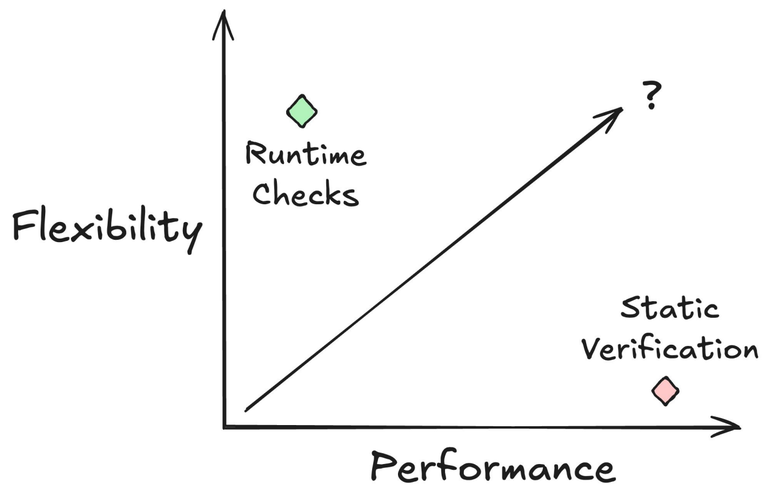
Fast, Flexible, and Practical Kernel Extensions

Kumar Kartikeya Dwivedi Rishabh Iyer Sanidhya Kashyap

Problem: Kernel extensions are either fast or flexible – not both

Static Verification	Runtime Checks	Current Solutions
Constrain admitted code	Allow arbitrary code	Sacrifice flexibility or performance
Low overhead	Poor performance	Use specific programming languages
Limit flexibility	Hurt performance	A difficult programming model



Insight: Separate kernel safety properties, use distinct mechanisms to enforce them

Kernel Interface Compliance	Extension Correctness
Kernel interface safety	Extension memory safety, termination
Narrow, well-defined	Broad, diverse behavior
Kernel-owned resources	Extension-owned resources

Static verification	Runtime checks
Enforce kernel interface compliance	Enforce extension correctness
Verify nuanced semantics	Permit diverse code

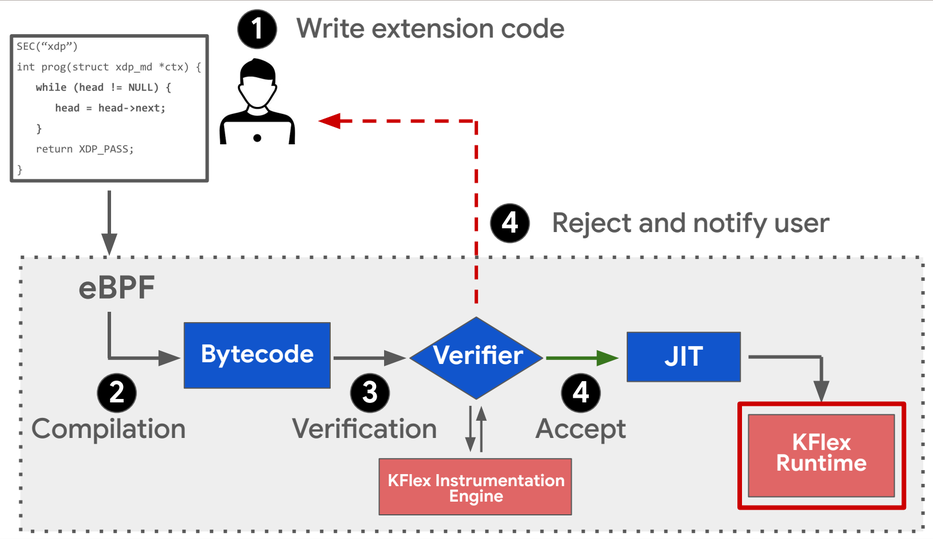
Co-design runtime checks with verification to reduce overhead

KFlex: Balances flexibility and performance

Key Ideas

- Separate kernel safety into two sub-properties
  - Kernel Interface Compliance
  - Extension Correctness
- Use dedicated, appropriate mechanisms to enforce safety for each sub-property
- Use static verification to reduce overhead of runtime mechanisms whenever possible

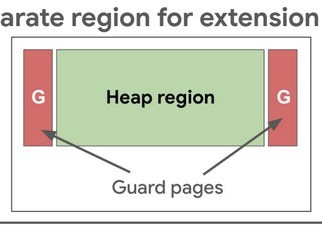
Overview

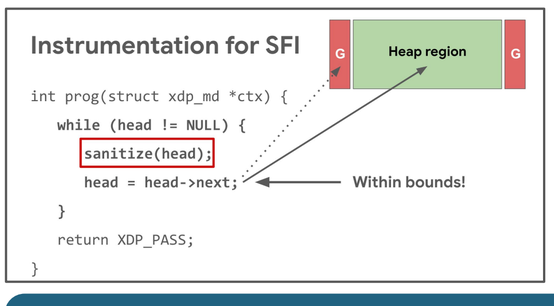


Design

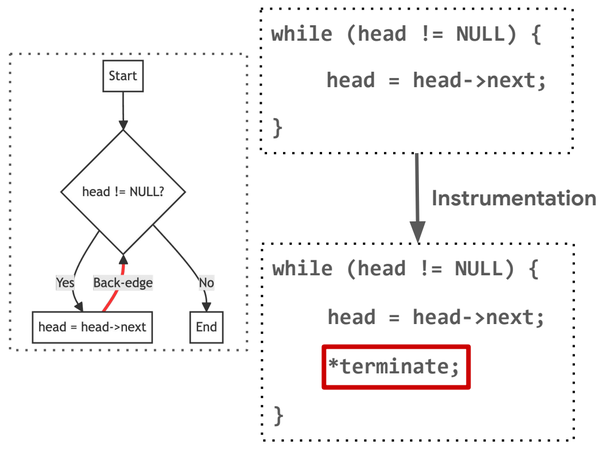
Heaps + SFI for extension memory safety

Separate region for extension data

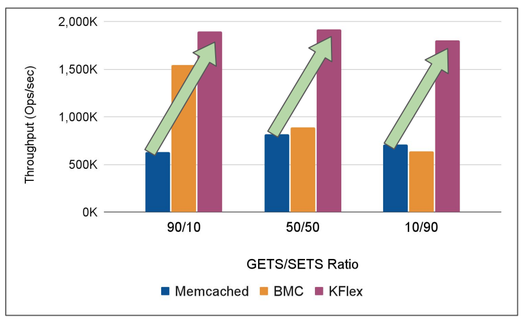


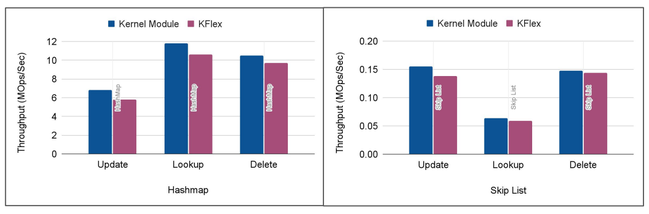


Extension cancellations for safe termination



Evaluation





- Up to 3x more throughput
- Offload both SETS/GETS
- No memory waste

- 7% latency overhead, 30% throughput overhead
- Implement arbitrary data structures
- 76% less instrumentation due to co-design

KFlex enables fast, flexible, and practical kernel extensions