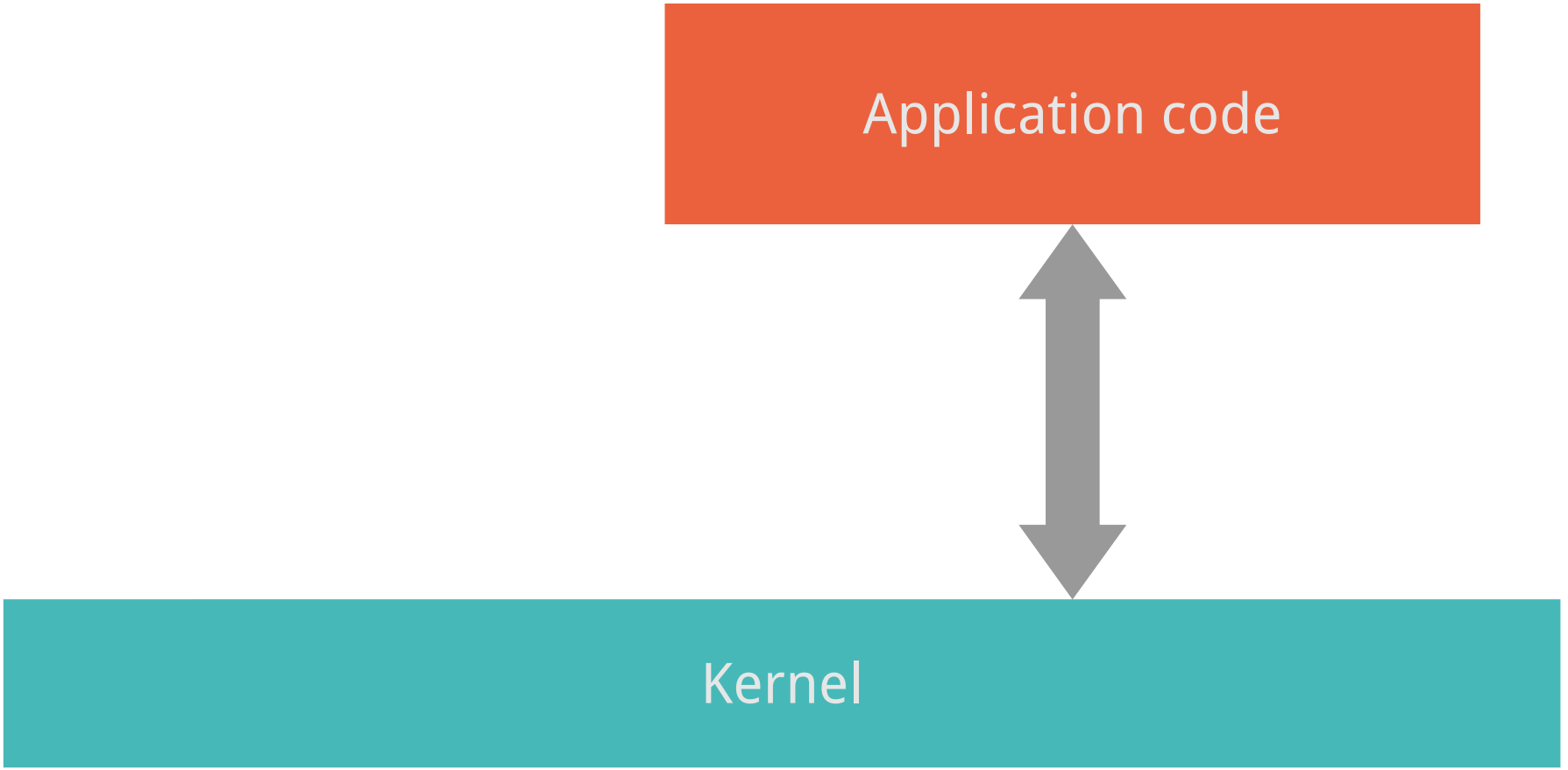
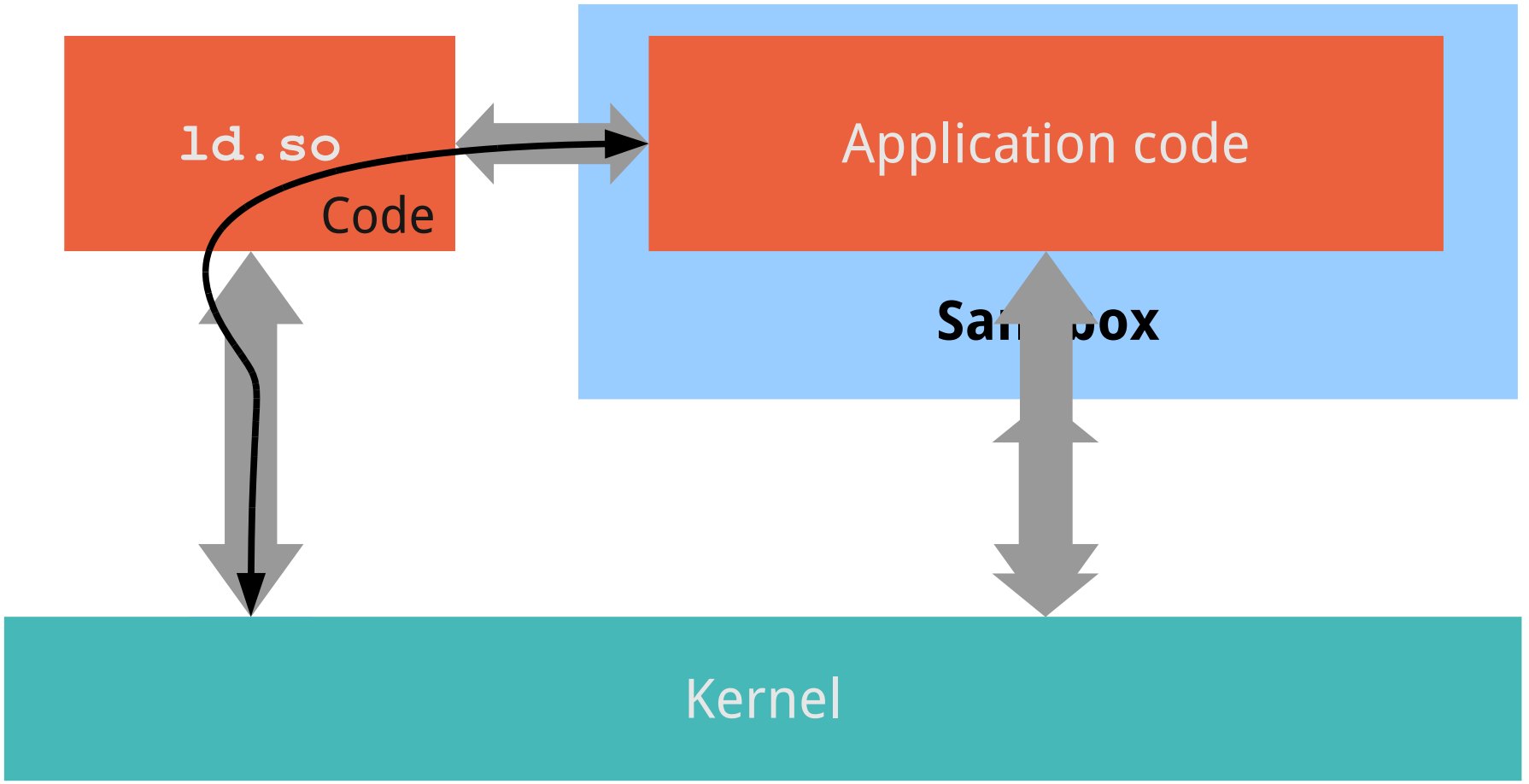
Safe Loading
A Foundation for Secure Execution of Untrusted Programs

Mathias Payer, Tobias Hartmann, Thomas R. Gross
Department of Computer Science
ETH Zurich, Switzerland

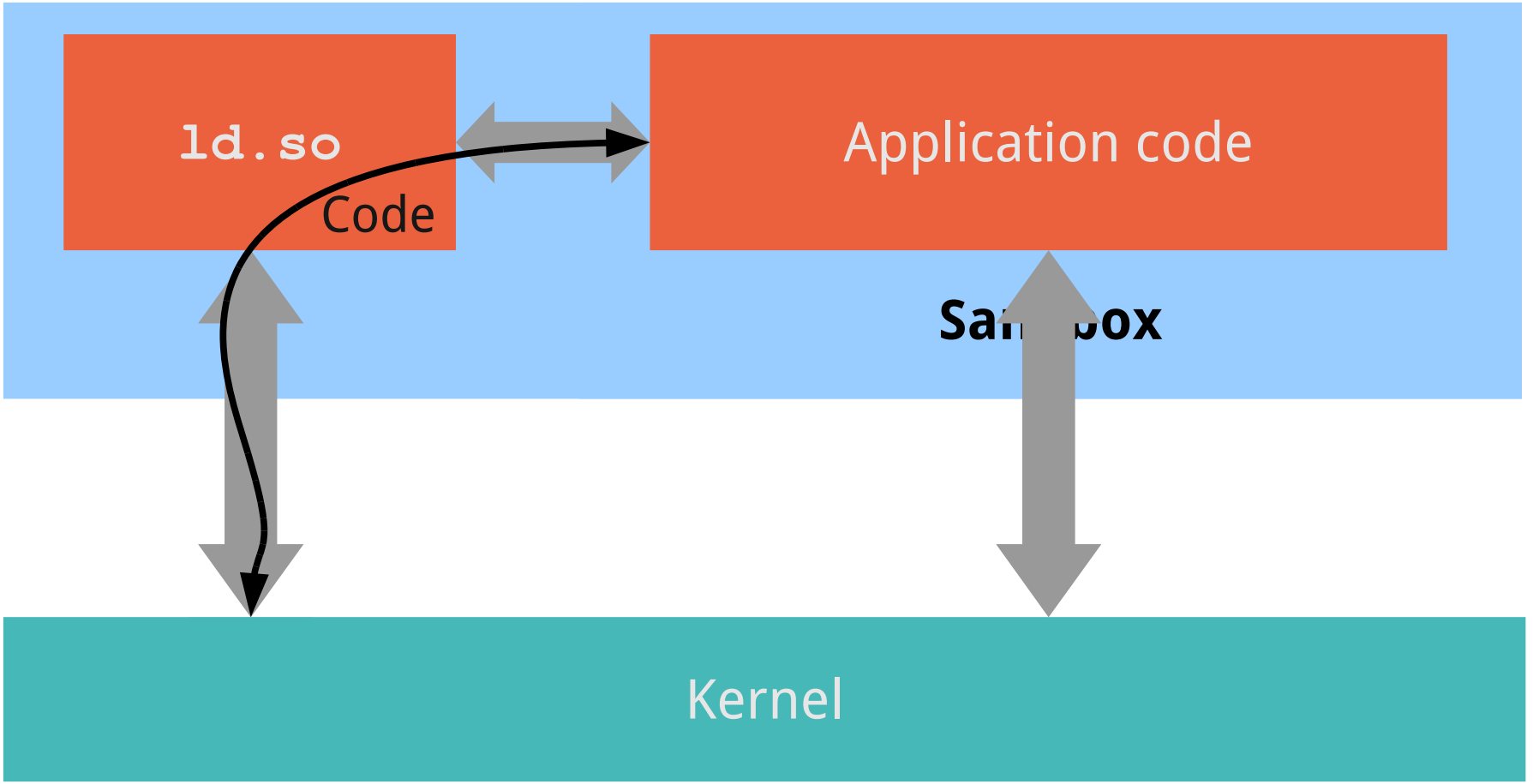
Motivation



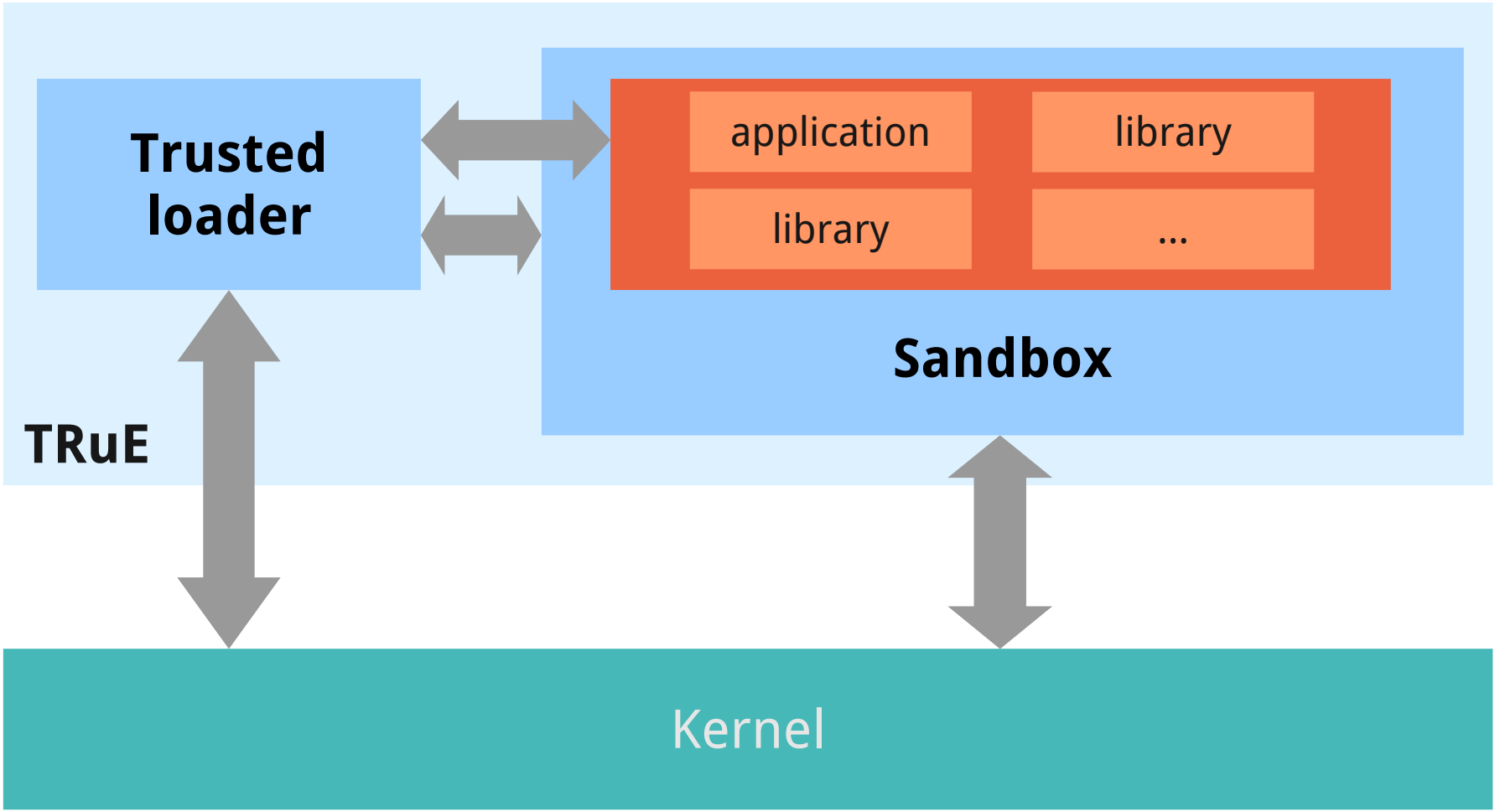
Motivation



Motivation



Trusted RUntime Environment

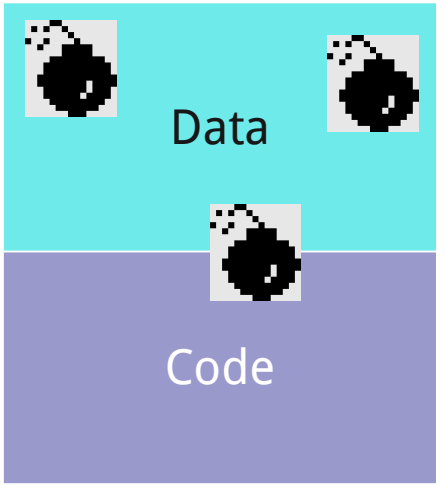


Outline

- Motivation
- **Attack and execution model**
- Trusted Runtime Environment
- Evaluation
- Related work
- Conclusion

Attack constraints

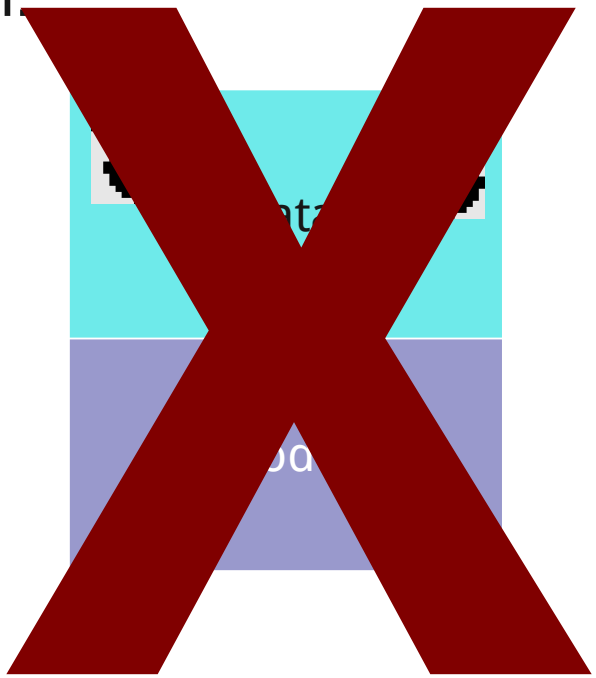
- Attacker tries to ***escalate*** privileges with:
  - Code injection
  - Code reuse (ROP / JOP*)
  - Data attacks



* ROP – Return Oriented Programming: Shacham, CCS'07
  JOP – Jump Oriented Programming: Bletsch et al., ASIACCS'11

Attack constraints

- Attacker tries to ***escalate*** privileges with:
  - Code injection
  - Code reuse (ROP / JOP*)
  - Data attacks
- Application is killed on policy violation



* ROP – Return Oriented Programming: Shacham, CCS'07
JOP – Jump Oriented Programming: Bletsch et al., ASIACCS'11

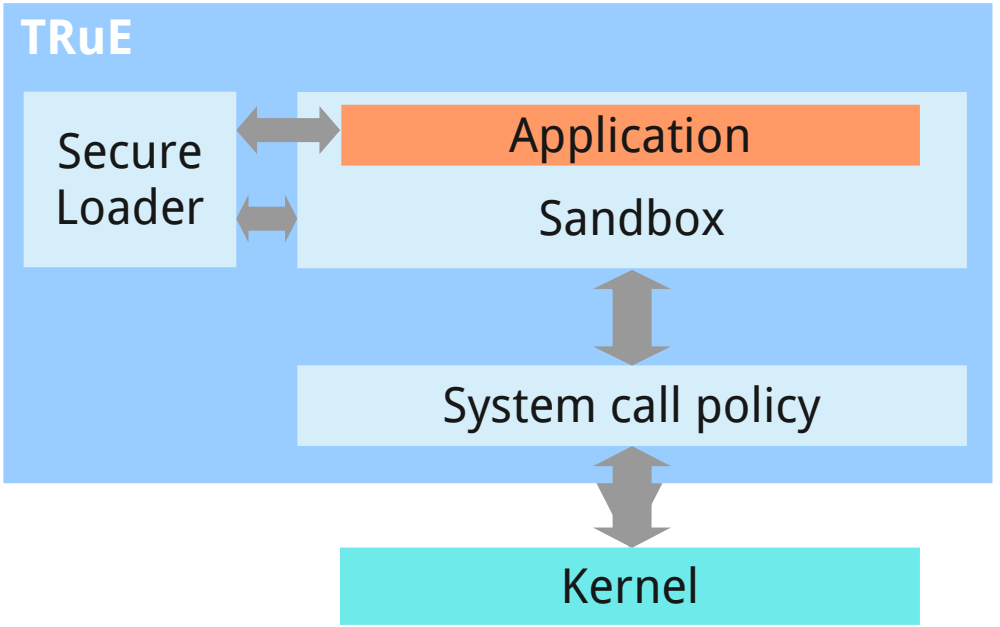
Execution model

- Application is untrusted (not malicious)
  - Symbol table and ELF information used in sandbox
- Secure execution uses
  - Secure loader to bootstrap application
  - Sandbox to protect from any code-based and data-based exploits

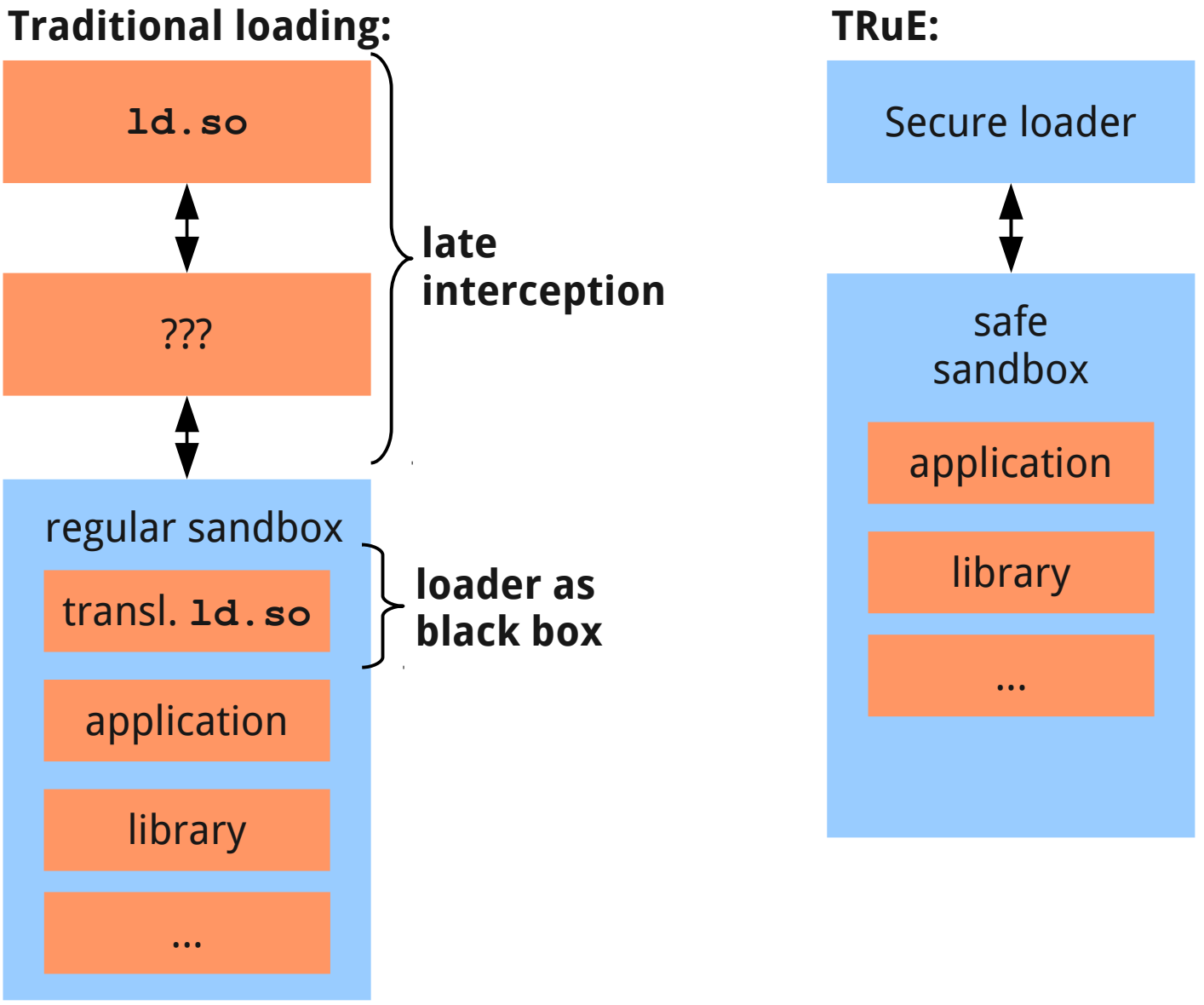
Outline

- Motivation
- Attack model
- **Trusted Runtime Environment**
  - **Security architecture**
  - **Safe loading**
  - **Sandbox & System call policy**
  - **Implementation**
- Evaluation
- Related work
- Conclusion

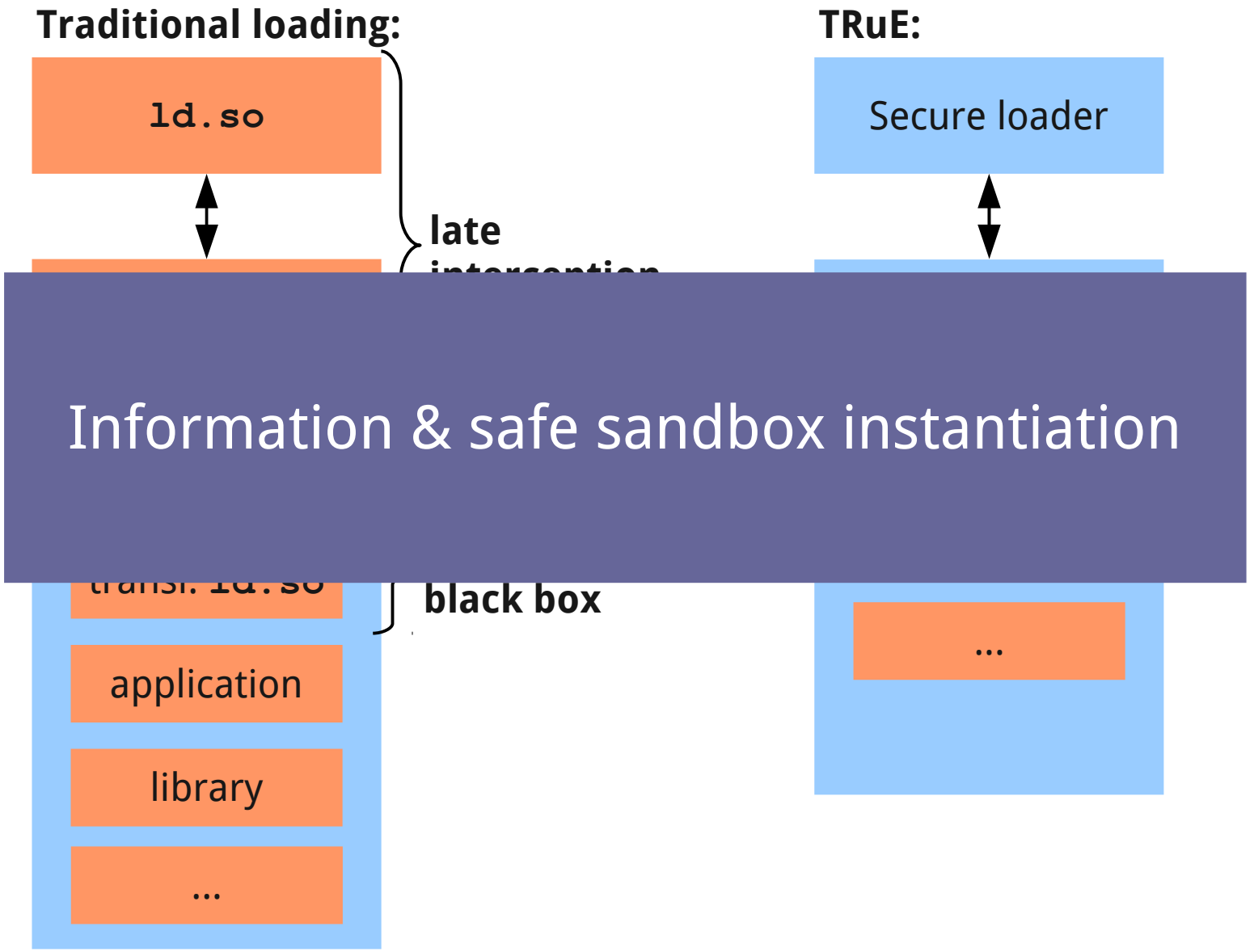
Security architecture



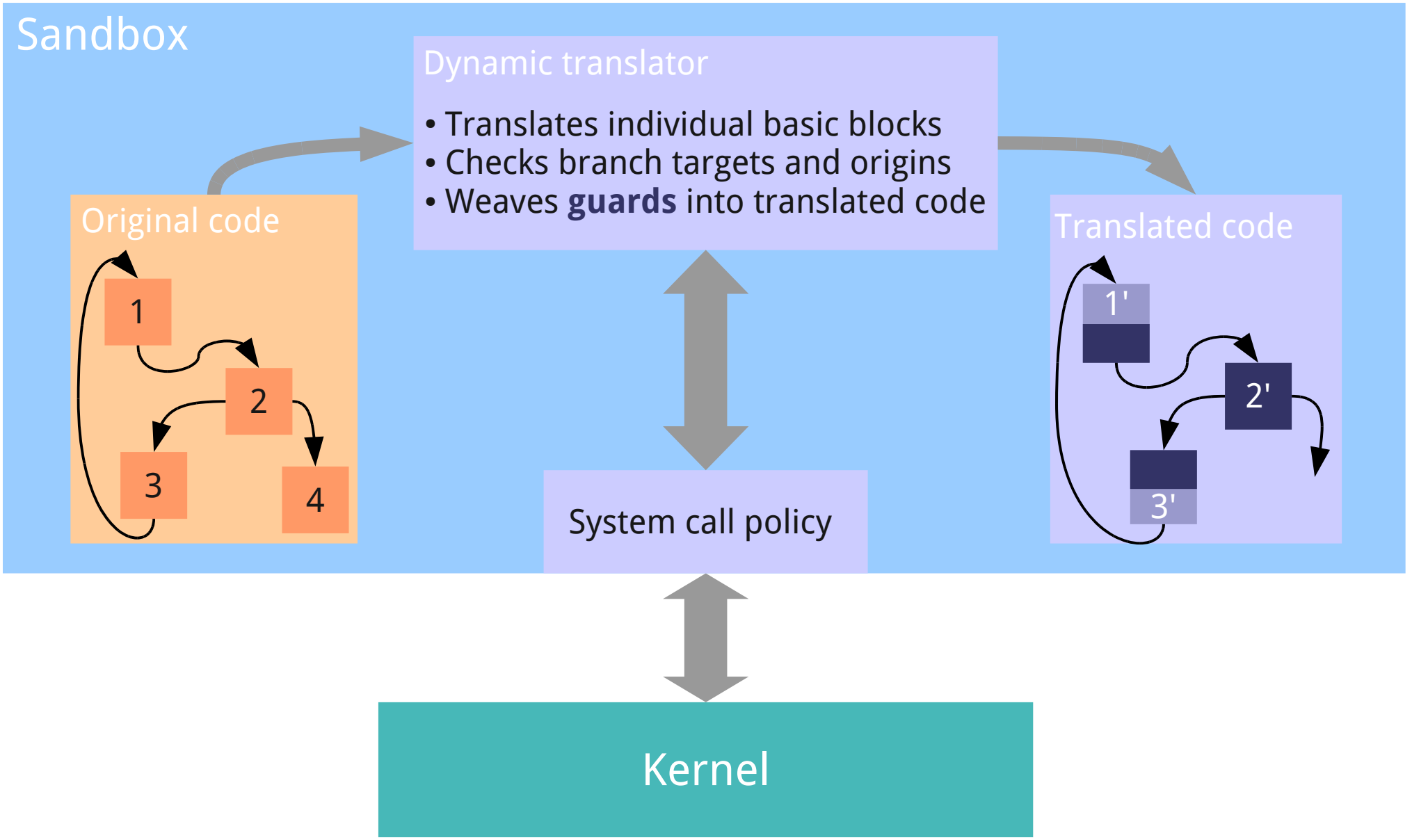
Secure loader



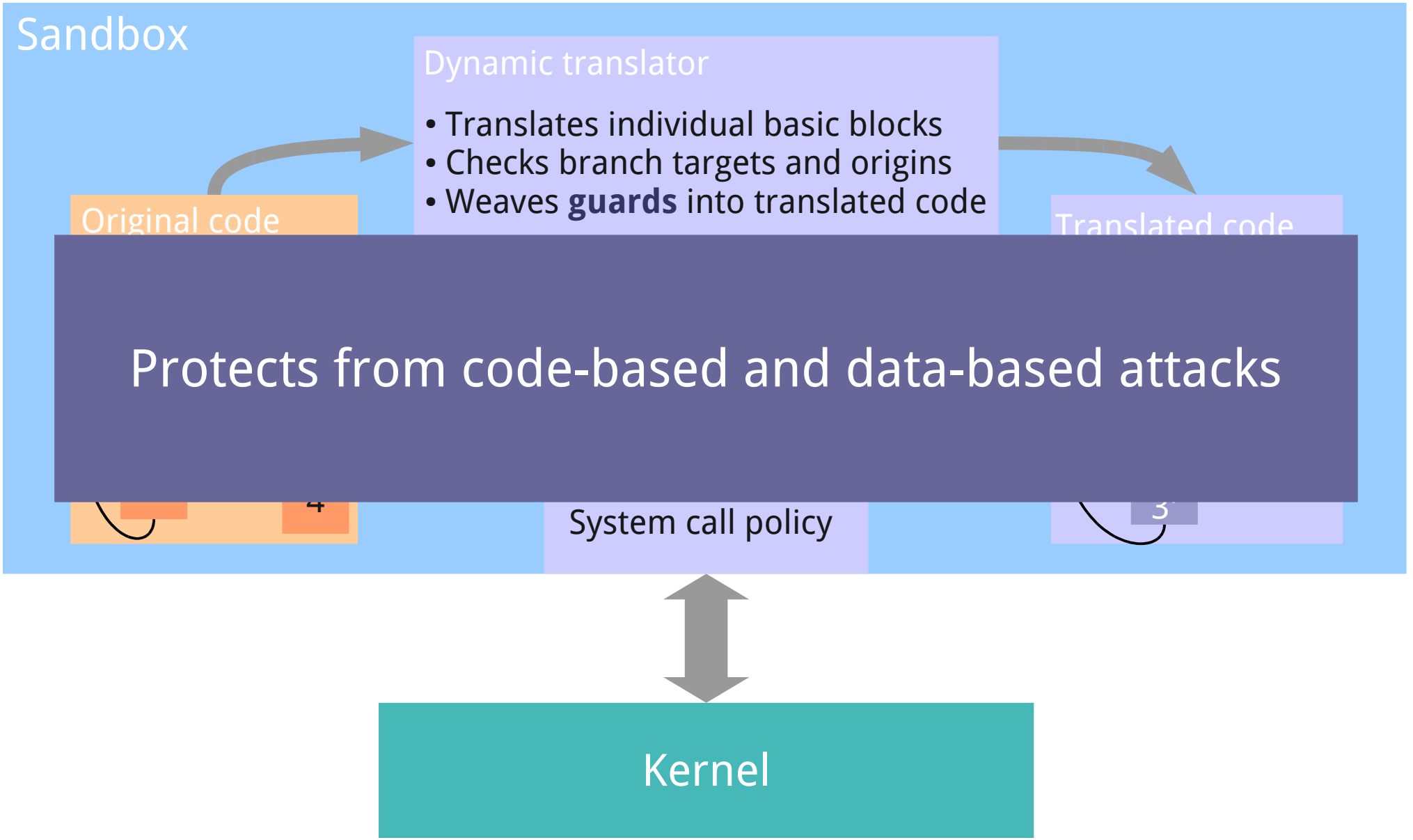
Secure loader



SFI sandbox


Sandbox
Dynamic translator
• Translates individual basic blocks
• Checks branch targets and origins
• Weaves **guards** into translated code
Original code
1
2
3
4
Translated code
1'
2'
3'
System call policy
Kernel

SFI sandbox


Sandbox
Dynamic translator
• Translates individual basic blocks
• Checks branch targets and origins
• Weaves **guards** into translated code
Original code
Translated code
Protects from code-based and data-based attacks
System call policy
Kernel

TRuE: implementation

- Prototype implementation (open source)
  - Focus on IA32 and Linux
  - Concept works for any ISA and operating system

- Small trusted computing base

	Code	Comments
Secure loader	5,400	2,100
Sandbox	15,200*	3,200

*4,900 LOC for the translation tables

Outline

- Motivation
- Attack model
- Trusted Runtime Environment
- **Evaluation**
  - **Security discussion**
  - **SPEC CPU2006 performance**
- Related work
- Conclusion

Security discussion

- Two execution domains
  - Privileged sandbox domain
  - Unprivileged application domain (traps into sandbox)
- Sandbox ensures code integrity
  - Protection from code-injection and return oriented programming
  - Policy protects from jump oriented programming and data attacks
- Secure loader enables safe program instantiation
  - Low complexity (bare bone functionality)
  - API for requests from the application

Security discussion

- Two execution domains
  - Privileged sandbox domain
  - Unprivileged application domain (traps into sandbox)
- Sand
  - Pr ng
  - Pc acks
- Secure loader enables safe program instantiation
  - Low complexity (bare bone functionality)
  - API for requests from the application

Protects unmodified, binary applications from attacks

SPEC CPU 2006 performance

- Benchmarks execute with well-defined policy
  - On Ubuntu Jaunty
  - Intel Xeon E5520 CPU at 2.27GHz
  - GCC version 4.3.3
- Three configurations:
  - native
  - Secure loader (without sandbox)
  - TRuE (secure loader plus sandbox)

SPEC CPU 2006 performance

Benchmark	Secure Loading	TRuE
400.perlbench	-0.3%	85%
401.bzip2	-0.1%	4.9%
429.mcf	-0.1%	0.5%
464.h264ref	-0.3%	41%
433.milc	0.1%	3.7%
Average*	**-0.1%**	**15%**

* Average is calculated over all 28 SPEC CPU2006 benchmarks

SPEC CPU 2006 performance

Benchmark	Secure Loading	TRuE
Average*	-0.1%	15%

Low performance impact

* Average is calculated over all 28 SPEC CPU2006 benchmarks

Outline

- Motivation
- Attack model
- Trusted Runtime Environment
- Evaluation
- **Related work**
- Conclusion

Related work

- System call interposition (Janus, AppArmor)
  - Only system calls checked, code is unchecked
- Software-based fault isolation (Libdetox, Vx32, Strata)
  - Only a sandbox is not enough, additional guards and system call authorization needed, no loader information
- Static binary translation (Google's NaCL, PittSFIeld)
  - Limits the ISA, static, special compilers needed
- Full system translation (VMWare, QEMU, Xen)
  - Management overhead, data sharing problem

Outline

- Motivation
- Attack model
- Trusted Runtime Environment
- Evaluation
- Related work
- **Conclusion**

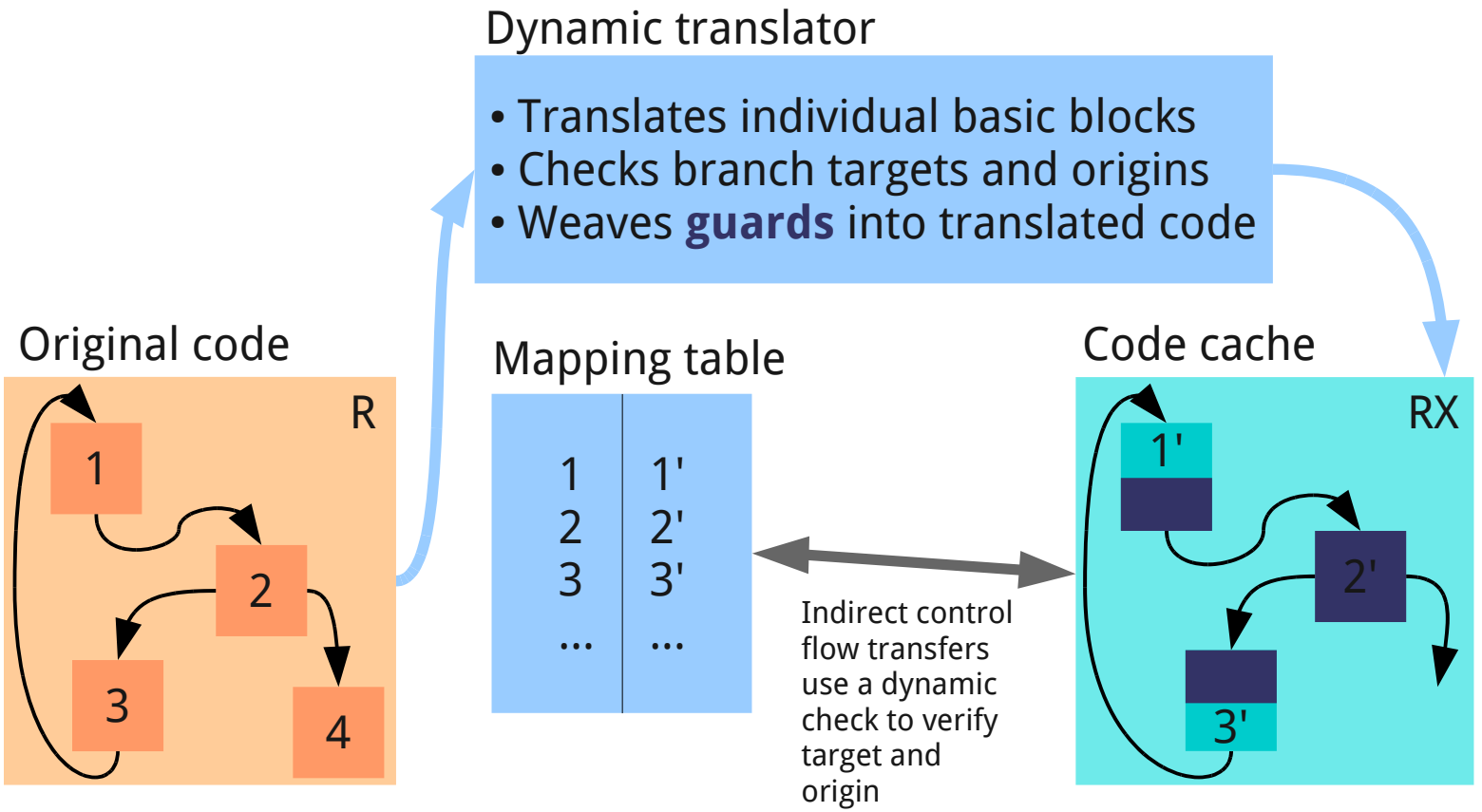
Conclusion

- TRuE protects from code-based and data-based exploits
  - **Secure loader** extracts information
  - **Sandbox** protects from code-based and data-based exploits
- Trusted secure loader increases security
  - Application needs no privileges to map code executable
  - Knowledge of program structure enables new guards
- TRuE protects unmodified applications in user-space

Questions?

http://nebelwelt.net/projects/TRuE/

Software based fault isolation



Implementation alternatives

- Static binary translation
  - No second protected domain
  - No dynamic library/module support
  - Restricted ISA
- Regular loader, hidden sandbox
  - Sandbox hidden by modifying loader data-structures
  - Loader treated as black-box

Malicious applications

- No information about internal control flow
  - Coarse-grained protection at system call level
- Application can use CPU time (inside the app)
  - System call policy protects from malicious system calls