Crashdump Improvements for Larger Systems

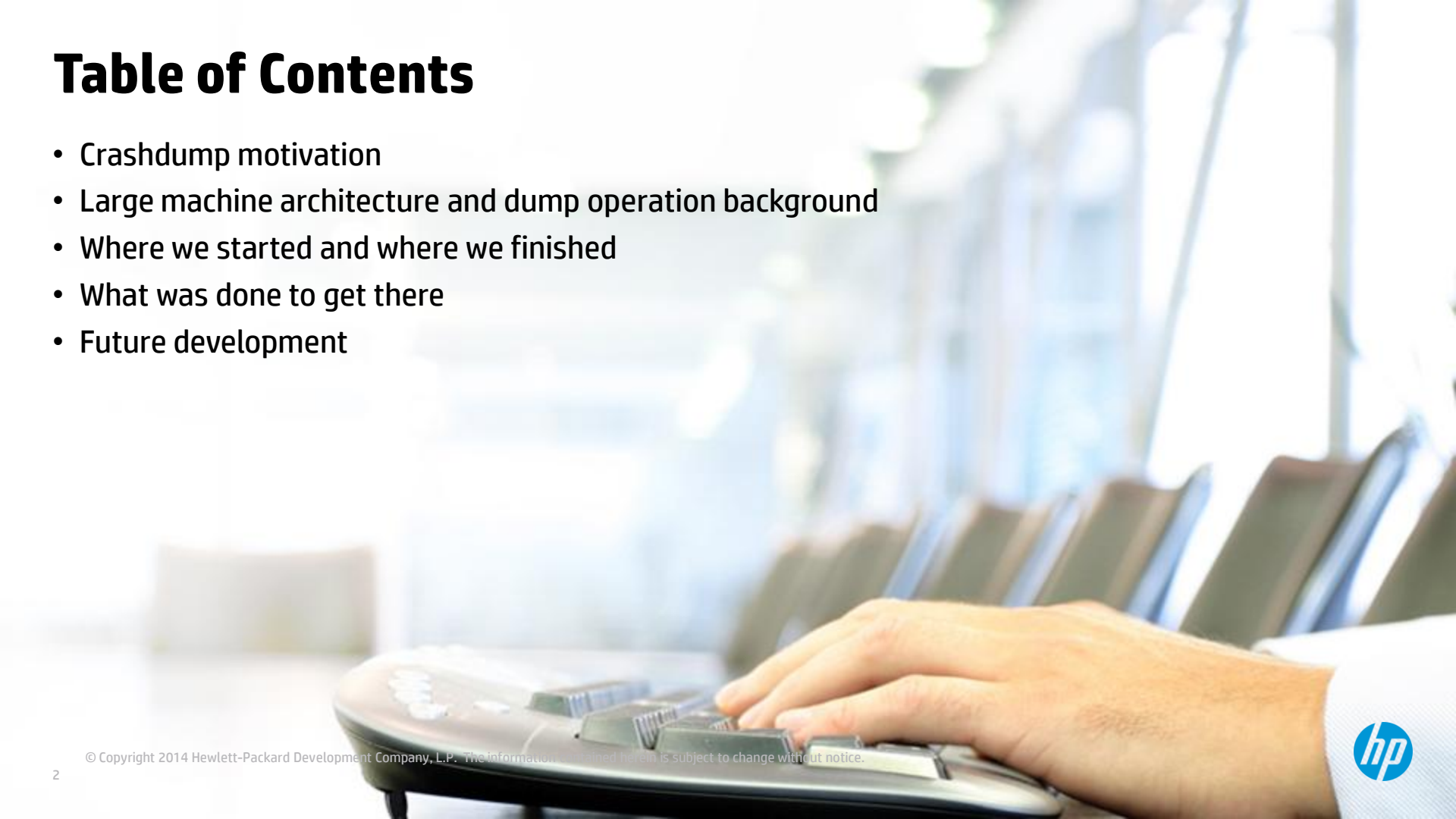
Tom Vaden, Distinguished Technologist, HP Server Linux Operating Environments

The Linux Foundation Enterprise End User Summit

June 2014

Table of Contents

- Crashdump motivation
- Large machine architecture and dump operation background
- Where we started and where we finished
- What was done to get there
- Future development



Motivation

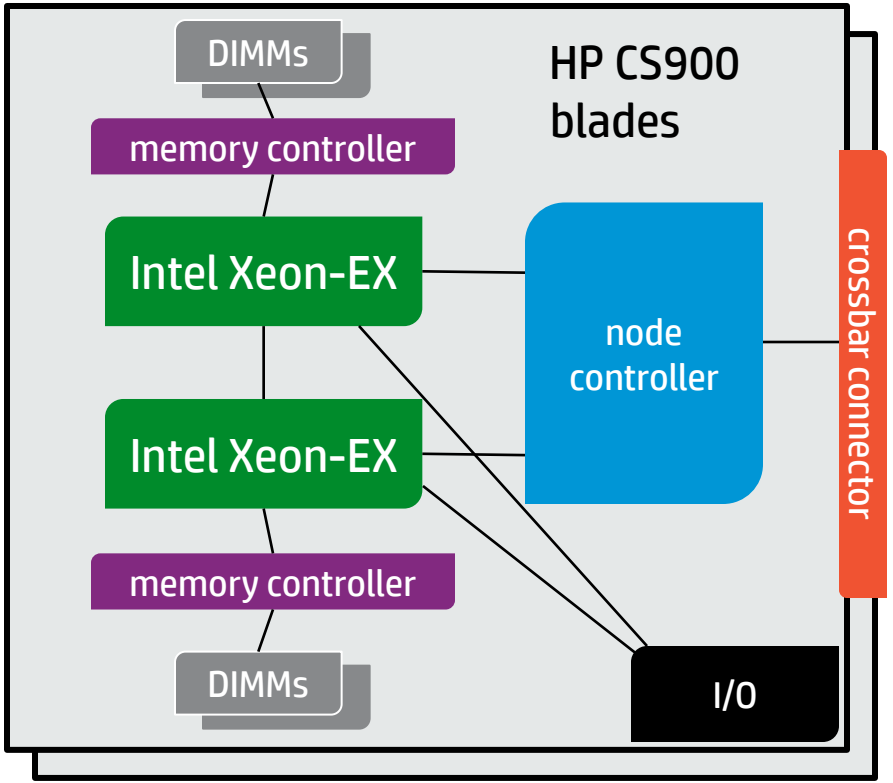
Crashdump motivation

- dump correctly
- dump quickly
- larger and larger systems
  - E.g. 12TB, 240 cores/480 threads
- crashdump isn't considered the most exciting system component but enterprise-class expectations for system downtime and recovery make it very important for some machine classes.
- BTW: a great example of initially community collaboration:
  - initially very small, but growing collaboration
  - several people, several companies/organizations

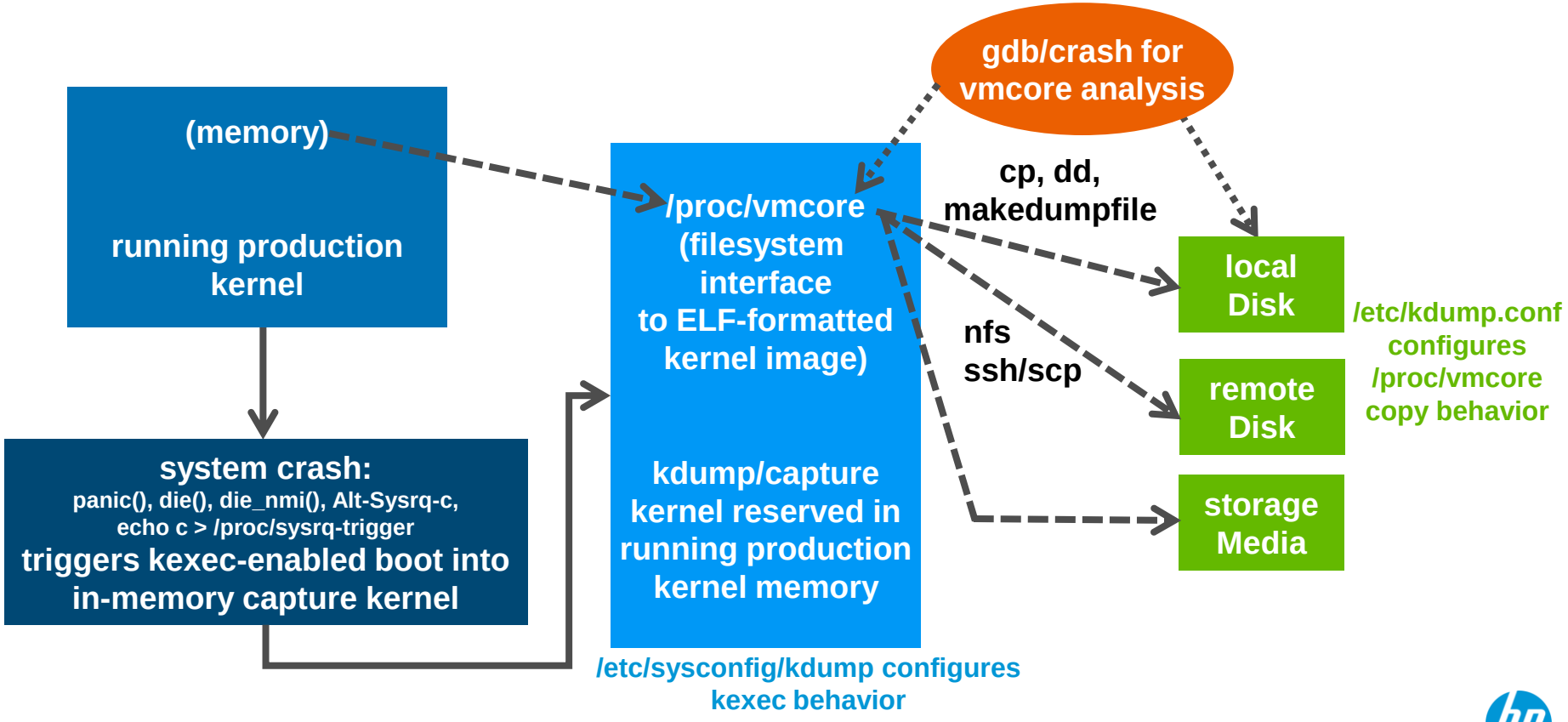
Large machine architecture and dump operation background

Large machine architecture

- HP CS900:
  - up to:
    - 8 blades
    - Compute: 16 CPUs, 240 cores, 480 threads
    - Memory: 384 DIMMs, 12TB capacity (with 32GB DIMMs)
    - I/O: 16 FlexLOMs, 24 Mezz cards
  - NUMA
  - NUIOA
  - non-attached I/O
  - UEFI



kdump/kexec operation



Where we started compared to where we “finished”

Where we started

- 6TB/240 cores
- 2.6 kernel and makedumpfile 1.4.1
- dump level 31
- lzo compression
- filter/compress/copy
- crashkernel=512M
- makedumpfile start to finish – **12 hours**

Where we “finished*”

- 12TB/240 cores
- 3.10 kernel and makedumpfile 1.5.4
- dump level 31
- lzo compression
- filter/compress/copy
- crashkernel=512M
- makedumpfile start to finish – **15 minutes**

*never finished

What was done to get there

What was done to get there: performance

- mmap for vmcore performance/scaling
- compression in makedumpfile
- parallelism
- responding to cyclic processing

vmcore access via mmap

- access to /proc/vmcore was slow
  - Originally via ioremap/iounmap
  - One page at a time
  - Affecting both copy and filtering
- use of mmap for /proc/vmcore greatly improves performance
  - Eliminating kernel->user copy and the associated memory meta-data handling
  - Cuts the copy/filter time to 1/3 of the original ioremap time

compression

- zlib as a baseline versus lzo and snappy compression
- lzo compression timing: ~60% of zlib
- snappy compression timing: ~40% of zlib
- lzo and snappy increase speed at the slight cost of larger dump files: ~50% in both cases

parallelism enablement

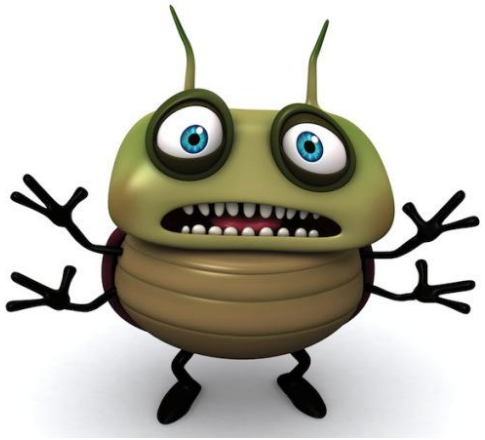
- parallelism prerequisites:
  - Support capture kernel sizes larger than 896M
  - Support capture kernels that load above 4GB
  - Support for booting capture kernel with more than 1 CPU
  - Disabling original kernel BSP in capture kernel
- pulled in some future parallelism plans to address functional issues (irq sufficiency)
- manually possible to split and dump to multiple HBAs but more on that in the future

responding to cyclic processing

- honorable mention of the work to introduce cyclic processing and then make it work faster
- a great idea for makedumpfile to cycle through regions of /proc/vmcore
  - enables memory size dump scaling
  - rinse, lather, repeat on each region until done
  - reduces capture kernel memory requirements
- make sure processing time for makedumpfile didn't suffer too much
  - allow non-cyclic dumping as an advantage for some configurations
  - tuning the cyclic buffer size to optimize number of cycles
  - better bitmap handling

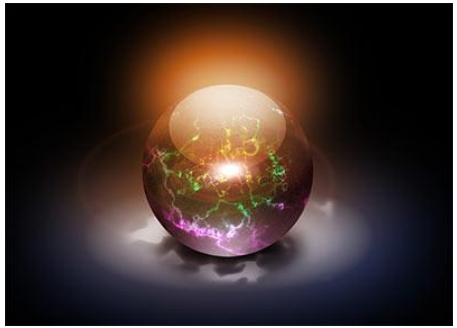
What was done to get there: function

- 64-bit issues
- 46-bit addressing kernel and makedumpfile
- more than 256 processors
- loading crash kernel above 4G to reduce competition for low memory space
- kexec - EFI in capture kernel
- iommu=on (TBD)



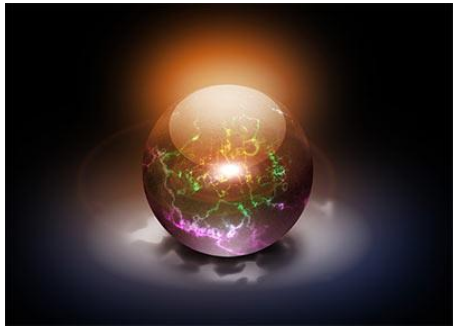
Future development

Future development



- huge pages filtering
- secure boot dump
- early/late dump not well covered (the bane of our existence)
- live dump
- dump analysis tools

Future development (continued)



- how best to use parallelism?
  - make use of parallel dump to multiple HBAs simultaneously
  - parallel compression and filtering in addition to I/O
  - just turning on all the cpus is the wrong answer
    - bottlenecks happen and dumps slow down
    - crash kernel size implications
    - requires intelligent choice of degrees of parallelism
  - NUMA and memory locale
  - NUIOA and I/O locale
- crash kernel sizing
  - Including consideration of configuration: I/O and processor
  - harder than it seems
- FCoE and dump
  - OpenFCoE infrastructure in capture kernel
- pre-dumping ahead of a crash
  - repetitively flush kernel memory for a makedumpfile image while the system is up
  - only dump the final delta when the system actually crashes

Thank you