Time Safety Check

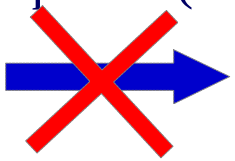
Tom Henzinger, Christoph Kirsch
Slobodan Matic

Time safety: Motivation

❑ RTOS

Traditional ***schedulability*** test checks for a feasible schedule

- for a given scheduling algorithm
- for given task assumptions (task models)

schedulability  determinism

❑ Embedded machine on top of RTOS

- interaction with the environment is completely separated from task execution
- E code is ***time safe*** if each task *completes before its outputs are read* (*and new inputs are provided*) - platform dependent property
- if E code is triggered by environment

time safety  determinism

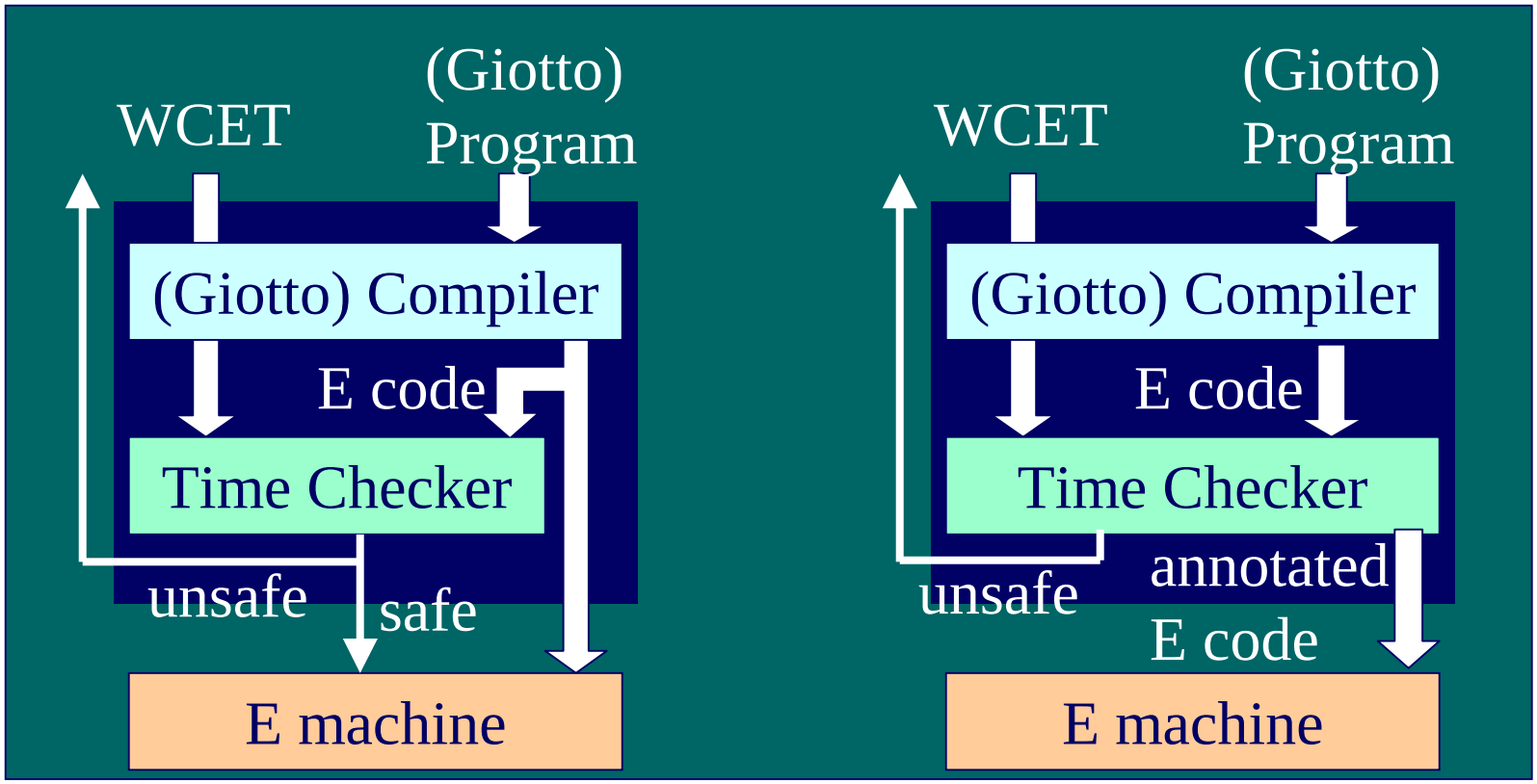
both timing and value predictability ensured

Approach

- Deadlines implicitly specified in the E code:

  Operate on E code directly, without extracting task set model

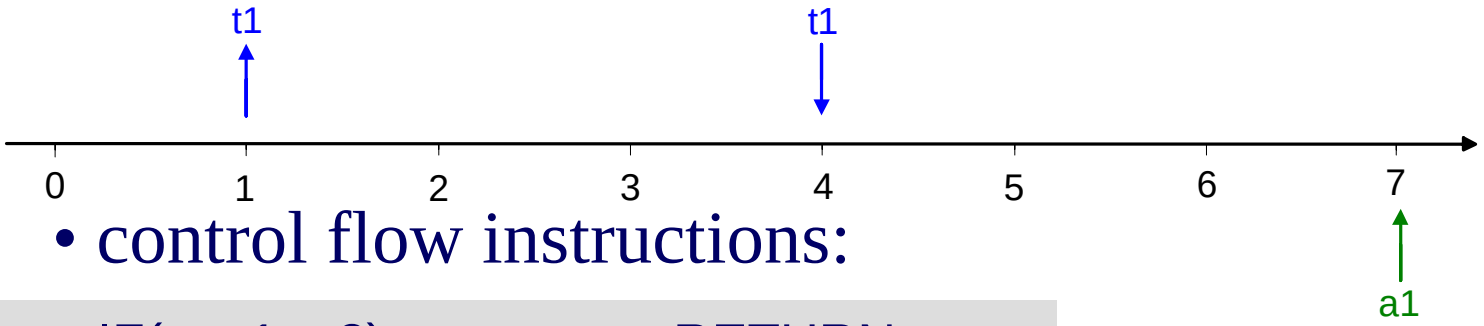
- more program analysis than schedulability test



Problem statement

- E code instructions (synchronous computation)
  - core instructions:

SCHEDULE(t1) CALL(d1) FUTURE(g,a1)



  - control flow instructions:

IF(c, a1, a2) RETURN

- time-triggered E code

  g(n): clk'=clk+n

- platform dependent parameters of the algorithm
  - *WCET* for each task
  - EDF scheduling scheme

Algorithm

- develop the schedule using EDF algorithm to see if all deadlines are met
- starting from initial trigger simulate execution of E code:

  at each step of the algorithm scan the E code with label *a* for

  - CALL and SCHEDULE **completed**(*a*)
  - SCHEDULE **scheduled**(*a*)
  - FUTURE **future**(*a*)

- each step can be considered as state transition:

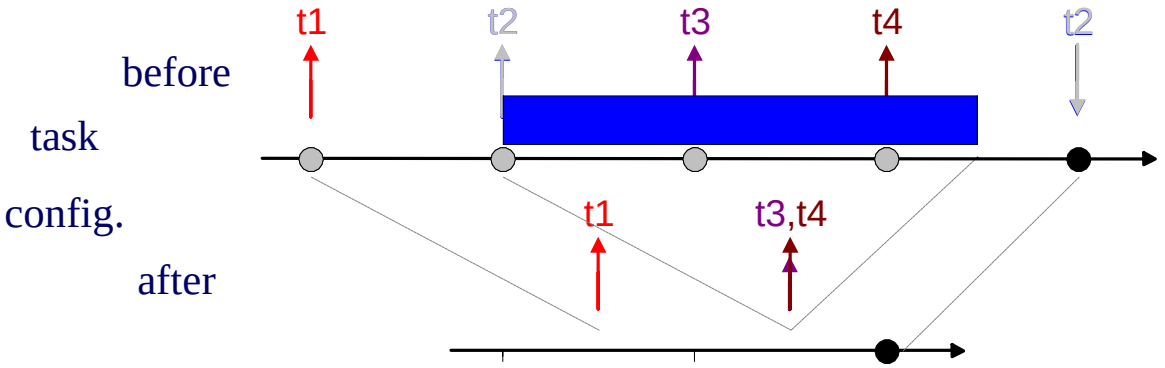
  what's the state? **(task_config, trigger_config)**

  simple numeric test for time safety of a configuration

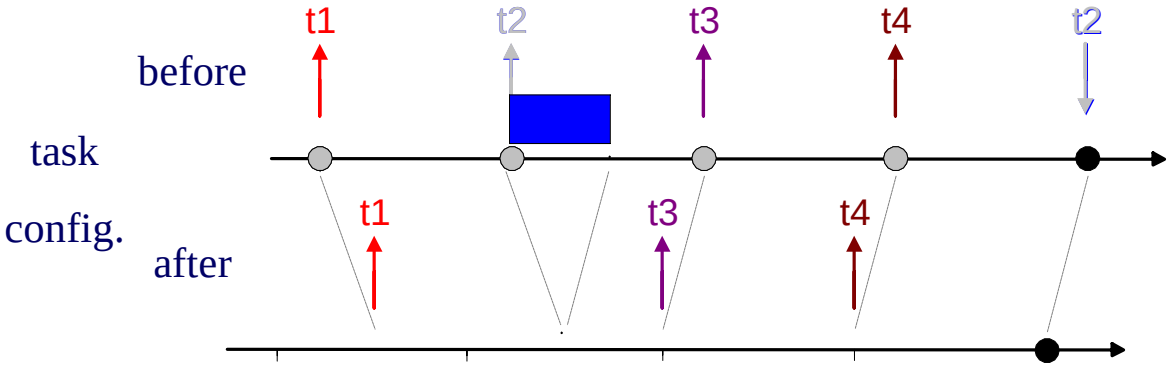
- simulate until new state already visited or time violation occured
- run the algorithm for the each assignment of conditional jumps

Basic state transition

WCET (t2) = 2.5



WCET (t2) = 0.5

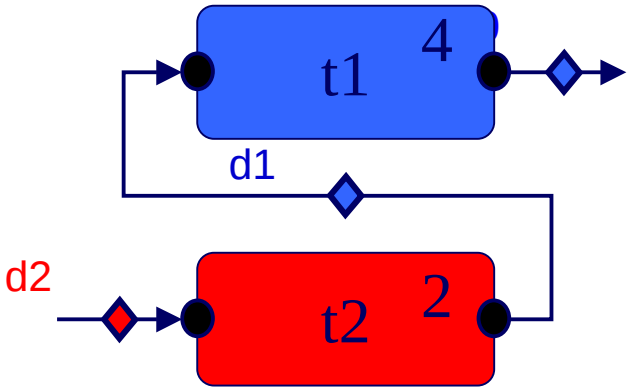


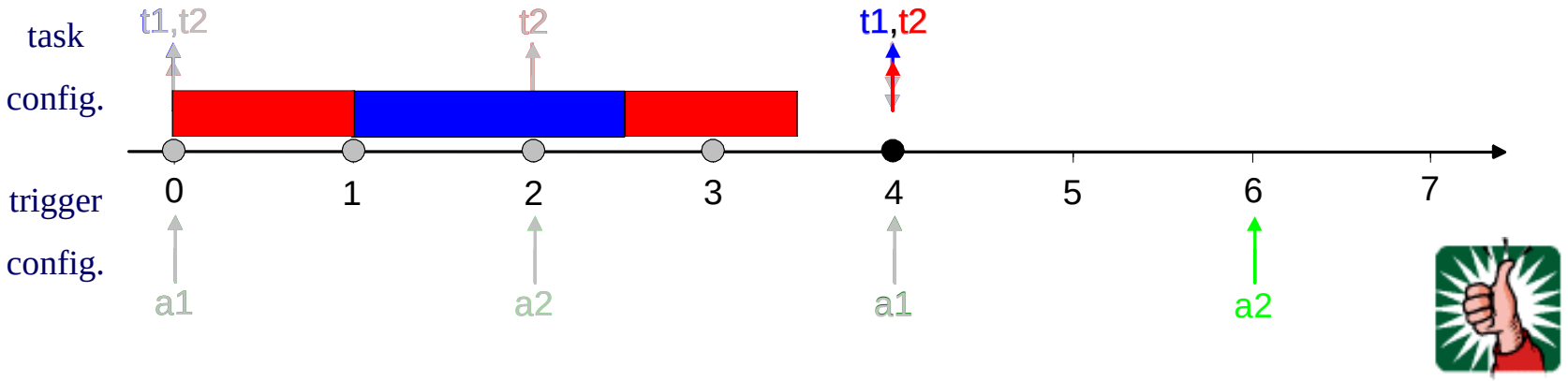
Two-task example

$Ao = \{(a1,0)\}$

$WCET\ (t1)=1.5,\ WCET\ (t2)=1$

a1:	a2:
CALL(d0)	CALL(d2)
CALL(d1)	SCHEDULE(t2)
SCHEDULE(t1)	FUTURE(g(2),a1)
CALL(d2)	RETURN
SCHEDULE(t2)	
FUTURE(g(2),a2)	
RETURN	





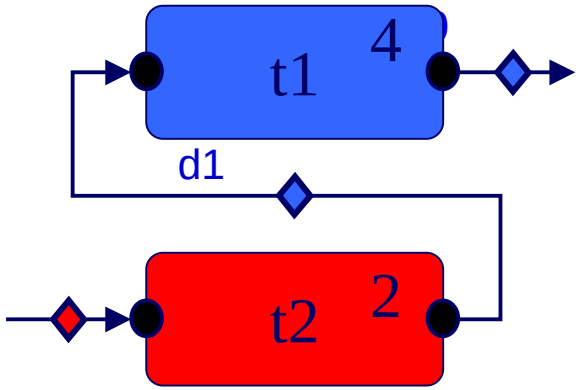
Two-task example (2)

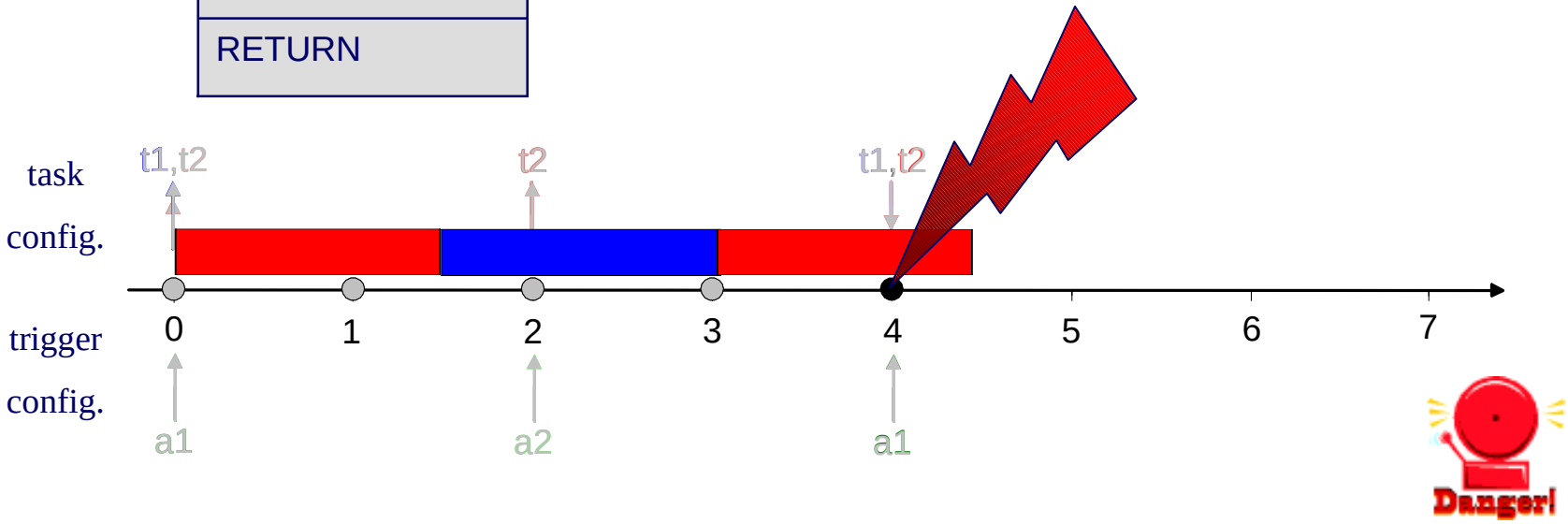
$Ao = \{(a1,0)\}$

$WCET\ (t1) = 1.5,\ WCET\ (t2) = 1.5$

a1:
CALL(d0)
CALL(d1)
SCHEDULE(t1)
CALL(d2)
SCHEDULE(t2)
FUTURE(g(2),a2)
RETURN

a2:
CALL(d2)
SCHEDULE(t2)
FUTURE(g(2),a1)
RETURN





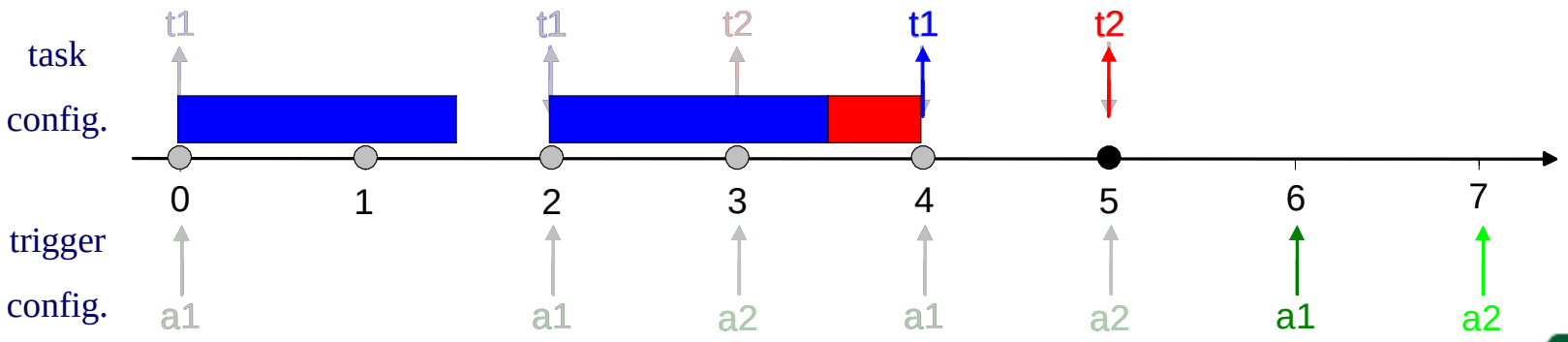
Phase-shifted task set

$Ao = \{(a1,0),(a2,3)\}$

$WCET\ (t1) = 1.5,\ WCET\ (t2) = 0.5$

a1:	SCHEDULE(t1)
	FUTURE(g(2),a1)
	RETURN

a2:	SCHEDULE(t2)
	FUTURE(g(2),a2)
	RETURN



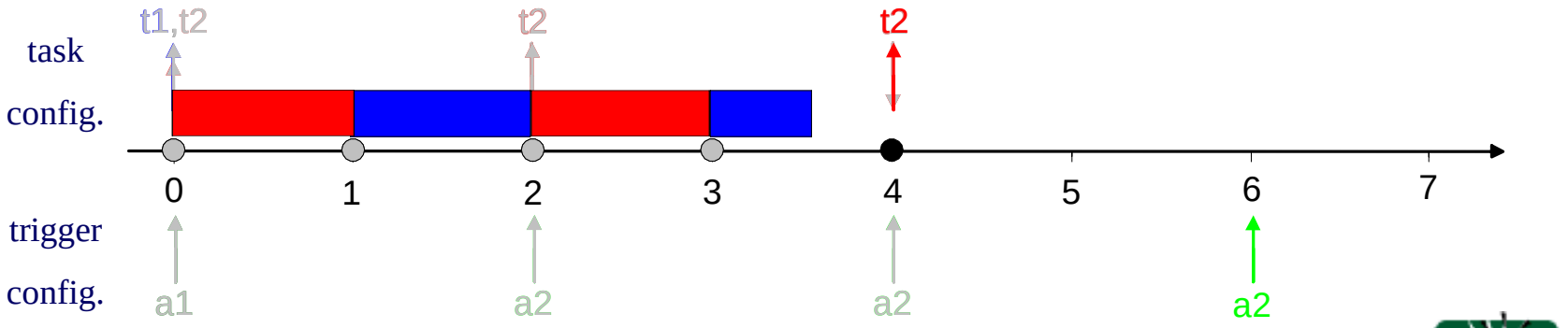
Non-EDF schedule

$Ao = \{(a1,0)\}$

$WCET\ (t1) = 1.5,\ WCET\ (t2) = 1$

a1:	SCHEDULE(t1)
	SCHEDULE(t2)
	FUTURE(g(2),a2)
	RETURN

a2:	SCHEDULE(t2)
	FUTURE(g(2),a2)
	RETURN



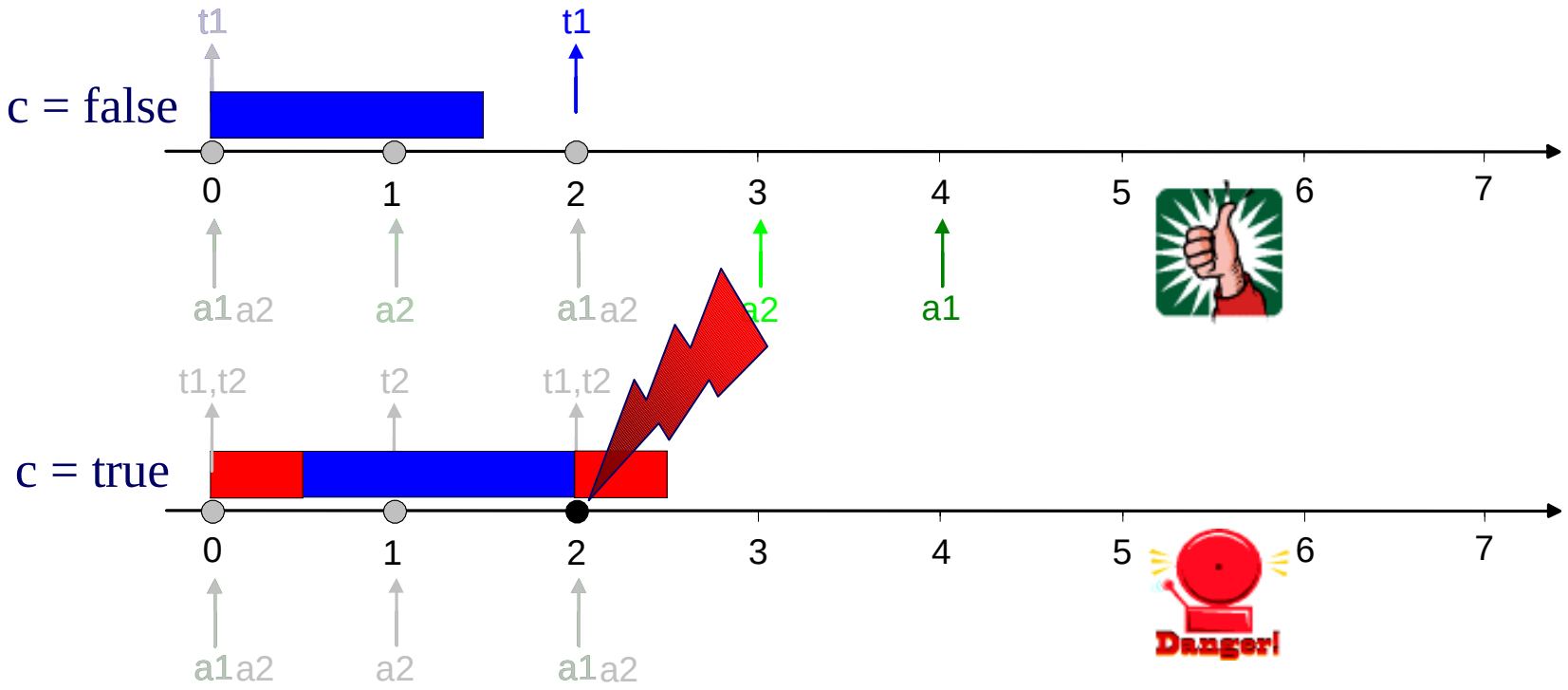
IF instruction example

Ao = {(a1,0),(a2,0)}

WCET (t1)=1.5, *WCET* (t2)=0.5

a1:	SCHEDULE(t1)
	FUTURE(g(2),a1)
	RETURN

a2:	IF(c,a2',a2'')
a2':	SCHEDULE(t2)
a2'':	FUTURE(g(1),a2)
	RETURN



Pseudo code

Algorithm 1 time_safety_check$(\mathcal{A}, \mathcal{T})$

```
if (A, T) ∉ H then
  H ← H ∪ {(A, T)}
  C ← enum_cond_assignment(number of if instructions)
  while C ≠ ∅ do
    C ← C ∪ {C}
    if is_safe(A, T, C) = false then
      time safety violation!
    end if
    T_n ← execute_tasks(A, T, C)
    T_n ← schedule_tasks(A, T_n, C)
    A_n ← future(A, C)
    time_safety_check(A_n, T_n)
  end while
end if
```

Algorithm 2 is_safe$(\mathcal{A}, \mathcal{T}, C)$

```
T_c ← completed(A, C)
T_r ← T, ω ← 0
while T_r ≠ ⟨⟩ do
  T_r ← ⟨(T, τ)⟩ ∘ T_r
  ω ← ω + Σ_{t∈T∩T_c} wcet(t) − τ
  ω ← max(0, ω)
end while
return ω = 0
```

Algorithm 3 execute_tasks$(\mathcal{A}, \mathcal{T}, C)$

```
T_c ← completed(A, C)
T ← execute_completed_tasks(T_c, T)
T ← merge_tasks(T)
T ← execute_tasks_early(T)
return T
```

Algorithm 4 execute_completed_tasks$(\mathcal{T}_c, \mathcal{T})$

```
T_i ← ⟨⟩, T_r ← T
while T_r ≠ ⟨⟩ do
  T_r ← ⟨(T, τ)⟩ ∘ T_r
  τ ← τ − Σ_{t∈T∩T_c} wcet(t)
  T ← T \ T_c
  T_i ← T_i ∘ ⟨(T, τ)⟩
end while
return T_i
```

Algorithm 5 merge_tasks$(\mathcal{T})$

```
T_i ← ⟨⟩, T_r ← T
while T_r ≠ ⟨⟩ do
  T_r ← ⟨(T, τ)⟩ ∘ T_r
  if τ ≤ 0 ∧ T_r ≠ ⟨⟩ then
    T_r ← ⟨(D, δ)⟩ ∘ T_r
    T_r ← ⟨(D ∪ T, δ + τ)⟩ ∘ T_r
  else if τ > 0 ∧ T = ∅ ∧ T_i ≠ ⟨⟩ then
    T_i ← T_i ∘ ⟨(D, δ)⟩
    T_i ← T_i ∘ ⟨(D, δ + τ)⟩
  else if T ≠ ∅ then
    T_i ← T_i ∘ ⟨(T, τ)⟩   {τ ≥ 0}
  end if
end while
return T_i
```

Algorithm 6 execute_tasks_early$(\mathcal{T})$

```
T_i ← T, T_c ← ⟨⟩, T_r ← ⟨⟩      {T = T_i ∘ T_c ∘ T_r}
ω ← 0
while T_i ≠ ⟨⟩ do
  T_i ← T_i ∘ ⟨(T, τ)⟩
  T_c ← ⟨(T, τ)⟩ ∘ T_c
  ω ← ω + Σ_{t∈T} wcet(t) − τ
  if ω > 0 then
    T_r ← T_c ∘ T_r
    T_c ← ⟨⟩
    ω ← 0
  end if
end while
return T_r
```

Pseudo code (2)

Algorithm 1 time_safety_check$(\mathcal{A}, \mathcal{T})$

```
if (A, T) ∉ H then
  H ← H ∪ {(A, T)}
  C ← enum_cond_assignment(number of if instructions)
  while C ≠ ∅ do
    C ← C ∪ {C}
    if is_safe(A, T, C) = false then
      time safety violation!
    end if
    T_n ← execute_tasks(A, T, C)
    T_n ← schedule_tasks(A, T_n, C)
    A_n ← future(A, C)
    time_safety_check(A_n, T_n)
  end while
end if
```

Algorithm 2 is_safe$(\mathcal{A}, \mathcal{T}, C)$

```
T_c ← completed(A, C)
T_r ← T, ω ← 0
while T_r ≠ ⟨⟩ do
  T_r ← ⟨(T, τ)⟩ ∘ T_r
  ω ← ω + Σ_{t∈T∩T_c} wcet(t) − τ
  ω ← max(0, ω)
end while
return ω = 0
```

Results and future work

- implemented as TimeChecker class in the current Giotto compiler
  - without IF : fast decision
  - with IF instructions :
    - and explicit enumeration of program branching paths : reasonable response times if n<10
    - and some simple code analysis : fast decision

    e.g. branching on task and driver guards can be avoided
- future
  - find patterns in configuration: simplified time safety proof
  - compositional time safety check