The Path to Best Effort

Vaughn Joy Mannon*
ERC
vjm@livingcomputation.com

ABSTRACT

Research and development in best effort computing has taken off since the VSW, in an emerging major technological transition that may ultimately rival the digital computing revolution itself. Here at the end of the so-called First Computing Century ('CC0', 1940–2039), we reflect on computation's long fixation on the ideas of hardware determinism and software efficiency, before their incompatibility with scalability, robustness, and security was widely appreciated. We go beyond questions like 'What were they even thinking?' and 'How could anybody stand to compute like that?', to highlight hardware, architecture, software, and systems innovations behind best effort computing, with its crucial reframing of computer security as an aspect of robustness and synchronization rather than correctness and isolation. Finally, we celebrate the emergence of the microcomputome and 'syncurity' as signs we are maturing beyond hardware determinism and the belief in the existence of the Last Bug.

Keywords

best effort; indefinite scalability; syncurity; microcomputome

1. WHAT WERE THEY EVEN THINKING?

Despite all the terabytes of the historical record—plus the schoolbook summaries, crack-n-hack movies, on and on—it is difficult, today, to fathom how truly bad computer security used to be, and how deep was the denial of all involved. It really *seemed okay* that literally one tiny physical location in space—a single 'CPU' chip—would touch everything from the lowest level hardware configuration details, to the user's most intimate medical and financial information, to all the catpix and the scum of the Internet. All of it. In one place.

The official reason for complacency about such an idea, mostly, was efficiency. Nearly a century ago, the ENIAC digital computer weighed 30 tons, ran about five thousand cycles per second [51], and broke down weekly or daily. Using efficient programs was an imperative—and given the machine's inflation-adjusted construction cost of about ₿1B, you really did have to make do with just the one.

In those early days, an implicit but hugely consequential deal was struck between hardware and software: *Hardware provides reliability; software provides desirability.* Hardware had the job of turning the unruly physical world into an utterly flawless automated logician. Software's job was turning that logician into enough money to pay for the hardware and the software both. To do that, software was all about producing valuable outputs *correctly* and *efficiently.*

For the ensuing half century at least, that deal was the unquestioned backbone of the digital computing revolution. For most thinkers, the idea there could be a beneficial purpose not just for inefficient software, but even for strictly incorrect software, was quite literally inconceivable.

How times have changed!

1.1 Best effort computing

Among all the lessons of the events surrounding the VSW, one is this: Putting all the responsibility for reliability on hardware is irresponsible. It gives software a free pass even though reliability, resilience, robustness, and security are whole-system properties. In digital computing's New Deal, hardware provides best-effort reliability (but may fail arbitrarily if necessary), and software provides best-effort desirability (and hiding hardware failures where possible).

It may not sound that different. If a computer scientist could be reading this report 25 years ago, they'd probably be nodding sagely and thinking something like: *Sure, fault tolerance is nice; if it's really needed.* But of course best effort is not fault tolerance—and it impacts *everything.*

1.2 Report outline

The main content of this report is summarized in Figure 1, and we beg the reader's indulgence in advance for the avalanche of material—the almost-uniquely systemic nature of computer security demands design considerations across the computational stack.

The bulk of the text, in Sections 2–4, is a review of milestones and challenges—in architecture and hardware, software and systems, and security and society, respectively—on humanity's long path to best effort.

In section 5, we survey the recent history of best effort computing, looking at the technical details behind its famous breakthrough products. Section 6 concludes with brief thoughts on distributed systems and the microcomputome, and looks forward to CC1, our second computing century.

*A pseudonym; see acknowledgments.

EALC 5th Annual Meeting, August 8–11, 2039, Albuquerque, Earth

doi:10.5281/zenodo.1304010

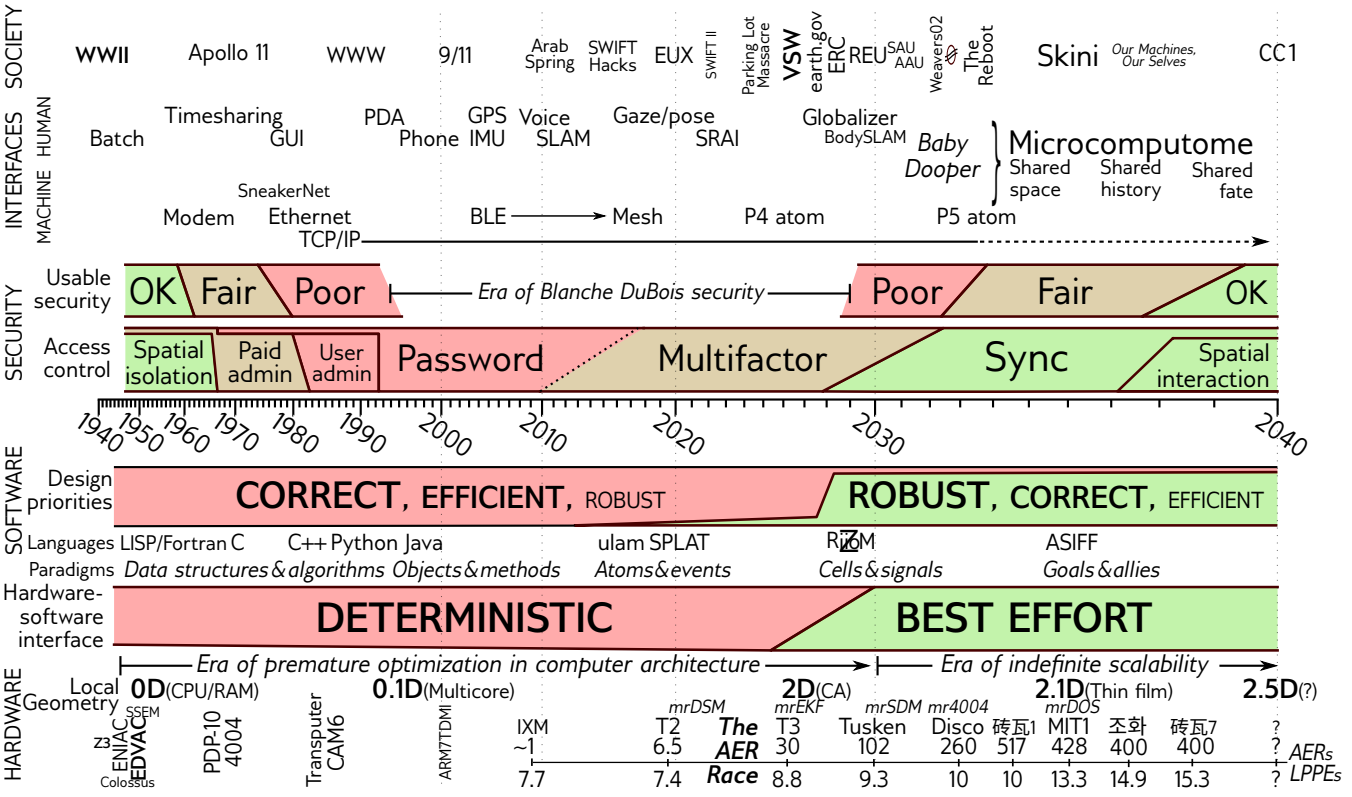

Figure 1: A timeline of the 'First Computing Century' (CC0). See entire report for details.

2. ARCHITECTURE AND HARDWARE

The *architecture* of a system is its designed structures that change most rarely. Though such structures are typically the biggest factors in how a system does and doesn't work, they are often taken for granted by system users—and they are where the path to best effort begins.

2.1 CPU and RAM

In the 1945 EDVAC draft report [2], von Neumann describes all of a programmable digital computer's elements: the central processing unit operating serially, the passive random-access memory holding both data and 'orders', the binary number representations, the need for flawless operation. He considers parallel processing but rejects it for simplicity; similarly he mentions errors and fault tolerance but those parts of the report remained unwritten or were lost.

Though decades of improvements and refinements followed, that original CPU/RAM architecture for *serial deterministic computation* remained essentially unchanged while becoming a towering success, triggering then dominating CC0.

2.1.1 The birth of good and evil

To help make the new digital hardware commercially viable, *timesharing* (e.g., [4]) was developed, allowing many users to access expensive machines like the DEC PDP-10 [6, 5] virtually at once. Crucially for our story, timesharing demanded trust boundaries and access controls *inside* the machine; physical machine security was no longer sufficient. Despite the creation of user accounts, administrator privileges, passwords and permissions, the basic programming model remained, but random-access memory was no longer *quite* so randomly accessible. Or at least, it wasn't *supposed* to be—but since the new security boundaries themselves were controlled by *more software* running atop fully RAM hardware, computer viruses (e.g., [16, 29]) soon followed.

The economic leverage offered by the programmable computer derives largely from *delayed functional commitment*: Huge one-time hardware development costs can be amortized over many different purposes that can be determined long after the machine has been designed, built, and sold. From our vantage point here at the end of CC0, we can understand that principle, and we can admit that alternatives are much more visible in hindsight, but still we conclude that deferring security commitments was a grievous mistake.

2.1.2 The rise of personal computing

In parallel with the timesharing of expensive systems, device technology improvements—especially for making amplifiers, or 'gates', suited to digital logic—were yielding relentless hardware cost reductions. The gate size kept shrinking as the gate count kept growing, from a few thousand in the Intel 4004 in the 1970s [26] to a few billion by the late-2000s. Data widths, RAM sizes, and CPU clock speeds kept increasing, and new capabilities became possible, then affordable, then universal, with each passing generation.

It was a golden age for computers. People of increasingly modest means could buy one. And then a new one. And inside other products, many more.

2.2 The twilight of hardware determinism

By the early 2000s, however, challenges were appearing.

2.2.1 The end of the MHz race

The industry's ability to increase CPU clock speeds—

which for decades had been their favorite performance improvement technique—was stalling not far above 3GHz, due in large part to power consumption and heat dissipation problems. As an alternative, limited amounts of true parallel processing were deployed, and the now-dethroned 'central' processor was rebranded from 'CPU' to 'core' (even though that was a previous name for *memory*).

Although such 'multicore' machines did provide performance improvements, they brought other problems as well. Parallel processing raises the sorts of coordination issues that caused von Neumann to avoid it at the outset. Computer architects developed *cache coherence* to maintain a weakened version of the deterministic execution that software developers relied upon, but its hardware costs grew rapidly with the core count, and even with it, notoriously hard-to-find "threading bugs" became commonplace.

2.2.2 Supercomputer warnings

The high-performance computing (HPC) community used software synchronization mechanisms like MPI [27] to avoid threading bugs while running a program simultaneously on up to tens of thousands of cores, but as the machines grew, hardware failures became increasingly frequent, reminiscent of ENIAC's problems half a century before.

Large-scale computations were typically 'checkpointed' to disk at regular intervals, to be restarted after a crashed computing node was fixed, but as researchers ran the numbers for an 'exascale' class machine—rated for a billion billion calculations per second—it seemed the machine might end up spending most of its time doing checkpoint-restart and not actually advancing the computation.

Alternatives were explored [34], but node crashes were not the only problem. As bad or worse was the growing risk of "silent data corruption" [40]: When supposedly deterministic big computations were rerun, sometimes they produced different outputs, due to *undetected* hardware errors.

Hardware's gold-plated reliability guarantee was failing—because it had never actually been unconditional anyway. Device manufacturers could provide whatever *non-zero* failure rate you chose, although eventually with eye-popping prices—but no matter what, if you ran enough of your chosen hardware long enough, it *would* have undetected failures.

2.2.3 Escaping the trap of hardware determinism

The experience of the HPC community was a canary in the coal mine for a systemic architectural problem in computing: **Global hardware determinism does not scale.**

Though a great start for small systems, hardware determinism served society less and less well as machine sizes grew and grew—but as the *core principle* of CPU + RAM architecture, it was virtually immune to incremental change.

But, a competing architectural core principle could arise.

2.3 Indefinitely scalable architecture

Although it gets tiresome to keep starting historical computing stories with John von Neumann, here we are again.

2.3.1 Cellular automata

Not only did von Neumann understand the limits of hardware determinism (see Section 4) and suggest alternatives based on thermodynamics [1], he also did early work on *cellular automata* (CA) [3], which ditched CPU+RAM in favor of many small *sites*, each with both processing and memory. The sites are arranged in some chosen spatial pattern, and interact with each other only within some local *neighborhood*, with no option for random access to memory. An *update rule* determines how a site changes its state depending on the state of its neighborhood. Overall, a computation is defined by a site update rule, plus initial states for all sites.

Von Neumann's CA interest was in formal properties of self-reproducing machines, rather than manufacturable computer architectures, and much subsequent CA research was likewise mathematical in nature. There was, however, recognition that CAs could be useful for scientific problems involving spatial dynamics—such as modeling fluid flows in 2D or 3D—and some hardware 'cellular automata machines' were designed and built in the 1980s and '90s [17, 24].

Such cellular automata models and machines presumed *synchronous* updating: Each site updated simultaneously based on the prior states of its neighbors. The assumption of synchrony was viewed as minor because several authors had proven that an *asynchronous* CA could always simulate a synchronous one, but as was pointed out in 2013 [43] such constructions demanded global hardware determinism. *Synchronous* CAs, like CPU+RAM, were only finitely scalable.

2.3.2 Hardware tiles

The primary concern of that 2013 paper was using CAs not as formal mathematical objects but to shape computer architectures that could finally escape finite scalability. To do so, both *asynchronous* operation and robustness to *even undetected* hardware errors would be required.

For convenient handling and to amortize hardware overhead, a 'subarray' of asynchronous CA sites could be collected into a hardware 'tile', and then *any* number of those tiles could be connected into an ever-growing machine, until we "run out of money or real estate or power or cooling" [49].

The "bespoke physics" paper, also published in 2013 [45], proposed several metrics to quantify and compare indefinitely scalable tiles, the most well-known of which is the *average event rate, scalable* (AER_s) defined as the average number of events per site per second, measured across a grid large enough that edge and intertile effects wash out.

2.3.3 The start of the AER race

Hardware tiles for computing date back at least to the Transputer systems (e.g., [13, 21]) in the 1980s, although the perspectives on software were quite different. The first tile in the direct line of descent to today's notions of indefinitely scalability was the "Illuminato X Machina" (IXM)—built using relatively old ARM7TDMI processors, and marketed briefly in 2009 [31]. Here we assign it $\sim$1 AER_s as an honorary degree, but since the event-processing software and benchmarking metrics were developed a few years later, no actual scalable values were ever obtained for the IXM.

The report on the T2 tile [55] did include hard numbers, setting the bar at 6.5 AER_s on `DReg` physics, although with unreported and apparently significant temporal distortion. However, little follow-up occurred until after the war, when simple but extremely tough signal processing systems—at first using mobile, robust Kalman filters [61] running on T3 tiles [60]—were demonstrated. Public and then private funding opened up, launching the now-well-known "AER race" among several groups, and producing a series of increasingly capable indefinitely scalable tiles [66, 72, 74].

The champion tile at present is the "TILE7" [76], offer-

ing a banquet of luscious specifications including less than 1 PPB spatial anisotropy, and less than 10% mean temporal distortion running at the emerging standard of 400 AER_s. And all that while scoring an amazing 15.3 on log Peak Power Efficiency (LPPE)—which is a 100x improvement in only two years, compared to the MIT1 tile [74] that debuted stochastic resonance for programmable transitions. TILE7 has yet to move into production, and initial prices will be high, but turning 400 AER_s cool to the touch opens up exciting possibilities indeed. Your author can barely wait.

2.3.4 *Indefinite scalability from hardware to software*

A programmable computer architecture defines an interface between hardware and software, creating roles for each. Although computer engineering is now brimming with radical innovations for indefinitely scalable tiles, the early best-effort designs used entirely standard hardware technologies. The real question was not *'How should we do it?'* but *'What should we want it to do?'*—and answering that took experience with best-effort software and systems, our next topic.

3. SOFTWARE AND SYSTEMS

The massive software systems built during `CC0` were absolutely stunning achievements. Whether measured by the number of 'moving parts' that they orchestrated in memory, or by the degree of the input-output nonlinearities they created during execution, those systems were, by far, the most complex machines ever manufactured by humanity.

3.1 Correctness and efficiency only

What those systems were *not*, however, was *correct*.

As mentioned in Section 1, the 'correct and efficient' software imperative is as old as digital hardware. However, the realization that "it was not so easy to get programs right as at one time appeared" is nearly as old [14]—and in the end, no matter what the purists wished and hoped, software's ultimate goal of being *useful* almost never required it to be literally, completely, formally, *strictly* correct.

The glory of `CC0`'s software research and development was its data structures and algorithms, evolving programming languages and computing paradigms, the software engineering methodologies, best practices, and so forth, as suggested by the 'Software' timeline in Figure 1. The widespread sentiment today is that those successes were not about *achieving correctness* but *managing complexity*, in the face of staggering exponential mountains of interacting design decisions in every significant piece of software.

For most of `CC0`, however—in programming classes and texts, in papers and discussions, and in the deepest intuitions of computer scientists and programmers—there was a ubiquitous belief that software is really about ***C**orrectness and **E**fficiency **O**nly* (CEO). As a software design priority, "robustness" was, at best, a distant third. Although holding such a viewpoint, today, seems to demand a willful credulity, to understand our long and winding path to best effort, we need to unpack the mindset in some depth.

'CEO thinking' was doggedly persistent, in part, because it was really a constellation of mutually reinforcing beliefs that all increasingly feel like relics today, including:

- ✗ Correctness is all-or-none; The Last Bug can be found.
- ✗ Efficiency is a pure virtue; the alternative is waste.
- ✗ Scale independence is possible; size doesn't matter.
- ✗ Faults, if any, will be rare and simple—i.i.d. random.

Indeed, such obsolete notions of software have an undeniable germ of truth—so long as *reliability is just a hardware problem*. And see how naturally they reinforced each other:

- ✗ Since small software for small problems can be *proven* correct, and size doesn't matter, provable correctness should be the gold standard for large software as well.
- ✗ Since memory is perfect and efficiency is always a virtue, any software that ever recomputes anything non-trivial is poorly written, wasting time and energy.
- ✗ Since faults are rare it's more efficient to do nothing until they arise, at which point—since they are i.i.d.—a little local modular redundancy will eliminate them.
- ✗ Since correctness is strict, yes or no, that means all *incorrect* outputs are *equally bad*, so comparisons are pointless and no useful distinctions can be made.

That last claim, in particular, was insidiously powerful because it implied there was a 'research desert' outside of strict correctness. Although one might imagine, for example, that *fault tolerance* work (e.g., [39]) would naturally deal with degrees of incorrectness, its actual mission is to improve the *probability* of reaching that unique strictly correct output.

3.1.1 *Embracing incorrectness*

But as we've seen, eventually hardware determinism *does* fail, taking any illusions of guaranteed strict correctness down with it. Now what? How well does some `Algorithm A` do, vs some `Algorithm B`? To put the question bluntly: Which algorithm will give the **better wrong answer**?

Starting in the 2000s the field of *approximate computing* (e.g., [35, 44, 36, 32, 46]) did begin to ask such questions, but even there, the allowable deviations from strict correctness were carefully controlled, and typically limited to data corruptions in numeric tasks like image processing. For example, one might allow random bits to flip, but only in floating point numbers, and even then, only in the mantissa.

A much more radical stance was taken in the 2013 essay [47] that coined the phrase 'CEO software'. Using pairwise comparisons to sort a standard deck of cards, the essay asked: *How incorrect* are typical sorting algorithms, if the 'pairwise comparator' can fail? Figure 2 shows the results.

The humble, much-maligned bubble sort utterly crushes the efficient algorithms. Its wrong outputs are so much better! And its glaring inefficiencies—it reconsiders card pairs it has already looked at, and only swaps cards one position at a time—are exactly why it tolerates failures so well.

Bubble sort shines here because—of course—one alternative to efficiency is not waste but *robustness*. And yes, this may all seem terribly obvious now. But let's listen to robust-first computing (RFC) and CEO talk a little bit longer:

CEO: Well so what if inefficient algorithms do better when hardware fails? In fact hardware rarely fails! And, bubble sort is *so* inefficient it's useless for large tasks!

RFC: Sure, but lots of problems are small. Bubble sort would work for my TODO list, my shopping list, my...

CEO: OK whatever. Well the fix is easy: Just oversample the comparisons! At 7x they're 99.99% correct, and 'MergeSort7x' is *still* faster than bubble at $n = 52$.

RFC: But you're assuming comparison failures are independent and identically distributed! What if they're not?

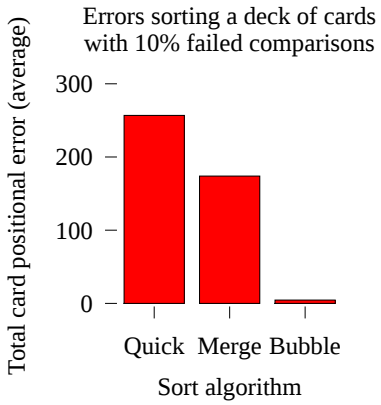

Figure 2: Output errors vs sorting algorithms when comparisons may fail. Lower is better. From [47]. See text.

CEO: Well then what *are* they? You have to specify the error model *first*! Or else nothing can be said!

And, scene. That exchange illustrates a crucial point: With the best effort computing contract, CEO's demand for an *a priori* error model must go unanswered. Hardware has the right to fail *arbitrarily*. If CEO therefore has nothing to say, we need to be talking to someone else.

3.2 The original best effort software

Section 2 showed how the core architectural principle of indefinite scalability blocked global hardware determinism; the replacement was best effort hardware. We just saw how best effort hardware undermines CEO software. But if software execution *isn't* a completely predictable problem-solving process, unfolding correctly and efficiently from beginning to end, step by tiny step, what could it be?

3.2.1 *Meaningful spaces*

The alternative software organizing principle was so obvious that it took us decades and disasters to see and accept: *Don't try to solve it; just make things better.*

Without global determinism, there's a vanishingly small chance that a correct and efficient sequence of steps will arrive at the desired output, as the sequence length grows. But if we replace that long chain of tiny steps with a set of broad goals, and empower myriads of tiny 'workers' or 'agents' to *make things better* according to those goals, given their immediate circumstances, then problems can be effectively addressed—if not strictly *solved*—in a robust fashion.

For indefinite scalability, each agent will have access only to limited processing power, limited internal memory, and limited communications with nearby agents. The best effort software programming task is to design ways for such an agent to decide, from a standing start, which of its actions is currently most likely to make things better. That determination need not be perfect, but does need to be *local*—and to achieve that, one basic software design strategy is to assign *broad meanings to large regions of space* within the machine—like house vs garage or kitchen vs bathroom. Then, appropriate processing depends on where you are, and large data movements depend not on finicky pointers in RAM but on broad directions in space.

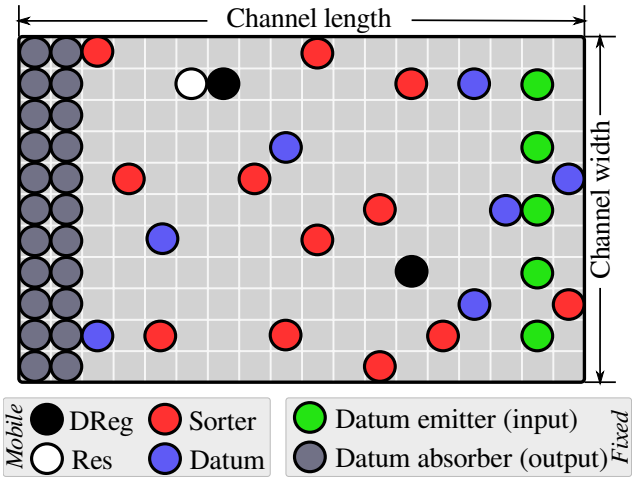

Figure 3: *Demon Horde Sort* schematic diagram. Best effort continuous flow sorting on a 2D grid. From [49]. See text.

3.2.2 *Demon Horde Sort*

Sorting, as an example, was one of the first applications demonstrated for the (at the time, simulated) indefinitely scalable tiles discussed in Section 2.3.2. The method presented, *Demon Horde Sort* (DHS), works on a spatial grid in which data "atoms" appearing on the right side of the grid must be transported to the left, while also moving vertically so that large items fall to the bottom and small items rise to the top (see Figure 3).

However, given that there are only a limited number of "output atoms" on the left, but many possible data values, even in the best case multiple data values will be aliased to each absorber. And worse than that, the data items to be sorted are not presented in neat batches. The "input atoms" (on the right in green) emit new "Datum" atoms (blue) into the grid asynchronously and opportunistically. So a Datum with a much bigger value *might* appear around an input, grid right, just as the previously biggest Datum was *about* to be absorbed at the bottom, grid left—and now what?

Demon Horde Sort is a *best effort method* for a *best effort task*: The method doesn't promise to get the correct answer, but then the task doesn't promise any correct answer exists. All DHS does promise is its best effort—to *be there* and to make things better. A 2013 video [41] showed DHS recovering from not just i.i.d. bit-flipping "X-rays" but also a "power outage" that blacked out fully two-thirds of the grid.

3.3 Modern software

Many properties of DHS apply to general robust computations on indefinitely scalable tiles. They:

- Are explicitly spatial processes, and do useful work with only local communications.
- Define custom *spatial semantics*—like left is output, down is big—so local actions can advance global goals.
- Run asynchronously in parallel, and tolerate variations in event delivery rates.
- Are self-stabilizing, self-repairing and/or self-rebuilding.

As originally presented, the 2013 DHS had one huge flaw: Its input and output grids were immobile, and cancerous. As can be seen in the above-cited video, the emitters and absorbers achieve robustness simply by growing vertically until they hit the "edge of the universe." Naturally, this approach will not play well with any northern or southern neighbors, and significant early work (e.g., [53]) focused on developing a variety of "membrane" mechanisms to help computations grow locally but remain bounded globally.

3.3.1 Movable robust software engineering

`mrDSM` [56]—a "movable robust distributed state machine"—was an influential demonstration of modularity and scalability in best effort software. It could grow from a 'seed' placed in any sufficiently large region of empty space, and could make position adjustments while maintaining and updating its state according to an arbitrary transition function specified in firmware. Its innovative but clean SPLAT code [54] made repurposing easy, and it spawned several generations of "mr" constructs, including the famous `mrEKF` [61] that arguably triggered the AER race, as discussed in Section 2.3.3.

More fancifully, it also attracted a coterie of retrocomputing fans dedicating to recreating famous vintage hardware and software. For a time at least, such nearly-pointless confections were produced to demonstrate the power of a new tile, beginning with the `mr4004` [58], a model Intel 4004 chip running on the DISCO tiles [66], and reaching an outrageous peak with `mrDOS` [75] running on prototype MIT1 tiles [74].

Today, to the properties displayed by the DHS, most software engineers would add that mobile, robust computations:

- Grow only into empty sites and manage their own size.
- Die cleanly if too damaged for safe self-repair.
- Participate in standard apoptosis signaling.
- Interact across layers only with your own type.

3.3.2 Computing in 2.1D

That last bullet point is concerned with multilayer tiles, which debuted with the T3 tile [60], long after the original DHS. Although the T3 tile—like the T2—was indefinitely scalable in only two directions, it was reconfigurable internally to support up eight "Z" layers; the X and Y extents shrunk to maintain the total site count. This "2.1D" or "thin film" style became standard on subsequent tiles, but starting with DISCO it was no longer configurable; instead, tiles were offered with varying layer counts and price points.

Thin-film computing is proving extremely popular and powerful, even though Z access is limited to the immediately adjacent layers, and then only within the Moore neighborhood. TILE1 tiles, currently the most widely used, have 3, 8, and 16 layer versions—but the 16 layer monsters are still hard to find at any price, with most production long since locked up by the big AI companies and Microcomputech.

3.3.3 Best effort today

The rise of a new computer architecture is an exceedingly rare and exciting event. Best effort software and systems are now rapidly gaining capabilities and design wins. Systems built on traditional hardware determinism are still widespread, but their functions, scale, and market are now shrinking in size and power.

Best effort is for systems with big responsibilities. Hardware determinism is for tools with small consequences. As it's said: "There is nothing wrong with the von Neumann machine that cannot be fixed by making it a small and individually insignificant piece of a larger robust system."

4. SECURITY AND SOCIETY

Throughout `CC0` there were fitful attempts to go beyond hardware determinism, including—and this is the last time, promise—von Neumann himself attacking it directly [1]:

> Thus the [future] logic of automata will differ from the present system of formal logic in two relevant respects:
>
> 1. The actual length of "chains of reasoning," that is, of the chains of operations, will have to be considered.
> 2. The operations of logic...will all have to be treated by procedures which allow exceptions (malfunctions) with low but non-zero probabilities.

But von Neumann's predictions of future computer architecture went nowhere, while his deterministic hardware went everywhere, for another eight decades.

The CEO paradigm was so dominant, so interlocked and so dug in, not even the Words of the Master could touch it. Ideas that increased robustness by increasing redundancy, like N-version programming (e.g., [9]), or undermined correctness for any reason, like the approximate computing discussed in Section 3.1.1, rarely gained much traction.

4.1 Security lost

Looking back, it almost seems the CEO paradigm could have controlled the narrative forever—except for its naked Achilles heel, its glowing kryptonite, its one very tragic flaw:

Hardware determinism makes computer security impossible.

For the security and securability of real-world systems—not just abstract correctness proofs—the already-discussed technical problems of hardware determinism are compounded by misaligned economic incentives and negative externalities for computer security—that are *also* created or exacerbated by hardware determinism. Here we review the history.

4.1.1 Physical space lost

In the earliest days, a computer was secured just like any other valuable—by locking it away from untrusted prying eyes and grubby hands. But that all changed with the rise of computer timesharing (Section 2.1.1), allowing people to sit at scattered terminals while cables teleported their eyes and hands straight into the guts of the machine.

With a timesharing architecture, unfettered access to the machine was prevented not by the computer room walls but, in the end, by a single bit. One modifiable bit value, in one single register, buried in a machine whose memory was now packed cheek to jowl with friends and strangers alike.

Security issues were soon found in timesharing systems—there were ample hints about the risks of this 'software-managed single-point security' idea (e.g., [8]). But after each hole was found, it was fixed, and any damage was minimal because the users were few and the stakes low.

'OK' computer security gave way to 'fair' computer security, but faith in the Last Bug said that would be temporary.

4.1.2 Human input lost

Timesharing was terrible for security—but at least, at one end of the cable there was usually a human, operating at human speeds, who had to sign in with a password before most interactions could occur. But that all changed with the rise of local area networking and 'personal computers' in the 1970s and 1980s: Now the human interacted with some local machine, which in turn interfaced with some remote host.

That change may seem minor, but in practice machine communications were often *trusted more* than human ones. With no human in the loop, machine-machine interactions both could be, and needed to be, much faster—and the simplest, fastest approach was to treat the remote machine like more local hardware. Short-range hardware protocols like PCMCIA and Firewire—at least in early versions—*deliberately* granted full memory access, with no authentication, to every remote device encountered. Add in hardware and software bugs, and longer-range machine protocols, like Ethernet and USB, had similar problems (e.g., [33, 42]).

'Remote exploits'—operating via networks—were found, and then fixed, as before. Now though, the value at risk was swelling as business processes and data poured onto the company network—but the potential efficiency gains looked dazzling, and anyway everybody was doing it.

And so 'fair' computer security gave way to 'poor' computer security. But hope for the Last Bug remained.

4.1.3 Shared fate lost

Local area networking was terrible for security—but at least, the machines were mostly controlled by the same organization that owned the local area network. A misbehaving computer could be unplugged and fixed or replaced; a misbehaving human could be summoned and punished or fired. But that all changed as the Internet went mainstream in the 1990s and 2000s, and local area networks began interconnecting everywhere, with IP addressing providing—for starters, stable and globally public—addresses for nearly every machine in the interconnected network.

Over the two decades starting in 1990, billions of Internet-capable personal computing devices were sold, Internet traffic approximately doubled every year, and computer security failures likewise exploded in frequency, scale, and cost. Physical security, and social or organizational enforcement, were all toothless when victim and attacker had different lands, languages, and laws.

Software-managed single-point security was left as the last line of defense, facing a net full of malware, and a churning ocean of software bugs—arising from new apps and new versions, from misconfigurations and unexpected interactions, from users unwilling to be unpaid administrators. And software-managed single-point security was like a picket fence in a tornado, like a stop sign in a stampede.

4.1.4 The end of computer security

Looking back today, for most purposes computer security ended with the rise of Internet-connected personal computing devices. Forget about the software users—even the software creators and vendors couldn't keep up with their own 'issues'. People tossed the Last Bug in with Santa Claus and the Easter Bunny, and adopted a security fatalism echoing Blanche DuBois, the famous Tennessee Williams character who 'always depended on the kindness of strangers.'

In the 2010s, bugs were so rampant and attacks so relentless the only safe assumption was that any Internet-connected device more than a year or so old was probably '0wned' by some miscreant somewhere. Successful attacks were sometimes overtly monetized via identify theft or 'ransomware' extortion—but sometimes the results were nearly imperceptible: A bit more video buffering for the owner, perhaps, and an imperceptible uptick in spam for everybody.

In one crazy sense, Blanche DuBois was right: Some attackers might take more dependable care than you did—acting, in effect, as a paid-by-crime system administrator for your device. And, for a while anyway, you survive as the oblivious patsy, a squatter on your own property.

By late in the decade, inside legions of Internet-connected devices—and many that officially weren't—small-time spammers and botnetters were being driven out or taken over by organized crime and hostile nation-states. The big players were assembling armies of hijacked devices, holding them at the ready, exploring their capabilities, quietly feuding and poaching among themselves, and biding their time.

4.2 The tipping point

This section draws heavily from Chapters 2 and 9 of [67].

We take as given the ERC consensus regarding the economic, climatological, social, and political factors in the run-up to the VSW. Here we briefly review the surrounding timeline, focusing on key events that emphasize pre- and postwar computational issues in both technology and society.

It's always about the people. *And* about the architecture.

4.2.1 Prewar instability

In the 2010s, concerns had been aired about humanity getting wiped out in the future by a "super genius AI" sprung from our information technology, but by 2025 it was seeming we'd be lucky to get that far. On February 2nd, we found out it took only level 2 autonomous cars, plus over-the-air software updates, plus ordinary human patience, ingenuity and malice, to amass a battalion of hacked empty vehicles that killed hundreds and maimed thousands as they exited the American football championship [59].

Some postwar reporting [65] has pointed to instabilities caused by an undisclosed eleven-digit theft from the SWIFT network, shortly after the EU fell apart in 2020—but, in the public mind at least, the Parking Lot Massacre was what put the Earth inexorably on the path to war. Over the following year, Internet QoS was punished as huge DDOS attacks targeted network and national infrastructures, and route instabilities and network partitions grew commonplace.

Sociopolitical polarizations hardened across the globe.

4.2.2 The VSW

On Friday morning, April 17, 2026, at 04:36:32 Eastern time, signal quality degraded on the Apollo2 transatlantic cable between La Rochelle and Toms River, though landing conditions were nominal and no warnings had been raised. Partial load shedding to the FA-1 cable began automatically, but for nearly ninety seconds a significant fraction of westbound packet payloads were corrupted, due in part to a misconfiguration in La Rochelle, and in part to a previously-unseen race condition in an upgraded router in Toms River.

By 04:41:00, in a likely related development—though no details are known, at least by the public—a 'go code' had been received by long-dormant malware in a residential electric meter in West Windsor Township, New Jersey, USA.

The activated malware began disconnecting and reconnecting the customer load—mostly a heat pump at the time—four times a minute, with an irregular duty cycle that proved to repeat each hour. This created a weak but distinctive pattern of pulses in the line voltage, which was detected by other hacked meters on the secondary circuit, and they activated as well. Soon, their synchronized switching caused voltage pulses to appear on the distribution transfomer primary.

By 04:50:00, complaints had begun coming into the Public Services Jersey reporting systems. At 04:56:07, PSJ performed a microgrid reset for all of Mercer County, to no avail. The power triacs had failed in some meters, but thousands were flipping in sync. At 05:14:00, the alarmed PSJ operators—reluctantly admitting that the affected electrical meters were ignoring WAN commands—disconnected their systems from the regional grid and blacked out the state.

But that was too late. The huge load swings had already been pulsing the HV transmission lines, and the rogue sync was starting to build on Long Island and in eastern Pennsylvania. Over the next two hours, regions from the northeast through the midwest went erratic and then dark.

By 07:30, it appeared the sync growth had been stopped by preemptive firebreak black-outs at the borders of the Eastern Interconnection, and at that point the western states still had power. Actually eradicating the malware would ultimately require a truck roll to each infected meter, but on that Friday morning, over a hundred million people in the USA and Canada had no power, and at 09:00 the USA President declared war on the Actors or Nations Responsible.

During 2025, cooler heads globally had become rare and unheeded, and by 12:30, GPS and most satcom had been taken out by orbiting lasers and kinetic interceptors, plus high altitude EMPs which also blacked-out most of the world.

By 15:00, four unlaunched missiles had exploded in their silos in North Dakota and Wyoming, and similar events had occurred in other countries. Still, by 22:30, six of the Earth's great metropolitan regions were burning, at least two by accident, along with many less densely populated areas.

4.2.3 *Shared fate*

Everyone knows how this ended.

At 00:13 Eastern time, Saturday, April 18, 2026, a group of fifty-one submarine commanders from nations around the world—organized by the "Six Heroes" deep in the North Atlantic—issued a joint statement that moved as flash traffic on all reachable military channels. Each commander also made a video and uploaded it, by undersea cable injection in some cases, and by every data carriage still accessible.

In their mother tongue, each commander announced their immediate and irrevocable defection from their respective country, and swore a nuclear-backed oath of loyalty "to the Earth and all its life." Each then made the same suggestions: "Stay calm and help others stay calm. Move if you must, stay put if you can and take in those who had to move. Help however you can, and respect those who are helping."

And each video ended the same way, with the same statement: "The war is over. Earth is rising."

Over the next days and months, people repeated it. The war is over. Earth is rising. In tent hospitals and on soup lines, out apartment windows and over back fences, in meme and ink and spray paint, people joined the sync, and the signal grew and grew.

Earth is rising. Earth is rising. Earth is rising.

4.2.4 *Postwar reconstruction*

Forty million people died on Friday, April 17, 2026. Ninety million more by New Years' Day. Vast regions are poisoned. Global disease and deformity will be elevated for centuries.

Now thirteen years on, though the dying has slowed, the survivors still grieve and ghosts are everywhere. But most of humanity lived, and many felt their heart turn to the Earth. Not everybody, never that, but enough, then, and multitudes more now. Much infrastructure survived. Food grows. There is rebuilding. There are children.

The EU reconstituted itself as the REU, an open regional cooperative that looks to the Earth; other interlinked sibling regionals soon followed. The Weavers design and deploy interaction protocols, including successes like our beloved ERC itself [63], our emerging public credit unit '𝒈' [68], and their famously-quirky 'Long Bond Awards' [71].

There is still competition, still advantage and disadvantage. There are still angels and jerks, good and bad ideas and luck, winners and losers. Innovation matters so we try to avoid undue damping, but opportunity also matters so we try to avoid undue accumulation. Connectivity matters, so we try to stay in touch and not lose anybody.

And our machines matter, so we try to compute better.

5. BEST EFFORT COMPUTING

To this day there is no public certainty whether the VSW was finally an accident, but with a systems eye it hardly matters. Unmanaged positive feedbacks had generated staggering potential differences all across our economic, social, and technological circuits; at such voltages, focusing on which capacitor exploded first is tiresome and disingenuous.

It was found that since only untriggered meters accepted WAN commands, broadcasting a thirty-seconds-wrong clock update would have stopped rogue sync growth. But so what.

Yes, and the malware was traced to a 2024 meter update, which had been altered, before signing, by the software contractor's compiler, hacked after a normal zero-day exploit.

But so what, and so what. Postwar, there was little appetite for such near-sighted cleverness and peephole blame.

The question was how to do it *all* much better.

5.1 Deleveraging information systems

The principle of leverage goes back to Archimedes: Given time and a lever long enough, even the tiniest force can move the biggest mountain. But the bigger the mountain to move, the more likely the lever or the fulcrum will break instead.

With guaranteed deterministic execution, hardware was treated as *literally unbreakable*. So computer scientists and software engineers just naturally kept ratcheting up the leverage. They relentlessly eliminated redundancy so the tiniest difference could have the hugest impact. They championed 'mechanism not policy' which amounted to making everything as programmable as possible. And they built programs using ever more tiny steps, containing ever more complex contraptions of compound levers and switches, to move ever more massive, more valuable, and more dangerous loads.

Eventually, the complexity of those software contraptions themselves became the major *source* of vulnerabilities, but it was the extraordinary leverage built into the machines that made those vulnerabilities so *damaging*.

And the device vendors pumped out a billion copies.

For our own safety, such relentless leveraging had to end.

5.1.1 Short-term mitigations

Some steps were easy. Software updating without an end-user unique physical action (UPA) was banned in all new devices. Autonomous cars, severely curtailed since 2025, acquired mechanical hit-stop lockouts with a UPA reset.

A '`#PasswordsR4People`' PSA was a hit—with its catchy tune and people wagging their finger as whiny animated TVs and fridges begged for WiFi—and the 'Internet of Things' cloud-backed-device model withered in earnest.

5.1.2 Software liability

Some steps were harder. Software liability finally took hold, vastly restricting the ability of device vendors to disclaim liability for damages caused by faulty software in the device. This impacted upstream software vendors, but those risk-sharing mechanisms were contractual rather than legislative. Open source developers were unaffected, though their code got ample scrutiny before commercial adoption.

Device purposes narrowed, prices rose, and after-market software updating became rarer. Fixed-function device manufacturers rolled out a "Pure X" certification and marketing campaign. Pure TV. Pure Fitness. Pure Firewall.

General-purpose programmable computers were sold as power tools rather than appliances. Purchasers and users accepted more risk and liability, beginning with *I/O leverage*: Who put it on the net? Who connected it to the chainsaw?

Such changes were not about making security *guarantees*, but for improved *computing hygiene* for the public health. Manfacturers squawked but there was broad consensus on `earth.gov` that internalizing security costs was the only obvious way for such industries to remain in private hands.

5.1.3 The great respatialization

Software liability hit phones the hardest. Dodgy chaotic 'app stores' gave way to a tightly-curated set of core applications—centering on text and voice, camera and gallery, and an enlightened HTTP/1.9-noscript browser.

Though long-distance travel decreased greatly after the war, navigation was still the most-missed phone app. But map data and satellite imagery was obsolete in many regions, and there was no immediate consensus on when or even whether to relaunch GPS or its ilk.

Undeterred, a group of students in Coimbra built a phone app that used 'Simultaneous Location And Mapping' [19, 28] (SLAM) to build local 3D maps based on camera input, while also inferring the phone's pose within the constructed model. Their '`MyHometown`' [62] app combined SLAM localization with IMU dead reckoning and speech-to-text, so the students could build and label open-source maps of their own neighborhoods just by walking and pointing and talking.

Before the students were done, `MyHometown` allowed peer-to-peer map fusion and label sharing via phone mesh networking, and offered a slick fish-eye view interface [15] for landmark navigation. Even though at first it had to be sideloaded into phones, it caught fire as the students and their friends, and then seemingly all of Coimbra, ran around fleshing out their city models and its surrounds. Label spam was minimized via consensus models and civic pride, plus use- and edit-wear visual cues [20] to hint at local model quality.

In 2030, the students formed an REU-affiliated startup, which was promptly acquired [64], and within two years most of Earth's populated lands were part of `MyHometown`. The students' localizer had become the Globalizer.

5.2 Living computation

As discussed in Section 2.3.3, there was a concerted post-war effort to develop a new best-effort architecture and an associated computing ecosystem. It is perhaps ironic, but after years of research and relatively small-niche milspec and ruggedized applications, the breakthrough best-effort product was a kid's toy. At this point it's hard *not* to know lots about "Baby Dooper," the star of the 2032 holiday season; here we focus on its technical innovations and their impact.

5.2.1 Hardware

Developed by Entire Gizmos Studio, with global marketing by Bandbro, Baby Dooper is an oblong plastic device fit for a child's hand. It has a plastic necklace at one end, a charging port at the other, a narrow reflective color touchscreen running up the front, and sensors on the back.

Inside is a sizable custom battery, more sensors, two piezos for vibration and a speaker—and two flip-mounted bare T3 chips, newly affordable since the Tusken tile announcement. The T3s provide a total of 130K live sites, configured in four layers, running at about 15 AER to save power.

5.2.2 Bringing up baby

Baby Dooper evokes the 'digital pets' that debuted in the '90s, but its gameplay is open-ended exploration, teaching and interaction. Aside from "instincts" like getting cranky on low battery, a new Baby Dooper just shows slow swirly colors, with a cute soft boop or daroop sound sometimes.

Baby Dooper's signature feature is its ability to learn from the player, Baby Dooper's 'parent'. It vibrates happily when rocked around its waist, and squawks displeasure if shaken along its length. The vigor of the motions provide degrees of parental feedback about whatever Baby Dooper is up to. Approval encourages more and similar; disapproval, change.

There's no inherent limit to the patterns and sounds and sequences you can teach your Baby Dooper, though part of the fun is you never know exactly *what* it will learn from your feedback. With some player investment, Baby Dooper can learn to imitate patterns drawn on the touchscreen, and sketch them more or less accurately or freely later.

5.2.3 Software

Inside the T3s, Baby Dooper runs a best-effort implementation of 'Simultaneous Recognition And Interaction' (SRAI) [57], a variant on the SLAM algorithms mentioned in Section 5.1.3. SRAI is built around a high-dimensional 'interactions map', which in Baby Dooper's case involves the screen state plus all its physiological and environmental sensor data. SRAI uses the SLAM math to estimate that map, while simultaneously estimating its current "pose" within it, which means deciding what is the current interaction, how far along is it, and what Baby Dooper should tell its crews of pixel painting and buzz generating atoms to do now.

Baby Dooper's 'Garden of Eden' state is loaded from a ROM, which also holds the state transition code. I/O elements, for handling sensors, piezos, and screen, go on the three open edges of each T3; custom SRAI `mrEKFs` get the outer layers, and a best-effort sparse distributed memory [18, 69] gets the inner layers for the interactions map. The teaching gestures inject 'change agent' atoms into memory, which diffuse in search of a marginal link to reinforce or flip.

5.2.4 Getting to know you

Aside from its preprogrammed instincts, Baby Dooper's 'interactions' emerge entirely from its unique shared history with, and feedback from, its parent. And the single biggest reason Baby Dooper was, and is still, so popular—and why it changed the shape of computing—is its uncanny ability to recognize the hand of its parent, and respond to no other.

The box instructions specify that only the child—Baby Dooper's new 'parent'—should handle it for the first few days. After a week or two, if anybody other than the child—perhaps a covetous sibling or nosy parent—picks up the child's Baby Dooper, within a couple minutes, almost invariably, Baby Dooper will start making angry blinking patterns. If it's not set down or given to its parent, it will start to cry, then really wail, then go dark. It will then take significant coaxing from Baby Dooper's parent to wake it up.

Even if the child *tries* to explain to someone how to handle their Baby Dooper properly, the child isn't really sure what's critical and what isn't. They just know that what they do works. And their interactions have usually become so subtle and fast nobody else can make them work anyway.

5.2.5 Birth and death

Baby Dooper uses a small ASIC to handle power, clock, watchdog, and sensorimotor conditioning to and from the T3s. But no CPU. No flash storage. And no power switch.

A brand new Baby Dooper has a one-time pull-string to engage the battery. A full charge supports a couple days of heavy activity, or a few weeks on standby—but the site SRAM draws power even at 0 AER, and if voltage falls too low, bit reliability decays and Baby Dooper begins to lose its mind. If caught in time, the Baby Dooper may survive, though likely with some aphasia, but if it's too damaged when power returns, the watchdog reset fires, and all is lost.

5.2.6 Impact

In recent news, it seems the world's oldest living Baby Dooper turned eight this year. Raised by one Janosch Cortez, a former Entire Gizmos Studio engineer, today the prelaunch prototype mostly snoozes in its solar charging cradle, and its touchscreen is iffy, but Baby and parent have still got it:

> As he tilts and jiggles it, a tiny cigar-shaped 'rocket' rises up the screen and explodes with a pop, then colorful sparkles rain down and spell out 'ECS'. Cortez laughs, "It's supposed to be EGS,"—his old employer's initials—"but Baby really doesn't like to cross the 'G'." [77]

Informal surveys suggest hundreds of seven-year-old Baby Doopers may still live, but even reaching three is uncommon. The usual cause of death is neglect, though sometimes the battery is drained deliberately, to start over or give it away.

Certainly, an unexpected Baby Dooper death can be traumatic, and some parents argued for adding flash memory or some other backup mechanism. But Entire Gizmos was adamant. Let someone else make that toy, they said. Life needs life. A little daily charge is not too much to ask.

5.3 Security in best-effort computing

It didn't help grieving owners, but researchers soon showed how to clone a live Baby Dooper fairly easily, since the T3 JTAG interfaces were pinned out [70]. It did take some care to get into the case without distressing the Baby Dooper, and thus rousing the ire of burgeoning machine rights groups.

A complete site SRAM snapshot contains a tremendous amount of information, and capturing one is absolutely a serious compromise. But a well-used Baby Dooper is full of partially redundant interactions with slightly different actions, and its model is always changing at least slightly. Without also knowing the parent's moves it's generally only possible to make approximate guesses about what will happen in actual use. The gameplay and appeal is all about Baby Dooper and its parent learning to 'dance' together, using poses and steps that only they know.

Researchers saw that the 'generalized synchronization' displayed by Baby Dooper and parent had potential applications in security, and 'sync-for-security' or "syncurity" was born. In particular, syncurity might provide a high-usability solution to the long-vexing problem of passwords.

5.3.1 Bad passwords

The *presentation of a shared secret* is a time-honored way to link a current interaction to some pre-existing relationship, allowing it to persist even if its participants interact only rarely. Users have 'logged in' with secret passwords since timesharing (Sections 2.1.1 and 4.1.1), and the weaknesses of passwords as proof of relationship are well-known.

5.3.2 Multifactor authentication

During the 2010s and '20s, rising damages due to poorly-chosen and stolen passwords spurred the spread of *multifactor* authentication. Unfortunately, each added 'factor' hurt usability, and for the exact same reason it helped security:

- *Something you* ***know***, like a good password, is a pain to remember because the whole point is it's supposed to be arbitrary.
- *Something you* ***have***, like a time-based token generator, is yet another thing to have to lug around, because the whole point is it's a physical object.
- *Something you* ***are***, a 'biometric' marker like a fingerprint, is hard to keep secret because the whole point is it's an actual part of the person.

Users naturally resisted such aggravations, especially since authentication is not per-user but per-relationship, so their necessary factors tended to multiply absurdly.

5.3.3 Password managers

Over the years a variety of hardware-, software-, and cloud-based 'password managers' were developed in attempts to tame that proliferation. The user needed only remember the password to the password manager itself, was the pitch, and then it would do the rest. Though it usually did not involve token generation or biometrics, the argument was really good passwords were helpful all by themselves.

But concentrating many secrets under one just increases the leverage on the one remaining secret. Without fundamental improvement in how that master secret is managed, the expected loss is unchanged but the variance grows: The system becomes more brittle, and the failures grow larger.

Perhaps, the thinking went, something like Baby Dooper could be that fundamental improvement.

5.3.4 Early attempts

The 2033 'PasswordFriend', claiming to be 'Baby Dooper inspired', was a regular Bluetooth (BLE) hardware password manager with an accelerometer added so a "pass gesture" could be used to open it. This did avoid telltale wear marks on the chiclet keyboard, as advertised, but since the pass gesture was defined during setup and not tuned interactively, its moves had to be big and slow for reliable recognition. The user traded a subtle long-term information leak for an extremely public password entry mechanism.

PasswordFriend had other problems. Out of the box it demanded a network pairing for software updates (with a pinhole button UPA). And it backed up your encrypted passwords to its servers by default. And so on. PasswordFriend ultimately became a case study of—and nerd slang for—*not* getting the lessons of Baby Dooper. It was an epic flop.

Similar password management products launched over the next two years. While none of them were PasswordFriend-bad, and some of them were profitable, they weren't particularly successful or compelling either. They were just *devices*.

5.3.5 Skini

During those same years, however, a startup named 'Microcomputech', spun out of CSIRO with heavy financing and networking out of Sydney and Shanghai, was quietly refining prototypes that *did* get the lessons of Baby Dooper. Microcomputech finally launched 'Skini' in July 2035, with physical design by Kinki Himego Chimu, UX flow by White/Space, and manufacturing and fulfillment by KKST Chennai.

Skini is "Baby Dooper all grown up." Still fits in the hand, still a necklace, but it's a keystone-shaped gold pendant with a black glass display inset on the front, sensors on the back, and hidden chip LIDAR on both sides. It charges wirelessly and has only one tiny hole on the back, plus two 'Skini catches'—custom 1.35mm twist-lock connectors—at the top.

The necklace is made of 'Skini cable', a 1.2mm gold wire braid covering two power lines and a plastic BIDI fiber, with a mating pair of Skini catches. Four 'Skini stones' also launched—an extra battery, extra sensors, WBAN interface, and conventional encrypted storage—each housed in a semi-precious stone fitting. Skini stones twist-lock onto each other or onto cable segments to form the necklace itself.

5.3.6 Beyond Baby Dooper

After unboxing, the owner removes the one-time pull-wire, then wears Skini around and plays with it, using the Baby Dooper teaching gestures to make patterns and sounds, as Skini builds up its interaction map. Going beyond Baby Dooper, Skini also uses 'BodySLAM' processing to build a crude map of the owner's body, and localize itself with respect to that. Skini interactions can depend, for example, on whether the pendant is facing the owner. With Skini bracelets and anklets, launched in 2036, extremely subtle whole-body gestures became possible—although of course they are yet more devices to keep charged, and they require a WBAN stone for communications [37].

Creating shared sequences is just the beginning with Skini; there are six 'topics' or modes, each associated with an 'achievement' for the owner-Skini team to unlock together. For example, like PasswordFriend, the pendant contains BLE, but with Skini the "Touch the World" achievement is required before that interface will even power up.

Interacting about a topic adds new inputs and actions that can become waypoints or endpoints in the team's shared sequences. As a result, there could never be a 'user manual' explaining how to access BLE, for example, because each team develops their own unique way to bring up that topic.

5.3.7 The security frontier

It usually takes at least a week of 'bonding' before the owner can even get the option of teaching Skini a password, but that's all part of team building—and again, storing passwords is just the beginning. Once BLE comes up, you can pair Skini with your phone. As achievements accumulate, different Bluetooth profiles become available, and Skini can filter and prioritize texts and other modalities, vibrating for urgent ones then displaying them when the owner looks at the pendant, and so forth.

The Skini pendant is built around two 16 layer TILE1 chips (Section 3.3.2), providing 2M sites clocked for about 10 to 100 AER depending on the layer. It also has a stock BLE controller, and a custom ASIC for tile control and interfacing to BLE and the Skini necklace.

One of the top questions on the Microcomputech FAQ [73] is: "How come Skini isn't a phone too?"—and if their answer is too simple, it does highlight Skini's core value proposition and security stance: "Skini's job is to interact with *you*, not with the world. Skini can operate phones on your behalf, but it will never *trust* one, let alone *be* one."

5.3.8 Attacking Skini

The Skini ecosystem has been growing for four years now, and several specialized and 'economy models' have followed the original celebrity launch. Millions are in use, and though social engineering never ends, nothing like a remote exploit against Skini itself has been reported.

Closer in, it's easy to snoop on the Skini cable, and not that hard to spoof a stone. On the other hand, the cable protocol is weak, and the packets are encrypted when it matters. Also, pendant communications are initiated by the best-effort hardware, against which timing and differential power attacks have, to date, proved virtually useless (see Section 5.3.9 below). Similarly, the BLE controller is minimally trusted and Bluetooth profile processing is sequenced by `mrDSM`s on two tile layers.

Although Skini is significantly hardened compared to Baby Dooper, researchers have no doubt that some physical cloning attack against the pendant itself would succeed—perhaps drilling the battery wire, while freezing the pendant to increase SRAM remanence, then going in for JTAG or decapping. Although no one knows what happens in the dark, no such attempts have been reported in the open literature.

Baby Dooper and Skini have driven a growing sense in society that *some* machines really deserve to be called 'alive'.

5.3.9 Security lessons from best-effort computing

Against `CC0`'s grim computer security history, two lessons stand out, here early in the era of best-effort computing.

First, whole families of attacks against traditional computing depend heavily or critically on hardware determinism. This includes not just overtly hardware attacks like the classic 'Rowhammer' [48], but also many side-channel [30] and weird machine [52] attacks.

Side-channel and differential attacks benefit from a 'quiet' execution environment, to ease detection of the attack's effects. Attempted mitigations often work by adding i.i.d.

noise, but if a probe can be repeated arbitrarily, arbitrary amounts of noise can be washed out. By contrast, as best-effort 'mr'-style components (Section 3.3.1) adjust and move, they create multiscale spatiotemporal non-determinism with respect to the fixed hardware, not just low-order noise. Add in 'Heisenberg effects'—where the attacks themselves also change the system state in uncertain ways—and successful side-channel attacks are inherently hard to mount.

Similarly, with deterministic execution, it is typically difficult to *avoid* creating 'weird machines', which assemble many unrelated pieces of code and data to form a remotely-programmable virtual machine. Although traditional code is typically brittle due to CEO design, weird machine code is *vastly* more brittle, because it is a high-order pastiche of nearly arbitrary co-occurrences in memory.

That leads to the second security lesson: Although indefinitely scalable architecture and best-effort execution *do* make things harder for the software developer 'defender', they are *asymmetrically much worse* for the attacker.

Best-effort software builds and maintains whatever object relationships it needs using exclusively short-range and low-order 'neighborhood' dependencies, as illustrated in Section 3.2.2. Even though that is usually less efficient then a tangle of pointers crisscrossing user RAM, it means that only short-range memory access needs to be provided, which vastly reduces the raw material available to the attacker to 'get weird with' during the most delicate stages of an exploit.

Also, as discussed above, the redundancy and mobility of best-effort software components largely decouples program semantics from specific hardware locations, except near the 'edges of the universe', where the uniform grid of sites gives way to device-specific I/Os. Of course there is no magic, and best-effort developers can squander their home court advantage with deliberately insecure elements. For example, a T2 tile programmer could use native code to create a state transition that routes site data directly to *bash*.

The inherent advantage for incumbents is that *they are there first*. For eighty years, by insisting on flat RAM and hardware determinism, the 'good guys' threw away their edge, in exchange for short-term programming convenience.

For shame.

6. ONWARD TO CC1

Here at long last, we have reached the present. Best effort design wins are still the exception rather than the rule, but research and development is white hot, and deployment is accelerating. CEO is fading, and robust-first is rising, and attitudes are shifting in computation and in society at large.

We have a just a few concluding thoughts.

6.1 The microcomputome

In the human body, the "microbiome" [38, 50] is the vast constellation of microbes—comparable in number to our own human cells—that live within and on our bodies. In exchange for a reasonably cozy place to live, many of these microbes perform helpful or essential services for us—such as crowding out other types of microbes that would cause us illness or death. By and large, it is advantageous for them to defend us because of *shared fate*: If we die, they die too.

Our microbiome is a buffer between us and the great unwashed microbial world outside, prepared to meet microscopic would-be invaders on their own terms. The "microcomputome" performs the same function for the digital world: It interacts with external computational entities on our behalf, and depends on us for its survival. At least intuitively and esthetically, the designers at Entire Gizmos understood that. The Skini designers viewed themselves as building the microcomputome explicitly—so much so that they built it into the 'Microcomputech' company name.

A microcomputome is physically close to us and continually observing and interacting with us. The only reason we let that happen is that we *know* it, know its capabilities and its limitations, as intimately as we know anything. And we know it is making its best effort to be *loyal* to us, and not to some device vendor or software developer or cloud services provider—let alone some tinpot dictator or would-be nation-state. Such a rent-free product has a higher price and a lower margin, and so justifying it to investors can be a heavy lift. We salute Microcomputech, and especially Entire Gizmos Studio, for finding ways to thread that needle.

6.2 Distributed systems vs tools

Although `CC0` was awash in CEO, especially pedagogically, it is important to note there were competing voices as well—often, in particular, from authors experienced in systems, distributed computing, or networking. Unlike the hospital-corners crispness of a good algorithms paper, systems papers were usually less abstract and declarative, and more specific and narrative, often organized around war stories or lists of principles (e.g., [7, 12, 23]).

A *system* is a complete unit embedded in the world, with a place and a size and, somehow, inevitably, with bills to pay. It is judged by its ability to generate positive net value—for its owner at least, and hopefully for society at large.

An algorithm, by contrast, is a *tool*, in itself incomplete and unconnected, and meant to be used by systems and other tools. Given it's correct, it is judged by its ease of use, its range of applicability, and the resources it requires.

Increased efficiency increases a tool's appeal but also its fragility, destroying its result quality when determinism fails, whether that's due to hardware aging, environmental stresses, bugs, or malice. But distributed systems thinking was too rare, and `CC0` was blind to the systemic costs of its tools. CEO birthed fragile architectures running fragile operating systems under towering card-house skyscapers of fragile libraries and applications—controlling an aggregate value-at-risk that only stopped growing due to global war.

Today, we say "How could anybody stand to compute like that?" If only they had known there was an alternative.

6.3 Space

Our first computing century began by obliterating space using random access memory, so there is perhaps some satisfaction, or at least narrative closure, as it ends with the reintroduction of spatial constraints and concerns across the computational stack, from neighbor grid sites, to intertile cache lines, to the microcomputome, and on up the scales.

It has been a very, *very* long path to best effort. But it is a good beginning for our second computing century.

Acknowledgments

Author use handle due to thir age. This report began as a "Computing in 2014" curation project. Thanks to `Vintabulary239` for the 'weirdo' 201X-era text restyling—*'L33T'*! Plus special thanks to Mr. Bonen for period idioms, and to Stet and NoSrslyStet for support. Keep circulating the bits!

7. REFERENCES

[1] John von Neumann. The general and logical theory of automata. In *Cerebral Mechanisms in Behavior*, pages 1–41. John Wiley and Sons, New York, NY, USA; London, UK; Sydney, Australia, 1941.

[2] John von Neumann. First draft of a report on the EDVAC. Technical report, University of Pennsylvania, June 1945. Report prepared for U.S. Army Ordinance Department under Contract W-670-ORD-4926. Reprinted in [10, pp. 177–246], [11, pp. 399–413], [22], and [25].

[3] John von Neumann and Arthur W. Burks, editors. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, IL, USA, 1966.

[4] John G. Kemeny and Thomas E. Kurtz. Dartmouth time-sharing. *Science*, 162(3850):223–228, October 1968.

[5] The PDP-10 Software Writing Group. PDP-10 timesharing handbook. Technical report, Digital Equipment Corporation, 1970. Retrieved April 8, 2039 from https: //archive.org/details/pdp10-timesharing-handbook.

[6] Daniel G. Bobrow, Jerry D. Burchfiel, Daniel L. Murphy, and Raymond S. Tomlinson. Tenex, a paged time sharing system for the PDP-10. *Commun. ACM*, 15(3):135–143, March 1972.

[7] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems, 1975.

[8] R. P. Abbott, J. S. Chin, J. E. Donnelley, W. L. Konigsford, S. Tokubo, and D. A. Webb. Security analysis and enhancements of computer operating systems. Technical Report NBSIR 76-1041, Institute for Computer Sciences and Technology, National Bureau of Standards, April 1976.

[9] Liming Chen and Algirdas Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. In *Proc. 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8)*, pages 3–9, 1978.

[10] Nancy B. Stern. *From ENIAC to UNIVAC: An appraisal of the Eckert–Mauchly computers*. Digital Press History of Computing Series. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, 1981.

[11] Brian Randell, editor. *The Origins of Digital Computers: Selected Papers*. Texts and monographs in computer science. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., third edition, 1982.

[12] Butler W. Lampson. Hints for computer system design. *SIGOPS Oper. Syst. Rev.*, 17(5):33–48, October 1983.

[13] Stephen Brain. Applying the transputer. *Electronic Product Design*, 5(1):43–48, January 1984.

[14] M.V. Wilkes. *Memoirs of a Computer Pioneer*. History of Computing Series. MIT Press, 1985.

[15] George W. Furnas. Generalized fisheye views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '86, pages 16–23, New York, NY, USA, 1986. ACM.

[16] Fred Cohen. Computer viruses. *Comput. Secur.*, 6(1):22–35, February 1987.

[17] Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines: A New Environment for Modeling (Scientific Computation)*. The MIT Press, 1987.

[18] Pentti Kanerva. *Sparse Distributed Memory*. MIT Press, Cambridge, MA, USA, 1988.

[19] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In Ingemar J. Cox and Gordon T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag, Berlin, Heidelberg, 1990.

[20] William C. Hill, James D. Hollan, Dave Wroblewski, and Tim McCandless. Edit wear and read wear. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 3–9, New York, NY, USA, 1992. ACM.

[21] B. J. Overeinder, P. M. A. Sloot, and L. O. Hertzberger. Time warp on a transputer platform: Pilot study with asynchronous cellular automata. In M. Valero, E. Onate, M. Jane, J. L. Larriba, and B. Suarez, editors, *Parallel Computing and Transputer Applications*, pages 1303–1312, Amsterdam, 1992. IOS Press.

[22] John von Neumann. First draft of a report on the EDVAC. *IEEE Annals of the History of Computing*, 15(4):28–75, October/December 1993. Edited and corrected by Michael D. Godfrey.

[23] Peter Deutch and James Gosling. The eight fallacies of distributed computing, 1994.

[24] Norman Margolus. CAM-8: A computer architecture based on cellular automata. 1995.

[25] Phillip Laplante, editor. *Great papers in computer science*. IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1996.

[26] William Aspray. The Intel 4004 microprocessor: What constituted invention? *IEEE Ann. Hist. Comput.*, 19(3):4–15, July 1997.

[27] Marc Snir and Steve Otto. *MPI-The Complete Reference: The MPI Core*. MIT Press, Cambridge, MA, USA, 1998.

[28] J. E. Guivant and E. M. Nebot. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *IEEE Transactions on Robotics and Automation*, 17(3):242–257, June 2001.

[29] Thomas M Chen and Jean-Marc Robert. The evolution of viruses and worms, December 2004.

[30] François Koeune and François-Xavier Standaert. A tutorial on physical security and side-channel attacks. In Alessandro Aldini, Roberto Gorrieri, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design III*, pages 78–108. Springer-Verlag, Berlin, Heidelberg, 2005.

[31] Priya Ganapati. Hardware hackers create a modular motherboard. Wired online, August 2009. Retrieved Apr 10, 2039 from https://web.archive.org/web/*/ www.wired.com/2009/08/modular-motherboard/.

[32] Krishna V Palem, Lakshmi NB Chakrapani, Zvi M Kedem, Avinash Lingamneni, and Kirthi Krishna Muntimadugu. Sustaining Moore's law in embedded computing through probabilistic and approximate design: Retrospects and prospects. In *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 1–10. ACM, 2009.

[33] Loïc Duflot, Yves-Alexis Perez, and Benjamin Morin. What if you can't trust your network card? In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, RAID'11, pages 378–397, Berlin, Heidelberg, 2011. Springer-Verlag.

[34] Kurt Ferreira, Jon Stearley, James H Laros III, Ron Oldfield, Kevin Pedretti, Ron Brightwell, Rolf Riesen, Patrick G Bridges, and Dorian Arnold. Evaluating the viability of process replication reliability for exascale systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 44. ACM, 2011.

[35] Debabrata Mohapatra, Vinay K Chippa, Anand Raghunathan, and Kaushik Roy. Design of voltage-scalable meta-functions for approximate computing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[36] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. 46(6):164–174, 2011.

[37] IEEE Standard for local and metropolitan area networks - Part 15.6: Wireless Body Area Networks. *IEEE Std 802.15.6-2012*, pages 1–271, February 2012.

[38] Curtis Huttenhower et al. Structure, function and diversity of the healthy human microbiome. *Nature*, 486(7402):207–214, June 14 2012.

[39] Robert S. Swarz, Philip Koopman, and Michel Cukier, editors. *IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2012, Boston, MA, USA, June 25-28, 2012*. IEEE Computer Society, 2012.

[40] David Fiala, Frank Mueller, Christian Engelmann, Kurt Ferreira, Ron Brightwell, and Rolf Riesen. Detection and correction of silent data corruption for large-scale high-performance computing. In *Proceedings of the* 25^{th} *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2012*, pages 78:1–78:12, Salt Lake City, UT, USA, November 10-16, 2012. ACM Press, New York, NY, USA.

[41] David H. Ackley. Robust-first computing: Demon Horde Sort (full version). Online video. http://www.youtube.com/watch?v=helScS3coAE, January 2013.

[42] Patrick Stewin and Iurii Bystrov. Understanding DMA malware. In *Proceedings of the 9th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'12, pages 21–41, Berlin, Heidelberg, 2013. Springer-Verlag.

[43] David H. Ackley, Daniel C. Cannon, and Lance R. Williams. A movable architecture for robust spatial computing. *The Computer Journal*, 56(12):1450–1468, 2013. http://dx.doi.org/10.1093/comjnl/bxs129.

[44] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *18th IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, May 2013.

[45] David H. Ackley. Bespoke physics for living technology. *Artificial Life*, 19(3_4):347–364, 2013.

[46] Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, page 113. ACM, June 2013.

[47] David H. Ackley. Beyond efficiency. *Commun. ACM*, 56(10):38–40, October 2013. Author preprint: http://nm8.us/1.

[48] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372, June 2014.

[49] David H. Ackley. Indefinite scalability for living computation. In *Proc. of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 4142–4146, 2016.

[50] Ron Sender, Shai Fuchs, and Ron Milo. Revised estimates for the number of human and bacteria cells in the body. In *PLoS biology*, August 19 2016.

[51] Mary Bellis. The history of the ENIAC computer. ThoughtCo, July 2017. Retrieved Apr 8, 2039 from https://web.archive.org/web/*/www.thoughtco.com/history-of-the-eniac-computer-1991601.

[52] Thomas F. Dullien. Weird machines, exploitability, and provable unexploitability. *IEEE Transactions on Emerging Topics in Computing*, December 2017.

[53] David H. Ackley. Digital protocells with dynamic size, position, and topology. In *Proceedings of the 2018 Conference on Artificial Life*, Tokyo, Japan, July 2018.

[54] David H. Ackley and Elena S. Ackley. The SPLAT spatial programming language. https://github.com/DaveAckley/SPLAT, 2018.

[55] David H. Ackley and Elena S. Ackley. The T2 tile achieves 6.5 AER indefinitely scalable on DReg/Res physics. In *Proceedings of the ULAM Programming and Systems Workshop (UPSW-2020)*, pages 43–51, January 2020.

[56] Arthur J. Dorn and Zoren G. Leonard. A movable, robust distributed state machine. In *Proceedings of The 2020 Conference on Artificial Life (ALIFE 2020)*, pages 506–513. MIT Press, 2020.

[57] Y. M. Egorieva. Incremental private basis compression of unstructured interaction spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(4):611–623, April 2022.

[58] Paula R. Magna and Fernando A. Malesuada. A movable, robust microcontroller. In *Unconventional Computation and Natural Computation 2024*, pages 67–77, 2024.

[59] Manda V. Mills. AI illusions and the Parking Lot Massacre. *Polarity Quoted*, 3(4):61–67, Summer 2025.

[60] Zoren G. Leonard and Arthur J. Dorn. The T3 tile achieves 30 AER_s with low intertile distortion. In *Proceedings of the ULAM Programming and Systems Workshop (UPSW-2026)*, pages 7–19, January 2026.

[61] Jhanvi Vedhika Puri, Qiang Li Feng, and Elizabeth Etiam. A movable, robust Kalman filter. In *Asilomar*

Conference on Signals, Systems and Computers (ACSSC 2027), pages 87–99, 2027.

[62] ZManic Software. My hometown localizer. https://github.com/zmanic/myhometown, 2028.

[63] The Weavers Collective. ERC: The 'Earth Rising Commons' children's curation protocol. RFC 15003, March 2028.

[64] Alvaro L. Velho. 'My Hometown' Startup Acquired by You-Know-Who. *The Register*, March 2030. https://www.theregister.co.uk/2030/mar/12/myhometown-startup-acquired-by-you-know-who.

[65] William T. Gurns, Jeanne M. Xi, and James West Wiggins. Was the VSW triggered by a massive 2022 cybertheft? *The New York Times*, CLXXIX(62,315), April 14 2030.

[66] Xiuying J. Bai, Lorem I. Dolor, Emil A.S. Hendrerit, and Frank V. Justo. The Discoideum tile achieves 260 AER via mobile caching. In *Proceedings of the International Symposium on Computer Architecture ISCA-2031*, pages 29–37, 2031.

[67] Earth Rising Commons. *ERC Revised report on the causes of the VSW*. Number 254-13 in ERC Consensus Documentation Series. ERC, Earth, 2031.

[68] The Weavers Collective. COIN: The 'Cooperative Opt-In Network' protocol. RFC 15947, August 2032.

[69] Entire Gizmos Studio. mrSDM: A movable robust sparse distributed memory. https://entiregizmos.com/whitepapers/mrSDM, October 2032.

[70] Solar Refiner. Super Dooper duper. *Rephracktory*, 3, April 2033.

[71] The Weavers Collective. Diametrically Supposed: Graph improvement via incentive-positive matched random bonding. RFC 16191, August 2033.

[72] Xiuying Zhong, Na Tian, Jie Hou, and Xiulan Duan. TILE1 achieves 517 AER. In *IEEE International Symposium on Circuits and Systems (ISCAS-2034)*, pages 127–131, 2034.

[73] Microcomputech. Frequently Asked Questions About Skini. https://microcomputech.com/skini/FAQ, 2035.

[74] Richard M. Phasellus and Danielle H. Sapien. Ultralow energy configurable nonlinearity by stressed stochastic resonance. *Science*, 440(7368), 2036.

[75] Dhruv Kabir Pandey, Henry F. Morbi, and Hinata Nakajima. mrDOS: A movable robust DOS 3.1. https://github.com/mrdos/mrDOS, 2036.

[76] Xiuying Zhong, Jaric Andersen, Guiying Ping He, and Xiulan Duan. TILE7 achieves 15.3 LPPE at 400 AER. In *IEEE International Symposium on Circuits and Systems (ISCAS-2037)*, pages 83–89, 2037.

[77] L. J. S. Winthorpe. World's oldest Baby Dooper has birthday. *Earth Today*, September 3 2039.