Microsoft

Microsoft® Research Faculty Summit 2012

ADVANCING THE STATE OF THE ART

Operating System Architecture For Parallel And Distributed Computing

Burton Smith
Microsoft Technical Fellow

July 17, 2012

Operating System Architecture Is Ancient

Most of it dates back to the 60's and 70's

CTSS (1961), VM/CMS (1968), UNIX (1969), VMS (1977)

It locks us in to old ways of doing things

Using a few identical processor cores
Using the OS kernel to assign cores to threads ("kernel threads")
Using OS kernel operations to synchronize threads (typically by locking)
Using interrupts to synchronize threads with devices
Using traps to let threads invoke operating system services
Using a variety of heuristics to allocate and schedule resources

These approaches don't cut it any more…

Hardware Has Changed

Available battery life limits user experiences

There isn't enough energy to run all of the hardware all of the time.
How can the state of battery charge be taken into account?

Processor cores are increasingly heterogeneous

Application software developers are trying to exploit them.
Can every core type access the OS services it needs?

New sensors and interface devices are emerging

Some of these require substantial or timely resource allocation.
How are these resource allocations decided and implemented by the OS?

Software Has Changed

Parallelism varies in both form and quantity

Even different phases of a single application can exhibit such diversity.
How can the OS adapt to varying application parallelism?

Media and games require responsiveness

Different platforms need different resource allocations even for the same function.
How can appropriate allocation be done, even given multiple simultaneous instances?

Applications are built from distributed services

These services provide search, data access, connectivity, computation, and the like.
How well does application performance compose from the services it comprises?

Vulnerability Has Changed

Attacks frequently target drivers and system services

These are increasingly large and complex software subsystems.
Must the OS share most of its resources with its device drivers and system services?

Distributed applications often connect via the internet

Messages among application components need authentication and privacy.
How do we ensure trust for both ends of these internet connections?

Internet security is an ongoing multi-player contest

The forces of good must keep on improving to keep on winning.
How can we defend against complete OS takeovers by the opposition?

Unconventional Approaches

Microkernels: KeyKOS (1992), L4 (1995), EROS (1999)

Remove drivers from the kernel, kernel threads, manage resources with capabilities

Exokernels: Xok (1997)

Unprivileged application use of hardware, kernel threads, capability-based memory sharing

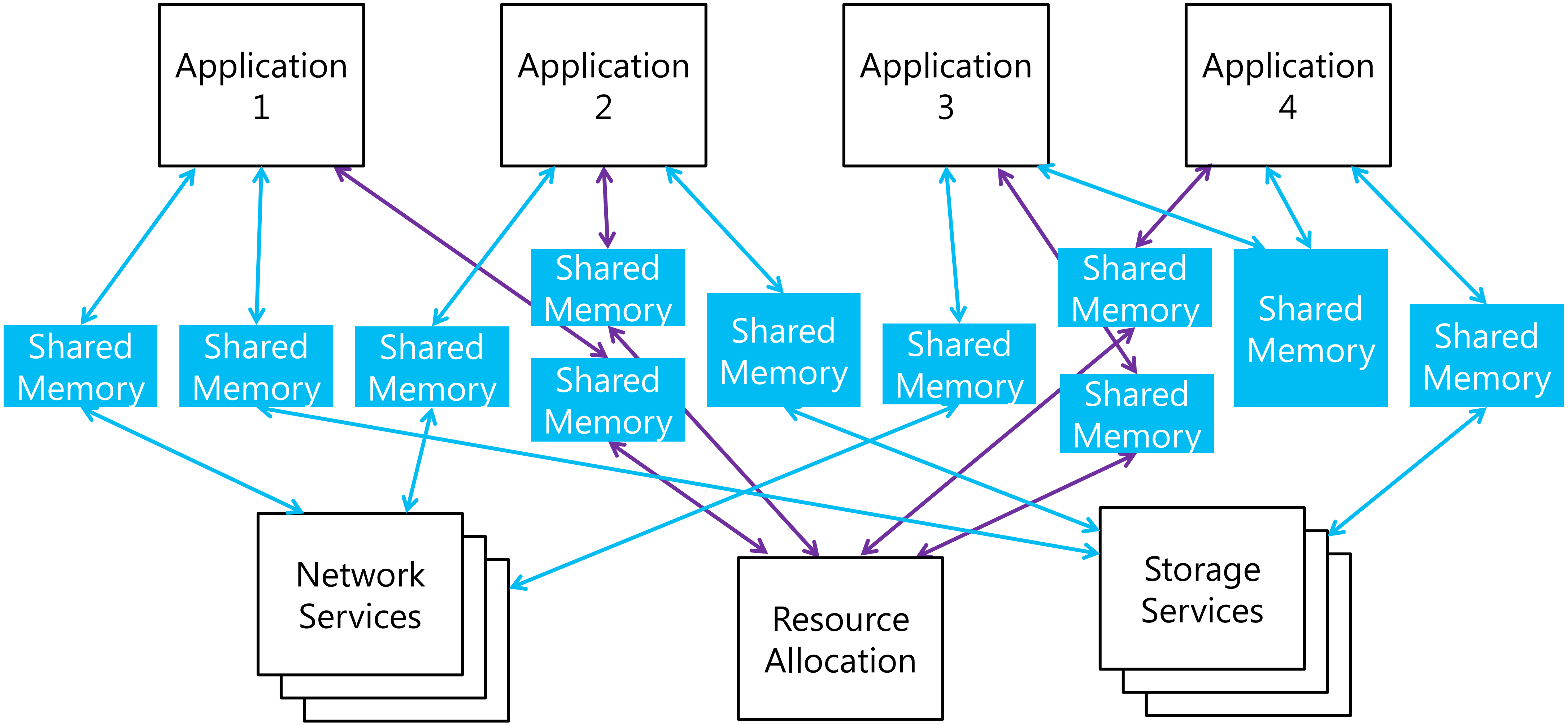
Microsoft: Singularity (2007), Barrelfish (2009)

Message oriented; S: safe pointers and kernel threads, B: capabilities and user threads.

My own work: TeraOS (1992)

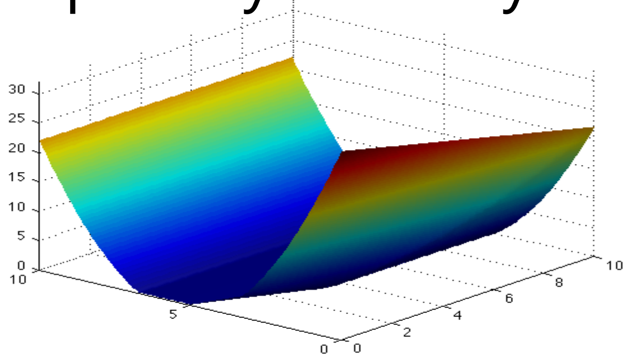
Reduced driver privilege, user threads, no interrupts

What Tomorrow's Client OS Might Look Like

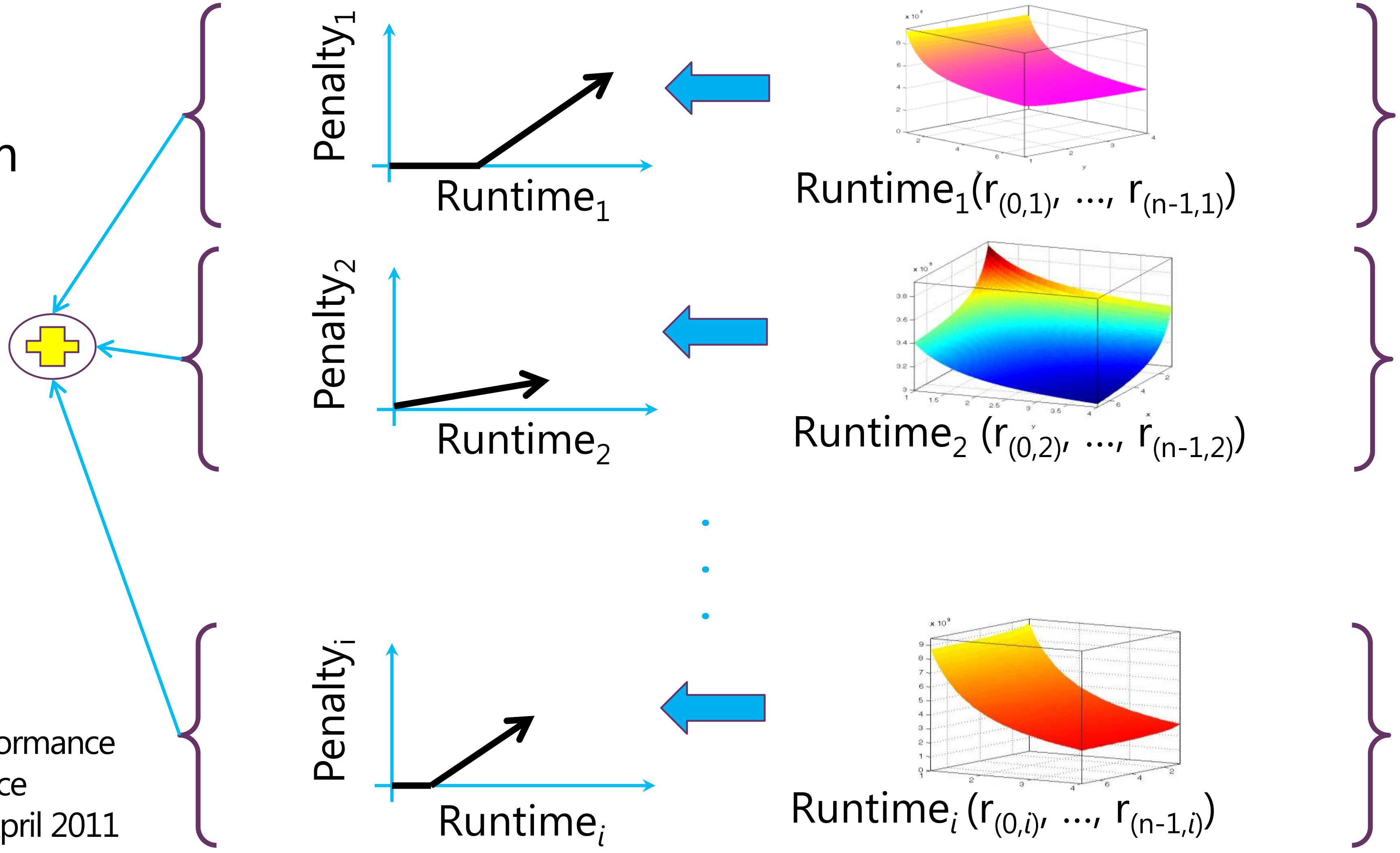


Convex Optimization For Resource Allocation

Continuously minimize the total penalty of the system

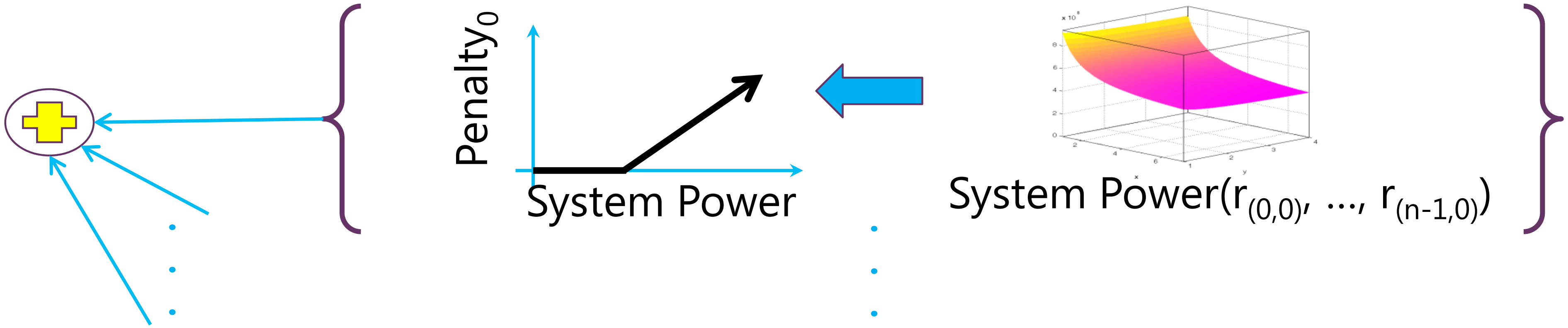


(subject to the total resources available)



S. L. Bird and B. J. Smith, "PACORA: Performance Aware Convex Optimization for Resource Allocation", Proc. HotPar11, Berkeley, April 2011

Power And Battery Energy Management



Treat the battery as a competitor for resources

The battery is associated with a distinct process, $Process_0$
All slack resources are assigned to $Process_0$ which tries to power them down
As resources power down the system power diminishes in a convex manner
System power thus acts as the “Runtime” for $Process_0$
$Penalty_0$ lets the system or user describe the relative importance of remaining battery life

Other Stuff Currently Underway

Dynamic Root of Trust Measurement (DRTM)

Measured from a small and secure Trusted Platform Module (TPM)

Asynchronous interfaces in Microsoft software

Asynchronous patterns in C#/F#/VB; reactive framework; more coming soon

C++ AMP for programming GPUs or CPUs

More work to do to make the two work together smoothly

Conclusions

Operating Systems are changing (at long last).

Opinions above are mine but maybe not Microsoft's.

Your questions and comments are welcome.

Microsoft