Energy Efficient Software

Invited presentation, EACO workshop 10-11, September 2014

Lee Smith, ARM Fellow

About me and about this talk

These days, I work mostly where technology meets business

- Technology has value when it creates or enables *business capabilities* that in turn generate *products*.

I spend most of my time interpreting technology for business-oriented colleagues

- Lawyers, salespeople, marketers, product managers, engineering managers...

ARM views of technology are informed to an unusual degree by the views of our most important partners and customers

- Our 2013 revenues were >$1Bn but our partners did many times more using our technology
- *ARM might "punch above its weight" in the industry but we also represent consensus*
  (Focussed around the boundary between differentiating and non-differentiating technology)
- ARM both leads and follows!

Inspiration and motivation

Why do *I* care about this topic *now*?

An historical *business* invariant for ARM

Since the foundation of ARM, directly or indirectly, every one of our market opportunities had energy efficiency, low energy operation, or low power operation as a central concern

- Not always the top concern but always there...
- Not always a direct concern – c.f. automotive electronics, set-top box, etc – the direct concern might be operation at high temperatures and/or fan-less operation at typical room temperature

A vision statement

(From ARM's 2012 Corporate Responsibility report)

http://ir.arm.com/phoenix.zhtml?c=197211&p=irol-csrhome

- We have a similar, employee-facing statement of our corporate vision...
- Note the juxtaposition of *energy-efficient* and *connecting the world*

More Data, Less Energy

Making Network Standby More Efficient in Billions of Connected Devices

http://www.iea.org/publications/freepublications/publication/MoreData_LessEnergy.pdf

In 2013, a relatively small portion of the global population relied on more than 14 billion network-enabled devices in homes and offices. As more people use a wider range of devices for increasingly diverse purposes, the total is expected to skyrocket to 50 billion network-enabled devices by 2020.

Left unchecked, by 2025 the corresponding energy demand would soar to 1,140 terawatt hours per year (TWh/yr) – more than the current annual electricity consumption of Canada and Germany combined. A vast majority of this energy would be consumed when devices are "ready and waiting", but not performing any particular function.

Drawn to my attention by ACM TechNews: *Energy Demands of Networked Devices Skyrocket, 16th July 2014* (http://technews.acm.org/archives.cfm?fo=2014-07-jul/jul-16-2014.html)

One of my modern heroes...

(Prof David J C Mackay, Regius Professor of Engineering, Cambridge University)

Sustainable Energy - without the hot air

http://www.withouthotair.com/c19/page_114.shtml

Chapter 19: *Every BIG helps*

(Chapter 19 begins by debunking the value of unplugging our phone chargers – they might sum to the electricity consumption of 66,000 households but that remains only 0.25% of UK consumption...)

A (or *The*) BIG problem...

(Busy doing nothing, idling the whole day through...)

- The energy consumed *busy doing nothing in particular* is spiralling out of control
- It's a BIG problem so worthy of our attention
- It's created by *connecting the world* (or, at least, the world's things...)
- *Aside*: I could not find a clear reference to the efficiency of low-power power supplies
  - 90% seems achievable (Google design for server power supplies)
  - Mobile phone chargers > 70% (several Google Scholar references) at extreme low cost
  - ➢ Very low 'off' power seems achievable
- Other segments such as Servers and HPC have similar first order issues (*explored in this presentation*)

I will assume that the hardware problems can be (mostly) solved, leaving software problems...

Finally, *Publish or Perish* reinvented

Received academic wisdom for 60 years

"Publish or perish"
(http://en.wikipedia.org/wiki/Publish_or_perish)
(emerged1932,?1938?)

MIT Media Lab reinterprets the wisdom 60 years on... after ~30 years of Internet

"Demo or die"
(Nicholas Negroponte, 1998)
(http://www.nettime.org/Lists-Archives/nettime-l-9807/msg00085.html)

And again after 30 years of the Worldwide Web

"Deploy or die"
(Joi Ito, 2014)
(http://www.ted.com/talks/joi_ito_want_to_innovate_become_a_now_ist)

Many fine technical ideas fail because we can't deploy them, a big personal concern of mine

Energy efficient software

A stack of concerns

Anecdote: iPhone 1 Battery Life Was Doubled in 6 Months

(According to its earliest customers...)

No new hardware or battery technology, one new firmware release

- The timescale tells us independent changes by individuals made the improvement!
  - *Software engineering lore*: $\{project\ elapsed\ months\} \geq 3\ \{person\text{-}months\}^{1/3}$

    $\{6\} \quad \geq 3 \quad \{\leq 8\}^{1/3}$
    - ➢ No independent, effort exceeded 8/6 people...
- Simple, local decisions about how software behaves can make a BIG difference to energy consumption
  - Most of the changes probably fixed "*energy bugs*" (see following slide)

Fast forward to 2014 – Carat on iOS and Android

https://amplab.cs.berkeley.edu/2012/06/14/carat-now-on-ios-and-android/

http://techcrunch.com/2012/06/14/carat-battery/

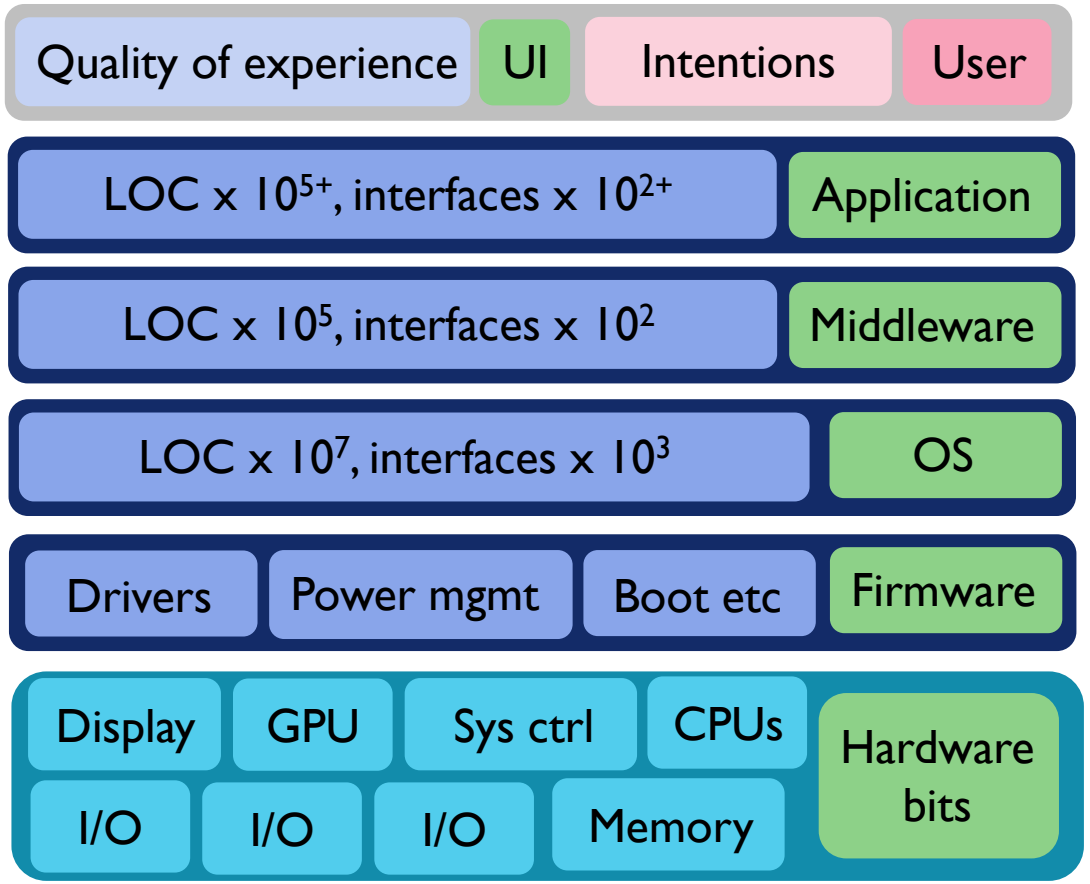
"Carat: The Brilliant App That Increases Your Battery Life By Showing What Other Apps To Kill"

- Identifies "*energy bugs*" and "*energy hogs*"
- Bugs (unsurprisingly) are very common...
- Hogs (unsurprisingly) require you to pay for what you get...

➢ Fixing *energy hogs* might require algorithmic changes or large-scale architectural changes

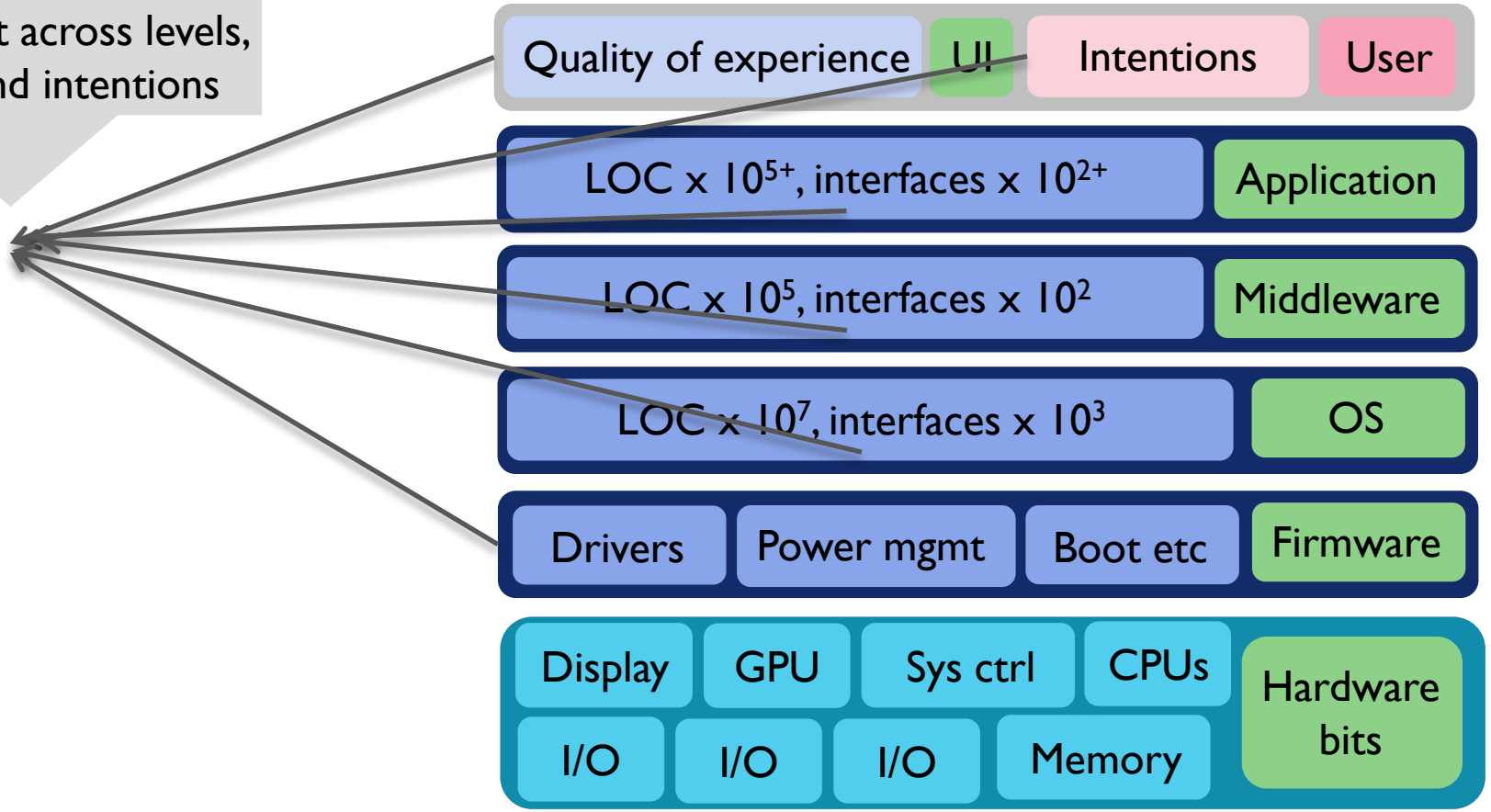
Hugely simplified mobile application stack

Some key challenges



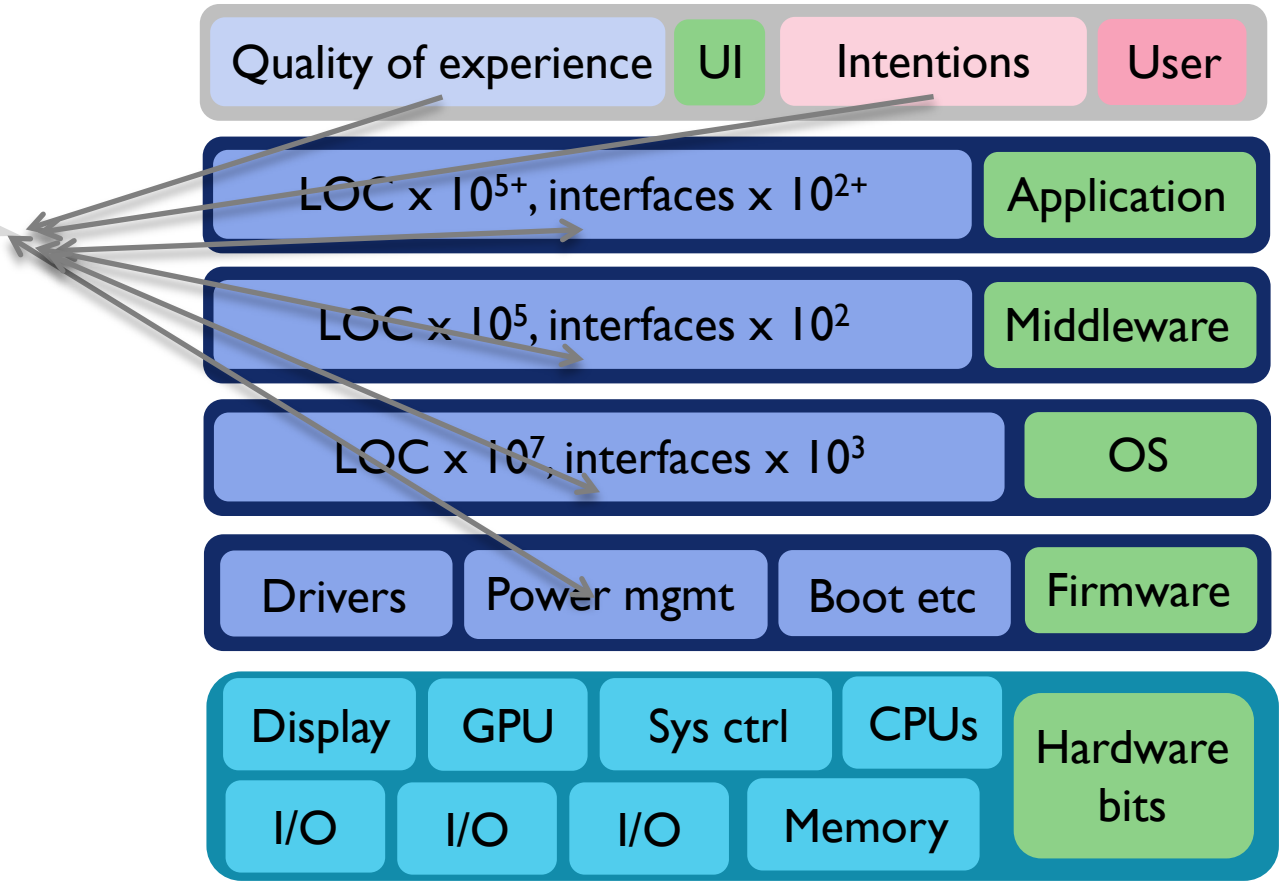
How shall we visualise what's going on?

Profiling, animation, measurement across levels, interfaces, conflicting policies and intentions



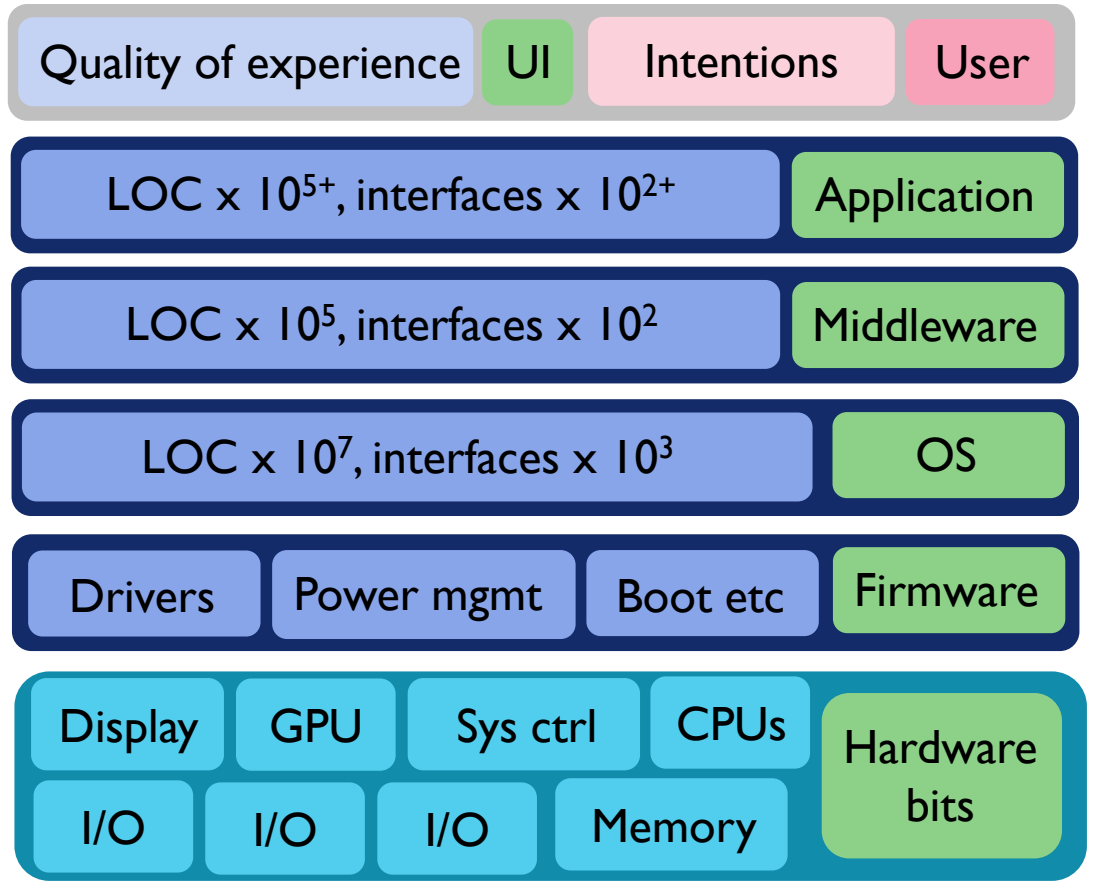
How shall we control power use?

Thermal monitoring + active control of power to avoid damage and meet the user's goals



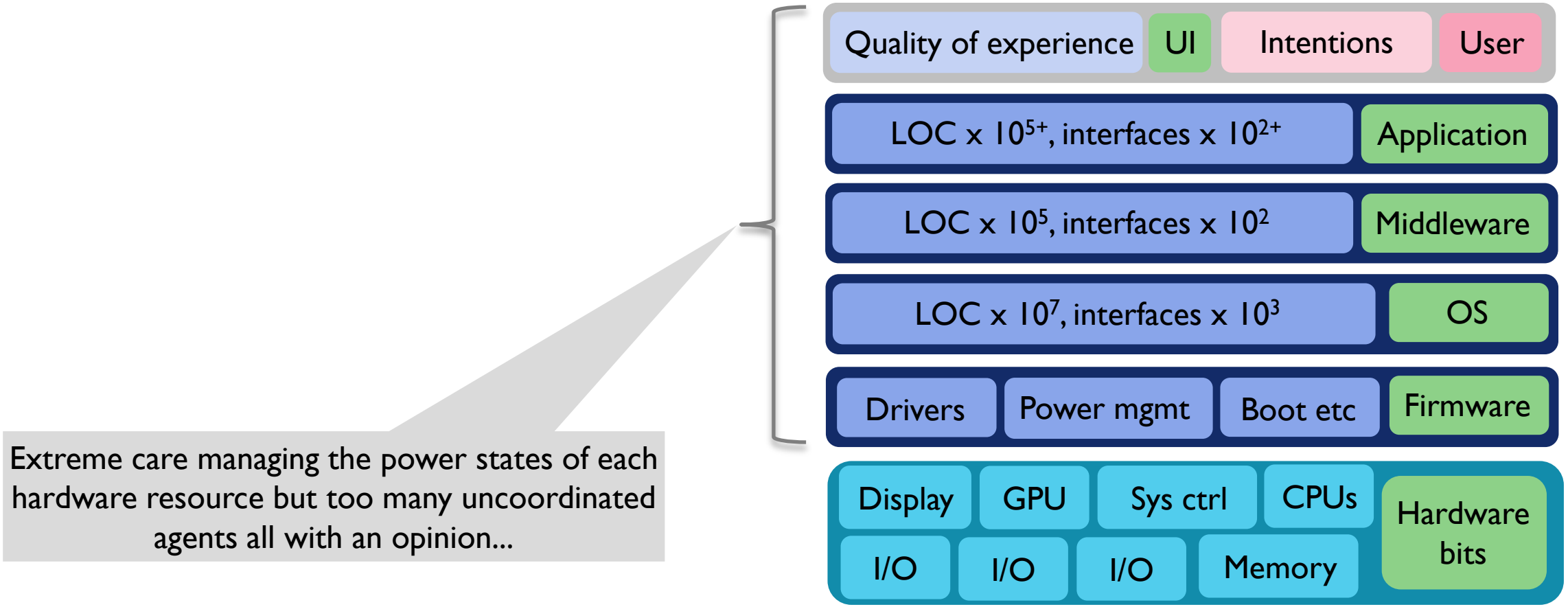
Energy-efficient software architecture or a mess?

Interfaces everywhere! Are they fit for purpose? How shall we model or reason about efficiency of software architectures? Matlab/Simevents?



Basic hygiene is harder than it needs to be

(even for 'turn it off when unused'...)



Issues requiring attention within a software component

Placement of code and data in the memory hierarchy

- There can be big ratios between the energy-cost of accessing on-chip SRAM, on-chip cache, on-chip FLASH, off-chip FLASH or DRAM
- Unfortunately, the more expensive accesses also suffer longer latencies which cause *busy idling*, not just by the CPU but by whole servers or HPC 'nodes' – the energy cost can be huge...

Design and implementation of the application, at every level of abstraction

- Choice of algorithm, mapping the algorithm to the available hardware, low-level coding practices

Quality of code generation by the compiler

- A minor factor until all others have been exploited or ruled out

(I admit that there is a discussion to be had here around what (only) compilers can do versus what (only) application authors can easily do in the context of what can (not) be feasibly funded and deployed...)

Some top-level models
(In the spirit of Every BIG helps)

Segment by segment analysis

High Performance Computing
(Big picture model)

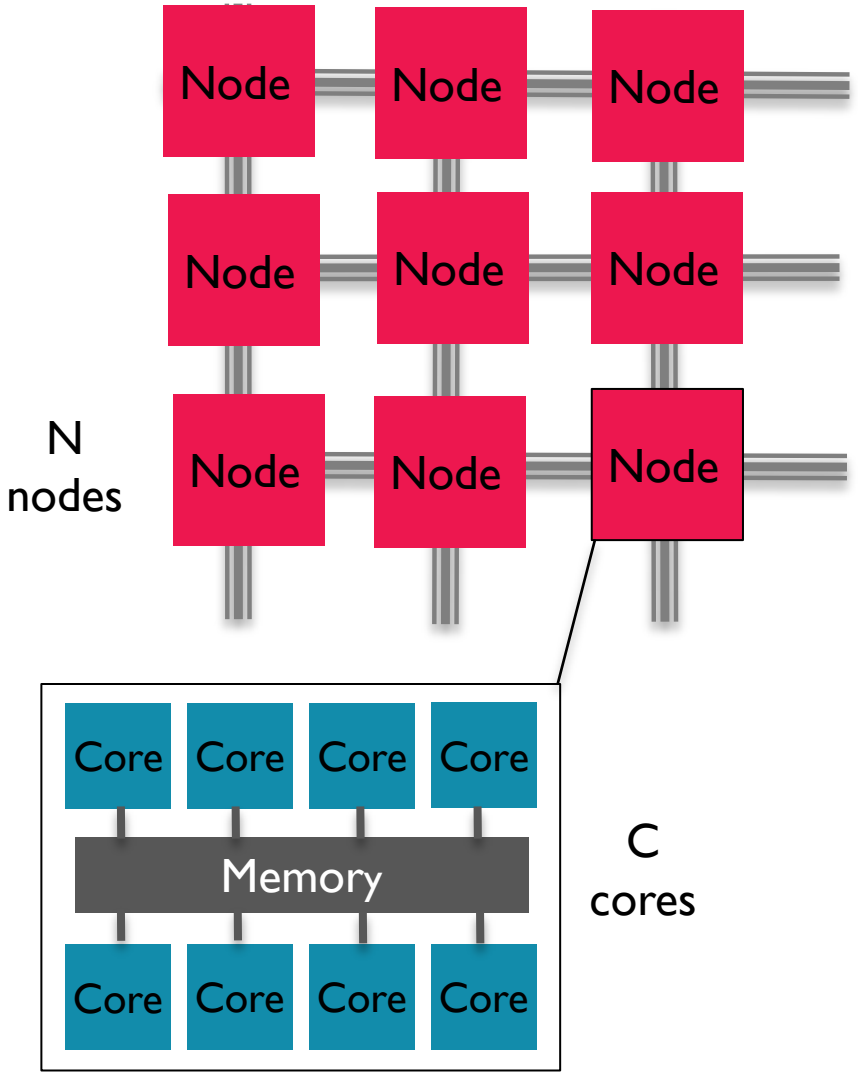
- An application is *scaled out* over many nodes; nodes exchange data over a network using MPI (*Message Passing Interface*)
- Nodes are units of power management and cooling

$$E^{Application} = N * W^{Node} * T^{Node} \qquad T^{Node} = T^{Busy} + T^{Idle}$$

- Energy is minimized when $T^{Idle} \rightarrow 0$ and $T^{Comm\text{-}busy} \rightarrow 0$
  - Node processing 100% overlapped with inter-node communication, which limits parallelization
  - MPI (or similar) busy overhead ($T^{Comm\text{-}busy}$) minimized (*also limiting*)
- Each node is a NUMA, shared-memory multi-processor; application scaled out using OpenMP® extensions, or similar

$$E^{Node} = (C * W^{Core} + W^{Memory}) * (T^{Busy} + T^{Idle})$$

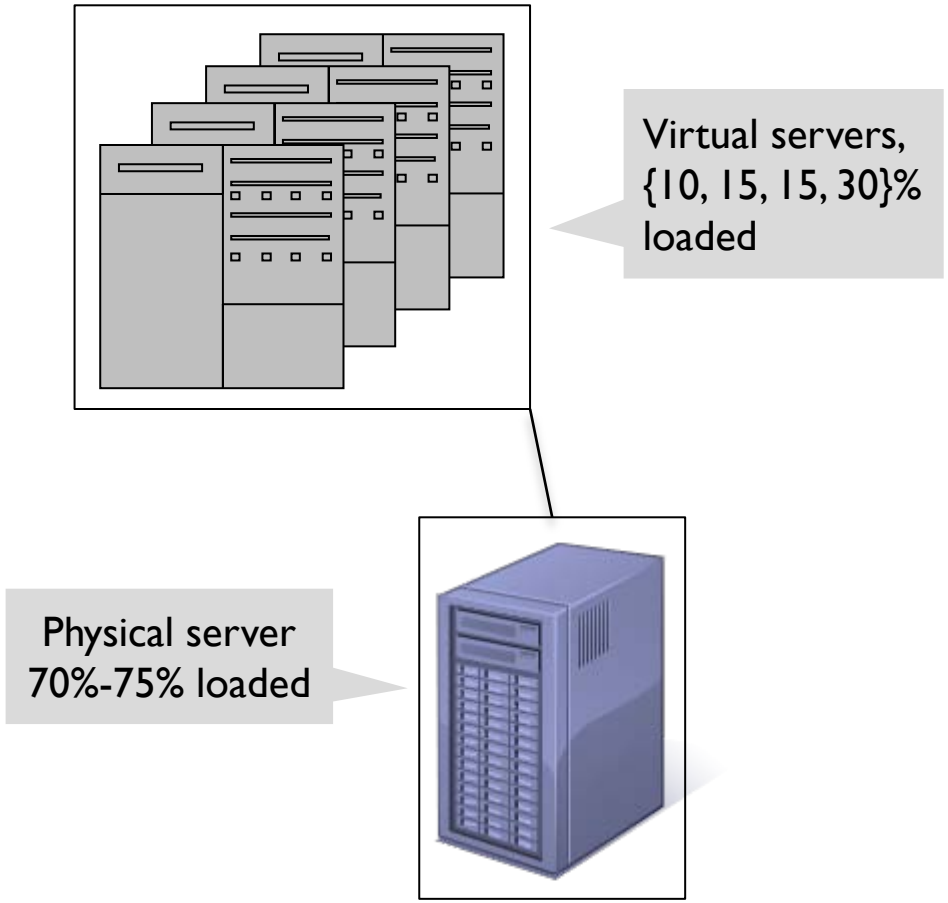
  - Important to minimize idle (stall) time, minimum per-node energy is almost 100% aligned with maximum node performance



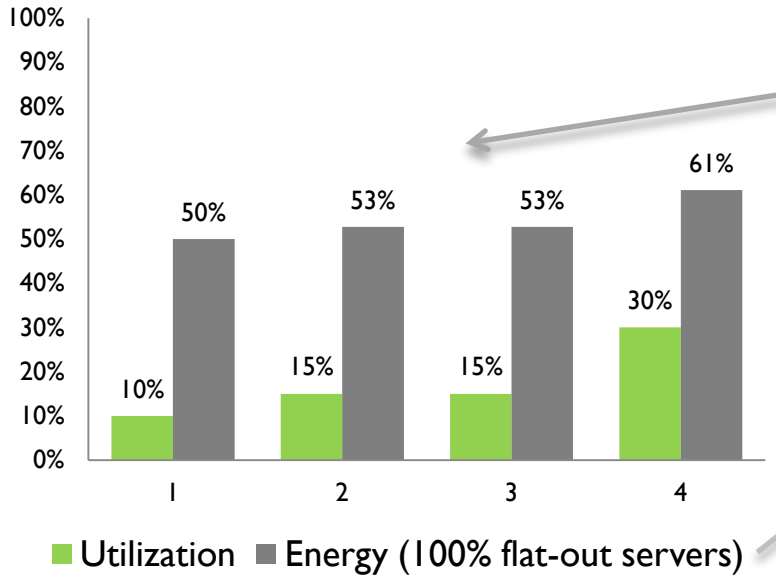
The OpenMP name is a registered trademark of the OpenMP Architecture Review Board.

Scale-out servers
(Idleness remains the enemy...)

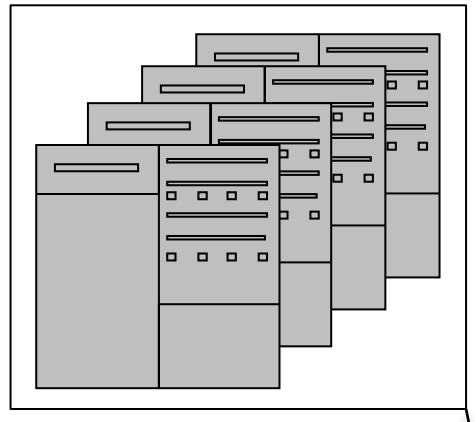
- Historically, server utilization is low – 10%-40%
  - Difficult to improve without threatening response times
- Virtualization lets a data centre pack multiple virtual servers onto fewer physical servers
  - Server power varies 50%-100% as load varies 10%-90%
  - Driving physical utilization up to 70% reduces the demand for physical servers by 1.75-7x
  - A double win for *Total Cost of Operation* (TCO) even though virtualization overhead reduces the gain
  - Virtual machines can be moved between physical machines at software/network speed (critical to managing virtualization)

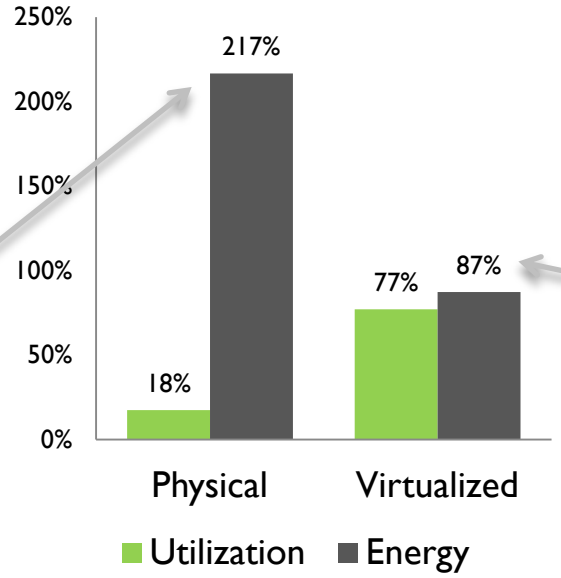


How virtualization saves energy



Servers loaded
{10, 15, 15, 30}%





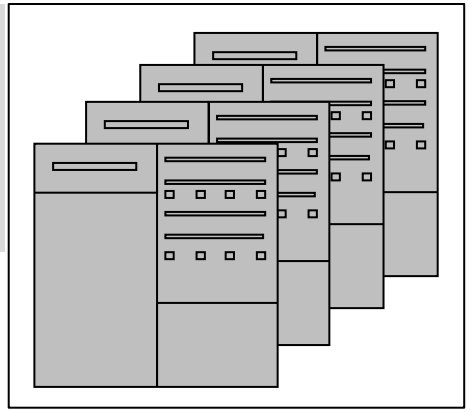
Energy ↓ 2x
Servers ↓ 4x
Capital (servers + software) ↓ >2x (guess)
TCO ↓ 1.6x

Big picture: Business model, taxation, and regulation matters

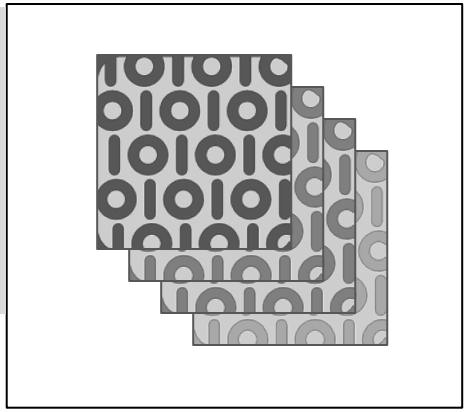
- *Assume* (IaaS):
  - Changing 1 line of application code costs $10, changing 100 lines increases performance by 10% (so 10% fewer instances)
  - Scale-out Linux instances cost 25¢/hr (http://en.wikipedia.org/wiki/Amazon_EC2#Cost...)
  - Project risk, NPV, etc double the break-even cost
  - Spending $1,000 to save 10% is justified by 80,000 instance-hours, ~10 instances for 1 year, 24/7...

  Who has enough scale to justify the cost of optimizing applications?
- *Contrast PaaS:*
  - The service provider has the scale *and* the business incentive (e.g. *Salesforce.com, etc...*)

Infrastructure as a Service
Rent virtual machines

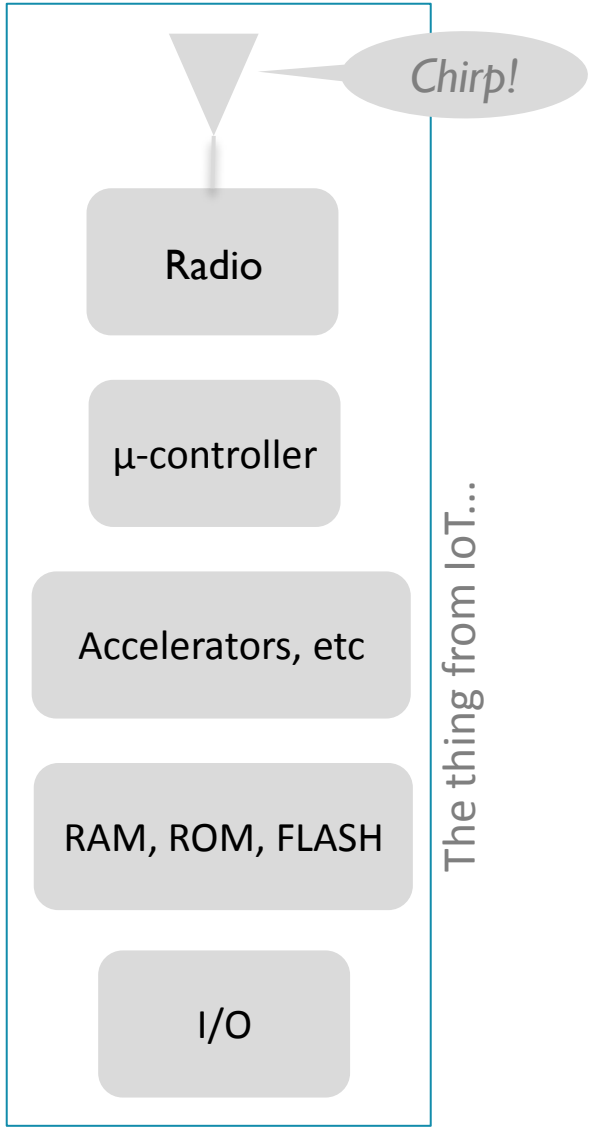


Platform as a Service
Rent application instances



Things (ain't what they used to be)

- *Radio* – < 10mW Rx/Tx (when chirping), < 1µW sleep
  - Achievable today – see for example http://sunrisemicro.com/
- *µ-controller* – < 10 µW/MHz (core only)
  - ARM® Cortex®-M0+ achieves this, equivalent to ~ 10-20pJ/instruction (*excluding memory-access cost*)
- RAM, ROM, FLASH, I/O
  - A less happy story…
  - On-chip FLASH ~100pJ/32-bit access? (*10 instructions' worth…*)
  - SRAM ~50pJ/32-bit access?
- Active/dormant cost ratios 100-10,000 so *busy-idle and inactive-not dormant* are the mortal enemies of energy efficiency
- Placement of code and data in memory is critical to efficiency

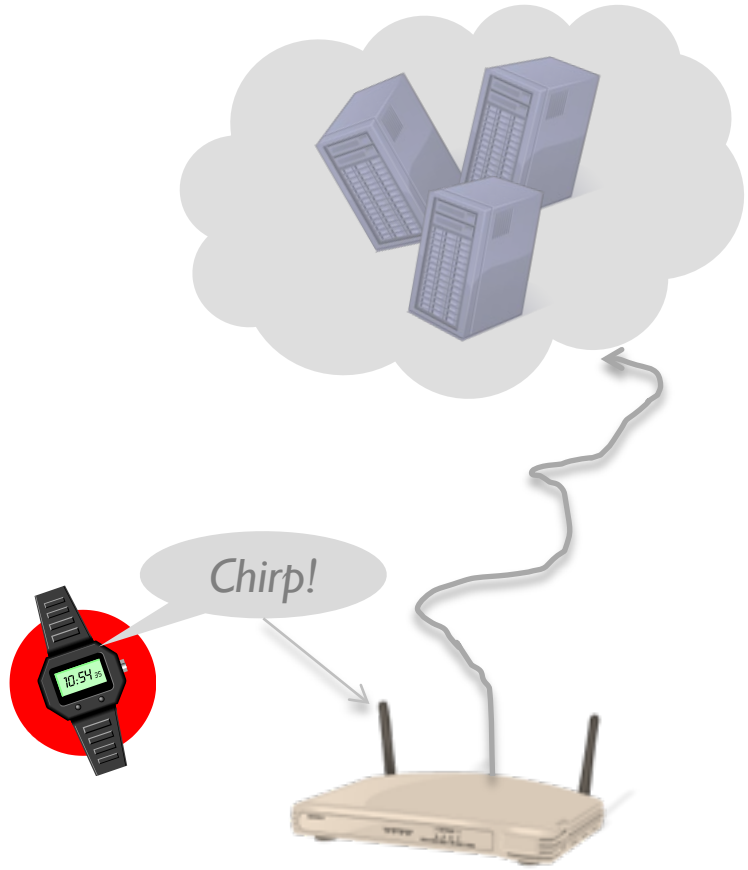


IoT enables the Cloud and the Cloud enables IoT

IoT is all about data aggregation and analysis, in the Cloud

- Every IoT transaction will have an echo in the Cloud...
- Energy efficiency is an end-to-end concern spanning the *thing*, *services* in the Cloud, and all the *things* en route to the Cloud...
- Energy efficiency is now an architectural problem and we will need efficient architectures not just efficient *things*...

Network architecture, service architecture, business architecture must all favour energy efficiency...



Software meets hardware

How it goes horribly wrong close to the metal

Some IC fundamentals

- Model the power state in any cycle as Off < Ready < Busy-idle < Busy}

  Each state includes the power of its predecessor

  - P_{Busy} is the power when functional units are typically busy (short of *power-virus* busy...)
  - $P_{Busy\text{-}idle}$ is the power when unused functional units are clock-gated, rest of system is running
  - P_{Leak} is the 'leakage' power dissipated by being powered on and *ready* but not clocked

$$E = P_{Busy} * T_{Busy} + P_{Busy\text{-}idle} * T_{Busy\text{-}idle} + P_{Leak} * T_{Ready}$$

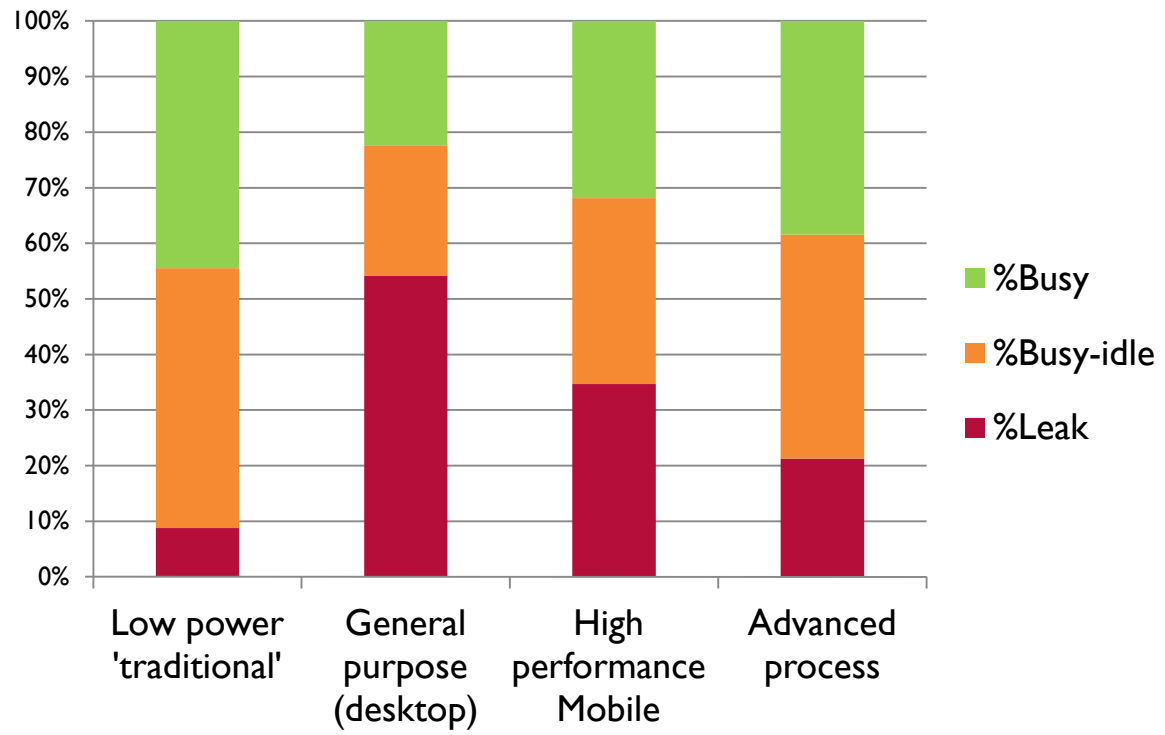
- E is minimum when there is no *Busy-idle* time and no *Ready* time or no *Leakage*
  - Ideally, run fast then enter a lower power ('*Off*') state or reduce voltage and frequency and run slower to eliminate *Busy-idle*

  Alas, limitations of DVFS and state save/restore/retention overheads often favour Busy-idle

Run, Busy-idle, and Leakage power vary

These proportions are indicative

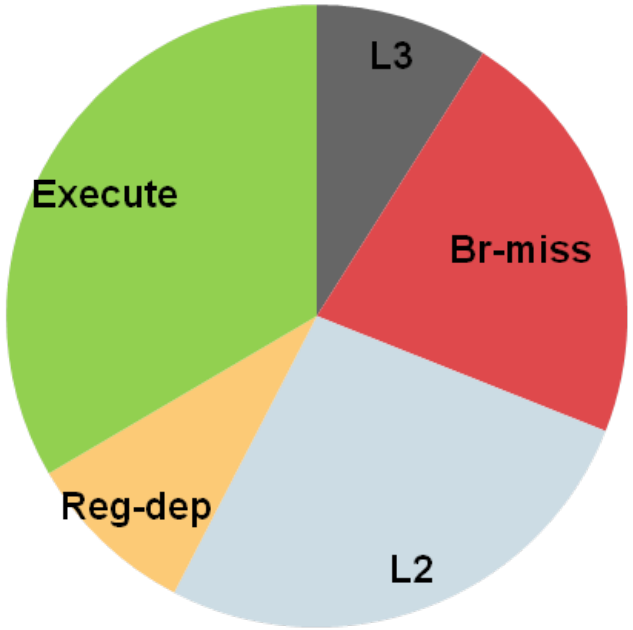
- Really good low-leakage processes show barely measureable leakage power
- *Busy-idle* power is significant in all cases
- Proportions depend on the ASIC and the system design not just the process
- Combine these stacks with cycles per instruction (CPI) to give energy per instruction (EPI)



CPI for Firefox 3.5 running Bbench on ARM® Cortex®-A8

(Some software fundamentals – real software sucks energy...)

Cause	Contribution
Access to L3 memory system (DRAM)	0.26
Branch misses	0.64
Access to L2 cache (55% TLB misses, 45% L1 misses)	0.77
LDR to dependent register delay	0.26 (estimated)
Total of the above	*~1.93*
Execution CPI absent above factors	~0.97
Total CPI	**2.9**



CPI = 2.9

How much energy might a mobile browser waste?

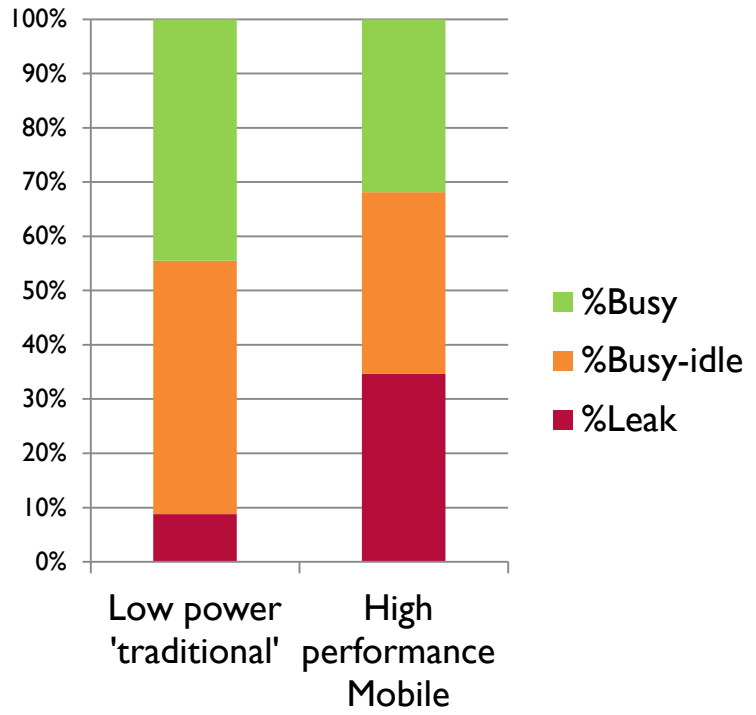
(*Two example mobile fabrication processes, 'typical' ASICs...*)

*Total EPI = 0.97 * 1.0 + 1.93 * %Busy-idle*

- Low power 2.03 (48% useful)
- High performance 2.28 (42% useful)

➢ < 50% of average EPI is used usefully; >50% *busy idling*

Corollary: fast code (few stalls) is good code; scaled performance code is good code (fewer stalls)



➢ Leakage is another story (*too long for this presentation*)

➢ So is GPU + display... CPU just one small component

An aside on out-of-order processors and the EPI model

A modern out-of-order machine is never truly Busy (unless power virus is running) and rarely truly Idle

- This matters little if a crude power-state model is composed with a cycle stack
  - The degree of busyness averages into the CPI number
  - We do rely on *Busy* power being reasonably linear between min-Busy and max-Busy

An aside on *busy-idle* versus truly idle

To some extent I have obsessed about the energy wasted in the busy-idle state, partly because it is a problem that spans HPC to IoT

- The concern I raised initially (IEA) was about devices being *on, ready*, but *doing nothing*
- Models (or levels) of idleness include
  - *Busy-idle* – stalled waiting on memory – ~50% dynamic power, 100% leakage
  - *Wait For Interrupt* (WFI), *clock stopped* – 0% dynamic power, 100% leakage
  - *WFI, state retention* – 0% dynamic power, much reduced% leakage
  - *WFI, power off* – 0% dynamic power, 0% leakage

  All applying at the component (e.g. CPU, GPU) level... IoT devices need to get to WFI, power off

Summary and open problems

Summary
(Energy-efficient software)

Every Big helps

- Find and tackle the big issues first...

System architecture is critical

- We can design efficiency in or out at the topmost level...

Busy idling is evil

- To first order, faster ⇒ more energy efficient so ideally run fast and stop but...
- State save/restore and state retention (stopping) also have overheads...
- Heterogeneous computing is another answer that swaps one evil for several others...
- Dynamic voltage and frequency scaling let us run slower more efficiently but scaling is limited...

Placement of code and data in the memory hierarchy is critical from HPC to IoT

- Critical to performance and energy efficiency (often doubly so)

Some open problems
(Energy-efficient software)

- *Lack of common ontology* impairs deployment, time to market, creation of reusable tools, education of engineers and programmers...
- *Lack of reliable, up-to-date, citable data* about energy and power consumption are problems for business and academics alike
  - *But these data are very sensitive in the industry... Need academics to do more measurement!*
- *An hierarchy of independent agents, policies, and intents* makes software fights itself or fight the hardware or firmware
  - *Xeon power management versus Linux power management now well known and rather tragic*
  - *Needs better, global coordination frameworks that allow independent agents to share policy intentions and resolve potential conflicts between them*
- *End-to-end energy-efficient system architecture*
  - *How can we define it, model it, implement it?*

The end

fin, Ende, loppu, τέλος, конец