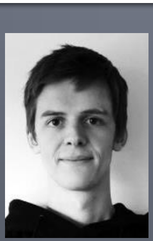
HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

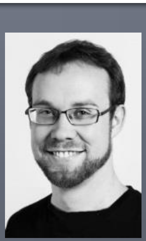
Microsoft Research

harmonium

Elastic Cloud Storage via File Motifs

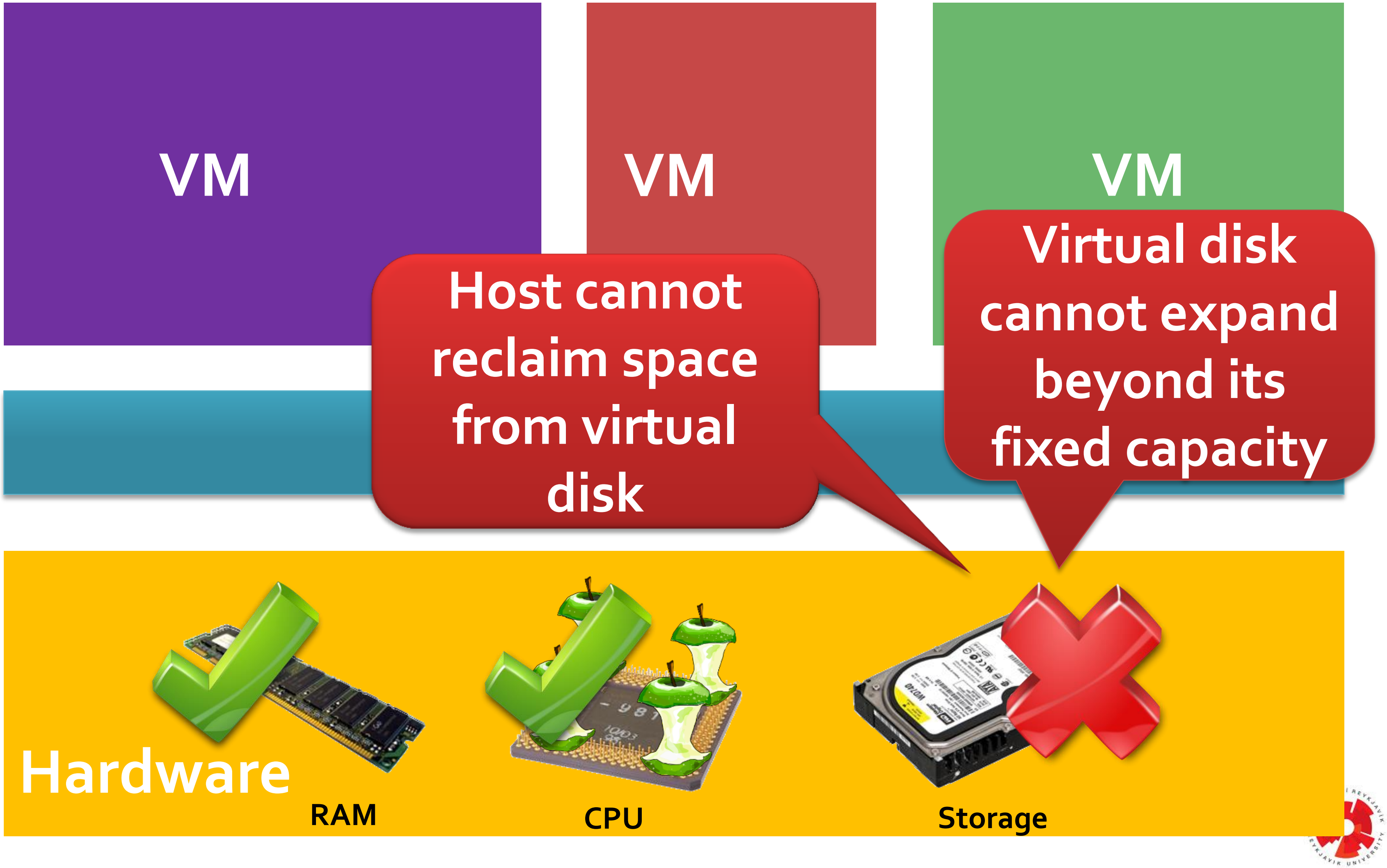
Helgi Kr. Sigurbjarnarson RU


Pétur O. Ragnarsson RU


Ymir Vigfusson RU

Mahesh Balakrishnan MSR

Elasticity



VM

VM

VM

Host cannot reclaim space from virtual disk

Virtual disk cannot expand beyond its fixed capacity

Hardware

RAM

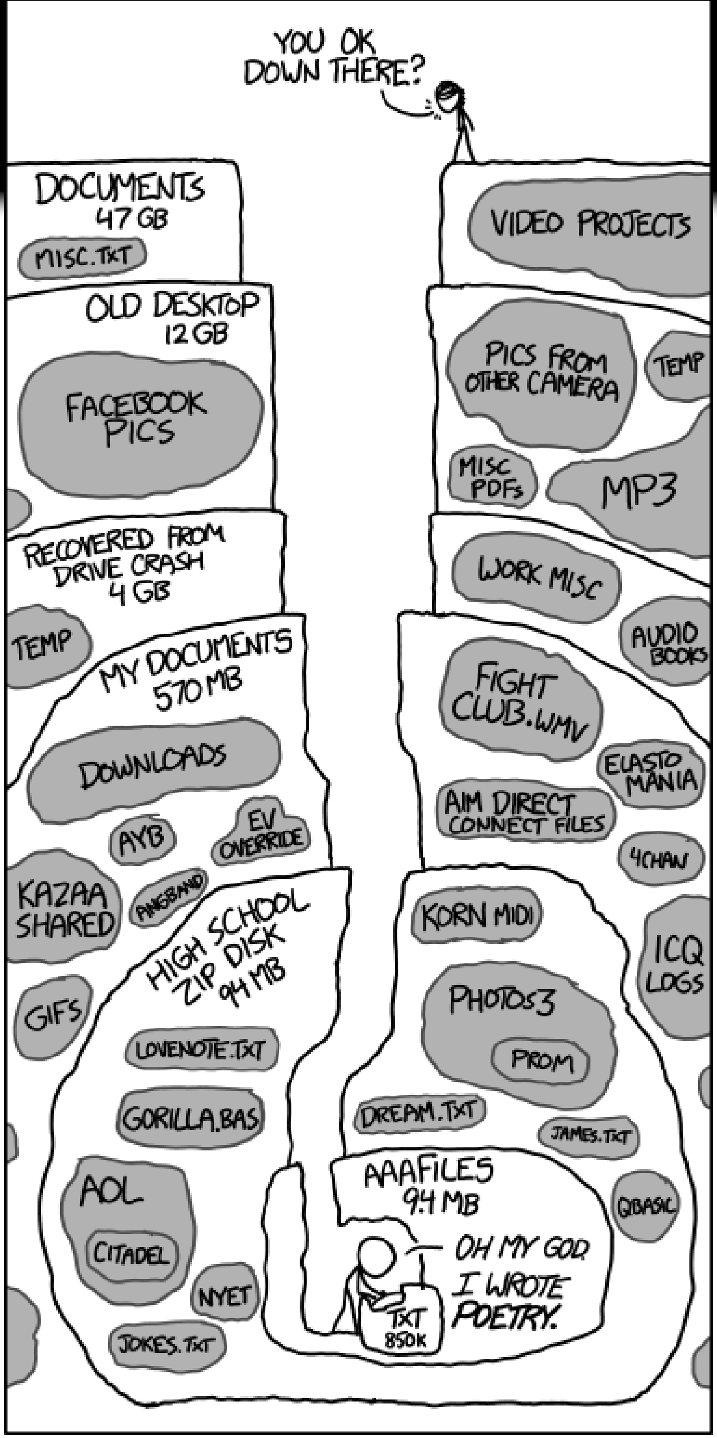
CPU

Storage

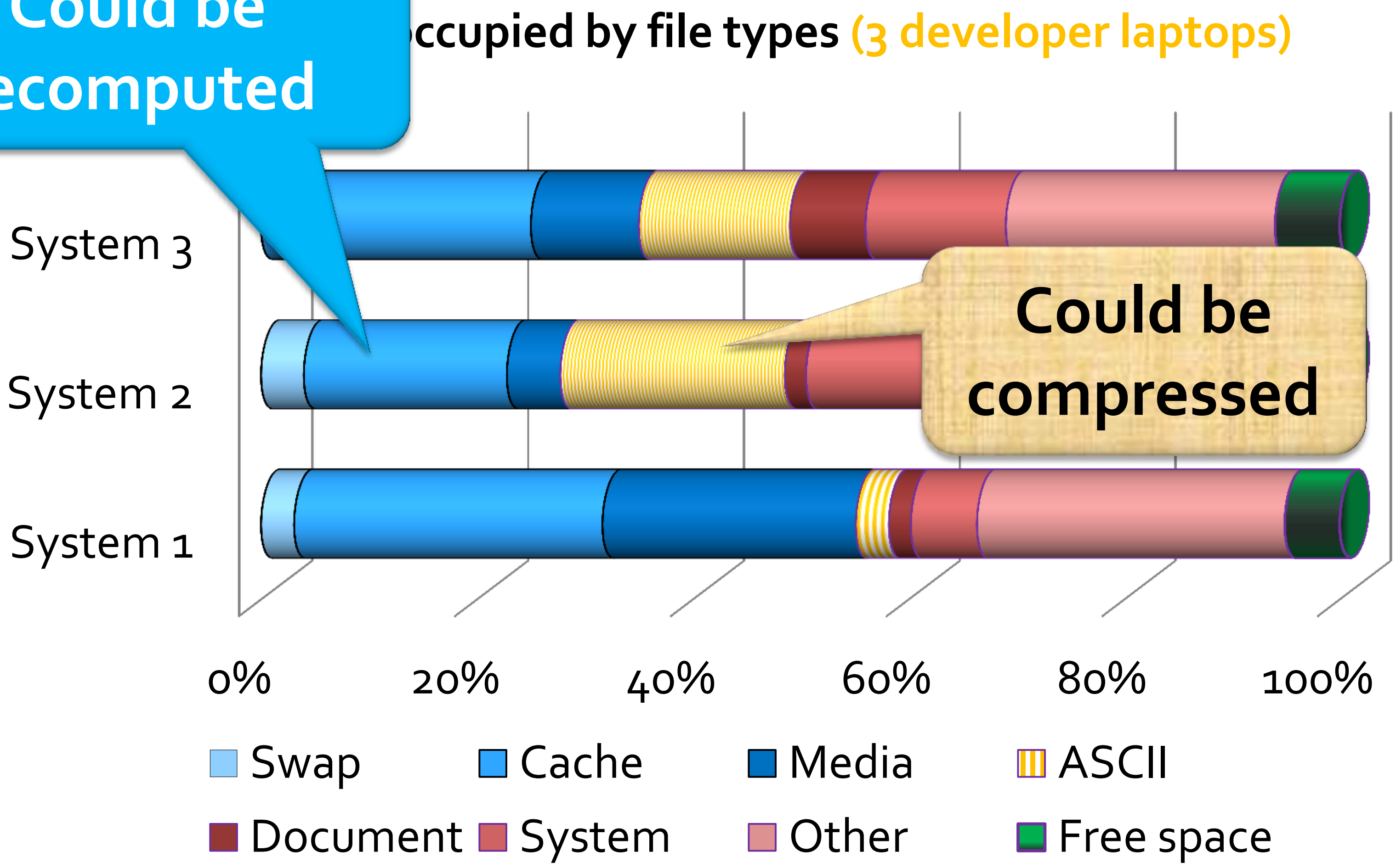
No storage elasticity

ALL
DATA
IS
EQUAL
NO
DATA
CAN
BE
LOST

http://xkcd.com/1360/



Not all data requires durability



Could be recomputed

Could be compressed

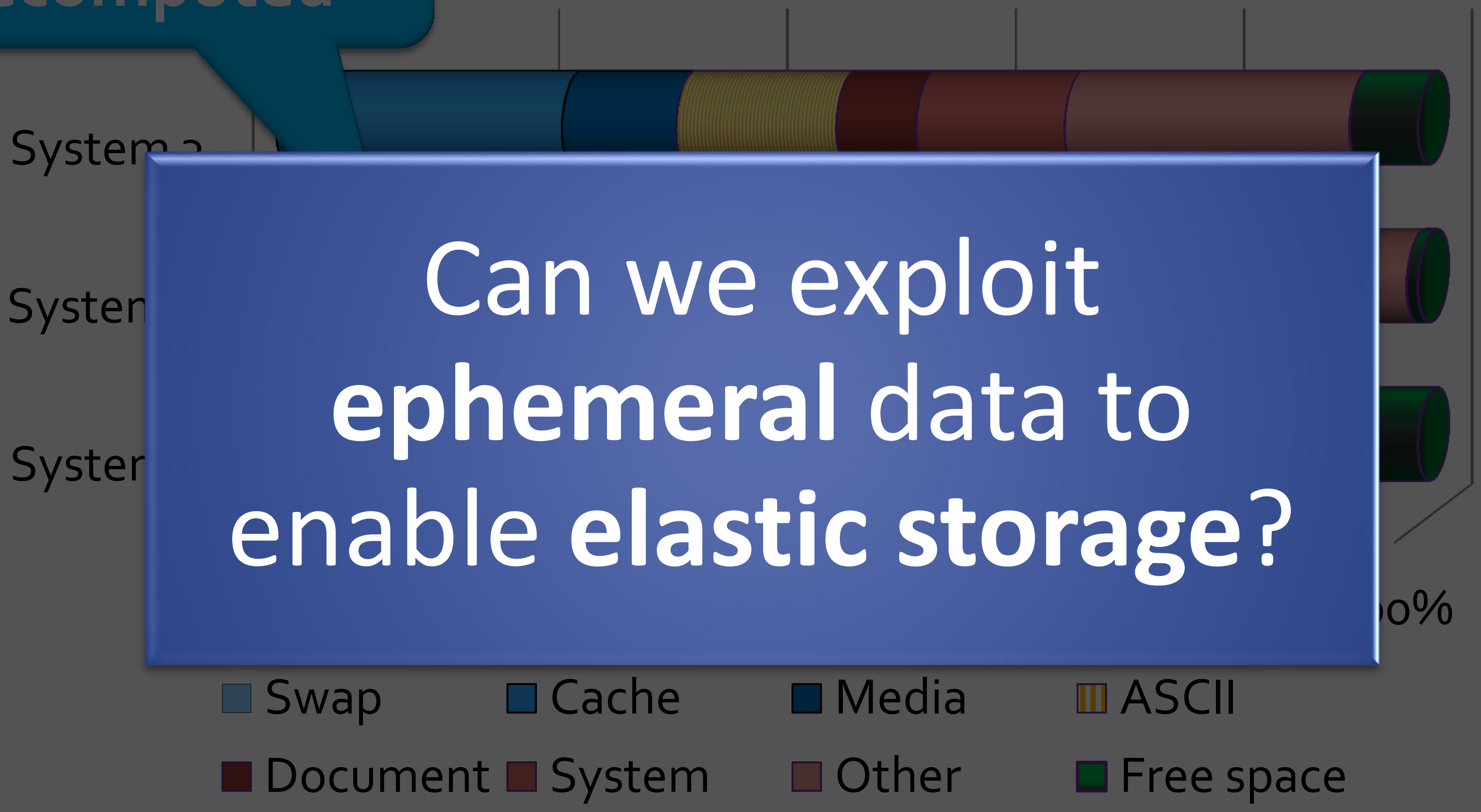
- **Significant ephemeral data**

Representative?

Not all data requires durability

Could be recomputed

ccupied by file types (3 developer laptops)



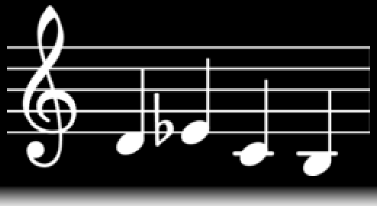
Can we exploit **ephemeral** data to enable **elastic storage**?

- **Significant ephemeral data**

Representative?

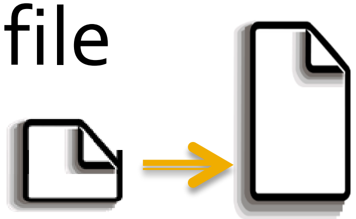
What if files could **contract** and **expand**?

The motif abstraction

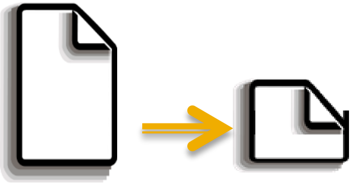


A **motif** is **code** to generate the **data** in a file

- **expand()** obtains the raw bytes of the file
  - Run **computations** (e.g. **compression**)
  - **Fetch data** across a network
  - Operate over **other files** on the FS



- **contract()** deletes raw bytes, retains motif code



Motifs can be recursive

Motifs can have circular dependencies

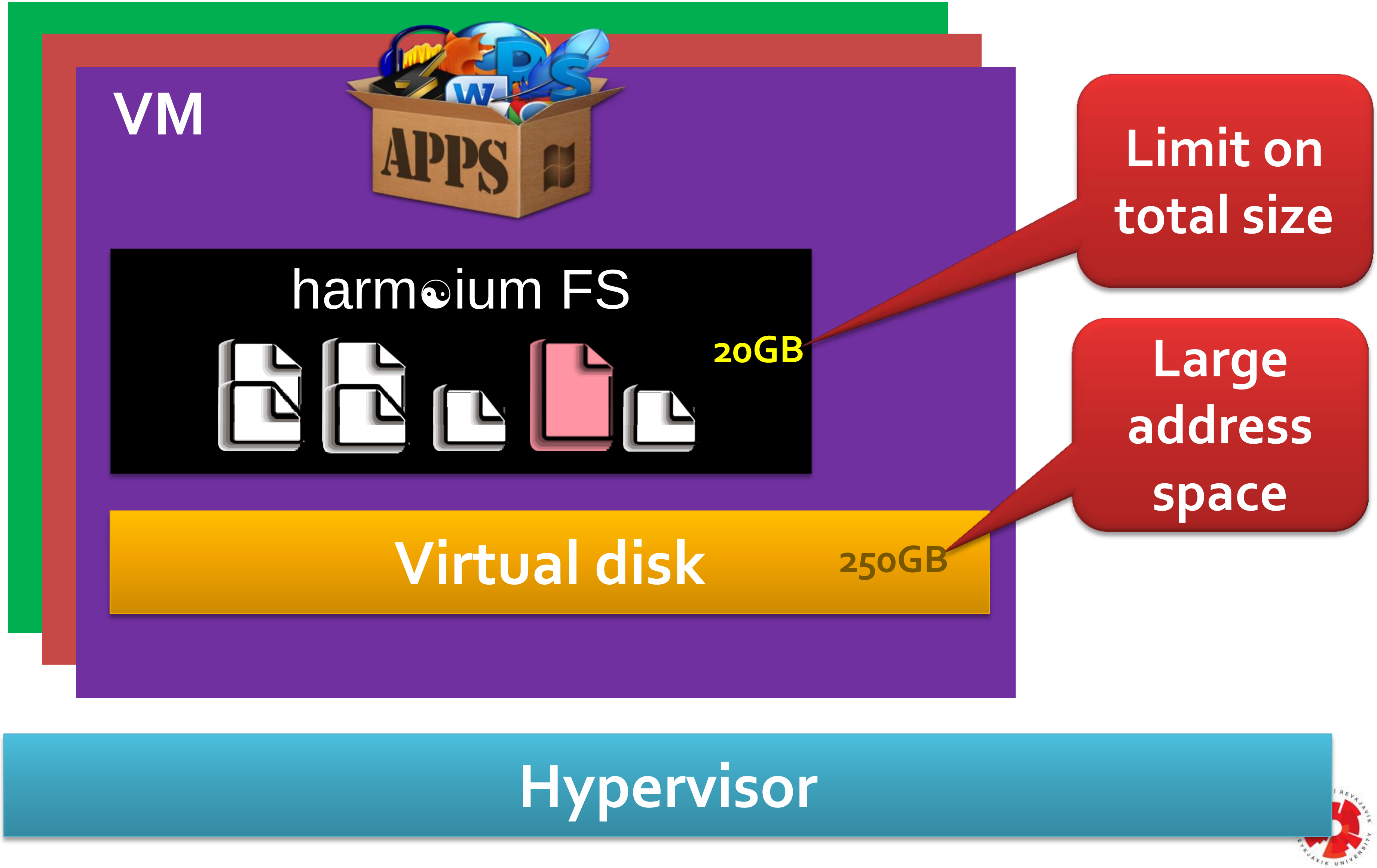
Files can have multiple motifs

Example *network storage* motif

```
class SCPMotif(object):
    def expand(self, fname, meta=None):
        p = fname.bypass()
        os.popen(’scp \
              fileserver1:storage%s "%s"’ % (p,p))

    def contract(self, fname, meta=None):
        p = fname.bypass()
        os.popen(’ssh fileserver1\
            "mkdir -p storage%s"’ %\
            os.path.dirname(p))
        if os.popen(’scp "%s"\
              fileserver1:storage%s’ % (p,p)) == 0:
            open(p, ’w’).close()
```

The harmonium file system



VM

harmonium FS

20GB

Virtual disk

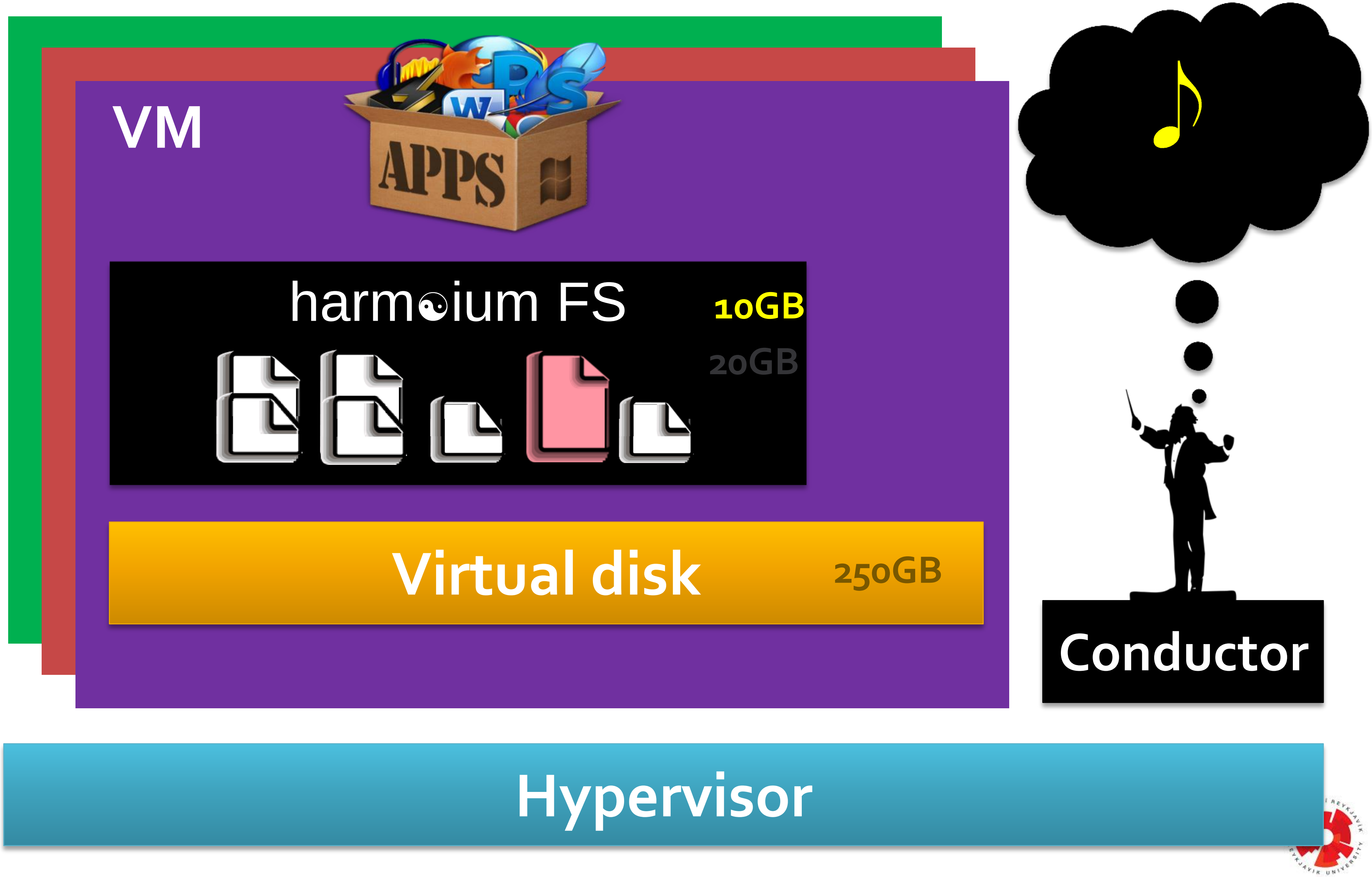
250GB

Hypervisor

Limit on total size

Large address space

The harmonium file system



VM

harmonium FS

10GB

20GB

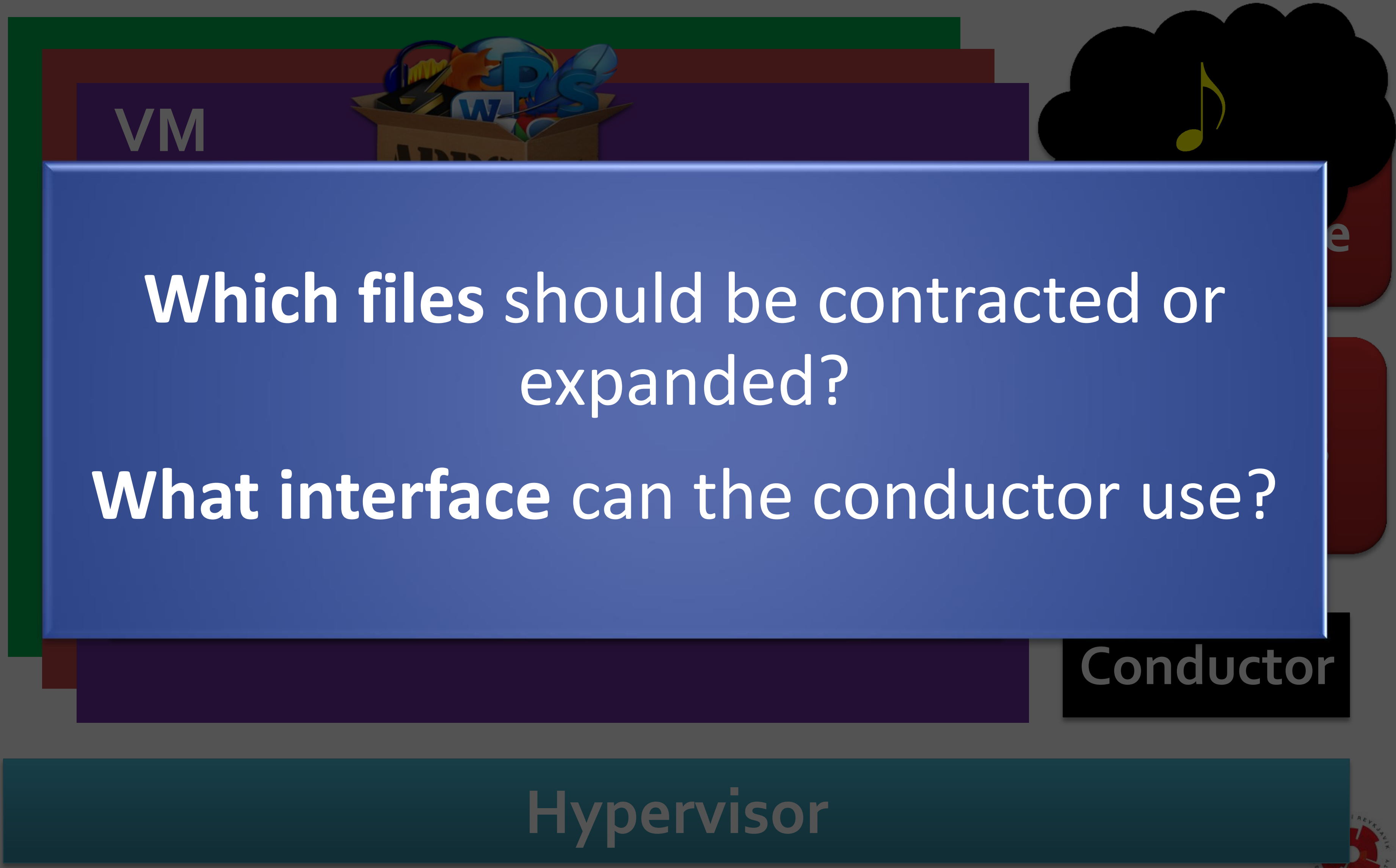
Virtual disk

250GB

Conductor

Hypervisor

The harmonium file system



Which files should be contracted or expanded?

What interface can the conductor use?

What files to contract/expand?

harm☯nium FS

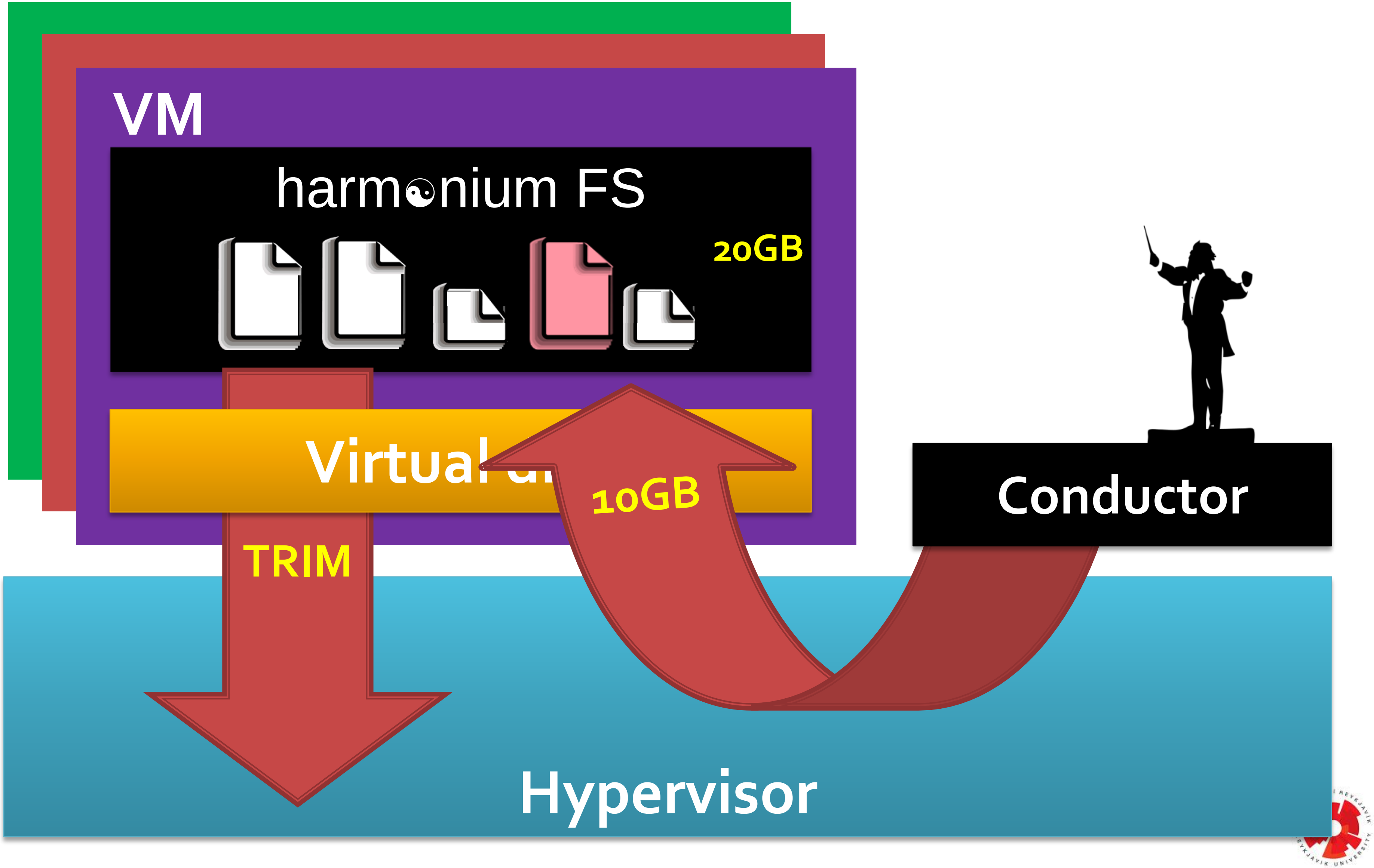
x GB limit

When will the file be accessed next?

How much time will it take to expand?

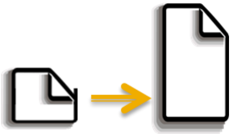
- Feed info into optimization mechanism
  - Choose files with minimum total expansion latency s.t. contraction saves sufficient space.
  - Most competitive algorithm: Greedily choose files on LRU list to maximize ratio of space savings to expansion latency

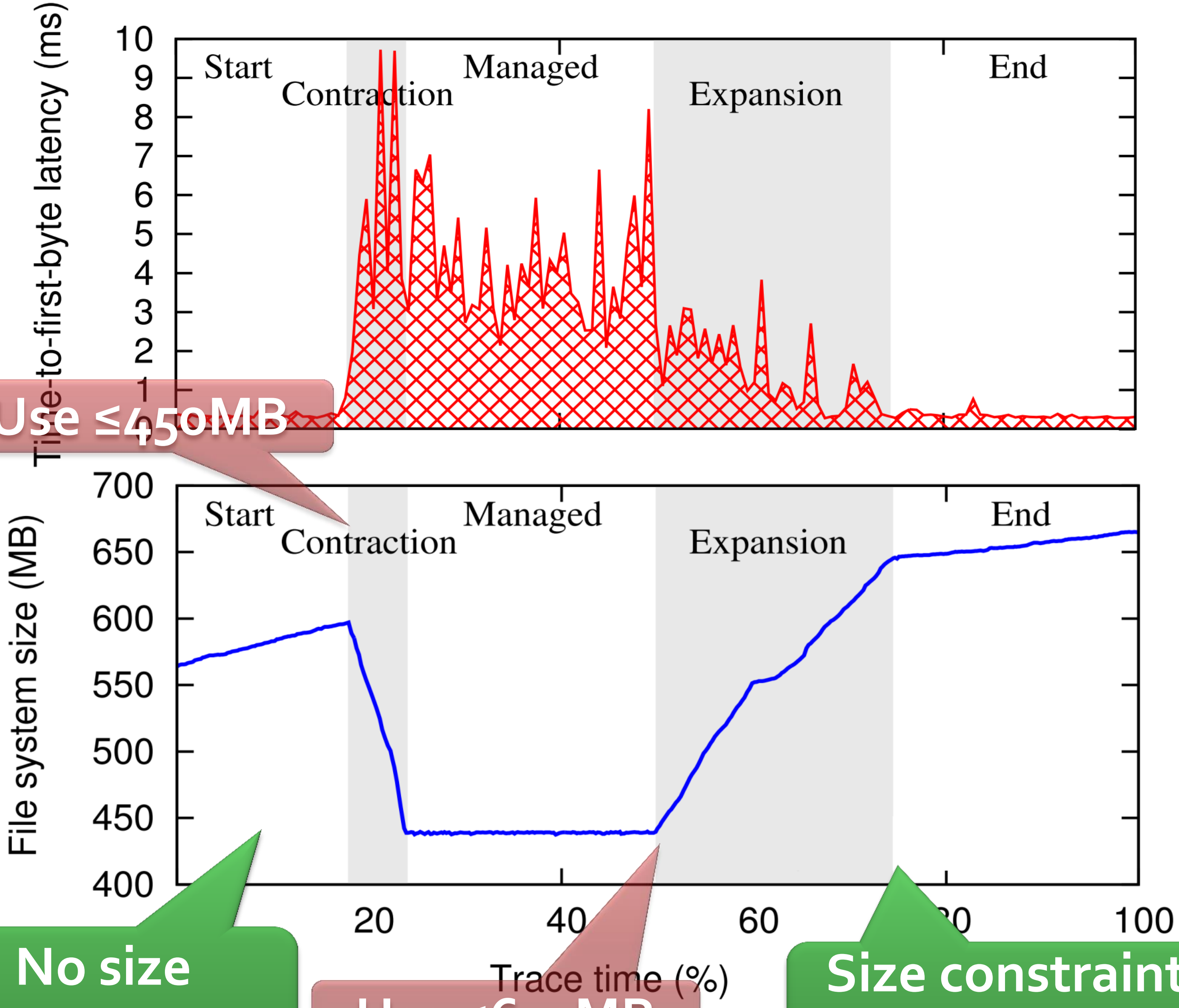
Conductor interface



Evaluation

- User-space FUSE prototype
  - Conductor: Python, via UNIX sockets
- Workload
  - Set of 54,000 patch files applied in chronological order to the Linux kernel

- Motif 
  - Network storage via scp
- Measure latency to access first byte



Use ≤450MB

No size constraints

Use ≤650MB

Size constraint lifted

Conclusion

Elastic performance/capacity trade-off for storage in VMs

Problem

Applications store **ephemeral** data on secondary storage

But storage stacks provide **durability** for every file

harm☯nium

Associate ***motifs*** with every file, allowing reconstruction

Contract/expand files to minimize access latency

Results

Best algorithm: LRU **greedily** maximizing space/latency

Fully functional **FUSE** prototype

Extra slides

Theoretical formulation

- 0-1 Knapsack (NP-complete)
  - S = space needed to save
  - e_i = expected expansion latency for file *i*
  - s_i = expected storage savings for file *I*

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} e_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} s_i x_i \geq S, \\
& x_i \in \{0,1\} \quad , \quad 1 \leq i \leq n
\end{aligned}
$$

- **Reduction:** Choose the files ***not*** to include
- **APX-knapsack** takes ***O(nW)*** time, prohibitive

Related work

- Trade-off between storage footprint and performance
  - Usually in distributed settings
  - Sierra (EuroSys 2011), Rabbit (SOCC 2011), Springfs (FAST 2014), …
- These systems maintain 1 to N copies of each file
- harm☯ium, however, maintains “0 to 1” copies of each file

But isn’t this just …

- A compression file system?
- A glorified cache?
- A de-duplication system?
  - Harm☯ium can support arbitrary motifs:
    - Compression of rarely accessed files
    - Remote network storage [scp/rsync/nfs/…]
    - Pull from repositories [git/svn/…]
    - Re-wget data in Downloads folder
    - Resume torrent download [remove partial files]
    - System packages [retrieve from apt/debian/…]
    - Regeneration of data sets [ala Nectar]
    - …

Security concerns

- Motifs are really arbitrary code
  - Can cause system to hang, crash, corrupt data or consume resources wastefully
- Our current implementation is vulnerable
  - Motifs execute within the same process as the FS
  - Isolation by virtualization too coarse-grained
  - Sandboxing great for security, may be slow
- Ongoing work: require authorization
  - Users specifically approve running of motifs generated by those of lesser privilege or fewer capabilities

Computation vs. storage

- *"But isn't computation more expensive than storage?"*
- Underlying principle of our work:

Computation, Network and Storage are fungible

- Harmonium allows use of Computation or Network when Storage is scarce
- Other parts of the trade-off interesting as well!