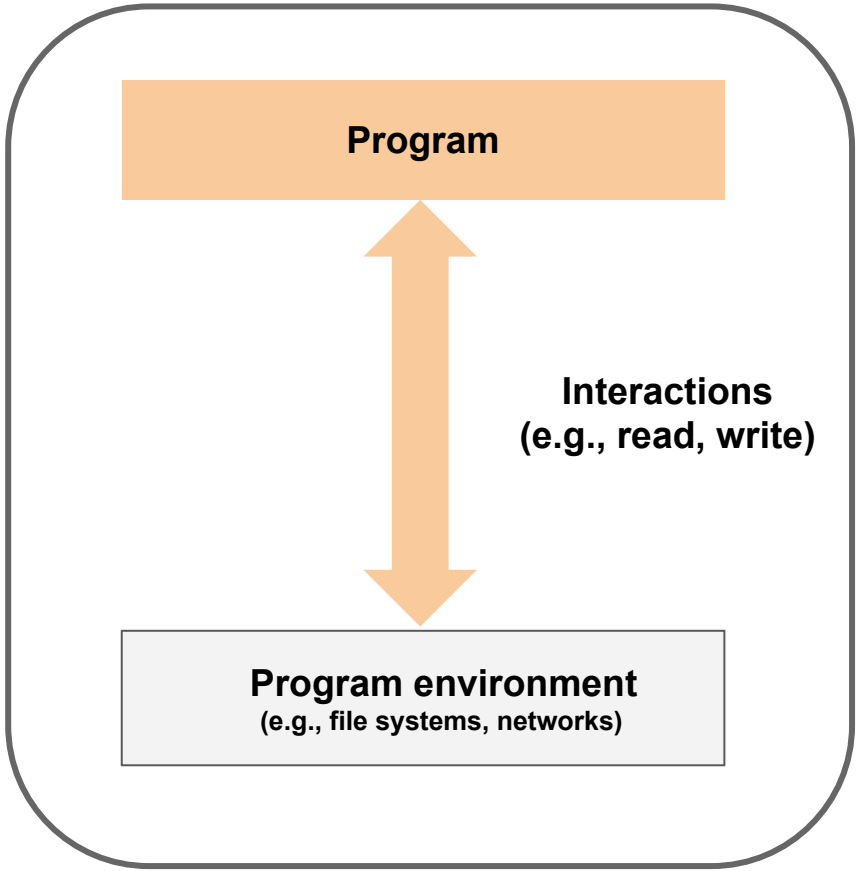
Mousse: A System for Selective Symbolic Execution of Programs with Untamed Environments

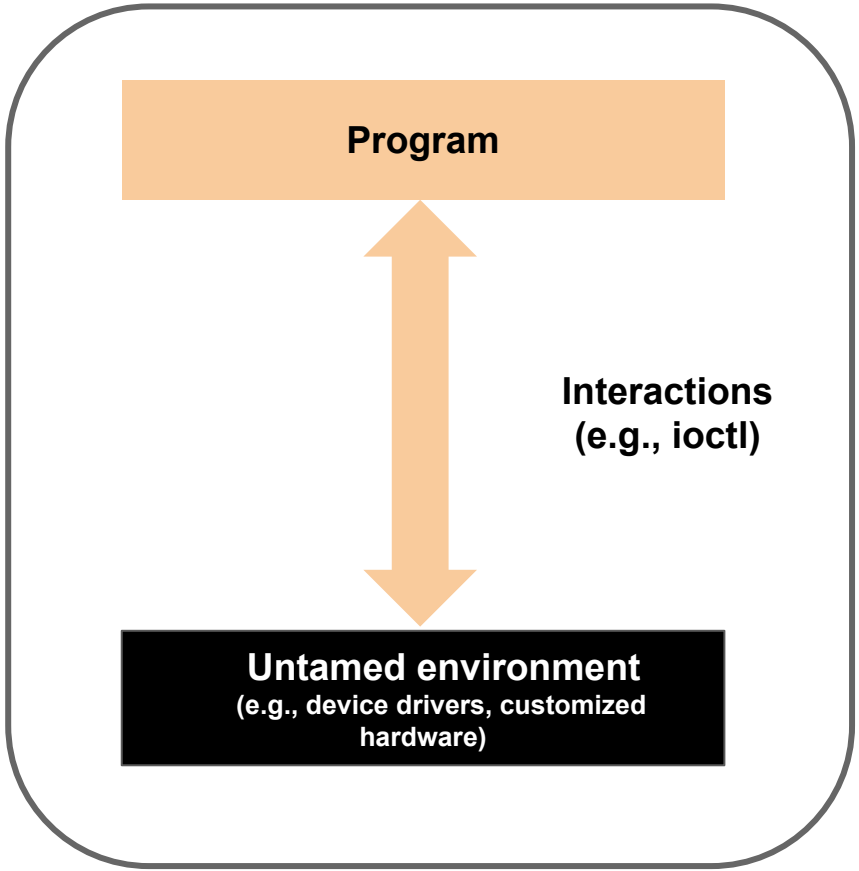
Yingtong Liu, Hsin-Wei Hung, Ardalan Amiri Sani
University of California, Irvine

What are programs with untamed environments?

Program environment



Untamed environment



Example programs with untamed environments

OS services

Example programs with untamed environments

OS services

Software frameworks for accelerators

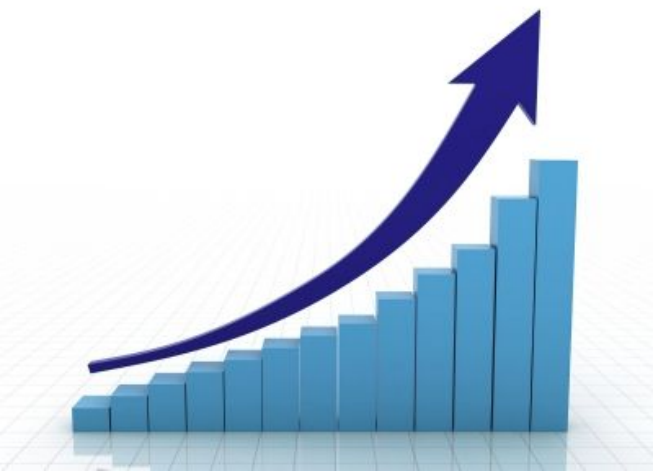
Example programs with untamed environments

OS services

Software frameworks for accelerators

customized applications

Programs with untamed environments are growing



Selective Symbolic Execution (SSE)

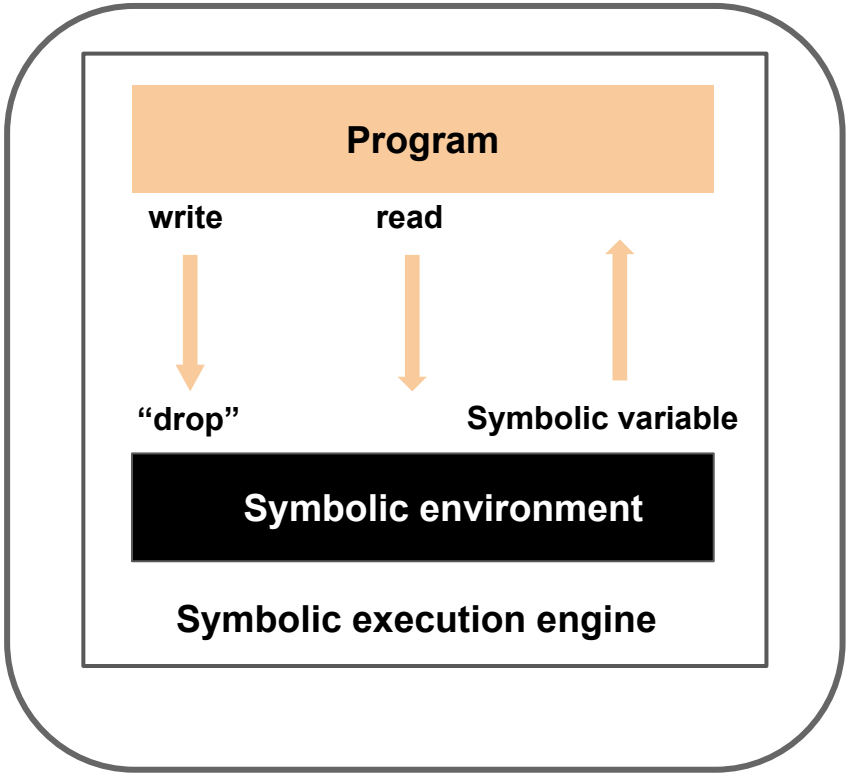
"Selective symbolic execution is a way to specify which parts of this big "program" should run concretely and which ones should run symbolically."

"Selective symbolic execution makes symbolic execution practical for large software that runs in real environments."

- **Selective symbolic execution [HotDep'09]**

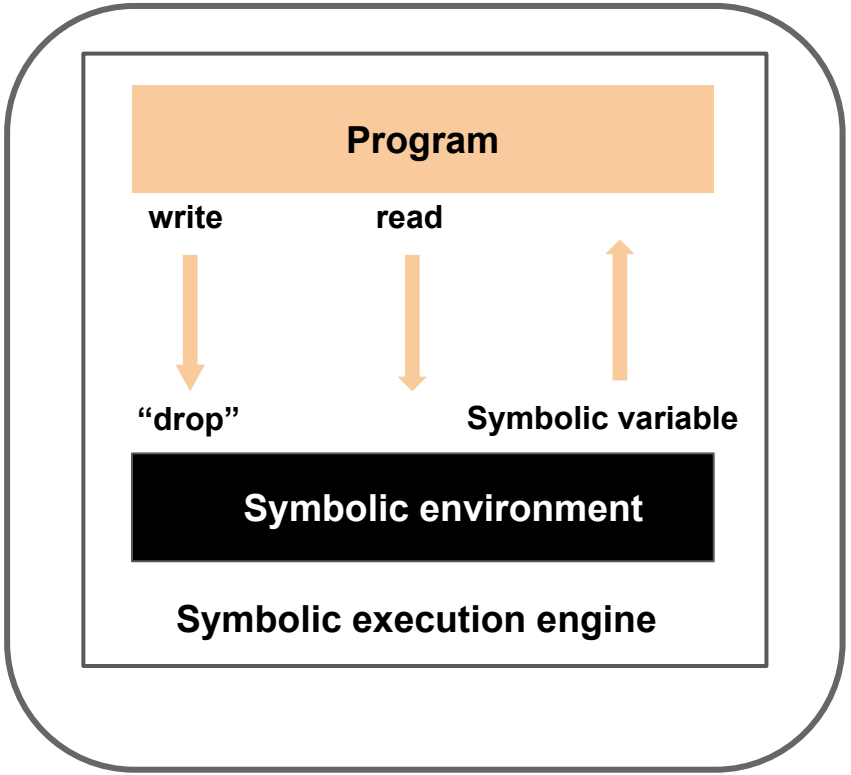
Existing approaches for analyzing programs with untamed environments using SSE

Existing approach - symbolic environment



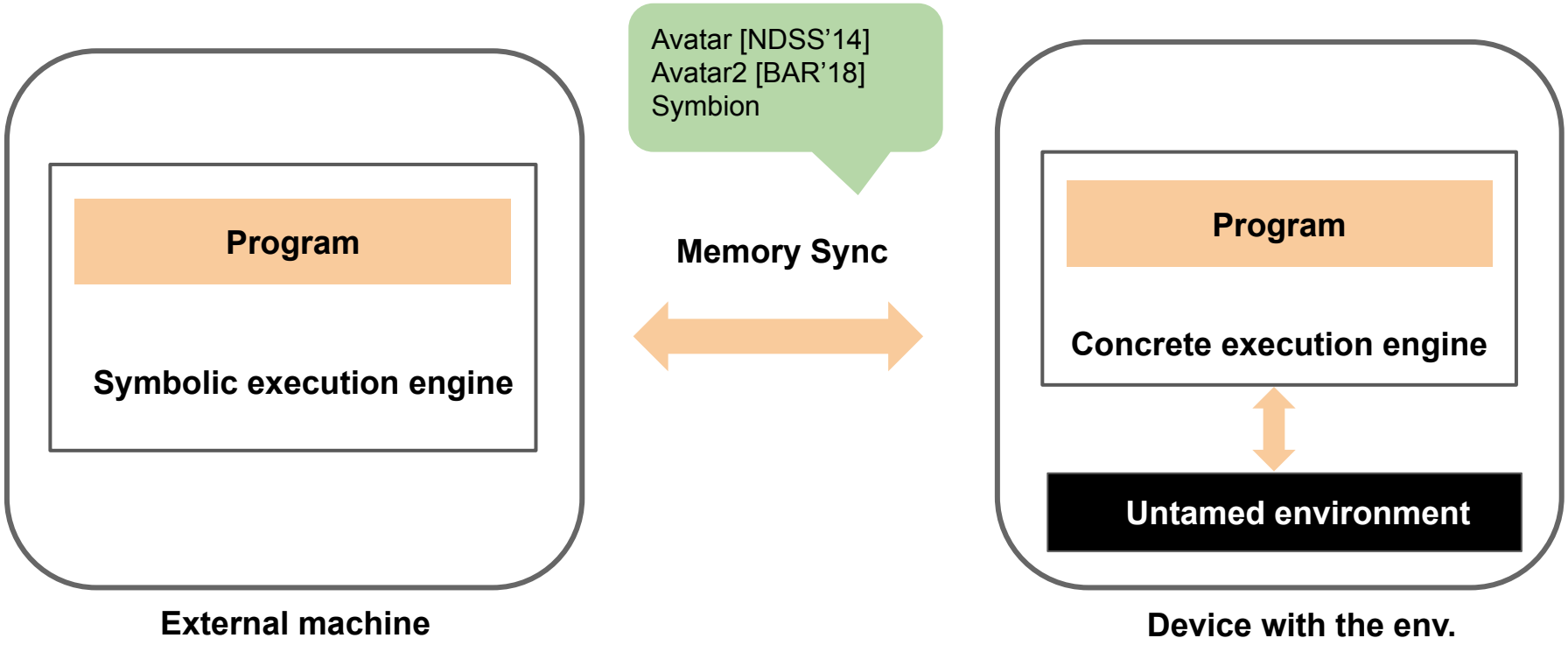
RevNIC [EuroSys'10]
DDT [USENIX ATC'10]
SymDrive [OSDI'12]

Existing approach - symbolic environment

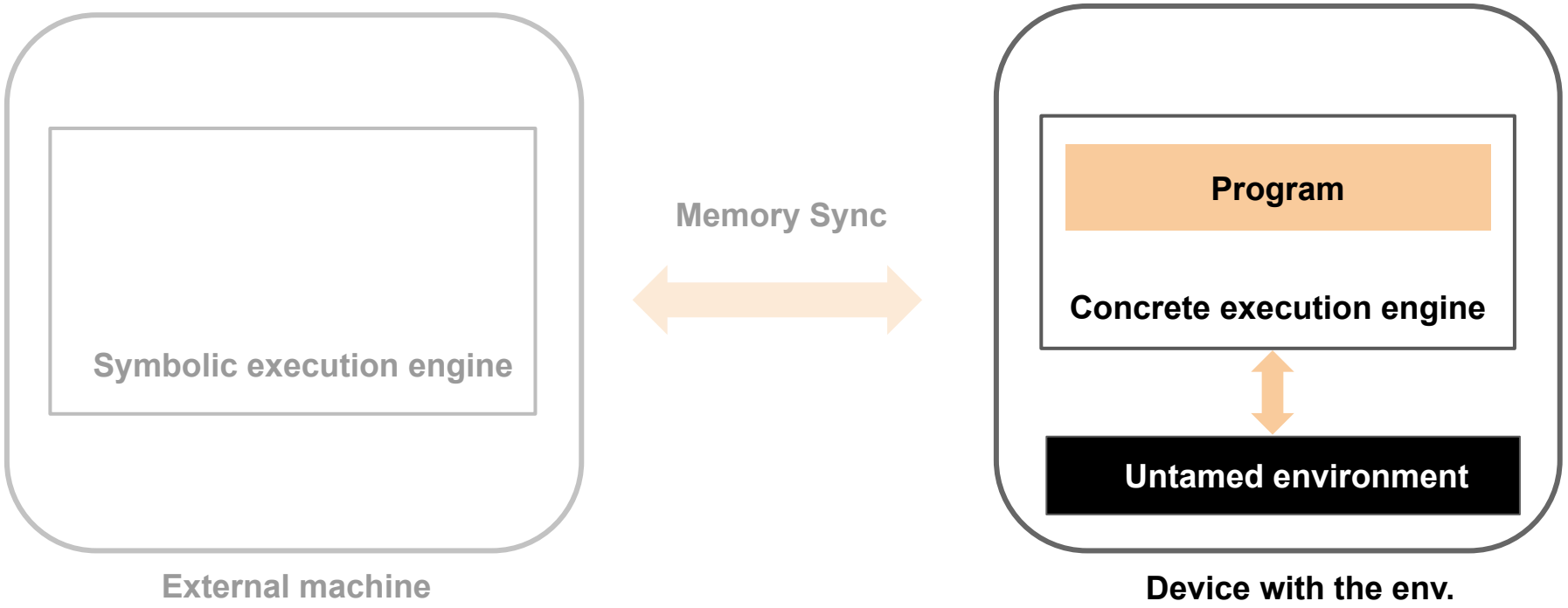


- False positives
- Path explosion
- Fail to initialize

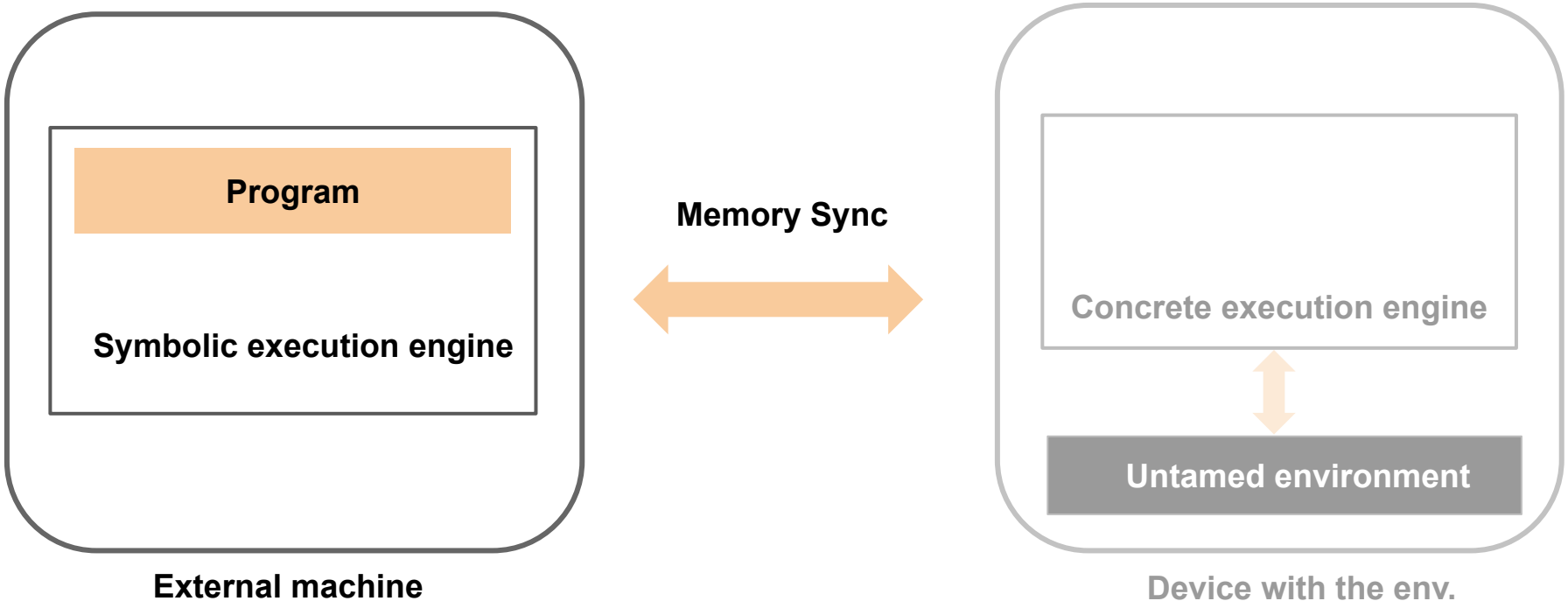
Existing approach - decoupled execution



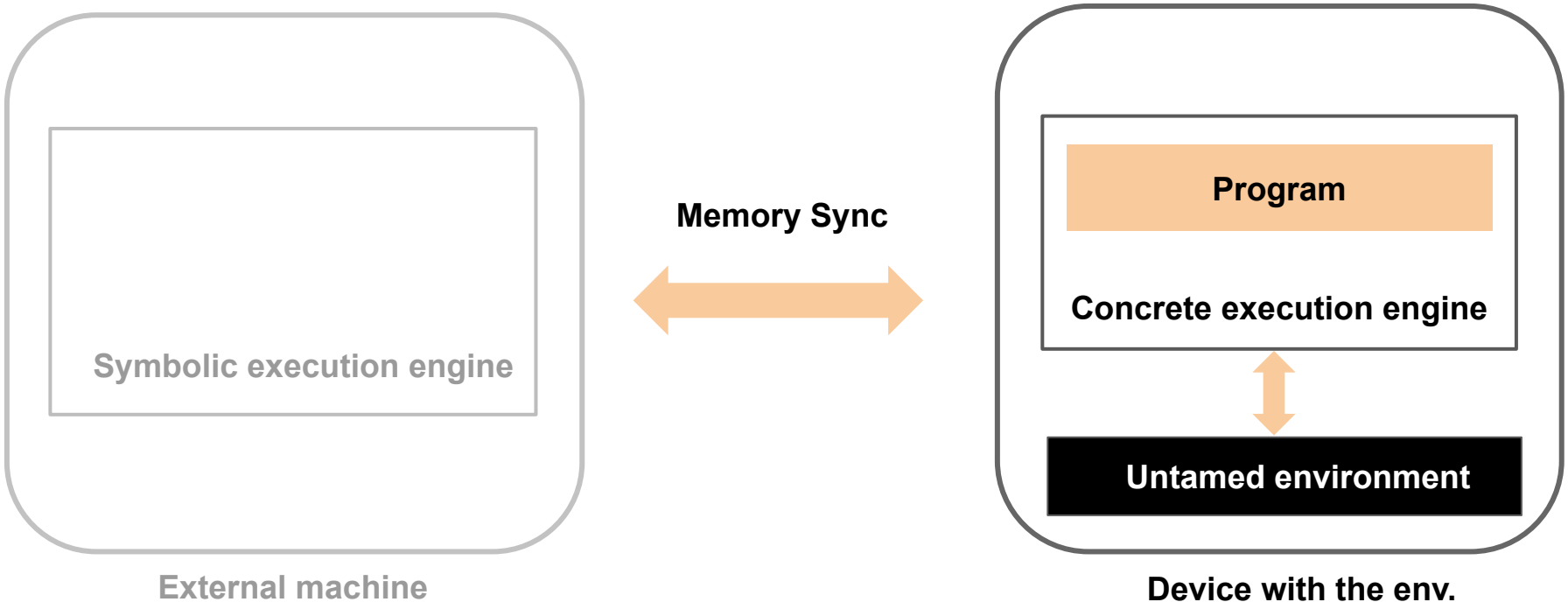
Existing approach - decoupled execution



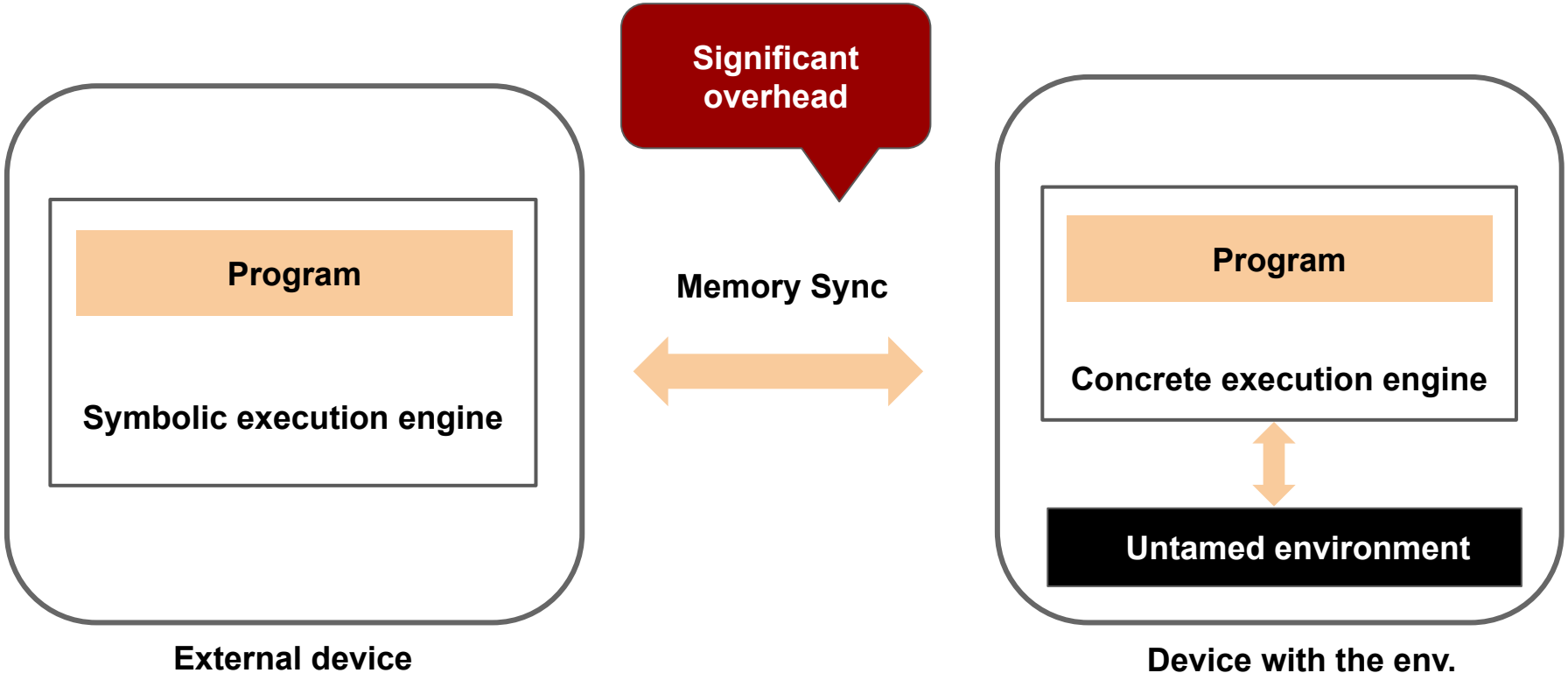
Existing approach - decoupled execution



Existing approach - decoupled execution



Existing approach - decoupled execution



Mousse
is tailored for programs with untamed environments achieving three important goals

Mousse’s goals

Real environments

High performance

Ease of use

Mousse’s solutions

Process-level SSE

Environment-aware concurrency

Distributed execution

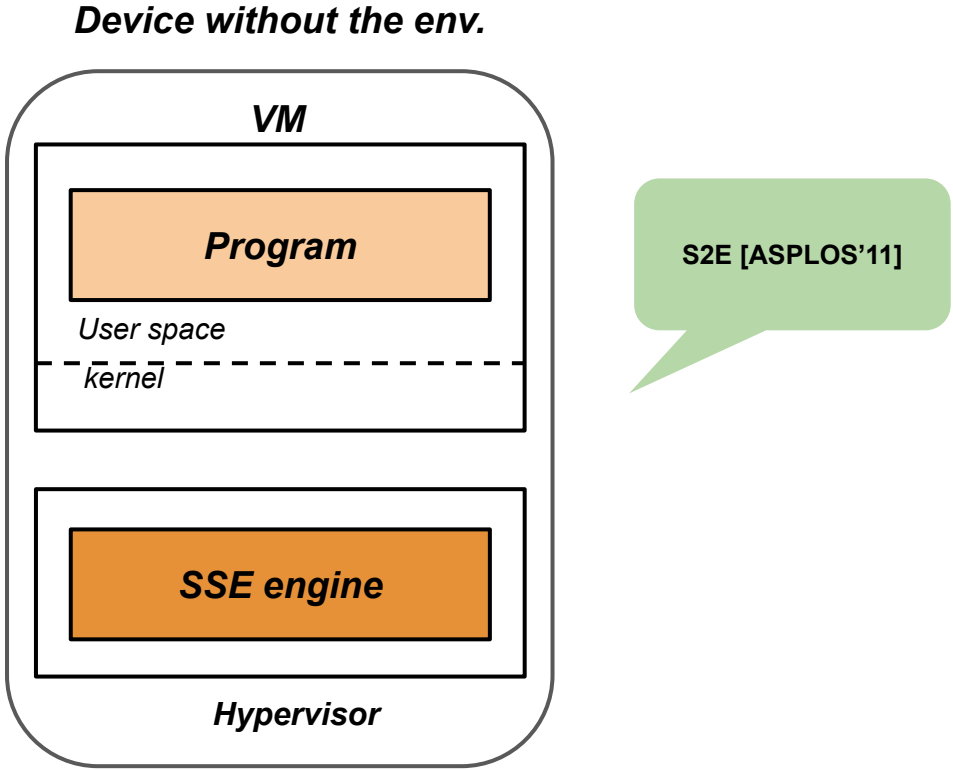
Mousse’s solutions

Process-level SSE

Environment-aware concurrency

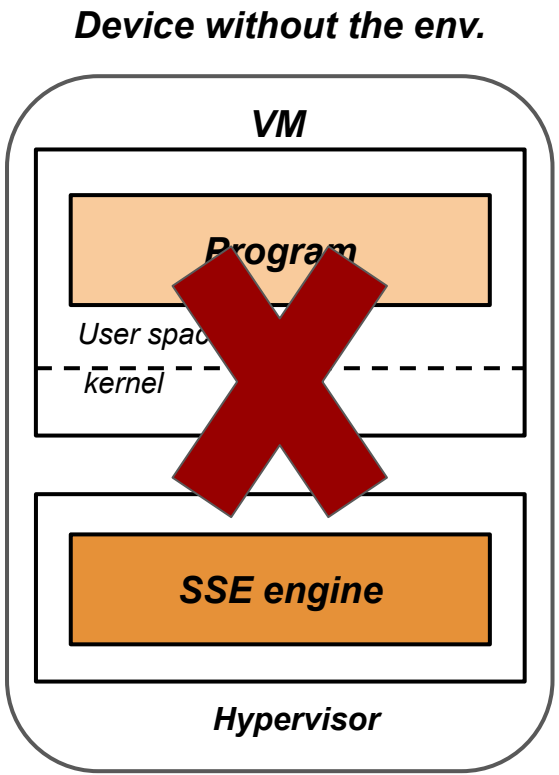
Distributed execution

VM-level SSE



VM-level SSE

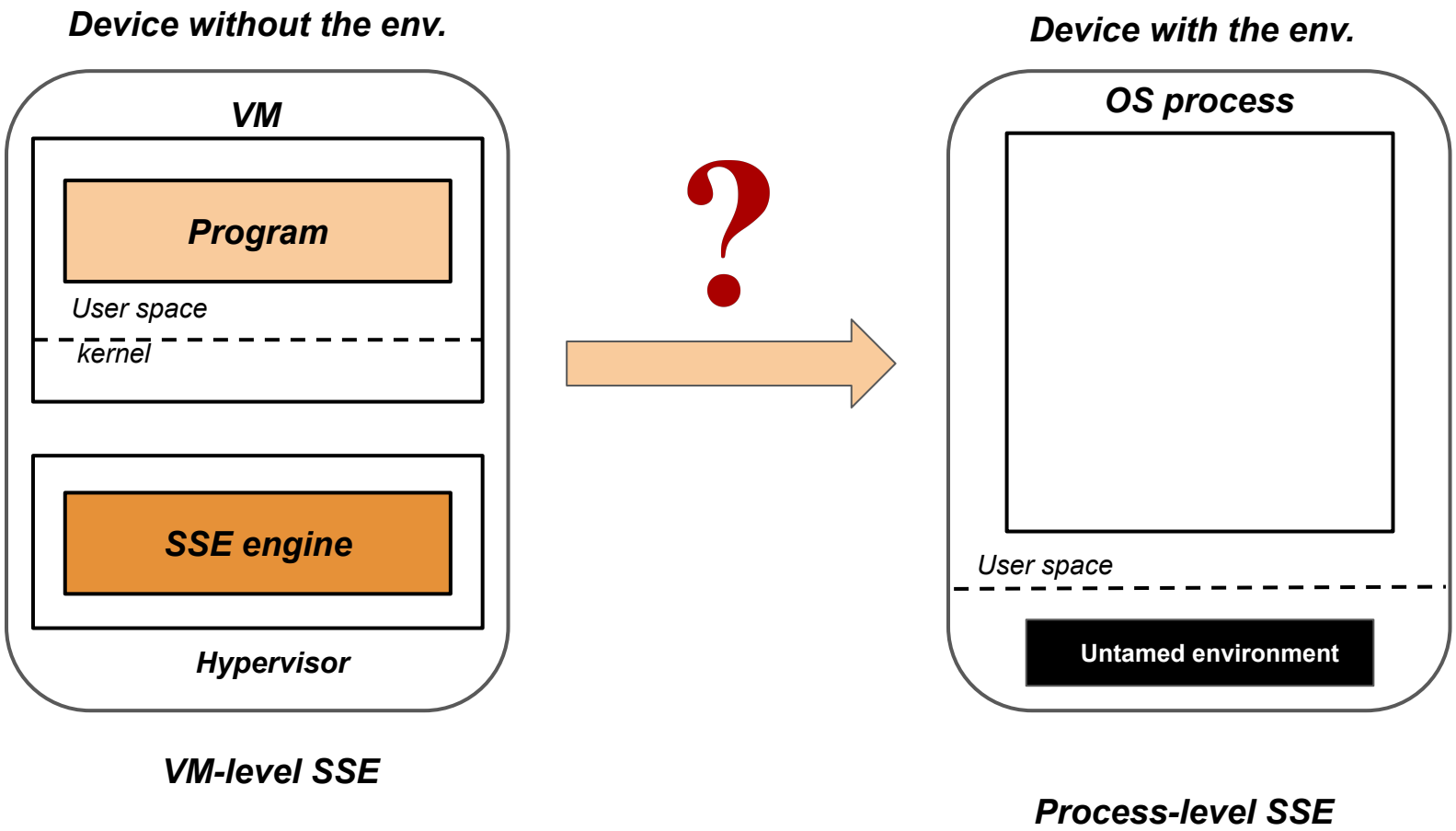
VM-level SSE



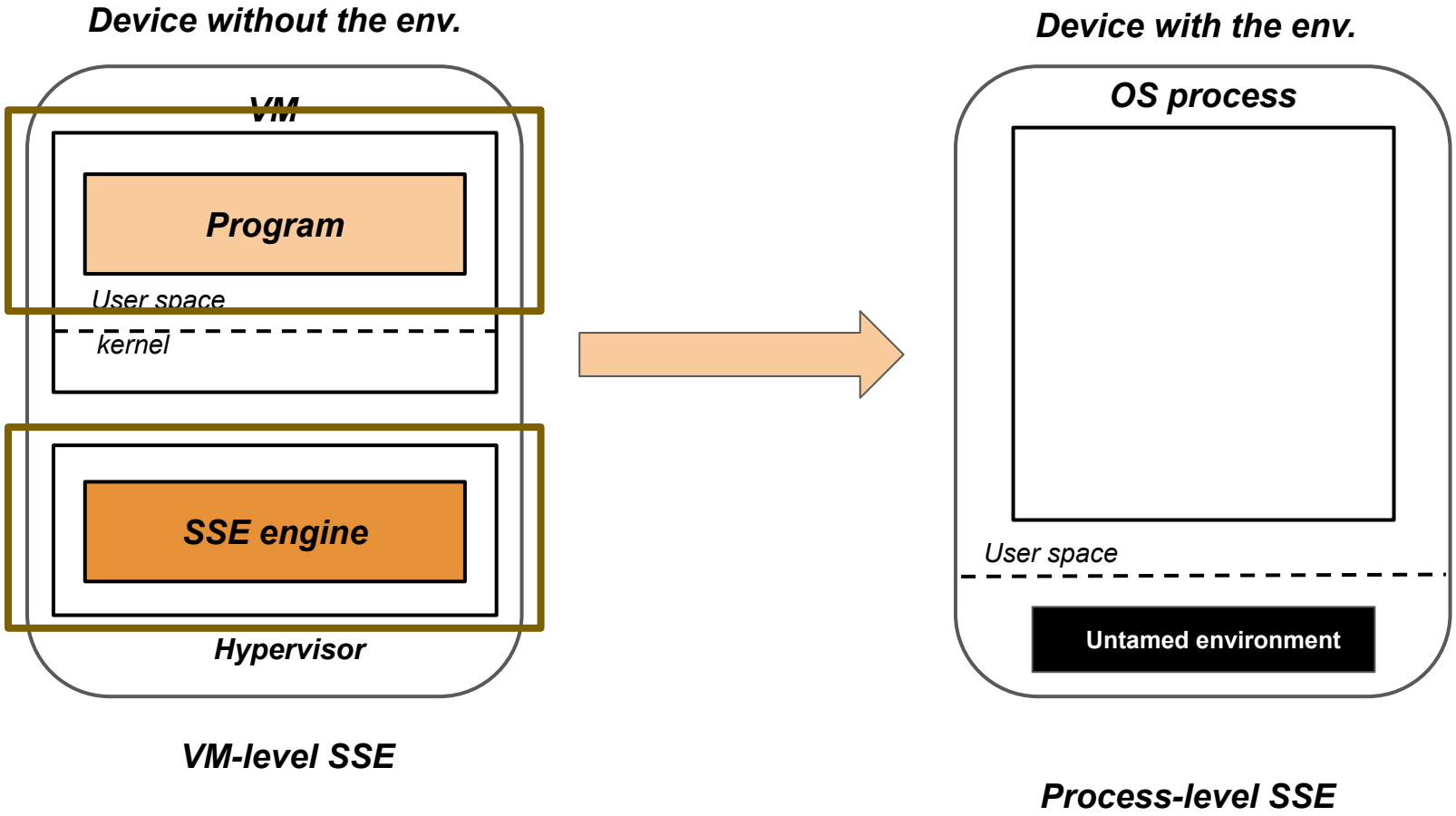
It does not work for programs with untamed environments because it requires virtualization.

VM-level SSE

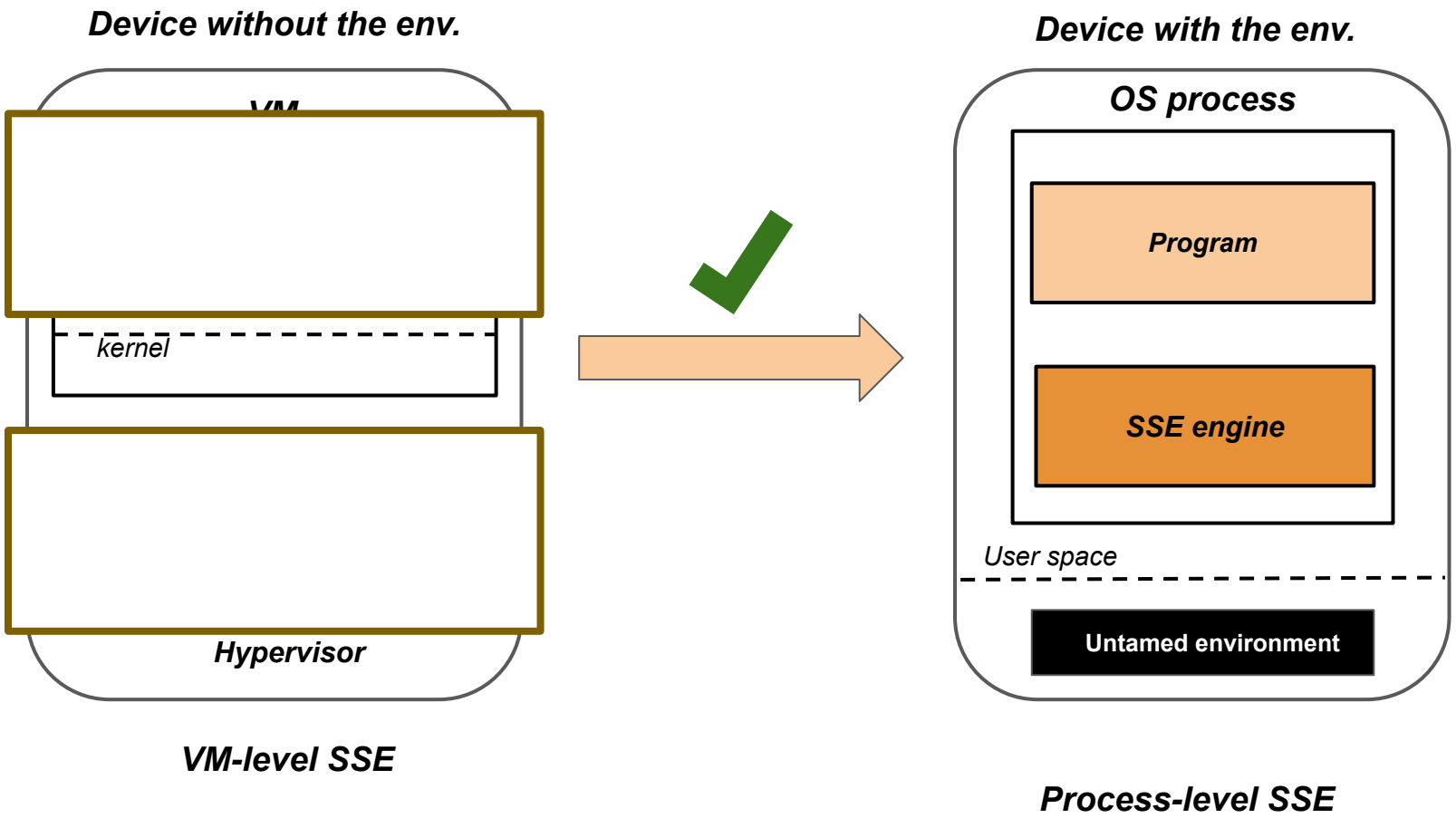
Process-level SSE: key idea



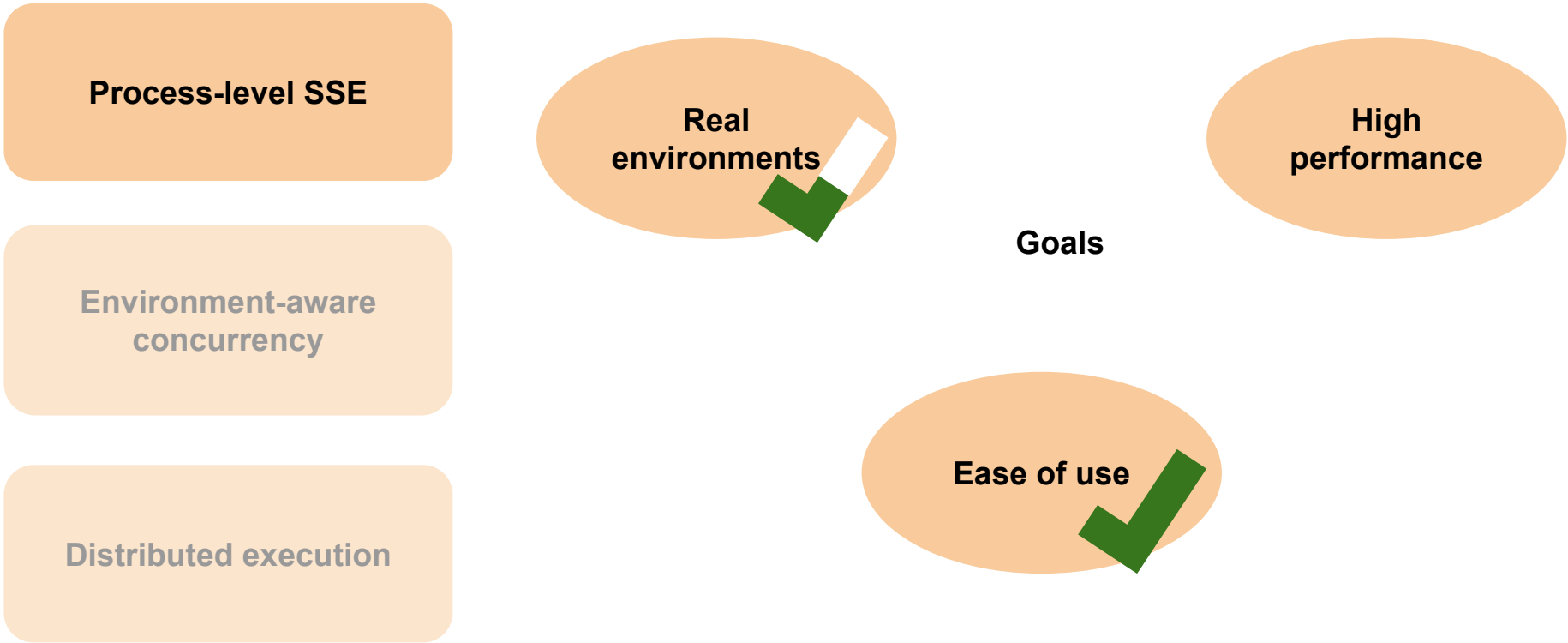
Process-level SSE: key idea



Process-level SSE: key idea



Process-level SSE: benefits



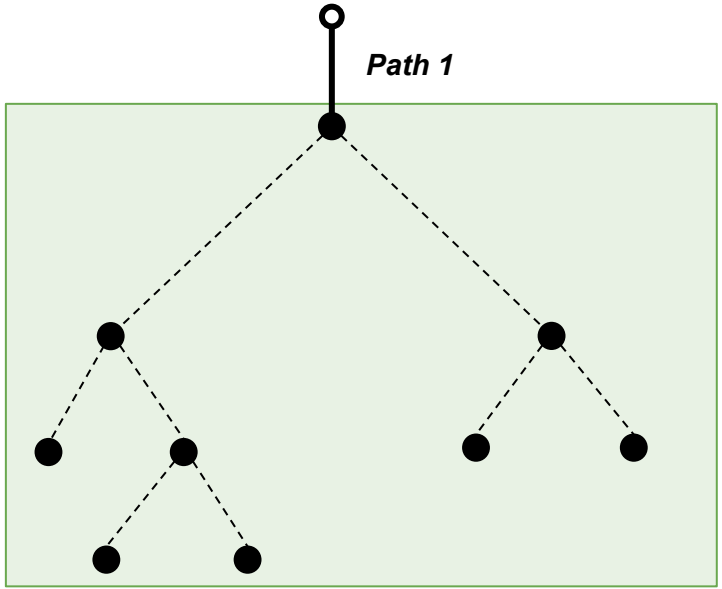
Mousse’s solutions

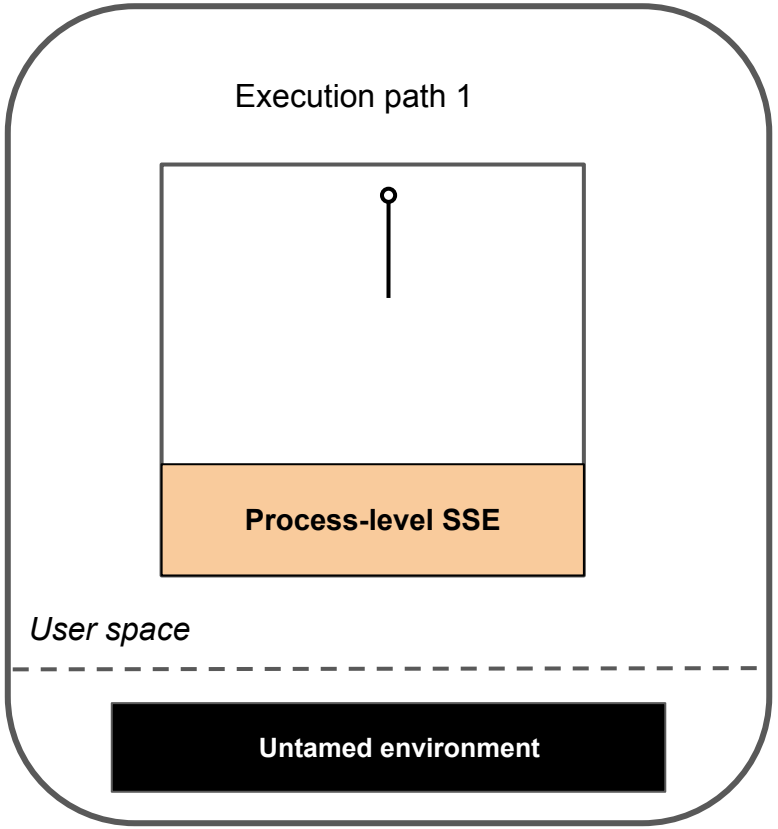
Process-level SSE

Environment-aware concurrency

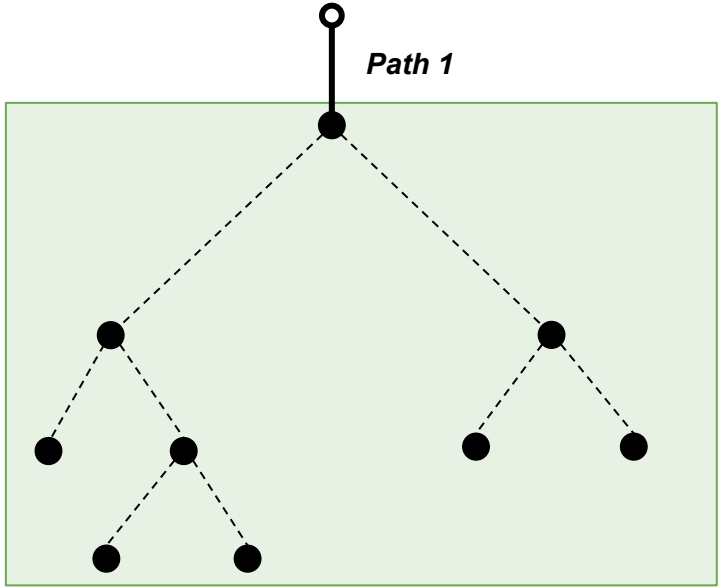
Distributed execution

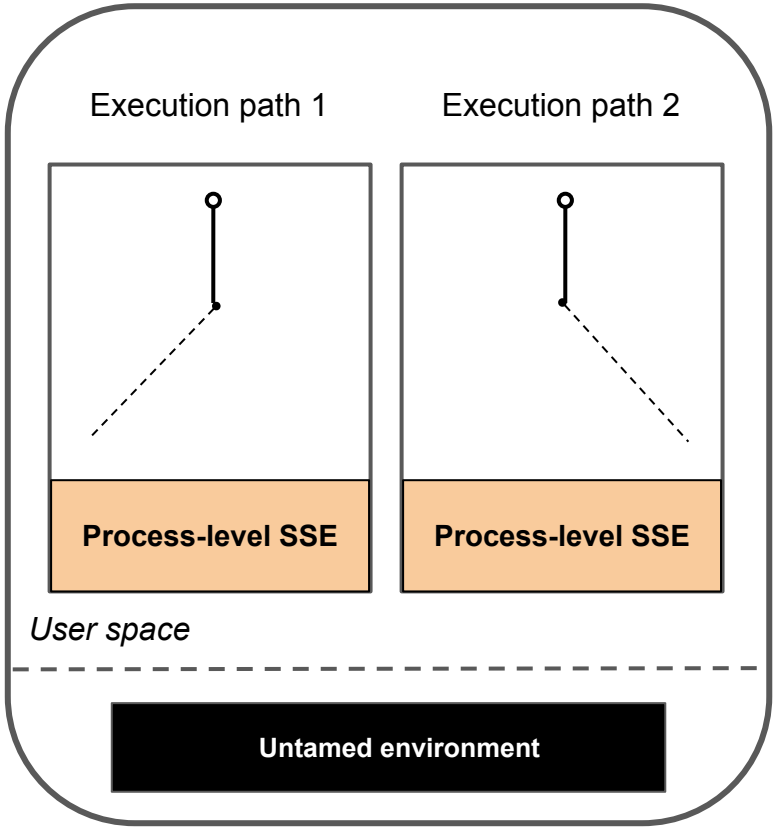
Environment-aware concurrency: key idea



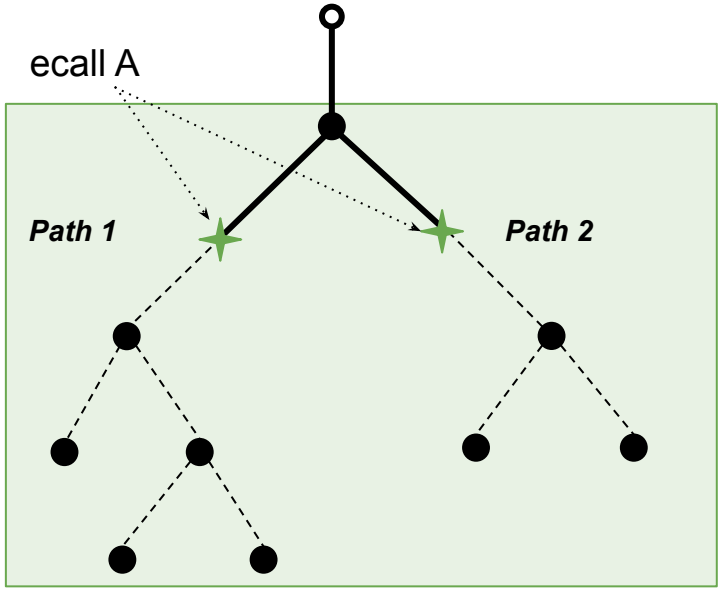


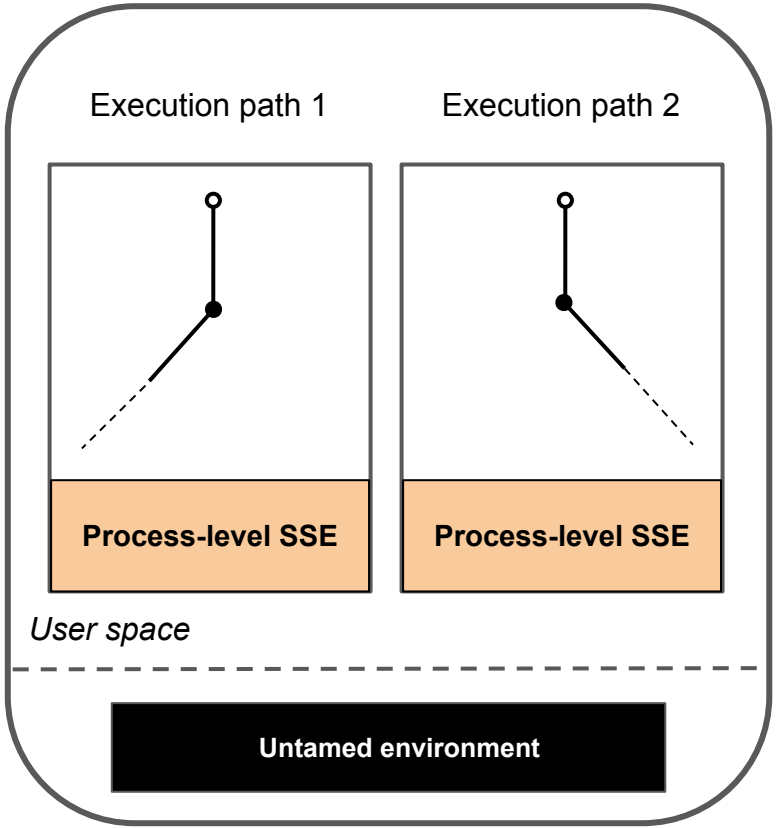
Environment-aware concurrency: key idea



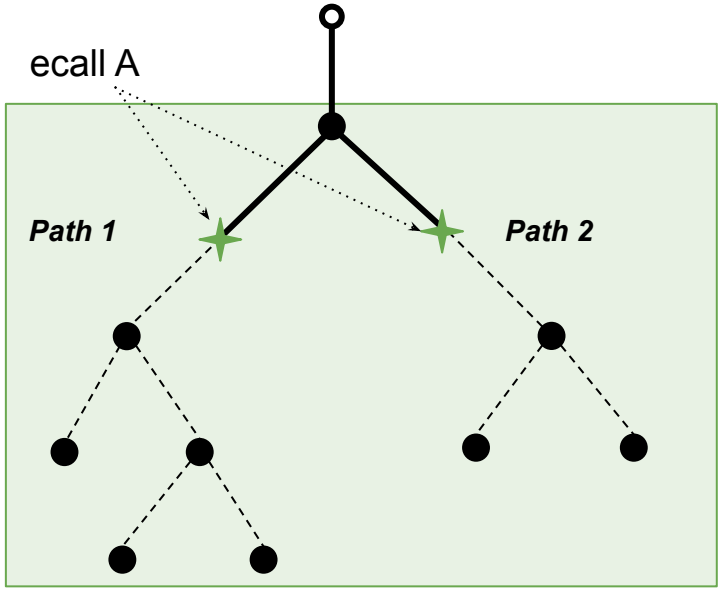


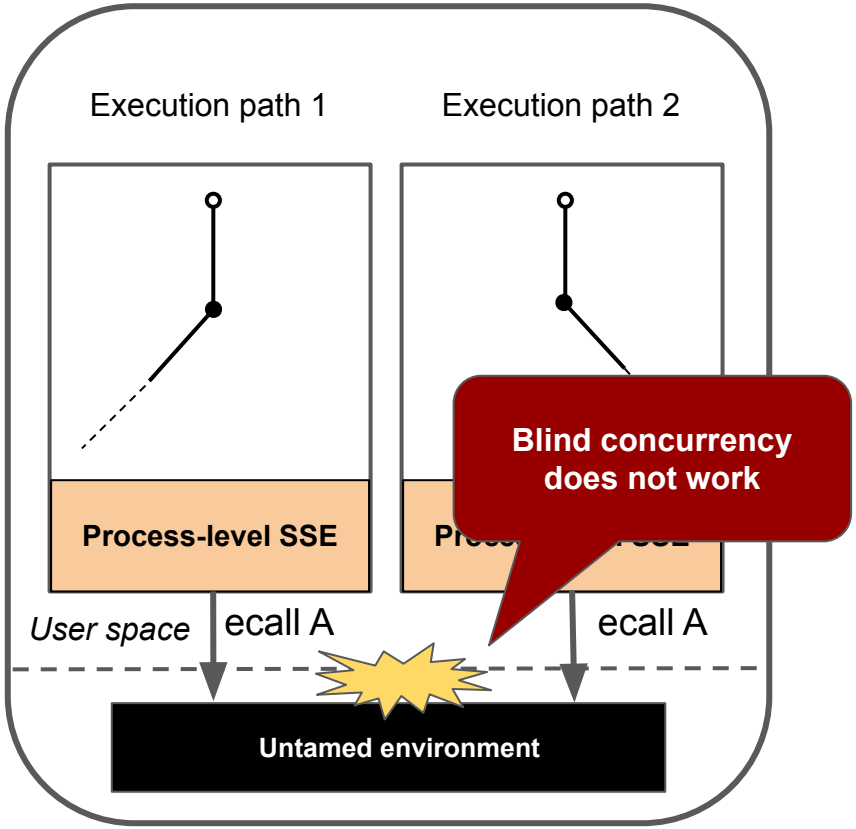
Environment-aware concurrency: key idea



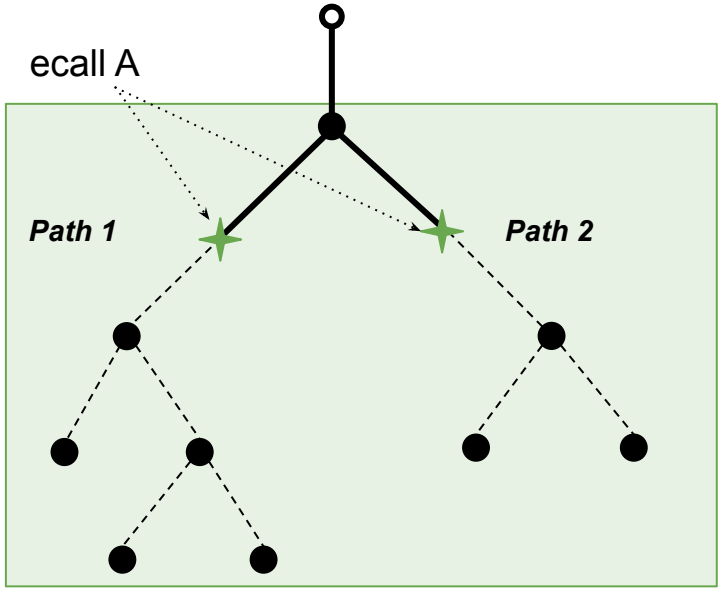


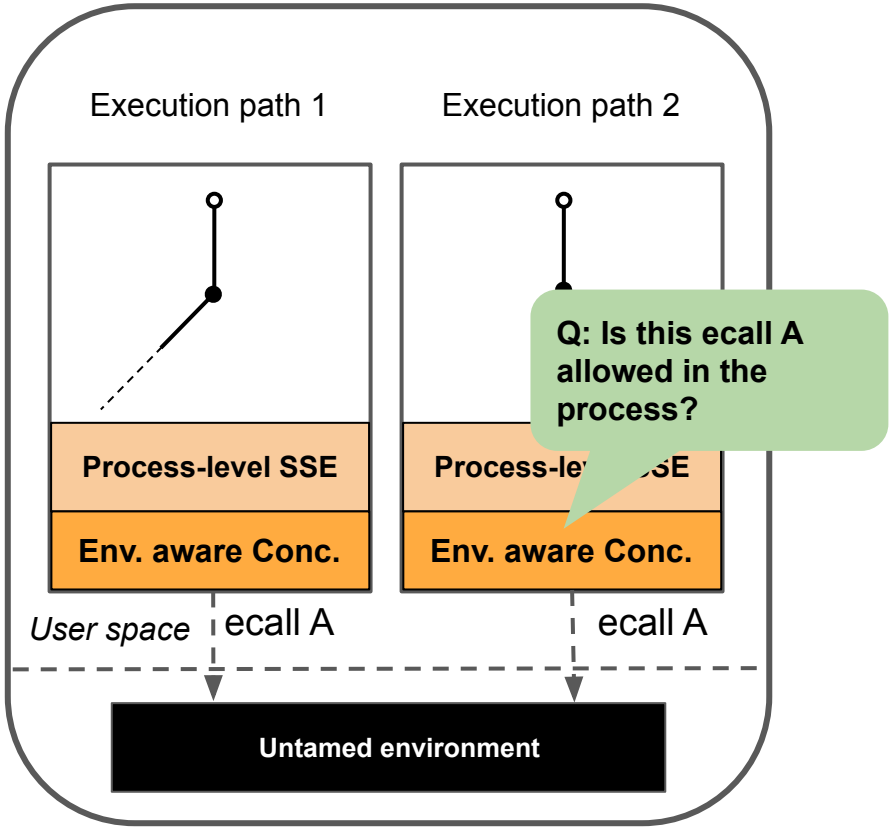
Environment-aware concurrency: key idea



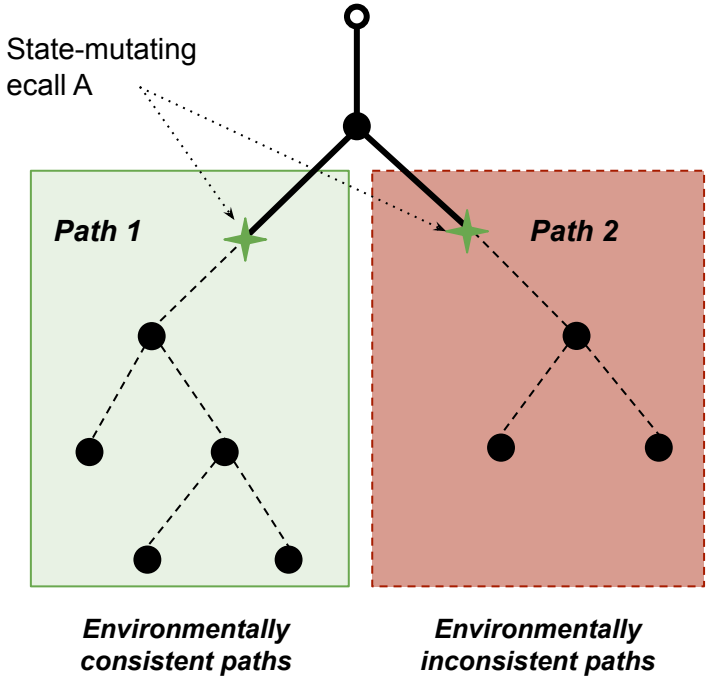


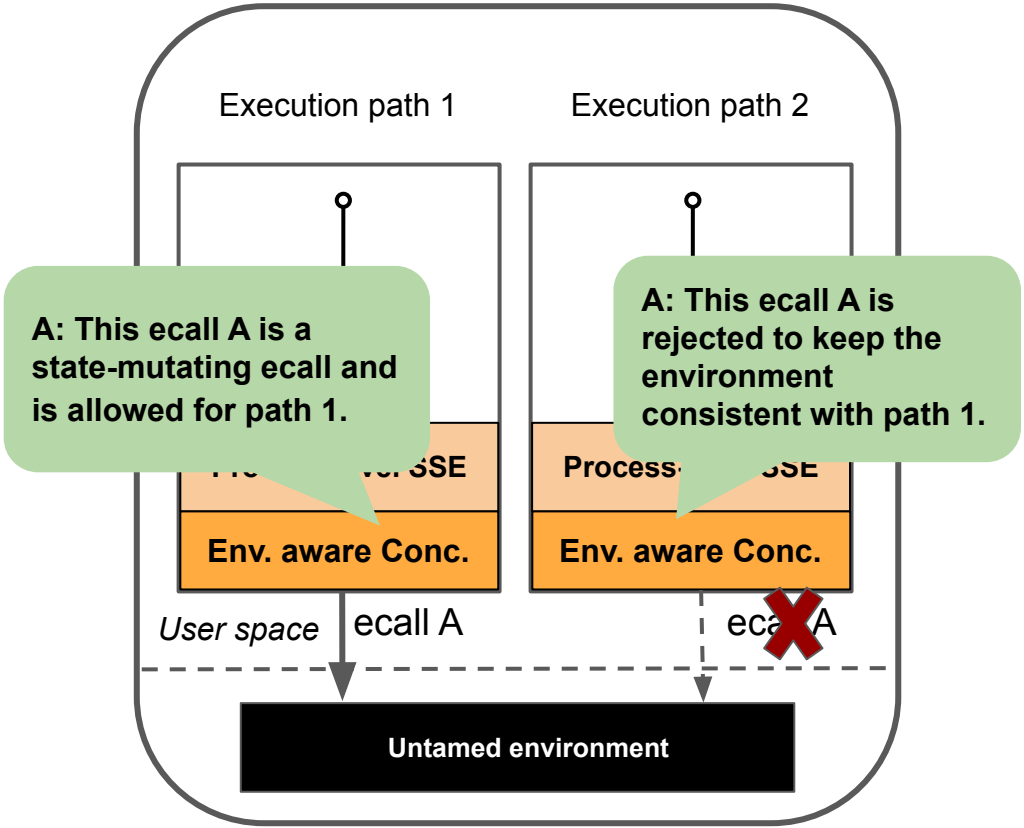
Environment-aware concurrency: key idea



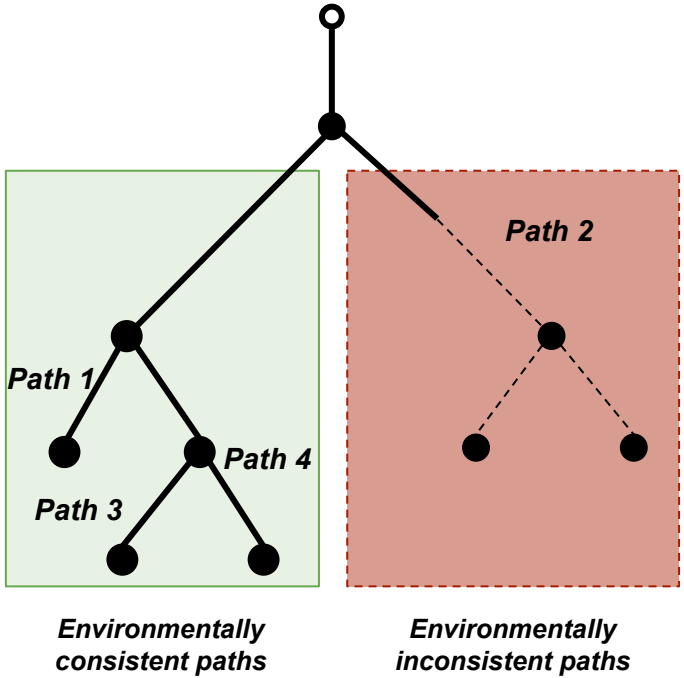


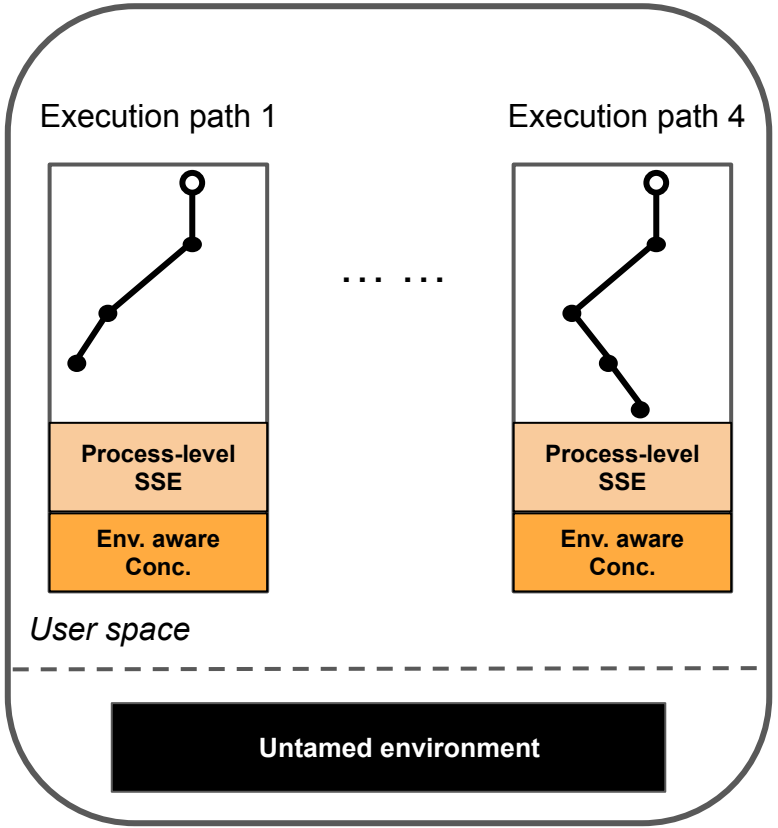
Environment-aware concurrency: key idea





Environment-aware concurrency: key idea



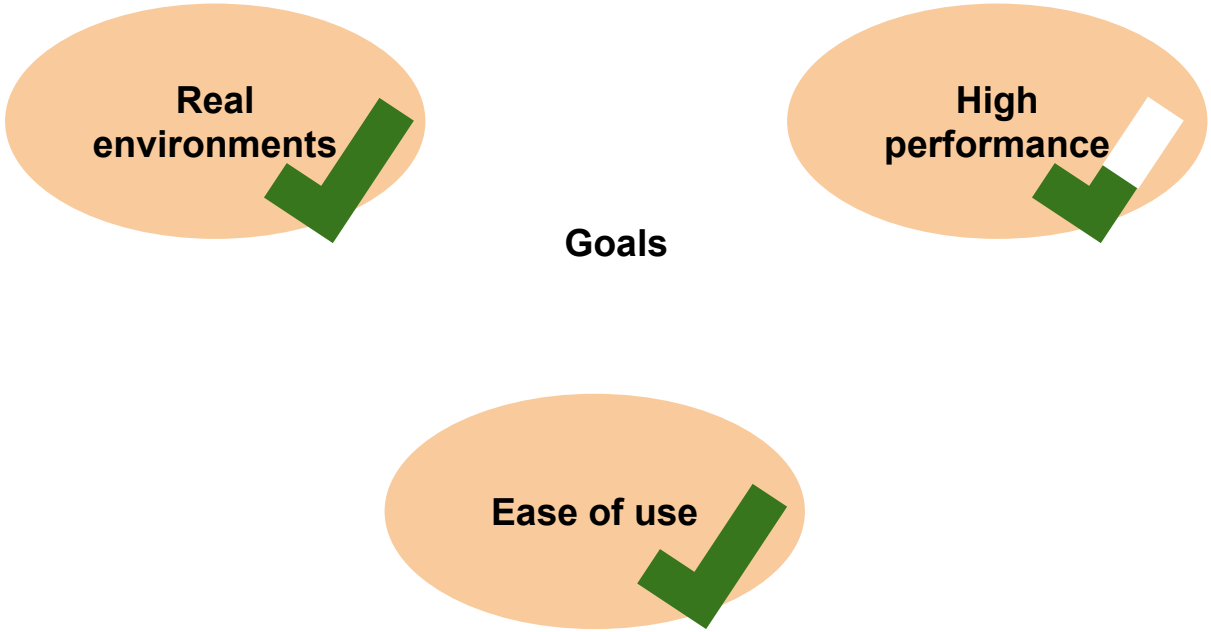


Environment-aware concurrency: benefits

Process-level SSE

Environment-aware concurrency

Distributed execution



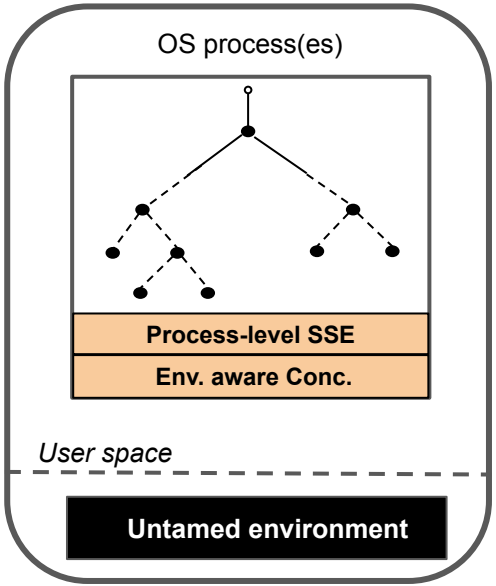
Mousse’s solutions

Process-level SSE

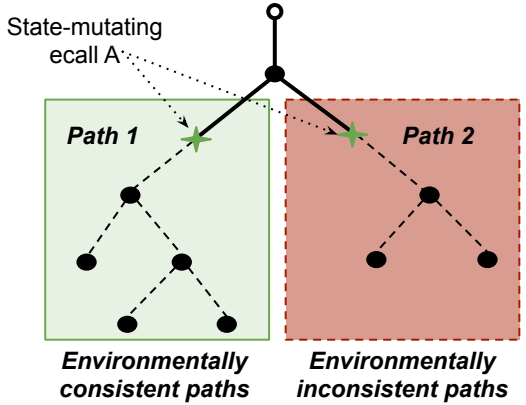
Environment-aware concurrency

Distributed execution

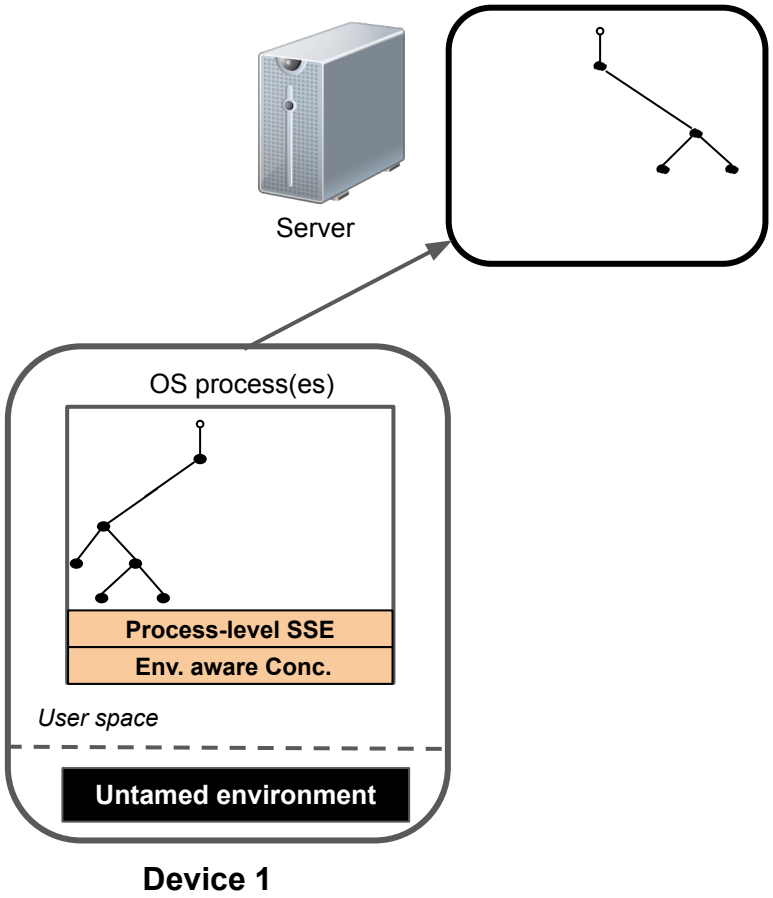
Distributed execution: key idea

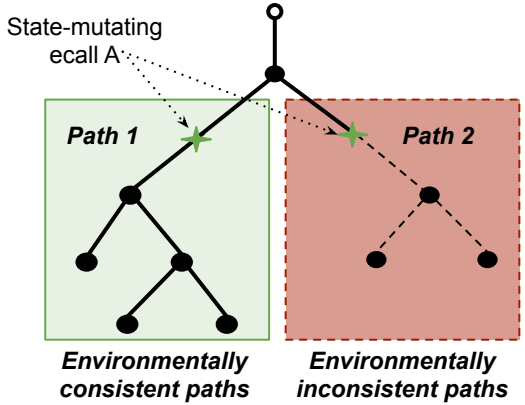


Device 1

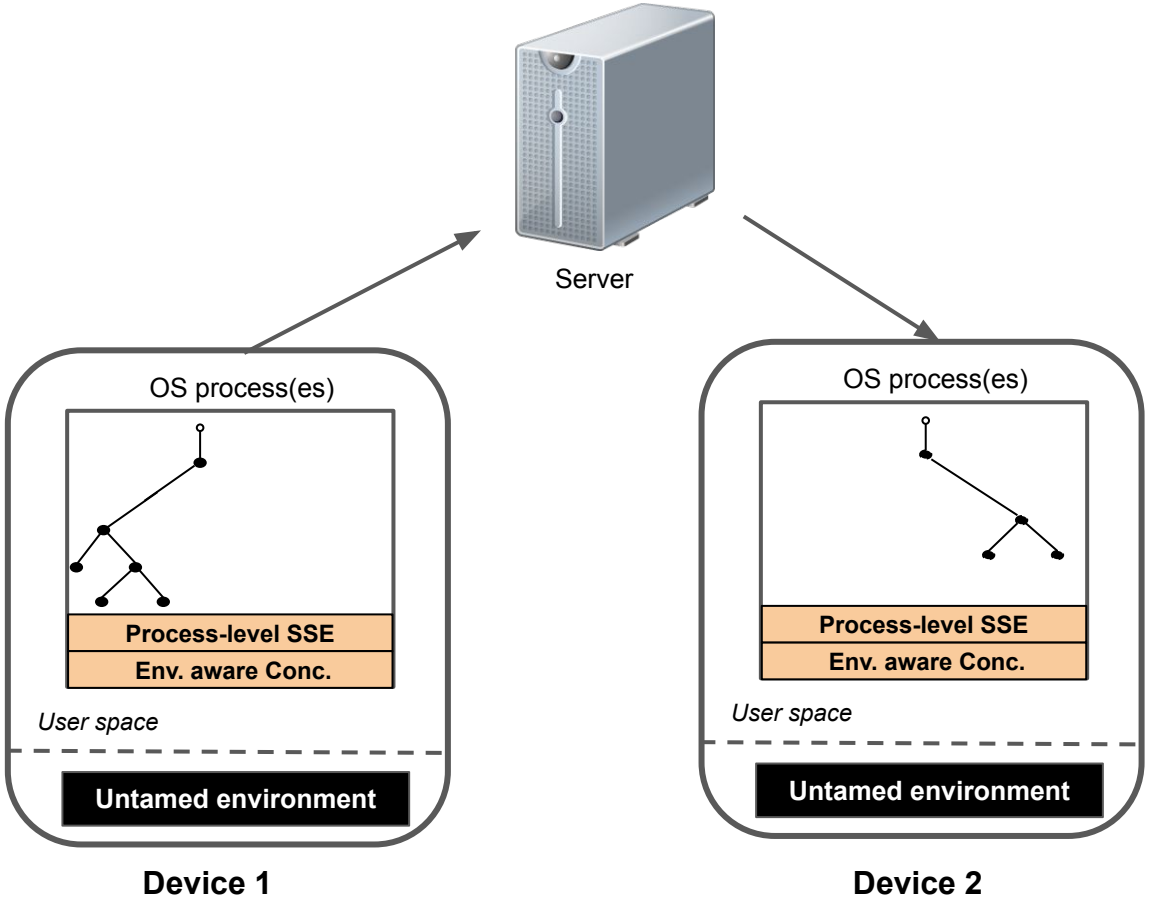


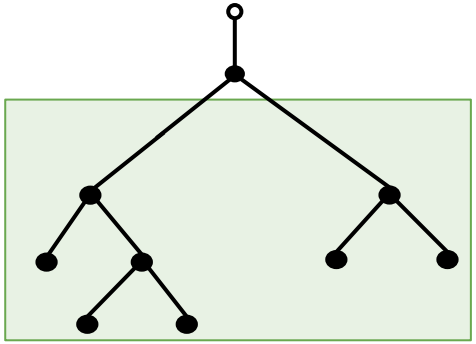
Distributed execution: key idea





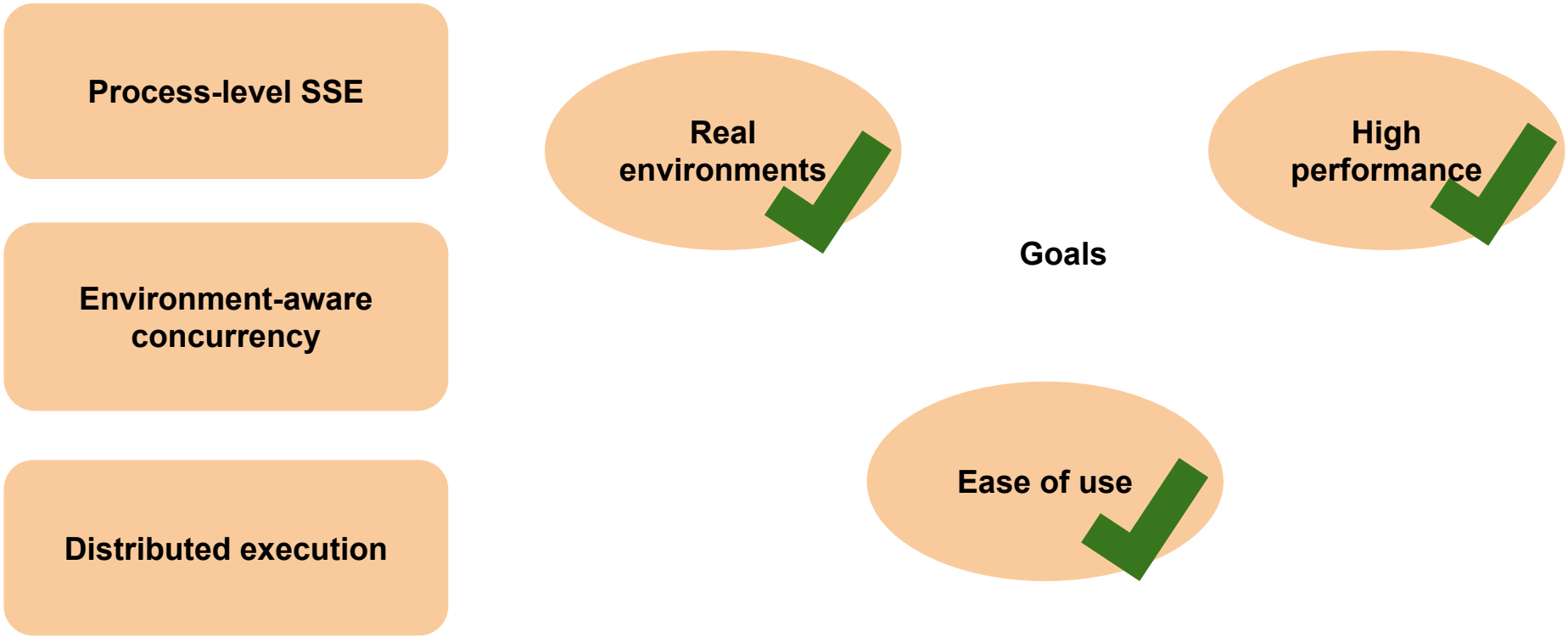
Distributed execution: key idea





Environmentally consistent paths

Distributed execution: benefits



Mousse’s goals

Real environments

High performance

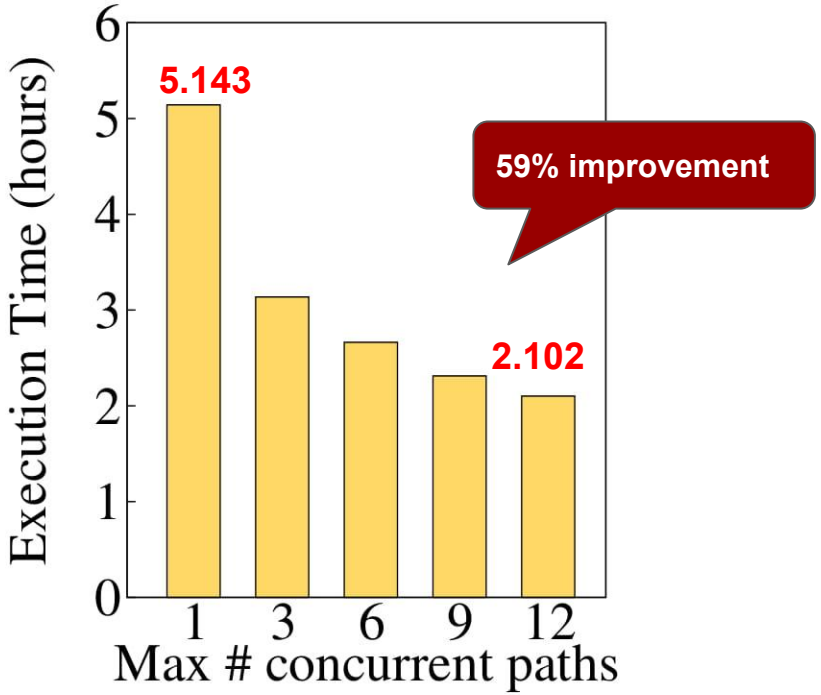
Ease of use

Evaluation

Evaluated Mousse on five Android OS services

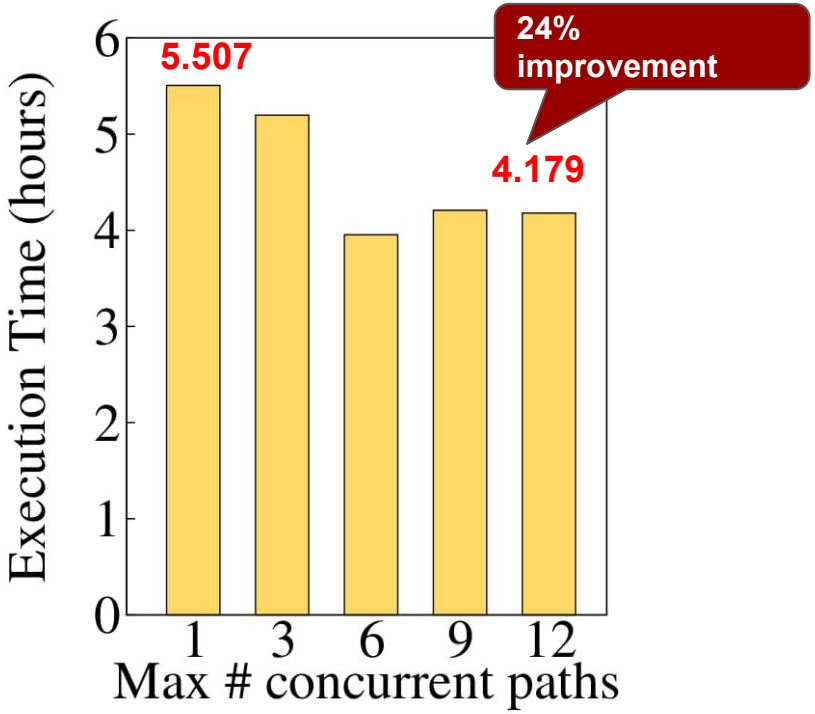
- AudioServer and AudioProvider services in Pixel 3
- CameraService and CameraDaemon services in Nexus 5X
- OpenGLES graphics libraries in Nexus 5

Env. aware conc. improves execution time



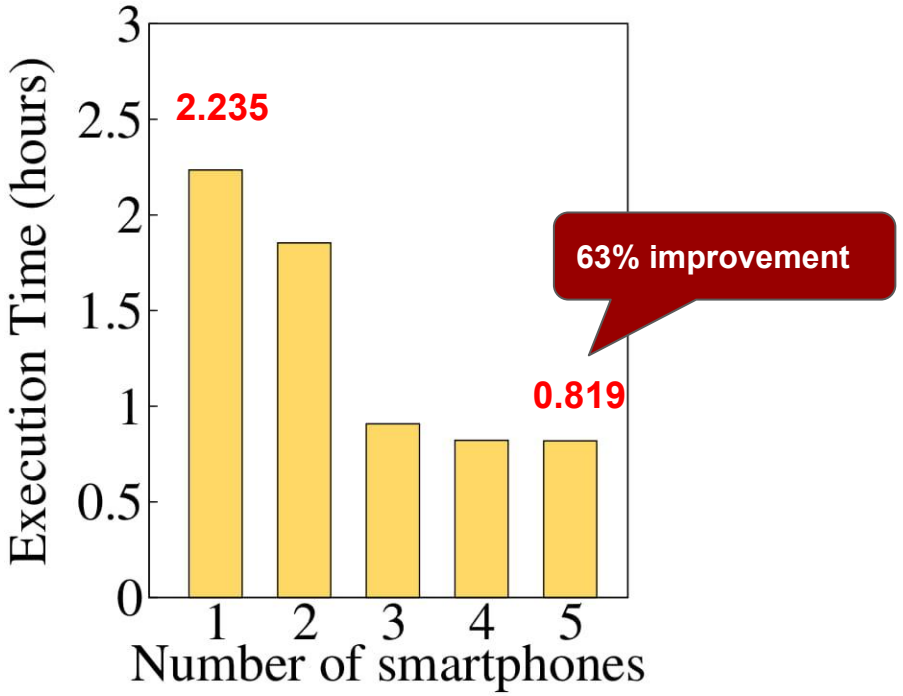
AudioProvider API: adev_set_parameters
(no state-mutating ecalls)

Env. aware conc. improves execution time



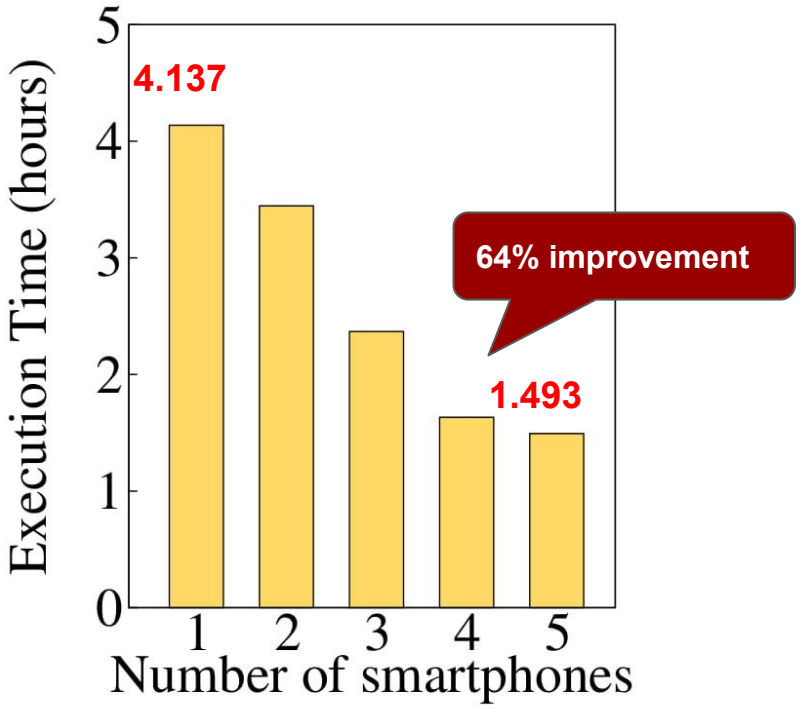
AudioProvider API: out_write
(issues state-mutating ecalls)

Distributed execution improves execution time



AudioProvider API: adev_set_parameters
(no state-mutating ecalls)

Distributed execution improves execution time

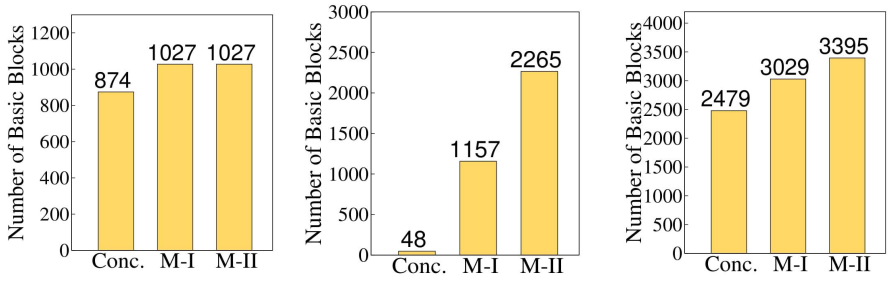


AudioProvider API: out_write
(issues state-mutating ecalls)

Env. aware conc. & distributed execution improve execution time

API name	**Execution time** (no concurrency, no distributed execution)	**Execution time** (max concurrent paths as 9, 5 smartphones)	**Improvement**
adev_set_parameters	5.143 hrs	0.819 hrs	**84%**
out_write	5.507 hrs	1.493 hrs	**73%**

More evaluation results



Coverage evaluation

- Bugs and vulnerabilities
  - Two null-pointer dereferences
  - Two double-free vulnerabilities
- Taint analysis
- Performance profiling

Analysis results

Service name	API name	Execution time (minutes)	# of path	# of off-loads due to Res.	# of off-loads due to Env.
GS	eglCreateWindowSurface	115.9	11	1	9
	eglQuerySurface	118.8	88	40	21
	eglGetDisplay	8.7	1	0	0
	glCreateShader	34.2	5	0	3
	glShaderSource	1605.8	371	148	95
	glViewport	14.6	6	5	0
AP	adev_open_output_stream	390.1	612	264	0
	adev_open_input_stream	170.1	566	234	0
	adev_open	2.2	12	0	0
	adev_set_parameters	107.7	237	122	0
	adev_set_mode	2.8	3	0	0
	adev_set_voice_volume	2.7	1	0	0
	adev_set_mic_mute	3.4	1	0	0
	out_write	89.6	50	24	10
	out_set_parameters	25.9	136	34	0
	out_drain	5.8	2	0	0
CS	getNumberOfCameras	47.6	46	28	3
	connectDevice	29.0	19	2	5
	getCameraCharacteristics	28.9	45	18	0
	supportsCameraApi	4.1	2	0	0
	submitRequestList	20.7	18	2	7
	cancelRequest	4.1	1	0	0
	endConfigure	4.2	1	0	0
	createStream	93.6	87	33	7
	createDefaultRequest	4.9	1	0	0

Table 1. *Single-API testing of OS services with Mousse. Abbreviations used in the table: GS = GPU Stack, AP = AudioProvider, CS = CameraServer, Res. = Resource constraint, Env. = Environment consistency.*

Execution time of more APIs

Summary

- We introduced Mousse, a system for analyzing programs with untamed environments using SSE.
- Mousse outperforms alternative solutions in terms of performance and code coverage.
- Mousse opens the opportunity to perform various analyses on programs with untamed environments.

Mousse is open sourced: https://trusslab.github.io/mousse/

Back up slides

Example to show blind concurrency does not work

```
/* Audio service out_write API */

1 static ssize_t out_write(struct audio_stream_out *stream, const void *buffer, size_t bytes) {
2       struct stream_out *out = (struct stream_out *)stream;
                ...
3       lock_output_stream(out); //This function calls pthread_mutex_lock(&out->lock);
                ...
4       long ns = (frames * (int64_t) NANOS_PER_SECOND) / out->config.rate;
5       request_out_focus(out, ns);
                ...
6       ret = pcm_write(out->pcm, (void *)buffer, bytes_to_write);
                ...
7       pthread_mutex_unlock(&out->lock);
                ...
8 }
```

Example to show blind concurrency does not work

```
/* Code in the audio driver where the error happens */

1 void *q6asm_is_cpu_buf_avail(int dir, struct audio_client *ac, uint32_t *size, uint32_t *index)
2 {
3       void *data;
4       unsigned char idx;
5       struct audio_port_data *port;
                ...
6       // dir 0: used = 0 means buf in use
7       // dir 1: used = 1 means buf in use
8       if (port->buf[idx].used == dir) {
9               // To make it more robust, we could loop and get the
10              // next avail buf, its risky though
11              pr_err("%s: Next buf idx[0x%x] not available, dir[%d]\n", __func__, idx, dir);
12              mutex_unlock(&port->lock);
13              return NULL;
14      }
                ...
15 }
```