# an embedded perspective on linux power management

discussions on pm technology by
a guy who works for an
embedded Linux OS vendor

Todd Poynor
MontaVista Software

# embedded pm today

increasing hardware pm complexity

uncertainty as to what saves power

ce focus on device pm not clock scaling

pm remains a top challenge for mobile devices

# community and commerce

os product using non-mainline dvfs mechanism

advantages to sync up with a community solution

hoping summit sets direction for embedded dvfs

proposing concepts from dpm for upstream

osv adds value on standard framework

# the powerop hardware layer

manages sets of arbitrary power parameters

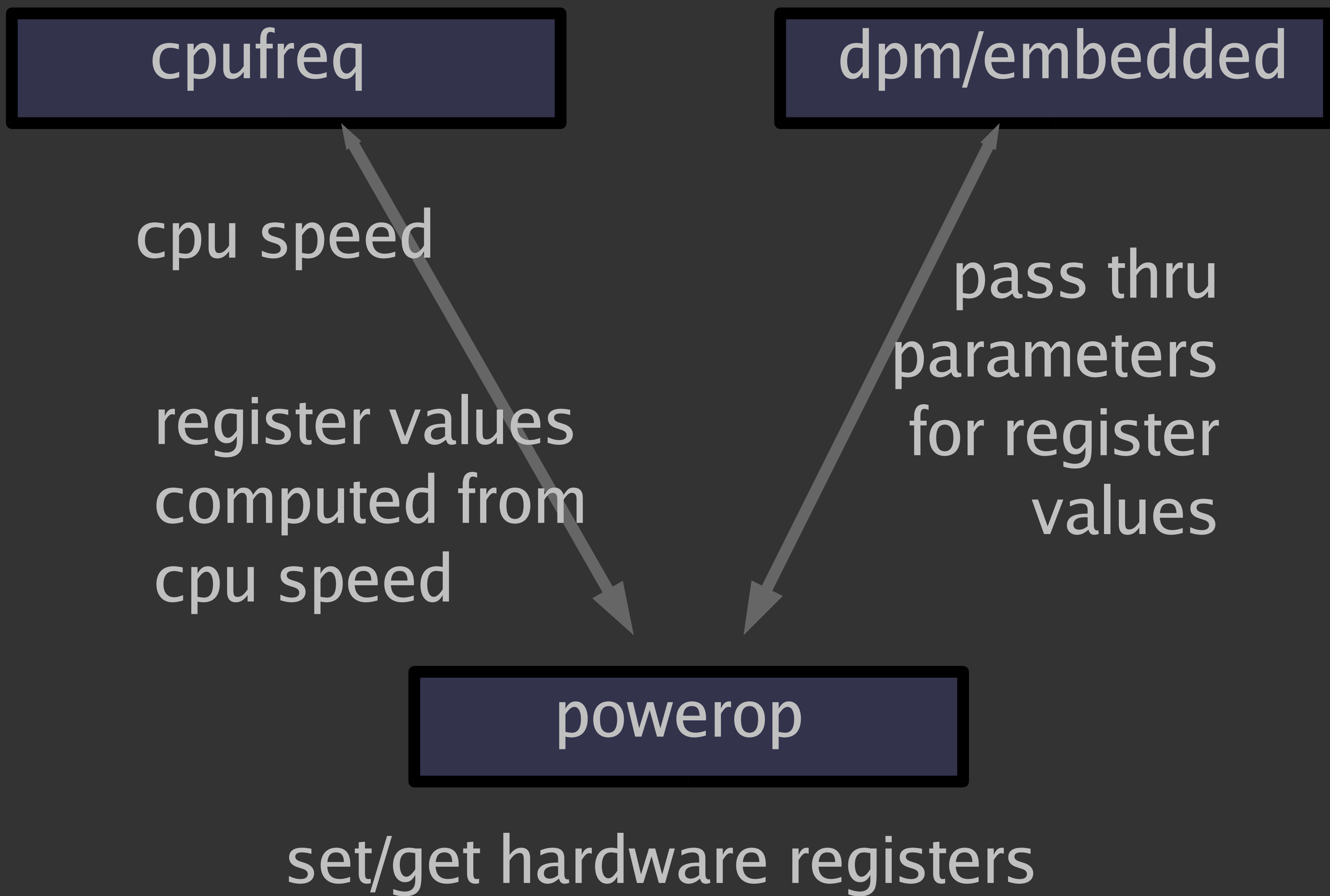
just the dpm “operating point”a bstraction

geared toward embedded hardware

dpm and cpufreq can share board-specific code

or maybe linux-pm has more ambitious plans

# powerop illustrated



# dpm maps states to operating points

policy P runs operating point O at state S

system states include idle, per-task states, sleep

conserve power during brief idle periods

apps can manage own custom state if desired

power state can be tied to scheduling priority

# resolve clocking conflicts per policy

what to do when device D needs clock C rate R?

dpm makes system designer choose in advance

chooses a valid operating point from that set

driver model extended with clock constraints

# device management is hot

formerly less complex, big savings

now multiple power and clock domains

with multiple power states and latencies

set policy via driver model or state->op style pm

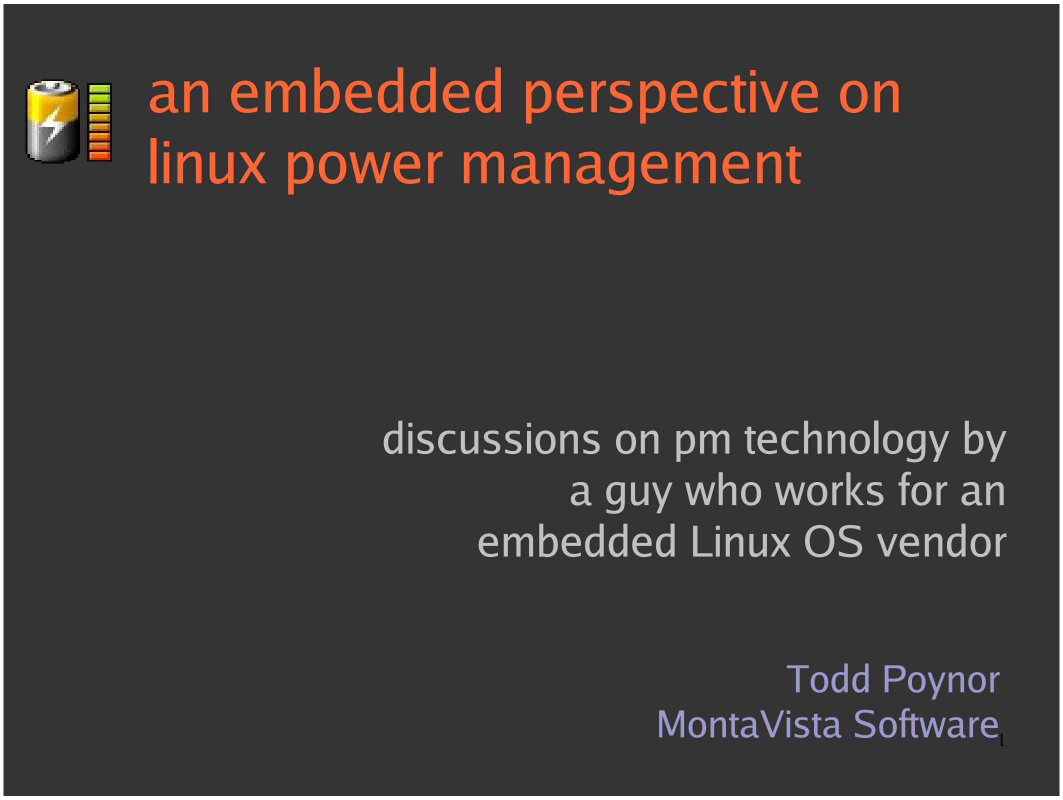
platform bus probably needs extensions

# more topics of interest

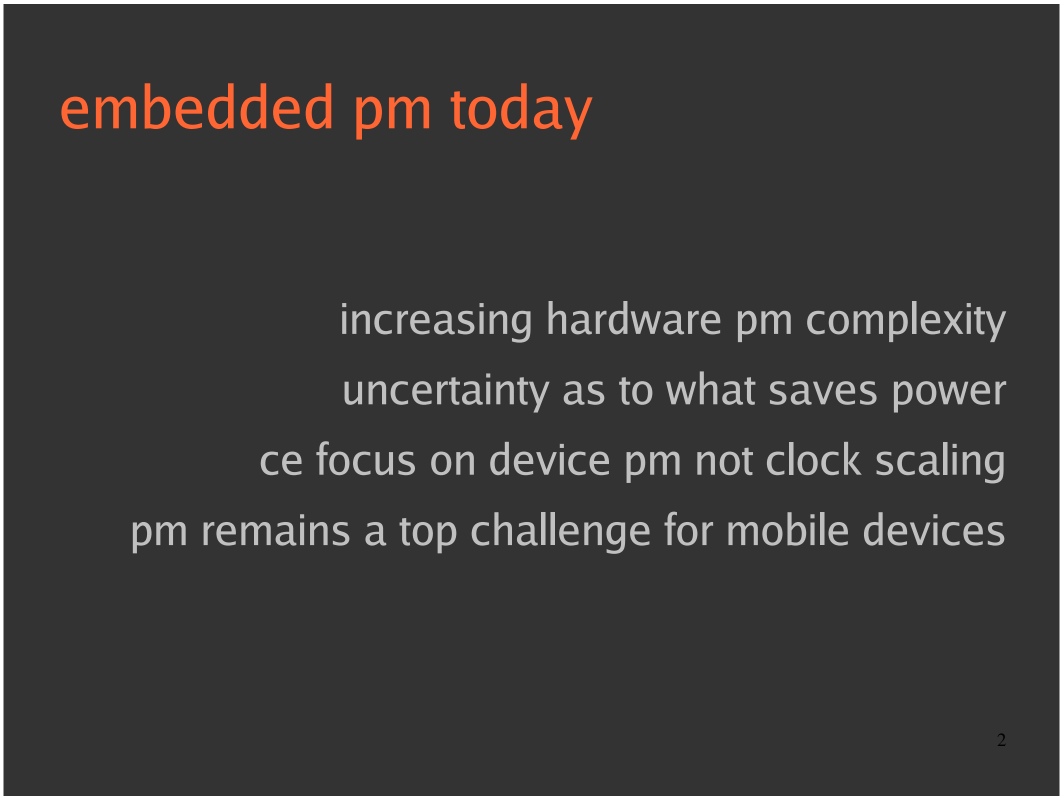
power event notification to userspace

reducing sources of unneeded idle ticks

assigning tasks to specific memory banks




Presented at the CELF Embedded Systems Conference 2006, and a portion at the 2006 Linux PM Summit.




Compare sizes of power/clocking chapter of an SoC family technical reference manual: past version ~115 pages, recent ~285, upcoming >400 pages (thanks R Woodruff).

h/w makers unsure how s/w will use features ("how should dpm be implemented on our new board?").

s/w makers unsure how to best use h/w pm features ("how should we use dpmon the new board?").

osv often the meeting point between the two.

a number of products only do device mgmt, no dvfs; dvfs considered complicated, a source of instability and less bang for buck.

Mark VandenBrink, Mot director of mobile devices s/w, NewForge interview T Bird sent out – PM primary challenge.

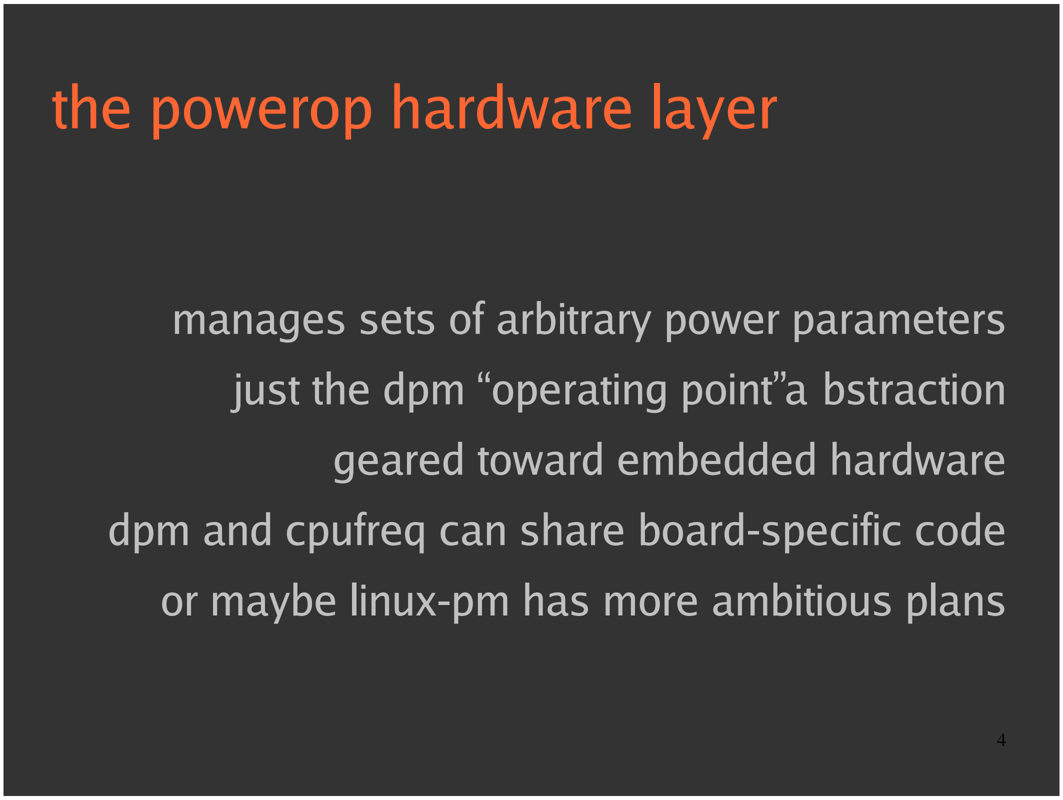

These are a developer's recommendations on the subject, not a statement of MontaVista Software.

DPM is the supported DVFS solution in the Mobilinux product, which is not in kernel.org.

OSVs can add value around a standard framework; adding an entire non-standard DVFS mechanism not the best place to be.

The PowerOP proposal is a step in that direction.

Plenty of room for OSV value add: initial board development, integrated offerings with preselected power policies, add policy selection technologies such as ARM IEM, etc.

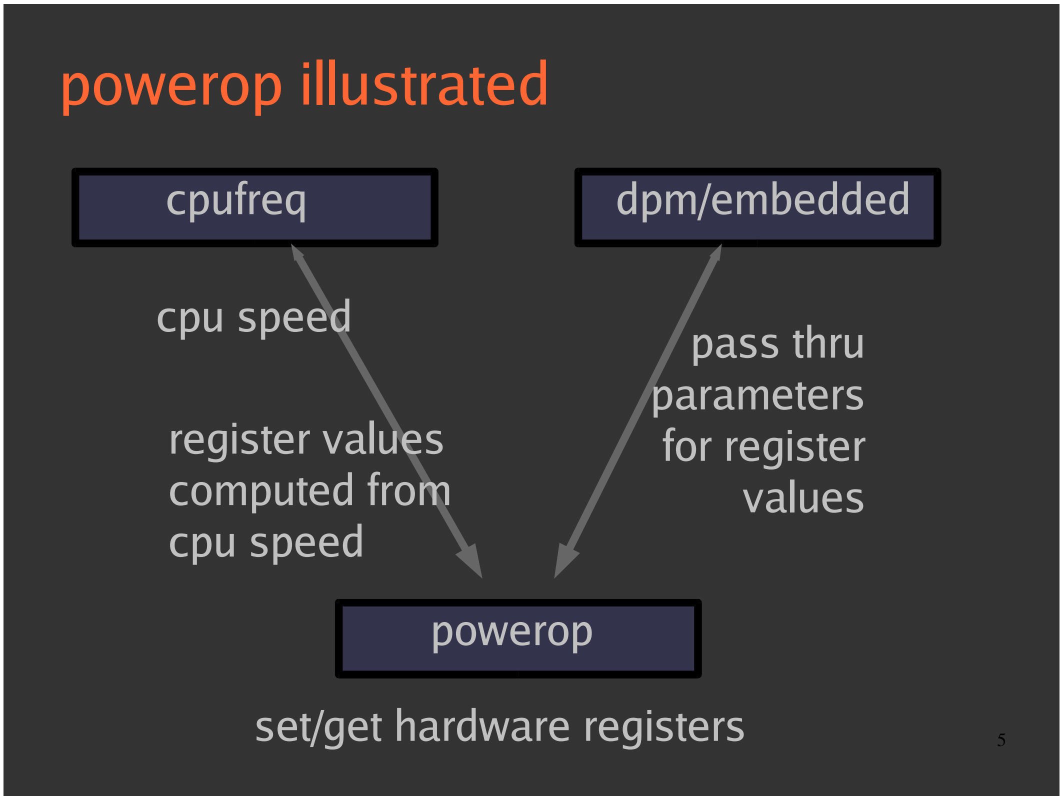



A proposed platform-specific API.

Suited for manipulating multiple power parameters independently, such as for Xscale PXA2xx Wireless Speedstep, OMAP, i.MX31...

Assumes we will continue to have cpufreq for desktop/laptop systems managed primarily by cpu speed (and often with ACPI, PM in BIOS) vs. embedded frameworks such as DPM (that manage more PM state in Linux). But it’s not certain that the two worlds cannot be merged into one set of s/w.

Could be subsumed by linux-pm directions toward a full PM stack. Found interest in tackling these and other topics when PowerOP discussed on list.




Hardware registers might include:

- multipliers and dividers that produce clocks for core PLL, bus, CPU, and peripheral-specific clocks such as LCD pixel clock
- voltage regulator values to set core voltage
- and pseudo-registers that set other behavior not easily described by a small set of hardware registers, such as clock domain autogating policy or suspend states

Valid combinations of these are an “operating point”. Where these come from, which ones available, etc. are the upper layer’s job.

## dpm maps states to operating points

policy P runs operating point O at state S

system states include idle, per-task states, sleep

conserve power during brief idle periods

apps can manage own custom state if desired

power state can be tied to scheduling priority



Much of the design from IBM Austin Research Lab, prototyped on cutting edge low-power PDA reference designs, heavily analyzed.

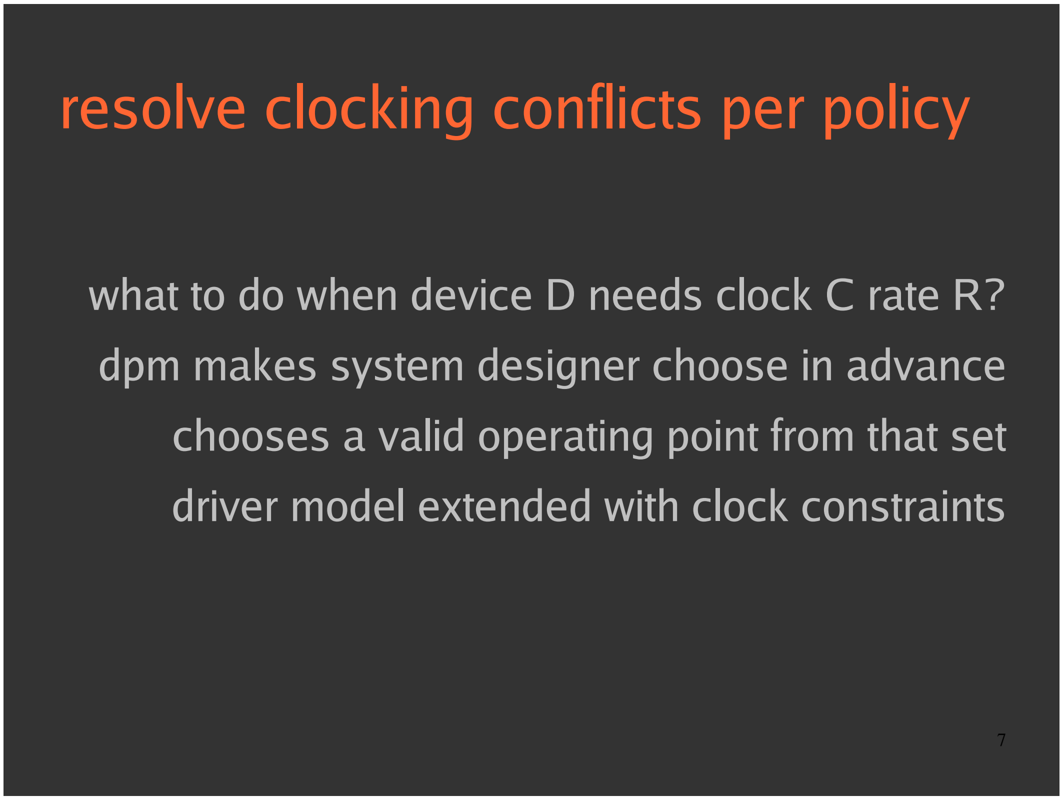
The system sets “states”; the “policy” determines what “operating point” to activate for that state.

Idle hook assume system can conserve power by modifying params during brief idle periods, as with multimedia playback.

All policy management normally in userspace.

IBM mpeg4 decoding example, watches rt deadlines and adjusts power/performance to meet.

Tying task power state to scheduler priority avoids PM priority inversion.




Dpm doesn’t try to resolve conflicts between device needs and current operating point on its own.

System designer creates sets of operating points that handle the possible device-constrained situations.

Add operating points that conserve more power when devices don’t need it, dpm chooses a valid operating point at runtime.

struct device has new field for constraining ranges of operating point power parameter values, identified by symbols for the associated clock and a range of values.

## device management is hot

formerly less complex, big savings

now multiple power and clock domains

with multiple power states and latencies

set policy via driver model or state->op style pm

platform bus probably needs extensions

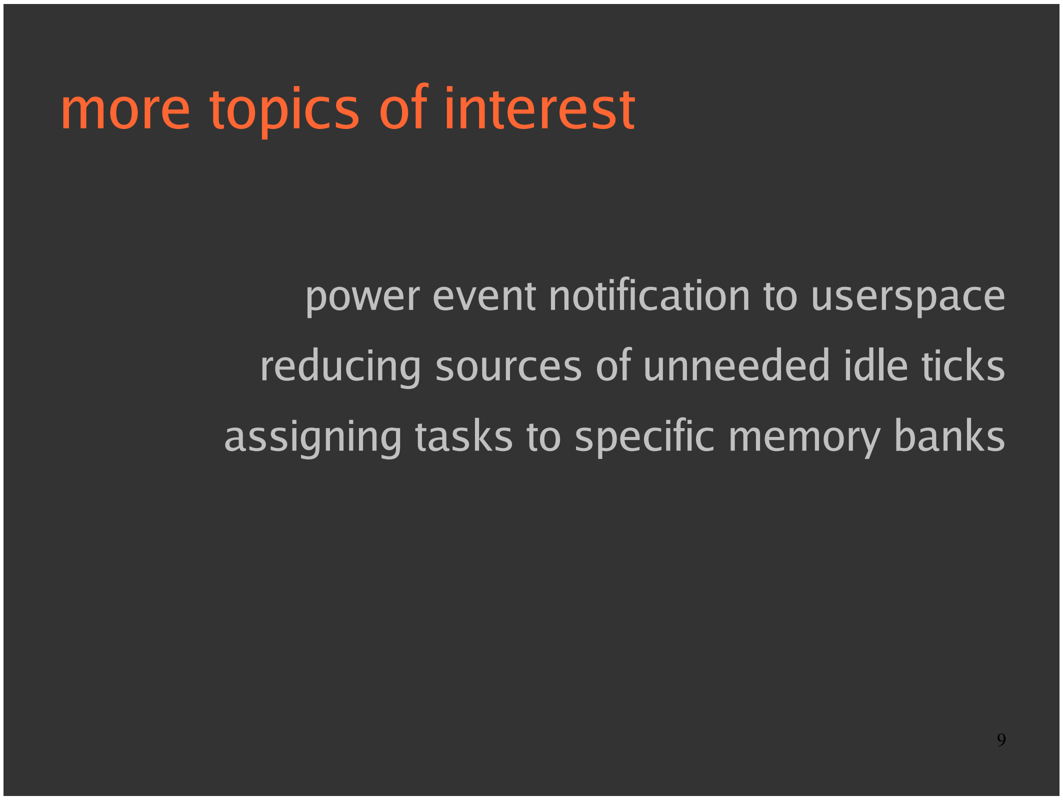
8

Power/clock domains and management needed is now making device pm at least as complicated as runtime dvfs.

Different ways of setting policies for the states of these being prototyped.

Tendency to move embedded devices to minimal "platform bus" and to move more processing into bus code makes it harder to add these types of features.

May need to extend platform bus with system-specific handling of PM features, capture actual bus topologies, etc.




ACPI uses /proc interface specific to ACPI, embedded can use something, should be kobject uevent / D-Bus?

In additional to “dynamic tick” / “tickless idle”, VST reduces periodic tasks when not needed, avoiding need to wakeup at the request of subsystem that does not need work to be done upon wakeup.

MTA Memory Type Allocation for assigning tasks to specific memory banks.  Can kill all tasks for a specific bank and power down that bank.  Based on NUMA support.

CELF member companies have been involved in work on some of these.