STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers
Virtual Conference
September 28-29, 2021

Enabling Heterogeneous Memory in Python

Dr. Daniel Waddington, IBM Research

Disclaimer

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

IN ADDITION, THIS INFORMATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.

IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

• CREATING ANY WARRANTY OR REPRESENTATION FROM IBM (OR ITS AFFILIATES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS); OR

• ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF IBM SOFTWARE.

IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Talk Outline

- Presentation around the emergence of heterogeneous memories and the challenge of legacy s/w integration
- Presentation of prototype PyMM solution demonstrating a new approach to integrating heterogeneous memory with legacy code in the Python data science ecosystem

Learning Objective 1: Understand the emergence of heterogeneous memories (e.g., Optane Persistent Memory, CXL-attached)

Learning Objective 2: Understand the challenges facing integration of legacy s/w with new memory technology

Learning Objective 3: Introduction and demonstration of PyMM

CXL and Emerging Heterogeneous Memory

CXL (Compute eXpress Link)

- Compute Express Link™ (CXL™) is a Cache-Coherent Interconnect for Processors, Memory Expansion and Accelerators
- Resource sharing for higher performance, reduced software stack complexity, and lower overall system cost
- Industry consortium with 57 members (including Intel, AMD, ARM, IBM, NVIDIA)
- CXL 2.0 specification now approved
- H/W IP available 2021
- Expect to see CXL-capable processors and server platforms in 2022

https://www.computeexpresslink.org/
https://www.computeexpresslink.org/resource-library

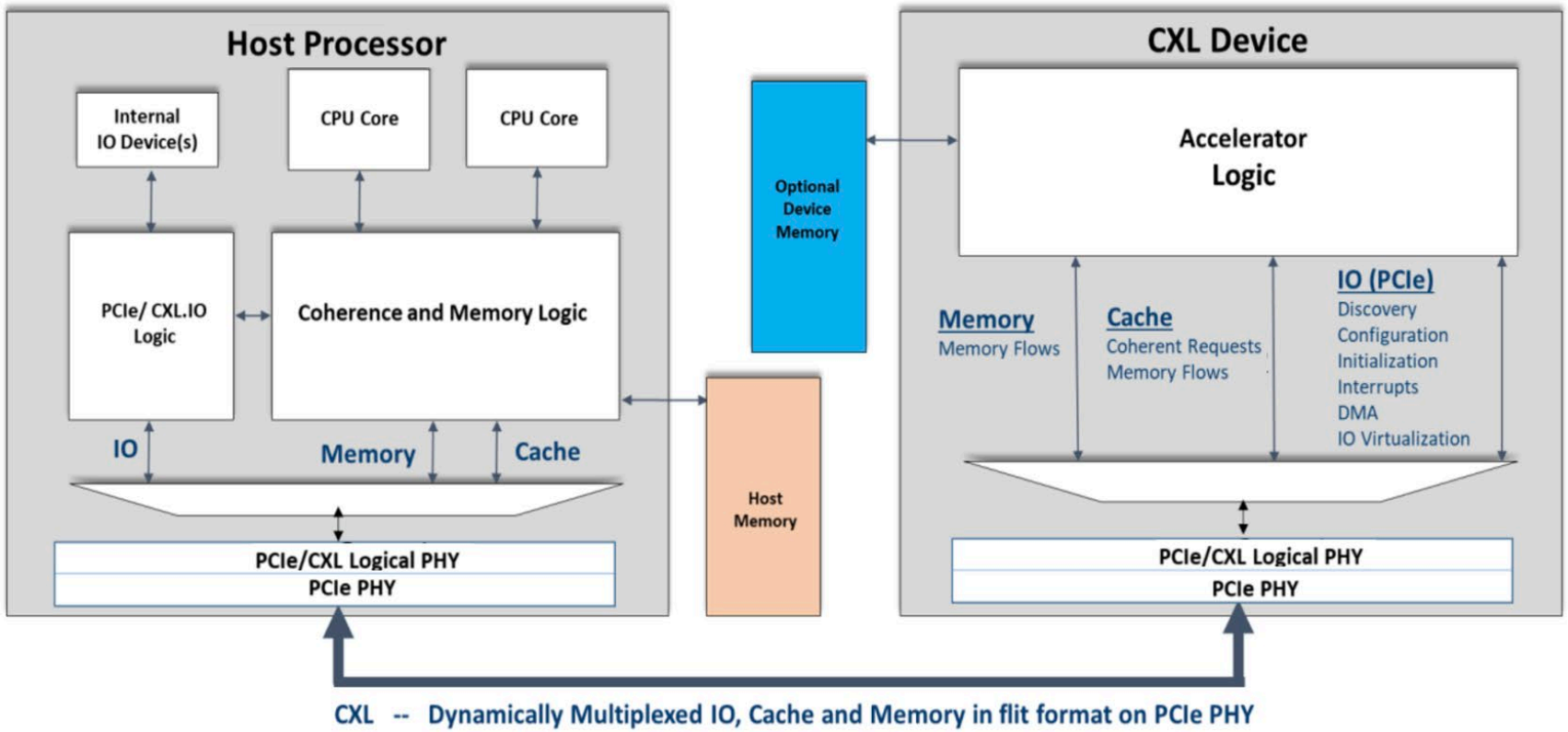
CXL Protocols

- The CXL transaction layer is compromised of three dynamically multiplexed sub-protocols on a single link

CXL.io
Discovery, configuration, register access, interrupts, etc.

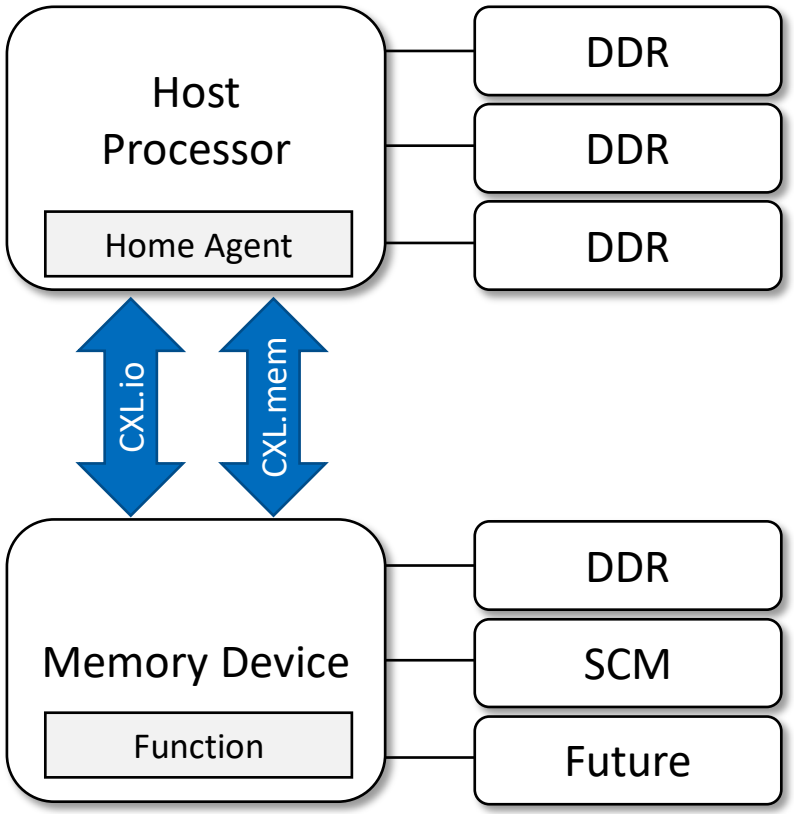
CXL.cache
Device access to processor memory

CXL.Memory
Processor access to device attached memory

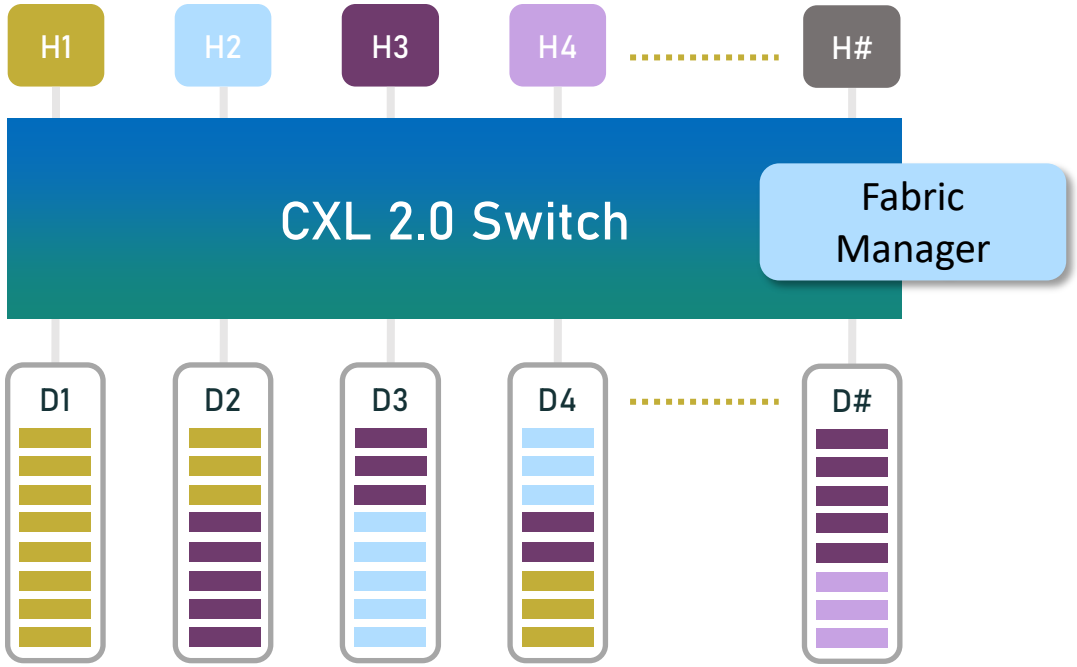


CXL Type 3 Devices

CXL1.1 Memory Expander



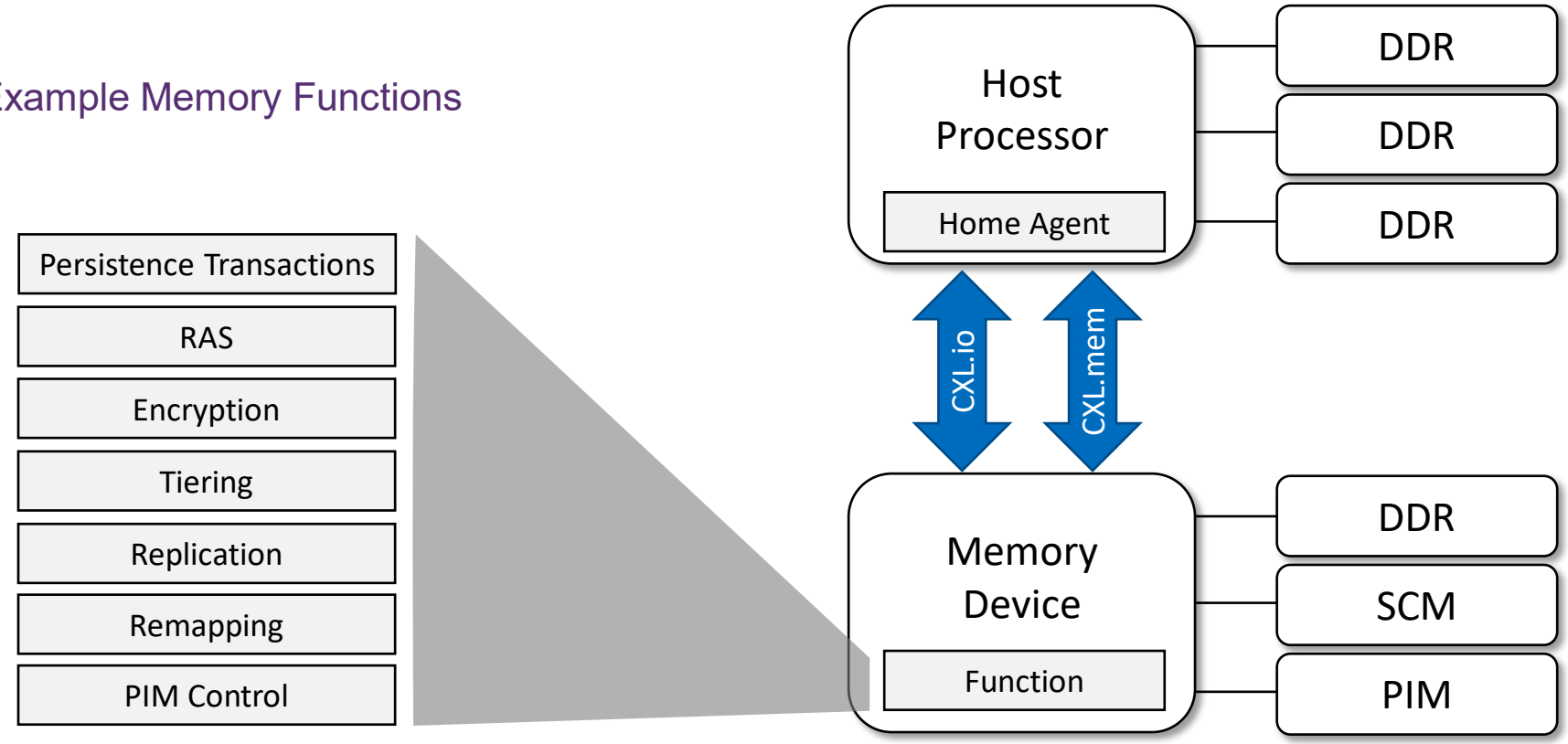
Memory Pooling with Multiple Logical Devices



https://members.computeexpresslink.org/wg/CXL/document/1218

Intelligent Memory Functions

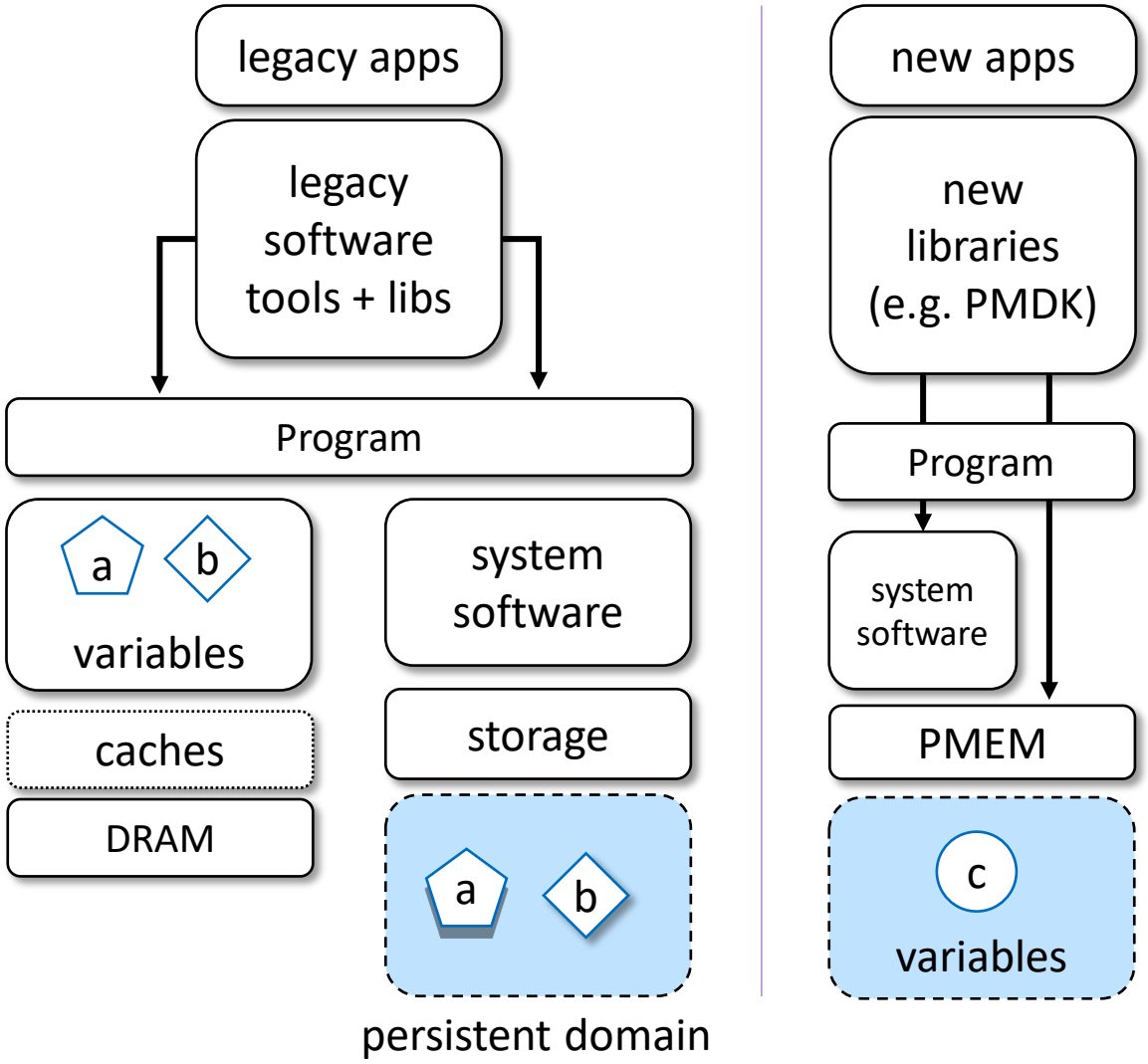
Example Memory Functions



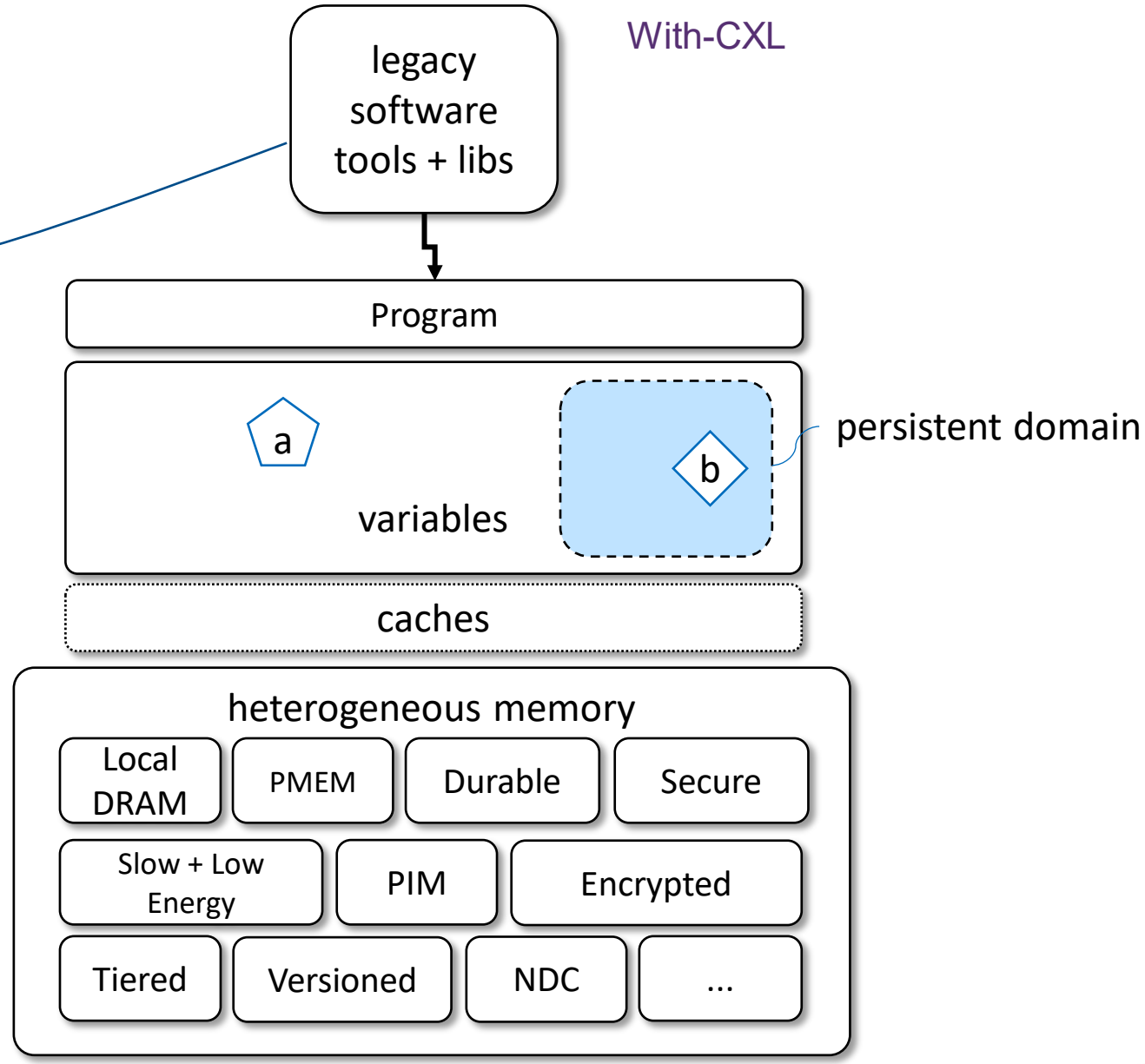
Software Integration of New Memories

The Legacy Problem

Pre-CXL



The Legacy Problem



A Perfect World...

- Legacy applications and libraries can still be used when exploiting new memory capabilities
  - easy to adopt
  - minimal code change for legacy apps
- Programming language, compiler and OS technology evolve to support new memory technology for existing (partial value) and new applications (max value)

OK we can’t go all the way, but let’s see what we can do...

PyMM: Python Memory Management

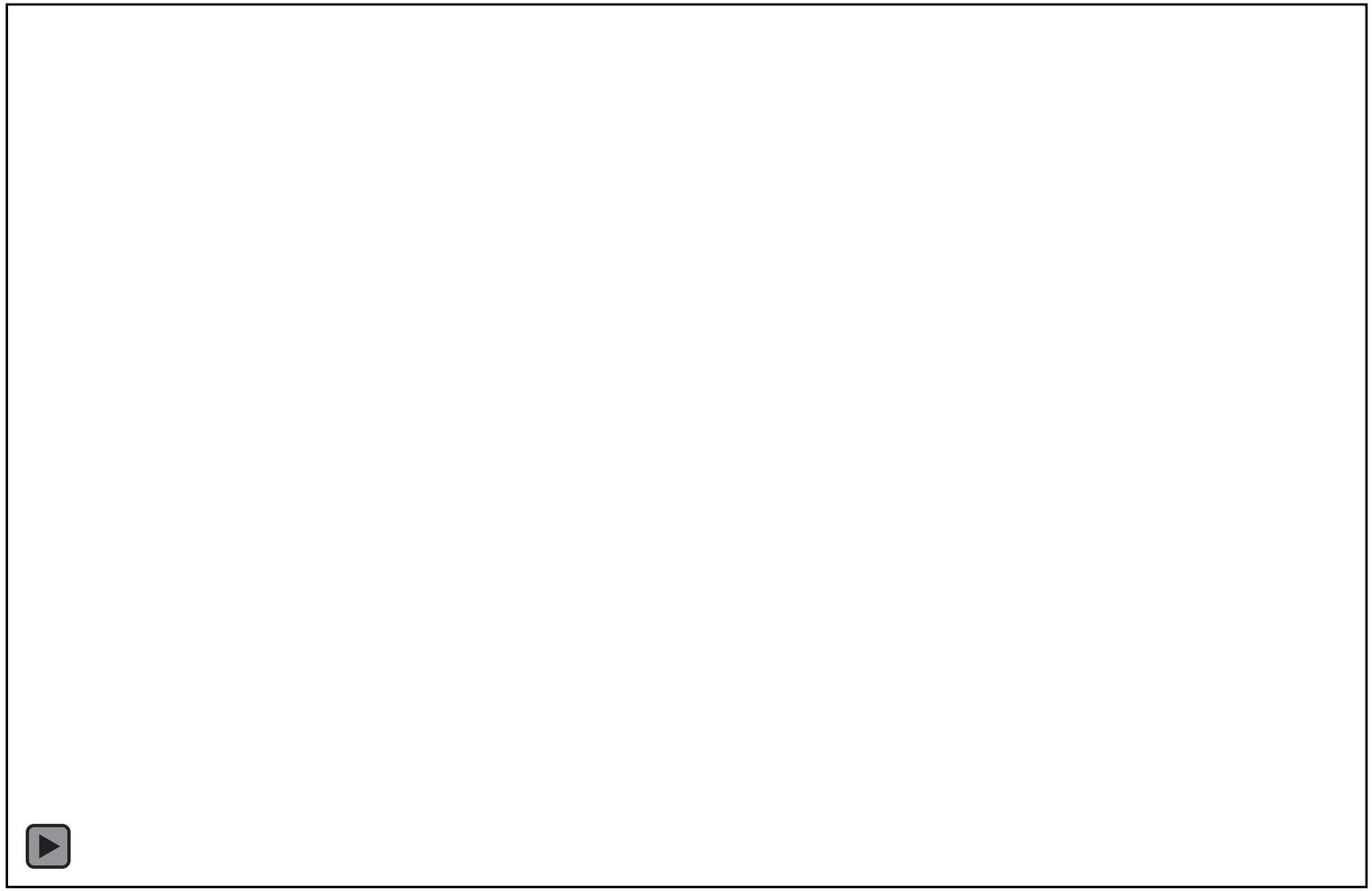
PyMM

- Provide Python 3 libraries that make the integration of different memory types very easy
  - volatile or non-volatile
  - prototype manages local DRAM, file mmap'ed DRAM and persistent memory
  - current testing with NumPy and PyTorch
- Focused on data science domain and data science programmers
- Available as part of the IBM MCAS (Memory Centric Active Storage) open-source project
  - Linux only
  - https://github.com/IBM/mcas/

Approach

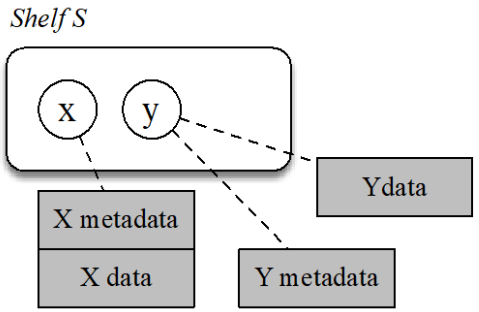
- Based around the concept of a “shelf” which is a logical collection of variables that are stored in a specific set of memory resources
- Only shelf types can be put on a shelf
- Leverages poly-morphism and sub-classing to create “special” handling of variables while retaining compatibility with existing libraries – shelf types
- Focus on heavily used types
  - basic types
  - data science types - NumPy ndarray, PyTorch tensor
- Persistent shelves (based on persistent memory) retain variables across program and machine resets

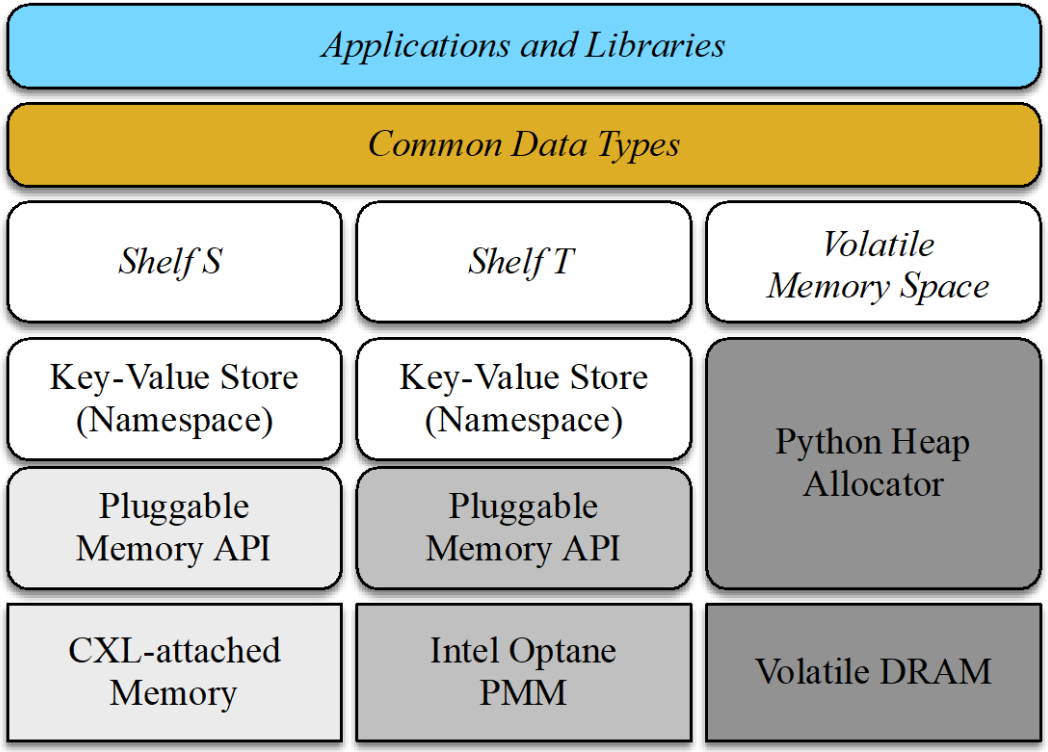
Shelf



Shelves – Under the hood

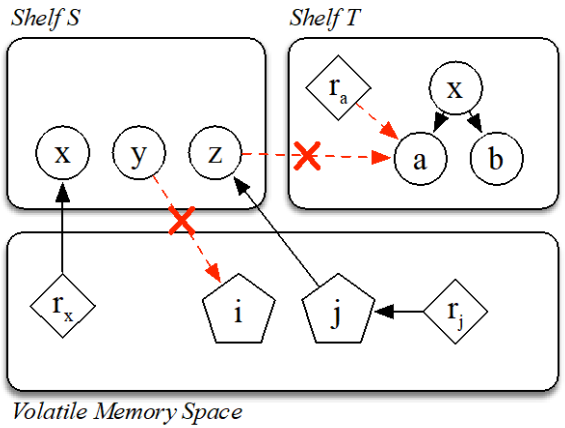
- Shelfs are supported by a key-value index and a heap allocator that is associated with one or more regions of a particular memory
  - variable names are scoped to the shelf
- Shelfs on persistent memory require crash-consistent index and allocator



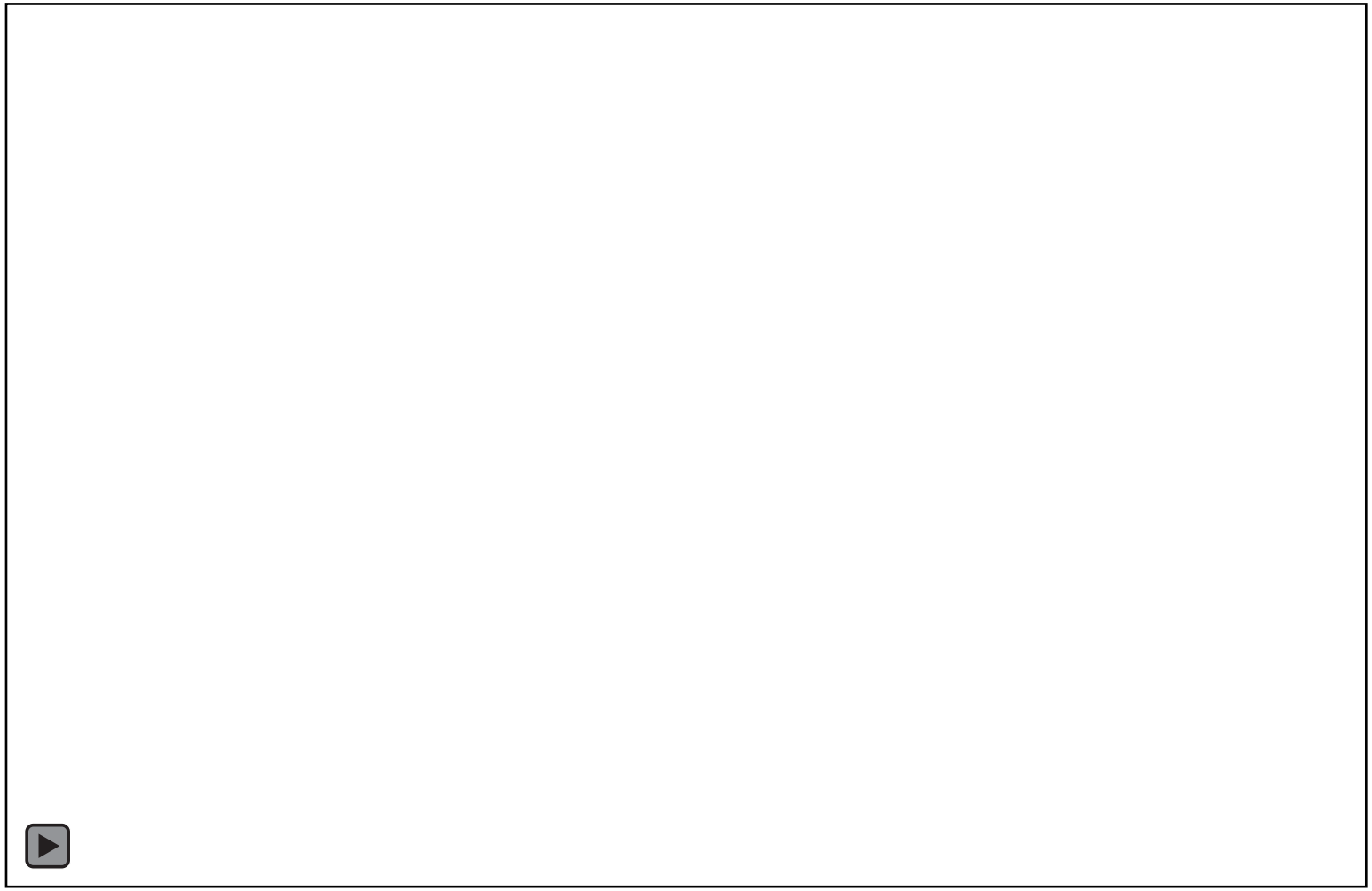


Shelf types and references

- Shelf types are instantiated from shadow types or through a copy-constructor that takes an instance of the volatile counterpart
  - shadow type (e.g., `shelf.x = pymm.ndarray((9,9,),dtype=np.int32)`)
  - RHS copy-constructor (e.g., `shelf.x = np.identity(9,dtype=np.int32)`)
- Shelves do not support conventional Python references
  - references between shelf objects are stored as variable names
- Shelved items do not reference other volatile variables

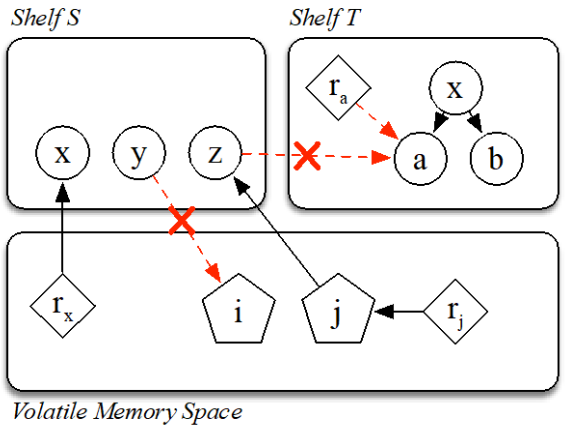


Construction

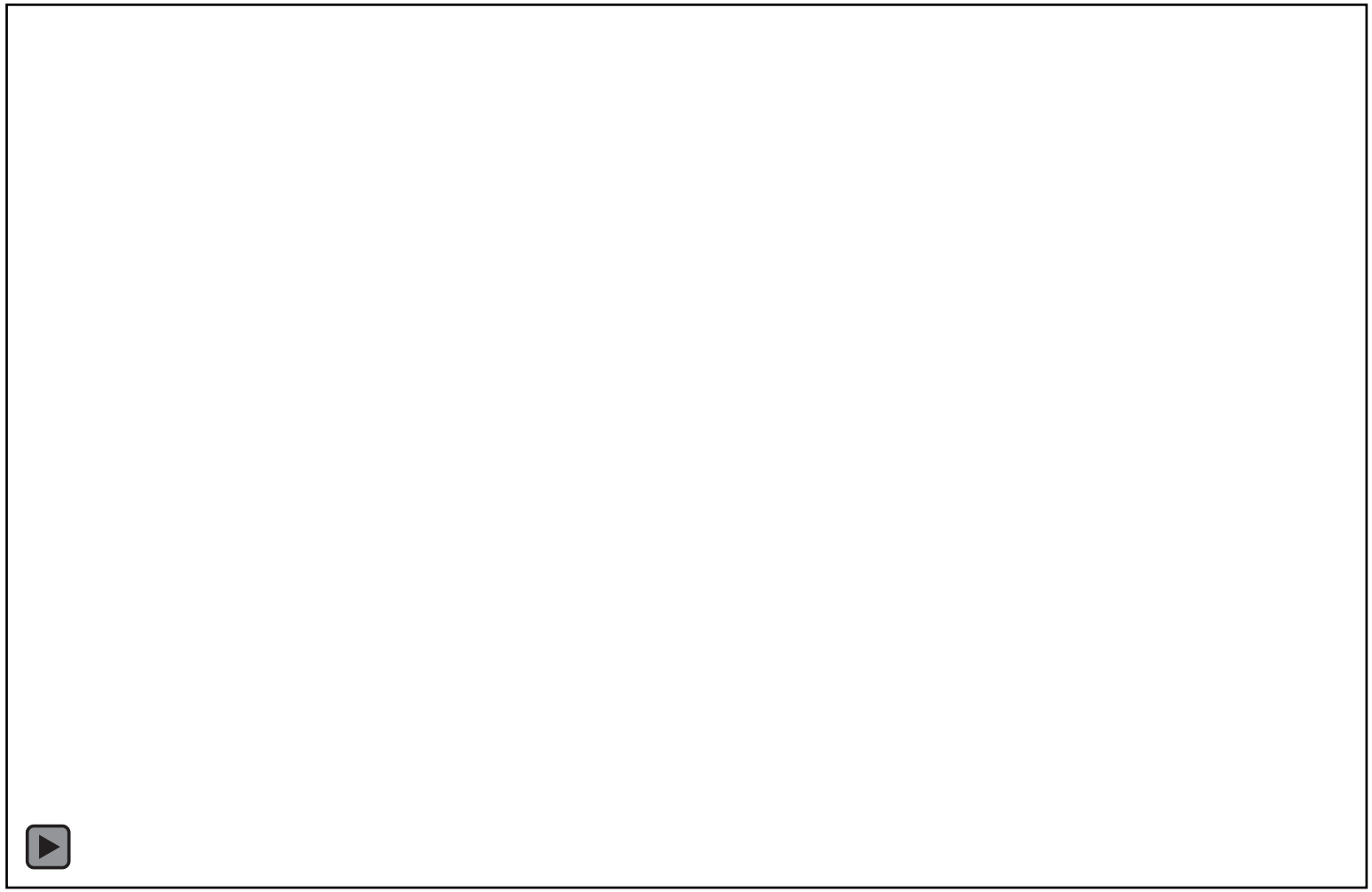


Shelf types and references

- Shelf types are instantiated from shadow types or through a copy-constructor that takes an instance of the volatile counterpart
  - shadow type (e.g., `shelf.x = pymm.ndarray((9,9,),dtype=np.int32)`)
  - RHS copy-constructor (e.g., `shelf.x = np.identity(9,dtype=np.int32)`)
- Shelves do not support conventional Python references
  - references between shelf objects are stored as variable names
- Shelved items do not reference other volatile variables

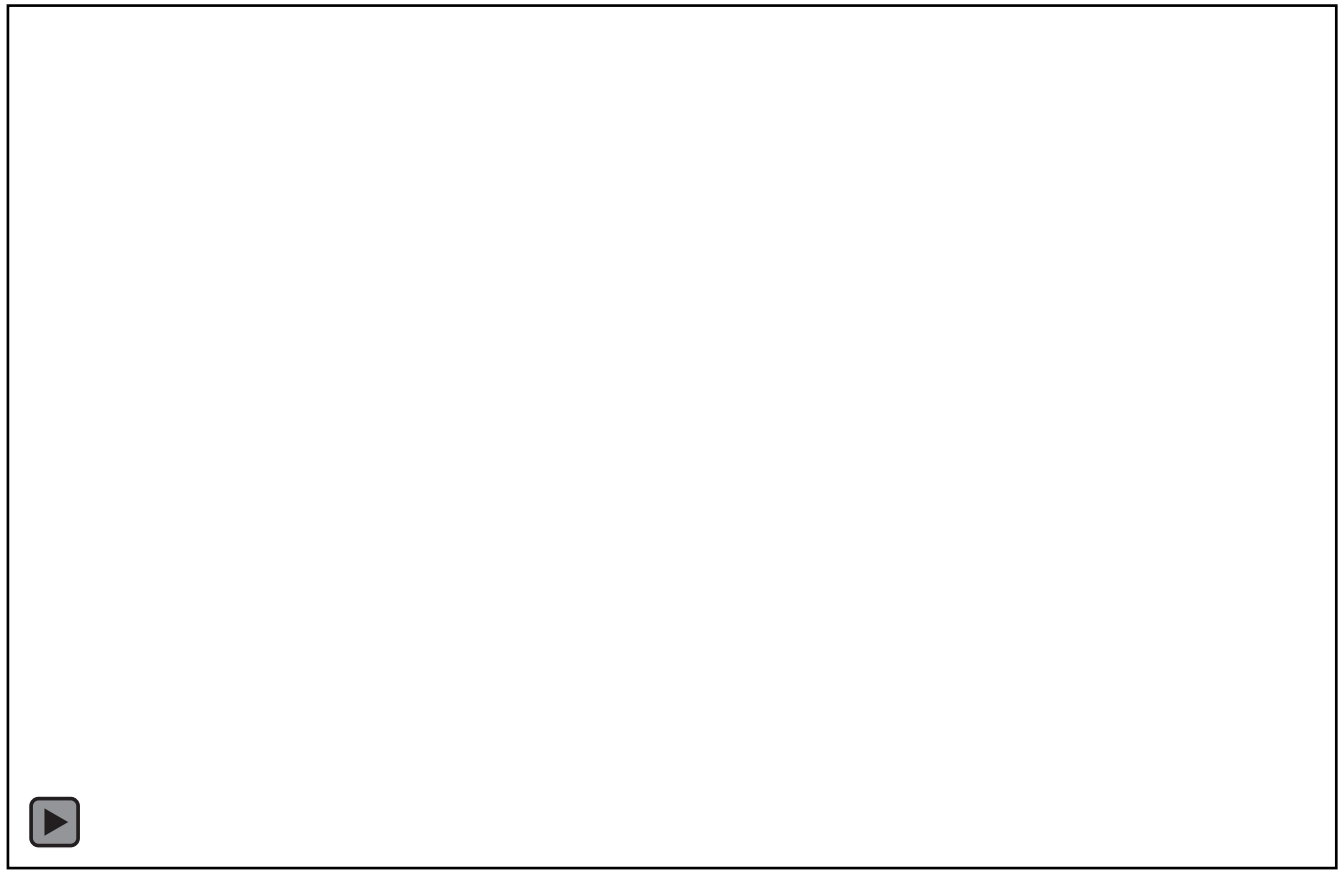


References



Type Conversion

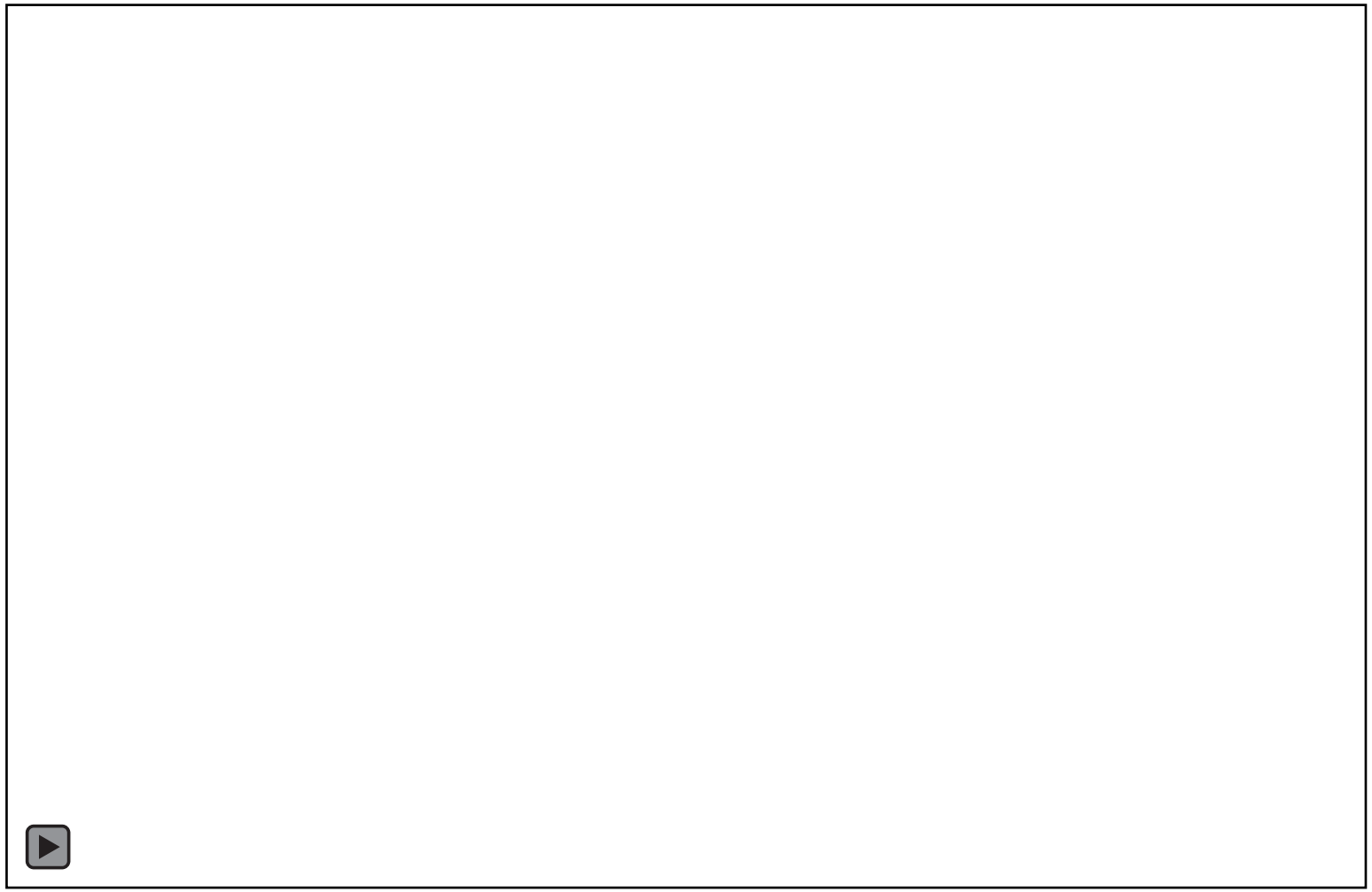
- Shelf types used in expressions are “coerced” to their volatile counterpart type
  - pymm.shelved_ndarray → np.ndarray



Transient Memory Mode

- Addresses the problem of needing equivalent memory footprint for RHS expression evaluation in main memory
- PyMM allows persistent memory or file-backed paged main memory to temporarily provide space for RHS evaluation
  - pmem fsdax file and/or backing file
- The memory is transient because it is released after instantiation

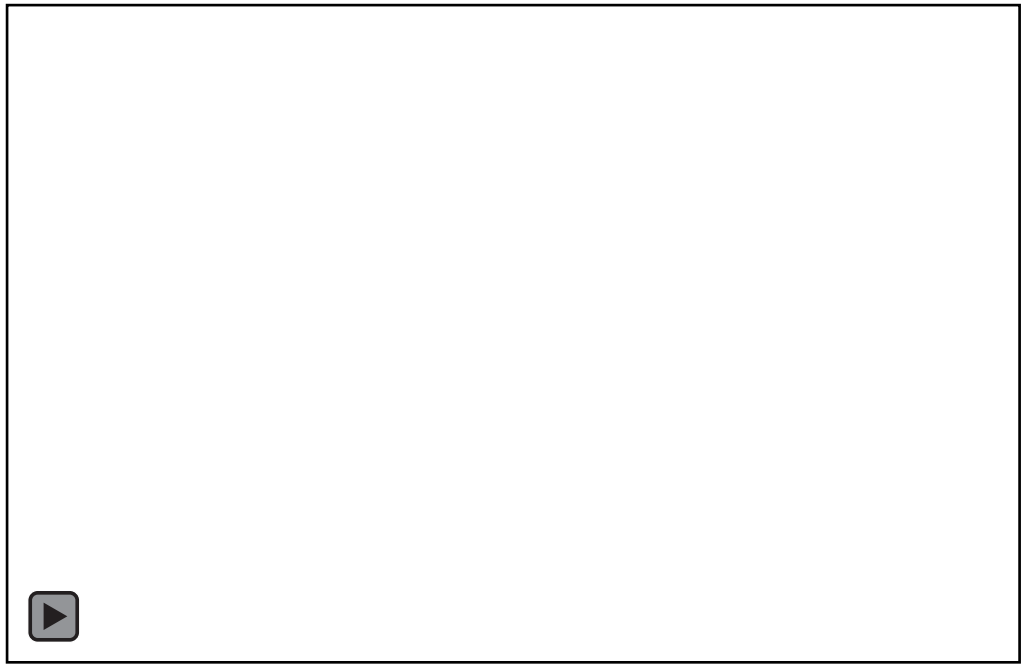
Transient Memory Mode



Persistent Memory

- Shelves in persistent memory (e.g. Intel Optane) are crash-consistent and transactional
  - reset-event at any time consistent state can always be recovered

```
def tx_begin(self):

    #pymmcore.valgrind_trigger(1)

    if self._use_sw_tx:
        if self._debug_level > 0:
            print('tx_begin')
        self.__tx_begin_swcopy()

def tx_commit(self):

    #pymmcore.valgrind_trigger(2)

    if self._use_sw_tx:
        if self._debug_level > 0:
            print('tx_commit')
        self.__tx_commit_swcopy()
```



Summary

- We believe that CXL will bring the adoption of new types of memory and intelligent memory capabilities
- Supporting this new technology with existing software ecosystems is critical to adoption
- PyMM is an early prototype that presents a new way of thinking about persistent and heterogeneous memory integration specifically in the Python ecosystem
- Available at: https://github.com/IBM/mcas/

Please take a moment to rate this session.

Your feedback is important to us.

Presenter biography

Dr. Waddington is currently a Principal Research Staff Member in the Data and Storage Systems Research Group, IBM Research Almaden, California. His current focus is around non-volatile memory and high-performance distributed storage systems. Dr. Waddington holds a Ph.D. from Lancaster University, UK and has over thirty research publications across a broad range of topics pertinent to systems building and performance optimization. He has over twenty years of industrial research experience and has previously worked at Bell Labs, Lockheed Martin Advanced Technology Labs, and Samsung Research America. During his career he has applied his research to a broad array of applications including telecommunication switches, mission control and avionics systems, mobile services and more recently storage systems and data processing solutions. He is a senior member of the ACM.

