Auto-Mobiles

Optimising Message-Passing Concurrency

Neil Brown

School of Computing
University of Kent
UK

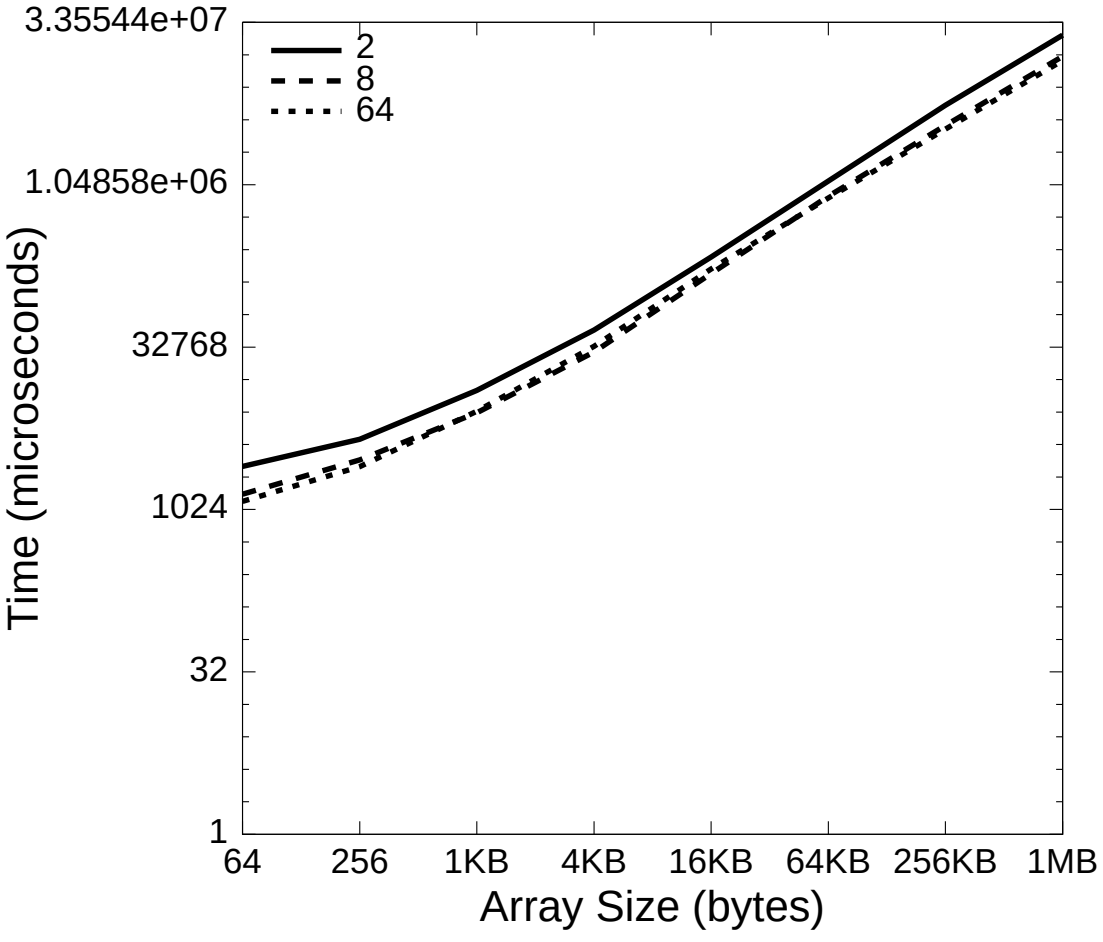
2 November 2009

University of Kent | Computing

Auto-Mobiles

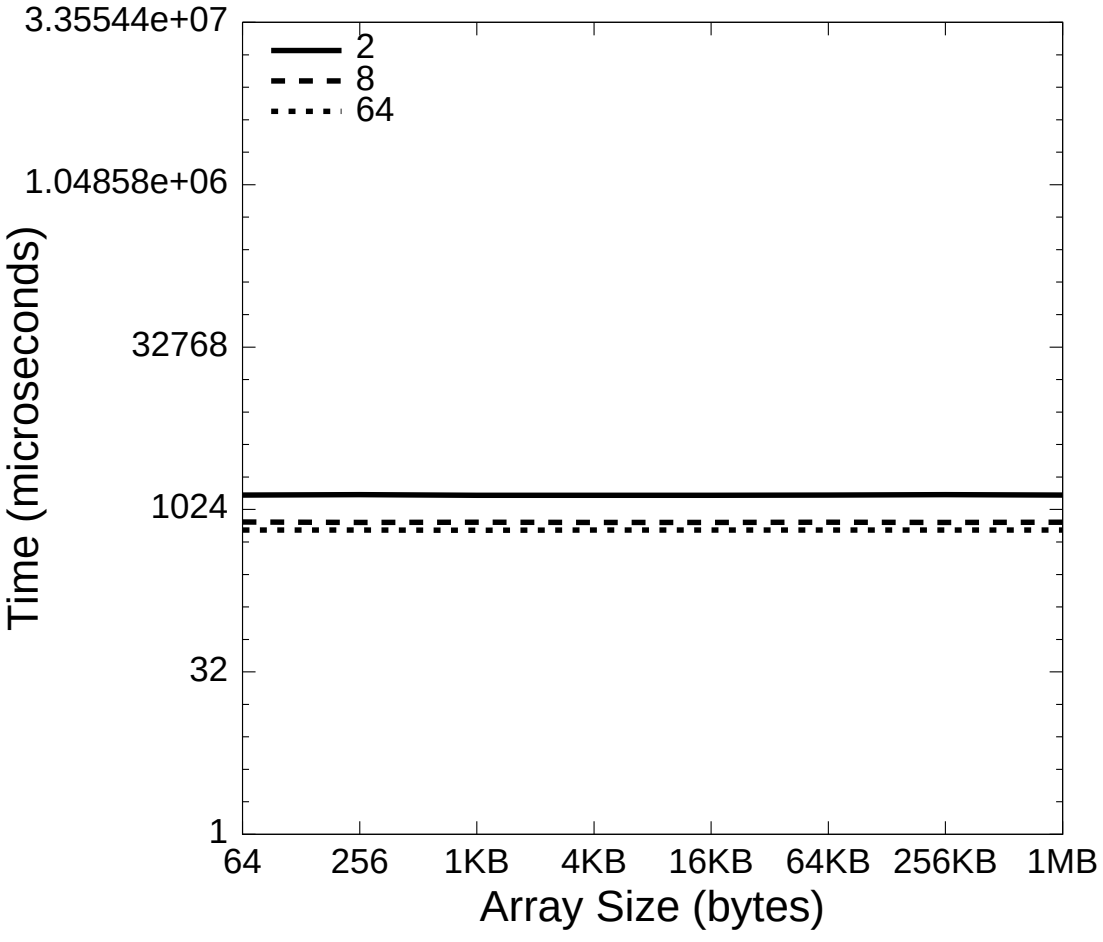
- Compiler Optimisation
  - For message-passing languages like occam 2
  - Increases speed
  - Reduces memory use
- No change to your code required

Best Case



University of Kent | Computing

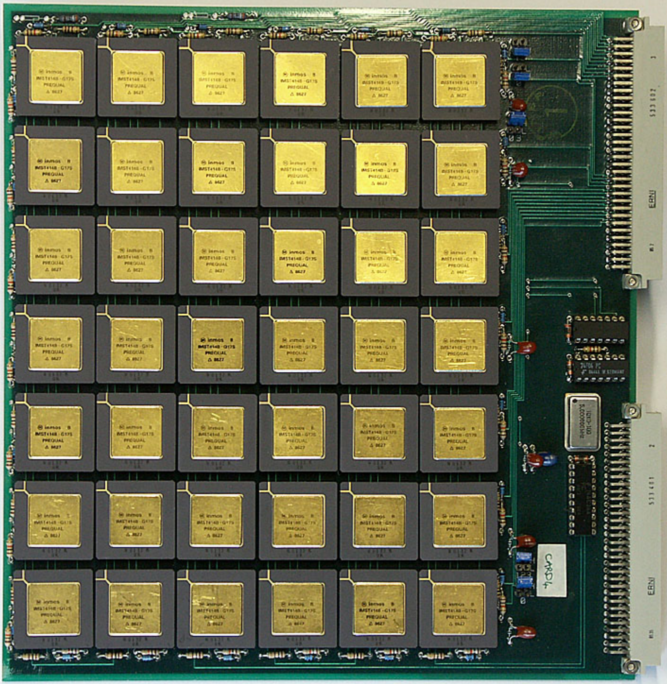
Best Case

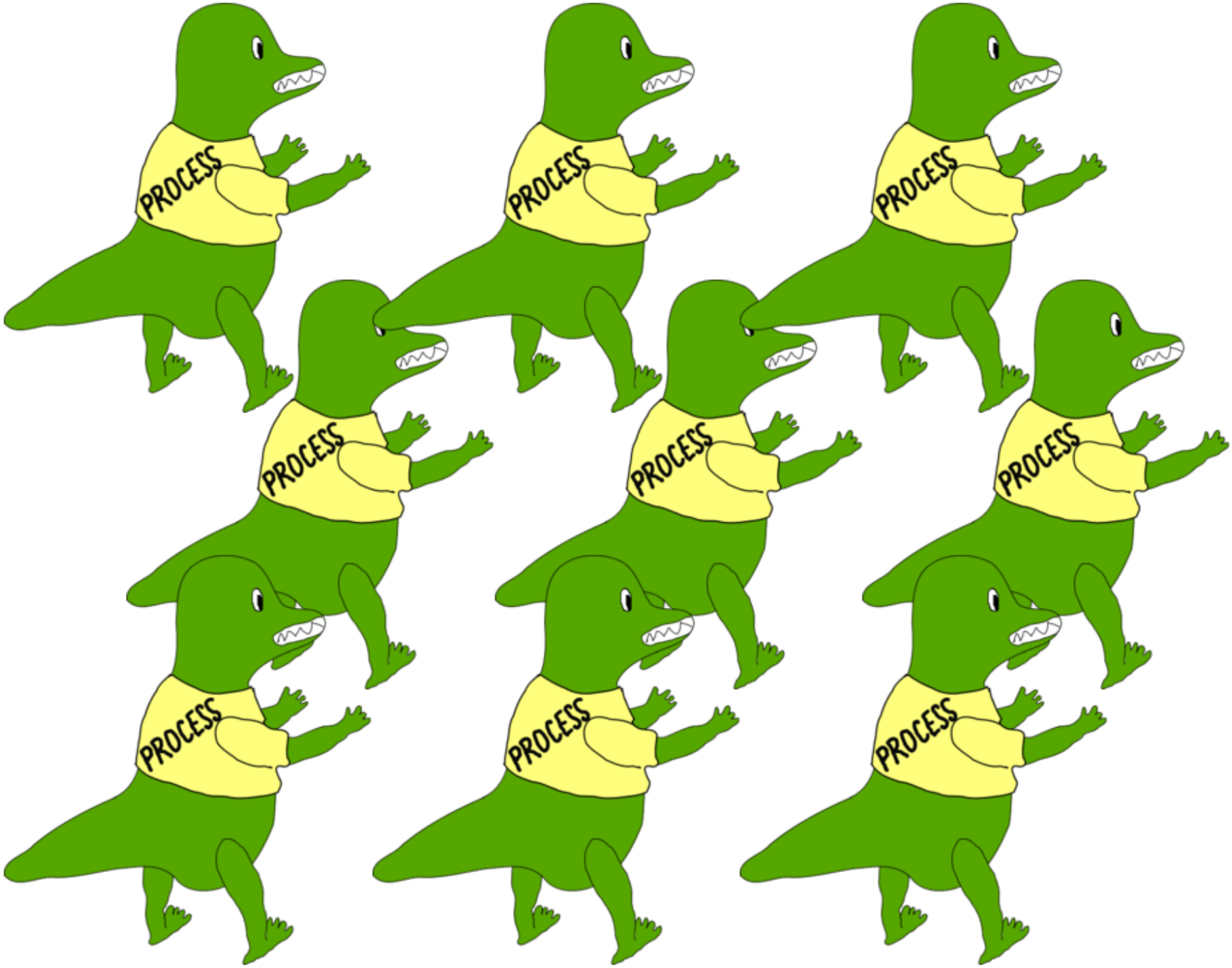


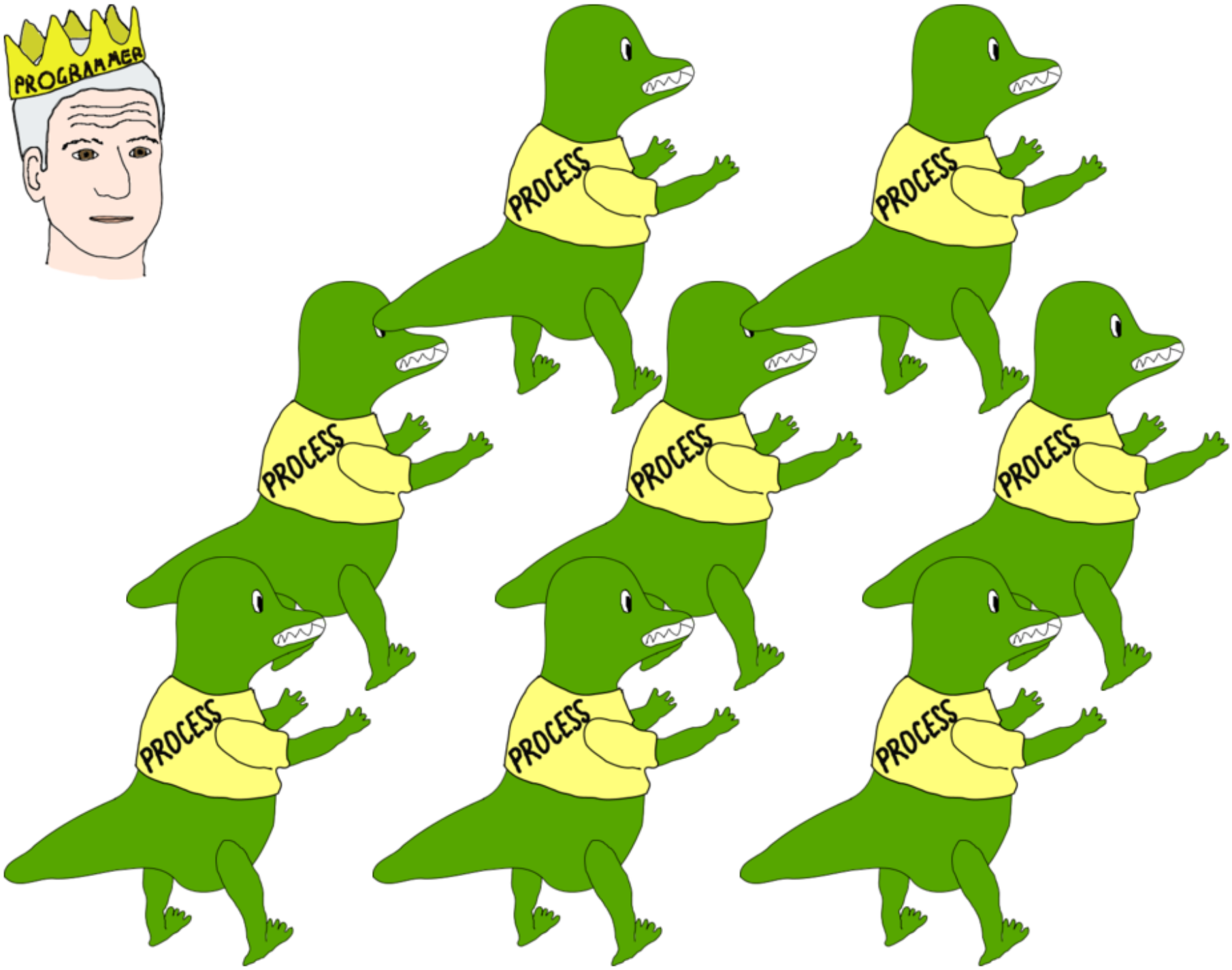
University of Kent | Computing

Auto-Mobiles

- Compiler Optimisation
  - For message-passing languages like occam 2
  - Increases speed
  - Reduces memory use
- No change to your code required

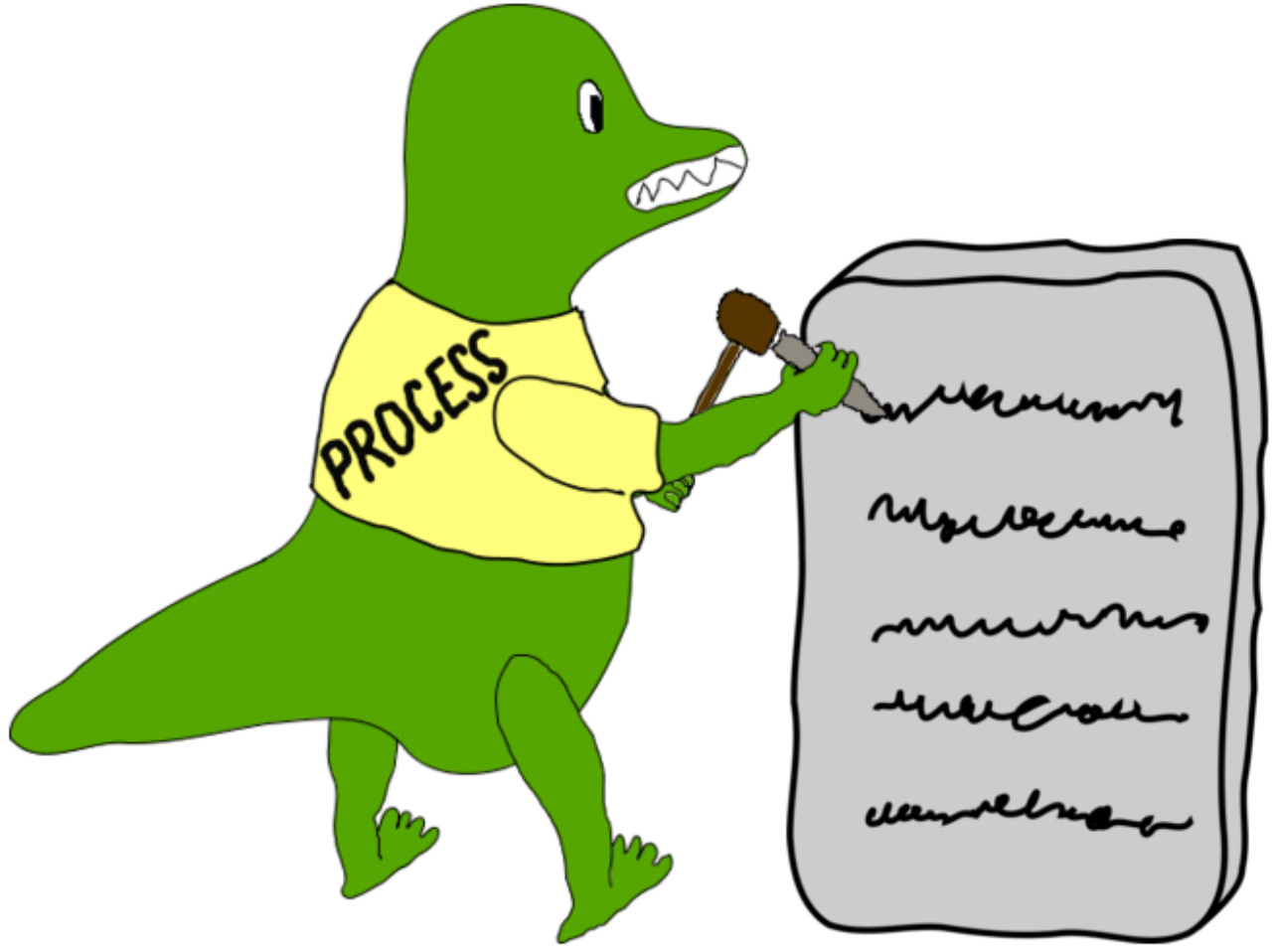

inmos
PREQUAL
ERNI


PROCESS
PROCESS
PROCESS
PROCESS
PROCESS
PROCESS
PROCESS
PROCESS
PROCESS

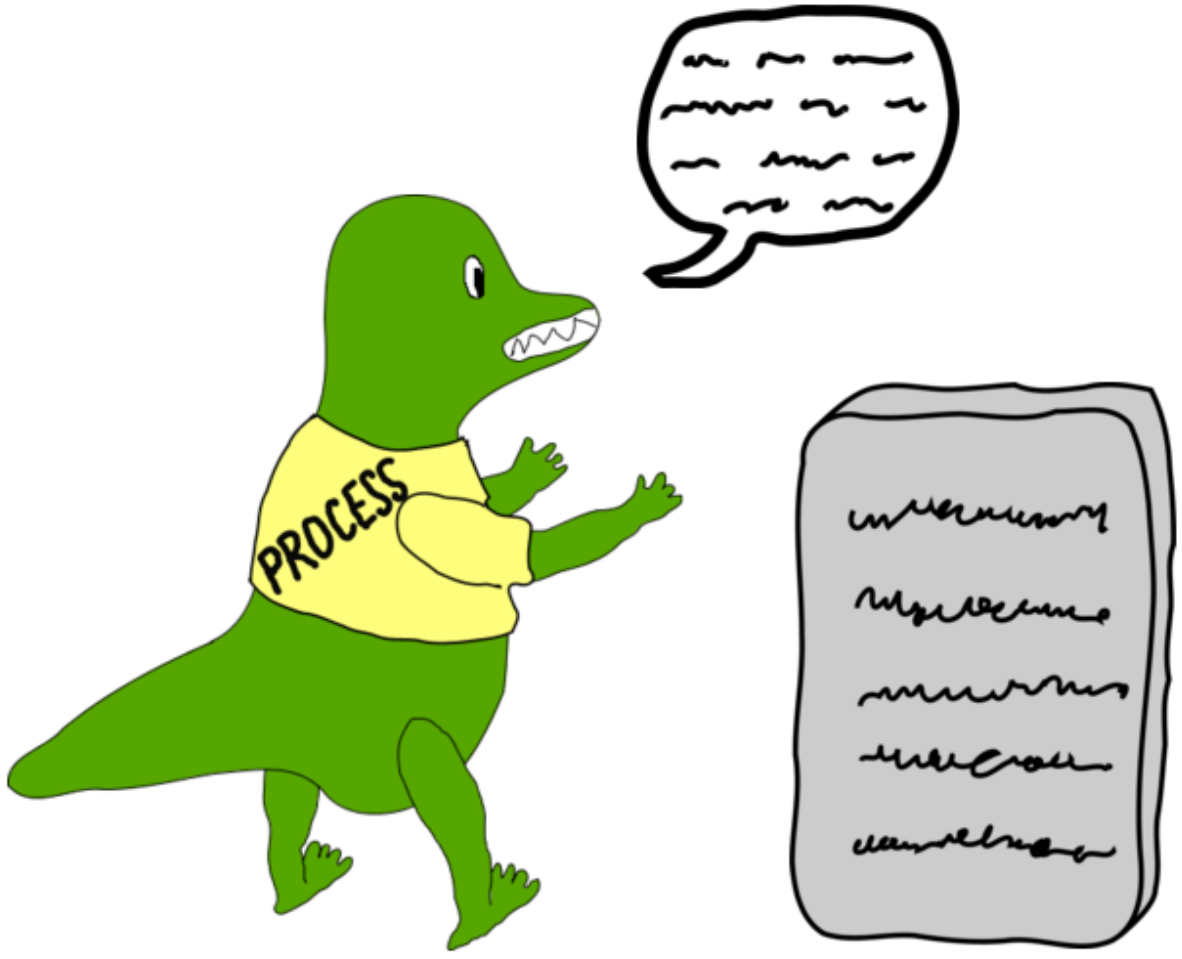

PROGRAMMER
PROCESS
PROCESS
PROCESS
PROCESS
PROCESS
PROCESS
PROCESS

Data Operations

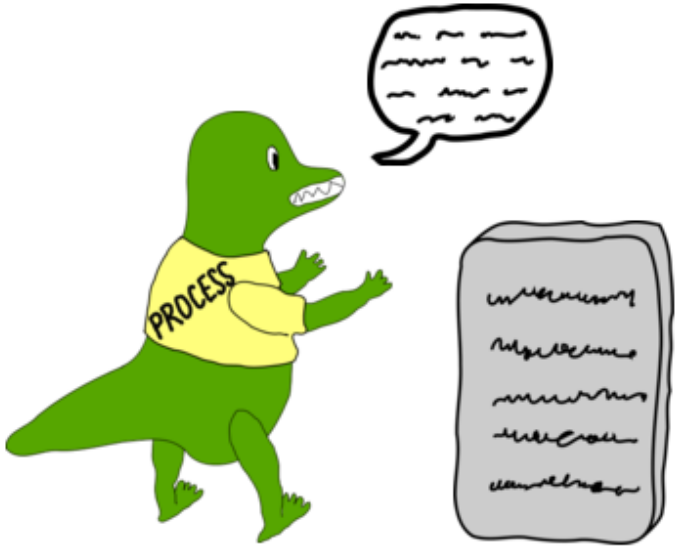
Write



Read



Communication



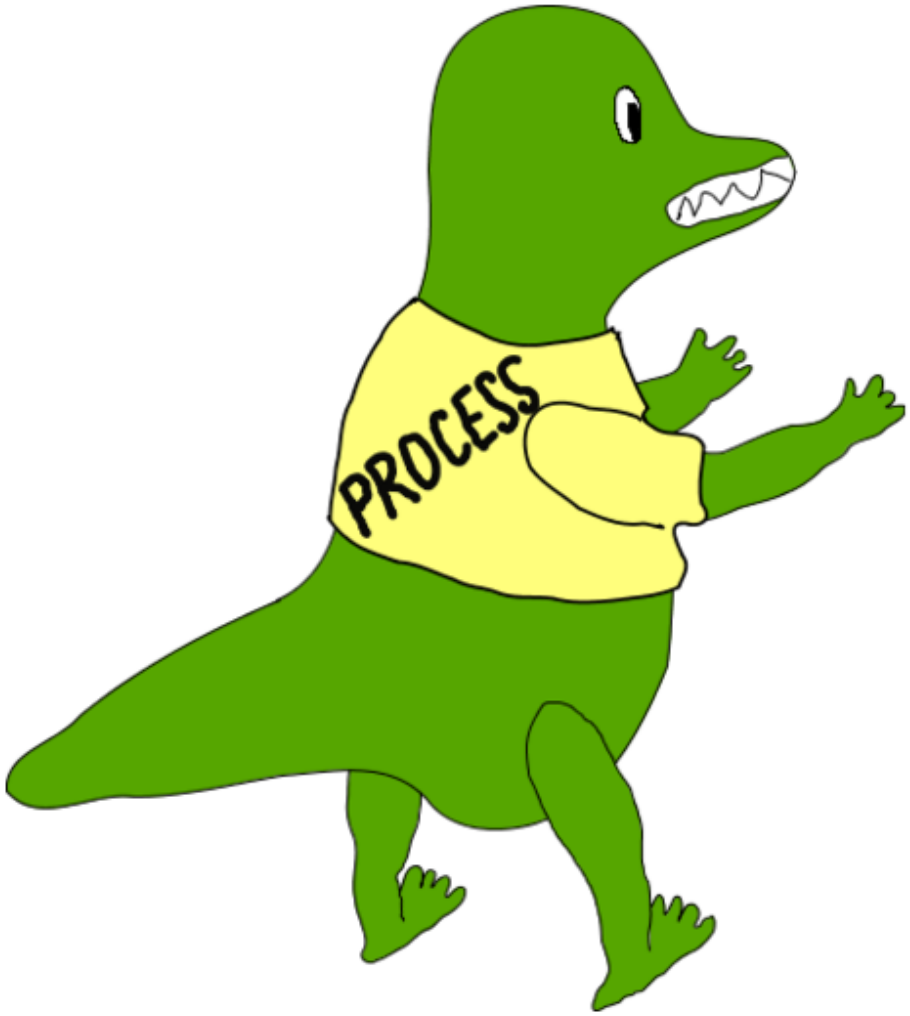
Reading

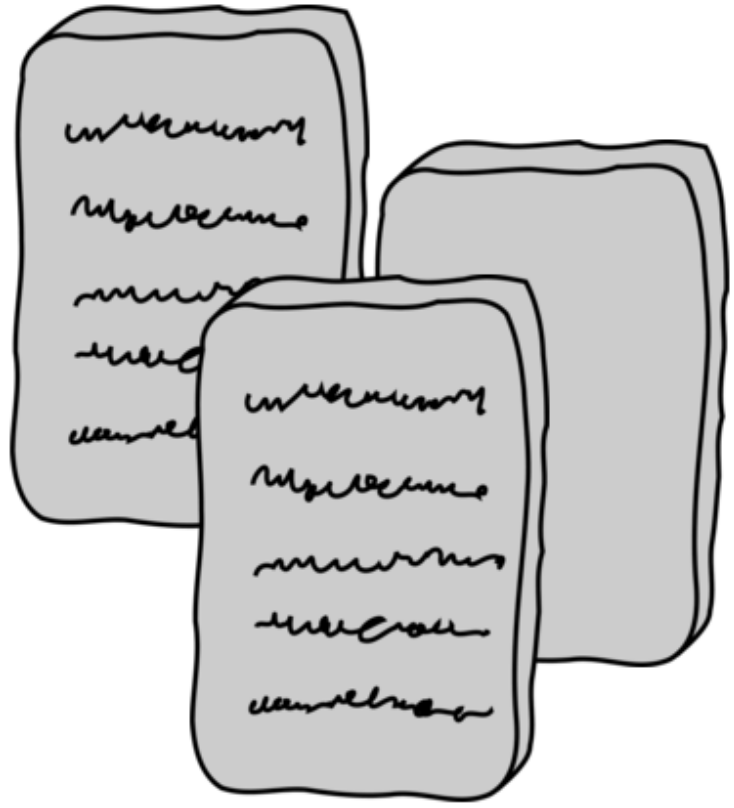
Writing

Communication

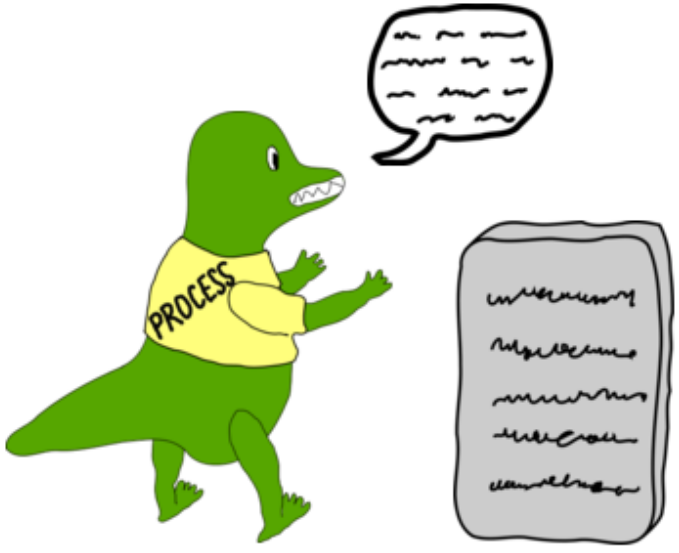
Problems with occam 2

Statically Sized Storage





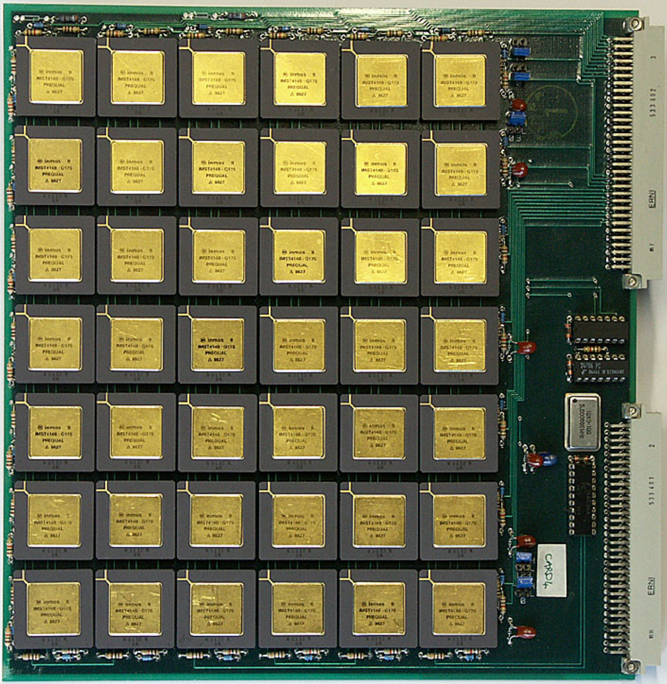
Communication



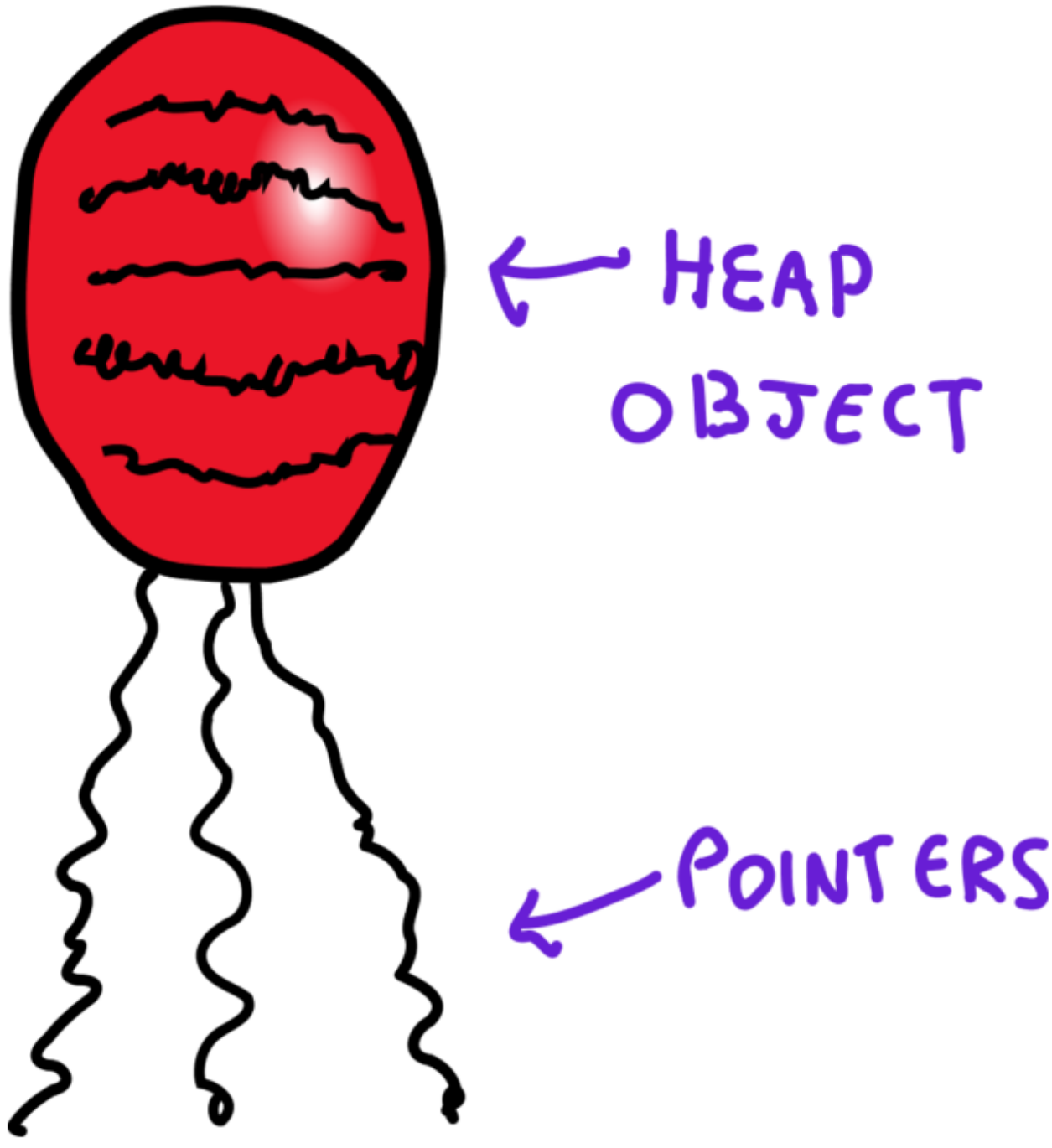
Slow for Large Data

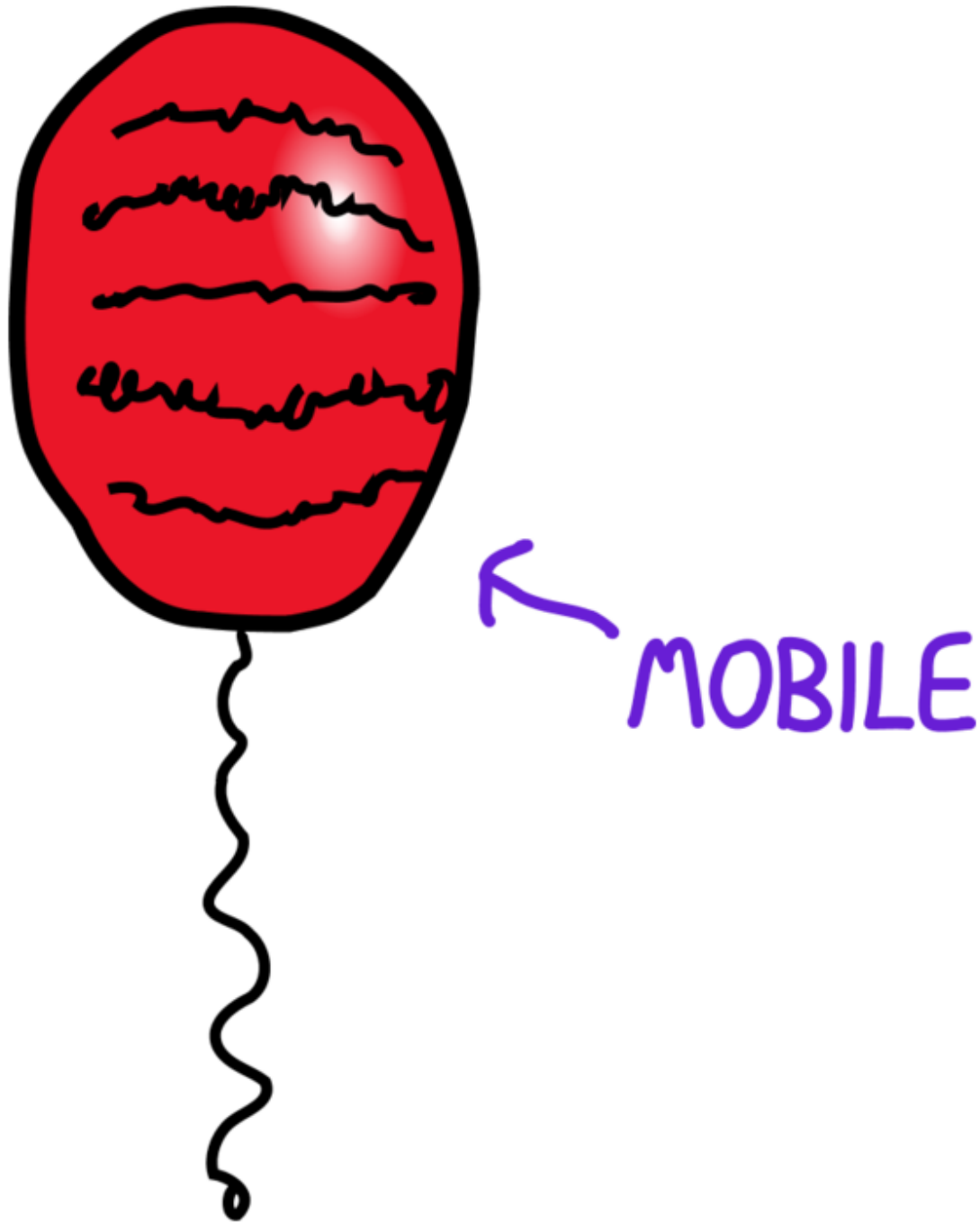
Slow for Large Data

Slow for Large Data

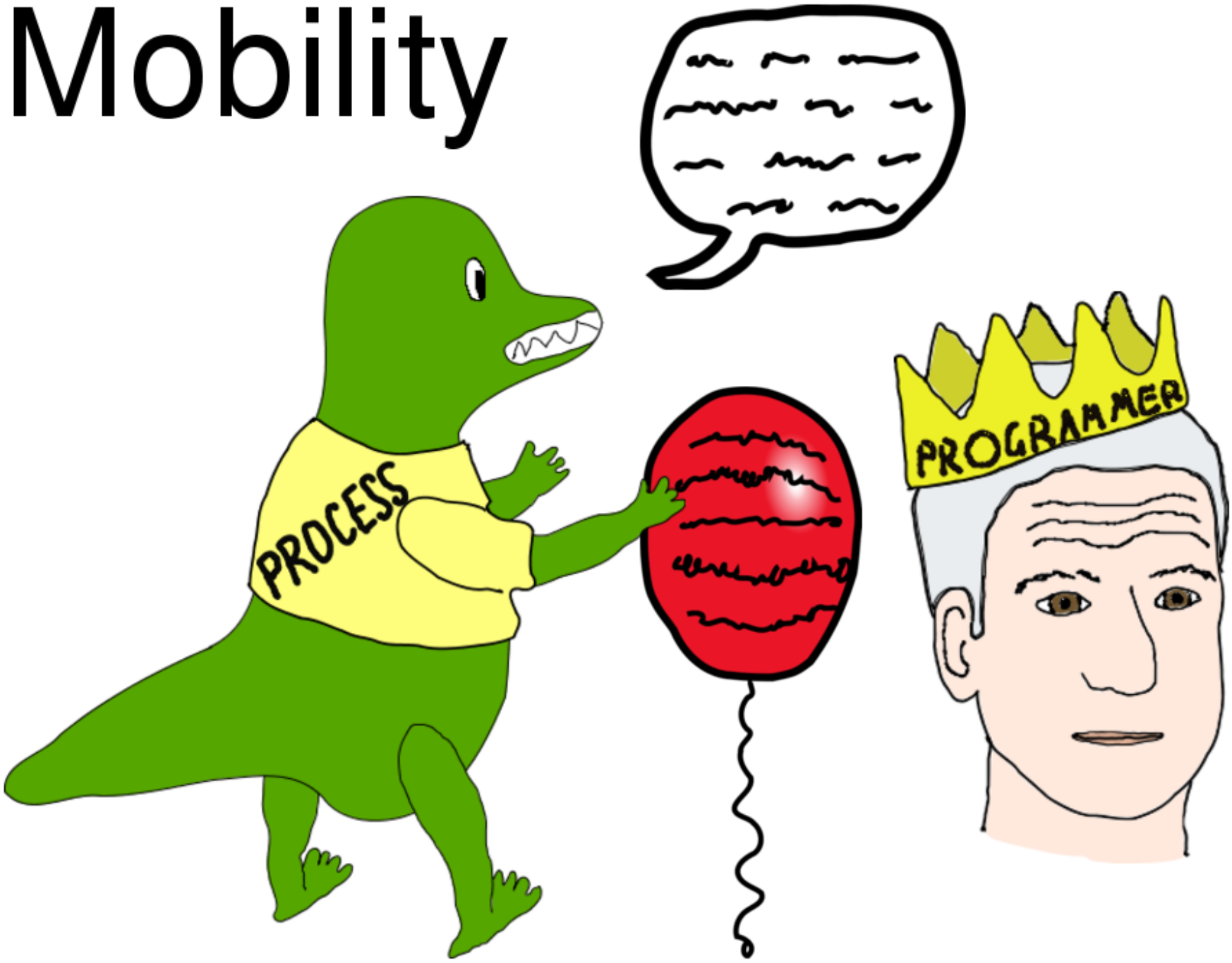

inmos
IMST414B-G17S
PREQUAL
8627
ERNI
CARD4

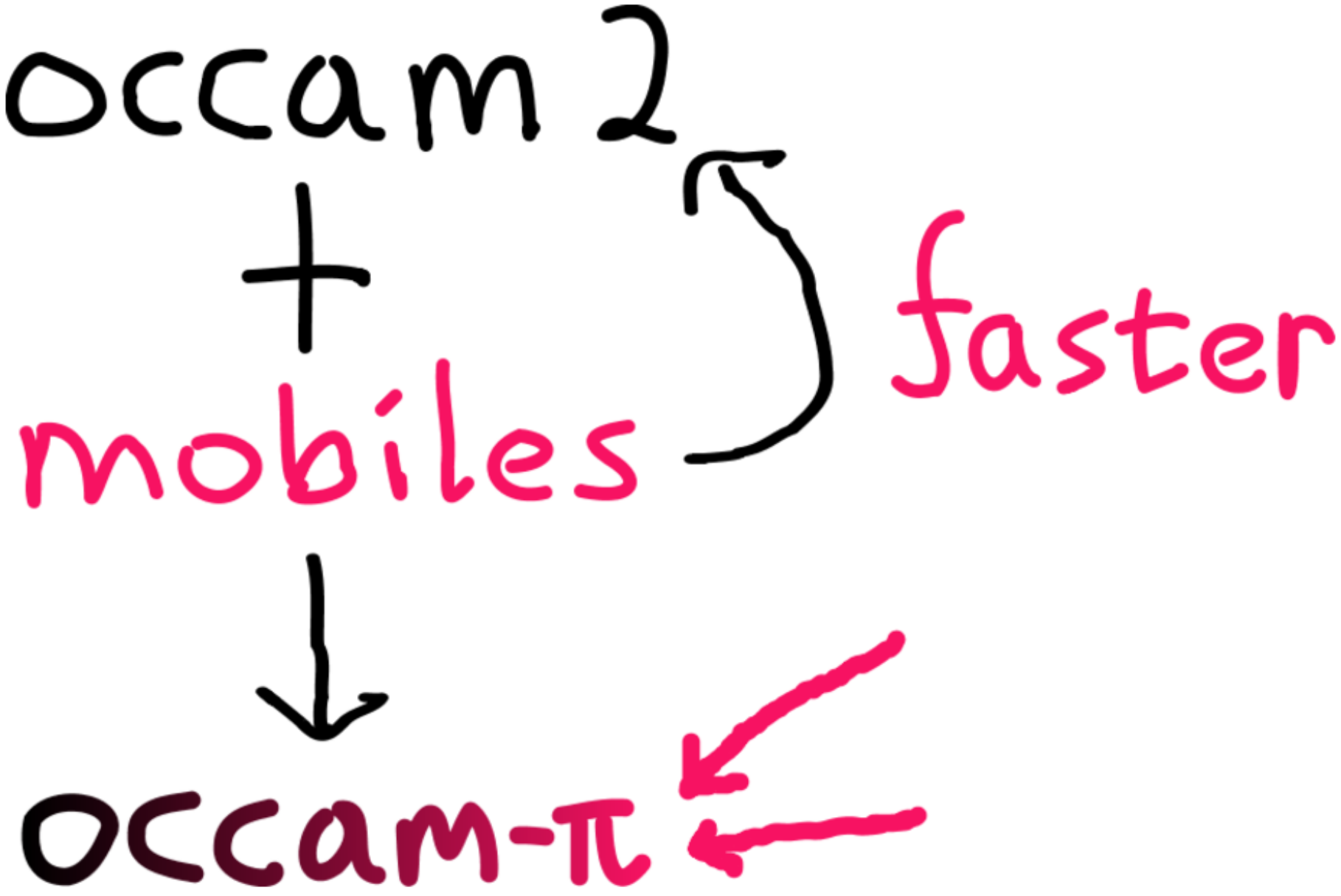

intel
Core™2 Duo
intel

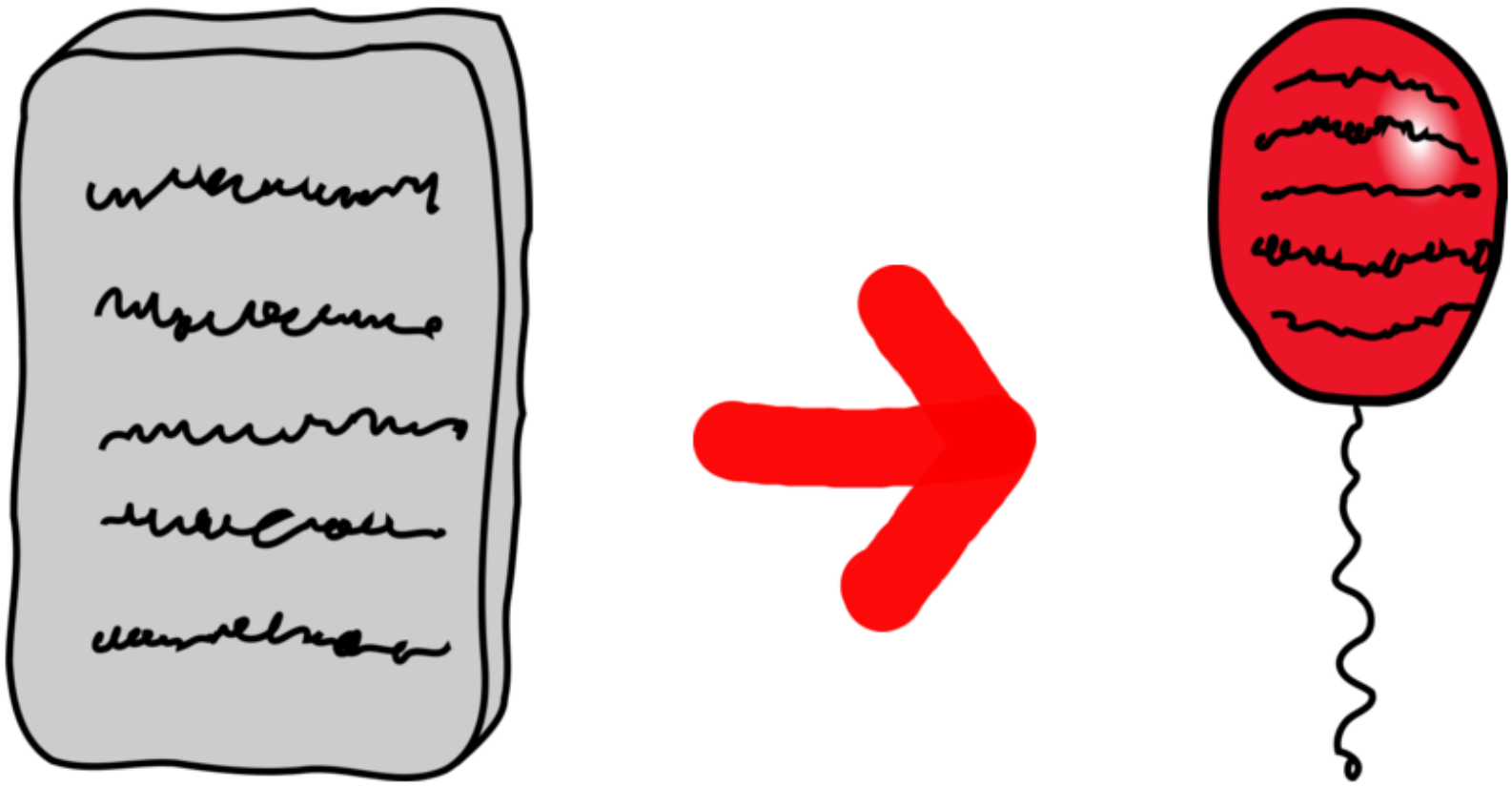

HEAP OBJECT
POINTERS


MOBILE

Mobility




occam 2
+
mobiles
faster
occam-π

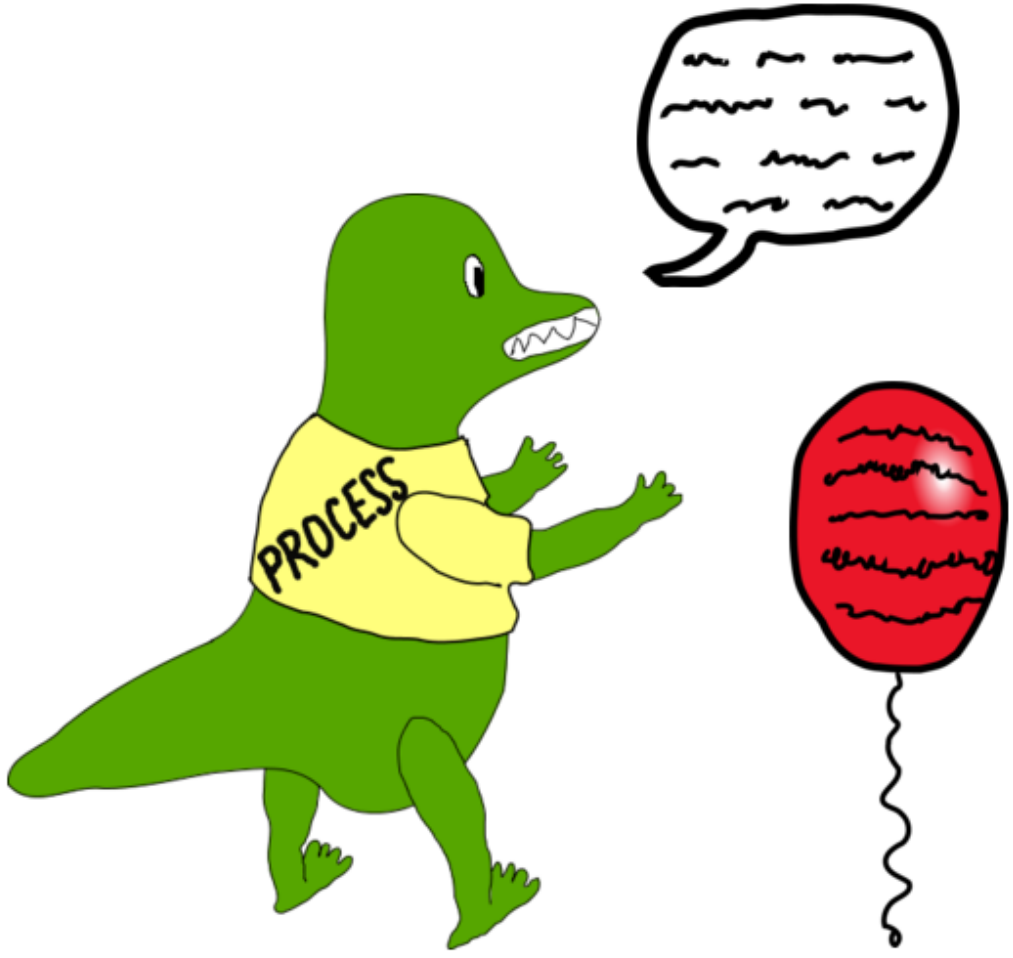


Reading

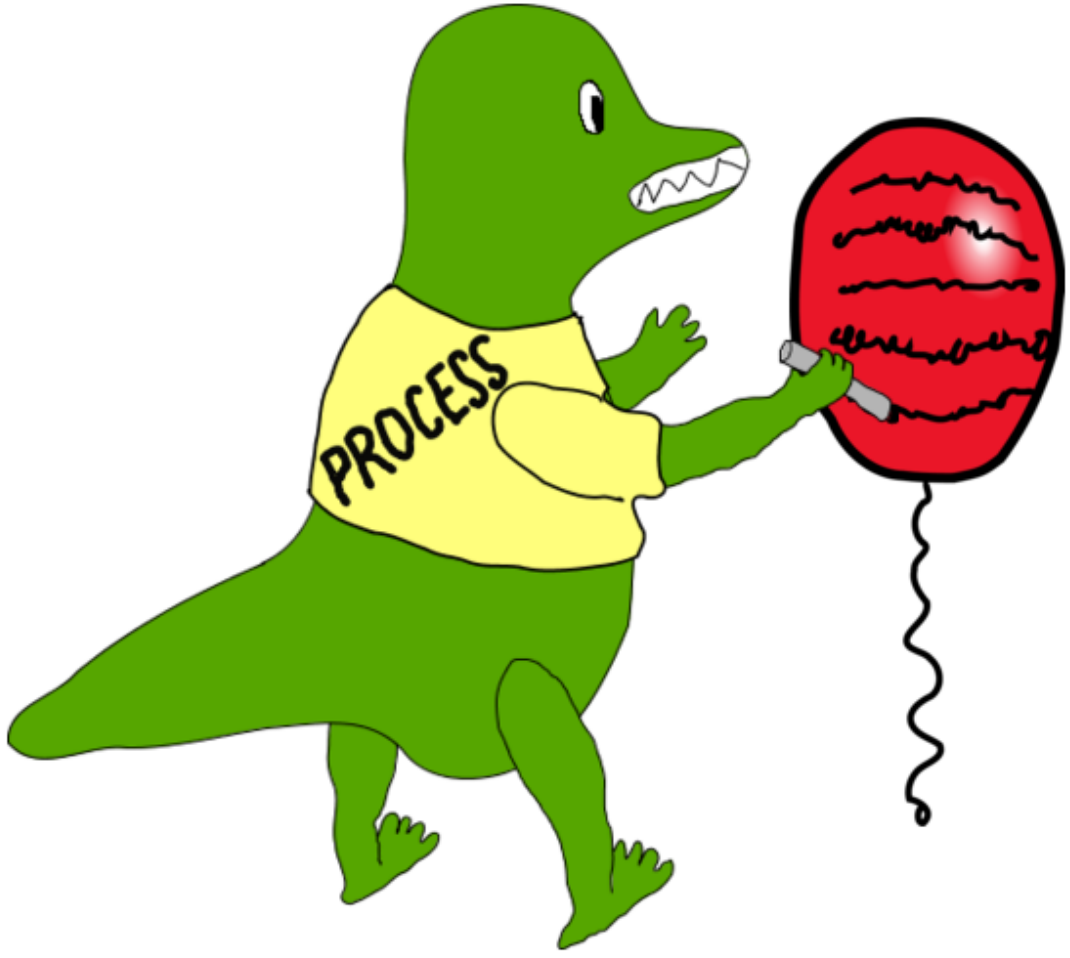
Writing

Communication

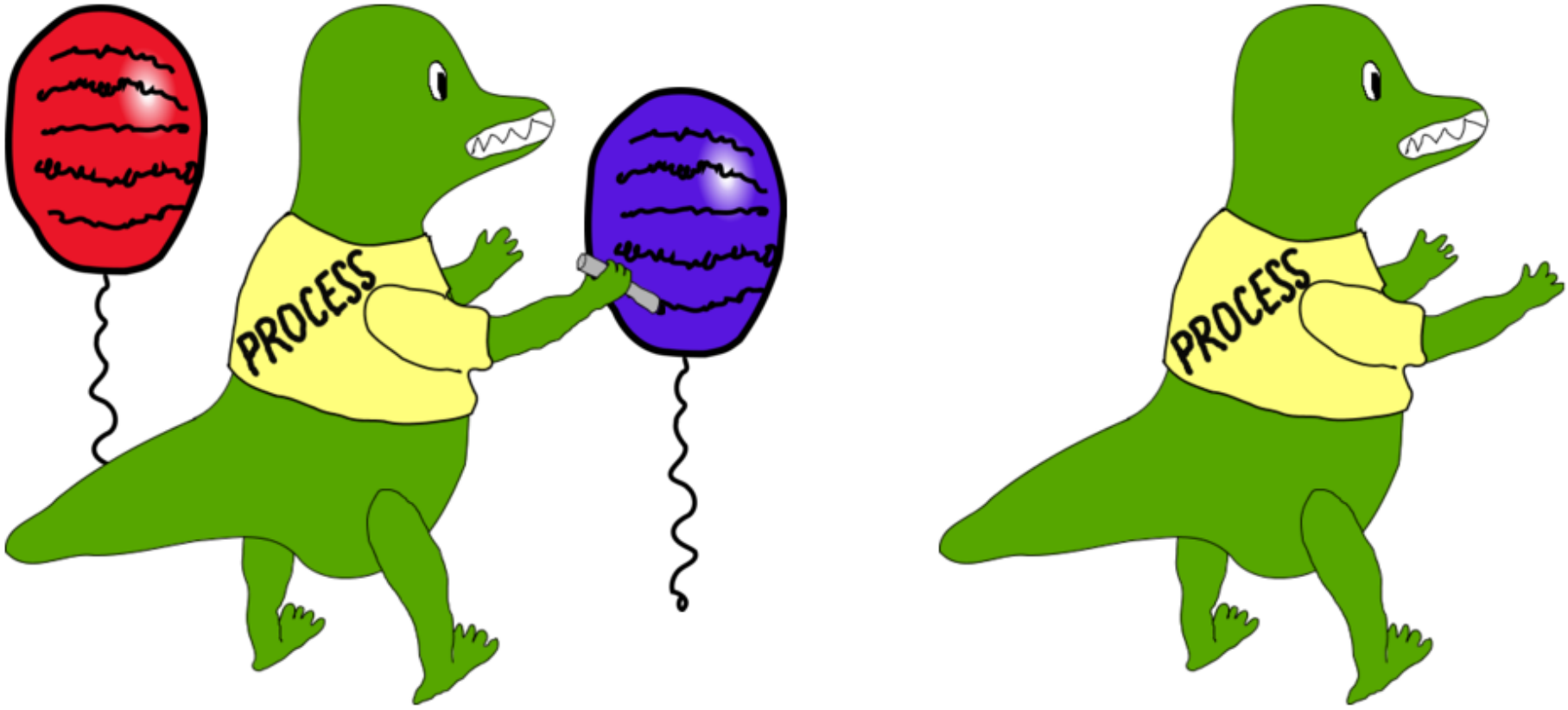
Read



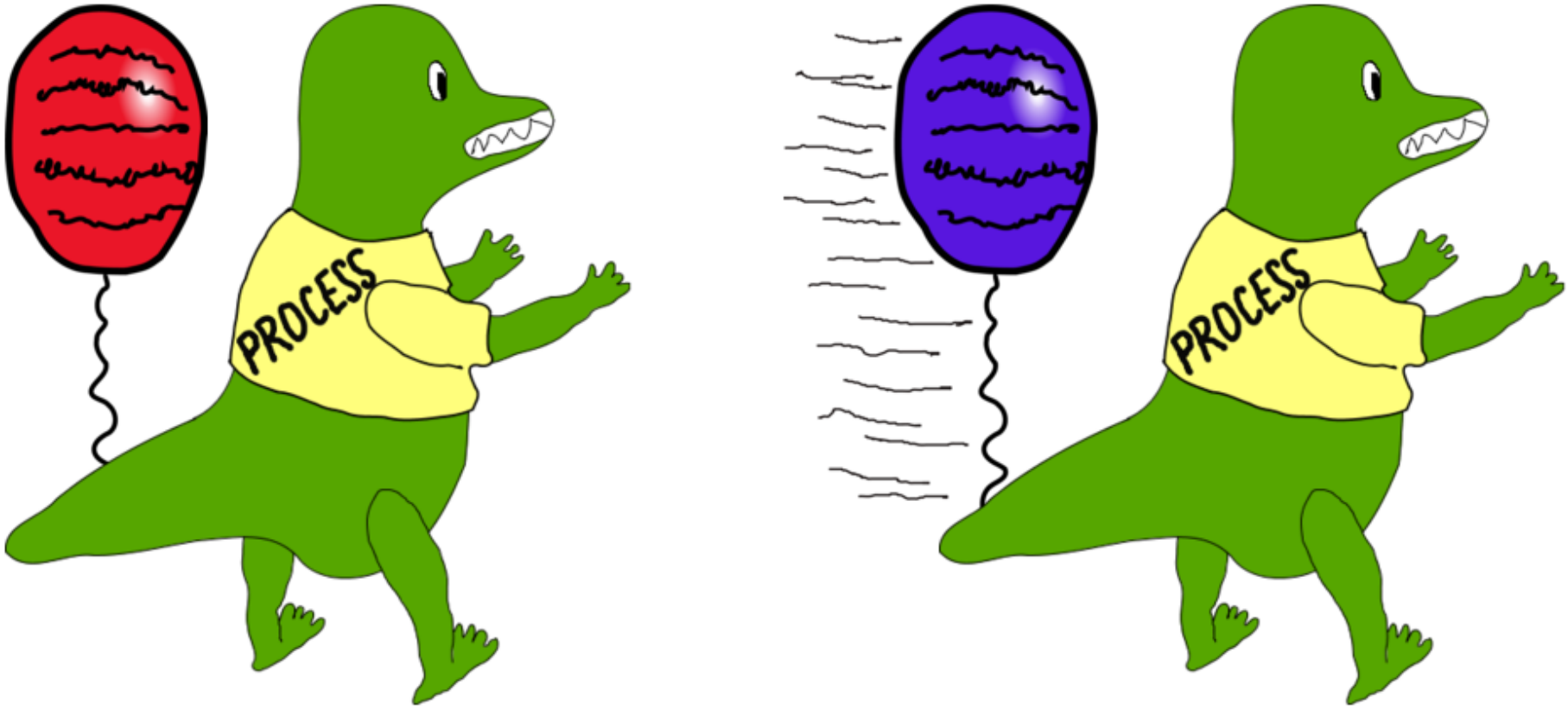
Write



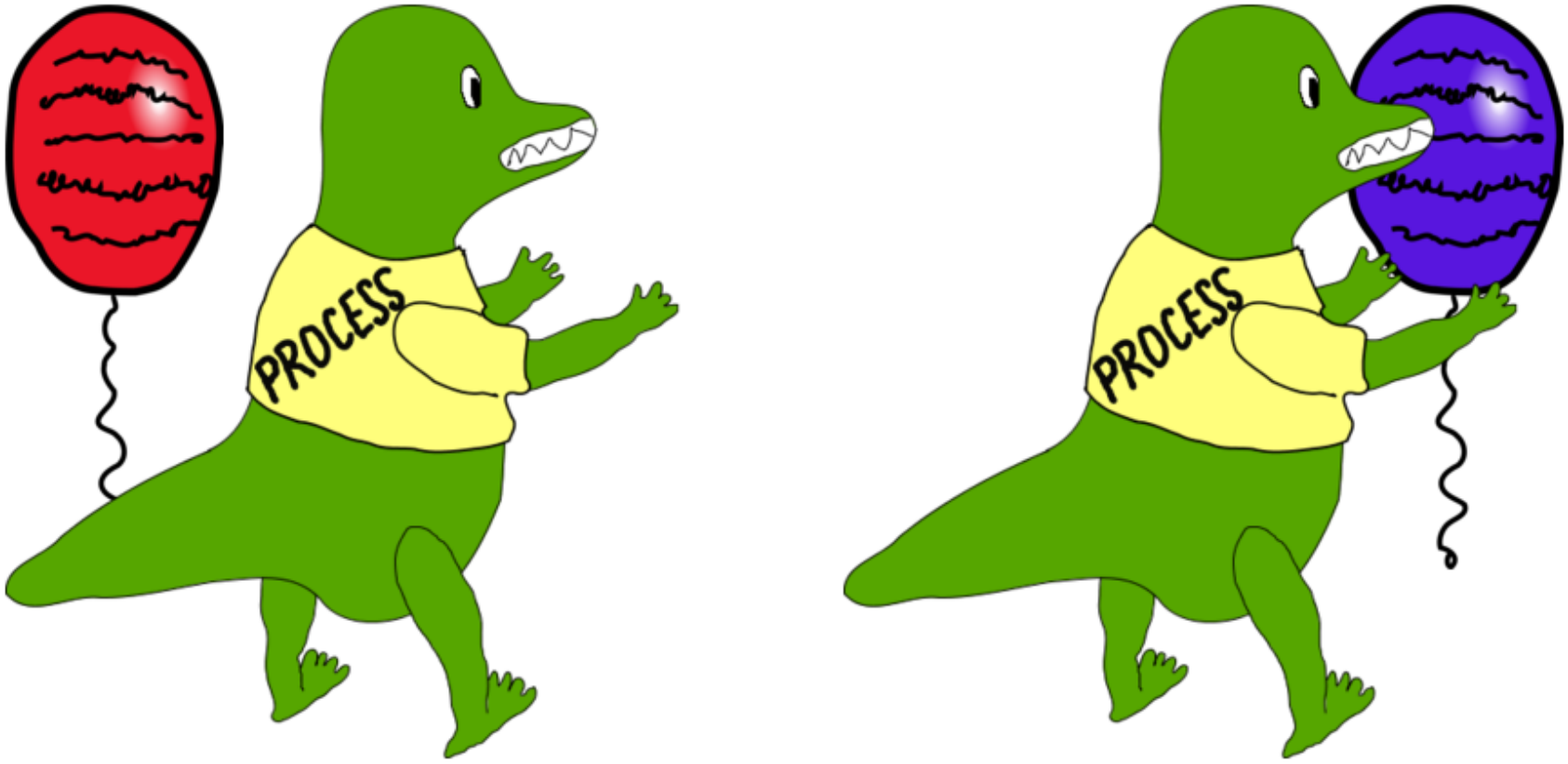
Communication



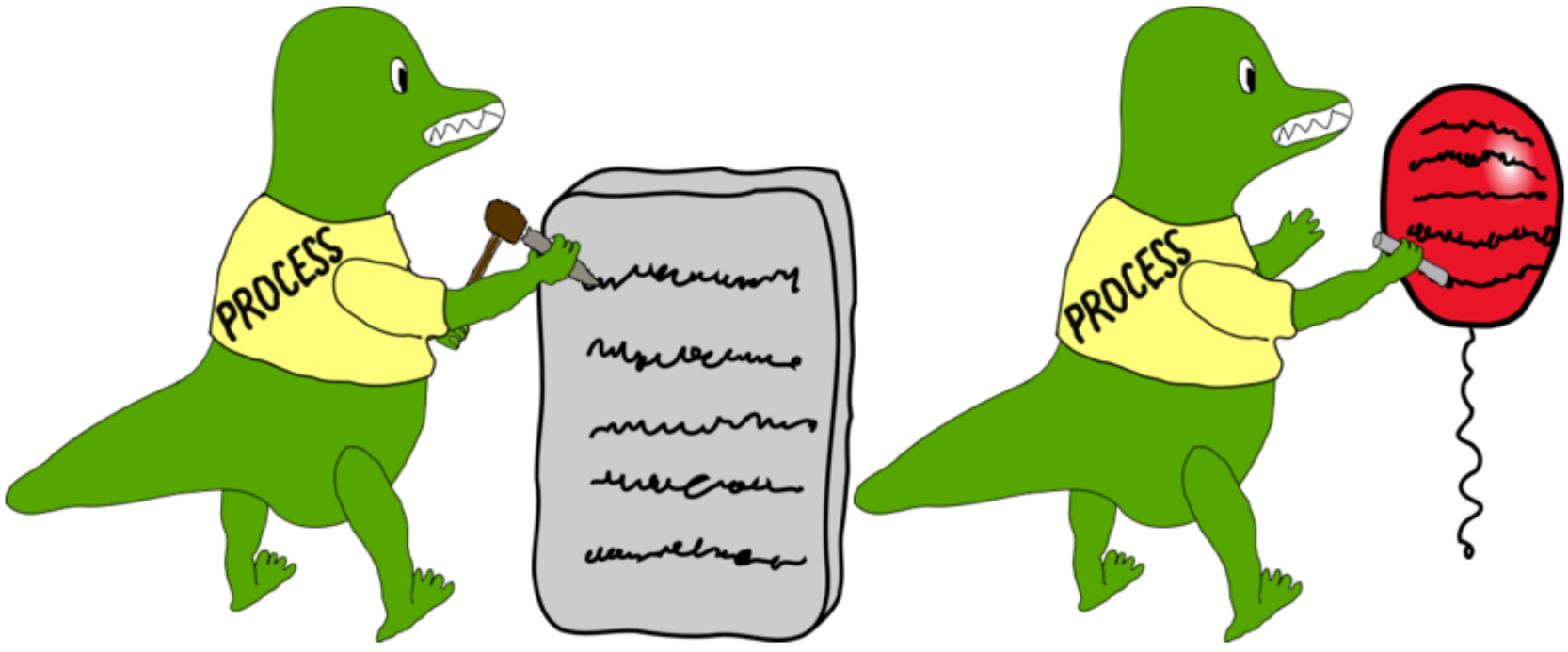
Communication



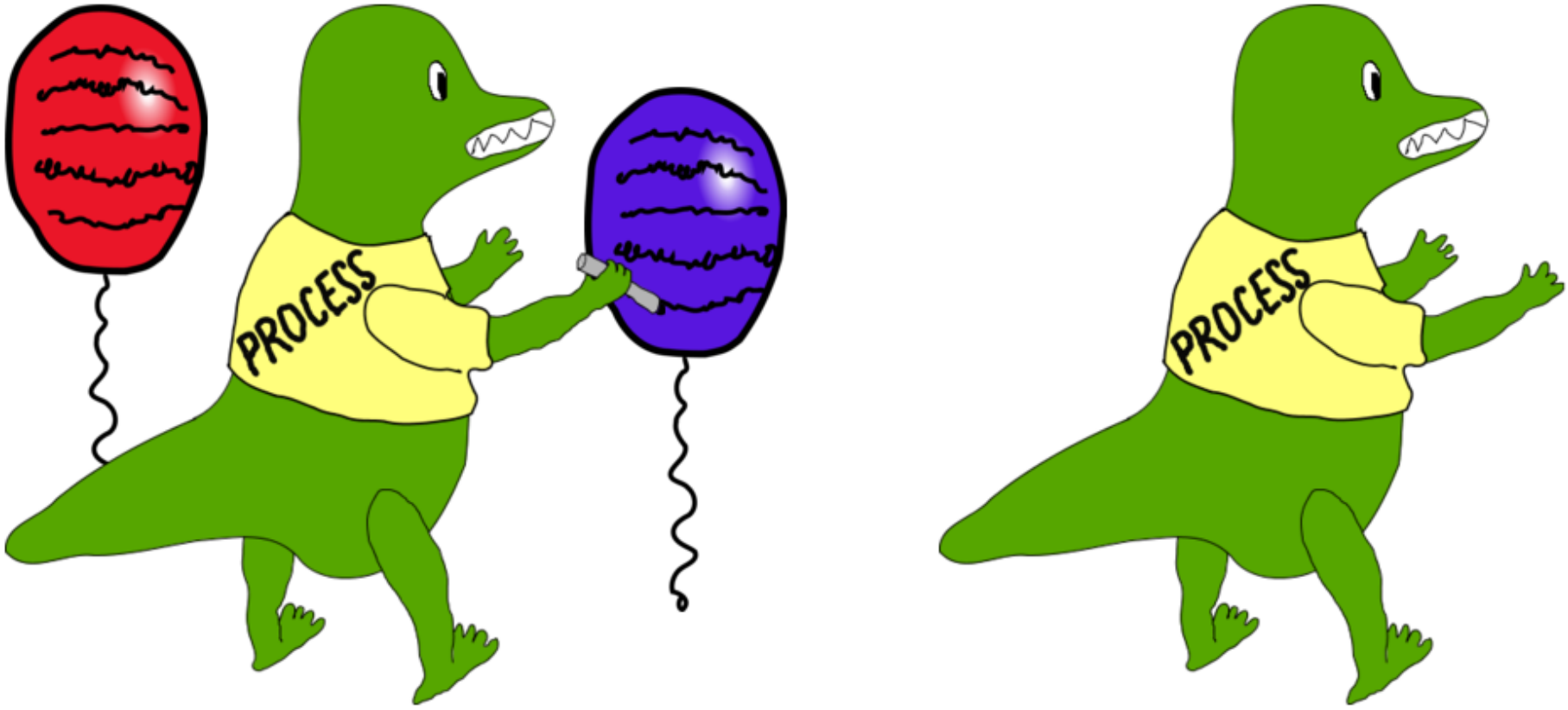
Communication



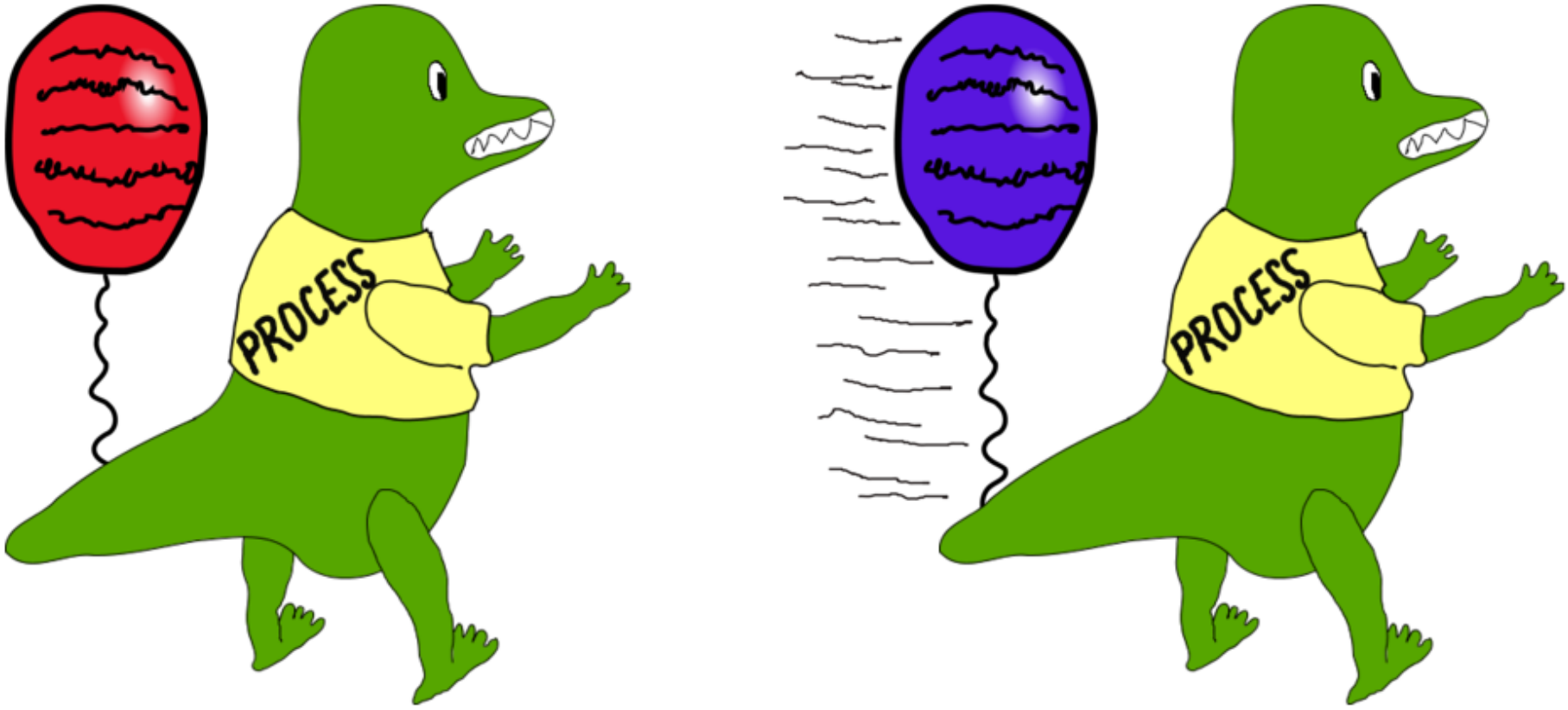
Nothing Gained (Yet)



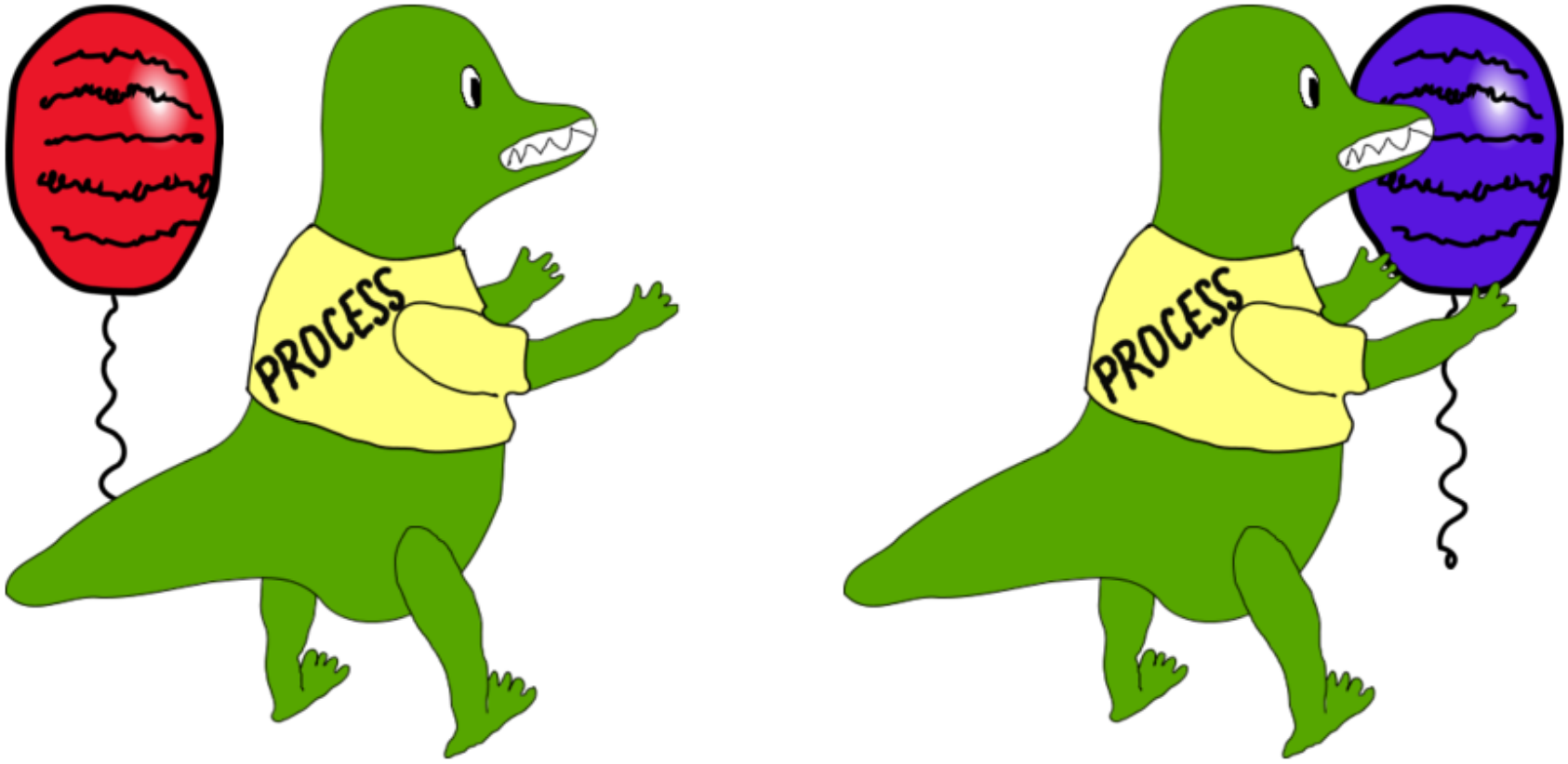
Communication



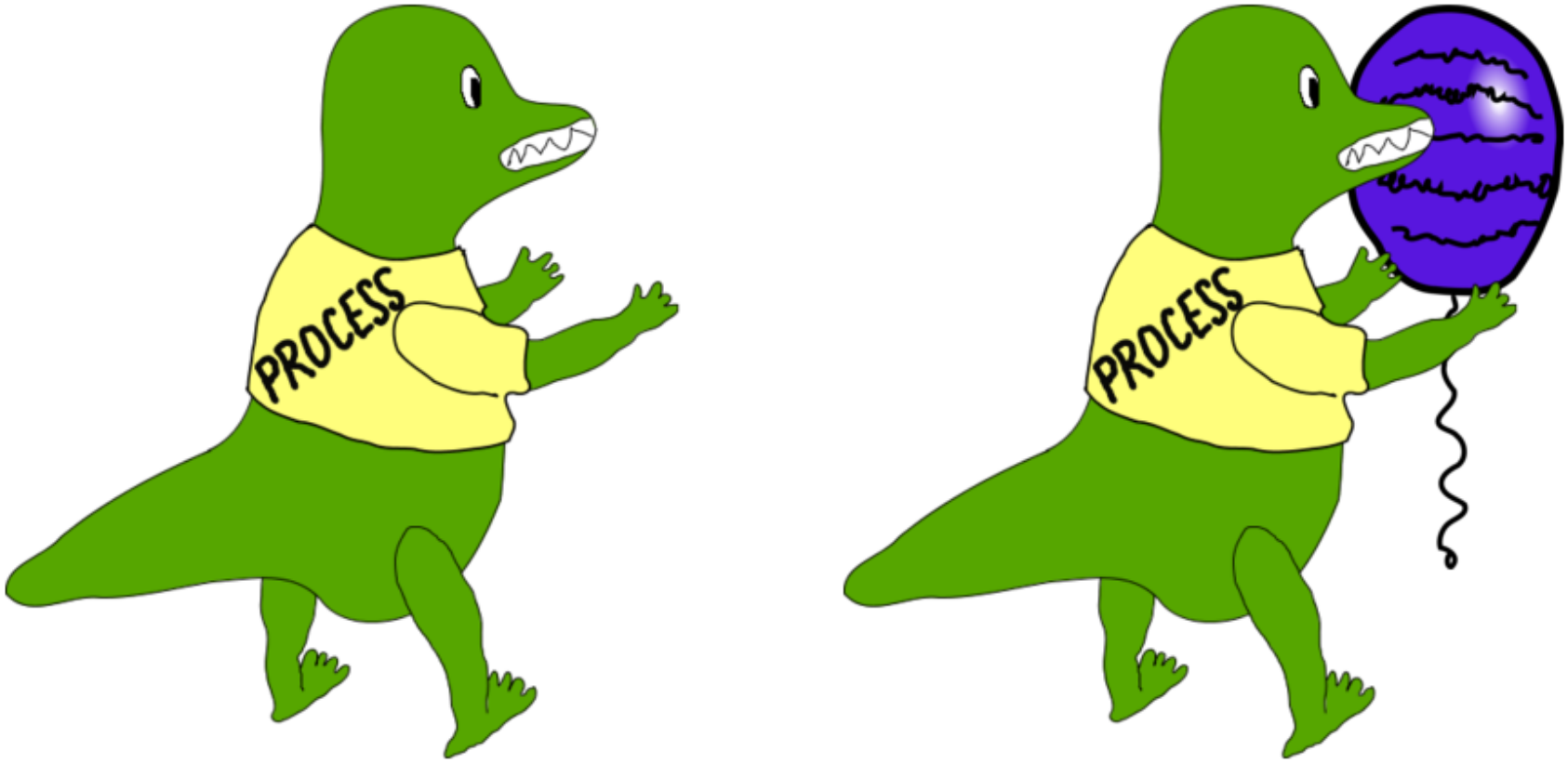
Communication



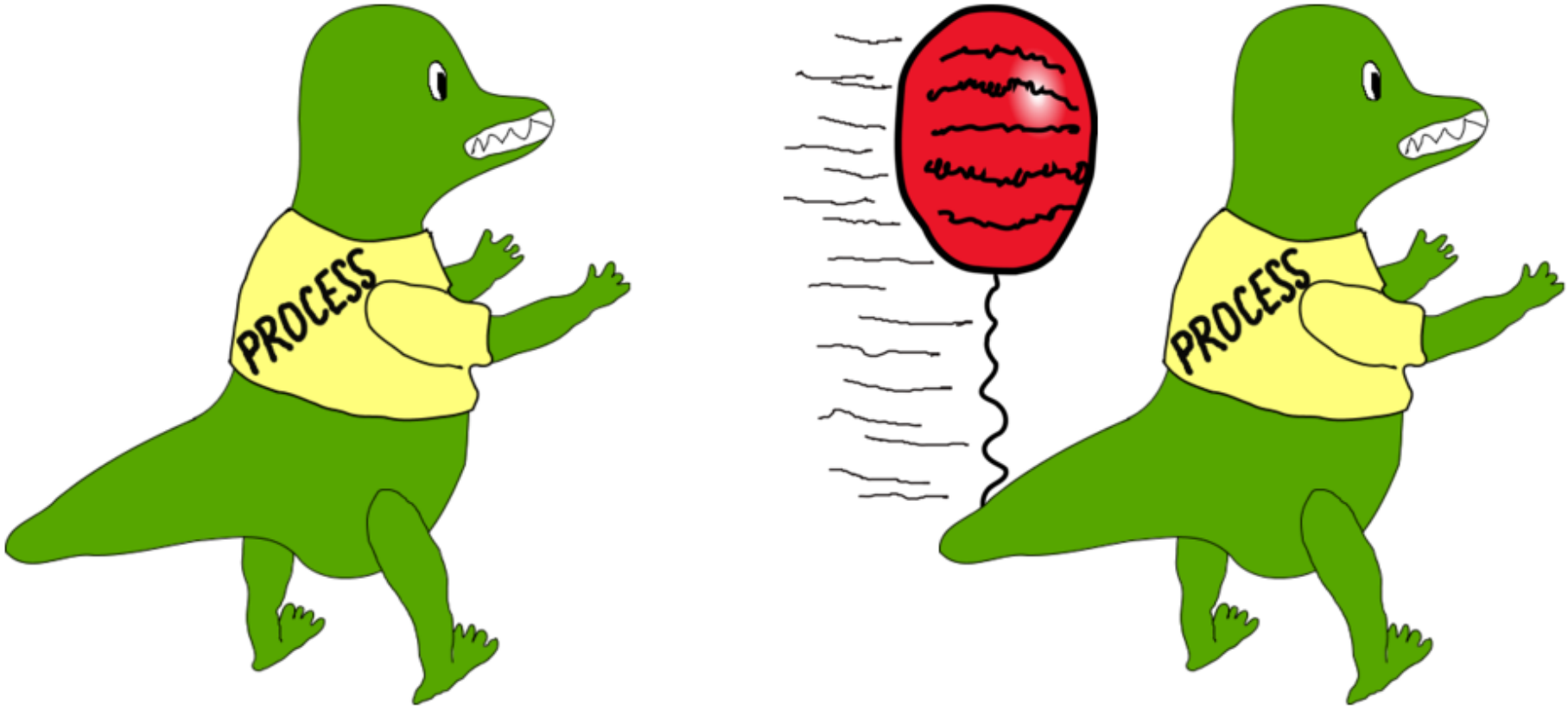
Communication



Communication



Communication



Idea behind Auto-Mobiles

- Make all non-tiny data mobile (i.e. a heap object)
- Normally, copy and send reference
- If you don't need your copy afterwards, send your original
- The recipient doesn't need to know which you did!

Idea behind Auto-Mobiles

- Make all non-tiny data mobile (i.e. a heap object)
- Normally, copy and send reference
- If you don't need your copy afterwards, send your original
- The recipient doesn't need to know which you did!

Idea behind Auto-Mobiles

- Make all non-tiny data mobile (i.e. a heap object)
- Normally, copy and send reference
- If you don't need your copy afterwards, send your original
- The recipient doesn't need to know which you did!

Idea behind Auto-Mobiles

- Make all non-tiny data mobile (i.e. a heap object)
- Normally, copy and send reference
- If you don’t need your copy afterwards, send your original
- The recipient doesn’t need to know which you did!

How do we know if we still need it?

How do we know if we still need it?

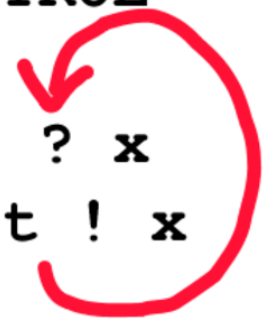
Compile-Time Program Flow Analysis

Flow Analysis Examples

```
PROC id (CHAN FOO in?, out!)
  FOO x:
  WHILE TRUE
    SEQ
      in ? x
      out ! x
:
```

Flow Analysis Examples

```
PROC id (CHAN FOO in?, out!)
  FOO x:
  WHILE TRUE
    SEQ
      in ? x
      out ! x
:
```

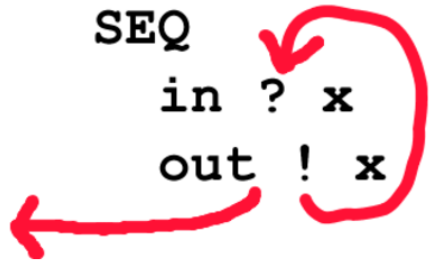


Flow Analysis Examples

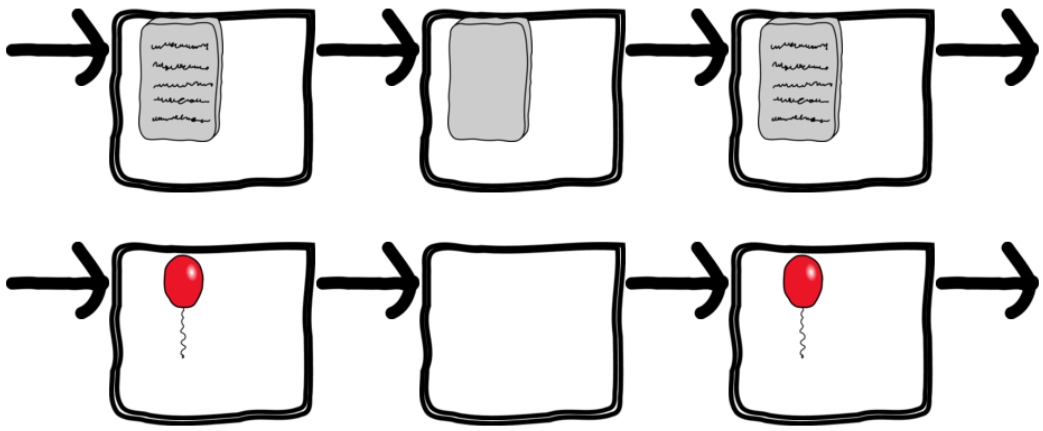
```
PROC id.100 (CHAN FOO in?, out!)
  FOO x:
  SEQ i = 0 FOR 100
    SEQ
      in ? x
      out ! x
:
```

Flow Analysis Examples

```
PROC id.100 (CHAN FOO in?, out!)
  FOO x:
  SEQ i = 0 FOR 100
    SEQ
      in ? x
      out ! x
:
```



Memory Use



Speedup Bounds

Simplifying assumptions:

- All data same size
- Consistent communication behaviour
- Allocation takes negligible time compared to copying

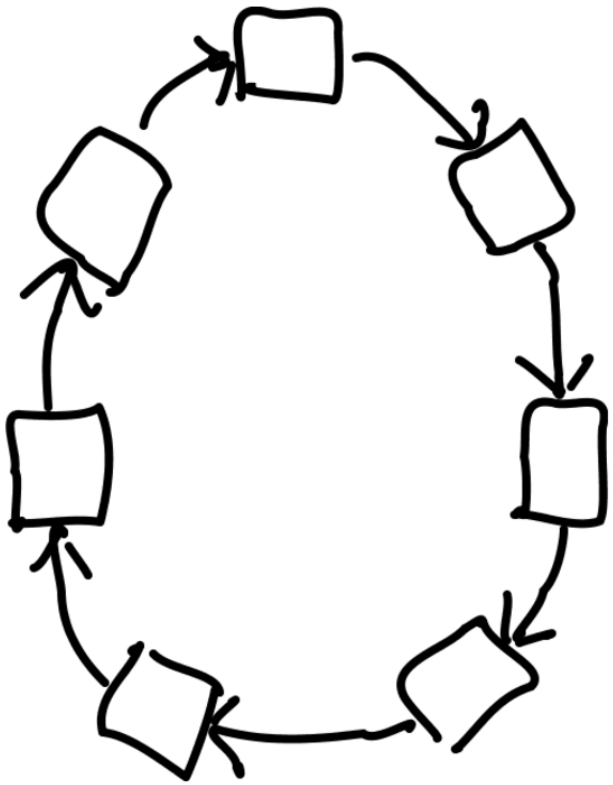
Speedup Bounds

Simplifying assumptions:

- All data same size
- Consistent communication behaviour
- Allocation takes negligible time compared to copying

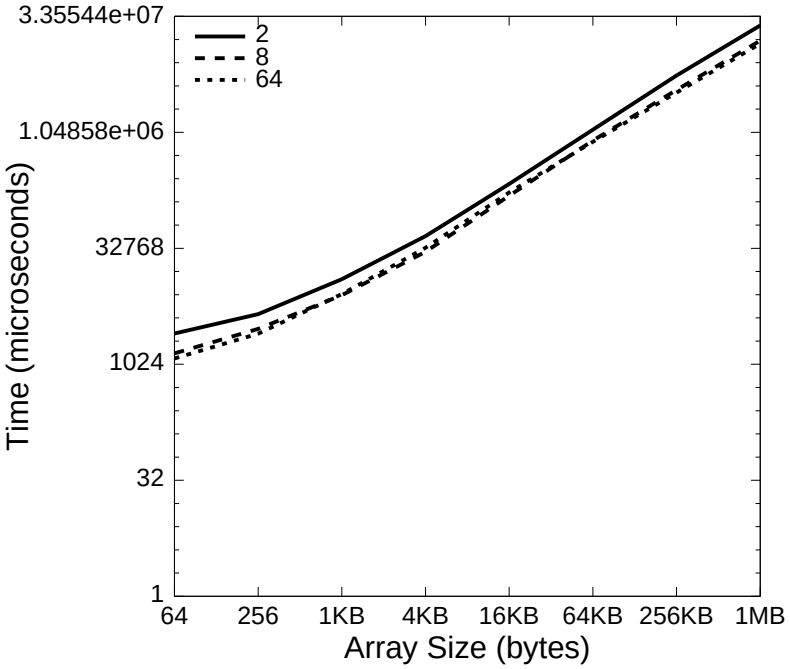
$$\text{Speedup Factor Bound} = \frac{1}{1-M}$$

Benchmark 1 of 3: The Ring



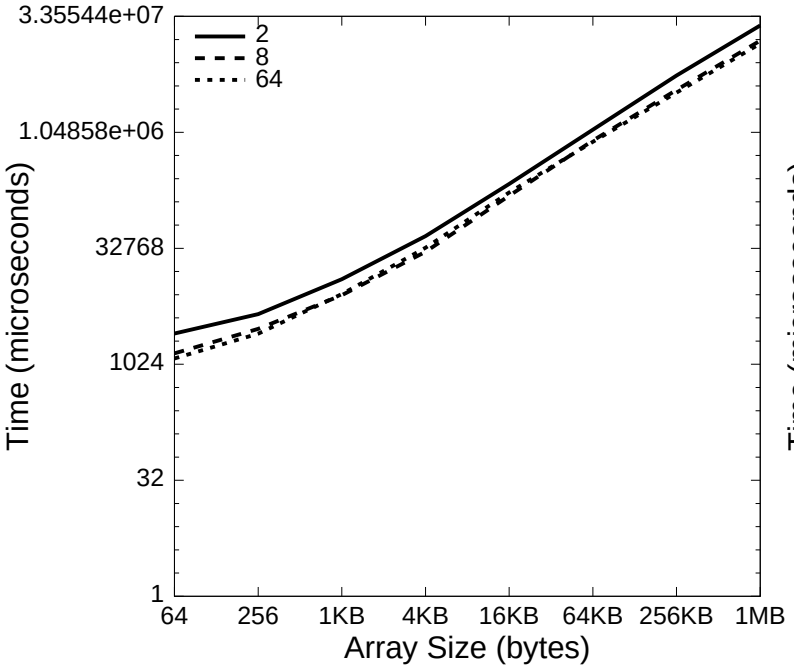
University of Kent | Computing

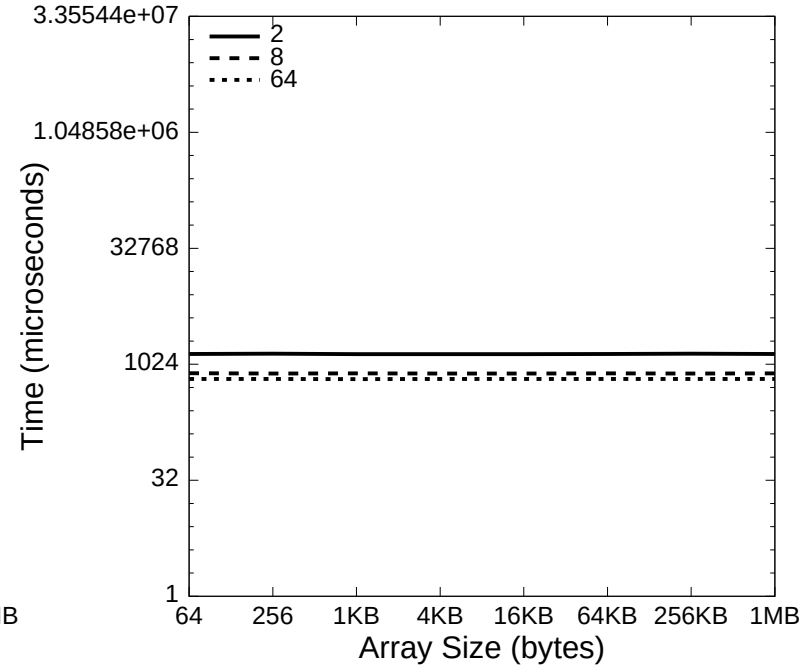
Benchmark 1 of 3: The Ring



University of Kent | Computing

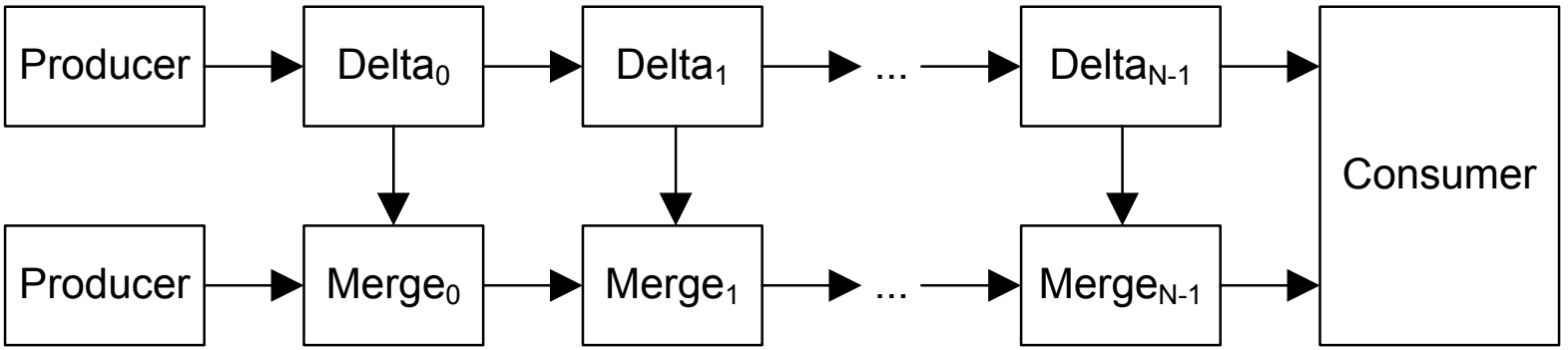
Benchmark 1 of 3: The Ring





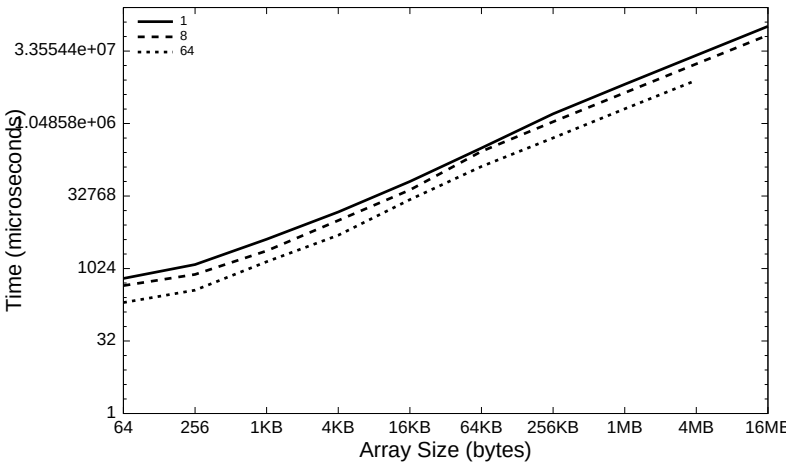
University of Kent | Computing

Benchmark 2 of 3: The Twin Pipeline



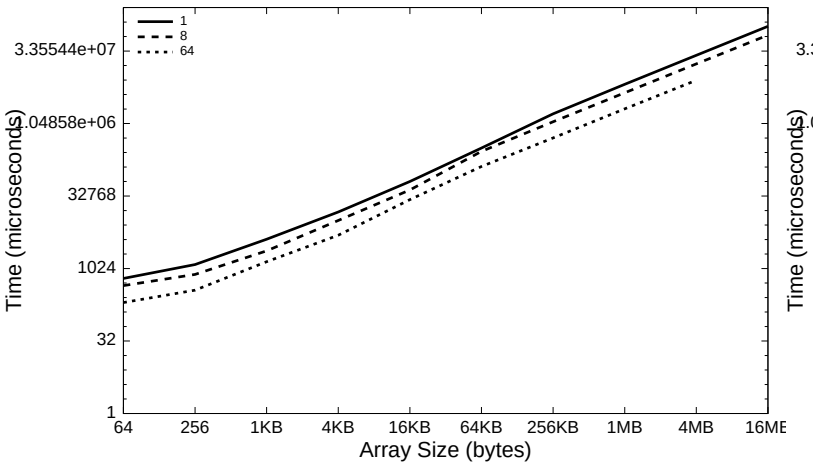
University of Kent | Computing

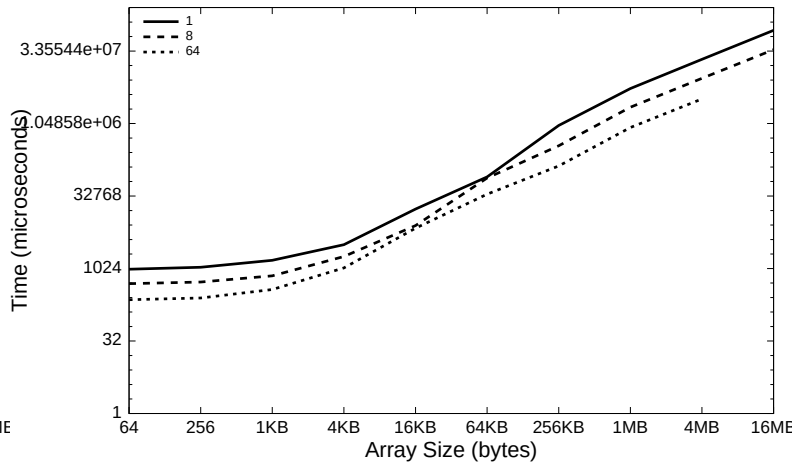
Benchmark 2 of 3: The Twin Pipeline



University of Kent | Computing

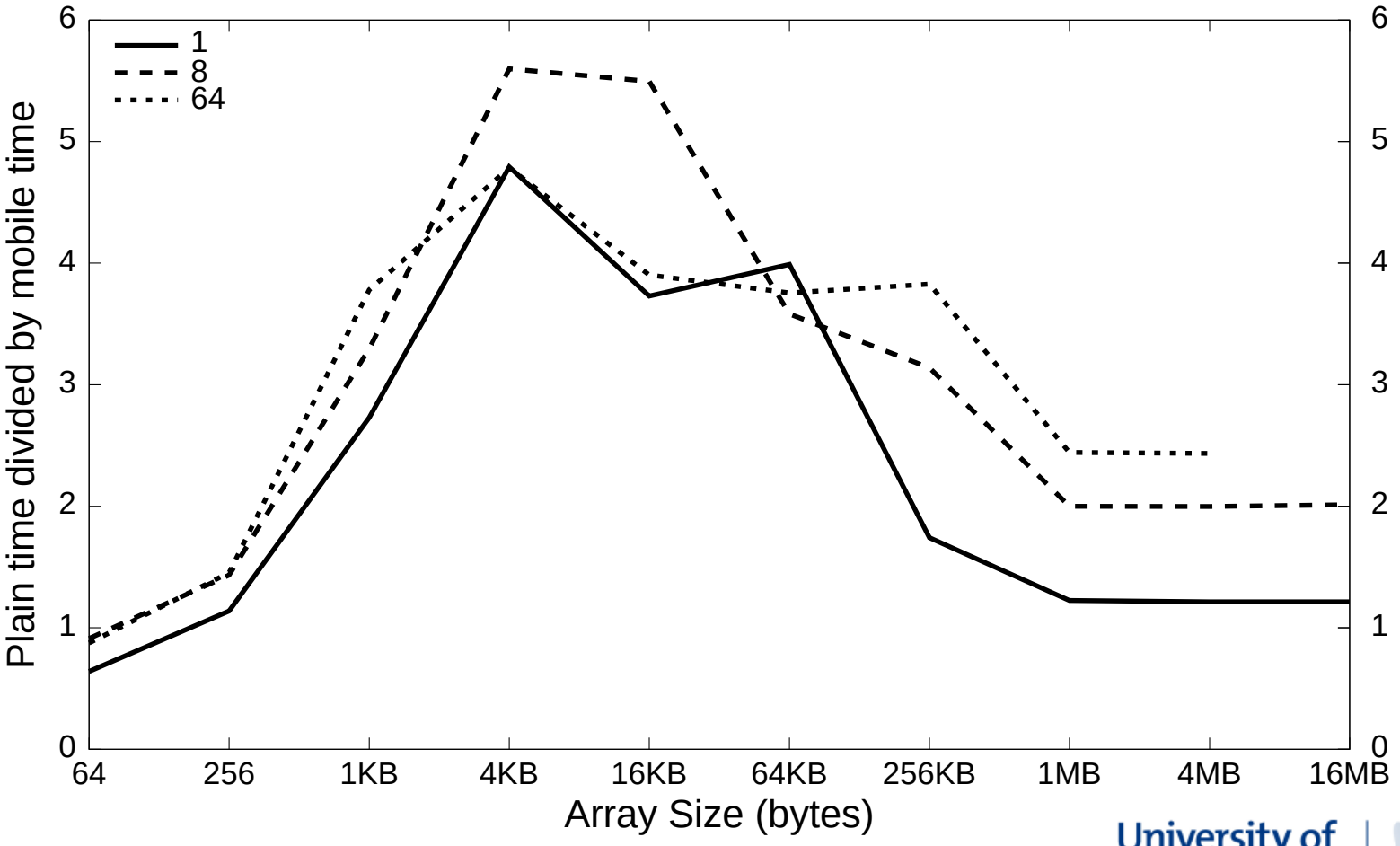
Benchmark 2 of 3: The Twin Pipeline





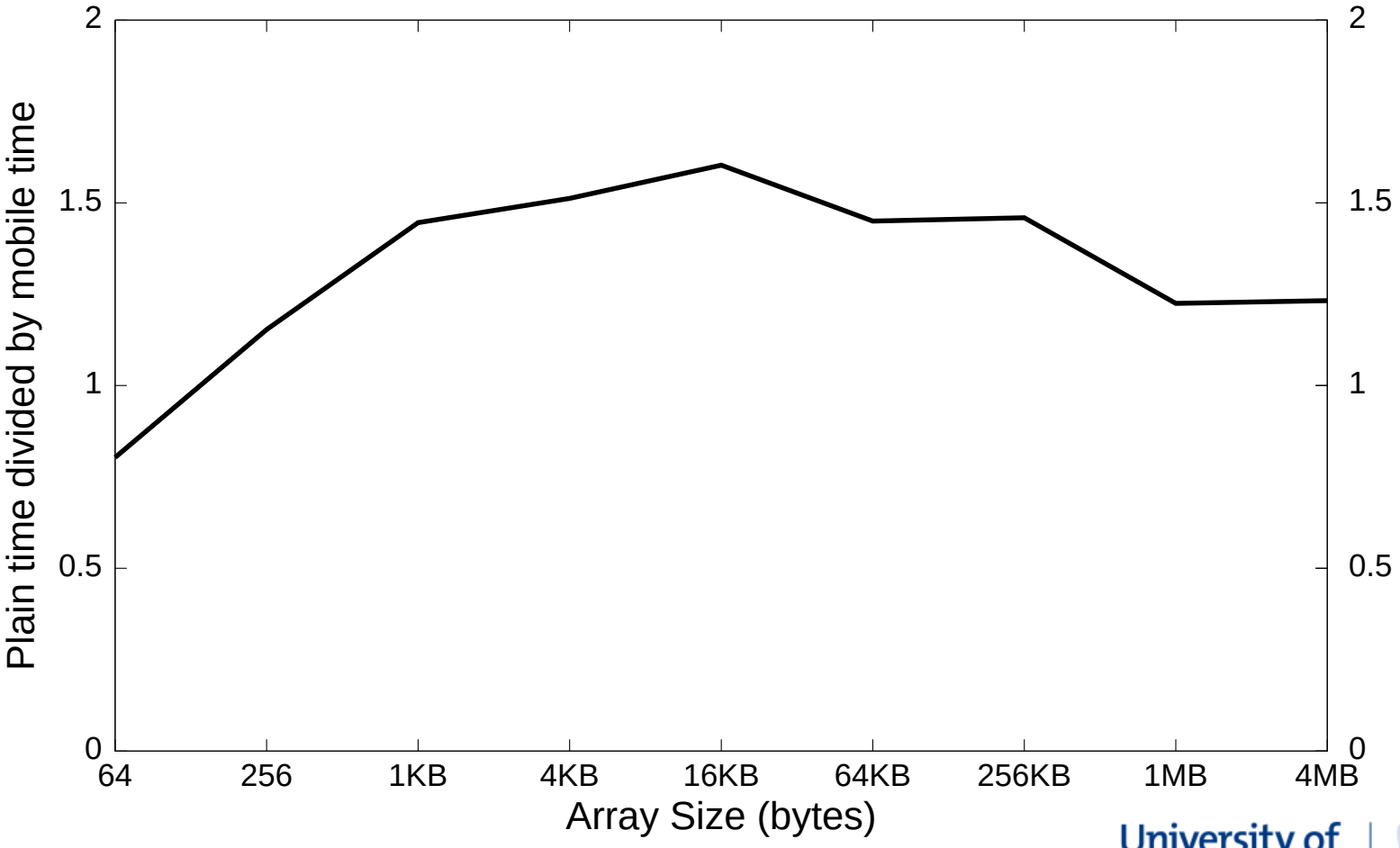
University of Kent | Computing

Benchmark 2 of 3: The Twin Pipeline



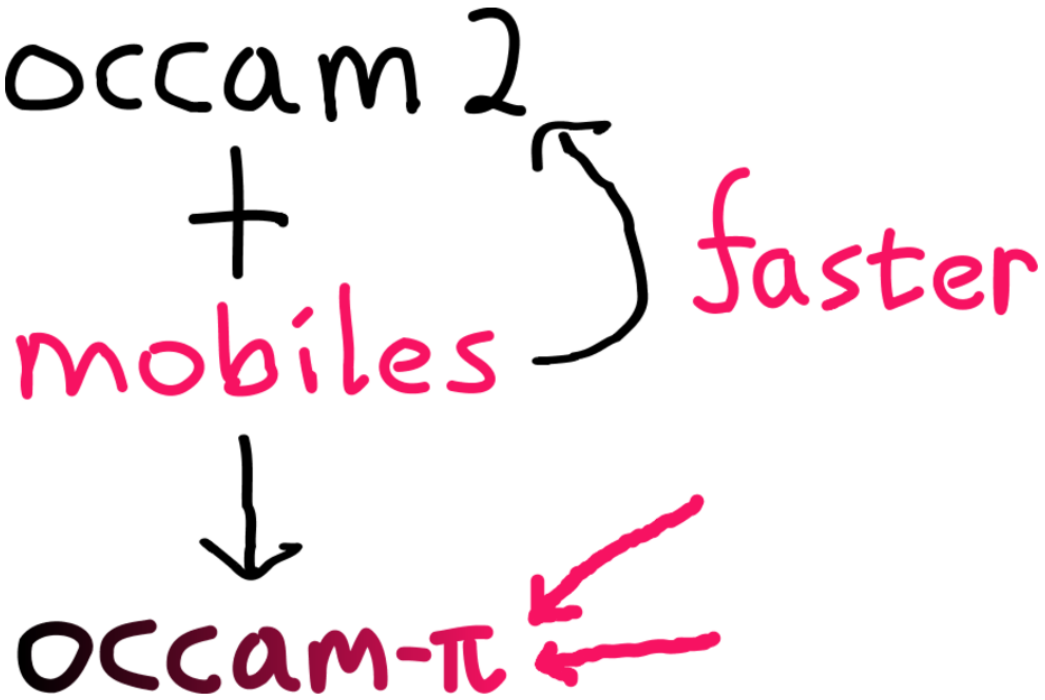
University of Kent | Computing

Benchmark 3 of 3: occam audio kit (oak)

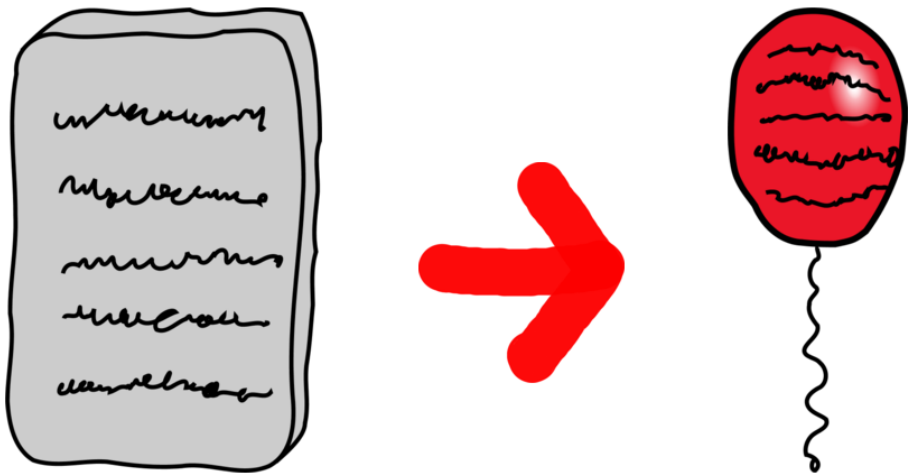


University of Kent | Computing

Summary



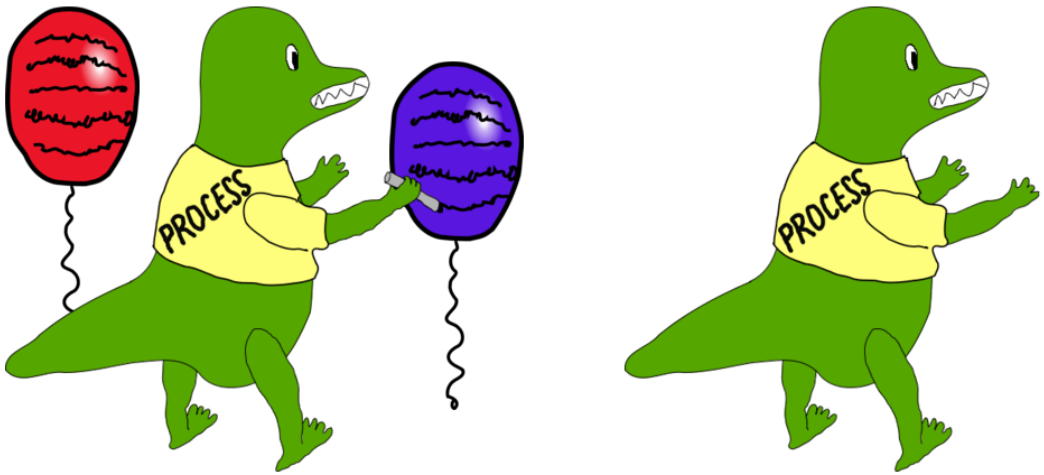
Summary



University of Kent | Computing

Summary

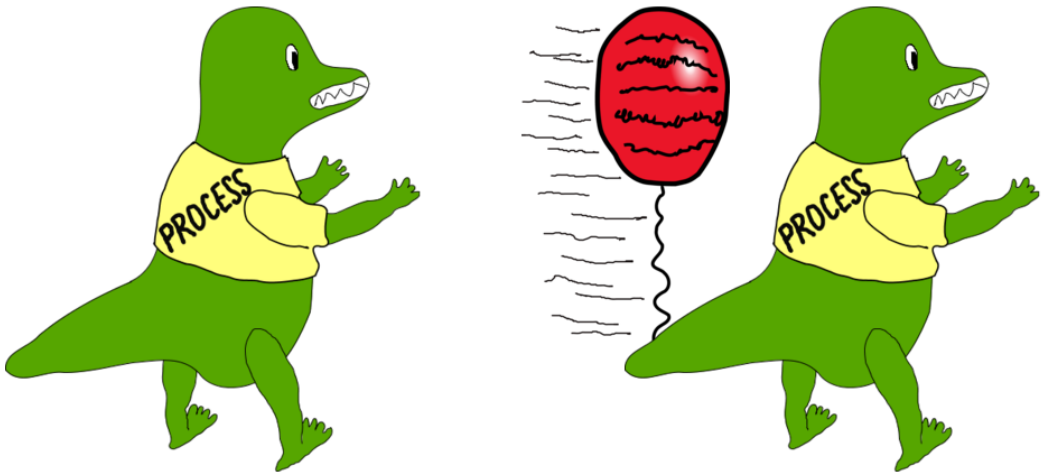
Communication



University of Kent | Computing

Summary

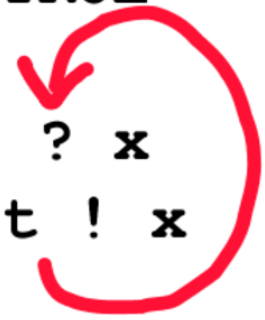
Communication



University of Kent | Computing

Summary

```
PROC id (CHAN FOO in?, out!)
  FOO x:
  WHILE TRUE
    SEQ
      in ? x
      out ! x
:
```



University of Kent | Computing

Conclusions

- It's already in Tock, enabled by compiler flag
- Better speed and memory use than plain occam 2
- No extra programmer burden (no change to code)
- Retains simple copy semantics
- Opportunity cost of mobile data ideas (e.g. mobile atoms)
  - Can use mobile channels as tokens
- Supporting dynamically-sized arrays is now easy

Questions

Allocation vs Copying

$$c(S) > (1 - M)(c(S) + a(S))$$

$$\frac{M}{1-M} > \frac{a(S)}{c(S)}$$

Copy-on-Write