KU LEUVEN

Capability machines as target for secure compilation

Dominique Devriese

imec-DistriNet, KU Leuven

Dagstuhl Seminar on Secure Compilation

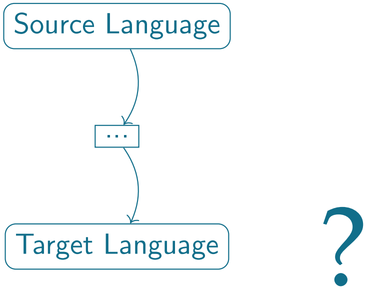
A realistic secure compiler?

$$\begin{tikzcd}
\boxed{\text{Source Language}} \arrow[d, bend left] \\
\boxed{\cdots} \arrow[d, bend left] \\
\boxed{\text{Target Language}}
\end{tikzcd}$$

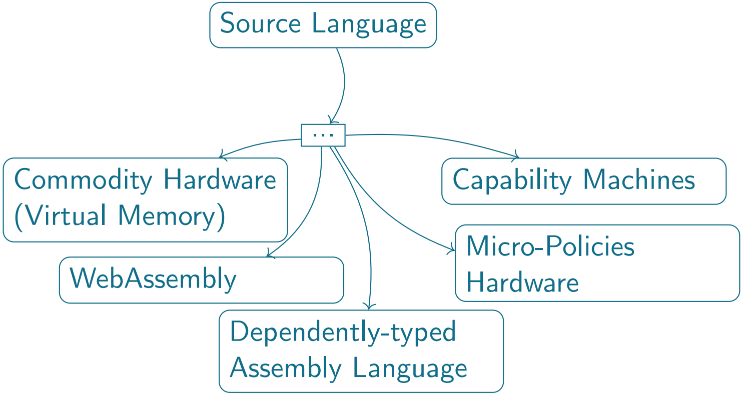
A realistic secure compiler?

$$\begin{tikzcd}
\boxed{\text{Source Language}} \arrow[d] & \\
\boxed{\cdots} \arrow[d] & ? \\
\boxed{\text{Target Language}} &
\end{tikzcd}$$

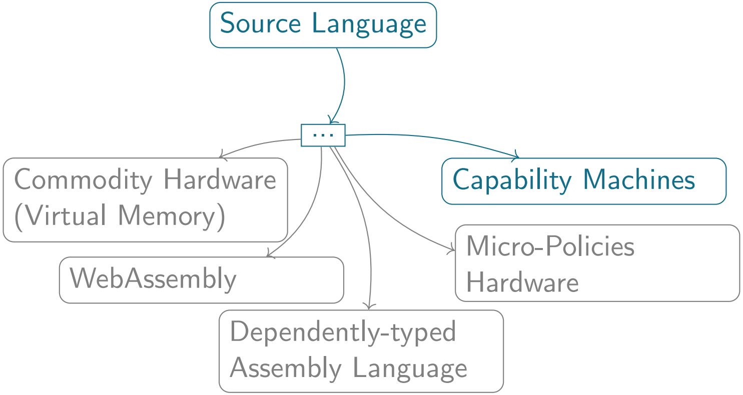
A realistic secure compiler?



A realistic secure compiler?



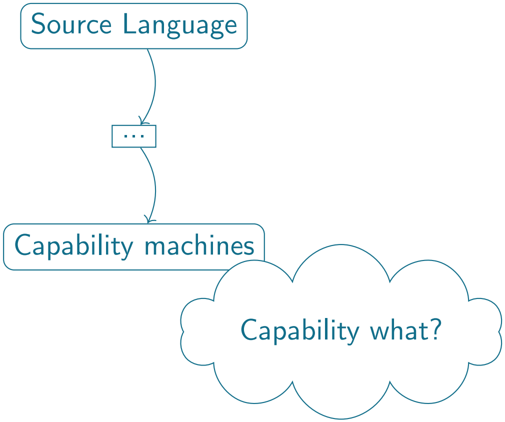
A realistic secure compiler?



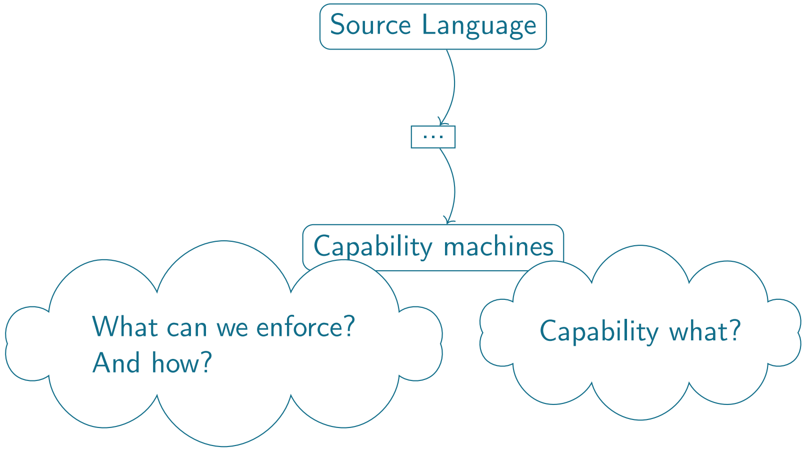
A realistic secure compiler?

$$\begin{tikzcd}
\boxed{\text{Source Language}} \arrow[d, bend left] \\
\boxed{\cdots} \arrow[d, bend left] \\
\boxed{\text{Capability machines}}
\end{tikzcd}$$

A realistic secure compiler?



A realistic secure compiler?



Outline

Outline

Capability Machines: Long history

Long history:

- [Dennis and Van Horn, 1966]
- Cambridge CAP, Hydra, PDP-1, ... [see Levy, "Capability-based Computer Systems", 1984]

Side product: Object capabilities in programming languages

- [Morris, 1973], E [Miller, 2006], Google Caja [Miller et al., 2008], JavaScript strict mode [ECMA, 2009]

CHERI

- CHERI capability machine [Woodruff et al., 2014] [Watson et al., 2015] [. . .]
- (long list of contributors)
- Hybrid security model
  - Virtual memory + capabilities
  - for gradual adoption
- 64-bit MIPS-based implementation, runs on FPGA
- Toolchain: modified CLang, LLVM, CheriBSD, QEmu etc.

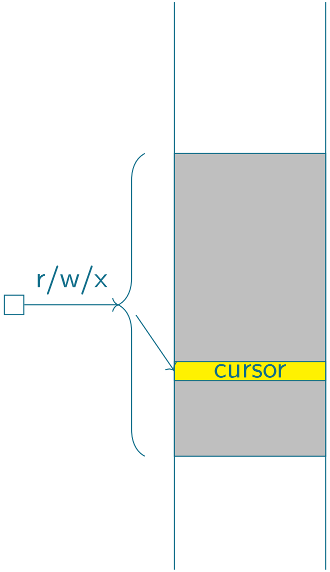
Capability Machine Principles

- Principles:
  - Least privilege
  - Fine-grained compartmentalization
  - Intentional use
- Practically
  - Capabilities represent authority
  - ~~Pointers = integers~~ Capabilities $\neq$ integers
    - tagged memory
    - guarded manipulation

Outline

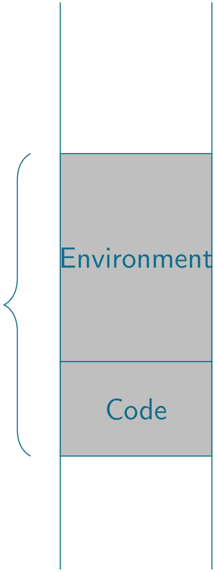
Memory capabilities

- Authority over memory array
- read/write/execute (depending on permissions)
- Hardware bounds check on every load/store



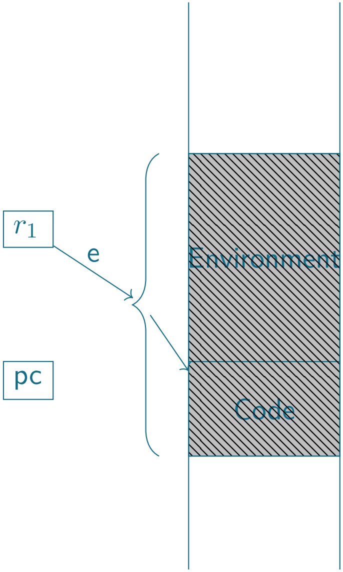
Object capabilities

- Low-overhead privilege separation
- $\sim$ encapsulated closures
  - Code pointer p
  - Environment $e = (e_1, \cdots, e_n)$
  - e becomes accessible after jumping to (p, e)
- Two example designs:
  - M-Machine [Carter,1994]
  - CHERI



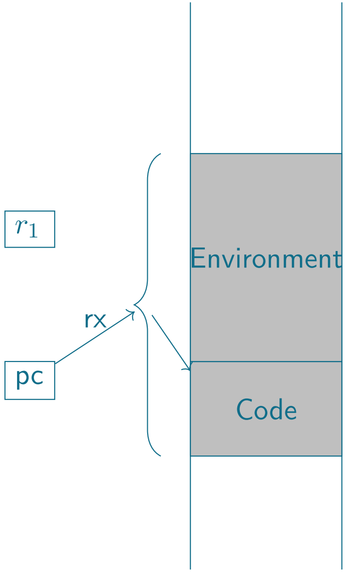
Object capabilities

- Low-overhead privilege separation
- $\sim$ encapsulated closures
  - Code pointer p
  - Environment $e = (e_1, \cdots, e_n)$
  - e becomes accessible after jumping to (p, e)
- Two example designs:
  - M-Machine [Carter,1994]
  - CHERI



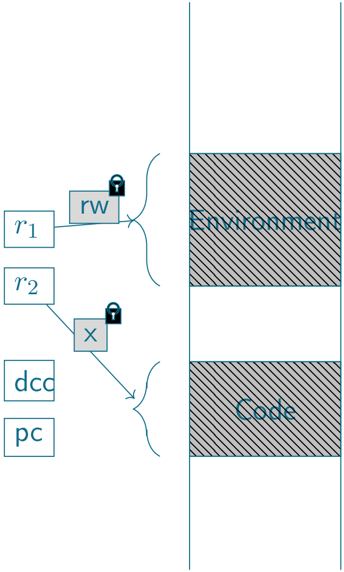
Object capabilities

- Low-overhead privilege separation
- $\sim$ encapsulated closures
  - Code pointer p
  - Environment $e = (e_1, \cdots, e_n)$
  - e becomes accessible after jumping to (p, e)
- Two example designs:
  - M-Machine [Carter,1994]
  - CHERI



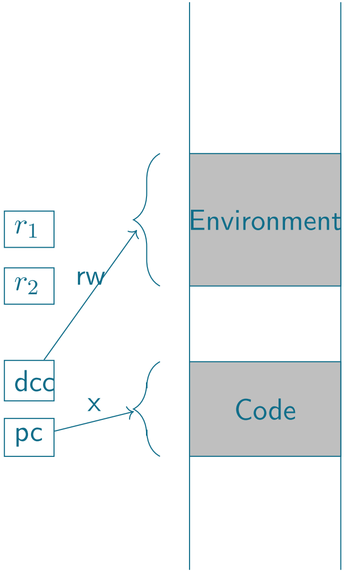
Object capabilities

- Low-overhead privilege separation
- $\sim$ encapsulated closures
  - Code pointer p
  - Environment $e = (e_1, \cdots, e_n)$
  - e becomes accessible after jumping to (p, e)
- Two example designs:
  - M-Machine [Carter,1994]
  - CHERI



Object capabilities

- Low-overhead privilege separation
- $\sim$ encapsulated closures
  - Code pointer p
  - Environment $e = (e_1, \cdots, e_n)$
  - e becomes accessible after jumping to (p, e)
- Two example designs:
  - M-Machine [Carter,1994]
  - CHERI



Essential features: applications

- Encapsulate C modules
  - (WIP)
  - El-Korashy, Tsampas, Patrignani, Garg, Piessens, Devriese
  - Enforce module-local state ("static")
- Encapsulate objects
  - Chisnall et al., ASPLOS 2017
  - Encapsulate Java objects in C/JNI code
  - direct buffers, syscalls, revocation etc.
- Other applications:
  - Fat pointers
  - Encapsulate closures?
  - Dynamic contract/type checking?

Outline

Revoking capabilities

- *Temporary* authority delegation
- Expensive in general
  - Ocaps: indirection
  - Mark-and-sweep revocation [Chisnall et al., ASPLOS 2017]
- Cheap restricted forms
  - CHERI: Local capabilities
  - Linear capabilities

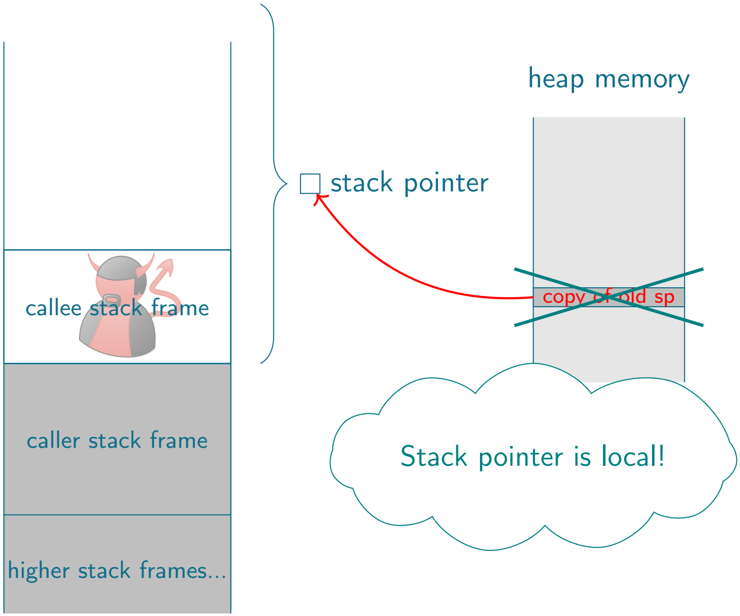
Local capabilities

- supported by CHERI
- Rules:
  - Capabilities can be tagged as local
  - Local capabilities cannot be stored in memory
  - Except through write-local capabilities
  - ... which have to be local themselves
- Application (1/2):
  - stack and return capabilities in CHERI
    - per-compartment stack
    - local stack pointer
    - protect components against themselves
    - (no actual revocation)

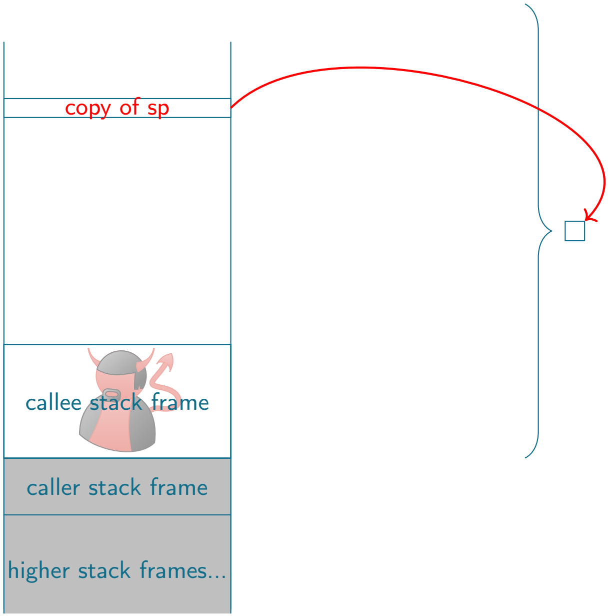
Local capabilities

- Application (2/2):
  - Skorstengaard, Devriese, Birkedal, ESOP 2018
    - single stack
    - local stack and return caps
    - revoke after return
    - requires stack clearing
- Revoking local capabilities requires efficient memory clearing
  - realistic?
  - Perhaps... [Joannou et al., ICCD 2017]

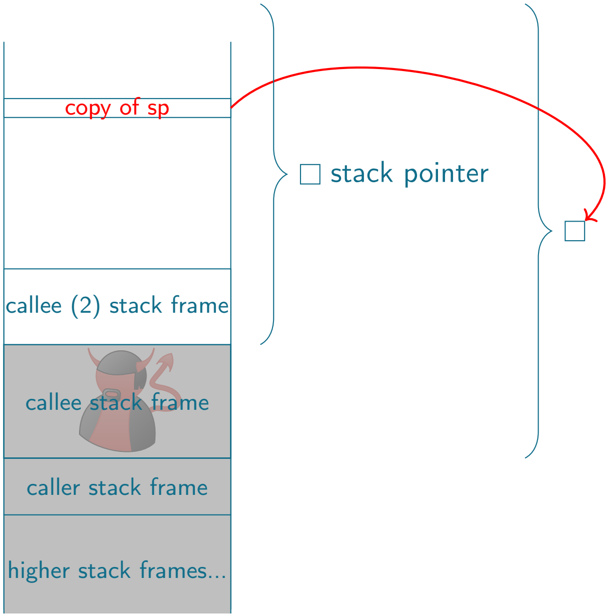
Local Stack Capabilities Prevent Attack 1



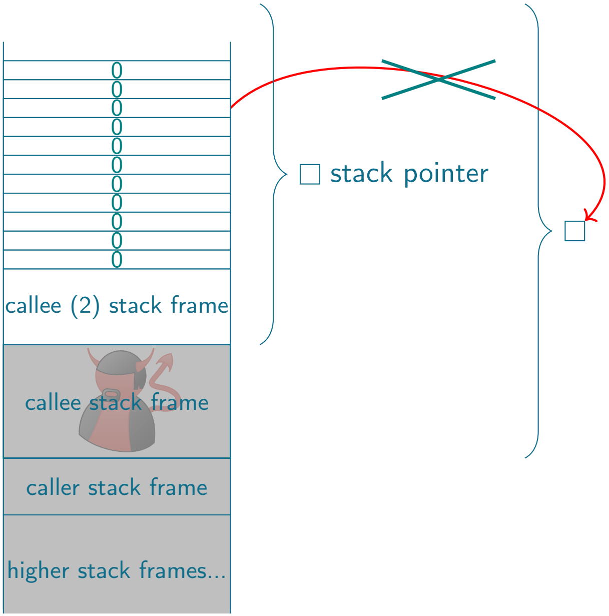
Local caps: Stack Clearing Prevents Attack 2



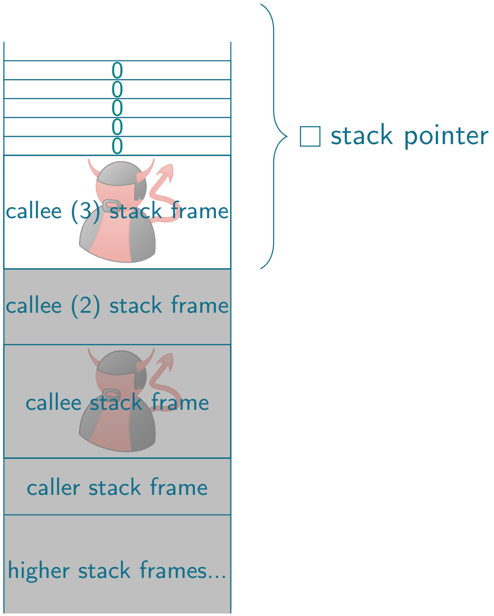
Local caps: Stack Clearing Prevents Attack 2



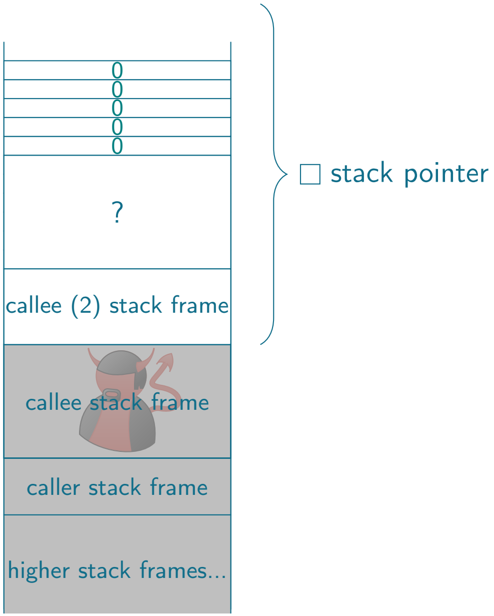
Local caps: Stack Clearing Prevents Attack 2



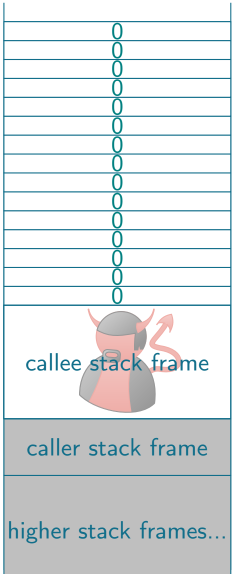
Local caps: Stack Clearing Prevents Attack 2



Local caps: Stack Clearing Prevents Attack 2



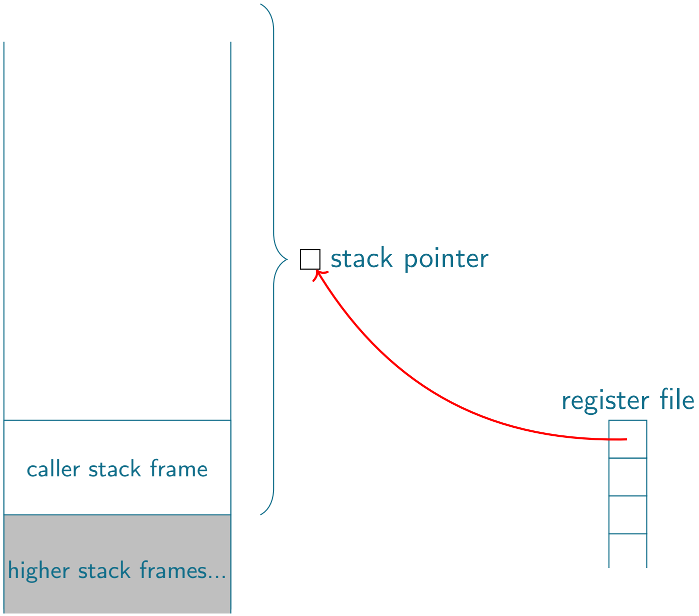
Local caps: Stack Clearing Prevents Attack 2



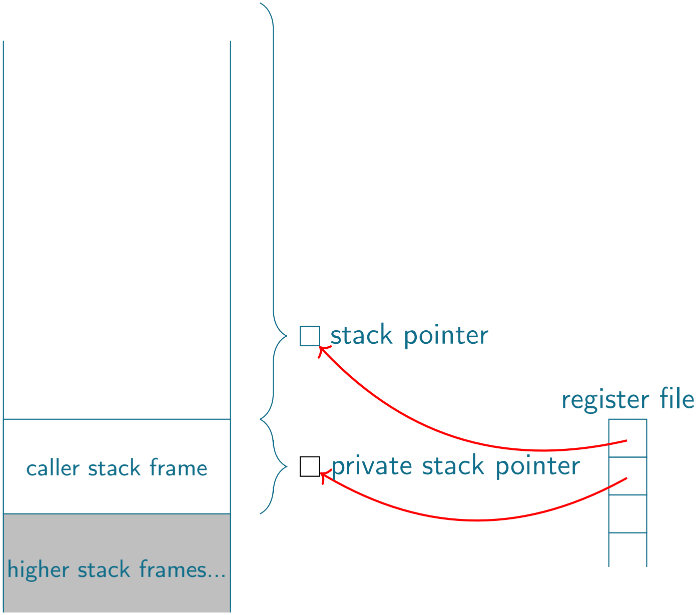
Linear capabilities

- Rules:
  - Non-duplicable
  - Some additional instructions (split, join)
- Better for revocation
  1. Hand out capability
  2. Check that you get it back
  3. If so, no other copies in system, i.e. capability revoked
- Applications
  - Stack and return capabilities
    - Skorstengaard, Devriese, Birkedal, WIP
    - single stack, linear stack and return capability

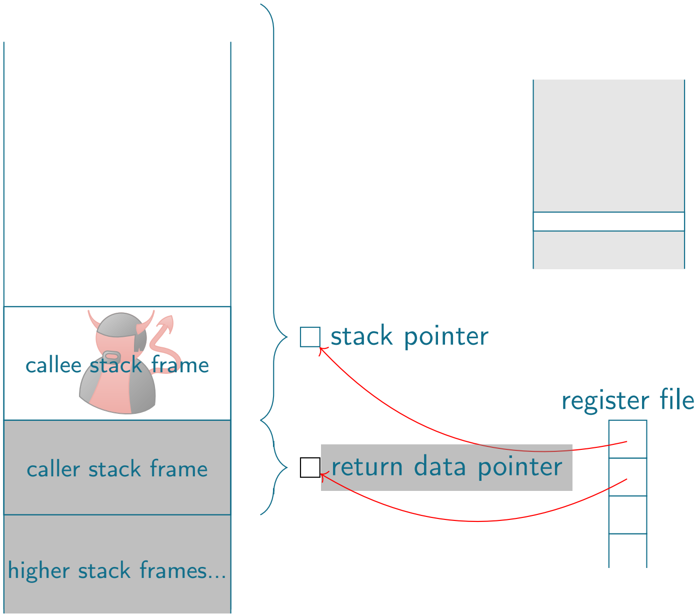
Linear Capabilities Prevent Attack 1



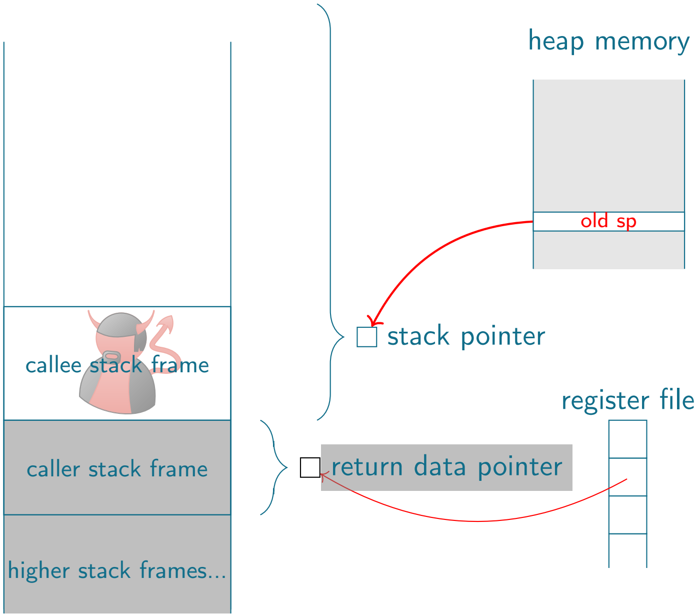
Linear Capabilities Prevent Attack 1



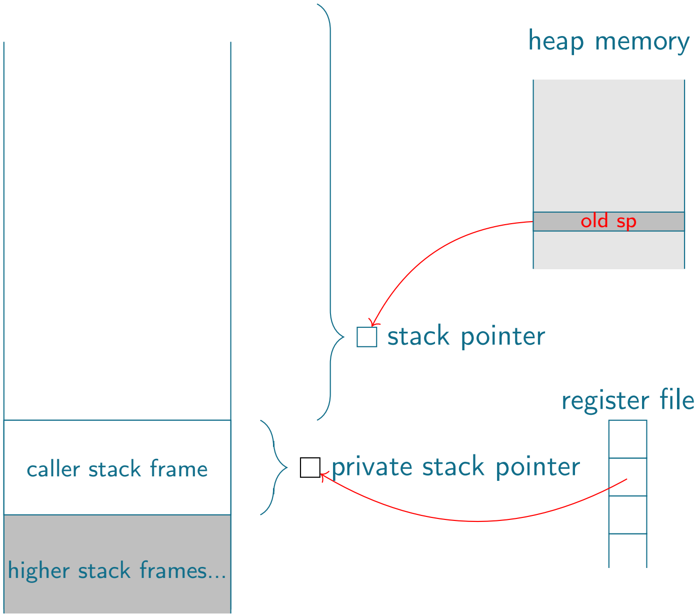
Linear Capabilities Prevent Attack 1



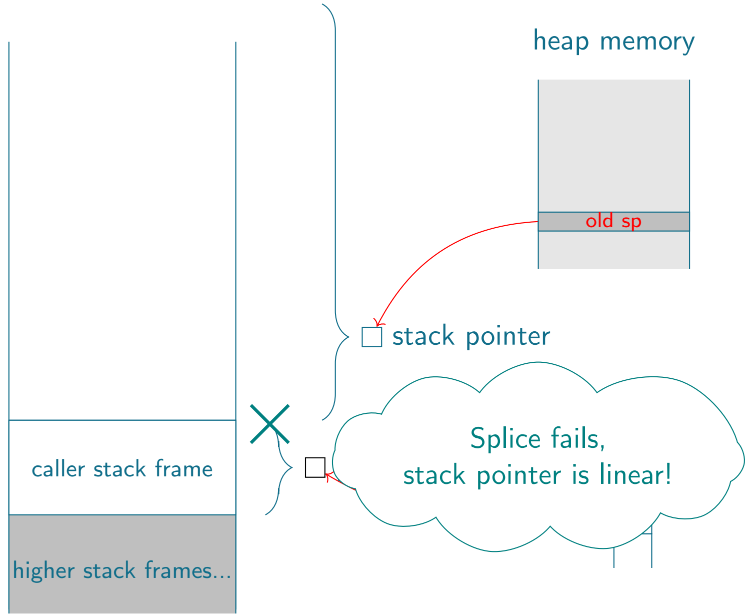
Linear Capabilities Prevent Attack 1



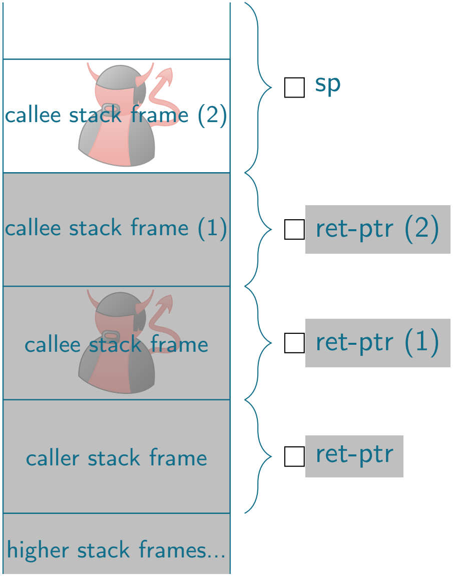
Linear Capabilities Prevent Attack 1



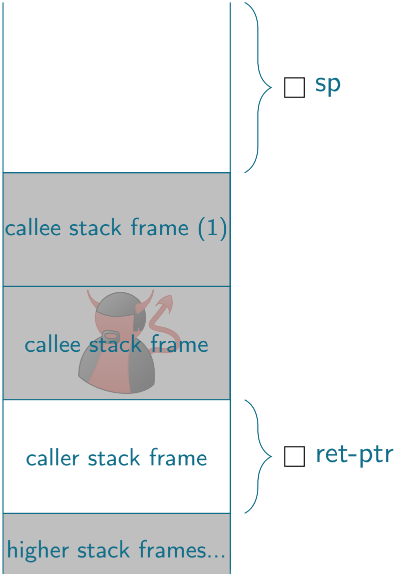
Linear Capabilities Prevent Attack 1



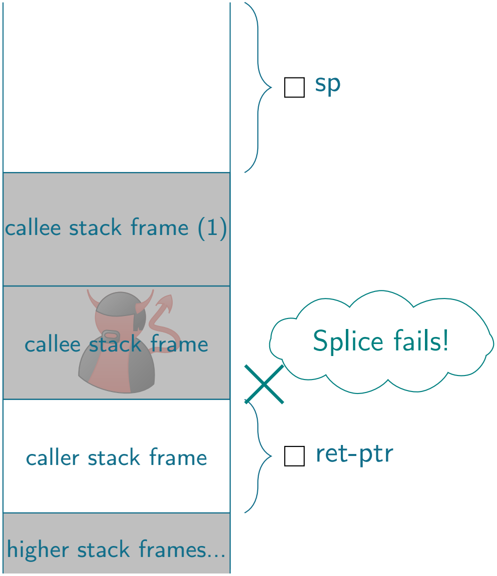
Linear Capabilities Prevent Attack 3



Linear Capabilities Prevent Attack 3



Linear Capabilities Prevent Attack 3



Linear capabilities

- Applications (ctd.)
  - Separation logic
    - Van Strydonck, Devriese, Piessens, WIP
    - Reify heap chunks
    - Reify ghost code
    - Fully abstract compiler
    - Other features useful too: sealing, non-linear caps...
  - Linear types?
- ~~Linear~~ Affine

Outline

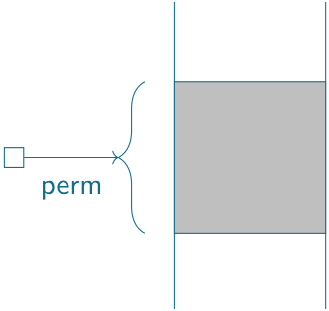
Sealed capabilities

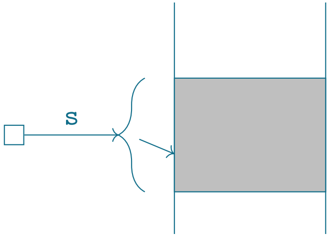
Sealed capabilities

- Seal permission
- Seal/unseal instructions
- (CHERI: also used for ocaps)
- perfect encryption/signatures

Applications:

- Abstract types?
- Typed references?
- Polymorphism? [Sumii and Pierce, 2000] [Devriese et al., 2018]
- Information flow?





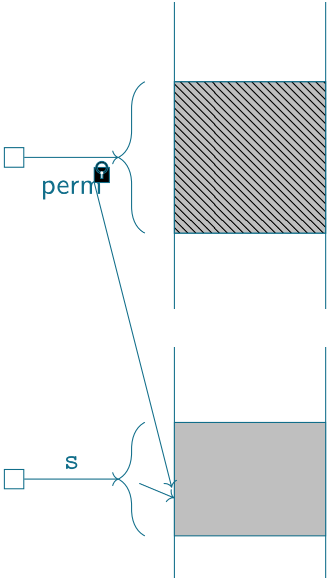
Sealed capabilities

Sealed capabilities

- Seal permission
- Seal/unseal instructions
- (CHERI: also used for ocaps)
- perfect encryption/signatures

Applications:

- Abstract types?
- Typed references?
- Polymorphism? [Sumii and Pierce, 2000] [Devriese et al., 2018]
- Information flow?



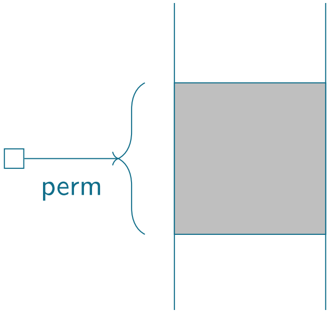
Sealed capabilities

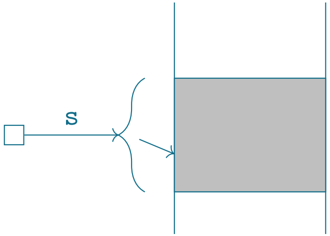
Sealed capabilities

- Seal permission
- Seal/unseal instructions
- (CHERI: also used for ocaps)
- perfect encryption/signatures

Applications:

- Abstract types?
- Typed references?
- Polymorphism? [Sumii and Pierce, 2000] [Devriese et al., 2018]
- Information flow?





Outline

What else do we need?

- Read-after-write capabilities?
  - Allow malloc to not clear
  - Prevent stack frame communication (in single-stack model)
- Features of other systems (e.g., SGX)
  - Attestation?
  - Protection from untrusted OS?
  - Secure I/O?
- … more?

Outline

Conclusion

Capability machines for secure compilation?

- Strengths:
  - Supports higher-order interfaces
  - Many protection domains (e.g. every object, every closure?)
  - Powerful extras: sealing, local, linear capabilities
  - Existing reasoning techniques apply
  - (Relatively) simple at hardware level? (no hardware tables)
- Weaknesses:
  - Only research prototypes (so far)
  - Dynamic checking everywhere
  - Reclaiming component memory? (with HO interfaces)

Backup slide: CheriBSD stack

- CheriBSD stack management
  stack per compartment, trusted stack manager
  - Many compartments ⇒ many stacks
  - Which compartment for an indirect call?