Predictable, System-Level Fault Tolerance in a Component-based RTOS

(Dissertation Defense)

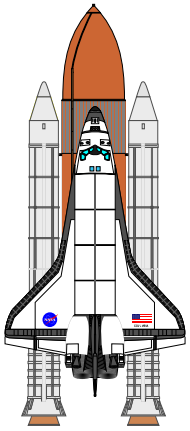
Presented by **Jiguo Song**

May 16, 2016

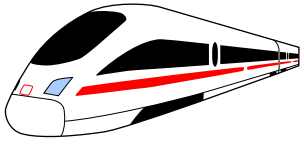
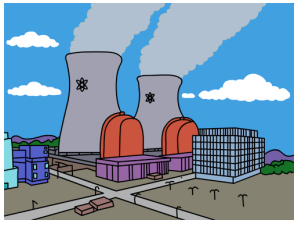
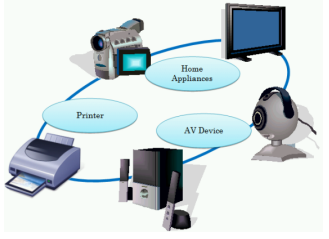
Computer Science Department
The George Washington University

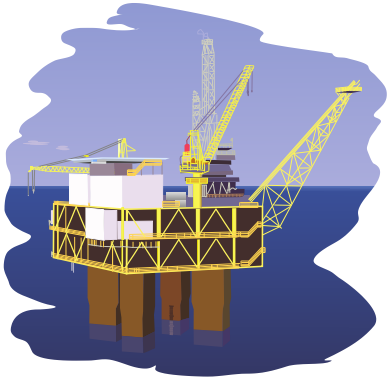
Committee: Prof. Gabriel Parmer (Advisor), Prof. Timothy Wood, Prof. Rahul Simha, Prof. Howie Huang

Real-Time and Embedded Systems



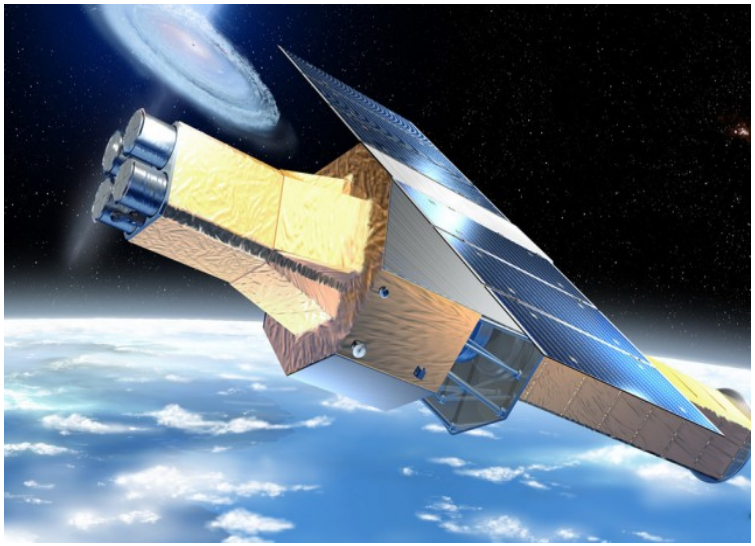




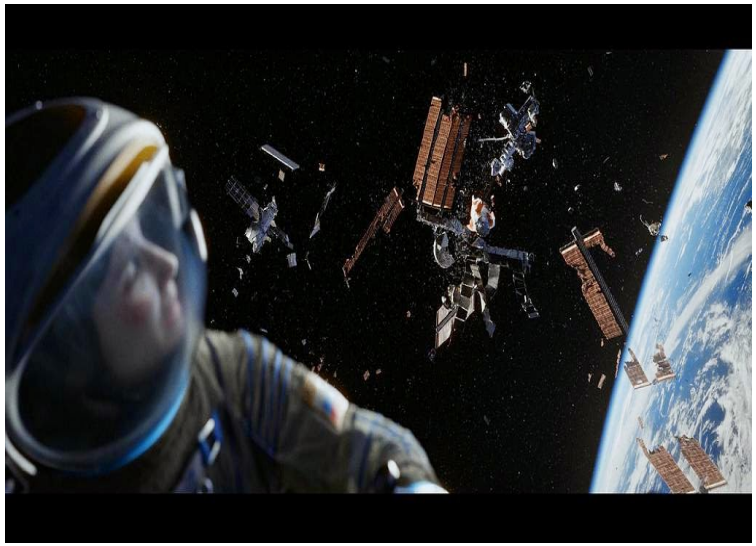

http://www.pcmag.com/encyclopedia/term/42554/embedded-system

Consequences of Embedded System Faults

Japanese X-ray astronomy satellite Hitomi lost in 2016



Fault

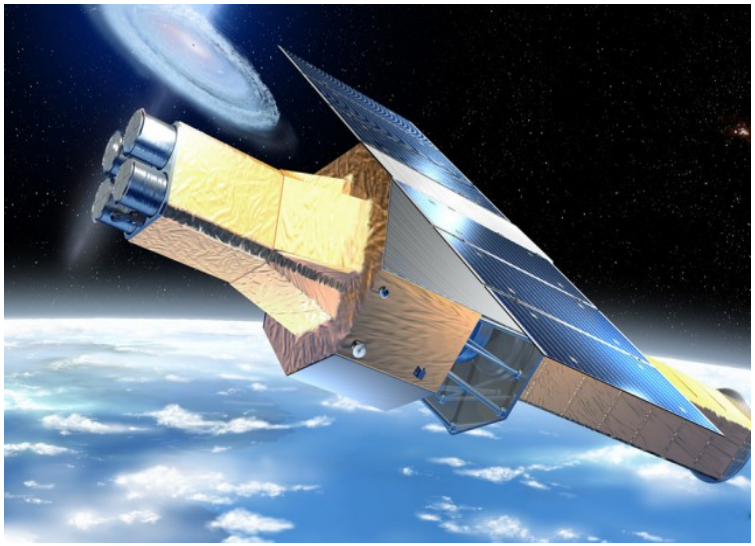


- Financial losses of $286 million USD
- "*It's a scientific tragedy*" - Richard Mushotzky, UMD

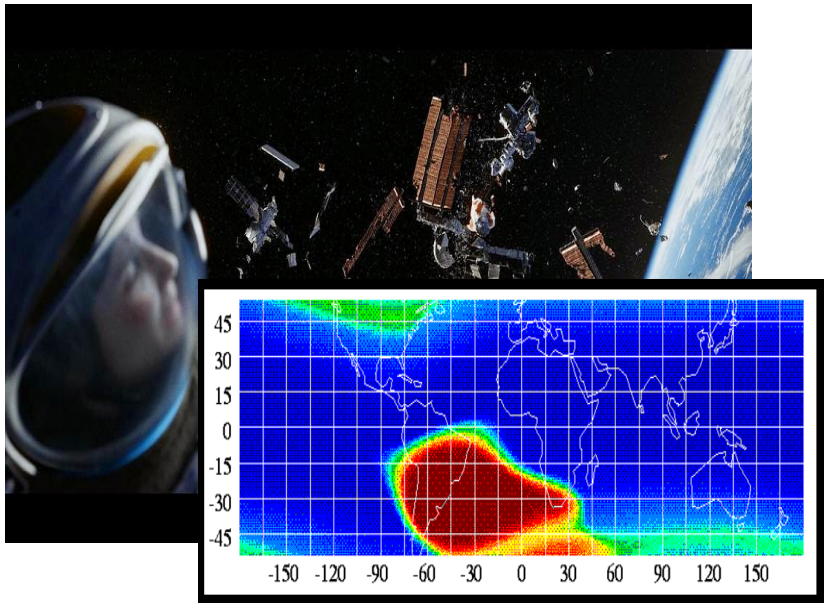
`http://www.nature.com/news/software-error-doomed-japanese-hitomi-spacecraft-1.19835`

Consequences of Embedded System Faults

Japanese X-ray astronomy satellite Hitomi lost in 2016



Fault



The South Atlantic Anomaly

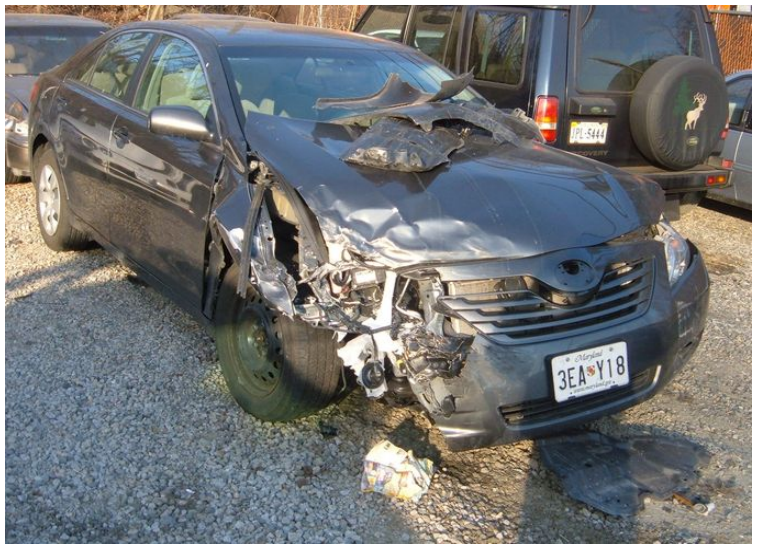
- Financial losses of $286 million USD
- "*It's a scientific tragedy*" - Richard Mushotzky, UMD

`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

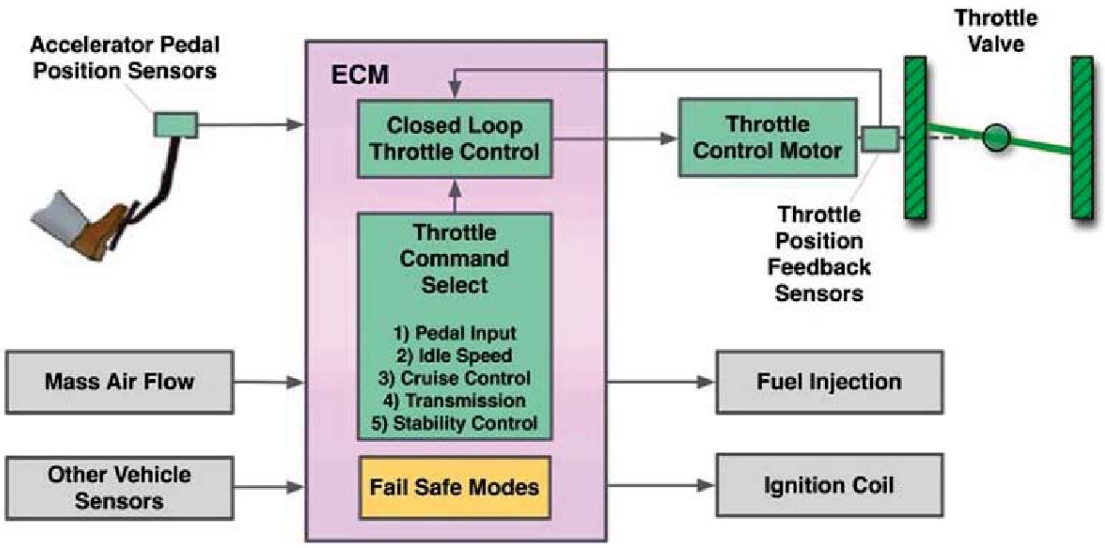
Fault

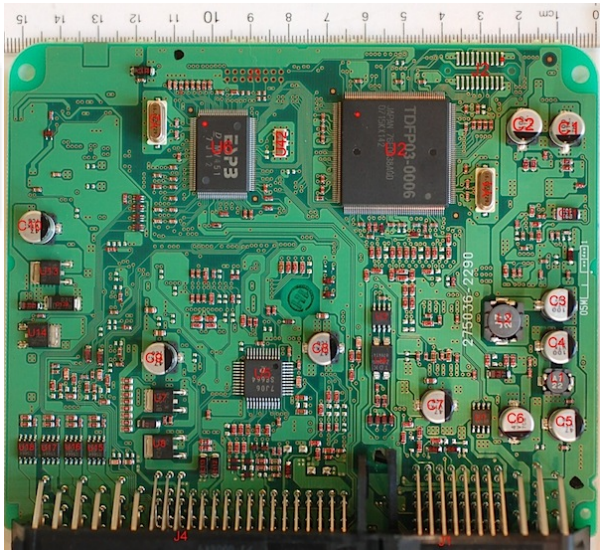


- 89 deaths as of May 2010 and nearly 400 U.S. lawsuits
- Recall 10+ million vehicles and pay $1.2 Billion USD fine

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

Sudden Unintended Acceleration



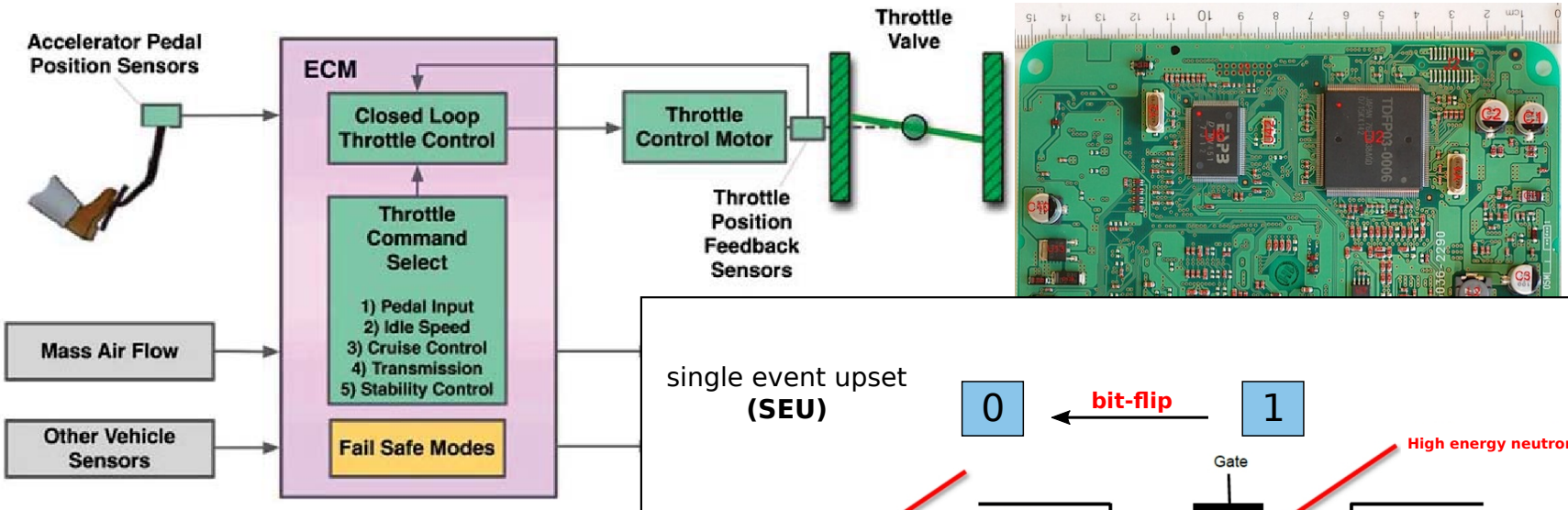


Uncontrolled acceleration in Toyota Camrys

- 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]

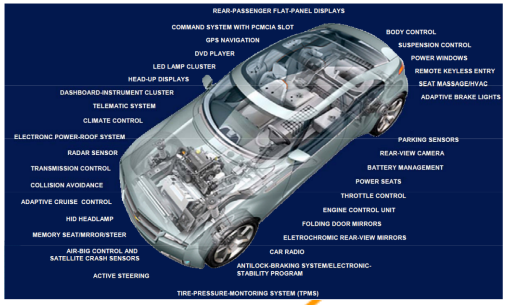
Uncontrolled acceleration in

- 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

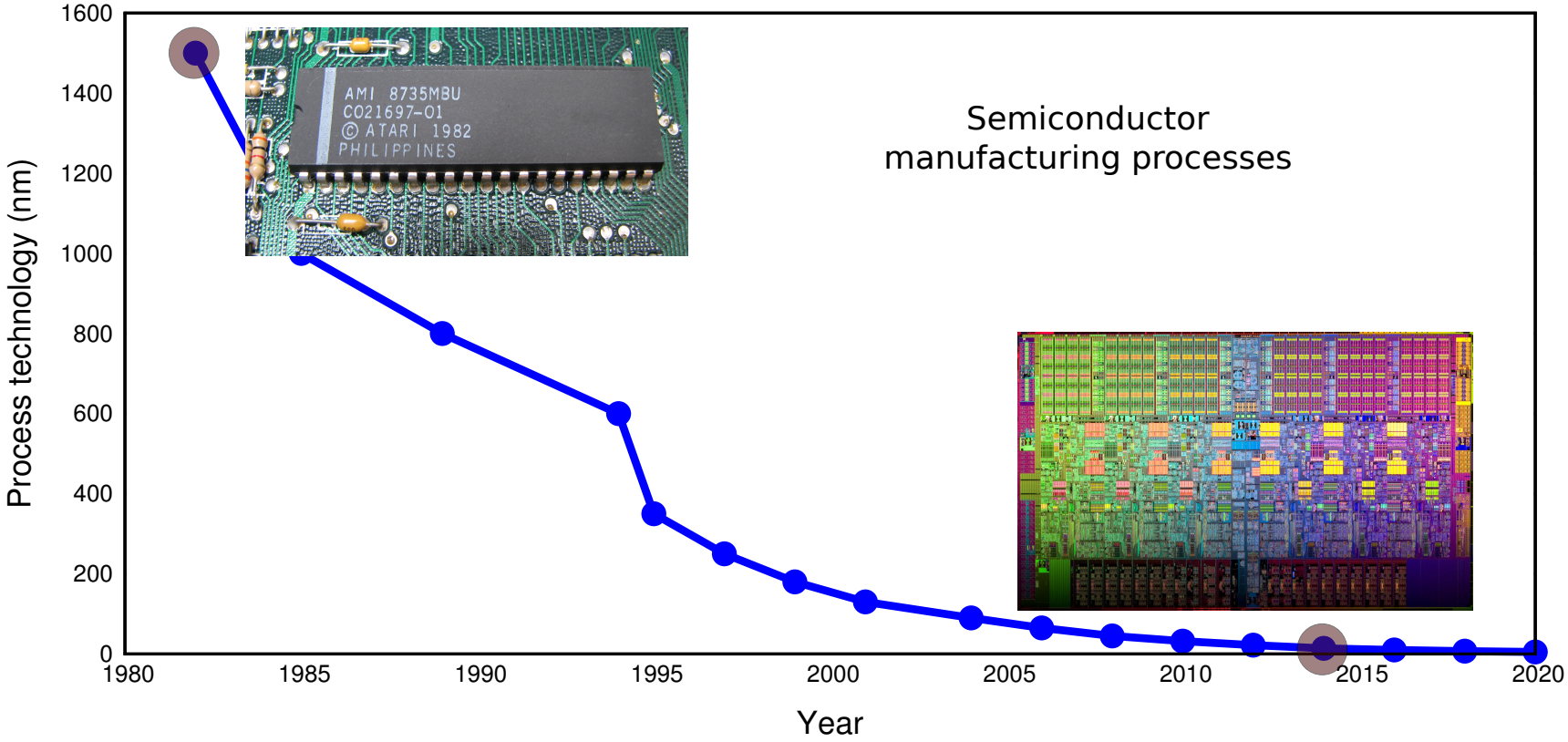
ABS: Antilock Brake System
ACC: Adaptive Cruise Control
BCM: Body Control Module
Cyl Deact: Cylinder Deactivation
ECS: Electronics, Controls, and Software
EGR: Exhaust Gas Recirculation.
GDI: Gas Direct Injection
OBD: Onboard Diagnostics
TCC: Torque Converter Clutch
PT: Powertrain
ESC: Electronic Stability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



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

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

Embedded faults: Bad Now, Worse Tomorrow

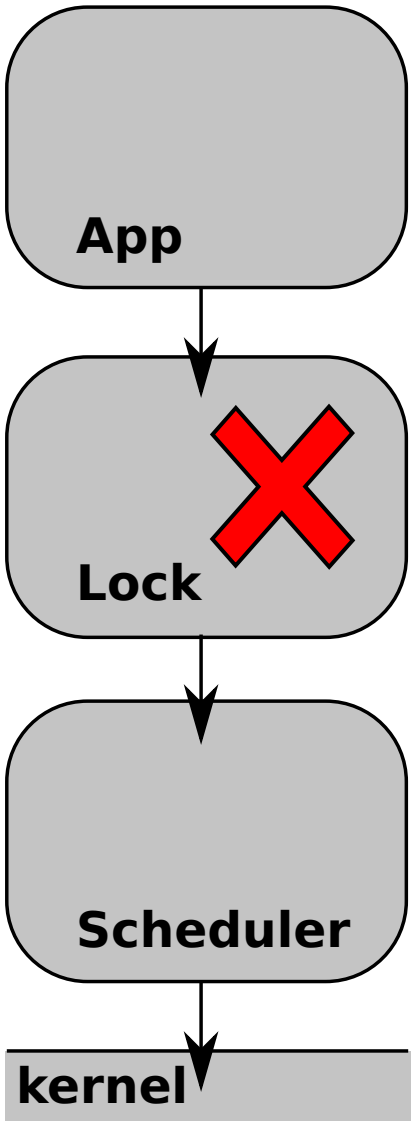


Decreasing process sizes $\rightarrow$ 5nm

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

Toward Dependable Real-Time Systems

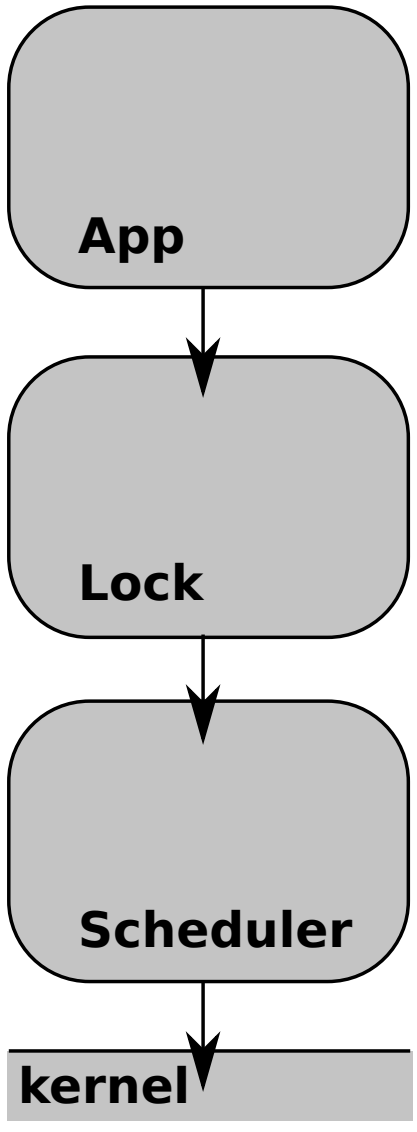
- Keys in dependable system design
  - isolation
  - detection
  - recovery
  - predictability



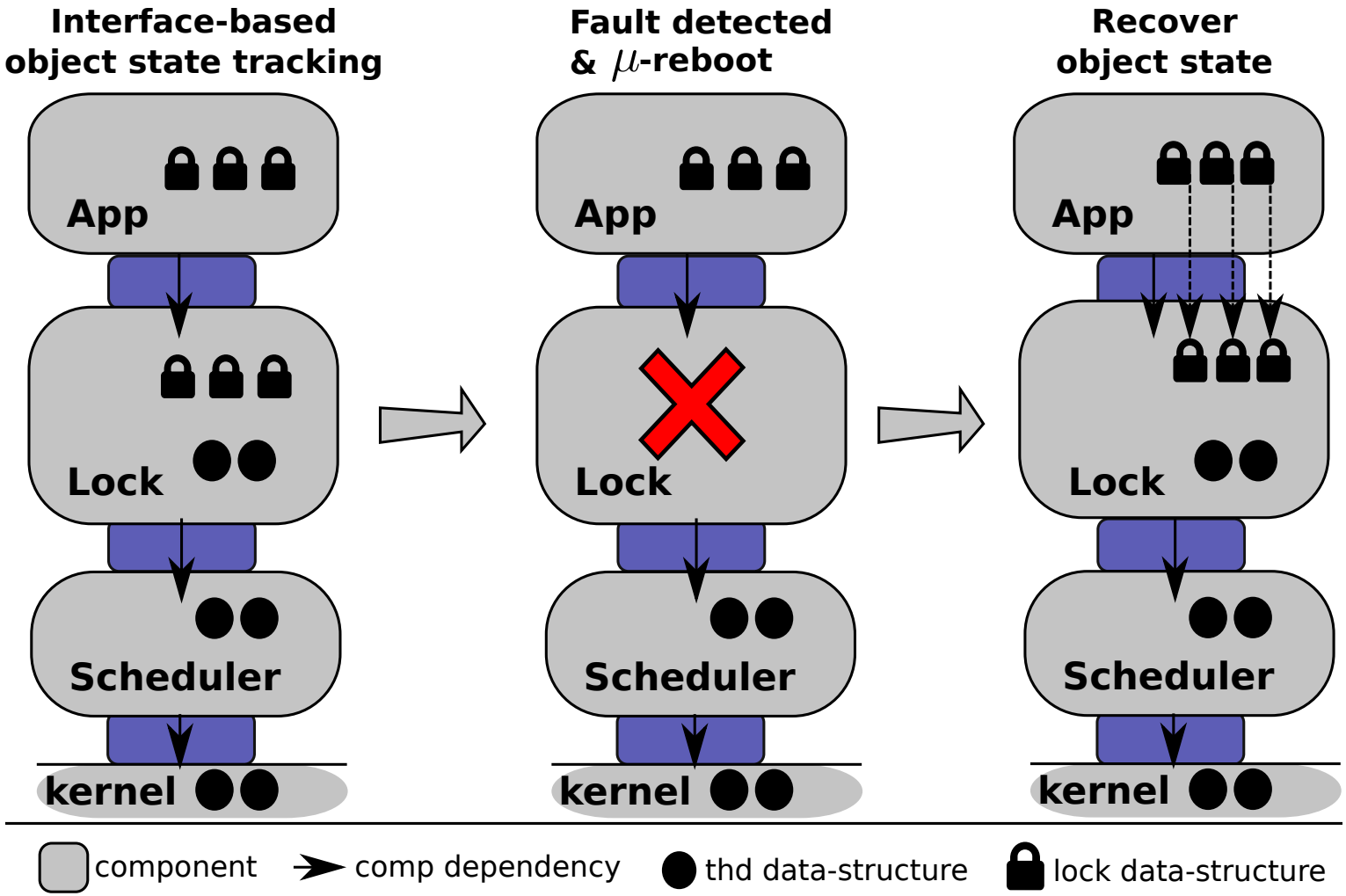
Toward Dependable Real-Time Systems

- Keys in dependable system design
  - isolation
  - detection
  - recovery
  - predictability

Goal
- predictably detect and recover
- system-level fault



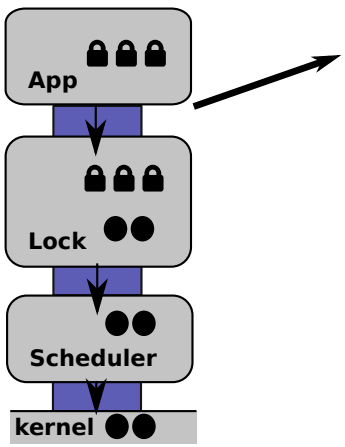
C^3 Review – System-Level Fault Recovery



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

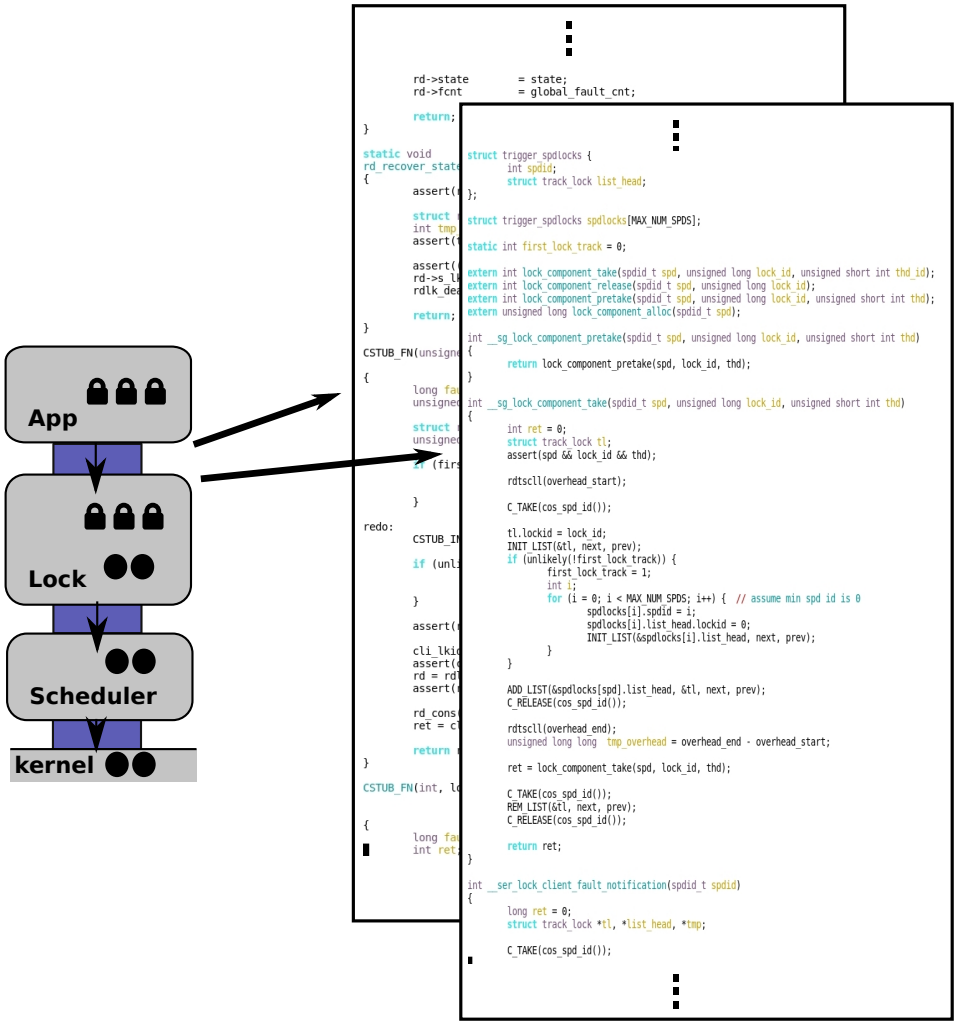
J. Song, J. Wittrock, and G. Parmer, "Predictable, efficient system-level fault tolerance in C^3" in RTSS, 2013

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;
```

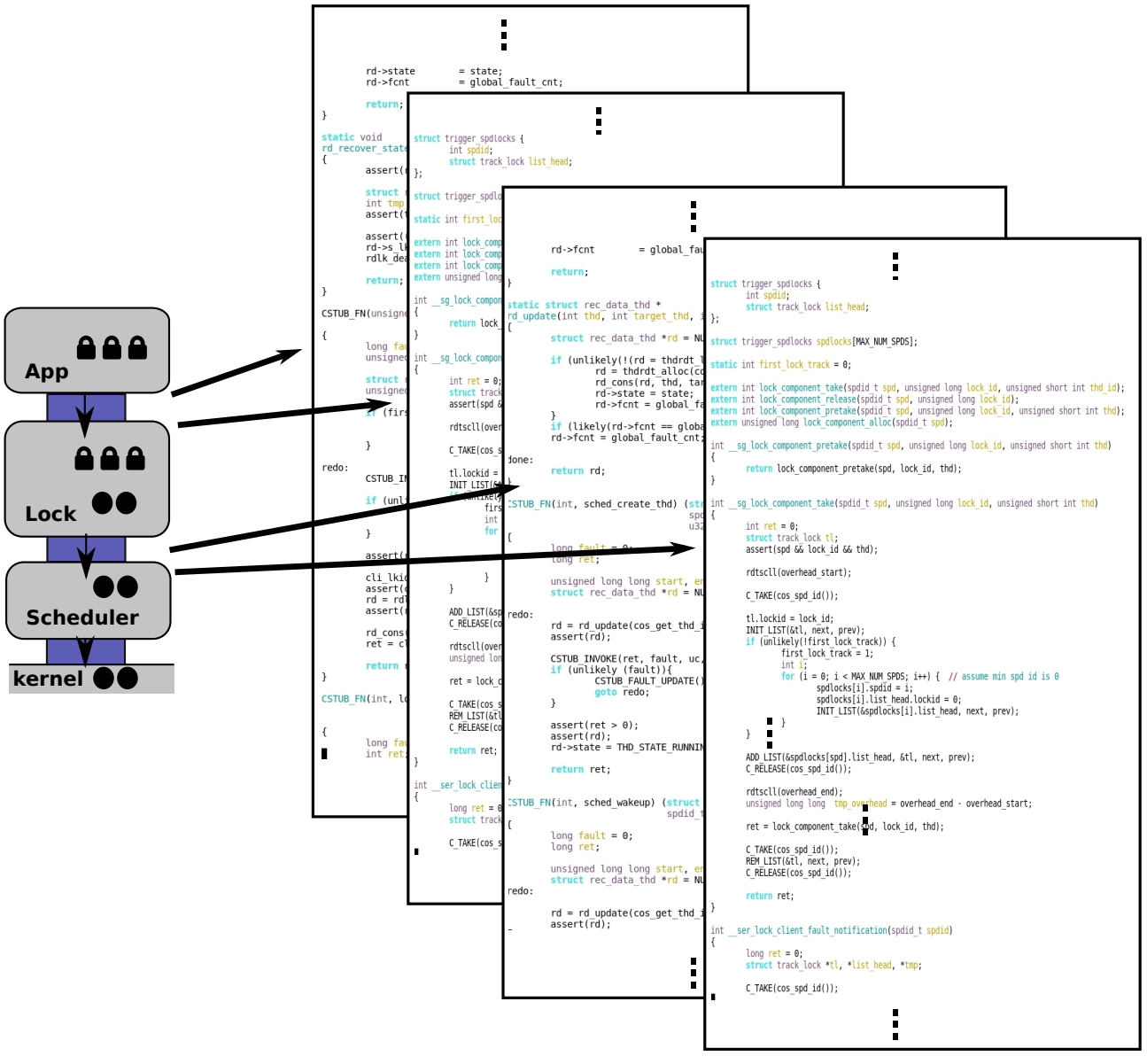
C³ Implementation – Writing Code Manually



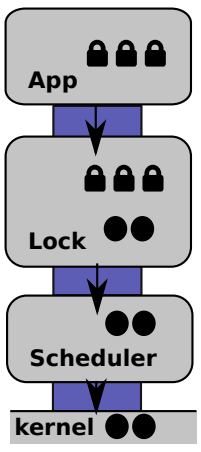

```
        rd->state        = state;
        rd->fcnt         = global_fault_cnt;

        return;
}

static void
rd_recover_state
{
        assert(

        struct
        int tmp
        assert(

        assert(
        rd->s_lk
        rdlk_dea

        return;
}

CSTUB_FN(unsigne

{
        long fa
        unsigne

        struct
        unsigne
        if (firs

        }

redo:
        CSTUB_IN

        if (unli

        }

        assert(

        cli_lkid
        assert(
        rd = rdt
        assert(

        rd_cons(
        ret = ct

        return

}

CSTUB_FN(int, lo

{
        long fa
        int ret;
```

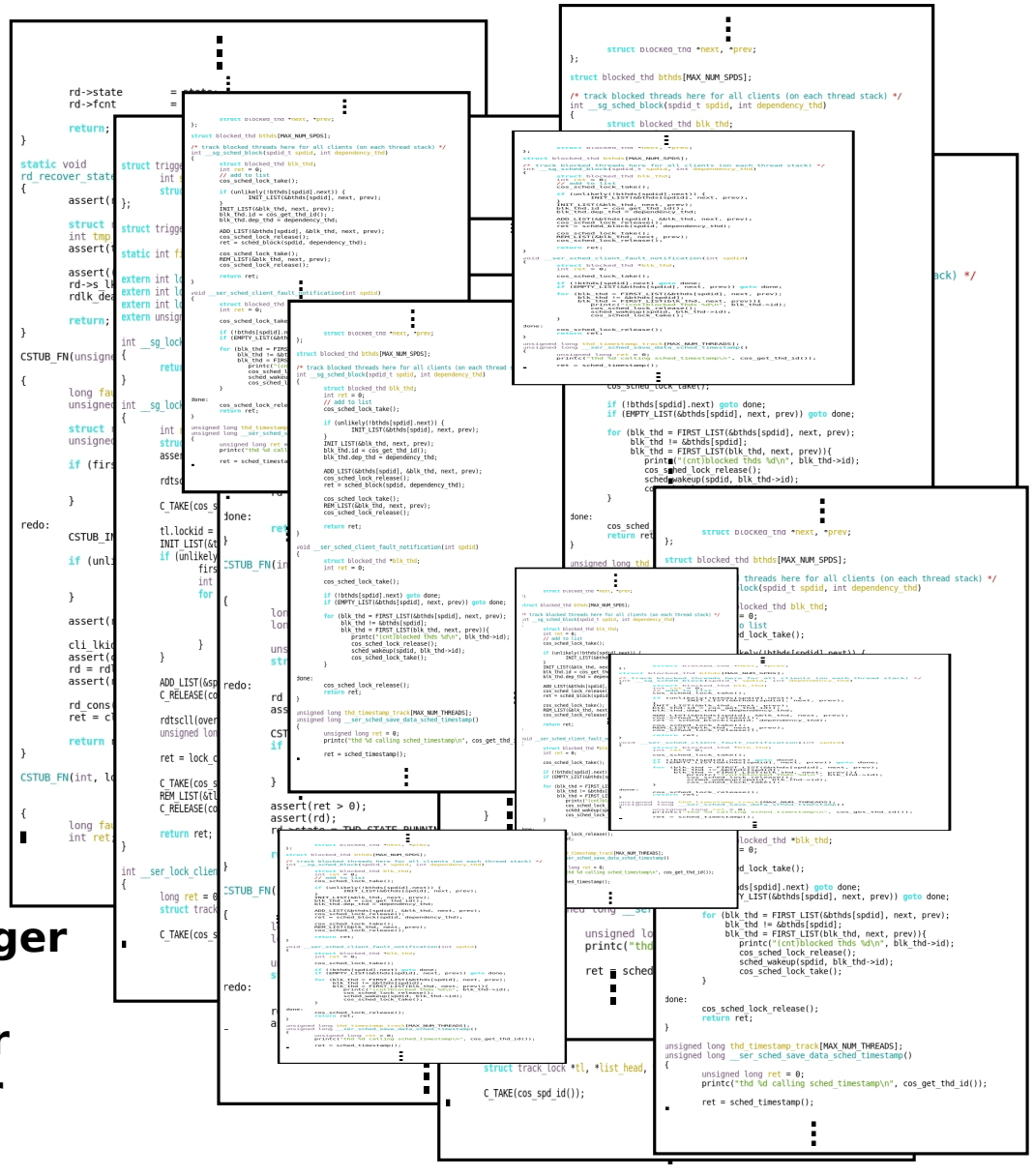
```
struct trigger_spdlocks {
        int spdid;
        struct track_lock list_head;
};

struct trigger_spdlocks spdlocks[MAX_NUM_SPDS];

static int first_lock_track = 0;

extern int lock_component_take(spdid_t spd, unsigned long lock_id, unsigned short int thd_id);
extern int lock_component_release(spdid_t spd, unsigned long lock_id);
extern int lock_component_pretake(spdid_t spd, unsigned long lock_id, unsigned short int thd);
extern unsigned long lock_component_alloc(spdid_t spd);

int __sg_lock_component_pretake(spdid_t spd, unsigned long lock_id, unsigned short int thd)
{
        return lock_component_pretake(spd, lock_id, thd);
}

int __sg_lock_component_take(spdid_t spd, unsigned long lock_id, unsigned short int thd)
{
        int ret = 0;
        struct track_lock tl;
        assert(spd && lock_id && thd);

        rdtscll(overhead_start);

        C_TAKE(cos_spd_id());

        tl.lockid = lock_id;
        INIT_LIST(&tl, next, prev);
        if (unlikely(!first_lock_track)) {
                first_lock_track = 1;
                int i;
                for (i = 0; i < MAX_NUM_SPDS; i++) {  // assume min spd id is 0
                        spdlocks[i].spdid = i;
                        spdlocks[i].list_head.lockid = 0;
                        INIT_LIST(&spdlocks[i].list_head, next, prev);
                }
        }

        ADD_LIST(&spdlocks[spd].list_head, &tl, next, prev);
        C_RELEASE(cos_spd_id());

        rdtscll(overhead_end);
        unsigned long long  tmp_overhead = overhead_end - overhead_start;

        ret = lock_component_take(spd, lock_id, thd);

        C_TAKE(cos_spd_id());
        REM_LIST(&tl, next, prev);
        C_RELEASE(cos_spd_id());

        return ret;
}

int __ser_lock_client_fault_notification(spdid_t spdid)
{
        long ret = 0;
        struct track_lock *tl, *list_head, *tmp;

        C_TAKE(cos_spd_id());
```

C³ Implementation – Writing Code Manually



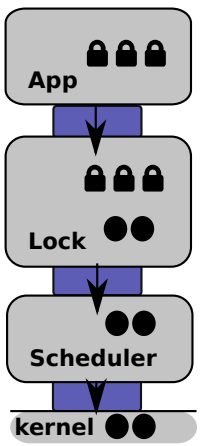
C³ Implementation – Writing Code Manually

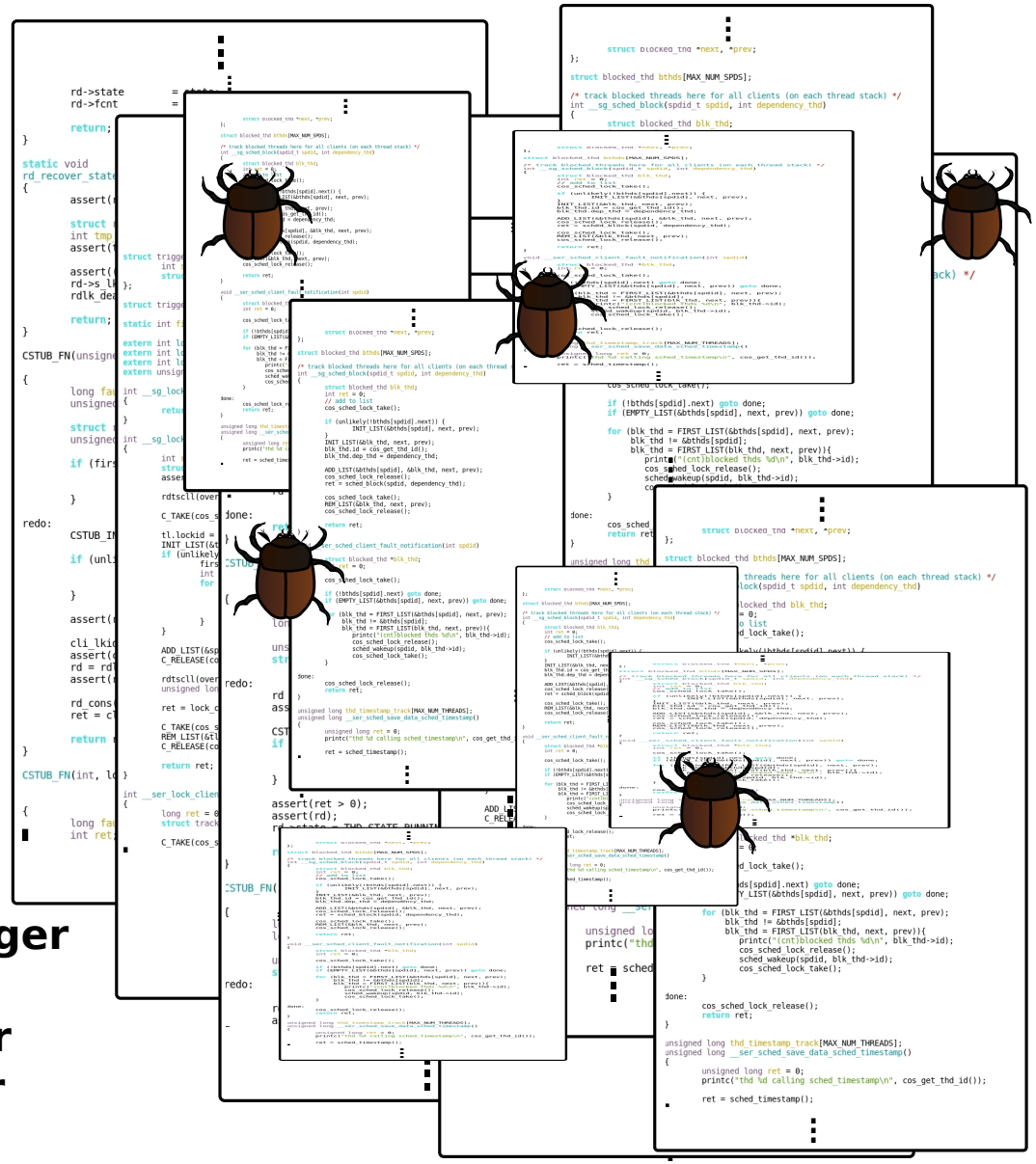




lock
scheduler
memory manager
file system
event manager
timer manager
...

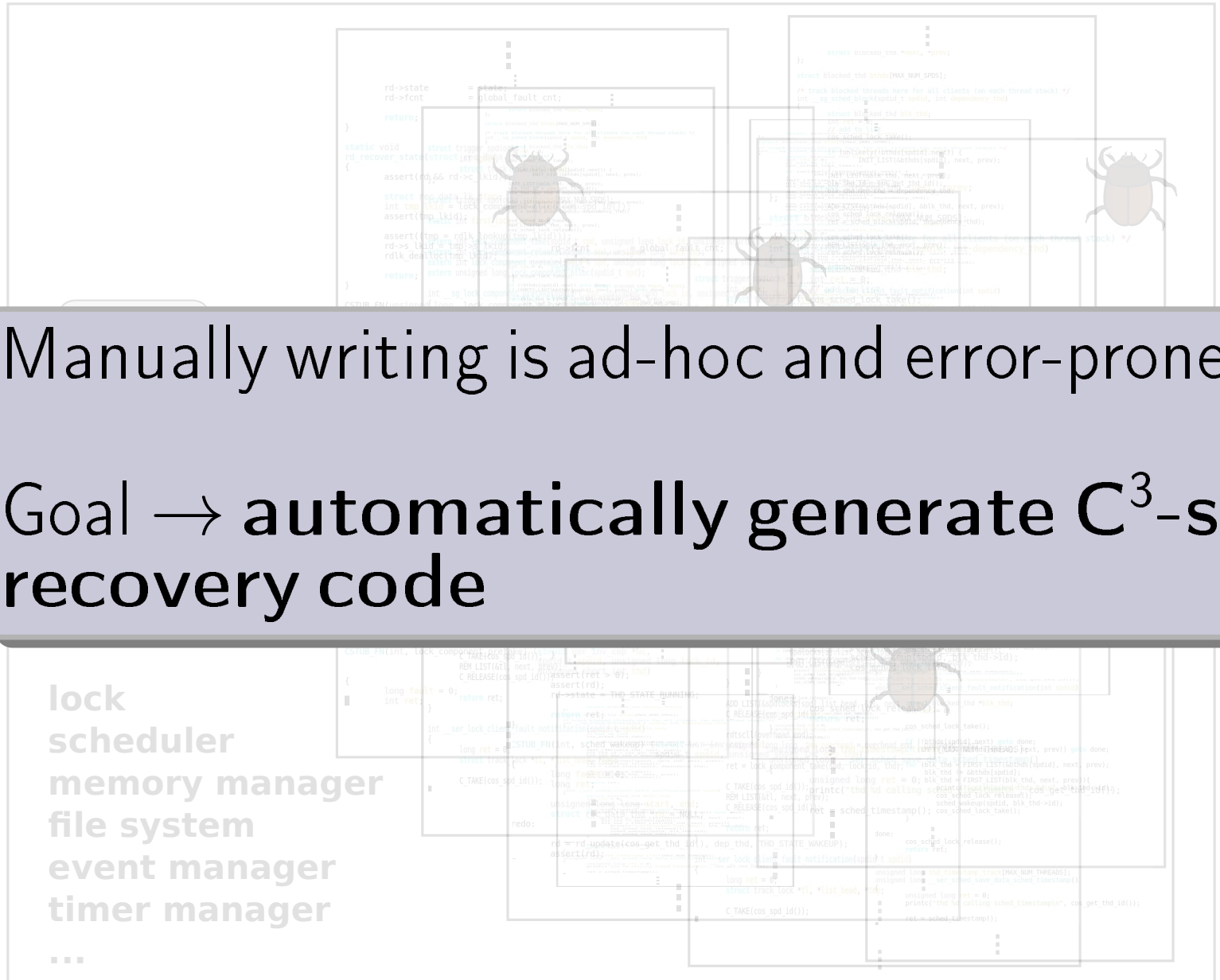
C³ Implementation – Writing Code Manually





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

C^3 Implementation – Writing Code Manually



Manually writing is ad-hoc and error-prone

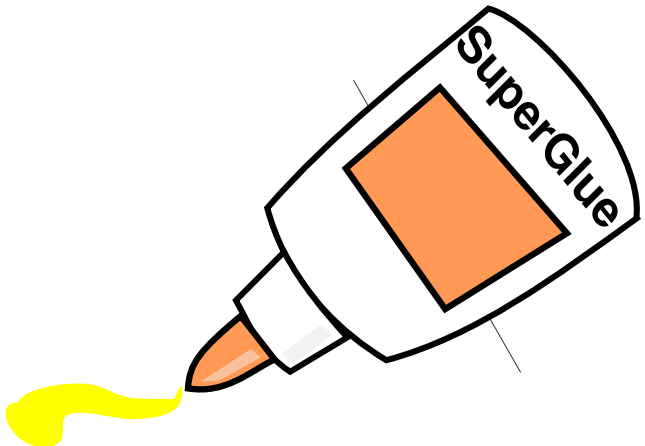
Goal $\rightarrow$ **automatically generate C^3-style recovery code**

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

SuperGlue – IDL-based System-Level Fault Tolerance

SuperGlue

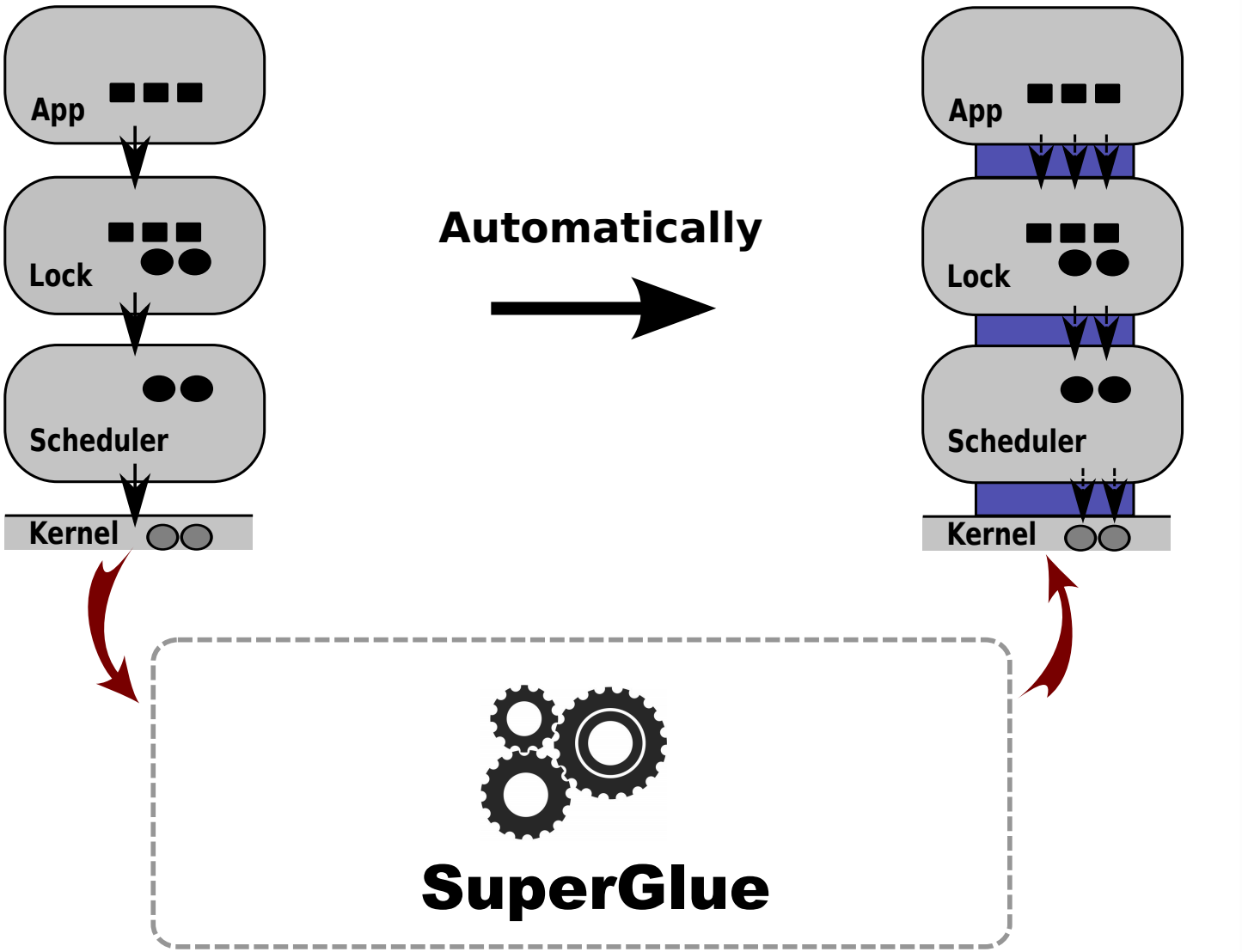
- automatically generate interface-driven recovery code
- for system-level services



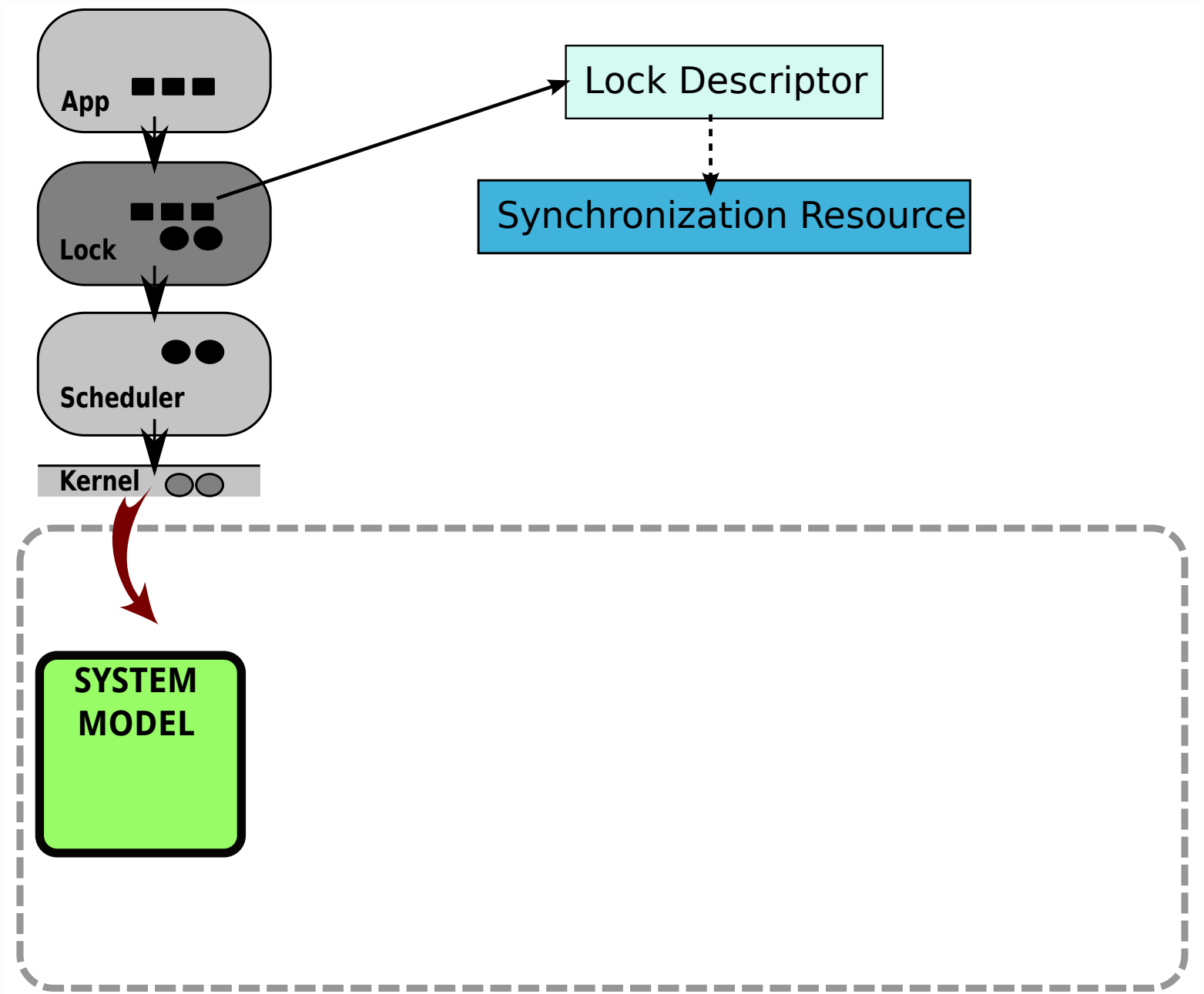
Main ideas

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

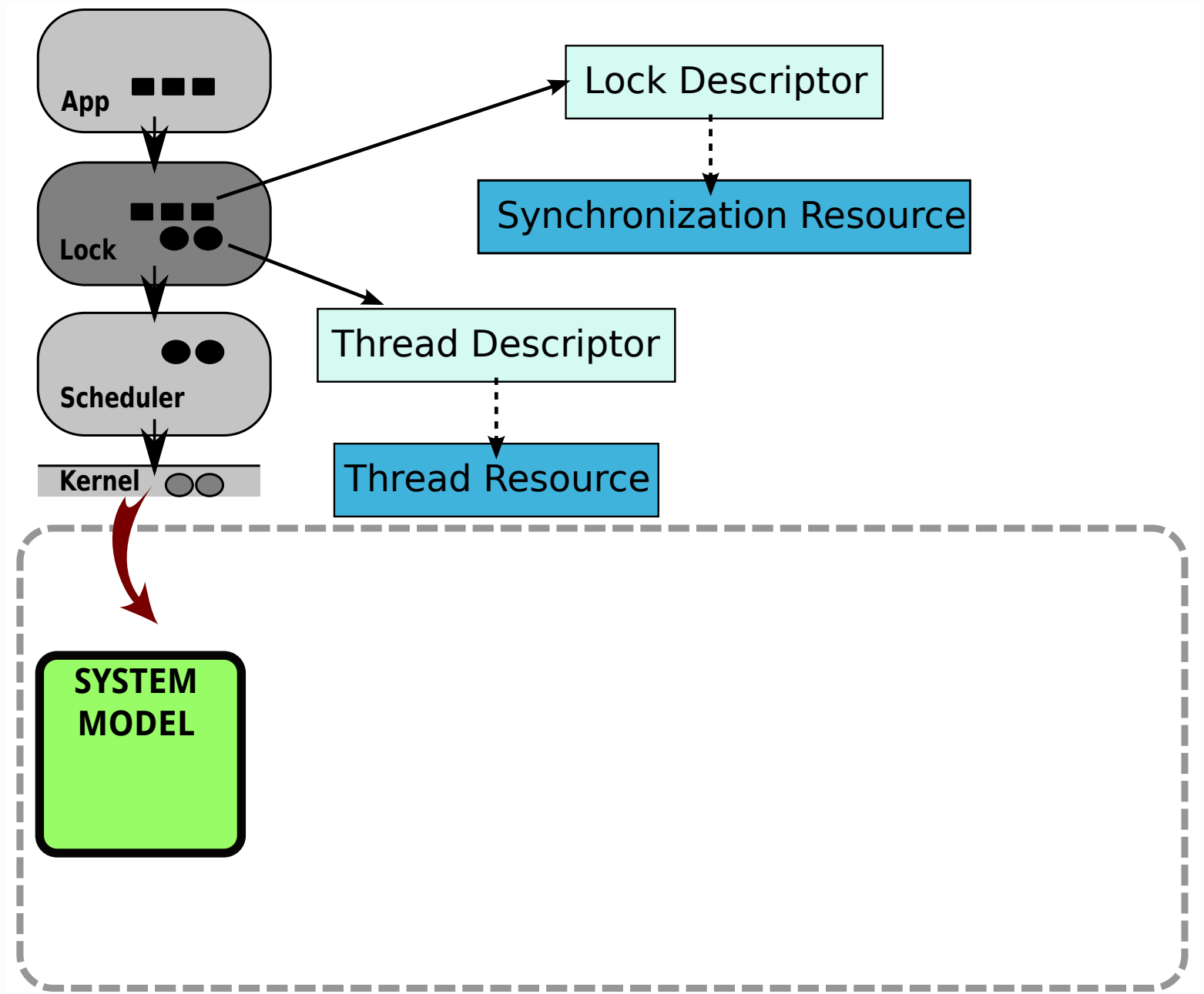
SuperGlue – IDL-based System-Level Fault Tolerance



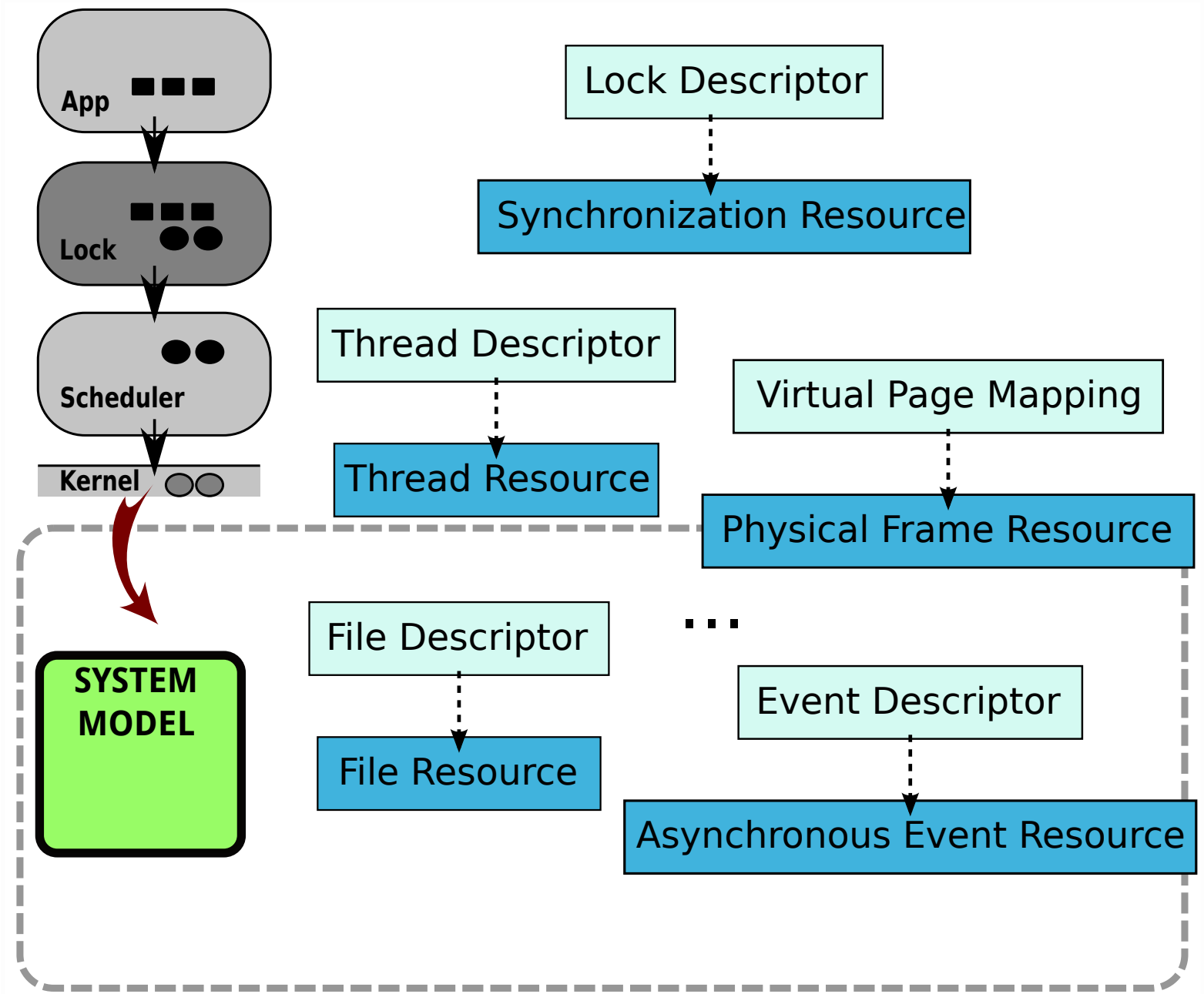
Descriptor-Resource Model



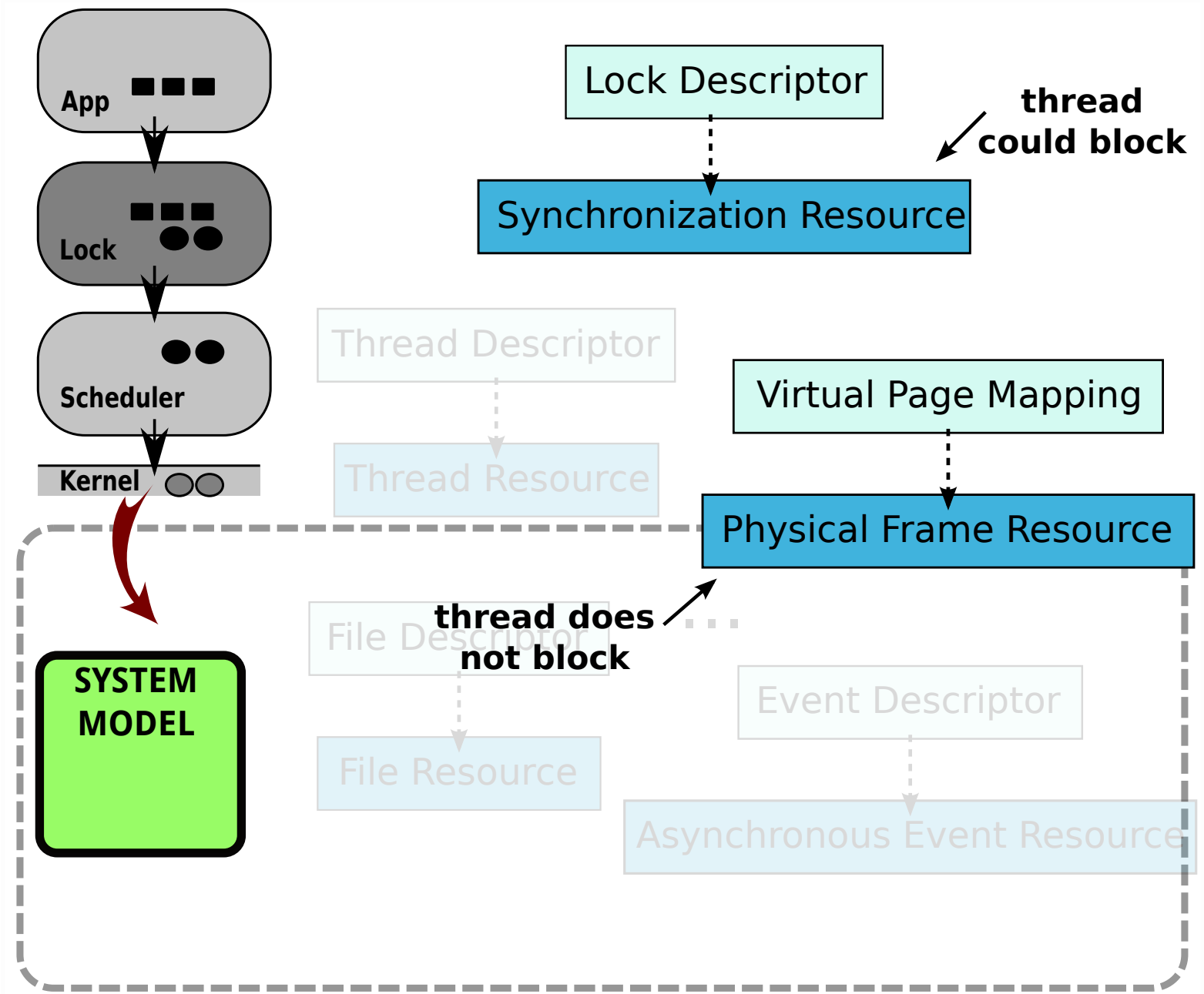
Descriptor-Resource Model



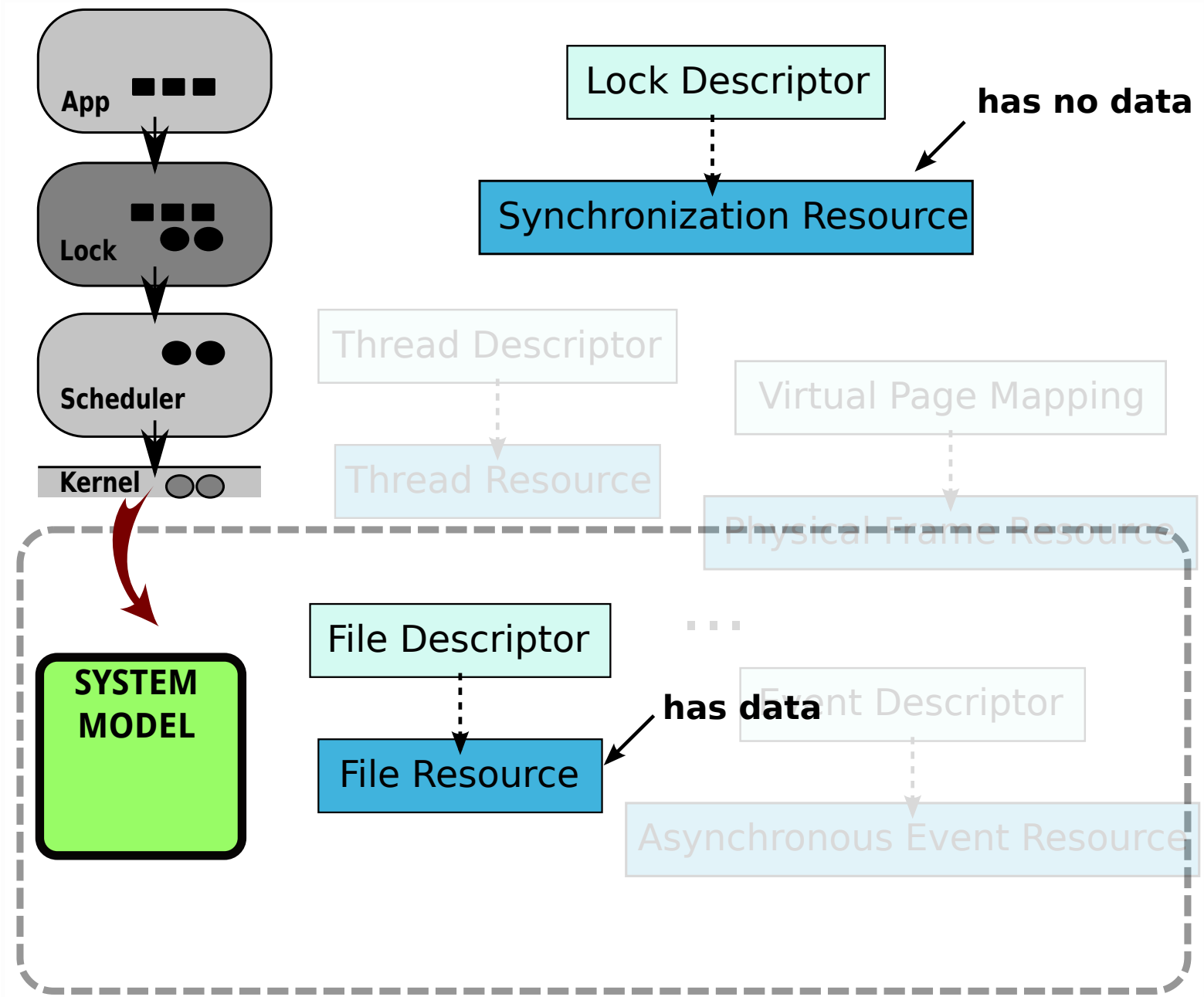
Descriptor-Resource Model



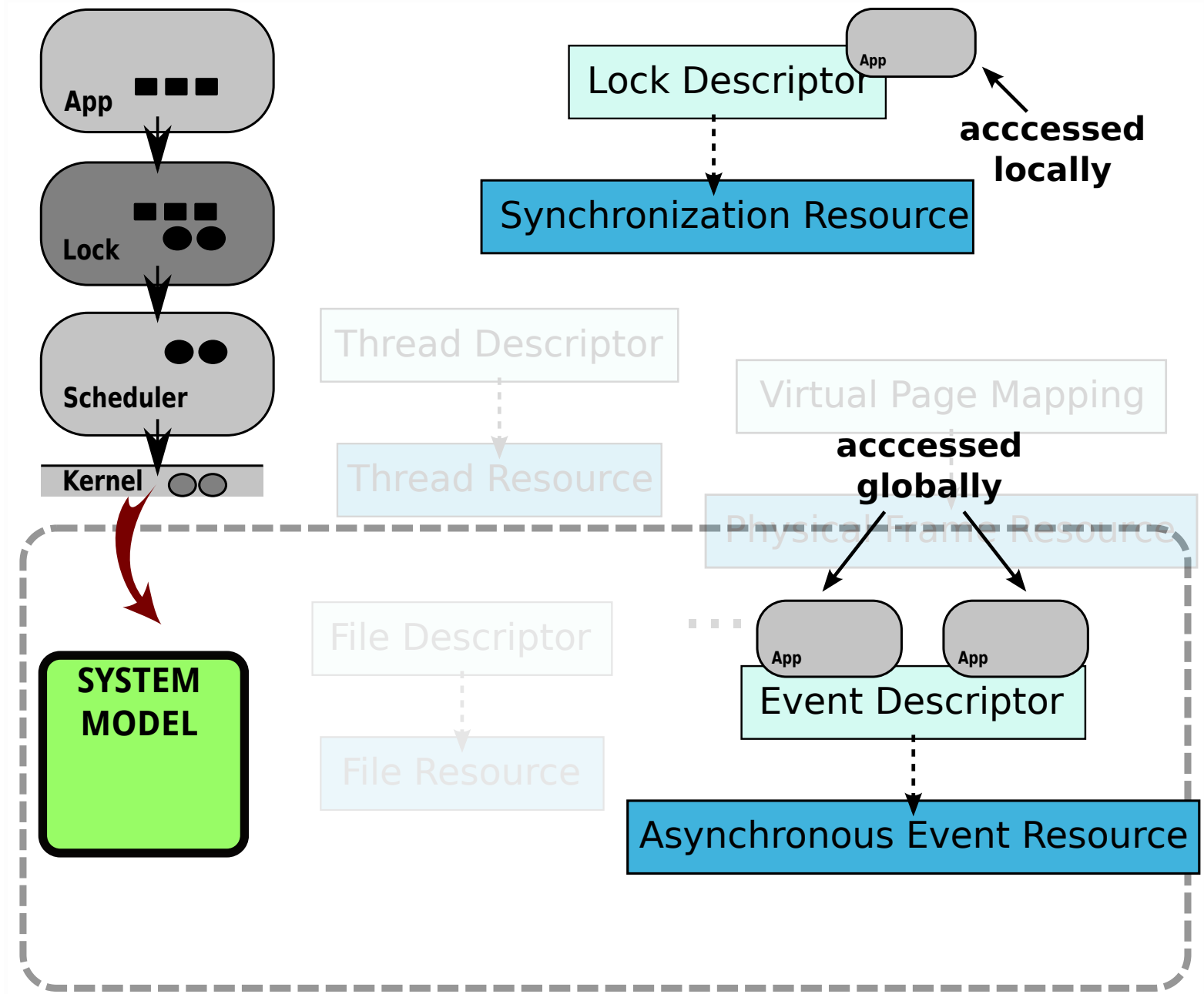
Descriptor-Resource Model



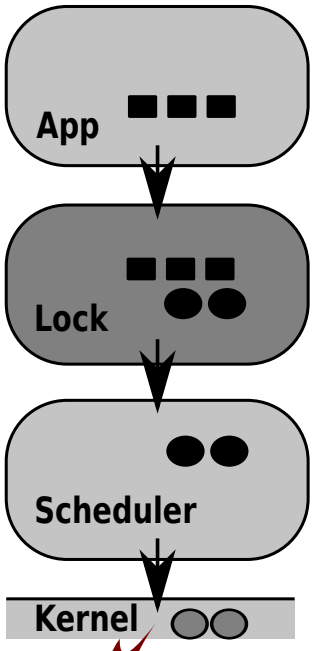
Descriptor-Resource Model



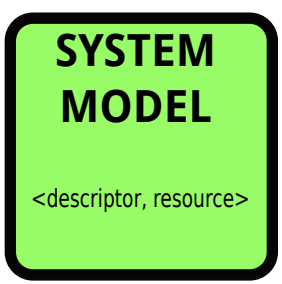
Descriptor-Resource Model



Descriptor-Resource Model



- **Resource**
  - e.g., synchronization, thread, file...
- **Descriptor**
  - e.g., lock, thread, file descriptor...

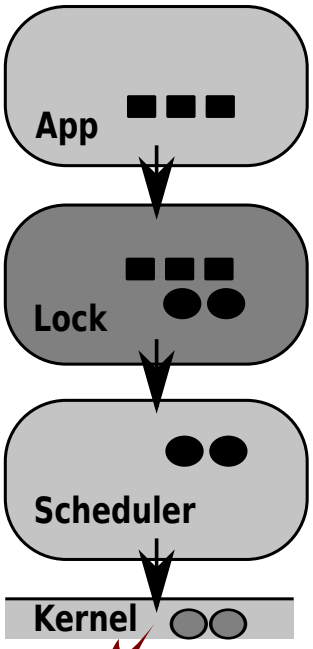


Descriptor-Resource Model --

$$(B_r, D_r, G_d, P_d, C_d, Y_d, D_d)$$

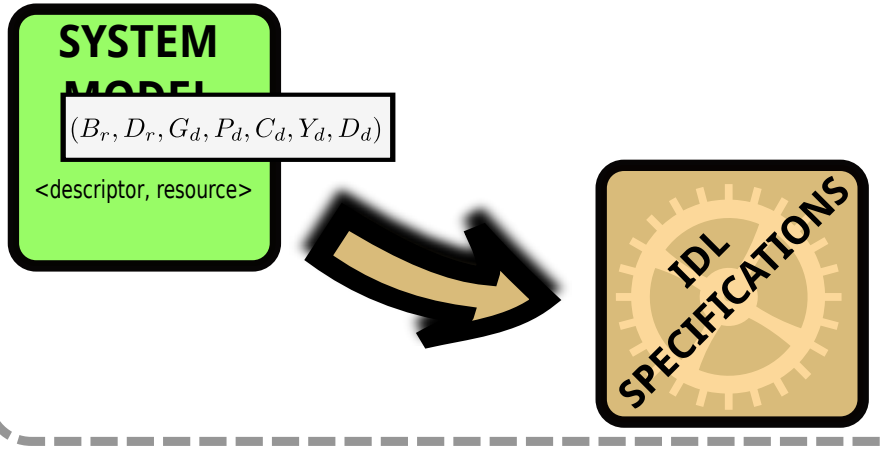
B_r : block when access resource
D_r : resource has meta data
G_d : can be accessed globally
P_d : has dependency descriptor
C_d : delete children when close
Y_d : need to delete tracking data
D_d : descriptor has meta data

Descriptor-Resource Model → IDL

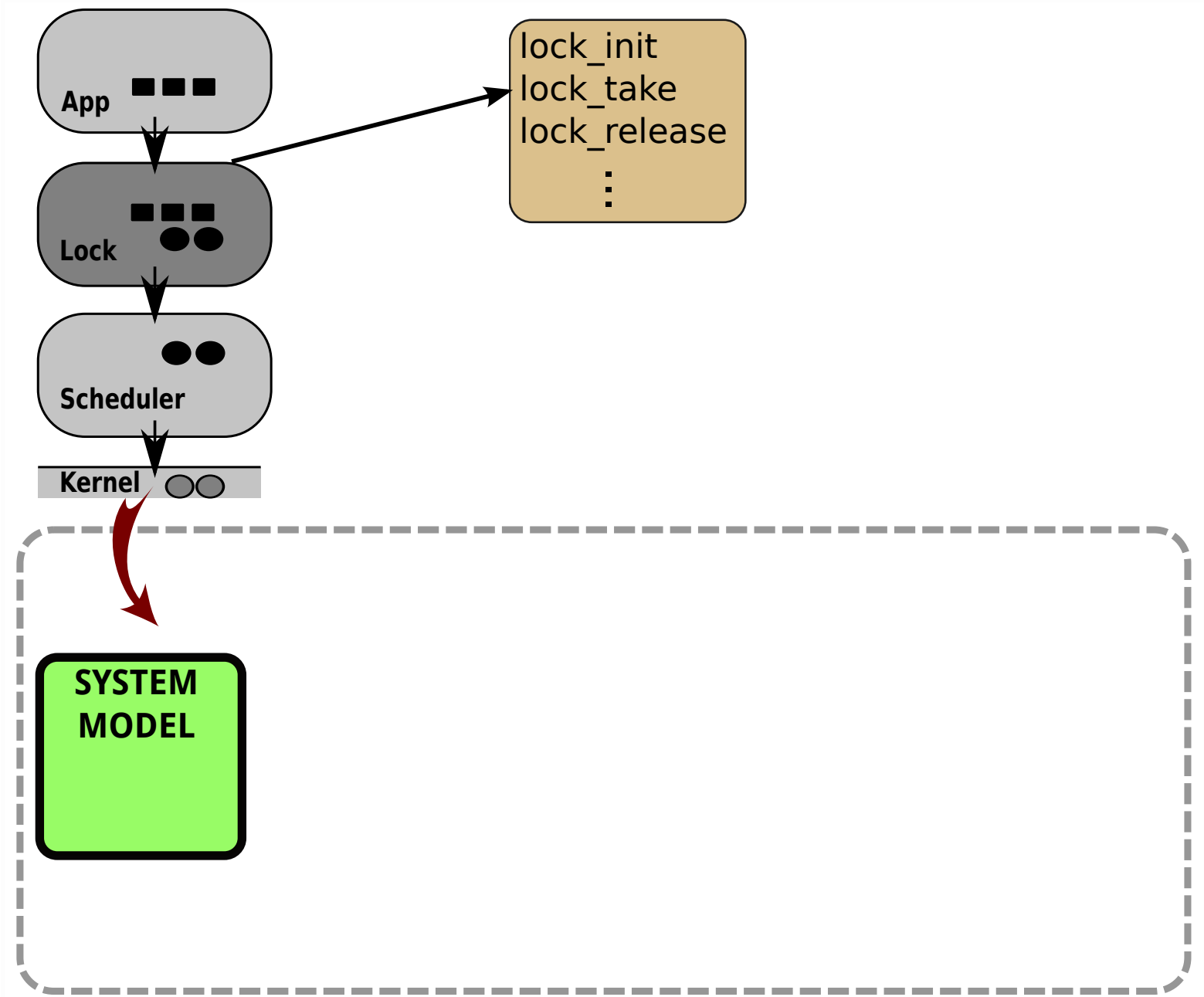



```
service_global_info = {
        service                 = lock,

        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,
};
```

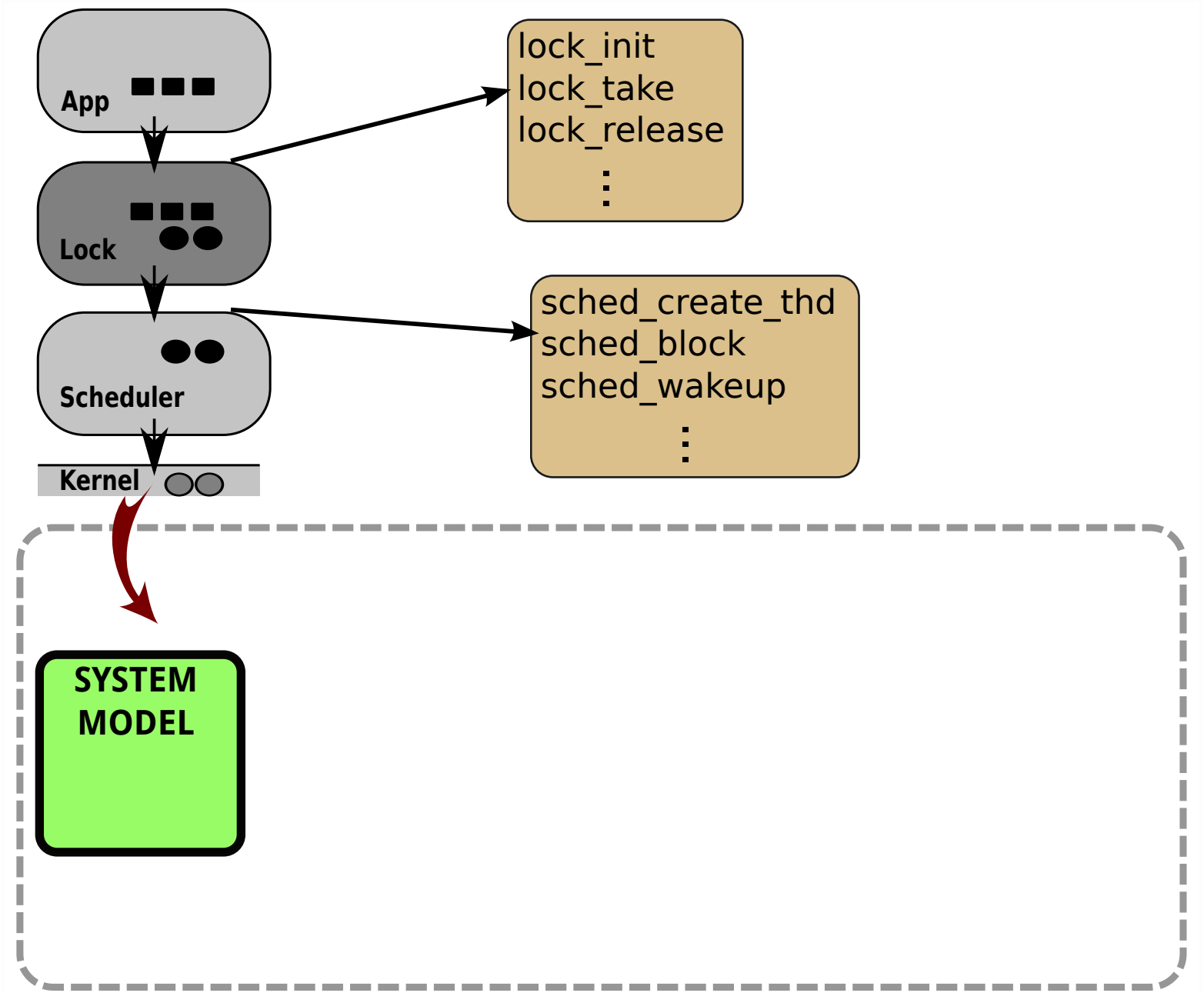
$$(B_r, D_r, G_d, P_d, C_d, Y_d, D_d)$$



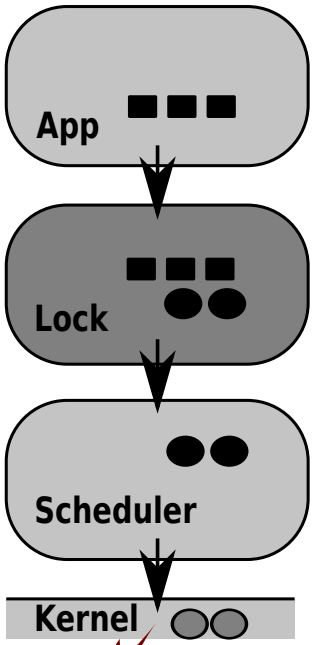
Component Interface Function



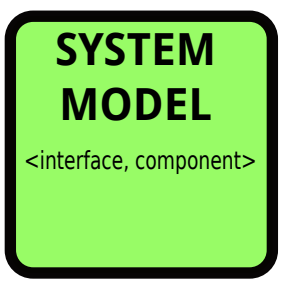
Component Interface Function



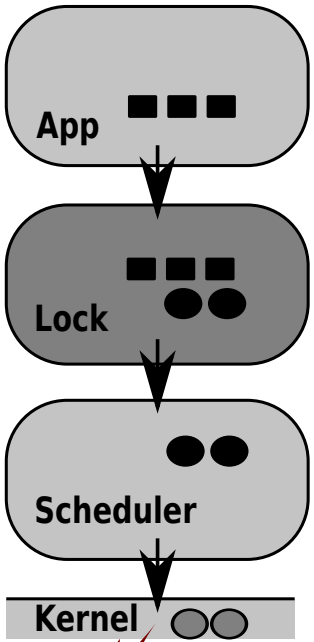
Component Interface Function

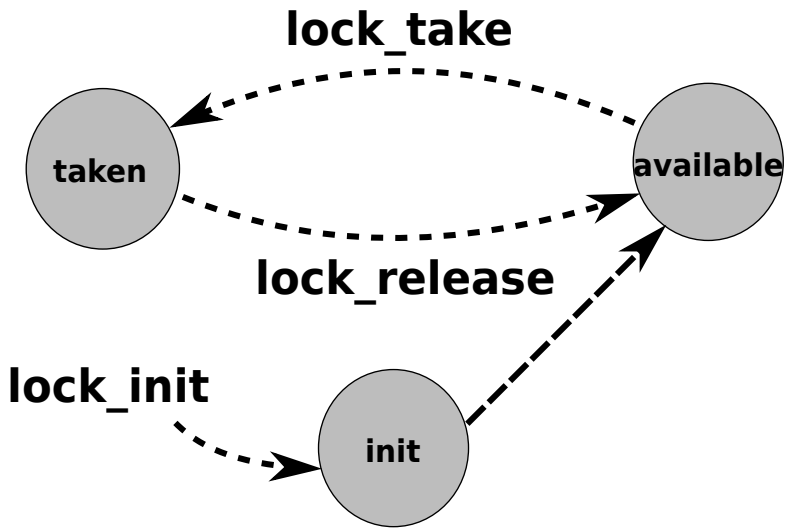


- **create descriptor** (e.g., lock_init)
- **terminate descriptor** (e.g., lock_free)
- **block invoking thread** (e.g., lock_take)
- **wake up thread** (e.g., lock_release)



Descriptor State Machine





lock descriptor state machine

SYSTEM MODEL

<interface, component>
<descriptor, resource>
<state machine>

Descriptor State Machine --

$$(I_d, S_d, s_0, s_f)$$

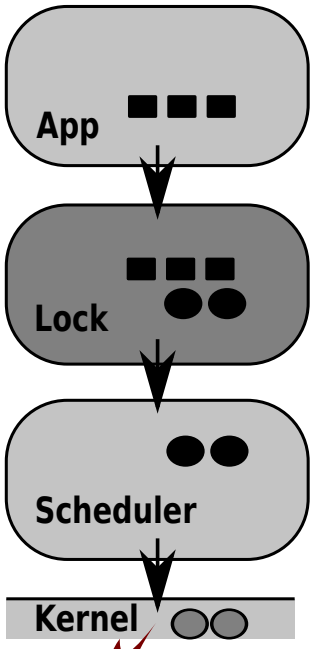
I_d : the set of interface functions
-- creation function
-- termination function
-- block function
-- wake up function

S_d : the set of descriptor states

s_0 : initial state

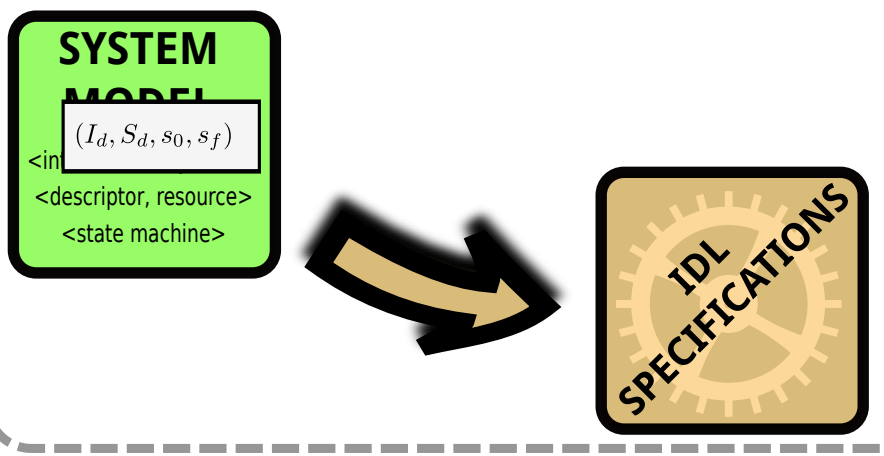
s_f : fault state

Descriptor State Machine → IDL

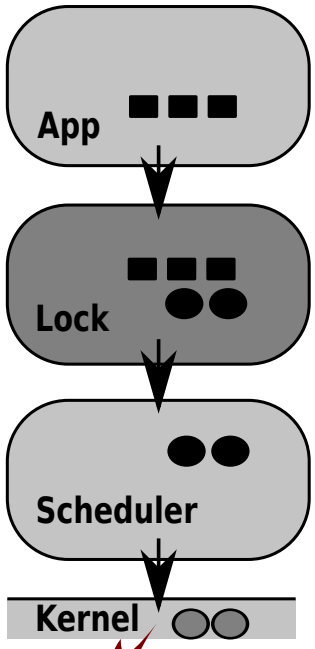



```
sm_creation(lock_component_alloc);
sm_transition(lock_component_alloc, lock_component_free);
sm_transition(lock_component_alloc, lock_component_pretake);
sm_transition(lock_component_pretake, lock_component_take);
sm_transition(lock_component_take, lock_component_release);
sm_transition(lock_component_release, 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);
```

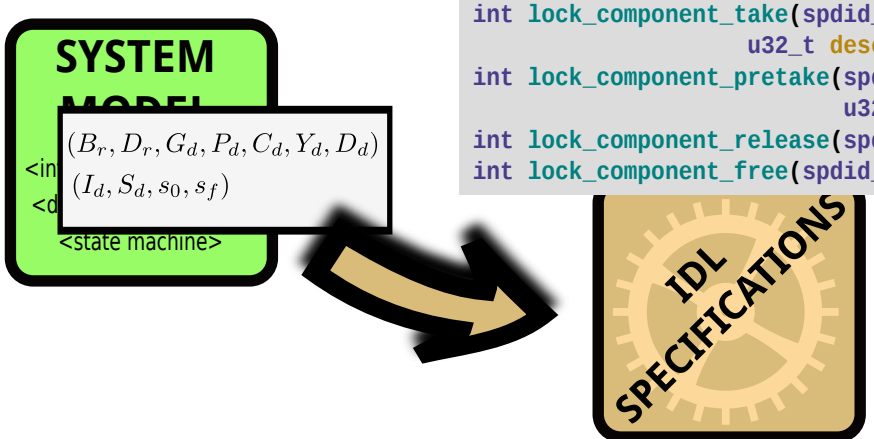
(I_d, S_d, s_0, s_f)



IDL-based Specification

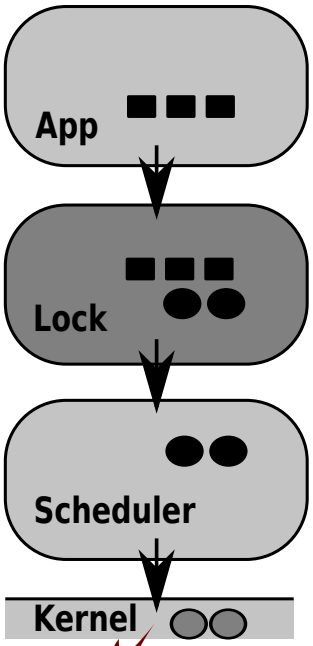



```
service_global_info = {
        service                 = lock,

        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));
```



Avg LOC = 37

Toward Generating Recovery Code

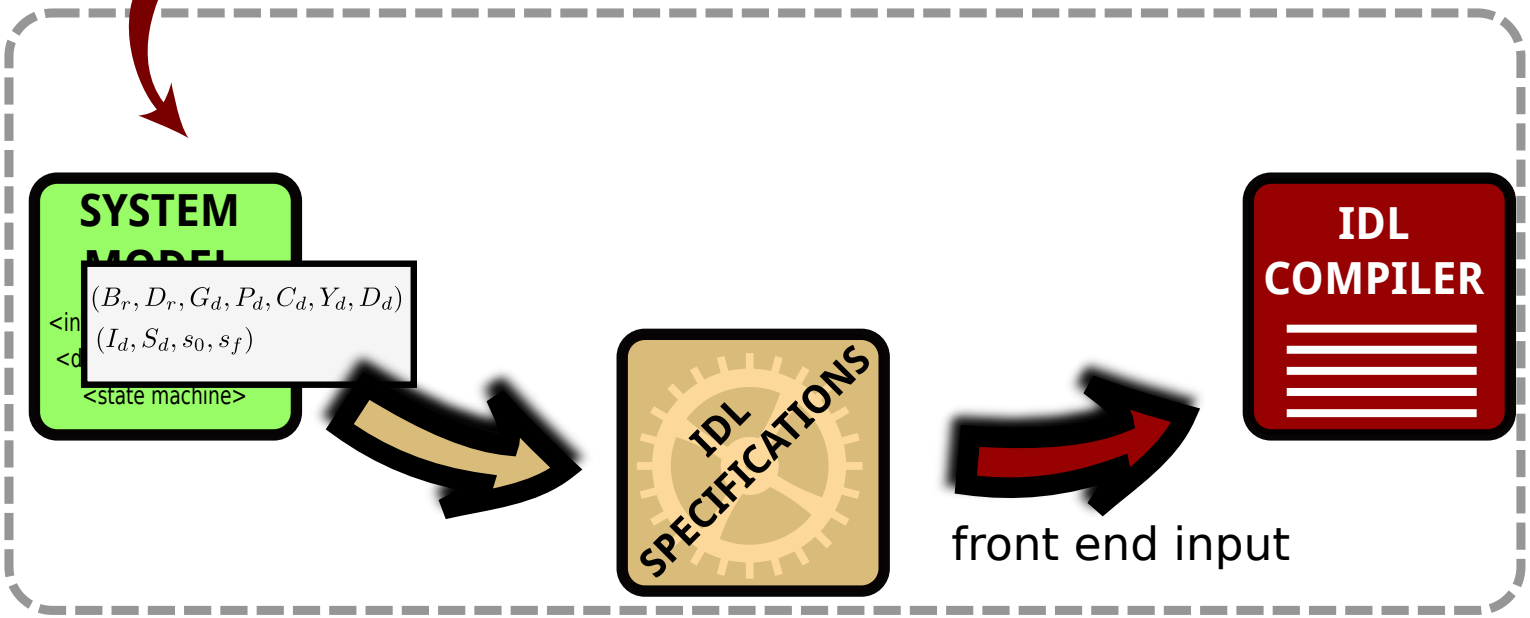


front end

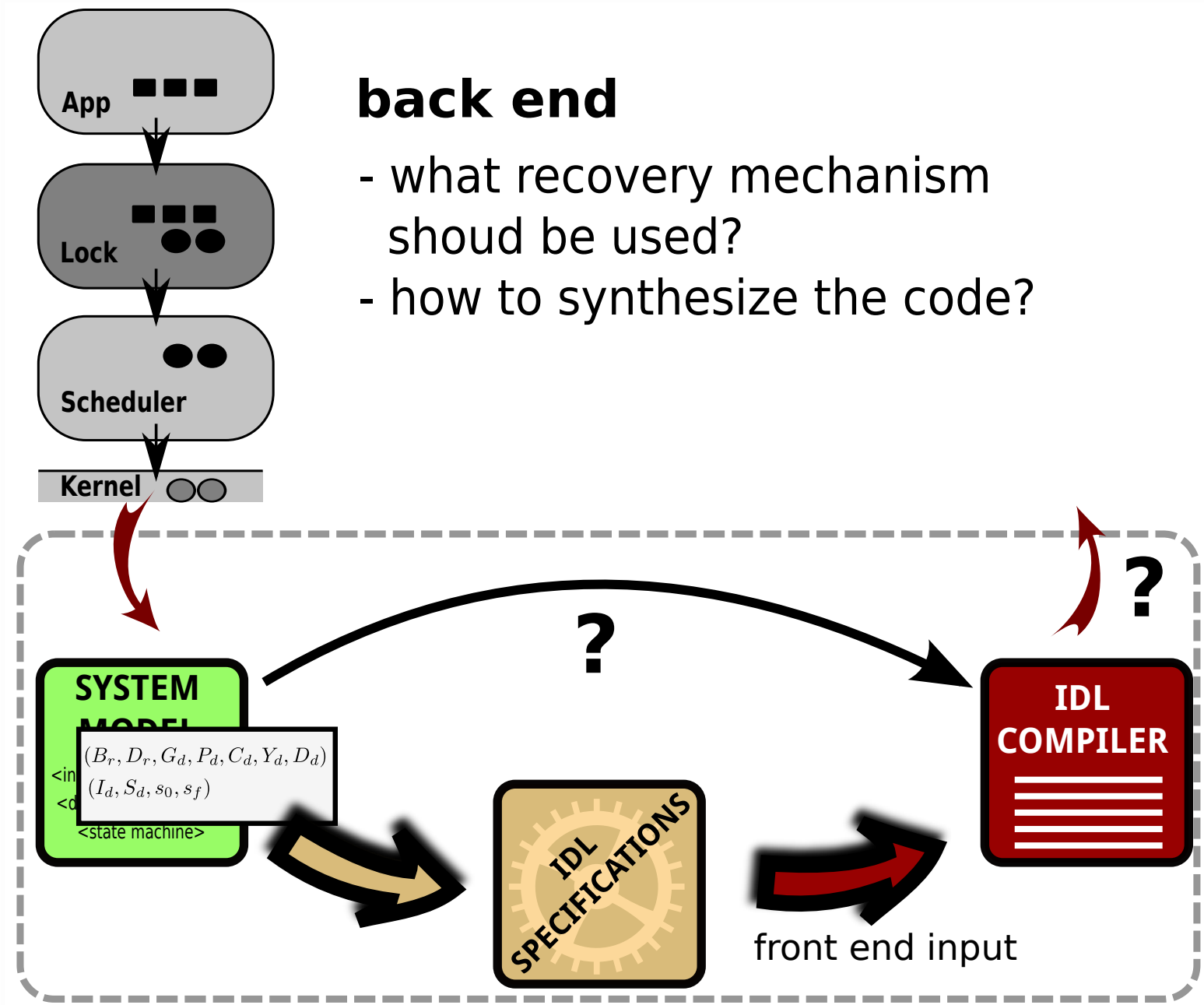
- parse IDL specification

intermediate representation

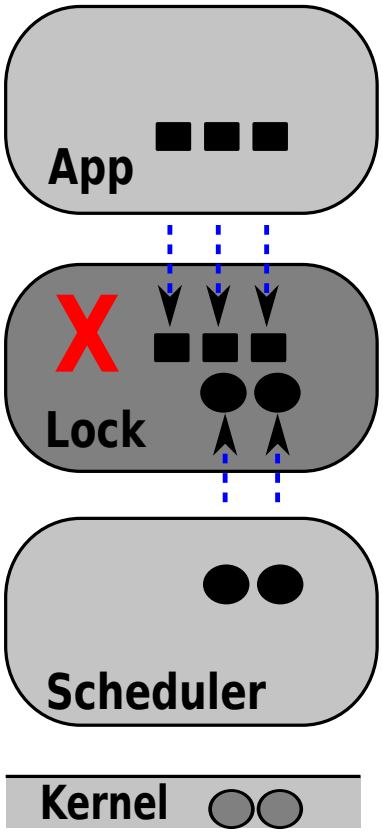
- encode the model information



Toward Generating Recovery Code



Interface-Driven Recovery Mechanisms



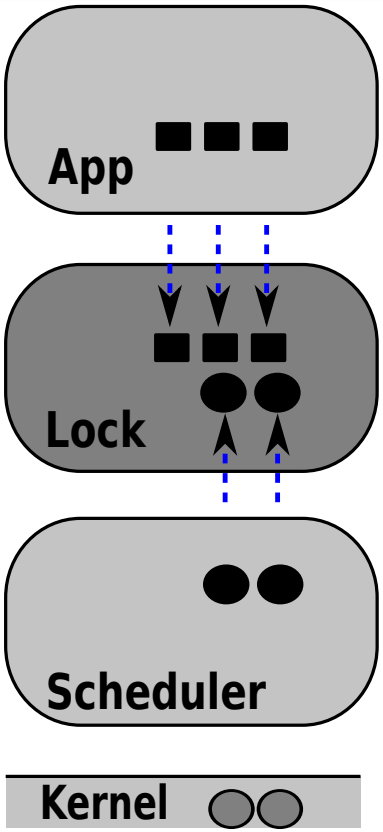
basic recovery

-- through component operation (**always**)

---

basic recovery

Interface-Driven Recovery Mechanisms



basic recovery

-- through component operation (**always**)

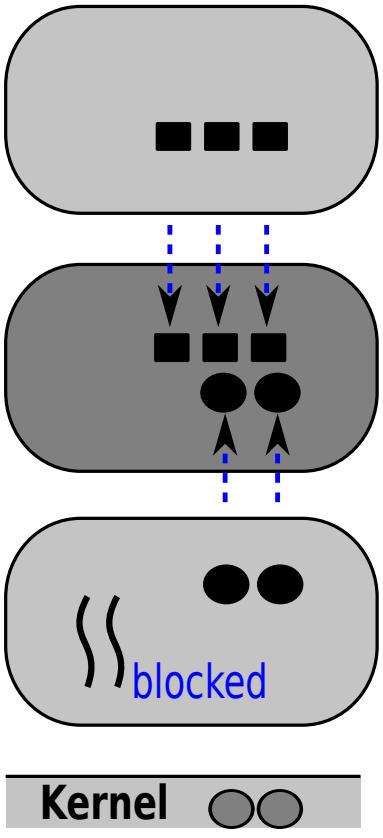
timing of recovery - on-demand

-- recover only when accessed (**always**)

---

basic recovery | timing of recovery

Interface-Driven Recovery Mechanisms



basic recovery

-- through component operation (**always**)

timing of recovery - on-demand

-- recover only when accessed (**always**)

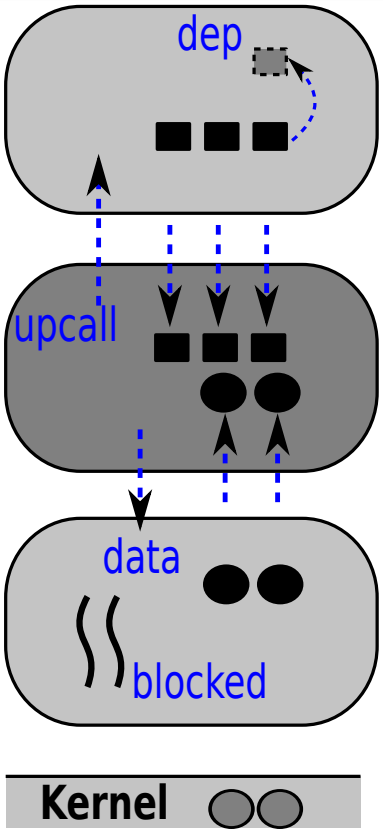
timing of recovery - eager

-- eagerly wake up blokcing threads (B_r)

---

basic recovery | timing of recovery

Interface-Driven Recovery Mechanisms



basic recovery

-- through component operation (**always**)

timing of recovery - on-demand

-- recover only when accessed (**always**)

timing of recovery - eager

-- eagerly wake up blokcing threads (B_r)

recovery with dependency

-- require to recontruct parent (P_d)

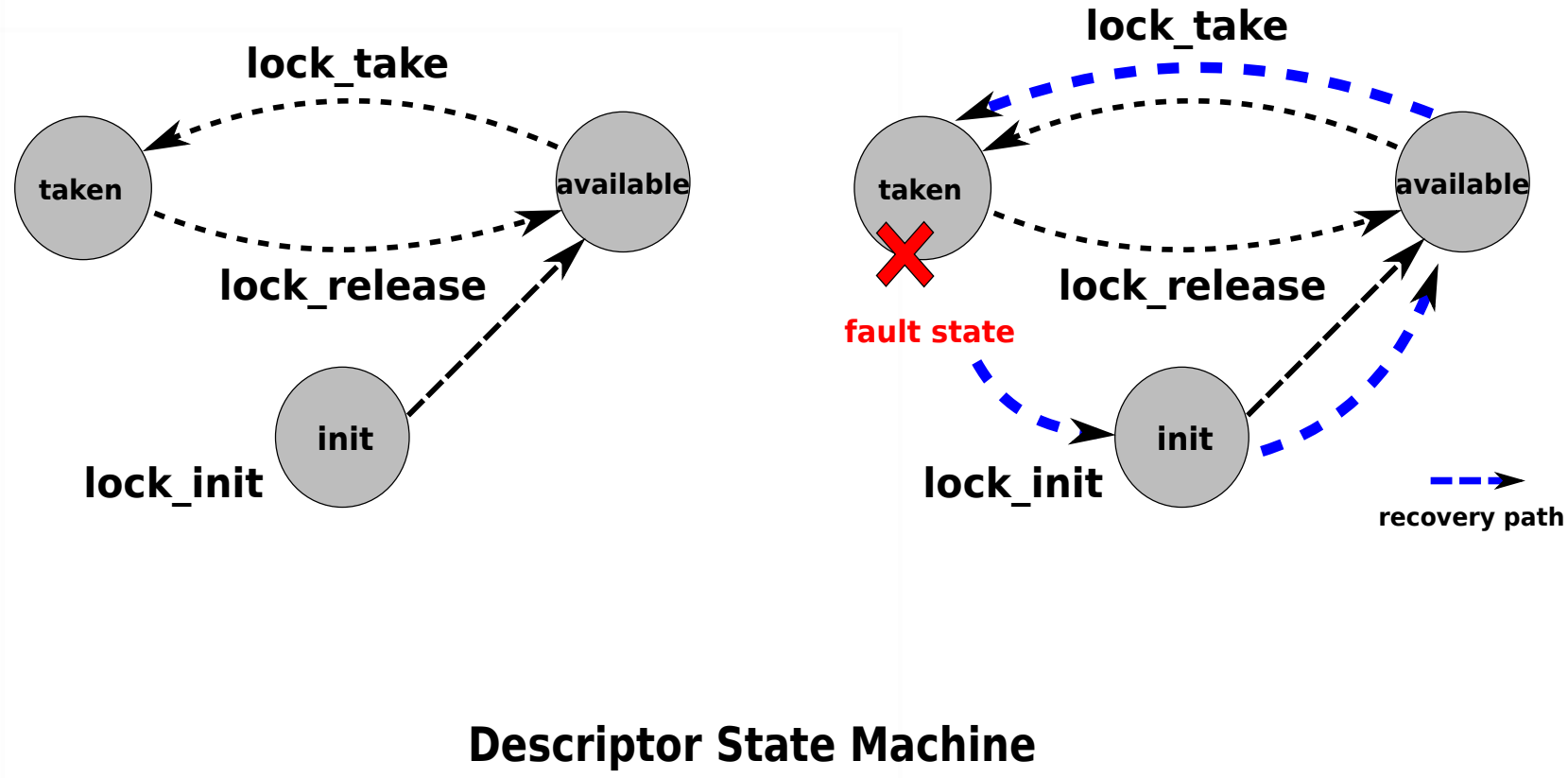
-- require to recontruct children (C_d)

⋮

---

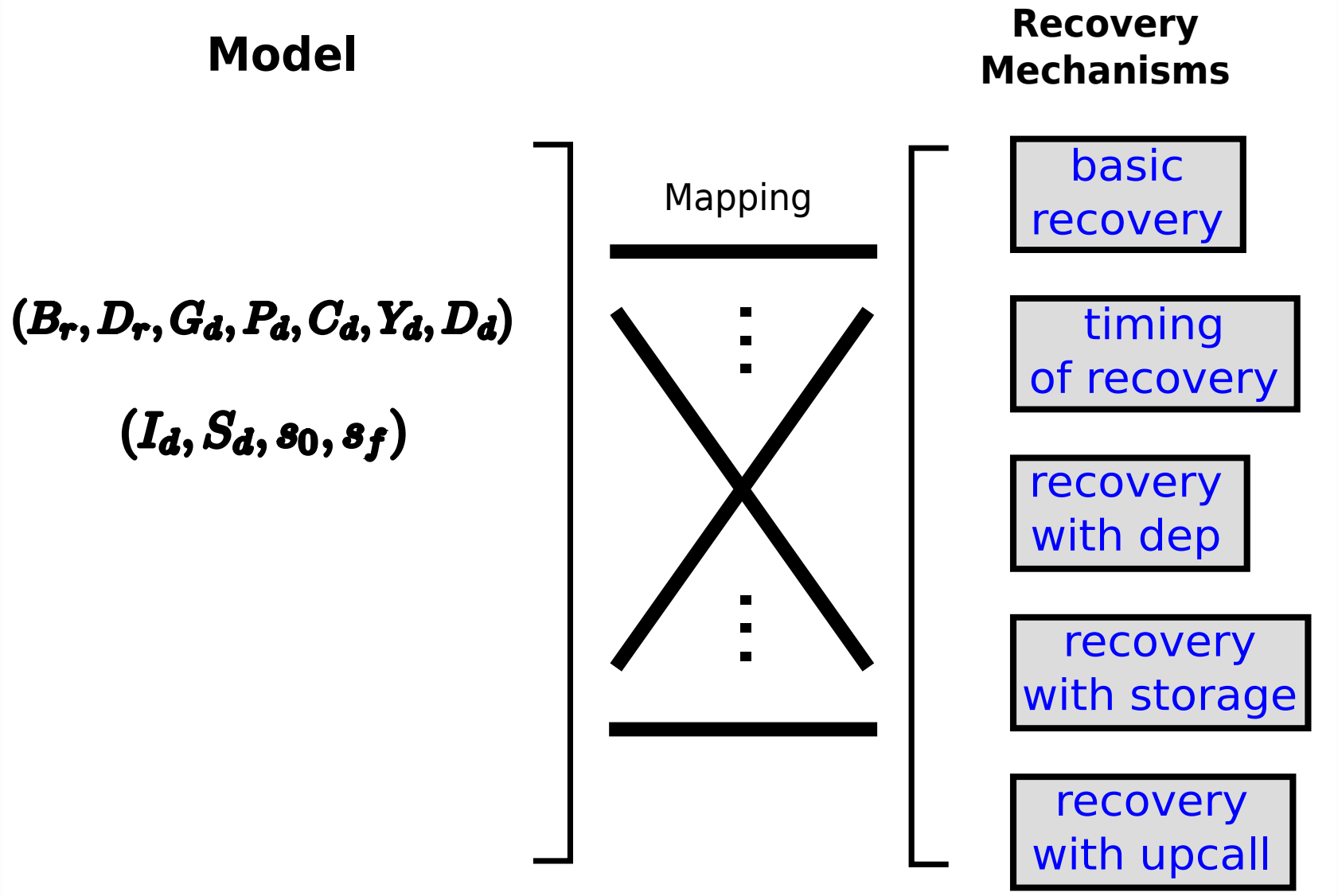
basic recovery	timing of recovery	recovery with dep	recovery with storage	recovery with upcall

Interface-Driven Recovery Mechanisms



Descriptor State Machine

Model → Recovery Mechanisms

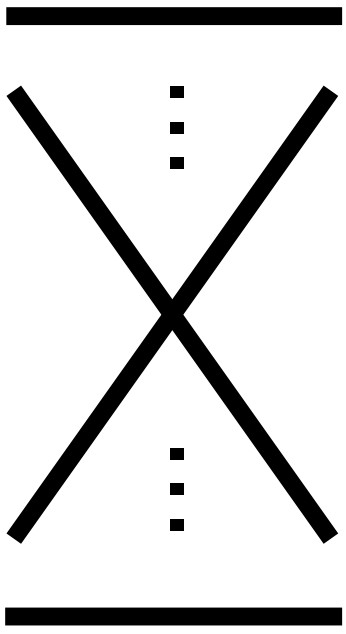


Synthesize the Recovery Code

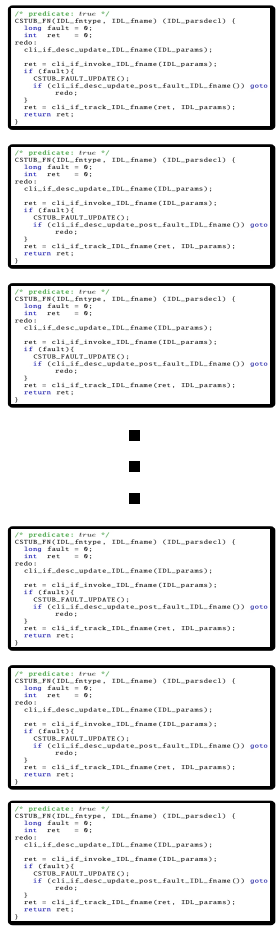
Predicates

$$\vdots$$
$$true$$
$$create\_fn \wedge \neg G_d$$
$$wakeup\_fn \wedge B_r$$
$$block\_fn \wedge B_r$$
$$terminate\_fn \wedge C_d$$
$$G_d \wedge P_d \wedge C_d \wedge D_d$$
$$D_r \wedge \neg P_d \wedge Y_d$$
$$\vdots$$

Mapping



Templates



Synthesize the Recovery Code

Predicates **Templates**

$true$

$create\_fn \wedge$

$wakeup\_fn \wedge$

$block\_fn \wedge$

$terminate\_f$

$G_d \wedge P_d \wedge C$

$D_r \wedge \neg P_d \wedge$

Mapping

```
/* 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;
}
```

Synthesize the Recovery Code

Predicates **Templates**

$true$

$create\_fn \wedge$

$wakeup\_fn \wedge$

$block\_fn \wedge$

$terminate\_f$

$G_d \wedge P_d \wedge C$

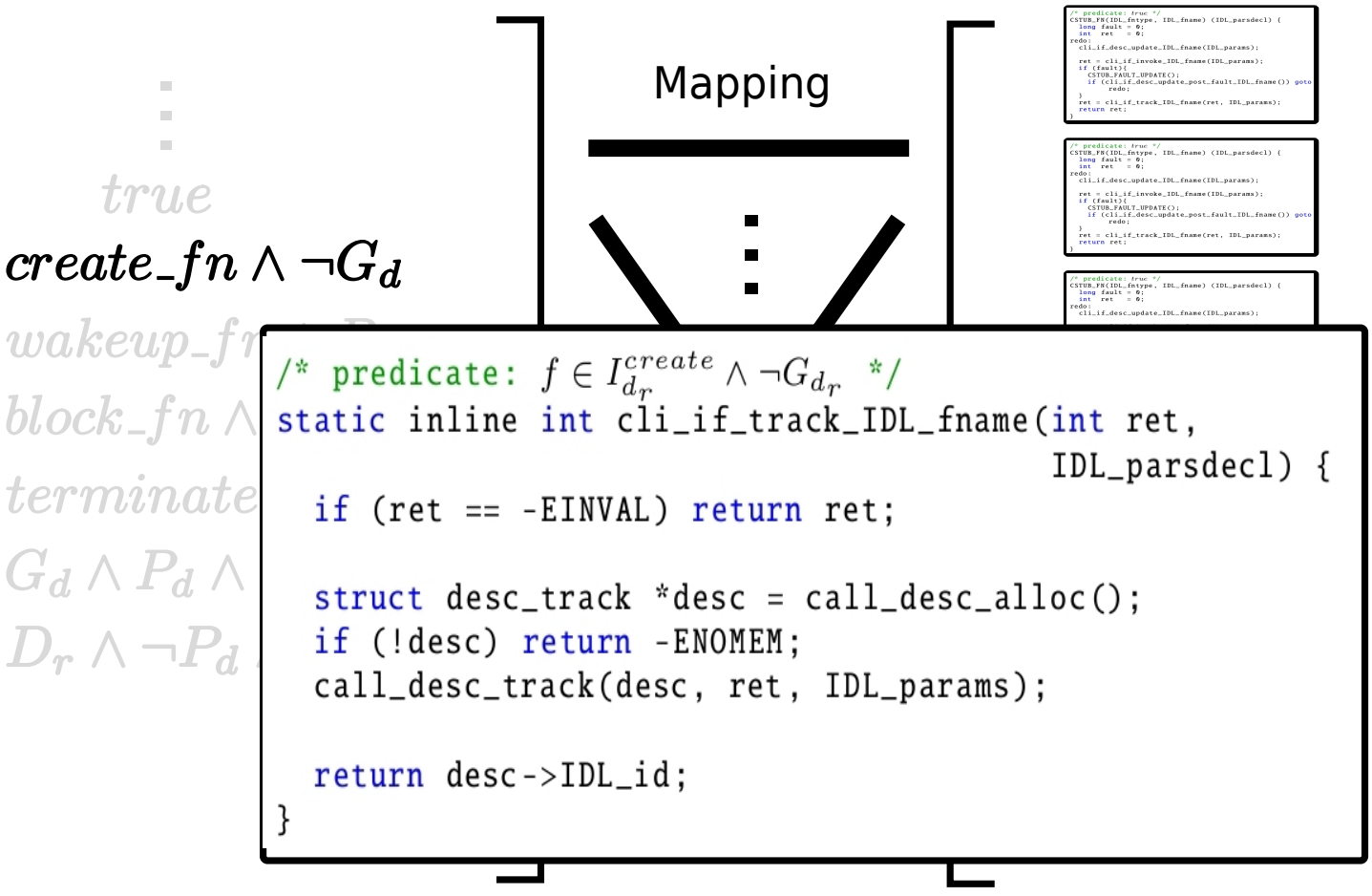
$D_r \wedge \neg P_d \wedge$

Mapping

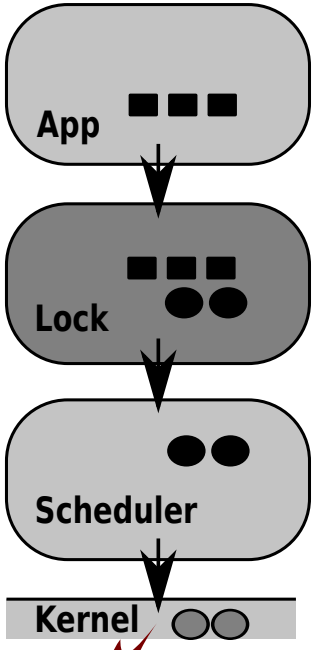
```
/* predicate: true */
CSTUB_FN(IDL_fntype, IDL_fname) (IDL_parsdecl) {
  long fault = 0;
  int  ret   = 0;
redo:
  cli_if_desc_update_IDL_fname(IDL_params);

  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;
}
```

Synthesize the Recovery Code

Predicates **Templates**



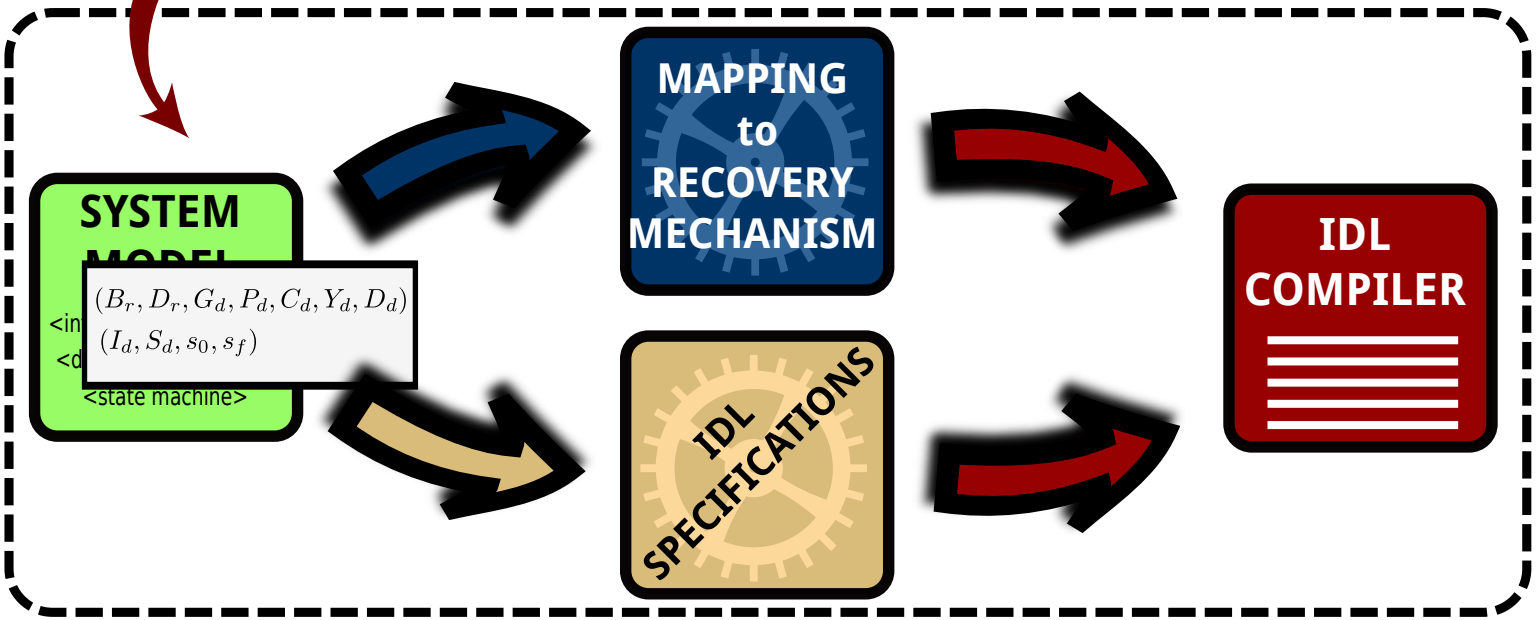
Generate Recovery Code



front end
- parse IDL-based specification

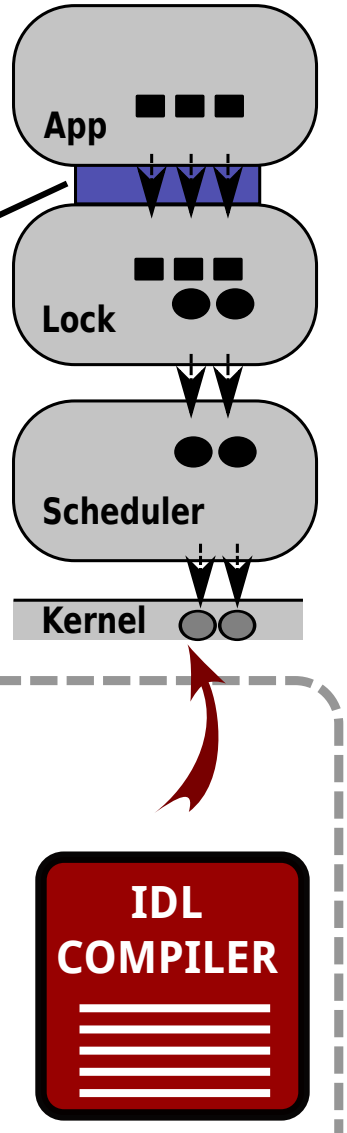
intermediate representation
- encode the model information

back end
- evaluate predicates
- generate the code from templates

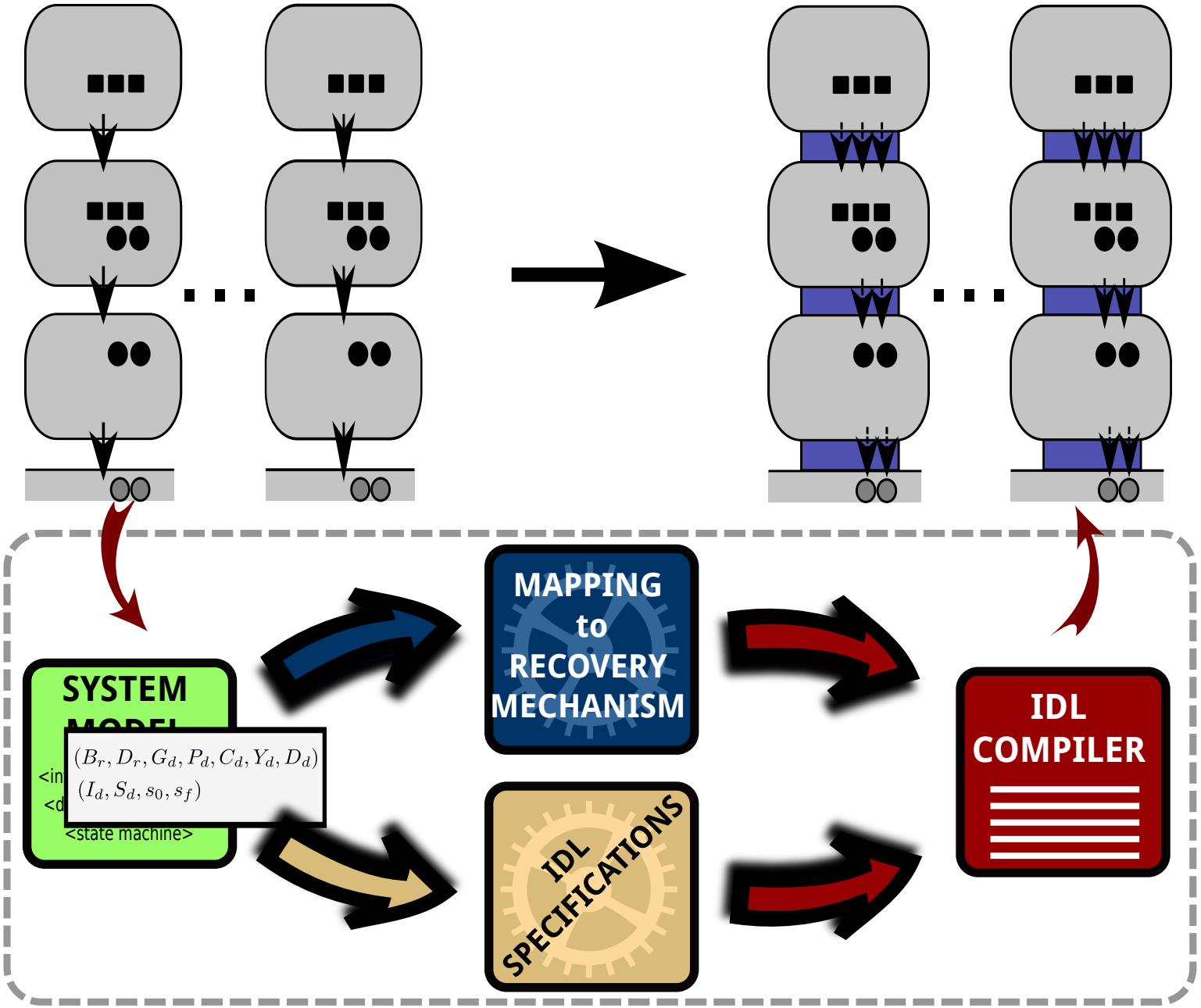


Generate Recovery Code

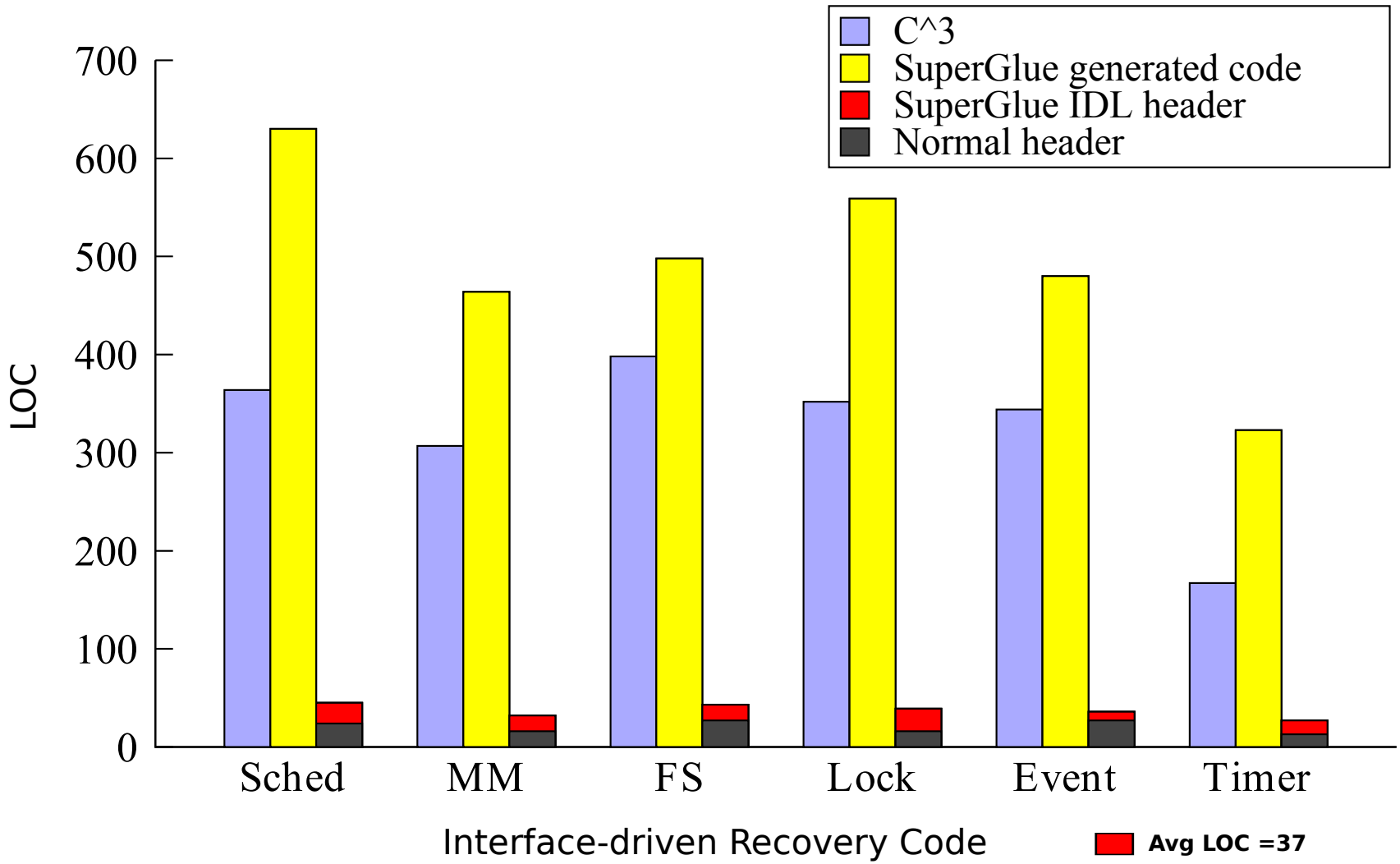
```
        desc->server_lock_id = new_desc->server_lock_id;
        desc->state = state_lock_component_alloc;
        desc->fault_cnt = global_fault_cnt;

        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;
                }
```



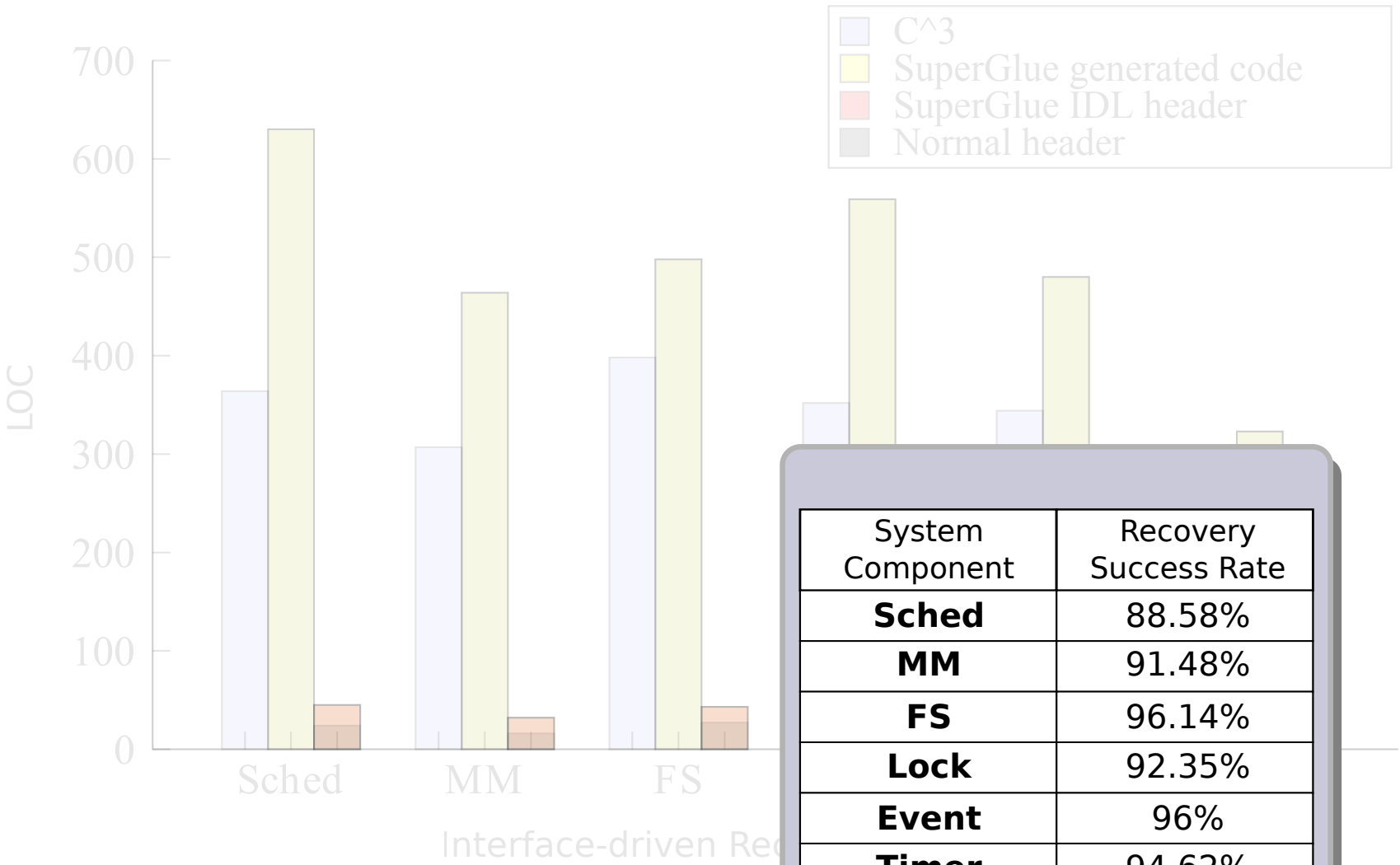
Generate Recovery Code



Code Generation Result



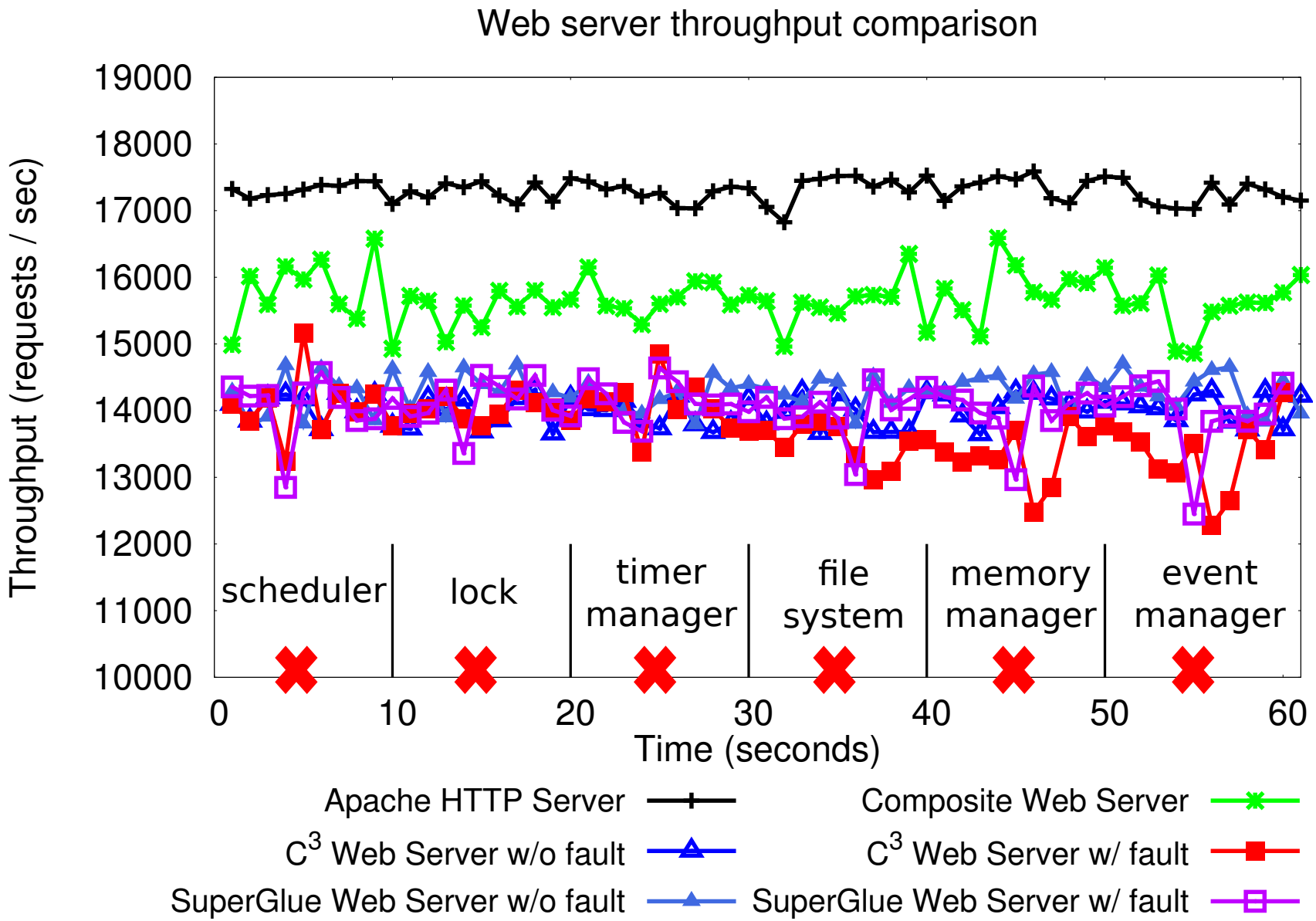
Fault Injection Result



System Component	Recovery Success Rate
Sched	88.58%
MM	91.48%
FS	96.14%
Lock	92.35%
Event	96%
Timer	94.62%

SWIFI-based Fault Injection Campaign

Web Server Evaluation with Injected Faults



Conclusion on SuperGlue

SuperGlue – code generation for system-level fault tolerance

- descriptor-resource model and descriptor state machine
- IDL-based declarative specifications
- compiler for synthesizing C^3-style recovery code

Publication

J. Song, G. Bloom, and G. Parmer, "SuperGlue: IDL-Based, System-Level Fault Tolerance for Embedded Systems " in DSN, 2016

Medical equipment (Merge Hemo) crashed in Feb 2016

- Equipment went black and manually rebooted
- 5 minutes delay

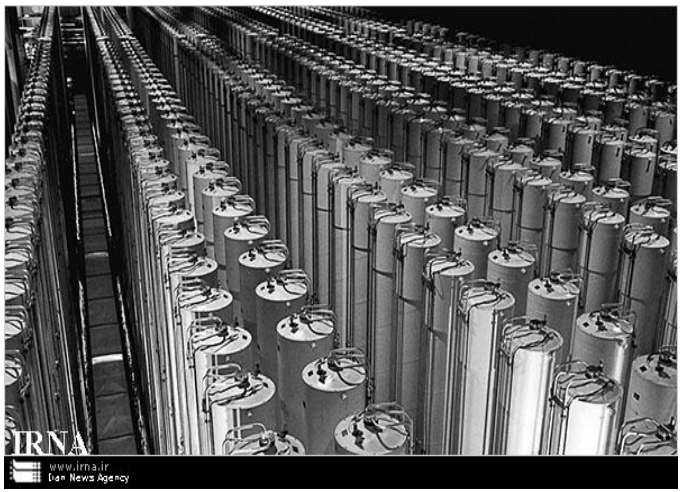
https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfmaude/detail.cfm?mdrfoi__id=5487204

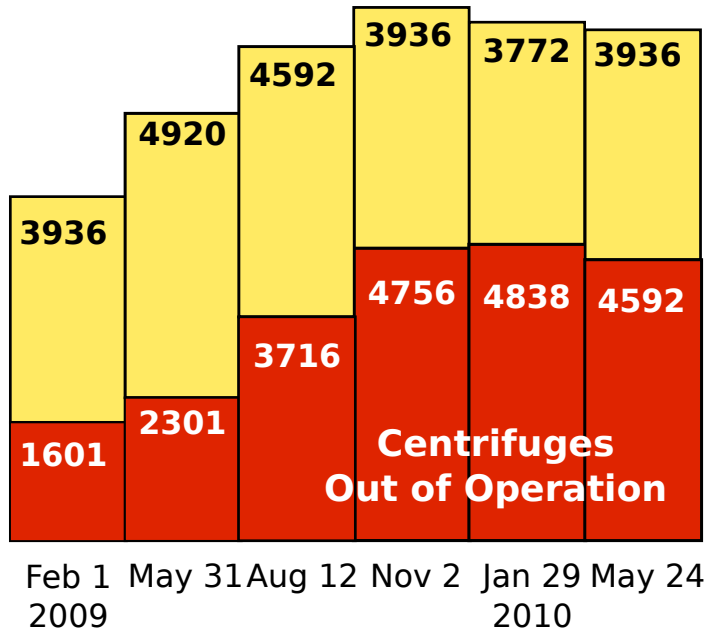
Medical equipment (Merge Hemo) crashed in Feb 2016

- Equipment went black and manually rebooted
- 5 minutes delay

https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfmaude/detail.cfm?mdrfoi__id=5487204

The World's First Cyber Weapon: Stuxnet

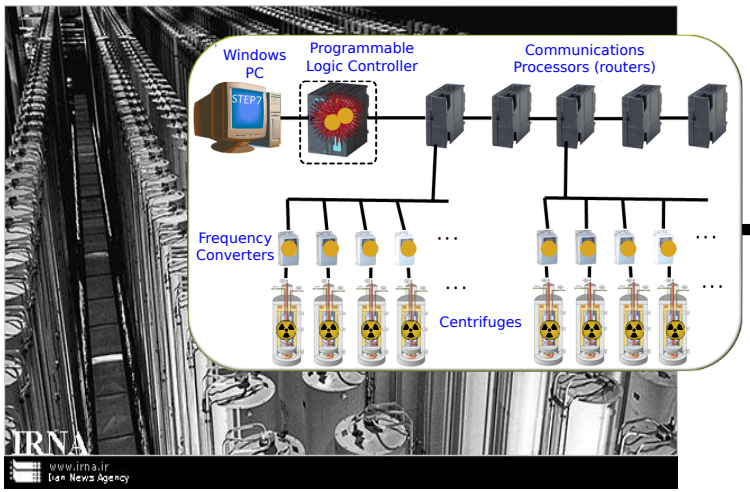


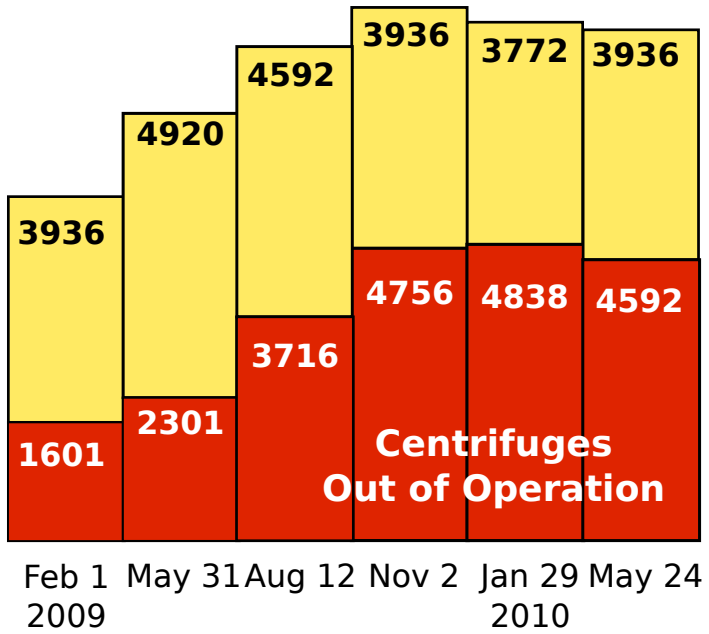


- Physical infrastructure destruction
- Could raise nuclear tensions between the nations

Adapted from "The 1-hour Guide to Stuxnet" presentation by Carey Nachenberg, Symantec Fellow

The World's First Cyber Weapon: Stuxnet





- Physical infrastructure destruction
- Could raise nuclear tensions between the nations

Adapted from "The 1-hour Guide to Stuxnet" presentation by Carey Nachenberg, Symantec Fellow

CAML– System-Level Anomaly Detection Infrastructure

CAML (*an ongoing collaborative project)*

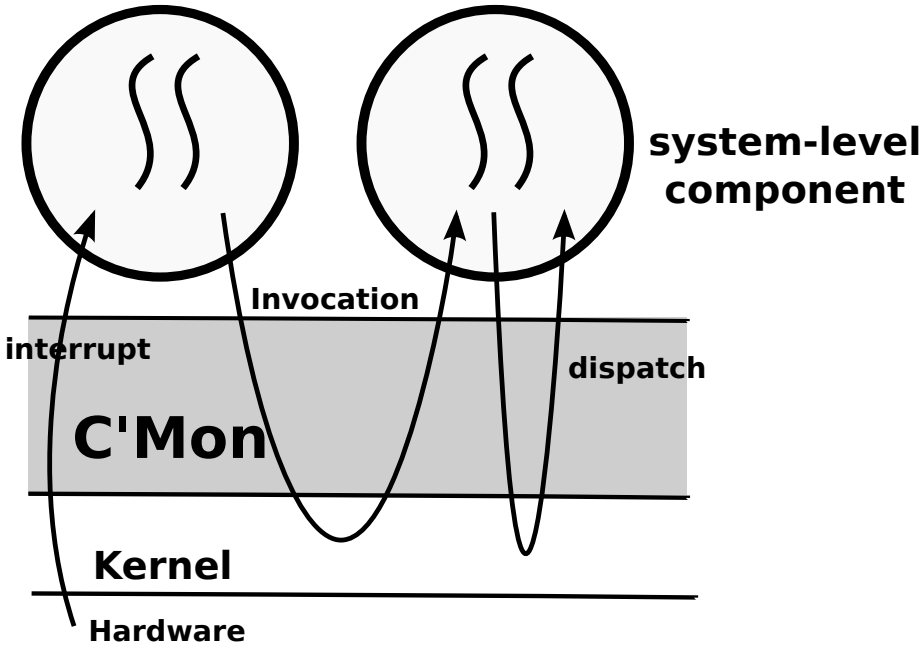
- system-level anomaly detection
- predictably
- using machine learning techniques
  (*by Charles River Analytics Inc)*

Main ideas

- Extend the system events logging infrastructure from C'Mon
- Pre-process logged events
- Use probabilistic techniques to detect the anomaly

C'Mon Review – System-Level Latent Fault Monitor

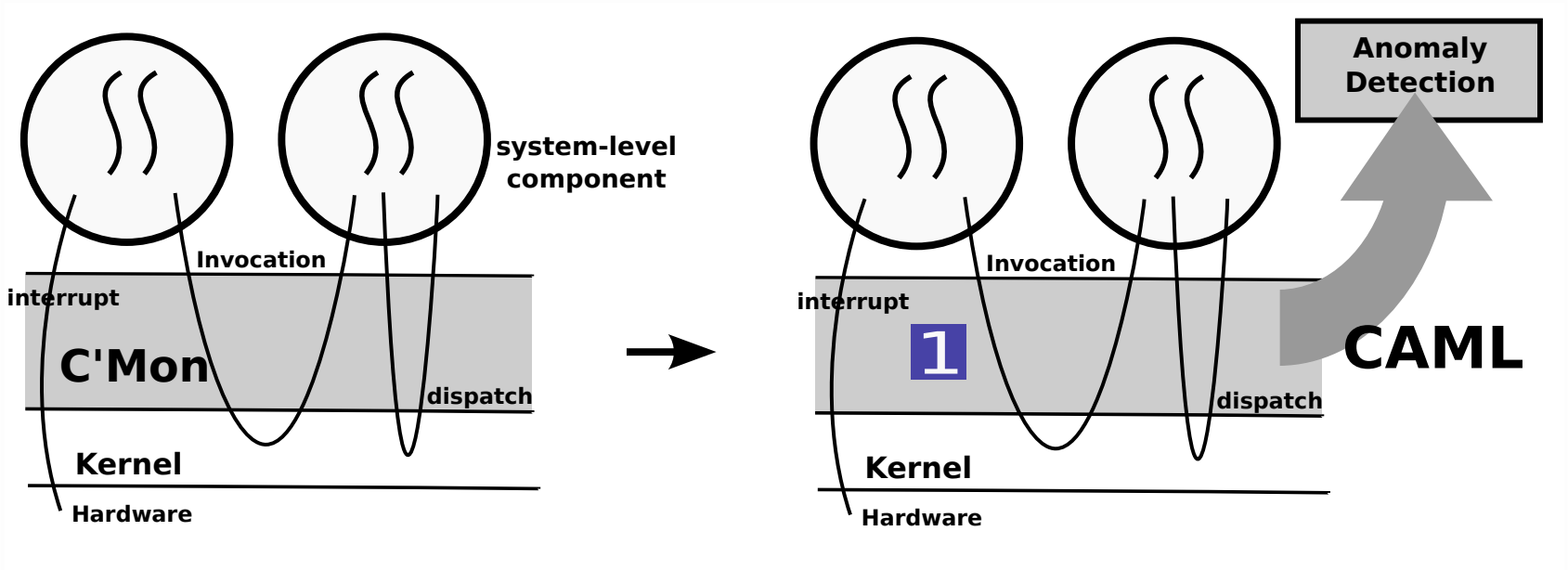
1. **Tracking**
   - system communication events
   - system timing information
2. **Detection**
   - temporal constraints violation
3. **User-level**
   - no kernel modification



Publication

J. Song, and G. Parmer, "C'Mon: a Predictable Monitoring Infrastructure for System-Level Latent Fault Detection and Recovery", in RTAS 2015

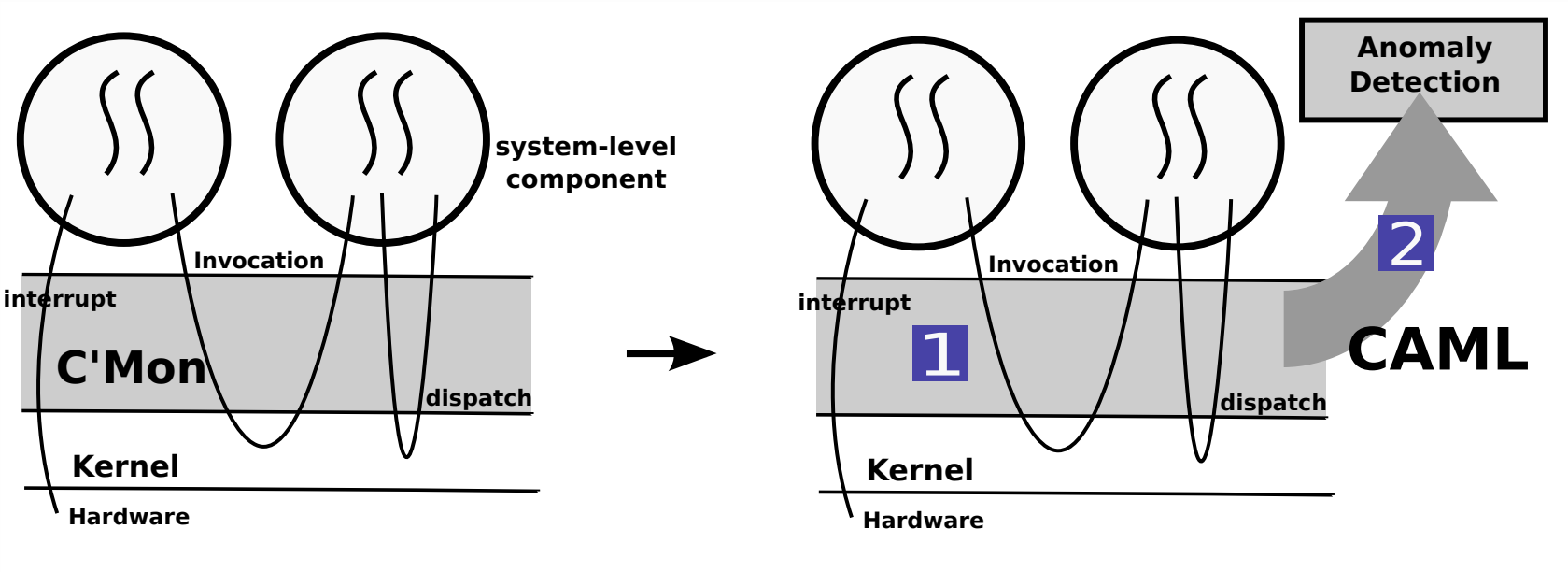
CAML– System-Level Anomaly Detection



Idea $\rightarrow$

1. logging events (same as in C'Mon)

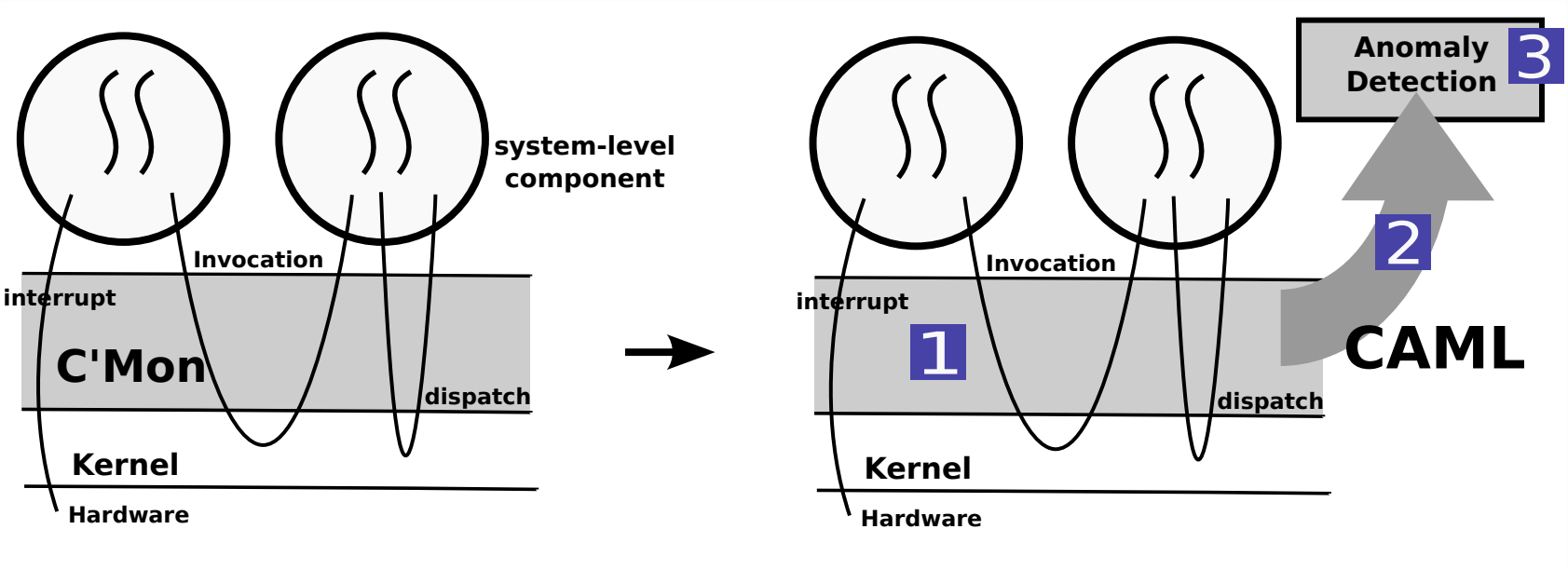
CAML– System-Level Anomaly Detection



Idea →

1. logging events (same as in C'Mon)
2. pre-processing logged events

CAML– System-Level Anomaly Detection

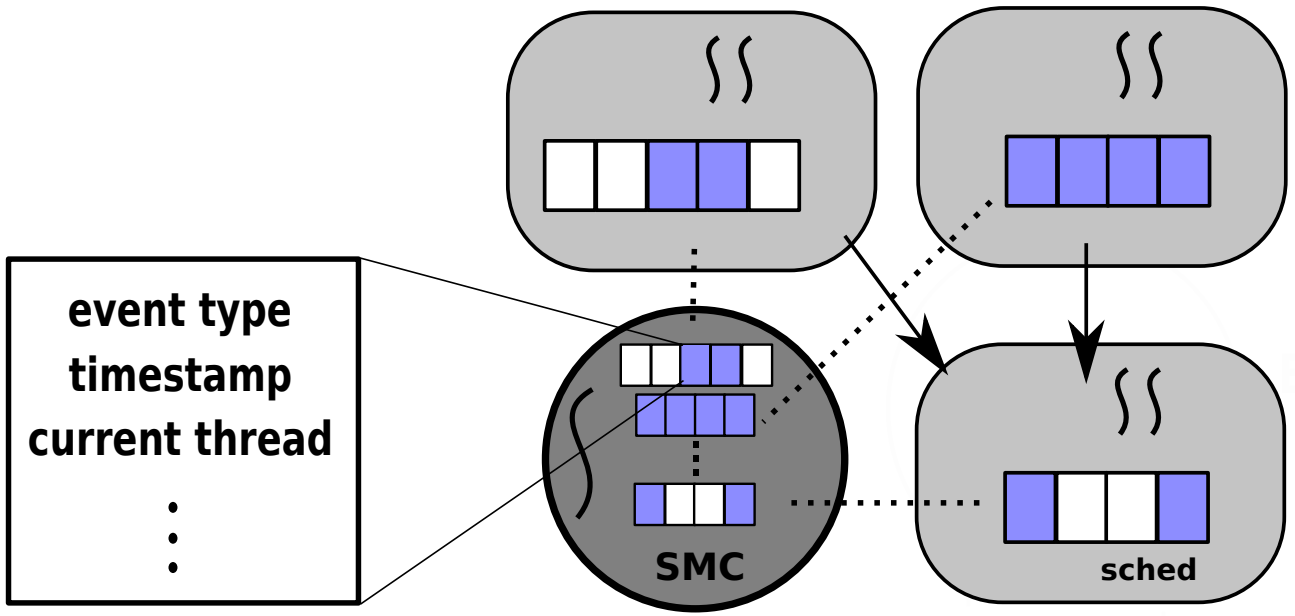


Idea $\rightarrow$

1. logging events (same as in C'Mon)
2. pre-processing logged events
3. detecting anomalies using machine learning techniques

SMC – System Monitor Component

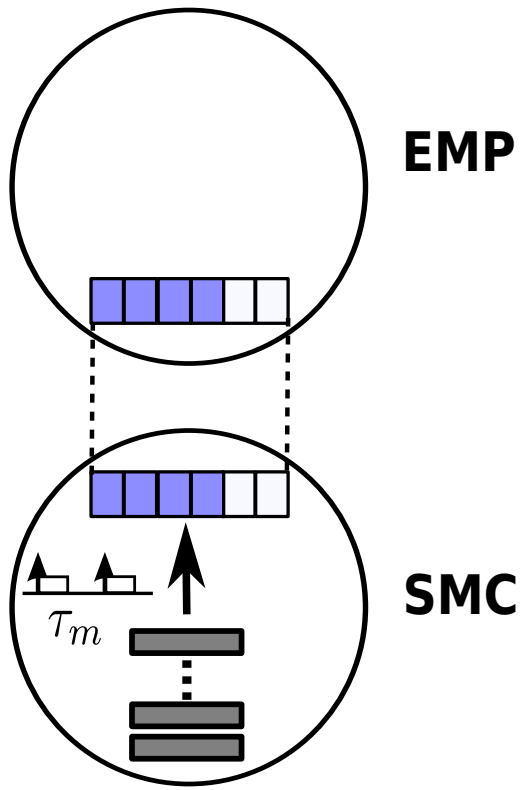
- SMC component
  - log events



- *wait-free* multi-producer single consumer buffer (same as in C'Mon)

SMC – System Monitor Component

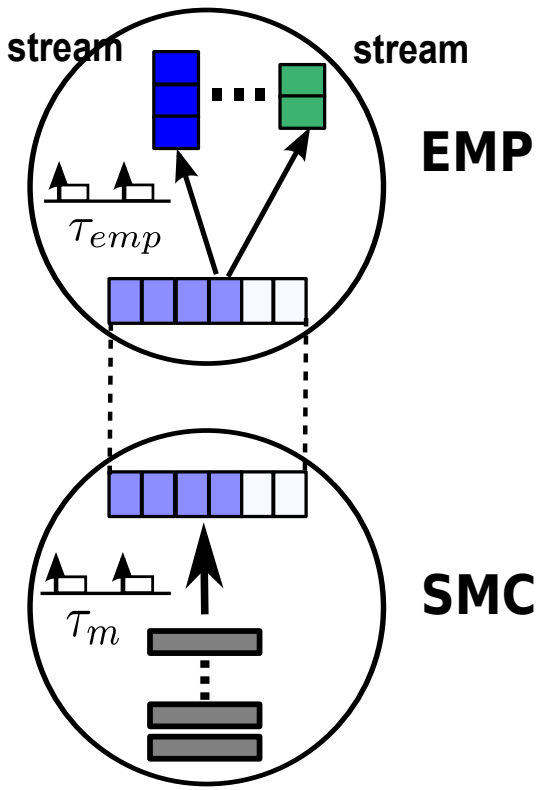
- SMC component
  - log events
  - copy events into the shared buffer between SMC and EMP



- assume shared buffer is large enough

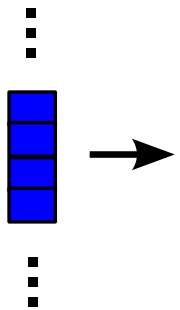
EMP – Event Multiplexing Component

- SMC component
  - log events
  - copy events into the shared buffer between SMC and EMP
- EMP component
  - multiplex logged events into streams



▶ pre-processing execution time

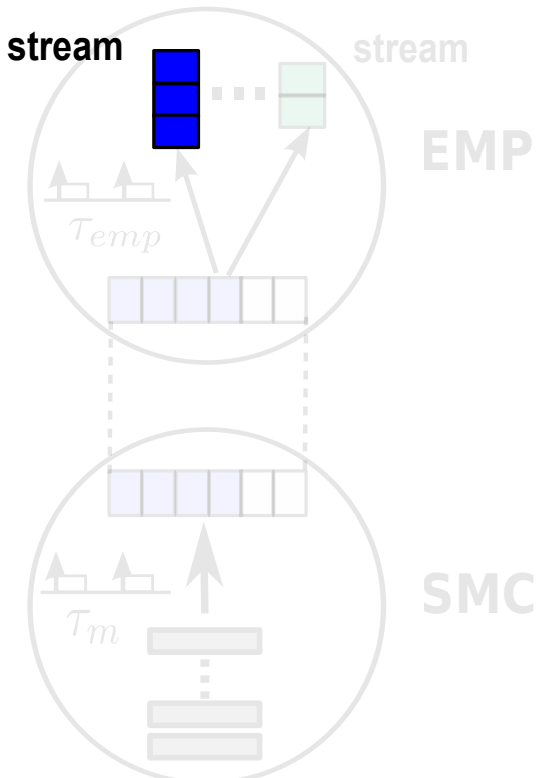
Stream



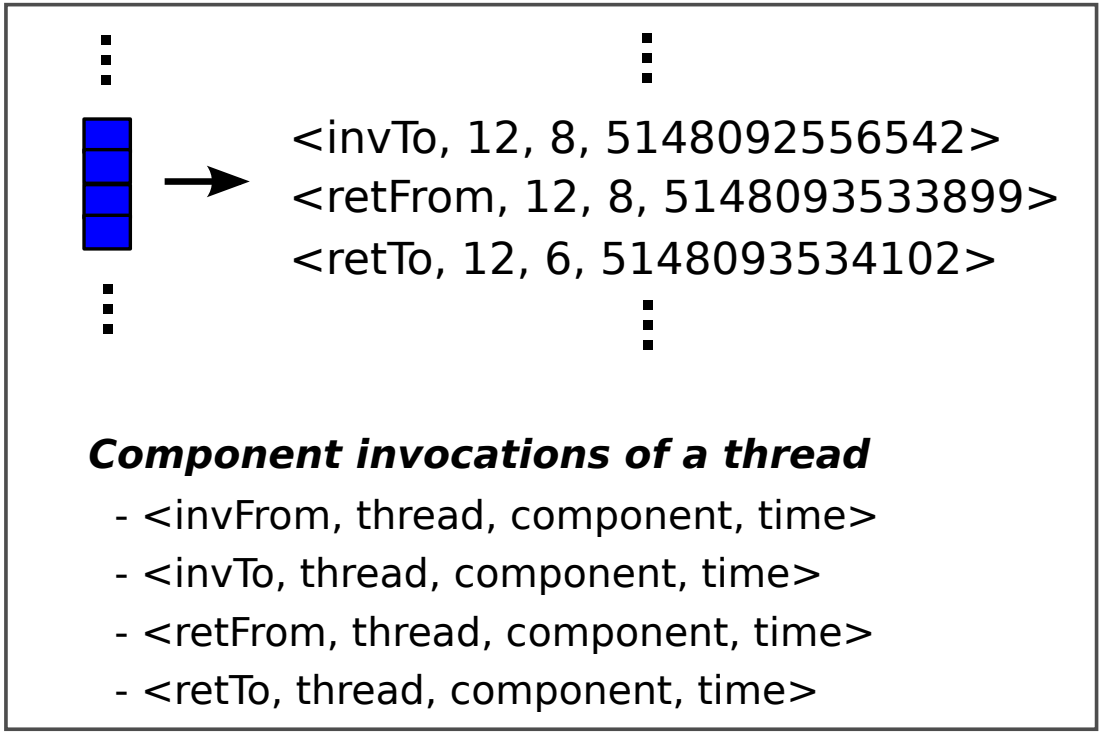
<invTo, 12, 8, 5148092556542>
<retFrom, 12, 8, 5148093533899>
<retTo, 12, 6, 5148093534102>

Component invocations of a thread

- <invFrom, thread, component, time>
- <invTo, thread, component, time>
- <retFrom, thread, component, time>
- <retTo, thread, component, time>

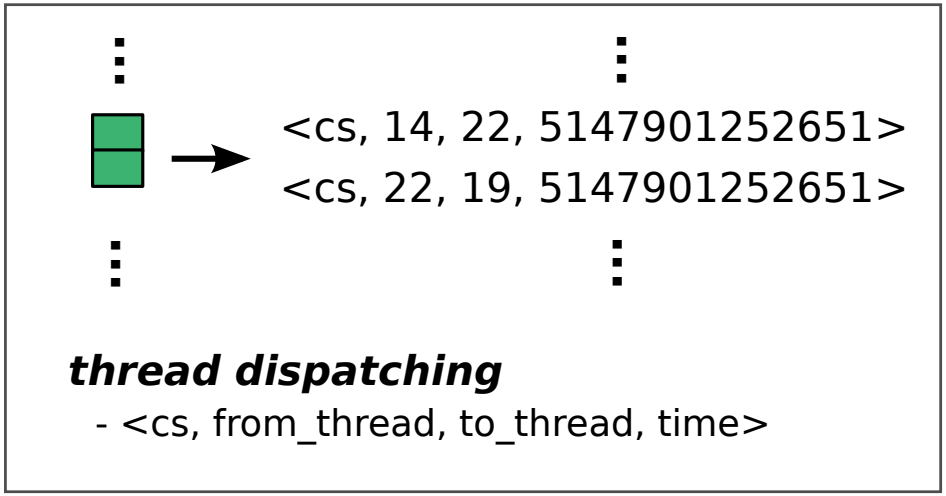


Stream



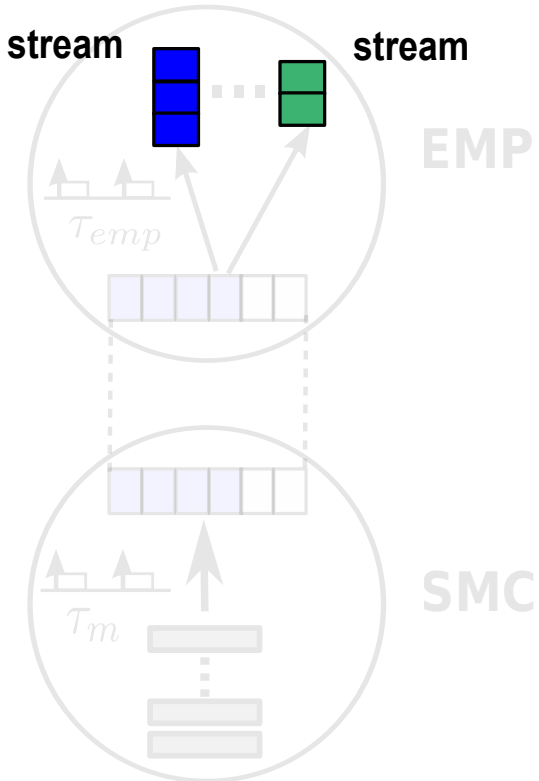
Component invocations of a thread

- <invFrom, thread, component, time>
- <invTo, thread, component, time>
- <retFrom, thread, component, time>
- <retTo, thread, component, time>



thread dispatching

- <cs, from_thread, to_thread, time>



Stream

Per-thread stream

-- sequence of component invocation/return of a thread

-- number of times context switch to/from a thread

⋮

Per-component stream

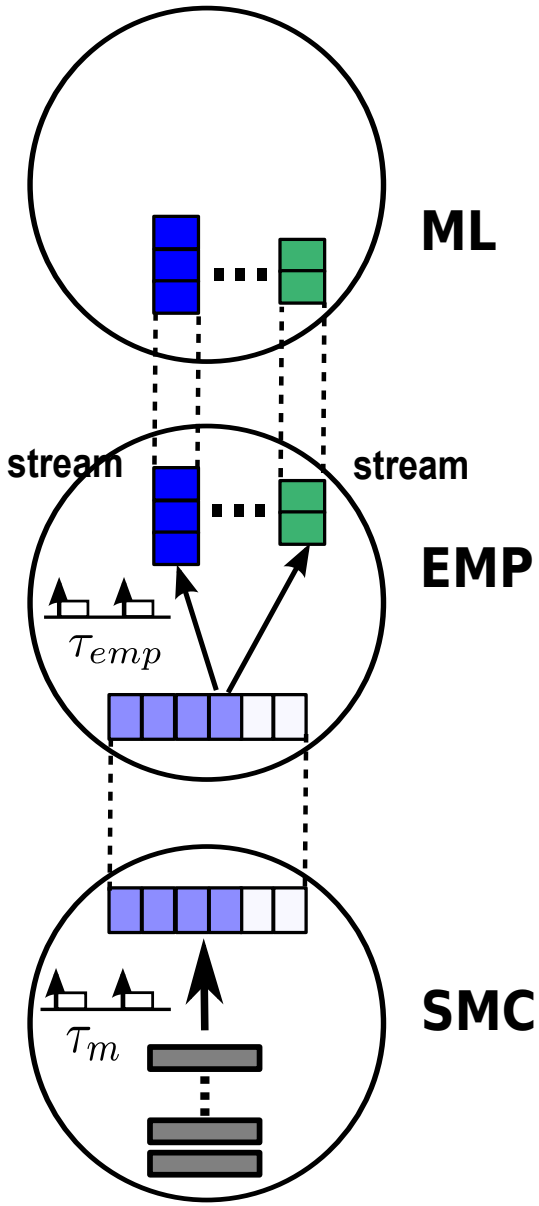
-- invocations/returns of a component by any thread

-- number of times any thread is interrupted in a component

⋮

EMP – Event Multiplexing Component

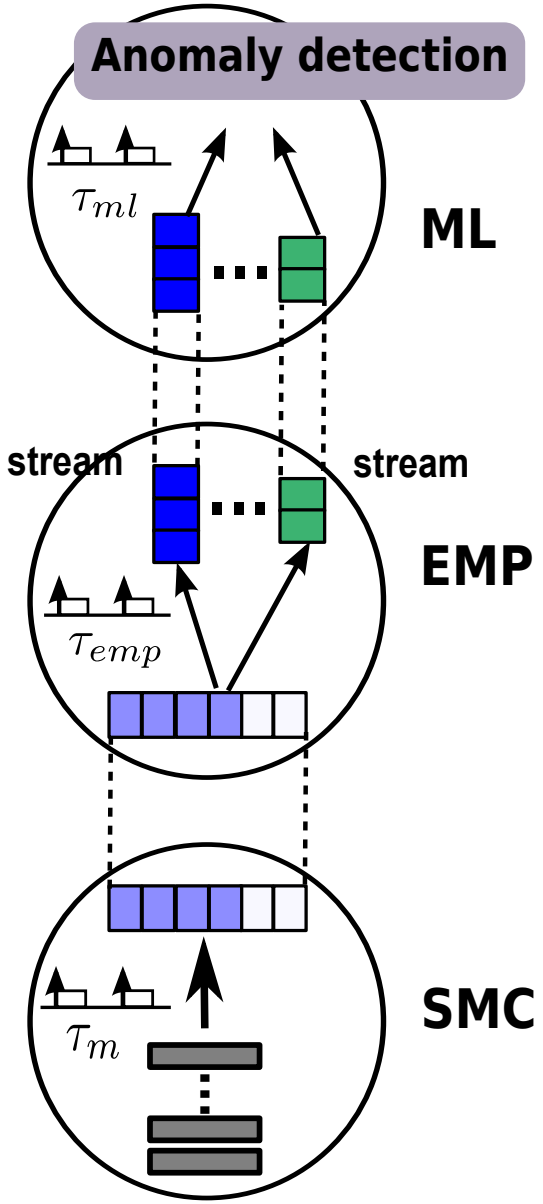
- SMC component
  - log events
  - copy events into the shared buffer between SMC and EMP
- EMP component
  - multiplex logged events into streams
  - copy streams into the shared buffers between EMP and ML



- stream → another layer of abstraction about the system execution

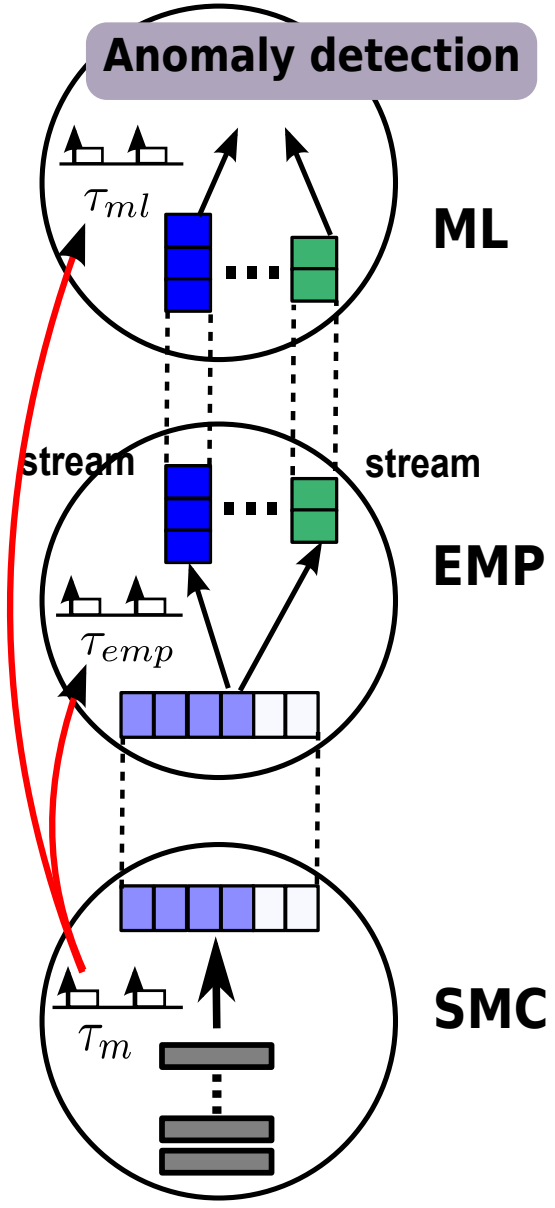
ML – Anomaly Detection Component

- SMC component
  - log events
  - copy events into the shared buffer between SMC and EMP
- EMP component
  - multiplex logged events into streams
  - copy streams into the shared buffers between EMP and ML
- ML component
  - run machine learning algorithms to detect the anomaly



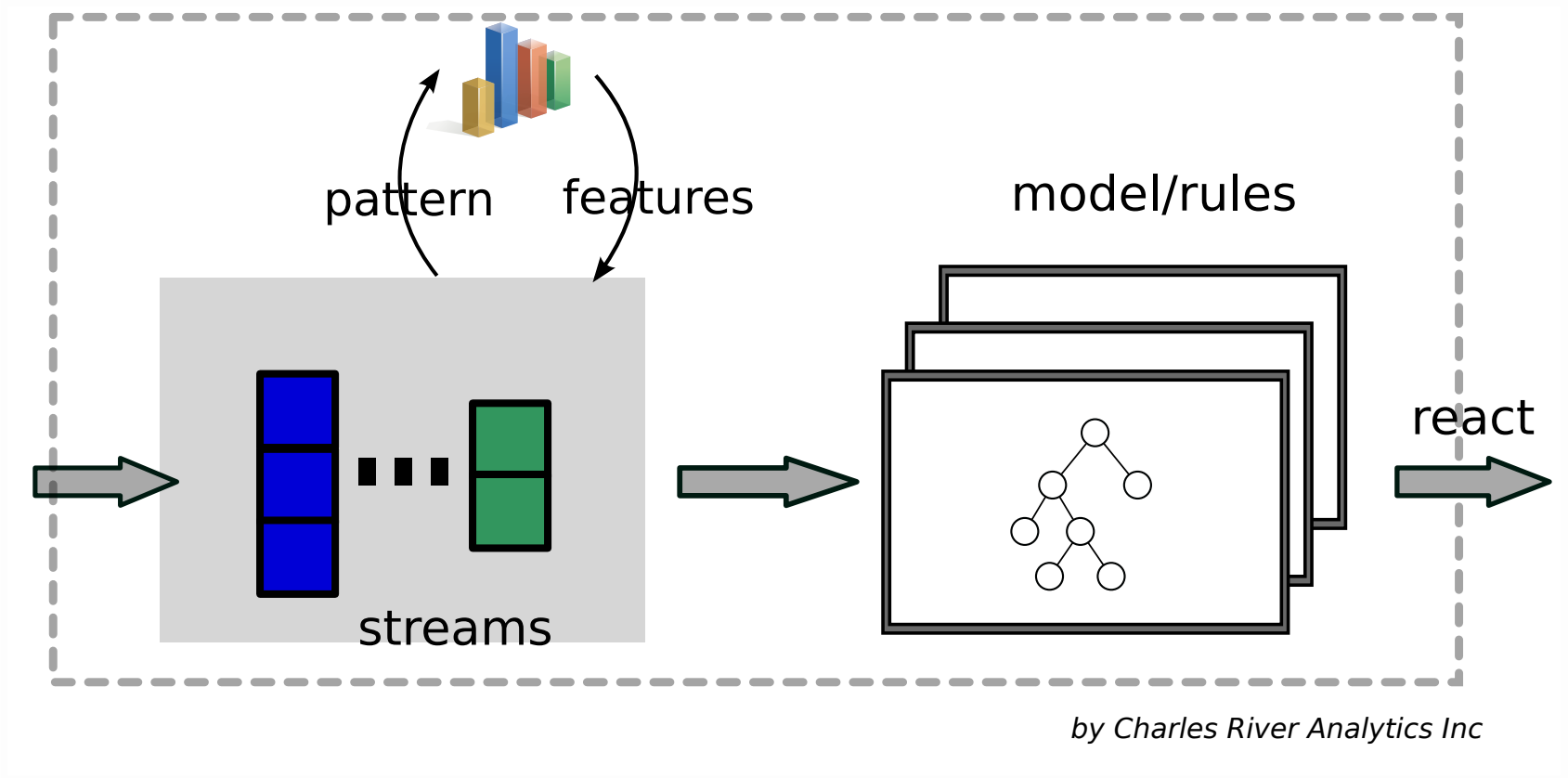
ML – Anomaly Detection Component

- SMC component
  - log events
  - copy events into the shared buffer between SMC and EMP
- EMP component
  - multiplex logged events into streams
  - copy streams into the shared buffers between EMP and ML
- ML component
  - run machine learning algorithms to detect the anomaly



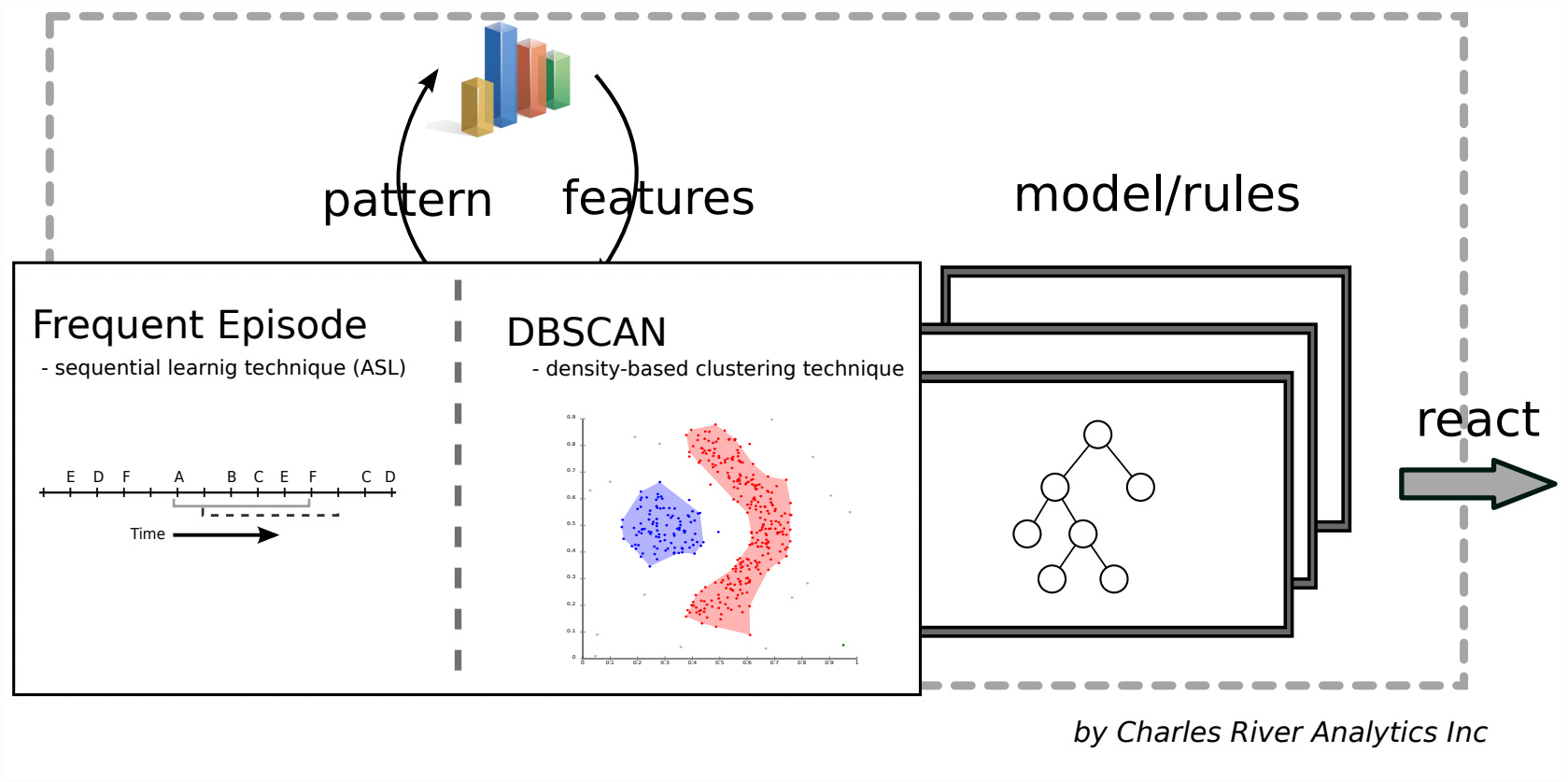
- needs to ensure tasks τ_{emp} and τ_{ml} are activated at right time

Machine Learning based Anomaly Detection



1. build rules
2. detect the violation of the rules (e.g., outlier)
3. determine the likelihood of the fault (*in progress*)

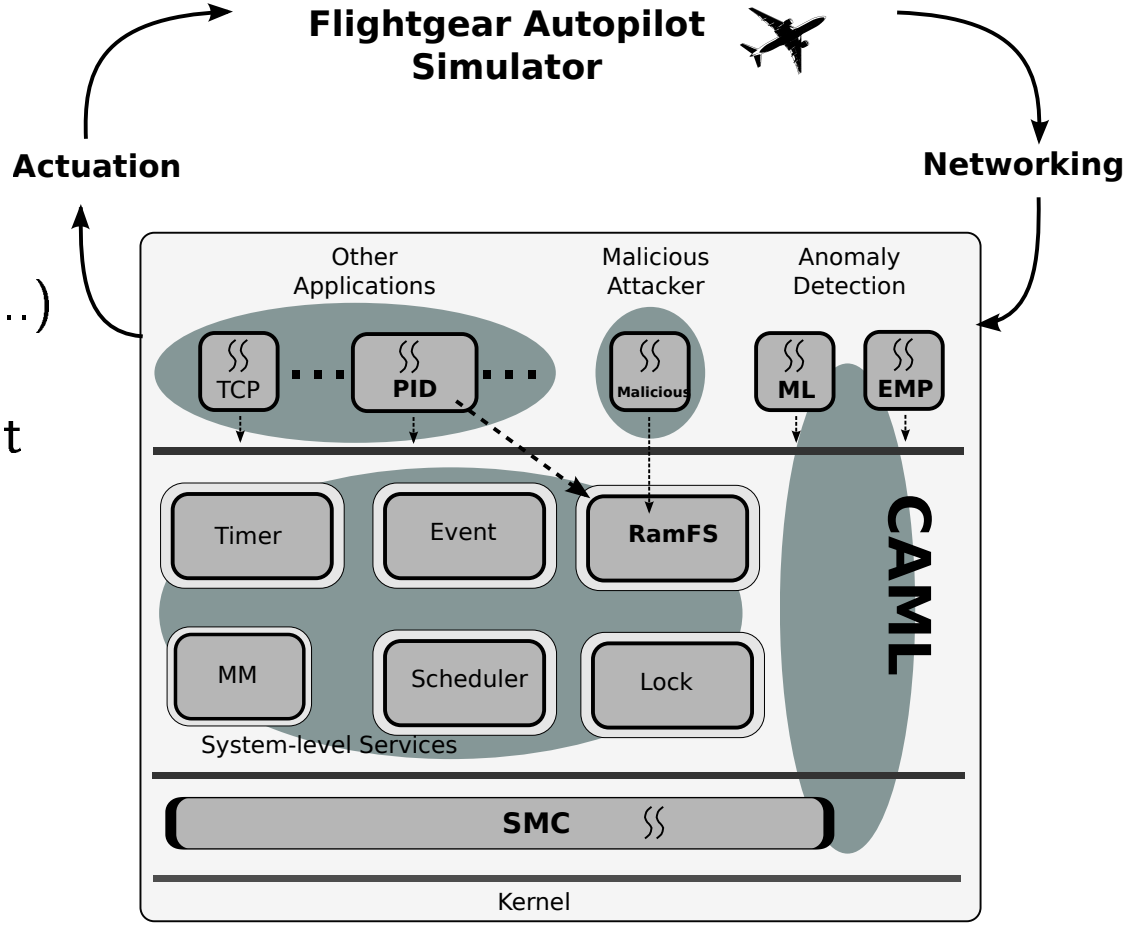
Machine Learning based Anomaly Detection



1. build rules
2. detect the violation of the rules (e.g., outlier)
3. determine the likelihood of the fault (*in progress*)

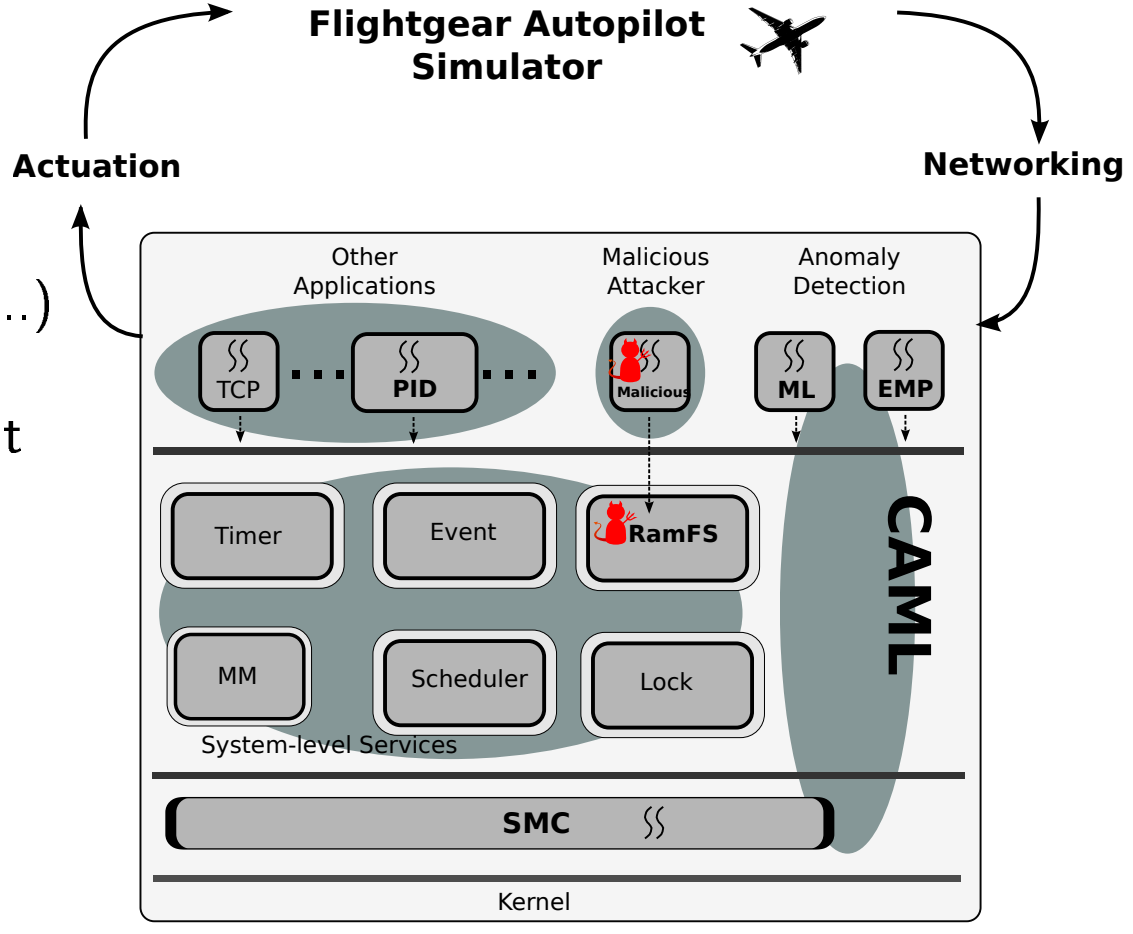
Evaluation – Flightgear Autopilot Simulator

- Normally
  1. sensor input (position, velocities, acceleration...)
  2. PID computation
  3. corrective action output



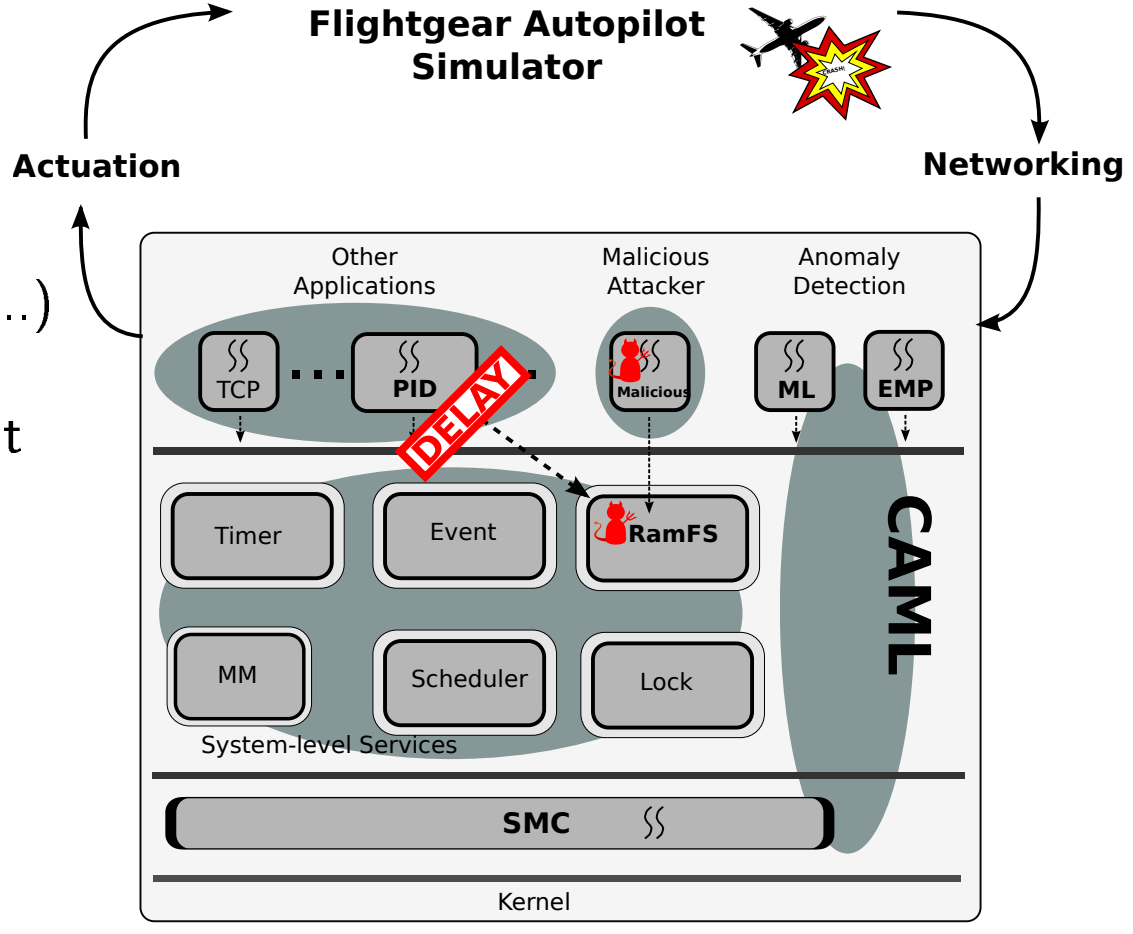
Evaluation – Flightgear Autopilot Simulator

- Normally
  1. sensor input (position, velocities, acceleration...)
  2. PID computation
  3. corrective action output
- Malicious user or software bugs
  - hog the FS

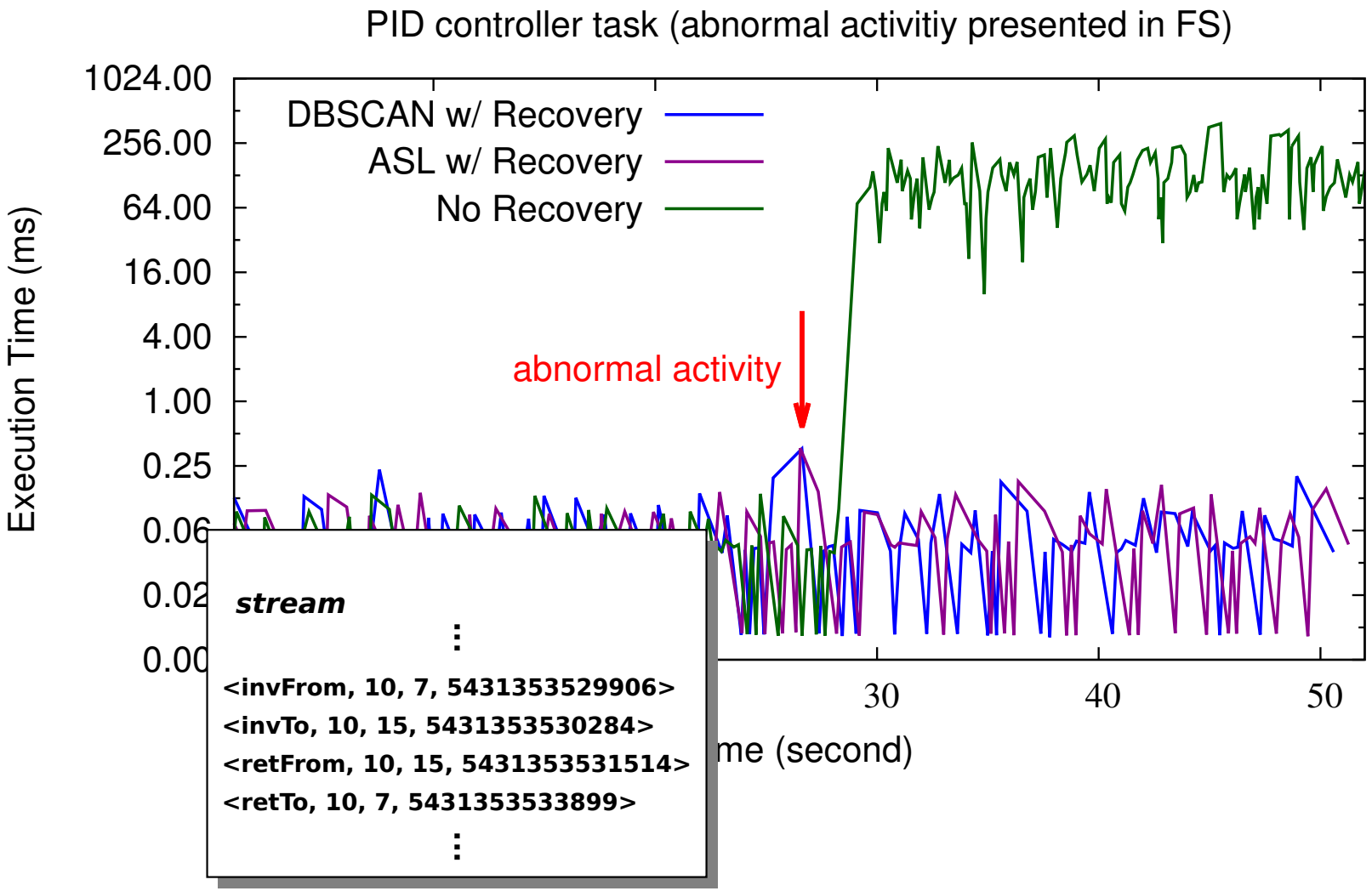


Evaluation – Flightgear Autopilot Simulator

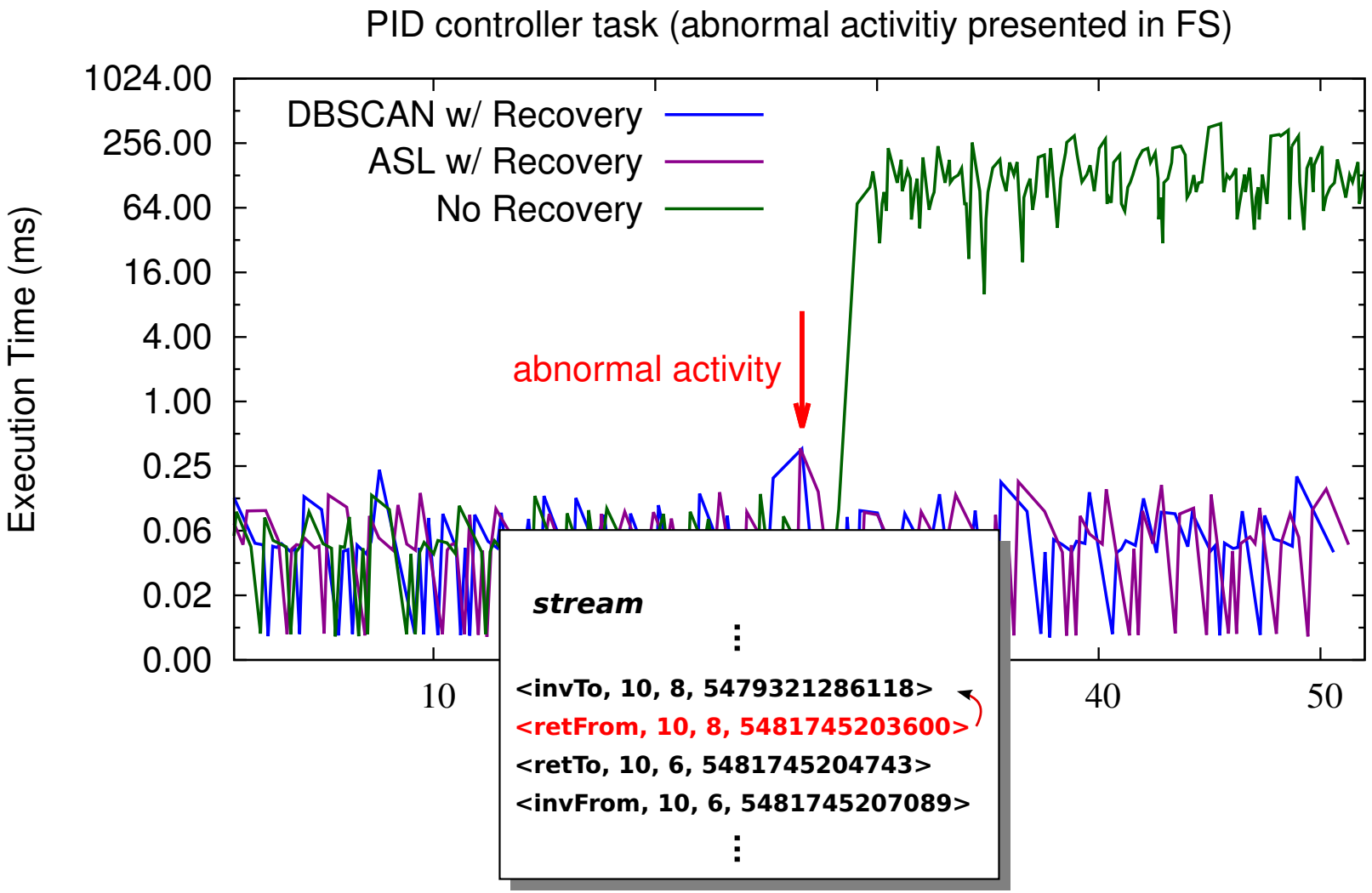
- Normally
  1. sensor input (position, velocities, acceleration...)
  2. PID computation
  3. corrective action output
- Malicious user or software bugs
  - hog the FS
  - delay PID computation



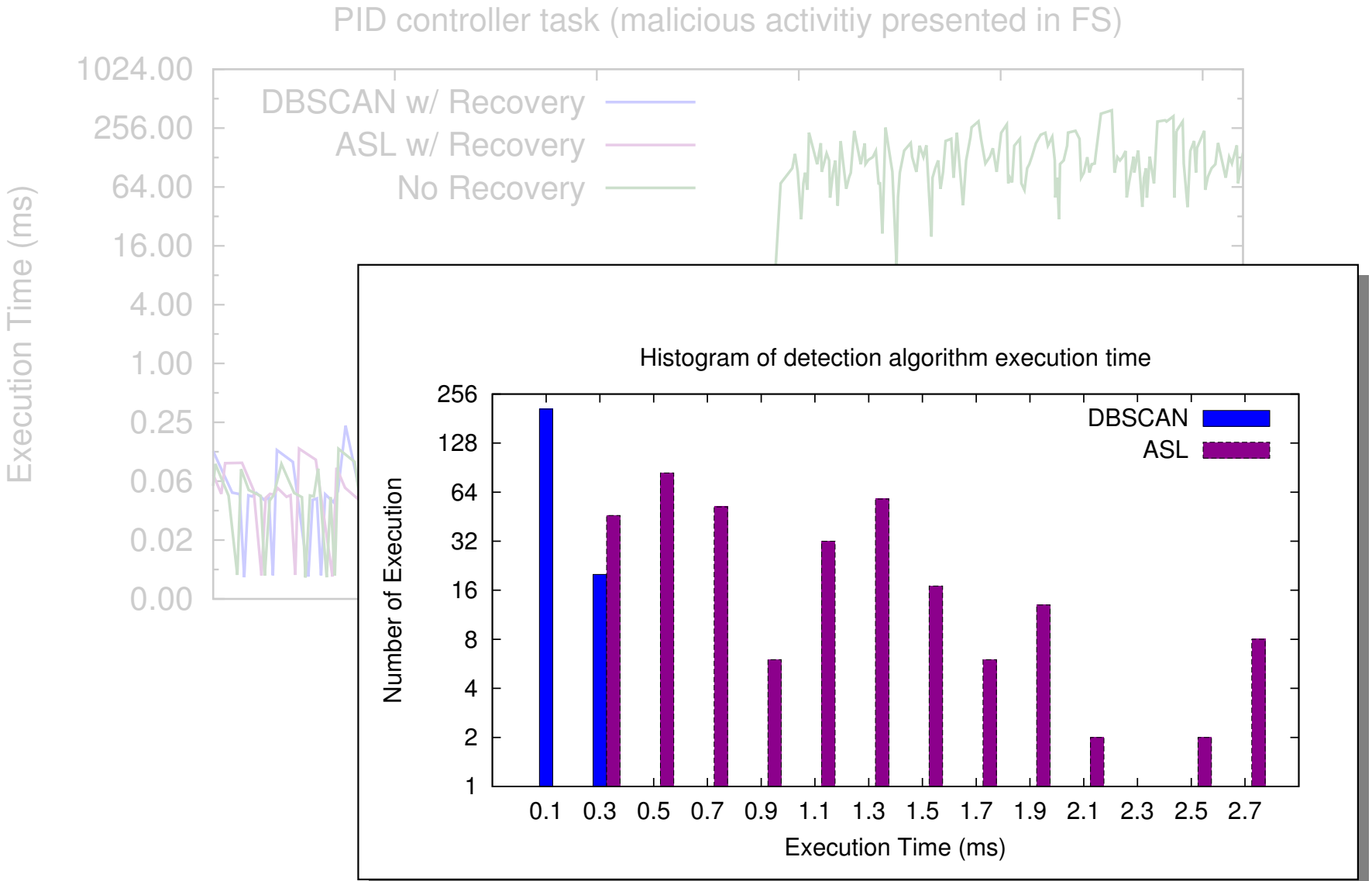
Anomaly Detection Experiment



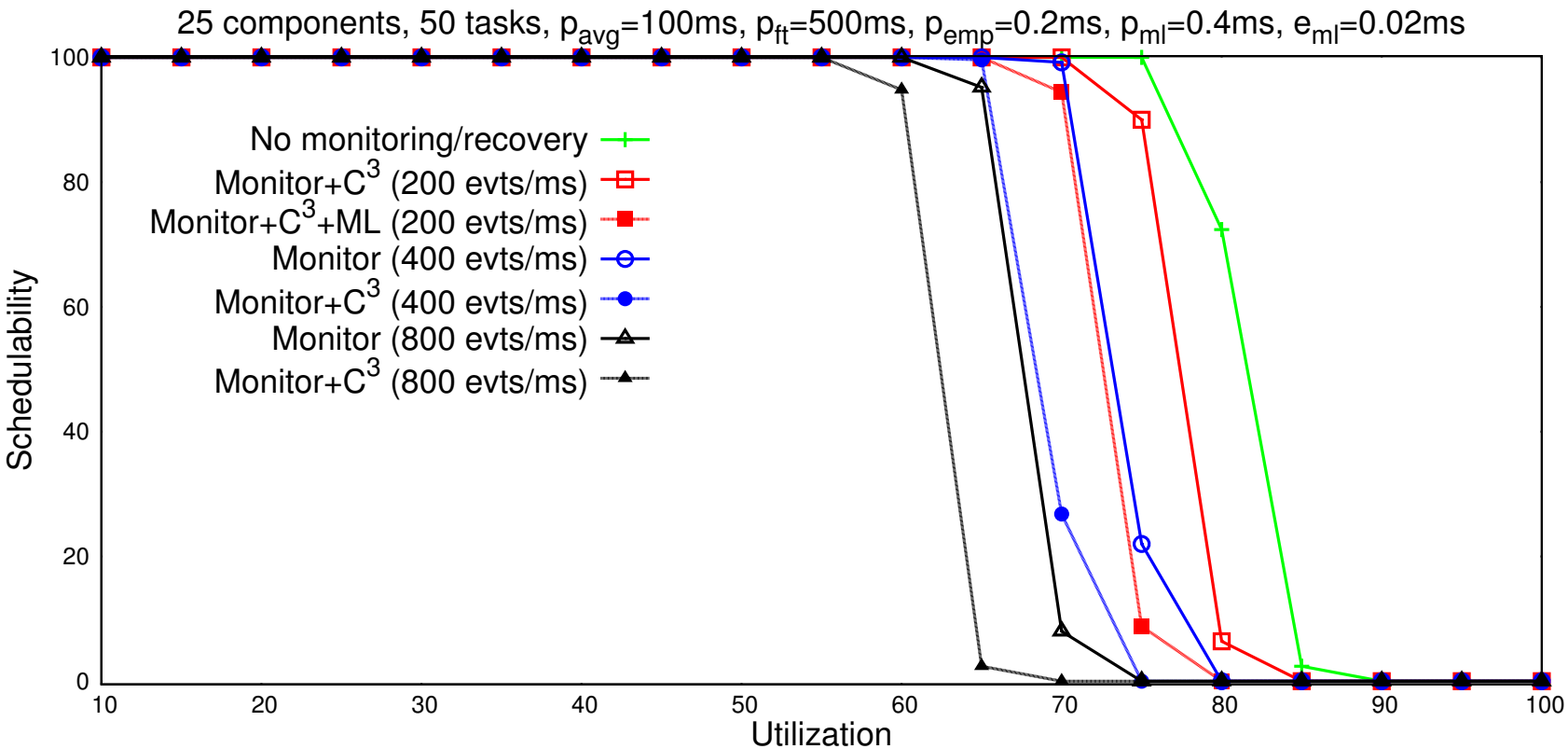
Anomaly Detection Experiment



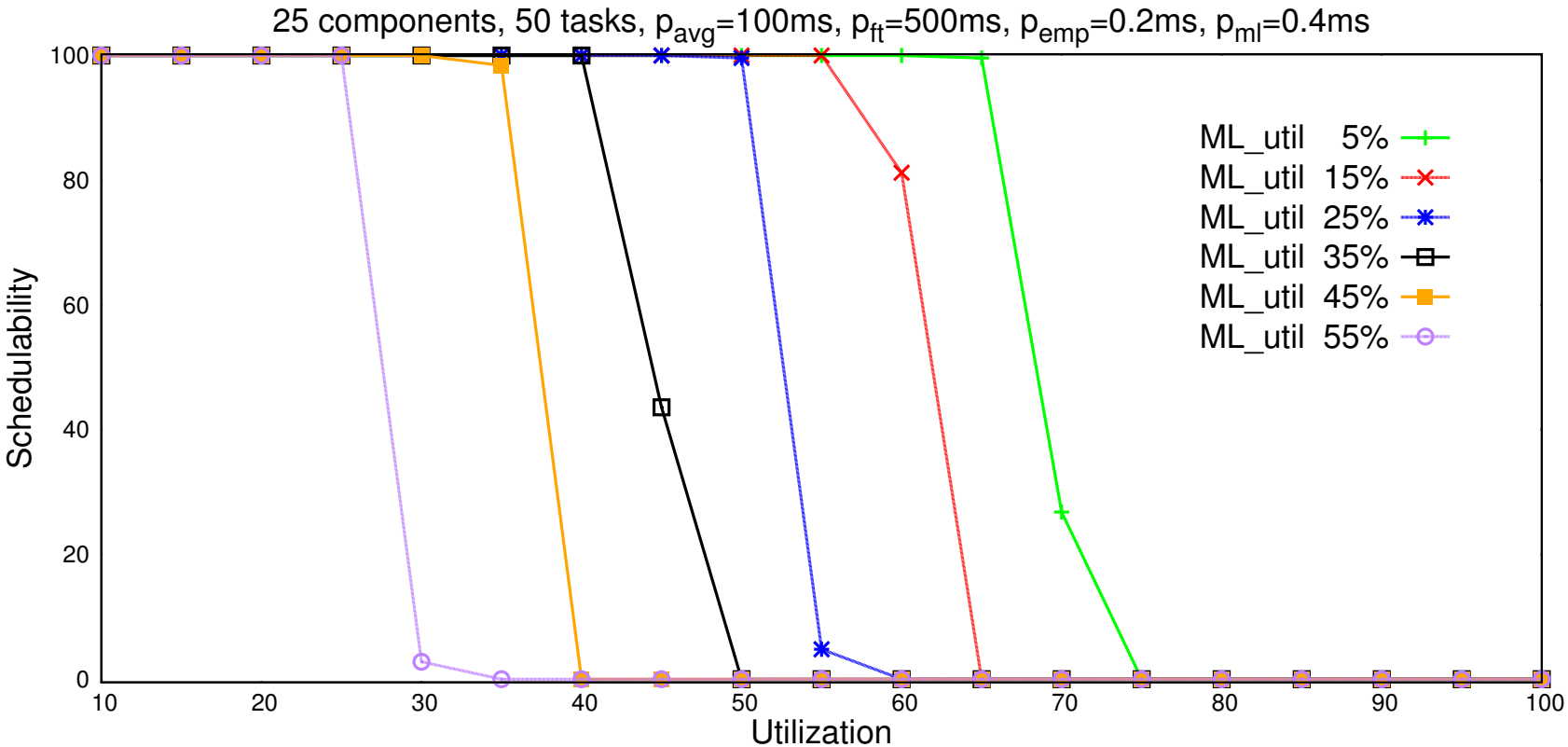
Anomaly Detection Experiment



Response Time Analysis (RTA) Result



RTA Result – Machine Learning Task Utilization



Conclusion on CAML

CAML – an ongoing project that

- aims at detecting system-level anomaly predictably
  - *allows the recovery of suspicious component*
- investigates the feasibility using machine learning techniques for anomaly detection
  - *collaborate with Charles River Analytics Inc*

Future Work in CAML

Issues that need to be further addressed include

- determine the likelihood of fault occurrence and localize the suspicious service from the rule violations
- study the impact of detection false positives and false negatives on the system schedulability

Publications

- J. Song, G. Bloom and G. Parmer, "SuperGlue: IDL-Based, System-Level Fault Tolerance for Embedded Systems", the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2016
- J. Song, and G. Parmer, "C'Mon: a Predictable Monitoring Infrastructure for System-Level Latent Fault Detection and Recovery", in the 20th RTAS, 2015
- J. Song, J. Wittrock, and G. Parmer, "Predictable, efficient system-level fault tolerance in C^3", in the 34th RTSS, 2013
- J. Song, Q. Wang, and G. Parmer, "The state of composite", in the Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), 2013
- J. Song and G. Parmer, "Toward predictable, efficient, system-level tolerance of transient faults", in the 5th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES), 2013
- E. Armbrust, J. Song, G. Bloom, and G. Parmer, "On Spatial Isolation for Mixed Criticality, Embedded Systems", in the Workshop on Mixed Criticality, 2014
- Q. Wang, J. Song, G. Parmer, G. Venkataramani, and A. Sweeney, "Increasing memory utilization with transient memory scheduling", in the 33rd RTSS, 2012
- Q. Wang, J. Song, and G. Parmer, "Stack management for hard real-time computation in a component-based OS", in the 32nd RTSS, 2011
- G. Parmer and J. Song, "Customizable and predictable synchronization in a component-based OS", in the International Conference on Embedded Systems and Applications (ESA), 2010

Thanks

? || /* */

composite.seas.gwu.edu

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC