The OPELL Partition/Overlay Manager

University of Delaware

CAPSL Group

Contributors

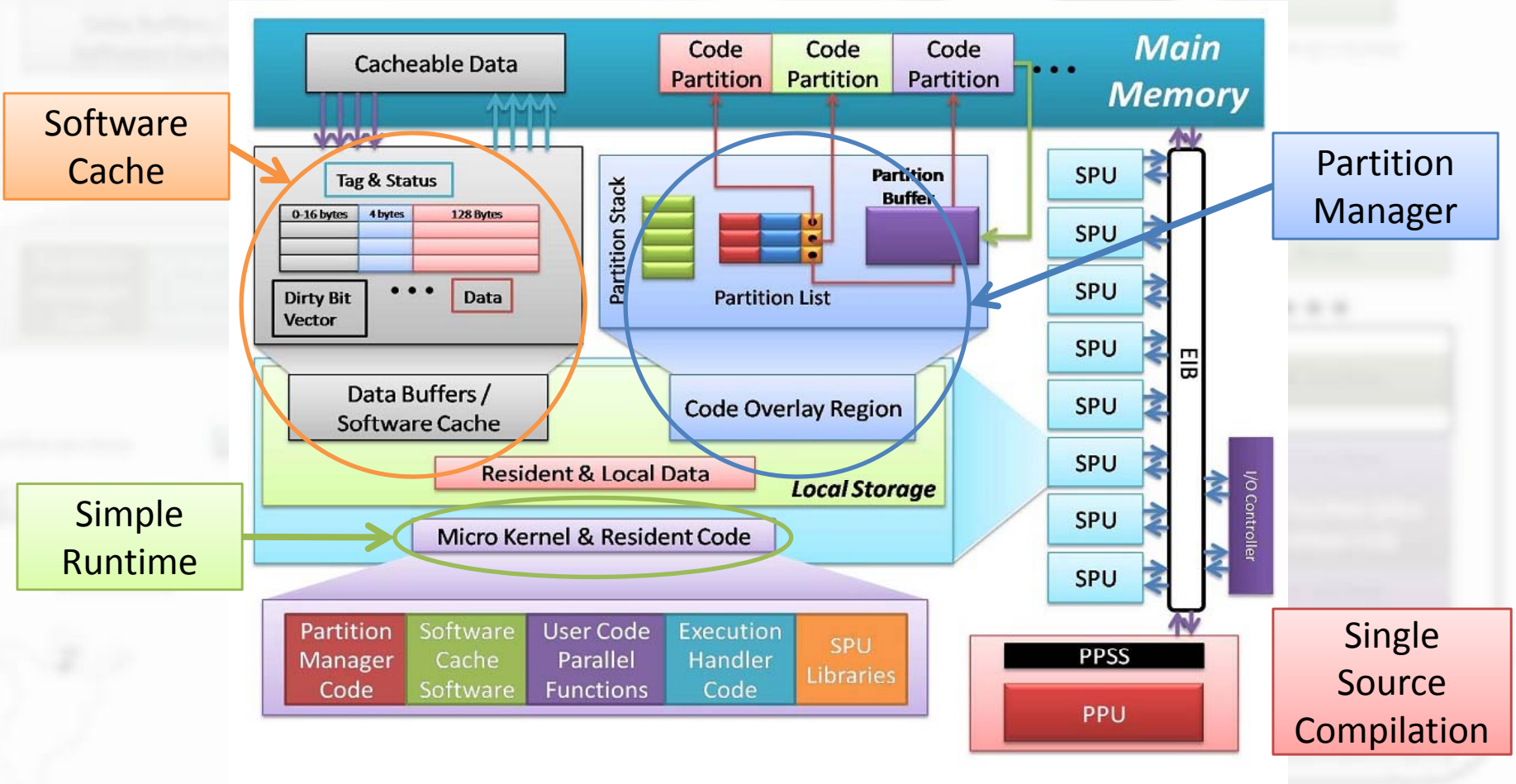
- Main Team
  - Joseph Manzano
  - Yi Jiang
  - Ge Gan
  - Ziang Hu
  - Andrew Russo
  - Guang R Gao
- Special Thanks to
  - EARTH Technologies International
  - The members of the CAPSL Research Group

Outline

- **Objective**
- Partition Manager: the why, the what and how?
- Cache Like Schemes: Modulus approach
- Cache Like Schemes: LRU-like approach
- Testbed, results, conclusions and future work

Objective

- A GNU based OpenMP CBE based



Outline

- Objective
- **Partition Manager: the why, the what and how?**
- Cache Like Schemes: Modulus approach
- Cache Like Schemes: LRU-like approach
- Testbed, results, conclusions and future work

Partition Manager
Why?

- Many-core architectures are on the rise
- Old Problems, new solutions
  - Explicit memory hierarchies
  - Heterogeneous computing environments
  - More dependency on System software than hardware solutions
- System software
  - Extendable and portable solutions applied to Many-core architectures
  - Partition Manager being part of a group of system software for many core architectures

Partition Manager
What?

A Ghost Function

- Do not [explicitely] use the stack
- The call is not visible to the programmer
- It cleans anything that it has done before and after

Function — Loads partitions when code is needed

The Partition — The unit in which the on-demand code will be loaded. It is composed of user functions.

A Typical Call

```
int foo()
{
    bar();
}
```

→ The Partition Manager Runtime Framework →

```
int bar(){
    printf("Hello");
}
```

Implemented in the CELL B.E. Toolchain v 1.1

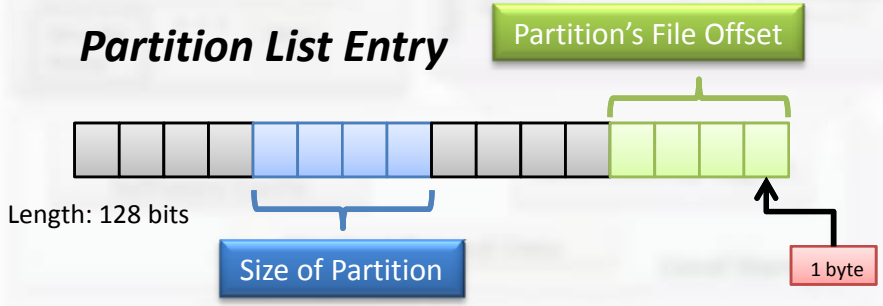
Partition Manager
How?

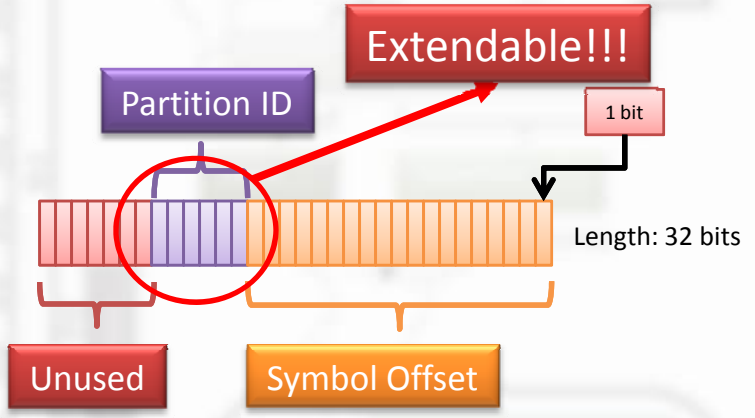
Compiler Options: *-fcode-partition* and *–fcode-partition-manu*

Language Pragmas: *#pragma partitions* and *#pragma keep_function*

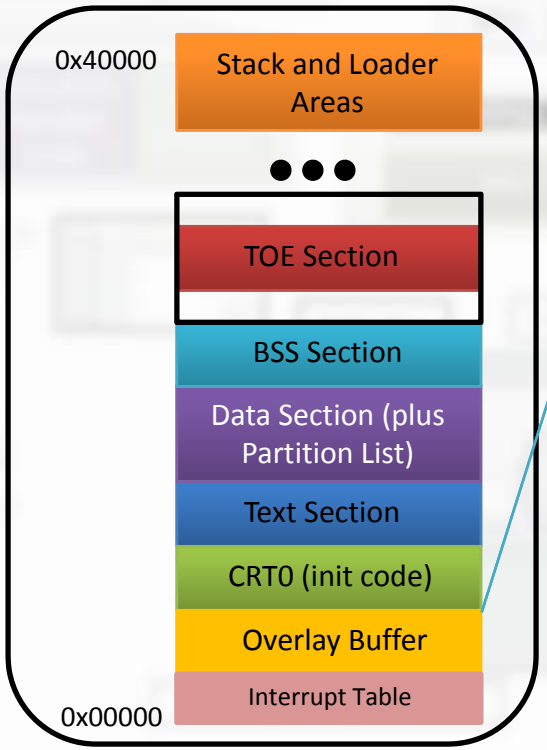
[SPU] Compiler	Create Partitions based on size and number of (static) calls. Insert New Assembly Directives	.partition .funcpointer .libcallee .pmcall
[SPU] Assembler	Generate code according to the new directives	Calls to Partition Manager for .pmcall and .funcpointer calls
[SPU] Linker	Create the partitions, helper structures and combine them into their respective sections Embed partition information in function offsets	Partition List 6-bit partition id on each function offset Insertion of linker script parameters (buffer size and location)

Partition Manager
How?



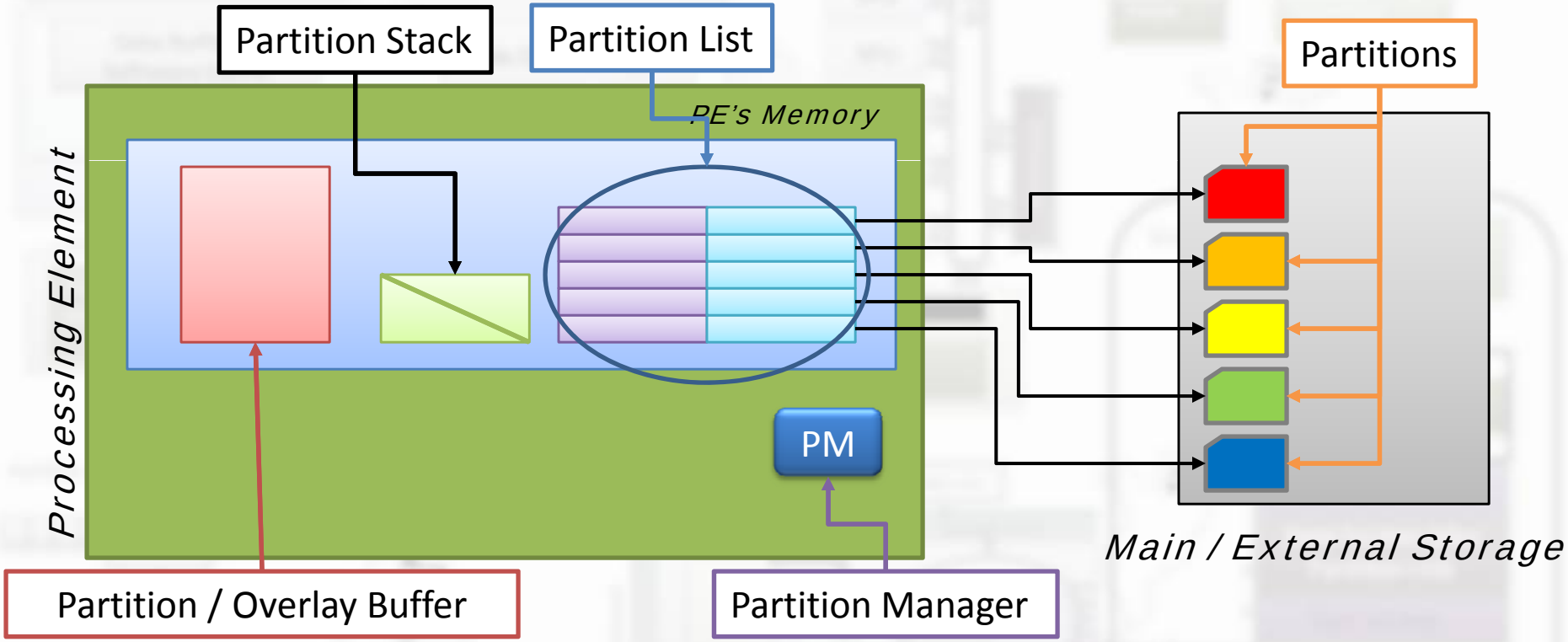


The Format of a Symbol

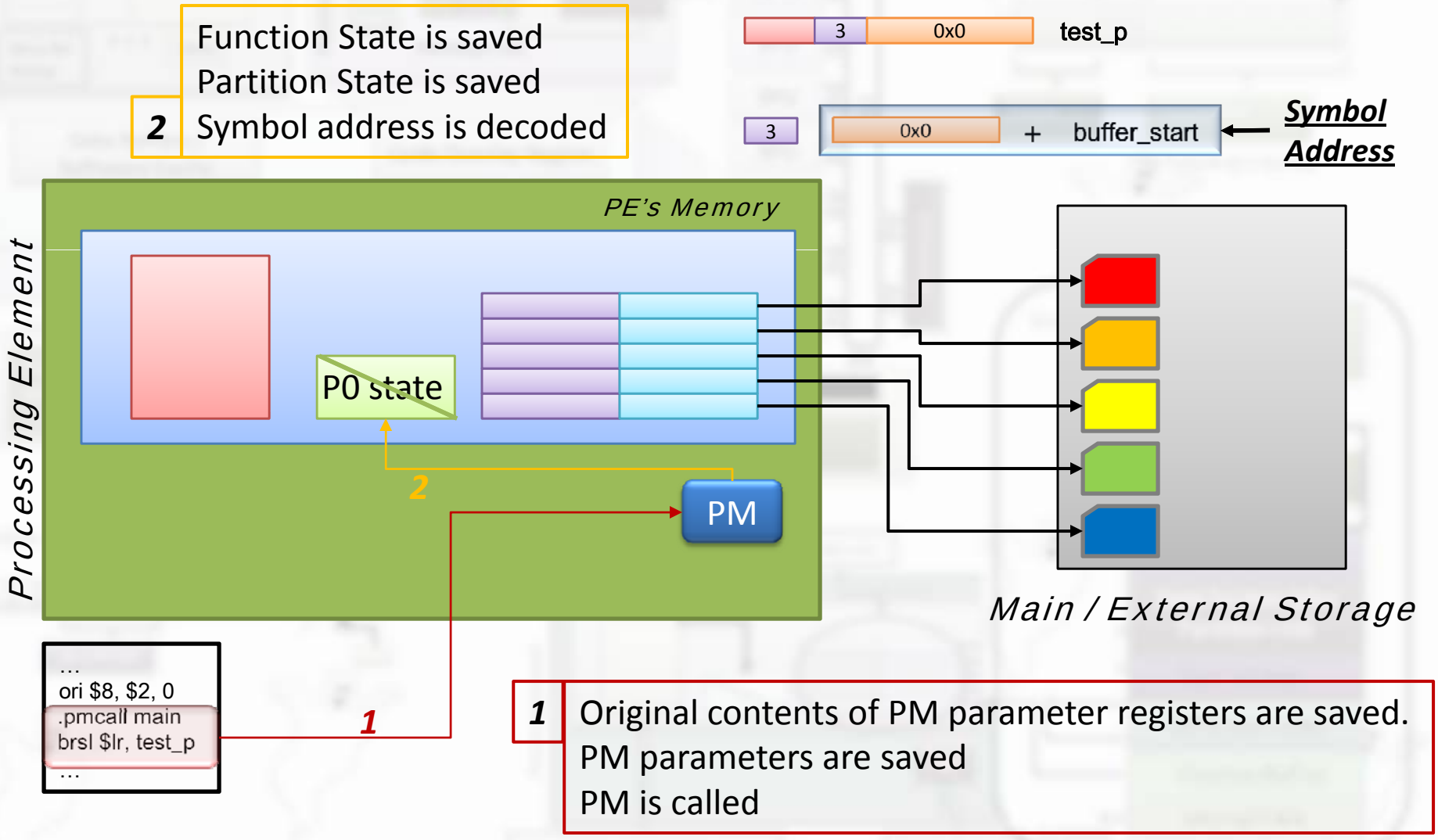


The ELF Image of Resulting Binary

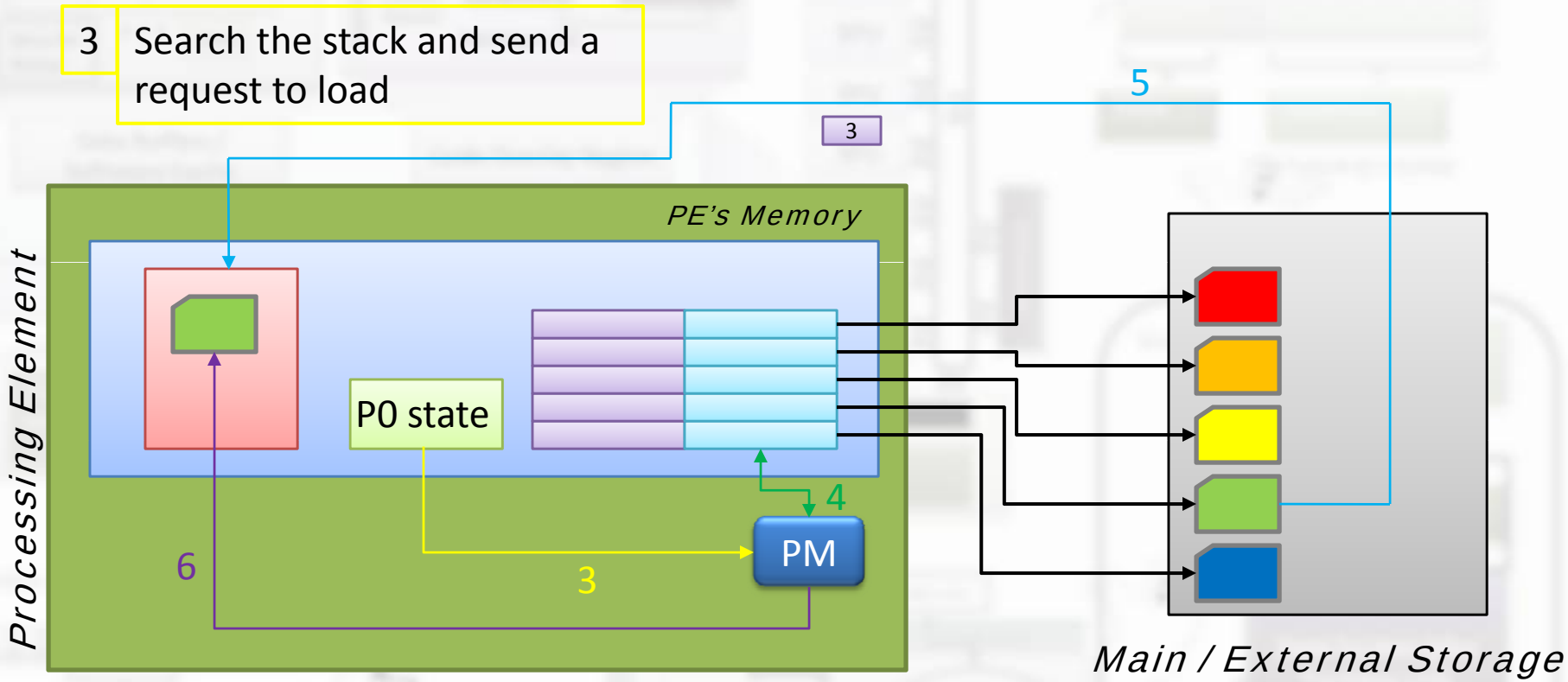
Partition Manager
How?



Partition Manager
How?



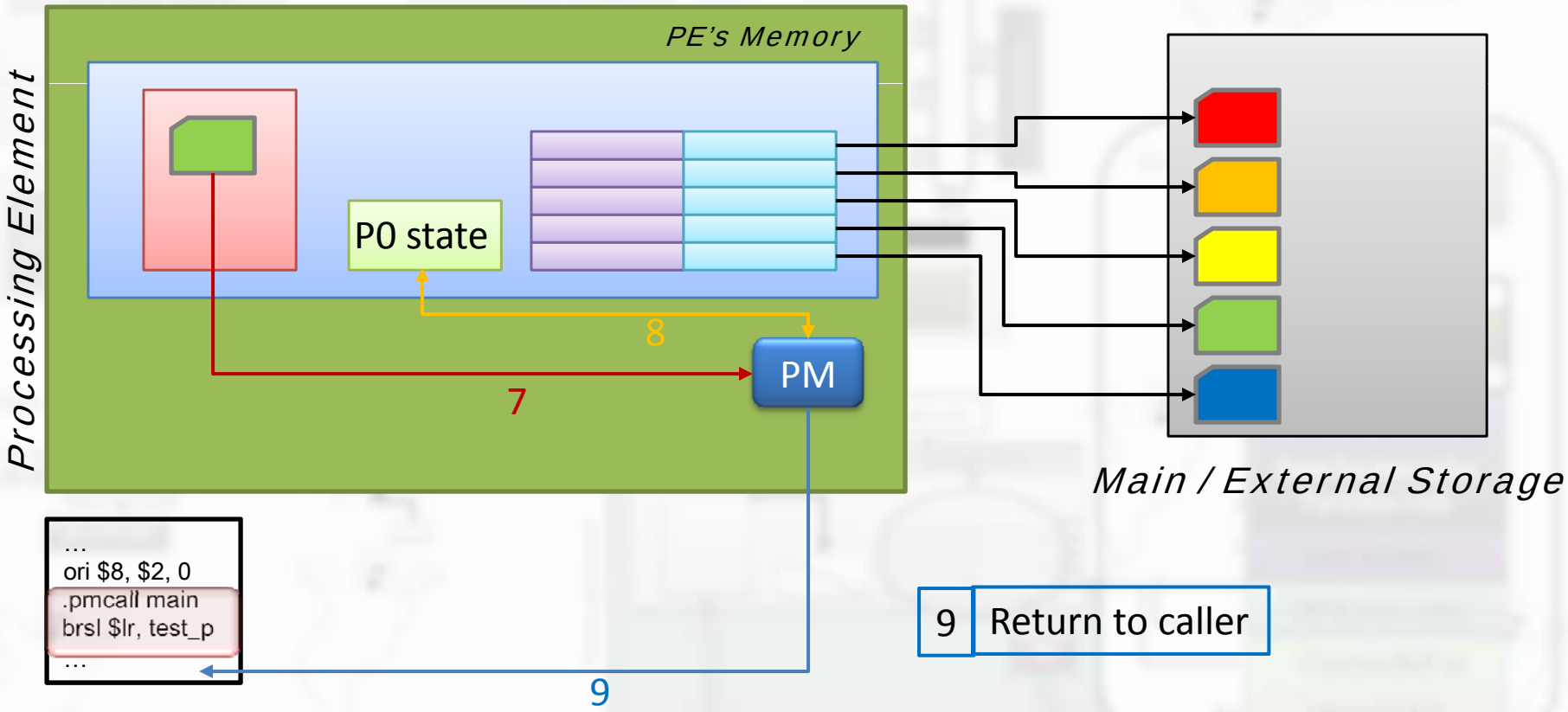
Partition Manager
How?



Partition Manager
How?

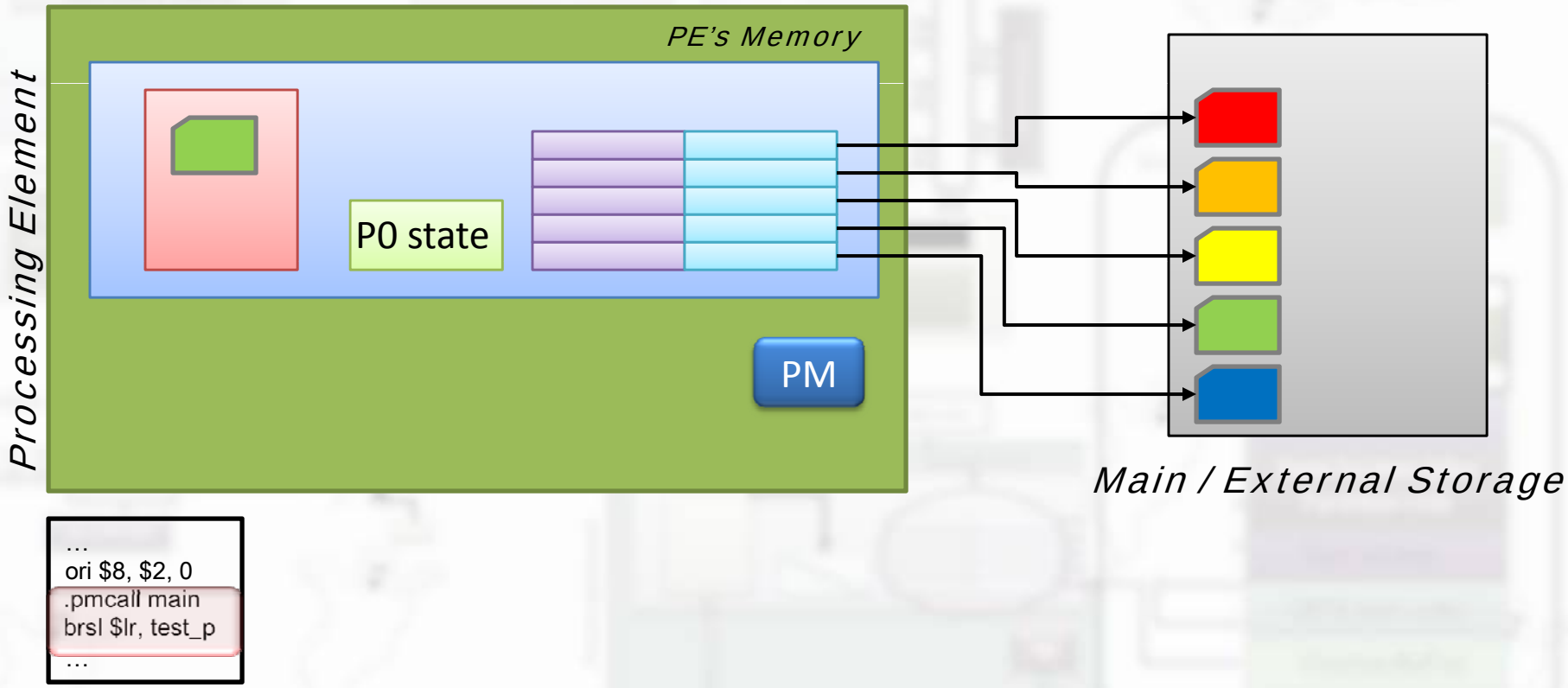
7 Return from function
Save returning function state

8 Restore partition state
Load partition if needed
Restore function state

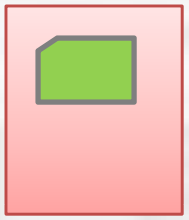


9 Return to caller

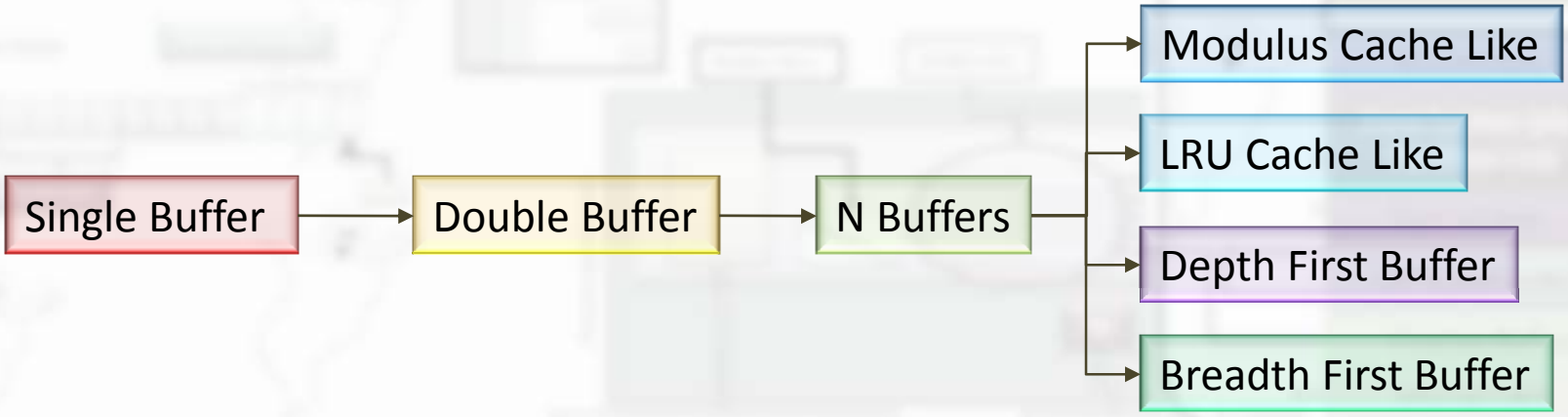
Partition Manager
How?



Partition Manager
How?



How to manage this component during runtime so that the number of DMA transfers is reduced?



Outline

- Objective
- Partition Manager: the why, the what and how?
- **Cache Like Schemes: Modulus approach**
- Cache Like Schemes: LRU-like approach
- Testbed, results, conclusions and future work

Cache-Like Schemes

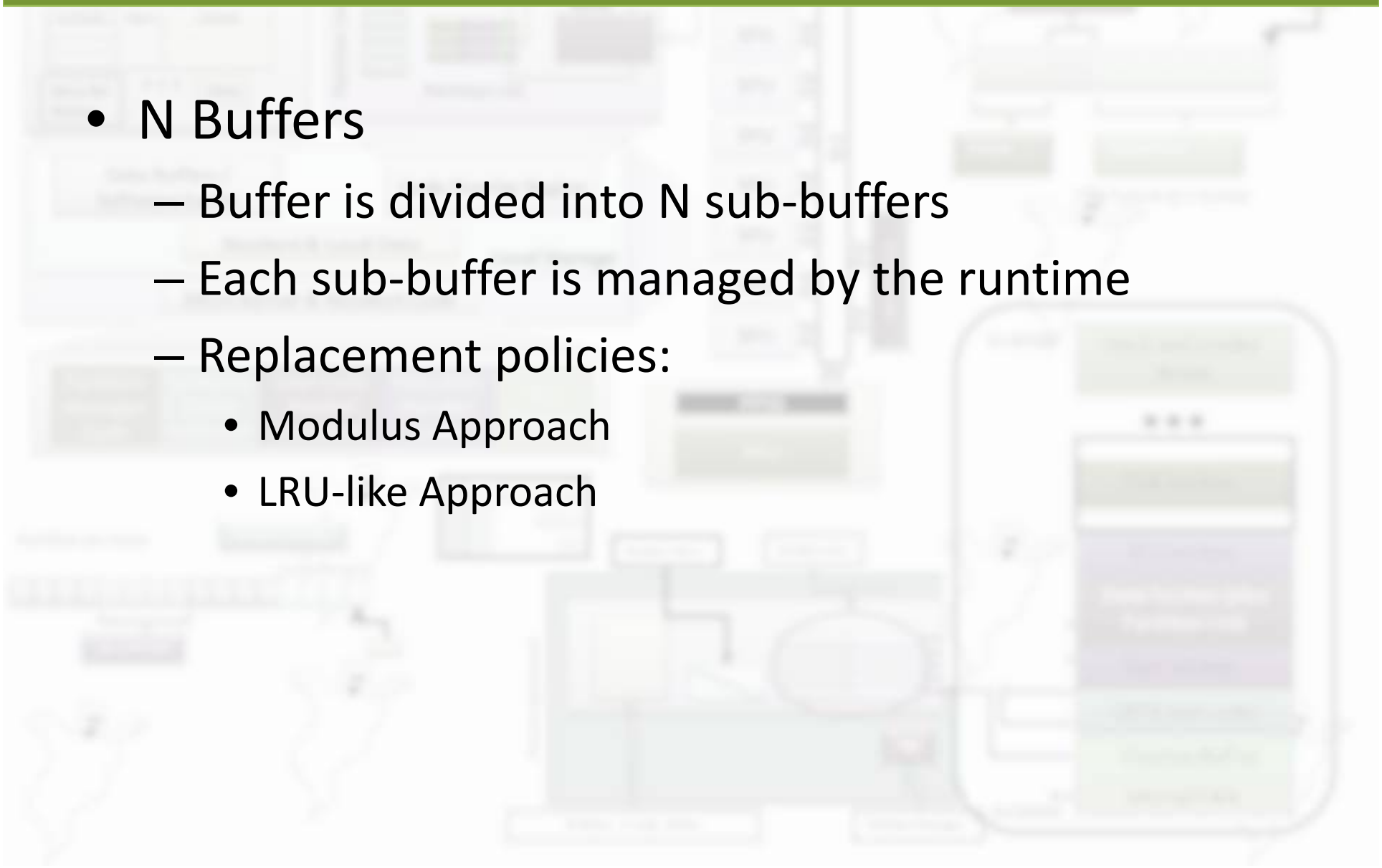
- Single Buffer
  - Trivial Case
  - One Buffer is equal to one active partition
  - Replacement Policy: Trivial ➔ always replace
- Double Buffer
  - Double buffer is a restricted case of N Buffers
  - Two buffers
  - Replacement Policy: Modular

Side Note
Dynamic Partition Types

Partition Type	Description
Active	Partition in which code is being executed
Inactive	Partition in which code was or will be executed and resides in one of the sub-buffers
Evicted	Partition which may or may not execute and it is not loaded yet. All partitions are of this type at the beginning
EWOR	***E**victed **W**ith **O**pportunity of **R**euse:* Partition which was executed and it will be required in the future but was replaced by another partition.

Cache-Like Schemes

- N Buffers
  - Buffer is divided into N sub-buffers
  - Each sub-buffer is managed by the runtime
  - Replacement policies:
    - Modulus Approach
    - LRU-like Approach

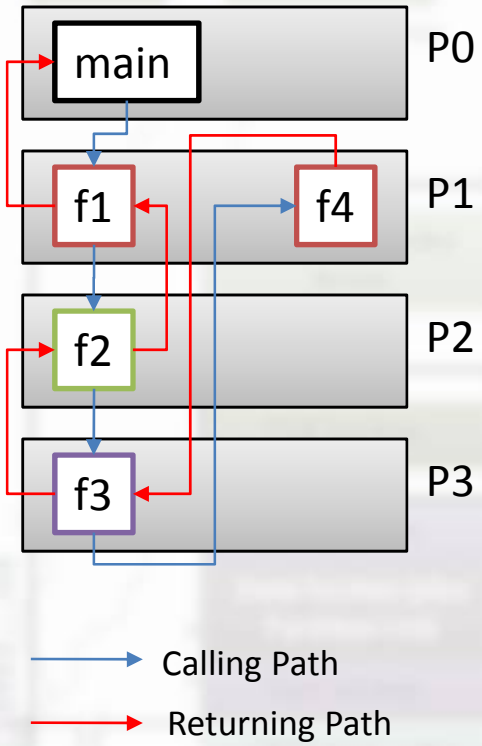


Cache Like Schemes

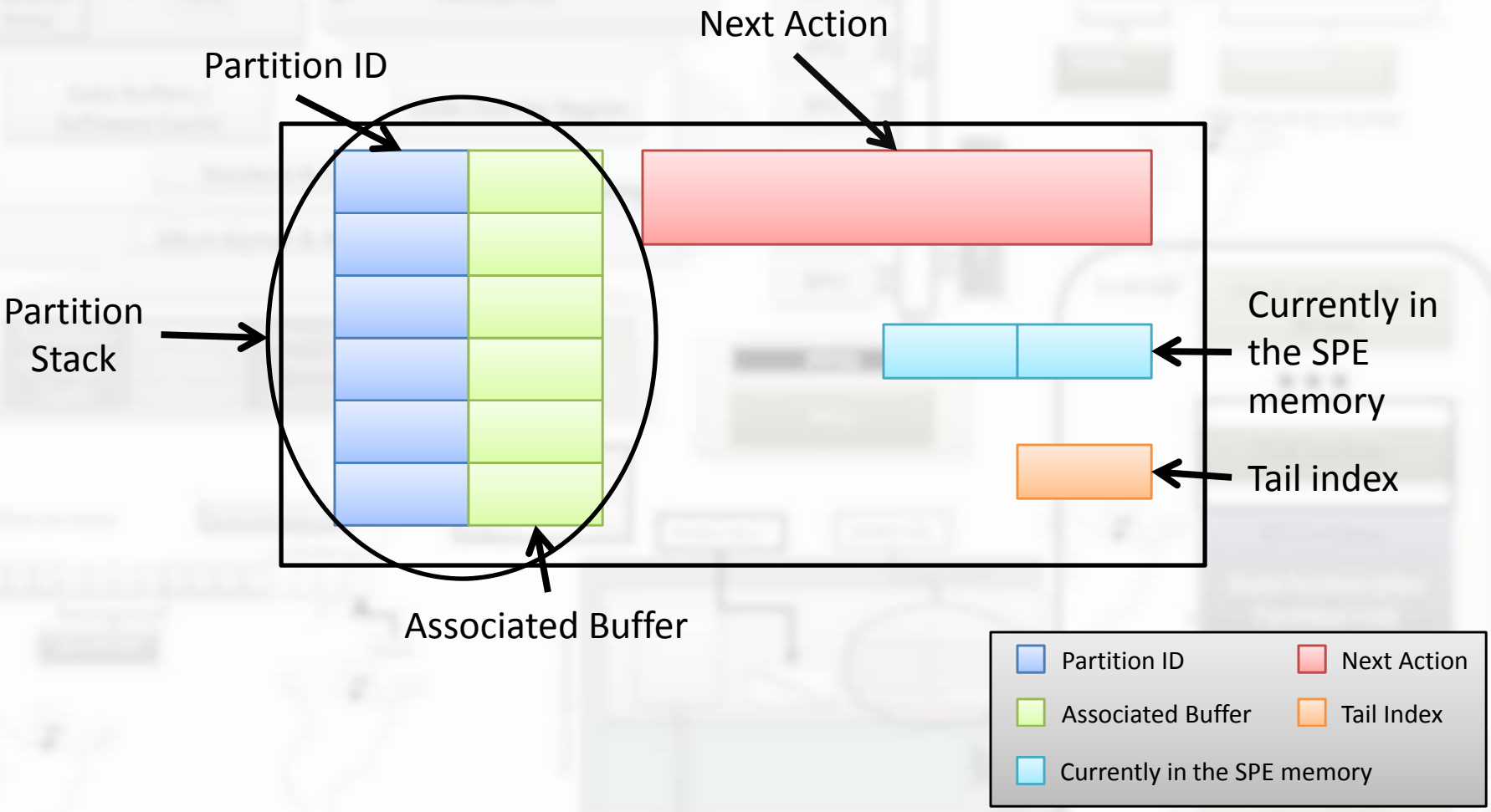
- Modulus approach
  - FIFO Approach: Oldest partition gets replaced
  - Helpful with long chains of functions
  - Relocation problem
    - What happens when a returning partition gets reloaded?
    - Associated buffer
      - An EWOR partition will be loaded in the buffer which was first loaded in.
      - Prevents relocation problems but negates replacement policies for returning partitions
  - Formula: Next loading buffer ➔ tail = (tail + 1) % N

An Example

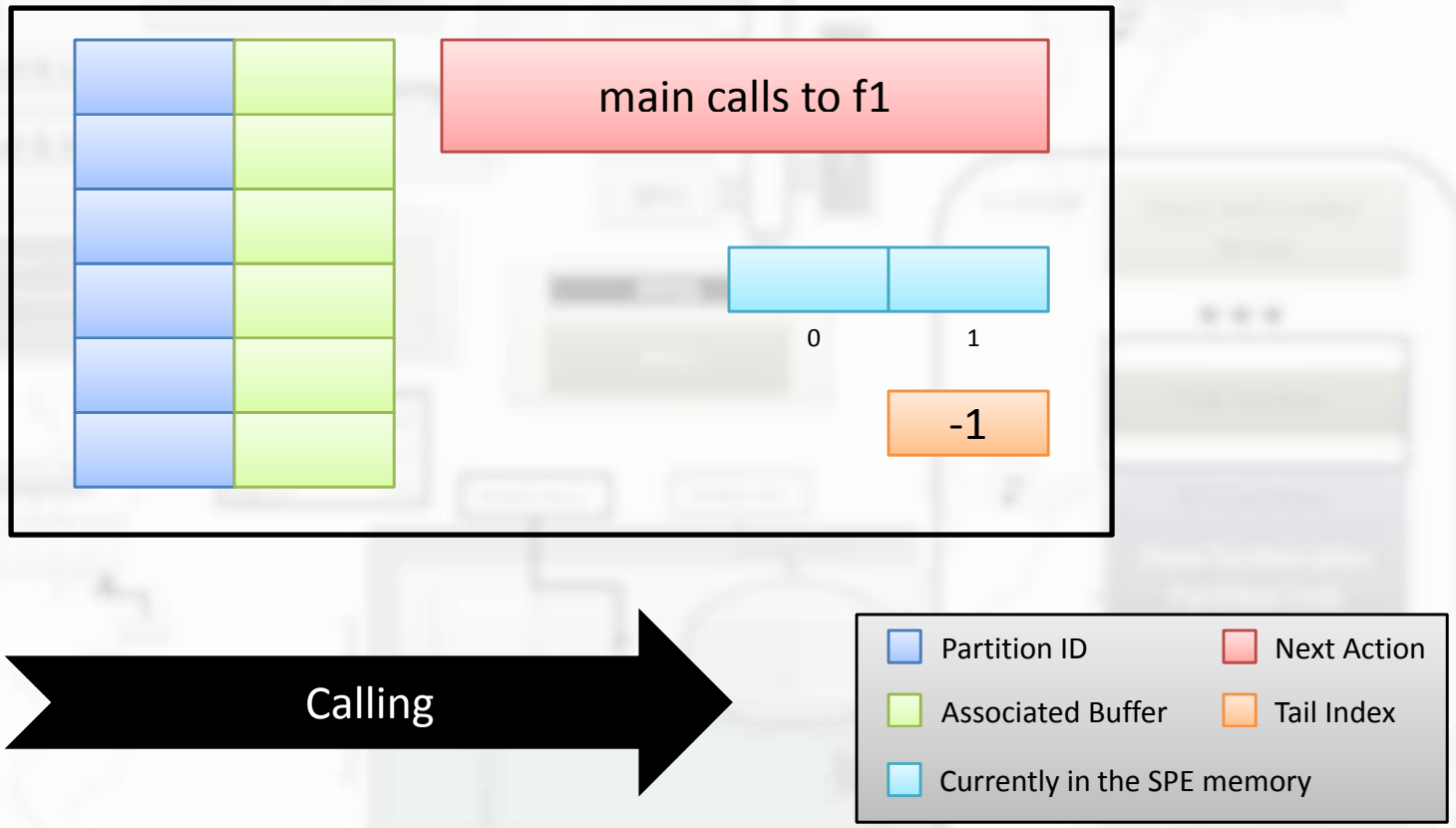
```
#pragma partition 0 main
#pragma partition 1 f1 f4
#pragma partition 2 f2
#pragma partition 3 f3
#pragma partition 4 f5
#pragma partition 5 f6
#include <stdio.h>
void f1() {f2();}
void f2() {f3();}
void f3() {f4();}
void f4() { printf(";-P"); }
typedef unsigned long long __ea_t;
int main(__ea_t spuid, __ea_t argp, __ea_t envp){
        f1();
        return 0;
}
```



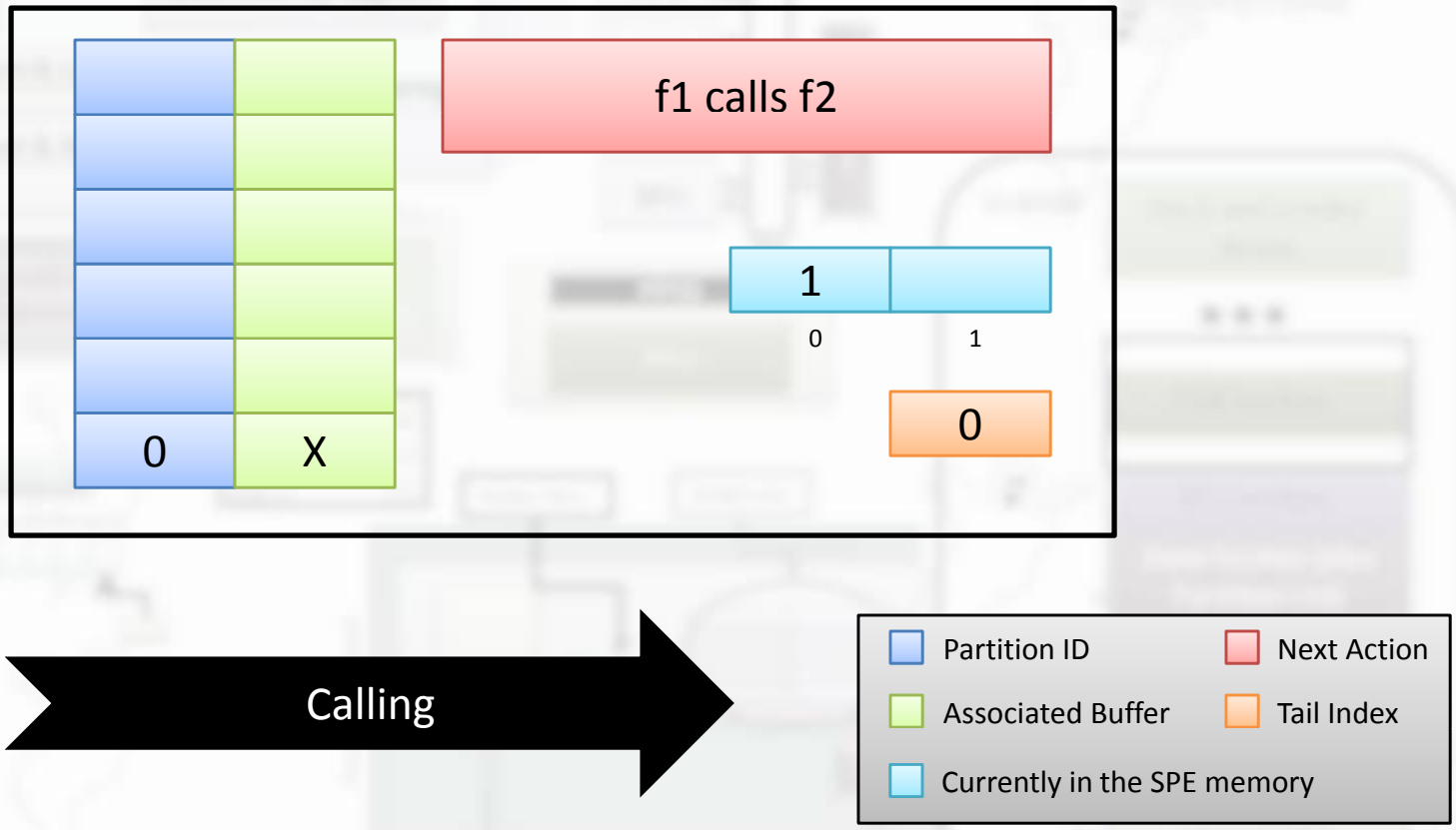
Side Note: Helper Structures



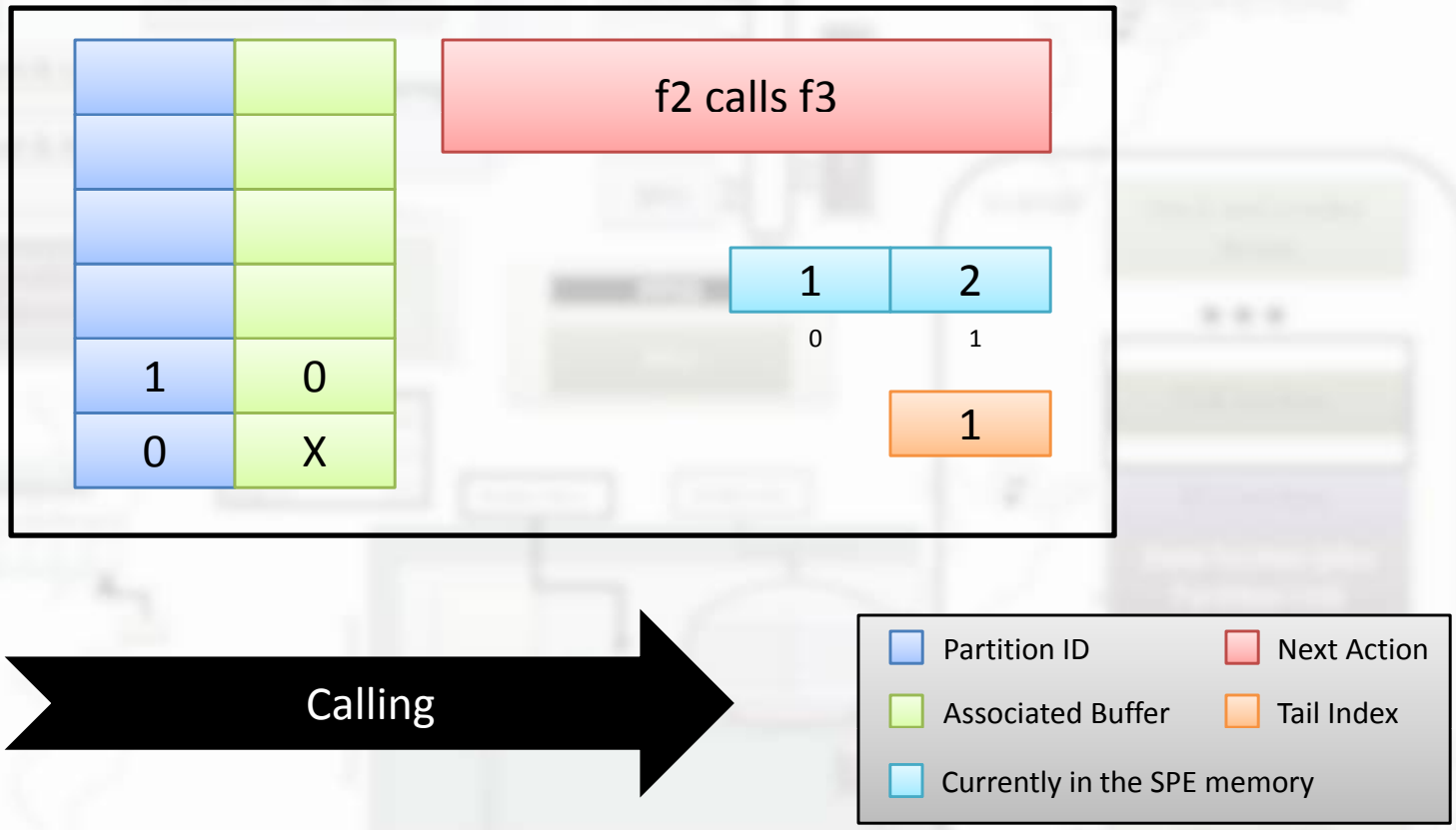
An Example


main calls to f1
0
1
-1
Calling
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

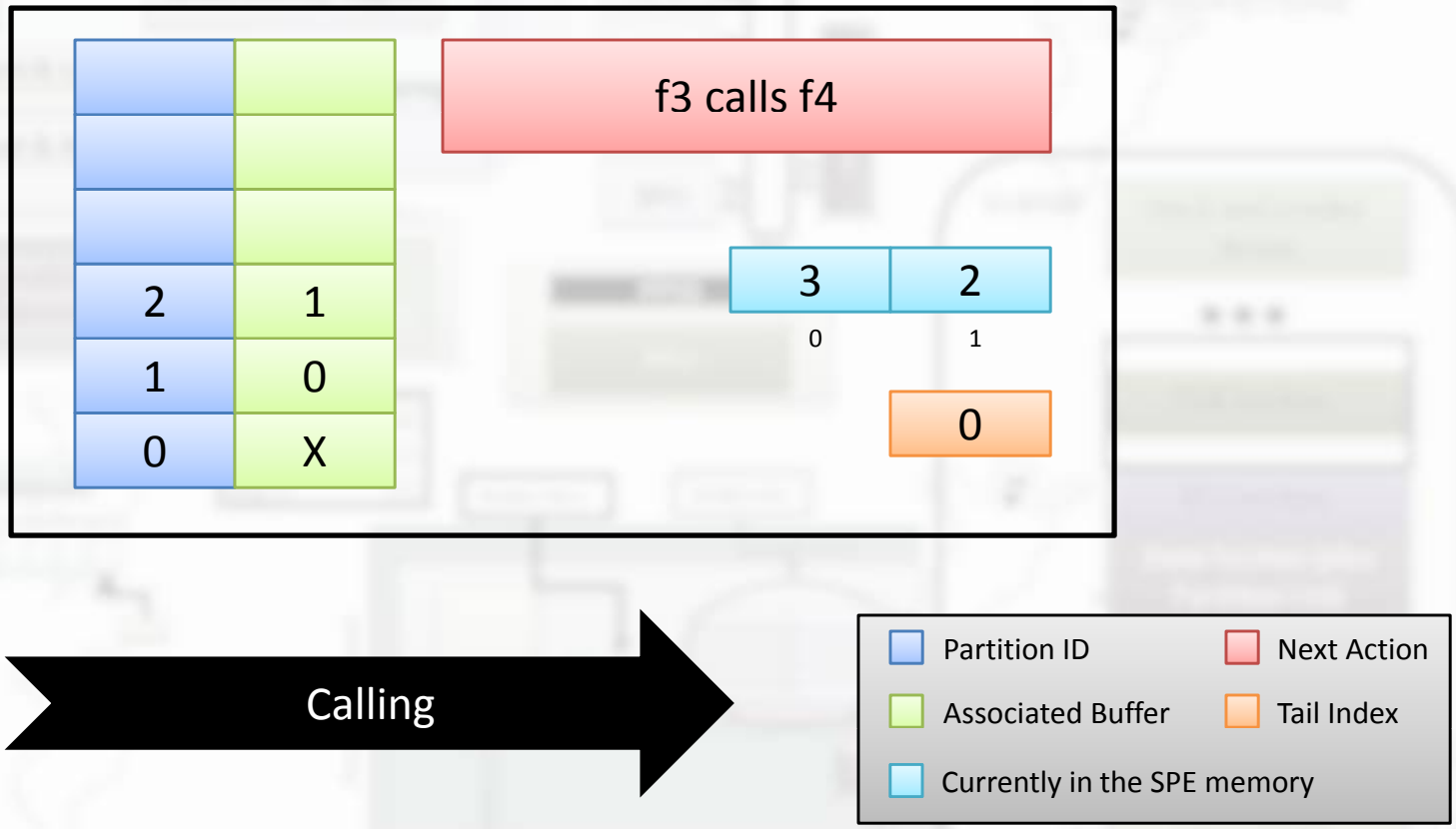
An Example


f1 calls f2
1
0
1
0
0
X
Calling
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

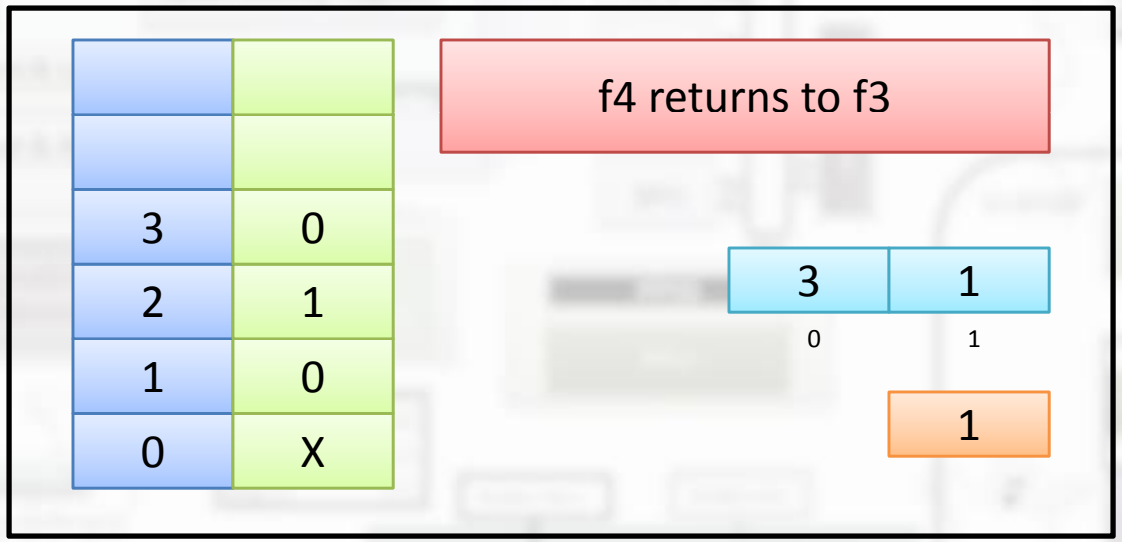
An Example


f2 calls f3
1
2
0
1
1
1
0
0
X
Calling
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

An Example


f3 calls f4
2
1
1
0
0
X
3
2
0
1
0
Calling
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

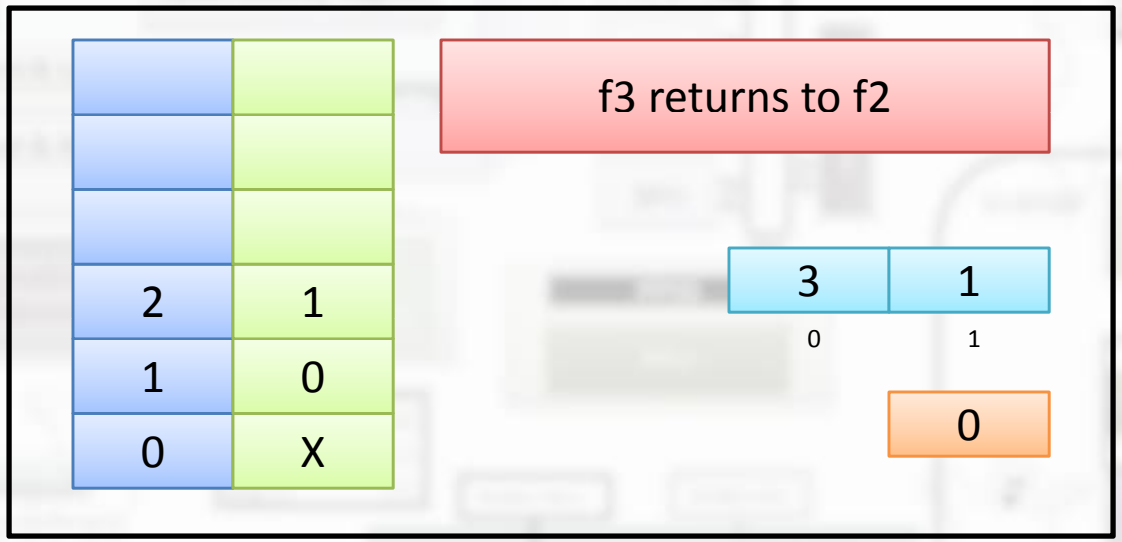
An Example



Returning

Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

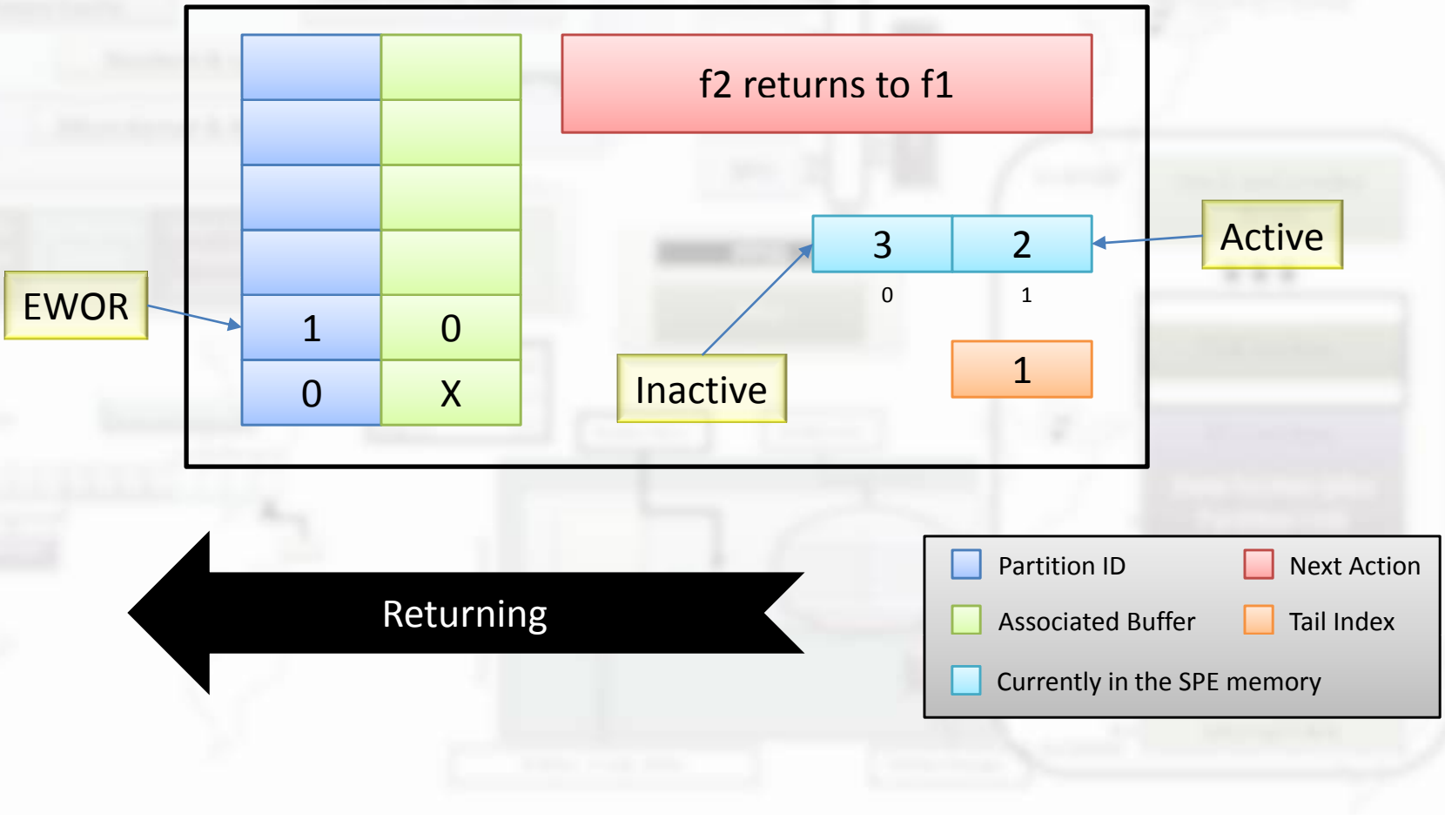
An Example



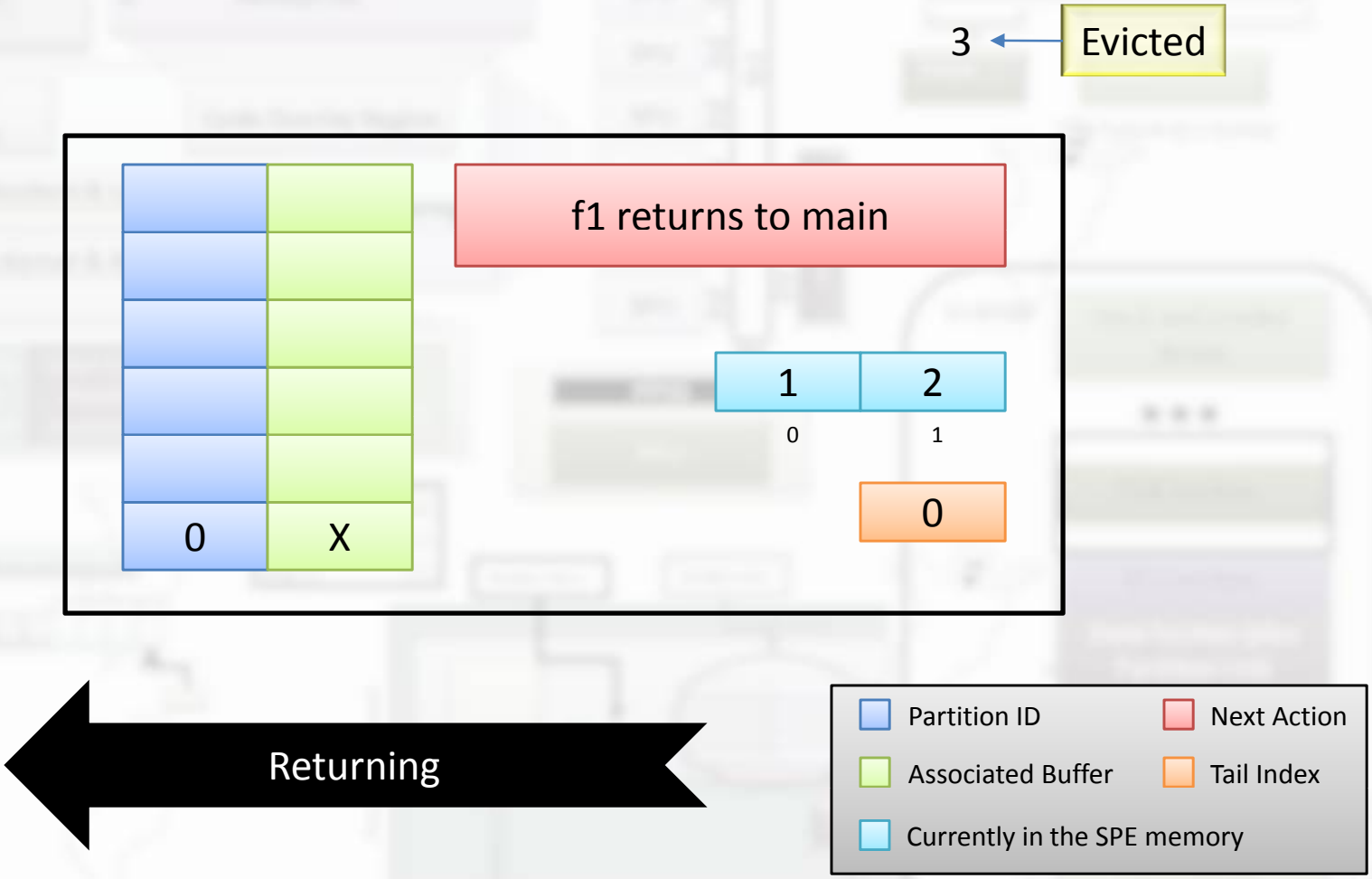
Returning

Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

An Example


f2 returns to f1
EWOR
1
0
0
X
3
2
0
1
Active
Inactive
1
Returning
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

An Example


3
Evicted
f1 returns to main
1
2
0
1
0
0
X
Returning
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

Side Note
Lexicon Time!!!!

- Saturation Point
  - In chemistry, when adding more solvent will not result in a higher concentration solution since all solvent molecules have been bonded
  - In our case, when sub dividing the buffer will not achieve a reduction of DMA transfers since the partitions will not fill them.

Outline

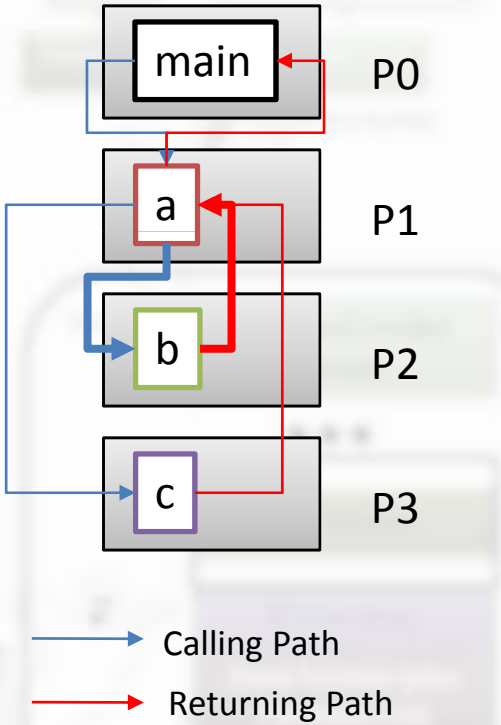
- Objective
- Partition Manager: the why, the what and how?
- Cache Like Schemes: Modulus approach
- **Cache Like Schemes: LRU-like approach**
- Testbed, results, conclusions and future work

Cache-Like Schemes

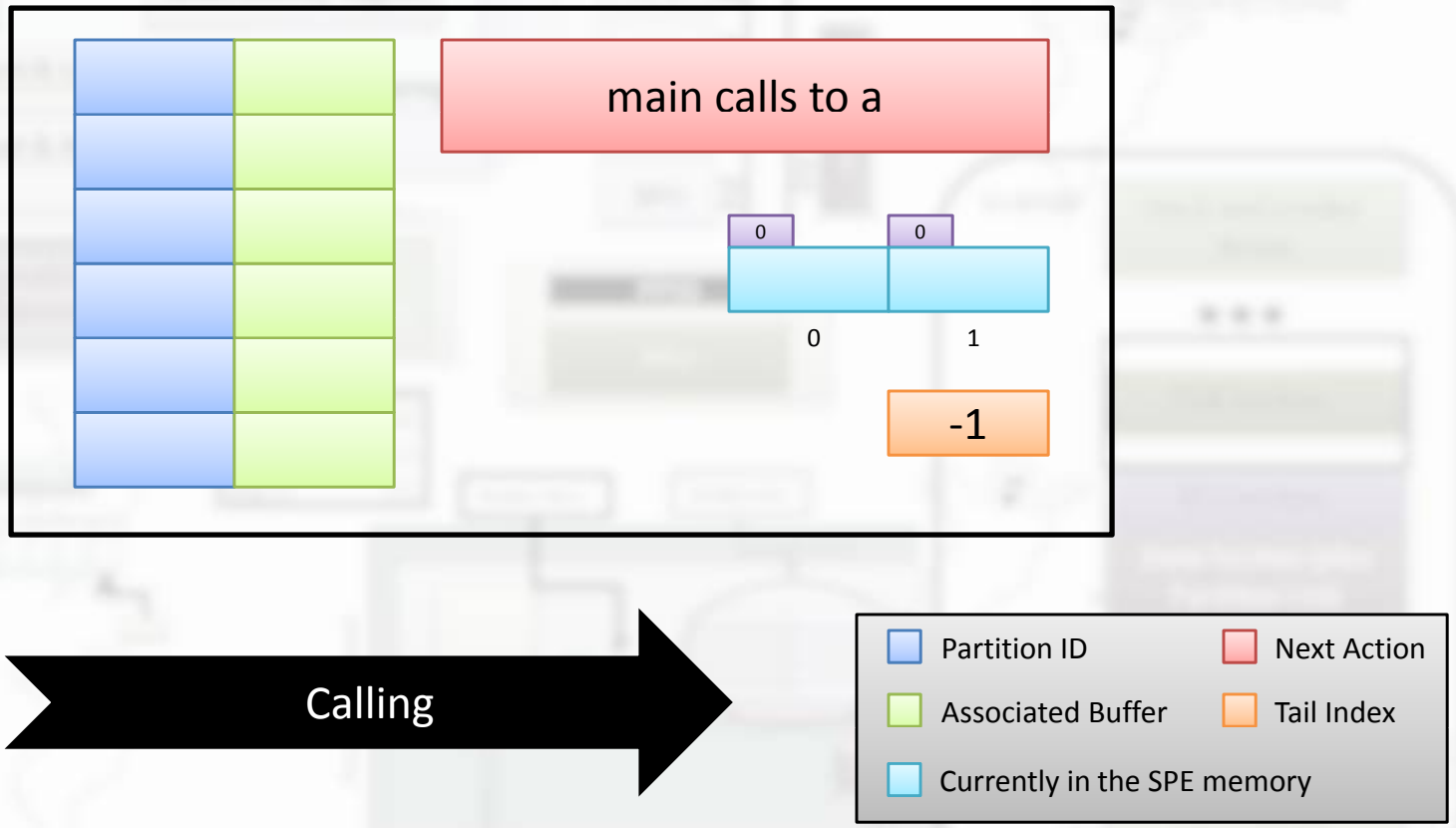
- LRU Like Approach
  - Each sub-buffer has a count
  - It decrements when a partition action takes and it does not involve the partition in that sub-buffer.
  - The smallest count is replaced
    - If a set of smallest partitions is encountered, a random one of the set is replaced

Another Example

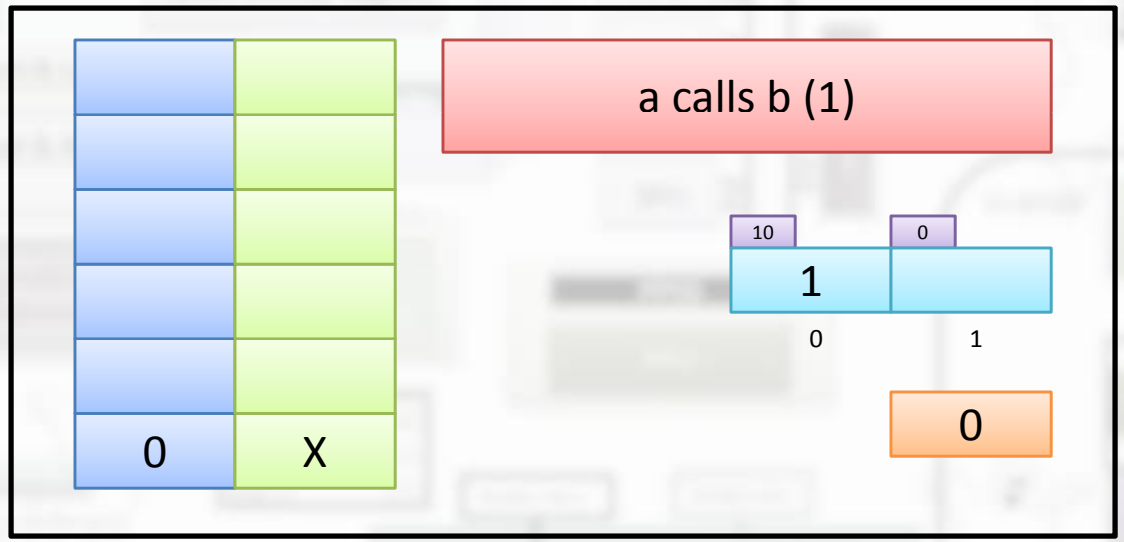
```
#pragma partition 0 main
#pragma partition 1 a
#pragma partition 2 b
#pragma partition 3 c
#include <stdio.h>
int c(int ff){ return  (ff-1); }
int b(int ff){ return ff*8; }
int a(int ff){
   int x;
   for(x = 0; x < 2; ++x){
      ff = b(ff);
   }
   return c(ff);
}
typedef unsigned long long __ea_t;
int main(__ea_t spuid, __ea_t argp, __ea_t envp){
            a();
            return 0;
}
```

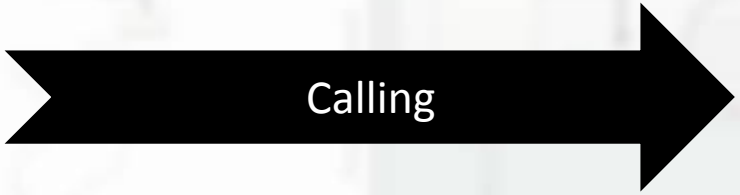


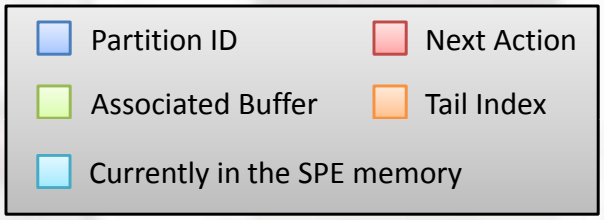
An Example


main calls to a
0
0
0
1
-1
Calling
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

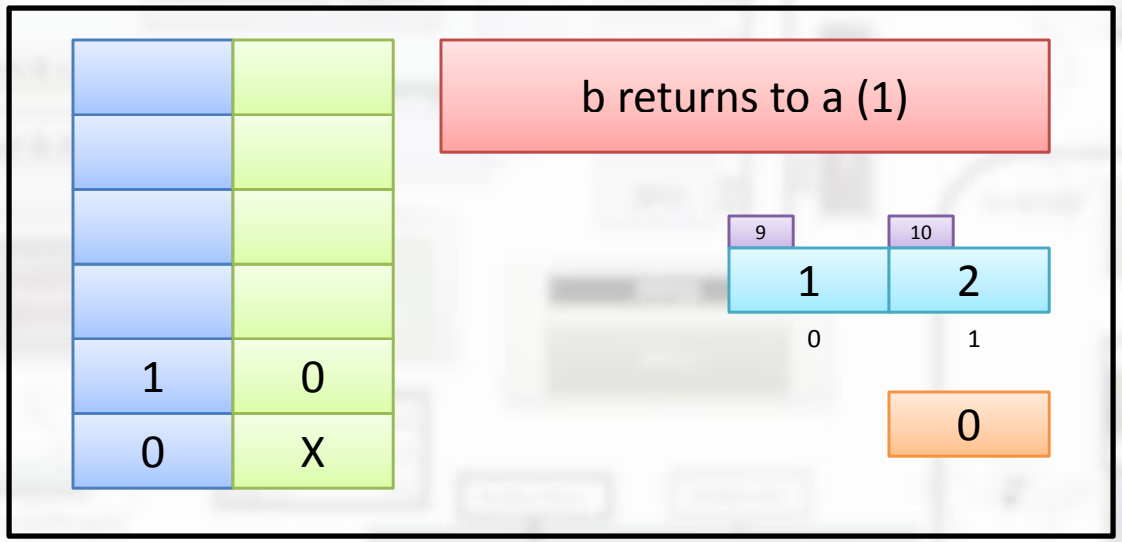
An Example


a calls b (1)
10
0
1
0
1
0
0
X

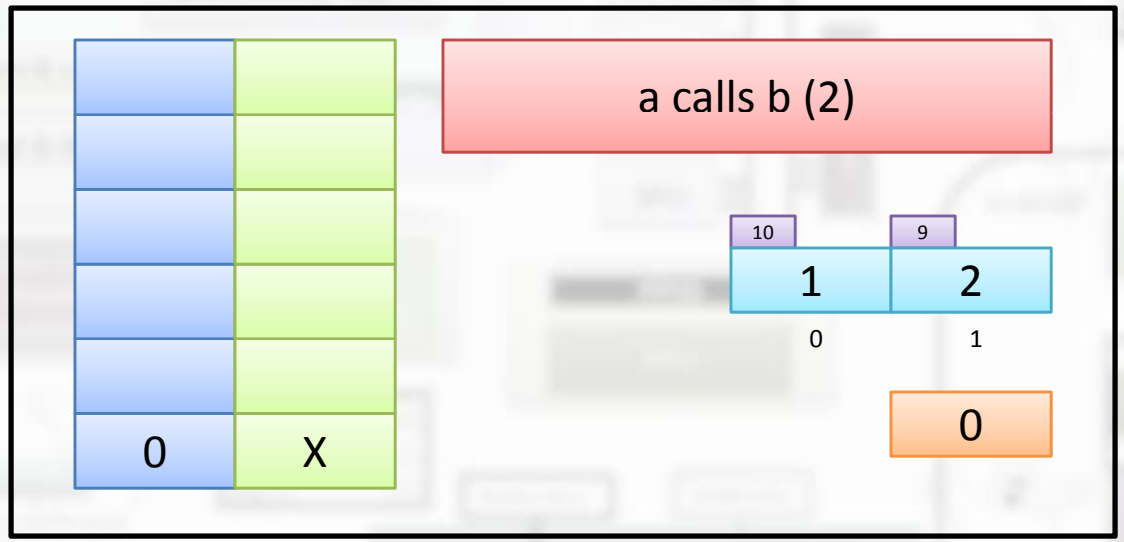

Calling

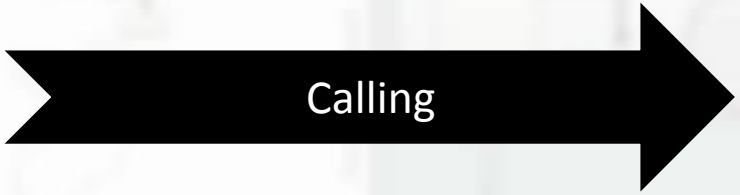

Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

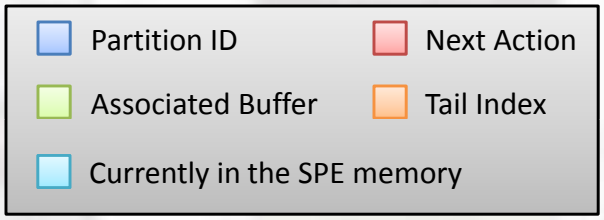
An Example


b returns to a (1)
9
10
1
2
0
1
0
1
0
0
X
Returning
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

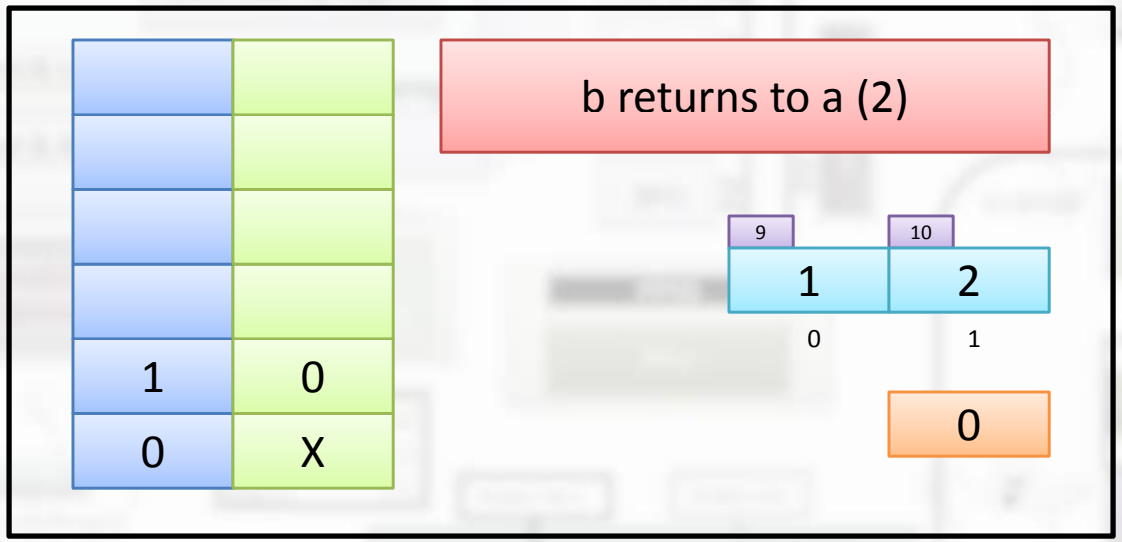
An Example


a calls b (2)
10
9
1
2
0
1
0
0
X

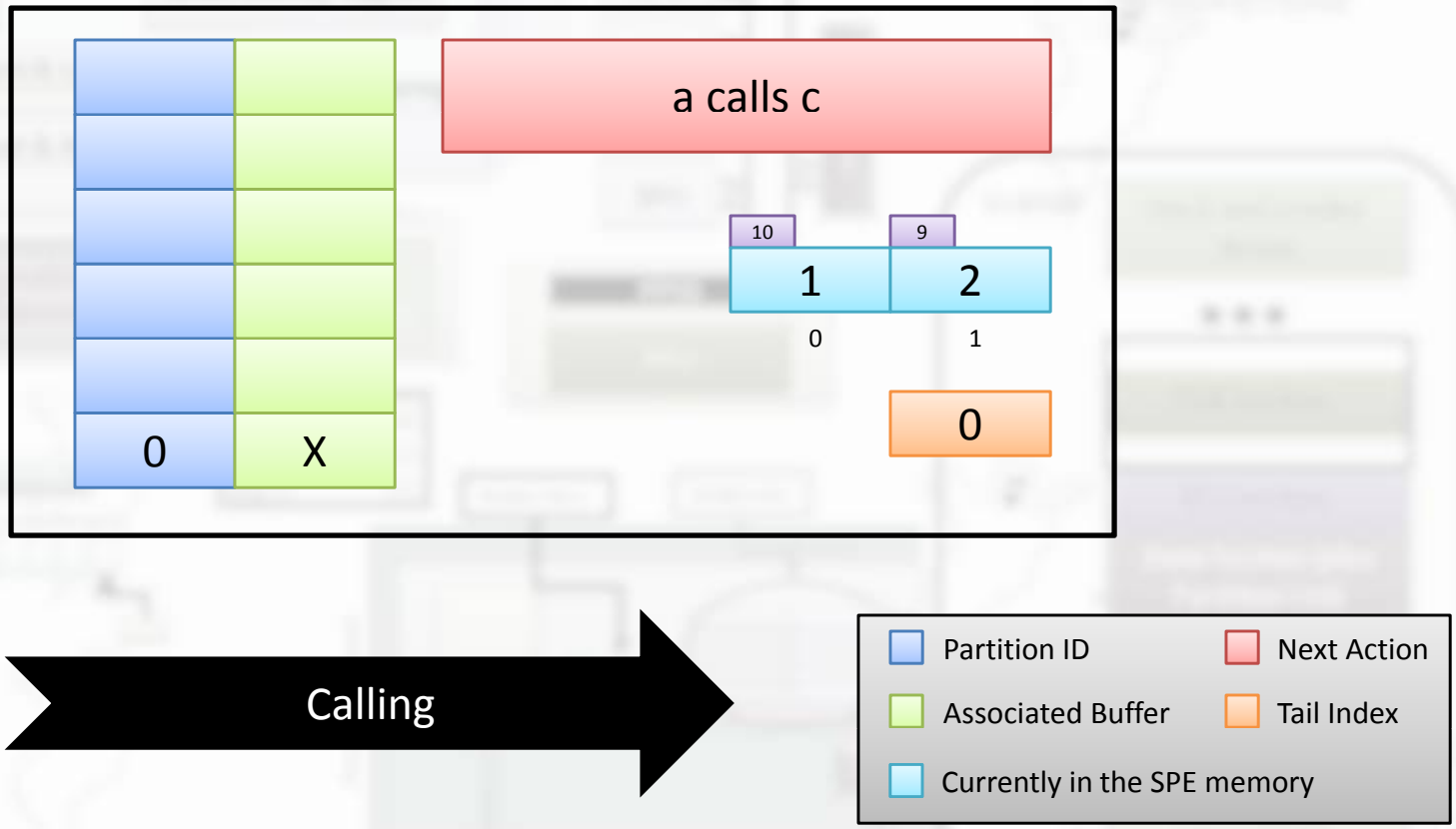

Calling


Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

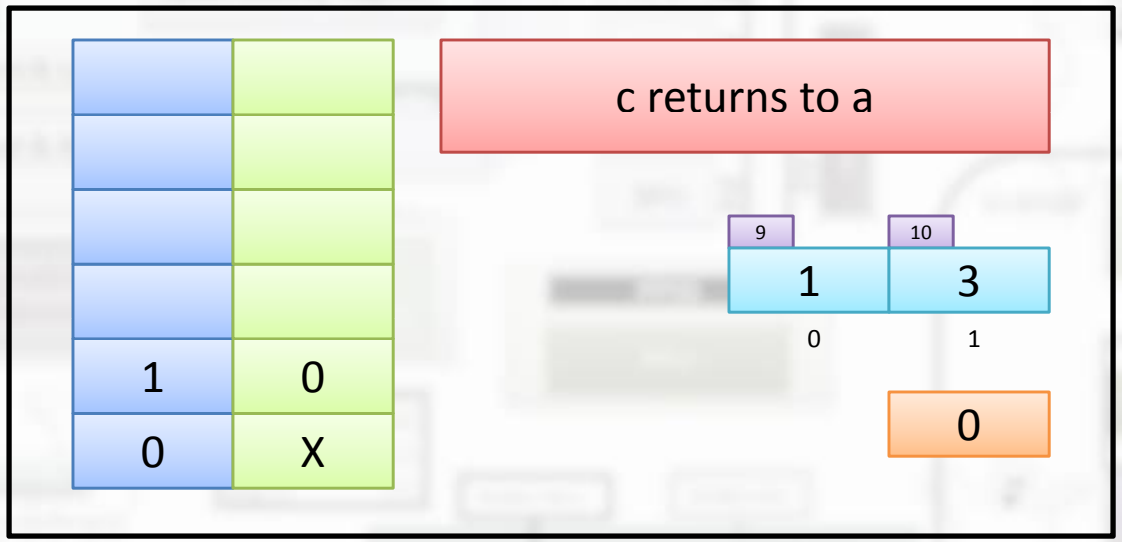
An Example


b returns to a (2)
9
10
1
2
0
1
0
1
0
0
X
Returning
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

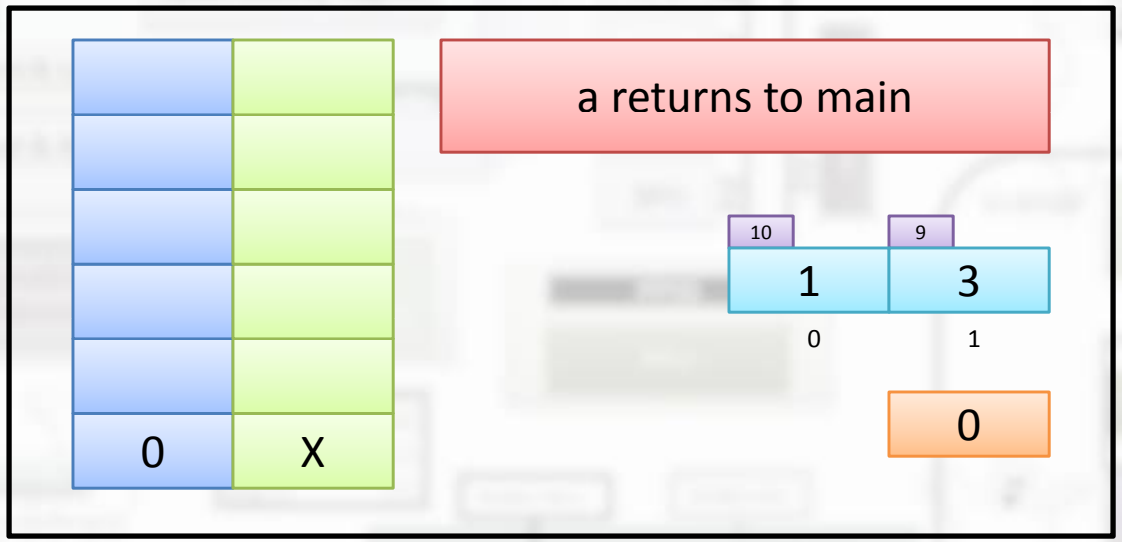
An Example


a calls c
10
9
1
2
0
1
0
0
X
Calling
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

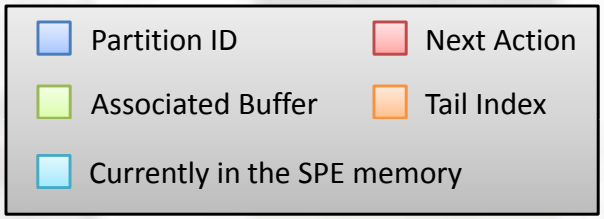
An Example


c returns to a
9
10
1
3
0
1
0
1
0
0
X
Returning
Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

An Example


a returns to main
10
9
1
3
0
1
0
0
X


Returning


Partition ID
Next Action
Associated Buffer
Tail Index
Currently in the SPE memory

Outline

- Objective
- Partition Manager: the why, the what and how?
- Cache Like Schemes: Modulus approach
- Cache Like Schemes: LRU-like approach
- **Testbed, results, conclusions and future work**

Testbed

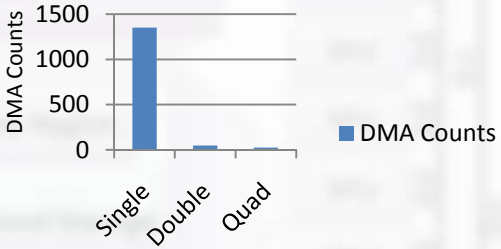
- A Playstation 3 system running YDL 5
- GCC 4.2 compiler
- CELL B.E. SDK 1.1
- A Set of Micro benchmarks
  - A set of DSP kernels
  - SPEC gzip compression utility
  - A jacobi simulation, a multi grid program, a laplace kernel, and a molecular dynamic simulator
  - A group of seven synthetic benchmarks

Results

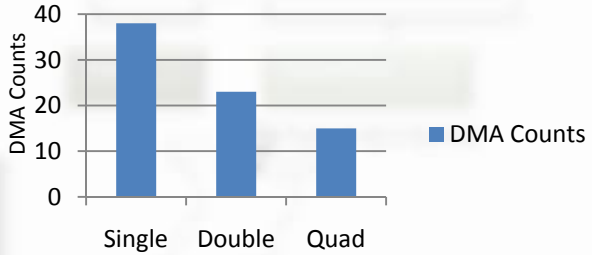
Single versus 2- versus 4-Buffers (Modulus)

DSP Kernels

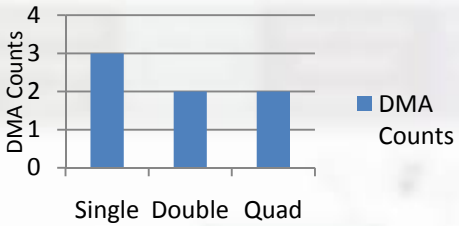
GZIP Compression

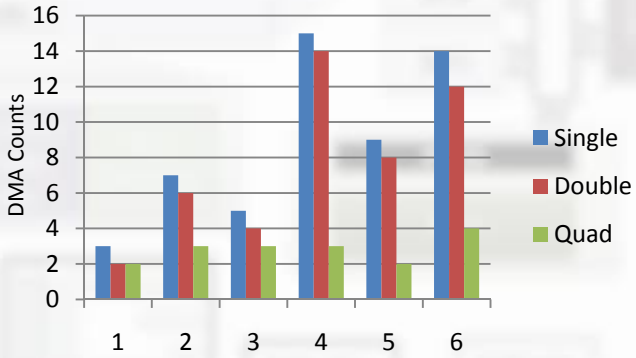


GZIP De compression

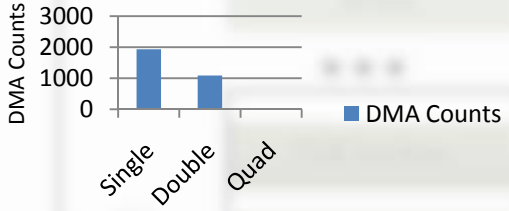


JACOBI

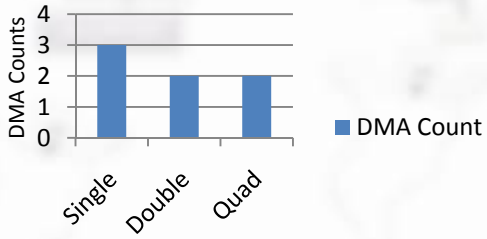




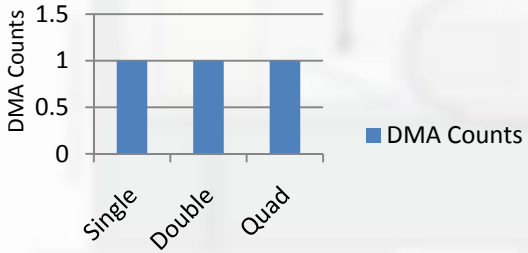
MGRID



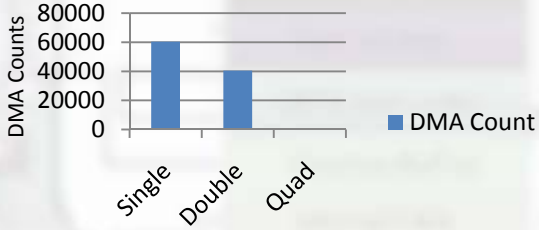
MD



LAPLACE



Synthetic Case 7



Results

4-Buffer-Modulus versus 4-Buffer-LRU

Conclusions

- Adding software buffers can greatly reduce the number of code DMA in the partition manager
- The replacement policies can have adverse or beneficial behavior according to the application behavior
- Applications possesses a “saturation” point which can be found to find an optimal N sub-buffering scheme

Future Work

- Find a heuristic for saturation points [The Critical Path of the Partition Graph]
- Work on the buffering schemes based on size and static frequency calls
- Introduce the concept of outlining for big functions and inlining for small ones
- Investigate how the EWOR partitions might be more effectively managed

Bibliography

- Jiang, Yi, “Design and Implementation of tool-chain framework to support OpenMP single source compilation on the CELL Platform.” winter 2007. University of Delaware
- Manzano, J, et. Al, “Towards an Automatic code layout framework.” International Workshop on OpenMP. June 3rd - June 7th, 2007. Tsinghua University, Beijing, China

Thanks!!!!

"But what ... is it good for?"
** Engineer at the Advanced Computing Systems Division of IBM, 1968, commenting on the microchip.*

"I'm not dumb, I just have a command of thoroughly useless information."
-- Bill Watterson, cartoonist of Calvin & Hobbes