User-Guided Device Driver Synthesis

Leonid Ryzhyk Adam Walker John Keys

Alexander Legg Arun Raghunath Michael Stumm

Mona Vij

The joys of driver development

- Drivers are hard to write
- … and even harder to debug
- They often delay product delivery
- … and are the most common source of OS failures

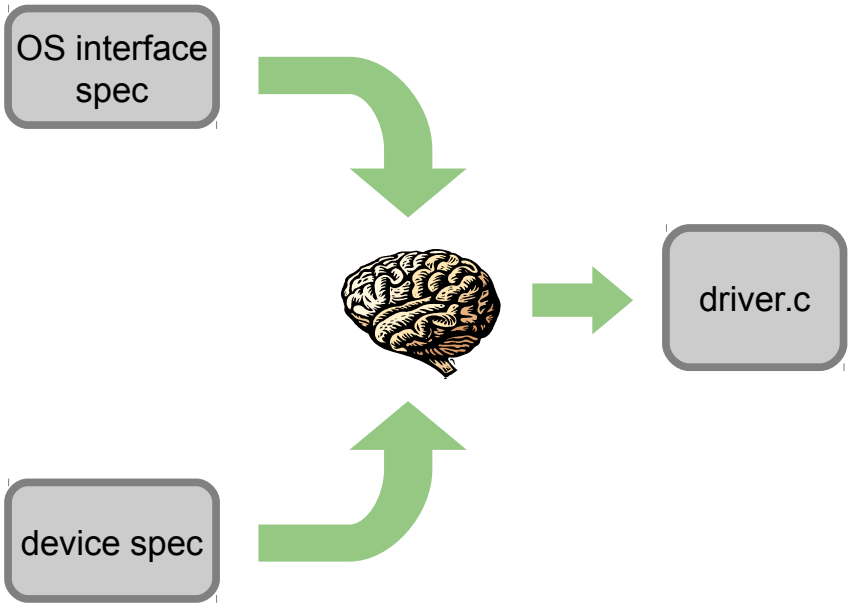
The joys of driver development

- Drivers are hard to write
- … and even harder to debug
- They often delay product delivery
- … and are the most common source of OS failures

Funded by a research grant from Intel

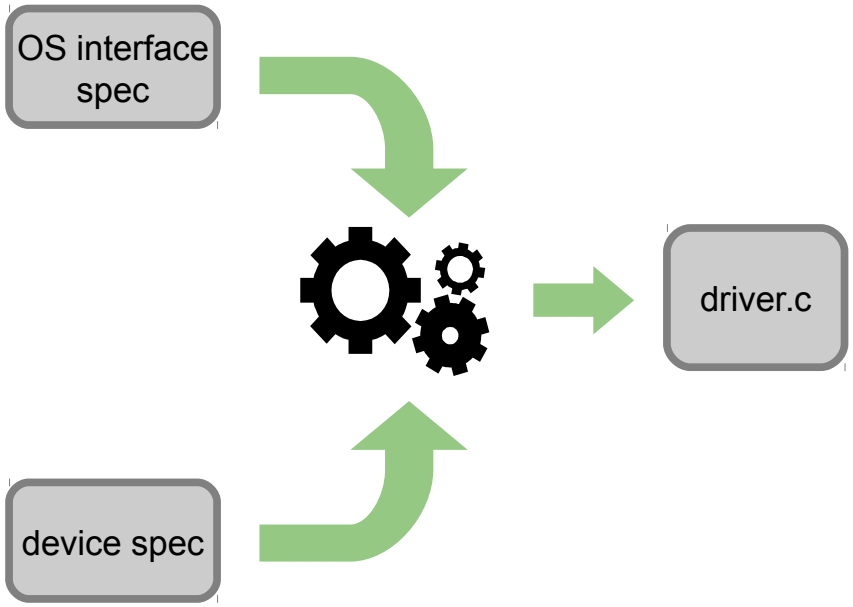
Observation

- Driver development is a mechanical task
- Can in principle be automated



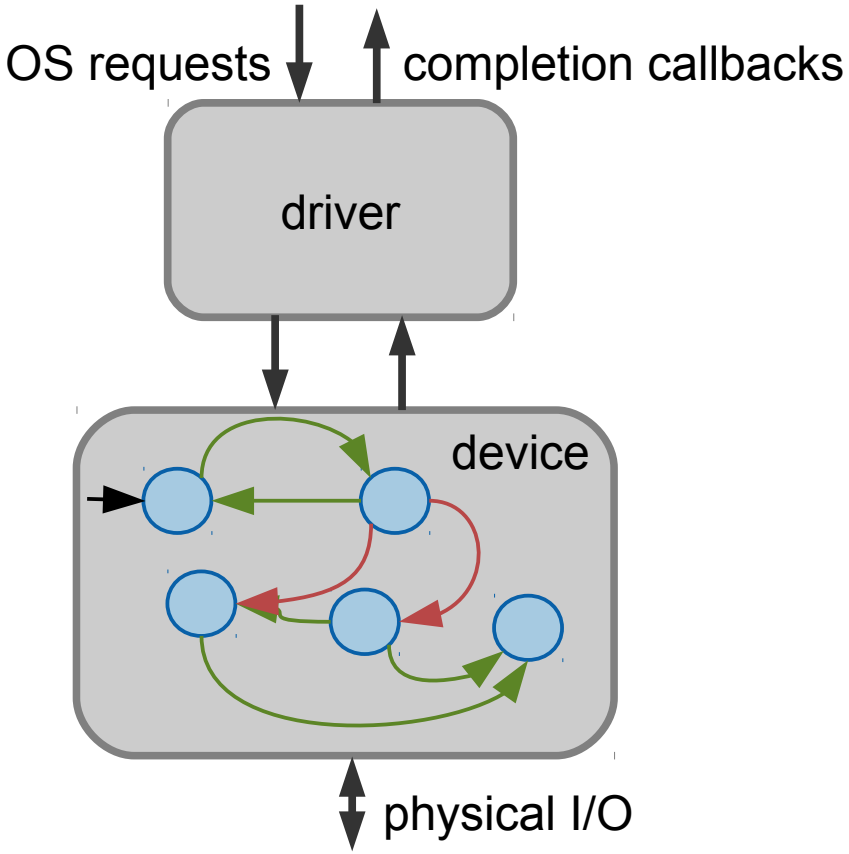
Observation

- Driver development is a mechanical task
- Can in principle be automated



Driver Synthesis as a Game

- Driver synthesis can be formalised as a two-player game: *driver* vs *(device + OS)*



Motivation

Motivation

Addresses an important problem

Motivation

Addresses an important problem

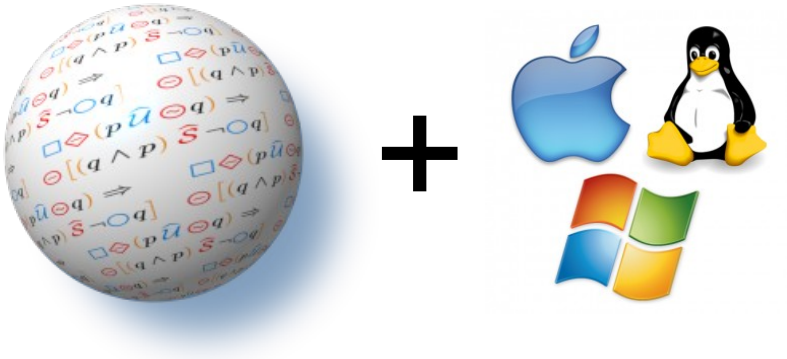
A simple, neat idea

Motivation

Addresses an important problem

A simple, neat idea

One of few applications of FM to OS (beyond verification)

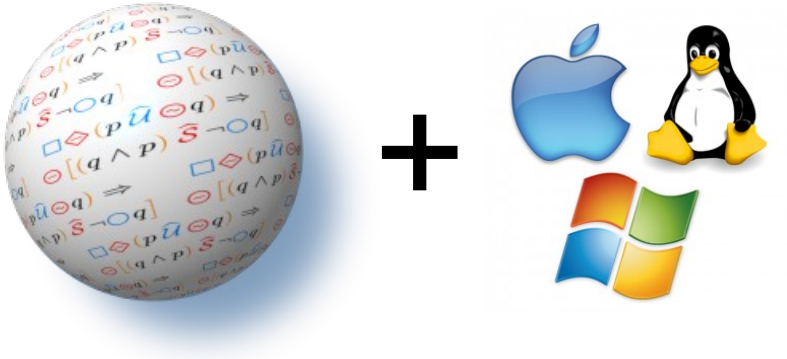


Motivation

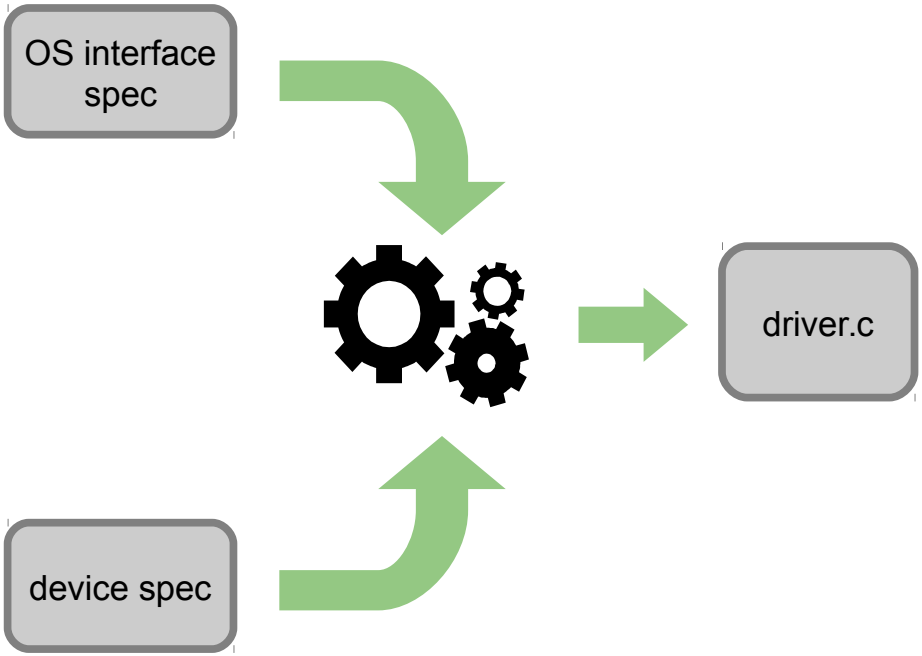
Addresses an important problem

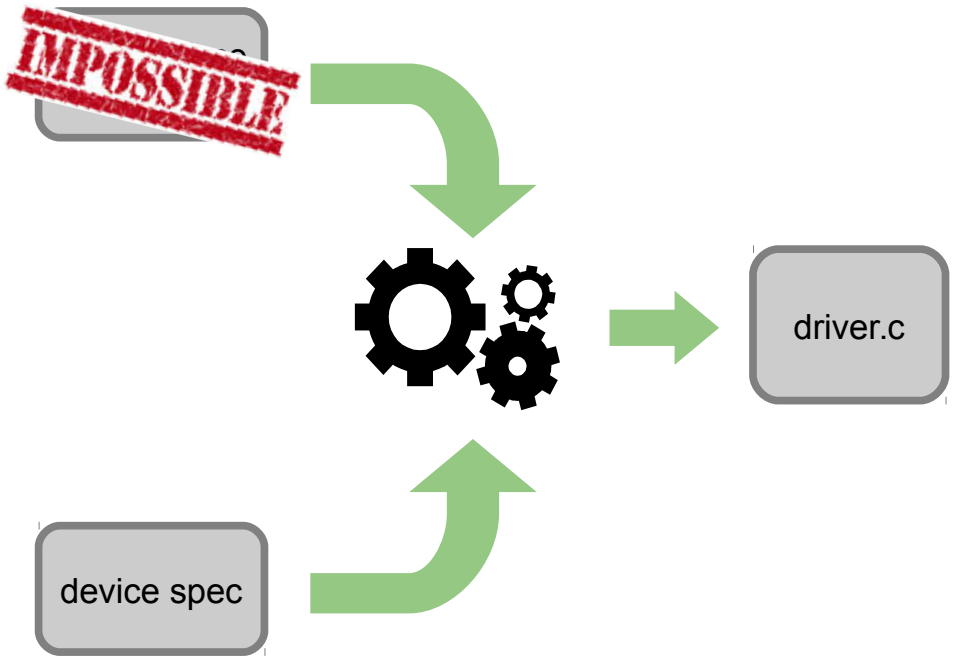
A simple, neat idea

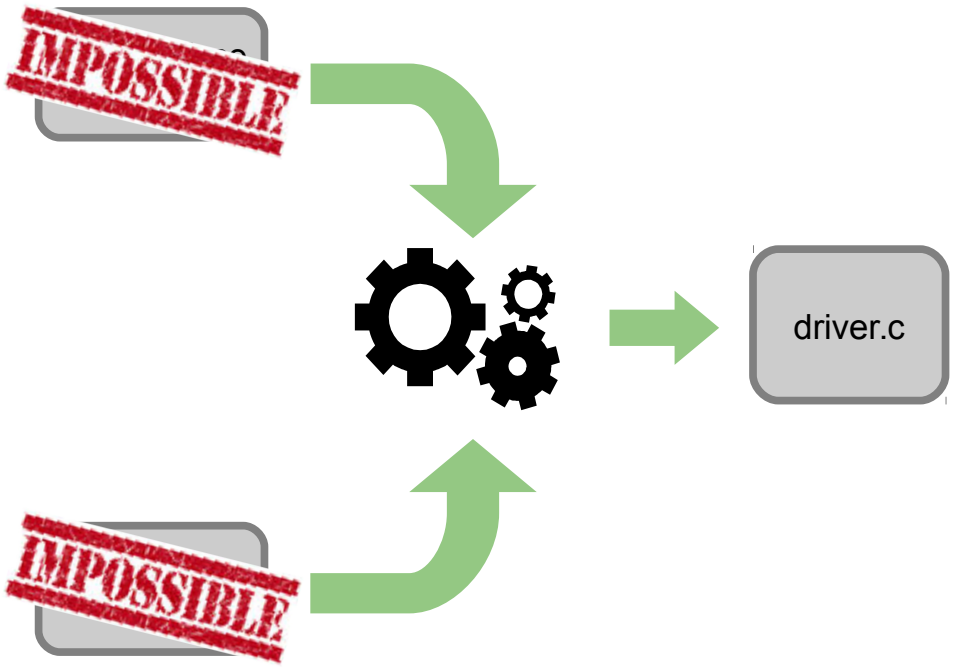
One of few applications of FM to OS (beyond verification)

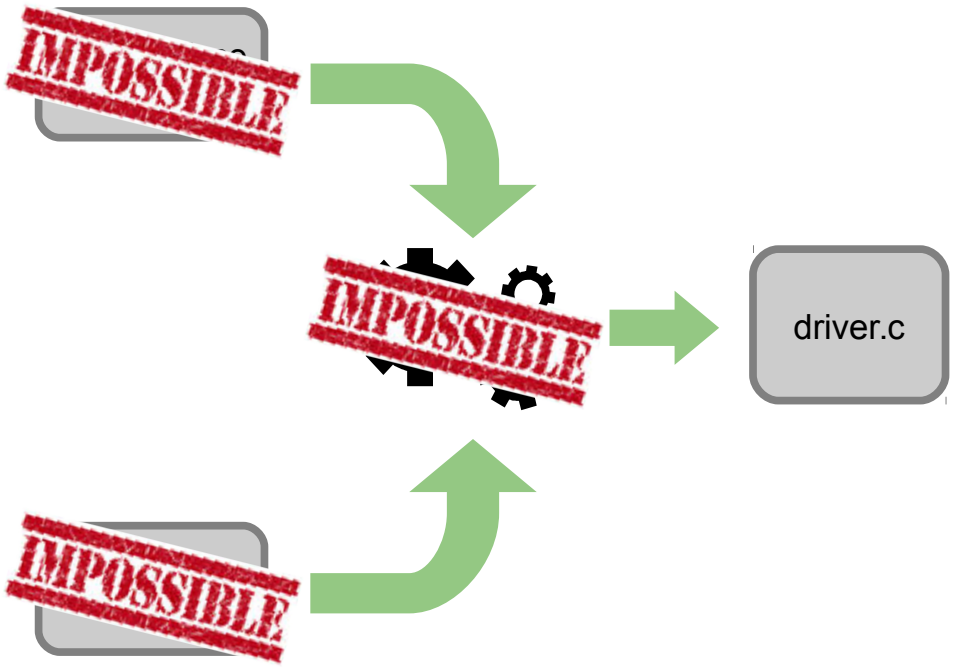


Considered impossible


OS interface spec
device spec
driver.c


IMPOSSIBLE
driver.c
device spec


IMPOSSIBLE
IMPOSSIBLE
driver.c


IMPOSSIBLE
IMPOSSIBLE
IMPOSSIBLE
driver.c

Driver Synthesis as a Game

Driver Synthesis as a Game

`request:set_time( 19:30:00)`

Driver Synthesis as a Game


```
request:set_time( 19:30:00)

write_hours( 19)
```

Driver Synthesis as a Game


```
request:set_time( 19:30:00)
write_hours( 19)
write_minutes(30)
```

Driver Synthesis as a Game


```
request:set_time( 19:30:00)
write_hours( 19)
write_minutes(30)
write_seconds(00)
```

Driver Synthesis as a Game


```
request:set_time( 19:30:00)
write_hours( 19)
TICK
```

Driver Synthesis as a Game


```
request:set_time( 19:30:00)
write_hours( 19)
TICK
```

Driver Synthesis as a Game

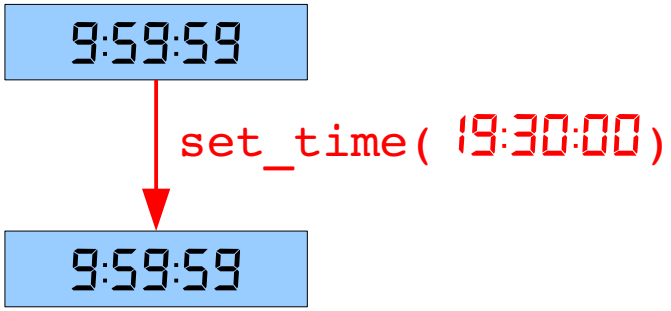

```
request:set_time( 19:30:00)
write_hours( 19)
TICK
write_minutes(30)
write_seconds(00)
```

Driver Synthesis as a Game

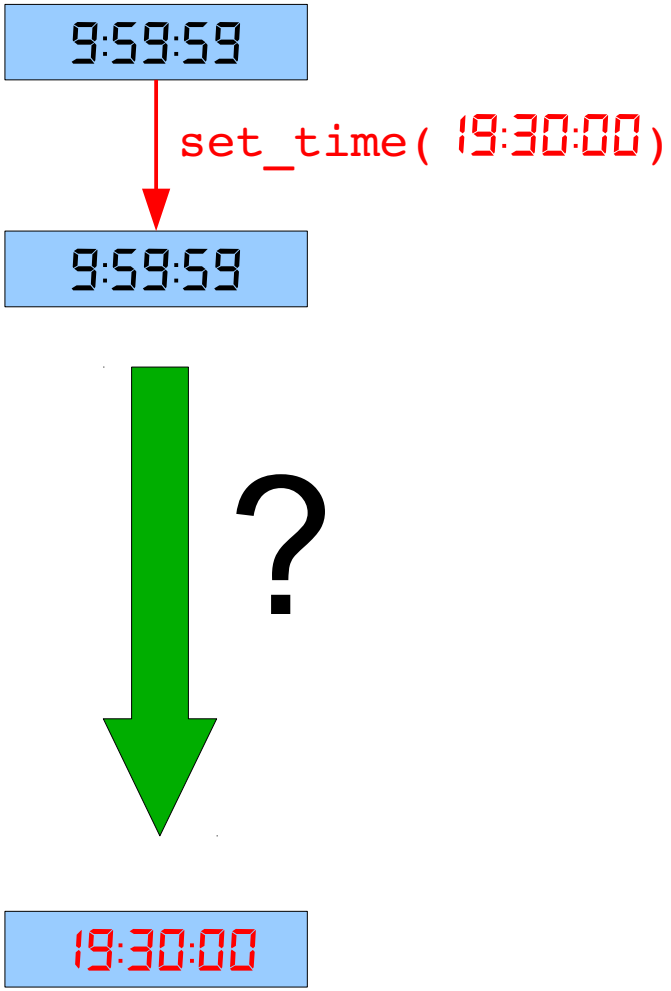

```
request:set_time( 19:30:00)
STOP
write_hours( 19)
write_minutes(30)
write_seconds(00)
START
```

Driver Synthesis as a Game

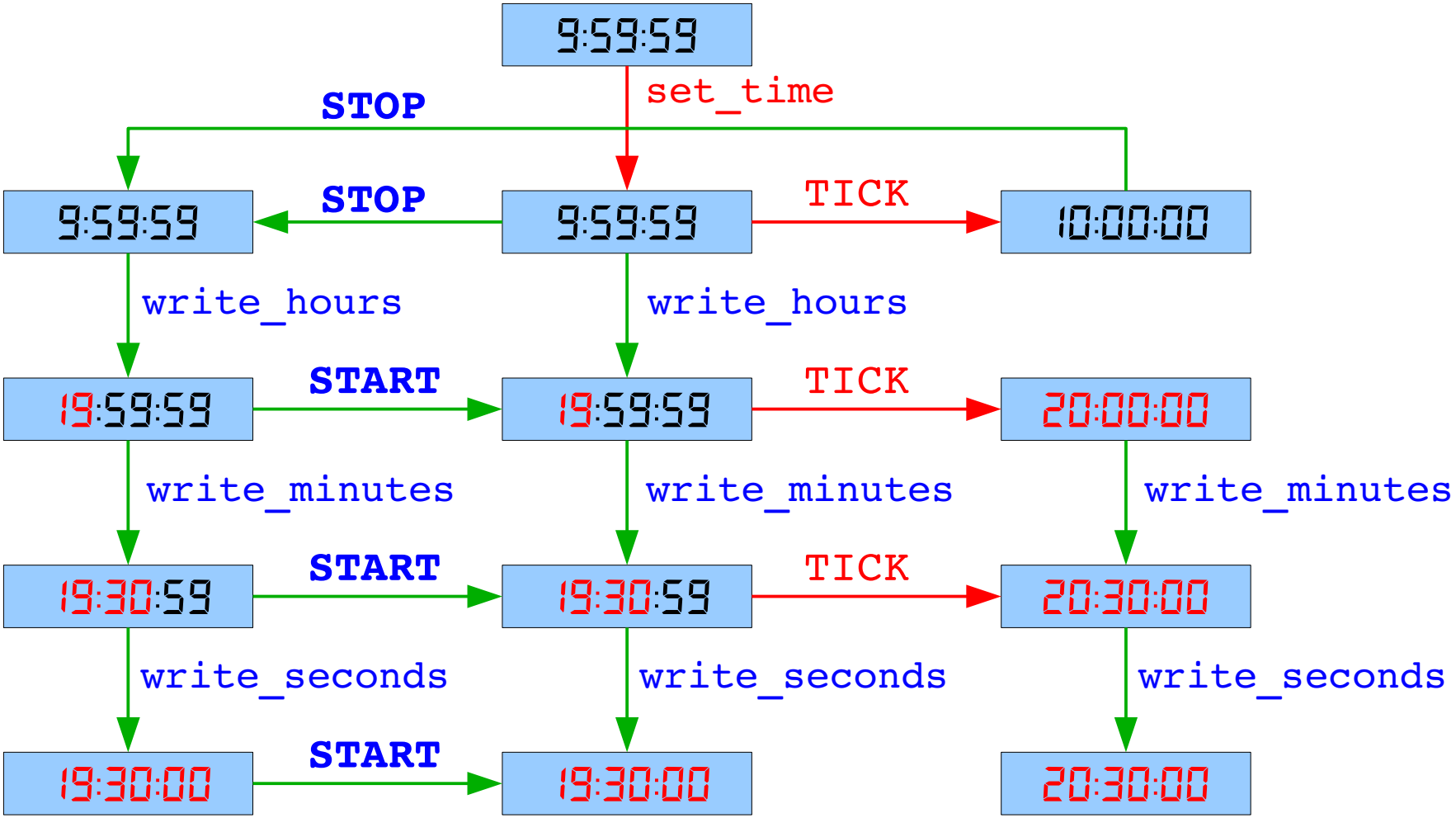
Driver Synthesis as a Game



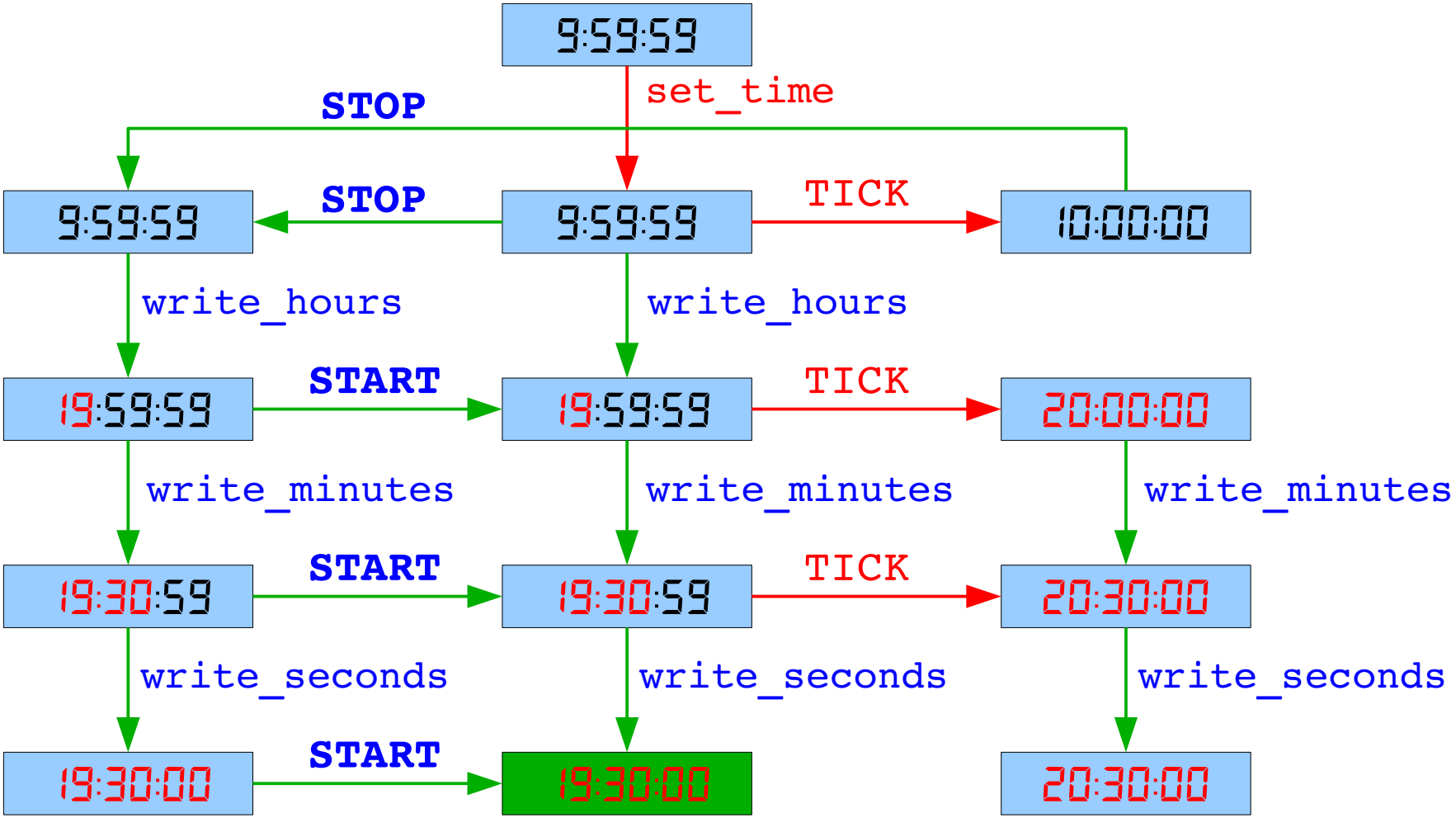
Driver Synthesis as a Game



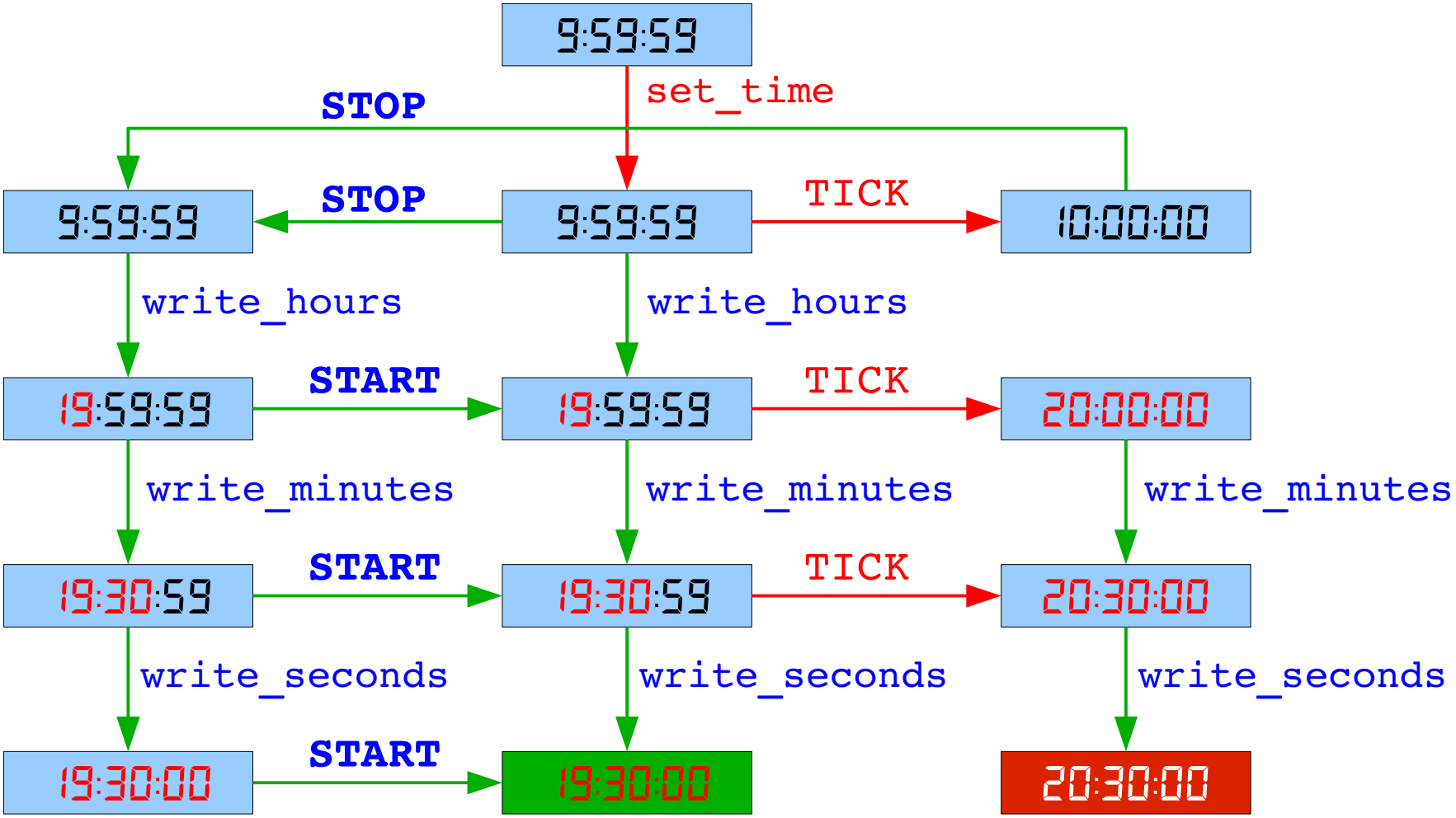
Driver Synthesis as a Game



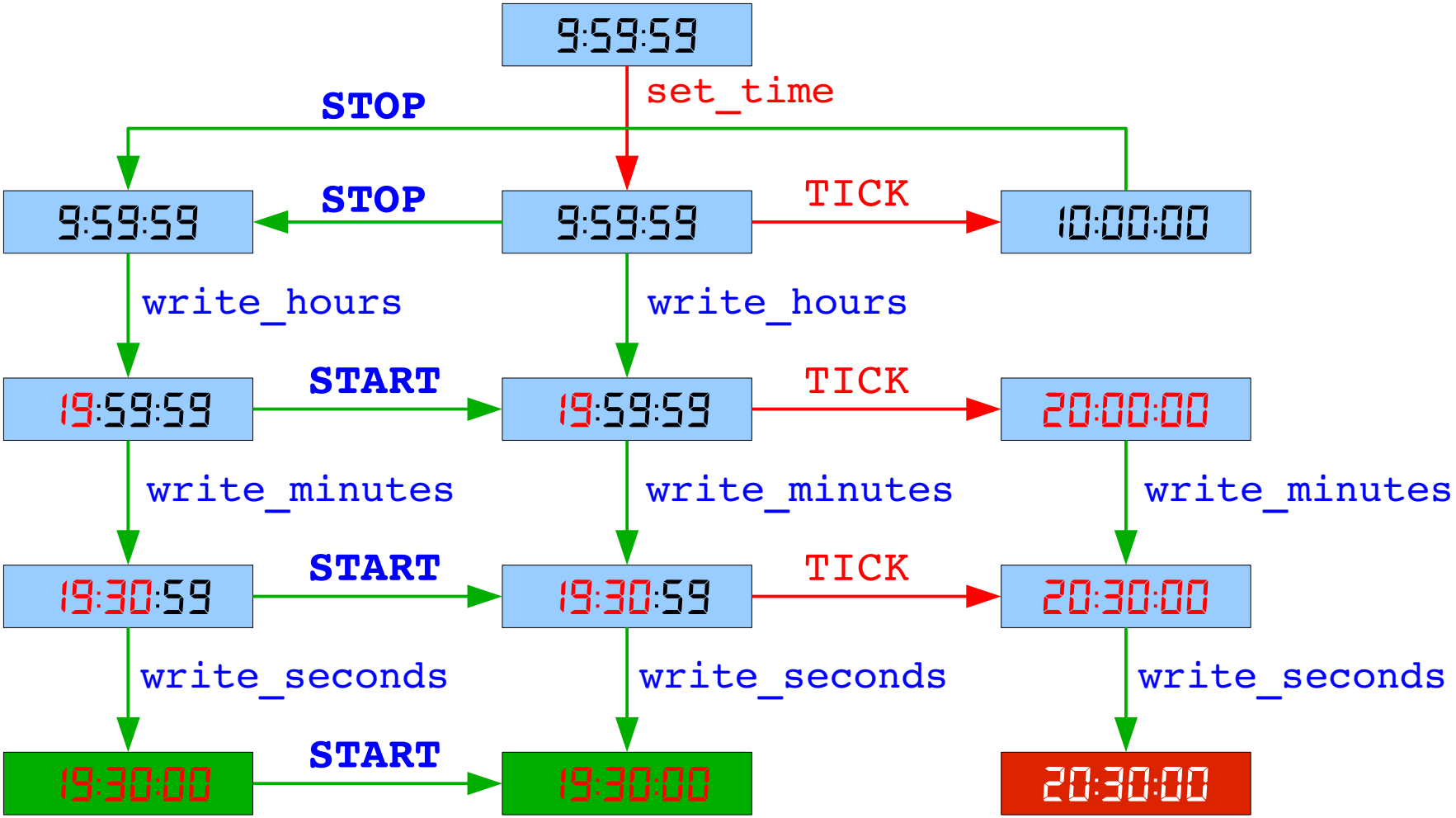
Driver Synthesis as a Game



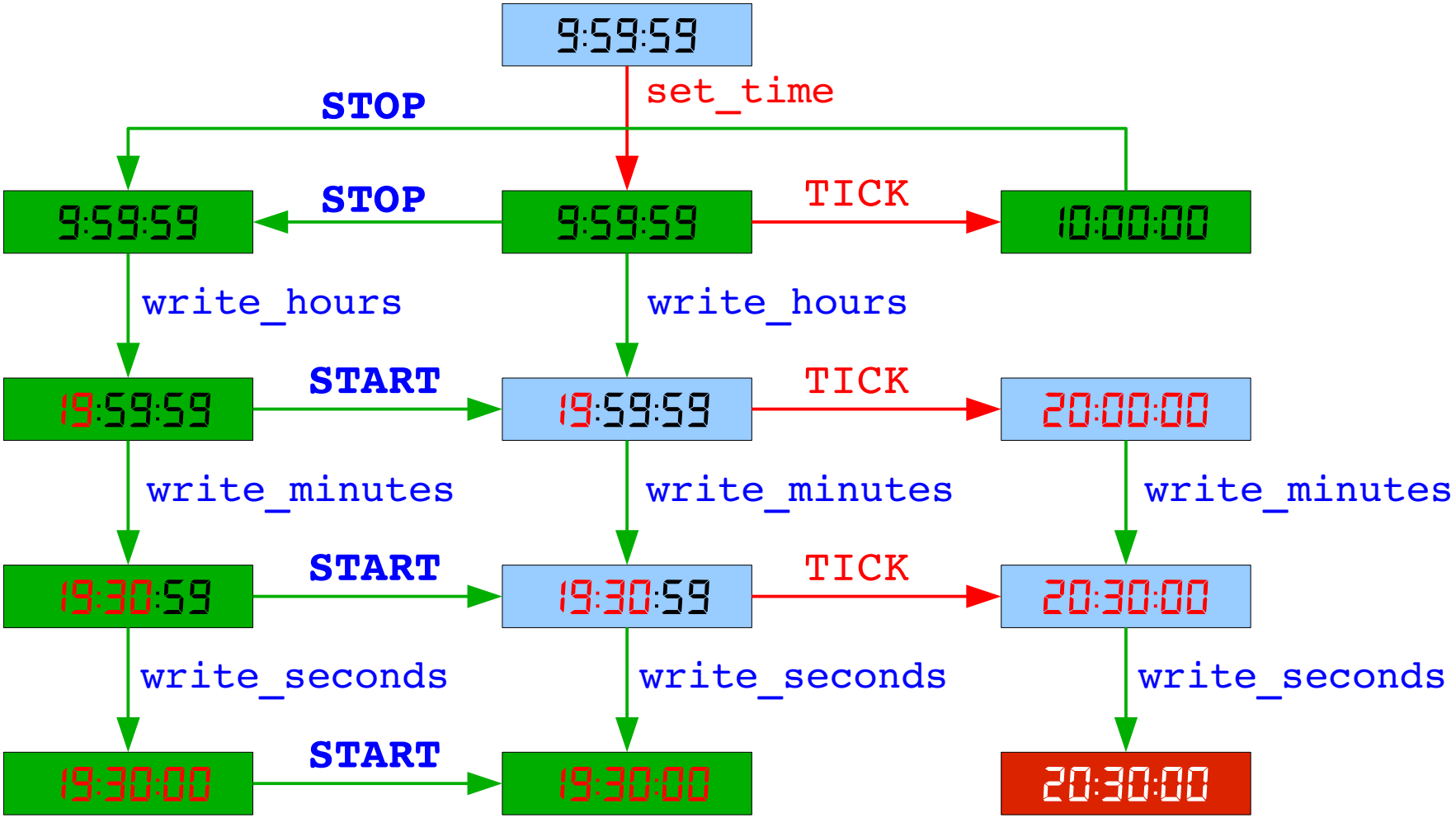
Driver Synthesis as a Game



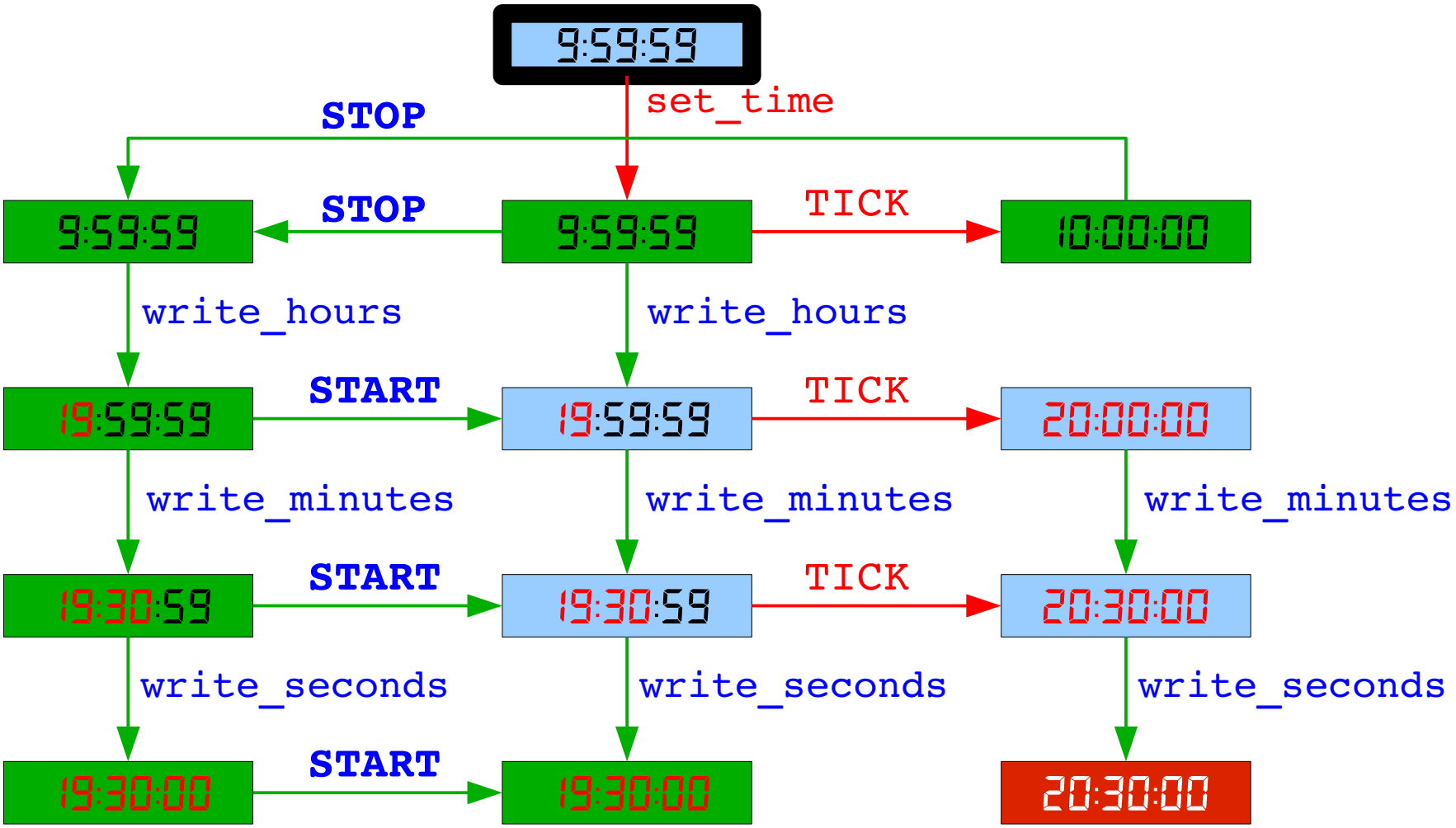
Driver Synthesis as a Game



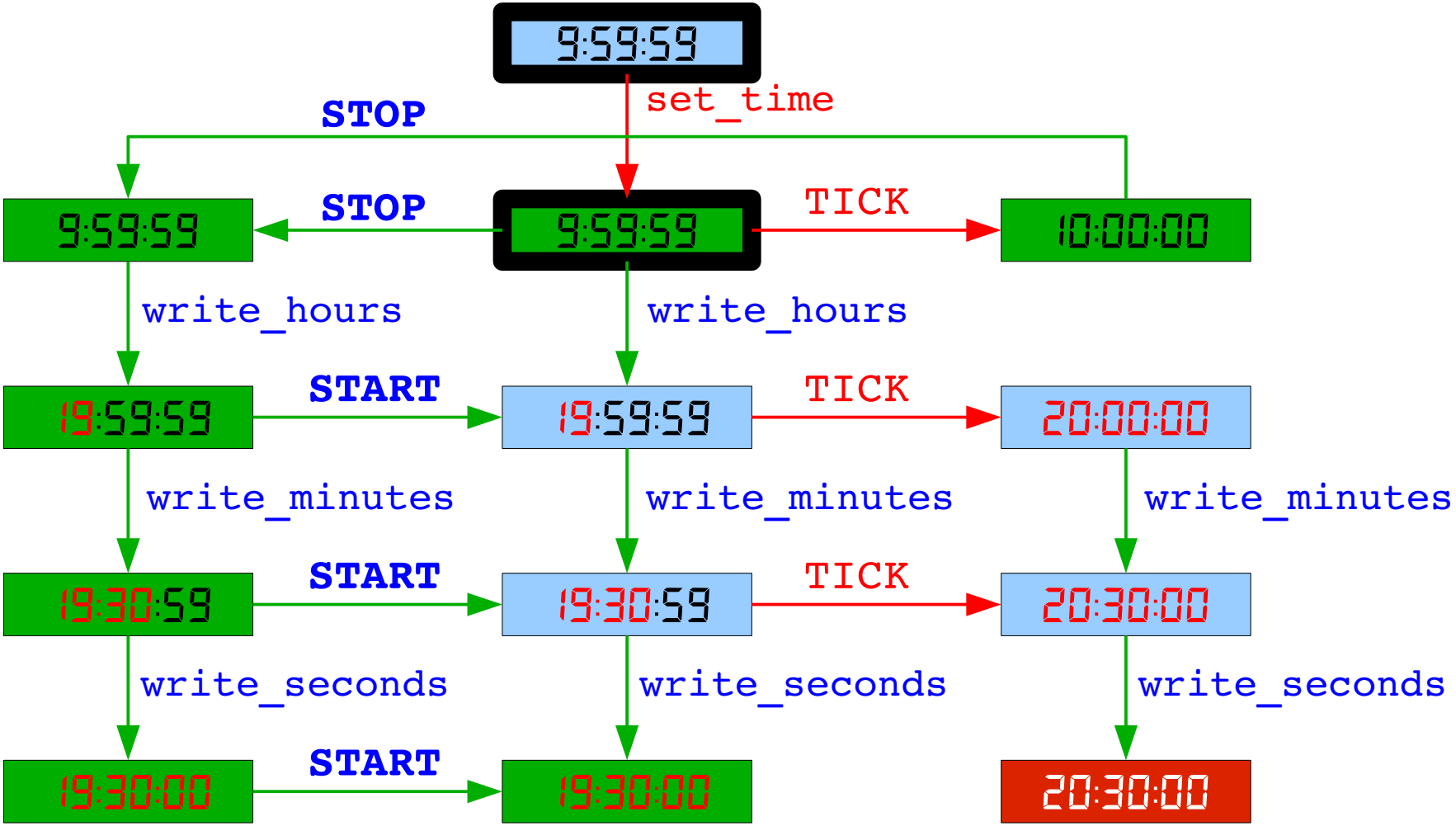
Driver Synthesis as a Game



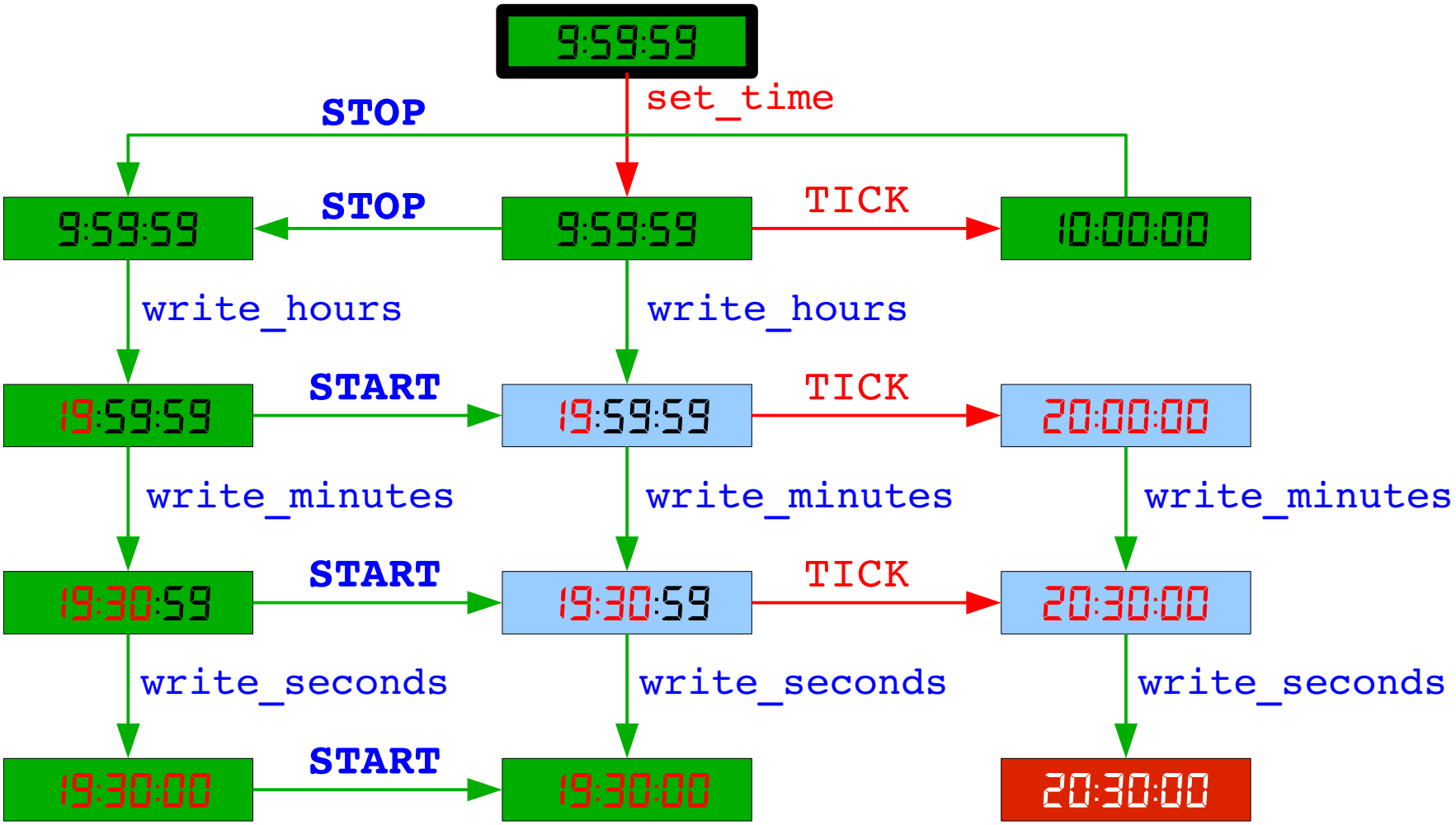
Driver Synthesis as a Game



Driver Synthesis as a Game



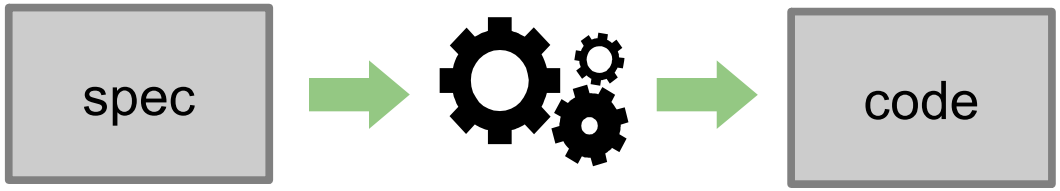
Driver Synthesis as a Game



Termite Tool Demo

Push-Button Synthesis (SOSP'09)

- **In theory:**



correct spec => correct implementation

- **In practice:** (based on our experience) taking control away from the developer is not a good idea

Push-Button Synthesis (SOSP'09)

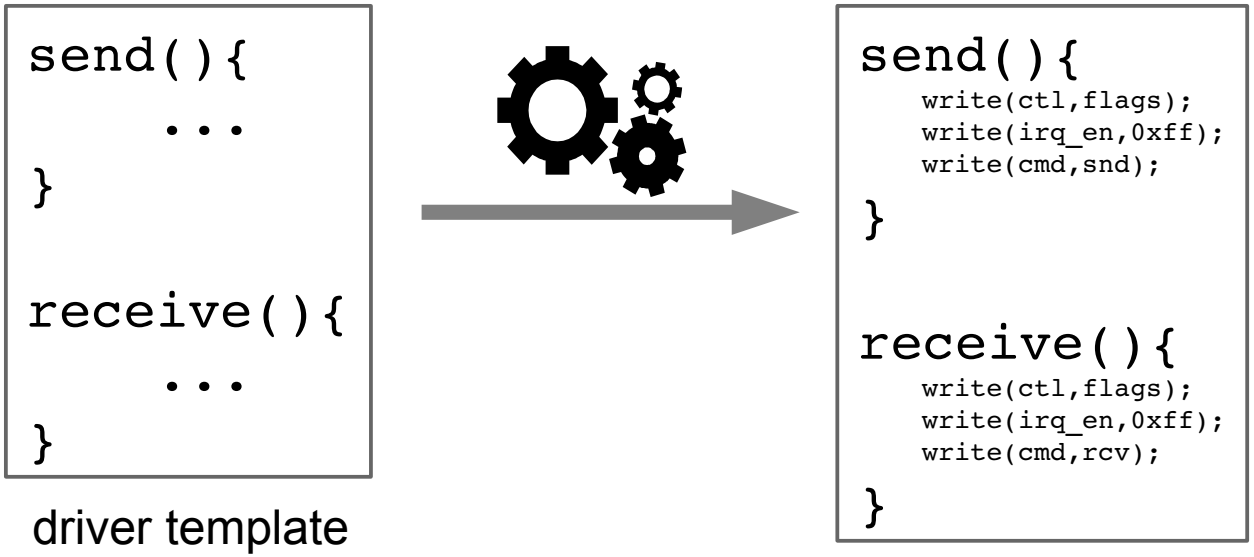
- Choosing a preferred implementation method is hard (e.g., polling vs interrupts)
- Non-functional properties (power, performance, timing, etc.) are hard to enforce
- Achieving “nice” code structure is hard

User-Guided Synthesis

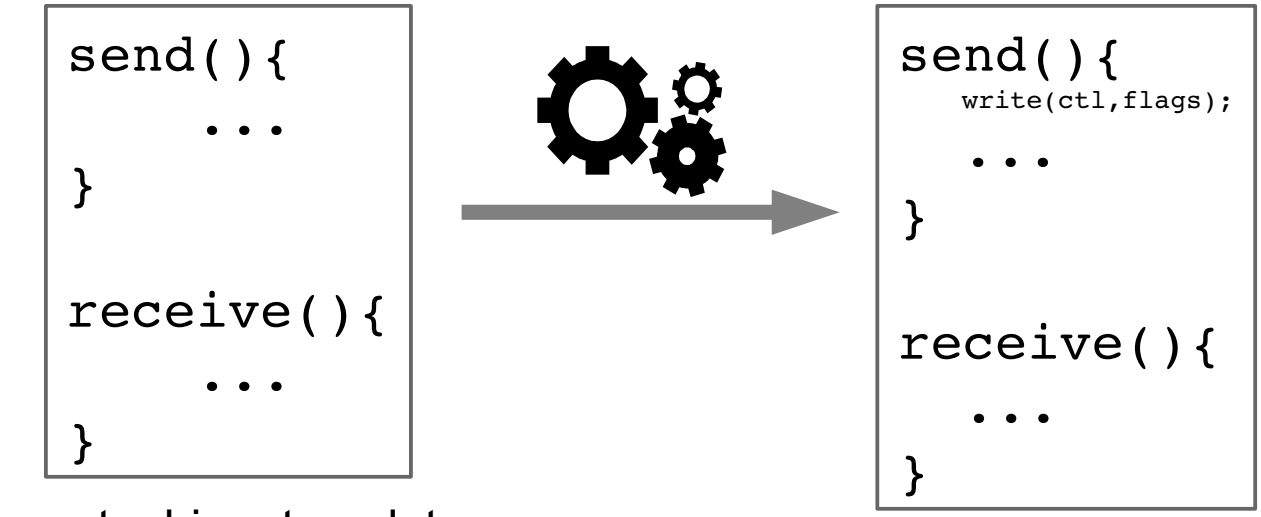
- The user is in control
  - can write arbitrary manual code or …
  - arbitrarily alter automatically generated code
- Synthesiser works as smart auto-complete
  - can generate a statement, a function, or even the whole driver (on demand)
  - never alters user code
  - completes synthesised+manual code to a correct implementation when possible
- The tool enforces correctness

Demo (continued)

Guided Synthesis
Scenario 1: Fully Automatic Synthesis



Guided Synthesis
Scenario 2: Hybrid Approach



empty driver template

partially
synthesised driver

Guided Synthesis
Scenario 2: Hybrid Approach

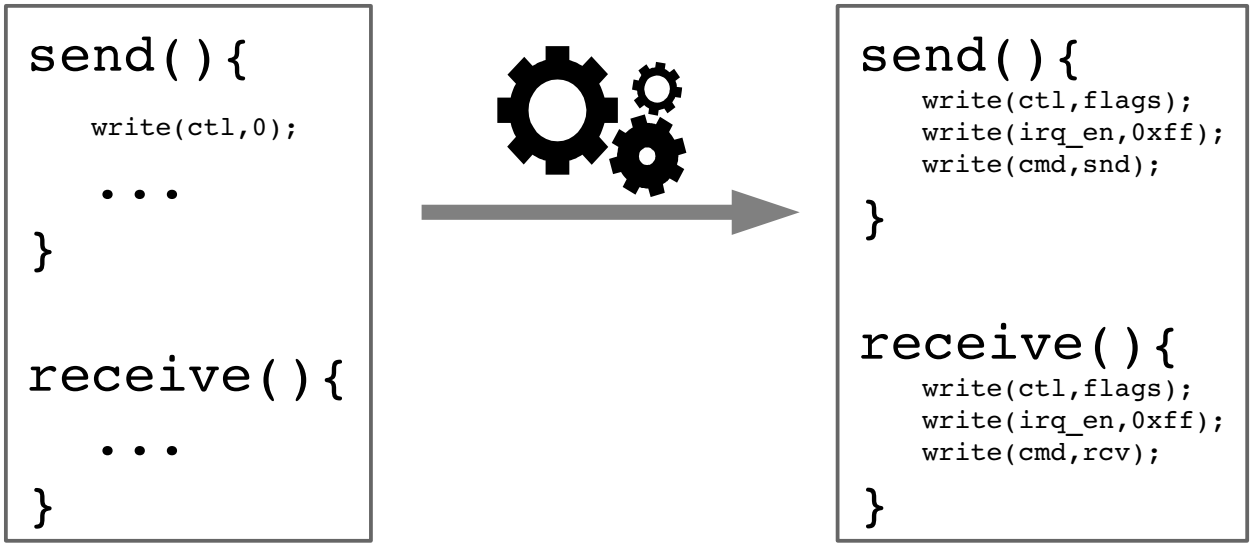
```
send(){
    write(ctl,0);
    ...
}

receive(){
    ...
}
```

modified driver template

```
send(){
    write(ctl,flags);
    ...
}

receive(){
    ...
}
```

partially
synthesised driver

Guided Synthesis
Scenario 2: Hybrid Approach



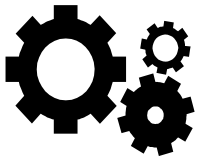
modified driver template

synthesised driver

Guided Synthesis
Scenario 3: Verification

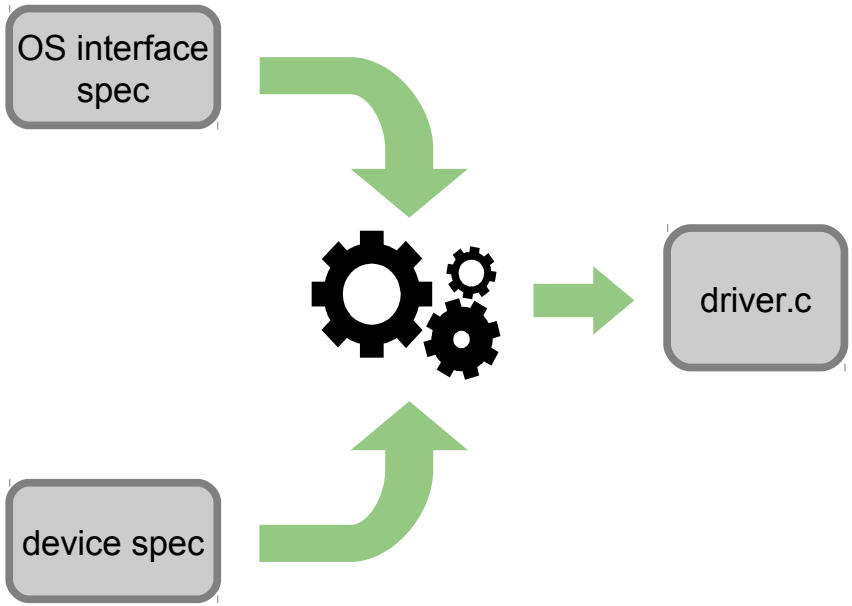
```
send(){
    write(ctl,flags);
    write(irq_en,0xff);
    write(cmd,snd);
}

receive(){
    write(ctl,flags);
    write(irq_en,0xff);
    write(cmd,rcv);
}
```

manually developed driver

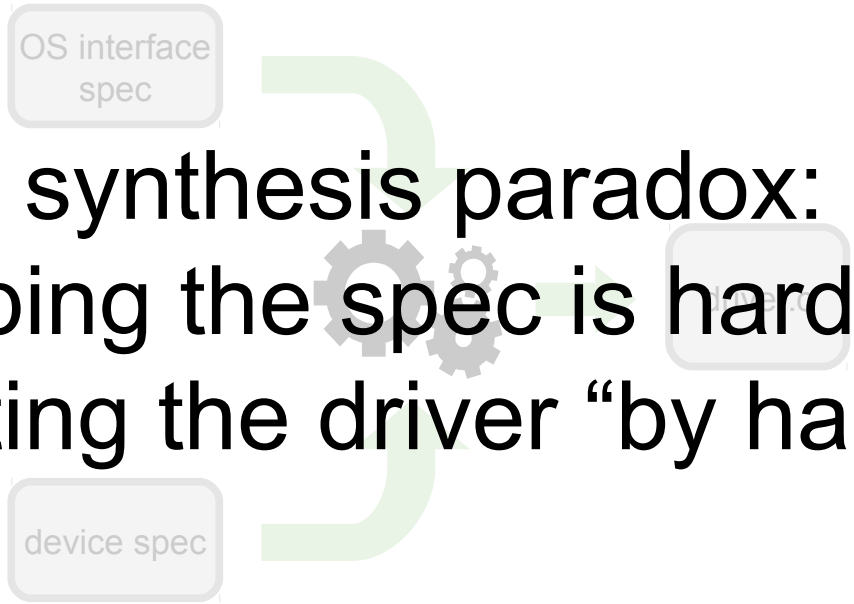


√

Obtaining Specs for Driver Synthesis

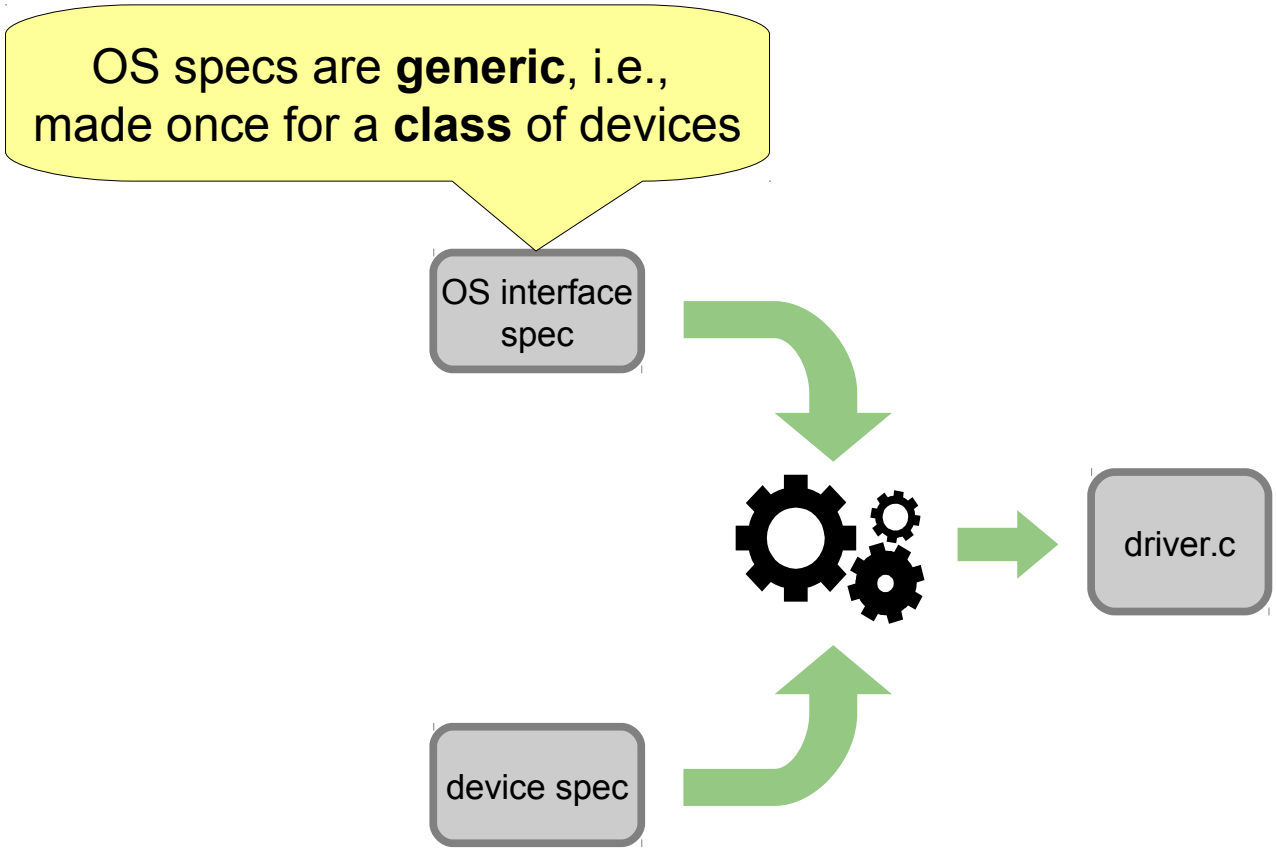


Obtaining Specs for Driver Synthesis

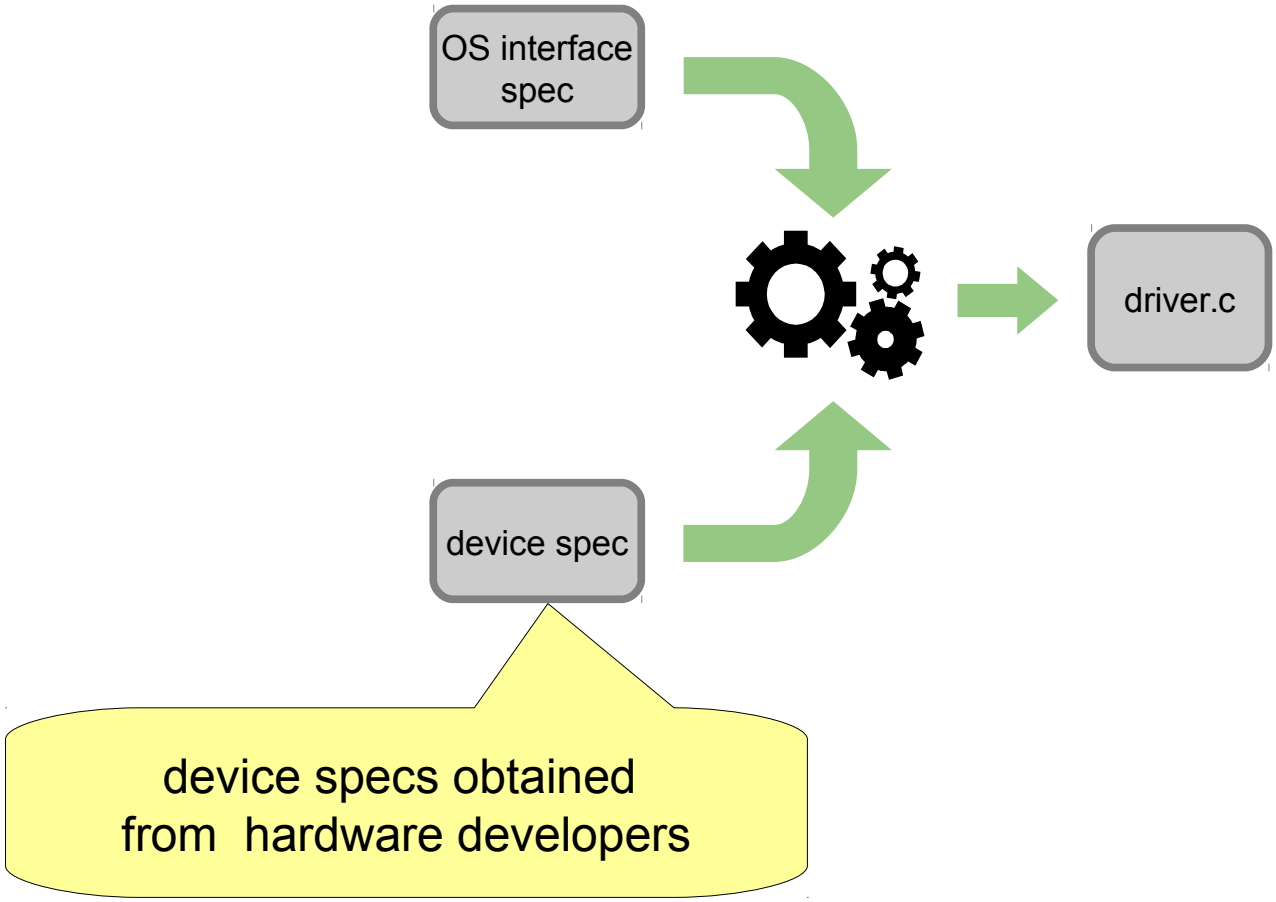


The synthesis paradox: developing the spec is harder than writing the driver “by hand”.

Obtaining Specs for Driver Synthesis



Obtaining Specs for Driver Synthesis



Synthesised Drivers

Device	Synthesis time (s)	locs
Real-time clock	74	56
IDE	71	94
STM32F103RB UART	309	74
exynos 5 UART	177	35
STM32F103RB I2C	39	119
exynos 5 I2C	40	77
webcam	190	113
SPI	15	27

Synthesised Drivers

Device	Synthesis time (s)	locs
Real-time clock	74	56
IDE	71	94
STM32F103RB UART	309	74
exynos 5 UART	177	35
STM32F103RB I2C	39	119
exynos 5 I2C	40	77
webcam	190	113
SPI	15	27

Scope and Limitations

- Focus on synthesising device control logic
  - Resource allocation, binding to OS interfaces, etc., must be written manually or synthesised using different techniques
- Sequential synthesis
  - Synchronisation synthesis as a separate step (jointly with CU Boulder and IST Austria)
- No DMA support
  - WiP

Summary

- Termite automates tedious driver development
- The user has full control over the source code, but Termite enforces correctness

`https://github.com/termite2`

`http://termite2.org`

Summary

- Termite automates tedious driver development
- The user has full control over the source code, but Termite enforces correctness
- Driver synthesis is less impossible than previously believed

`https://github.com/termite2`

`http://termite2.org`