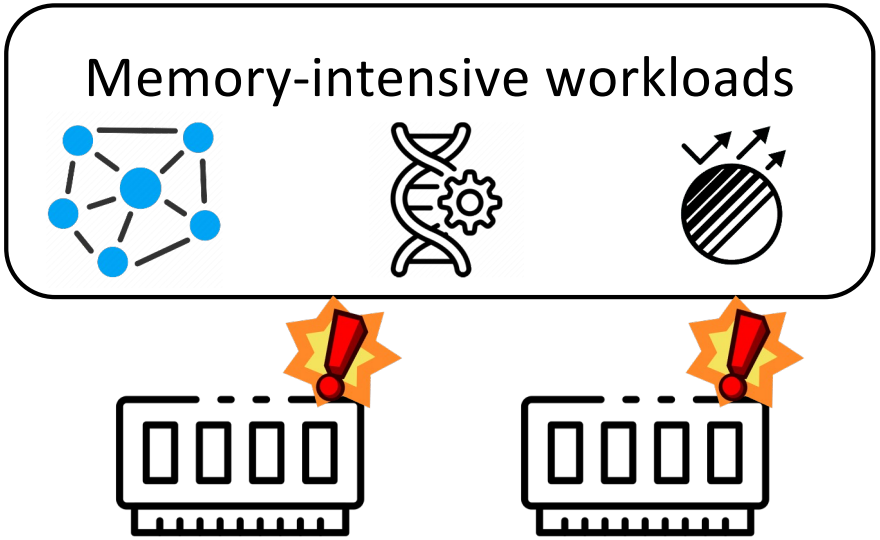
Towards Efficient Python Interpreter for Tiered Memory Systems

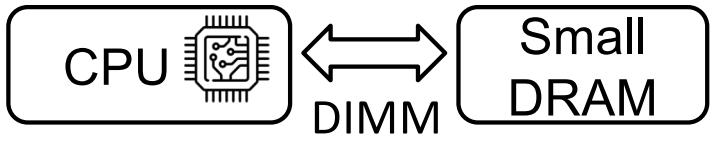
Yuze Li[1], Shunyu Yao[1], Jaiaid Mobin[2], M. Mustafa Rafique[2],
Dimitrios Nikolopoulos[1], Kirshanthan Sundararajah[1], Huaicheng Li[1], Ali R. Butt[1]

[1]Virginia Tech, [2]RIT

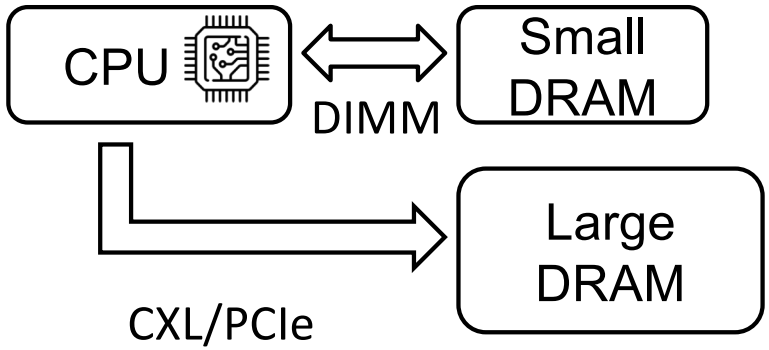
Memory Capacity Bottleneck



Tiered Memory Systems



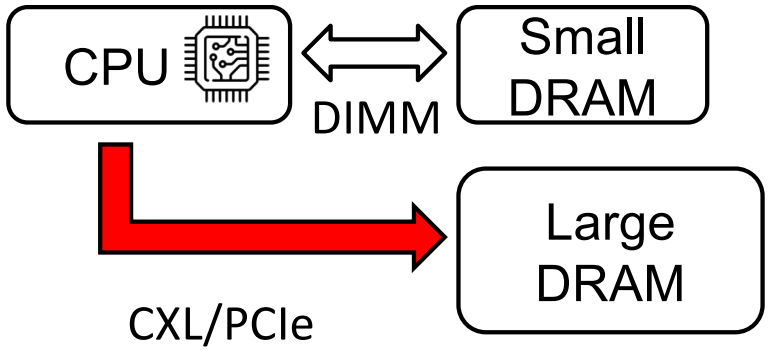
Tiered Memory Systems



CXL offers practical memory expansion ☺

- Load/Store access over PCIe
- Large capacity with lower cost

Tiered Memory Systems



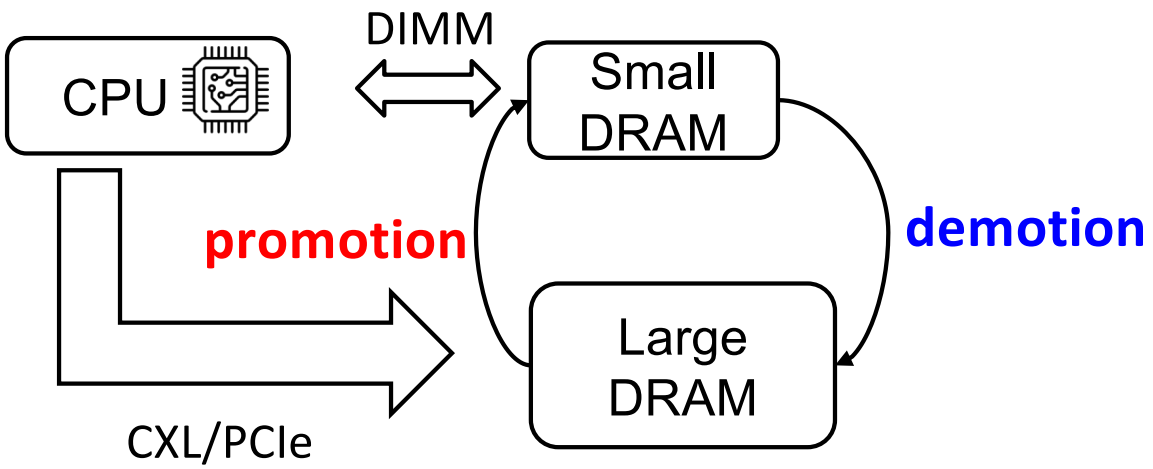
CXL offers practical memory expansion ☺

- Load/Store access over PCIe
- Large capacity with lower cost

CXL has higher access latency than local DRAM ☹

- With additional 70~90ns (~2x)

Tiered Memory Systems



State-of-the-art Data Migration on Tiered Memory System

- OS level
  - Transparency

State-of-the-art Data Migration on Tiered Memory System

- OS level
  - Transparency
  - Hard to apply program-behavior-specific optimizations

State-of-the-art Data Migration on Tiered Memory System

- OS level
  - Transparency
  - Hard to apply program-behavior-specific optimizations
  - Coarse granularity at page level

State-of-the-art Data Migration on Tiered Memory System

- OS level
  - Transparency ☺
  - Hard to apply program-behavior-specific optimizations ☹
  - Coarse granularity at page level ☹
- Runtime level
  - Explicitly control fine-grained object-level offloading ☺

State-of-the-art Data Migration on Tiered Memory System

- OS level
  - Transparency ☺
  - Hard to apply program-behavior-specific optimizations ☹
  - Coarse granularity at page level ☹
- Runtime level
  - Explicitly control fine-grained object-level offloading ☺
  - Intrusive ☹

State-of-the-art Data Migration on Tiered Memory System

- OS level
  - Transparency ☺
  - Hard to apply program-behavior-specific optimizations ☹
  - Coarse granularity at page level ☹
- Runtime level
  - Explicitly control fine-grained object-level offloading ☺
  - Intrusive ☹
  - Supports only C/C++, JVM-based  

Python Popularity

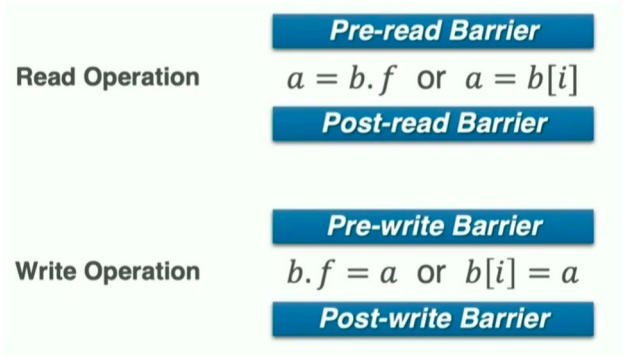
*IEEE Spectrum

Can we achieve both transparency and object level tracing in Python runtime?

Challenge 1: How to obtain Python object temperature?

Challenge 1: How to obtain Python object temperature?

JVM-based

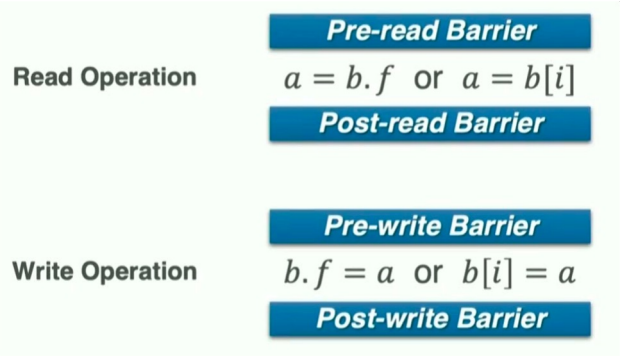


C++

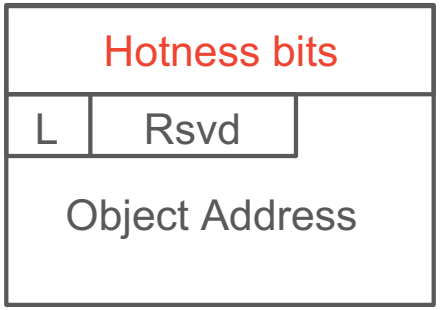
CPython

Challenge 1: How to obtain Python object temperature?

JVM-based



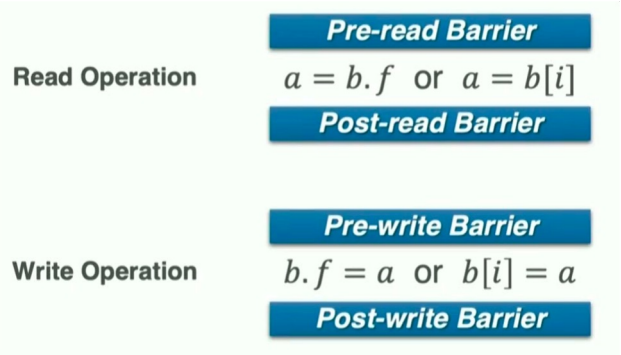
C++



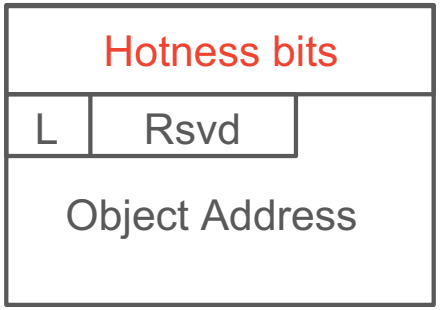
CPython

Challenge 1: How to obtain Python object temperature?

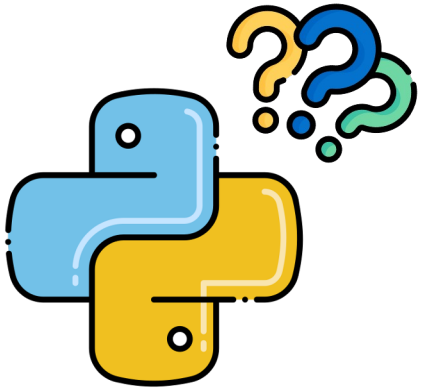
JVM-based

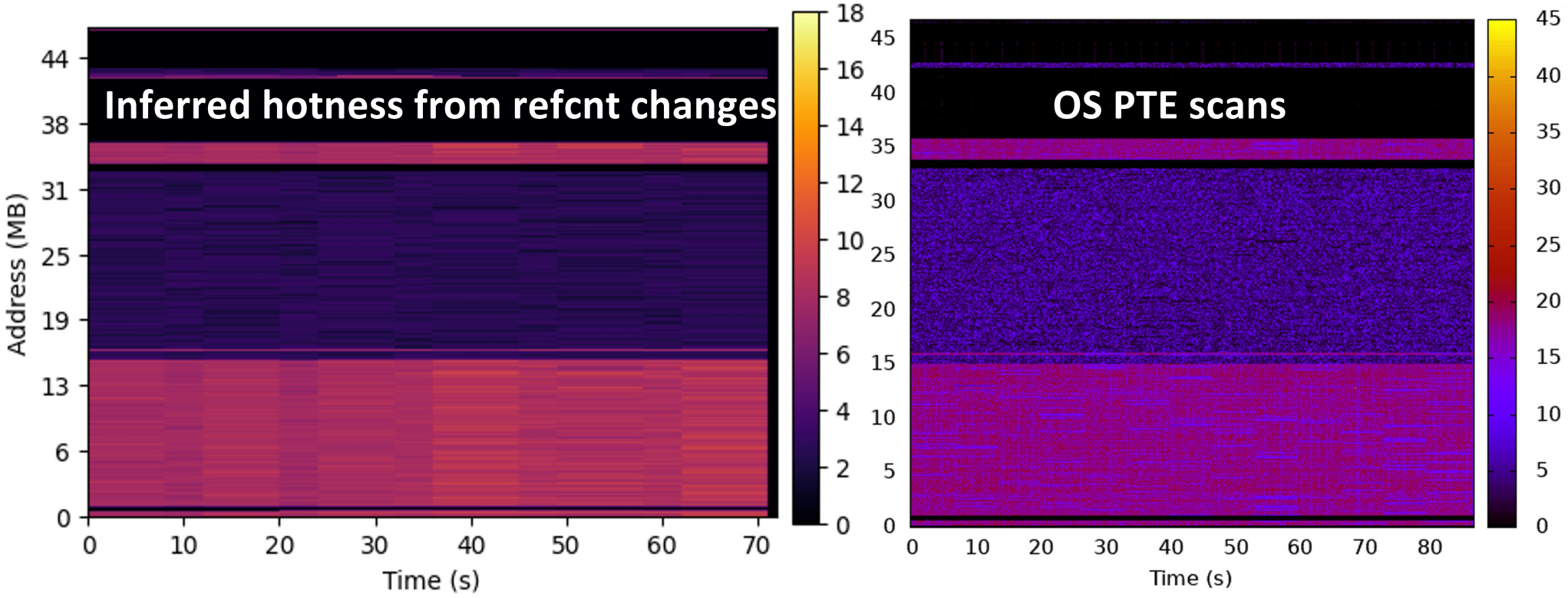


C++



CPython





Key observation: Use CPython reference counting **changes** to infer Python object accesses

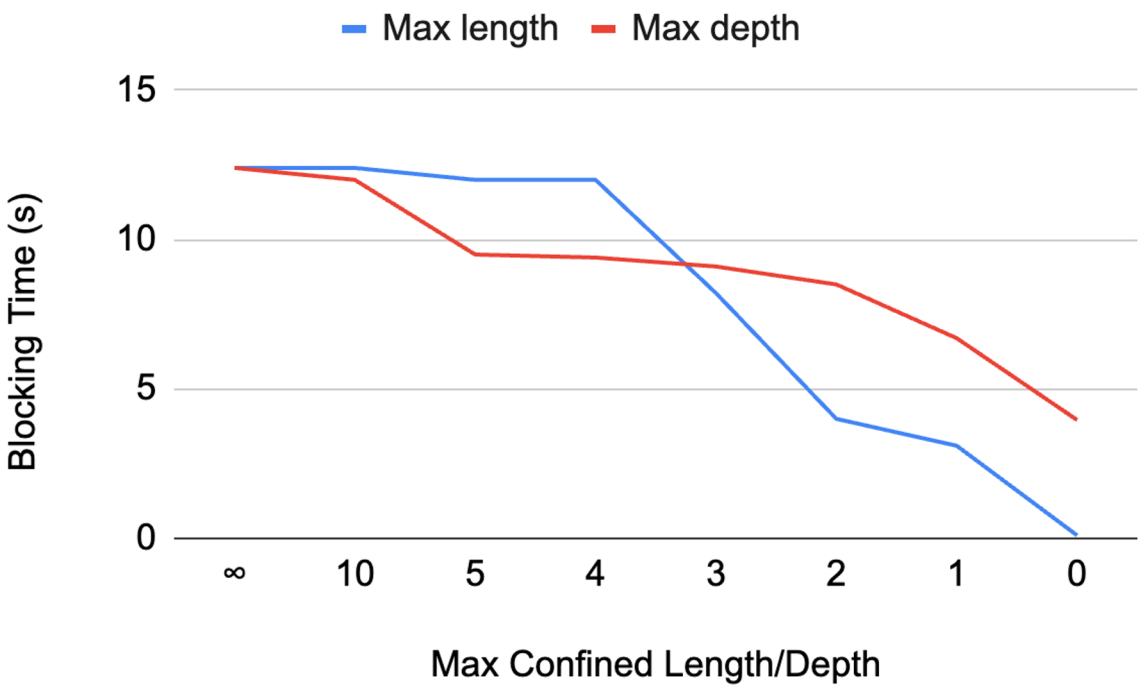
Challenge 1: How to obtain Python object temperature?

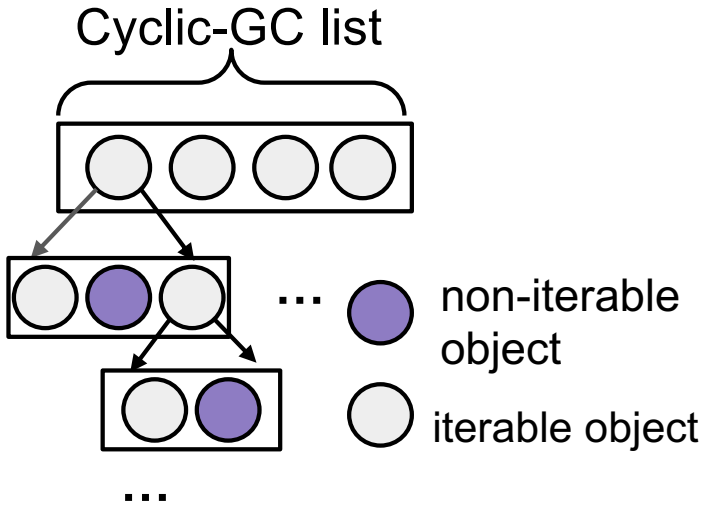
- Infer from reference count changes

Challenge 2: How to obtain live objects with low overhead?

- CPython GC holds only container objects
- Global Interpreter Lock

Cascade Tracing





Real heatmap from OS-based profiling.

Challenge 1: How to obtain Python object temperature?

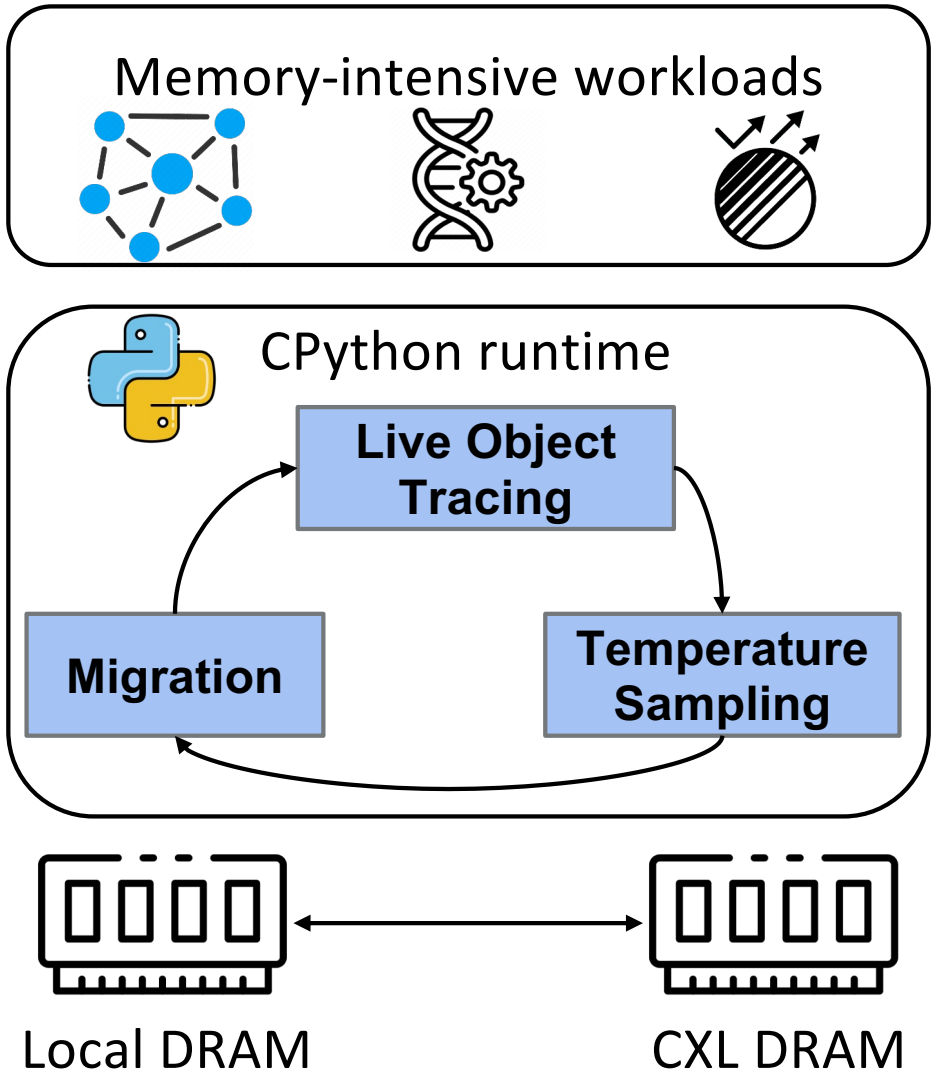
- Infer from reference count changes

Challenge 2: How to obtain live objects with low overhead?

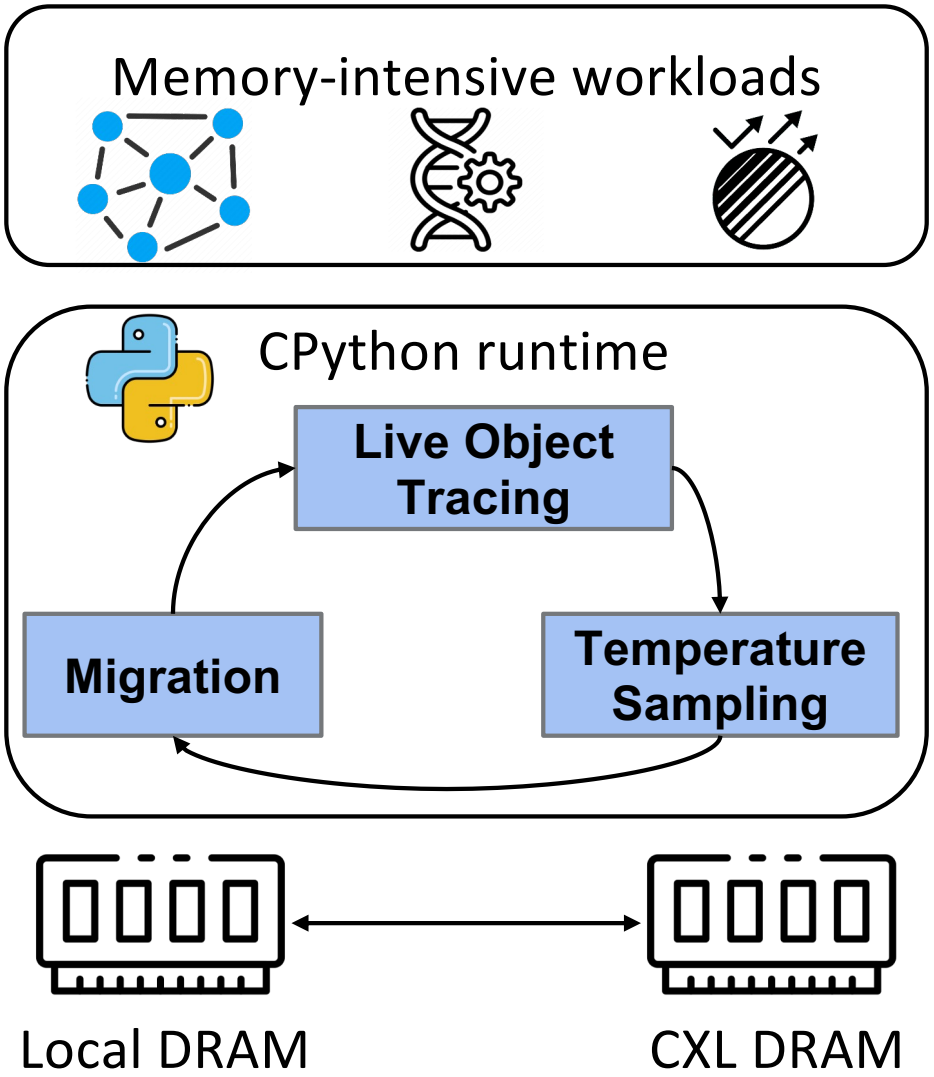
- Cascade tracing with confined length/depth

Challenge 3: How to handle native executions (external libs)?

- Rely on existing OS-based solutions, or function call stack inferences

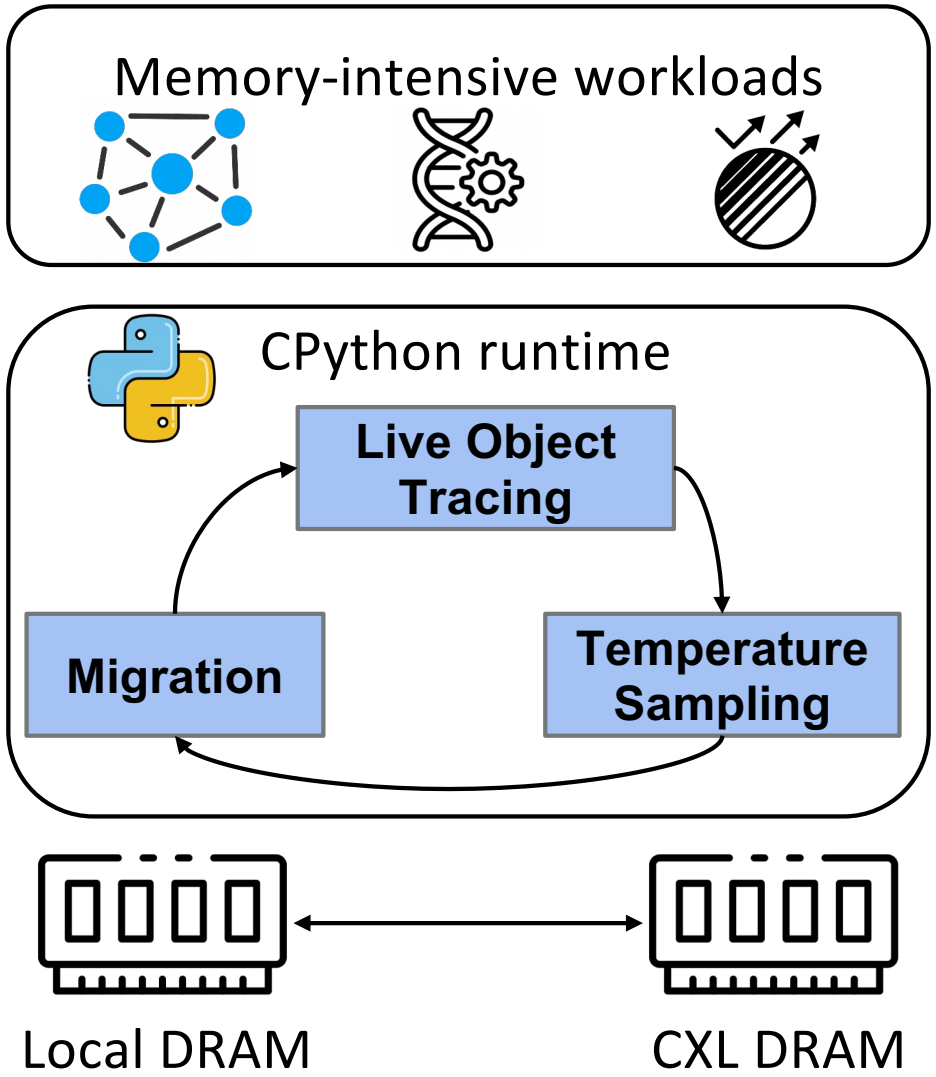


Fine-grained ✓



Fine-grained ✓

Transparency ✓



Fine-grained ✓
Transparency ✓
CPU efficiency ✓

Challenge 1: How to obtain Python object temperature?

- In progress

Challenge 2: How to obtain live objects with low overhead?

- Completed

Challenge 3: How to handle native executions (external libs)?

- In progress