

# Kdump: towards fast and stable crash dumping on terabyte- scale memory system

HATAYAMA, Daisuke  
FUJITSU LIMITED.  
LinuxCon Japan 2013

# Terabyte-scale memory system

## ■ Common on enterprise-class model line-up

Company	Model	Max Memory Size
SGI	SGI UV 2000	64TiB
HP	HP ProLiant DL980 G7	4TiB
IBM	System x3950 X5	3TiB
Fujitsu	PRIMEQUEST 1800E	2TiB

## ■ Use cases

- In-memory database
- VM consolidation
- HPC

# Kdump issues on terabyte-scale memory system

- Collecting crash dump takes several hours without special handling
  - Dump filtering
    - reduce size of crash dump – create “mini dump”
- Design and implementation had no longer scaled
  - Even dump filtering could take tens of minutes
  - Crash dump could fail due to too much memory consumption
- Still, full dump is necessary for Mission Critical use

# What this talk explains

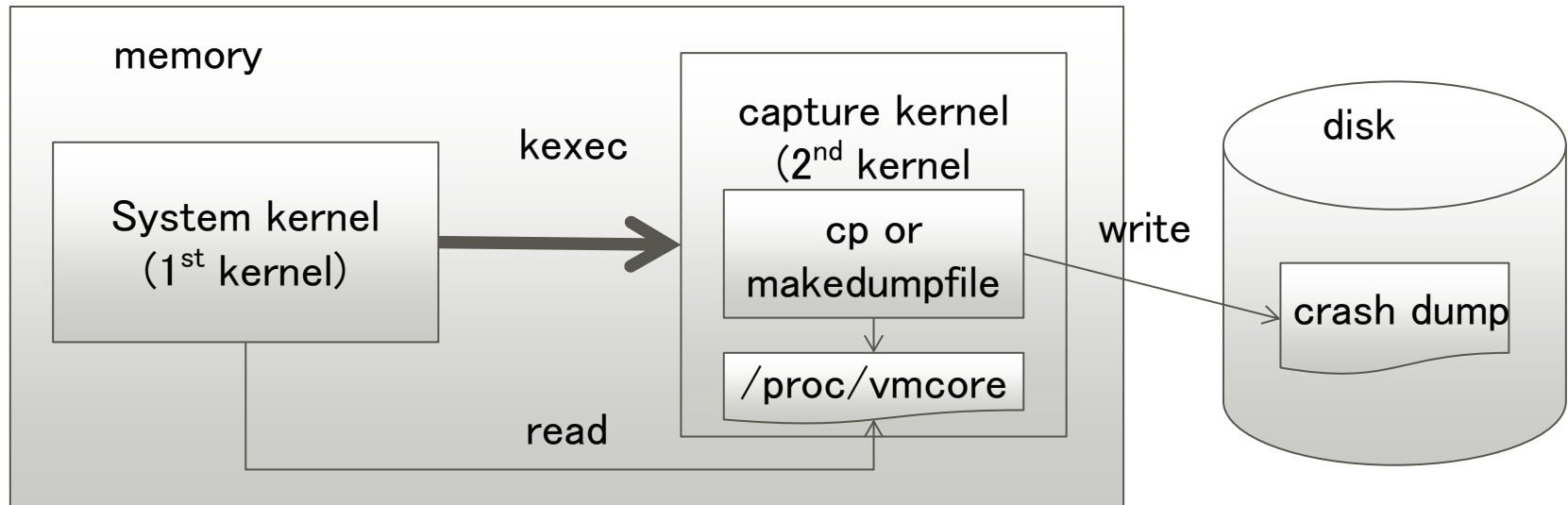
- How several issues on creating “mini dump” were resolved since last year
  - → For most users who think “mini dump” is enough and they don’t require rigorous failure analysis
- How can we collect “full dump” in a reasonable time and what challenges still there are
  - → For users who think “full dump” is essential and require rigorous failure analysis on mission critical use

- Review kdump framework
- Recently fixed issues for “mini dump”
  - Memory consumption issue
  - Dump filtering performance degradation issue
- Ongoing and future work for “full dump”
  - Towards fast crash dumping for “full dump”

# Review kdump framework

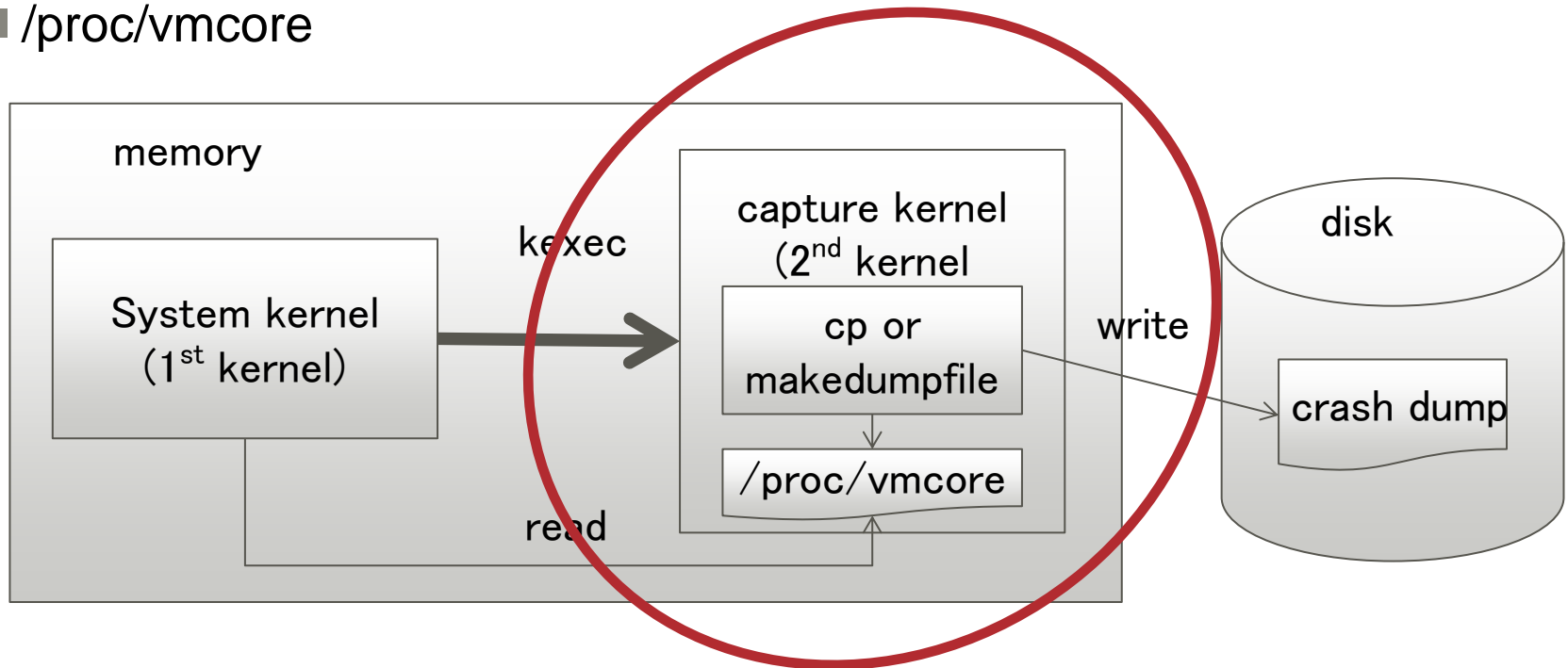
# Kdump: kexec-based crash dumping

- Linux kernel standard crash dump feature
- Merged at 2.6.13
- Components related to crash dumping
  - Capture kernel
  - makedumpfile
  - /proc/vmcore



# Kdump: kexec-based crash dumping

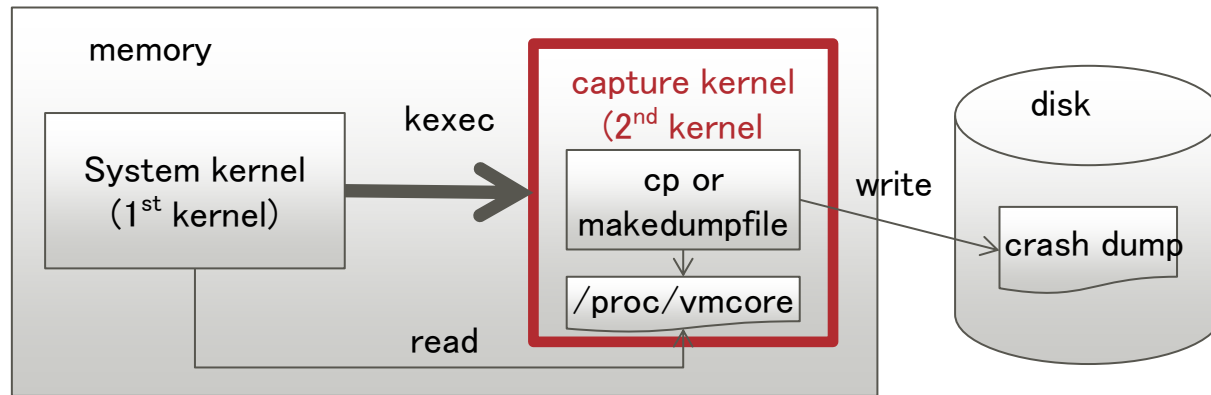
- Linux kernel standard crash dump feature
- Merged at 2.6.13
- Components related to crash dumping
  - Capture kernel
  - makedumpfile
  - /proc/vmcore





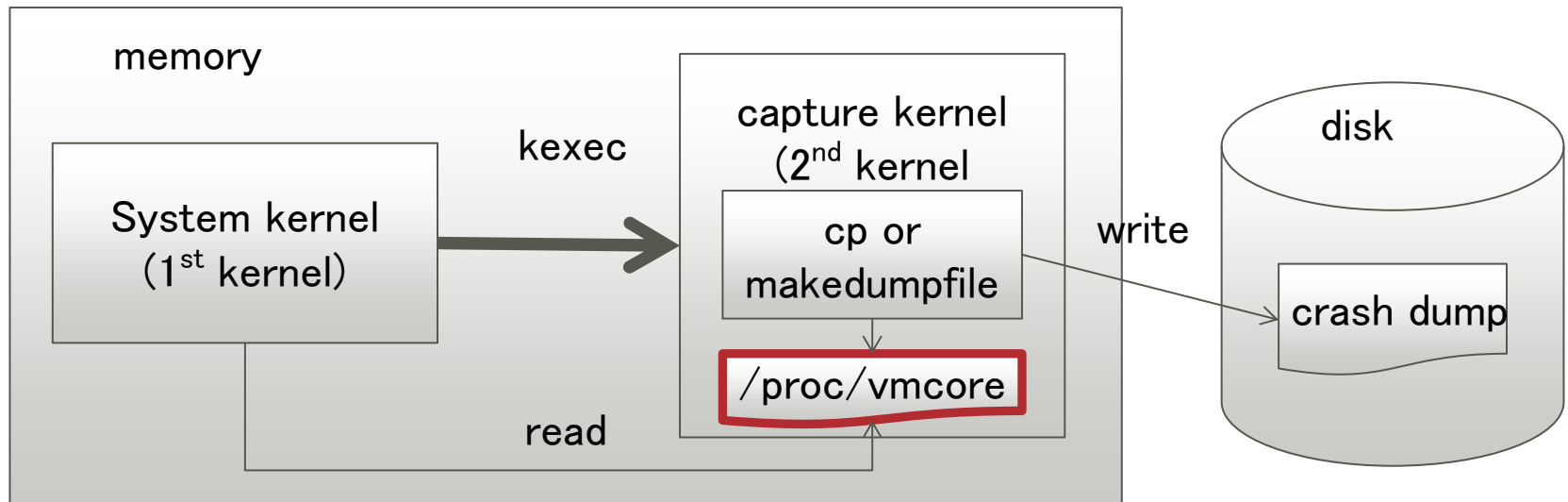
# Capture kernel

- Kernel booted from system kernel
- Running on the memory reserved at system kernel
  - crashkernel=<memory size>
    - 128MiB ~ 512MiB
- Has /proc/vmcore



## ■ File interface for capture kernel to access system kernel as ELF binary format

- Use user-space tools for copying crash dump
  - Make some additional processing if necessary
- `cp /proc/vmcore /mnt/dump`



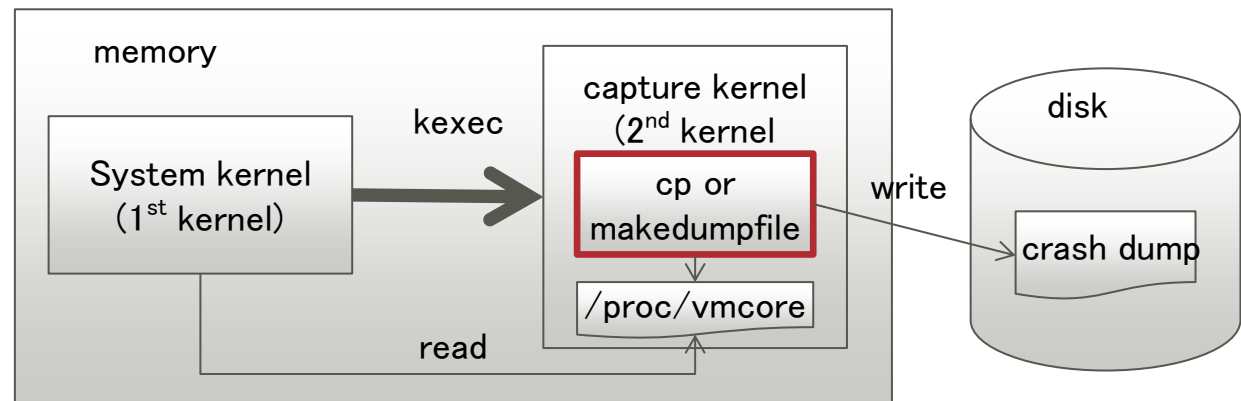
# makedumpfile

## ■ Copy vmcore with some additional processing

### ■ Compression

### ■ Dump filtering --- create mini dump

- Free pages
- Page cache
- Page cache (meta data)
- User data
- Zero page



## Recently fixed issues for “mini dump”

- Memory consumption issue
- Dump filtering performance degradation issue

## Recently fixed issues for “mini dump”

- Memory consumption issue
- Dump filtering performance degradation issue

# Capture kernel has severe memory limitation

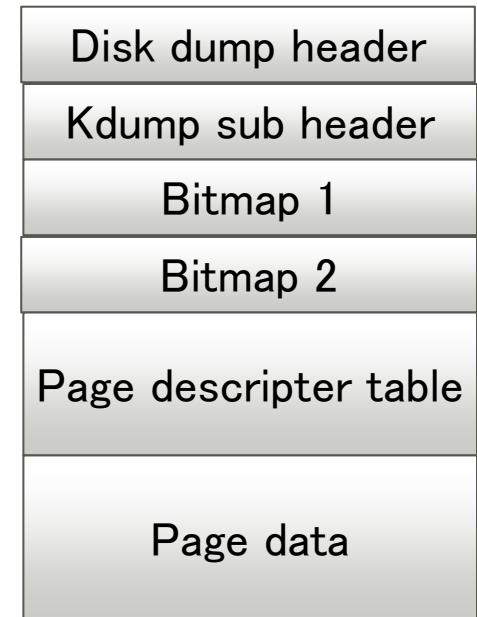
- Reserved memory: 512 MiB
- Cannot assume other disks
  - Continue working in ramdisk
    - System kernel's rootfs can be broken
  - On network configuration dump device is not connected
    - `scp /proc/vmcore user@192.168.122.22:/mnt/dump`
- ➔ important to use memory efficiently

## ■ Problem

- Makedumpfile had consumed much memory on terabyte-scale memory system
- Killed by OOM killer, failing to collect crash dump

## ■ Cause

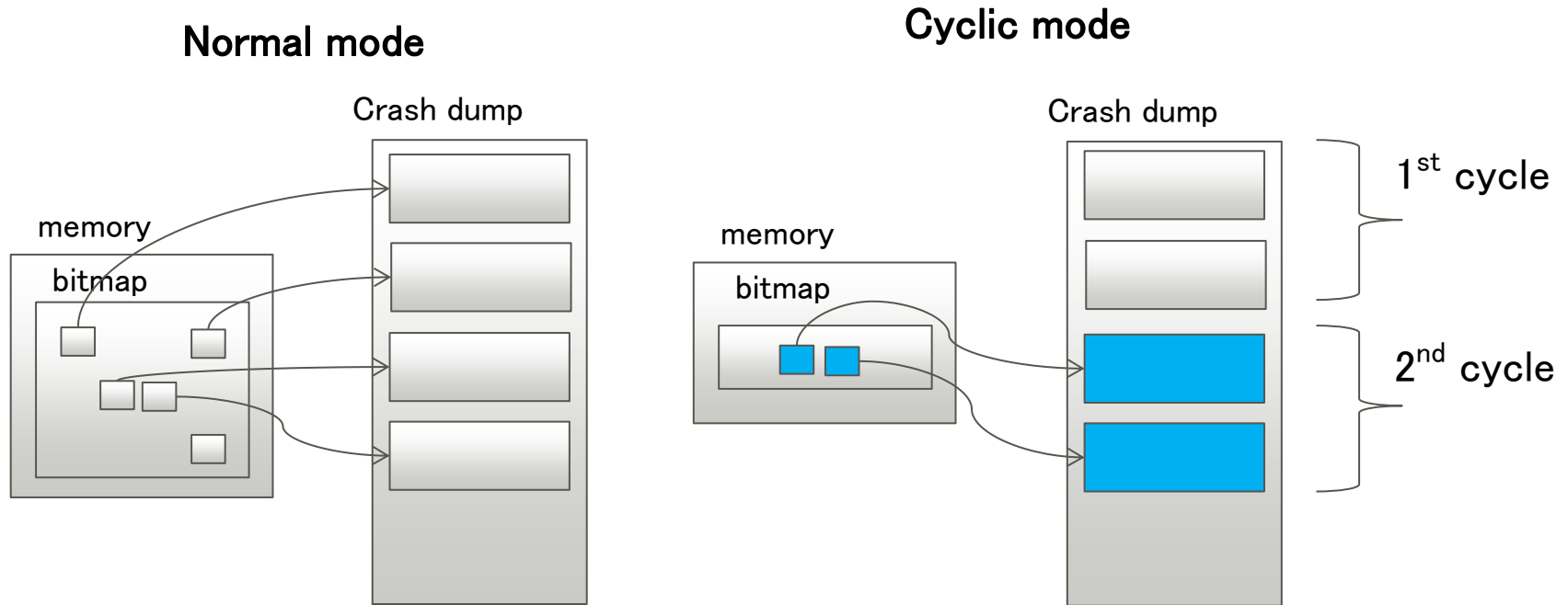
- Makedumpfile's dump format has two bitmaps that grow in proportion to system memory size
  - 1TiB memory  $\Leftrightarrow$  32 MiB bitmap
- Two bitmaps are allocated at a time on memory
  - On 8TiB, bitmap size exceeds size of reserved memory
  - $2 * 32 * 8 = 512\text{MiB}$



# Solution: divide process into cycles

■ If N-cycles, bitmap size on memory is reduced to  $1/N$ .

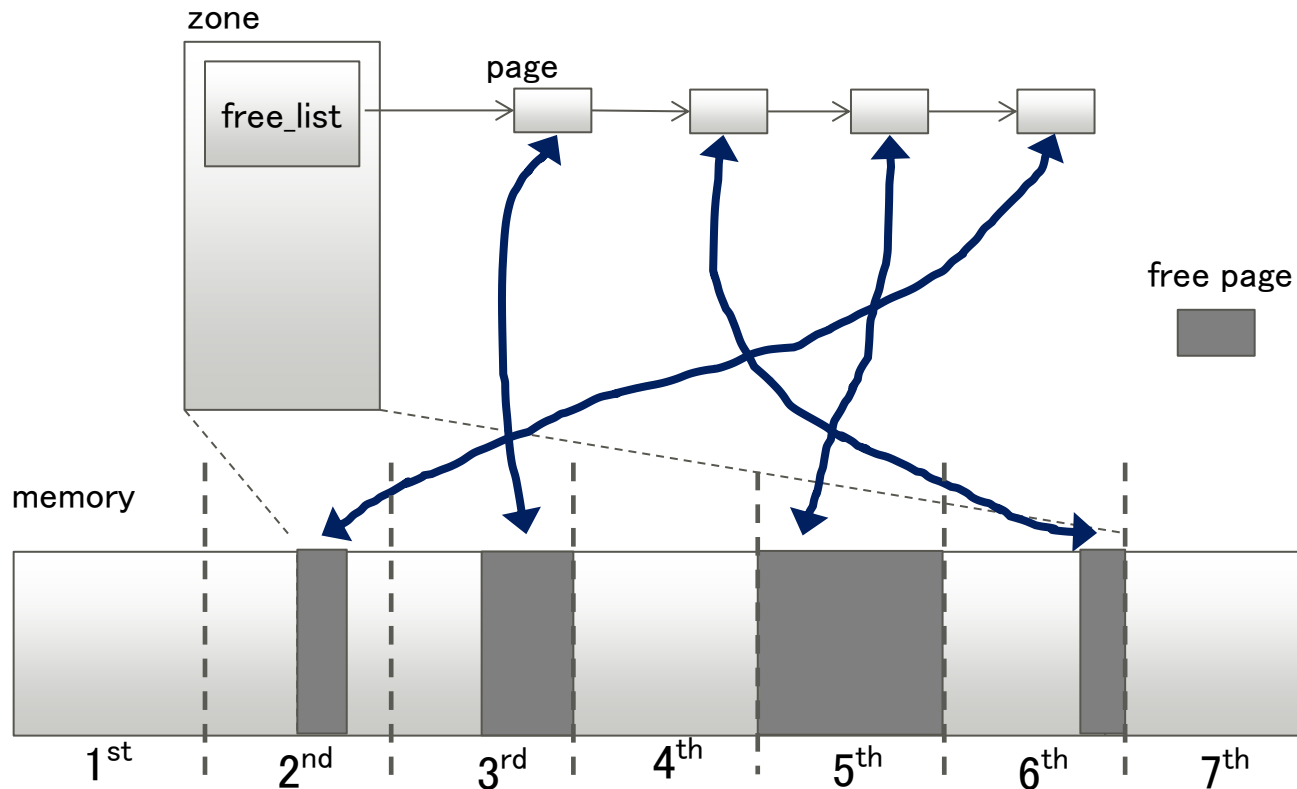
- Repeat the following processing N-cycles:
  - Create bitmap
  - Write the corresponding pages in crash dump





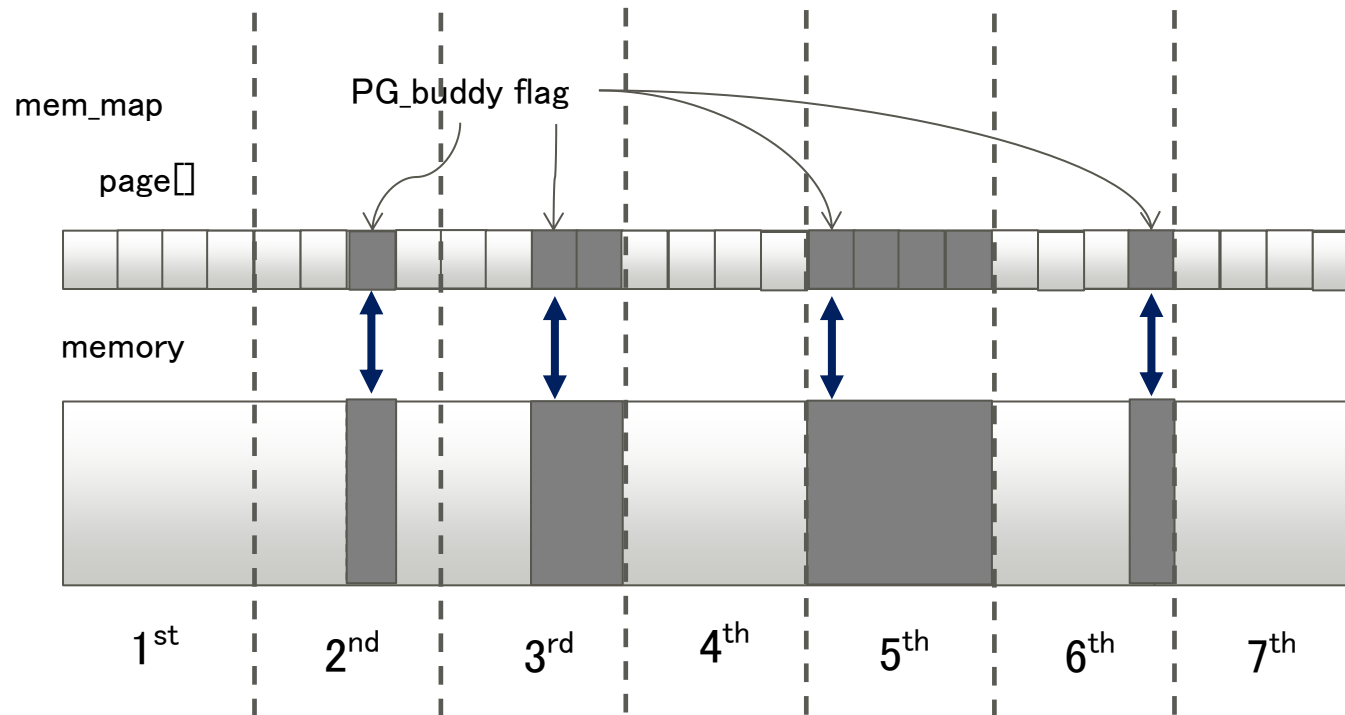
# free\_list cannot be divided into cycles

- All the processing in makedumpfile needs to be divided into cycles
- Unfortunately, current method of filtering free pages cannot be divided into cycles
  - Free\_list is not sorted: linear search costs  $O(\text{memory size}^2)$



# Look up mem\_map for free page filtering

- Like other memory types, looking up mem\_map array (page descriptors) in case of free pages
- Each page descriptor is sorted w.r.t. page frame numbers



# Further merits of mem\_map array logic

- More robust against data corruption
  - If free\_list is corrupted, cannot follow anymore.
  - Infinite loop at worst case.

## ■ Supported versions

- v1.5.0: Cyclic-mode, by Atsushi Kumagai

- **default mode**

- v1.5.1: New free pages filtering logic, by HATAYAMA

## ■ Usage

- --cyclic-buffer <memory in kilobytes>

- 80% of free memory is specified at default

- --non-cyclic

- Switch back to the original mode

# Support for above-4GiB bzImage load

- Yinghai Lu at kernel v3.9
- There is no longer 896MiB limit
- Kernel parameter
  - `crashkernel=size[KMG],high`
  - `crashkernel=size[KMG],low`
- Each component needs to support new bzImage protocol
  - System kernel: v3.9~
  - Kexec: v2.0.4~
  - Capture kernel: v3.9~

## Recently fixed issues for “mini dump”

- Memory consumption issue
- Dump filtering performance degradation issue

# Dump filtering performance degradation issue

- Dump filtering time becomes no longer ignorable on terabyte-scale memory

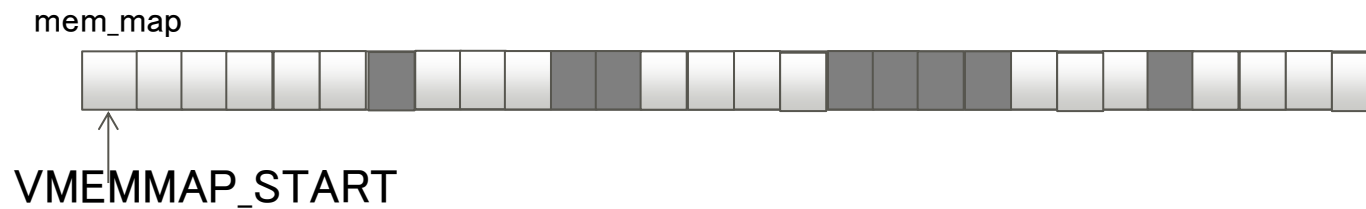
Memory: 2TiB

makedumpfile	kernel	Dump filter (s)	Copy data (s)	Total (s)
1.5.1	3.7.0	470.14	1269.25	1739.39

- 27% of Total is used for dump filtering
- Causes are in
  - makedumpfile
  - /proc/vmcore

# makedumpfile: paging was bottleneck

- Problem: on 4-level paging, reads /proc/vmcore 5 times
  - 4-level page walk + target memory access
  - mem\_map is mapped in VMEMMAP area, not direct mapping area



## ■ Solution: add cache for /proc/vmcore

- By Petr Tesarik
- Keep previously read 8 pages
  - Reduce the number of reads to /proc/vmcore for page table access
- Simple but very effective



## ■ Problem:

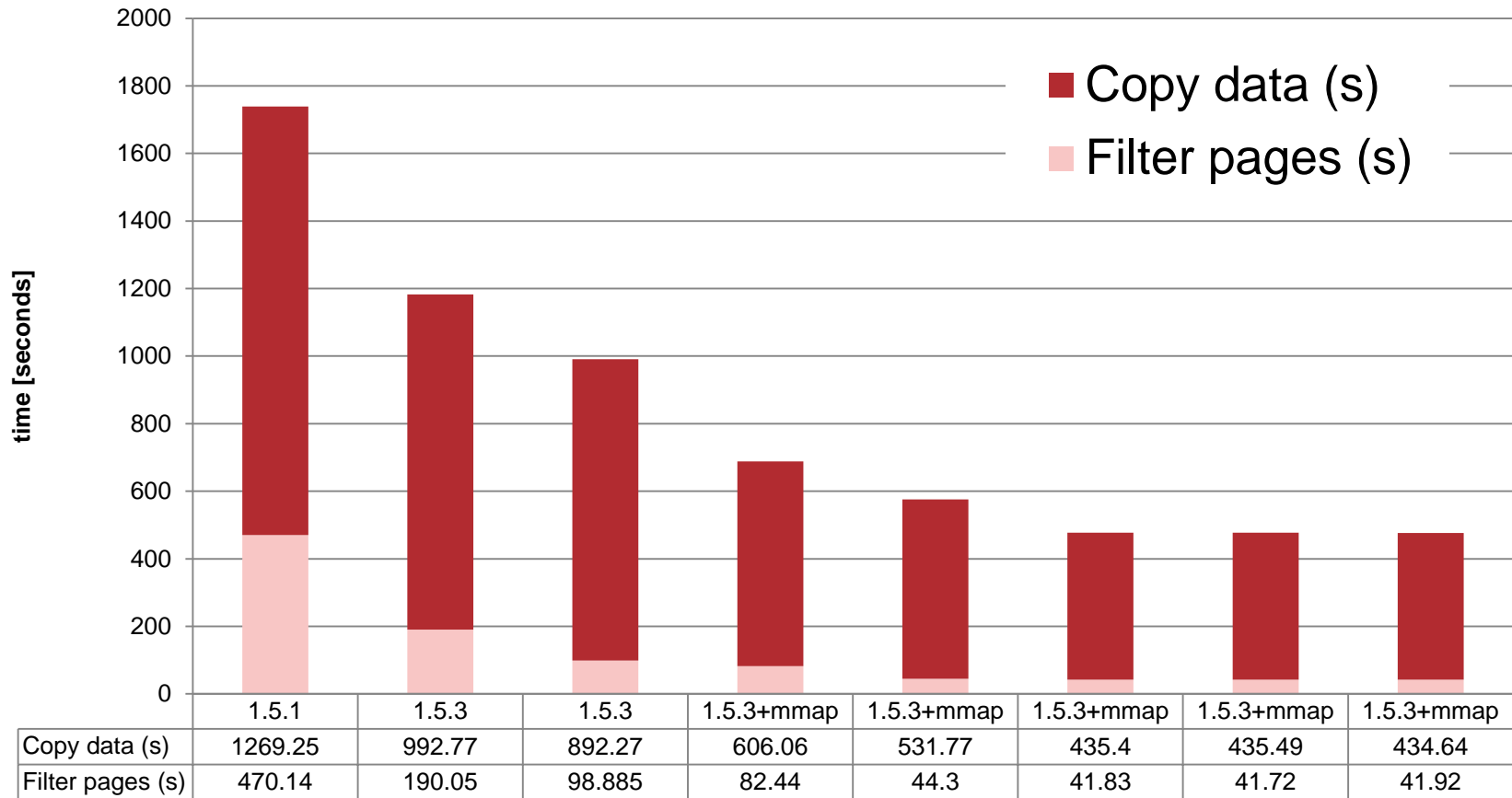
- ioremap/ionunmap is called per a single page
- TLB flush is called on each ionunmap call

## ■ Solution: Support mmap() on /proc/vmcore

- Reduce the number of TLB flush by large mapping size
  - Both mem\_map for dump filtering and page frames are consecutive data
- No copy from kernel-space to user-space

# Filtering Benchmark (2TB, 1CPU) [1/2]

■ Smaller is better



Contributed by Jingbai Ma (<https://lkml.org/lkml/2013/3/27/19>)

- Two experimental kernel-side filtering work
  - Cliff's 240 seconds on 8TiB memory system  
<http://lists.infradead.org/pipermail/kexec/2012-November/007177.html>
  - Ma's 17.50 seconds on 1TB memory system  
<https://lkml.org/lkml/2013/3/7/275>
- Rough comparison

	Filter pages [sec/TiB]	
Cliff	30	Kernel-side at capture kernel
Ma	18	Kernel-side at system kernel
This work	21	User-side at capture kernel

## ■ Kernel

- Mmap support on /proc/vmcore, v3.10-rc2 mmotm by HATAYAMA

## ■ makedumpfile

- Cache for /proc/vmcore, v1.5.2, by Petr Tesarik
- Mmap support, v1.5.3-devel, by Atsushi Kumagai

# Ongoing and future work for “full dump”

# Importance of full crash dump

- Bug on modern system is complicated
- Failure analysis by experienced developers with full dump is a must
  - dump filtering could lost important data
- E.g. qemu/KVM system
  - Communication between guests and host
  - Virtualization assist features
    - Transparent hugepage, compaction
  - → full dump including qemu guests' images
  - → view from guests helps failure analysis

# Toward fast crash dumping

- Goal: Complete **2TiB** full crash dump within an hour

- How:

  - Fast compression algorithm

    - Fast enough even if data is not reduced at all

  - Data striping like RAID0

    - Write data concurrently into multiple disks

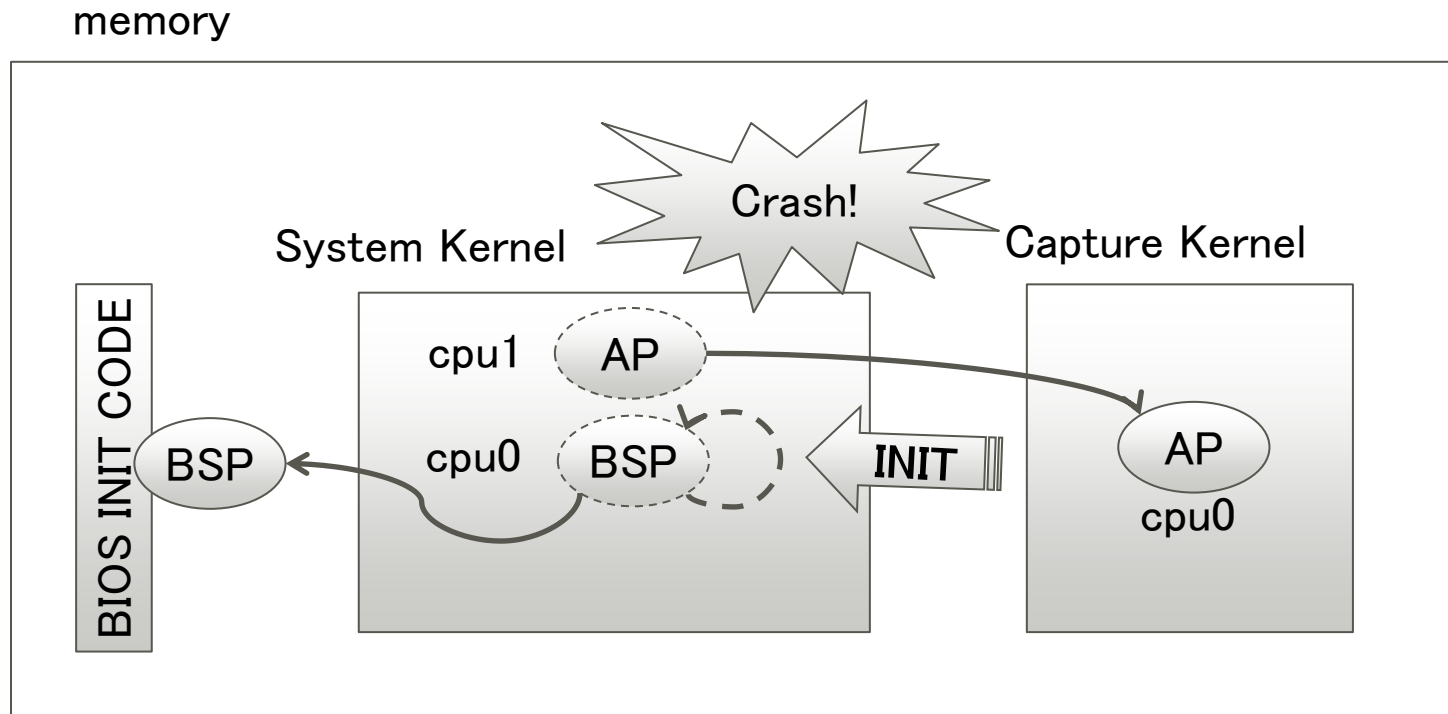
- Problem

  - On x86 capture kernel cannot use multiple CPUs now

    - Limited to 1 at default
      - maxcpus=1

# x86 boot protocol issue

- If BSP receives INIT IPI, then BSP jumps at BIOS init code
  - System hang or system reset





# Experimental benchmark

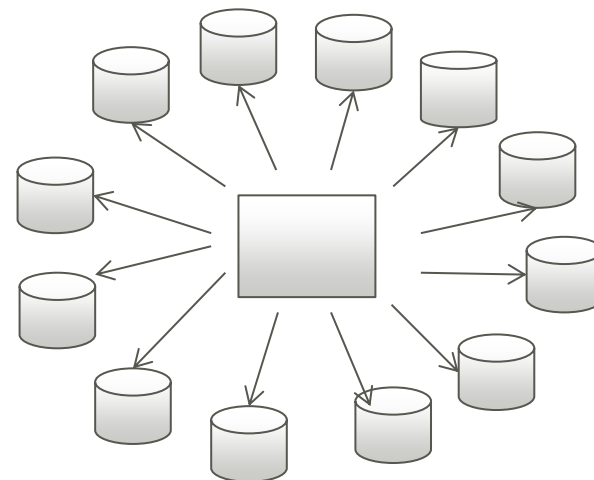
## ■ Purpose: see performance under the configuration with

- fast compression: LZO, snappy
- parallel I/O
- **multiple CPUs**

## ■ How

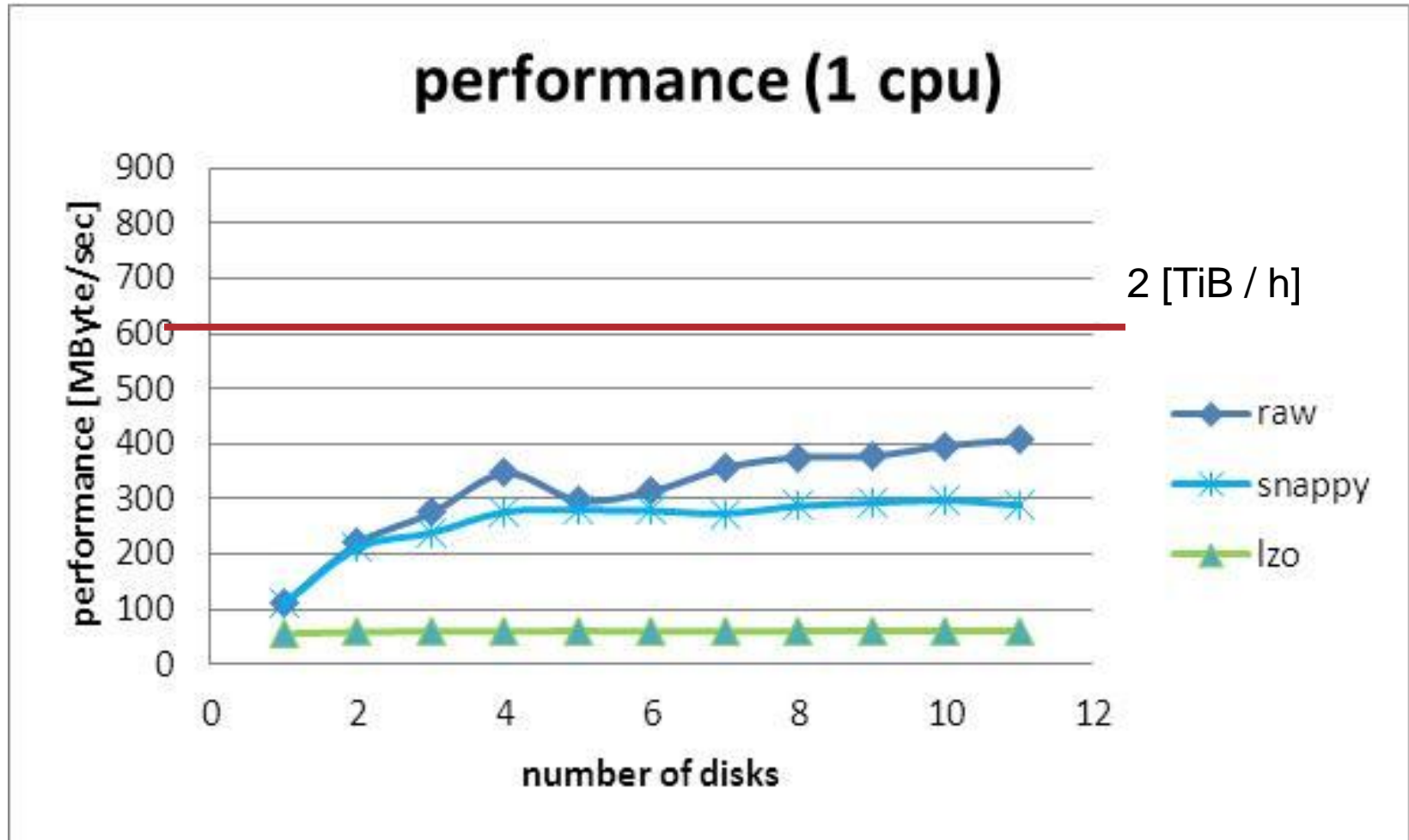
- Process 10GiB data in **system kernel**
- The data is randomized not to be reduced by compression
  - To see if fast compression can be used for free even at worst case

OS	RHEL 6.3
Kernel	2.6.32-279.el6.x86_64
CPU	Intel® Xeon® CPU E7540@2.00GHz
System	PRIMEQUEST1800E
Disk	(147GiB, 6 Gb/s) x 12



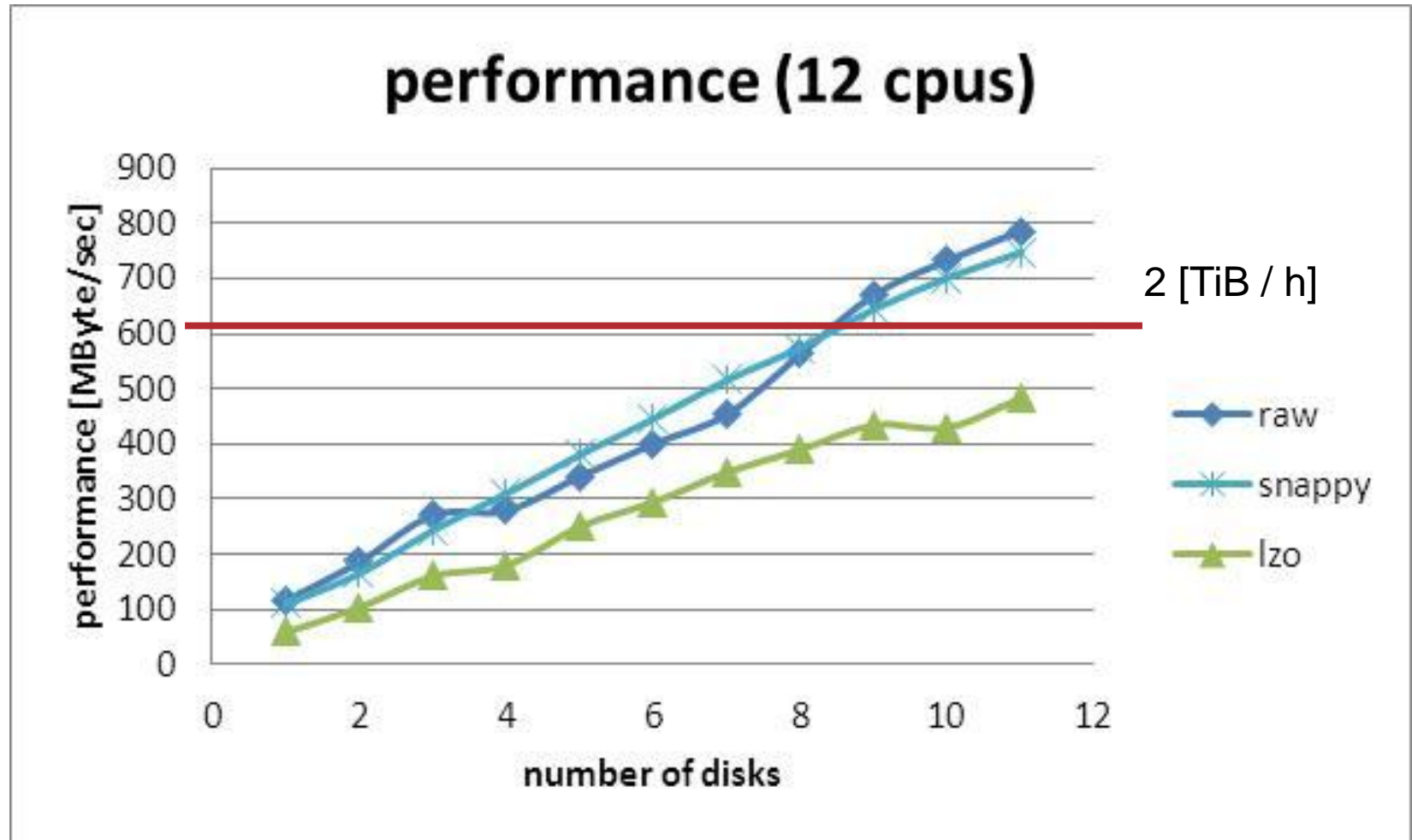
# Benchmark result (1 CPU)

■ Larger is better



# Benchmark result (12 CPUs)

■ Larger is better

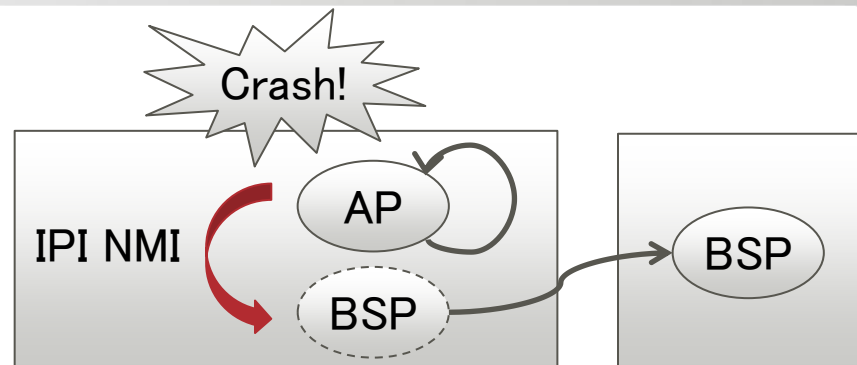


# Community development status

- Enter capture kernel with BSP by switching IPI NMI

➔ Naked

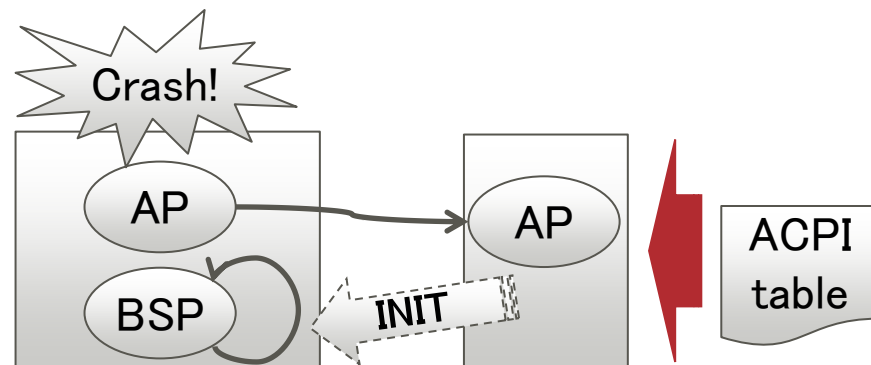
Depending IPI NMI reduces reliability



- Disable BSP at Capture Kernel

- Cannot execute rdmsr to read BSP flag bit in MSR for the CPU not running
  - Chicken-and-egg problem

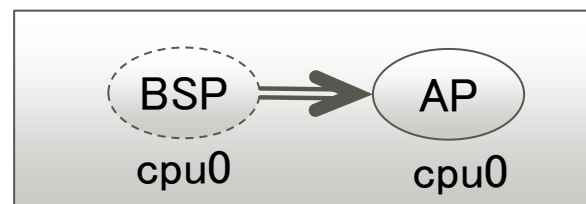
- Look up APIC/MP table to check if given CPU is BSP



- Unset BSP flag bit in MSR reg using wrmsr instruction in System Kernel

- Some say there should be firmware assuming BSP flag bit to be kept throughout runtime

System kernel



- Several issues for “mini dump” has been being improved
  - Memory consumption issue
  - Dump filtering performance degradation issue
  - ➔ Now mini dump can be collected with no problem
- Ongoing work for “full dump”
  - Multiple CPUs are essential for “full dump” configuration
  - Under discussion in community

## ■ Improvements for Capture Kernel to show performance like the experimental benchmark

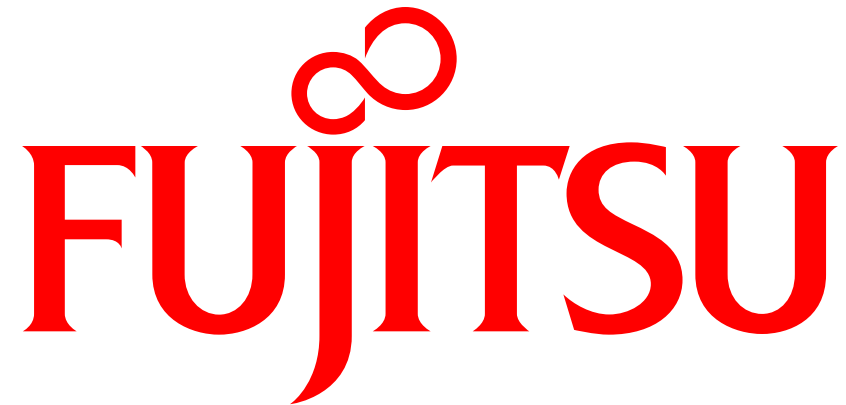
### ■ Kernel

- Multiple CPUs on capture kernel
- Support large pages on remap\_pfn\_range

### ■ Makedumpfile

- Support direct I/O for makedumpfile
- Make compression block size configurable

## ■ A lot of benchmark



shaping tomorrow with you