Monitoring Cyber-Physical Systems: From Design to Integration

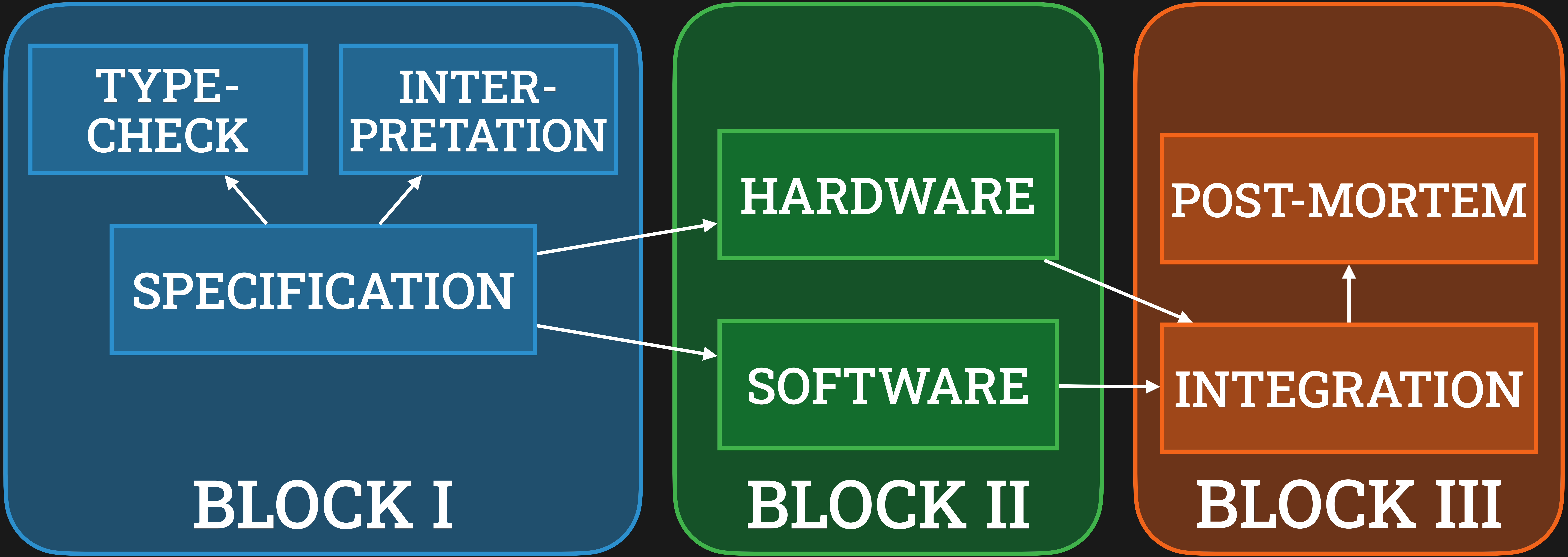
Maximilian Schwenger

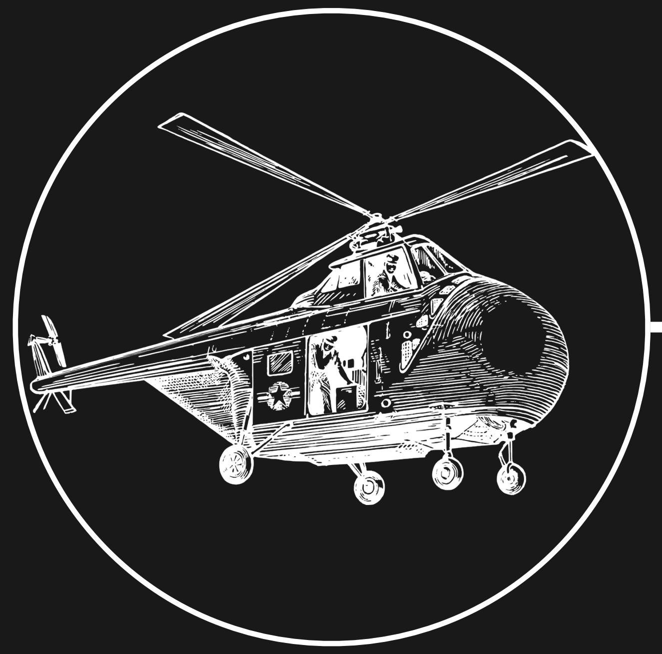
CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY


DLR
artis

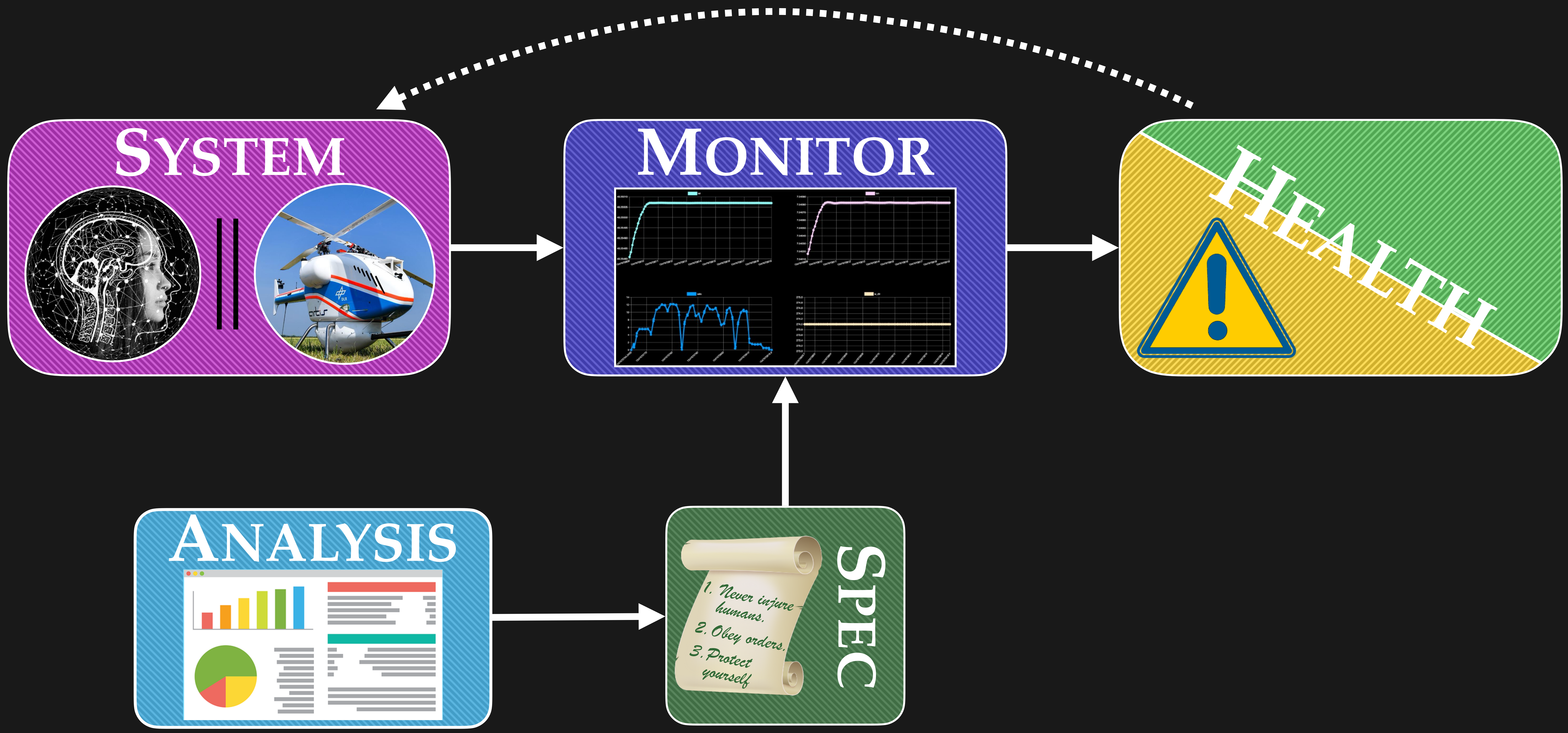

DLR

Overview


TYPE-CHECK
INTER-PRETATION
SPECIFICATION
BLOCK I
HARDWARE
SOFTWARE
BLOCK II
POST-MORTEM
INTEGRATION
BLOCK III

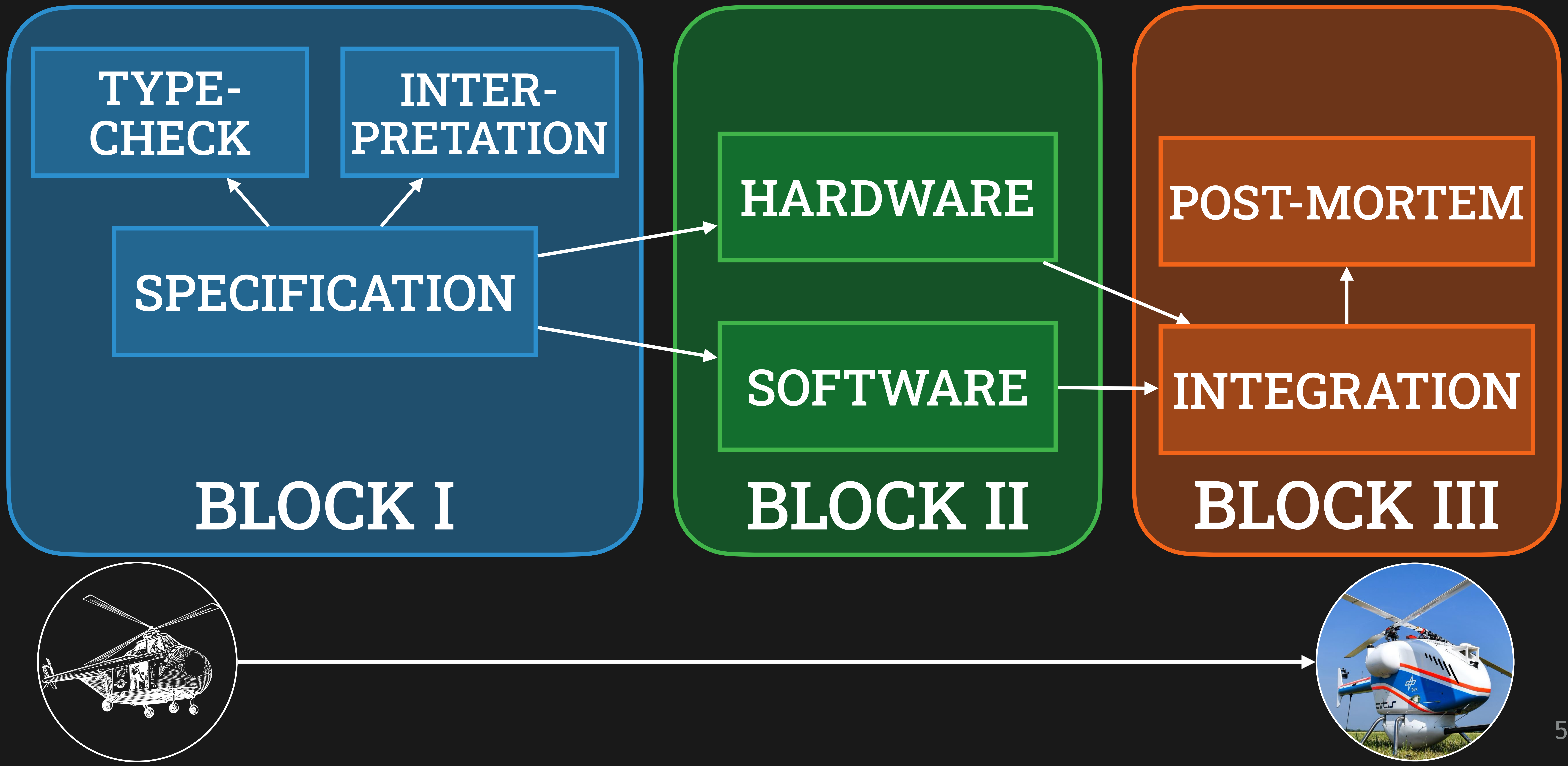


Our Setup



System
Monitor
Health
Analysis
Spec
1. Never injure humans.
2. Obey orders.
3. Protect yourself

Block I Specification



Block I — Property Spectrum

	Sensor Level	Mission Level
Timeliness		
Arithmetic Challenge		
Input Data		
Locality		
Example		

Property Spectrum

	Sensor Level	Mission Level
Timeliness		
Arithmetic Challenge		
Input Data		
Locality		
Example	Data Validation: *"Altimeter must produce positives values."*	Mission Statistics: *"Low correlation between WP distance and relative path deviation."*

Block I Property Spectrum

	Sensor Level	Mission Level
Timeliness	Critical	Lax
Arithmetic Challenge		
Input Data		
Locality		
Example	Data Validation: *"Altimeter must produce positives values."*	Mission Statistics: *"Low correlation between WP distance and relative path deviation."*

Block I Property Spectrum

	Sensor Level	Mission Level
Timeliness	Critical	Lax
Arithmetic Challenge	Low (bounds checks, counting)	High (statistics, prediction)
Input Data		
Locality		
Example	Data Validation: *“Altimeter must produce positives values.”*	Mission Statistics: *“Low correlation between WP distance and relative path deviation.”*

Block I Property Spectrum

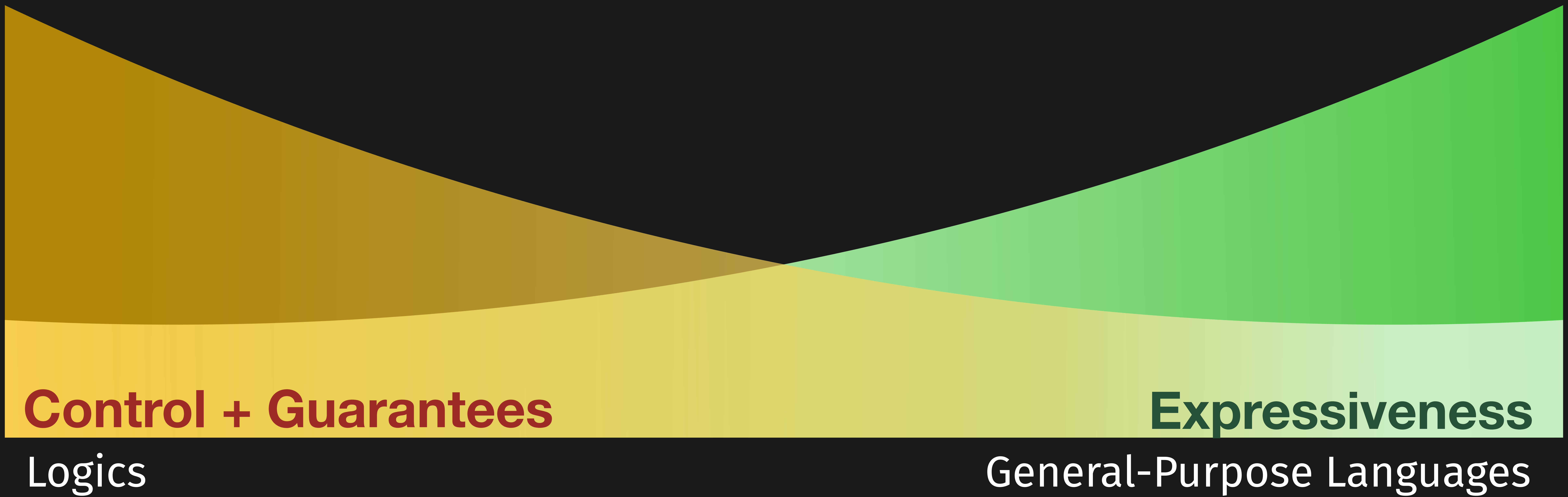
	Sensor Level	Mission Level
Timeliness	Critical	Lax
Arithmetic Challenge	Low (bounds checks, counting)	High (statistics, prediction)
Input Data	Raw	Refined
Locality		
Example	Data Validation: *"Altimeter must produce positives values."*	Mission Statistics: *"Low correlation between WP distance and relative path deviation."*

Block I Property Spectrum

	Sensor Level		Mission Level
Timeliness	Critical		Lax
Arithmetic Challenge	Low (bounds checks, counting)		High (statistics, prediction)
Input Data	Raw		Refined
Locality	Local	Inter-component	System-wide
Example	Data Validation: *"Altimeter must produce positives values."*		Mission Statistics: *"Low correlation between WP distance and relative path deviation."*

One Language to Rule Them All?

One Language to Rule Them All?



One Language to Rule Them All?

Points to consider:

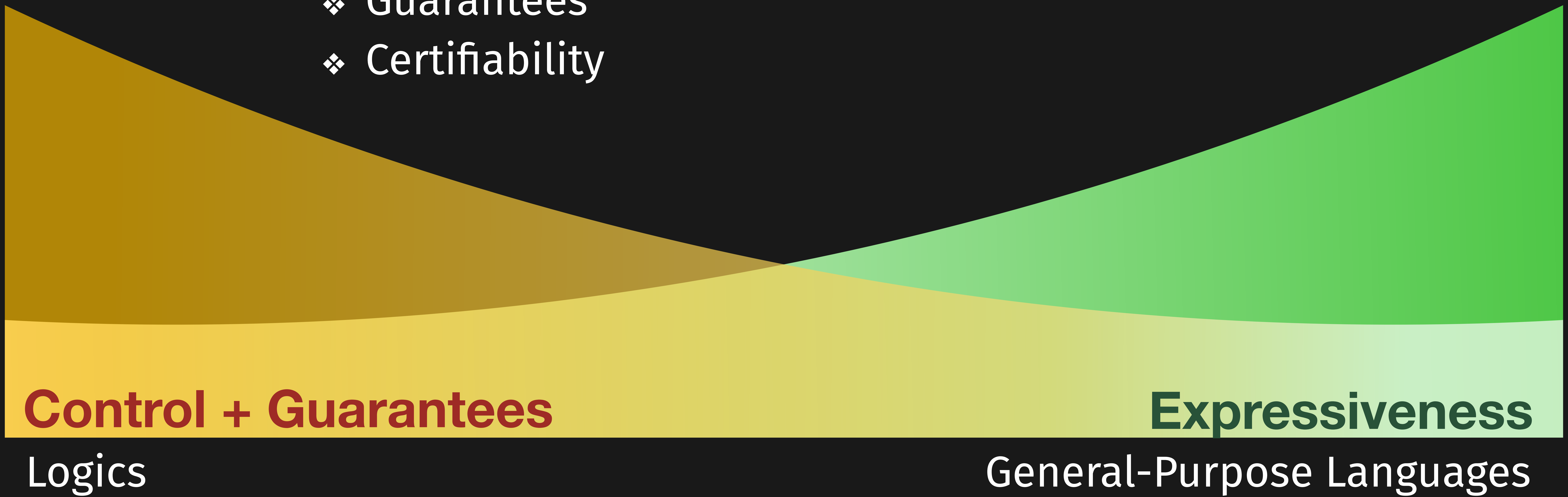
- Output Quality
- Expressiveness
- Integration
- Guarantees
- Certifiability

Logics General-Purpose Languages

One Language to Rule Them All?

Points to consider:

- ❖ Output Quality → Stream-based + Quantitative
- ❖ Expressiveness
- ❖ Integration
- ❖ Guarantees
- ❖ Certifiability



Block I One Language to Rule Them All?

Points to consider:

- Output Quality → Stream-based + Quantitative
- Expressiveness → Arithmetic + Clarity
- Integration
- Guarantees
- Certifiability

Logics

General-Purpose Languages

ONE LANGUAGE TO RULE THEM ALL?

Points to consider:

- ❖ Output Quality → Stream-based + Quantitative
- ❖ Expressiveness → Arithmetic + Clarity
- ❖ Integration → HW + SW compilation
- ❖ Guarantees
- ❖ Certifiability

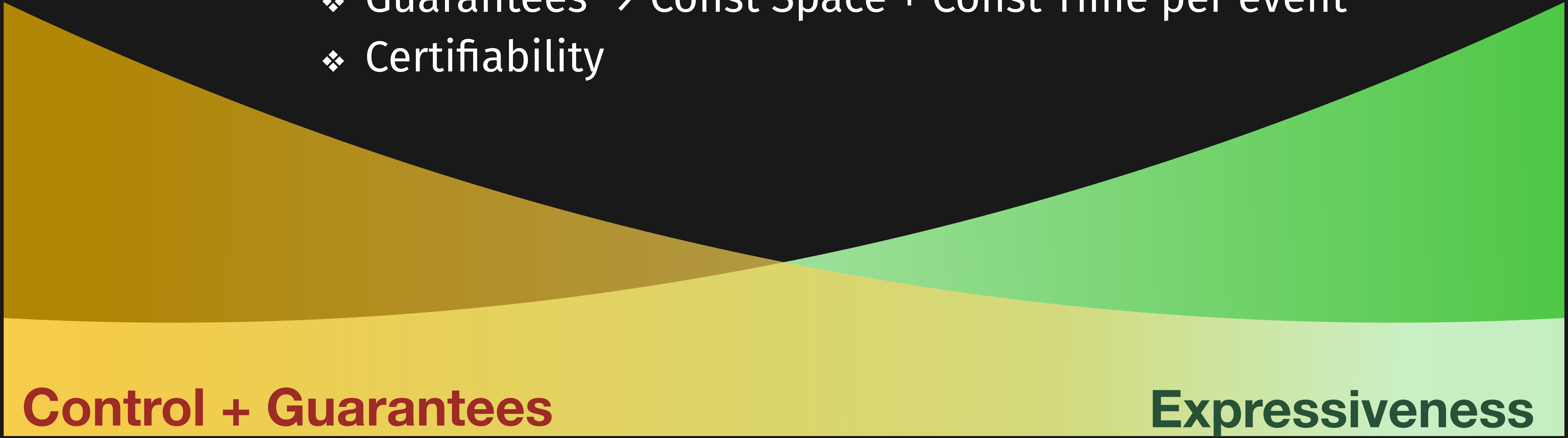
Logics

General-Purpose Languages

Block I One Language to Rule Them All?

Points to consider:

- ❖ Output Quality → Stream-based + Quantitative
- ❖ Expressiveness → Arithmetic + Clarity
- ❖ Integration → HW + SW compilation
- ❖ Guarantees → Const Space + Const Time per event
- ❖ Certifiability

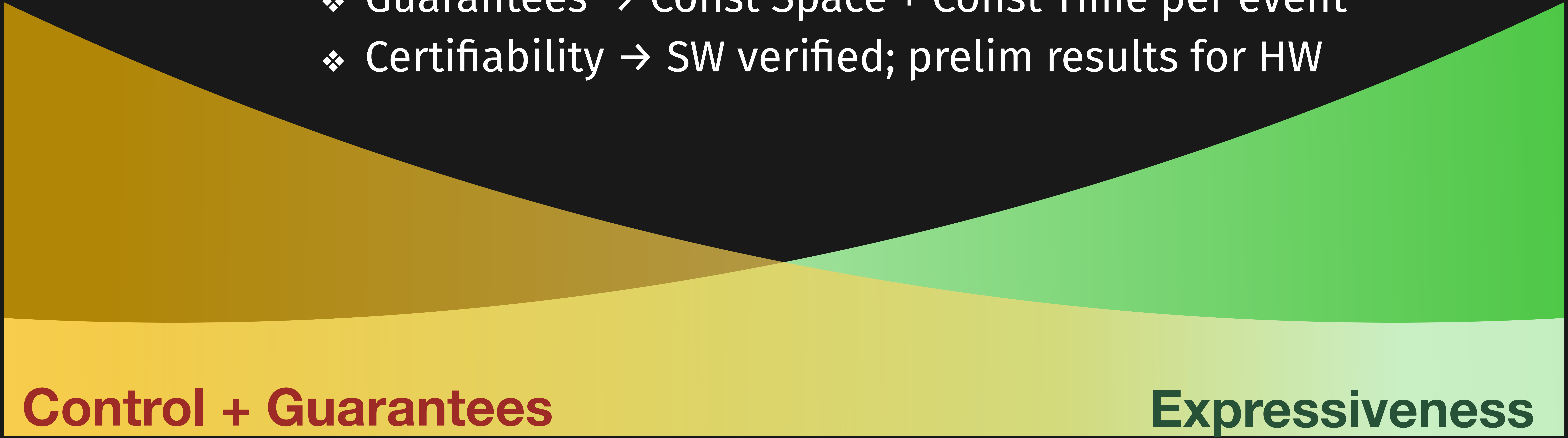


Logics General-Purpose Languages

Block I One Language to Rule Them All?

Points to consider:

- Output Quality → Stream-based + Quantitative
- Expressiveness → Arithmetic + Clarity
- Integration → HW + SW compilation
- Guarantees → Const Space + Const Time per event
- Certifiability → SW verified; prelim results for HW



Logics

General-Purpose Languages

RTLola by Example I

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 “Altimeter reports negative values.”
```

RTLola by Example I

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 “Altimeter reports negative values.”
```

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 ∨ read_ps < 9 “Barometer count irregular.”
```

RTLOLA BY EXAMPLE I

Sensor Validation 1: *Altimeter readings must be non-negative.*

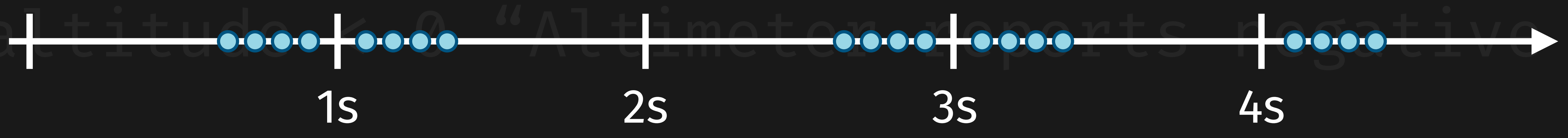
```
input altitude: Float32
trigger altitude < 0 “Altimeter reports negative values.”
```

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 v read_ps < 9 “Barometer count irregular.”
```

Block I RTLola by Example I

Barometer must produce 4 readings per second.



Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 ∨ read_ps < 9 “Barometer count irregular.”
```

Block I RTLola by Example I

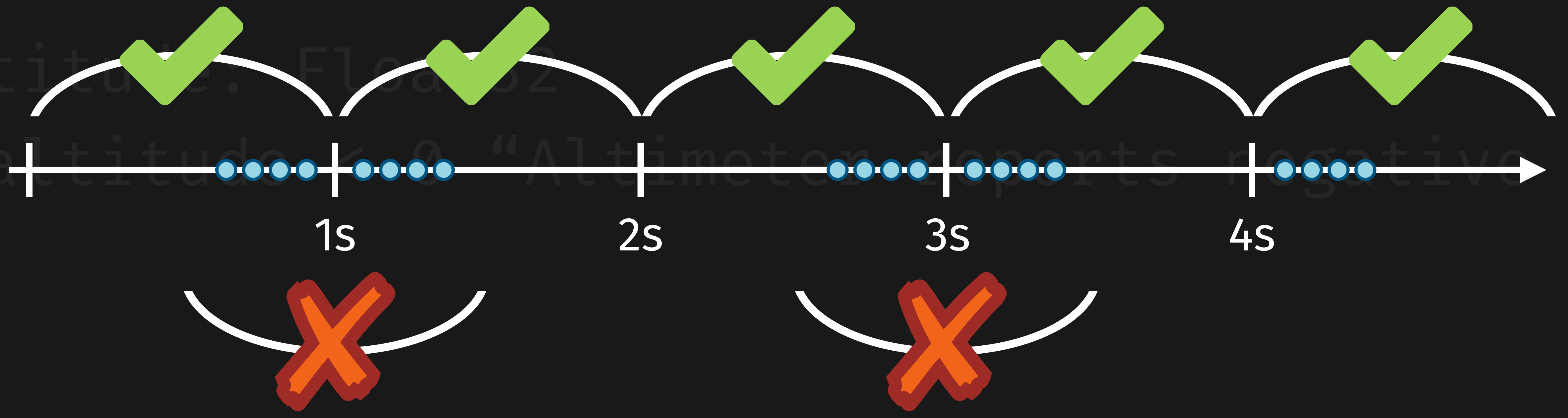
Barometer must produce 4 readings per second.

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 v read_ps < 9 “Barometer count irregular.”
```

RTLola by Example I

Barometer must produce 4 readings per second.

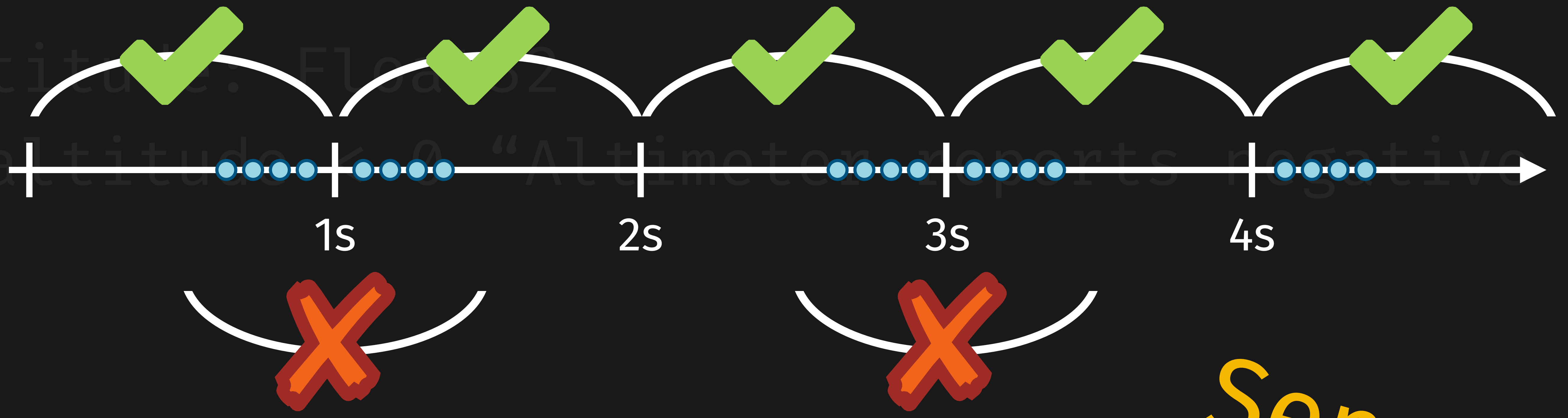


Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 ∨ read_ps < 9 "Barometer count irregular."
```

RTLola by Example I

Barometer must produce 4 readings per second.



Semantically different!

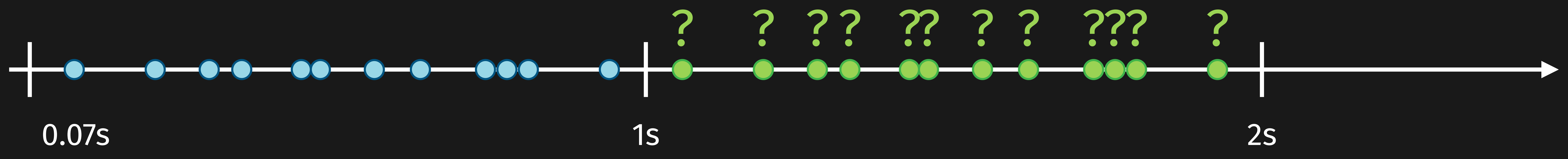
Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 ∨ read_ps < 9 “Barometer count irregular.”
```

Cost of Constant Memory

Every request needs to be granted within a second.

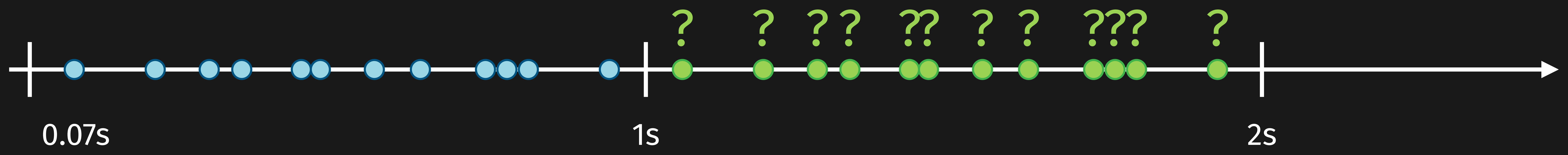
MTL: $\Box(r \rightarrow \Diamond_{1s}\, g)$



Block I   Cost of Constant Memory

Every request needs to be granted within a second.

MTL: $\Box(r \rightarrow \Diamond_{1s}\, g)$

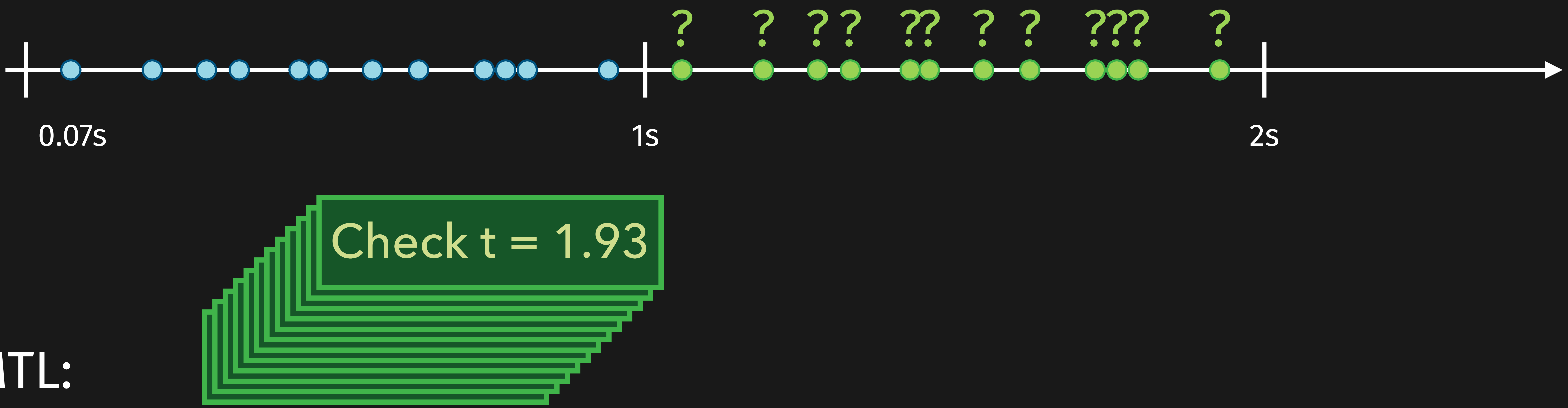


MTL:

Block I Cost of Constant Memory

Every request needs to be granted within a second.

MTL: $\Box(r \rightarrow \Diamond_{1s}\, g)$

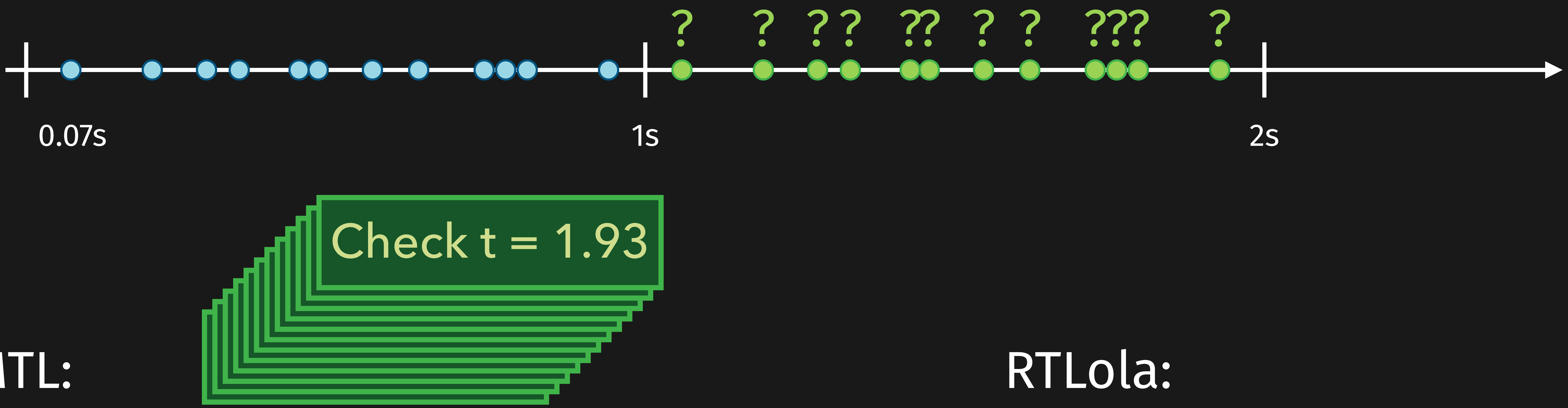


MTL:

Cost of Constant Memory

Every request needs to be granted within a second.

MTL: $\square(r \rightarrow \diamond_{1s}\, g)$



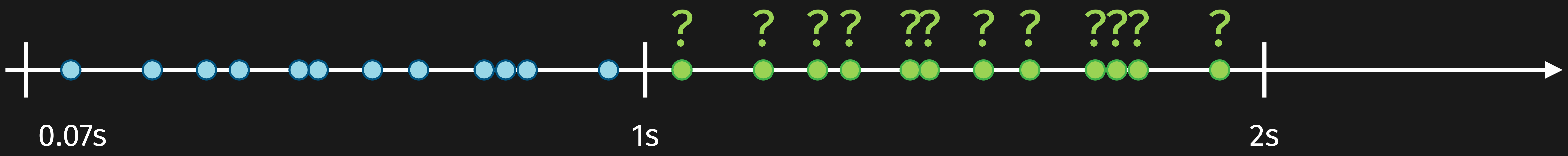
MTL:

RTLola:

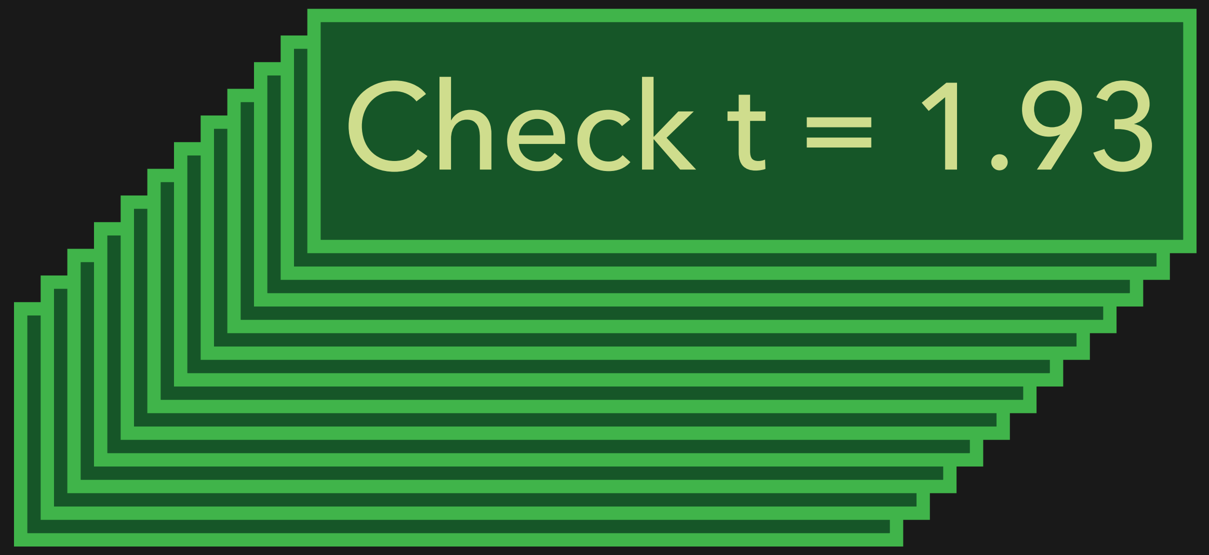
Cost of Constant Memory

Every request needs to be granted within a second.

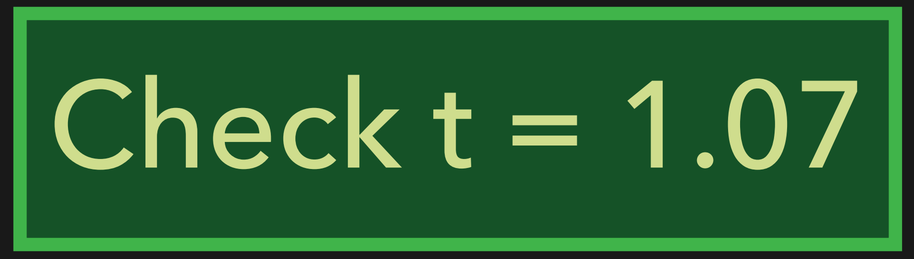
MTL: $\Box(r \rightarrow \Diamond_{1s}\, g)$



MTL:



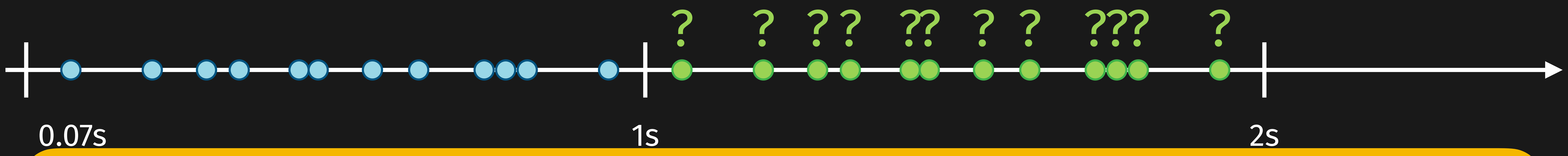
RTLola:



Block I — Cost of Constant Memory

Every request needs to be granted within a second.

MTL: $\square(r \rightarrow \Diamond_{1s}\, g)$

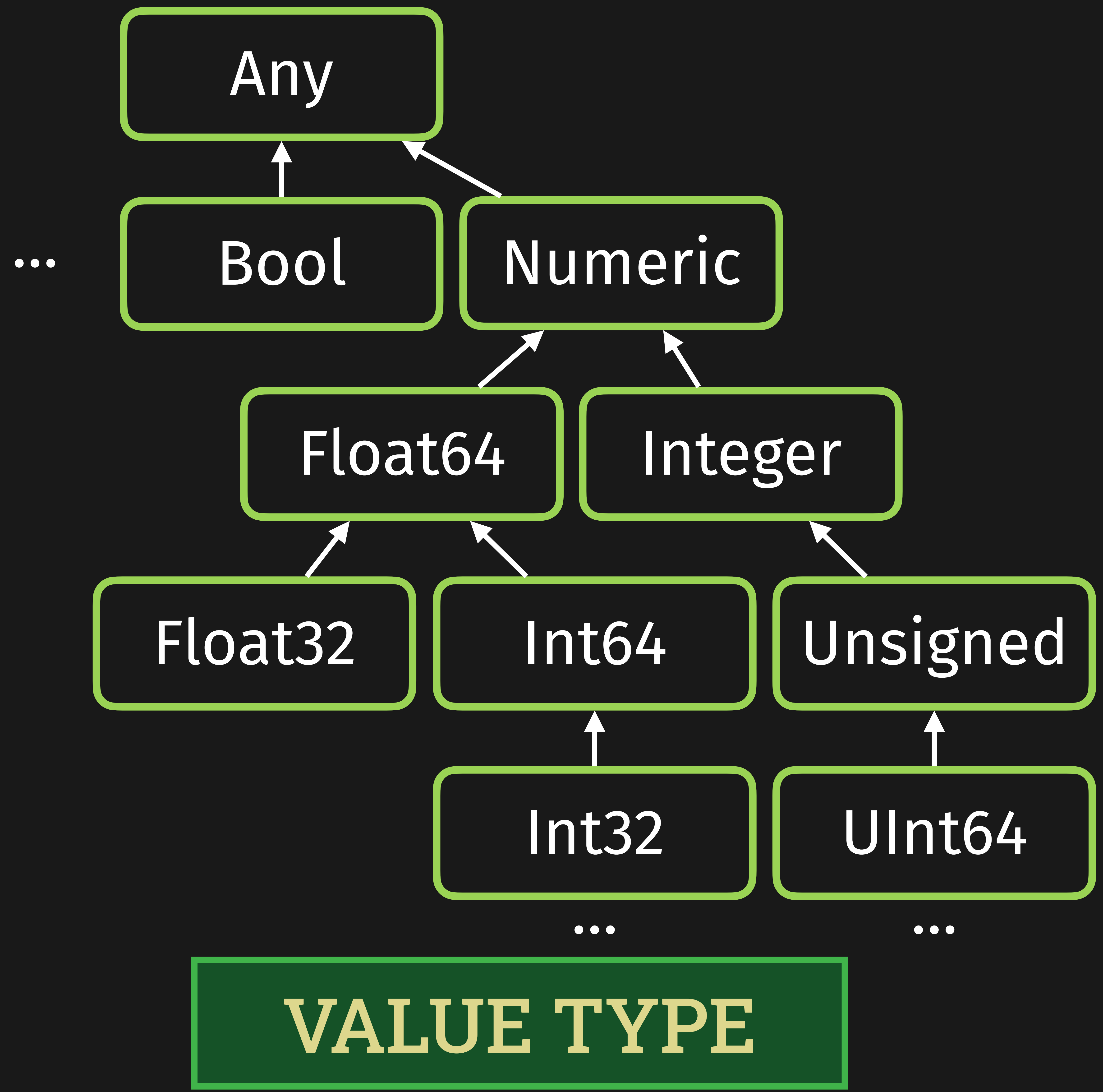


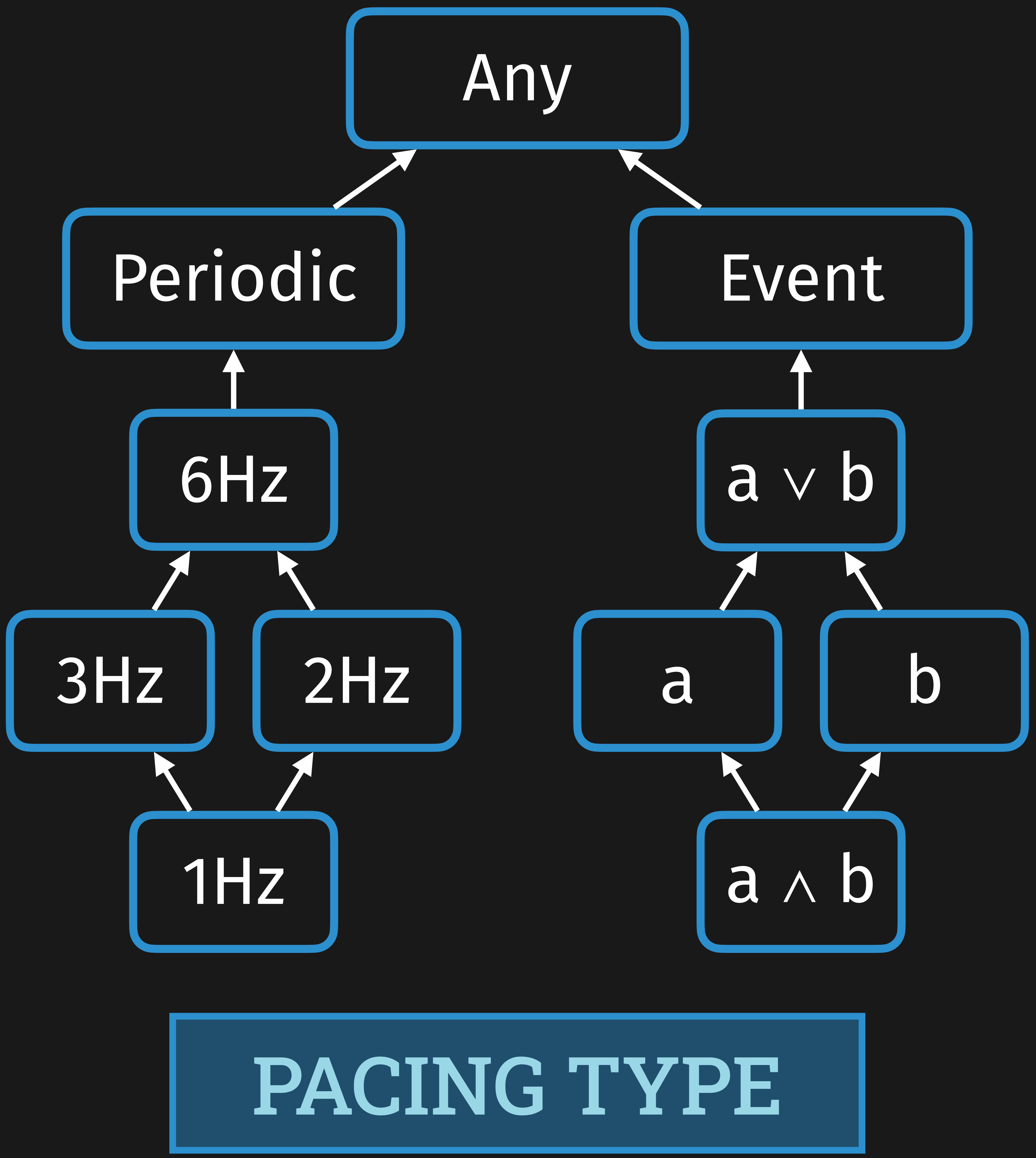
MTL:

RTLola restricts the specifier to enable static memory bound

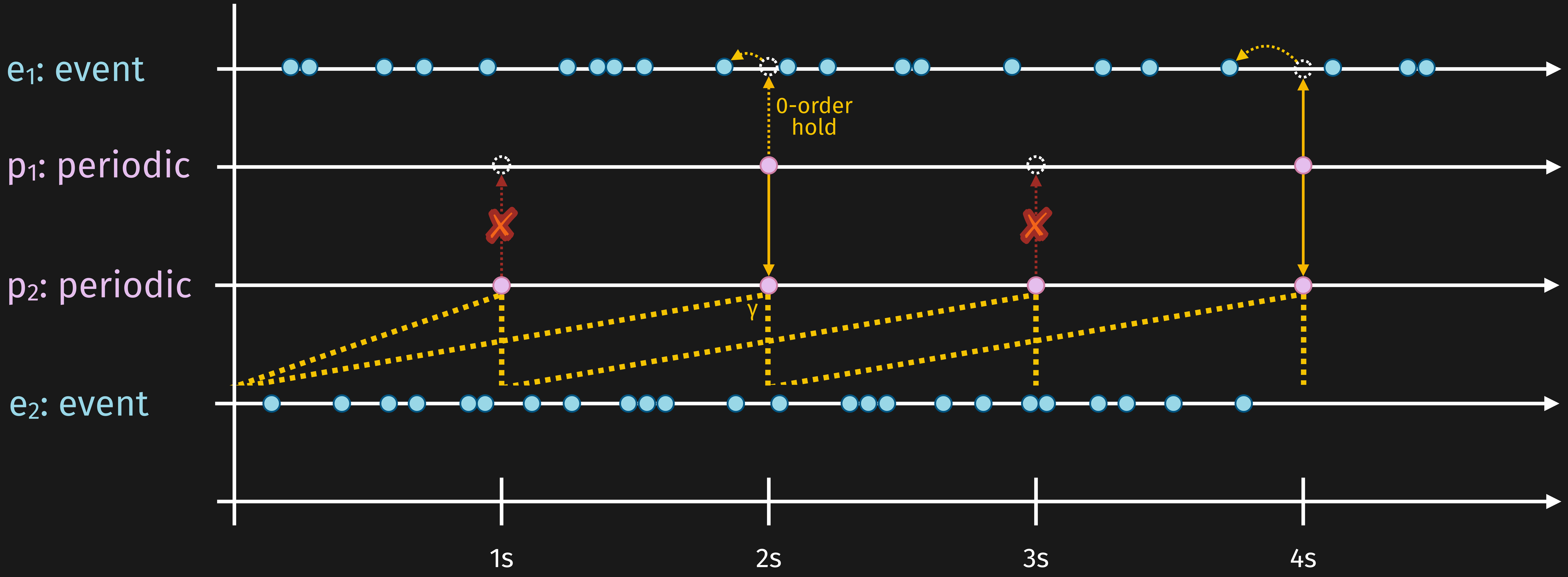
→ Enforcement Through Type System

Block I RTLola's Type System I

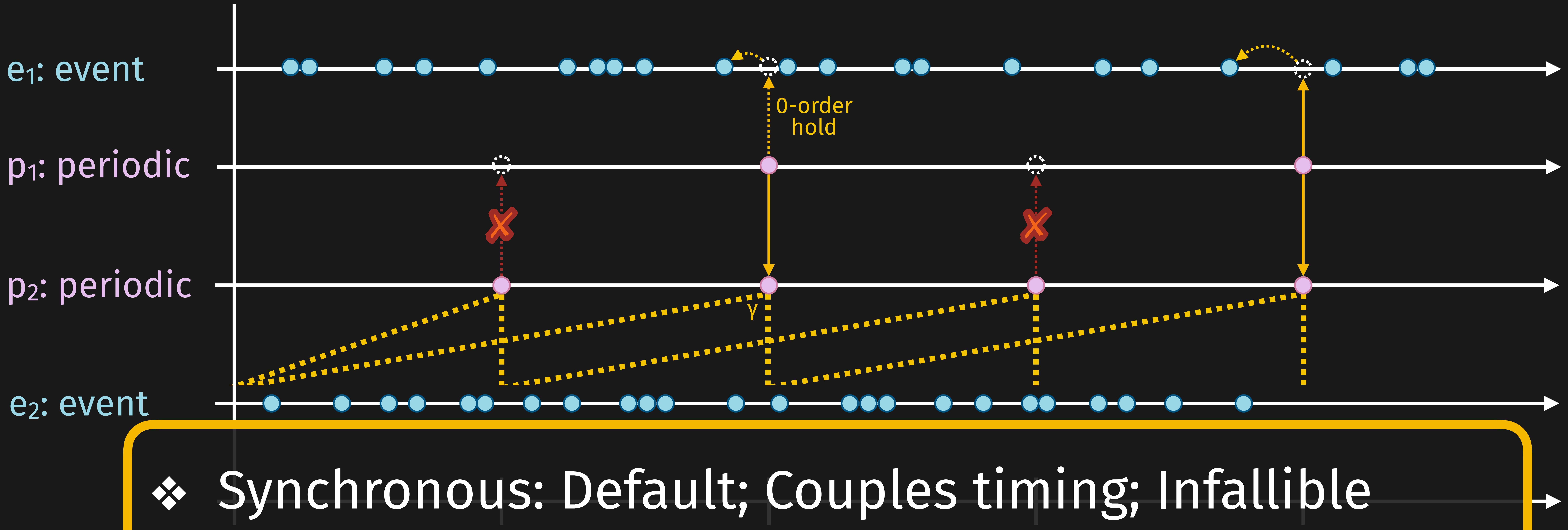




Block I 2-Dimensional Time



2-Dimensional Time



- ❖ Synchronous: Default; Couples timing; Infallible
- ❖ Holds + aggregations: Decouple timing; Fallible
- ❖ Aggregations only permitted in periodic streams

RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 “Altimeter reports negative values.”
```

RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 “Altimeter reports negative values.”
```

Float32 | altitude

RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 “Altimeter reports negative values.”
```

Float32 | altitude

Bool | altitude

RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 "Altimeter reports negative values."
```

Float32 | altitude

Bool | altitude

Ensures timeliness

RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32        Float32 | altitude
trigger altitude < 0 “Altimeter reports negative values.”
                                                  Bool | altitude
```

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 ∨ read_ps < 9 “Barometer count irregular.”
```

RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 "Altimeter reports negative values."
```

Float32 | altitude

Bool | altitude

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 ∨ read_ps < 9 "Barometer count irregular."
```

Float32 | pressure

RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 “Altimeter reports negative values.”
```

Float32 | altitude

Bool | altitude

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 v read_ps < 9 “Barometer count irregular.”
```

Float32 | pressure

UInt32 | 1Hz

Block I RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 "Altimeter reports negative values."
```

Float32 | altitude

Bool | altitude

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 ∨ read_ps < 9 "Barometer count irregular."
```

Float32 | pressure

UInt32 | 1Hz

Bool | 1Hz

Block I RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32                       Float32 | altitude
trigger altitude < 0 "Altimeter reports negative values."
                                                        Bool | altitude
```

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32   Float32 | pressure       UInt32 | 1Hz
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 v read_ps < 9 "Barometer count irregular."
             Bool | 1Hz

output x := pressure.aggregate(…)
output y := read_ps * pressure
```

Block I RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 "Altimeter reports negative values."
```

Float32 | altitude

Bool | altitude

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 v read_ps < 9 "Barometer count irregular."
```

Float32 | pressure

UInt32 | 1Hz

Bool | 1Hz

~~output x := pressure.aggregate(…)~~ aggregation w/o period

```
output y := read_ps * pressure
```

RTLola's Type System II

Sensor Validation 1: *Altimeter readings must be non-negative.*

```
input altitude: Float32
trigger altitude < 0 "Altimeter reports negative values."
```

Float32 | altitude

Bool | altitude

Sensor Validation 2: *Barometer must produce 9 — 11 readings per second.*

```
input pressure: Float32
output read_ps @ 1Hz := pressure.aggregate(over: 1s, using: count)
trigger read_ps > 11 v read_ps < 9 "Barometer count irregular."
```

Float32 | pressure

UInt32 | 1Hz

Bool | 1Hz

```
output x := pressure.aggregate(…)
output y := read_ps * pressure
```

(both lines struck through)

aggregation w/o period

mixes periodic and events

RTLola's Type System III

Int32 | velo_1   Int32 | velo_2

```
input velo_1, velo_2: Int32


output deviation                        := abs(velo_1 - velo_2)
```

Block I RTLola's Type System III

Int32 | velo_1 Int32 | velo_2

```
input velo_1, velo_2: Int32
```

UInt32 | velo_1 ∧ velo_2

```
output deviation  @ velo_1 ∧ velo_2 := abs(velo_1 - velo_2)
```

Block I RTLola's Type System III

Int32 | velo_1   Int32 | velo_2

```
input velo_1, velo_2: Int32
```

UInt32 | velo_1 ∧ velo_2

```
output deviation  @ velo_1 ∧ velo_2 := abs(velo_1 - velo_2)
```

UInt32 | velo_1 ∨ velo_2

```
output deviation’ @ velo_1 v velo_2
        := abs(velo_1              - velo_2          )
```

Block I RTLola's Type System III

Int32 | velo_1   Int32 | velo_2

```
input velo_1, velo_2: Int32
```

UInt32 | velo_1 ∧ velo_2

```
output deviation  @ velo_1 ∧ velo_2 := abs(velo_1 - velo_2)
```

UInt32 | velo_1 ∨ velo_2

```
output deviation’ @ velo_1 v velo_2
        := abs(velo_1.hold(or: 0) - velo_2.hold(or: 0))
```

Block I RTLola's Type System III

Int32 | velo_1   Int32 | velo_2

```
input velo_1, velo_2: Int32
```

UInt32 | velo_1 ∧ velo_2

```
output deviation  @ velo_1 ∧ velo_2 := abs(velo_1 - velo_2)
```

UInt32 | velo_1 ∨ velo_2

```
output deviation’ @ velo_1 v velo_2
        := abs(velo_1.hold(or: 0) - velo_2.hold(or: 0))
```

Int32 | Any   Int32 | Any

Block I RTLola's Type System III

Int32 | velo_1   Int32 | velo_2

```
input velo_1, velo_2: Int32
```

UInt32 | velo_1 ∧ velo_2

```
output deviation  @ velo_1 ∧ velo_2 := abs(velo_1 - velo_2)
```

UInt32 | velo_1 ∨ velo_2

```
output deviation' @ velo_1 ∨ velo_2
          := abs(velo_1.hold(or: 0) - velo_2.hold(or: 0))
```

Int32 | Any   Int32 | Any

Strong type system supports specifier.
→ increases confidence in spec.

RTLola by Example II

Mission Statistic: *Does the WP-distance correlate with the relative path deviation?*

```
input wp, pos: (Float64, Float64)
```

Block I RTLola by Example II

Mission Statistic: *Does the WP-distance correlate with the relative path deviation?*

```
input wp, pos: (Float64, Float64)

output wp_dist := abs(wp - wp.offset(by: -1, dft: wp))
```

RTLola by Example II

Mission Statistic: *Does the WP-distance correlate with the relative path deviation?*

```
input wp, pos: (Float64, Float64)

output wp_dist := abs(wp - wp.offset(by: -1, dft: wp))
output dist_total := pos - pos.offset(by: -1, dft: START)
                         + dist_total.offset(by: -1, dft: 0)
```

Block I RTLola by Example II

Mission Statistic: *Does the WP-distance correlate with the relative path deviation?*

```
input wp, pos: (Float64, Float64)

output wp_dist := abs(wp - wp.offset(by: -1, dft: wp))
output dist_total := pos - pos.offset(by: -1, dft: START)
                        + dist_total.offset(by: -1, dft: 0)
output total_dist_at_wp @ wp := dist_total.hold(or: 0)
```

Block I RTLola by Example II

Mission Statistic: *Does the WP-distance correlate with the relative path deviation?*

```
input wp, pos: (Float64, Float64)

output wp_dist := abs(wp - wp.offset(by: -1, dft: wp))
output dist_total := pos - pos.offset(by: -1, dft: START)
                        + dist_total.offset(by: -1, dft: 0)
output total_dist_at_wp @ wp := dist_total.hold(or: 0)
output devi @ wp := abs( wp_dist.offset(by: -1, dft: 0) -
    (total_dist_at_wp - total_dist_at_wp.offset(by: -1, dft: 0) )
```

RTLola by Example II

Mission Statistic: *Does the WP-distance correlate with the relative path deviation?*

```
input wp, pos: (Float64, Float64)

output wp_dist := abs(wp - wp.offset(by: -1, dft: wp))
output dist_total := pos - pos.offset(by: -1, dft: START)
                        + dist_total.offset(by: -1, dft: 0)
output total_dist_at_wp @ wp := dist_total.hold(or: 0)
output devi @ wp := abs( wp_dist.offset(by: -1, dft: 0) -
    (total_dist_at_wp - total_dist_at_wp.offset(by: -1, dft: 0) )

output dist_v_devi @ wp := (wp_dist, devi)
output cov @ 1Hz := dist_v_devi.aggregate(over: ∞, using: cov)
output var_dist @ 1Hz := wp_dist.aggregate(over: ∞, using: var)
output var_devi @ 1Hz := devi.aggregate(over: ∞, using: var)
output corr := cov / (var_devi^2 * var_dist^2)
```

RTLola by Example II

Mission Statistic: *Does the WP-distance correlate with the relative path deviation?*

```
input wp, pos: (Float64, Float64)

output wp_dist := abs(wp - wp.offset(by: -1, dft: wp))
output dist_total := pos - pos.offset(by: -1, dft: START)
                        + dist_total.offset(by: -1, dft: 0)
output total_dist_at_wp @ wp := dist_total.hold(or: 0)
output devi @ wp := abs( wp_dist.offset(by: -1, dft: 0) -
    (total_dist_at_wp - total_dist_at_wp.offset(by: -1, dft: 0) )

output dist_v_devi @ wp := (wp_dist, devi)
output cov @ 1Hz := dist_v_devi.aggregate(over: ∞, using: cov)
output var_dist @ 1Hz := wp_dist.aggregate(over: ∞, using: var)
output var_devi @ 1Hz := devi.aggregate(over: ∞, using: var)
output corr := cov / (var_devi^2 * var_dist^2)
```

RTLola provides primitives for abstract, mission-level properties.

Great, we've got a spec and the type checker is happy.

What next?

A) Further increase confidence

B) Analyze complexity

Validation via Interpretation

Block I Validation via Interpretation

Specification:

GPS frequency validation

GPS/IMU jump detection

Hover phase detection

Results:

433,000 events

1,545ns per event @ 146%

Stack size < 1kB, no heap

Block I — Validation via Interpretation

Specification:

GPS frequency validation

GPS/IMU jump detection

Hover phase detection

Results:

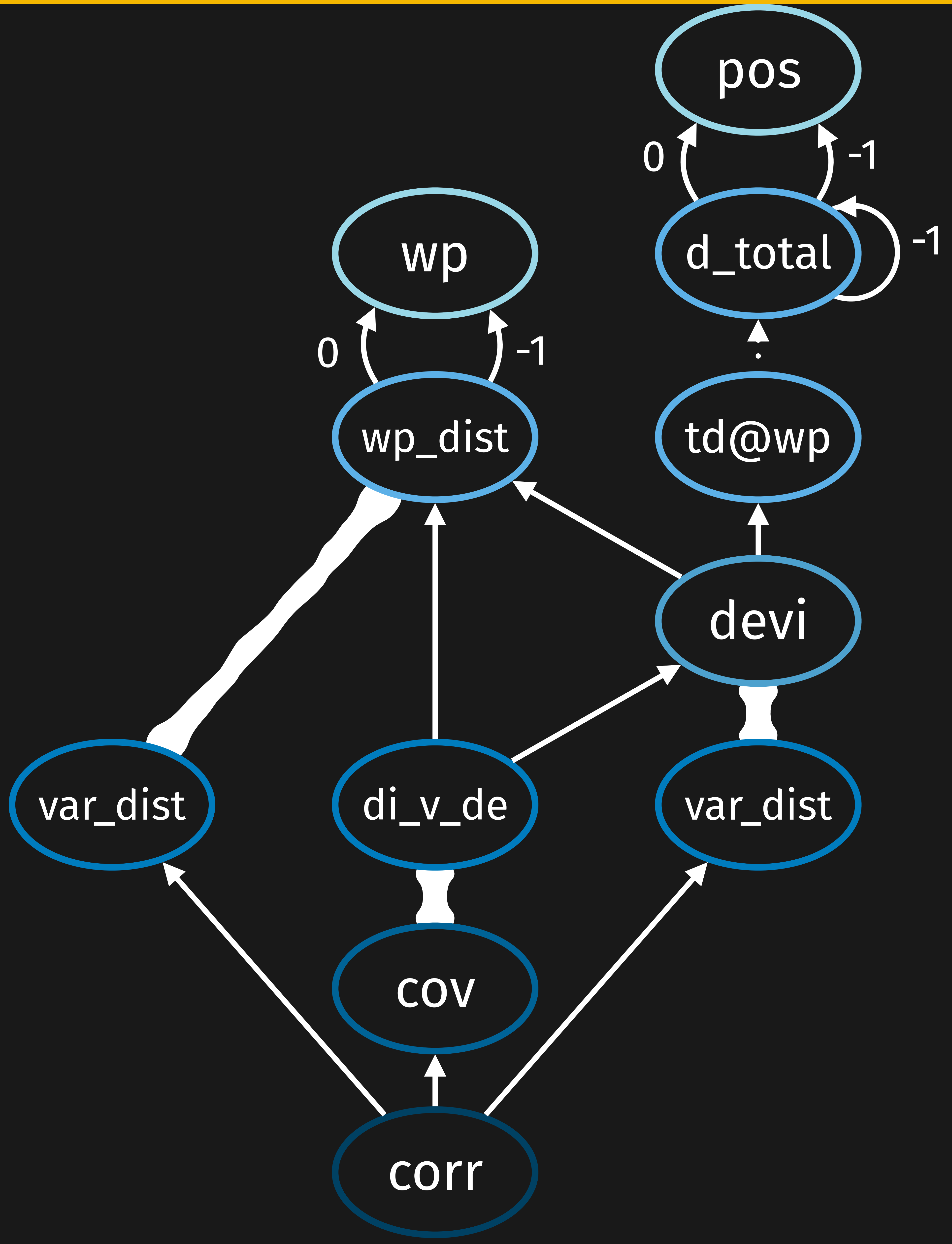
433,000 events

1,545ns per event @ 146%

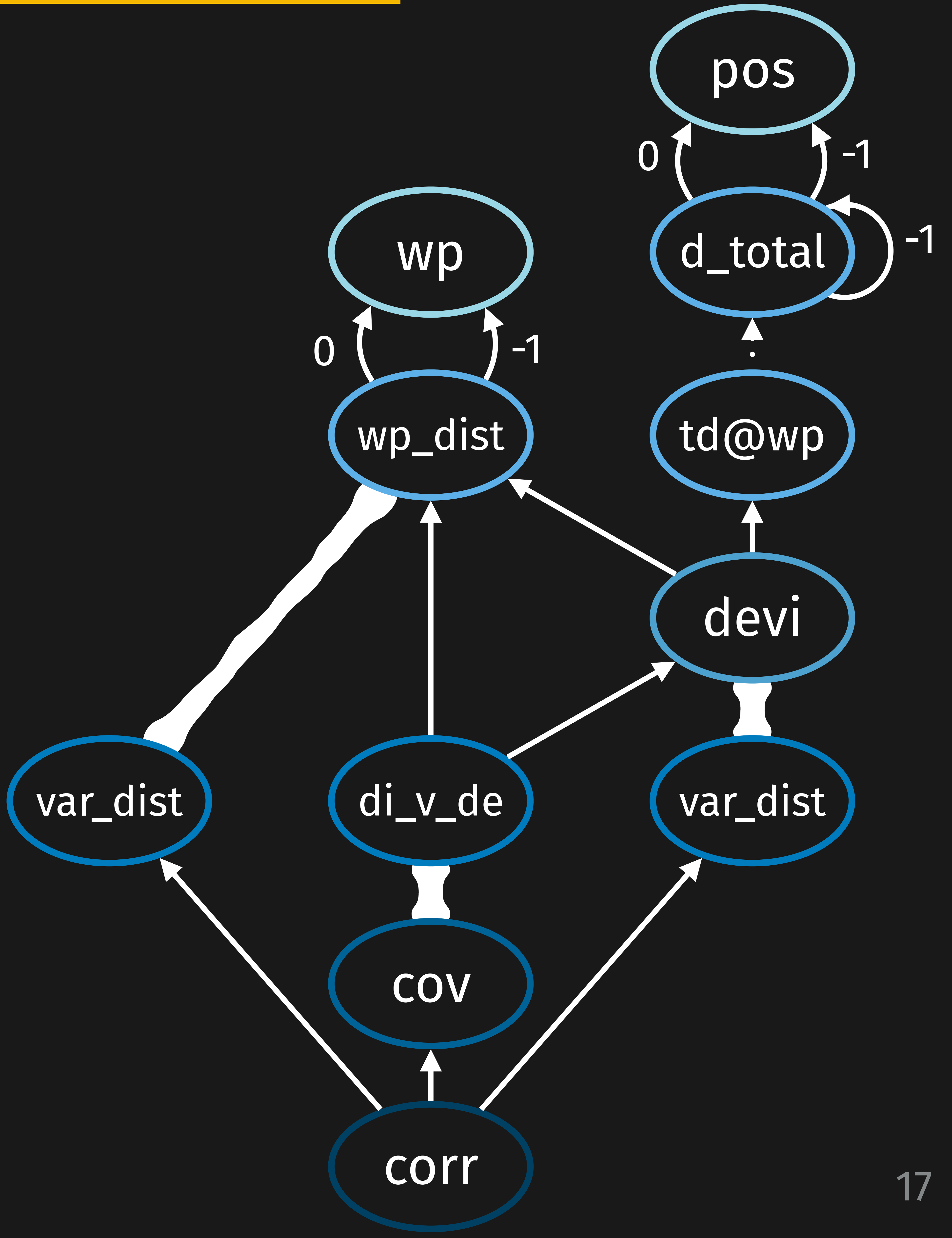
Stack size < 1kB, no heap

Enables rapid development.

Block I Specification Analysis

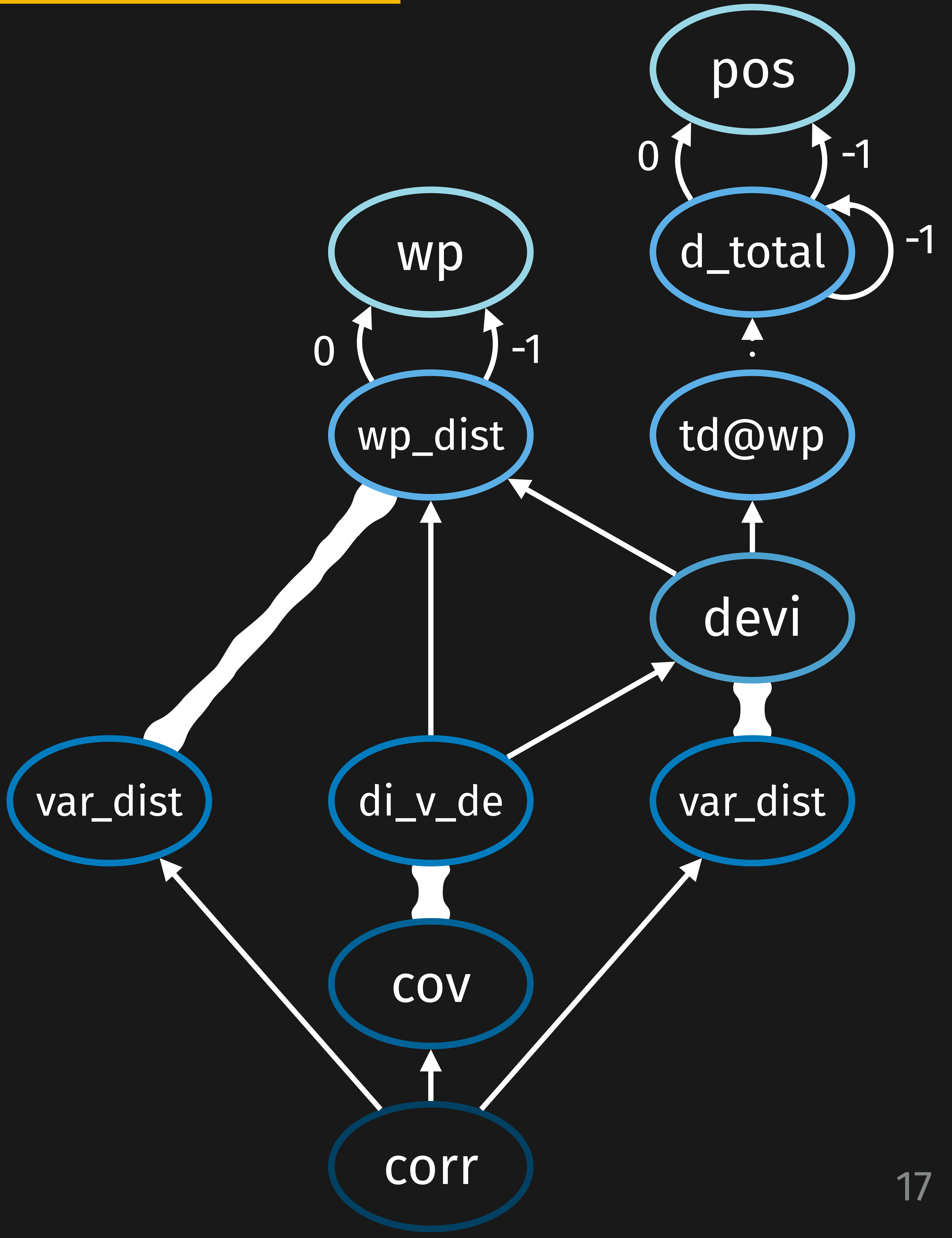


Block I   Specification Analysis



Block I Specification Analysis

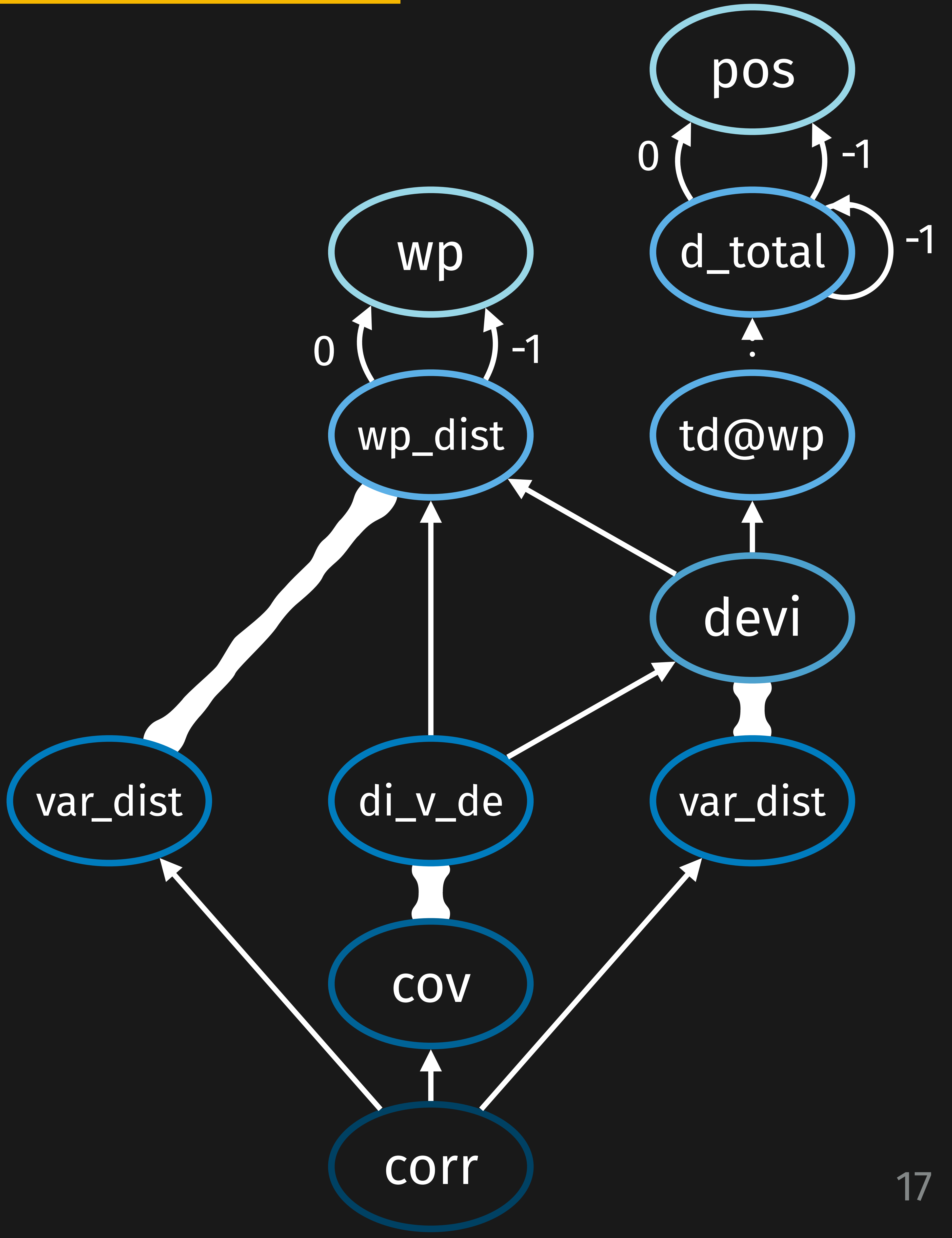
Stream	#values	Size	Windows	Total
pos	2	128		256
wp	2	128		256
d_total	2	64		128
wp_dist	1	64		64
d_s_wp	1	64		64
devi	1	64		64
var_dist	1	64	128	192
di_v_de	1	64		64
var_dist	1	64	128	192
cov	1	64	128	192
corr	1	64		64



Block I Specification Analysis

Stream	#values	Size	Windows	Total
pos	2	128		256
wp	2	128		256
d_total	2	64		128
wp_dist	1	64		64
d_s_wp	1	64		64
devi	1	64		64
var_dist	1	64	128	192
di_v_de	1	64		64
var_dist	1	64	128	192
cov	1	64	128	192
corr	1	64		64

Σ 1536B



Block I Summary RTLola

BLOCK I SUMMARY RTLOLA

	Sensor Level		Mission Level
Timeliness	Critical		Lax
Arithmetic Challenge	Low (bounds checks, counting)		High (statistics, prediction)
Input Data	Raw		Refined
Locality	Local	Inter-component	System-wide
Example	Data Validation: *"Altimeter must produce positives values."*		Mission Statistics: *"Low correlation between WP distance and relative path deviation."*

Block I Summary RTLola

Time

Arith
Chall

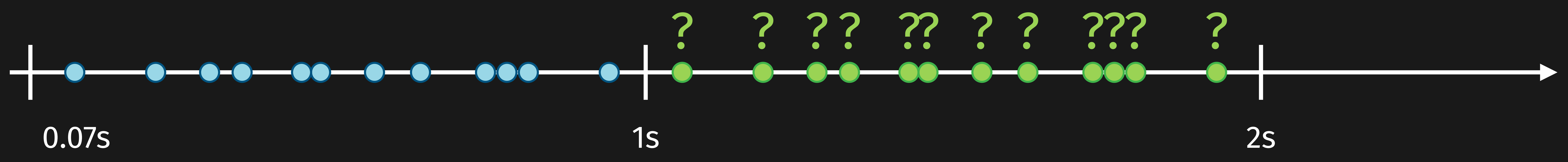
Input

Loc

Exar

Every request needs to be granted within a second.

MTL: $\square(r \rightarrow \Diamond_{1s}\, g)$



Check t = 1.93

MTL:

RTLola: Check t = 1.07

Block I Summary RTLola

Time

Arith
Chall

Input

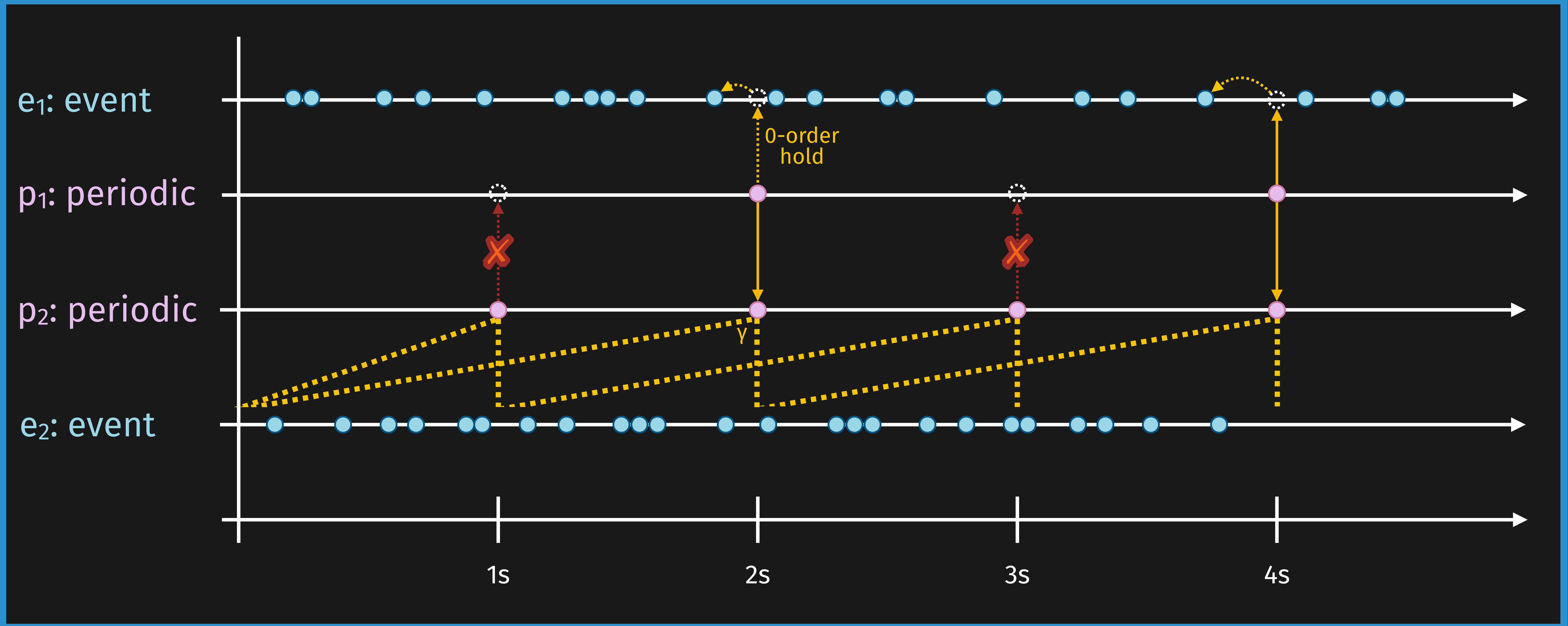
Loc

0.0

Exar

Every request needs to be granted within a second.

MTL:



Block I Summary RTLola

Time

Arith
Chall

Input

Loc

Exar

Every request needs to be granted within a second.

e_1: ev

p_1: pe

p_2: pe

0.0

e_2: ev

MTL:

Specification:

GPS frequency validation

GPS/IMU jump detection

Hover phase detection

Results:

433,000 events

1,545ns per event @ 146%

Stack size < 1kB, no heap

Block I Summary RTLola

Every request needs to be granted within a second.

Time

Arith Chall

Input

Loc

Exar

0.0

MTL:

e_1: ev

p_1: pe

p_2: pe

e_2: ev

Spe

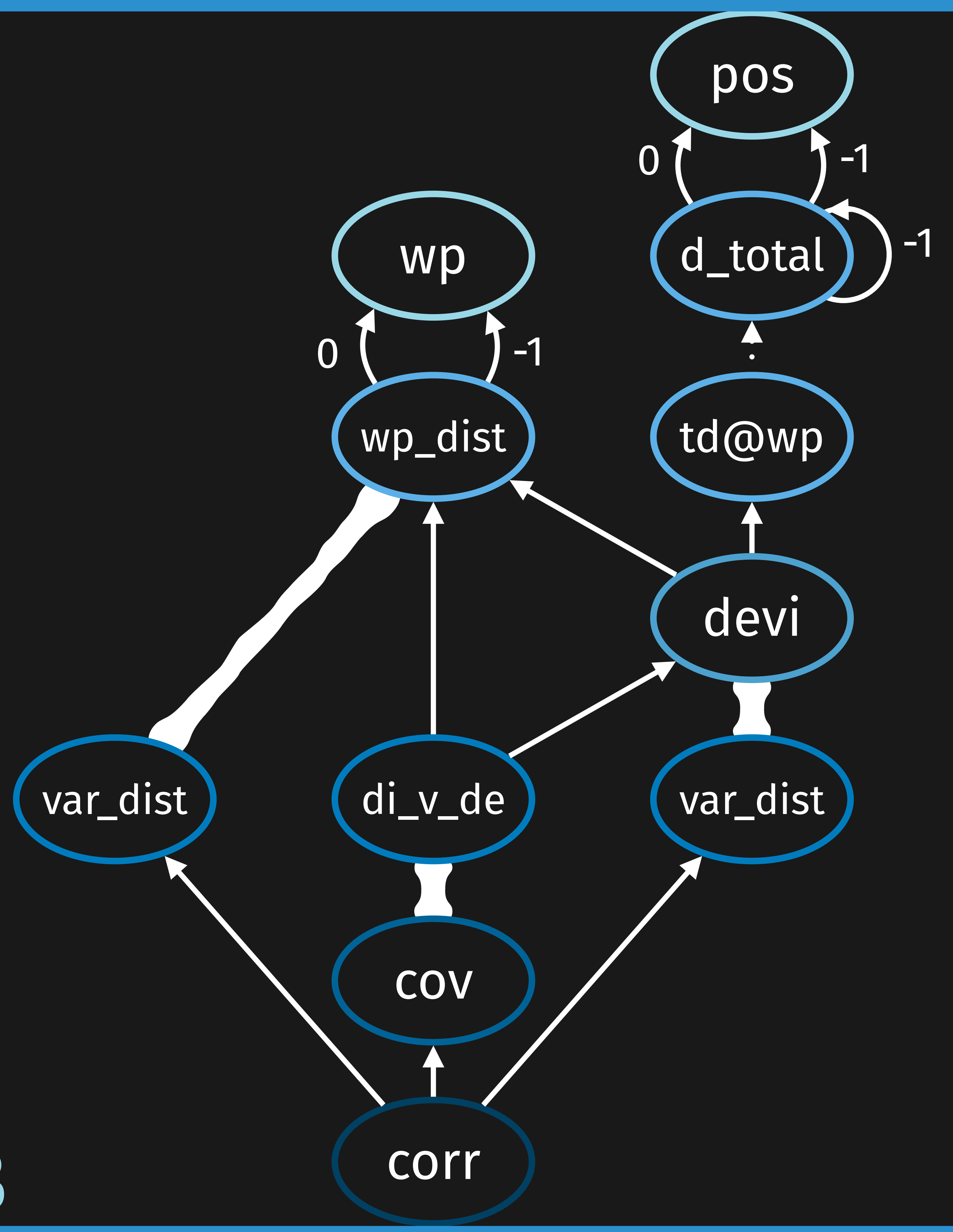
GPS f

GPS/

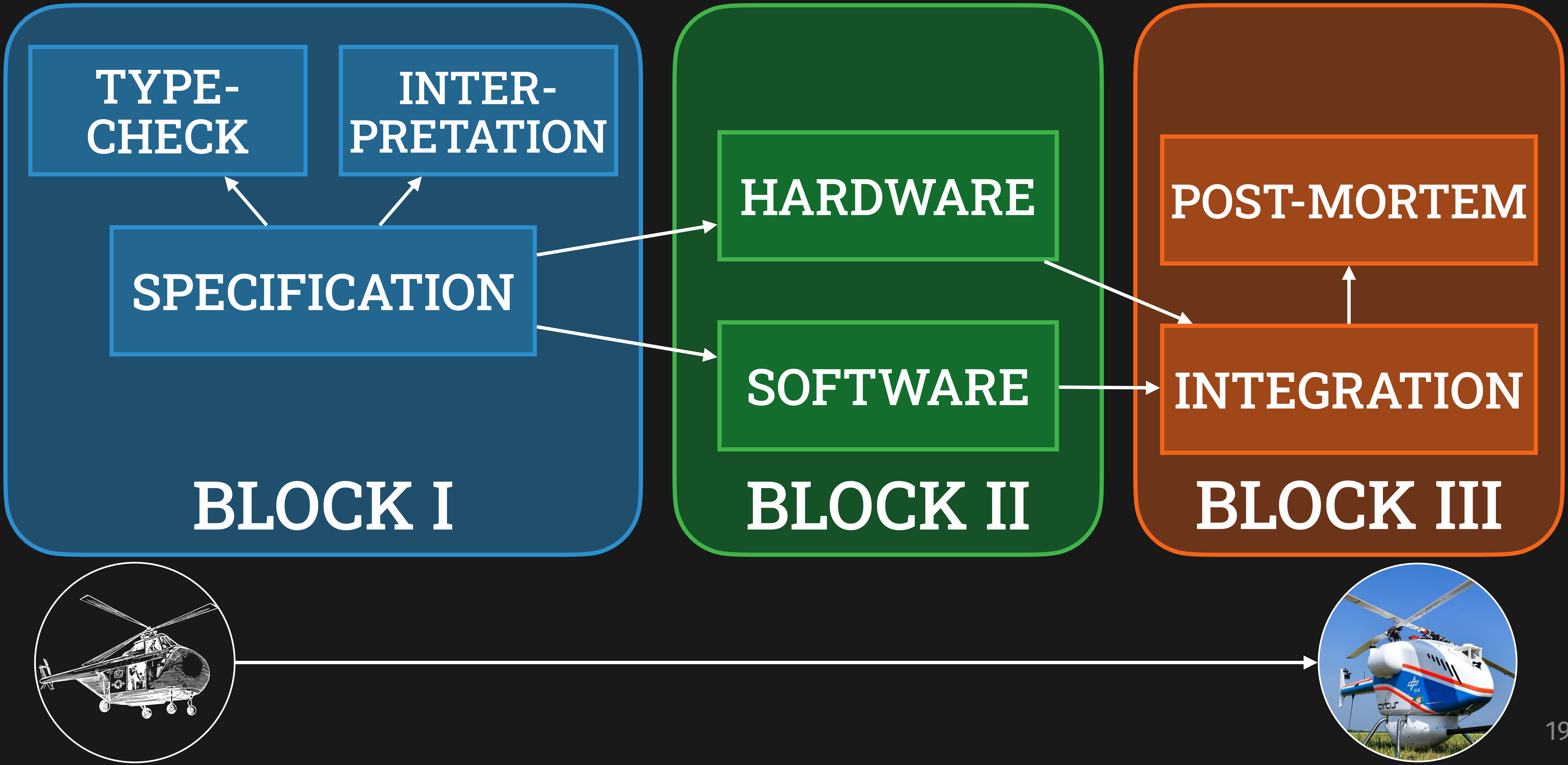
Hove

Stream	#values	Size	Windows	Total
pos	2	128		256
wp	2	128		256
d_total	2	64		128
wp_dist	1	64		64
d_s_wp	1	64		64
devi	1	64		64
var_dist	1	64	128	192
di_v_de	1	64		64
var_dist	1	64	128	192
cov	1	64	128	192
corr	1	64		64

Σ 1536B



Block II Overview



Block II Interpretation v Compilation

Block II Interpretation v Compilation

Observes

Monitor

Compilation

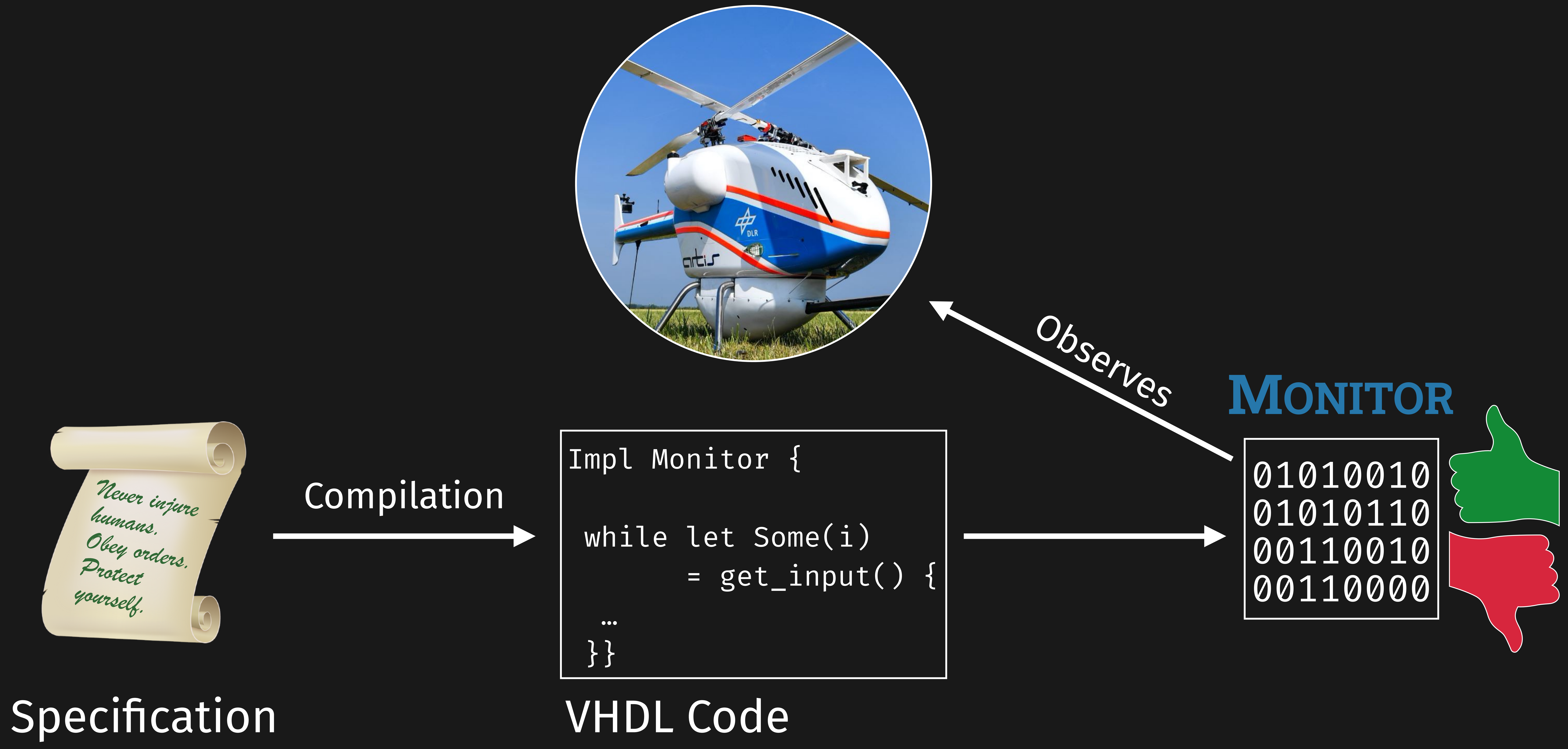
```
Impl Monitor {

 while let Some(i)
       = get_input() {
  …
 }}
```

```
01010010
01010110
00110010
00110000
```

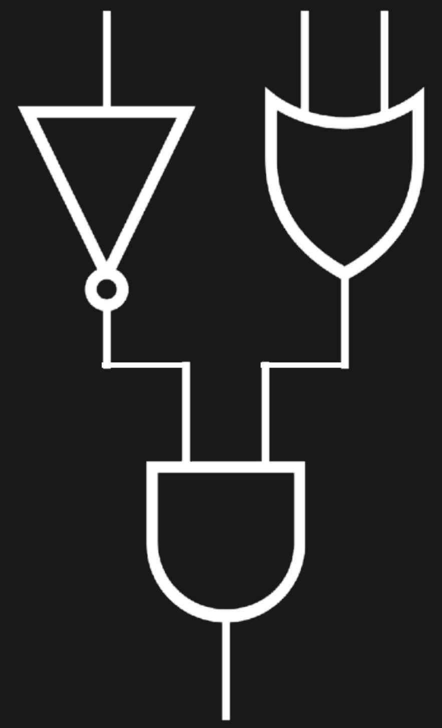
Specification

High Level Code

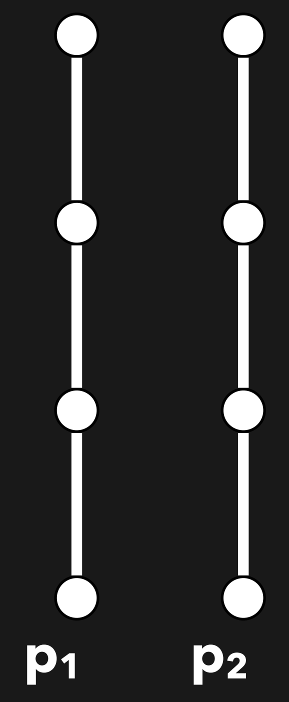
Block II Interpretation v Compilation



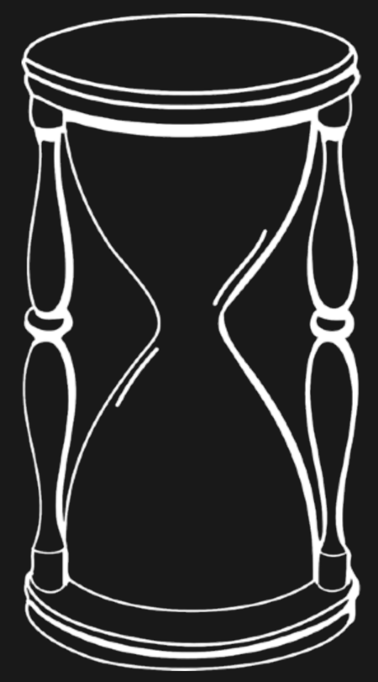
Block II Challenges



Reduce Circuit Cost

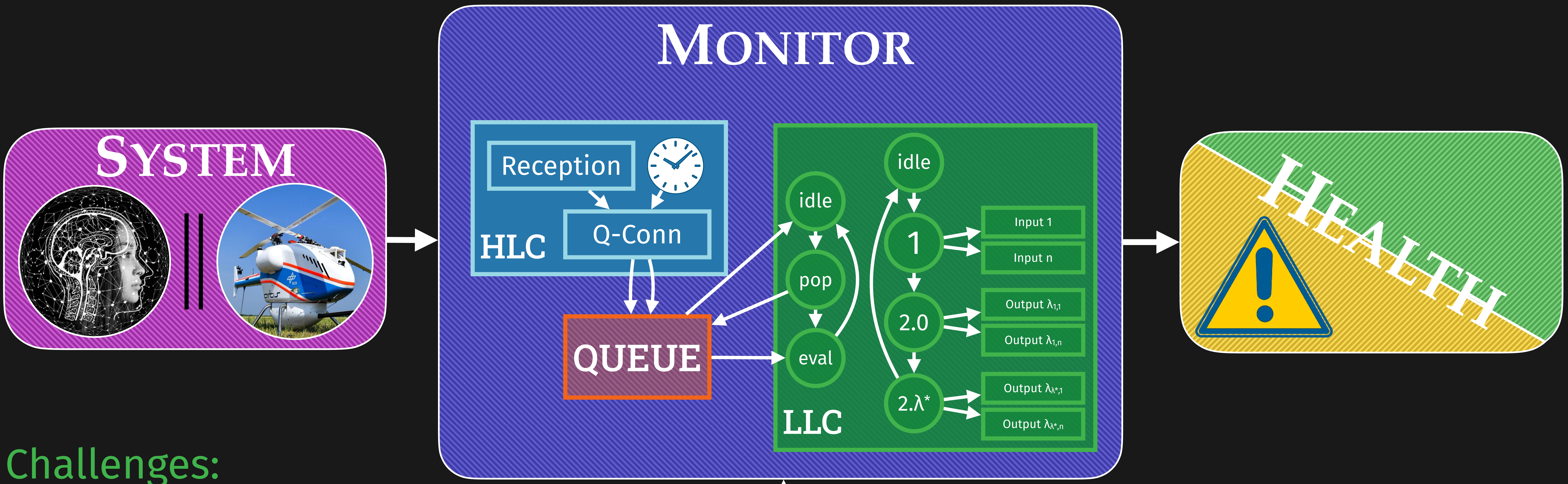


Utilize Parallel Execution

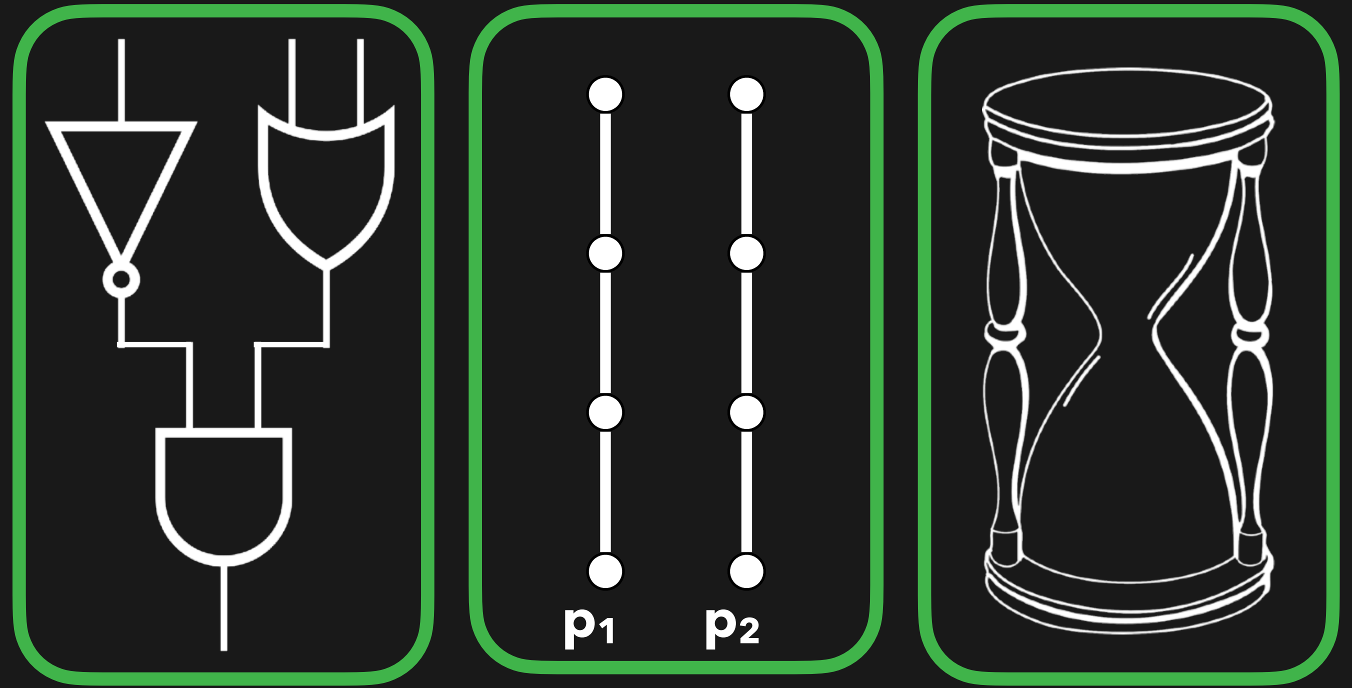


Periodic versus Event-Based

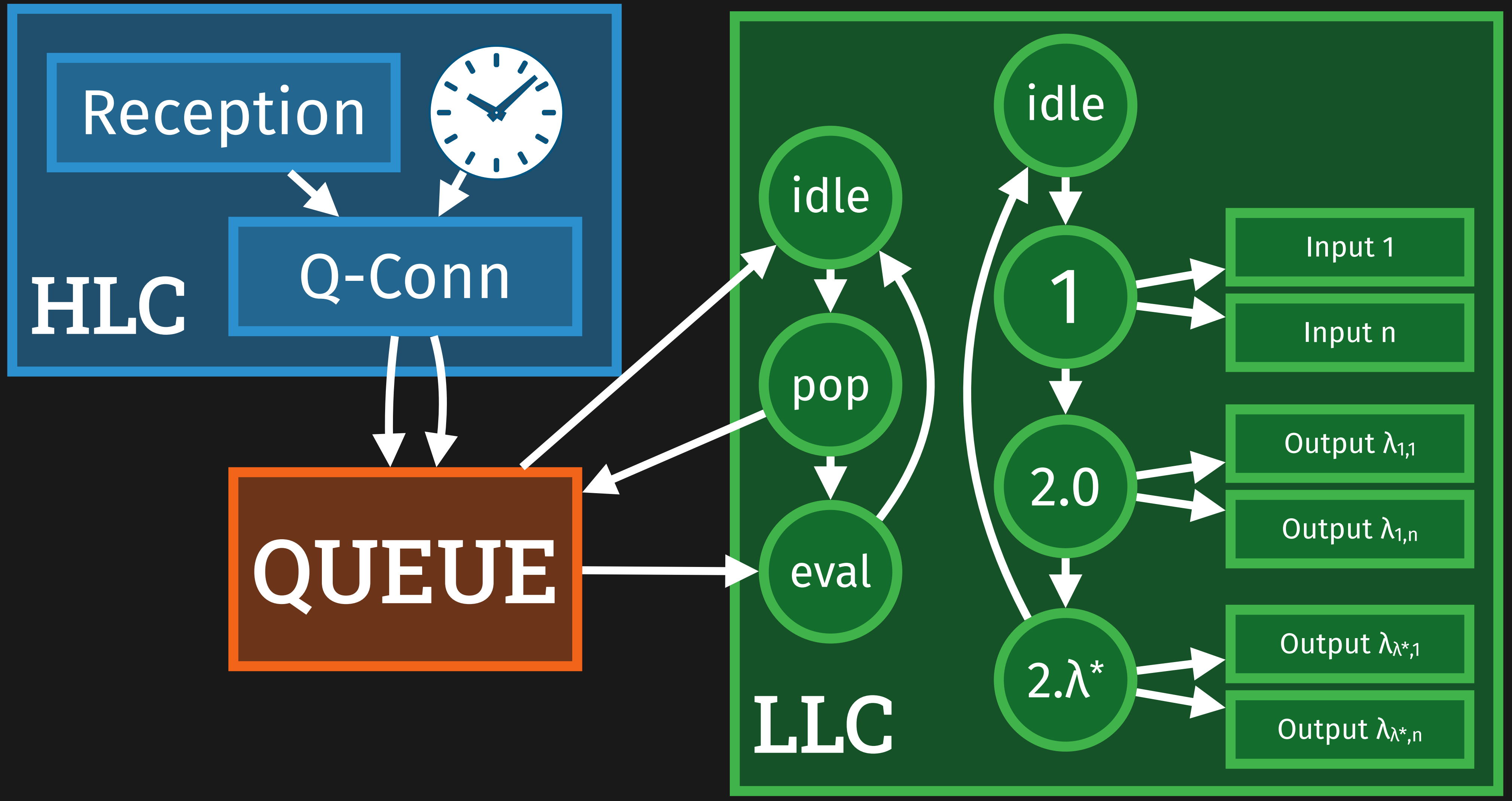
Block II VHDL Compilation Overview



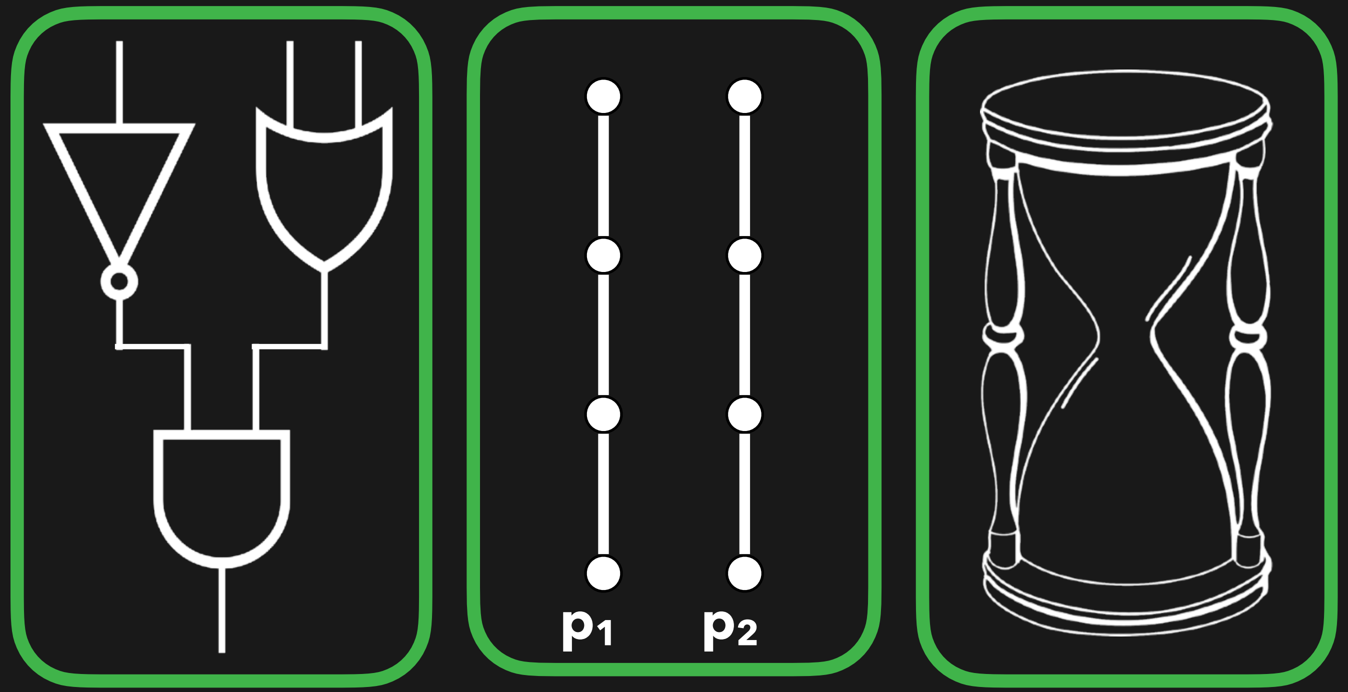
Challenges:



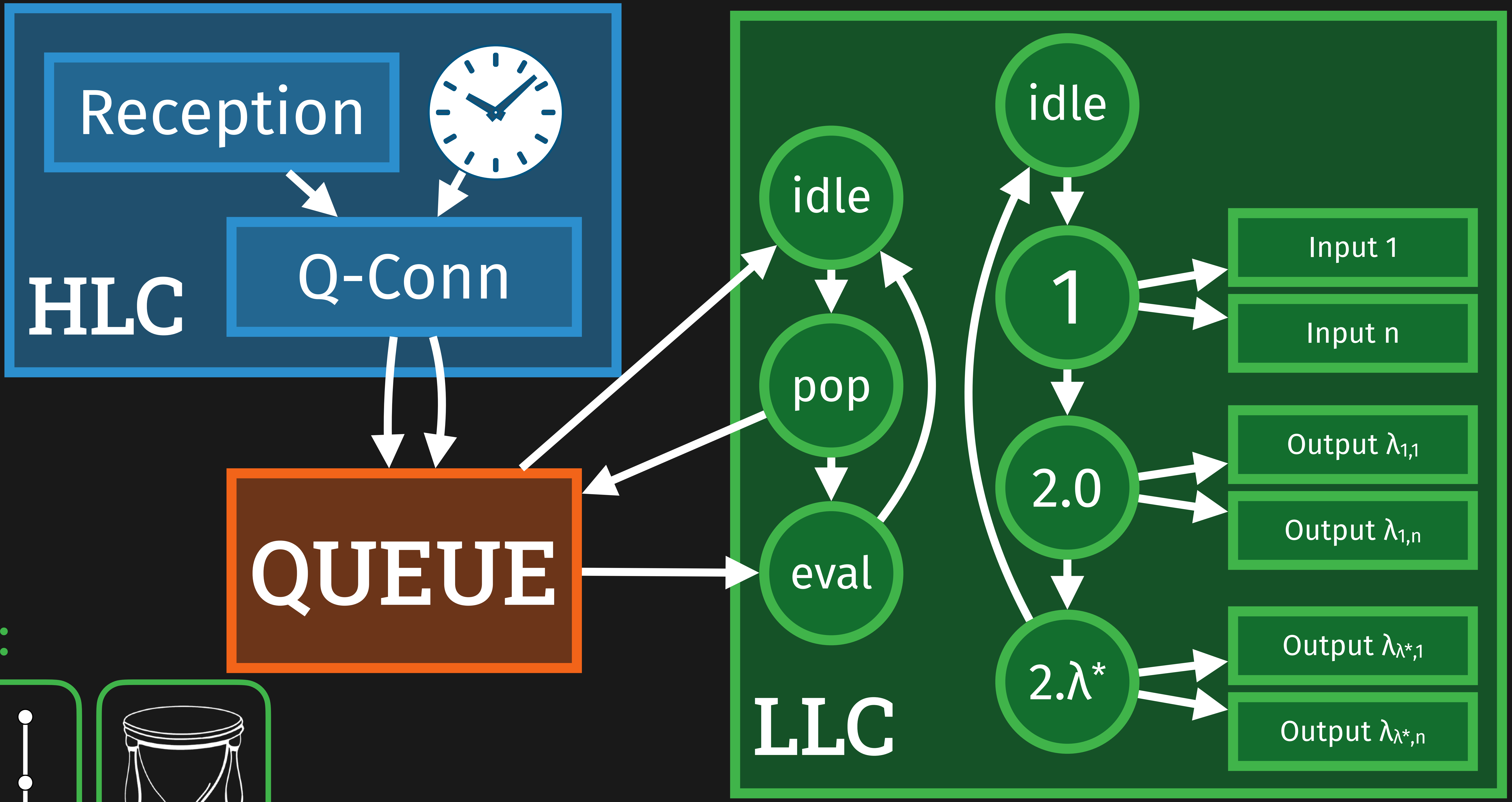
Block II VHDL Compilation Overview



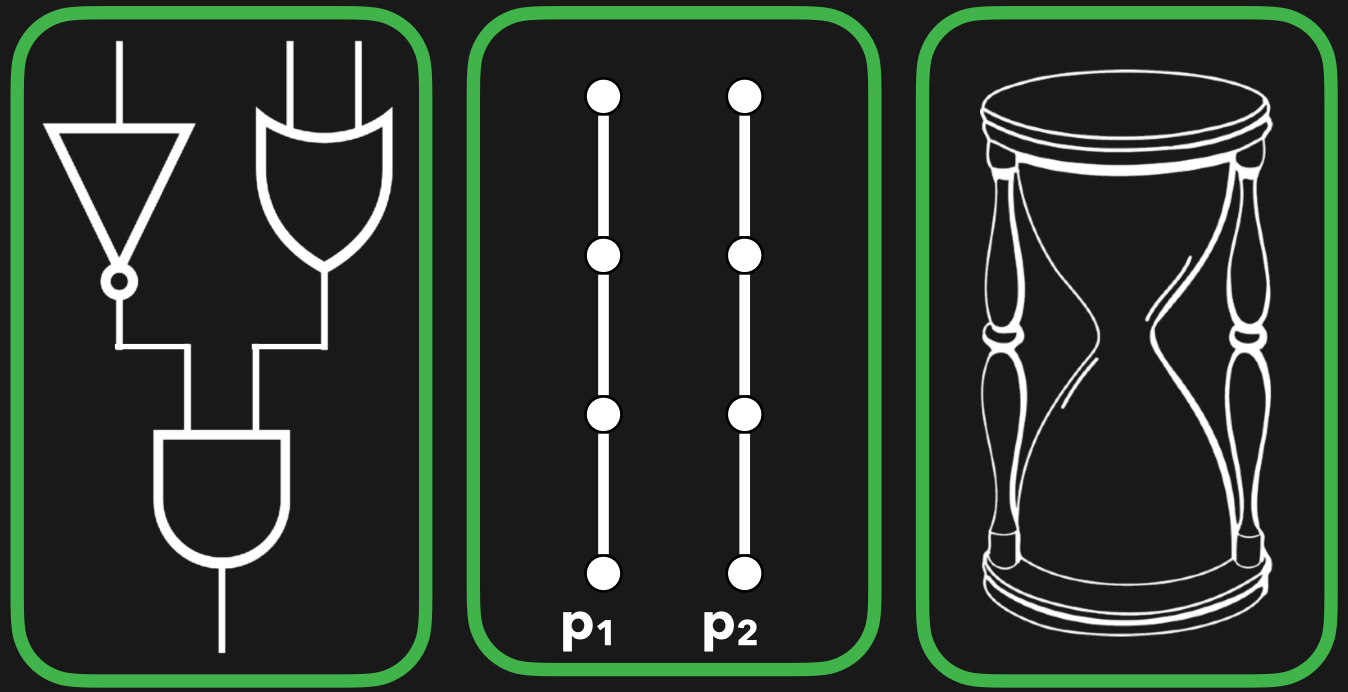
Challenges:



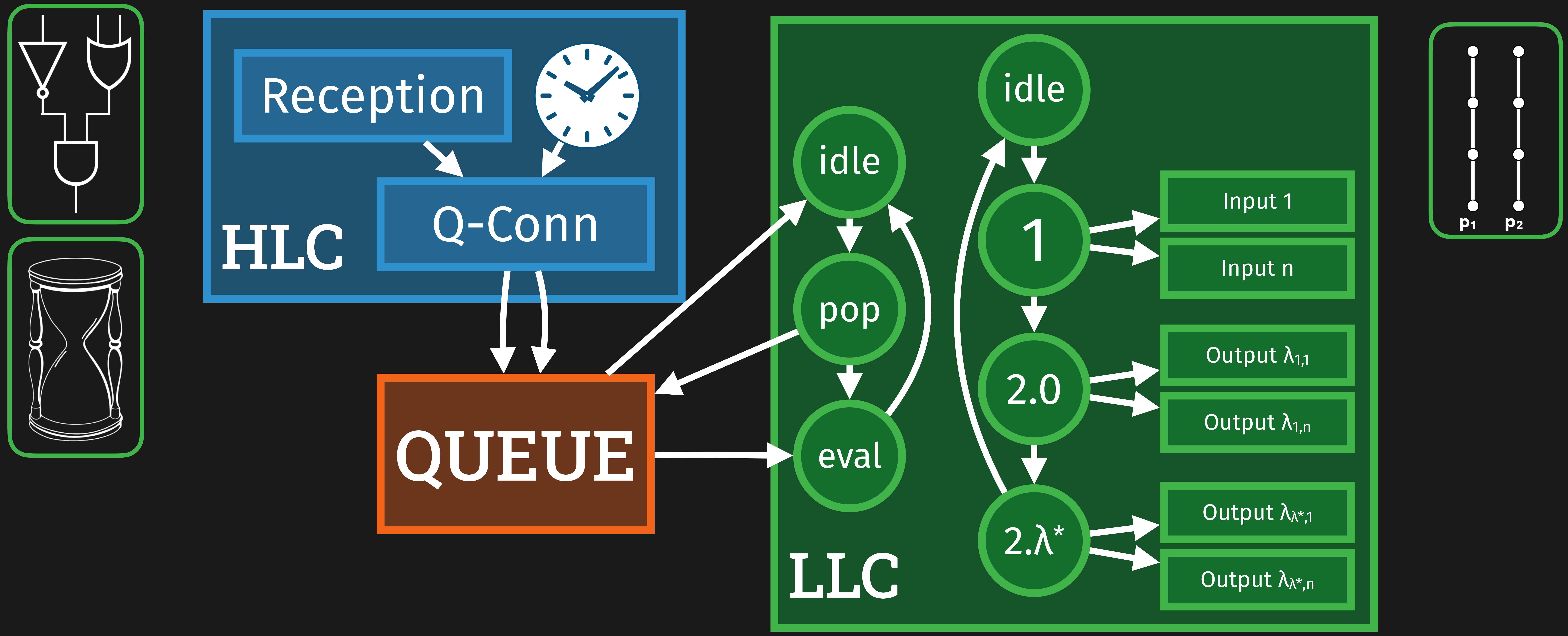
BLOCK II VHDL COMPILATION OVERVIEW



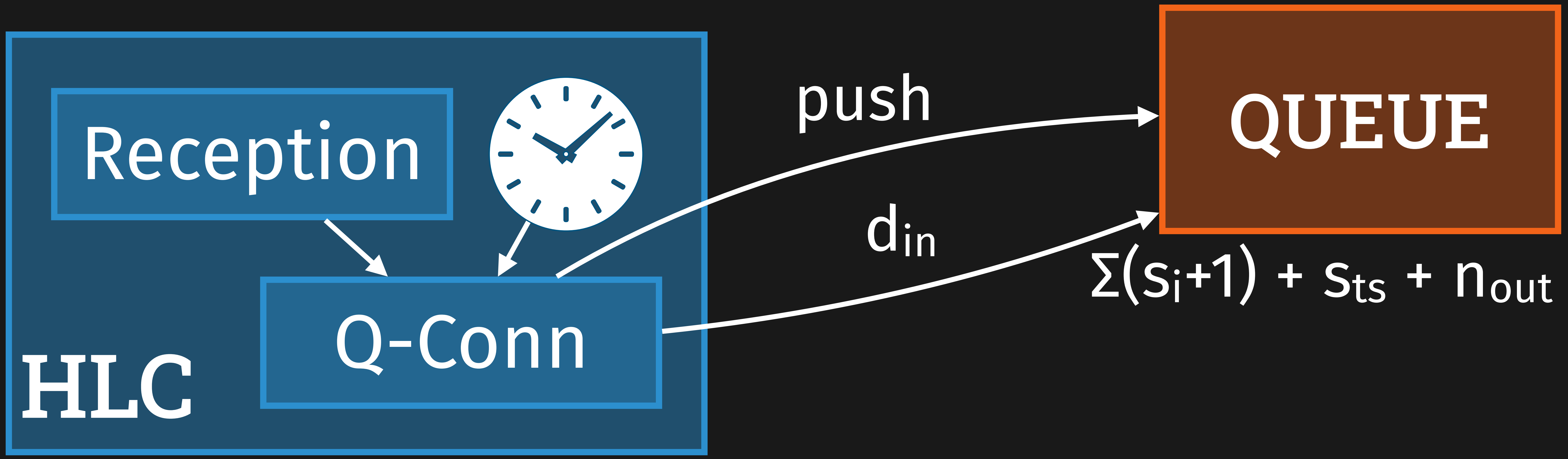
Challenges:



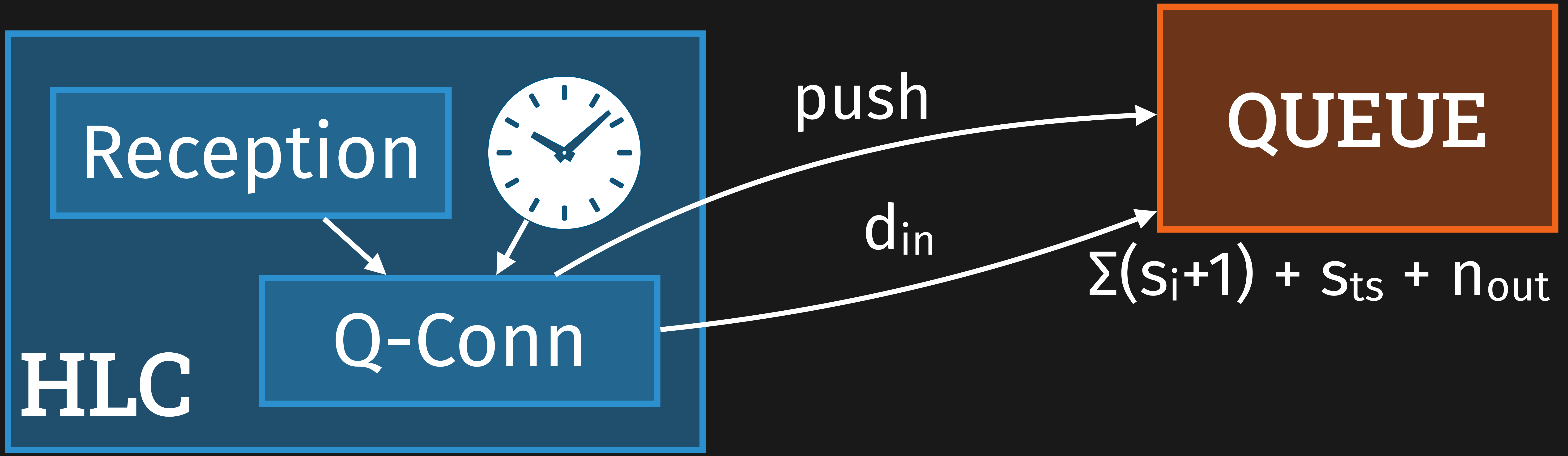
Block II VHDL Compilation Overview



Block II High-Level Controller

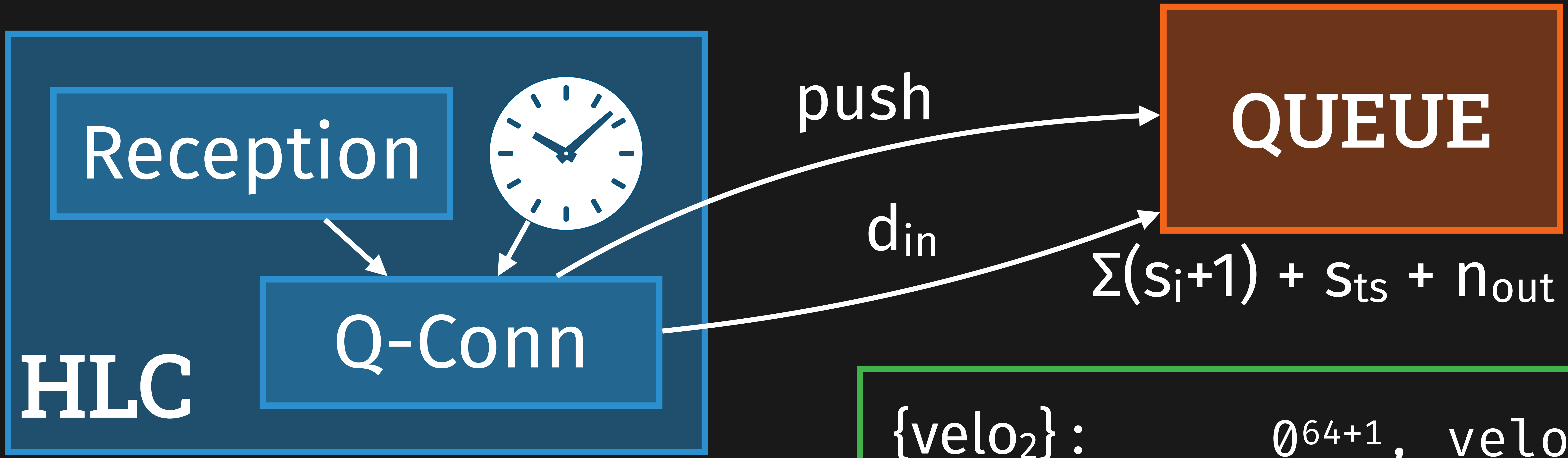


Block II High-Level Controller




```
input velo_1: Int64
input velo_2: Int64
output devi := abs(velo_1 - velo_2)
output lasting_devi := devi > 5
    ∧ devi.offset(by: -1, dft: 0) > 5
    ∧ devi.offset(by: -2, dft: 0) > 5
trigger lasting_devi "Lasting deviation in measured velocities."
output avg_devi @10mHz := devi.aggregate(over: 10min, using: avg)
trigger avg_devi > 4 "High average deviation."
```

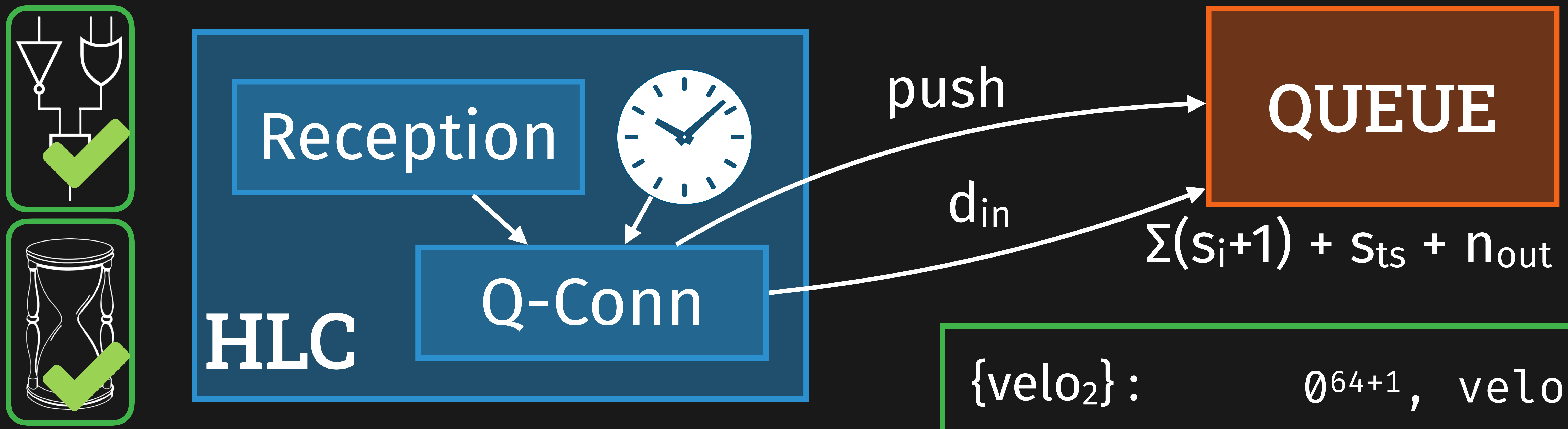
Block II High-Level Controller



$\{velo_2\}$:	0^{64+1}, $velo_2$, 1, ts, 00000
$\{velo_1, velo_2\}$:	$velo_1$, 1, $velo_2$, 1, ts, 11100
$t = 100s$:	0^{64+1}, 0^{64+1}, ts, 00011

```
input velo_1: Int64
input velo_2: Int64
output devi := abs(velo_1 - velo_2)
output lasting_devi := devi > 5
    ∧ devi.offset(by: -1, dft: 0) > 5
    ∧ devi.offset(by: -2, dft: 0) > 5
trigger lasting_devi "Lasting deviation in measured velocities."
output avg_devi @10mHz := devi.aggregate(over: 10min, using: avg)
trigger avg_devi > 4 "High average deviation."
```

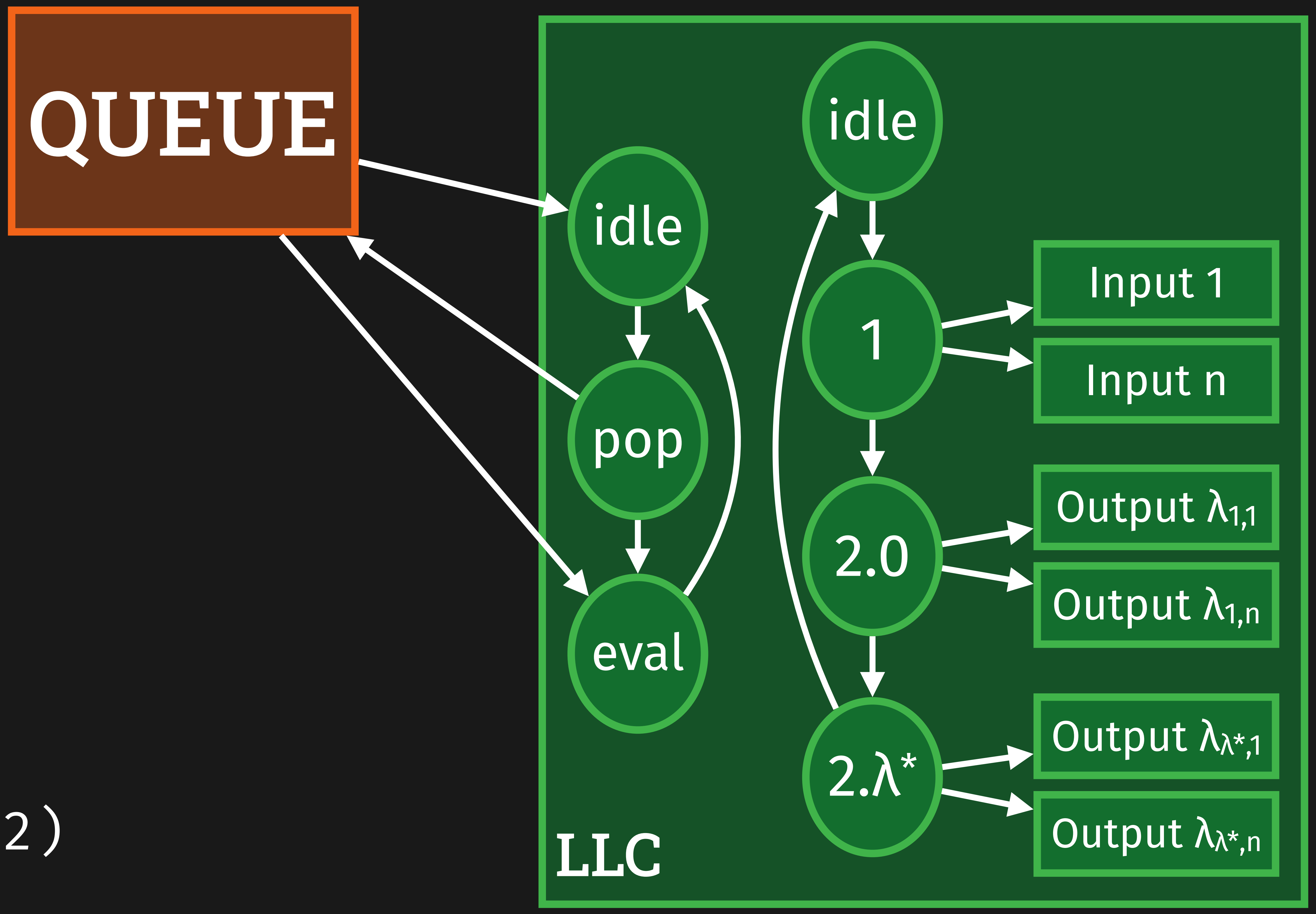
Block II High-Level Controller



{velo₂} :	0^{64+1}, velo₂, 1, ts, 00000
{velo₁, velo₂} :	velo₁, 1, velo₂, 1, ts, 11100
t = 100s :	0^{64+1}, 0^{64+1}, ts, 00011

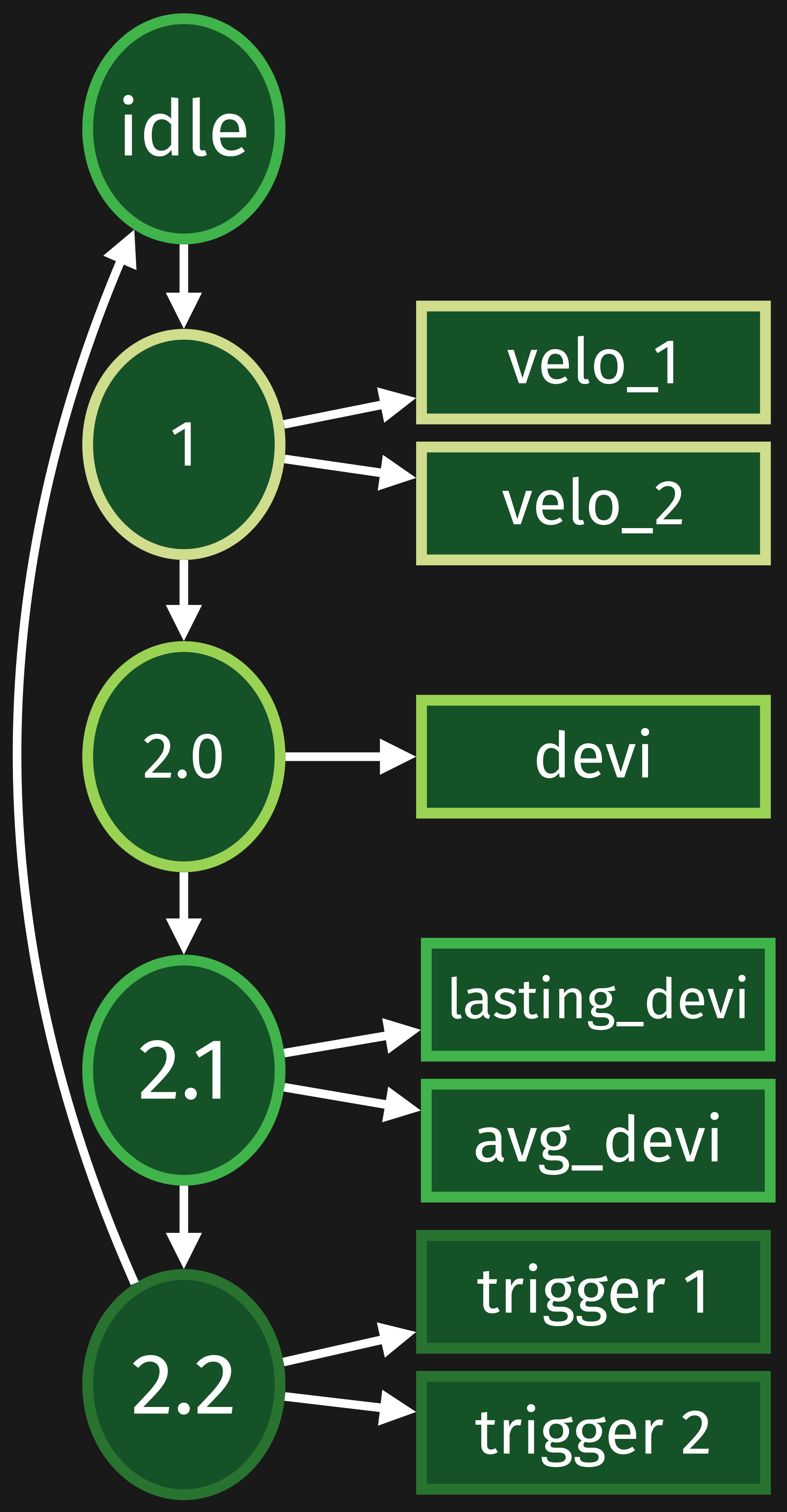
```
input velo_1: Int64
input velo_2: Int64
output devi := abs(velo_1 - velo_2)
output lasting_devi := devi > 5
    ∧ devi.offset(by: -1, dft: 0) > 5
    ∧ devi.offset(by: -2, dft: 0) > 5
trigger lasting_devi "Lasting deviation in measured velocities."
output avg_devi @10mHz := devi.aggregate(over: 10min, using: avg)
trigger avg_devi > 4 "High average deviation."
```

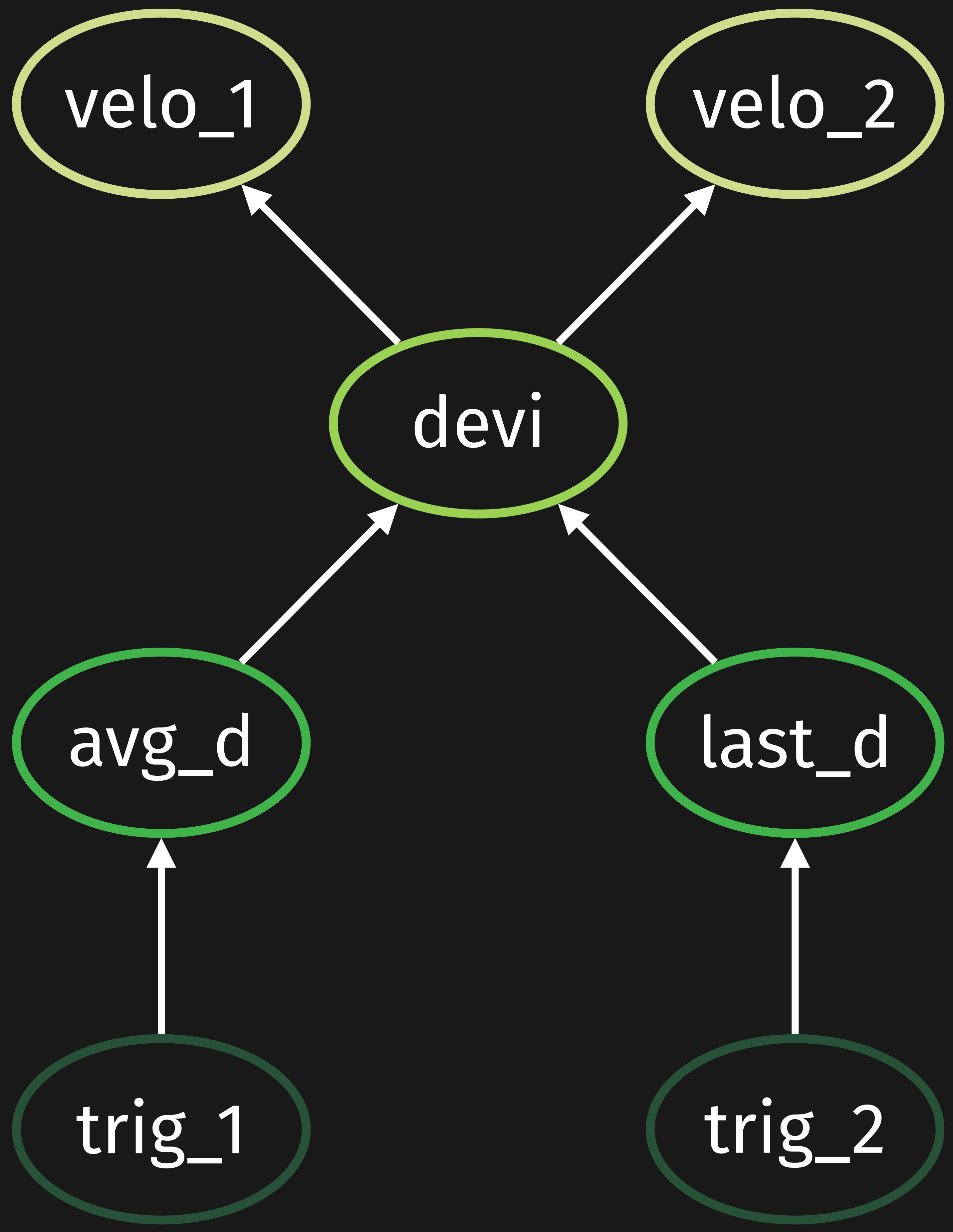
Block II Low-Level Controller



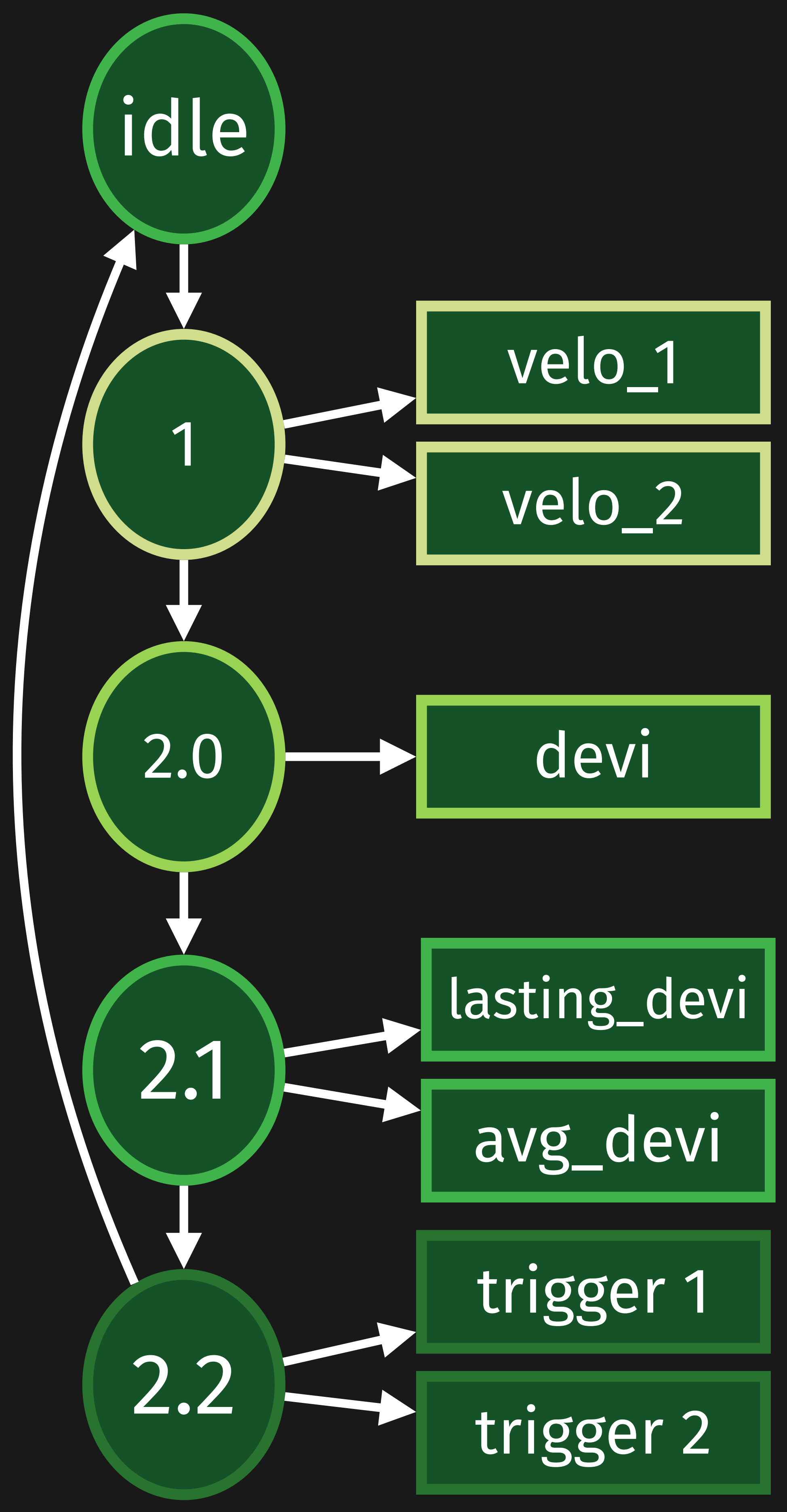

```
input velo_1: Int64
input velo_2: Int64
output devi := abs(velo_1 - velo_2)
output lasting_devi := devi > 5
    ∧ devi.offset(by: -1, dft: 0) > 5
    ∧ devi.offset(by: -2, dft: 0) > 5
trigger lasting_devi "Lasting deviation in measured velocities."
output avg_devi @10mHz := devi.aggregate(over: 10min, using: avg)
trigger avg_devi > 4 "High average deviation."
```

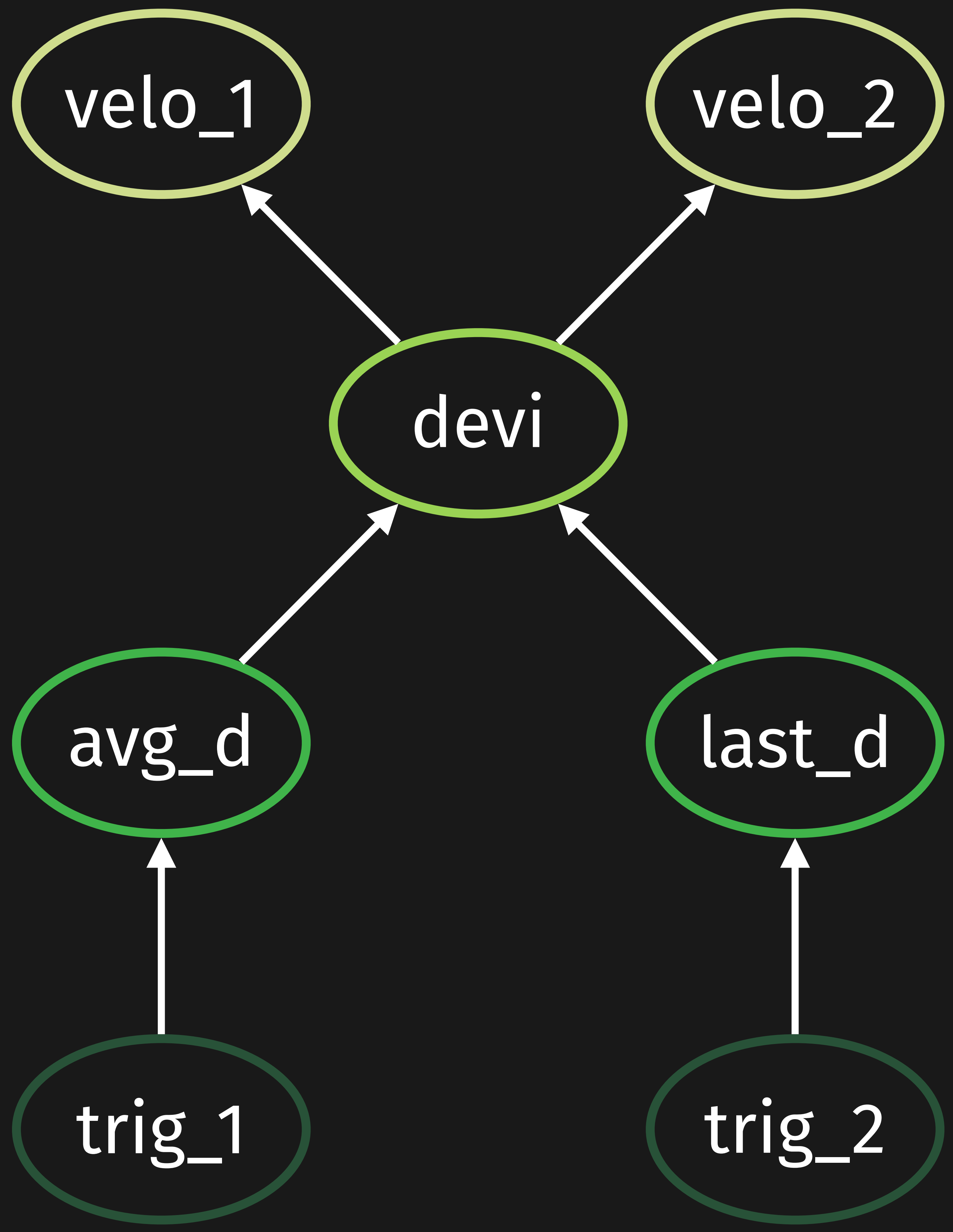
Parallel Computation

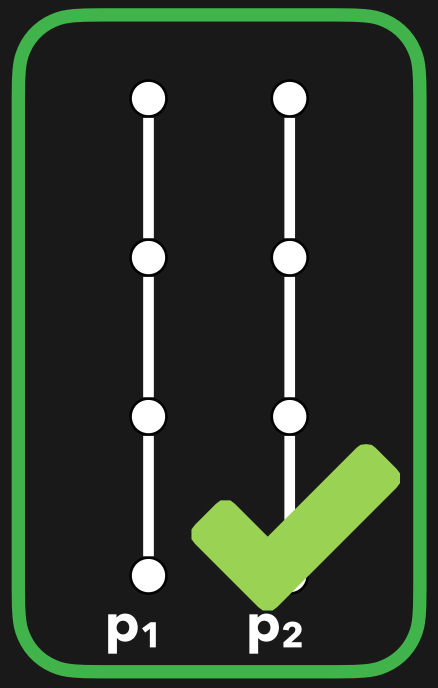




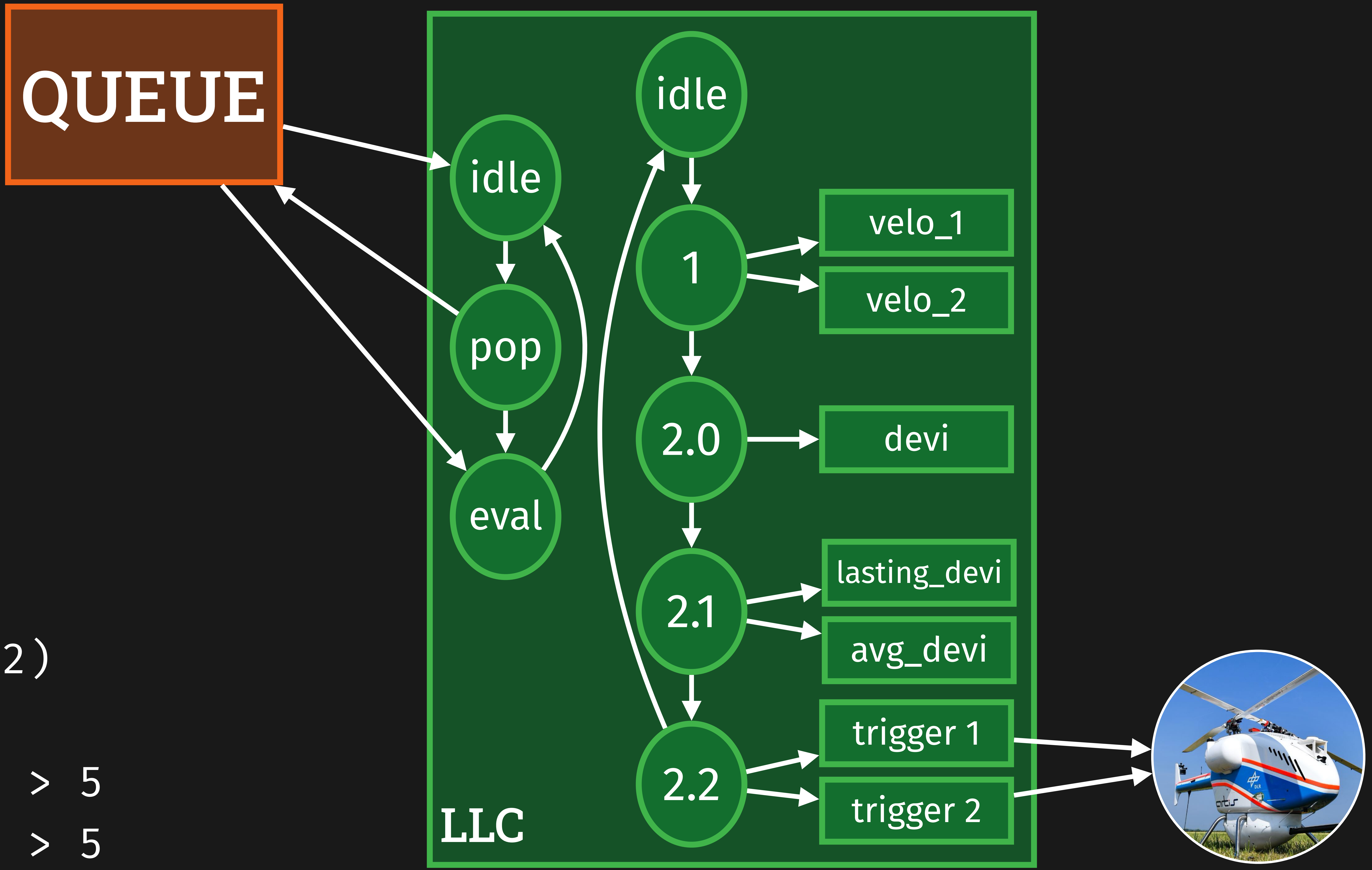
Block II Parallel Computation







Block II Low-Level Controller

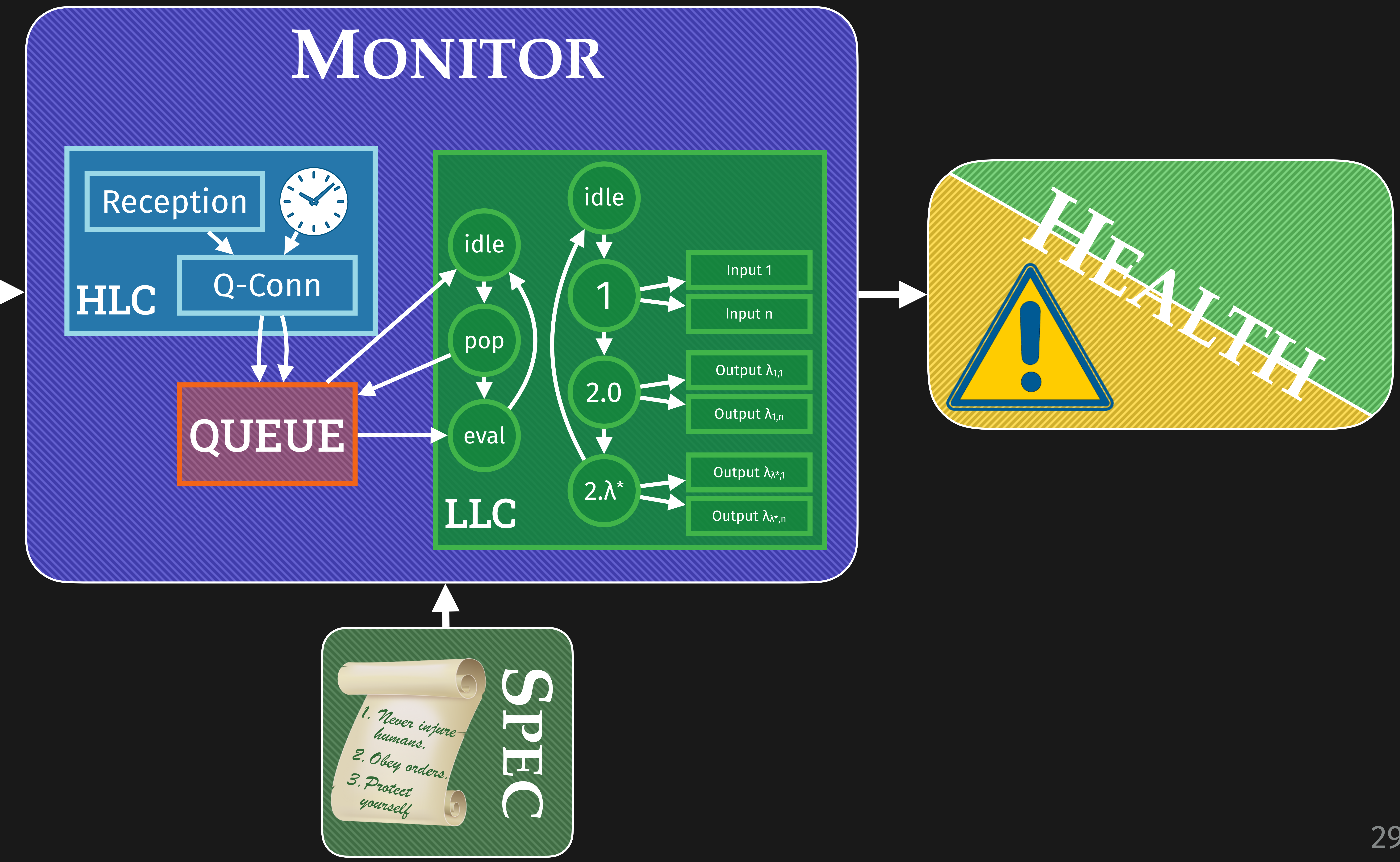



```
input velo_1: Int64
input velo_2: Int64
output devi := abs(velo_1 - velo_2)
output lasting_devi := devi > 5
    ∧ devi.offset(by: -1, dft: 0) > 5
    ∧ devi.offset(by: -2, dft: 0) > 5
trigger lasting_devi "Lasting deviation in measured velocities."
output avg_devi @10mHz := devi.aggregate(over: 10min, using: avg)
trigger avg_devi > 4 "High average deviation."
```

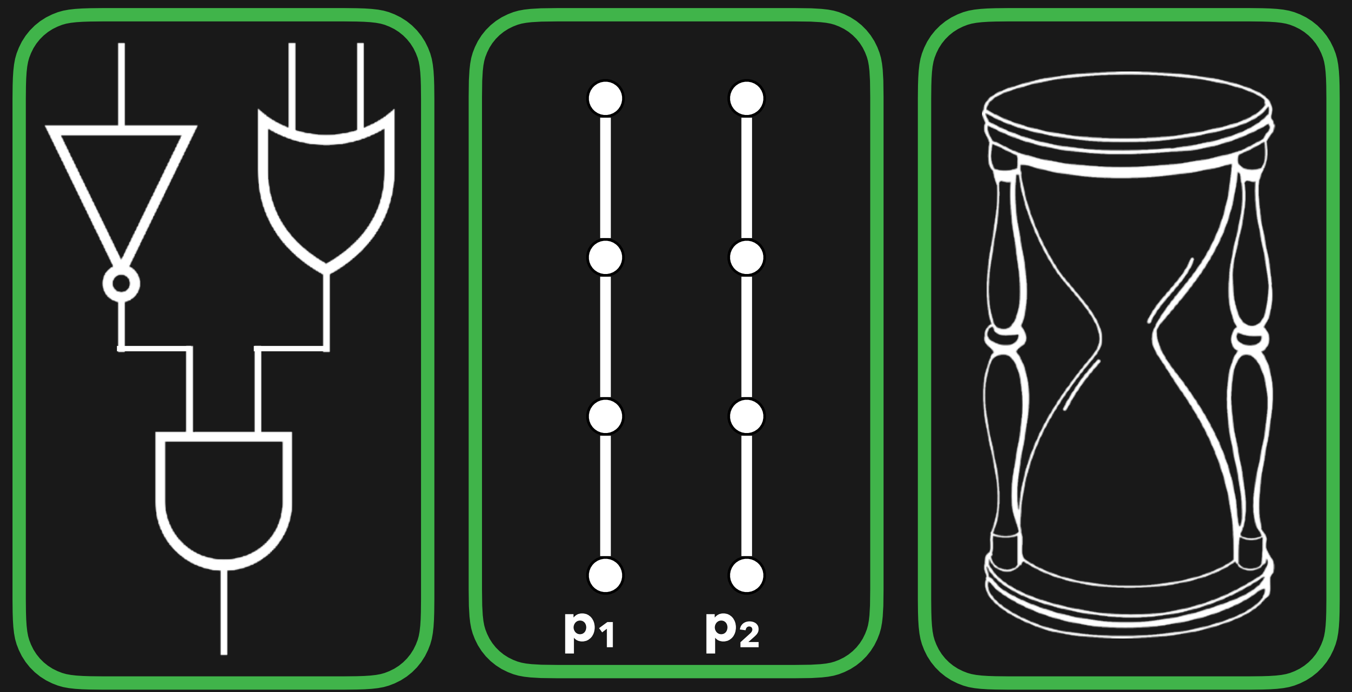
Block II Empirical Evaluation

		FF	LUT	MUX	CA	MULT	Pwr [W]	Time [μs]
Drone	Mon	3036	3685	26	656	10	1.620	4.28
	HLC	901	156	0	22	0		
	Q	543	442	0	43	0		
	LLC	1281	2820	0	576	10		
Network	Mon	1905	1533	23	226	23	1.570	3.20
	HLC	550	161	0	37	0		
	Q	330	342	0	28	0		
	LLC	895	927	0	161	0		
Cmd Resp Par	Mon	6379	13794	0	849	0	1.582	3.77
	HLC	936	232	0	30	0		
	Q	540	326	0	28	0		
	LLC	4903	13236	0	971	0		
Cmd Resp Seq	Mon	6909	14768	0	851	0	1.581	43.83
	HLC	936	232	0	30	0		
	Q	534	326	0	28	0		
	LLC	5433	14210	0	973	0		

Block II Summary HW Compilation



Challenges:



Block II Software Compilation

observes

Lola Specification

Compilation

```
Impl Monitor {

 while let Some(i)
       = get_input() {
  …
 }}
```

High Level Code

Monitor

```
01010010
01010110
00110010
00110000
```

Block II Software Compilation

observes

Monitor

Lola Specification

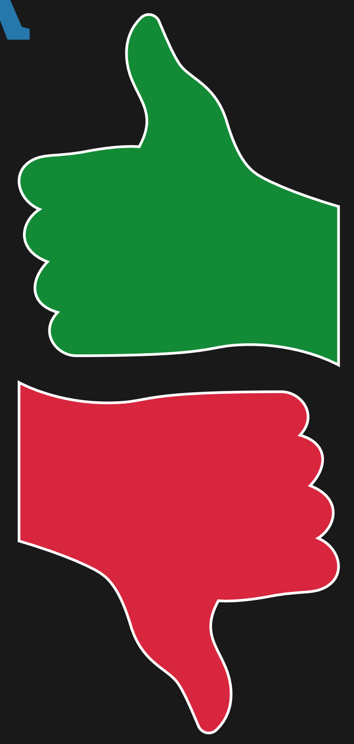
Compilation →

```
Impl Monitor {

 while let Some(i)
       = get_input() {
  …
 }}
```

Rust Code

→

```
01010010
01010110
00110010
00110000
```



Block II Software Compilation

observes

Monitor

Lola Specification

Compilation

+ Annotation Generation

```
Impl Monitor {
 #[invariant = … ]
 while let Some(i)
       = get_input() {
  …
 }}
```

Rust Code

```
01010010
01010110
00110010
00110000
```

Block II Software Compilation

VIPER

verifies

observes

Lola Specification

Compilation

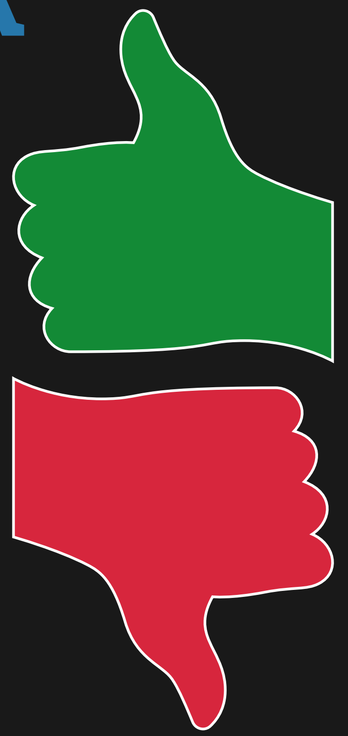
+ Annotation Generation

```
Impl Monitor {
 #[invariant = … ]
 while let Some(i)
       = get_input() {
  …
 }}
```

Rust Code

Monitor

```
01010010
01010110
00110010
00110000
```



Block II Software Compilation

verifies

observes

Lola Specification

Compilation

+ Annotation Generation

```
Impl Monitor {
 #[invariant = … ]
 while let Some(i)
         = get_input() {
  …
 }}
```

Rust Code

Monitor

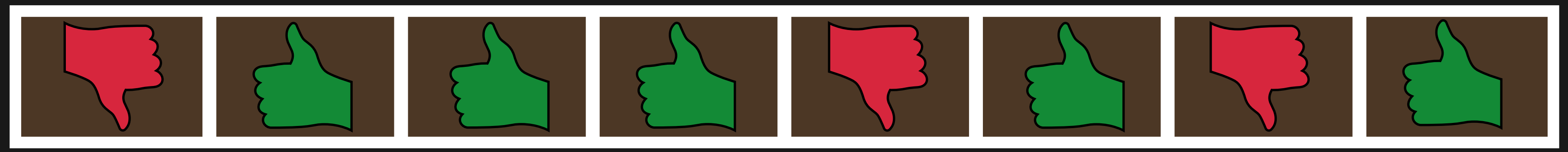
```
01010010
01010110
00110010
00110000
```

Block II The Original Lola

sensors

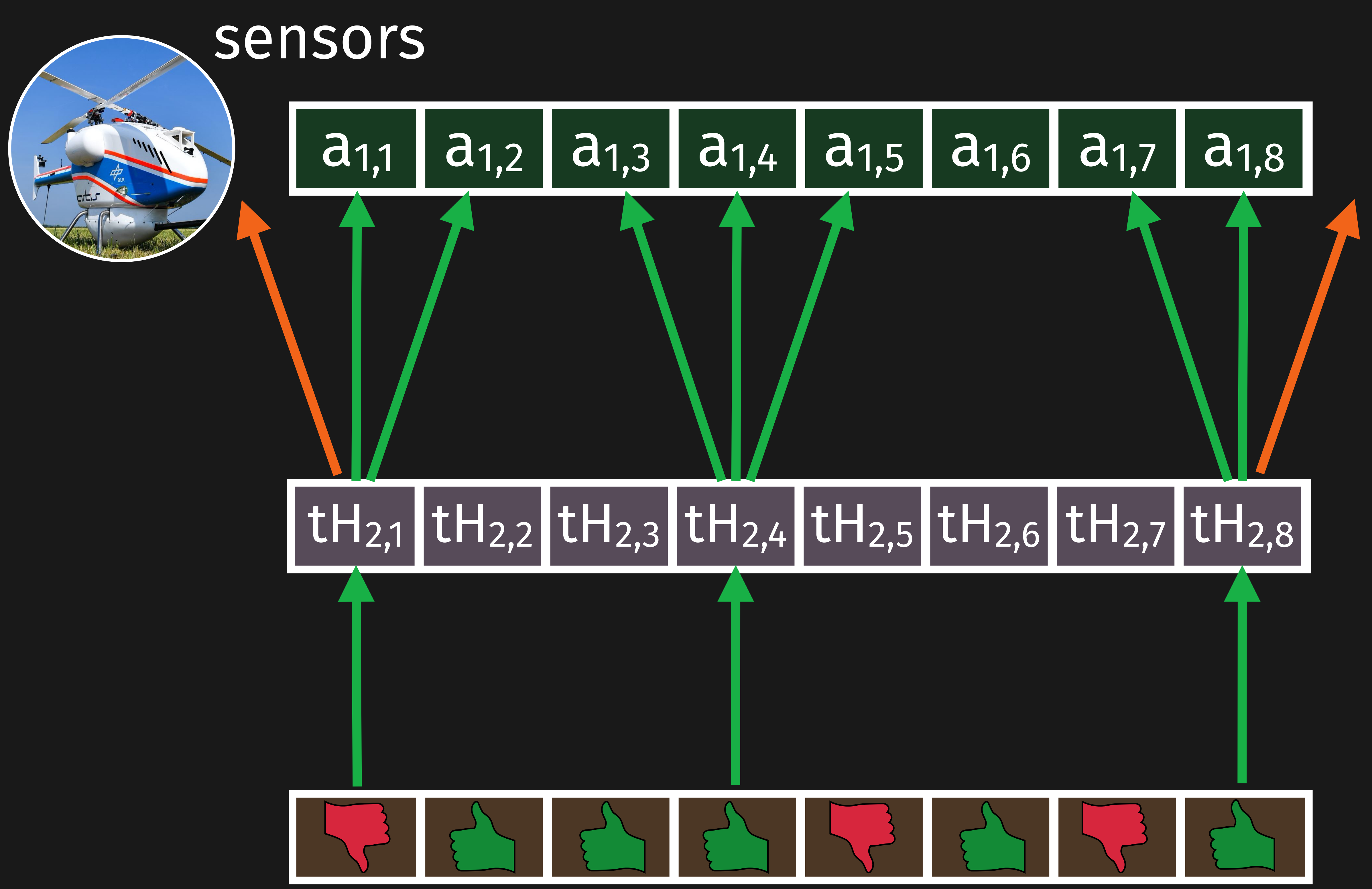
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$	$a_{1,8}$

$tH_{2,1}$	$tH_{2,2}$	$tH_{2,3}$	$tH_{2,4}$	$tH_{2,5}$	$tH_{2,6}$	$tH_{2,7}$	$tH_{2,8}$



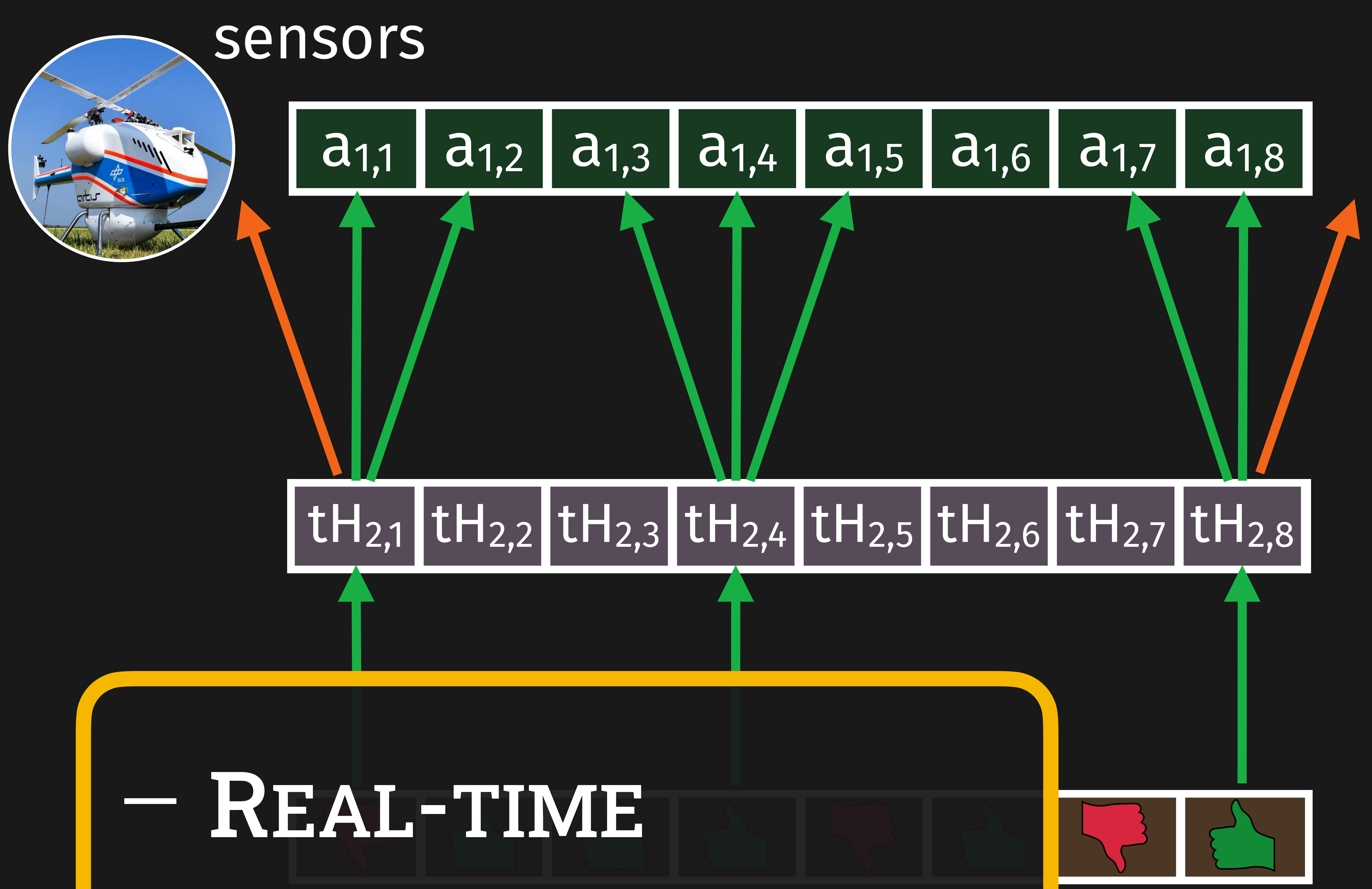
```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

Block II The Original Lola




```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

Block II The Original Lola

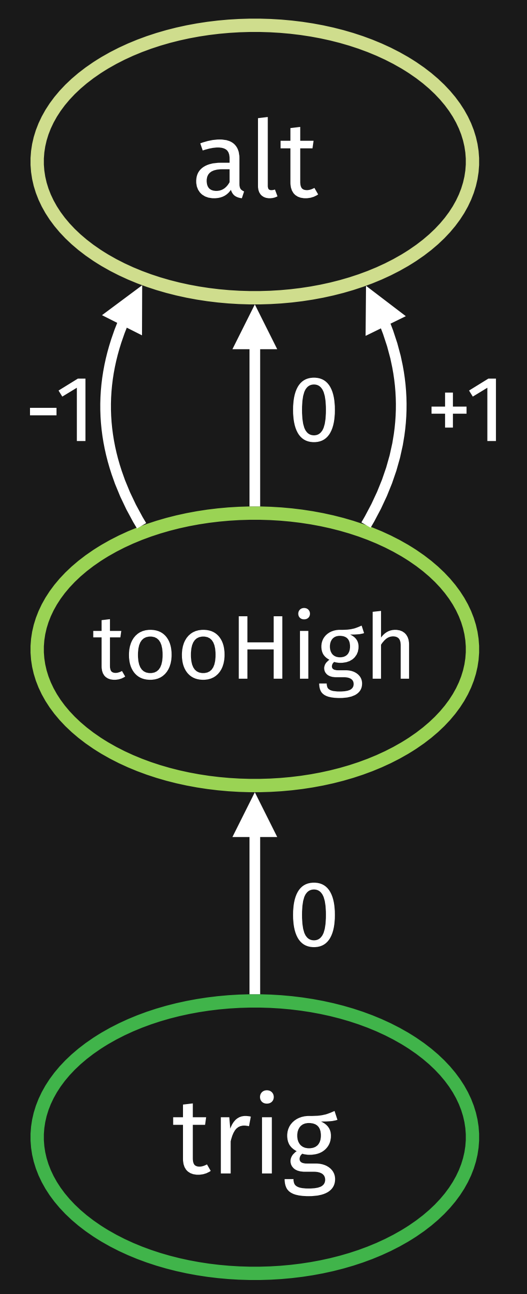


- Real-time
- No asynchrony
+ Future Offsets

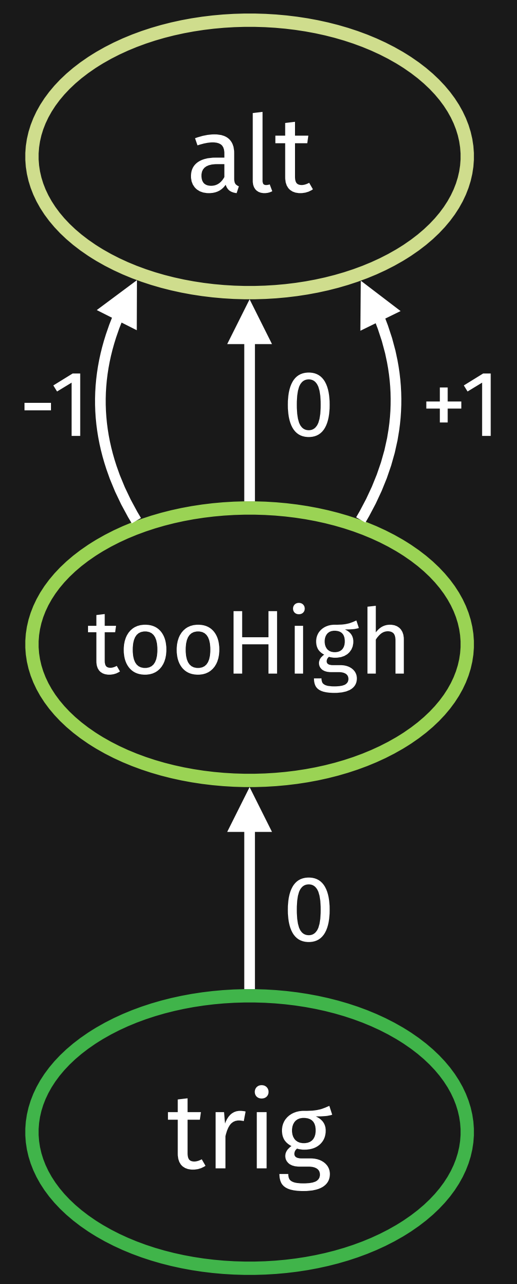
```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

Block II Shift & Memory Requirement

```
input alt
output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500
trigger tooHigh
```

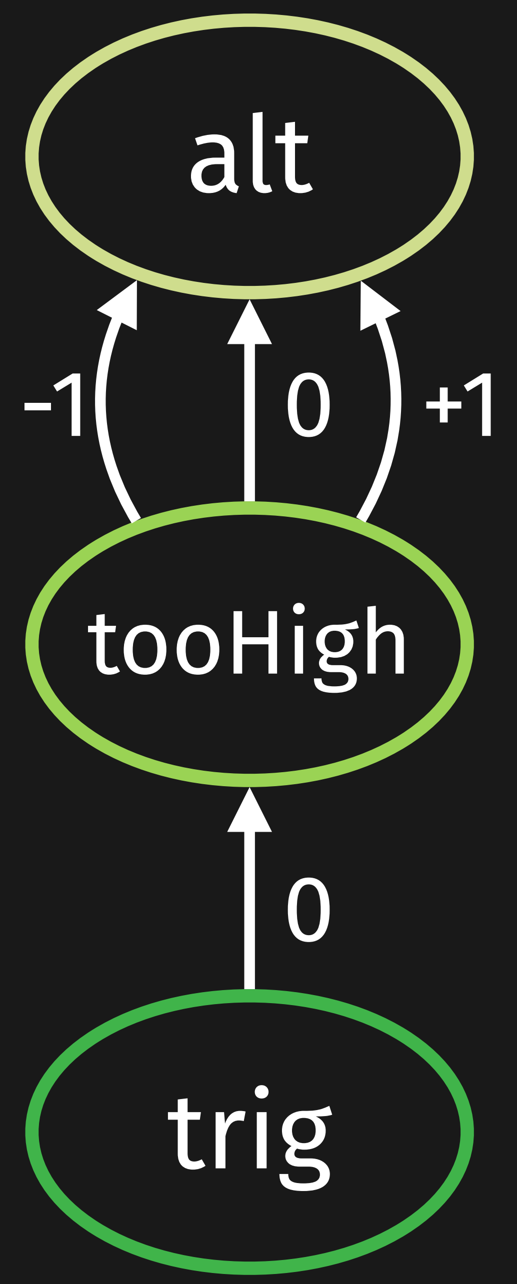


Block II Shift & Memory Requirement




```
input alt
output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500
trigger tooHigh
```

Block II Shift & Memory Requirement



Def Shift:

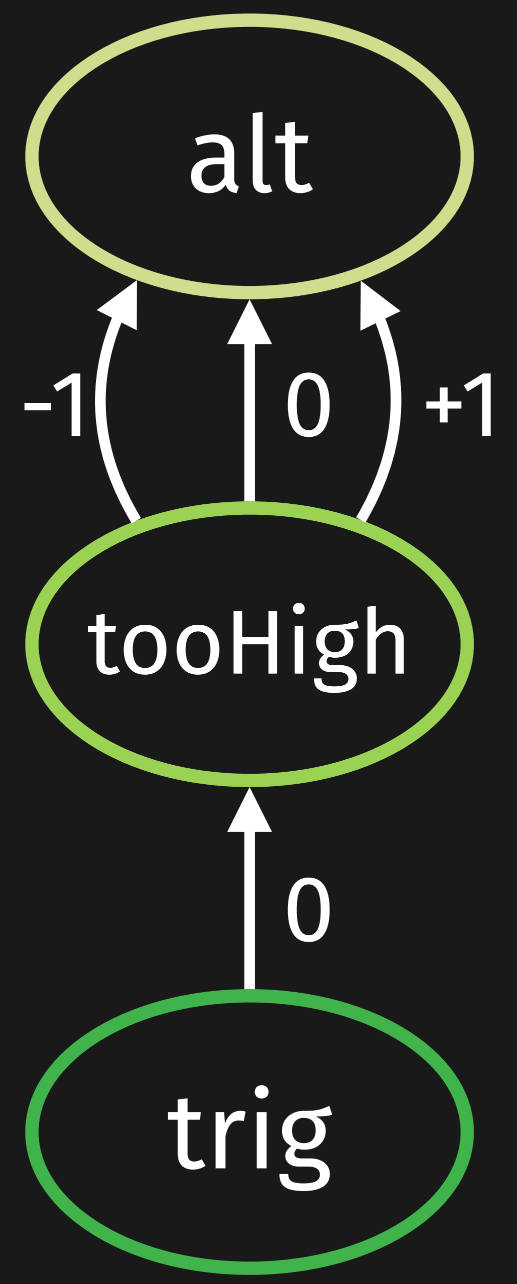
$\Delta(s) = \max(0, \max\{w + \Delta(s') \mid (s,w,s') \in R\})$

$\Delta(alt) = 0$

$\Delta(tooHigh) = \Delta(trig) = 1$

```
input alt
output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500
trigger tooHigh
```

Block II Shift & Memory Requirement




```
input alt
output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500
trigger tooHigh
```

Def Shift:

$\Delta(s) = \max(0, \max\{w + \Delta(s') \mid (s,w,s') \in R\})$

$\Delta(alt) = 0$

$\Delta(tooHigh) = \Delta(trig) = 1$

Def Memory Requirement:

$\mu(s) = \max\{\Delta(s') - \Delta(s) - w \mid (s',w,s) \in E\}$

$\mu(alt) = 2$

$\mu(tooHigh) = \Delta(trig) = 0$

Block II (In-)Fallible Accesses

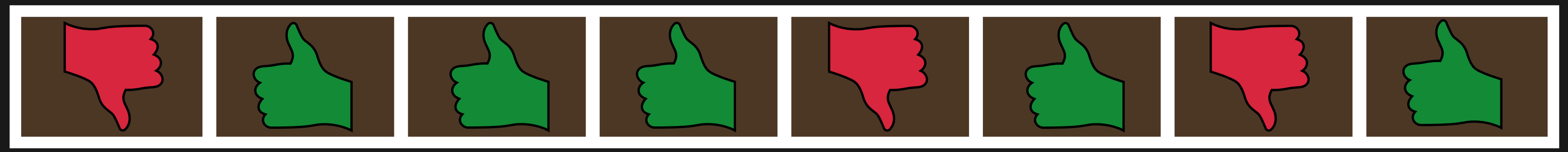

```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

Block II (In-)Fallible Accesses

sensors

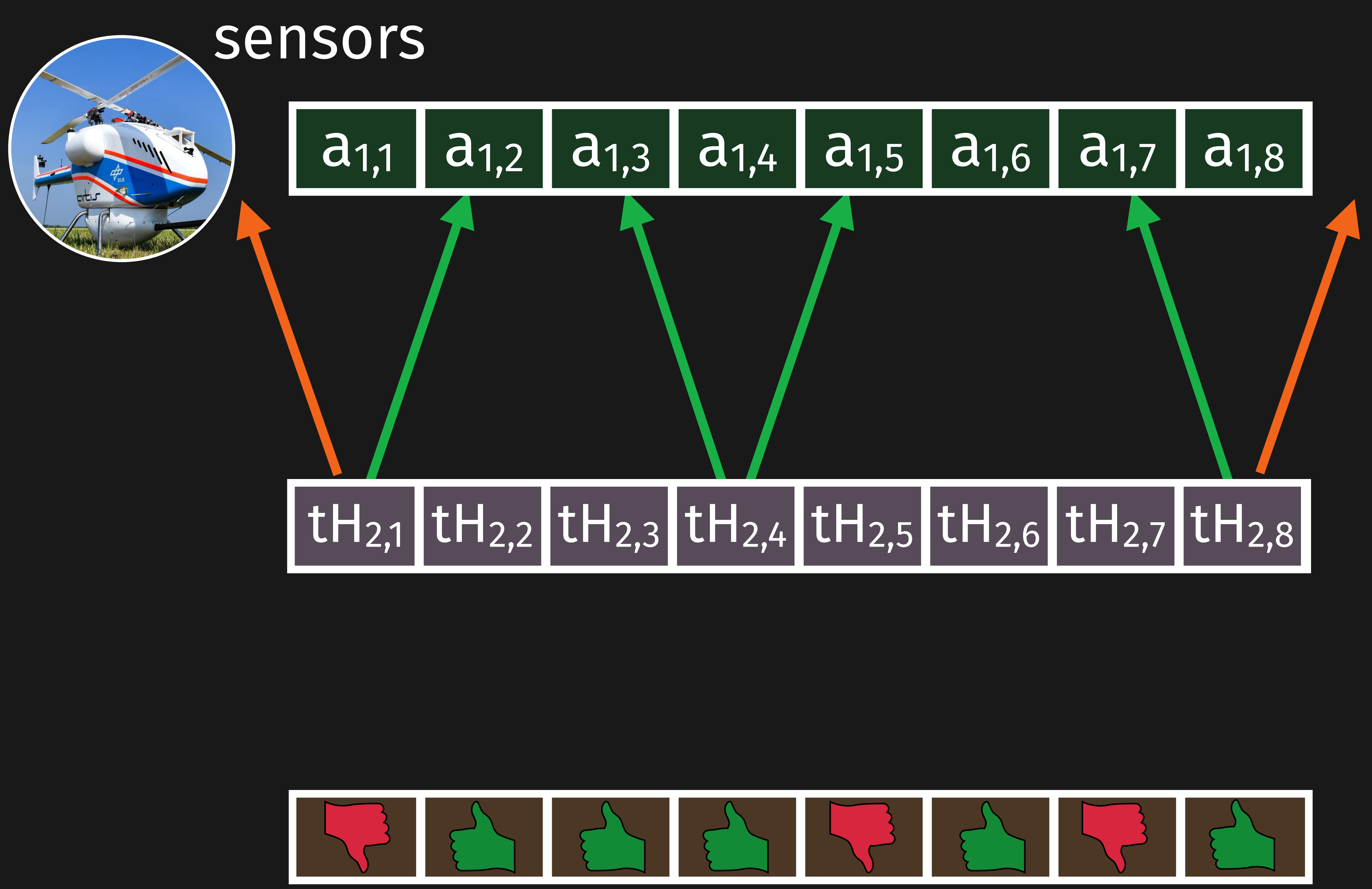
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,4}$ $a_{1,5}$ $a_{1,6}$ $a_{1,7}$ $a_{1,8}$

$tH_{2,1}$ $tH_{2,2}$ $tH_{2,3}$ $tH_{2,4}$ $tH_{2,5}$ $tH_{2,6}$ $tH_{2,7}$ $tH_{2,8}$



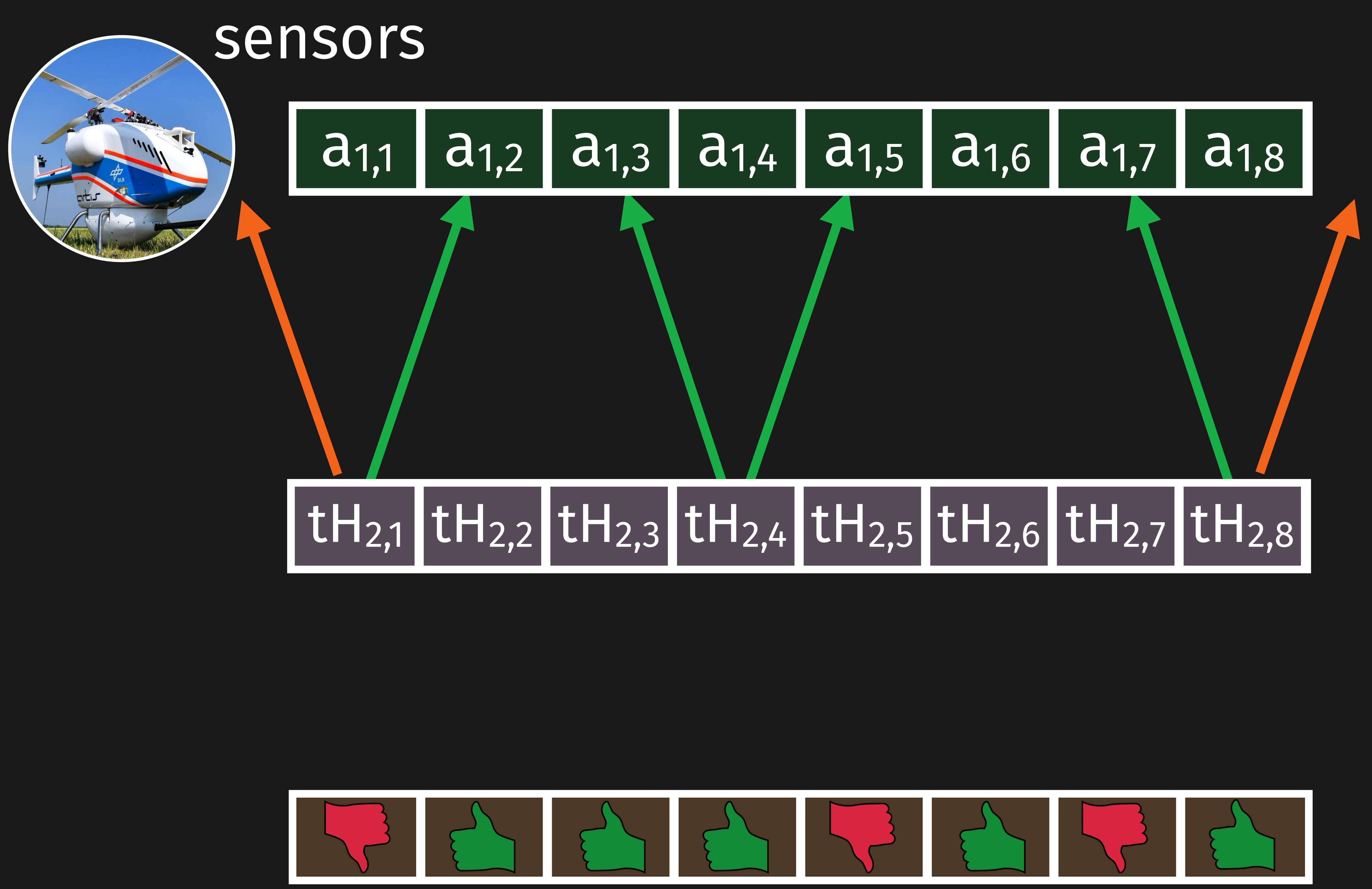
```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

Block II (In-)Fallible Accesses




```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

Block II (In-)Fallible Accesses

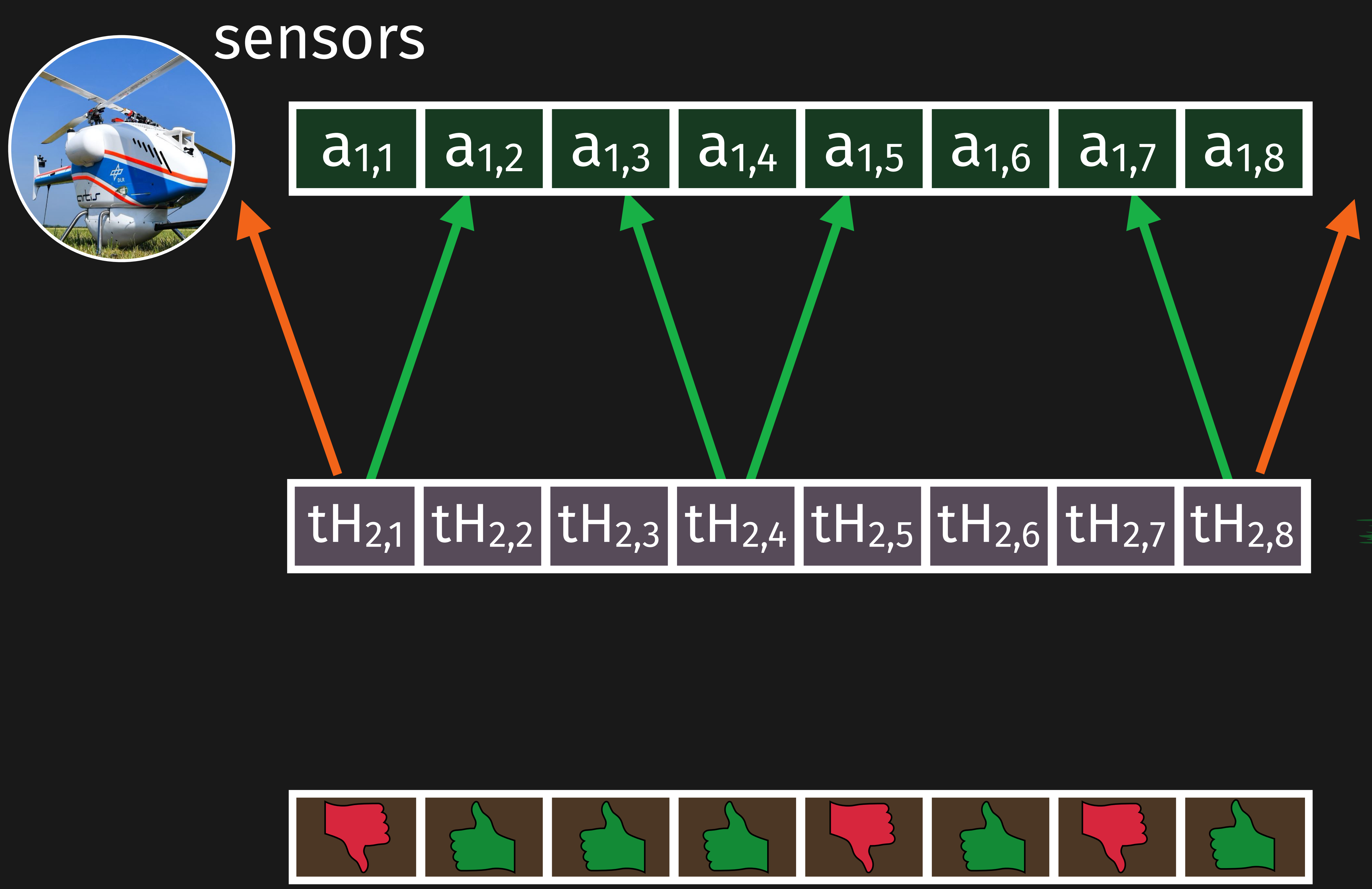



```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

fails @ t=1

fails @ t=|σ|

Block II (In-)Fallible Accesses

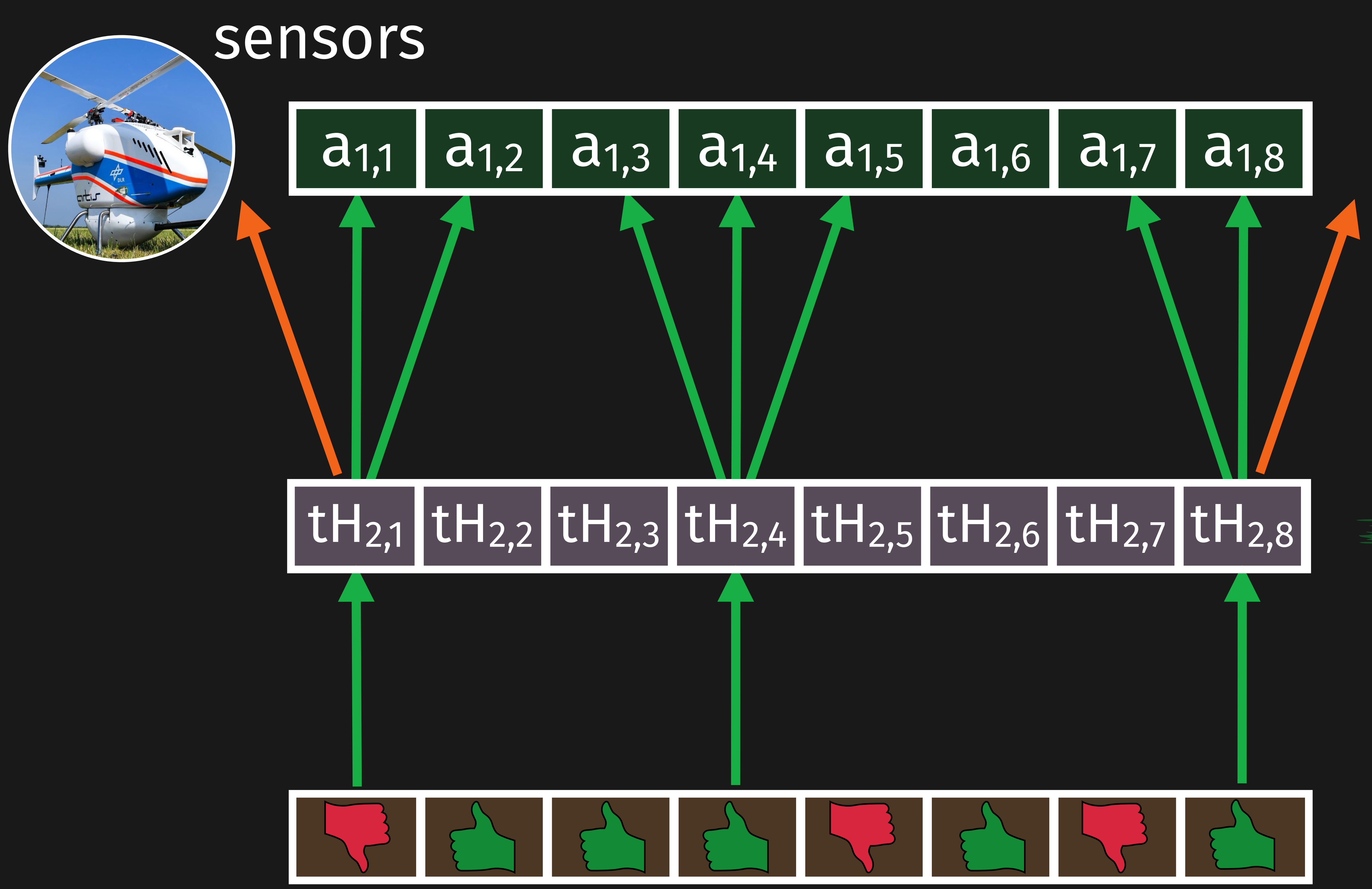



```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

fails @ t=1

fails @ t=|σ|

Block II (In-)Fallible Accesses

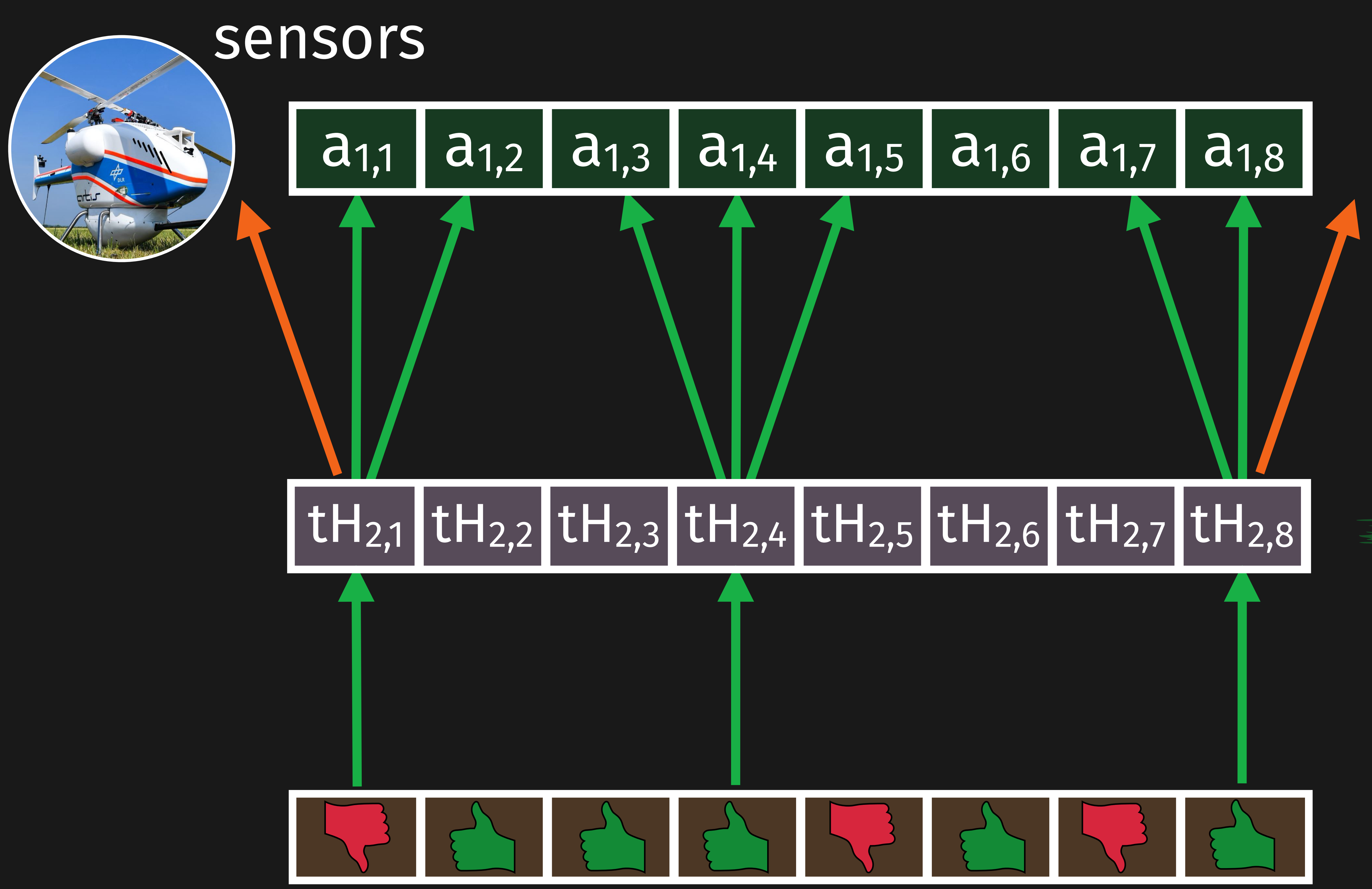



```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

fails @ t=1

fails @ t=|σ|

Block II (In-)Fallible Accesses



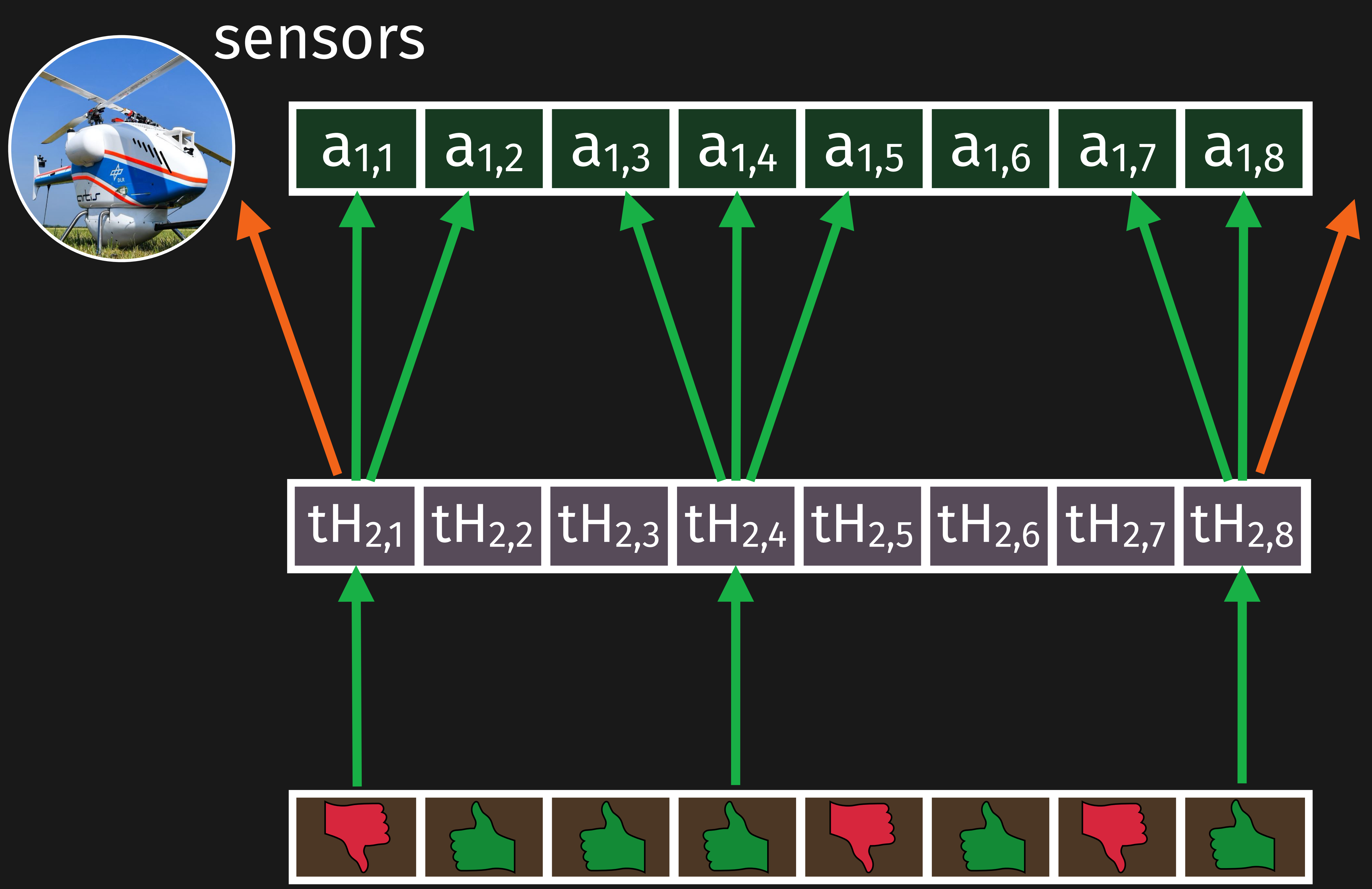

```
input alt

output tooHigh :=
  alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
```

fails @ t=1

never fails

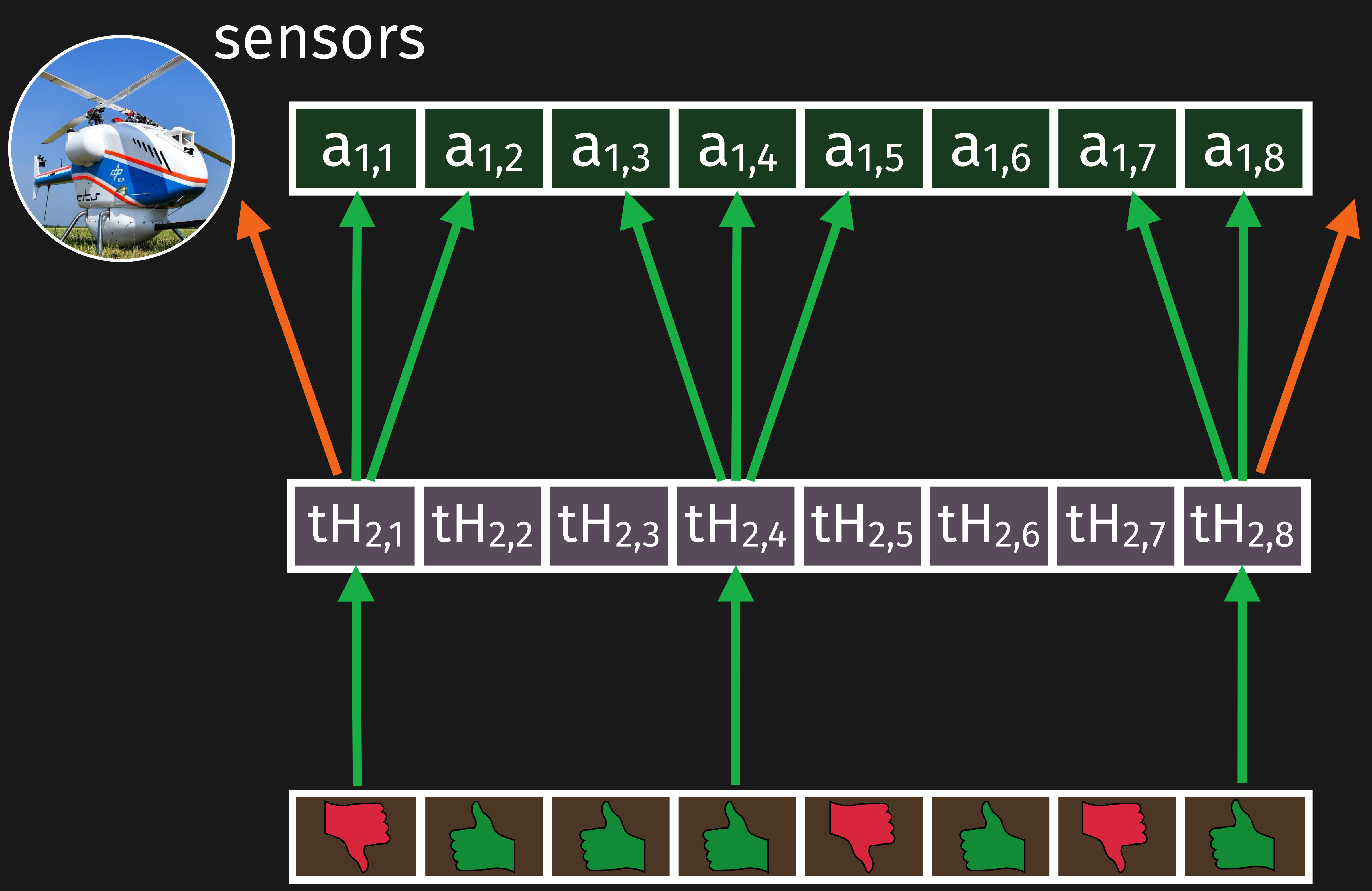
fails @ t=|σ|

Block II (In-)Fallible Accesses



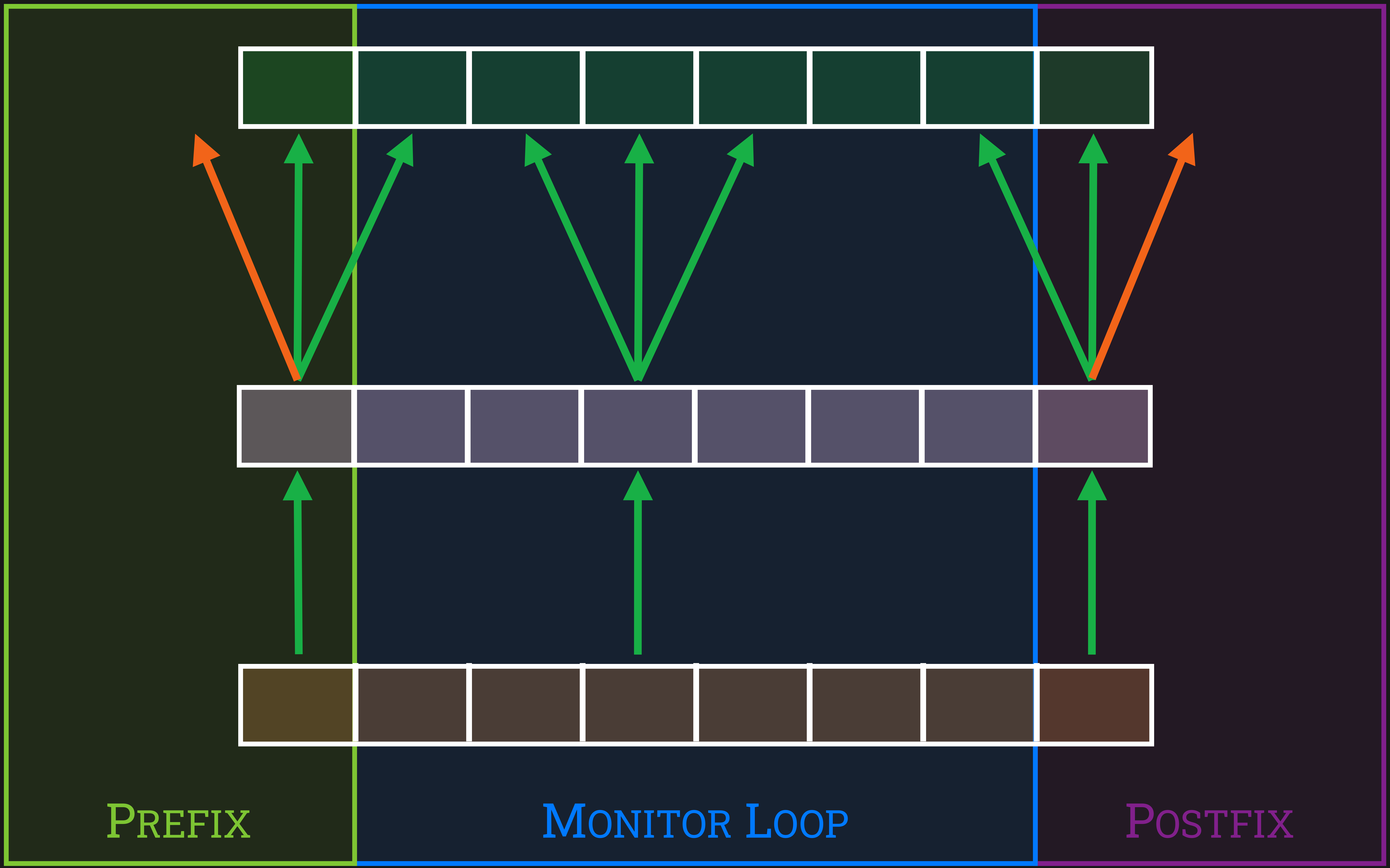

```
let alt_past =
        mem.get_alt(-1).unwrap_or(0);

let alt_future =
        mem.get_alt(+1).unwrap_or(0);

let alt_current = mem.get_alt_sync(0);

let tooHigh =
            alt_past > 500
         && alt_current > 500
         && alt_future > 500
```

Block II (In-)Fallible Accesses




```
let alt_past =
      mem.get_alt(-1).unwrap_or(0);

let alt_future =
      mem.get_alt(+1).unwrap_or(0);

let alt_current = mem.get_alt_sync(0);

let tooHigh =
          alt_past > 500
       && alt_current > 500
       && alt_future > 500
```

Block II Three Phases



Block II Three Phases

```
fn prefix() {
 a₋₁ > 500
   ∧ a₀ > 500
   ∧ a₊₁ > 500
}
```

Prefix

```
while let Some(…)
               = get_input() {
   a₋₁ > 500
      ∧ a₀ > 500
      ∧ a₊₁ > 500
}
```

Monitor Loop

```
fn postfix() {
 a₋₁ > 500
   ∧ a₀ > 500
   ∧ a₊₁ > 500
}
```

Postfix

Block II Three Phases

```
fn prefix() {
   0 > 500
   ∧ a₀ > 500
   ∧ a₊₁ > 500
}
```

Prefix

```
while let Some(…)
            = get_input() {
   a₋₁ > 500
      ∧ a₀ > 500
      ∧ a₊₁ > 500
}
```

Monitor Loop

```
fn postfix() {
 a₋₁ > 500
  ∧ a₀ > 500
  ∧ a₊₁ > 500
}
```

Postfix

Block II Three Phases

```
fn prefix() {
  0 > 500
   ∧ a₀ > 500
   ∧ a₊₁ > 500
}
```

Prefix

```
while let Some(…)
            = get_input() {
  a₋₁ > 500
    ∧ a₀ > 500
    ∧ a₊₁ > 500
}
```

Monitor Loop

```
fn postfix() {
  a₋₁ > 500
   ∧ a₀ > 500
   ∧ a₊₁ > 500
}
```

Postfix

Block II Three Phases

```
fn prefix() {
  0 > 500
  ∧ a₀ > 500
  ∧ a₊₁ > 500
}
```

```
while let Some(…)
          = get_input() {
  a₋₁ > 500
    ∧ a₀ > 500
    ∧ a₊₁ > 500
}
```

```
fn postfix() {
  a₋₁ > 500
  ∧ a₀ > 500
  ∧   0 > 500
}
```

Prefix | Monitor Loop | Postfix

Block II Three Phases

I. Eradicate Most Conditionals
II. Replace Memory Accesses with Constants

```
fn prefix() {
  0 > 500
  ∧ a₀ > 500
  ∧ a₊₁ > 500
}
```

```
while let Some(…)
          = get_input() {
  a₋₁ > 500
    ∧ a₀ > 500
    ∧ a₊₁ > 500
}
```

```
fn postfix() {
  a₋₁ > 500
  ∧ a₀ > 500
  ∧   0 > 500
}
```

Prefix | Monitor Loop | Postfix

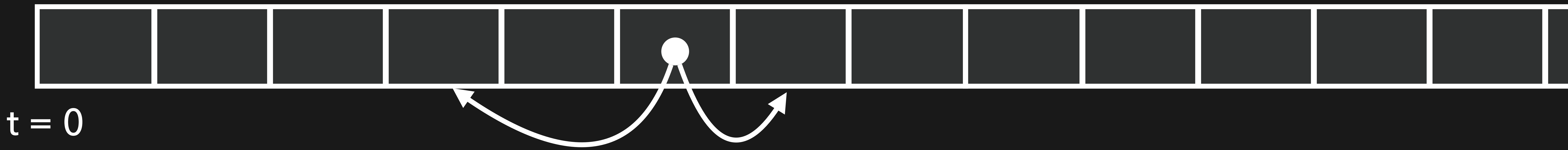
Block II Performance Benefit

Interpreter	438ns	1.535μs
Compilation	6ns (1.4%)	63ns (4%)

Block II Verification — Idea

stream ≜ infinite sequence of values

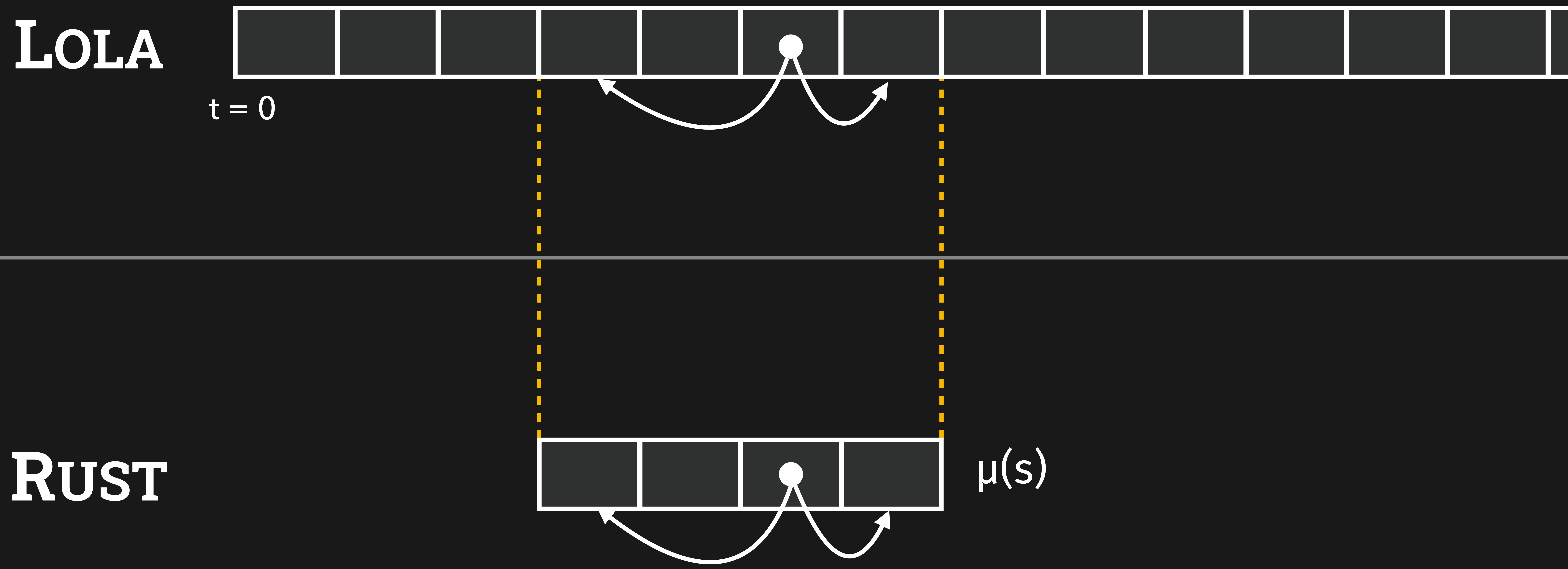
Lola



Rust

Block II Verification — Idea

stream ≜ infinite sequence of values

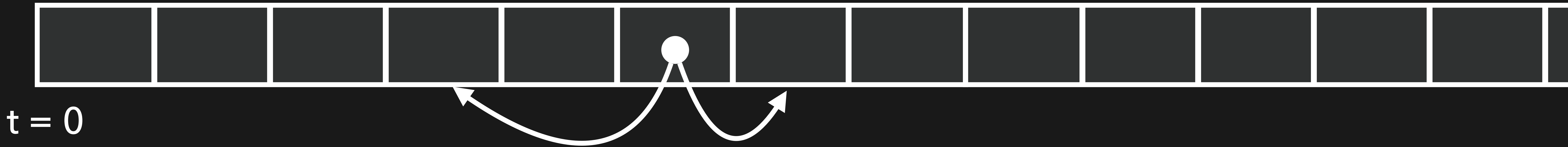


memory ≜ finite excerpt of stream

Block II Verification — Idea

stream ≜ infinite sequence of values

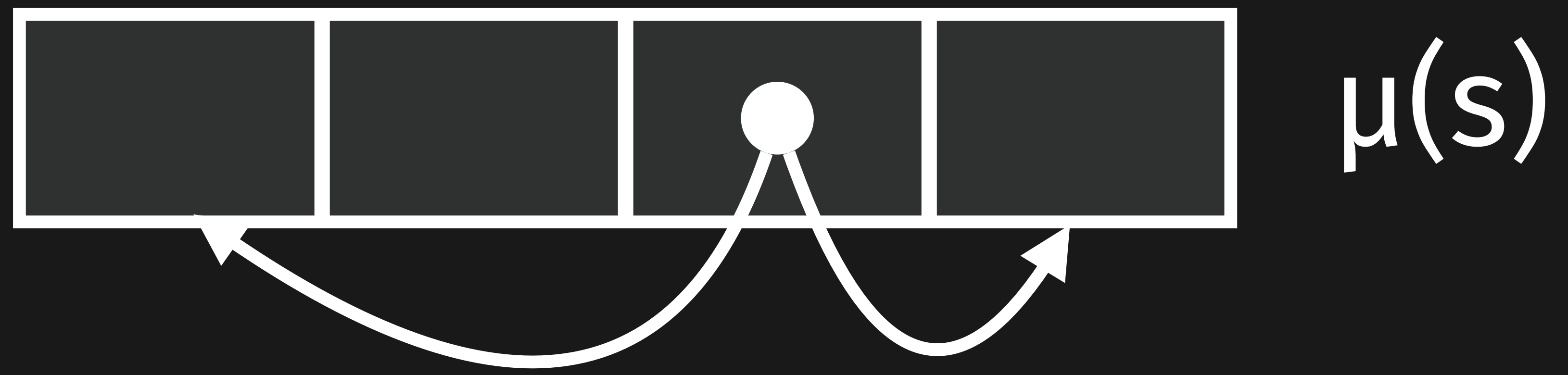
Lola



Viper

Ghost Memory

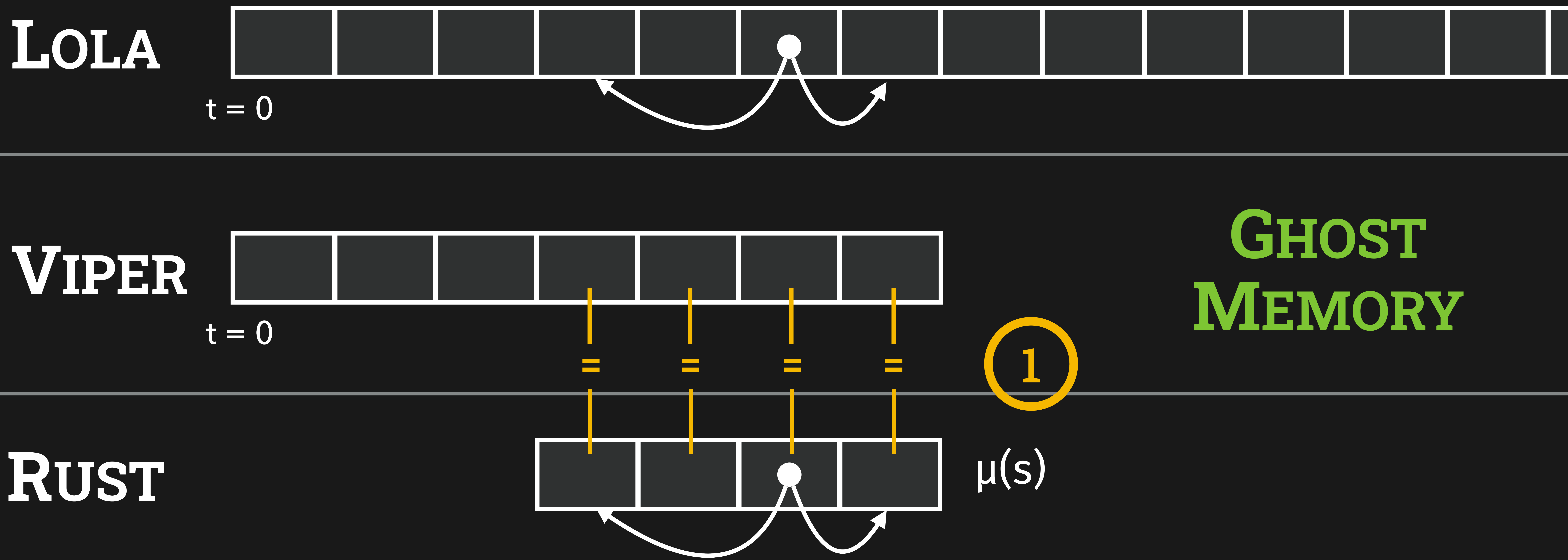
Rust



memory ≜ finite excerpt of stream

Block II Verification — Idea

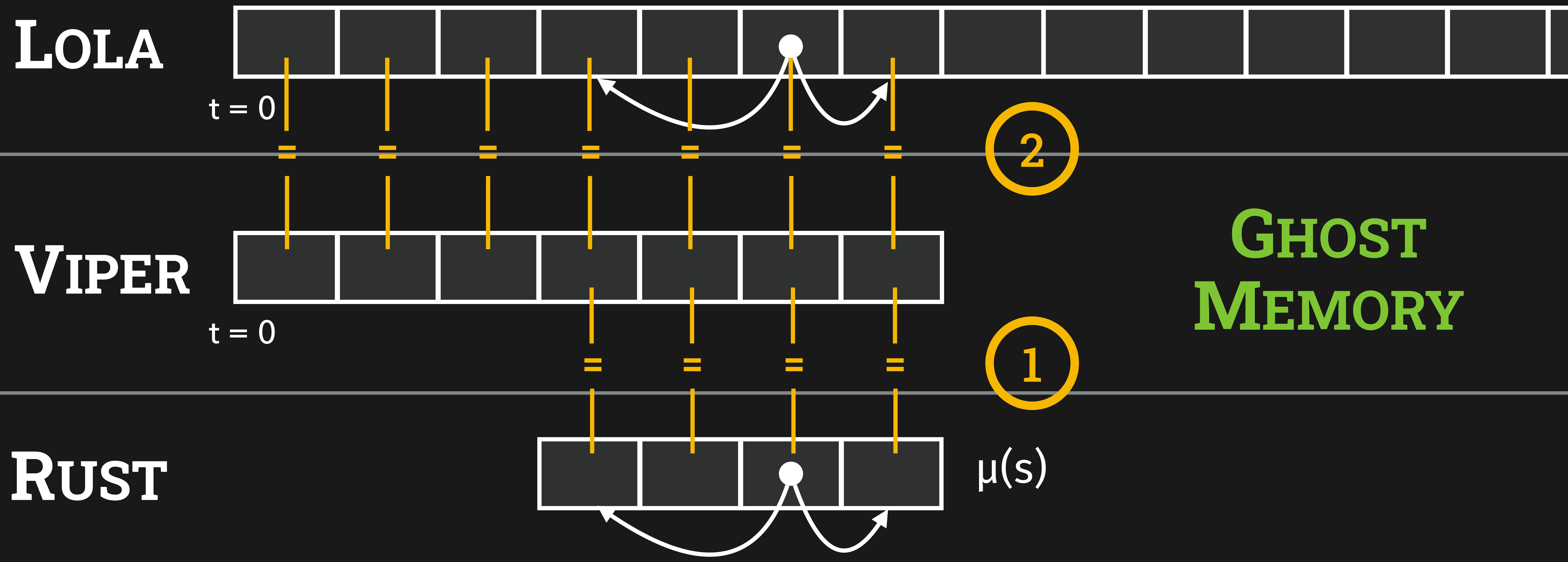
stream ≜ infinite sequence of values



memory ≜ finite excerpt of stream

Block II Verification — Idea

stream ≜ infinite sequence of values



memory ≜ finite excerpt of stream

Block II Verification — Realization

```
while let Some(input) = get_input() {

    mem.add_input(&input);

    [[ EVALUATION LOGIC ]]

    mem.store(new_tooHigh);
    gm.store(new_tooHigh);

    if trigger_1 { emit( trigger_1_msg) }
}
```

Block II Verification — Realization

```
#[invariant="forall i: usize ::
    (0 ≤ i && i < μ(a))
        ⟹ mem.get_a(i) == gm.get_a(iter - i)
"]
```

①

```
while let Some(input) = get_input() {
    mem.add_input(&input);
    [[ EVALUATION LOGIC ]]
    mem.store(new_tooHigh);
    gm.store(new_tooHigh);
    if trigger_1 { emit( trigger_1_msg) }
}
```

Block II Verification — Realization

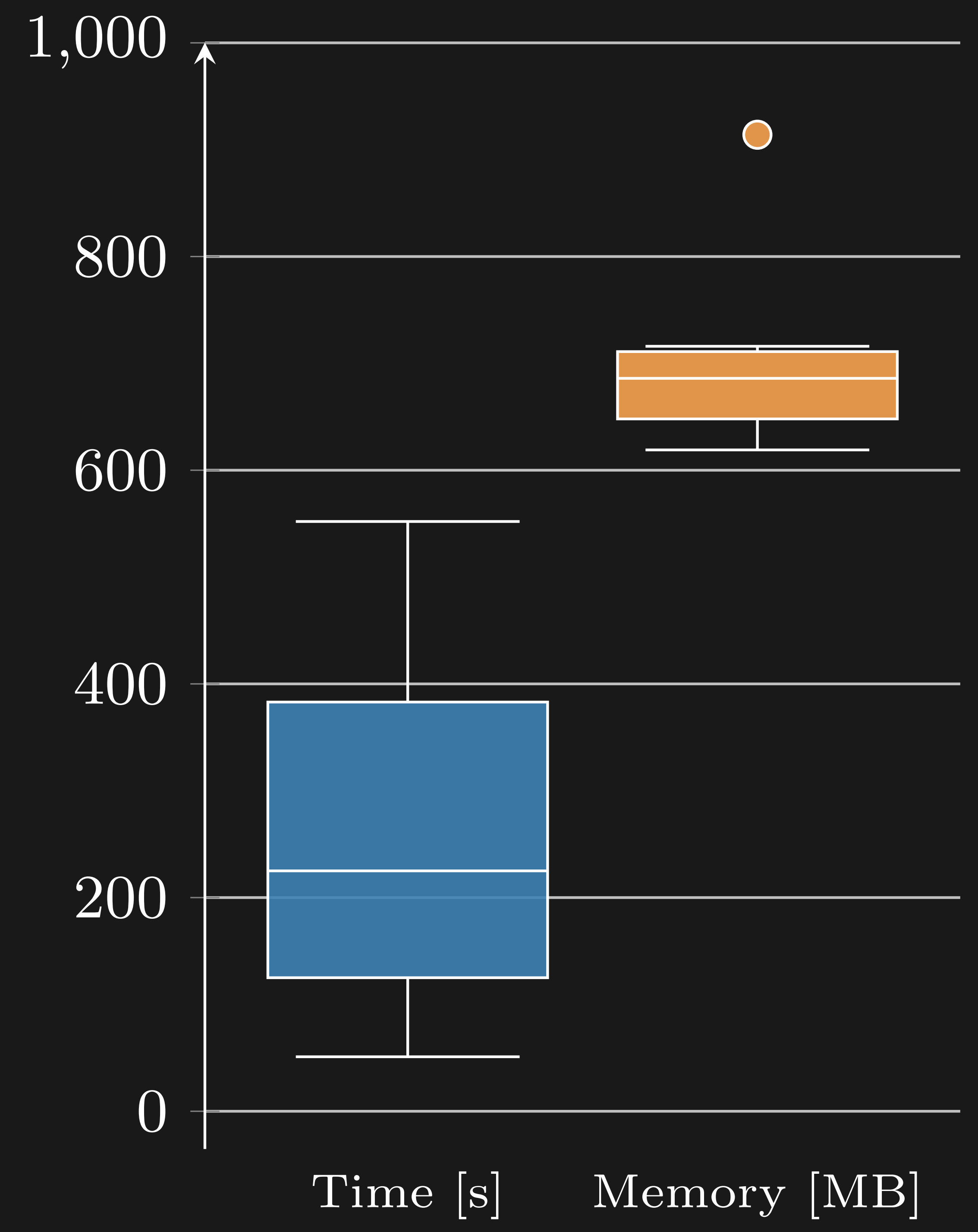
```
#[invariant="forall i: usize ::
    (0 ⩽ i && i < μ(a))
        ⟹ mem.get_a(i) == gm.get_a(iter - i)
"]
```
①

```
#[invariant="new_tooHigh == gm.get_a(iter - 2) > 500 ∧ …"]
```
②

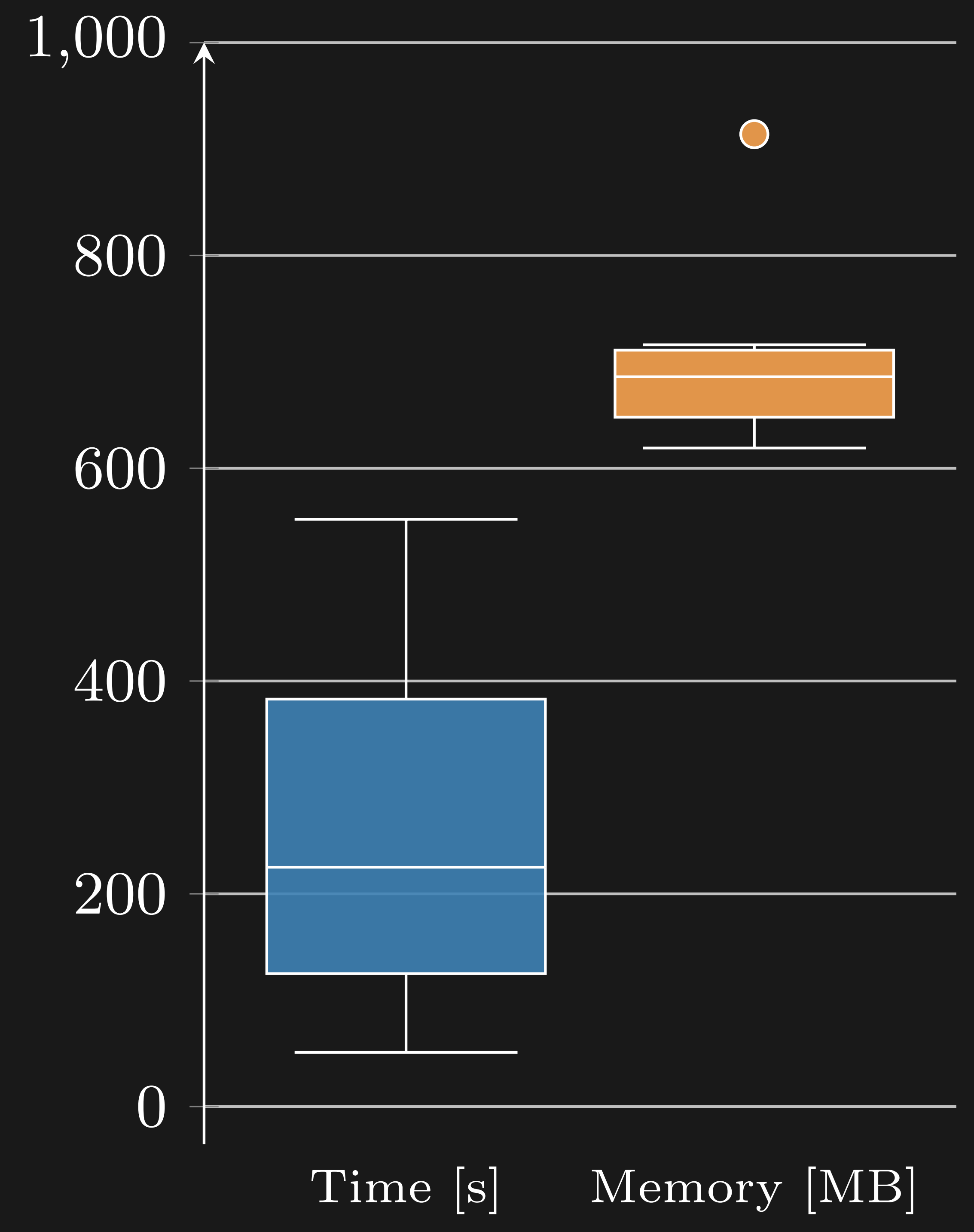
```
while let Some(input) = get_input() {
    mem.add_input(&input);
    [[ EVALUATION LOGIC ]]
    mem.store(new_tooHigh);
    gm.store(new_tooHigh);
    if trigger_1 { emit( trigger_1_msg) }
}
```

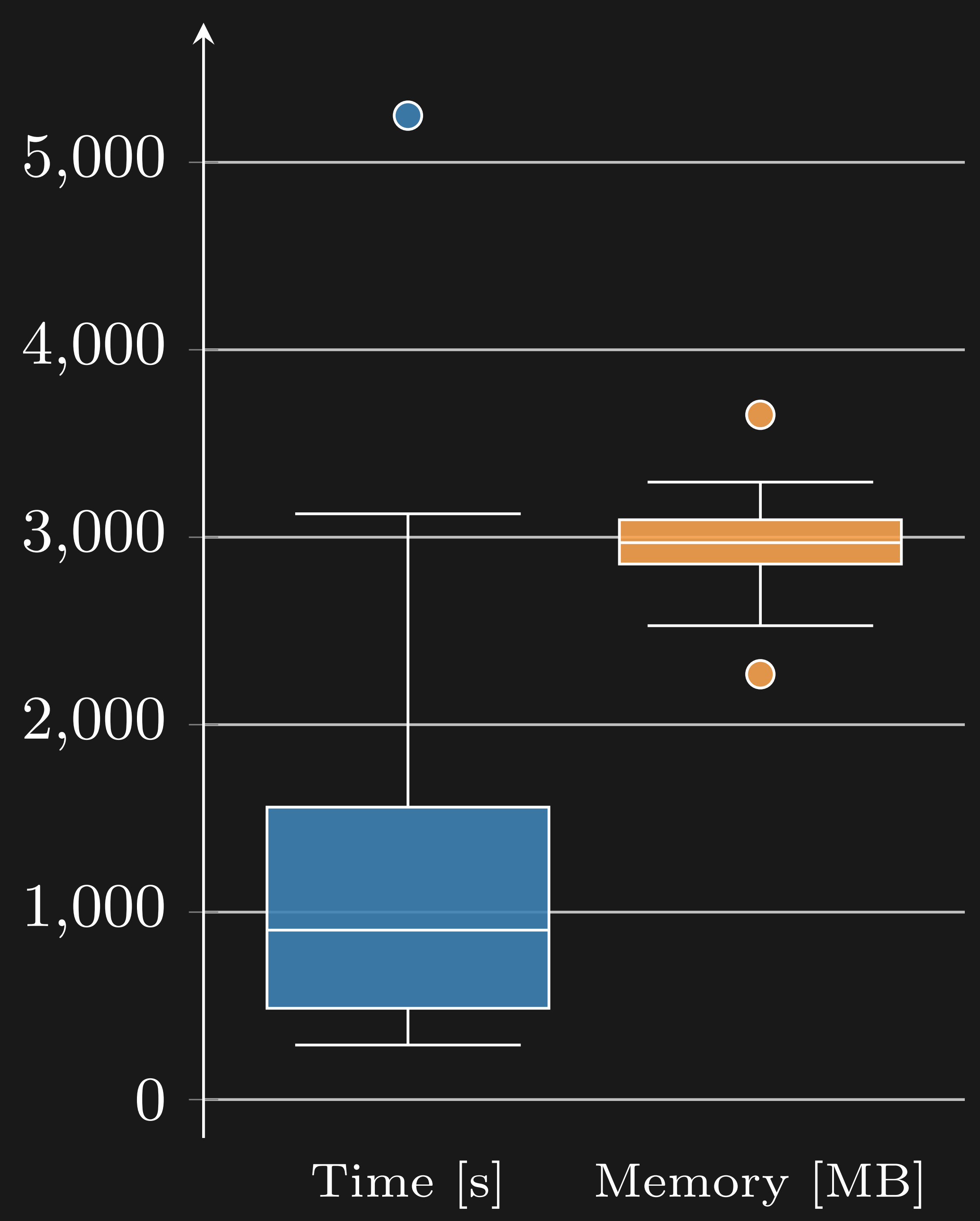
Block II Viability

Block II Viability

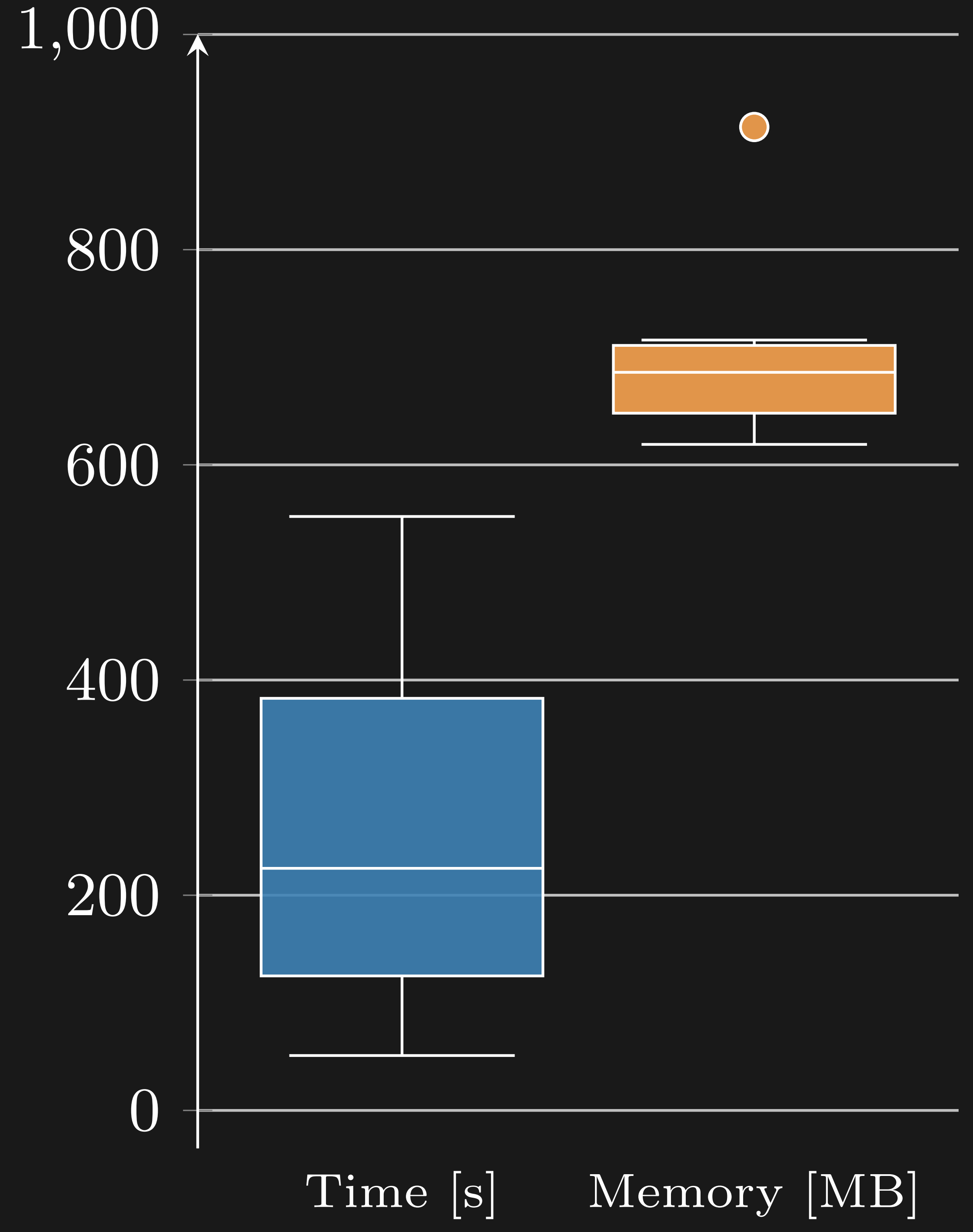


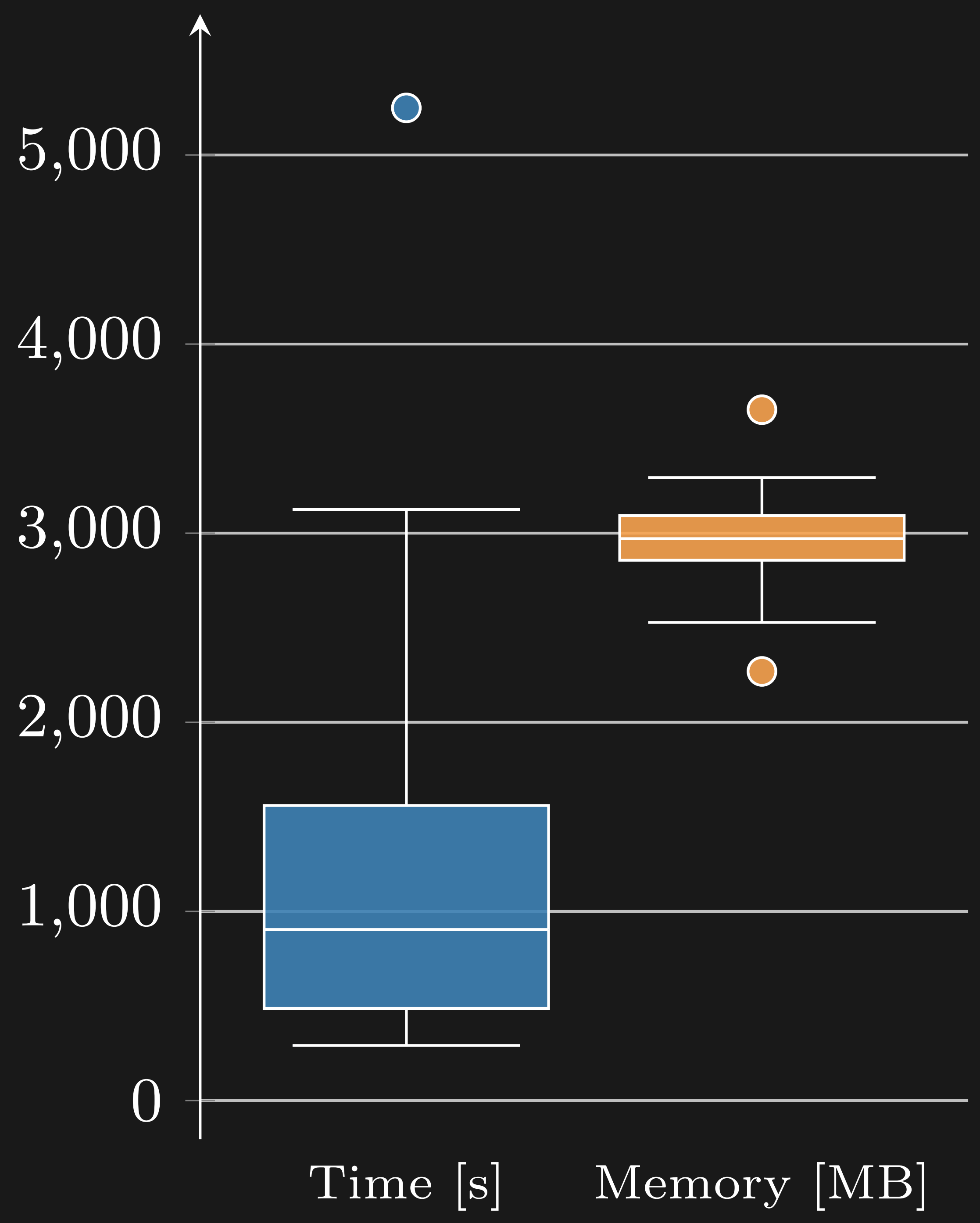
Block II Viability



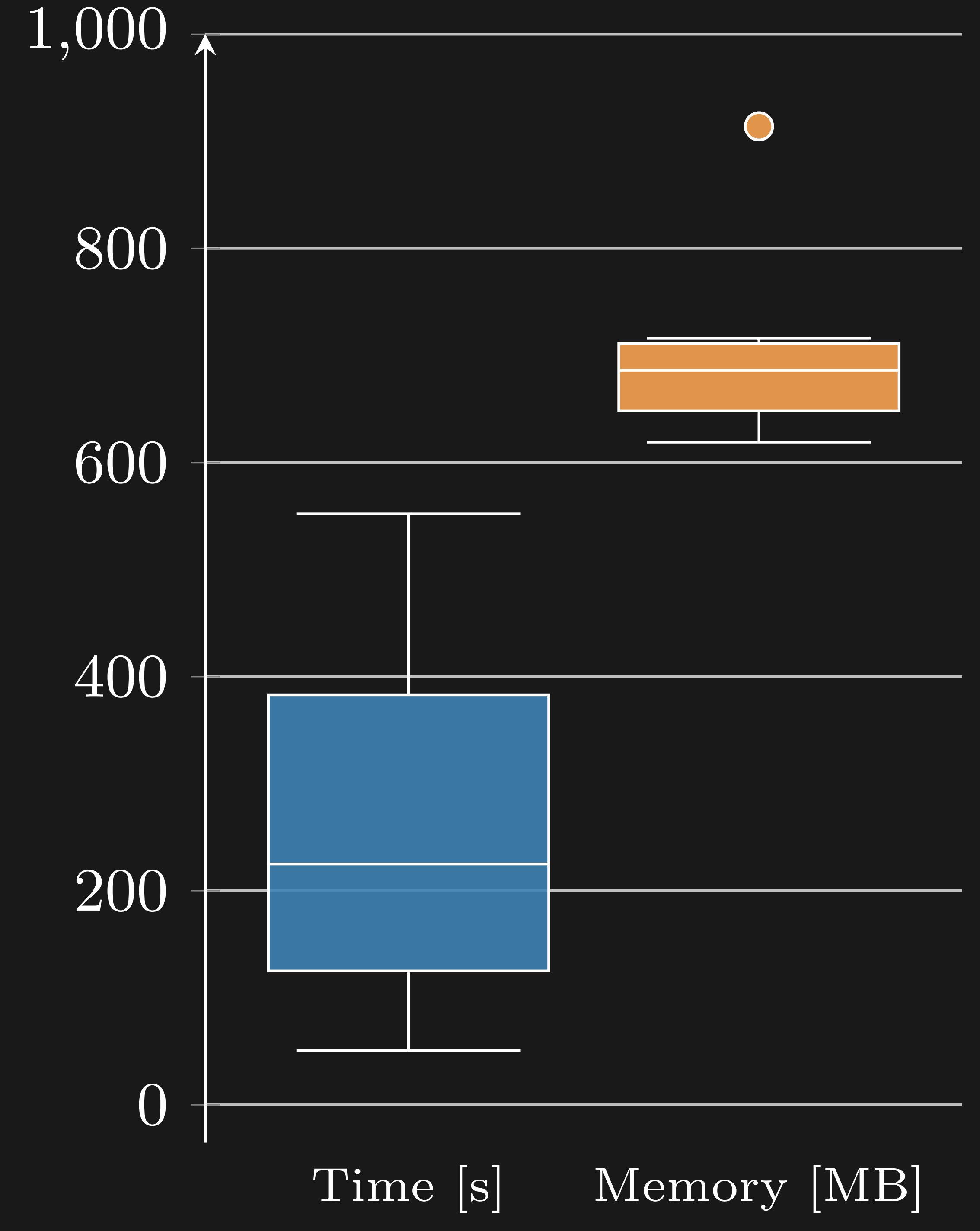


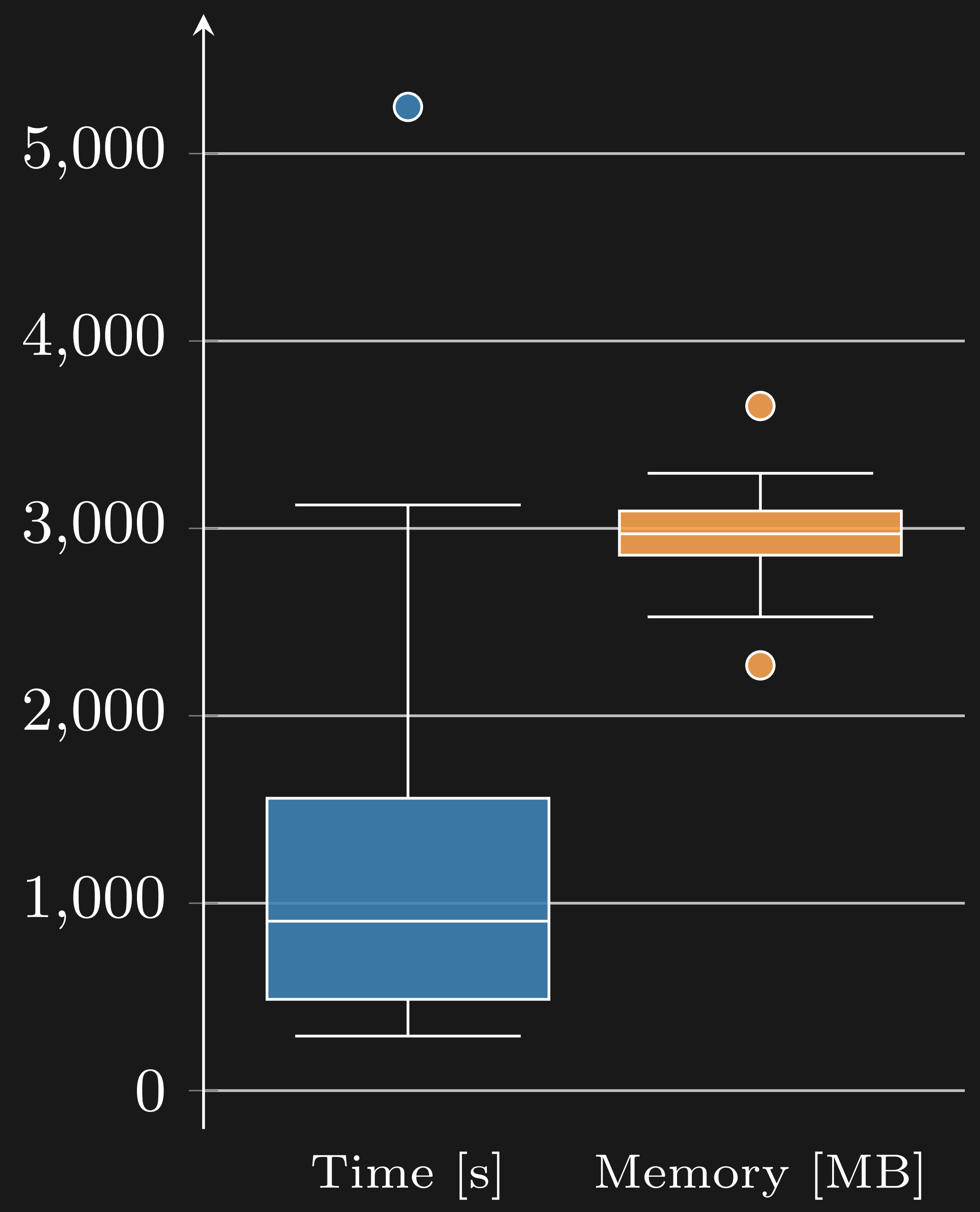
Block II Viability





Block II Viability





Detected implicit assumption on input stream!

Block II Viability


flight phase

Detected implicit assumption on input stream!

```
input time
input sensor

output δtime :=
    abs(time - time.offset(by: -1, dft: 0))
output δsensor :=
    abs(sensor - sensor.offset(by: -1, dft: 0))
output diff := δsensor / δtime
```

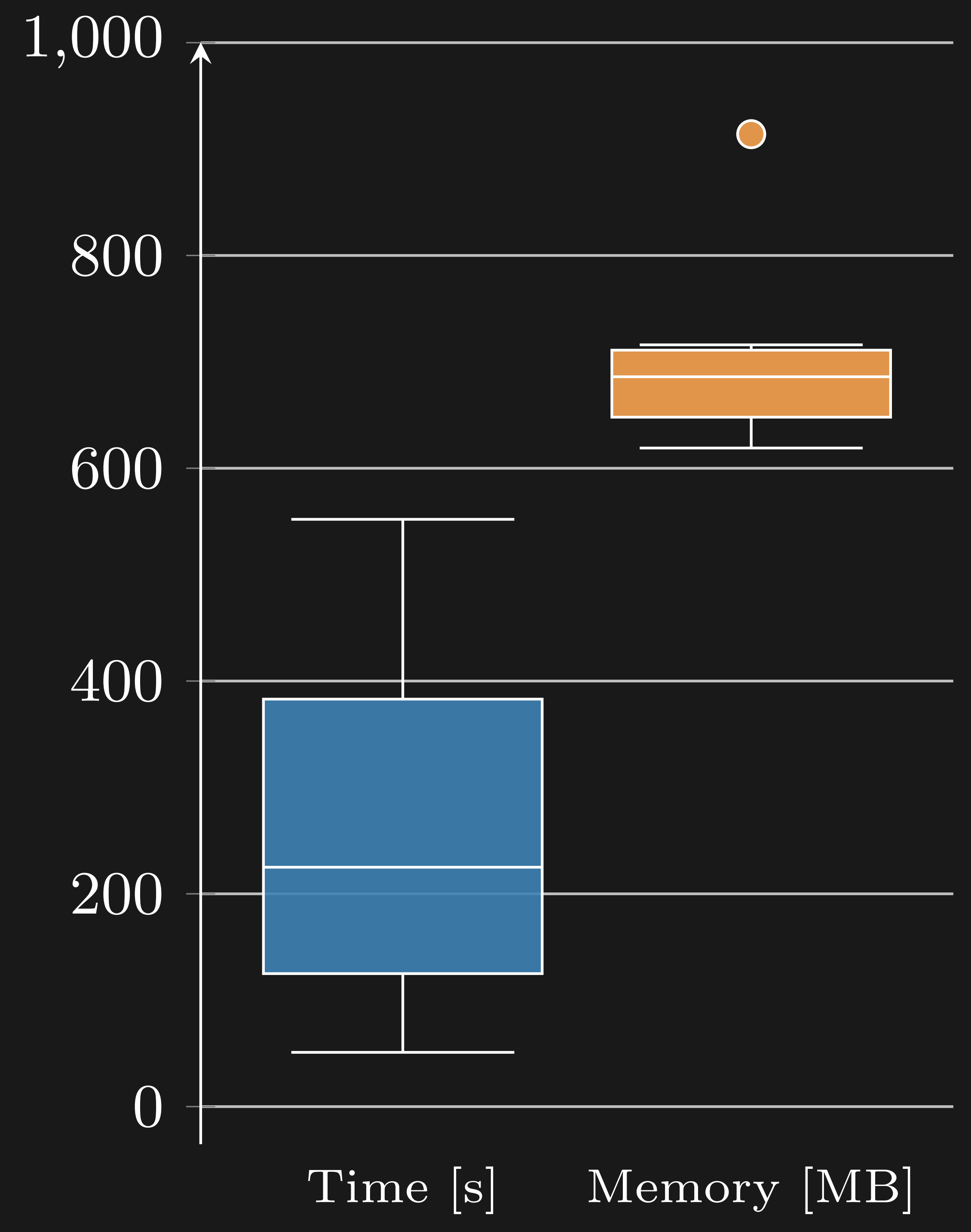
Block II Viability

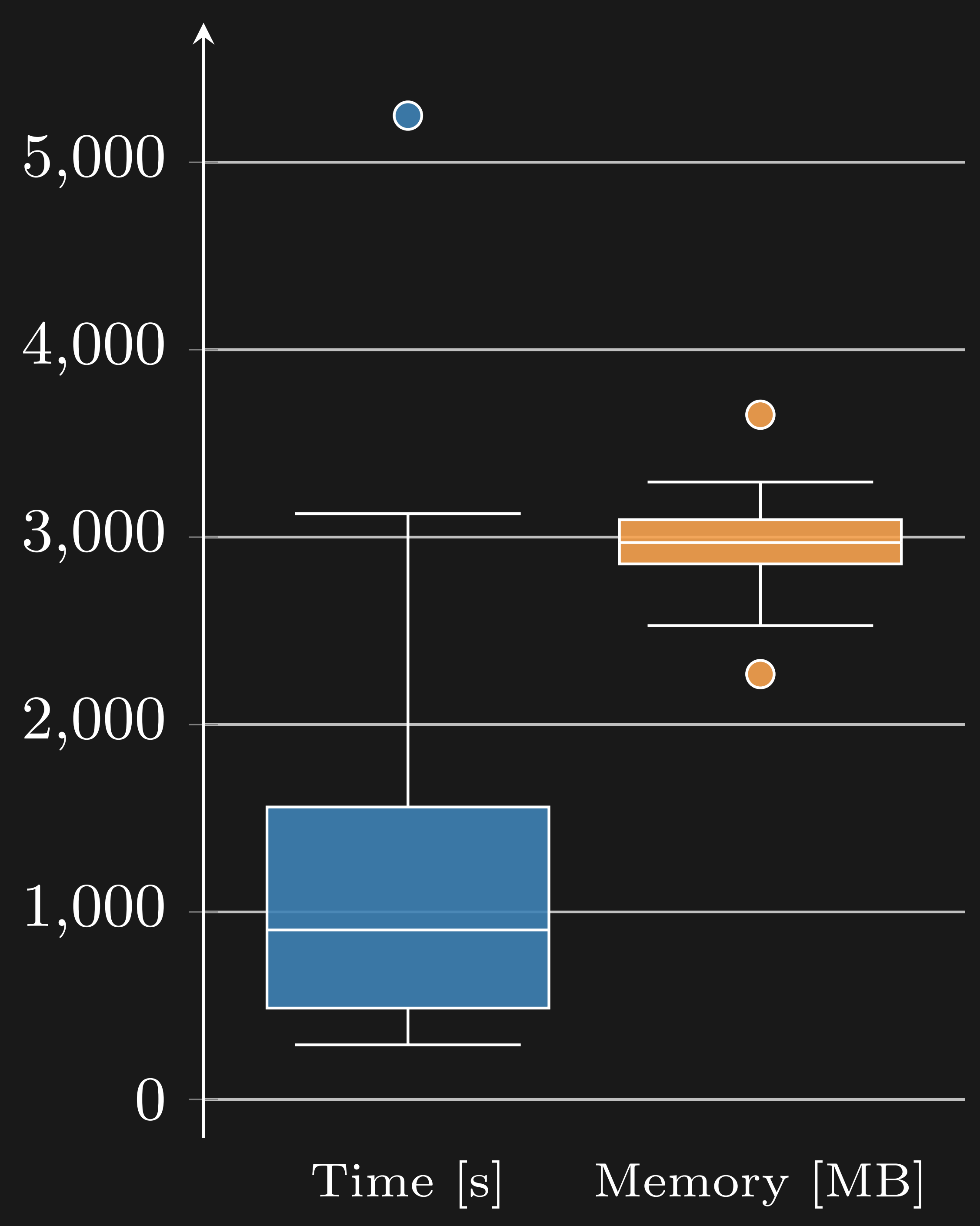

flight phase

Detected implicit assumption
on input stream!

```
input time
input sensor

output δtime :=
    abs(time - time.offset(by: -1, dft: 0))
output δsensor :=
    abs(sensor - sensor.offset(by: -1, dft: 0))
output diff := δsensor / δtime
```

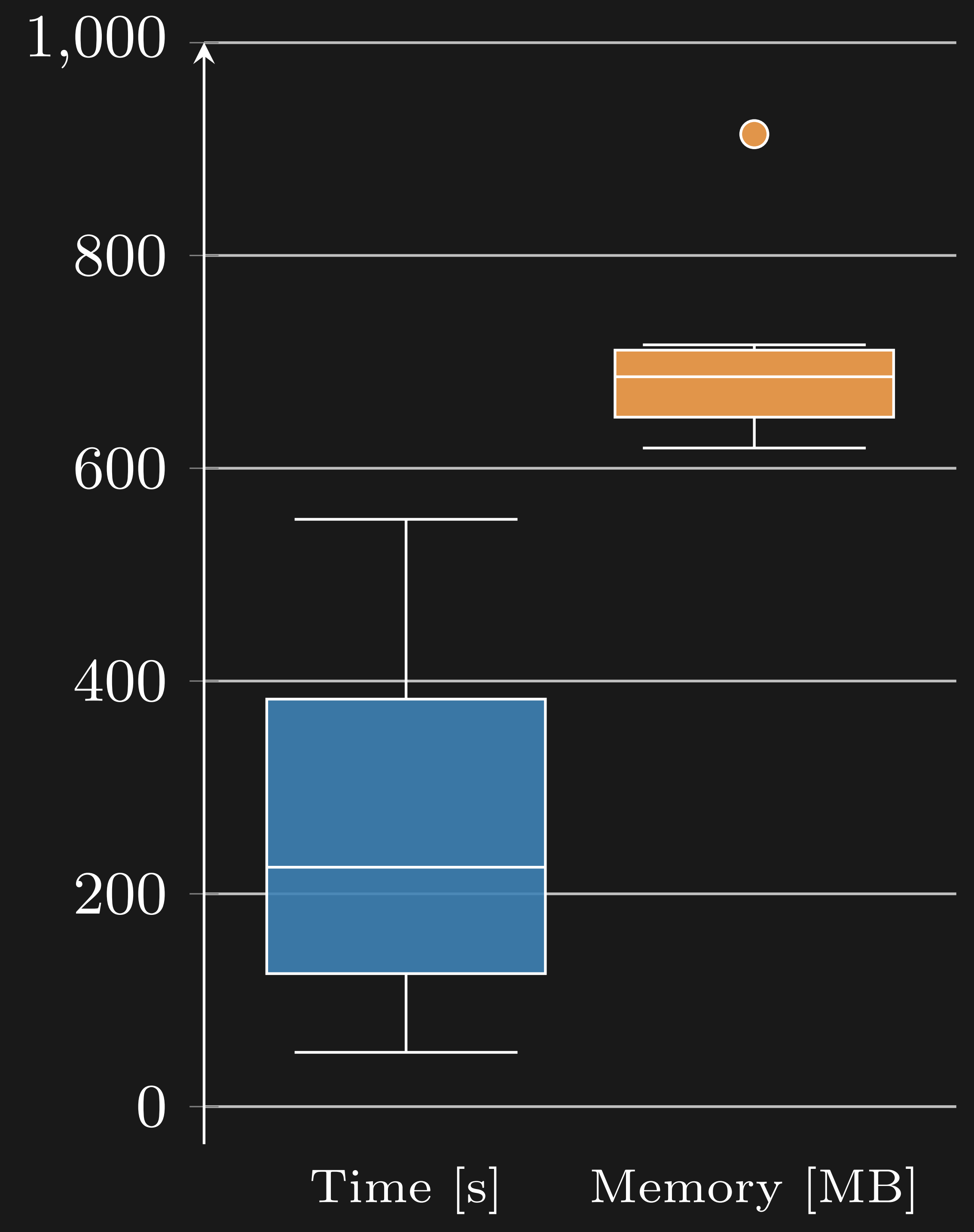
Block II Viability

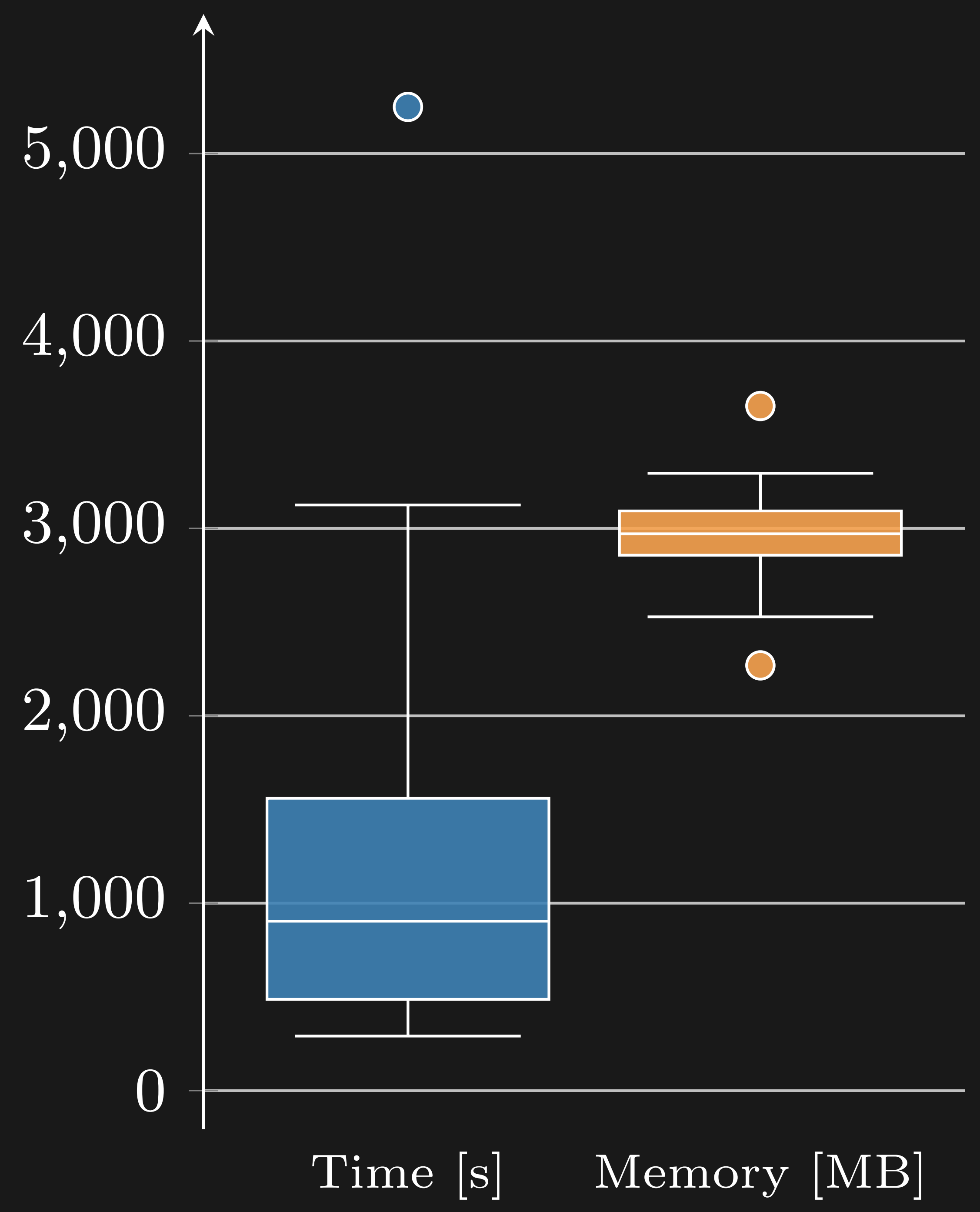




Detected implicit assumption on input stream!

Block II Viability





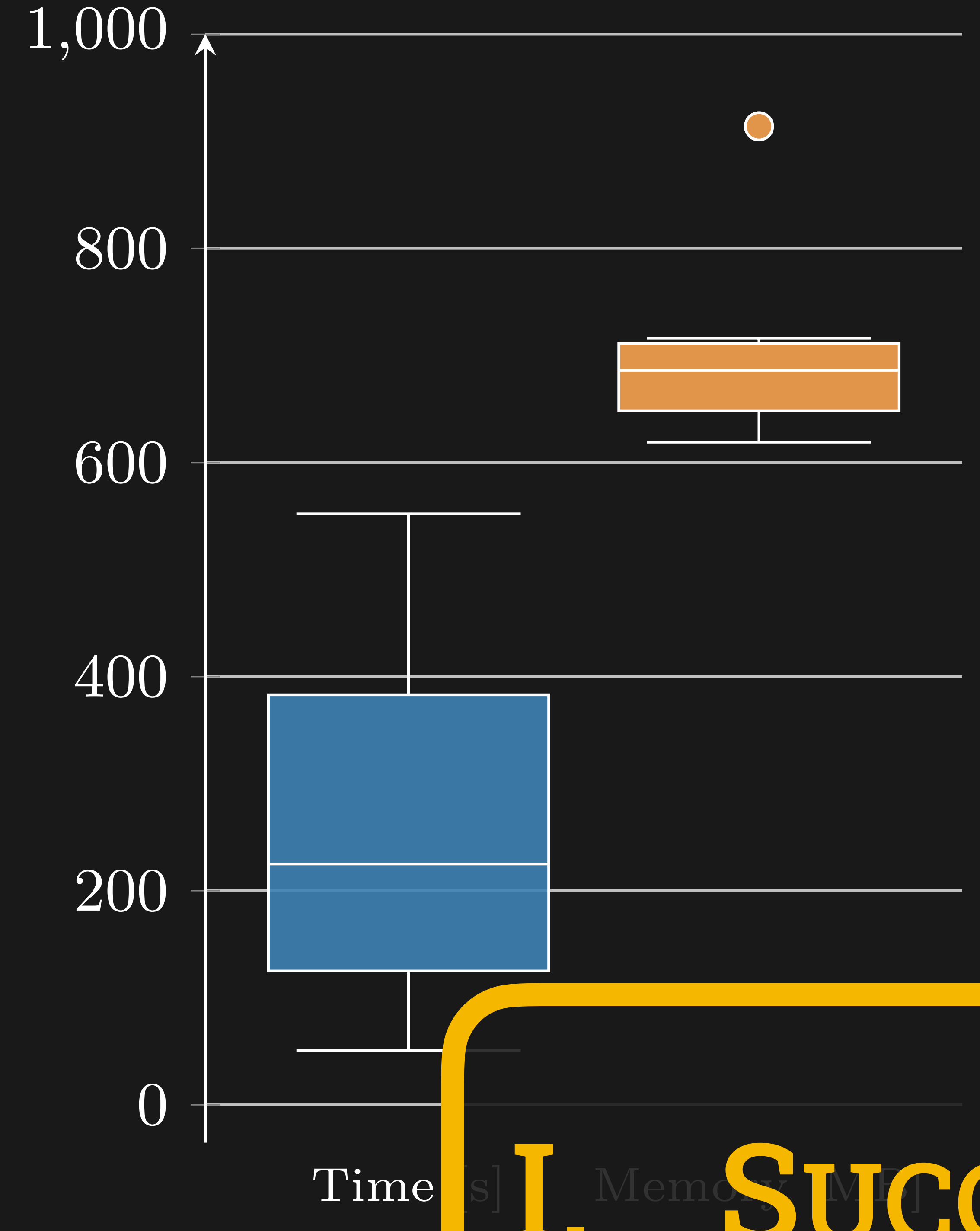
Detected implicit assumption on input stream!

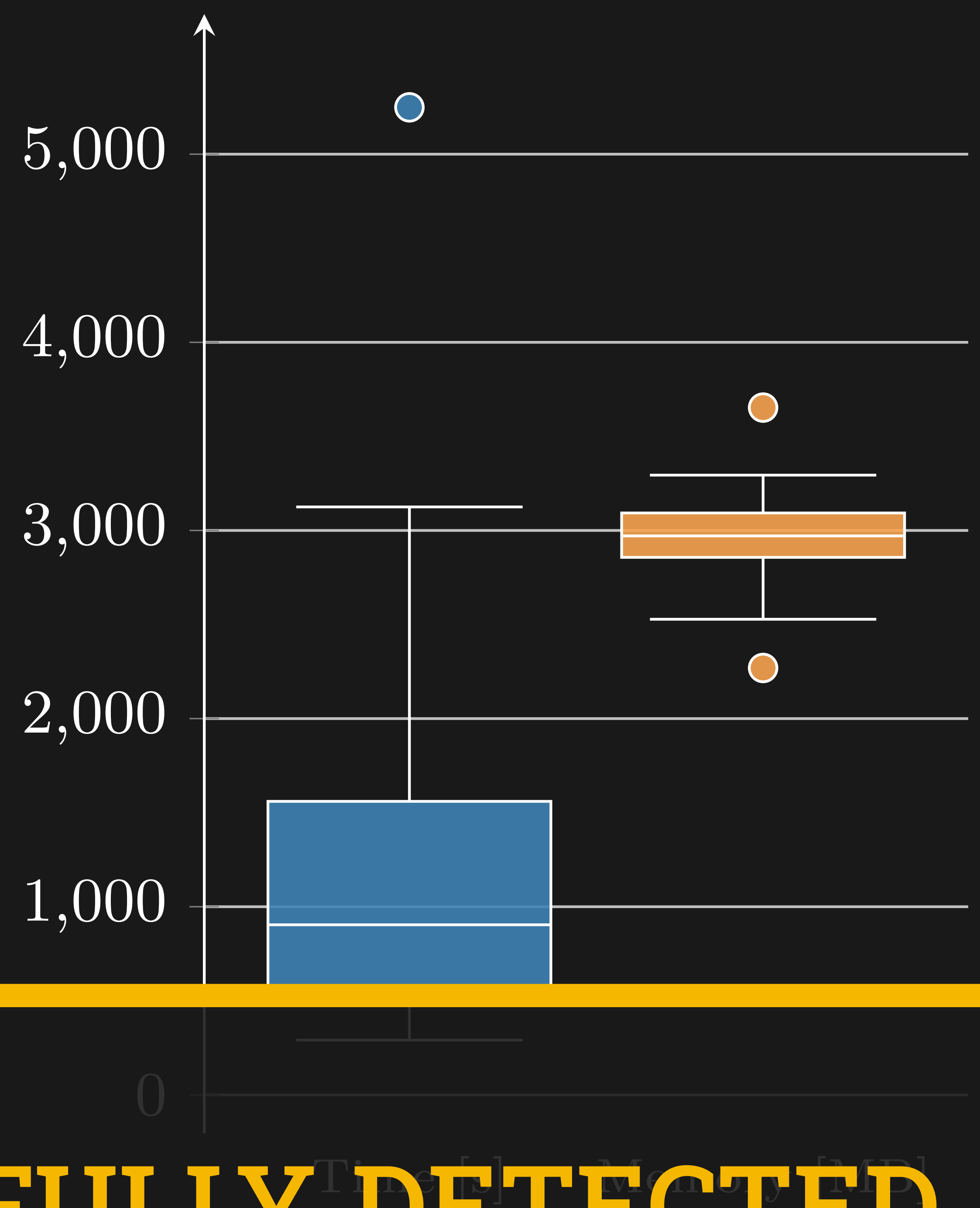
On corrected spec:

6 – 16min
1.38 – 1.66GB

2 T/O (10%)
4 fails (20%)

Block II Viability





Detected implicit assumption on input stream!

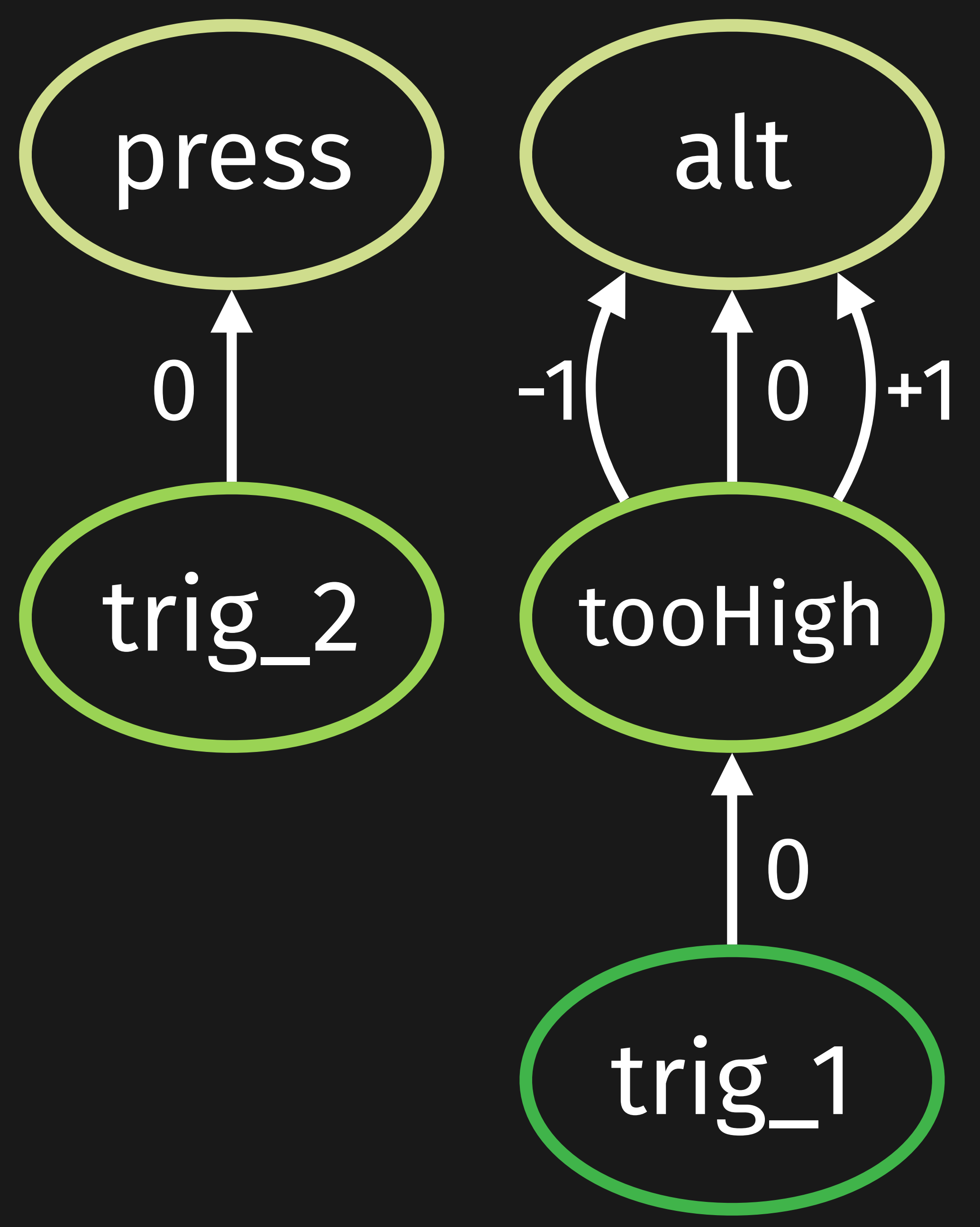
On corrected spec:

6 – 16min
1.38 – 1.66GB

I. Successfully detected specification error
II. Verified monitors for complex specifications

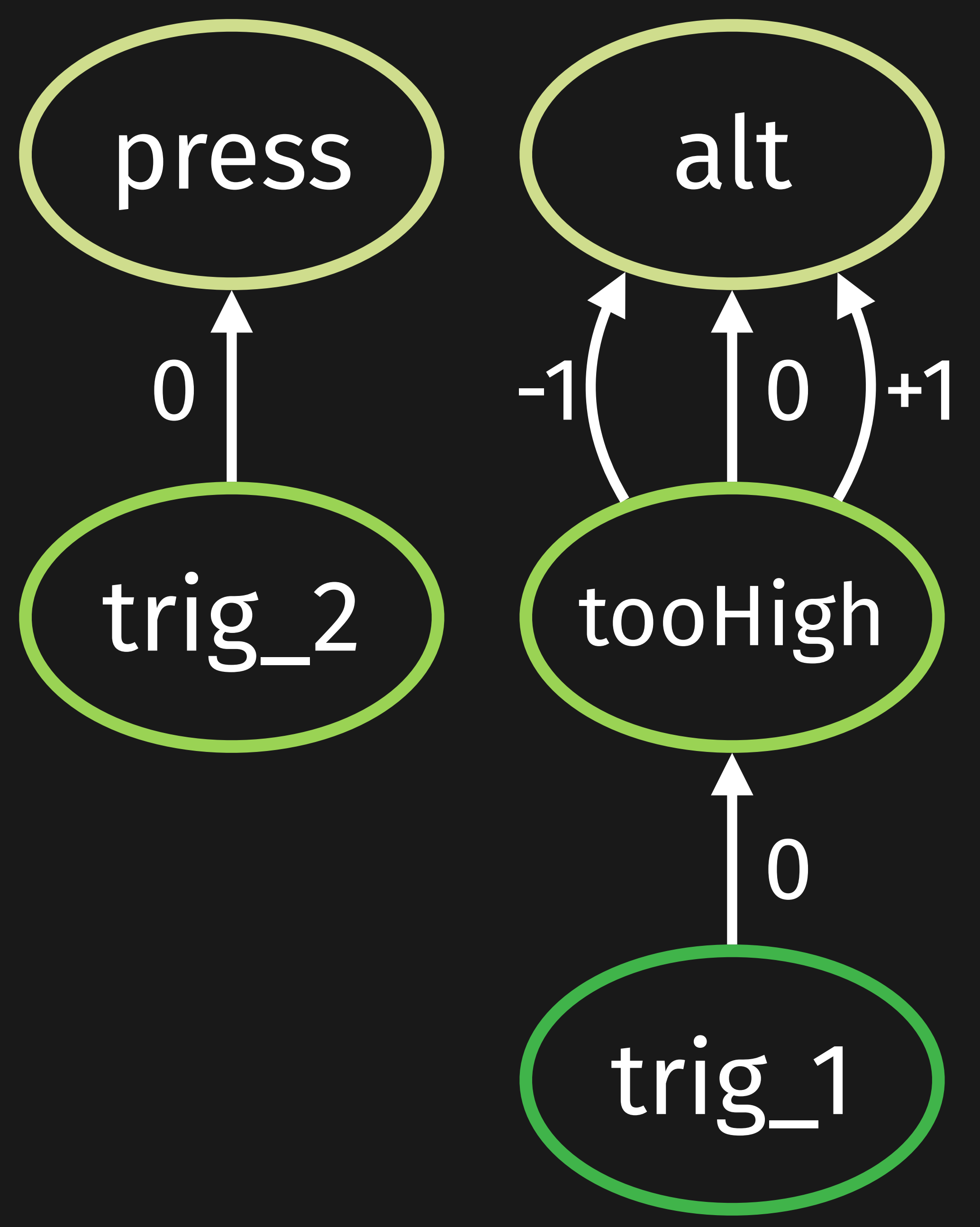
Block II Concurrent Evaluation

```
input alt
input pressure

output tooHigh :=
    alt.offset(by: -1, dft: 0) > 500
  ∧ alt > 500
  ∧ alt.offset(by: +1, dft: 0) > 500

trigger tooHigh
trigger pressure < 0
```

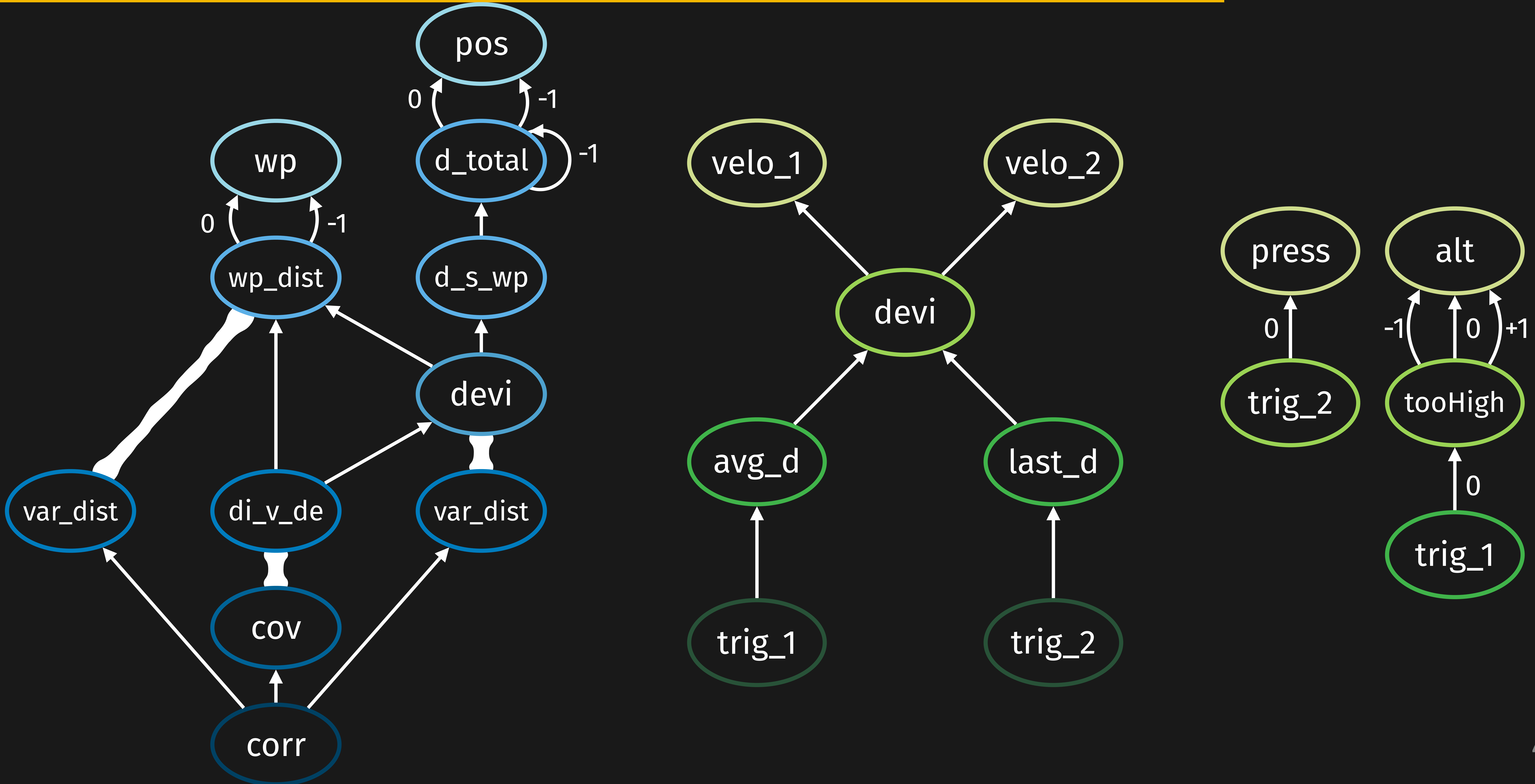


Block II Concurrent Evaluation

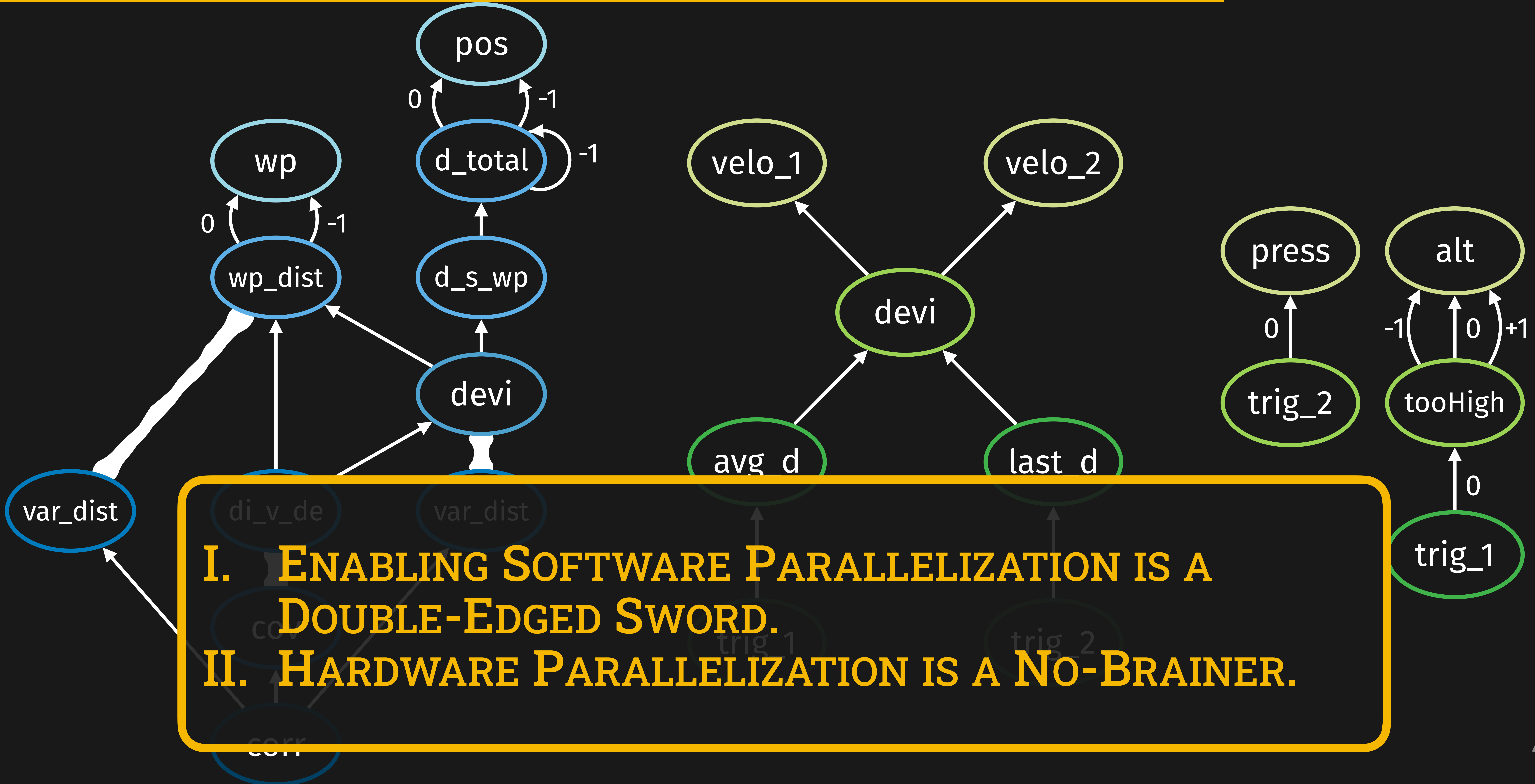
```
let (v_1, ..., v_n) = crossbeam::scope(|scope| {
    let handle_tooHigh = scope.spawn(move |_| {
        eval_tooHigh(&memory)
    });
    let handle_trigger_2 = scope.spawn(move |_| {
        eval_trigger_2(&memory)
    });
    (
        handle_s1.join().unwrap(),
        ...,
        handle_sn.join().unwrap()
    )
}).unwrap()
```



Block II A Double-Edged Sword



Block II A Double-Edged Sword



I. Enabling Software Parallelization is a Double-Edged Sword.
II. Hardware Parallelization is a No-Brainer.

Block II Summary SW Compilation

VIPER

verifies

observes

Lola Specification

Compilation

+ Annotation Generation

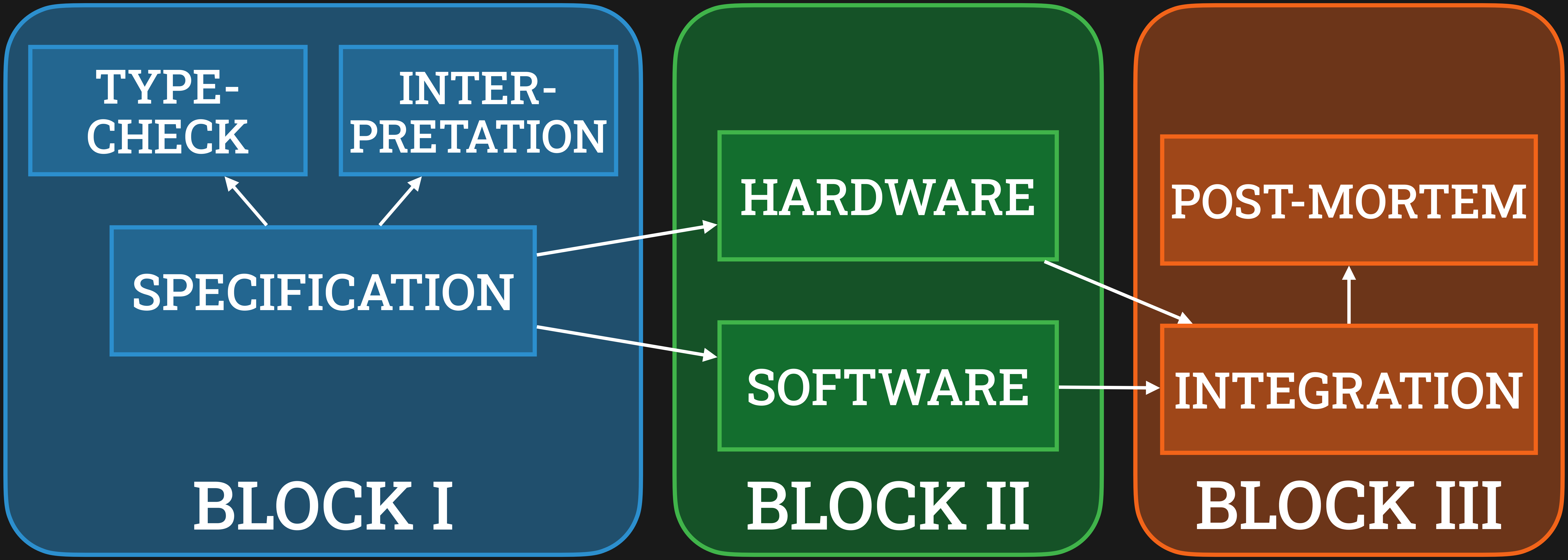
```
Impl Monitor {
 #[invariant = … ]
 while let Some(i)
        = get_input() {
  …
 }}
```

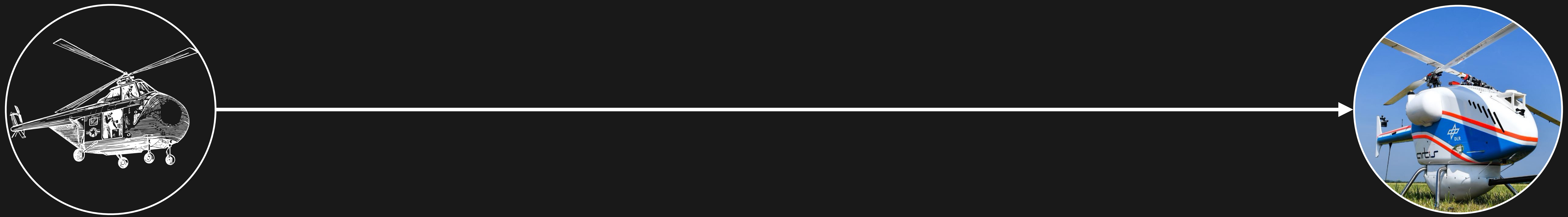
Rust Code

Monitor

```
01010010
01010110
00110010
00110000
```

Block III Integration



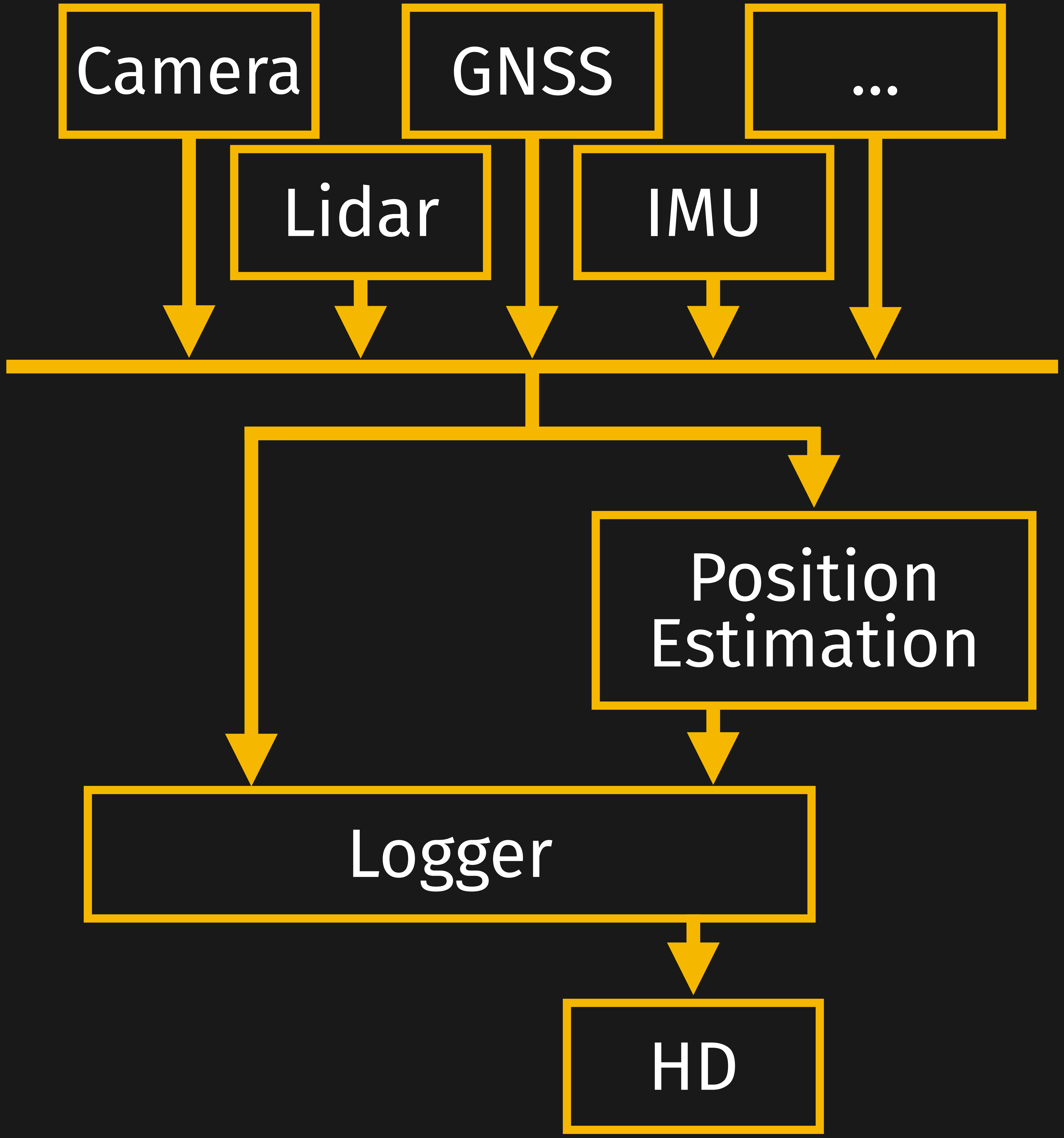


Block III: DLR's superARTIS

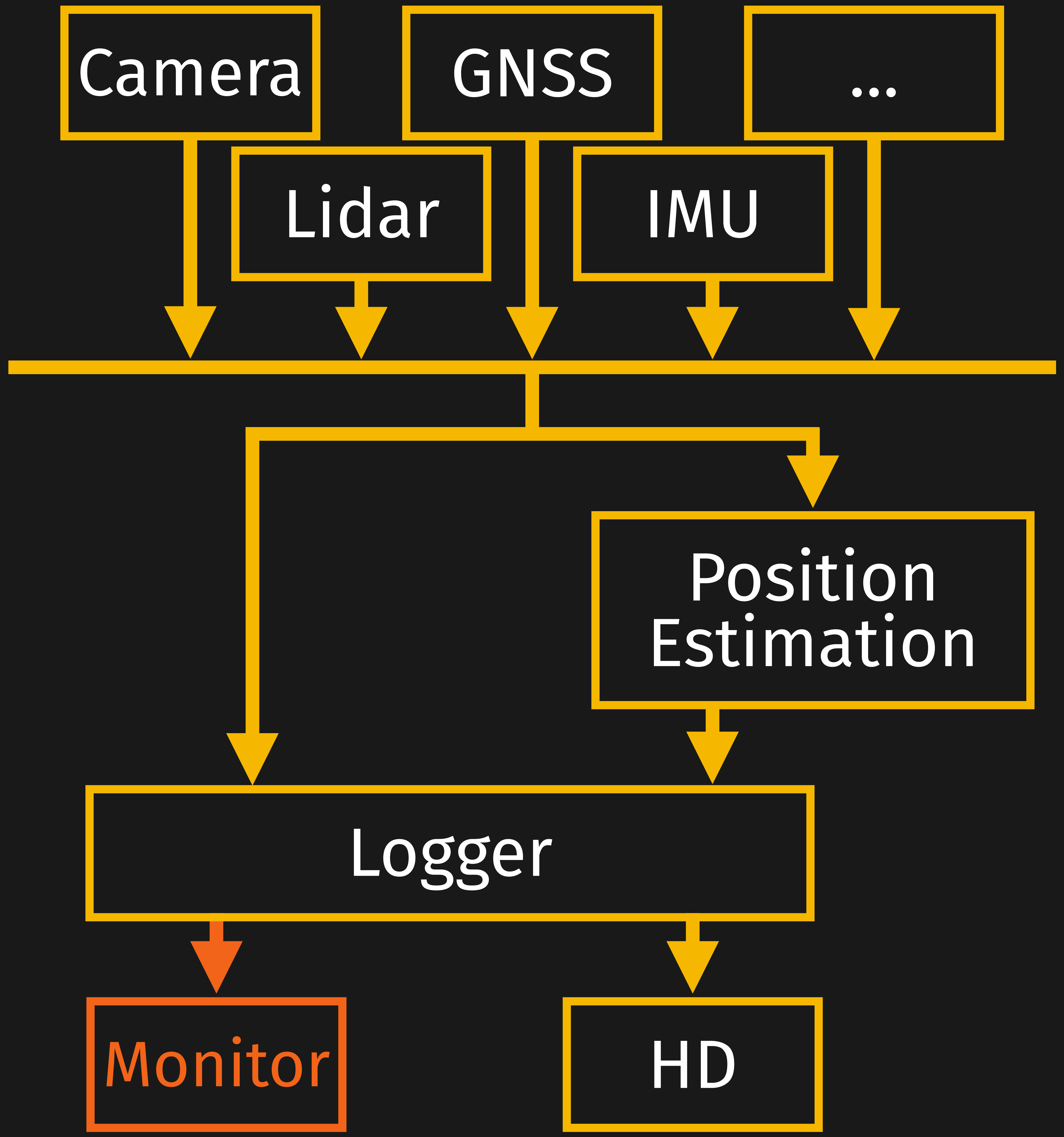
Block III: DLR's superARTIS


TIS

Block III DLR's superARTIS



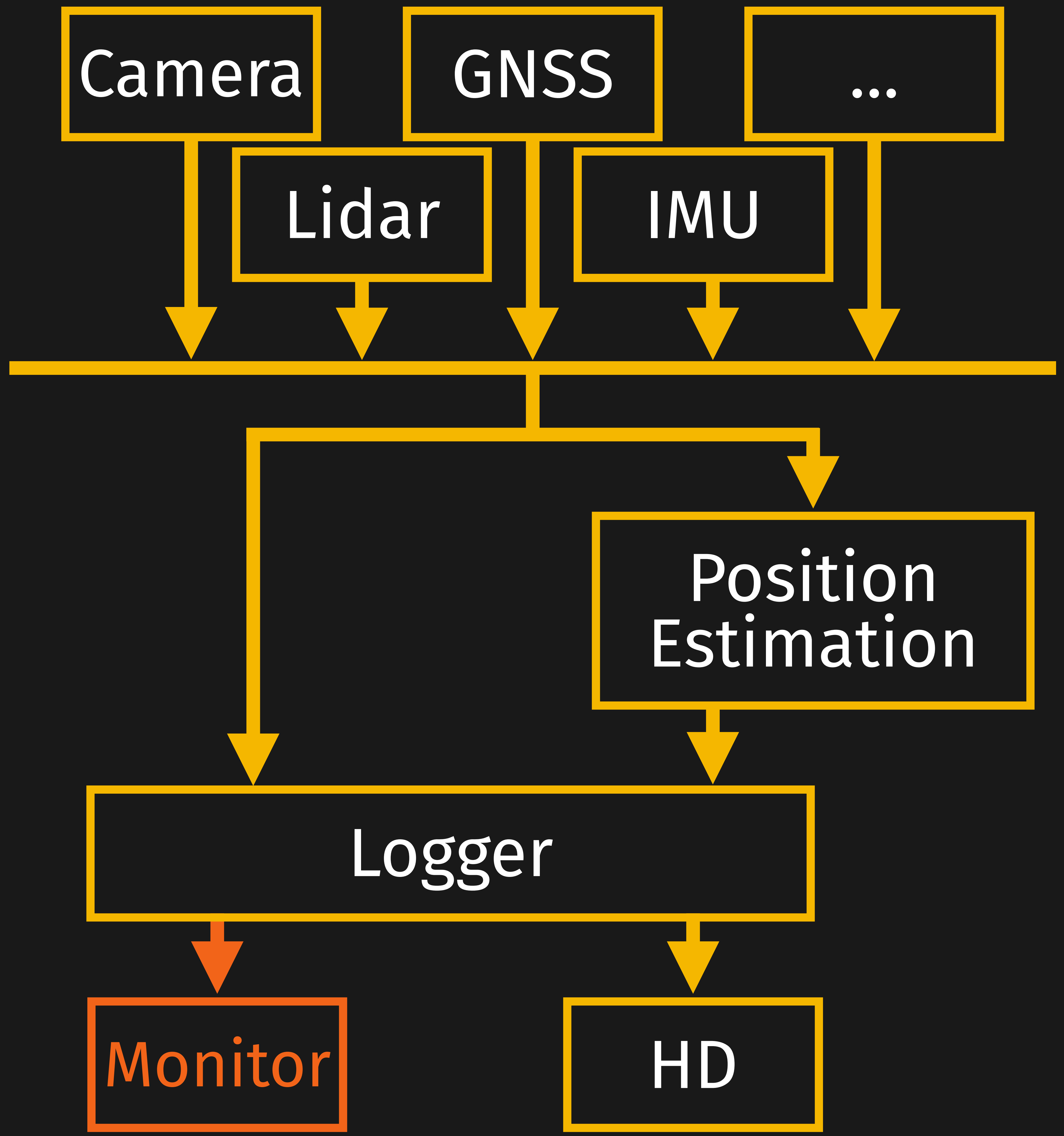
Block III DLR's superARTIS



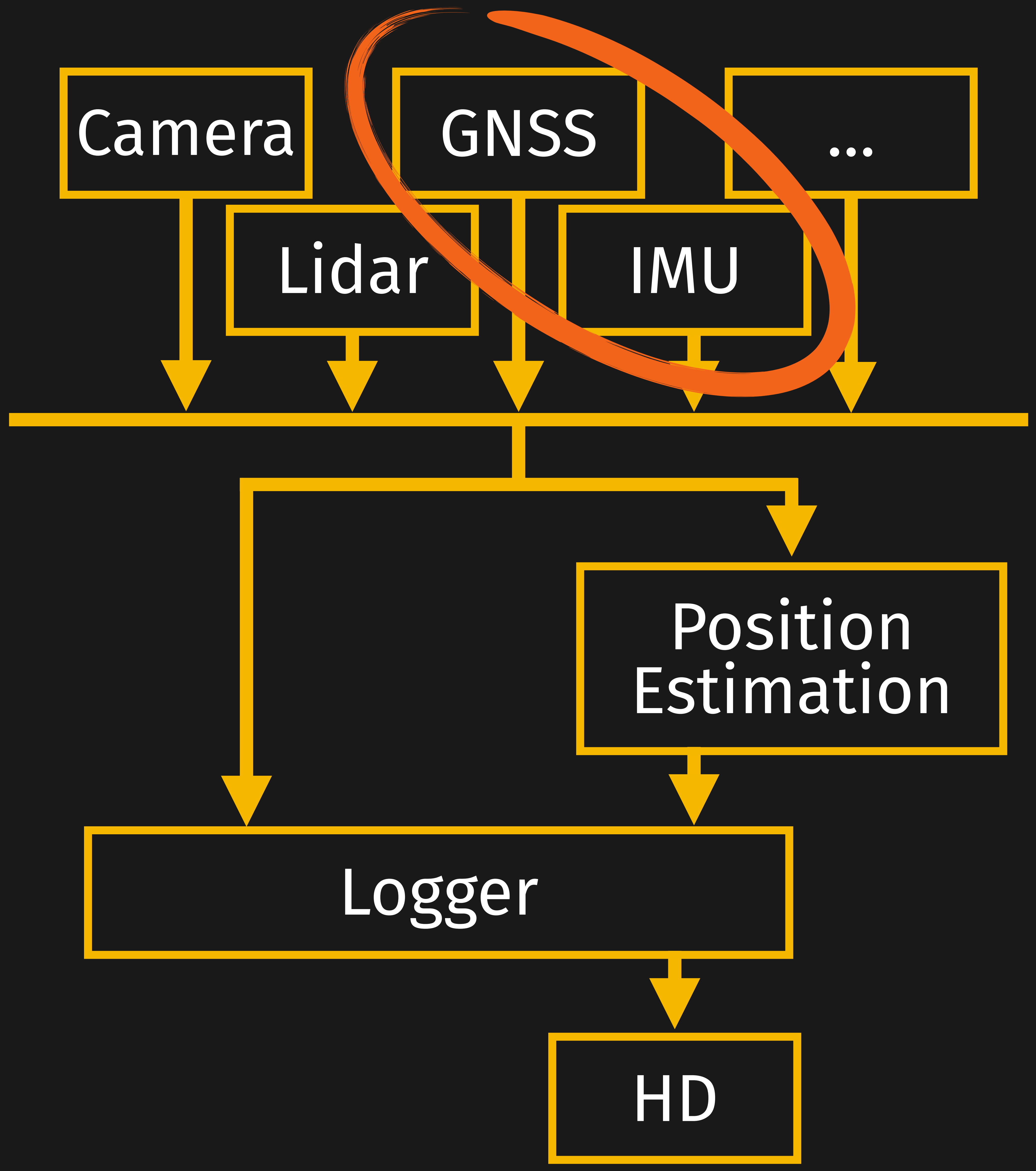
Snooping v Messages

Factors:

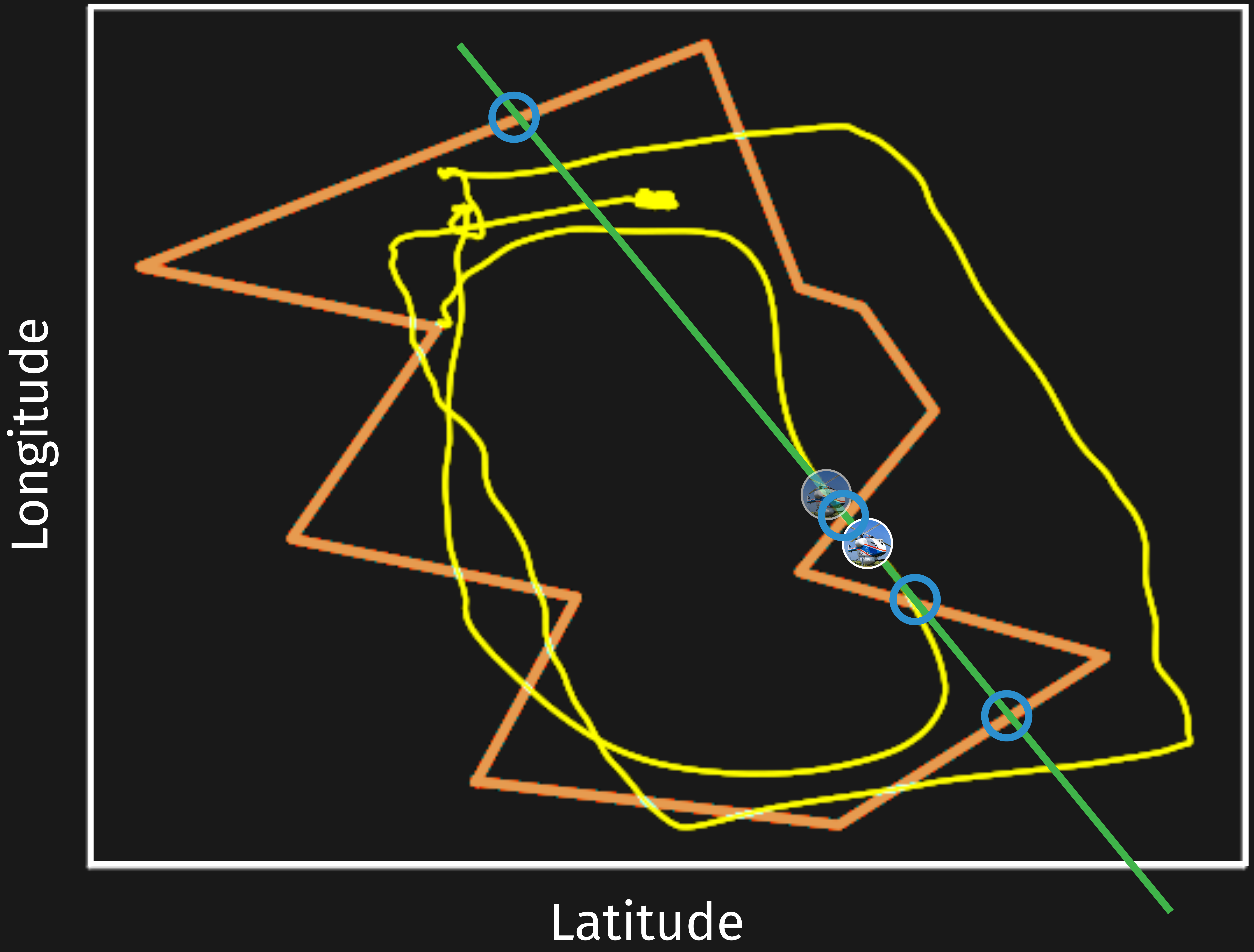
- ❖ Bus Utilization / Bus Allocation
- ❖ Resource Availability



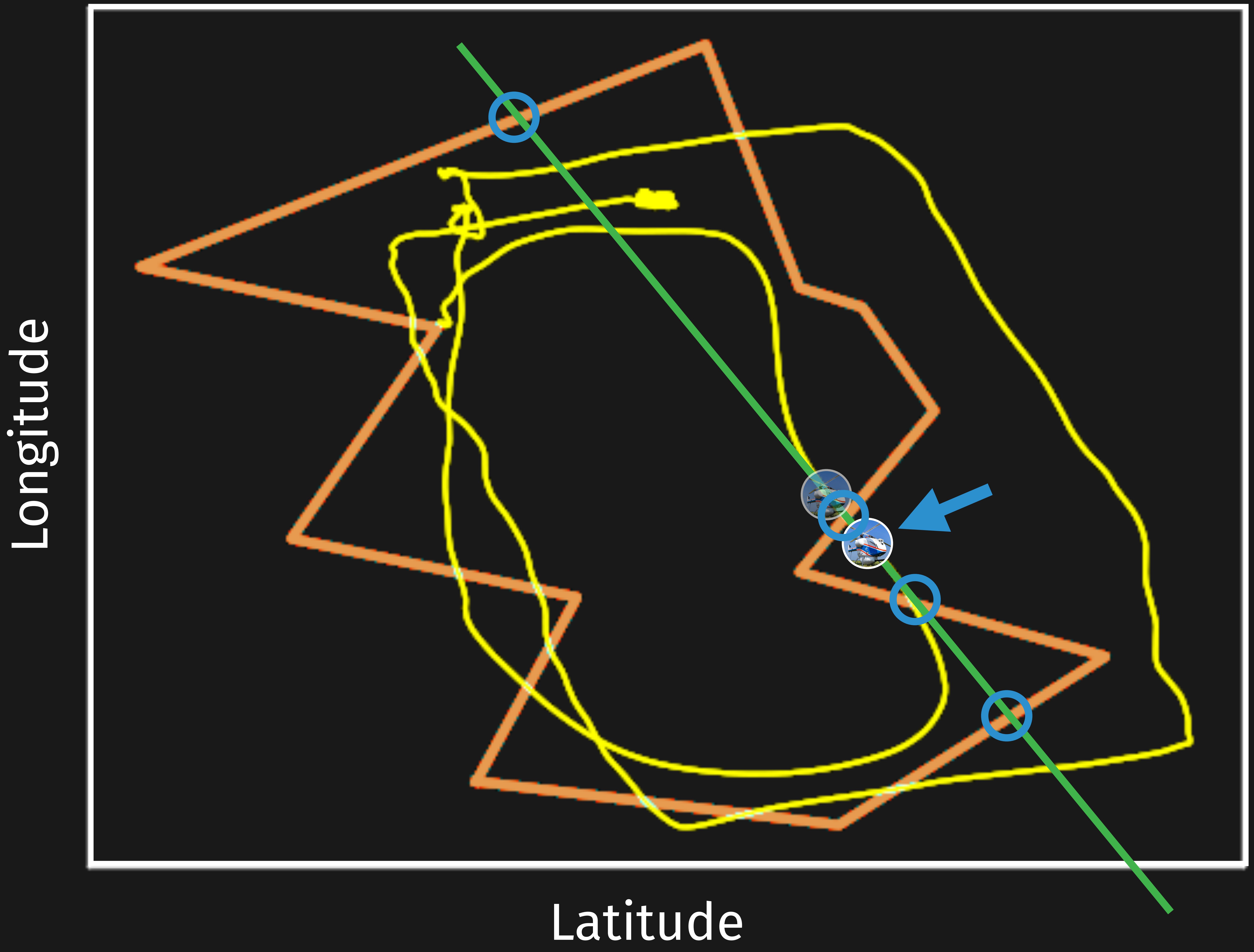
Block III Specs I: (Cross-) Validation



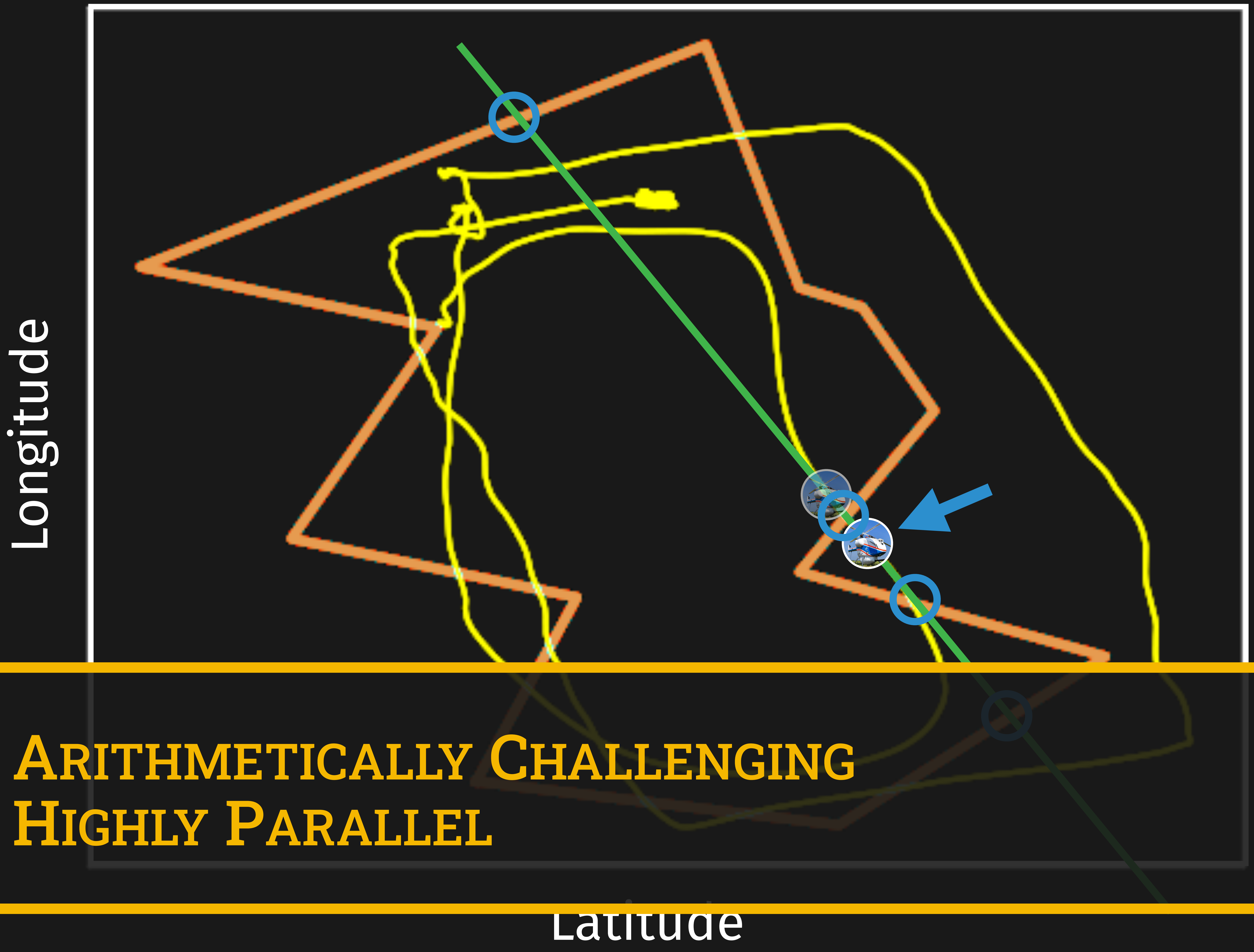
Block III Specs II: Geo-Fencing



Block III Specs II: Geo-Fencing

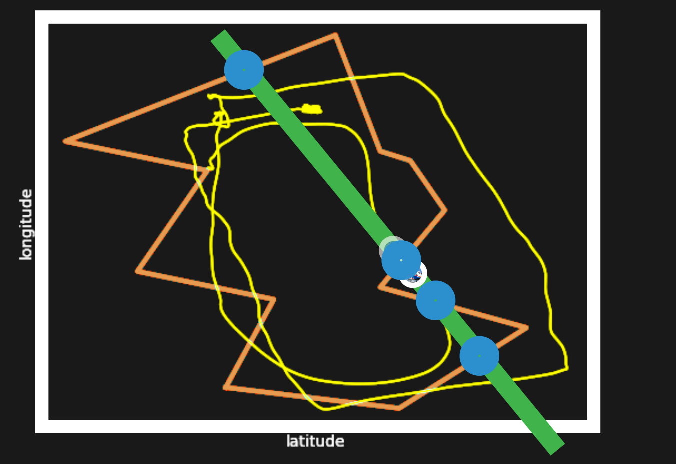
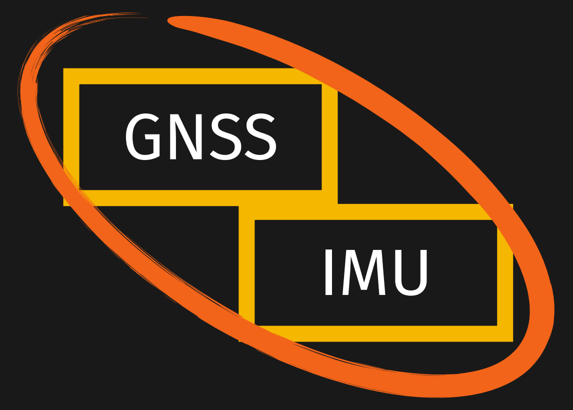


Block III Specs II: Geo-Fencing

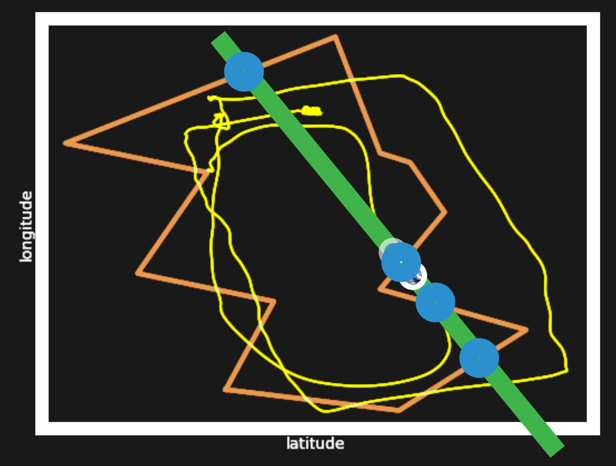


I. Arithmetically Challenging
II. Highly Parallel

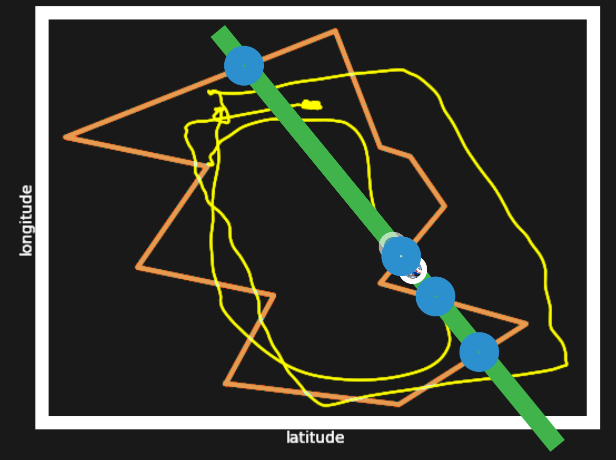
Block III Resource Consumption

	FF (%)	LUT	MUX	Pwr Idle [mW]	Pwr Peak [W]
	2,853 (3)	26,181 (71)	4	149	1.871
	4,792 (5)	34,630 (67)	104	156	2.085
GNSS IMU	3,441 (4)	23,261 (46)	99	150	1.911

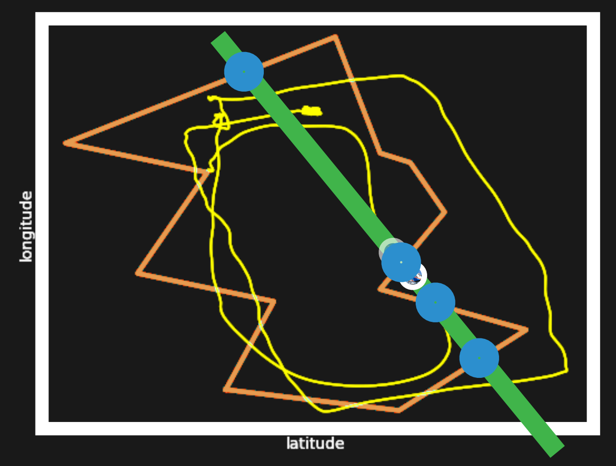
Block III Resource Consumption

	FF (%)	LUT	MUX	Pwr Idle [mW]	Pwr Peak [W]
	2,853 (3)	26,181 (71)	4	149	1.871
GNSS / IMU	4,792 (5)	34,630 (67)	104	156	2.085
GNSS / IMU	3,441 (4)	23,261 (46)	99	150	1.911

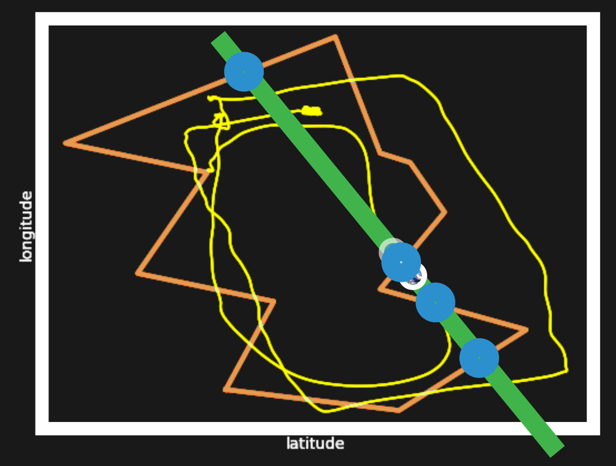
Block III Resource Consumption

	FF (%)	LUT	MUX	Pwr Idle [mW]	Pwr Peak [W]
	2,853 (3)	26,181 (71)	4	149	1.871
	4,792 (5)	34,630 (67)	104	156	2.085
	3,441 (4)	23,261 (46)	99	150	1.911

Block III Resource Consumption

	FF (%)	LUT	MUX	Pwr Idle [mW]	Pwr Peak [W]
	2,853 (3)	26,181 (71)	4	149	1.871
	4,792 (5)	34,630 (67)	104	156	2.085
GNSS IMU	3,441 (4)	23,261 (46)	99	150	1.911

Block III Resource Consumption

	FF (%)	LUT	MUX	Pwr Idle [mW]	Pwr Peak [W]
	2,853 (3)	26,181 (71)	4	149	1.871
	4,792 (5)	34,630 (67)	104	156	2.085
	3,441 (4)	23,261 (46)	99	150	1.911

Block III Post-Mortem Analysis

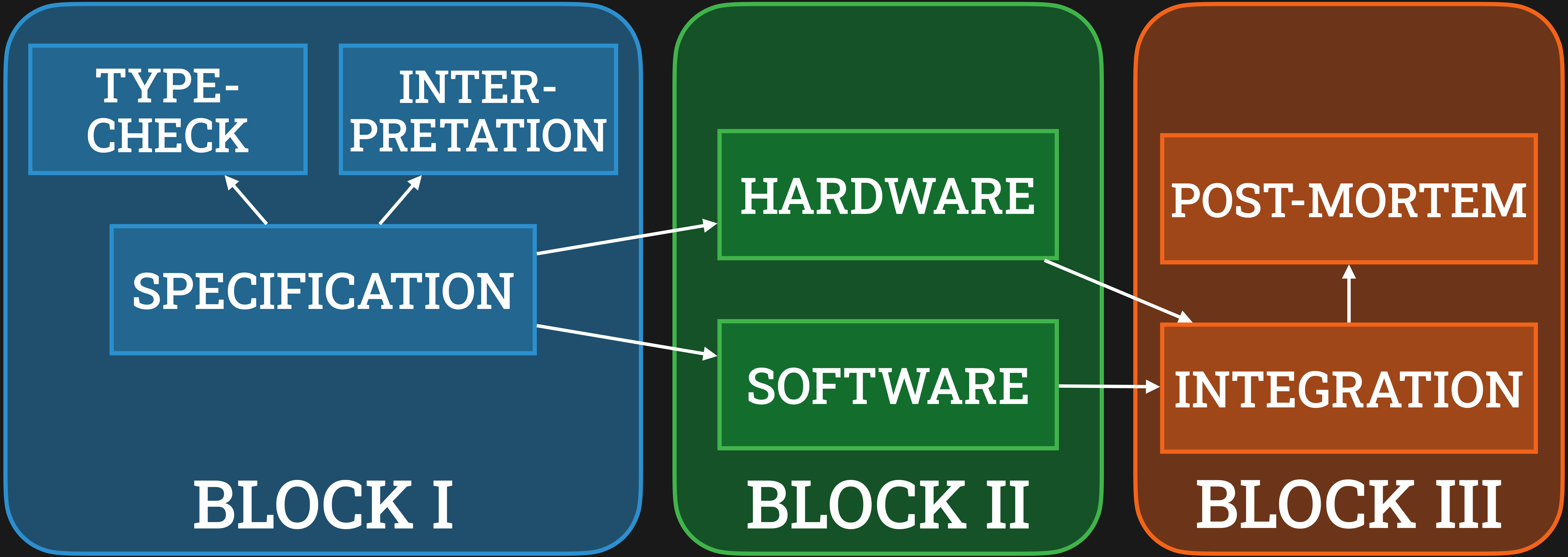
Block III Post-Mortem Analysis

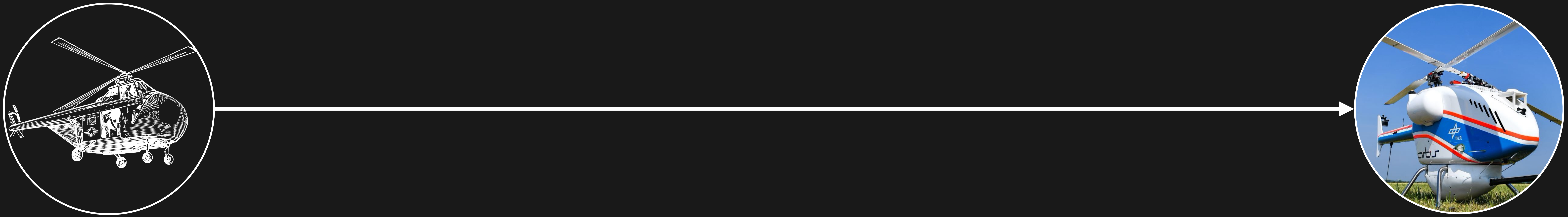
I. (Quantitative) Stream-based RV superior to Boolean Verdicts

II. Monitor naturally refines and Filters data.

III. Access to crucial data.

Block III Conclusion





Block III Conclusion

For a Monitor To Show its Full Potential, It Needs To Be Co-Developed With the CPS!