OS and Compiler Issues in Disparate Memory Allocation

Presented at NVRAMOS 2010 Spring Workshop

April 20, 2010

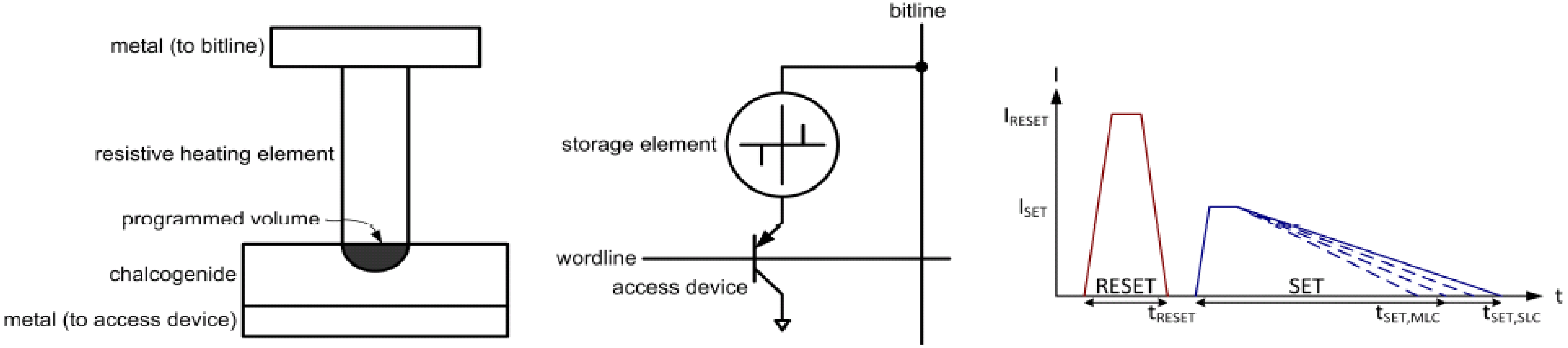
Joo-Young Hwang
Advanced S/W Research Team
Memory Division, Samsung Electronics Co., Ltd.

Agenda

- PRAM Introduction
- DRAM/PRAM Hybrid Memory Architecture
- 정적 할당 메모리 배치
  - Compilation
  - Allocation
  - Linux Kernel Page Binding
  - Results
- 동적 할당 메모리 배치
  - Application write pattern analysis
  - Calling Context Based Prediction
- Summary
- References

PRAM Introduction

- Memory Cell
  - Phase change material (typically GST) is set (crystalline) or reset (amorphous) by resistive heating mechanism.
- Advantages
  - Scalability: high density is possible.
  - Comparable read speed to DRAM
- Challenges
  - Slow write speed compared to DRAM
  - Limited endurance compared to DRAM



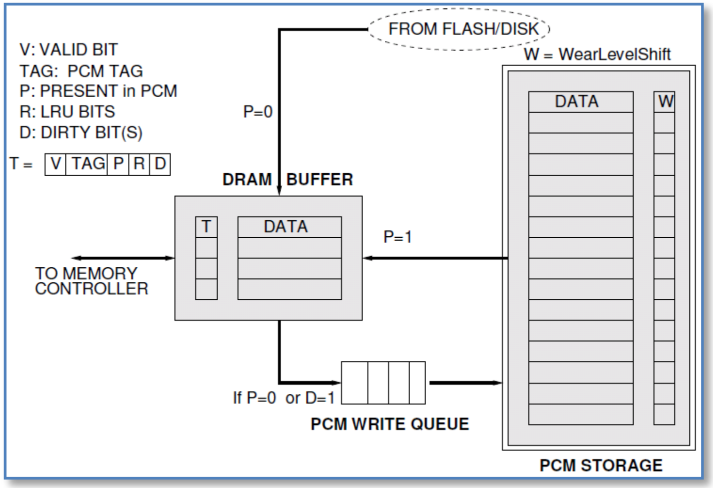
* Reference: [1] Architecting Phase Change Memory as a Scalable DRAM Alternative, Benjamin C. Lee, Engin Ipek (Microsoft Research), Onur Mutlu (Carnegie Mellon University), Doug Burger (Microsoft Research), International Symposium on Computer Architecture (ISCA) ‘09

DRAM/PRAM Hybrid Memory Architecture

- DRAM/PRAM hybrid memory 구성을 통한 high density, energy efficient main memory subsystem 구조 연구 [3,4]
- PRAM의 slow write speed로 인한 실행 성능 저하를 막기 위해서는 적절한 메모리 배치 방법이 필요:
  - DRAM – PRAM 간 page migration traffic을 줄이는 것이 중요
  - 코드, 정적 할당 메모리, 동적 할당 메모리

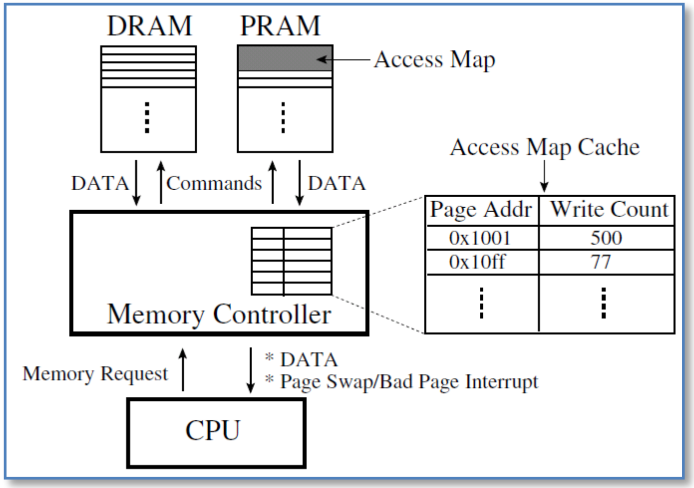
DRAM as buffer for PRAM [3]

Presented at ISCA '09



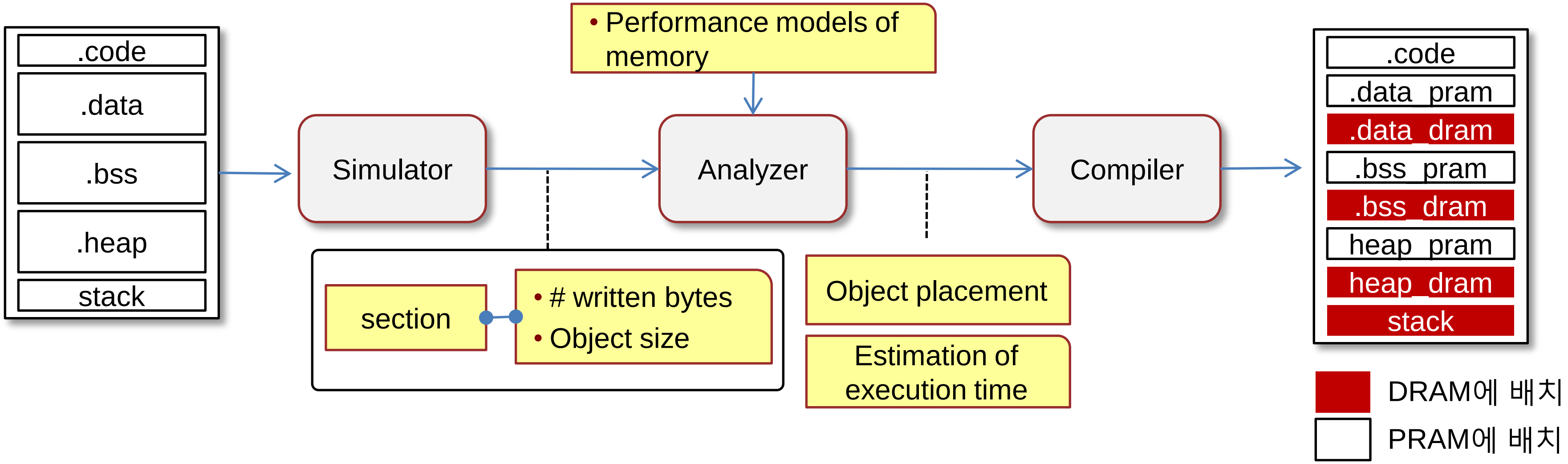
Side-by-side memories [4]

Presented at DAC '09



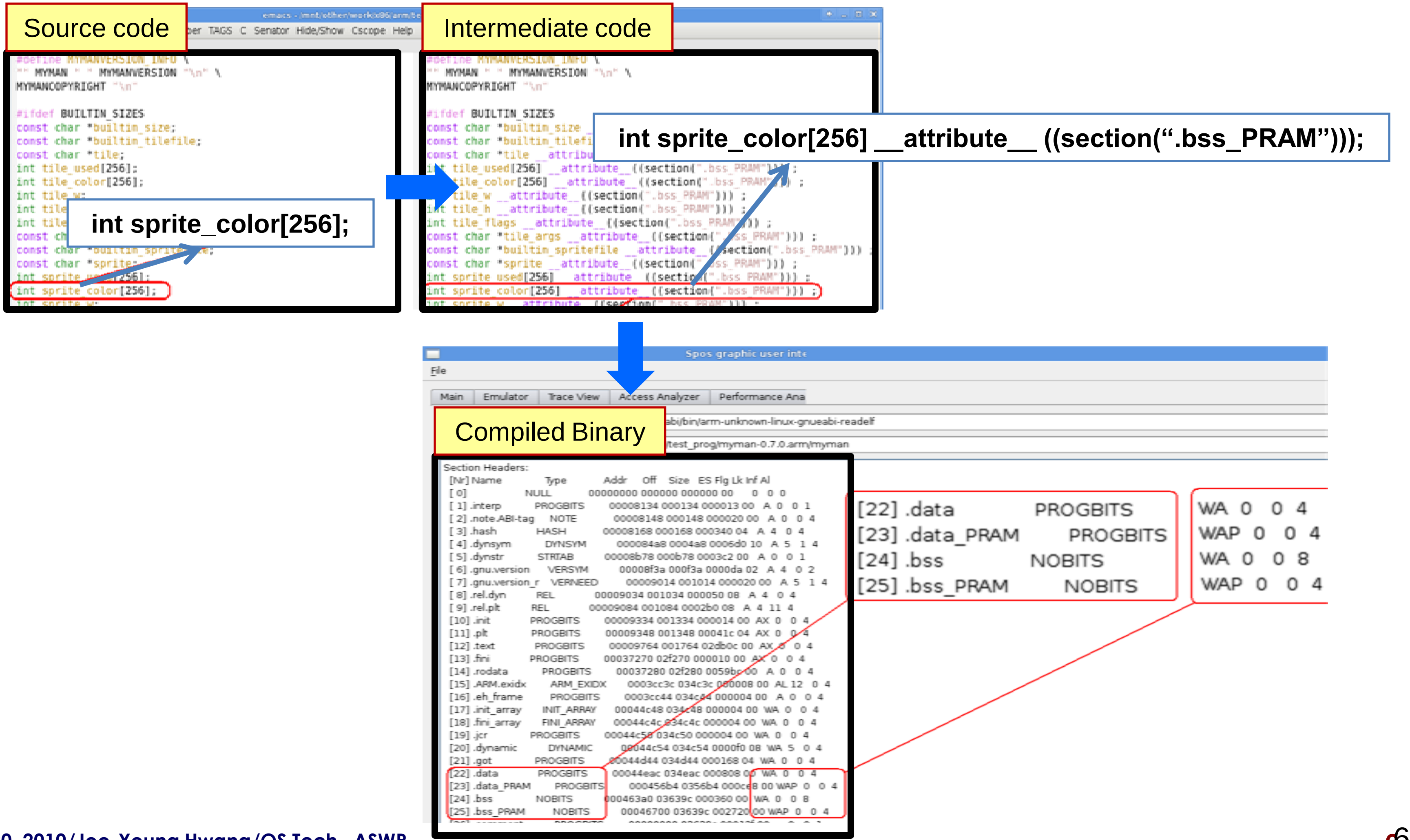
정적 할당 메모리 배치

- Simulation Environment
  - Simulation tool – QEMU
  - S/W: Linux/gcc, glibc
- Assumption
  - System simulated - SMDK6400, ARM1176JZF-S, I/D-cache 16KB, 256MB DRAM, 256MB PRAM
  - PRAM read latency는 DRAM과 동일, PRAM write latency는 DRAM x4
- Constraints
  - DRAM only 구성 vs. DRAM/PRAM hybrid 구성의 total memory 크기는 동일
  - DRAM only 구성에서의 execution time 보다 10% 이상 늘어나지 않도록 제한



Compilation

- Compiler directives를 사용하여 memory allocation hint를 명시함.



Source code

```
#define MYMANVERSION_INFO \
"" MYMAN " " MYMANVERSION "\n" \
MYMANCOPYRIGHT "\n"

#ifdef BUILTIN_SIZES
const char *builtin_size;
const char *builtin_tilefile;
const char *tile;
int tile_used[256];
int tile_color[256];
int tile_w;
int tile
int tile
const ch
const char *builtin_sprite e;
const char *sprite;
int sprite_used[256];
int sprite_color[256];
int sprite_w;
```

int sprite_color[256];

Intermediate code

```
#define MYMANVERSION_INFO \
"" MYMAN " " MYMANVERSION "\n" \
MYMANCOPYRIGHT "\n"

#ifdef BUILTIN_SIZES
const char *builtin_size
const char *builtin_tilefi
const char *tile __attribu
int tile_used[256] __attribute__((section(".bss_PRAM"))) ;
int tile_color[256] __attribute__((section(".bss_PRAM"))) ;
int tile_w __attribute__((section(".bss_PRAM"))) ;
int tile_h __attribute__((section(".bss_PRAM"))) ;
int tile_flags __attribute__((section(".bss_PRAM"))) ;
const char *tile_args __attribute__((section(".bss_PRAM"))) ;
const char *builtin_spritefile __attribute__((section(".bss_PRAM"))) ;
const char *sprite __attribute__((section(".bss_PRAM"))) ;
int sprite_used[256] __attribute__((section(".bss_PRAM"))) ;
int sprite_color[256] __attribute__((section(".bss_PRAM"))) ;
int sprite_w __attribute__((section(".bss_PRAM"))) ;
```

int sprite_color[256] __attribute__ ((section(“.bss_PRAM”)));

Compiled Binary

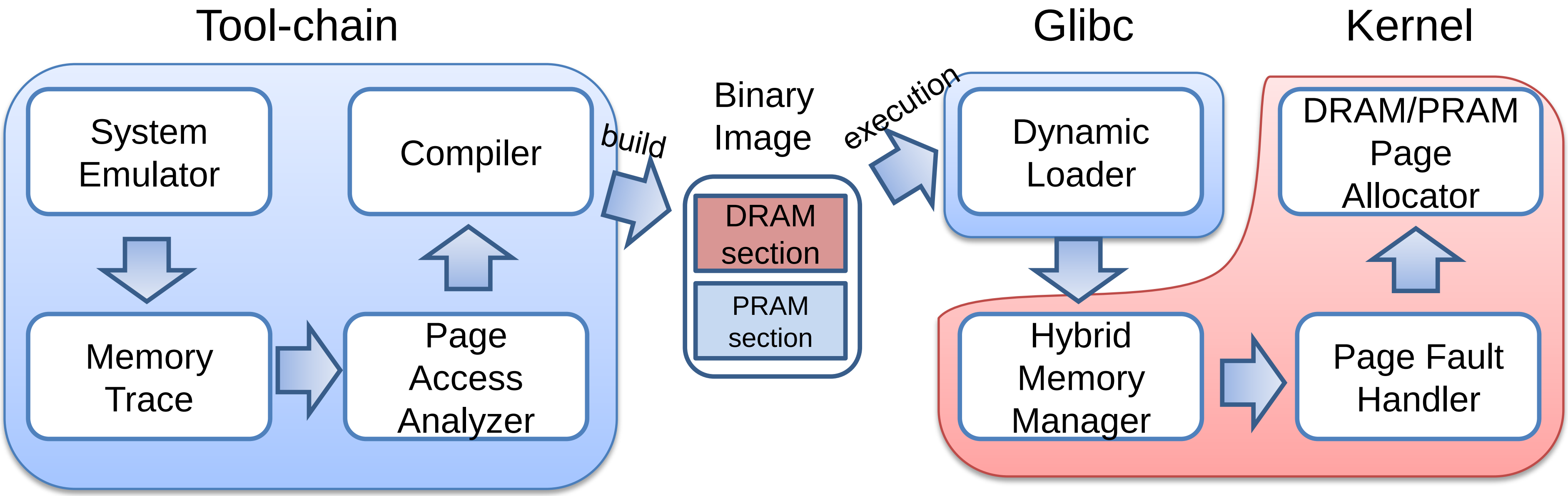
```
abi/bin/arm-unknown-linux-gnueabi-readelf
/test_prog/myman-0.7.0.arm/myman

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        00008134 000134 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            00008148 000148 000020 00   A  0   0  4
  [ 3] .hash             HASH            00008168 000168 000340 04   A  4   0  4
  [ 4] .dynsym           DYNSYM          000084a8 0004a8 0006d0 10   A  5   1  4
  [ 5] .dynstr           STRTAB          00008b78 000b78 0003c2 00   A  0   0  1
  [ 6] .gnu.version      VERSYM          00008f3a 000f3a 0000da 02   A  4   0  2
  [ 7] .gnu.version_r    VERNEED         00009014 001014 000020 00   A  5   1  4
  [ 8] .rel.dyn          REL             00009034 001034 000050 08   A  4   0  4
  [ 9] .rel.plt          REL             00009084 001084 0002b0 08   A  4  11  4
  [10] .init             PROGBITS        00009334 001334 000014 00  AX  0   0  4
  [11] .plt              PROGBITS        00009348 001348 00041c 04  AX  0   0  4
  [12] .text             PROGBITS        00009764 001764 02db0c 00  AX  0   0  4
  [13] .fini             PROGBITS        00037270 02f270 000010 00  AX  0   0  4
  [14] .rodata           PROGBITS        00037280 02f280 0059bc 00   A  0   0  4
  [15] .ARM.exidx        ARM_EXIDX       0003cc3c 034c3c 000008 00  AL 12   0  4
  [16] .eh_frame         PROGBITS        0003cc44 034c44 000004 00   A  0   0  4
  [17] .init_array       INIT_ARRAY      00044c48 034c48 000004 00  WA  0   0  4
  [18] .fini_array       FINI_ARRAY      00044c4c 034c4c 000004 00  WA  0   0  4
  [19] .jcr              PROGBITS        00044c50 034c50 000004 00  WA  0   0  4
  [20] .dynamic          DYNAMIC         00044c54 034c54 0000f0 08  WA  5   0  4
  [21] .got              PROGBITS        00044d44 034d44 000168 04  WA  0   0  4
  [22] .data             PROGBITS        00044eac 034eac 000808 00  WA  0   0  4
  [23] .data_PRAM        PROGBITS        000456b4 0356b4 000ce8 00 WAP  0   0  4
  [24] .bss              NOBITS          000463a0 03639c 000360 00  WA  0   0  8
  [25] .bss_PRAM         NOBITS          00046700 03639c 002720 00 WAP  0   0  4
```

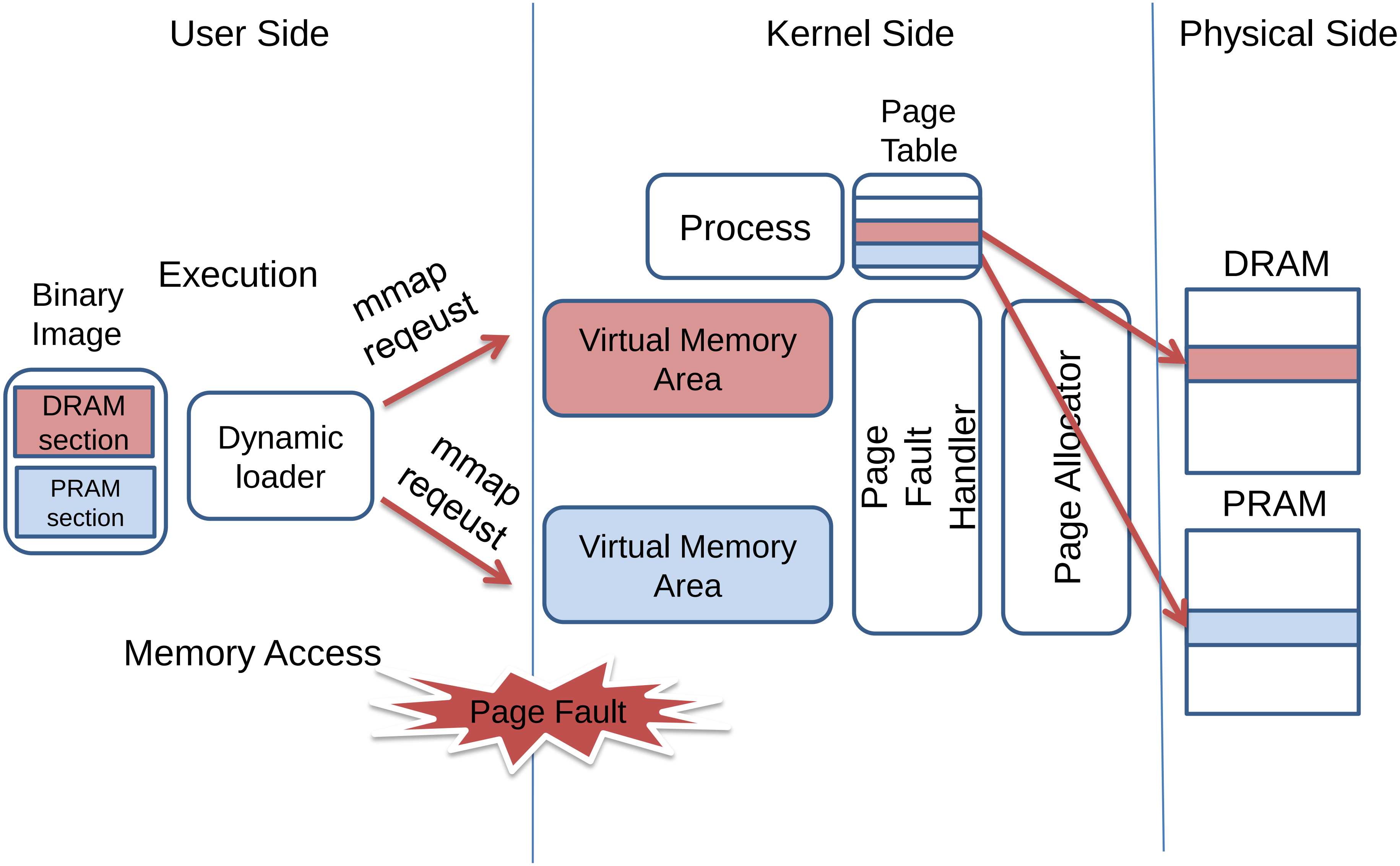
```
[22] .data          PROGBITS        WA   0  0  4
[23] .data_PRAM     PROGBITS        WAP  0  0  4
[24] .bss           NOBITS          WA   0  0  8
[25] .bss_PRAM      NOBITS          WAP  0  0  4
```

Allocation

- SPOS Toolchain을 통해 Application의 메모리 접근 패턴 분석
- Hybrid Memory 상의 배치 정보를 포함한 Binary Image Layout 생성
- Dynamic Loader를 통해 DRAM/PRAM 메모리 매핑
- Kernel Page Allocator를 통한 물리 메모리 할당



Linux Kernel Page Binding

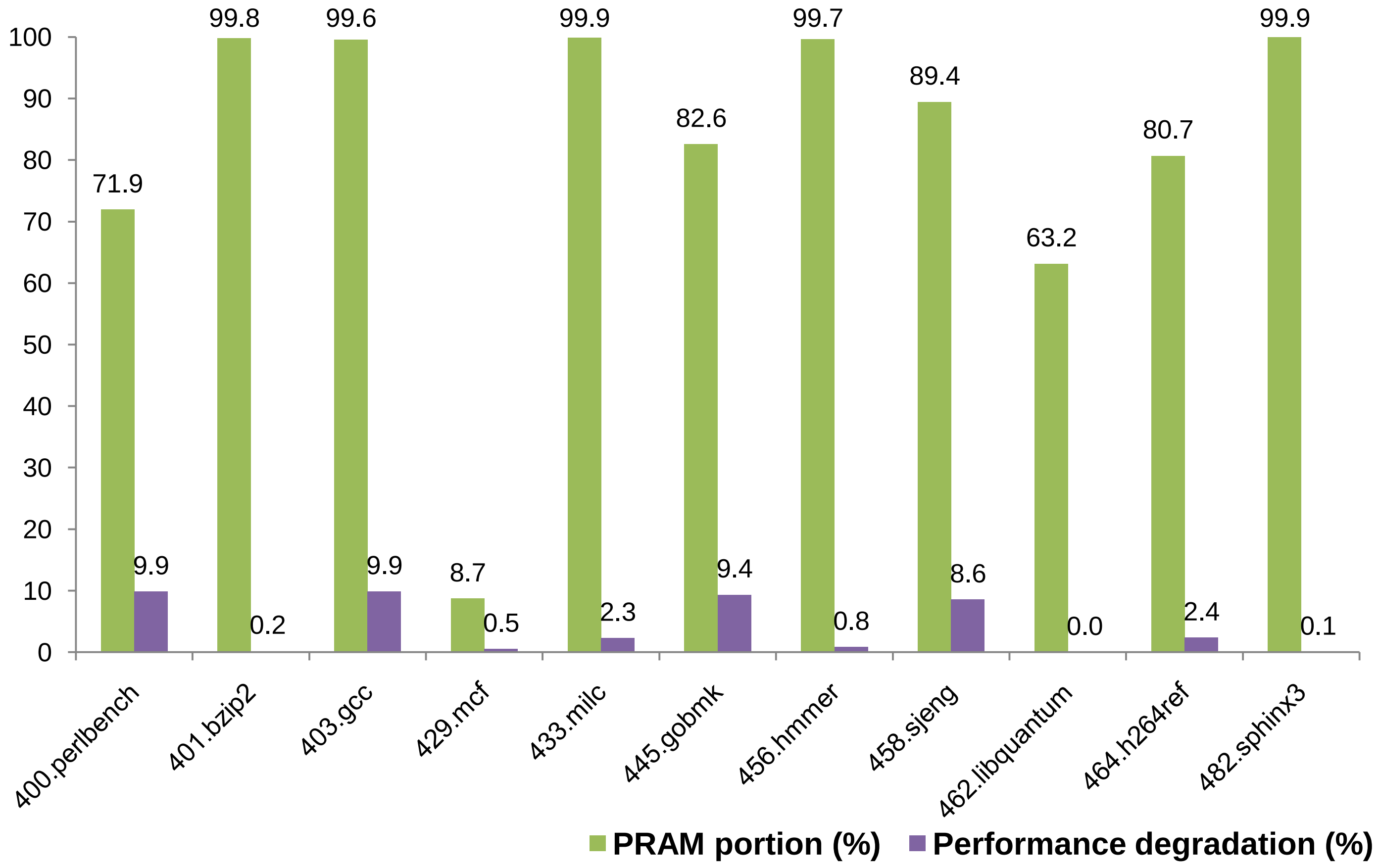


Results

- **정적 할당 메모리 영역 배치 결과**

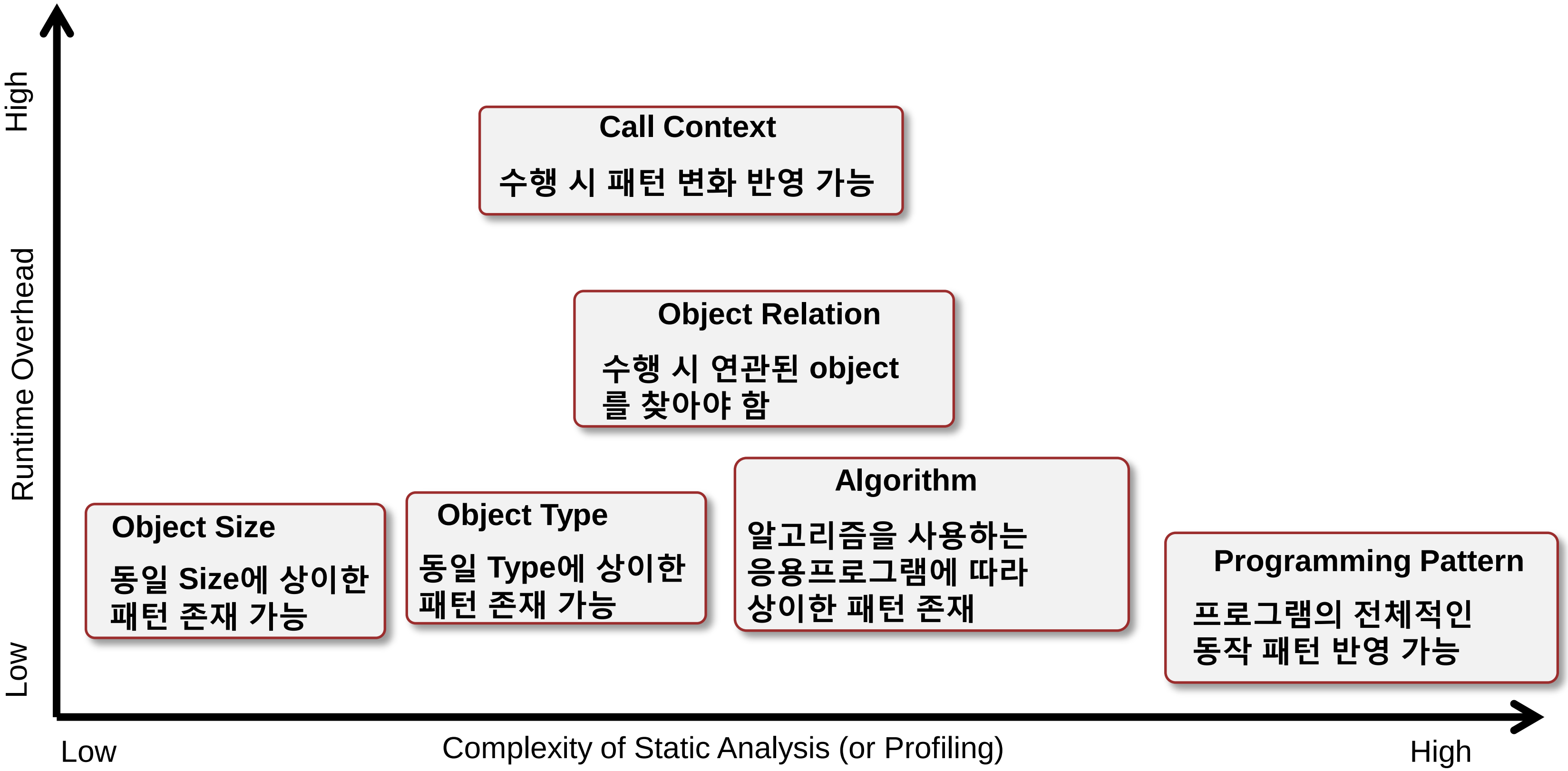
  : PRAM상에 배치하여 직접 access 가능한 메모리 – 63.2~99.9% (평균 81.4%)

[SPEC2006 분석 결과]



동적 할당 메모리 배치

- **동적 할당 메모리 access pattern profiling의 어려움**
  - **Object identification**
  - **Very large memory trace processing**
- **Write pattern prediction 방법 선정**
  - 정확성, runtime overhead, 구현 용이성 등을 종합 고려 필요



Call Context

수행 시 패턴 변화 반영 가능

Object Relation

수행 시 연관된 object 를 찾아야 함

Object Size

동일 Size에 상이한 패턴 존재 가능

Object Type

동일 Type에 상이한 패턴 존재 가능

Algorithm

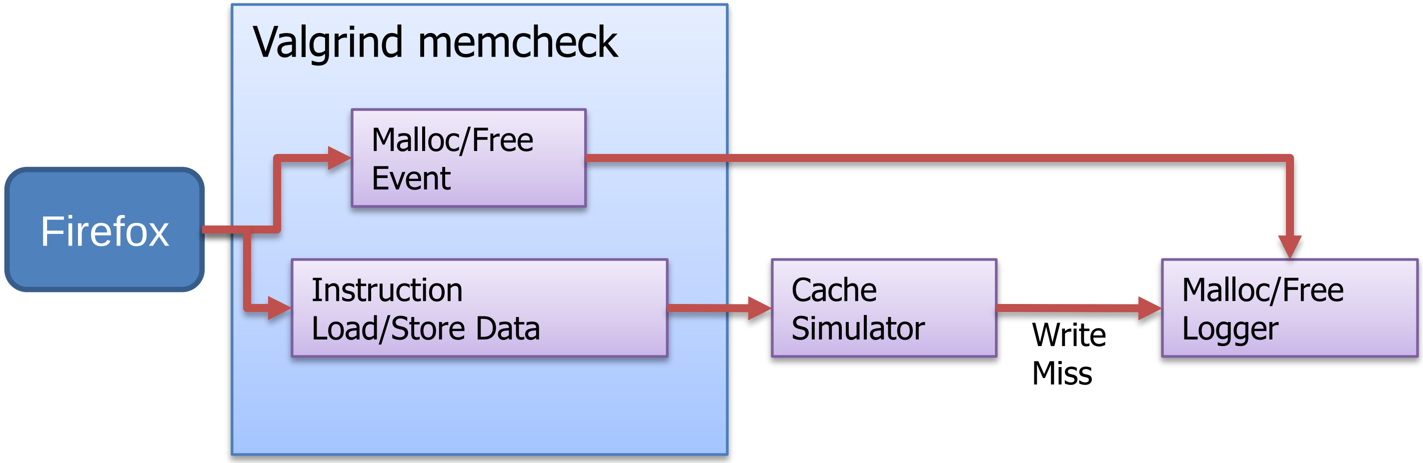
알고리즘을 사용하는 응용프로그램에 따라 상이한 패턴 존재

Programming Pattern

프로그램의 전체적인 동작 패턴 반영 가능

Application Write Pattern Analysis

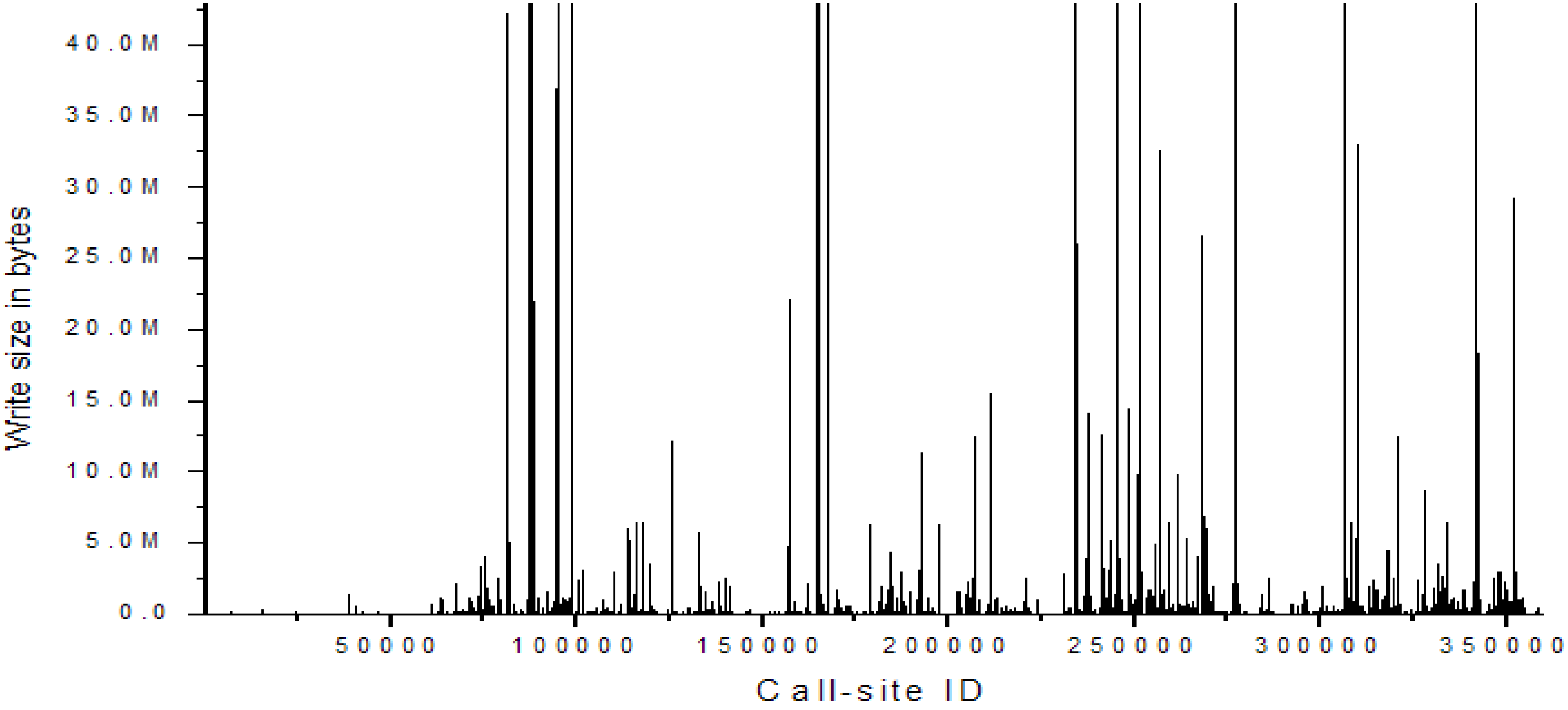
- Valgrind memcheck (for x86) module based write-pattern analysis
  - Disable dangling pointer checks (--freelist-vol=0)
    - Prevent unnecessary heap increasing
  - Kernel memory references are ignored.
- Application to analyze: mozilla firefox
  - Load three web-pages concurrently: www.google.co.kr, www.youtube.com, www.valgrind.org



Calling Context Based Prediction

- 대부분의 call-sites가 significant write traffic을 발생시킴
- write-most objects를 분리하기 위해서 call-site 정보 외에 다른 정보도 함께 사용할 필요가 있음

[Write-traffic after L2-cache]



Summary

- **Hybrid memory 상에 정적 할당 메모리 배치 방법 Simulation**

  : 정적 할당 data 중 PRAM에 할당 가능한 비중: 평균 약 81.4%

- **동적 할당 메모리 배치 연구 이슈**

  : Object의 write pattern을 정확히 파악하는 것이 중요

  : Access pattern analysis - profiling vs. run-time monitoring

  : Impact of CPU cache

  – Cache hierarchy and structure, read write policy, replacement policy

  : Architectural support

- **Hybrid Memory 관리를 위한 OS/Compiler 연구 필요**

References

[1] Architecting Phase Change Memory as a Scalable DRAM Alternative, Benjamin C. Lee, Engin Ipek (Microsoft Research), Onur Mutlu (Carnegie Mellon University), Doug Burger (Microsoft Research), International Symposium on Computer Architecture (ISCA) ‘09

[2] A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology, Ping Zhou, Bo Zhao, Jun Yang, Youtao Zhang (University of Pittsburgh), ISCA ‘09

[3] Scalable High Performance Main Memory System Using Phase-Change Memory Technology, Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, Jude A. Rivers (IBM T.J. Watson Research Center), ISCA ‘09

[4] PDRAM: A Hybrid PRAM and DRAM Main Memory System, Gaurav Dhiman, Raid Ayoub, Tajana Rosing, Design Automation Conference (DAC) ‘09