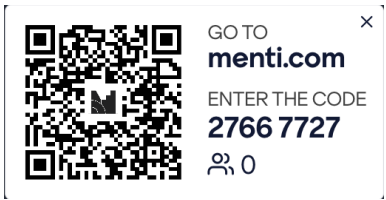
Scheduling and compiling rate-synchronous programs with end-to-end latency constraints

Timothy Bourke Marc Pouzet Vincent Bregeon (Airbus S.A.S.)

Inria Paris — PARKAS Team
École normale supérieure, PSL University

14 July 2023, ECRTS in Vienna



Context

- Set of periodic tasks communicating through variables:
  - » read data from sensors via a bus,
  - » compute via sequences of tasks, and
  - » write to actuators via the bus.

Context

- Set of periodic tasks communicating through variables:
  - » read data from sensors via a bus,
  - » compute via sequences of tasks, and
  - » write to actuators via the bus.

- Lustre:
  synchronous-reactive dataflow language for "model-based design"
  [Halbwachs, Caspi, Raymond, and Pilaud (1991): The synchronous dataflow programming language LUSTRE]

- Scade:
  modernized, industrial version
  [Colaço, Pagano, and Pouzet (2017): Scade 6: A Formal Language for Embedded Critical Software Development]

Context

- Set of periodic tasks communicating through variables:
  - » read data from sensors via a bus,
  - » compute via sequences of tasks, and
  - » write to actuators via the bus.

- Lustre:
  synchronous-reactive dataflow language for "model-based design"
  [Halbwachs, Caspi, Raymond, and Pilaud (1991): The synchronous dataflow programming language LUSTRE]

- Scade:
  modernized, industrial version
  [Colaço, Pagano, and Pouzet (2017): Scade 6: A Formal Language for Embedded Critical Software Development]

Airbus project "All-in-Lustre"

- *Original system*: $\approx 5\,000$ Lustre nodes
  $+$ separate constraints on order
- *Desired system*: a single Lustre program with features for periods and end-to-end latencies.
- Nodes at 10ms, 20ms, 40ms, and 120ms.
- *Implementation*: sequential code, period $= 5$ms

Context

- Set of periodic tasks communicating through variables:
  - » read data from sensors via a bus,
  - » compute via sequences of tasks, and
  - » write to actuators via the bus.

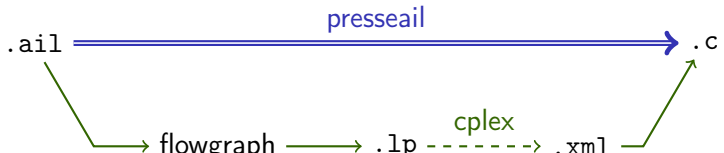
- Lustre:
  synchronous-reactive dataflow language for "model-based design"
  [Halbwachs, Caspi, Raymond, and Pilaud (1991): The synchronous dataflow programming language LUSTRE]

- Scade:
  modernized, industrial version
  [Colaço, Pagano, and Pouzet (2017): Scade 6: A Formal Language for Embedded Critical Software Development]

Airbus project "All-in-Lustre"

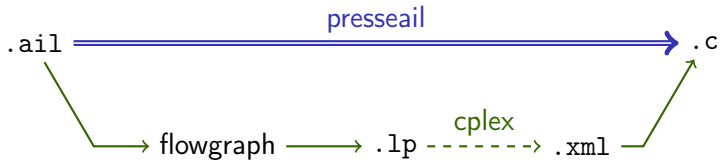
- *Original system*: $\approx 5\,000$ Lustre nodes
  $+$ separate constraints on order
- *Desired system*: a single Lustre program with features for periods and end-to-end latencies.
- Nodes at 10ms, 20ms, 40ms, and 120ms.
- *Implementation*: sequential code, period $= 5$ms
- Workload already chopped up into small pieces.
- Each node loops in < 5ms.

Overview: compilation using Integer Linear Programming (ILP)



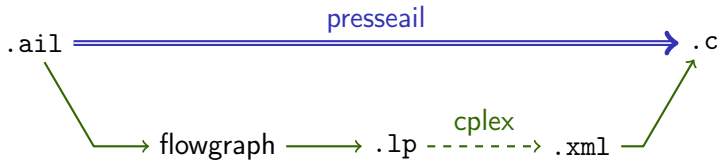
Overview: compilation using Integer Linear Programming (ILP)

- Like Prelude [Forget, Boniol, Lesens, and Pagetti (2010): A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems]
  But, no WCET, no deadlines, no real-time tasks
- Rates expressed as $1/n$ of the base clock



Overview: compilation using Integer Linear Programming (ILP)

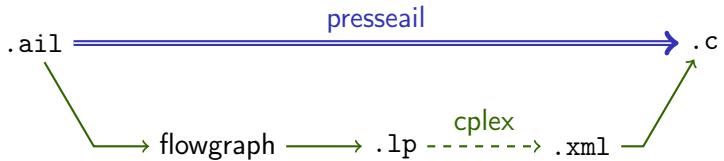
- Like Prelude [Forget, Boniol, Lesens, and Pagetti (2010): A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems]
  But, no WCET, no deadlines, no real-time tasks
- Rates expressed as $1/n$ of the base clock
- One or more `step` functions
- Called cyclically at the base rate



Overview: compilation using Integer Linear Programming (ILP)

- Like Prelude [Forget, Boniol, Lesens, and Pagetti (2010): A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems]
  But, no WCET, no deadlines, no real-time tasks
- Rates expressed as $1/n$ of the base clock

- One or more `step` functions
- Called cyclically at the base rate

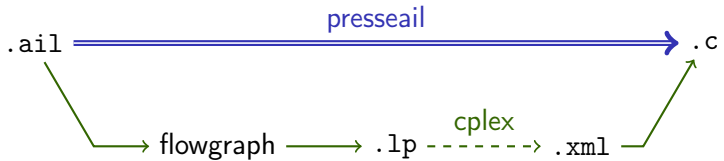


- Vertex = node
- Arc from producer to consumer
- Independent of source language

Overview: compilation using Integer Linear Programming (ILP)

- Like Prelude [Forget, Boniol, Lesens, and Pagetti (2010): A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems]
  But, no WCET, no deadlines, no real-time tasks
- Rates expressed as $1/n$ of the base clock

- One or more `step` functions
- Called cyclically at the base rate

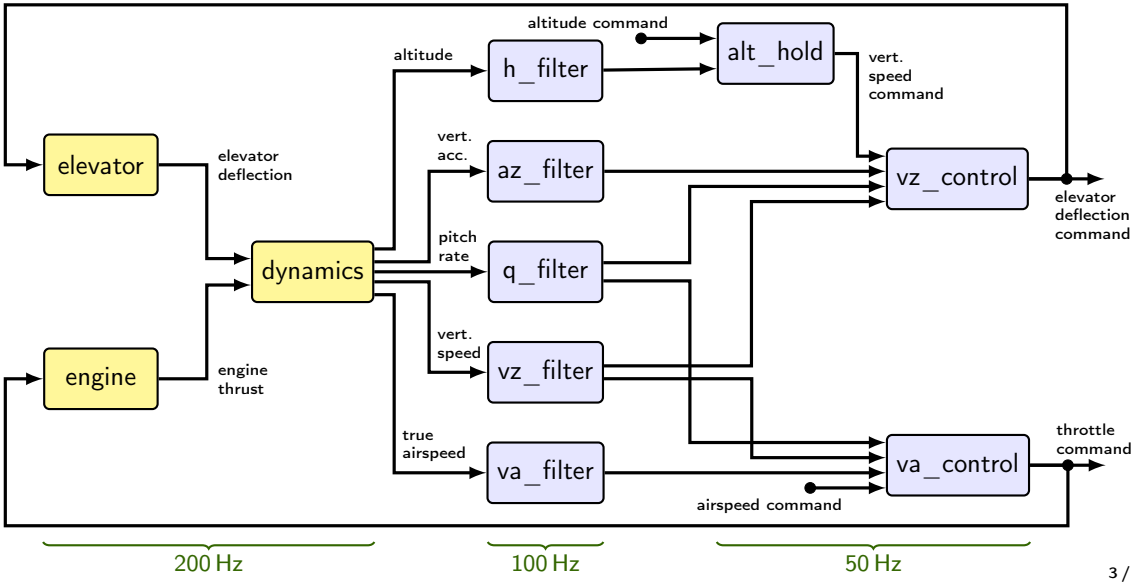


- Vertex = node
- Arc from producer to consumer
- Independent of source language

- Data dependencies
- Load balancing
- End-to-end latency

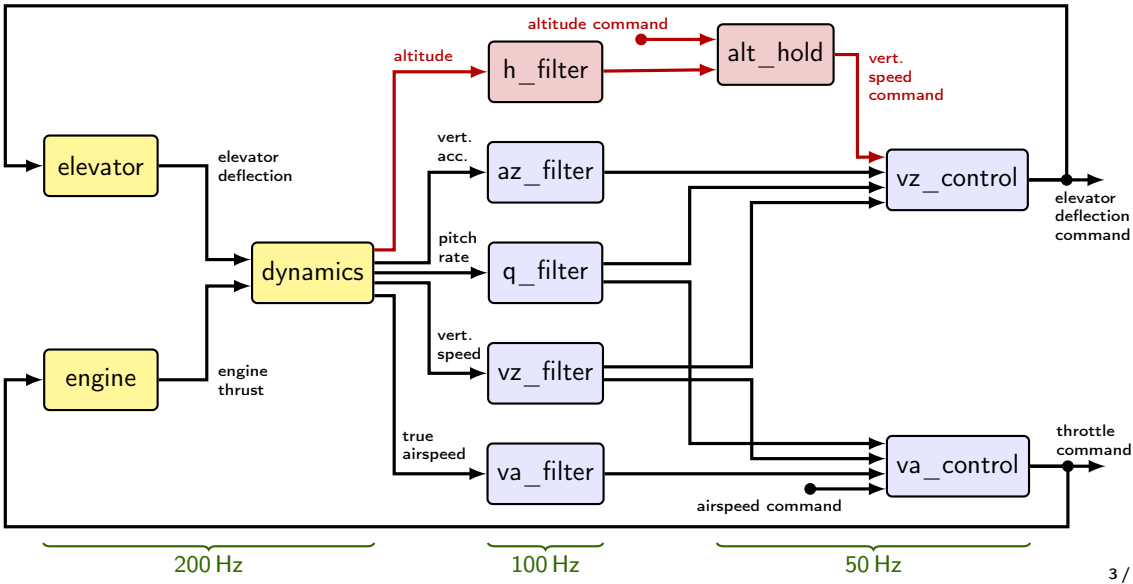
The ROSACE Case Study

[Pagetti, Saussié, Gratia, Noulard, and Siron (2014): The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution]



The ROSACE Case Study

[Pagetti, Saussié, Gratia, Noulard, and Siron (2014): The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution]



The ROSACE Case Study

[Pagetti, Saussié, Gratia, Noulard, and Siron (2014): The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution]

```
resource ops : int
node alt_hold (h_c, h_f : float) returns (vz_c : float) requires (ops = 201);
...
const H200 : rate = 1 / 2 (* base clock = 400Hz *)
const H100 : rate = 1 / 4
const H50  : rate = 1 / 8
const H10  : rate = 1 / 40

node assemblage1( h_c : float :: H10 last = 0.;   (* altitude command *)
                  va_c : float :: H10 last = 0.)  (* airspeed command *)
returns (d_th_c : float :: H50 last = 1.6402;     (* throttle command *)
         d_e_c  : float :: H50 last = 0.0186)     (* elevator deflection command *)
var  h_f : float :: H100; (* altitude *)
    vz_c : float :: H50;  (* vertical speed command *)
    ...
let
   h_f =  h_filter( h when (? % 2) );
  vz_c = alt_hold( current(h_c, (? % 5)), h_f when (? % 2) );
  ...
  resource balance ops;
  latency assemblage exists <= 2
    (dynamics, h_filter, alt_hold, vz_control, elevator);
tel
```

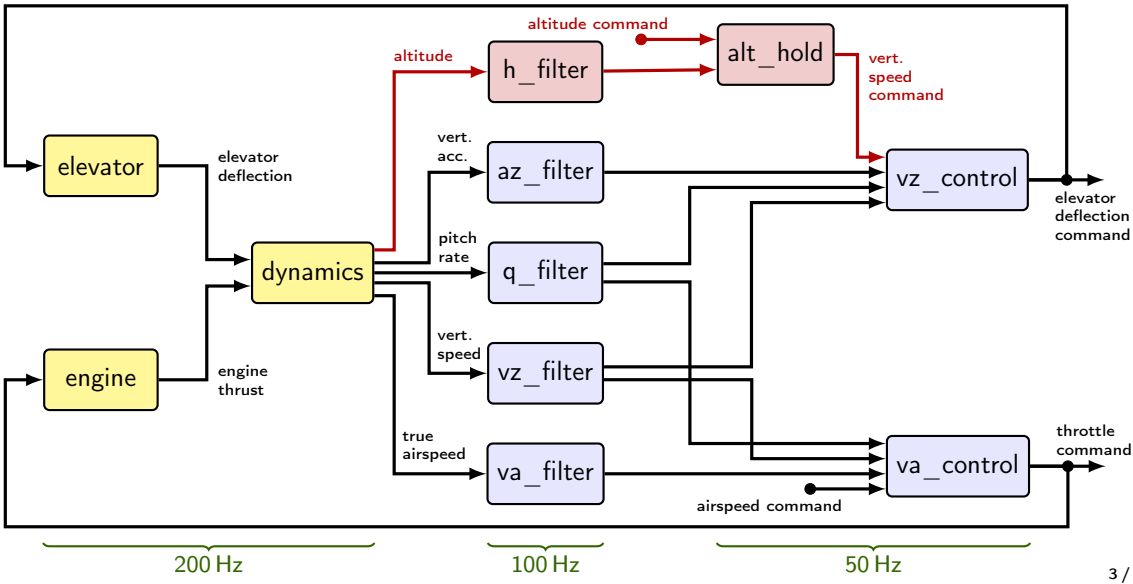
elevator deflection command

throttle command

200 Hz 100 Hz 50 Hz

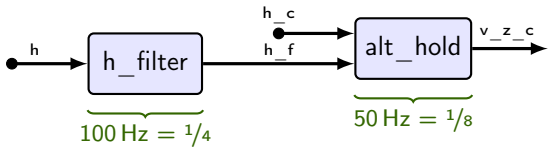
The ROSACE Case Study

[Pagetti, Saussié, Gratia, Noulard, and Siron (2014): The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution]



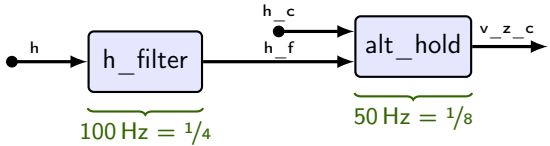
Compilation: Model ⇒ Scheduling ⇒ Generated Code

```
 h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```



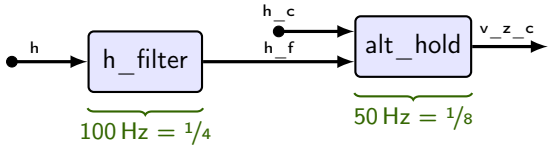
Compilation: Model ⇒ Scheduling ⇒ Generated Code

```
 h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```



Compilation: Model ⇒ Scheduling ⇒ Generated Code

```
h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```

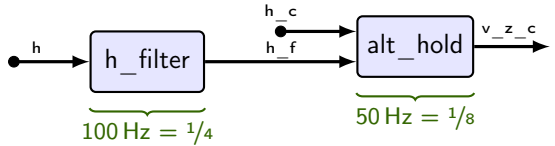



```
static int c_30 = 0;

void step0()
{
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter();      // ***
            ...
        }
    } else {
        ...
    }
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold();      // ***
            vz_control();
            break;
    }
    c_30 = (c_30 + 1) % 8;
}
```

Compilation: Model ⇒ Scheduling ⇒ Generated Code

```
h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```



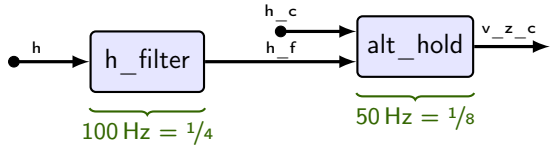
- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

```
static int c_30 = 0;

void step0()
{
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter();      // ***
            ...
        }
    } else {
        ...
    }
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold();      // ***
            vz_control();
            break;
    }
    c_30 = (c_30 + 1) % 8;
}
```

Compilation: Model ⇒ Scheduling ⇒ Generated Code

```
h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```



- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

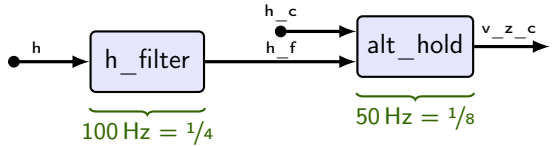
```
static int c_30 = 0;

void step0()
{
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter();      // ***
            ...
        }
    } else {
        ...
    }
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold();      // ***
            vz_control();
            break;
    }
    c_30 = (c_30 + 1) % 8;
}
```

f (concomitance)

Compilation: Model ⇒ Scheduling ⇒ Generated Code

```
h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```



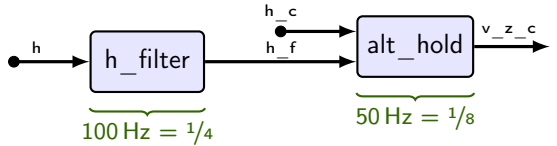
- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

```
static int c_30 = 0;

void step0()
{
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter();       // ***
            ...
        }
    } else {
        ...
    }
    switch (c_30) {
    case 0: va_control(); break;
    case 6: alt_hold();       // ***
            vz_control();
            break;
    }
    c_30 = (c_30 + 1) % 8;
}
```

Compilation: Model ⇒ Scheduling ⇒ Generated Code

```
h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```



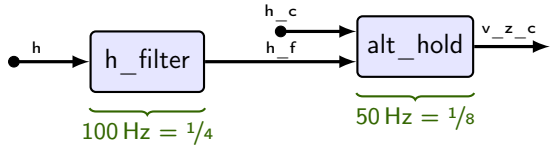
- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

```
static int c_30 = 0;

void step0()
{
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold();      // ***
            vz_control();
            break;
    }
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter();      // ***
            ...
        }
    } else {
        ...
    }
    c_30 = (c_30 + 1) % 8;
}
```

Compilation: Model ⇒ Scheduling ⇒ Generated Code

```
h_f =  h_filter(h when (? % 2));
vz_c = alt_hold(current(h_c, (? % 5)),
                h_f when (? % 2));
```



- Source: dataflow semantics
- Target: C code implicitly writing and reading static variables

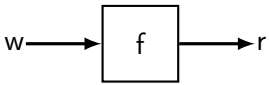
```
static int c_30 = 0;

void step0()
{
    switch (c_30) {
    case 2: va_control(); break;
    case 6: alt_hold();      // ***
            vz_control();
            break;
    }
    if (c_30 % 2 == 0) {
        if (c_30 % 4 == 2) {
            h_filter();      // ***
            ...
        }
    } else {
        ...
    }
    c_30 = (c_30 + 1) % 8;
}
```

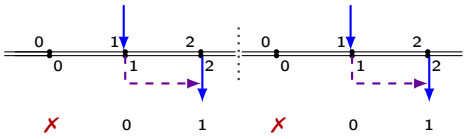
b (concomitance)

Direct Communications

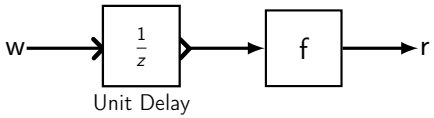
```
r = f(w)
```



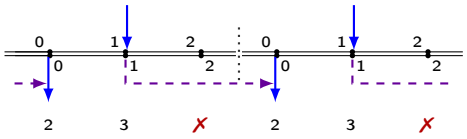
- D_f^w: Direct Write-before-read
  (forward concomitance)
- Dependency constraint: p:$w \leq$ p:r
- $0 \leq$ p:$r -$ p:$w <$ *period*




```
r = f(last w)
```



- D_b^r: Direct Read-before-write
  (backward concomitance)
- Dependency constraint: p:$r \leq$ p:w
- $0 <$ p:$r -$ p:$w +$ *period* $\leq$ *period*

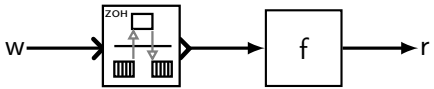


Rate Transitions

`r = f(w when (1 % 3))`

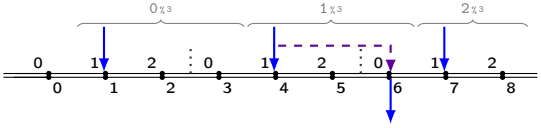
`(i % n)`: take value `i` of every `n`

`(? % n)`: take any of every `n` values



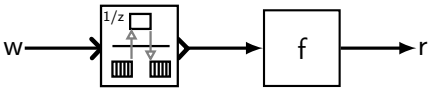
Fast-to-Slow Transition

- $/_{nf}$: Fast-to-slow
  (forward concomitance)



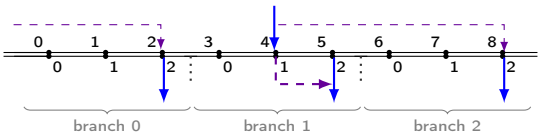
`r = f(current(w, (1 % 3)))`

`(i % n)`: `i` initial values, then repeat `n` times



Slow-to-Fast Transition

- $*_{nf} \mid *_{nb}$: Slow-to-fast
  (forward or backward concomitance)



Valid programs are defined by clock typing

$$\frac{e_1 :: 1/n \qquad e_2 :: 1/n}{e_1 \oplus e_2 :: 1/n}$$

$$\frac{x :: 1/n}{\texttt{last}\ x :: 1/n}$$

$$\frac{x :: 1/m}{x\ \texttt{when}\ (\cdot \%\, n) :: 1/mn}$$

$$\frac{x :: 1/mn}{\texttt{current}(x, (\cdot \%\, n)) :: 1/m}$$

- No phase offsets in clock types, unlike
  - » Prelude: `rate(100, 0)`
    [Forget, Boniol, Lesens, and Pagetti (2010): A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems]
  - » Lucy-n: `(010), 00(00100)`
    [Cohen, Duranton, Eisenbeis, Pagetti, Plateau, and Pouzet (2006): N-Synchronous Kahn networks: a relaxed model of synchrony for real-time systems]
  - » 1-Synchronous: `[0, 2]`
    [Iooss, Pouzet, Cohen, Potop-Butucaru, Souyris, Bregeon, and Baufreton (2020): 1-Synchronous Programming of Large Scale, Multi-Periodic Real-Time Applications with Functional Degrees of Freedom]
  - » Simulink: `[Ts, To]`
- Dataflow semantics: independent of phase offsets
- Generated code: phase offsets implement data dependencies.

Valid programs are defined by clock typing

$$\frac{e_1 :: 1/n \qquad e_2 :: 1/n}{e_1 \oplus e_2 :: 1/n}$$

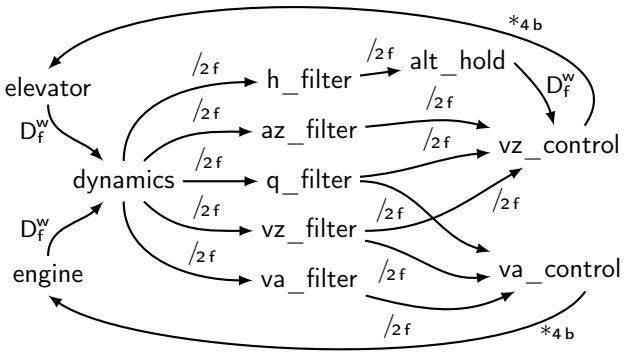
$$\frac{x :: 1/n}{\texttt{last}\ x :: 1/n}$$

$$\frac{x :: 1/m}{x\ \texttt{when}\ (\cdot \%\, n) :: 1/mn}$$

$$\frac{x :: 1/mn}{\texttt{current}(x, (\cdot \%\, n)) :: 1/m}$$

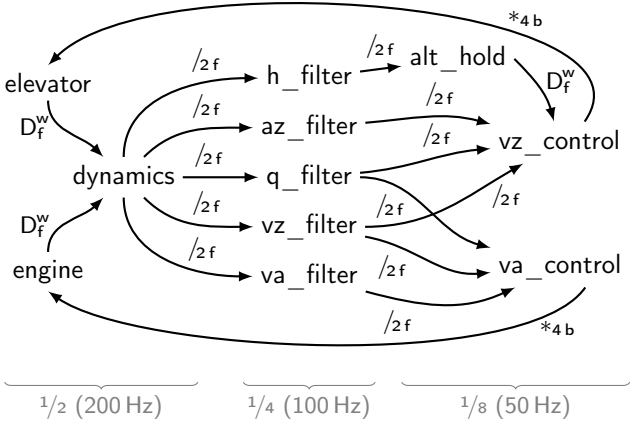
- No phase offsets in clock types, unlike
  - » Prelude: `rate(100, 0)`
    [Forget, Boniol, Lesens, and Pagetti (2010): A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems]
  - » Lucy-n: `(010), 00(00100)`
    [Cohen, Duranton, Eisenbeis, Pagetti, Plateau, and Pouzet (2006): N-Synchronous Kahn networks: a relaxed model of synchrony for real-time systems]
  - » 1-Synchronous: `[0, 2]`
    [Iooss, Pouzet, Cohen, Potop-Butucaru, Souyris, Bregeon, and Baufreton (2020): 1-Synchronous Programming of Large Scale, Multi-Periodic Real-Time Applications with Functional Degrees of Freedom]
  - » Simulink: `[Ts, To]`
- Dataflow semantics: independent of phase offsets
- Generated code: phase offsets implement data dependencies.

Flowgraphs and Latency chains



- Generate flowgraph from program, annotations:
  - » rate transitions
  - » concomitance (order within cycle)
- Identify and eliminate cycles
- Transform path into an ILP constraint to constrain the schedule

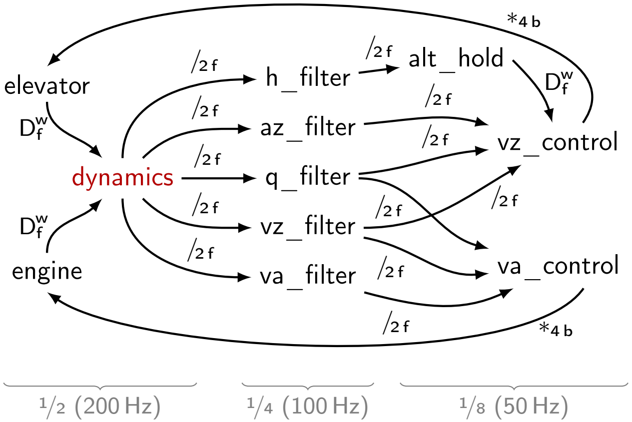
Flowgraphs and Latency chains



- Generate flowgraph from program, annotations:
  - » rate transitions
  - » concomitance (order within cycle)
- Identify and eliminate cycles
- Transform path into an ILP constraint to constrain the schedule

```
latency exists <= 2 (dynamics, h_filter, alt_hold, vz_control, elevator);
```

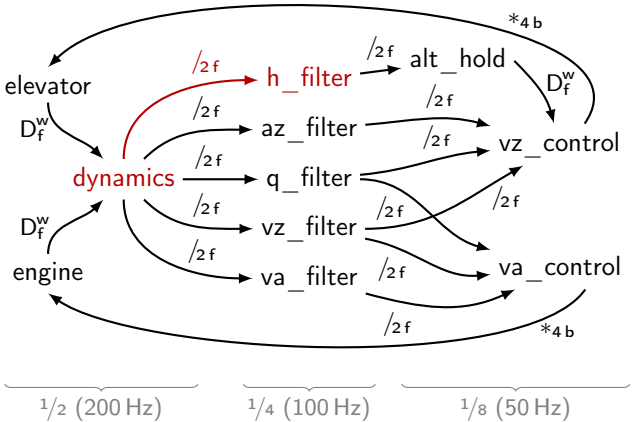
Flowgraphs and Latency chains



- Generate flowgraph from program, annotations:
  - rate transitions
  - concomitance (order within cycle)
- Identify and eliminate cycles
- Transform path into an ILP constraint to constrain the schedule

```
latency exists <= 2 (dynamics, h_filter, alt_hold, vz_control, elevator);
```

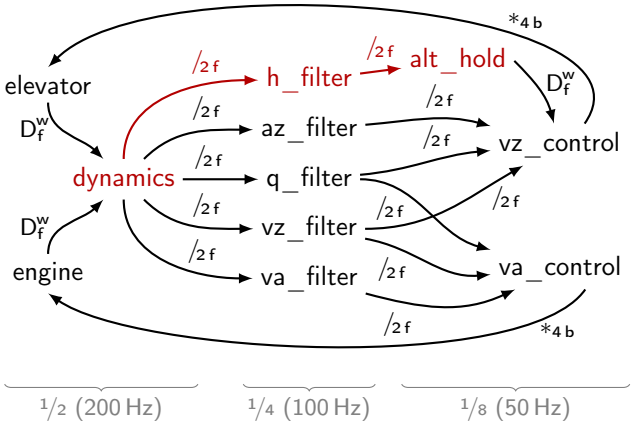
Flowgraphs and Latency chains



- Generate flowgraph from program, annotations:
  - » rate transitions
  - » concomitance (order within cycle)
- Identify and eliminate cycles
- Transform path into an ILP constraint to constrain the schedule

```
latency exists <= 2 (dynamics, h_filter, alt_hold, vz_control, elevator);
```

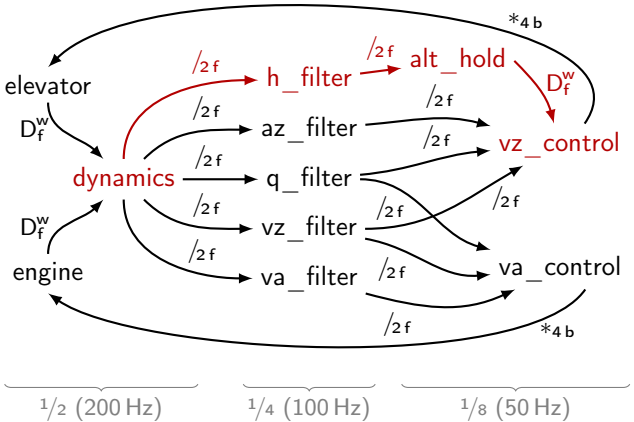
Flowgraphs and Latency chains



- Generate flowgraph from program, annotations:
  - » rate transitions
  - » concomitance (order within cycle)
- Identify and eliminate cycles
- Transform path into an ILP constraint to constrain the schedule

```
latency exists <= 2 (dynamics, h_filter, alt_hold, vz_control, elevator);
```

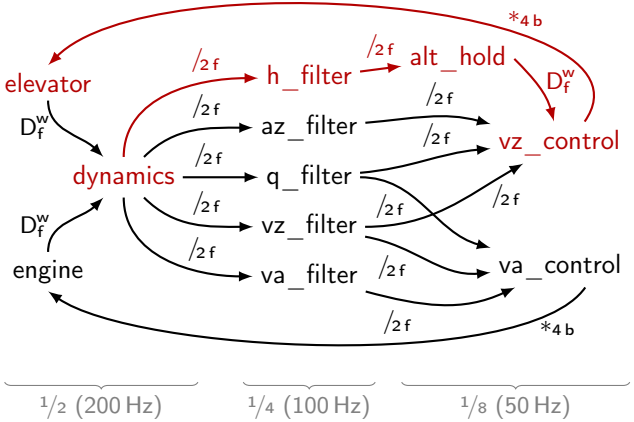
Flowgraphs and Latency chains



- Generate flowgraph from program, annotations:
  - » rate transitions
  - » concomitance (order within cycle)
- Identify and eliminate cycles
- Transform path into an ILP constraint to constrain the schedule

```
latency exists <= 2 (dynamics, h_filter, alt_hold, vz_control, elevator);
```

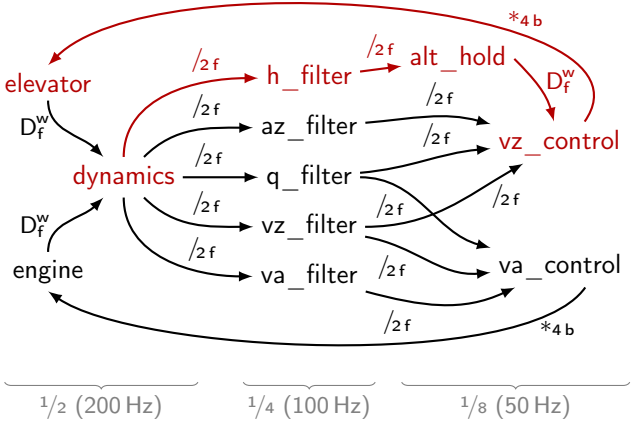
Flowgraphs and Latency chains



- Generate flowgraph from program, annotations:
  - » rate transitions
  - » concomitance (order within cycle)
- Identify and eliminate cycles
- Transform path into an ILP constraint to constrain the schedule

```
latency exists <= 2 (dynamics, h_filter, alt_hold, vz_control, elevator);
```

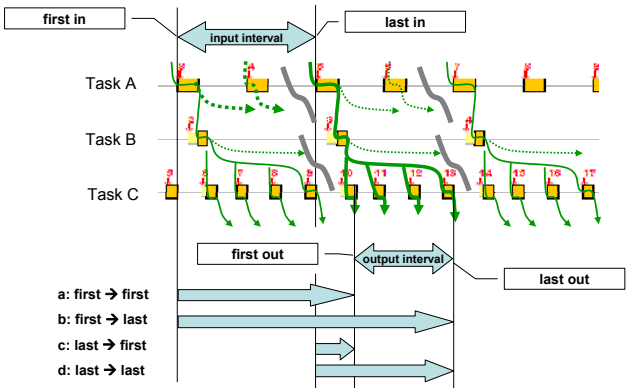
Flowgraphs and Latency chains



- Generate flowgraph from program, annotations:
  - » rate transitions
  - » concomitance (order within cycle)
- Identify and eliminate cycles
- Transform path into an ILP constraint to constrain the schedule

```
latency exists <= 2 (dynamics, h_filter, alt_hold, vz_control, elevator);
latency exists <= 2 (d       , h       , a       , v          , e       );
```

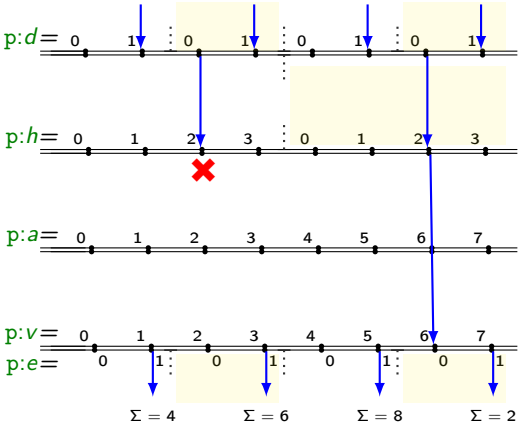
End-to-End Latency



[Feiertag, Richter, Nordlander, and Jonsson (2008): A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics]

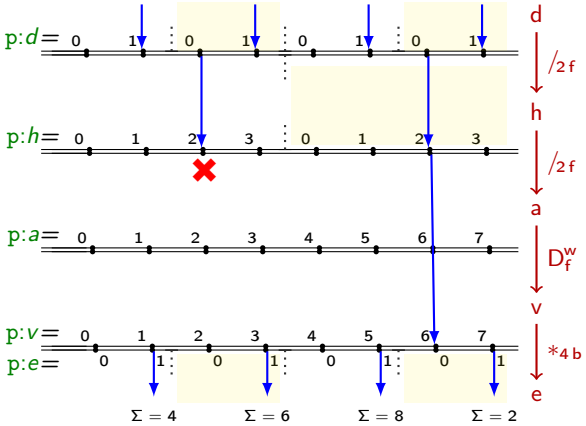
- first-to-first = reaction time = `forward`
- last-to-last = data age = `backward`
- at least one backward path = `exists`
- Lots of other related work
- We ignore execution time and jitter

Constraining End-to-end Latency



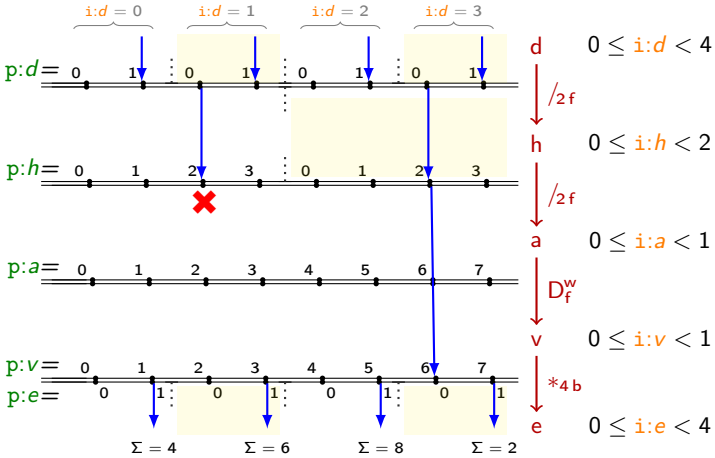
(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency



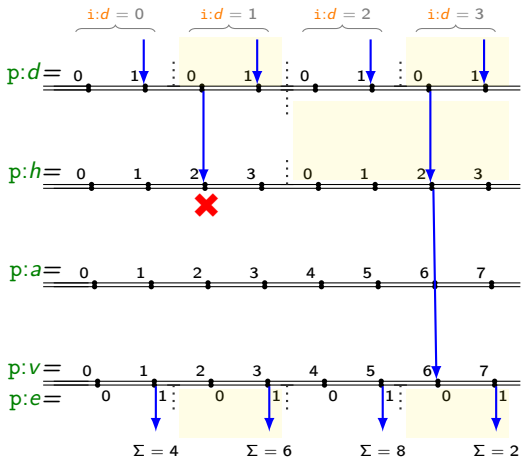
(View online at https://www.tbrk.org/dataflow/showlatency)

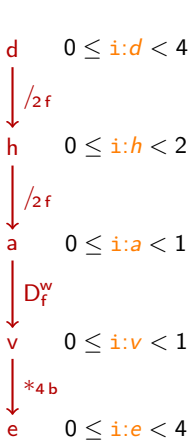
Constraining End-to-end Latency



(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$0 \le \mathtt{i}{:}d < 4$

$0 \le \mathtt{i}{:}h < 2$

$0 \le \mathtt{i}{:}a < 1$

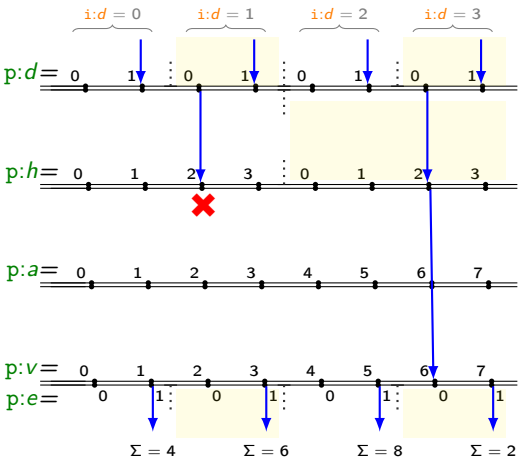
$0 \le \mathtt{i}{:}v < 1$

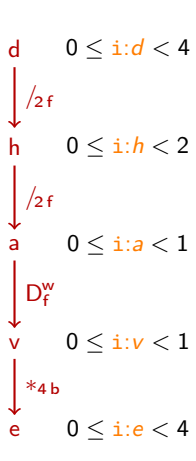
$0 \le \mathtt{i}{:}e < 4$

$$0 \le \mathtt{lat}_{d,h} < 2$$
$$2 \cdot \mathtt{i}{:}d + \mathtt{p}{:}d + \mathtt{lat}_{d,h} = 4 \cdot \mathtt{i}{:}h + \mathtt{p}{:}h$$

(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$0 \le \mathrm{i}{:}d < 4$

$0 \le \mathrm{i}{:}h < 2$

$0 \le \mathrm{i}{:}a < 1$

$0 \le \mathrm{i}{:}v < 1$

$0 \le \mathrm{i}{:}e < 4$

$0 \le \mathrm{lat}_{d,h} < 2$

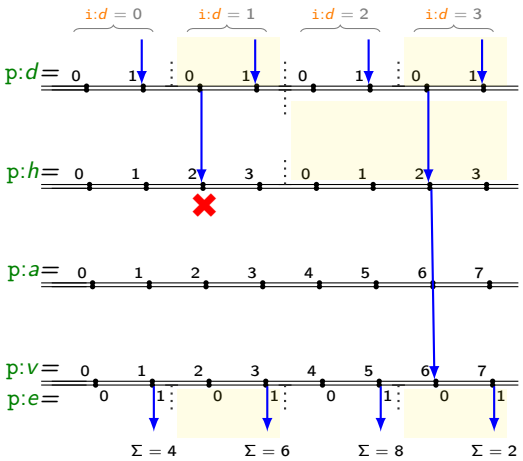
$2 \cdot \mathrm{i}{:}d + \mathrm{p}{:}d + \mathrm{lat}_{d,h} = 4 \cdot \mathrm{i}{:}h + \mathrm{p}{:}h$

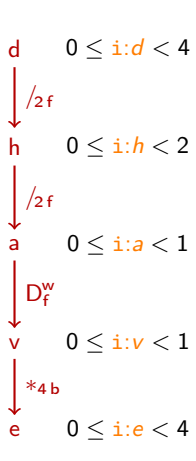
$0 \le \mathrm{lat}_{h,a} < 4$

$4 \cdot \mathrm{i}{:}h + \mathrm{p}{:}h + \mathrm{lat}_{h,a} = 8 \cdot \mathrm{i}{:}a + \mathrm{p}{:}a$

(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$0 \le \mathrm{i}{:}d < 4$

$0 \le \mathrm{i}{:}h < 2$

$0 \le \mathrm{i}{:}a < 1$

$0 \le \mathrm{i}{:}v < 1$

$0 \le \mathrm{i}{:}e < 4$

$0 \le \mathrm{lat}_{d,h} < 2$

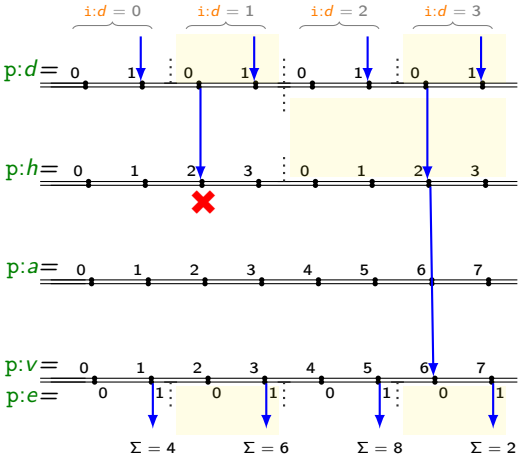
$2 \cdot \mathrm{i}{:}d + \mathrm{p}{:}d + \mathrm{lat}_{d,h} = 4 \cdot \mathrm{i}{:}h + \mathrm{p}{:}h$

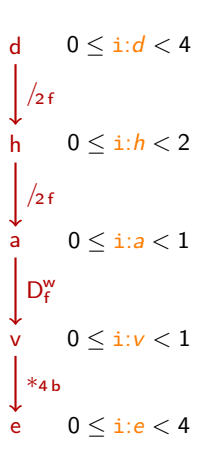
$0 \le \mathrm{lat}_{h,a} < 4$

$4 \cdot \mathrm{i}{:}h + \mathrm{p}{:}h + \mathrm{lat}_{h,a} = 8 \cdot \mathrm{i}{:}a + \mathrm{p}{:}a$

(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$0 \le \mathrm{i}{:}d < 4$

$0 \le \mathrm{i}{:}h < 2$

$0 \le \mathrm{i}{:}a < 1$

$0 \le \mathrm{i}{:}v < 1$

$0 \le \mathrm{i}{:}e < 4$

$0 \le \mathtt{lat}_{d,h} < 2$

$2 \cdot \mathrm{i}{:}d + \mathrm{p}{:}d + \mathtt{lat}_{d,h} = 4 \cdot \mathrm{i}{:}h + \mathrm{p}{:}h$

$0 \le \mathtt{lat}_{h,a} < 4$

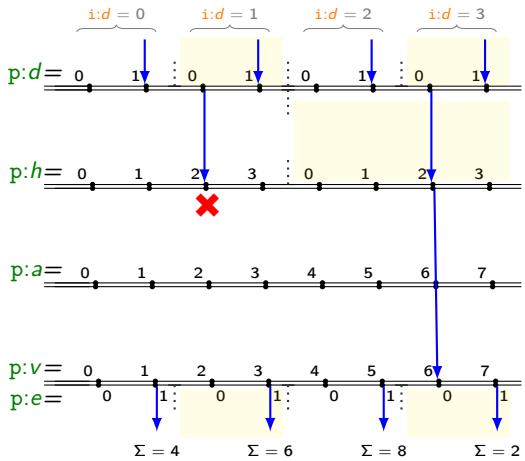
$4 \cdot \mathrm{i}{:}h + \mathrm{p}{:}h + \mathtt{lat}_{h,a} = 8 \cdot \mathrm{i}{:}a + \mathrm{p}{:}a$

$0 \le \mathtt{lat}_{a,v} < 8$

$8 \cdot \mathrm{i}{:}a + \mathrm{p}{:}a + \mathtt{lat}_{a,v} = 8 \cdot \mathrm{i}{:}v + \mathrm{p}{:}v$

(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency



d — $0 \le \text{i:}d < 4$

$/_{2\,\text{f}}$

h — $0 \le \text{i:}h < 2$

$/_{2\,\text{f}}$

a — $0 \le \text{i:}a < 1$

D_{f}^{w}

v — $0 \le \text{i:}v < 1$

$*_{4\,\text{b}}$

e — $0 \le \text{i:}e < 4$

$$0 \le \text{lat}_{d,h} < 2$$
$$2 \cdot \text{i:}d + \text{p:}d + \text{lat}_{d,h} = 4 \cdot \text{i:}h + \text{p:}h$$

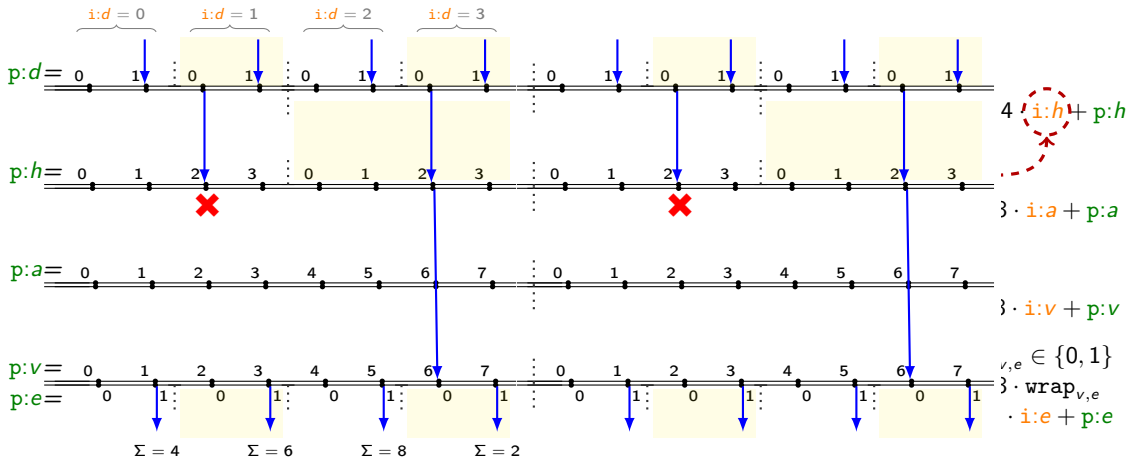
$$0 \le \text{lat}_{h,a} < 4$$
$$4 \cdot \text{i:}h + \text{p:}h + \text{lat}_{h,a} = 8 \cdot \text{i:}a + \text{p:}a$$

$$0 \le \text{lat}_{a,v} < 8$$
$$8 \cdot \text{i:}a + \text{p:}a + \text{lat}_{a,v} = 8 \cdot \text{i:}v + \text{p:}v$$

$$0 < \text{lat}_{v,e} \le 8 \qquad \text{wrap}_{v,e} \in \{0, 1\}$$
$$8 \cdot \text{i:}v + \text{p:}v + \text{lat}_{v,e} - 8 \cdot \text{wrap}_{v,e} = 2 \cdot \text{i:}e + \text{p:}e$$

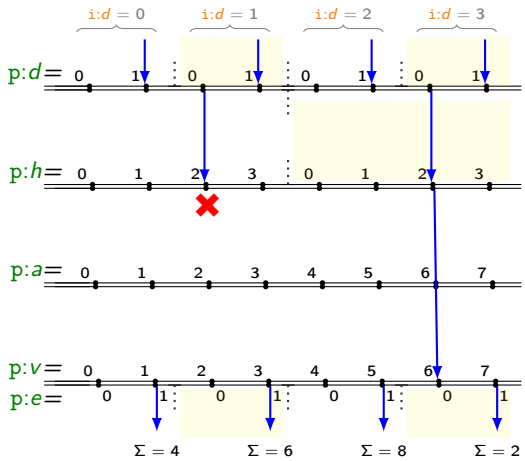
(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency



(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency



d — $0 \le \text{i:}d < 4$

$/_{2\,\text{f}}$

h — $0 \le \text{i:}h < 2$

$/_{2\,\text{f}}$

a — $0 \le \text{i:}a < 1$

D_{f}^{w}

v — $0 \le \text{i:}v < 1$

$*_{4\,\text{b}}$

e — $0 \le \text{i:}e < 4$

$$0 \le \text{lat}_{d,h} < 2$$
$$2 \cdot \text{i:}d + \text{p:}d + \text{lat}_{d,h} = 4 \cdot \text{i:}h + \text{p:}h$$

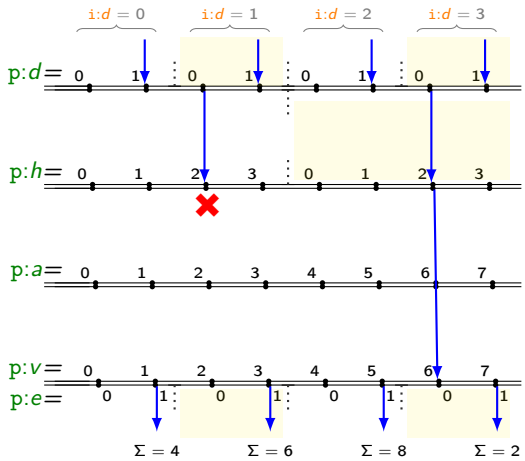
$$0 \le \text{lat}_{h,a} < 4$$
$$4 \cdot \text{i:}h + \text{p:}h + \text{lat}_{h,a} = 8 \cdot \text{i:}a + \text{p:}a$$

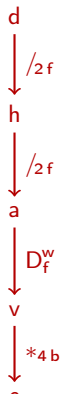
$$0 \le \text{lat}_{a,v} < 8$$
$$8 \cdot \text{i:}a + \text{p:}a + \text{lat}_{a,v} = 8 \cdot \text{i:}v + \text{p:}v$$

$$0 < \text{lat}_{v,e} \le 8 \qquad \text{wrap}_{v,e} \in \{0, 1\}$$
$$8 \cdot \text{i:}v + \text{p:}v + \text{lat}_{v,e} - 8 \cdot \text{wrap}_{v,e} = 2 \cdot \text{i:}e + \text{p:}e$$

(View online at https://www.tbrk.org/dataflow/showlatency)

Constraining End-to-end Latency





$0 \le \text{i:}d < 4$

$0 \le \text{i:}h < 2$

$0 \le \text{i:}a < 1$

$0 \le \text{i:}v < 1$

$0 \le \text{i:}e < 4$

$0 \le \text{lat}_{d,h} < 2$

$2 \cdot \text{i:}d + \text{p:}d + \text{lat}_{d,h} = 4 \cdot \text{i:}h + \text{p:}h$

$0 \le \text{lat}_{h,a} < 4$

$4 \cdot \text{i:}h + \text{p:}h + \text{lat}_{h,a} = 8 \cdot \text{i:}a + \text{p:}a$

$0 \le \text{lat}_{a,v} < 8$

$8 \cdot \text{i:}a + \text{p:}a + \text{lat}_{a,v} = 8 \cdot \text{i:}v + \text{p:}v$

$0 < \text{lat}_{v,e} \le 8 \qquad \text{wrap}_{v,e} \in \{0, 1\}$

$8 \cdot \text{i:}v + \text{p:}v + \text{lat}_{v,e} - 8 \cdot \text{wrap}_{v,e} = 2 \cdot \text{i:}e + \text{p:}e$

$\text{lat}_{d,h} + \text{lat}_{h,a} + \text{lat}_{a,v} + \text{lat}_{v,e} \le 2$

(View online at https://www.tbrk.org/dataflow/showlatency)

Conclusion

- Programming language for composing tasks
  - » Particularity: tasks must terminate in one cycle
  - » Semantics, static analysis (clock types), compilation
- Use an ILP solver for scheduling
  - » Load balancing
  - » End-to-end latency
- Prototype compiler in OCaml
  with ILP scheduling and basic code generation
- Tested on Airbus example with 5000 nodes
  (compiles in approx. 45 minutes).

References I

- Biernacki, D., J.-L. Colaço, G. Hamon, and M. Pouzet (June 2008). "Clock-directed modular code generation for synchronous data-flow languages". In: *Proc. 9th ACM SIGPLAN Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES 2008)*. Tucson, AZ, USA: ACM Press, pp. 121–130.
- Cohen, A., M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau, and M. Pouzet (Jan. 2006). "N-Synchronous Kahn networks: a relaxed model of synchrony for real-time systems". In: *Proc. 33rd ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL 2006)*. Charleston, SC, USA: ACM Press, pp. 180–193.
- Colaço, J.-L., B. Pagano, and M. Pouzet (Sept. 2017). "Scade 6: A Formal Language for Embedded Critical Software Development". In: *Proc. 11th Int. Symp. Theoretical Aspects of Software Engineering (TASE 2017)*. Nice, France: IEEE Computer Society, pp. 4–15.
- Feiertag, N., K. Richter, J. Nordlander, and J. Jonsson (Nov. 2008). "A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics". In: *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2008, co-located with RTSS 2008)*. Barcelona, Spain.

References II

- Forget, J., F. Boniol, D. Lesens, and C. Pagetti (Mar. 2010). "A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems". In: *Proc. 25th ACM Symp. Applied Computing (SAC'10)*. Ed. by S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal, and C.-C. Hung. Sierre, Switzerland: ACM, pp. 527–534.
- Halbwachs, N., P. Caspi, P. Raymond, and D. Pilaud (Sept. 1991). "The synchronous dataflow programming language LUSTRE". In: *Proc. IEEE* 79.9, pp. 1305–1320.
- Iooss, G., M. Pouzet, A. Cohen, D. Potop-Butucaru, J. Souyris, V. Bregeon, and P. Baufreton (Mar. 2020). "1-Synchronous Programming of Large Scale, Multi-Periodic Real-Time Applications with Functional Degrees of Freedom". preprint.
- Pagetti, C., J. Forget, F. Boniol, M. Cordovilla, and D. Lesens (Sept. 2011). "Multi-task implementation of multi-periodic synchronous programs". In: *Discrete Event Dynamic Systems* 21.3, pp. 307–338.
- Pagetti, C., D. Saussié, R. Gratia, E. Noulard, and P. Siron (Apr. 2014). "The ROSACE Case Study: From Simulink Specification to Multi/Many-Core Execution". In: *20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2014)*. IEEE. Berlin, Germany, pp. 309–318.