A Record/Playback method for simulating a device-under-test (DUT)

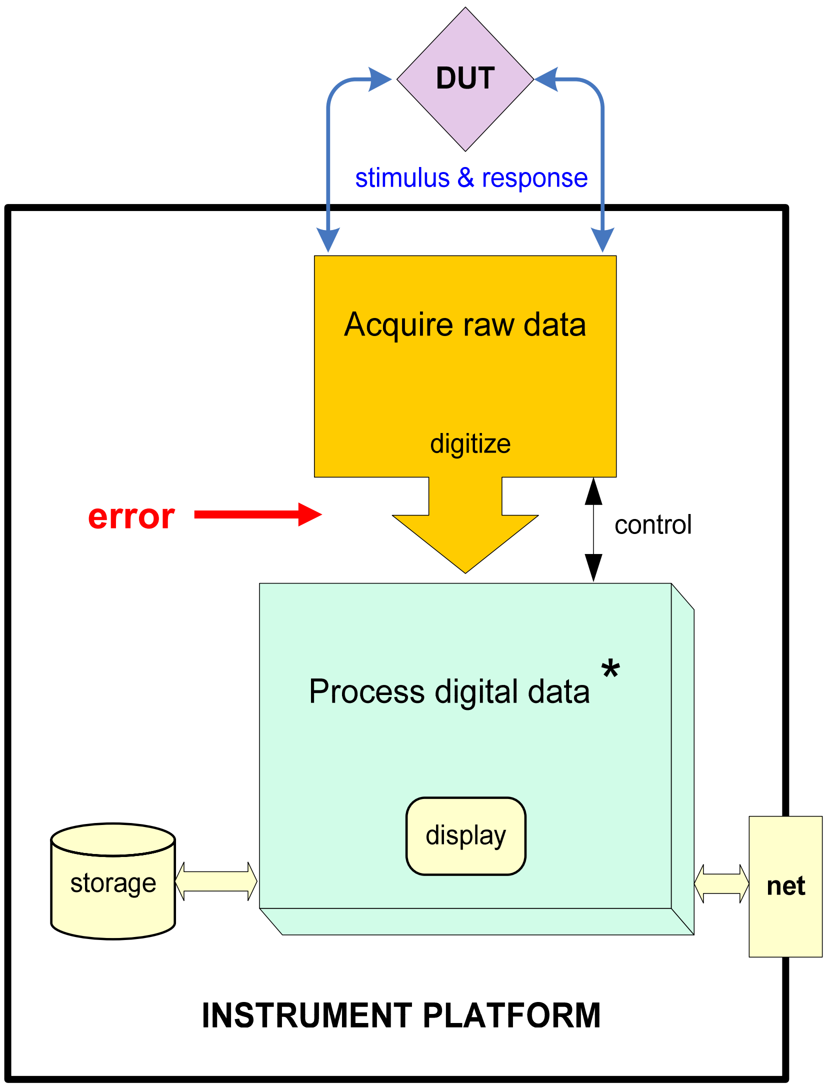
- David Bozarth (M.S. Candidate)
- Sue Wood (Product Development Engineer)

Contemporary test & measurement platforms acquire physical signal samples and process these digitally. The processing software of such an instrument must be carefully designed and tested. Sample error contributions propagate through digital processing stages, introducing uncertainty that can be costly to remedy or work around during performance testing of the instrument software.

We describe a method for mitigating this uncertainty, and we outline the design of a working software system that employs the method.

The method also suggests a potential side benefit: elimination of costly instrument hardware used in software development, by moving software testing activities onto general-purpose PC platforms that run the instrument processing software.

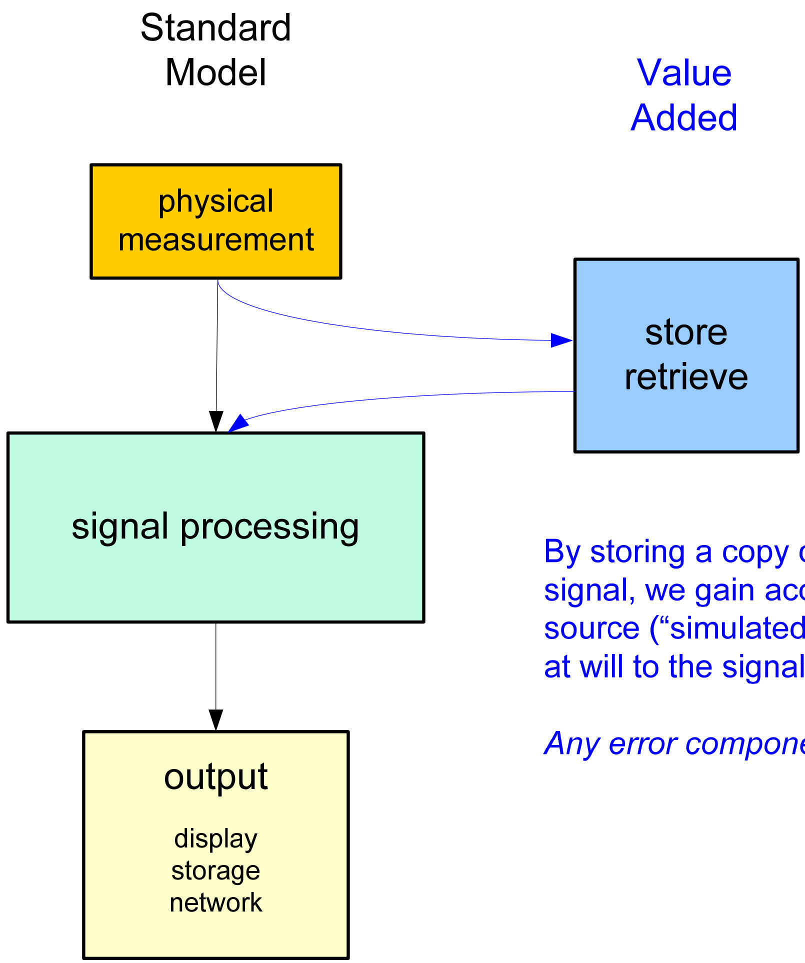
Test & Measurement
Basic Setup



-$- PROBLEMS TO ADDRESS -$-

- **instrument software must be tested** *
- **repeatable data sets for software testing**
  - **random & systematic error**
  - **expensive setup/calibration**
  - **store only the processed data!**
- **cost & bulk of the instrument platform**

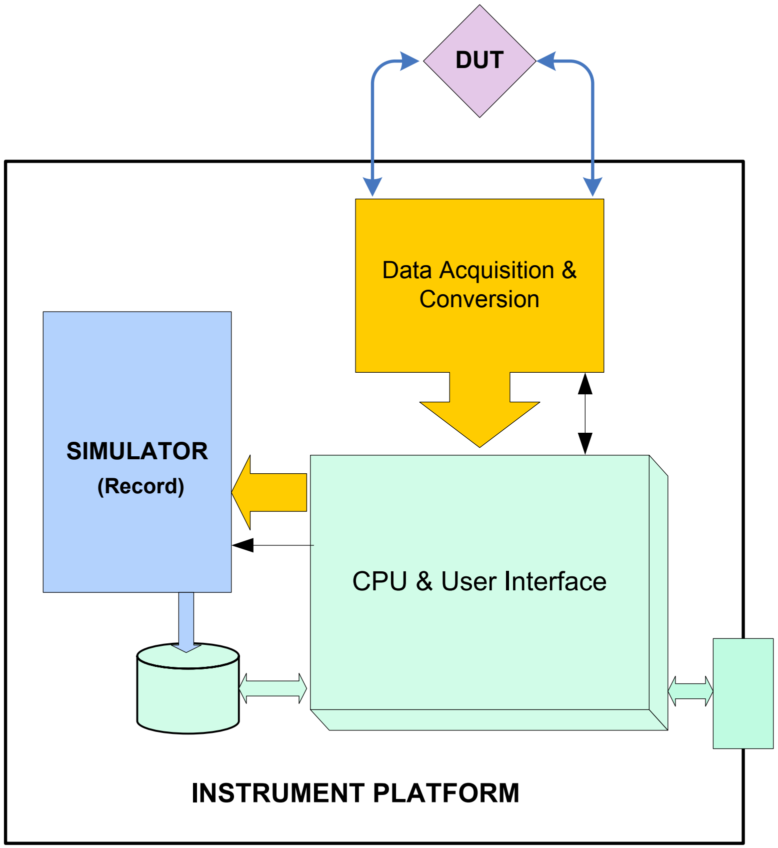
A simple idea for reducing costs and gaining leverage in testing



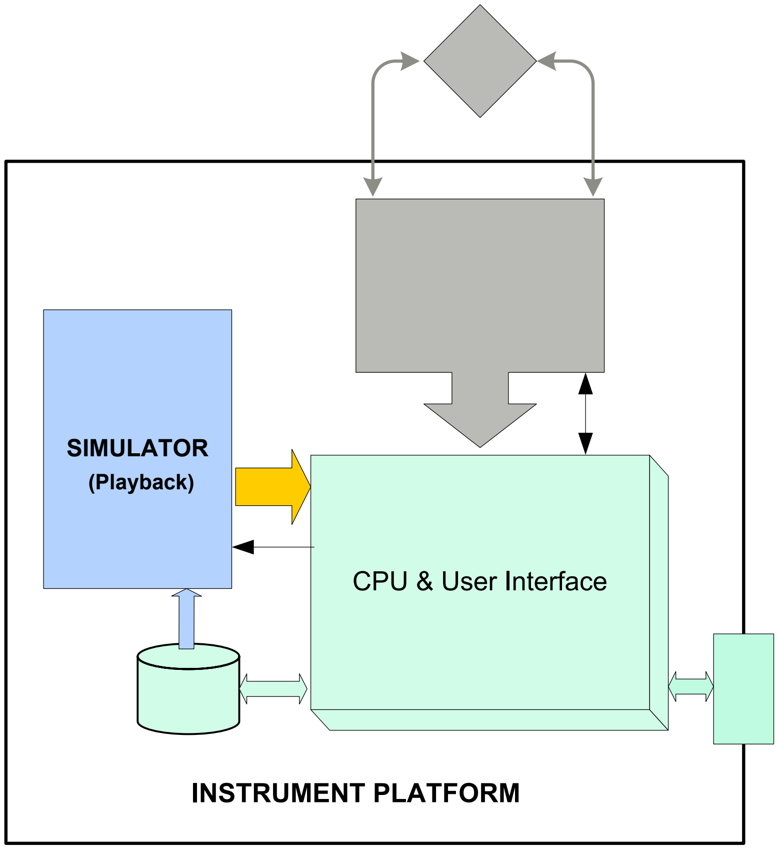
By storing a copy of the freshly digitized raw signal, we gain access to an invariant signal source ("simulated DUT") that can be applied at will to the signal processing system.

Any error components are also invariant.

Record/Playback Simulator

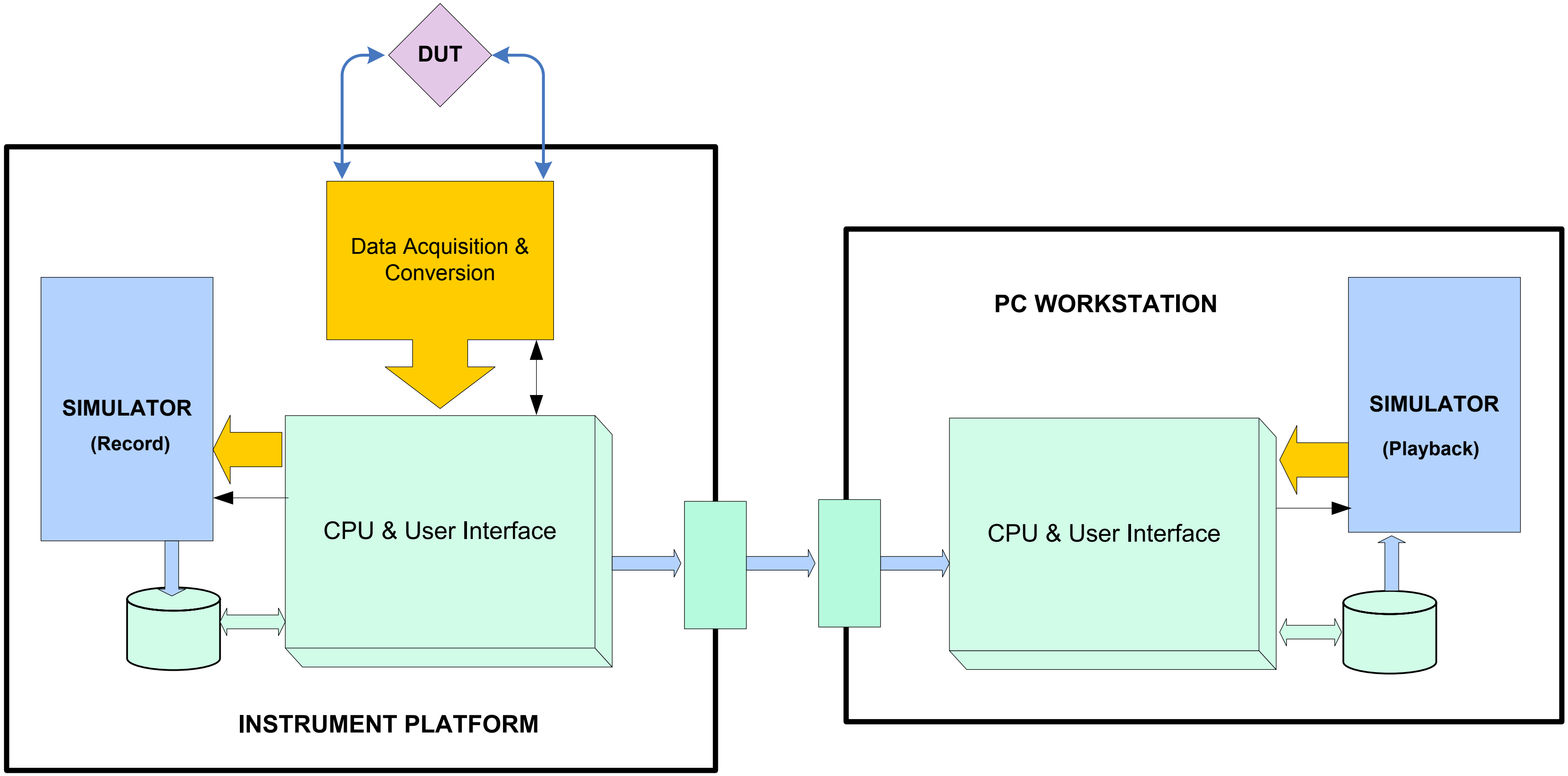


Record Mode



Playback Mode

Playback of recorded samples by "Virtual Instrument"



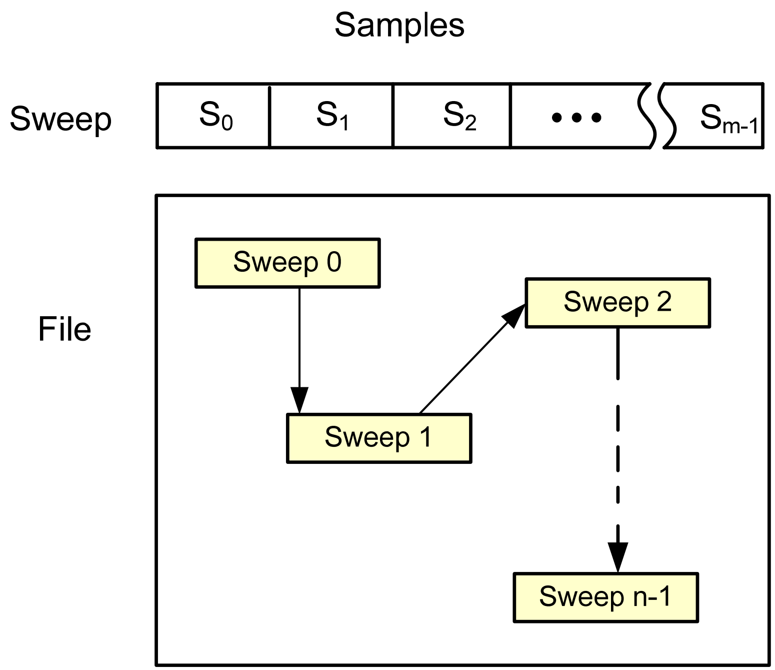
Glossary of Terms

Sample	collection of signal sample values that represents a single atomic measurement event (data point)
Sweep	ordered sequence of Samples associated with a sweep event performed by the instrument hardware
Simulator	design or implementation of a Record/Playback Simulator software system
Instrument	system that runs software, interfaces with a Simulator, and processes Samples
State	Simulator-dependent subset of Instrument state (variables that are critical for Simulator performance)
Operator	human or program control of Instrument & Simulator

Basic Requirements

- collect Samples from DUT
  - stimulus and response signals
  - include all noise & drift
- store Samples in a file
  - portable
  - editable
  - algorithmic model
- retrieve and playback Samples
  - target Instrument
  - Virtual Instrument

Sample - Sweep - File

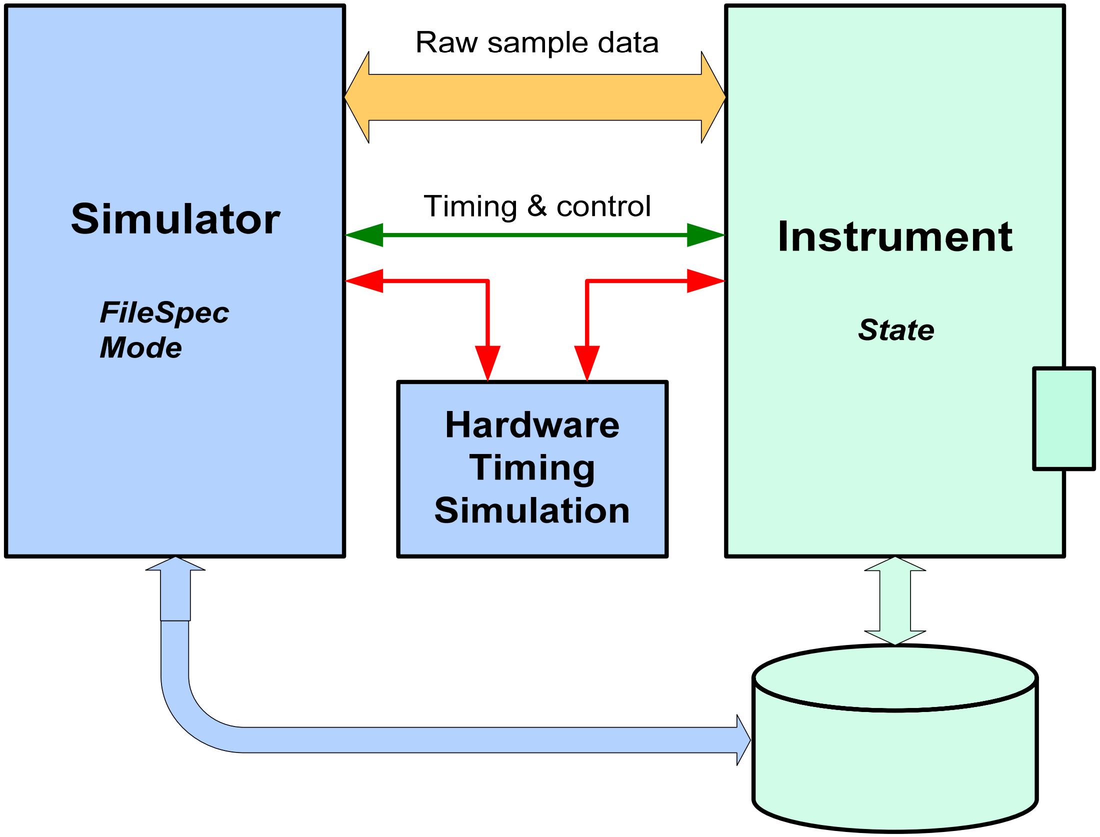


A Sweep is a sequence of Samples.
A File is a container of Sweeps ordered sequentially.

Practical concerns

- State at time of Playback may have to match record-time State
  - scaling of independent variable
  - interpretation of Samples
- Virtual Instrument may require limited hardware emulation
  - timing of events
  - user interface
- Samples - raw value may not have explicit real-world meaning
  - mathematical transformation
  - modeling or simulation

Simulator-Instrument architecture



Timing & control path for Virtual Instrument shown in red.

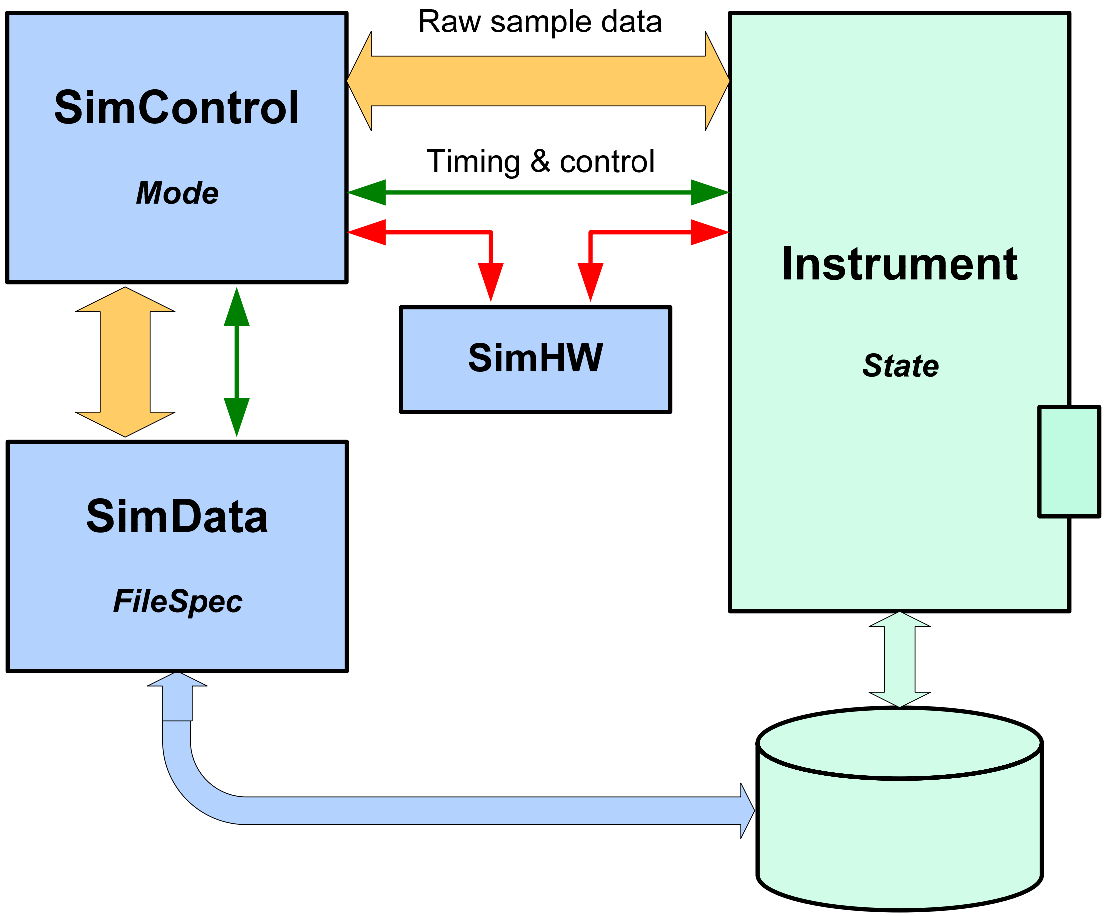
The fundamental use case

- A physical instrument has State A.
  Operator addresses this instrument.
- Operator sets Simulator FileSpec = Z.
- Operator sets Simulator Mode = Record.
- Instrument performs a sequence of n sweep events,
  processing Samples from the SUT.
- Operator sets Simulator Mode = Off.

…

- Some Instrument has State A and can access file Z.
  Operator addresses this instrument.
- Operator sets Simulator FileSpec = Z.
- Operator sets Simulator Mode = Playback.
- Instrument performs a sequence of n sweep events,
  processing Samples from file Z.
- Operator sets Simulator Mode = Off.

Simulator-Instrument architecture …

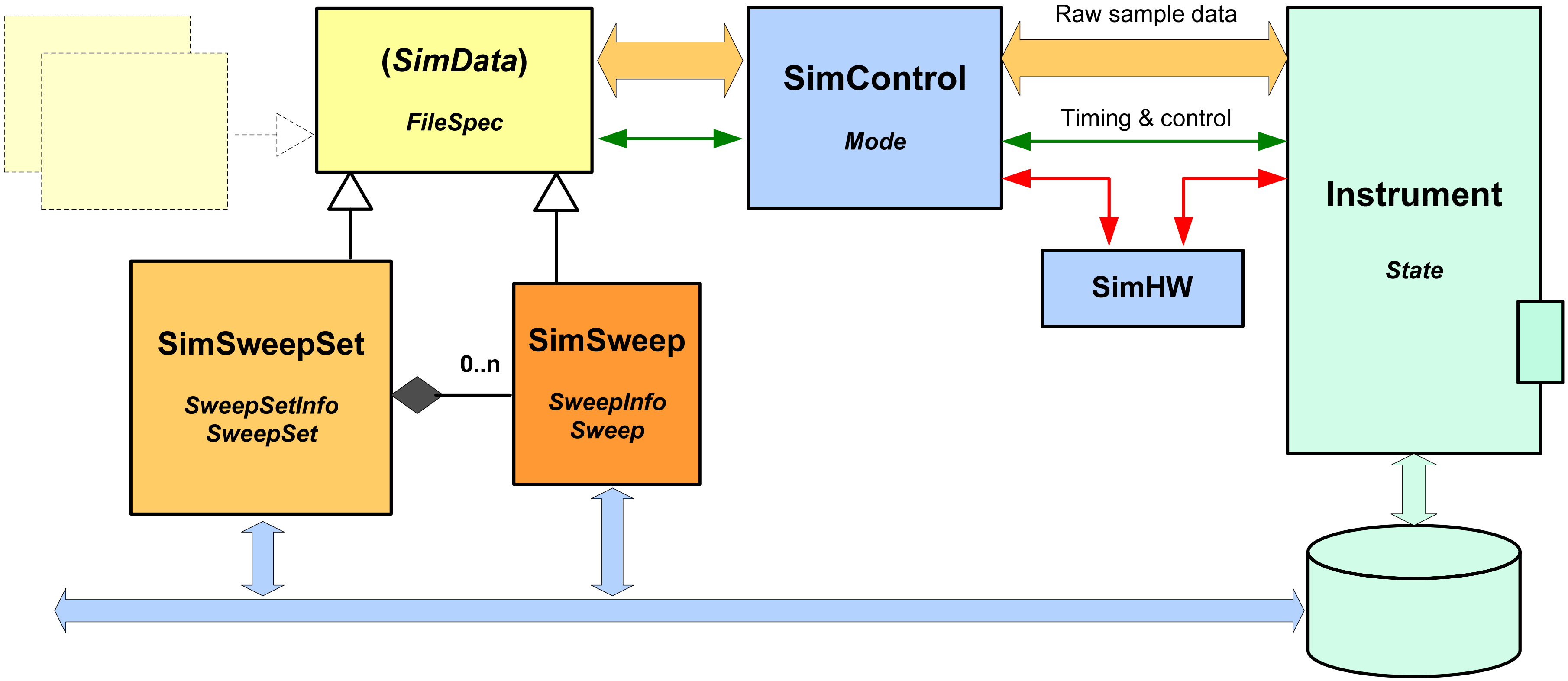


… with internal partition of Simulator.

Simulator functional blocks

- SimControl
  - stream buffering
  - multiplexing
  - state sequencing
- SimData
  - data formatting
  - file system operations
- SimHW
  - simulates hardware signals
  - used for Virtual Instrument

Simulator-Instrument architecture …



… with SimData class relationships.

SimData class family features

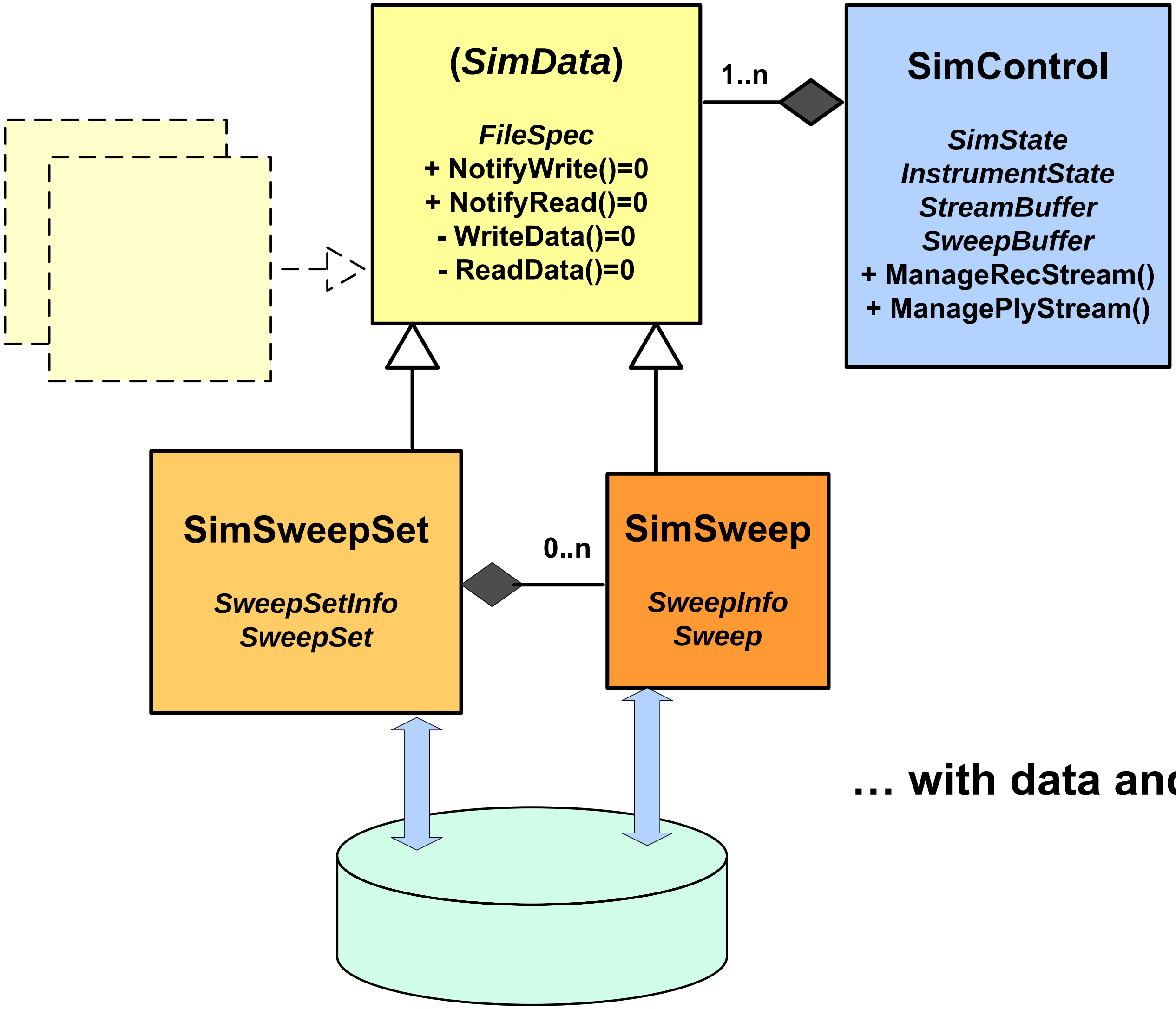
- SimData
  - abstract base class
  - derived container classes
  - polymorphic to SimControl
  - info plus data
  - self-contained read/write

- SimSweep
  - most basic: one Sweep
  - basic currency of file management

- SimSweepSet
  - array of SimSweep objects
  - models one File

- new containers …
  - contain any other SimData class(es)
  - info plus data: encapsulation

SimData class relationships …

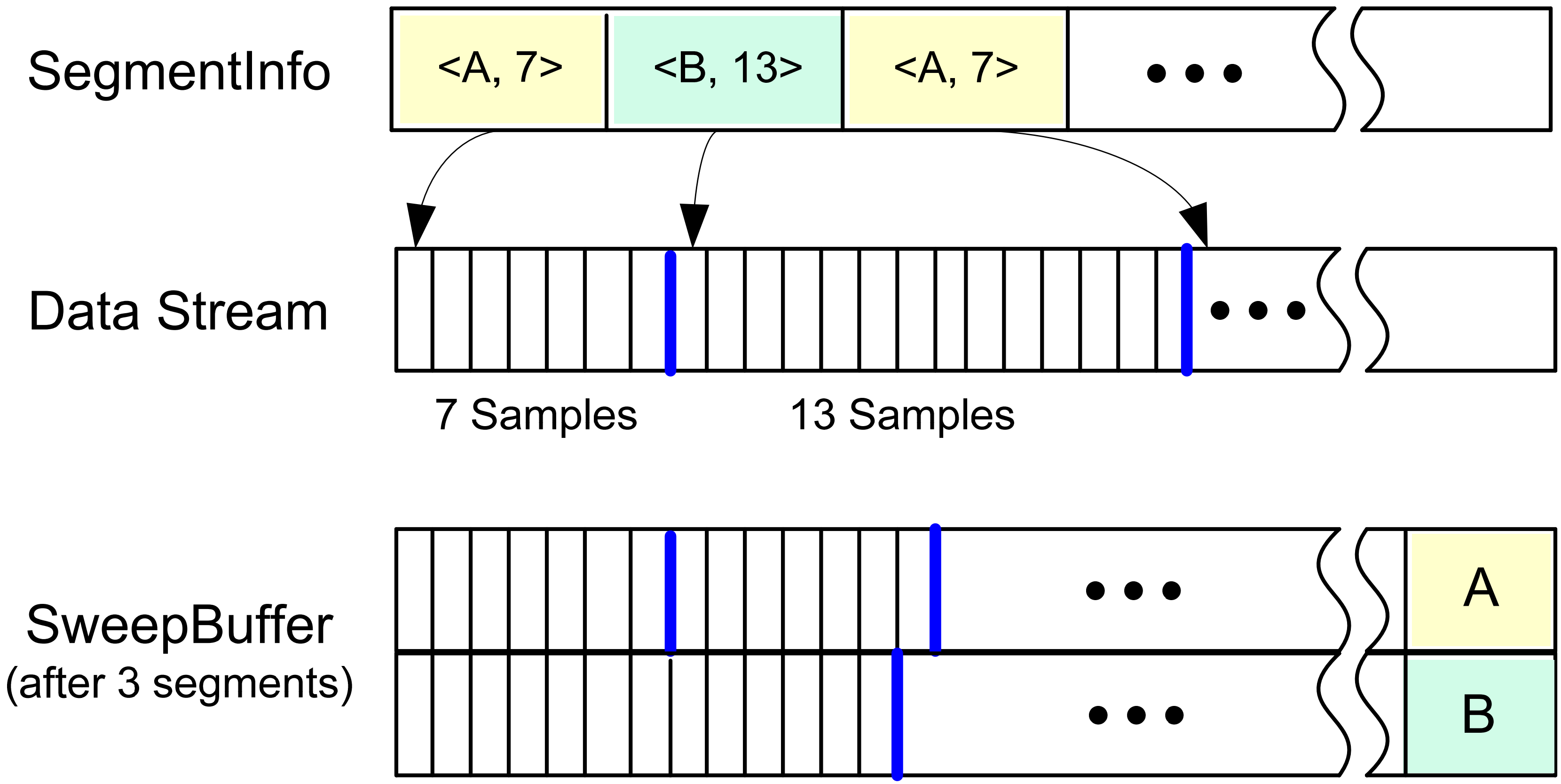


… with data and methods.

Software development environment

- C++ language using Standard Template Library (STL)
  - Microsoft Windows XP application
  - embedded target
  - large legacy code base
- Component Object Model (COM)
  - interprocess communication
  - address remote objects via C++ or VB Script
  - build custom interface for Operator controls
- Microsoft Visual Studio 2005
  - unmanaged C++ code
  - Whole Tomato Visual Assist X
- IBM Rational Clear Case
  - source code management
  - version & release control
- TWiki – open source knowledge management platform

Decoding a Sample stream



File, data structure & container issues

- inefficiency
  - info plus data → repetitive info
  - space and time budgets
  - write flags

- container internal structure
  - LocationIndex: vector of integers
  - iterator, insert, delete, modify
  - write flag

- alternate file format
  - native binary file
  - primary & alternate file specifiers
  - flag or enum
  - write flags

Does it work?

✓ stream of raw Samples

- tap from Instrument in Record Mode
- input to Instrument in Playback Mode
- Yes or No – either they match or not!

✓ processed signal derived from Samples

- real-world meaning
- simple to test
- store original output while Recording
- compare with output from Playback

Effect of noise on derived signal

trace.sep.org.txt - Not...

File Edit Format View Help

```
BEGIN
1.3104224,-0.23653361
0.50342727,-0.81367773
-0.51052684,-0.62249565
-0.78961253,0.38799244
-0.026279878,1.1178192
0.87351203,0.64004916
0.72115666,-0.51550376
-0.28145111,-0.8485263
-1.0460054,0.023354081
-0.50791025,0.91967368
0.53594887,0.72294939
0.81327432,-0.29423821
-0.026409611,-0.92370147
```

```
-0.50053614,-0.015068259
-0.21460229,0.52414864
0.24779847,0.47617948
0.5197506,0.007957221
0.22562358,-0.50701249
-0.58412665,-0.47113314
-0.51762193,0.24182753
-0.068772323,0.47582954
0.26167494,0.10934047
0.078488499,-0.2507205
-0.16116427,-0.081975222
0.032307241,-0.015149697
0.017169464,-0.22985497
END
```

trace.sep.new.txt - No...

File Edit Format View Help

```
BEGIN
1.3147618,-0.23310187
0.50286126,-0.81359589
-0.50967747,-0.62243837
-0.78950858,0.38712472
-0.025756406,1.1181841
0.87281156,0.64105815
0.72209156,-0.51606119
-0.28311563,-0.84895754
-1.0457363,0.02344471
-0.50775951,0.91948533
0.5353753,0.72259963
0.81203079,-0.29397655
-0.026501229,-0.92343915
```

```
-0.49977401,-0.015326727
-0.21460439,0.52523047
0.2471875,0.47673726
0.52001768,0.0073199137
0.22591124,-0.50549918
-0.58326757,-0.47005546
-0.51798505,0.24156898
-0.069592677,0.47554135
0.26116556,0.10798951
0.078184038,-0.25160921
-0.16142903,-0.082506768
0.032701604,-0.014436521
0.0179208,-0.22930118
END
```

Signal from Record and from Playback

trace.org.txt - Notepad

File Edit Format View Help

```
BEGIN
1.3085769,-0.23345217
0.50325823,-0.81402045
-0.50998622,-0.62219727
-0.78982878,0.38733667
-0.025088435,1.1177561
0.87216932,0.64021963
0.7217322,-0.51608777
-0.28219843,-0.84778357
-1.0469918,0.024442753
-0.50850981,0.92023557
0.53549159,0.72177929
0.81221962,-0.294027
-0.025584033,-0.92443269
```

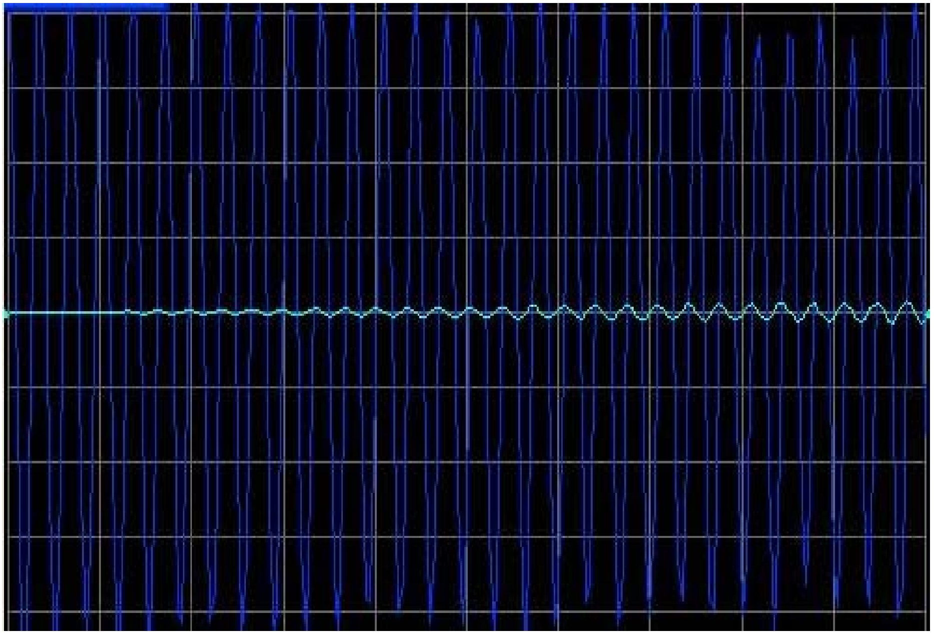
```
-0.50073051,-0.014679158
-0.21593559,0.52542627
0.2474933,0.47675157
0.51955992,0.0081026601
0.22609197,-0.50631678
-0.58351457,-0.46973184
-0.51758796,0.24241361
-0.068486258,0.47544035
0.26157933,0.10930297
0.078121327,-0.25116768
-0.16174096,-0.081873089
0.032612171,-0.014848446
0.0172925,-0.22964528
END
```

trace.sim.txt - Notepad

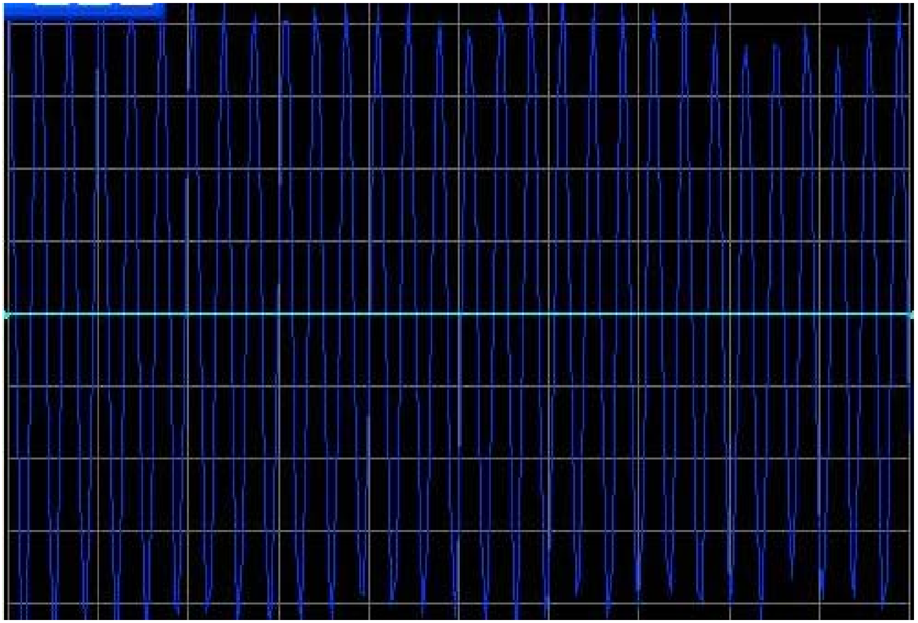
File Edit Format View Help

```
BEGIN
1.3085769,-0.23345217
0.50325823,-0.81402045
-0.50998622,-0.62219727
-0.78982878,0.38733667
-0.025088435,1.1177561
0.87216932,0.64021963
0.7217322,-0.51608777
-0.28219843,-0.84778357
-1.0469918,0.024442753
-0.50850981,0.92023557
0.53549159,0.72177929
0.81221962,-0.294027
-0.025584033,-0.92443269
```

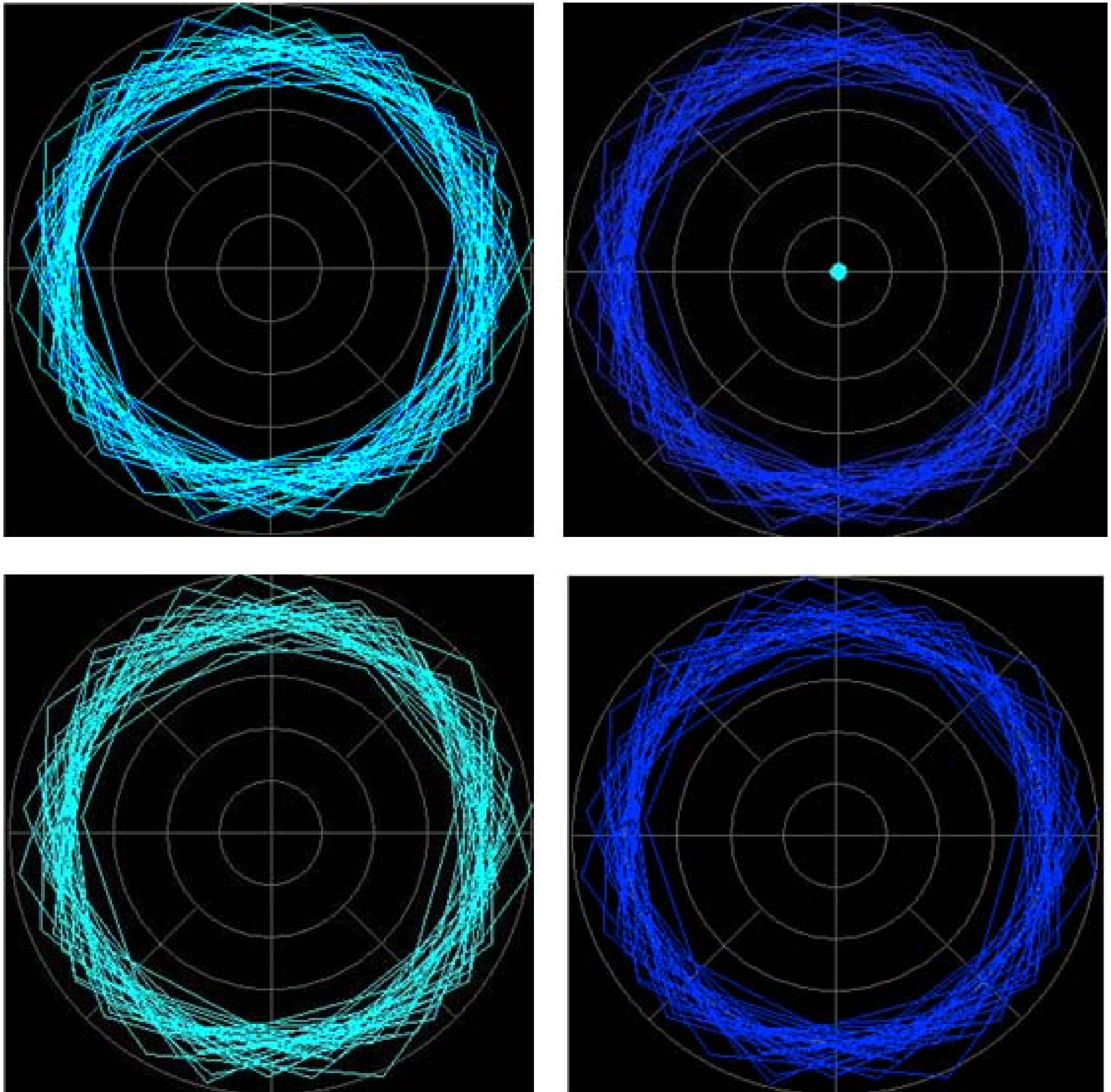
```
-0.50073051,-0.014679158
-0.21593559,0.52542627
0.2474933,0.47675157
0.51955992,0.0081026601
0.22609197,-0.50631678
-0.58351457,-0.46973184
-0.51758796,0.24241361
-0.068486258,0.47544035
0.26157933,0.10930297
0.078121327,-0.25116768
-0.16174096,-0.081873089
0.032612171,-0.014848446
0.0172925,-0.22964528
END
```



Below
1 physical DUT event and its playback …
original signal (dark blue),
difference signal (light blue)



Above
2 physical DUT events, moments apart in time …
original signal (dark blue),
difference signal (light blue)



top
2 physical events

left
signals overlapped

right
original + difference

bottom
1 event + playback

What was accomplished?

- ✓ specification
  - problems of error and cost management
  - solution: a software system
  - virtual source & processing of Sample streams
- ✓ development
  - generalized design template
  - implementation-dependent features
  - C++ class package with COM interface
- ✓ verification
  - unit testing of package internals
  - integration with working code branch
  - use-case behavioral testing

Test & measurement trends

- 21st century challenges - interdependent
  - climate change
  - energy production & conservation
  - economic realignment
  - health & environment
- science & technology - convergent
  - engineering
  - physical & life
  - information
- software as organizer - emergent
  - embedded computing
  - digital signal processing (DSP)
  - object-oriented systems

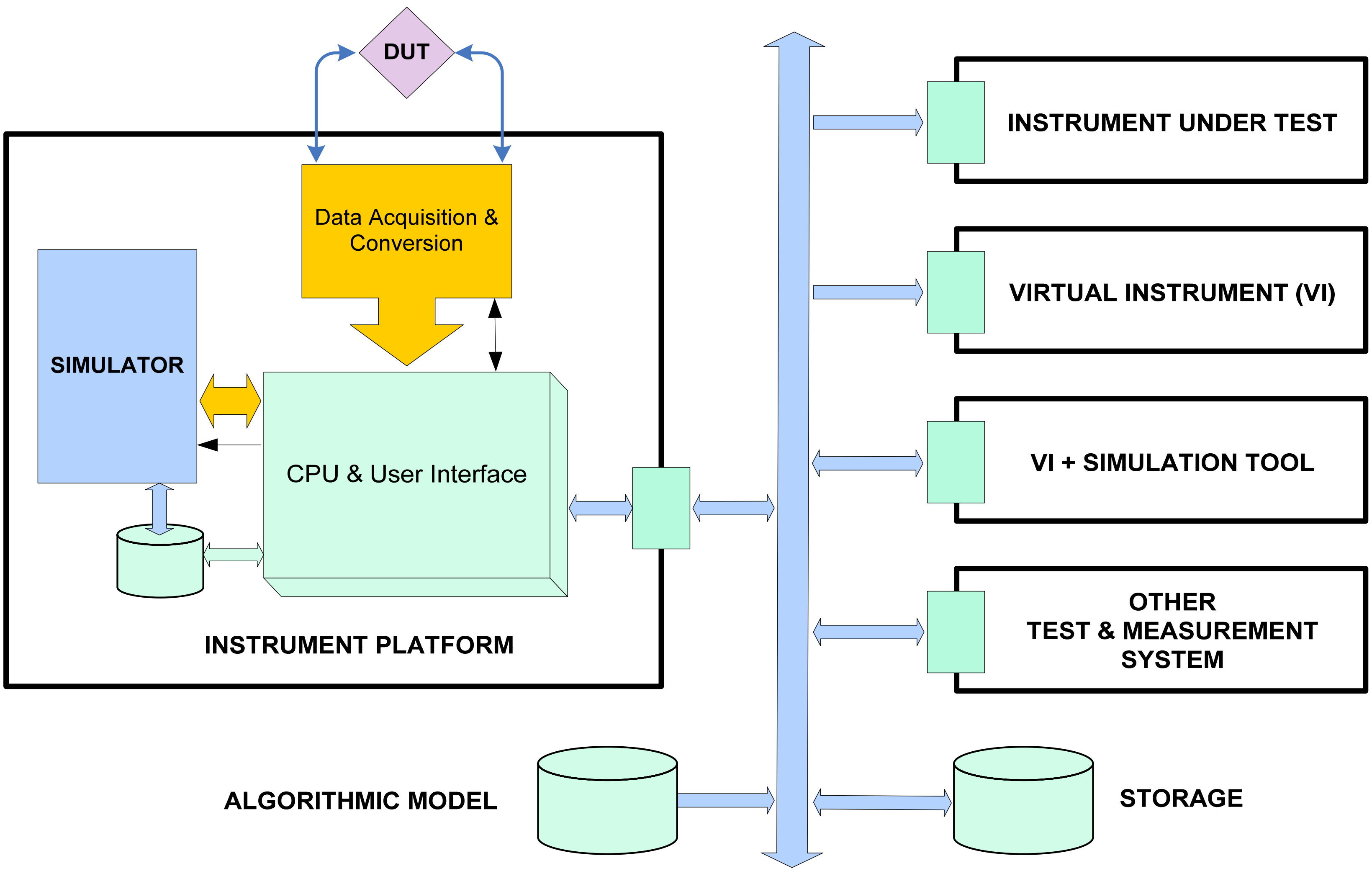
Role of the Record/Playback method

- leverage
  - exactly specified sequence of Samples
  - free of noise & drift between procedure runs
  - representation of DUT

- known applications
  - recorded Sweeps for testing software
  - verify any processor of Sample streams
  - include complex setups in test data
  - capture & reproduce rare/exceptional events

- potential applications
  - stimulus-response model of DUT
  - algorithmic model for testing or external use
  - automated production testing
  - external simulation platform

hypothetical automated test system employing DUT simulation



Record/Playback file compression

- must be **lossless** (requirement to play-back exact sample sequence)
- some open-source packages with recognized performance, reliability:
  - **LZMA**: hybrid (dictionary + statistical) encoder
  - **bzip2**: Burrows-Wheeler pre-compressor + statistical encoder

		compression factor	
file type	**size, bytes**	**lzma**	**bzip2**
ANSI text	59,703	3.469	3.703
Word file with images	668,160	1.120	1.096
Record/Playback	19,508	1.246	1.162
Record/Playback	97,008	1.257	1.198
Record/Playback	**1,250,130**	**1.253**	**1.215**

Comparing two popular open-source compressors

From Shannon information theory we have the quantity **entropy**, an index of the "surprise" information content of a data set.
Entropy H is a measure of the "absence of" redundancy R contained in the set:

$$H = H_{max} - R$$

where H_{max} is the maximum entropy attainable by the symbol set, and R (a positive number representing a negative amount of redundancy) equates to a positive quantity of entropy. For a data set comprised of n distinct symbols with probability mass function p, H_{max} is the entropy of a uniformly-distributed set of the same n symbols:

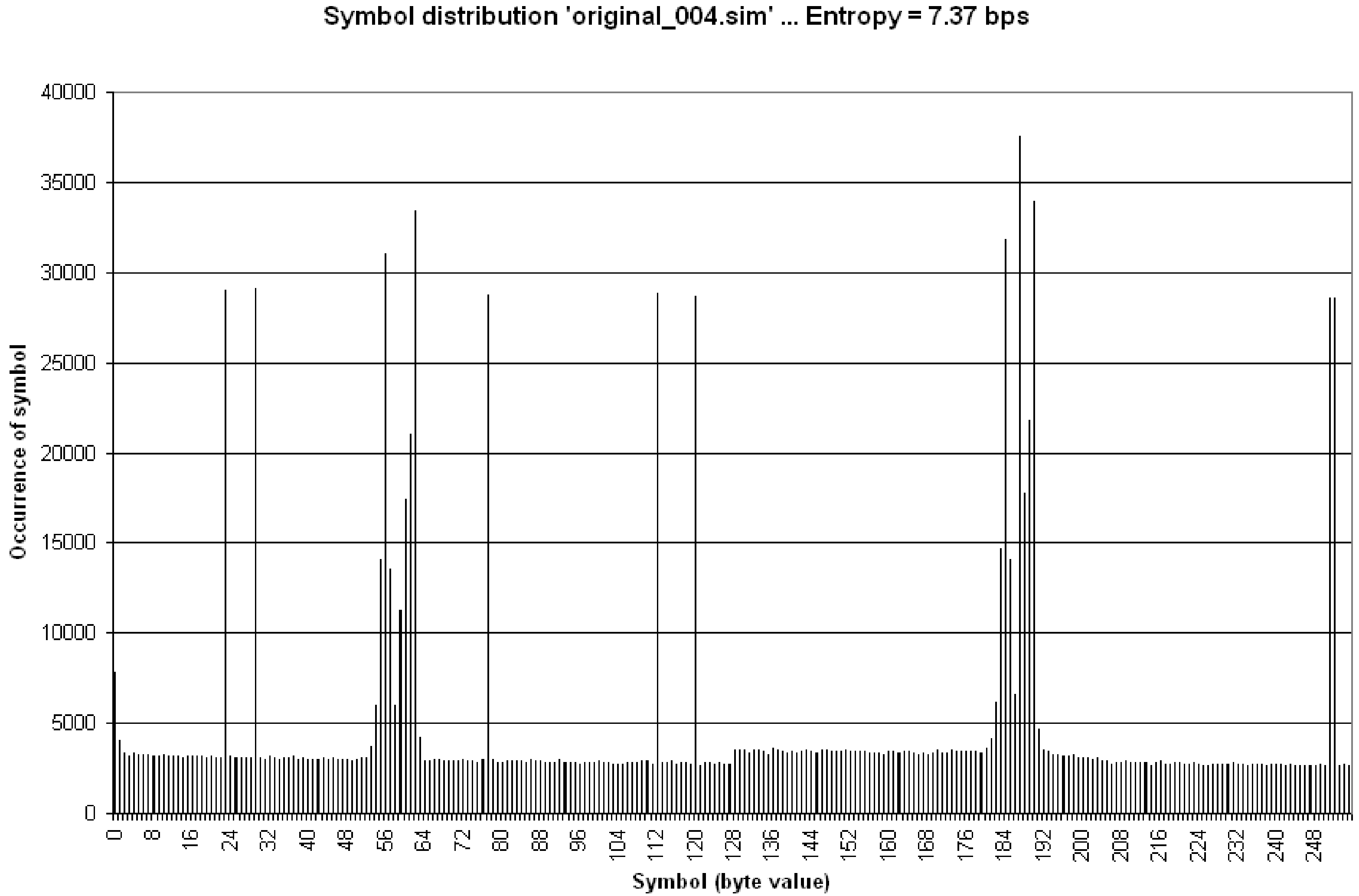
$$H = \sum_{k=1 \rightarrow n} p_k \log(1 / p_k) = n\, p \log(1 / p) - R$$
$$H = n\,(1/n) \log n - R$$
$$H = \log n - R$$

Using base-2 logarithms, H is the expected value of the number of bits required to encode a single symbol. With a uniform distribution, every symbol would require the same number of bits to encode – that's maximum entropy: a condition we recognize intuitively as perfect "randomness" or "disorder". The greater R, the more variation exists in the number of bits needed to encode various symbols - and the more compressible is the data set, in principle.

Given the file's symbol distribution …

Estimate the best compression factor we could achieve by using statistical encoding.

Hmax = 8 bits (one byte) per symbol
Hmax / H = 1.085



Algorithmic entropy

Length of the shortest <u>recipe</u> that will reversibly encode a sequence of symbols

Consider the following three sequences of integers:

(a) 3 2 5 1 2 5 4 3 4 5 2 1 3 4 1
(b) 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
(c) 1 1 1 4 4 4 5 5 5 2 2 2 3 3 3

All three sequences contain the same **Shannon entropy**, but have different kinds of **local redundancy**.

Sequence (b) might be encoded as follows:

1 2 3 4 5 # 3

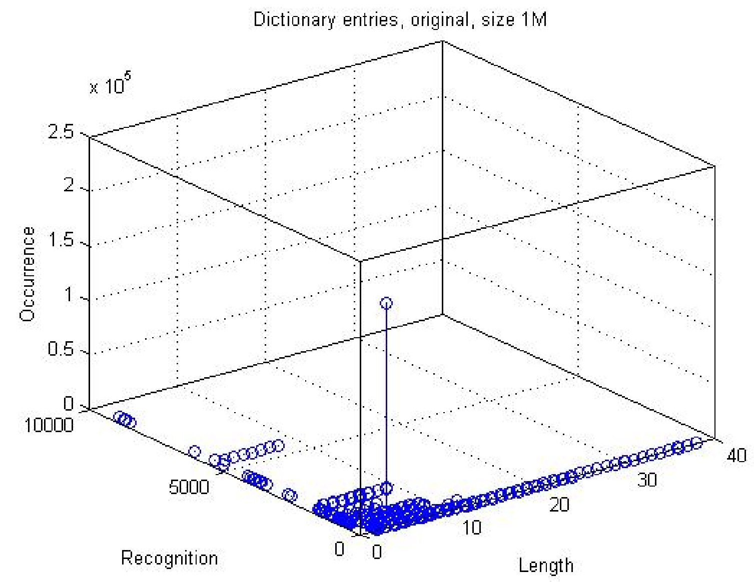
Burrows-Wheeler Transform

Permute blocks of symbols in order to maximize local redundancy.

Result is processed to remove the enhanced redundancy → more efficient removal of R

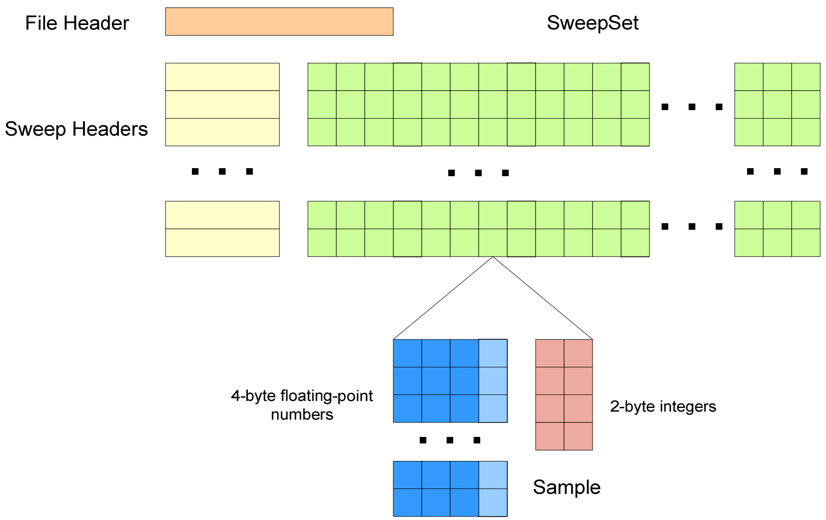
Dictionary-based encoding

Input stream, output stream, dictionary. Monitor the input. When a new unique sequence is found, store it in the dictionary, and output it. When a matching sequence is found, output only a short key to its dictionary entry.

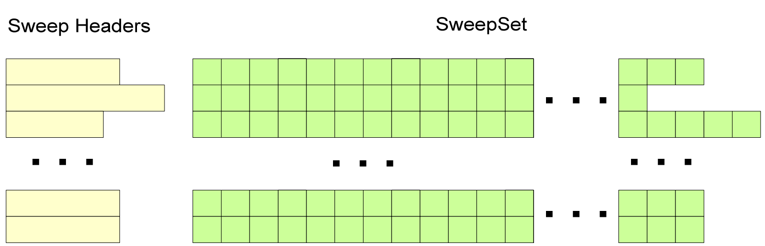


Record/Playback file is not very “dictionary-friendly”.

SimZip: custom pre-compression + LZMA



Data components of Record/Playback file



SimZip handles jagged data components crrectly.

- Built **data tools** to analyze Record/Playback file.
- Identified 5 data components with characteristic local redundancy patterns
- Built **transform** functions to enhance local redundancy of each component.
- *Exception (dark blue) – no redundancy, LZMA gave negative compression.*
- Also built inverse transform functions for decoding.
- Lots of row & column rearrangements, matrix transposition.
- Transform methods are implemented as **C# delegate** instances.

SimZip encoder:
- Extract each data component from the source file.
- Apply a corresponding transformation to each component.
- Compress each (transformed) component individually, using LZMA.
- Prepend a byte count to each compressed segment.
- Concatenate the segments, and prepend a non-compressed file header.

SimZip decoder:
- Read the header, note segment information.
- Decatenate the segments (use byte count), decompress using LZMA.
- Apply each corresponding inverse transformation..
- Load the destination file with the recovered data components.

SimZip compression performance

Amdahl's Law bounds the performance improvement to a system with a non-improving component:

$$s \leq 1 / f$$

where s is the overall performance improvement, and f is the proportion of the system's performance determined by the non-improving component.

Here, f is the *dark blue* component (previous figure) = 0.622 of file's byte count.
So theoretical max compression is 1.61 … LZMA alone yielded 1.25 …

Estimate SimZip compression factor as **1.3 to 1.5**

On the 1.25 MB Record/Playback test file, the SimZip utility improved the compression factor by about **14%** relative to that of the basic LZMA utility.

Size, bytes	File name	Compress factor	Comment
871,919	original.cim	1.43	SimZip (pre-compression + LZMA)
997,397	original.lzma	1.25	standard LZMA compression
1,250,130	original.sim	1.00	uncompressed file