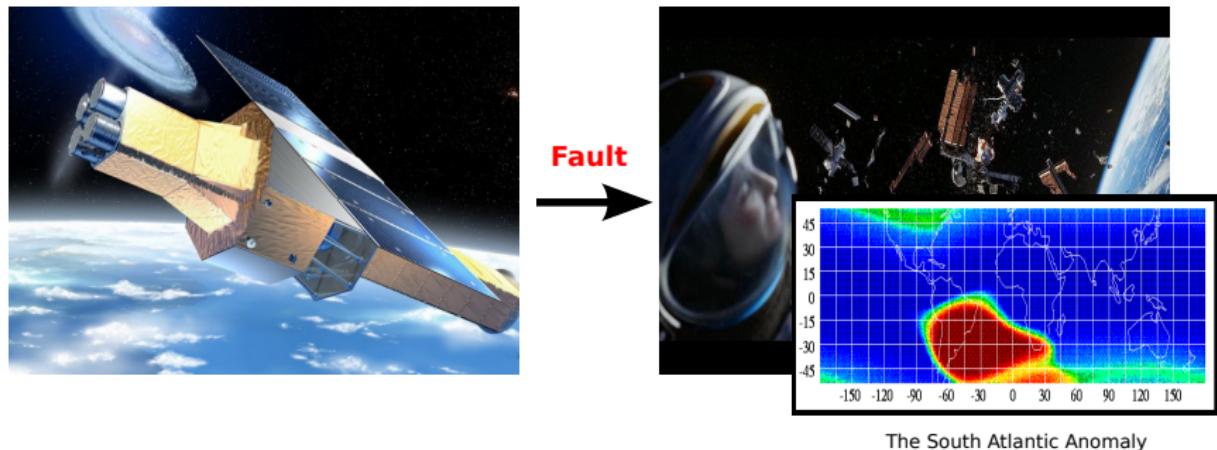
Japanese X-ray astronomy satellite Hitomi lost in 2016



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

# Consequences of Embedded System Faults

Japanese X-ray astronomy satellite Hitomi lost in 2016



- Financial losses of \$286 million USD
- “*It’s a scientific tragedy*” - Richard Mushotzky, UMD

# Consequences of embedded system faults

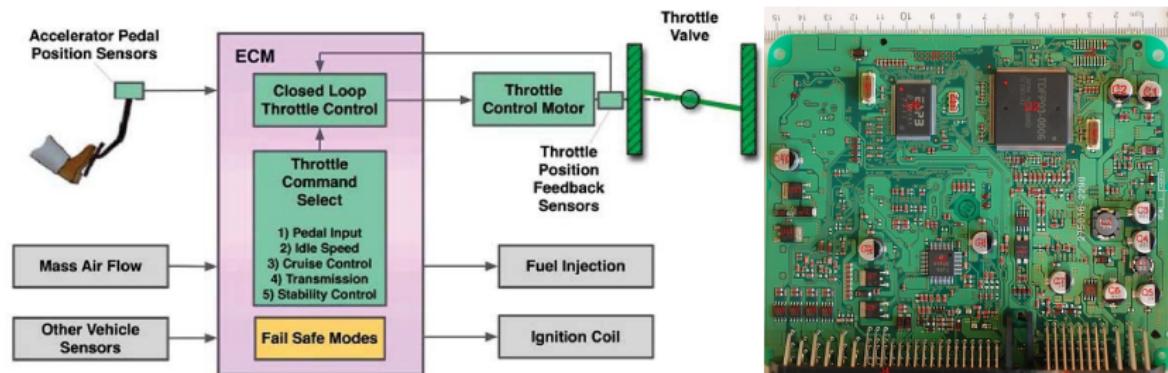
## Toyota Sudden Unintended Acceleration (SUA) in 2004 – 2010



- 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

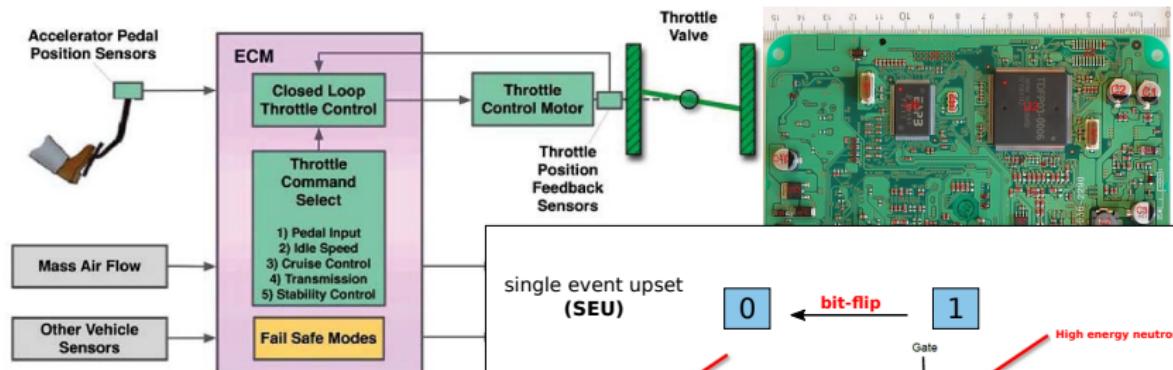


## Uncontrolled acceleration in Toyota Camrys

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

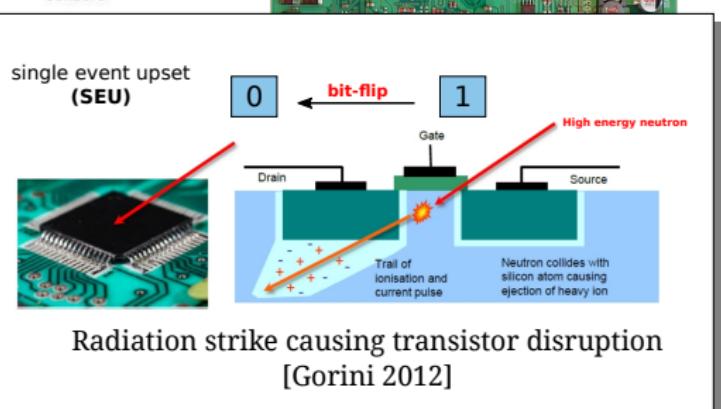
*“a bit-flip there, will have the effect of killing one of the tasks”*  
– Bookout v Toyota

# Sudden Unintended Acceleration



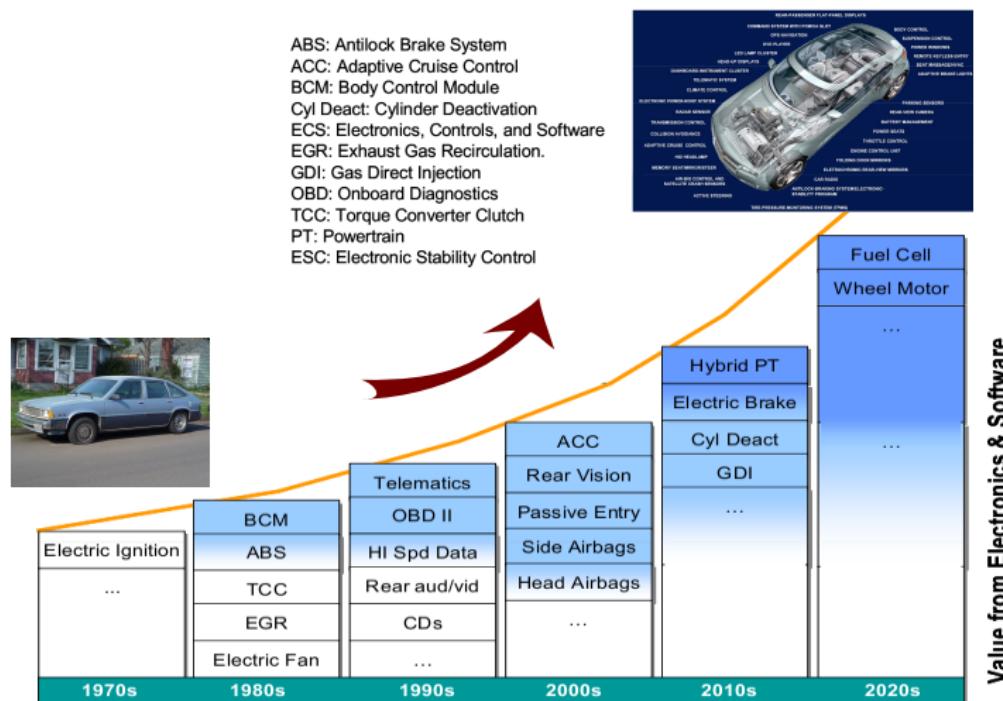
Uncontrolled acceleration in

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



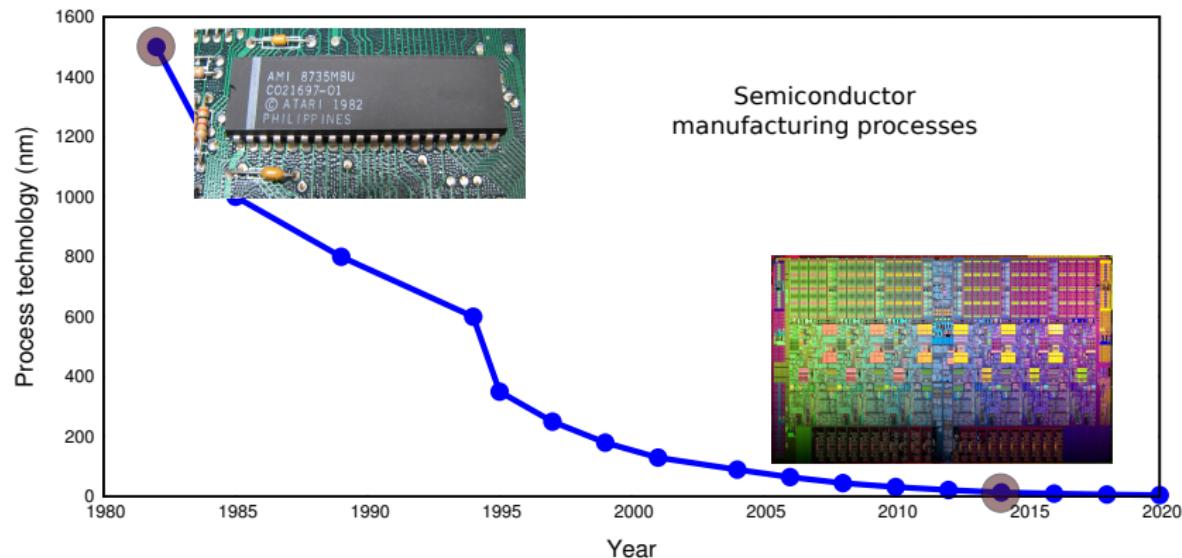
*“a bit-flip there, will have the effect of killing one of the tasks”*  
– Bookout v Toyota

# Embedded faults: Bad Now, Worse Tomorrow



- + more functionality
- more complexity → dependability more challenging

# Embedded faults: Bad Now, Worse Tomorrow



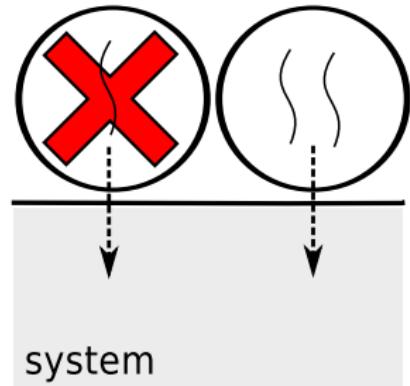
Decreasing process sizes → 5nm

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

# Application-Level Fault Tolerance

## Application fault tolerance

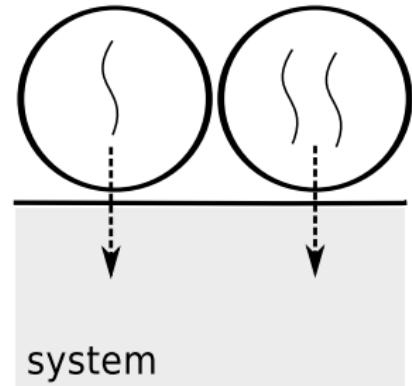
- example recovery techniques
  - recovery blocks
  - checkpointing
  - re-fork
- temporal redundancy
  - detect fault by job completion
  - replay execution from saved state



# Application-Level Fault Tolerance

## Application fault tolerance

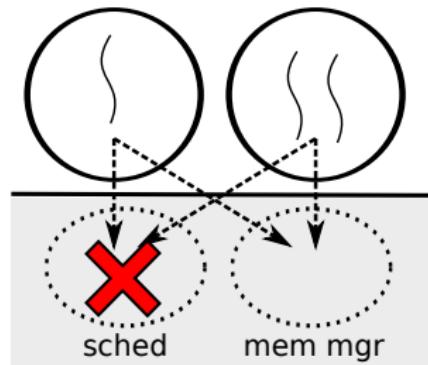
- example recovery techniques
  - recovery blocks
  - checkpointing
  - re-fork
- temporal redundancy
  - detect fault by job completion
  - replay execution from saved state



# System-Level Fault Tolerance

## System-level fault tolerance

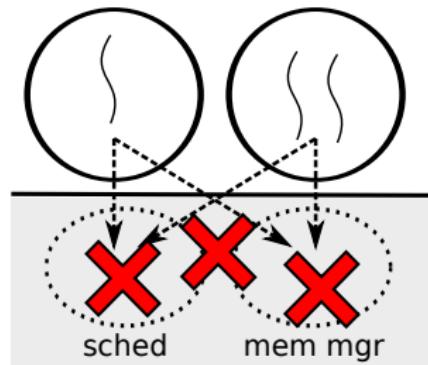
- failures in
  - scheduler
  - memory mapping manager
  - file-systems
  - synchronization manager
  - ...



# System-Level Fault Tolerance

## System-level fault tolerance

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



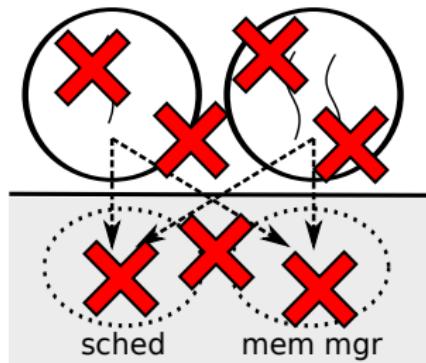
System components contain state for all tasks

- 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 recover scheduler
- memory...to recover memory 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



Recovery requires resources

- processing time...to recover scheduler
- memory...to recover memory manager

# Outlines

1 Motivation and Challenges

2 System-Level Fault Recovery

3 SuperGlue

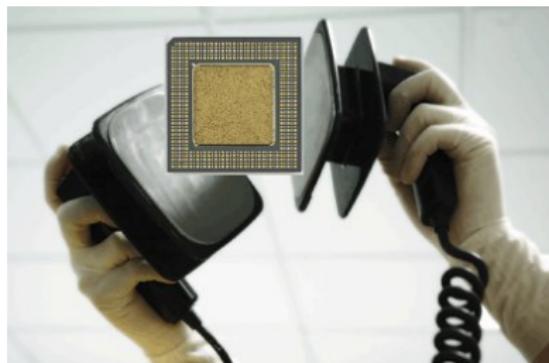
4 Evaluation

5 Conclusion

# $C^3$ : The Computational Crash Cart

## $C^3$ : Computational Crash Cart

- resuscitate system
- from system-level faults
- predictably



### Main ideas

- pervasive **fault isolation** → restrict propagation
- efficient  $\mu$ -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. Palmer, “Predictable, efficient system-level fault tolerance in  $C^3$ ” in RTSS, 2013

# Composite: A Component-Based OS

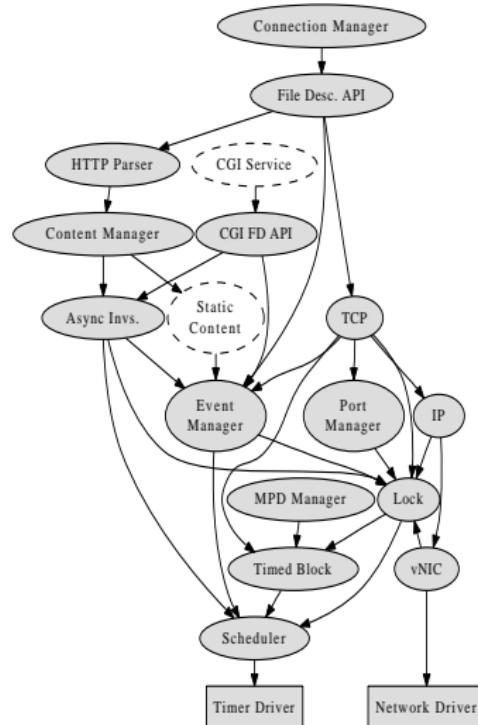
System functionality as *components*

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

Low-level services are components

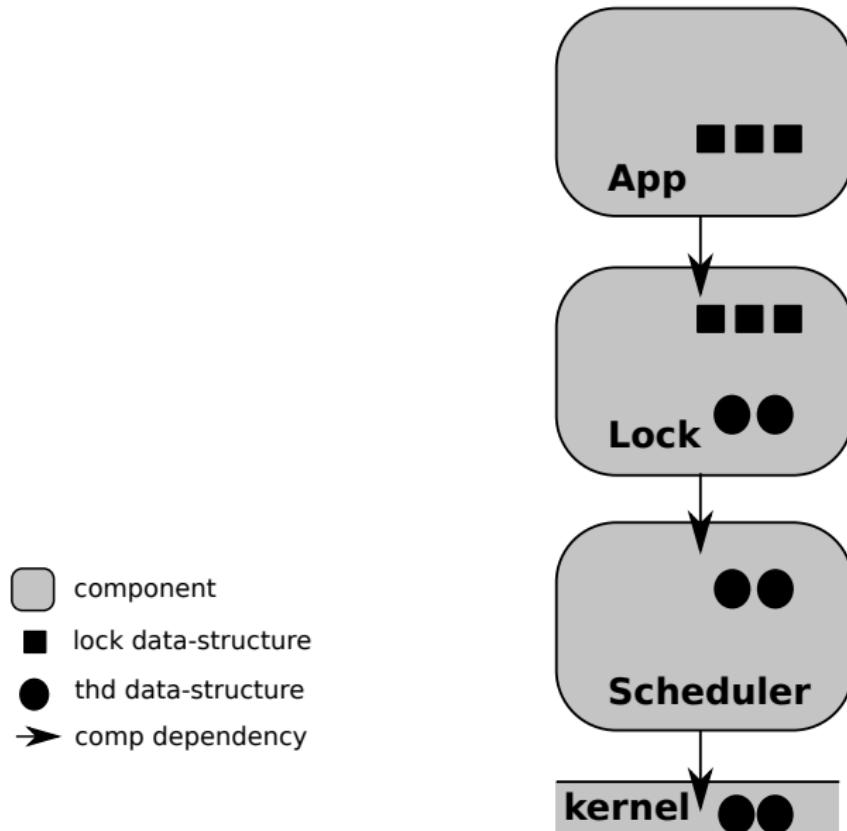
- kernel is small and policy-less
- scheduling, memory mapping, synchronization...are in user level

Components interact via *invocation* of interface functions



Example component graph

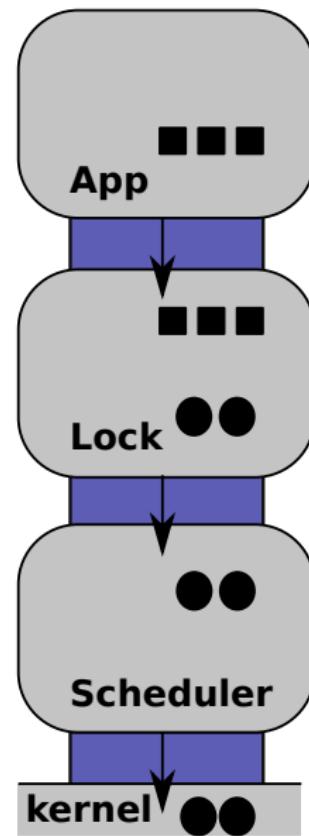
# C<sup>3</sup> System-Level Fault Recovery



# $C^3$ System-Level Fault Recovery

Recovery sequence in  $C^3$

- [grey square] component
- [black square] lock data-structure
- [black circle] thd data-structure
- [arrow] comp dependency
- [blue rectangle]  $C^3$  interface code

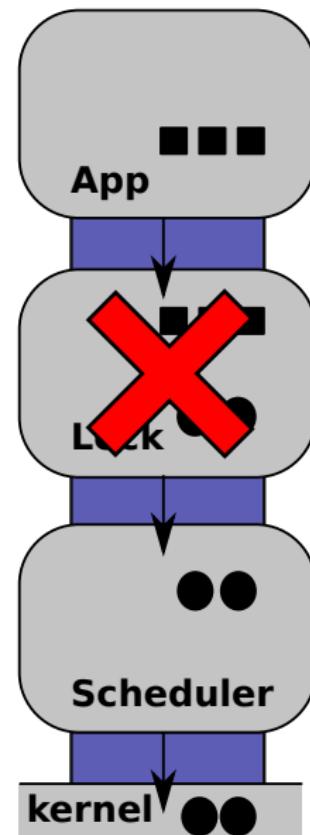


# $C^3$ System-Level Fault Recovery

## Recovery sequence in $C^3$

### 1 fault detection

- component
- lock data-structure
- thd data-structure
- comp dependency
- $C^3$  interface code

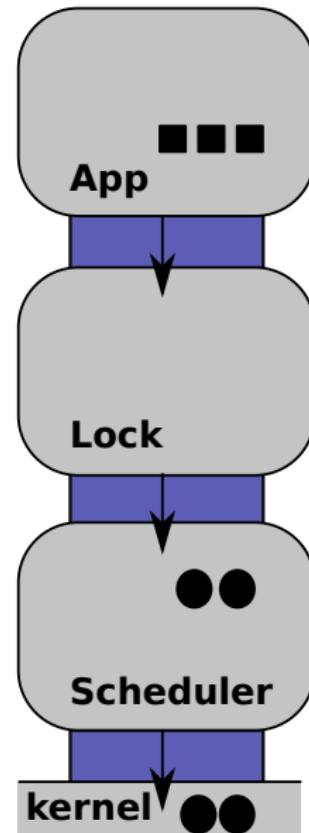


# $C^3$ System-Level Fault Recovery

## Recovery sequence in $C^3$

- 1 fault detection
- 2 safe-state recovery
  - $\mu$ -reboot

- component
- lock data-structure
- thd data-structure
- comp dependency
- $C^3$  interface code

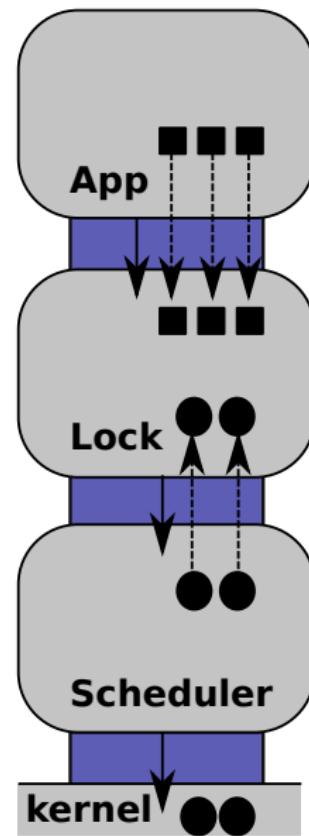


# $C^3$ System-Level Fault Recovery

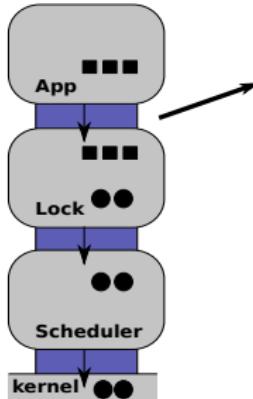
## Recovery sequence in $C^3$

- 1 fault detection
- 2 safe-state recovery
  - $\mu$ -reboot
- 3 consistent-state recovery
  - object state recovery

- component
- lock data-structure
- thd data-structure
- comp dependency
- C<sup>3</sup> interface code



# C<sup>3</sup> Implementation – Writing Code Manually



```
rd->state      = state;
rd->font      = global_fault_cst;

return;

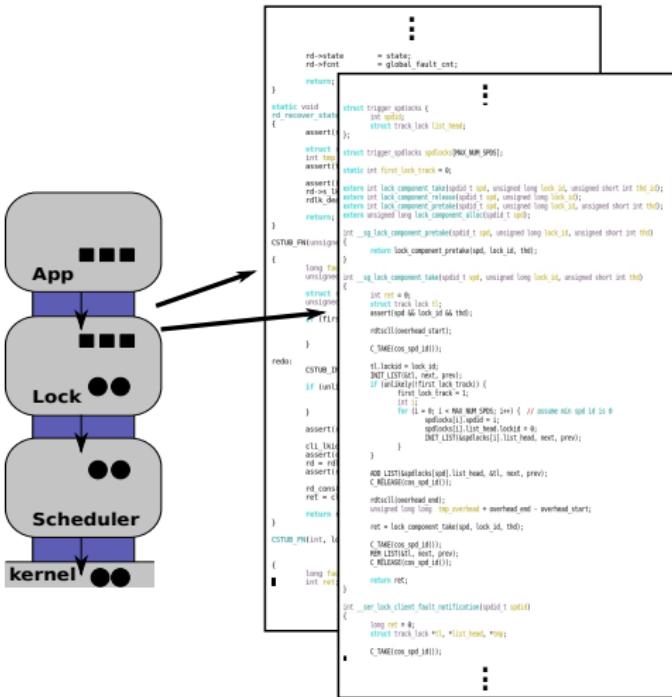
static void
rd_recover_state(struct rec_data_lk *rd)
{
    assert(rd == rd->rc_lk);
    struct rec_data_lk *tmp;
    int tmp_kid = lock_component_alloc(cos_spd_id());
    assert(tmp_kid);

    assert(tmp == rdk.lockup(tmp_kid));
    rd->rc_lk = tmp;
    rdk.dealloc(tmp_kid);
    return;
}

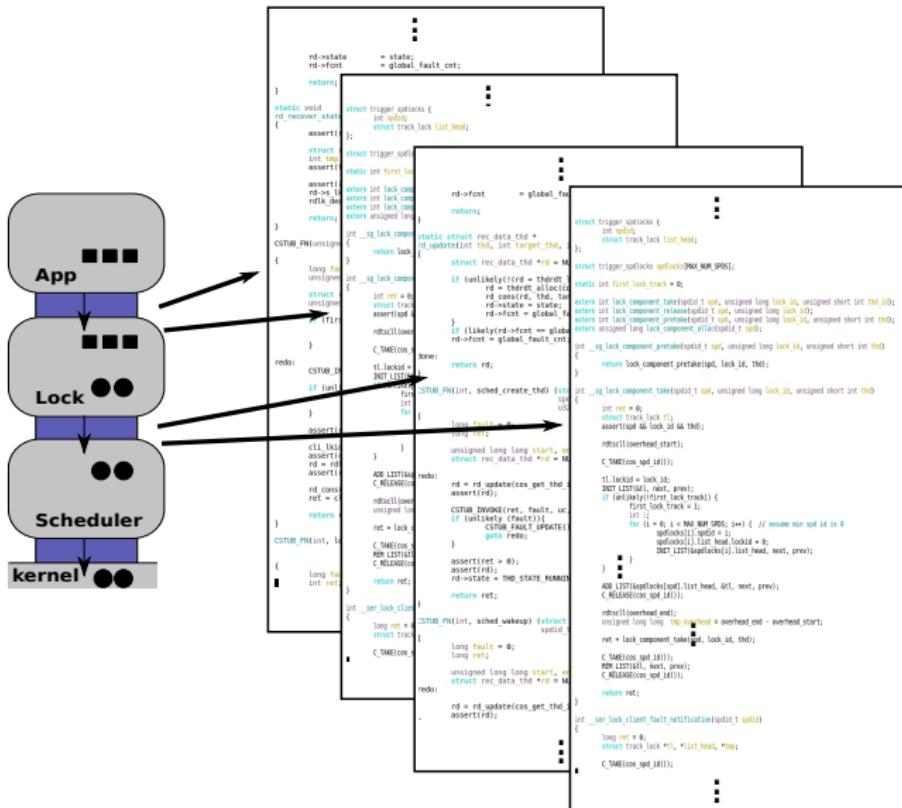
CSTUB_PtUnaligned long lock_component_alloc() {struct user_inx_cap *sc,
    spidid_t spidid;
{
    long fault;
    unsigned long ret;
    struct rec_data_lk *rd = NULL;
    unsigned long ser_lkid, cll_lkid;
    if (rlfirst == 0) {
        rd = init_static(64*16_lkids);
        rlfirst = 1;
    }
    redo:
    CSTUB_DWOKER(ret, fault, ut, lk, spidid);
    if (unlikely(fault)) {
        CSTUB_FAULT_UPDATE();
        goto redo;
    }
    assert(ret > 0);
    cll_kid = rdk.alloc();
    assert(ccll_kid == lk);
    rd = rdk.alloc(ccll_kid);
    assert(rd);
    rd.consrid(cos_spd_id(), cll_lkid, ret, LOCK_ALLOC);
    ret = cll_kid;
    return ret;
}

CSTUB_PtInt, lock_component_pretake) {struct user_inx_cap *sc,
    spidid_t spidid, unsigned long lock_id,
    unsigned short int tsd}
{
    long fault = 0;
    int ret;
```

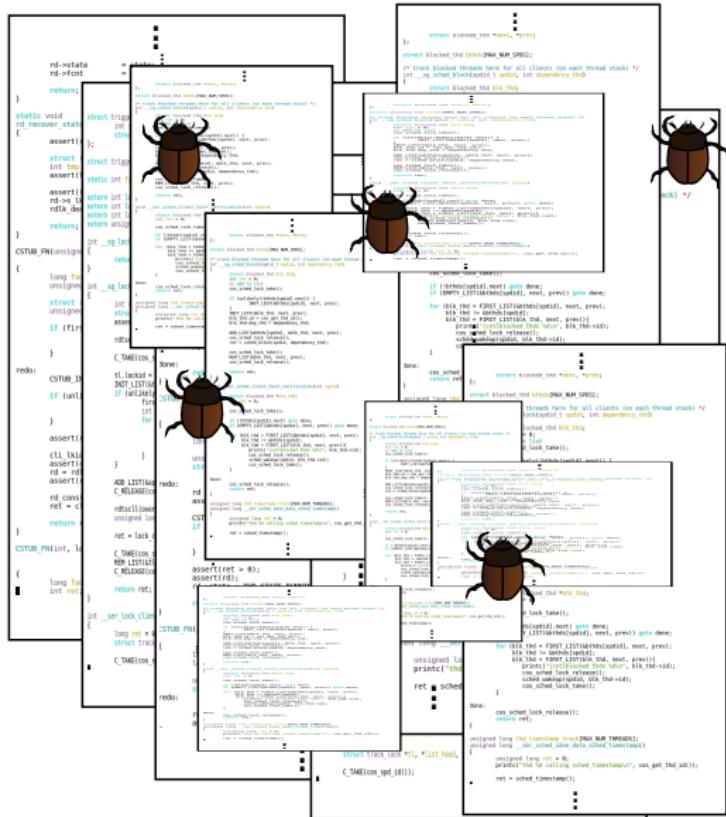
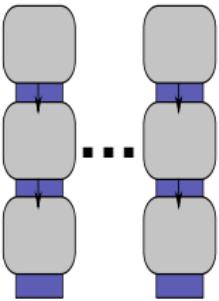
# C<sup>3</sup> Implementation – Writing Code Manually



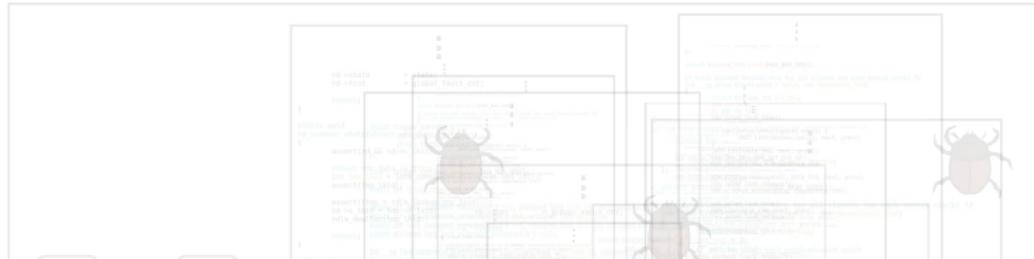
# C<sup>3</sup> Implementation – Writing Code Manually



# C<sup>3</sup> Implementation – Writing Code Manually



# C<sup>3</sup> Implementation – Writing Code Manually



Manually writing is ad-hoc and error-prone

Goal → automatically generate C<sup>3</sup>-style recovery code



# Outlines

1 Motivation and Challenges

2 System-Level Fault Recovery

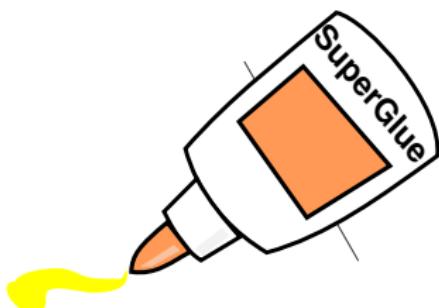
3 SuperGlue

4 Evaluation

5 Conclusion

## SuperGlue

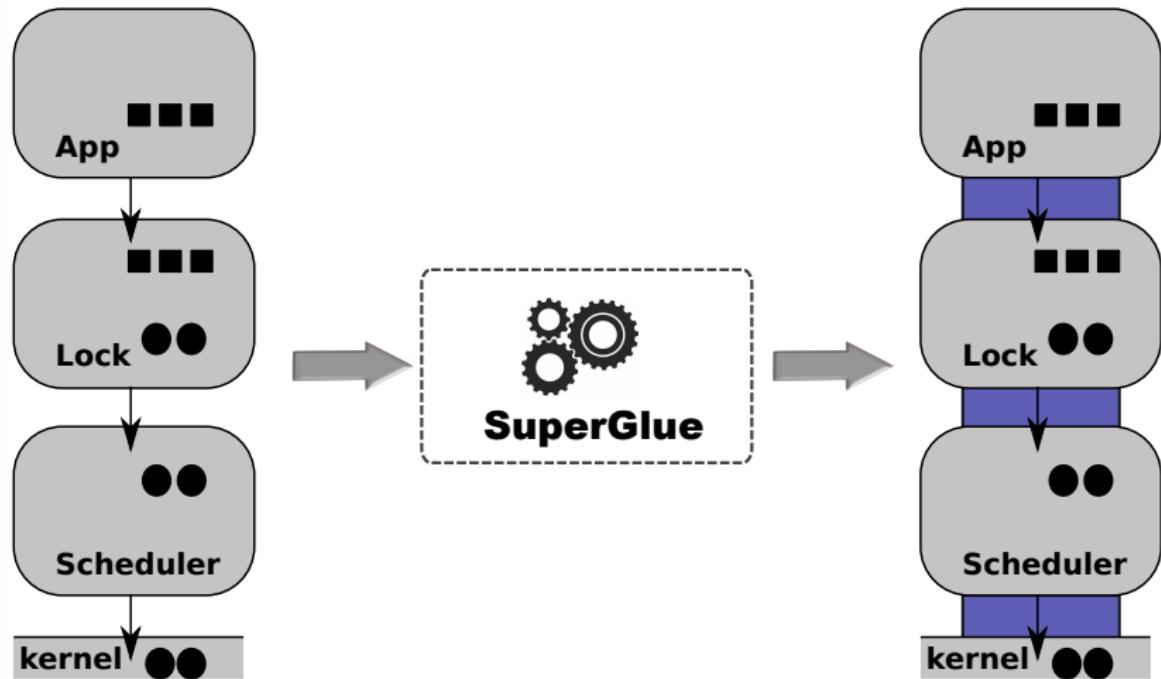
- automatically generate C<sup>3</sup> fault recovery code
- for all system-level services
- a system **model** and **interface** description language (IDL)
- an **IDL compiler** synthesizes recovery code from the model



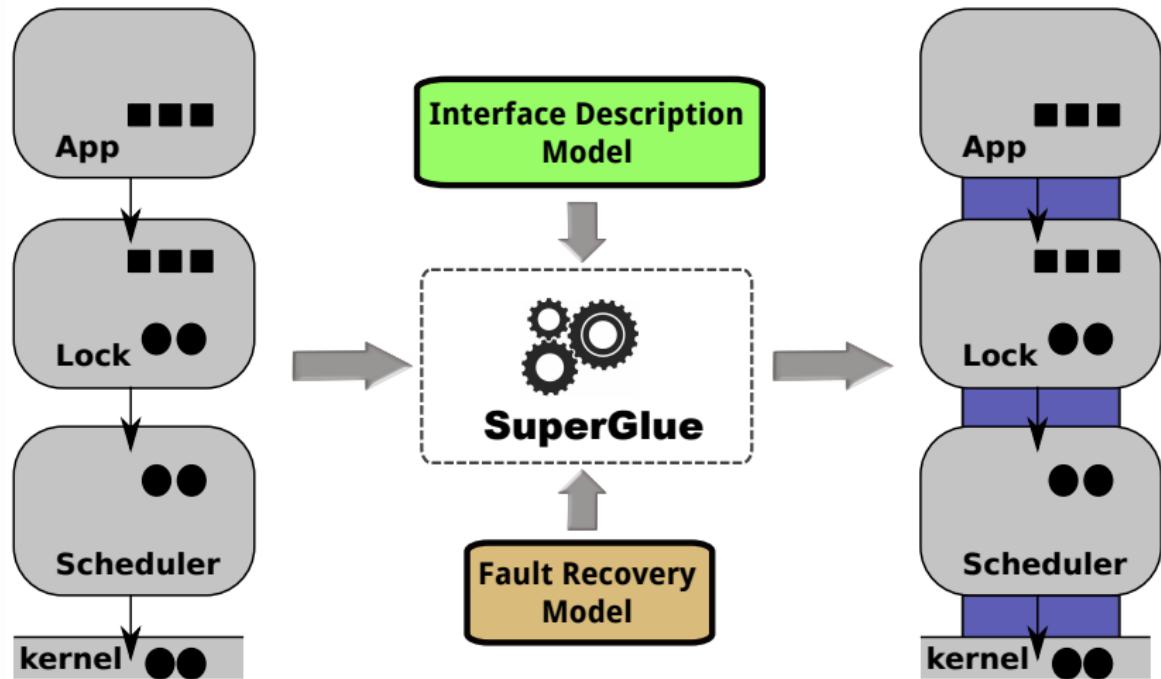
Fault Model (same as C<sup>3</sup>):

- on-chip single-event upset (SEUs)
- fault is detected immediately (*fail-stop*)

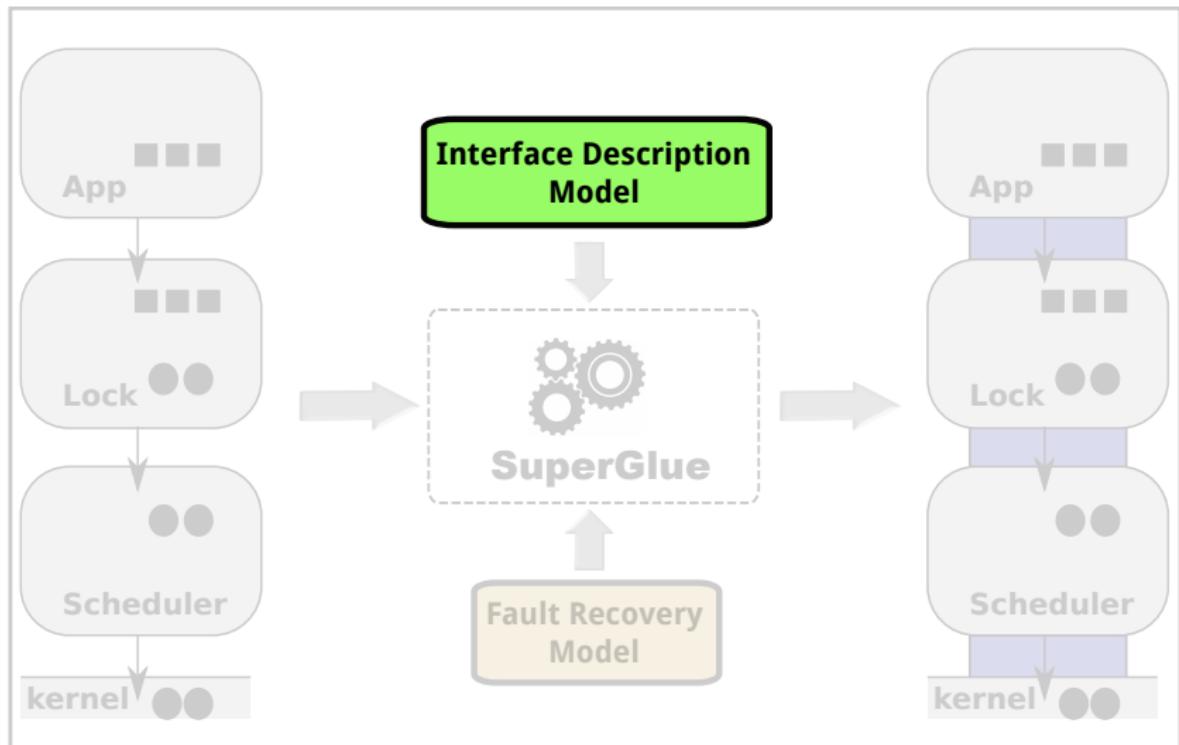
# SuperGlue – IDL-based System-Level Fault Tolerance



# SuperGlue – IDL-based System-Level Fault Tolerance

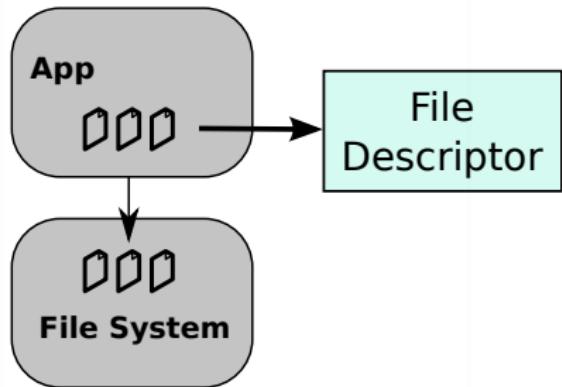


# SuperGlue – IDL-based System-Level Fault Tolerance

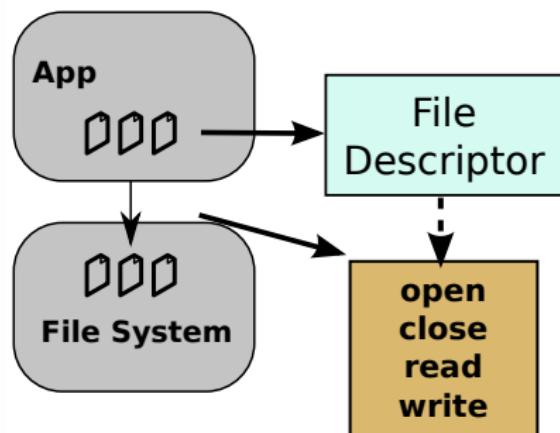


# Interface Description Model – Descriptor and Resource

- File descriptor (descriptor)

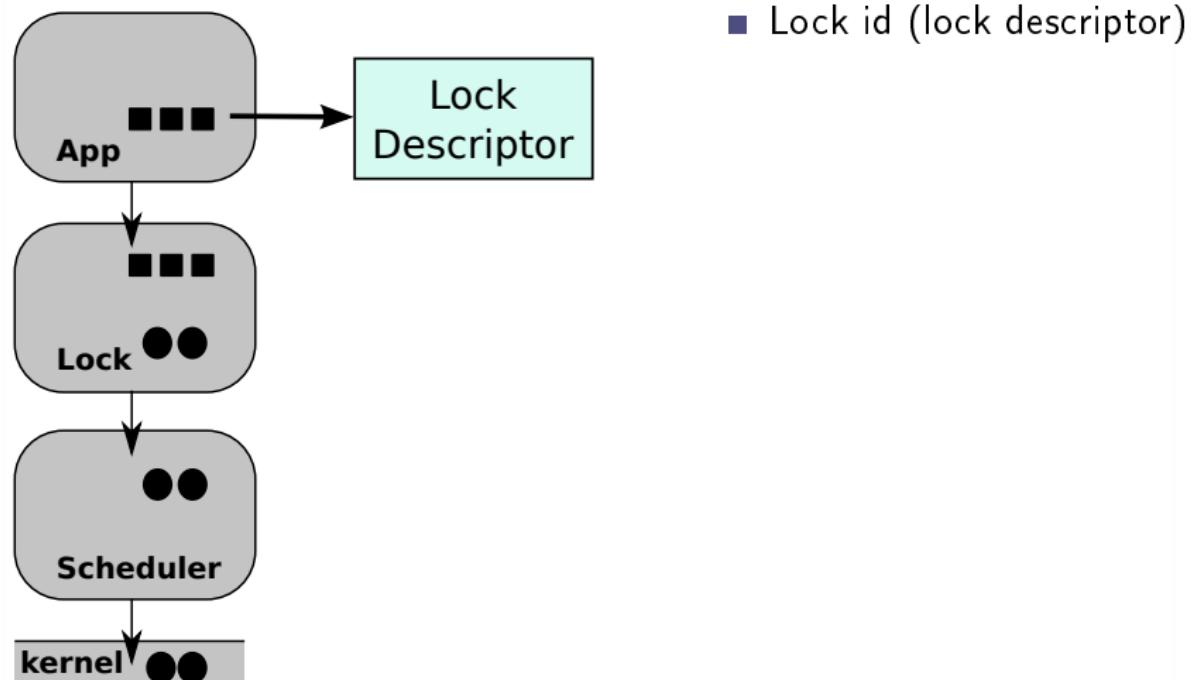


# Interface Description Model – Descriptor and Resource

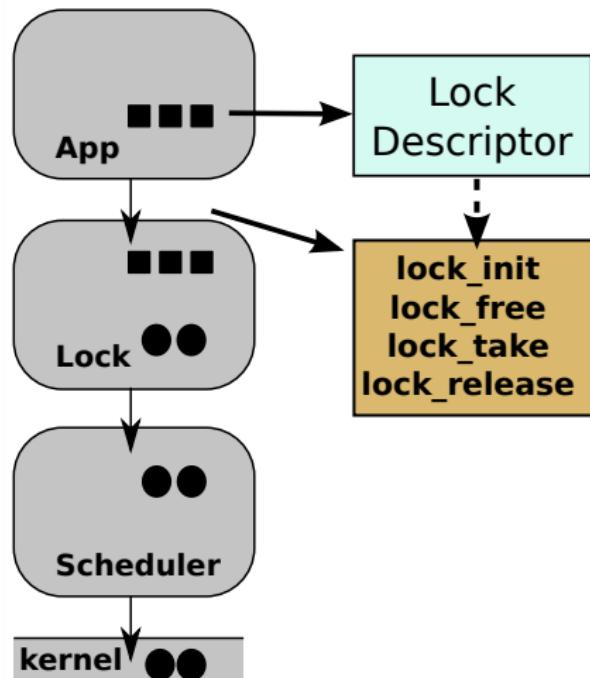


- File descriptor (descriptor)
- Interface functions operate on the file descriptor
  - open/close/read/write...

# Interface Description Model – Descriptor and Resource

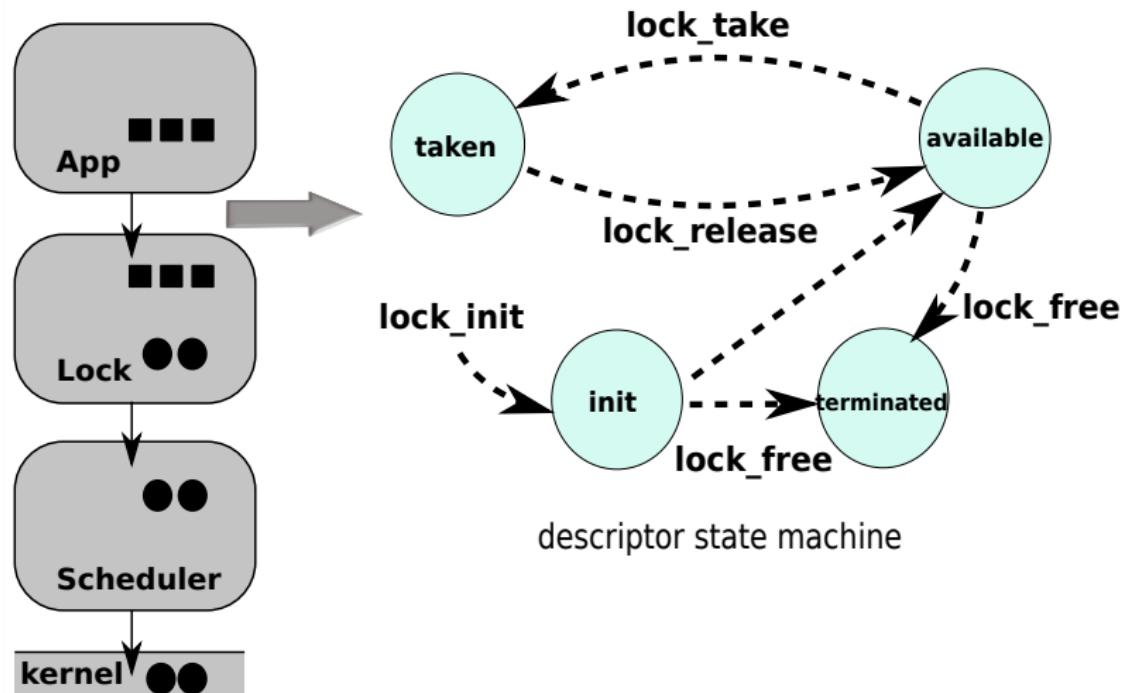


# Interface Description Model – Descriptor and Resource

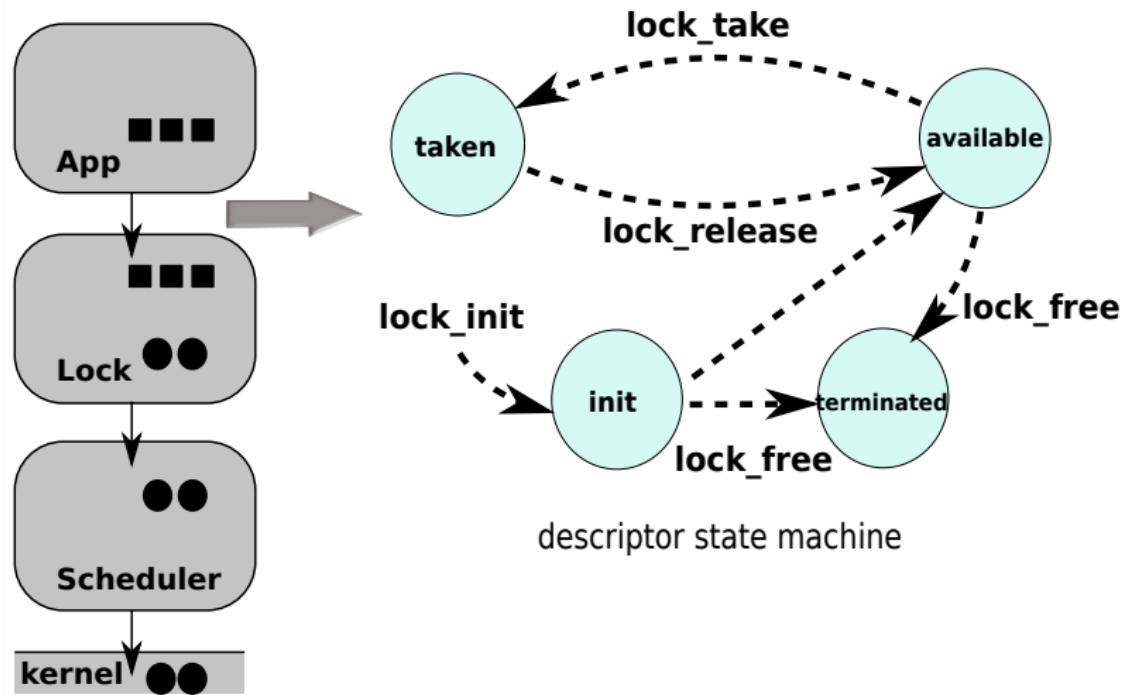


- Lock id (lock descriptor)
- Interface functions operate on lock descriptor
  - take/release lock...

# Interface Description Model – Descriptor and Resource

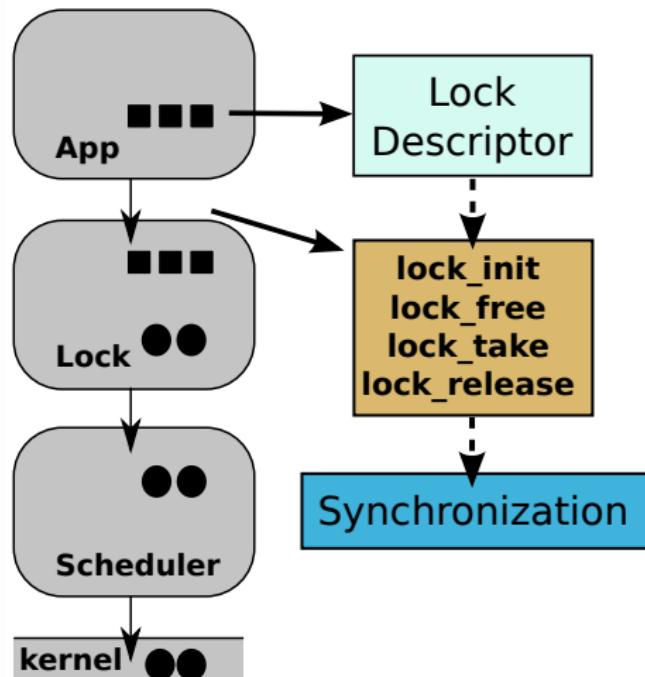


# Interface Description Model – Descriptor and Resource



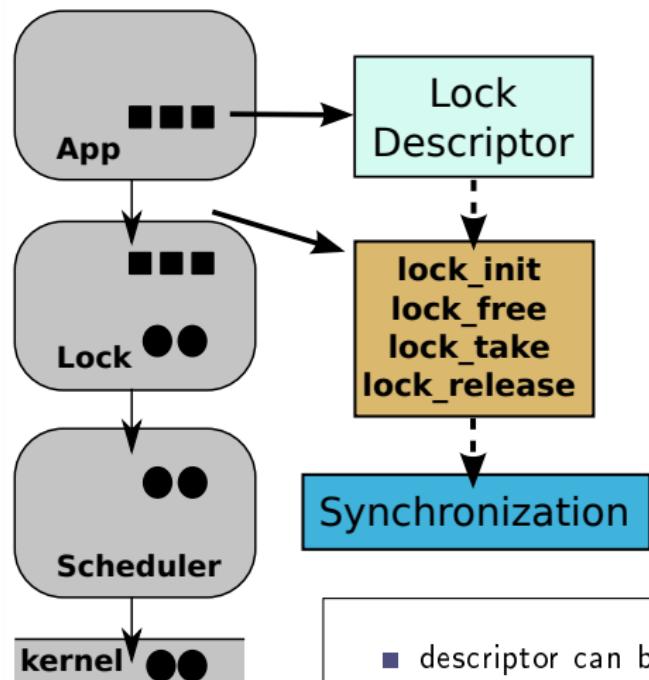
*lock, scheduler, memory manager, file system, event...*

# Interface Description Model – Descriptor and Resource



- Lock id (lock descriptor)
- Interface functions operate on lock descriptor
  - take/release lock...
- Access the synchronization (lock resource)

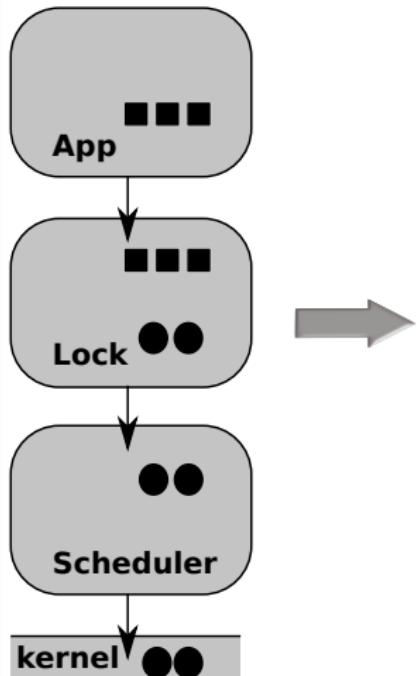
# Interface Description Model – Descriptor and Resource



- Lock id (lock descriptor)
- Interface functions operate on lock descriptor
  - take/release lock...
- Access the synchronization (lock resource)

- descriptor can be accessed by multiple components
- thread can block when access a resource
- ...

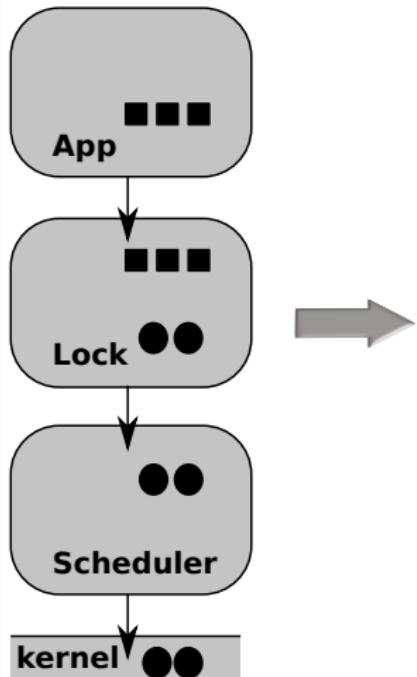
# System Model in IDL-based Specification



```
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(spid_t desc_data(spid));  
int lock_component_take(spid_t spid, ul_t desc(lock_id),  
                        u32_t desc_data(thd_id));  
int lock_component_pretake(spid_t spid, ul_t desc(lock_id),  
                           u32_t thd_id);  
int lock_component_release(spid_t spid, ul_t desc(lock_id));  
int lock_component_free(spid_t spid, ul_t desc_terminate(lock_id));
```

“descriptor and resource” + “interface functions”  
→ described using IDL

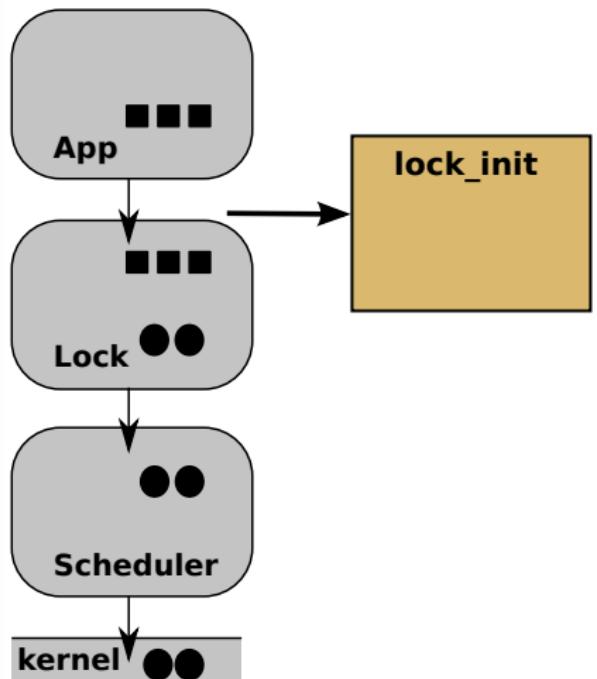
# System Model in IDL-based Specification



```
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(spid_t desc_data(spid));  
int lock_component_take(spid_t spid, ul_t desc(lock_id),  
                       u32_t desc_data(thd_id));  
int lock_component_pretake(spid_t spid, ul_t desc(lock_id),  
                          u32_t thd_id);  
int lock_component_release(spid_t spid, ul_t desc(lock_id));  
int lock_component_free(spid_t spid, ul_t desc_terminate(lock_id));
```

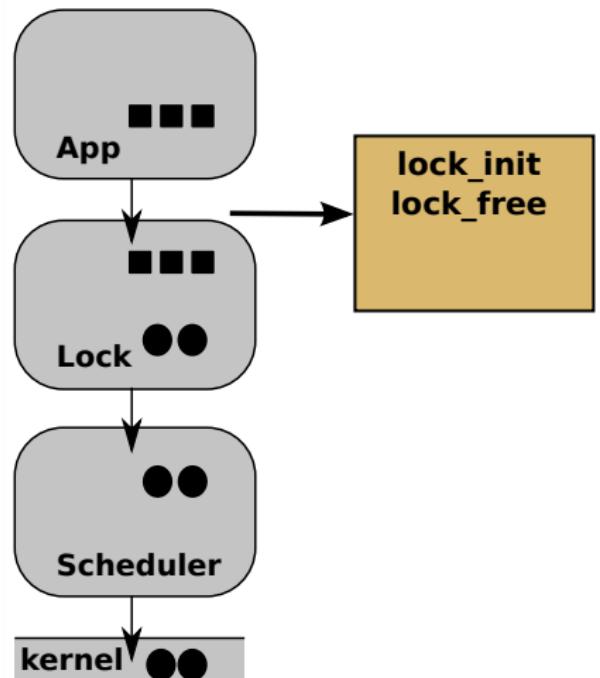
“descriptor and resource” + “interface functions”  
→ described using IDL in average 37 LOC

# Component Interface Function Types



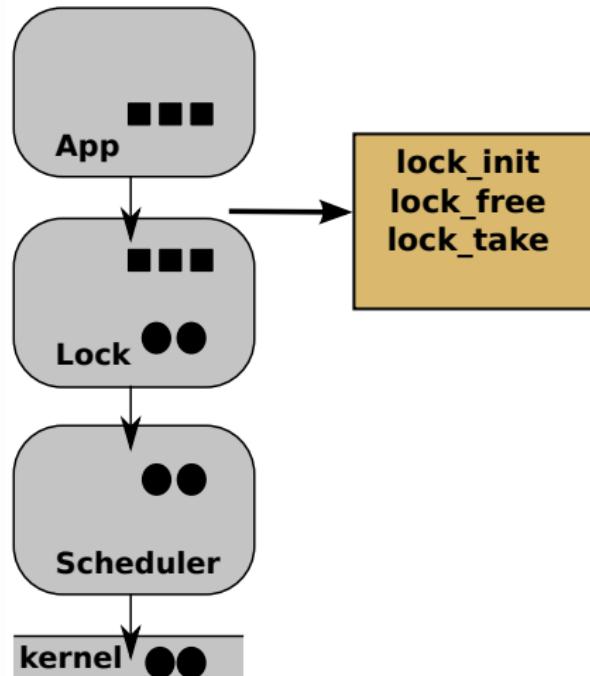
- create new descriptor

# Component Interface Function Types



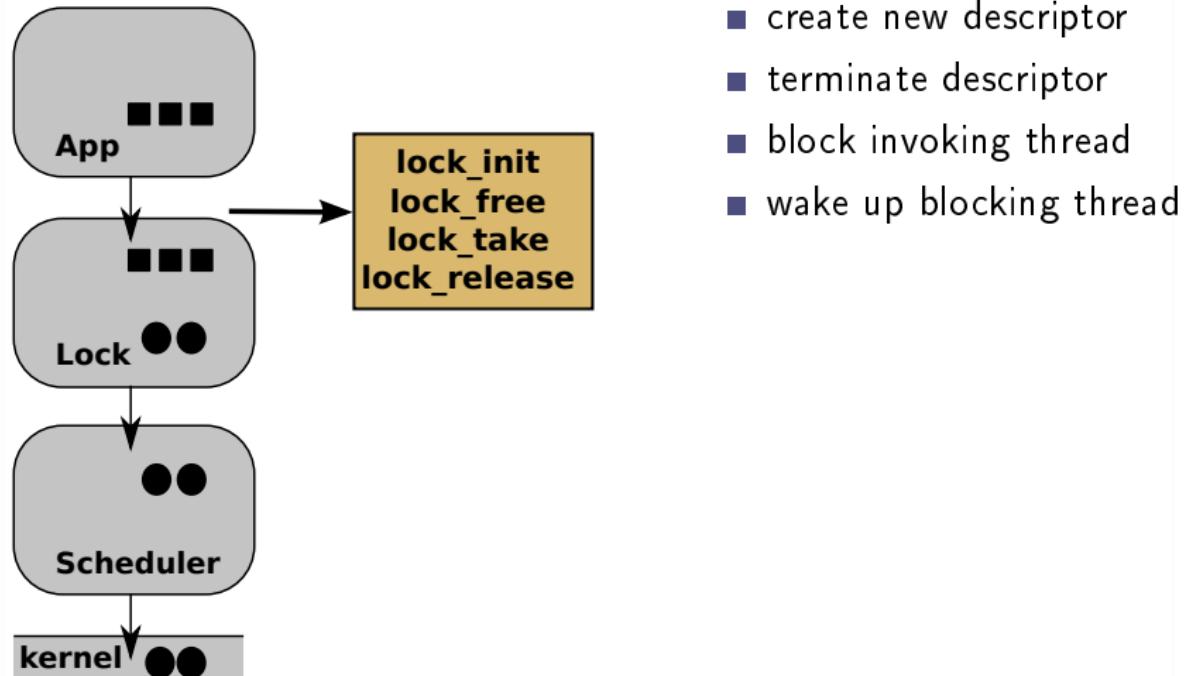
- create new descriptor
- terminate descriptor

# Component Interface Function Types

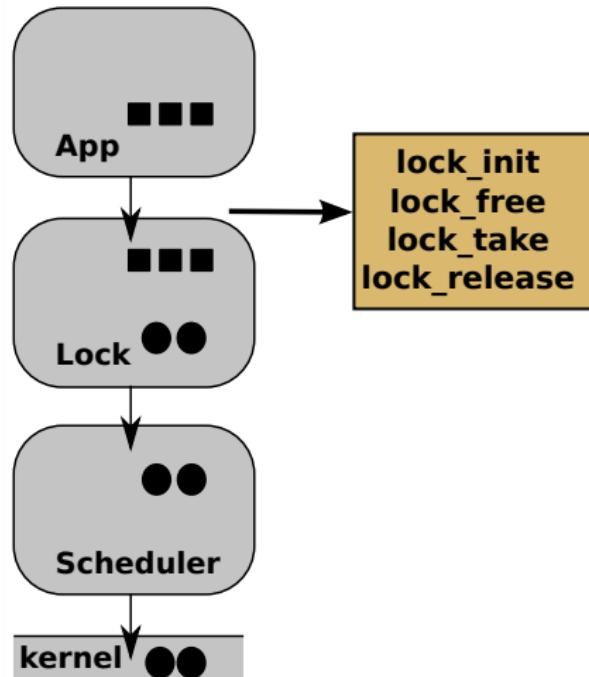


- create new descriptor
- terminate descriptor
- block invoking thread

# Component Interface Function Types



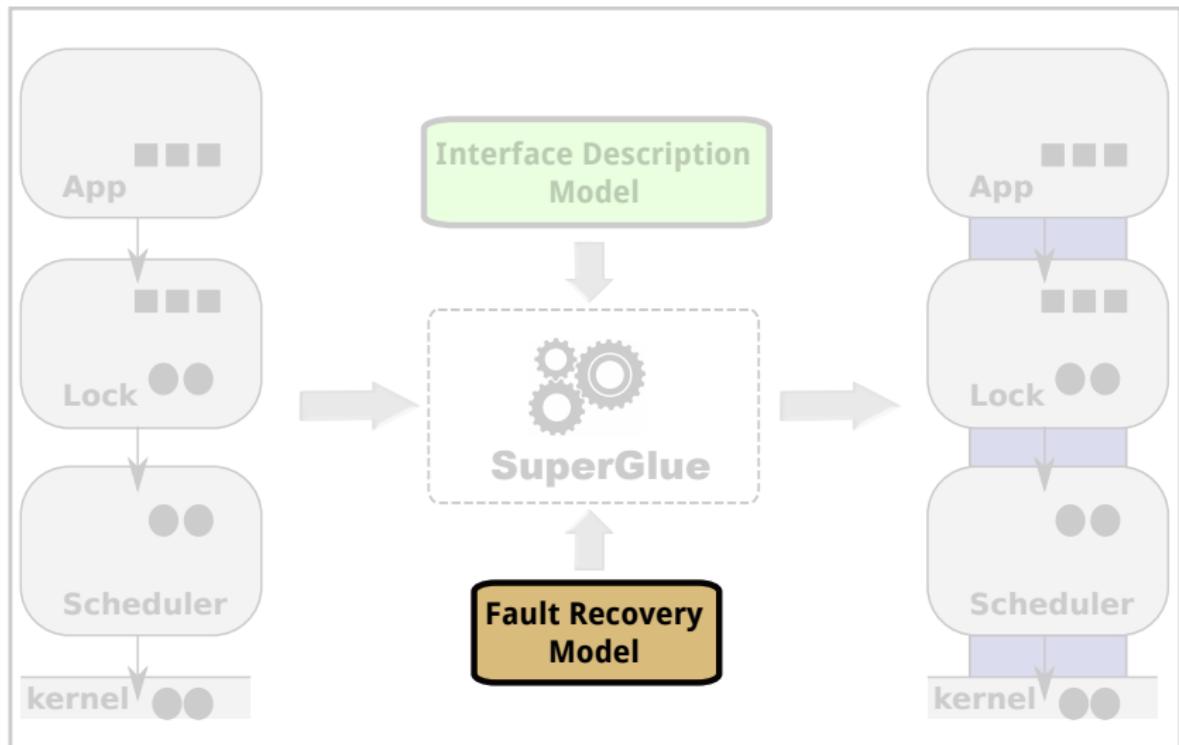
# Component Interface Function Types



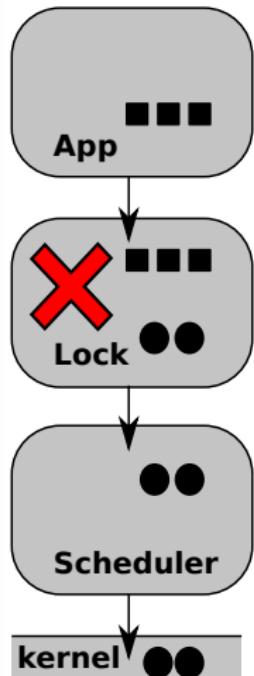
- create new descriptor
- terminate descriptor
- block invoking thread
- wake up blocking thread

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

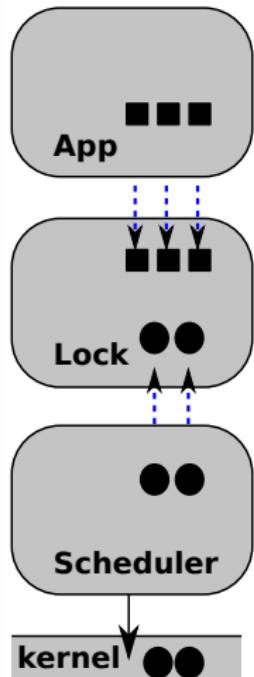
# SuperGlue – IDL-based System-Level Fault Tolerance



# Fault Recovery Model



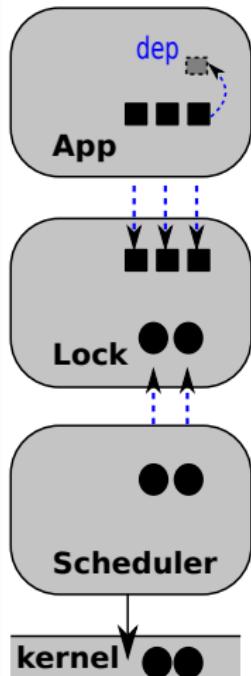
# Fault Recovery Model



## *Basic Recovery*

- always through component operation

# Fault Recovery Model



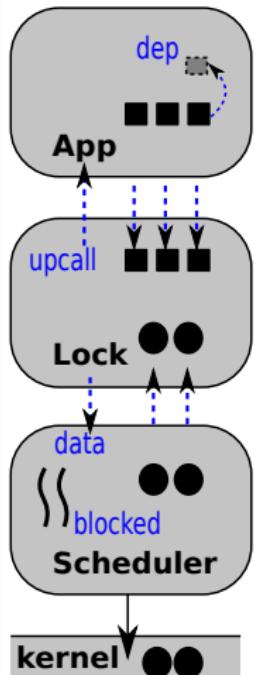
## Basic Recovery

- always through component operation

## Recovery with Dependency

- recover the dependency descriptor first
- e.g.,  $new\_fd = accept(fd)$ . Descriptor  $fd$  needs to be recovered before  $new\_fd$

# Fault Recovery Model



## Basic Recovery

- always through component operation

## Recovery with Dependency

- recover the dependency descriptor first
- e.g.,  $\text{new\_fd} = \text{accept(fd)}$ . Descriptor  $fd$  needs to be recovered before  $\text{new\_fd}$

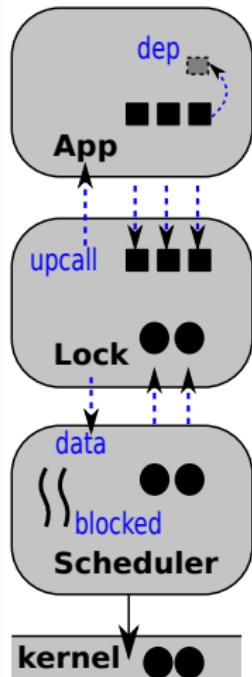
## Timing of Recovery

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

## Recovery with Data Storage

## Recovery with Upcall

# Fault Recovery Model



## *Basic Recovery*

- always through component operation

## *Recovery with Dependency*

- recover the dependency descriptor first
- e.g.,  $\text{new\_fd} = \text{accept(fd)}$ . Descriptor  $fd$  needs to be recovered before  $\text{new\_fd}$

## *Timing of Recovery*

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

## *Recovery with Data Storage*

## *Recovery with Upcall*

*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_retbl(u1_t, lock_id)  
lock_component_allc(spidid_t desc_data(spidid));  
int lock_component_take(spidid_t spdid, u1_t desc(lock_id),  
                        u32_t desc_data(thd_id));  
int lock_component_pretake(spidid_t spdid, u1_t desc(lock_id),  
                           u32_t thd_id);  
int lock_component_release(spidid_t spdid, u1_t desc(lock_id));  
int lock_component_free(spidid_t spdid, u1_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,  
    desc_desc_release_desc = false,  
    desc_desc_pretake_desc = false,  
    desc_desc_take_desc   = false,  
    desc_desc_update_desc = false,  
    desc_desc_update_pretake_desc = false,  
    desc_desc_update_take_desc = false,  
};  
  
sm_creation()  
sm_transition()  
sm_transition(lock_component_mkdir, 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_retrval(ul_t, lock_id)  
lock_component_allc(spidid_t desc_data(spidid));  
int lock_component_take(spidid_t spdid, ul_t desc(lock_id),  
                        u32_t desc_data(thd_id));  
int lock_component_pretake(spidid_t spdid, ul_t desc(lock_id),  
                           u32_t thd_id);  
int lock_component_release(spidid_t spdid, ul_t desc(lock_id));  
int lock_component_free(spidid_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 → Recovery Mechanisms

## System Model in IDL

```
service_global_info = {  
    service           = lock,  
  
    desc_close_self_only   = true,  
    desc_desc_create_desc = true,  
    desc_desc_destroy_desc = true,  
    desc_desc_get_desc    = true,  
    desc_desc_set_desc    = true,  
    desc_desc_update_desc = true,  
    desc_desc_update_desc = true,  
};  
  
sm_creation()  
sm_transitio  
sm_transitio  
sm_transitio  
sm_transitio  
sm_transitio  
sm_transitio  
sm_transitio  
sm_terminal(  
sm_block(loc  
sm_wakeup(loc  
  
desc_data_retv  
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: an IDL compiler parses IDL specifications and generates the code

### Recovery with upcall

dependency  
page

## Example – Client Invocation Stub

- Predicate ⇒ “*always included*”
- Code Template ⇒

```
/* predicate: true */
CSTUB_FN(IDL_ftype, 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){
        CSTUBFAULT_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 ⇒ “*always included*”
- Code Template ⇒

```
/* predicate: true */
CSTUB_FN(IDL_ftype, 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){
        CSTUBFAULT_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 creates descriptor AND the descriptor is not accessed between components*”
- Code Template  $\Rightarrow$

```
/* predicate:  $f \in I_{dr}^{create} \wedge \neg G_{dr}$  */  
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;
}

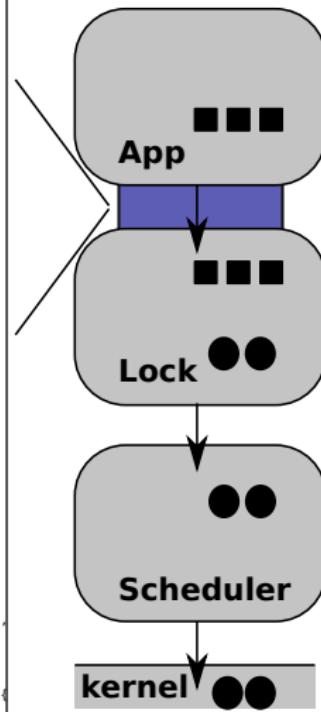
CSTUB_FN(int, lock_component_take)(struct usr_inv_cap *uc, spdid_t spdid,
                                    ul_t lock_id, u32_t thd_id) {
    long fault = 0;
    int ret = 0;

    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;
}

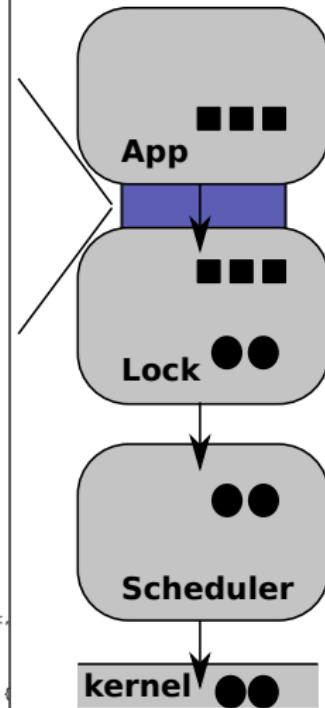
CSTUB_FN(int, lock_component_take)(struct usr_inv_cap *uc, spdid_t spdid,
                                    ul_t lock_id, u32_t thd_id) {
    long fault = 0;
    int ret = 0;

    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

1 Motivation and Challenges

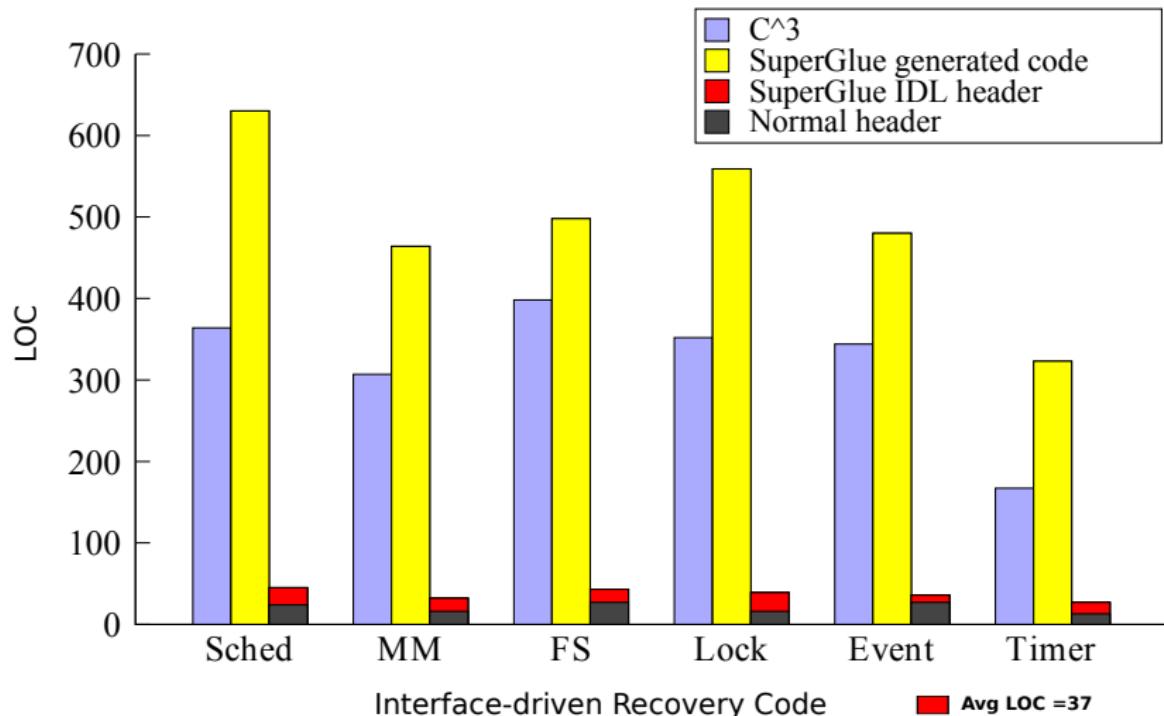
2 System-Level Fault Recovery

3 SuperGlue

4 Evaluation

5 Conclusion

# Code Generation Result



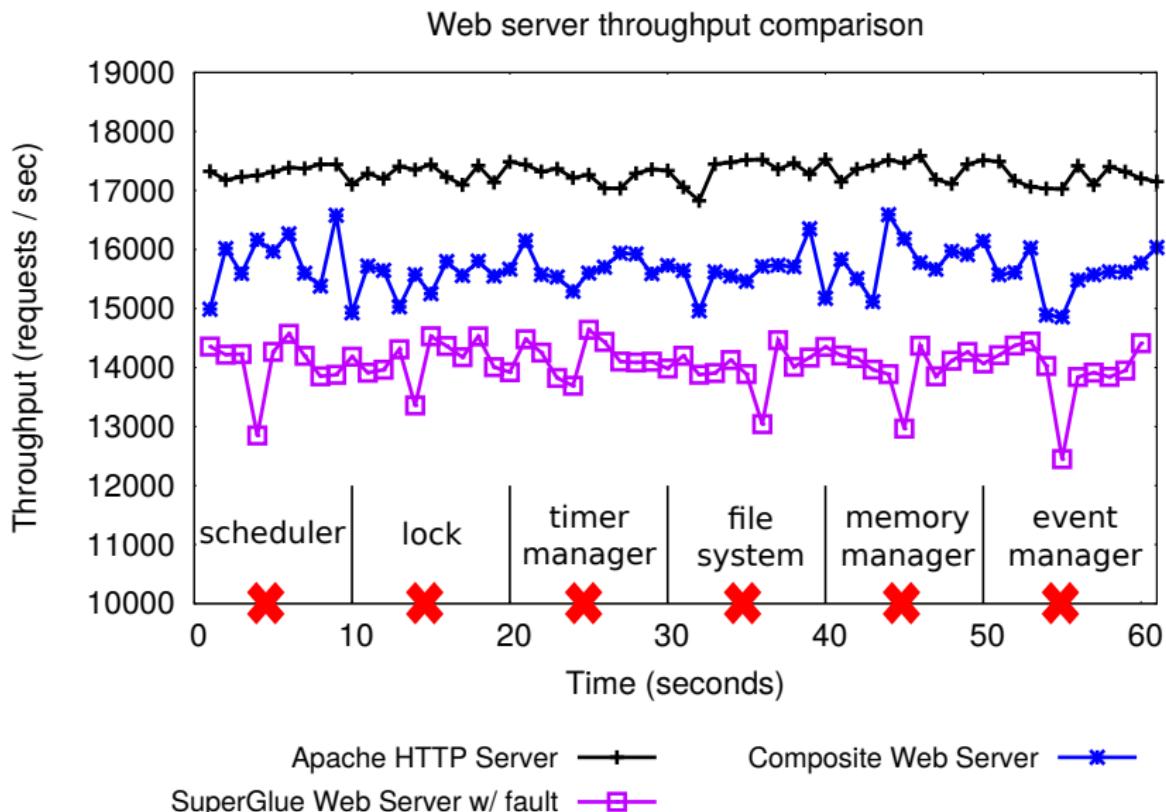
# Fault Injection Campaign Results

- SuperGlue implemented in *Composite* component-based OS
- Transient faults modeled as random register bit-flips
  - Software-implemented Fault Injection (SWIFI)

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

1 Motivation and Challenges

2 System-Level Fault Recovery

3 SuperGlue

4 Evaluation

5 Conclusion

# Conclusion

SuperGlue –

- automatically generates system-level fault recovery code
- effectively eases the programmer burden
- provides efficient and predictable fault recovery

Thanks

? || /\* \*/

composite.seas.gwu.edu

---

THE GEORGE  
WASHINGTON  
UNIVERSITY  

---

WASHINGTON, DC