

TDD For Embedded Systems... All The Way Down To The Hardware

**Neil Johnson
XtremeEDA
njohnson@xtreme-eda.com
@nosnhojn**

What Do I Mean By Hardware

- ASIC
 - Application Specific Integrated Circuit
 - Static structure
 - Digital or mixed signal
 - High NRE/Low cost
- FPGA
 - Field Programmable Gate Array
 - Reprogrammable structure
 - Primarily digital
 - No NRE/High cost
- SoC
 - Either of the above + embedded processor(s) + software



SoC Development Basics

- Typical SoC design flow

- Specification
- Design
- Verification
- Physical design
- Fabrication
- Validation
- Integration

Pre-silicon

Production

Software

Documentation

Code

“Stuff”

Chip

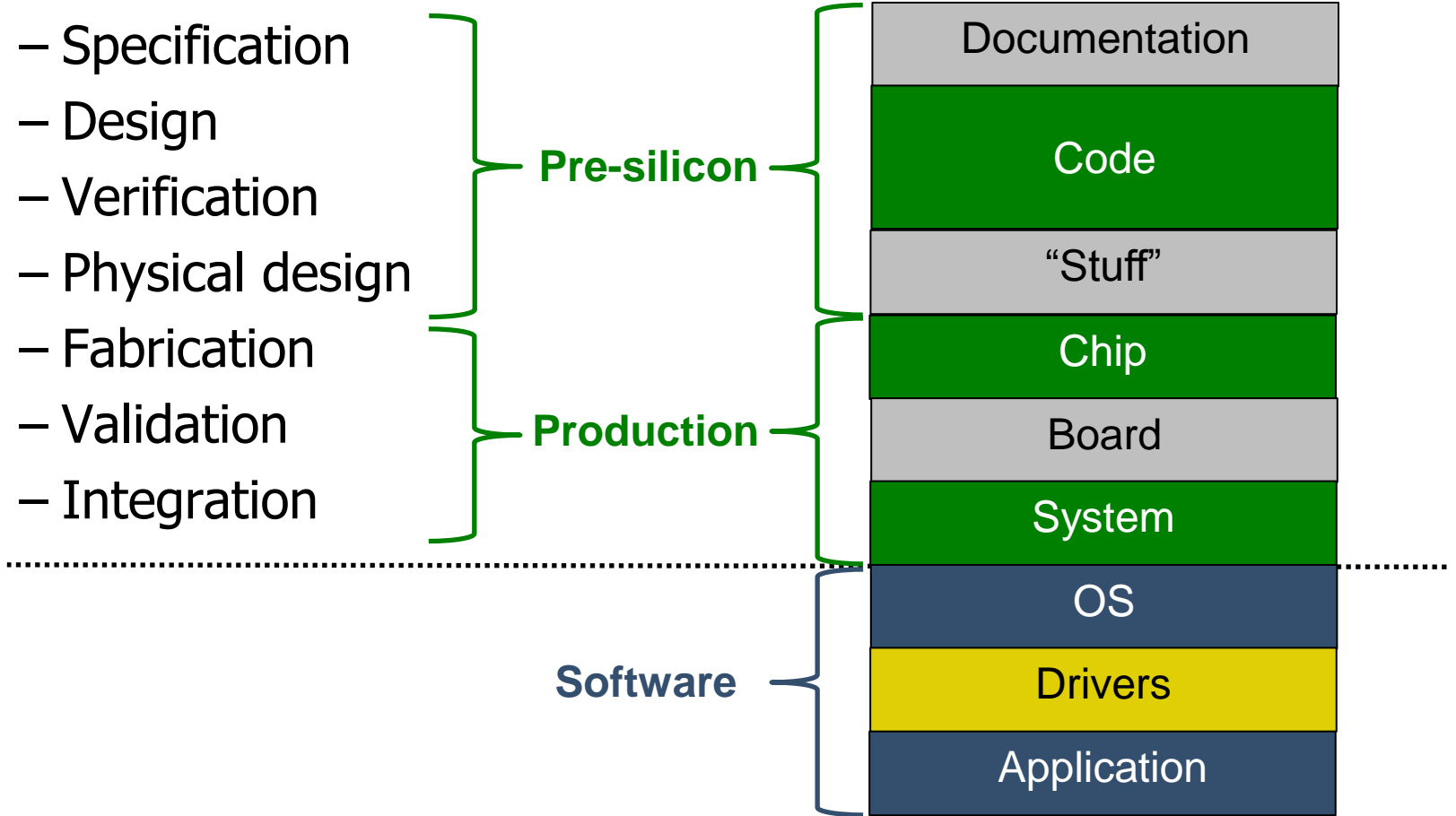
Board

System

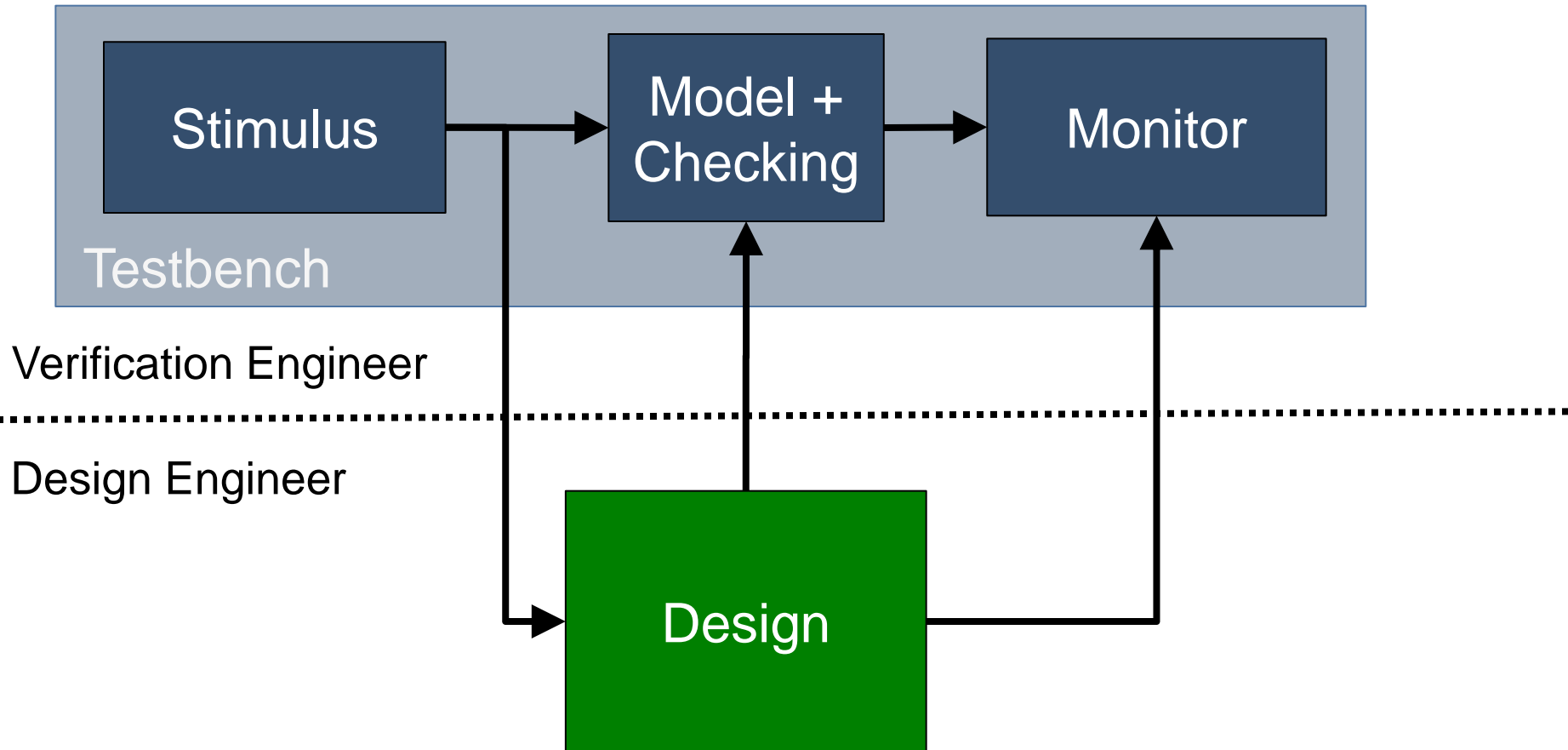
OS

Drivers

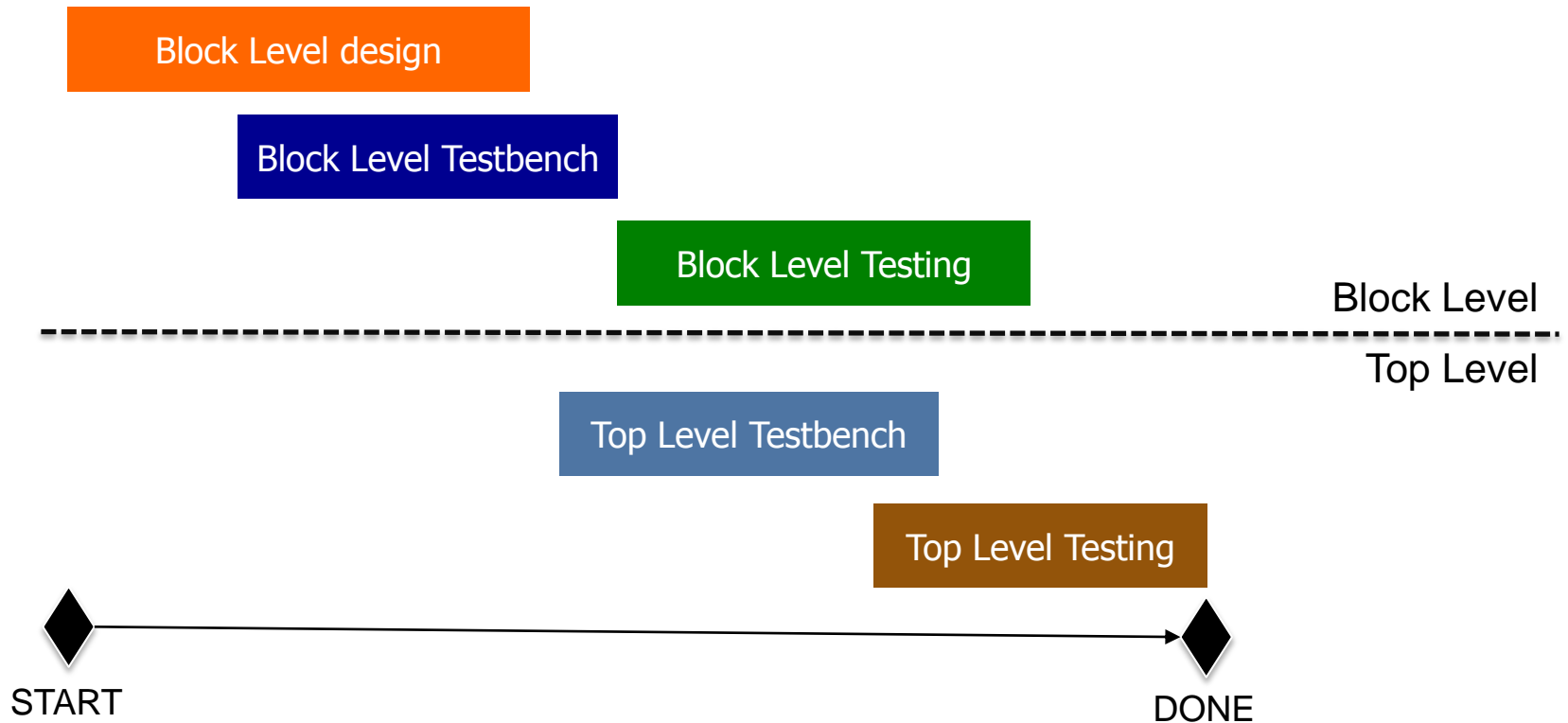
Application



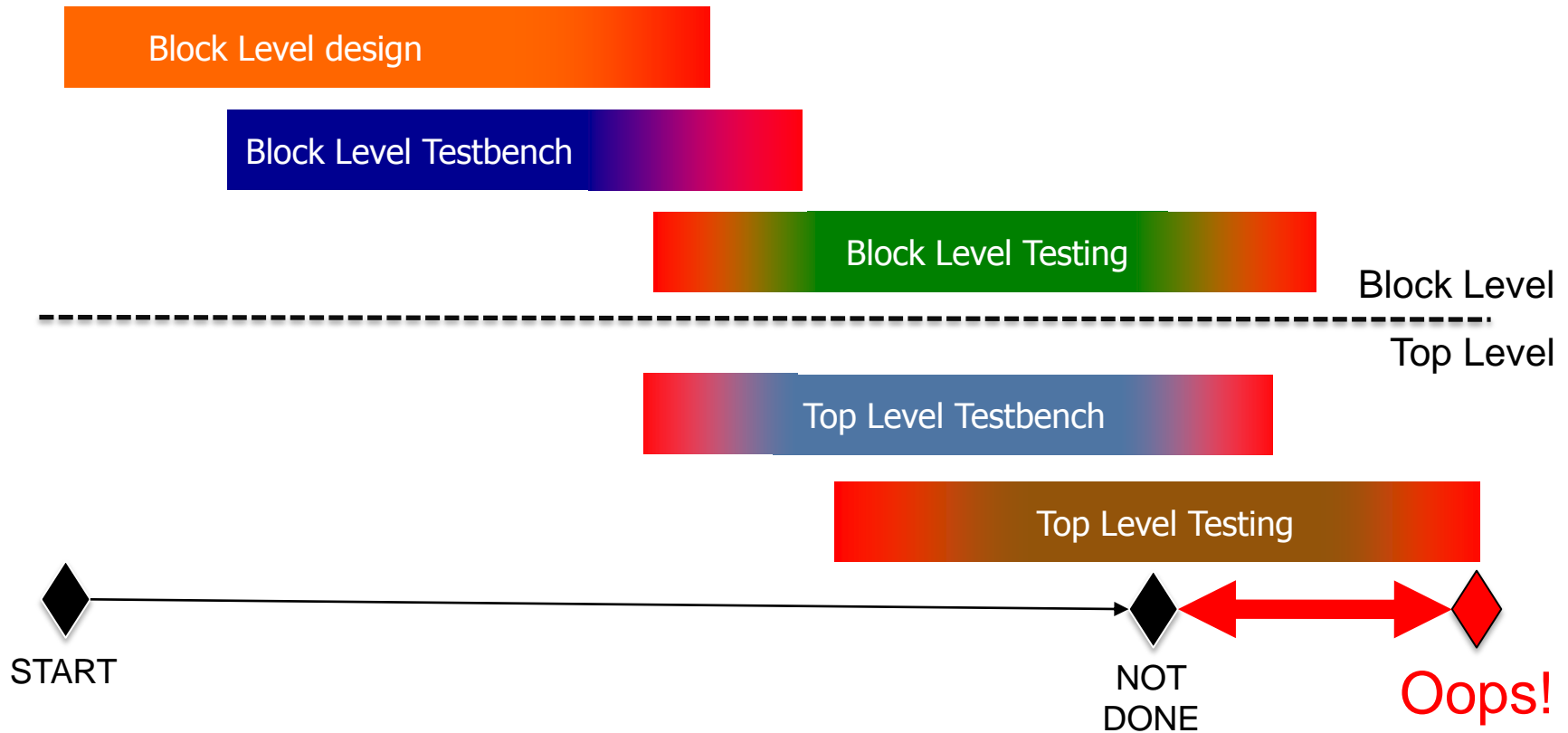
Design and Verification



Design and Verification – Planned?



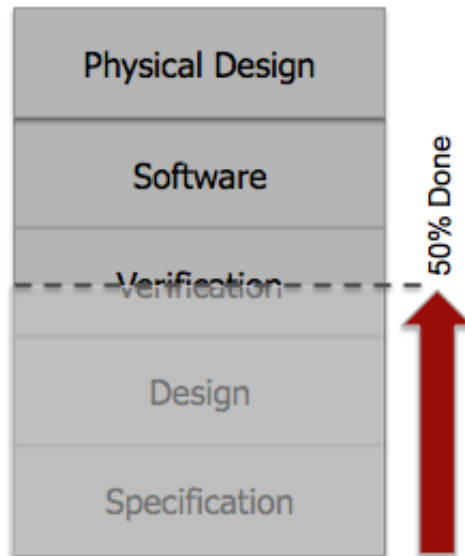
Design and Verification – Actual!



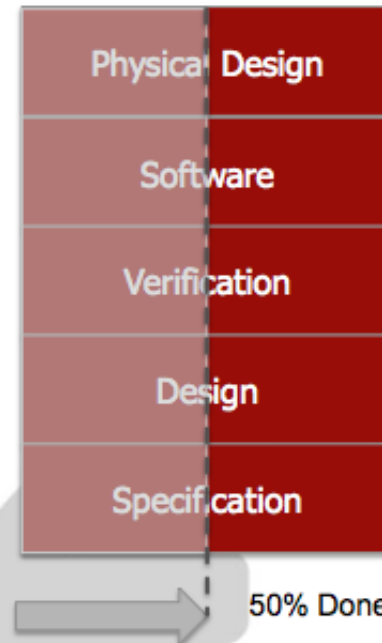
Working Software (Hardware)

Working Software (Hardware)

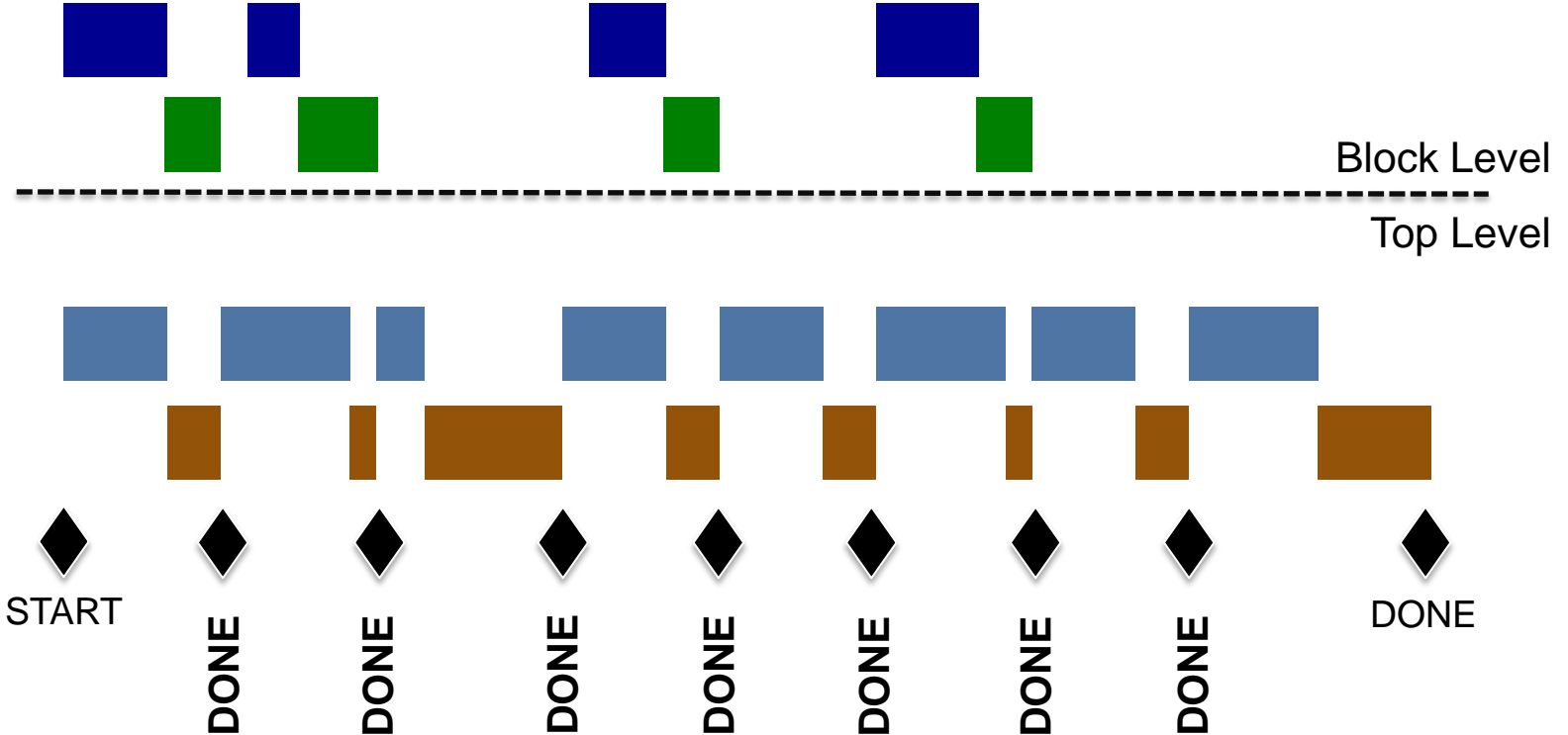
- Waterfall Model



- Agile model



Design and Verification – Incremental



The Perfect Place To Start

- Feature-of-the-week
 - “Tell your customer you’re going to give them something that works in a week”
 - Jonathan Rassmuson, *Agile in a Nutshell*, APLN April 2009
 - The Agilesoc Blog: Remote Development And The Feature-of-the-week

The feature-of-the-week is not Agile, it's frequent delivery... which is hard to argue against.

Feature Of The Week

- Client experience
 - Goal
 - Convince myself that incremental development is possible
 - Situation
 - Functional testing of a sub-system
 - Design was done, test harness was partially complete
 - Planning
 - 4 increments, 1 for each major feature
 - Detailed plan included 1-2 week sub-milestones
 - 2 increments planned in detail

Feature Of The Week

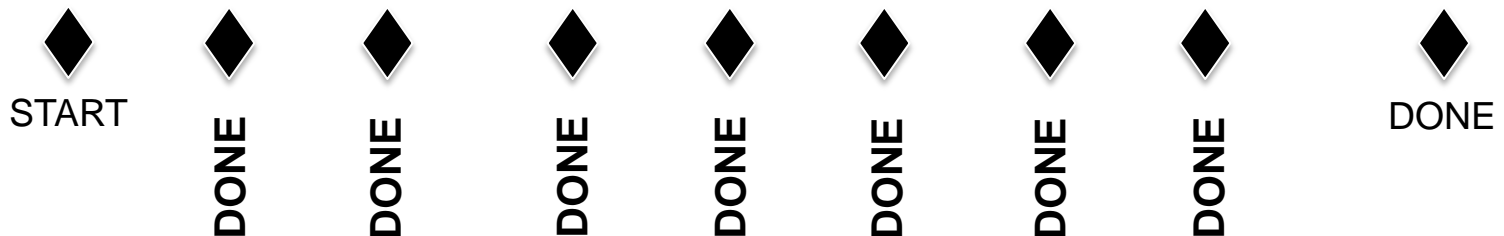
- Client experience - Highlights
 - Planning
 - The planning was different but no convincing was required
 - Increment 1
 - Uh oh... I've committed to delivering something in a week
 - First up: remove everything I don't need

Feature Of The Week

- Client experience – Highlights (con't)
 - Increment 2
 - I was focused and delivering on time
 - Functional milestones allowed me to react to new priorities
 - Increment 3
 - Functioning code was great for gaining confidence and/or being corrected
 - I wasn't so concerned with building infrastructure

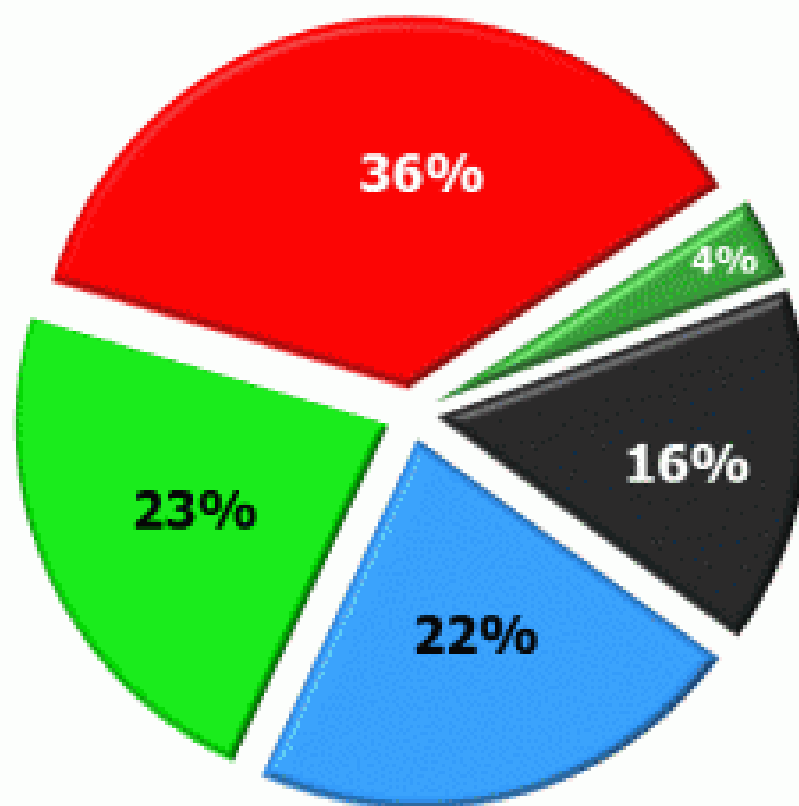
Feature Of The Week

- Client experience – Highlights (con't)
 - Increment 2
 - I was focused and delivering on time
 - Functional milestones allowed me to react to new priorities
 - Increment 3
 - Functioning code was great for gaining confidence and/or being corrected
 - I wasn't so concerned with building infrastructure
 - Summary
 - Convince myself incremental development can work



Effort and Results

Mean time Non-FPGA verification engineers spends in different tasks

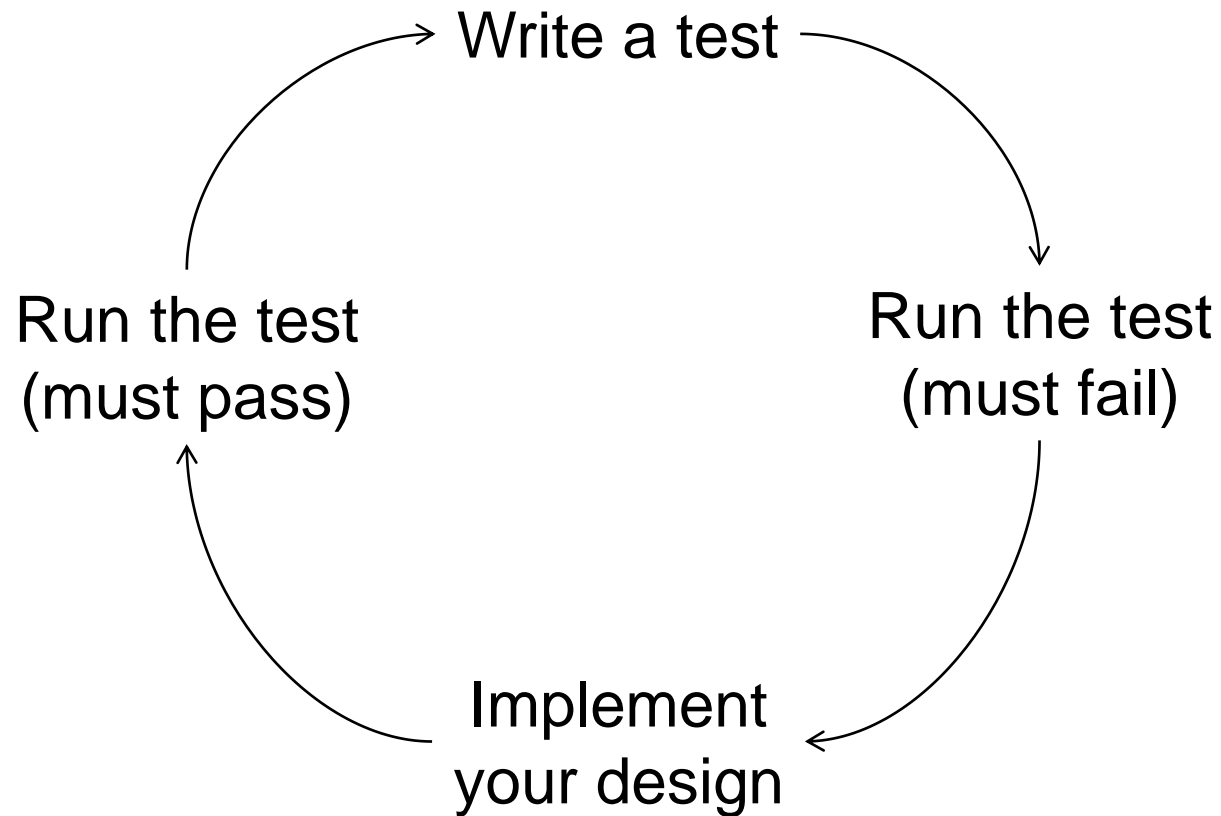


More time spent in debug than any other task!

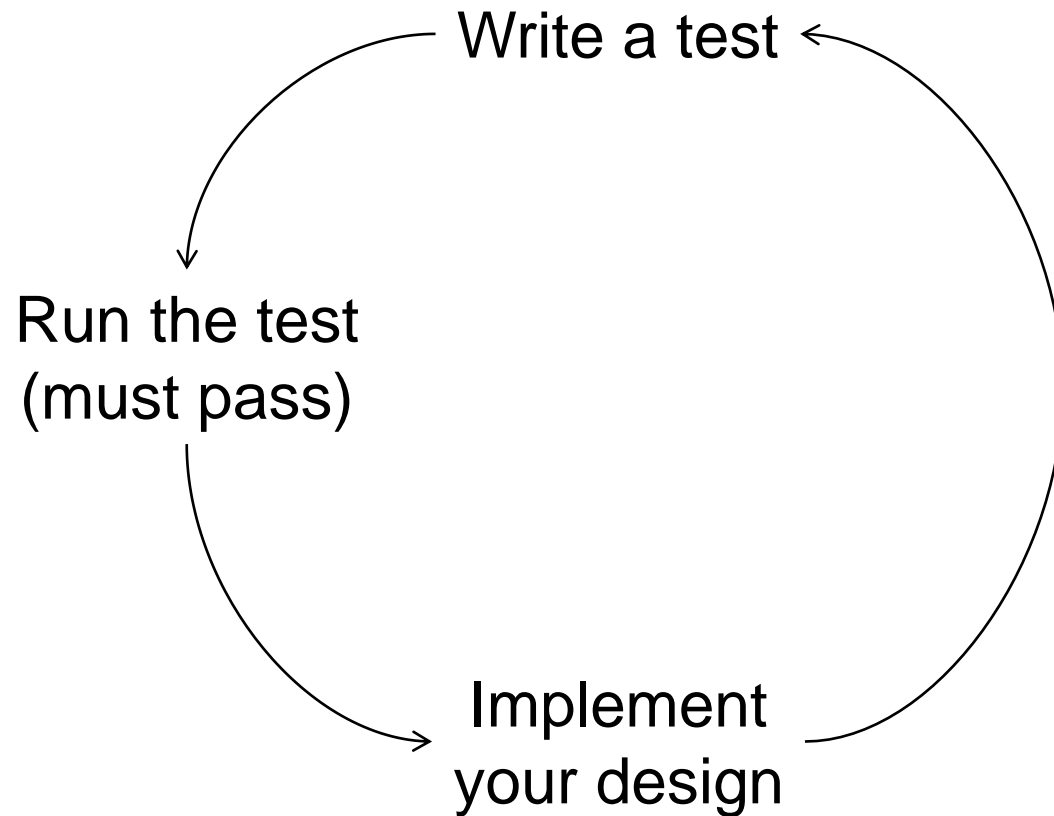
- Test Planning
- Testbench Development
- Creating and Running Test
- Debug
- Other

Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission

TDD Mechanics



Unit Testing Mechanics

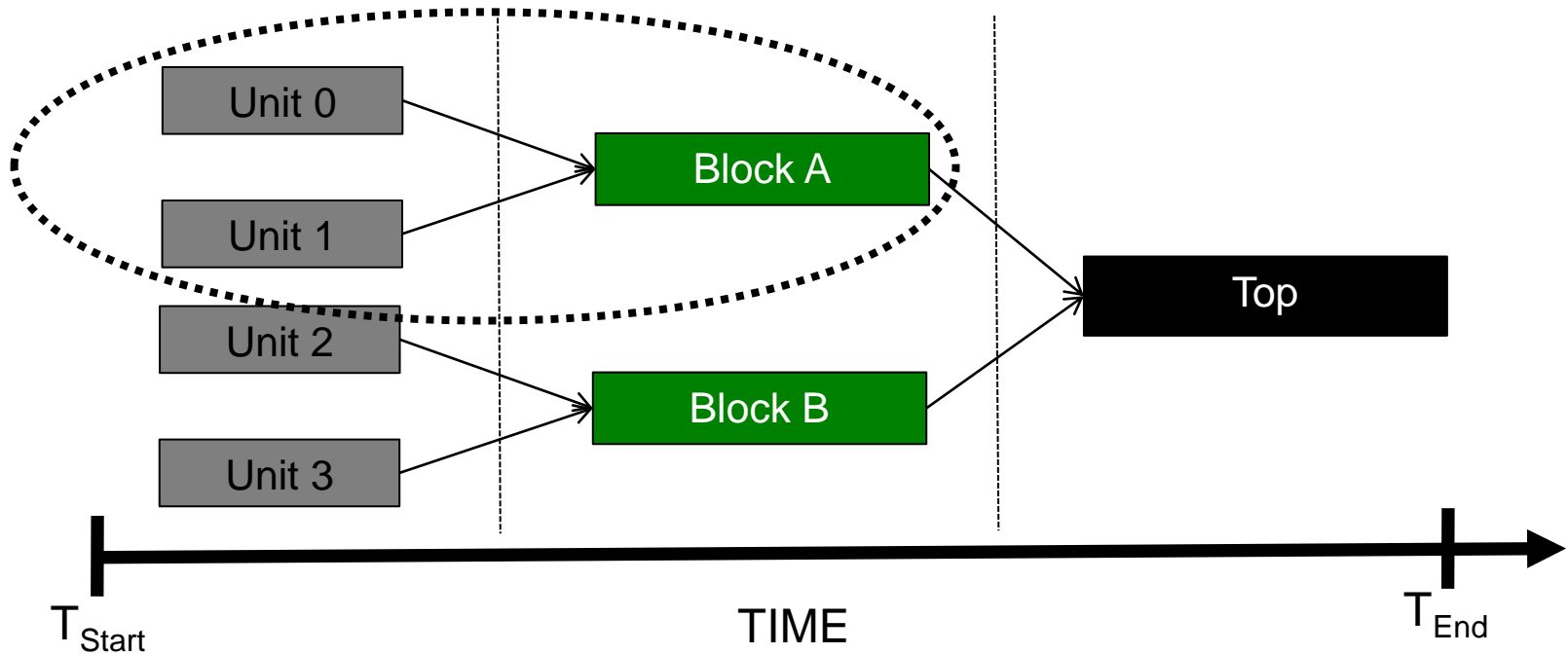


TDD To Improve Initial Code Quality

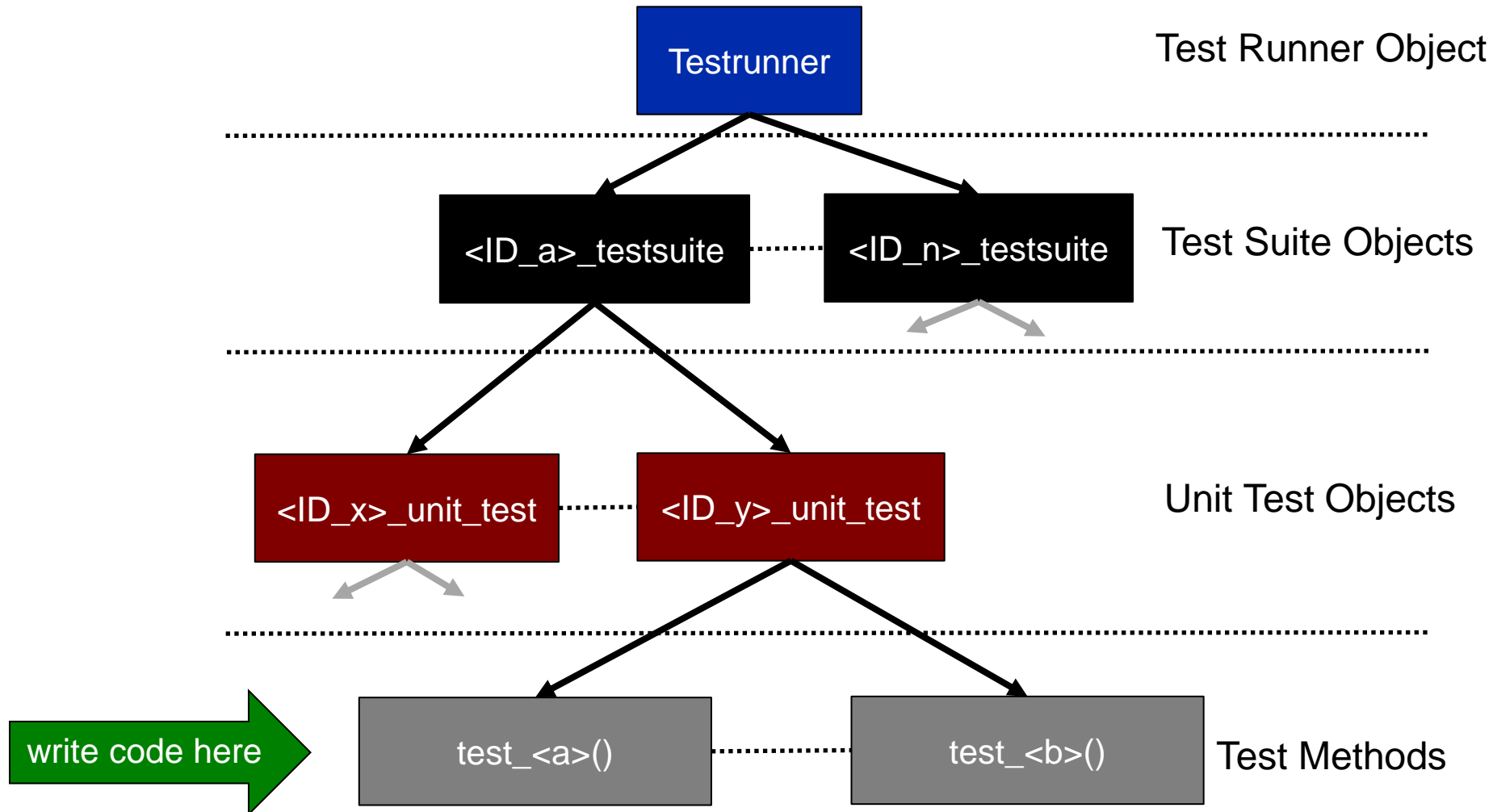
- Each module in a design has a list of dedicated unit tests
 - tests exhaustively cover functionality of the module (“exhaustively” within reason)
 - unit tests are automated
 - anytime the code changes for a given module, the corresponding tests are run to make sure it isn’t broken
- Unit tests informally increase quality, not red tape
 - tests are planned/written while building a component
 - documentation/tracking requirements are very light
- “Use-this” modules are integrated when their unit tests pass

TDD To Improve Initial Code Quality

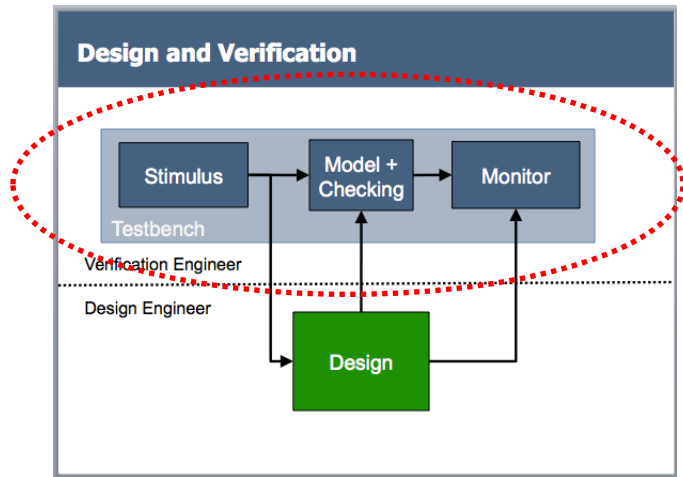
- Unit testing happens before anything else
 - then block and/or top level testing
 - Unit testing does not replace current verification practices
 - change... perhaps. replace... no.



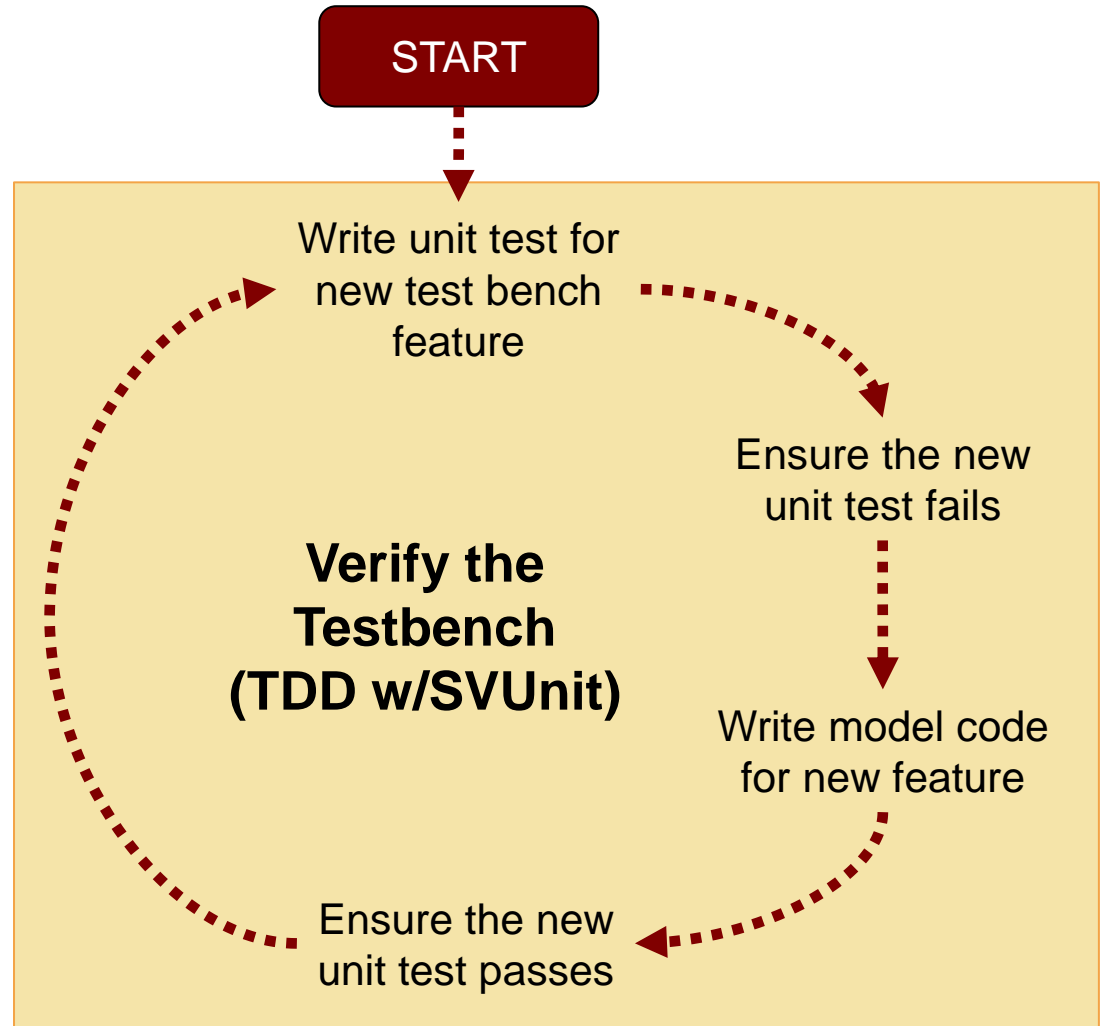
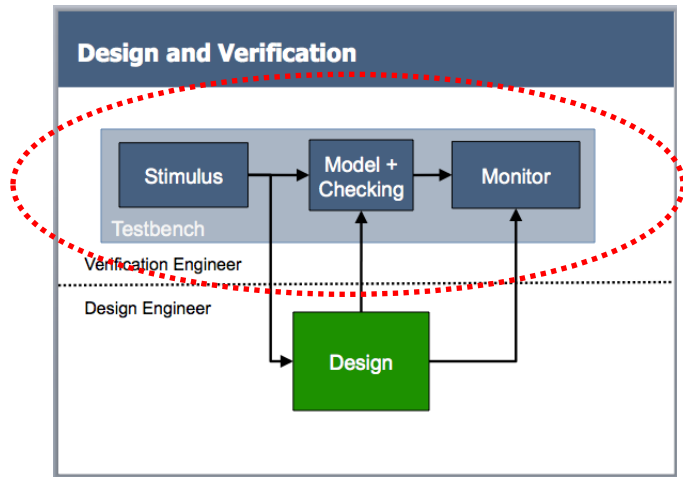
SVUnit Testing Framework



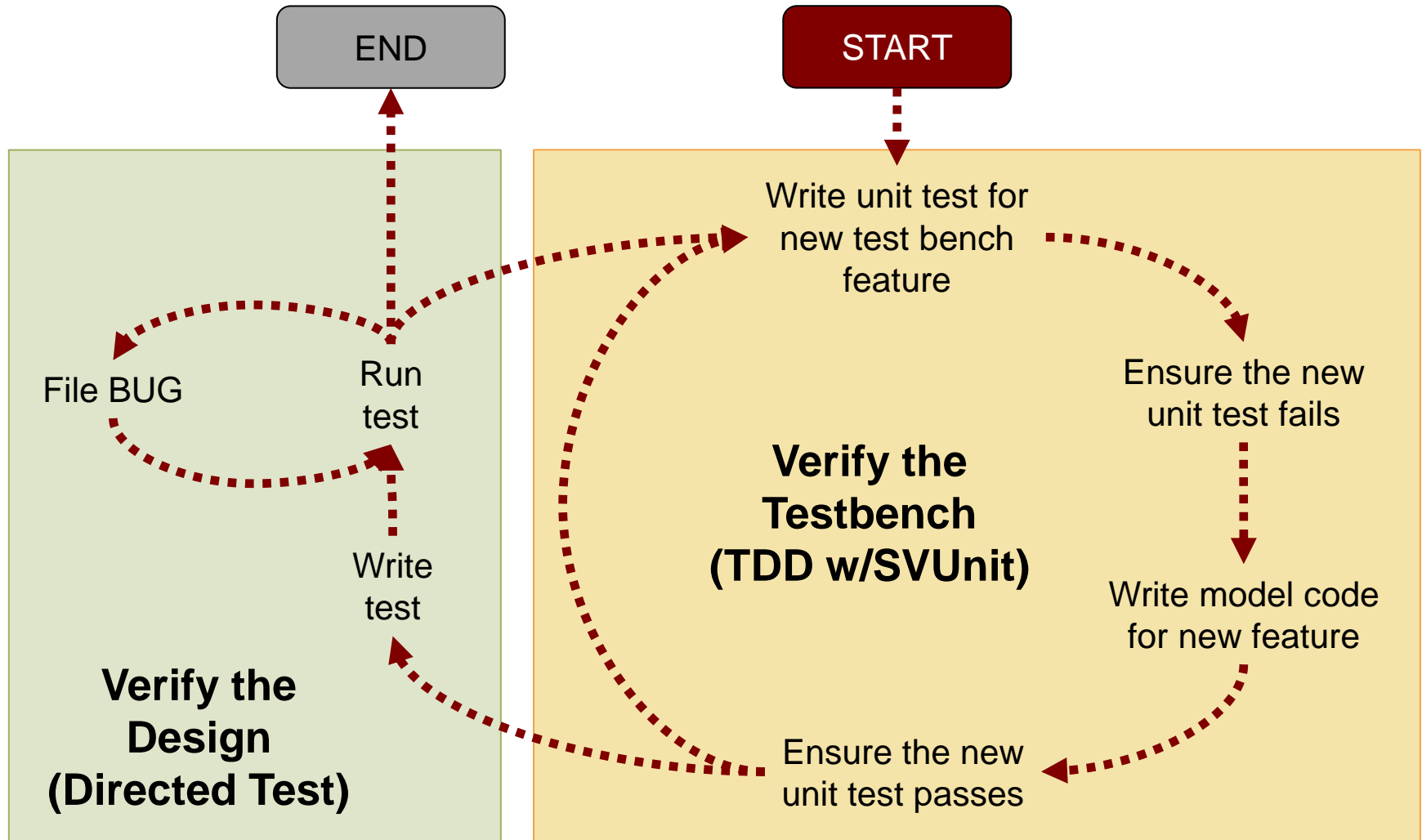
SVUnit In Practice - Mechanics



SVUnit In Practice - Mechanics



SVUnit In Practice - Mechanics



TDD To Improve Initial Code Quality

- TDD for Testbench Code
 - UVM testbench for SoC subsystem
 - ~3000 lines of code total
 - SVUnit unit tests for almost every line
 - 116 unit tests
 - ~5000 lines of test code total

TDD To Improve Initial Code Quality

- TDD for Testbench Code
 - UVM testbench for SoC subsystem
 - ~3000 lines of code total
 - SVUnit unit tests for almost every line
 - 116 unit tests
 - ~5000 lines of test code total
 - 9 weeks of effort
 - Bugs found:
 - 30 RTL bugs
 - 2 testbench bugs

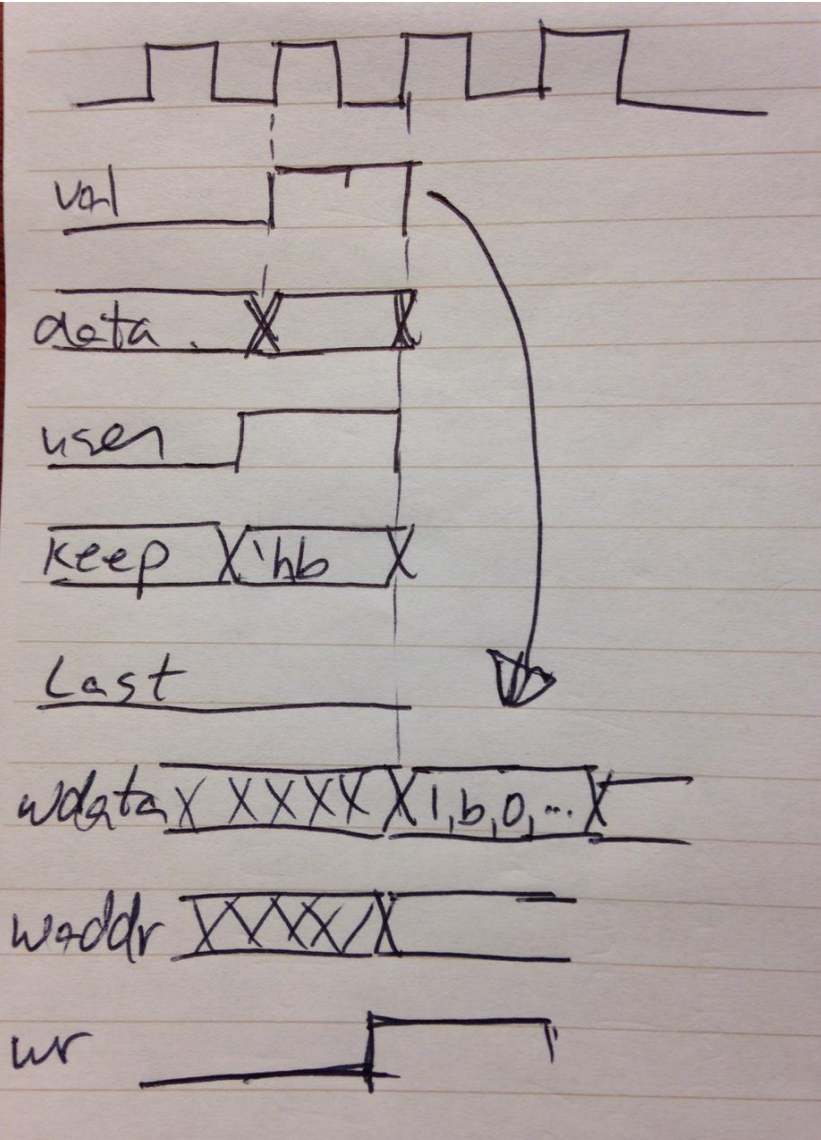
15:1



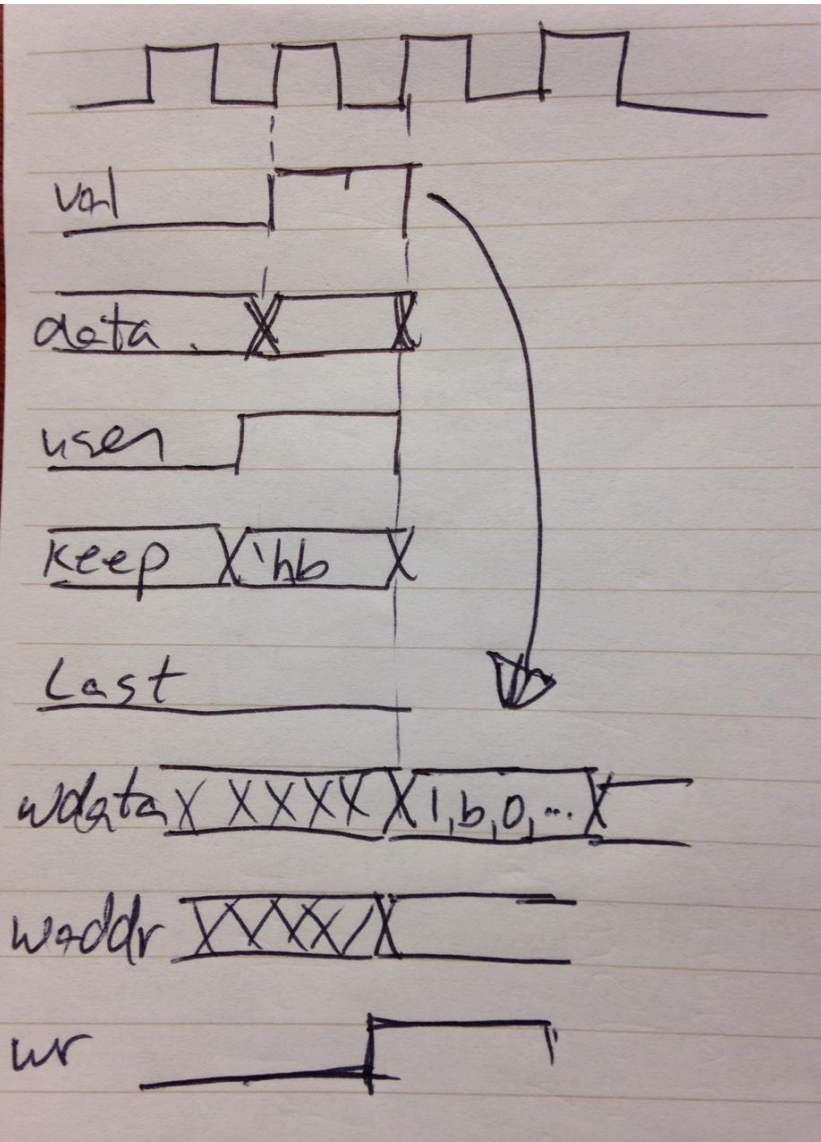
Design Lessons Learned

- Can TDD be effective given low level interactions?
- Is code partitioning TDD-friendly?
- Can concurrency be hidden/simplified?

Problem: Low Level Hardware Interactions



Problem: Low Level Hardware Interactions



```
`SVTEST(ingress_write_1_pixel)
  @(posedge clk);

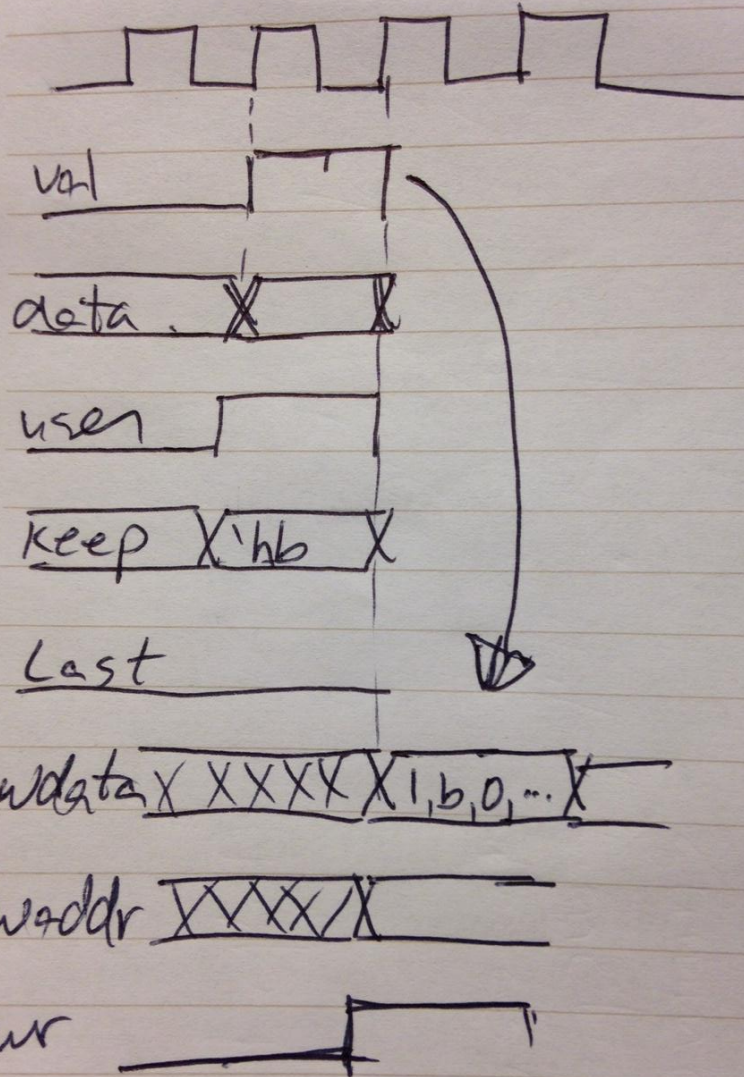
  iTVALID = 1;
  iTDATA = 'haa55bb;
  iTUSER = 1;
  iTKEEP = 'hb;
  iTLAST = 0;

  @(posedge clk);

  #1;

  `FAIL_UNLESS(wdata === { 1 , 'hb , 0 , 'haa55bb });
  `FAIL_UNLESS(waddr == 0);
  `FAIL_UNLESS(wr === 1);
`SVTEST_END
```

Solution: Create Higher Level API



```
`SVTEST(ingress_write_1_pixel)
  setIngressPixel('haa55bb);
  step();

  expectRamWrite(0, 'haa55bb);
`SVTEST_END
```

```
`SVTEST(ingress_write_1_pixel)
  @(posedge clk);

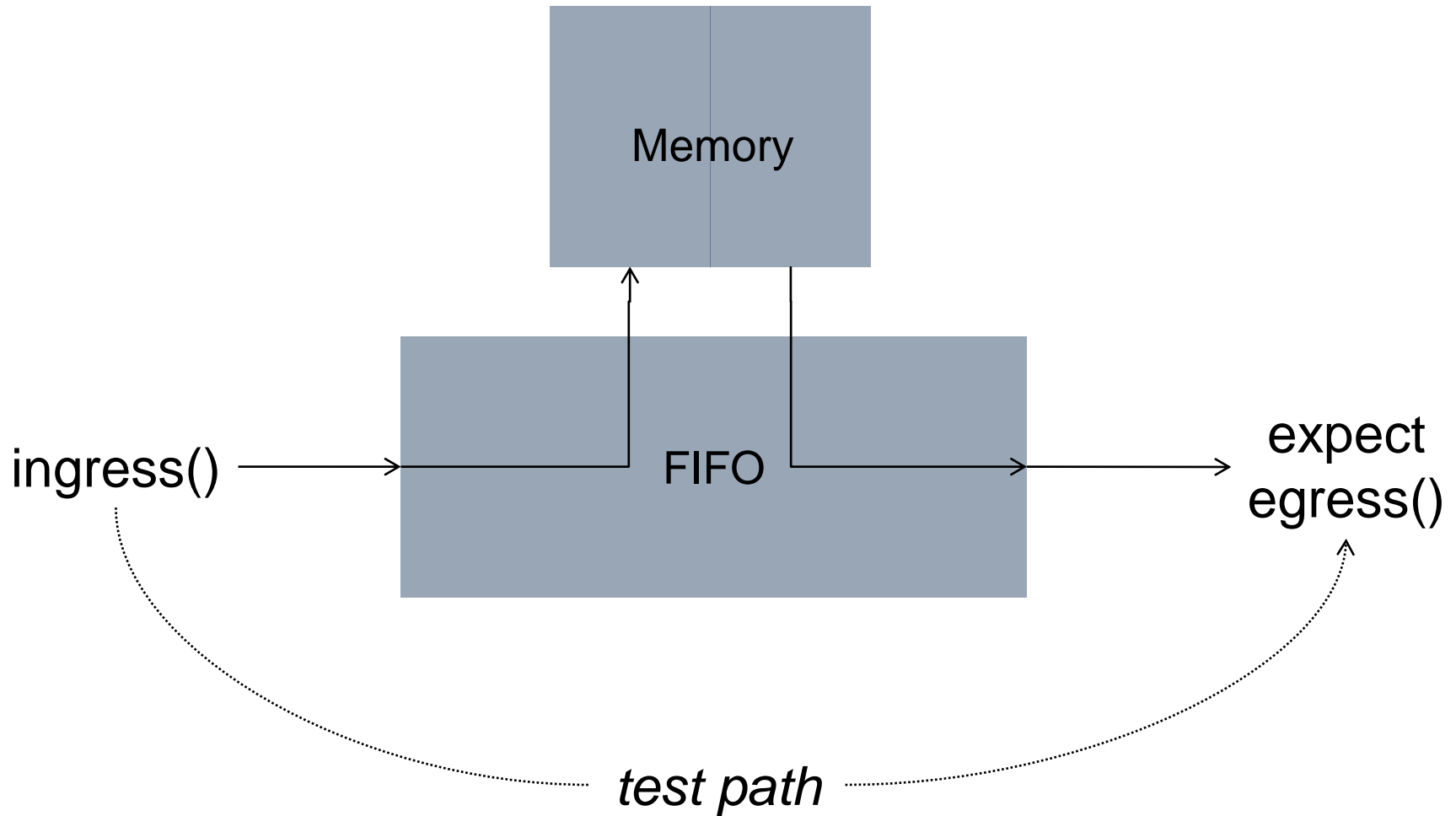
  itVALID = 1;
  itDATA = 'haa55bb;
  itUSER = 1;
  itKEEP = 'hb;
  itLAST = 0;

  @(posedge clk);

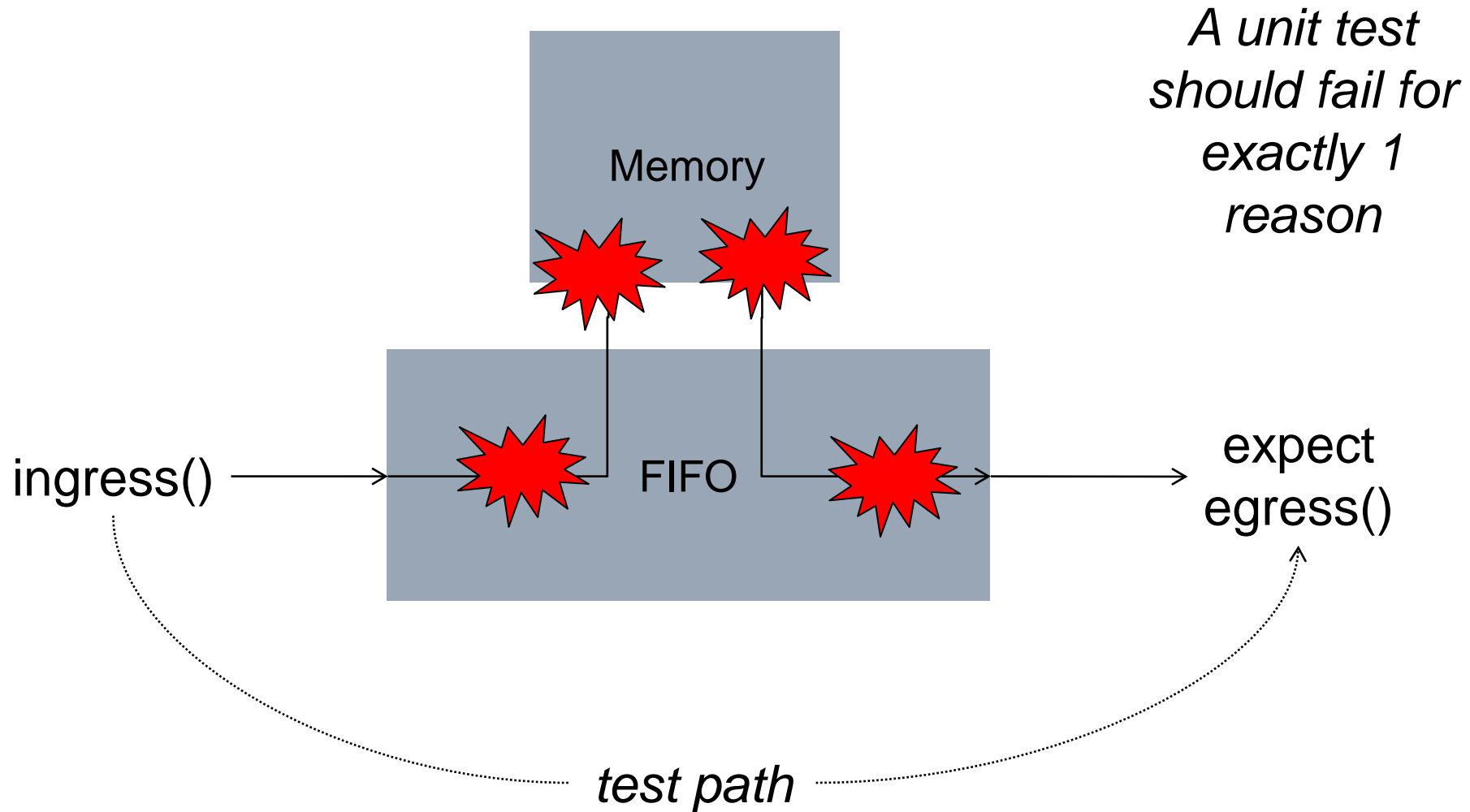
  #1;

  `FAIL_UNLESS(wdata === { 1 , 'hb , 0 , 'haa55bb });
  `FAIL_UNLESS(waddr == 0);
  `FAIL_UNLESS(wr === 1);
`SVTEST_END
```

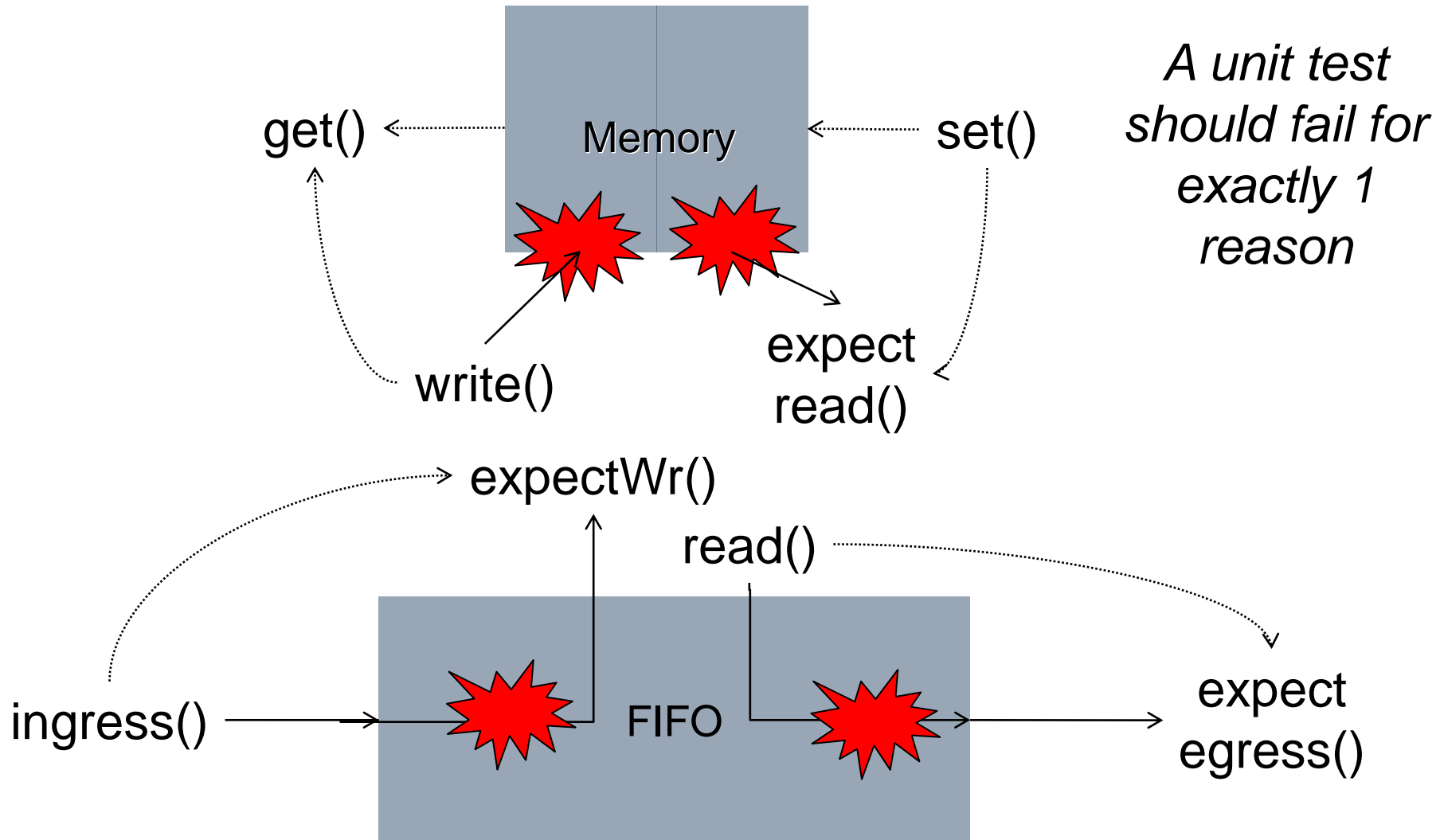
Problem: Not Structured for Unit Testing



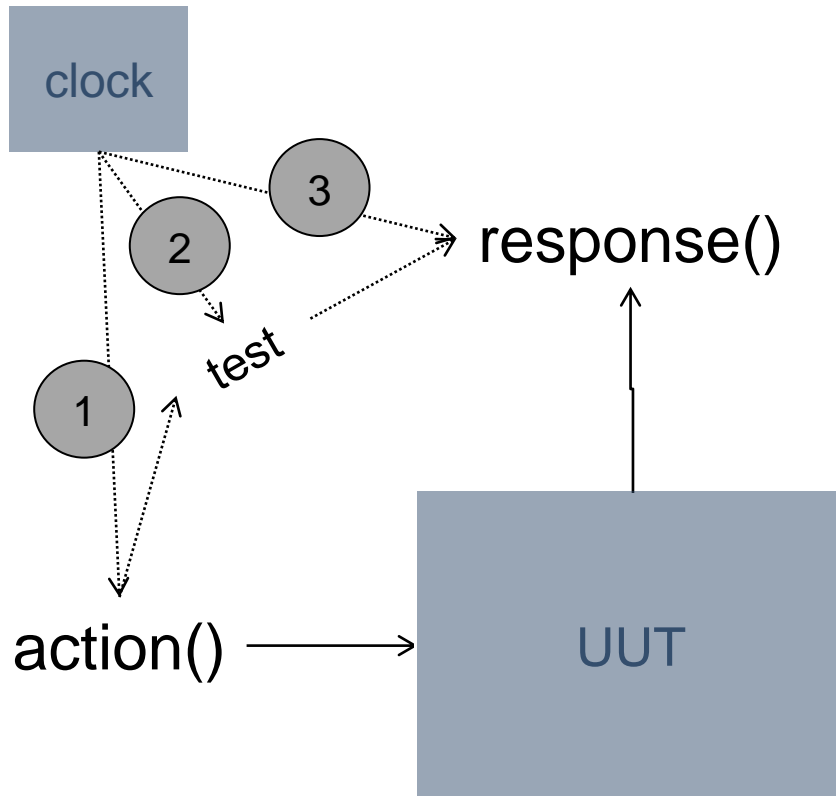
Problem: Not Structured for Unit Testing



Solution: Restructure/Isolate Functionality



Problem: Hardware is Multi-Threaded

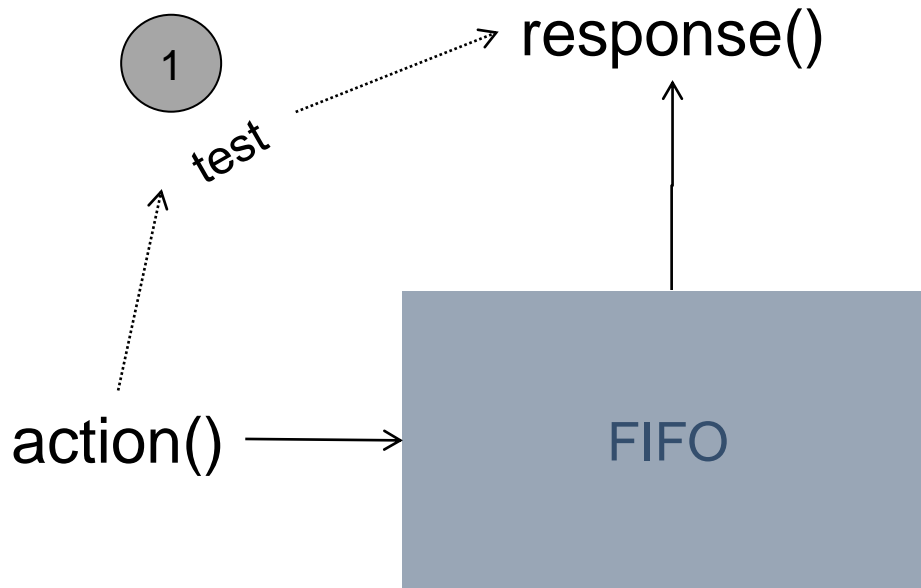


```
`SVTEST(test_with_3_threads)
  // synchronize to clk thread
  @(negedge clk);

  fork
    begin
      // some action on input thread
    end

    begin
      // wait for response thread
      `FAIL_IF(someCondition);
    end
  join
`SVTEST_END
```


Solution: Maintain a Single Thread



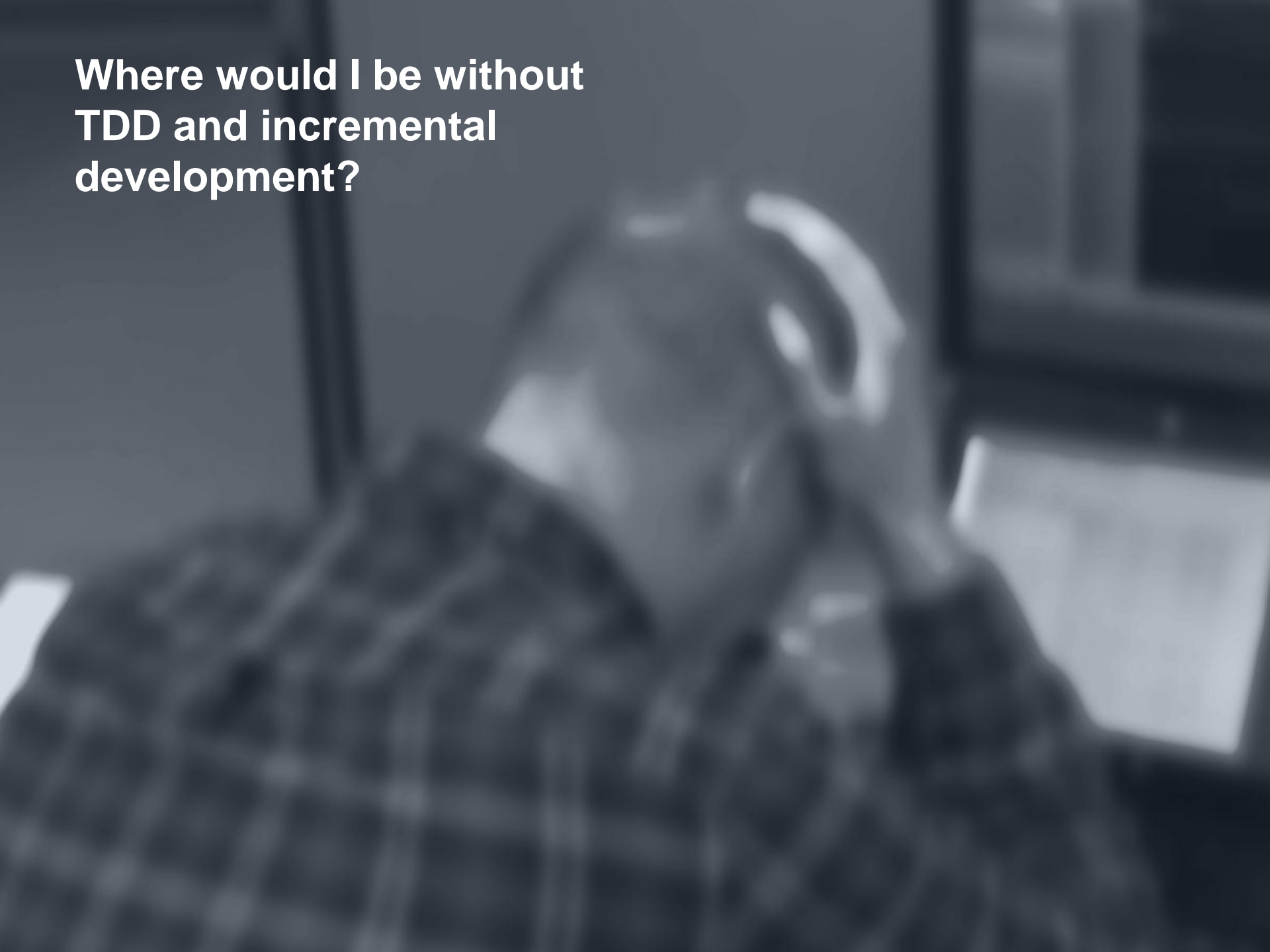
```
`SVTEST(ingress_write_1_pixel)
  setIngressPixel('haa55bb);
  step();

  expectRamWrite(0, 'haa55bb);
`SVTEST_END
```

Design Lessons Learned

- Low level interactions can work with TDD
- Design partitioning can be TDD-friendly
- Concurrency can be hidden/simplified in unit tests
- Hardware is special... but TDD still fits

**Where would I be without
TDD and incremental
development?**





SVUnit 2016

User Group Meeting

Rich Desmarais

[Signature]

Erik Juen

[Signature]

Doug

Sponsored by:

DELMOND

to 1st

Eden

Jonut Cioarlen

[Signature]

Sam Sokorac



Rob Briel

[Signature]

Mark Villalpando

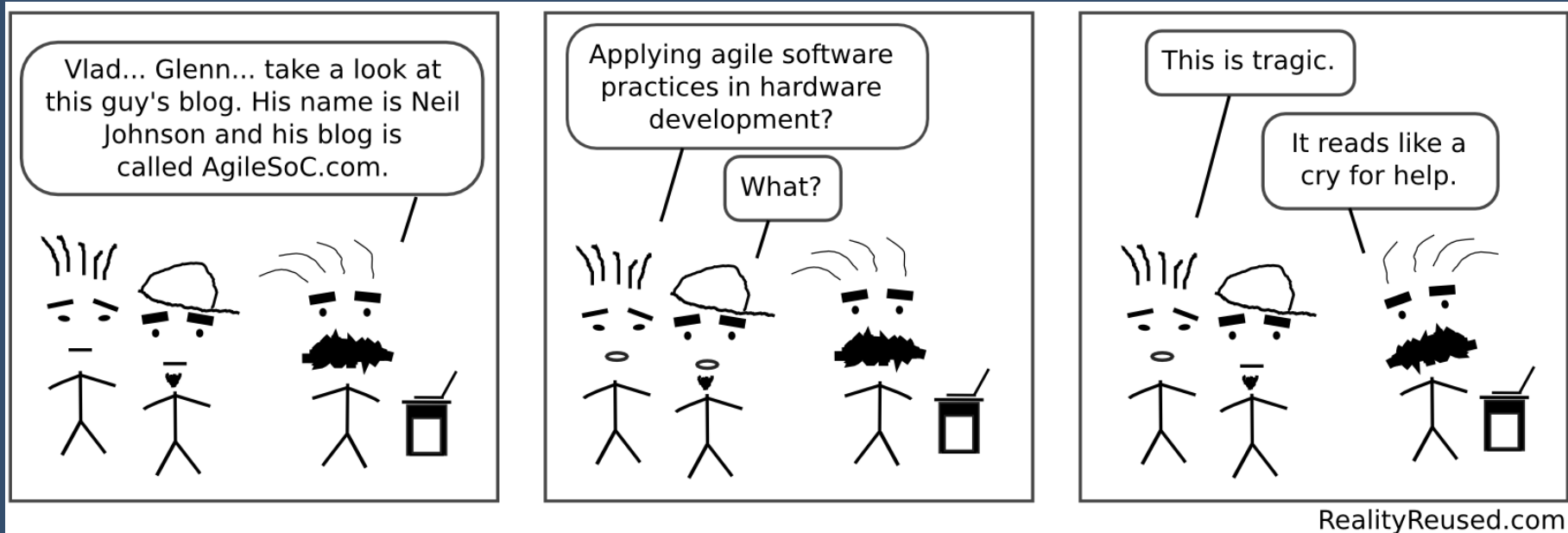
[Signature]

[Signature]

McBeck

[Signature]

[Signature]



Agile
SoC.com

Agile Hardware is real...
and it's partly my fault.

nosnhojn@gmail.com
[@nosnhojn](https://twitter.com/nosnhojn)



SVUnit Case Study: UVM-UTest

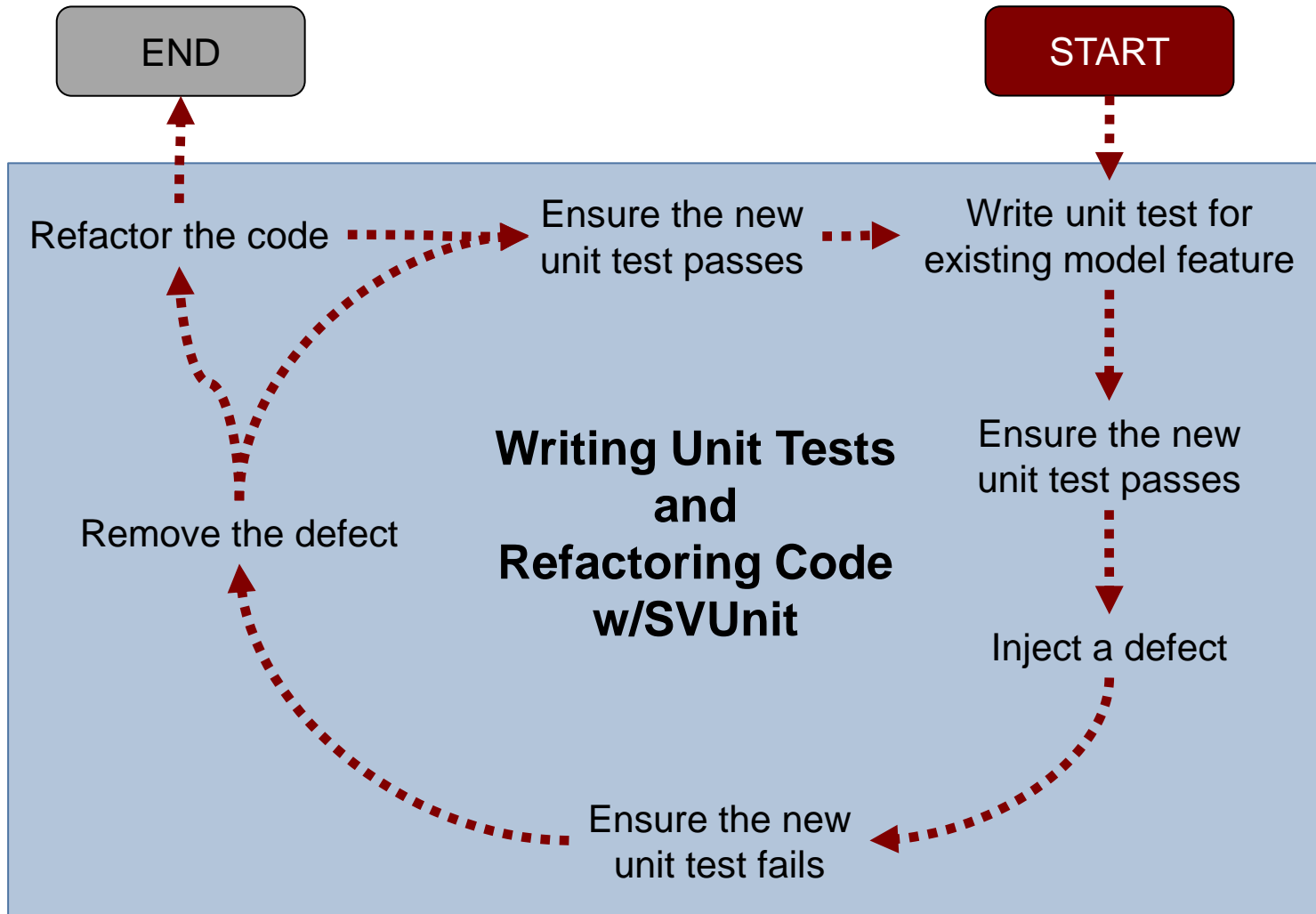
- Unit testing legacy code: UVM-1.1d
 - Lock down functionality to ease maintenance
 - finer granularity tests catch issues we don't target directly
 - unit tests provide a backstop when maintaining code
 - UVM-1.1d == thousands and thousands of lines of code
 - >550 unit tests total
 - 6 weeks of effort

SVUnit Case Study: UVM-UTest

- Unit testing legacy code: UVM-1.1d
 - Lock down functionality to ease maintenance
 - finer granularity tests catch issues we don't target directly
 - unit tests provide a backstop when maintaining code
 - UVM-1.1d == thousands and thousands of lines of code
 - >550 unit tests total
 - 6 weeks of effort
 - 10 defect reports filed

Viewing Issues (1 - 10 / 10) [Print Reports]								
[CSV Export]								
	P	ID	Type	Category	Severity	Status	Updated▼	Summary
		0004602	TBD	BCL	minor	new	2013-06-19	uvm_printer::print_object_header cannot support user specified scope_separator
		0004600	TBD	BCL	minor	new	2013-06-19	Incomplete string/separator handling in uvm_leaf_scope
		0004640	TBD	BCL	minor	new	2013-06-17	uvm_has_wildcard is the only function that treats '+' as a wildcard
		0004638	TBD	BCL	minor	new	2013-06-17	uvm_is_array returns true for malformed string inputs
		0004637	TBD	BCL	minor	new	2013-06-17	incomplete is_wildcard handling in uvm_get_array_index_string leaves is_wildcard in erroneous state
		0004636	TBD	BCL	minor	new	2013-06-17	uvm_get_array_index_string accepts illegal index characters
		0004635	TBD	BCL	minor	new	2013-06-17	incomplete is_wildcard handling in uvm_get_array_index_int leaves is_wildcard in erroneous state
		0004634	TBD	BCL	minor	new	2013-06-17	uvm_get_array_index_int treats indices with radix specified as illegal
		0004609	TBD	BCL	minor	new	2013-05-28	uvm_printer::print_real argument to adjust_name is incorrect
		0004601	TBD	BCL	minor	new	2013-05-24	uvm_vector_to_string incorrectly displays negative numbers

SVUnit Case Study: UVM-UTest



My Commitment to Agile Hardware development is to...

Build a platform to showcase TDD as a credible technique for doing hw/sw co-development.

this is why I'm making this commitment... code quality and synchronization between hw/sw developers are of major importance in embedded systems development. I think this platform could be a great tool for addressing both issues; TDD being applied in both domains on a platform that encourages and enables collaboration.

“...showcase TDD as a credible technique for doing hw/sw co-development.”

This is how I intend to meet my commitment... we'll use ~~an~~ an FPGA board to implement some simple yet visual application. The app will be split between SW running on an ARM Cortex A9 and logic on the fpga. We'll build the code on both sides w/ TDD, likely GoogleTest for the C/C++ and ~~SW~~ SUUnit on the verilog side. For people that use the platform, they'll have access to all the code so they can add/change/move features in or between sw/hw domains. then they can rebuild/deploy and watch it go. Ease of use is top priority. We want people to be comfortable playing w/ it. This is how I'll know if I've achieved my commitment...

“code quality and synchronization between hw/sw developers are of major importance in embedded systems development.”

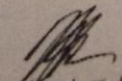
I'll take it to Agile 2014 and set it up in the coaching area. If I get 10 people sitting down, changing code, rebuild and redeploy then I'm happy. less than 10 will be a miserable failure!!

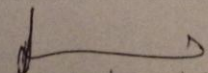
“we'll use an FPGA board to implement some simple yet visual application.”

I plan to meet my commitment by...

Agile 2014 conference is July 28th. Done and polished before then.

PS: “We” is me and Soheil. This is a 2-man commitment!

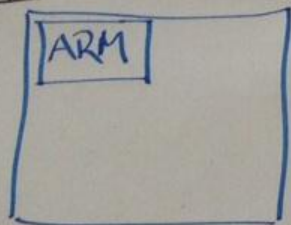
 Apr 10/2014
David Johnson


Soheil Salehian April 16/2014

Agile Soc.com

Agile Hardware w/TDD

I



Programming Steps

Program
FPGA

II

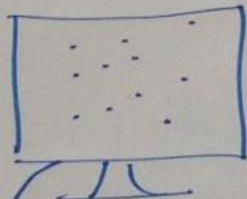
Run SW



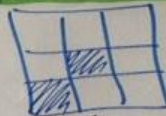
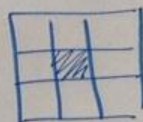
Erase



I

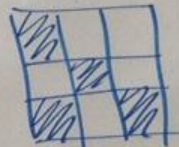
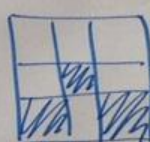


Application



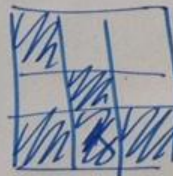
< 2 neighbours

underpopulated



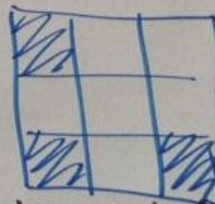
2 || 3 neighbours

sustained



> 4 neighbours

overpopulated



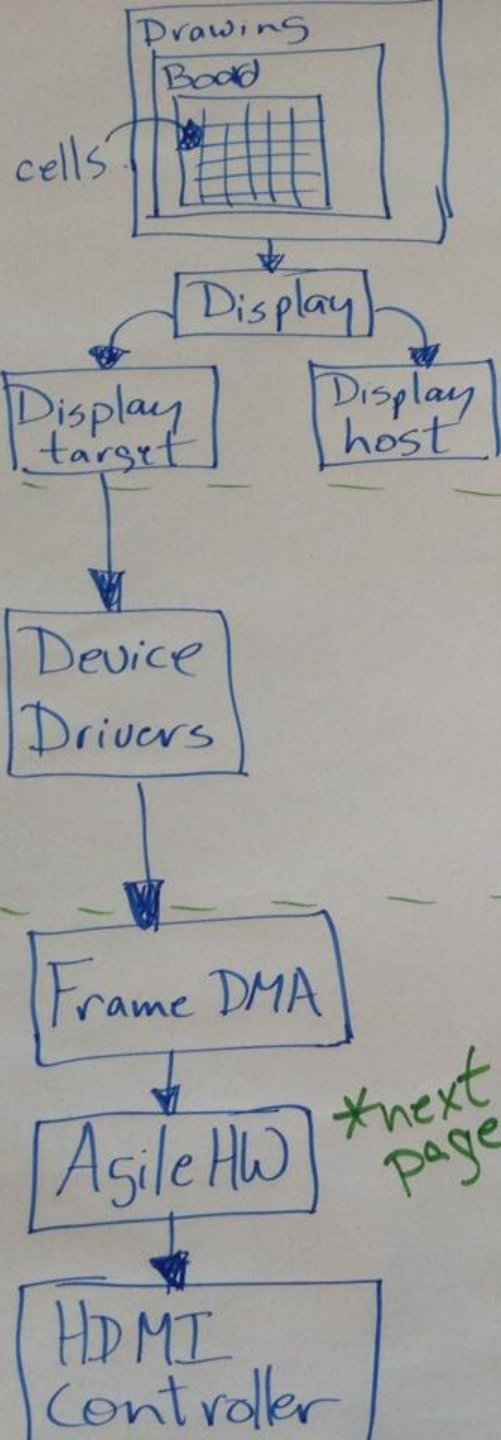
3 neighbours

new life

Drawing

*

TDD



TDD'ing Hardware

C++

- * Googletest
- * Googlemock
- * Mock/isolate class interaction

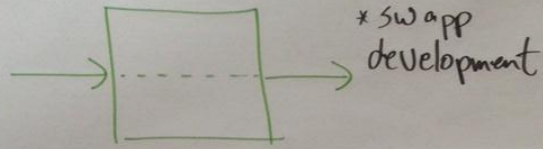
C/C++

- * Googletest
- * Googlemock
- * Mock/isolate driver interaction

Verilog

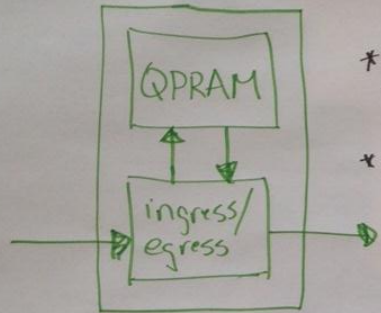
- * SV Unit
- * Verilog
- * IP

Agile HW v0.1 (wires) Last year

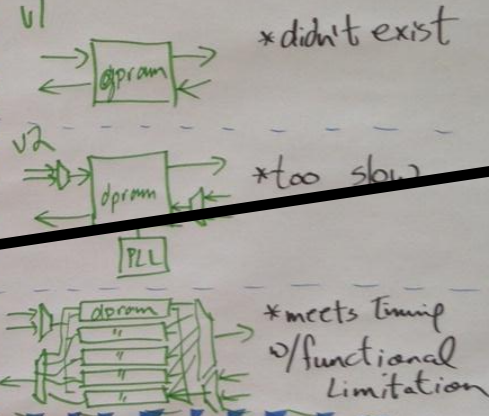


Iteration 1

Agile HW v1 (fifo)

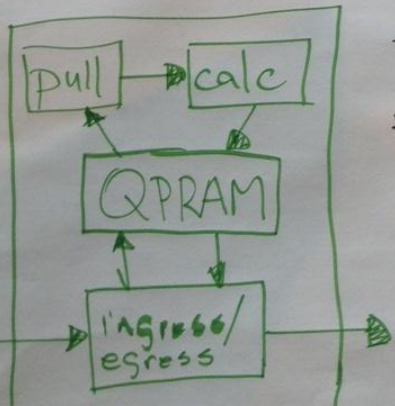


QPRAM (refactored)



Iteration 2

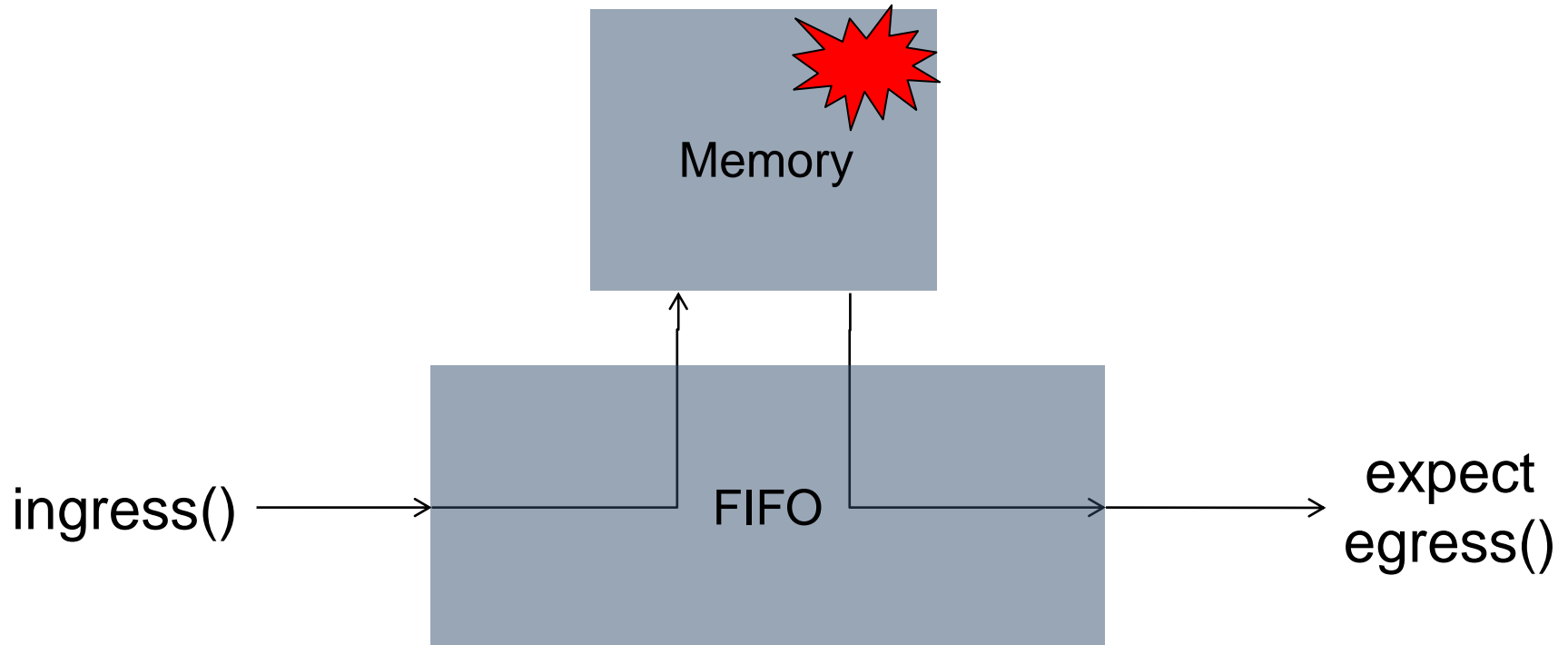
Agile HW v2 (shadow)



This year

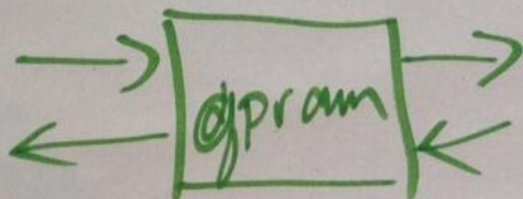
Iteration 3

Problem: Design Doesn't Meet Timing



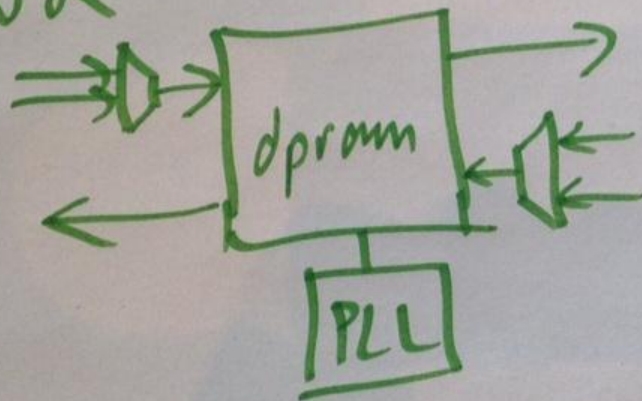
QPRAM (refactored)

v1



* didn't exist

v2



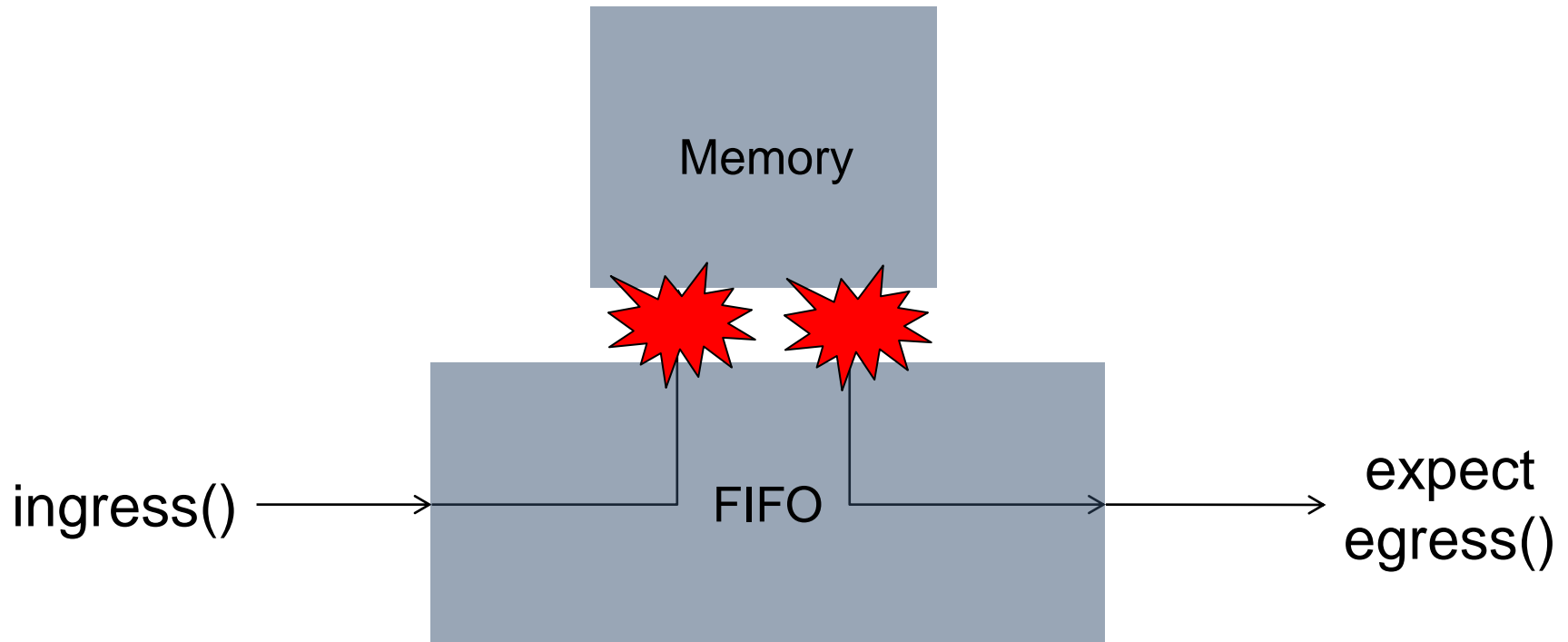
* too slow



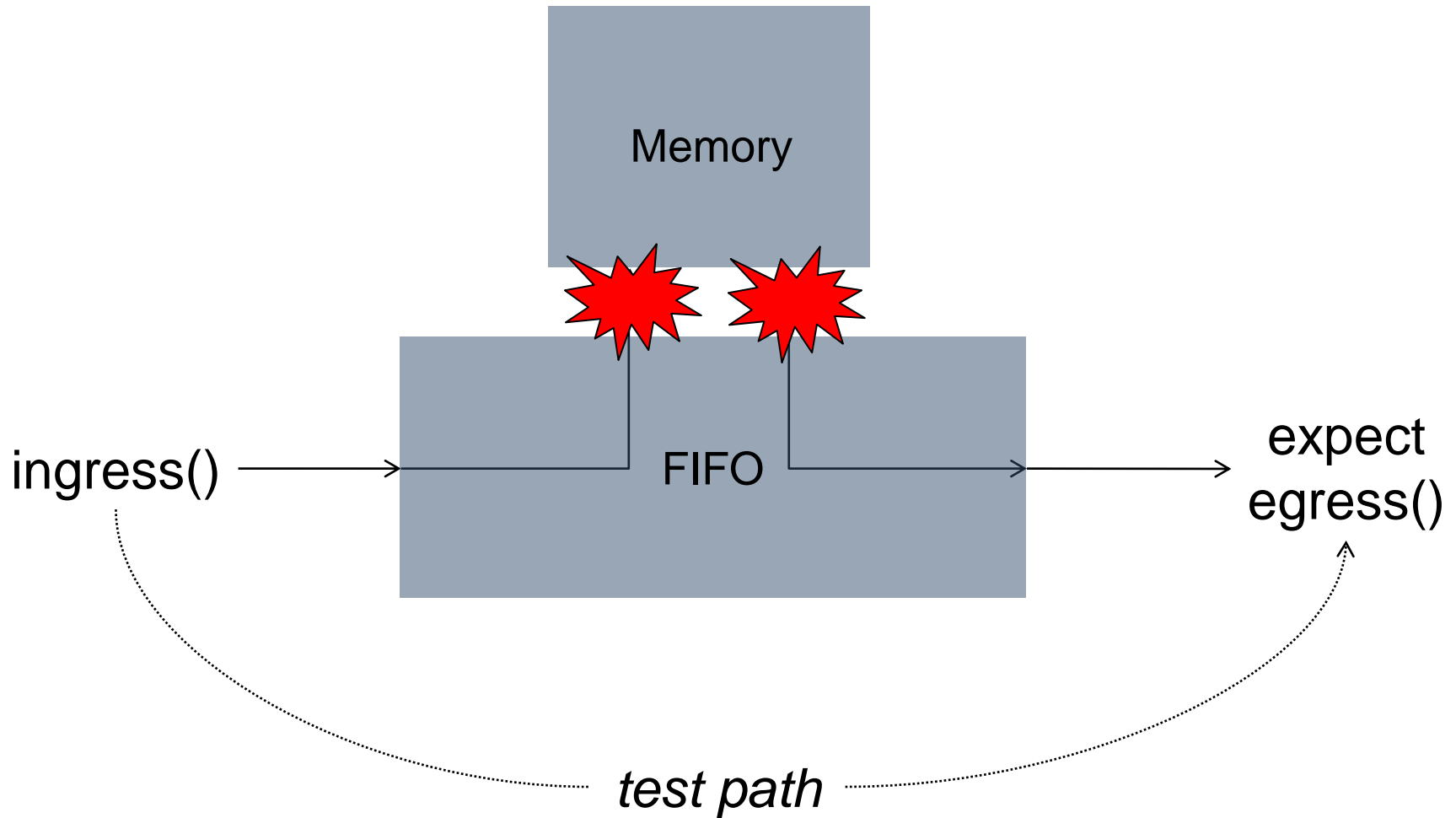
* meets timing
w/functional
Limitation

fo
sic
ram

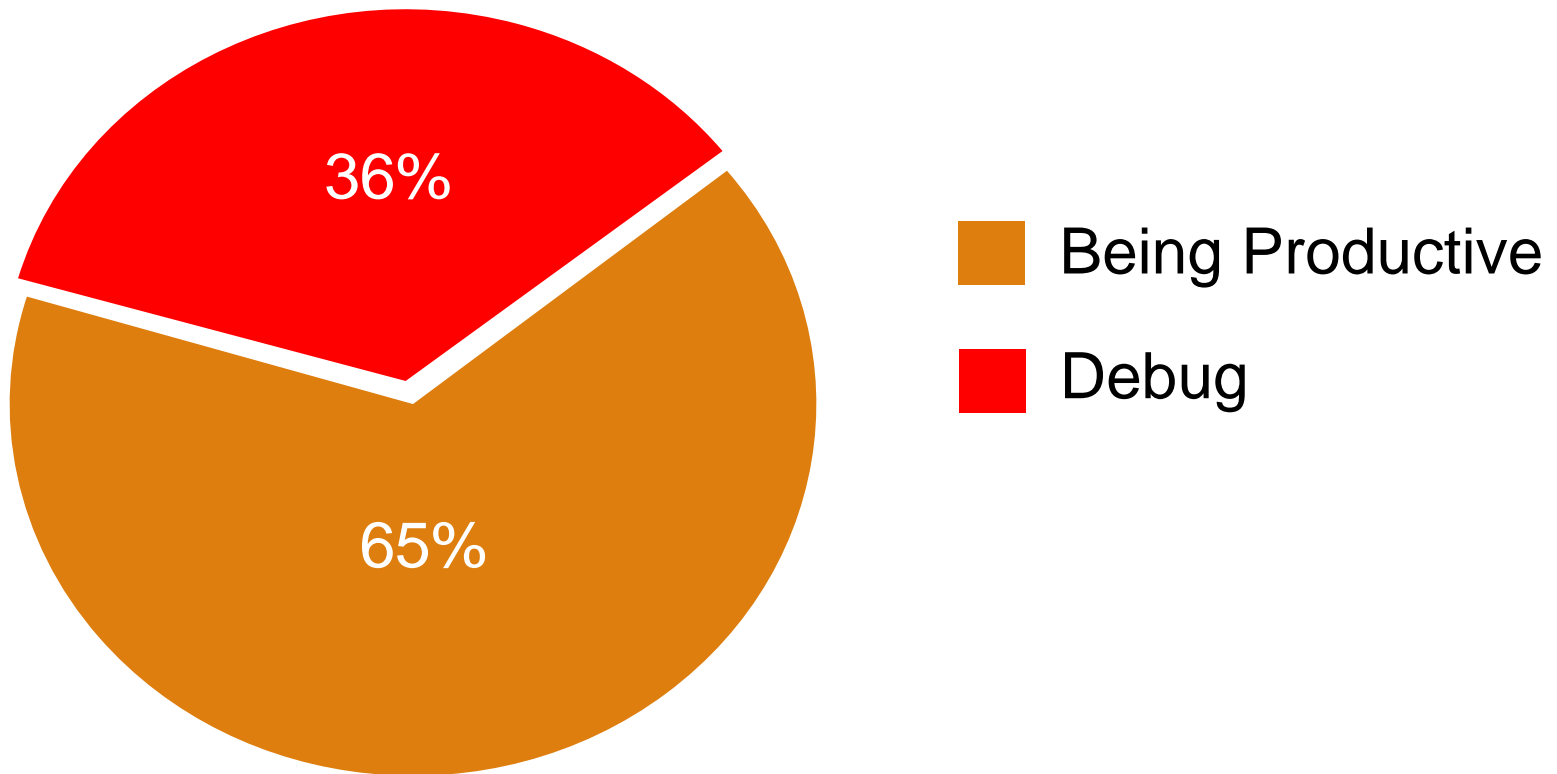
Problem: It's Not Over With TDD



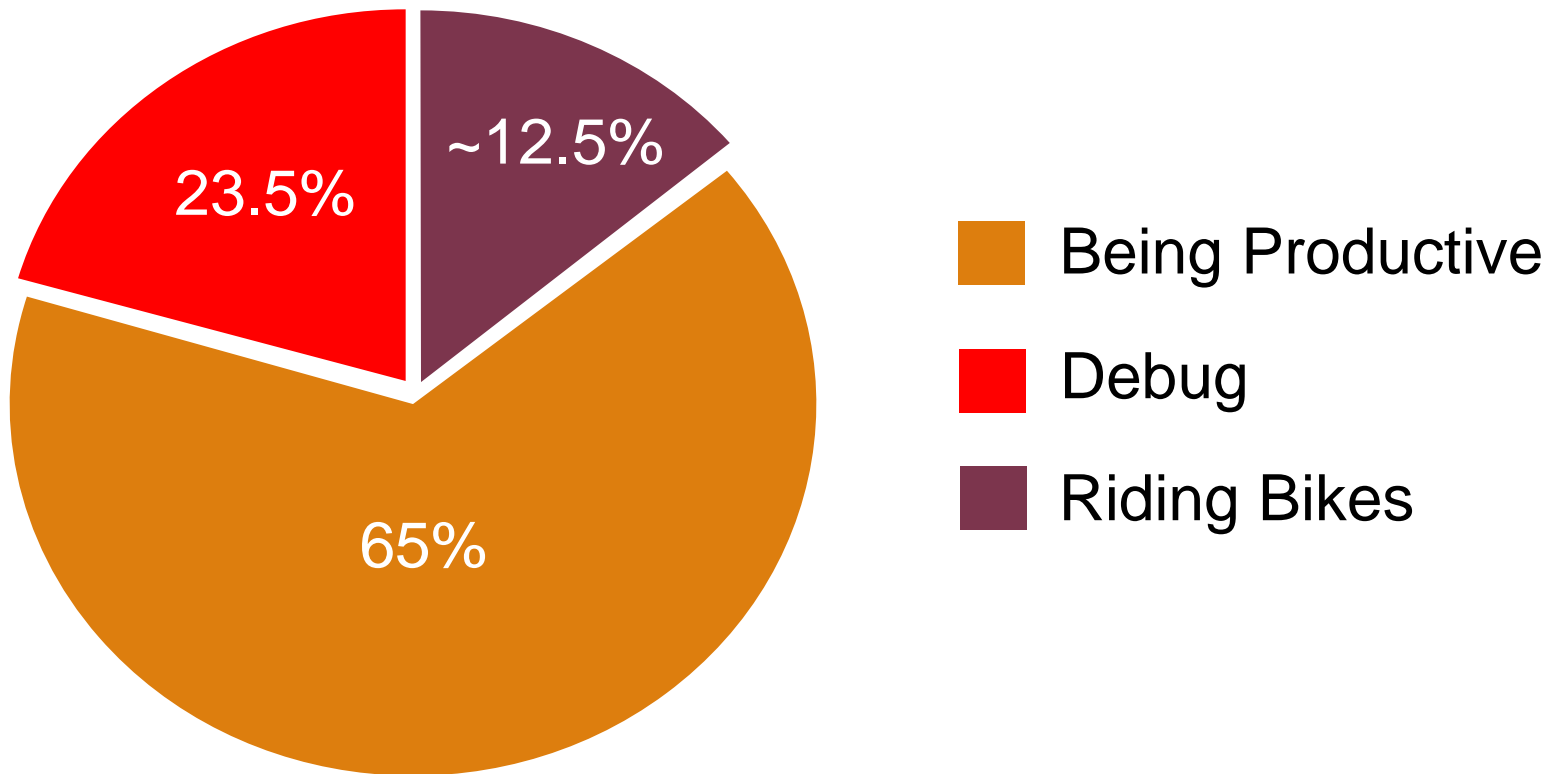
Solution: Integration Tests For Integration



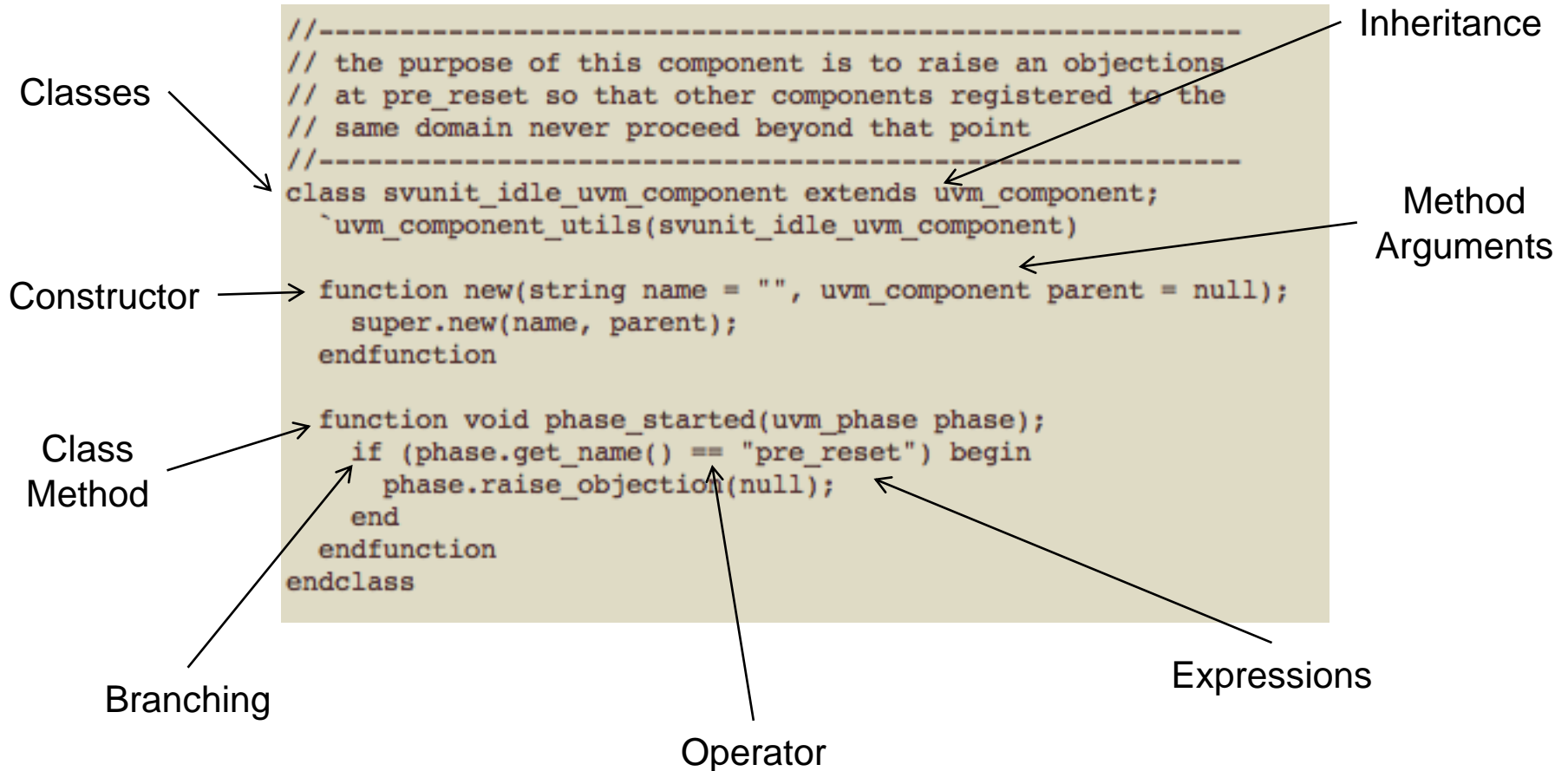
Where Verification Engineers Spend There Time



Where Verification Engineers Spend Their Time



Language Basics (Verification)



Language Basics (Design)

```
//-----  
//-----  
// calculate the waddr to the memory based on the  
// frame position flags  
//-----  
//-----  
  
always @(negedge rst_n or posedge clk) begin  
    if (!rst_n) begin  
        next_waddr <= 0;  
    end  
  
    else begin  
        if (calc_strobe && first_row_flag && first_column_flag) begin  
            next_waddr <= EFFECTIVE_WIDTH;  
        end  
        else if (calc_strobe && last_row_flag && first_column_flag) begin  
            next_waddr <= next_waddr + EFFECTIVE_WIDTH;  
        end  
        else if (calc_strobe && first_row_flag && !first_column_flag ||  
            calc_strobe && last_row_flag && !first_column_flag ||  
            strobe_3_of_4) begin  
            next_waddr <= next_waddr - EFFECTIVE_WIDTH;  
        end  
        else if (strobe_2_of_2 || strobe_2_of_4 || strobe_4_of_4) begin  
            next_waddr <= next_waddr + EFFECTIVE_WIDTH + 1;  
        end  
        else if (next_wr) begin  
            if (next_waddr < 6 * EFFECTIVE_WIDTH-1) next_waddr <= next_waddr + 1;  
            else  
                next_waddr <= 0;  
        end  
    end  
end  
end
```

Synchronous
Logic

Reset State

Flip-flop

AND/OR gates

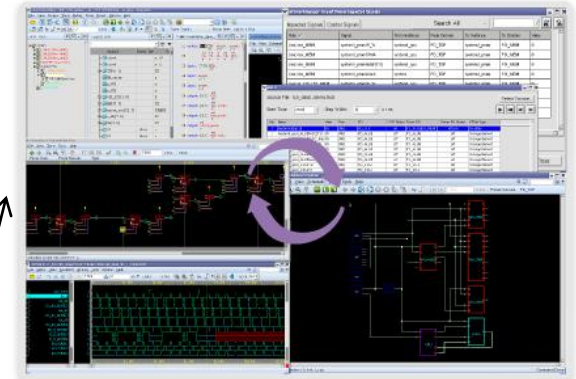
Adder

Hardware Simulation

```
//-----  
// calculate the waddr to the memory based on the  
// frame position flags  
//-----  
//  
always @(negedge rst_n or posedge clk) begin  
    if (!rst_n) begin  
        next_waddr <= 0;  
    end  
  
    else begin  
        if (calc_strobe && first_row_flag && first_column_flag) begin  
            next_waddr <= EFFECTIVE_WIDTH;  
        end  
        else if (calc_strobe && last_row_flag && first_column_flag) begin  
            next_waddr <= next_waddr + EFFECTIVE_WIDTH;  
        end  
        else if (calc_strobe && first_row_flag && !first_column_flag ||  
                calc_strobe && last_row_flag && !first_column_flag ||  
                strobe_3_of_4) begin  
            next_waddr <= next_waddr - EFFECTIVE_WIDTH;  
        end  
        else if (strobe_2_of_2 || strobe_2_of_4 || strobe_4_of_4) begin  
            next_waddr <= next_waddr + EFFECTIVE_WIDTH + 1;  
        end  
        else if (next_wr) begin  
            if (next_waddr < 6 * EFFECTIVE_WIDTH - 1) next_waddr <= next_waddr + 1;  
            else  
                next_waddr <= 0;  
            end  
        end  
    end  
end
```

```
//-----  
// the purpose of this component is to raise an objections  
// at pre_reset so that other components registered to the  
// same domain never proceed beyond that point  
//-----  
class svunit_idle_uvm_component extends uvm_component;  
    `uvm_component_utils(svunit_idle_uvm_component)  
  
    function new(string name = "", uvm_component parent = null);  
        super.new(name, parent);  
    endfunction  
  
    function void phase_started(uvm_phase phase);  
        if (phase.get_name() == "pre_reset") begin  
            phase.raise_objection(null);  
        end  
    endfunction  
endclass
```

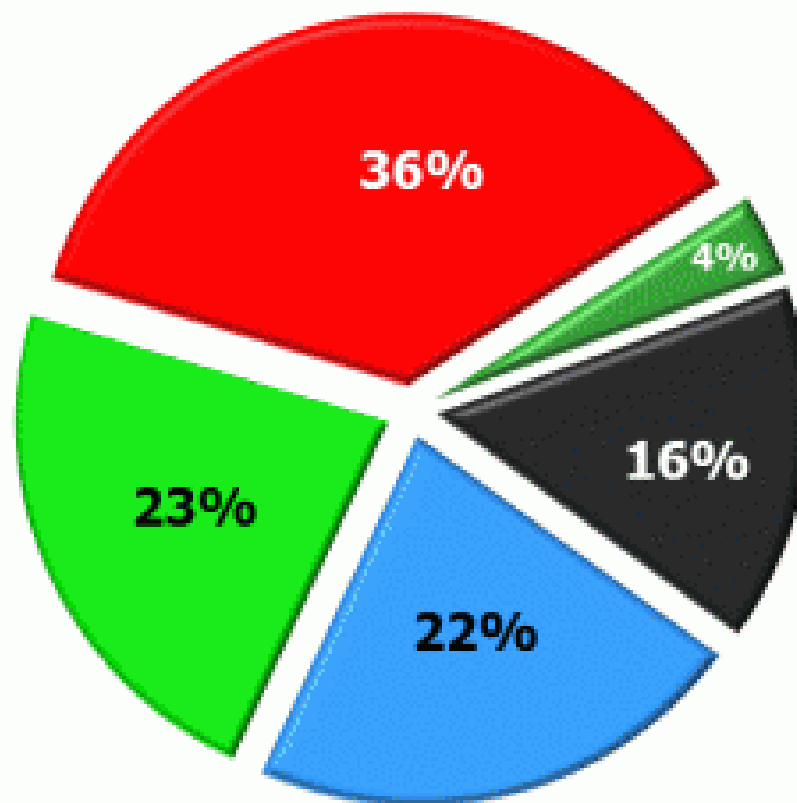
Simulator



results.log

Effort and Results

Mean time Non-FPGA verification engineers spends in different tasks

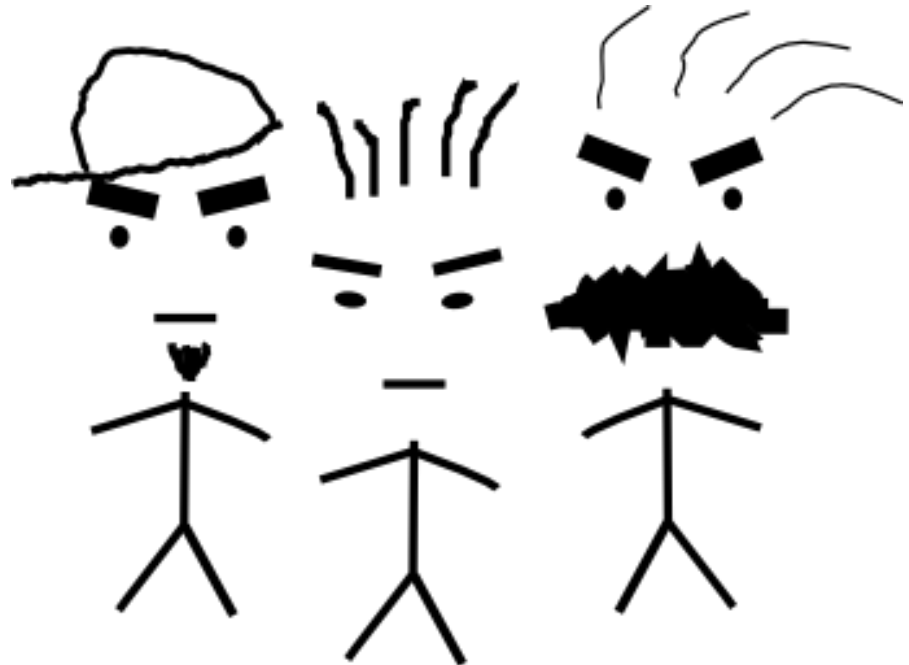
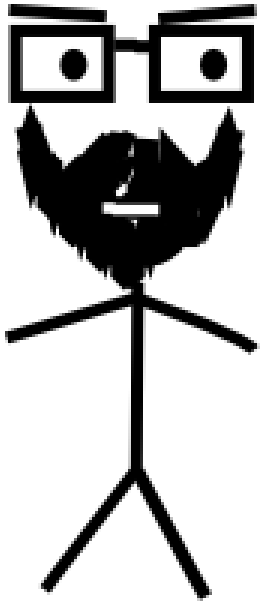


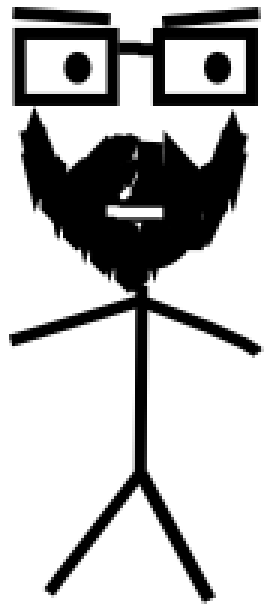
More time spent in **REDO** than any other task!

- Test Planning
- Testbench Development
- Creating and Running Test
- **REDO**
- Other

Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission

TDD is a great idea
that can save us a
lot of money by
helping us avoid
writing buggy code.





We're too busy to
save money.

This guy
is an idiot

It's lunch time.
Why are you
telling us this at
lunch time?

