Proofcraft

Proof Robustness

in the seL4® verification

Gerwin Klein | Rafal Kolanski | HCSS 2021

4 Why Robustness?

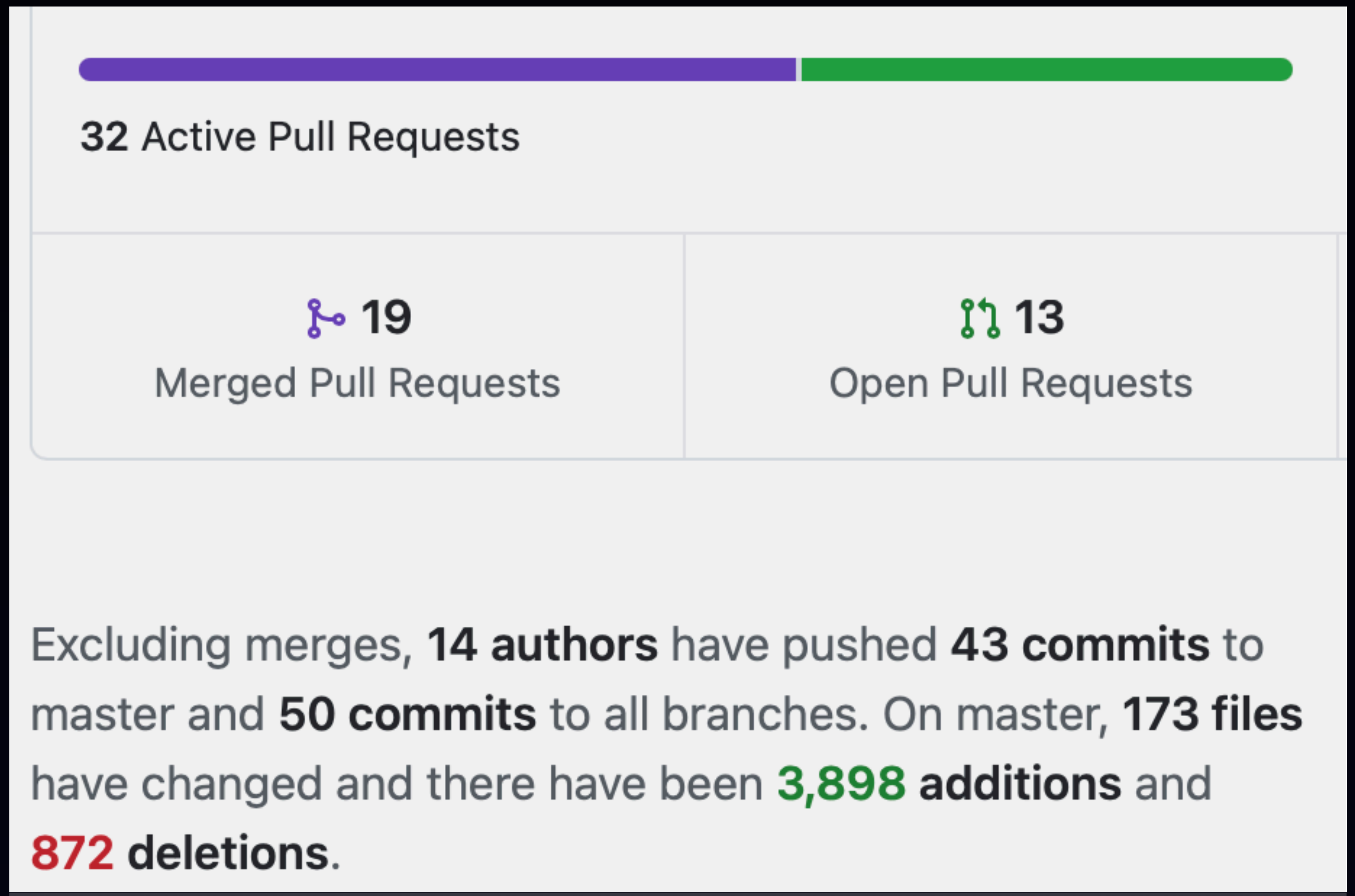
The seL4 verification

- The verified seL4 microkernel:
  - high-assurance code base
  - large, successful proof
  - **interactive** proof in Isabelle/HOL

The seL4 verification

- The verified seL4 microkernel:
  - high-assurance code base
  - large, successful proof
  - **interactive** proof in Isabelle/HOL

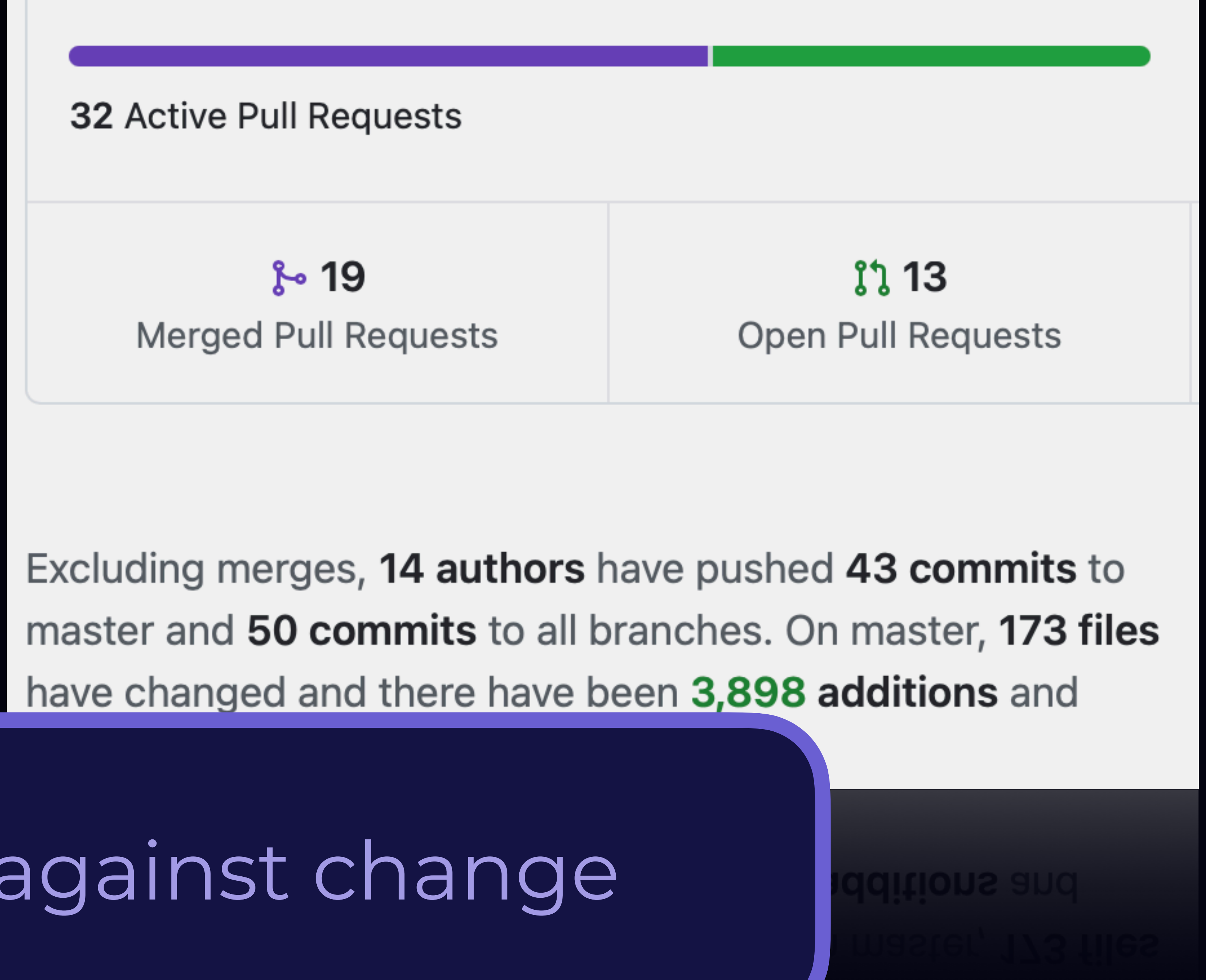
- Change is inevitable
  - change is painful in normal software
  - more painful in high-assurance software
  - proofs can help, but:
    - changing proofs is additional cost



The seL4 verification

- The verified seL4 microkernel:
  - high-assurance code base
  - large, successful proof
  - **interactive** proof in Isabelle/HOL

- Change is inevitable
  - change is painful in normal software
  - more painful in high-assurance software
  - proofs can help, but:
    - changing proofs is additional cost



Need robustness of proofs against change

Indications of robustness in seL4 proofs

- Started as research project:
  - 200k lines of proof, 10k lines of C code
  - 1 platform and architecture
  - functional correctness down to C

Indications of robustness in seL4 proofs

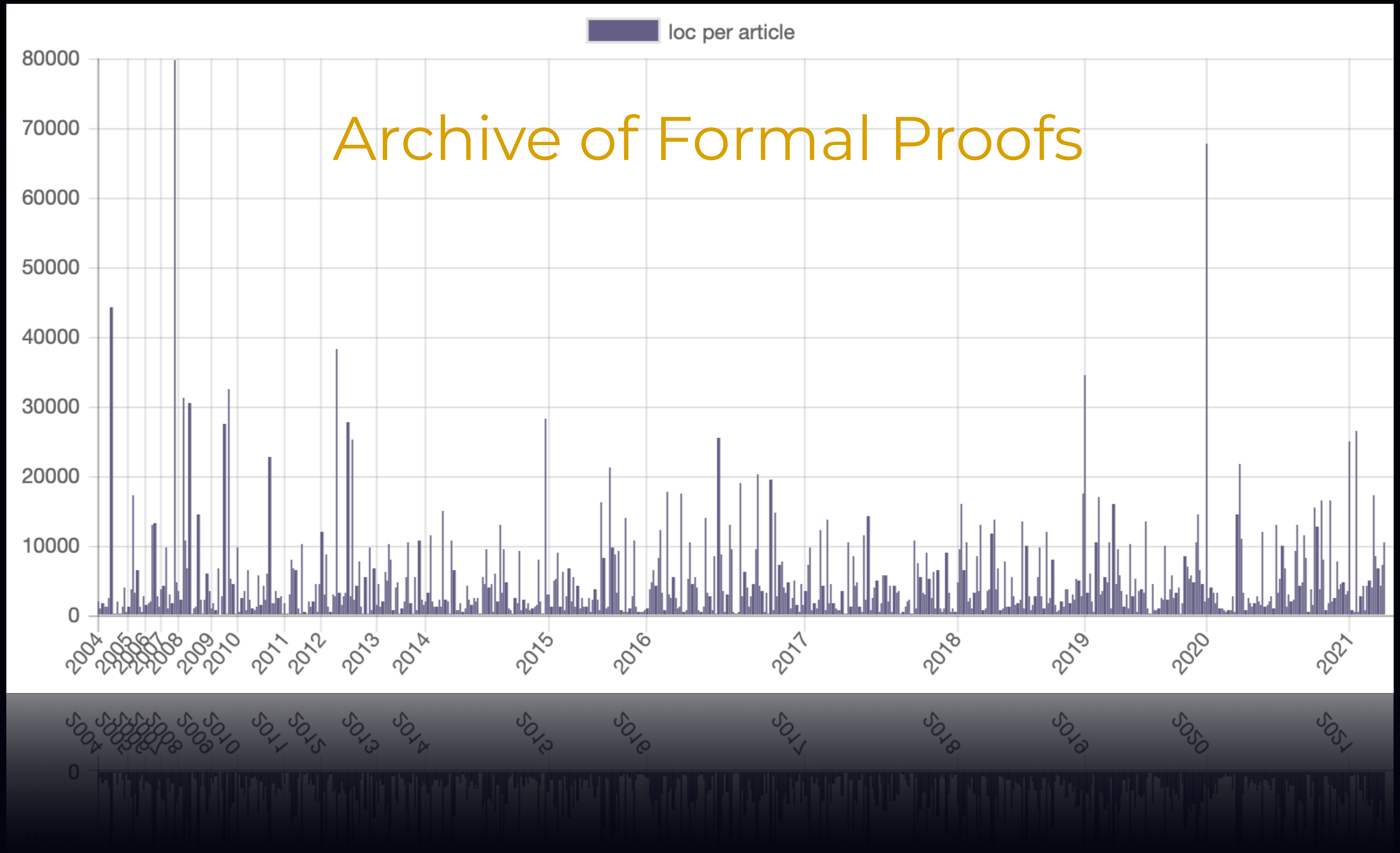
- Started as research project:
  - 200k lines of proof, 10k lines of C code
  - 1 platform and architecture
  - functional correctness down to C

- Now: larger, deeper, more versatile
  - 3 architectures, multiple configuration
  - deep security properties
  - proofs down to binaries
  - 1 million lines of proof

Indications of robustness in seL4 proofs

- Started as research project:
  - 200k lines of proof, 10k lines of C code
  - 1 platform and architecture
  - functional correctness down to C

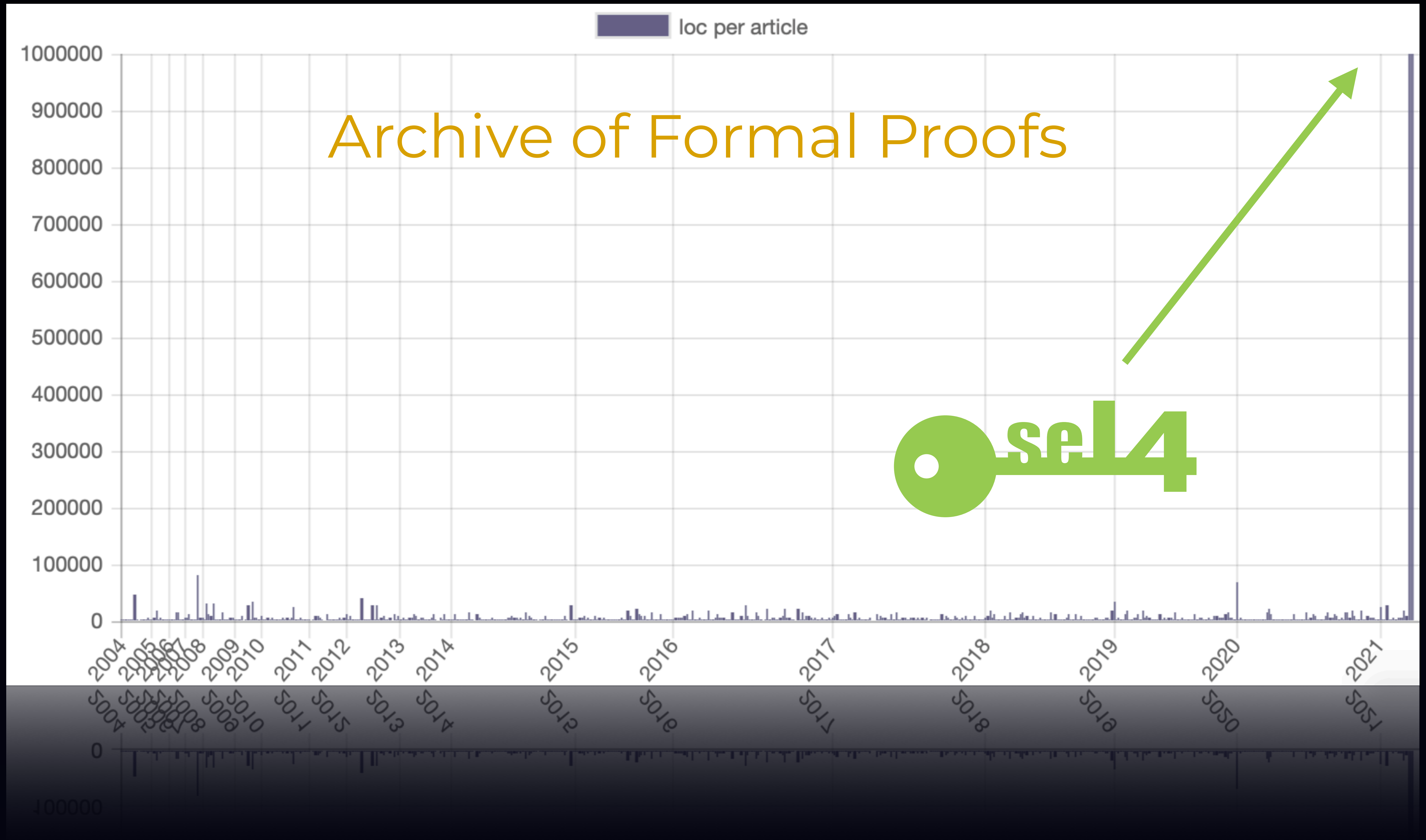
- Now: larger, deeper, more versatile
  - 3 architectures, multiple configuration
  - deep security properties
  - proofs down to binaries
  - 1 million lines of proof



Indications of robustness in seL4 proofs

- Started as research project:
  - 200k lines of proof, 10k lines of C code
  - 1 platform and architecture
  - functional correctness down to C

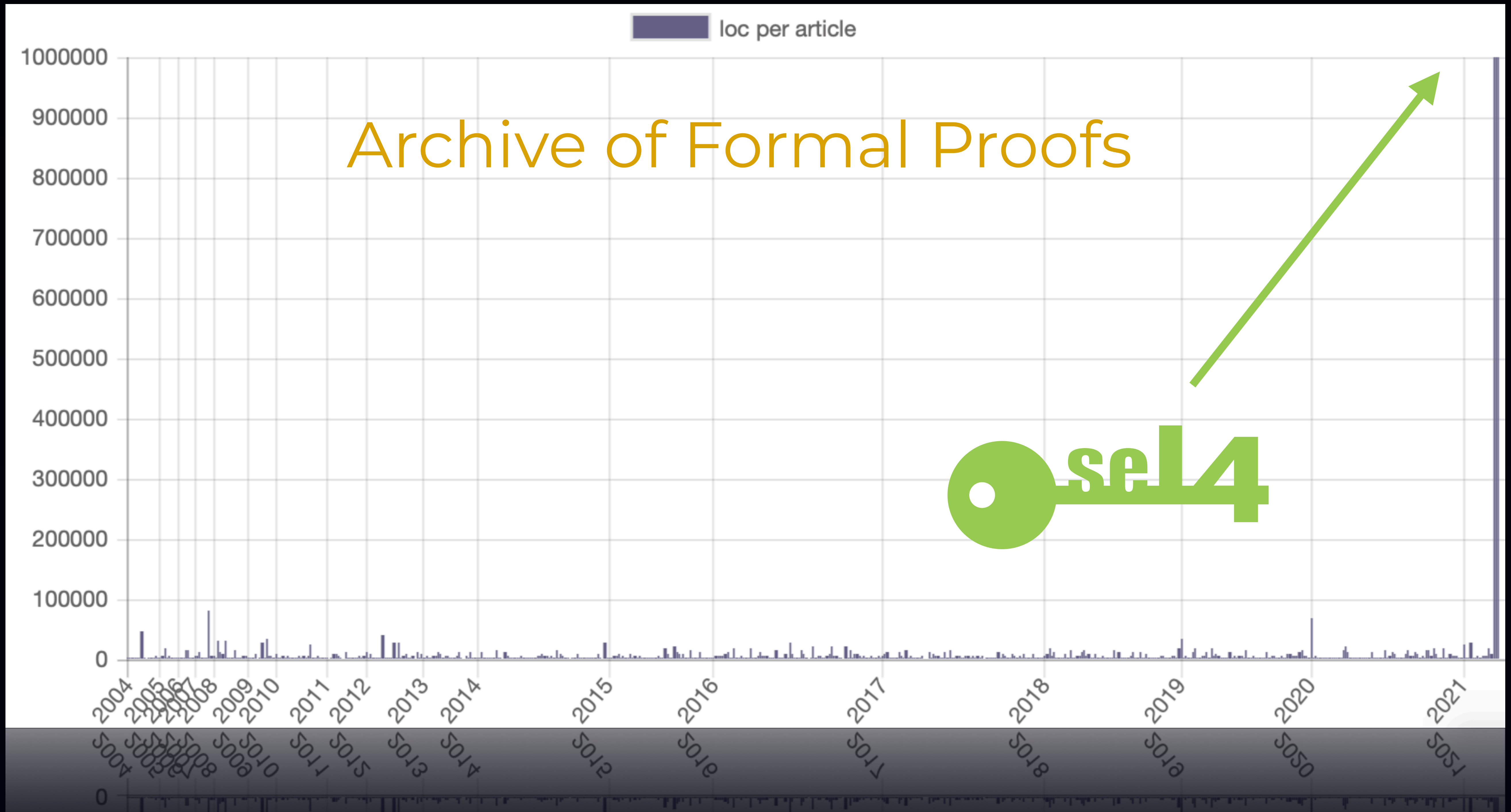
- Now: larger, deeper, more versatile
  - 3 architectures, multiple configuration
  - deep security properties
  - proofs down to binaries
  - 1 million lines of proof



Indications of robustness in seL4 proofs

- Started as research project:
  - 200k lines of proof, 10k lines of C code
  - 1 platform and architecture
  - functional correctness down to C

- Now: larger, deeper, more versatile
  - 3 architectures, multiple configuration
  - deep security properties
  - proofs down to binaries
  - 1 million lines of proof



- And: we, the proof engineers, are still alive...

Why care about robustness?

Why care about robustness?

- Mathematical truth vs customer wishes

Why care about robustness?

- Mathematical truth vs customer wishes
- Research project vs commercial interest

Why care about robustness?

- Mathematical truth vs customer wishes
- Research project vs commercial interest
- Scale of proof and scale of team

Why care about robustness?

- Mathematical truth vs customer wishes
- Research project vs commercial interest
- Scale of proof and scale of team
- Agility
  - Cost & Effort
  - Time to market
  - Open-source contributions

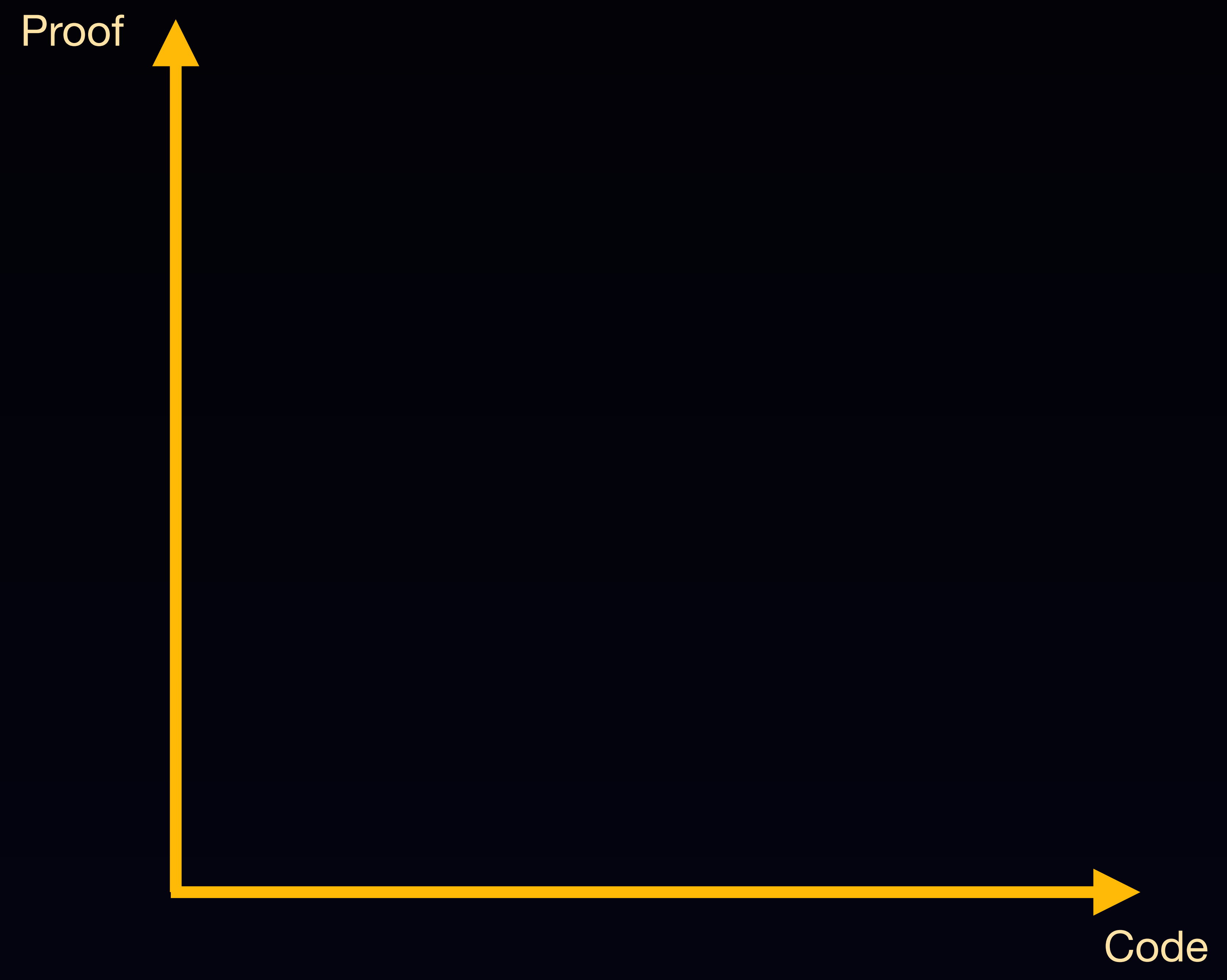
Dimensions of change

- Code
  - new feature (new system-call), new architecture (RISC-V), new platform (imx8)
  - refactoring
  - optimisation

Dimensions of change

- Code
  - new feature (new system-call), new architecture (RISC-V), new platform (imx8)
  - refactoring
  - optimisation

- Proof
  - new property
  - refactoring for faster proofs
  - refactoring for nicer proofs

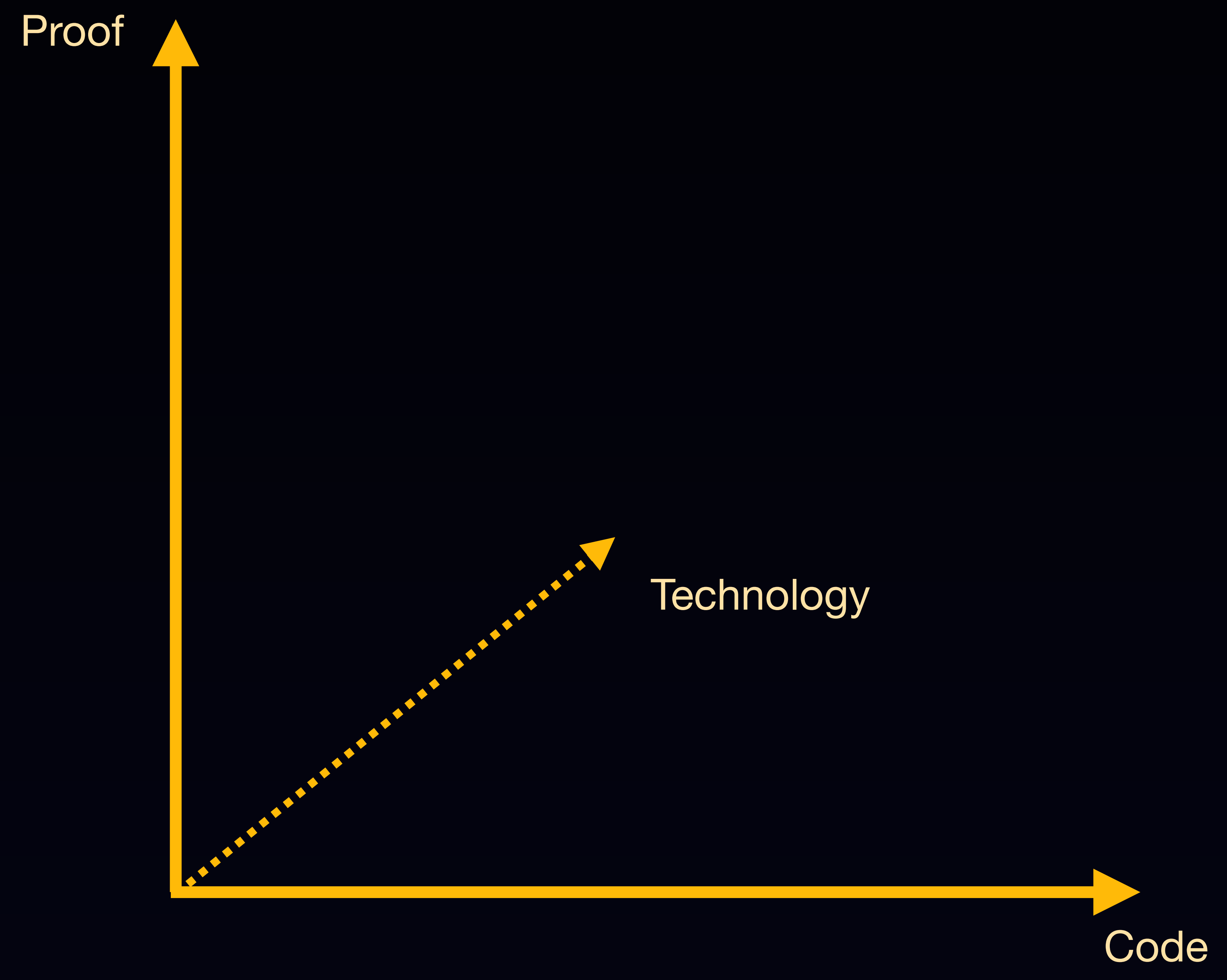


Dimensions of change

- Code
  - new feature (new system-call), new architecture (RISC-V), new platform (imx8)
  - refactoring
  - optimisation

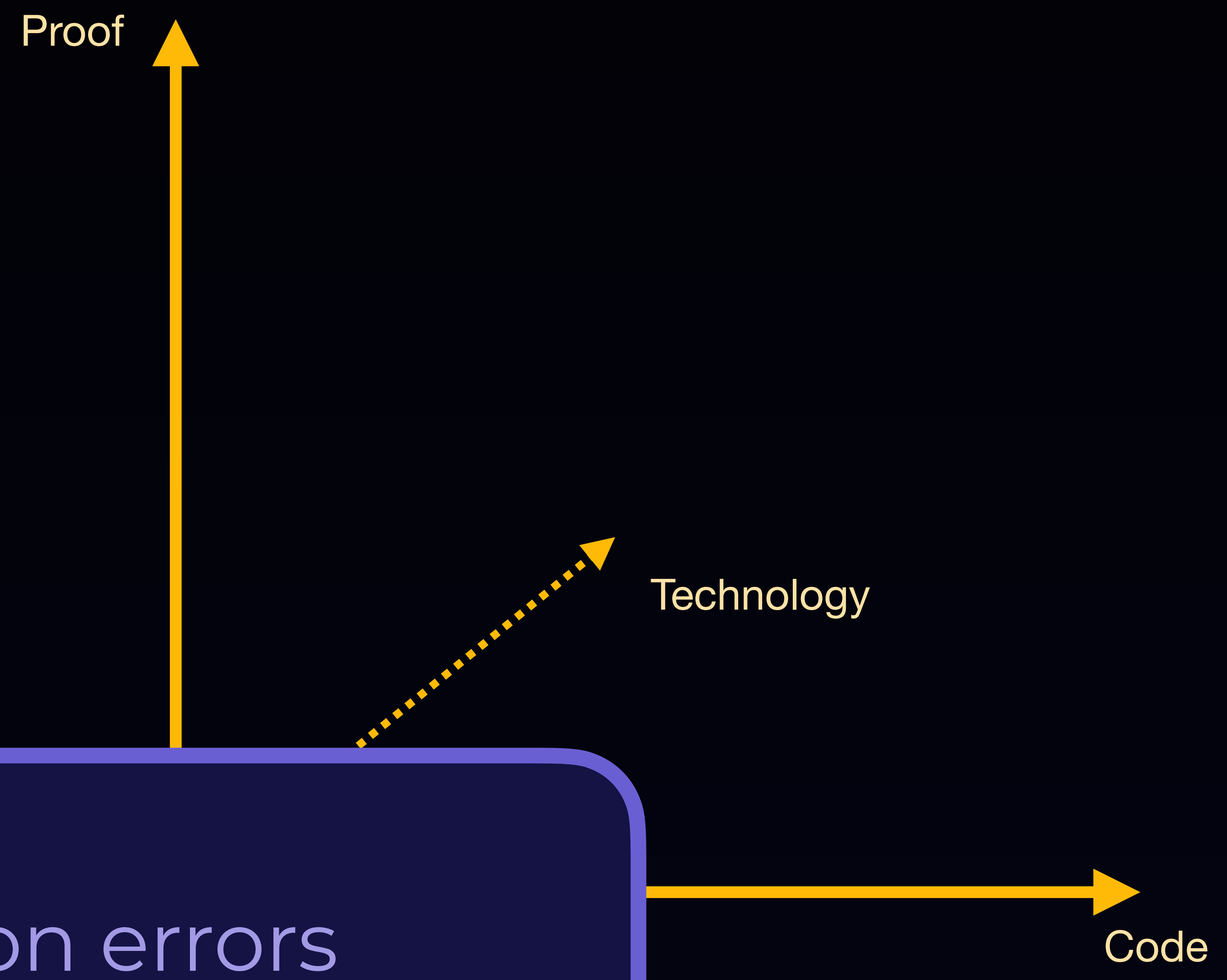
- Proof
  - new property
  - refactoring for faster proofs
  - refactoring for nicer proofs

- Technology
  - prover update



Dimensions of change

- Code
  - new feature (new system-call), new architecture (RISC-V), new platform (imx8)
  - refactoring
  - optimisation
- Proof
  - new property
  - refactoring for faster proofs
  - refactoring for nicer proofs
- Technology
  - prover update



Not: implementation errors

(but maybe spec changes)

4 Dealing with Change

Approaches to increasing robustness

- Types
- Automation
- Semantic
  - Abstraction
  - Modularity & Parametricity
- Process

Types

- Automatic. Free theorems!

Types

- Automatic. Free theorems!
- Basic:
  - Make invalid states unrepresentable
  - Invariants: "ASIDs have at most 7 bits"
  - Effect level: "function is read-only, but can fail"

Types

- Automatic. Free theorems!
- Basic:
  - Make invalid states unrepresentable
  - Invariants: "ASIDs have at most 7 bits"
  - Effect level: "function is read-only, but can fail"
- More advanced:
  - State projections to constrain properties:
    - "only depends on TCB contents"
  - State projections to constrain effects (lenses):
    - "only operates on threads"
  - Combination produces free independence theorems

Types

- Automatic. Free theorems!
- Basic:
  - Make invalid states unrepresentable
  - Invariants: "ASIDs have at most 7 bits"
  - Effect level: "function is read-only, but can fail"
- More advanced:
  - State projections to constrain properties:

- Why not just always more types?
  - introducing more types also is change, needs effort/benefit trade-off
  - can be too much hassle (e.g. 7-bit word in Haskell)
  - potentially type system not powerful enough

- Types
- **Automation**
- Semantic
  - Abstraction
  - Modularity & Parametricity
- Process

Automation

- Automation is cheaper than manual labor
  - Higher chance that proof still works
  - But:
    - Needs more expertise to implement
    - Needs foresight to help against change
  - Information density in some seL4 proofs still low
- The "easy" way out
  - Can replace other techniques
- Every bit helps
  - Automating small tasks frees up time for deeper things

Automation

- Automation is cheaper than manual labor
  - Higher chance that proof still works
  - But:
    - Needs more expertise to implement
    - Needs foresight to help against change
  - Information density in some seL4 proofs still low
- The "easy" way out
  - Can replace other techniques
- Every bit helps
  - Automating small tasks frees up time for deeper things

seL4 proof example: `crunches`

```
crunches tcbSchedAppend, tcbSchedDequeue, tcbSchedE
  for typ_at'[wp]: "λs. P (typ_at' T p s)"
  and tcb_at'[wp]: "tcb_at' t"
  and ctes_of[wp]: "λs. P (ctes_of s)"
  and irq_states[wp]: valid_irq_states'
  and irq_node'[wp]: "λs. P (irq_node' s)"
  and ct'[wp]: "λs. P (ksCurThread s)"
  and global_refs'[wp]: valid_global_refs'
  and ifunsafe'[wp]: if_unsafe_then_cap'
  and cap_to'[wp]: "ex_nonz_cap_to' p"
  and state_refs_of'[wp]: "λs. P (state_refs_of' s)
  and idle'[wp]: valid_idle'
  (simp: unless_def crunch_simps)
```

Automation

- Automation is cheaper than manual labor
  - Higher chance that proof still works
  - But:
    - Needs more expertise to implement
    - Needs foresight to help against change
  - Information density in some seL4 proofs still low
- The "easy" way out
  - Can replace other techniques
- Every bit helps
  - Automating small tasks frees up time for deeper t

seL4 proof example: `crunches`

```
crunches tcbSchedAppend, tcbSchedDequeue, tcbSchedE
  for typ_at'[wp]: "λs. P (typ_at' T p s)"
  and tcb_at'[wp]: "tcb_at' t"
  and ctes_of[wp]: "λs. P (ctes_of s)"
  and irq_states[wp]: valid_irq_states'
  and irq_node'[wp]: "λs. P (irq_node' s)"
  and ct'[wp]: "λs. P (ksCurThread s)"
  and global_refs'[wp]: valid_global_refs'
  and ifunsafe'[wp]: if_unsafe_then_cap'
  and cap_to'[wp]: "ex_nonz_cap_to' p"
  and state_refs_of'[wp]: "λs. P (state_refs_of' s)
  and idle'[wp]: valid_idle'
  (simp: unless_def crunch_simps)
```

- Why not just automate the whole proof?
  - First examples exist, but not there yet for our domain
  - Automatic proof repair: first steps exist, but much more to do

- Types
- Automation
- **Semantic**
  - **Abstraction**
  - **Modularity & Parametricity**
- Process

Semantic approaches

- Abstraction, Parametricity, Modularity

Semantic approaches

- Abstraction, Parametricity, Modularity

- Why do they work for robustness?
  - Hide details
    - "Free" robustness against change of these details
  - Can be extremely effective

Semantic approaches

- Abstraction, Parametricity, Modularity

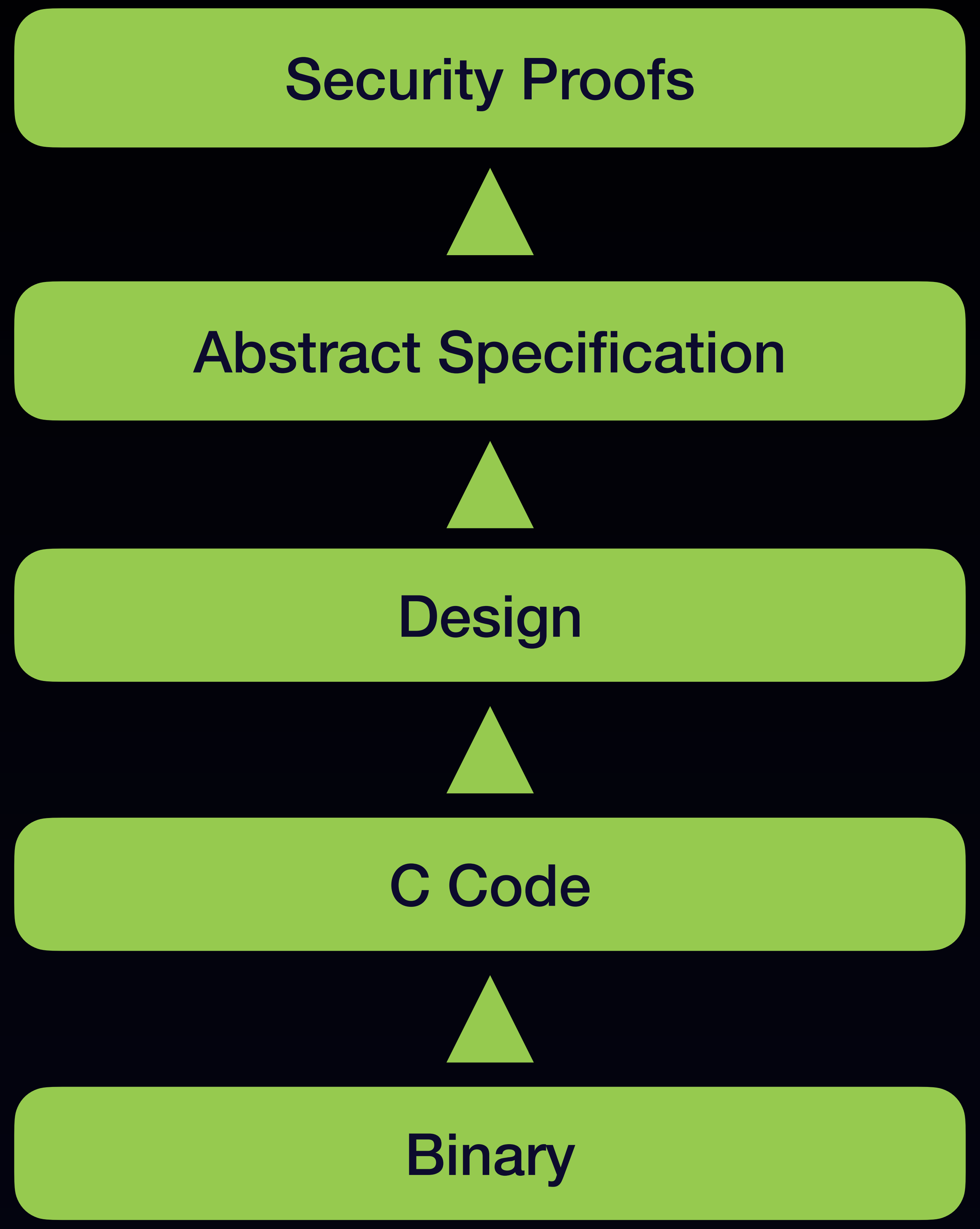
- Why do they work for robustness?
  - Hide details
    - "Free" robustness against change of these details
  - Can be extremely effective

- Why not just use them everywhere?
  - Requires expertise and foresight
  - If change breaks the abstraction or interface, cost can be high

Abstraction

- Abstraction in seL4 proof stack:
  - Abstract spec + refinement stack
  - Security proofs much lower effort
  - Robust against many optimisations

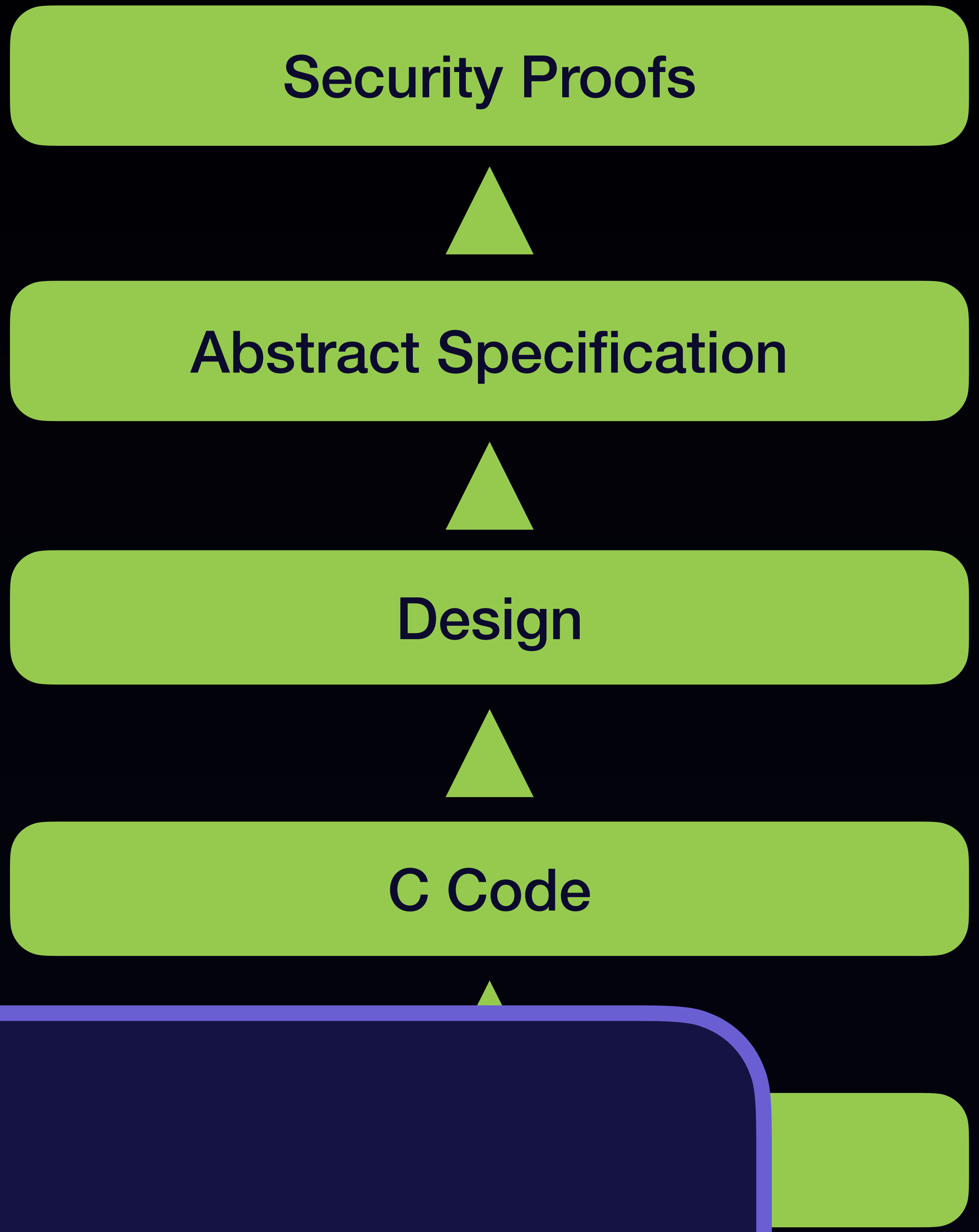
- Abstraction in proof scripts:
  - Use rule collections (bit_sizes) instead of specific rule (PT_16_bit_def)
  - Use proof method (unfold_bit_size) instead of rule applications (simp add: ..)



Abstraction

- Abstraction in seL4 proof stack:
  - Abstract spec + refinement stack
  - Security proofs much lower effort
  - Robust against many optimisations

- Abstraction in proof scripts:
  - Use rule collections (bit_sizes) instead of specific rule (PT_16_bit_def)
  - Use proof method (unfold_bit_size) instead of rule applications (simp add: ...)



- Why not just more abstraction?
  - Needs brain power and experience (expensive)
  - Needs abstractable surface
  - Counter-examples: mixed-criticality features, multicore

Modularity & Parametricity

- Example: split proof into arch-specific and generic part
  - Generic part is a parametric module
  - Has been effective, but used only for part of proof
  - More of this in development
- Example: parametric page table structures in seL4/RISC-V
  - Regular structure
  - Much faster proof completion
- Example: proof libraries and tools
  - C-Parser, AutoCorres, wp, word library, monad library
  - Can be maintained independently
  - But: tech upgrade can break proofs

Modularity & Parametricity

- Example: split proof into arch-specific and generic part
  - Generic part is a parametric module
  - Has been effective, but used only for part of proof
  - More of this in development
- Example: parametric page table structures in seL4/RISC-V
  - Regular structure
  - Much faster proof completion
- Example: proof libraries and tools
  - C-Parser, AutoCorres, wp, word library, monad library
  - Can be maintained independently
  - But: tech upgrad

- Why not just everything modular?
  - Yes, as far as possible
  - Can fight with code structure and performance

- Types
- Automation
- Semantic
  - Abstraction
  - Modularity & Parametricity
- **Process**

Process

- Code change often originates outside verification
  - New feature idea, platform port, optimisation, etc
  - Open-source contributions

Process

- Code change often originates outside verification
  - New feature idea, platform port, optimisation, etc
  - Open-source contributions
- Many on non-verified platforms or feature-combinations
  - Should be invisible to the proofs
  - But are not always
  - Provide way for developer to check proof impact:
    - Pre-process test on GitHub
    - Proof testboard

```
if (config_set(CONFIG_ARM_HYPERVISOR_
```

```
if (0 && !0) {
```

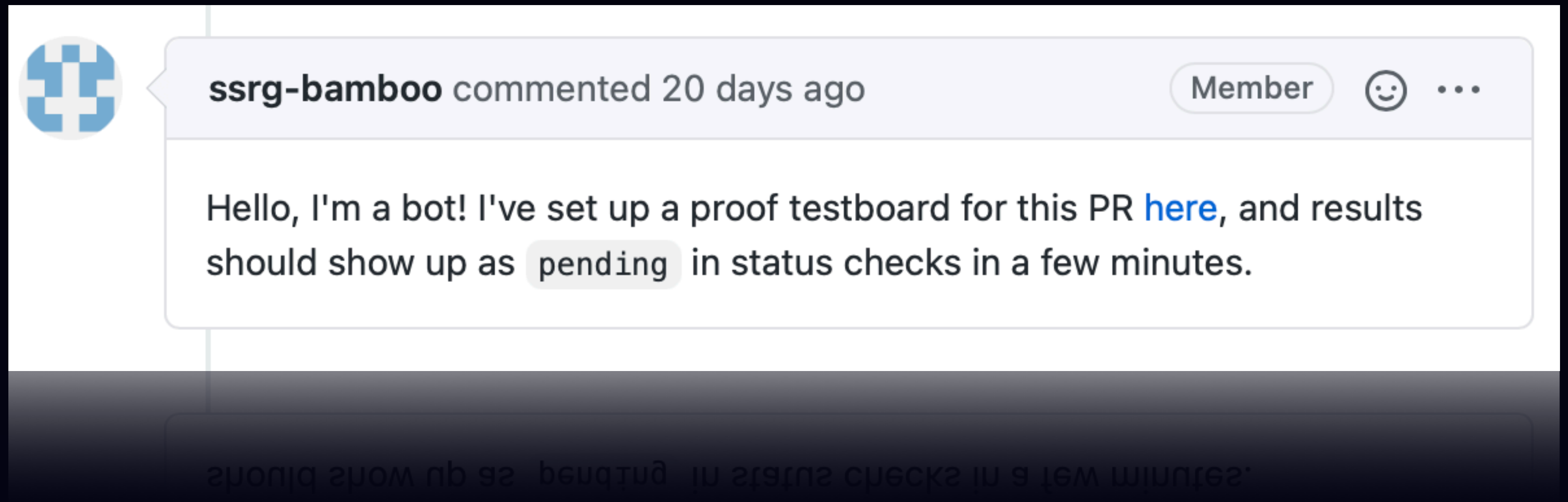
Process

- Code change often originates outside verification
  - New feature idea, platform port, optimisation, etc
  - Open-source contributions
- Many on non-verified platforms or feature-combinations
  - Should be invisible to the proofs
  - But are not always
  - Provide way for developer to check proof impact:
    - Pre-process test on GitHub
    - Proof testboard

```
if (config_set(CONFIG_ARM_HYPERVISOR_
```

```
if (0 && !0) {
```

- CI pipeline for seL4 proofs:
  - Automatically check proof for code and proof changes
  - Automatically record which proof versions apply to which code version
  - Proofs always releasable



We covered:

- Types
- Automation
- Semantic
    - Abstraction
    - Modularity & Parametricity
- Process

4 Summary

Some robustness required

- Managing software is hard
  Large-scale software engineering is far from solved
- Should not expect large-scale proof engineering to be easy

Some robustness required

- Managing software is hard
  Large-scale software engineering is far from solved
- Should not expect large-scale proof engineering to be easy

- High assurance still takes time, still not cheap
- But:
  - Situation is constantly improving
  - High assurance can be continually maintained
  - Robustness can be increased
  - **Commercially viable**

Some robustness required

- Managing
  Large-sca
- Should no

We're betting the company on it

Proofcraft

https://proofcraft.systems

- **Commercially viable**