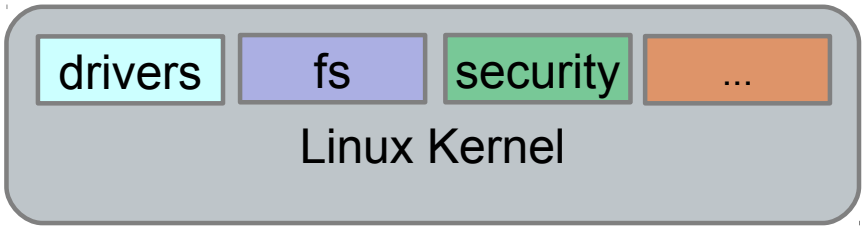
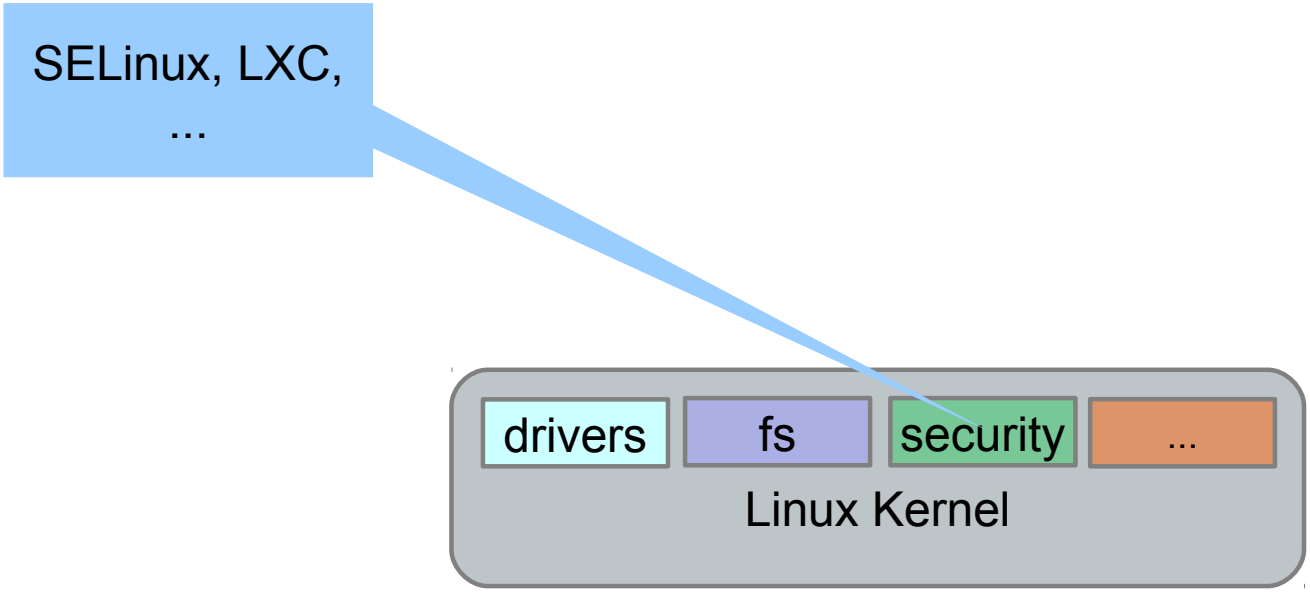
Attack Surface Metrics and Automated Compile-Time Kernel Tailoring

Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Wolfgang Schröder-Preikschat, Daniel Lohmann and Rüdiger Kapitza

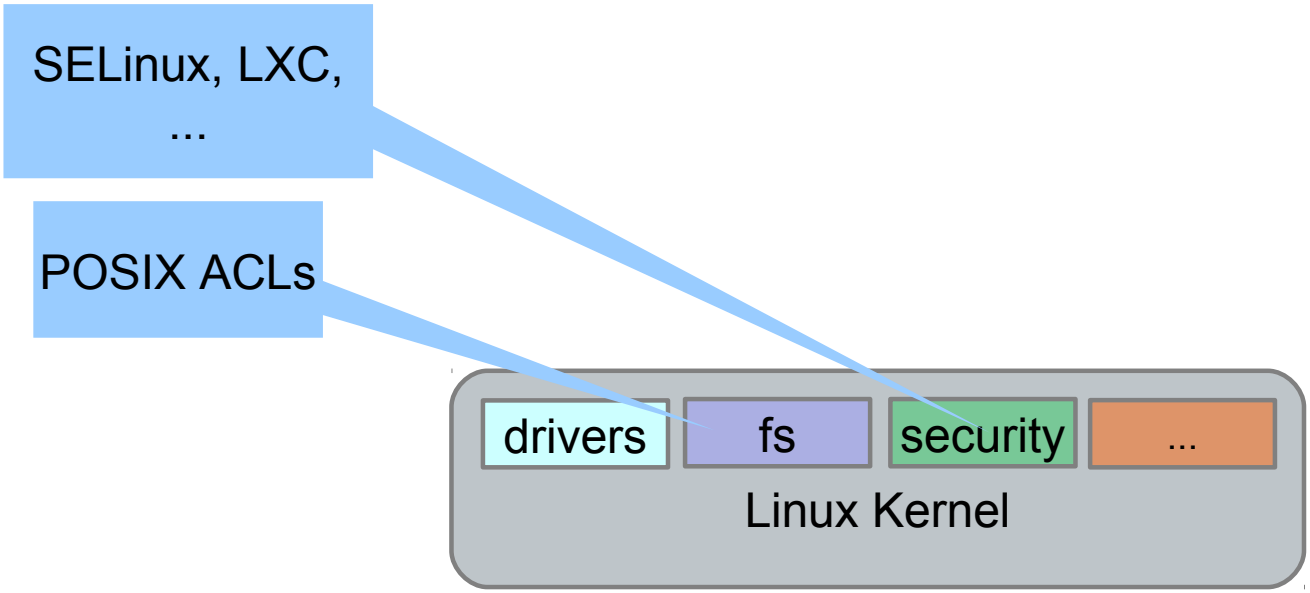
Why (still) care about kernel security?



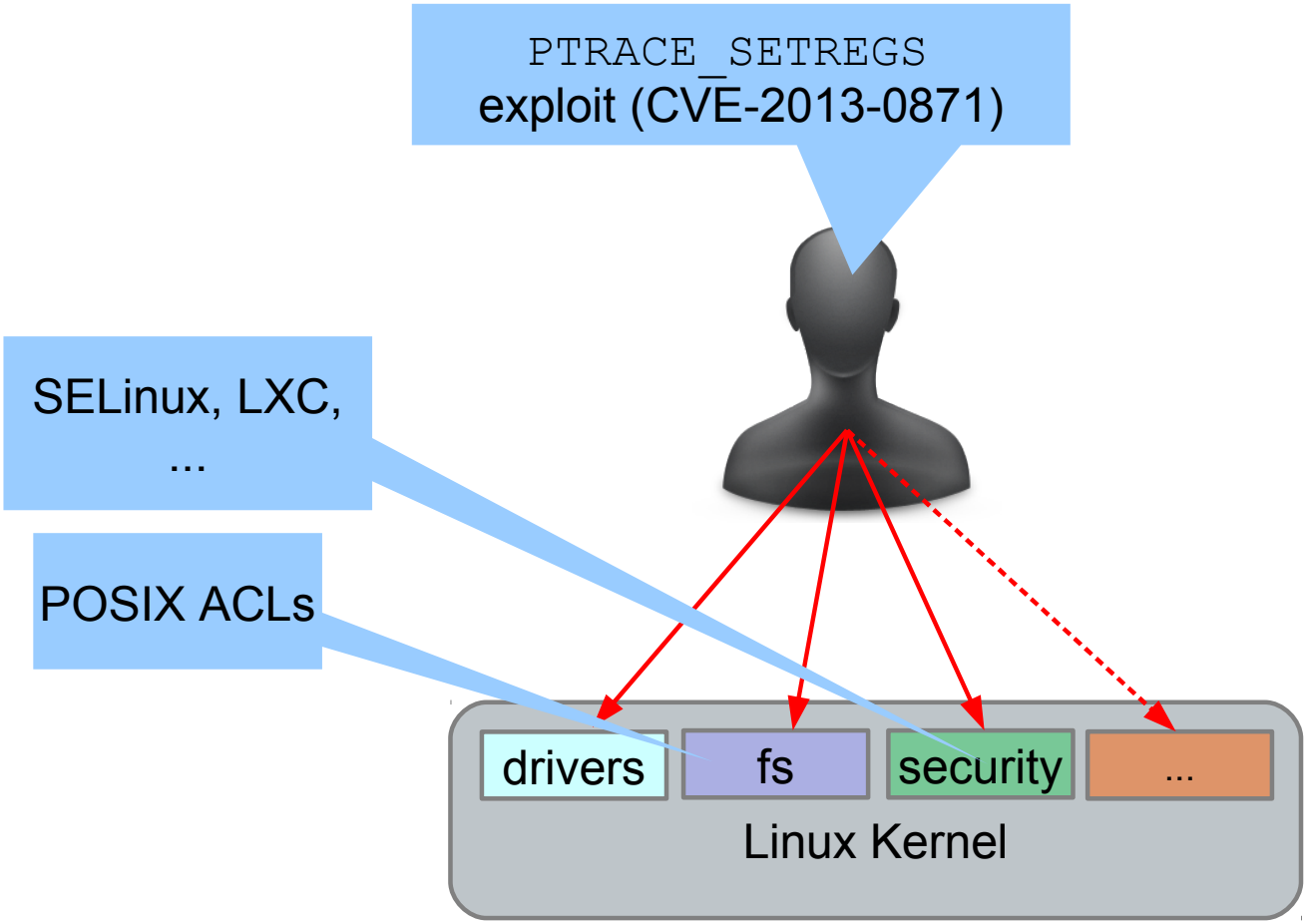
Why (still) care about kernel security?



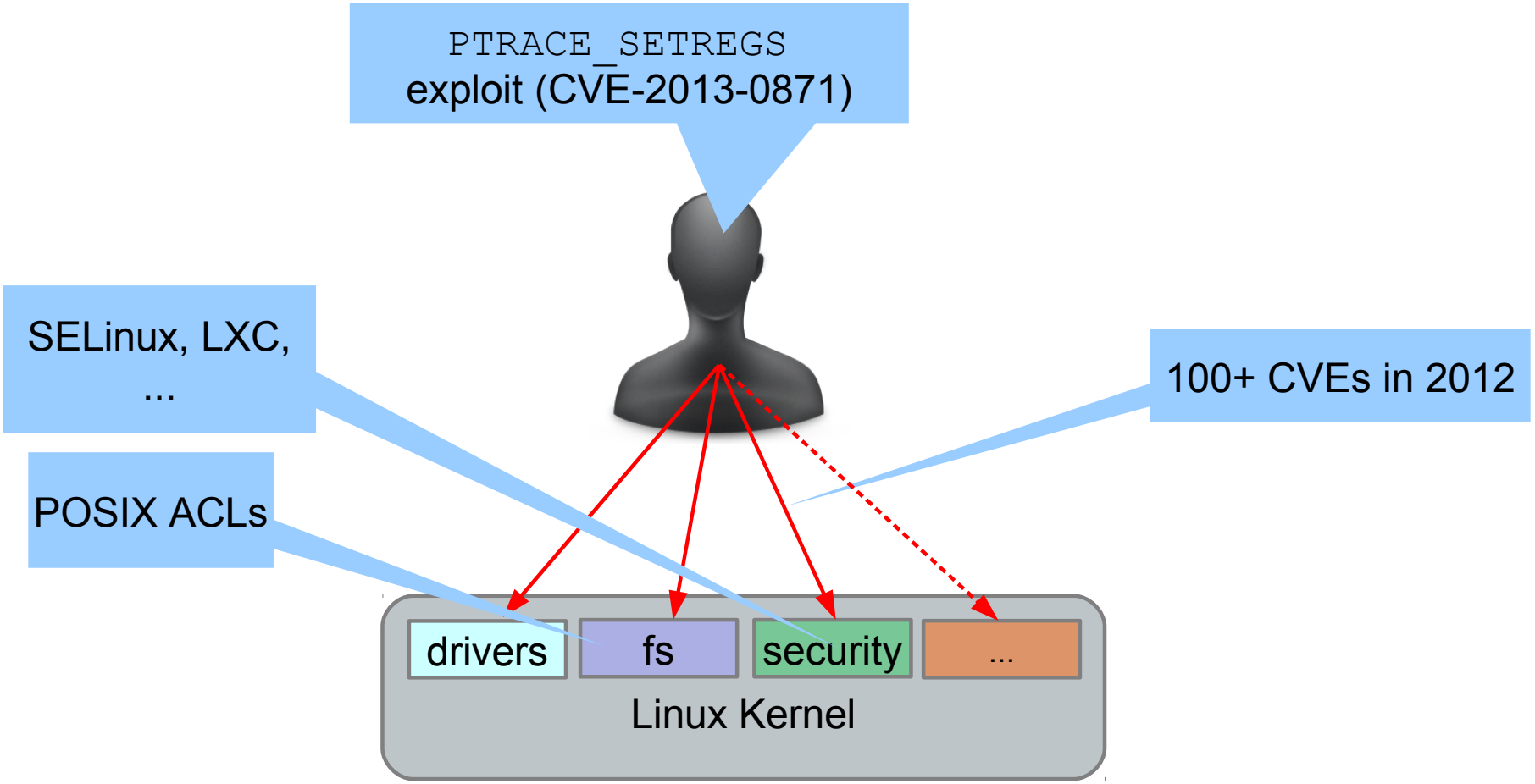
Why (still) care about kernel security?



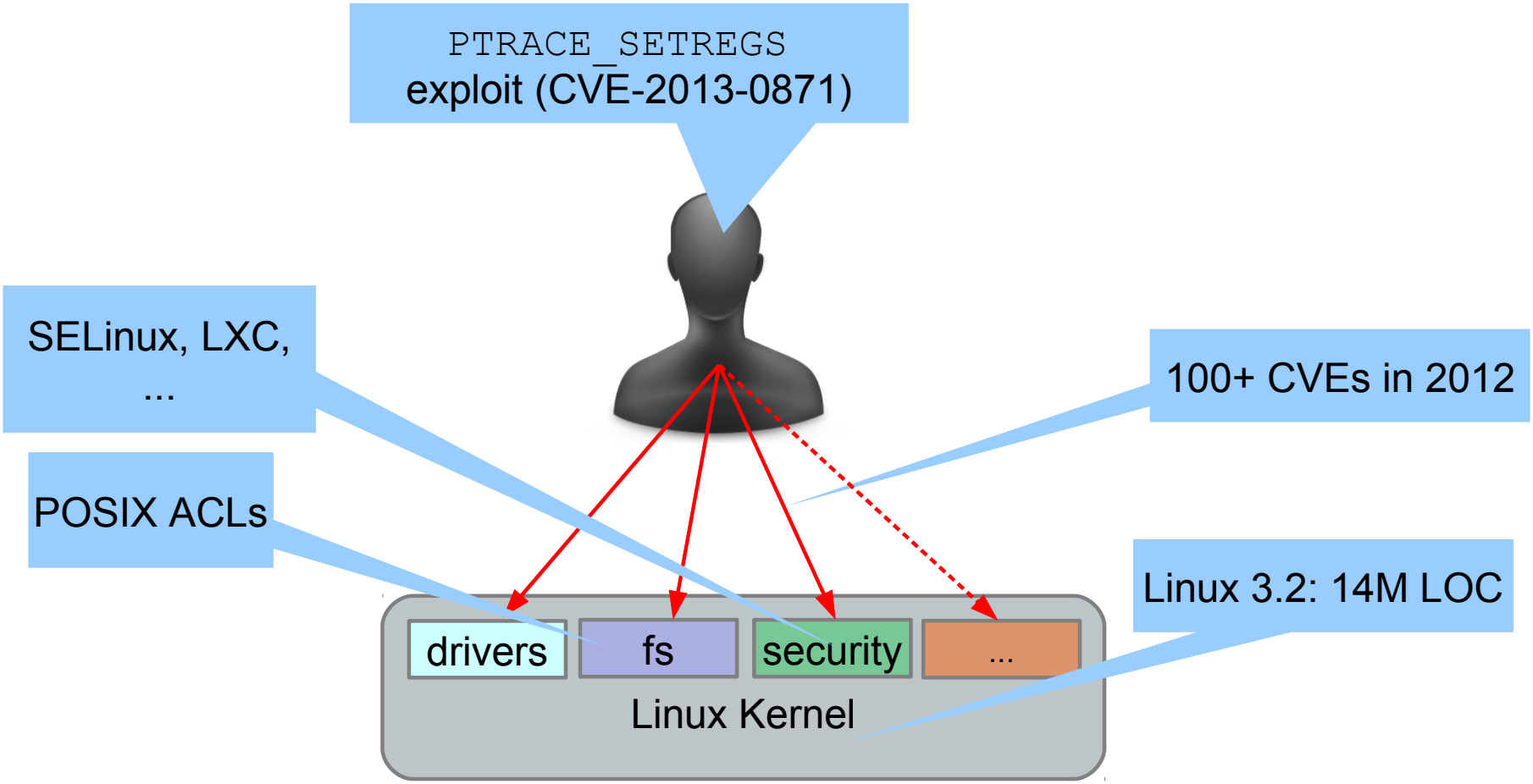
Why (still) care about kernel security?



Why (still) care about kernel security?



Why (still) care about kernel security?



Making the kernel smaller

~ 5000 features
(ubuntu 12.04)

Making the kernel smaller

RDS
CVE-2010-3904

~ 5000 features
(ubuntu 12.04)

Making the kernel smaller

~ 5000 features
(ubuntu 12.04)

~ 500 features
(realistic use case)

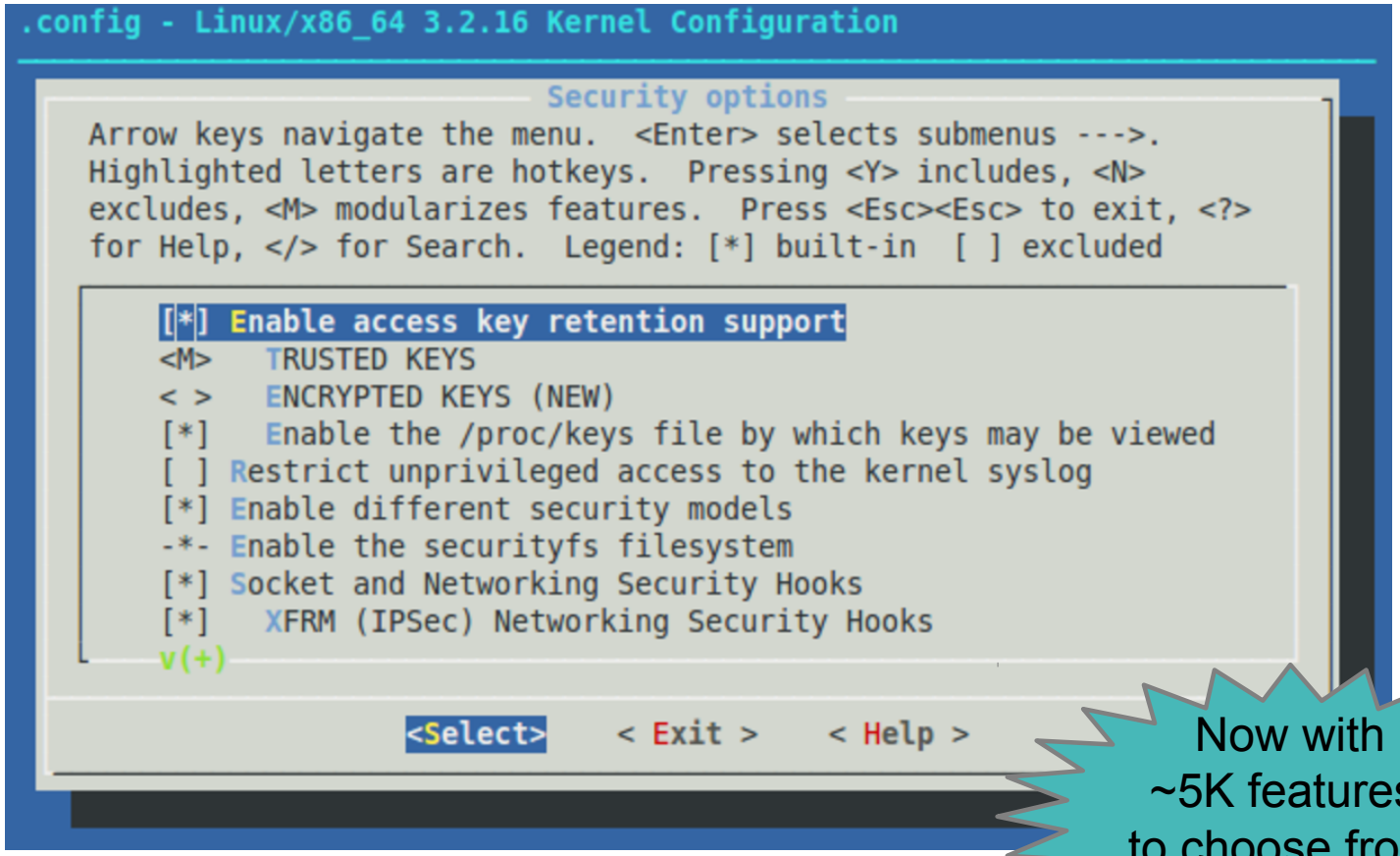
Making the kernel smaller

~ 5000 features
(ubuntu 12.04)

~ 500 features
(realistic use case)

Remove unnecessary features from the kernel by leveraging built-in configurability

Make (menuconfig) your way to a smaller kernel



Don't take my word for it

[RFC] Simplifying kernel configuration for distro issues

87 messages

Linus Torvalds <torvalds@linux-foundation.org> Fri, Jul 13, 2012 at 10:37 PM
To: Dave Jones <davej@redhat.com>, Greg Kroah-Hartman <greg@kroah.com>, Ubuntu Kernel Team <kernel-team@lists.ubuntu.com>, Debian Kernel Team <debian-kernel@lists.debian.org>, OpenSUSE Kernel Team <opensuse-kernel@opensuse.org>
Cc: Linux Kernel Mailing List <linux-kernel@vger.kernel.org>

So this has long been one of my pet configuration peeves: as a user I am perfectly happy answering the questions about what kinds of hardware I want the kernel to support (I kind of know that), but many of the "support infrastructure" questions are very opaque, and I have no idea which of the them any particular distribution actually depends on.

And it tends to change over time. For example, F14 (iirc) started using TMPFS and TMPFS_POSIX_ACL/XATTR for /dev. And starting in F16, the initrd setup requires DEVTMPFS and DEVTMPFS_MOUNT. There's been several times when I started with my old minimal config, and the resulting kernel would boot, but something wouldn't quite work right, and it can be very subtle indeed.

Similarly, the distro ends up having very particular requirements for exactly *which* security models it uses and needs, and they tend to change over time. And now with systemd, CGROUPS suddenly aren't just esoteric things that no normal person would want to use, but are used for basic infrastructure. And I remember being surprised by OpenSUSE suddenly needing the RAW table support for netfilter, because it had a NOTRACK rule or something.

Don't take my word for it

[RFC] Simplifying kernel configuration for distro issues
87 messages

Linus Torvalds <torvalds@linux-foundation.org> Fri, Jul 13, 2012 at 10:37 PM
To: Dave Jones <davej@redhat.com>, Greg Kroah-Hartman <greg@kroah.com>, Ubuntu Kernel Team <kernel-team@lists.ubuntu.com>, Debian Kernel Team <debian-kernel@lists.debian.org>, OpenSUSE Kernel Team <opensuse-kernel@opensuse.org>
Cc: Linux Kernel Mailing List <linux-kernel@vger.kernel.org>

"many of the support infrastructure questions are very opaque, and I have no idea which of them any particular distribution actually depends on."

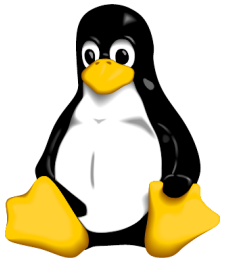
And it tends to change over time. For example, F14 (iirc) started using TMPFS and TMPFS_POSIX_ACL/XATTR for /dev. And starting in F16, the initrd setup requires DEVTMPFS and DEVTMPFS_MOUNT. There's been several times when I started with my old minimal config, and the resulting kernel would boot, but something wouldn't quite work right, and it can be very subtle indeed.

Similarly, the distro ends up having very particular requirements for exactly *which* security models it uses and needs, and they tend to change over time. And now with systemd, CGROUPS suddenly aren't just esoteric things that no normal person would want to use, but are used for basic infrastructure. And I remember being surprised by OpenSUSE suddenly needing the RAW table support for netfilter, because it had a NOTRACK rule or something.

Automatic Kernel-Configuration Tailoring

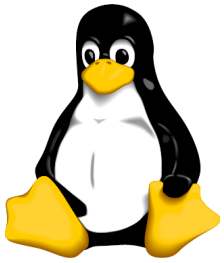
Automatic Kernel-Configuration Tailoring

Distribution kernel
and use case



Automatic Kernel-Configuration Tailoring

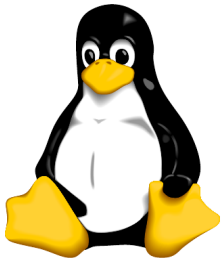
Distribution kernel
and use case



Tailored kernel

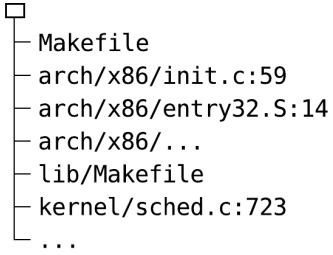
Automatic Kernel-Configuration Tailoring

Distribution kernel
and use case



Tailored kernel

run workload
and collect **trace**



correlate to
source line locations
and **#ifdefs**

```
B00 <-> CONFIG_X86
&&
B1 <-> CONFIG_NUMA
&&
B2 <-> ! B1
&&
...
```

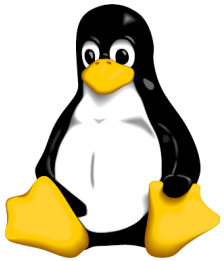
correlate to **features**
and take into account
feature dependencies

```
CONFIG_X86=y
CONFIG_NUMA=y
CONFIG_SCSI=m
...
...
```

solve formula
and derive a **kernel configuration**

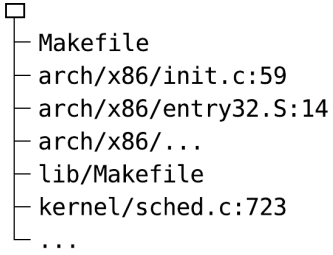
Automatic Kernel-Configuration Tailoring

Distribution kernel
and use case



Tailored kernel

run workload
and collect **trace**



correlate to
source line locations
and **#ifdefs**

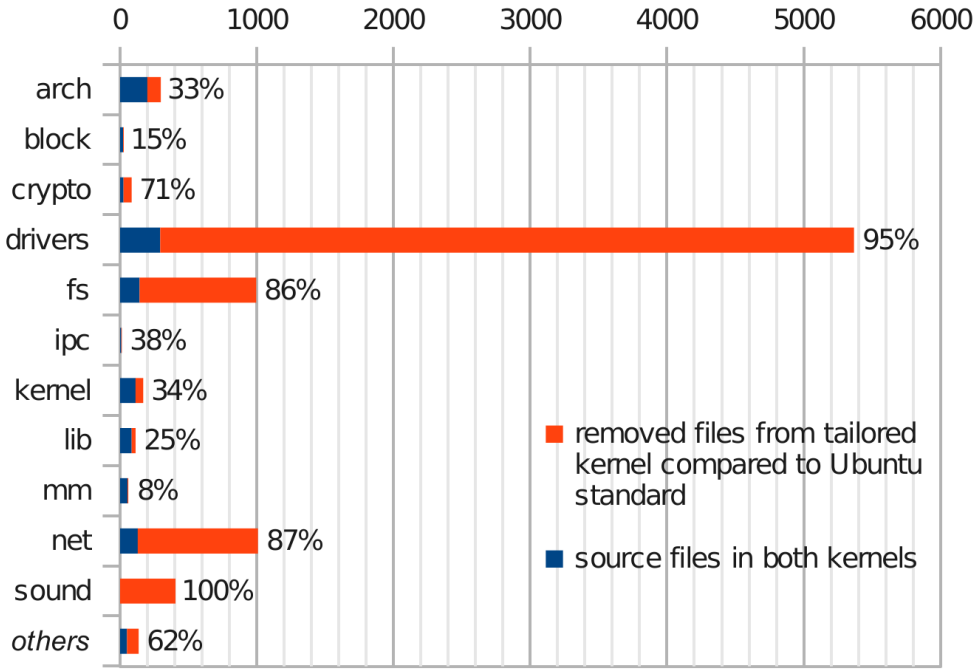
```
B00 <-> CONFIG_X86
&&
B1 <-> CONFIG_NUMA
&&
B2 <-> ! B1
&&
...
```

correlate to **features**
and take into account
feature dependencies

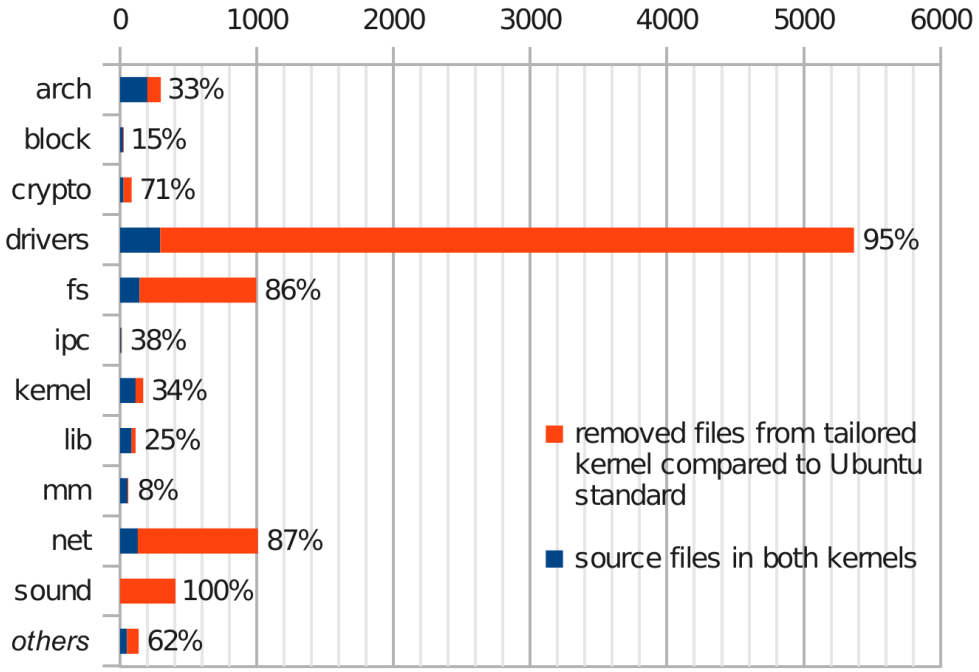
```
CONFIG_X86=y
CONFIG_NUMA=y
CONFIG_SCSI=m
...
...
```

solve formula
and derive a **kernel configuration**

Resulting kernel

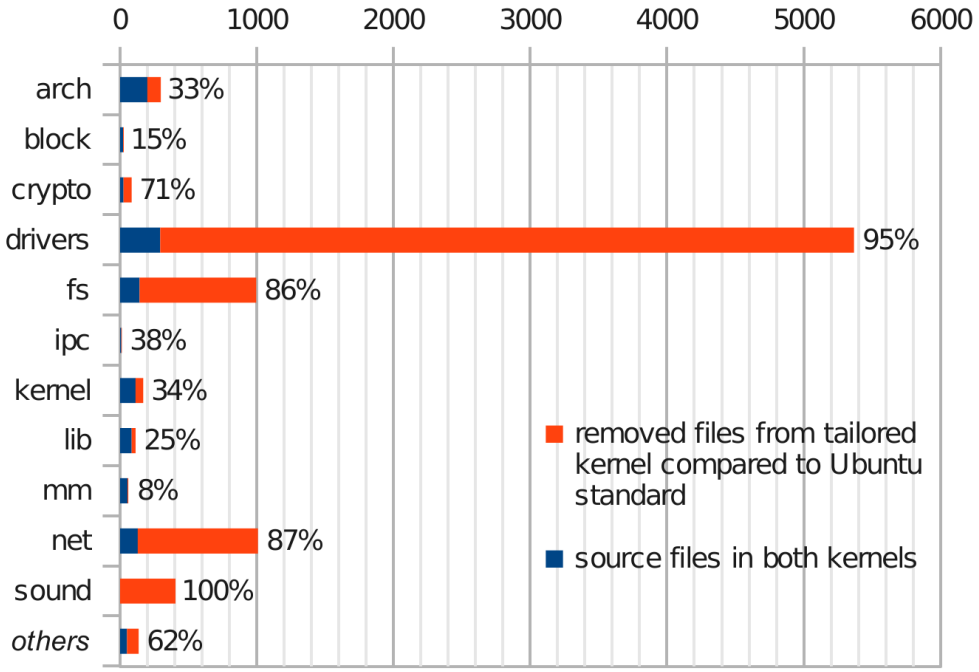


Resulting kernel



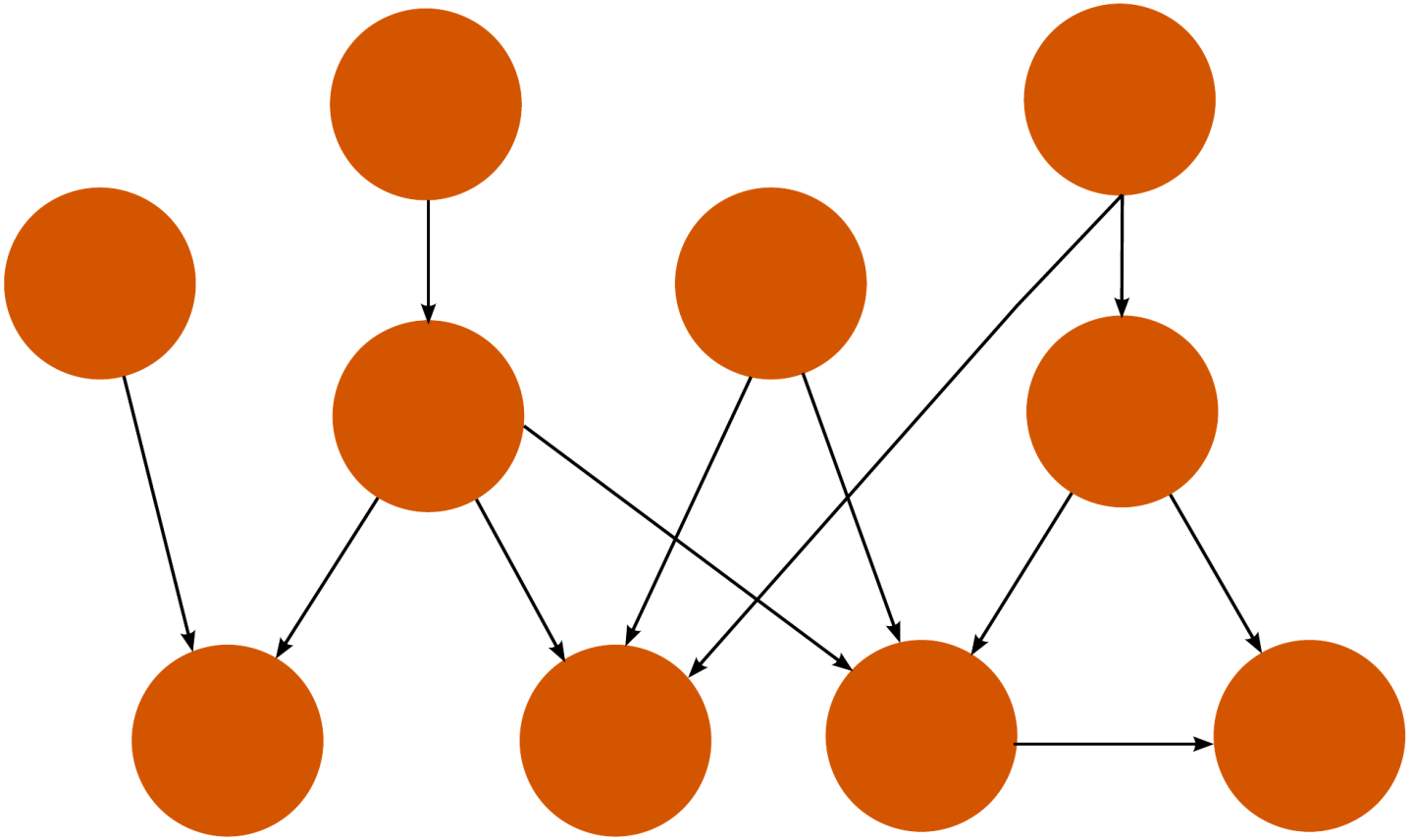
- Is there a **performance** difference?
- Is **tracing** sufficient?
- How much **attack surface reduction**?

Resulting kernel

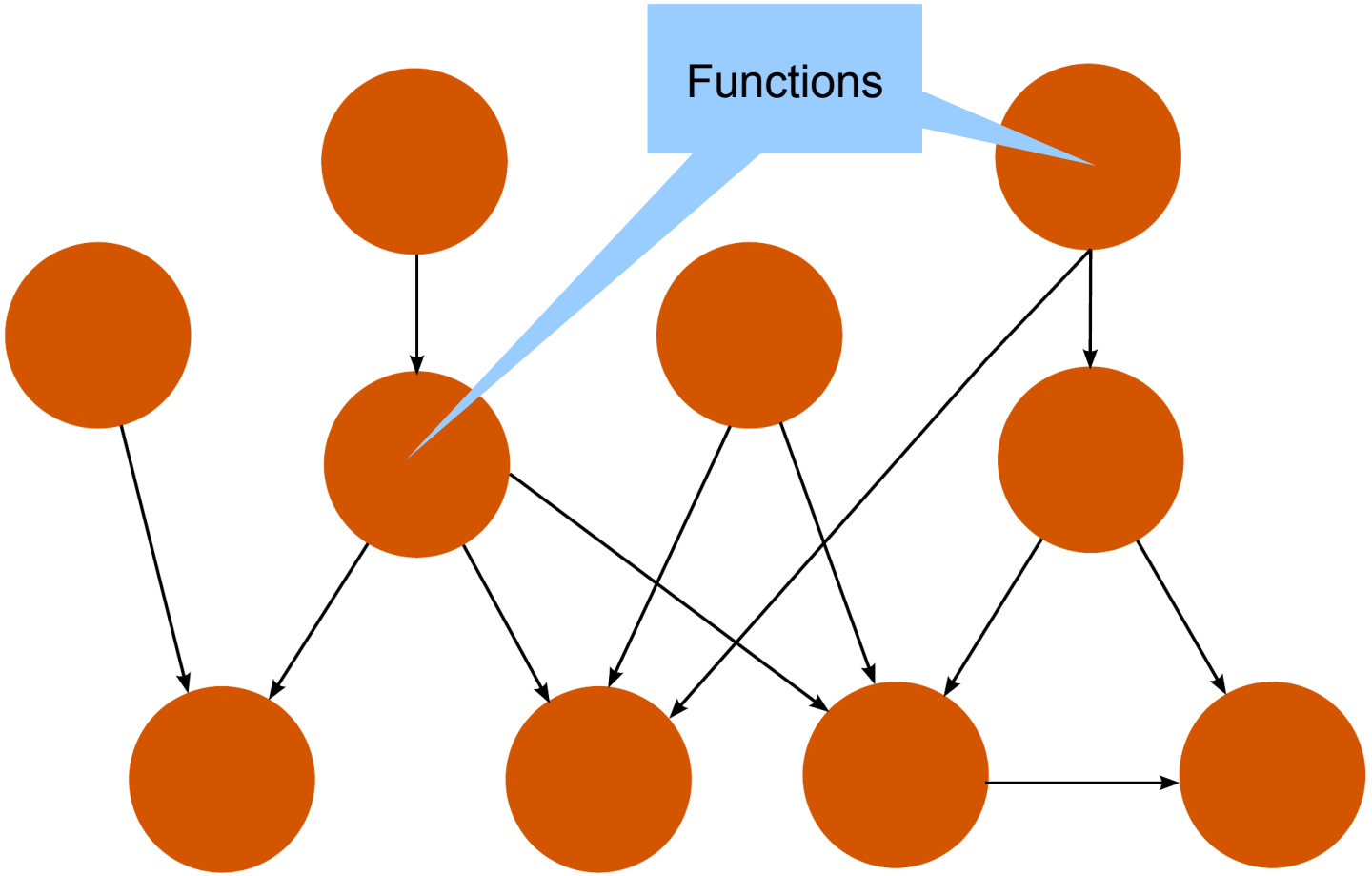


- Is there a **performance** difference?
- Is **tracing** sufficient?
- How much **attack surface reduction**?

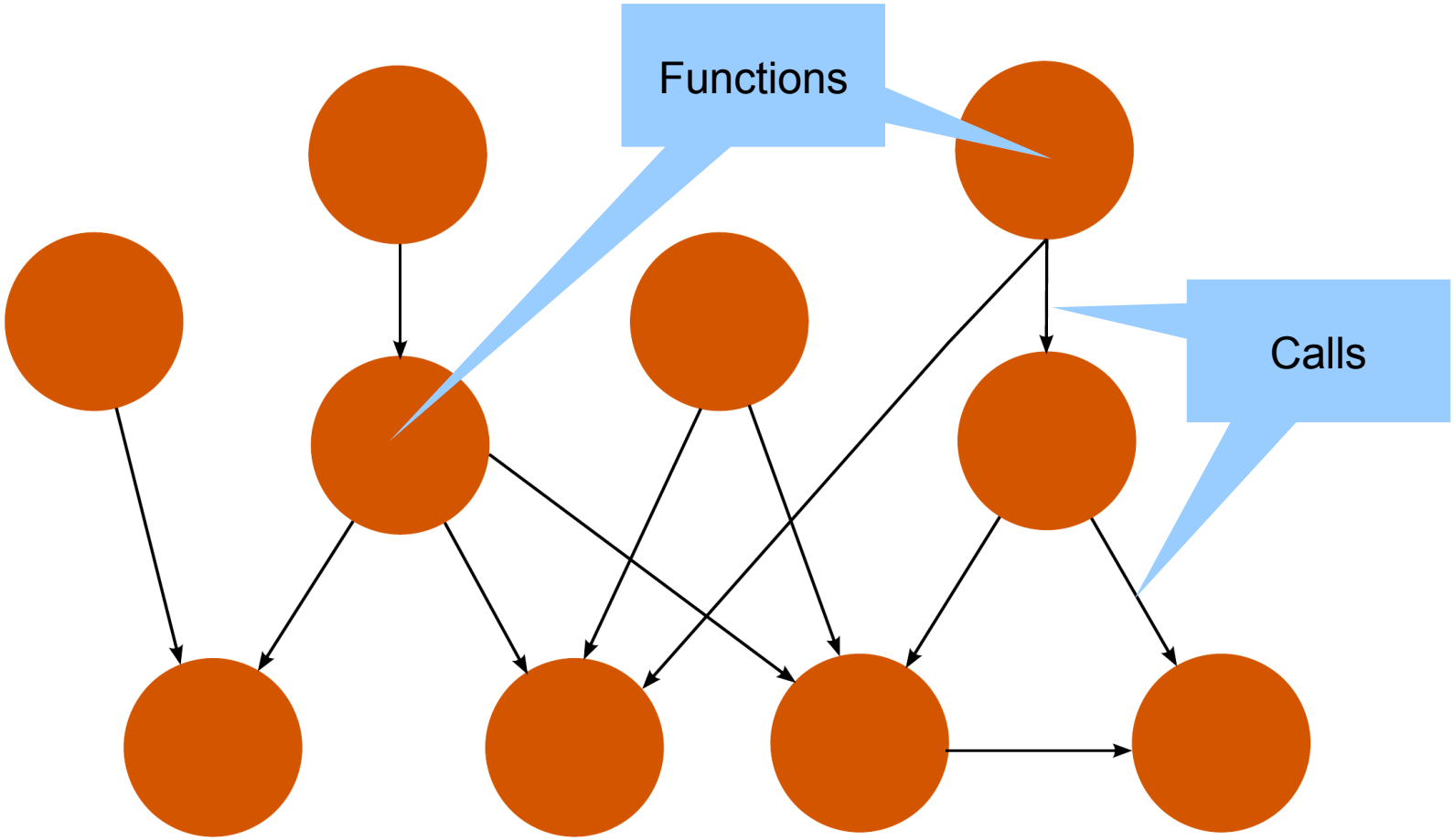
Obtaining the attack surface: an example



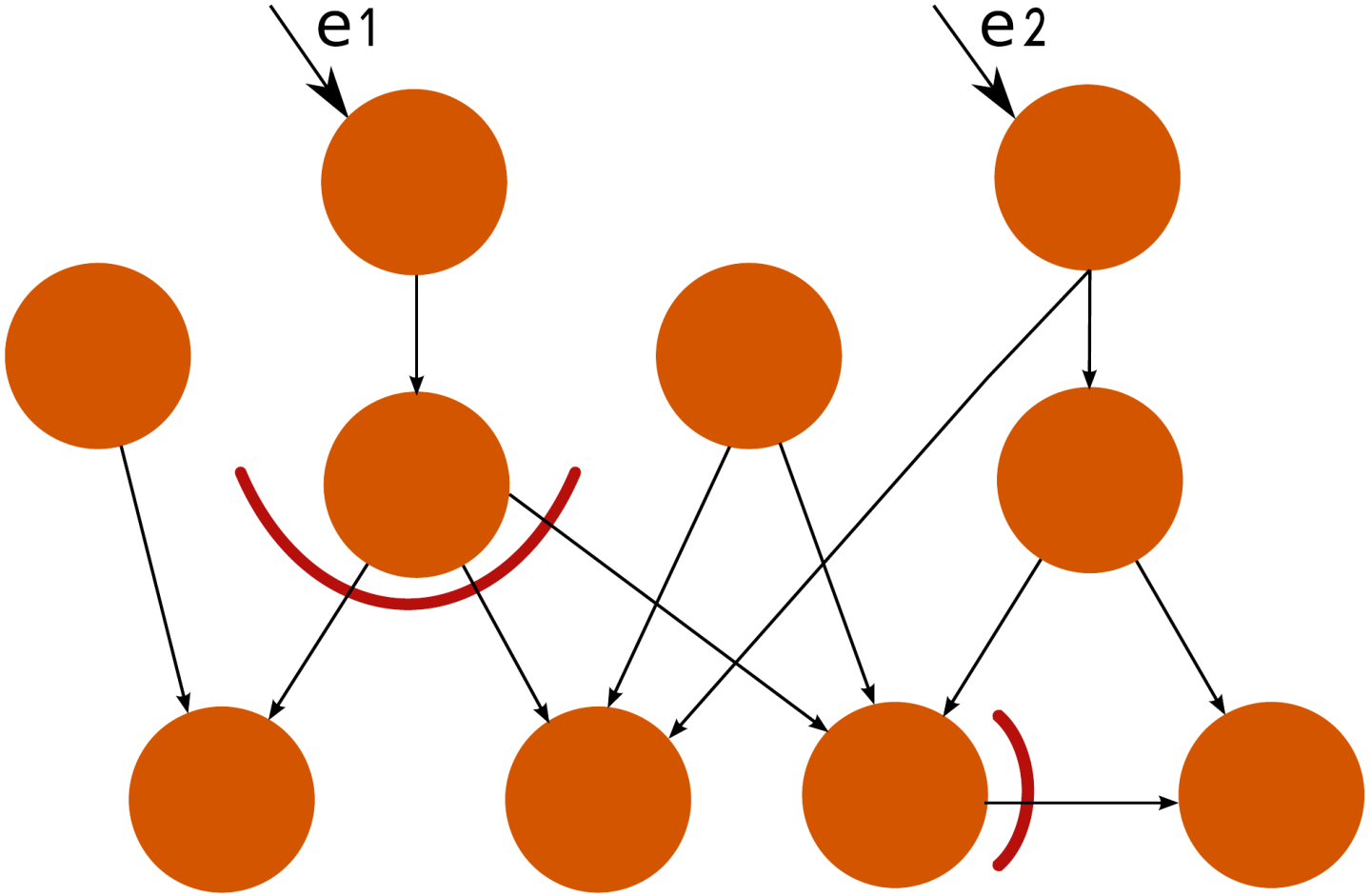
Obtaining the attack surface: an example



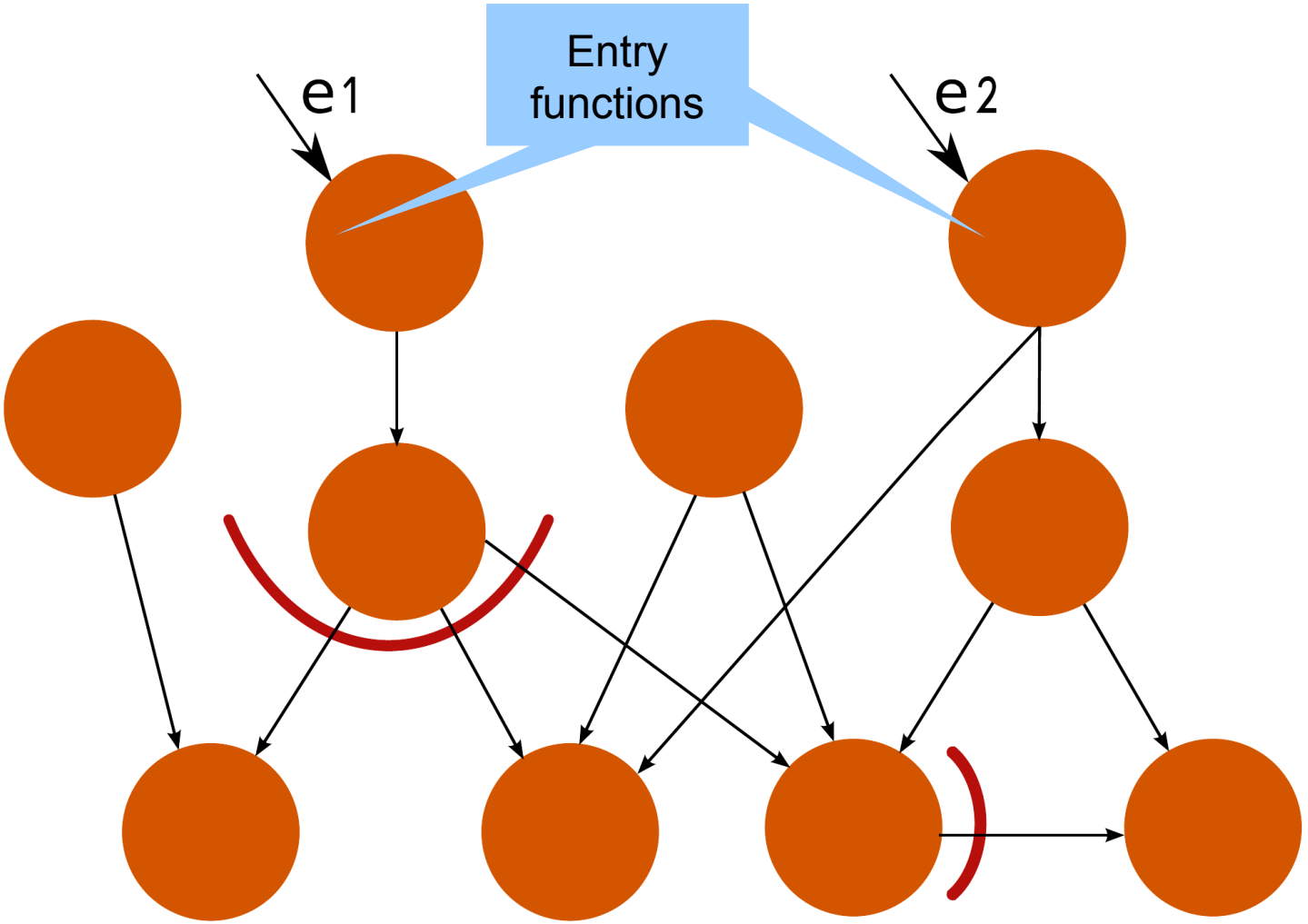
Obtaining the attack surface: an example



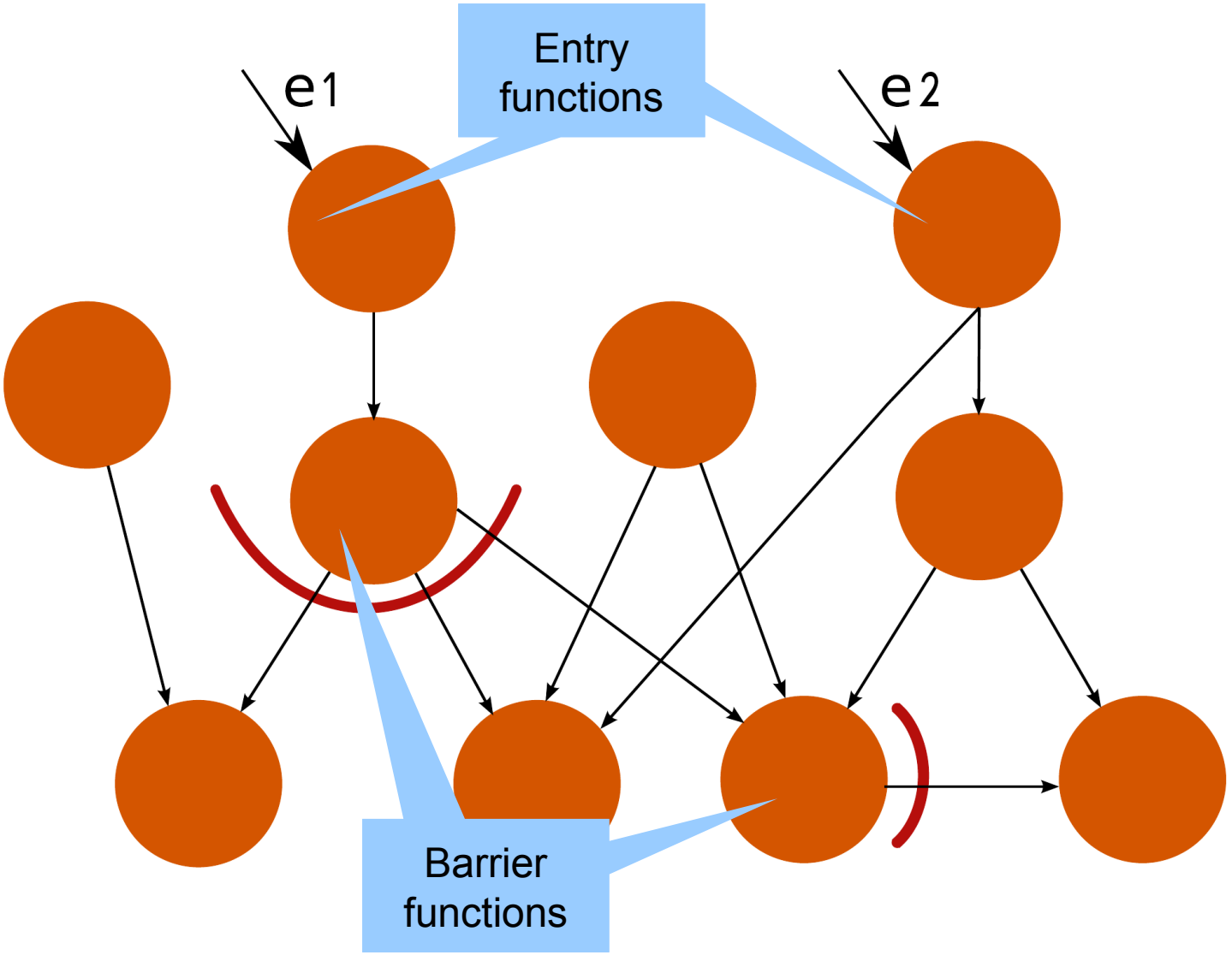
Obtaining the attack surface: an example



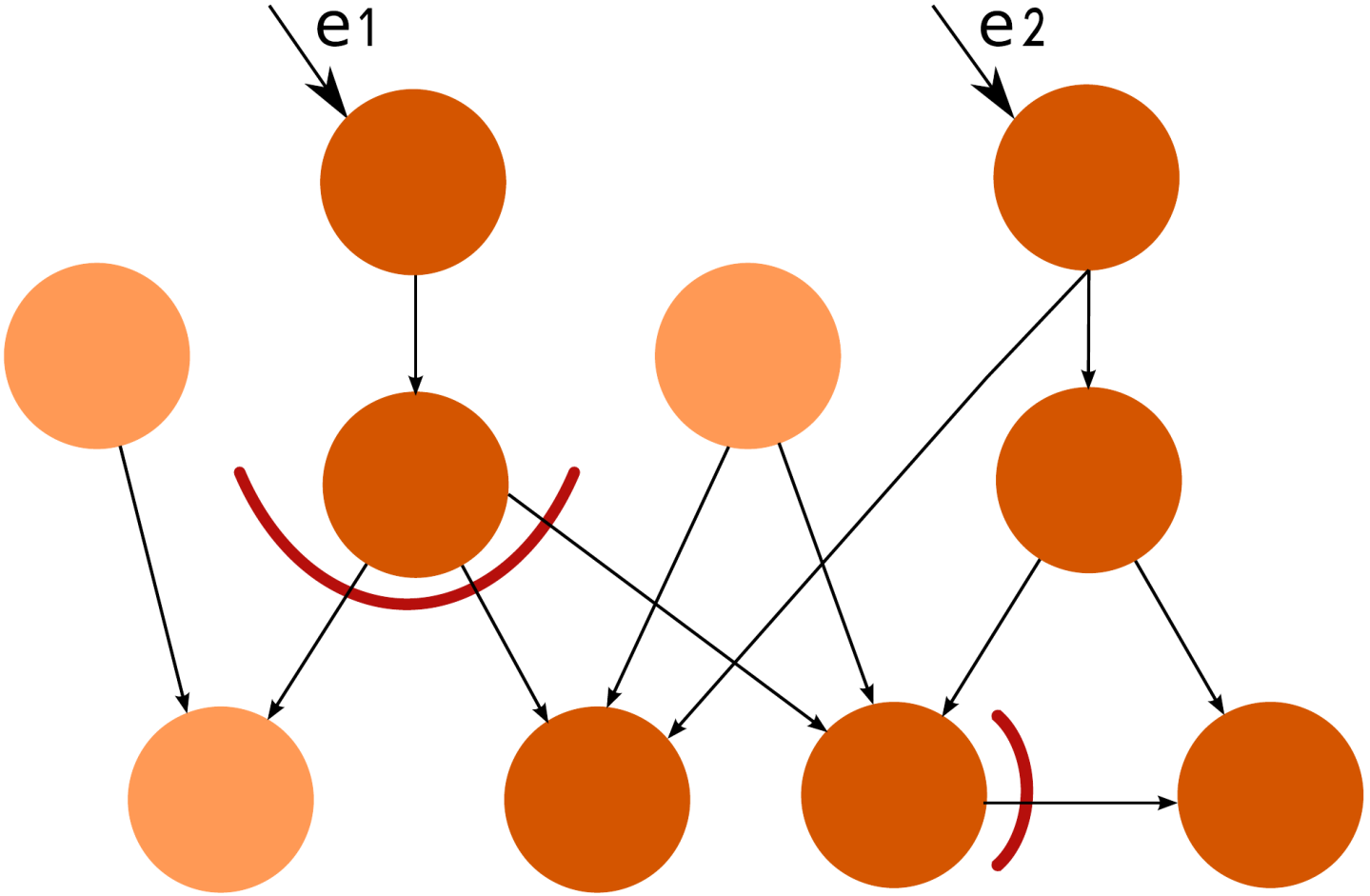
Obtaining the attack surface: an example



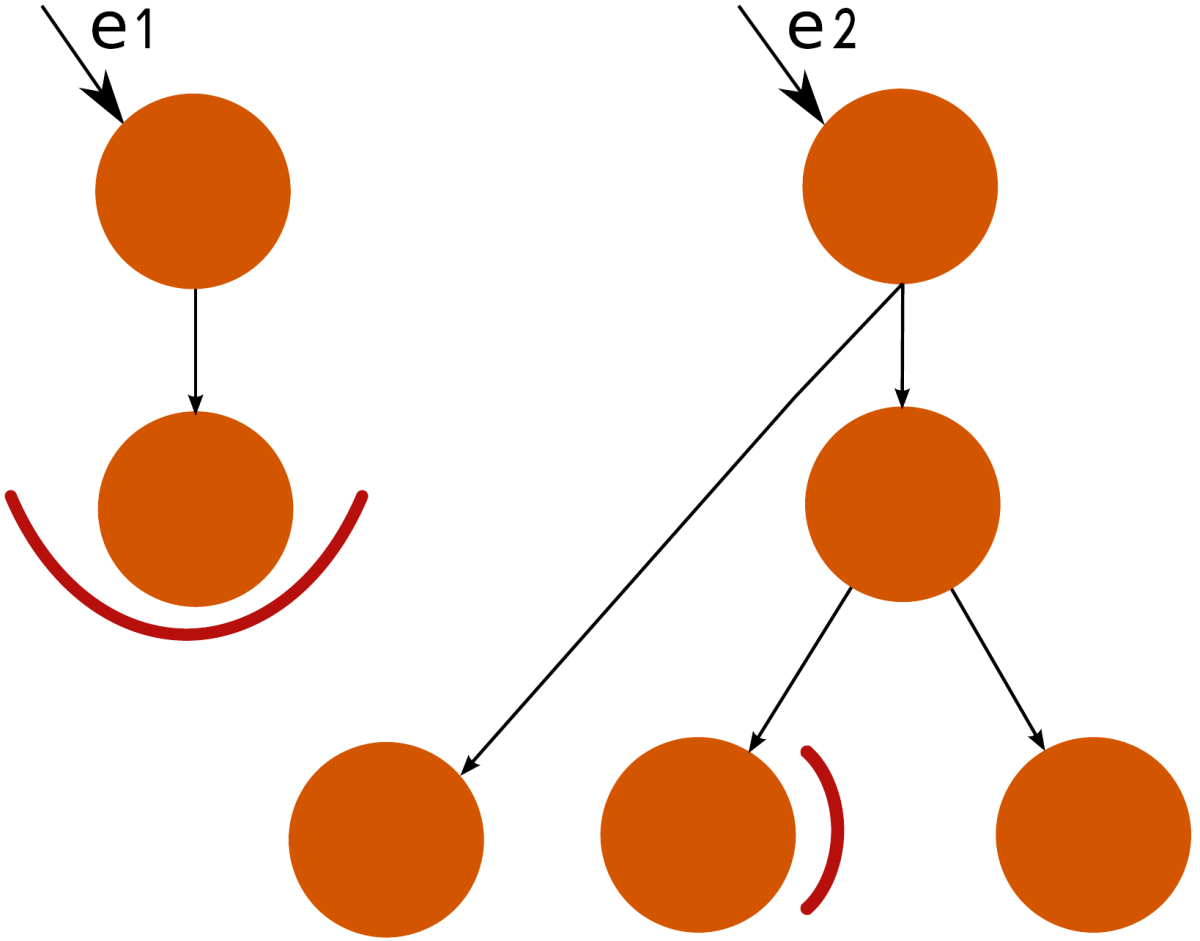
Obtaining the attack surface: an example



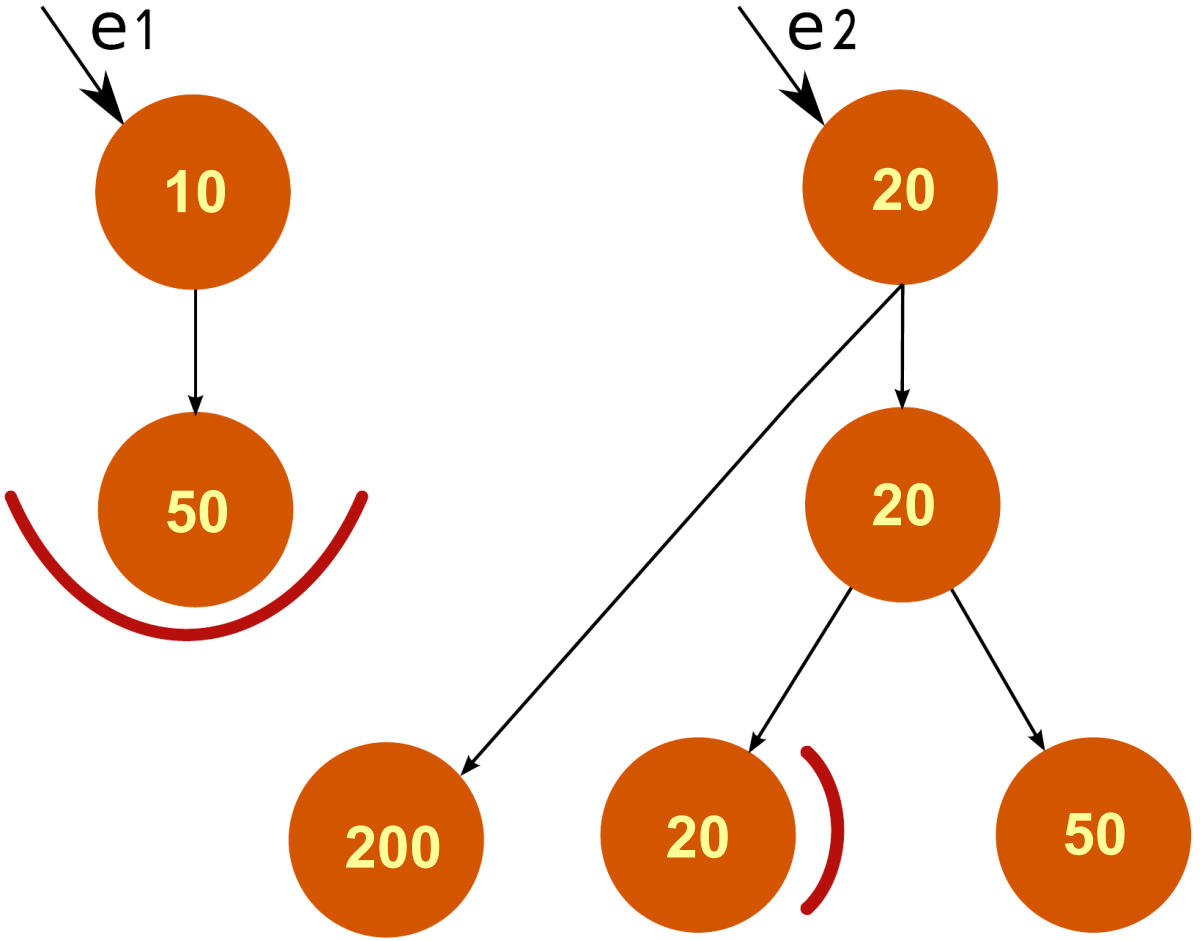
Obtaining the attack surface: an example



Obtaining the attack surface: an example



Code-quality metrics



Attack surface metrics

- Example:

$$AS1_{\mu}(G_{AS}) = \sum_{i \in F_{AS}} \mu(i)$$

- Code-quality metrics used:
  - source lines of code (SLOC),
  - cyclomatic complexity,
  - CVE-based metric

- See paper for formal definitions and an alternative attack surface metric (AS2)

Attack surface metrics

- Example:

Attack surface metric

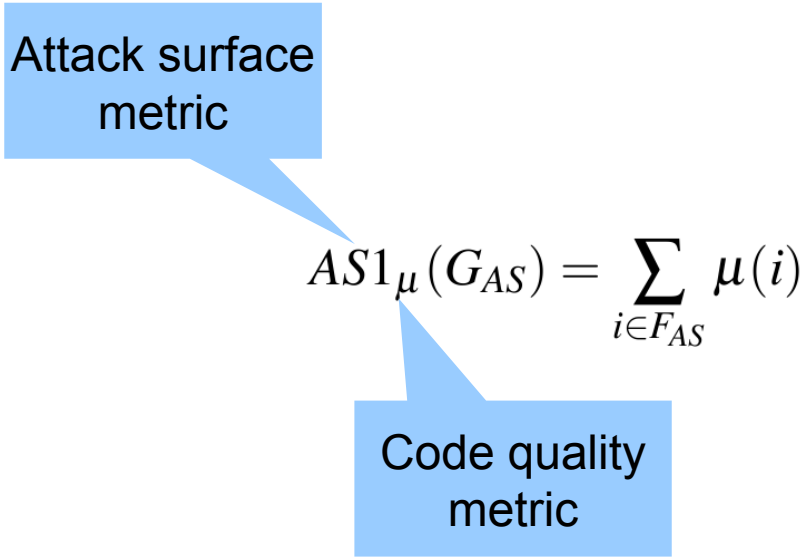
$$AS1_{\mu}(G_{AS}) = \sum_{i \in F_{AS}} \mu(i)$$

- Code-quality metrics used:
  - source lines of code (SLOC),
  - cyclomatic complexity,
  - CVE-based metric

- See paper for formal definitions and an alternative attack surface metric (AS2)

Attack surface metrics

- Example:

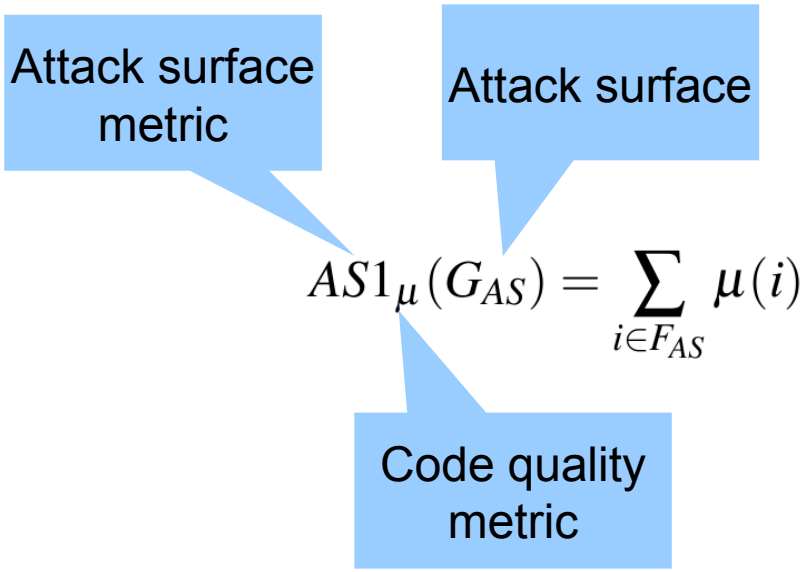


- Code-quality metrics used:
  - source lines of code (SLOC),
  - cyclomatic complexity,
  - CVE-based metric

- See paper for formal definitions and an alternative attack surface metric (AS2)

Attack surface metrics

- Example:

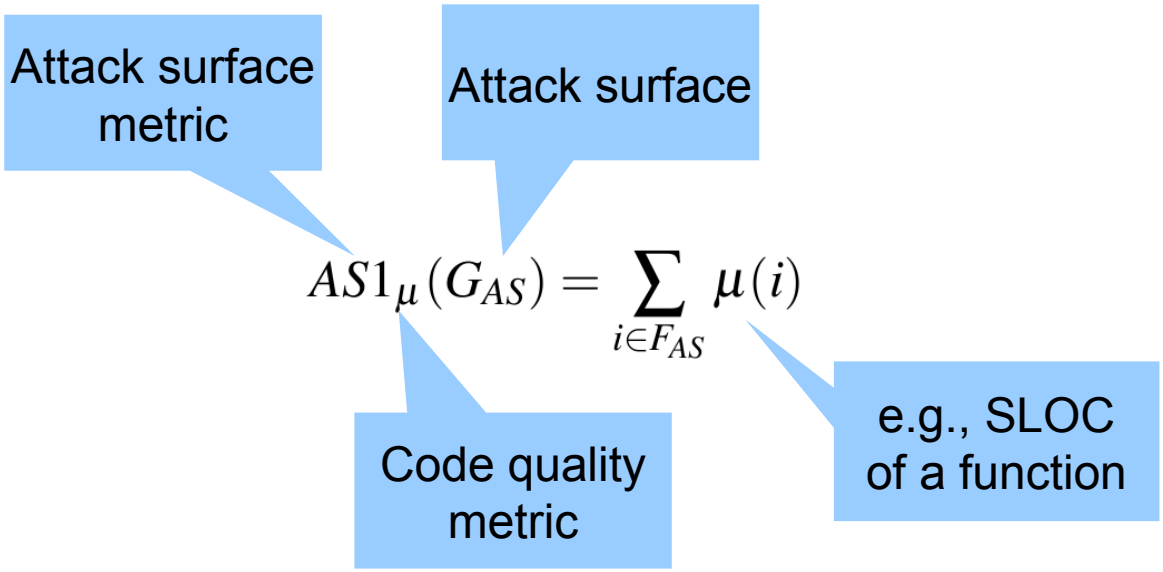


- Code-quality metrics used:
  - source lines of code (SLOC),
  - cyclomatic complexity,
  - CVE-based metric

- See paper for formal definitions and an alternative attack surface metric (AS2)

Attack surface metrics

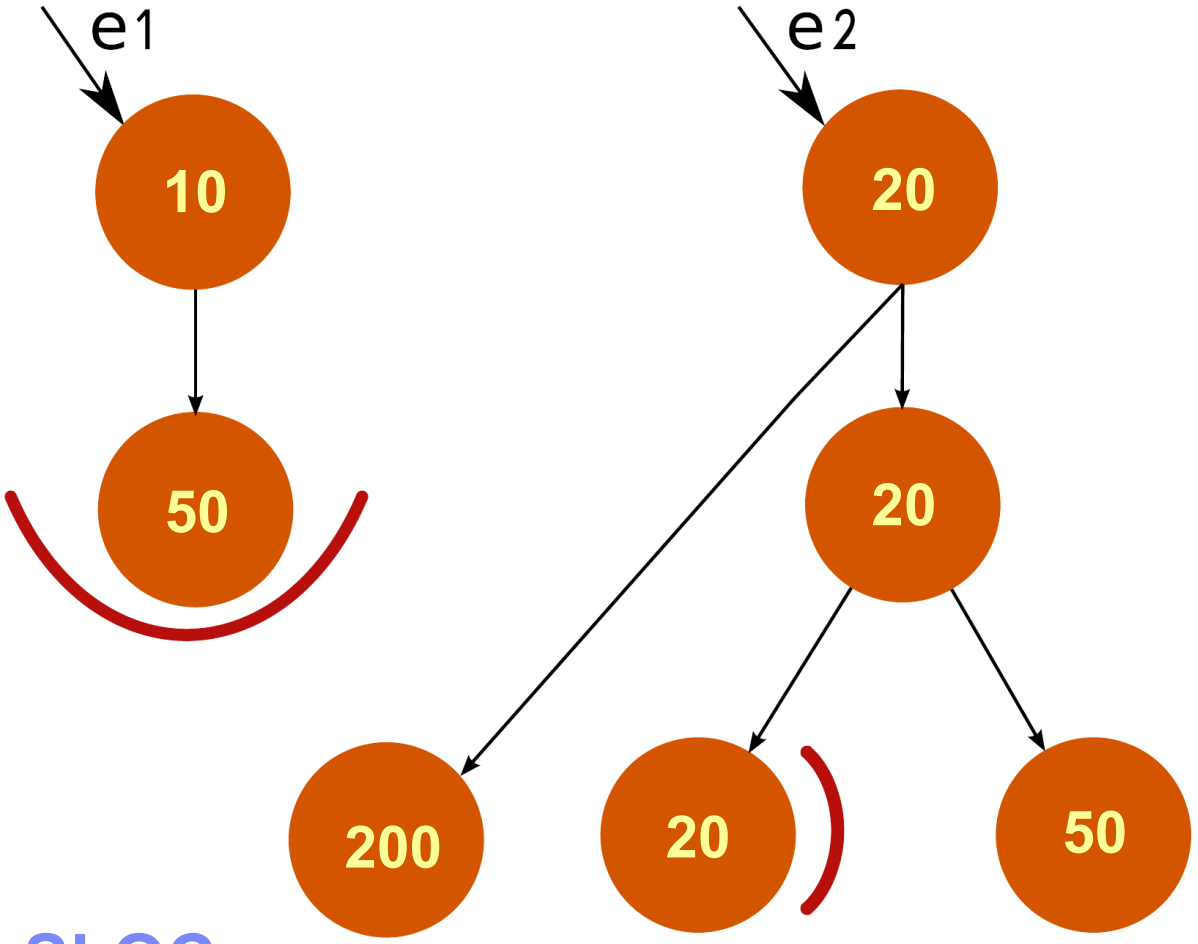
- Example:



- Code-quality metrics used:
  - source lines of code (SLOC),
  - cyclomatic complexity,
  - CVE-based metric

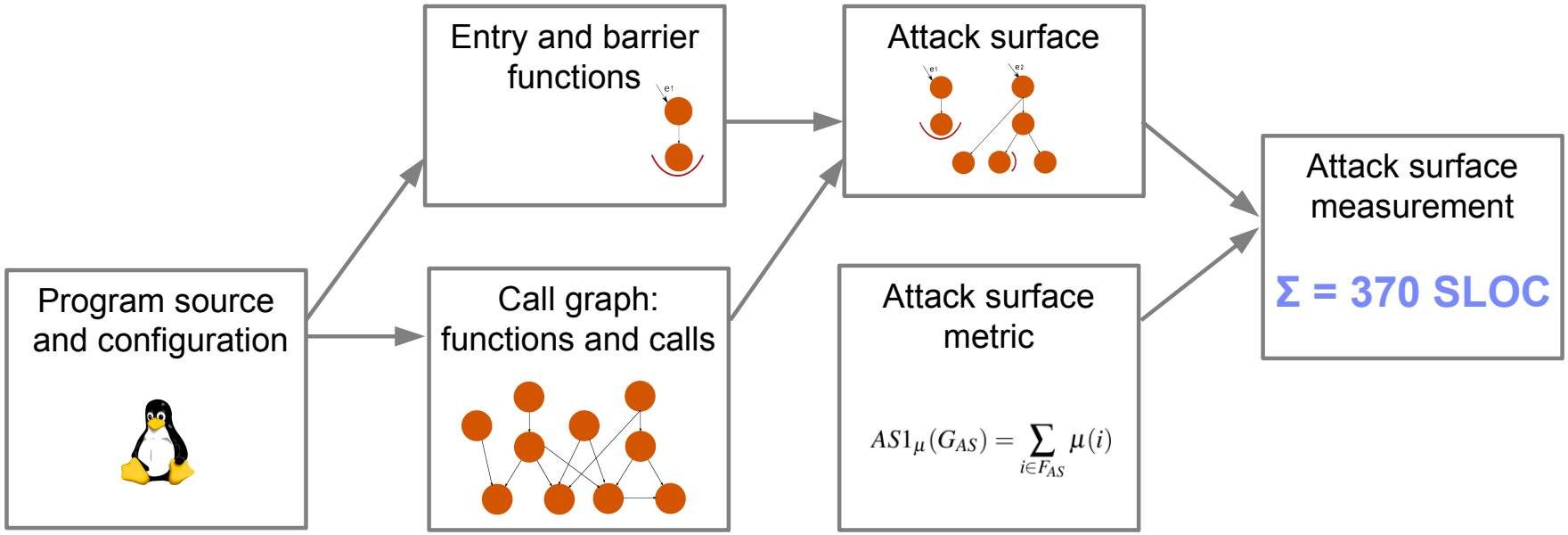
- See paper for formal definitions and an alternative attack surface metric (AS2)

Attack surface measurement: AS1

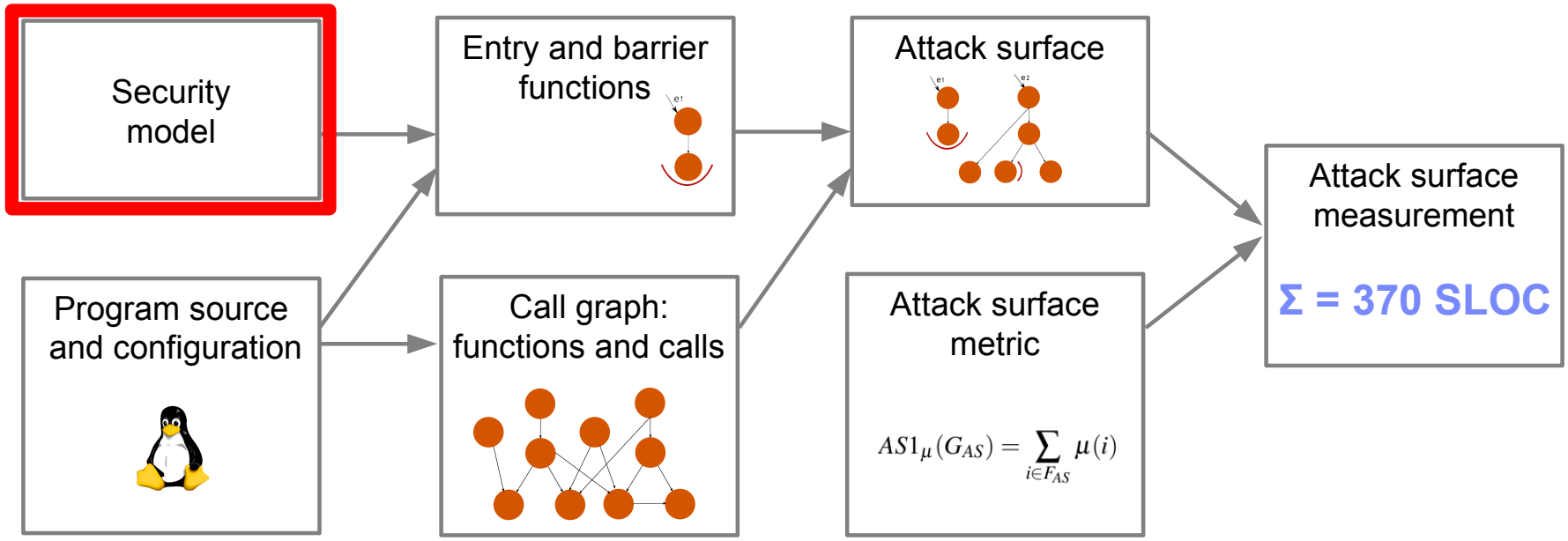


Σ = 370 SLOC

Attack surface measurements: summary

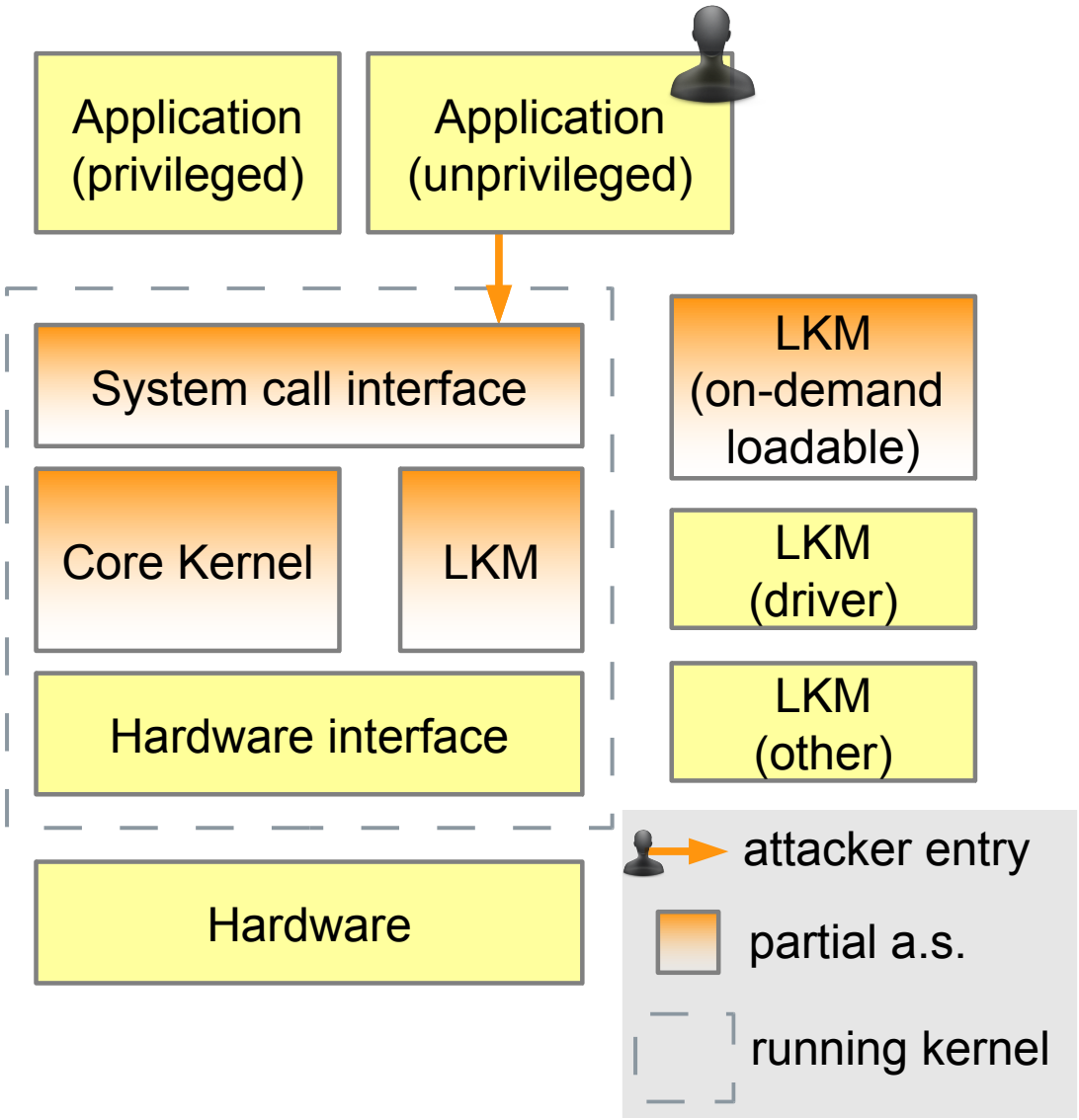


Attack surface measurements: summary

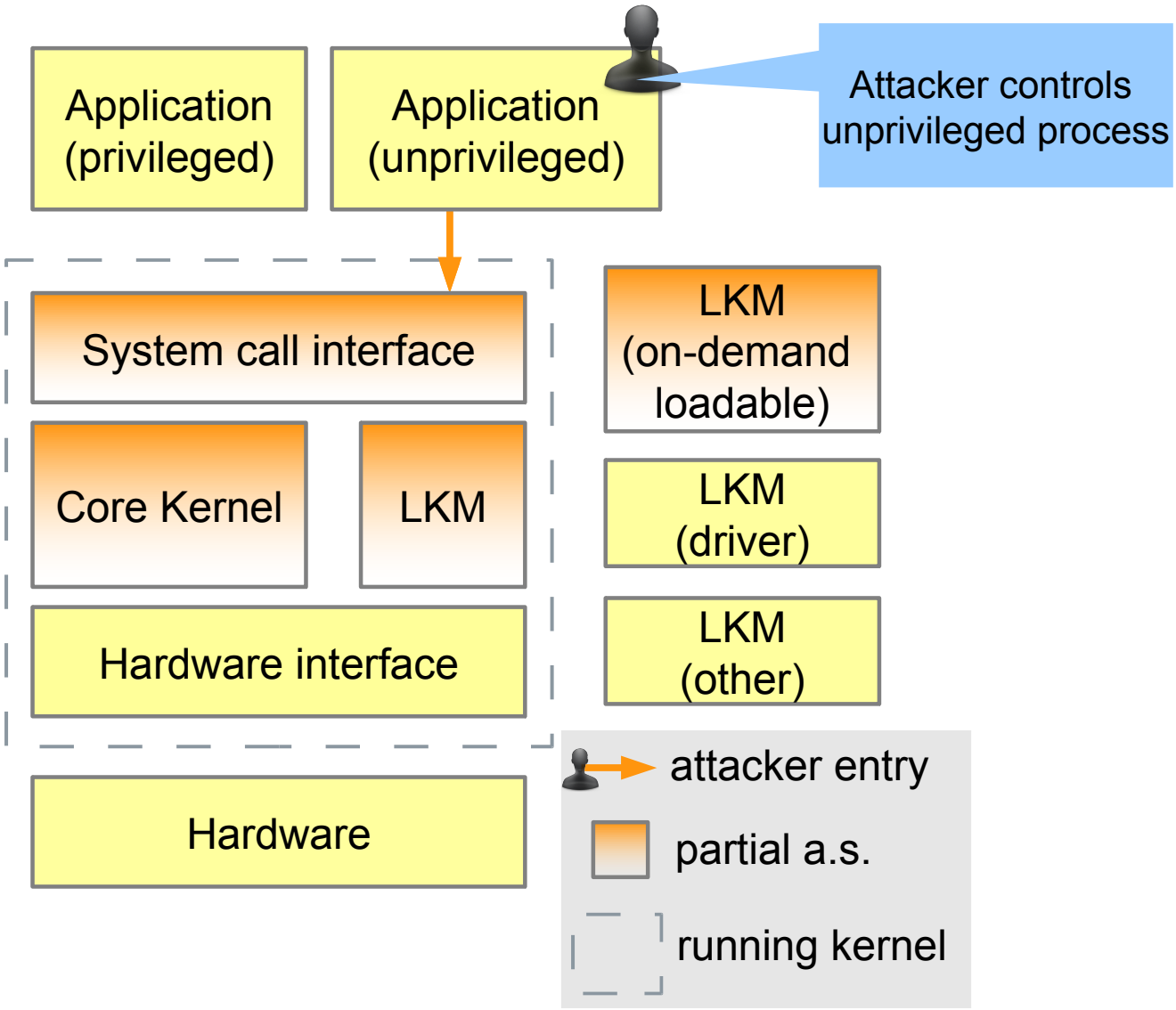


What security model?

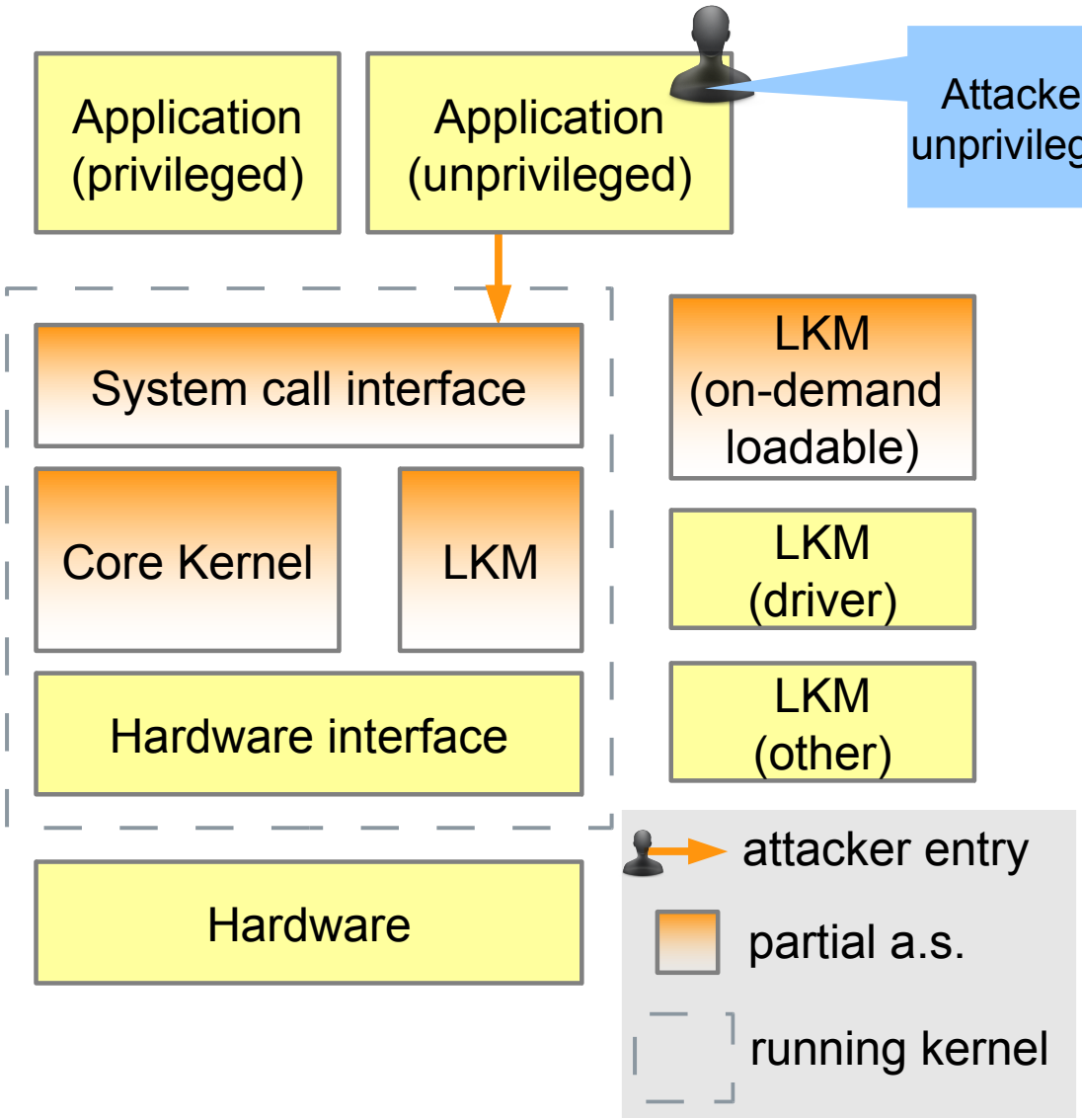
IsolSec Linux Kernel Security Model



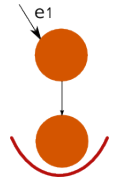
IsolSec Linux Kernel Security Model



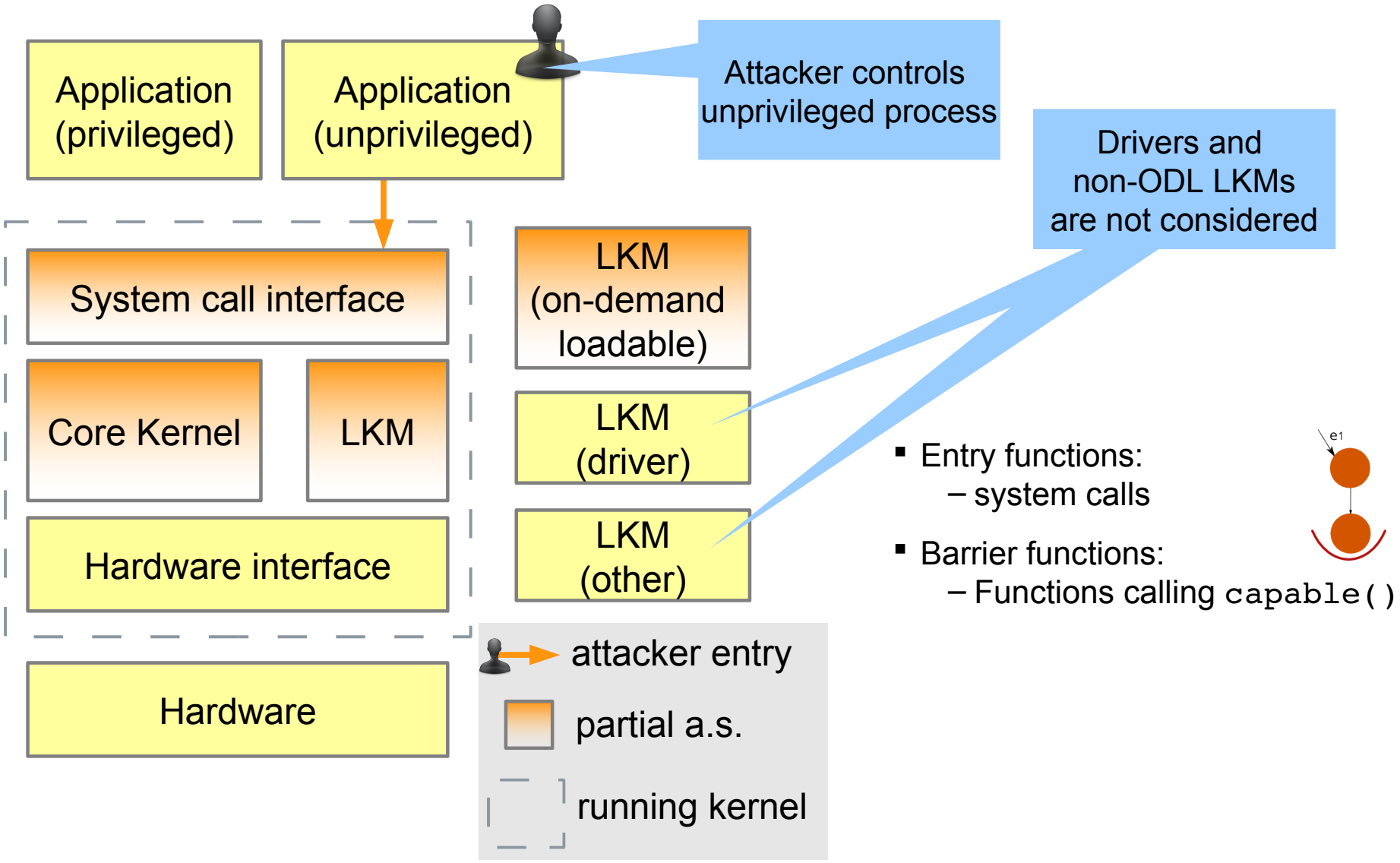
IsolSec Linux Kernel Security Model



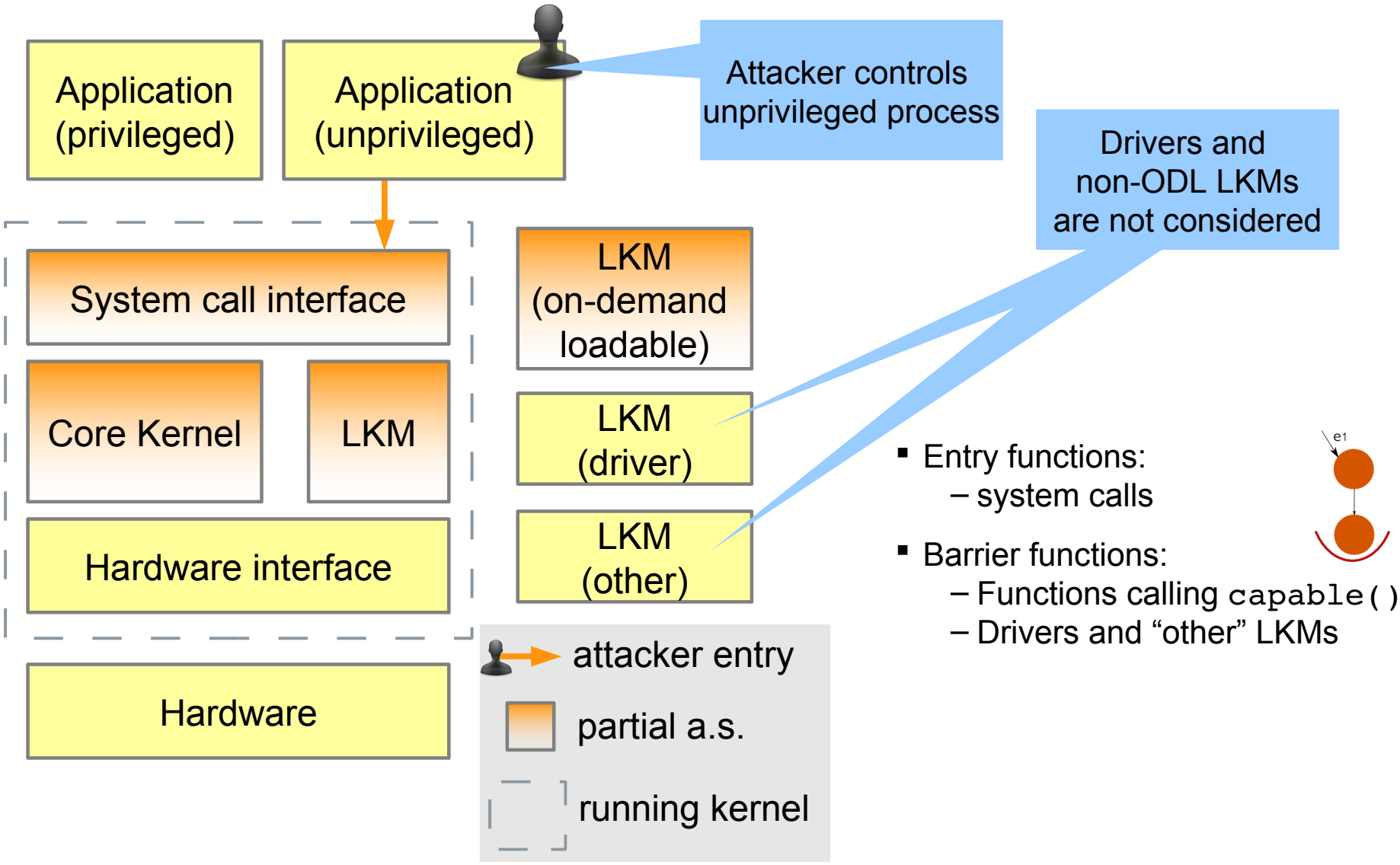
- Entry functions:
  - system calls
- Barrier functions:
  - Functions calling `capable()`



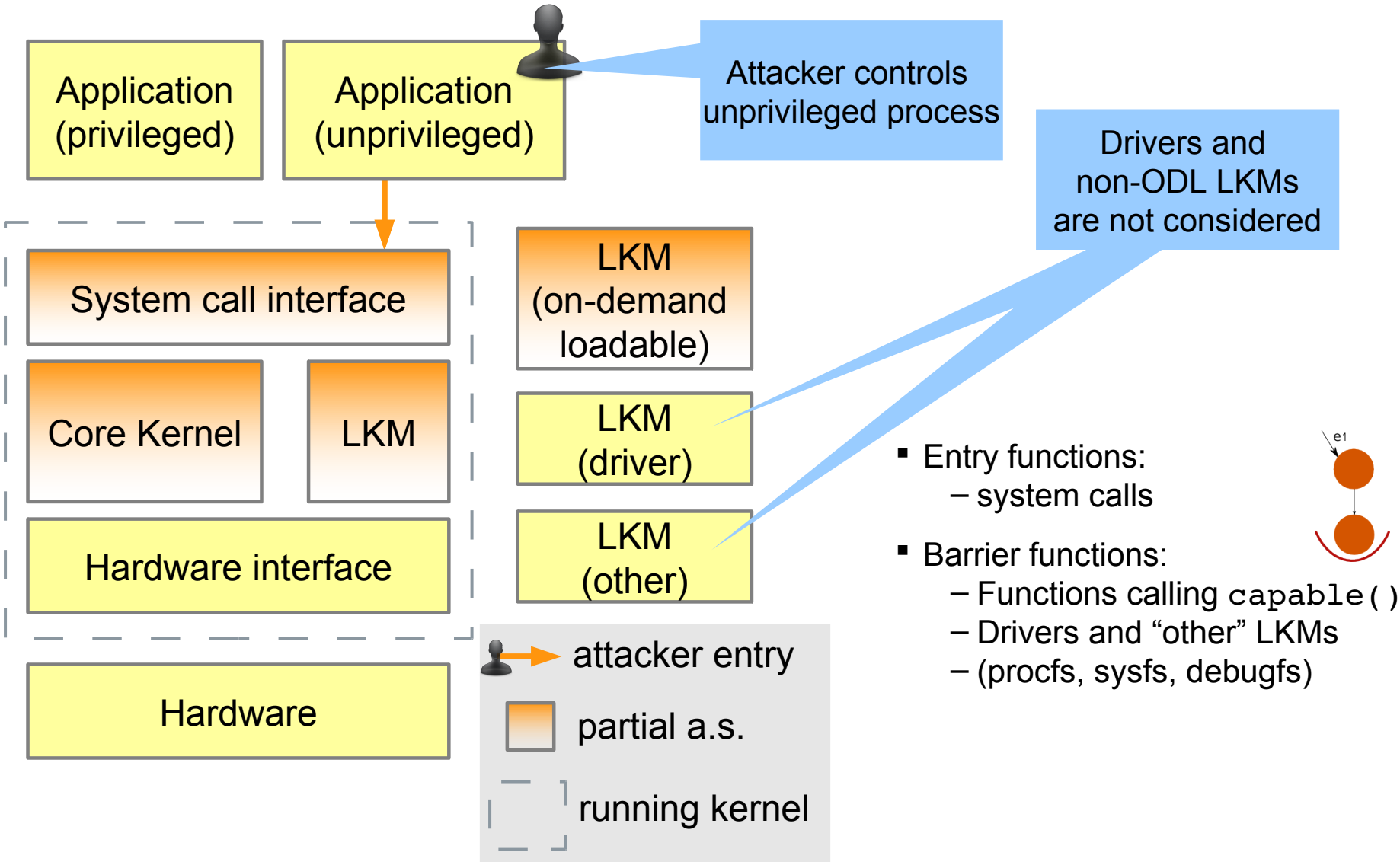
IsolSec Linux Kernel Security Model



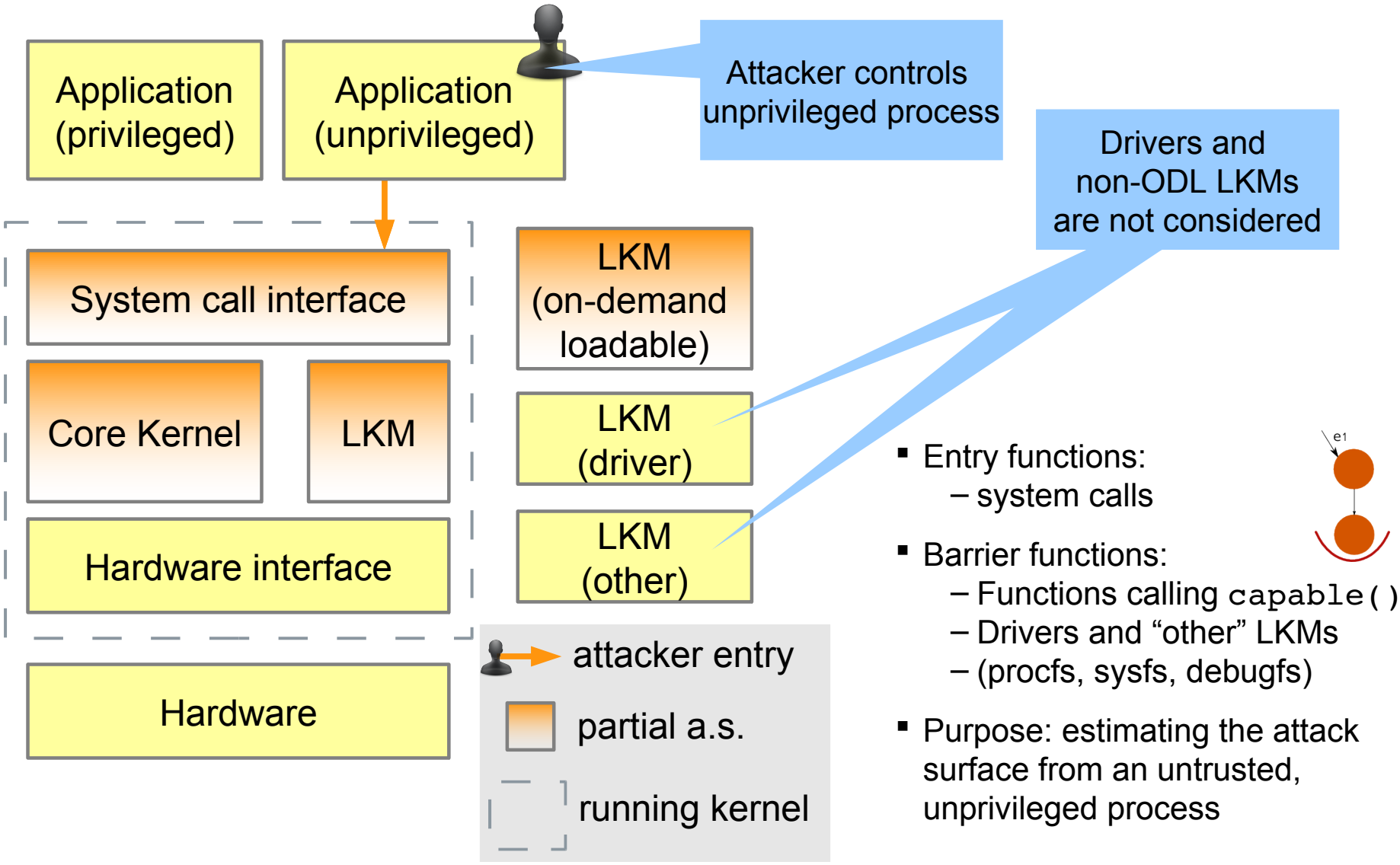
IsolSec Linux Kernel Security Model



IsolSec Linux Kernel Security Model

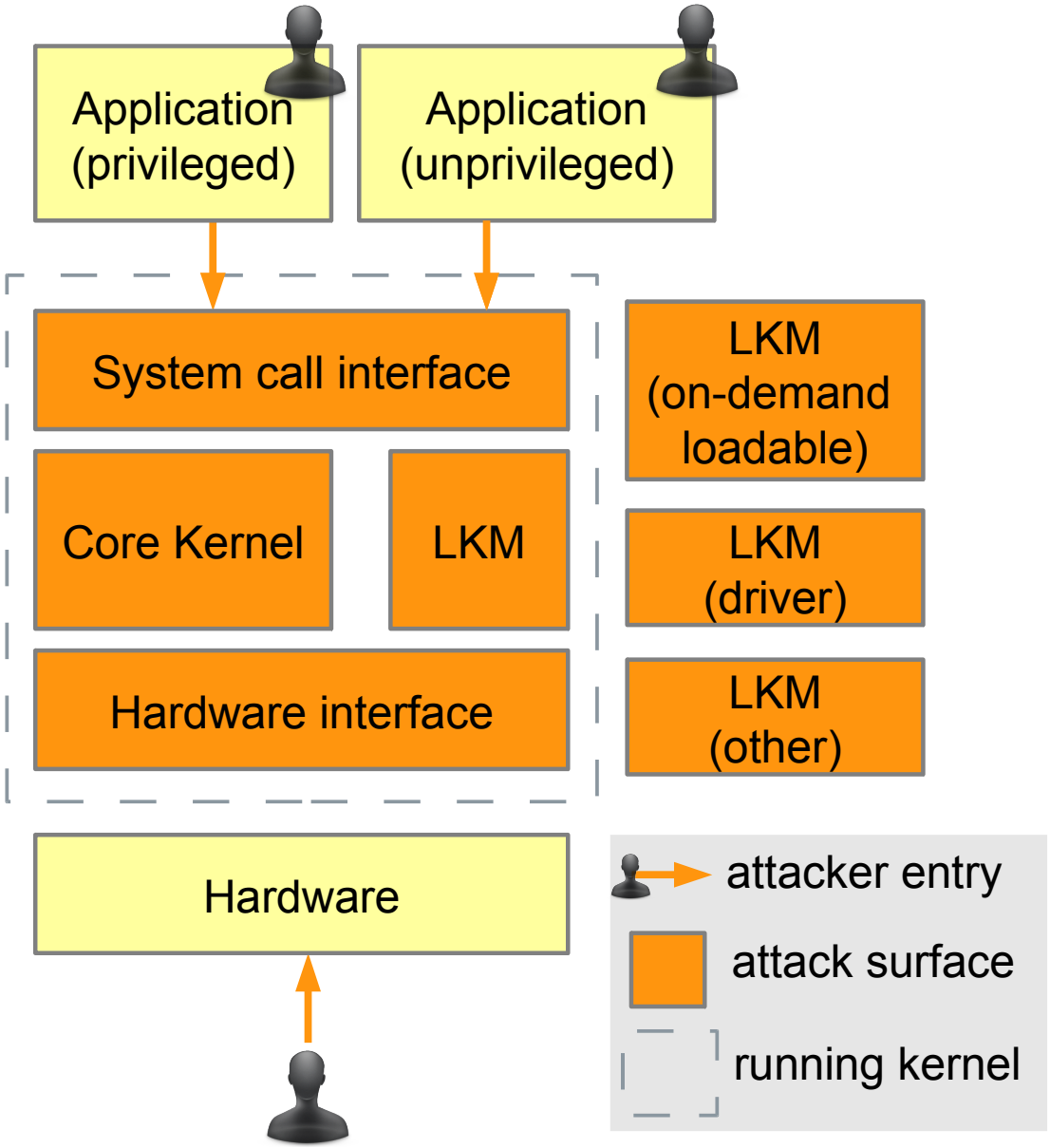


IsolSec Linux Kernel Security Model

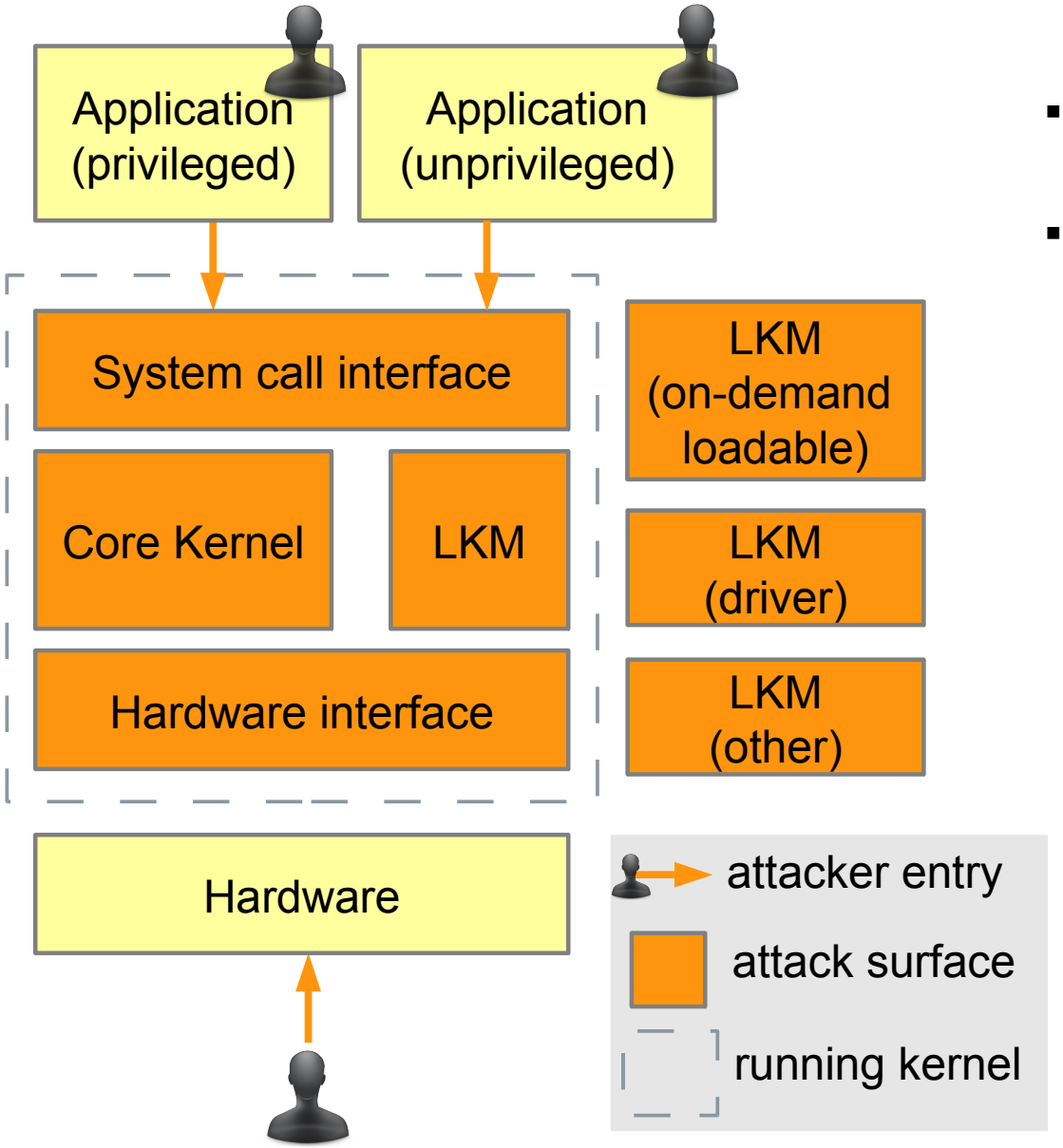


- Entry functions:
  - system calls
- Barrier functions:
  - Functions calling `capable()`
  - Drivers and “other” LKMs
  - (procfs, sysfs, debugfs)
- Purpose: estimating the attack surface from an untrusted, unprivileged process

GenSec Linux Kernel Security Model

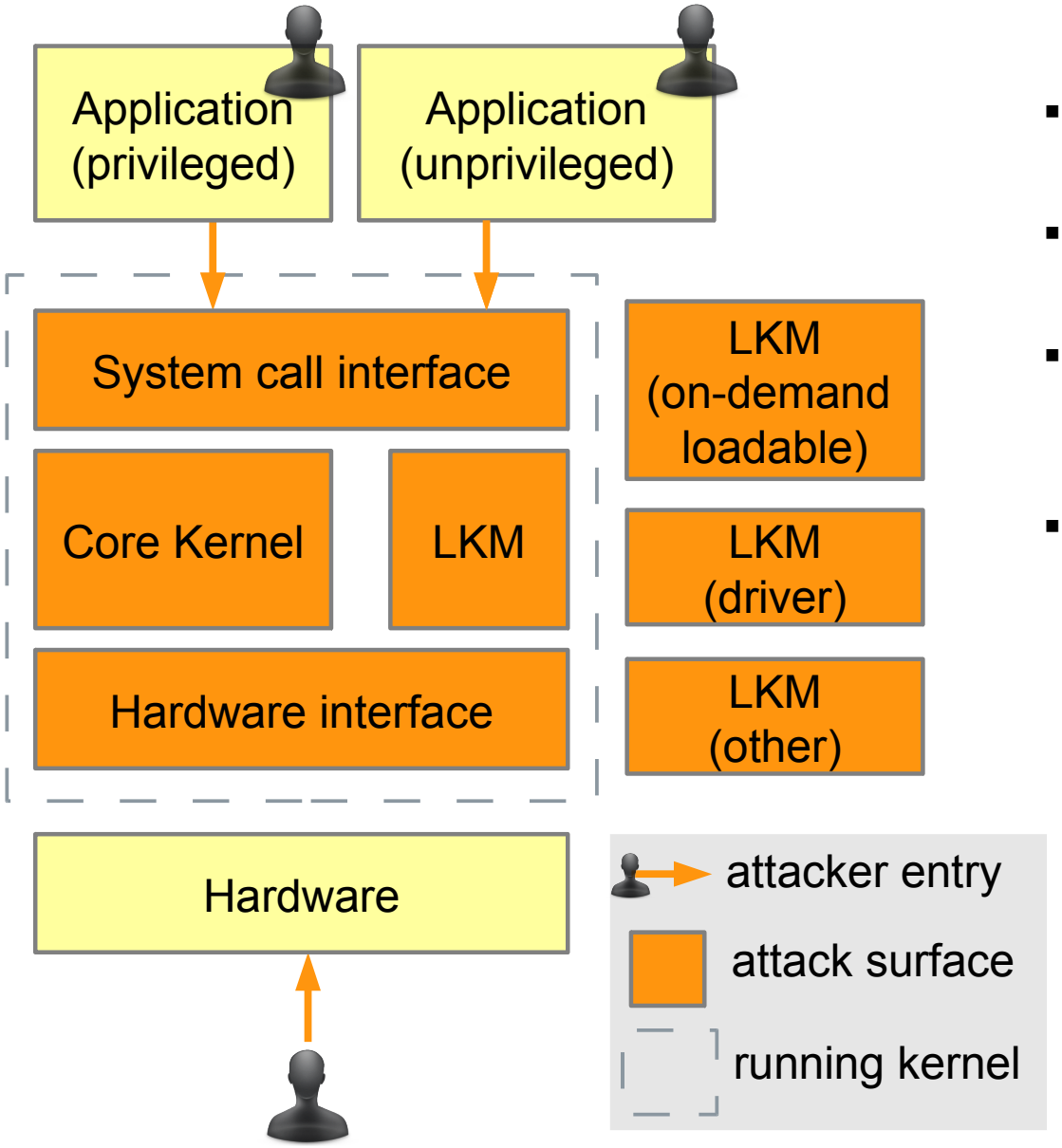


GenSec Linux Kernel Security Model



- Entry functions:
  - all
- Barrier functions:
  - none

GenSec Linux Kernel Security Model



- Entry functions:
  - all
- Barrier functions:
  - none
- Overestimates attack surface
  - attacker is privileged?
  - not all LKMs can be loaded
- Purpose:
  - upper bound
  - TCB point of view

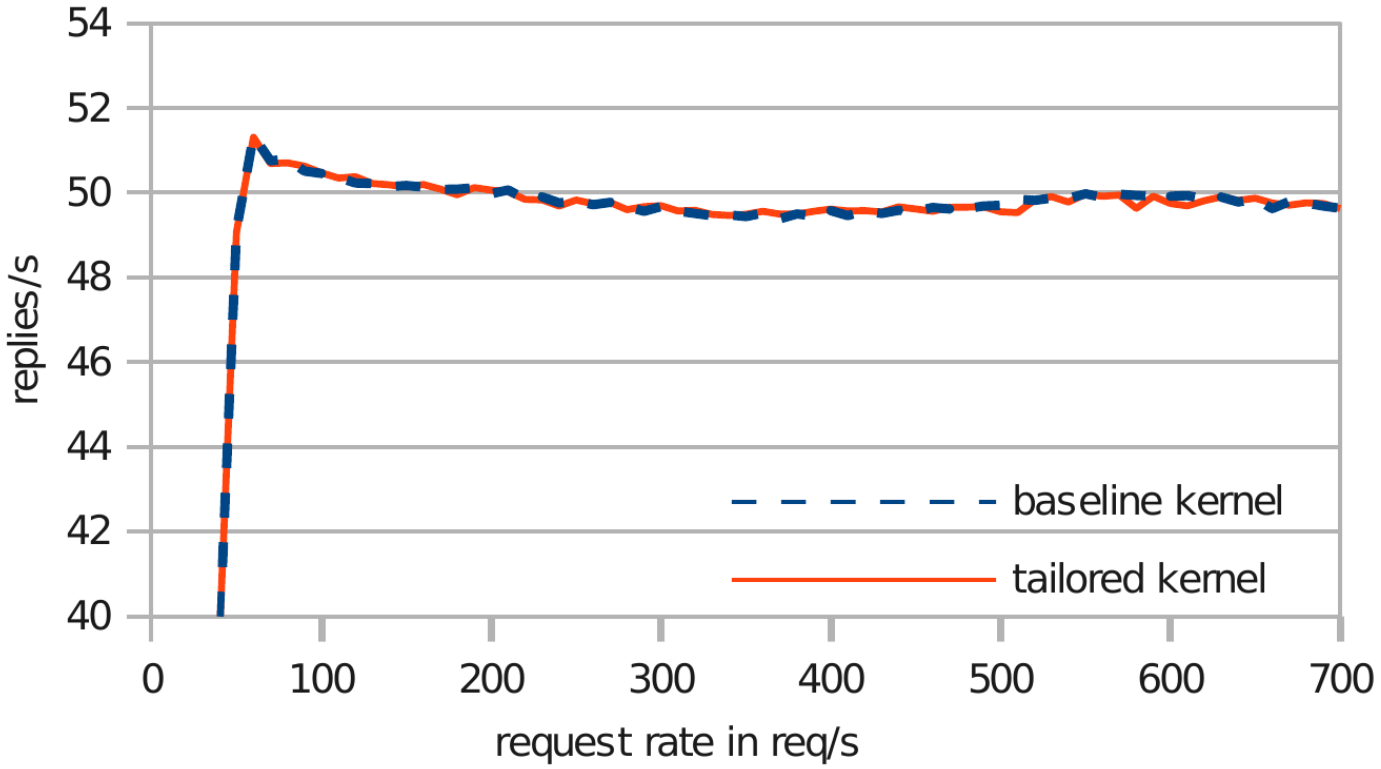
Evaluation

- Questions to answer experimentally:
  - Is there a **performance** difference?
  - Is **tracing** sufficient?
  - How much **attack surface reduction**?

- Popular and recent Linux distribution
- Typical server use case: LAMP

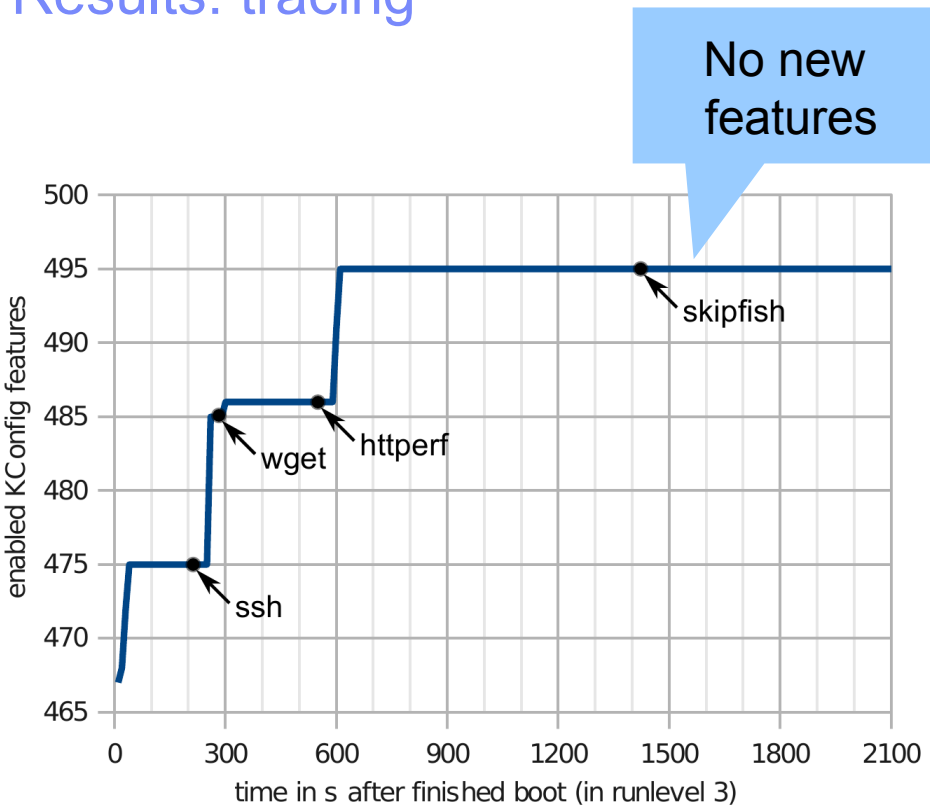

- There is also an NFS server use case in the paper.

Results: performance



No significant performance difference

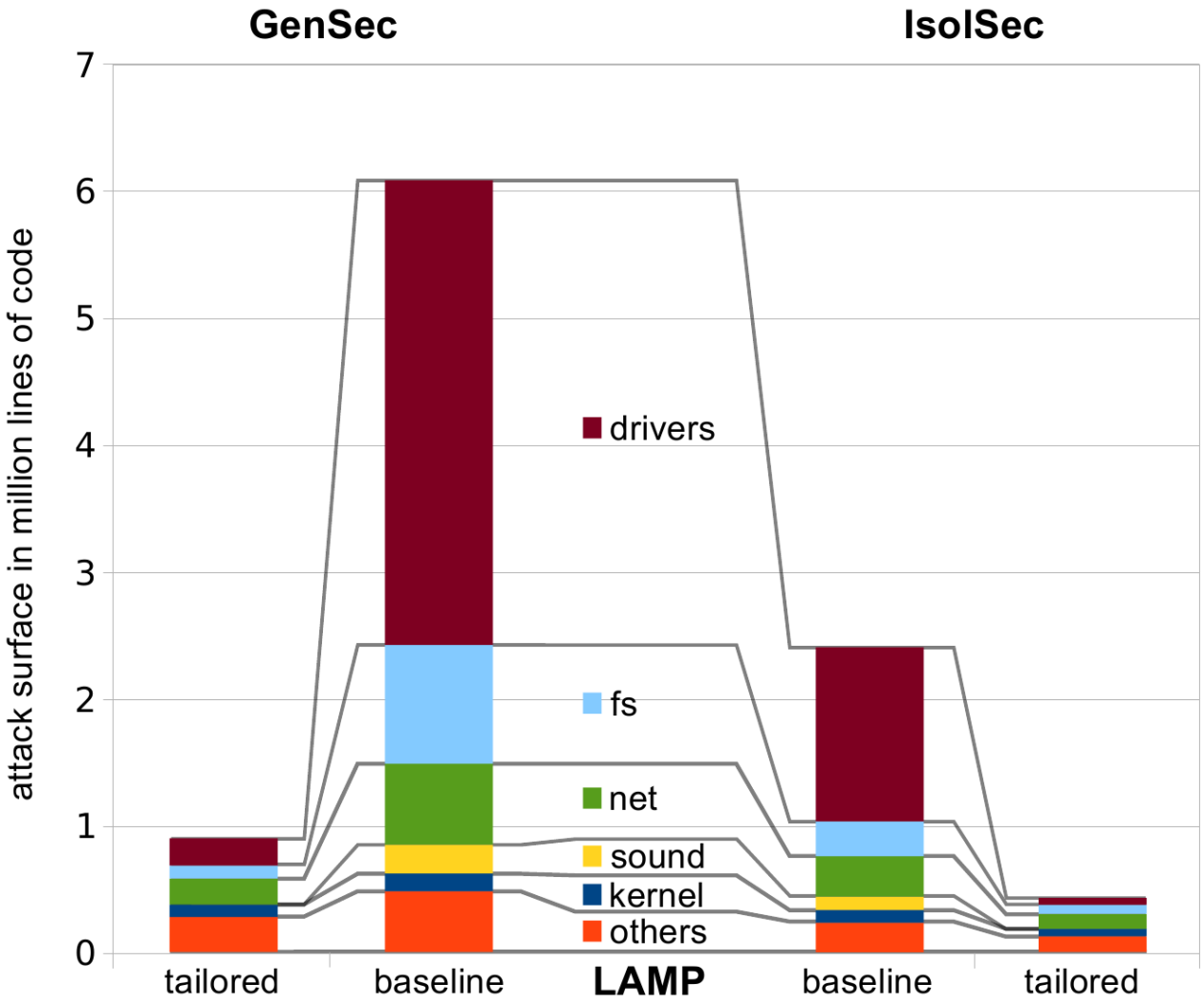
Results: tracing



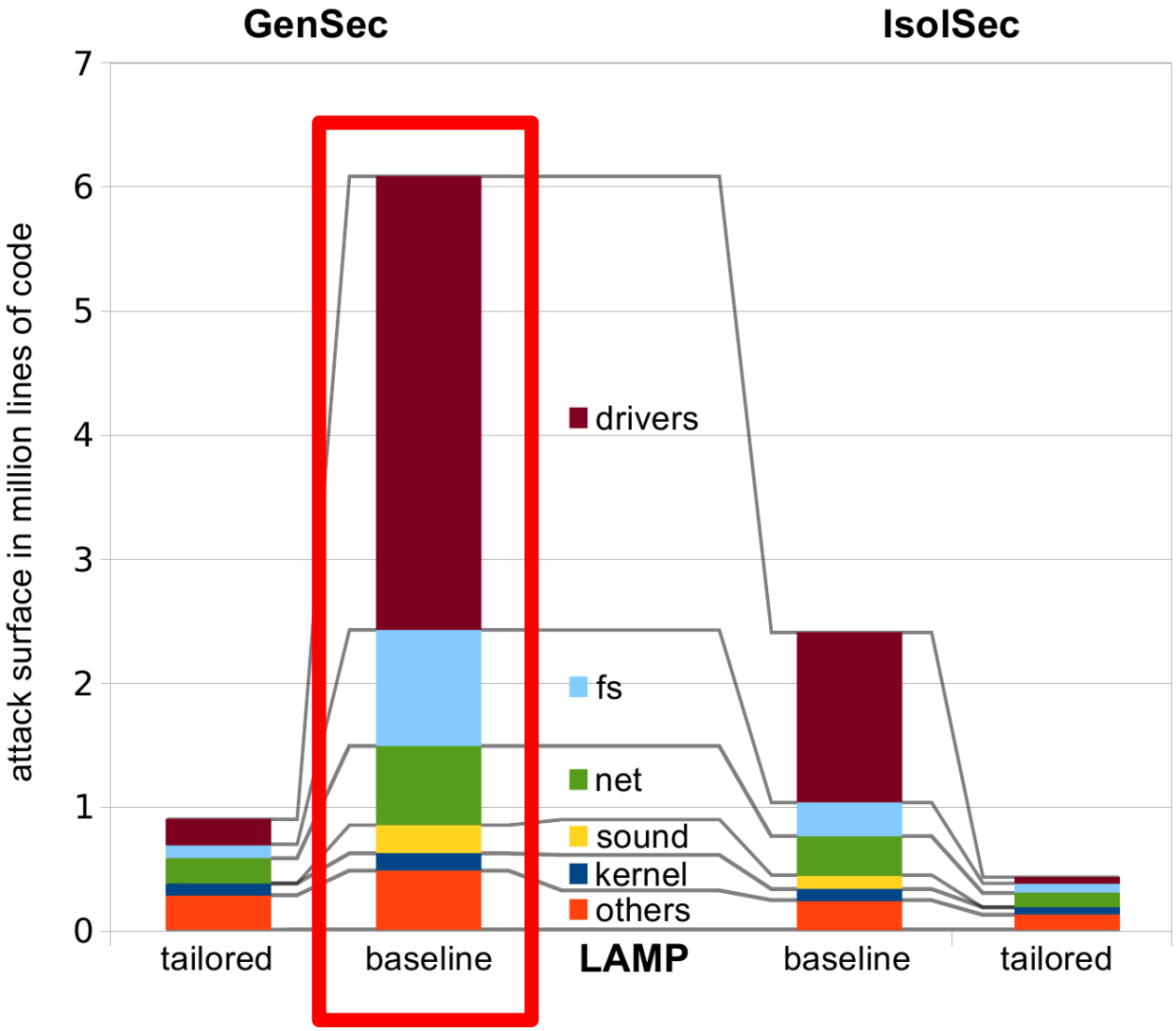
- Httperf benchmark triggers new features
  - Stabilizes at 495 features
- Skipfish: high coverage of the web application
  - Goes beyond real-world workload

Tracing at feature-granularity stabilizes quickly

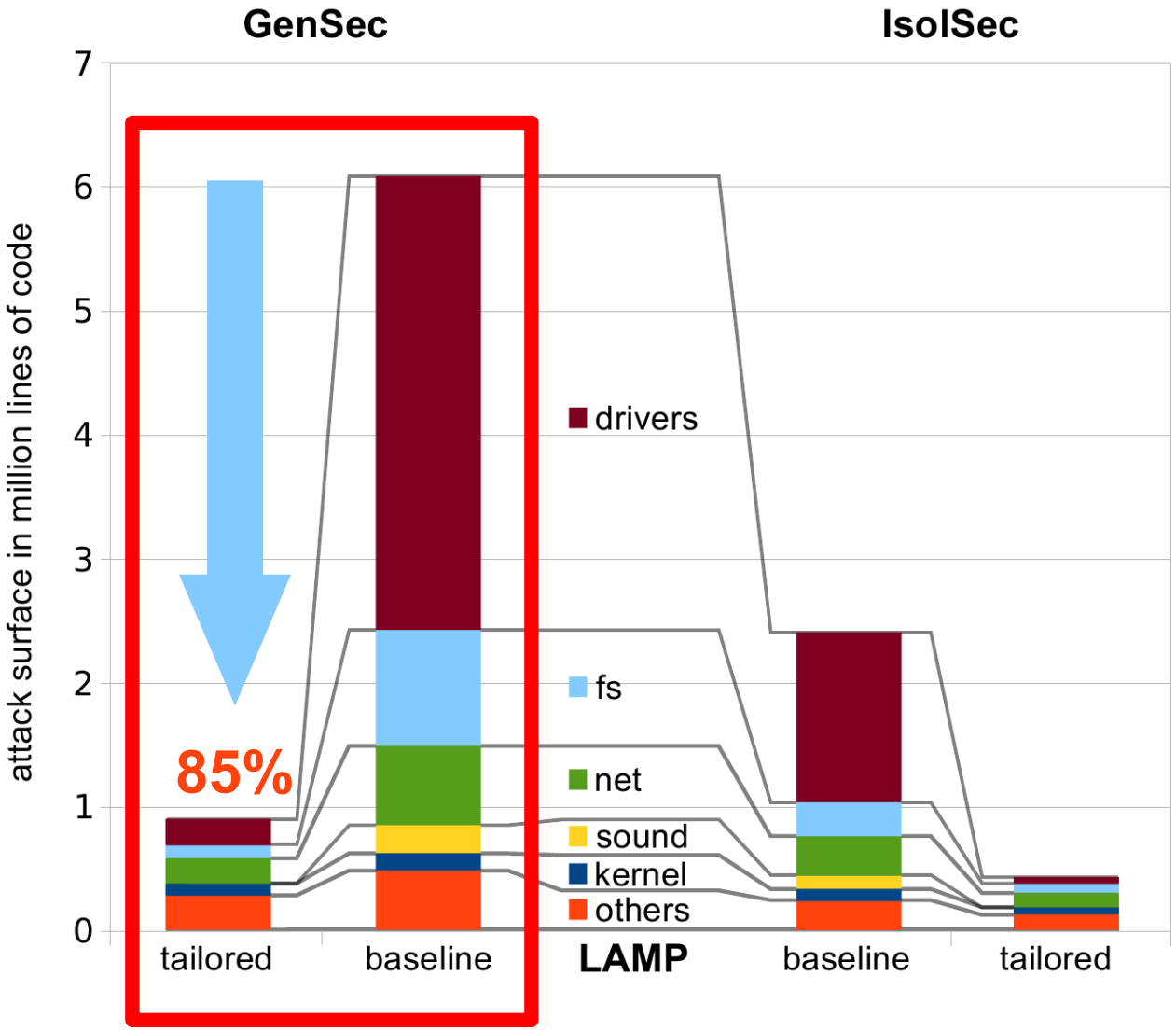
Results: attack surface reduction



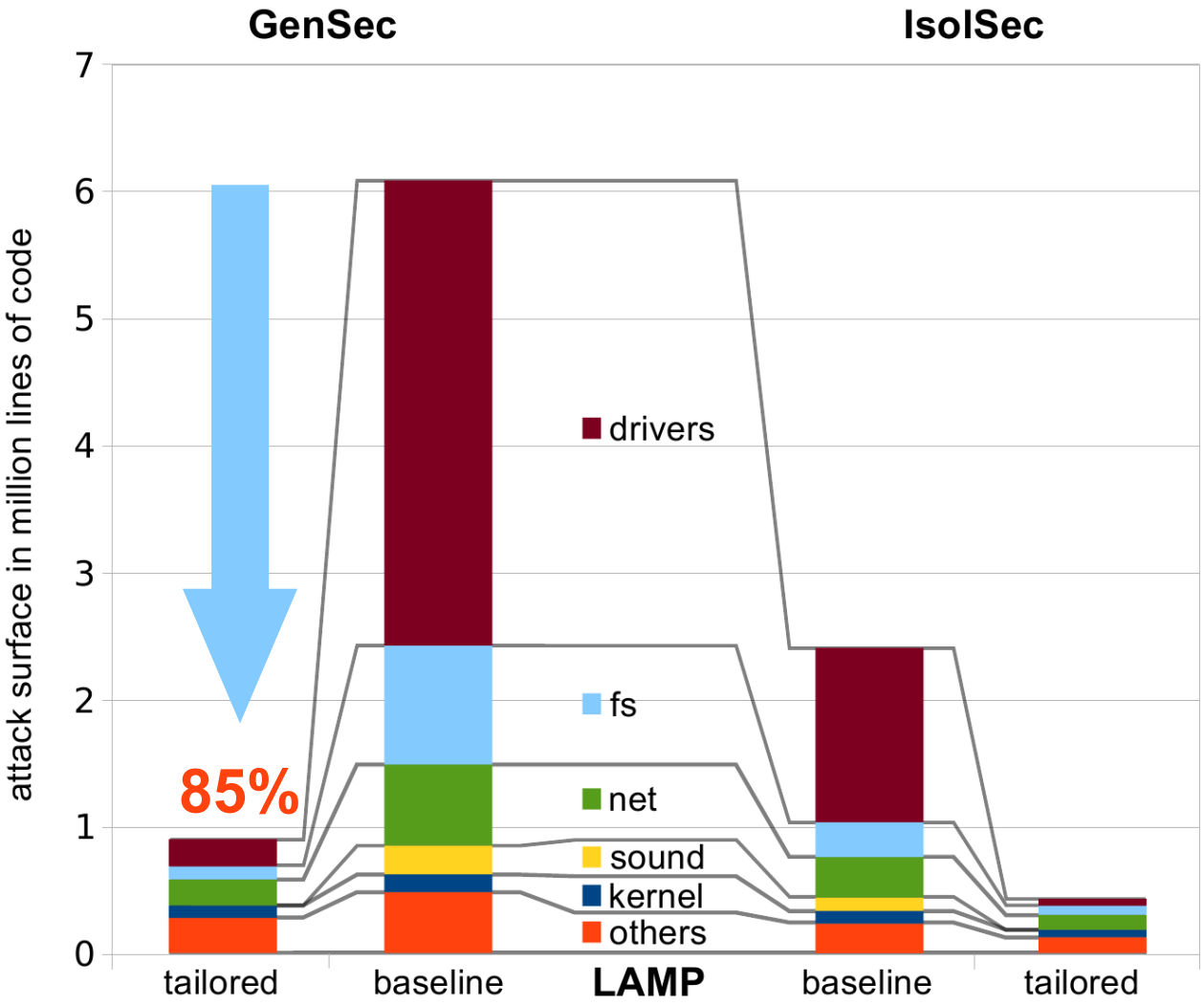
Results: attack surface reduction



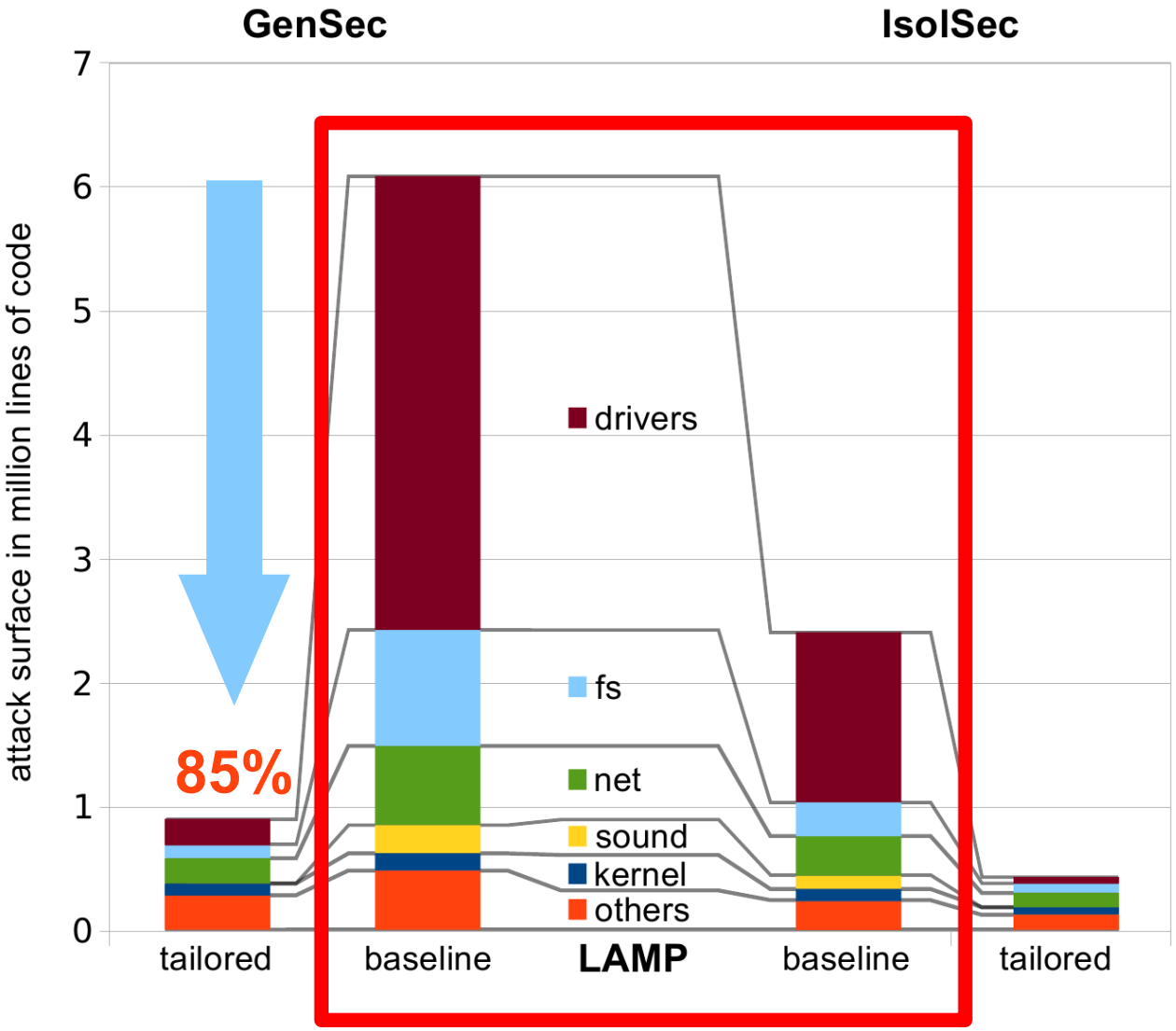
Results: attack surface reduction



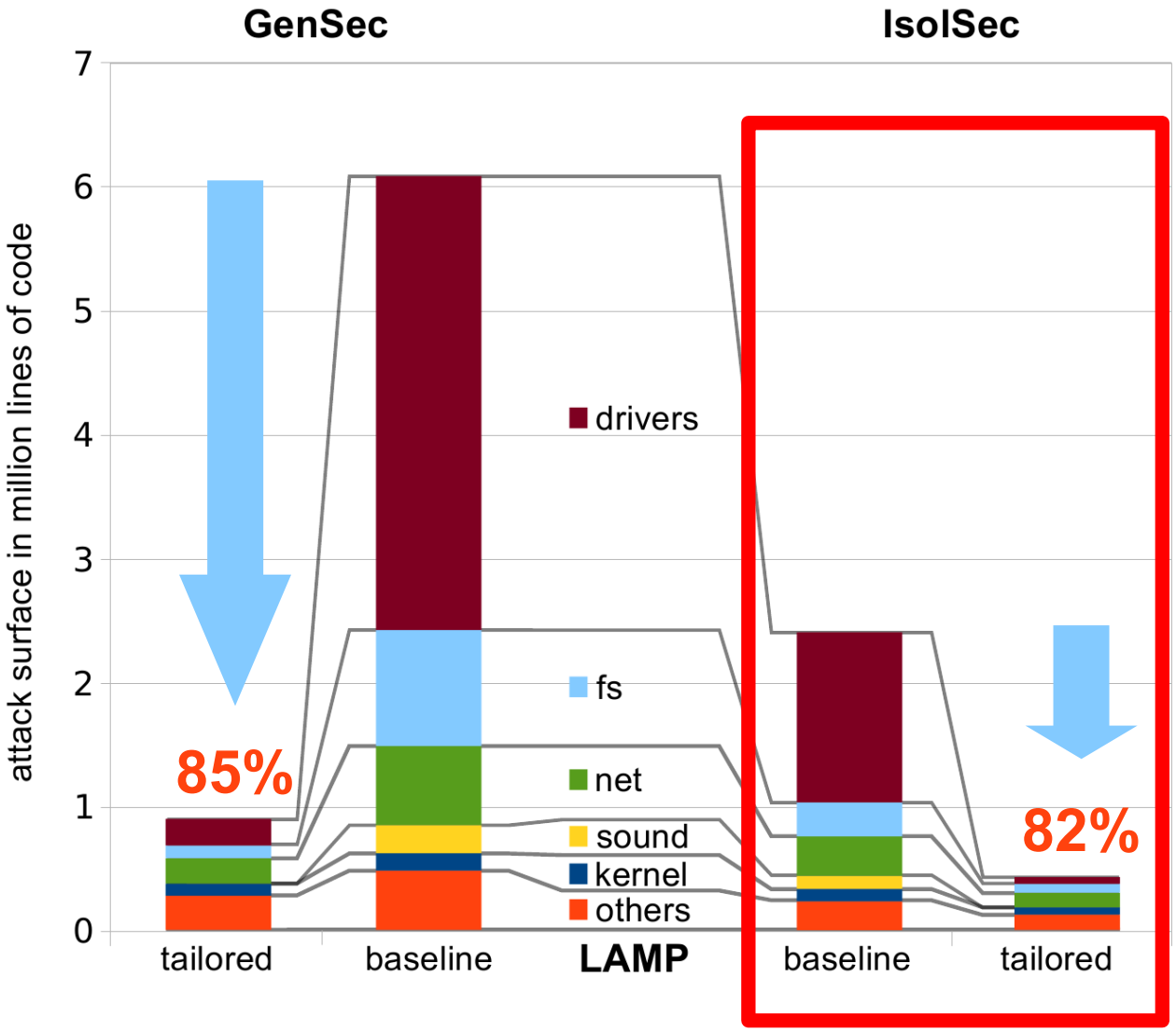
Results: attack surface reduction



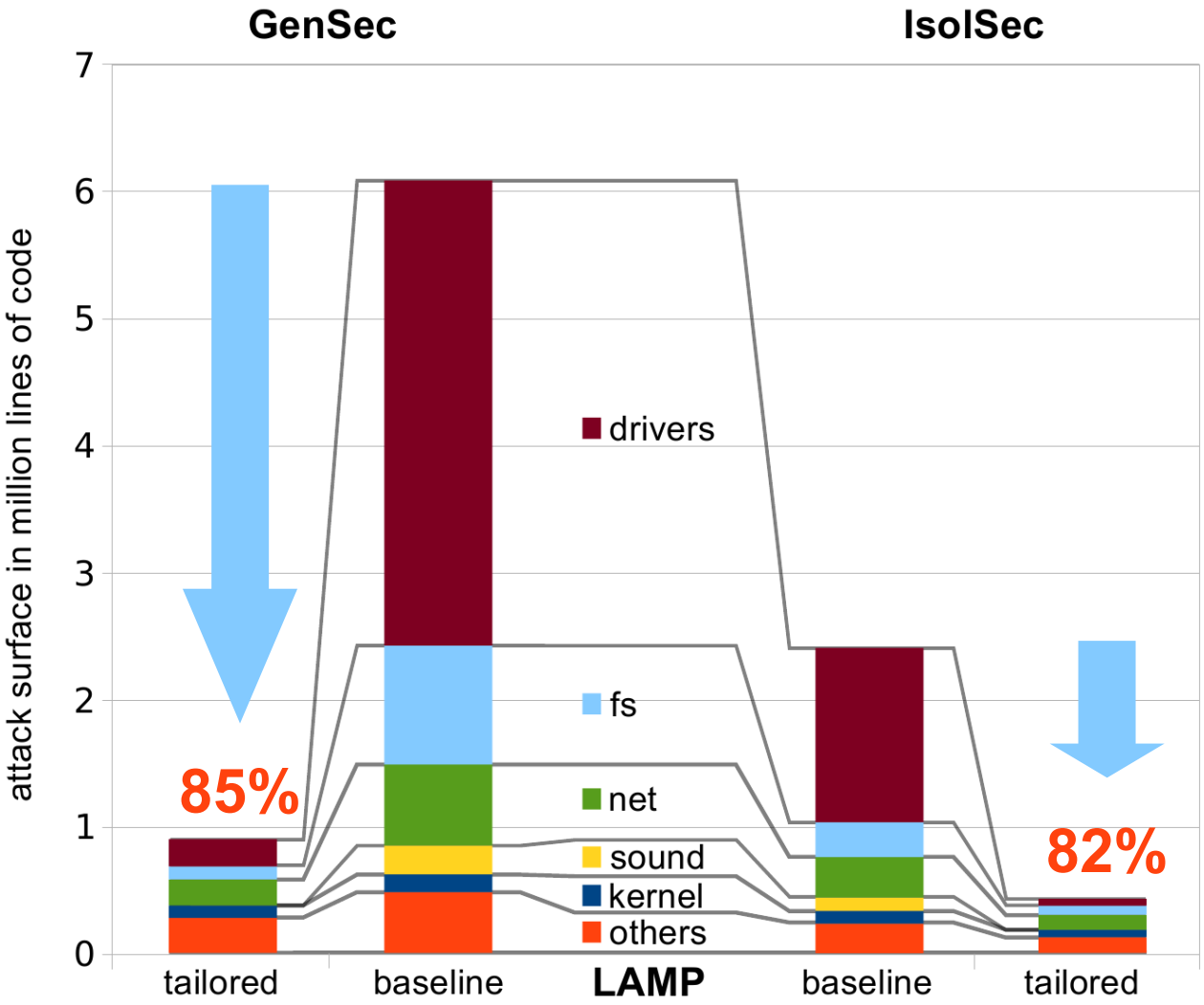
Results: attack surface reduction



Results: attack surface reduction



Results: attack surface reduction



Conclusion

- We presented a **framework for measuring attack surfaces**
  - Takes into account the security model, kernel configuration (e.g., unlike *sloccount*)
  - Designed for the Linux kernel, but may be adapted to other programs
  - Coherent results under different use cases, code-quality metrics, security models

- Kernel tailoring is a **proactive approach reducing kernel attack surface**
  - Effective: 80-85% attack surface reduction (SLOC), 50-65% in terms of CVEs
  - Assumes well-defined use cases (e.g., servers, embedded systems)
  - **Automated**, **no performance overhead**

- Source code available at:
  http://vamos.informatik.uni-erlangen.de/trac/undertaker/wiki/UndertakerTailor

- Future work:
  - Android and KVM use cases.
  - Implications in run-time attack surface reduction.

Questions?

		Baseline		Tailored		Reduction	
		LAMP	NFS	LAMP	NFS	LAMP	NFS
Kernel (vmlinux) size in Bytes			9,933,860	4,228,235	4,792,508	56%	52%
LKM total size in Bytes			62,987,539	2,139,642	2,648,034	97%	96%
Options set to 'y'			1,537	452	492	71%	68%
Options set to 'm'			3,142	43	63	99%	98%
Compiled source files			8,670	1,121	1,423	87%	84%
GENSEC	Call graph nodes		230,916	34,880	47,130	85%	80%
	Call graph arcs		1,033,113	132,030	178,523	87%	83%
	$AS1_{SLOC}$		6,080,858	895,513	1,122,545	85%	82%
	$AS1_{cycl}$		1,268,551	209,002	260,189	84%	79%
	$AS1_{CVE}$		848	338	429	60%	49%
	$AS2_{SLOC}$		58,353,938,861	11,067,605,244	11,578,373,245	81%	80%
	$AS2_{cycl}$		2,721,526,295	1,005,337,180	1,036,833,959	63%	62%
	$AS2_{CVE}$		20,023	7,697	9,512	62%	52%
ISOLSEC	Call graph nodes	92,244	96,064	15,575	21,561	83%	78%
	Call graph arcs	443,296	462,433	64,517	89,175	85%	81%
	$AS1_{SLOC}$	2,403,022	2,465,202	425,361	550,669	82%	78%
	$AS1_{cycl}$	504,019	518,823	99,674	126,710	80%	76%
	$AS1_{CVE}$	485	524	203	276	57%	47%
	$AS2_{SLOC}$	15,753,006,783	15,883,981,161	4,457,696,135	4,770,441,587	72%	70%
	$AS2_{cycl}$	918,429,105	929,197,559	374,455,910	391,855,241	59%	57%
	$AS2_{CVE}$	10,151	11,127	4,287	5,489	57%	51%

Backup

Comparison to kernel extension fault isolation

	Ideal LKM isolation	Kernel Tailoring		Both combined	
		LAMP	Workstation/NFS	LAMP	Workstation/NFS
$AS1_{SLOC}$	2,064,526	425,361	550,669	420,373	489,732
$AS1_{cycl}$	444,775	99,674	126,710	98,534	113,735
$AS1_{CVE}$	390	203	276	203	240
$AS2_{SLOC}$	11,826,476,219	4,457,696,135	4,770,441,587	4,452,329,879	4,663,745,009
$AS2_{cycl}$	851,676,457	374,455,910	391,855,241	374,214,950	386,472,434
$AS2_{CVE}$	7,725	4,287	5,489	4,287	4,849