U.S. NAVAL RESEARCH LABORATORY

USENIX ATC'22 Best Paper Award!

Co-opting Linux Processes for High-Performance Network Simulation

Rob Jansen, U.S. Naval Research Laboratory
Jim Newsome, Tor Project
Ryan Wails, Georgetown University & U.S. Naval Research Laboratory

Rob Jansen, Ph.D.
Computer Security Research Scientist
Center for High Assurance Computer Systems
U.S. Naval Research Laboratory

USENIX Annual Technical Conference
Carlsbad, CA, USA
July 11th, 2022

U.S. NAVAL RESEARCH LABORATORY

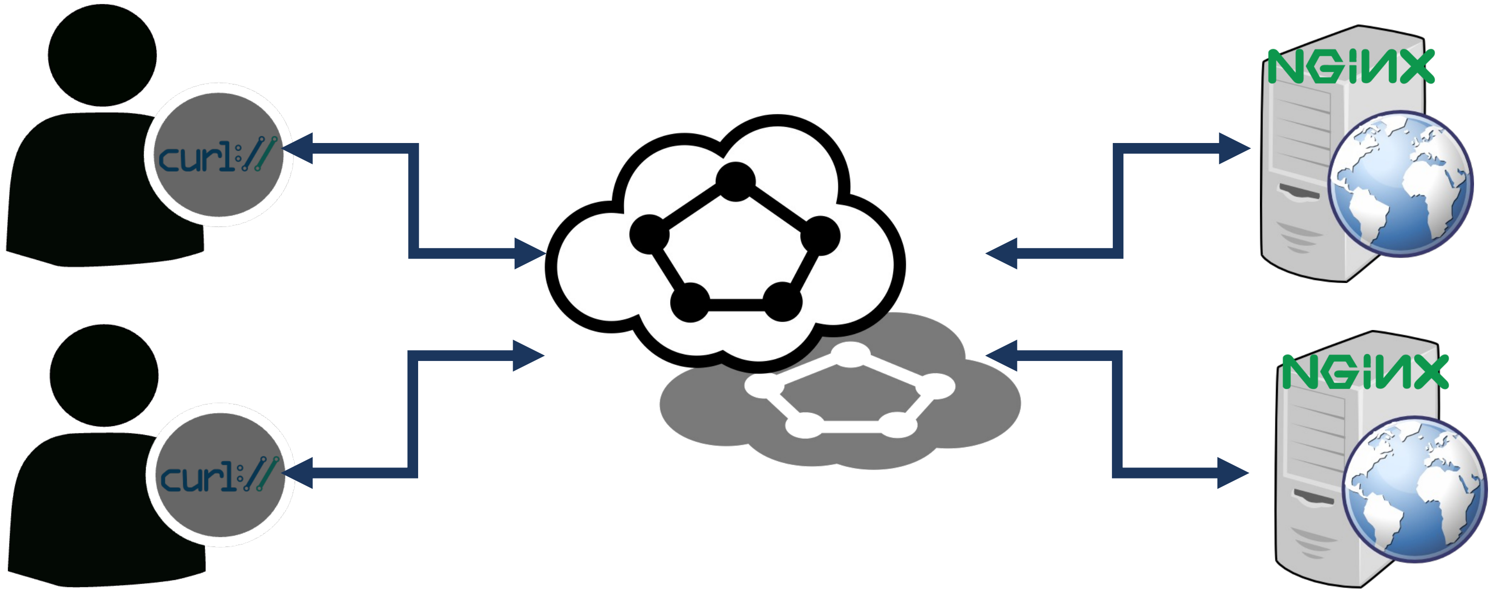
Main Takeaways

Read the paper!

shadow on github
shadow.github.io

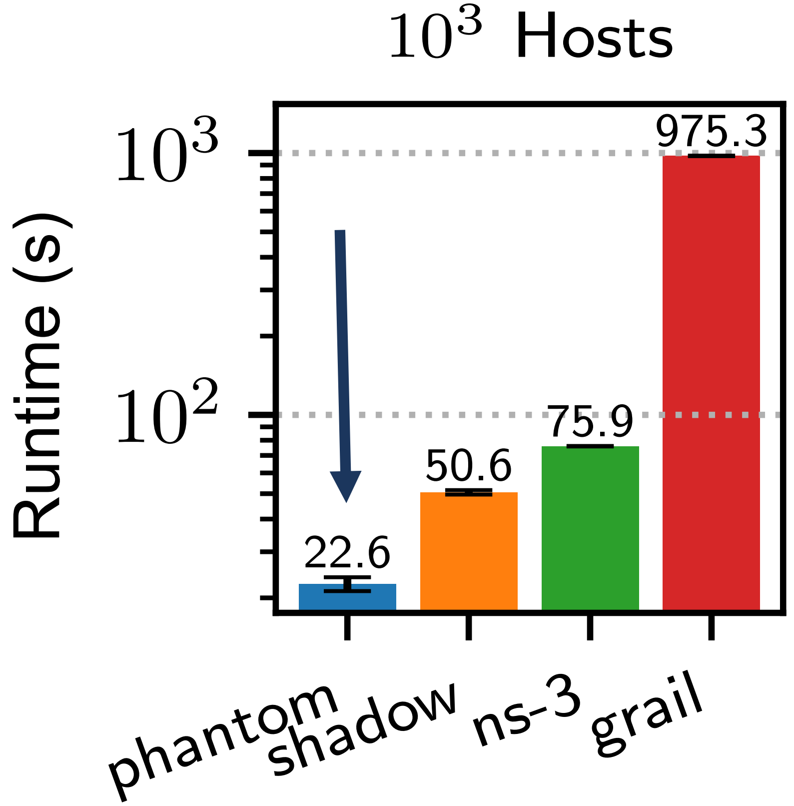
Designed a new, hybrid network simulator/emulator

- co-opts Linux processes into a discrete-event network simulation that emulates kernel functionality
- enables large-scale, distributed system experiments



- Merged into the open-source Shadow project and synonymous with **Shadow v2**
- Artifacts: https://netsim-atc2022.github.io

- **2.3x** faster than Shadow v1
- **3.4x** faster than NS-3
- **43x** faster than gRalL [ToN'19]



Outline

motivation

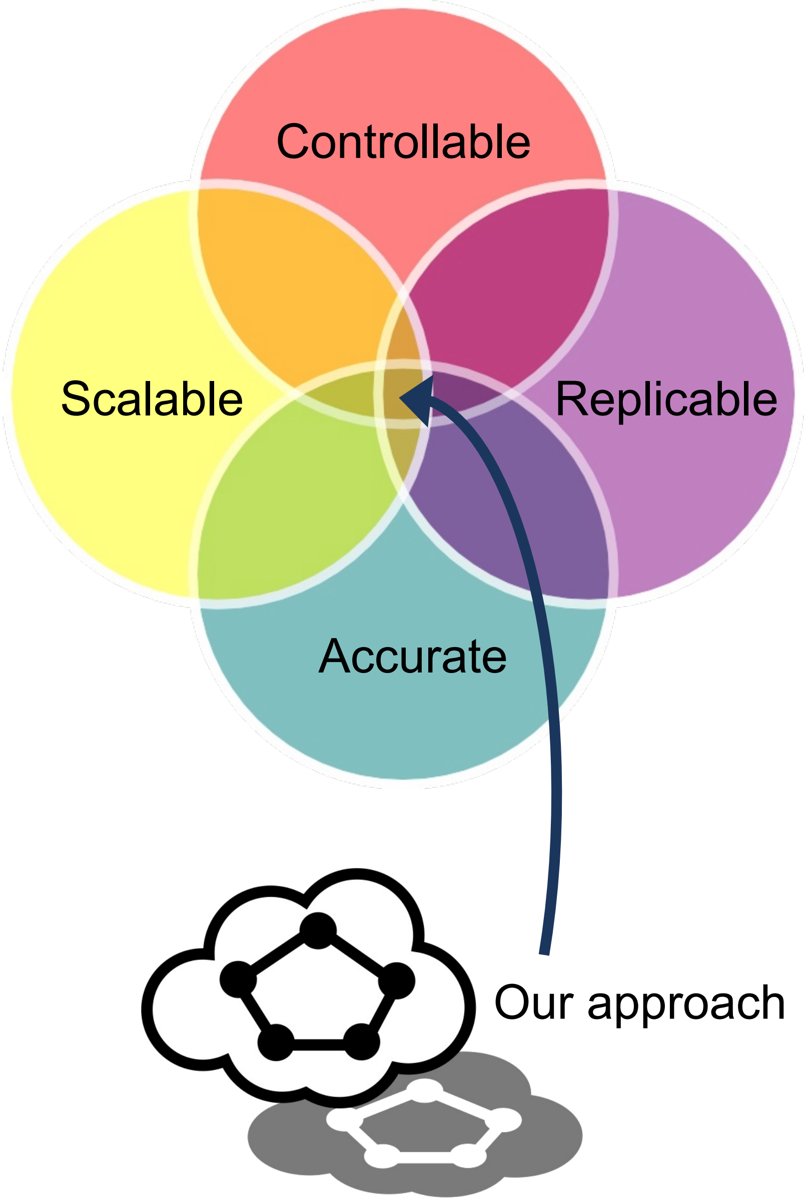
design

evaluation

Requirements for Large Distributed System Experimentation

- Important properties of test networks:
  - **Controllable**: isolate important factors
  - **Replicable**: identically replicate experiments (determinism)

- Requirements for large distributed systems:
  - **Accurate**: directly execute system software (not an abstraction)
  - **Scalable**: decouple from time, computational constraints of host



Problems with Traditional Approaches

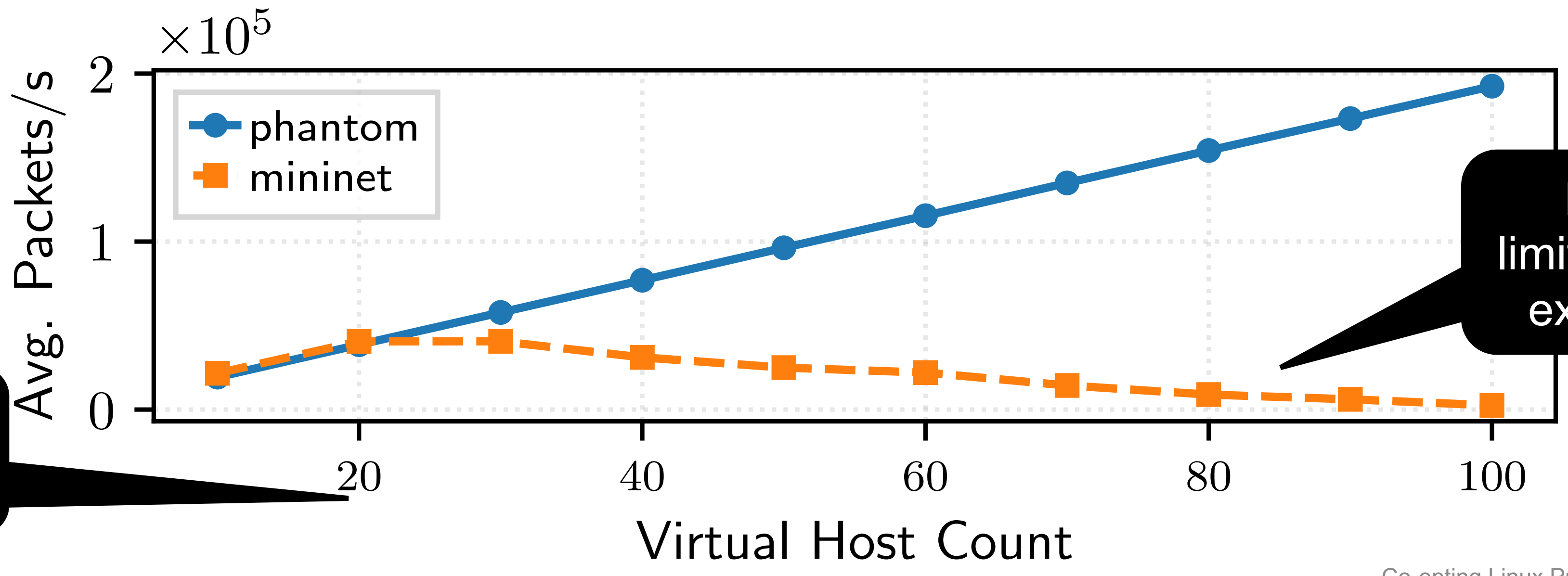
- Simulation (e.g., ns-3)
  - Not realistic: runs abstractions instead of real applications
  - Hard to maintain and can lead to invalid results
- Emulation (e.g., mininet)
  - Not controllable: results will not be identical
  - Not scalable: CPU overload → time distortion

Mininet > sudo mn

Problems with Traditional Approaches

- Simulation (e.g., ns-3)
  - Not realistic: runs abstractions instead of real applications
  - Hard to maintain and can lead to invalid results
- Emulation (e.g., mininet)
  - Not controllable: results will not be identical
  - Not scalable: CPU overload → time distortion

Mininet > sudo mn



Forwarding capacity limited: fewer packets than expected are forwarded

CPU overload with >20 hosts

U.S. NAVAL RESEARCH LABORATORY

Hybrid Architectures and Challenges

- Hybrid architecture
  - Network simulation, but directly execute application code
  - Enjoys advantages of both simulation and emulation
  - Best opportunity to scale to large-scale distributed systems

Architecture	Example Tool	Scalability	Realism	Control
Emulation	Mininet	✗	✓	✗
Simulation	NS-3	✓	✗	✓
Hybrid	This work	✓	✓	✓

Limitations of Hybrid Architectures (1)

Executing application code via plugin (link-map) namespaces

- appid = dlmopen(app.so)
- func = dlsym(appid, “main”)
- func()

NS-3-DCE, Shadow

Limitations

- Compatibility (must build PIC/PIE)
- Correctness (intercept libcalls only)
- Maintainability (custom ld, threading)

Limitations of Hybrid Architectures (1)

Executing application code via plugin (link-map) namespaces

- appid = dlmopen(app.so)
- func = dlsym(appid, “main”)
- func()

NS-3-DCE, Shadow

Limitations

- Compatibility (must build PIC/PIE)
- Correctness (intercept libcalls only)
- Maintainability (custom ld, threading)

Table 2: Application Properties Supported in Hybrid Simulators

Application Property	Shadow	Phantom
Multiple threads (e.g., support for `pthreads`)	●	●
Multiple processes (e.g., support for `fork`)	◐	◐
Not position-independent (i.e., `PIC` or `PIE`)	○	●
Not dynamically linked to `libc`	○	●
Symbols not exported to dynamic symbol table	○	●
System calls made in statically linked code	○	●
System calls made in assembly (i.e., avoiding `libc`)	○	●
100% statically linked (e.g., some go programs)	○	◐

○ Does not work in tool or architecture ● Works in tool & architecture
◐ Not implemented in tool (as of writing) but supported by architecture

Limitations of Hybrid Architectures (2)

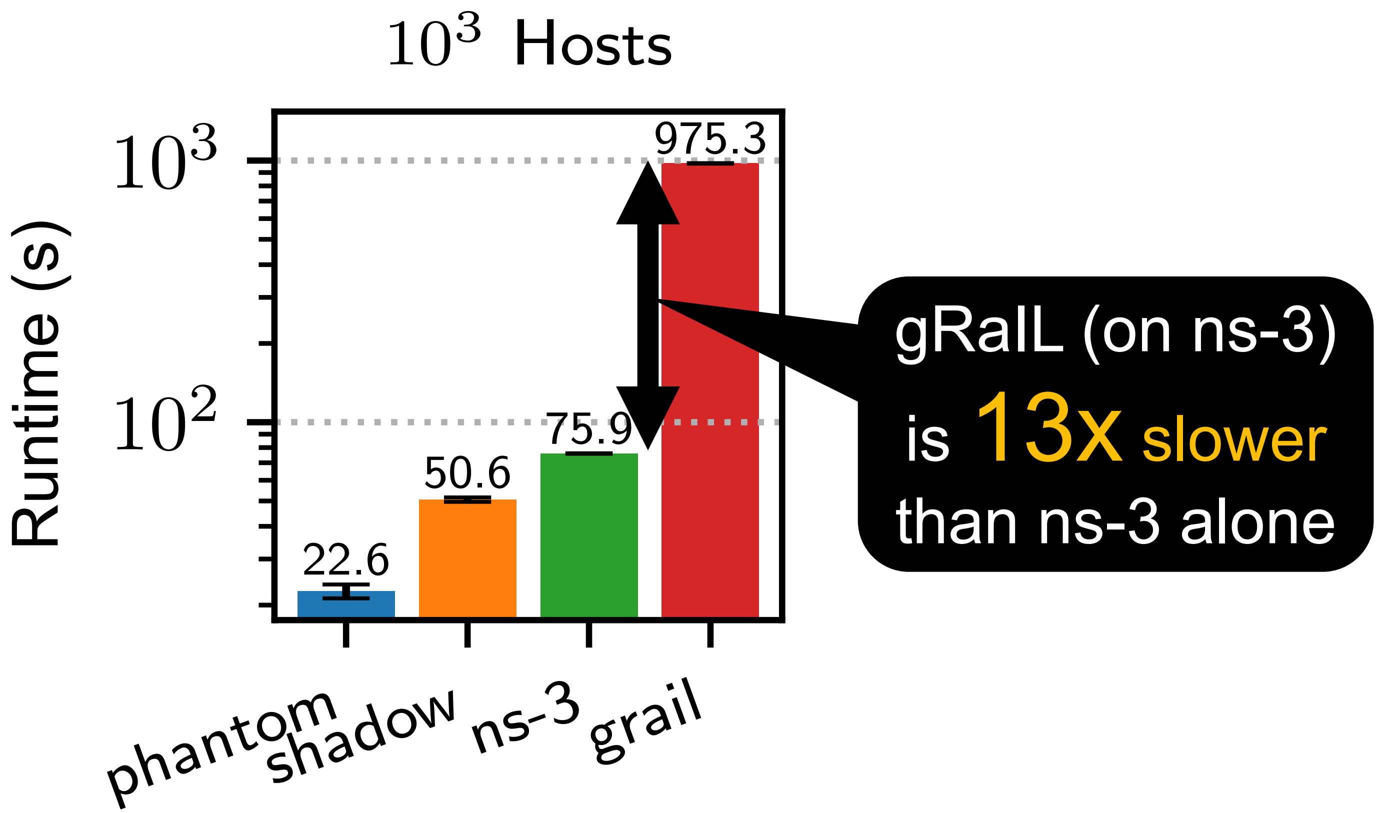
Executing code via Linux processes

- fork() + exec() → ptrace()

gRaIL [ToN'19]

Limitations: ptrace is slow!

- Process control: overhead **quadratic** in total number of processes
- Syscall interception: 4 context switches **for every syscall**
- Data transfer: extra syscall + mode change **for every word of memory**



Our Research Challenge

Can we design a tool with the **performance benefits** of a uni-process plugin-based architecture

AND

the improved **modularity** and **isolation** of a mutli-process architecture?

U.S. NAVAL RESEARCH LABORATORY

Our Research Challenge

Outline

motivation

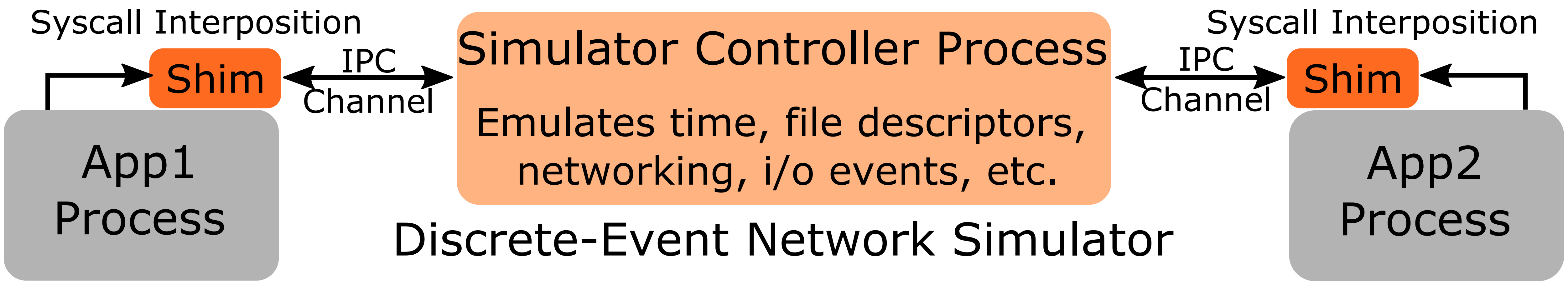
design

evaluation

Design Overview

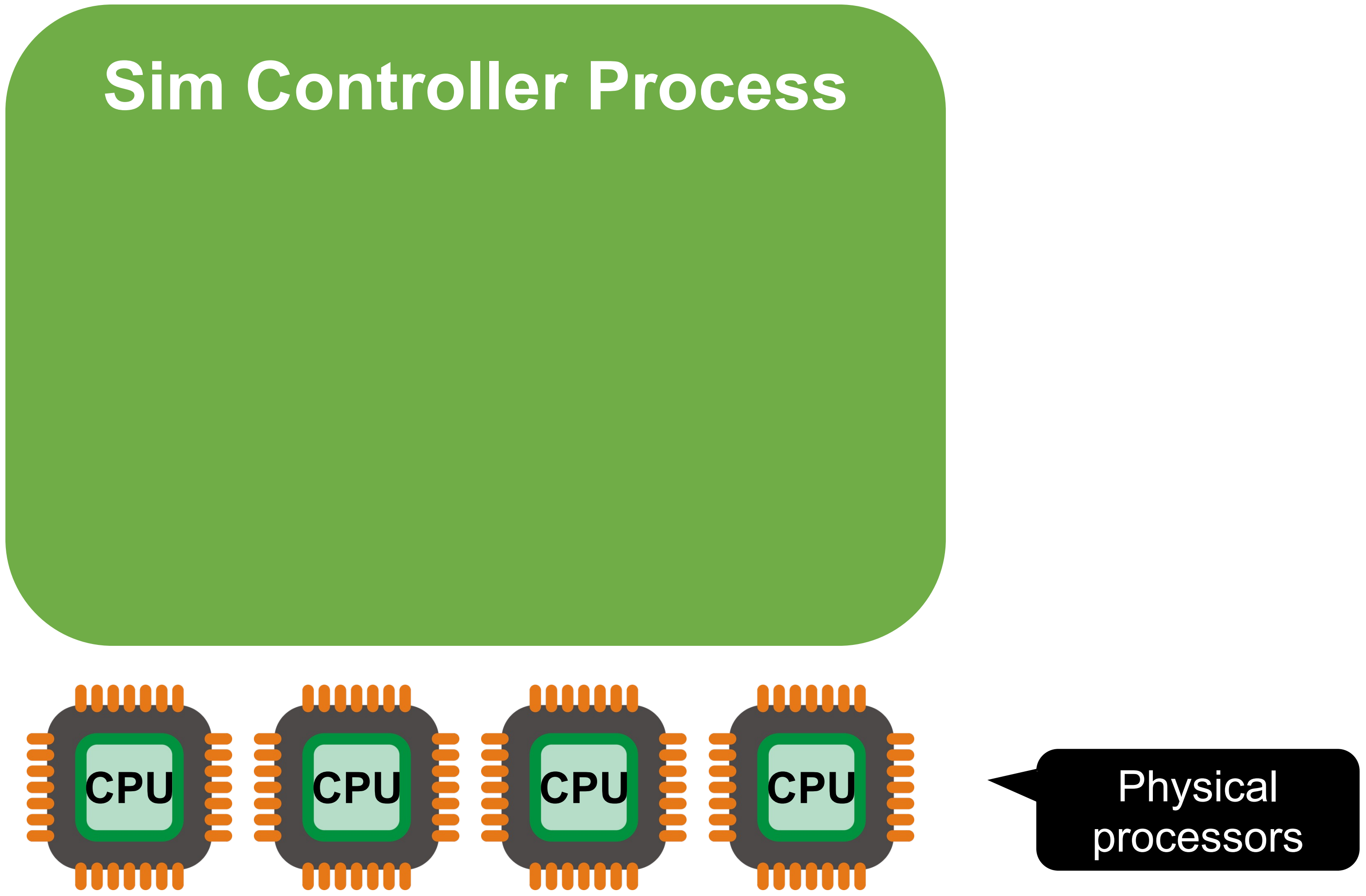
- Discrete-event packet-level network simulator
- Directly executes apps as standard Linux processes
- Intercepts all system calls made by apps and emulates them

- Simulates system call behavior and networking
    - File descriptors (files, sockets, pipes)
    - Event notification (poll, epoll, select)
    - Networking (buffers, protocols, ifaces)
    - DNS and routing (latency, bandwidth)



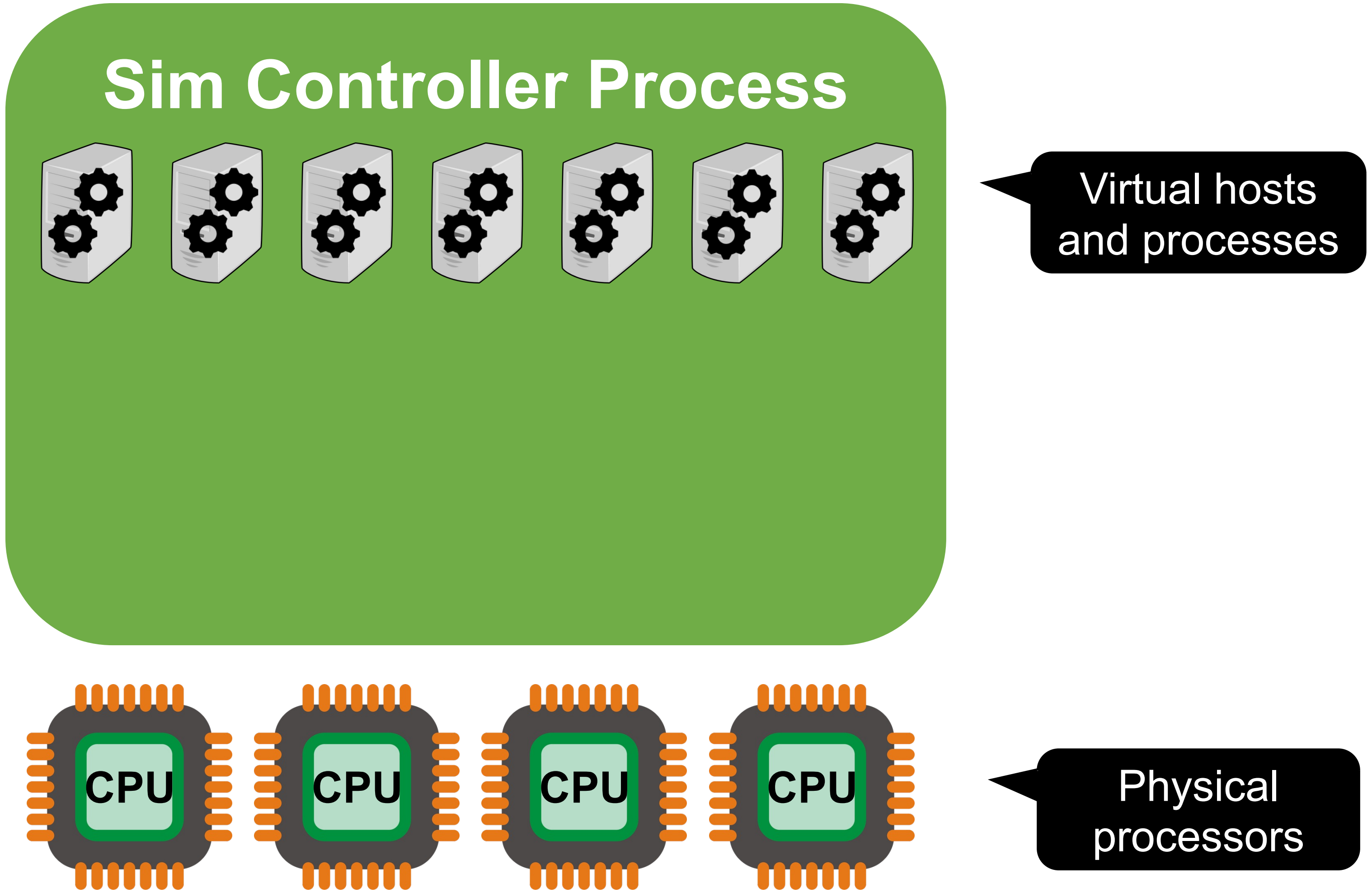
Parallel Worker Threads

Goal: efficiently parallelize simulation workload



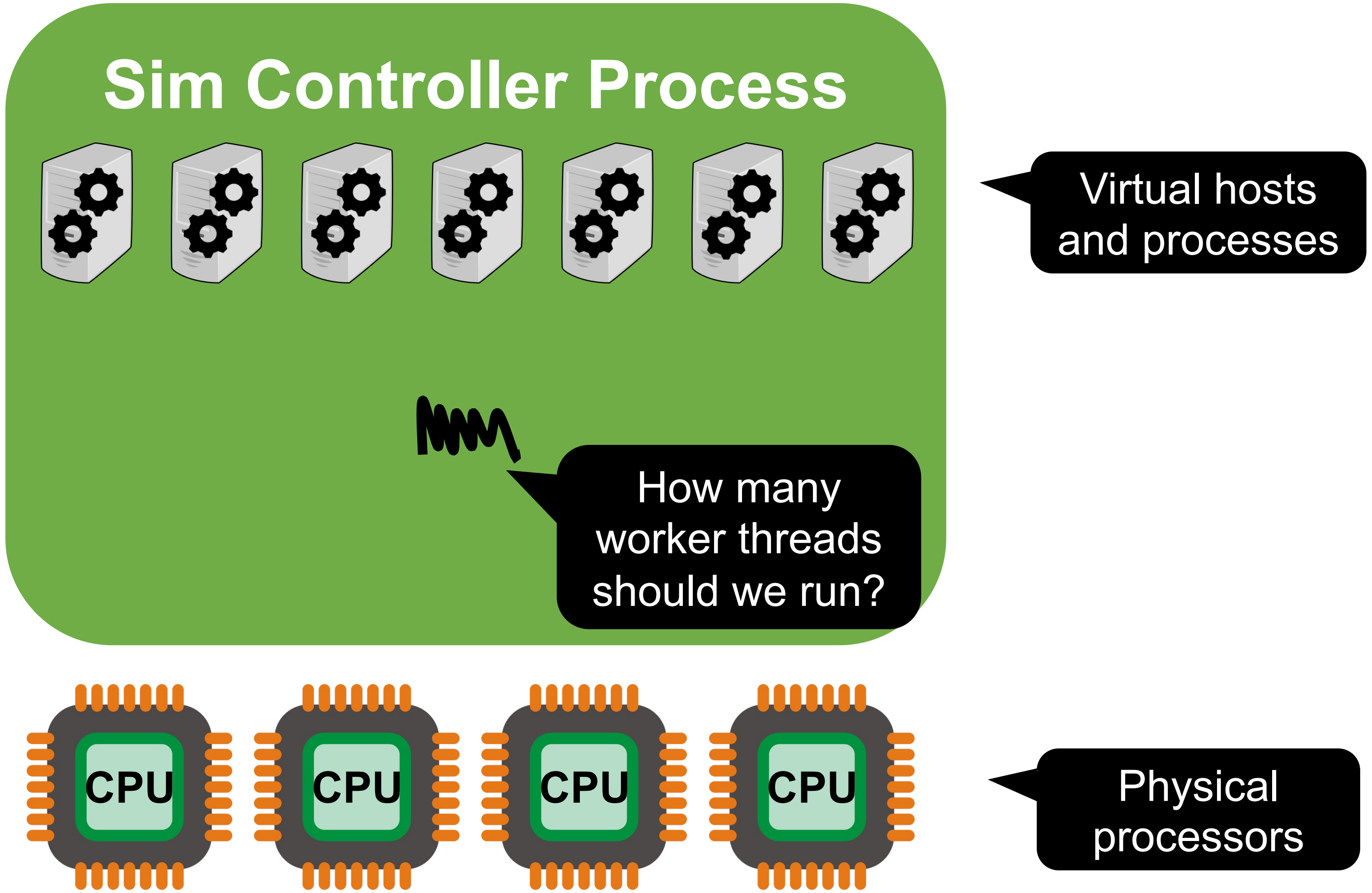
Parallel Worker Threads

Goal: efficiently parallelize simulation workload



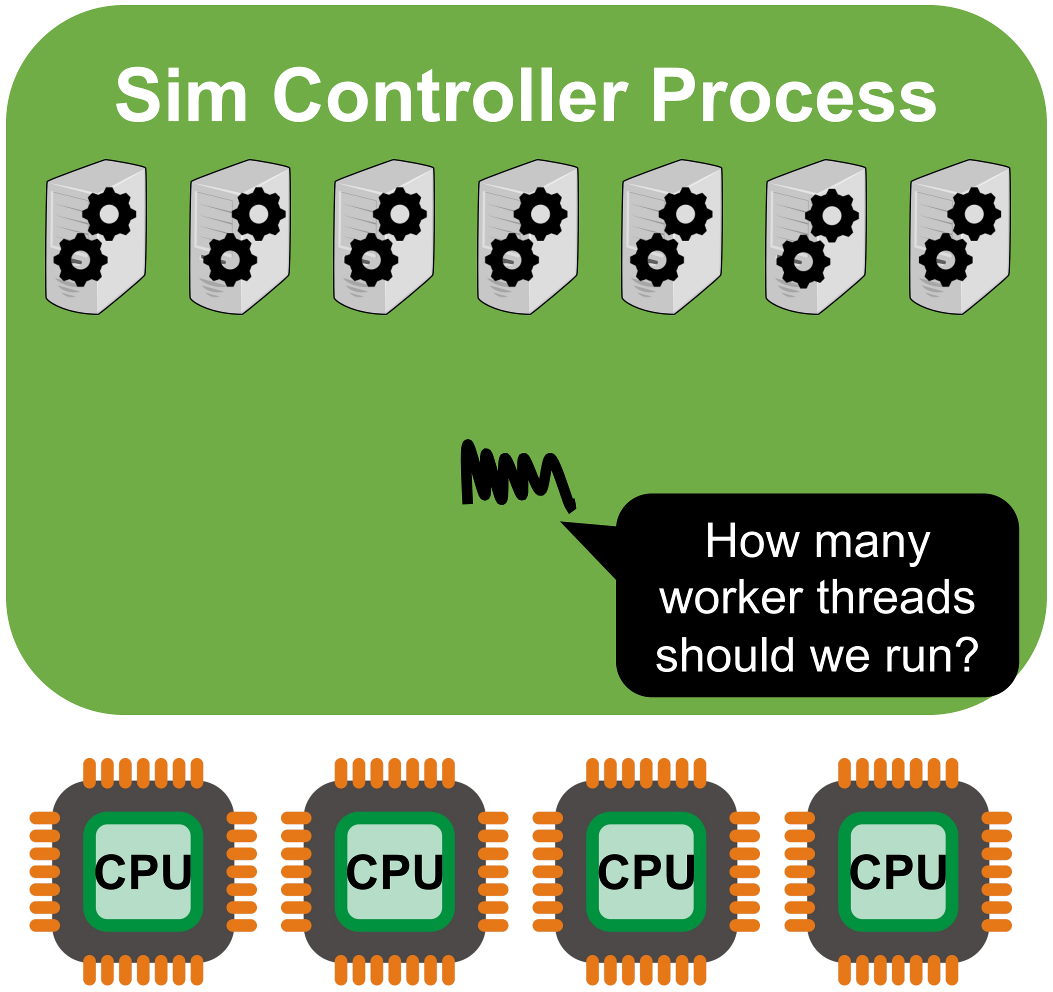
Parallel Worker Threads

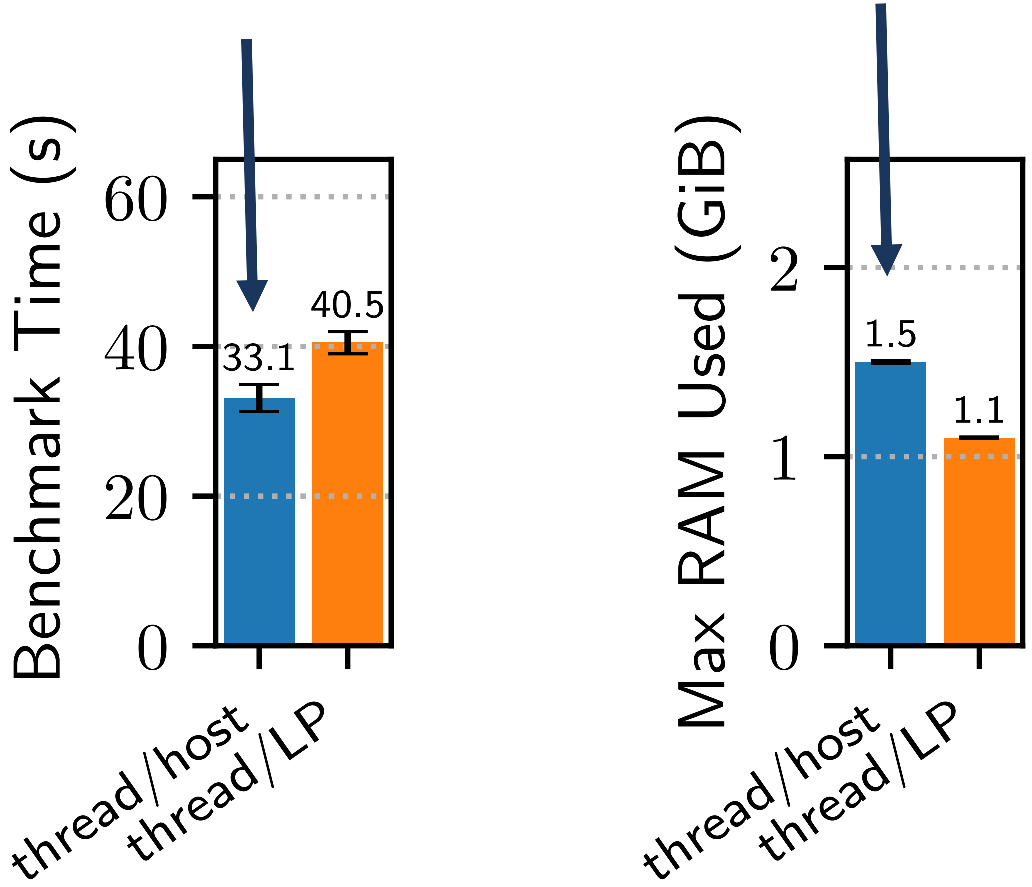
Goal: efficiently parallelize simulation workload



Parallel Worker Threads

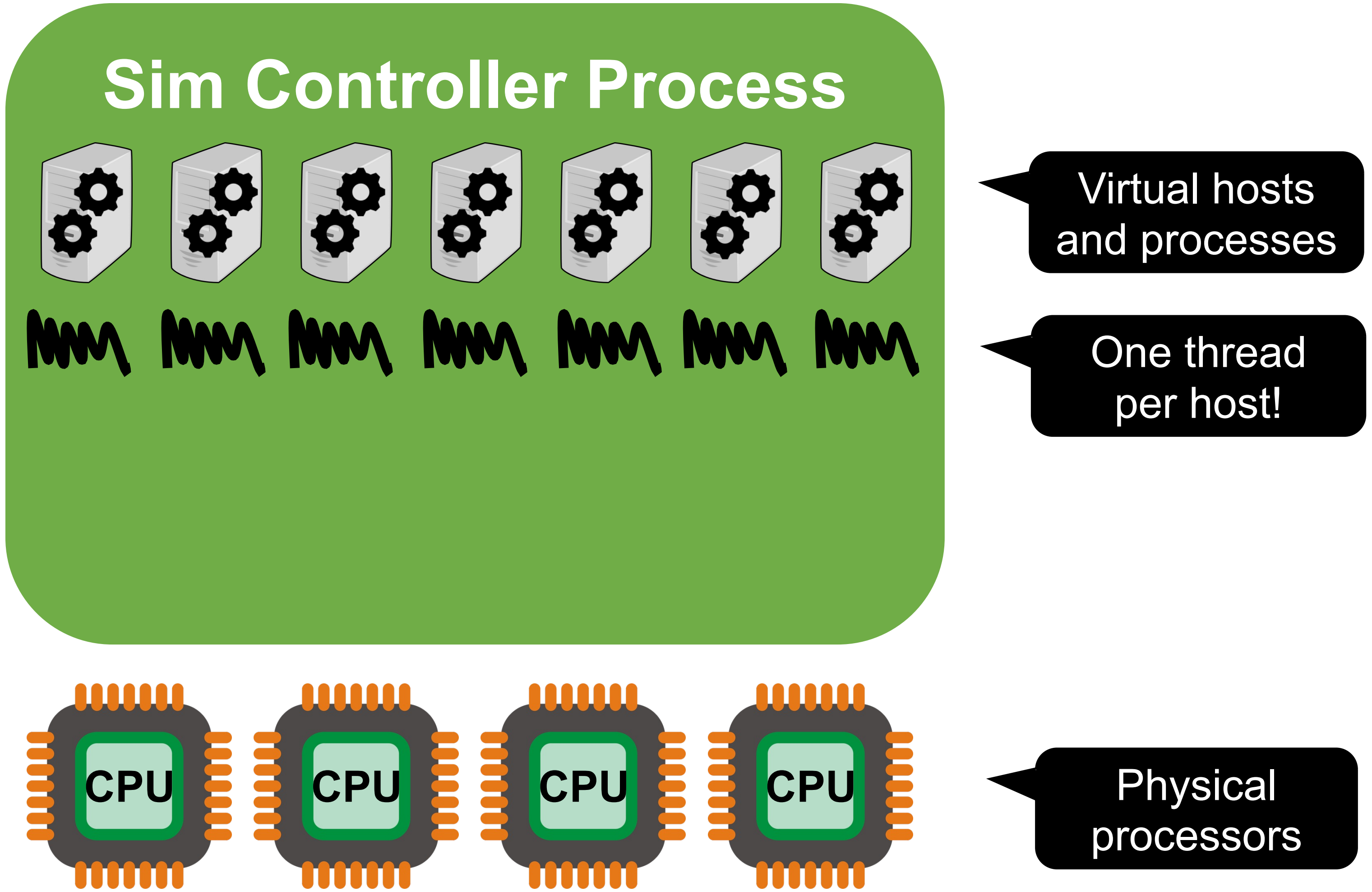
Goal: efficiently parallelize simulation workload





Parallel Worker Threads

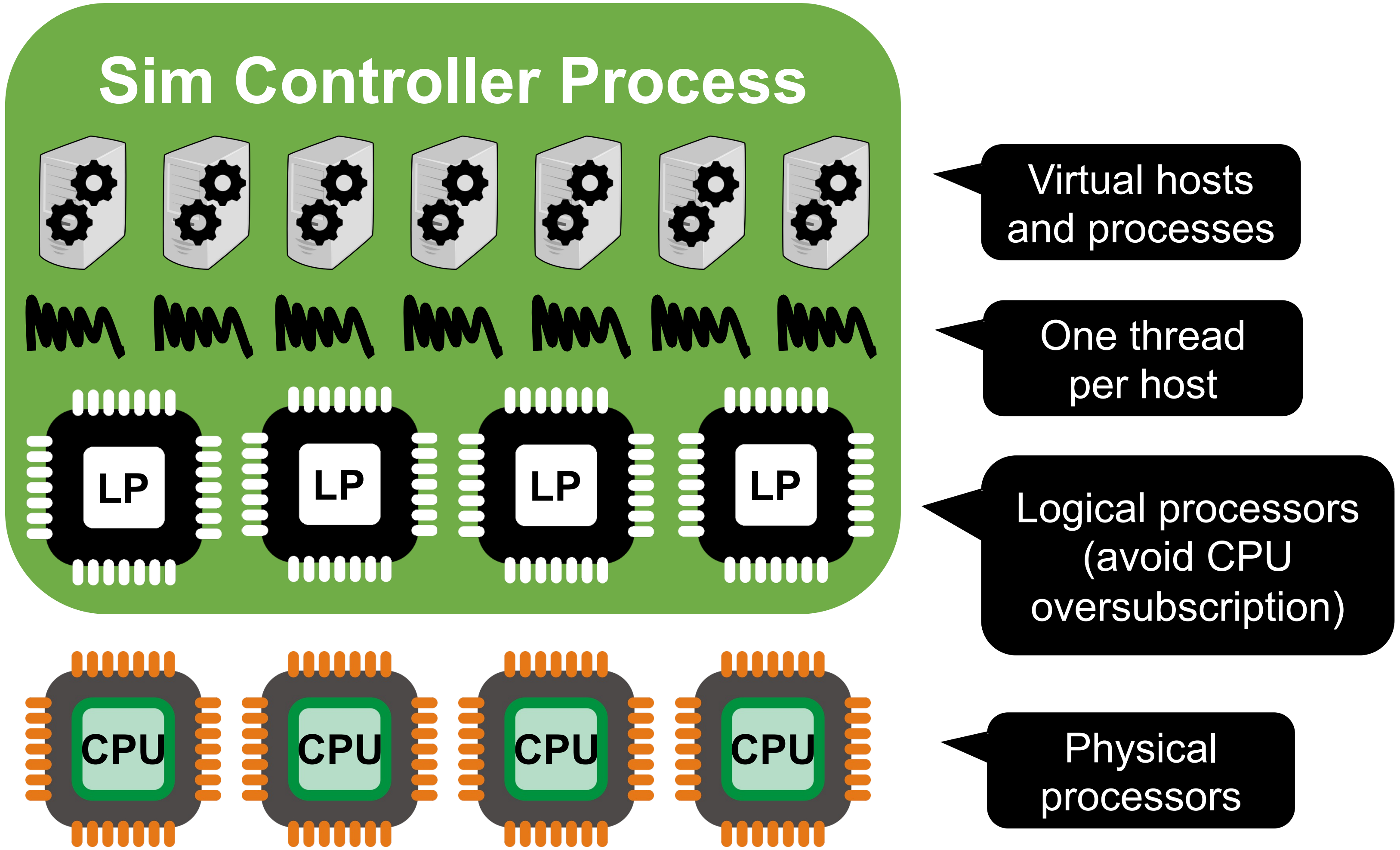
Goal: efficiently parallelize simulation workload



U.S. NAVAL RESEARCH LABORATORY

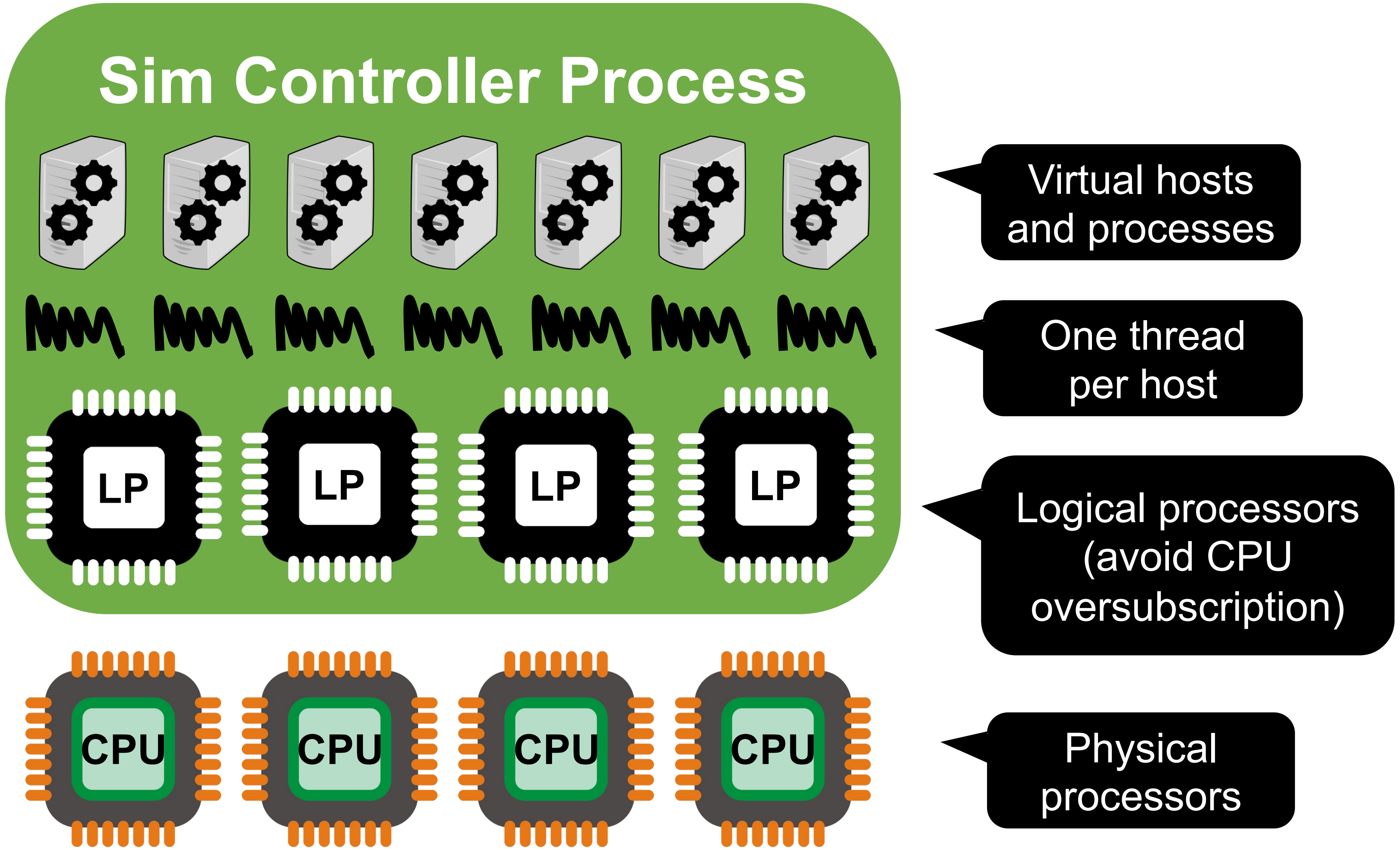
Parallel Worker Threads

Goal: efficiently parallelize simulation workload



Parallel Worker Threads

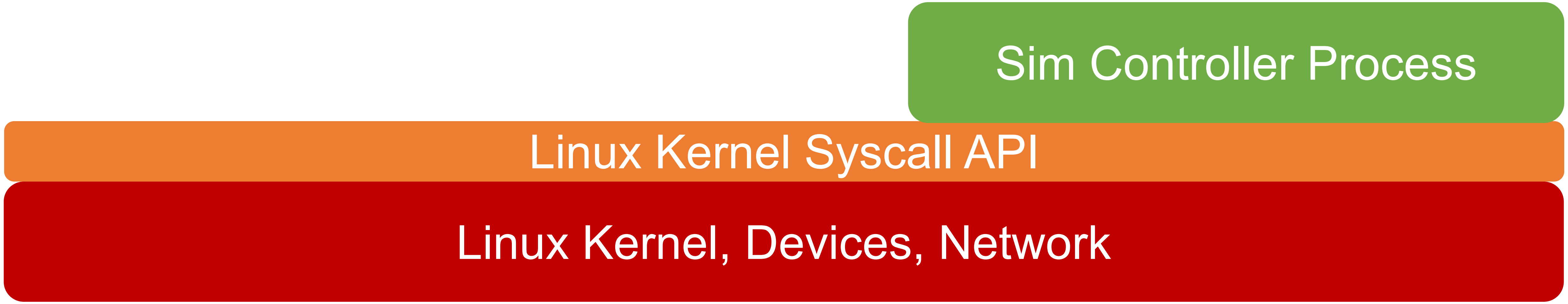
Goal: efficiently parallelize simulation workload



Thread scheduling:

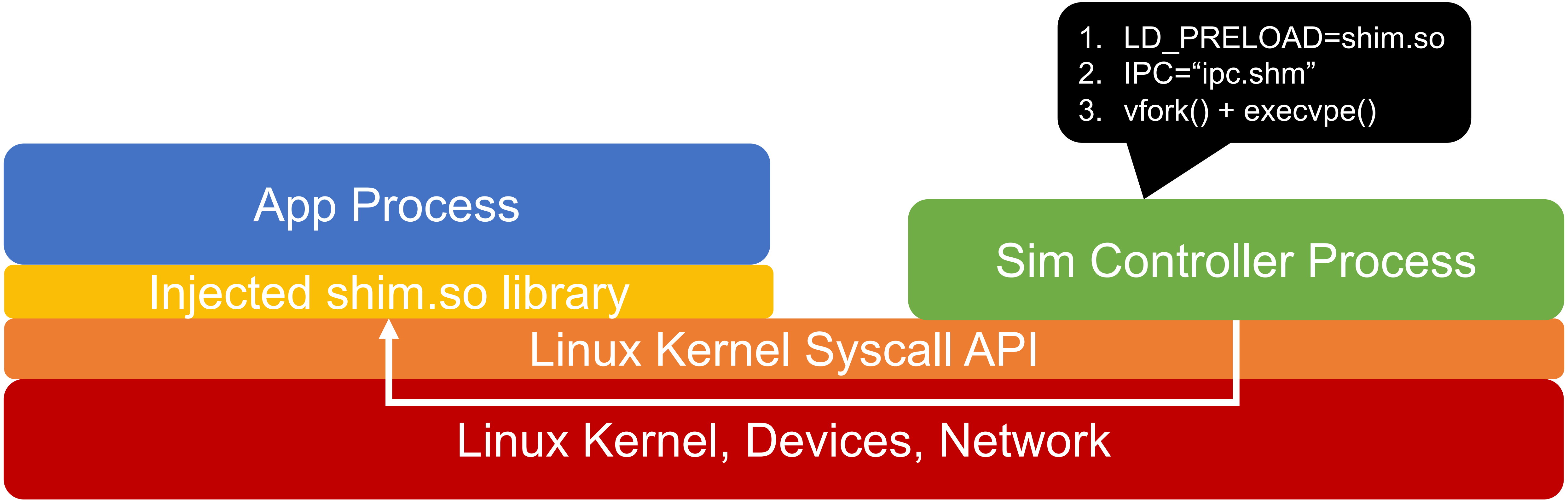
- Work stealing
- Each LP starts a thread
  1. Runs all assigned events in current round (1 ms)
  2. Set thread to *waiting*
  3. Starts next *waiting* thread (if any)
- When all threads *waiting*
  - Advance round clock
  - Repeat

Direct Execution



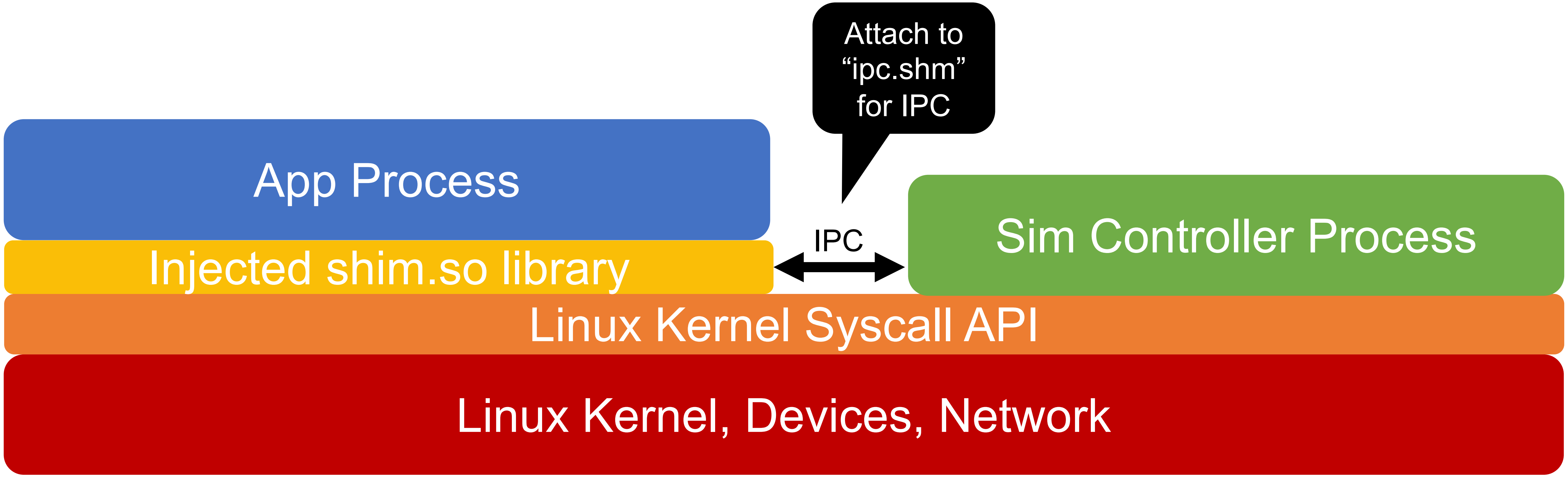
Direct Execution

1. parallel workers
2. **direct execution**
3. syscall interposition
4. syscall emulation
5. IPC
6. process control
7. CPU affinity



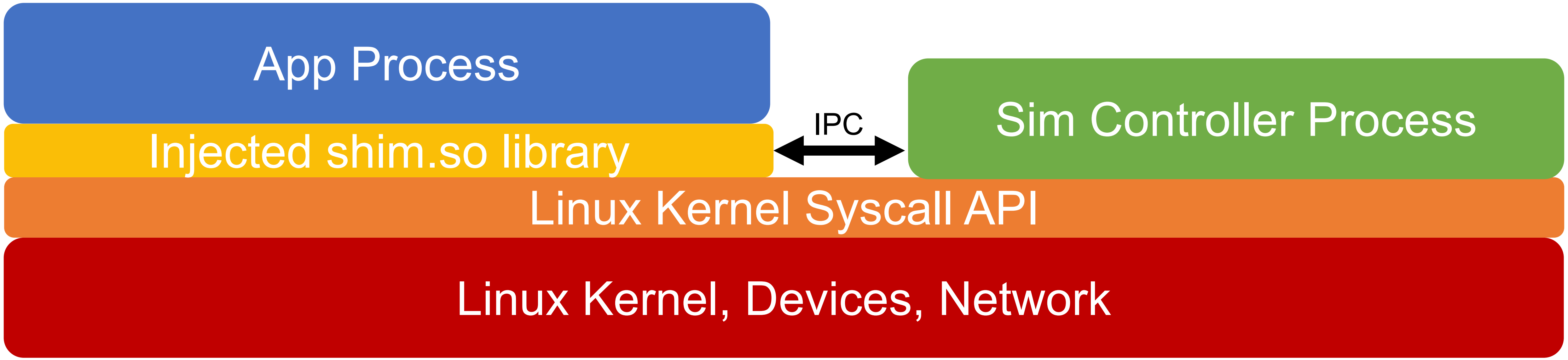
Direct Execution

1. parallel workers
2. **direct execution**
3. syscall interposition
4. syscall emulation
5. IPC
6. process control
7. CPU affinity

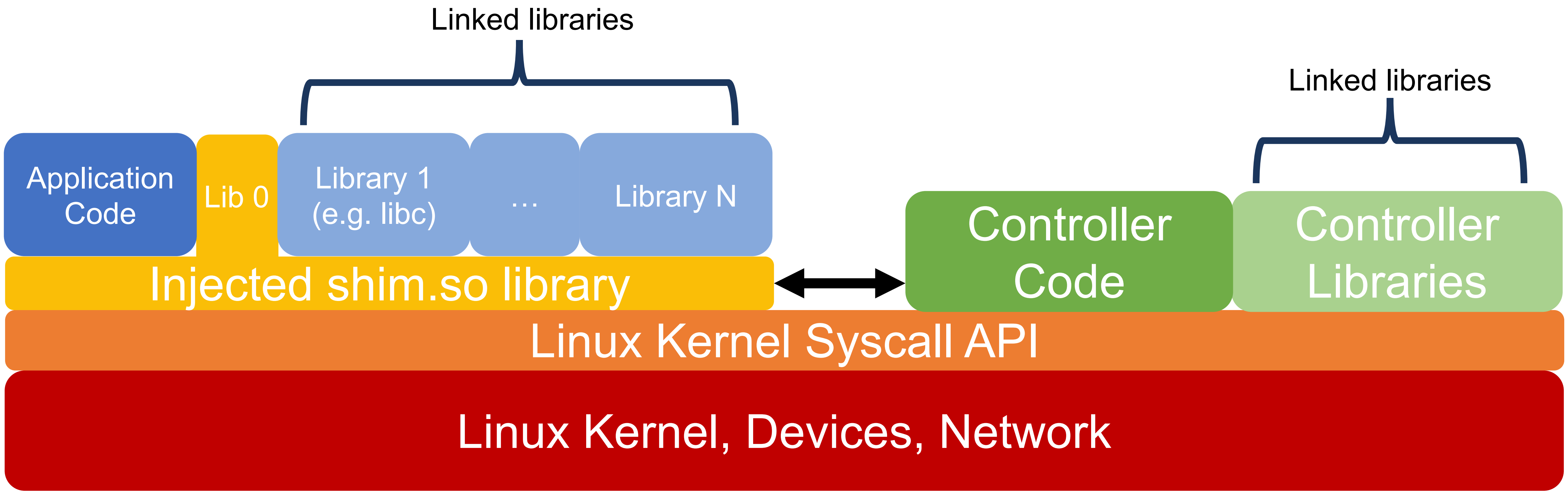


Direct Execution

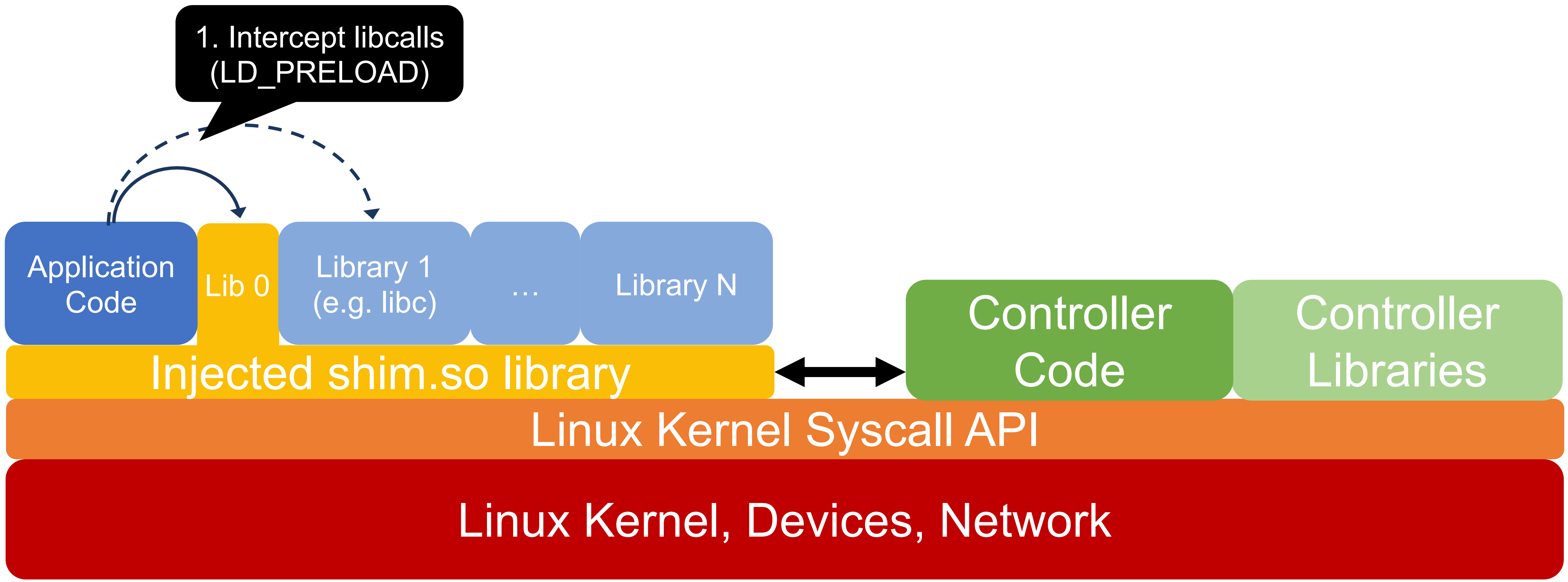
App is running and can communicate with controller



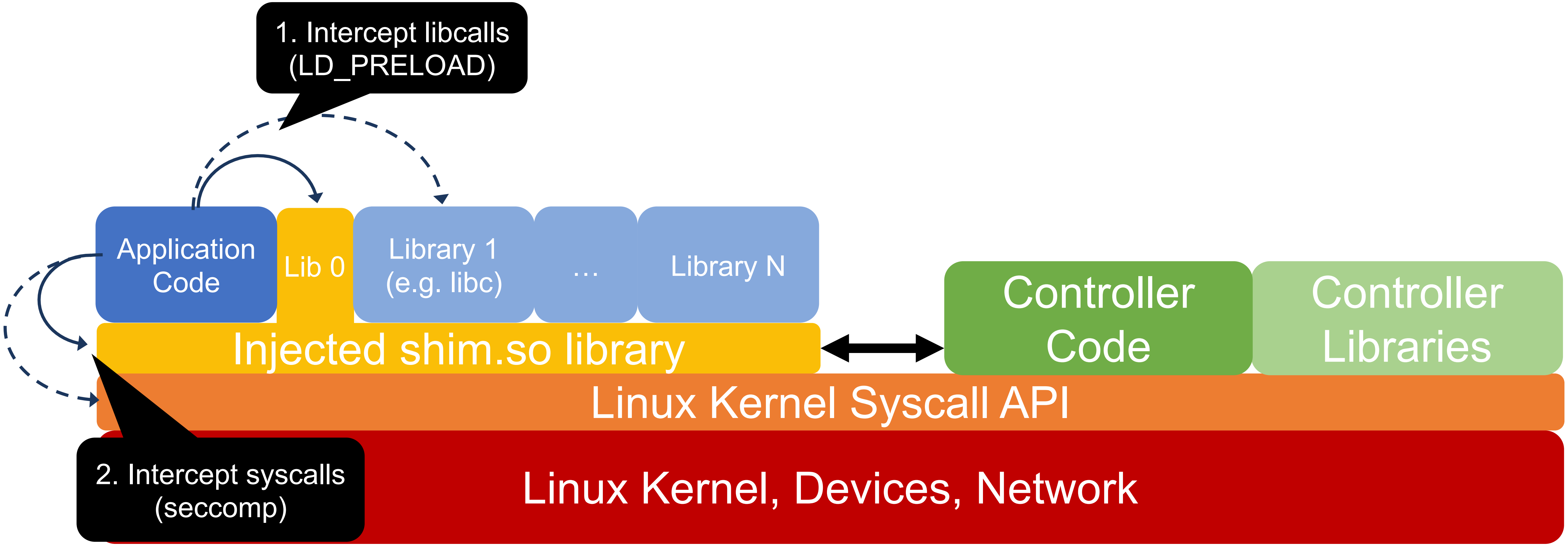
Syscall Interposition



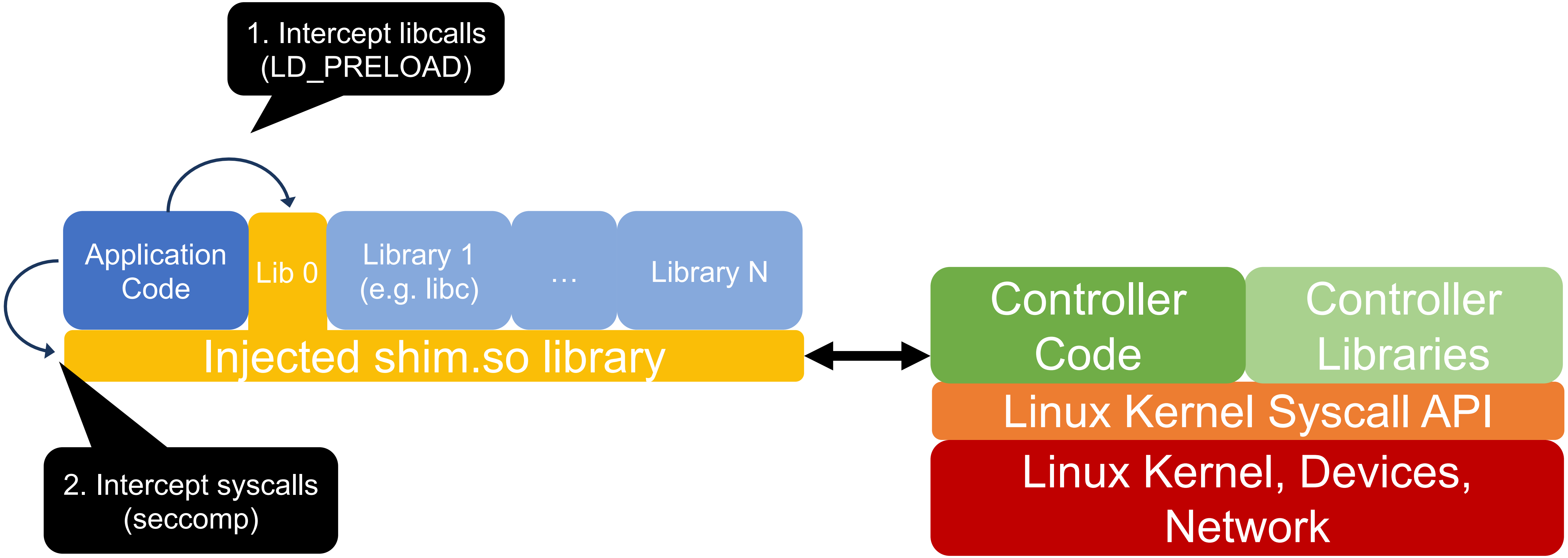
Syscall Interposition



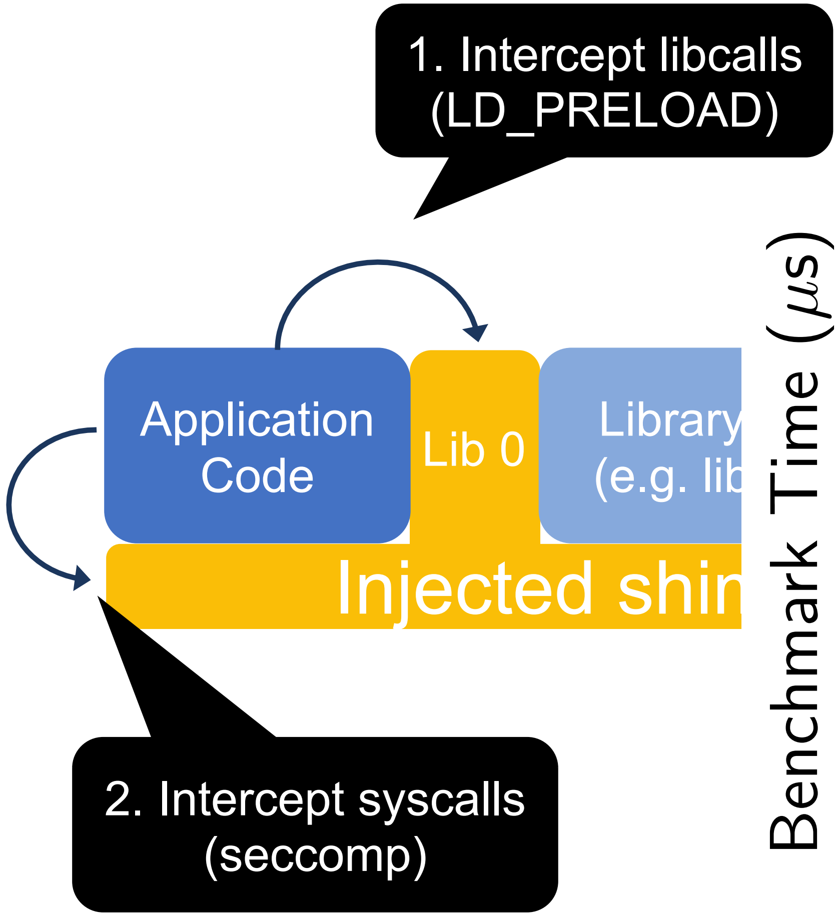
Syscall Interposition

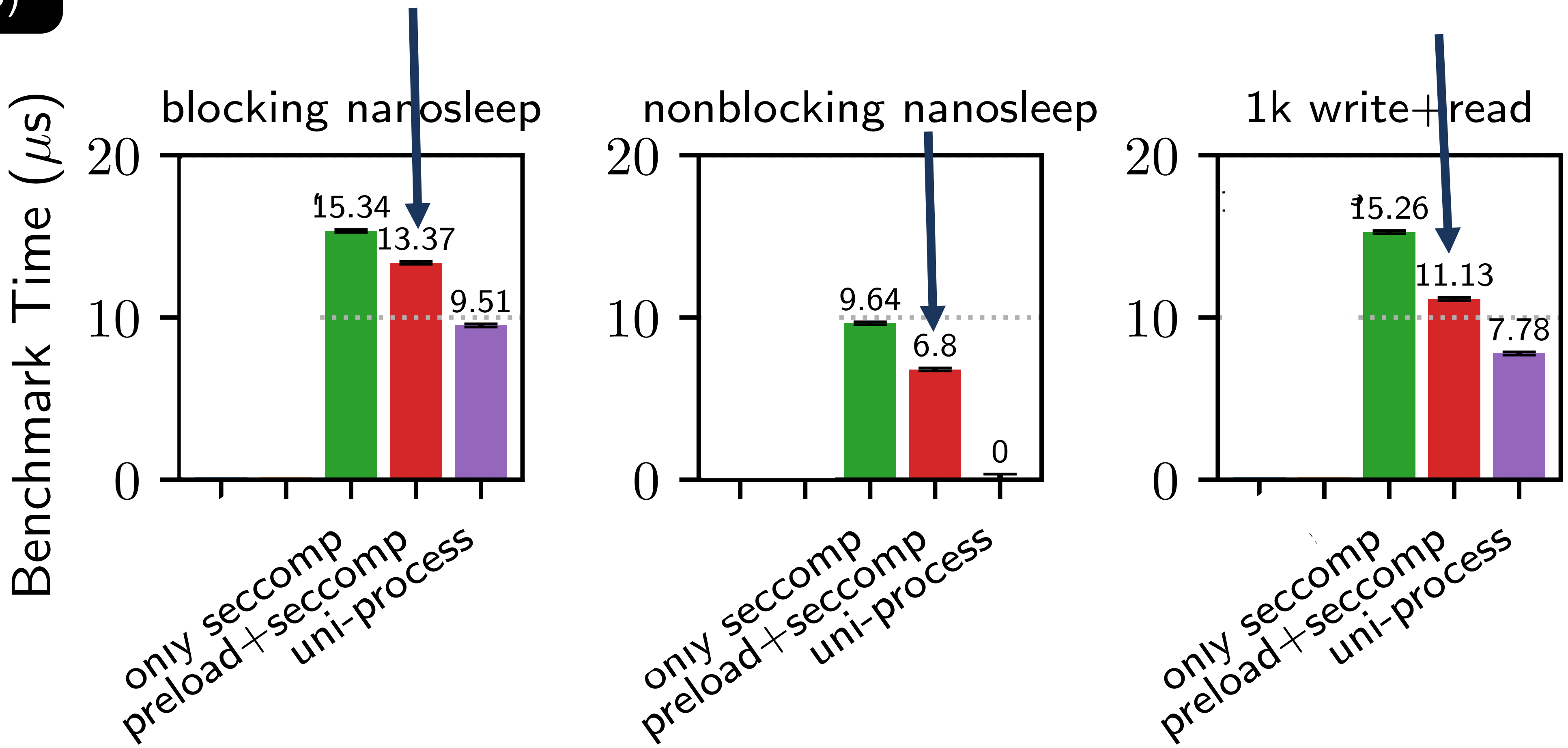


Syscall Interposition



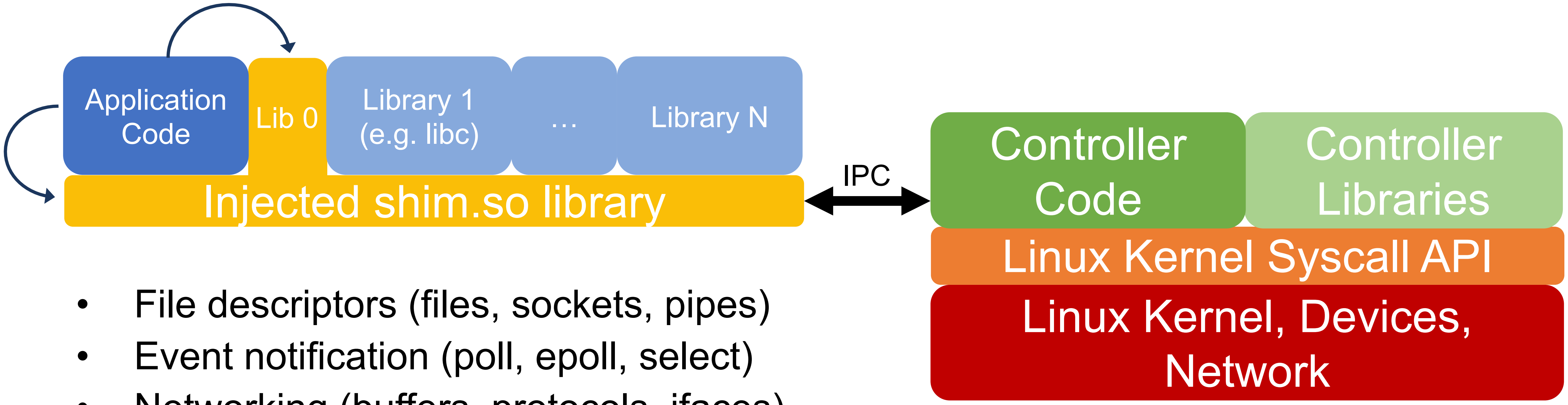
Syscall Interposition





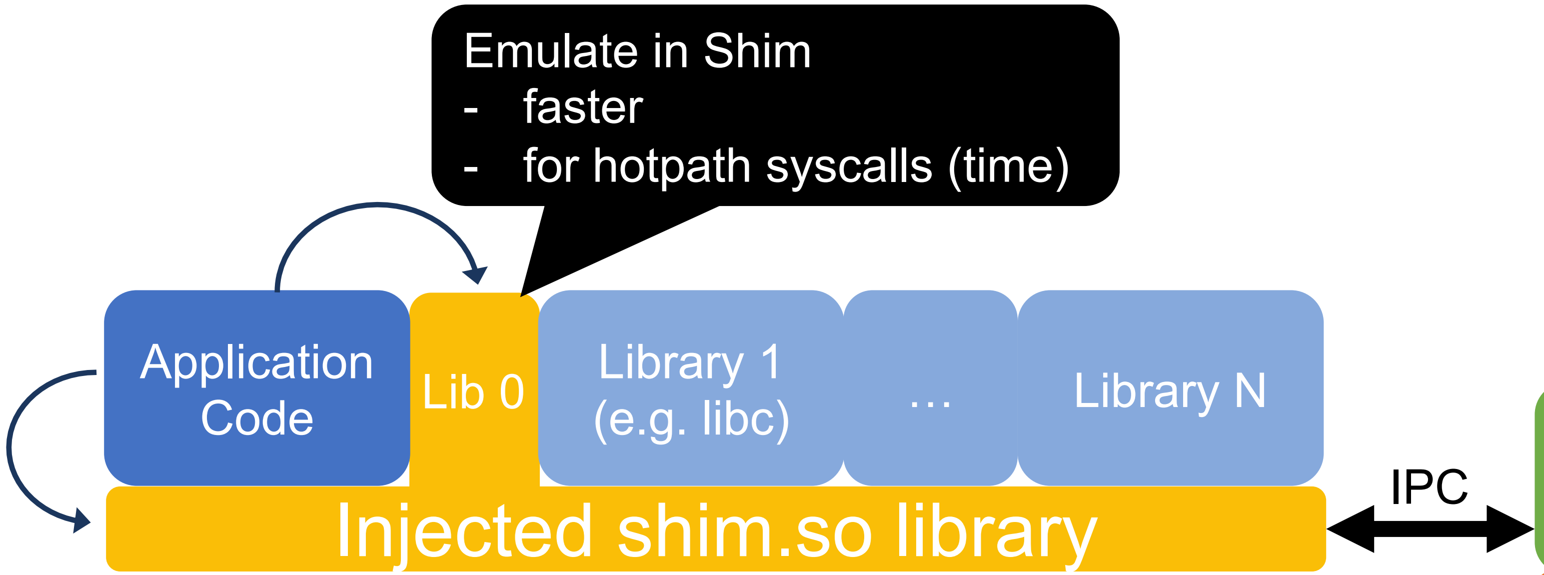
Syscall Emulation

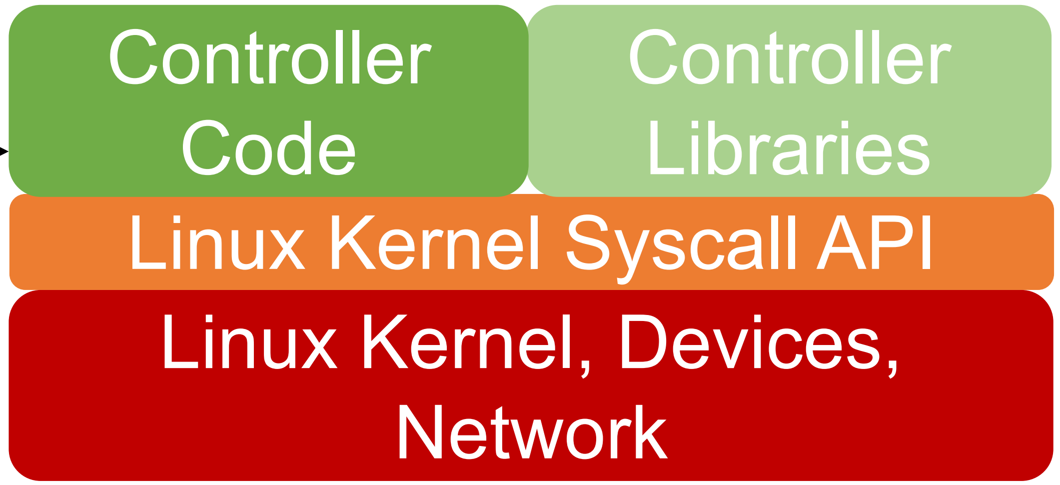
1. parallel workers
2. direct execution
3. syscall interposition
4. **syscall emulation**
5. IPC
6. process control
7. CPU affinity



- File descriptors (files, sockets, pipes)
- Event notification (poll, epoll, select)
- Networking (buffers, protocols, ifaces)
- DNS and routing (latency, bandwidth)

Syscall Emulation



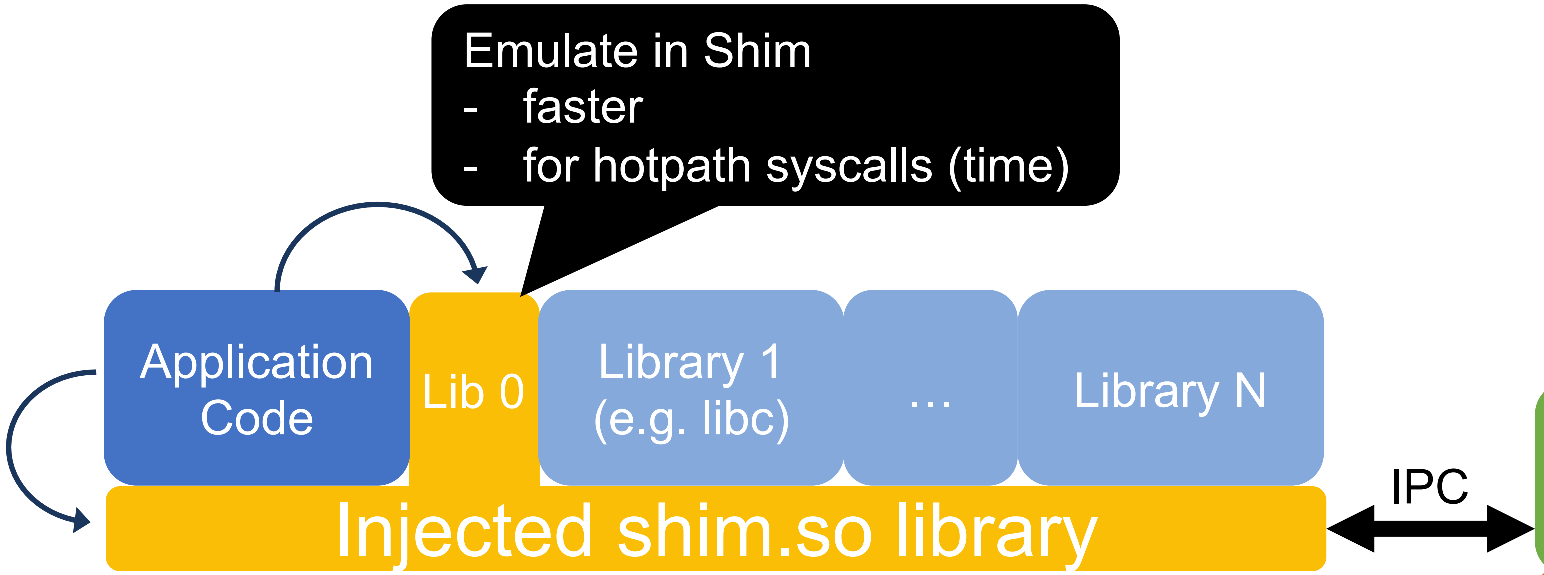


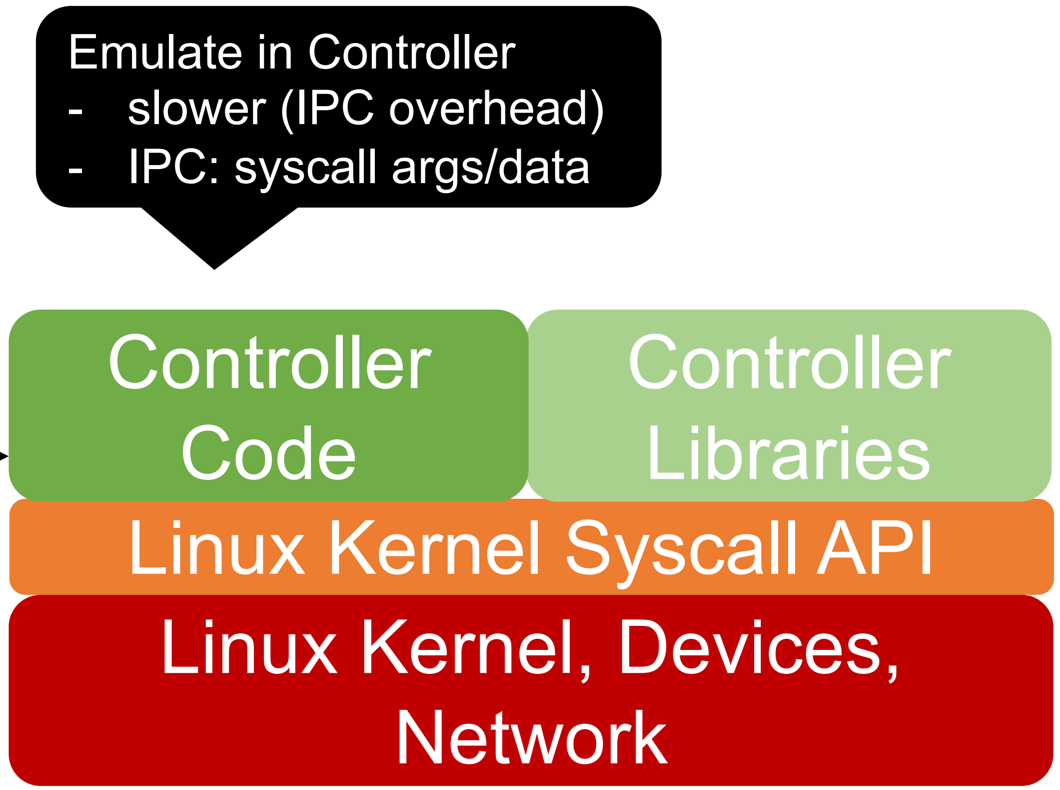
- File descriptors (files, sockets, pipes)
- Event notification (poll, epoll, select)
- Networking (buffers, protocols, ifaces)
- DNS and routing (latency, bandwidth)

U.S. NAVAL RESEARCH LABORATORY

Syscall Emulation

1. parallel workers
2. direct execution
3. syscall interposition
4. **syscall emulation**
5. IPC
6. process control
7. CPU affinity

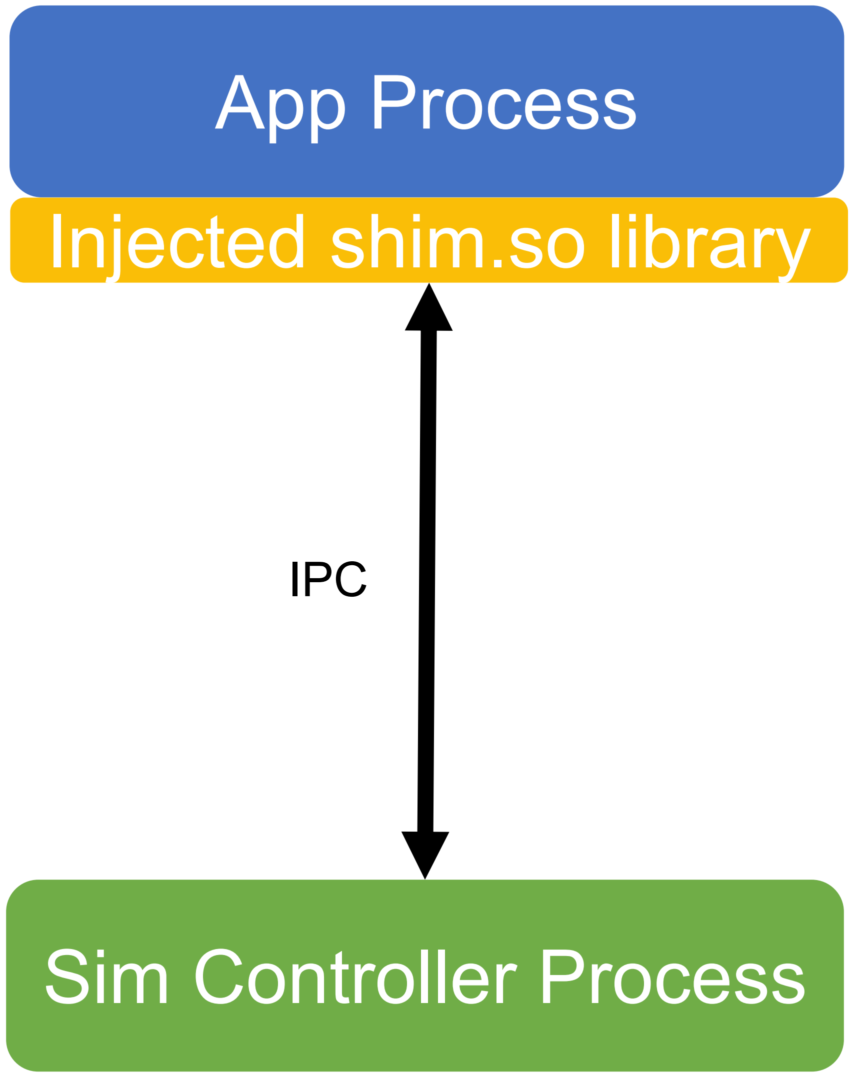




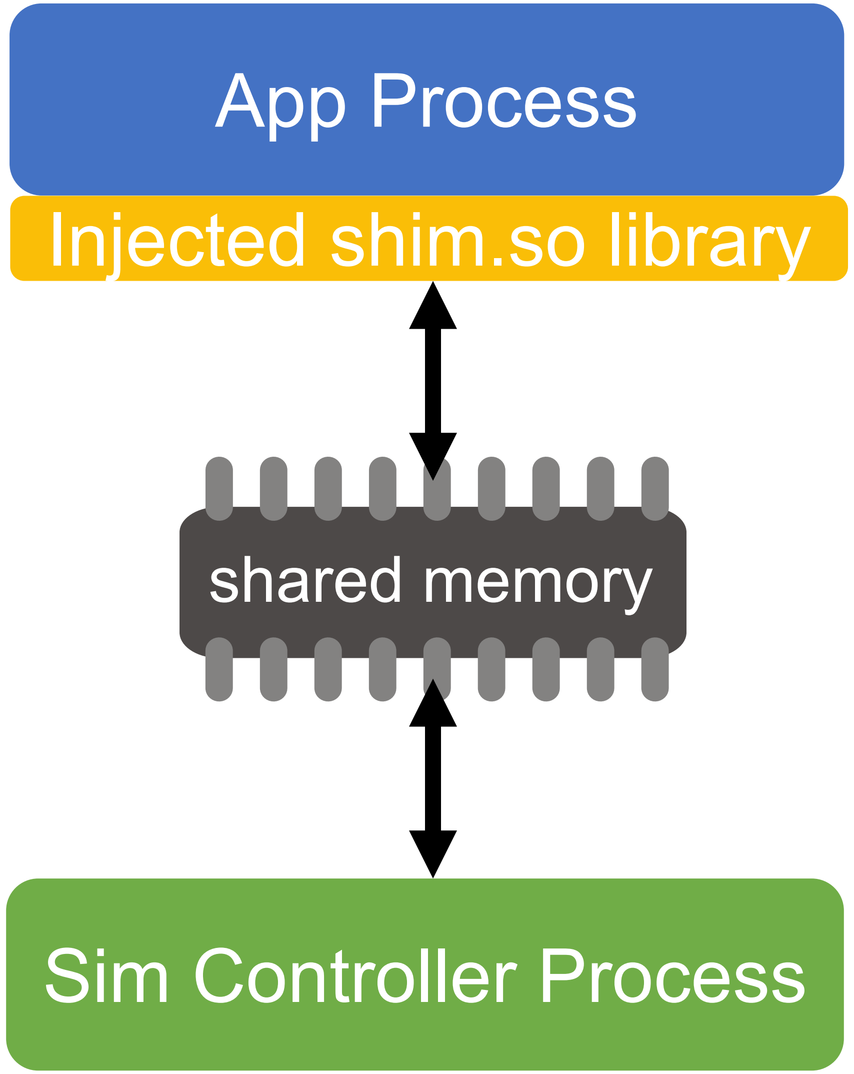
- File descriptors (files, sockets, pipes)
- Event notification (poll, epoll, select)
- Networking (buffers, protocols, ifaces)
- DNS and routing (latency, bandwidth)

Inter-Process Communication

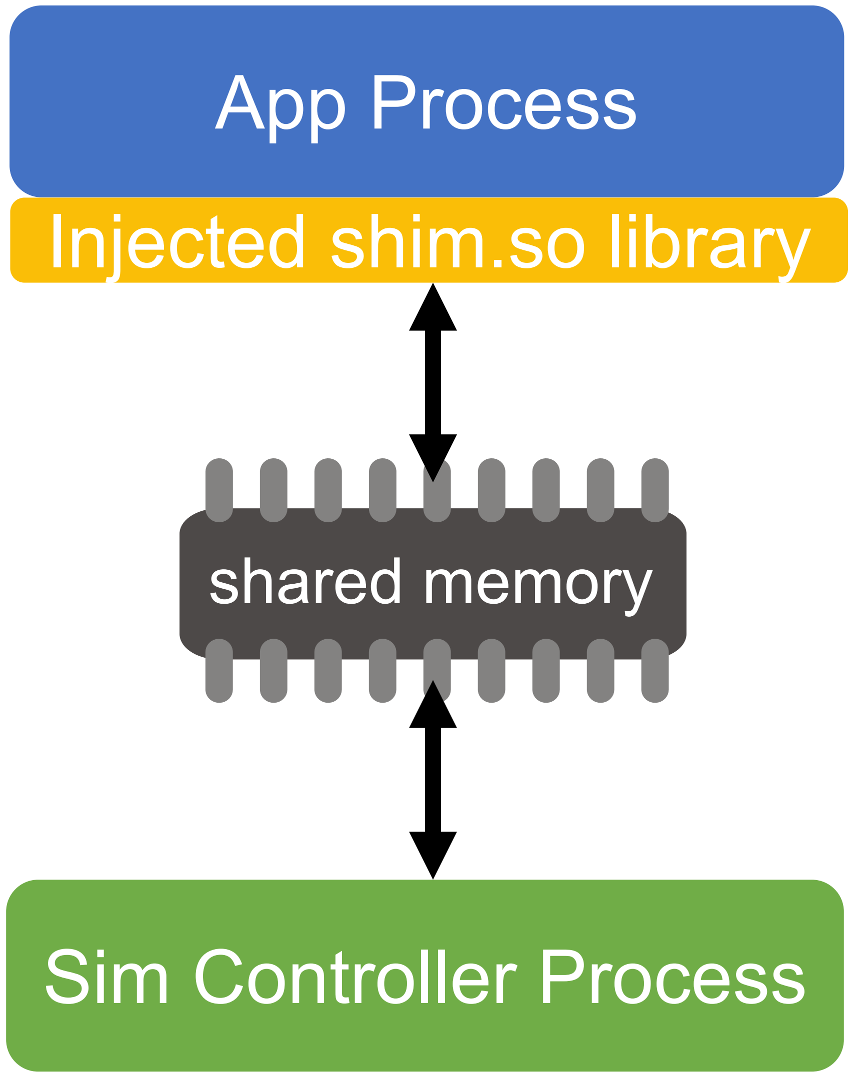
1. parallel workers
2. direct execution
3. syscall interposition
4. syscall emulation
5. **IPC**
6. process control
7. CPU affinity



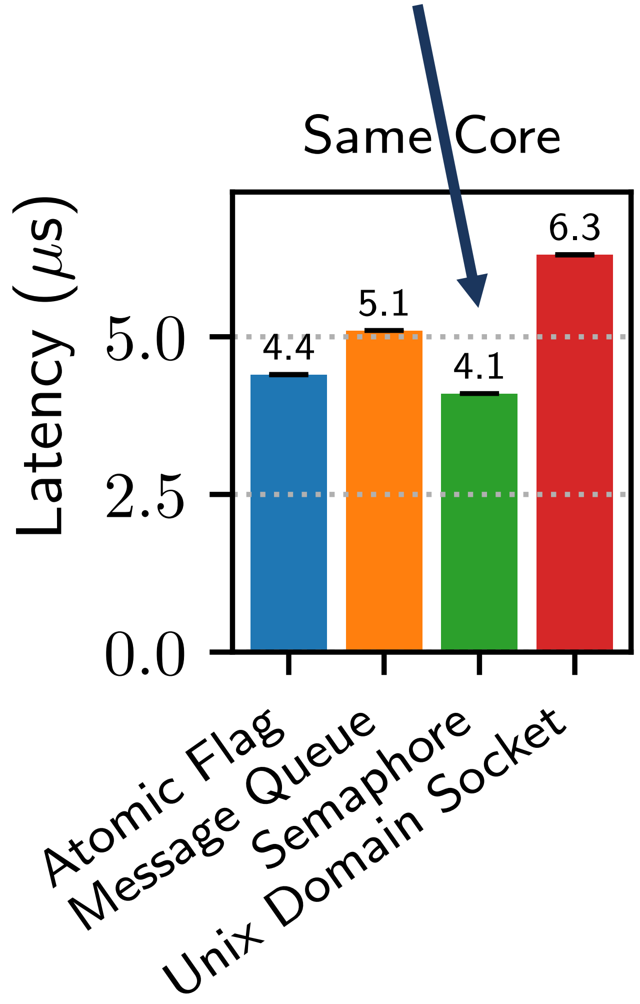
Inter-Process Communication



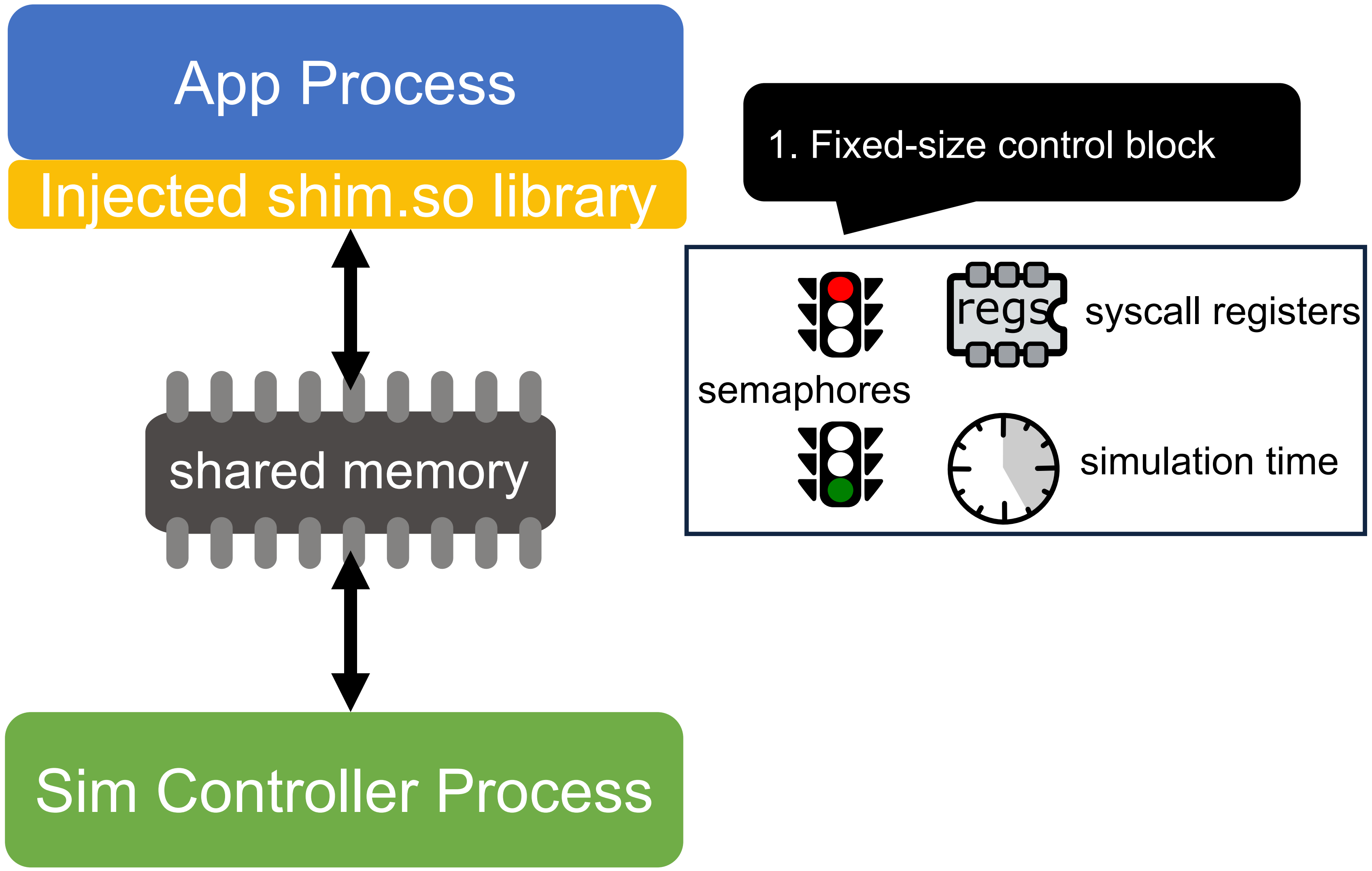
Inter-Process Communication



Shared memory + semaphores is the fastest IPC method for two processes running on the same core

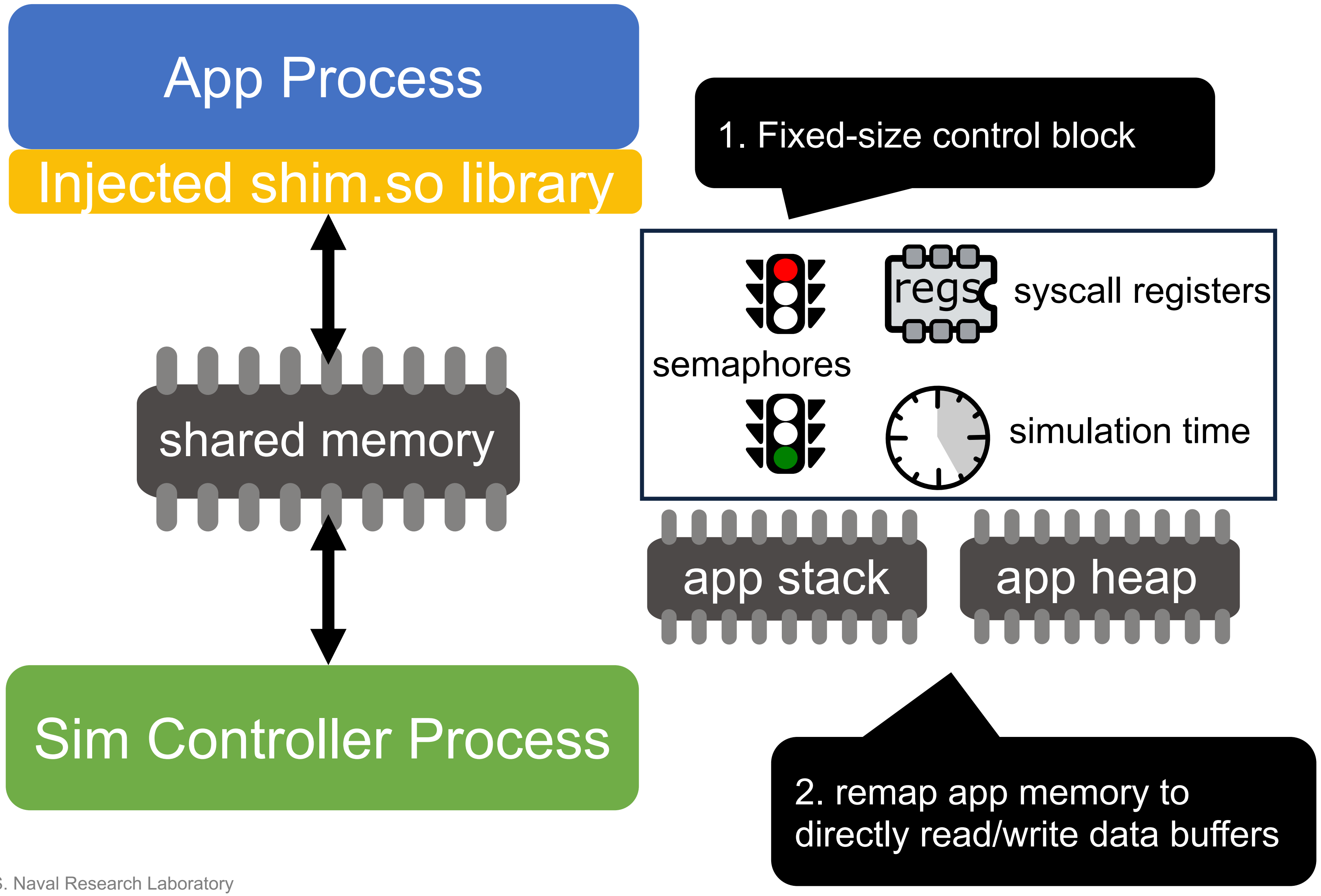


Inter-Process Communication



Inter-Process Communication

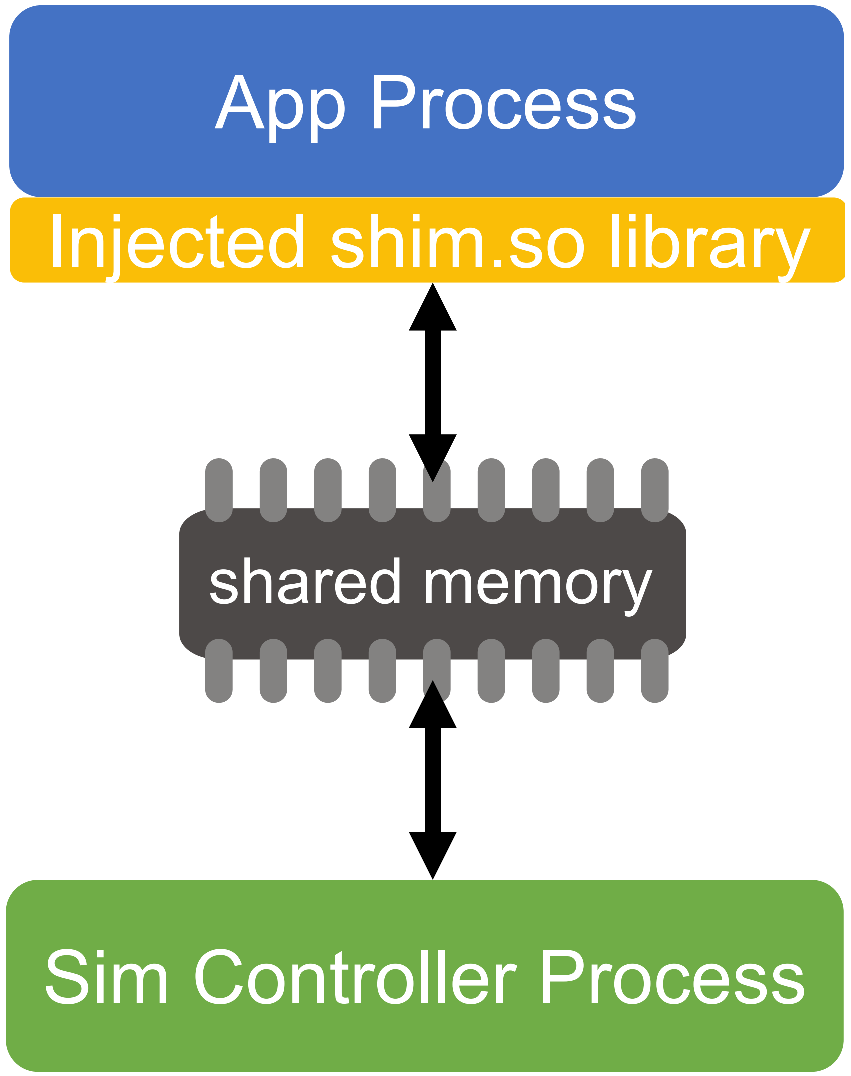
1. parallel workers
2. direct execution
3. syscall interposition
4. syscall emulation
5. **IPC**
6. process control
7. CPU affinity

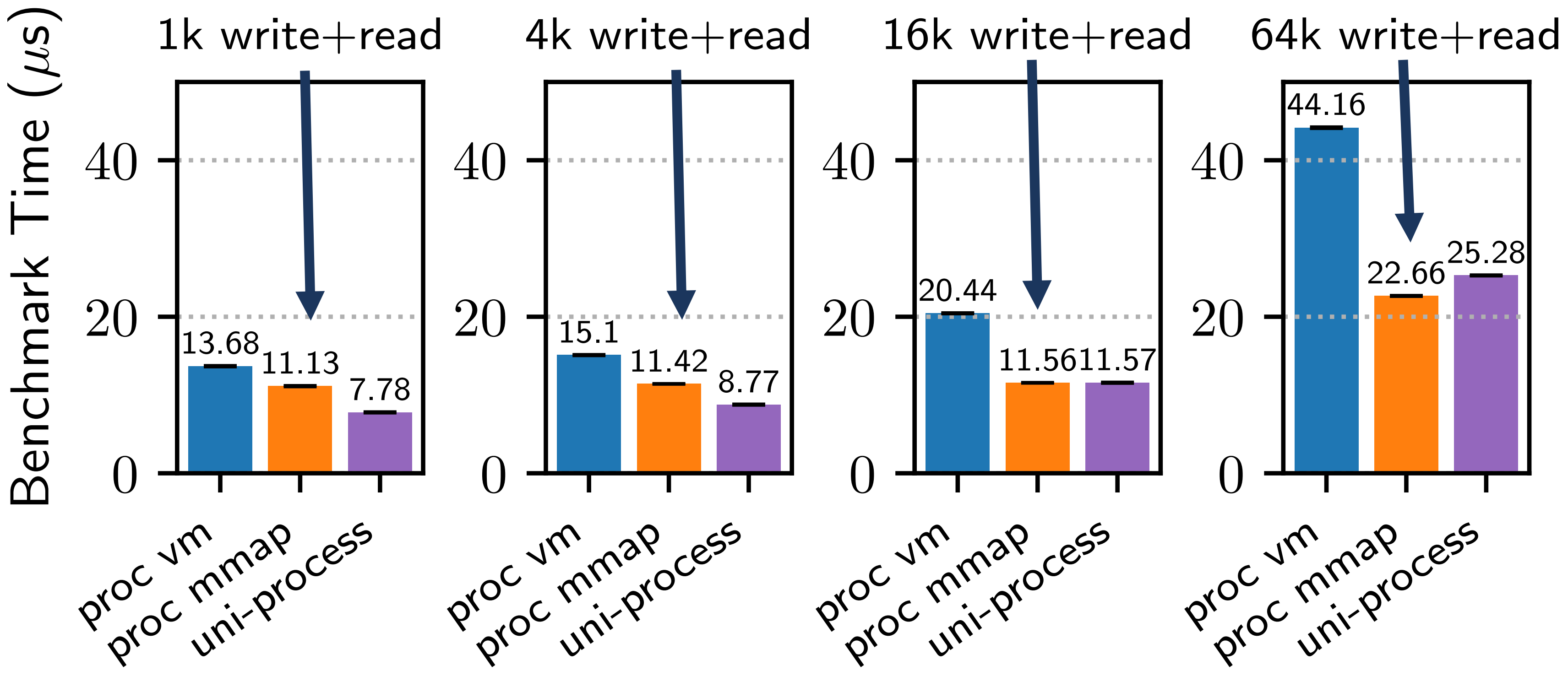


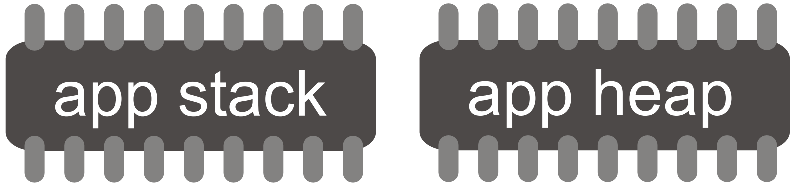
U.S. NAVAL RESEARCH LABORATORY

Inter-Process Communication

1. parallel workers
2. direct execution
3. syscall interposition
4. syscall emulation
5. **IPC**
6. process control
7. CPU affinity

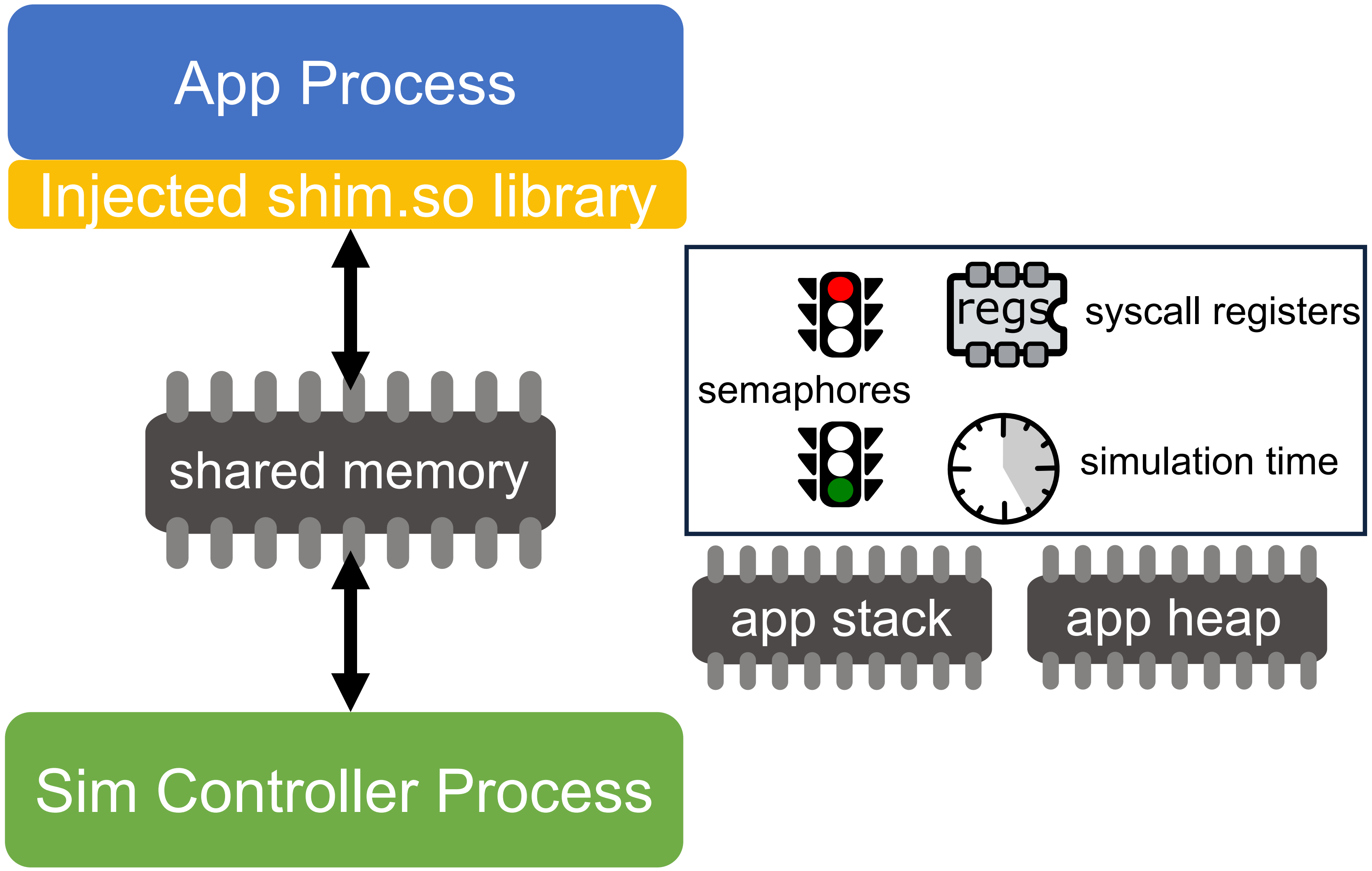




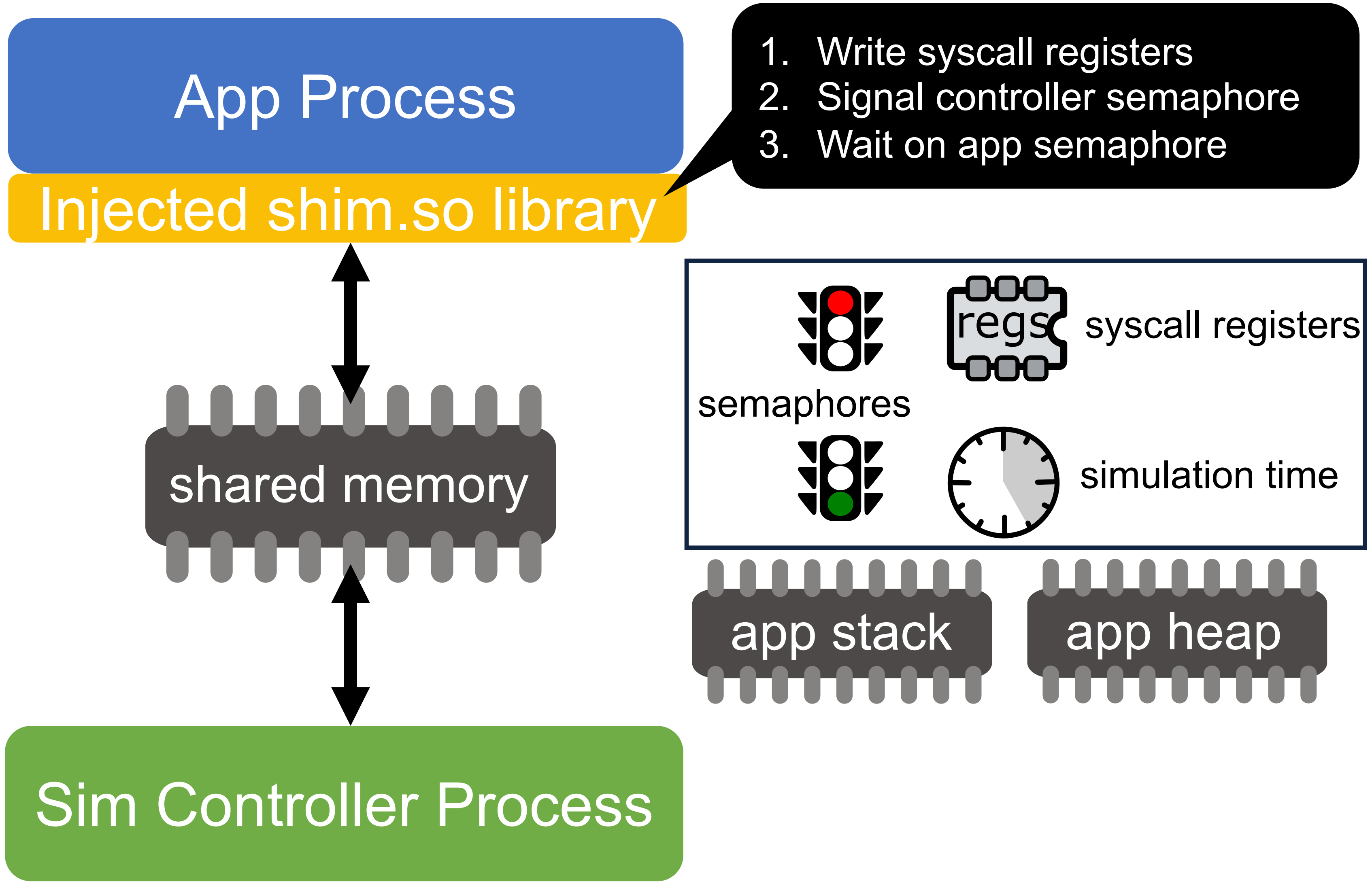


2. remap app memory to directly read/write data buffers

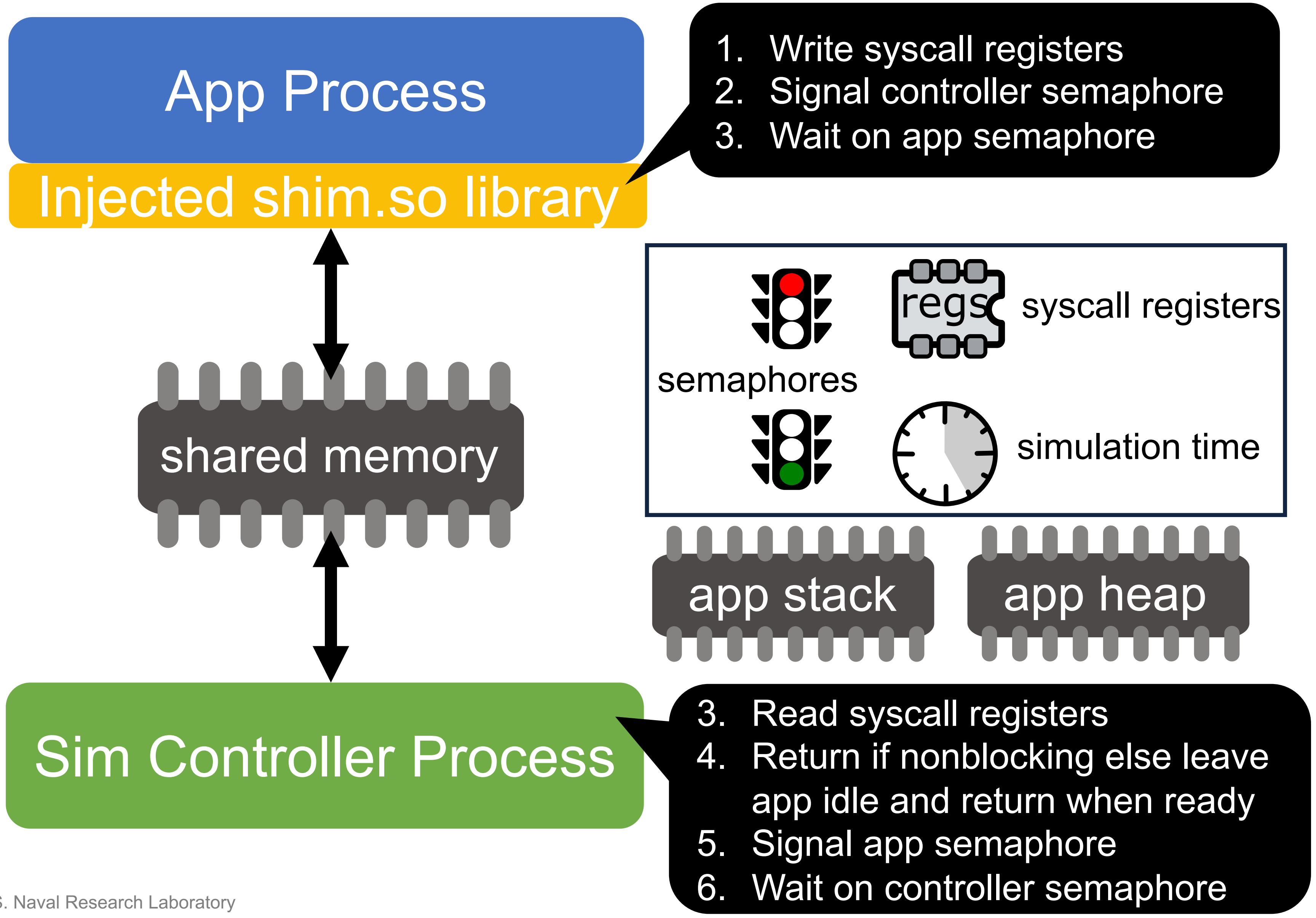
Process Control



Process Control

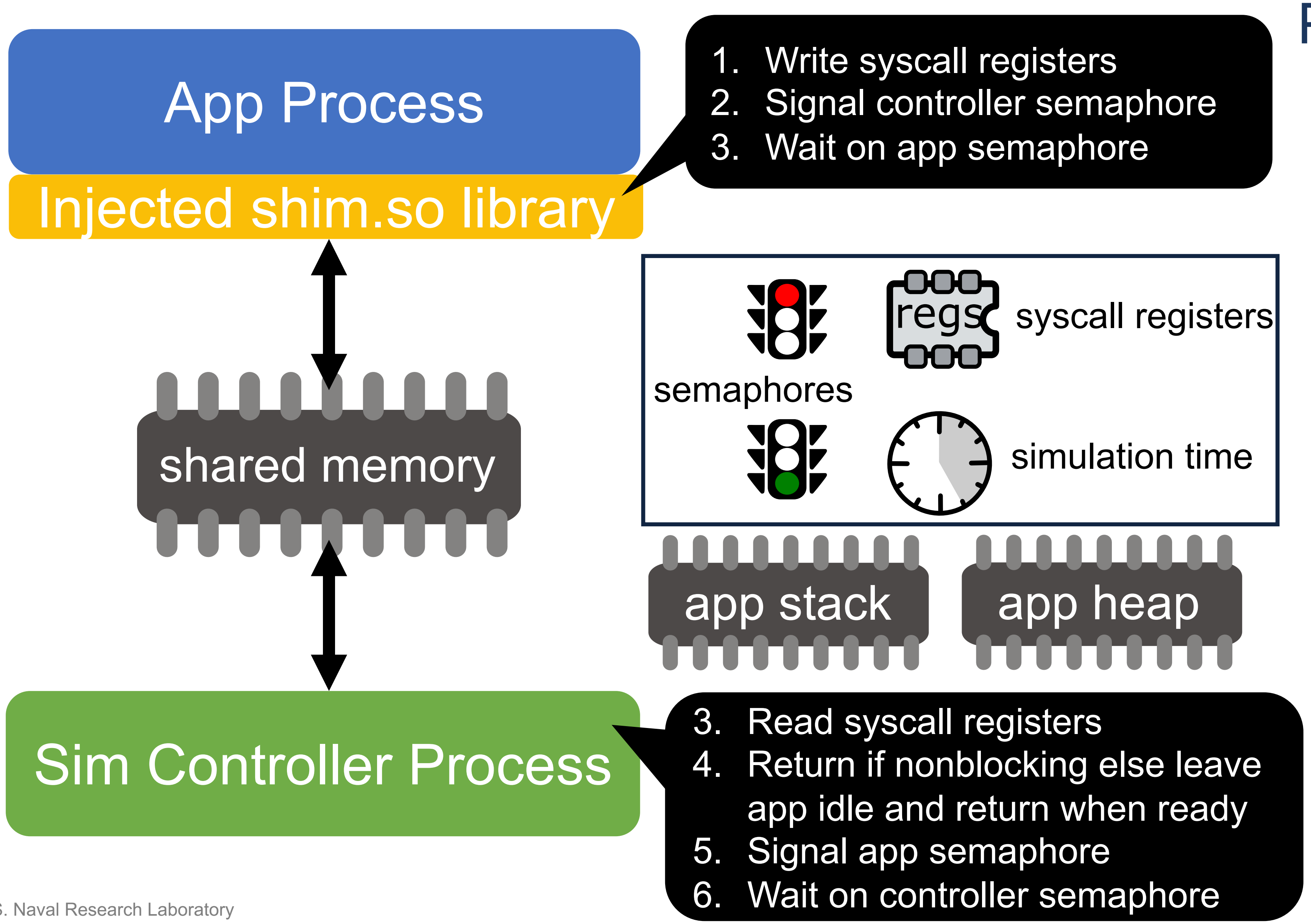


Process Control



U.S. NAVAL RESEARCH LABORATORY

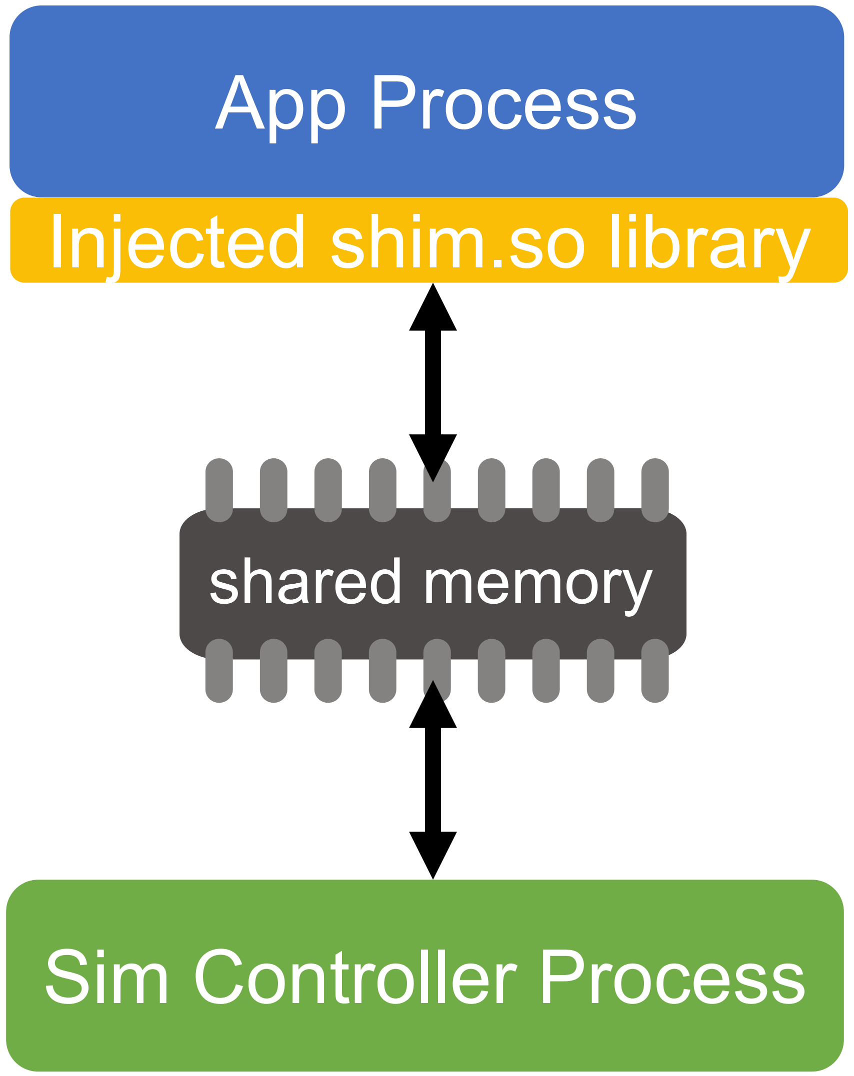
Process Control



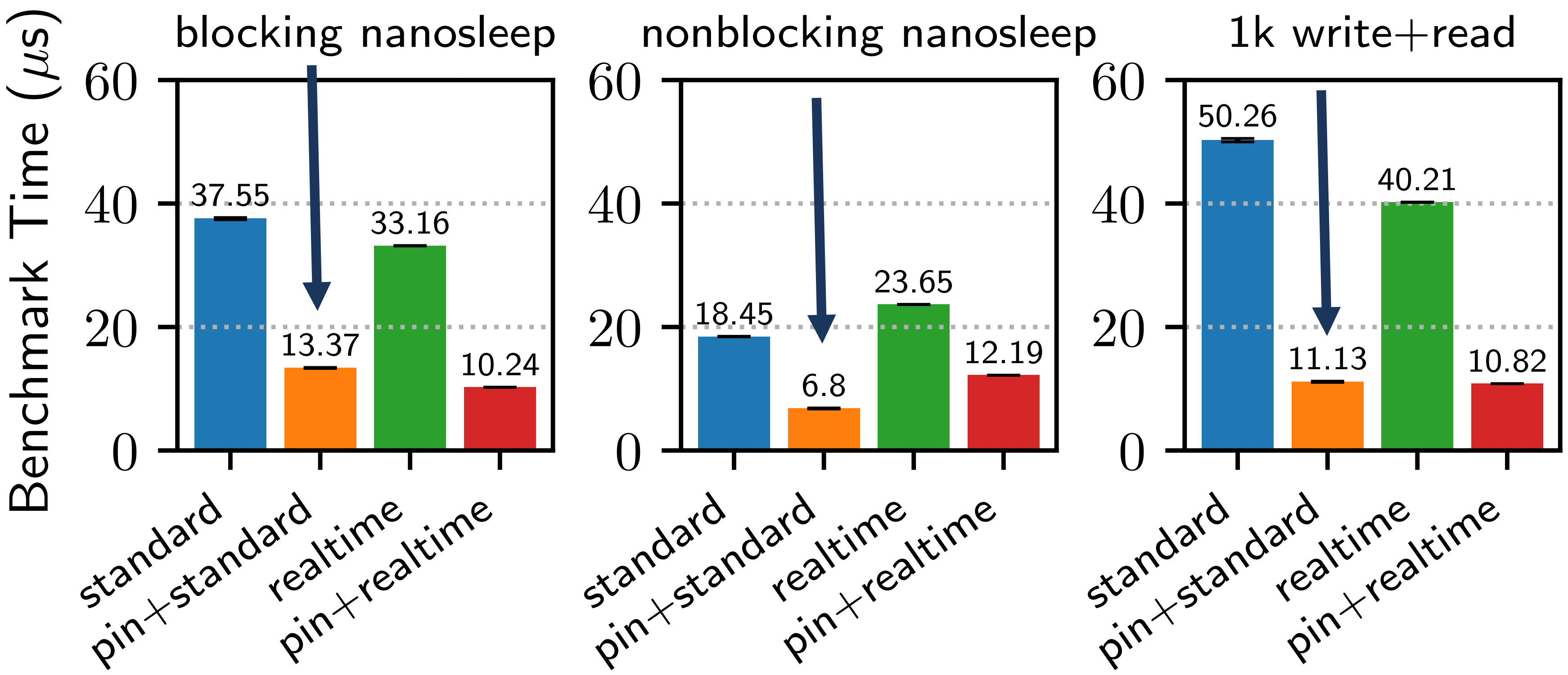
Properties:

- Controller worker thread and its app process run synchronously
- Ensures nonconcurrent access to app stack and heap memory

Process Control



Use CPU pinning to pin each worker thread and all of its managed processes to the same core



Outline

motivation

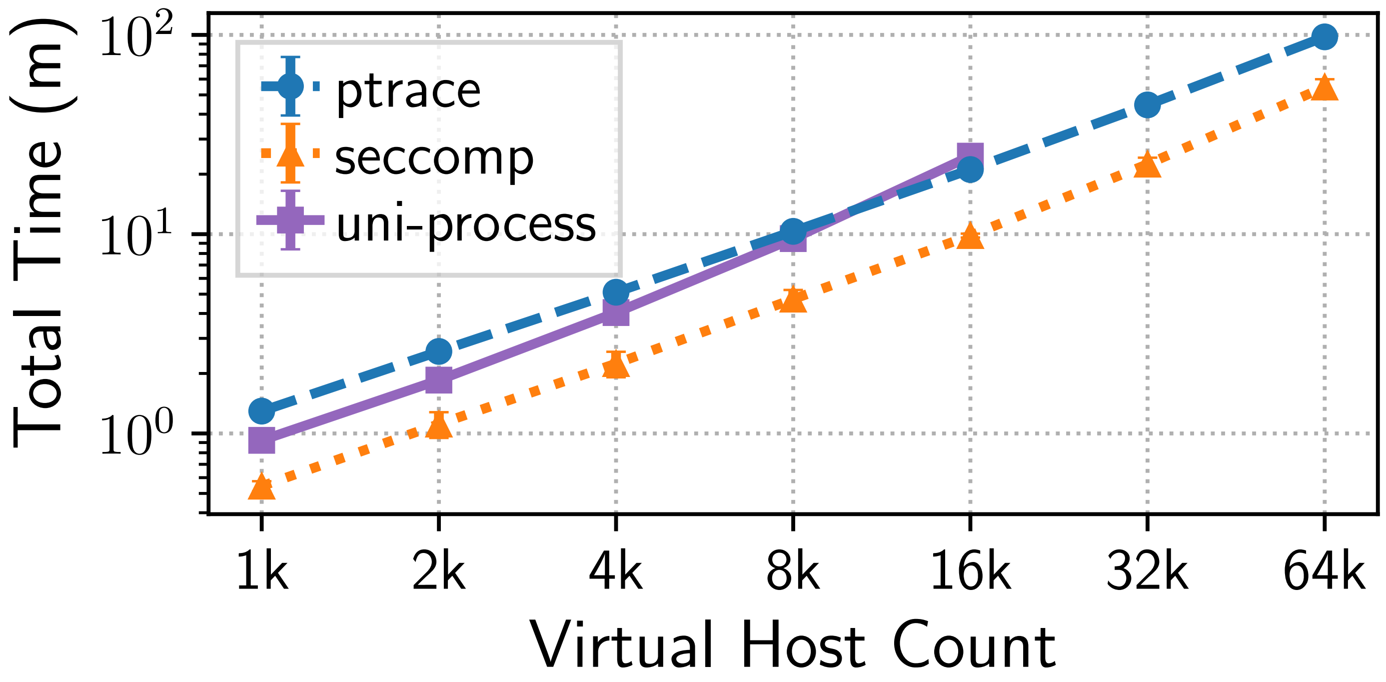
design

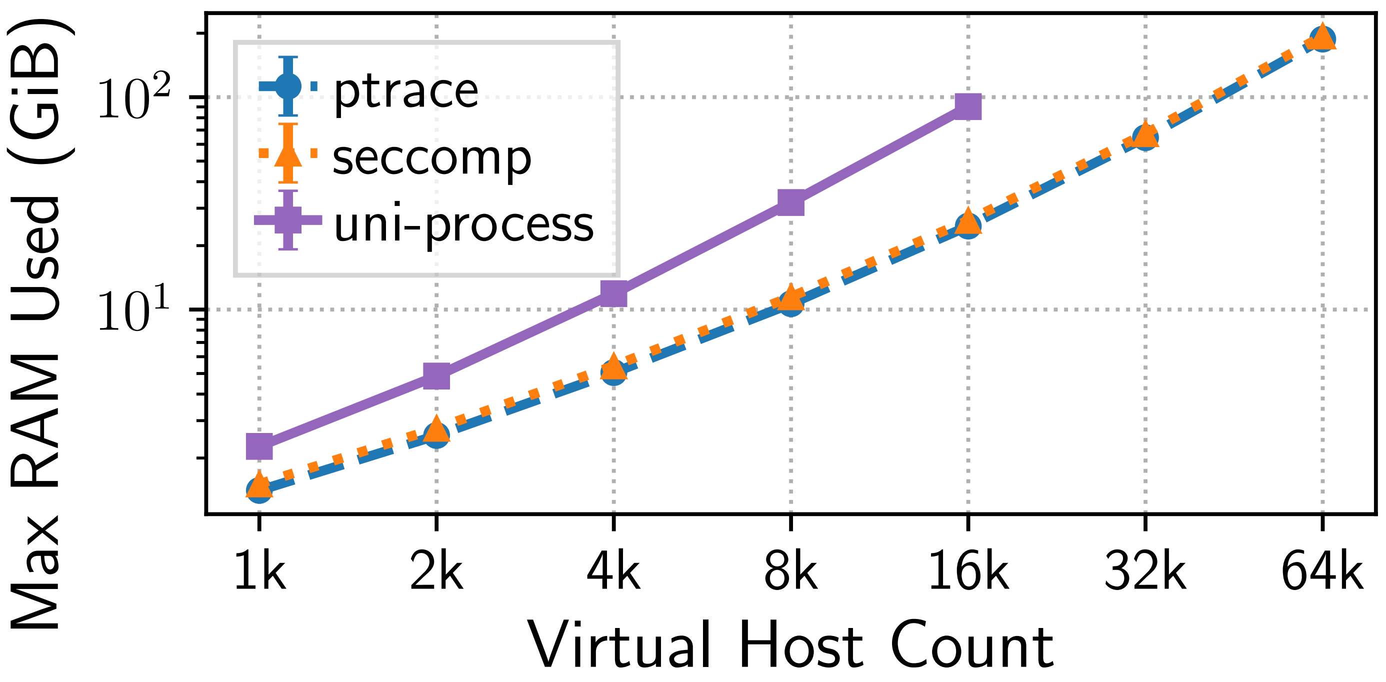
evaluation

Evaluation: Large P2P Benchmarks

Faster and more scalable than the uni-process, plugin architecture!

Uses significantly less memory than the uni-process, plugin architecture!

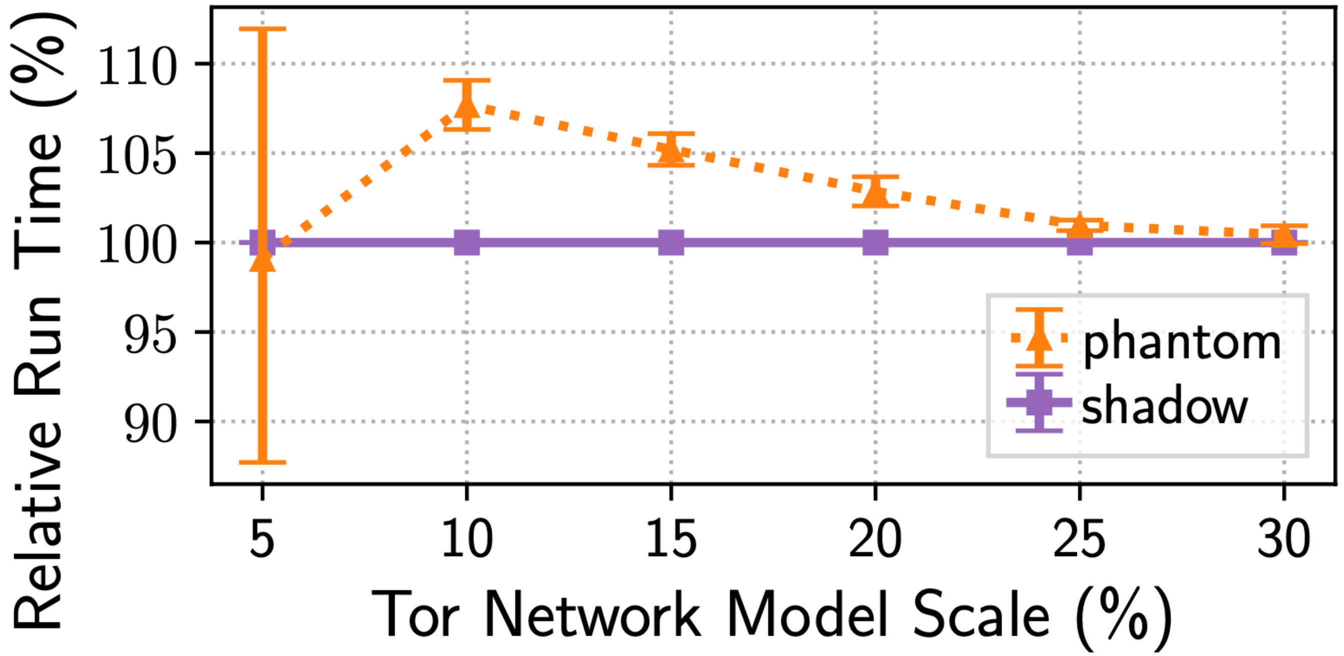


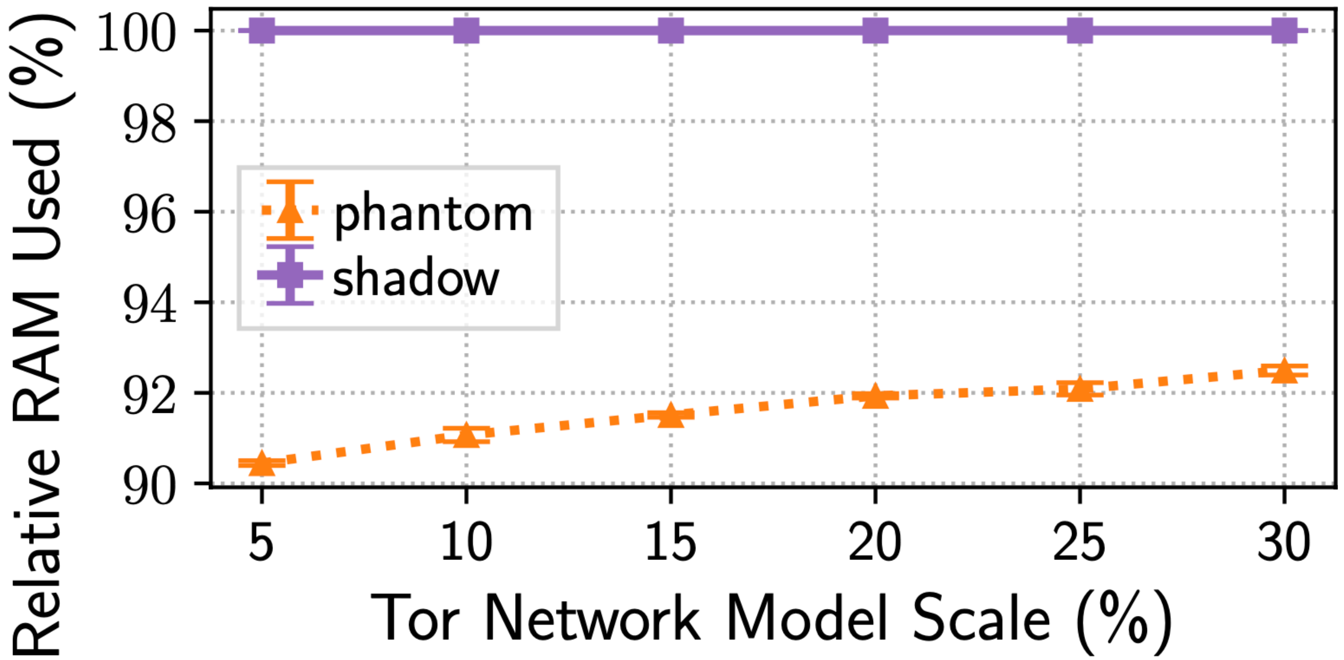


Evaluation: Large Tor Networks

Performance comparable to the state of the art for large Tor networks

Uses significantly less memory than the uni-process, plugin architecture!





U.S. NAVAL RESEARCH LABORATORY

Main Takeaways

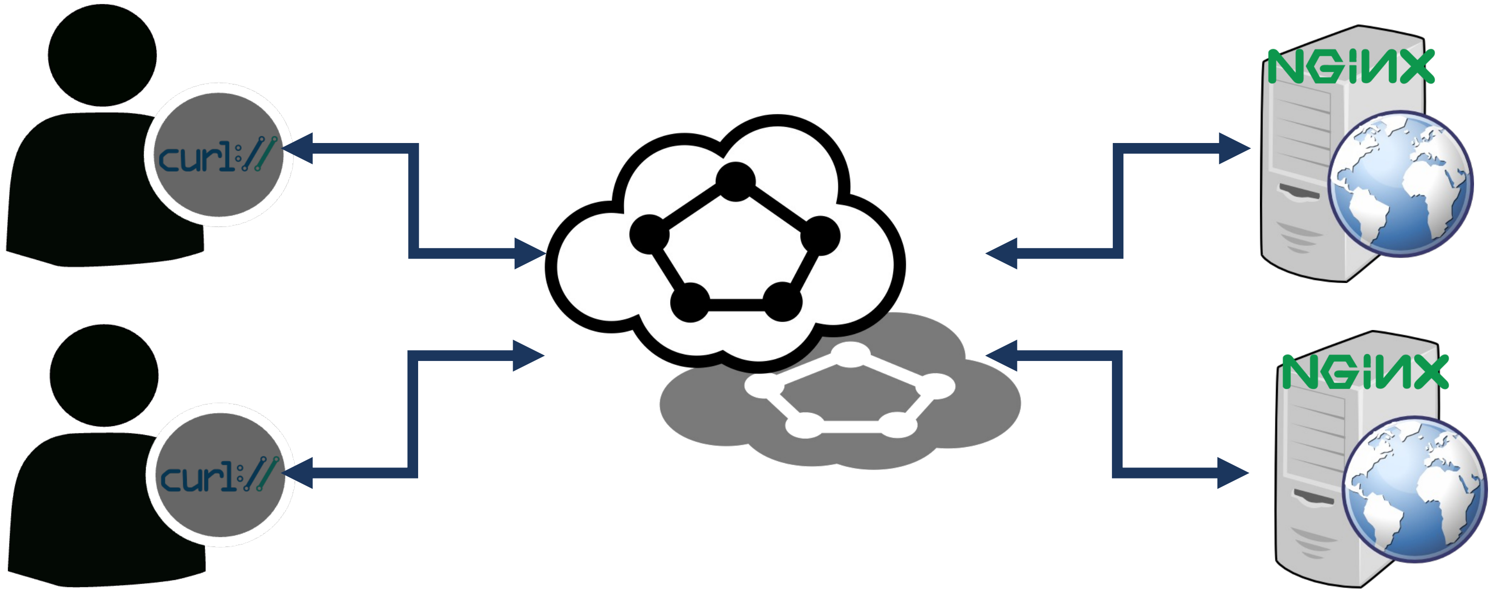
Read the paper!

shadow on github

shadow.github.io

Designed a new, hybrid network simulator/emulator

- co-opts Linux processes into a discrete-event network simulation that emulates kernel functionality
- enables large-scale, distributed system experiments



- Merged into the open-source Shadow project and synonymous with Shadow v2
- Artifacts: https://netsim-atc2022.github.io

- 2.3x faster than Shadow v1
- 3.4x faster than NS-3
- 43x faster than gRaIL [ToN'19]