Veriamos Project

SSD I/O via a DSL @ Ring 0

PhaistOS DSL

2019

Dr. Nick Papoulias http://parsenet.info

Veriamos Project

SSD I/O via a DSL @ Ring 0

Motivation ??

Mitigate:

HW / SW Mismatch

A hardware innovation invalidates the assumptions of the software stack (kernel or other) resulting in sub-optimal abstractions

Need for new abstractions and/or domain specific fine-tuning

Veriamos Project

SSD I/O via a DSL @ Ring 0

Motivation ??

HW / SW Mismatch

Random I/O is expensive

Random I/O is cheap

Example:

Can handle thousands IOPS

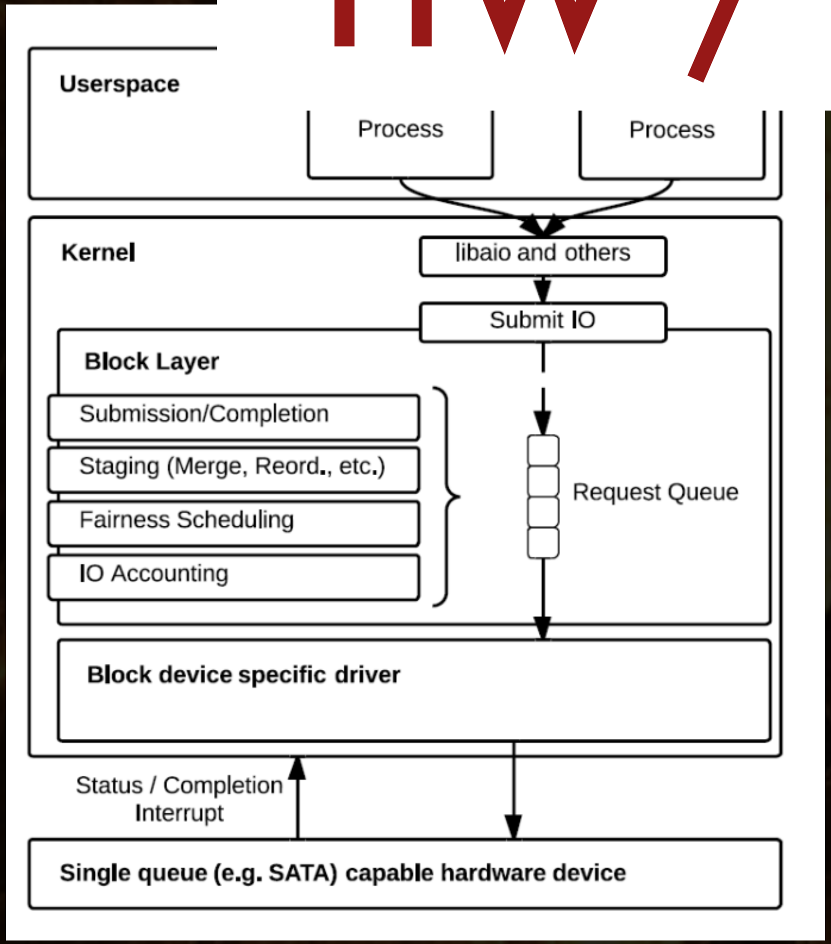
Can deliver tens of millions IOPS

Veriamos Project

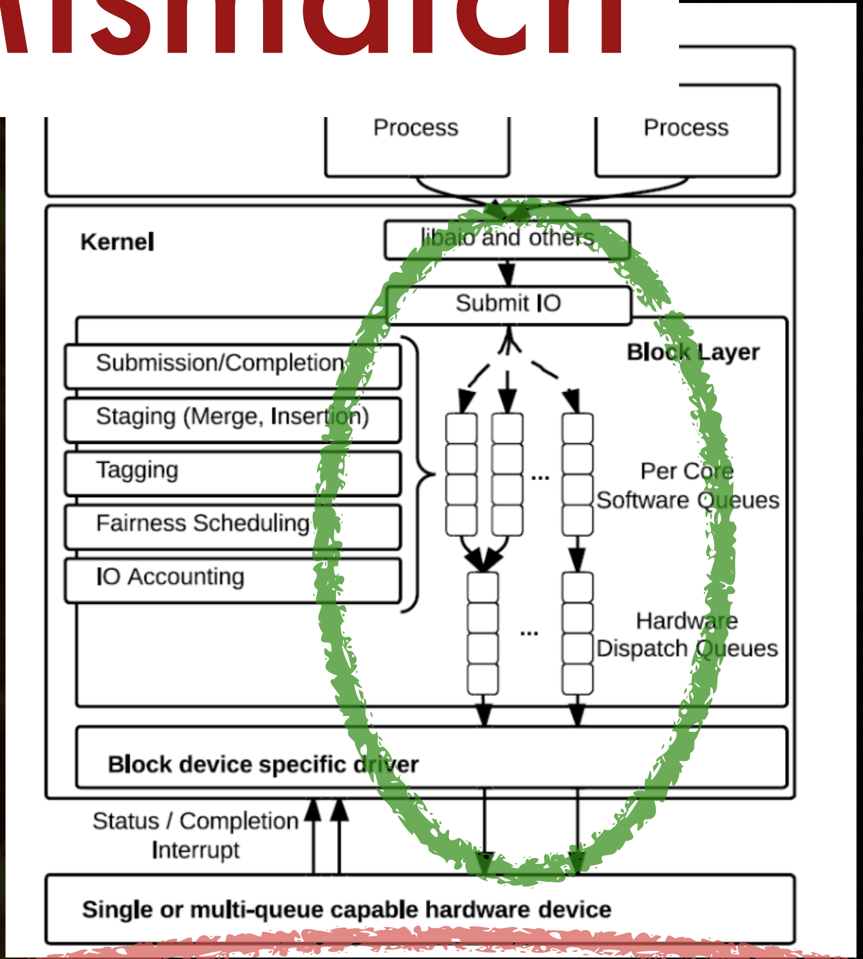
SSD I/O via a DSL @ Ring 0

Motivation ??

HW / SW Mismatch


Userspace
Process
Process
Kernel
libaio and others
Submit IO
Block Layer
Submission/Completion
Staging (Merge, Reord., etc.)
Fairness Scheduling
IO Accounting
Request Queue
Block device specific driver
Status / Completion Interrupt
Single queue (e.g. SATA) capable hardware device

2014



(CFQ, BFQ, Kyber)*, Deadline, Noop ..

MQ-Deadline, Noop ..

~~* mq-equiv still used for non-SSD~~

Solved IOPS bottleneck, but MQ has less scheduling options main trade-off: throughput / latency

Veriamos Project

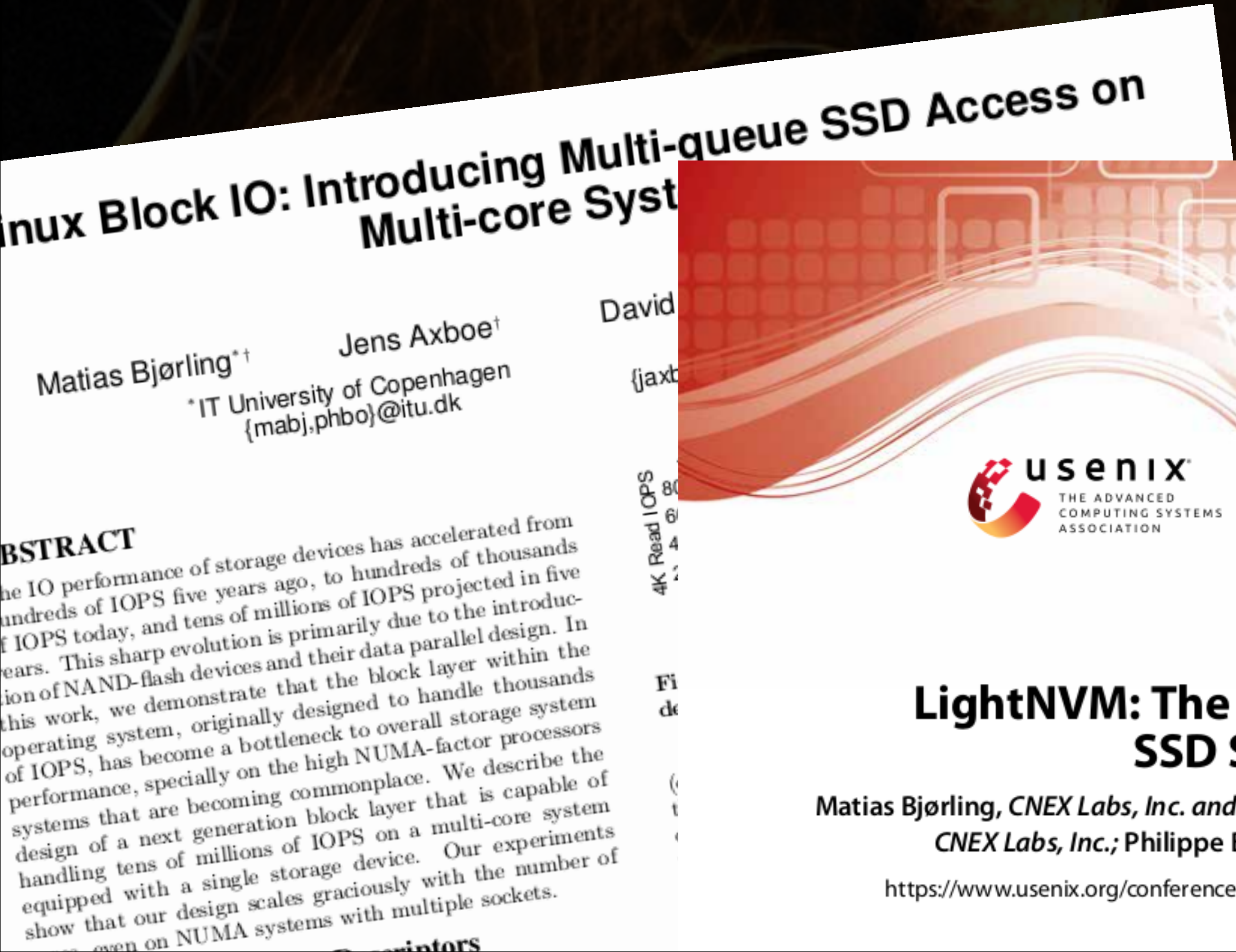
SSD I/O via a DSL @ Ring 0

Rel. Work

HW / SW Mismatch

See the CLyDE Project: Univ. Copenhaghen + Inria

https://clydeitu.wordpress.com/

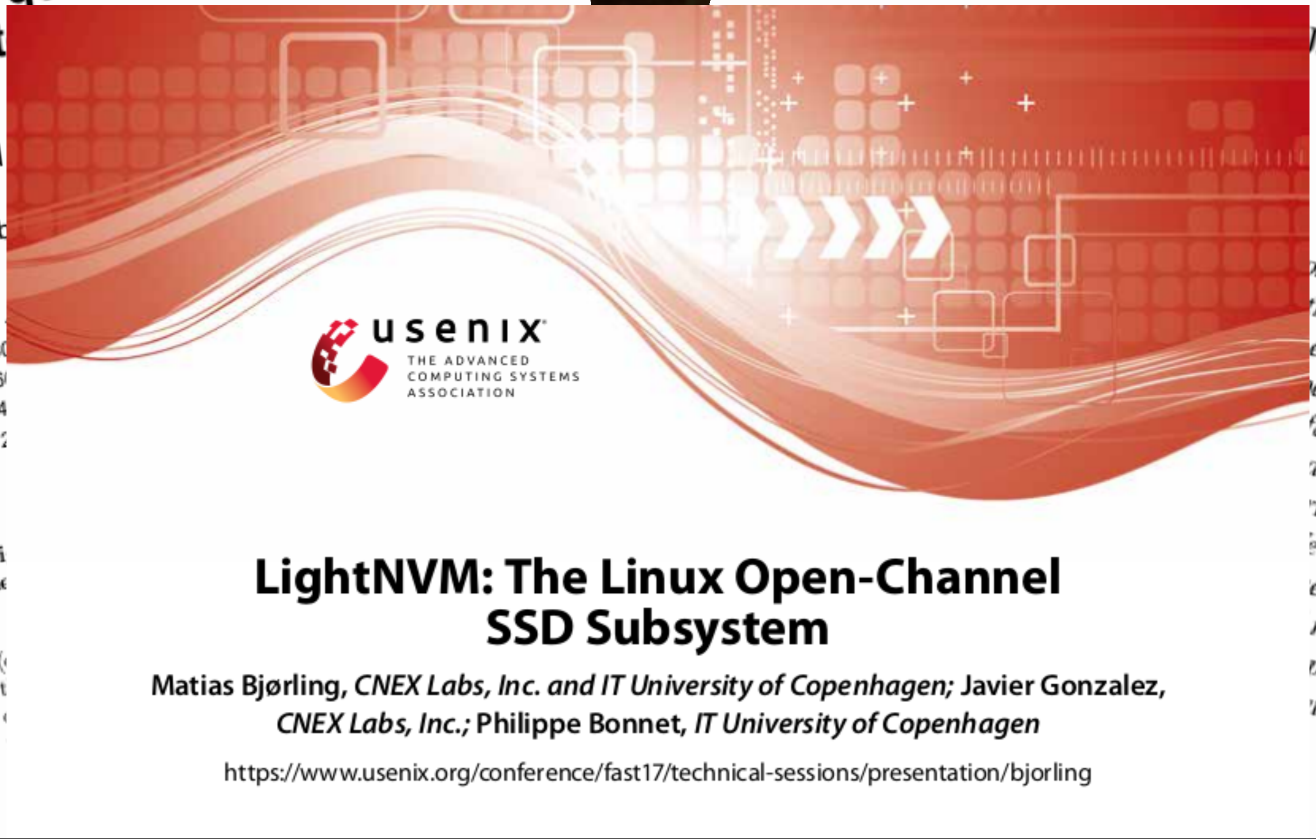


inux Block IO: Introducing Multi-queue SSD Access on Multi-core Syst

Matias Bjørling*† Jens Axboe† David
*IT University of Copenhagen
{mabj,phbo}@itu.dk
{jaxb

BSTRACT

he IO performance of storage devices has accelerated from undreds of IOPS five years ago, to hundreds of thousands f IOPS today, and tens of millions of IOPS projected in five ears. This sharp evolution is primarily due to the introduc- ion of NAND-flash devices and their data parallel design. In this work, we demonstrate that the block layer within the operating system, originally designed to handle thousands of IOPS, has become a bottleneck to overall storage system performance, specially on the high NUMA-factor processors systems that are becoming commonplace. We describe the design of a next generation block layer that is capable of handling tens of millions of IOPS on a multi-core system equipped with a single storage device. Our experiments show that our design scales graciously with the number of ... even on NUMA systems with multiple sockets.

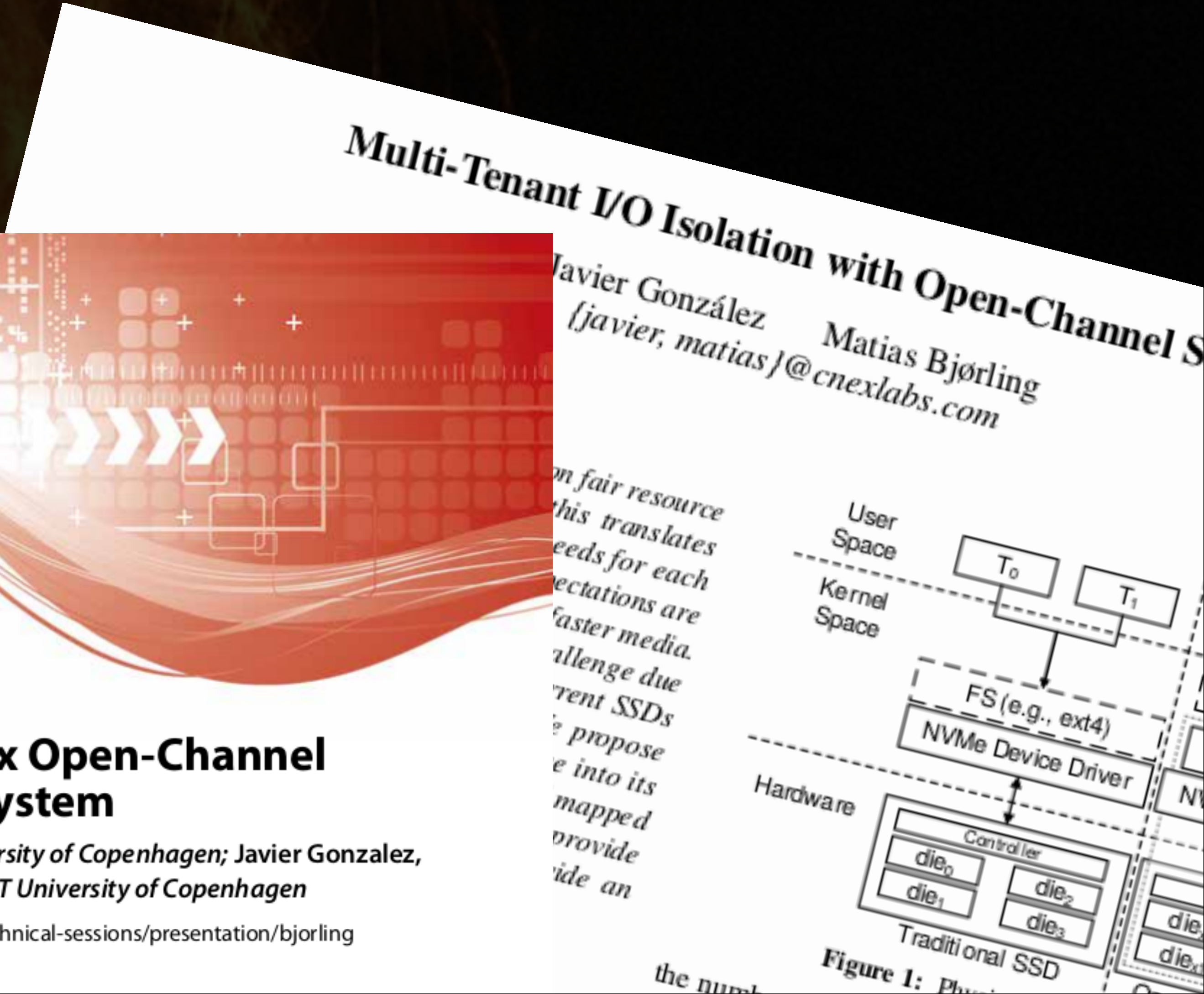


usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

LightNVM: The Linux Open-Channel SSD Subsystem

Matias Bjørling, *CNEX Labs, Inc. and IT University of Copenhagen;* Javier Gonzalez, *CNEX Labs, Inc.;* Philippe Bonnet, *IT University of Copenhagen*

https://www.usenix.org/conference/fast17/technical-sessions/presentation/bjorling



Multi-Tenant I/O Isolation with Open-Channel S

Javier González Matias Bjørling
{javier, matias}@cnexlabs.com

n fair resource this translates eeds for each ectations are aster media. allenge due rent SSDs e propose e into its mapped provide ide an

the num

Figure 1: Ph

Veriamos Project

SSD I/O via a DSL @ Ring 0

Problem ??

HW / SW Mismatch

Institution
Public/Private

- Needs to optimize I/O
- Does not have kernel devs on staff
- Additional risk (faults, security) of ad-hoc solution not acceptable

Accept unpredictable trade-off of MQ: throughput / latency, write your own scheduler or LightNVM solution

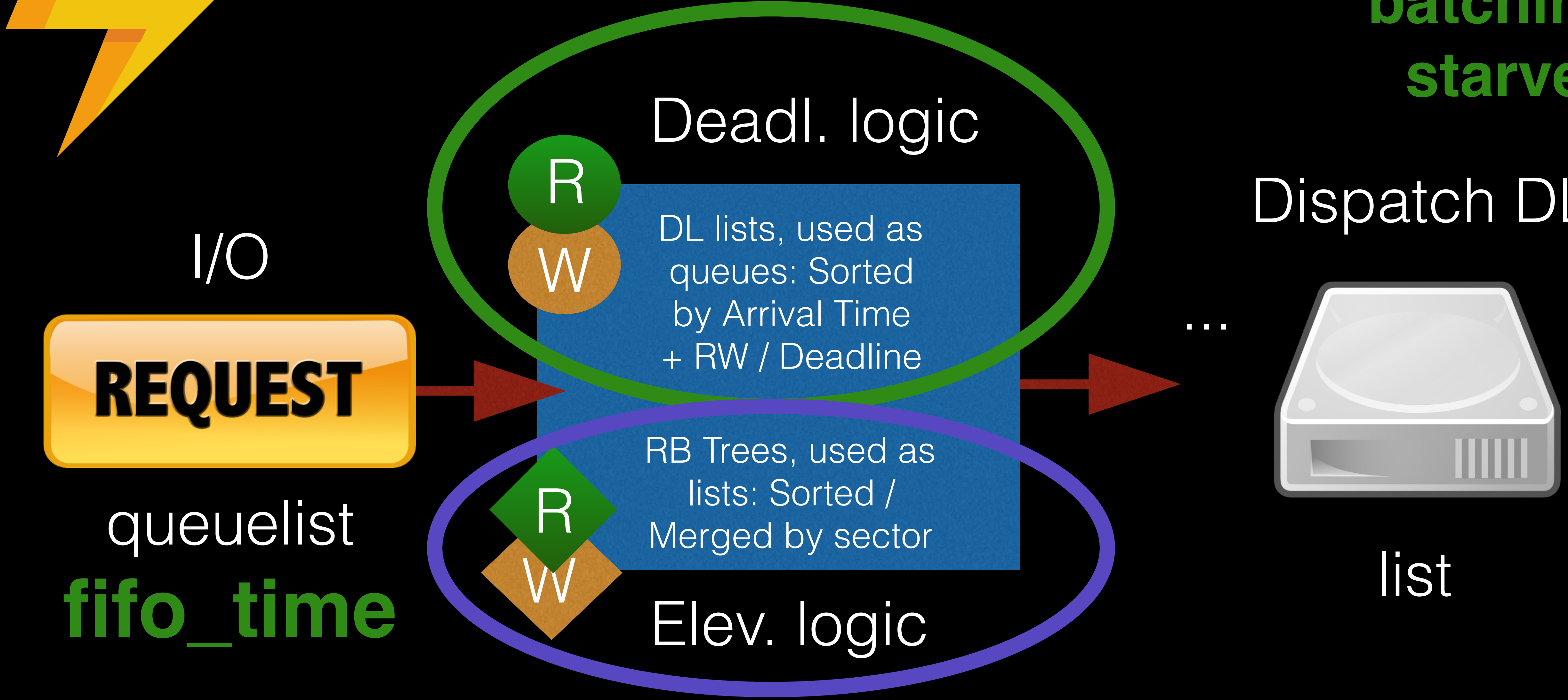
Brief Reminder: Deadline

Events

- INIT
- INSERT
- HAS WORK

- MERGE
- DISPATCH
- REMOVE
- EXIT

fifo_expire[2]
fifo_batch
writes_starved
batching
starved

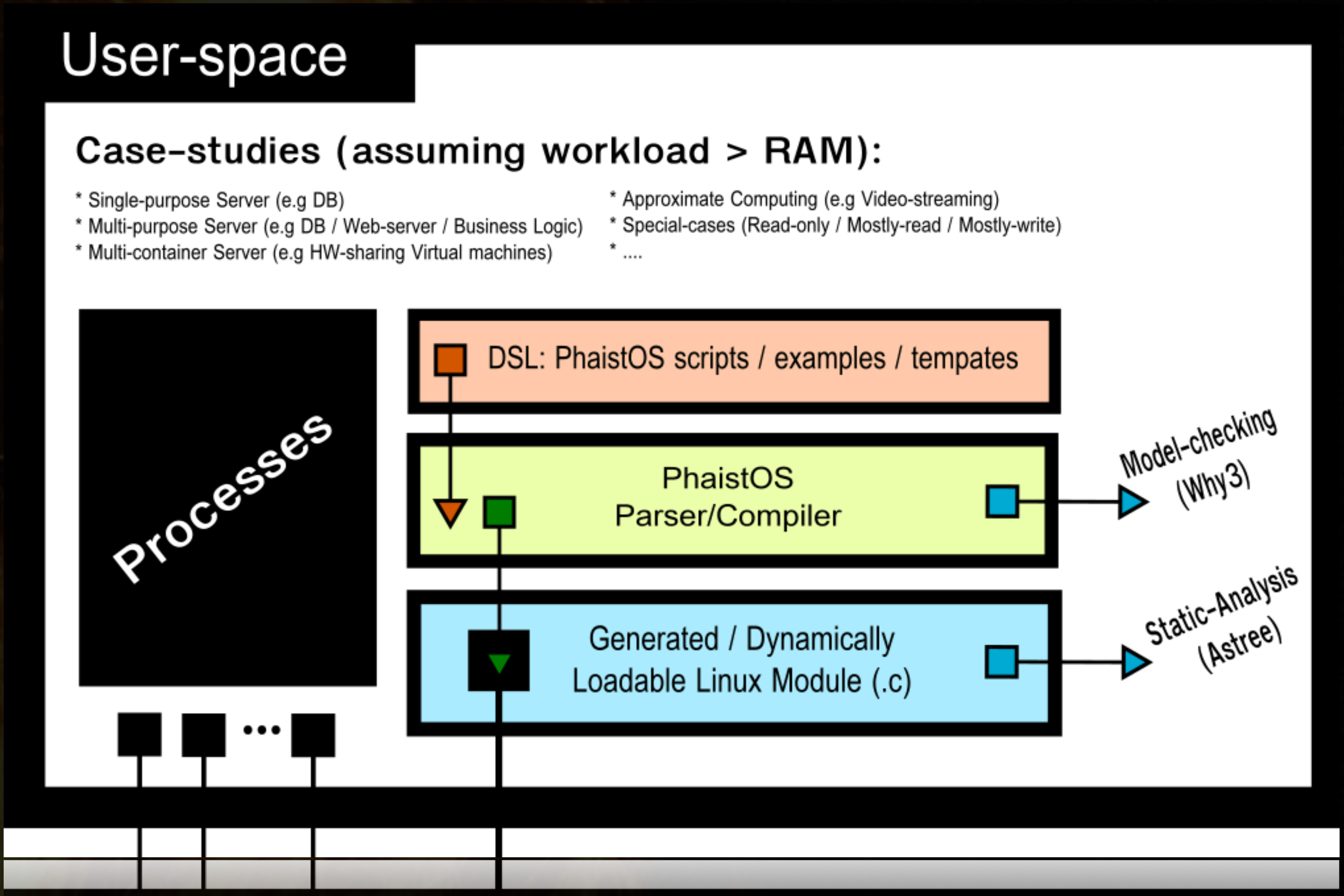

Deadl. logic
R
W
DL lists, used as queues: Sorted by Arrival Time + RW / Deadline
RB Trees, used as lists: Sorted / Merged by sector
Elev. logic
I/O
REQUEST
queuelist
fifo_time
Dispatch DL
...
list

Veriamos Project

SSD I/O via a DSL @ Ring 0

Architecture

PhaistOS DSL

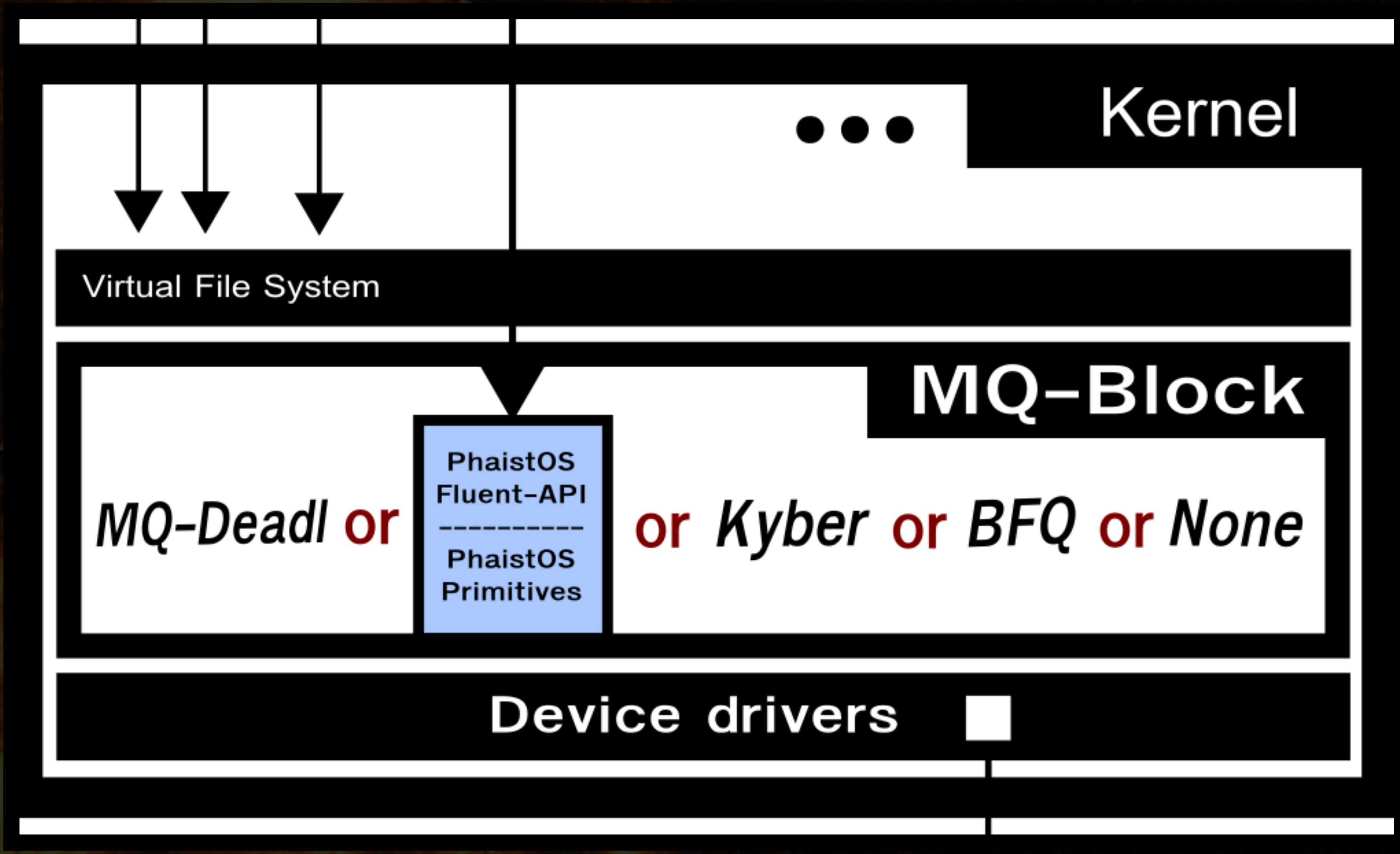

User-space
Case-studies (assuming workload > RAM):
* Single-purpose Server (e.g DB)
* Multi-purpose Server (e.g DB / Web-server / Business Logic)
* Multi-container Server (e.g HW-sharing Virtual machines)
* Approximate Computing (e.g Video-streaming)
* Special-cases (Read-only / Mostly-read / Mostly-write)
* ....
Processes
DSL: PhaistOS scripts / examples / tempates
PhaistOS Parser/Compiler
Generated / Dynamically Loadable Linux Module (.c)
Model-checking (Why3)
Static-Analysis (Astree)
...

Veriamos Project

SSD I/O via a DSL @ Ring 0

Architecture

PhaistOS DSL



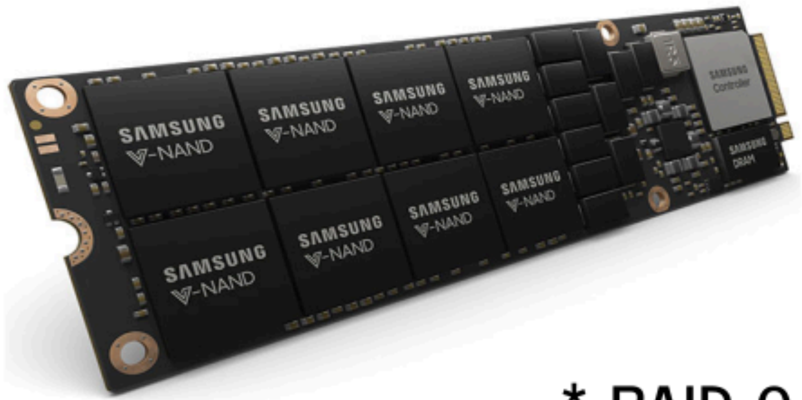
Veriamos Project

SSD I/O via a DSL @ Ring 0

Architecture

PhaistOS DSL

SSD / NVMe Setup



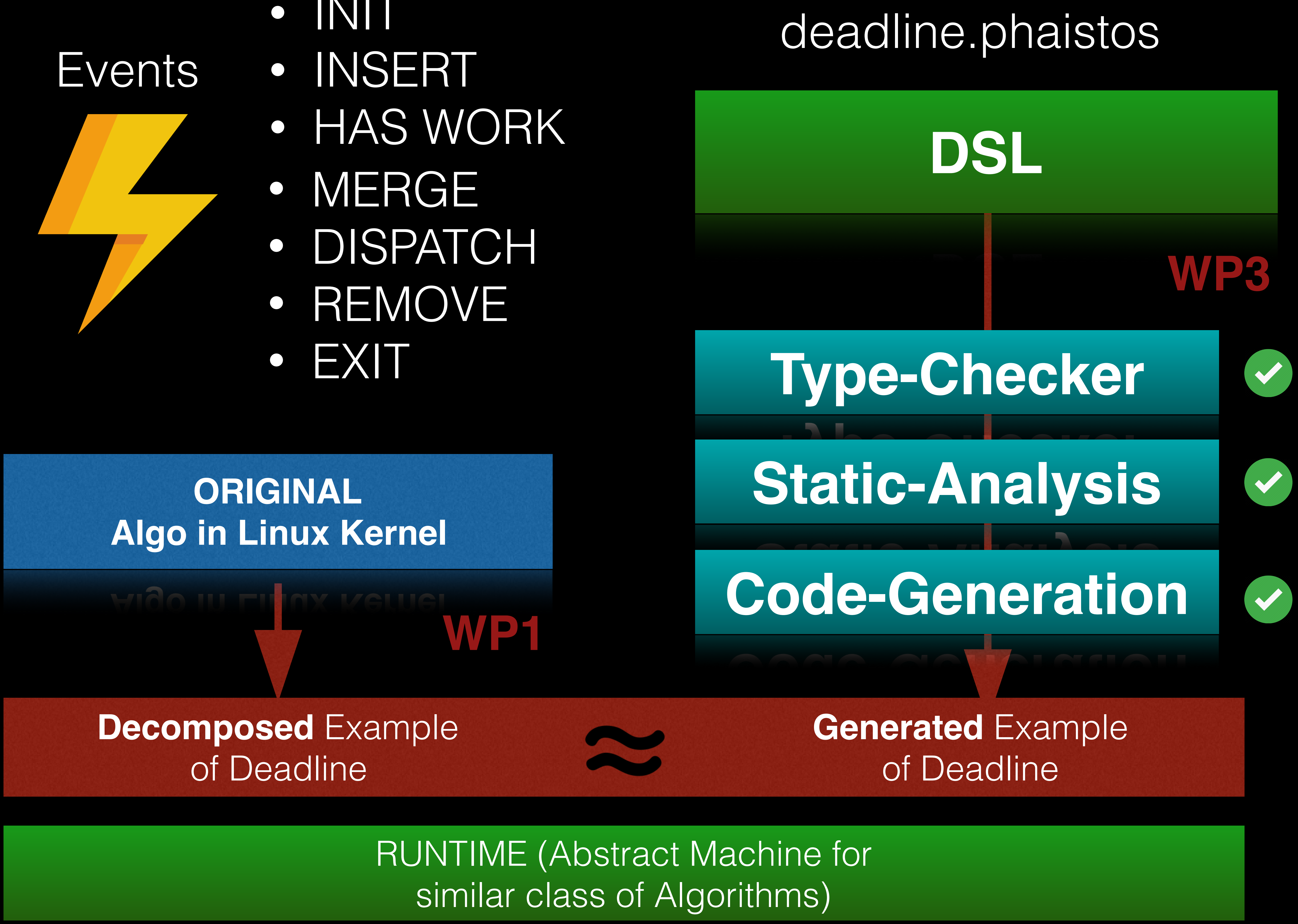
* Vanilla

* RAID 0 or 5 etc

* Set-up eg: https://www.hetzner.com/

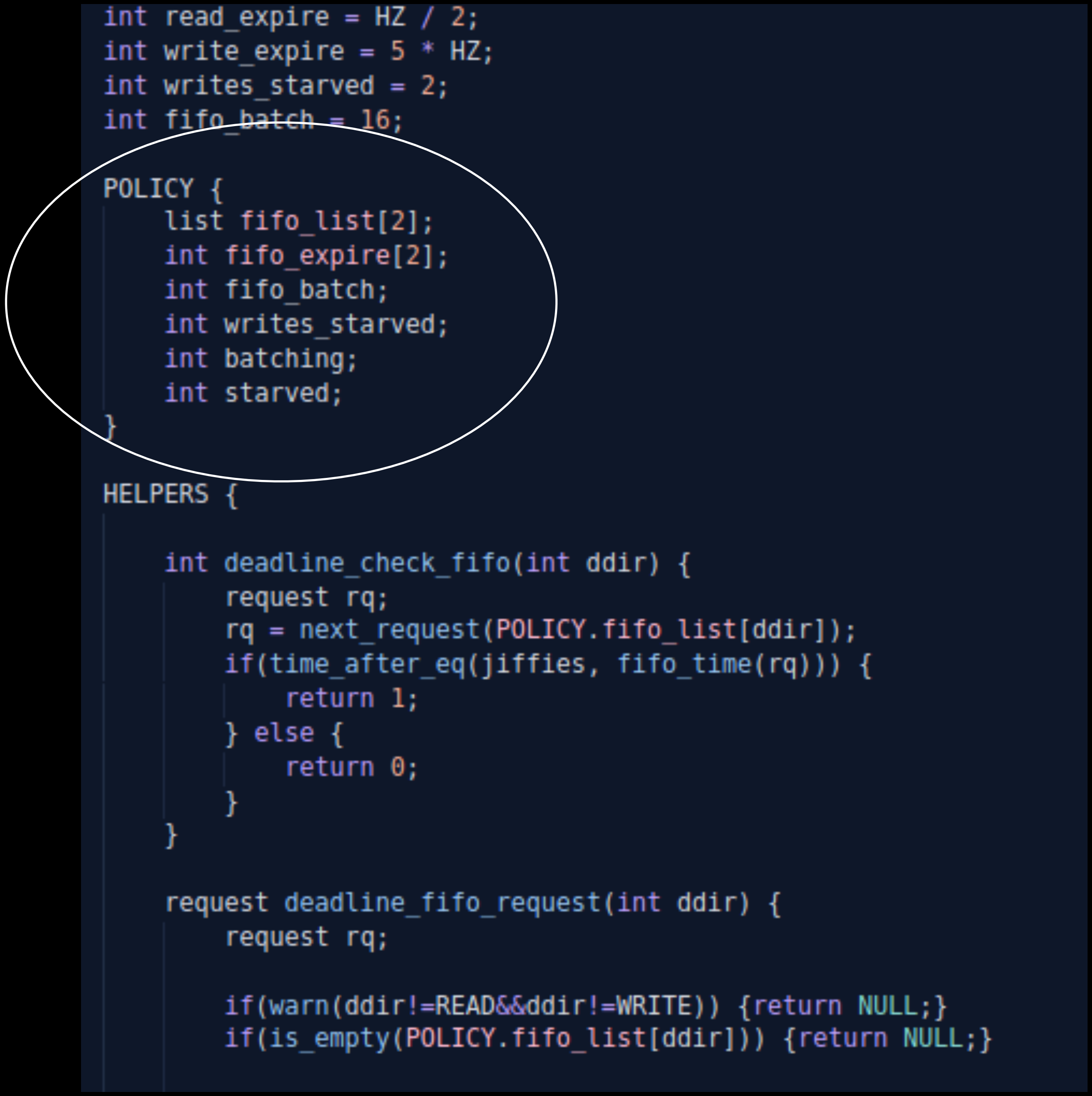
Events

- INIT
- INSERT
- HAS WORK
- MERGE
- DISPATCH
- REMOVE
- EXIT

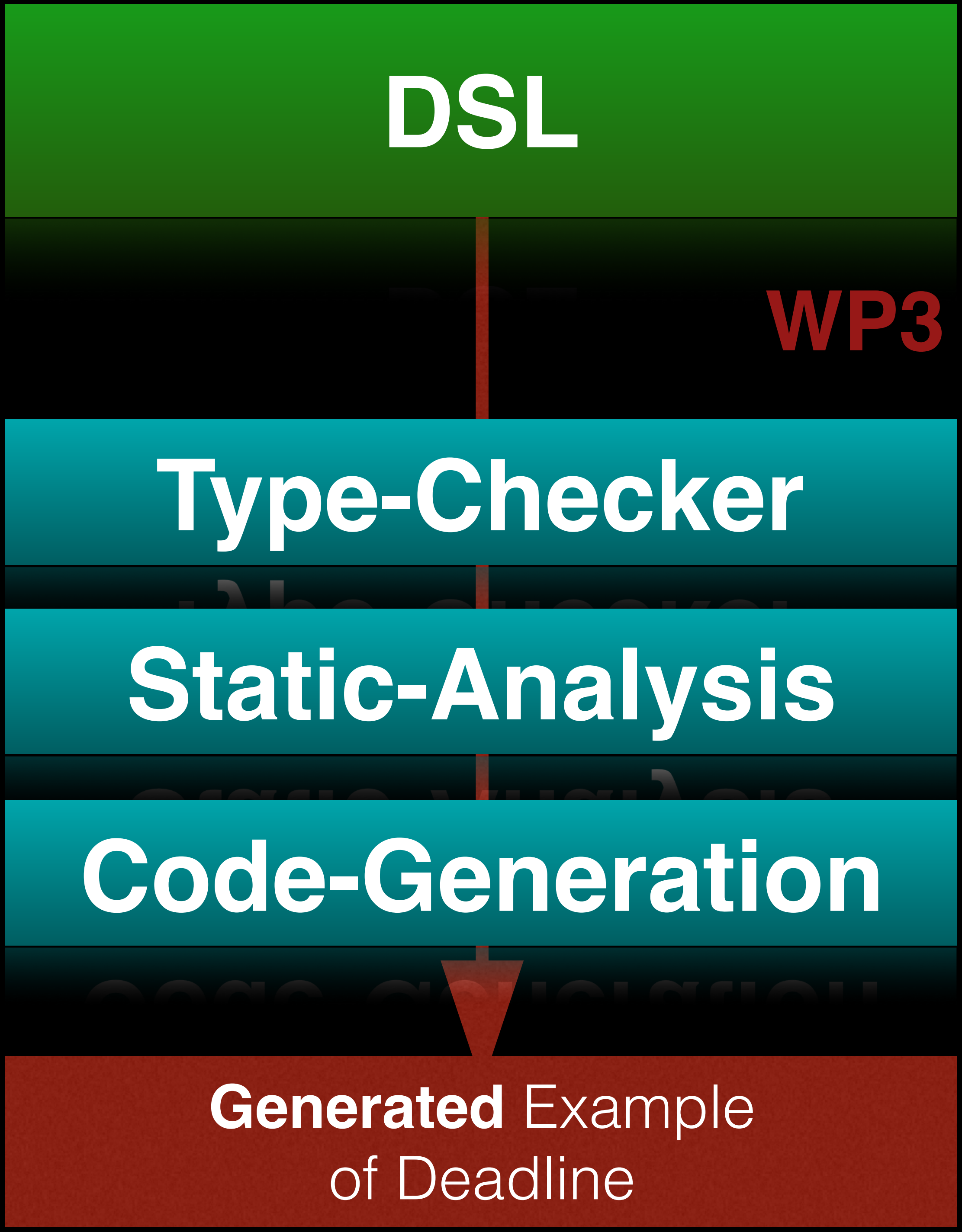

deadline.phaistos
DSL
WP3
Type-Checker ✓
Static-Analysis ✓
Code-Generation ✓
ORIGINAL
Algo in Linux Kernel
WP1
Decomposed Example of Deadline
≈
Generated Example of Deadline
RUNTIME (Abstract Machine for similar class of Algorithms)

list fifo_list[2]

deadline.phaistos

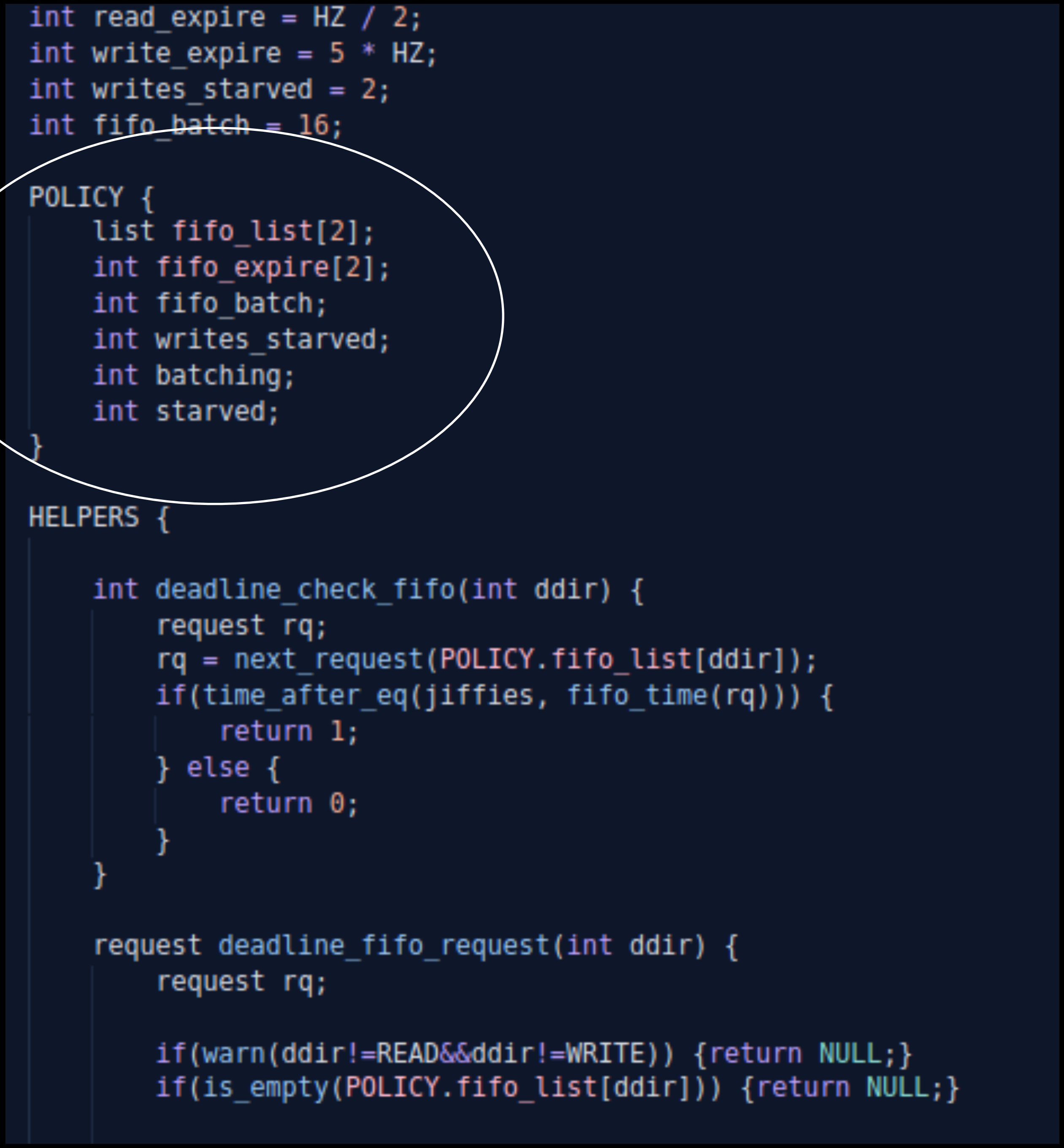


```
int read_expire = HZ / 2;
int write_expire = 5 * HZ;
int writes_starved = 2;
int fifo_batch = 16;

POLICY {
    list fifo_list[2];
    int fifo_expire[2];
    int fifo_batch;
    int writes_starved;
    int batching;
    int starved;
}

HELPERS {

    int deadline_check_fifo(int ddir) {
        request rq;
        rq = next_request(POLICY.fifo_list[ddir]);
        if(time_after_eq(jiffies, fifo_time(rq))) {
            return 1;
        } else {
            return 0;
        }
    }

    request deadline_fifo_request(int ddir) {
        request rq;

        if(warn(ddir!=READ&&ddir!=WRITE)) {return NULL;}
        if(is_empty(POLICY.fifo_list[ddir])) {return NULL;}
```



RUNTIME (Abstract Machine for
similar class of Algorithms)

list fifo_list[2]



```
int read_expire = HZ / 2;
int write_expire = 5 * HZ;
int writes_starved = 2;
int fifo_batch = 16;

POLICY {
    list fifo_list[2];
    int fifo_expire[2];
    int fifo_batch;
    int writes_starved;
    int batching;
    int starved;
}

HELPERS {

    int deadline_check_fifo(int ddir) {
        request rq;
        rq = next_request(POLICY.fifo_list[ddir]);
        if(time_after_eq(jiffies, fifo_time(rq))) {
            return 1;
        } else {
            return 0;
        }
    }

    request deadline_fifo_request(int ddir) {
        request rq;

        if(warn(ddir!=READ&&ddir!=WRITE)) {return NULL;}
        if(is_empty(POLICY.fifo_list[ddir])) {return NULL;}
```

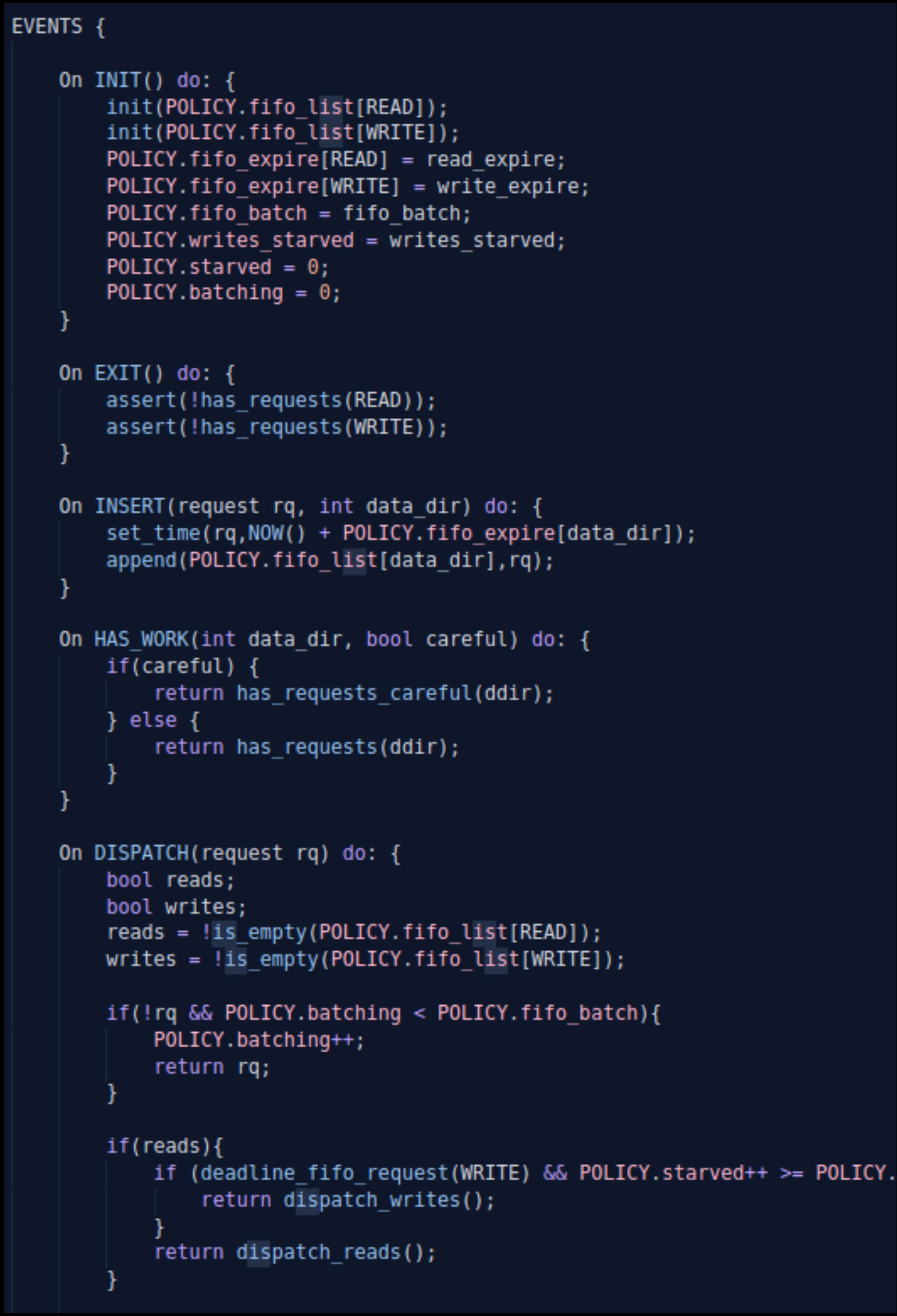
List API

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)

List API used in Events+Helpers:

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)

list fifo_list[2]



```
EVENTS {

    On INIT() do: {
        init(POLICY.fifo_list[READ]);
        init(POLICY.fifo_list[WRITE]);
        POLICY.fifo_expire[READ] = read_expire;
        POLICY.fifo_expire[WRITE] = write_expire;
        POLICY.fifo_batch = fifo_batch;
        POLICY.writes_starved = writes_starved;
        POLICY.starved = 0;
        POLICY.batching = 0;
    }

    On EXIT() do: {
        assert(!has_requests(READ));
        assert(!has_requests(WRITE));
    }

    On INSERT(request rq, int data_dir) do: {
        set_time(rq,NOW() + POLICY.fifo_expire[data_dir]);
        append(POLICY.fifo_list[data_dir],rq);
    }

    On HAS_WORK(int data_dir, bool careful) do: {
        if(careful) {
            return has_requests_careful(ddir);
        } else {
            return has_requests(ddir);
        }
    }

    On DISPATCH(request rq) do: {
        bool reads;
        bool writes;
        reads = !is_empty(POLICY.fifo_list[READ]);
        writes = !is_empty(POLICY.fifo_list[WRITE]);

        if(!rq && POLICY.batching < POLICY.fifo_batch){
            POLICY.batching++;
            return rq;
        }

        if(reads){
            if (deadline_fifo_request(WRITE) && POLICY.starved++ >= POLICY.
                return dispatch_writes();
            }
            return dispatch_reads();
        }
```

DSL Static Analysis Goal:

Consistency of lists between API calls given the semantics of events

NOTE

List API used in Events+Helpers:

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)

Events

- INIT
- INSERT
- HAS WORK
- MERGE
- DISPATCH
- REMOVE
- EXIT

Our framework computes <u>execution paths</u> and observes <u>execution events</u> for:

- list_id
- list_id [n]
- list_id [*]
- *

- execution: path list
- path: event list
- event: (list, op, in_loop)

Reasoning over the ordered events with a small <u>query language</u> given a specific execution <u>entry point</u>

DSL Static Analysis Goal:

Consistency of lists between API calls given the semantics of events

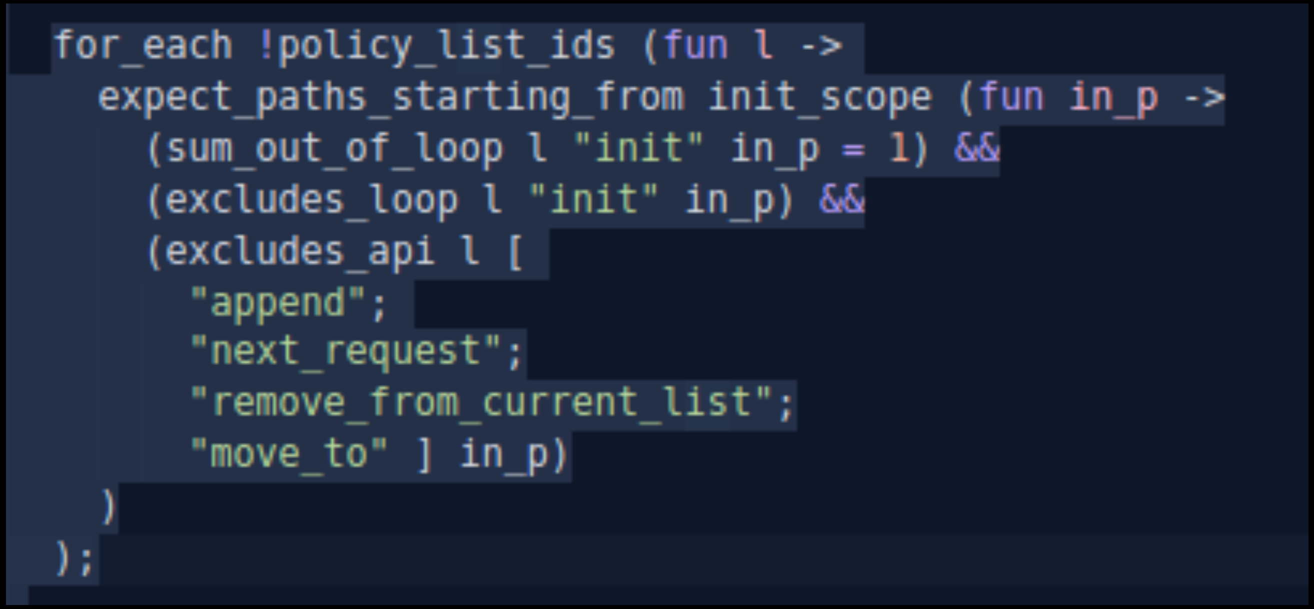
INIT

List API used in Events+Helpers:

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)

- INIT
- INSERT
- HAS WORK
- MERGE
- DISPATCH
- REMOVE
- EXIT

Events



```
for_each !policy_list_ids (fun l ->
  expect_paths_starting_from init_scope (fun in_p ->
    (sum_out_of_loop l "init" in_p = 1) &&
    (excludes_loop l "init" in_p) &&
    (excludes_api l [
      "append";
      "next_request";
      "remove_from_current_list";
      "move_to" ] in_p)
  )
);
```

∀ l = list defined in a policy
∀ p = execution path starting from event

<u>Every</u> list should be initialized <u>exactly once</u>, outside of a loop.

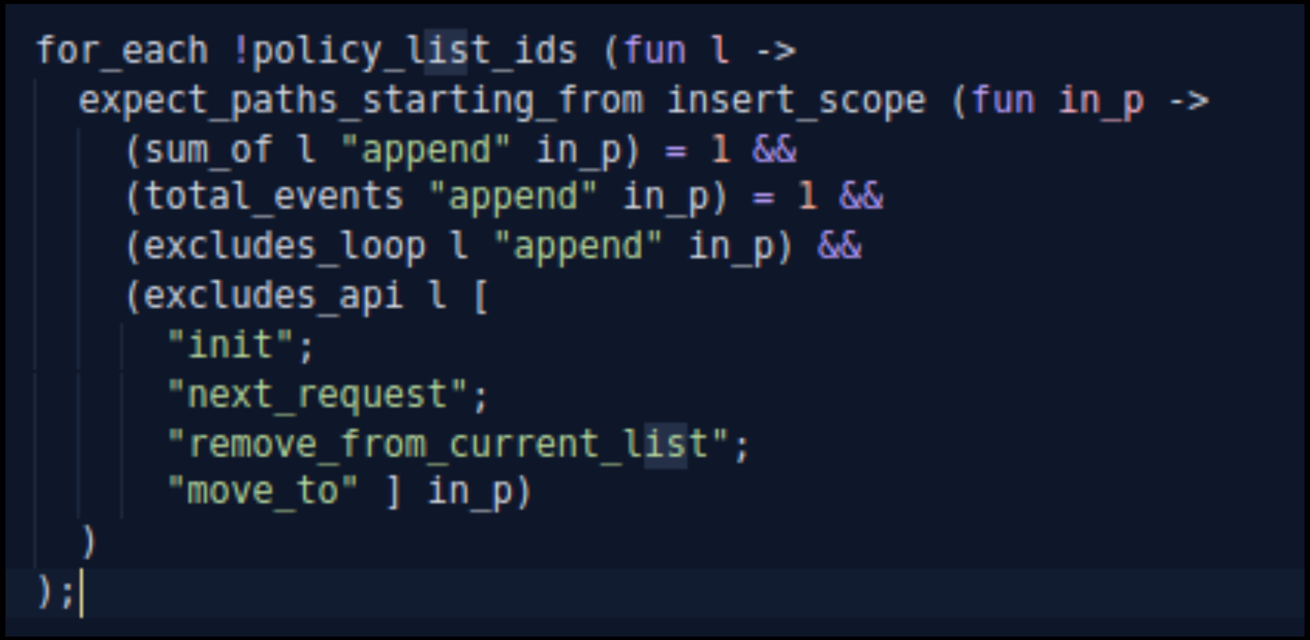
DSL Static Analysis Goal:

Consistency of lists between API calls given the semantics of events

INSERT

List API used in Events+Helpers:

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)



```
for_each !policy_list_ids (fun l ->
  expect_paths_starting_from insert_scope (fun in_p ->
    (sum_of l "append" in_p) = 1 &&
    (total_events "append" in_p) = 1 &&
    (excludes_loop l "append" in_p) &&
    (excludes_api l [
      "init";
      "next_request";
      "remove_from_current_list";
      "move_to" ] in_p)
  )
);
```

∀ l = list defined in a policy
∀ p = execution path starting from event

- INIT
- INSERT
- HAS WORK
- MERGE
- DISPATCH
- REMOVE
- EXIT

Events

Every list should be able to be appended exactly once and outside of a loop. Multiple appends of diff. lists per path are not allowed.

DSL Static Analysis Goal:

Consistency of lists between API calls given the semantics of events

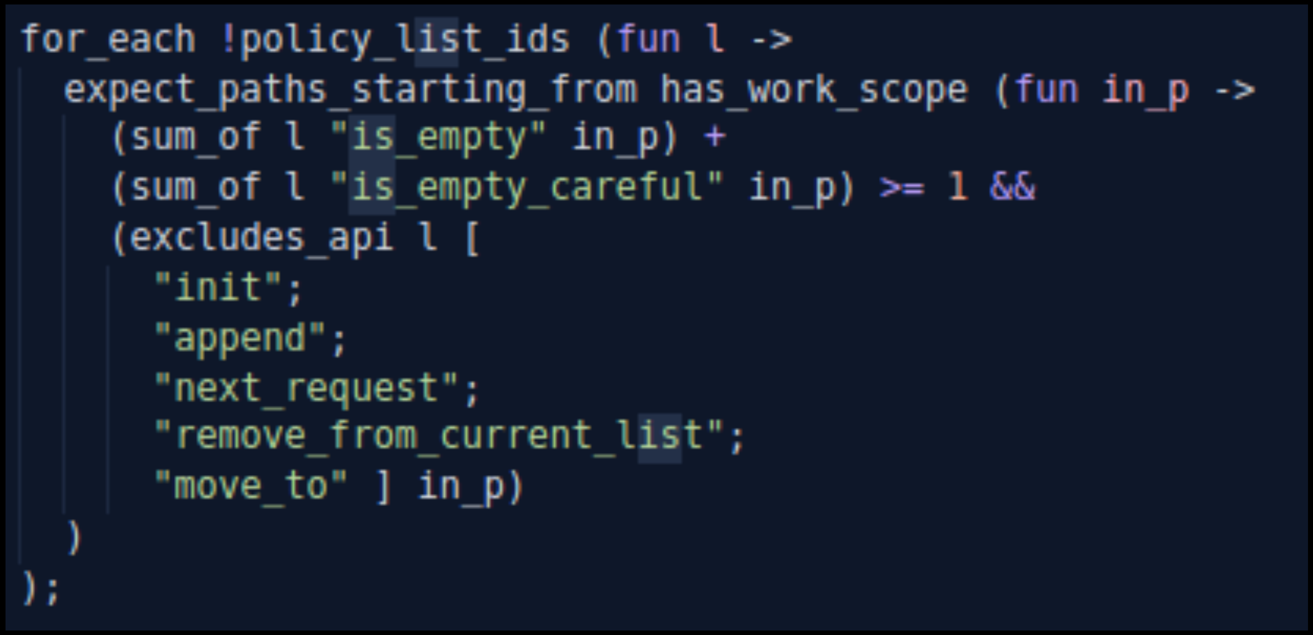
HAS WORK

List API used in Events+Helpers:

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)

- INIT
- INSERT
- HAS WORK
- MERGE
- DISPATCH
- REMOVE
- EXIT

Events



```
for_each !policy_list_ids (fun l ->
  expect_paths_starting_from has_work_scope (fun in_p ->
    (sum_of l "is_empty" in_p) +
    (sum_of l "is_empty_careful" in_p) >= 1 &&
    (excludes_api l [
      "init";
      "append";
      "next_request";
      "remove_from_current_list";
      "move_to" ] in_p)
  )
);
```

∀ l = list defined in a policy
∀ p = execution path starting from event

Every list should be able to be checked at least once.

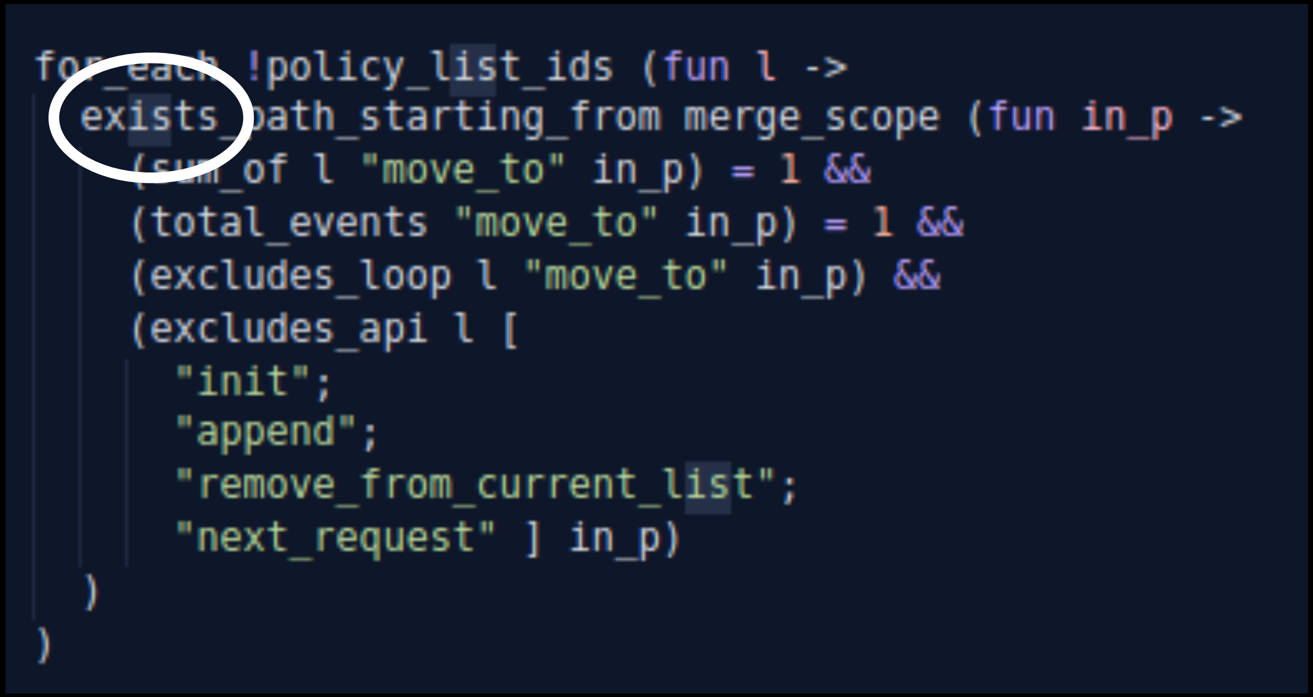
DSL Static Analysis Goal:

Consistency of lists between API calls given the semantics of events

MERGE

List API used in Events+Helpers:

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)



```
for_each !policy_list_ids (fun l ->
  exists path_starting_from merge_scope (fun in_p ->
    (sum_of l "move_to" in_p) = 1 &&
    (total_events "move_to" in_p) = 1 &&
    (excludes_loop l "move_to" in_p) &&
    (excludes_api l [
      "init";
      "append";
      "remove_from_current_list";
      "next_request" ] in_p)
  )
)
```

∀ l = list defined in a policy
∀ p = execution path starting from event

- INIT
- INSERT
- HAS WORK
- MERGE
- DISPATCH
- REMOVE
- EXIT

Events

Every list should be able to merge requests exactly once and outside of a loop. Multiple merges of diff. lists are not allowed.

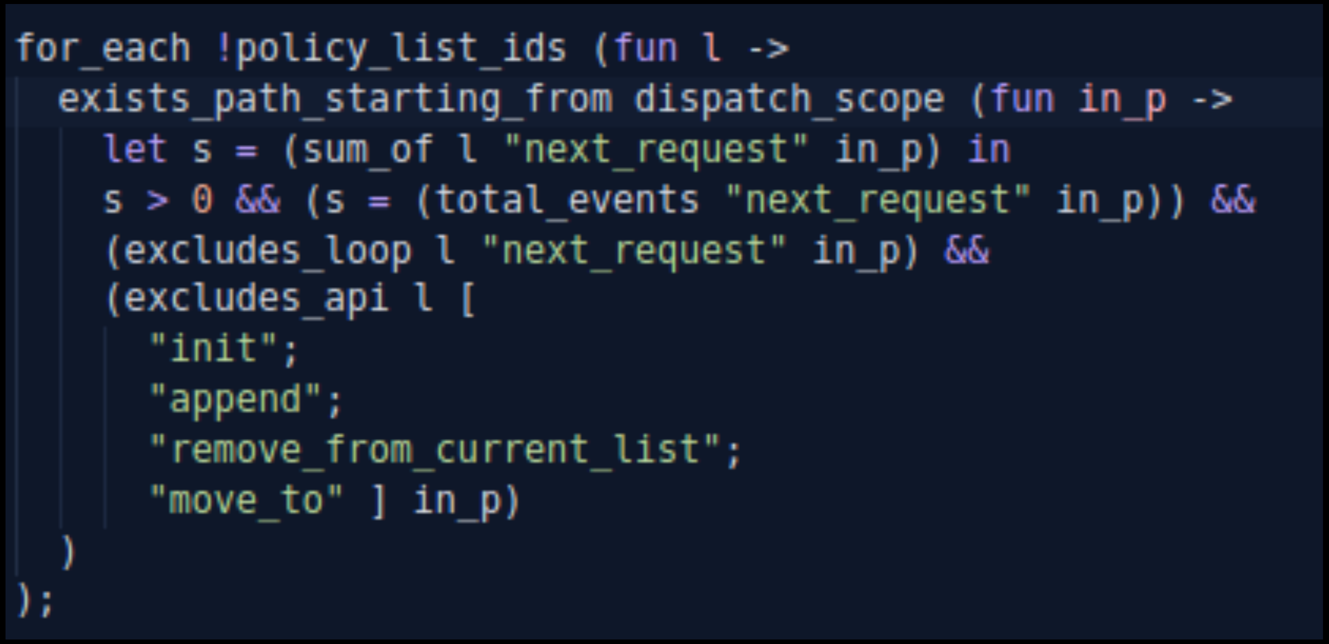
DSL Static Analysis Goal:

Consistency of lists between API calls given the semantics of events

DISPATCH

List API used in Events+Helpers:

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)



```
for_each !policy_list_ids (fun l ->
  exists_path_starting_from dispatch_scope (fun in_p ->
    let s = (sum_of l "next_request" in_p) in
    s > 0 && (s = (total_events "next_request" in_p)) &&
    (excludes_loop l "next_request" in_p) &&
    (excludes_api l [
      "init";
      "append";
      "remove_from_current_list";
      "move_to" ] in_p)
  )
);
```

∀ l = list defined in a policy
∀ p = execution path starting from event

- INIT
- INSERT
- HAS WORK
- MERGE
- DISPATCH
- REMOVE
- EXIT

Events

<u>Every</u> list should be able to handle requests <u>at least once</u> and outside of a loop. <u>Multiple</u> requests from diff. lists are <u>not allowed</u>.

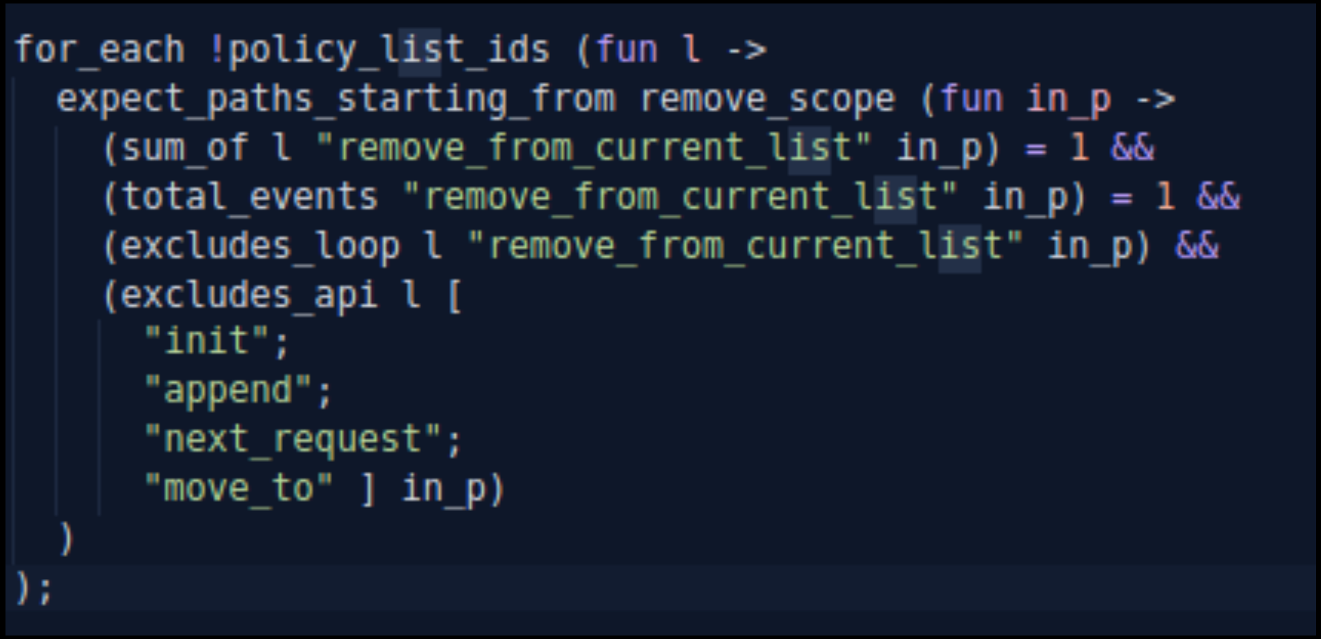
DSL Static Analysis Goal:

Consistency of lists between API calls given the semantics of events

REMOVE

List API used in Events+Helpers:

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)



```
for_each !policy_list_ids (fun l ->
  expect_paths_starting_from remove_scope (fun in_p ->
    (sum_of l "remove_from_current_list" in_p) = 1 &&
    (total_events "remove_from_current_list" in_p) = 1 &&
    (excludes_loop l "remove_from_current_list" in_p) &&
    (excludes_api l [
      "init";
      "append";
      "next_request";
      "move_to" ] in_p)
  )
);
```

∀ l = list defined in a policy
∀ p = execution path starting from event

- INIT
- INSERT
- HAS WORK
- MERGE
- DISPATCH
- REMOVE
- EXIT

Events

Every list should be able to remove requests exactly once and outside of a loop. Multiple requests from diff. lists are not allowed.

DSL Static Analysis Goal:

Consistency of lists between API calls given the semantics of events

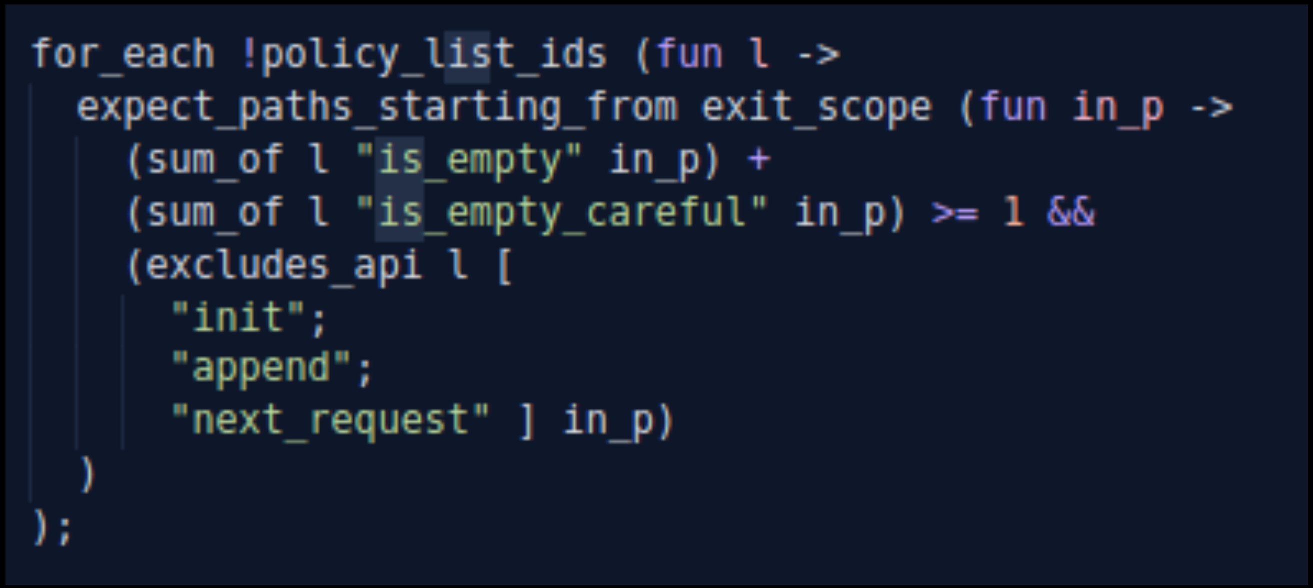
EXIT

List API used in Events+Helpers:

- init(list)
- append(list,request)
- remove_from_current_list (request)
- move_to(request,request)
- next_request(list)
- is_empty(list)
- is_empty_careful(list)

- INIT
- INSERT
- HAS WORK
- MERGE
- DISPATCH
- REMOVE
- EXIT

Events



```
for_each !policy_list_ids (fun l ->
  expect_paths_starting_from exit_scope (fun in_p ->
    (sum_of l "is_empty" in_p) +
    (sum_of l "is_empty_careful" in_p) >= 1 &&
    (excludes_api l [
      "init";
      "append";
      "next_request" ] in_p)
  )
);
```

∀ l = list defined in a policy
∀ p = execution path starting from event

Every list should be able to be checked at least once upon exit.

Veriamos Project

SSD I/O via a DSL @ Ring 0

PhaistOS DSL

2019

Dr. Nick Papoulias http://parsenet.info