Bunshin: Compositing Security Mechanisms through Diversification

Meng Xu, Kangjie Lu, Taesoo Kim, Wenke Lee

Georgia Institute of Technology

Memory Corruptions Are Costly…

Heartbleed bug 'will cost millions'

Revoking all SSL certificates leaked by Heartbleed will cost millions of dollars, according to Cloudflare, which provides services to website hosts

Image: Codenomicon

June 08, 2017

InfoSec 2017: Memory-based attacks on printers on the rise, says HP

Increase in use of printers as an attack vector for hackers: recommended that purchasing decisions include security considerations, not just price.

Name your phone “Nexus 5X %x.%x”

Battle against Memory Errors

Existing security mechanisms: W⊕R, ASLR, CFI

→ Not hard to by pass

Battle against Memory Errors

Existing security mechanisms: W⊕R, ASLR, CFI

→ Not hard to by pass

Protect all dangerous operation using **sanity checks**:

→ Auto-applied at compile time

```
void foo(T *a) {
   *a = 0x1234;
}
```

Sanitize →

```
void foo(T *a) {
   if(!is_valid_address(a) {
      report_and_abort();
   }
   *a = 0x1234;
}
```

Battle against Memory Errors

Memory Error	Main Causes	Defenses
Out-of-bound read/write	Lack of length check Integer overflow Format string bug Bad type casting	Softbound AddressSanitizer
Use-after-free	Dangling pointer Double free	CETS AddressSanitizer
Uninitialized read	Lack of initialization Data structure alignment Subword copying	MemorySanitizer
Undefined behaviors	Divide-by-zero Pointer misalignment Null-pointer dereference	UndefinedBehaviorSanitizer

Comprehensive Protection: Goal and Reality

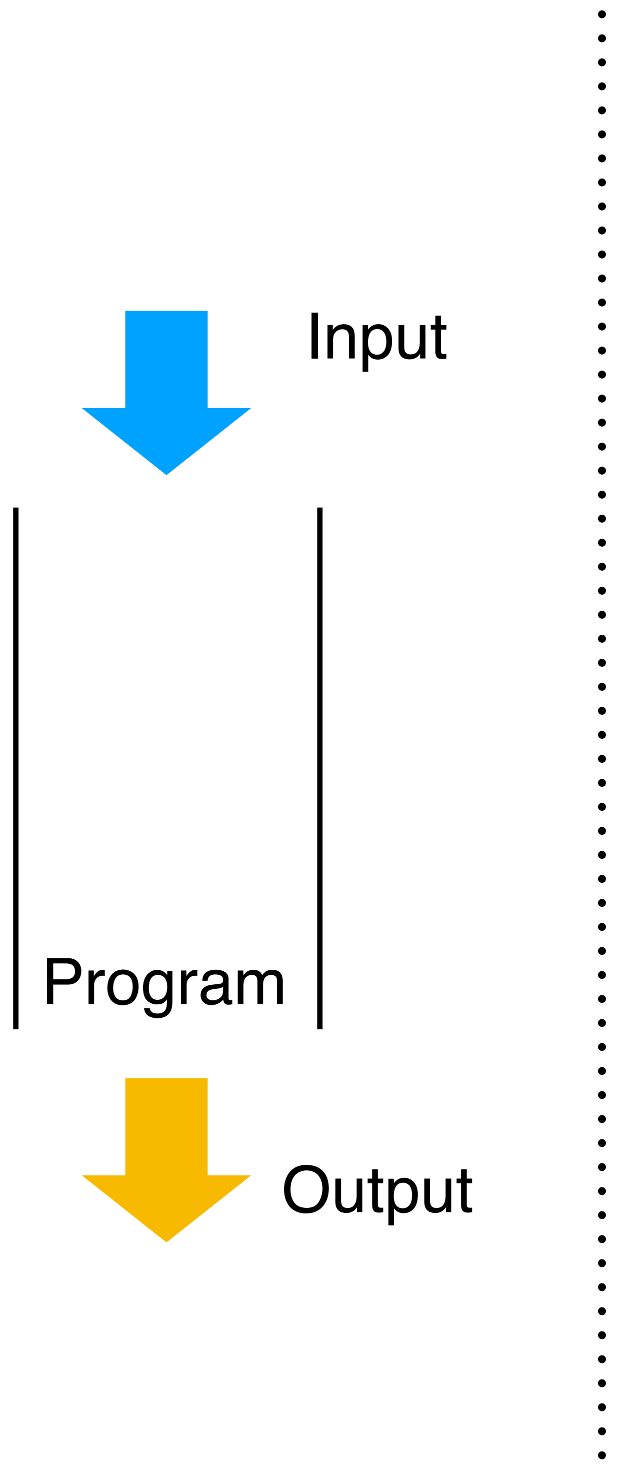
- Accumulated execution slowdown
  - Example: Softbound + CETS → **110%** slowdown
- Implementation conflicts
  - Example: AddressSanitizer and MemorySanitizer

Comprehensive Protection with Bunshin

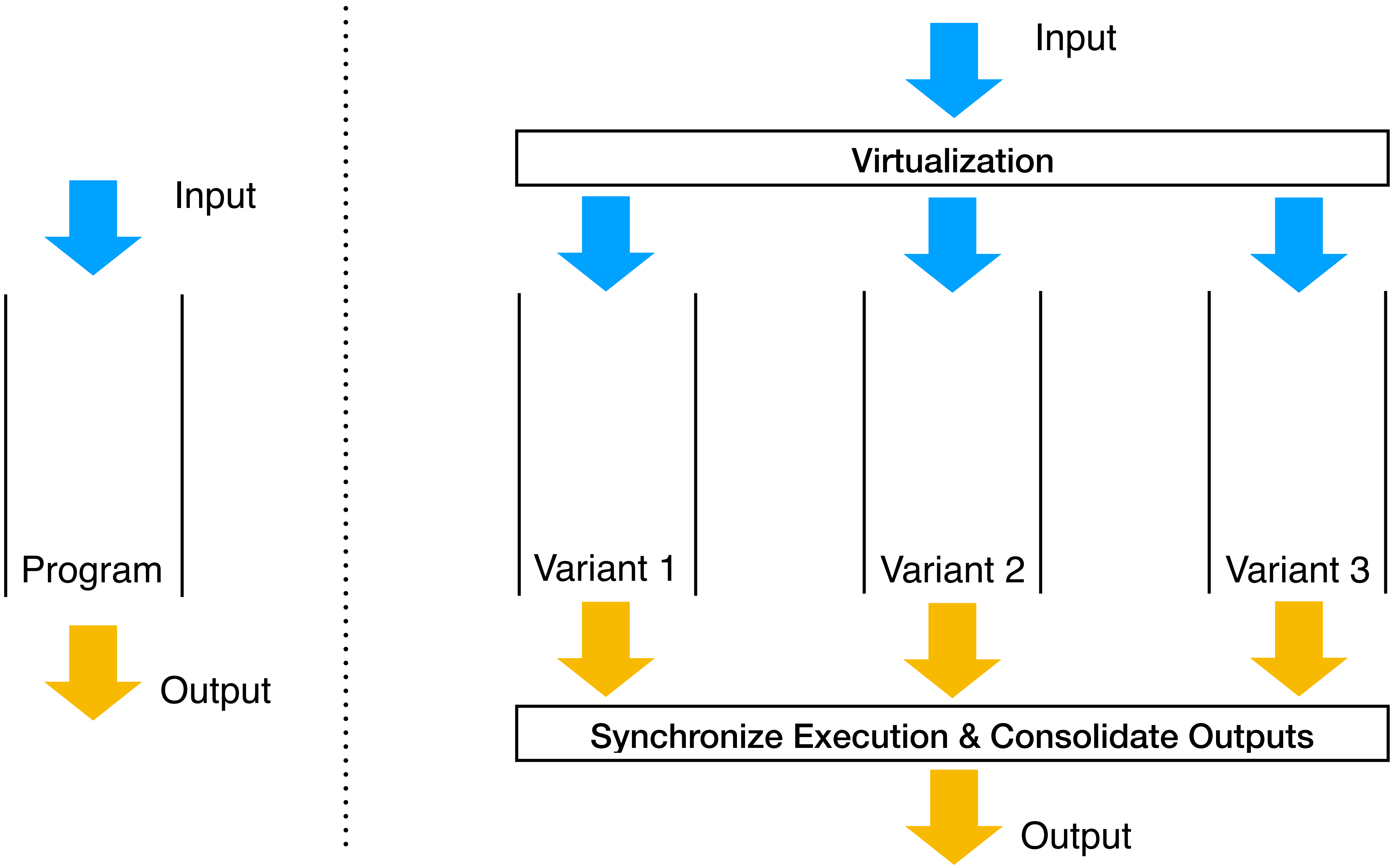
- Accumulated execution slowdown
  - Example: Softbound + CETS → **110%** slowdown
  - Bunshin: Reduce to 60% or 40% (depends on the config)

- Implementation conflicts
  - Example: AddressSanitizer and MemorySanitizer
  - Bunshin: Seamlessly enforce conflicting sanitizers

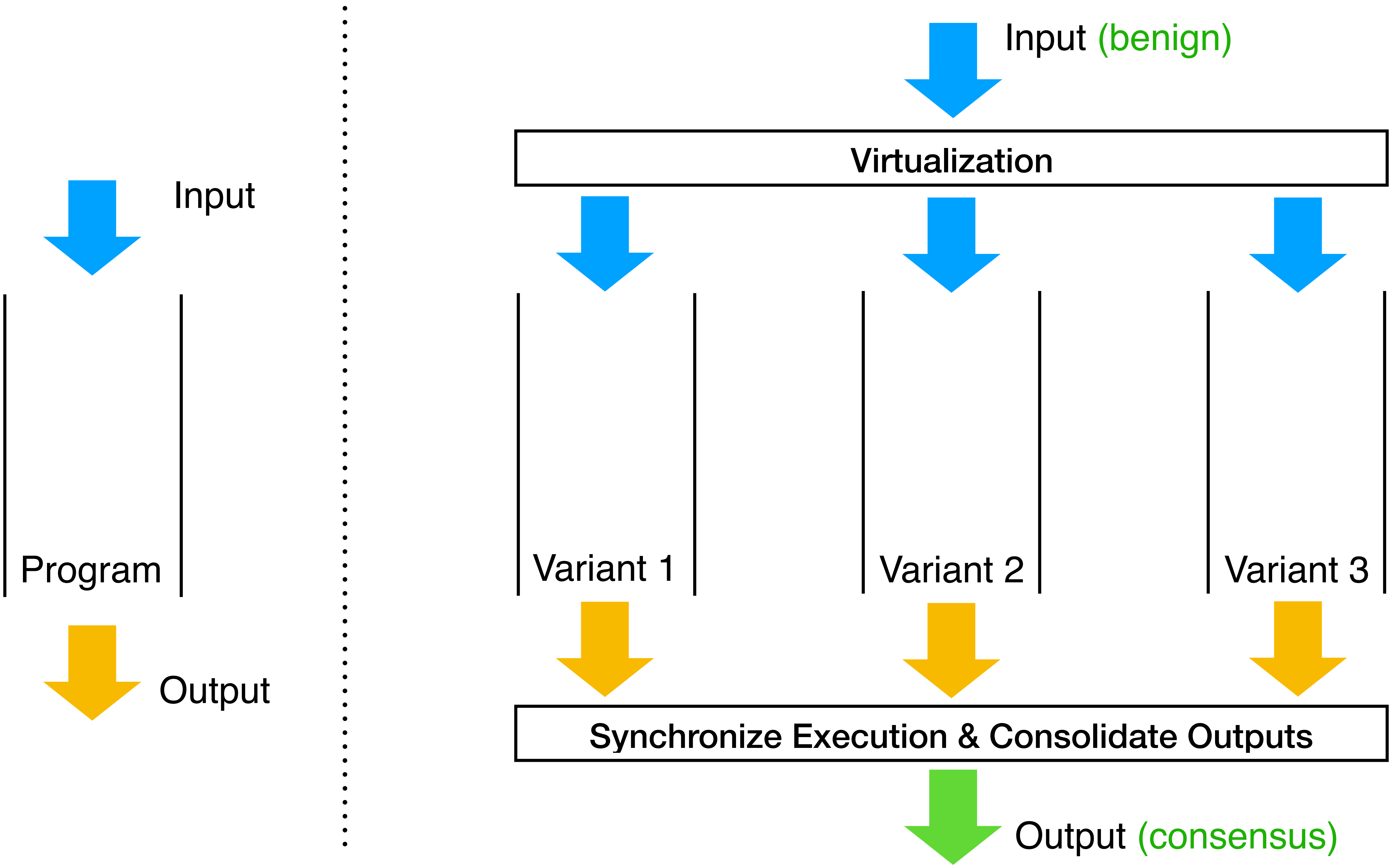
The N-Version Way



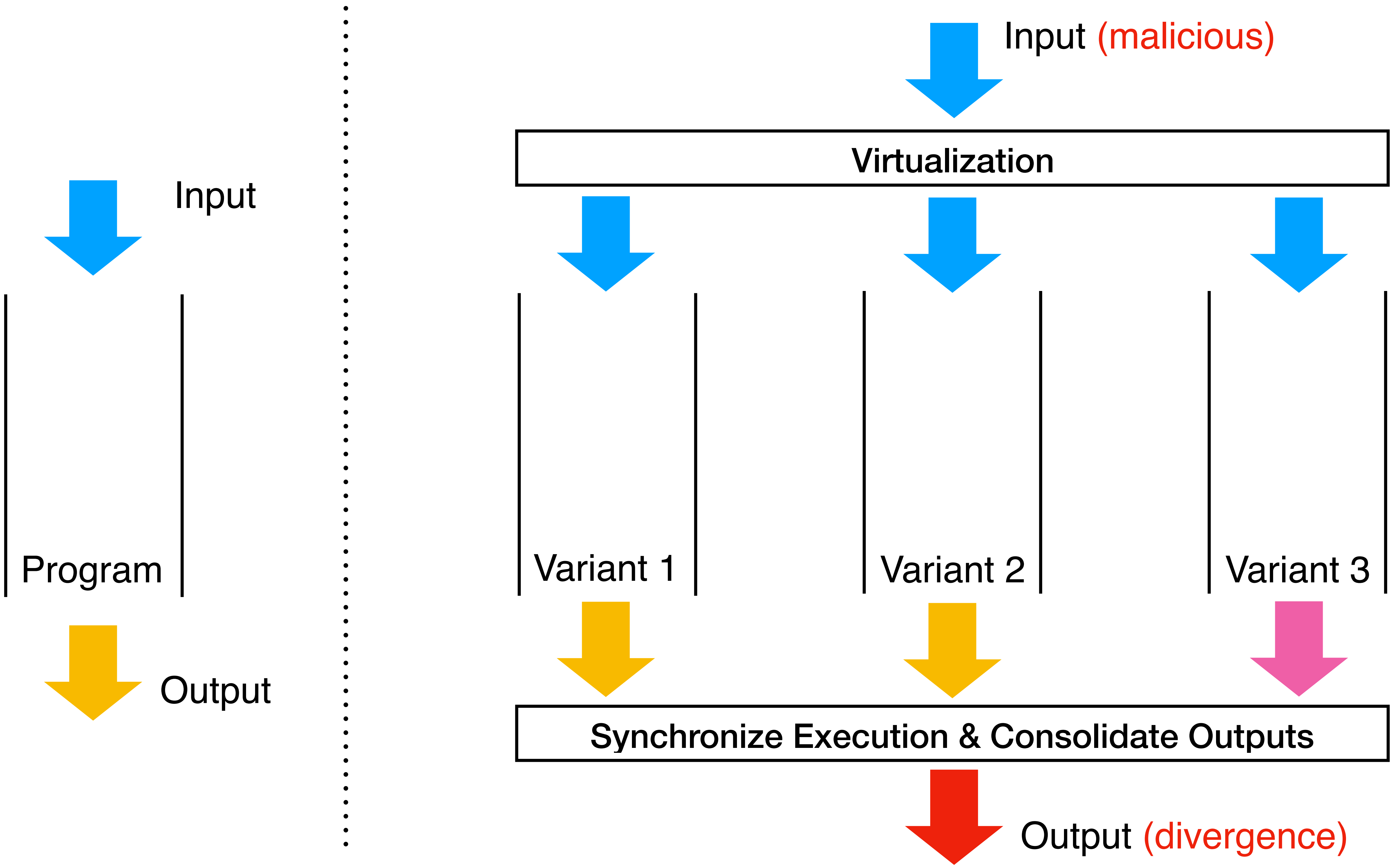
The N-Version Way



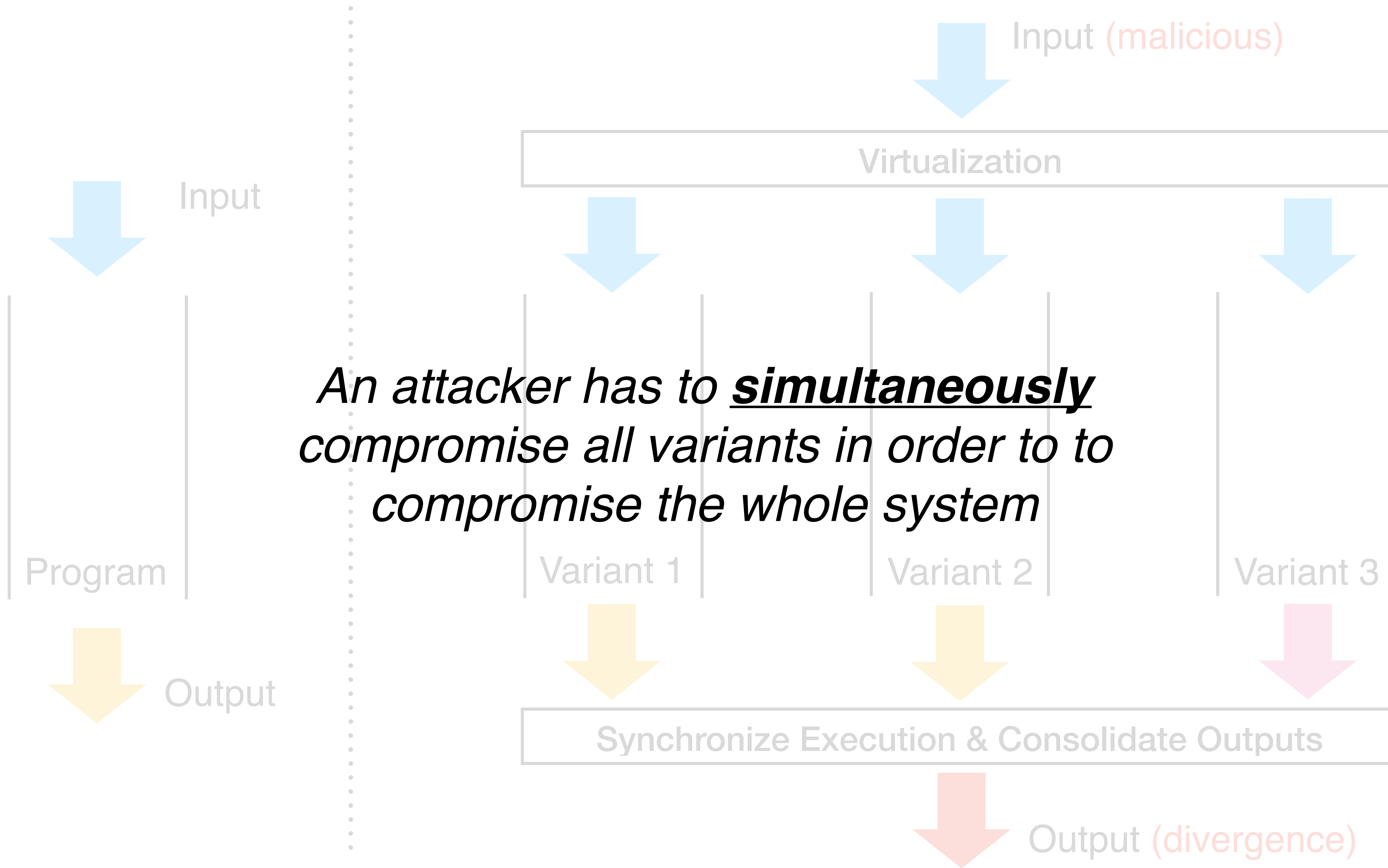
The N-Version Way



The N-Version Way



The N-Version Way



An attacker has to ***simultaneously*** *compromise all variants in order to to compromise the whole system*

Similar Ideas

- Two variants placed in disjoint memory partitions [*N-Variant Systems*]
- Two variants with stacks growing in different directions [*Orchestra*]
- Multiple variants with randomized heap object locations [*DieHard*]
- Multiple versions of the same program [*Varan, Mx*]

Bunshin Overview

- Goal:
    - Reduce slowdown caused by security mechanisms
    - Enable different or even conflicting mechanisms

Challenges for Bunshin

- How to generate these variants?
- What properties they should have?
- How to make them appear as one to outsiders?
- What is a “behavior” and what is a divergence?
- What if the sanitizers introduces new behaviors?
- Multi-threading support?

Variant Generation Intuitions

- Scope of protection required → Sanitizers selected

Memory Error	Defenses
Out-of-bound read/write	Softbound, AddressSanitizer
Use-after-free	CETS, AddressSanitizer
Uninitialized read	MemorySanitizer
Undefined behaviors	UndefinedBehaviorSanitizer

- Instrumented checks by each sanitizer

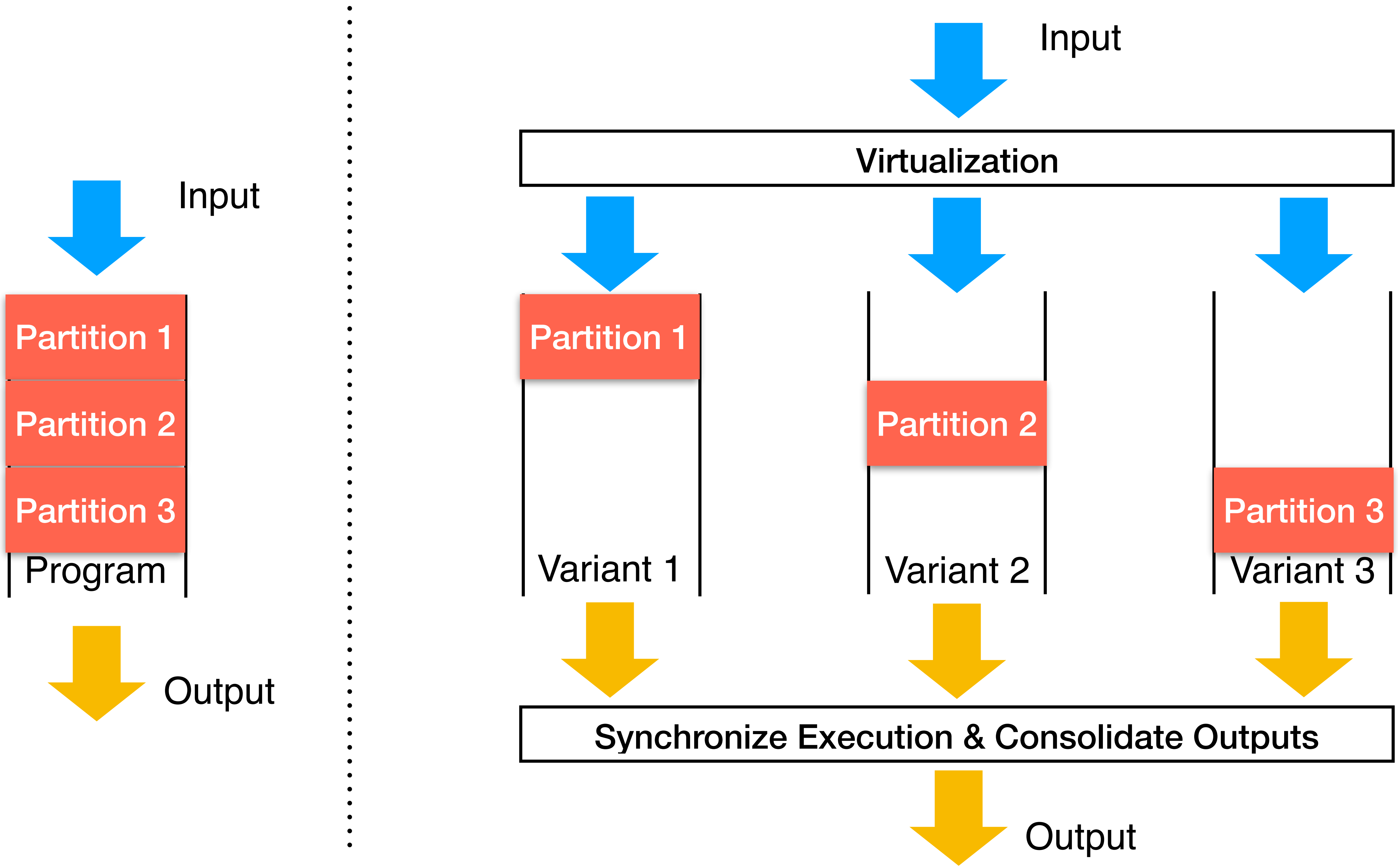
```
void foo(T *a) {
  if(!is_valid_address(a) {
    report_and_abort();
  }
  *a = 0x1234;
}
```

```
void bar(T *b) {
  if(!is_valid_address(b) {
    report_and_abort();
  }
  *b = 0x5678;
}
```

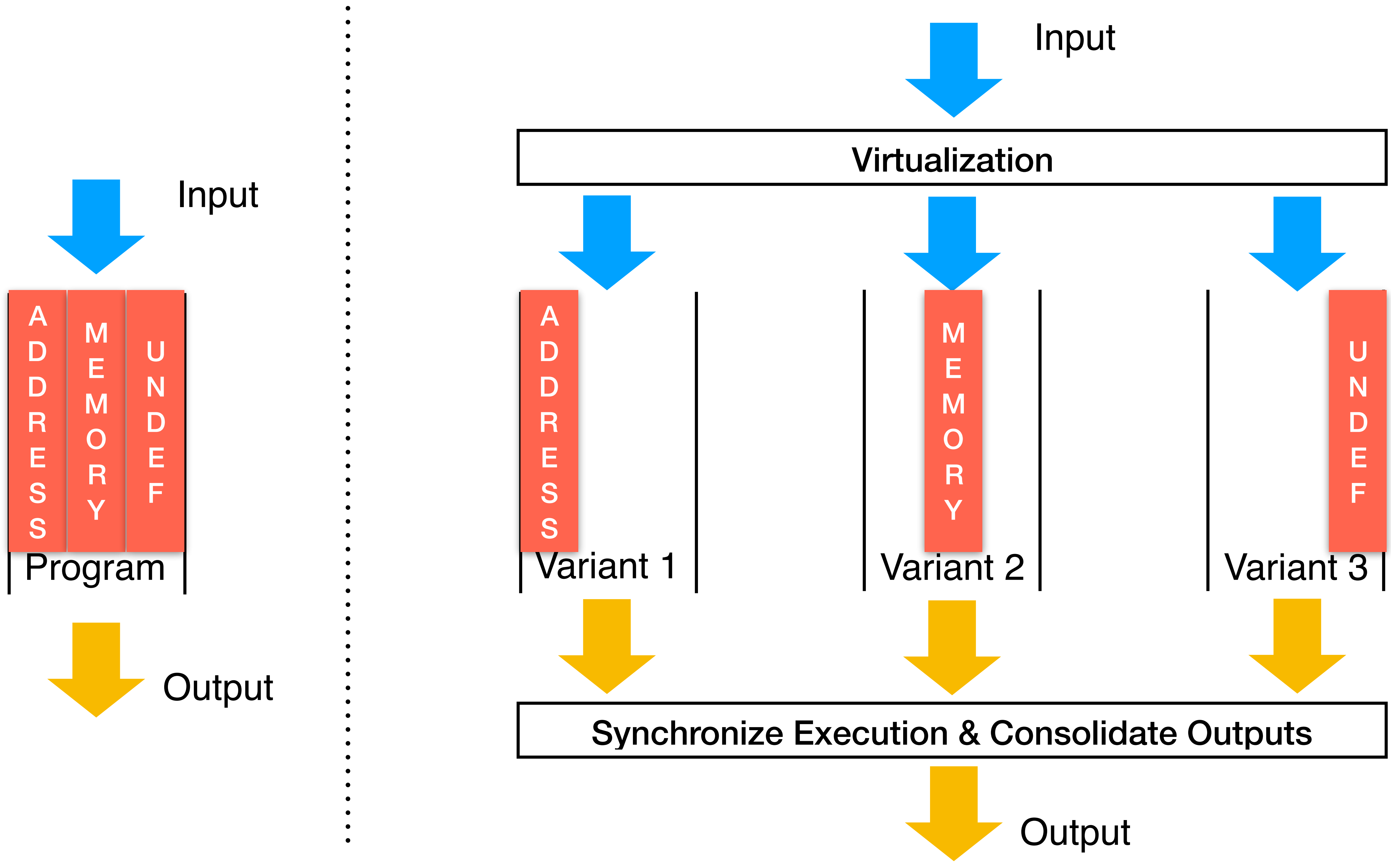
Variant Generation Principles

- Check distribution
- Sanitizer distribution

Check Distribution



Sanitizer Distribution



Cost Profiling

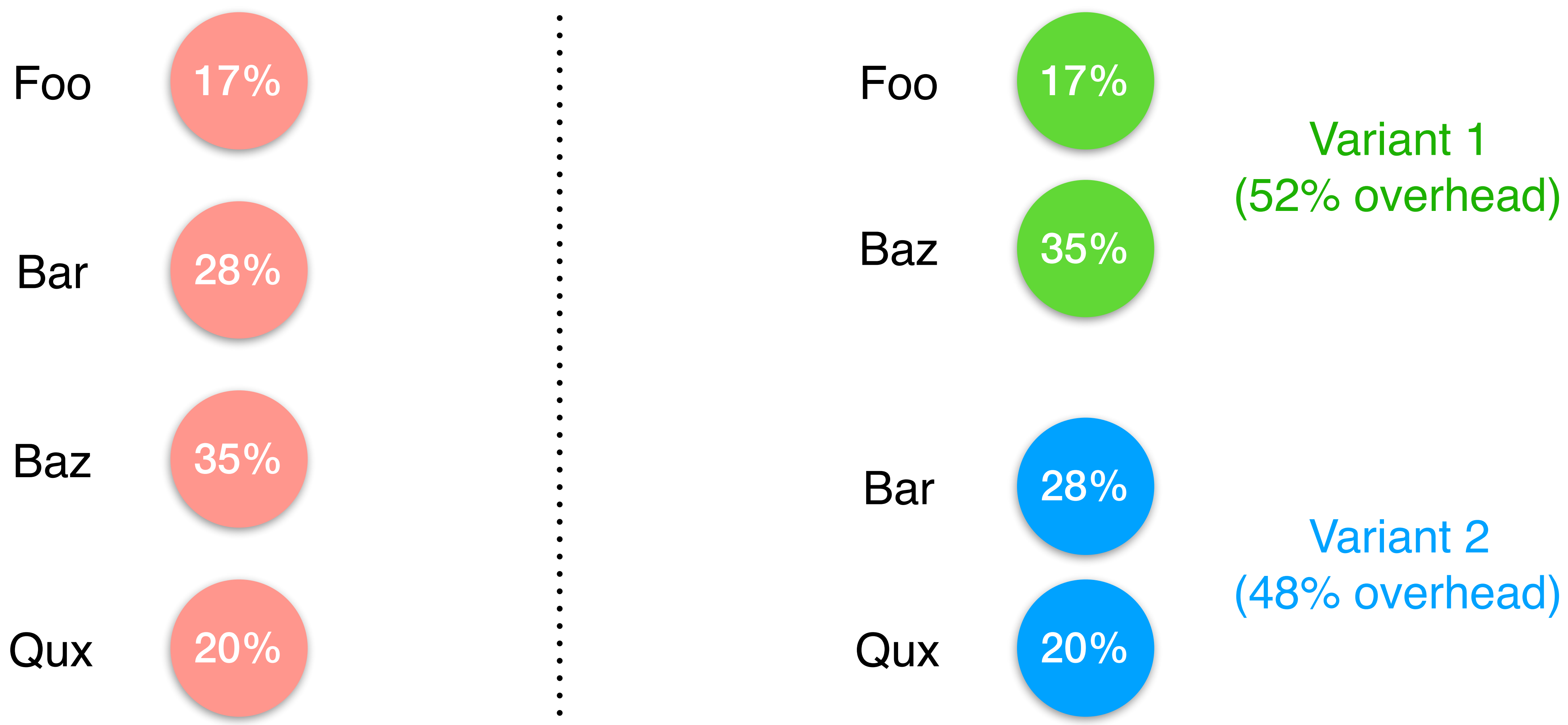
- Calculate the slowdown caused by the sanity checks

```
void foo(T *a) {
   timing_start();
   *a = 0x1234;
   timing_end();
}
```

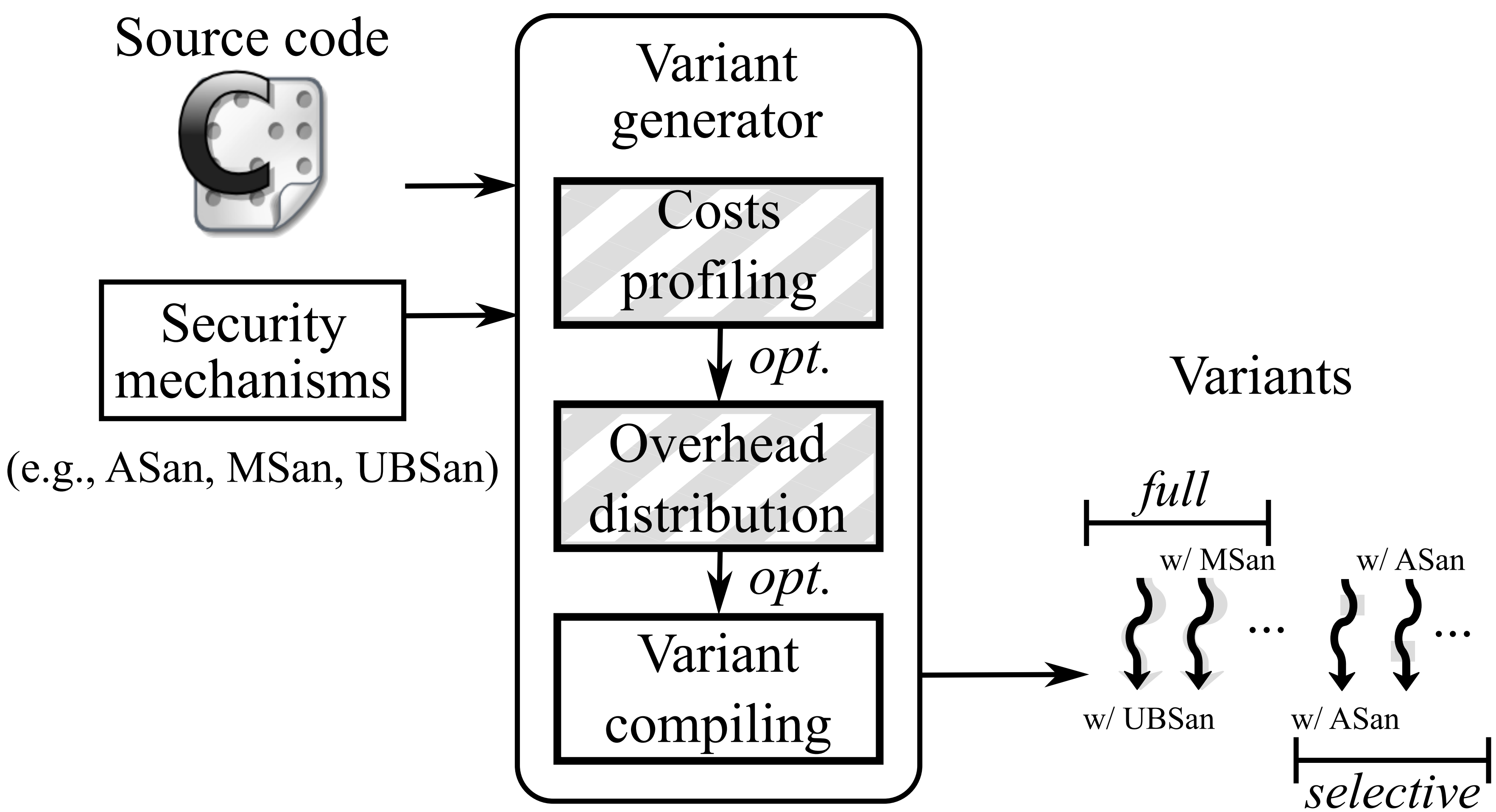
```
void foo(T *a) {
   timing_start();
   if(!is_valid_address(a) {
      report_and_abort();
   }
   *a = 0x1234;
   timing_end();
}
```

Cost Distribution

- Equally distribute overhead to variants so that they execute at the same speed



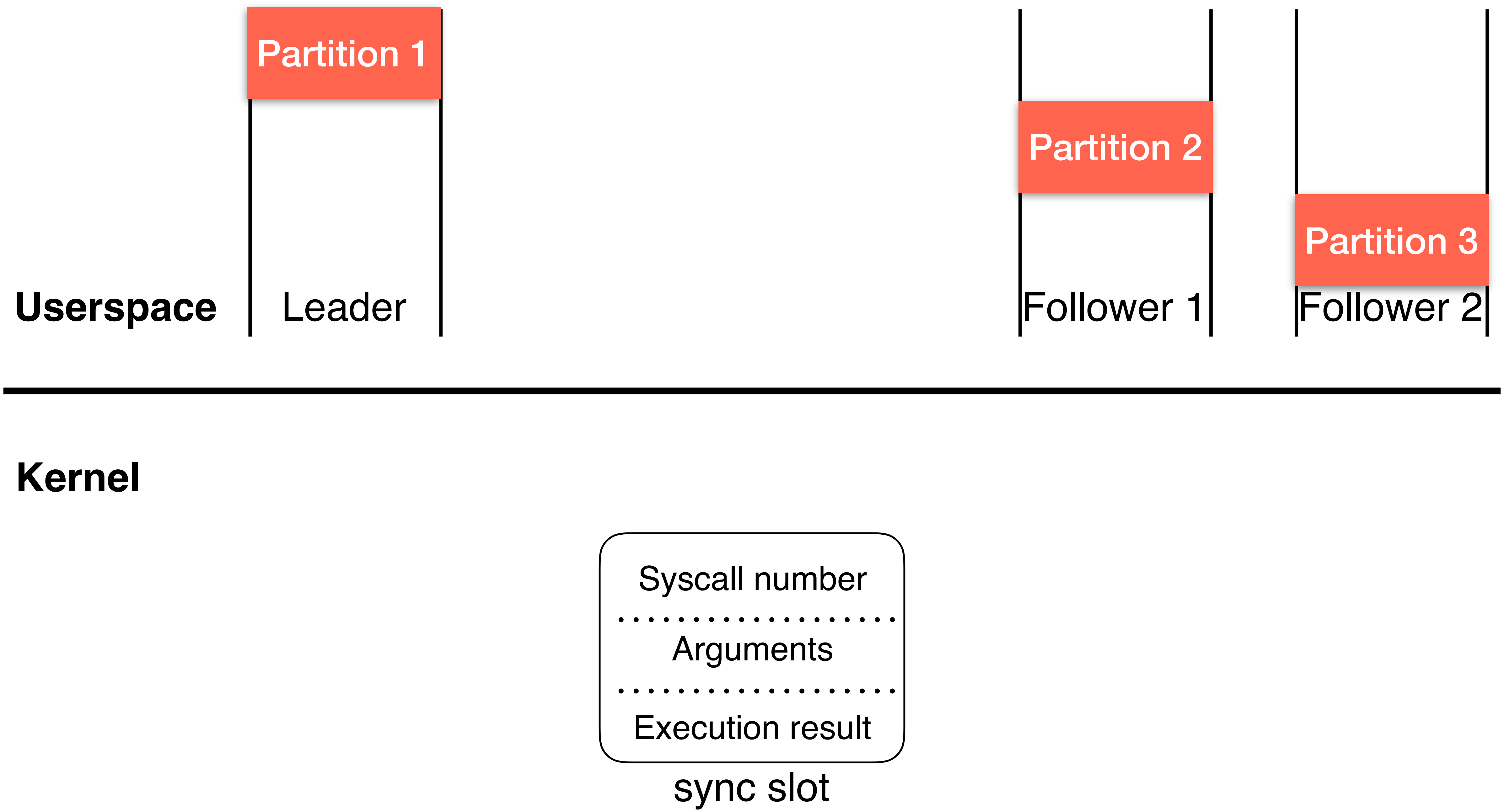
Variant Generation Process



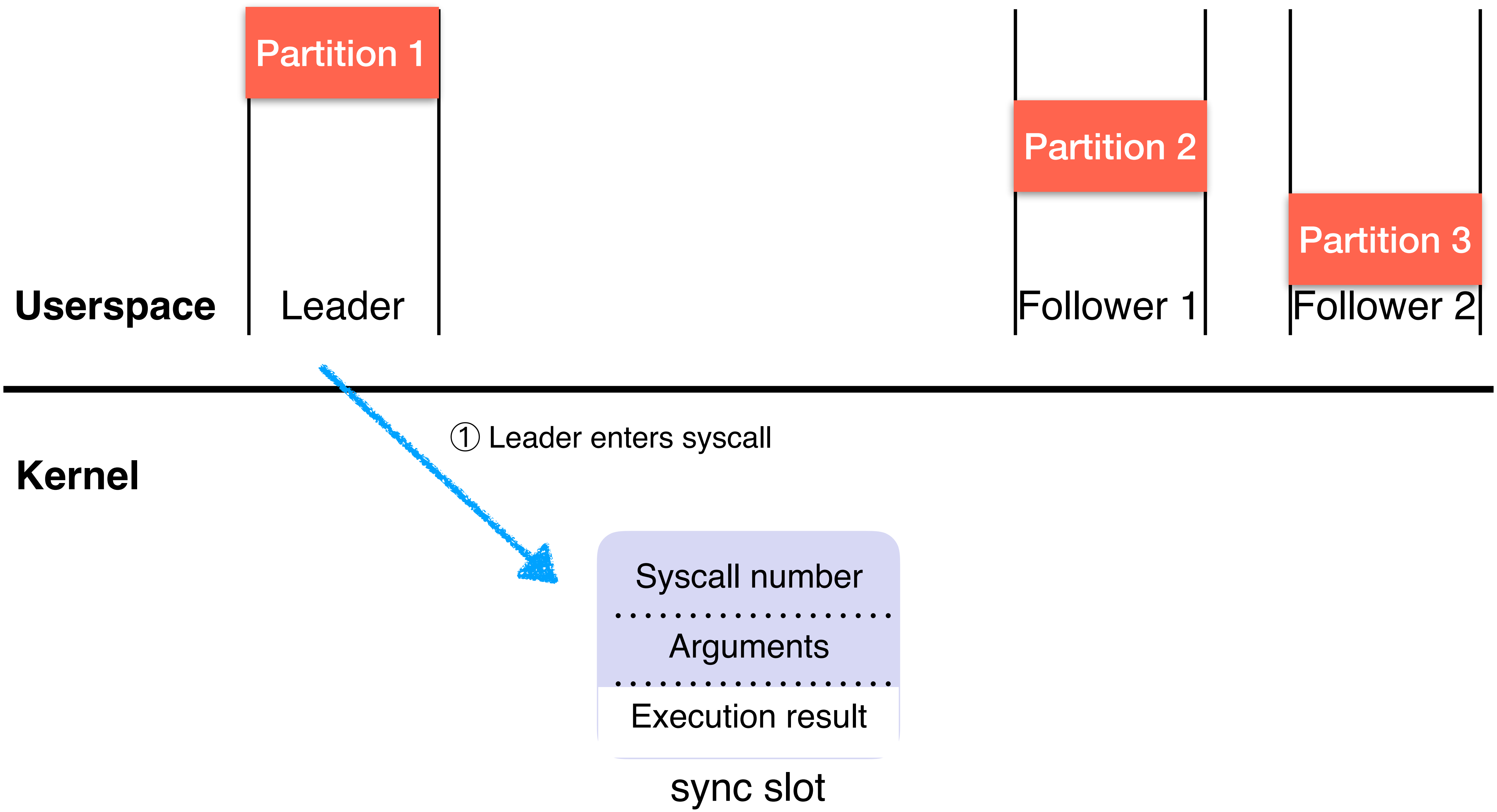
Variant Sync Considerations

- What is a behavior and what is a divergence?
  - System call (both order and arguments)
- How to hook it?
  - By patching the system call table with a kernel module
- What if different sanitizers introduce different system calls?
  - Sync only when a program is in its `main` function
  - Do not check system calls for memory management

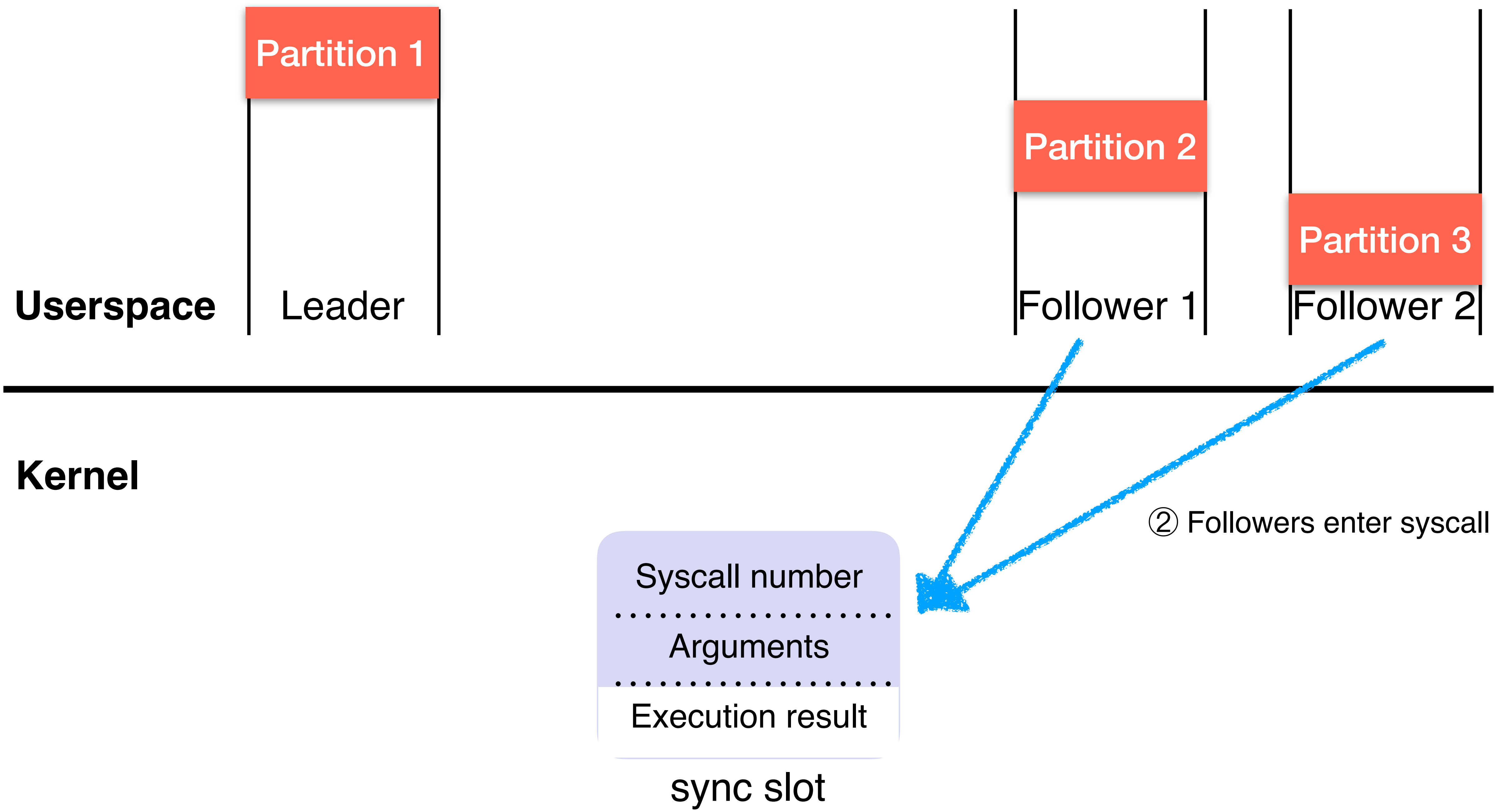
System Call Synchronization



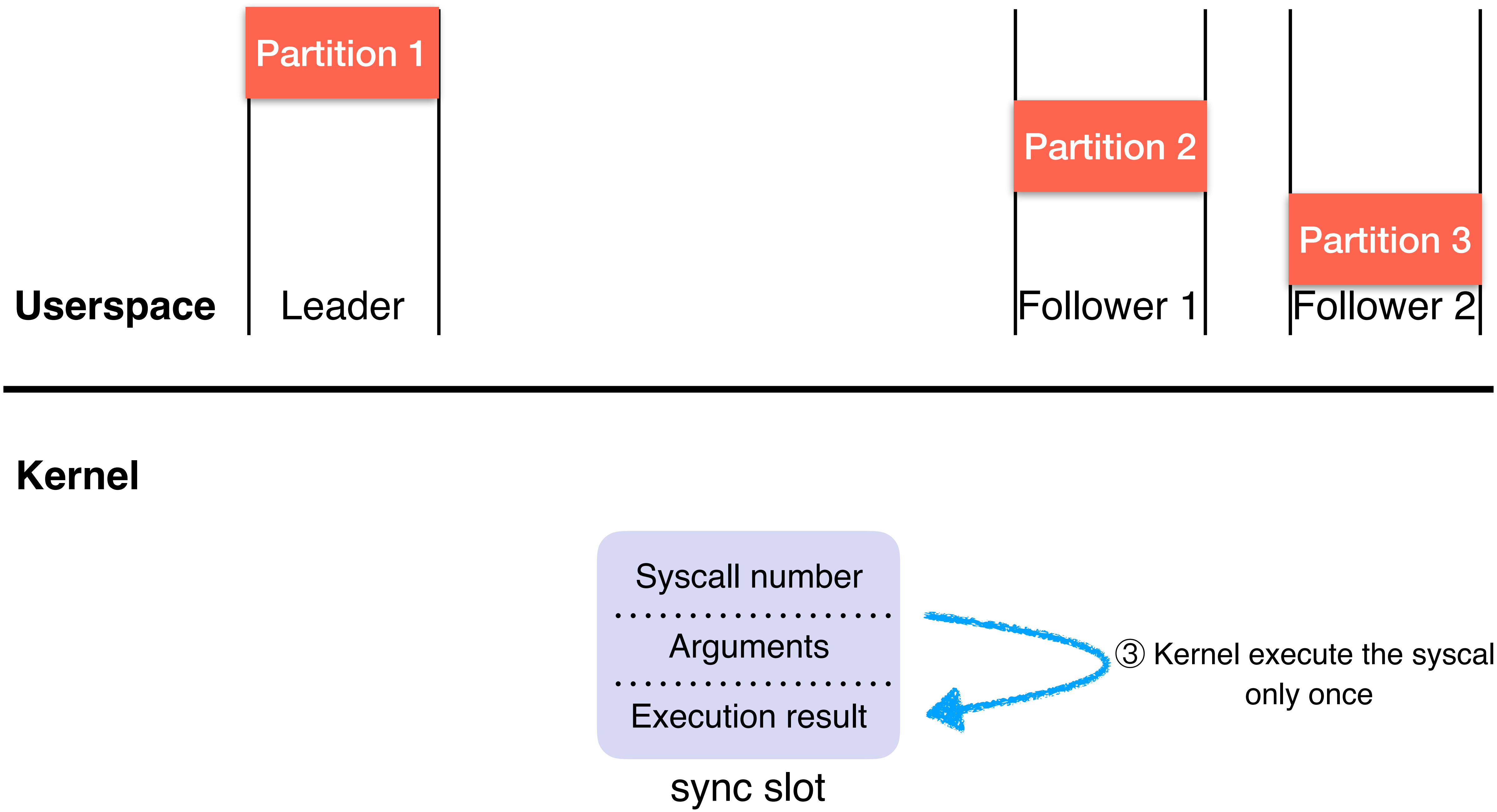
System Call Synchronization



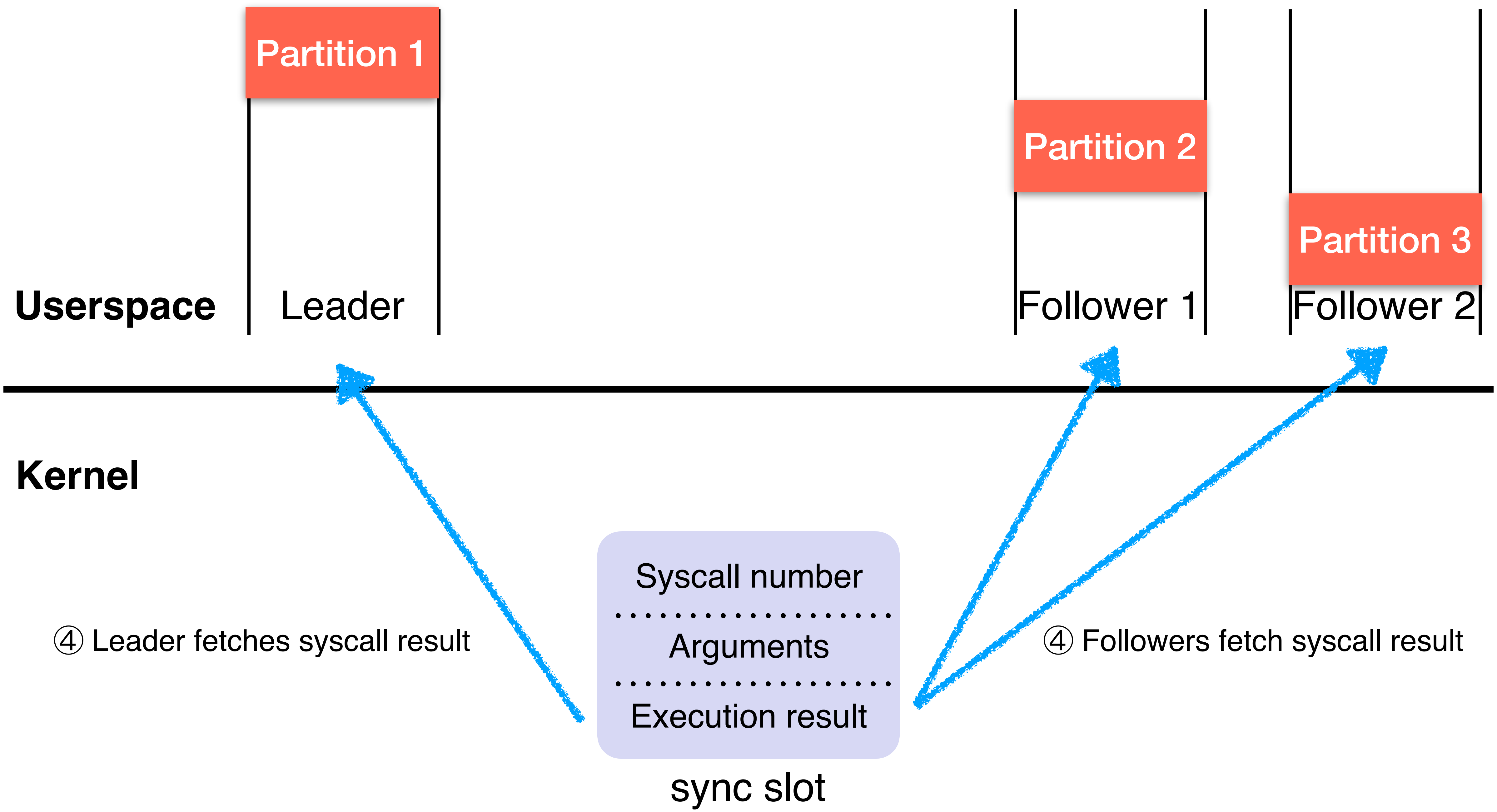
System Call Synchronization



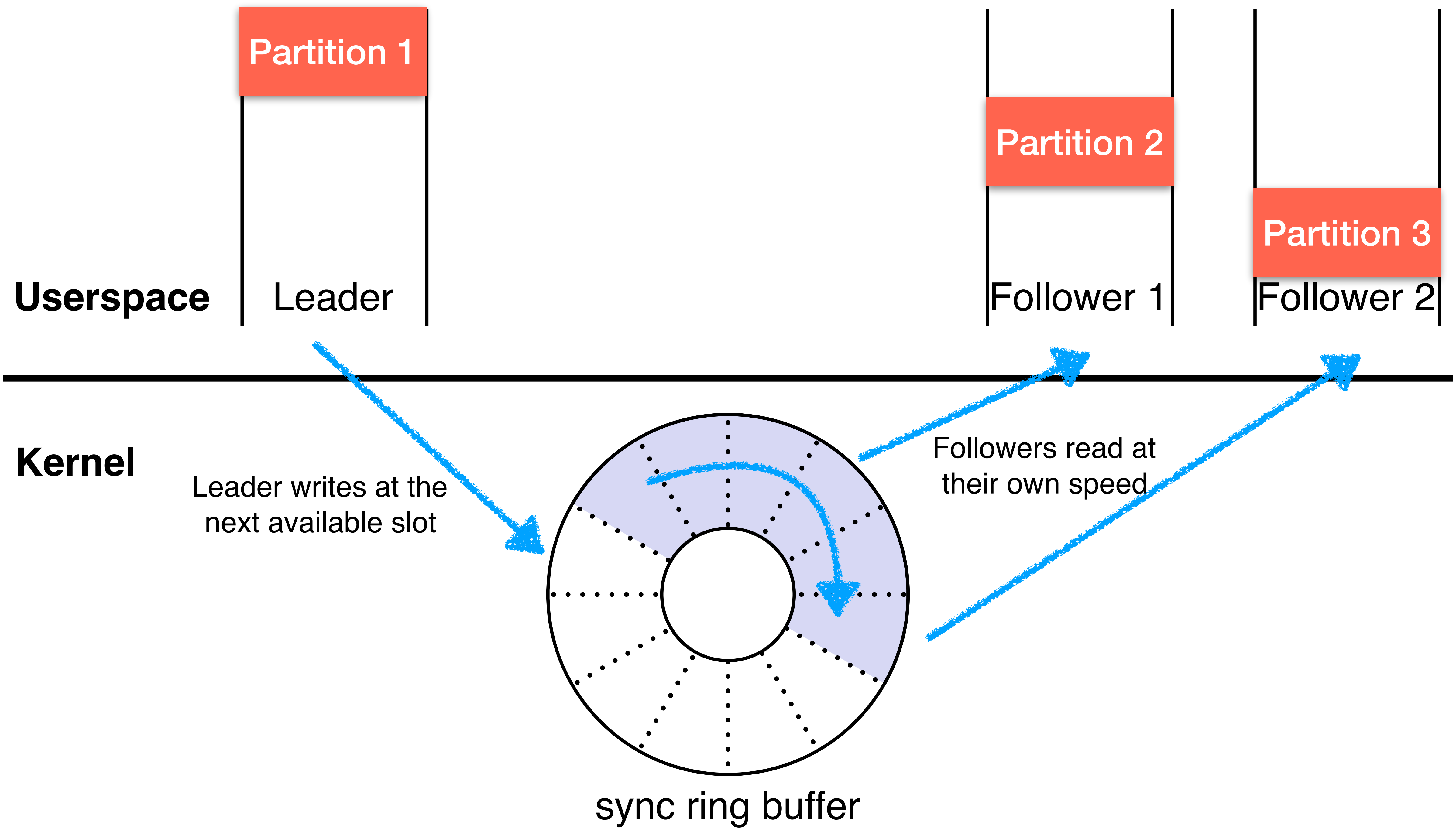
System Call Synchronization



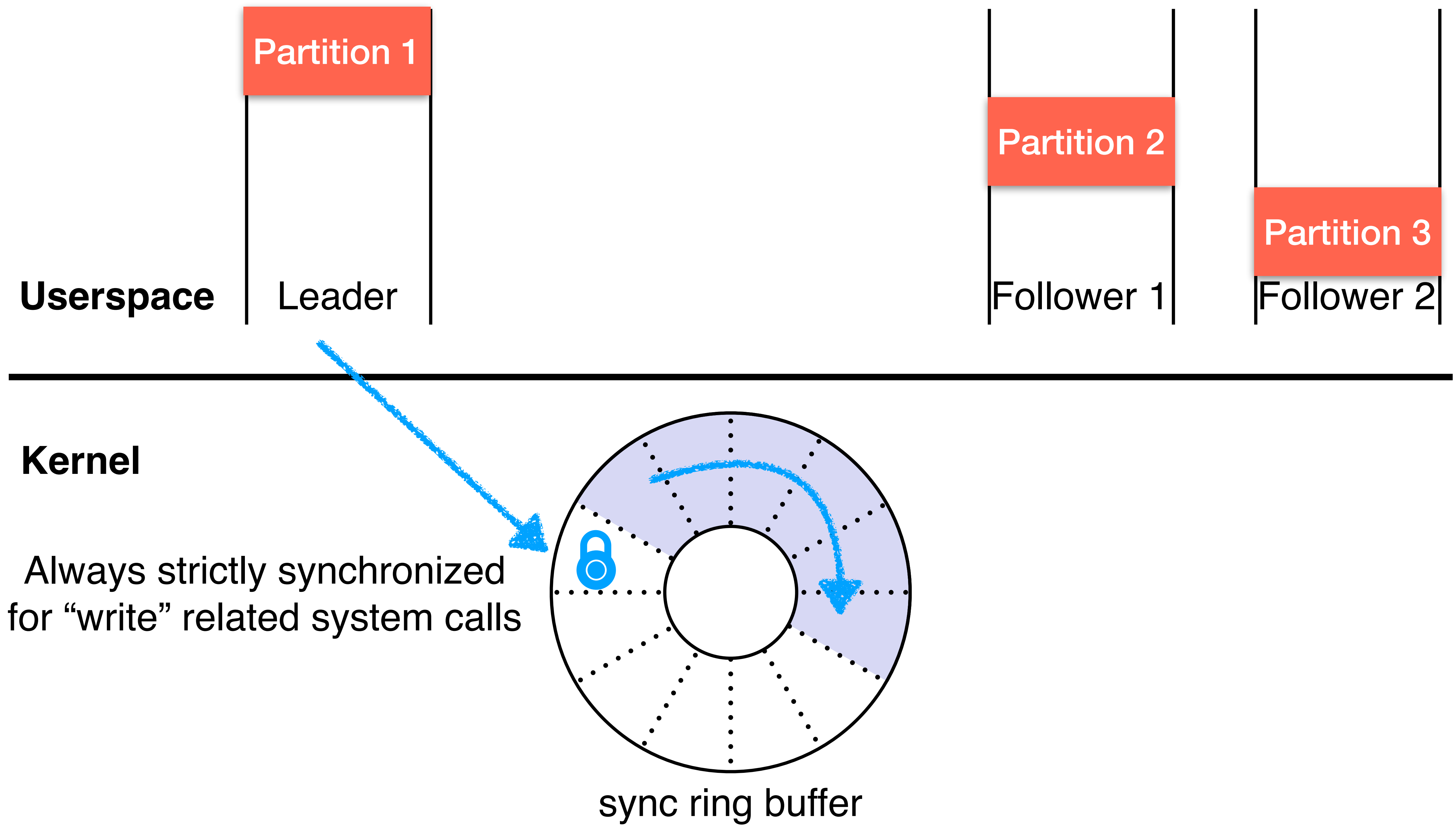
System Call Synchronization



Strict and Selective Lockstep



Strict and Selective Lockstep

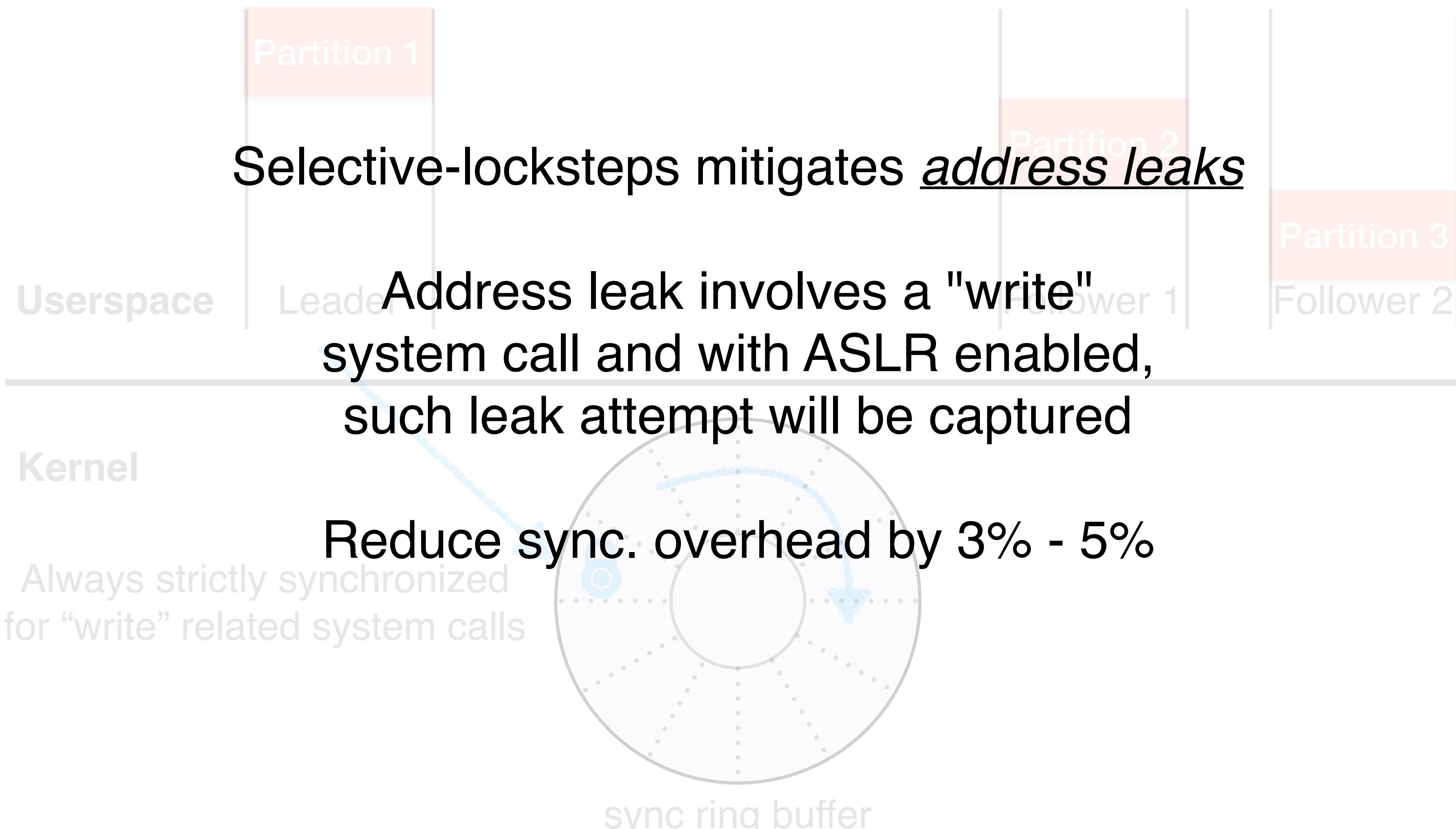


Strict and Selective Lockstep

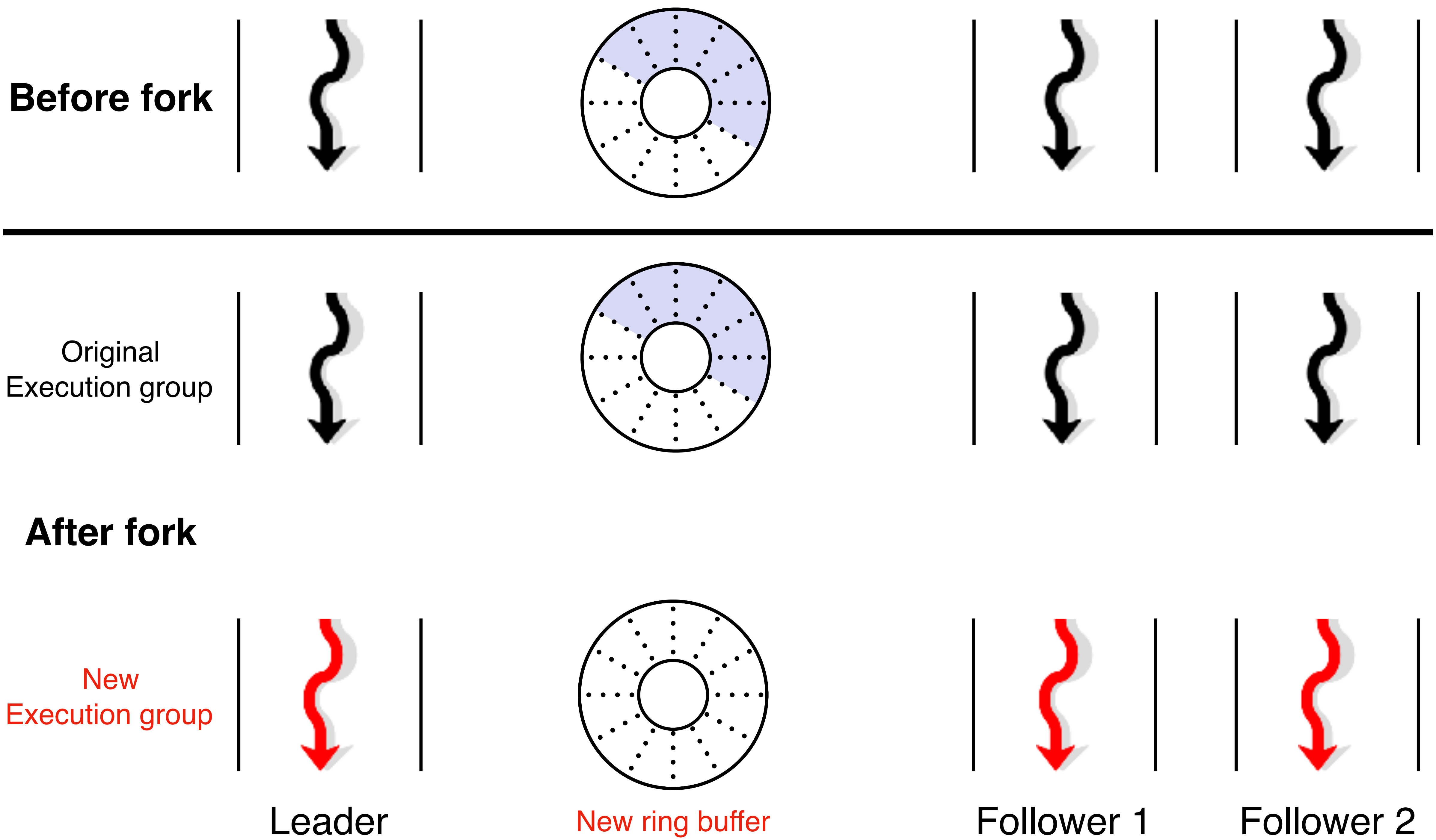
Selective-locksteps mitigates *address leaks*

Address leak involves a "write" system call and with ASLR enabled, such leak attempt will be captured

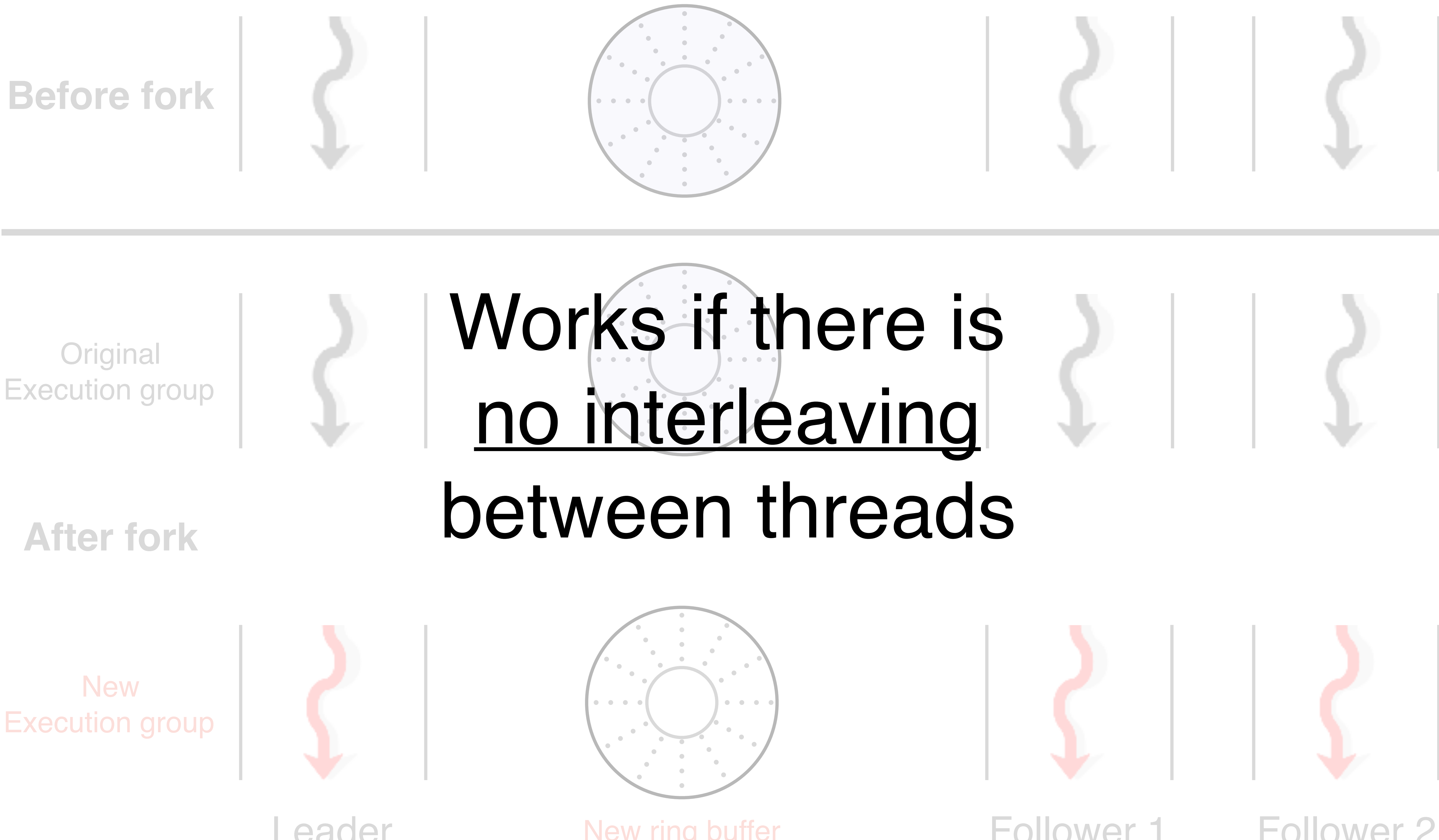
Reduce sync. overhead by 3% - 5%



Multi-threading Support

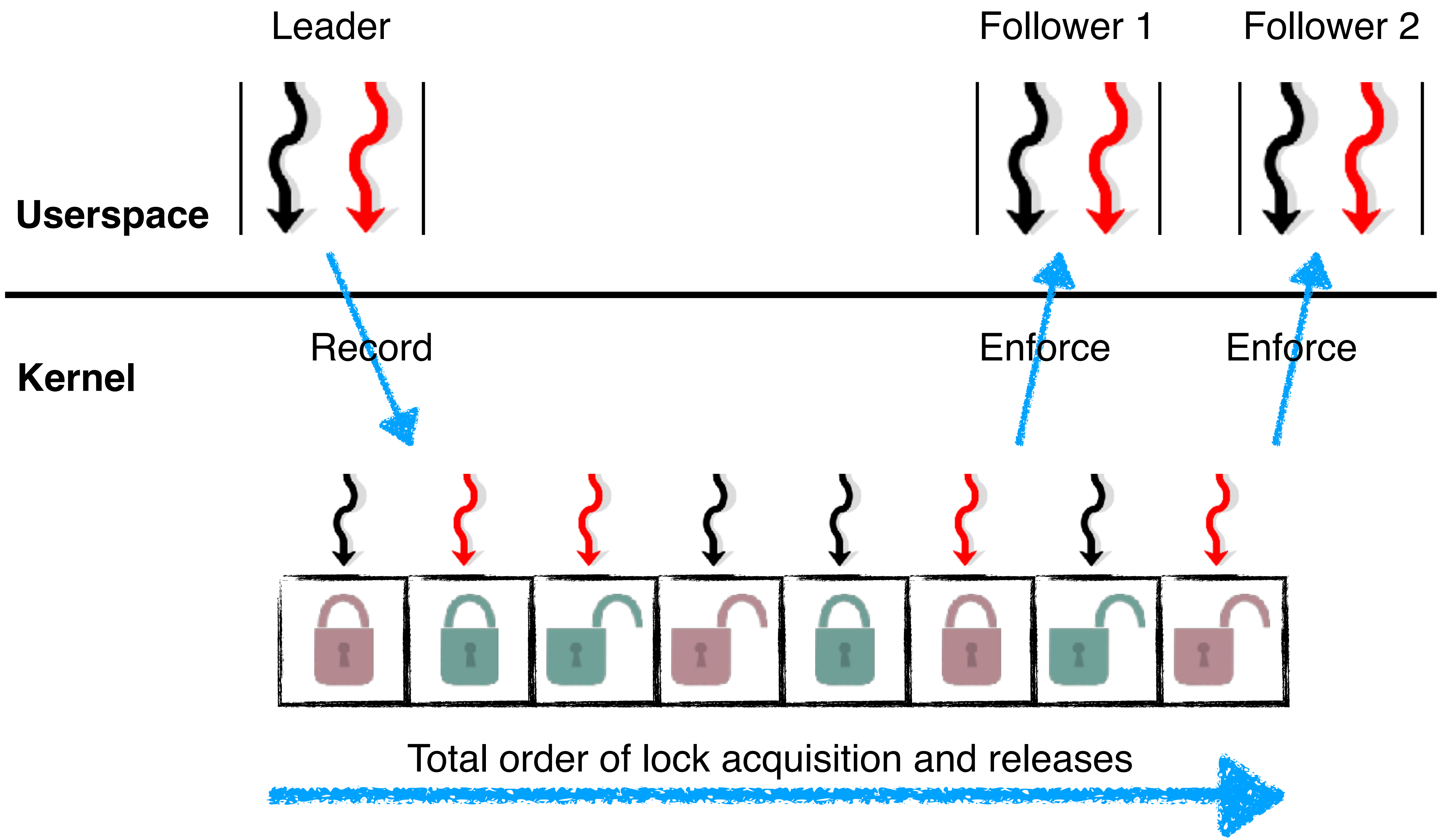


Multi-threading Support



Works if there is
<u>no interleaving</u>
between threads

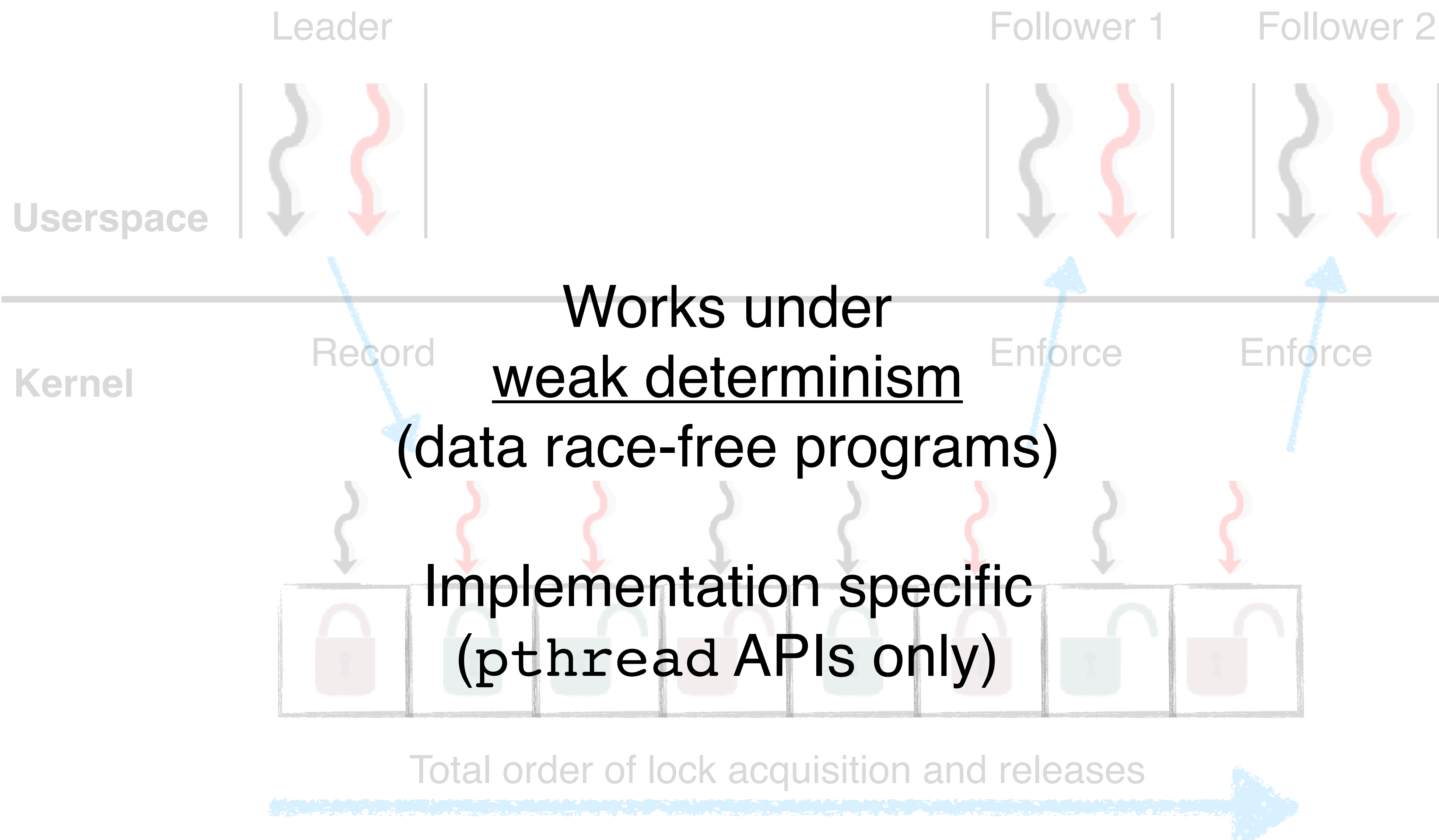
Multi-threading Support



Multi-threading Support

Works under
<u>weak determinism</u>
(data race-free programs)

Implementation specific
(`pthread` APIs only)



Evaluate Bunshin

- Robustness and Security
- Efficiency and Scalability
- Protection Distribution Case Studies

Robustness

Benchmark	Single/Multi-thread	Featuer	Pass ?	
SPEC CPU2006	Single	CPU Intensive	✓	
SPLASH-2x	Multi		✓	
PARSEC	Multi		✗	6 out of 13
lighttpd	Single	I/O Intensive	✓	
nginx	Multi		✓	
python, php	Single	Interpreter	✓	

Security

- RIPE Benchmark

Config	Succeed	Probabilistic	Failed	Not possible
Default	114	16	720	2990
AddressSanitizer	8	0	842	2990
Bunshin	8	0	842	2990

- Real-world CVEs

Config	CVE	Exploits	Sanitizer	Detect
nginx-1.4.0	2013-2028	Blind ROP	AddressSanitizer	✓
cpython-2.7.10	2016-5636	Integer overflow	AddressSanitizer	✓
php-5.6.6	2015-4602	Type confusion	AddressSanitizer	✓
openssl-1.0.1a	2014-0160	Heartbleed	AddressSanitizer	✓
httpd-2.4.10	2014-3581	Null dereference	UndefinedBehaviorSanitizer	✓

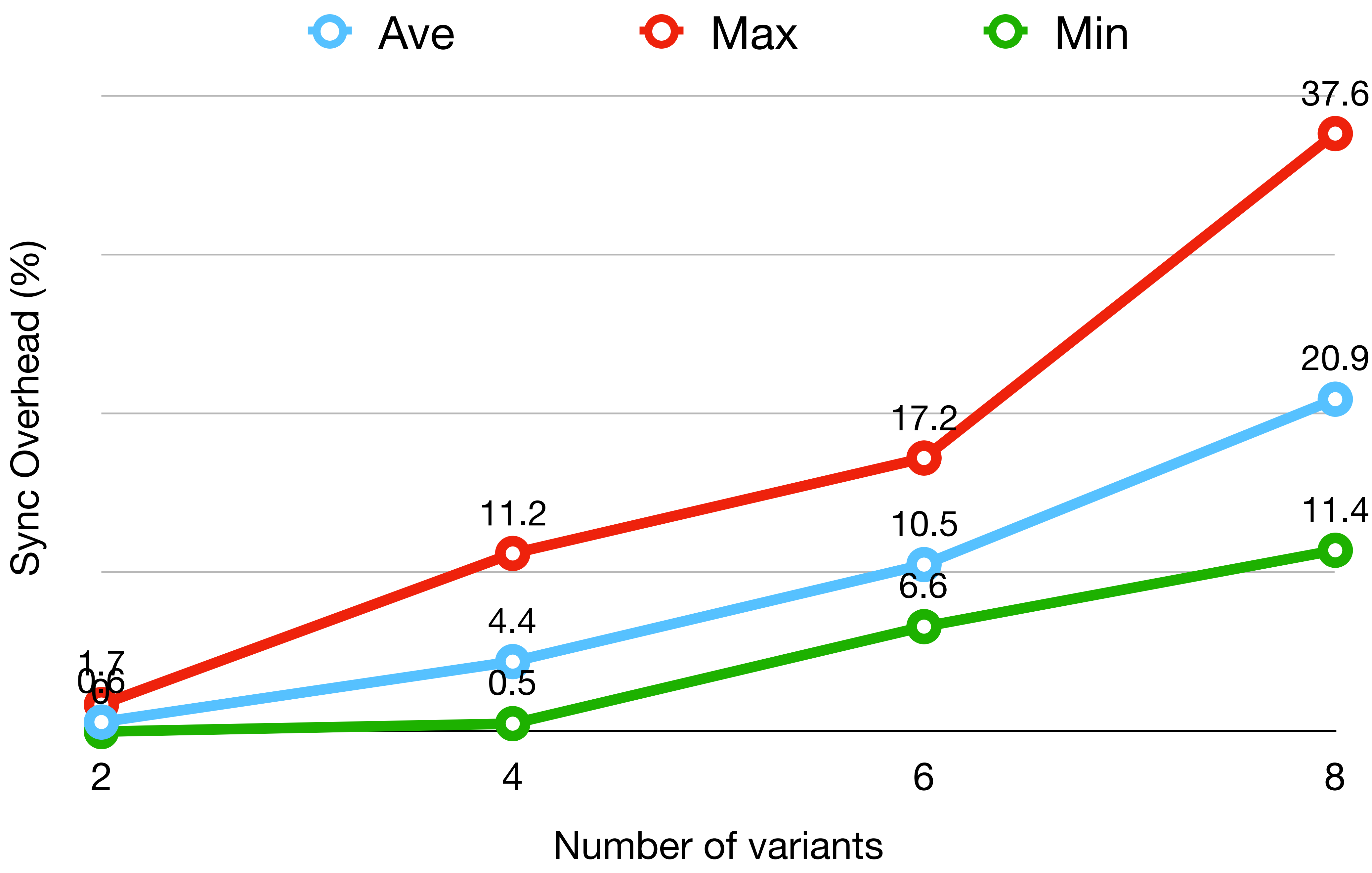
Performance

Benchmark	Items	Strict-Lockstep	Selective-Lockstep
SPEC CPU2006 (19 Programs)	Max	17.5%	14.7%
	Min	1.6%	1.0%
	Ave	8.6%	5.6%
SPLASH-2X / PARSEC (19 Programs)	Max	21.4%	18.9%
	Min	10.7%	6.6%
	Ave	16.6%	14.5%
`lighttpd` 1MB File Request	Ave	1.44%	1.21%
`nginx` 1MB File Request	Ave	1.71%	1.41%

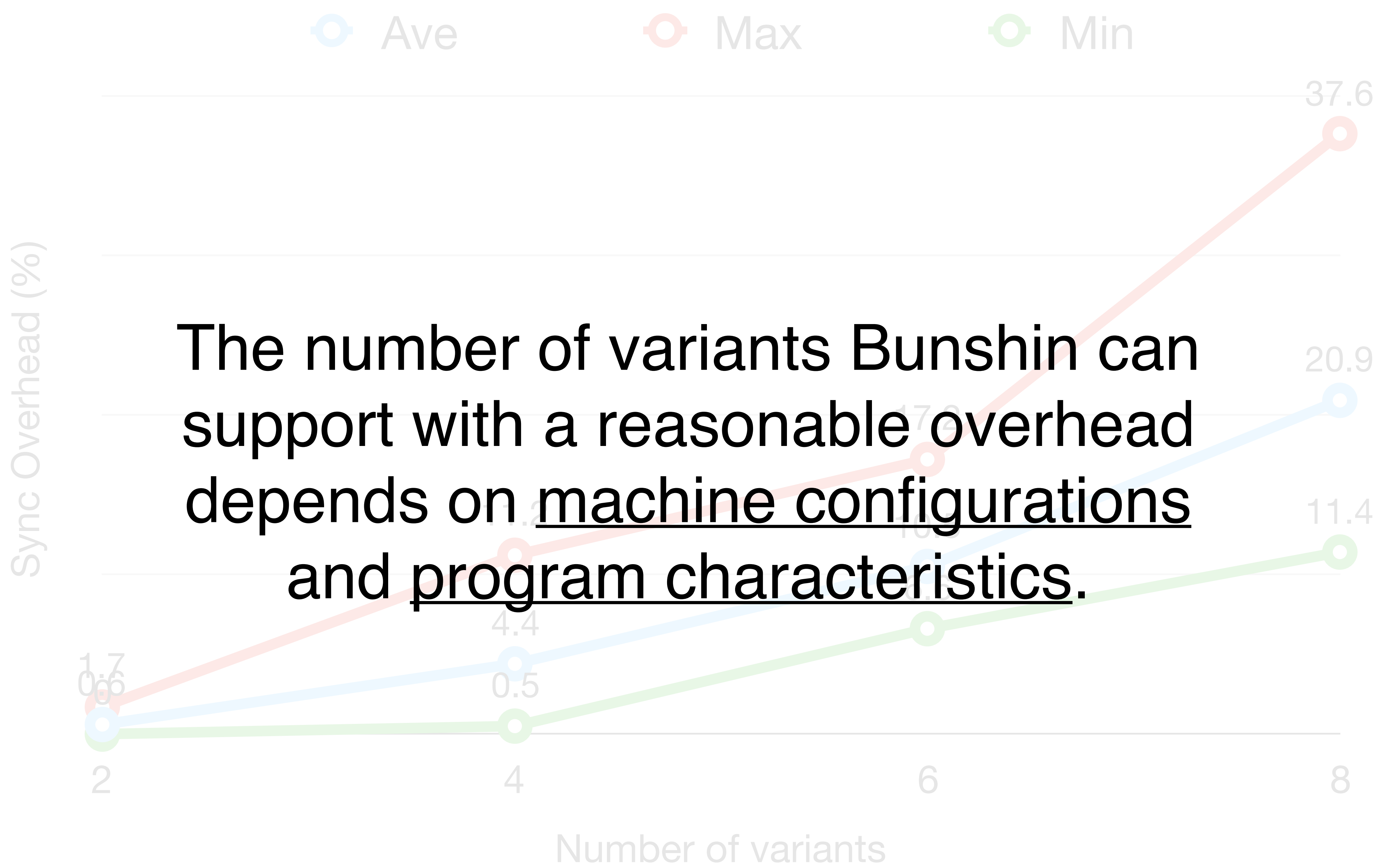
Performance Highlights

- Low overhead (5% - 16%) for standard benchmarks
- Negligible overhead (<= 2%) for server programs
- Extra cost of ensuring weak determinism is 8%
- Selective-lockstep saves around 3% overhead

Scalability - Number of Variants

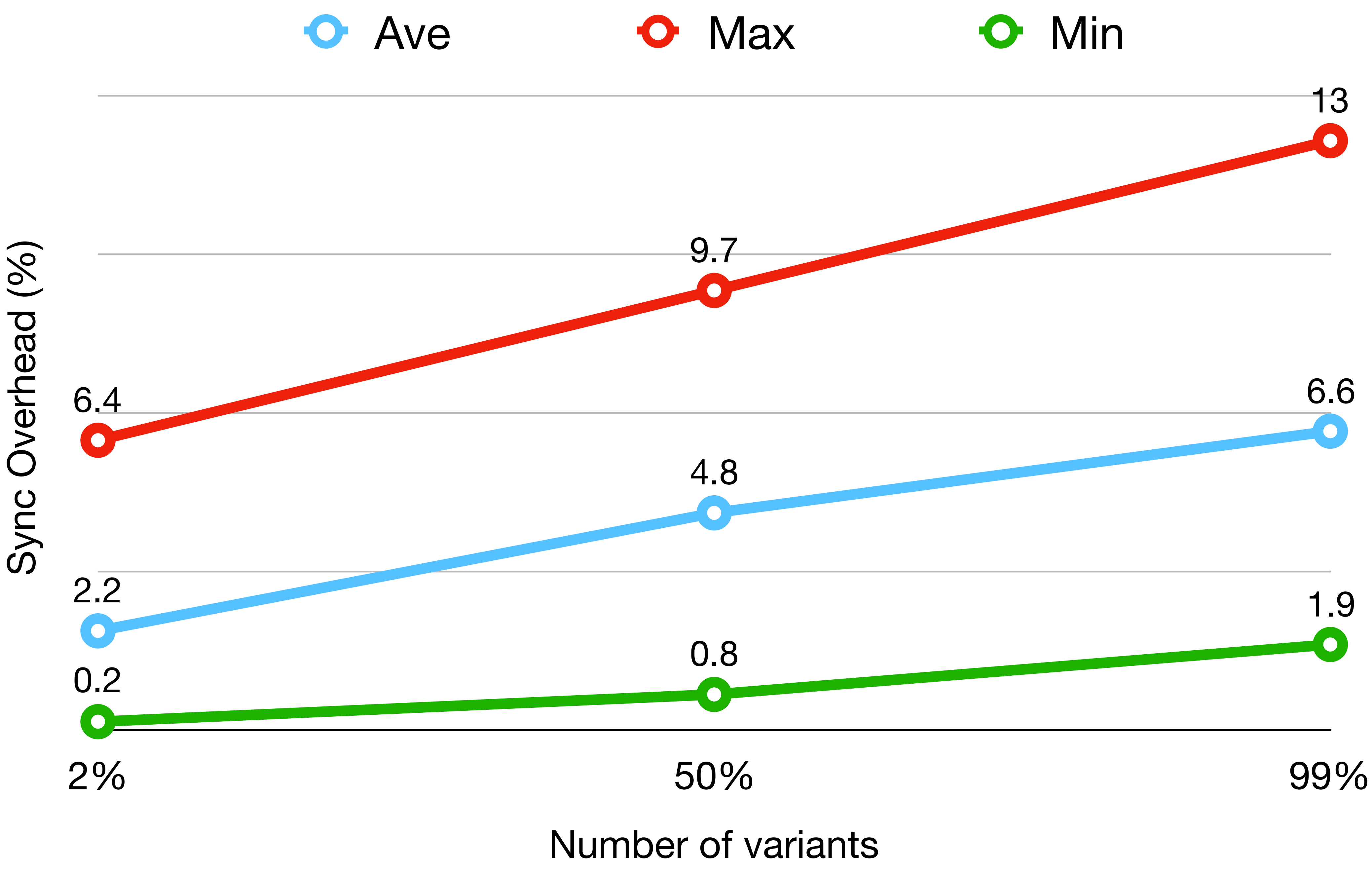


Scalability - Number of Variants

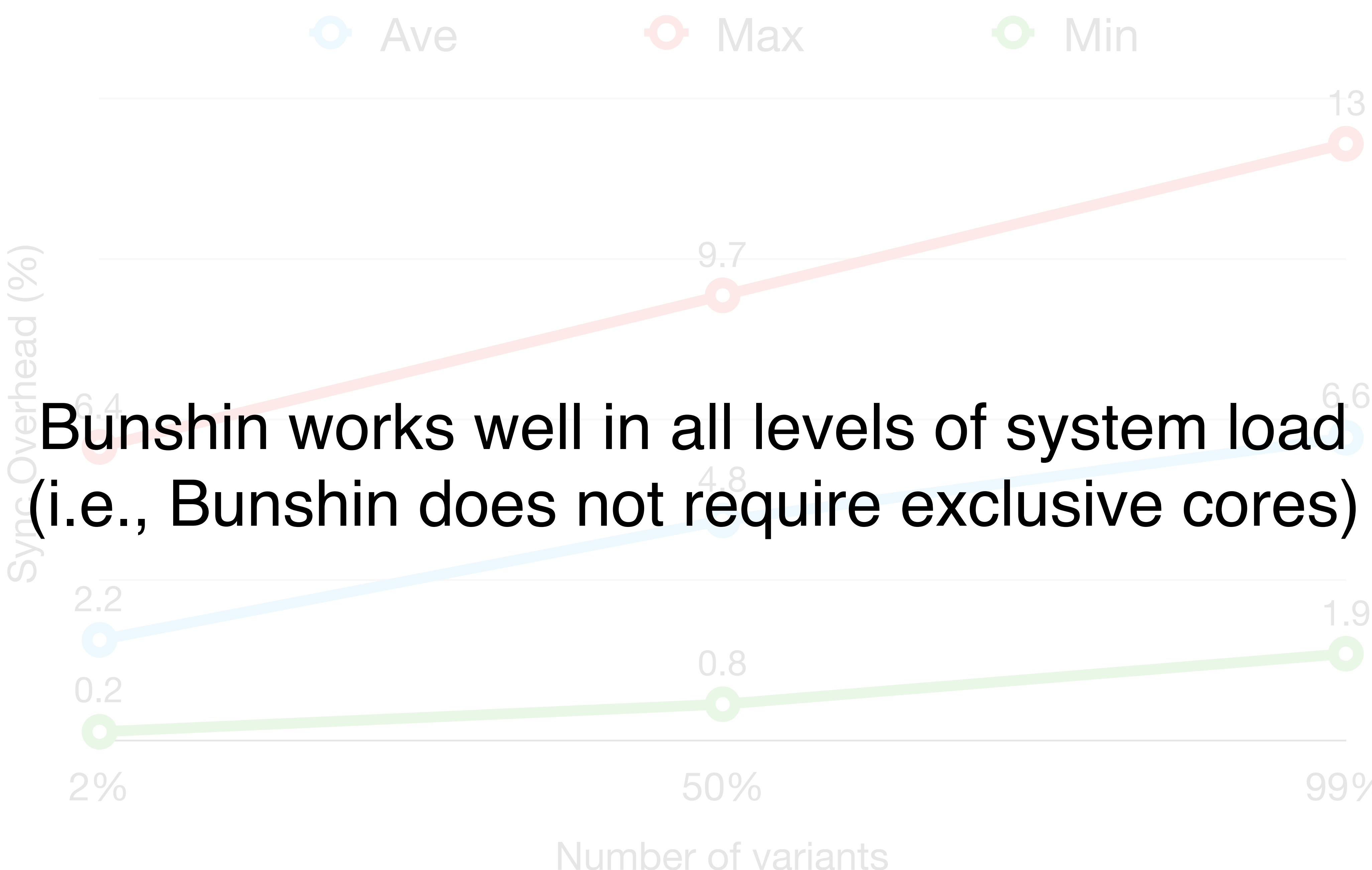


The number of variants Bunshin can support with a reasonable overhead depends on <u>machine configurations</u> and <u>program characteristics</u>.

Scalability - System Load

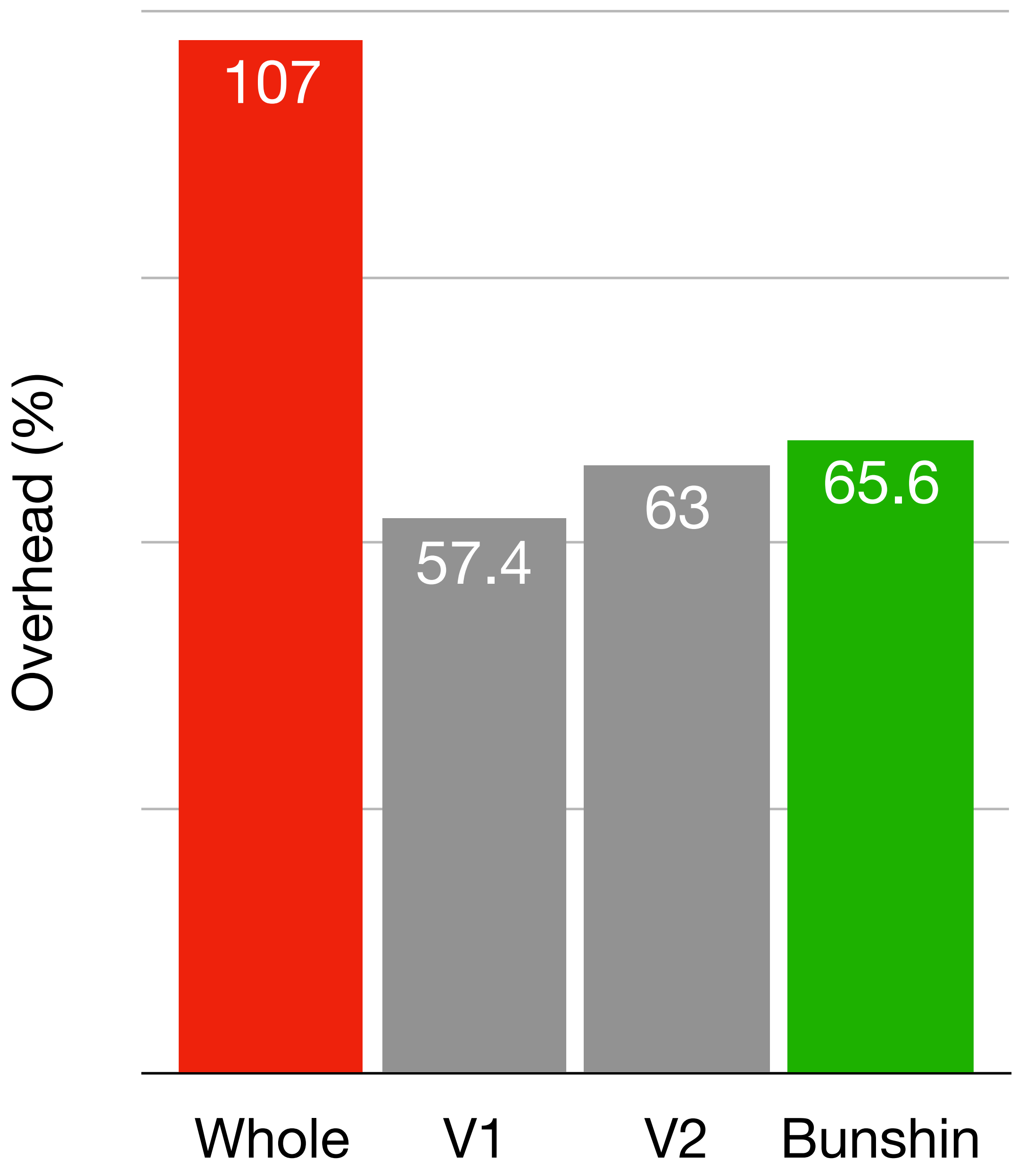


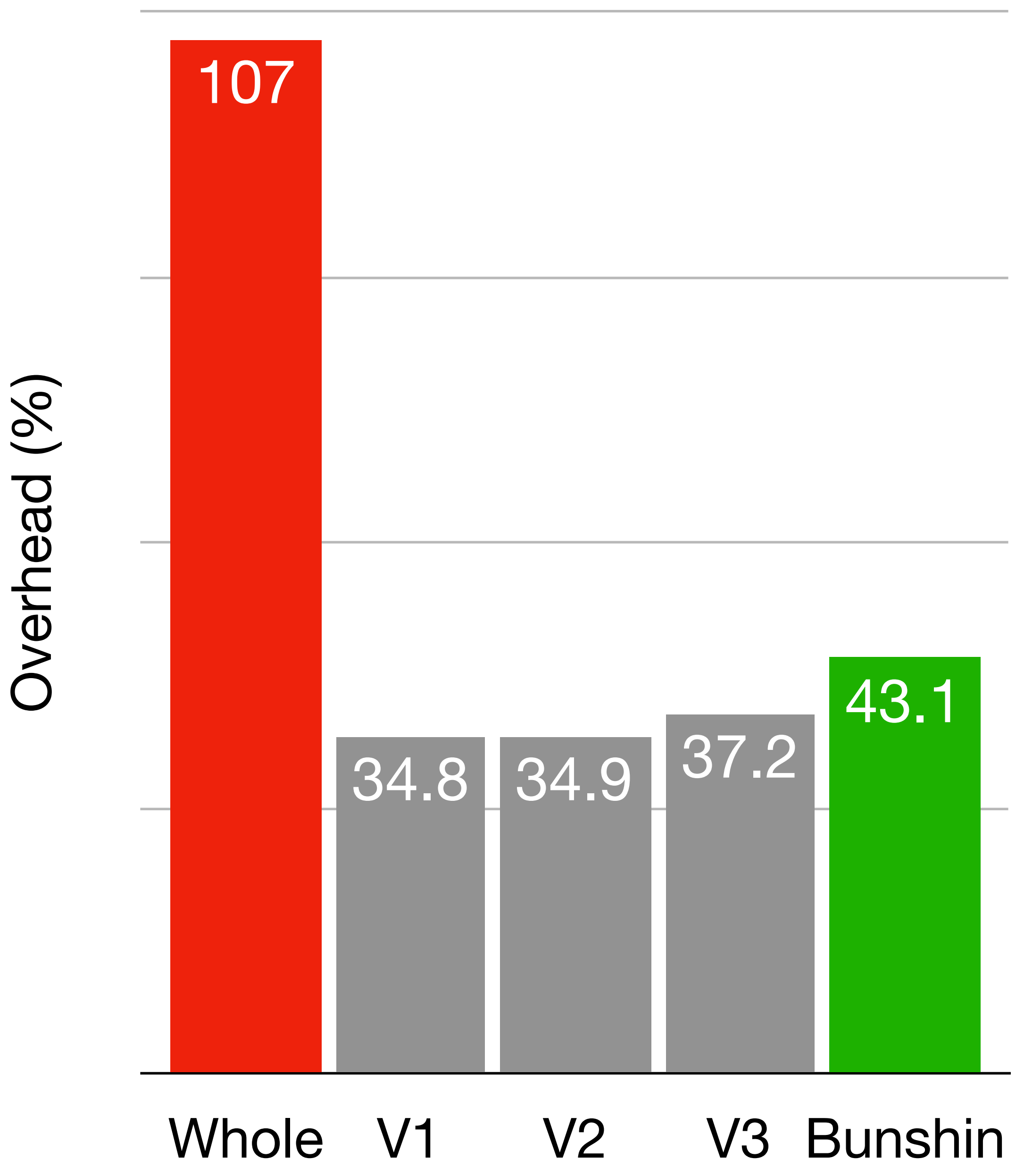
Scalability - System Load



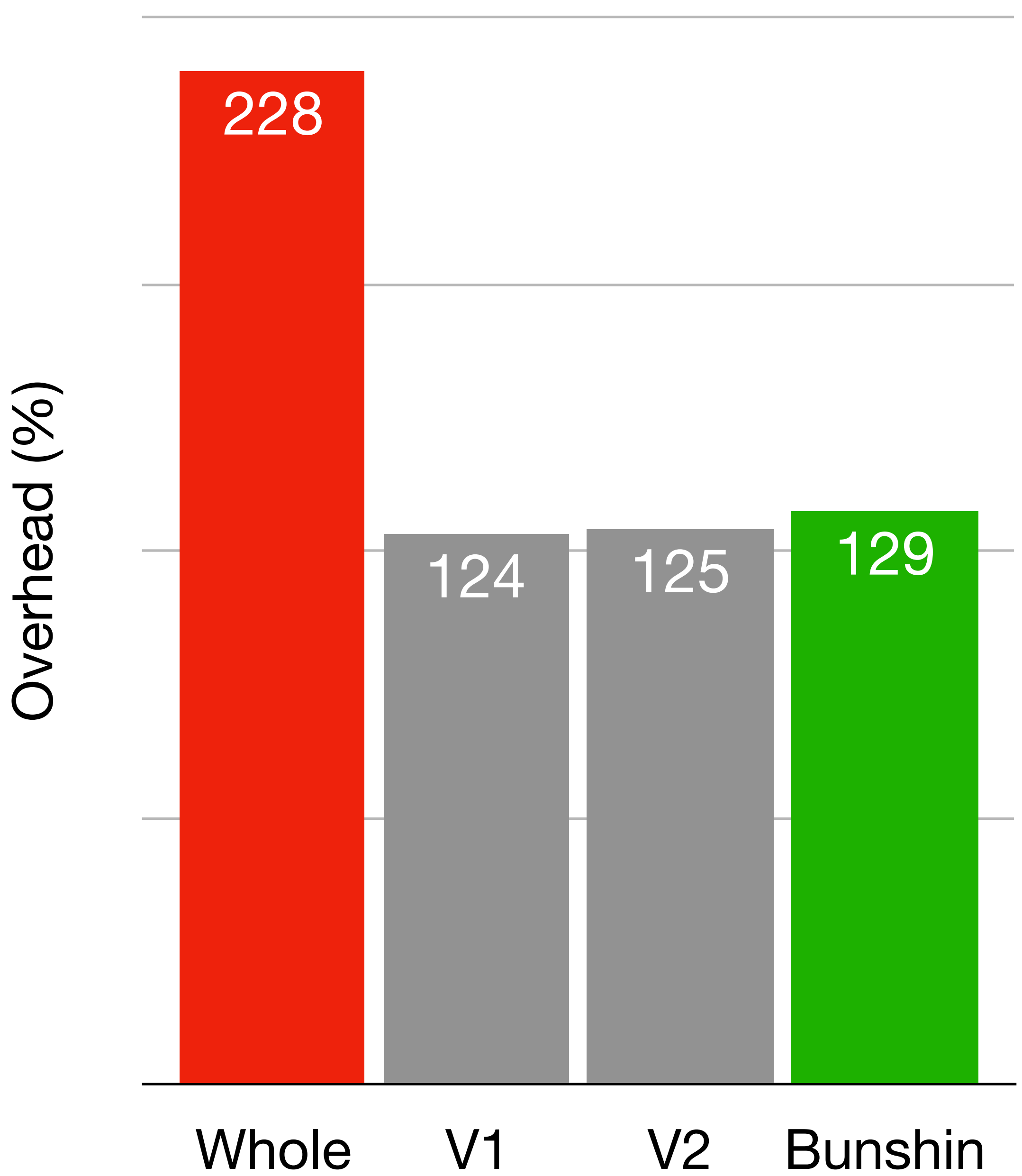
Bunshin works well in all levels of system load
(i.e., Bunshin does not require exclusive cores)

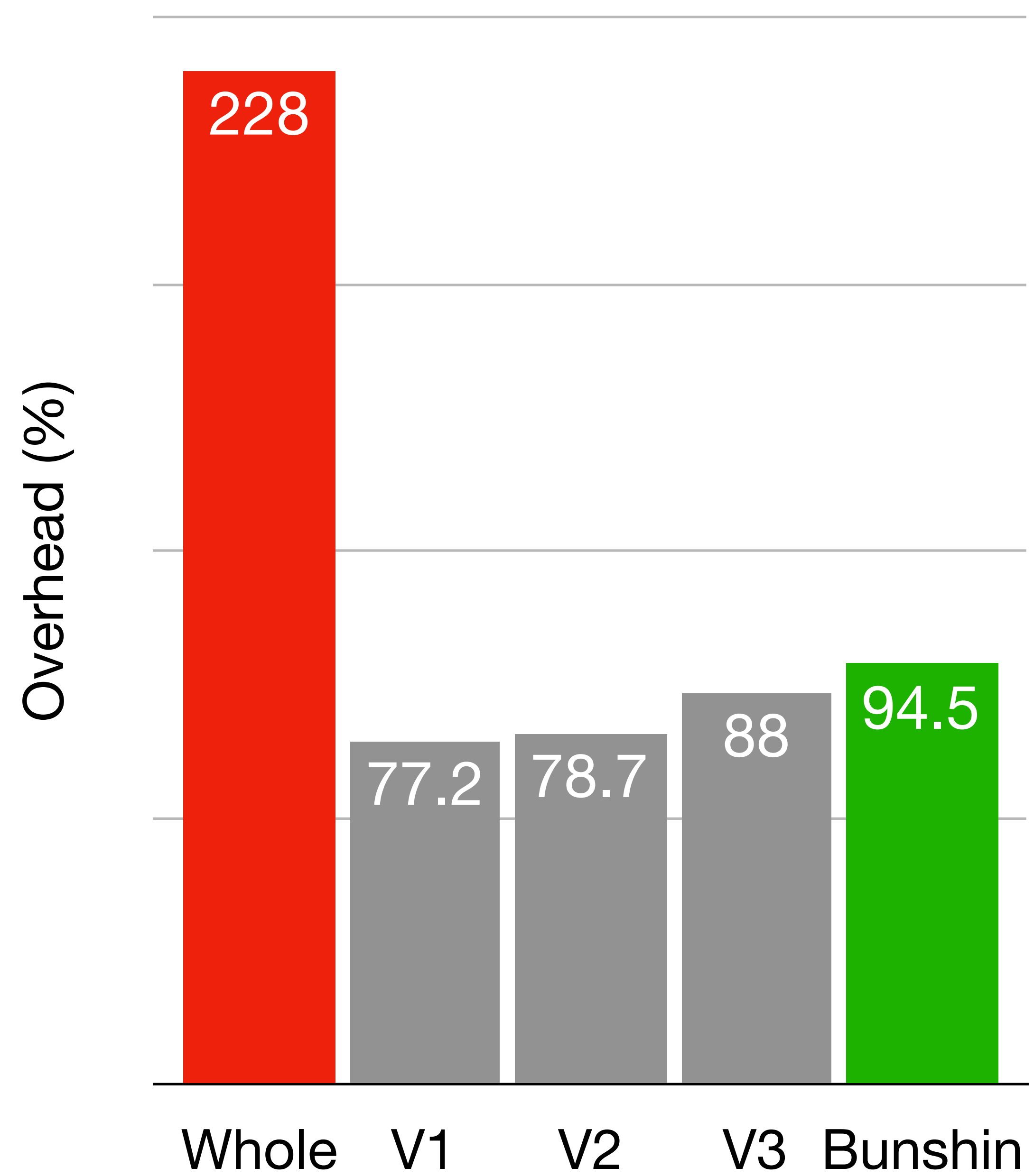
Check Distribution - ASan



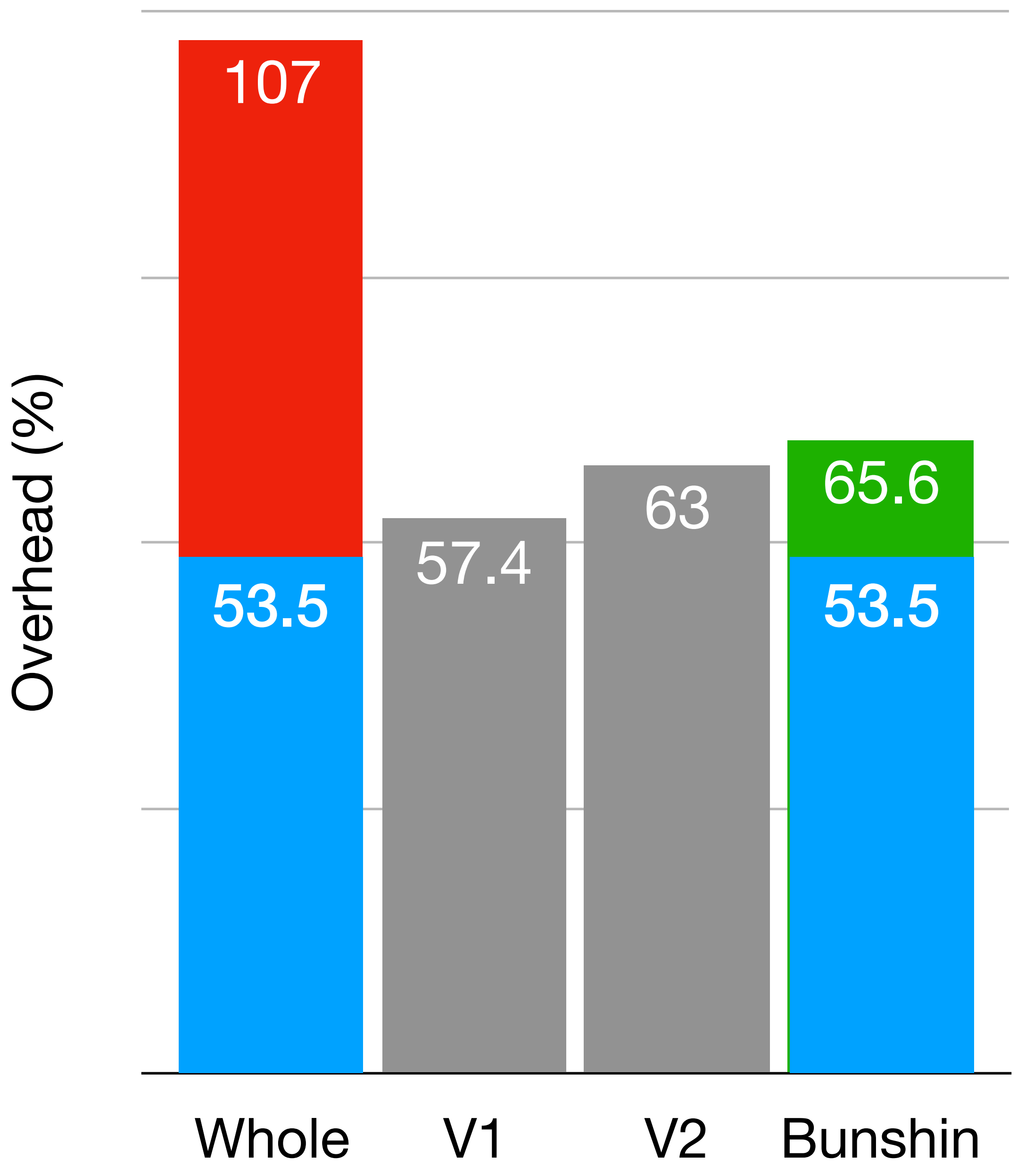


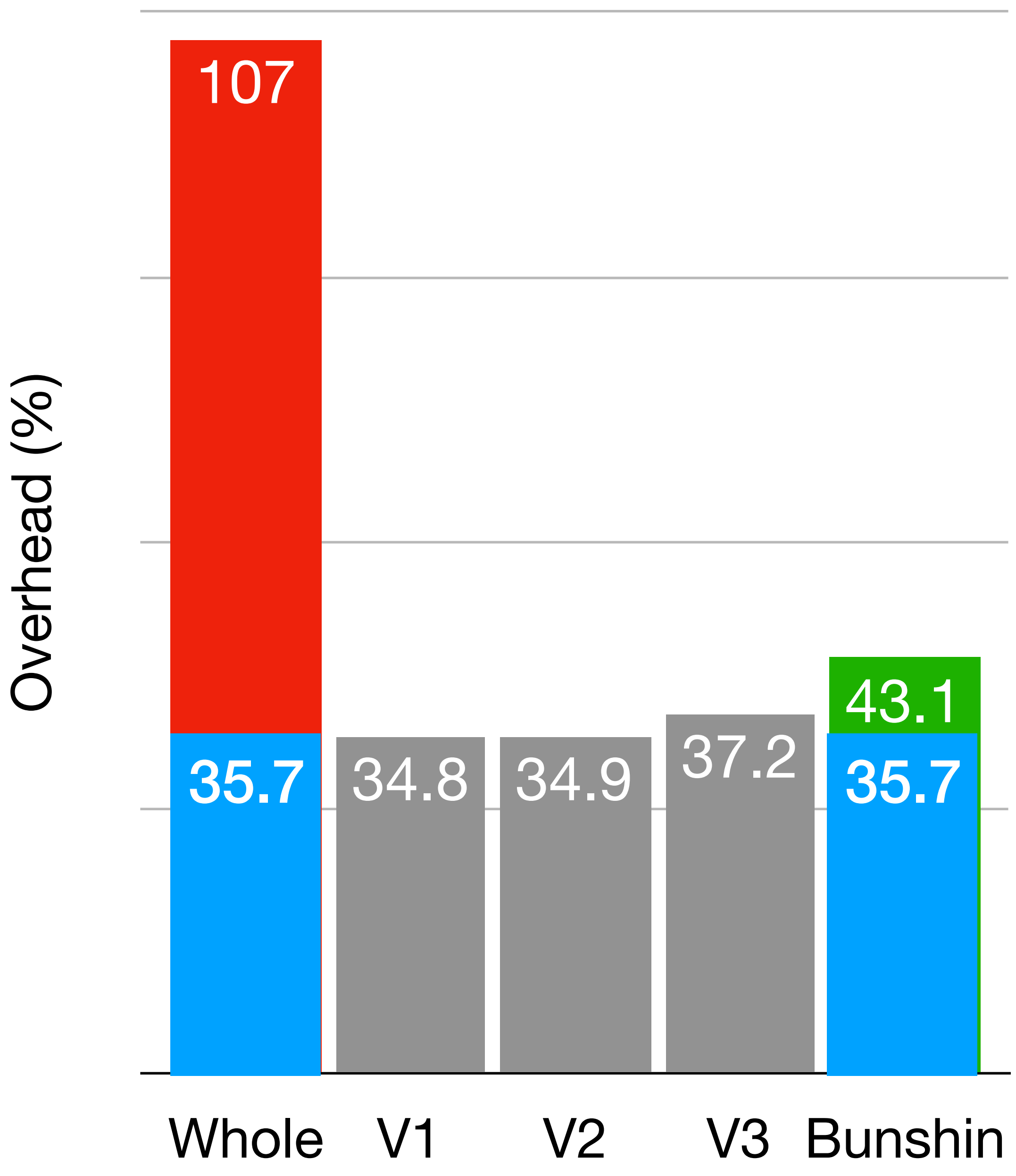
Sanitizer Distribution - UBSan



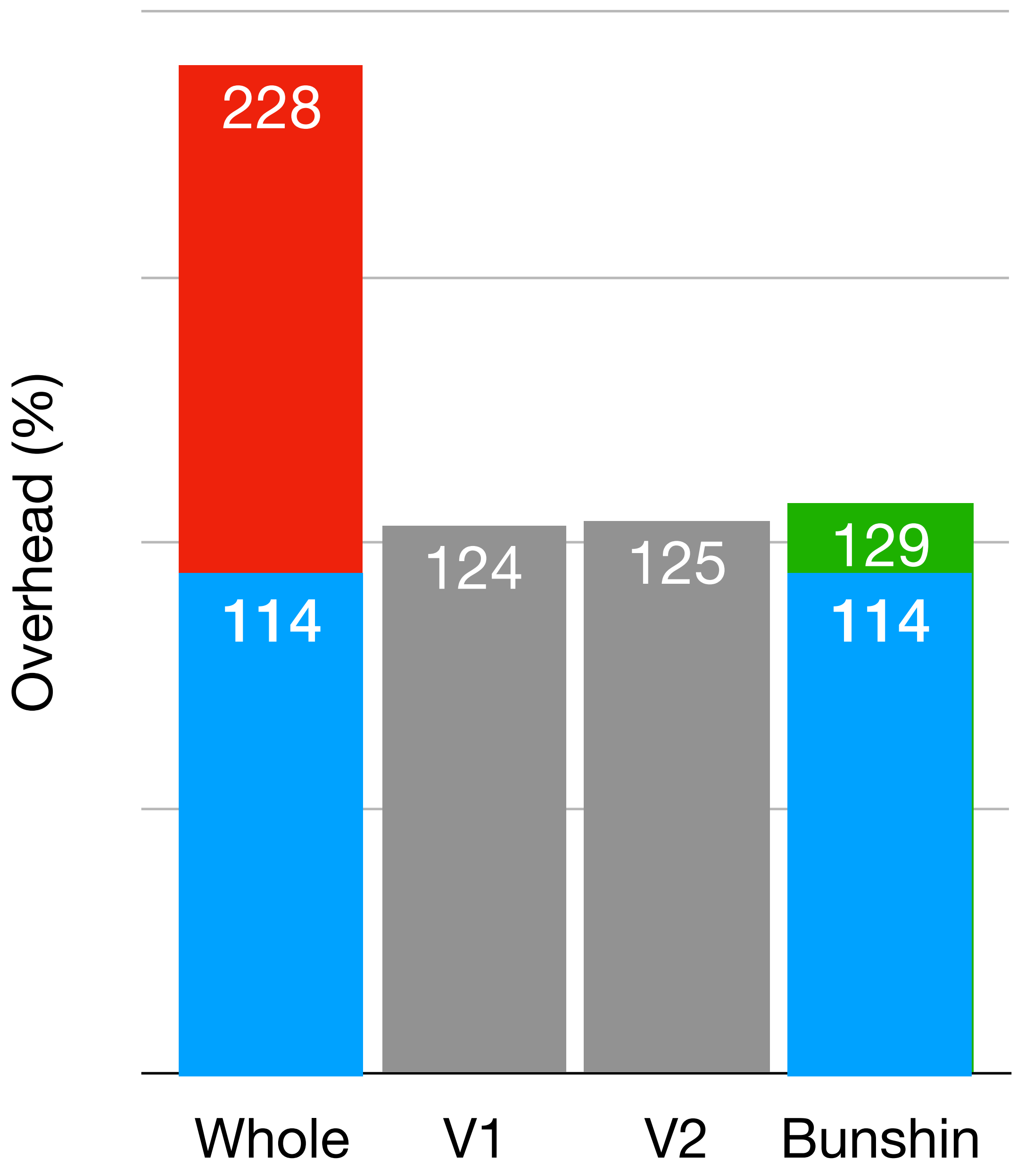


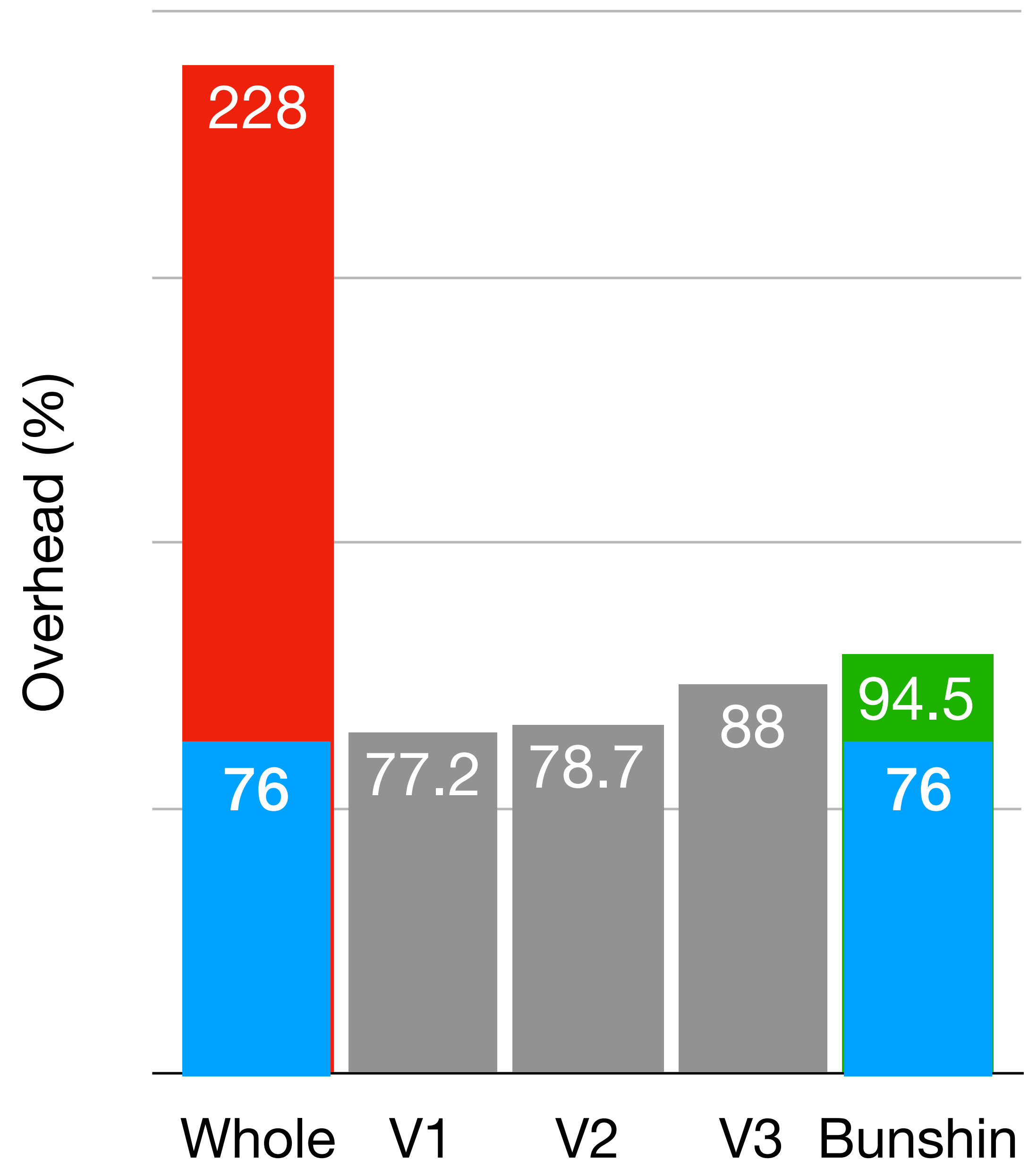
Deviation from Optimal - ASan





Deviation from Optimal - UBSan

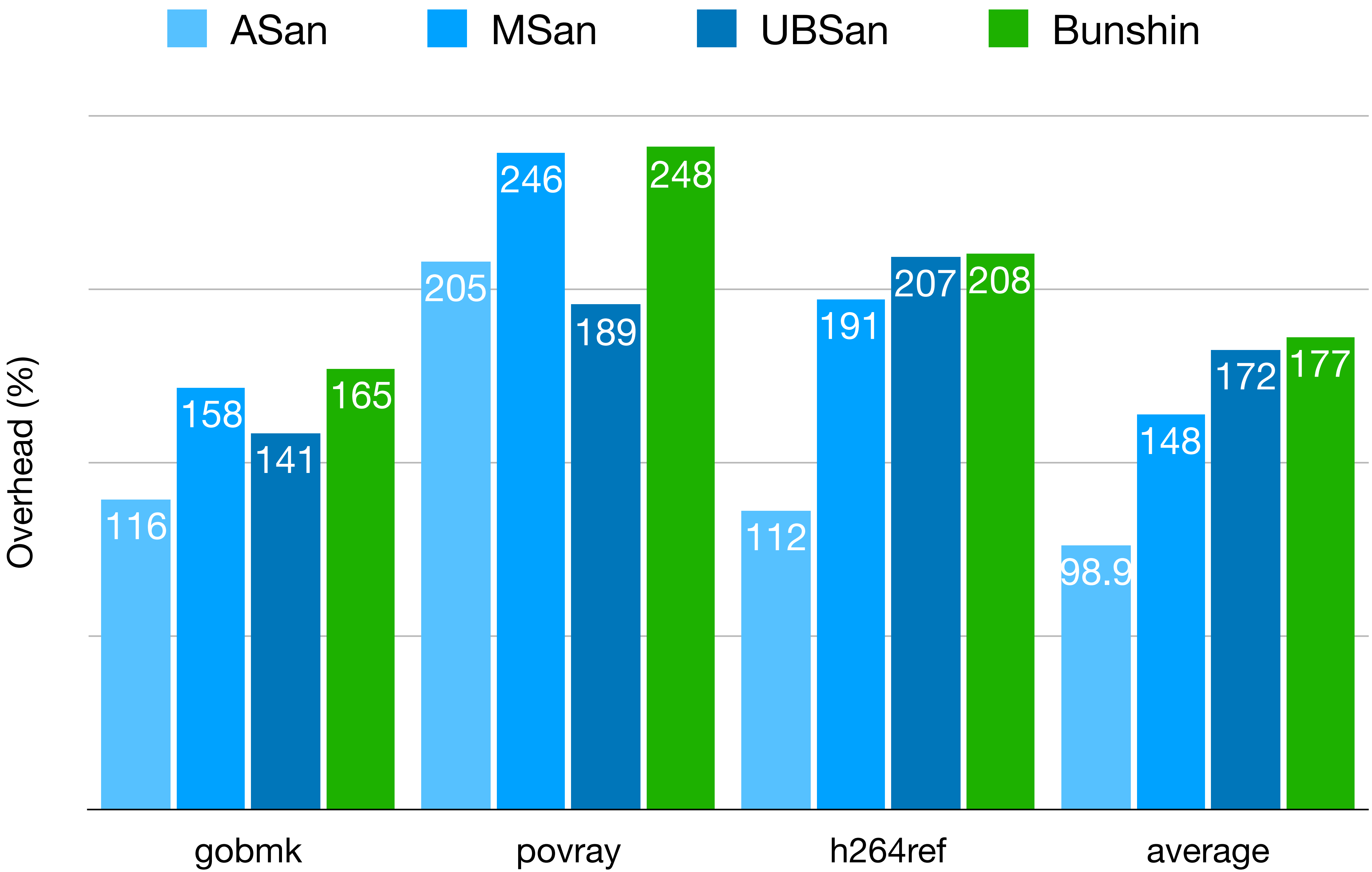




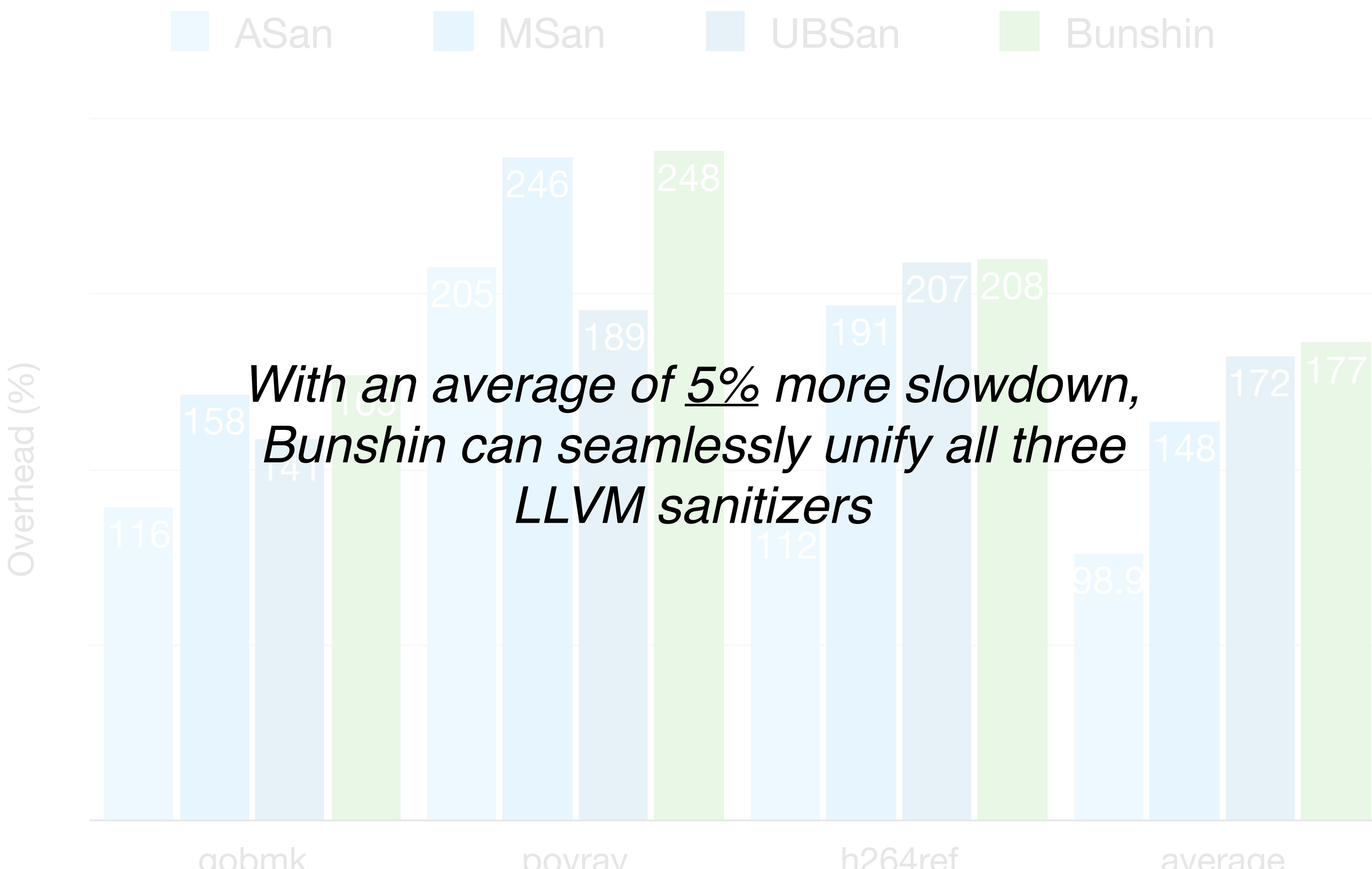
Reasons for Deviation from Optimal

- Synchronization overhead
- Inaccuracy in profiling
- Suboptimal distribution
- Non-distributable overhead

Unifying LLVM Sanitizers



Unifying LLVM Sanitizers



With an average of 5% more slowdown, Bunshin can seamlessly unify all three LLVM sanitizers

Limitations and Future Work

- Finer-grained check distribution
- Sanitizer integration
- Record-and-replay

Conclusion

- It is feasible to achieve both comprehensive protection and high throughput with an N-version system
- Bunshin is effective in reducing slowdown caused by sanitizers
  - 107% → 47.1% for ASan, 228% → 94.5% for UBSan
- Bunshin can seamlessly unify three LLVM sanitizers with 5% extra slowdown

https://github.com/sslab-gatech/bunshin

(Source code will be released soon)