Microsoft Research

SMT@Microsoft

FMICS, Trento, 2011

Leonardo de Moura
Microsoft Research

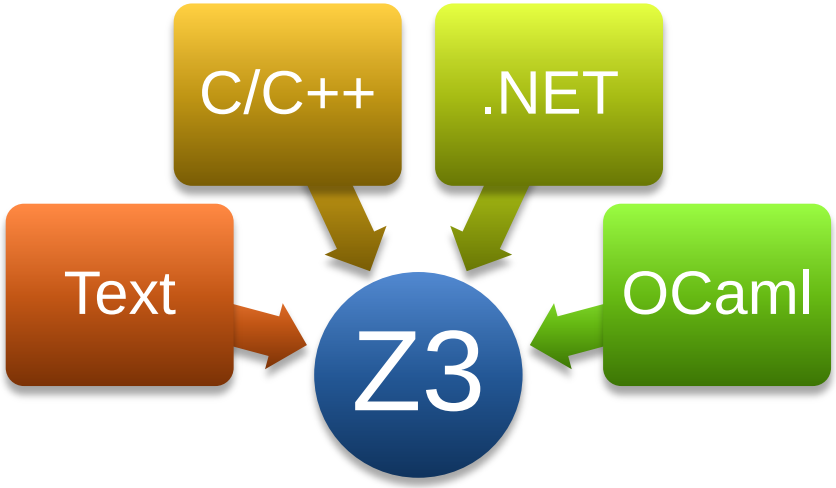
Satisfiability Modulo Theories (SMT)

A Satisfiability Checker

with built-in support for useful theories

SMT@Microsoft: Solver

- Z3 is a solver developed at Microsoft Research.
- Development/Research driven by internal customers.
- Free for non-commercial use.
- Interfaces:



- http://research.microsoft.com/projects/z3

Microsoft Research

(RiSE) Research in Software Engineering

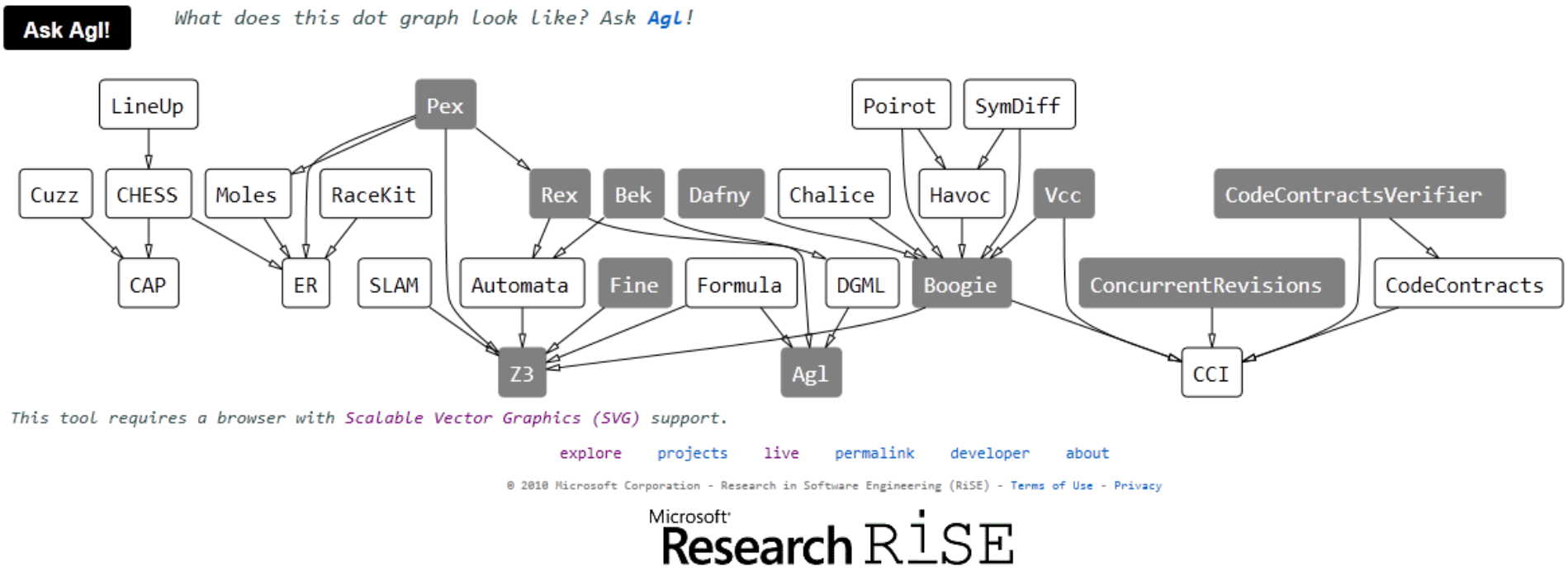
An Efficient Theorem Prover

```
(declare-const x Int)
(declare-const y Int)
(assert (> x 10))
(assert (or (< x 5) (> y x)))
(check-sat)
(model)
```

ask z3 *Is this SMT formula satisfiable?* ***Click 'ask z3'!*** *Read more or watch the video.*

```
sat
("model" "
(define x 11)
(define y 12)")
```

Some Microsoft Tools using Z3



http://research.microsoft.com/projects/z3

Some Microsoft Tools using Z3

Property Driven

Execution Guided

Model Based

Vcc

F7 1.0

Price: FREE

F7 is created to be a simple to use yet ideal typechecker for the F# programming language. F7 supports static checking of properties expressed with refinement types.

Filesize: 3.2 MB

Download now

Details Screenshot

F7 1.0 simple to use to use Simple To programming language

F

Aud

HAVOC

SLAyer

Type Safety

BOOGIE

SLAM

Over-Approximation

Yogi

Under-Approximation

SAGE

Pex

The Spec# Programming System

Testing

M3

Analysis

BEK

FORMULA

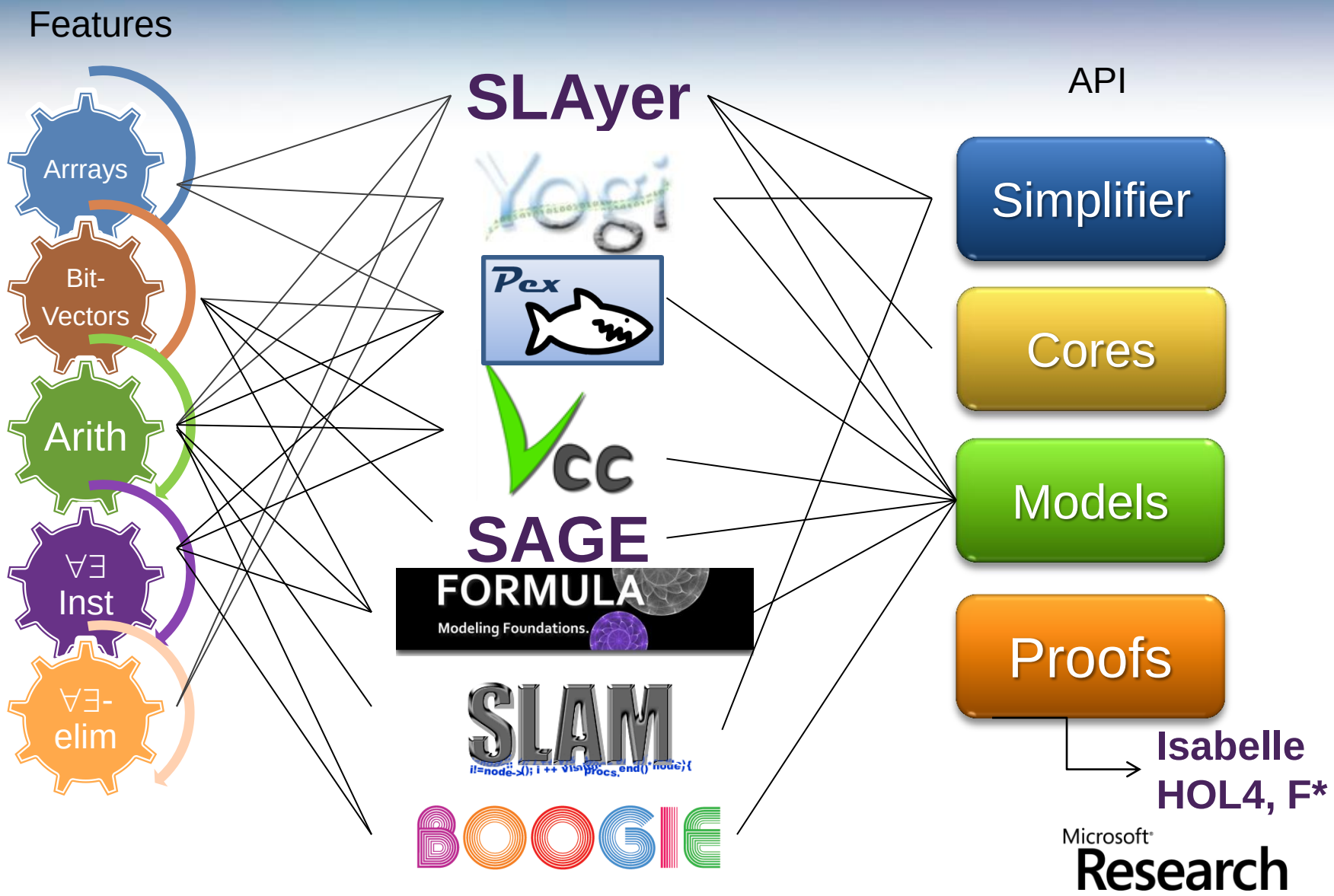
Modeling Foundations.

Synthesis

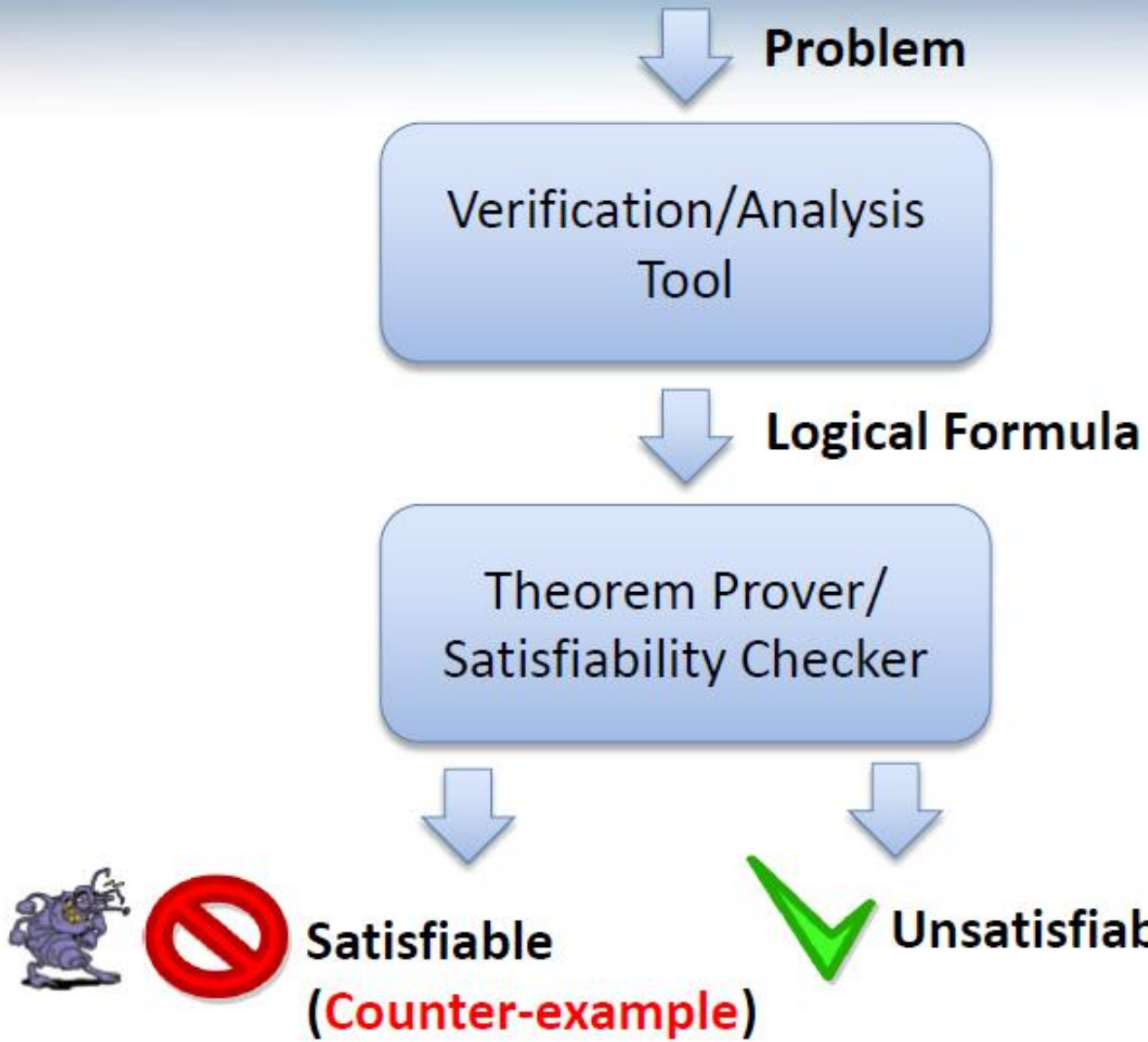
Z3

Microsoft Research

Feature Usage



Verification/Analysis Tool: "Template"



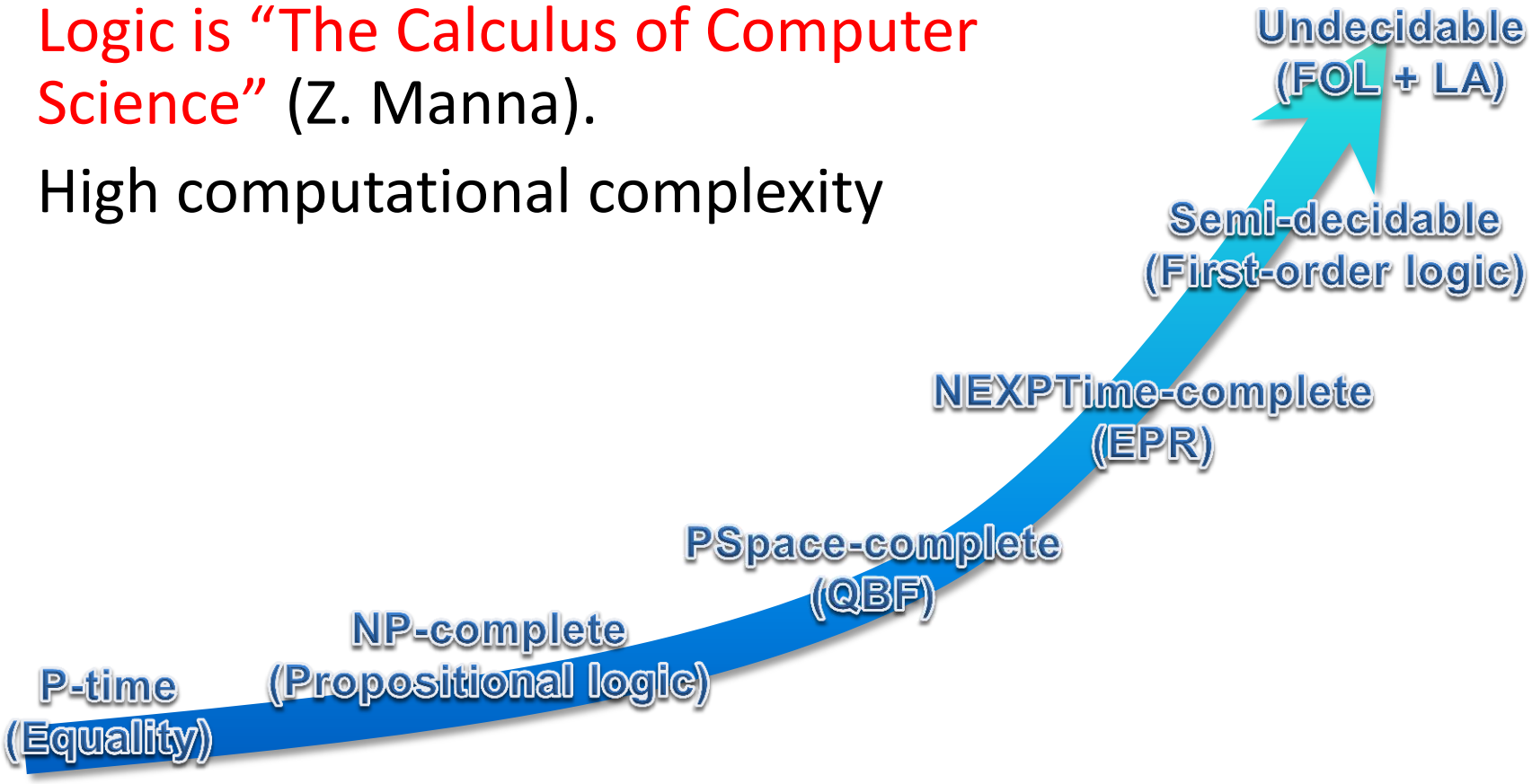
Microsoft Research

Symbolic Reasoning

Verification/Analysis tools need some form of **Symbolic Reasoning**

Symbolic Reasoning

- Logic is "The Calculus of Computer Science" (Z. Manna).
- High computational complexity

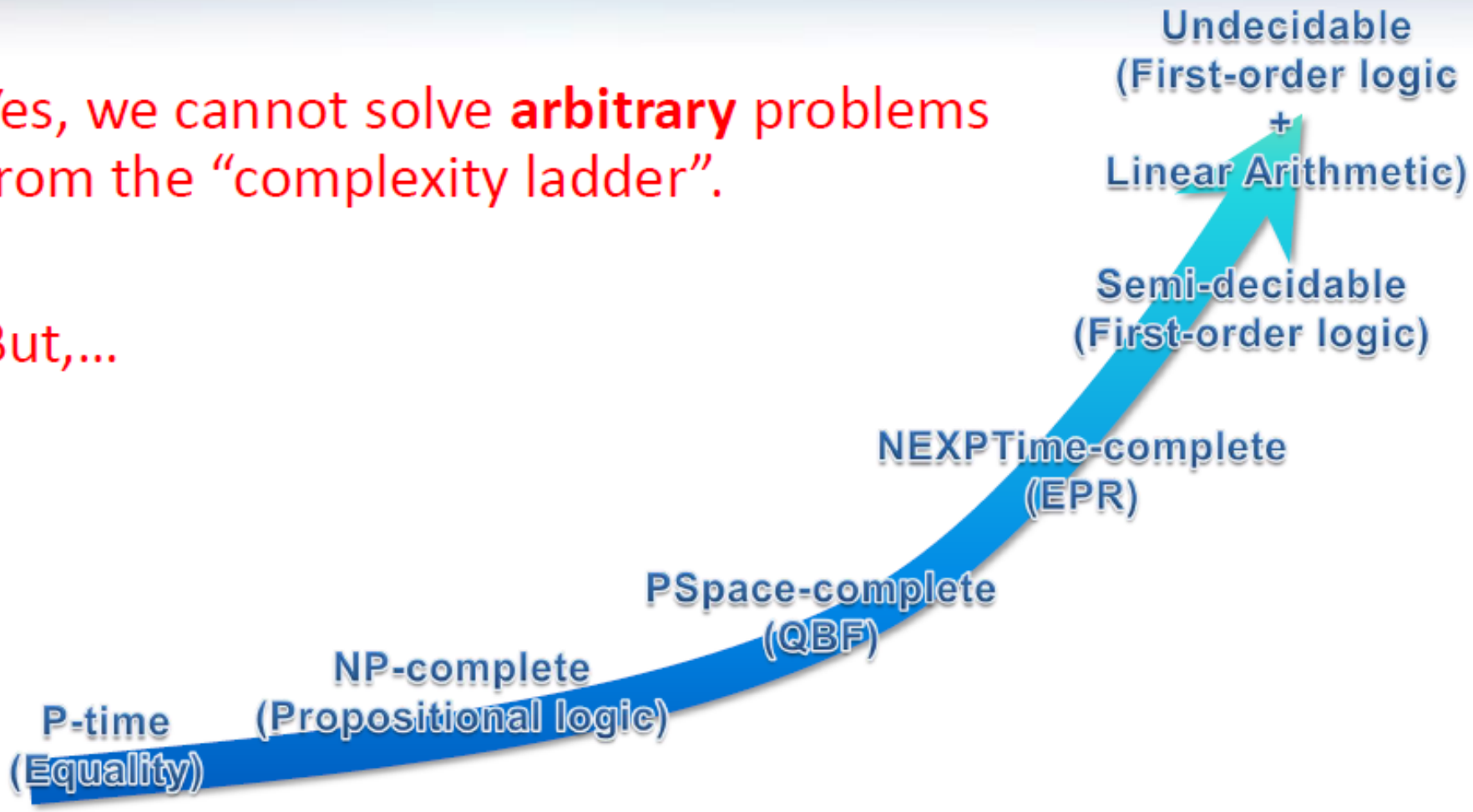


Microsoft Research

Symbolic Reasoning

Yes, we cannot solve **arbitrary** problems from the "complexity ladder".

But,…



Microsoft Research

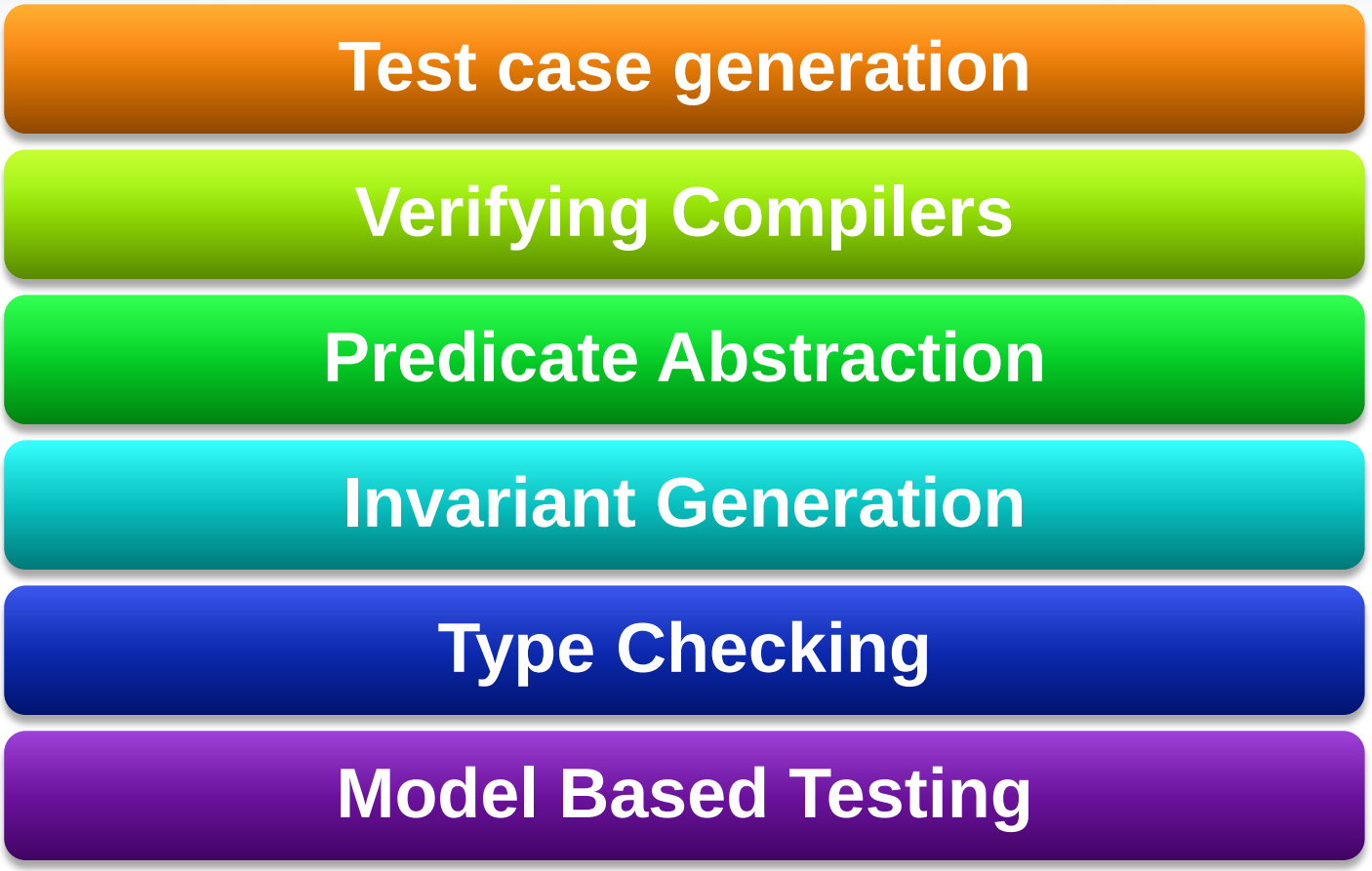
Symbolic Reasoning

We can try to solve the problems we find in

real applications

Microsoft Research

Applications



Satisfiability Modulo Theories (SMT)

$$b + 2 = c \;\; and \;\; f(read(write(a,b,3), c-2) \neq f(c-b+1)$$

Satisfiability Modulo Theories (SMT)

$$b + 2 = c \quad \textit{and} \quad f(read(write(a,b,3), c-2) \neq f(c-b+1)$$

Arithmetic

Satisfiability Modulo Theories (SMT)

$$b + 2 = c \;\; and \;\; f(read(write(a,b,3), c-2) \neq f(c-b+1)$$

Array Theory

Satisfiability Modulo Theories (SMT)

$$b + 2 = c \;\; and \;\; f(read(write(a,b,3), c-2) \neq f(c-b+1)$$

Uninterpreted Functions

Application Scenarios

"Big" and hard formulas

Thousands of "small" and easy formulas

Short timeout (< 5secs)

Microsoft Research

Application Scenarios

"Big" and hard formulas

HAVOC

VCC

Thousands of "small" and easy formulas

Short timeout (< 5secs)

SAGE

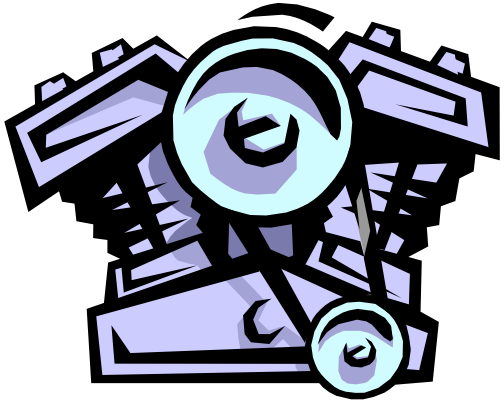
Microsoft Research

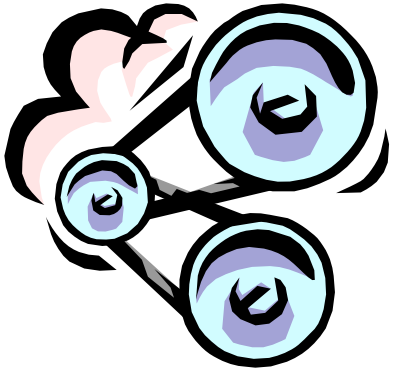
Combining Engines

Current SMT solvers provide

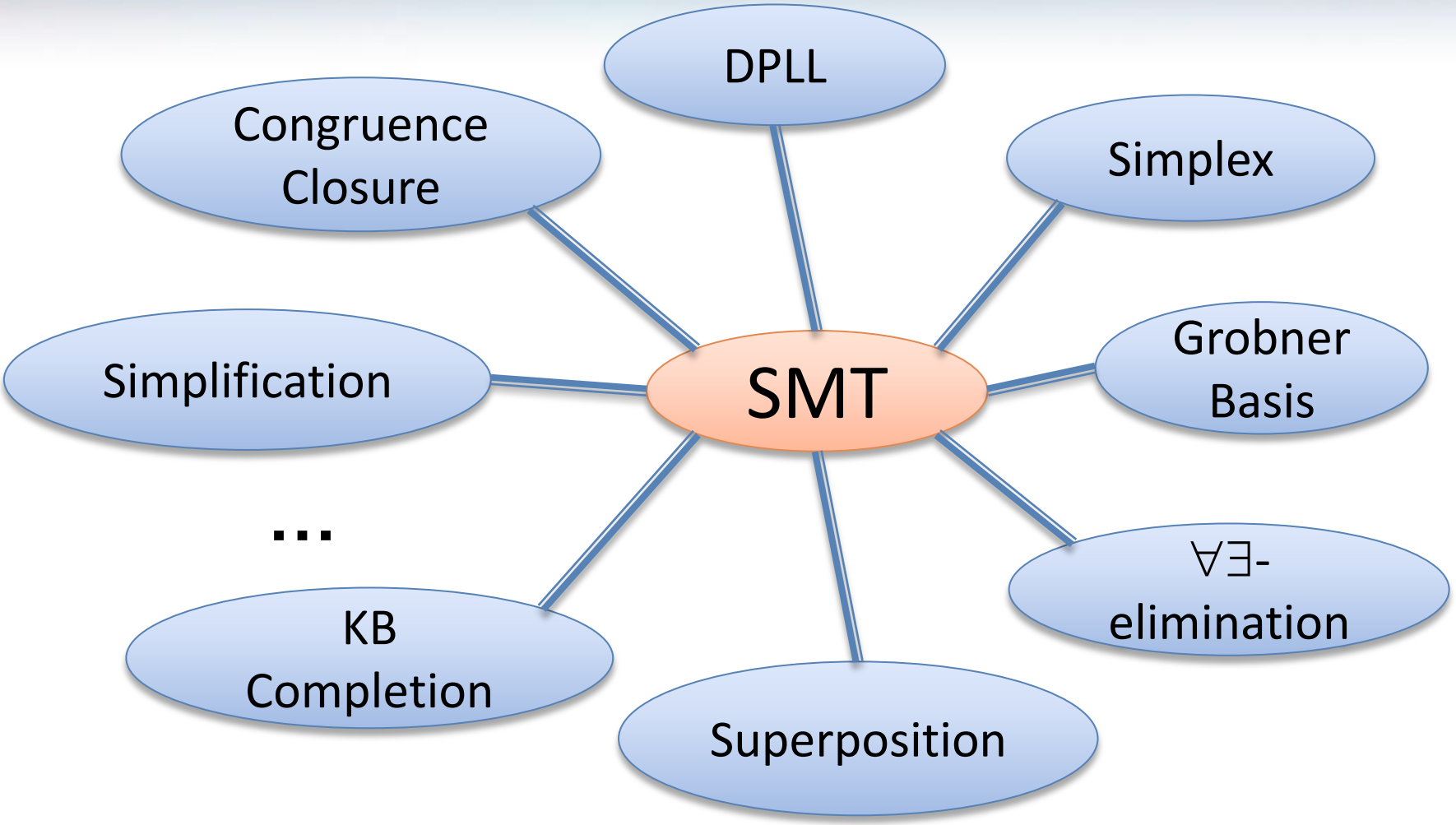
a combination

of different engines

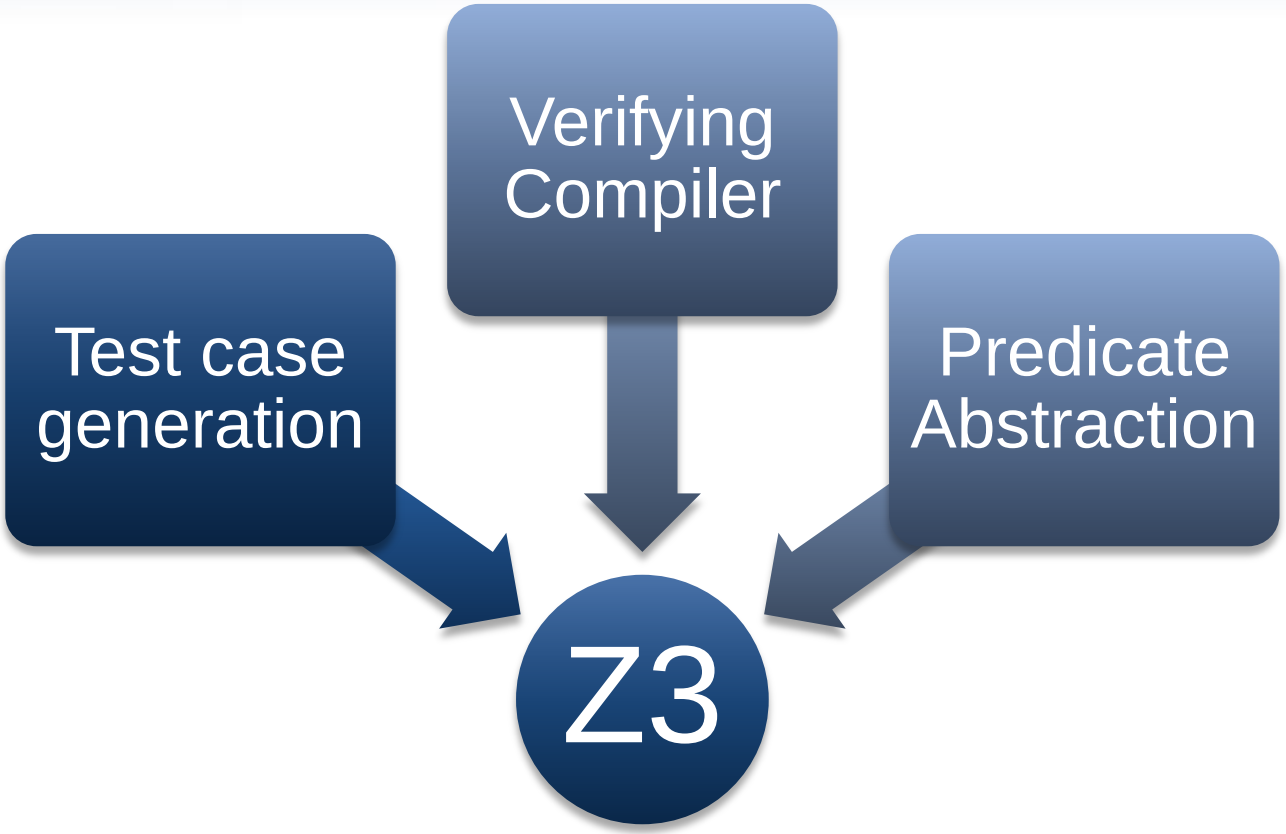




Combining Engines

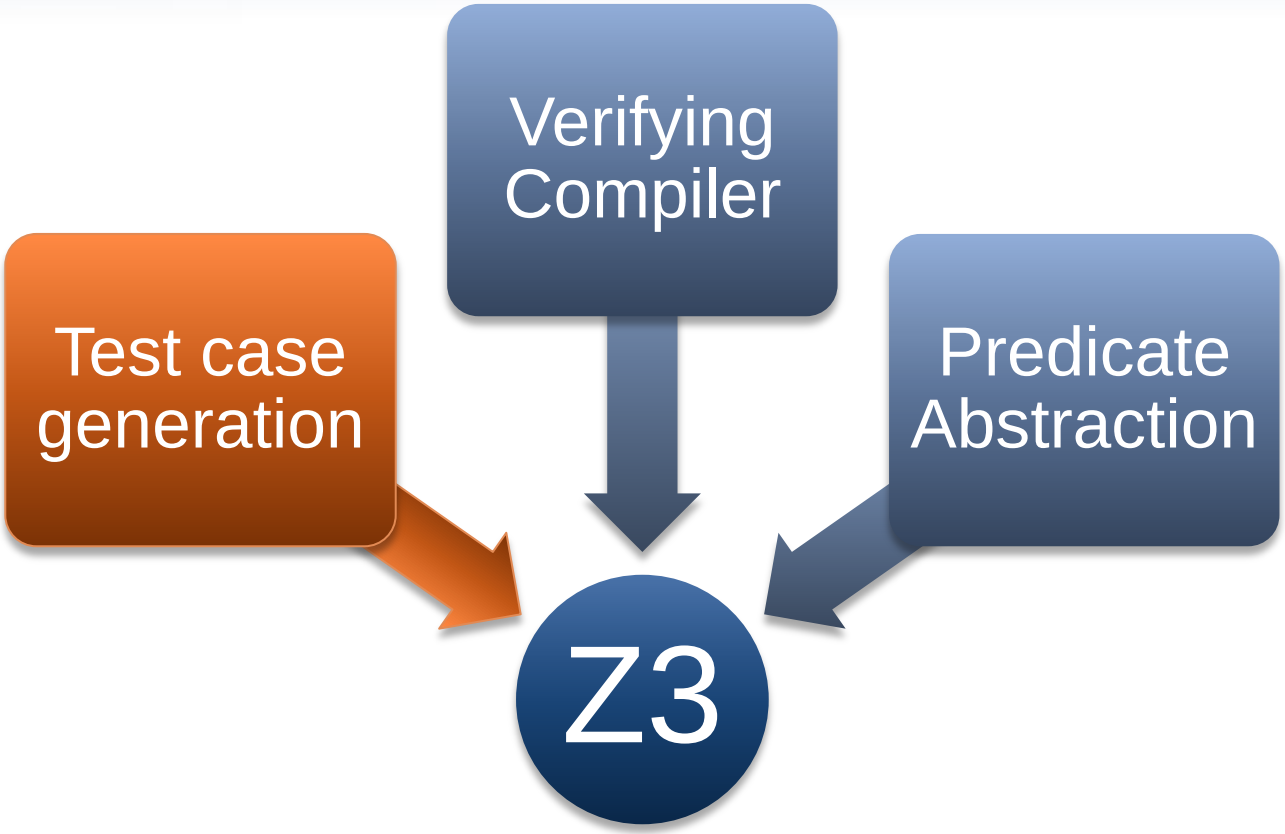


SMT: Some Applications



Microsoft Research

SMT: Some Applications



Microsoft Research

Test-case generation

- Test (correctness + usability) is 95% of the deal:
  - Dev/Test is 1-1 in products.
  - Developers are responsible for unit tests.
- Tools:
  - Annotations and static analysis (SAL + ESP)
  - File Fuzzing
  - Unit test case generation

Security is critical

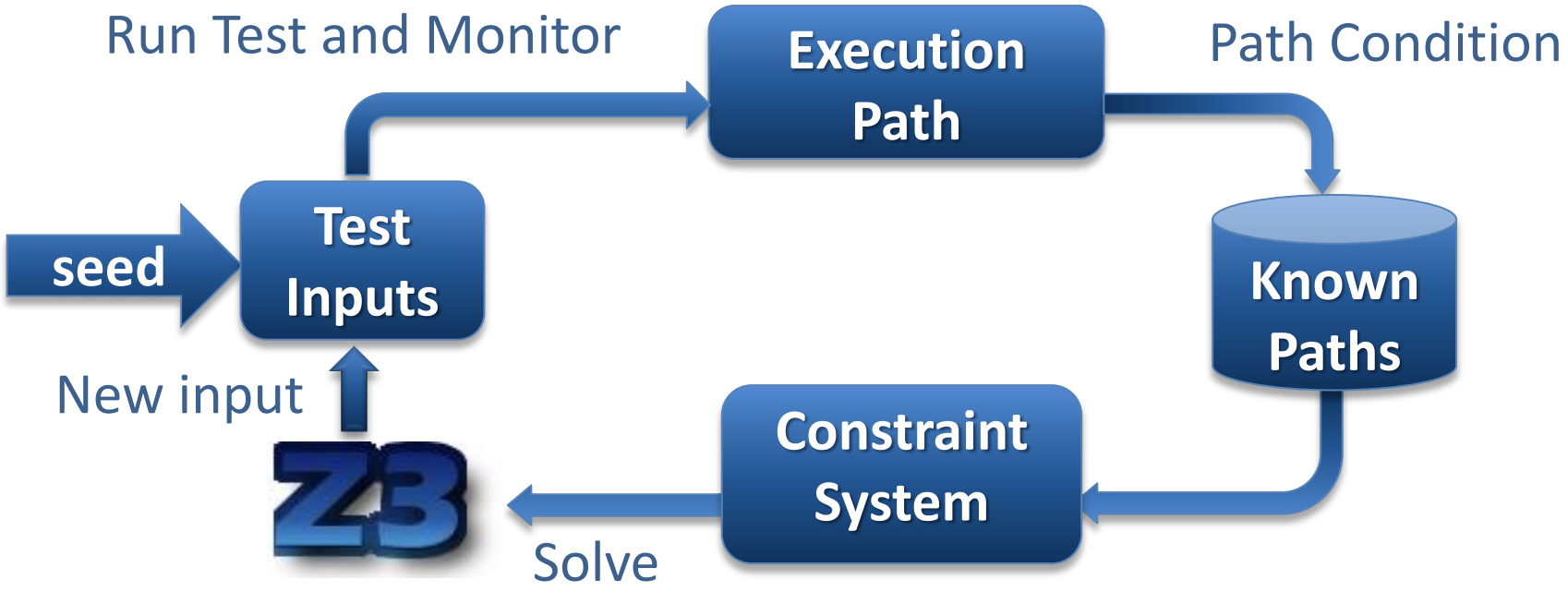
- Security bugs can be very expensive:
  - Cost of each MS Security Bulletin: $600k to $Millions.
  - Cost due to worms: $Billions.
  - The real victim is the customer.
- Most security exploits are initiated via files or packets.
  - Ex: Internet Explorer parses dozens of file formats.
- Security testing: hunting for million dollar bugs
  - Write A/V
  - Read A/V
  - Null pointer dereference
  - Division by zero

Hunting for Security Bugs.

- Two main techniques used by *"black hats"*:
  - Code inspection (of binaries).
  - *Black box fuzz testing.*

- **Black box** fuzz testing:
  - A form of black box random testing.
  - Randomly *fuzz* (=modify) a well formed input.
  - Grammar-based fuzzing: rules to encode how to fuzz.
- **Heavily** used in security testing
  - At MS: several internal tools.
  - Conceptually simple yet effective in practice

Directed Automated Random Testing (DART)



Microsoft Research

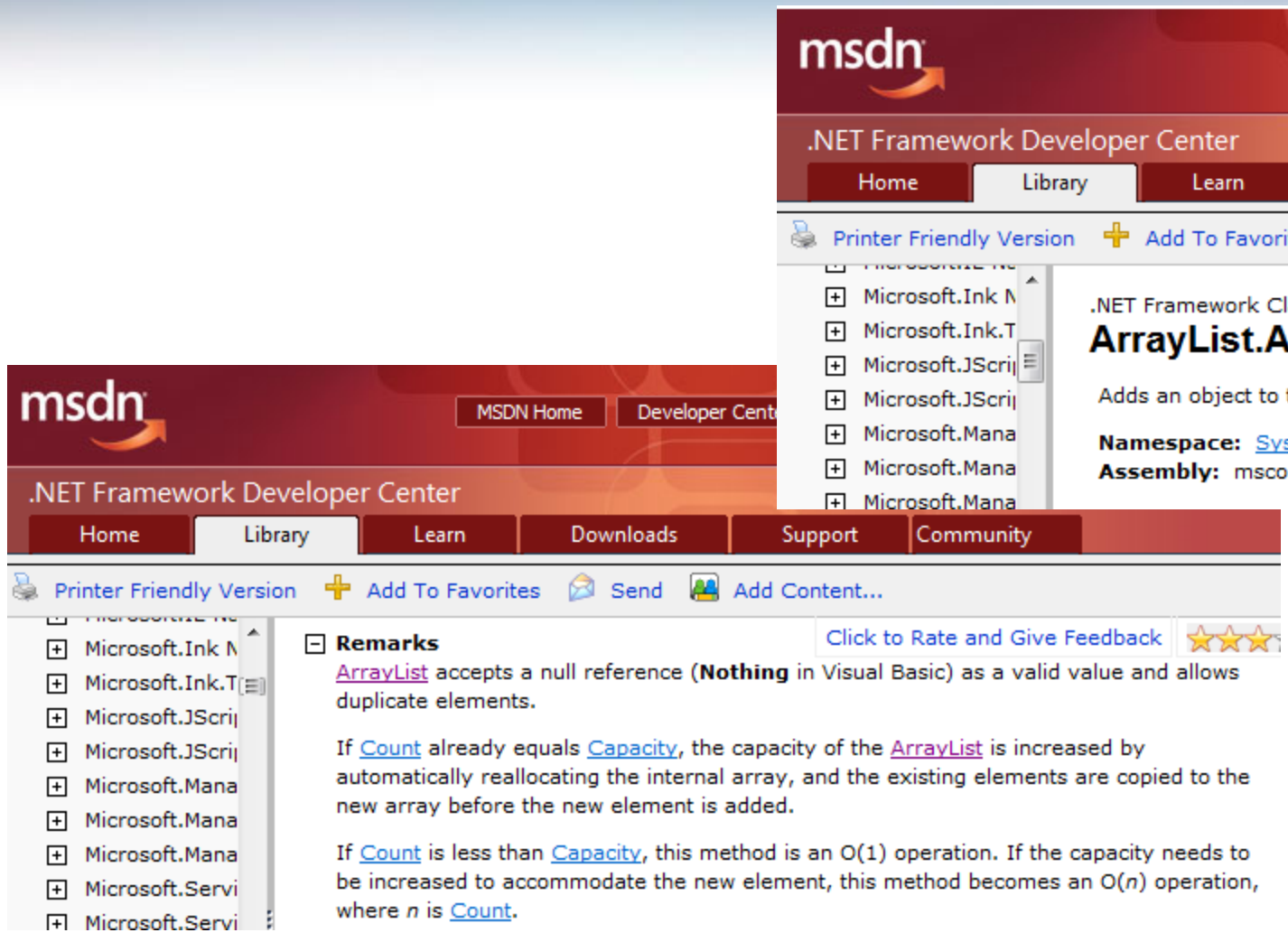
DARTish projects at Microsoft

PEX	Implements DART for .NET.
SAGE	Implements DART for x86 binaries.
YOGI	Implements DART to check the feasibility of program paths generated statically using a SLAM-like tool.
Vigilante	Partially implements DART to dynamically generate worm filters.

What is Pex?

- Test input generator
  - Pex starts from parameterized unit tests
  - Generated tests are emitted as traditional unit tests

ArrayList: The Spec



Microsoft Research

ArrayList: AddItem Test

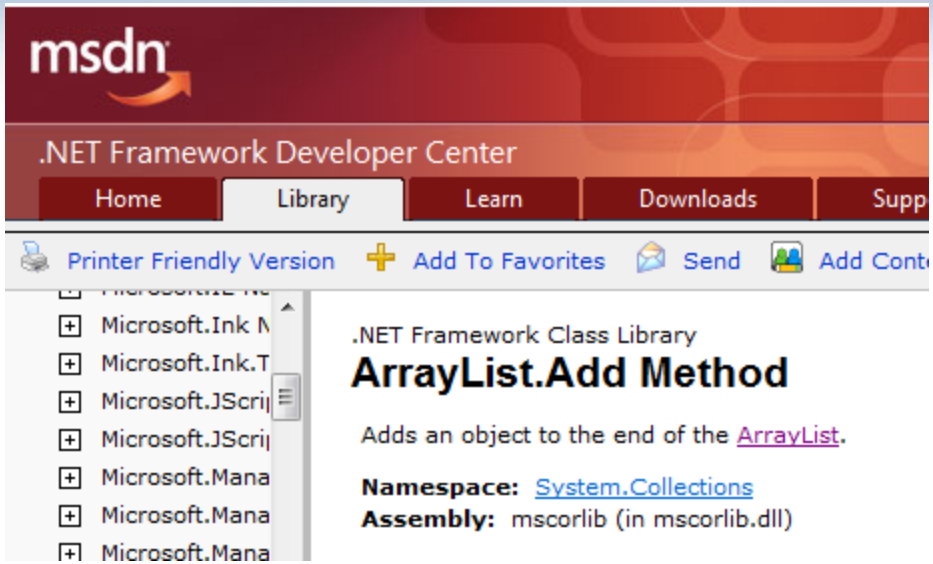
```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```



Microsoft Research

ArrayList: Starting Pex…

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

Inputs

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Microsoft Research

ArrayList: Run 1, (0,null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Inputs
(0,null)

Microsoft Research

ArrayList: Run 1, (0,null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

	Inputs	Observed Constraints
	(0,null)	!(c<0)

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

c < 0 → false

Microsoft Research

ArrayList: Run 1, (0,null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

`0 == c  →  true`

Microsoft Research

ArrayList: Run 1, (0,null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c

item == item → true

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Microsoft Research

ArrayList: Picking the next branch to cover

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c		

Microsoft Research

ArrayList: Solve constraints using SMT solver

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	

Microsoft Research

ArrayList: Run 2, (1, null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

0 == c → false

Microsoft Research

ArrayList: Pick new branch

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

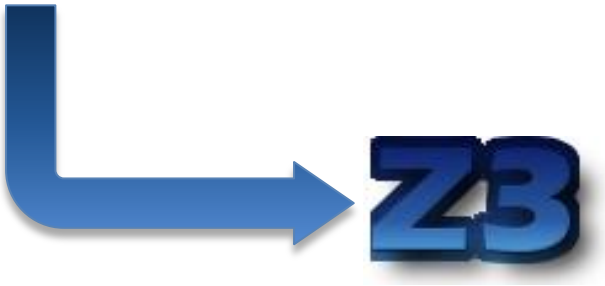
```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c
c<0		



Microsoft Research

ArrayList: Run 3, (-1, null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c
c<0	(-1,null)	

Microsoft Research

ArrayList: Run 3, (-1, null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c
c<0	(-1,null)	c<0

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

c < 0 → true

Microsoft Research

ArrayList: Run 3, (-1, null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c
c<0	(-1,null)	c<0

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Microsoft Research

ArrayList: Run 3, (-1, null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c
c<0	(-1,null)	c<0

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Microsoft Research

White box testing in practice

How to test this code?

(Real code from .NET base class libraries.)

```
[SecurityPermissionAttribute(SecurityAction.LinkDemand, Flags=SecurityPermissionFlag.SerializationFormatter)]
public ResourceReader(Stream stream)
{
    if (stream==null)
        throw new ArgumentNullException("stream");
    if (!stream.CanRead)
        throw new ArgumentException(Environment.GetResourceString("Argument_StreamNotReadable"));

    _resCache = new Dictionary<String, ResourceLocator>(FastResourceComparer.Default);
    _store = new BinaryReader(stream, Encoding.UTF8);
    // We have a faster code path for reading resource files from an assembly.
    _ums = stream as UnmanagedMemoryStream;

    BCLDebug.Log("RESMGRFILEFORMAT", "ResourceReader .ctor(Stream).  UnmanagedMemoryStream: "+(_ums!=null));
    ReadResources();
}
```

White box testing in practice

```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader is closed!");
            BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
#if !FEATURE_PAL
            _typeLimitingBinder = new TypeLimitingDeserializationBinder();
            bf.Binder = _typeLimitingBinder;
#endif
            _objFormatter = bf;
            try {
                // Read ResourceManager header
                // Check for magic number
                int magicNum = _store.ReadInt32();
                if (magicNum != ResourceManager.MagicNumber)
                    throw new ArgumentException(Environment.GetResourceString("Resources_StreamNotValid"));
                // Assuming this is ResourceManager header V1 or greater, hopefully
                // after the version number there is a number of bytes to skip
                // to bypass the rest of the ResMgr header.
                int resMgrHeaderVersion = _store.ReadInt32();
                if (resMgrHeaderVersion > 1) {
                    int numBytesToSkip = _store.ReadInt32();
                    BCLDebug.Log("RESMGRFILEFORMAT", LogLevel.Status, "ReadResources: Unexpected ResMgr header
                    BCLDebug.Assert(numBytesToSkip >= 0, "numBytesToSkip in ResMgr header should be positive!"
                    _store.BaseStream.Seek(numBytesToSkip, SeekOrigin.Current);
                } else {
                    BCLDebug.Log("RESMGRFILEFORMAT", "ReadResources: Parsing ResMgr header v1.");
                    SkipInt32();    // We don't care about numBytesToSkip.
                    // Read in type name for a suitable ResourceReader
```

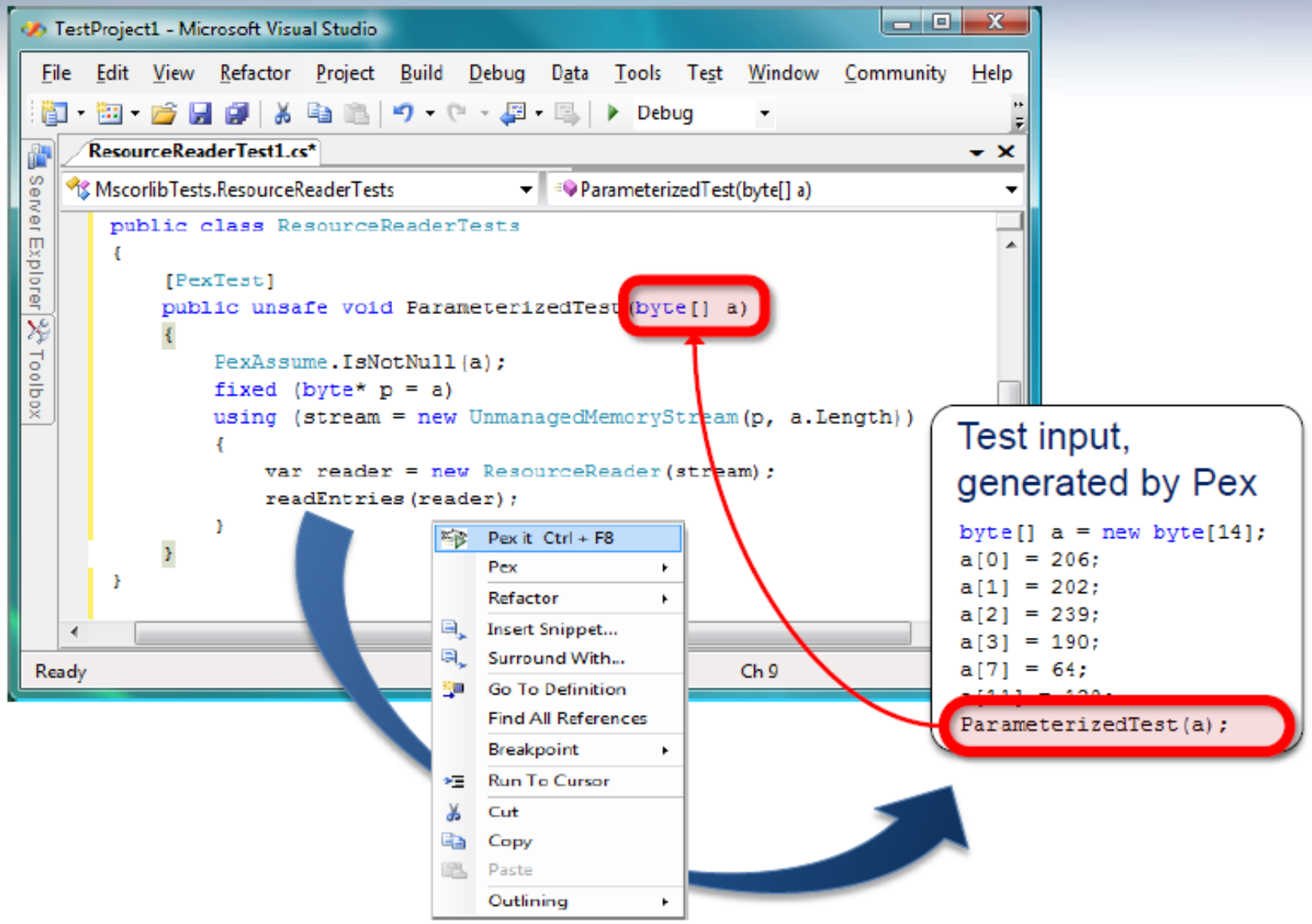
White box testing in practice

```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader is closed!");
            BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
#if !FEATURE_PAL
            _typeLimitingBinder = new TypeLimitingDeserializationBinder();
            bf.Binder = _typeLimitingBinder;
#endif
            _objFormatter = bf;
            try {
                // Read ResourceManager header
                // Check for magic number
                int magicNum = _store.ReadInt32();
                if
                //
                //
                int
                if
                } e
                // Read in type name for a suitable ResourceReader
```

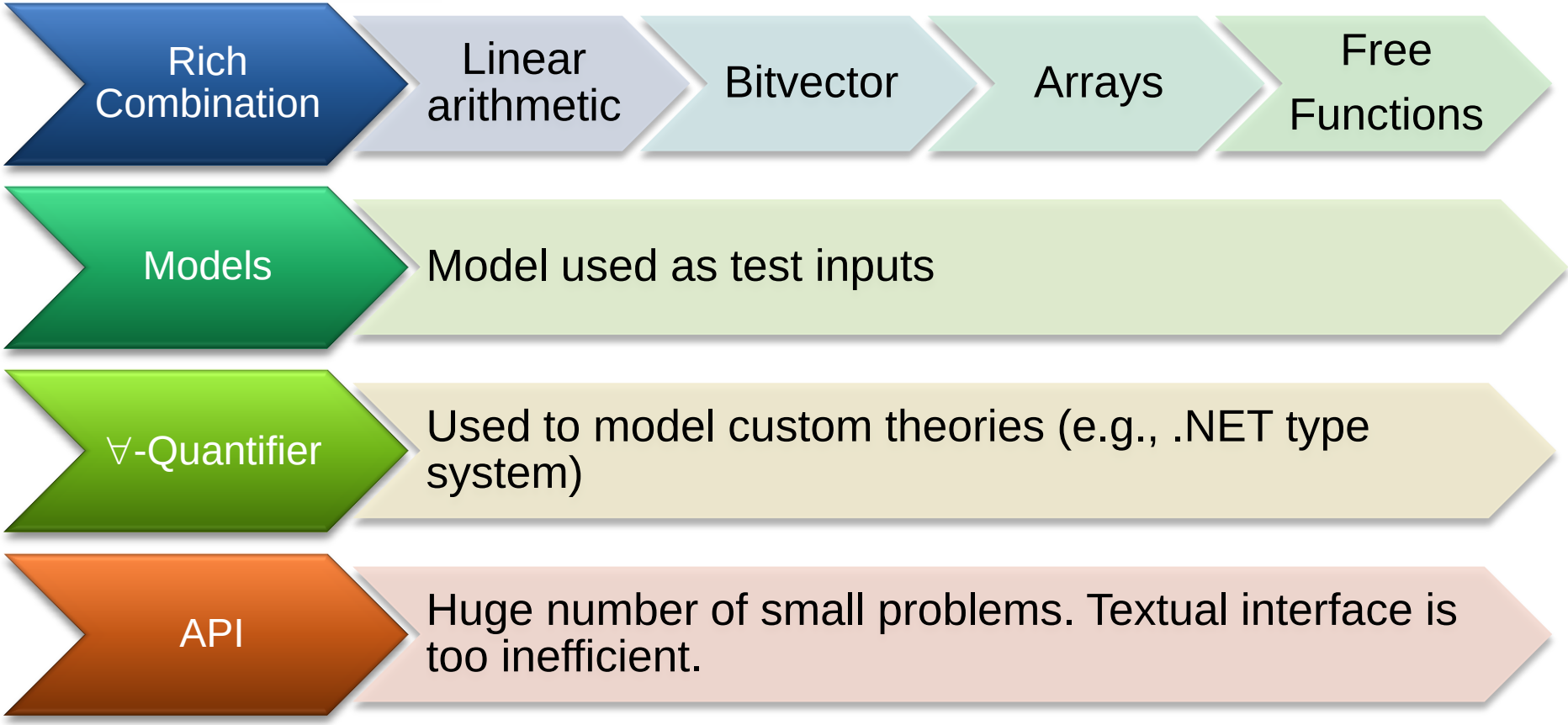
```
public virtual int ReadInt32() {
    if (m_isMemoryStream) {
        // read directly from MemoryStream buffer
        MemoryStream mStream = m_stream as MemoryStream;
        BCLDebug.Assert(mStream != null, "m_stream as MemoryStream != null");

        return mStream.InternalReadInt32();
    }
    else
    {
        FillBuffer(4);
        return (int)(m_buffer[0] | m_buffer[1] << 8 | m_buffer[2] << 16 | m_buffer[3] << 24);
    }
}
```

Pex – Test Input Generation



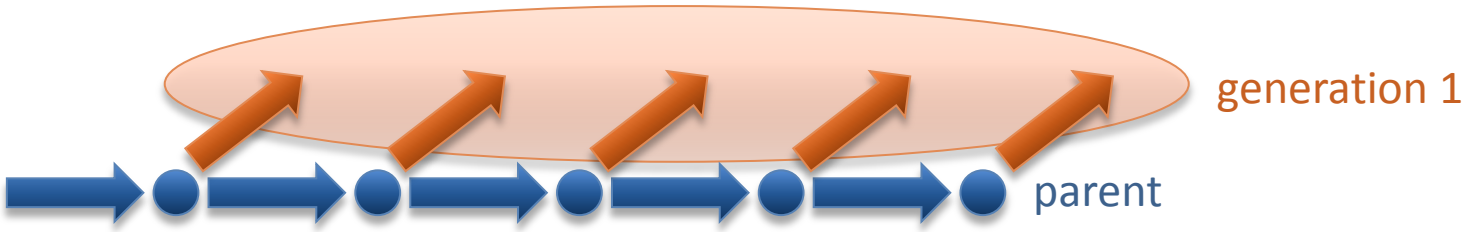
PEX ↔ Z3



Microsoft Research

SAGE

- Apply DART to large applications (not units).
- Start with well-formed input (not random).
- Combine with generational search (not DFS).
  - Negate 1-by-1 each constraint in a path constraint.
  - Generate many children for each parent run.



Microsoft Research

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 0 – seed file

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 00 00 00 00 00 00 00 00 00 00 00 00 ; RIFF............
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 1

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 00 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF....***.....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 2

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 3

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 00 00 00 00 ; ....strh........
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 4

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 5

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 00 00 00 00 00 00 00 00 ; ....strf........
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 6

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 00 00 00 00 28 00 00 00 ; ....strf....(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 7

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 00 00 00 00 28 00 00 00 ; ....strf....(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 C9 9D E4 4E ; ............ÉäN
00000060h: 00 00 00 00                                     ; ....
```

Generation 8

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 00 00 00 00 28 00 00 00 ; ....strf....(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 9

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 B2 75 76 3A 28 00 00 00 ; ....strf²uv:(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 10 – CRASH

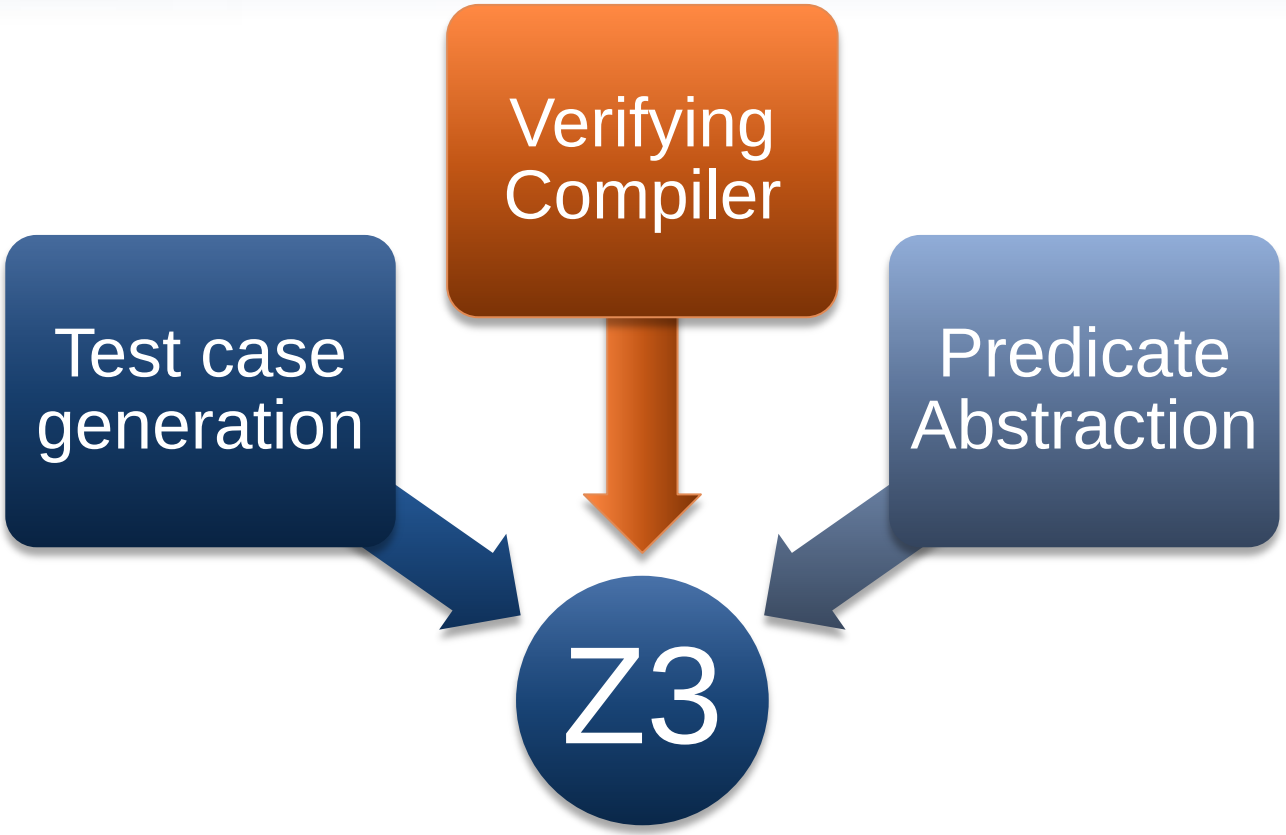
SAGE (cont.)

- SAGE is very effective at finding bugs.
- Works on large applications.
- Fully automated
- Easy to deploy (x86 analysis – any language)
- Used in various groups inside Microsoft
- Powered by Z3.

SAGE ↔ Z3

- Formulas are usually big conjunctions.
- SAGE uses only the bitvector and array theories.
- Pre-processing step has a huge performance impact.
  - Eliminate variables.
  - Simplify formulas.
- Early unsat detection.

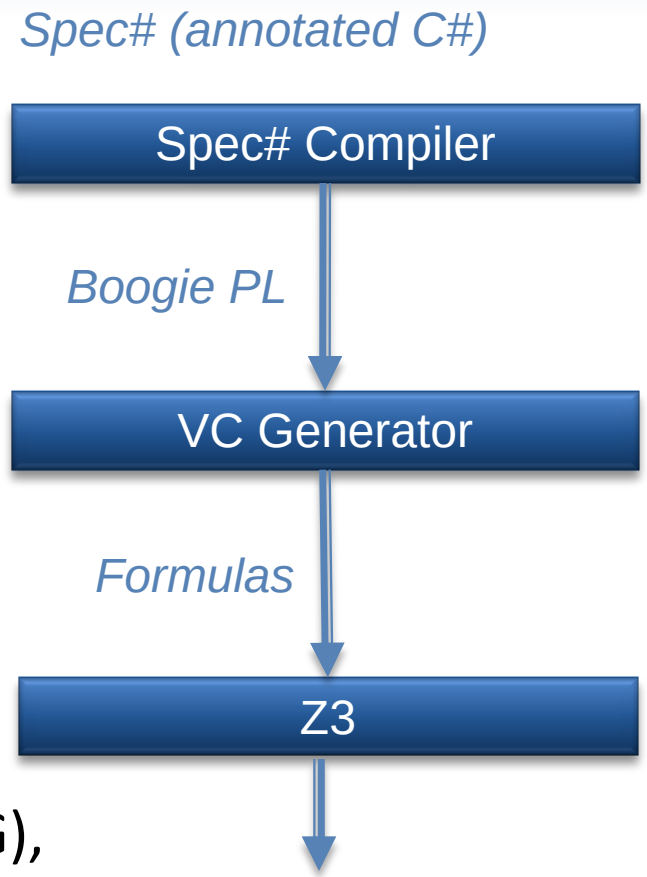
SMT: Some Applications



Microsoft Research

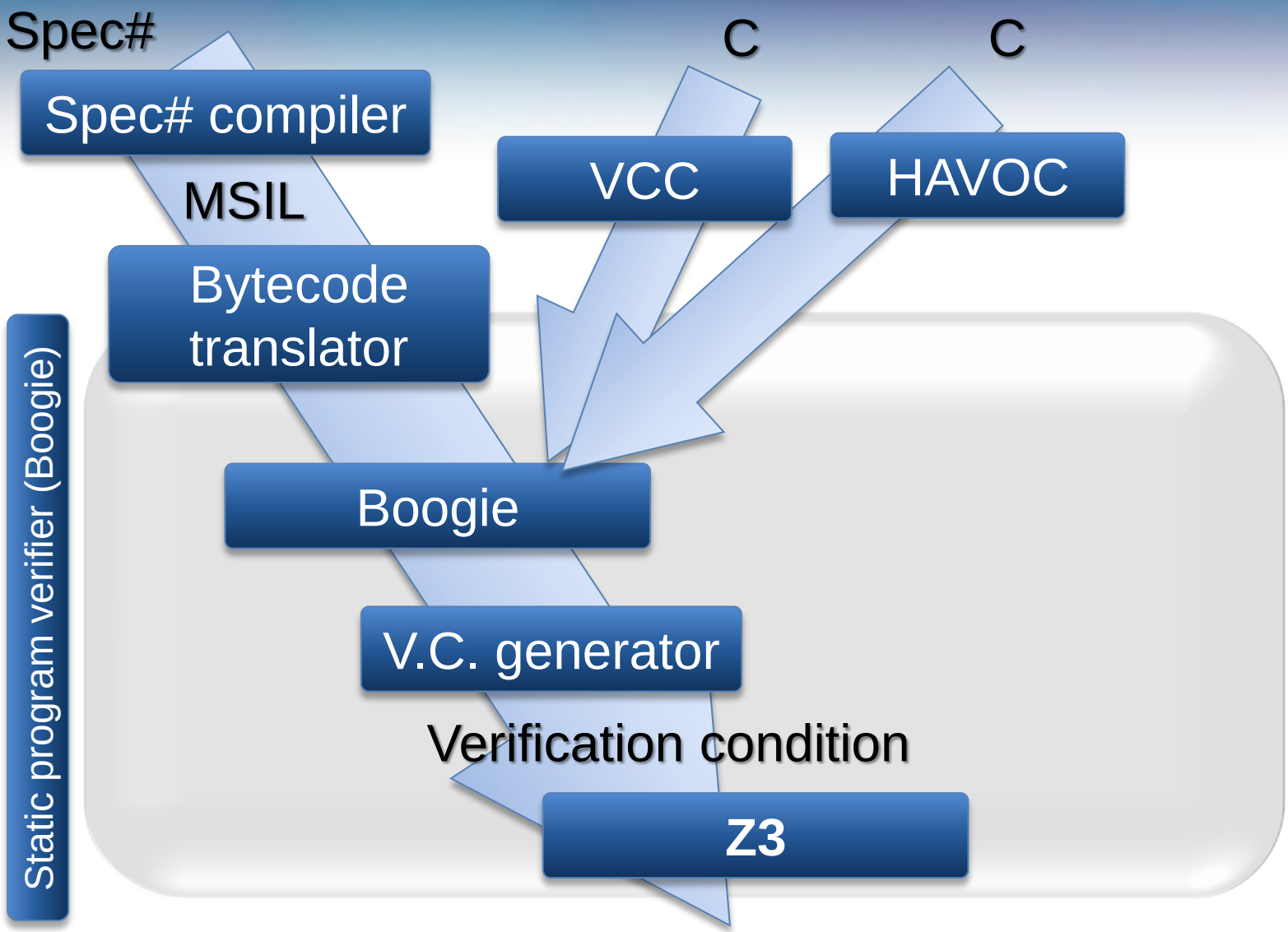
Spec# Approach for a Verifying Compiler

- *Source Language*
  - C# + goodies = Spec#
- *Specifications*
  - method contracts,
  - invariants,
  - field and type annotations.
- *Program Logic:*
  - *Dijkstra's weakest preconditions.*
- *Automatic Verification*
  - type checking,
  - verification condition generation (VCG),
  - automatic theorem proving Z3



Microsoft Research

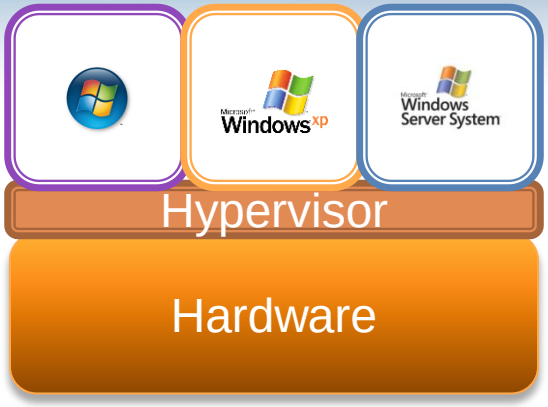
Verification architecture



A Verifying C Compiler

- VCC translates an *annotated C program* into a *Boogie PL* program.
- A C-ish memory model
  - Abstract heaps
  - Bit-level precision
- Microsoft Hypervisor: verification grand challenge.

Hypervisor: A Manhattan Project



- **Meta OS**: small layer of software between hardware and OS
- **Mini**: 60K lines of non-trivial concurrent systems C code
- **Critical:** must provide functional resource abstraction
- **Trusted**: a verification grand challenge

Hypervisor: Some Statistics

- VCs have several Mb
- Thousands of non ground clauses
- Developers are willing to wait at most 5 min per VC

Main Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime

$\forall$ h,o,f:

IsHeap(h) $\wedge$ o $\neq$ null $\wedge$ read(h, o, alloc) = t

$\Rightarrow$

read(h,o, f) = null $\vee$ read(h, read(h,o,f),alloc) = t

Main Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime
- Frame axioms

$$\forall\ o, f:$$
$$o \neq null \wedge read(h_0, o, alloc) = t \Rightarrow$$
$$read(h_1,o,f) = read(h_0,o,f) \vee (o,f) \in M$$

Main Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime
- Frame axioms
- User provided assertions

$\forall$ i,j: i $\leq$ j $\Rightarrow$ read(a,i) $\leq$ read(b,j)

Main Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime
- Frame axioms
- User provided assertions
- Theories
  - $\forall$ x: p(x,x)
  - $\forall$ x,y,z: p(x,y), p(y,z) $\Rightarrow$ p(x,z)
  - $\forall$ x,y: p(x,y), p(y,x) $\Rightarrow$ x = y

Main Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime
- Frame axioms
- User provided assertions
- Theories
- Solver must be fast in satisfiable instances.

We want to find bugs!

Bad news

There is no sound and refutationally complete procedure for linear integer arithmetic + free function symbols

Many Approaches

- Heuristic quantifier instantiation
- Combining SMT with Saturation provers
- Complete quantifier instantiation
- Decidable fragments
- Model based quantifier instantiation

Challenge: modeling runtime

- Is the axiomatization of the runtime consistent?
- False implies everything
- Partial solution: SMT + Saturation Provers
- Found many bugs using this approach

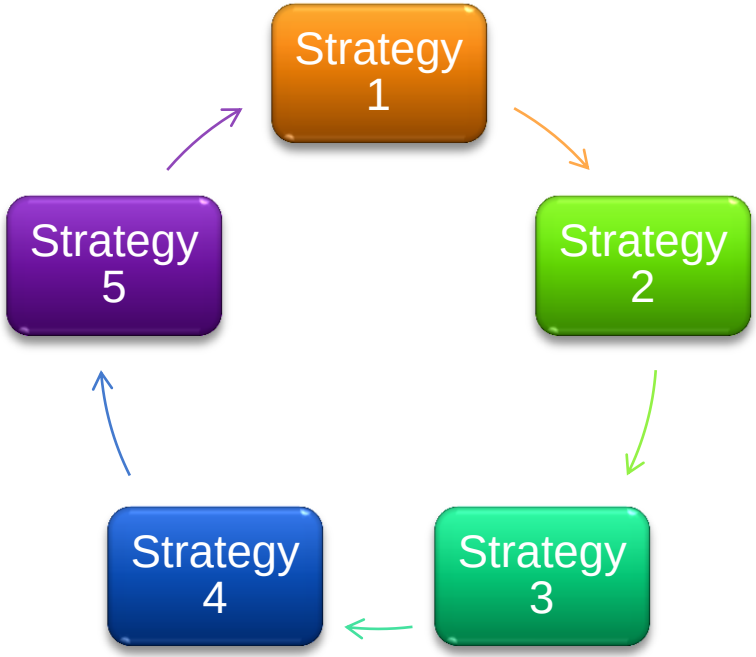
Challenge: Robustness

- Standard complain

  "I made a small modification in my Spec, and Z3 is timingout"
- This also happens with SAT solvers (NP-complete)
- In our case, the problems are undecidable
- Partial solution: parallelization

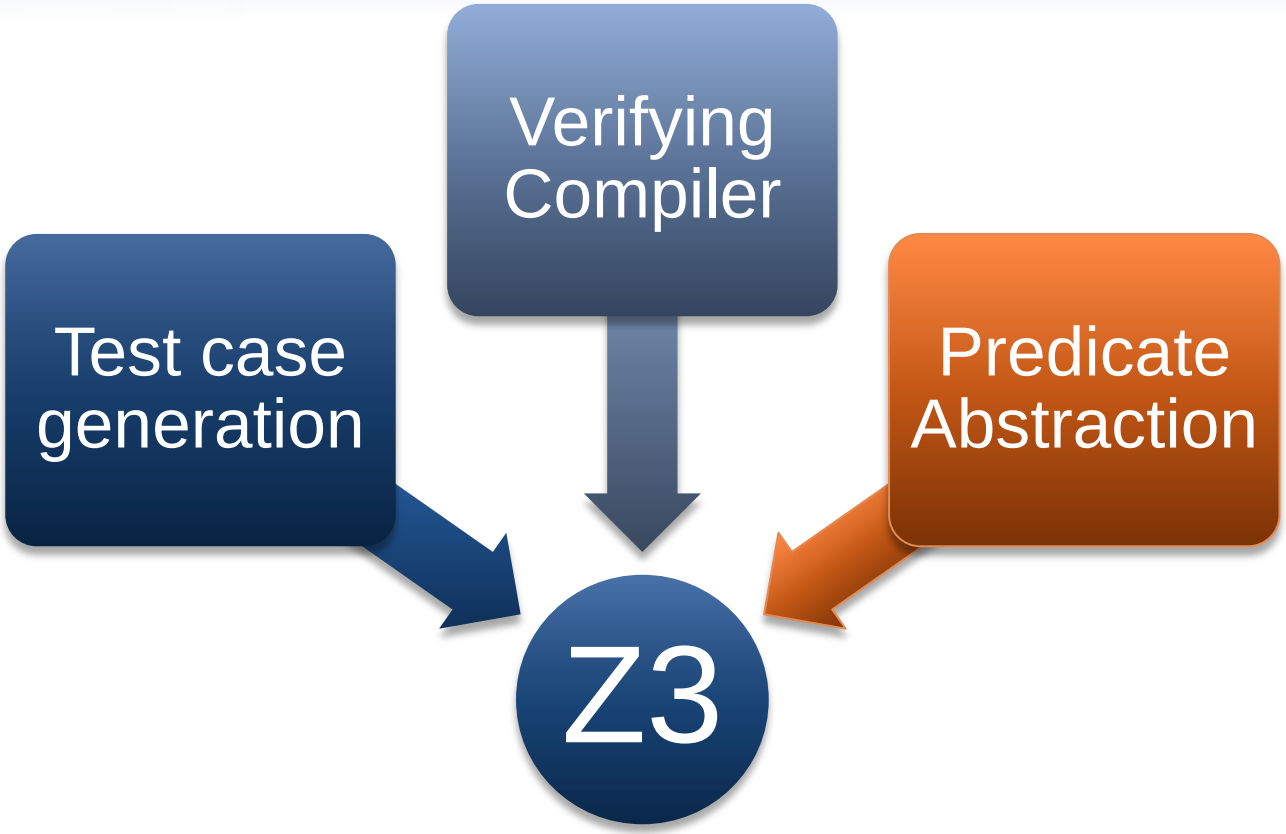
Parallel Z3

- Joint work with Y. Hamadi (MSRC) and C. Wintersteiger
- Multi-core & Multi-node (HPC)
- Different strategies in parallel
- Collaborate exchanging lemmas



Microsoft Research

SMT: Some Applications



Microsoft Research

Overview

- *http://research.microsoft.com/slam/*
- *SLAM/SDV* is a software model checker.
- Application domain: *device drivers*.
- Architecture:

  c2bp C program → boolean program (*predicate abstraction).*

  bebop Model checker for boolean programs.

  newton Model refinement (check for path feasibility)
- SMT solvers are used to perform predicate abstraction and to check path feasibility.
- c2bp makes several calls to the SMT solver. The formulas are relatively small.

Predicate Abstraction: *c2bp*

- **Given** a C program P and $F = \{p_1, \ldots, p_n\}$.
- **Produce** a Boolean program $B(P, F)$
  - Same control flow structure as P.
  - Boolean variables $\{b_1, \ldots, b_n\}$ to match $\{p_1, \ldots, p_n\}$.
  - Properties true in $B(P, F)$ are true in P.
- Each p_i is a pure Boolean expression.
- Each p_i represents set of states for which p_i is true.
- Performs modular abstraction.

Abstracting Expressions via *F*

- $Implies_F\ (e)$
  - Best Boolean function over *F* that implies *e.*
- $ImpliedBy_F\ (e)$
  - Best Boolean function over *F* that is implied by *e.*
  - $ImpliedBy_F\ (e) = not\ Implies_F\ (not\ e)$

Computing $Implies_F(e)$

- minterm $m = l_1 \wedge \ldots \wedge l_n$, where $l_i = p_i$, or $l_i = not\ p_i$.
- $Implies_F\ (e)$: disjunction of all minterms that imply e.
- Naive approach
  - Generate all 2^n possible minterms.
  - For each minterm m, use SMT solver to check validity of $m \Rightarrow e$.
- Many possible optimizations

Computing $Implies_F(e)$

- F = { x < y, x = 2}
- *e* : y > 1
- Minterms over F
  - !x<y, !x=2 implies y>1
  - x<y, !x=2 implies y>1
  - !x<y, x=2 implies y>1
  - x<y, x=2 implies y>1

$Implies_F(y>1) = \neg b_{x<y} \wedge b_{x=2}$

Newton

- Given an error path *p* in the Boolean program *B*.
- Is *p* a feasible path of the corresponding C program?
  - Yes: found a bug.
  - No: find predicates that explain the infeasibility.
- Execute path symbolically.
- Check conditions for inconsistency using Z3.

SLAM ↔ Z3

- All-SAT
  - Better (more precise) Predicate Abstraction
- Unsatisfiable cores
  - Why the abstract path is not feasible?
  - Fast Predicate Abstraction

SLAM $\leftrightarrow$ Z3: Unsatisfiable cores

- Let S be an unsatisfiable set of formulas.
- $S' \subseteq S$ is an unsatisfiable core of S if:
  - S' is also unsatisfiable, and
  - There is not $S'' \subset S'$ that is also unsatisfiable.
- Computing $\text{Implies}_F(e)$ with $F = \{p_1, p_2, p_3, p_4\}$
  - Assume $p_1, p_2, p_3, p_4 \Rightarrow e$ is valid
  - That is $p_1, p_2, p_3, p_4, \neg e$ is unsat
  - Now assume $p_1, p_3, \neg e$ is the unsatisfiable core
  - Then it is unnecessary to check:
    - $p_1, \neg p_2, p_3, p_4 \Rightarrow e$
    - $p_1, \neg p_2, p_3, \neg p_4 \Rightarrow e$
    - $p_1, p_2, p_3, \neg p_4 \Rightarrow e$

Microsoft Research

Other Microsoft clients

- Model programs (M. Veanes – MSRR)
- Termination (B. Cook – MSRC)
- Security protocols (A. Gordon and C. Fournet - MSRC)
- Business Application Modeling (E. Jackson - MSRR)
- Cryptography (R. Venki – MSRR)
- Verifying Garbage Collectors (C. Hawblitzel – MSRR)
- Model Based Testing (L. Bruck – SQL)
- Semantic type checking for D models (G. Bierman – MSRC)
- More coming soon…

Future

Opening the “Black Box”

SMT solvers are collections of little engines.

They should provide access to these engines.

Users should be able to define their own strategies.

New application domains.

Conclusion

- SMT is hot at Microsoft.
- Many applications.
- Ideal: Programs that understand programs.
- Z3 is a very efficient SMT solver.
- http://research.microsoft.com/projects/z3

Thank You!