Supporting a Vendor ABI Variant in Clang

Paul T. Robinson, Sony Interactive Entertainment

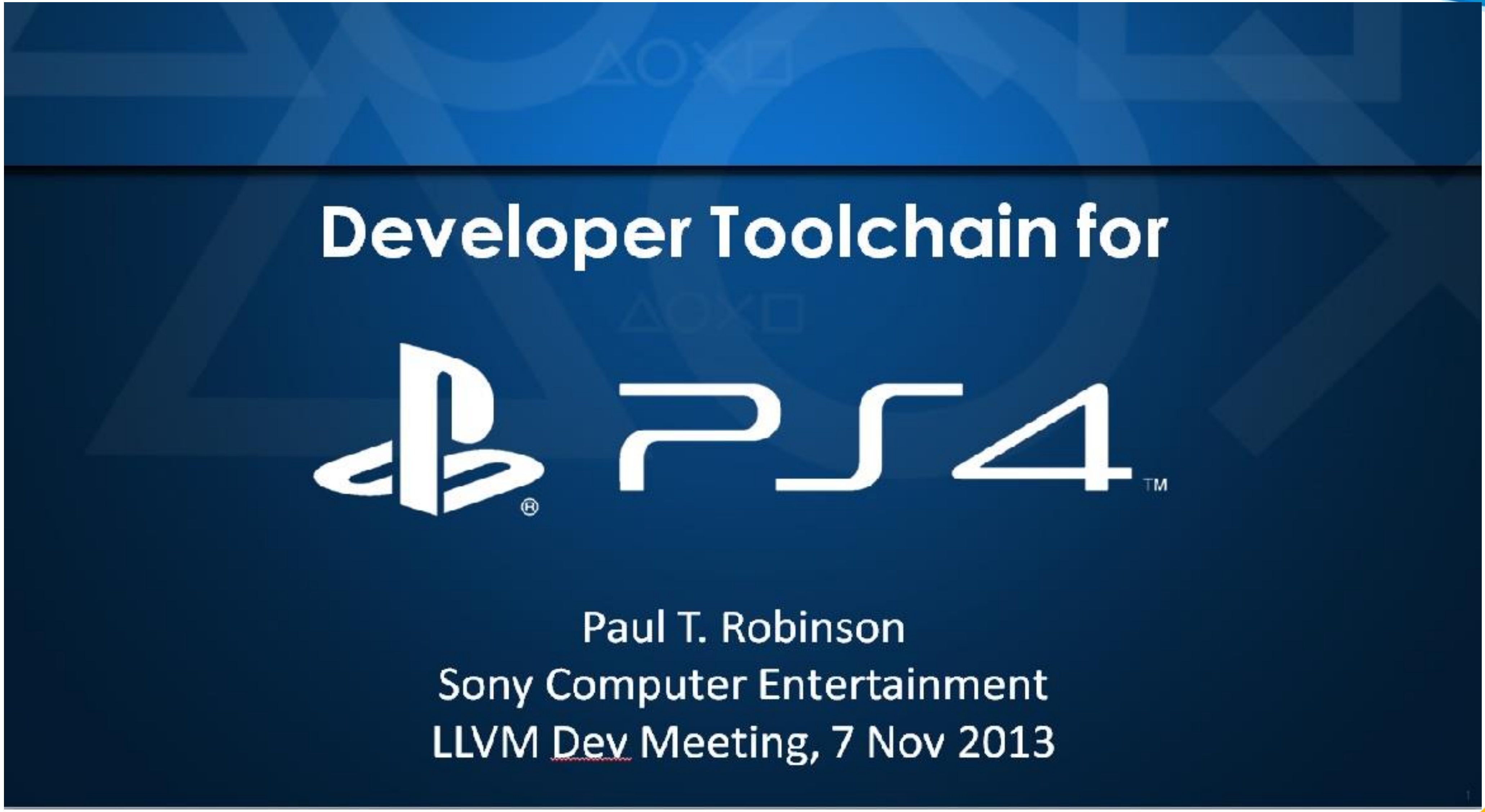
LLVM Developers' Meeting, October 2019

In the Beginning…

First-release PS4® toolchain centered on Clang 3.2

- With an assortment of tweaks, customizations, etc
- Used to build the OS, apps, games – *everything* that runs on PS4
- Very well received by the studios

Product Launch – 2013



Developer Toolchain for

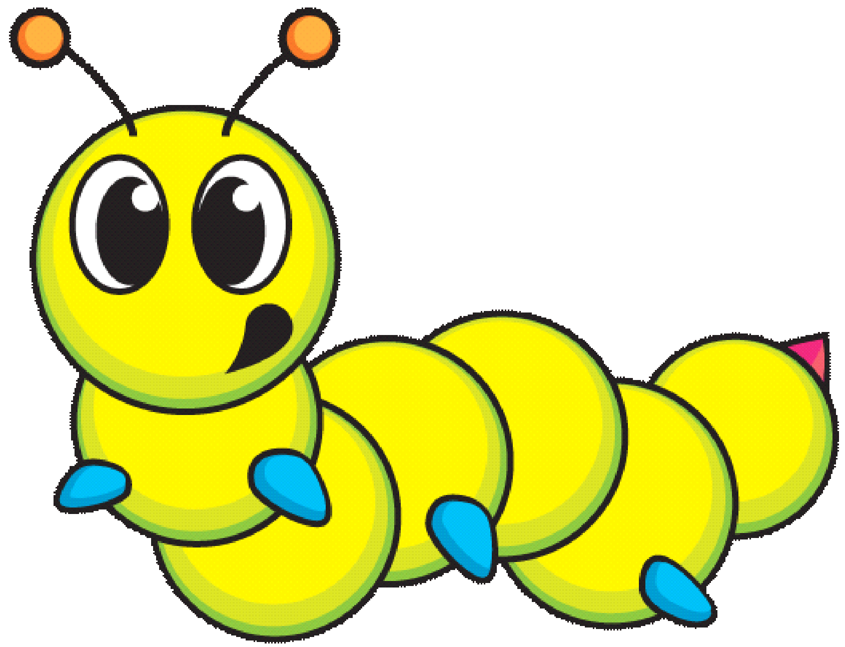
Paul T. Robinson

Sony Computer Entertainment

LLVM Dev Meeting, 7 Nov 2013

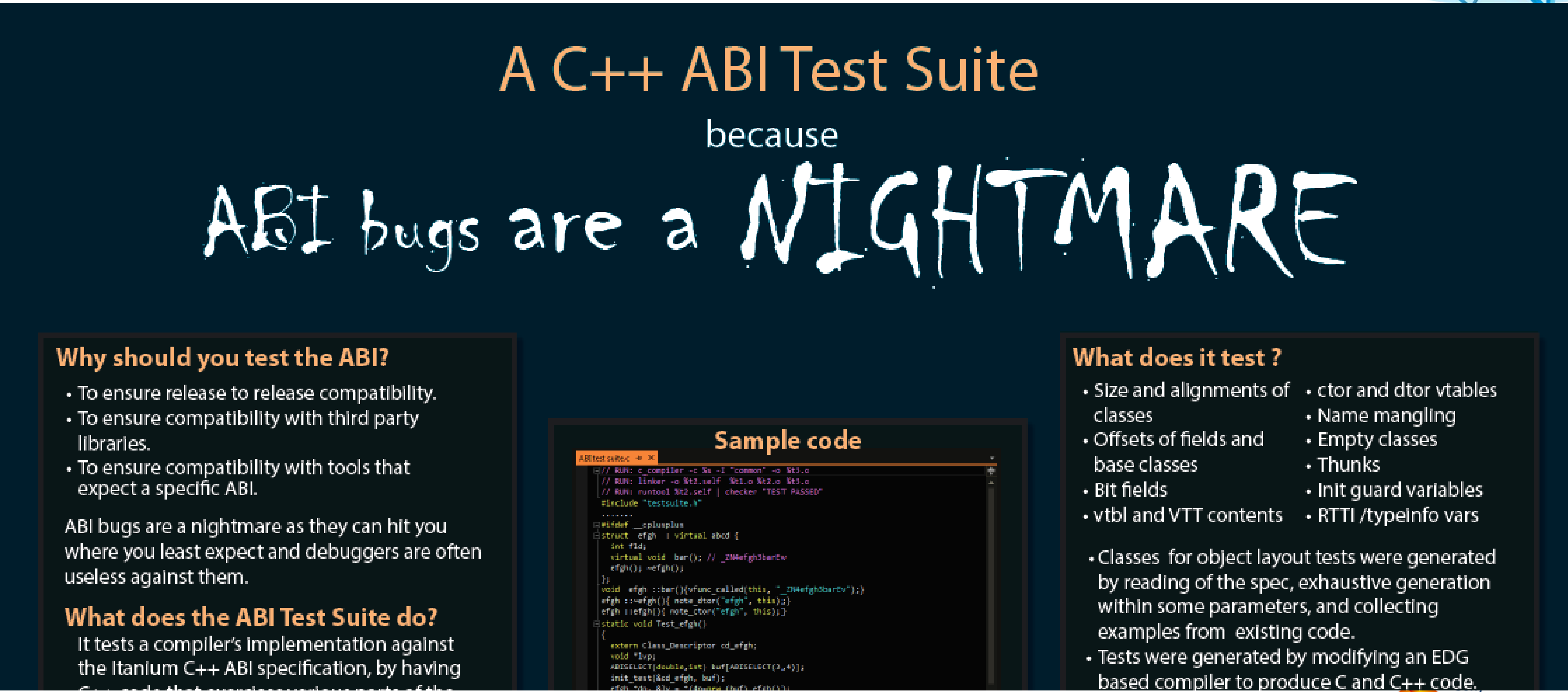
CPU Compiler ABI Overview

The First ABI Bug



The First ABI Bug – FIXED upstream

Oopsie



A C++ ABI Test Suite

because

ABI bugs are a NIGHTMARE

Why should you test the ABI?

- To ensure release to release compatibility.
- To ensure compatibility with third party libraries.
- To ensure compatibility with tools that expect a specific ABI.

ABI bugs are a nightmare as they can hit you where you least expect and debuggers are often useless against them.

What does the ABI Test Suite do?

It tests a compiler's implementation against the Itanium C++ ABI specification, by having

Sample code

ABI test suite.c

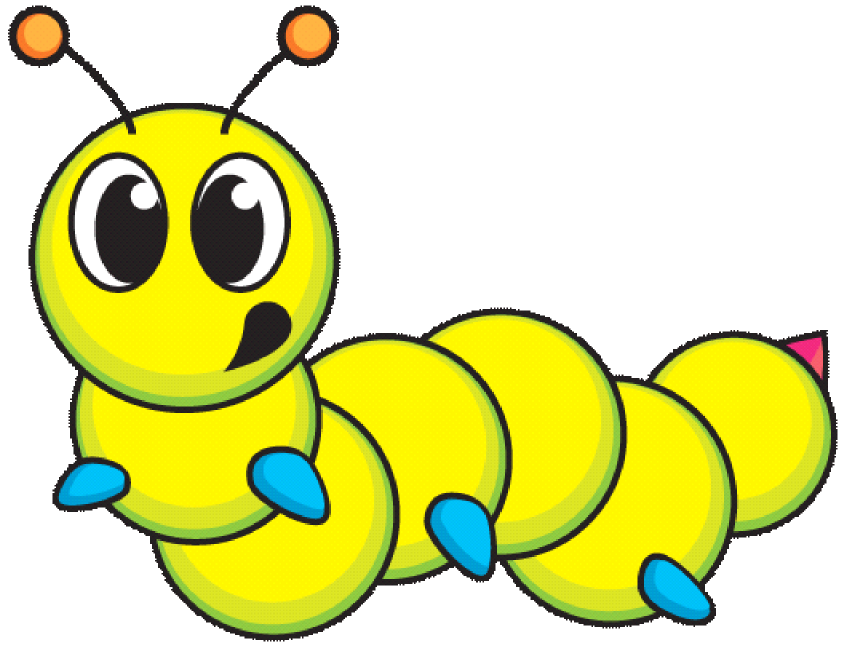
```
// RUN: c_compiler -c %s -I "common" -o %t3.o
// RUN: linker -o %t2.self  %t1.o %t2.o %t3.o
// RUN: runtool %t2.self | checker "TEST PASSED"
#include "testsuite.h"
.......
#ifdef __cplusplus
struct  efgh  : virtual abcd {
  int fld;
  virtual void  bar(); // _ZN4efgh3barEv
  efgh(); ~efgh();
};
void  efgh ::bar(){vfunc_called(this, "_ZN4efgh3barEv");}
efgh ::~efgh(){ note_dtor("efgh", this);}
efgh ::efgh(){ note_ctor("efgh", this);}
static void Test_efgh()
{
  extern Class_Descriptor cd_efgh;
  void *lvp;
  ABISELECT(double,int) buf[ABISELECT(3,4)];
  init_test(&cd_efgh, buf);
```

What does it test ?

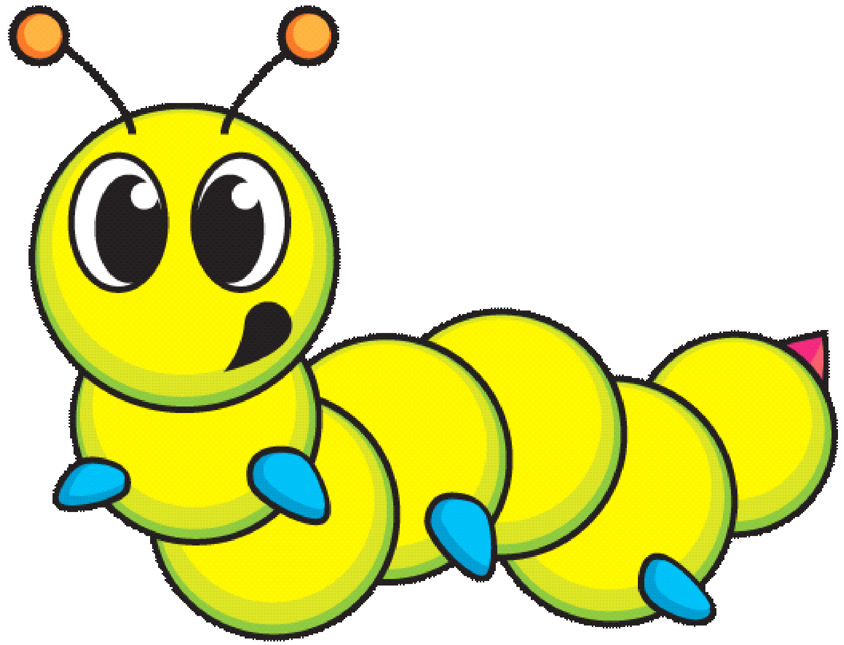
- Size and alignments of classes
- Offsets of fields and base classes
- Bit fields
- vtbl and VTT contents
- ctor and dtor vtables
- Name mangling
- Empty classes
- Thunks
- Init guard variables
- RTTI /typeinfo vars

- Classes for object layout tests were generated by reading of the spec, exhaustive generation within some parameters, and collecting examples from existing code.
- Tests were generated by modifying an EDG based compiler to produce C and C++ code.

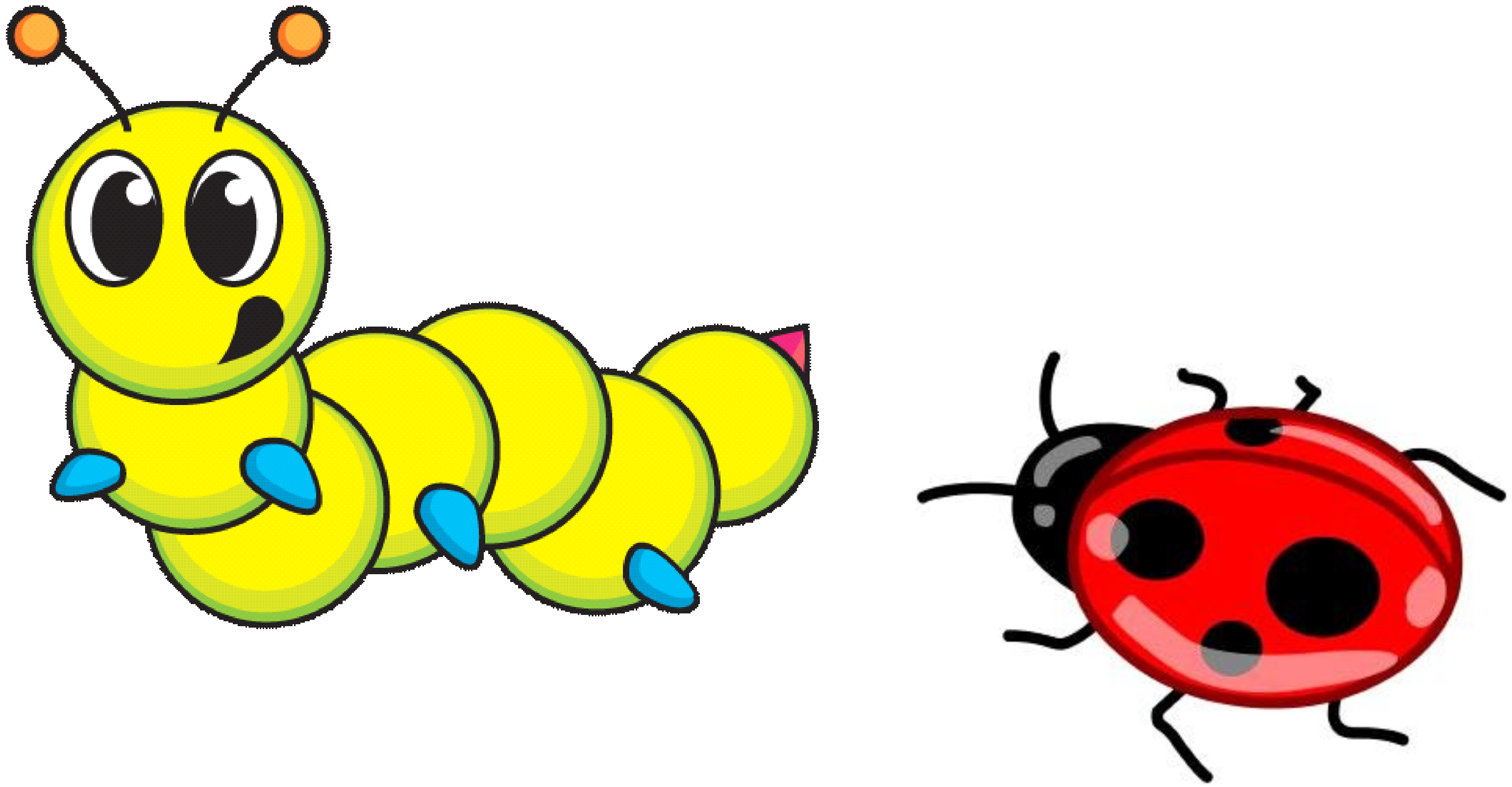
Nice buggie…



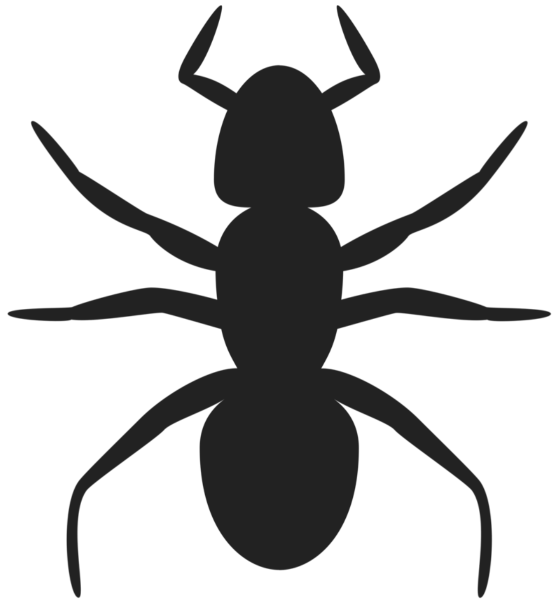
Um… how many are there?

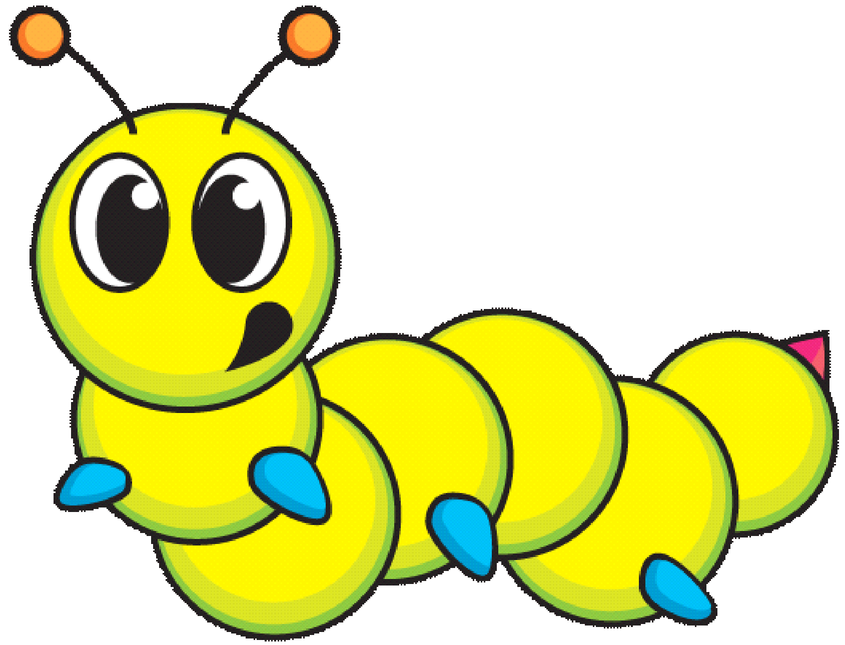


Um… how many are there?

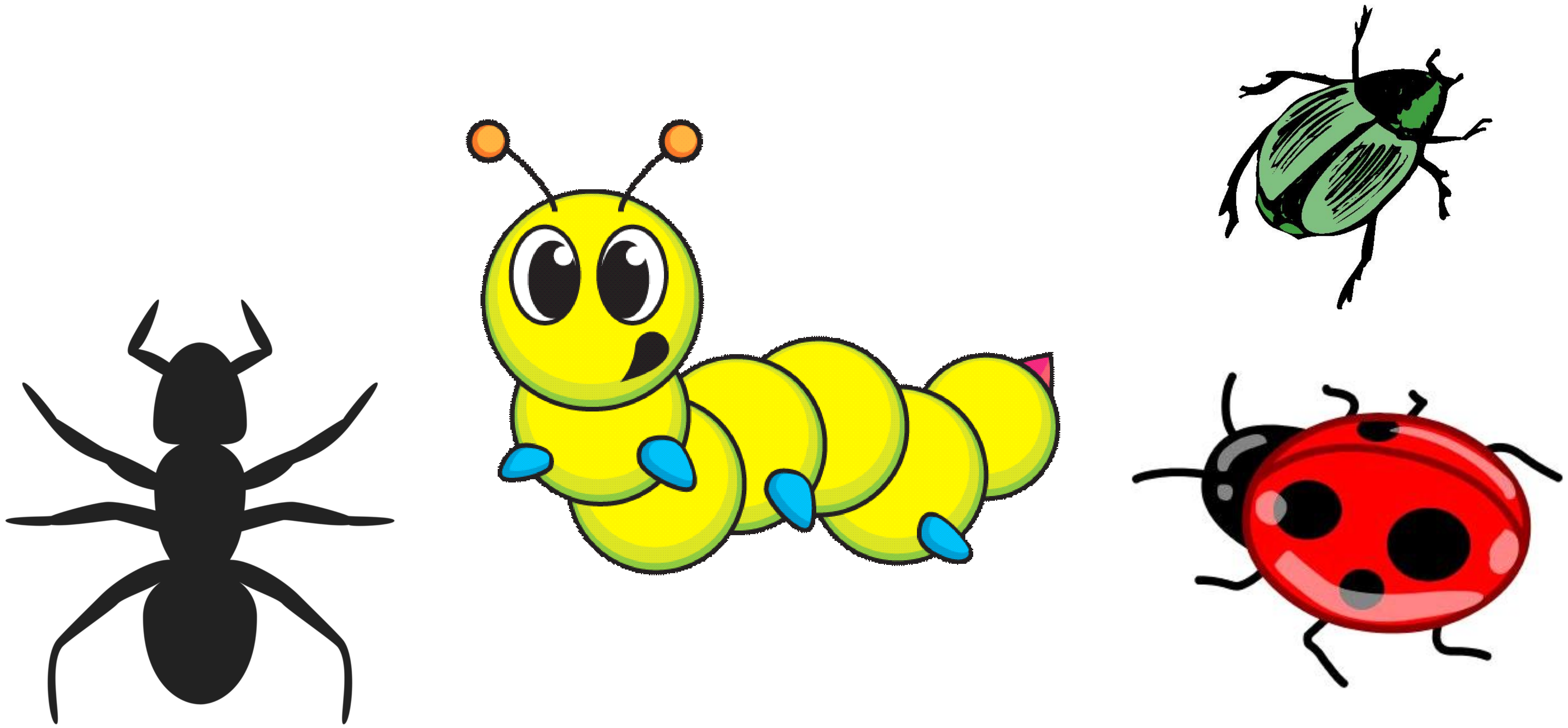


Um... how many are there?

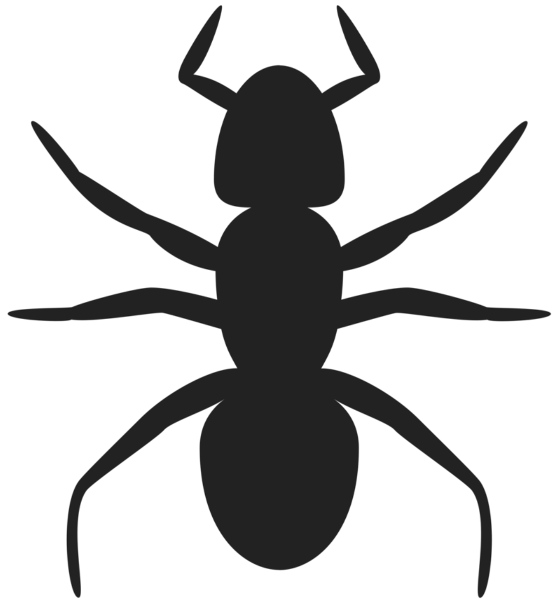


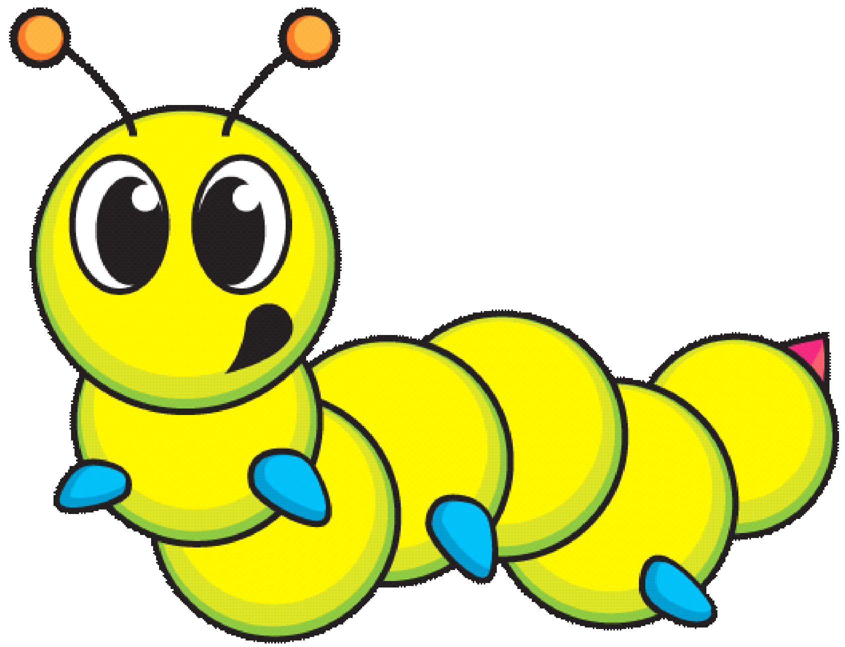


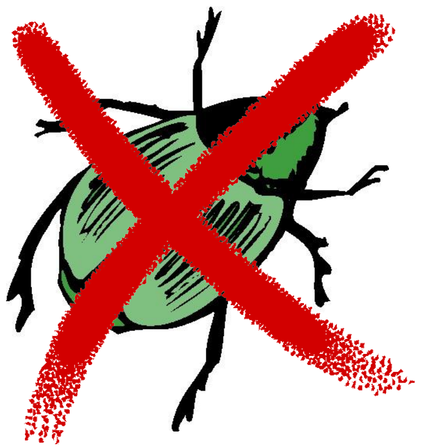
Um… how many are there?

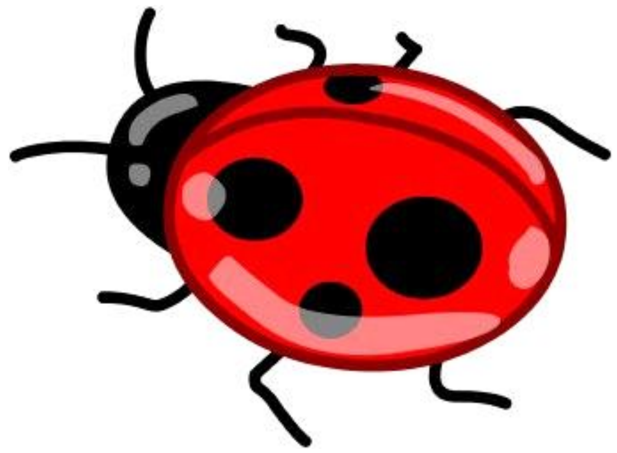


…and how many do we keep?

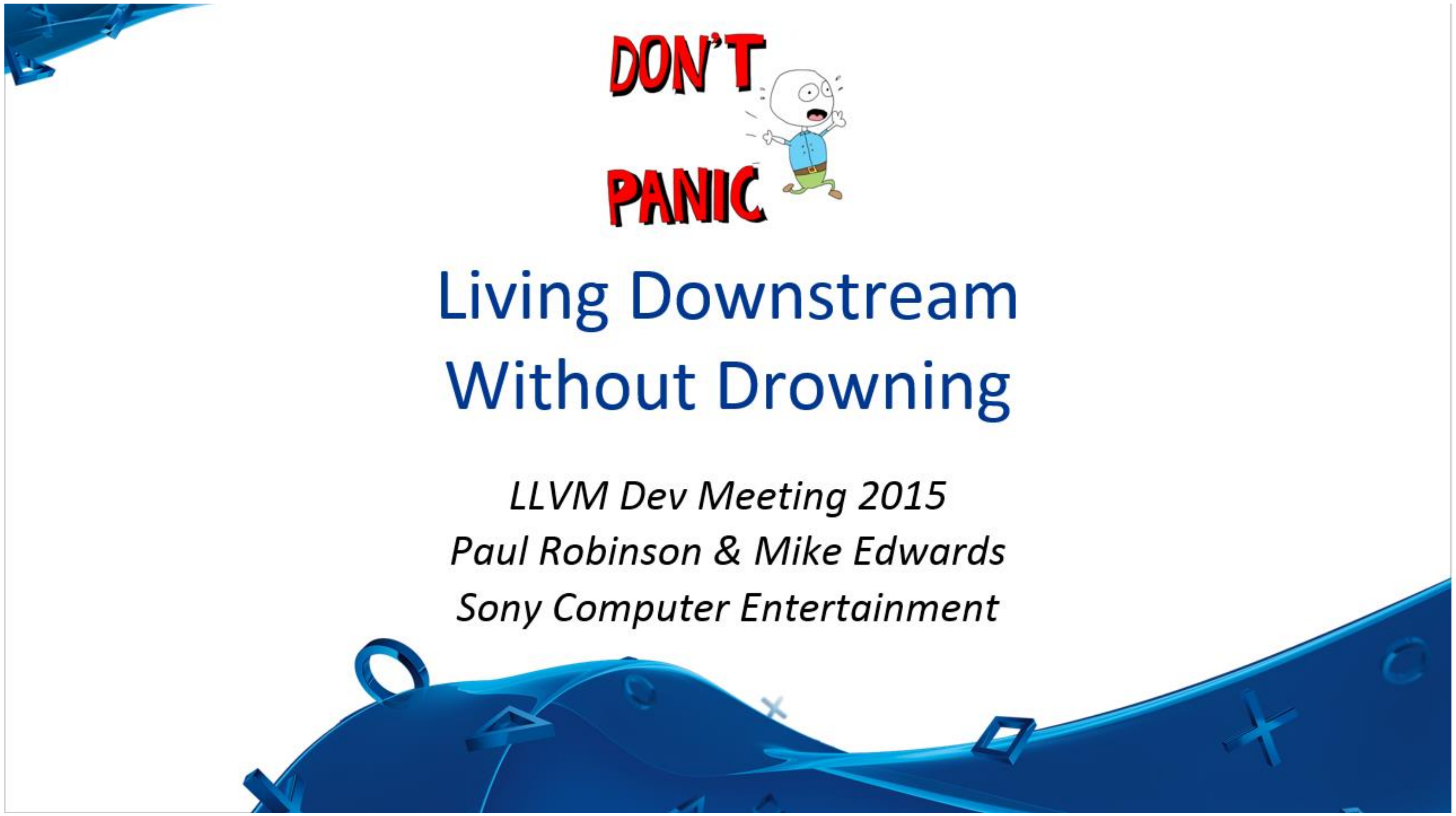








Don’t Panic



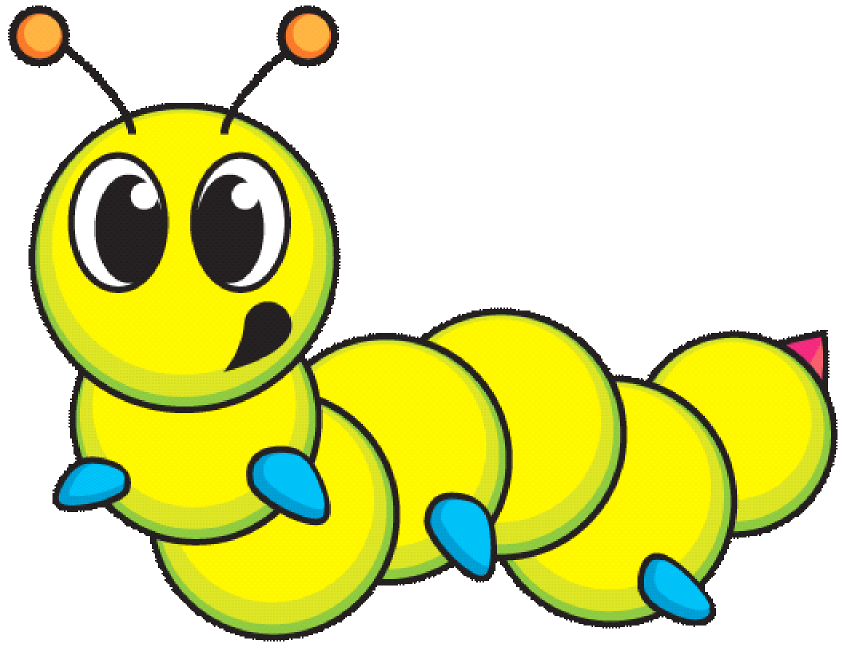
Living Downstream Without Drowning

LLVM Dev Meeting 2015

Paul Robinson & Mike Edwards

Sony Computer Entertainment

That’s a ~~bug~~ feature!



Clang ABI Compatibility Mode

Specify **`-fclang-abi-compat=<version>`**

- Attempts to match the ABI behavior of Clang <version>
- Useful in certain scenarios

This is the WRONG APPROACH for a “closed” platform like PS4!

- With strict backward compatibility requirements
  - Version 1.000 games must run on ALL later system versions
- With no need for compatibility with other compilers
  - There really is only one ABI

PS4 ABI “Mode”

PS4 ABI selected by –target alone

- Places that check Clang ABI compatibility are often the right places to put a PS4 target check
- Plus a small number of places where we just do our own thing

Not all ABI bug fixes are bad!

- Mangling changes in particular are often acceptable
  - We verify our exported symbols don’t depend on them
- These obviously don’t need a PS4 target check

Summary

- Test the heck out of your ABI differences (that you know about!)
    - Compare a “v1” compiler to your latest
- For a closed ecosystem with strict backward compatibility requirements, check Triple not Clang ABI mode
- Keep a close eye on upstream changes that could affect you
    - RecordLayoutBuilder.cpp
    - ItaniumMangle.cpp
    - Any patch with a Clang ABI Mode check
    - Phabricator’s Herald rules are your friend
- When things change, it *might* be okay
    - We’ve allowed some mangling changes that our symbols don’t use