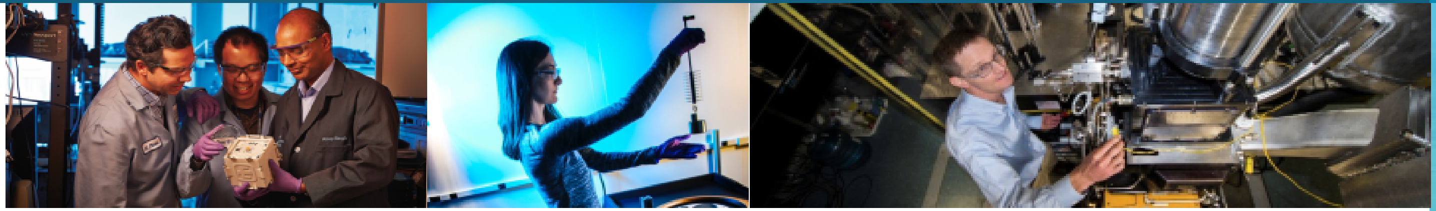
This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2022-13934C

Oses for High Consequence Applications; Might SeL4 benefit Nuclear Weapons?



PRESENTED BY

Noah Evans, 8741

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Our project: Deep Specifications of Sandia Systems of interest

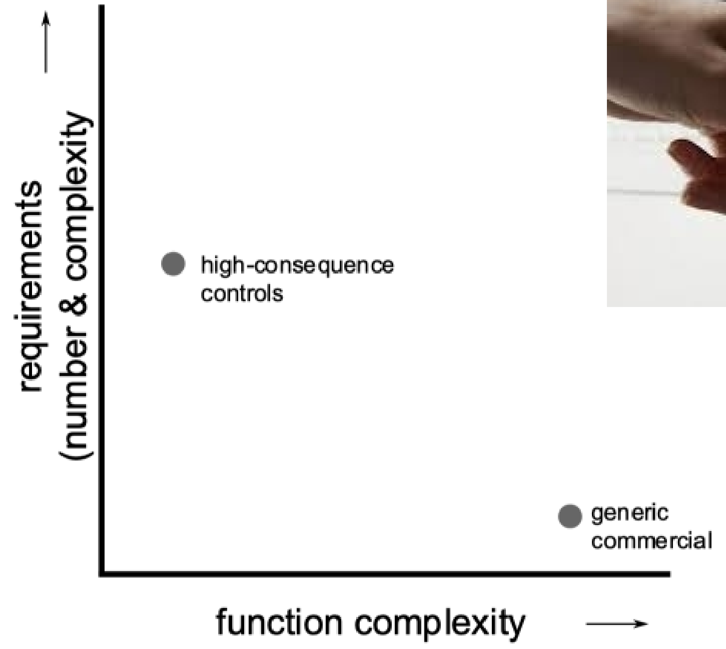
- Essential idea: write specifications and hardware together in formal language, proving:
  A. That the specification is complete (i.e. not underspecified)
  B. That the implementation obeys the specification for all possible executions of the hardware artifact.
- Can think of this approach as 100% unit testing.
- This approach was previously impractical (10-20 person years for an OS/Hardware system) for everyone without billions of dollars to spend (i.e. Intel).
- New advances in "Proof Engineering" (Software engineering but for mathematics) makes it possible to modularly and reusably write proofs for large systems.
- Most people at the conference are likely familiar with this kind of work.
- We're looking at applying these techniques to Sandia Systems of Interest.

Requirements for Sandia Systems of Interest

Our control systems are mostly low complexity, relatively easy to analyze, like a dishwasher.

But, they often have a large number of complex, high-consequence safety, security, and reliability requirements.

Low complexity + high consequence + complex requirements = ideal for a formal approach to design and/or verification.



Heavily resource constrained: Back to the 80s future

This system is simple, dumb, and resource constrained

We build from scratch

Our own fab, our own processor, our own peripherals

Processor:

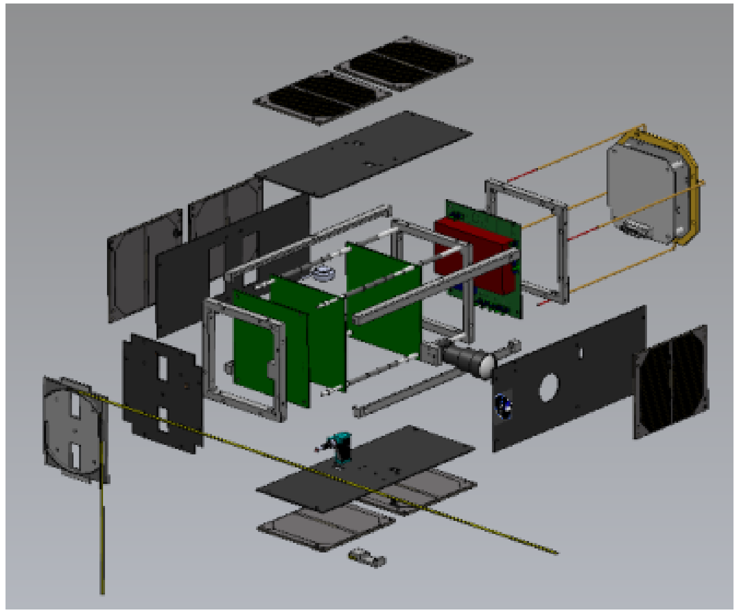
- 5-10 Mhz (can go 10-50 Mhz, for higher requirements, or Khz for low power)
- X Mbytes of Ram
- 100k total storage for bootimages
- No MMU

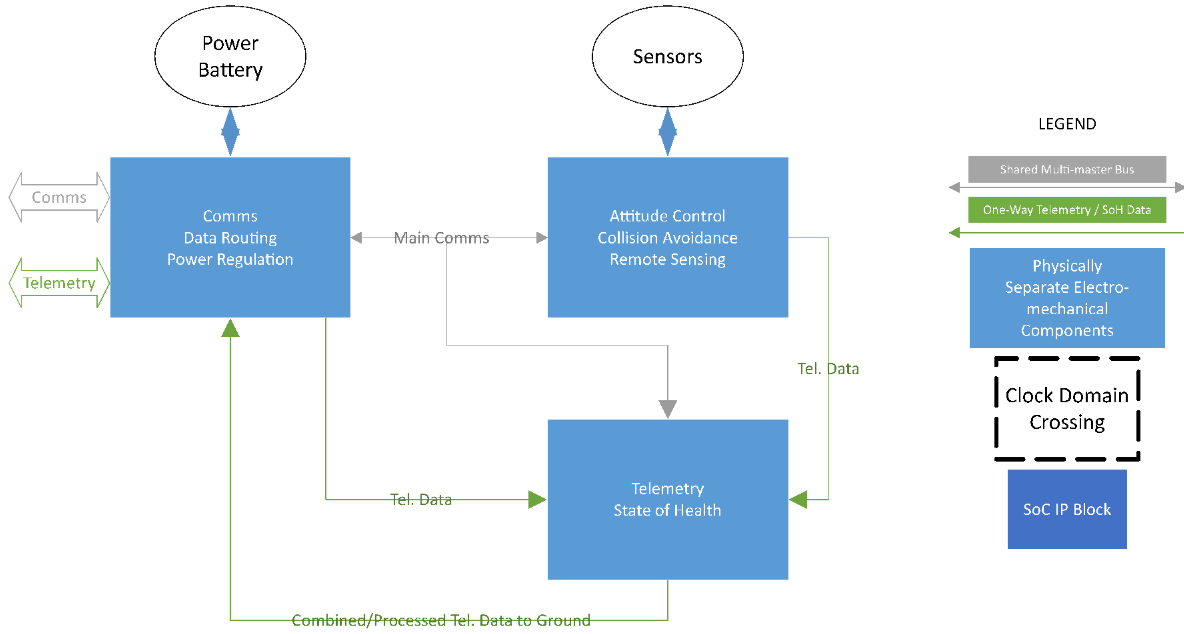
We write custom firmware to drive this currently.

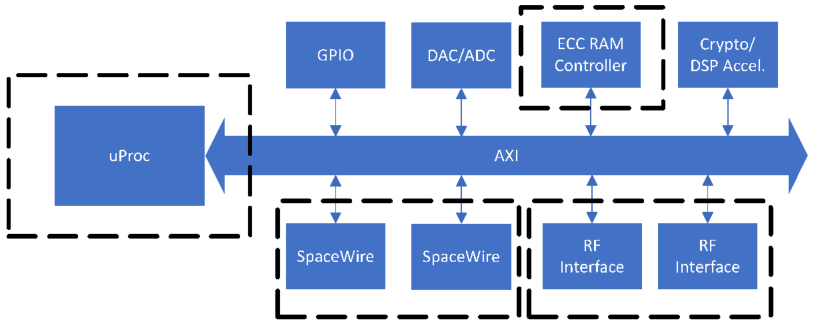
Proxy Architecture: a CubeSat analogous to Sandia Systems

Observation: Cubesats provide an unclassified analog to Sandia's typical sensitive systems of interest.

- Low complexity: 3 microcontroller class systems communicating over a bus
- High consequence: Any deviations from the specification mean you burn up in the atmosphere.
- Complex requirements: RF Comms, Collision Avoidance, Telemetry







Proxy firmware is hard real time and hard to verify

The hypothetical firmware for this device is a simple event loop which counts cycles and makes sure that certain events fire at particular multiples of the clock frequency to meet real time deadlines.

Very close to an old Nintendo Entertainment system where games were implemented by using an event loop and cycle counting was used to blank the screen and communicate.

This leads to code which is classic ”spaghetti code”. No clear separation of concerns makes it very to modify let alone verify.

=

=

An operating system would make real time and verification easier

Productivity and security argument

We want to be productive.

Adding functionality is combinatorial

Every new capability must be shown to not interfere with other capabilities meeting their deadlines.

If we could write processes as independent executables and let the OS handle the real time constraints it would make it *much* easier to meet production deadlines.

Security:

Would allow us to compose proofs. Prove individual processes meet our needs, rather than having to prove ad hoc over the entire firmware.

SeL4 meets a lot of our needs, verification and agility

seL4 has…

- Proofs of correctness
- Full process model
- Capabilities and message passing make it much easier to reason about and prove things about security.

Would make system development at Sandia easier to formalize and more “agile”

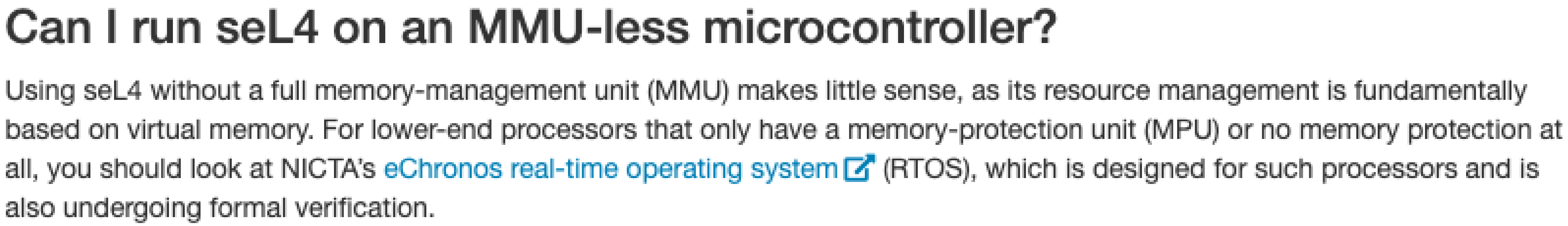
- Don’t have to implement real-time by hand.
- Much easier to reason about formally
- Implement firmware as dynamic communicating processes
  - Separate dynamic threads for communication channels using capabilities
  - Threads for system housekeeping, telemetry data.

But that still leaves us one big problem…

Roadblock: The missing MMU

Not enough area on our ASIC.
Also doesn’t give us reliable real time guarantees
So, until we get more die and determinism, MMU’s are a no go

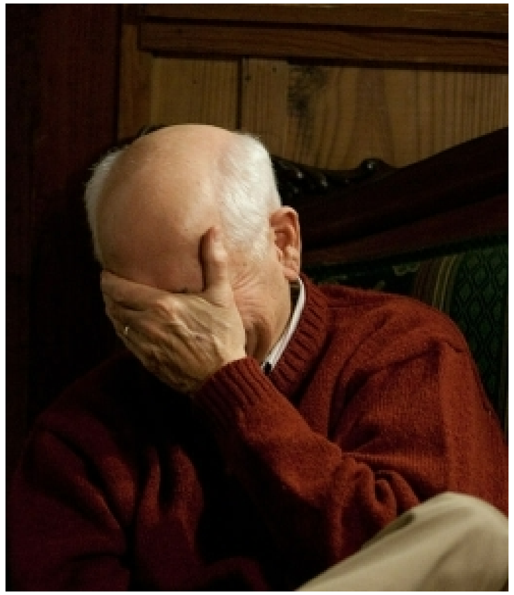
If only someone had thought of MMU-less before



Can I run seL4 on an MMU-less microcontroller?

Using seL4 without a full memory-management unit (MMU) makes little sense, as its resource management is fundamentally based on virtual memory. For lower-end processors that only have a memory-protection unit (MPU) or no memory protection at all, you should look at NICTA's eChronos real-time operating system (RTOS), which is designed for such processors and is also undergoing formal verification.

Thought experiment, are there alternatives?



Why not eChronos?

Microkernel idea vs RTOS

- RTOSes are very much a roll your own package, why not standardize as much much as possible?
- Roll your own communication, we'd love a standardized message passing
- Static process DAG, we'd love to experiment with dynamic process spawning
- RTOSes have no isolation guarantees, more proof obligations at the process level.

Why not use a pre-existing message passing implementation (seL4)?

Would really like a happy medium, "full OS" flavor of an OS like seL4 but predictability and lightweight requirements of eChronos

Alternative, Single Address Space OS

MMU contention causes “QoS crosstalk”

- Makes it very hard to meet fine grained deadlines
- You never know how long it will take to get a page (am I going to fault, is it in cache?)

Why not stuff everything in the same address space?

- Process switching no longer requires switching between memory contexts.

Much easier to give real time guarantees and prevent QoS cross-talk (e.g. Nemesis back in the 90s)

Implementable on our existing hardware.

However, big potential security problems.

- Requires proof of process isolation per process (no process touches any other processes memory without permission)
- Still fails in response to physical insult (e.g. particle strikes)

Formal methods make a single address space possible

Can prove isolation of individual processes

If processes obey isolation guarantees and OS does not break those guarantees (e.g. through system call information leakage), then the system is closed under isolation (i.e. you don't need an MMU assuming you are using position independent code)

Keep track of memory bounds (base and offset) checking instead of virtual page tables and indirection.

Future work: side channel attacks, potential use case for CHERI archicture/PUMP-Dover coprocessors

DRAPER DOVER MICROSYSTEMS

Not a full solution. Proofs assume nominal behavior

What happens to the OS when you processor is damaged?

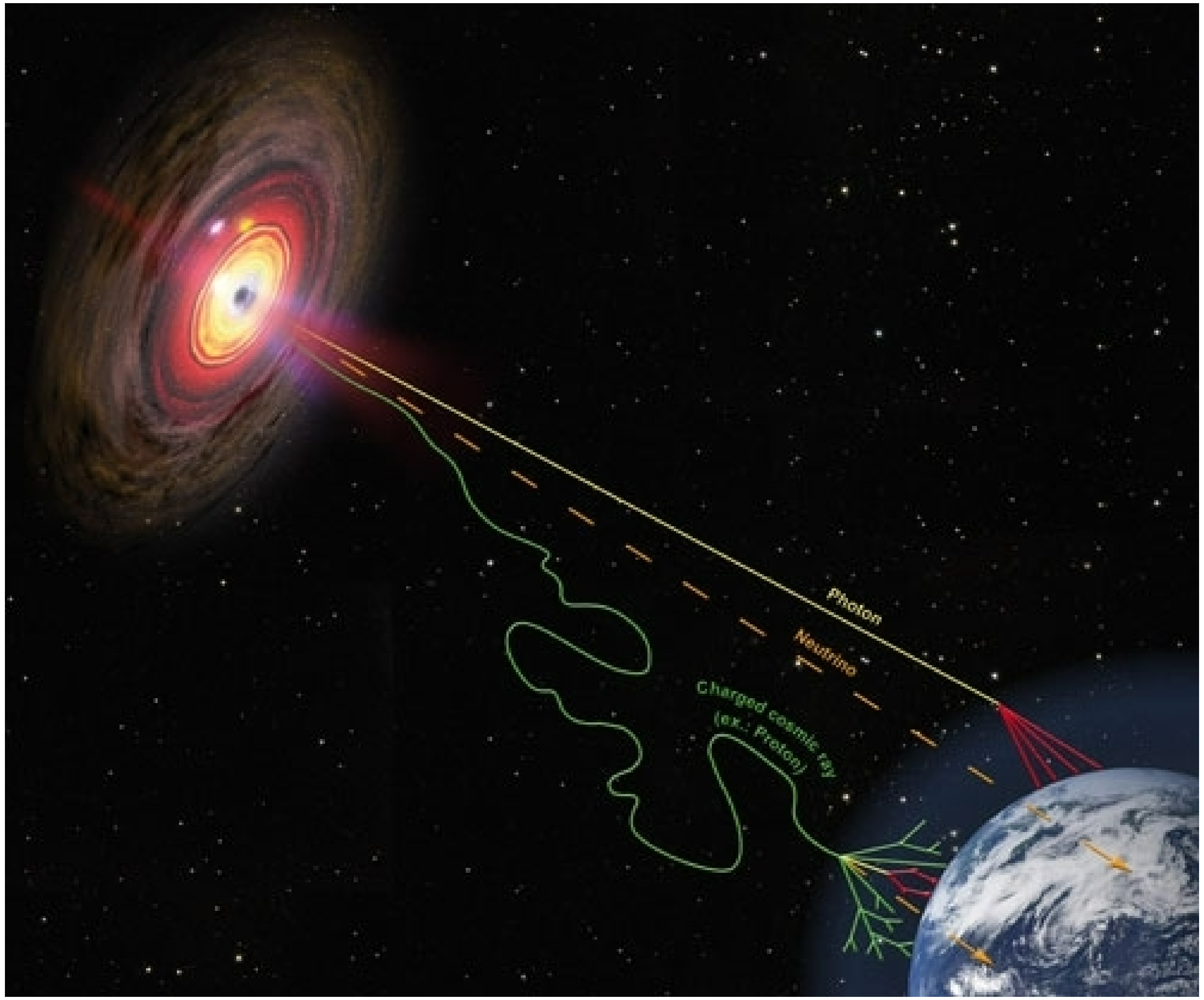
- Malicious memory strobing
- Particle strikes to memory address lines

Proofs of nominal behavior don't help you here.

- All of the behavioral assumptions are out the window.

Potentially another argument for the PUMP-Dover approach

But it's also possible to do in the processor…



Back to the future 2: Segmentation in a single address space?

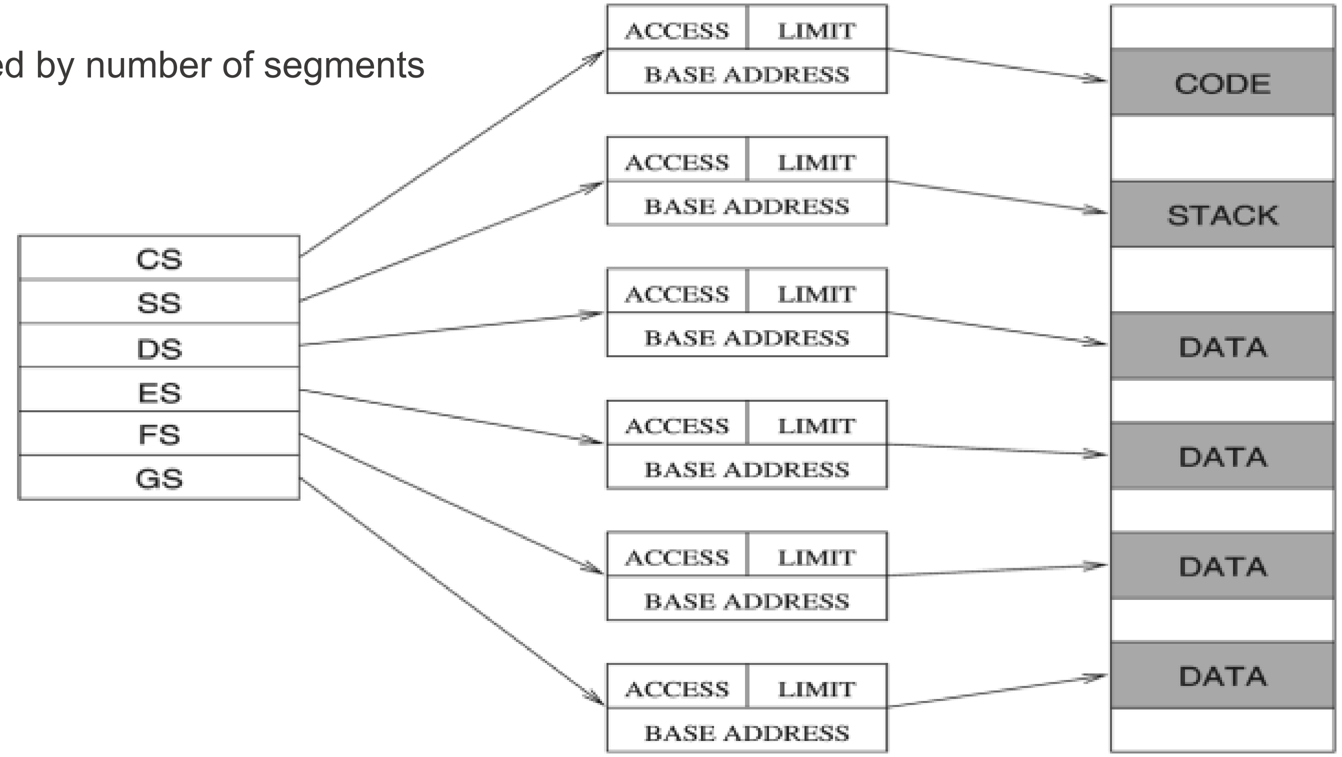
If our systems are the best the 80s have to offer, what about the best in 80s memory protection?

Makes verifying a single address space OS much easier.

- Hardware isolation
- Proofs of process correctness much easier in the presence of physical insult to the process memory

Still begs certain questions

- How many segments to support? Processes limited by number of segments
- How to partition processes among segments?
  - One segment per process?
  - Separate data segments?



Segmentation Image Credit: http://www.c-jump.com/CIS77/ASM/Memory/lecture.html

Conclusions

For high assurance low complexity systems a fully protected microkernel is too much for our current hardware

However, a single address space system with rigorous proofs of isolation between processes assisted by segmentation might work.

- This allows you to minimize context switching (context switches become setjmps/longjmps)
- Deterministic behavior makes real time easier
- Traditional security issues are (potentially) obviated by formal methods and hardware support

Idea is not unique to me (Arun and Silviu at Draper and Dover microsystems have compelling visions here)

It’s worth exploring a middle ground between verified systems like fully protected microkernels like seL4 and RTOSes like eChronos.

Would potentially make systems development easier to verify and “agile” for high consequence systems.

Acknowledgements

Ray Richards (DARPA): Invitation and Support

Lok Yan (AFRL): seL4 background and System Tradeoffs

Abe Clements (Sandia): Segmentation background

Ratish Punoose (Sandia): History of RTOSes at Sandia

Jon Aytac (Sandia): Proxy Firmware Details

Arun Thomas and Silviu Chiricescu (Draper): Ideas about single address space systems for mission systems

UPSAT project (University of Patras and Libre Space Foundation): Open Source Cubesat Implementation