ARA: Static Initialization of Dynamically-Created System Objects

Björn Fiedler, Gerion Entrup, Christian Dietrich, Daniel Lohmann

Leibniz Universität Hannover

March 12, 2021

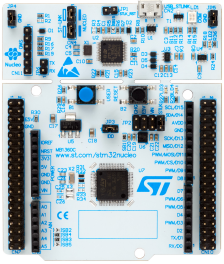
supported by DFG

SSP: On Your Marks, Get Set, Go!

```
QueueHandle_t q1;

void f1() { xQueueSend(q1, create()); }
void f2() { msg_t e; while(xQueueReceive(q1, &e)) process(e); }

void init_os() {
  xTaskCreate("T1", f1, 4);
  xTaskCreate("T2", f2, dyn_prio());
  q1 = xQueueCreate(10, sizeof(msg_t));
}

int main() {
  init_serial();
  init_os();
  printf("System ready!");
  vTaskStartScheduler();
}
```

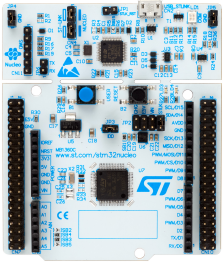
Dynamic Boot Time

- Run the example on a Nucleo STM32-F103RB
- Boot time: **40,048 cycles**



Dynamic Boot Time

- Run the example on a Nucleo STM32-F103RB
- Boot time: **40,048 cycles**
- Unwanted **dynamic** OS overhead
- Do the same thing every boot again
- First deadline to reach



Dynamic Boot Time

- Run the example on a Nucleo STM32-F103RB
- Boot time: **40,048 cycles**
- Unwanted **dynamic** OS overhead
- Do the same thing every boot again
- First deadline to reach

Experiment: Skip the OS initialization

```
int main() {
  init_serial();
  // init_os();
  printf("System ready!")
  vTaskStartScheduler();
}
```

Dynamic Boot Time

- Run the example on a Nucleo STM32-F103RB
- Boot time: **40,048 cycles**
- Unwanted **dynamic** OS overhead
- Do the same thing every boot again
- First deadline to reach

Experiment: Skip the OS initialization

```
int main() {
  init_serial();
  // init_os();
  printf("System ready!")
  vTaskStartScheduler();
}
```

Boot time: **15,628 cycles**

Dynamic Boot Time

- Run the example on a Nucleo STM32-F103RB
- Boot time: **40,048 cycles**
- Unwanted **dynamic** OS overhead
- Do the same thing every boot again
- First deadline to reach

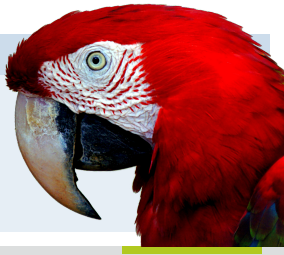
Experiment: Skip the OS initialization

```
int main() {
  init_serial();
  // init_os();
  printf("System ready!")
  vTaskStartScheduler();
}
```

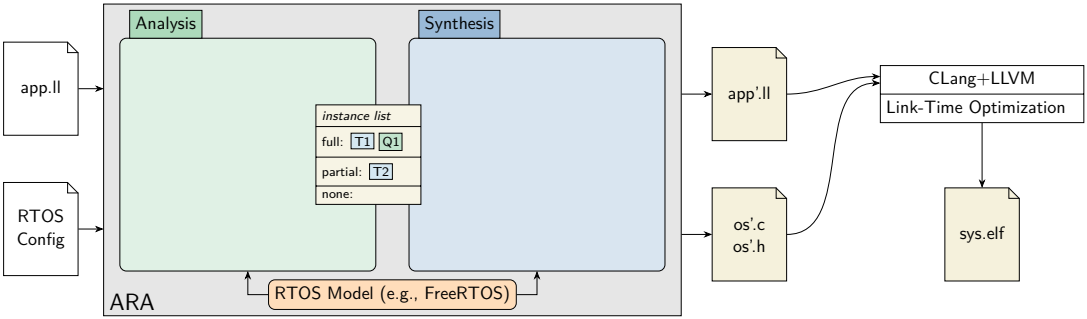
Boot time: **15,628 cycles**

ARA

- Whole system optimizer
- Static initialization
- Fully automatic and OS independent



ARA in a Nutshell



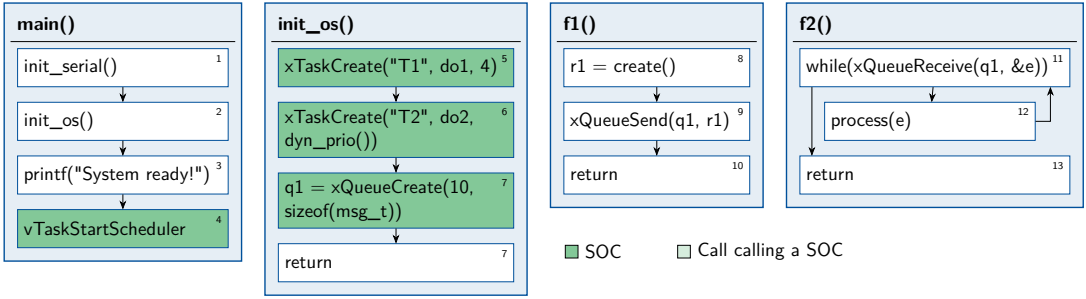
The Instance List

```
QueueHandle_t q1;

void f1() { xQueueSend(q1, create()); }
void f2() { msg_t e; while(xQueueReceive(q1, &e)) process(e); }

void init_os() {
  xTaskCreate("T1", f1, 4);
  xTaskCreate("T2", f2, dyn_prio());
  q1 = xQueueCreate(10, sizeof(msg_t));
}

int main() {
  init_serial();
  init_os();
  printf("System ready!");
  vTaskStartScheduler();
}
```

System Objects	T1	T2	Q1
Created before scheduler	✓	✓	✓
Created exactly once	✓	✓	✓
RTOS allocates memory	✓	✓	✓
All parameters known	✓	✗	✓

How to get it automatically?

Interprocedural CFG

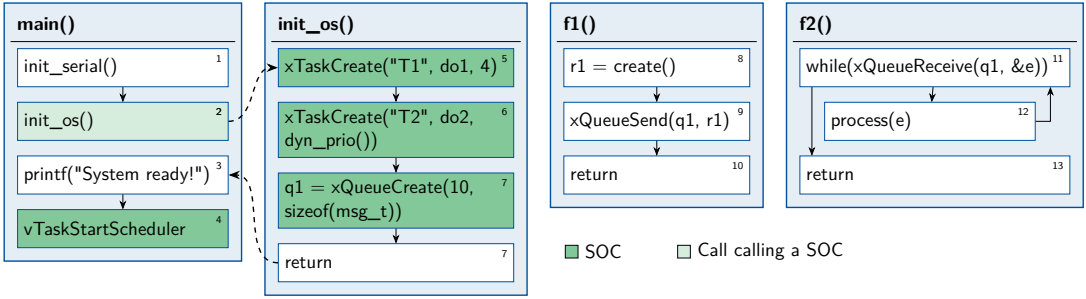


Difficulty: Function Pointer

Simple pointer: Andersen's pointer analysis

Complex pointer: Type based signature comparison

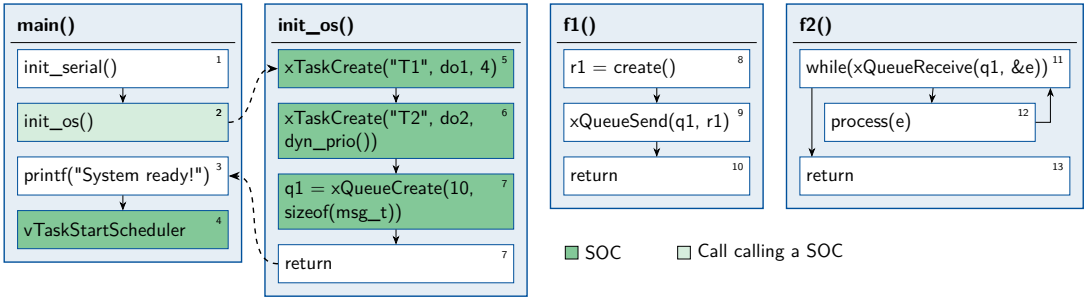
Interprocedural CFG



Difficulty: Function Pointer

Simple pointer:	Andersen's pointer analysis
Complex pointer:	Type based signature comparison

Static Instance Analysis



Entry Points:

main()

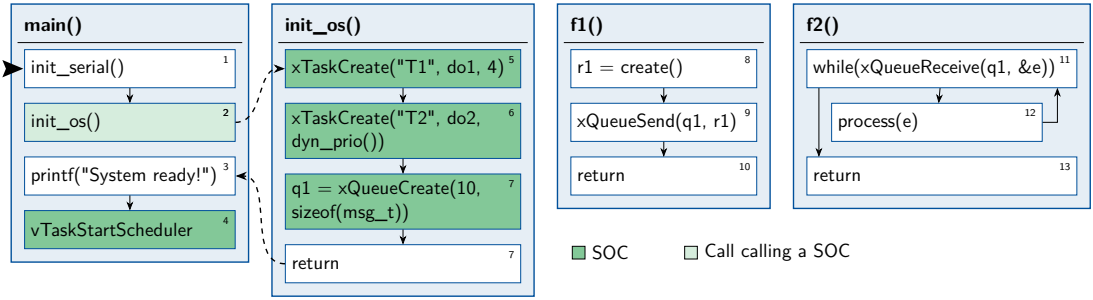
Call Stack:

System Objects:

Created before scheduler
Created exactly once
RTOS allocates memory
All parameters known

Parameter:

Static Instance Analysis



Entry Points:

Call Stack:

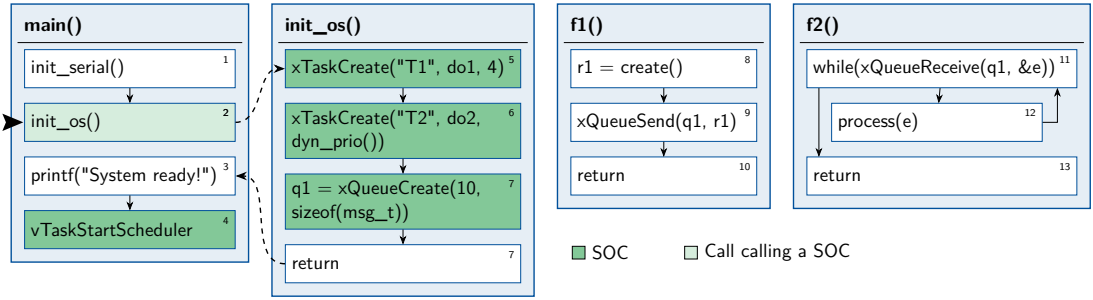
_: main()

System Objects:

Created before scheduler
Created exactly once
RTOS allocates memory
All parameters known

Parameter:

Static Instance Analysis



Entry Points:

Call Stack:

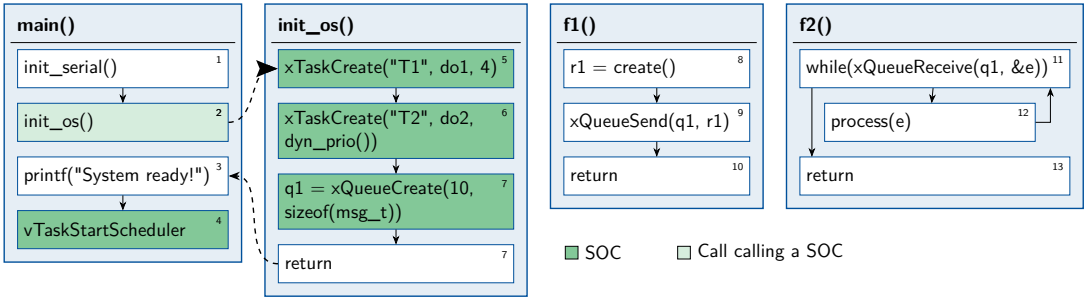
_: main()

System Objects:

Created before scheduler
Created exactly once
RTOS allocates memory
All parameters known

Parameter:

Static Instance Analysis



Entry Points:

f1()

Call Stack:

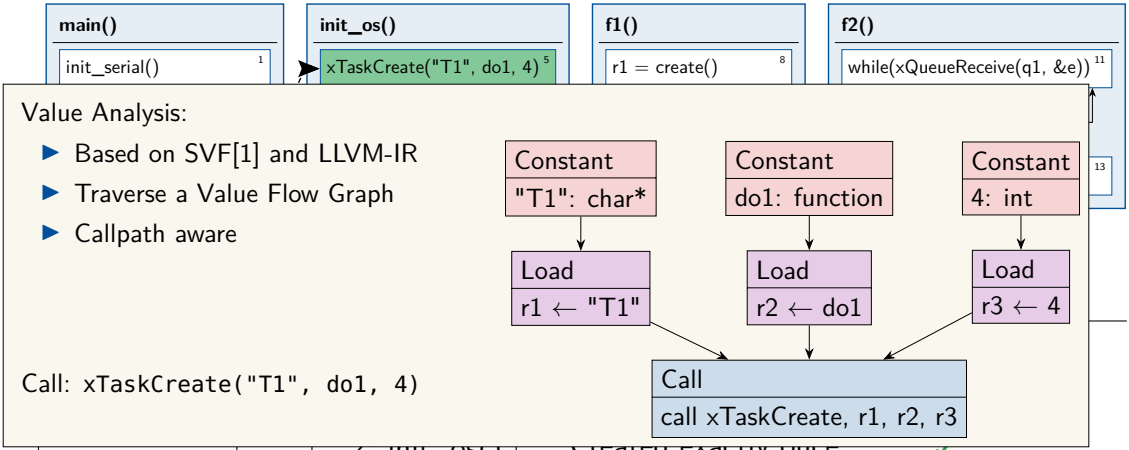
_: main()

2: init_os()

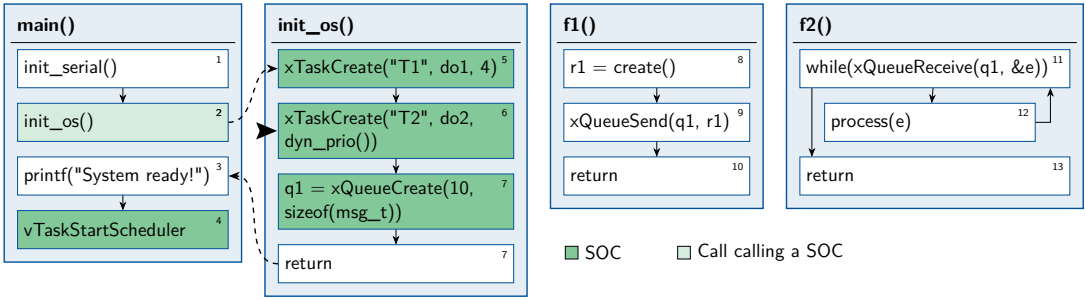
Parameter: "T1", do1, 4

System Objects:	T1
Created before scheduler	✓
Created exactly once	✓
RTOS allocates memory	✓
All parameters known	✓

Static Instance Analysis



Static Instance Analysis



Entry Points:

- f1()
- f2()

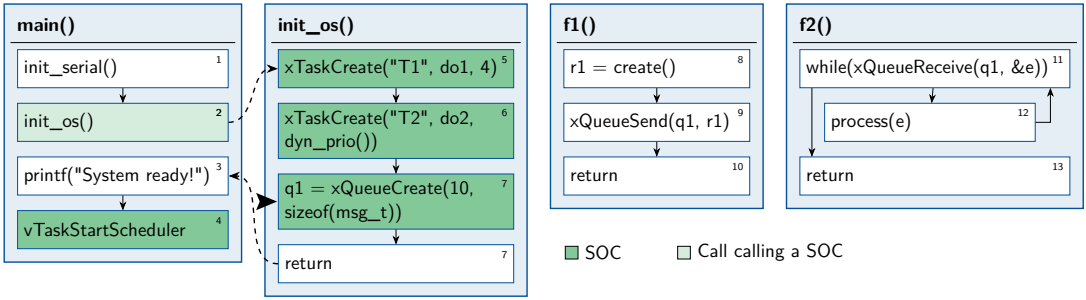
Call Stack:

- _: main()
- 2: init_os()

Parameter: "T2", do2, ?

System Objects:	T1	T2
Created before scheduler	✓	✓
Created exactly once	✓	✓
RTOS allocates memory	✓	✓
All parameters known	✓	✗

Static Instance Analysis



Entry Points:

- f1()
- f2()

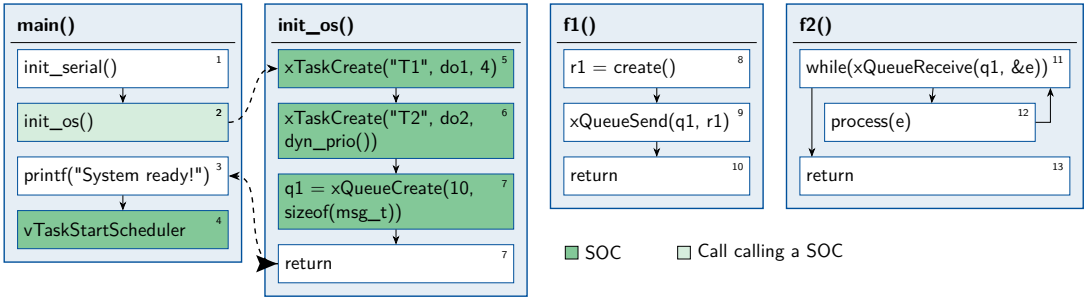
Call Stack:

- _: main()
- 2: init_os()

Parameter: 10, 8

System Objects:	T1	T2	Q1
Created before scheduler	✓	✓	✓
Created exactly once	✓	✓	✓
RTOS allocates memory	✓	✓	✓
All parameters known	✓	✗	✓

Static Instance Analysis



Entry Points:

- f1()
- f2()

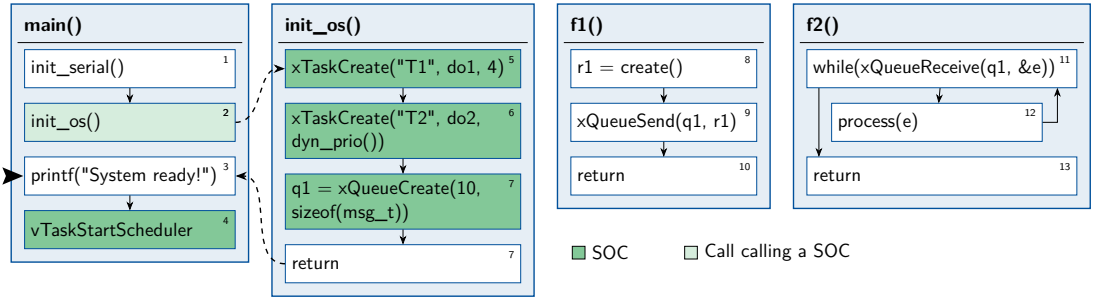
Call Stack:

- _: main()
- 2: init_os()

Parameter:

System Objects:	T1	T2	Q1
Created before scheduler	✓	✓	✓
Created exactly once	✓	✓	✓
RTOS allocates memory	✓	✓	✓
All parameters known	✓	✗	✓

Static Instance Analysis



Entry Points:

- f1()
- f2()

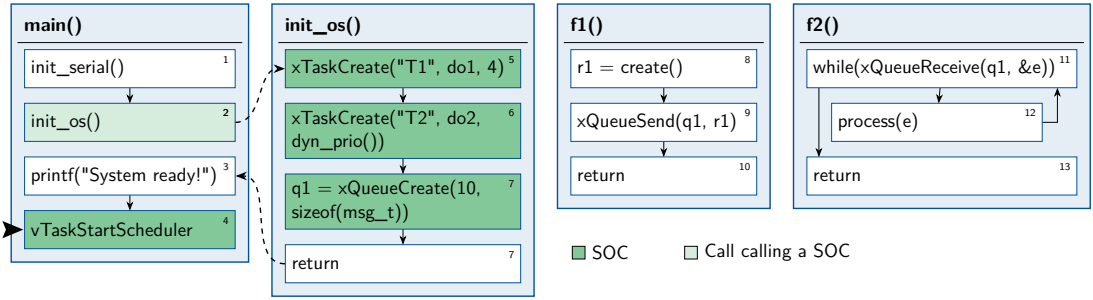
Call Stack:

_: main()

Parameter:

System Objects:	T1	T2	Q1
Created before scheduler	✓	✓	✓
Created exactly once	✓	✓	✓
RTOS allocates memory	✓	✓	✓
All parameters known	✓	✗	✓

Static Instance Analysis



Entry Points:

f1()

f2()

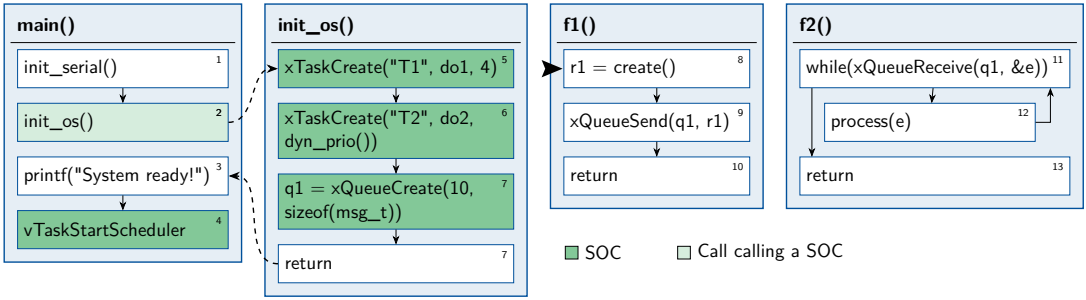
Call Stack:

_: main()

Parameter:

System Objects:	T1	T2	Q1
Created before scheduler	✓	✓	✓
Created exactly once	✓	✓	✓
RTOS allocates memory	✓	✓	✓
All parameters known	✓	✗	✓

Static Instance Analysis



Entry Points:

f2()

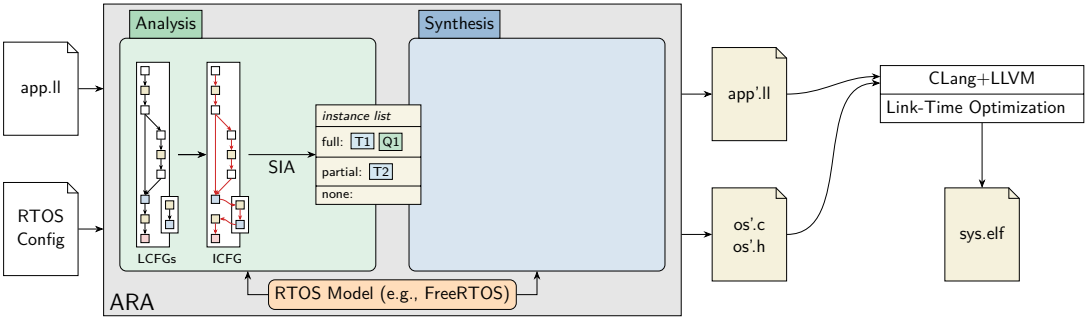
Call Stack:

_: t1()

Parameter:

System Objects:	T1	T2	Q1
Created before scheduler	✓	✓	✓
Created exactly once	✓	✓	✓
RTOS allocates memory	✓	✓	✓
All parameters known	✓	✗	✓

ARA in a Nutshell



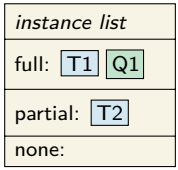
The Instance List Revisited

System Objects	T1	T2	Q1
Created before scheduler	✓	✓	✓
Created exactly once	✓	✓	✓
RTOS allocates memory	✓	✓	✓
All parameters known	✓	✗	✓

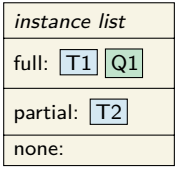
The Instance List Revisited

System Objects	T1	T2	Q1
Created before scheduler	✓	✓	✓
Created exactly once	✓	✓	✓
RTOS allocates memory	✓	✓	✓
All parameters known	✓	✗	✓

Categorization of Specialization Depth

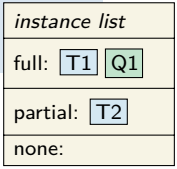


Synthesize and Replace



Synthesize and Replace

```
+ InitStack_t<512> t1_stack(f2);

+ TCB_t t1_tcb = {
+  .StateListItem = {
+    .Previous = &ReadyLists[4].end,
+    .Next = &idle_task_tcb },
+  .Priority = 0,
+  .Stack = &t1_stack,
+    /* ... */
+ };

+ List_t ReadyLists[5] = { {
+  .ListEnd = {
+    .Next = &task2_tcb,
+    .Previous = &idle_task_tcb
+  },
+  .NumberOfItems = 2,
+  }, /* ... */
+ };
```



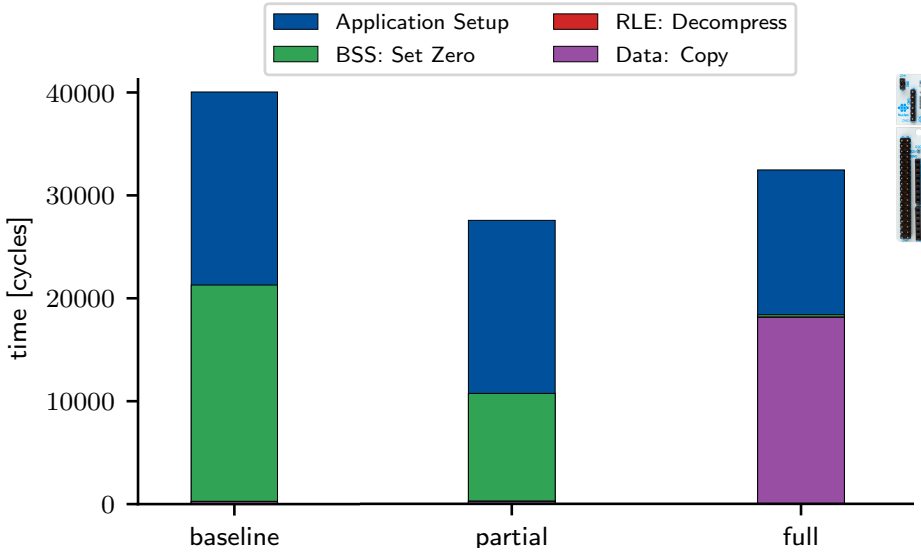
Synthesize and Replace

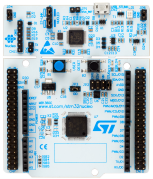
```
+ InitStack_t<512> t1_stack(f2);

+ TCB_t t1_tcb = {
+  .StateListItem = {
+    .Previous = &ReadyLists[4].end,
+    .Next = &idle_task_tcb },
+  .Priority = 0,
+  .Stack = &t1_stack,
+    /* ... */
+ };

+ List_t ReadyLists[5] = { {
+  .ListEnd = {
+    .Next = &task2_tcb,
+    .Previous = &idle_task_tcb
+  },
+  .NumberOfItems = 2,
+  }, /* ... */
+ };
```

instance list
full: T1 Q1
partial: T2
none:

```
  QueueHandle_t q1;

  void f1() { /* ...  */ }
  void f2() { /* ...  */ }

  void init_os() {
-   xTaskCreate("T1", f1, 4);
-   xTaskCreate("T2", f2, dyn_prio());
+   xTaskActivate(t2_tcb, dyn_prio());
-   q1 = xQueueCreate(10, sizeof(msg_t));
  }

  int main() {
    init_serial();
    init_os();
    printf("System ready!");
-   vTaskStartScheduler();
+   vDispatch();
  }
```

First Results





Binary Post Processing

- Link-time optimization
  - Constant propagation
  - Inlining
- Sparse populated data
  - OS objects (stack, queue, ...)
  - Libraries (SdFat, ...)

```
xTaskCreate("t1");
SdFatFile::SdFatFile() {
  mtx = xCreateMutex();
}
SdFatFile file();
```

ARA + LLVM

```
char t1_stack[512] = {arg, ret, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..., 0);
TCB_t tcb_t1 = { /* ... */};
mutex_t mtx_anon_1 = { /* ... */ };
SdFatFile = {.mtx = &mtx_anon_1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..., 0};
}
```

Binary Post Processing

- Link-time optimization
  - Constant propagation
  - Inlining
- Sparse populated data
  - OS objects (stack, queue, ...)
  - Libraries (SdFat, ...)
- Run-length encoding
  - Less flash usage
  - Less copy time

```
xTaskCreate("t1");
SdFatFile::SdFatFile() {
  mtx = xCreateMutex();
}
SdFatFile file();
```

ARA + LLVM

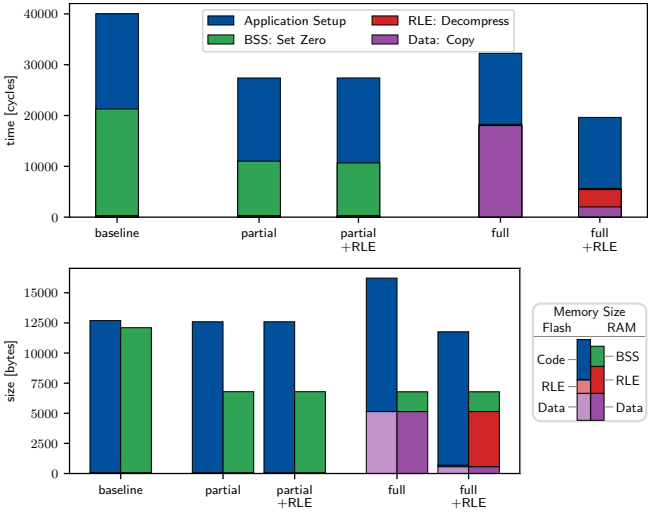
```
char t1_stack[512] = {arg, ret, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..., 0);
TCB_t tcb_t1 = { /* ... */};
mutex_t mtx_anon_1 = { /* ... */ };
SdFatFile = {.mtx = &mtx_anon_1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..., 0};
}
```

ARA + LLVM + RLE

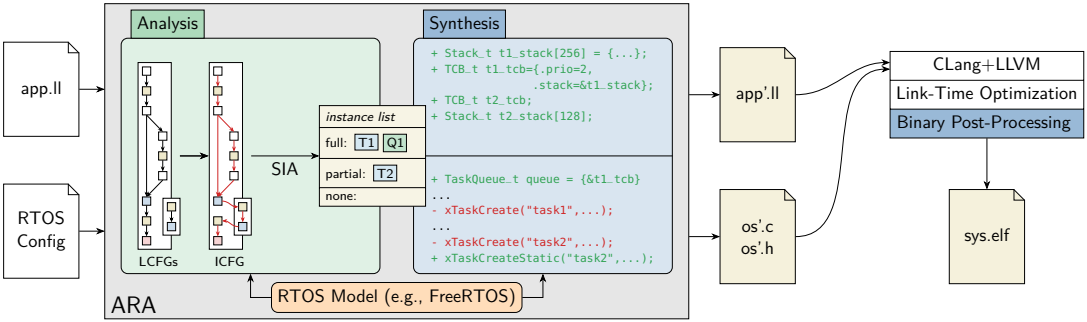
```
char stack1_comp = {(1<<31) + 2, arg, ret, 510, 0};
char file_comp = {1, &mtx_anon_1, 342, 0};
```

Binary Post Processing

- Link-time optimization
  - Constant propagation
  - Inlining
- Sparse populated data
  - OS objects (stack, queue, ...)
  - Libraries (SdFat, ...)
- Run-length encoding
  - Less flash usage
  - Less copy time



ARA in a Nutshell



Evaluation

- Microbenchmarks

- Real-world applications

Evaluation

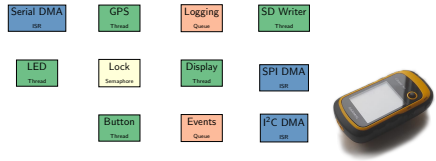
- Microbenchmarks
  - Queues
  - Tasks, pre scheduler
  - Tasks, post scheduler
- Real-world applications

```
1  int main() {
2    xTaskCreate("T001", f1, ...);
3    xTaskCreate("T002", f1, ...);
4    xTaskCreate("T003", f1, ...);
5    xTaskCreate("T004", f1, ...);
6    xTaskCreate("T005", f1, ...);
7    xTaskCreate("T006", f1, ...);
8    xTaskCreate("T007", f1, ...);
9    xTaskCreate("T008", f1, ...);
10   xTaskCreate("T009", f1, ...);
11   /* ... */
12   xTaskCreate("T030", f1, ...);
13   vTaskStartScheduler();
14 }
```

Evaluation

- Microbenchmarks
  - Queues
  - Tasks, pre scheduler
  - Tasks, post scheduler
- Real-world applications
  - GPSLogger

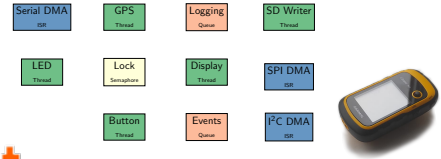
```
1  int main() {
2    xTaskCreate("T001", f1, ...);
3    xTaskCreate("T002", f1, ...);
4    xTaskCreate("T003", f1, ...);
5    xTaskCreate("T004", f1, ...);
6    xTaskCreate("T005", f1, ...);
7    xTaskCreate("T006", f1, ...);
8    xTaskCreate("T007", f1, ...);
9    xTaskCreate("T008", f1, ...);
10   xTaskCreate("T009", f1, ...);
11   /* ... */
12   xTaskCreate("T030", f1, ...);
13   vTaskStartScheduler();
14 }
```



Evaluation

- Microbenchmarks
  - Queues
  - Tasks, pre scheduler
  - Tasks, post scheduler
- Real-world applications
  - GPSLogger
  - Librepilot

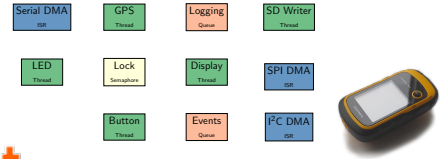
```
1  int main() {
2    xTaskCreate("T001", f1, ...);
3    xTaskCreate("T002", f1, ...);
4    xTaskCreate("T003", f1, ...);
5    xTaskCreate("T004", f1, ...);
6    xTaskCreate("T005", f1, ...);
7    xTaskCreate("T006", f1, ...);
8    xTaskCreate("T007", f1, ...);
9    xTaskCreate("T008", f1, ...);
10   xTaskCreate("T009", f1, ...);
11   /* ... */
12   xTaskCreate("T030", f1, ...);
13   vTaskStartScheduler();
14 }
```



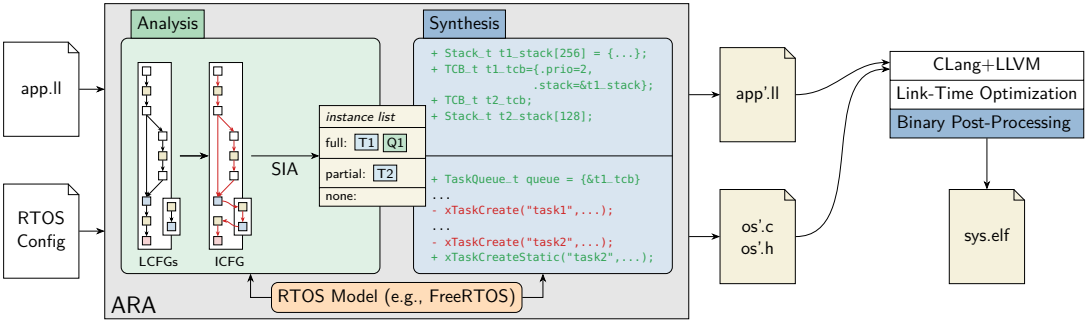
Evaluation

- Microbenchmarks
  - Queues
  - Tasks, pre scheduler
  - Tasks, post scheduler
- Real-world applications
  - GPSLogger
  - Librepilot
- Result:
  - Up to 43% boot-time reduction
  - Moderate flash usage increase

```
1  int main() {
2    xTaskCreate("T001", f1, ...);
3    xTaskCreate("T002", f1, ...);
4    xTaskCreate("T003", f1, ...);
5    xTaskCreate("T004", f1, ...);
6    xTaskCreate("T005", f1, ...);
7    xTaskCreate("T006", f1, ...);
8    xTaskCreate("T007", f1, ...);
9    xTaskCreate("T008", f1, ...);
10   xTaskCreate("T009", f1, ...);
11   /* ... */
12   xTaskCreate("T030", f1, ...);
13   vTaskStartScheduler();
14 }
```

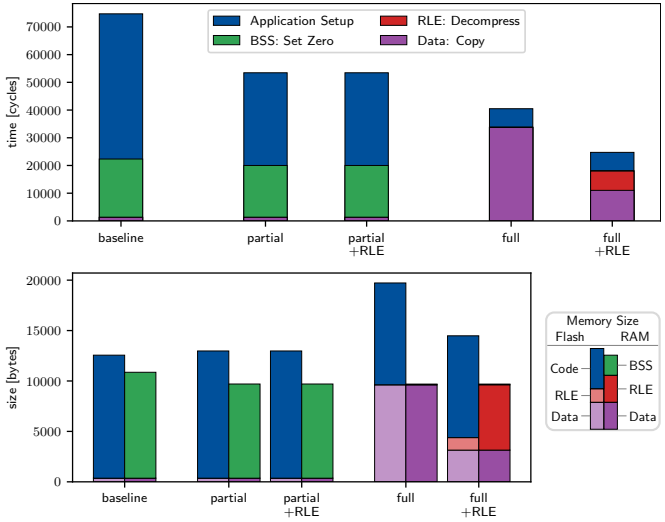


ARA: Static Initialization of Dynamically-Created System Objects

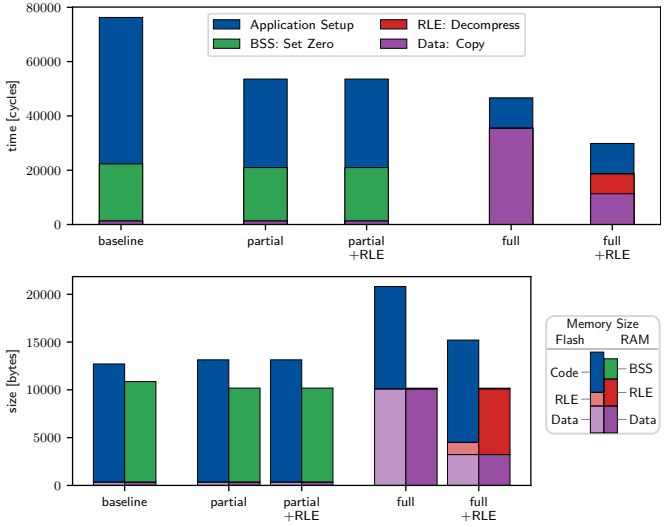


- Evaluated at microbenchmarks and two real-world applications
  - Up to 43% boot-time reduction
  - Moderate flash usage increase
- Open Source (`https://github.com/luhsra/ara`)
- Accepted at RTAS'21

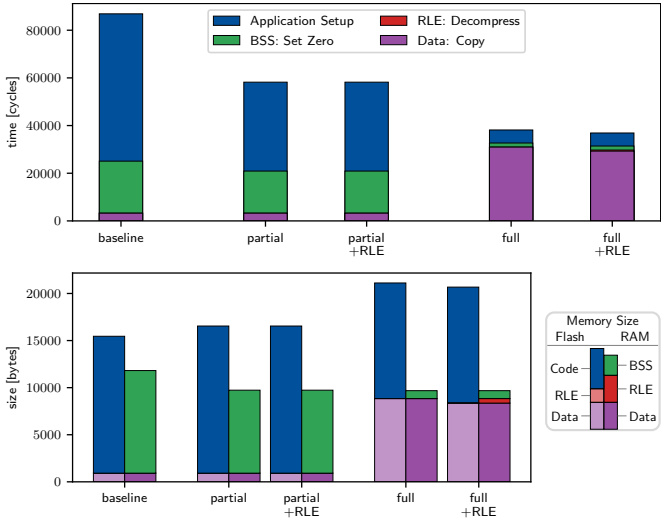
Evaluation – Tasks pre Scheduler



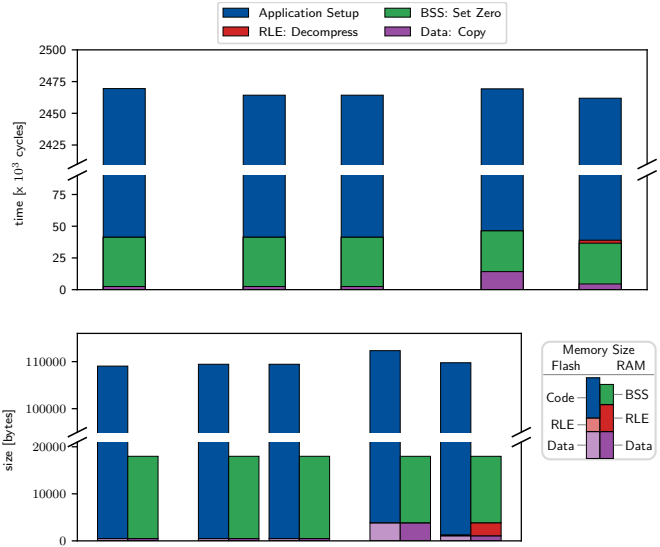
Evaluation – Tasks post Scheduler



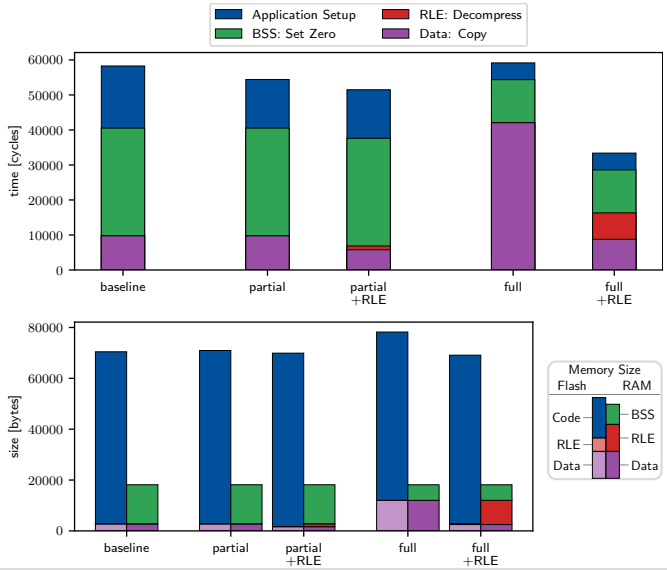
Evaluation – Queues



Evaluation – LibrePilot



Evaluation – GPSLogger



References I

Yulei Sui and Jingling Xue. “SVF: Interprocedural Static Value-Flow Analysis in LLVM”. In: *Proceedings of the 25th International Conference on Compiler Construction*. CC 2016. Barcelona, Spain: Association for Computing Machinery, 2016, pp. 265–266. ISBN: 9781450342414. DOI: 10.1145/2892208.2892235. URL: https://doi.org/10.1145/2892208.2892235.