Language Composition

Laurence Tratt
http://tratt.net/laurie/

Software Development Team, King's College London

2013-02-27

Talk aims

1. The programming language status quo limits us.

Talk aims

1. The programming language status quo limits us.
2. Language composition might offer a way forward.

Talk aims

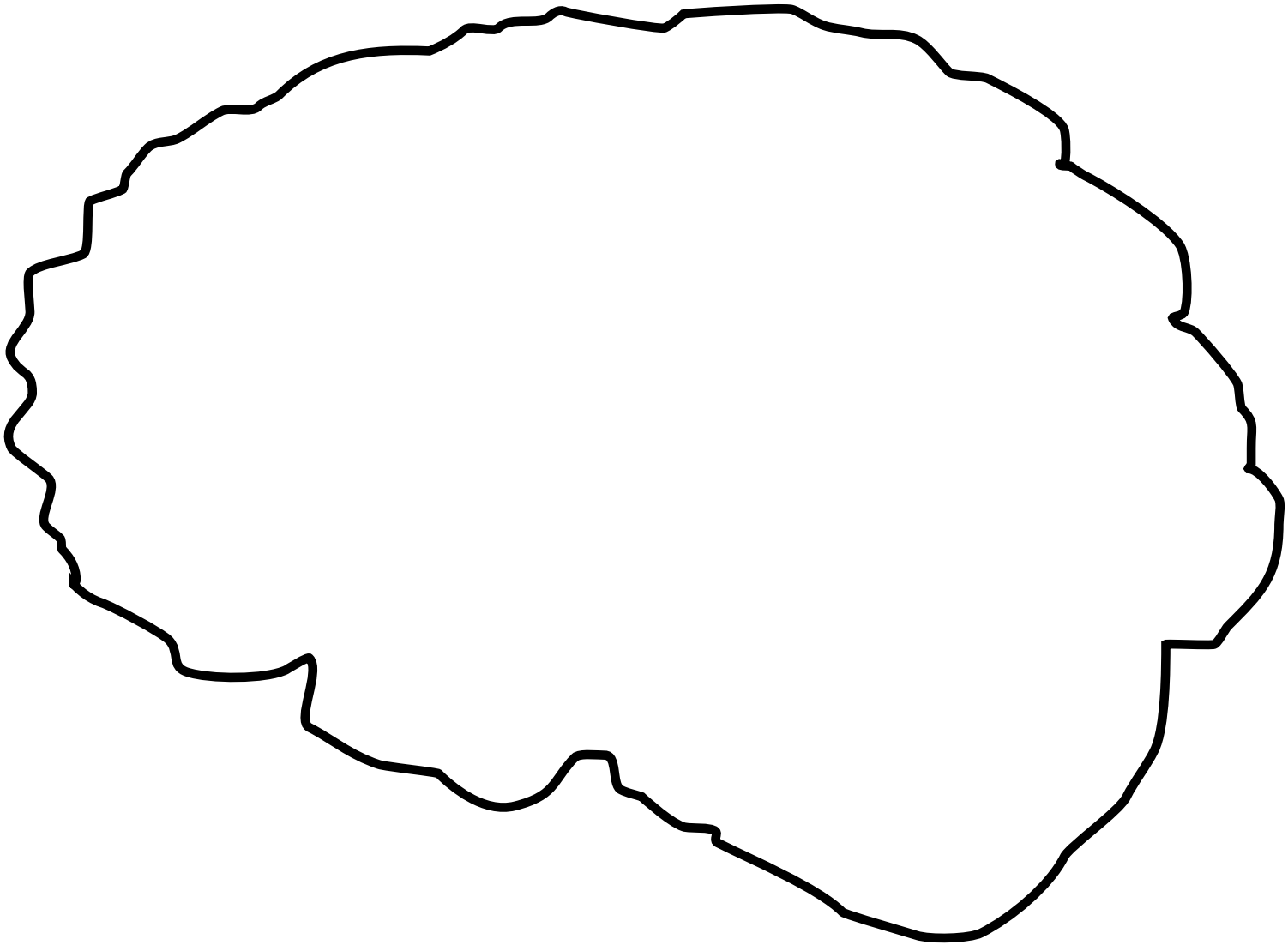
1. The programming language status quo limits us.
2. Language composition might offer a way forward.
3. We're not very good at it yet.

Talk aims

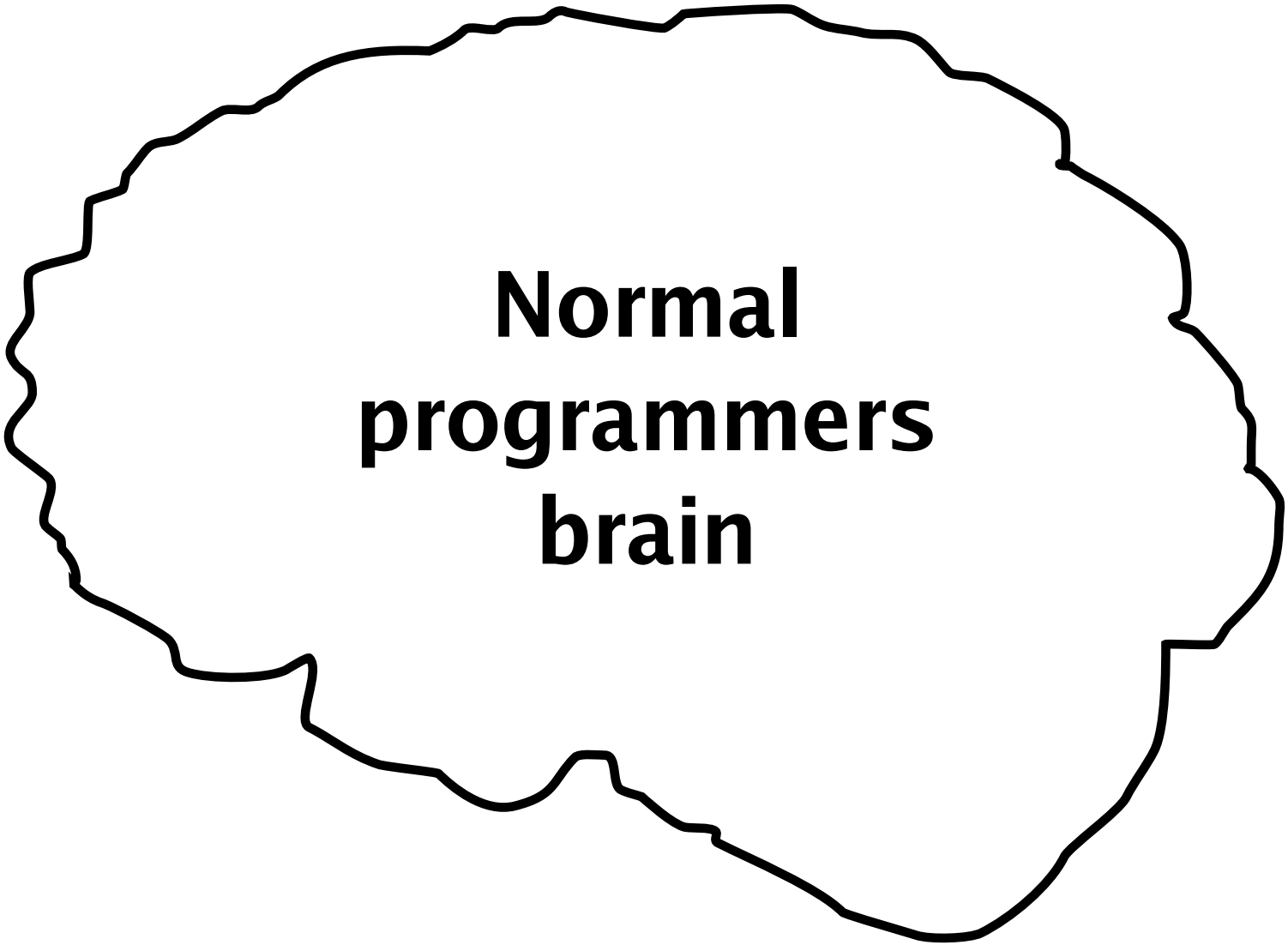
1. The programming language status quo limits us.
2. Language composition might offer a way forward.
3. We're not very good at it yet.
4. Possible future routes.

What's the problem with the status quo?

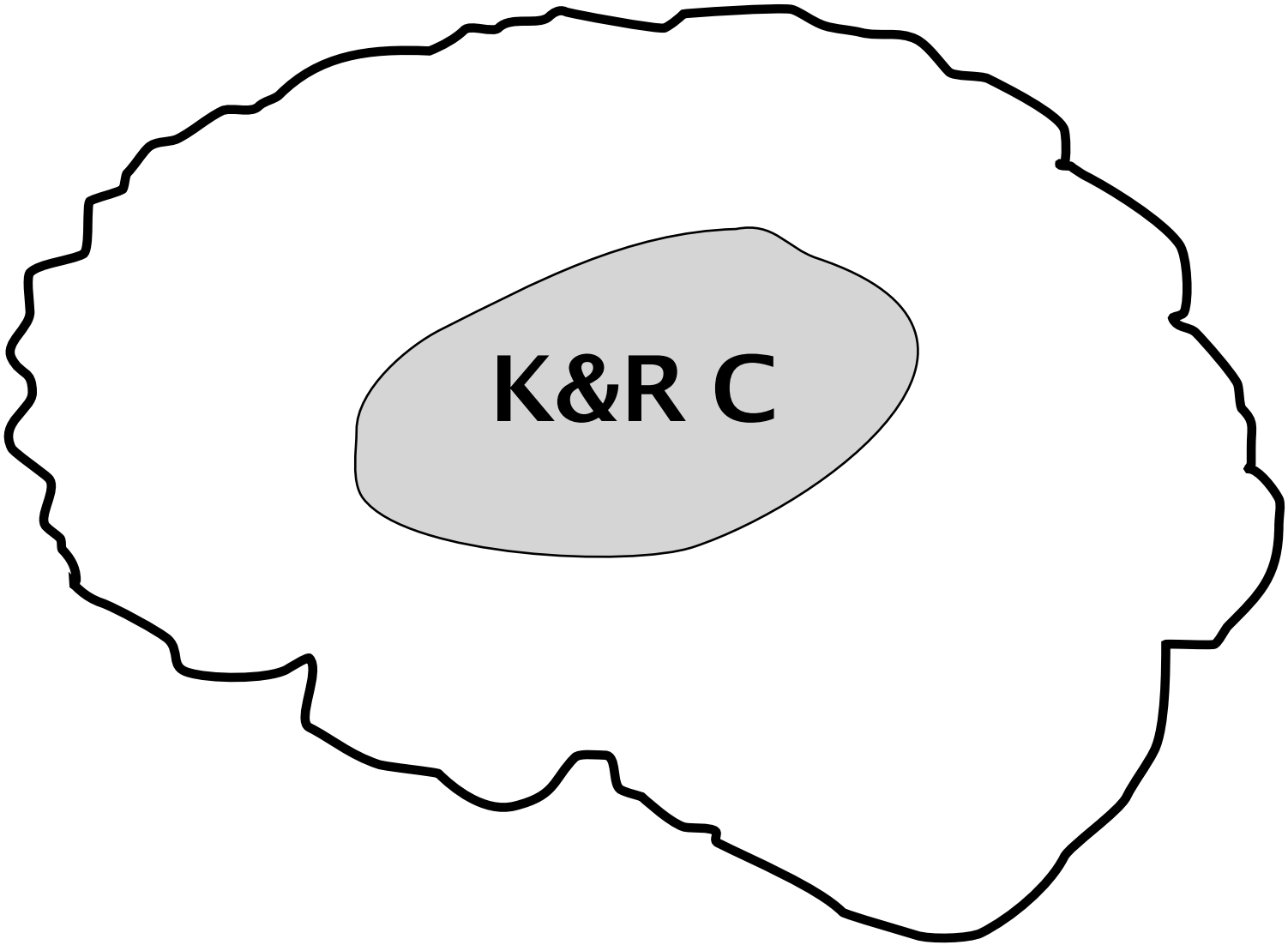
Languages conceptual size



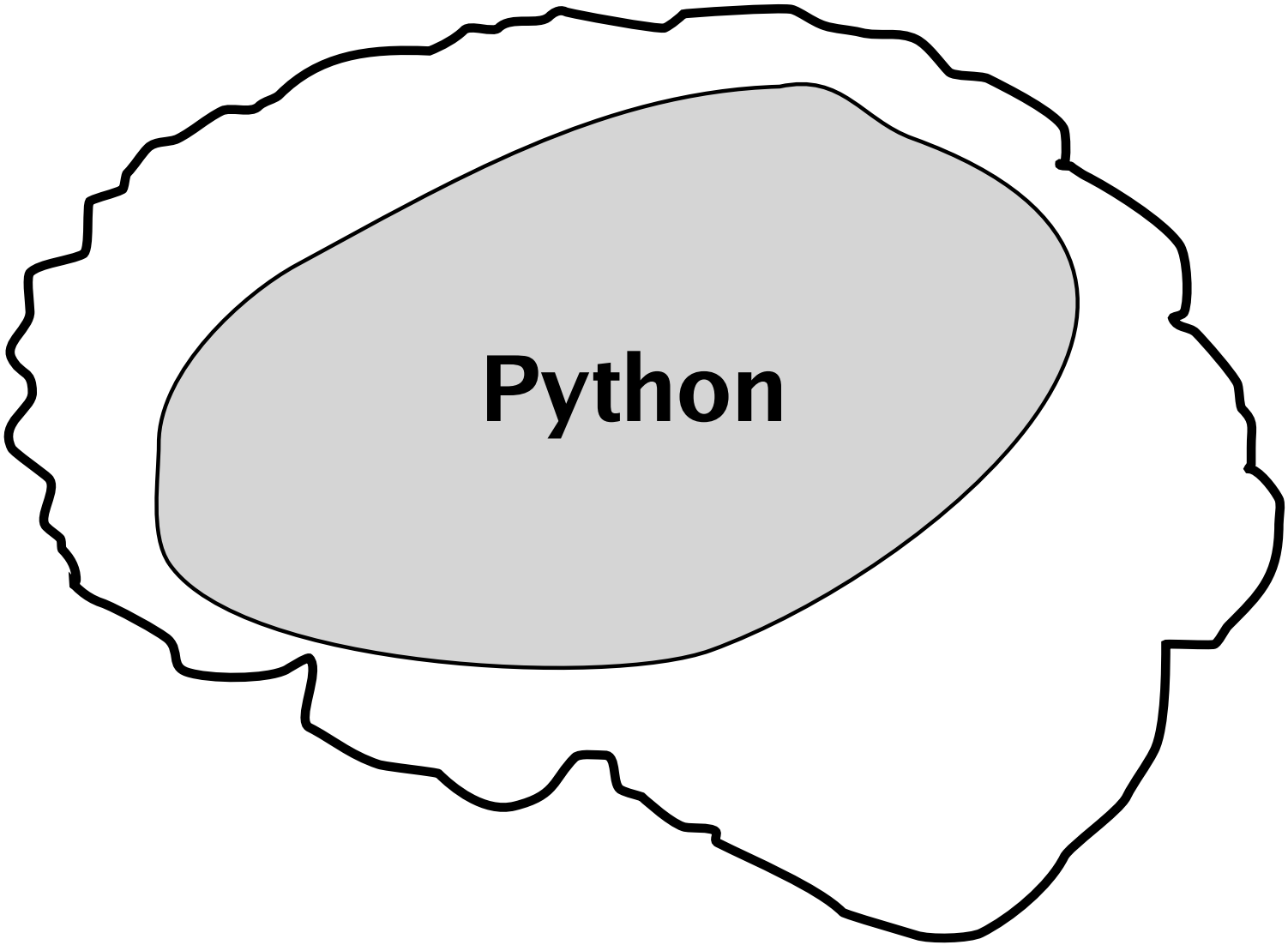
Languages conceptual size



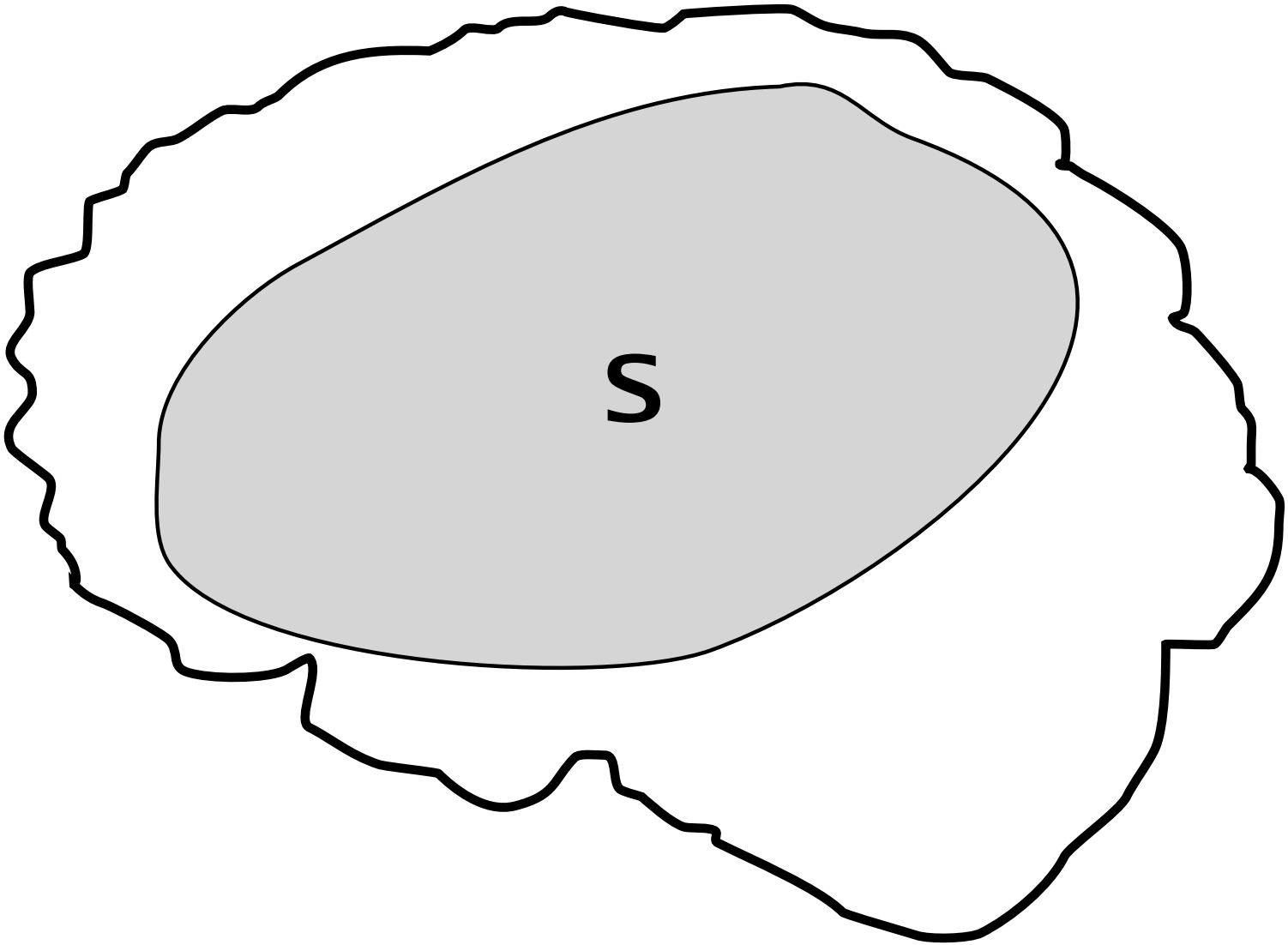
Languages conceptual size



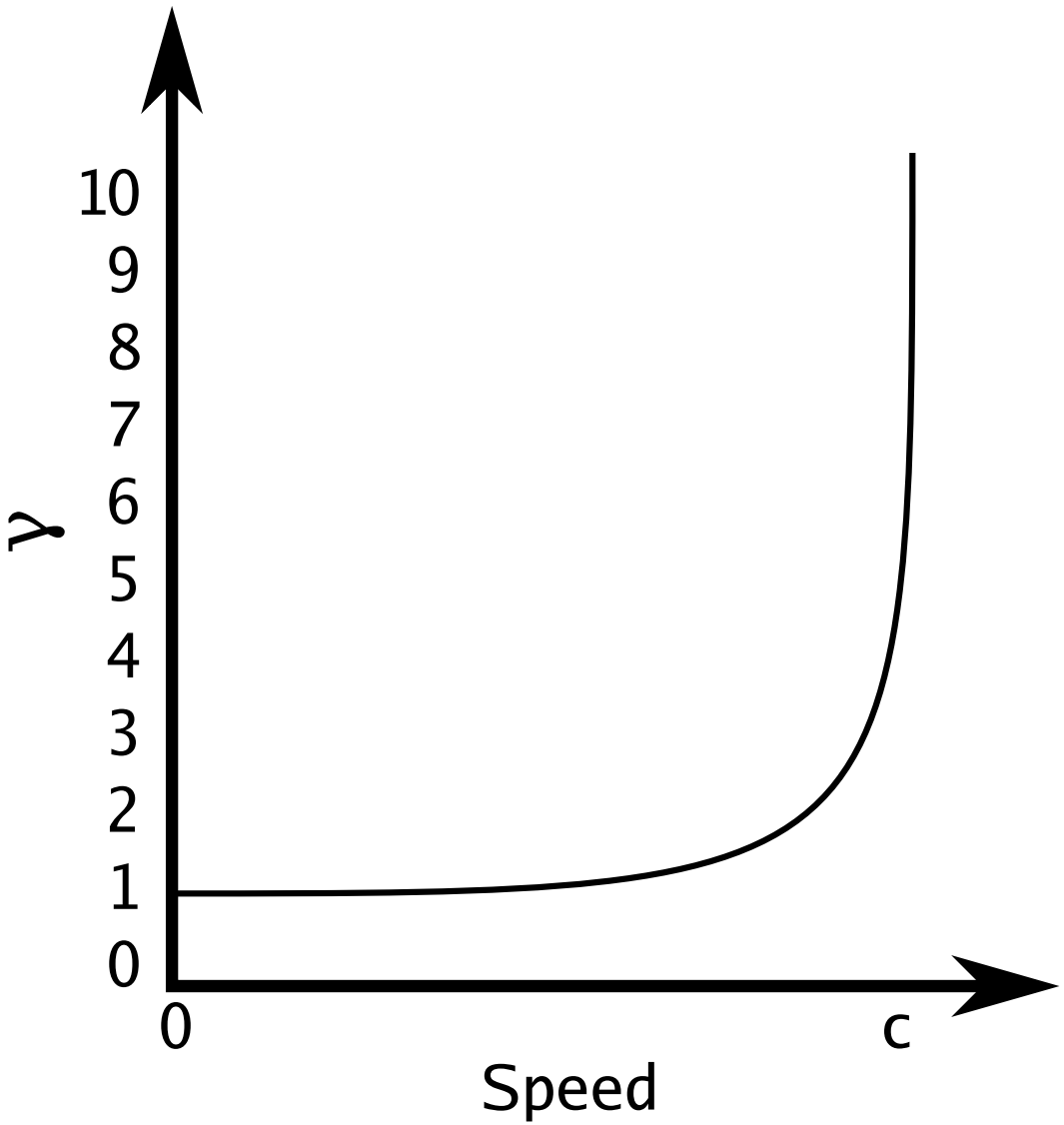
Languages conceptual size



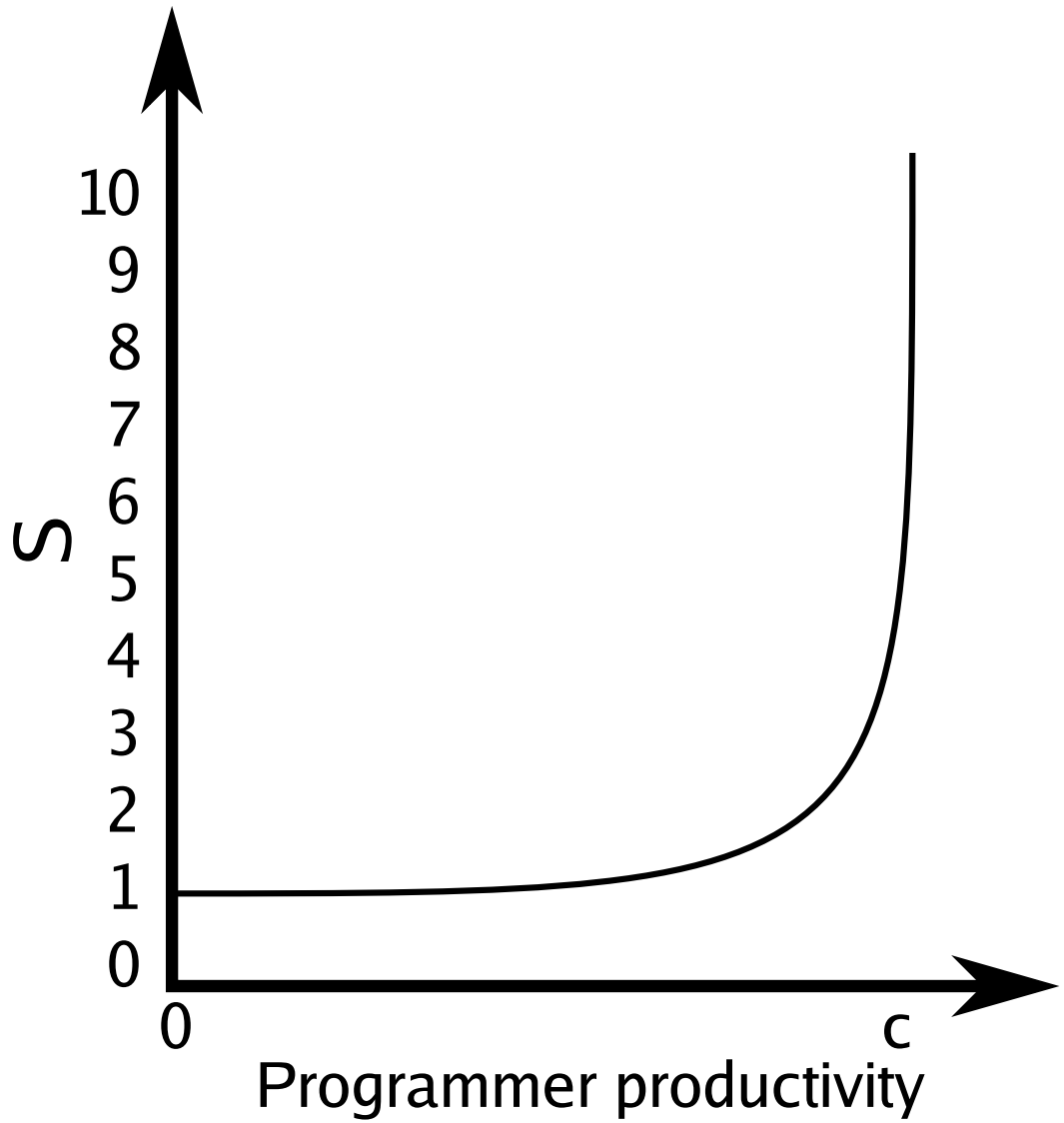
Languages conceptual size



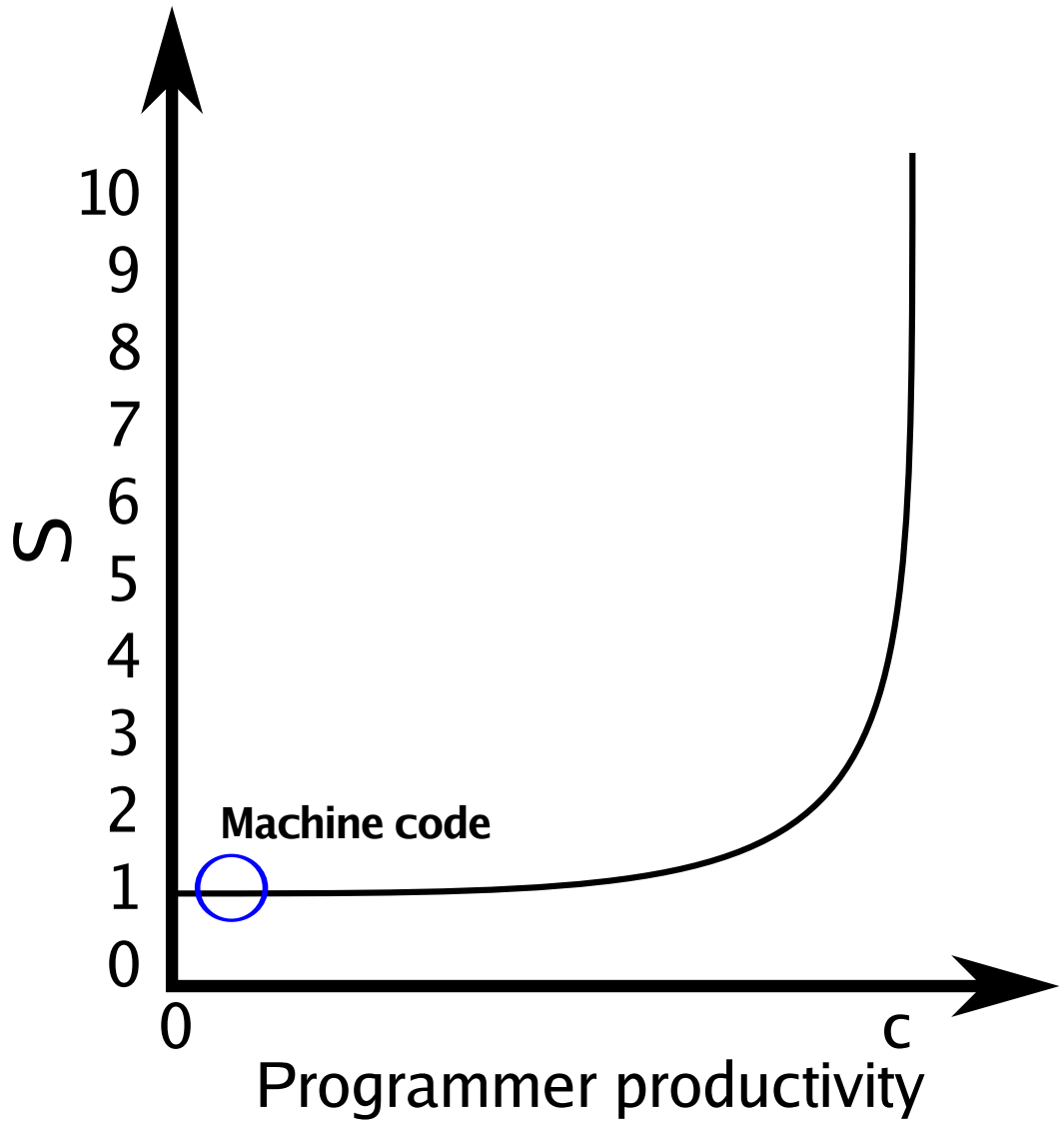
Programming languages' speed of light



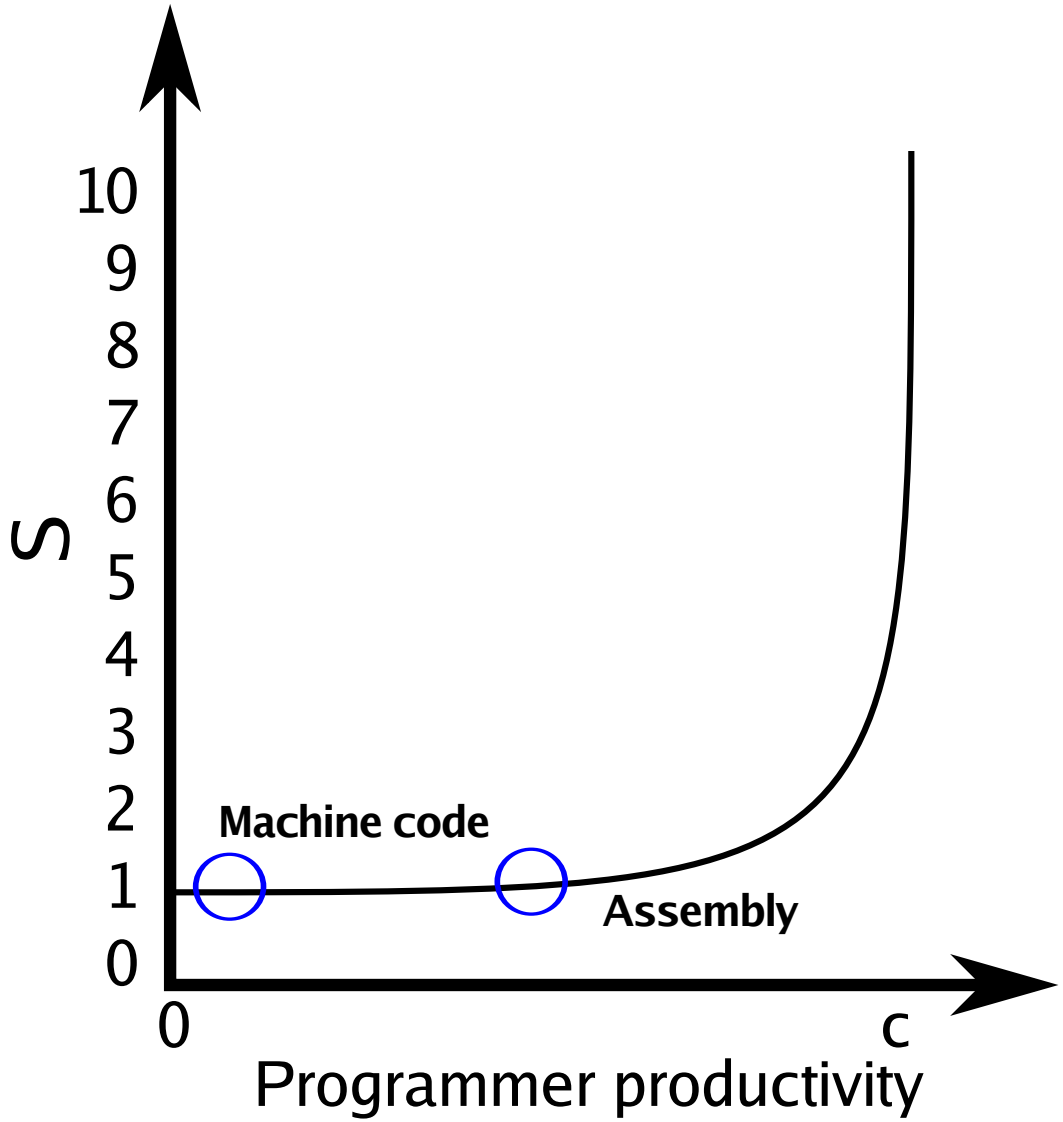
Programming languages' speed of light



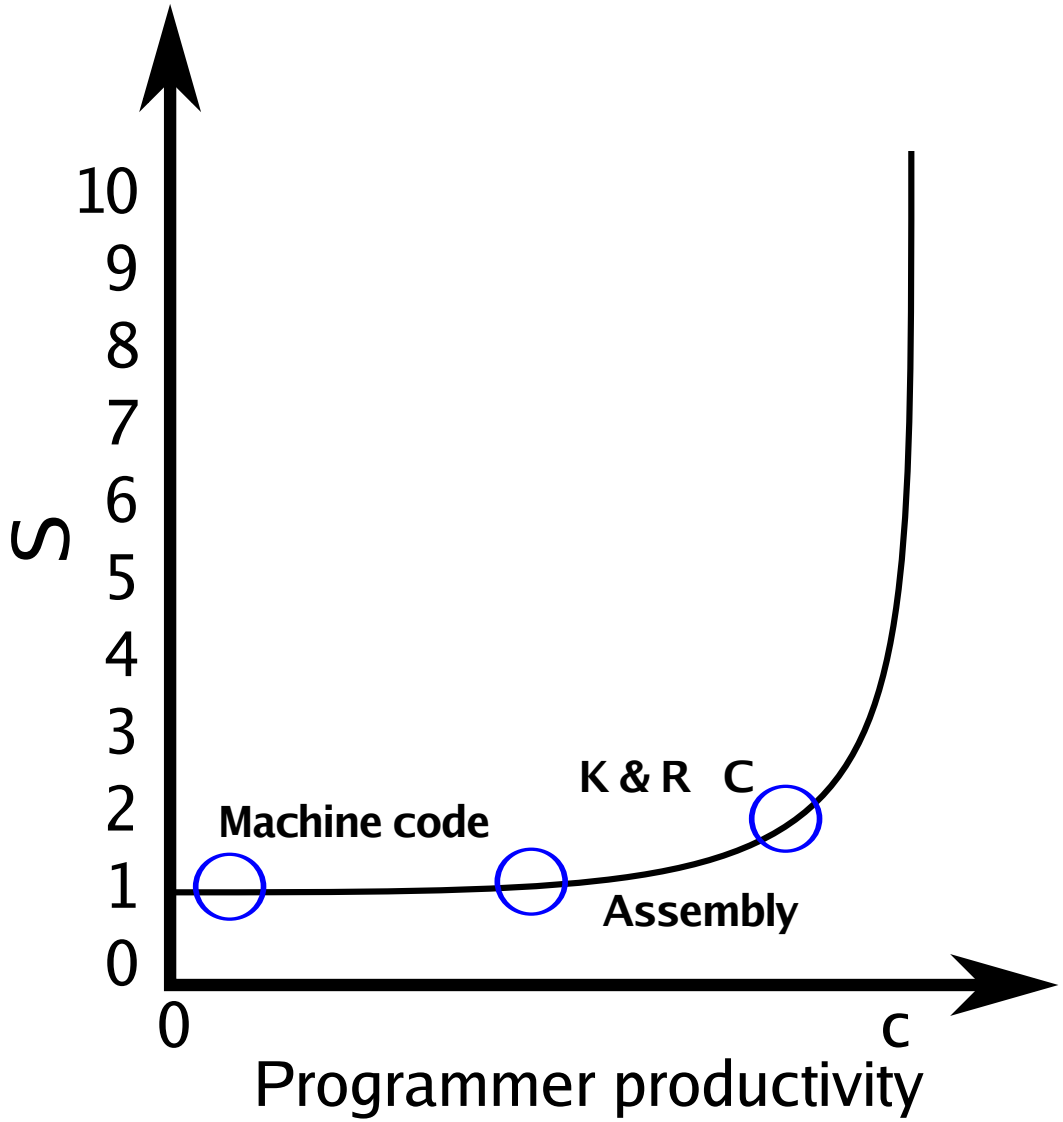
Programming languages' speed of light



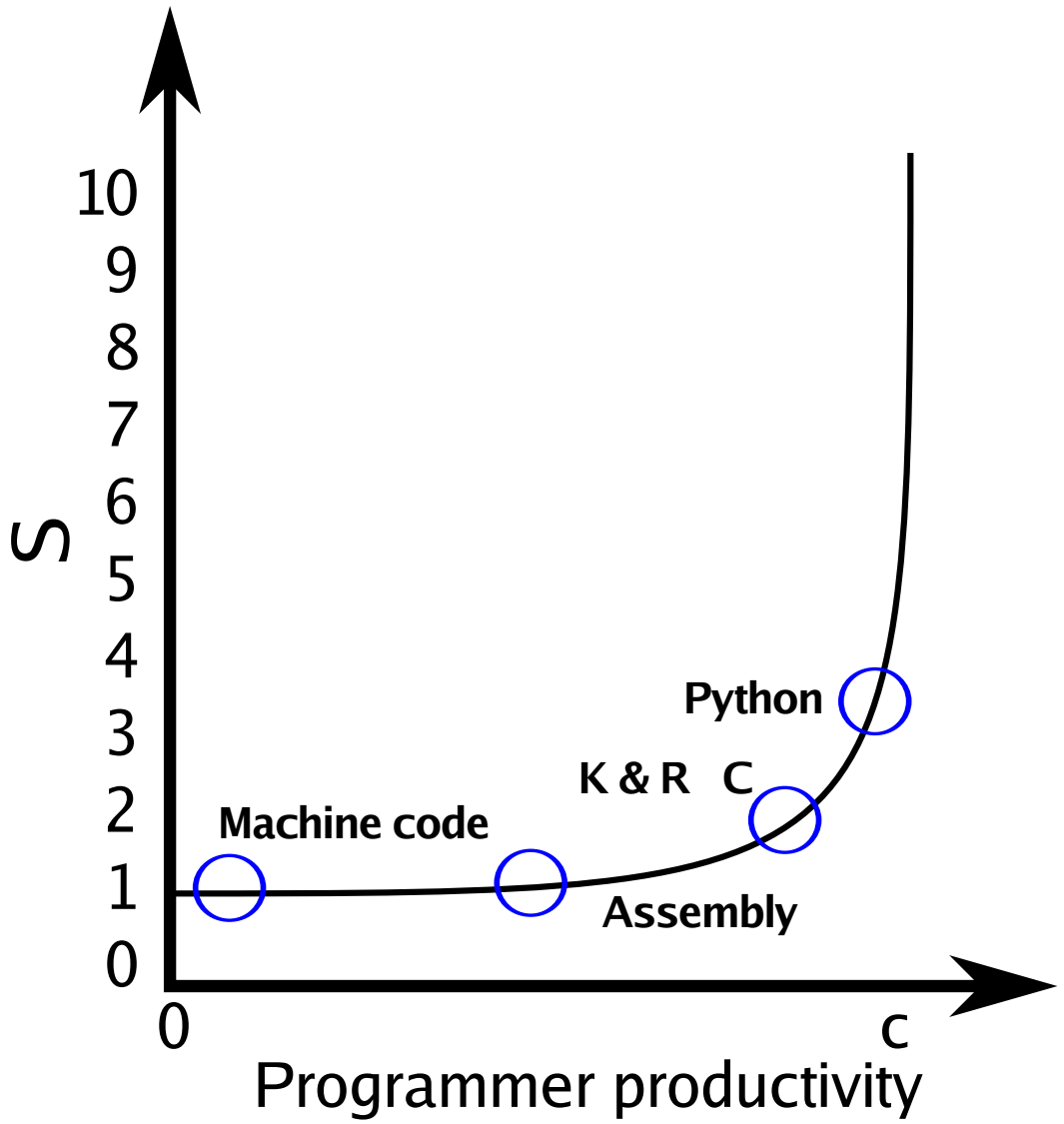
Programming languages' speed of light



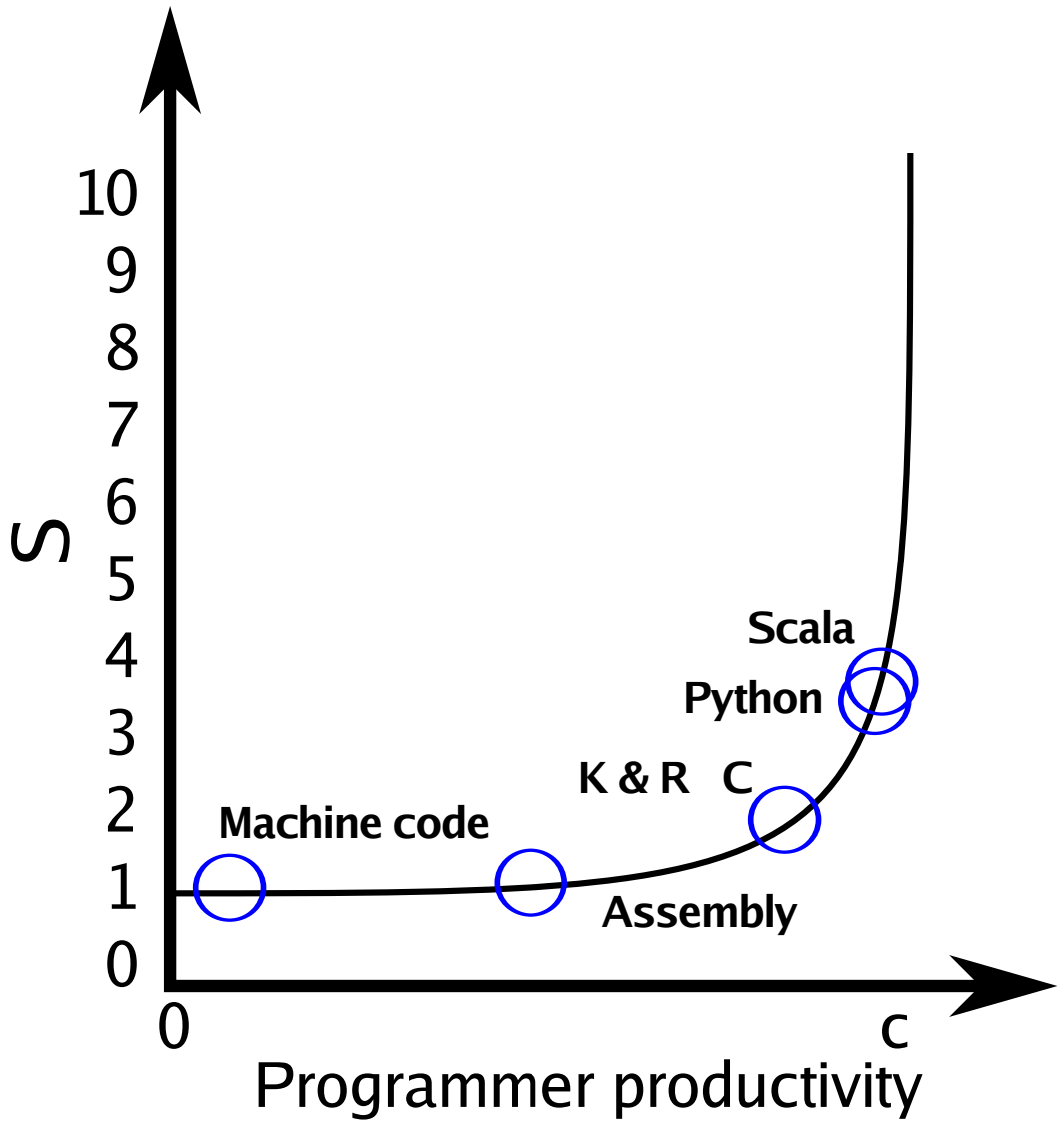
Programming languages' speed of light



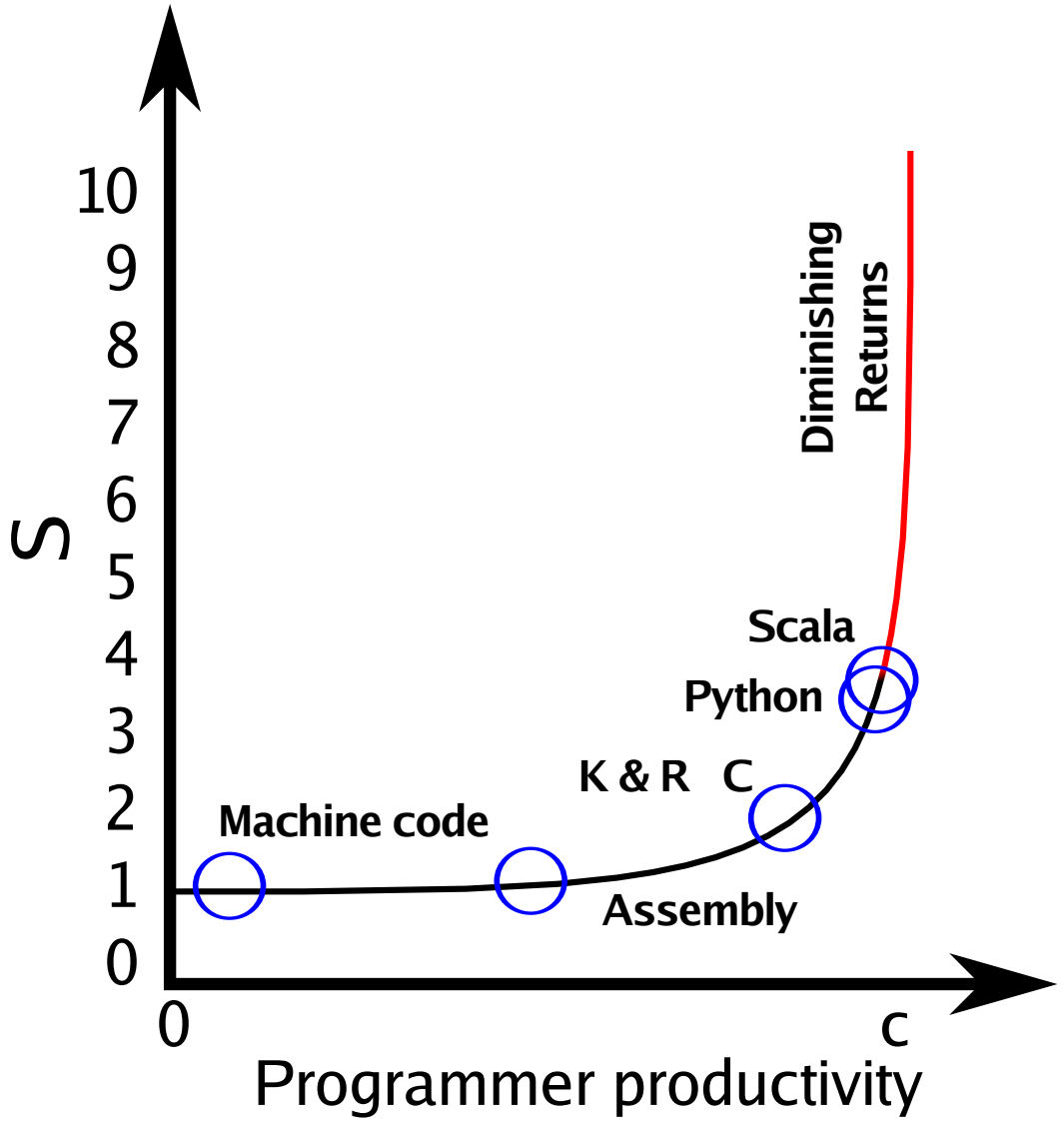
Programming languages' speed of light



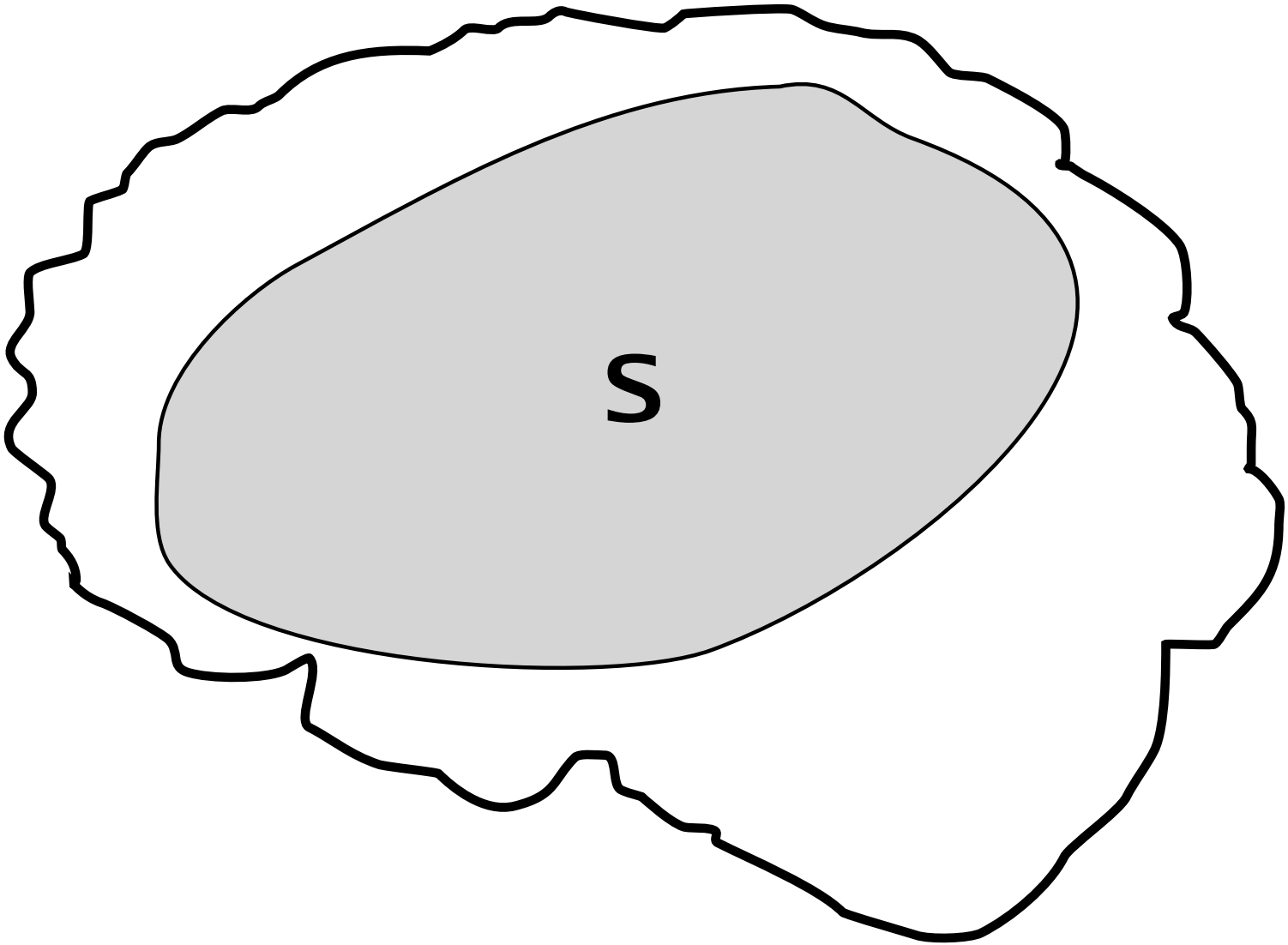
Programming languages' speed of light



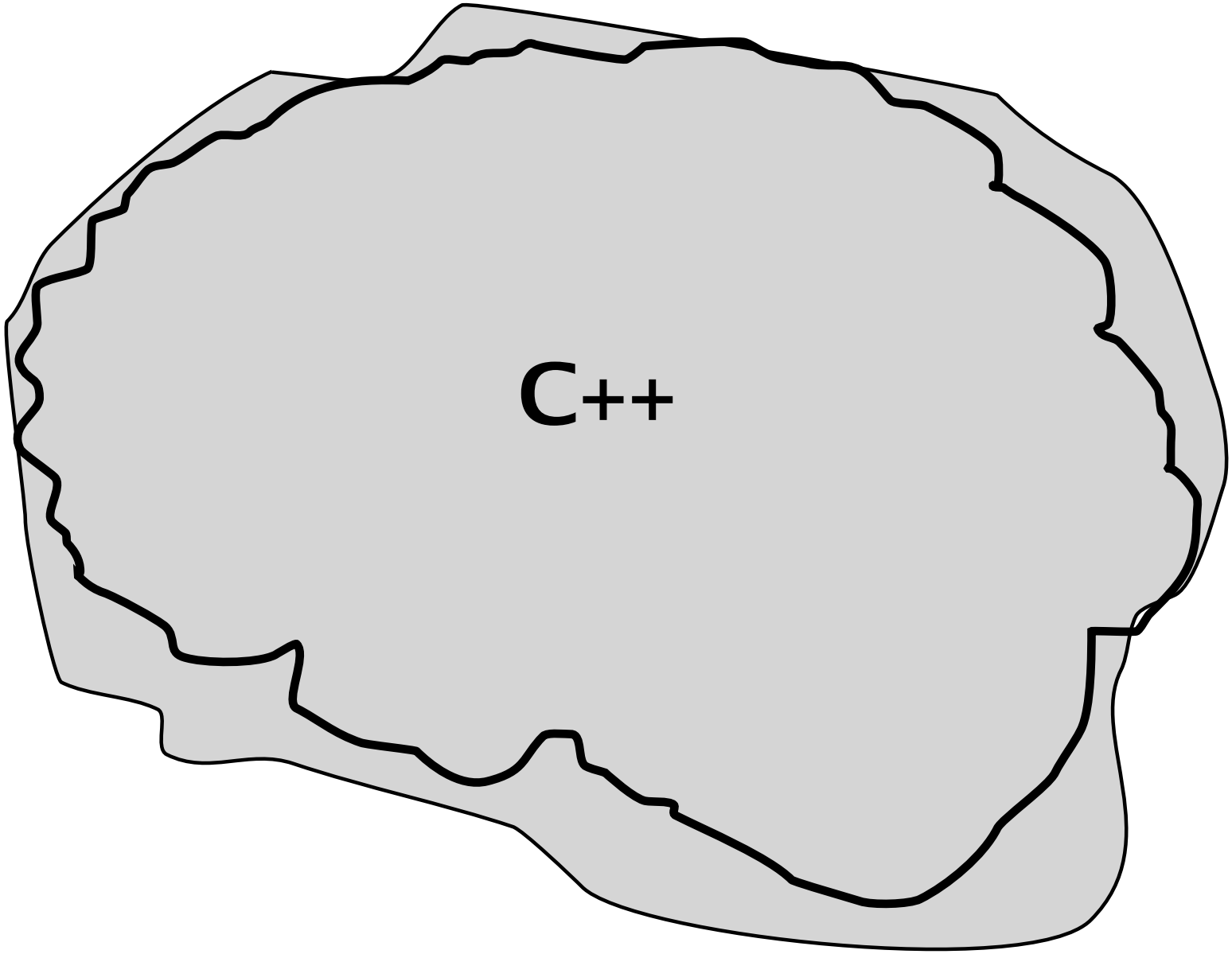
Programming languages' speed of light



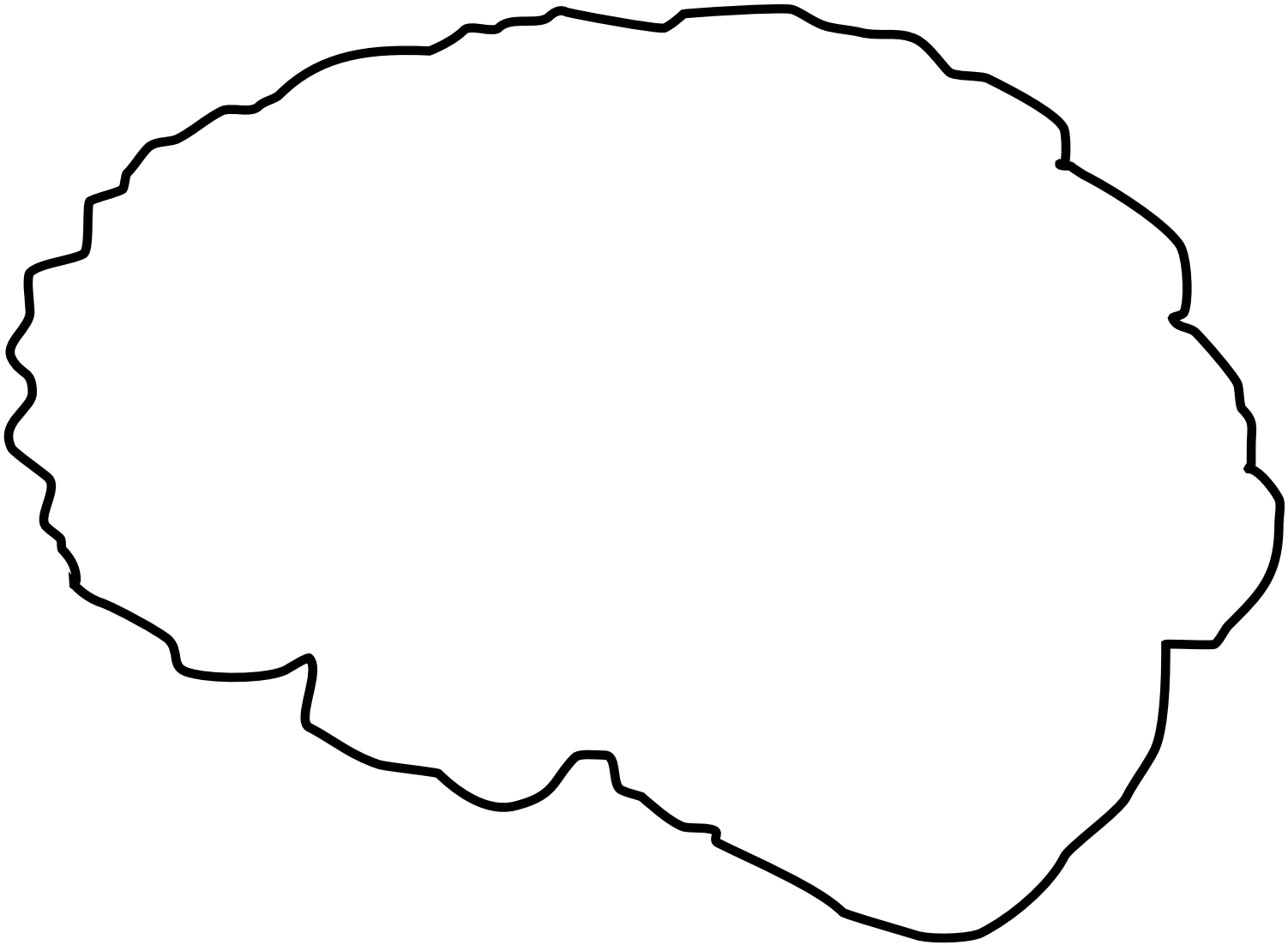
Can S become too big?



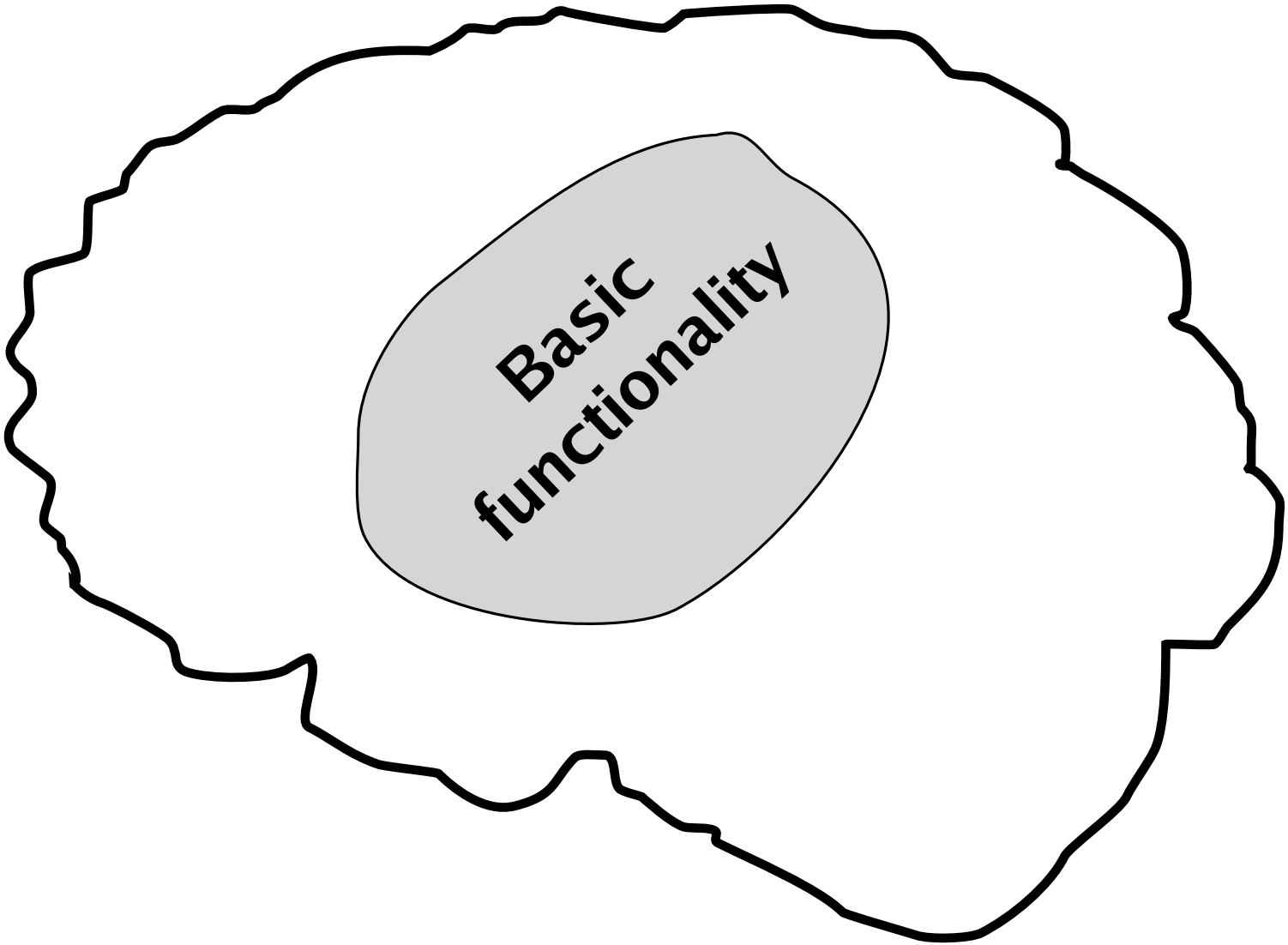
Can S become too big?



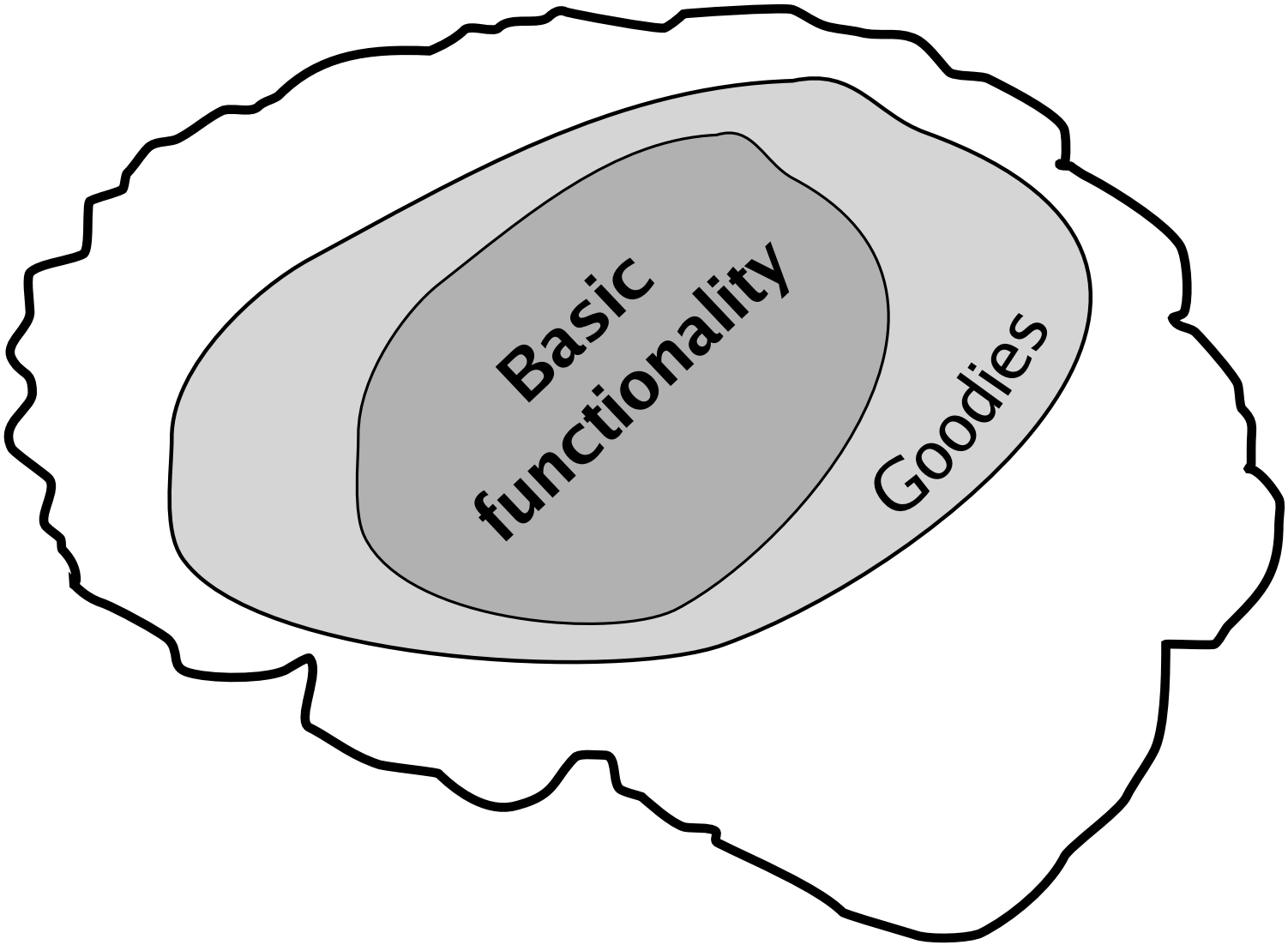
Wiggle room



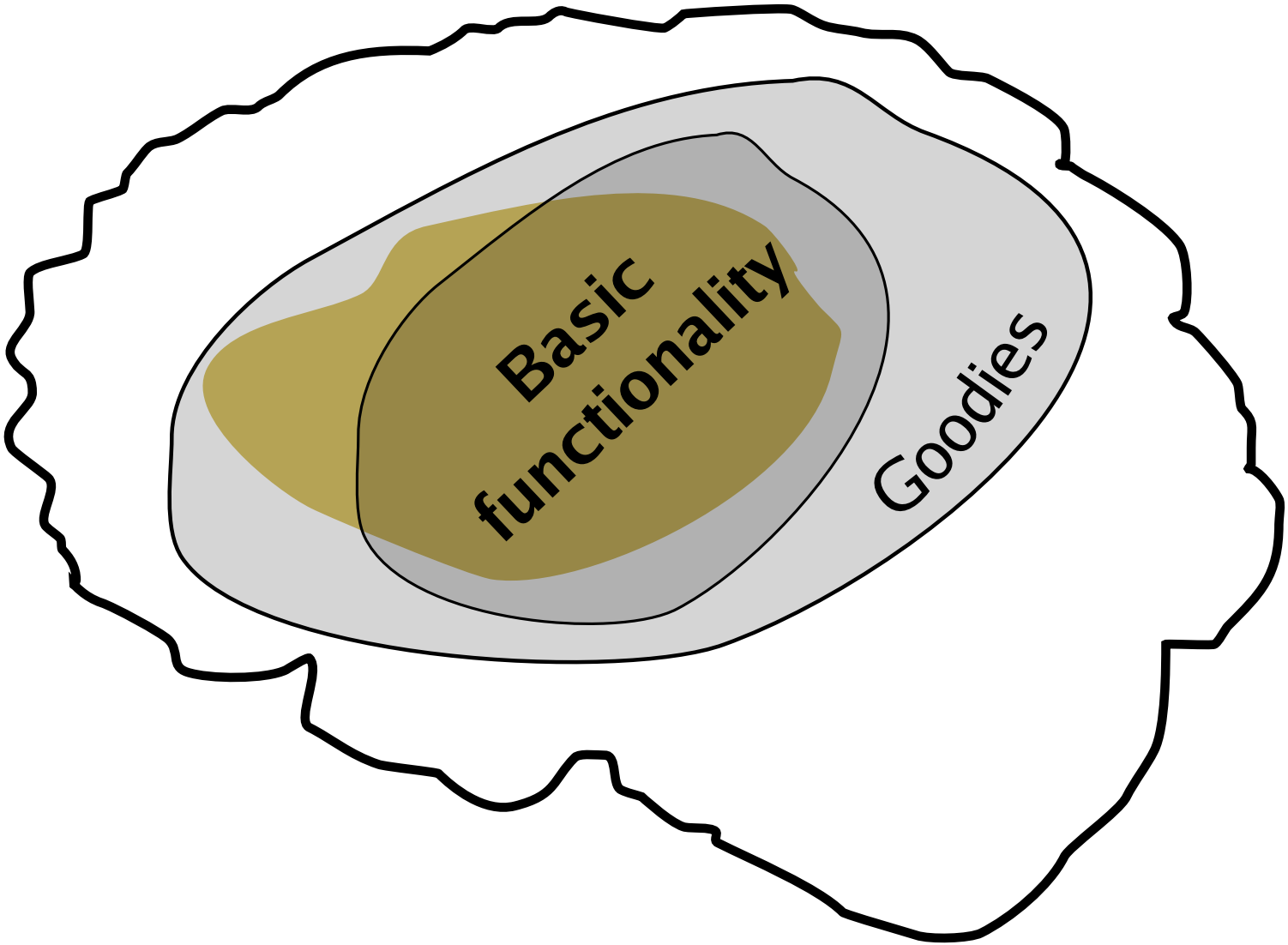
Wiggle room



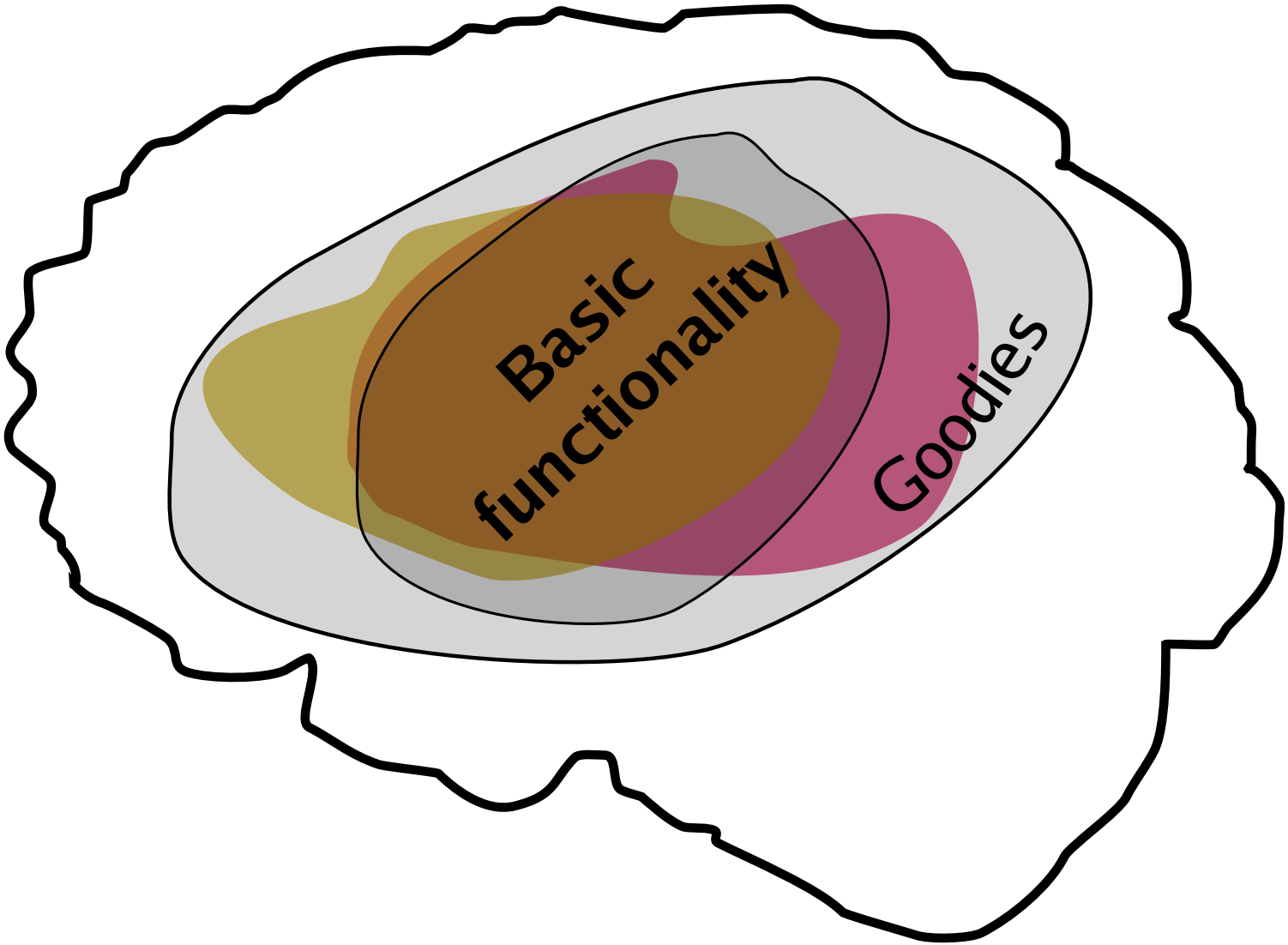
Wiggle room



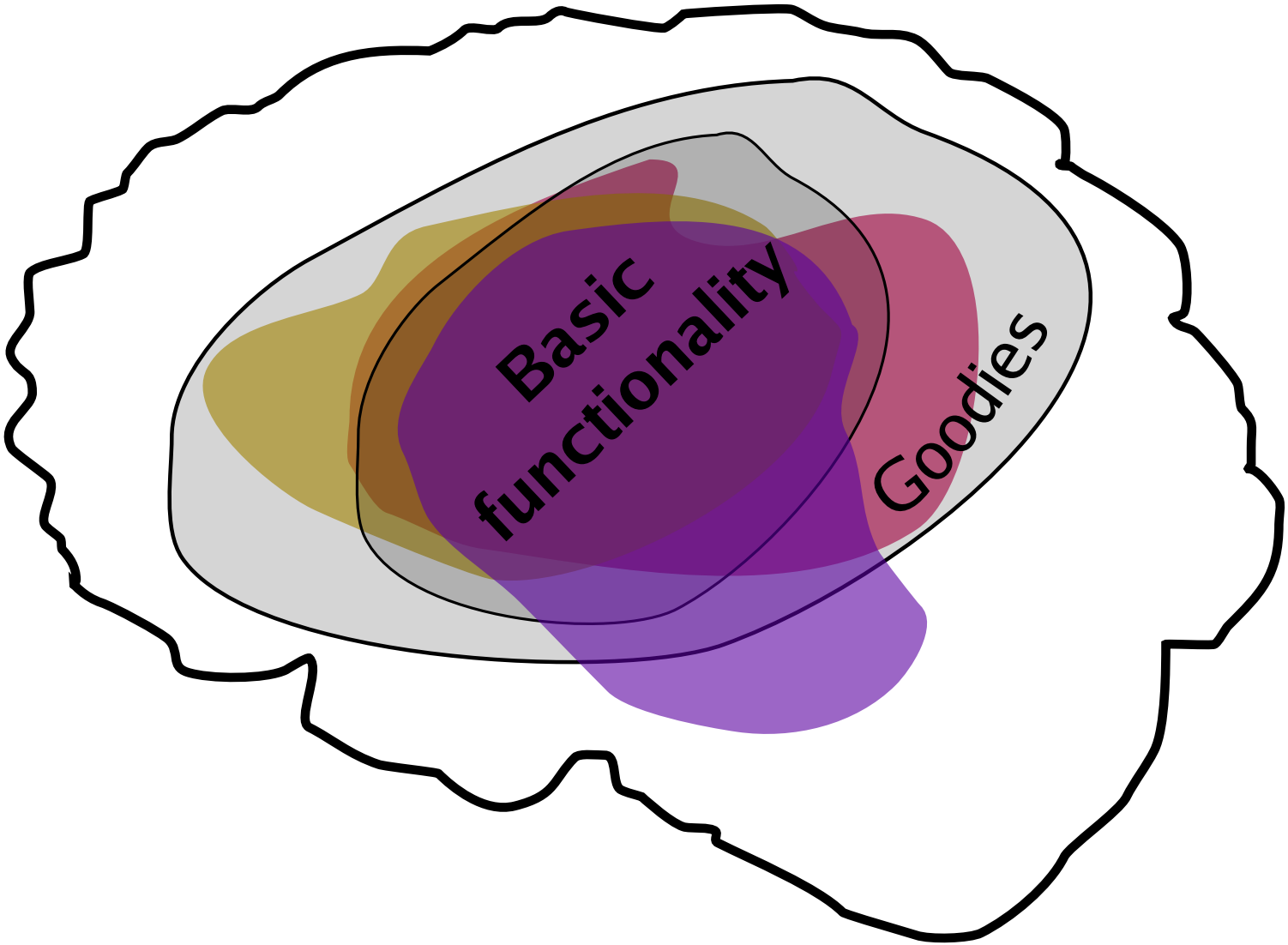
Even worse



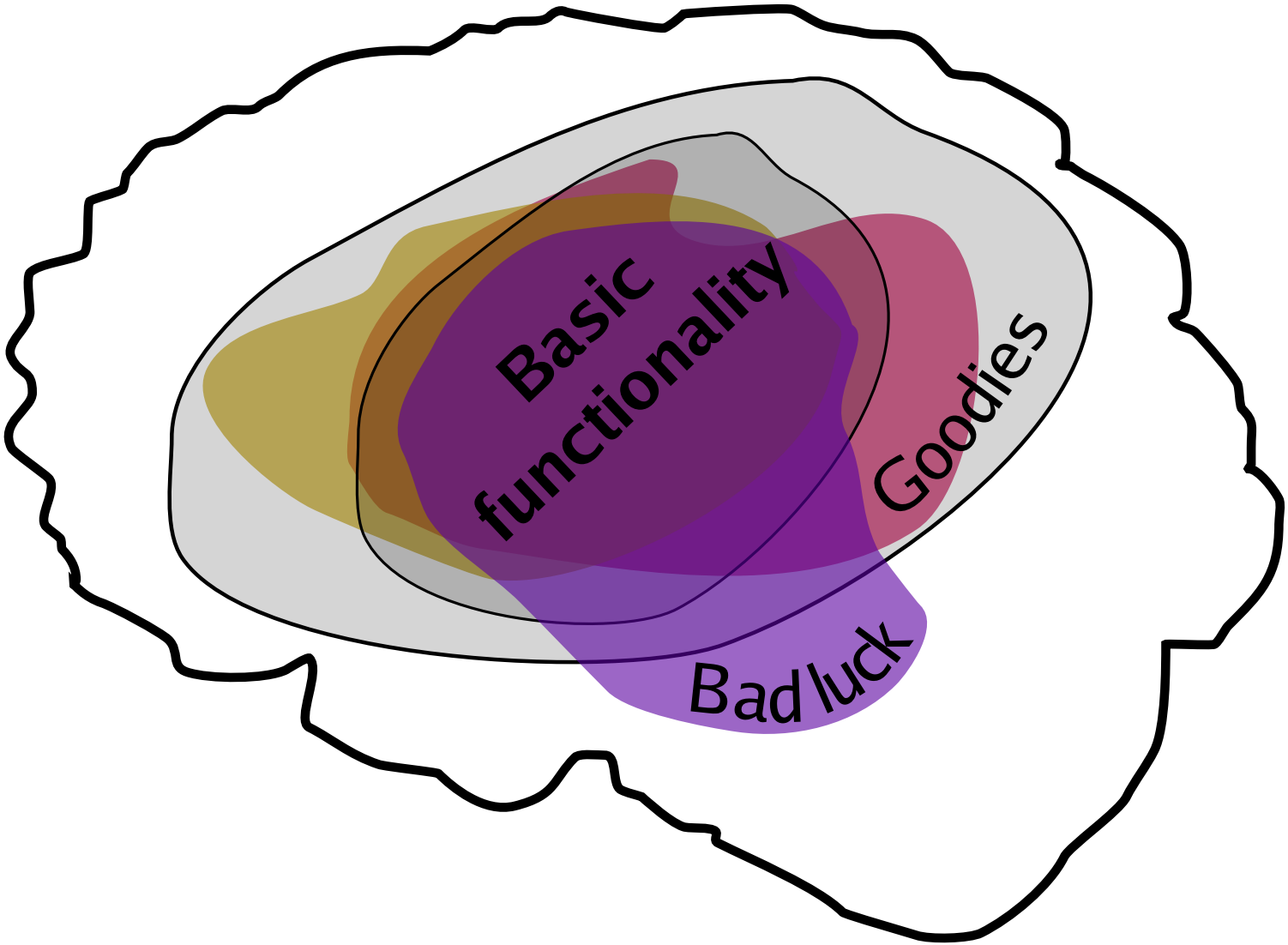
Even worse



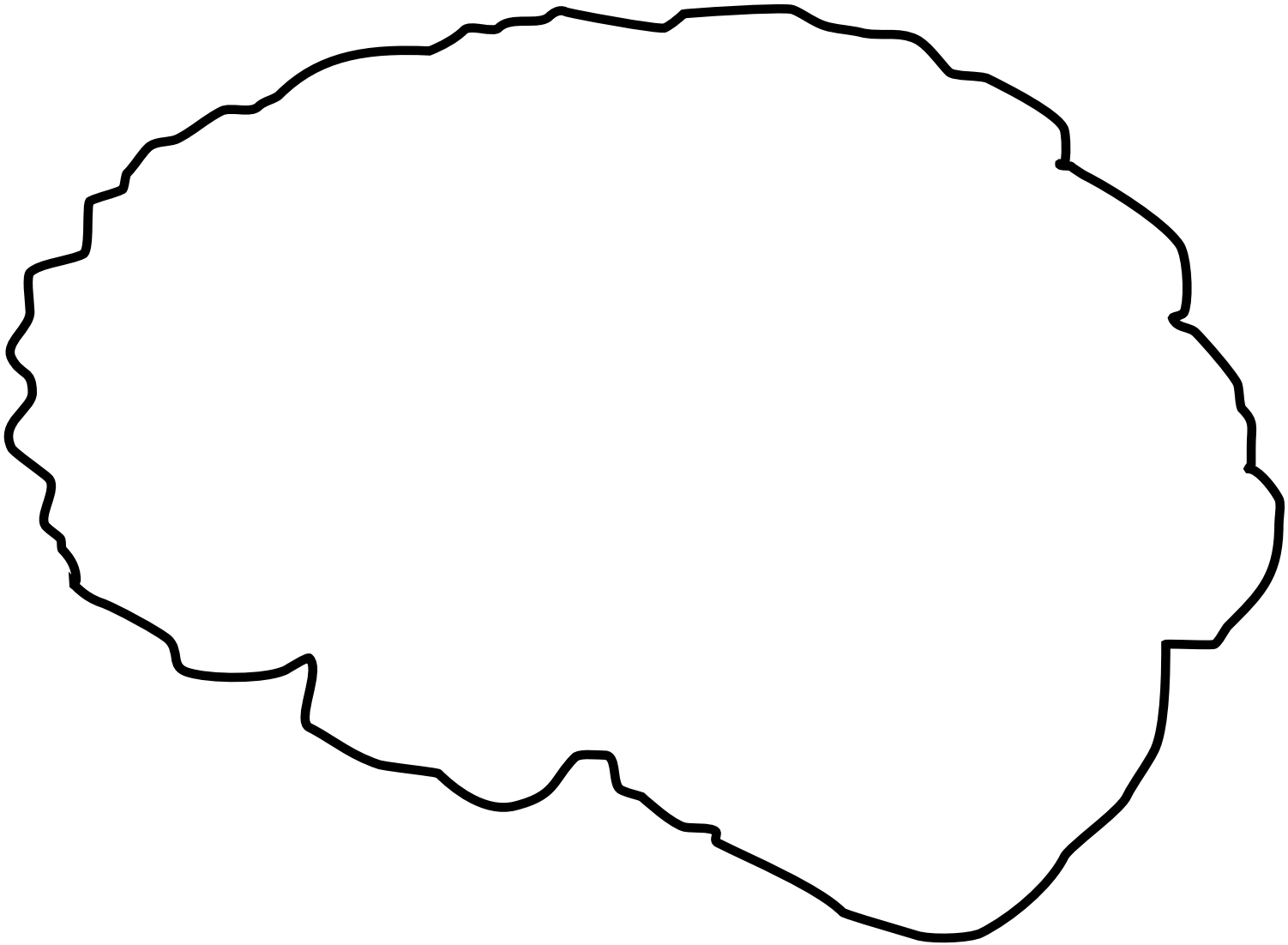
Even worse



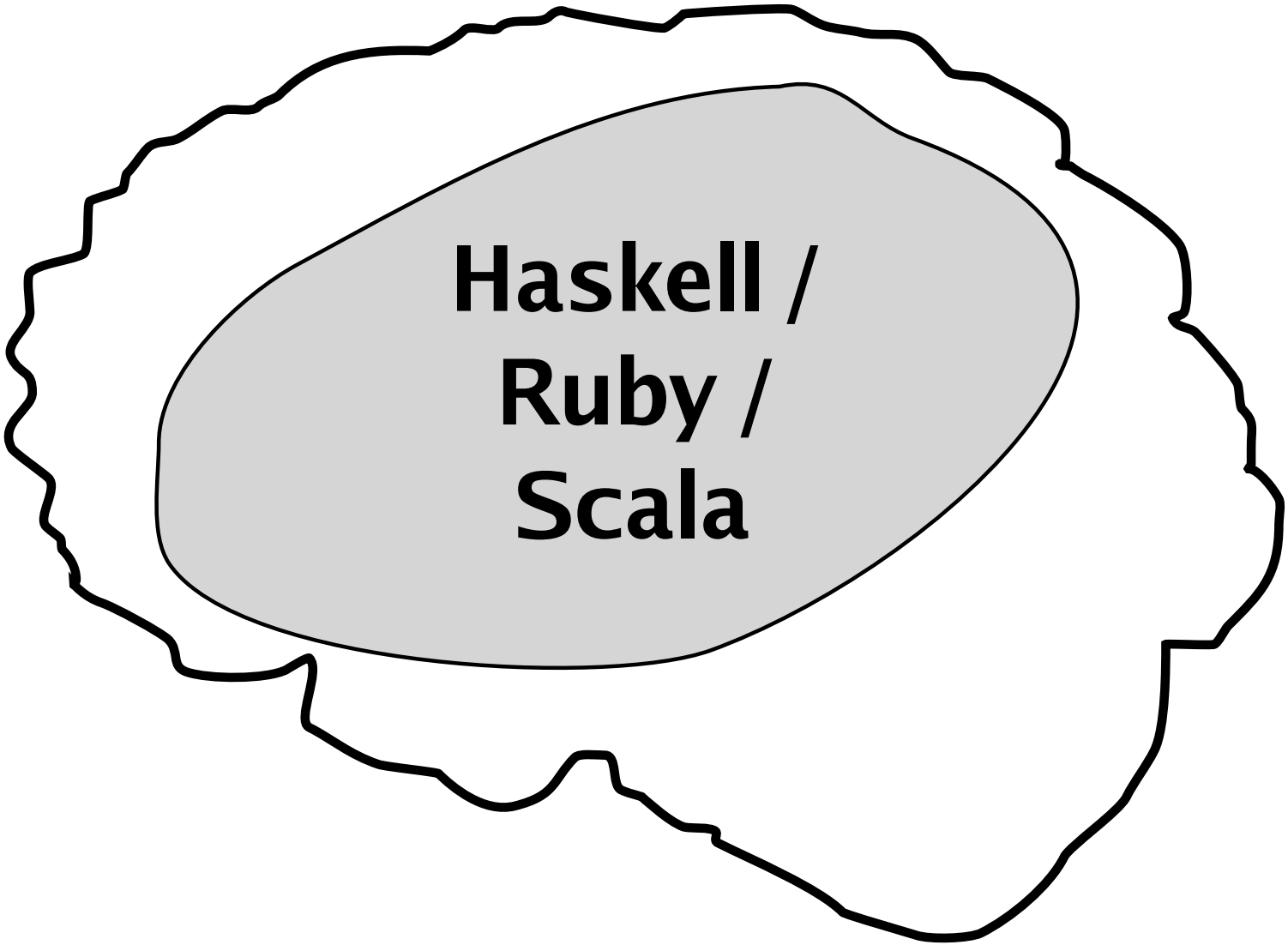
Even worse



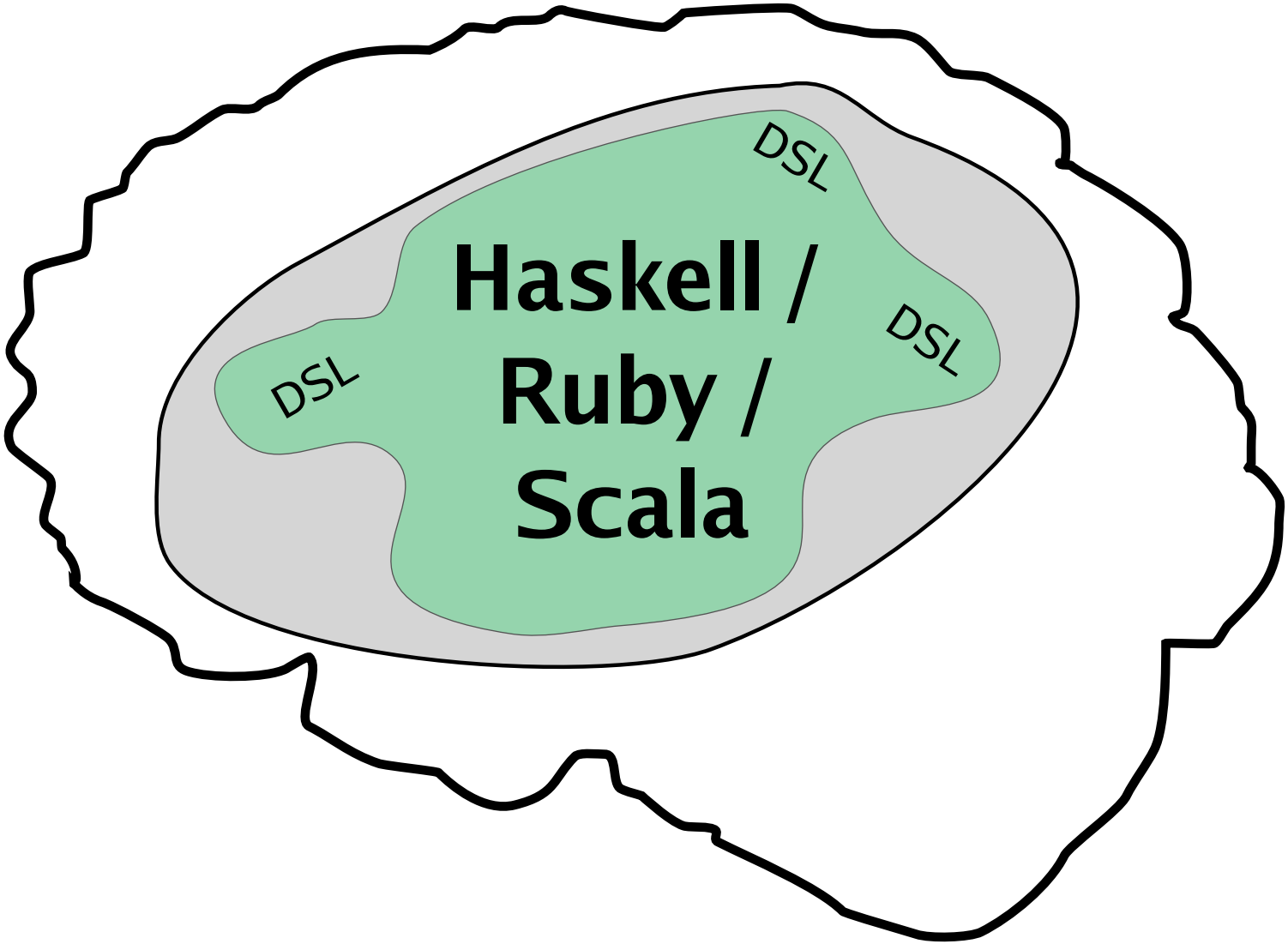
Is this about DSLs?



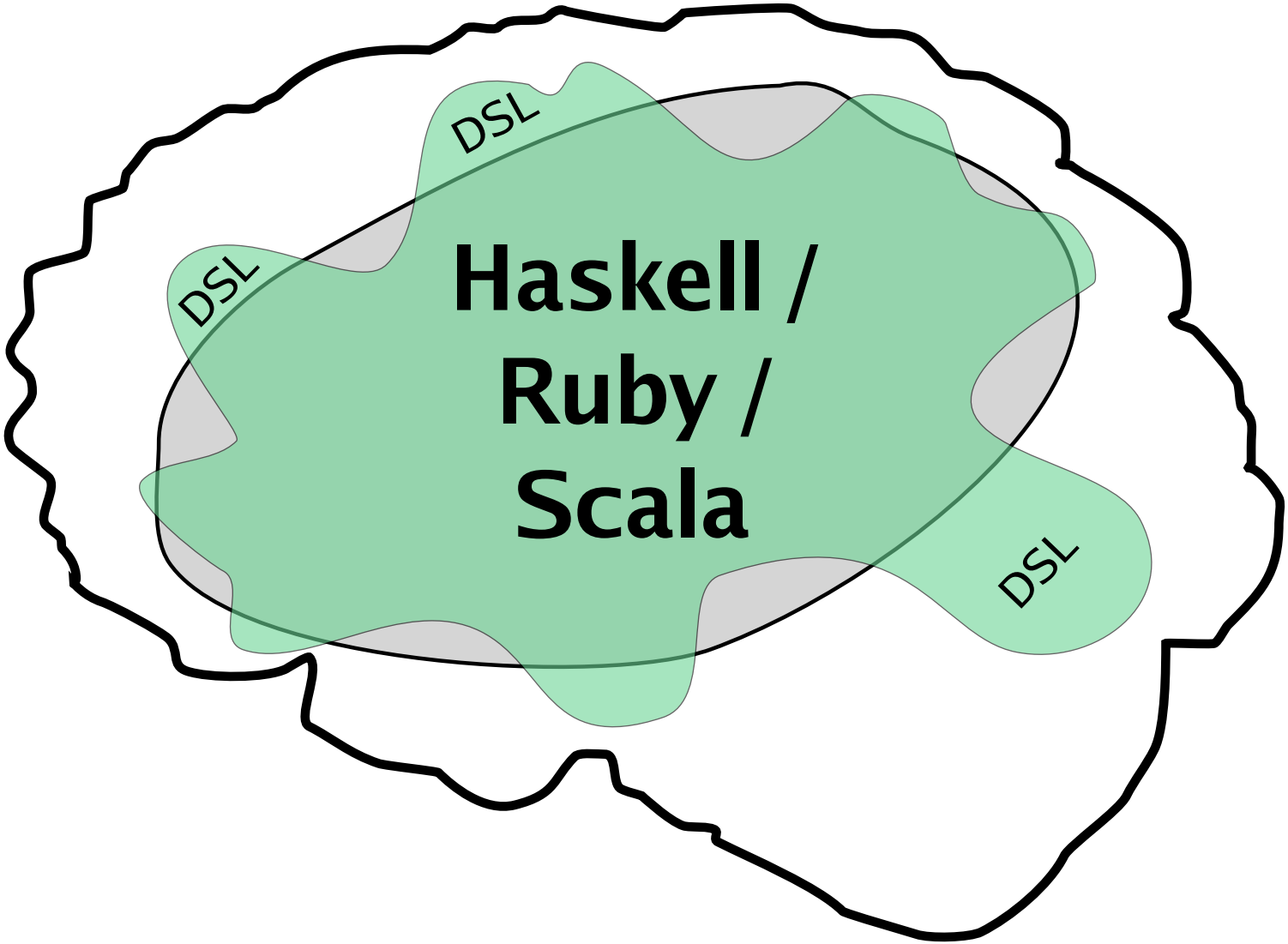
Is this about DSLs?



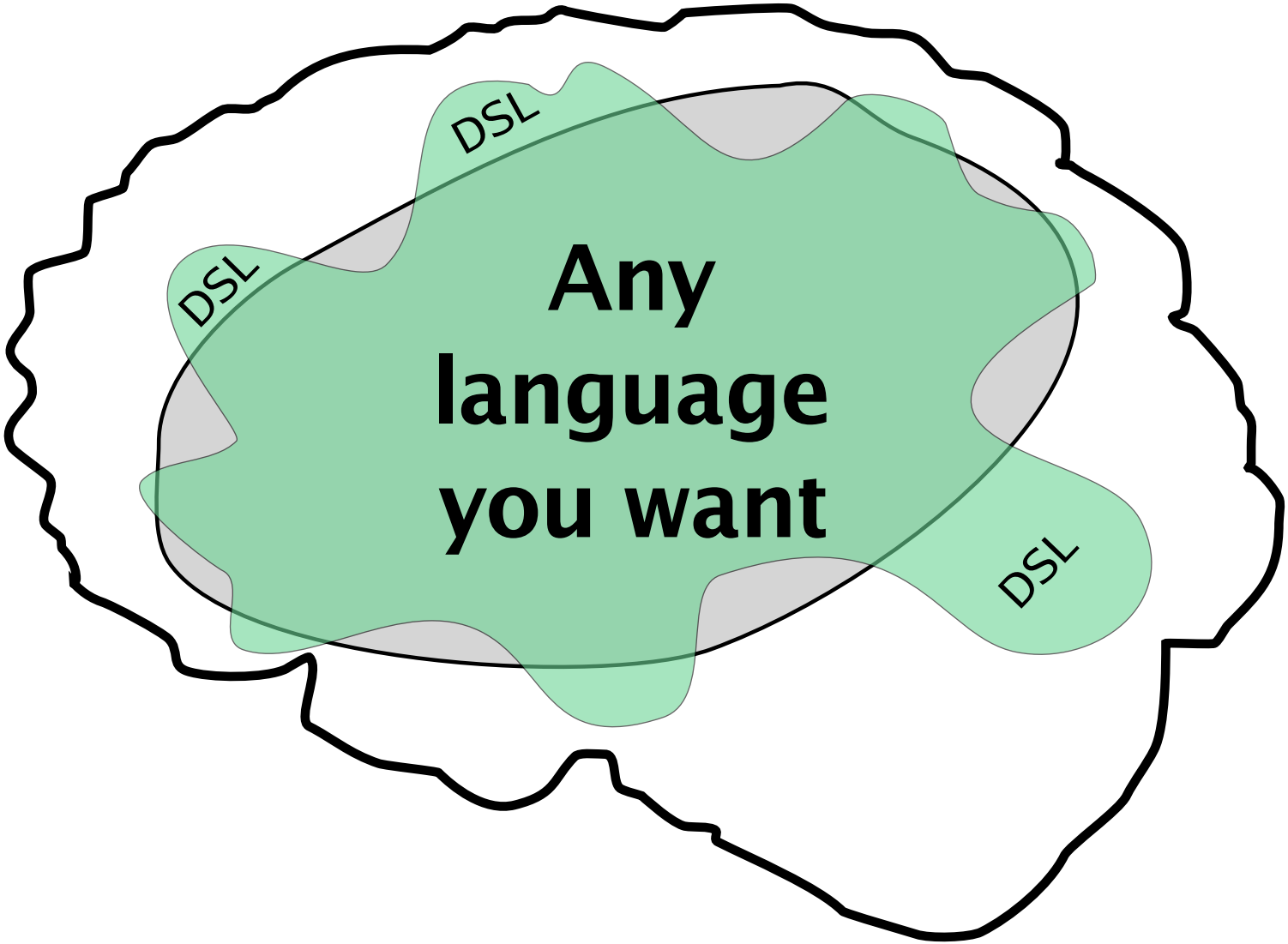
Is this about DSLs?



Is this about DSLs?



Is this about DSLs?



A way forward?

Idea: allow users to *compose* languages.

What does composition mean?

What does composition mean?

$$\text{Language} \triangleq$$
$$\text{syntax} + \text{semantics}$$

What does composition mean?

$$\text{Language implementation} \triangleq \text{compiler} + \text{runtime}$$

What does composition mean?

$$\text{Language implementation} \triangleq \text{compiler} + \text{virtual machine}$$

What does composition mean?

$$\text{Compiler} \triangleq$$
$$\text{parser} + \text{code generator}$$

What does composition mean?

$$\text{Compiler} \triangleq$$
$$\text{parser} + \cancel{\text{code generator}}$$

What does composition mean?

Compose:

- parsers
- virtual machines

Example (1)

Example (1)

SQL and Java

```
for (pid : SELECT pid FROM personnel WHERE salary > 100000) {
   if (!is_worth_it(pid))
     UPDATE personnel SET salary=0 WHERE pid=pid;
}
```

Example (2)

Example (2)

Tax code

```
income tax {
  2010-2011 {
    allowance {
      age < 65: £6,475
      age >= 65 and age <= 74: £9,490
      age > 74: £9,640

      reduction: if income > £100,000 then
        max(0, allowance - ((income - £100,000) / 2))
    }
  }
}
```

Part III

Why aren't we (me?) very good at it yet?

Converge

Converge

$$\text{Converge} \triangleq \text{Python} + \text{macros}$$

Converge

$$\text{Converge} \triangleq \text{Python} + \text{compile-time meta-programming}$$

Compile-time meta-programming

Code (as trees, not text) is programmatically generated.

Compile-time meta-programming

Code (as trees, not text) is programmatically generated.

Expression	`2 + 3`	evaluates to `5`.
Splice	`$<x>`	evaluates `x` at compile-time; the AST returned overwrites the splice.
Quasi-quote	`[| 2 + 3 |]`	evaluates to a *hygienic* AST representing `2 + 3`.
Insertion	`[| 2 + ${x} |]`	'inserts' the AST `x` into the AST being created by the quasi-quotes.

Compile-time meta-programming

Code (as trees, not text) is programmatically generated.

Expression	`2 + 3`	evaluates to `5`.
Splice	`$<x>`	evaluates `x` at compile-time; the AST returned overwrites the splice.
Quasi-quote	`[| 2 + 3 |]`	evaluates to a *hygienic* AST representing `2 + 3`.
Insertion	`[| 2 + ${x} |]`	'inserts' the AST `x` into the AST being created by the quasi-quotes.
DSL Block	`$«x»: ...`	passes the text '...' to the function `x` at compile-time.

An example

Where it falls apart

- Parser composition: a mess.

Where it falls apart

- Parser composition: a mess.
- Extension languages second-class citizens.

Parsing composition

Should be easy

Parsing composition

- **LR**
- **Earley**
- **PEG**

Parsing composition

- **LR** composition undefined (in general).
- **Earley**
- **PEG**

Parsing composition

- **LR** composition undefined (in general).
- **Earley** composition ambiguous (in general).
- **PEG**

Parsing composition

- **LR** composition undefined (in general).
- **Earley** composition ambiguous (in general).
- **PEG** composition can shadow (in general).

Where it falls apart (2)

- Parser composition: a mess.
- Extension languages second-class citizens.

Where it falls apart (2)

- Parser composition: a mess.
- Extension languages second-class citizens.
- Text only.

Example (3)

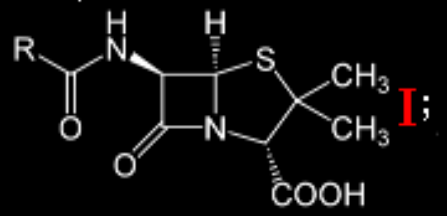
Example (3)

example.fnd - /home/ltratt/

File Edit Search Preferences Shell Macro Windows Help

/home/ltratt/example.fnd L: 24

```
func custom_prescription(Patient p) : Medicine
{
    if (p.penicillin_allergy())
        return NULL;

    Medicine m =  

I;

    candidate = generate(P, m);
    if (!check_with_doctor(candidate))
        return NULL;
    m.set_variable(R, candidate);

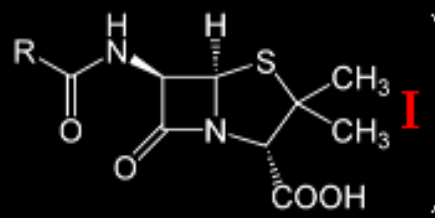
    return m;
}
```

Example (4)

example.fnd - /home/ltratt/

File Edit Search Preferences Shell Macro Windows Help

/home/ltratt/example.fnd L:224

```
func check_all_suitabile(trial_id):
  for patient_id in SELECT pid FROM trial WHERE id=${trial_id}:
    if | SELECT * FROM prescribed                    | > 0:
       | WHERE contains(drug,  )|
      warn("Patient ${patient_id} currently prescribed a "
        "penicillin derived anti-biotic and must be "
        "seen by a specialist before trial begins.")
```

What are our options?

Abandon parsing...

Abandon parsing...

...for SDE?

Language composition editor

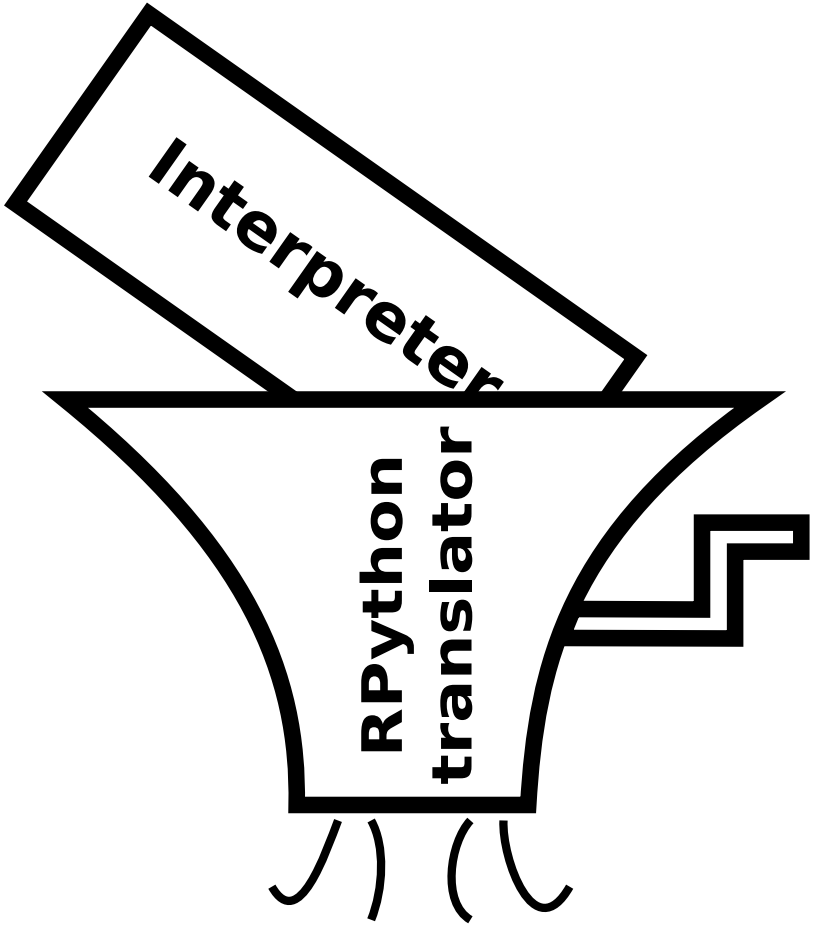
This research graciously funded by Oracle.

Boil down to the JVM

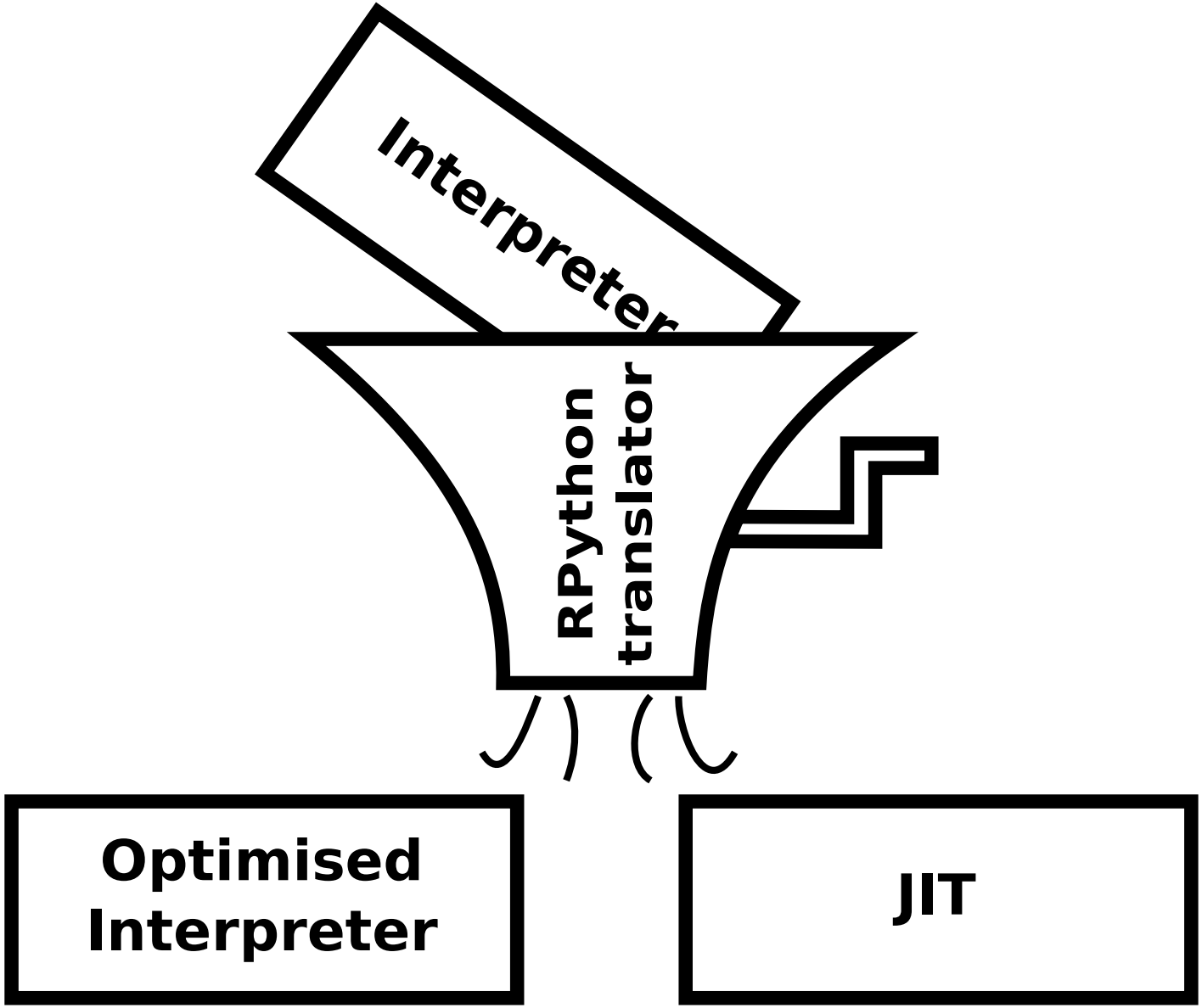
~~Boil down to the JVM~~

Meta-tracing to the rescue

RPython translation



RPython translation



Adding a JIT to an RPython interpreter

```
...
pc := 0
while 1:

    instr := load_next_instruction(pc)
    if instr == POP:
        stack.pop()
        pc += 1
    elif instr == BRANCH:
        off = load_branch_jump(pc)

        pc += off
    elif ...:
        ...
```

Observation: interpreters are big loops.

Adding a JIT to an RPython interpreter

```
...
pc := 0
while 1:
    jit_merge_point(pc)
    instr := load_next_instruction(pc)
    if instr == POP:
        stack.pop()
        pc += 1
    elif instr == BRANCH:
        off = load_branch_jump(pc)
        if off < 0: can_enter_jit(pc)
        pc += off
    elif ...:
        ...
```

Observation: interpreters are big loops.

Tracing JITs

User program (lang *FL*)

```
if x < 0:
  x = x + 1
else:
  x = x + 2
x = x + 3
```

Tracing JITs

User program (lang *FL*)	Trace when x is set to 6
`if x < 0:`	`guard_type(x, int)`
`  x = x + 1`	`guard_not_less_than(x, 0)`
`else:`	`guard_type(x, int)`
`  x = x + 2`	`x = int_add(x, 2)`
`x = x + 3`	`guard_type(x, int)`
	`x = int_add(x, 3)`

Tracing JITs

User program (lang *FL*)	Optimised trace

`if x < 0:`
`  x = x + 1`
`else:`
`  x = x + 2`
`x = x + 3`
	`guard_type(x, int)`
	`guard_not_less_than(x, 0)`
	`x = int_add(x, 5)`

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)		
Effort (man months)		
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	
Effort (man months)		
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)		
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)	18	
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)	18	3
Performance		

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)	18	3
Performance	x	

Converge 1 vs. Converge 2 VMs

	Converge 1	Converge 2
Size (KLoc)	13	5.5
Effort (man months)	18	3
Performance	x	2-150x

Dhrystone benchmark

	50000	5000000
C (GCC 4.6.3)	0.004 ± 0.002	0.179 ± 0.010
HotSpot (1.7.0_09)	0.107 ± 0.006	0.240 ± 0.010
Converge1 (git #68c795d2be)	2.053 ± 0.029	207.274 ± 3.048
Converge2 (2.0)	0.118 ± 0.004	1.914 ± 0.022
Lua (5.2.1)	0.201 ± 0.008	19.417 ± 0.474
LuaJIT2 (2.0.0)	0.014 ± 0.006	0.879 ± 0.016
CPython (2.7.3)	0.368 ± 0.010	35.072 ± 0.537
Jython (2.5.3)	1.820 ± 0.029	28.432 ± 0.466
PyPy–nonopt (1.9*)	0.127 ± 0.006	5.898 ± 0.071
PyPy (1.9)	0.069 ± 0.008	1.085 ± 0.014
Ruby (1.9.3-p327)	0.312 ± 0.008	29.819 ± 0.257
JRuby (1.7.1)	2.050 ± 0.039	10.576 ± 0.304

Richards benchmark

	10	100
C (GCC 4.6.3)	0.012 ± 0.006	0.079 ± 0.006
HotSpot (1.7.0_09)	0.109 ± 0.010	0.169 ± 0.014
Converge1 (git #68c795d2be)	9.931 ± 0.102	100.216 ± 1.356
Converge2 (2.0)	0.637 ± 0.006	2.850 ± 0.014
Lua (5.2.1)	0.665 ± 0.024	6.574 ± 0.139
LuaJIT2 (2.0.0)	0.085 ± 0.006	0.763 ± 0.010
CPython (2.7.3)	1.585 ± 0.022	15.698 ± 0.227
Jython (2.5.3)	2.820 ± 0.069	13.870 ± 0.345
PyPy–nonopt (1.9*)	0.515 ± 0.010	2.839 ± 0.016
PyPy (1.9)	0.267 ± 0.006	0.544 ± 0.008
Ruby (1.9.3-p327)	0.793 ± 0.018	7.159 ± 0.061
JRuby (1.7.1)	2.130 ± 0.025	3.640 ± 0.053

Fannkuch-redux benchmark

	10		11	
C (GCC 4.6.3)	0.163	± 0.006	1.992	± 0.010
HotSpot (1.7.0_09)	0.350	± 0.008	3.448	± 0.029
Converge1 (git #68c795d2be)	†		†	
Converge2 (2.0)	2.658	± 0.041	33.484	± 0.517
Lua (5.2.1)	7.683	± 0.321	100.536	± 2.475
LuaJIT2 (2.0.0)	0.339	± 0.008	4.180	± 0.010
CPython (2.7.3)	9.167	± 0.237	114.001	± 2.189
Jython (2.5.3)	7.776	± 0.419	76.069	± 4.753
PyPy–nonopt (1.9*)	1.402	± 0.022	16.989	± 0.220
PyPy (1.9)	1.256	± 0.024	15.239	± 0.223
Ruby (1.9.3-p327)	13.152	± 0.200	172.098	± 2.168
JRuby (1.7.1)	6.313	± 0.127	61.934	± 1.513

Meta-tracing composition

- Composition of interpreters is feasible.

Meta-tracing composition

- Composition of interpreters is feasible.
- Challenges:

Meta-tracing composition

- Composition of interpreters is feasible.
- Challenges:
  1. Isolation.

Meta-tracing composition

- Composition of interpreters is feasible.
- Challenges:
  1. Isolation.
  2. Communication.

Meta-tracing composition

- Composition of interpreters is feasible.
- Challenges:
  1. Isolation.
  2. Communication.
  3. Performance.

Meta-tracing composition

- Composition of interpreters is feasible.
- Challenges:
  1. Isolation.
  2. Communication.
  3. Performance.
- EPSRC ‘Cooler’ project starting June 2013.

Put it together

Compose:

- parsers
- virtual machines

Put it together

Compose:

- parsers *Incremental parsing*
- virtual machines

Put it together

Compose:

- parsers *Incremental parsing*
- virtual machines *Meta-tracing*

Summary

Summary

- The status quo needn't be so.

Summary

- The status quo needn't be so.
- Language composition might offer a way forward.

Summary

- The status quo needn't be so.
- Language composition might offer a way forward.
- We're not very good at it yet.

Summary

- The status quo needn't be so.
- Language composition might offer a way forward.
- We're not very good at it yet.
- Incremental parsing and meta-tracing *might* save us.

Further reading

- *Parsing: the solved problem that isn't*, Tratt
- The impact of meta-tracing on VM design and implementation, Bolz, Tratt
- Converge: http://convergepl.org/

Thank you for listening