From Logical Execution Time to Principled Systems Engineering

Christoph M. Kirsch, University of Salzburg, Austria

Dagstuhl Seminar on Logical Execution Time, Schloss Dagstuhl, Wadern, Germany

Joint Work

- Joshua Auerbach
- David Bacon
- Krishnendu Chatterjee
- Perry Cheng
- Silviu Craciunas
- Arkadeb Ghosal
- David Grove
- Matthias Hauswirth
- Thomas Henzinger
- Michael Hind
- Ben Horowitz
- Daniel Iercan
- Eduardo Marques
- Rupak Majumdar
- V.T. Rajan
- Harald Röck
- Alberto Sangiovanni-Vincentelli
- Marco Sanvido
- Raja Sengupta
- Ana Sokolova
- Daniel Spoonhower
- Rainer Trummer
- Martin Vechev
- Eran Yahav

Topics 1996-2015

Concurrent Data Structures
[POPL13,POPL15]

Automated Theorem Proving
[CADE97,LICS99]

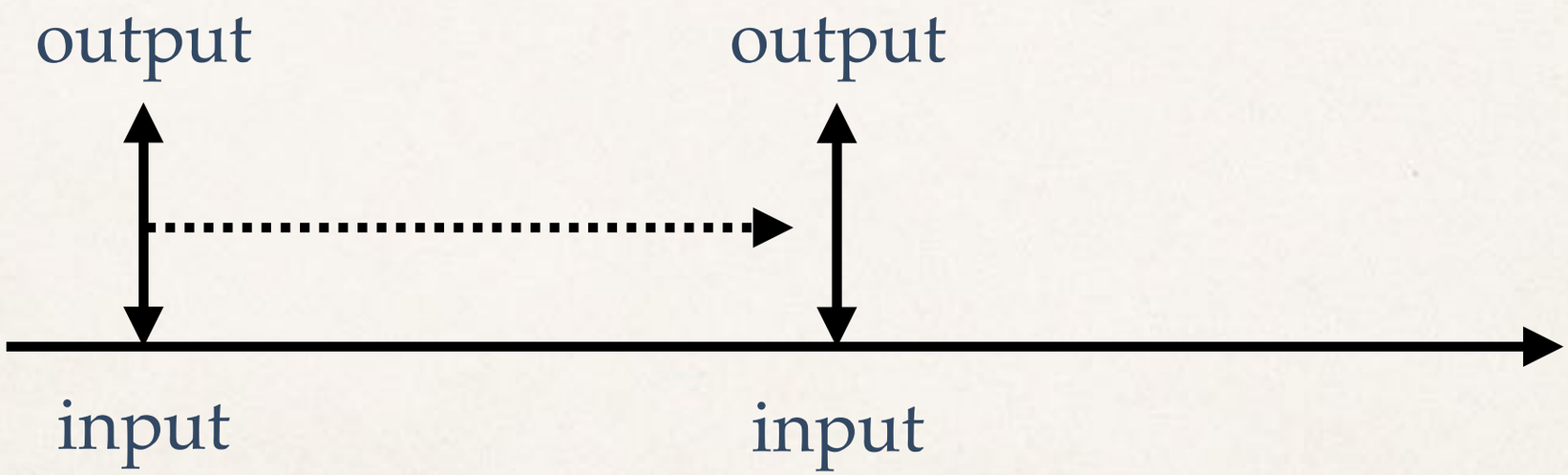
Memory Management
[USENIX08,OOPSLA15]

1996 Max Planck 1999 Berkeley 2004 Salzburg 2015

Real-Time Programming
[EMSOFT01,PLDI02,ProcIEEE03,TOPLAS07,RTSS09]

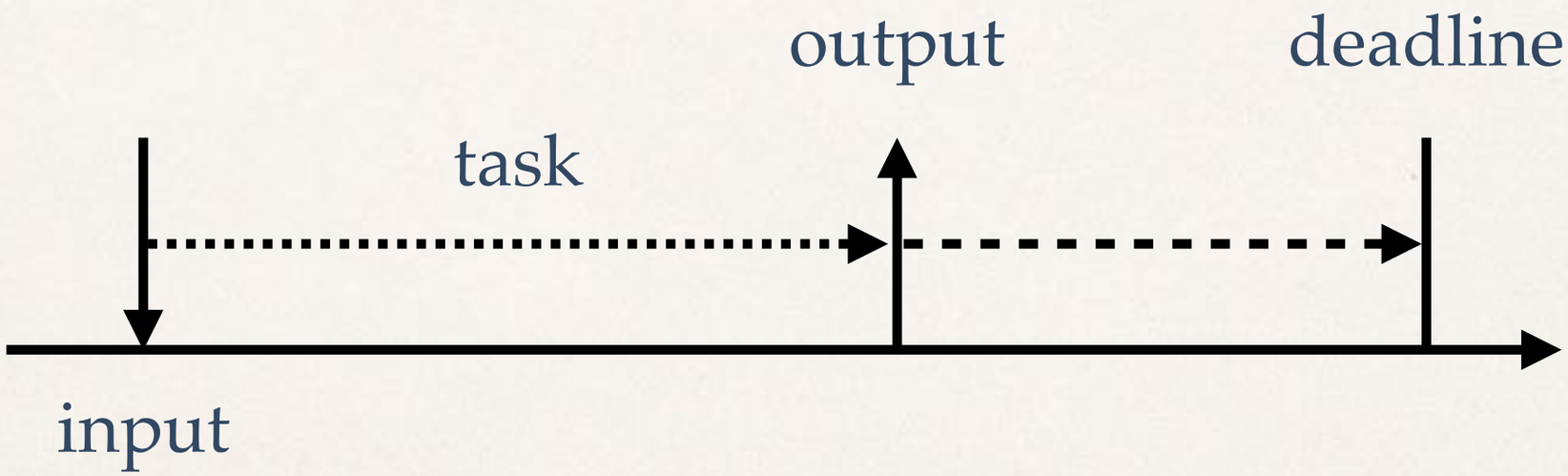
Logical Execution Time (LET) [EMSOFT01,ProcIEEE03]

Zero Execution Time (ZET) [HRTES07]



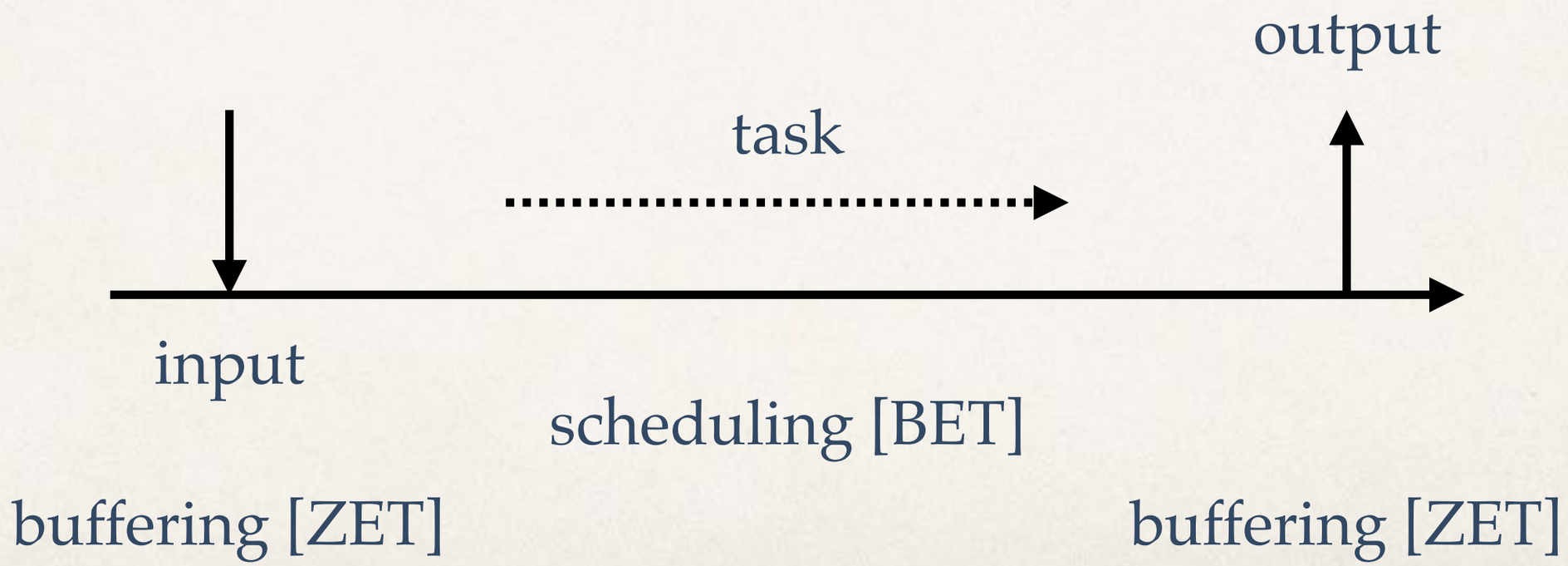
synchronous reactive programming

Bounded Execution Time (BET) [HRTES07]

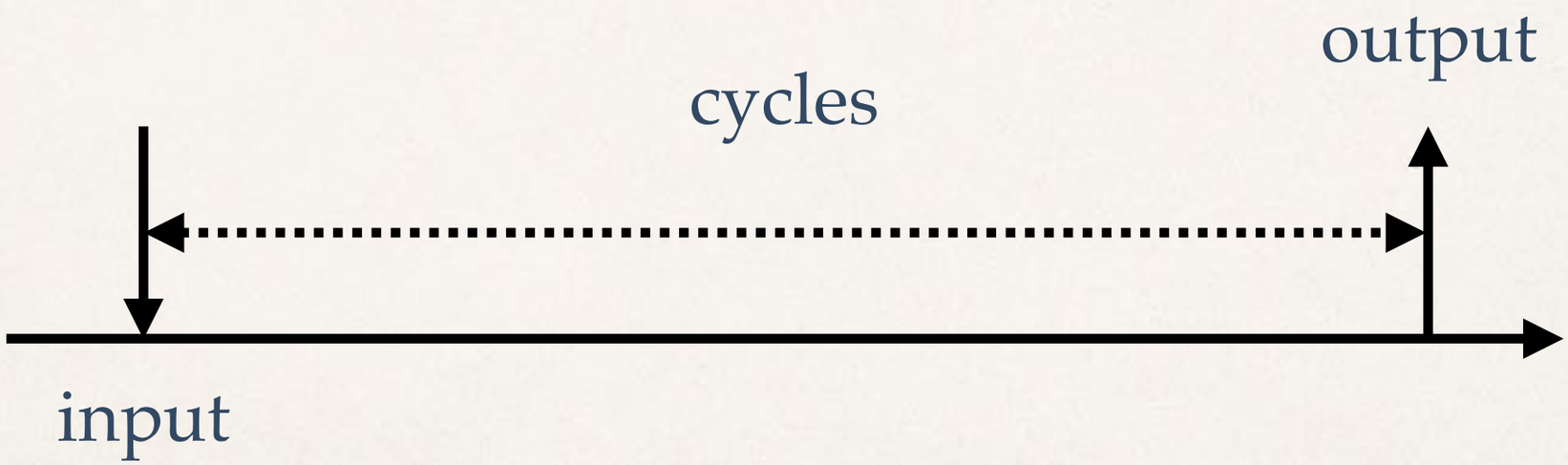


standard task model of real-time scheduling theory

Logical Execution Time (LET) [EMSOFT01,ProcIEEE03]

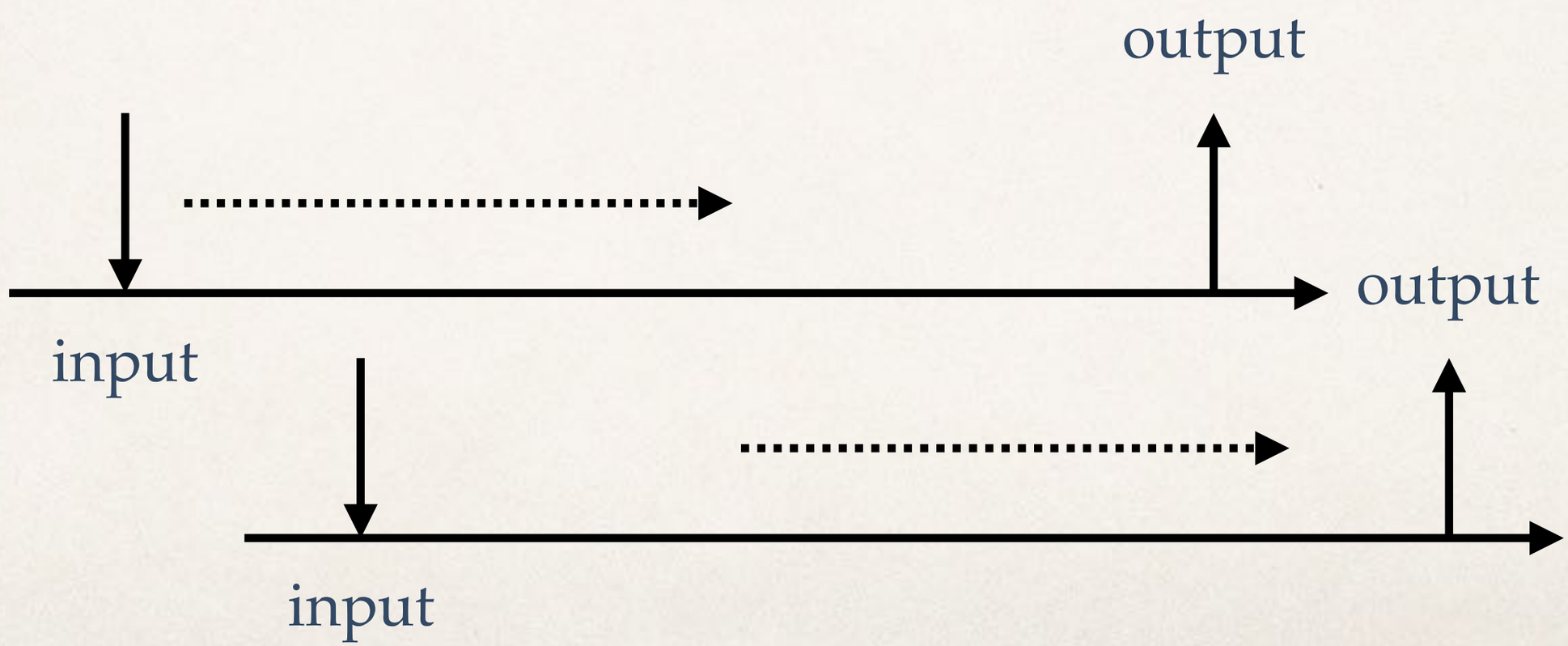


Physical Execution Time (PET) [HRTES07]



same or similar lower and upper bounds on cycle count per instruction

Concurrency with LET [EMSOFT01,ProcIEEE03]



LET 2001-2016

Giotto
[EMSOFT01]
[ProcIEEE03]

Schedule-Carrying Code
[EMSOFT03]

JAviator
[AIAA-GNC08]

Embedded Machine
[PLDI02]
[TOPLAS07]

Hierarchical Timing Language
[EMSOFT06]
[RTSS09]

Time Determinism [EMSOFT01]

The execution of a LET program f is time-deterministic
if, for all sequences I of input values and times,
the program produces the same sequences f(I) of output values and times

Time Safety [EMSOFT01]

The execution of a LET program is time-safe
if all tasks of the program complete within their LETs

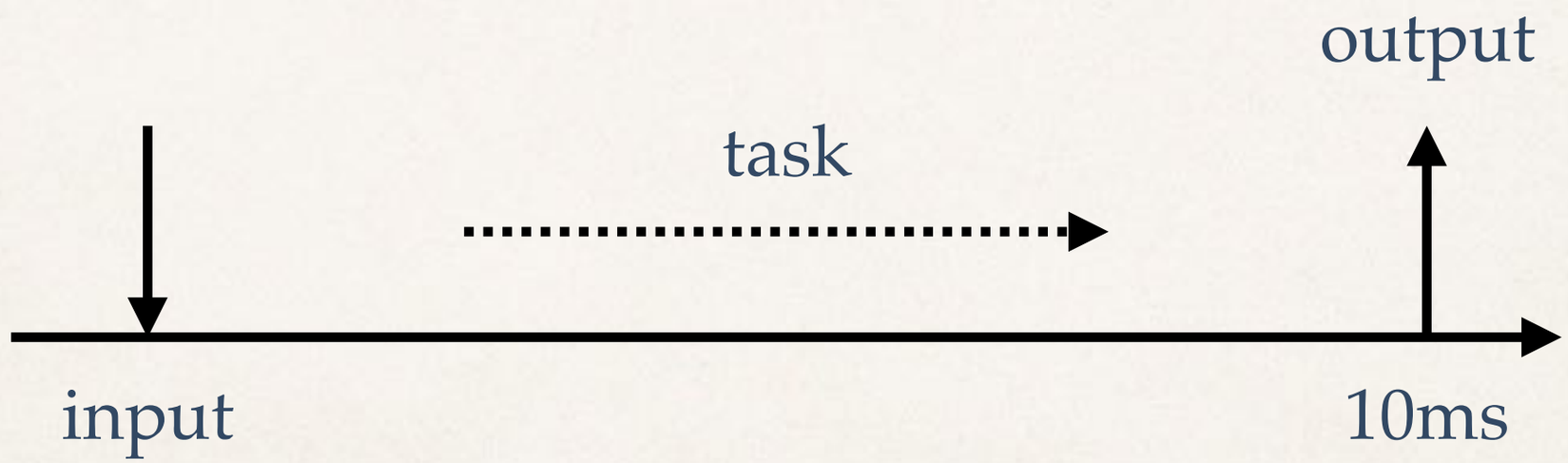
Giotto [EMSOFT01,ProcIEEE03]

The execution of a Giotto program is time-deterministic
if the execution of the program is time-safe

E Machine [PLDI02,TOPLAS07]

```
A: write(output)
   read(input)
   release(task)
   future(10ms, A:)
```

```
write(output)
read(input)
release(task)
future(10ms, A:)
```



E Machine [PLDI02,TOPLAS07]

The execution of E code compiled from a Giotto program
is time-deterministic if the execution of the program is time-safe

If the time safety check was wrong
the E machine throws an exception

Schedule-Carrying Code (SCC) [EMSOFT03]

SCC is E code plus S code which determines when to run which task

It is generally easier to check whether SCC is time-safe
than generating SCC that is time-safe

Hierarchical Timing Language (HTL) [EMSOFT06,RTSS09]

A concrete HTL program is time-safe
if it refines a time-safe, abstract HTL program

Standard Workflow Applies

- Time safety checking
- E code generation
- Separate compilation
- Incremental compilation
- Dynamic linking and loading

xGiotto [Ghosal et al.]

Flexible LET [Derler et al.]

Network Code [Fischmeister et al.]

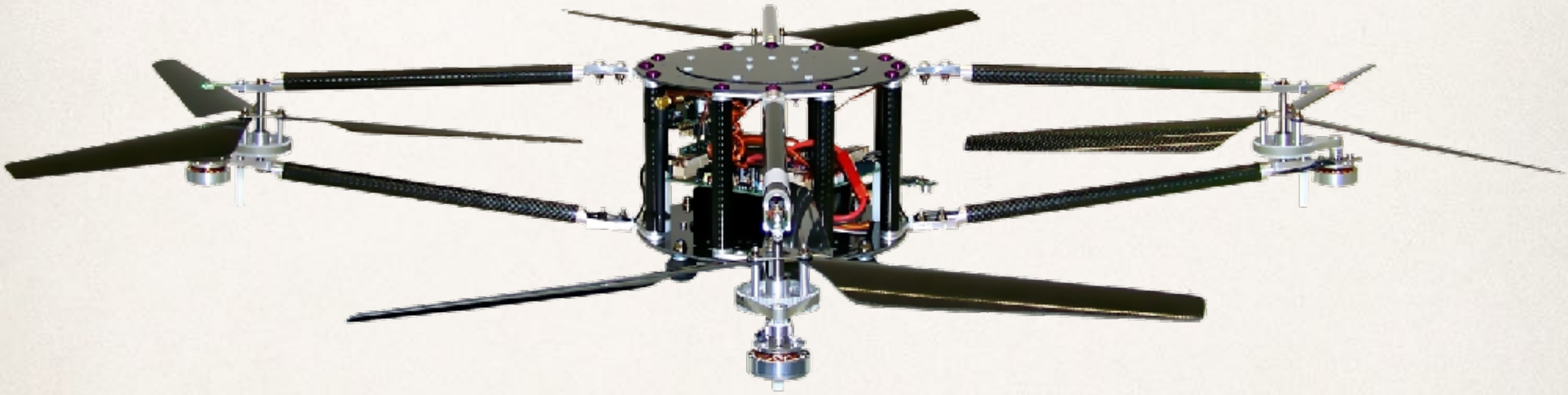
Timing Definition Language (TDL) [Pree et al.]

LET in AUTOSAR for Multicore [Di Natale et al.]

LET in SystemC [Puschner et al.]

LET on Time-Predictable Multicore [Ungerer et al.]

javiator.cs.uni-salzburg.at



[American Institute of Aeronautics and Astronautics GNC 2008]

selfie.cs.uni-salzburg.at

Joint Work

- Alireza Abyaneh
- Martin Aigner
- Sebastian Arming
- Christian Barthel
- Simon Bauer
- Thomas Hütter
- Alexander Kollert
- Michael Lippautz
- Cornelia Mayer
- Philipp Mayer
- Christian Moesl
- Simone Oblasser
- Clement Poncelet
- Sara Seidl
- Ana Sokolova
- Manuel Widmoser

Selfie: Teaching Computer Science [selfie.cs.uni-salzburg.at]

- *Selfie* is a self-referential 10k-line C implementation (in a single file) of:

1. a self-compiling compiler called ***starc*** that compiles a tiny subset of C called C Star (C*) to a tiny subset of RISC-V called RISC-U,

2. a self-executing emulator called ***mipster*** that executes RISC-U code including itself when compiled with starc,

3. a self-hosting hypervisor called ***hypster*** that virtualizes mipster and can host all of selfie including itself,

4. a self-executing symbolic execution engine called ***numster*** that executes RISC-U code symbolically when compiled with starc which includes all of selfie,

5. a tiny C* library called ***libcstar*** utilized by all of selfie, and

6. a tiny, experimental SAT solver called ***babysat***.

Also, there is a…

- linker (in-memory only)
- disassembler (w/ source code line numbers)
- debugger (tracks full machine state w/ rollback)
- profiler (#proc-calls, #loop-iterations, #loads, #stores)

Discussion of Selfie reached 3rd place on Hacker News

news.ycombinator.com

Website

selfie.cs.uni-salzburg.at

Book (Draft)

leanpub.com/selfie

Code

github.com/cksystemsteaching/selfie

```
> ./selfie -c selfie.c

./selfie: this is selfie's starc compiling selfie.c

./selfie: 176408 characters read in 7083 lines and 969 comments
./selfie: with 97779(55.55%) characters in 28914 actual symbols
./selfie: 261 global variables, 289 procedures, 450 string literals
./selfie: 1958 calls, 723 assignments, 57 while, 572 if, 243 return
./selfie: 121660 bytes generated with 28779 instructions and 6544
bytes of data
```

compiling `selfie.c` *with x86* `selfie` *executable*

(takes seconds)

```
> ./selfie -c selfie.c -m 2 -c selfie.c

./selfie: this is selfie's starc compiling selfie.c

./selfie: this is selfie's mipster executing selfie.c with 2MB of
physical memory

selfie.c: this is selfie's starc compiling selfie.c

selfie.c: exiting with exit code 0 and 1.05MB of mallocated memory

./selfie: this is selfie's mipster terminating selfie.c with exit code
0 and 1.16MB of mapped memory
```

compiling `selfie.c` *with x86* `selfie` *executable into a RISC-U executable*

and

then running that RISC-U executable to compile `selfie.c` *again*

(takes ~6 minutes)

```
> ./selfie -c selfie.c -m 2 -c selfie.c -m 2 -c selfie.c
```

compiling `selfie.c` *with x86* `selfie` executable

and

then running that executable to compile `selfie.c` *again*

and

then running that executable to compile `selfie.c` *again*

(takes ~24 hours)

```
> ./selfie -c selfie.c -m 2 -c selfie.c -y 2 -c selfie.c
```

compiling `selfie.c` *with x86* `selfie` *executable*

and

then running that executable to compile `selfie.c` *again*

and

then ***hosting*** *that executable in a virtual machine to compile* `selfie.c` *again*

(takes ~12 minutes)

Ongoing Work

Verification

- SAT/SMT Solvers (microsat/boolector)
- Symbolic Execution Engine (KLEE/SAGE)
- Inductive Theorem Prover (ACL2)

-> microsat in C* is as fast as in C (forget structs, arrays, &&, ||, goto)

ISAs

1. Large memory and multicore support
2. x86 support through binary translation
3. ARM support?

Thank you!

AUSTRIAN COMPUTER SCIENCE DAY 2018

15.06.2018 / SALZBURG

acsd2018.cs.uni-salzburg.at