COSEINC®

*Solid Security.Verified.*

# Industrial Bug Mining

## Extracting, Grading and Enriching the Ore of Exploits

COSEINC®

Solid Security.Verified.

# The Bug Mining Analogy

- Phase 1: Extraction
- Phase 2: Grading
- Phase 3: Enrichment
- Phase 4: ???
- Phase 5: Profit!

COSEINC®

Solid Security.Verified.

# The Bug Mining Analogy

- Phase 1: Extraction
- Phase 2: Grading
- Phase 3: Enrichment
- Phase 4: ???
- Phase 5: Profit!

COSEINC®

Solid Security.Verified.

# Welcome to the Mt era

- 2009: We use 8 servers to build a virtualised fuzzfarm
  - Sustained testing speed: 30 t/s (2.5Mt / day )
- April 2010: MS describe their ‘fuzzing botnet’
  - “12 million iterations in a weekend” (6Mt/day)
  - Now upwards of 10Mt/d
  - 1000 – 2000 machines at peak opt-in ( ? )
- May 2010:  Project MAN VERSUS BORG!!11! (Bugmine 2.0)
  - Same hardware, complete stripdown and rebuild
  - 8 machines, 64 cores
  - Test and optimise code / architecture at every step
  - Sustained testing speed:  >= 1.12Mt/h

COSEINC®

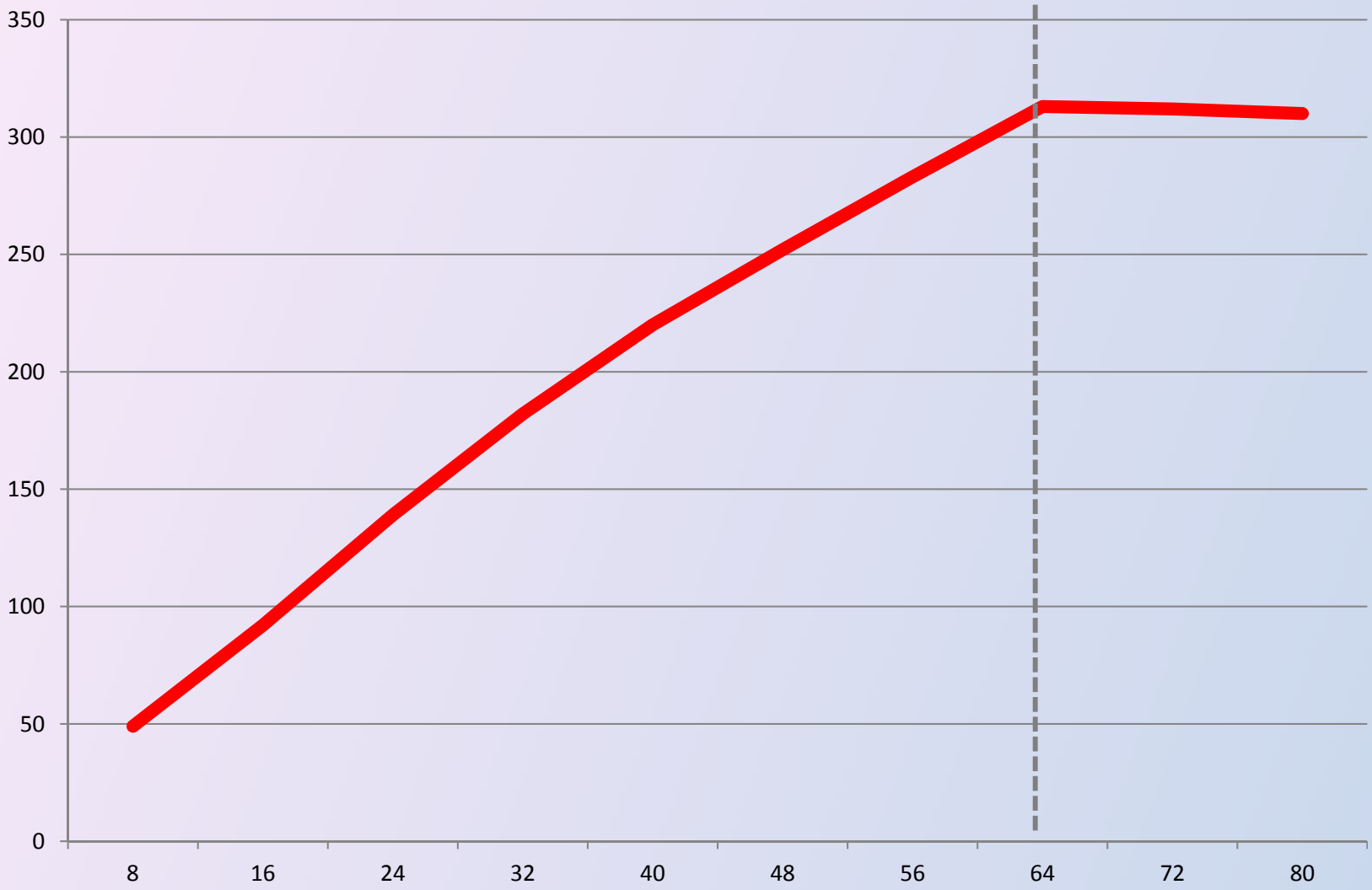
Solid Security.Verified.

# Scale

1. Make each node faster by eliminating bottlenecks
   - network, disk, IO, serialisation, extraneous target code, node OS overhead….
   - You're not doing it right until the last bottleneck is CPU time spent on the real target code
2. When adding new nodes, scale as close to perfectly as possible

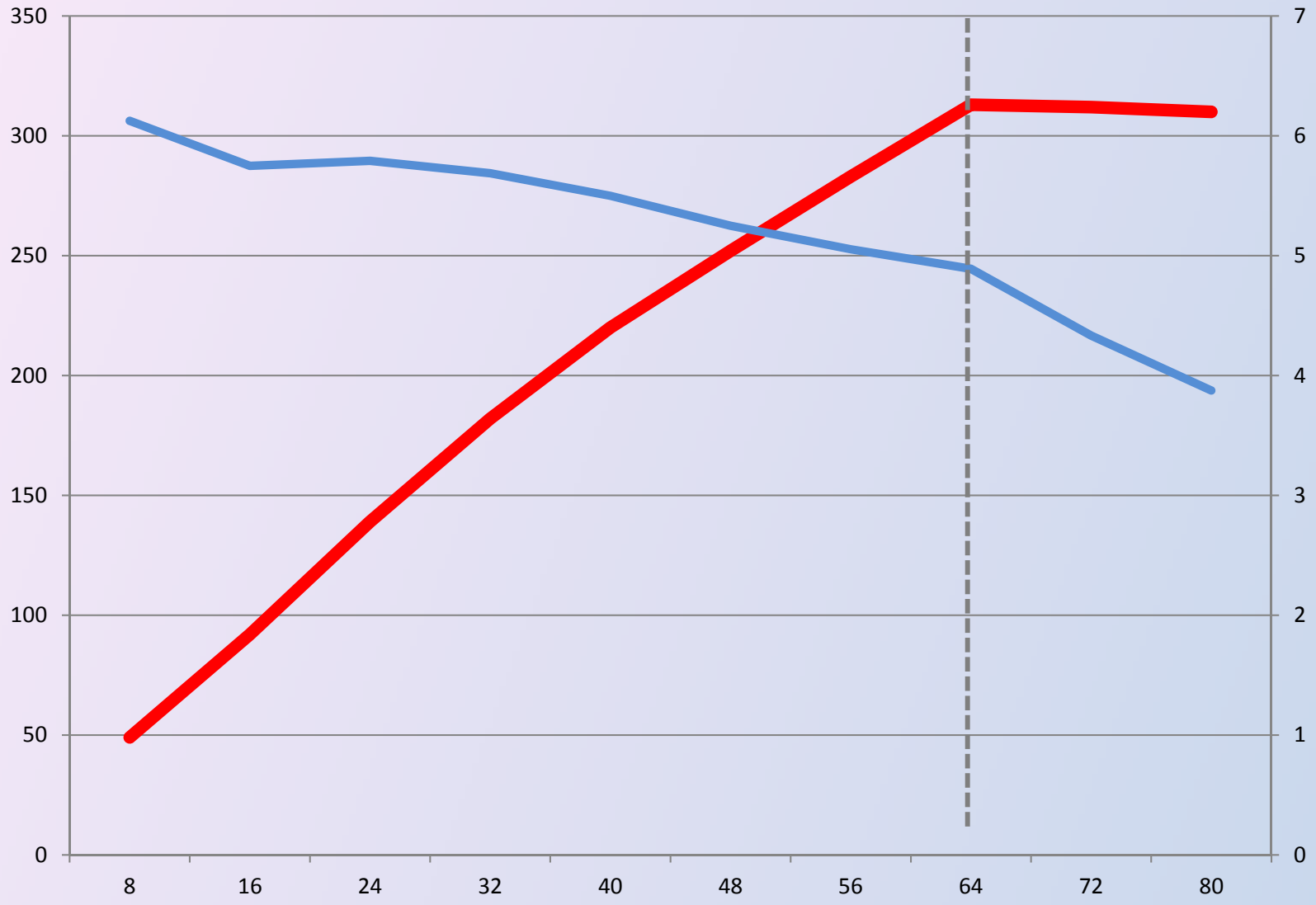
COSEINC®

*Solid Security.Verified.*



COSEINC®

*Solid Security.Verified.*



COSEINC®

Solid Security.Verified.

# Building It

- Switch from ESXi to KVM
  - Real linux, we know how use it
  - Performance is apparently ‘comparable’
- Move storage to a dedicated network
  - Open iSCSI, 4 x 160GB SSD in RAID 0, 4xGigE NIC
  - Oracle Cluster Filesystem (OCFS2) on top
- Optimise Harness
  - Ruby is slow anyway, but I removed the worst problems
- Optimise Fuzzbots
  - Kill explorer.exe for ~15% speedup?!
  - Don’t open a brand new Office process every time

COSEINC®

Solid Security.Verified.

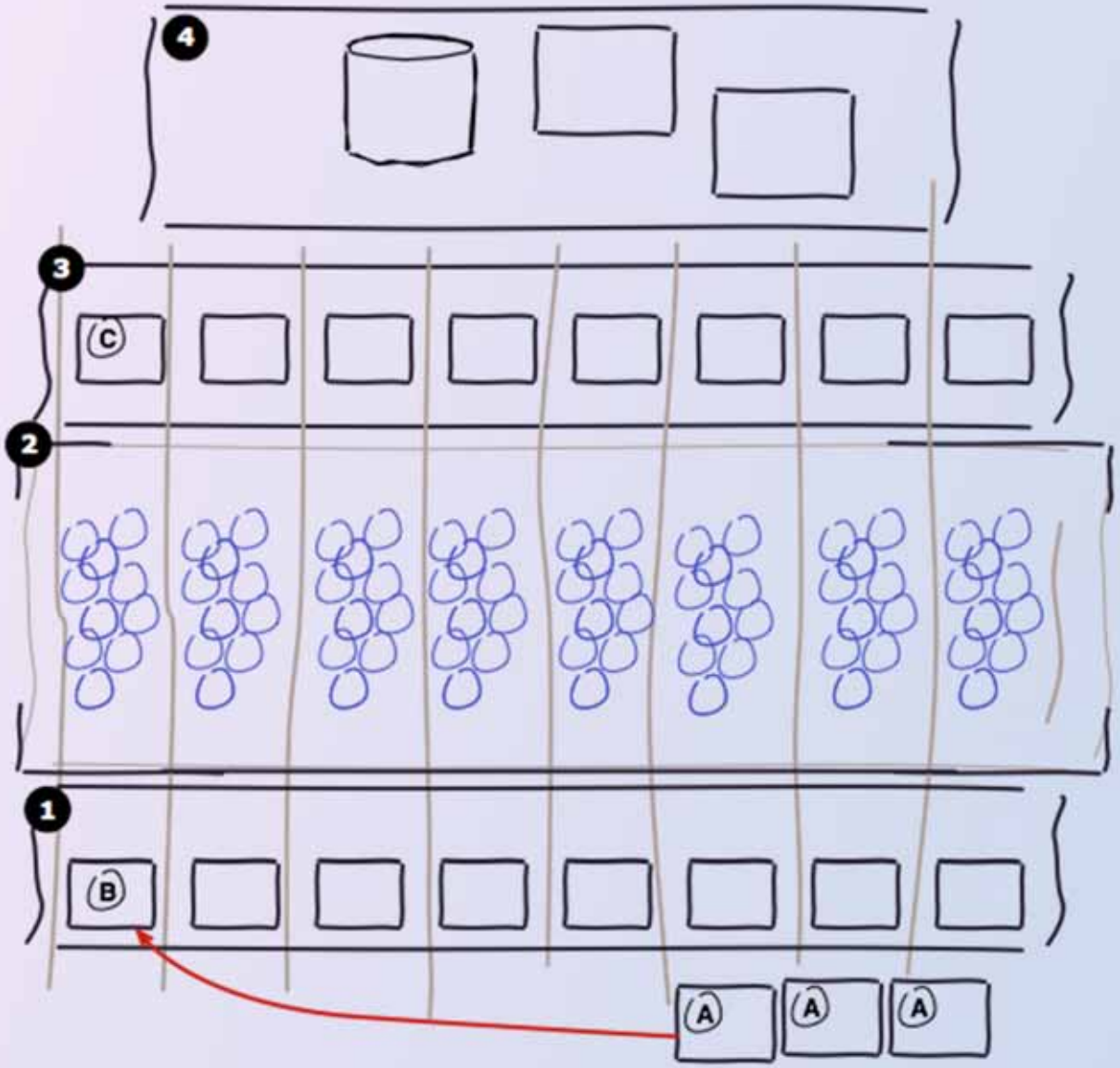
# Building It

- Easier Provisioning
  - One fuzzbot template
  - Multiple “overlays” (aka “linked clones”)
  - “Snapshot Mode” on top of that
  - Template changes and new rollouts happen in minutes.
- Easier and more powerful management
  - … assuming you like bash and ssh

- Total Software Cost $0 (using MSDN licenses)
- Total Hardware Cost ~ 30k USD

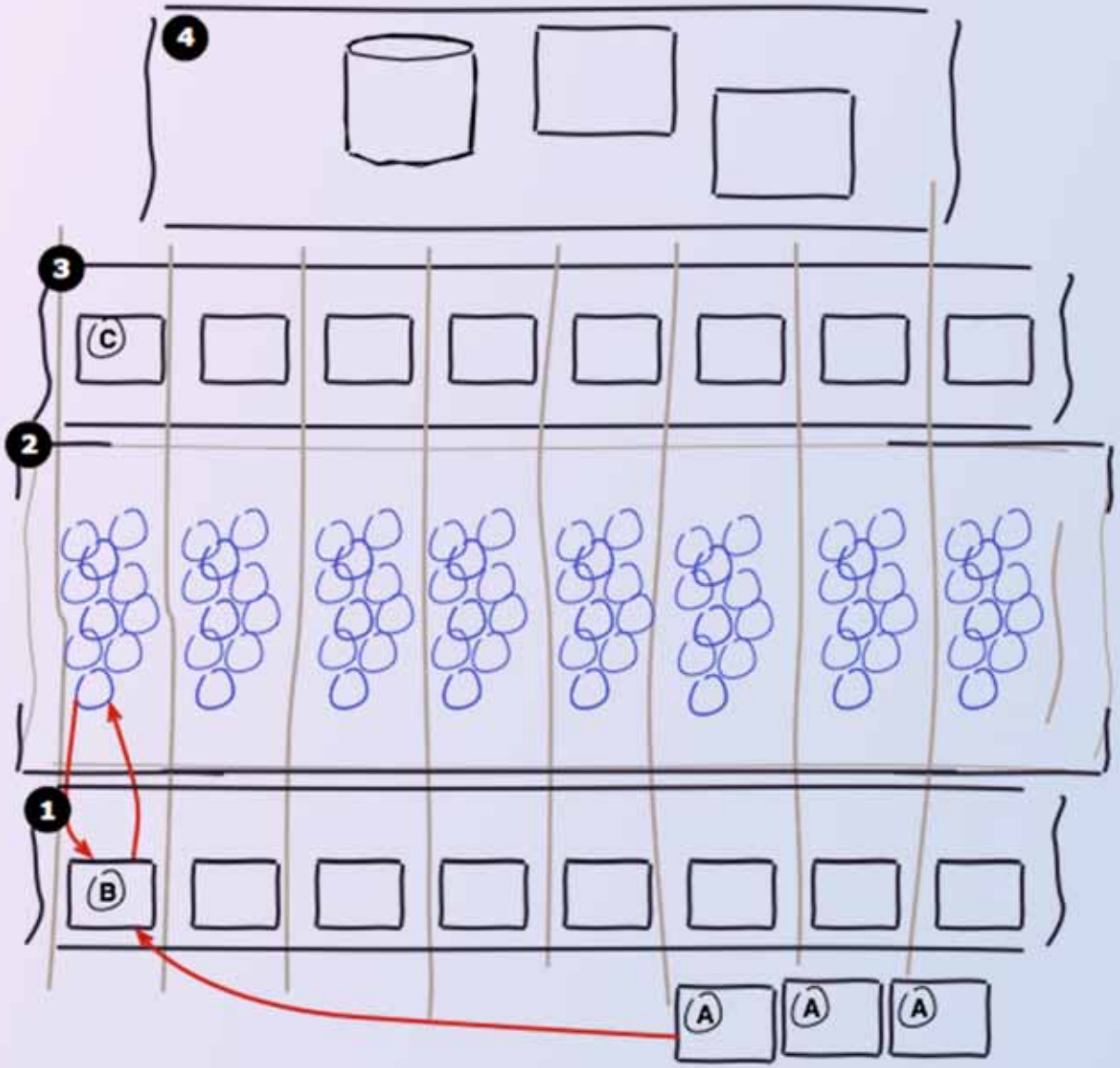
COSEINC®

*Solid Security. Verified.*



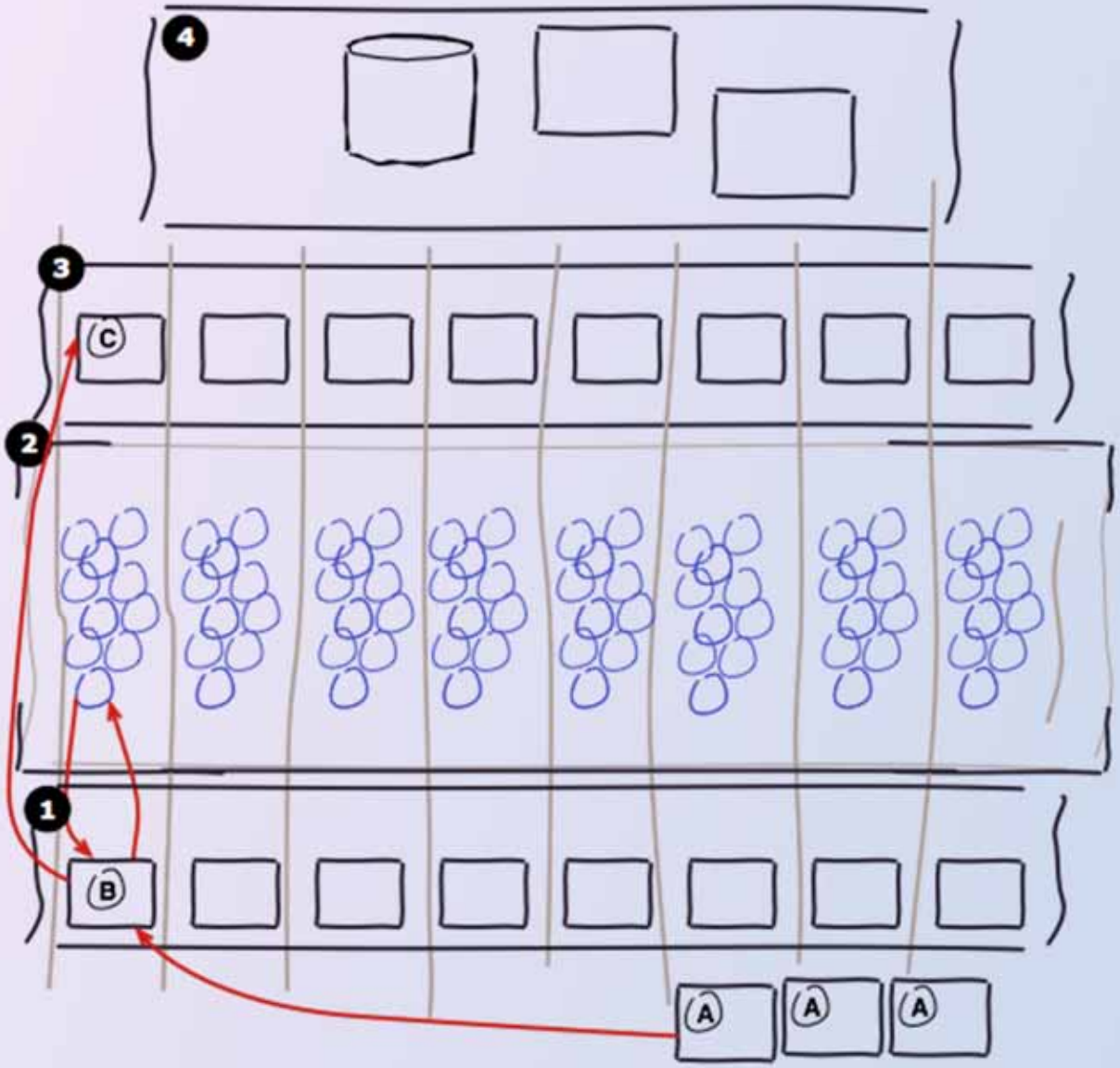
COSEINC®

*Solid Security.Verified.*



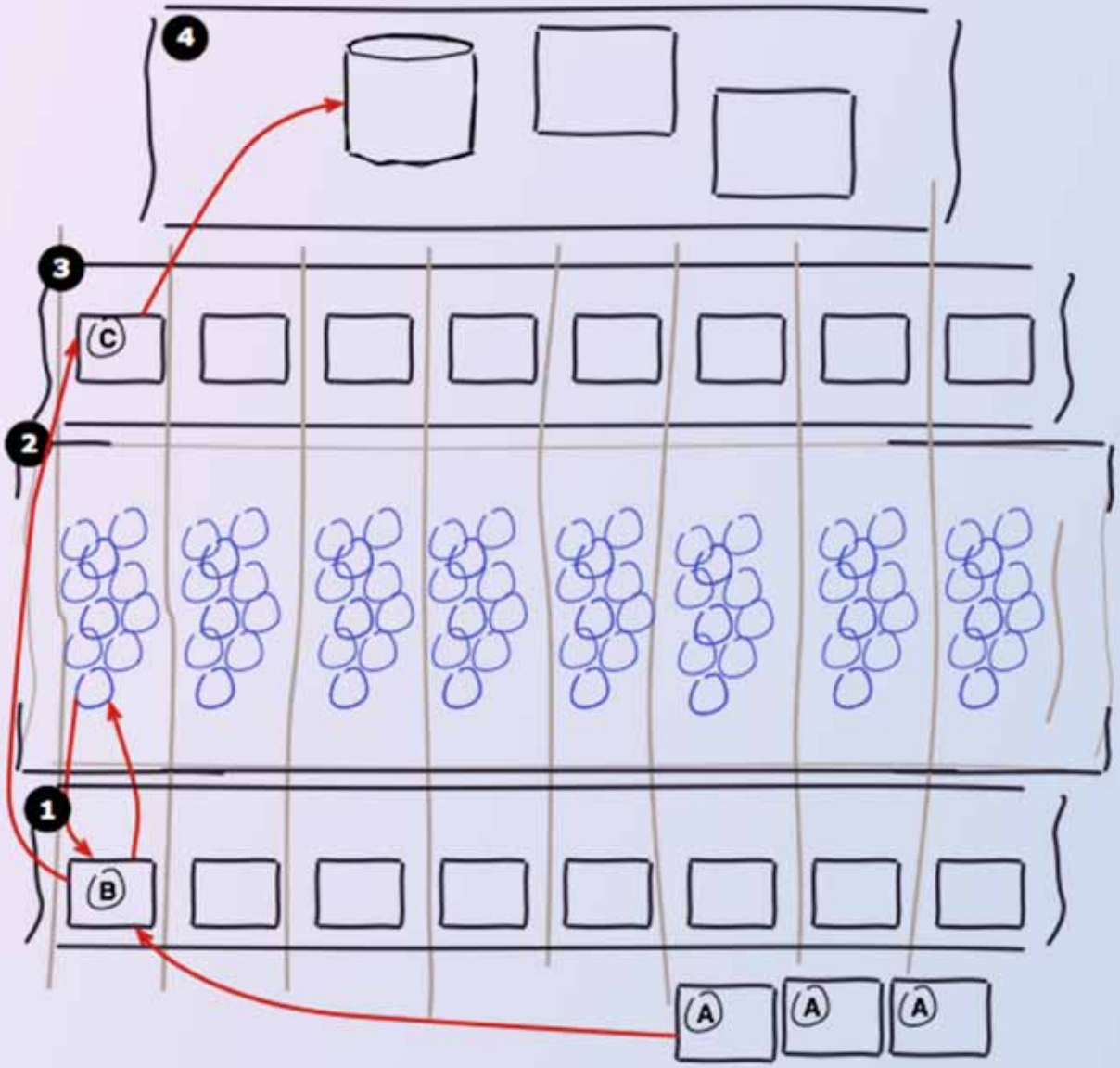
COSEINC®

*Solid Security.Verified.*



COSEINC®

*Solid Security.Verified.*



COSEINC®

Solid Security.Verified.

# Features

- Software “Hot swap”
- Tagged queues
- Any DB backend
- Any case producer frontend
- Everything scales horizontally – n producers, n distribution nodes etc etc

COSEINC®

Solid Security.Verified.

# The Bug Mining Analogy

- Phase 1: Extraction
- Phase 2: Grading
- Phase 3: Enrichment
- Phase 4: ???
- Phase 5: Profit!

COSEINC®

Solid Security.Verified.

# Bug Triage

- There is “exploitable” and EXPLOITABLE.
- !exploitable rocks, but not for this.
- Here are some examples from Word 2007

```
head -15 summary.txt
=========SUMMARY===============
<none?>: 229
total: 59965
PROBABLY NOT EXPLOITABLE: 21409
UNKNOWN: 31013
PROBABLY EXPLOITABLE: 6032
EXPLOITABLE: 1282
621 Buckets. 373 unique EIPs.
<none?>: 1
EXPLOITABLE: 88
UNKNOWN: 336
PROBABLY NOT EXPLOITABLE: 86
PROBABLY EXPLOITABLE: 110
================================
```

# Bug Examples

```
--- 0x11293429.0x5a33235d (count: 4) ---
EXPLOITABLE: Exploitable - User Mode Write AV starting at
mso!Ordinal7111+0x233 (Hash=0x11293429.0x5a33235d)
326c4726 885e21          mov     byte ptr [esi+21h],bl
ds:0023:32688488=c3
eax=c9330048 ebx=00000000 ecx=32688467 edx=00000000 esi=32688467
edi=00000000 eip=326c4726 esp=001252e8 ebp=001252fc
```

Potentially overwrite a byte with null. Then what?

# Bug Examples

```
--- 0x4f145f2f.0x51025f72 (count: 29) ---
PROBABLY EXPLOITABLE: Probably Exploitable - Read Access Violation
on Control Flow starting at wwlib!wdGetApplicationObject+0xaf49f
(Hash=0x4f145f2f.0x51025f72)
31a46cad ff5028          call    dword ptr [eax+28h]
ds:0023:00000029=????????
eax=00000001 ebx=00000000 ecx=022b6590 edx=00121b1c esi=001218b0
edi=06440000 eip=31a46cad esp=001210d0 ebp=00122140
```

Only awesome if eax is controlled as a 32 bit value…

# Bug Examples

```
--- 0x43317a27.0x3f100938 (count: 1) ---
EXPLOITABLE: Exploitable - Read Access Violation on Control Flow
starting at wwlib!FMain+0x10fcb3 (Hash=0x43317a27.0x3f100938)
315adb58 ff5004          call    dword ptr [eax+4]
ds:0023:b4b4b4b8=????????
eax=b4b4b4b4 ebx=00000000 ecx=01f8cf6c edx=0012e6a8 esi=01f8cf6c
edi=06e2366c eip=315adb58 esp=0012e674 ebp=0012e680
```

… like this

COSEINC®

Solid Security.Verified.

# Bug Examples

```
--- 0x15045215.0x15045251 (count: 3) ---
PROBABLY NOT EXPLOITABLE: Read Access Violation near NULL
starting at wwlib!DllGetLCID+0xdb8e4
(Hash=0x15045215.0x15045251)
3213a302 d7              xlat    byte ptr [ebx]
ds:0023:00000000=??
eax=00120000 ebx=00000000 ecx=01efff68 edx=0012ca20 esi=01ef9568
edi=07971bec eip=3213a302 esp=01efa093 ebp=fff501f8
3213a302 xlat byte ptr [ebx]
3213a303 std
3213a304 fdivr st,st(5)
3213a306 fscale
```

# !exploitable fail

COSEINC®

Solid Security.Verified.

# Bug Examples

```
eax=00000000 ebx=00000000 ecx=07a4e008 edx=07a4e440
esi=07a4e43c edi=00000003 eip=07a4e000 esp=0012f650
ebp=0000000d iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010246
07a4e000 0000              add     byte ptr [eax],al
ds:0023:00000000=??
EVENT:DEBUG_EVENT_EXCEPTION
e06d7363 Exception in winext\msec.dll.exploitable debugger
extension.
      PC: 7c812afb  VA: 0006d098  R/W: 19930520  Parameter:
10026bfc
```

## EPIC !exploitable fail

COSEINC®

Solid Security.Verified.

# Bug Examples

```
--- 0x7e7b062c.0x2159673f (count: 16) ---
EXPLOITABLE: Exploitable - Read Access Violation at the
Instruction Pointer starting at Unknown Symbol @ 0x0
01010101 ??  ???
eax=06510000 ebx=0012dc14 ecx=064f56a8 edx=00000000 esi=001297cc
edi=00000000 eip=01010101 esp=00125320 ebp=00129724
```

!exploitable needs a new ‘KACHING’ classification

COSEINC®

Solid Security.Verified.

# Bugmine Futures

- Template mining / selection ( Prospector )
- Initial auto-analysis of crashes (Foundry )
  - Taint
  - Influence
  - Bug localisation
- Fancy pie charts / DB stuff ( Dashboard )
- New hardware ( 48 cores per box )
- Better crash repro / binning

COSEINC®

Solid Security.Verified.

# The Bug Mining Analogy

- Phase 1: Extraction
- Phase 2: Grading
- Phase 3: Enrichment
- Phase 4: ???
- Phase 5: Profit!

COSEINC®

Solid Security.Verified.

# Curse you, Aitel!

**“Take each basic block and number it. Execute the program twice, once with your crashing file, and once with your template. This generates two signals, which have a stream of numbers in them (from the execution trace). Then you can do interesting things[…]”**

**-- Dailydave ML, 6/8/09**

COSEINC®

Solid Security.Verified.

# Curse you, Aitel!

**“I'm not sure what the interesting thing here is that magically tells you something is worth really digging into? Maybe you take your two signals, and subtract their frequencies and visualize how different they are? Throw that at a HMM/NN and make it tell you something?”**

**-- Dailydave ML, 6/8/09**

COSEINC®
Solid Security.Verified.

# Step 0 - Runtrace

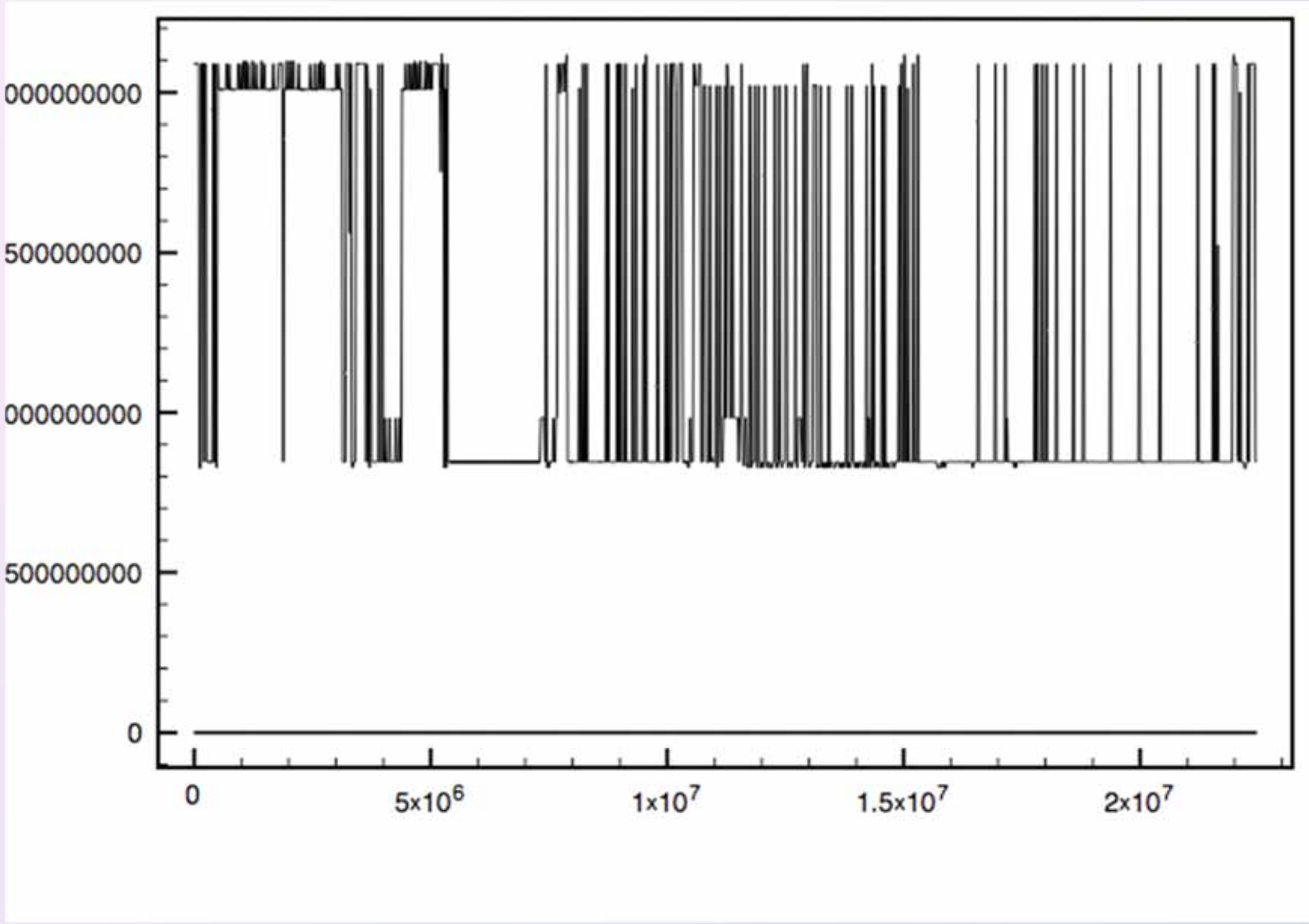
- We initially used DynamoRIO
  - VERY fast, Clean API
  - Kind of a pain in the butt to work with (cmake etc)
  - Mysteriously just stopped working on our XP tracer
- Now we're using Pin (http://www.pintool.org)
  - Strange API, slower
  - More reliable, easier to build tools, crossplatform
- (All the runtrace side was written by The Grugq)

COSEINC®

Solid Security.Verified.

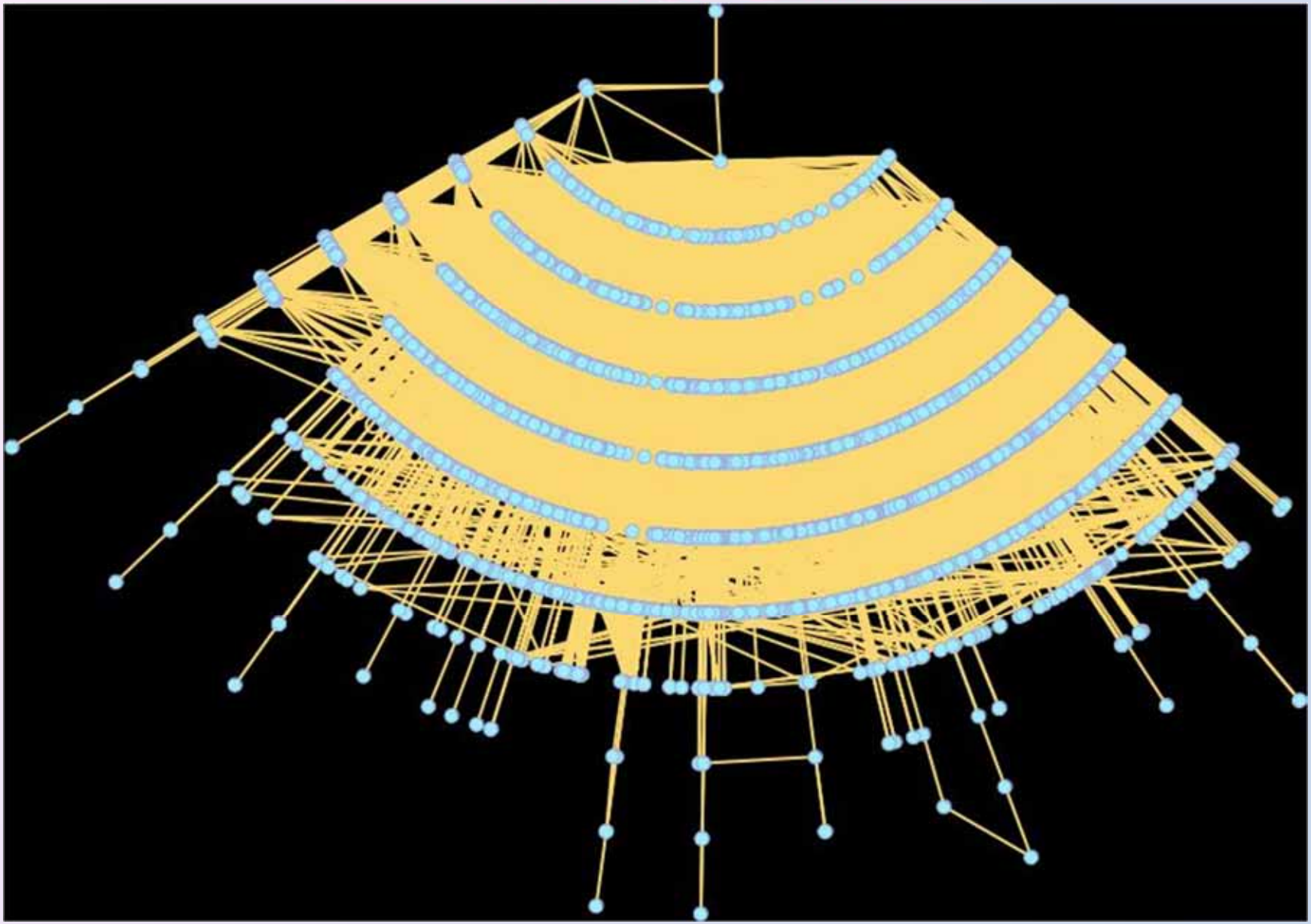
# Problems

- The ‘streams of numbers’ are really long
- FFT and HMM etc were red herrings
- NN is buzzword bingo
- Visualisation turned out not to be useful



COSEINC®

*Solid Security.Verified.*



COSEINC®

Solid Security.Verified.

# How about ‘diff’ ?

- But the sequences are crazy long
- Humans can’t read them effectively
- Heuristics for ‘this looks weird’ are hard.

IDEA – code paths are very structured….

COSEINC®

Solid Security.Verified.

# Heirarchical Grammars

pease porridge hot
pease porridge cold
pease porridge in the pot
nine days old
some like it hot
some like it cold
some like it in the pot
nine days old

The SEQUITUR Algorithm
http://sequitur.info/
Written by
Craig Nevill-Manning, Rutgers University,
Ian Witten, University of Waikato, New Zealand

COSEINC®

*Solid Security.Verified.*

```
0 -> 1 2 3 4 3 5 6 2 6 4 6 5 \n
1 -> p e a s 7 r r i d g 8
2 -> h o t
3 -> \n 1
4 -> c 9
5 -> 10 _ t h 7 t \n n 10 8 d a y s _ 9
6 -> \n s o m 8 l i k 8 i t _
7 -> 8 p o
8 -> e _
9 -> o l d
10 -> i n
```

The SEQUITUR Algorithm
http://sequitur.info/
Written by
Craig Nevill-Manning, Rutgers University,
Ian Witten, University of Waikato, New Zealand

COSEINC®
Solid Security.Verified.

# Problems

- The grammar is generated on the fly, so different files will generate different grammars
- Wrote a ‘recompressor’
  - Convert the grammar to a ‘Trie’
  - Use the Trie to apply a selected grammar to a selected sequence

COSEINC®

Solid Security.Verified.

# The approach

- Take transitions addrA ->addrB
- Store in a DB, use the tuple index as sequence elements
- Recompress the original and variant TIS
- Diff
- Post process the diff to add some interesting info

COSEINC®

Solid Security.Verified.

# Annoying, hard to read green screen demo…

COSEINC®

*Solid Security.Verified.*

# Beer!

But questions may be asked first.

ben at coseinc dot com