SONY
make.believe

An Insight into Advanced XIP File System (AXFS)

Software Architecture Division (SARD)
Sony India Software Center Pvt Ltd

Aaditya Kumar

AGENDA

- Introduction
- XIP (Execute In Place)
- AXFS - Overview
- AXFS - Profiling
- AXFS – Implementation
  - Byte Tables
  - Image Format
  - Details
- AXFS – Performance
  - Other tuning tools
  - Launch performance
  - Flash footprint.
  - Memory footprint
- Summary

SONY
make.believe

BACKGROUND

- We recently pushed Advanced XIP File System (AXFS) to 3.4-LTSI
- We think AXFS is more powerful than its counterparts (CRAMFS, SQUASHFS) in many aspects.
- We think AXFS deserves more wider usage than it has currently.
- This talk is about giving an insight into AXFS.
- So more people can start to playing with it and using it more in their products.
- And make it even better!

SONY
make.believe

INTRODUCTION

• Cost, power, size and other constraints of embedded systems make compressed and read only file systems a good choice as a rootFS

• Commonly used root file system for embedded systems such as CRAMFS and SQUASHFS are read only and support compression.

• SQUASHFS supports compression of files in the file system.

• CRAMFS supports execute in place as well as compression of the files.

SONY
make.believe

INTRODUCTION (cont)

- CRAMFS supports execute in place and compression at file granularity.

- i.e. CRAMFS file can either be a execute in place file or a compressed file.

- AXFS is a read only file system which supports execute in place at a page level granularity.

- i.e. Each page in an AXFS file can either be made a compressed page or it can be a execute in place page.

- This presentation gives an insight into design and implementation of AXFS.

SONY
make.believe

INTRODUCTION (cont)

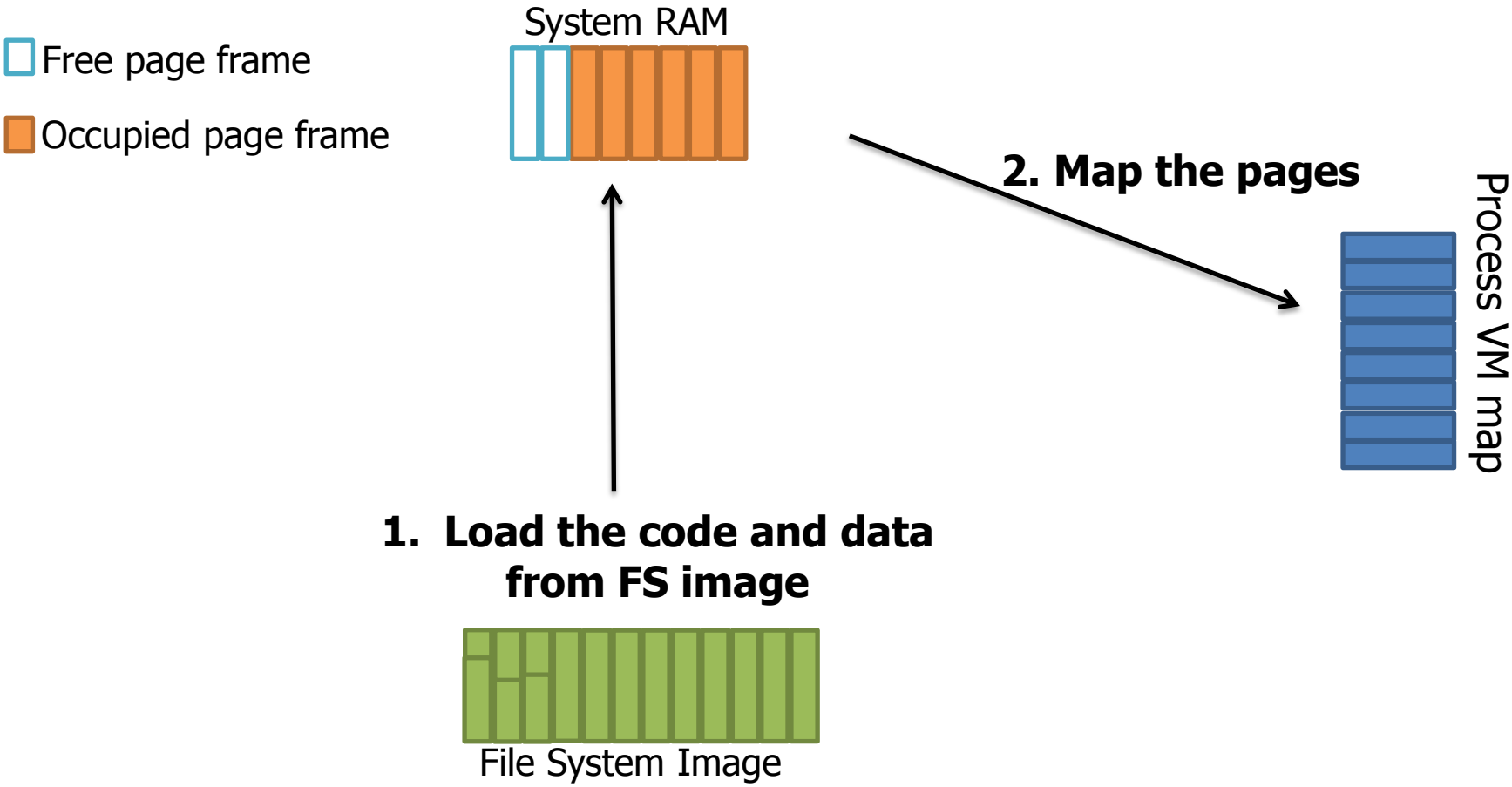
- We will start with introduction to the concept of eXecute In Place (XIP).
- We will look into the advantages and disadvantages of the XIP approach.
- We will look into AXFS file system image format.
- We will then take a short dive into the AXFS source code.

SONY
make.believe

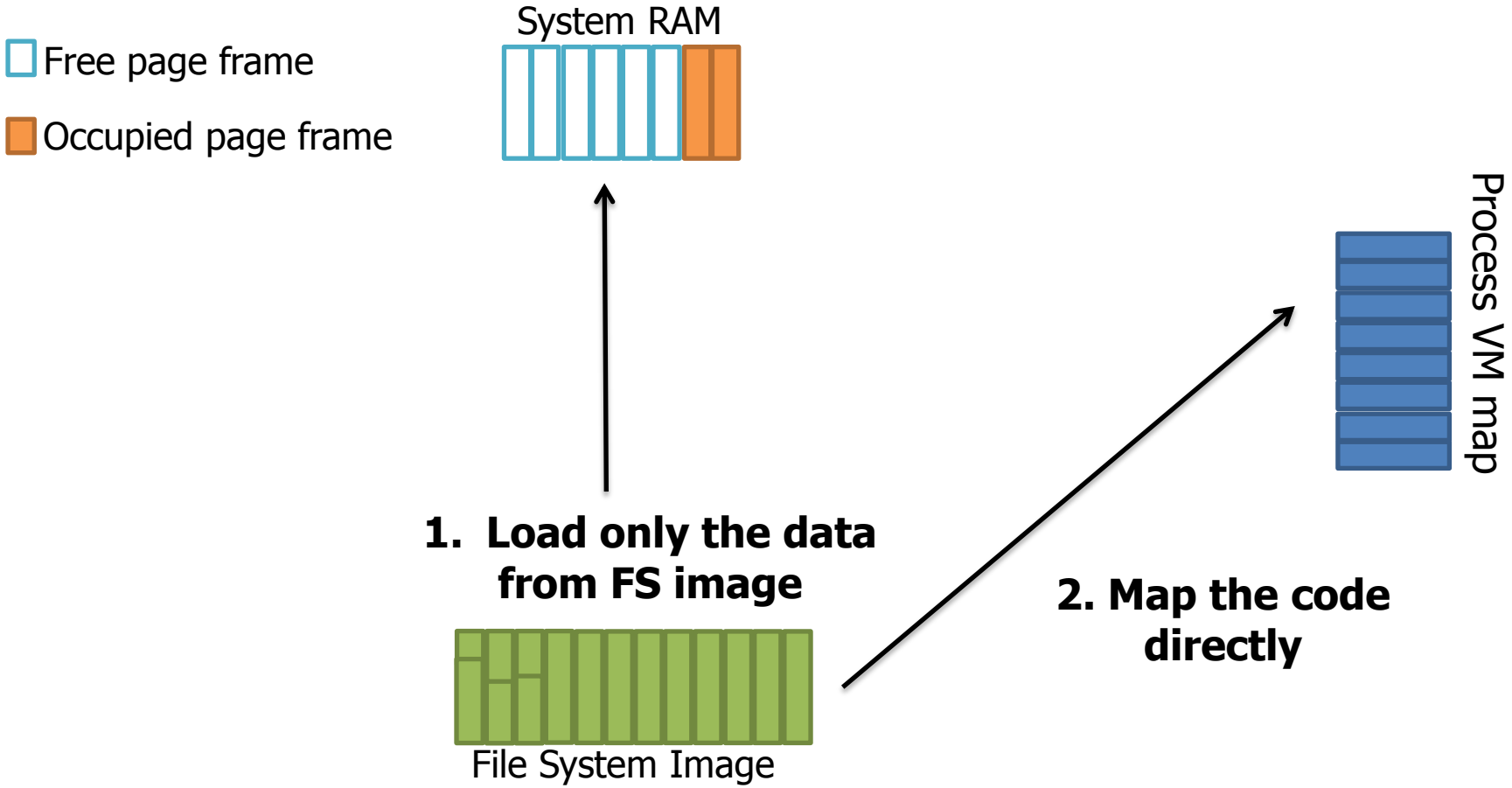
XIP - Overview

- XIP is a short form for eXecute In Place.
- Only possible on memory map-able devices. Such as NOR, ROM, RAM, etc.
- Device access speed of devices should be comparable to system RAM speed.
- The process's virtual memory is made to directly map to physical address of XIP image on media.
- No need to load code pages from media to page cache.

XIP – Overview (cont)



XIP – Overview (cont)



SONY
make.believe

XIP - PROS

- **Performance :**
  - XIP reduces boot and application launch time.
  - XIP is faster by virtue of saving the load time of the page from secondary memory.
- **Cost :**
  - Reduces requirement for system RAM.
  - Reduces power consumption by requiring less system RAM.
  - Helps in reducing cost.

SONY
make.believe

XIP - CONS

- **Slow for hotspots :**
  - •NOR flash usually used as memory device for XIP is slower than system RAM.
  - • XIP not suitable for code which are hotspots in the application or system.
- **Extra pass :**
  - • Needs a profiling pass to optimally use XIP.

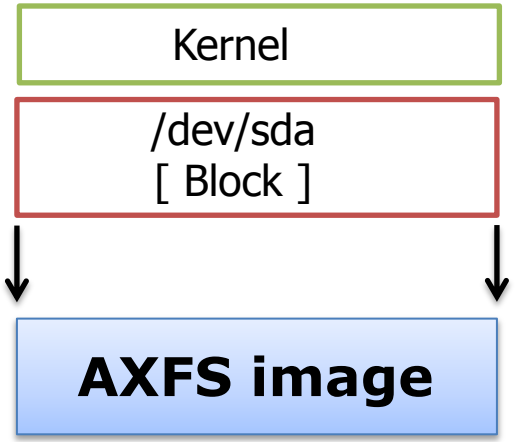
SONY
make.believe

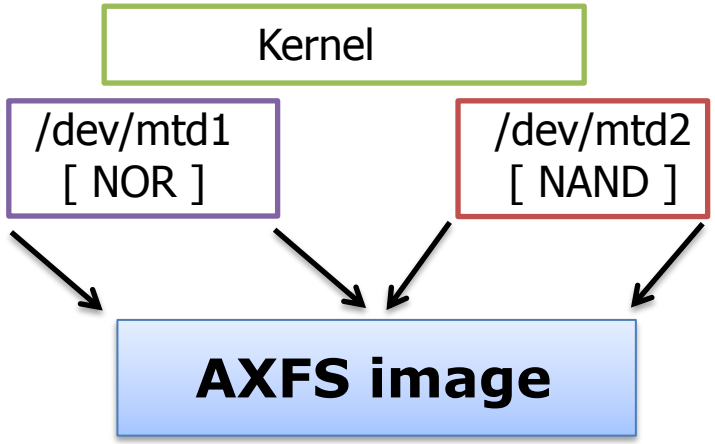
AXFS - OVERVIEW

- AXFS is a 64 bit, big endian and read only file system.
- Useful for small systems where memory and other resource are scarce.
- Allows XIP at page granularity.
- Supports compression in blocks of size 4KB to 4GB.
- Can mount directly from MTD device.

AXFS - OVERVIEW (cont)

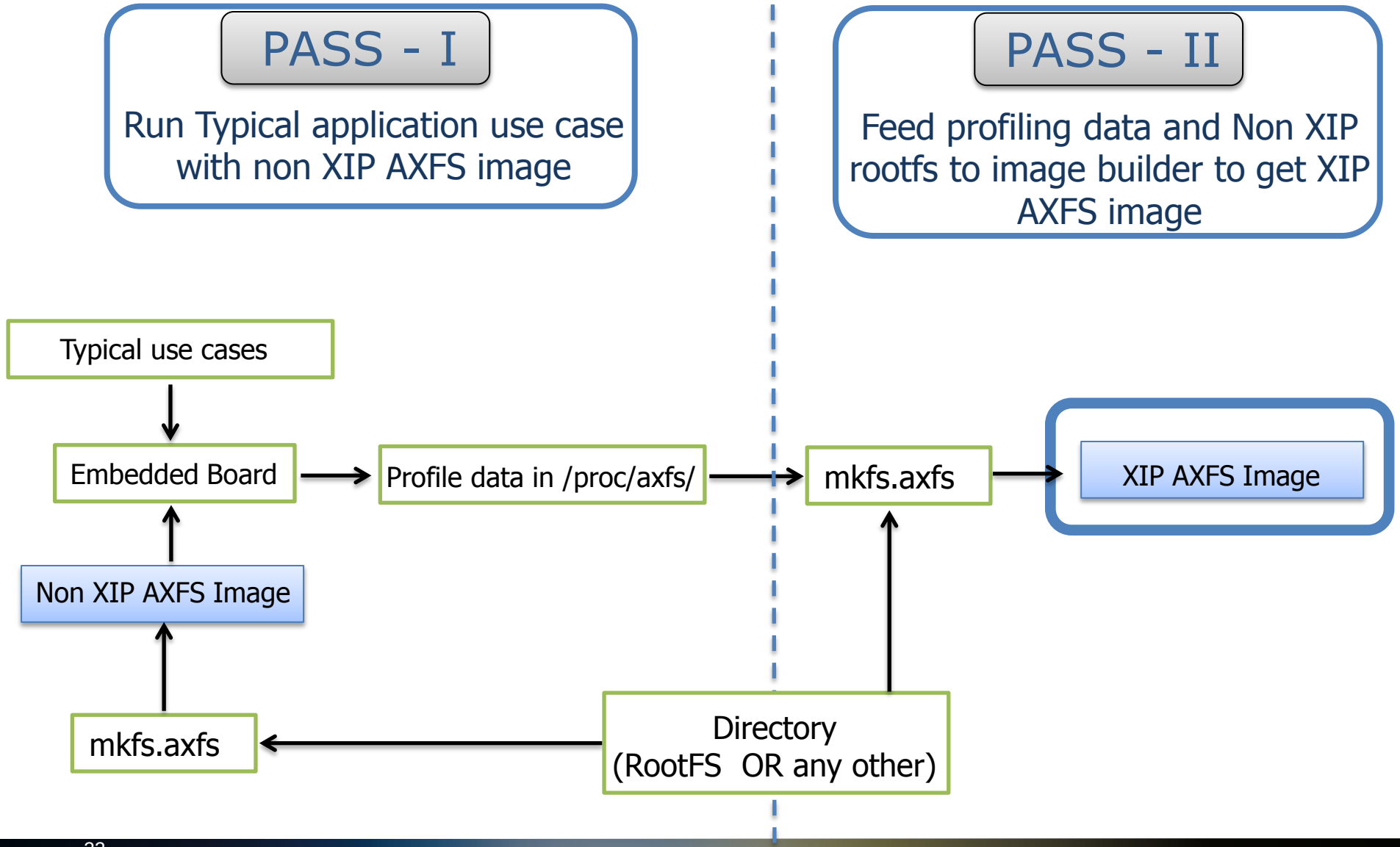
- AXFS allows the image to be mounted from two devices.
- This allows the XIP part of image to be on NOR flash and compressed part on the NAND.
- This device spanning is possible only if first device is a memory map-able device.
- This feature helps make system more economical.





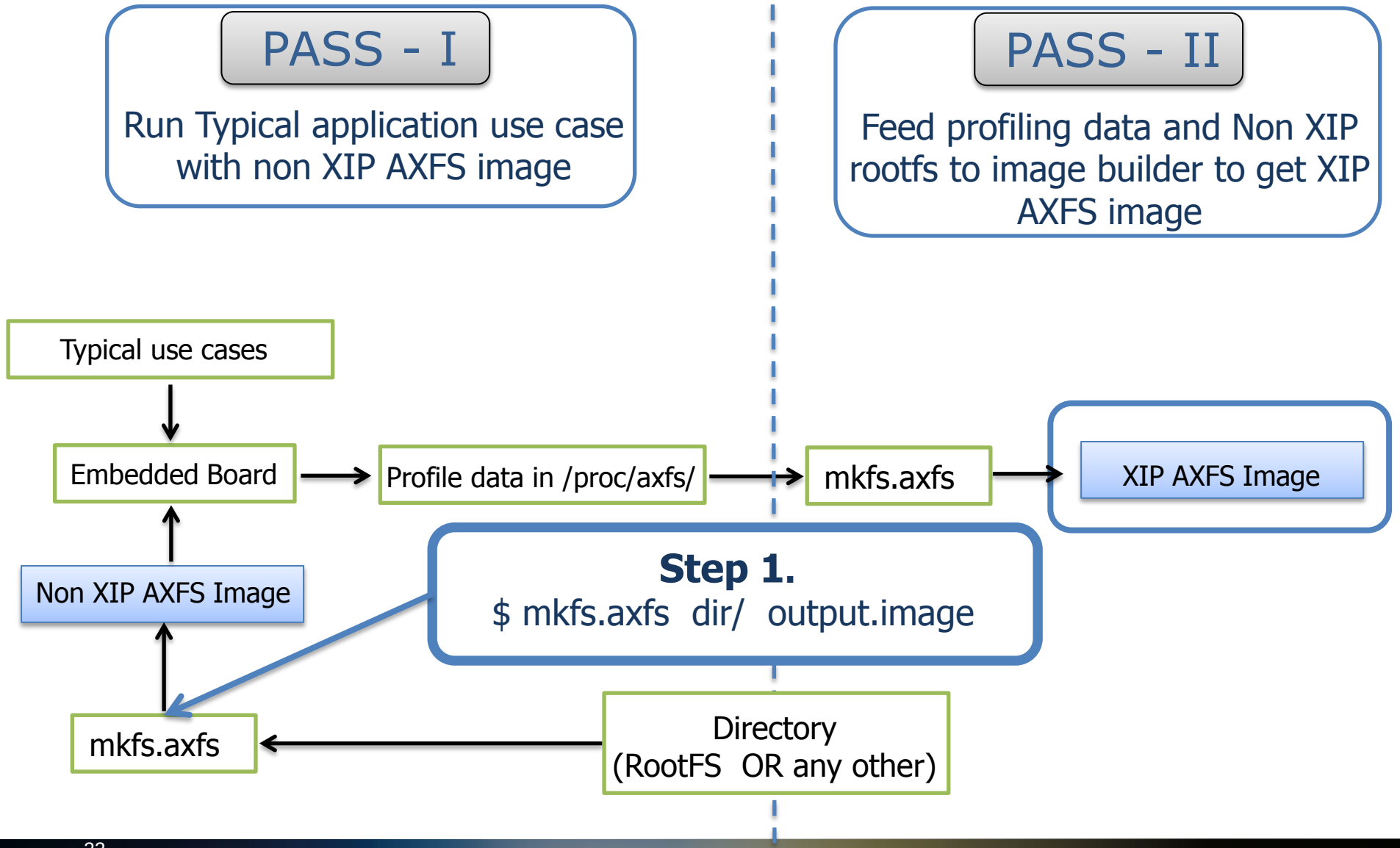
AXFS - PROFILING

SONY
make.believe



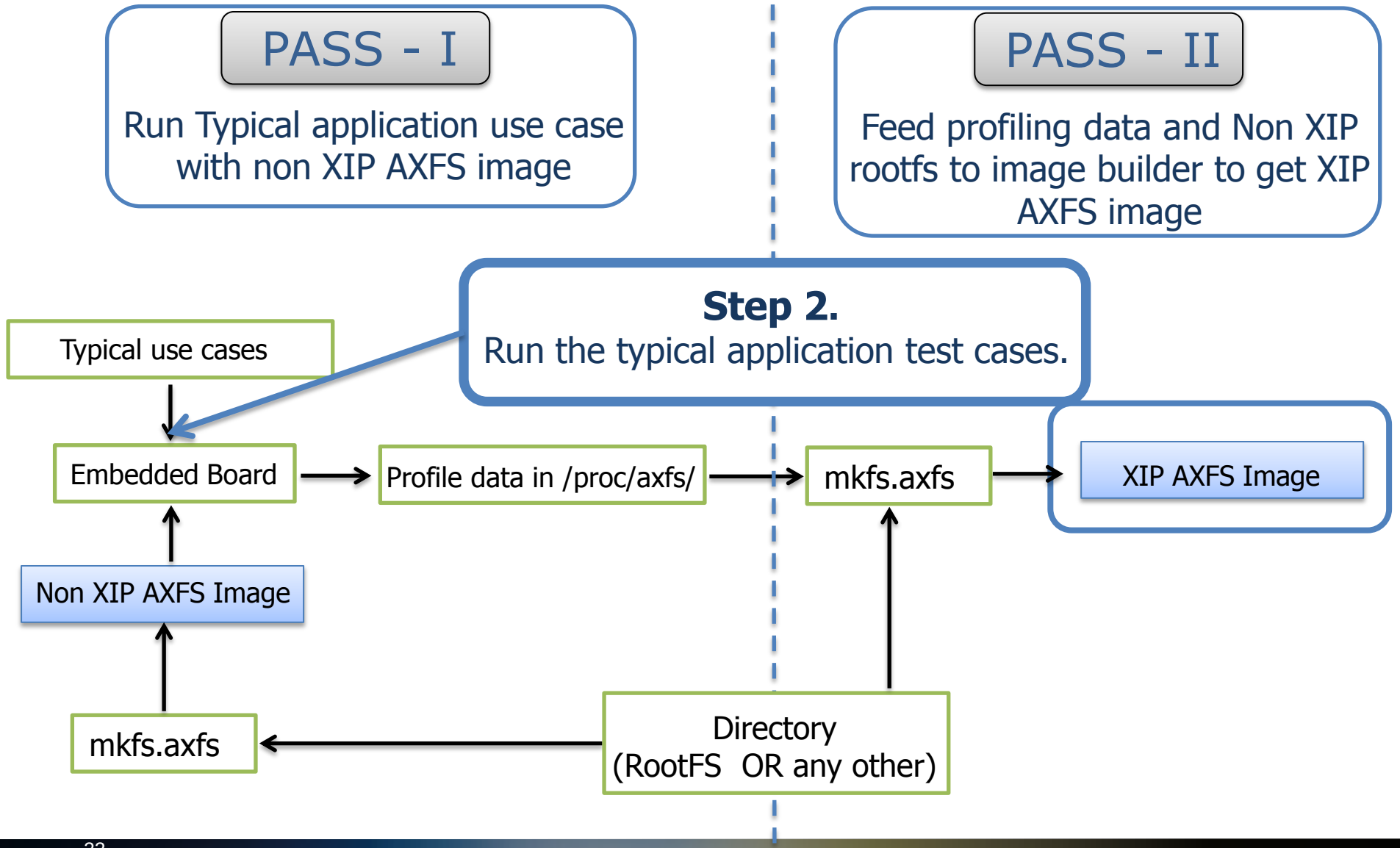
AXFS - PROFILING

SONY
make.believe

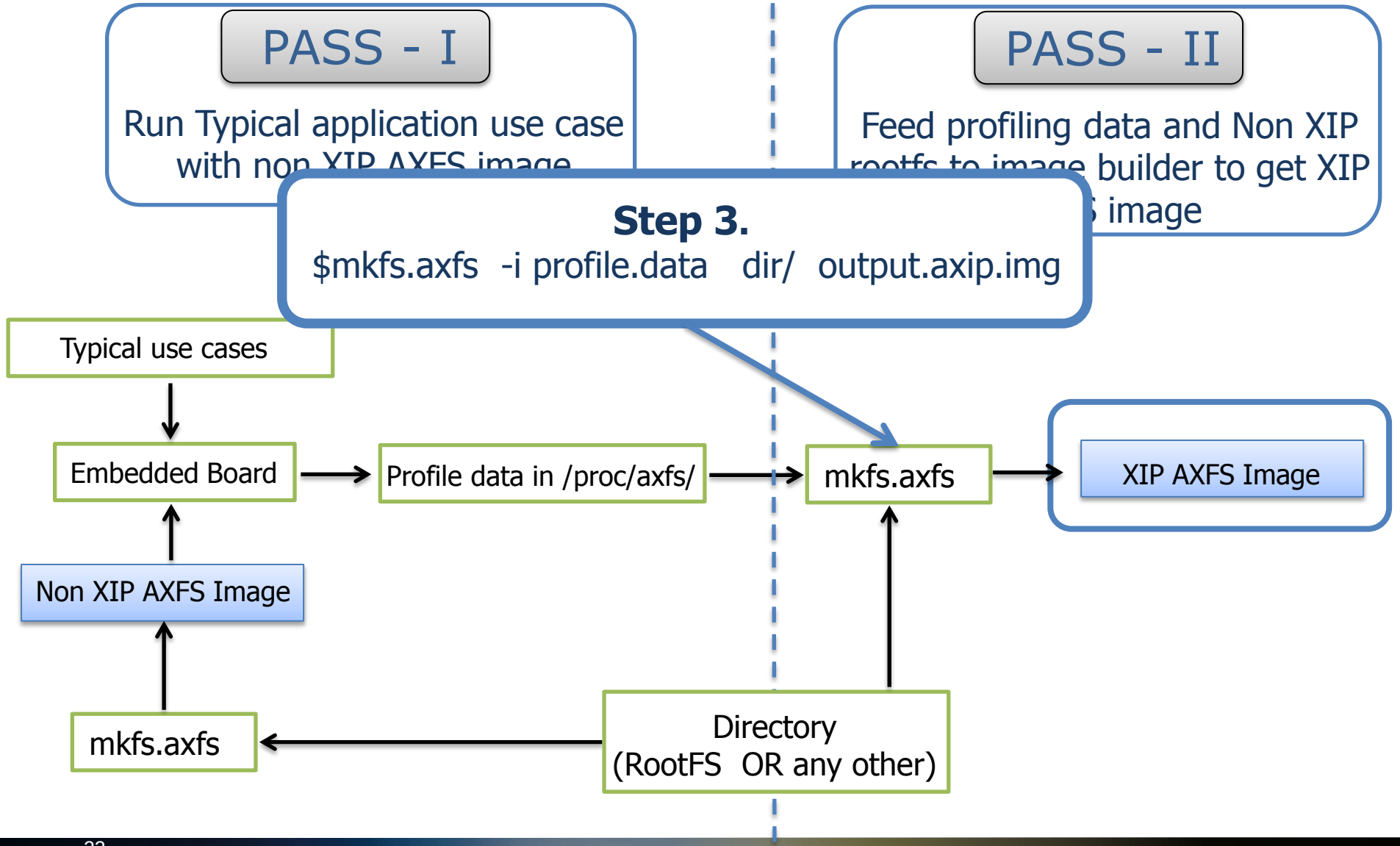


AXFS - PROFILING

SONY
make.believe



AXFS - PROFILING



SONY
make.believe

AXFS - PROFILING (cont)

- Pages that should be XIP can be found by profiling running application for executable pages.
- Profiling should cover all the important applications and there use cases.
- AXFS contains a built-in profiler for profiling applications.
- CONFIG_AXFS_PROFILING enables the inbuilt profiler.
- The log created by profiling pass is fed to the image builder.
- The built-in profiler should be compiled out after the image is created.

SONY
make.believe

AXFS – BYTE TABLES

• Byte Table are the mechanism which allows AXFS very less overhead in supporting 64 bit offsets.

• Each Byte Table contains only that many bytes required to hold the maximum value of a number.

• Number of bytes used in the Byte Table is the depth of a Byte Table.

• So, Number less than 256 can be represented with Byte Table of depth 1.

• Similarly, numbers less than, say, 500 need Byte Table of depth 2.

AXFS – BYTE TABLES (cont)

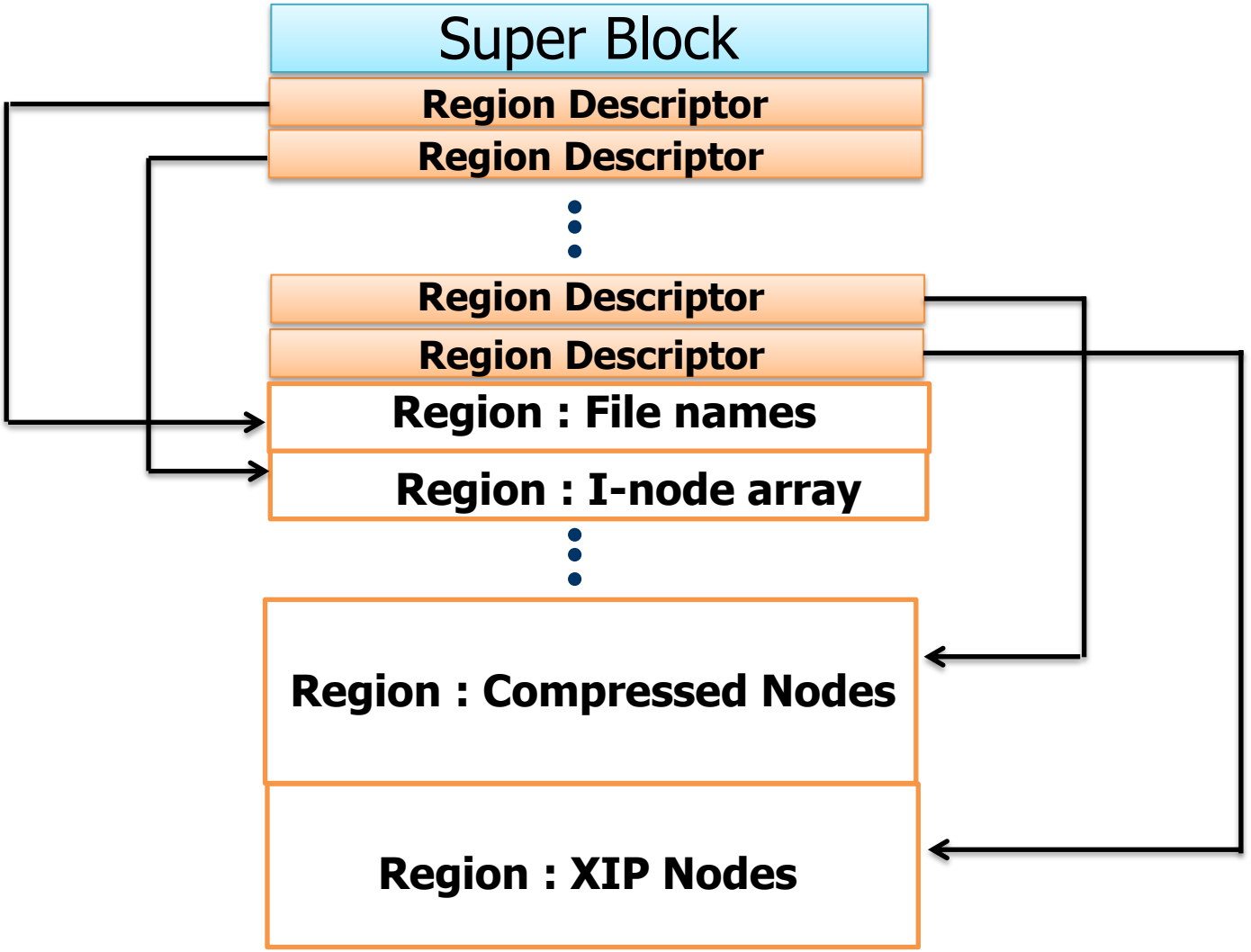
- Byte Tables is the key scheme which allows AXFS images sizes to be small and yet be a 64 bit file systems.

- Following is the code snippet for implementing Byte Tables in AXFS.

```
static inline u64 axfs_bytetable_stitch(u8 depth, u8 *table, u64 index)
{
    ….

    for (i = 0; i < depth; i++) {
        j = index * depth + i;
        bits = 8 * (depth - i - 1);
        byte = table[j];
        output += byte << bits;
    }
    return output;
}
```

AXFS – IMAGE FORMAT



SONY
make.believe

AXFS – IMAGE FORMAT (cont)

• In AXFS terminology a region is a contiguous segment of file system image that contains data.

• Region descriptors stores the location of a region in the file system image and also its attributes.

• Attributes stored in region descriptor include size of region, whether the region is compressed or XIP.

• Attributes store the depth and the length of Byte Table if the region contains Byte Table.

AXFS – IMAGE FORMAT (cont)

- Following is the on-media representation of region descriptor.

```
struct axfs_region_desc_onmedia
{
    u64 fsoffset;
    u64 size;
    u64 compressed_size;
    u64 max_index;
    u8 table_byte_depth;
    u8 incore;
};
```

- 'fsoffset' and 'size' is the offset from starting of the file system image and the size of the region.

- 'table_byte_depth' and 'max_index' are depth and length of the Byte Table.

AXFS – IMAGE FORMAT (cont)

SONY
make.believe

- Regions contain data as well as meta-data.
- Most of the meta-data is contained in regions containing Byte Table.
- Regions which contain data are “XIP region”, “Compressed region” and “Byte aligned region”.
- “XIP region” and “Compressed region” contains XIP and compressed pages respectively.
- “Byte aligned region” contains data which do not compress and so is stored uncompressed.
- The data/pages from files are stored in ‘nodes’ Which usually is 4KiB in size.
- Nodes have type. E.g. XIP type of nodes, compressed nodes or byte aligned type of nodes.

AXFS – IMAGE FORMAT (cont)

- Following is the snippet of the AXFS on-media super block.

```
struct axfs_super_onmedia {
    ....
    __be64 blocks;              /* number of nodes in fs */
    __be64 mmap_size;           /* size of the memory mapped part of image */
    __be64 strings;             /* offset to strings region descriptor */
    __be64 xip;                 /* offset to xip region descriptor */
    __be64 byte_aligned;        /* offset to the byte aligned region desc */
    __be64 compressed;          /* offset to the compressed region desc */
    ....
    __be64 cblock_offset;       /* offset to cblock offset region desc */
    __be64 inode_file_size;     /* offset to inode file size desc */
    __be64 inode_name_offset; /* offset to inode num_entries region desc */
    __be64 inode_num_entries; /* offset to inode num_entries region desc */
   ....
};
```

AXFS – IMAGE FORMAT (cont)

- Following are some of the important meta-data regions.
  - "Strings region"
  - "Node index region"
  - "Node type region"
  - "inode array index region"

- "Strings region" contains name of the files and is the only meta-data region which is not Byte Table.

- Offset to a filename in the "Strings region" is stored in "inode name offset region"

- "inode array index region" contains the offset into "Node index region" for each page of the file.

AXFS – IMAGE FORMAT (cont)

- Following are the access functions for “Strings” and “inode array index ” meta-data regions.

```
char *axfs_get_inode_name(struct axfs_super *sbi, u64 index)
{
    u64 ofs = axfs_get_inode_name_offset(sbi, index);
    u8 *virt = sbi->strings.virt_addr;

    return (char *)(ofs + virt);
}
```

```
u64 axfs_get_inode_array_index(struct axfs_super *sbi, u64 index)
{
    u64 depth = sbi->inode_array_index.table_byte_depth;
    u8 *vaddr = (u8 *) sbi->inode_array_index.virt_addr;

    return axfs_bytetable_stitch(depth, vaddr, index);
}
```

SONY
make.believe

AXFS – IMAGE FORMAT (cont)

- "Node type region" and "Node index region" contains the type of the node (XIP, compressed or Byte aligned) and index of the data of the node.

- This 'index' is an index into one of the data nodes.

- The type of the node determines which 'data node' is it an index into.

- 'index' for XIP type of node is the page offset into XIP region.

- 'index' for Byte aligned type of node is offset into a Byte Table region which contains offset into Byte aligned region.

- 'index' for Compressed node type is offset into two separate Byte Table region.

SONY
make.believe

AXFS – IMAGE FORMAT (cont)

• The two Byte Table region are “cblock offset region” “cnode offset region” .

• cblock is a block is a block of data that is compressed.

• The uncompressed size of all cblock is same for a file system image and is set by image builder.

• cnode is a data node that will be compressed .

• One or more cnode are combined and then compressed. This compressed node are called cblocks.

SONY
make.believe

AXFS – IMAGE FORMAT (cont)

- cblocks are stored in "compressed region"

- The offset from "cblock offset region" points to cblock in the compressed region.

- The cblock is then uncompressed in system RAM.

- Offset from "cnode offset region" points to the location of cnode in the uncompressed cblock.

AXFS – IMAGE FORMAT (cont)

- Following is the simplified code snippet for using the node index to find the actual data location in a compressed region.

```
….
 node_index = axfs_get_node_index(sbi, array_index);
 node_type = axfs_get_node_type(sbi, array_index);

 if (node_type == Compressed) {
        cnode_offset = axfs_get_cnode_offset(sbi, node_index);
        cnode_index = axfs_get_cnode_index(sbi, node_index);
        ofs = axfs_get_cblock_offset(sbi, cnode_index);
        len = axfs_get_cblock_offset(sbi, cnode_index + 1);
        len -= ofs;
        axfs_copy_data(sb, cblk1, &(sbi->compressed), ofs, len);
        axfs_uncompress_block(cblk0, cblk_size, cblk1, len);
        len = cblk_size - cnode_offset;
        src = (void *)((unsigned long)cblk0 + cnode_offset);
        memcpy(pgdata, src, len);
 } else if (node_type == Byte_Aligned) {
….
```

SONY
make.believe

AXFS – PERFORMANCE

- **Environment :**
  - NEC uPD35001 System-on-chip Evaluation board (NE1-TB) (Also known as NaviEngine board)
    - CPU: ARM11v6 MPCore
    - SMP with 4 cores.
    - Memory : Used 20MB (mem= 20M on kernel command line)
    - L1 I-Cache : 32 KB
    - L1 D-Cache : 32 KB
    - L2 Cache : Not present.
    - Kernel : 3.0
    - NOR flash: 64MB
      - Connected to 16bit AHB (133.33Mhz)
      - Spansion S29GL512N11TFI02 (supports 16-byte page mode read)
  - GCC : 4.5.1 version
  - File system mounted from NOR Flash.

SONY
make.believe

AXFS – PERFORMANCE

- **Performance Parameters :**
  - Application launch (boot) times.
  - Total Flash memory usage.
  - System RAM footprint size.

SONY
make.believe

AXFS + OTHER TOOLS

- We can use other tools or technique in combination with AXFS for further performance improvement.
- One of the techniques tried was about improving code locality.
- Implemented a tool for improving locality.
- Tool records calling order of functions in a program.
- Then a host tool generates a linker script to place functions called together, near each other.

SONY
make.believe

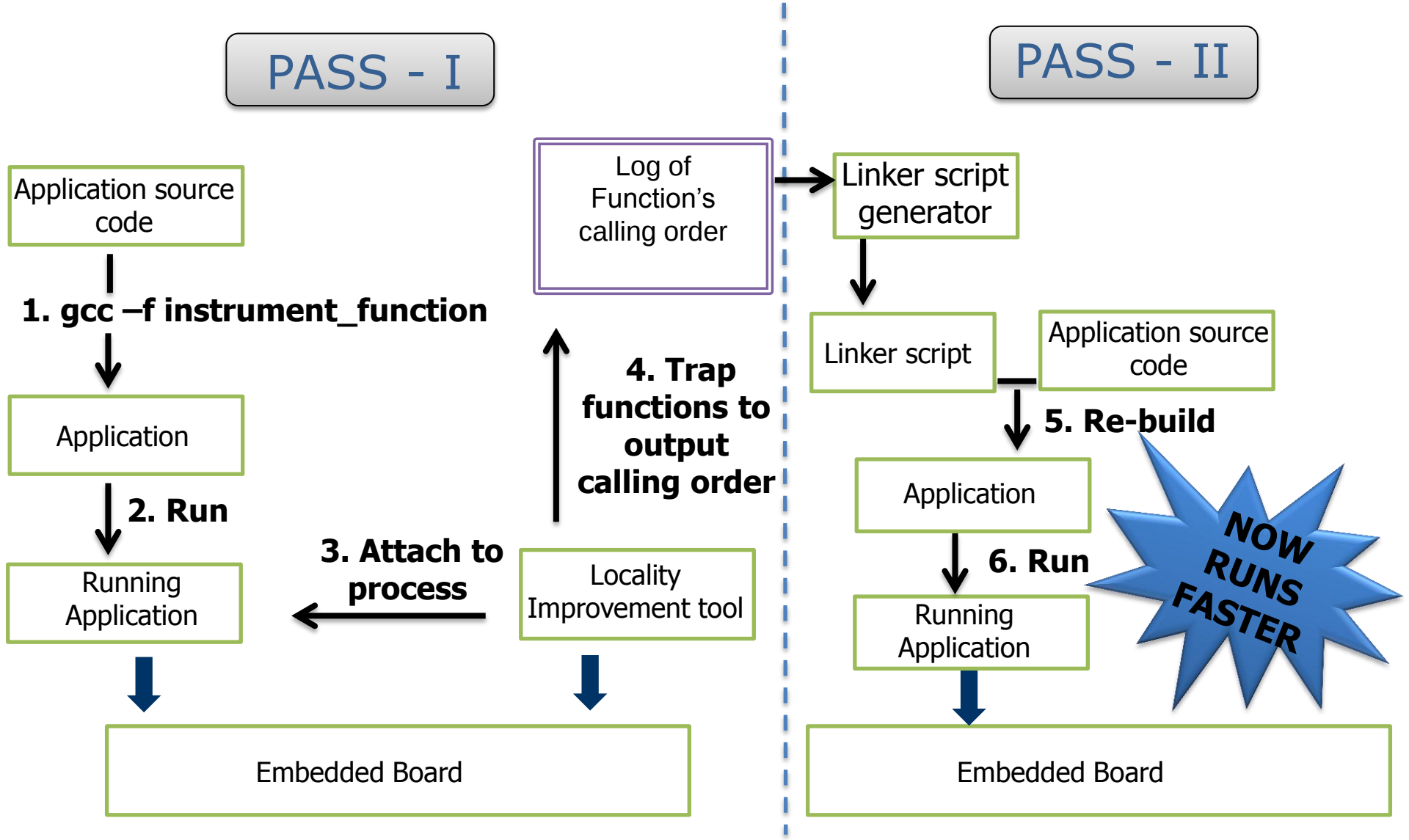
AXFS + OTHER TOOLS (Cont)

- Improving locality reduces the number of page faults when the program is run.

- The reduction in page faults :
  - Improves program/application speed.
  - Reduces system RAM used by program/application.

SONY
make.believe

TOOL FOR LOCALITY IMPROVEMENT



SONY
make.believe

AXFS – PERFORMANCE

- **Application used :**
  - A dummy program which simulates large number of page loading during launch.
  - Calls 500 functions in pseudo-random order.
    - Pseudo-random numbers : Numbers that produce same sequence with same initial state (seed).
  - Each function's size is 1600 bytes.
    - Most instruction are NOPs
    - Few of instructions are for memory access
  - This application used as init process of system.

SONY
make.believe

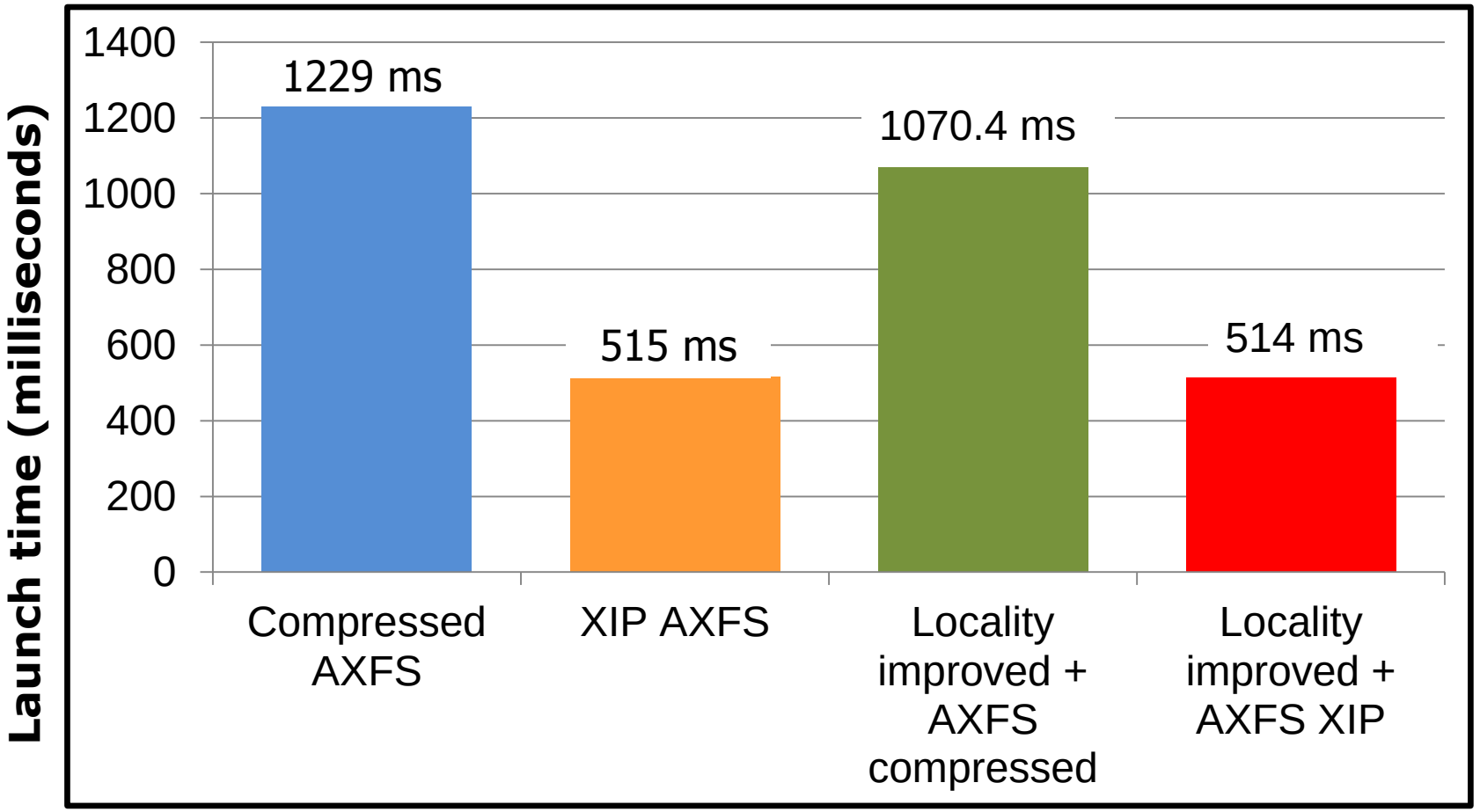
AXFS – PERFORMANCE

- **Application used (cont):**
  - We embed calls to save the current value of a clock into a buffer along with a string.
  - The calls are embedded just before kernel calls init process and just after finishing the execution of 500 functions.
  - We measure the difference of clock time between two calls.

AXFS – PERFORMANCE (Application launch times)

- **Measurement Plan :**
  - Have four AXFS images.
    - AXFS compressed image.
    - AXFS XIP image.
    - Code locality improved binary on AXFS compressed image.
    - Code locality improved binary on AXFS XIP image.
  - Application launch time is difference of the clock values stored:
    - Just before application launch and
    - just after finishing executing 500 dummy functions.
  - We measure and compare launch time for all four FS images.

AXFS – PERFORMANCE
(Application launch times)



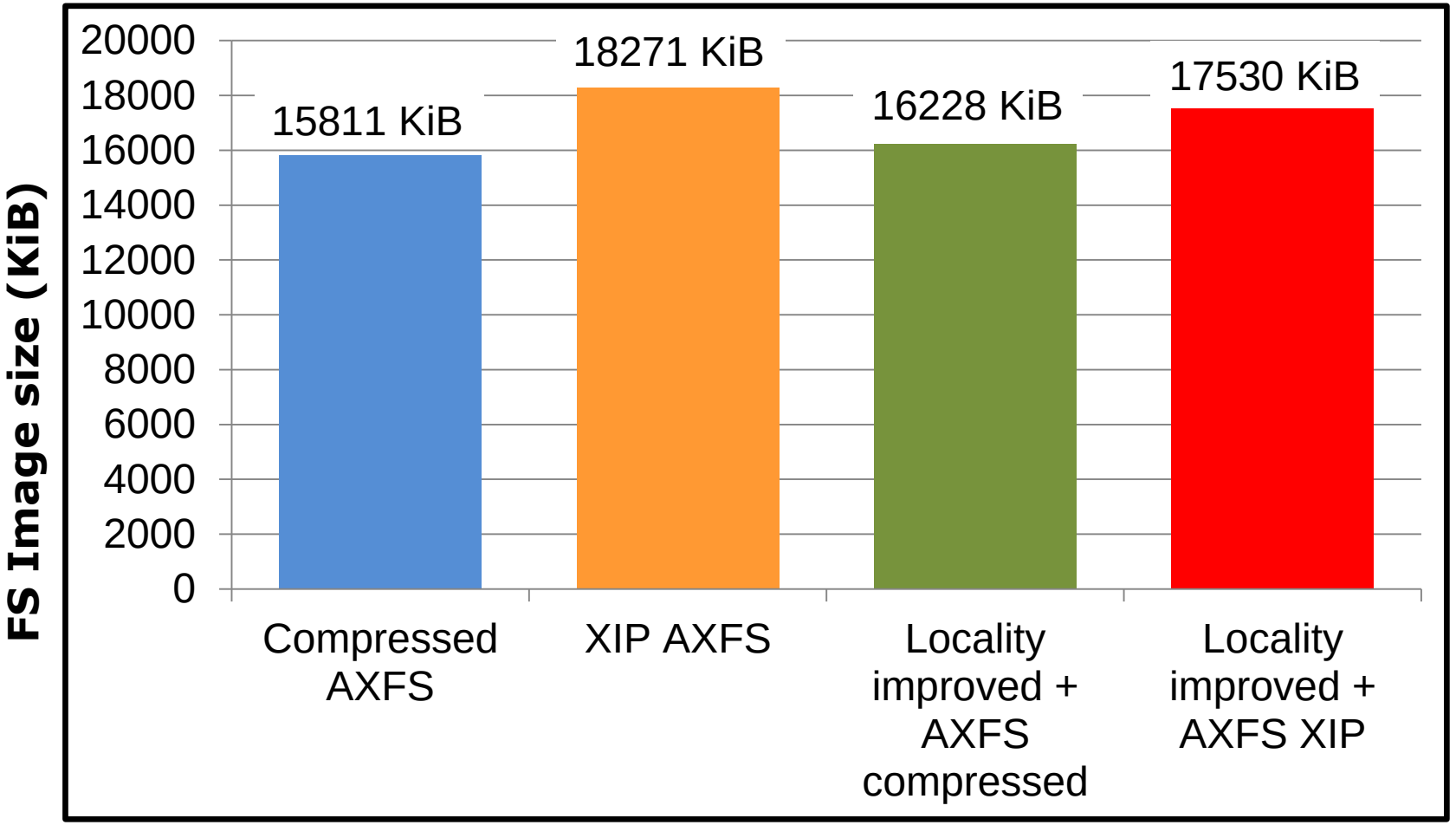
SONY
make.believe

AXFS – PERFORMANCE
(Total Flash memory usage)

- **Measurement Plan :**
  - For the following four AXFS images:
    - AXFS compressed image.
    - AXFS XIP image.
    - Code locality improved binary on AXFS compressed image.
    - Code locality improved binary on AXFS XIP image.
  - Total Flash memory usage is size of the image.
  - We measure and compare size of the images.

AXFS – PERFORMANCE
(Total Flash memory usage)

SONY
make.believe



AXFS – PERFORMANCE
(System RAM footprint)

SONY
make.believe

- **Measurement Plan :**
  - For the following four AXFS images:
    - AXFS compressed image.
    - AXFS XIP image.
    - Code locality improved binary on AXFS compressed image.
    - Code locality improved binary on AXFS XIP image.

  - **System RAM footprint (Actual definition):**
    - Total memory used in the system by the running the application.

  - **System RAM footprint (What we mean here):**
    - Total memory used in the system by virtue of running the application.

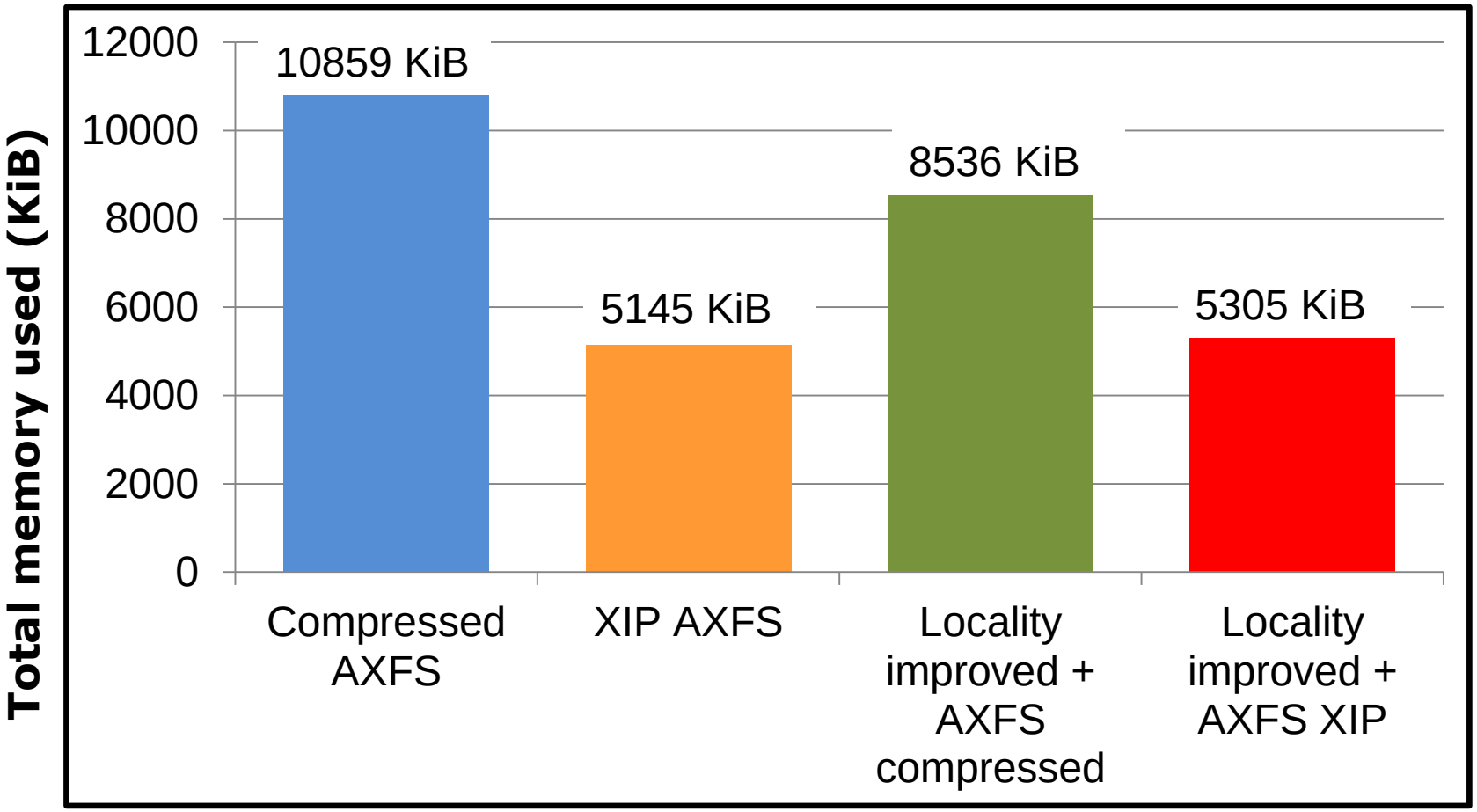
SONY
make.believe

AXFS – PERFORMANCE (System RAM footprint)

- We measure total memory used in the system while running our application.

- Fair , as our application is the only process created in the system while it runs.

AXFS – PERFORMANCE
(System RAM footprint)



SONY
make.believe

Summary

- Key point of AXFS : It allows XIP at page granularity.
- XIP AXFS requires a profiling pass.
- AXFS image can span multiple devices.
- Can mount on MTD layer instead of block layer.
- Suitable for code mostly used during boot / application launch.
- Not suitable for hotspots in the system.

SONY
make.believe

Questions ?

SONY
make.believe

Thank You.