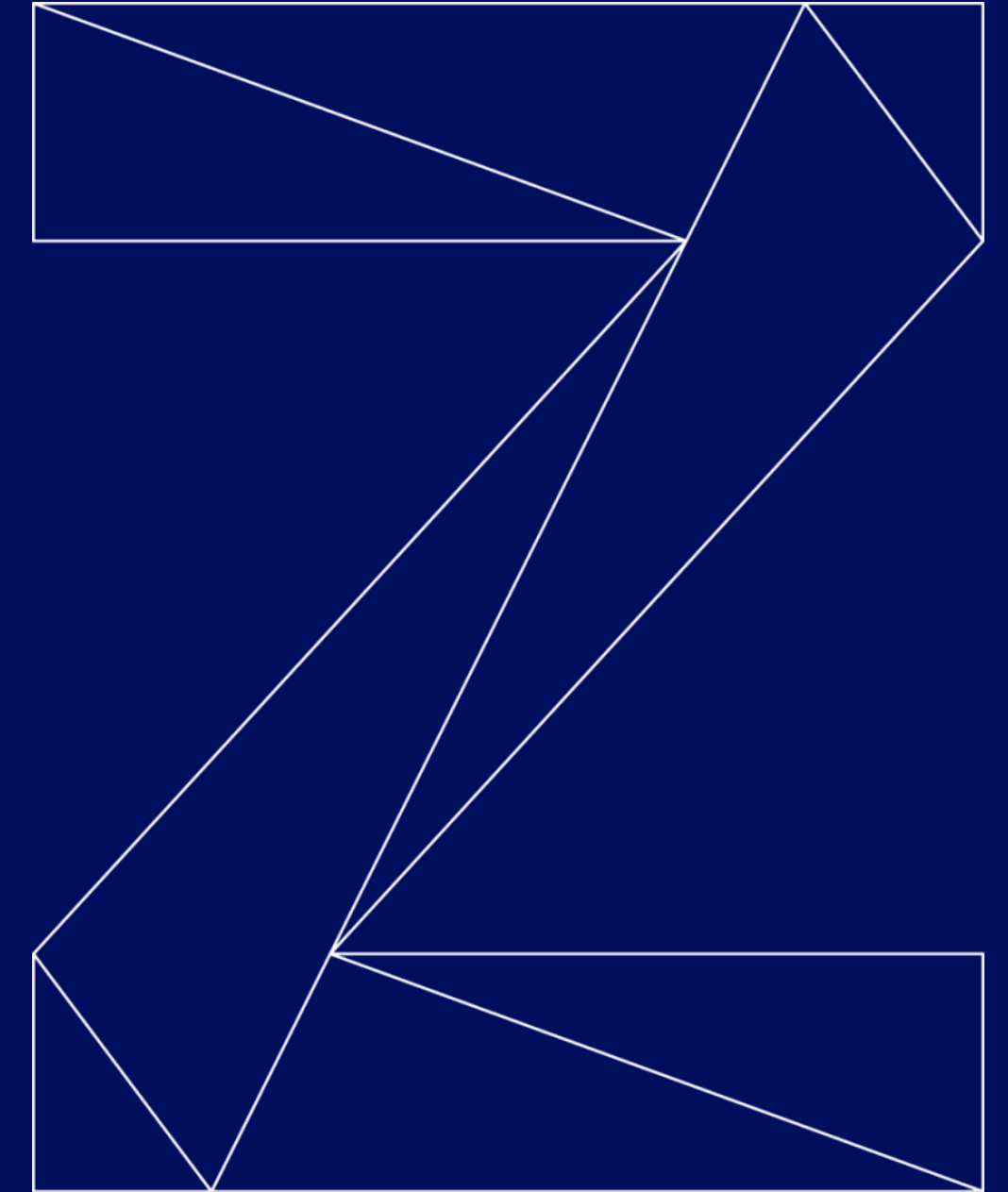


Managing Matryoshkas: Testing Nested Guests

—
Marc Hartmayer
<mhartmay@de.ibm.com>



Trademarks & Disclaimer

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries. For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

IBM, the IBM logo, IBM Z, IBM z Systems, IBM z15, IBM z14, WebSphere, DB2 and Tivoli are trademarks of IBM Corporation in the United States and/or other countries. For a list of additional IBM trademarks, please see <https://ibm.com/legal/copytrade.shtml>.

The following are trademarks or registered trademarks of other companies: Java and all Java based trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries or both Microsoft, Windows, Windows NT and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both. Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries or both. Linux is a trademark of Linus Torvalds in the United States, other countries, or both. Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc. InfiniBand is a trademark of the InfiniBand Trade Association.

Other company, product, or service names may be trademarks or service marks of others.

NOTES: Linux penguin image courtesy of Larry Ewing (lewing@isc.tamu.edu) and The GIMP

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Users of this document should verify the applicable data for their specific environment. IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Information is provided “AS IS” without warranty of any kind. All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

Trademarks & Disclaimer #2

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area. All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices are suggested US list prices and are subject to change without notice. Starting price may not include a hard drive, operating system or other features. Contact your IBM representative or Business Partner for the most current pricing in your geography. Any proposed use of claims in this presentation outside of the United States must be reviewed by local IBM country counsel prior to such use. The information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any

Notice Regarding Specialty Engines

Any information contained in this document regarding Specialty Engines (“SEs”) and SE eligible workloads provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g., zIIPs, zAAPs, and IFLs). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at www.ibm.com/systems/support/machine_warranties/machine_code/aut.html (“AUT”).

No other workload processing is authorized for execution on an SE.

IBM offers SEs at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.

Agenda

What is Nested Virtualization?

Testing Nested Virtualization

Demo

New Approach

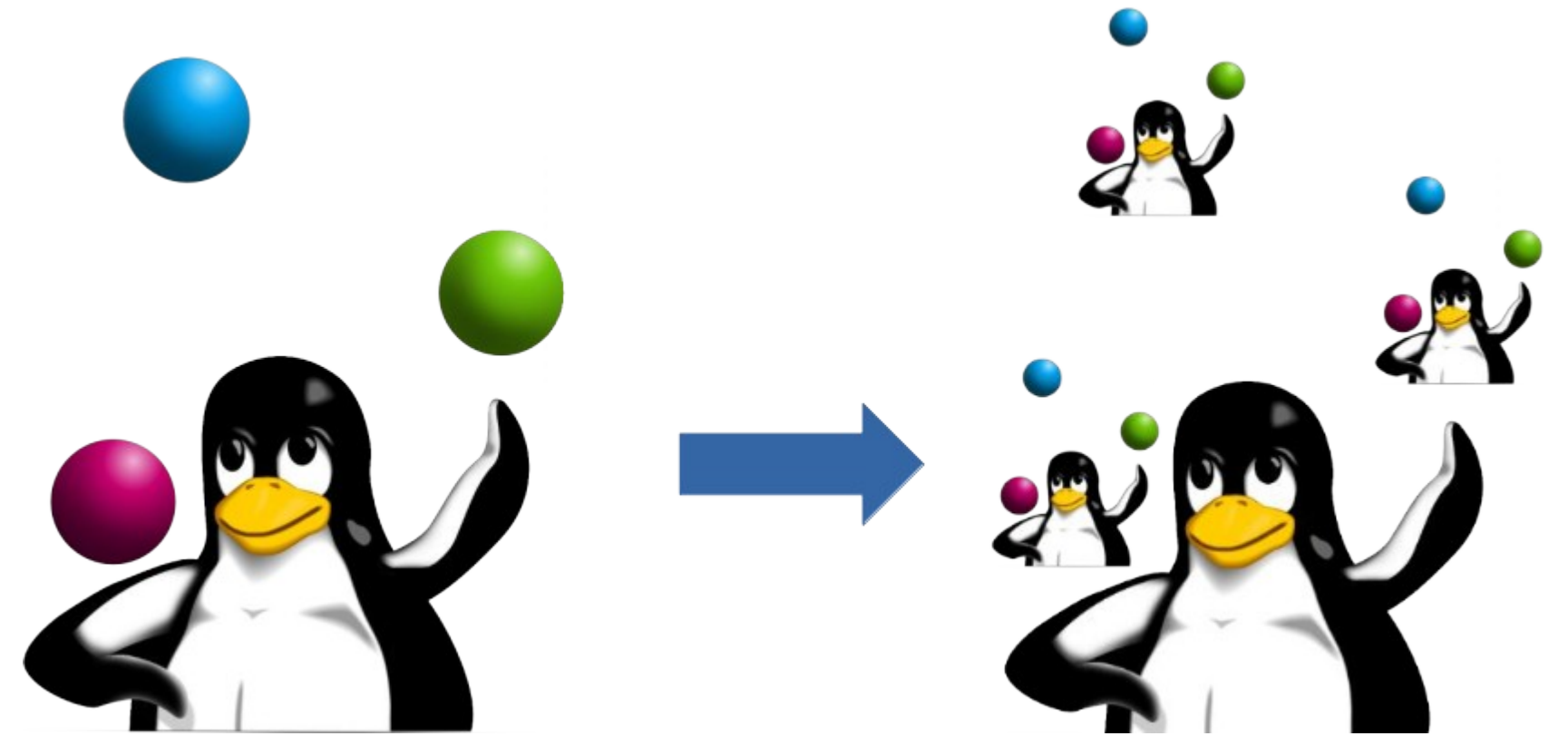
Design Goals, Ideas, and Details

What's next?



Nested Virtualization

- Turn guest into host
- Use cases:
 - Development/Testing
 - Production
 - Training
- Terminology:
 - “L0” - bare metal host, running KVM
 - “L1” - VM running on “L0”, acting as hypervisor
 - “L2” - VM running on “L1” - called nested guest
 - And so on...

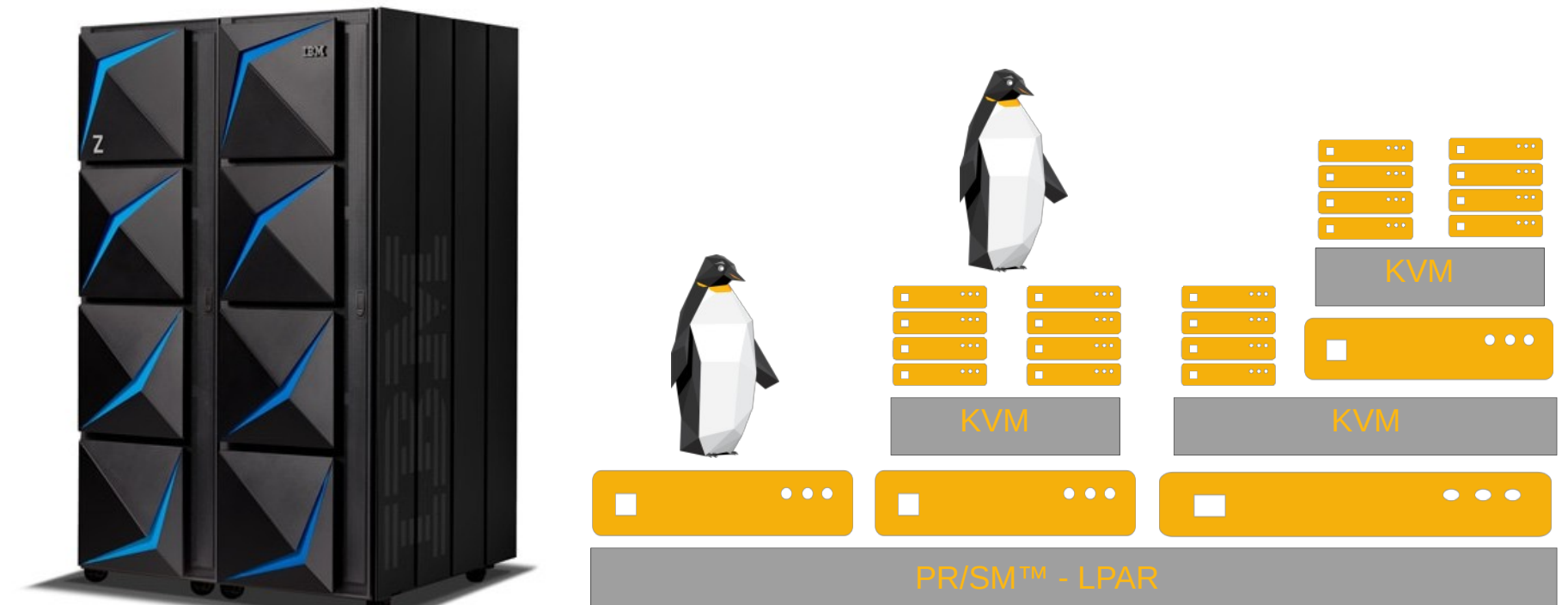


Source:
http://events17.linuxfoundation.org/sites/events/files/slides/Nesting%20KVM%20on%20s390x%20-%20David%20Hildenbrand_0.pdf

Nested Virtualization on s390x

- Requirements:
 - Kernel ≥ 4.8
 - `kvm.nested=1`
 - QEMU ≥ 2.9
 - CPU host model
- Multi-level nesting support
- Hardware assisted via SIE (*Start Interpretive Execution*)
- Migration works between different levels
- Supports migration of L1 guest with L2 guest running¹

[1] See https://www.linux-kvm.org/page/Nested_Guests



Source:

https://mp.s81c.com/pwb-production/4a422bbd1af7c77c051f5edcfc9adc9f/additionalOfferingImg__1_5ba956f1-33db-4219-9496-3f4347e4ed27_859e77ce-b463-4f06-a27a-5a5bc70e23b8.jpg

Source:

http://events17.linuxfoundation.org/sites/events/files/slides/Nesting%20KVM%20on%20s390x%20-%20David%20Hildenbrand_0.pdf

Testing nested virtualization

Available Test Suites/Frameworks for QEMU/KVM (and libvirt)

- Avocado-VT: tp-libvirt, tp-qemu
 - Avocado_qemu
 - kvm-unit-tests
 - Libvirt TCK
 - Linux Virtualization Tests (virt-test) (legacy only)
 - Supernested
- Avocado-VT seems to be the most evolved framework

Shortcomings for nested virtualization tests

Shortcomings of Avocado-VT for nested virtualization testing:

- Interaction with host using Python
- Interaction with the guests is done via SSH using bash

Why?

Wouldn't it be great if we can simply reuse our host code in the nested guest, the new "host"?

Guest interaction (aka linux_hw_check.py)

```
priv_key = os.path.join(self.vm_hw['key_path'], 'id_rsa')
with ssh.Session(('127.0.0.1', ssh_port),
                 ('root', priv_key)) as session:

    # cpu
    proc_count_cmd = 'egrep -c "^processor\s\:" /proc/cpuinfo'
    self.assertEqual(int(self.vm_hw['smp']),
                     int(session.cmd(proc_count_cmd).stdout_text.strip()))

    # memory
    match = re.match(r"^MemTotal:\s+(\d+)\s kB",
                    session.cmd('cat /proc/meminfo').stdout_text.strip())
    self.assertIsNotNone(match)
    exact_mem_kb = int(self.vm_hw['memory']) * 1024
    guest_mem_kb = int(match.group(1))
    self.assertGreaterEqual(guest_mem_kb, exact_mem_kb * 0.9)
    self.assertLessEqual(guest_mem_kb, exact_mem_kb)
```

Source: Presentation "The functional test beast: tame it, bring it home and make it your pet" (<https://events19.linuxfoundation.org/wp-content/uploads/2017/12/The-Functional-Test-Beast-Tame-it-Bring-it-Home-and-Make-it-your-Pet-Cleber-Rosa-Red-Hat-Inc..pdf>)

Shortcomings for nested virtualization tests

- Debugging of nested guest code is hard
- Common tasks, e.g. hot (un)plugging a device

- Workarounds are often used:

- Sleeps
- Busy loops for polling
- Coarse-grained “udevadm settle”

Wouldn't it be great to use pyudev¹ everywhere?

- Different semantics

- e.g. `Popen(...)` vs. `session.cmd()`

Wouldn't it be great to have the exact same semantic in the guest as in the host?

- Error handling? Stack traces?

[1] See <https://pyudev.readthedocs.io>

Use your host
code in the
guest

Example Test Case

```
class ExampleTestCase(SshTestCase):
```

```
    DOMAIN_NAMES = ['test1']
```

```
    def runTest(self):
```

```
        guest = self.guests['test1']
```

```
        stdout = guest.call(subprocess.check_output, ['hostname']).decode()
```

```
        self.assertEqual(stdout, 'qemus390x\n')
```

```
        self.assertEqual(guest.call(socket.gethostname), stdout.strip())
```

DEMO TIME

Current state...
Prototype only.

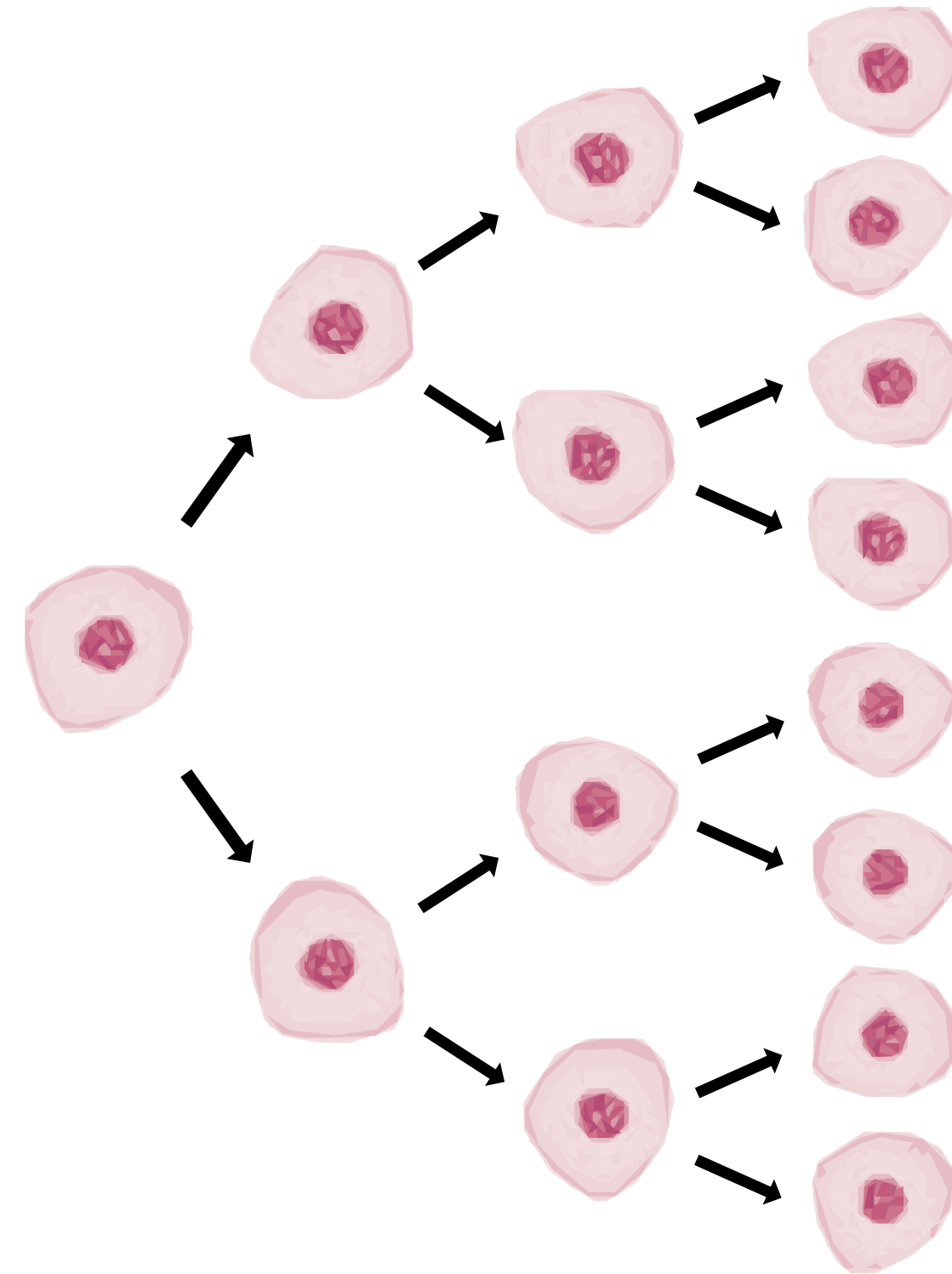
Usage of Mitogen¹

“[...] make it childsplay to run Python code on remote machines [...]” - David Wilson²

- Python library for writing distributed self-replicating programs
- Support for Python2 \geq 2.4 and Python3.x
- Python interpreter and SSH client must be installed
 - Zero Python dependencies
(uses only Python standard library)

[1] <https://github.com/dw/mitogen> created by David Wilson

[2] <https://sweetness.hmmz.org/page/7/> [visited 23.10.2019]



Source: <https://sweetness.hmmz.org/images/mito1/mitogen.svg>

Mitogen Overview

- Bootstrap: Spins up and hooks up a "remote" Python, e.g. via SSH → This forms a **new context**
 - Function calls in this context:
 - Uses `pickle`¹ for marshalling
- Resolves Python dependencies in the remote context transparently²
- Has concept of services³ with a state
 - User-supplied class with explicitly exposed methods, which can be called by other contexts
- Forwards Stdio and logs (`logging` package)
- Supports asynchronous calls

[1] See <https://docs.python.org/3/library/pickle.html> for details

[2] See <https://mitogen.networkgenomics.com/#module-forwarder> for details.

[3] See <https://mitogen.networkgenomics.com/services.html> for details.

```
"""Get number of logical CPUs on host_2 using host_1 as an SSH hop.
```

```
Usage: $ python3 count.py host_1 host_2"""
```

```
import sys
```

```
import mitogen
```

```
import psutil
```

```
@mitogen.main()
```

```
def main(router):
```

```
    host_1_ctxt = router.ssh(hostname=sys.argv[1])
```

```
    host_2_ctxt = router.ssh(via=host_1_ctxt,  
hostname=sys.argv[2])
```

```
    print(host_2_ctxt.call(psutil.cpu_count, logical=True))
```


Mitogen is not enough...

- 1) Handle (un)pickling
- 2) Mutable state in remote context(s),
e.g. even placed in a native library (`libvirt-python`)
- 3) Lifetime of the objects in the remote context(s)

```
def lookup(name):  
    import libvirt  
    conn = libvirt.open()  
    dom = conn.lookupByName(name)  
    return dom
```

```
dom = guest_1.call(lookup, "demo")  
dom.create()
```

Remote invocation via proxy objects

Solution for pickling and state problem

- Contexts return proxy objects and methods for everything
 - Can be used as argument in function calls
- Register trusted classes to the (un)pickler
- Special “dunder method” `__value__` for receiving the actual object
 - Enforce the unpickling of non-registered classes by using `trust` keyword

```
def lookup(name):  
    global dom  
    import libvirt  
    conn = libvirt.open()  
    dom = conn.lookupByName(name)  
    return dom
```

```
dom_proxy = guest_1.call(lookup, "demo")  
assert isinstance(dom_proxy, ProxyObj)  
dom_proxy.create()  
# will raise an "mitogen.core.CallError:  
# builtins.TypeError: can't pickle PyCapsule objects"  
# as a libvirt.virDomain object is not pickleable  
dom = dom_proxy.__value__(trust=True)
```

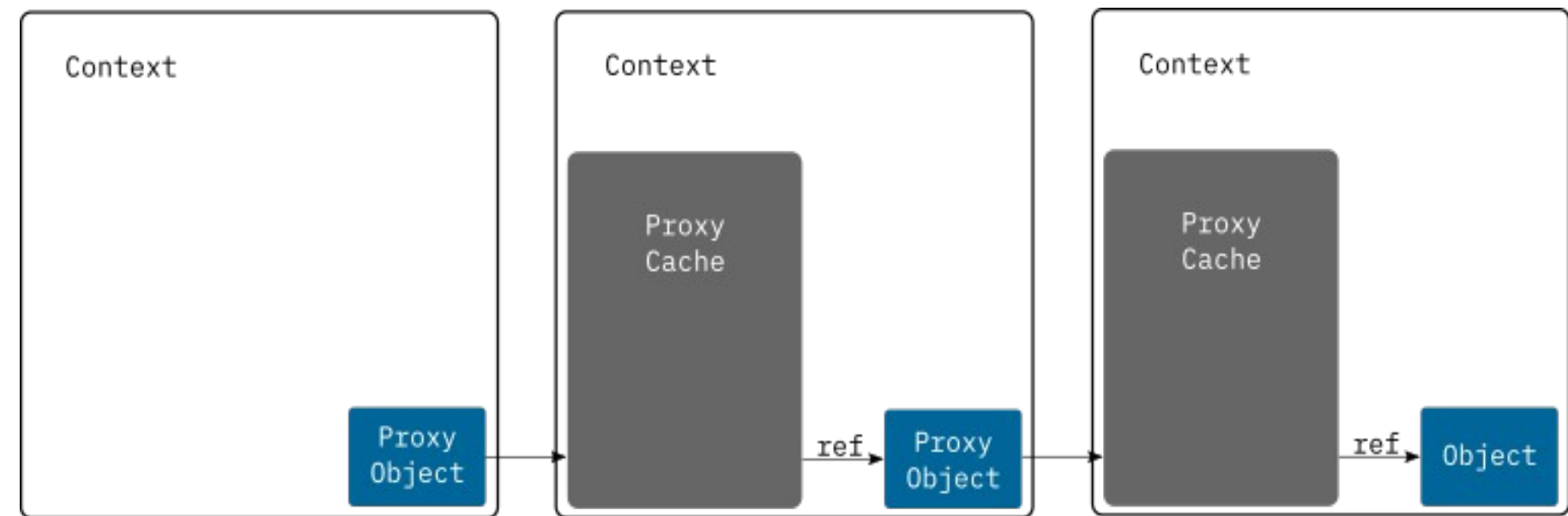
[1] See <https://docs.python.org/3/library/stdtypes.html>

[2] See <https://docs.python.org/3/reference/datamodel.html>

Connect Python Object life cycles

Solution for lifetime problem

- Connect life cycle of proxy objects and the actual objects.
 - As long as a proxy object is alive the referred object must not be garbage collected
 - “Transitive proxy object chains”



Conclusions

Advantages

- Guests can run Python host code transparently
 - Less duplicated code
- Usage of Python packages like pyudev in the guests
- Redirection of Stdio/logging in the guests
- Usage of shell commands can be minimized to a minimum

Limitations

- Native dependencies are not copied automatically, e.g. `libvirt.so`
- Python interpreter must be available in the guest and there is an overhead caused by the Python interpreter start
- Pickling limitations
 - use other pickle module, e.g. `dill`¹

[1] See <https://docs.python.org/3/library/pickle.html> for details.

Summary

- There is still much to do :/
- Test approaches for nested virtualization already exists, but...
- This new approach:
 - Allows the interaction with the host and guests using Python
 - Recursive reuse of host code in guests
 - Easy management of (nested) guests

What's next?

- Finish our implementation
- Upstreaming changes to Mitogen
- Make it easier to debug:
 - Implement remote Python Debugger (“remote PDB”)?
- Integrate our framework/tests into Avocado-VT?

The screenshot shows a web browser displaying the Avocado framework documentation. The browser's address bar shows the URL: `https://avocado-framework.readthedocs.io/en/72.0/api/utis/avocado.utis.html#`. The page title is "Utilities APIs". The left sidebar contains a navigation menu with "Utilities APIs" expanded, listing various subpackages and submodules. The main content area has a heading "Utilities APIs" and a sub-heading "Subpackages". A "Note" box contains a warning about hidden knowledge of avocado logging streams. Below that, there are sections for "Subpackages" and "Submodules". The "Submodules" section highlights the "avocado.utis.archive module".

Utilities APIs

Subpackages

- avocado.utis.archive module
- avocado.utis.asset module
- avocado.utis.astring module
- avocado.utis.aurl module
- avocado.utis.build module
- avocado.utis.cloudinit module
- avocado.utis.configure_network module
- avocado.utis.cpu module
- avocado.utis.crypto module
- avocado.utis.data_factory module
- avocado.utis.data_structures module
- avocado.utis.datadrainer module
- avocado.utis.debug module
- avocado.utis.diff_validator module
- avocado.utis.disk module
- avocado.utis.distro module
- avocado.utis.download module
- avocado.utis.file_utils module
- avocado.utis.filelock module
- avocado.utis.gdb module
- avocado.utis.genio module
- avocado.utis.git module
- avocado.utis.iso9660 module
- avocado.utis.kernel module
- avocado.utis.linux module
- avocado.utis.linux_modules module
- avocado.utis.lv_utils module

Docs » Utilities APIs

Utilities APIs

This is a set of utility APIs that Avocado provides as added value to test writers. It's suppose to be generic, without any knowledge of Avocado and reusable in other projects.

Note

In the current version there is a hidden knowledge of avocado logging streams. This issue can be found here <https://trello.com/c/4QyUgWsW/720-get-rid-of-avocado-logging-streams-from-avocado-utis>

Subpackages

- avocado.utis.external package
 - Submodules
 - avocado.utis.external.gdbmi_parser module
 - avocado.utis.external.spark module
 - Module contents

Submodules

avocado.utis.archive module

Module to help extract and create compressed archives.

exception `avocado.utis.archive.ArchiveException`

Bases: `Exception`

Base exception for all archive errors.

class `avocado.utis.archive.ArchiveFile(filename, mode='r')`

Bases: `object`

Class that represents an Archive file.

Read the Docs v: 72.0

Thank you.