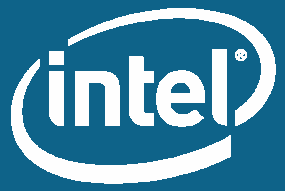


Creating optimized XIP systems

CELF ELC 2006

Jared Hulbert

Justin Treon

Agenda

Part A - Why create optimized XIP systems?

- What is XIP?
- How can XIP:
  - Perform as good as SnD?
  - Perform better than SnD?
  - Save battery life?
  - Save RAM?
  - Save money?

Part B - How do you create an optimized XIP system?

Glossary

XIP = eXecute In Place (next slide explains more)

SnD = Store and Download. Default mode of operation for Linux. Requires code is stored on Flash or a disk and downloaded to RAM before executing

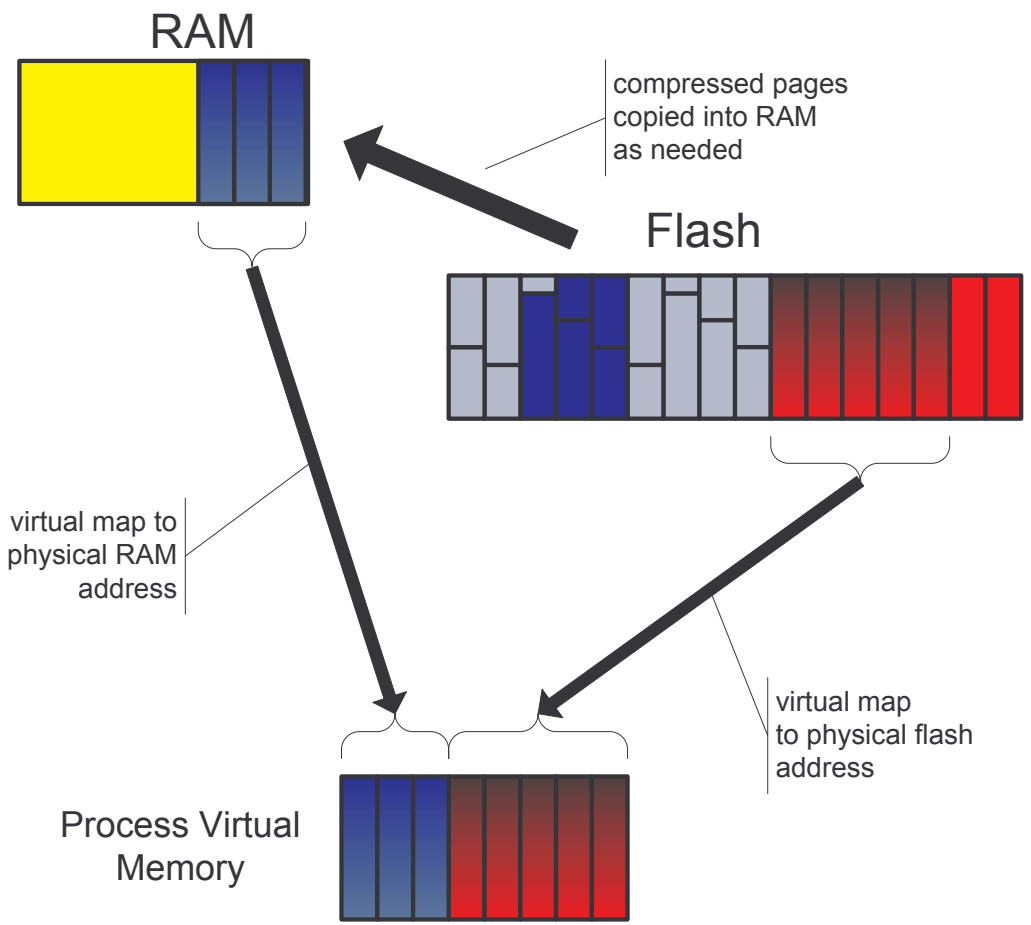
Demand Paging = Refers to the dynamic process the kernel uses to get pages of code or data from a file system as needed

What is XIP

Code is run directly from non-volatile memory, NOT copied to RAM

NOR Flash and ROM can XIP – short RAM like reads [~100ns]

NAND Flash can not XIP – Long sector based reads [~20us]



XIP = Executing directly from NOR flash

intel

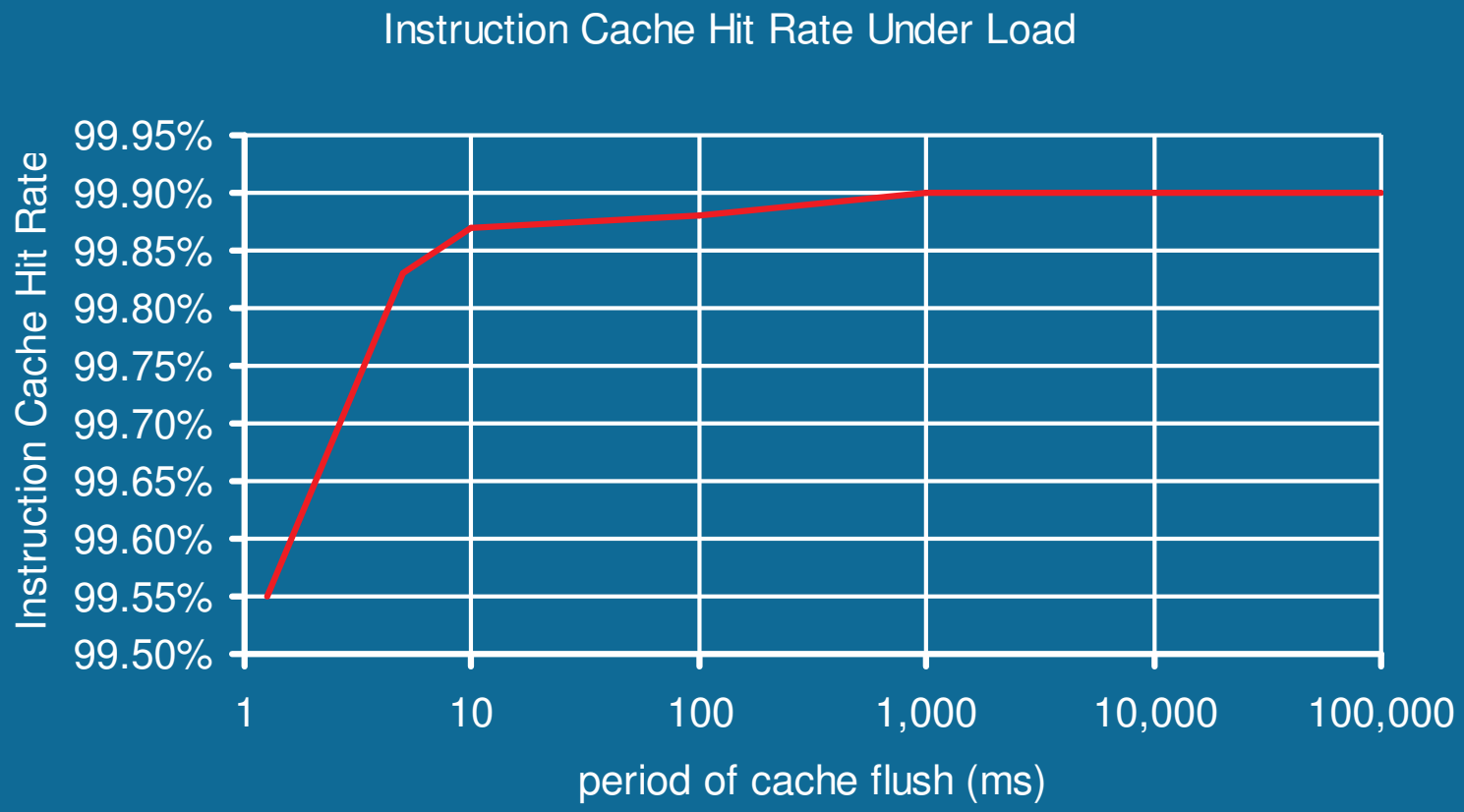
History and present

- ~1999 - Agenda VR Linux PDA
  - Linear XIP cramfs patches
  - Kernel XIP
- ~2002 - MontaVista 3.0
- 2005 - 2.6.10 mainlined XIP for arm architecture
- 2005 – XIP Linux cell phones in Japan
- 2005 - XIP aware MTD
- 2005 - COW Cramfs patches
- 2006 – AXFS: Advanced XIP File System

Higher Performance

Performance

- But NOR Flash is slower than RAM!!!



- Processor Instruction caches hide difference
- High instruction cache hit rates

Caches equalize RAM / NOR code performance

Performance
- Boot and application launch

SnD slower due to NAND read speed and demand paging overhead

SnD even slower without Erase Block Summary turned on in JFFS2

	Boot	Quake	Browser	Media Player
XIP	32.2	20.0	2.6	2.3
SnD	71.6	26.7	3.9	3.4
SnD slower by	**2.2X**	**1.3X**	**1.5X**	**1.5X**

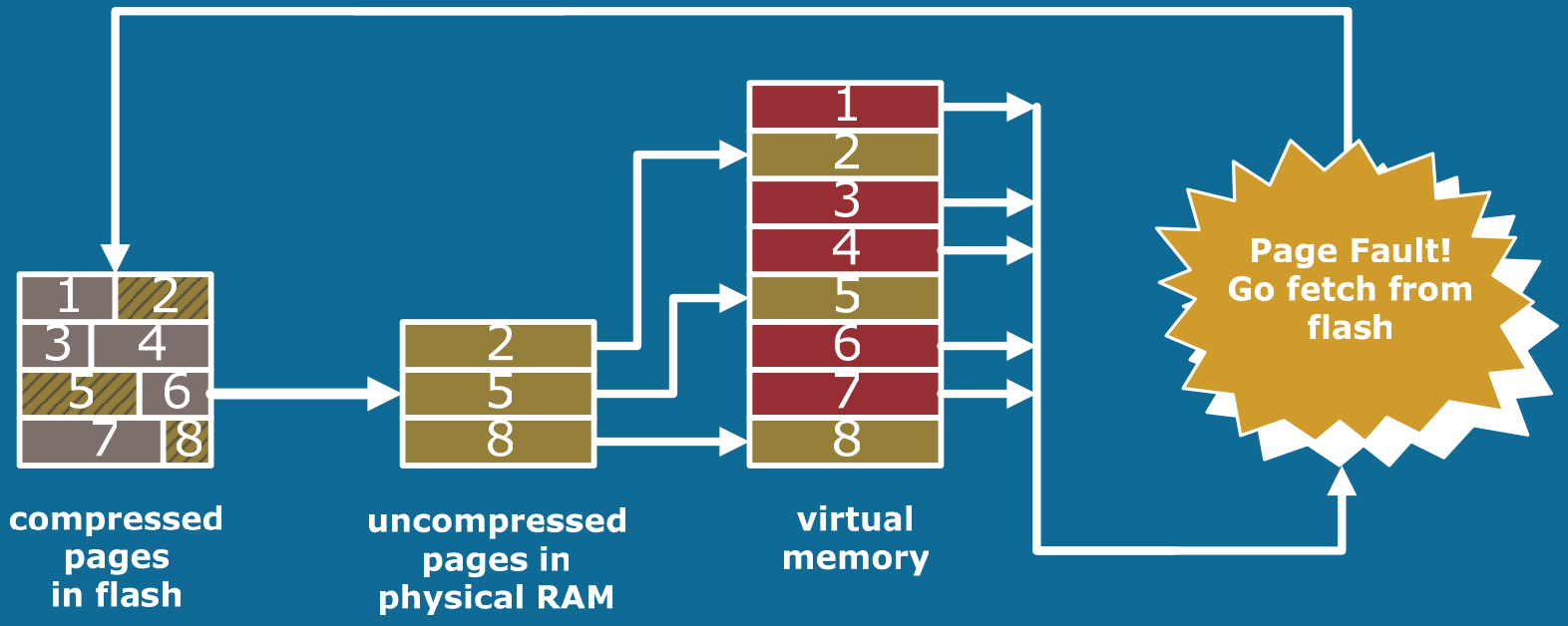
XIP is faster starting applications and booting

Demand paging

Pages in virtual memory but not in physical RAM cause page fault when accessed

Typically won't have enough RAM for all the code

Limited page pool is reused as different code pages are swapped in and out



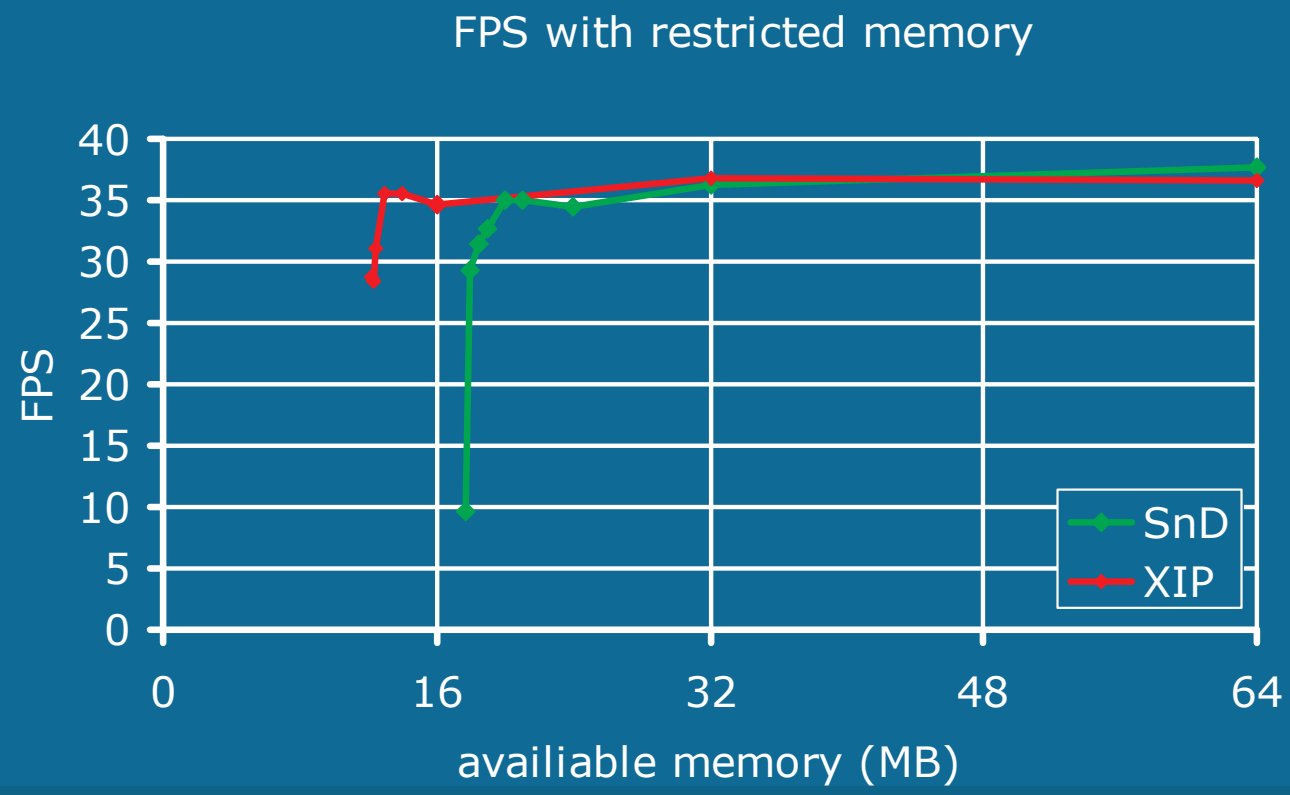
Adds complexity to performance measurements

Performance

Restricting memory shows impact of demand paging on performance

XIP is faster unless excessive RAM memory is available for paging

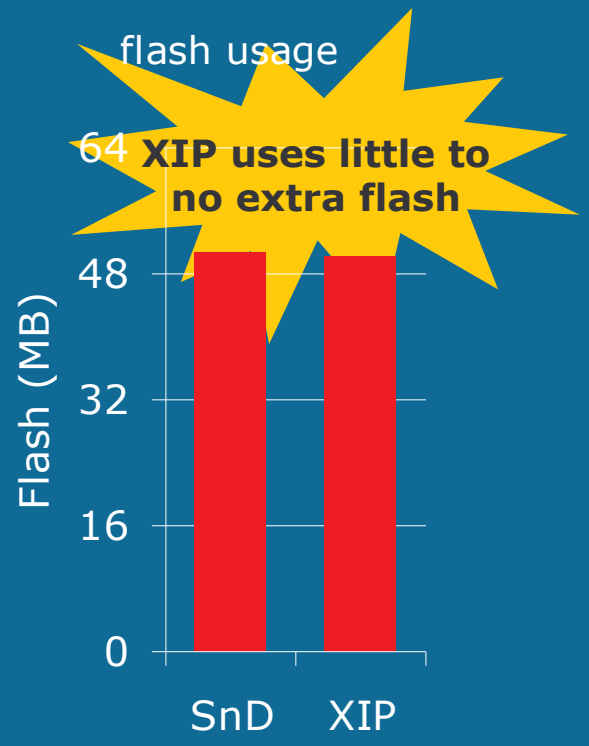
The SnD performance falls off quickly once the page cache is restricted

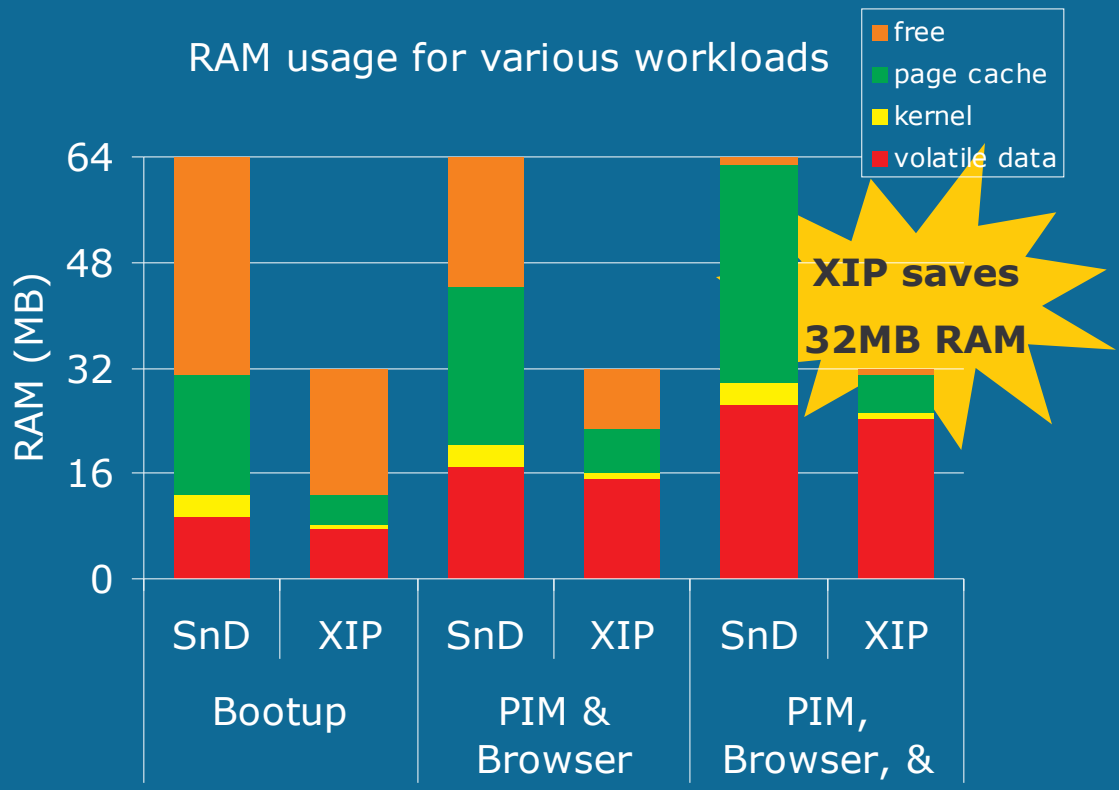


SnD can’t run this workload with 16MB, XIP can

Lower RAM Density

Flash / RAM usage





With Summary support on JFFS2 SnD uses more flash than XIP

XIP saves 32MB RAM without using extra flash

How does it save RAM

Biggest component of RAM savings is code in the page cache

Saving RAM in a SnD system = lower performance

Demand paged systems require code to be in flash and in RAM

Libqte.so = ~3 MB uncompressed = ~1.5MB compressed

	Flash	RAM
SnD	**1.5** MB	**3** MB
XIP	**3** MB	**0** MB
	1.5 MB **Extra flash for XIP**	**3** MB **Extra RAM for SnD**

1MB more XIP code = 2MB less RAM

Longer Battery Life

Standby Power

More RAM = More power

- SDRAM power proportional to array size (bits)
- Significant while in low power standby

Experiment

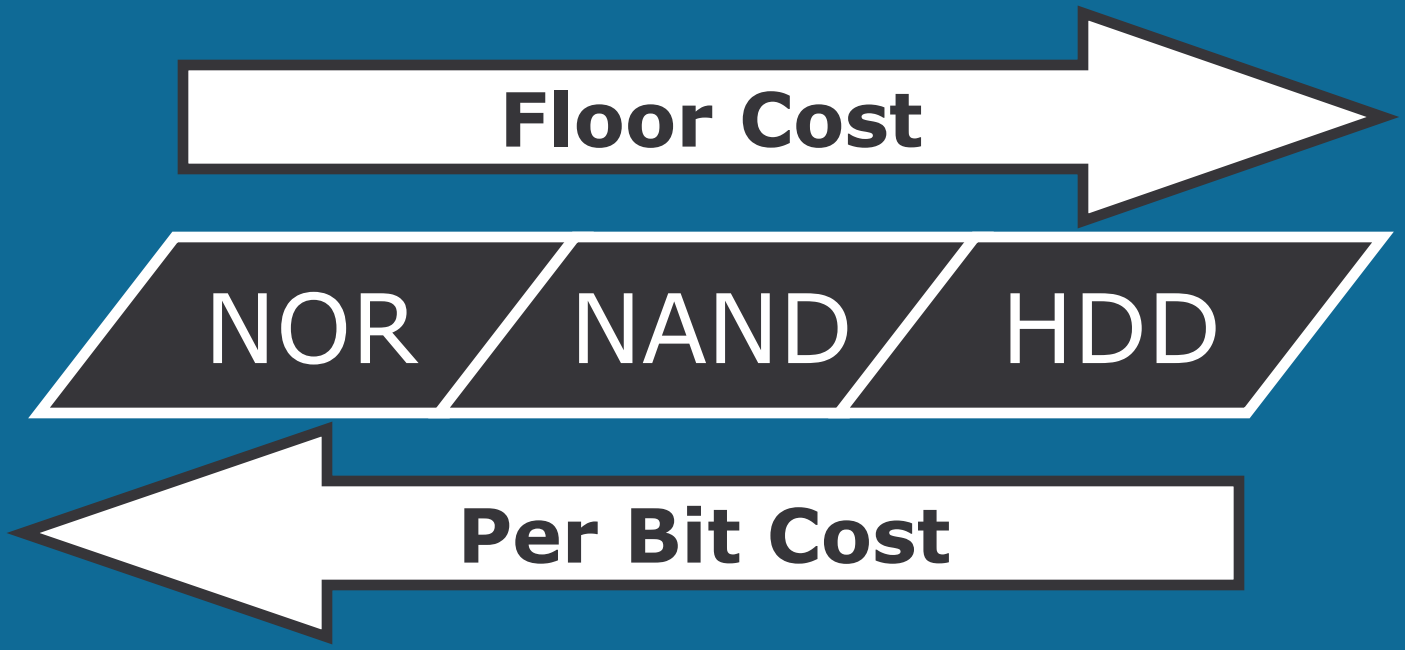
- Measured current used by handsets during standby mode
- Determined current required for different memory subsystems
  - SnD = 64MB LPSDRAM & 64MB NAND
  - XIP = 32MB LPSDRAM & 64MB NOR
- XIP system saves:
  - 8.4% of standby energy
  - 2.1 days of standby battery

Lower RAM can mean extra days of standby

Lower Cost

Memory Tradeoffs

Floor Costs = Minimum cost for medium



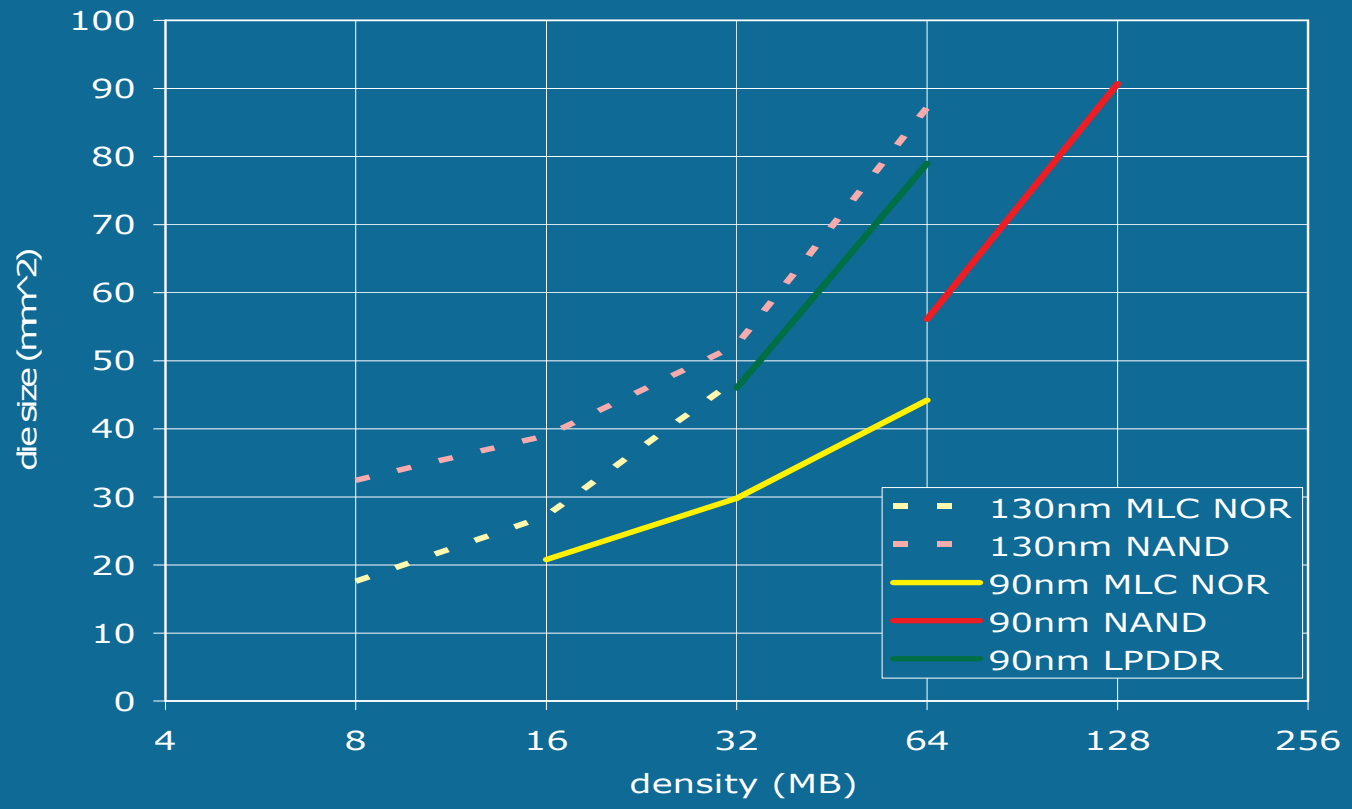
Per Bit Cost = Overhead in medium

Technologies makes sense at different densities

Sweet spots

RAM and NAND are more efficient at higher density, less efficient at lower densities

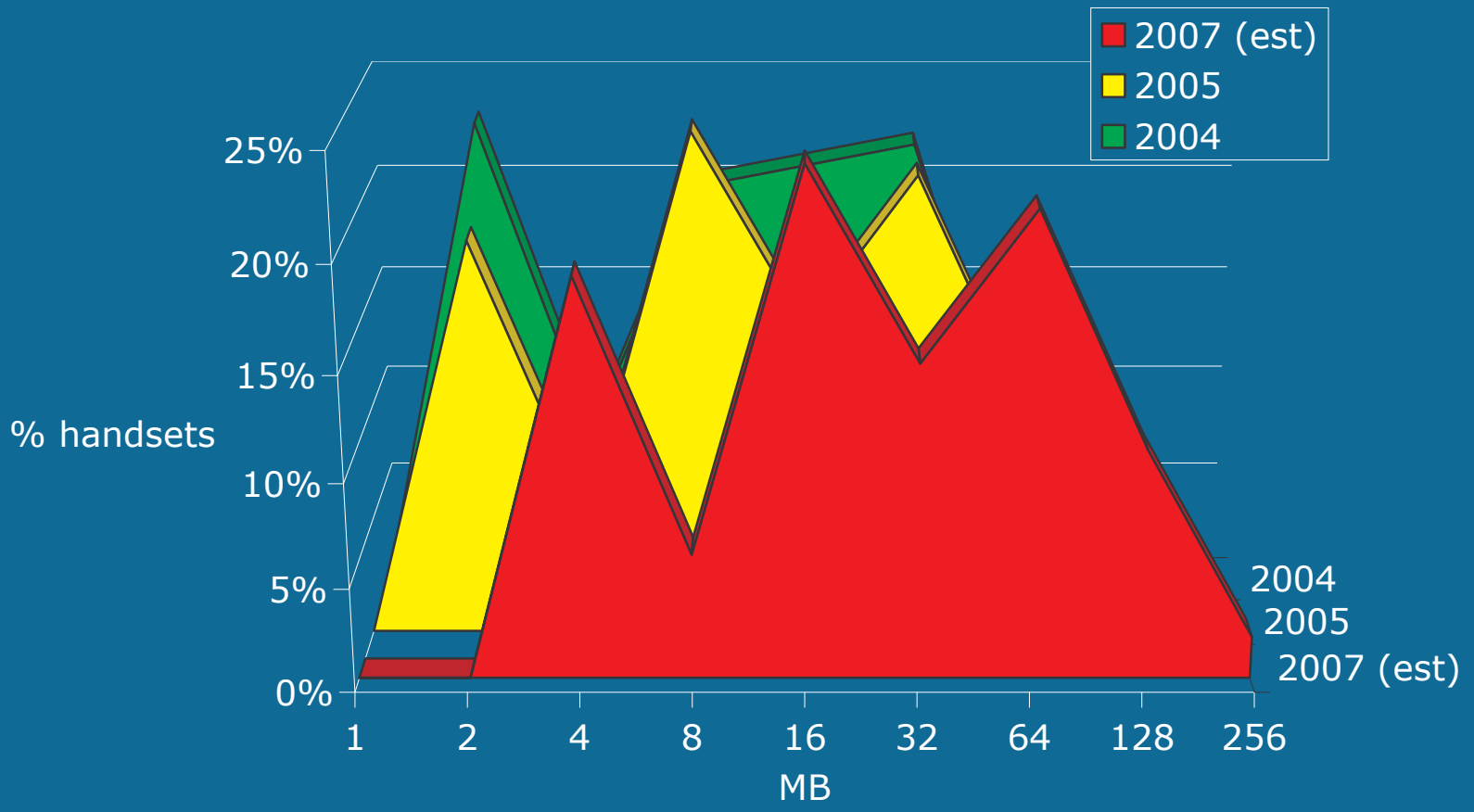
NOR more efficient at the lower density, less efficient at higher densities



NOR die is smaller than NAND at low densities

intel

Cell Phone Requirements

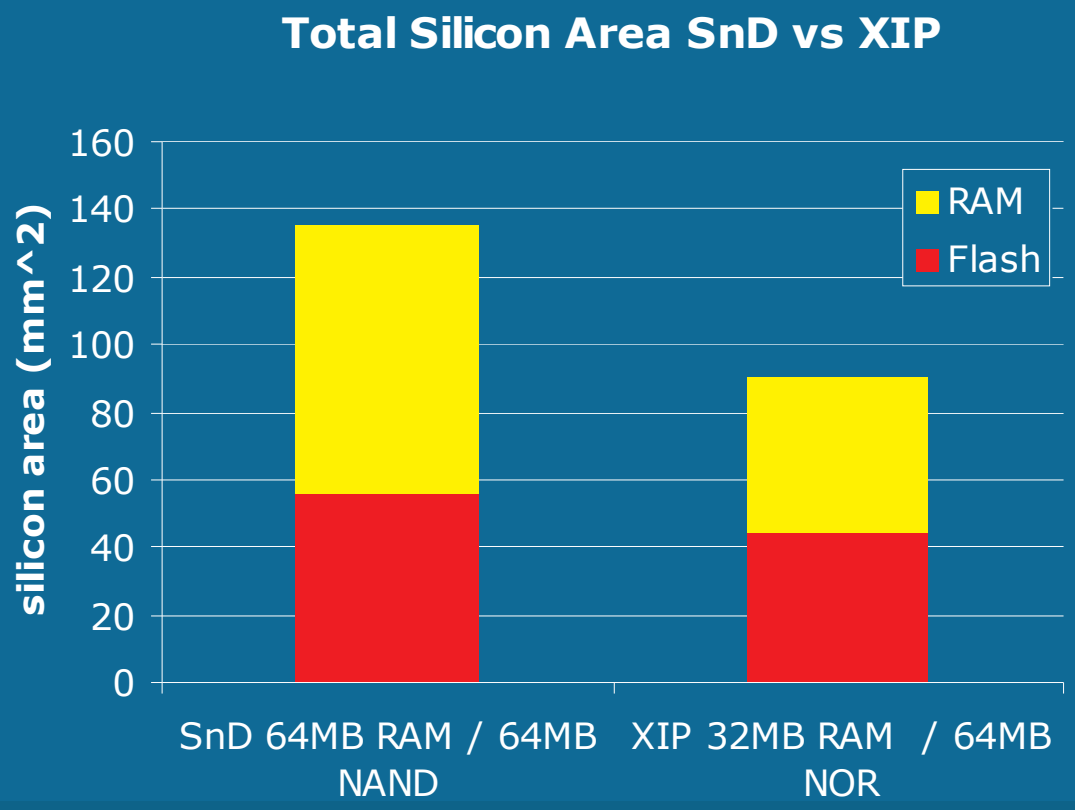


Cellular requirements fit in XIP NOR sweet spot

Total silicon area

Assume two designs:

- SnD = 64MB of RAM and 64MB NAND
- XIP = 32MB of RAM and 64MB NOR



If design is in NOR sweet spot, XIP saves $$

Optimizing an XIP system

Goals

1. Have code XIP for quick launching
2. Save RAM for power and cost
3. Use as little extra flash as possible
4. Fit system into single die densities
   (32MB or 64MB not 48MB)

CRAMFS

CRAMFS

- Began as a read only filing system for bootable floppies
- Originally written by Linus Torvalds
- Various patches to support Linear XIP CRAMFS
- A patch for 2.6.14 is available on the CELF website

To XIP applications

- Executable binaries are marked as XIP with the “sticky bit” unix permission bit
- To use less RAM with fast application launch time choosing the right files to XIP is crucial

Find code

Find which files in the system are being mapped as code

- Files mapped using mmap() save RAM
- ramust – RAM usage scan tool
  - http://sourceforge.net/projects/ramust
  - Reports how much RAM is being used and how it is being used
  - Uses libproc to search the /proc filesystem
  - Outputs summary in tab delimited form

Selecting files to XIP

- Run Ramust on the development platform to see what files have been memory mapped on the system

```
./ramust –f
```

  - Only memory mapped file can run XIP
- Obtain the memory mapped file list for several use cases
  - Use models
    - All Personal Information Management applications
    - All Games
    - Media Viewers and Players
    - Browsers
- Ignore all entries to deleted temporary files and files under /dev

XIP code for often used application

Find other data & play with possibilities

Find other files that you want to XIP

- cfsst – compressed file system sizing tool
  - http://sourceforge.net/projects/cfsst
  - Helps find good candidate files for being stored XIP
  - Explores changes to filesystem image size
  - Outputs is tab delimited (uncompressed size, compressed size, type, etc)

Selecting additional files

Look through the list for additional files that should not be compressed using cfsst on you host system

Select files that:

- Get bigger when compressed
- Are already compressed
- Are used at boot time, but not shown in Ramust

Creating the Linear CRAMFS

Add the sticky bit to files you wish to XIP

```
chmod +t FILENAMES
```

When you have added the sticky bit to all the files you will XIP create the filing system with the mkfs.cramfs tool with the Linear CRAMFS mkfs.cramfs patch applied

Create the Linear CRAMFS

```
mkfs.cramfs -x rootfs rootfs.bin
```

Configuring the Kernel for XIP support

Kernel Execute-In-Place from ROM

Set the kernel to make an xipImage and the root filing system to CRAMFS

Boot options

Default Kernel command string:

Add:

root=/dev/null rootflags=physaddr=0xXXXX

rootfstype=cramfs

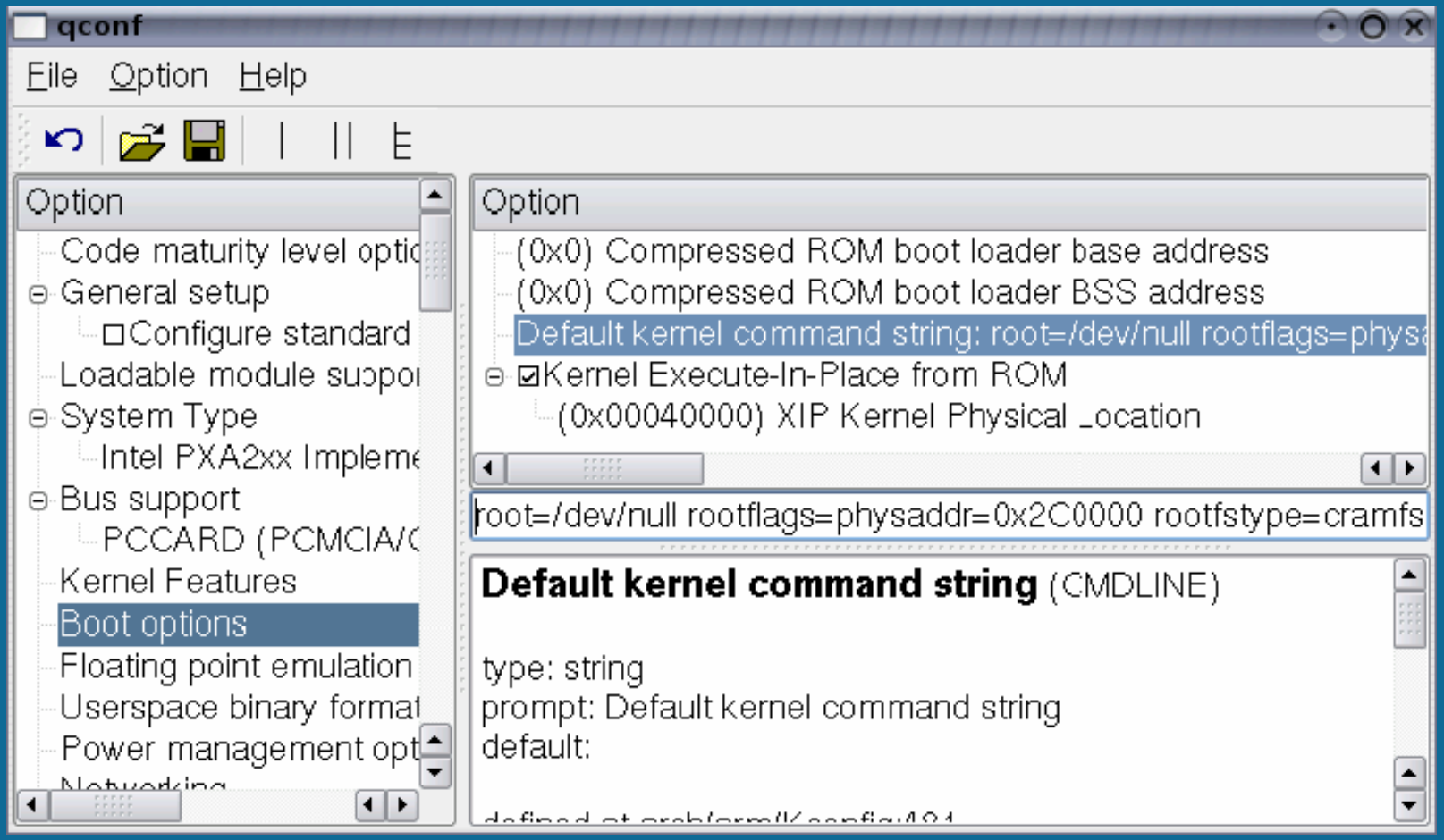
XXXX is the address of your CRAMFS (Use upper case)

Remove the previous root and rootsftype options

→ Kernel Execute-In-Place from ROM

Set the address to match that of the kernels location in FLASH

Kernel Execute-In-Place from ROM



XIP aware MTD support

The XIP aware MTD support option allows an XIP kernel to reside anywhere flash rather than on an 8MB boundary

Build in XIP aware MTD support

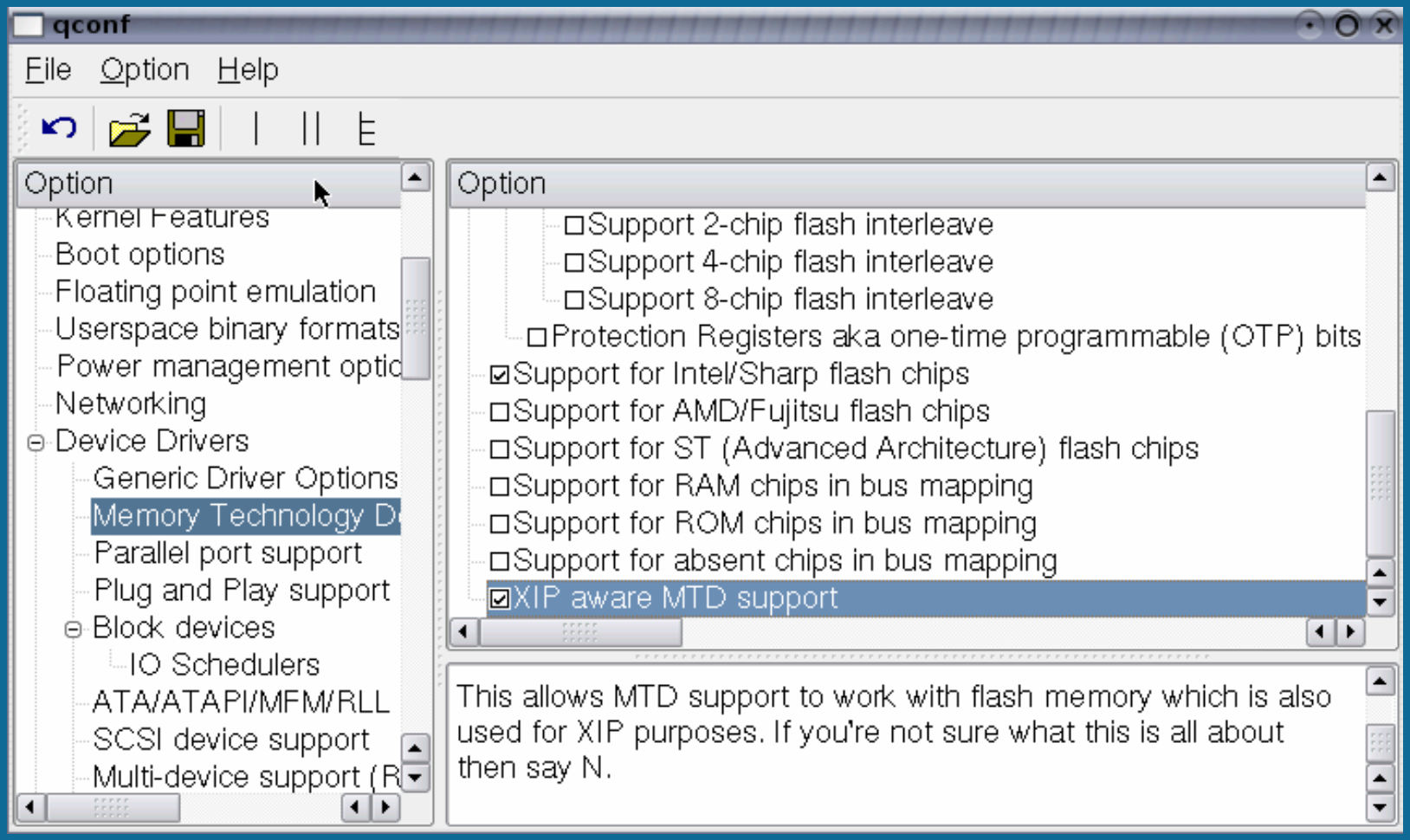
Device Drivers

→ Memory Technology Devices

—→ RAM/ROM/FLASH chip drivers

——→ XIP aware MTD support

XIP aware MTD support



Enabling the kernel for Linear XIP CRAMFS

Once you have applied the Linear XIP CRAMFS patch to your kernel you can build in Linear XIP CRAMFS support

Build in Cramfs root and XIP support

File Systems

→ Miscellaneous filing systems

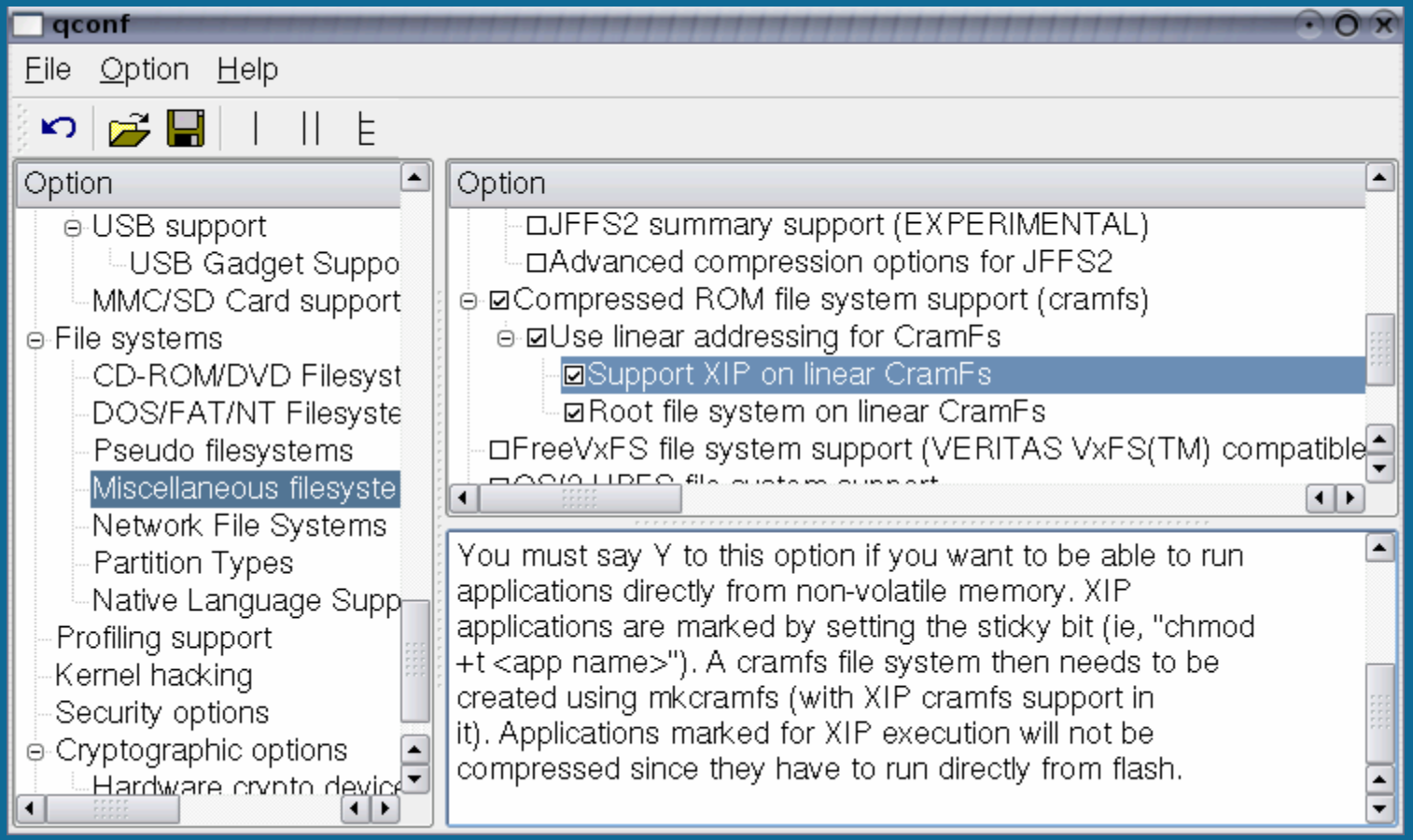
—→ Compressed ROM file system support (cramfs)

——→ Use linear addressing for CramFs

———→ Support XIP on linear CramFs

———→ Root file system on linear CramFs

Enabling the kernel for Linear XIP CRAMFS



Building the kernel

Once you have saved the changes you can build an xipImage

```
make xipImage
```

AXFS

AXFS – Advanced XIP file system

- http://sourceforge.net/projects/axfs
- New project started by Intel
- Replacement for Linear CRAMFS
- Read only file system for Linux
- Only uncompress pages that contribute to RAM savings
- Will include tools to identify pages that should be uncompressed
- **Hitting alpha very soon**