

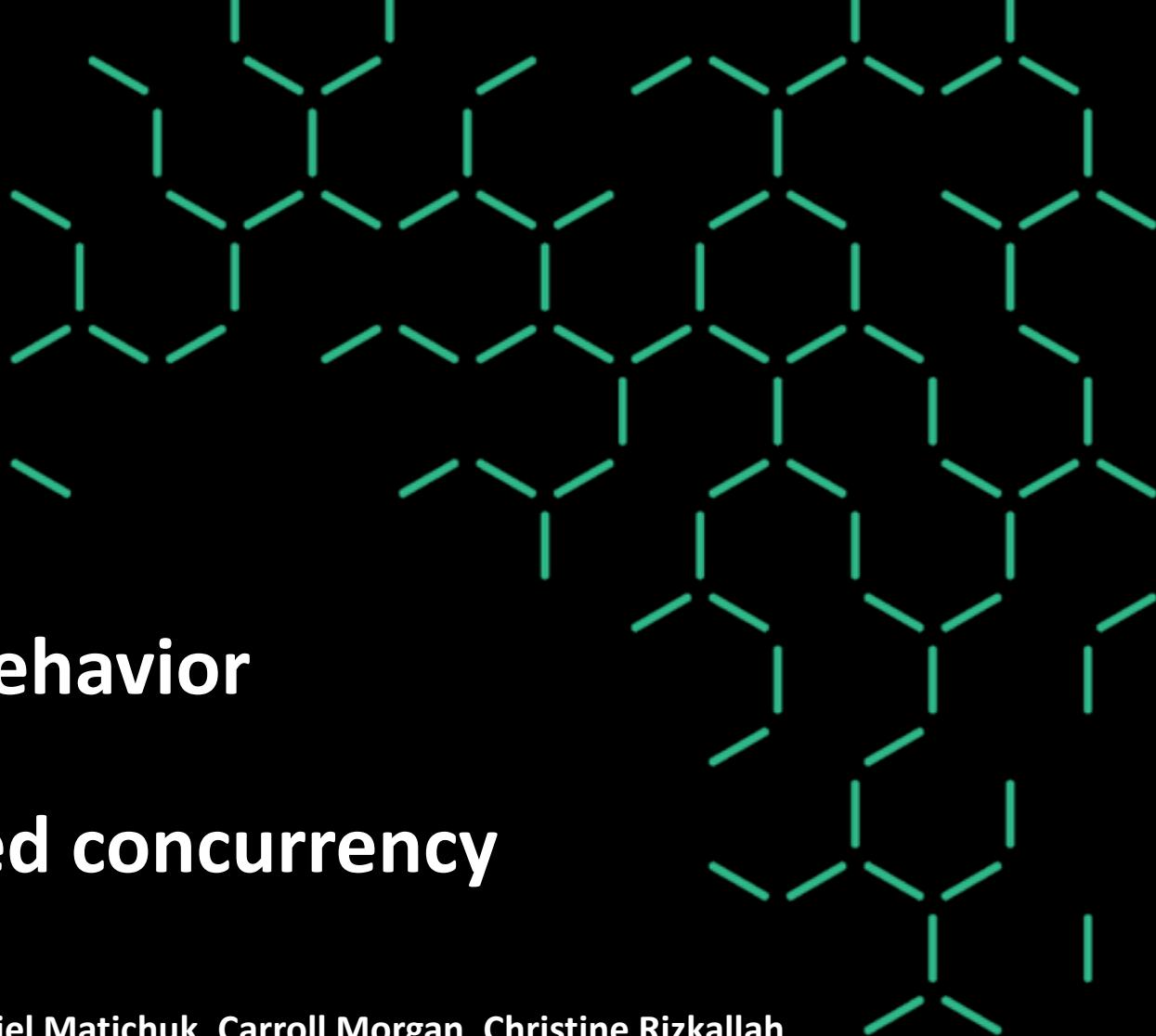
DATA
61

Proof of
OS scheduling behavior
in the presence of
interrupt-induced concurrency

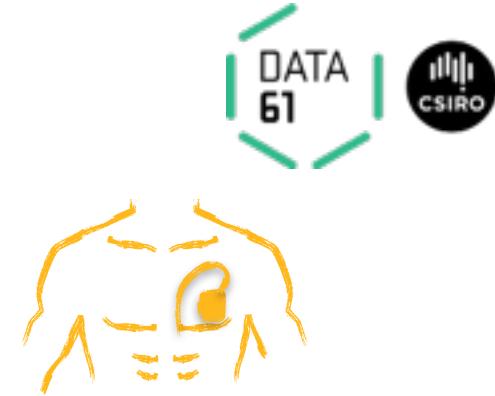
June Andronick, Corey Lewis, Daniel Matichuk, Carroll Morgan, Christine Rizkallah

May 2016

<http://trustworthy.systems/>



Mission and Approach



eChronos

Embedded OS

(*interruptible*, single-core, *preemptive* multi-threaded)

constrained HW
no memory protection
low latency



Simple foundational
concurrency method

Owicki-Gries

conceptually simple
shared-variable reasoning
+ AWAIT-painting

*model-level proof of
scheduling correctness

Concurrency:
shared-memory

racy
controlled



Modern
theorem prover

Isabelle/HOL

machine-checked proofs
automation
+ proof engineering

Embedded OSes — eChronos

(Joint development with Breakaway Consulting)



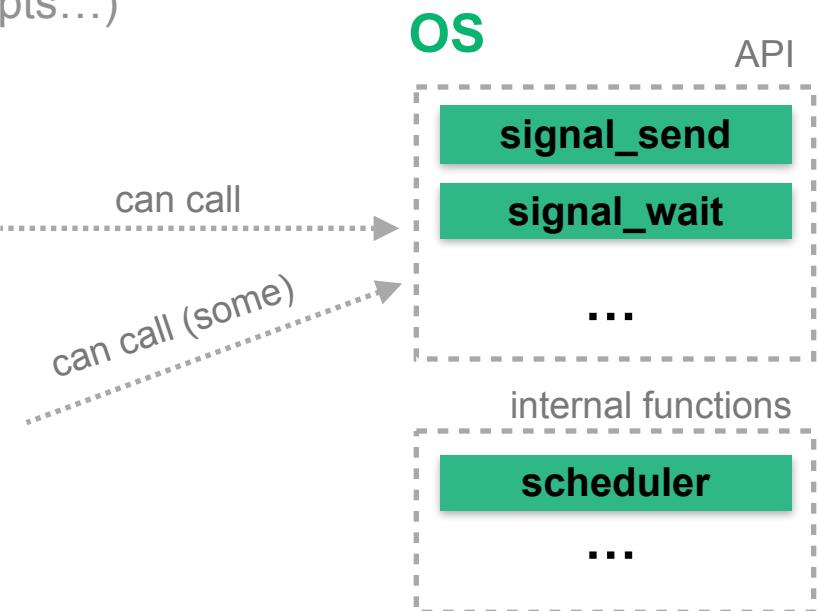
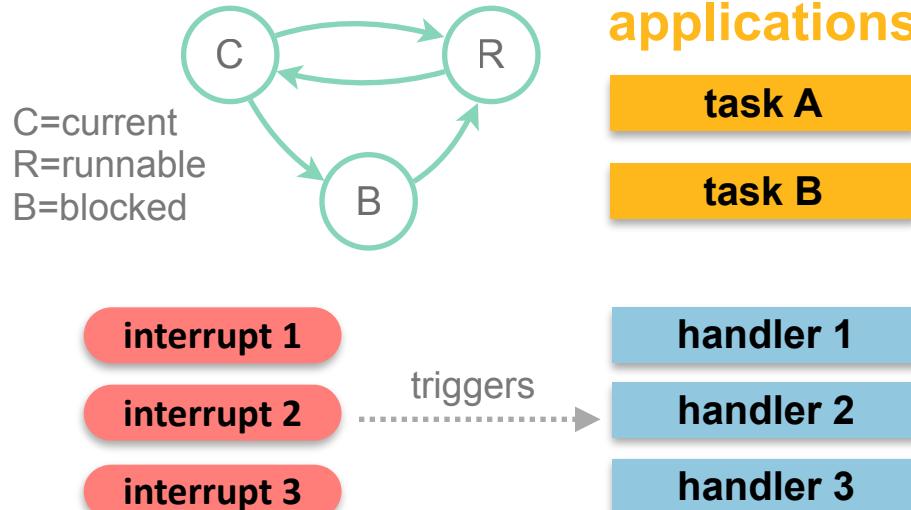
What:

- ▶ small OS library (~500 SLOC)
- ▶ allows applications to be organised in tasks
- ▶ provides library of synchronisation primitives
- ▶ schedules tasks according to some policy (e.g. priorities)

“running task is highest priority runnable task”

Target:

- ▶ preemptive scheduling (not cooperative)
- ▶ ARM platform (nested interrupts...)



Scheduling behavior, informally



task A

task B

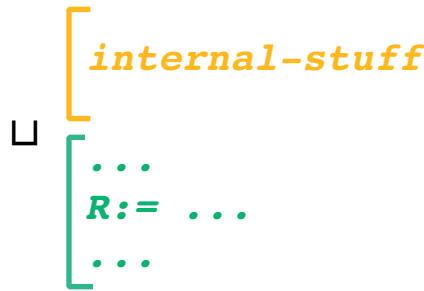
scheduler

handler 1

handler 2

handler 3

The system



$R := \text{handleEvent } R \ E$
 $\text{next} := \text{schedPolicy } R$
 switch next

$E := \dots$
 schedReq
 rfi

Execution can non-deterministically jump to rfi from anywhere else
R and E read and updated without locks

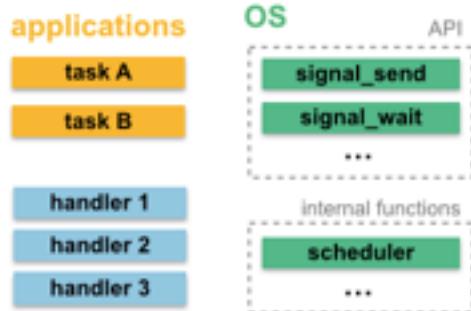
The property

When executing **internal-stuff**, current thread should be $(\text{policy } (\text{update } R \ E))$

Overview



Model of The system



Generic interleaving

```
definition interleaving ... ≡  
A1|| ...|| An|| Sched|| H1|| ...|| Hm
```

HW API

```
definition ITake(X) ≡ ...  
definition IRet(X) ≡ ...
```

eChronos instantiation

```
definition echronos-sys ≡  
interleaving ... || ...
```

Model of The property

"running task is highest priority runnable task"

theorem

$$\vdash_i \{ \text{sched_inv} \} \{\top\} \text{echronos_sys} \{\perp\}$$

definition **sched_inv** ≡

$$(\text{AT} \in \text{User_tasks} \wedge \text{sched} \in \text{EI}) \Rightarrow \text{schedPolicy}(\text{handleEvent E R}) = \text{AT}$$

~3,000 subgoals

done

Interleaving



1

application code is
interruptible and
preemptible

task A

task B

sched

handler 1

handler 2

2

OS code is
interruptible but
not preemptible

*Our model covers all
these interleavings*

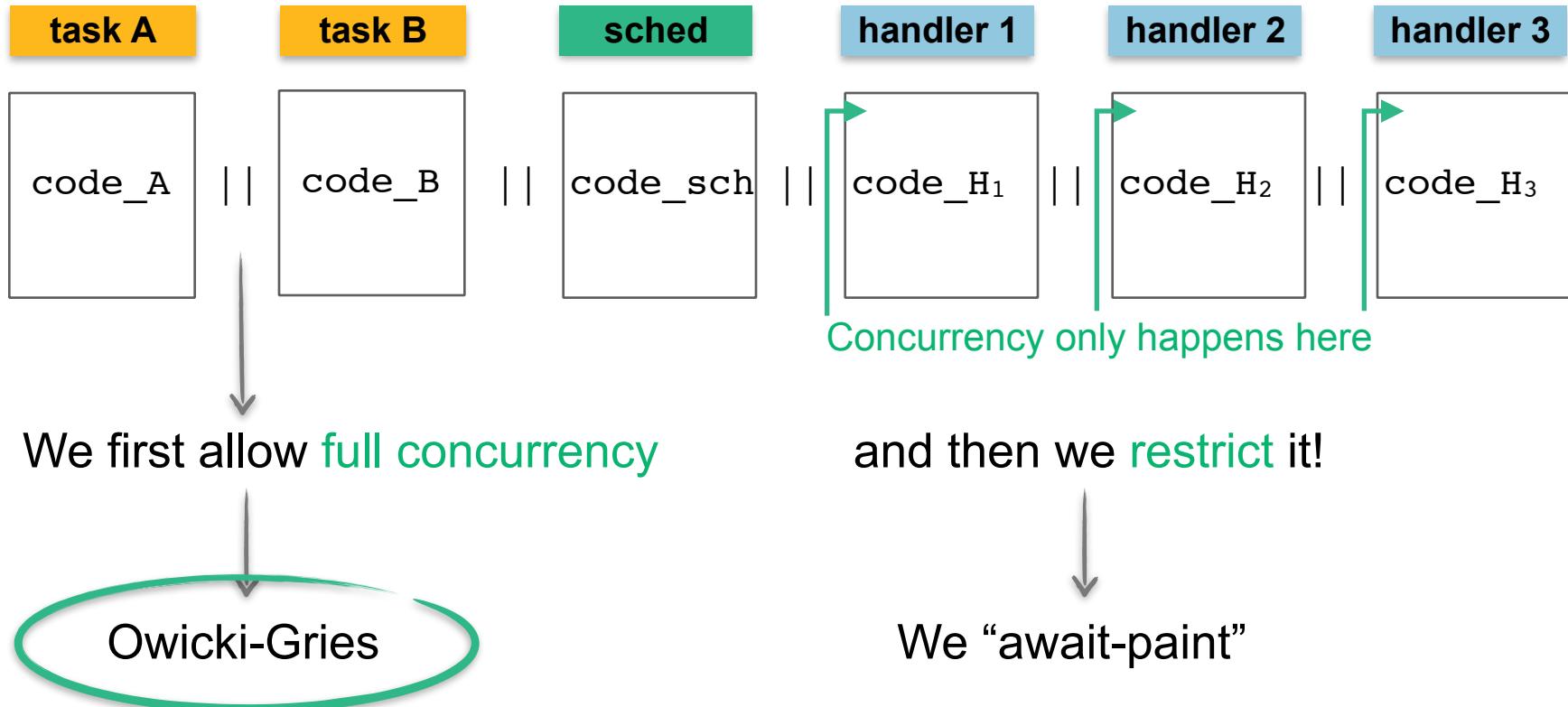
3

scheduler code is
interruptible but
not preemptible

4

handlers are
interruptible
(not preemptible)

Interleaving: model



Owicki-Gries



What: Extension of Hoare logic to shared-variable parallel programs

(Suzanne Owicky and David Gries, 1976)



(Leonor Prensa Nieto, 2002)

Hoare logic:

$\{P\} \quad c \quad \{Q\}$

$c \equiv \begin{array}{|l} x := v \\ | \quad c_1 ; c_2 \\ | \quad \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \text{ FI} \\ | \quad \text{WHILE } b \text{ DO } c \text{ OD} \end{array}$

VCG

OG:

$\begin{array}{|l} \text{cobegin } c_1 || c_2 || \dots || c_n \text{ coend} \\ | \quad \text{AWAIT } b \text{ THEN } c \text{ END} \end{array}$

$\{P\} \quad c_1; \{P_2\}$

$\{P_2\} \quad c_2; \{P_3\}$

$\{P_3\} \quad c_3; \{Q\}$

$\{P\} \quad c_1; \quad \{P'\} \quad c_1';$

$\{P_2\} \quad c_2; \quad || \quad \{P_2'\} \quad c_2';$

$\{P_3\} \quad c_3; \quad \{P_3'\} \quad c_3';$

$\{Q\} \quad \{Q'\}$

Owicki-Gries



What: Extension of Hoare logic to shared-variable parallel programs

(Suzanne Owicky and David Gries, 1976)

Hoare logic:

$\{P\} \quad c \quad \{Q\}$

$c \equiv \begin{array}{|l} x := v \\ | \quad c_1 ; c_2 \\ | \quad \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \text{ FI} \\ | \quad \text{WHILE } b \text{ DO } c \text{ OD} \end{array}$

VCG

OG:

$\begin{array}{|l} \text{cobegin } c_1 || c_2 || \dots || c_n \text{ coend} \\ | \quad \text{AWAIT } b \text{ THEN } c \text{ END} \end{array}$

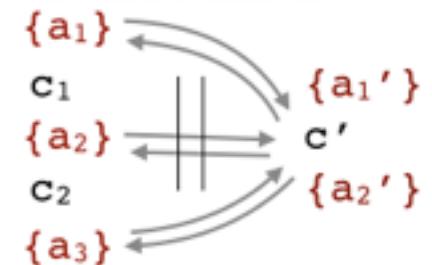
VCG

▶ local correctness

prove each $\{a_i\} c_i \{a_{i+1}\}$

▶ interference freedom

for each assertion a in P ,
and each command c' in P' ,
prove that $\{a \wedge a'\} c' \{a\}$



Owicki-Gries



What: Extension of Hoare logic to shared-variable parallel programs

(Suzanne Owicky and David Gries, 1976)

Hoare logic:

$\{P\} \quad c \quad \{Q\}$

$c \equiv \begin{array}{l} x := v \\ | \quad c_1 ; c_2 \\ | \quad \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \text{ FI} \\ | \quad \text{WHILE } b \text{ DO } c \text{ OD} \end{array}$

VCG

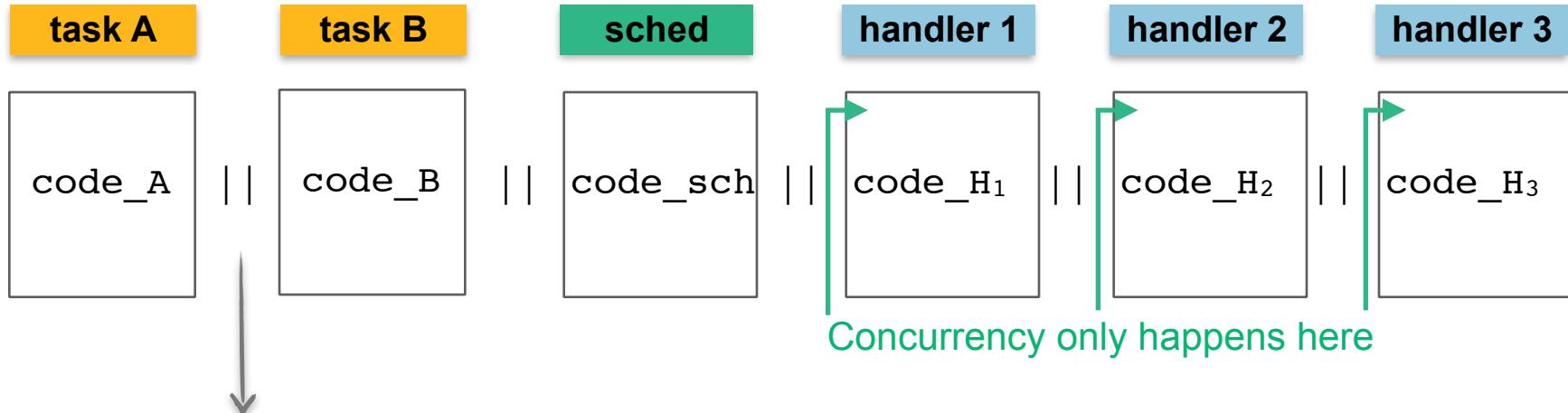
OG:

$\begin{array}{l} \text{cobegin } c_1 || c_2 || \dots || c_n \text{ coend} \\ | \quad \text{AWAIT } b \text{ THEN } c \text{ END} \end{array}$

VCG

- ! requires fully annotated program
- ! quadratic explosion of verification conditions
- ! not compositional

Interleaving: model



We first allow **full concurrency**

Owicki-Gries

and then we **restrict** it!

We “await-paint”

Await Painting



task A task B

code_A

||

code_B

- We introduce: Variable **AT** (Active Task)
- We “**AWAIT-paint**” **almost** all statements:
- Interleaving is controlled by hardware operations:
taking interrupts, return-from-interrupt, context switch



AT:=task_id;

AWAIT AT=A THEN {P₁}a₁;
AWAIT AT=A THEN {P₂}a₂;
AWAIT AT=A THEN {P₃}a₃;

||

AWAIT AT=B THEN {R₁}b₁;
AWAIT AT=B THEN {R₂}b₂;
AWAIT AT=B THEN {R₃}b₃;

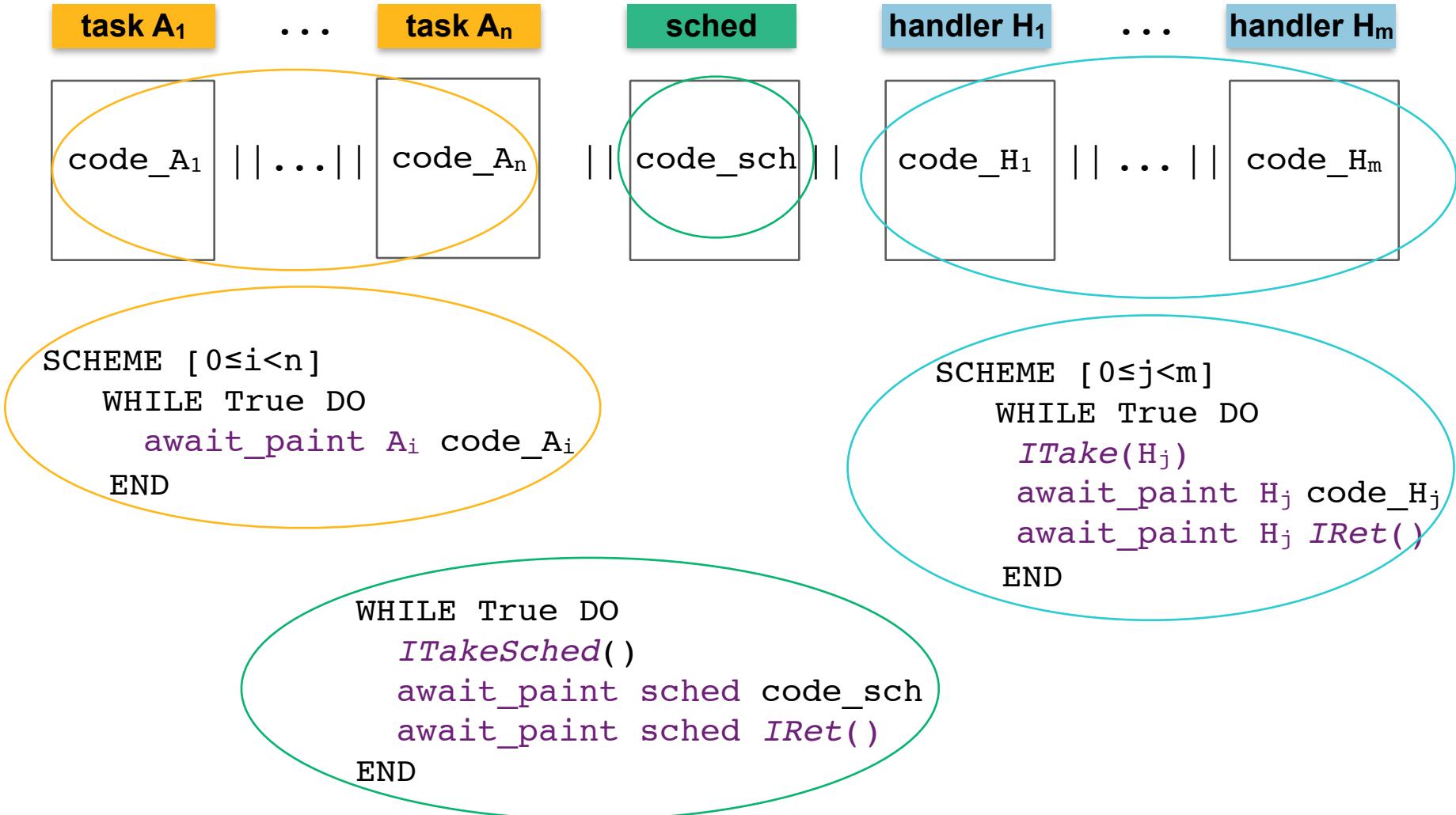
await_paint A code_A;

||

await_paint B code_B;

SCHEME [0≤i<n]
 WHILE True DO
 await_paint A_i code_A_i
 END

Interleaving: model



Full model

Model variables

AT, EI, ATStack



Generic interrupt-induced interleaving

```
definition interleaving code_Ai code_sch code_Hj ≡
  SCHEME [0≤i<n]
    WHILE True DO
      await_paint Ai code_Ai
    END || WHILE True DO
      ITakeSched()
      await_paint sched code_sch
      await_paint sched IRet()
    END || SCHEME [0≤j<m]
      WHILE True DO
        ITake(Hj)
        await_paint Hj code_Hj
        await_paint Hj IRet()
      END
```

HW API

```
definition ITake(X) ≡ ...
definition IRet(X) ≡ ...
definition ITakeSched() ≡ ...
```

```
definition Int-Disable(X) ≡ EI := EI - X
definition Int-Enable(X) ≡ EI := EI ∪ X
```

Intuitively: $\text{ITake } H_j \equiv \text{AT} := H_j$

More precisely: $\text{ITake } H_j \equiv$
 $\text{AWAIT } H_j \in \text{EI} - \text{ATStack}$
 $\wedge H_j \in \text{interrupt-policy (AT)}$
 $\text{THEN } \text{ATStack} := \text{AT} \# \text{ATStack}; \text{AT} := H_j$
 END

Full model

Model variables

AT, EI, ATStack



Generic interrupt-induced interleaving

```
definition interleaving code_Ai code_sch code_Hj ≡
  SCHEME [0≤i<n]
    WHILE True DO
      await_paint Ai code_Ai
    END || WHILE True DO
      ITakeSched()
      await_paint sched code_sch
      await_paint sched IRet()
    END || SCHEME [0≤j<m]
      WHILE True DO
        ITake(Hj)
        await_paint Hj code_Hj
        await_paint Hj IRet()
      END
```

HW API

```
definition ITake(X) ≡ ...
definition IRet(X) ≡ ...
definition ITakeSched() ≡ ...
```

```
definition Int-Disable(X) ≡ EI := EI - X
definition Int-Enable(X) ≡ EI := EI ∪ X
```

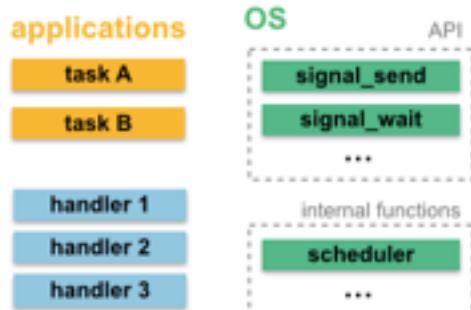
eChronos instantiation (~200 lines of parallel program)

```
definition echronos-sys ≡ interleaving echr_Ai echr_sch echr_Hj
  echr_Ai ≡
    internal-stuff
    Int-Disable(sched)
    OS function call
    Int-Enable(sched)
  echr_sch ≡
    R := handleEvent R E
    next := schedPolicy R
    switch next
  echr_Hj ≡
    E := ...
    schedReq
    rfi
```

Overview



Model of The system



Generic interleaving

```
definition interleaving ... ≡  
A1|| ...|| An|| Sched|| H1|| ...|| Hm
```

HW API

```
definition ITake(X) ≡ ...  
definition IRet(X) ≡ ...
```

eChronos instantiation

```
definition echronos-sys ≡  
interleaving ... ... ...
```

Model of The property

"running task is highest priority runnable task"



theorem

$$\vdash_i \{ \text{sched_inv} \} \{\top\} \text{echronos_sys} \{\perp\}$$

definition **sched_inv** ≡

$$(AT \in \text{User_tasks} \wedge \text{sched} \in EI) \Rightarrow \text{schedPolicy}(\text{handleEvent } E R) = AT$$

~3,000 subgoals



done

eChronos scheduling correctness



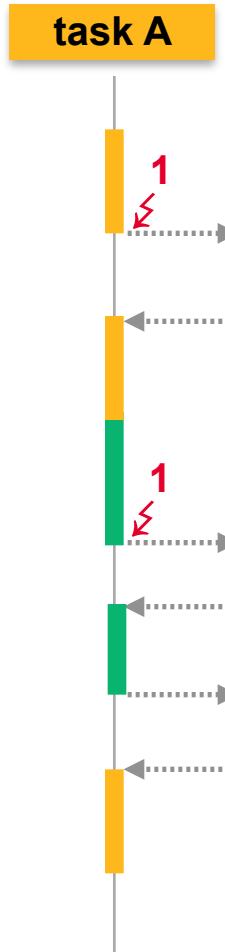
English: ***"the running task is the highest-priority runnable task"***

Formally:

theorem $\vdash_i \{ \text{sched_inv} \} \{\top\} \text{echronos_sys} \{\perp\}$

Where:

definition `sched_inv ≡`
 $(AT \in \text{User_tasks} \wedge \text{sched} \in EI) \Rightarrow$
`schedPolicy (handleEvent E R) = AT`



Proof framework



OG gives us derivation rules:

 $\vdash p \ c \ q$

(Leonor Prena
Nieto, 2002)

But here it needs an invariant :

 $\vdash_i I \ p \ c \ q$ 

We add support for **assumed** invariants:

 $I' \ \vdash_i I \ p \ c \ q$

We add support for proving invariants **compositionally**:

$$\frac{I' \parallel_{-i} I \ p \ c \ q \quad \parallel_{-i} I' p' c' q' \quad \text{merge-prog-com } c \ c' = \text{Some } c''}{\parallel_{-i} (I' \cap I) (p \cap p') c'' (q \cap q')}$$

Proof overview



Goal:

```
theorem ⊢ᵢ {sched_inv} {⊤} echronos_sys {⊥}
```

Using compositionally lemma:

```
lemma {helper_invs} ⊢ᵢ {sched_inv} {⊤} echronos_sys {⊥}
```

```
lemma ⊢ᵢ {helper_invs} {⊤} echronos_sys {⊥}
```

Where:

```
definition helper_invs ≡ I₁ ∧ ... ∧ I₉
```

Examples:

```
definition last-stack-inv ≡ last (AT#ATStack) ∈ User_tasks
```

Proof engineering!



Goal:

```
lemma {helper_invs} ⊢ᵢ {sched_inv} {⊤} echronos_sys {⊥}
```

```
lemma ⊢ᵢ {helper_invs} {⊤} echronos_sys {⊥}
```



Proof engineering!



Goal:

```
lemma {helper_invs} ⊢i {sched_inv} {⊤} echronos_sys {⊥}
```

```
lemma ⊢i {helper_invs} {⊤} echronos_sys {⊥}
```

VCG
~3,000 subgoals

(~90s)

custom tactic

tweak the tactic

0 subgoals ✓ (~1h) (each time you change annotations!)

apply (tactic T)

run T' in parallel on all subgoals

(clean-up – **not** using simp)

subst or simp only with specific rules

(prework – **not** using simp)

forward rules, no simp, only by assumption

try T1 orelse

simp with some only...

try T2 orelse

blast

try T3 orelse

clar_simps

...

fastforce

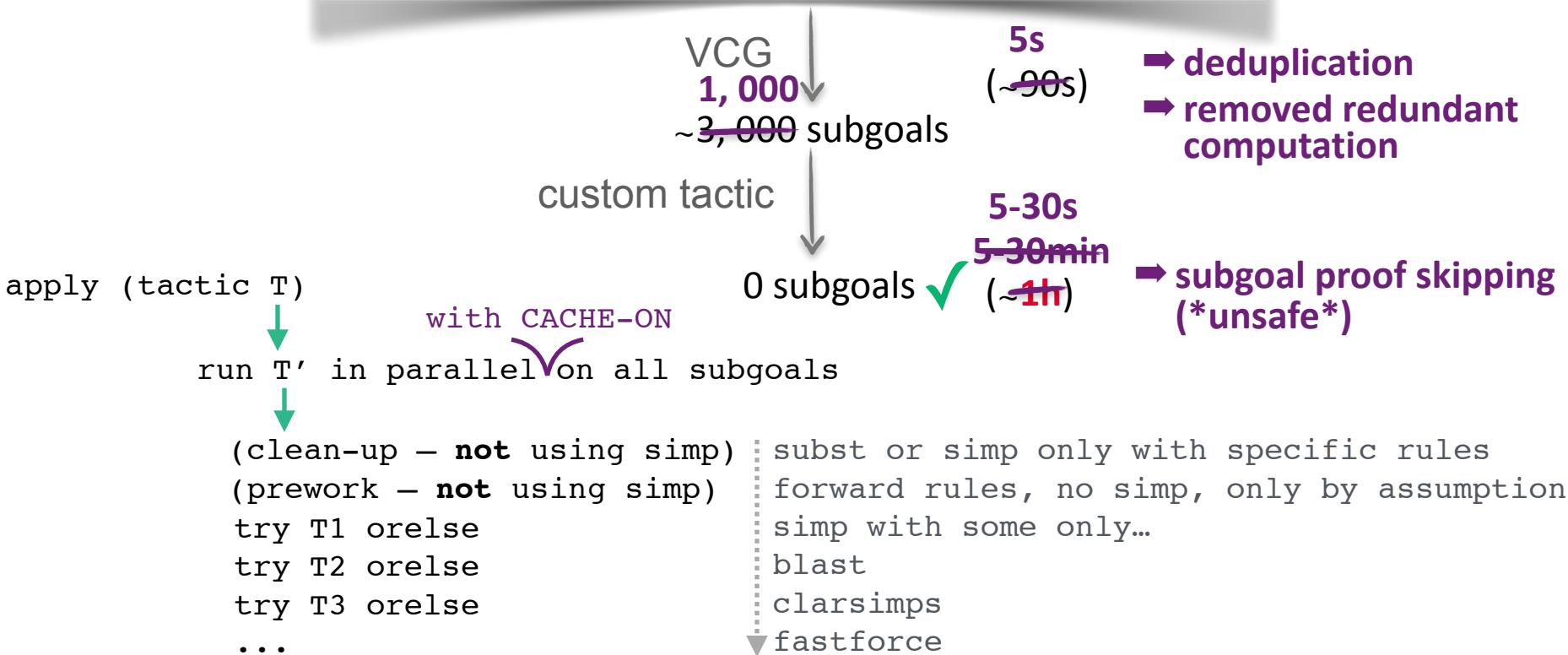
Proof engineering!



Goal:

```
lemma {helper_invs} ⊢i {sched_inv} {⊤} echronos_sys {⊥}
```

```
lemma ⊢i {helper_invs} {⊤} echronos_sys {⊥}
```

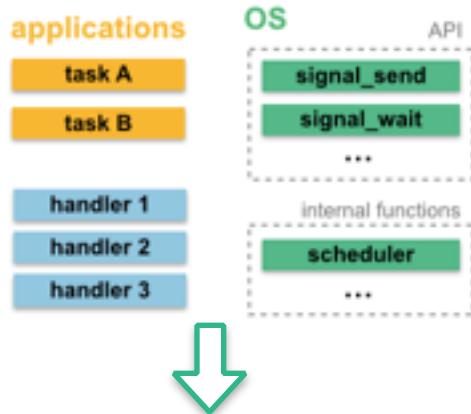


Summary



Model of

The system



Generic interleaving

```
definition interleaving ... ≡  
A1|| ...|| An|| Sched|| H1|| ...|| Hm
```

HW API

```
definition ITake(X) ≡ ...  
definition IRet(X) ≡ ...
```

eChronos instantiation

```
definition echronos-sys ≡  
interleaving ... ... ...
```

Model of

The property

"running task is highest priority runnable task"



theorem

$$\Vdash_i \{ \text{sched_inv} \} \{ \top \} \text{echronos_sys} \{ \perp \}$$

definition **sched_inv** ≡

$$(AT \in \text{User_tasks} \wedge \text{sched} \in EI) \Rightarrow \text{schedPolicy}(\text{handleEvent } E R) = AT$$


~3,000 subgoals



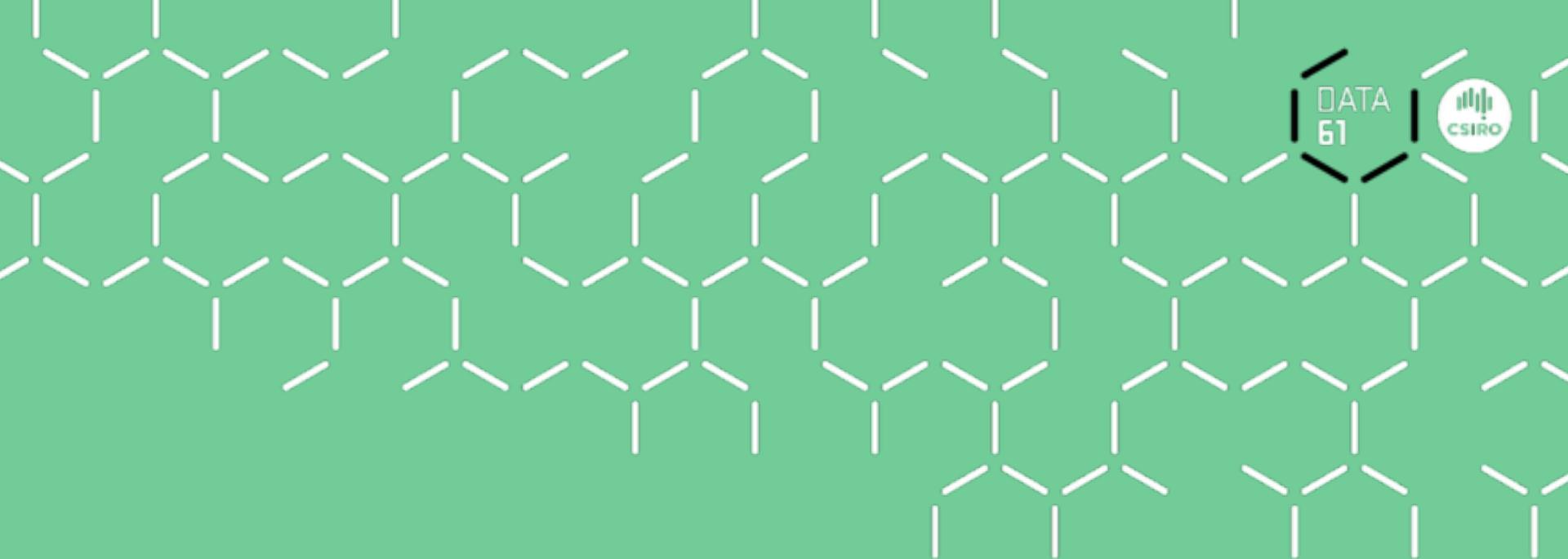
done

Next?



- down to the code!
- link to more detailed API Spec

```
definition echronos-sys ≡ interleaving echr_Ai echr_sch echr_Hj  
          echr_Ai ≒  
          [ internal-stuff  
            Int-Disable(sched)  
            OS function call  
            Int-Enable(sched) ]  
          echr_sch ≒  
          [ R := handleEvent R E  
            next := schedPolicy R  
            switch next ]  
          echr_Hj ≒  
          [ E := ...  
            schedReq  
            rfi ]
```



DATA
61



Thank you