Forum on specification & Design Languages (FDL'20)

FDL stimulates scientific and controversial discussions in a friendly and productive environment.

New trends and traditional topics in the broad fields of embedded/electronics/software systems and languages merge in a lively and cross-discipline research & industrial community.

Calls for Special sessions, Full (8 pp), short (4 pp), and WiP/PhD Forum/Poster (2 pp) papers.

Keynotes: Edward Lee / UC Berkeley,
Manuel Serrano / Inria & Université Côte d'Azur,
Hauke Fuhrmann / Scheidt & Bachmann

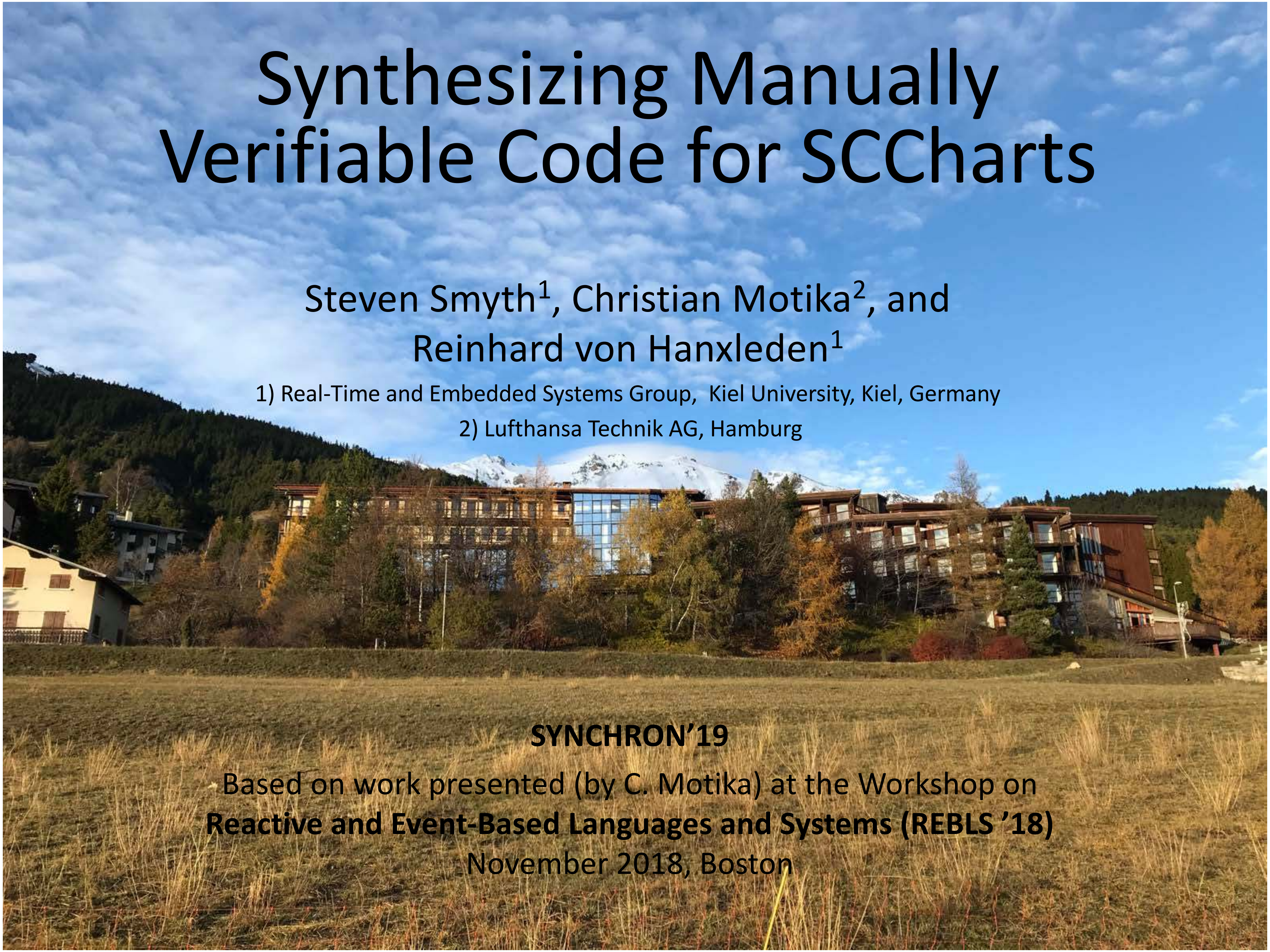
7–9 September 2020 | Kiel, Germany

Deadlines:

Special Sessions:	March 22, 2020
Paper Deadline:	**May 29, 2020**
PhD/WiP Deadline:	June 12, 2020
Author Notification:	June 28, 2020
Final Version:	July 19, 2020

Website: www.fdl-conference.org | Contact: fdl2020@easychair.org

Re FDL'19: Open call for ACM TECS Special Issue on Specification and Design Languages
Deadline: Feb. 1, 2020 (firm)
Contacts: Alain Girault, Reinhard von Hanxleden



Synthesizing Manually Verifiable Code for SCCharts

Steven Smyth[1], Christian Motika[2], and Reinhard von Hanxleden[1]

1) Real-Time and Embedded Systems Group, Kiel University, Kiel, Germany

2) Lufthansa Technik AG, Hamburg

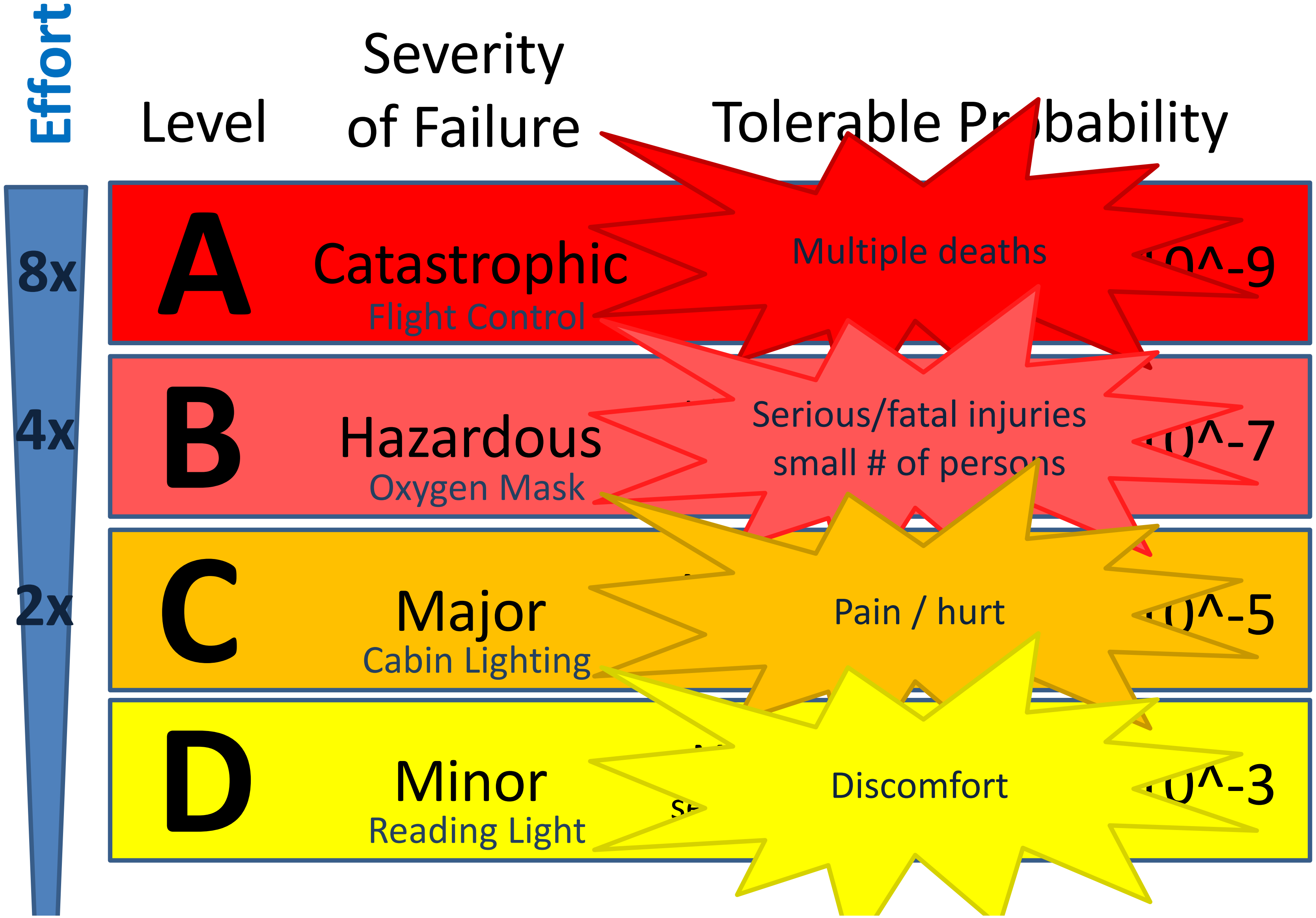
SYNCHRON'19

Based on work presented (by C. Motika) at the Workshop on
Reactive and Event-Based Languages and Systems (REBLS '18)
November 2018, Boston

Development Assurance Level (DAL)

*Level of **rigor** w.r.t. development assurance tasks*

(defined during safety assessment)

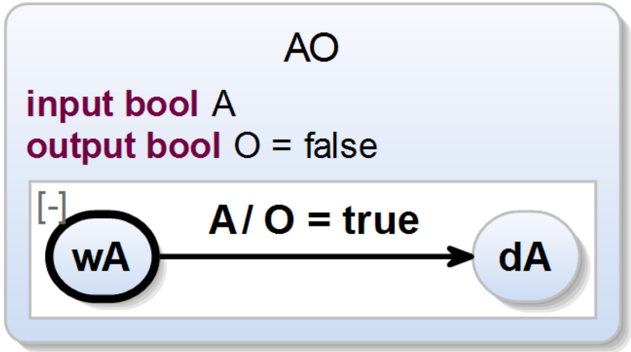


Aerospace Software

Statemachines in DAL-B/DAL-C Software

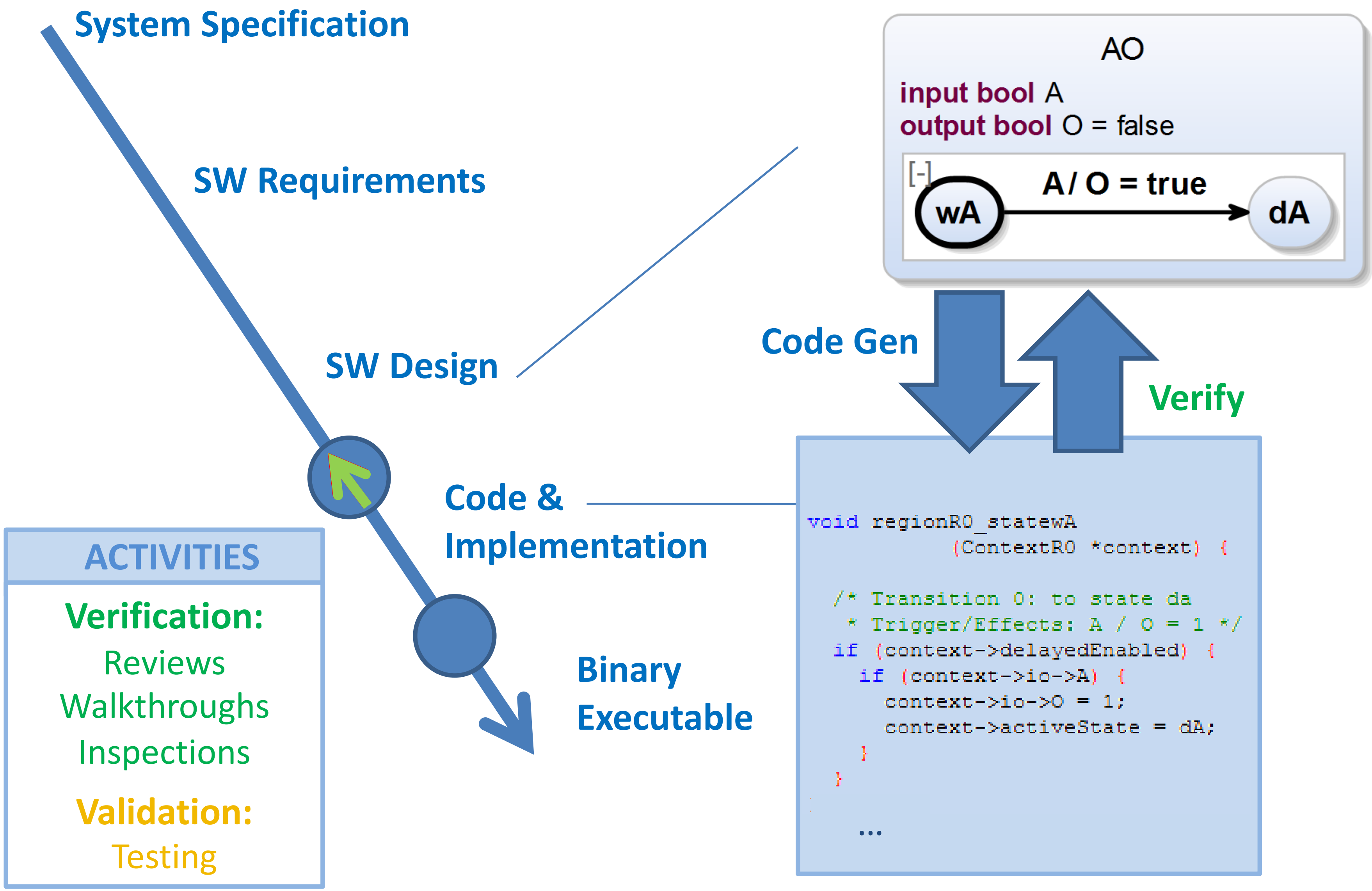
- Statemachines used in specification, SW requirement and/or SW design phase
- Code automatically synthesized

⇒ **Manual verification required**



Aerospace Software

GOAL: Ease Manual Verification Steps

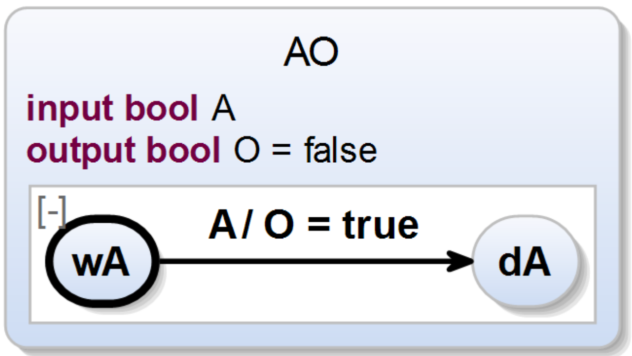


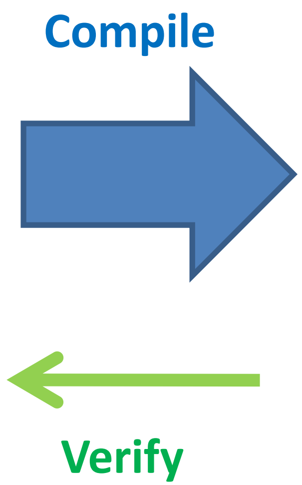
Goal: Generate Statecharts Code

that is Manually Verifiable

Outline:

1. SCCharts
2. State-based code generation
3. User Study



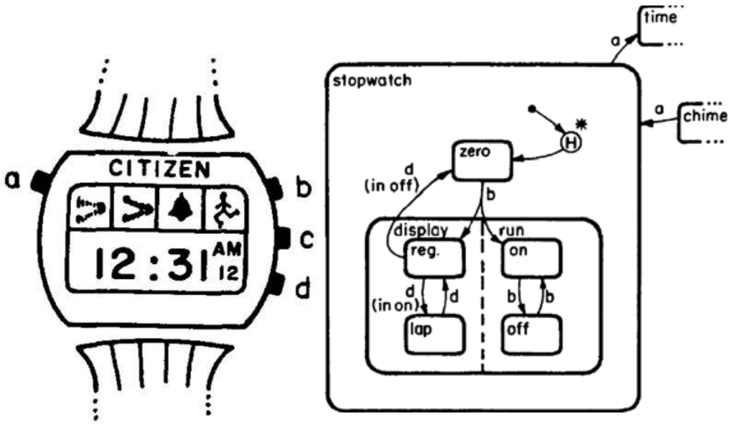



```
void regionR0_statewA
            (ContextR0 *context) {

  /* Transition 0: to state da
   * Trigger/Effects: A / O = 1 */
  if (context->delayedEnabled) {
    if (context->io->A) {
      context->io->O = 1;
      context->activeState = dA;
    }
  }

    ...
```

Part I

SCCharts Intro

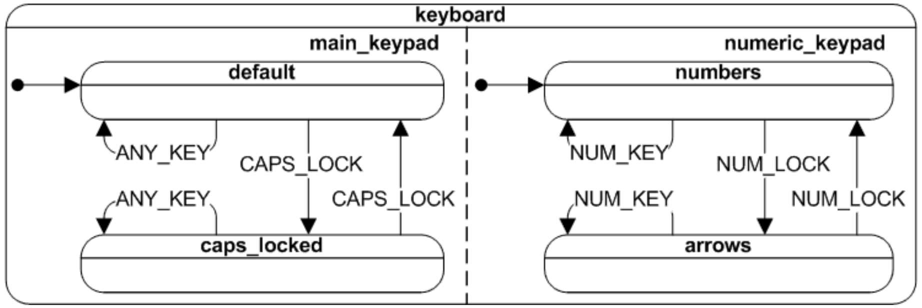
Statechart Dialects



Harel Statecharts - "an almost synchronous language" ('80)

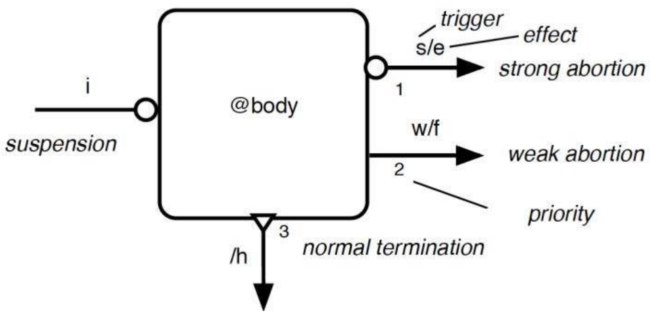
[Dagstuhl Report 104]

Harel
Statecharts: A Visual Formalism for Complex Systems
Science of Computer Programming, 1987



UML State Machines ('97) – "… a … variant of Harel statechart"

[Wikipedia]



SCADE Safe State Machines / SyncCharts ('95)

Charles André
SyncCharts: A Visual Representation of Reactive Behaviors
Research Report 95-52, I3S, Sophia Antipolis, 1995

SCCharts ('13)

- Successor of SyncCharts
- Sequentially Constructive Model of Computation

Reinhard von Hanxleden, Björn Duderstadt, Christian Motika, Steven Smyth, Michael Mendler, Joaquín Aguado, Stephen Mercer, Owen O'Brien.
SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications.
PLDI'14, Edinburgh, UK, June 2014. ACM.

- Collaborations:

THE UNIVERSITY OF AUCKLAND Te Whare Wananga o Tamaki Makaurau NEW ZEALAND

NATIONAL INSTRUMENTS™

CAU SEIT 350 JAHREN GANZ WEIT OBEN

UNIVERSITÄT BAMBERG

SCHEIDT&BACHMANN

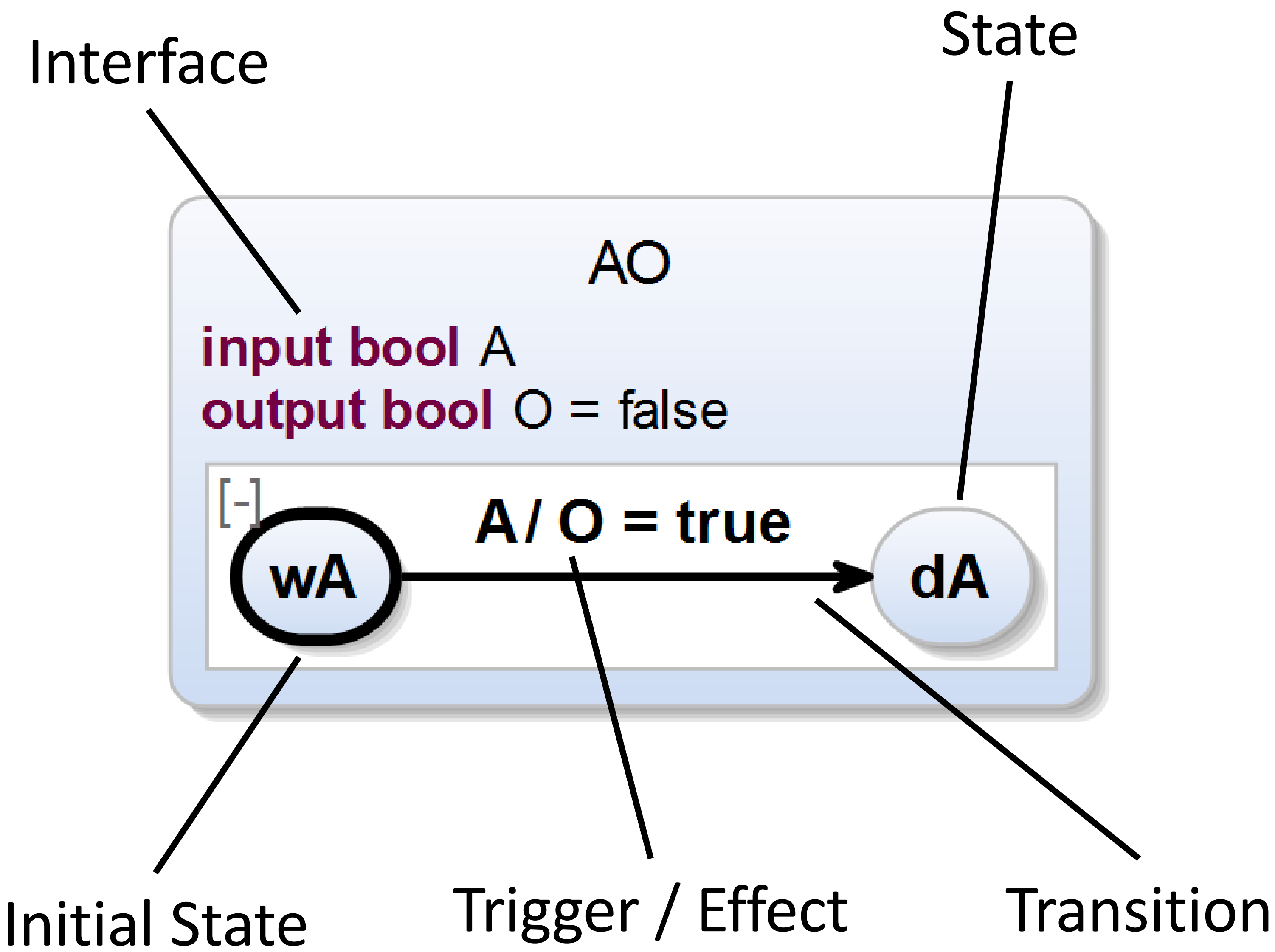
DFG Deutsche Forschungsgemeinschaft

- In Eclipse: KIELER
- In the browser: KEITH

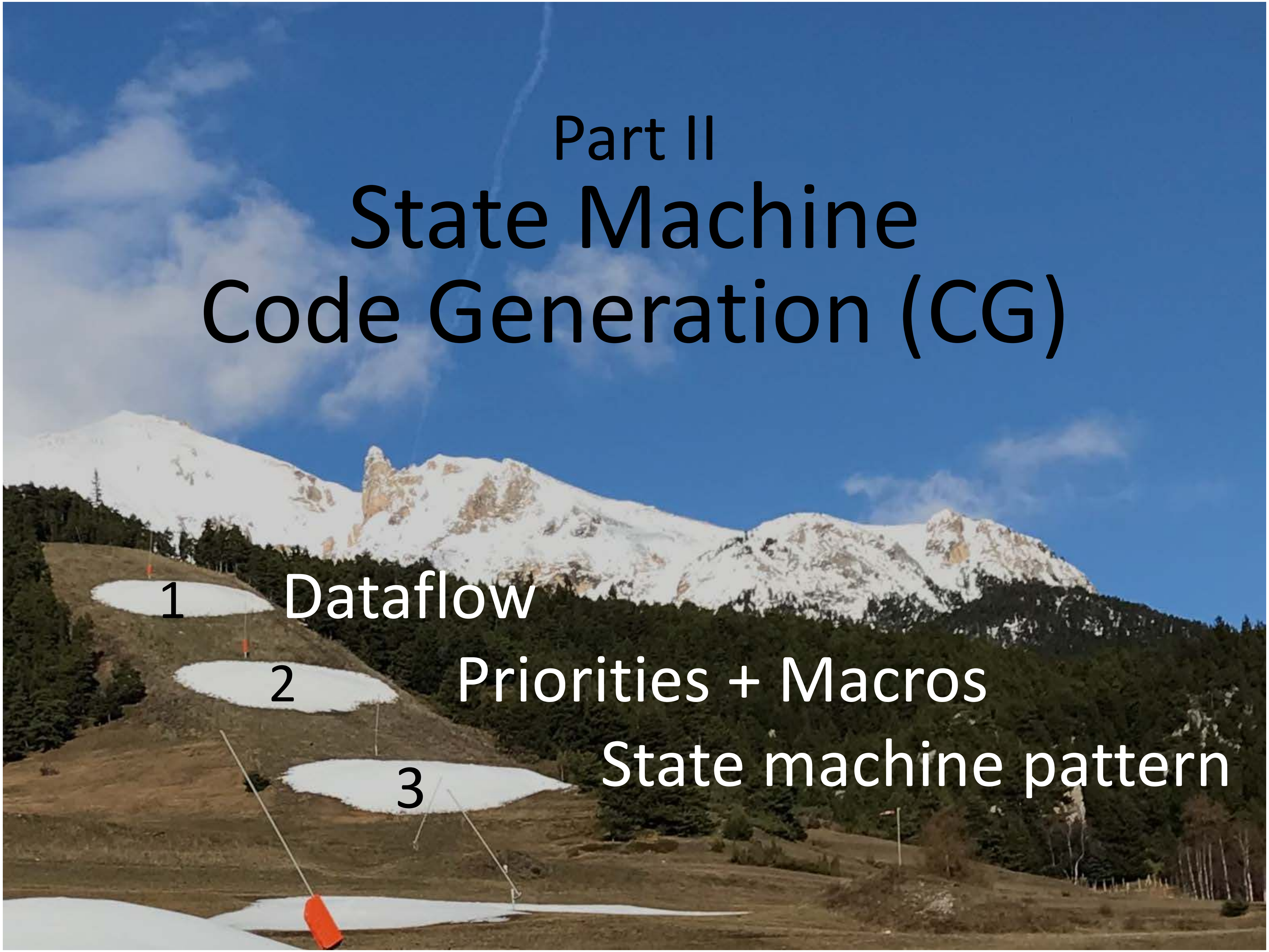
KIELER The Key to Efficient Modeling

RTNS Artifact Evaluated

AO SCChart



Hierarchy, Concurrency, Signals, ...



Part II

State Machine Code Generation (CG)

1 Dataflow

2 Priorities + Macros

3 State machine pattern

SCCharts defined/compiled by M2M Transformations:

Extended SCCharts

⇒

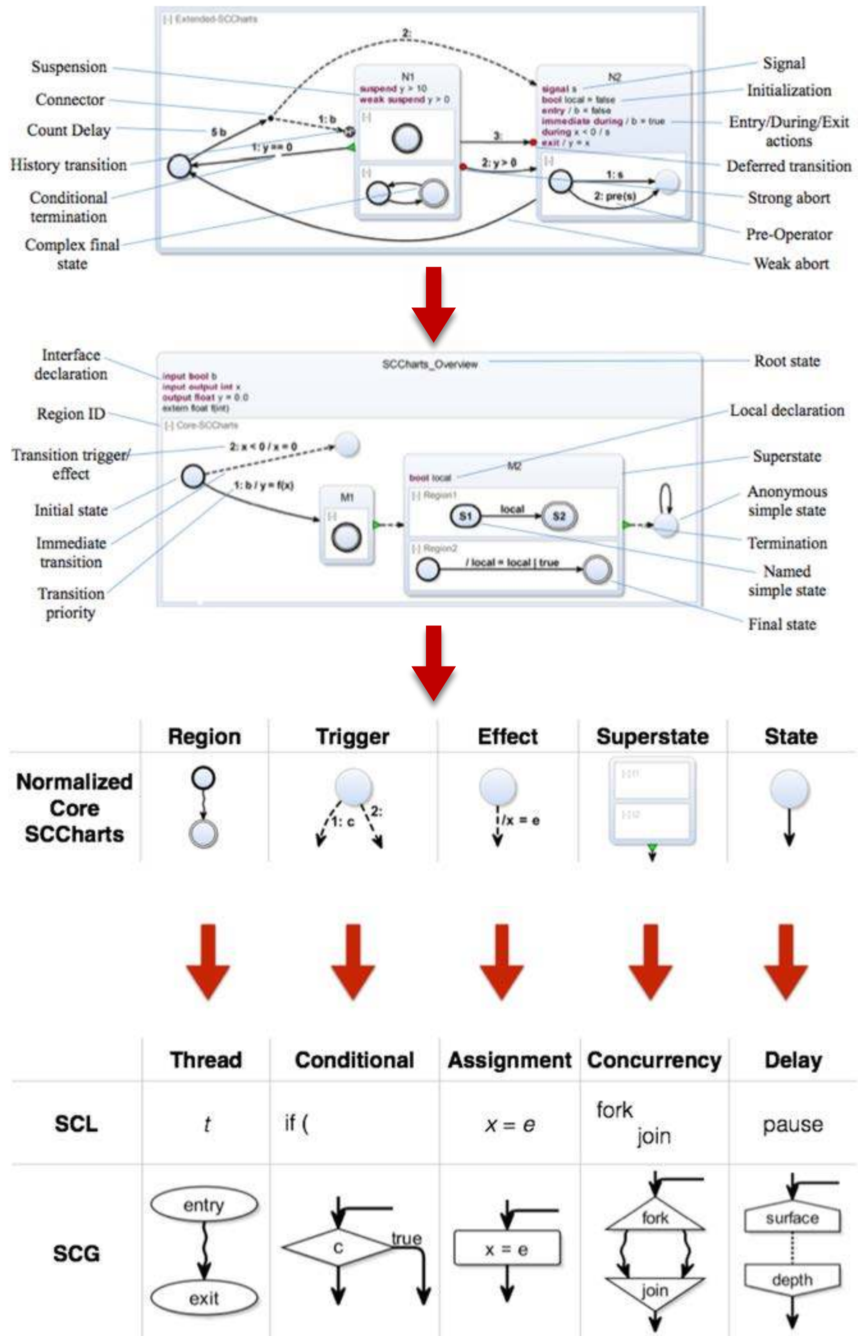
Core SCCharts

⇒

Normalized Core SCCharts

⇒

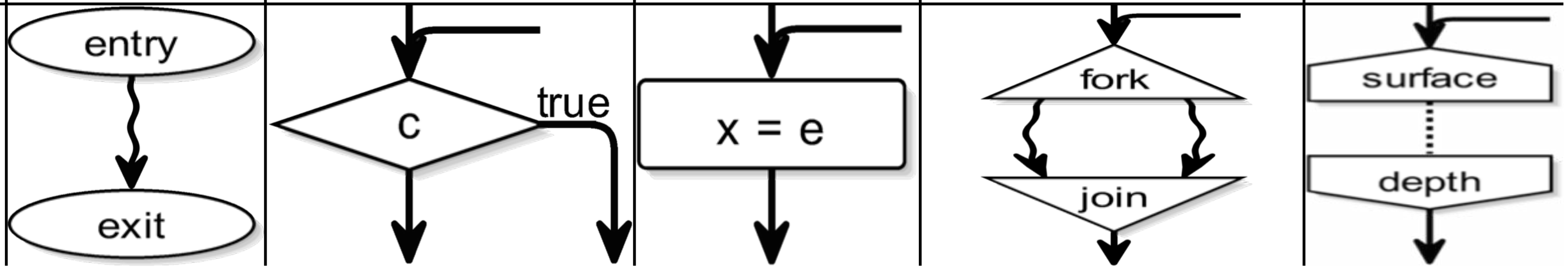
SCL/SCG



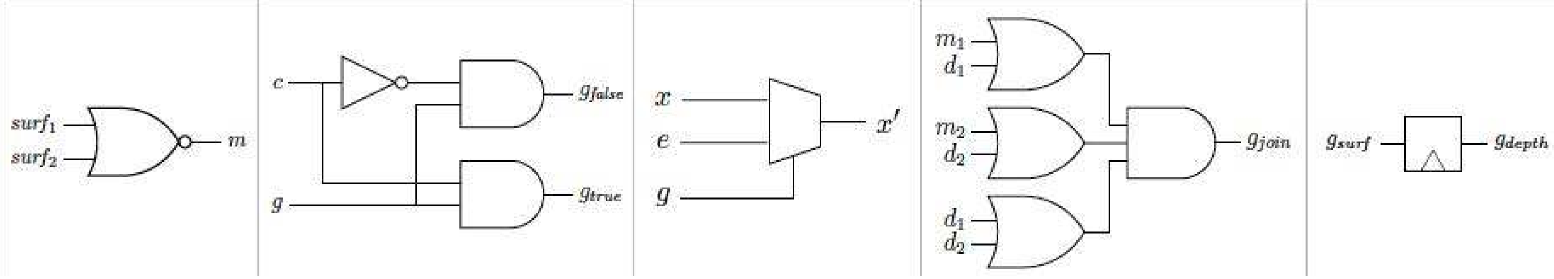
Dataflow Synthesis

	Thread	Conditional	Assign-ment	Concurrency	Delay
SCL	t	if (c) s_1 else s_2	$x = e$	fork t_1 par t_2 join	pause
Data-Flow Code	$d = g_{exit}$ $m = \neg \bigvee_{surf \in t} g_{surf}$	$g = \bigvee g_{in}$ $g_{true} = g \wedge c$ $g_{false} = g \wedge \neg c$	$g = \bigvee g_{in}$ $x' = g\ ?\ e : x$	$g_{join} = (d_1 \vee m_1)$ $\wedge (d_2 \vee m_2)$ $\wedge (d_1 \vee d_2)$	$g_{surf} = \bigvee g_{in}$ $g_{depth} =$ pre (g_{surf})

SCG



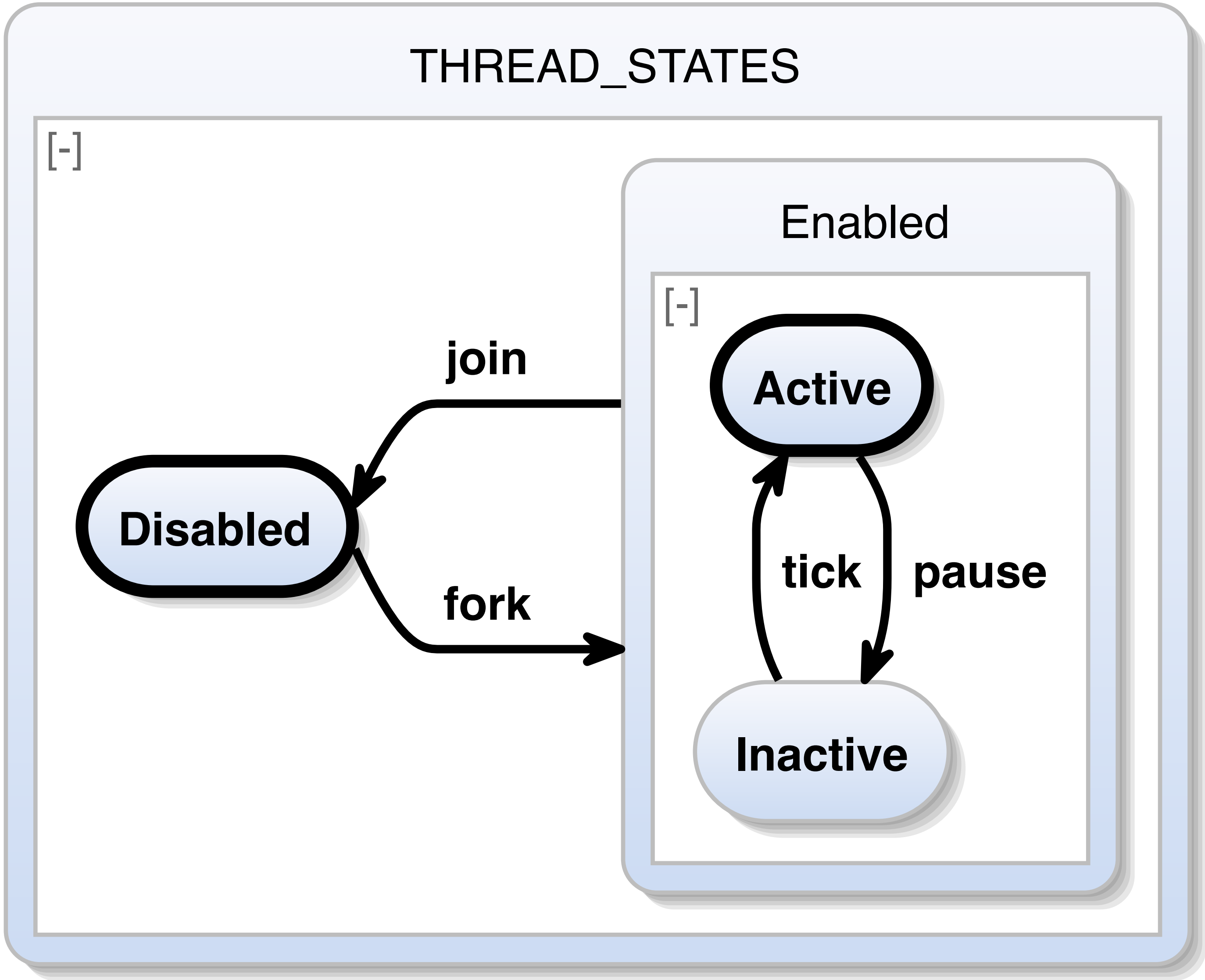
Circuits

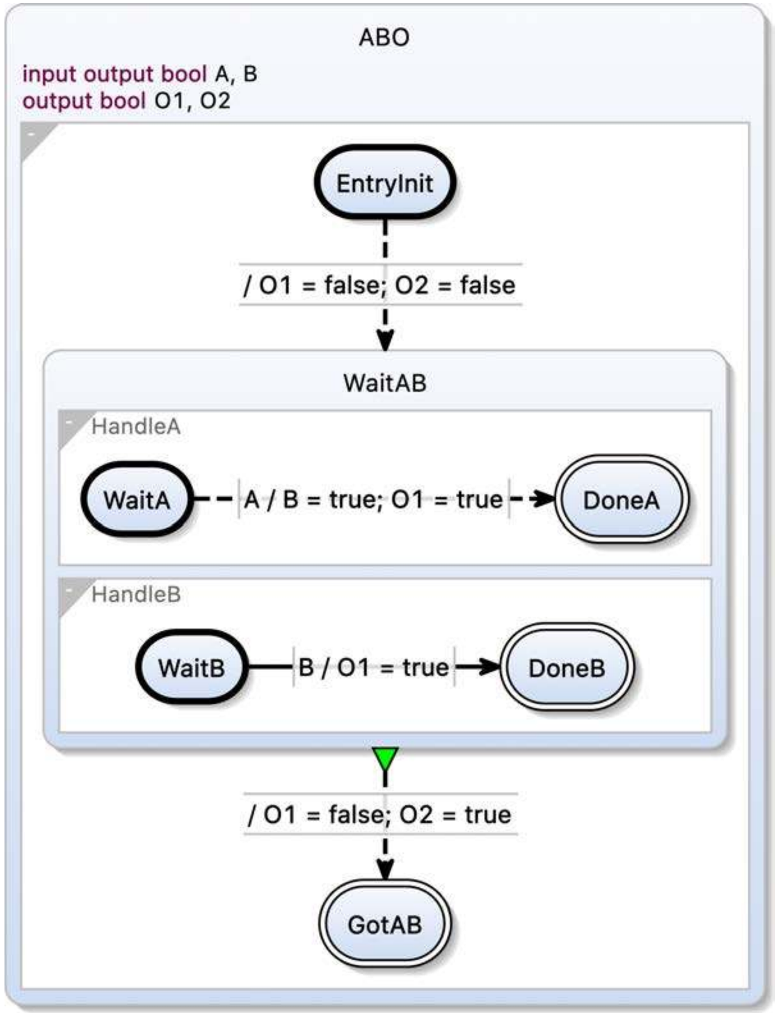


Priority-Based Synthesis

- More software-like
- Don't emulate control flow with guards/basic blocks, but with program counters/threads
- Priority-based thread dispatching
- SCL_P: SCL + PrioIDs
- In C: implemented as macros, using computed gotos
- In Java: no macros, no gotos;
  use while + break to emulate gotos
- Already more readable than dataflow/circuit synthesis, but **model structure still lost**

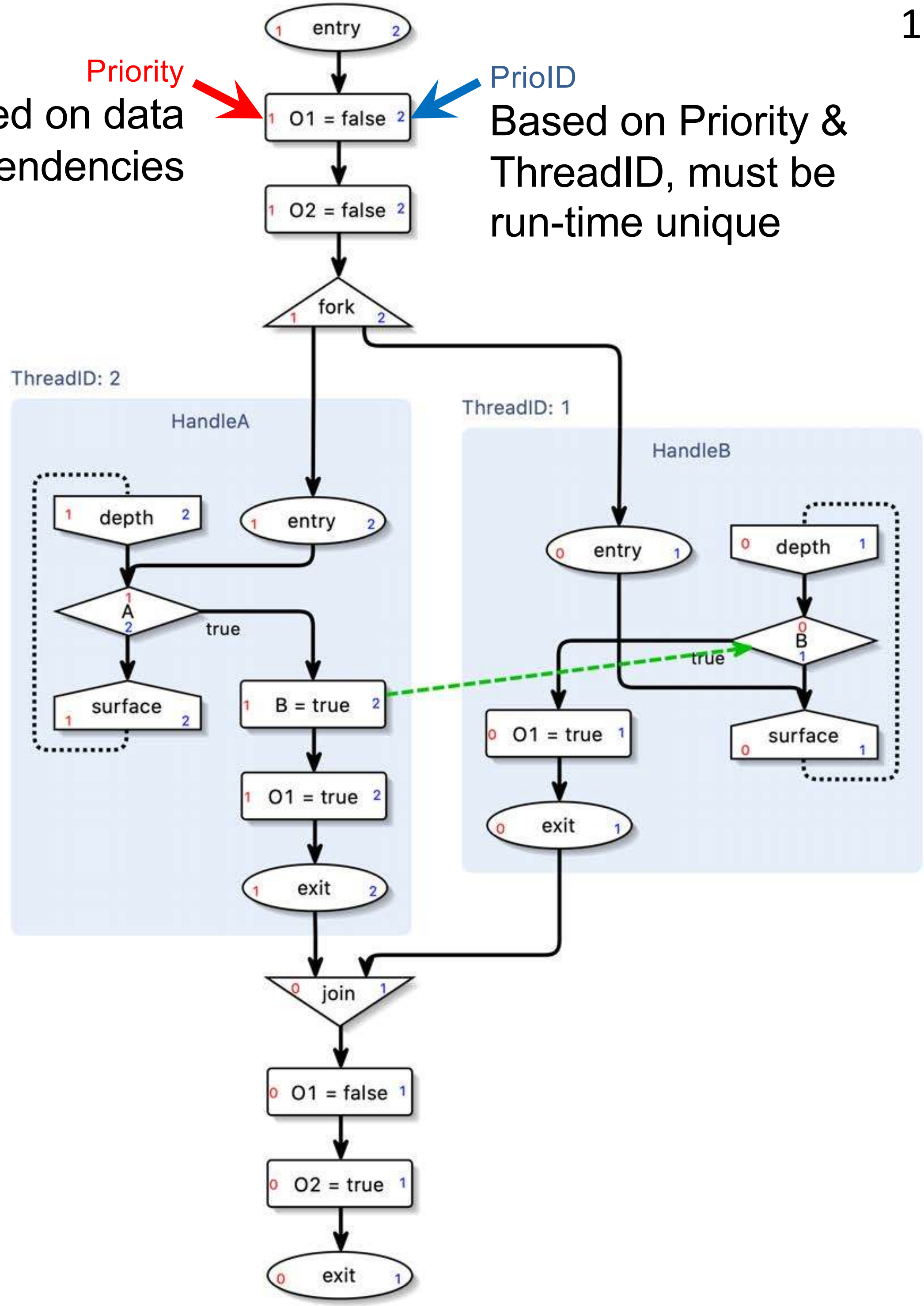
Priority-Based Synthesis



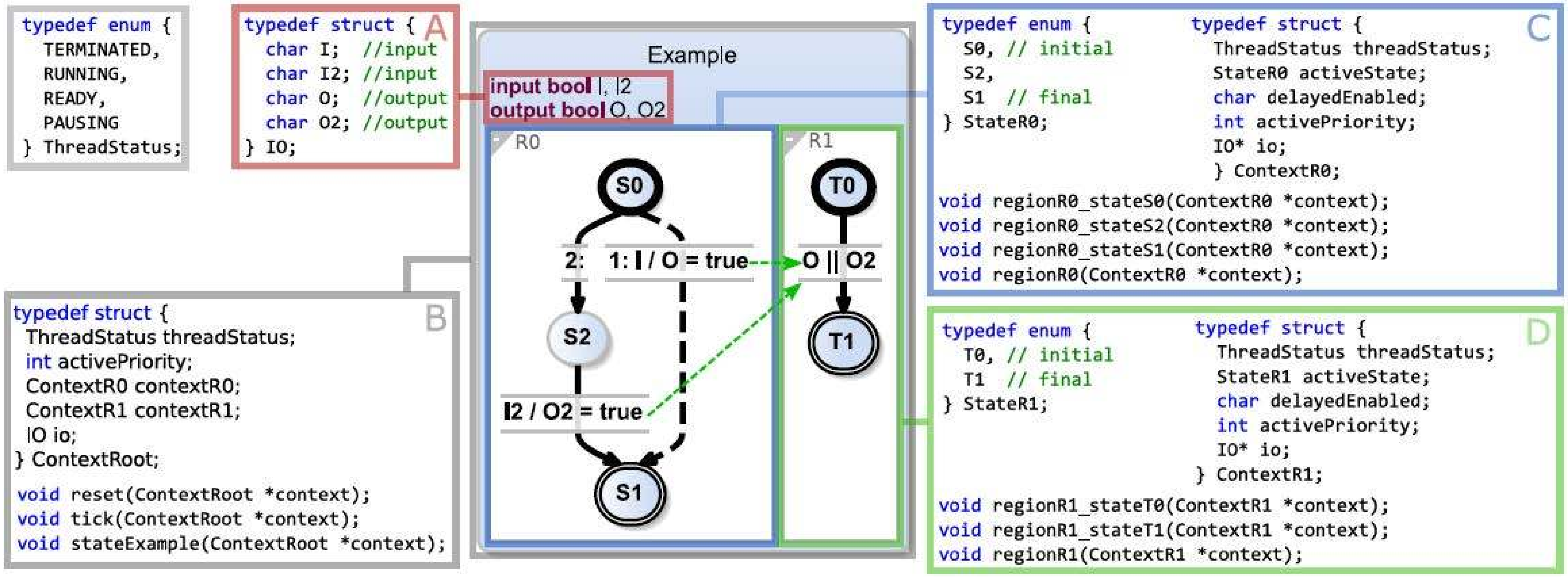


Priority
Based on data dependencies

PrioID
Based on Priority & ThreadID, must be run-time unique



Now: State-Based Synthesis



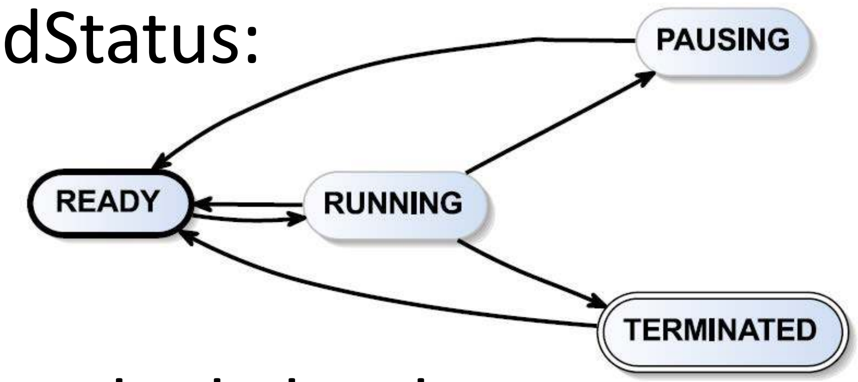
A: Interface

B: Root context

C: Region R0

D: Region R1

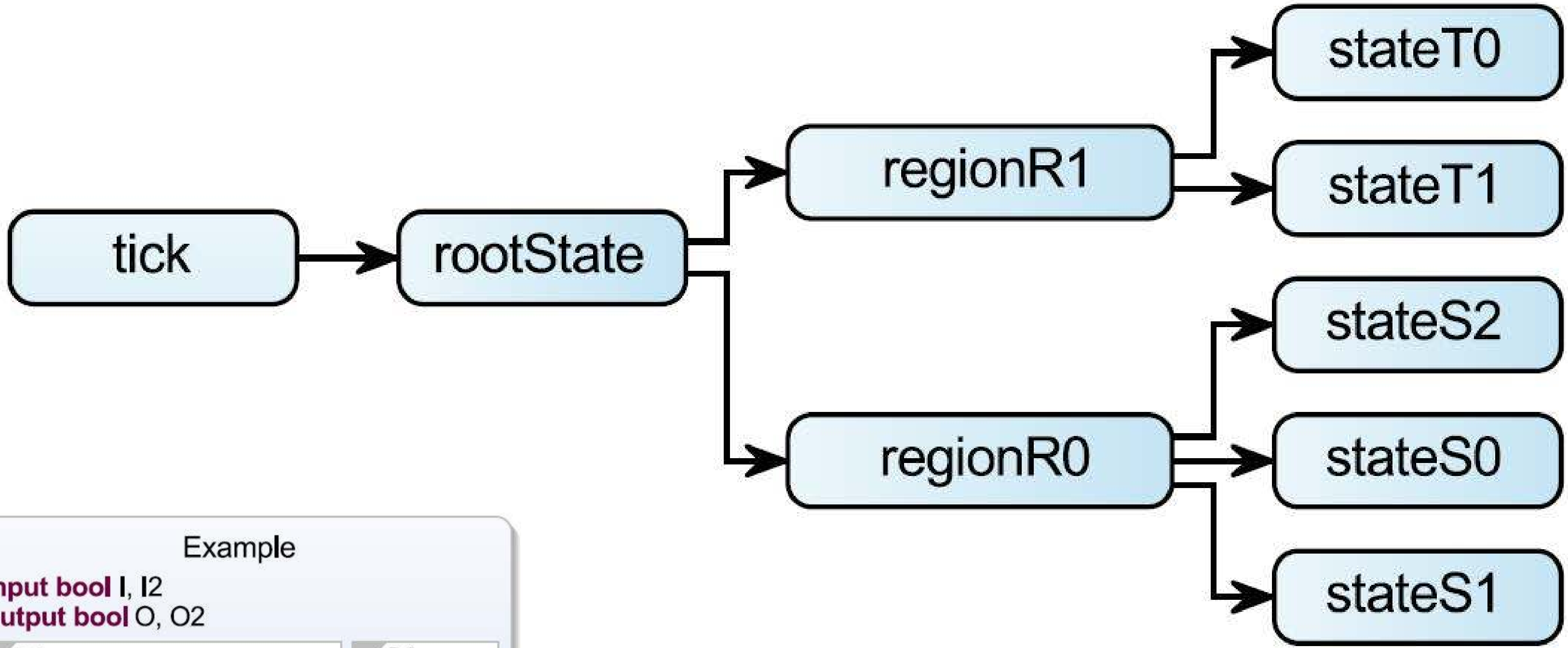
- Env. calls reset() & tick()
- ThreadStatus:

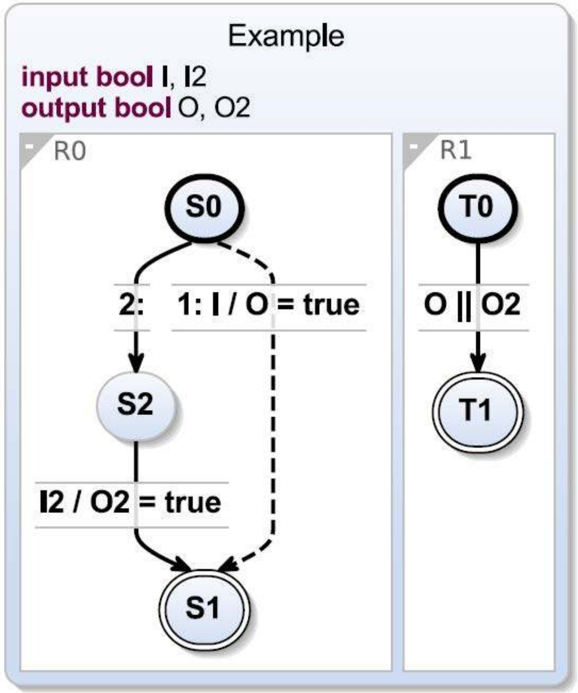


All regions and the root have a context struct

Data dependency (green dashed arrow)

Hierarchical Call Tree

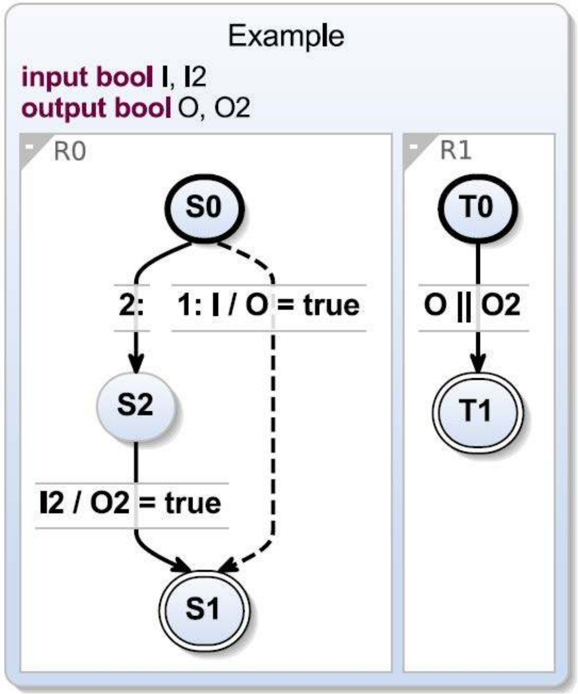




rootState: **stateExample()**

State Machine Pattern I

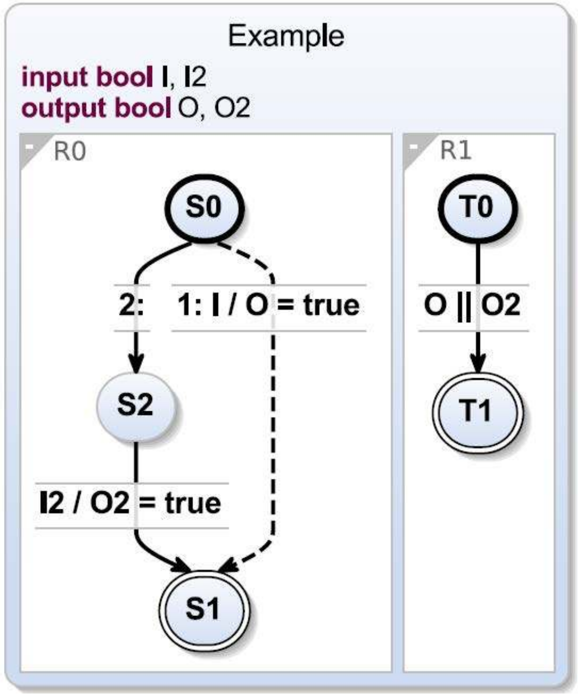
```
void regionR0(ContextR0 *context) {
  /* Cycle through the states of the region as long as this thread
   * is set to RUNNING. */
  while(context->threadStatus == RUNNING) {
    switch(context->activeState) {
      case S0:
        regionR0_stateS0(context);
        break;

      case S2:
        regionR0_stateS2(context);
        break;

      case S1:
        regionR0_stateS1(context);
        break;
    }
  }
}
```



- Respect naming
- Automated comments
- Hierarchical hide details in functions

State Machine Pattern II

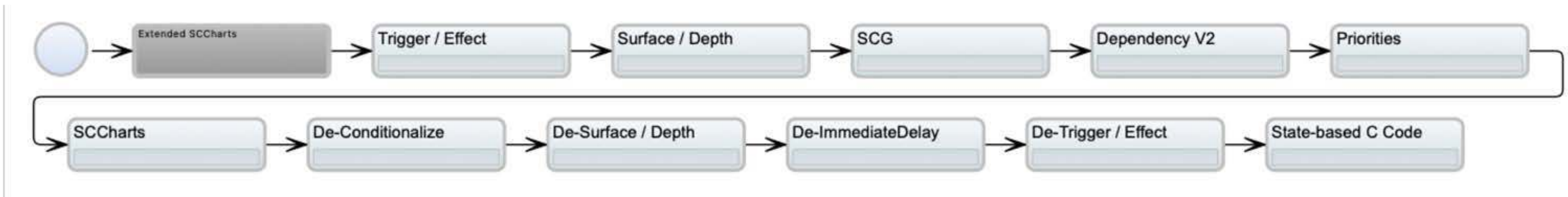
- State functions include outgoing transitions
- Trigger/effects naming
- Transition priorities -> Order




```
void regionR0_stateS0(ContextR0 *context) {
  /* Transition 0: immediate to final state S1
   *   Trigger/Effects: I / O = 1 */
  if (context->io->I) {
    context->io->O = 1;
    context->activeState = S1;
    context->delayedEnabled = 0;
  } else if (context->delayedEnabled) {
    /* Transition 1: delayed to state S2
     *   This is the default transition, the trigger is always true. */
    context->activeState = S2;
    context->delayedEnabled = 0;
  } else {
    // Wait for next tick if no transition was taken.
    context->threadStatus = PAUSING;
  }
}
```

Priority-Based State Machines

1. Transform away extended SCChart features
2. Transform core SCChart down to SCG
3. Schedule, at SCG node granularity
4. Try to recover SCChart structure
5. Translate to C/Java



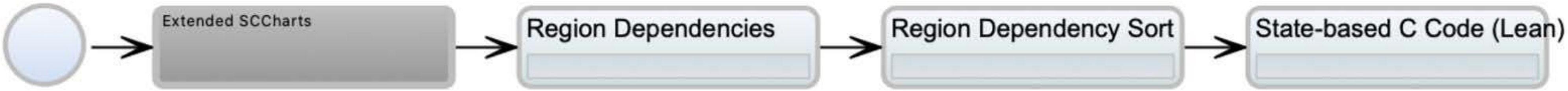
- **Pro:** Can handle arbitrary (static) schedules
- **Con:** May loose some of original structure/naming

„Lean“ State Machines

1. Transform away extended SCChart features
2. Transform core SCChart down to SCG
3. Schedule, at SCG node granularity
4. Try to recover SCChart structure
5. Translate to C/Java

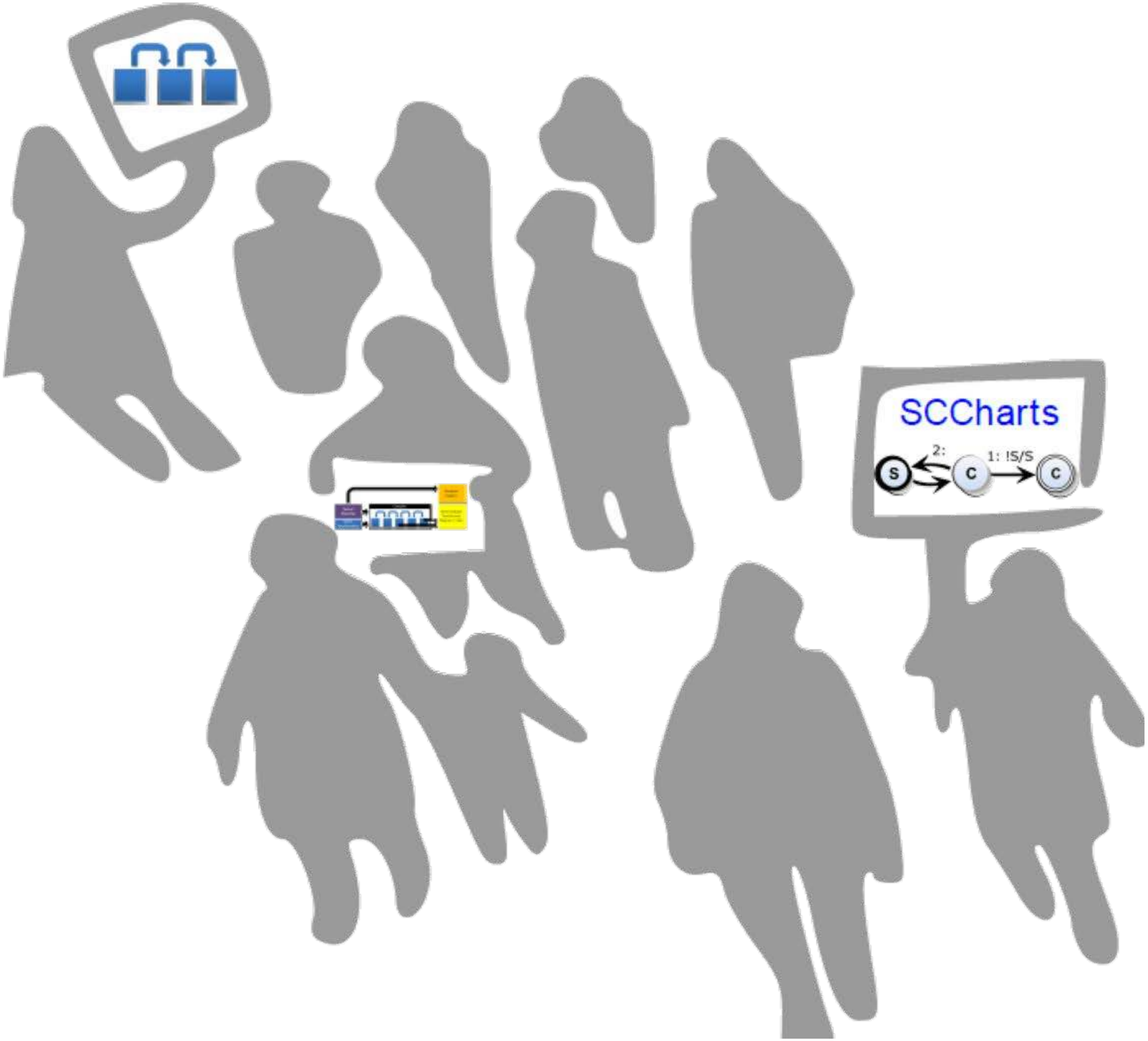
„Lean“ State Machines

1. Transform away extended SCChart features
2. Schedule, at SCChart-region granularity
3. Translate to C/Java



- **Pro:** Compact code, close to original model
- **Con:** Cannot handle back-and-forth communication

Demo



Part III
User Study

Study Goal & Setup

GOAL

Increase readability of SM code

Assumption* : Increased readability essential eases manual verification step

(* to be validated in future work)

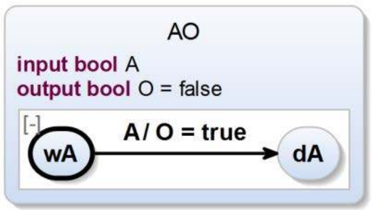
- Compare SM code generation to **multiple** other approaches (netlist & priority)
- Compare versions with and without auto generated comments

Generated code

```
... g0 = _GO;
if(g0){ O = 0;  }
g2 =(PRE_g1);
_cg2 = A;
g1 =(g0||(g2&&(!(_cg2))));
g3 =(g2&&_cg2);
if(g3){ O = 1; }
g5 =(PRE_g4);
g4 =(g3||g5); ...
```

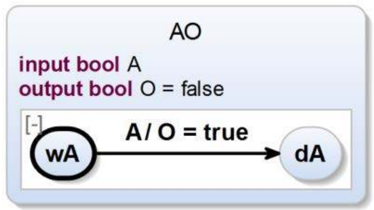
Reverse engineer (task)

Reverse eng. SCChart



- **Confidence**
- **Time**

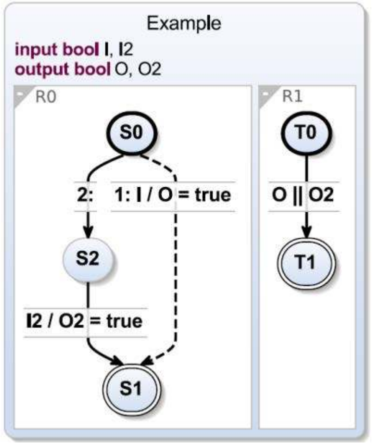
Rate functionality and appearance

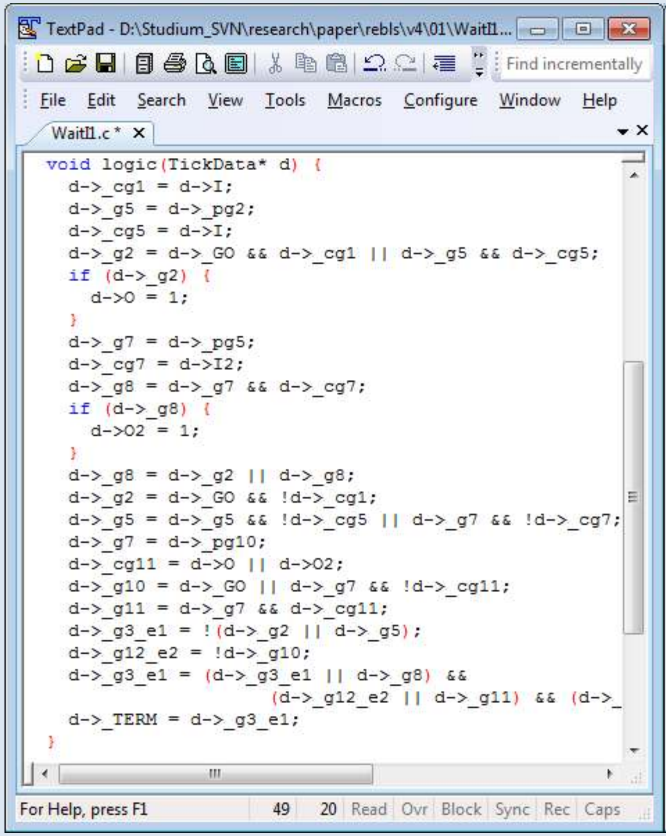


Original SCChart

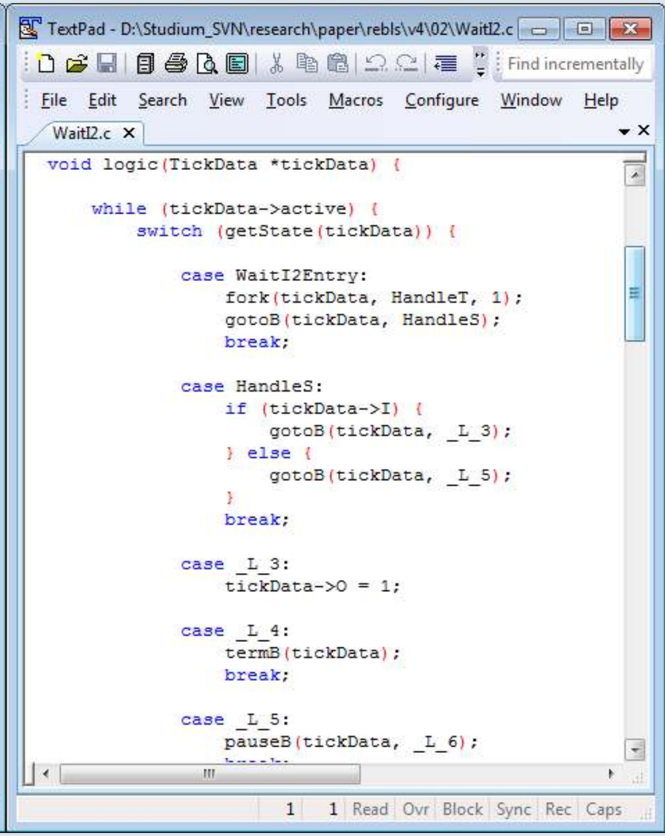
Study

All based on similar SCCharts

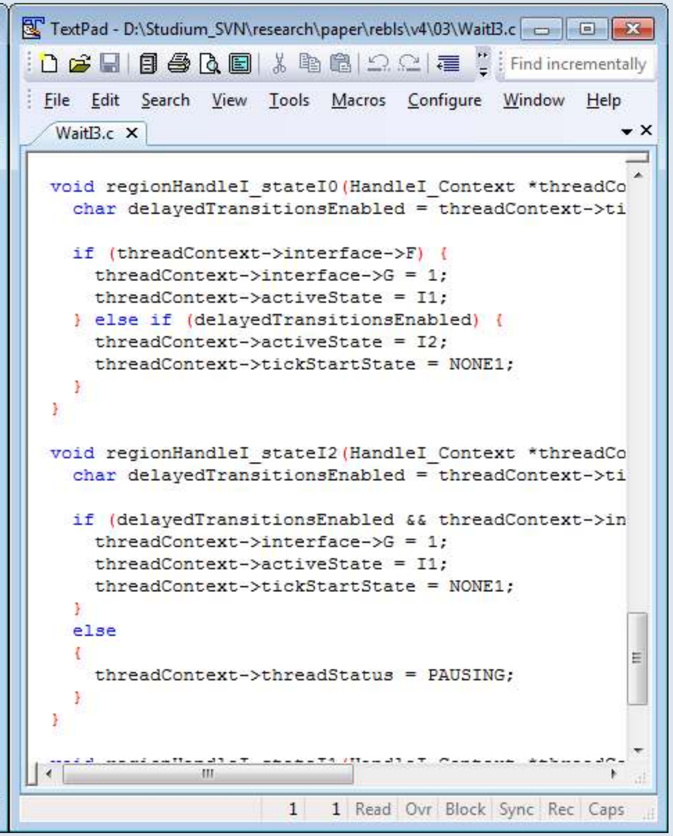




Netlist

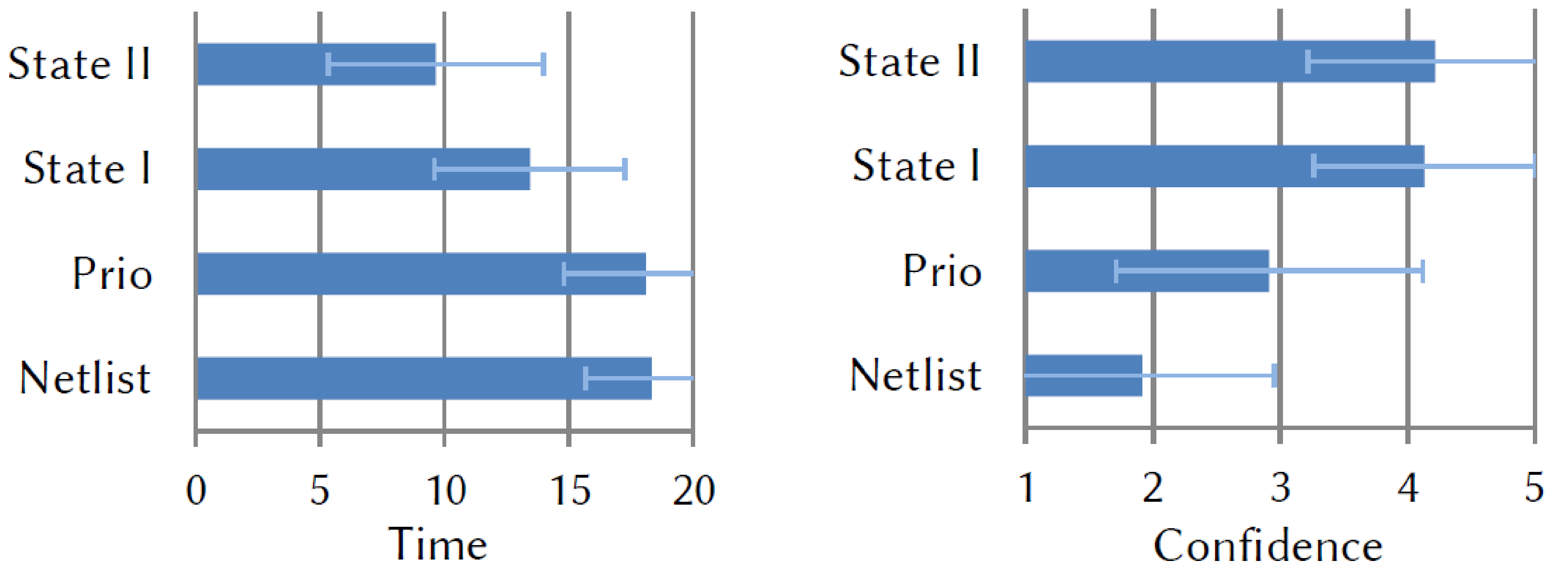


Priority



State-based

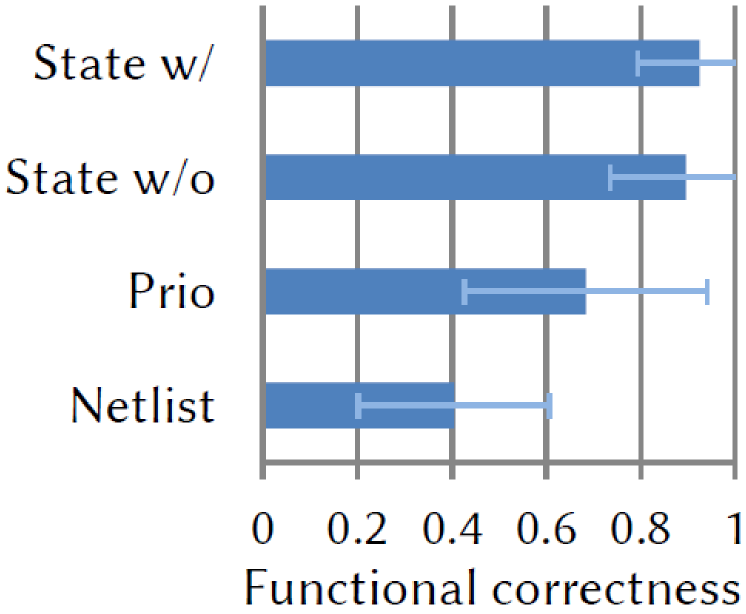
Study Results I

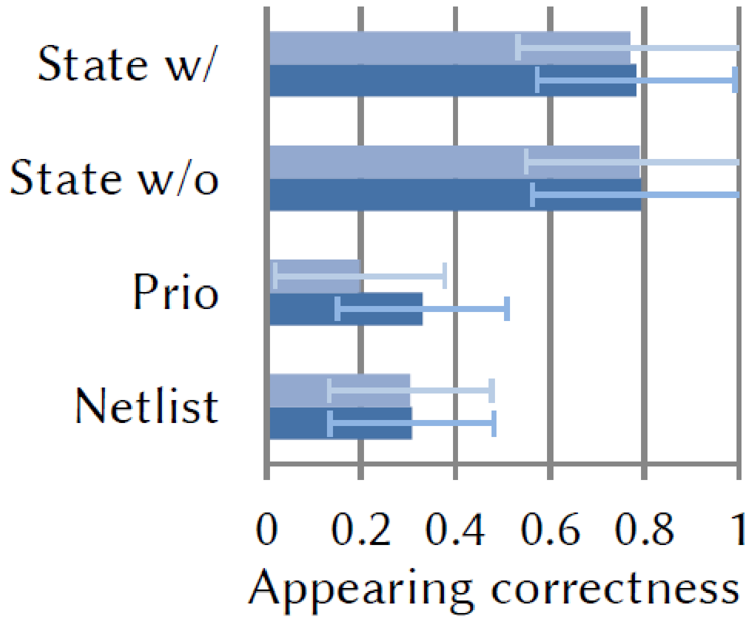


- Experiments aborted after 20 Minutes
- State I and II, two groups get first commented or non-commented version

- State-based: Significantly better in time AND confidence

Study Results II

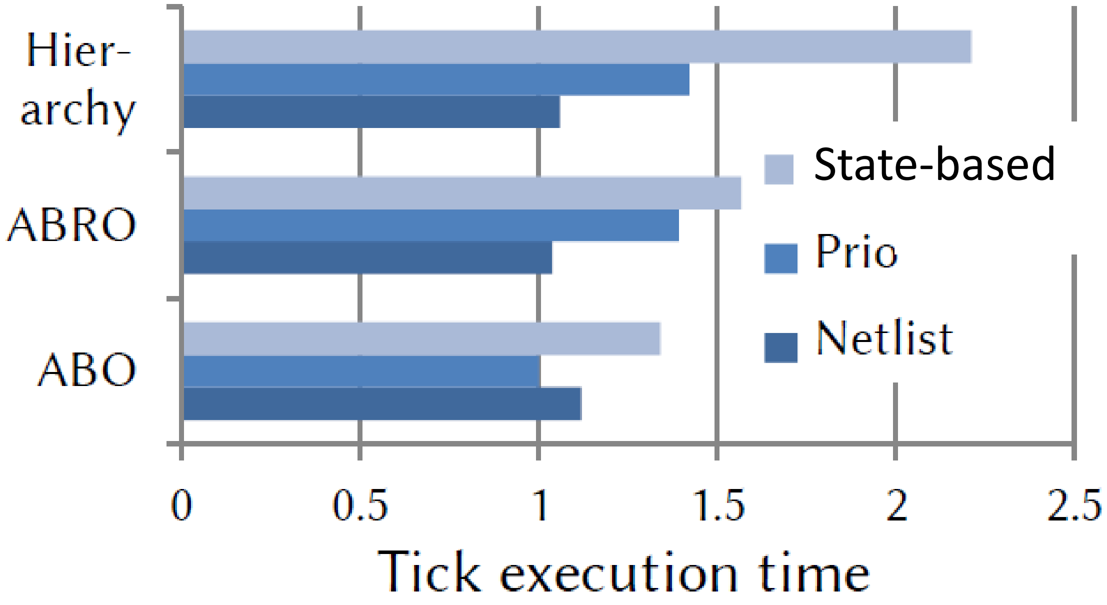




[Dark: Naming, Light:+superflous states/regions]

- Comments helped to increase functional and appearance correctness
- Prio has advantage over netlist-based approach
- State-based: Significantly better in both categories

Study Results III



- Study how benefits affect the execution time
- Result: **Affected, but limited / reasonable weakness (trade-off)**

-> Future Work

To Go Further

- ## Statemachine-Based Compilation (this presentation)
  Christian Motika, Steven Smyth and Reinhard von Hanxleden. *Synthesizing Manually Verifiable Code for Statecharts*. Reactive and Event-based Languages & Systems (**REBLS '18**), Boston, Nov. 2018.
- ## SCCharts Overview
  Reinhard von Hanxleden, Björn Duderstadt, Christian Motika, Steven Smyth, Michael Mendler, Joaquín Aguado, Stephen Mercer, Owen O'Brien. *SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications.*
  Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (**PLDI'14**), Edinburgh, UK, June 2014. ACM.
- ## Interactive Model-based Compilation
  - Christian Motika, Steven Smyth and Reinhard von Hanxleden. *Compiling SCCharts — A case-study on interactive model-based compilation.* **ISoLA 2014**, Corfu, Greece, October 2014
  - Christian Motika. SCCharts – Language and Interactive Incremental *Compilation.* **PhD Thesis**, Kiel University, December 2017
- ## SCCharts Netlist-based Compilation
  Steven Smyth, Christian Motika and Reinhard von Hanxleden. *A Data-Flow Approach for Compiling the Sequentially Constructive Language (SCL). 18. Kolloquium Programmiersprachen und Grundlagen der Programmierung (**KPS 2015**), Pörtschach, Austria, 5-7 October 2015*
- ## OO SCCharts
  Alexander Schulz-Rosengarten, Steven Smyth and Michael Mendler. *Towards Object-Oriented Modeling in SCCharts. Forum on Specification and Design Languages (**FDL 2019**), Southampton, Sep. 2019*
- ## Timed SCCharts
  Alexander Schulz-Rosengarten, Reinhard von Hanxleden, Frédéric Mallet, Robert de Simone and Julien Deantoni. *Timed SCCharts. Forum on Specification and Design Languages (**FDL 2018**), Verona, Sep. 2018*
- ## Hardware Synthesis
  Francesca Rybicki, Steven Smyth, Christian Motika, Alexander Schulz-Rosengarten and Reinhard von Hanxleden. *Interactive Model-Based Compilation Continued – Interactive Incremental Hardware Synthesis for SCCharts.* Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (**ISoLA 2016**), LNCS, 2016.
- ## Underlying Sequentially Constructive Model of Computation
  Reinhard von Hanxleden, Michael Mendler, Joaquín Aguado, Björn Duderstadt, Insa Fuhrmann, Christian Motika, Stephen Mercer, Owen O'Brien, Partha Roop. *Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation.* **ACM Transactions on Embedded Computing Systems**, Special Issue on Applications of Concurrency to System Design, 13(4s):144:1–144:26, July 2014.

Summary

State-based approach:

- Synthesized code preserves structure of model
- Trade-off between code simplicity and generality
- Used in aerospace and railway domain

Future work:

- Further optimizations
- Performance analysis
- Debugging integrated with host code