CPU Toolchain Launch Postmortem

Greg Bedwell

x86-64 AMD “Jaguar”
8-core CPU

1.84 TFLOPS AMD Radeon™
based GPU

8GB GDDR5 RAM

http://llvm.org/devmtg/2013-11/

postmortem *noun*

“an analysis or discussion of an event after it is over”

Now that we have successfully launched PlayStation®4 it is a good time to look back on our initial period of development up to that point

http://www.merriam-webster.com/dictionary/postmortem

condensed

First, some history...

SN Systems Ltd. was founded in 1988 to provide development tools for the games industry

1990 *Psy-Q*

16-bit home systems

Psy-Q included a version of GCC that was highly customized for the needs of game developers

1994 *Psy-Q*

PlayStation®

Continued to provide GCC but started researching a proprietary compiler technology – “SNC”

2000 *ProDG*

PlayStation®2

Provided SNC as part of the ProDG suite of tools although GCC was also available

2004 *ProDG*

PSP®

(PlayStation®Portable)

Sony Computer Entertainment Inc. acquired SN Systems in 2005

Provided SNC as part of the ProDG suite of tools although GCC was also available

2006 *ProDG*

PlayStation®3

CPU Compiler is SNC

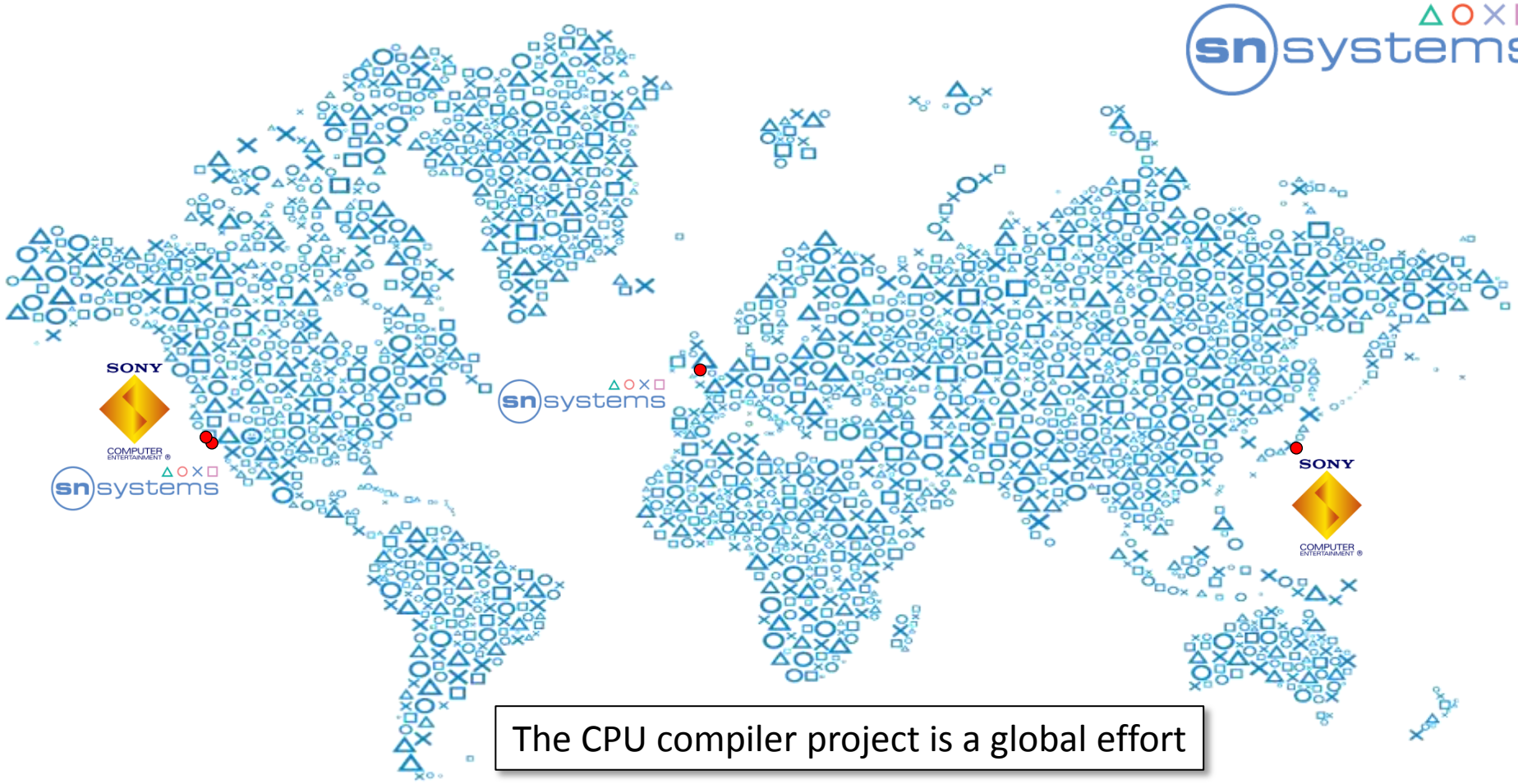
2011

PlayStation®Vita

CPU Compiler is Clang

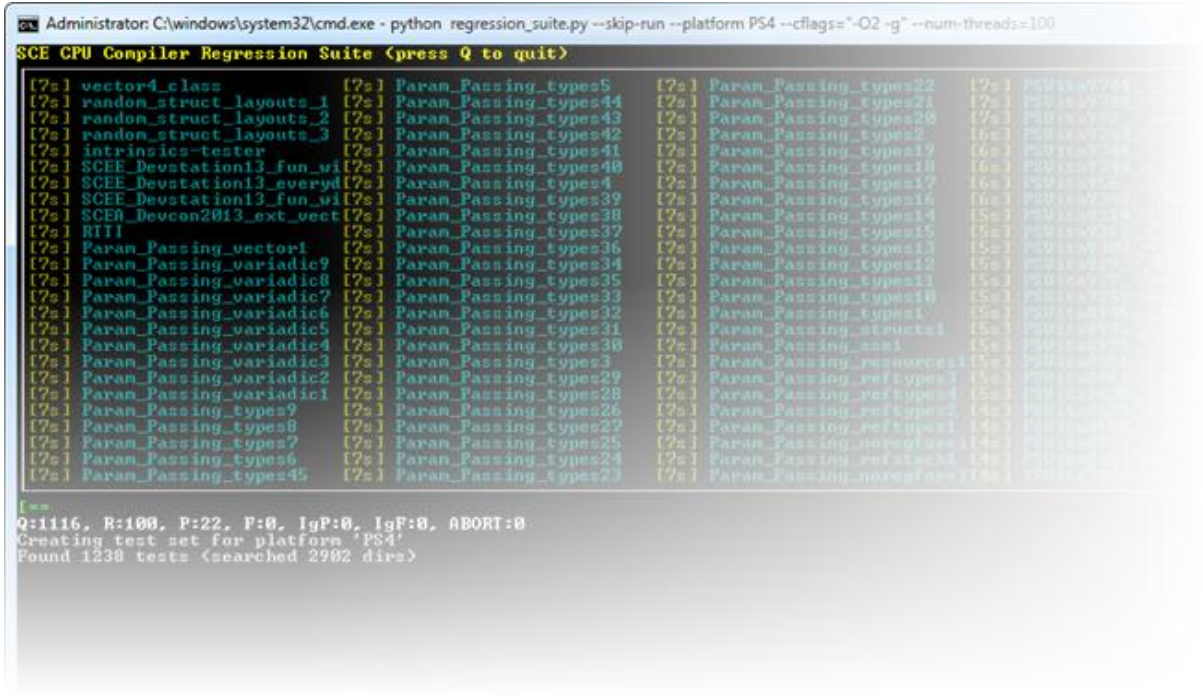
2013

PlayStation®4


sn systems
SONY
COMPUTER ENTERTAINMENT
sn systems
sn systems
SONY
COMPUTER ENTERTAINMENT
The CPU compiler project is a global effort

Builds and build systems

(and test systems)

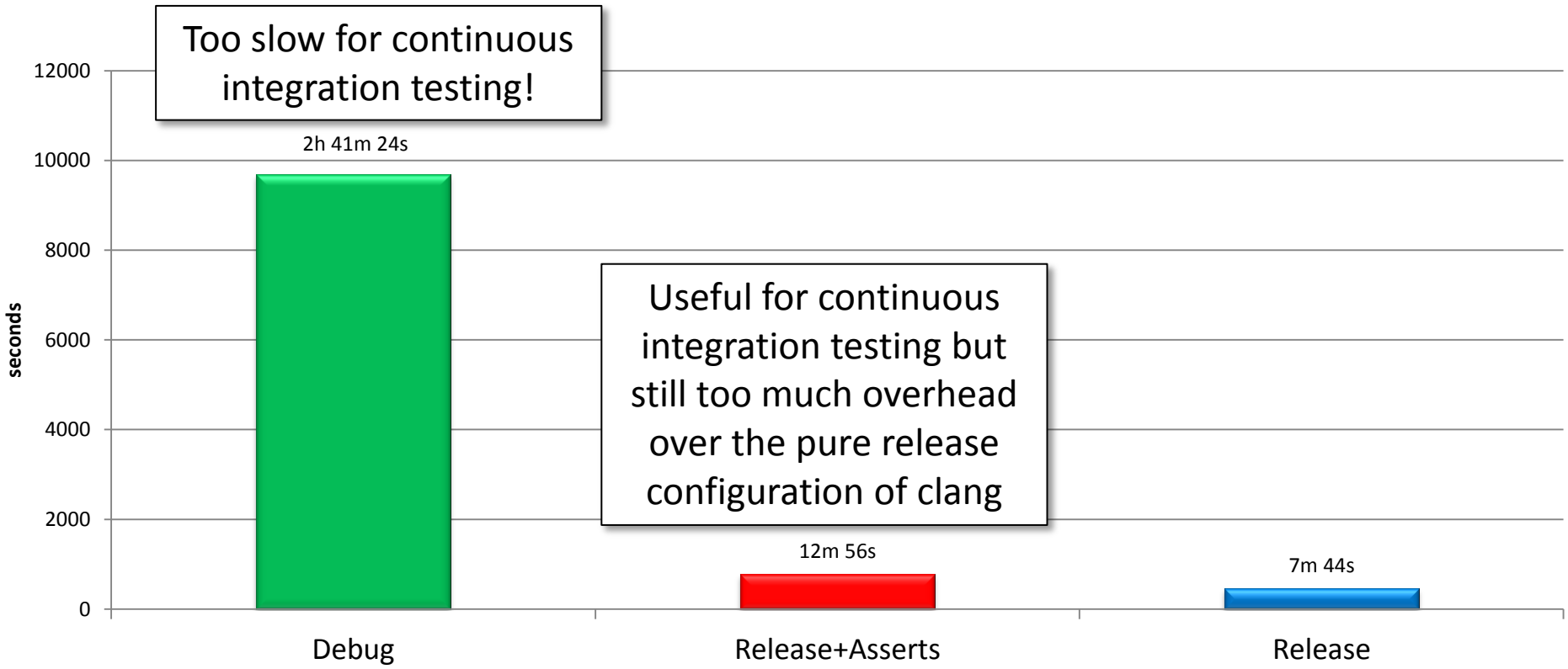


We wanted to make use of all of our pre-existing test suites and test systems, which are shared across all targets

	Optimized	Assertions	Debug Info
Debug	✗	✓	✓
MinSizeRel	✓	✗	✗
Release	✓	✗	✗
RelWithDebInfo	✓	✗	✓

	Optimized	Assertions	Debug Info
Debug	✗	✓	✓
MinSizeRel	✓	✗	✗
Release	✓	✗	✗
RelWithDebInfo	✓	✗	✓

Clang configuration effect on game build time



	Optimized	Assertions	Debug Info
Debug	✗	✓	✓
Checking	✓	✓	✗
Release	✓	✗	✗

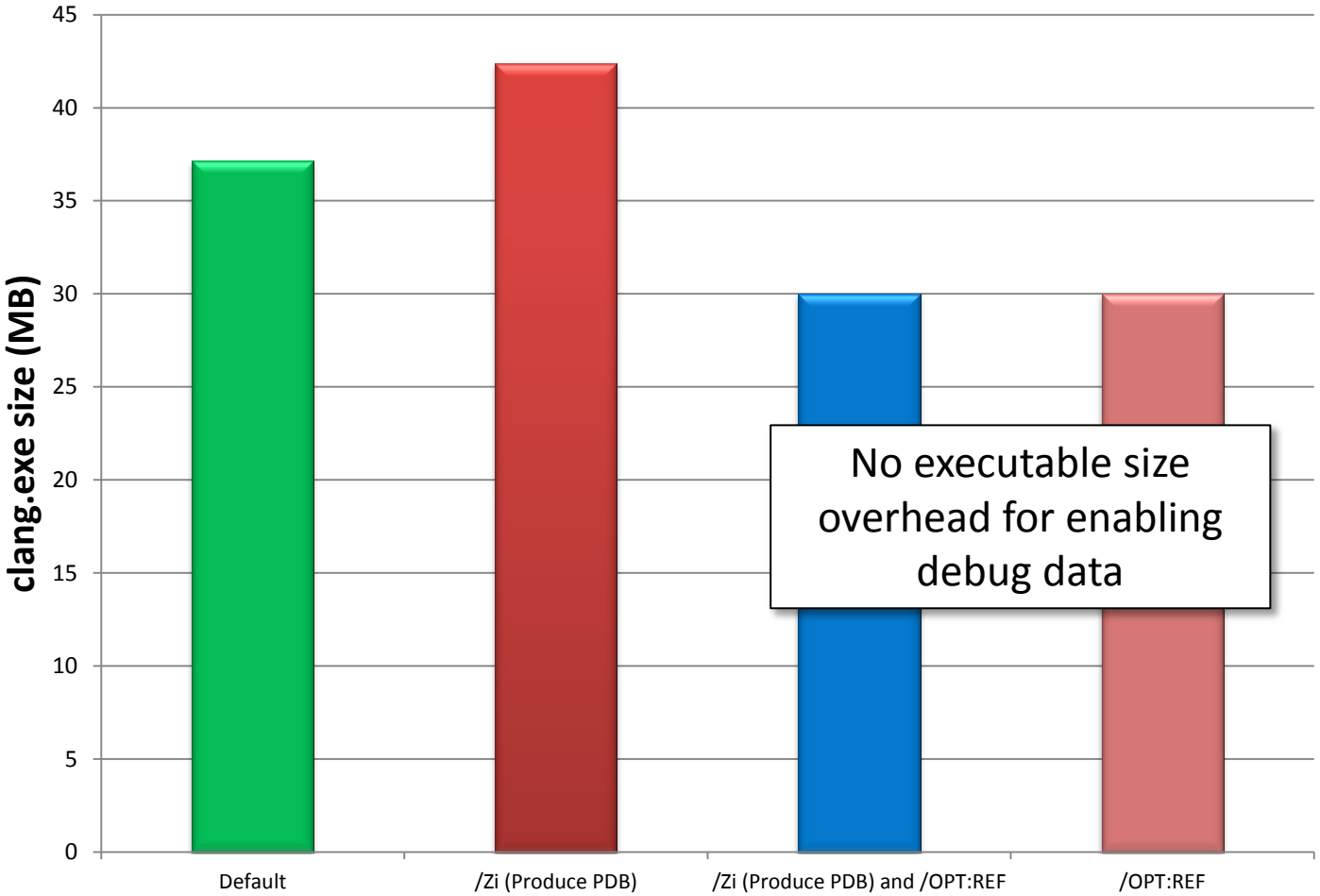
The same set of build configurations as we use for SNC

“Debug” for debugging

“Release” for releasing

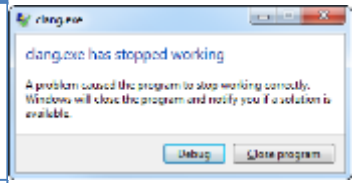
“Checking” (Release+Asserts) for continuous integration testing

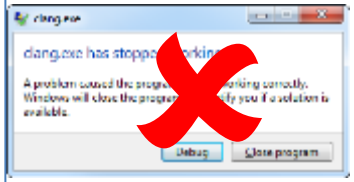
	Optimized	Assertions	Debug Info
Debug	✗	✓	✓
Checking	✓	✓	✗
Release	✓	✗	✗

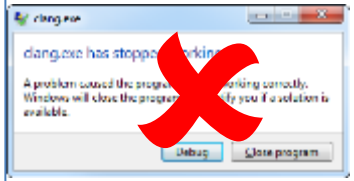

clang.exe size (MB)
45
40
35
30
25
20
15
10
5
0
Default
/Zi (Produce PDB)
/Zi (Produce PDB) and /OPT:REF
/OPT:REF
No executable size overhead for enabling debug data

	Optimized	Assertions	Debug Info
Debug	✗	✓	✓
Checking	✓	✓	✓
Release	✓	✗	✓

	Optimized	Assertions	Debug Info
Debug	✗	✓	✓
Checking	✓	✓	✓
Release	✓	✗	✓







Suppress Windows crash dialog box for Checking and Release builds

Improving test coverage

Things we know

Game code is usually

BIG

For a new platform, the amount of code that exists is

small

Things we know

Users value

CORRECTNESS

over all else

*Why write **tests***

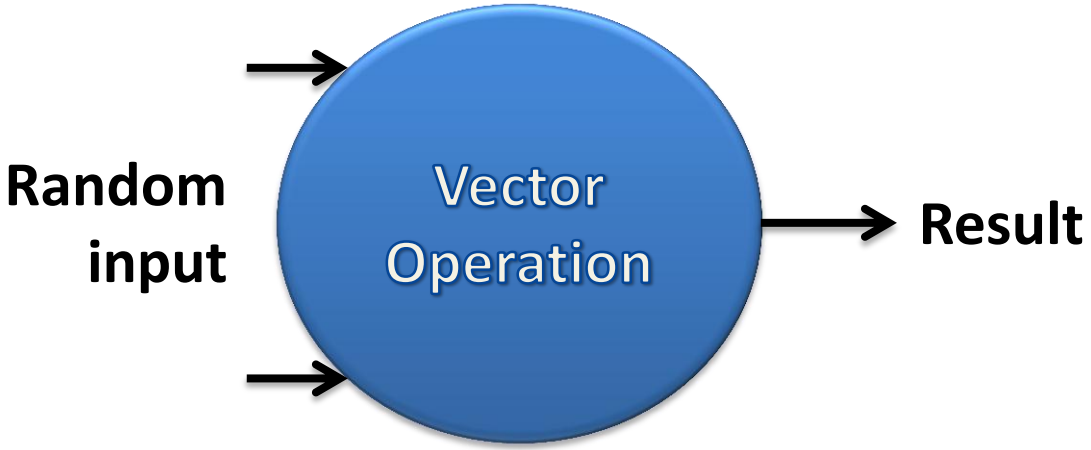
*when I can write a **test generator**?*

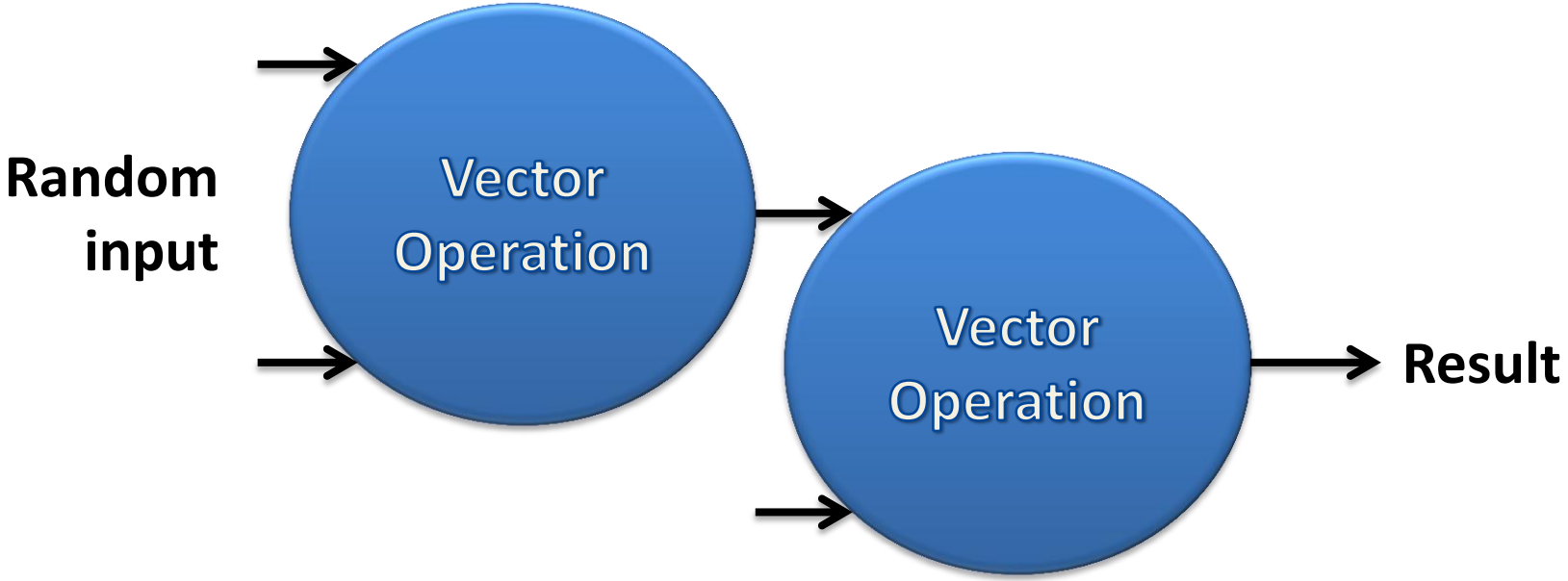
Why write ***tests***

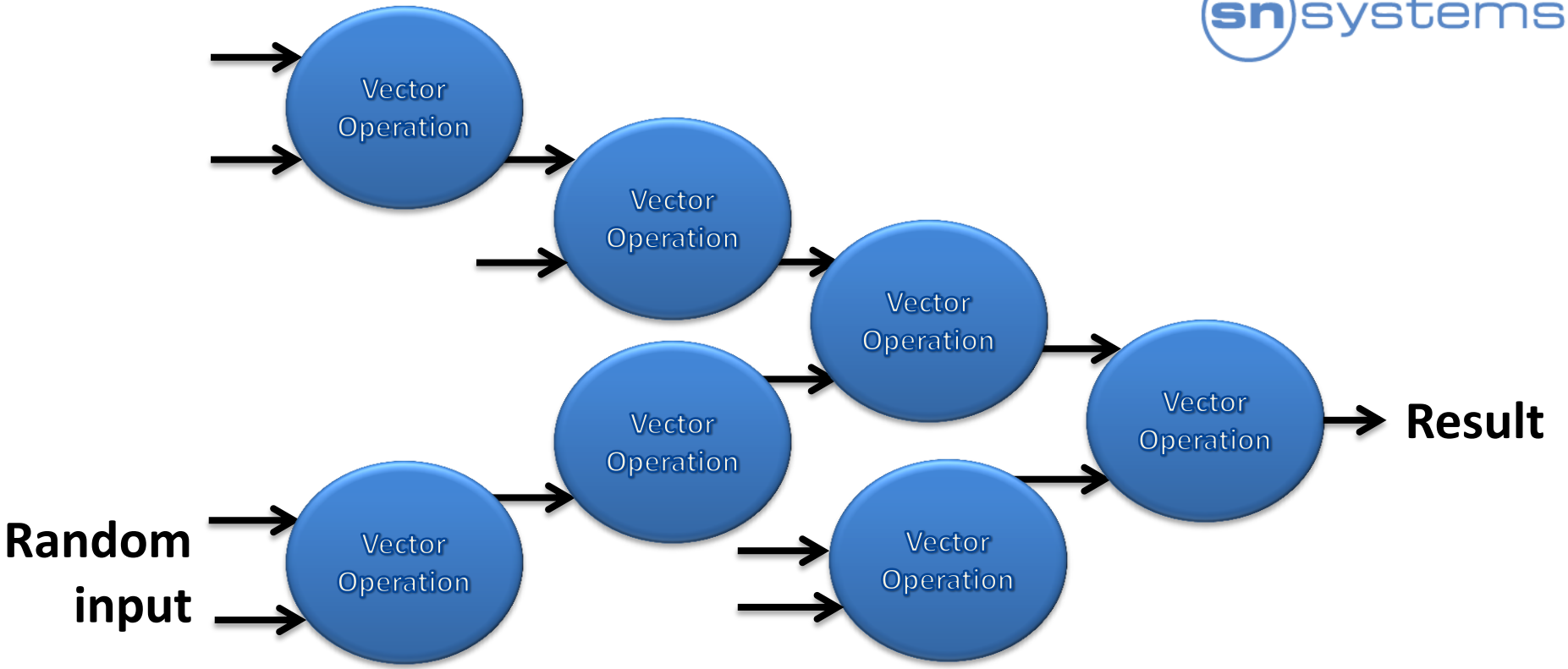
when I ~~can write~~ a ***test generator****?*

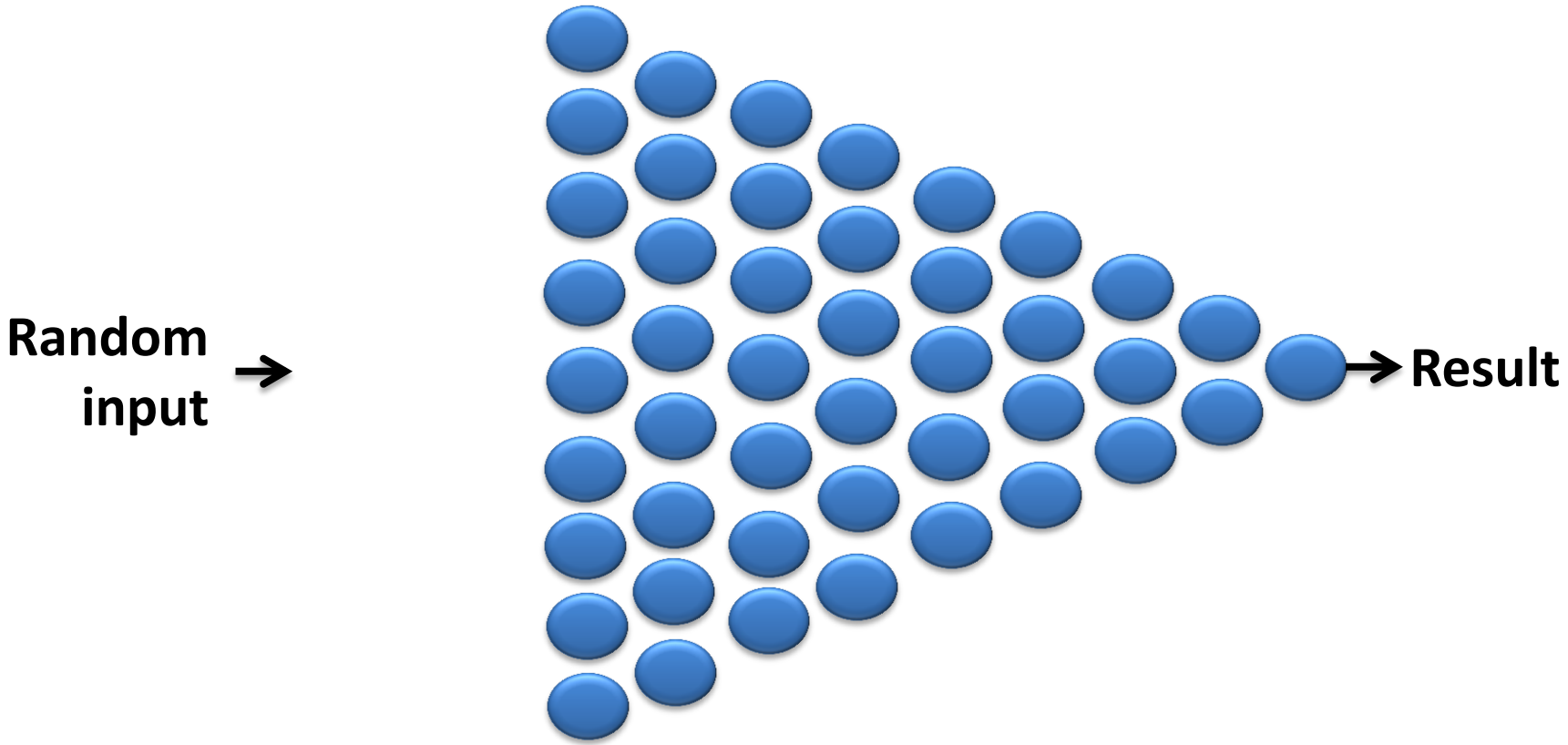
already have

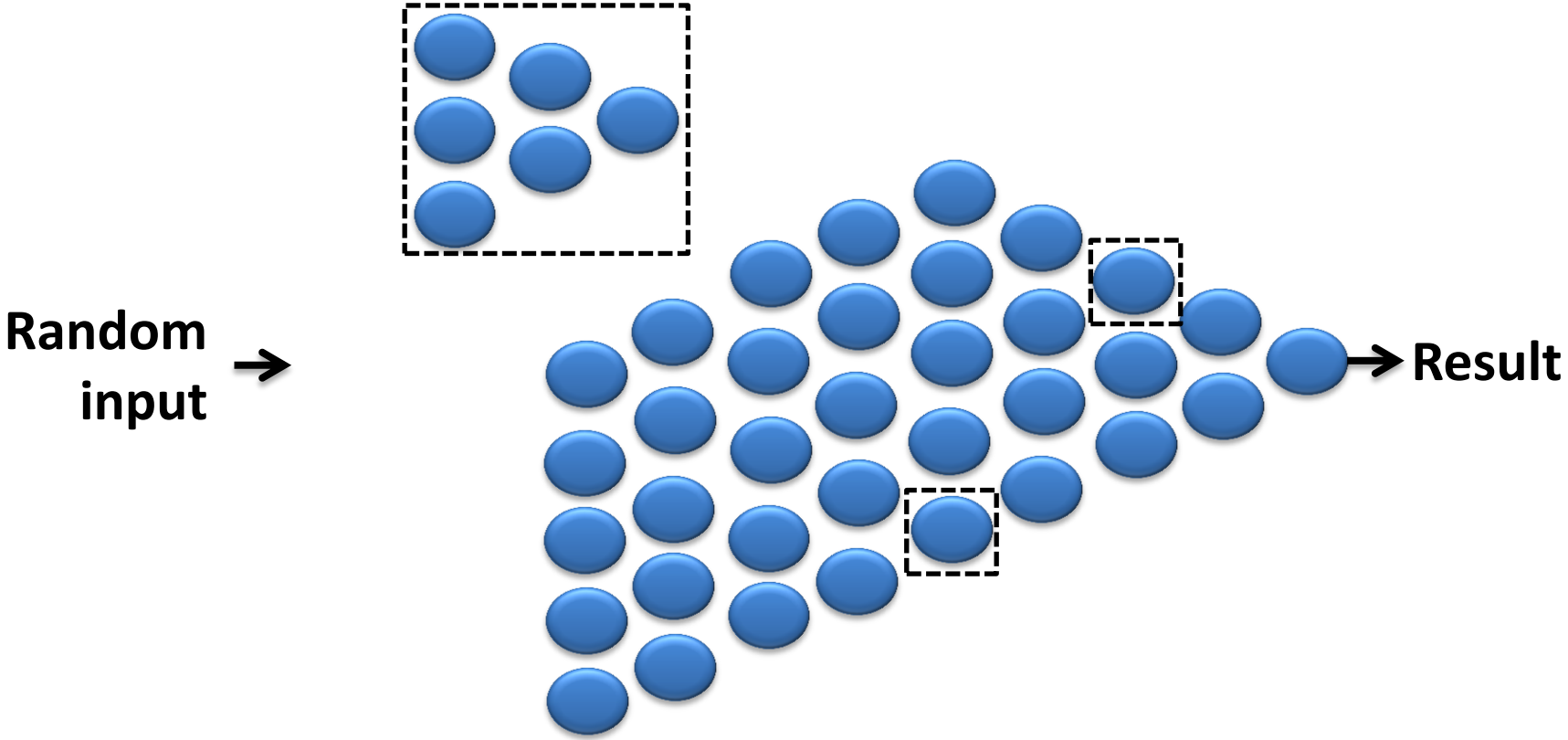
Generate random C++ tests using SIMD language extensions and intrinsics to increase test coverage

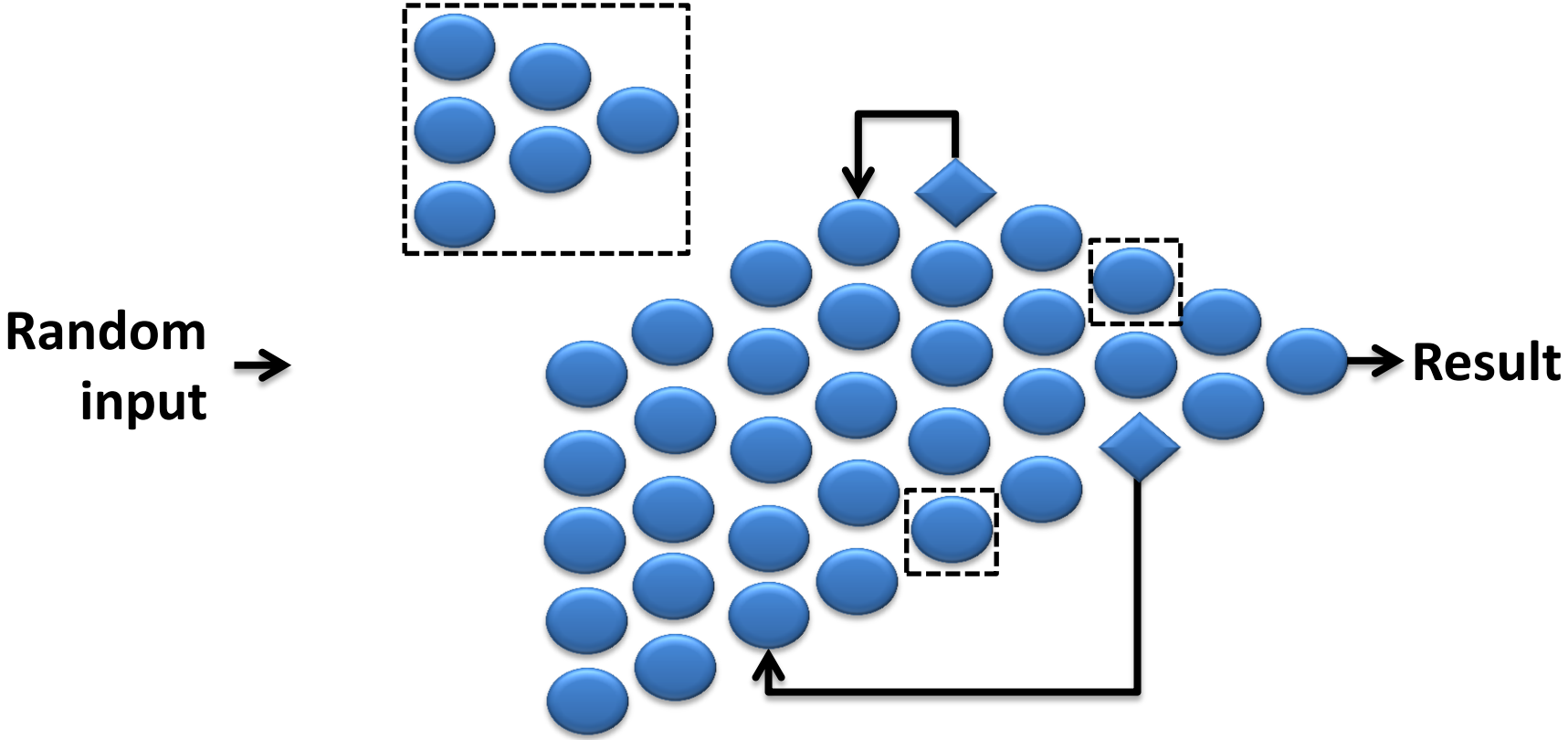



Random input
Vector Operation
Vector Operation
Result


Vector Operation
Vector Operation
Vector Operation
Vector Operation
Vector Operation
Vector Operation
Vector Operation
Result
Random input


Random input
Result


Random input
Result


Random
input
Result

Undefined behaviour makes runtime-behaviour random testing hard!

Solution: Use a ‘safe’ wrapper to make all undefined behaviour defined for the purpose of the test

Reducing optimization bugs

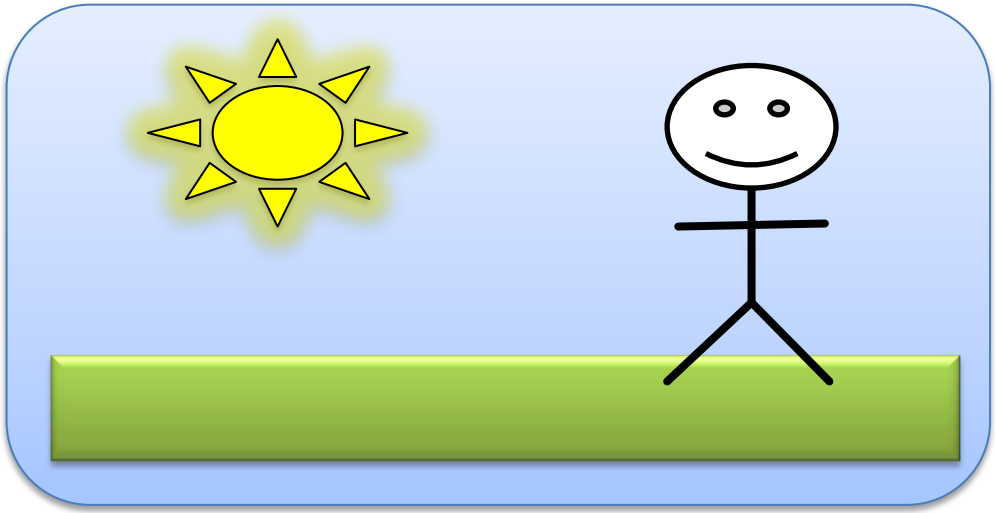
bugpoint

Windows

Clang integration

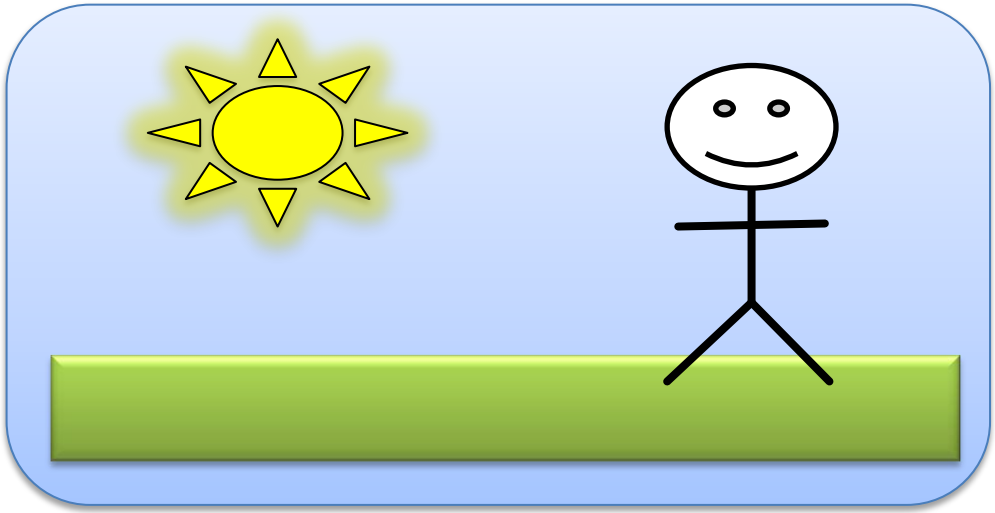
An alternative approach

SNC's max_opts

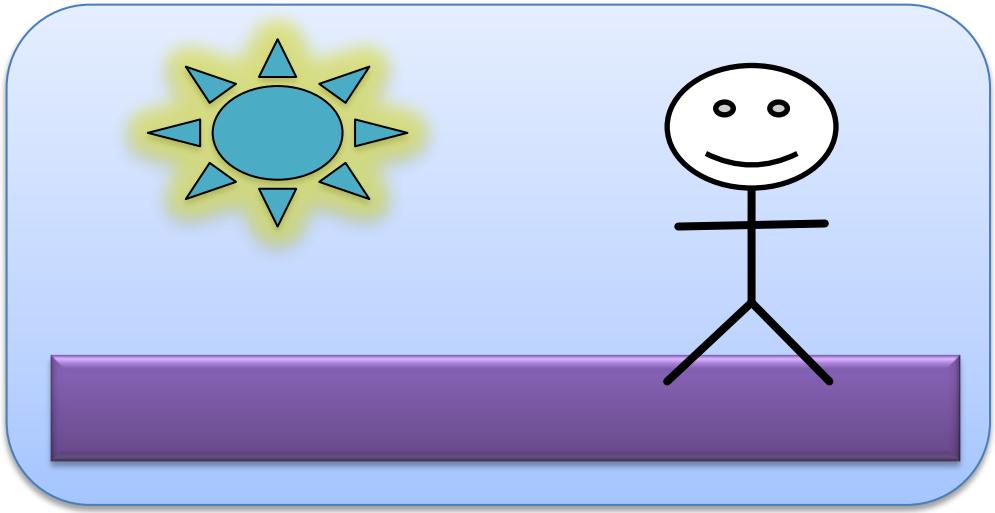


*Not representative of actual visuals

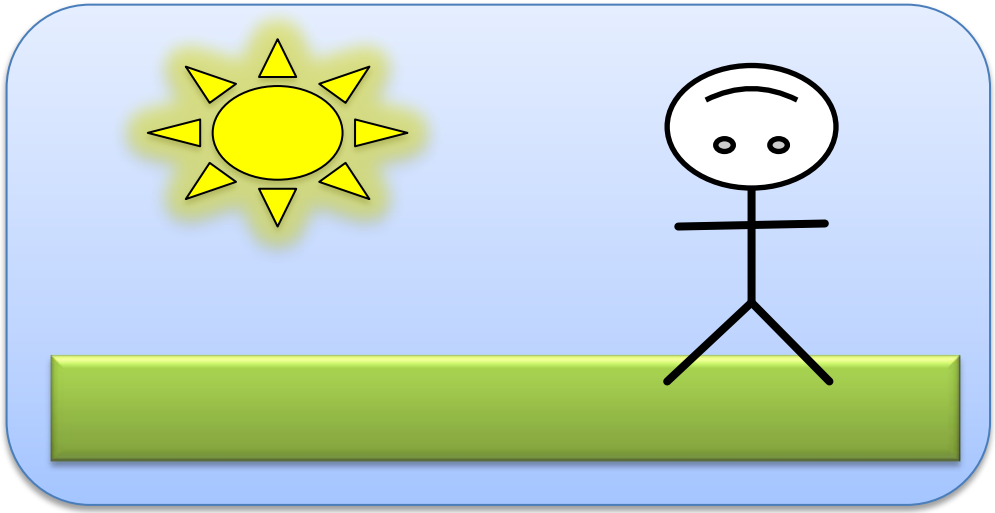
SNC's max_opts



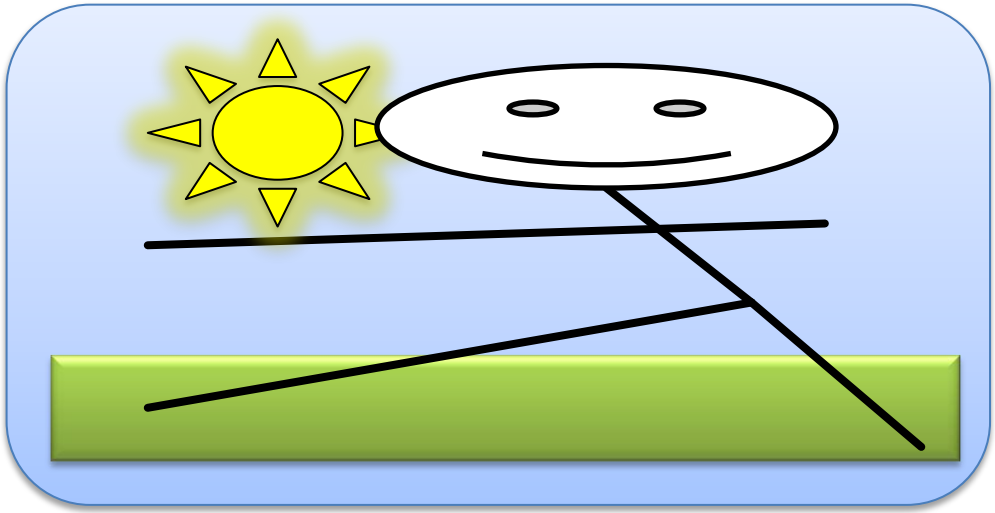
SNC's max_opts



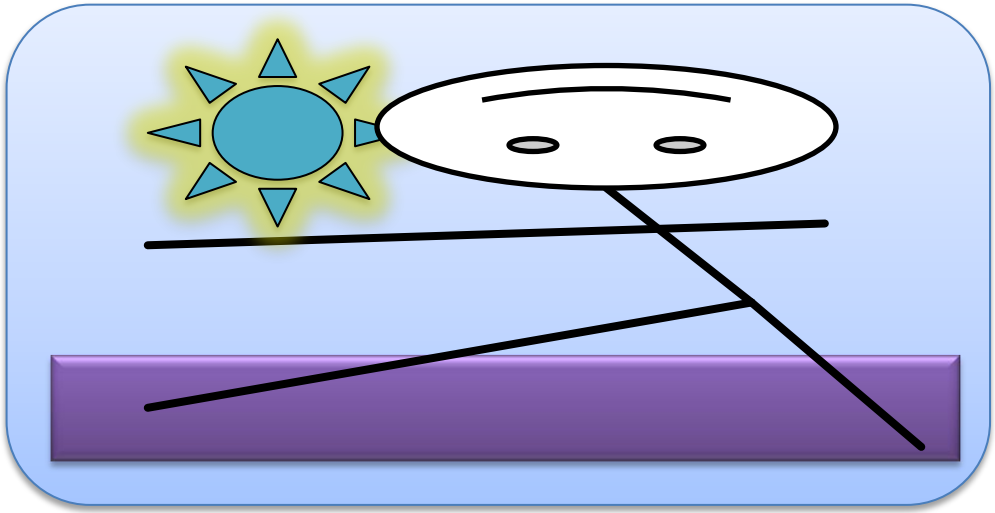
SNC's max_opts



SNC's max_opts



SNC's max_opts



SNC's max_opts

SNC optimizer

SSA Form

Rule Based

Every transformation is guarded by a specific check

SNC keeps an internal counter of the number of transformations it performs

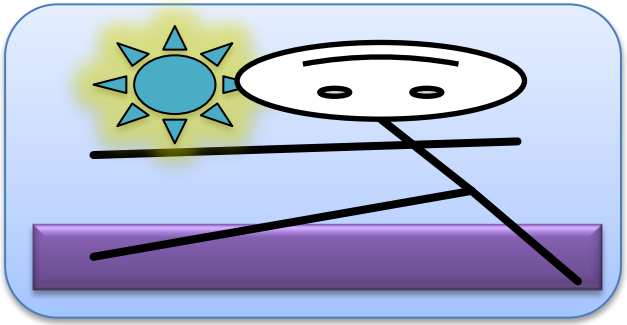
```
if (/* conditions match */ &&
    Opt_Enabled(permute_converted_to_opscalartovector))
{
  Trace_Opt(permute_converted_to_opscalartovector);
  /* Perform optimization */
}
```

Allows the user to specify a limit on the command line after which no further transformations can be performed

SNC's max_opts

"Autochop" harness

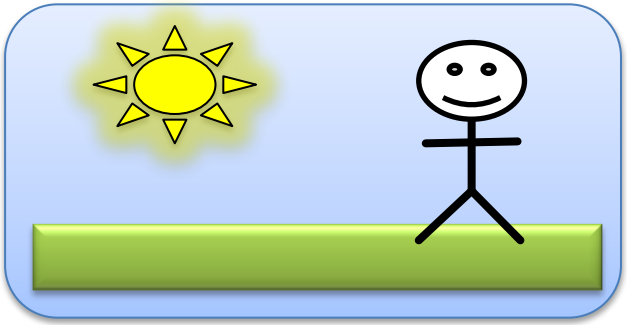
-O2



SNC's max_opts

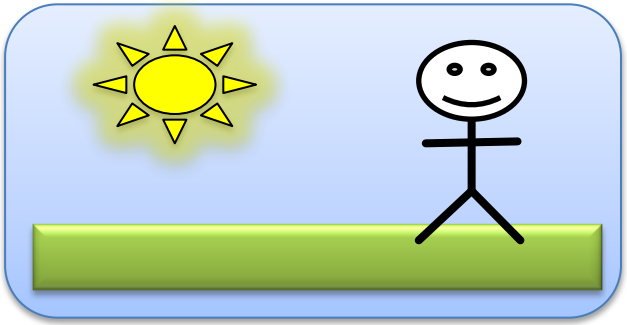
"Autochop" harness

-O0



SNC's max_opts

"Autochop" harness

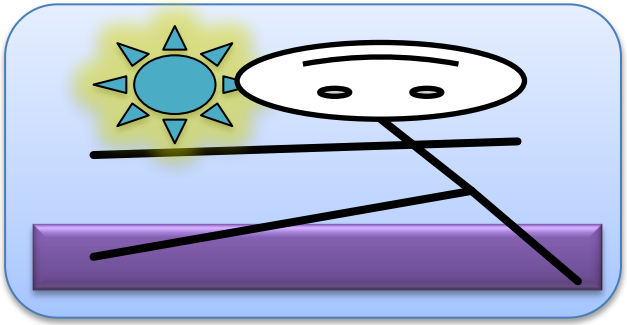


SNC's max_opts

"Autochop" harness

-O0

-O2

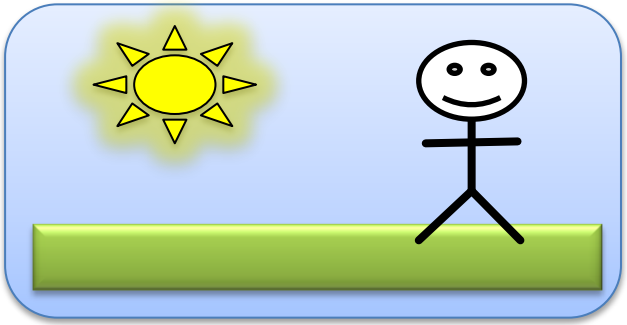


SNC's max_opts

"Autochop" harness

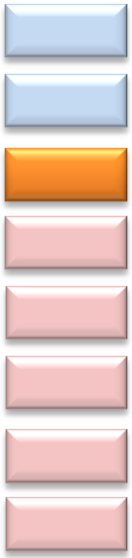
-O0

-O2



SNC's max_opts

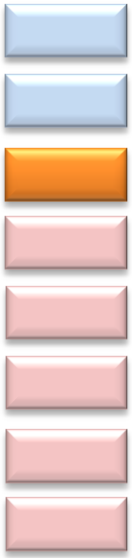
"Autochop" harness



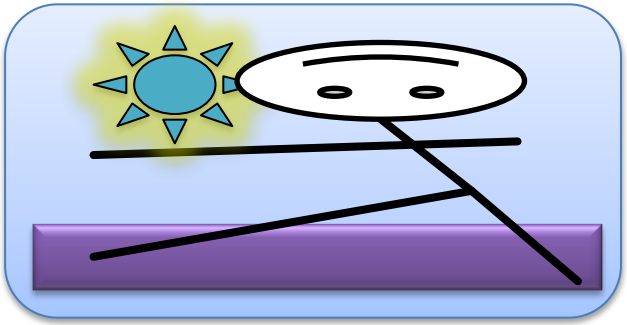
We've found the game source file with the bad transformation

SNC's max_opts

"Autochop" harness

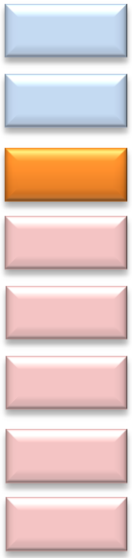


-Xmax_opts=2048

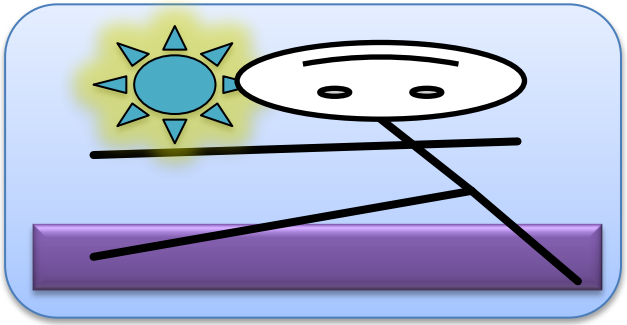


SNC's max_opts

"Autochop" harness

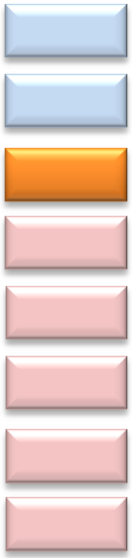


-Xmax_opts=1024

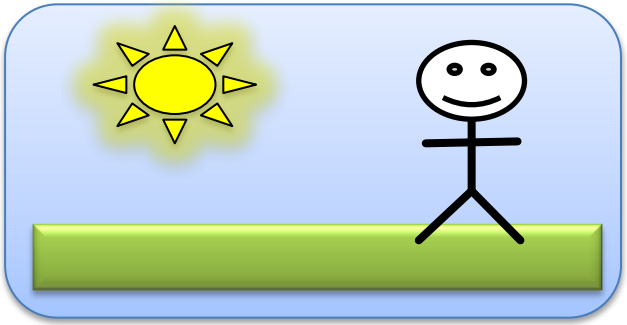


SNC's max_opts

"Autochop" harness

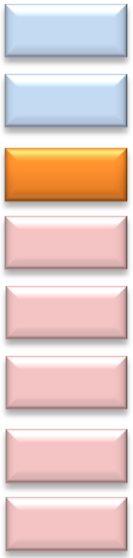


-Xmax_opts=512

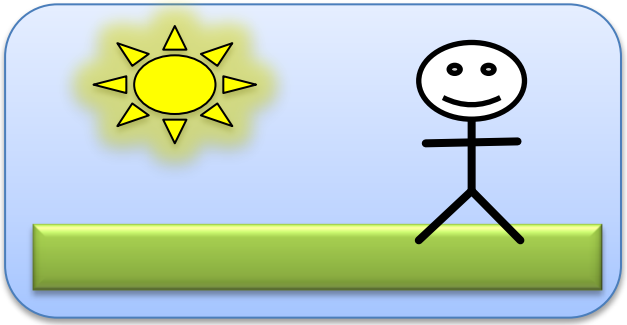


SNC's max_opts

"Autochop" harness

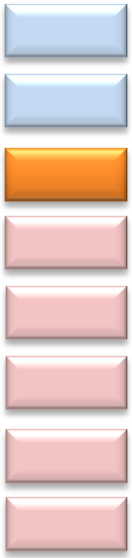


-Xmax_opts=988

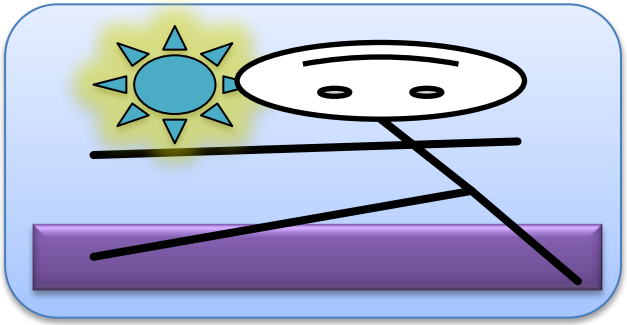


SNC's max_opts

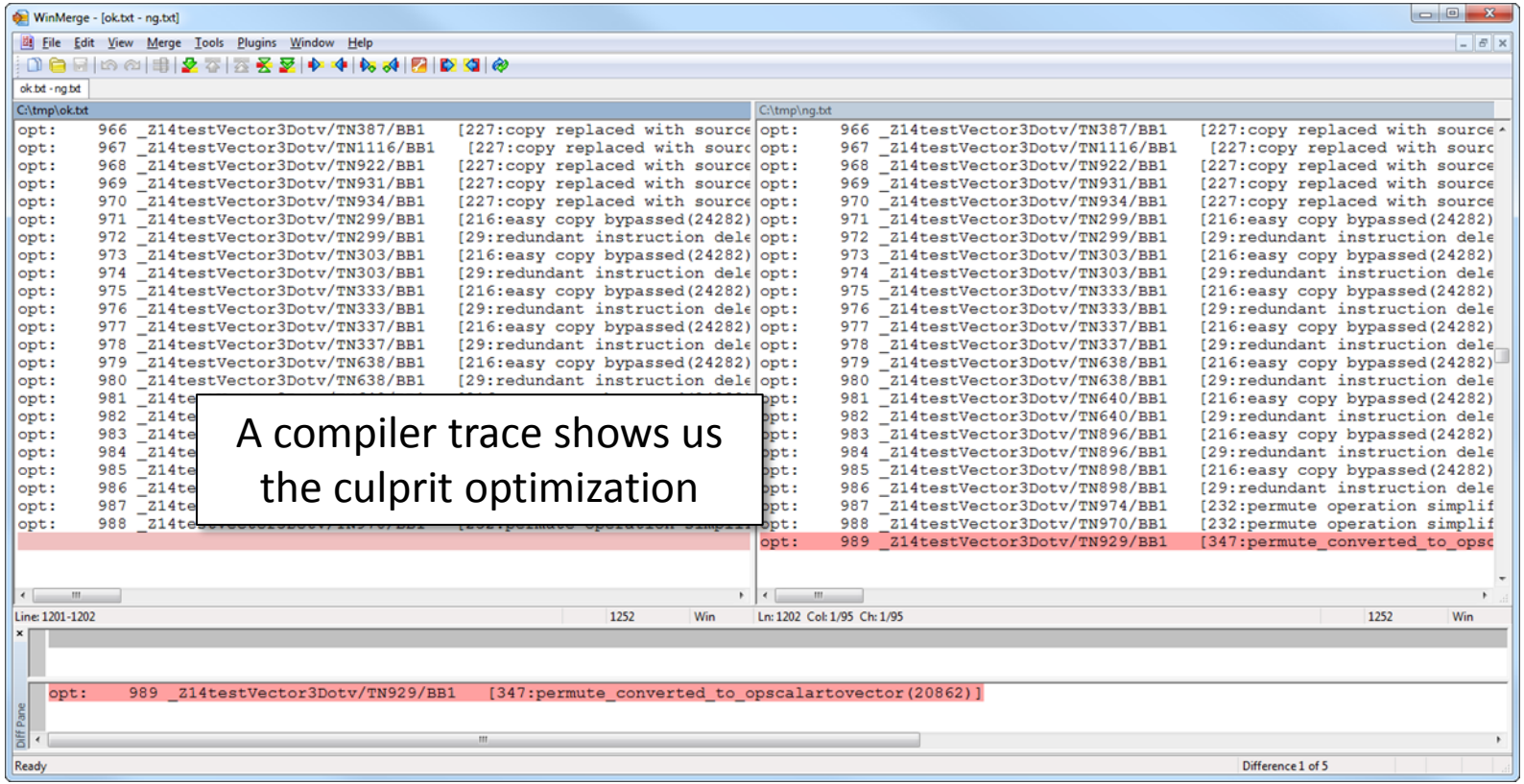
"Autochop" harness



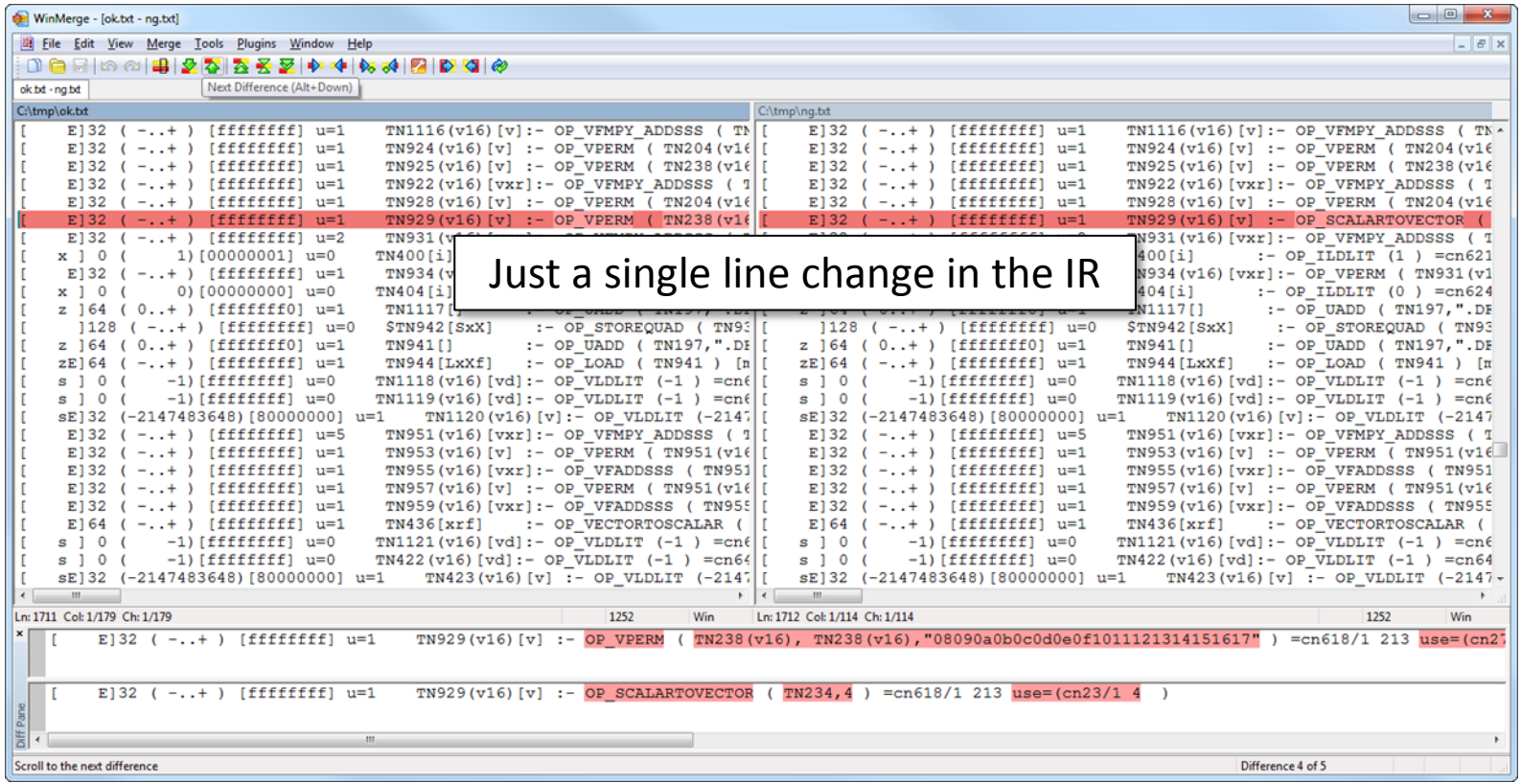
-Xmax_opts=989



Now we've found the specific transformation causing the miscompile


WinMerge - [ok.txt - ng.txt]
File Edit View Merge Tools Plugins Window Help
ok.txt - ng.txt
C:\tmp\ok.txt
C:\tmp\ng.txt
opt: 966 _Z14testVector3Dotv/TN387/BB1 [227:copy replaced with source
opt: 967 _Z14testVector3Dotv/TN1116/BB1 [227:copy replaced with sourc
opt: 968 _Z14testVector3Dotv/TN922/BB1 [227:copy replaced with source
opt: 969 _Z14testVector3Dotv/TN931/BB1 [227:copy replaced with source
opt: 970 _Z14testVector3Dotv/TN934/BB1 [227:copy replaced with source
opt: 971 _Z14testVector3Dotv/TN299/BB1 [216:easy copy bypassed(24282)
opt: 972 _Z14testVector3Dotv/TN299/BB1 [29:redundant instruction dele
opt: 973 _Z14testVector3Dotv/TN303/BB1 [216:easy copy bypassed(24282)
opt: 974 _Z14testVector3Dotv/TN303/BB1 [29:redundant instruction dele
opt: 975 _Z14testVector3Dotv/TN333/BB1 [216:easy copy bypassed(24282)
opt: 976 _Z14testVector3Dotv/TN333/BB1 [29:redundant instruction dele
opt: 977 _Z14testVector3Dotv/TN337/BB1 [216:easy copy bypassed(24282)
opt: 978 _Z14testVector3Dotv/TN337/BB1 [29:redundant instruction dele
opt: 979 _Z14testVector3Dotv/TN638/BB1 [216:easy copy bypassed(24282)
opt: 980 _Z14testVector3Dotv/TN638/BB1 [29:redundant instruction dele
opt: 981 _Z14testVector3Dotv/TN640/BB1 [216:easy copy bypassed(24282)
opt: 982 _Z14testVector3Dotv/TN640/BB1 [29:redundant instruction dele
opt: 983 _Z14testVector3Dotv/TN896/BB1 [216:easy copy bypassed(24282)
opt: 984 _Z14testVector3Dotv/TN896/BB1 [29:redundant instruction dele
opt: 985 _Z14testVector3Dotv/TN898/BB1 [216:easy copy bypassed(24282)
opt: 986 _Z14testVector3Dotv/TN898/BB1 [29:redundant instruction dele
opt: 987 _Z14testVector3Dotv/TN974/BB1 [232:permute operation simplif
opt: 988 _Z14testVector3Dotv/TN970/BB1 [232:permute operation simplif
opt: 989 _Z14testVector3Dotv/TN929/BB1 [347:permute_converted_to_opsc
A compiler trace shows us the culprit optimization
Line: 1201-1202 1252 Win
Ln: 1202 Col: 1/95 Ch: 1/95 1252 Win
Diff Pane
opt: 989 _Z14testVector3Dotv/TN929/BB1 [347:permute_converted_to_opscalartovector(20862)]
Ready
Difference 1 of 5

Narrowed down to a single difference in IR



SNC's max_opts

A question for the community:

Would this work in LLVM/Clang?

(even if just at pass level)

The release process

Docs

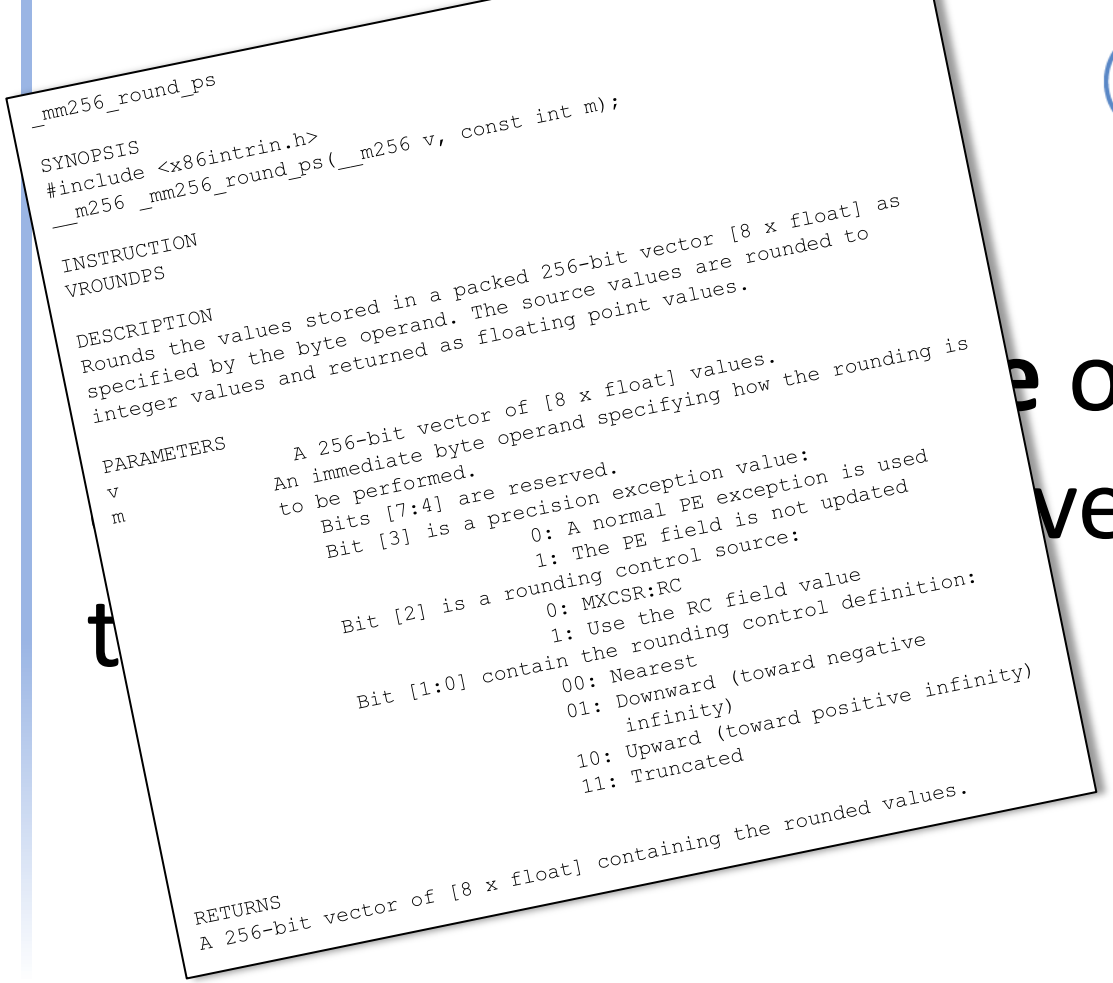
End-user documentation is lacking

Release notes aimed at Clang/LLVM developers, not users

Docs

We plan to **contribute** our documentation improvements to the **community**

Docs



```
_mm256_round_ps

SYNOPSIS
#include <x86intrin.h>
__m256 _mm256_round_ps(__m256 v, const int m);

INSTRUCTION
VROUNDPS

DESCRIPTION
Rounds the values stored in a packed 256-bit vector [8 x float] as
specified by the byte operand. The source values are rounded to
integer values and returned as floating point values.

PARAMETERS
v          A 256-bit vector of [8 x float] values.
m          An immediate byte operand specifying how the rounding is
           to be performed.
             Bits [7:4] are reserved.
             Bit [3] is a precision exception value:
               0: A normal PE exception is used
               1: The PE field is not updated
             Bit [2] is a rounding control source:
               0: MXCSR:RC
               1: Use the RC field value
             Bit [1:0] contain the rounding control definition:
               00: Nearest
               01: Downward (toward negative
                   infinity)
               10: Upward (toward positive infinity)
               11: Truncated

RETURNS
A 256-bit vector of [8 x float] containing the rounded values.
```

t e our

vements

Forward compatibility

ABI

v3.x

v3.x

System libraries

ABI

v3.x

v4.x

System libraries

ABI

v3.x

v5.x

System libraries

ABI

v3.x

v6.x

System libraries

ABI

v3.x

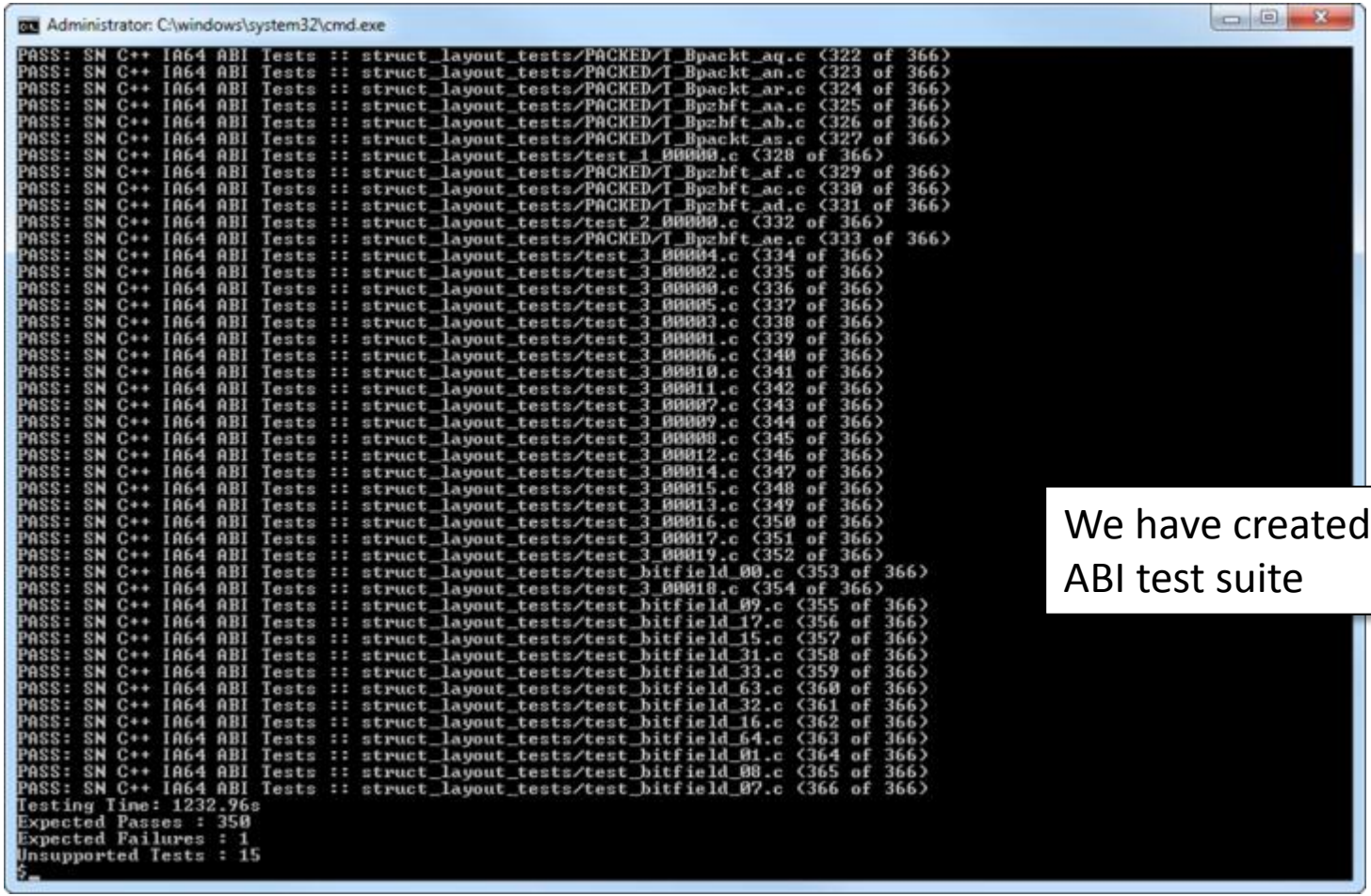
v13.x

System libraries

Maintaining a stable ABI is a

MUST

(including maintaining existing ABI bugs)



We have created a full IA64 ABI test suite

ABI TEST SUITE

We hope to **contribute** our test suite to the **community**

(some logistics still to be worked out)

Developer reaction

Developer Toolchain for PS4™

Paul T. Robinson
Sony Computer Entertainment
LLVM Dev Meeting, 7 Nov 2013

snsystems

Game Developers Love It!

"Toolchain is really nice, **link time is ~10 seconds, versus 2-4 minutes** on PC."
--Sammy Fatnassi, Eidos Montreal

"The quality of diagnostics is also incredible! It's **as pretentious as Google Search** when it comes to **correcting typos** for us and that's a good thing."
--Jean-François Marquis, Ubisoft

Game Developers Love It!

Quotes from 3rd-party studios (not SCE):

"Clang for PS4™ is a **huge improvement over GCC** for PS3™. The **same codebase** (more or less) on the same hardware **went from ~25 minutes to ~1.5 minutes.** Clang's **improved warning and error messages** also pointed us to some very questionable legacy stuff."
--Steven Houchard, Gearbox

But…

#pragma optimize

Most requested feature by an

order of magnitude

and already supported by all the other major compilers

#pragma optimize

This is the most common use-case:

```
void CriticalToPerformance() {
    ....
}

void MaybeHasABugInIt() {
    ....
}

void AlsoCriticalToPerformance() {
    ....
}
```

Game runs too slowly at –O0, but is very hard to debug at –O2

#pragma optimize

```
void CriticalToPerformance() {
    ....
}

#pragma optimize off
void MaybeHasABugInIt() {
    ....
}
#pragma optimize on

void AlsoCriticalToPerformance() {
    ....
}
```

Solution: use a pragma to selectively disable optimization on a small set of functions to be debugged

We proposed this on the mailing lists, but it is a major change and we got a limited response

#pragma optimize

```
void CriticalToPerformance() {
    ....
}

__attribute__((optnone))
void MaybeHasABugInIt() {
    ....
}


void AlsoCriticalToPerformance() {
    ....
}
```

Short term solution: function level attribute to disable optimization

Not user-friendly for more than a very small number of functions at a time!

#pragma optimize

```
#if SNC
    #define OPT_OFF ...
    #define OPT_ON ...
#elif GCC
    #define OPT_OFF ...
    #define OPT_ON ...
#elif MSVC
    #define OPT_OFF ...
    #define OPT_ON ...
#elif CLANG
    ;,-(
#endif
```

Many of our users abstract this away behind a compiler-independent interface. Function attribute does not fit this model!

We still need a range-based solution

In summary

Our initial experience with Clang and LLVM has been **very positive**

Thanks to all of you who helped make Clang and LLVM great!

There are still **improvements** that can be made…

We will be working alongside you to make them