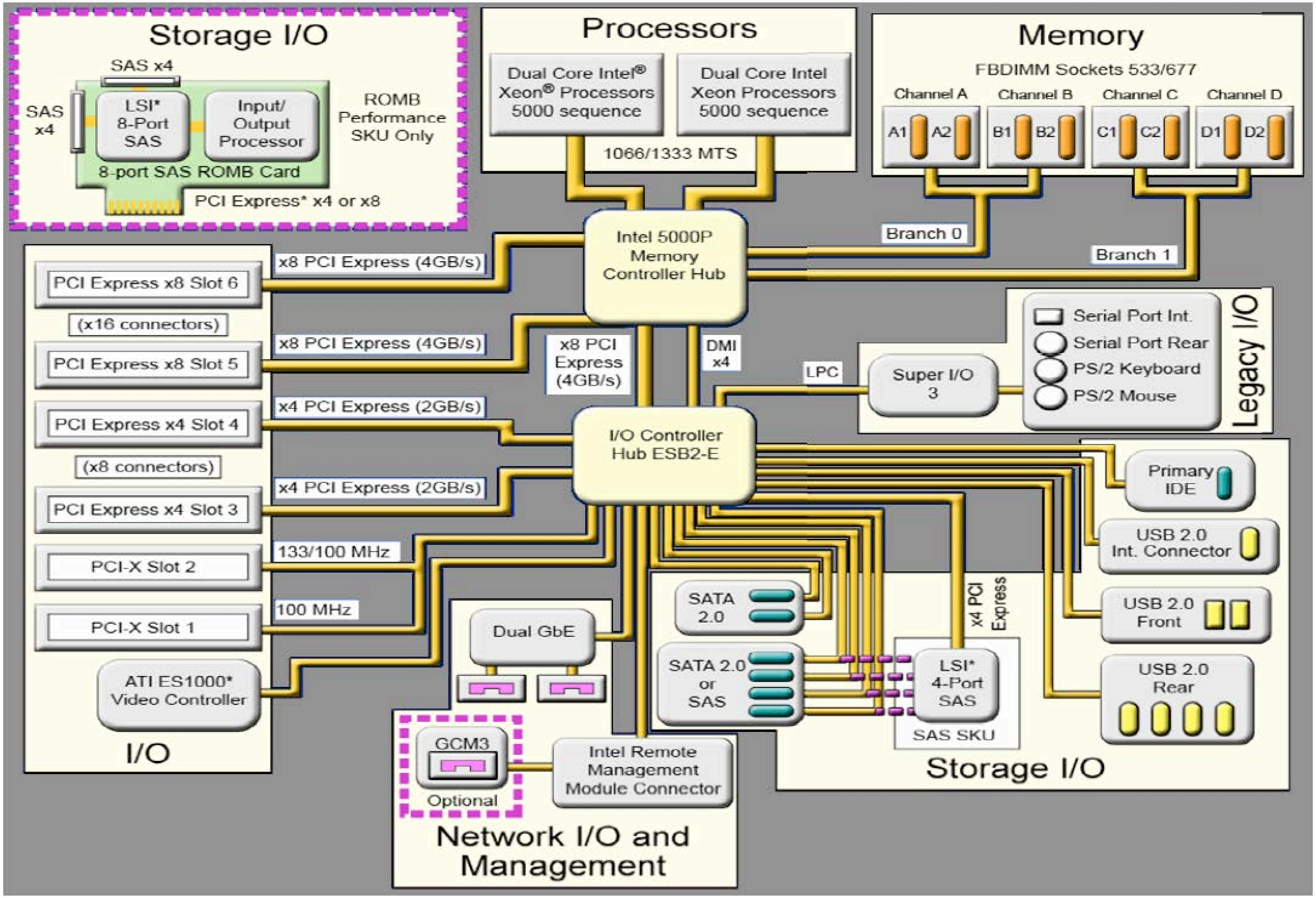
Beyond Virtualization: A Novel Software Architecture for Multi-Core SoCs

Jim Ready
September 18, 2012

How HW guys view the world

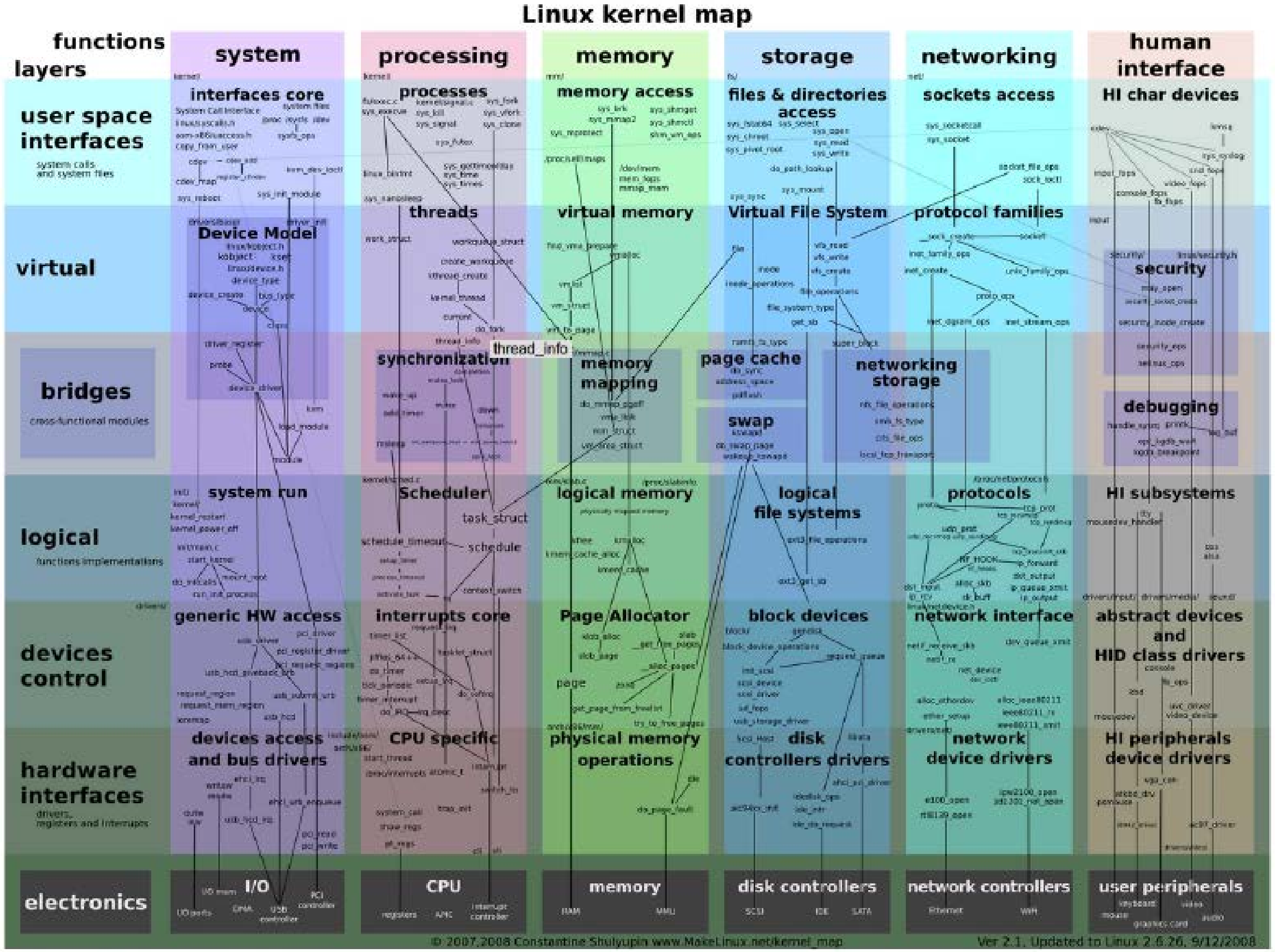
SW → Software

HW



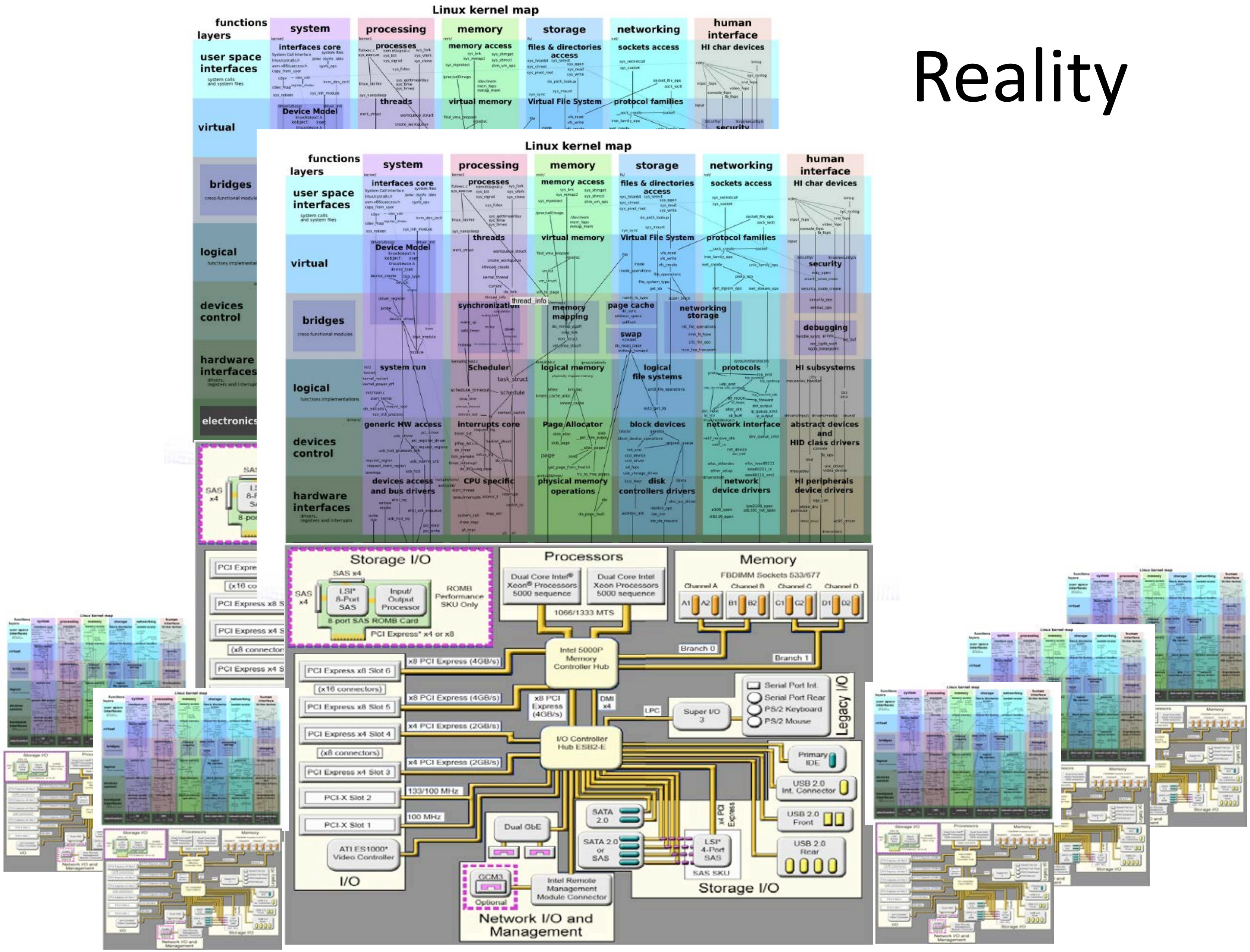
How SW guys view the world

SW



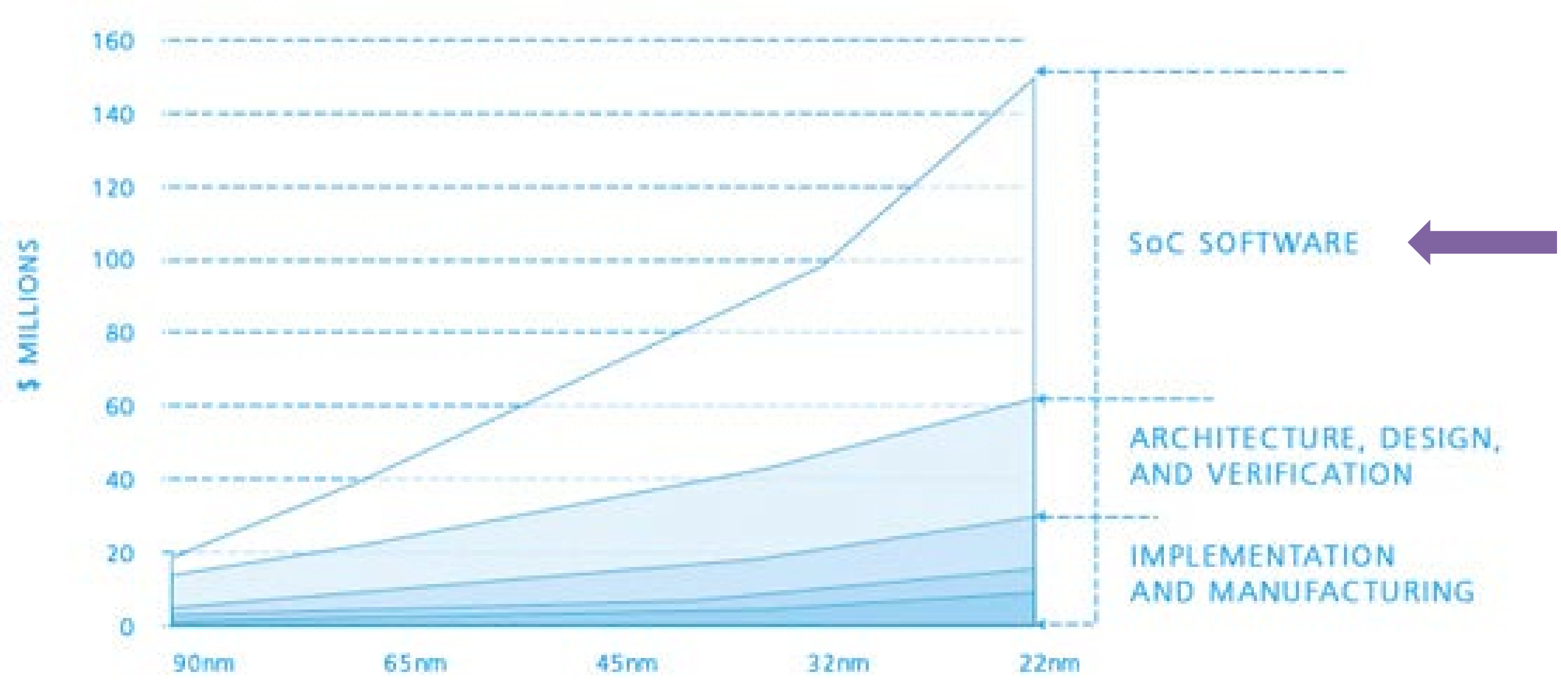
HW →

Reality


Linux kernel map
functions
layers
system
processing
memory
storage
networking
human interface
user space interfaces
interfaces core
processes
memory access
files & directories access
sockets access
HI char devices
virtual
Device Model
threads
virtual memory
Virtual File System
protocol families
security
bridges
synchronization
memory mapping
page cache
swap
networking storage
debugging
logical
system run
Scheduler
logical memory
logical file systems
protocols
HI subsystems
devices control
generic HW access
interrupts core
Page Allocator
block devices
network interface
abstract devices and HID class drivers
hardware interfaces
devices access and bus drivers
CPU specific
physical memory operations
disk controllers drivers
network device drivers
HI peripherals device drivers
electronics
Storage I/O
SAS x4
LSI 8-Port SAS
Input/ Output Processor
8-port SAS ROMB Card
PCI Express x4 or x8
ROMB Performance SKU Only
Processors
Dual Core Intel Xeon Processors 5000 sequence
Dual Core Intel Xeon Processors 5000 sequence
1066/1333 MTS
Memory
FBDIMM Sockets 533/677
Channel A
Channel B
Channel C
Channel D
A1
A2
B1
B2
C1
C2
D1
D2
Branch 0
Branch 1
Intel 5000P Memory Controller Hub
x8 PCI Express (4GB/s)
x8 PCI Express (4GB/s)
x8 PCI Express (4GB/s)
DMI x4
x4 PCI Express (2GB/s)
x4 PCI Express (2GB/s)
133/100 MHz
100 MHz
PCI Express x8 Slot 6
(x16 connectors)
PCI Express x8 Slot 5
PCI Express x4 Slot 4
(x8 connectors)
PCI Express x4 Slot 3
PCI-X Slot 2
PCI-X Slot 1
ATI ES1000 Video Controller
I/O
LPC
Super I/O 3
Serial Port Int.
Serial Port Rear
PS/2 Keyboard
PS/2 Mouse
Legacy I/O
I/O Controller Hub ESB2-E
Primary IDE
USB 2.0 Int. Connector
USB 2.0 Front
USB 2.0 Rear
SATA 2.0
SATA 2.0 or SAS
x4 PCI Express
LSI 4-Port SAS
SAS SKU
Storage I/O
Dual GbE
GCM3
Optional
Intel Remote Management Module Connector
Network I/O and Management

The SoC "Software Crisis"

SW costs dominate SoC Development

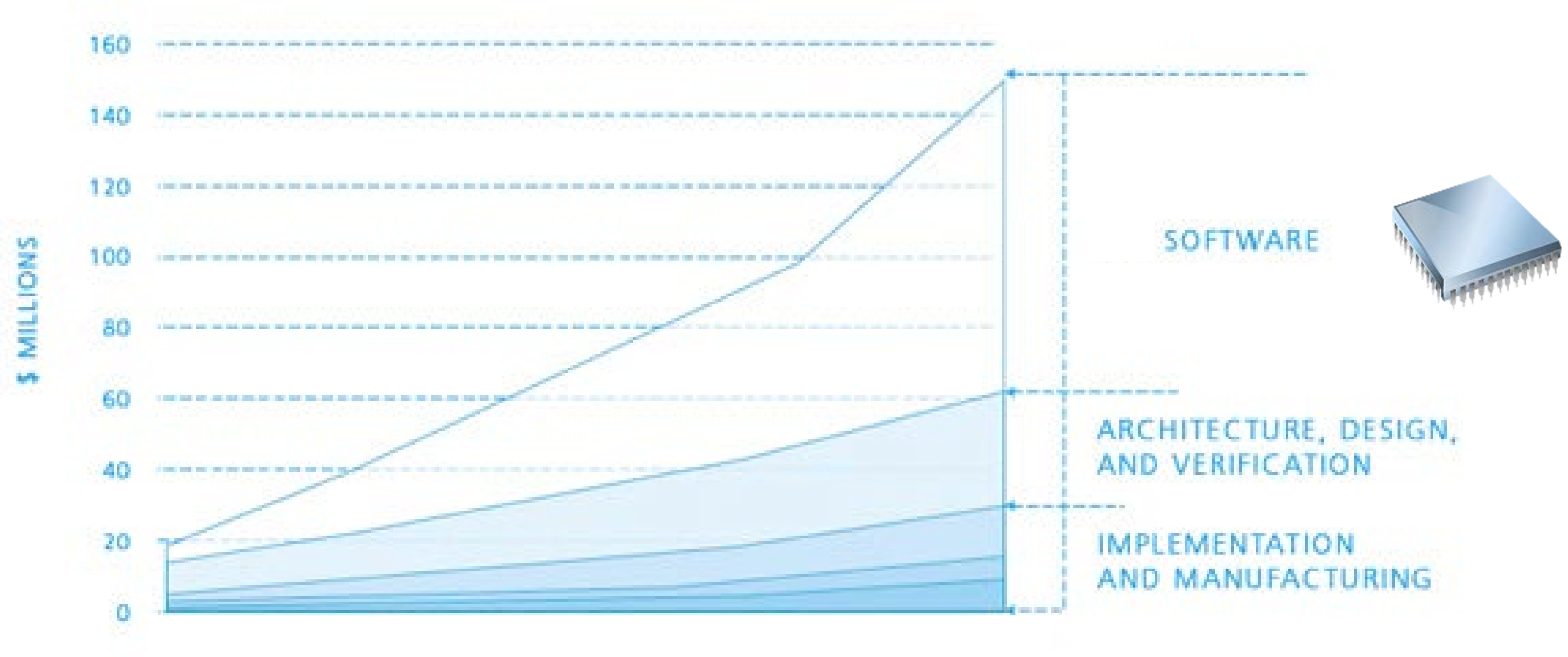


What's going on here?

We really need to know why this is happening and what can be done about it.

The recurring “Software Crisis”: 1968 – 2012…

(We’ve seen this movie before…)



SoC Crisis 2012

1968 2008

Escalating software costs for aircraft carriers and jets, first noted in 1968 NATO Software Engineering Conference at Garmisch, Germany, gave rise to the term “Software Crisis”. The “crisis” has been continuous for electronic systems ever since.

The “No Silver Bullet” principal

“There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability, in simplicity.”

”Skepticism is not pessimism, however. Although we see no startling breakthroughs, and indeed, believe such to be inconsistent with the nature of software, many encouraging innovations are underway….There is no royal road***, but there is a road***”.

Frederick P. Brooks

“No Silver Bullet – Essence and Accident in Software Engineering”

The nature of software development*

Accidental Complexity is the difficulty of transforming the conceptual representation of software into the reality of running on a particular piece of hardware. The single largest gain in software productivity has been made by the introduction of high-level languages, which automated this transformation process.

Accidental complexity is amenable to tooling solutions.

Essential Complexity is what's left – **the really hard part**. Solutions that attack essential complexity include:

- Buy versus build
- Requirements refinement & rapid prototyping
- Incremental development
- Great Designers

A new SoC:

Generates substantial Accidental Complexity -> New Tools

Can benefit from "outside" SW technology **to reduce Essential Complexity -> *OSS and SW IP***

***From Frederick P. Brooks**
"No Silver Bullet – Essence and Accident in Software Engineering"

The SoC software challenge

(CW: SoC = Multi-Core + Virtualization)

- Use Cases for virtualization in the IT world
  - Server consolidation
  - Underutilization
  - Management of numerous OSs and dependant applications

- Hardware Considerations
  - Very uniform server hardware platforms, esp. I/O
  - Extensive processor support for virtualization

- Huge uniform market
  - Numerous successful companies of very large scale

Embedded is different yet again

- Embedded devices are already highly optimized
  - Size, power consumption, cpu utilization etc.

- No layer of software makes a processor go faster

- So far not a big $$$ market

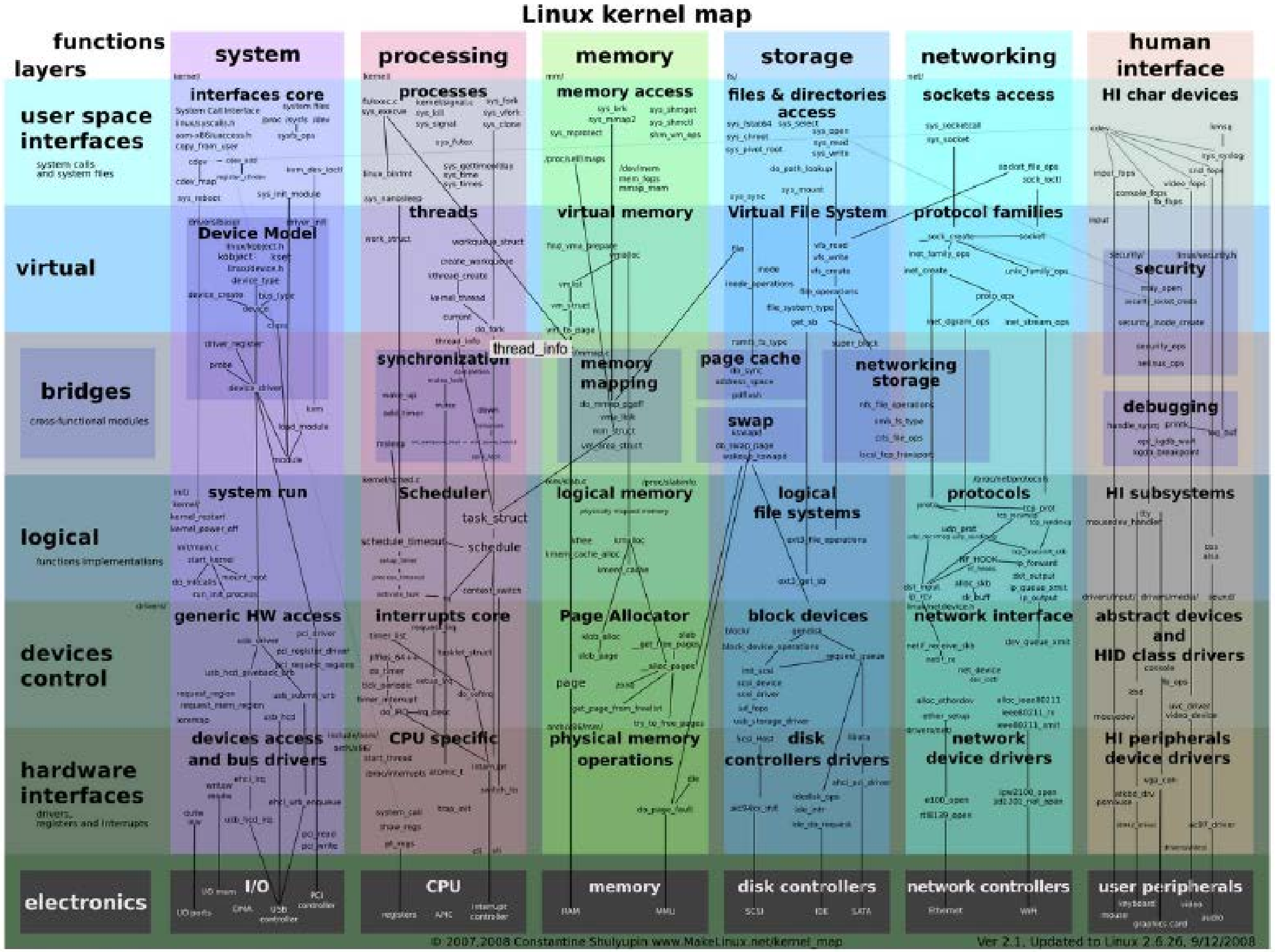
- Multi-core does NOT automatically mean:
  - RTOS for data plane
  - Hypervisors/virtualization
  - Multiple OSs
  - Many-core

The real SoC trend will be a small number of cores + extensive application-specific hardware acceleration. What does this mean for software?

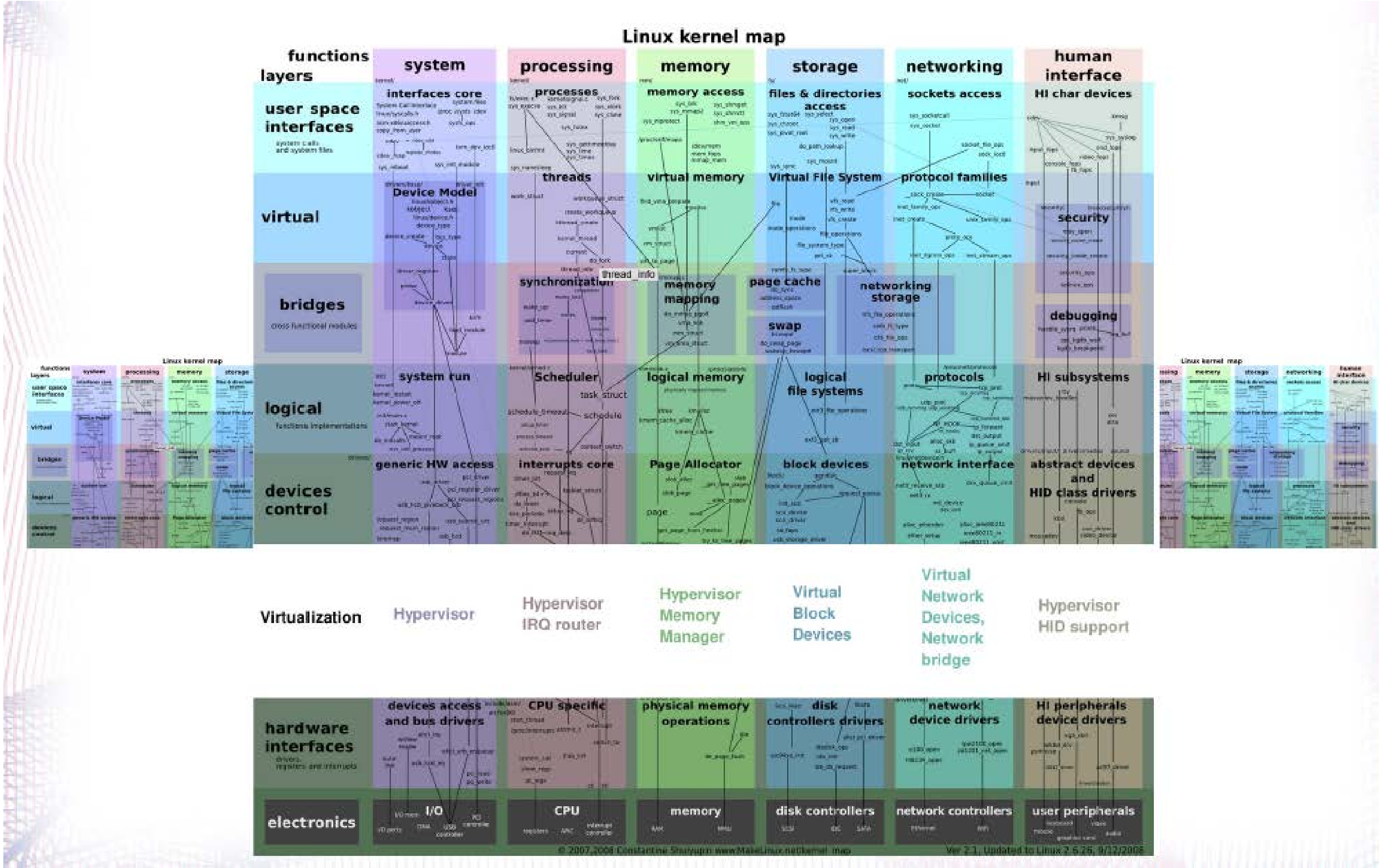
“If you are going to have Linux on your device anyway, why not just have Linux do it all?”

Linux Resources for Supporting Complex SoCs

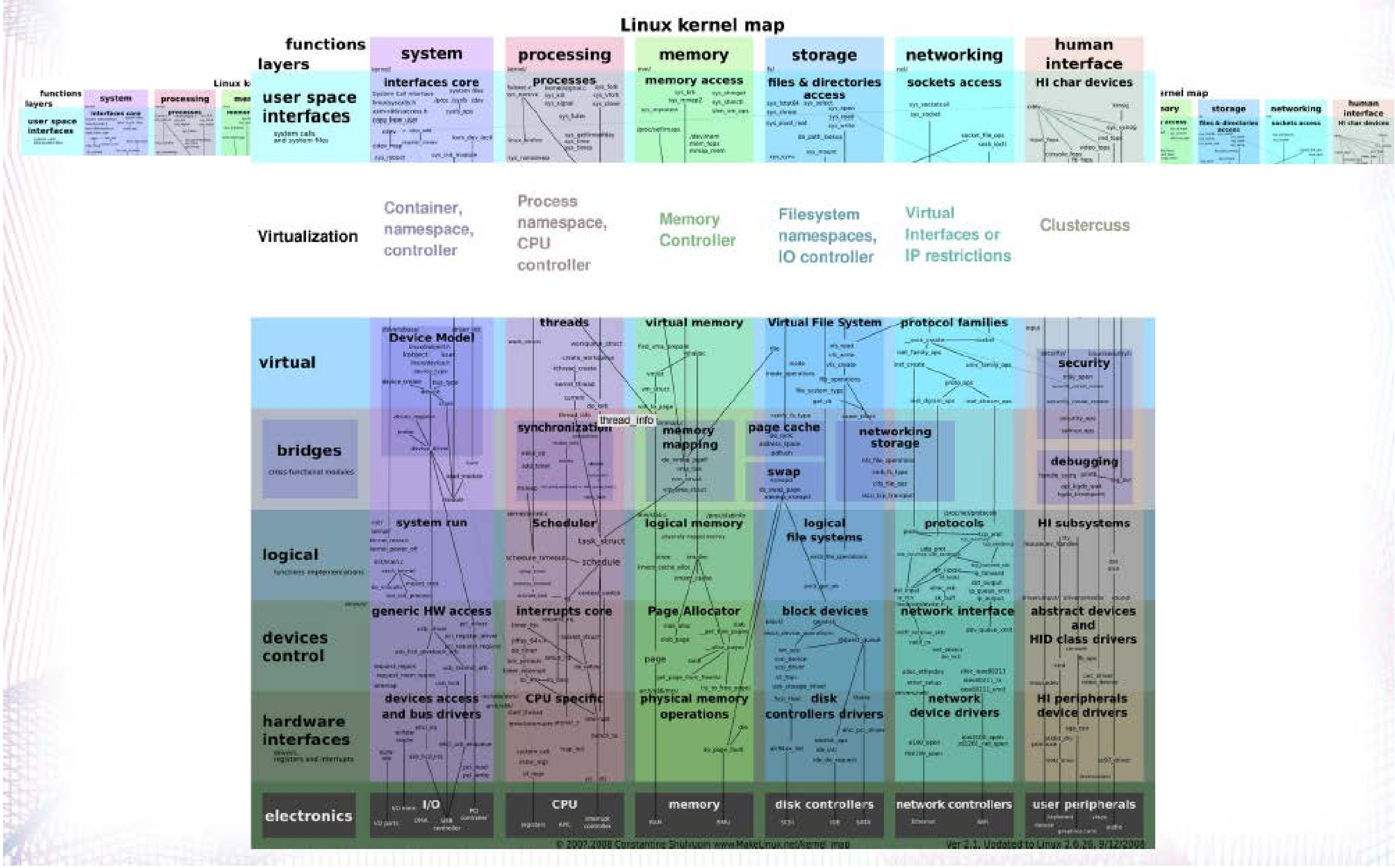
Linux


Linux kernel map
functions
layers
system
processing
memory
storage
networking
human interface
user space interfaces
system calls and system files
interfaces core
processes
memory access
files & directories access
sockets access
HI char devices
virtual
Device Model
threads
virtual memory
Virtual File System
protocol families
security
bridges
cross-functional modules
synchronization
thread_info
memory mapping
page cache
swap
networking storage
debugging
logical
functions implementations
system run
Scheduler
task_struct
schedule
logical memory
logical file systems
protocols
HI subsystems
devices control
generic HW access
interrupts core
Page Allocator
page
block devices
network interface
abstract devices and HID class drivers
hardware interfaces
drivers, registers and interrupts
devices access and bus drivers
CPU specific
physical memory operations
disk controllers drivers
network device drivers
HI peripherals device drivers
electronics
I/O
CPU
memory
disk controllers
network controllers
user peripherals
© 2007,2008 Constantine Shulyupin www.MakeLinux.net/kernel_map
Ver 2.1, Updated to Linux 2.6.26, 9/12/2008

Linuxes on KVM



Linux on Containers



Virtualization

Container, namespace, controller

Process namespace, CPU controller

Memory Controller

Filesystem namespaces, IO controller

Virtual Interfaces or IP restrictions

Clustercuss

Linux “Bare Metal”

- High performance, ultra low overhead Linux
  - “Bare-metal-like” performance
  - Goal: ~99% of CPU is dedicated to the process
  - Real Time scheduling for prioritized execution, ultra low latency
  - RTOS replacement
- User mode environment with direct access to hardware
  - No kernel mode programming required
  - All development done in user mode
- Specific enhancements
  - Processor affinity
  - Cgroups
  - Interrupt vectoring
  - Memory mapping – especially I/O
  - Tickless configuration
    - Only get timer interrupts when needed instead of periodically

Example: MontaVista Bare Metal Engine

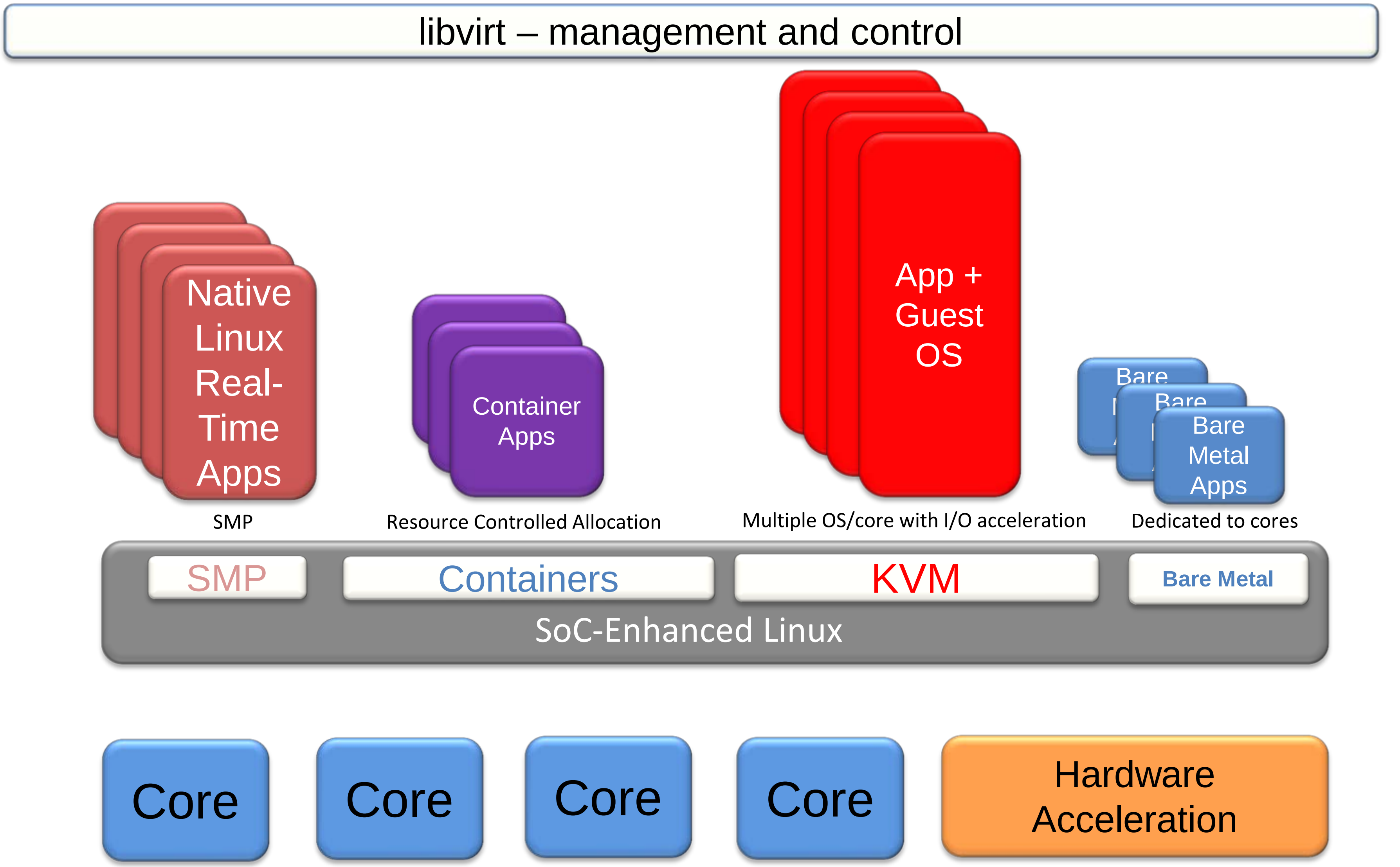
(Similar results from Tilera and others)

- Equal level of performance as semi’s dedicated run-times, >99%

Forwarding Performance (% line rate, 64byte frames)		% Difference	Avg Latency (ns)	
Linux/ BME	SimpleExec		Linux/BME	SimpleExec
84.13	84.69	-0.67	3750	3207

Following are numbers on the revision 2.1 63xx board, with SDK 2.1 toolchain configurations.
EP6300C board revision major:1, minor:1, serial #: 97800
OCTEON CN6335-AAP pass 2.1, Core clock: 1250 MHz, IO clock: 800 MHz, DDR clock: 533 MHz (1066 Mhz data rate)

Linux SoC Architecture



libvirt – management and control

Native Linux Real-Time Apps

SMP

Container Apps

Resource Controlled Allocation

App + Guest OS

Multiple OS/core with I/O acceleration

Bare Metal Apps

Dedicated to cores

SMP

Containers

KVM

Bare Metal

SoC-Enhanced Linux

Core

Core

Core

Core

Hardware Acceleration

Thanks