Perspective: A Principled Framework for Pliable and Secure Speculation in Operating Systems

Tae Hoon Kim, David Rudo, Kaiyang Zhao, Zirui Neil Zhao†, Dimitrios Skarlatos

Carnegie Mellon University †University of Illinois Urbana-Champaign

†The University of Texas at Austin (New Affil.)

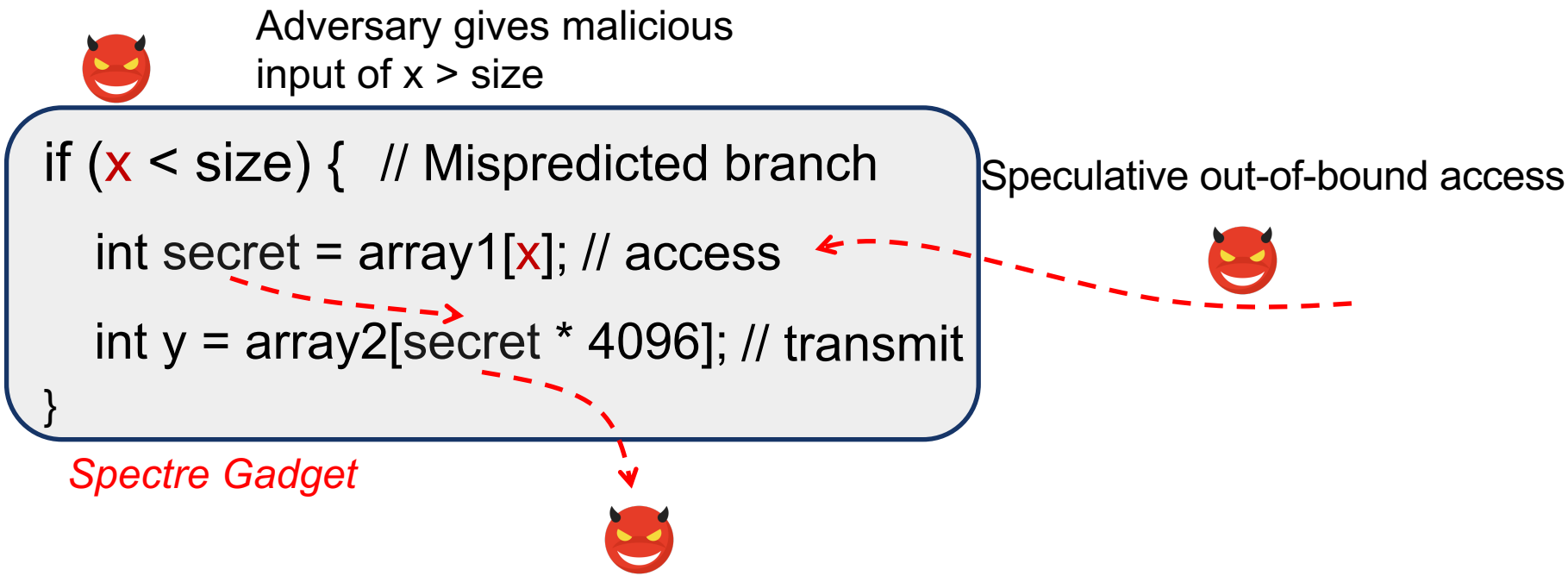
ISCA '24 – Session 6B: Security

Carnegie Mellon University Computer Science Department

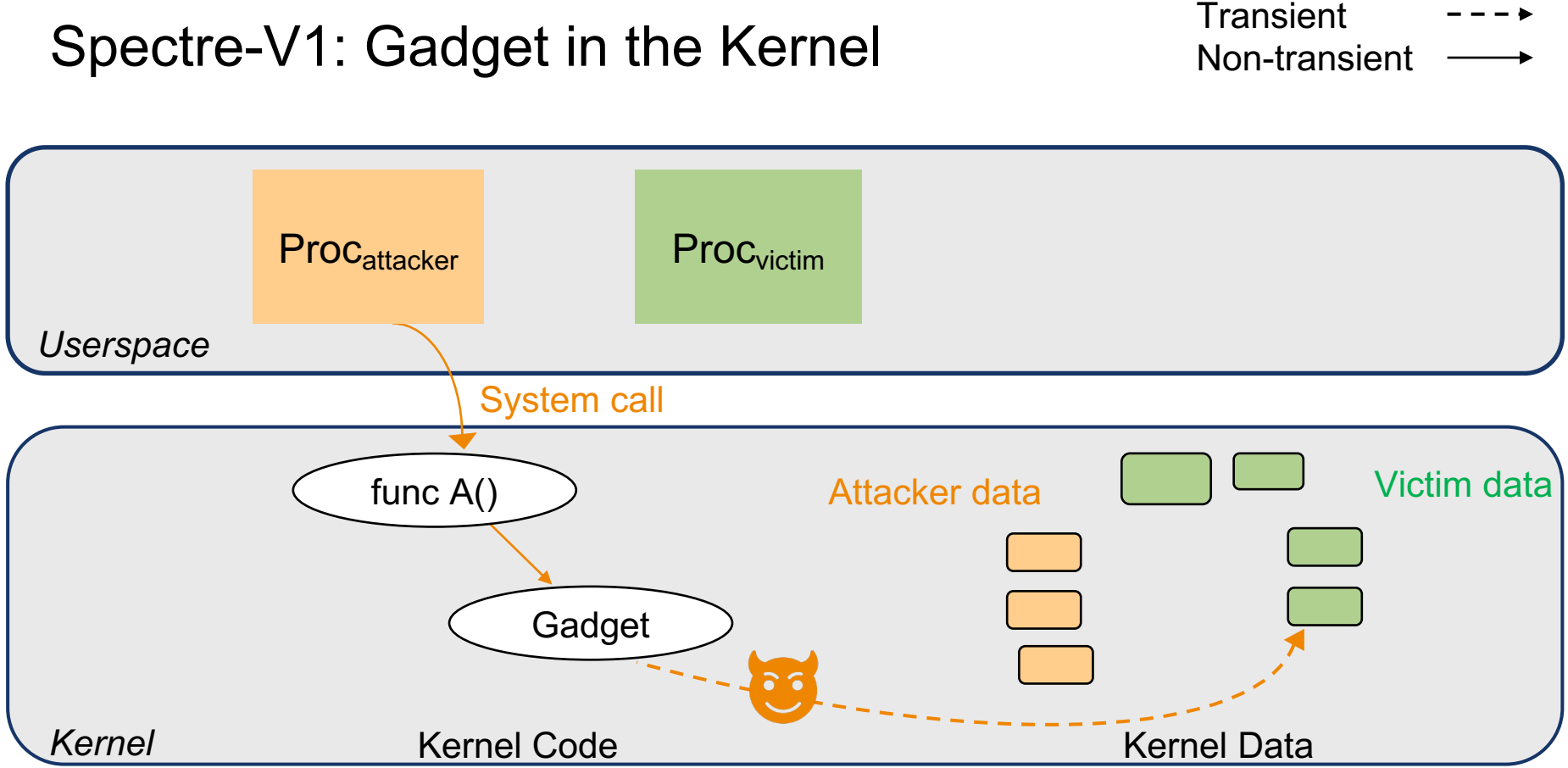
ILLINOIS

Computer Architecture Operating Systems Group

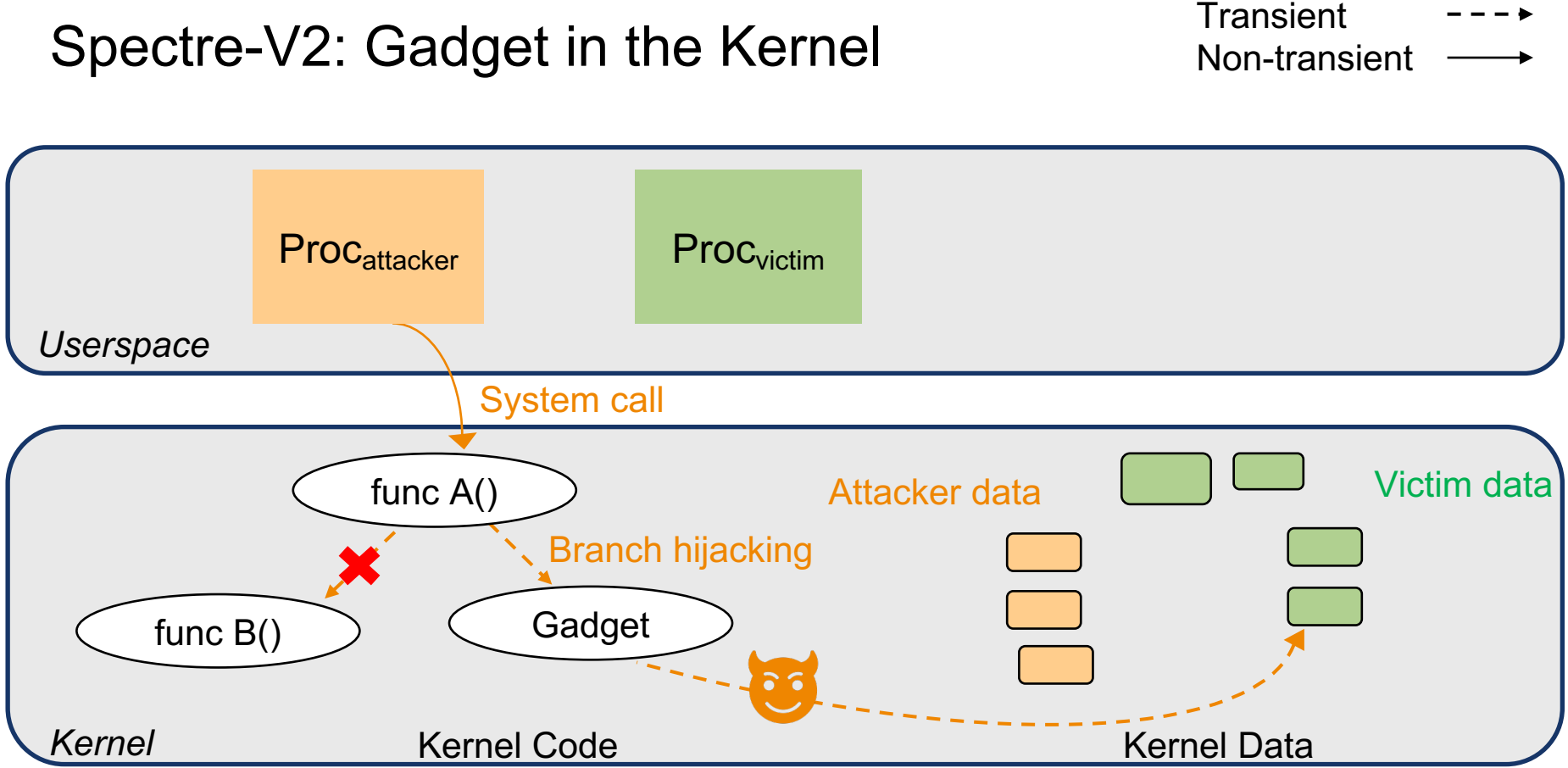
Spectre-V1: A Quick Recap

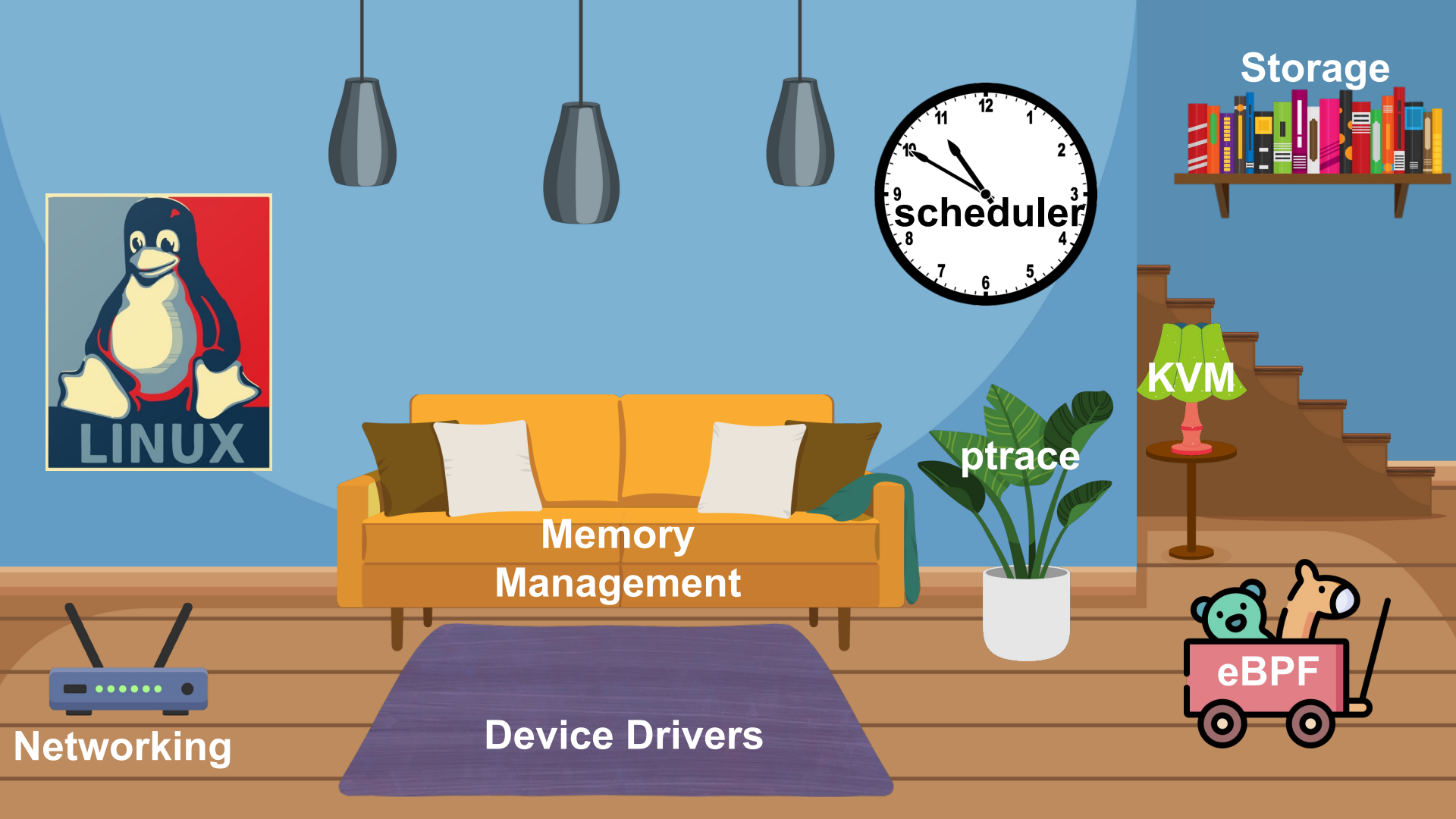


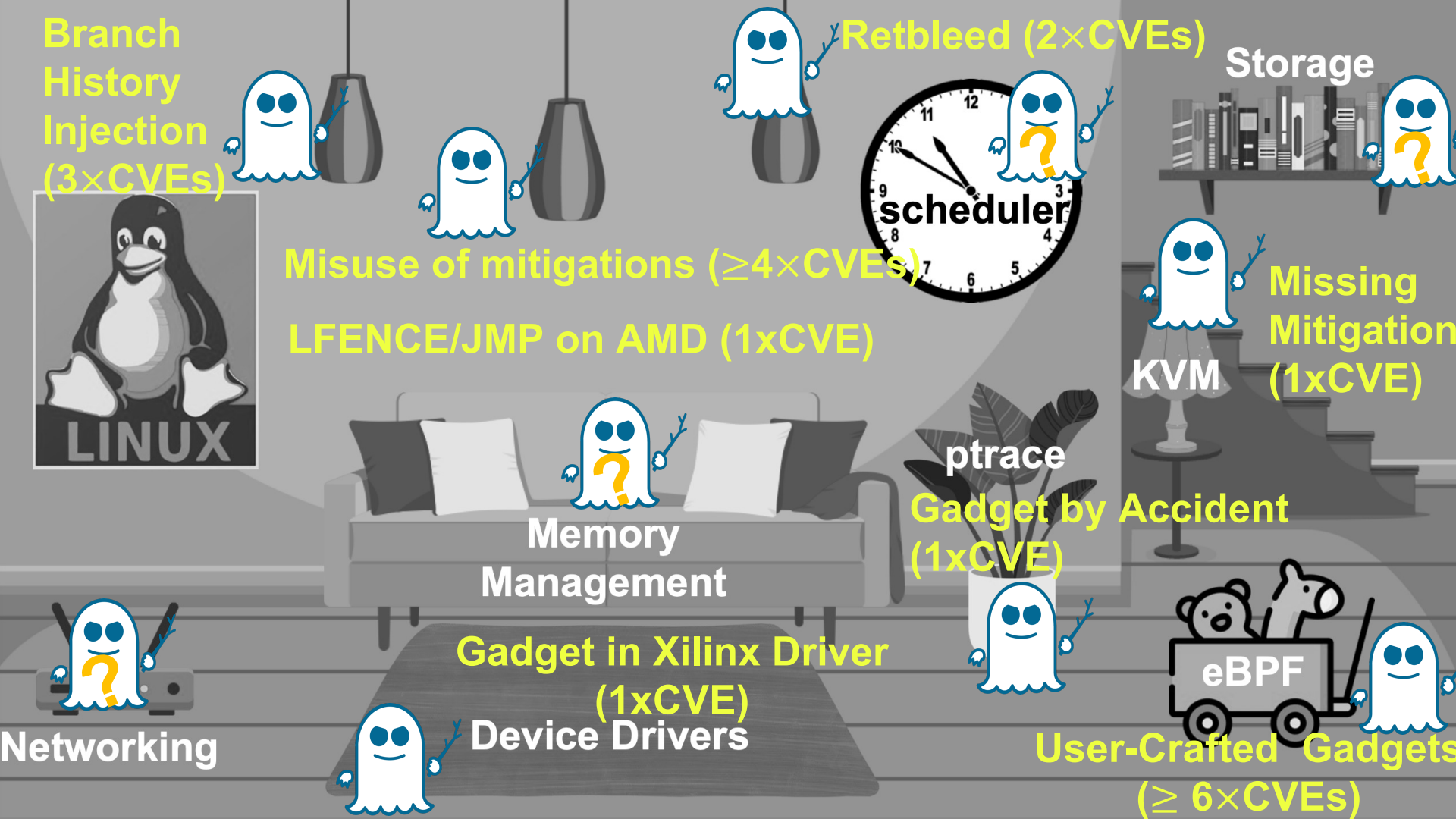
Spectre-V1: Gadget in the Kernel



Spectre-V2: Gadget in the Kernel




Storage
scheduler
LINUX
KVM
ptrace
Memory
Management
eBPF
Networking
Device Drivers


Branch History Injection (3×CVEs)
Retbleed (2×CVEs)
Storage
scheduler
Misuse of mitigations (≥4×CVEs)
LFENCE/JMP on AMD (1xCVE)
LINUX
KVM
Missing Mitigation (1xCVE)
ptrace
Gadget by Accident (1xCVE)
Memory Management
Gadget in Xilinx Driver (1xCVE)
Networking
Device Drivers
eBPF
User-Crafted Gadgets (≥ 6×CVEs)

Existing Defenses are Software Agnostic

```
if (x < size) {   // Unresolved branch
    int secret = array1[x]; // access
    int y = array2[secret * 4096]; // transmit
}
```

Block or protect speculative accesses (SLH, Fence, InvisiSpec, DOM, …)

Block speculative transmissions (STT, NDA, …)

Existing Defenses → Complex HW + Over-Protection

Existing Defenses are software agnostic
But these solutions introduce complex hardware ☹
Over-protect with high performance overhead

```
if (x < size) {   // Unresolved branch
    int secret = array1[x]; // access
    int y = array2[secret * 4096]; // transmit
}
```

Block or protect speculative accesses (SLH, Fence, InvisiSpec, DOM, …)

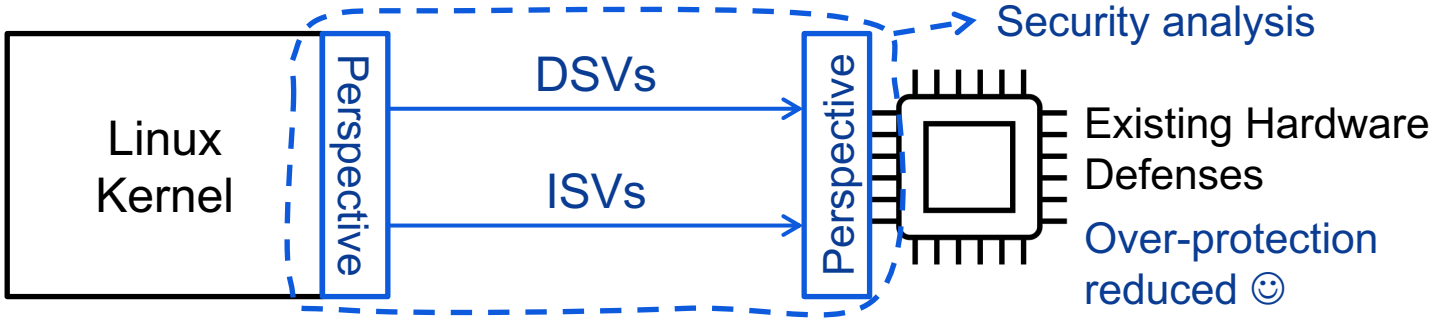
Block speculative transmissions (STT, NDA, …)

Perspective: An OS-HW Interface for Secure Speculation in OS

- **A taxonomy of transient execution attacks in the kernel**

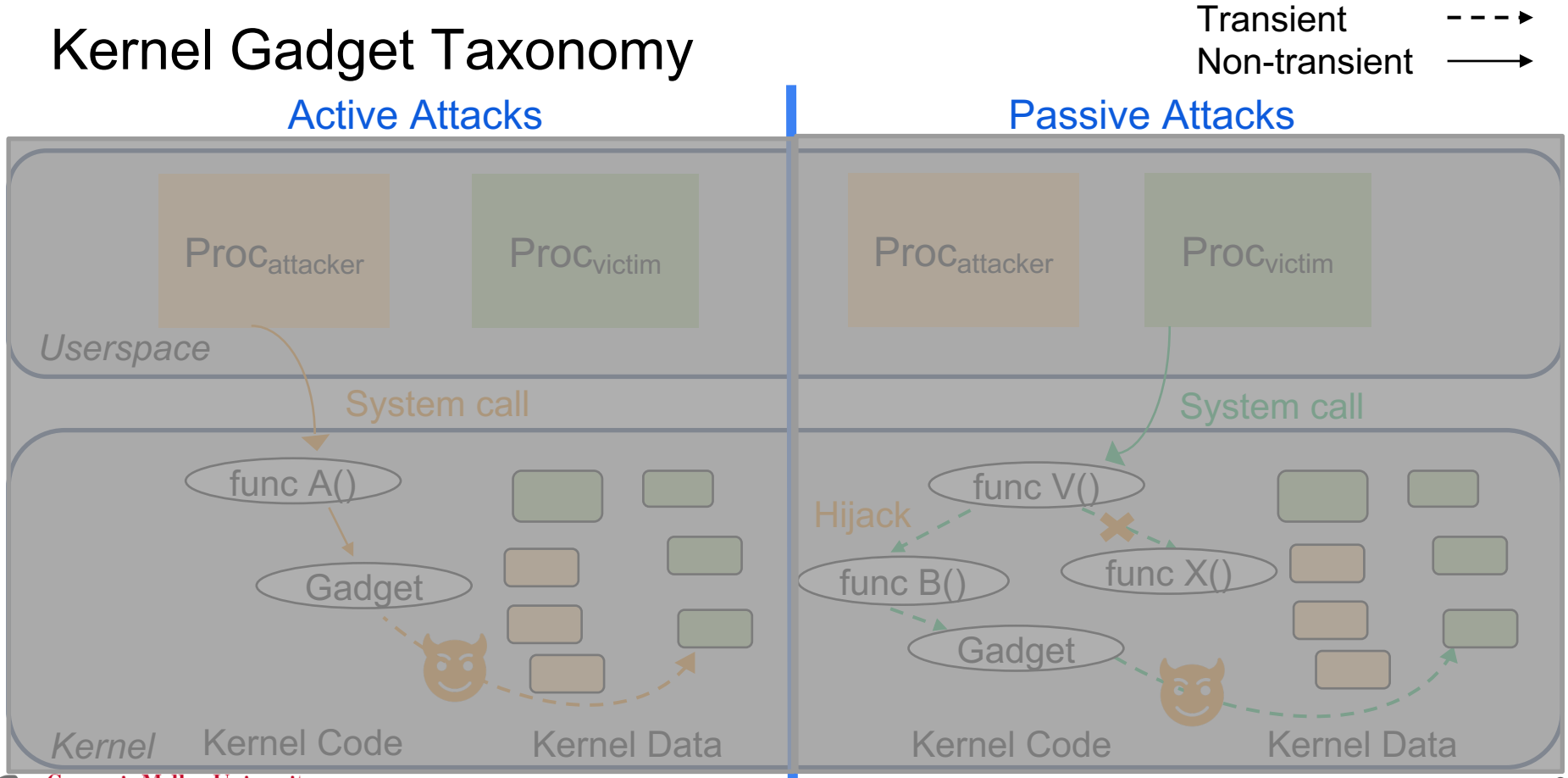
Active and passive attack scenarios ⇒ Agnostic to attack variants (e.g., Spectre v1, v2, …)

- **Perspective OS-HW interface ⇒ Tailored defense**
  - Data speculation views (DSVs): which *data* can be speculatively accessed
  - Instruction speculation views (ISVs): which *instructions* can speculatively execute
  - Both views can be associated with an execution context (e.g., a process)

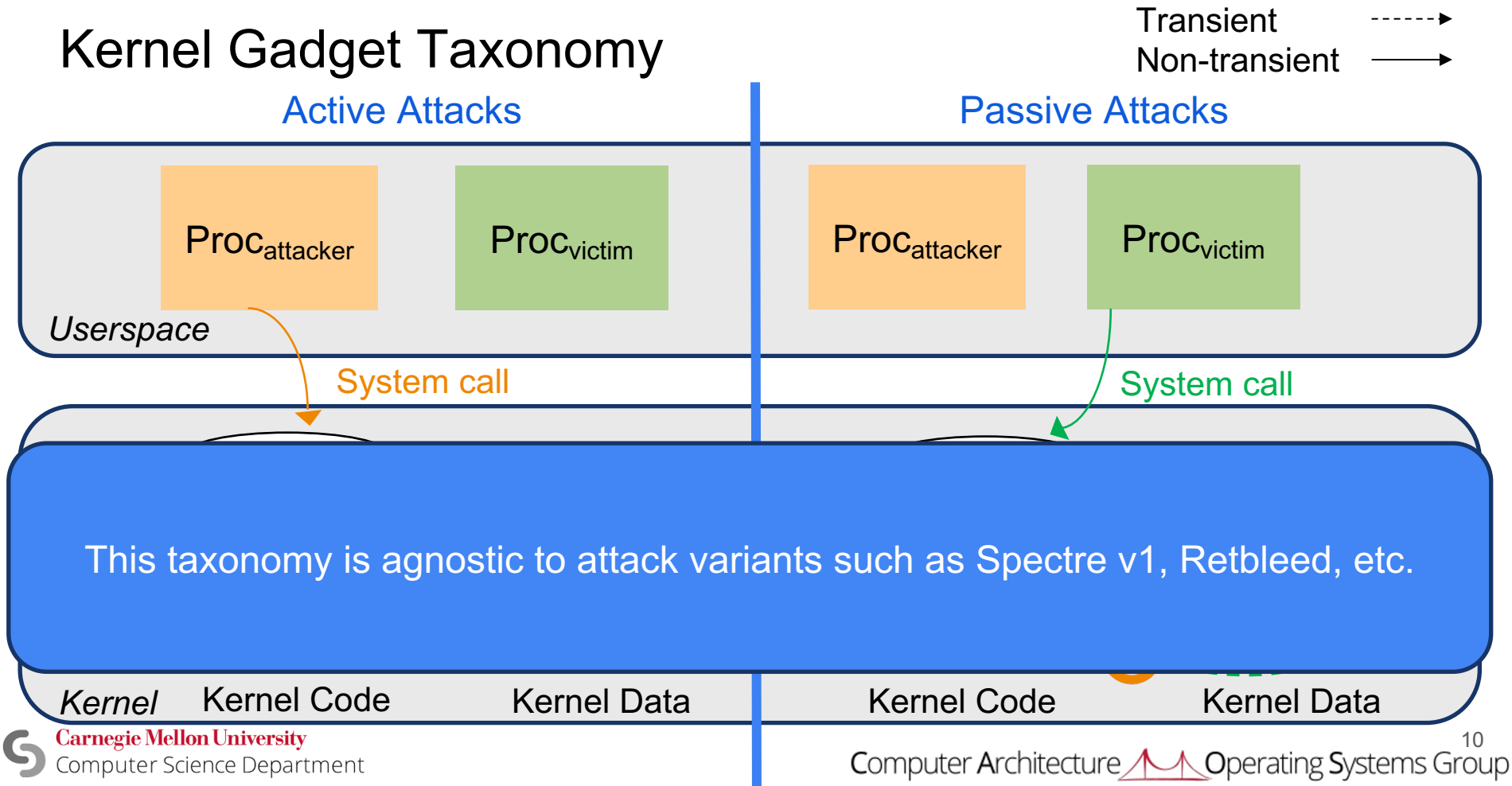


Perspective's **tailored protection** allows the use of **simple** hardware defenses like Fence while maintaining **a low execution overhead** (e.g., 1.2% on datacenter applications)

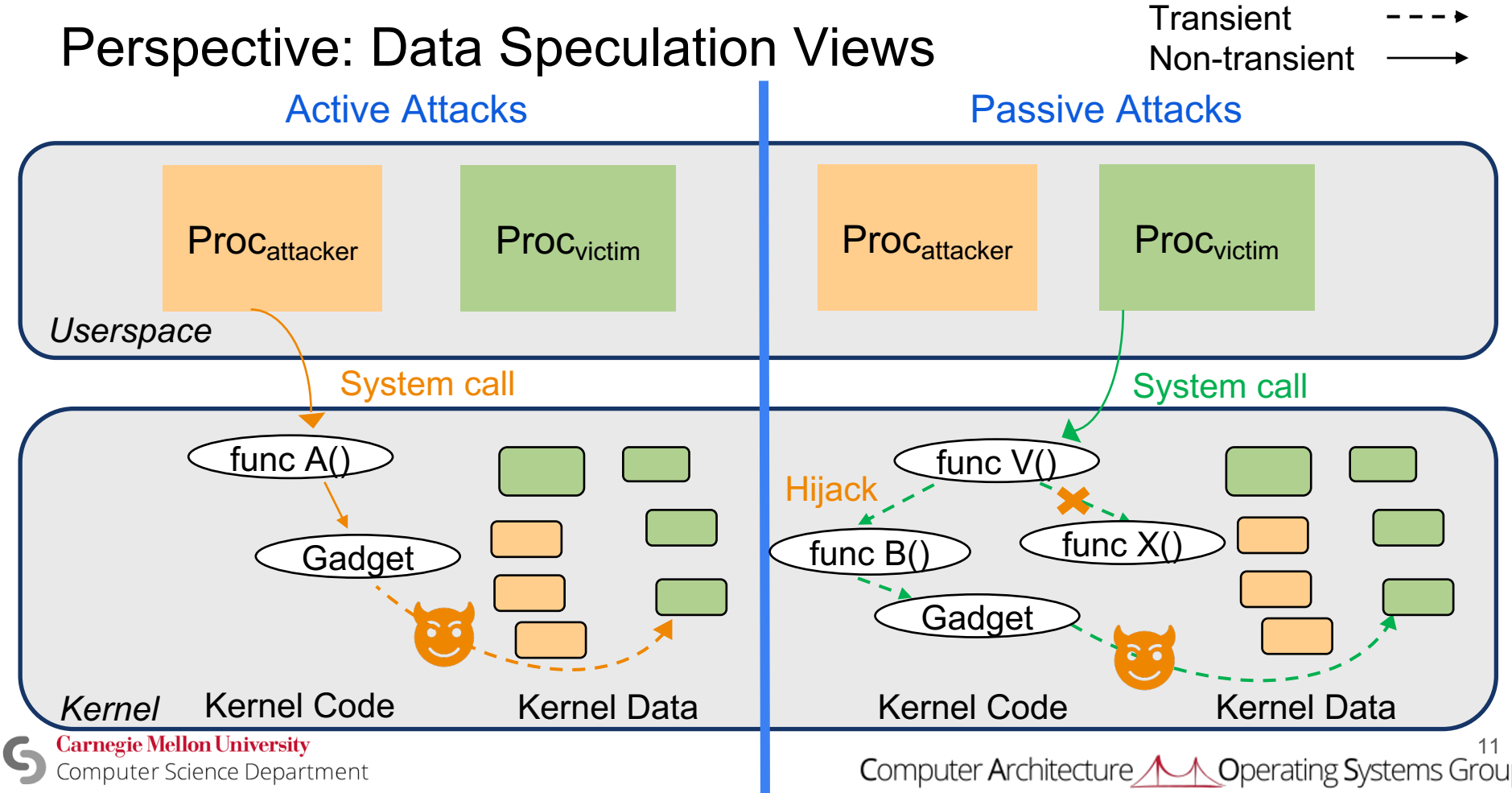
Kernel Gadget Taxonomy



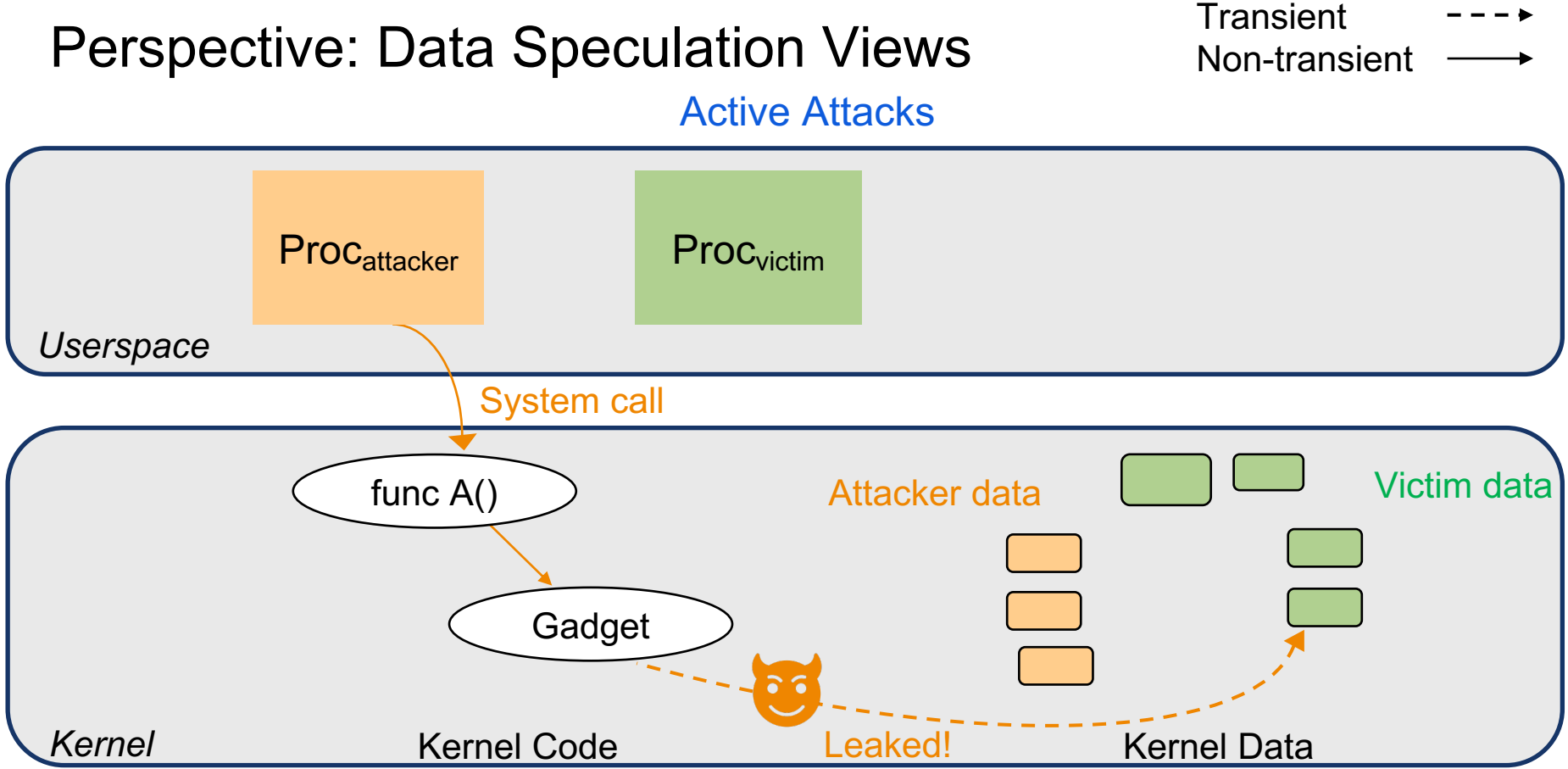
Kernel Gadget Taxonomy



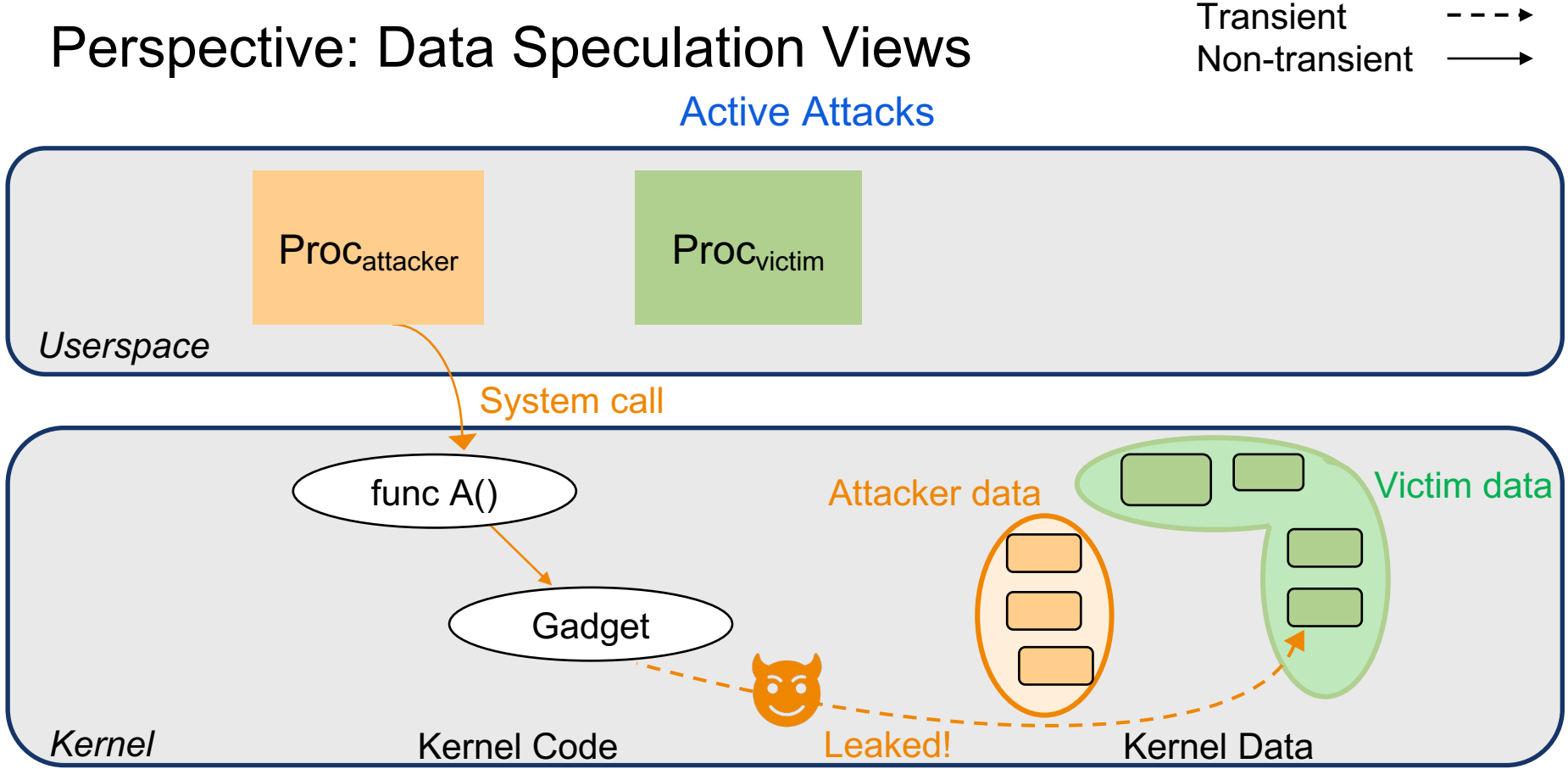
Perspective: Data Speculation Views



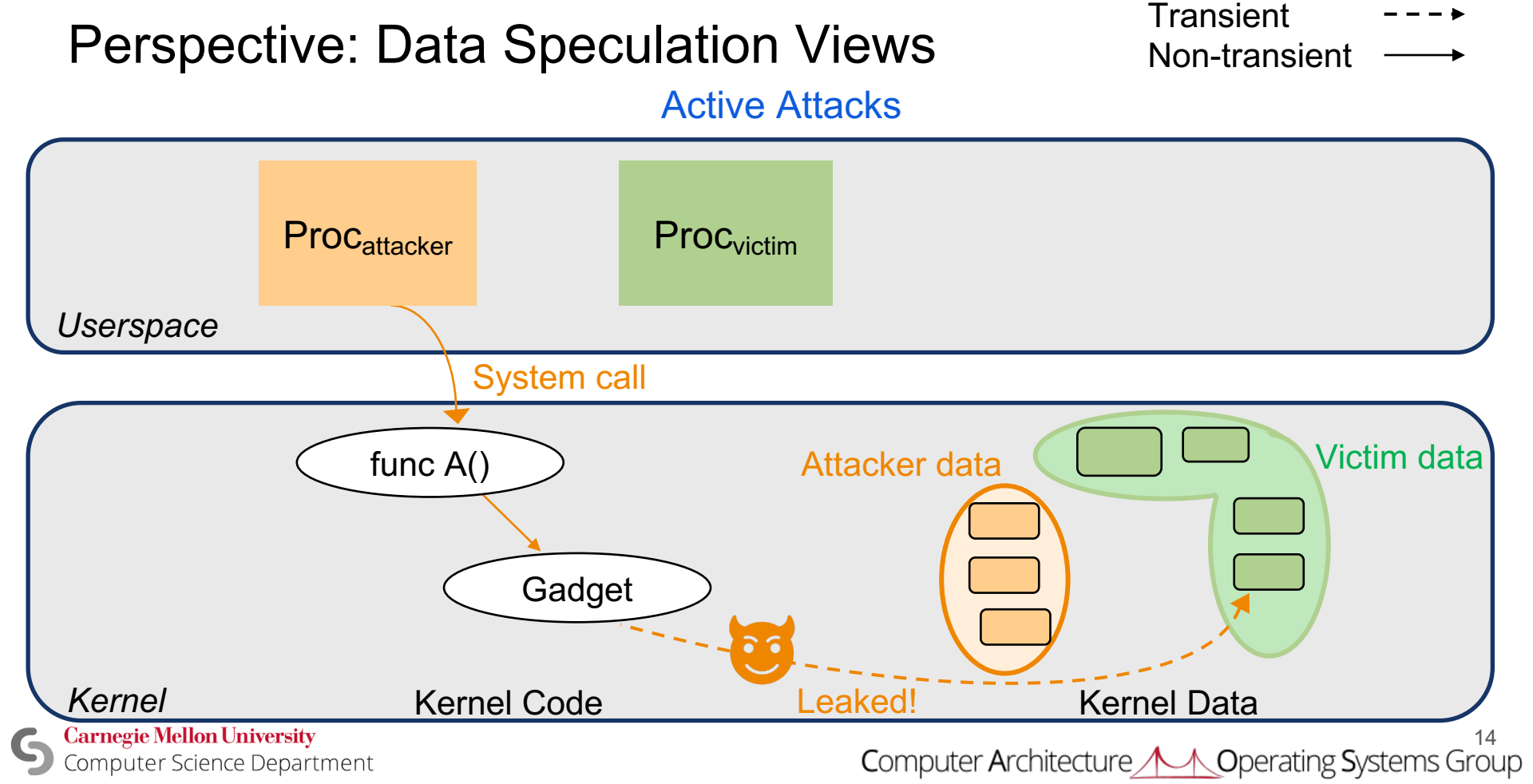
Perspective: Data Speculation Views



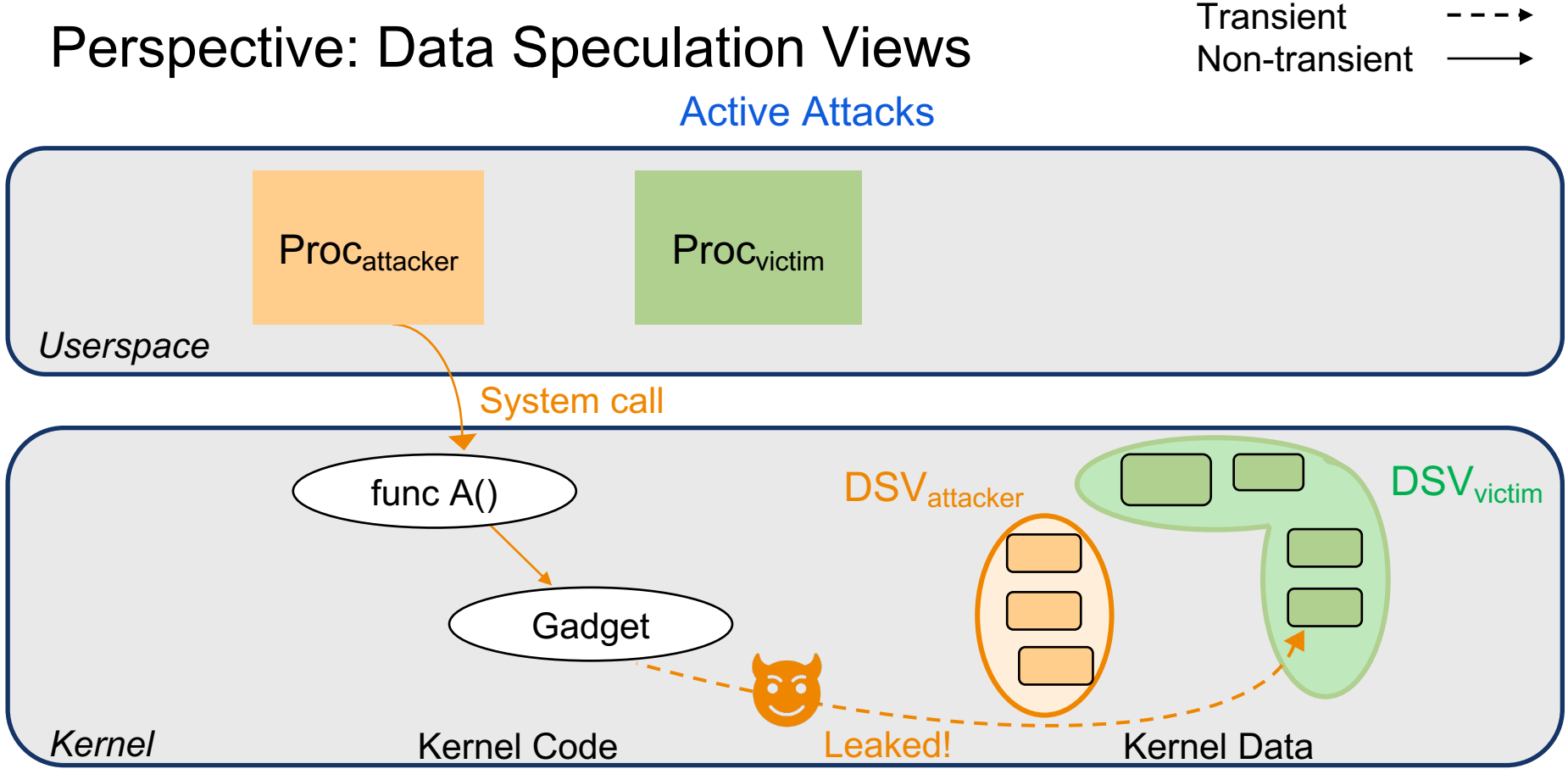
Perspective: Data Speculation Views



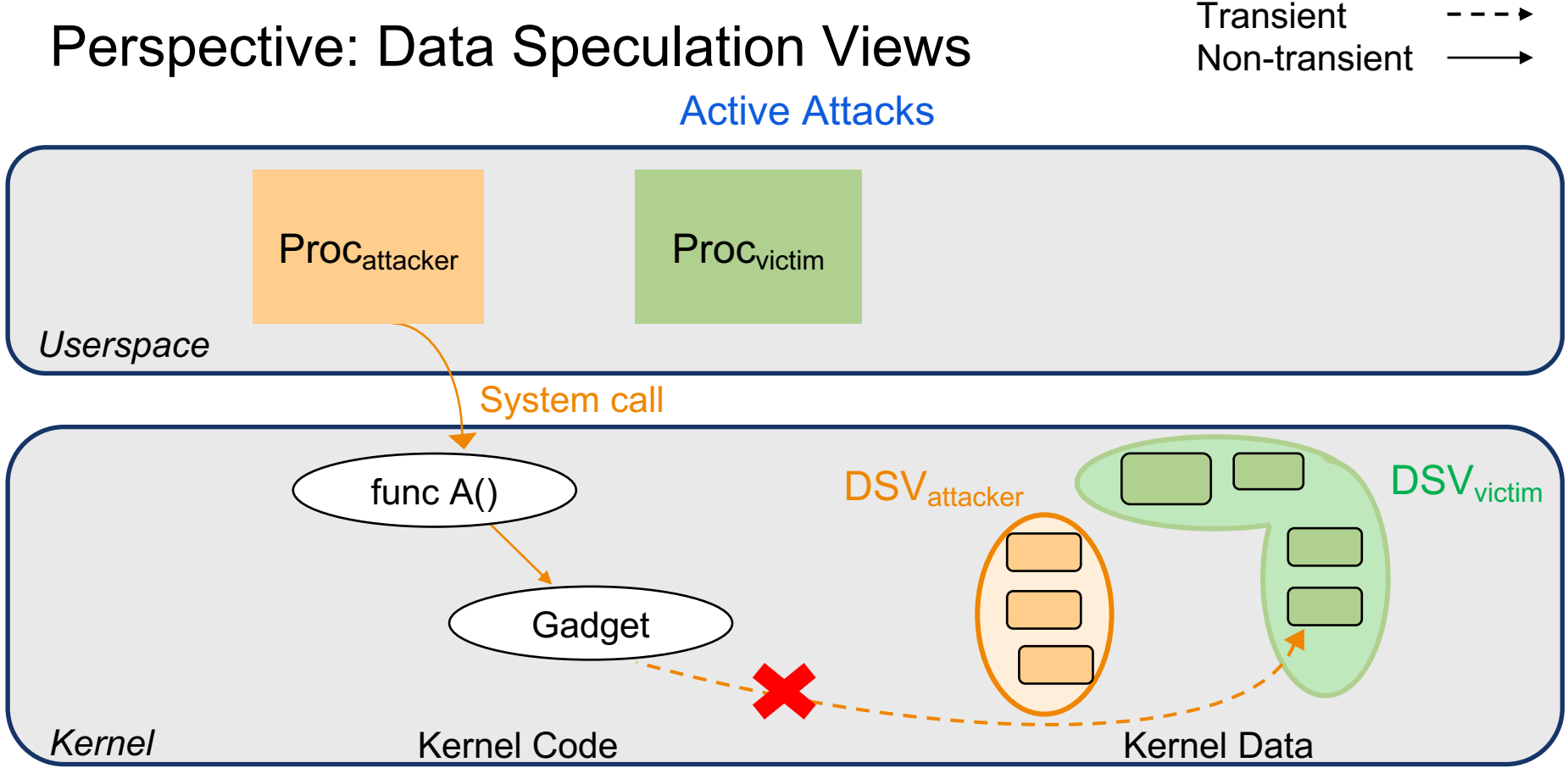
Perspective: Data Speculation Views



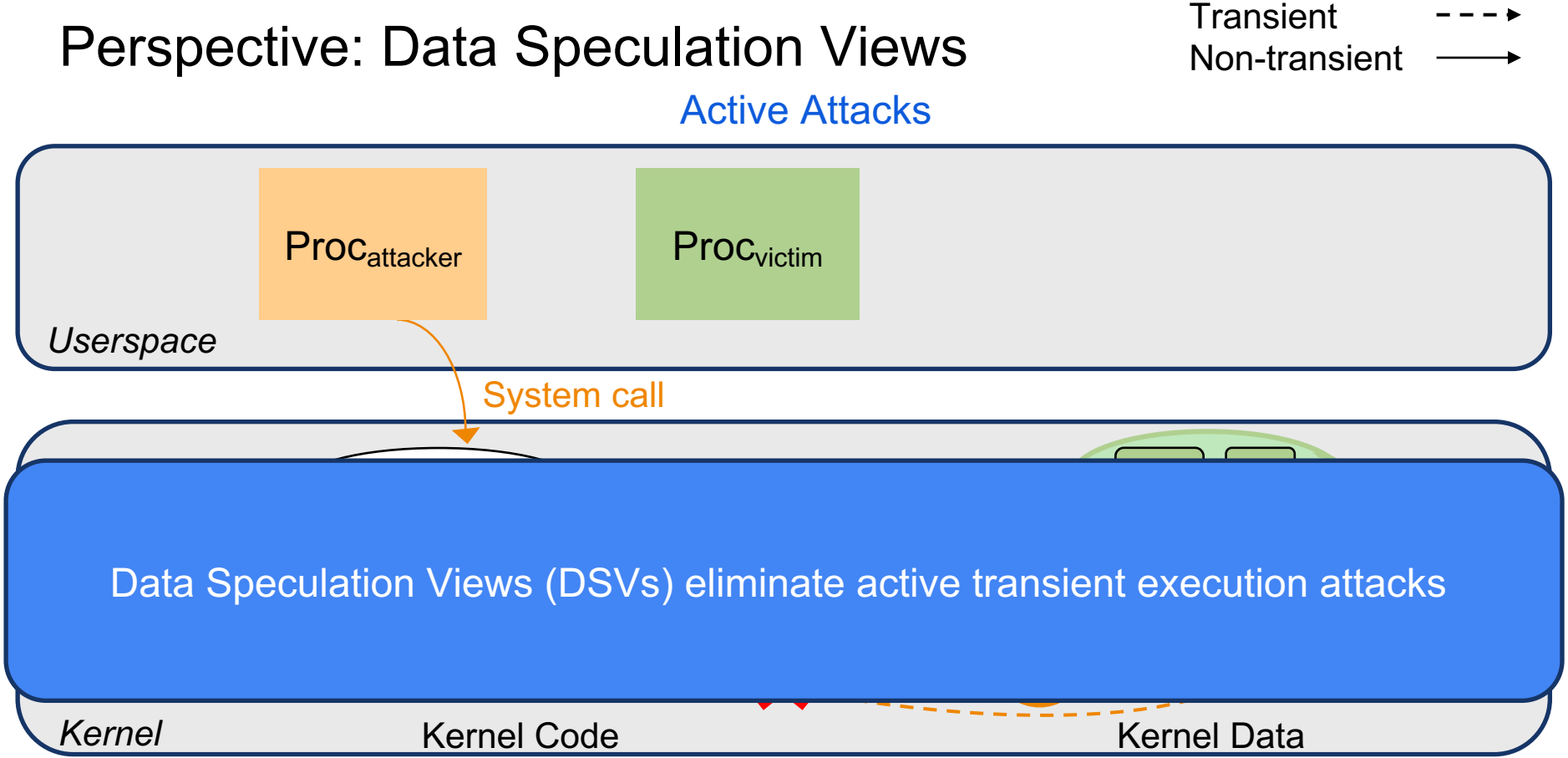
Perspective: Data Speculation Views



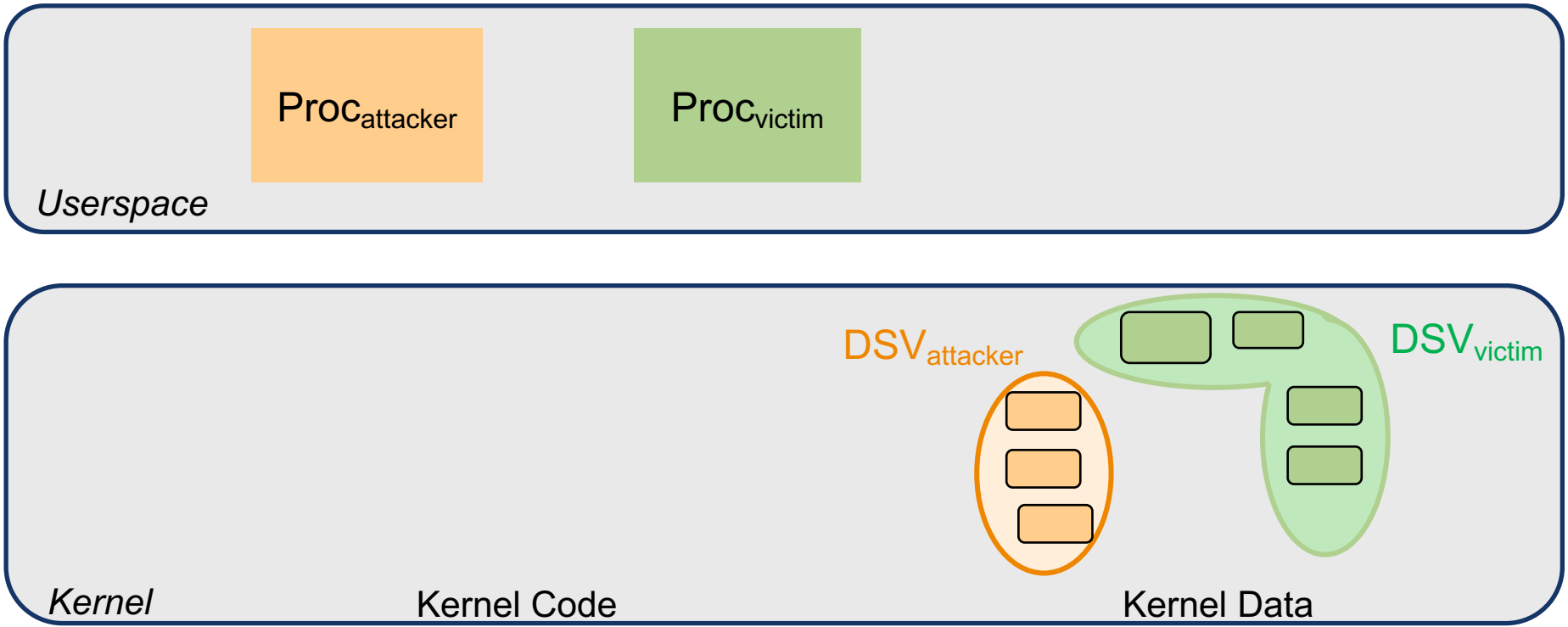
Perspective: Data Speculation Views



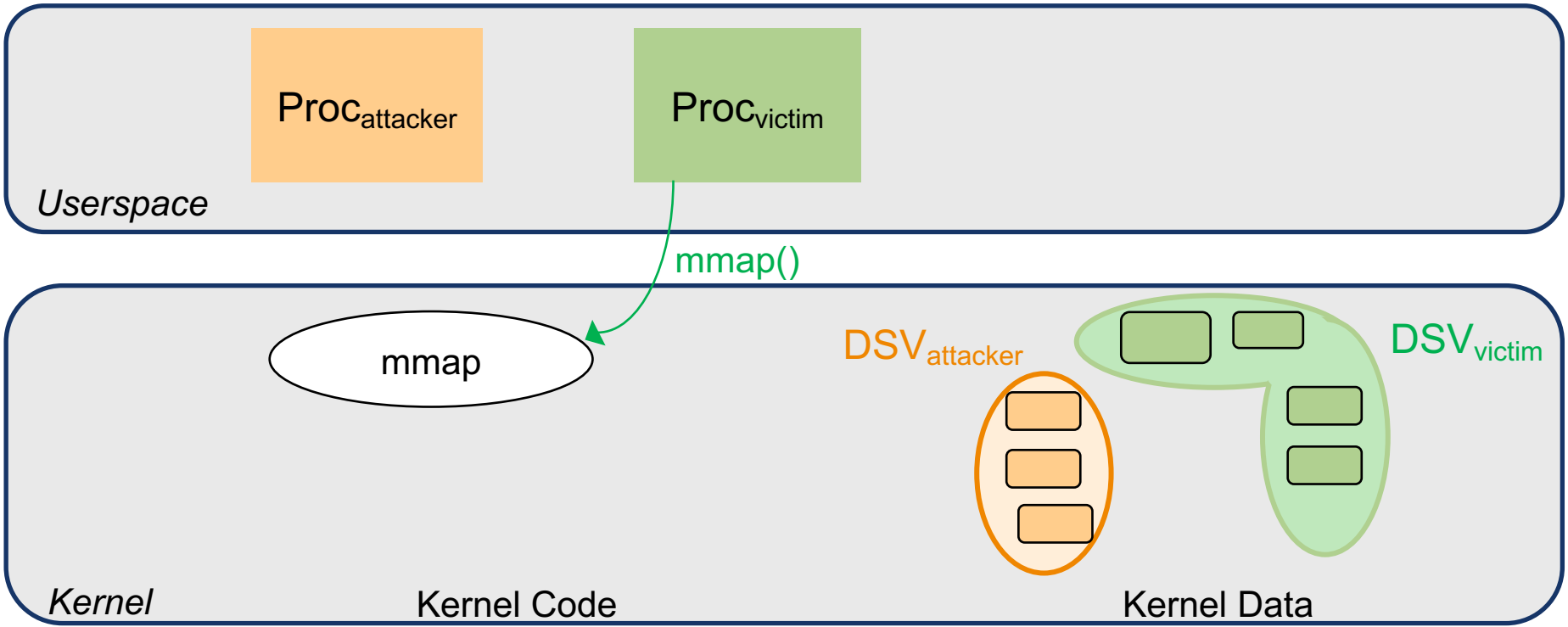
Perspective: Data Speculation Views



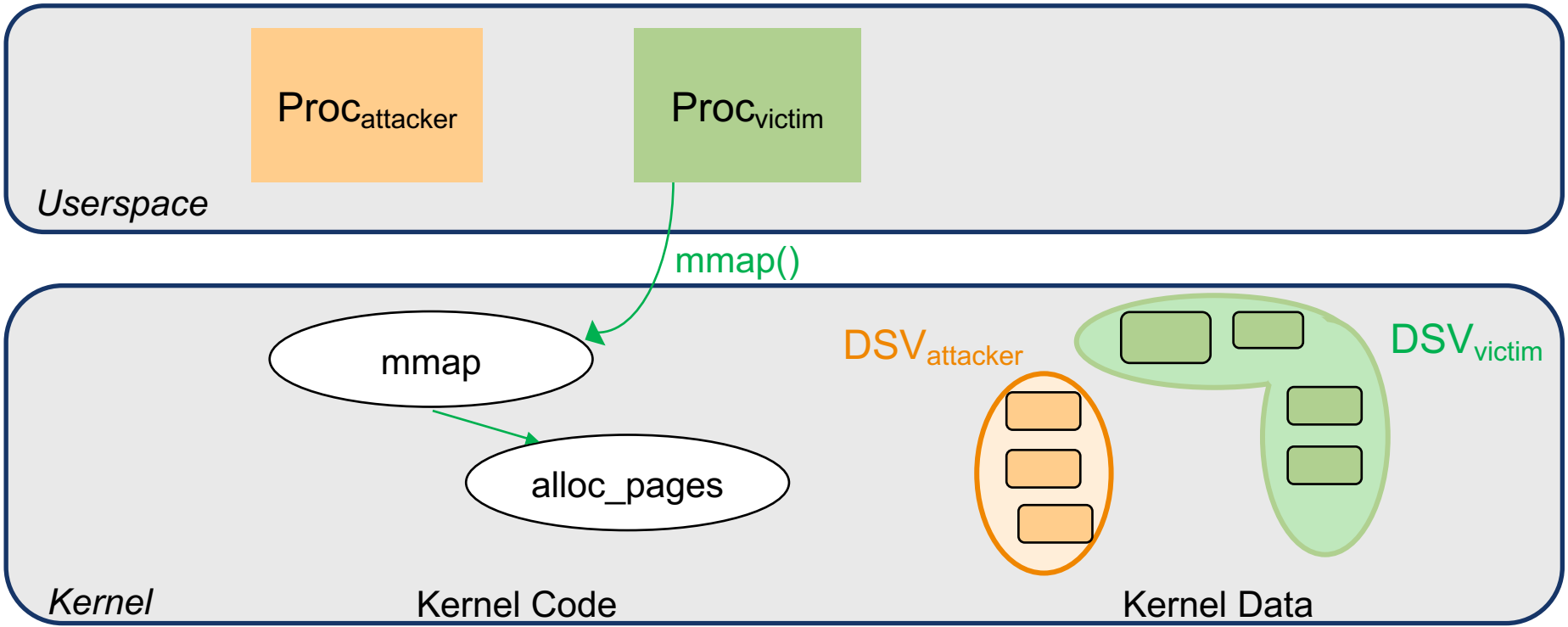
DSV Through Allocation



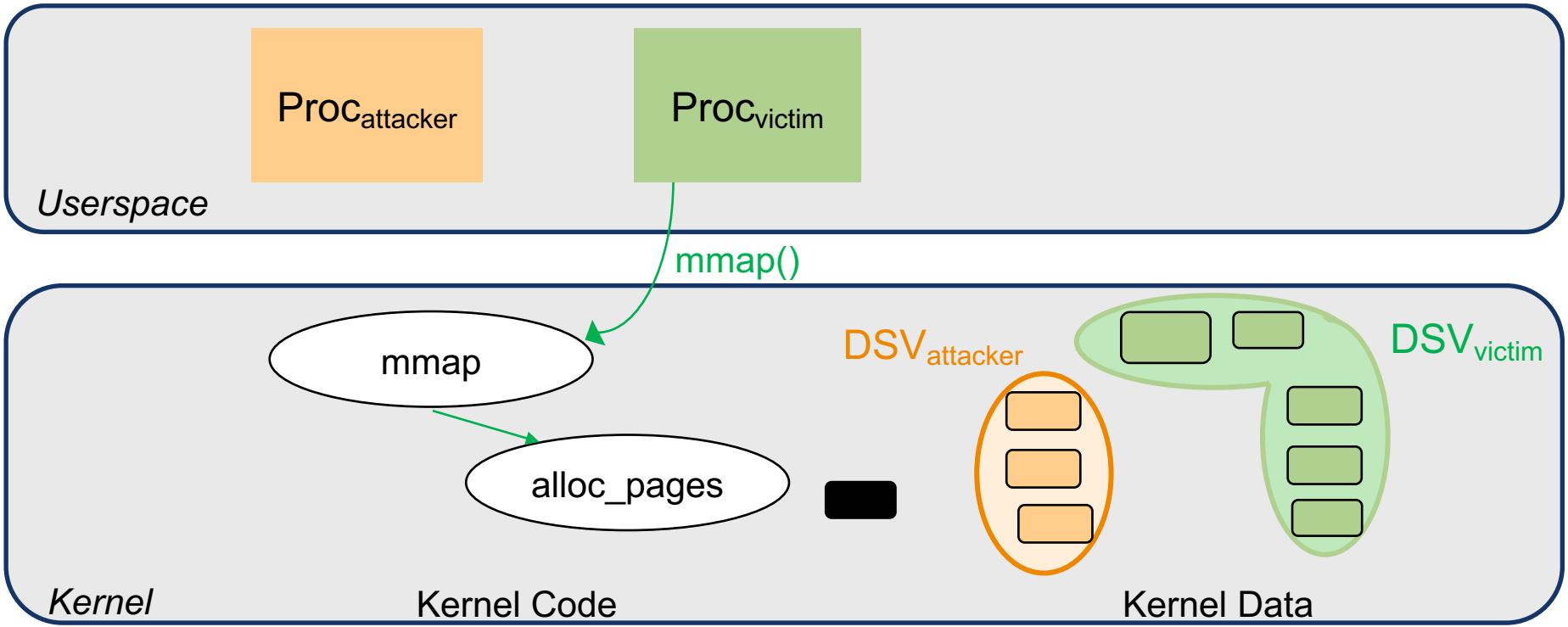
DSV Through Allocation



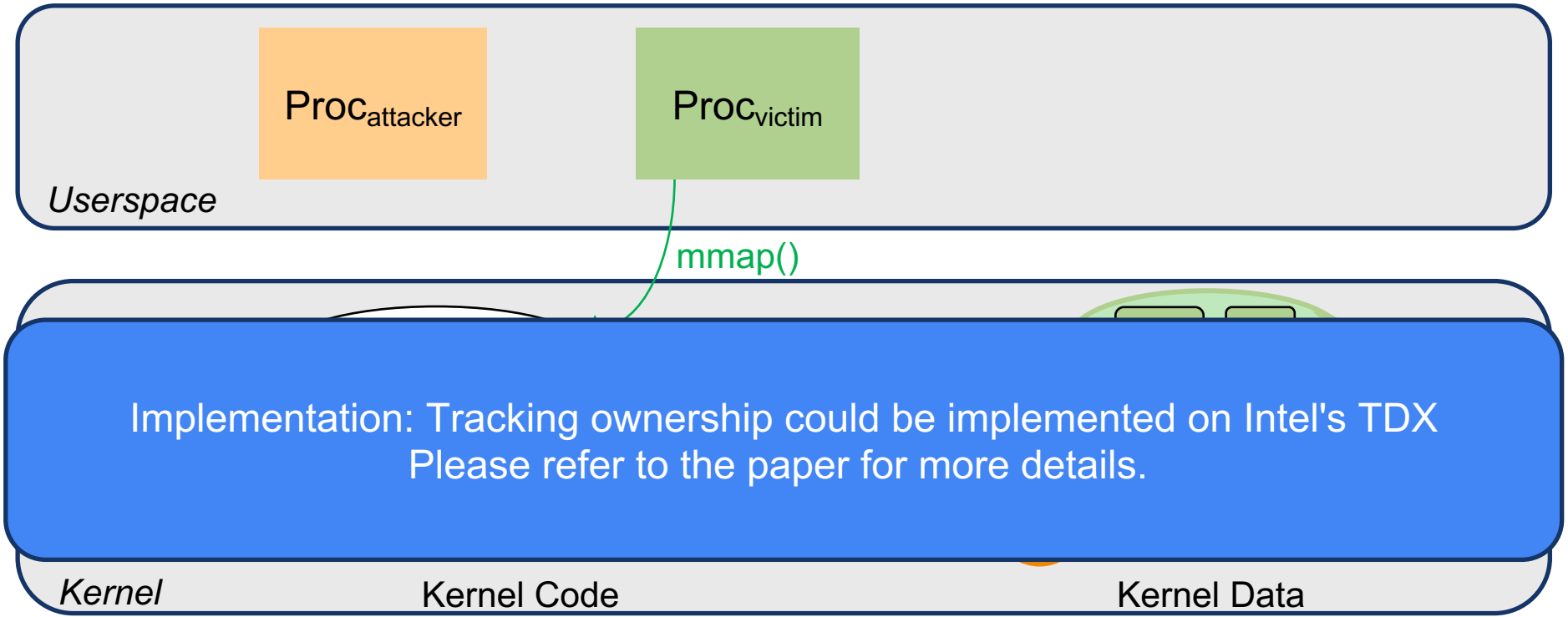
DSV Through Allocation



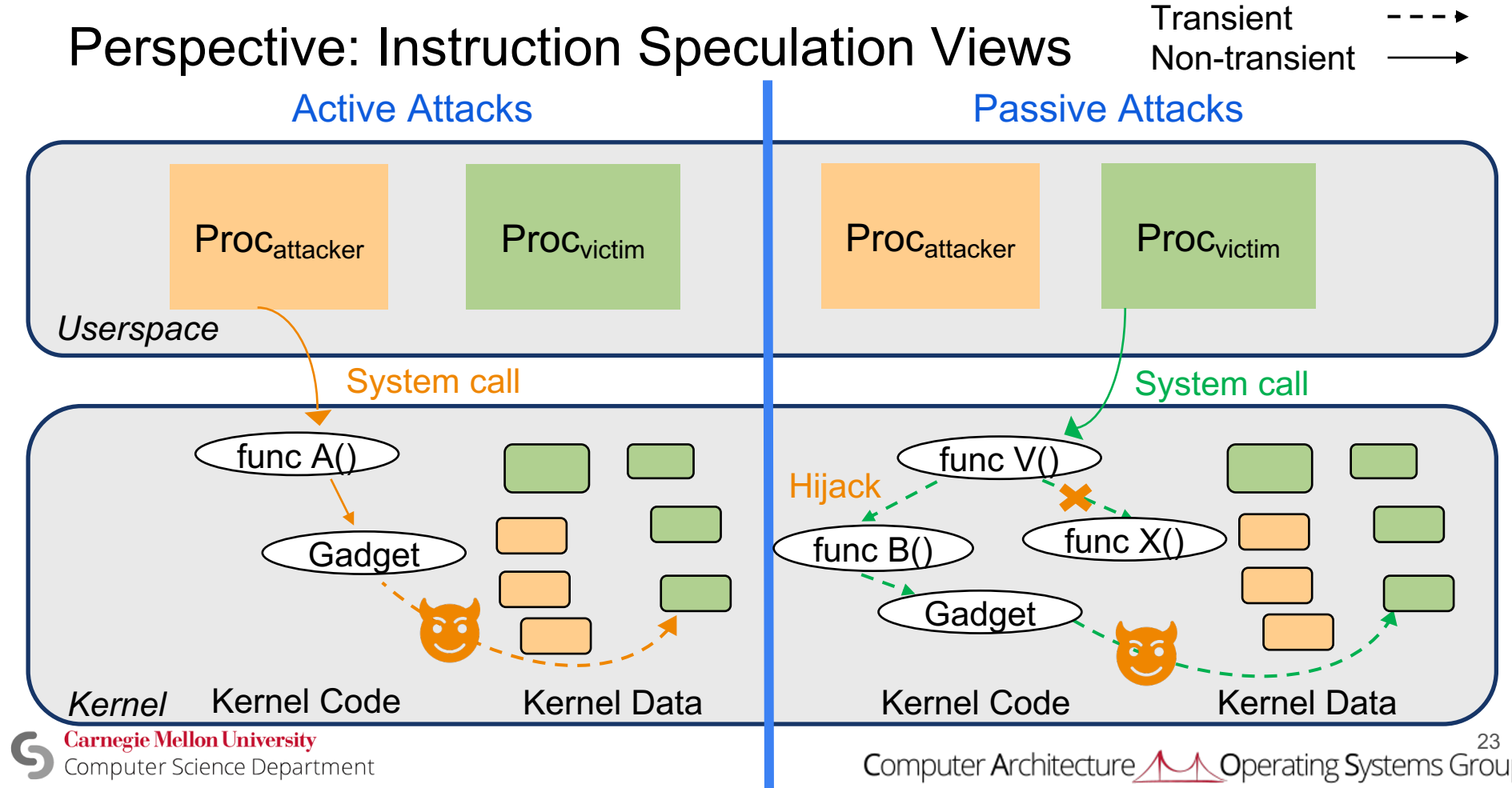
DSV Through Allocation



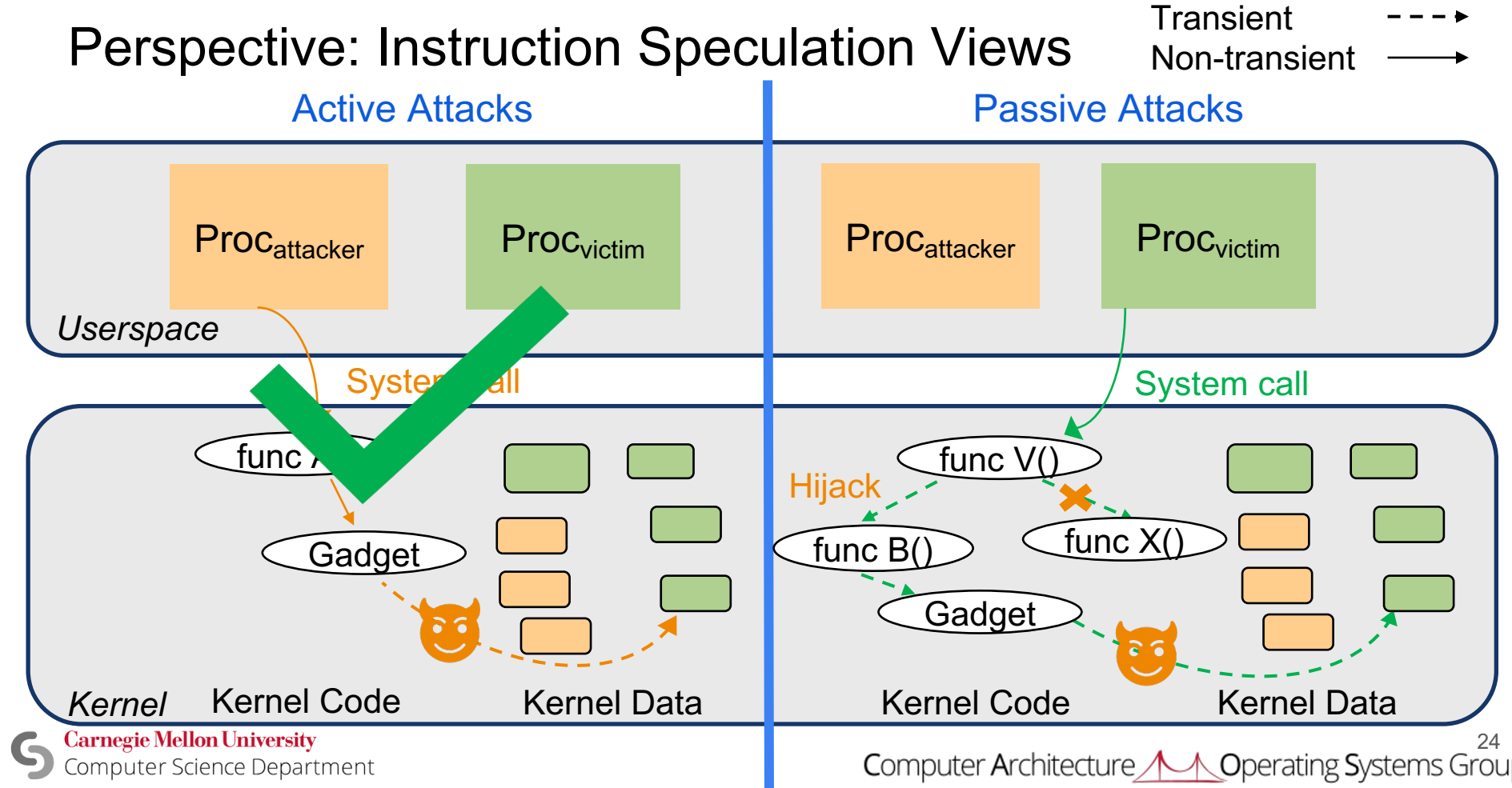
DSV Through Allocation



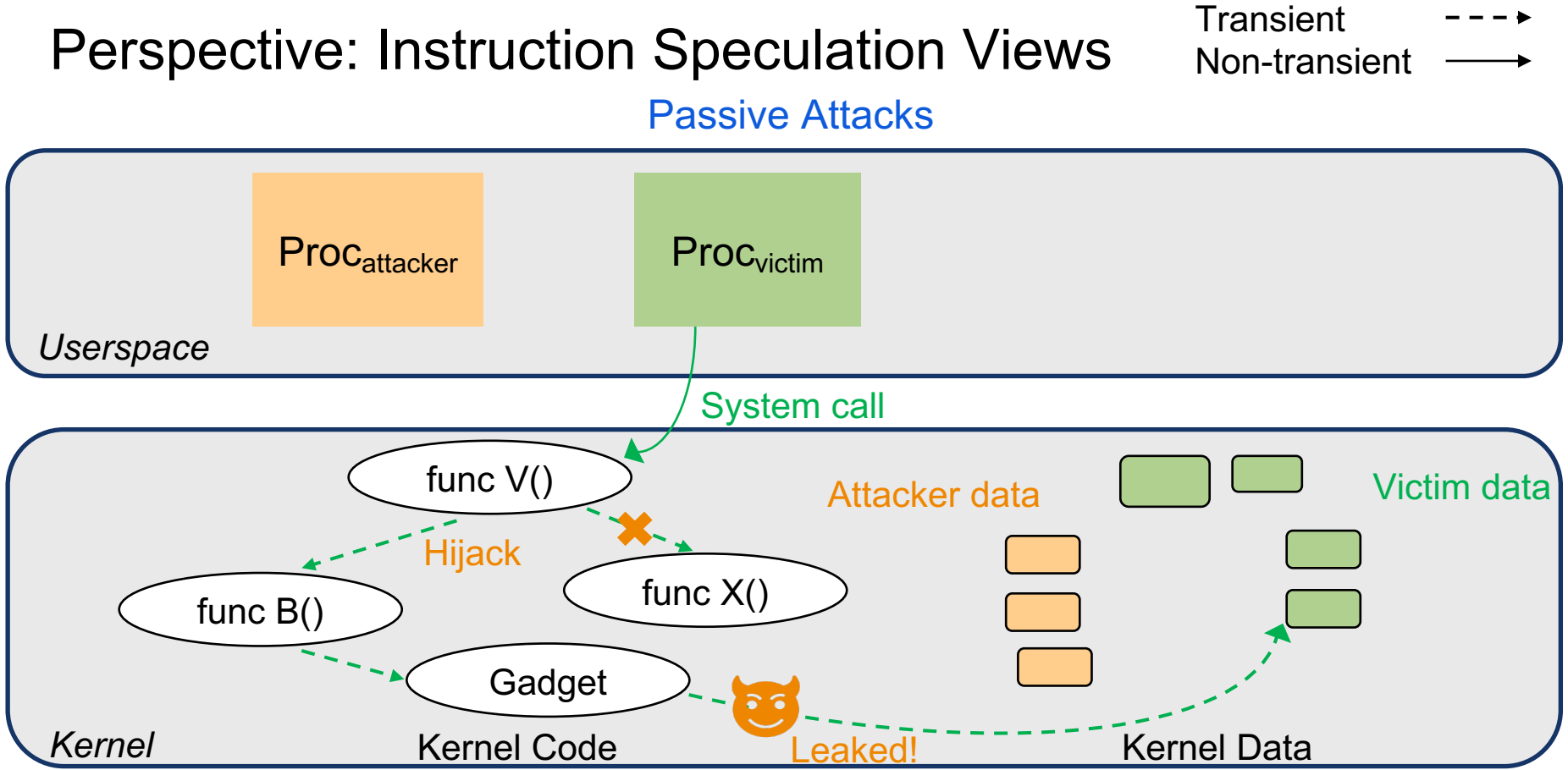
Perspective: Instruction Speculation Views



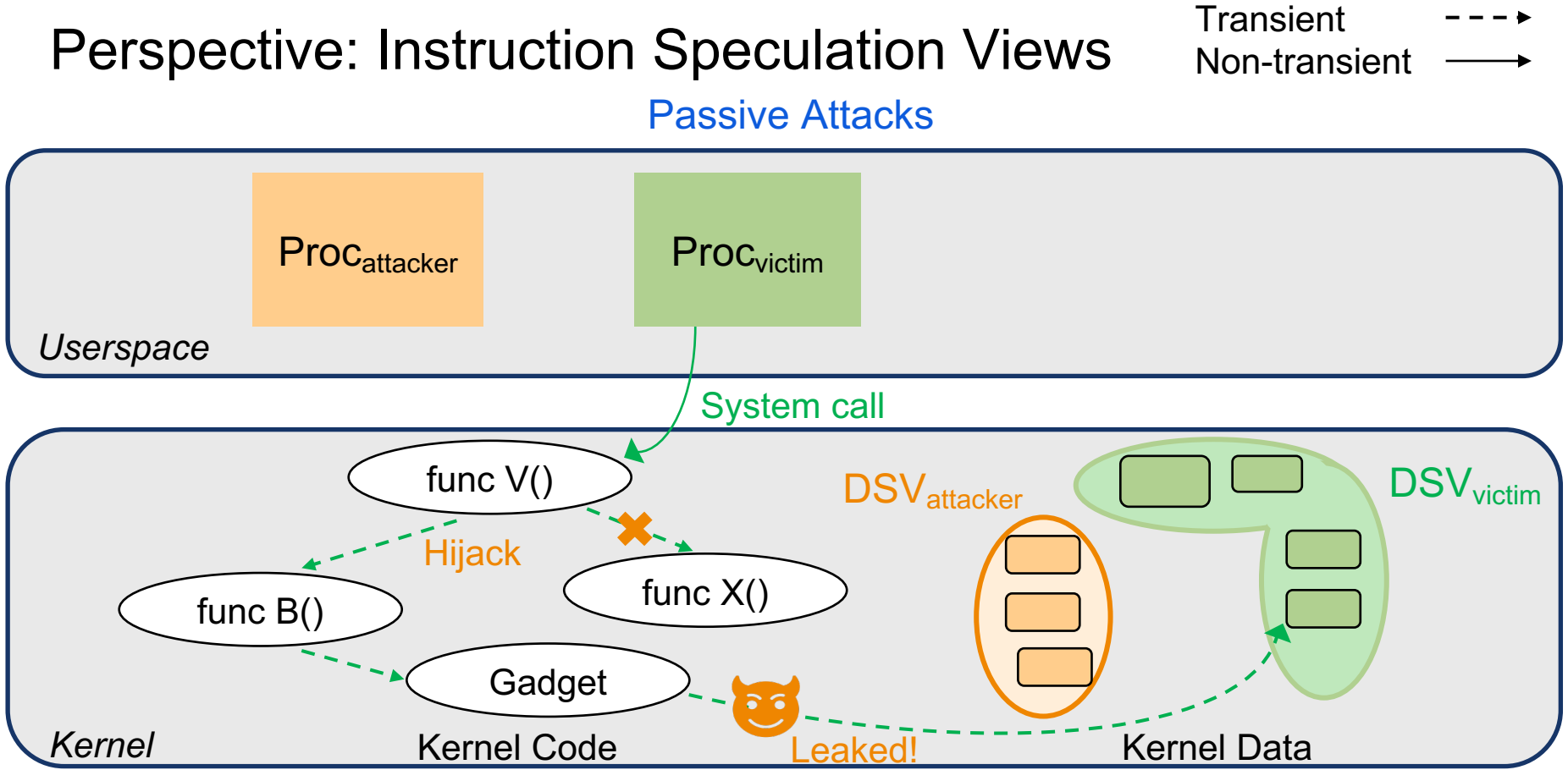
Perspective: Instruction Speculation Views



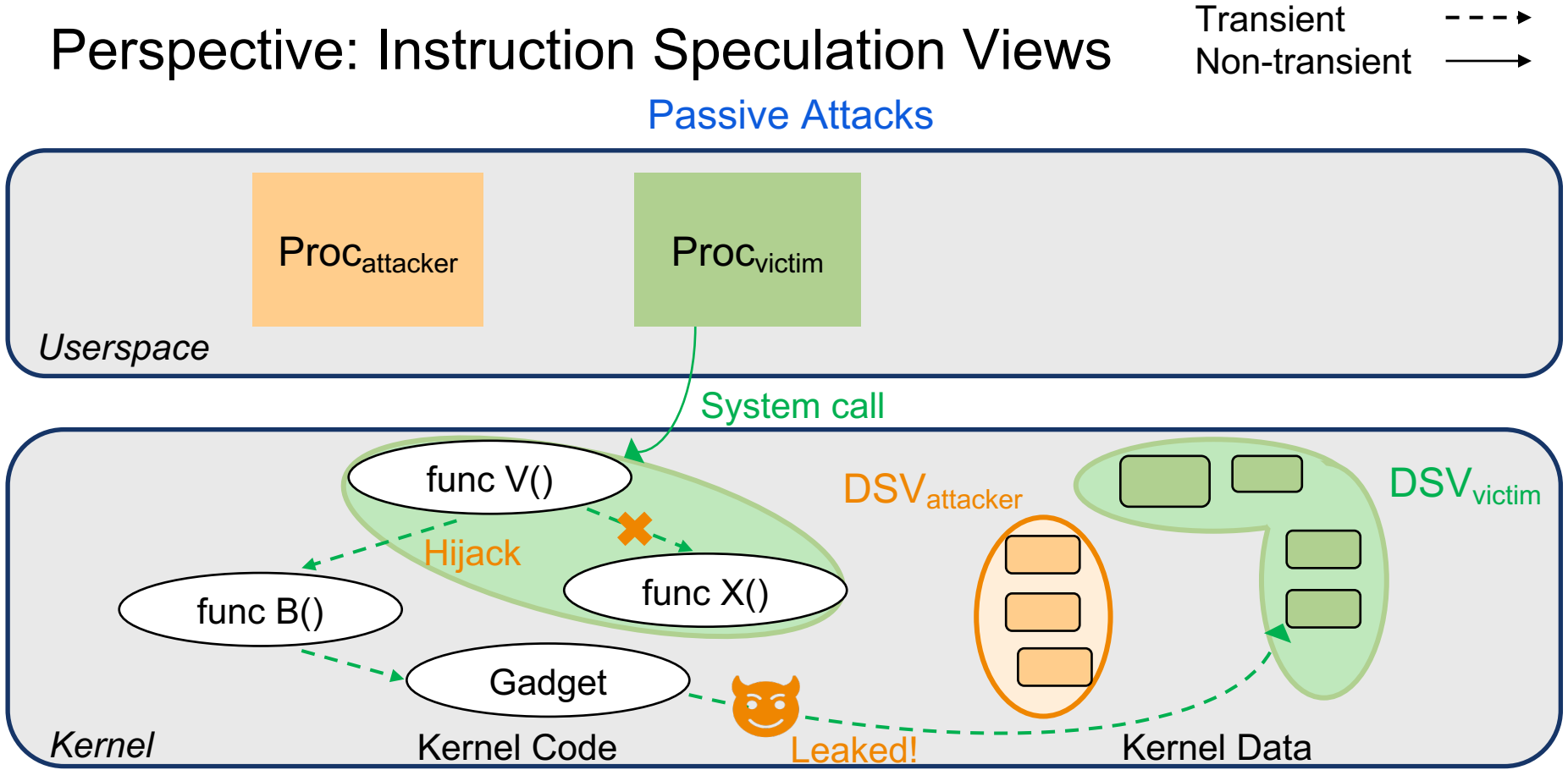
Perspective: Instruction Speculation Views



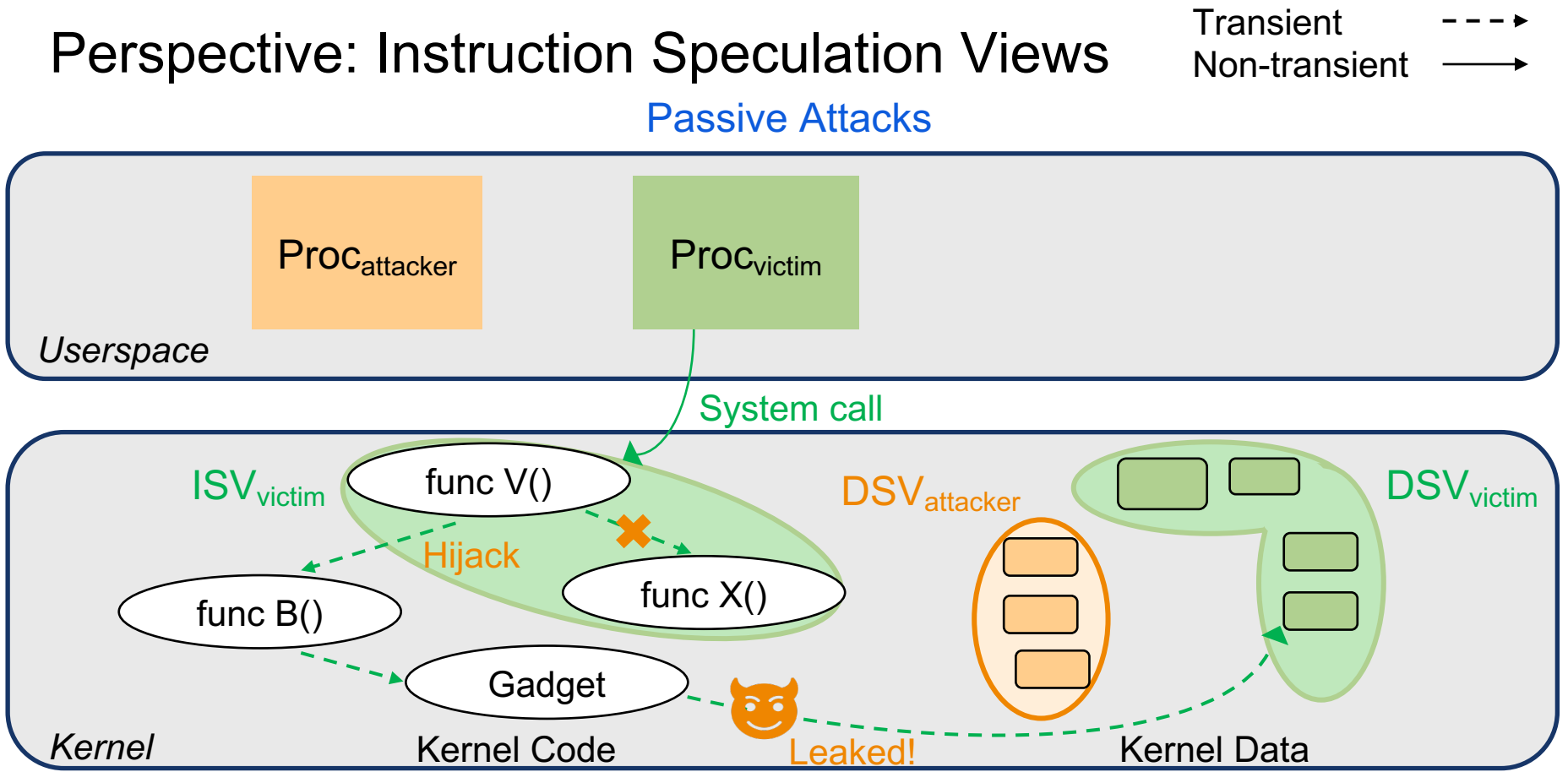
Perspective: Instruction Speculation Views



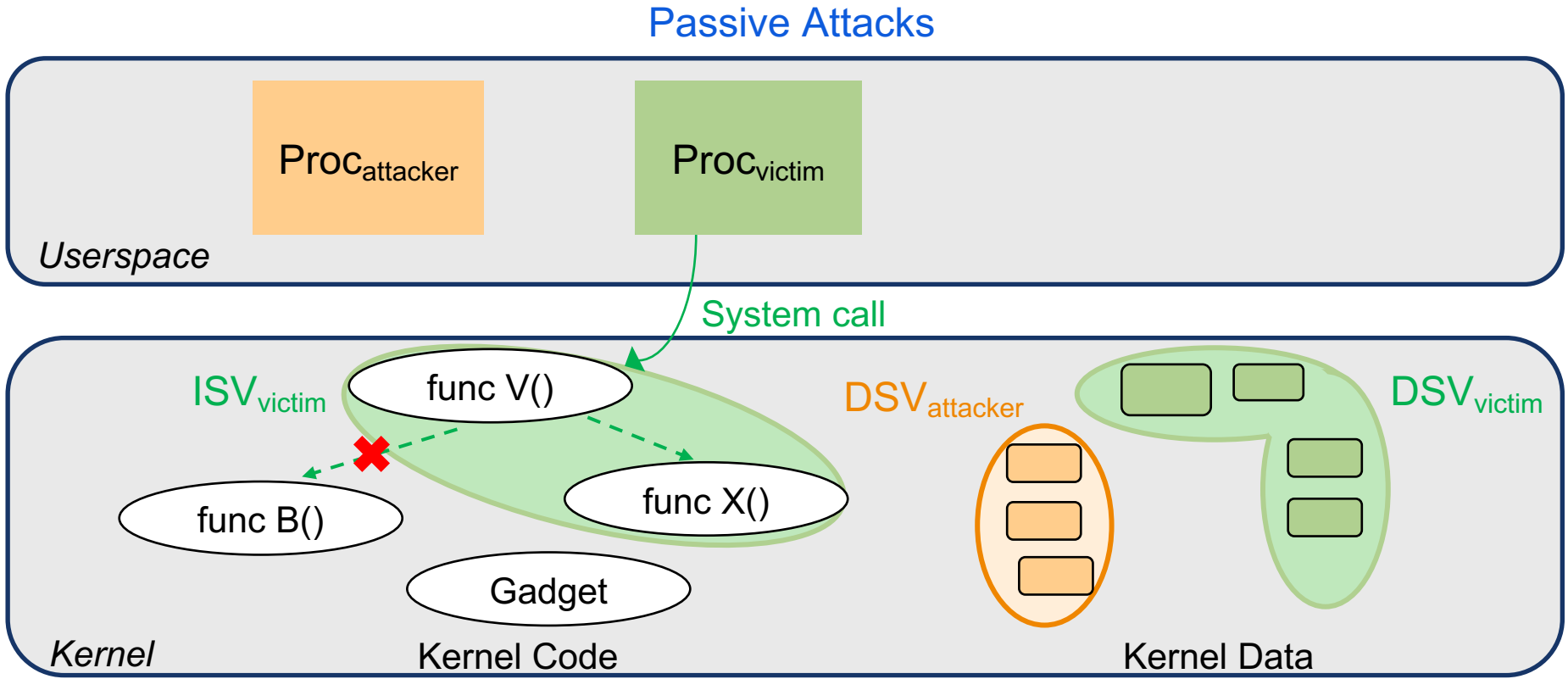
Perspective: Instruction Speculation Views



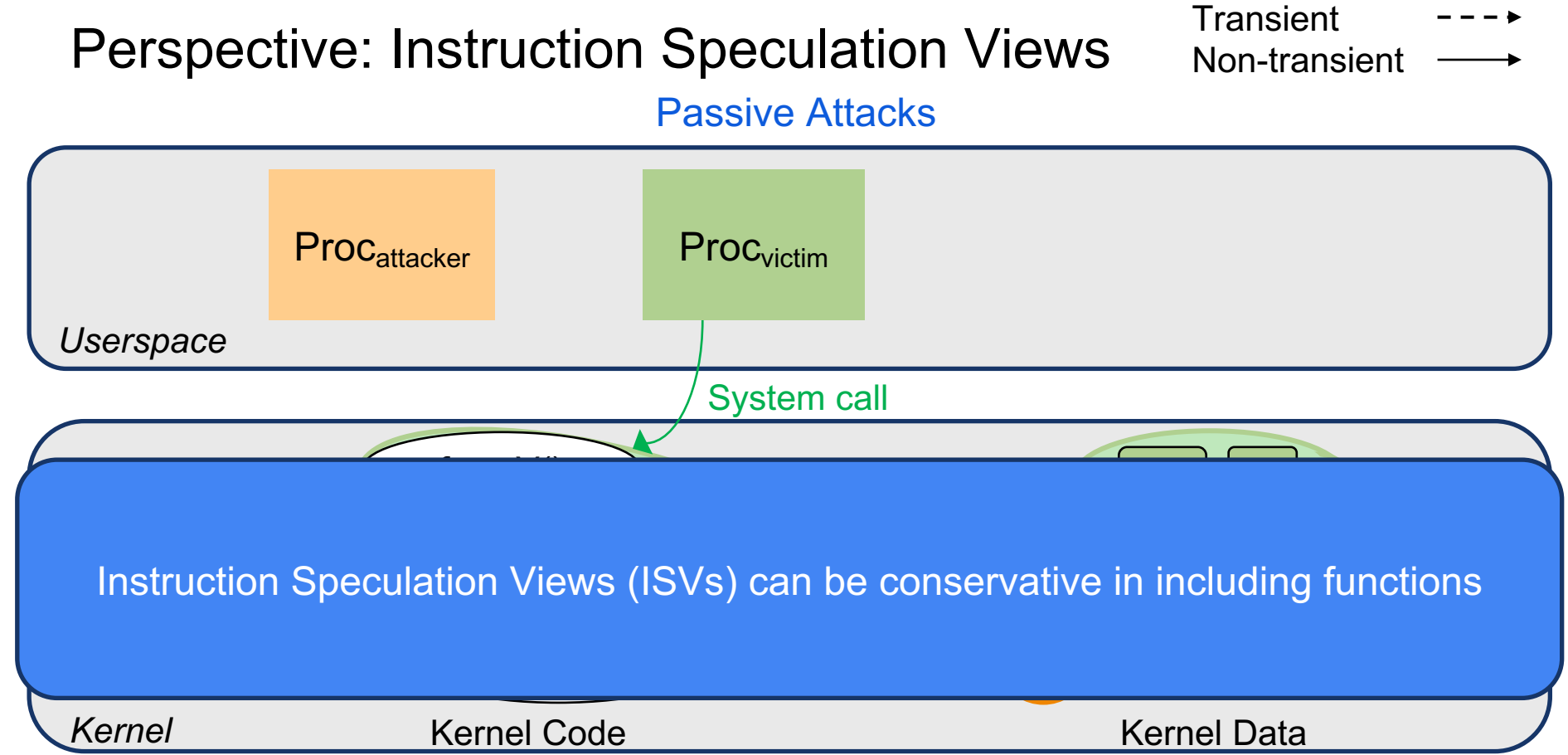
Perspective: Instruction Speculation Views



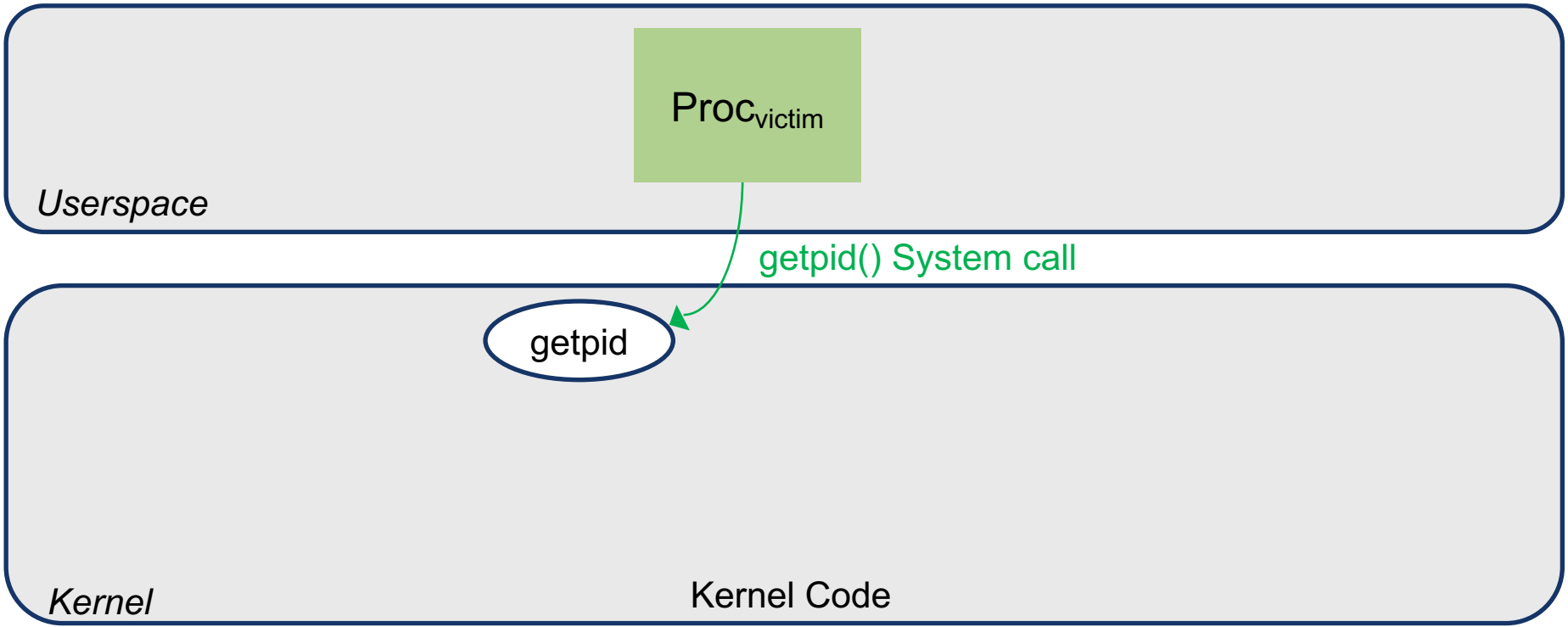
Perspective: Instruction Speculation Views



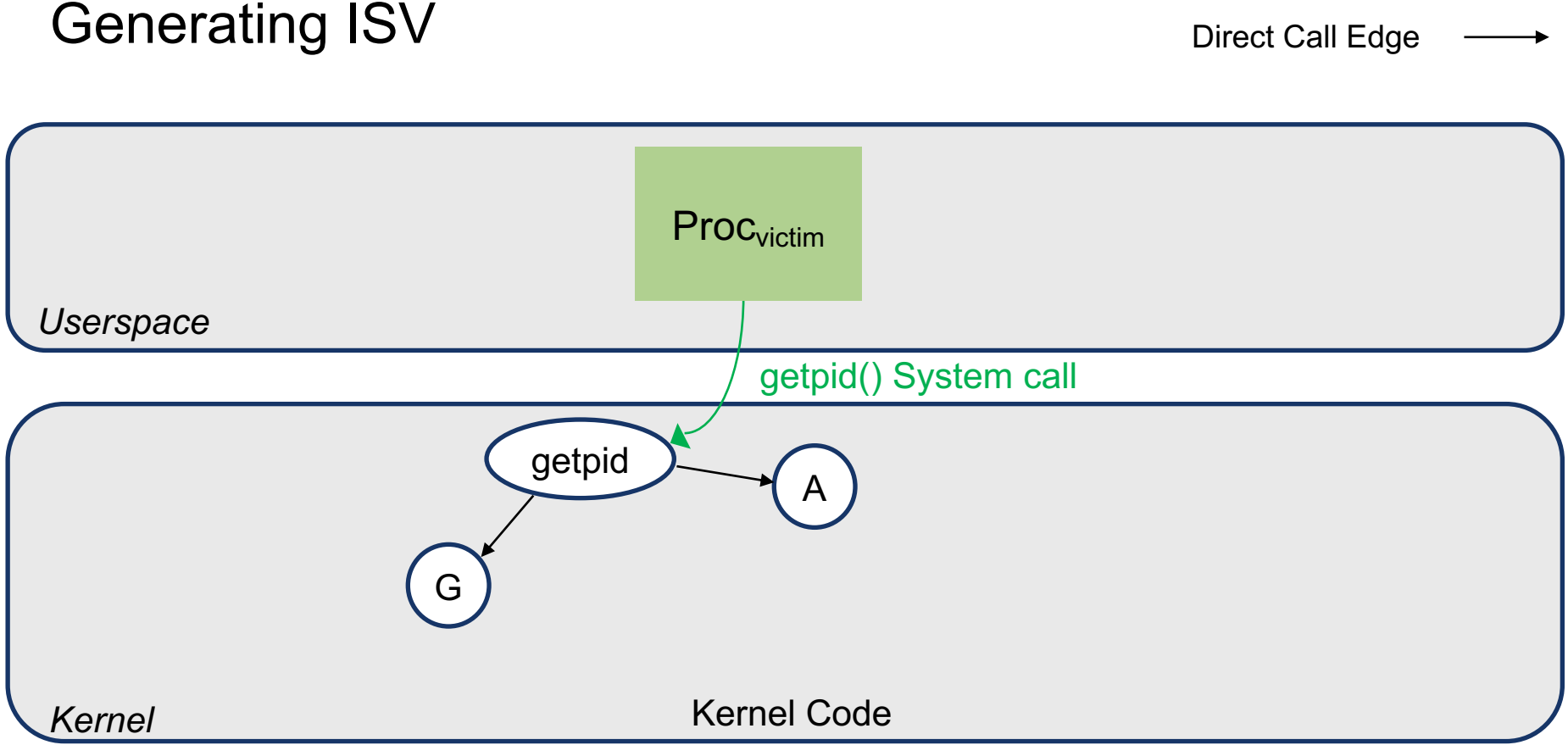
Perspective: Instruction Speculation Views



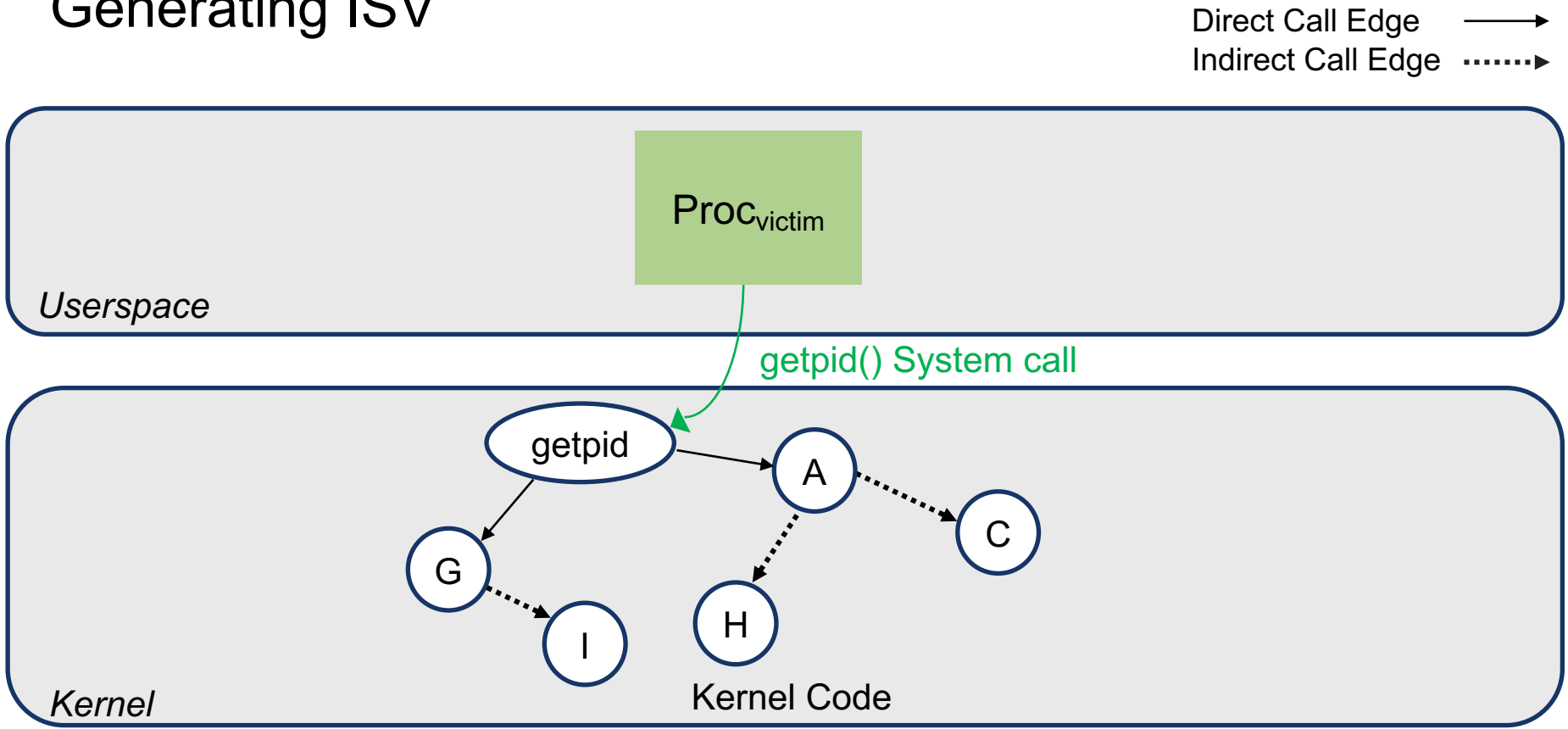
Generating ISV



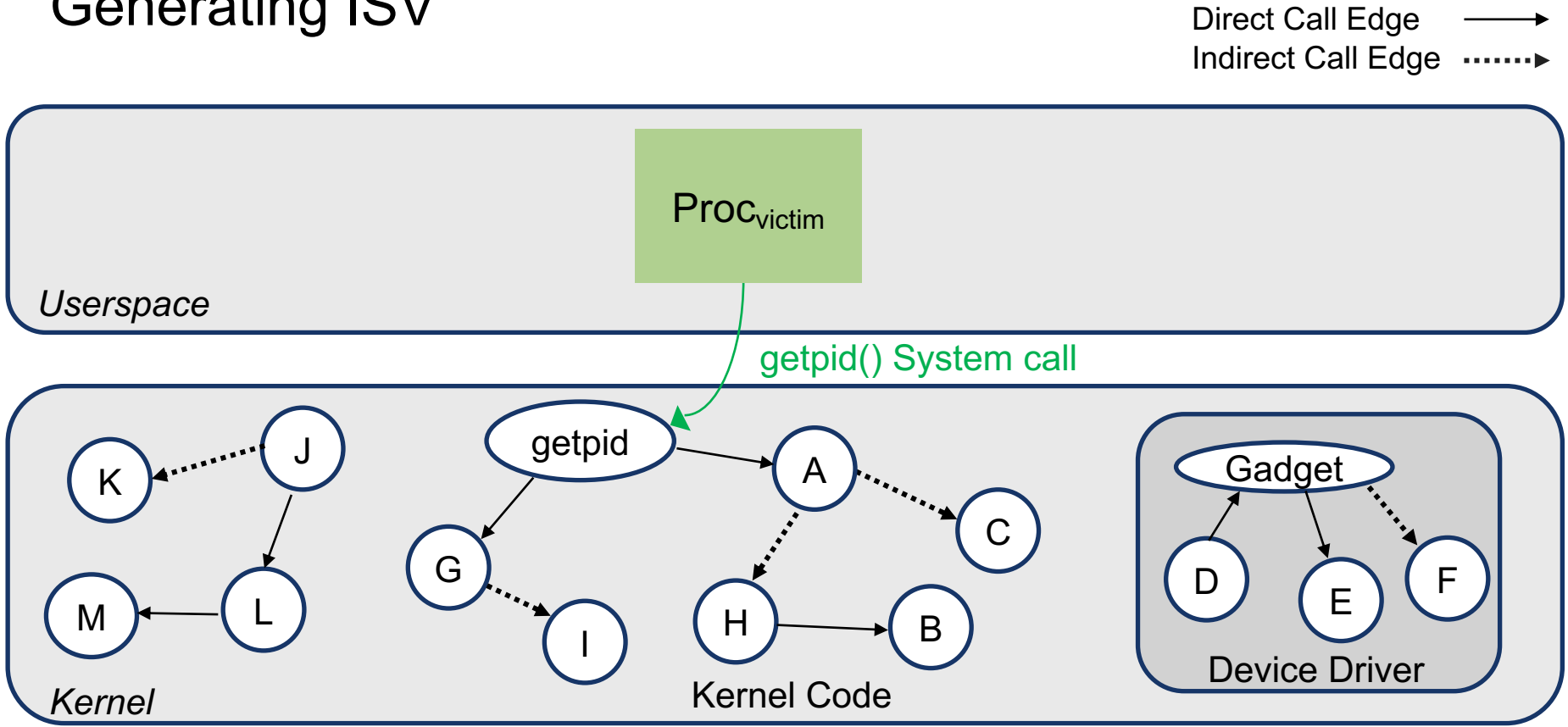
Generating ISV



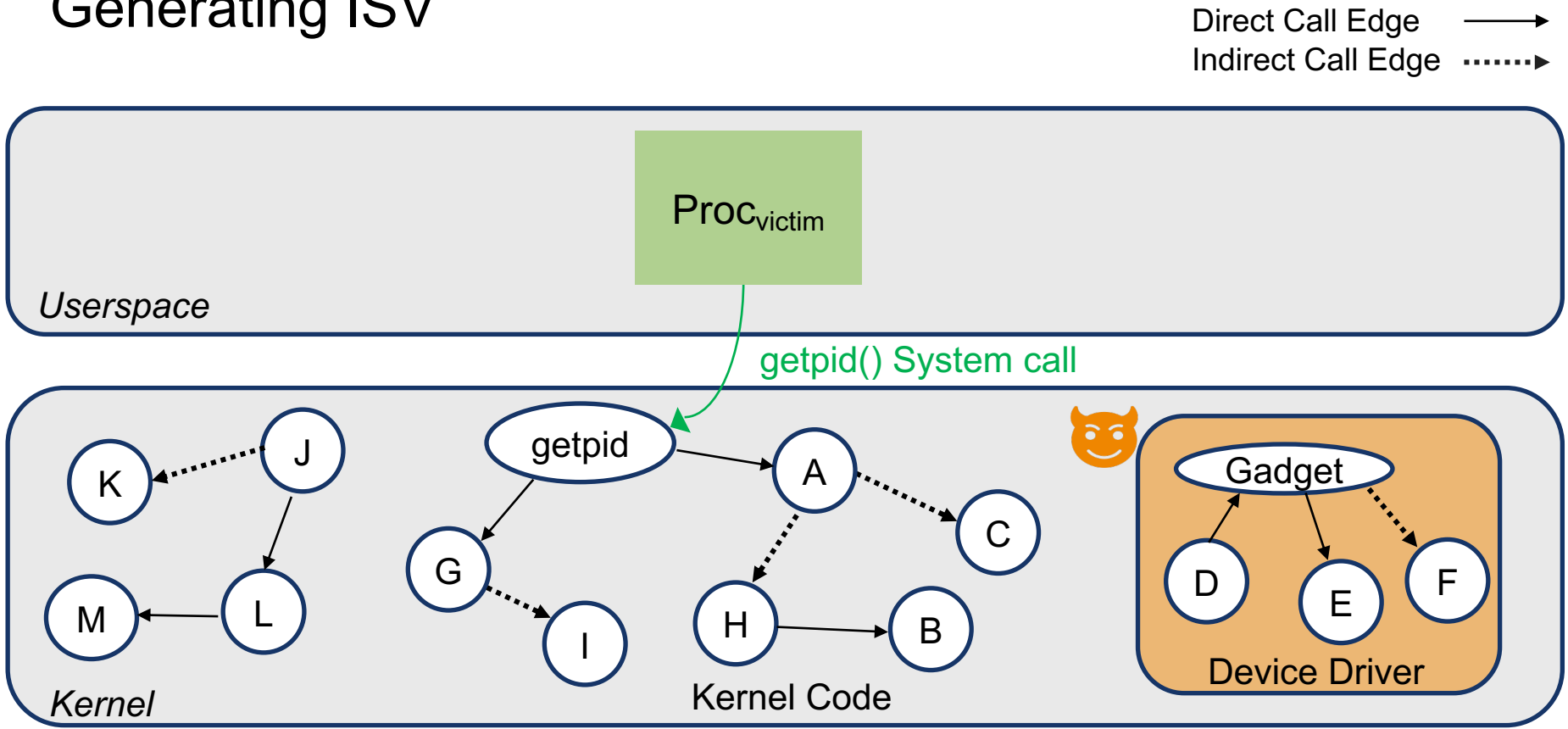
Generating ISV



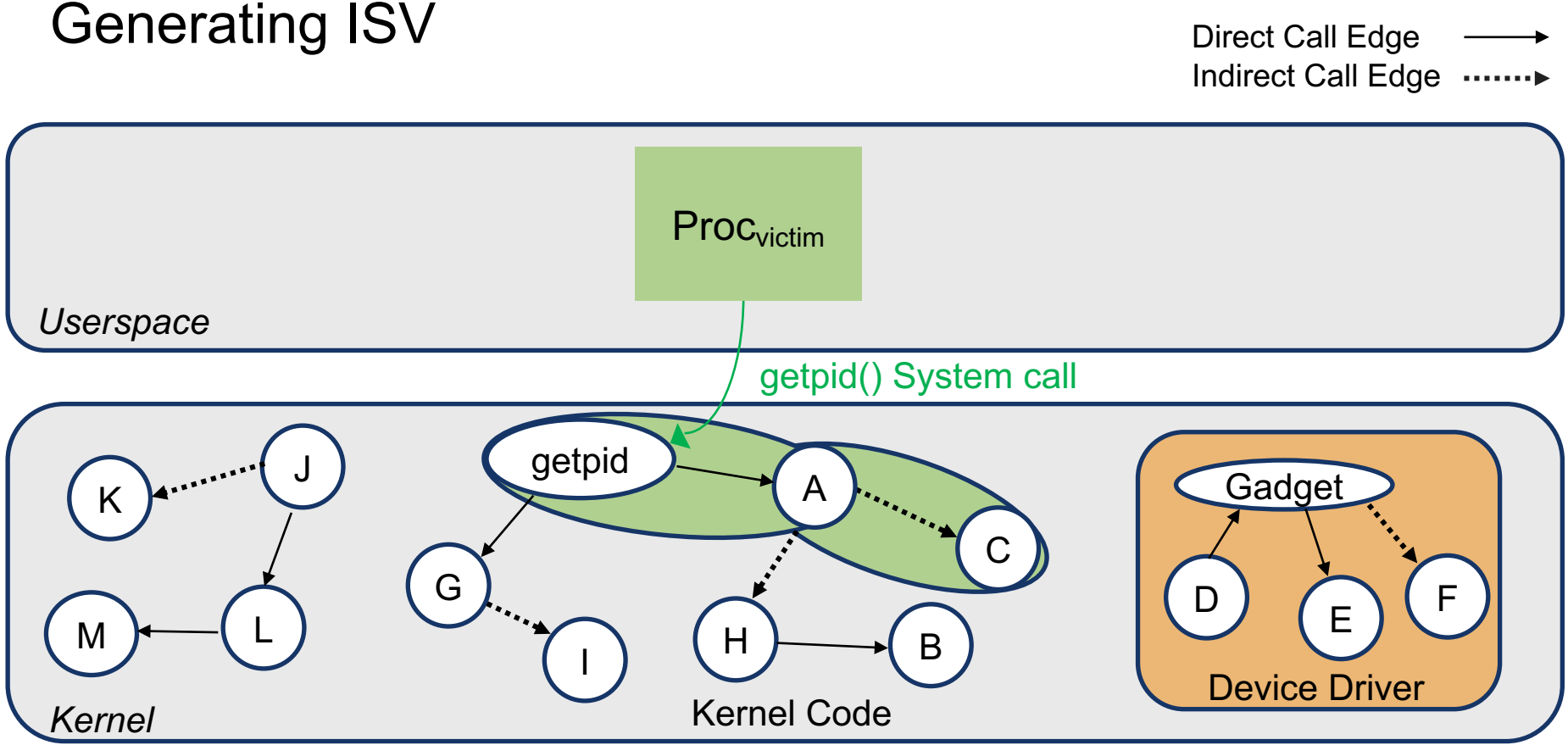
Generating ISV



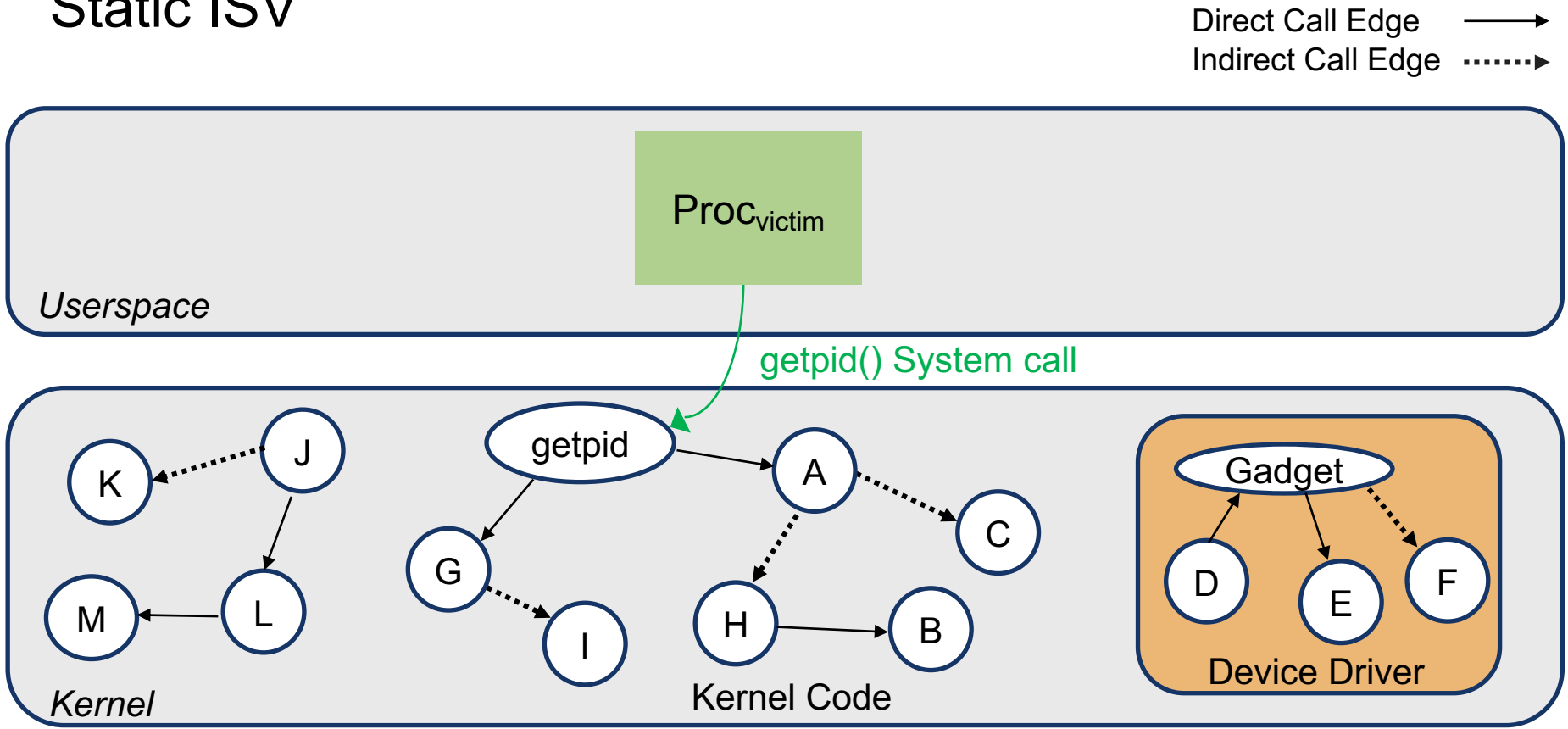
Generating ISV



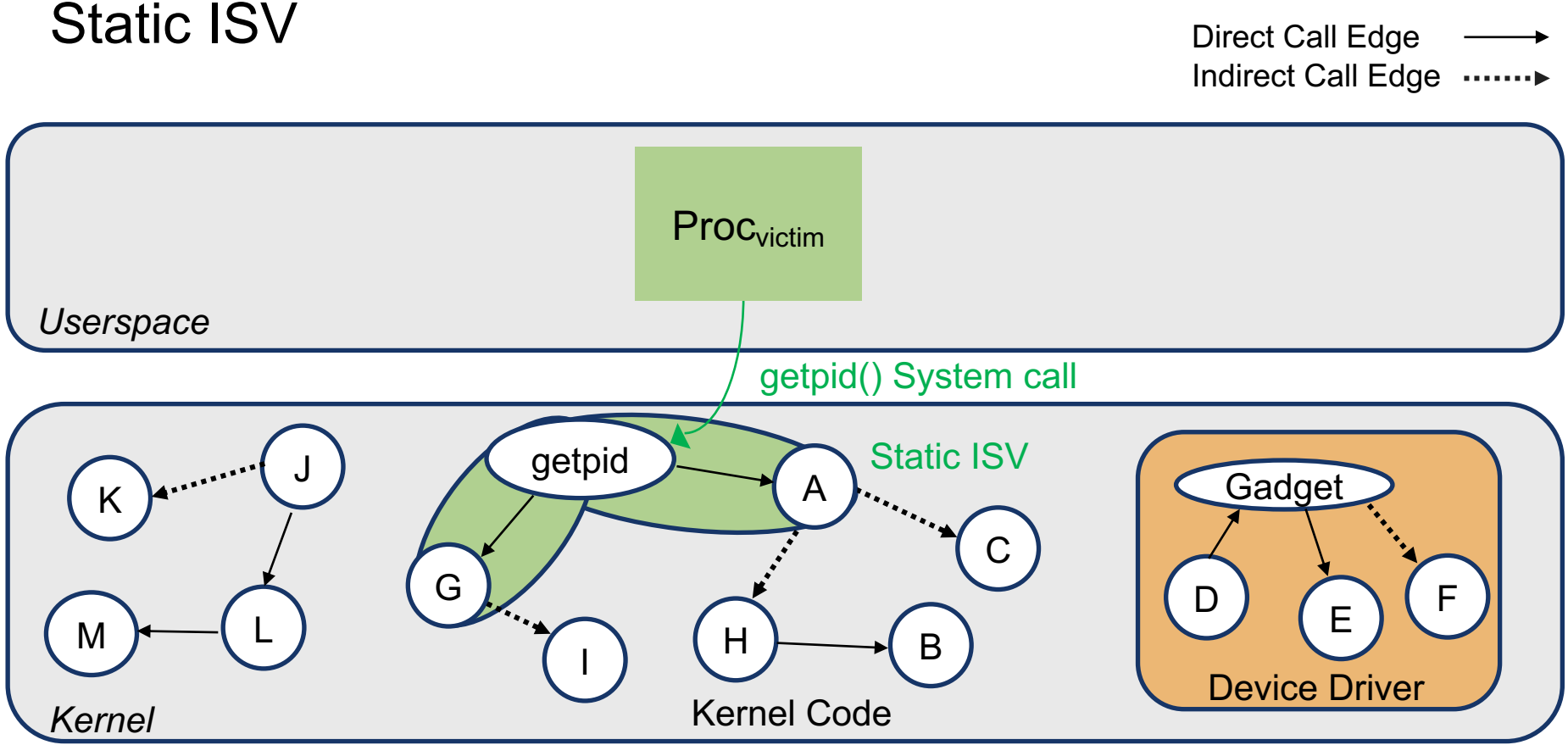
Generating ISV



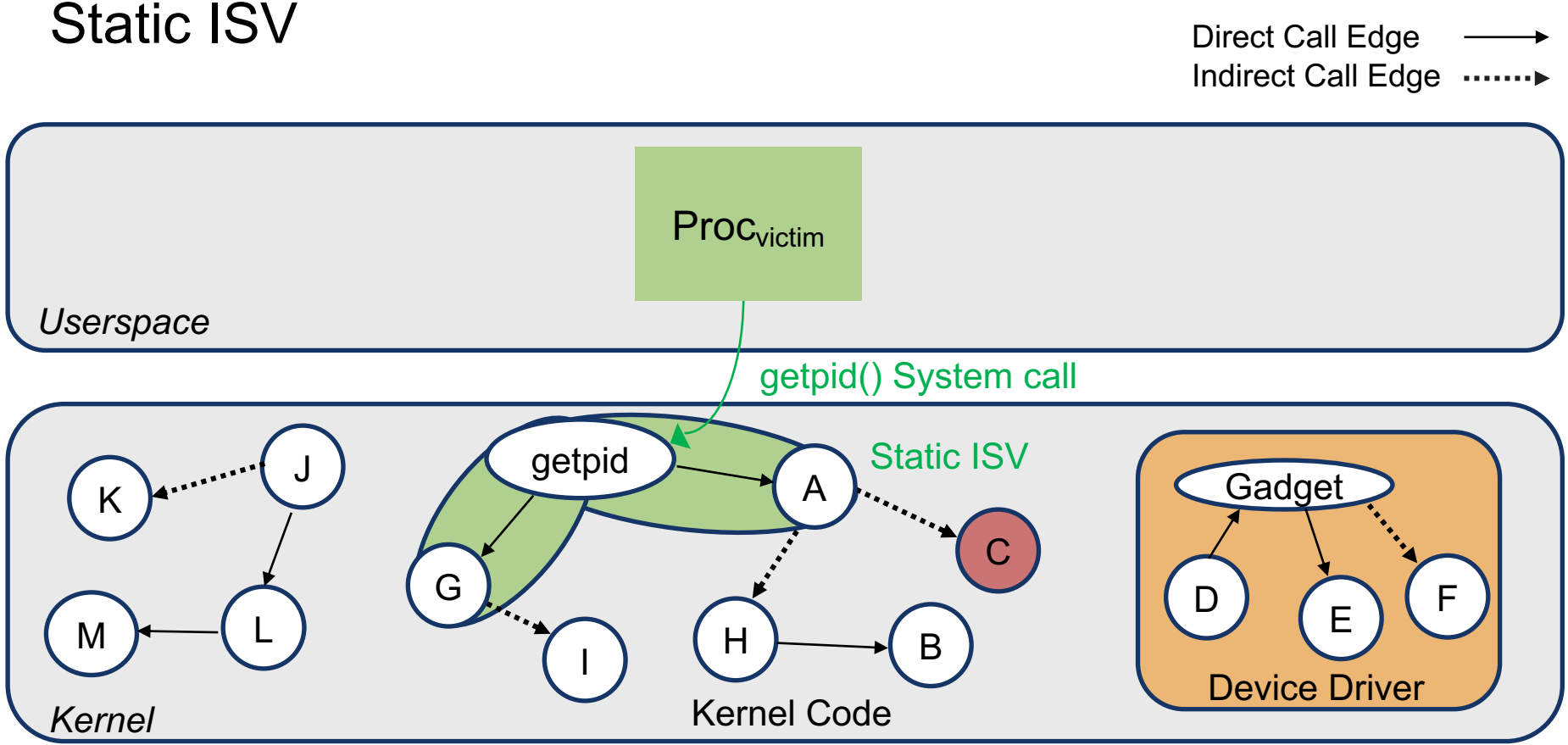
Static ISV


Direct Call Edge
Indirect Call Edge
Proc victim
Userspace
getpid() System call
K
J
M
L
Kernel
getpid
G
I
A
C
H
B
Kernel Code
Gadget
D
E
F
Device Driver

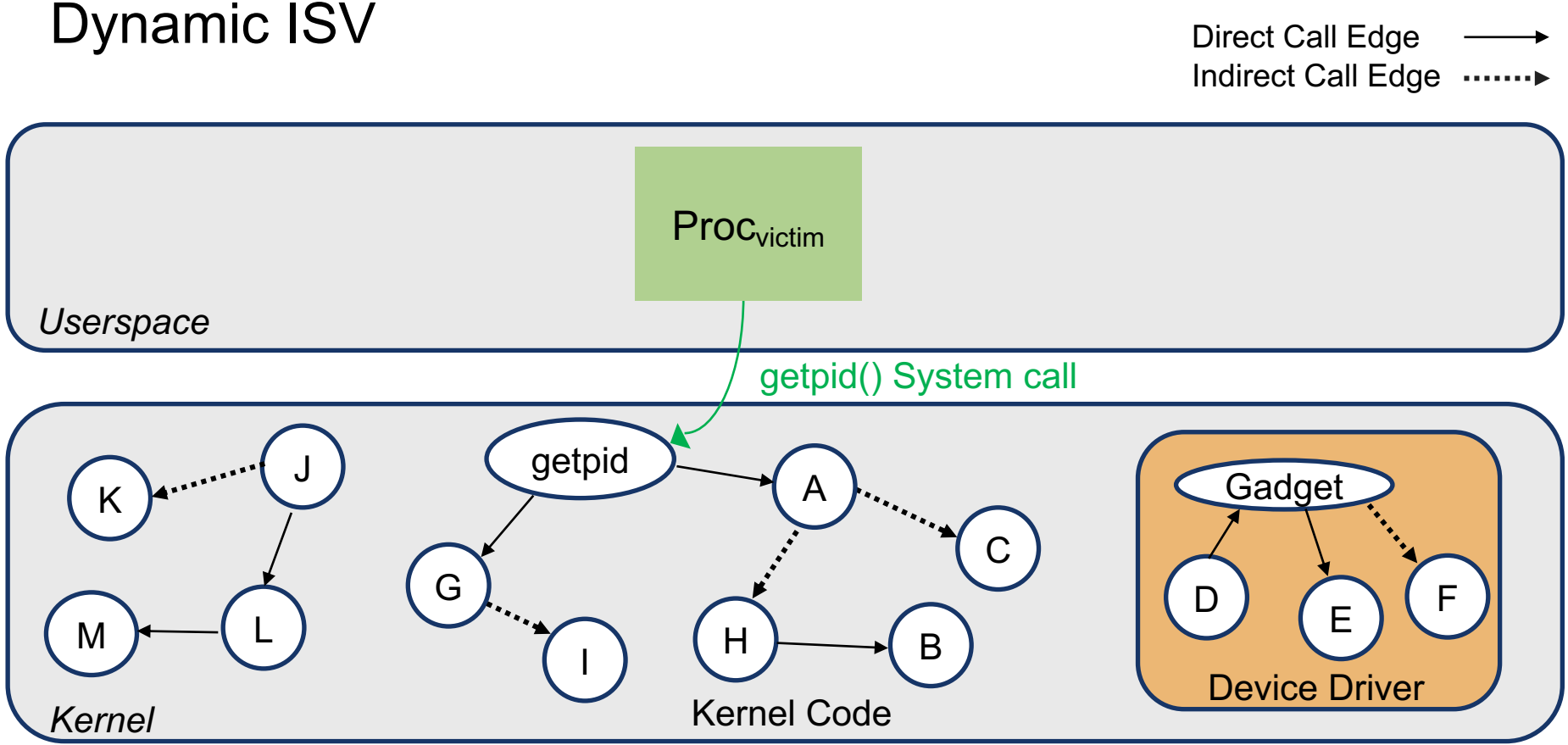
Static ISV


Direct Call Edge
Indirect Call Edge
Proc victim
Userspace
getpid() System call
K
J
M
L
Kernel
getpid
A
G
I
H
B
C
Static ISV
Kernel Code
Gadget
D
E
F
Device Driver

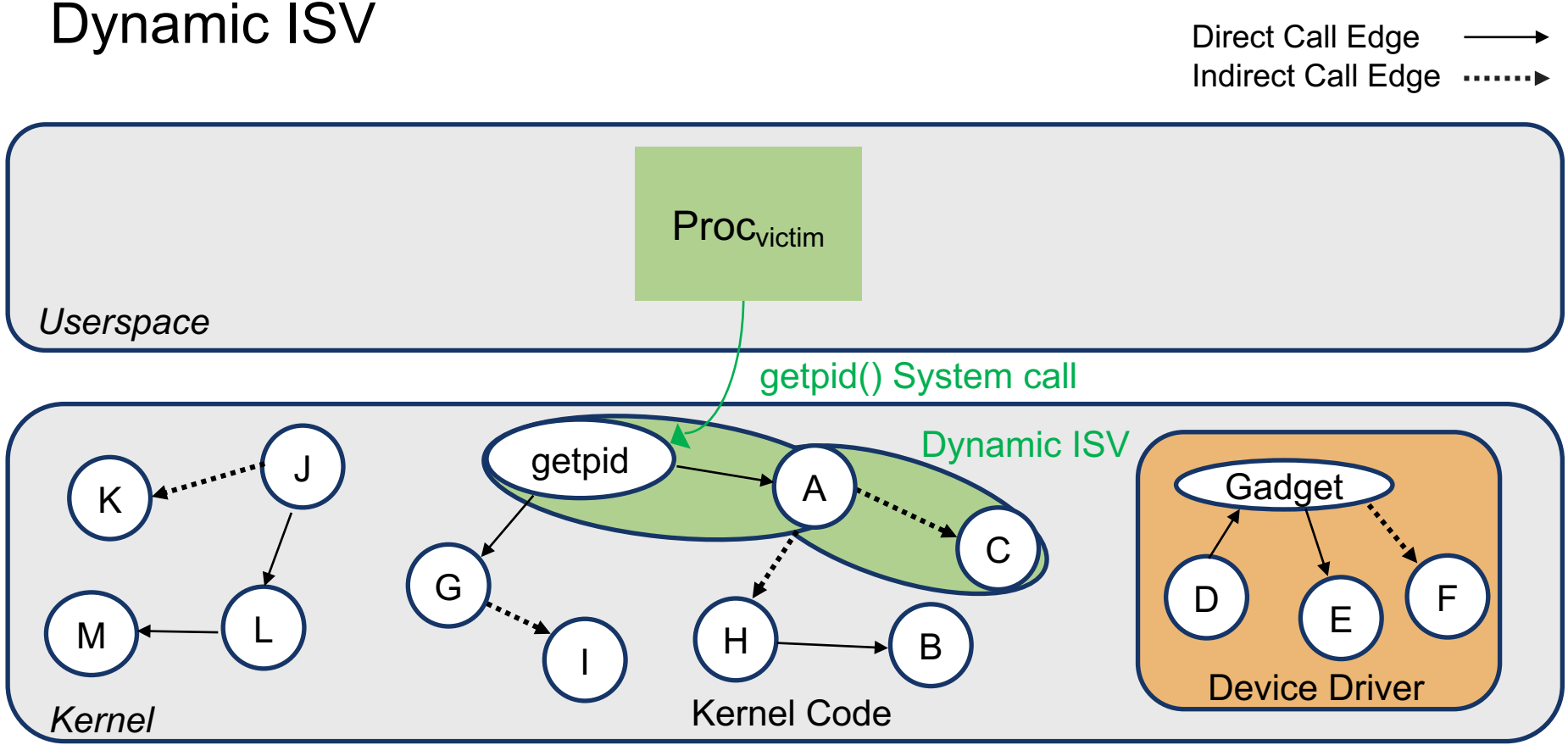
Static ISV


Direct Call Edge
Indirect Call Edge
Proc victim
Userspace
getpid() System call
K
J
M
L
Kernel
getpid
A
Static ISV
C
G
I
H
B
Kernel Code
Gadget
D
E
F
Device Driver

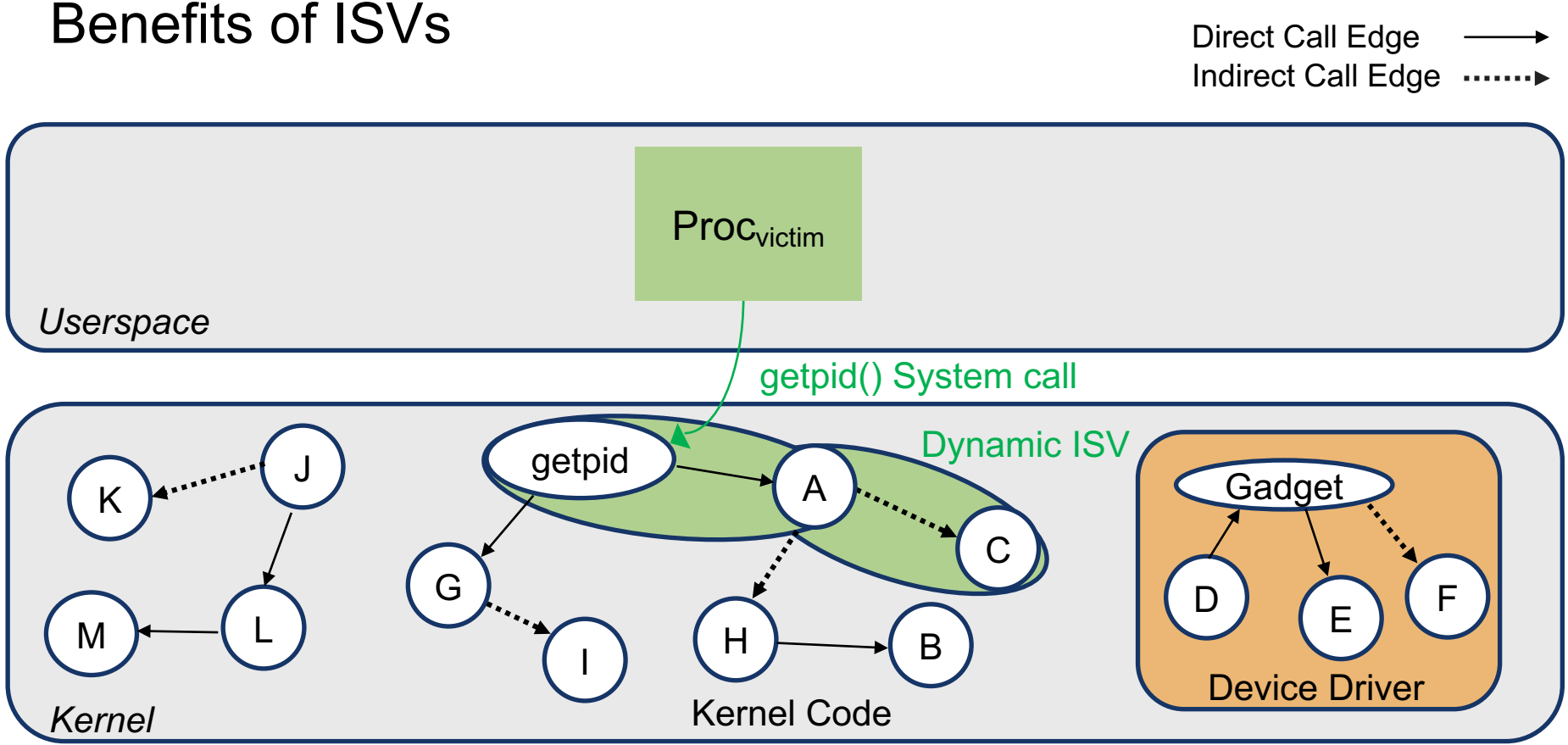
Dynamic ISV


Direct Call Edge
Indirect Call Edge
Proc victim
Userspace
getpid() System call
K
J
M
L
Kernel
getpid
G
I
A
C
H
B
Kernel Code
Gadget
D
E
F
Device Driver

Dynamic ISV


Direct Call Edge
Indirect Call Edge
Proc victim
Userspace
getpid() System call
Dynamic ISV
getpid
A
C
G
I
H
B
K
J
M
L
Kernel
Kernel Code
Gadget
D
E
F
Device Driver

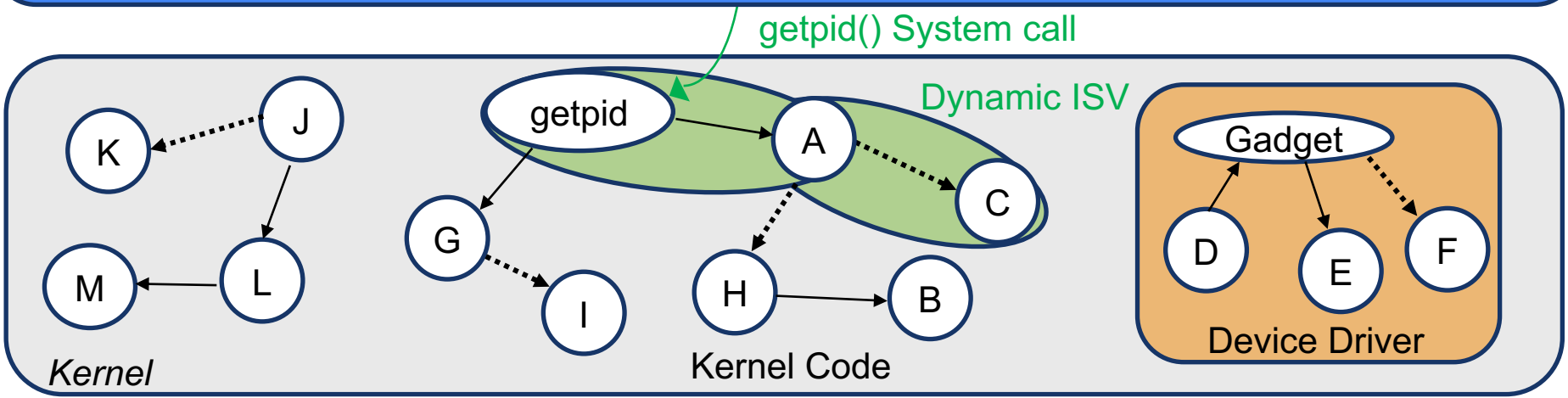
Benefits of ISVs



Benefits of ISVs

Direct Call Edge ——▸
Indirect Call Edge ·····▸

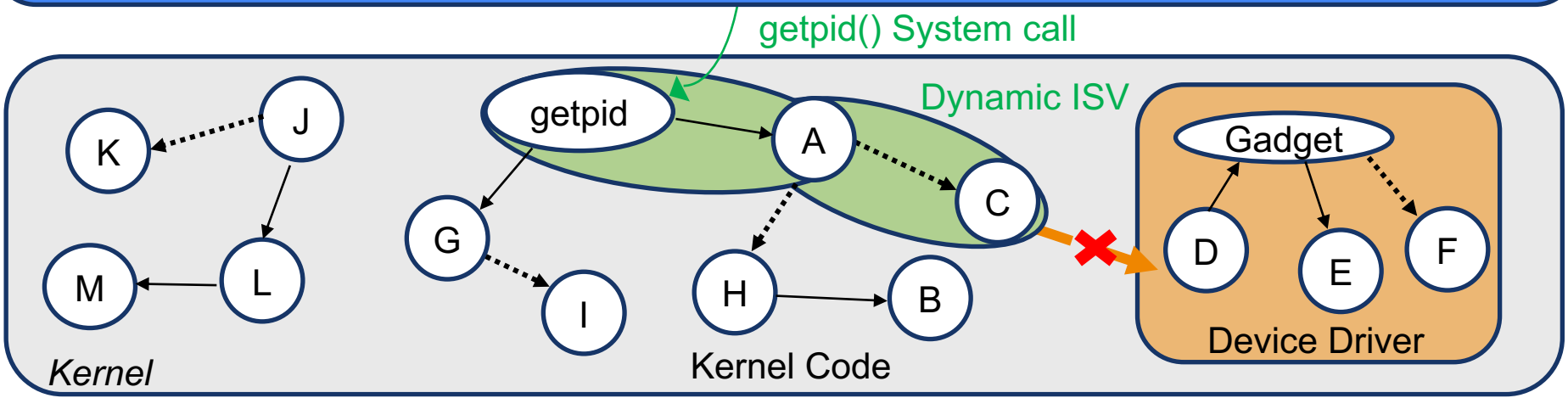
Reduce Attack Surface: Exclude unused or infrequently used kernel functions



Benefits of ISVs

Direct Call Edge ——▸
Indirect Call Edge ·····▸

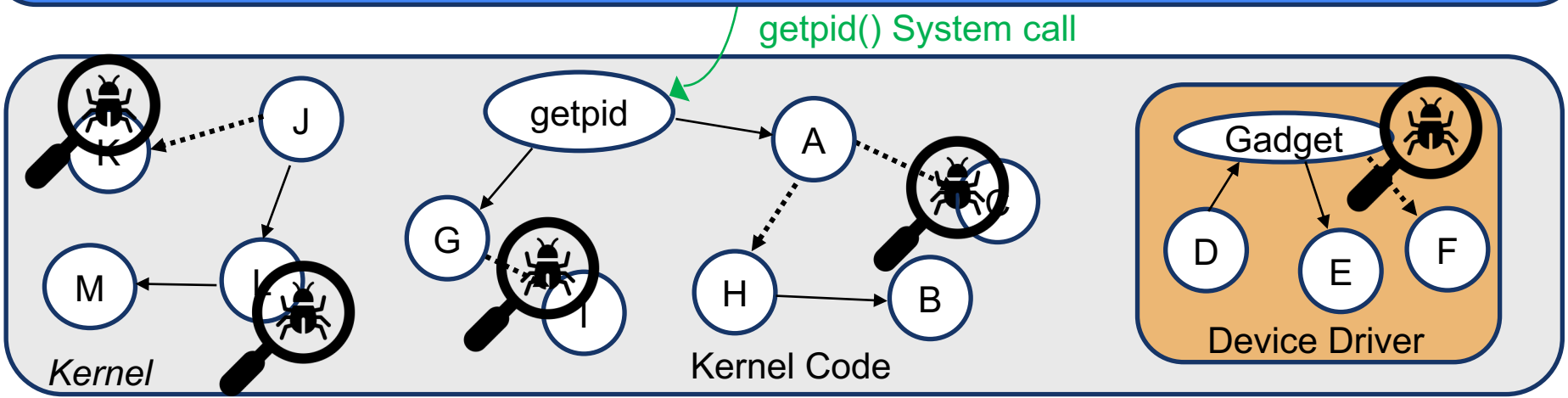
Reduce Attack Surface: Exclude unused or infrequently used kernel functions



Benefits of ISVs

Direct Call Edge ——▸
Indirect Call Edge ·····▸

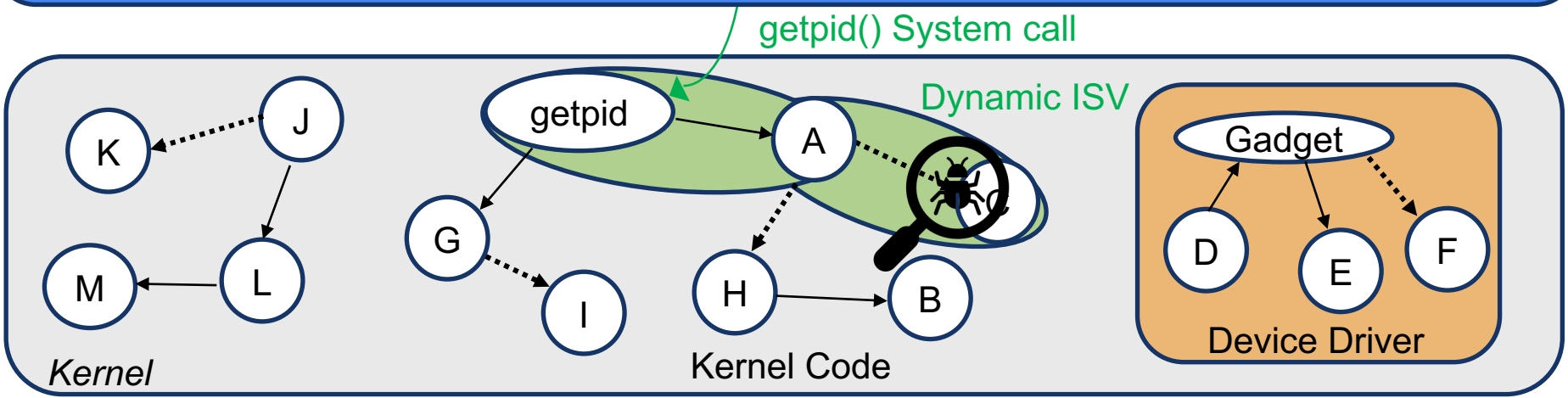
Accelerate Gadget Finding: Focus on a smaller set of functions



Benefits of ISVs

Direct Call Edge ——▶
Indirect Call Edge ·····▶

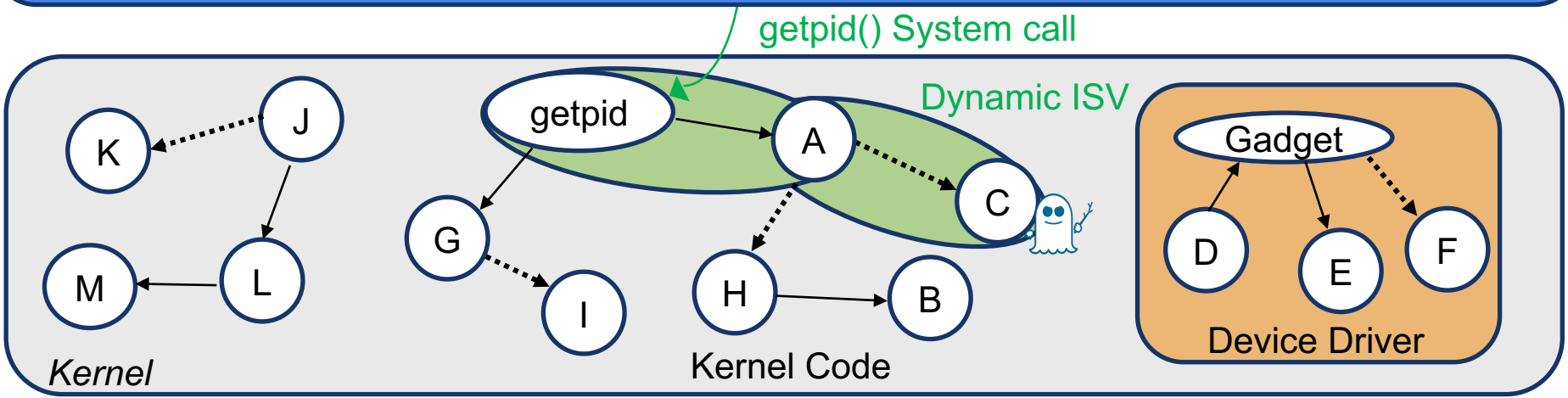
Accelerate Gadget Finding: Focus on a smaller set of functions



Benefits of ISVs

Direct Call Edge ——→
Indirect Call Edge ·····→

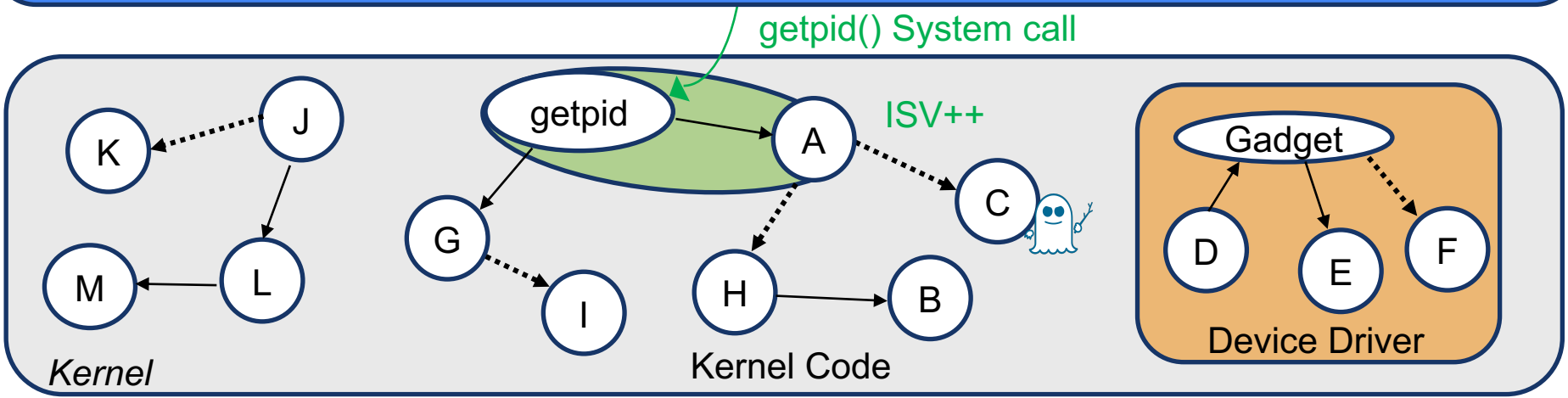
Enhance ISVs: Leverage results of gadget finding to generate stricter ISVs (ISV++)
Swift Mitigation: Quickly exclude vulnerable kernel functions from ISVs



Benefits of ISVs

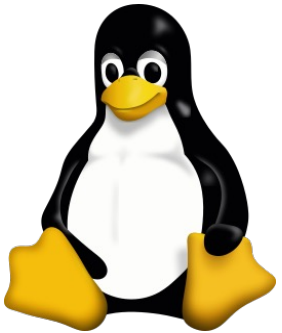
Direct Call Edge ——▸
Indirect Call Edge ·····▸

Enhance ISVs: Leverage results of gadget finding to generate stricter ISVs (ISV++)
Swift Mitigation: Quickly exclude vulnerable kernel functions from ISVs

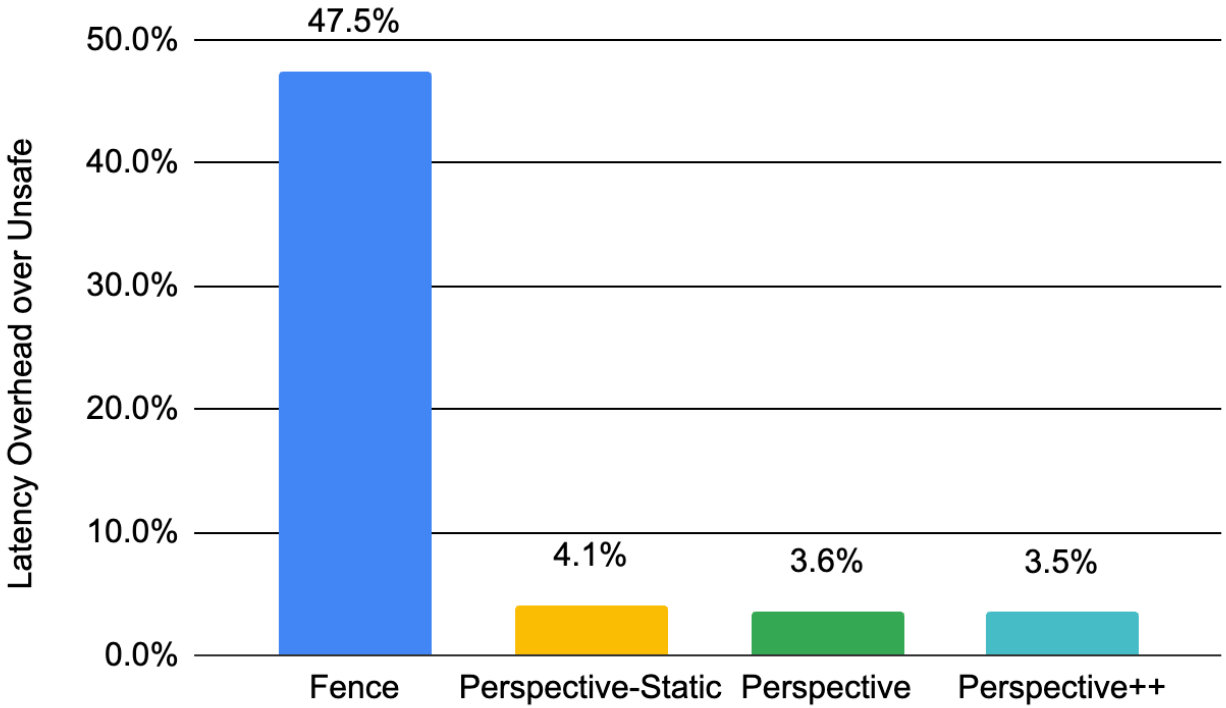


Experiment Setup

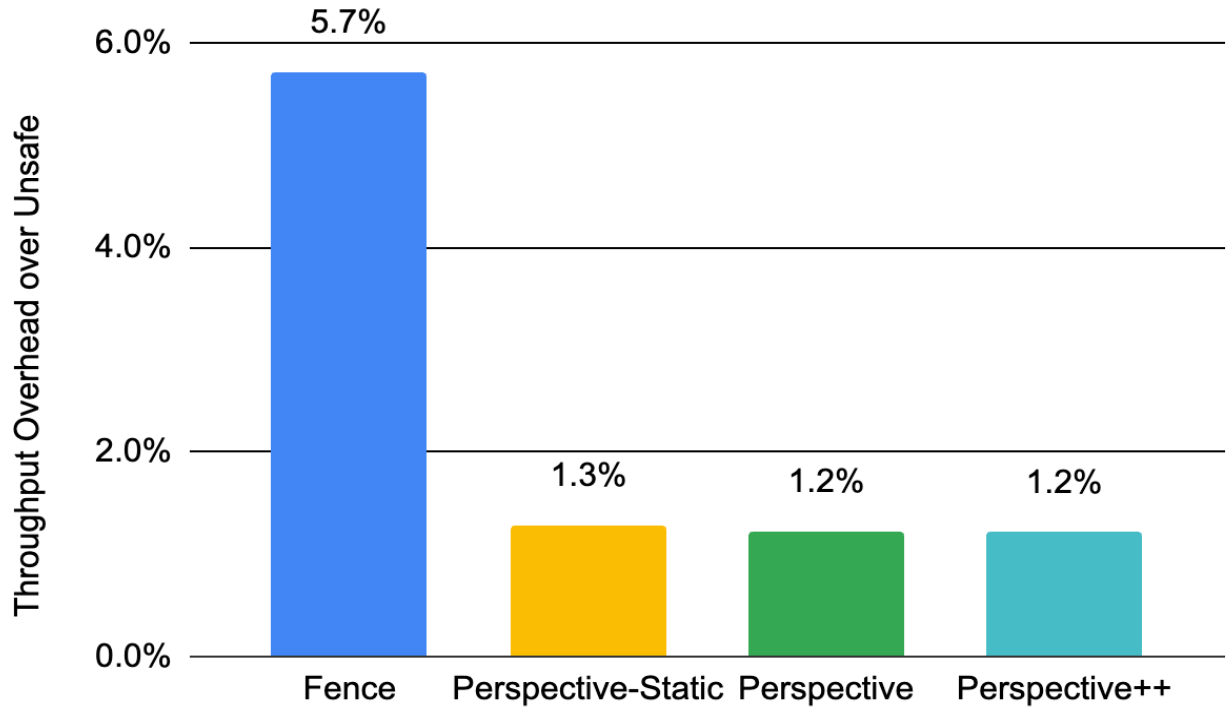
- Gem5 with Linux kernel (v5.4.49) modified for Perspective
- Workloads:
  - LEBench
  - Datacenter Applications -- httpd, nginx, redis, memcached
- Defenses
  - Fence
  - Perspective-Static
  - Perspective
  - Perspective++



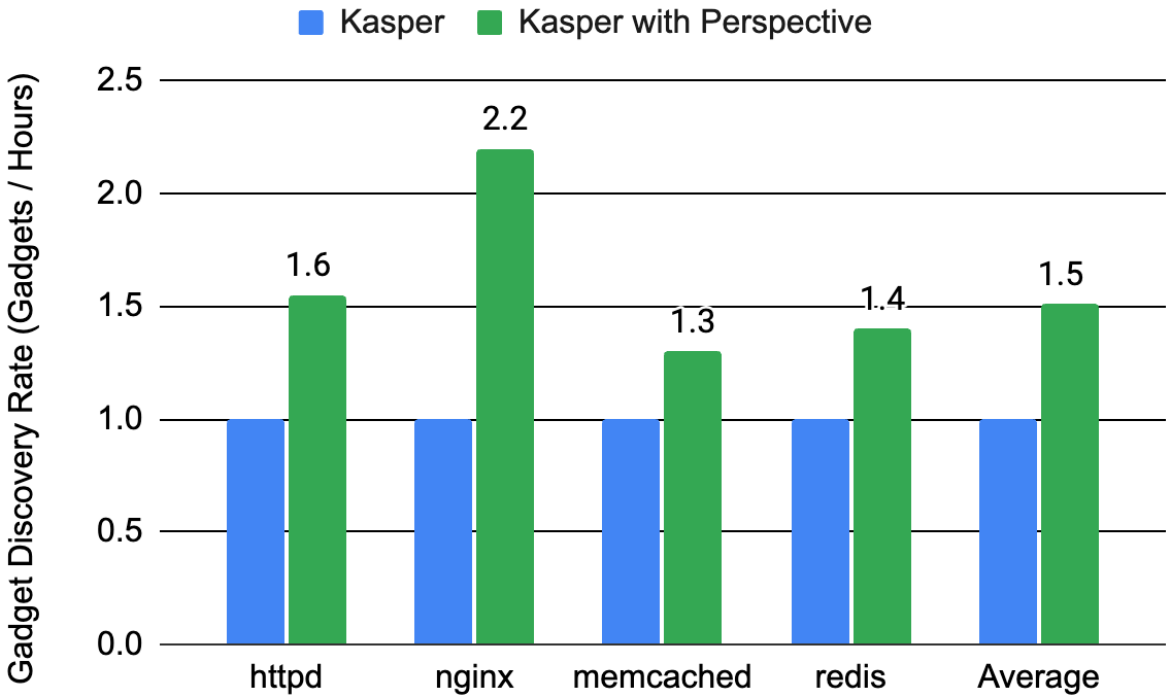
LEBench Evaluation: Average Latency Overhead



Applications Evaluation: Average Throughput Overhead

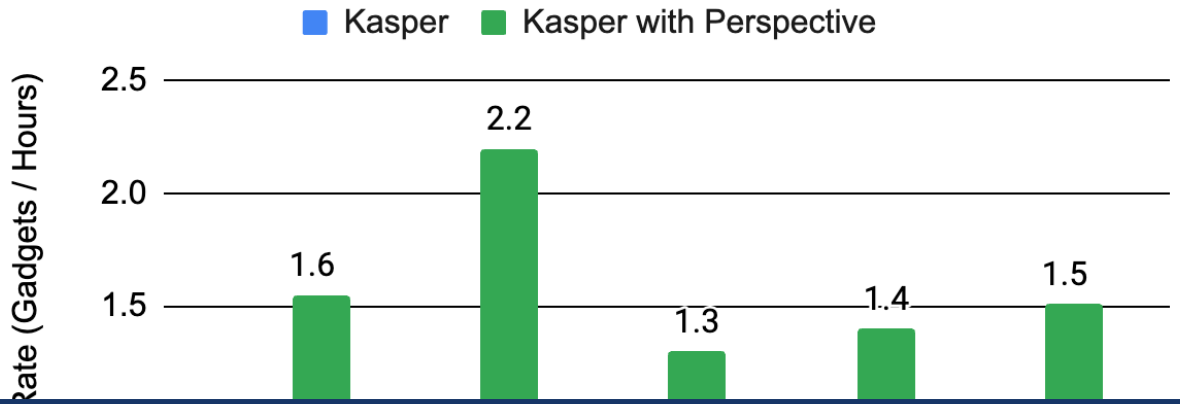


Security Evaluation of ISVs via Kasper[1]



[1]B. Johannesmeyer, J. Koschel, K. Razavi, H. Bos, and C. Giuffrida, "Kasper: Scanning for Generalized Transient Execution Gadgets in the Linux Kernel," in NDSS, Apr. 2022.

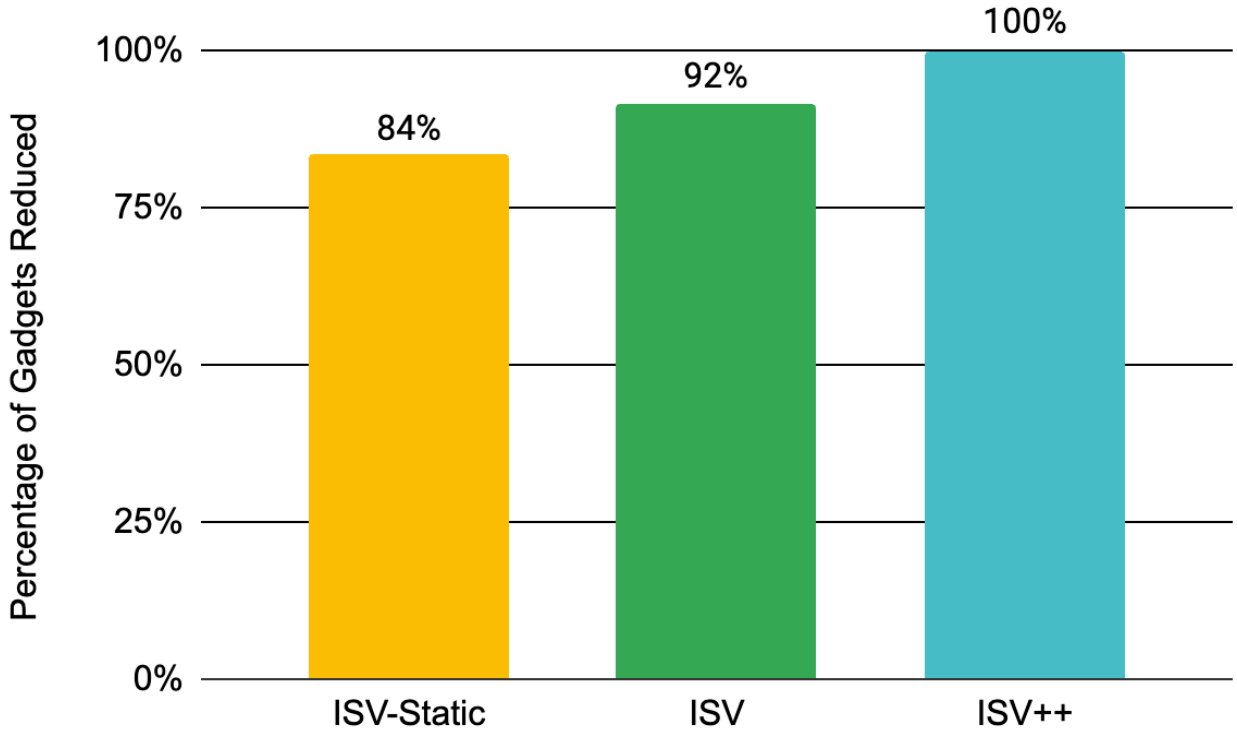
Security Evaluation of ISVs via Kasper[1]



ISVs allow for speedup in search for relevant gadgets

[1]B. Johannesmeyer, J. Koschel, K. Razavi, H. Bos, and C. Giuffrida, “Kasper: Scanning for Generalized Transient Execution Gadgets in the Linux Kernel,” in NDSS, Apr. 2022.

Security Evaluation: Average Gadget Reduction Percent



But Wait There's More!

- Conducted further security evaluations
- Hardware structure evaluations
- Secure kernel memory slab allocator
- ISV hardware caches and virtual address space

Takeaways

- **A taxonomy of transient execution attacks in the kernel**
  Active and passive attack scenarios
- **Two Perspective OS-HW interfaces ⇒ Tailored defense**
  - Data speculation views (DSVs) and Instruction speculation views (ISVs)
- **Security**
  - DSVs completely eliminate active attacks
  - ISVs can
    - Reduce the passive attack surface
    - Accelerate Spectre gadget discovery in the kernel
    - Swiftly mitigate newly discovered gadgets
- **Simple hardware and low execution overhead**
  - Execution overhead of 3.5% on LEBench and 1.2% on datacenter applications