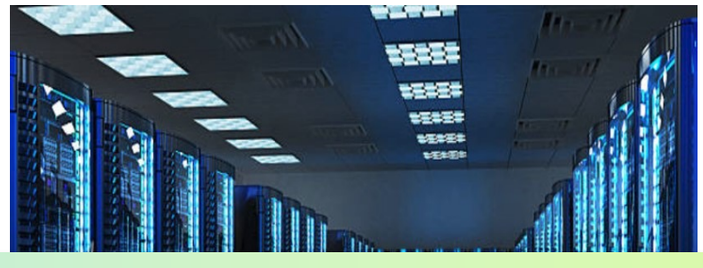
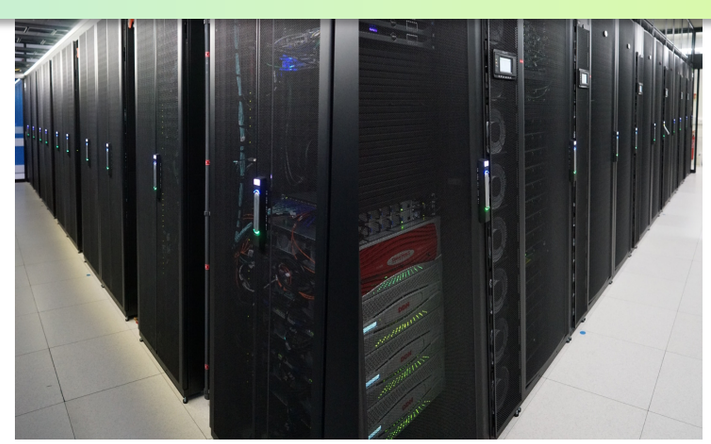
Pinpointing Resource Wastage with Profiling

Pengfei Su
Assistant Professor
Department of Computer Science and Engineering

Resource Concerns are Everywhere





Programs need to be efficient at all scales

Time is Money

Drop sales by 1%
every 0.1s of latency

Drop traffic by 20%
every 0.5s of latency

Drop visitors by 40%
after 3s of latency

Classic Profiling Techniques

- Identify hotspot — high resource utilization
  - Time
  - CPU usage
  - Cache misses
  - Memory allocations
- Limitations
  - Not tell if resources are “**well spent**”
  - Need significant manual efforts to investigate root causes

Pinpoint resource wastage instead of utilization

Agenda

Profiling

- CPU memory wastage — ICSE'19, FSE'19, ICSE'22, CGO'23, TACO'23, CGO'26
- GPU memory wastage — ASPLOS'23, USENIX ATC'24, ASPLOS'26 (under review)

Wasteful Memory Operations

Silent load

```
x = A[i];
y = A[i];     y = x;
```

Silent store

```
A[i] = 10;
   x = A[i];
A[i] = 10;
```

Dead store

```
A[i] = 0;
A[i] = 10;
```

Two operations involved:
one is silent/dead
because of the killing one

Solution: binary analysis

A Motivating Example: SPEC CPU2006 Hmmer

- The compiler may do harm than good

Unoptimized

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];   First store
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k])
    mc[k] = sc;   Second store      Used
```

-O3 optimized

```
1 for (k = 1; k <= M; k++) {
2    R1 = mpp[k-1] + tpmm[k-1];
3    mc[k] = R1;   Dead store
4    if ((sc = ip[k-1] + tpim[k-1]) > R1)
5      mc[k] = sc;   Second store
```

The compiler: they "alias" to same memory location

Necessity of Binary Analysis

```
1 for (k = 1; k <= M; k++) {
2    R1 = mpp[k-1] + tpmm[k-1];
3    [redacted]
4    if ((sc = ip[k-1] + tpim[k-1]) > R1)
5      mc[k] = sc;
```

```
else
   mc[k] = R1;
```

Dead store

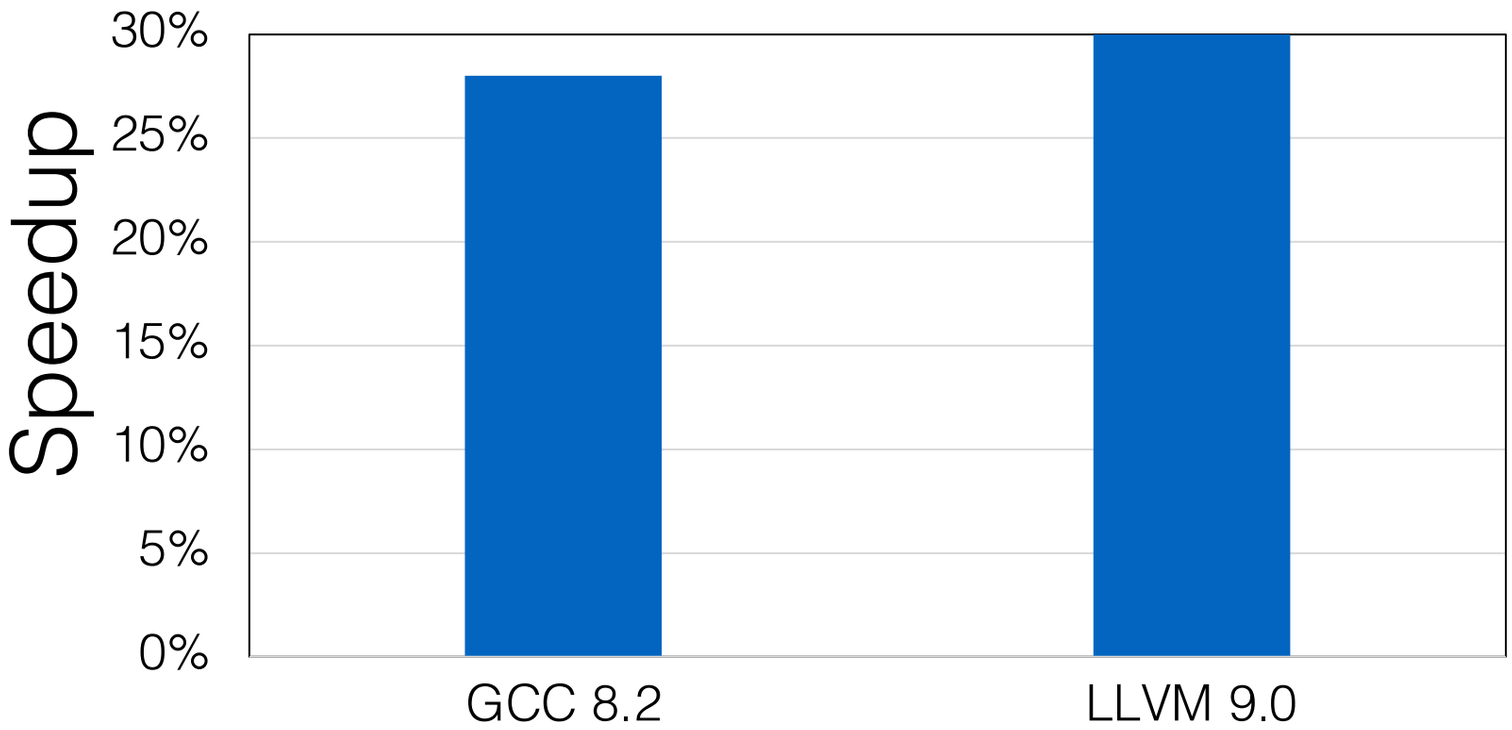
```
1 mov %r10,%rax,4), %ecx
2 add 0x0(%r13,%rax,4),%ecx   #mpp[k-1]+tpmm[k-1]
3 mov %ecx, 0x4(%rdx)         #assign mc[k]
```

The value of mc[k] in memory location 0x4(%rdx) is unused

```
10 mov %ecx, 0x4(%rdx)        #assign mc[k]
```

Optimization: conditional check

Hmmer's Speedup after Optimization



User Inputs

- Rodinia-3.1 backprop

```
1 for (j = 1; j <= ndelta; j++) {
2    new_dw = ETA * delta[j];
3    w[k][j] += new_dw;
4 }
```

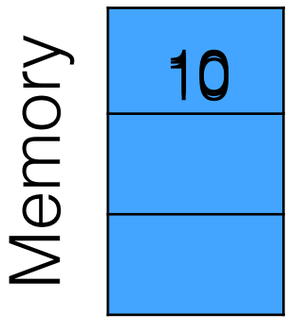
Optimization

```
1 for (j = 1; j <= ndelta; j++) {
2    if (delta[j] == 0) continue;
3    new_dw = ETA * delta[j];
4    w[k][j] += new_dw;
5 }
```

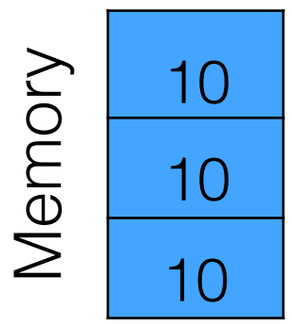
Optimization: conditional check
Speedup: 13%

Type of Silent Loads

- Temporal silent load
  - Repeatedly load the same value from the same memory location

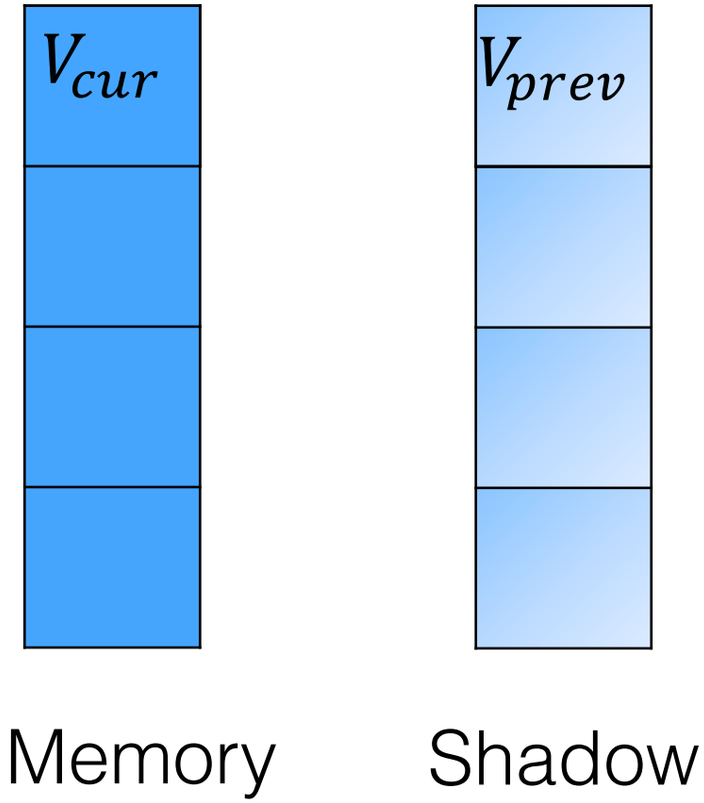


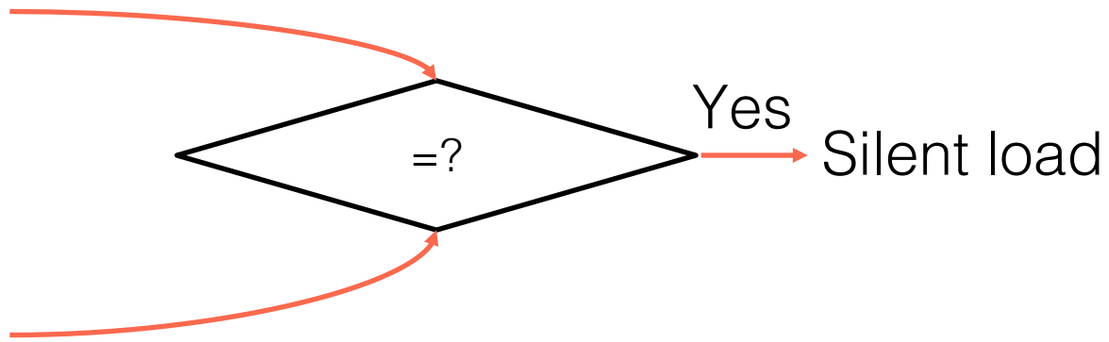
- Spatial silent load
  - Repeatedly load the same value from nearby memory locations



Temporal Silent Load Detection

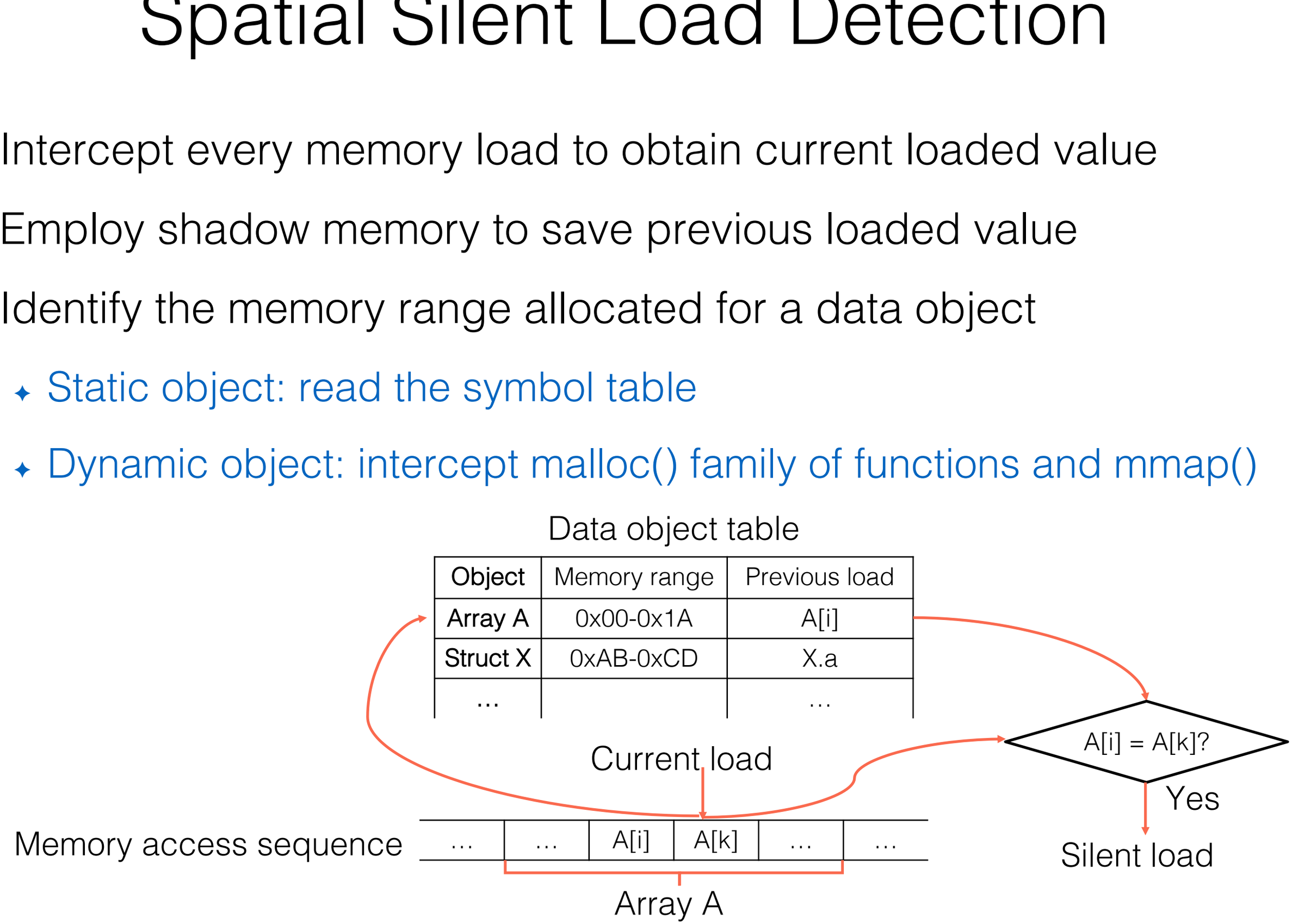
- Intercept every memory load to obtain current loaded value (V_{cur})
- Employ shadow memory to save previous loaded value (V_{prev})





Spatial Silent Load Detection

- Intercept every memory load to obtain current loaded value
- Employ shadow memory to save previous loaded value
- Identify the memory range allocated for a data object
  - Static object: read the symbol table
  - Dynamic object: intercept malloc() family of functions and mmap()



Case Studies

	Program		SLoC	Inefficiency	% of silent load reduction	Speedup
Benchmark	SPEC CPU2006	lbm	3K	Redundant computation	60%	1.25x
	SPEC CPU2017	imagick_r	274K	Redundant computation	51%	1.25x
	Rodinia-3.1	lavaMD	800	Redundant function calls	93%	1.39x
		srad_v1	600	Inefficient register usage	29%	1.11x
		srad_v2	200	Inefficient register usage	32%	1.12x
		particlefilter	600	Linear search	97%	9.8x
Application	Apache Avro-1.8.2		46K	Missing inline substitution	37%	1.19x
	Hoard-3.12		22K	Redundant computation	2%	1.14x
	MASNUM-2.2		121K	Linear search	36%	1.79x
	USQCD Chroma-3.43		929K	Missing inline substitution	7%	1.06x
	Shogun-6.0		546K	Missing inline substitution	2%	1.06x
	Facebook Stack RNN		2K	Redundant computation	15%	1.09x

Agenda

Profiling

CPU memory wastage

ICSE'19, FSE'19, ICSE'22, CGO'23, TACO'23, CGO'26

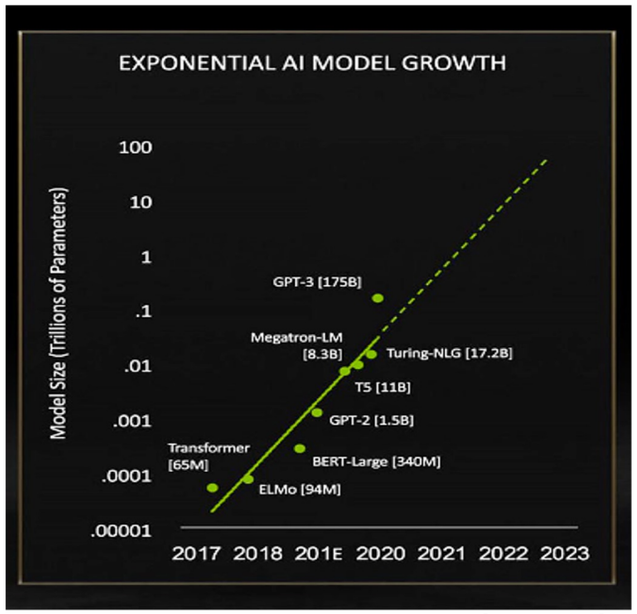
GPU memory wastage

ASPLOS'23, USENIX ATC'24, ASPLOS'26 (under review)

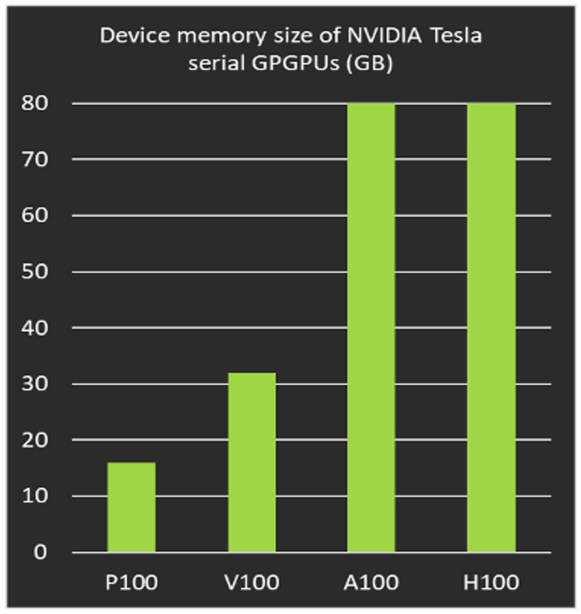
GPU Memory

- Problem 1: Fast-growing memory demand in DL
  - ✦ The trend of developing deeper and wider neural networks

    A trillion-parameter model with 3D parallelism requires 300+ A100 GPUs
- Problem 2: Slow-growing GPU memory budget
  - ✦ HBM must fit in small package space and chip periphery
  - ✦ Compared to increasing memory capacity, reducing memory latency is GPU vendors' top priority



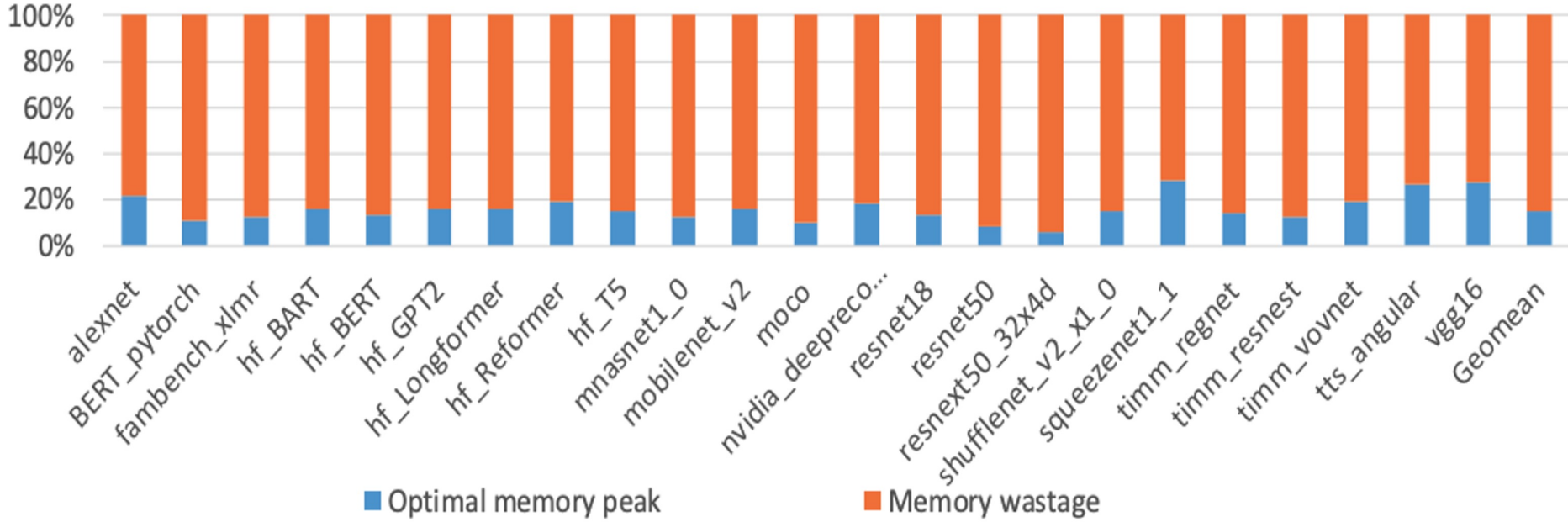
Fast-growing memory demand



Slow-growing GPU memory budget

GPU Memory

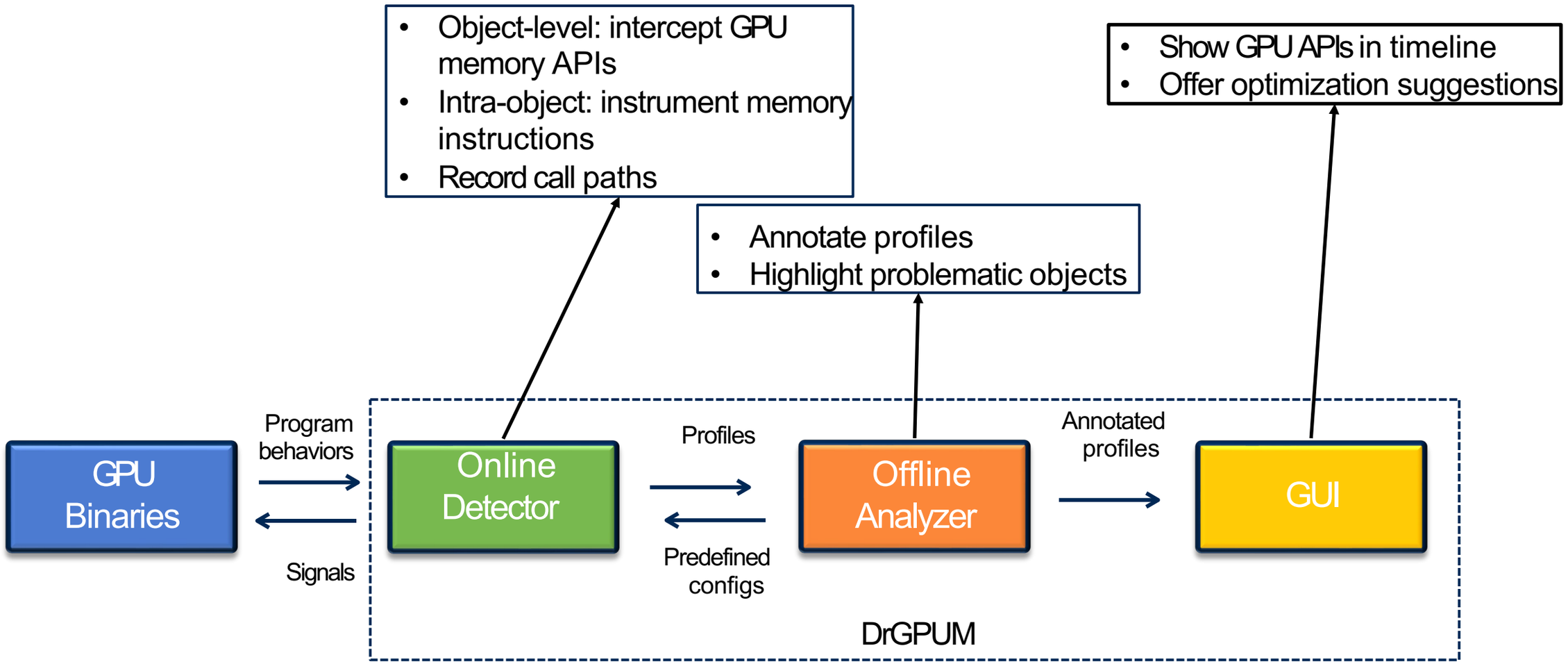
- Problem 3: Memory wastage is significant and pervasive across DL workloads



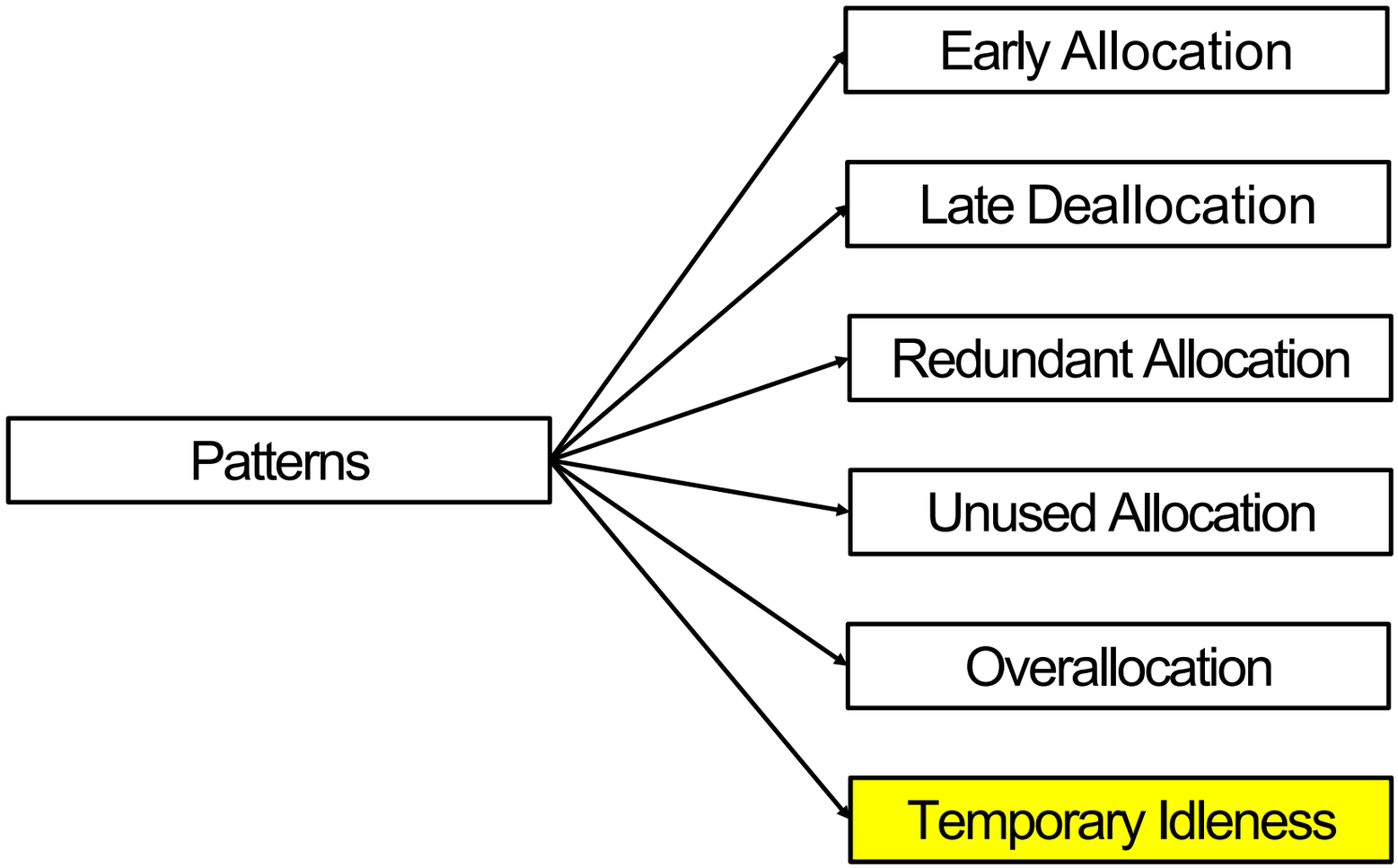
Memory wastage in popular DL models

Our Solution: DrGPUM

- A data-object-centric GPU profiler, including
  - Object-level analysis (e.g., arrays and tensors)
  - Intra-object analysis (e.g., individual elements in an array/tensor)

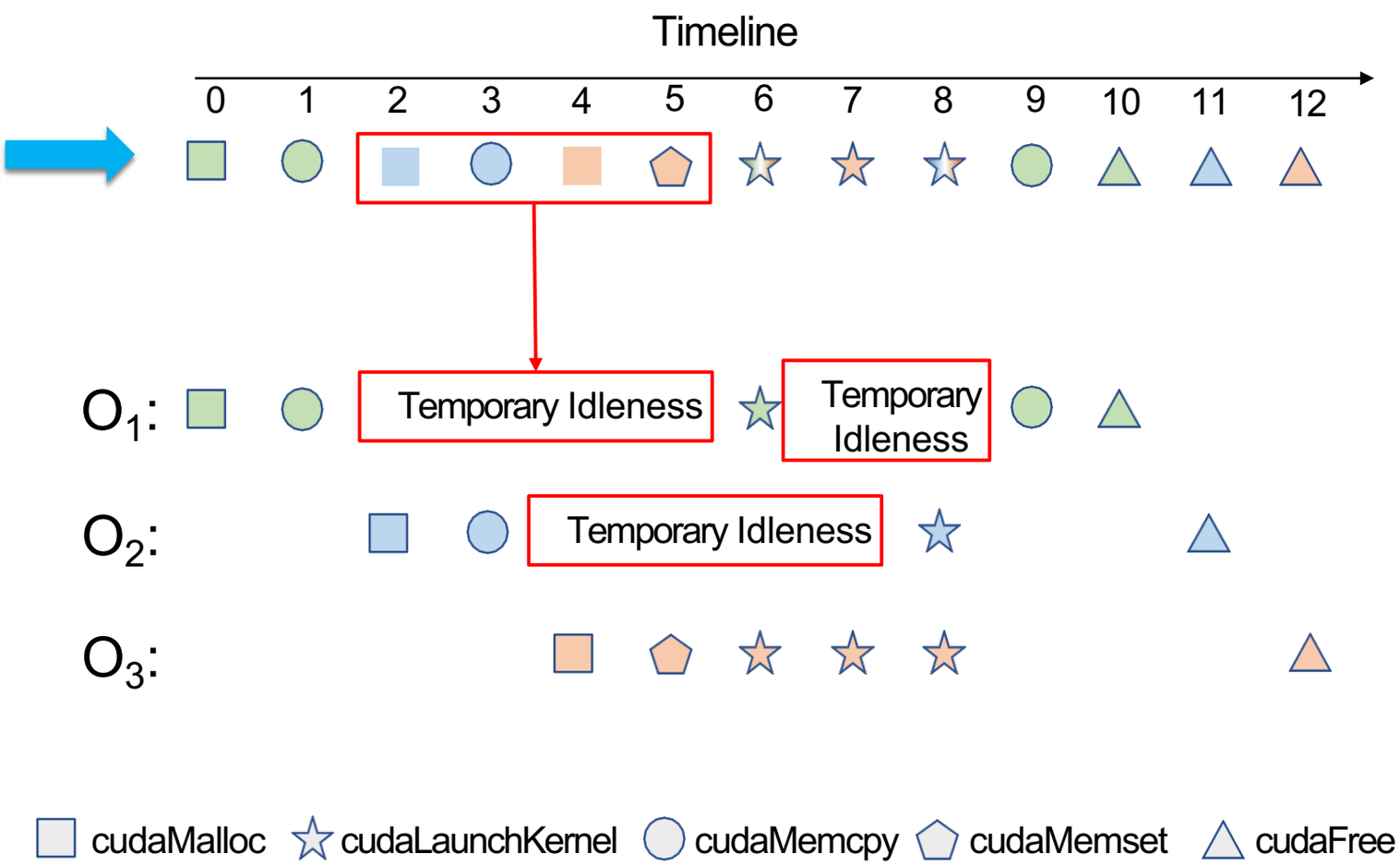


Identified Patterns of Memory Wastage



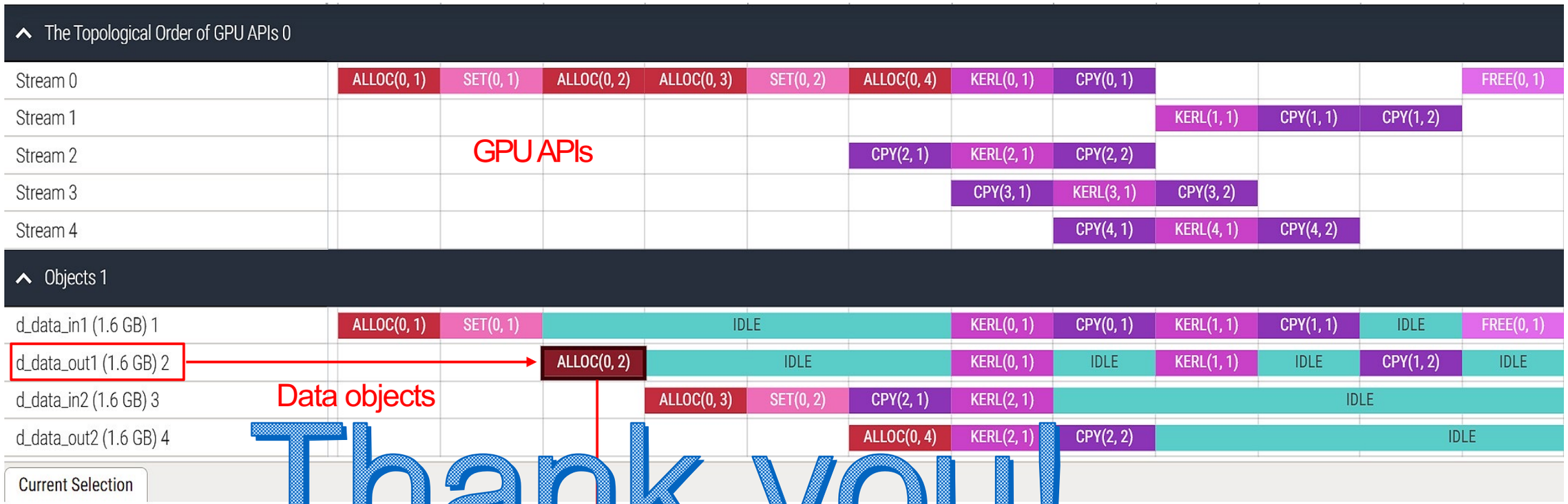
Late Deallocation

```
1. cudaMalloc(void*& O1, size)
2. cudaMemcpy(O1, hostptr, size,
cudaMemcpyDefault)

3. cudaMalloc(void*& O2, size)
4. cudaMemcpy(O2, hostptr, size,
cudaMemcpyDefault)

5. cudaMalloc(void*& O3, size)
6. cudaMemset(O3, 0, size)

7. kernel1<<<block, grid>>>(O1, O3)
8. kernel2<<<block, grid>>>(O3)
9. kernel3<<<block, grid>>>(O2, O3)

10.cudaMemcpy(hostptr, O3, size,
cudaMemcpyDefault)

11. cudaFree(O1)
12. cudaFree(O2)
13. cudaFree(O3)
```



Demo

- SimplyMultiCopy
  - A multi-stream program released with NVIDIA CUDA Toolkit



Thank you!

ALLOC(i, j): j^{th} cudaMalloc on Stream i
SET(i, j): j^{th} cudaMemset on Stream i
CPY(i, j): j^{th} cudaMemcpy on Stream i
KERL(i, j): j^{th} kernel launch on Stream i
FREE(i, j): j^{th} cudaFree on Stream i