Microsoft

Introduction to CHERIoT

RISC-V Summit 2023

Kunyan Liu (Microsoft)

11/8/2023

What is CHERIoT

CHERI for IoT

- Proposed RV32 ISA extension providing memory safety and software compartmentalization
- Optimized for IoT/embedded applications
- Minimal changes to application software source code
- Hardware, software & toolchain co-development
- Fully open-source (contributions welcome!)

The CHERI concept

- Capability Hardware Enhanced RISC Instructions
- Originated at Univ. Cambridge in 2010
- Memory safety enforced through ISA extension
- Previous work: MIPS, Arm, RV-64

CHERIoT Collaboration

- Microsoft
- SCI Semiconductor (UK)
- Univ. Cambridge
- Univ. of Oxford
- Google
- lowRISC

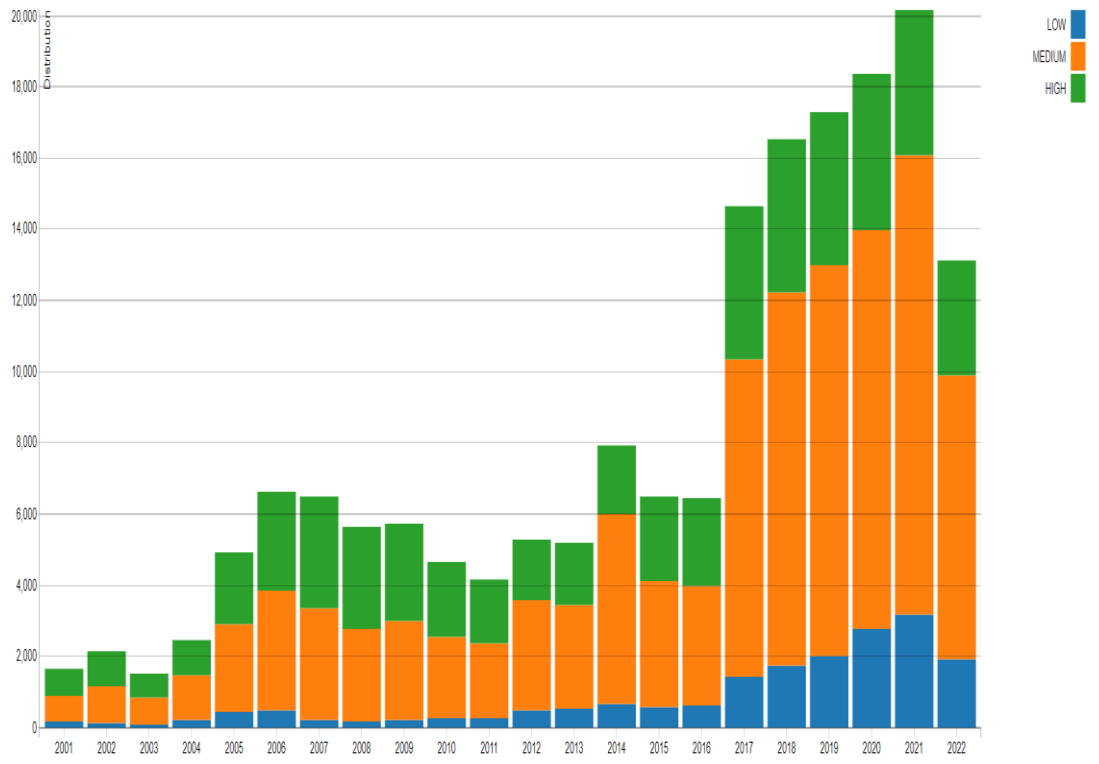
Our Deliverables

- ISA specification & SAIL model (CHERIoT-Sail)
- MCU hardware design (CHERIOT-Ibex)
- CHERIoT LLVM C/C++ compiler/linker
- Reference RTOS design (CHERIoT-RTOS)
- Low-cost FPGA platform (CHERIoT-SAFE)

CVEs and High Severity Bugs from (Lack of) Memory Safety

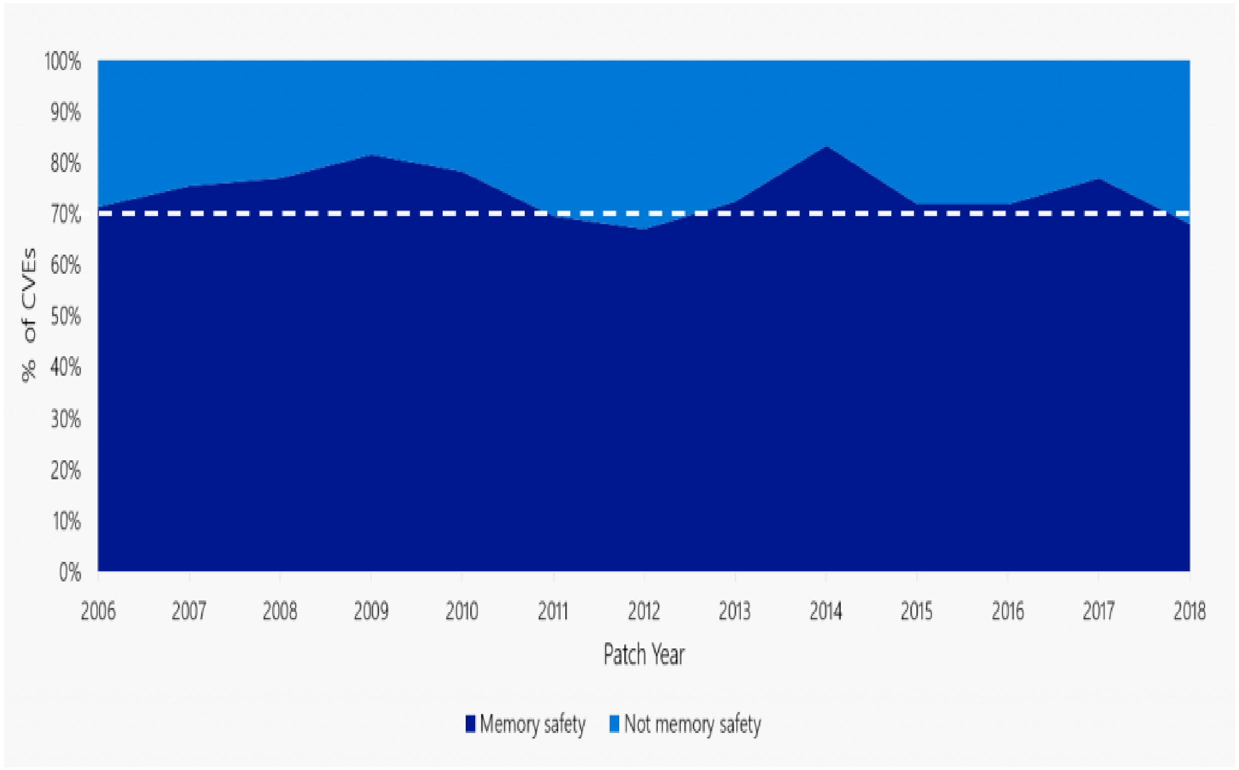
# of connected IoT devices worldwide in 2023: > 15, 000, 000, 000

CVSS Severity Count Over Time (as of 22 Jul 2022)



[NVD - CVSS Severity Distribution Over Time (nist.gov)]

Percentage of CVE's due to memory safety issues



Matt Miller. *Trends, challenge, and shifts in software vulnerability mitigation*. (BlueHatIL 2019)

CHERIoT Core Security Features

Spatial memory safety

- Data/code is only used as intended
- Bounds/permission checking on all memory accesses

Software compartmentalization

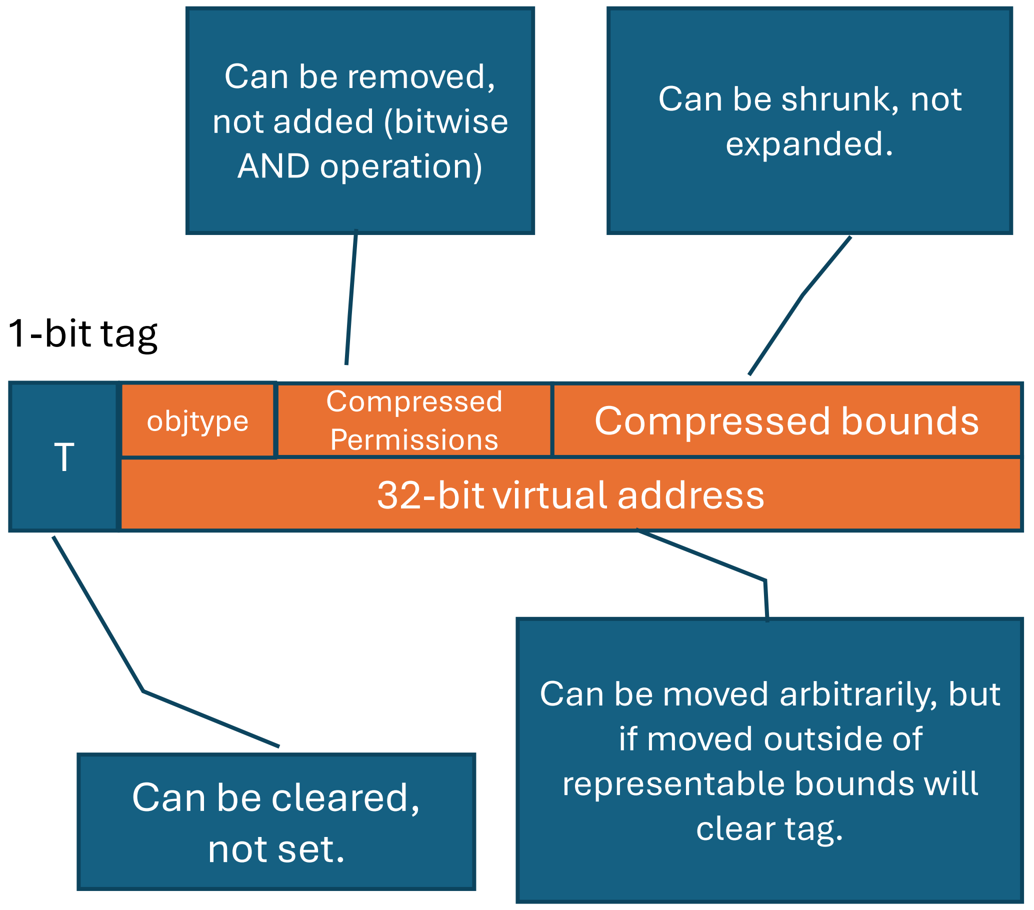
- Fine-granularity memory isolation between objects
- Sharing only as explicitly specified
- Light-weight switching
- Automated flow

Temporal memory safety

- Address use-after-free problem in heap allocation
- Hardware-assisted revocation

Overview of Capabilities

- Pointers in C/C++ are represented by capabilities
    - Hardware-supported data structures with properties like address bounds and permissions
        - 64+1 bit encoding to minimize overhead in a 32bit ISA
    - All memory accesses governed by corresponding capabilities
    - Unforgeable
        - Capabilities can only be derived, never created
        - Monotonic non-increase
    - Permissions
        - How a capability can be used (r/w/ex, etc.) and stored
- Where do capabilities come from?
    - System starts with a set of "root" capabilities
    - Trusted loader/switcher program derives & assigns initial capabilities for loaded software compartments
    - Software compartments derive further capabilities from the assignments.
    - Capability handling instructions generated by CHERI-aware complier/linker

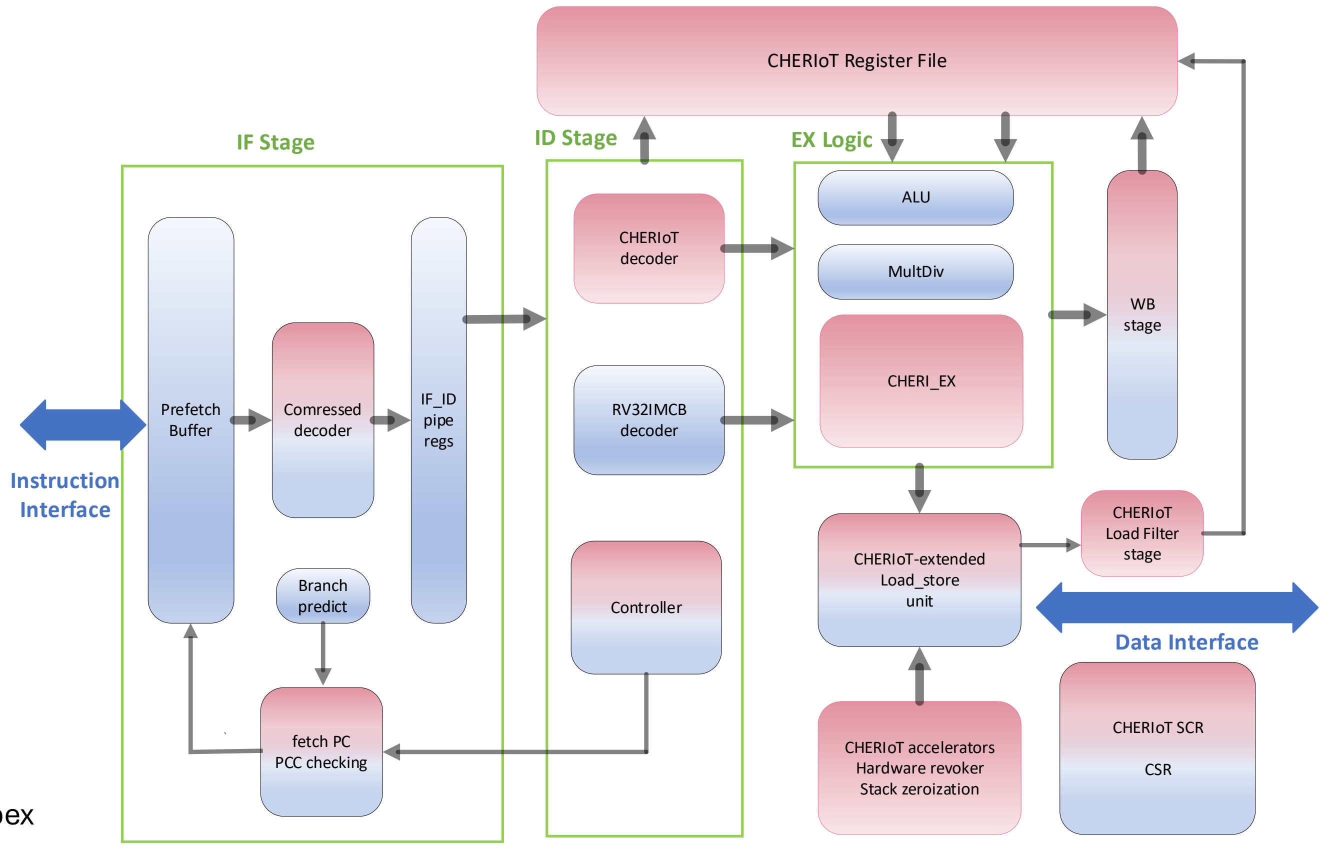

Can be removed, not added (bitwise AND operation)
Can be shrunk, not expanded.
1-bit tag
T
objtype
Compressed Permissions
Compressed bounds
32-bit virtual address
Can be cleared, not set.
Can be moved arbitrarily, but if moved outside of representable bounds will clear tag.

Proposed RISC-V ISA Extension

- CHERIoT ISA
  - Specified in CHERIoT technical report
  - Cheriot-sail as formal description
- ~30 new instructions
  - RV32-compatible format
  - Query & test capabilities
    - extract fields (bounds, permissions, etc)
  - Deriving/copying capabilities
    - The monotonic non-increase rule always followed
    - set/increase address, set bounds, seal/unseal, auipcc, etc.
  - Load/store capabilities from memory
    - Atomic load/store with compression/decompression
    - Tagged memory required
  - Control program flow
    - CJAL/CJALR
- New SCRs (special capability registers)
  - Analogous to CSR
  - Store root capabilities
  - Track execution (PCC)
  - MEPC, MTVEC extended for capability-aware exception handling
- Memory access checking
  - Data load/store accesses
  - Instruction fetches (PCC-based)
- Error handling
  - Exceptions (load/store/jump/fetch)
  - Tag clearing (derivation)

The CHERIOT-Ibex design

- Ibex core as the starting point
  - Small 32-bit, 3-stage pipeline RV32IMCB core
  - Open-source maintained by lowRISC
- CHERIOT-Ibex work at Microsoft
  - Full CHERIoT ISA support
    - All instructions 1-cycle except for CLC/CSC
  - Capability-extended register file
  - Extended 33-bit data load/store interface
    - MSB bit carries capability tag
  - Bounds/permission checking and fault handling
    - for all data accesses and instruction fetches
  - SCR (special capability registers)
  - HW assistance for RTOS
    - Capability load filtering and background revocation
    - Stack zeroization for compartment switching
  - RV32IMCB backward compatibility mode
    - Run-time selectable, LEC verified with the original Ibex
  - CHERIoT-aware RISC-V debugging support



More about CHERIOT-Ibex

- Timing, area and power vs the original ibex
  - Evaluation at 5nm and 28nm process nodes
    - TSMC 5ffp and TSMC 28LP respectively
  - Fmax essentially same
  - Area increase ~10%
  - Power consumption similar

5ffp results		CHERIOT-Ibex	Ibex (16PMP)
Frequency		550MHz	550MHz
Area		2028.5	1900.38
Total Power		0.55mW	0.76mW
VT Mix	LVTLL	67.24%	62.29%
	LVT	18.73%	26.75%
	ULVT	14.04%	10.96%

- CHERIOT-SAFE FPGA Platform
  - Open-source platform targeting existing low-cost FPGA boards (Diligent Arty A7-100, etc.)
    - Internal tagged data memories
    - PLIC, DMA, UART, SPI, I2C, GPIO, JTAG
    - AXI bus fabric and external memory interface
  - Integrated CHERIoT-aware RISC-V debugging
  - Verilator/VCS simulation environment
  - CHERIoT application building

- Validation & verification ongoing
  - Formal verification against cheriot-sail
    - Univ. Oxford team
  - Simulation with randomized instruction sequence
    - Microsoft team

Software Development Flow

Add compartmentalization to C/C++

```
// Declaration adds an attribute to indicate
// the compartment containing the implementation
void  __attribute__((cheri_compartment(“kv_store_sdk")))
publish(char *key, uint8_t *buffer, size_t size);
// -------------------------------------------------------
// Call site looks like normal C.
// Compiled to a direct call in compartments build with
// -cheri-compartment=kv_store_sdk
// Compiled to a cross-domain call in all other cases.
uint8_t buffer[BUFFER_SIZE];
publish("key_id", buffer, sizeof(buffer));
```

Compile with CHERIoT LLVM compiler

Link in CHERIoT RTOS in a trusted environment

Sign and deploy the final image

Further Information

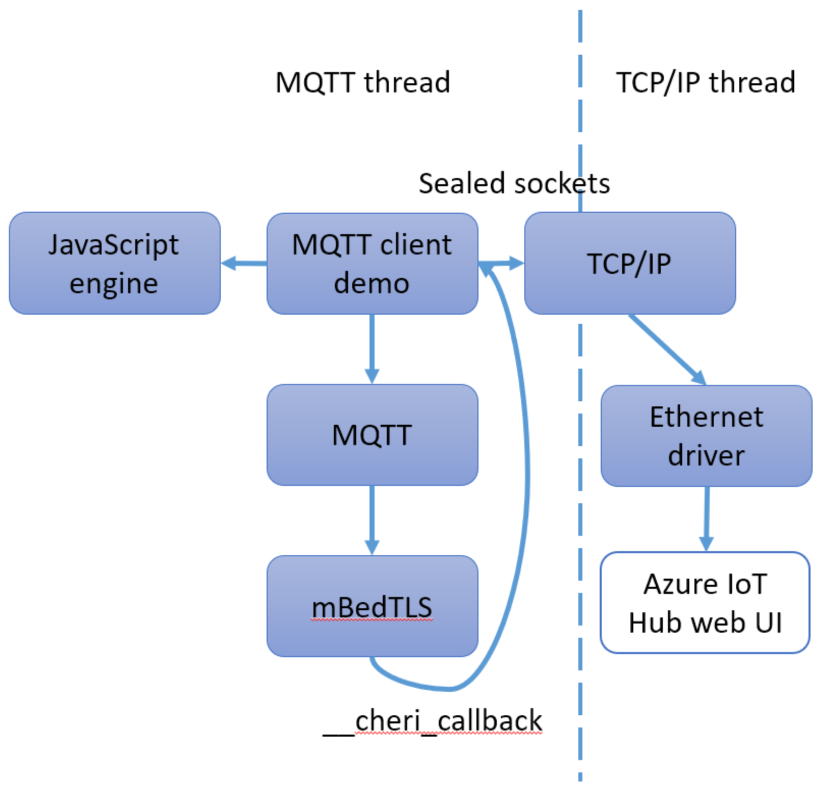
- Open-source GitHub Repos
  - *The ISA sail model and architecture specification: https://github.com/microsoft/cheriot-sail*
  - *The MCU core design: https://github.com/microsoft/cheriot-ibex*
  - *The embedded OS: https://github.com/microsoft/cheriot-rtos*
  - *The CHERIoT-enabled LLVM compiler: https://github.com/CHERIoT-Platform/llvm-project*
  - *The CHERIoT FPGA platform: https://github.com/microsoft/cheriot-safe*
- Tech reports, papers and blogposts
  - *https://cheriot.org*
  - *CHERIoT: Complete Memory Safety for Embedded Devices, IEEE/ACM MICRO 2023*
  - *CHERIoT: Rethinking security for low-cost embedded systems*
  - *First steps in CHERIoT Security Research*
  - *What's the smallest variety of CHERI? – Microsoft Security Response Center*

Backups

The CHERIoT FPGA Demo

- Application running on CHERIOT-Ibex FPGA platform
  - FPGA design
    - CHERIOT-Ibex
    - TCM/AXI/Peripherals
- CHERIoT-RTOS runtime
- CHERIoT LLVM compiler
- Application software
  - Annotated for compartment boundaries
  - MQTT
  - TPM

Link to demo video clip

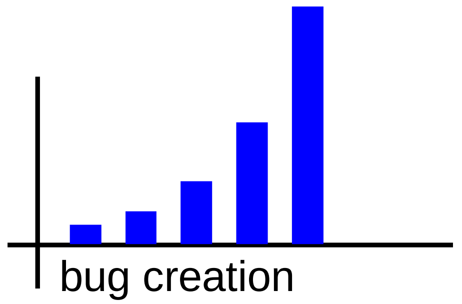


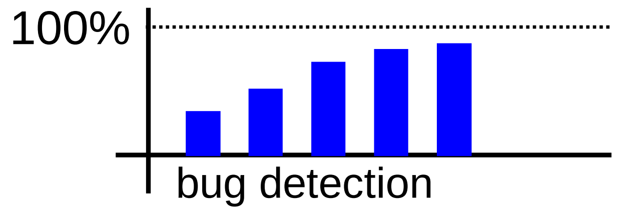
Total lines of imported code: 71,457

CHERI changes: +235, -193

Formal Verification

- Processors getting ever more complex
- The relentless *verification gap*:





- And *CHERI* processors are radically novel.

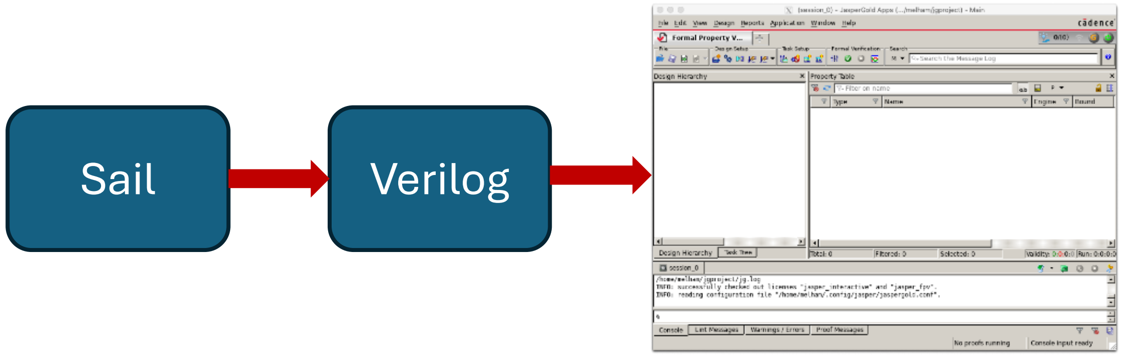
- Simulation can test only a tiny fraction of inputs and operating states.
- ***Formal verification*** - use of algorithmic mathematical reasoning to *prove* a chip design meets a specification.
- Much greater coverage and can find deep corner case bugs.

- Research programme at Oxford to verify increasingly complex CHERI processors:
  - CHERI-RISC-V Flute (FMCAD 2021)
  - CHERIoT-ibex (ongoing)
- Outcomes:
  - new methodology and tools
  - higher confidence in correctness

Gao and Melham, End-to-end formal verification of a RISC-V processor extended with capability pointers. In *Proceedings of the 21st Conference on Formal Methods in Computer-Aided Design –- FMCAD 2021*, volume 2, pp 24-33. TUI Wien Academic Press, October 2021.

Innovative use of Sail Specifications

- The Oxford verification effort is driving a collaborative effort with Cambridge on a Sail to Verilog compiler. With CHERIoT-ibex as the driving case study.
- This allows the Sail specification to be used in commercial, best-in-class formal verification software tools:



```
/home/melham/jgproject/jg.log
INFO: successfully checked out licenses "jasper_interactive" and "jasper_fpv".
INFO: reading configuration file "/home/melham/.config/jasper/jaspergold.conf".
```

- This is seeding momentum towards a general proof kit for RISC-V.
- We hope there will eventually be full commercial solutions from the EDA sector.