Embedded System Programming

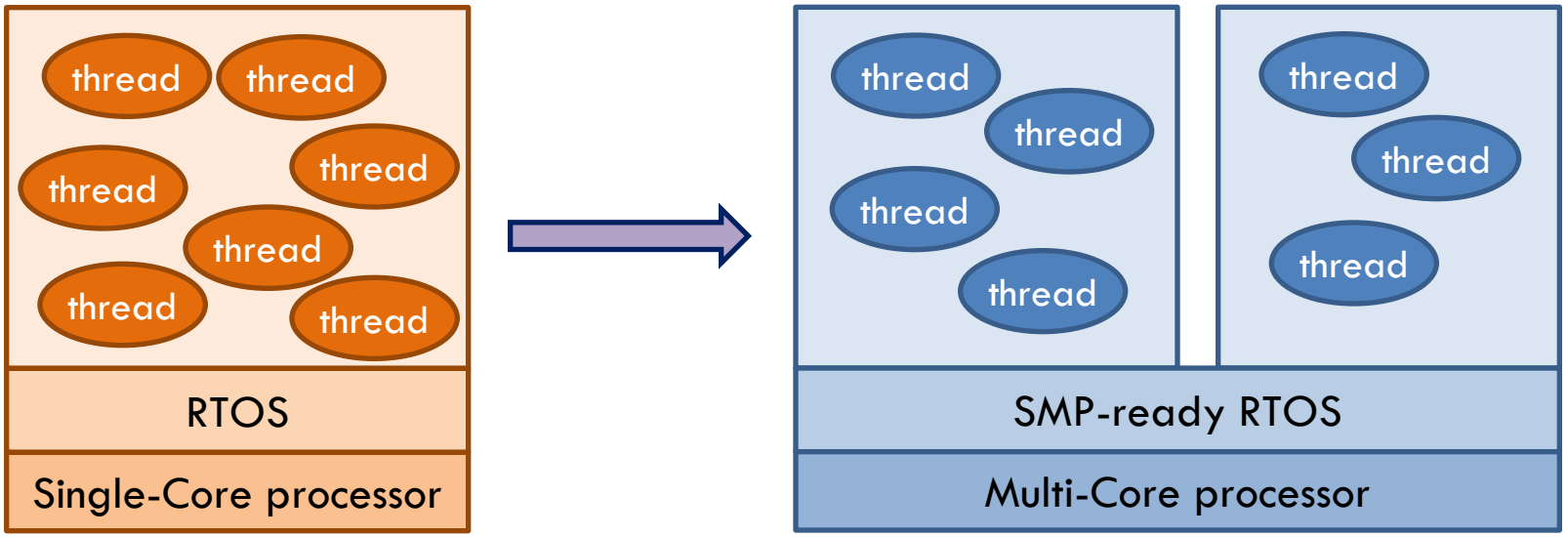
Multicore ES

(Module 40)

Yann-Hang Lee
Arizona State University
yhlee@asu.edu
(480) 727-7507

Summer 2014

The Era of Multi-core Processors



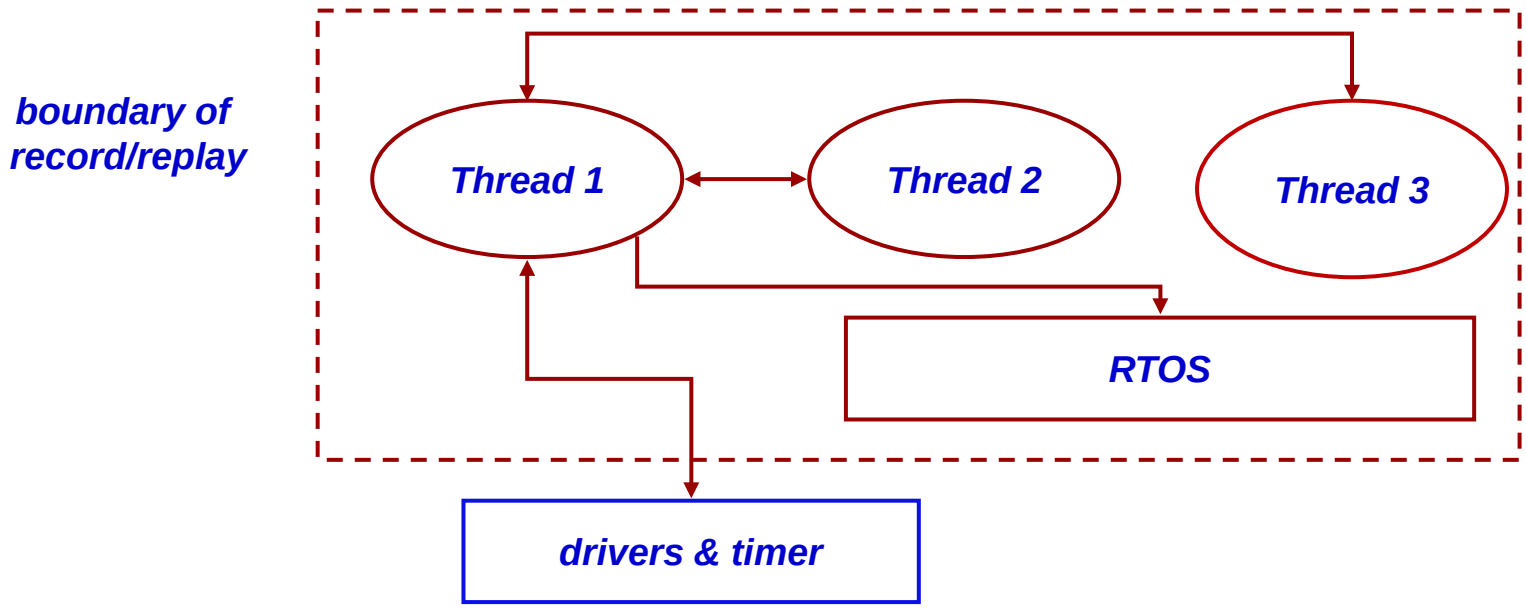
- **Will the application run correctly**
  - a benign data race may become a true race
  - scheduling anomaly
- **How can we debug and monitor ES on multicore processors**

Debugging Embedded Software

- **In a 2002 NIST survey, an average bug found in post-product release takes 15.3 hours to fix.**
  - Cost of software development and product liability
  - Testing process and software release time
- **Finding bugs in multithreaded programs is difficult**
  - The bug and symptom are widely separated in space and time
  - The system is nondeterministic
  - The occurrence of potential errors may only be triggered after a long period of execution.
- **Why is it challenging?**
  - Probe effect may alter program behavior
  - Logged data could be enormous

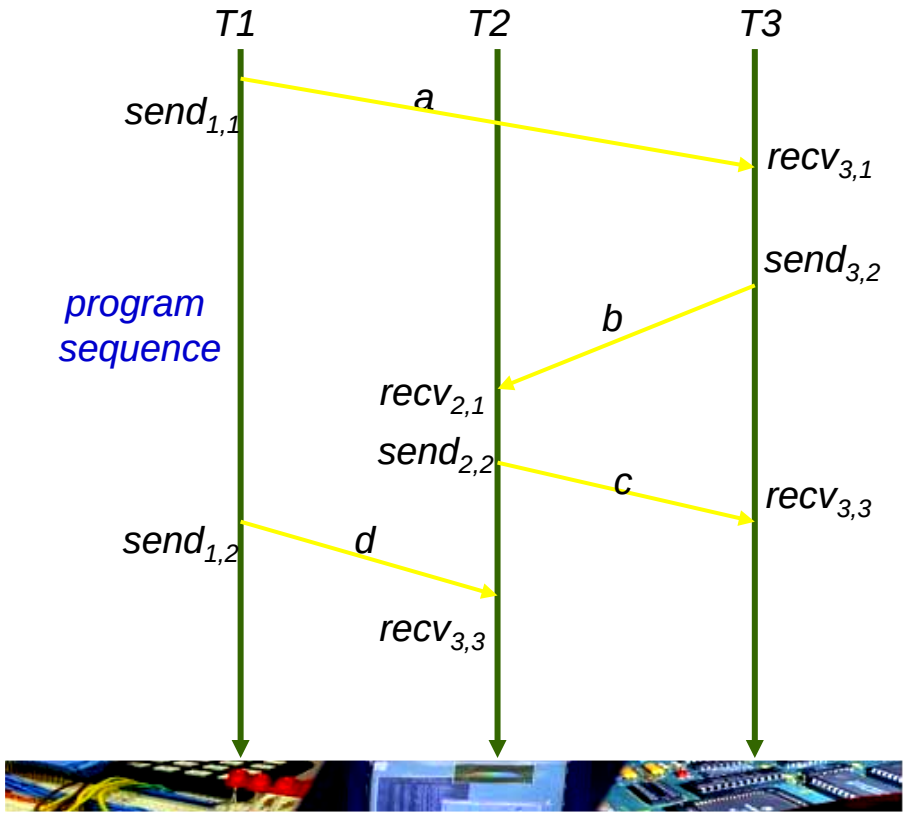
Reproducible Execution

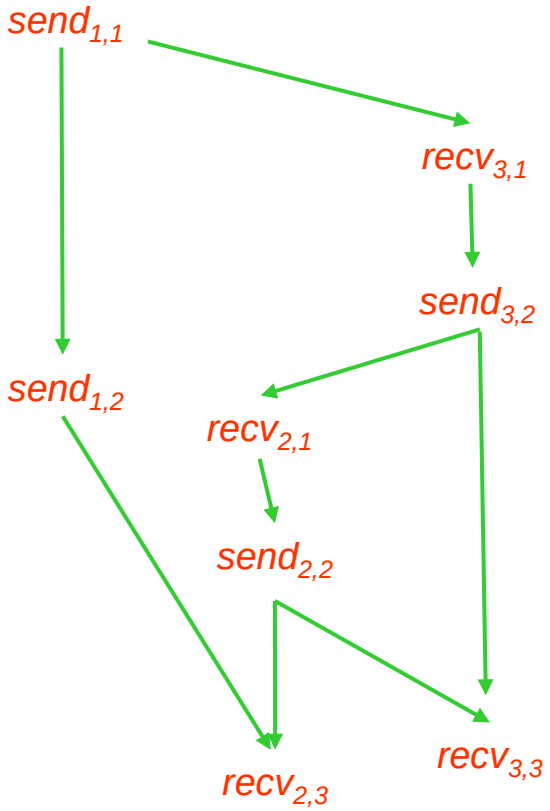
- Execution information must be logged for re-execution
- Overhead – ordering information or data, probe effect
- Static or dynamic (instrumentation at source or object code level)



Approach to Reproducible Execution

- Execution sequence → Partial order of synchronous events
- Preserve the order and apply the same IO events → reproducible execution



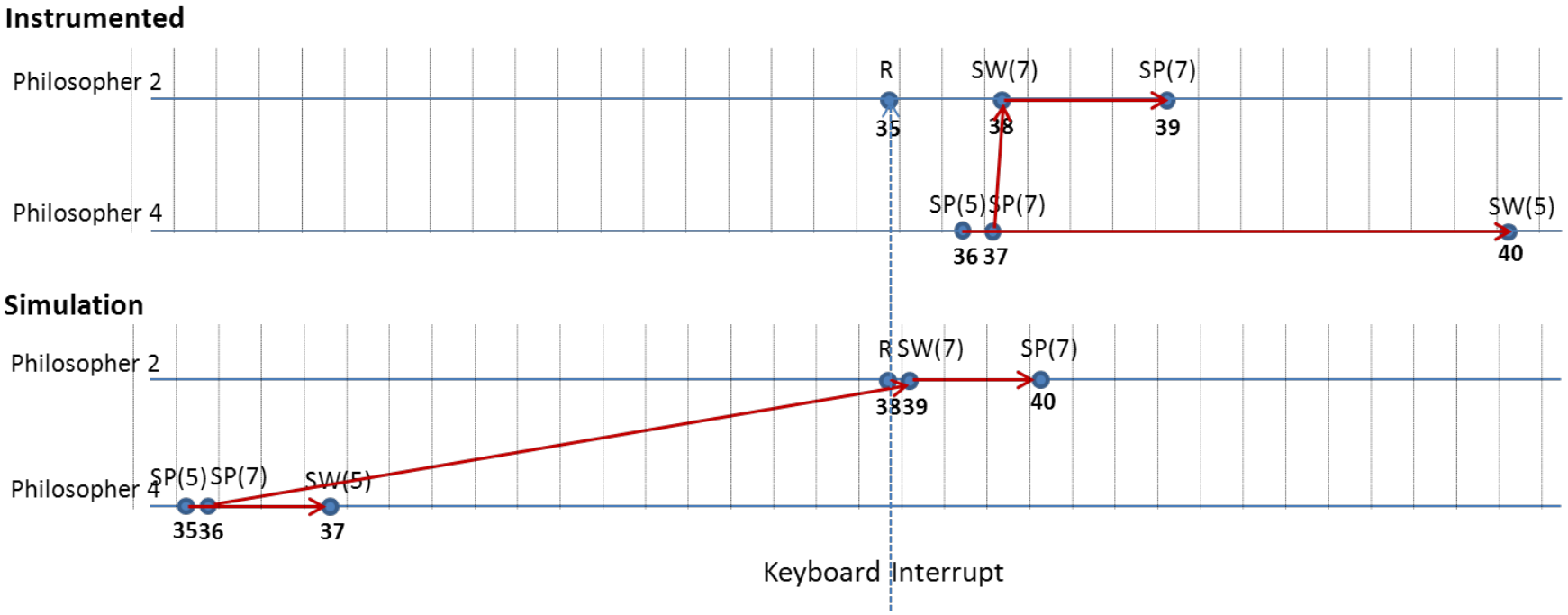


Existence of Probe Effect

- **Any instrumentation of multi-threaded program execution may**
  - change the temporal behavior of program execution
  - result in different ordering of execution events
- **To detect event order variations caused by instrumentation**
  - simulate program execution based on execution time (w/o overhead), arrival events, synchronization and scheduling actions.
  - program events from instrumented execution with execution time
  - interrupts arrive at absolute time

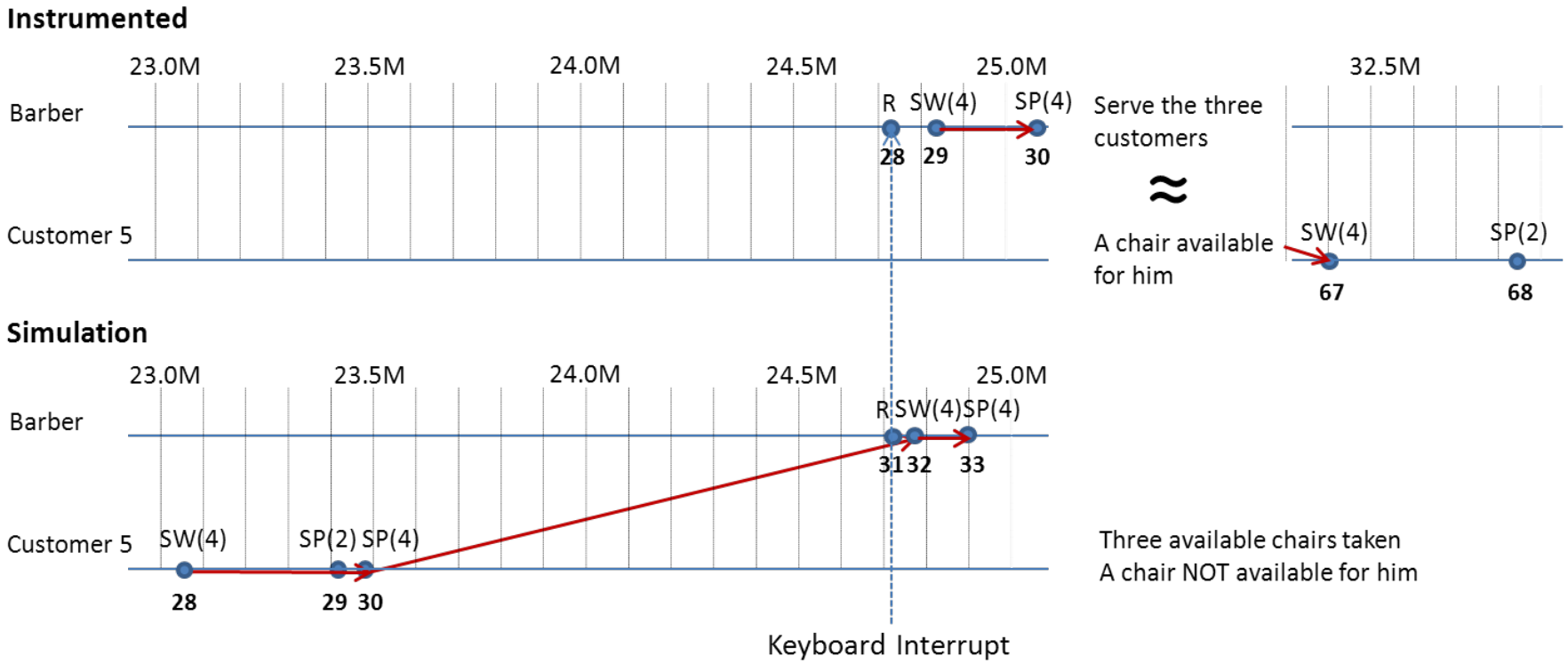
Test Cases on Probe Effect (1)

- Total order is changed but with same partial order



Test Cases on Probe Effect (2)

- Different logical order leading to different execution path



Data Race Detectors

- **A shared location is accessed by two different threads that**
  - are not ordered by happens-before relation
  - at least one of the accesses is a write
- **Many detectors for Java programs**
- **Static detectors – false alarms**
- **Dynamic detectors – need to instrument data accesses**
- **LockSet algorithms (Eraser) -- imprecise**
- **Happens-before algorithms – based on Lamport's vector clock**

Race Detector with Dynamic Granularity

- **Vector clock based data race detector for C/C++ programs**
  - On top of FastTrack and using Intel PIN for dynamic instrumentation
  - No need for a full VC on variables
  - VC from O(n) to O(1)

- **Share vector clock with neighboring memory locations**
  - Neighboring memory locations tend to be protected by the same lock (e.g. array, struct)

Performance Benchmark (1)

- **Comparison with Valgrind DRD and Inspector XE**

Benchmark program	Base time (sec)	Base Mem. (MB)	Slowdown			Memory Overhead			Data race detected		
			Valgrind DRD	Intel Inspector XE	Dynamic granularity	Valgrind DRD	Intel Inspector XE	Dynamic granularity	Valgrind DRD	Intel Inspector XE	Dynamic granularity
facesim	6.1	288	59	128	102	2.2	6.0	4.6	8909	31	8909
ferret	6.7	146	748	87	52	2.6	5.0	8.9	108	4	2
fluidanimate	2.0	248	--	89	81	--	12.4	2.2	--	7	1
raytrace	9.5	170	42	17	27	1.9	4.1	2.0	16	0	13
dedup	7.7	2682	--	--	85	--	--	1.0	--	--	0
streamcluster	3.8	30	66	108	137	4.2	17.5	3.7	1067	61	1079
ffmpeg	3.0	95	120	--	109	2.6	--	3.1	0	--	1
pbzip2	5.7	67	64	99	39	2.9	8.6	3.4	0	0	0
hmmsearch	26.6	23	74	64	45	4.4	21.9	4.3	1	2	1
Average			168	85	75	3	11	4			

Conclusion

- **Continuous improvement for the replay mechanism**
  - Record network messages at a sniff server
  - Checkpointing for long running systems
- **Multicore**
  - To overcome potential problems caused by concurrency and scheduling