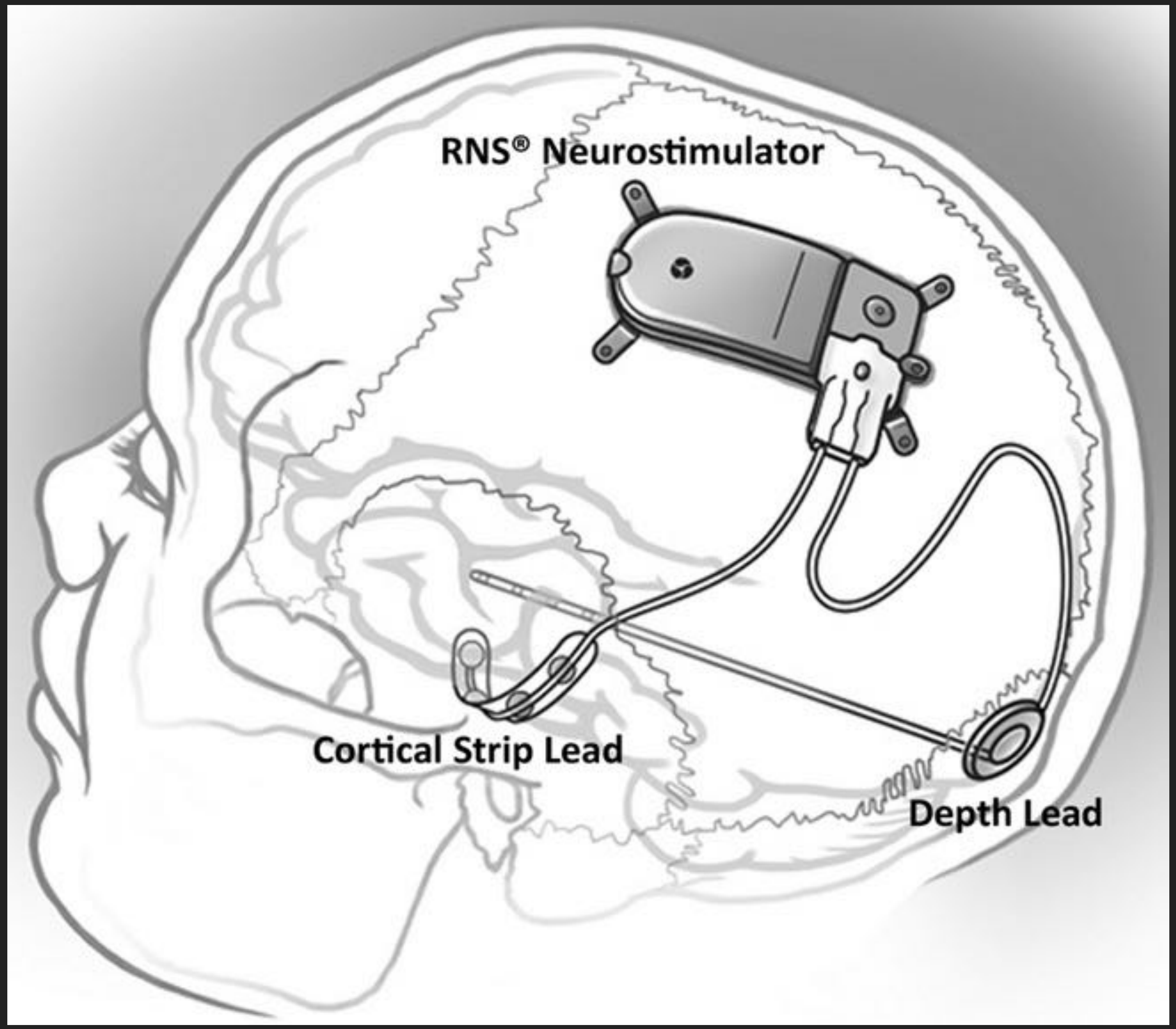
Let’s not Trust Experience Blindly: Formal Monitoring of Humans and other CPS

Maximilian Schwenger

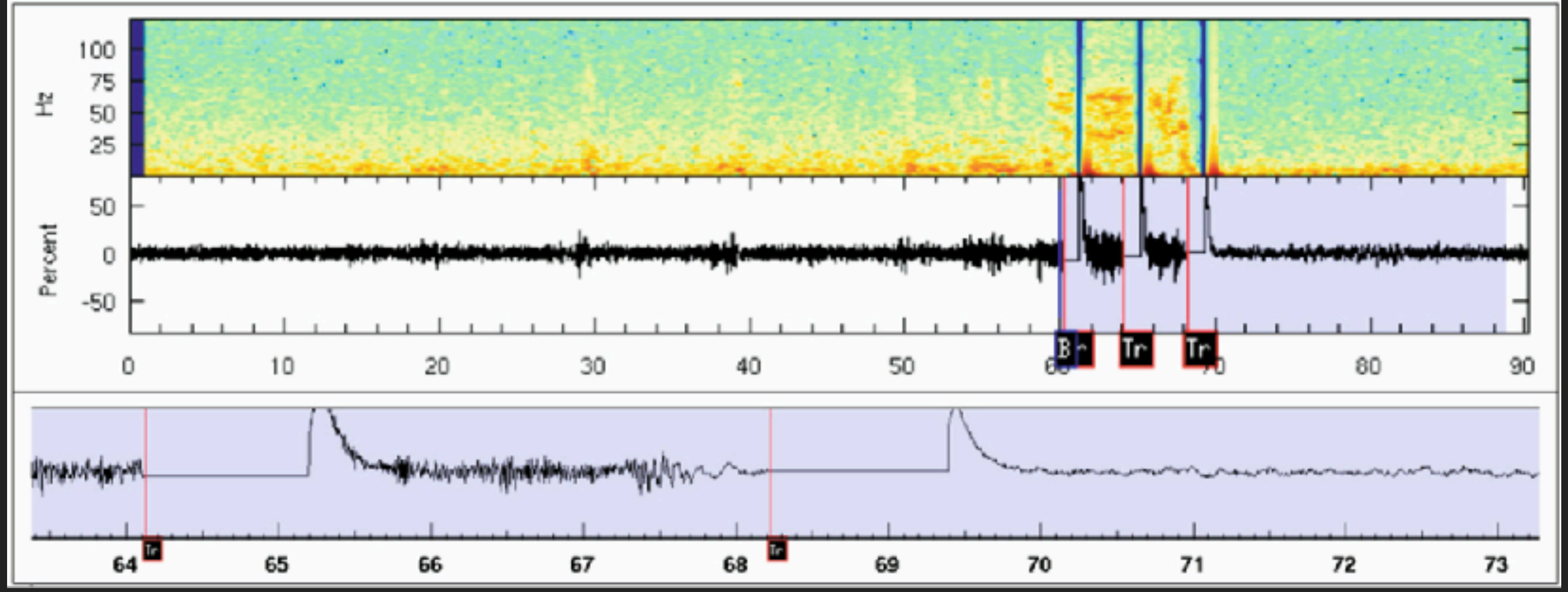
Saarland University

Responsive Neurostimulator



Heck et al., *"Two-year seizure reduction in adults with medically intractable partial onset epilepsy treated with responsive neurostimulation: Final results of the RNS System Pivotal trial"*, Epilepsia 2014

CORTICAL ELECTROENCEPHALOGRAM



Sun et al., *"Responsive Cortical Stimulation for the Treatment of Epilepsy"*, Neurotherapeutics 2008

Kossoff et al., *"Effect of an External Responsive Neurostimulator on Seizures and Electrographic Discharges during Subdural Electrode Monitoring"*, Epilepsia 2004

Responsive Neurostimulator

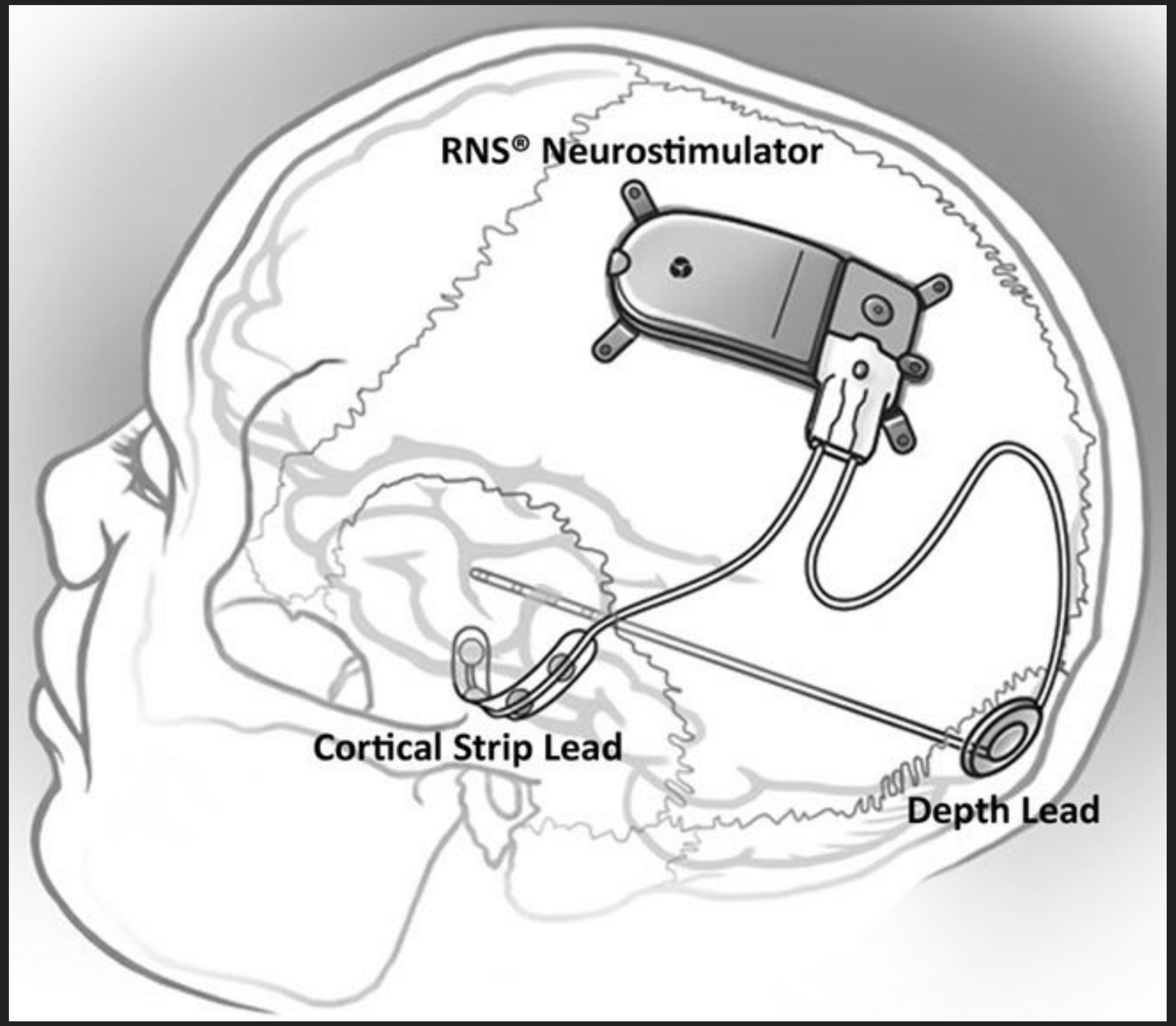
Cortical Electroencephalogram

Hz
100
75
50
25
Percent
50
0
-50
0
10
20
30
40
50
60
70
80
90
B
Tr
Tr
Tr
Tr
Tr
64
65
66
67
68
69
70
71
72
73

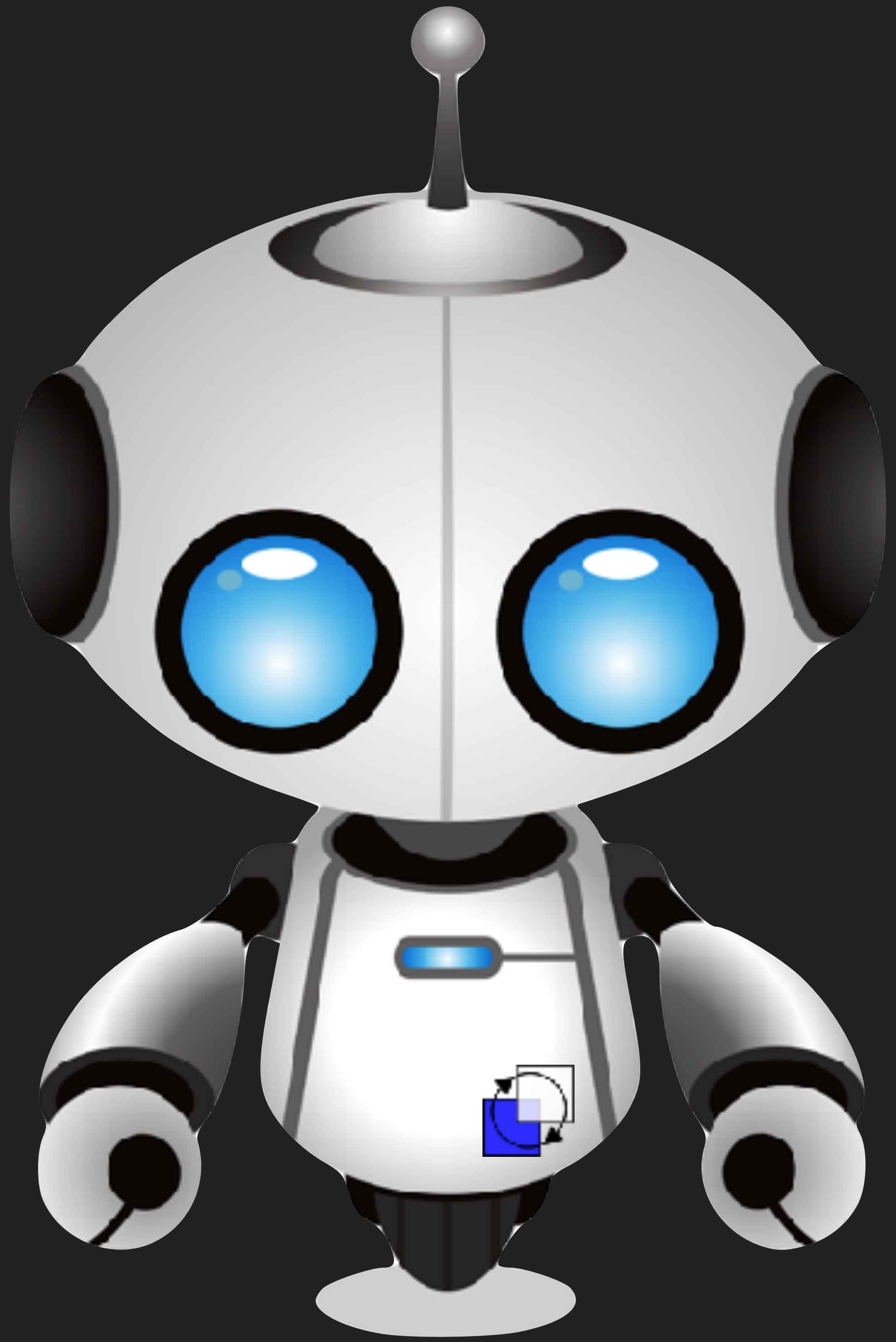
medically intractable partial onset epilepsy treated with responsive neurostimulation: Final results of the RNS System Pivotal trial", Epilepsia 2014

Kossoff et al., *"Effect of an External Responsive Neurostimulator on Seizures and Electrographic Discharges during Subdural Electrode Monitoring"*, Epilepsia 2004

Our Goal

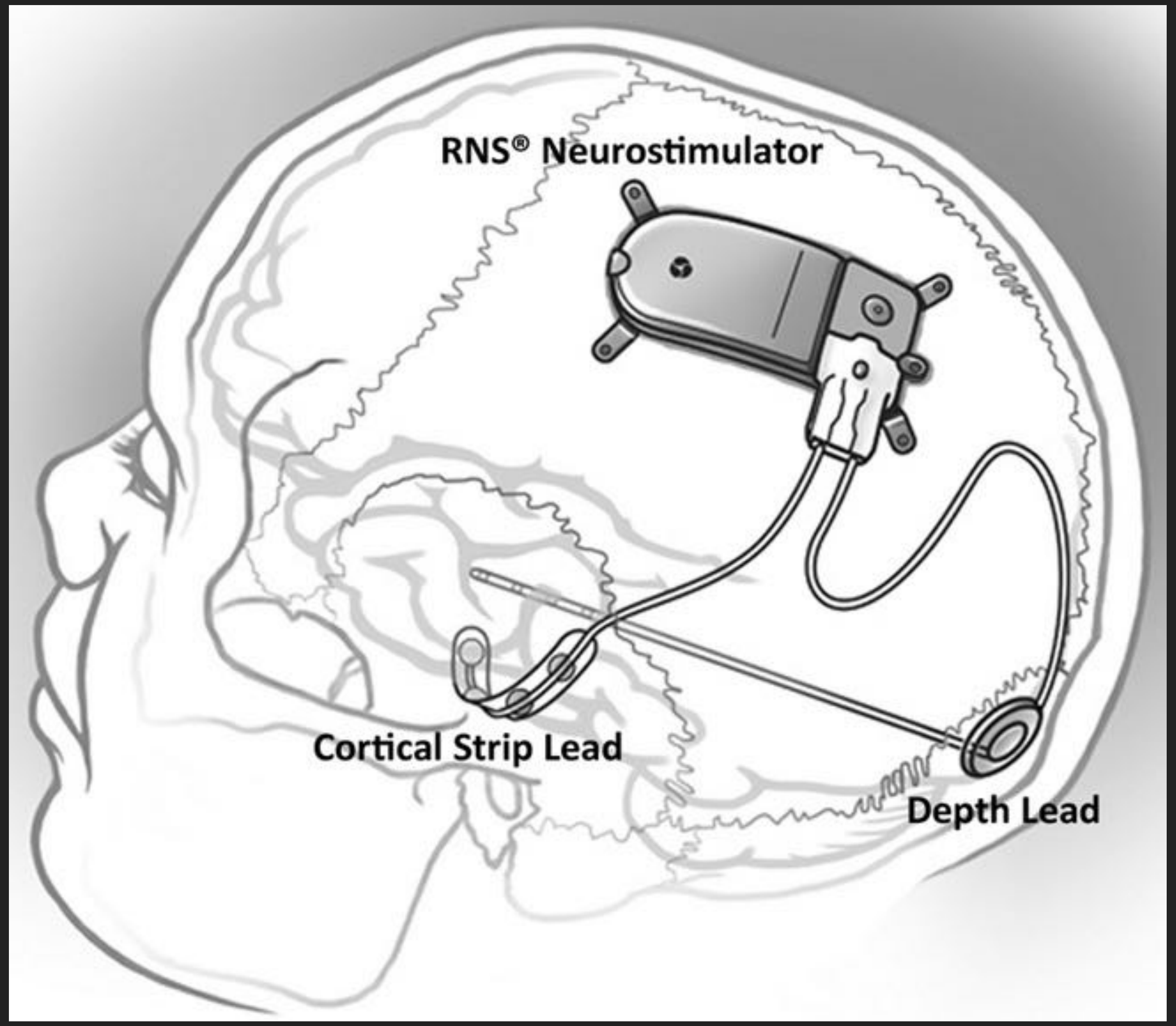


Trust me,
I'm a (medical) doctor

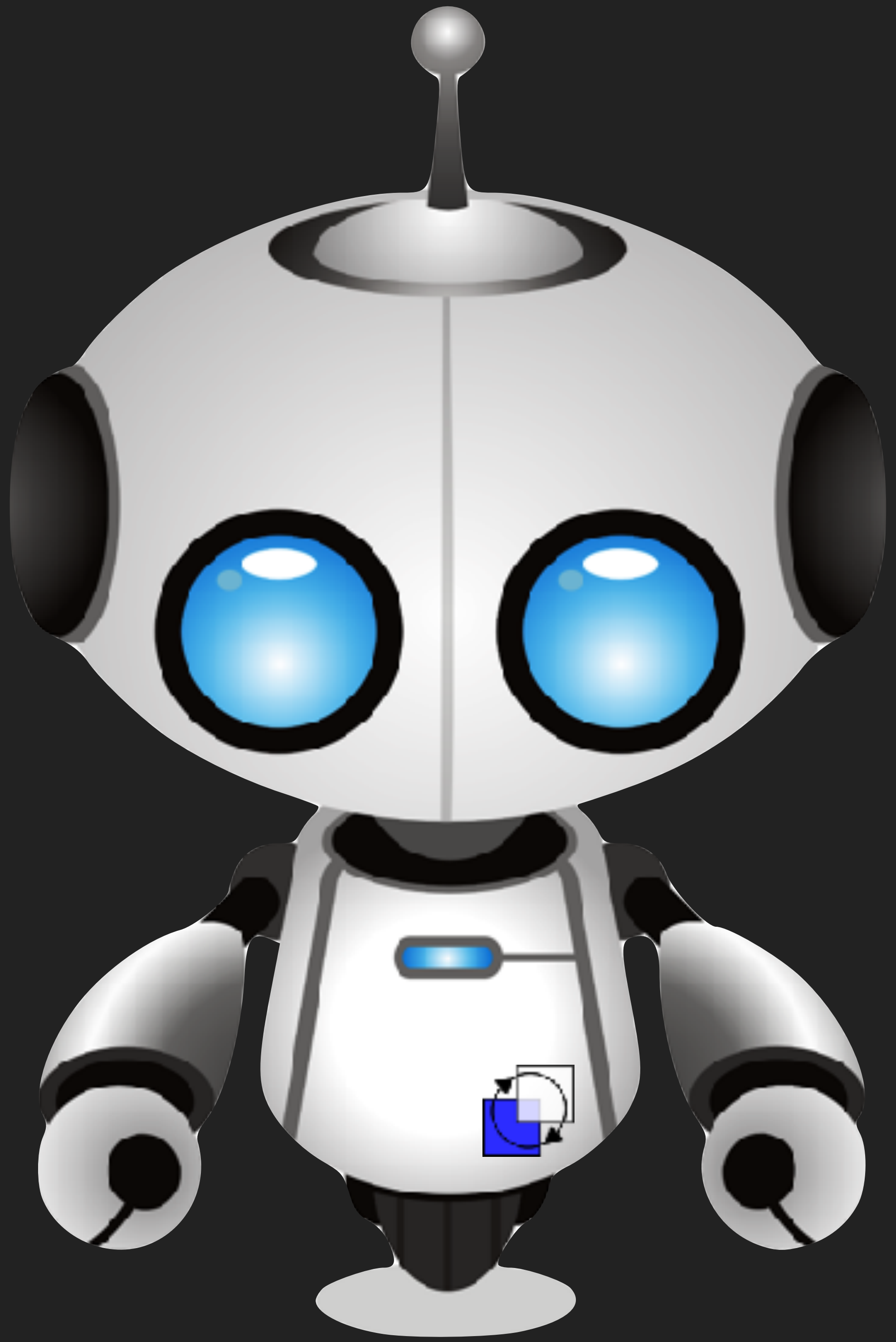


Trust me,
I'm an engineer

Our Goal



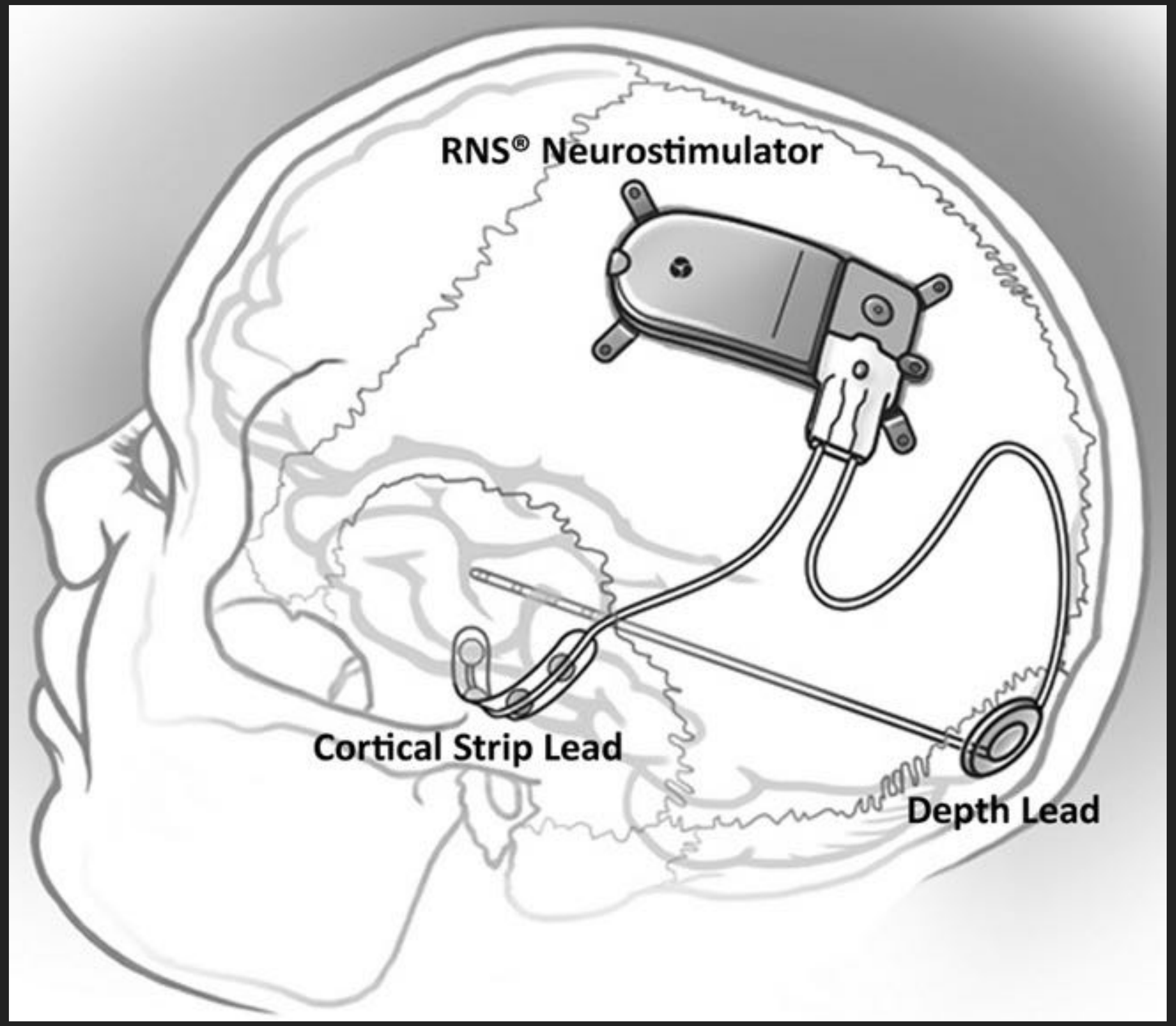
Trust me,
I'm a (medical) doctor



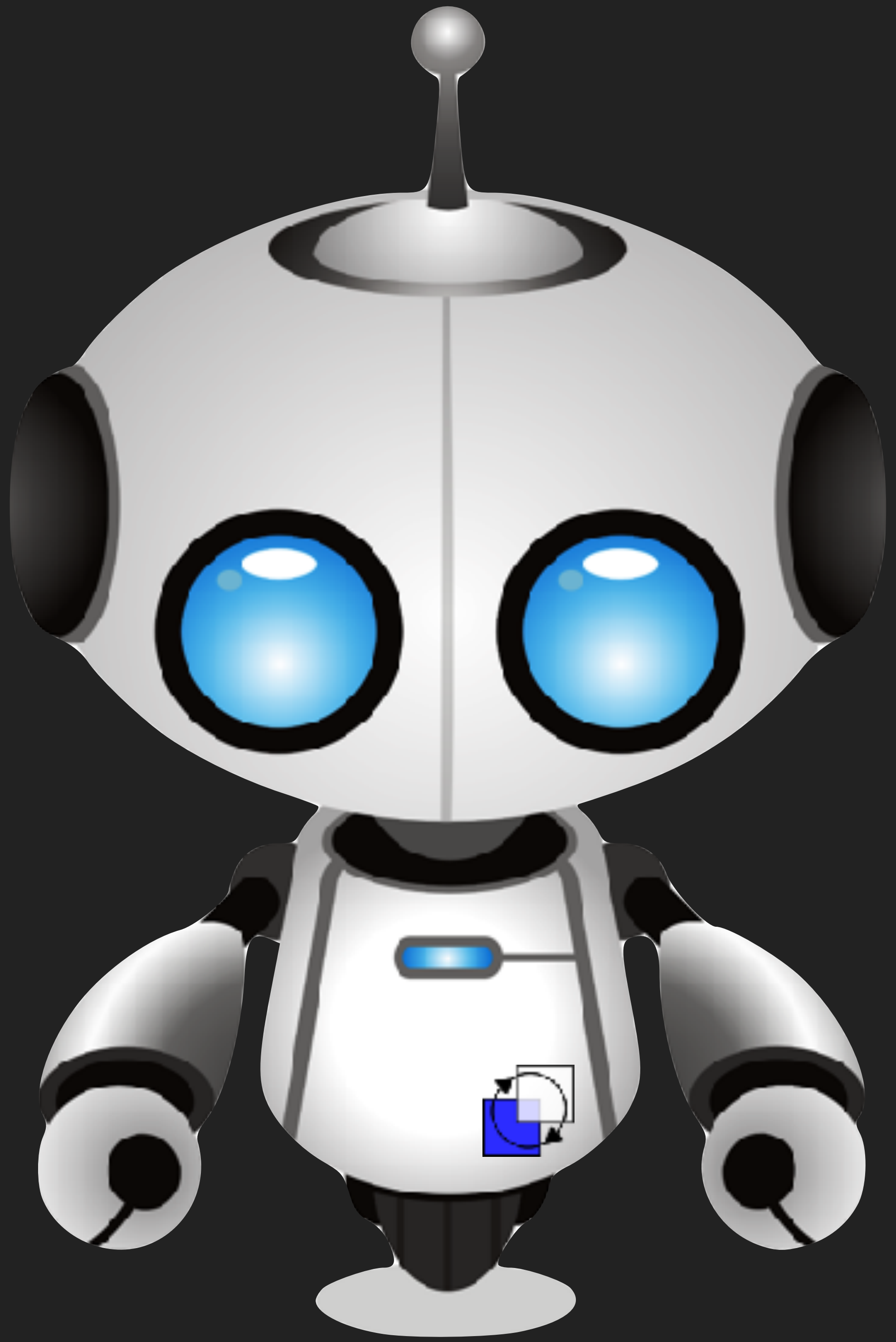
¯_(ツ)_/¯

It's ML

Our Goal



**Trust me,
I'm a (medical) doctor**



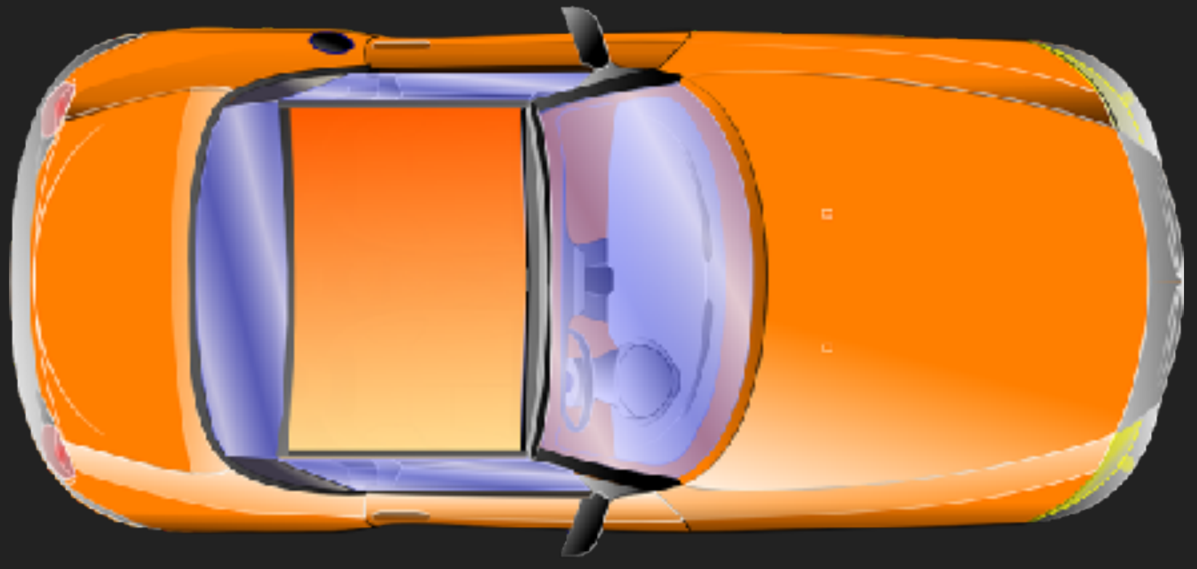
¯_(ツ)_/¯

It's ML

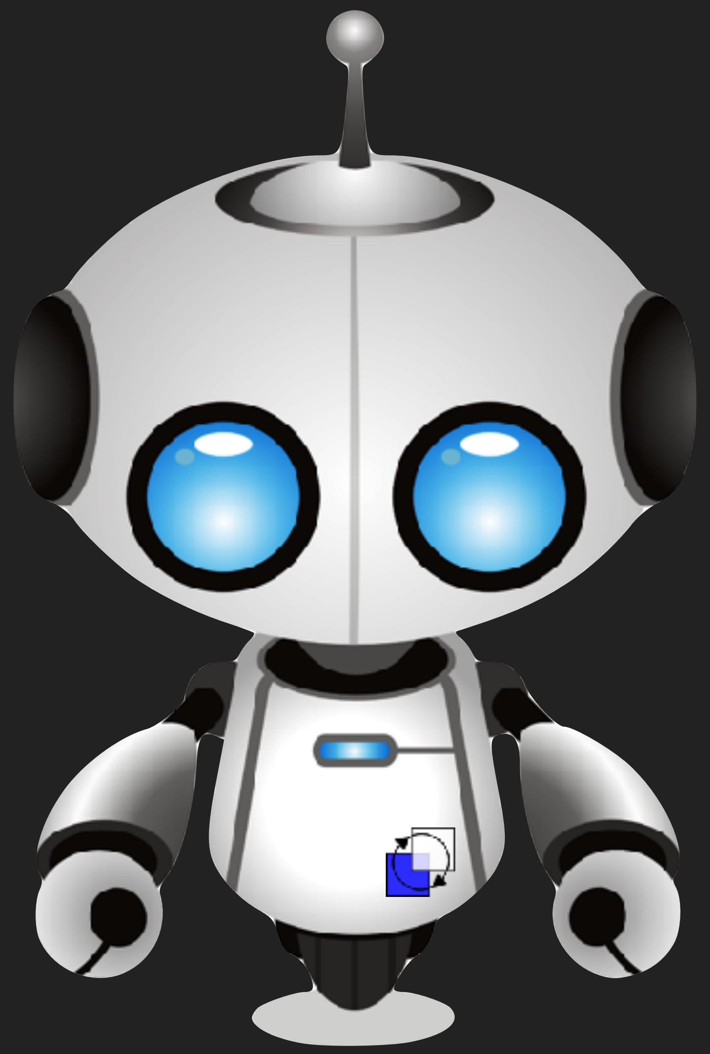
+

Formal Guarantees on Runtime Behavior

Static Verification



System S



Controller C

1. Never injure humans.
2. Obey orders.
3. Protect yourself

Specification φ

Verify:

$$\forall \sigma \in \mathit{runs}(S \parallel C) : \sigma \models \varphi$$

When Static Verification Fails

Complexity

$$\dot{p} = Rv$$

$$\dot{R} = R\hat{\omega}$$

$$\dot{v} = -\omega \times v + R^T \bar{g} + f_v(\omega, v, \alpha, \beta, \omega_r, \delta_c, \delta_r)$$

$$\dot{\omega} = -J^{-1}(\omega \times J\omega) + f_w(\omega, v, \alpha, \beta, \omega_r, \delta_c, \delta_r)$$

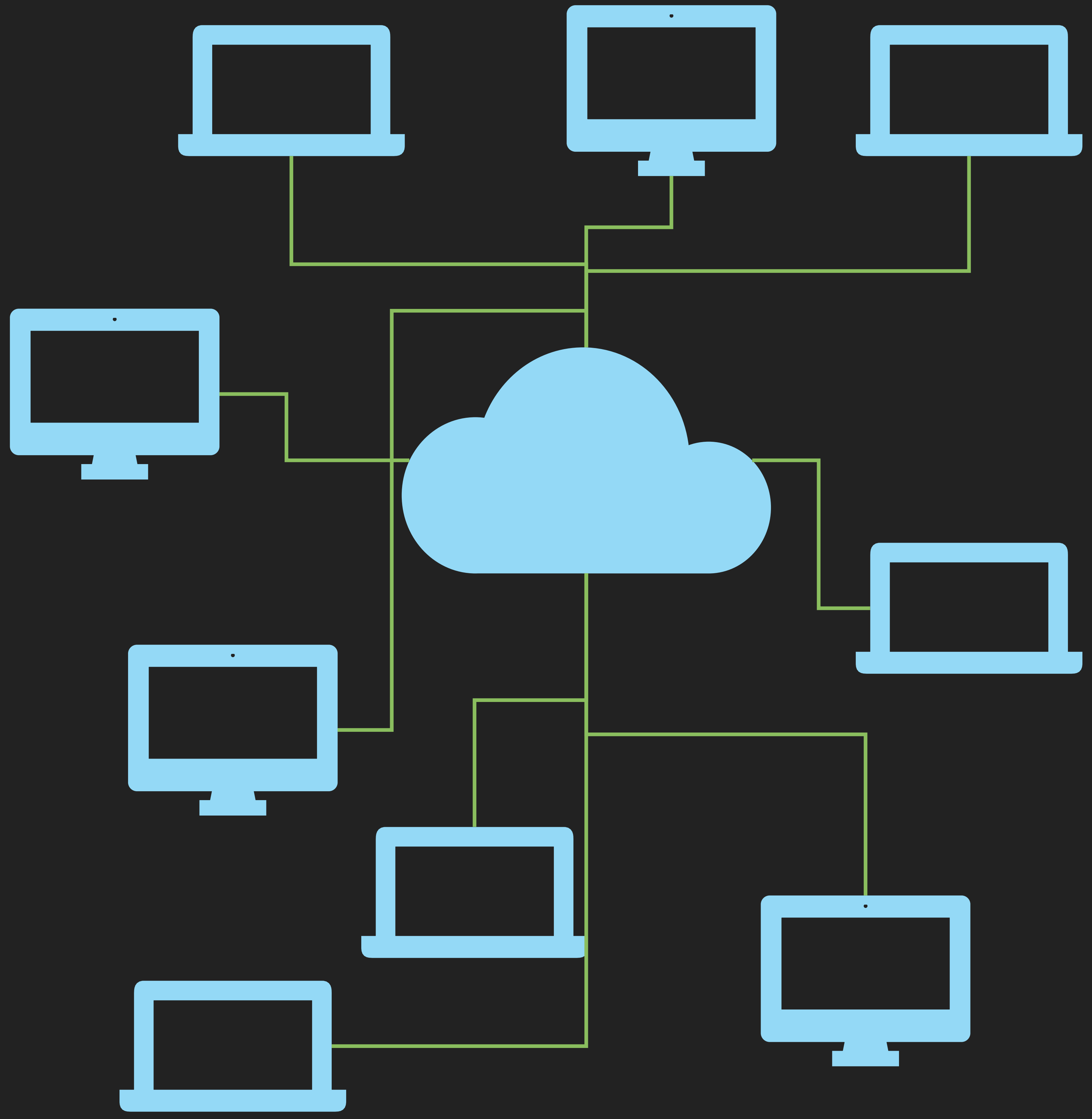
$$\dot{\alpha} = f_\alpha(\omega, v, \alpha, \beta, \omega_r, \delta_a, \delta_e)$$

$$\dot{\beta} = f_\beta(\omega, v, \alpha, \beta, \omega_r, \delta_a, \delta_e)$$

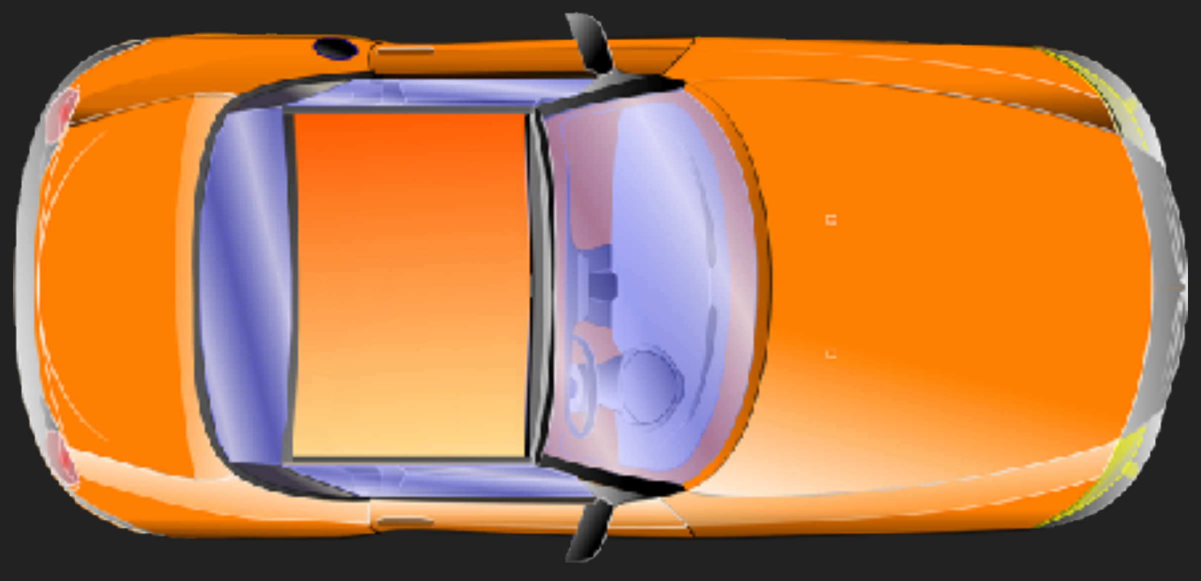
$$\dot{\omega}_r = f_r(\omega, v, \omega_r, \delta_c, \delta_r)$$

Lack Of Knowledge

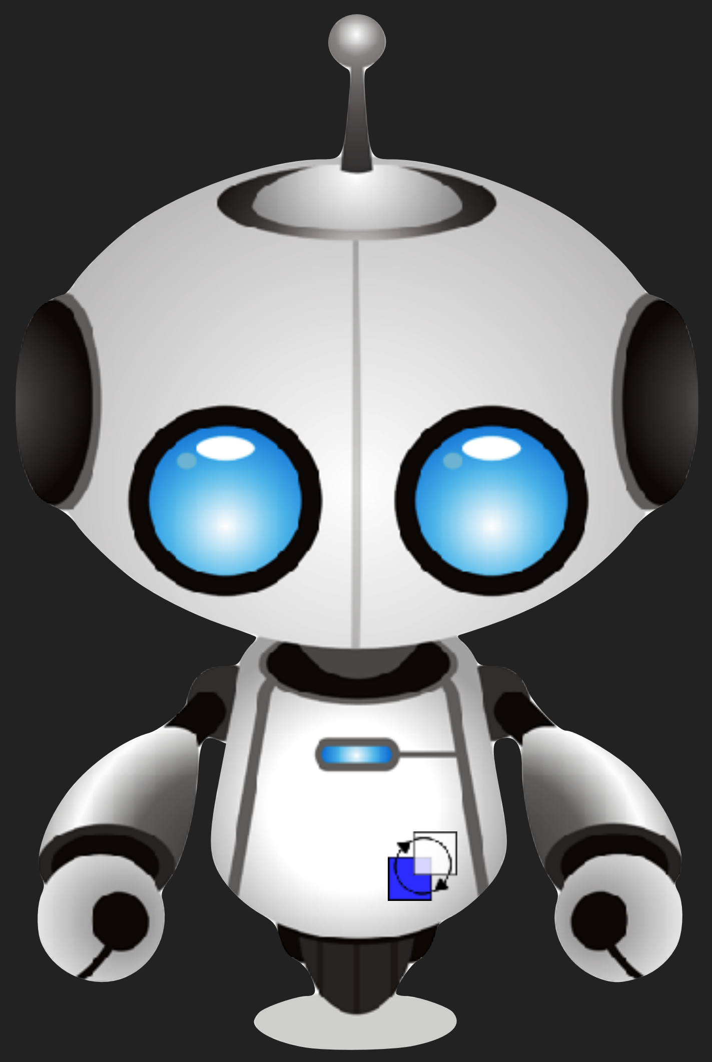
Non-Determinism



Static Verification



System S



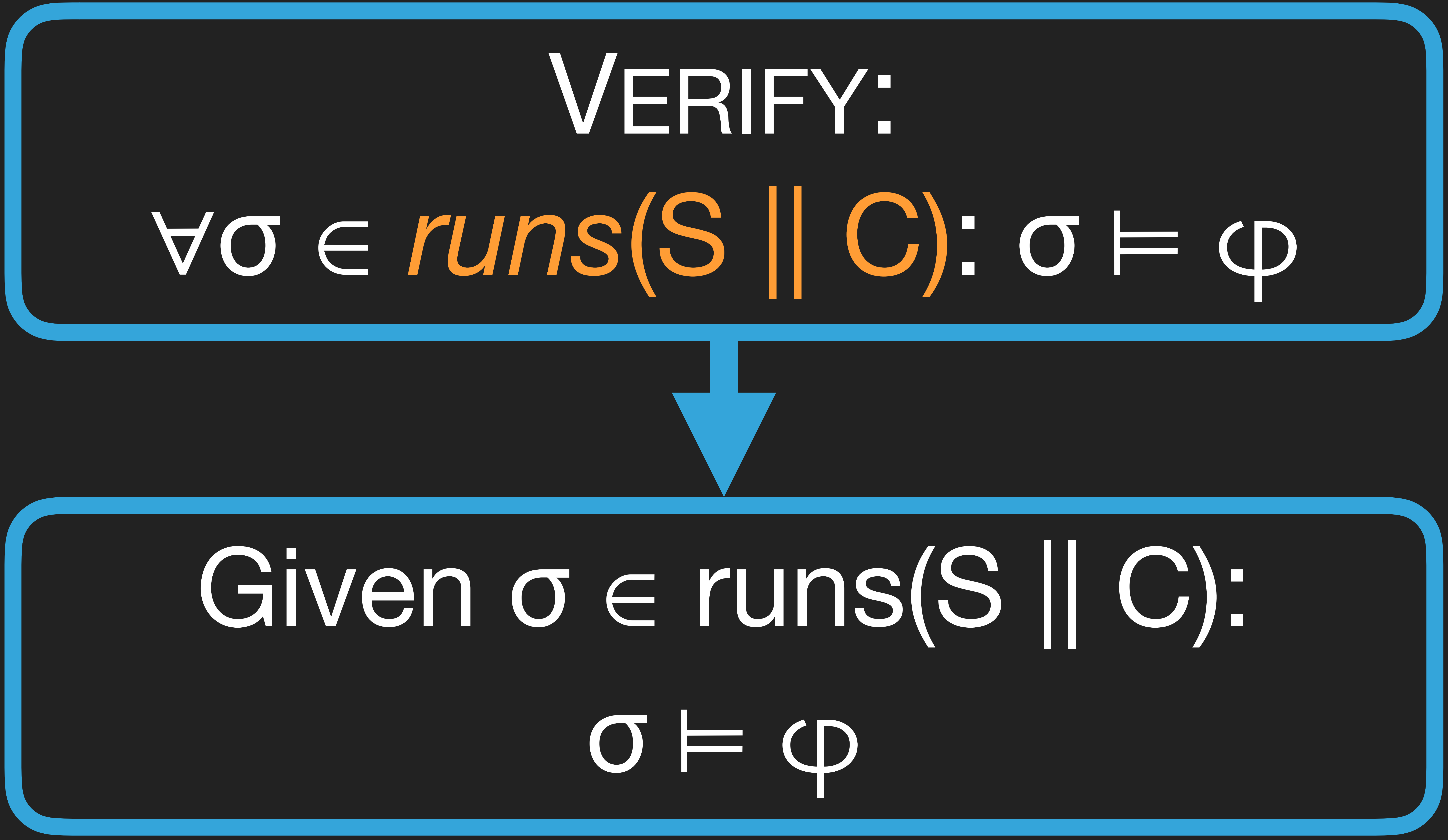
Controller C

Specification φ

Verify:

$$\forall \sigma \in \mathit{runs}(S \,\|\, C) : \sigma \models \varphi$$

Runtime Monitoring



Verify:

$\forall\sigma \in \mathit{runs}(S \mathbin{||} C) \colon \sigma \models \varphi$

Given $\sigma \in \mathrm{runs}(S \mathbin{||} C)$:

$\sigma \models \varphi$

Requirements

Embedded

Real-Time

Statistics

Fast

Formal

Through the Zoo of RV Approaches*

↯ RT ↯ St

Logics

RV-LTL [2]

rLTL [3]

LTL$_3$ [1]

↯ Fs ↯ Em

MTL + Aggr [5]

STL/MTL [4]

Data-Driven

SW Tie-Ins

[1] A. Bauer, M. Leucker, C. Schallhart. “*Runtime verification for LTL and TLTL*”. ACM Trans. Softw. Eng. Methodol. 2011

[2] A. Bauer, M. Leucker, C. Schallhart. “*The good, the bad, and the ugly, but how ugly is ugly*”, RV 2007

[3] C. Mascle, D. Neider, M. Schwenger, P. Tabuada, A. Weinert, M. Zimmermann, "*From LTL to rLTL Monitoring: Improved Monitorability through Robust Semantics*“, arxiv 2019

[4] O. Maler, D. Nickovic, “*Monitoring Temporal Properties of Continuous Signals*”, FORMATS 2004

[5] D. Basin F. Klaedtke, S. Marinovic, E. Zalinescu, “*Monitoring of temporal first-order properties with aggregations*”, FSMD 2015

* rather a tiny fraction thereof

Through the Zoo of RV Approaches*

↯RT ↯St

Logics

RV-LTL [2]

rLTL [3]

LTL$_3$ [1]

STL/MTL [4]

↯Fs ↯Em

MTL + Aggr [5]

Data-Driven

↯RT

Lustre [6]

(↯RT)

Striver [7]

↯RT ↯RT

Lola [9] -2.0 [10];

RT- [11]; FPGA- [12]

SW Tie-Ins

[6] P. Caspi, D. Pilaud, N. Halbwachs, J. Plaice, *"Lustre: A Declarative Language for Programming Synchronous Systems"*, POPL 1987

[7] F. Gorostiaga, C. Sánchez, *"Striver: Stream Runtime Verification for Real-Time Event-Streams"*, RV 2018

[9] B. D'Angelo, S. Sankaranarayanan, C Sánchez, W. Robinson, B. Finkbeiner, H. Sipma, S. Mehrotra, Z. Manna, *"LOLA: Runtime Monitoring of Synchronous Systems"*, TIME 2005

[10] P. Faymonville, B. Finkbeiner, S. Schirmer, H. Torfah, *"A Stream-Based Specification Language for Network Monitoring"*, RV 2016

[11] P. Faymonville, B. Finkbeiner, M. Schledjewski, M. Schwenger, M. Stenger, L. Tentrup, H. Torfah, *"StreamLAB: Stream-based Monitoring of Cyber-Physical Systems"*, CAV 2019

[12] J. Baumeister, B. Finkbeiner, M. Schwenger, H. Torfah, *"FPGA-based Monitoring of Real-time Properties"*, EMSOFT 2019

* rather a tiny fraction thereof

Through the Zoo of RV Approaches*

↯ RT ↯ St

Logics

RV-LTL [2]

rLTL [3]

LTL$_3$ [1]

STL/MTL [4]

↯ Fs ↯ Em

MTL + Aggr [5]

Data-Driven

↯ RT

Lustre [6]

(↯ RT)

Striver [7]

↯ RT ↯ RT

Lola [9] -2.0 [10];

RT- [11]; FPGA- [12]

JavaMOP [13]

Aspects [14]

DTrace [15]

↯ Em

SW Tie-Ins

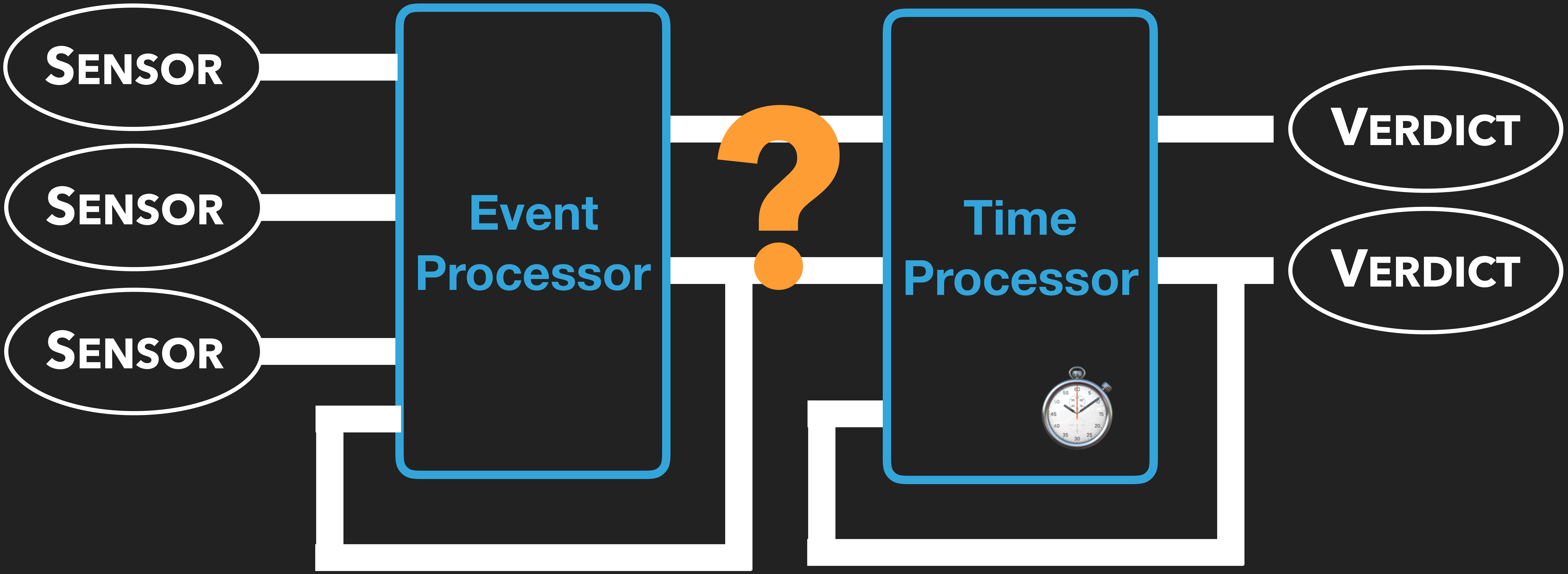
[13] F. Chen, G. Roşu, *"Java-MOP: A Monitoring Oriented Programming Environment for Java"*, TACAS 2005

[14] K. Havelund, E. Van Wyk, *"Aspect-Oriented Monitoring of C Programs"*

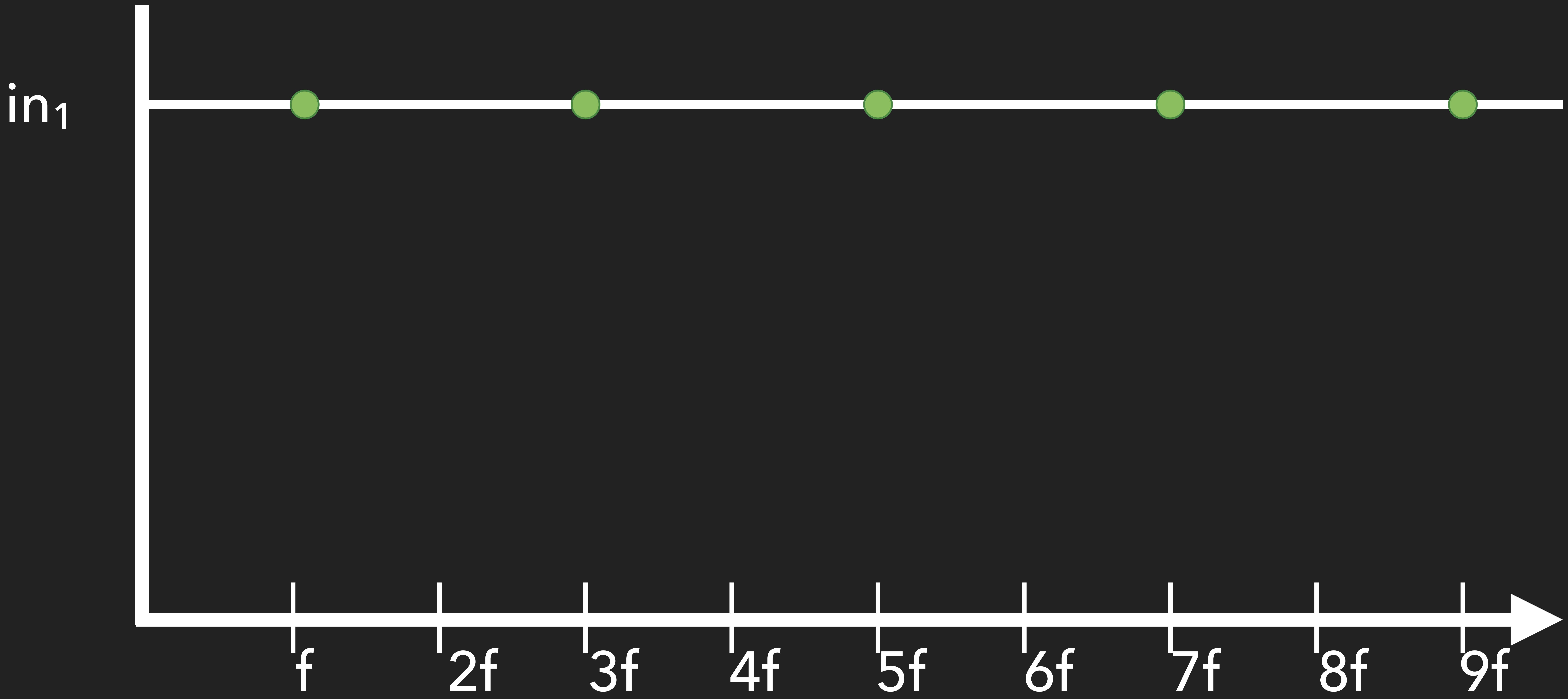
[15] C. Rosenberg, M. Steffen, V. Stolz, *"Leveraging DTrace for Runtime Verification"*, RV 2016

* rather a tiny fraction thereof

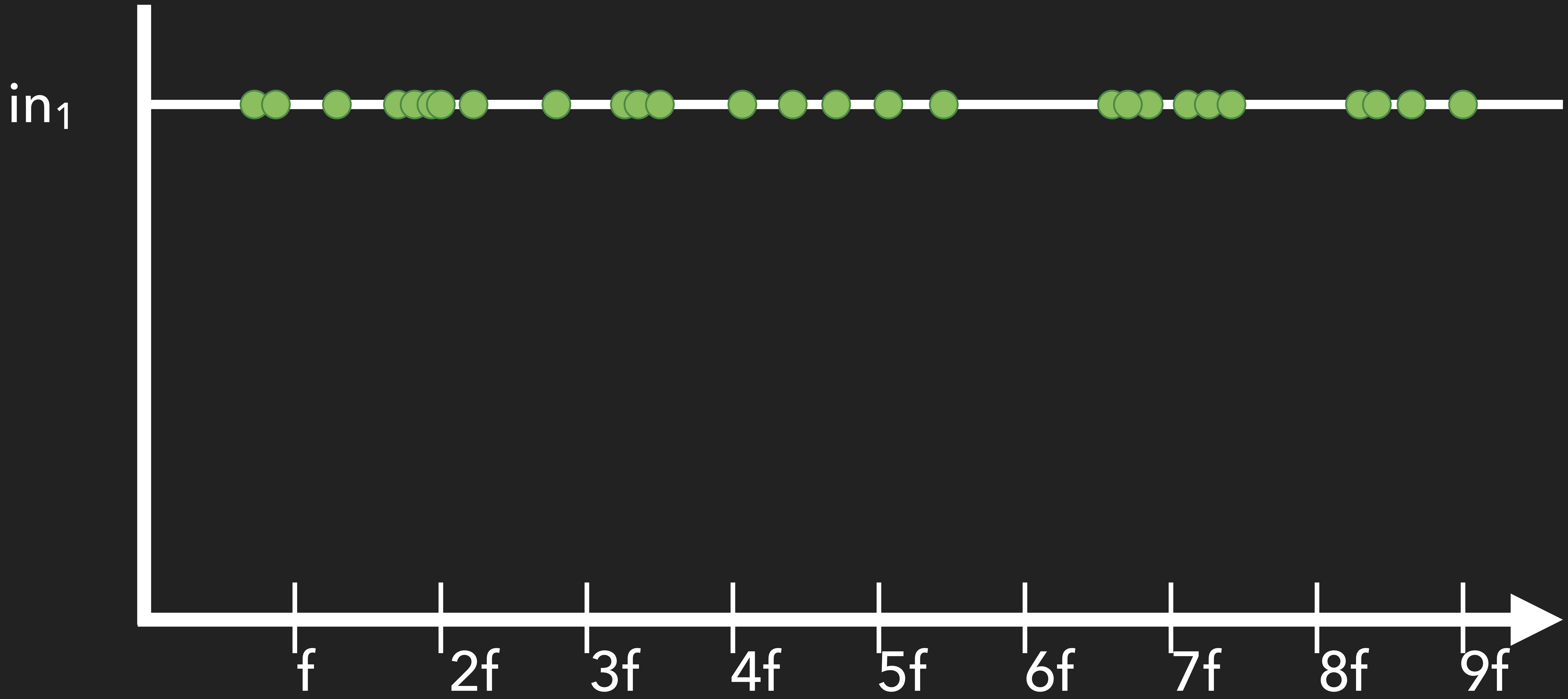
RTLola in a Nutshell



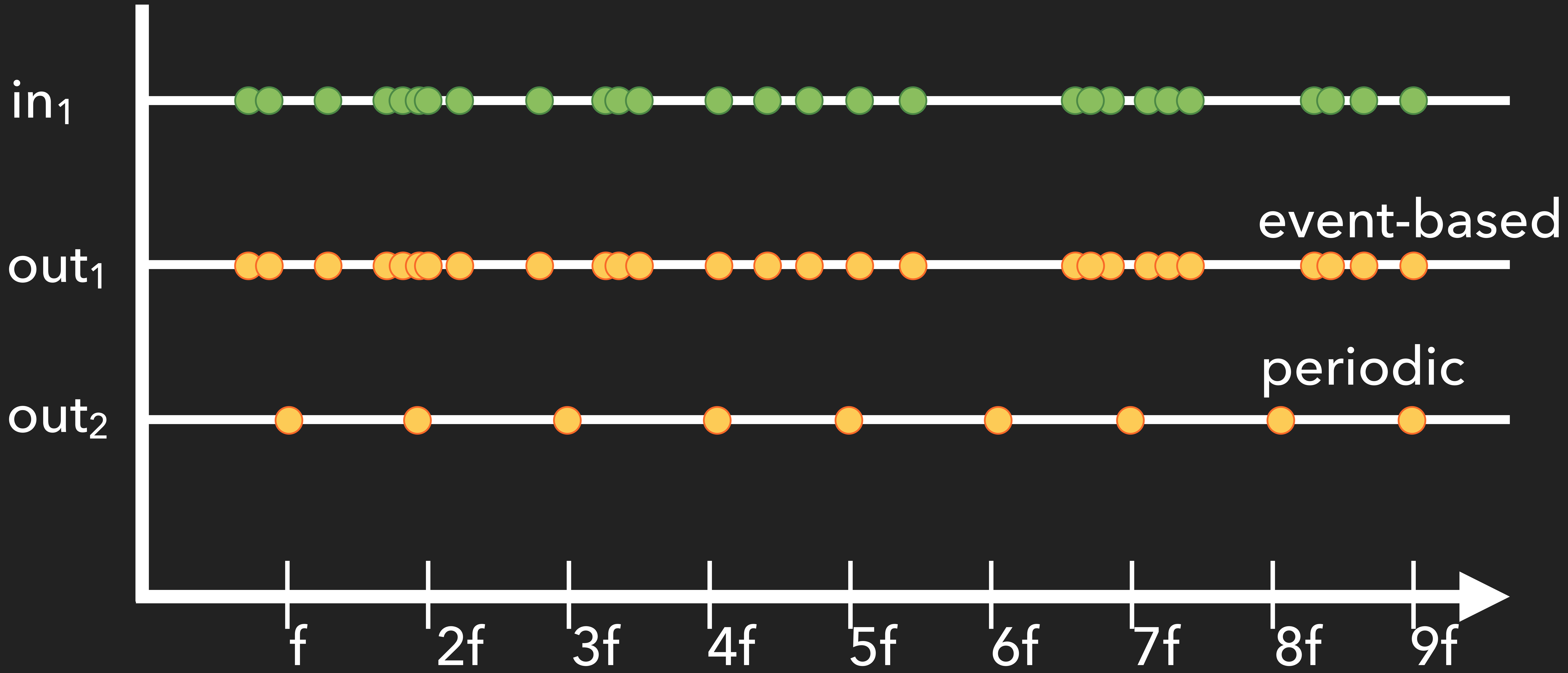
Asynchrony



Asynchrony



Asynchrony



RTLola: Syntax + Types

```
input a: Int32
input b: Float32
```

Int32	*{a}*
Float32	*{b}*

RTLola: Syntax + Types

`input a: Int32`	*Int32*	*{a}*
`input b: Float32`	*Float32*	*{b}*
`output c := a.offset(by: -1)`	*Int32*	*{a}*

RTLola: Syntax + Types

`input a: Int32`	*Int32*	*{a}*
`input b: Float32`	*Float32*	*{b}*
`output c := a.offset(by: -1)`	*Int32*	*{a}*
`output d := a + b`	*Int32* $\sqcap$ *Float32* = *Float32*	*{a,b}*

RTLola: Syntax + Types

`input a: Int32`	*Int32*	*{a}*
`input b: Float32`	*Float32*	*{b}*
`output c := a.offset(by: -1)`	*Int32*	*{a}*
`output d := a + b`	*Int32 ⊓ Float32 = Float32*	*{a,b}*
`output e        := a.hold()`	*Int32*	**?**

RTLola: Syntax + Types

```
input a: Int32
input b: Float32
output c := a.offset(by: -1)
output d := a + b
output e @3Hz := a.hold()
```

Int32	*{a}*
Float32	*{b}*
Int32	*{a}*
Int32 ⊓ Float32 = Float32	*{a,b}*
Int32	*3Hz*

RTLola: Syntax + Types

`input a: Int32`	Int32	{a}
`input b: Float32`	Float32	{b}
`output c := a.offset(by: -1)`	Int32	{a}
`output d := a + b`	Int32 ⊓ Float32 = Float32	{a,b}
`output e @3Hz := a.hold()`	Int32	3Hz
`output f := a = b`	ϟ ϟ ϟ	{a,b}

RTLola: Syntax + Types

`input a: Int32`	*Int32*	*{a}*
`input b: Float32`	*Float32*	*{b}*
`output c := a.offset(by: -1)`	*Int32*	*{a}*
`output d := a + b`	*Int32 ⊓ Float32 = Float32*	*{a,b}*
`output e @3Hz := a.hold()`	*Int32*	*3Hz*
`output f := a = b`	↯ ↯ ↯	*{a,b}*
`output g := e + a`	*Int32*	↯ ↯

RTLola: Syntax + Types

`input a: Int32`	*Int32*	*{a}*
`input b: Float32`	*Float32*	*{b}*
`output c := a.offset(by: -1)`	*Int32*	*{a}*
`output d := a + b`	*Int32 ⊓ Float32 = Float32*	*{a,b}*
`output e @3Hz := a.hold()`	*Int32*	*3Hz*
`output f := a = b`	ϟ ϟ ϟ	*{a,b}*
`output g := e + a`	*Int32*	ϟ ϟ
`output h        := a.aggr(over: 3s, using: γ)`		

$\text{Int32}^{*} \rightarrow \text{T}$

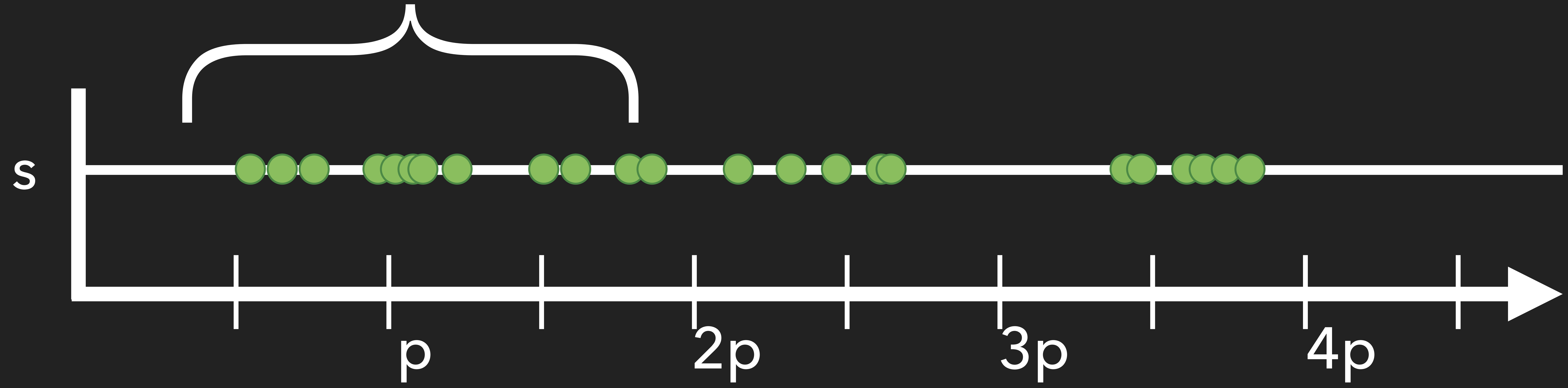
RTLola: Syntax + Types

`input a: Int32`	*Int32*	*{a}*
`input b: Float32`	*Float32*	*{b}*
`output c := a.offset(by: -1)`	*Int32*	*{a}*
`output d := a + b`	*Int32 ⊓ Float32 = Float32*	*{a,b}*
`output e @3Hz := a.hold()`	*Int32*	*3Hz*
`output f := a = b`	ϟ ϟ ϟ	*{a,b}*
`output g := e + a`	*Int32*	ϟ ϟ
`output h @2Hz := a.aggr(over: 3s, using: γ)`	*T*	*2Hz*

γ: Int32* → T

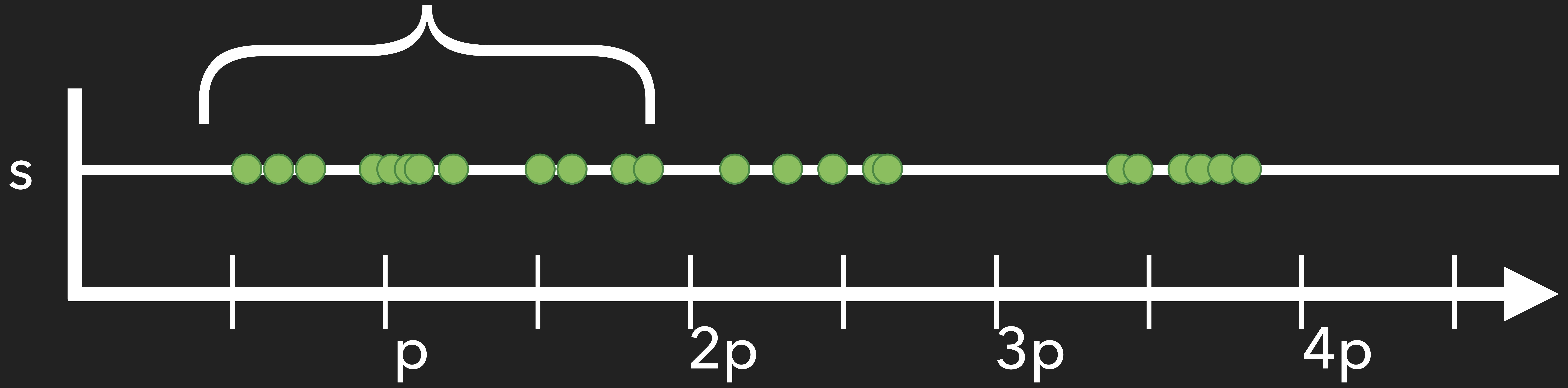
Sliding Windows

```
output h          := s.aggr(over: 1.5p, using: γ)
```



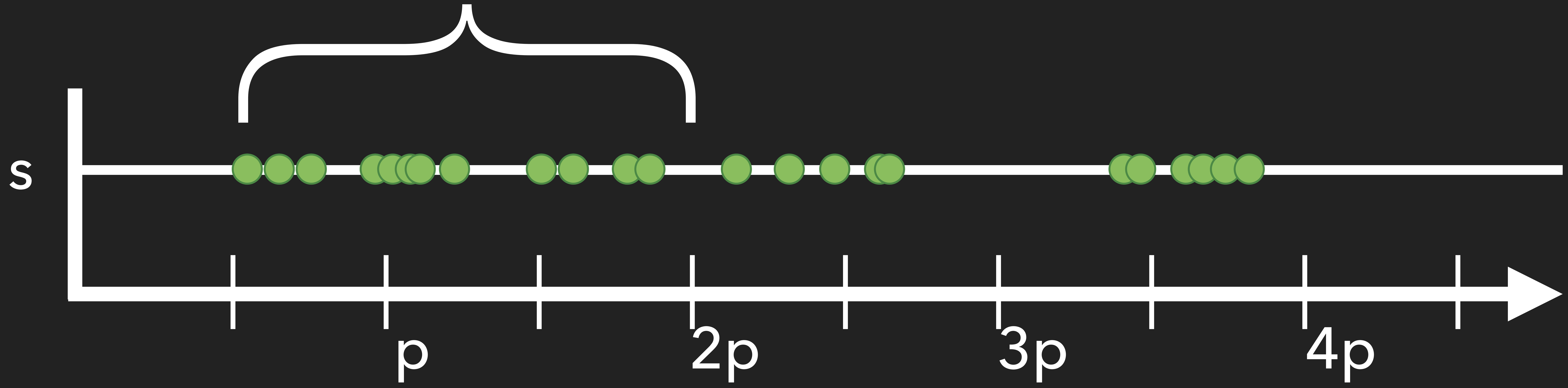
Sliding Windows

```
output h          := s.aggr(over: 1.5p, using: γ)
```



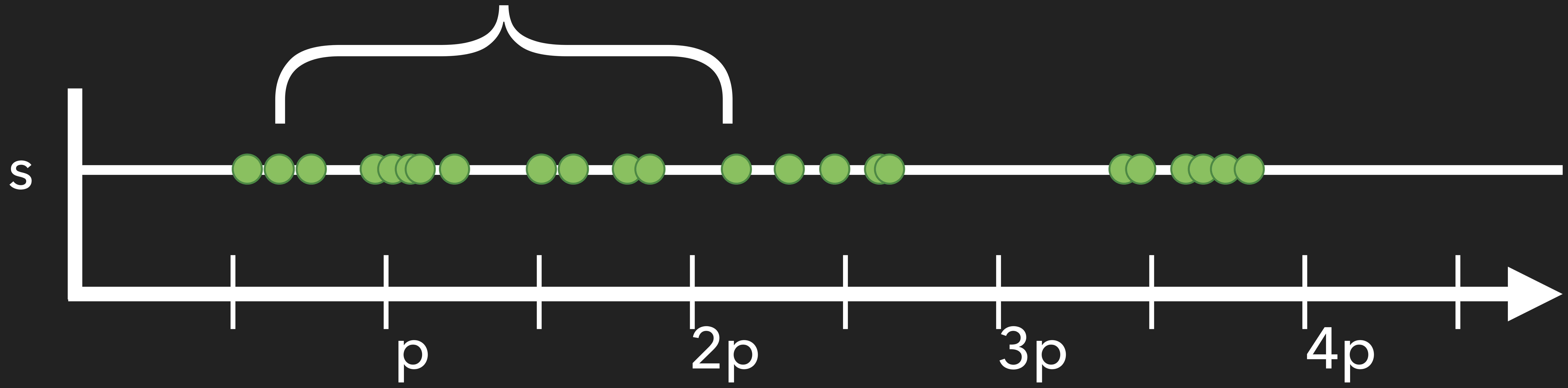
Sliding Windows

```
output h           := s.aggr(over: 1.5p, using: γ)
```



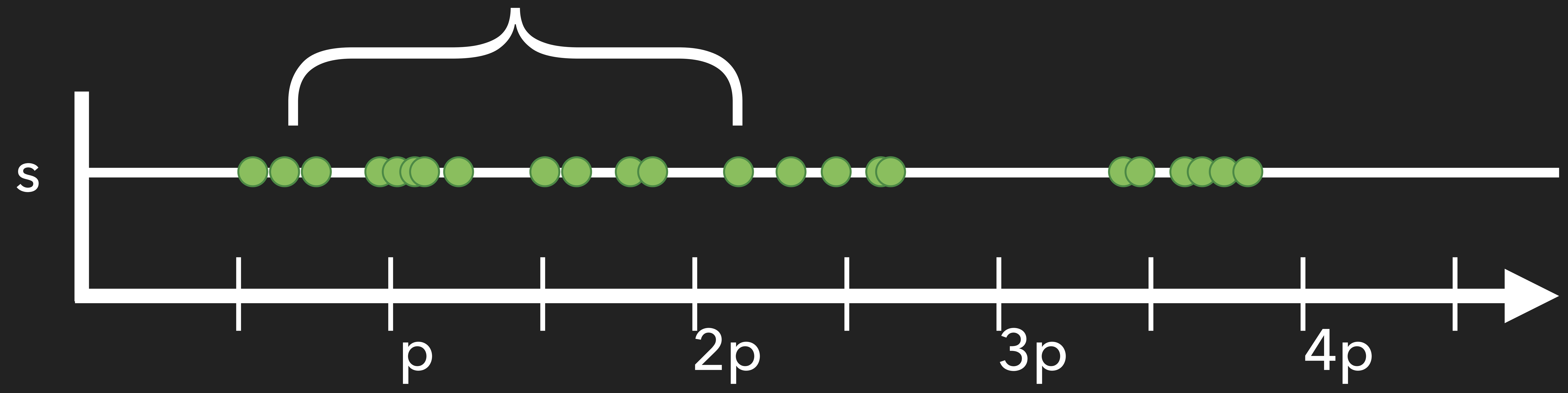
Sliding Windows

```
output h          := s.aggr(over: 1.5p, using: γ)
```



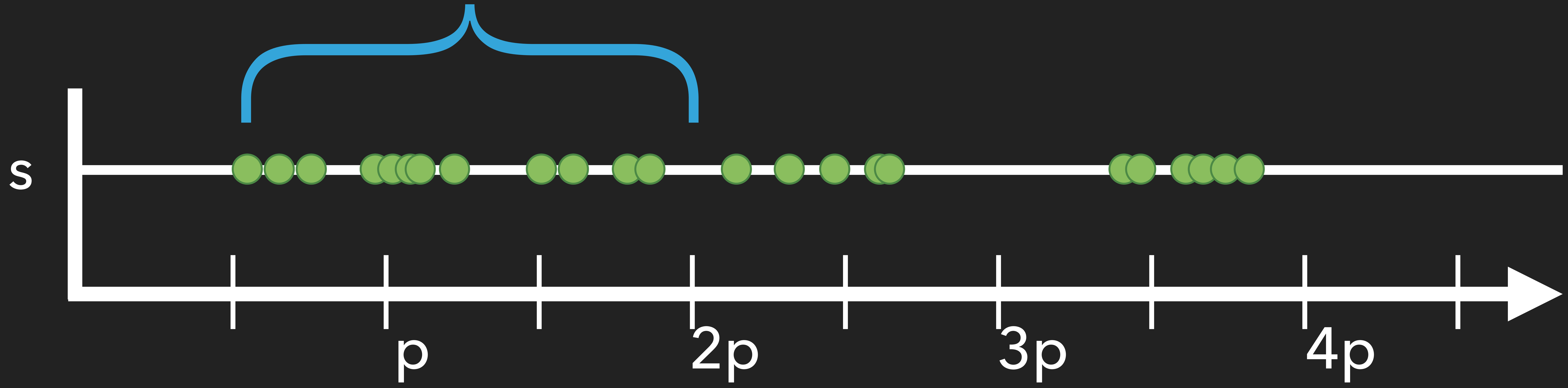
Sliding Windows

```
output h          := s.aggr(over: 1.5p, using: γ)
```



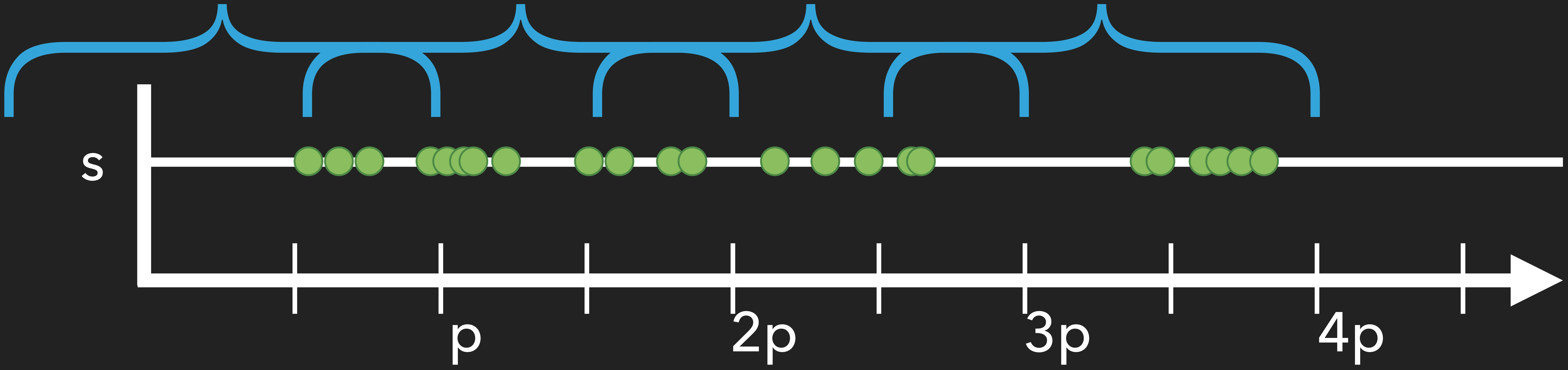
Sliding Windows

```
output h @p⁻¹Hz := s.aggr(over: 1.5p, using: γ)
```



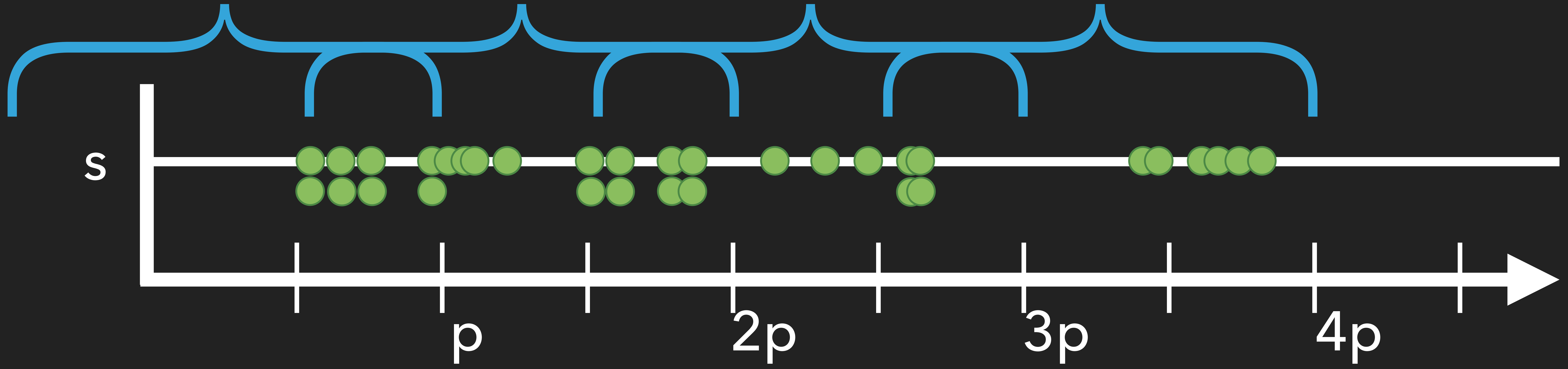
Sliding Windows

```
output h @p⁻¹Hz := s.aggr(over: 1.5p, using: γ)
```



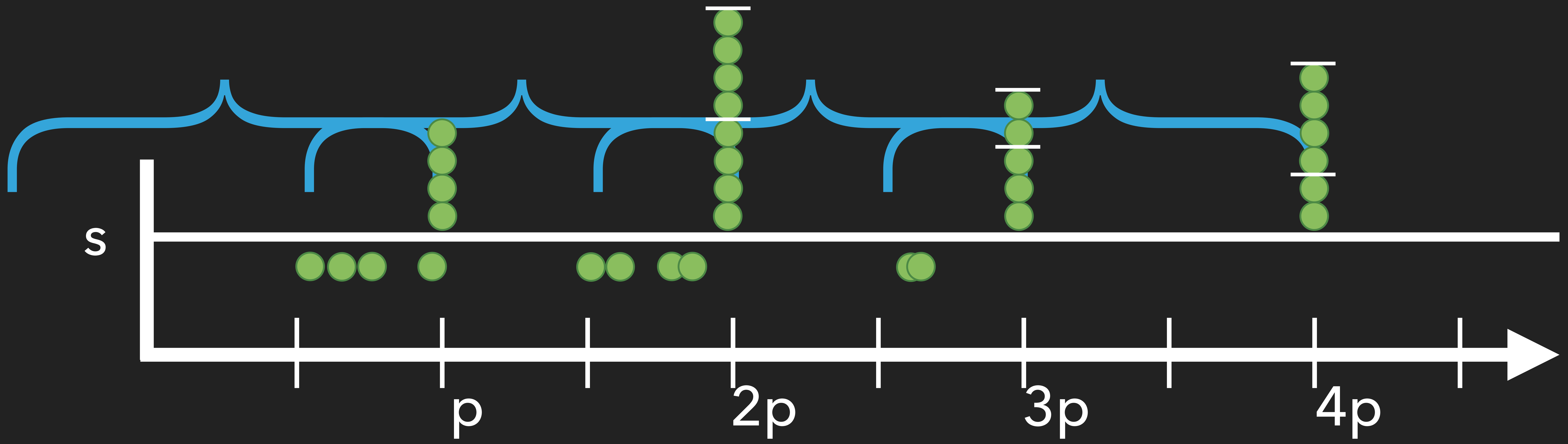
Sliding Windows

```
output h @p⁻¹Hz := s.aggr(over: 1.5p, using: γ)
```



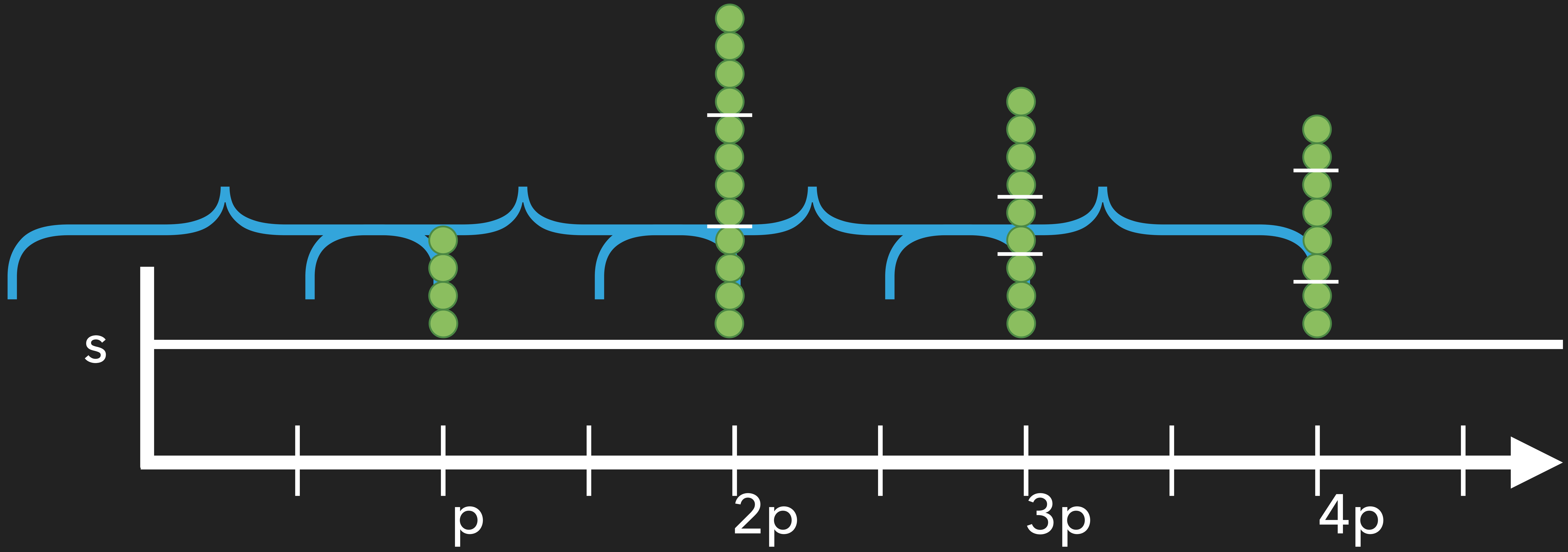
Sliding Windows

```
output h @p⁻¹Hz := s.aggr(over: 1.5p, using: γ)
```



Sliding Windows

```
output h @p⁻¹Hz := s.aggr(over: 1.5p, using: γ)
```



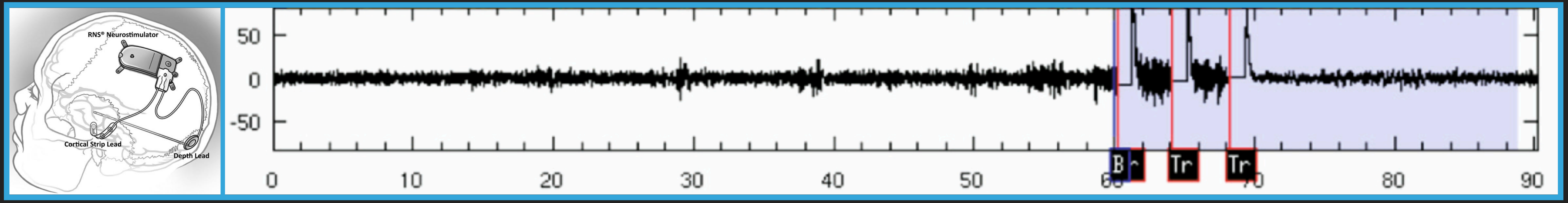
Practicality

```
input CSL, DL: Float64
input rec, stim: Bool

output twitch := abs(derive(3,CLS))

output avg_long  @100mHz := twitch.aggr(over: 2000s, using: avg)
output avg_short @1kHz := twitch.aggr(over: 2ms, using: avg)
output spike     @1kHz := avg_short > avg_long.hold() + ε

trigger spike ∧ ¬rec.aggr(over: 2ms, using: any)
                                “seizure not recognized”
trigger @1kHz rec.aggr(any, 5ms) ∧ ¬stim.aggr(any, 3ms)
                                “stimulation not triggered”
```

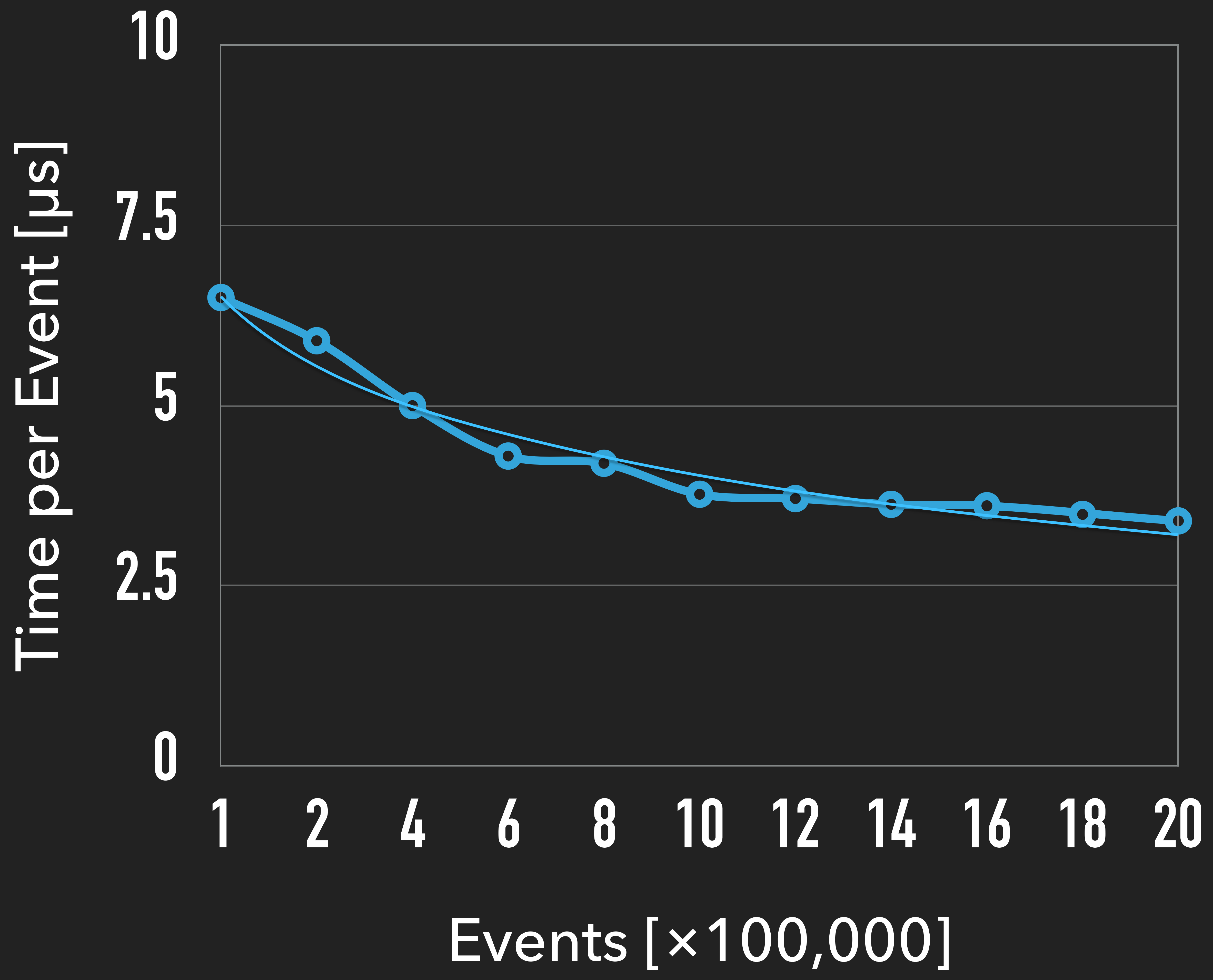


Requirements

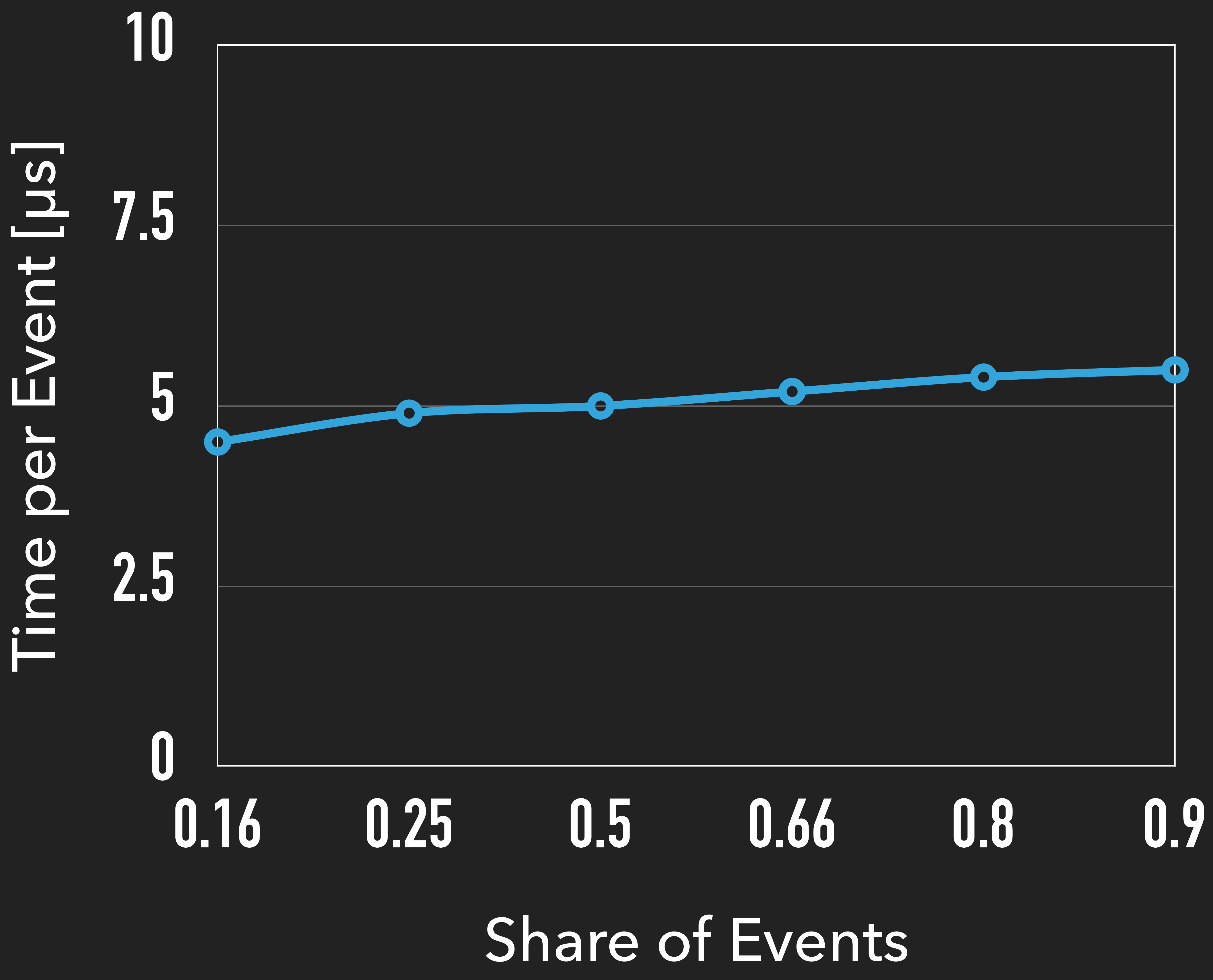
- Embedded ✓
- Real-Time ✓
- Statistics ✓
- Fast
- Formal ✓

Running Time

50% EV, 50% P



200K EV+P



Huge thanks to Leander, Marvin, and Malte!

Requirements

Embedded ✓

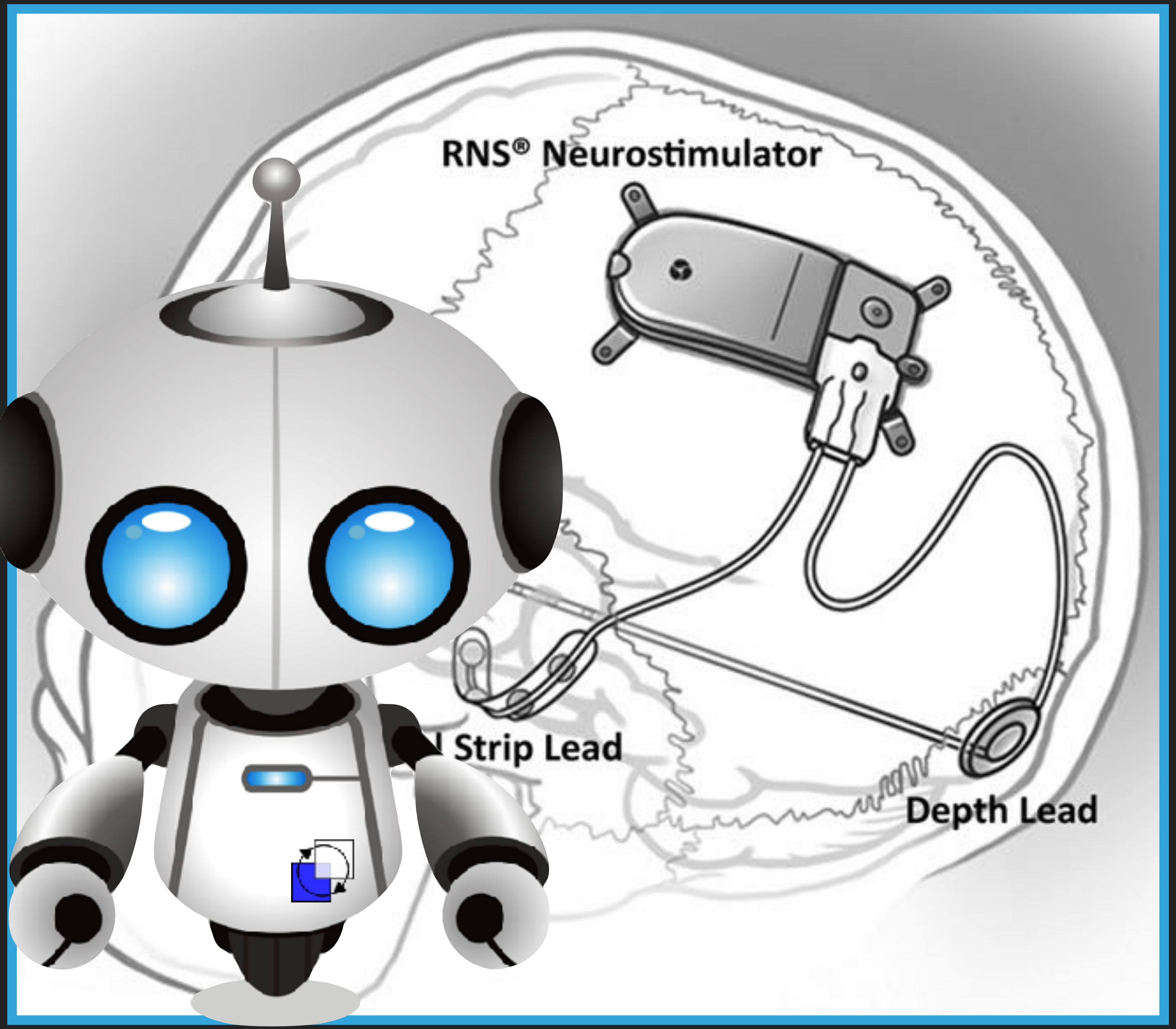
Real-Time ✓

Statistics ✓

Fast ✓

Formal ✓

Summary



- Embedded ✓
- Real-Time ✓
- Statistics ✓
- Fast ✓
- Formal ✓

Summary

RNS® Neurostimulator

RTLola / StreamLAB

Sensor

Sensor

Sensor

Event Processor

Time Processor

Verdict

Verdict

Outlook

Outlook

Outlook