MANCHESTER
1824
The University of Manchester

# Real-Time Garbage Collection for history traces in plasticity algorithms on SpiNNaker

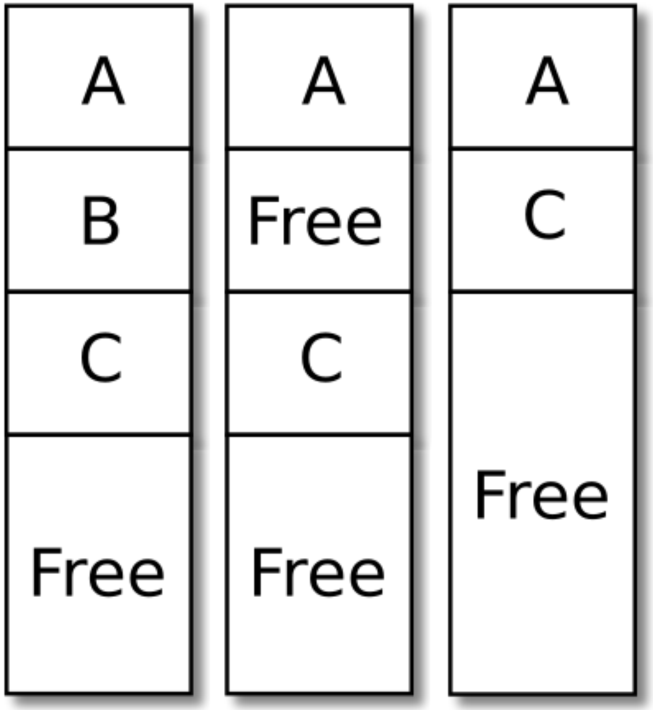
Mantas Mikaitis

# Garbage collection

- Automatic memory management
- Most commonly applied to RAM where programs allocate many small memory blocks
- JAVA has a big set of garbage collectors
- C does not have a garbage collector by default
- On SpiNNaker, we do not have memory management of any sort
- Therefore memory allocation and freeing is fully controlled by a programmer

# Garbage collection – simple example

A, B, C are memory objects. B is a dead object– no program will use it anymore. Thus garbage collector does the following in states 2 and 3:



Originated in 1960s in LISP programming language*

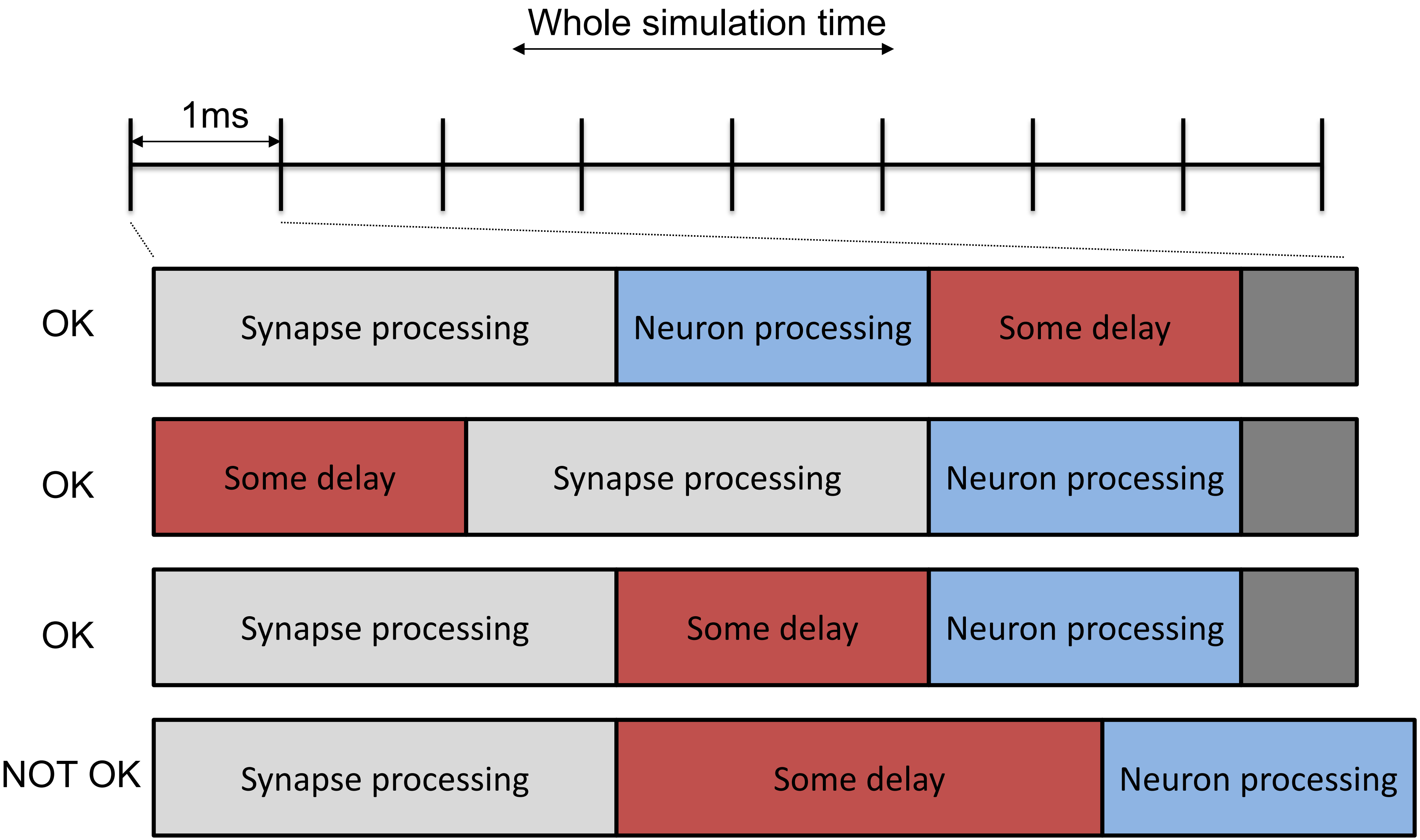
* McCarthy, Recursive functions of symbolic expressions and their computation by machine, part I, 1960

# Hard real-time systems

- Real-time garbage collection is important in hard real-time systems
- For example, passenger jets
- In these systems mutator* must not be interrupted
- Additionally, collection must preserve correct memory state when exiting early
- SpiNNaker is semi-hard real-time system

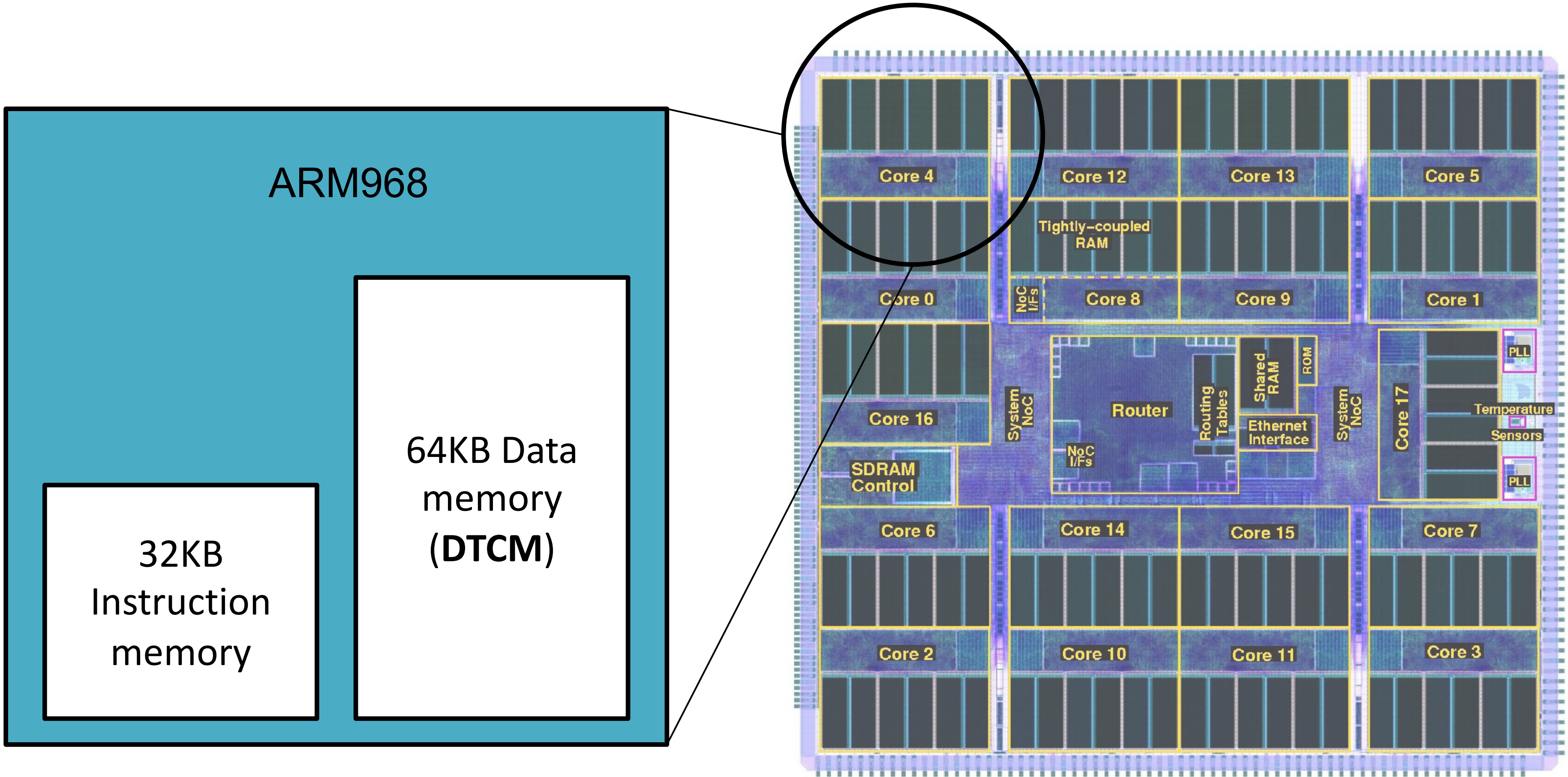
* Program that is reading and writing memory

# Semi-hard real-time system



MANCHESTER
1824
The University of Manchester

# SpiNNaker Chip Tear-Down



ARM968

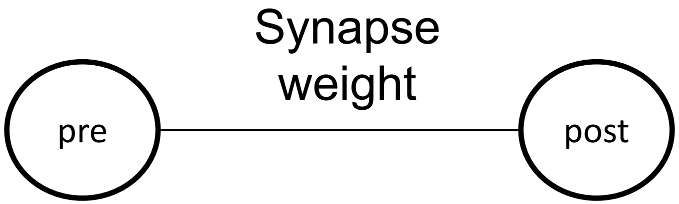
32KB Instruction memory

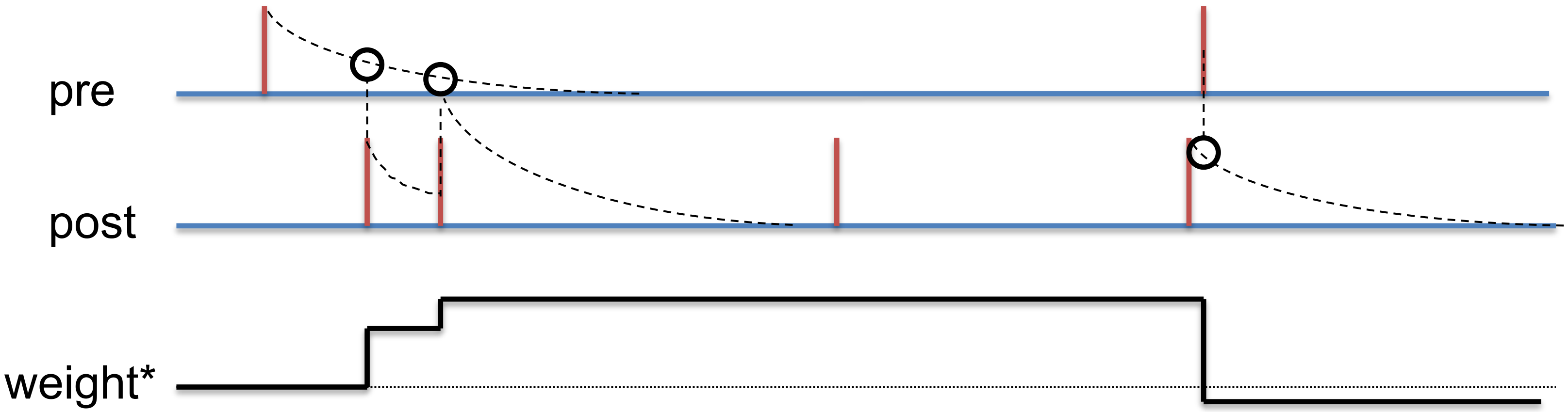
64KB Data memory (**DTCM**)

Each ARM968 has a personal Direct-Memory-Access Controller

18 ARM968 cores
and 128MB shared SDRAM

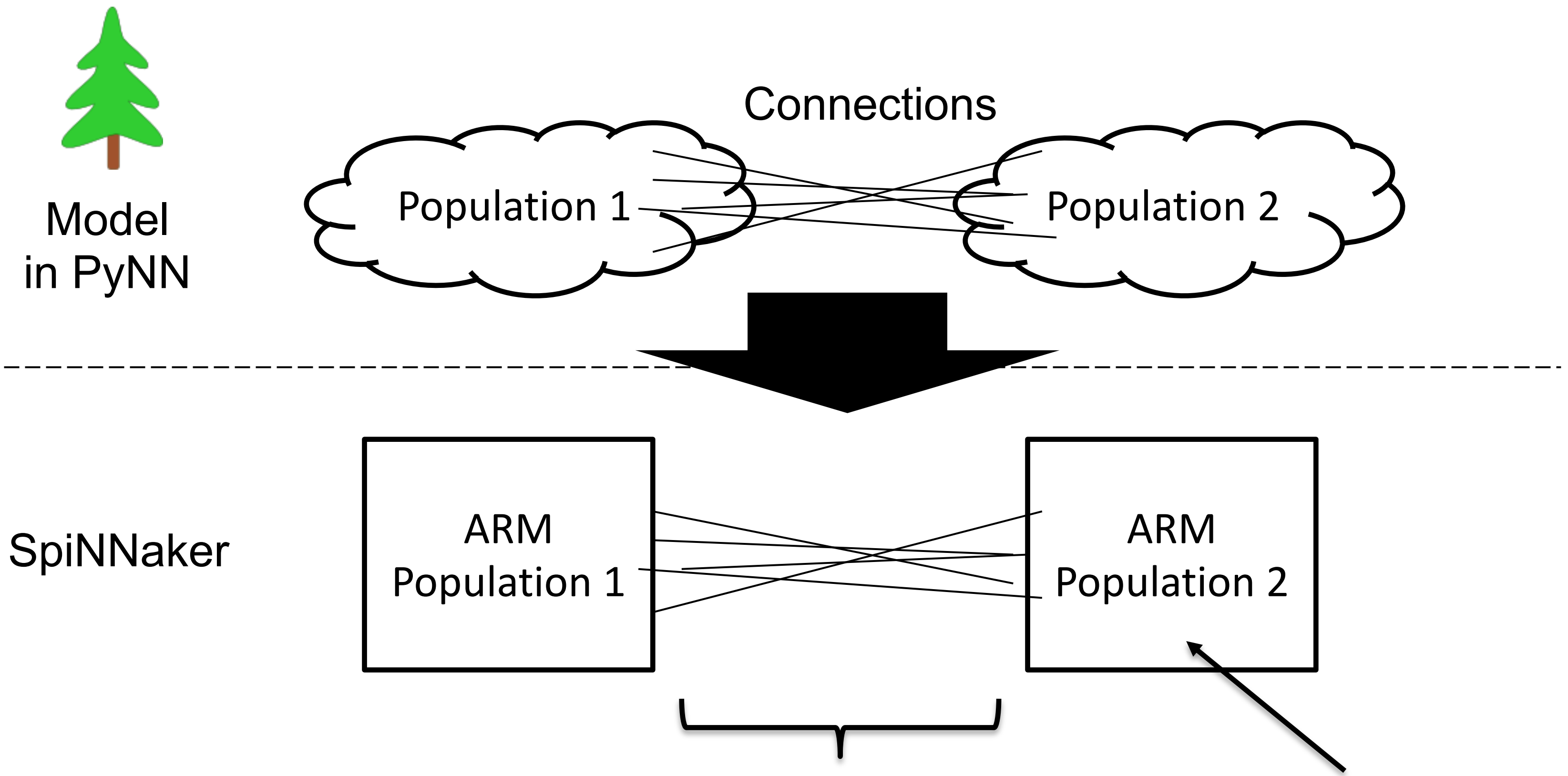
# Spike-Timing-Dependant-Plasticity(STDP)





* Additive weigh dependence. Other rules exist.

# Simulating plastic neural networks on SpiNNker



Model
in PyNN

Connections

Population 1

Population 2

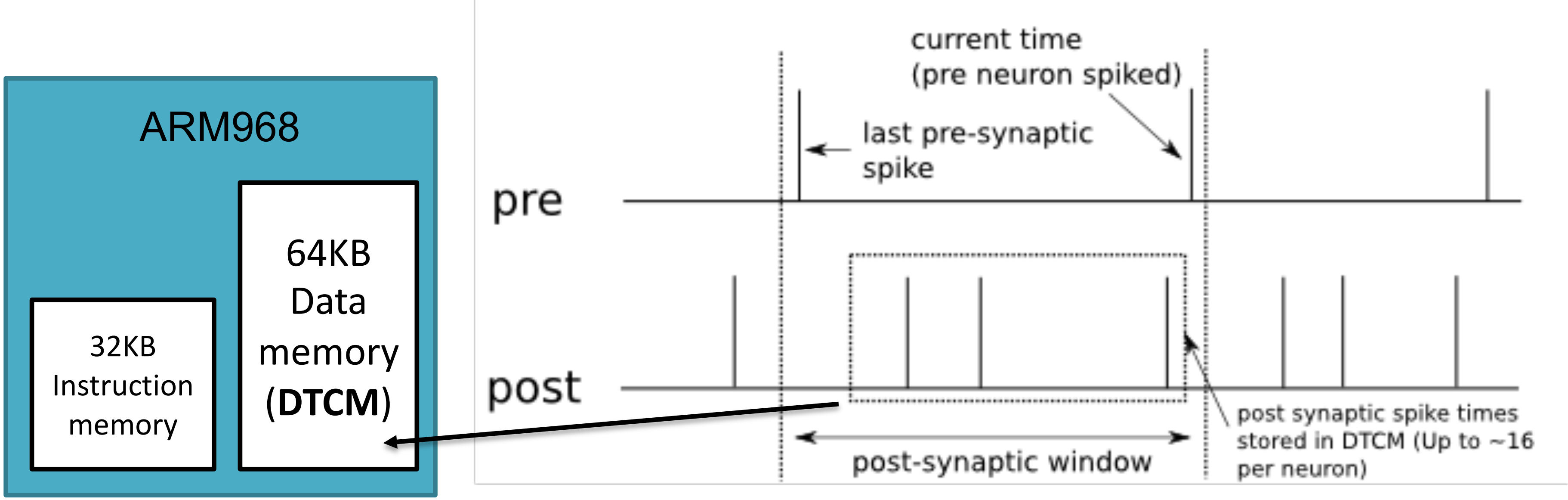
SpiNNaker

ARM
Population 1

ARM
Population 2

Plasticity of **all connecting synapses** is processed here

# Simulating plastic neural networks on SpiNNker



ARM968

32KB Instruction memory

64KB Data memory (**DTCM**)

current time (pre neuron spiked)

last pre-synaptic spike

pre

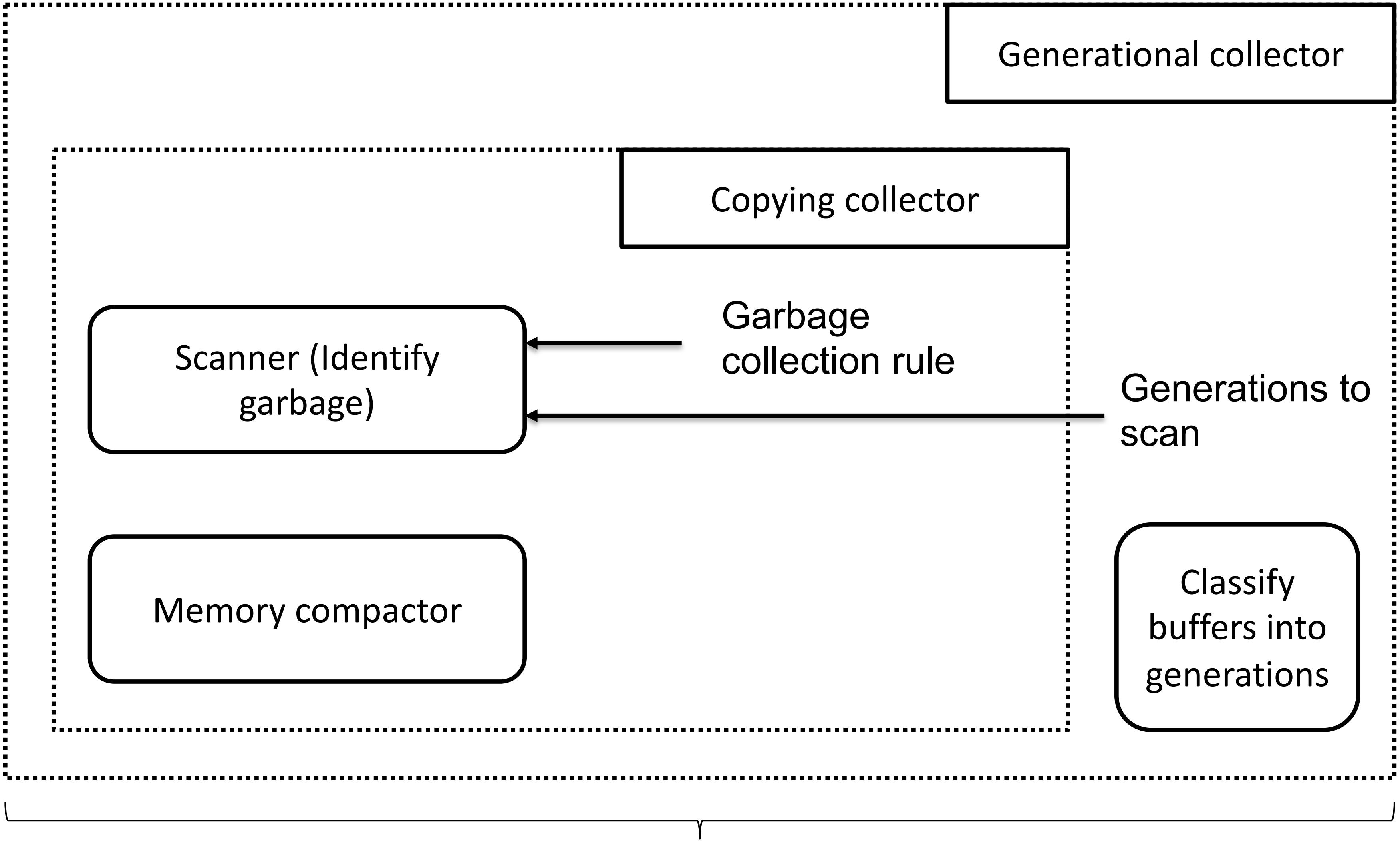
post

post-synaptic window

post synaptic spike times stored in DTCM (Up to ~16 per neuron)

Overflow of memory can happen in history trace buffers

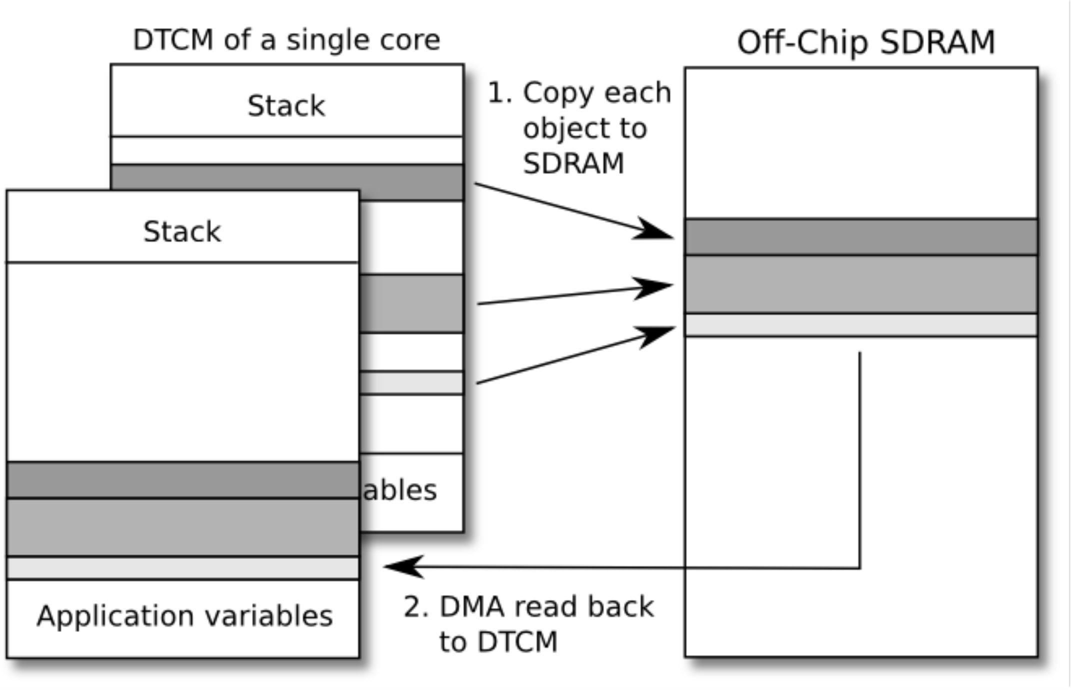
# Implementation details: Main components



All of these require a fast memory copying function

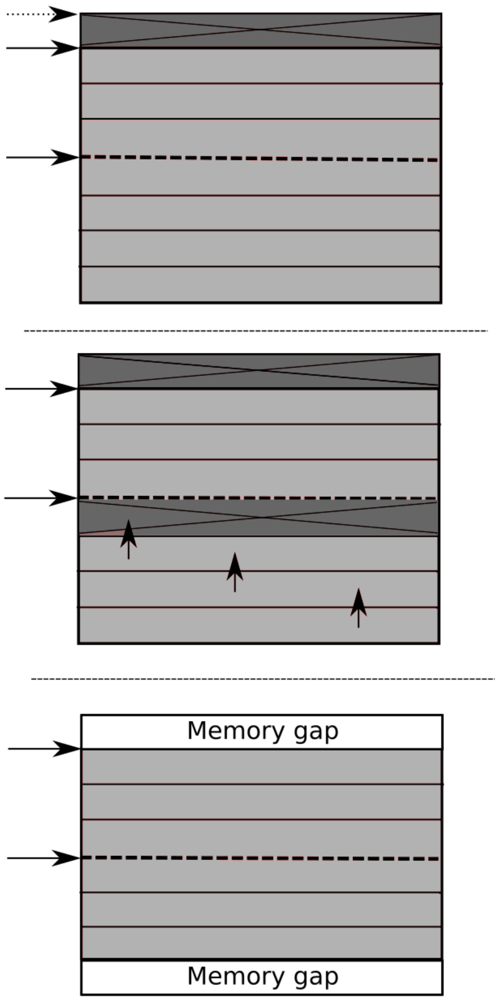
# Memory compactor

- Memory is potentially almost full and list of object pointers is not ordered
- Therefore use SDRAM for the working space of the compactor

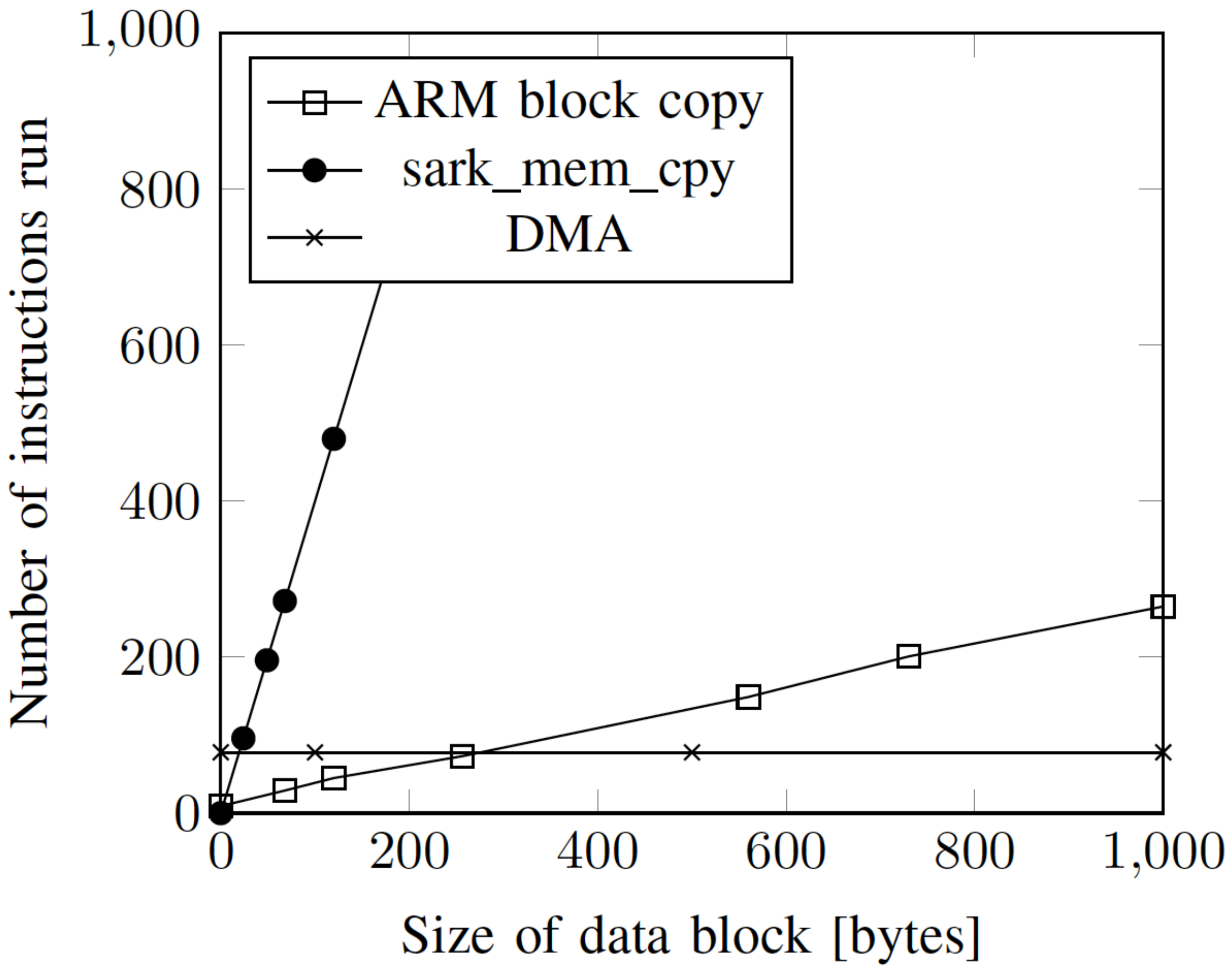


# Scanner (Identify garbage)

- Garbage collection rule is used to find ”dead” objects
- A trace contains two elements: time of spike and an actual trace in different parts of a buffer
- To remove a trace, shift elements up and move pointers
- An exhaustive search over all objects
- What is dead trace: I have used ”older than 500ms”

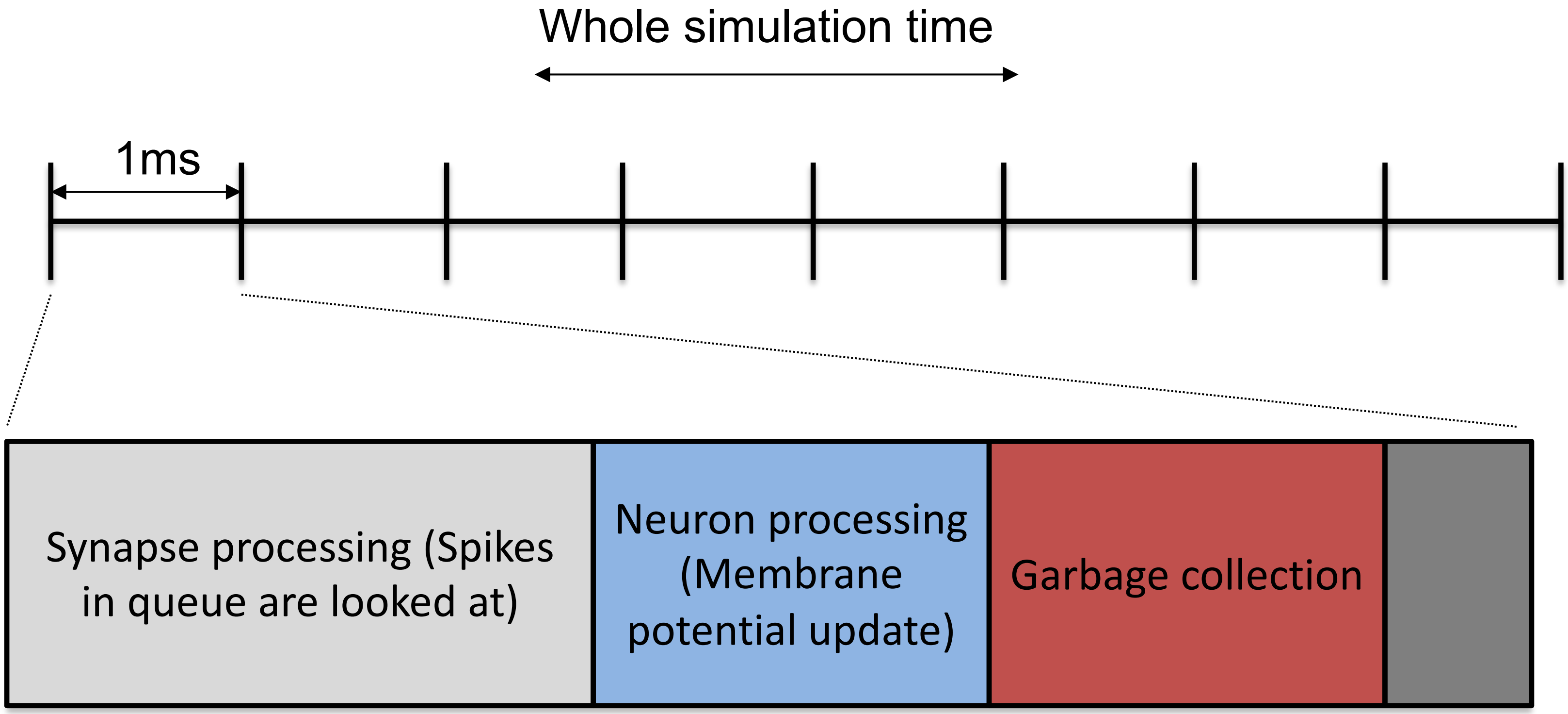


# Fast memory copying function – ARM block copy



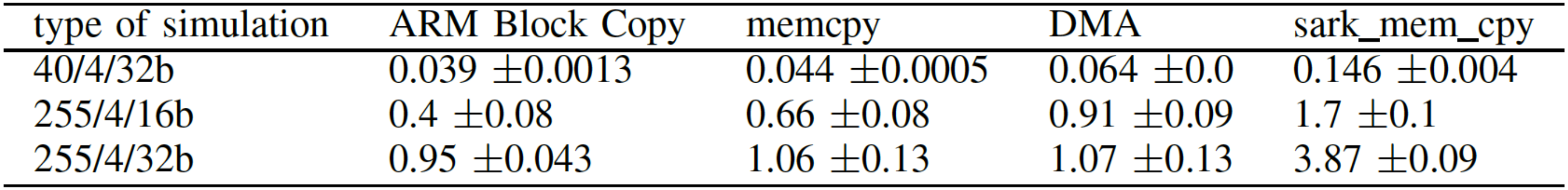
* ARM block copy uses LDM instruction instead of LDR or LDRB

# Real-time simulation constraint


Whole simulation time
1ms
Synapse processing (Spikes in queue are looked at)
Neuron processing (Membrane potential update)
Garbage collection

- Synapse processing and neuron processing run for ~0.55 ms using stdp_example.py simulation.
- Garbage collection includes compactor and scanner.

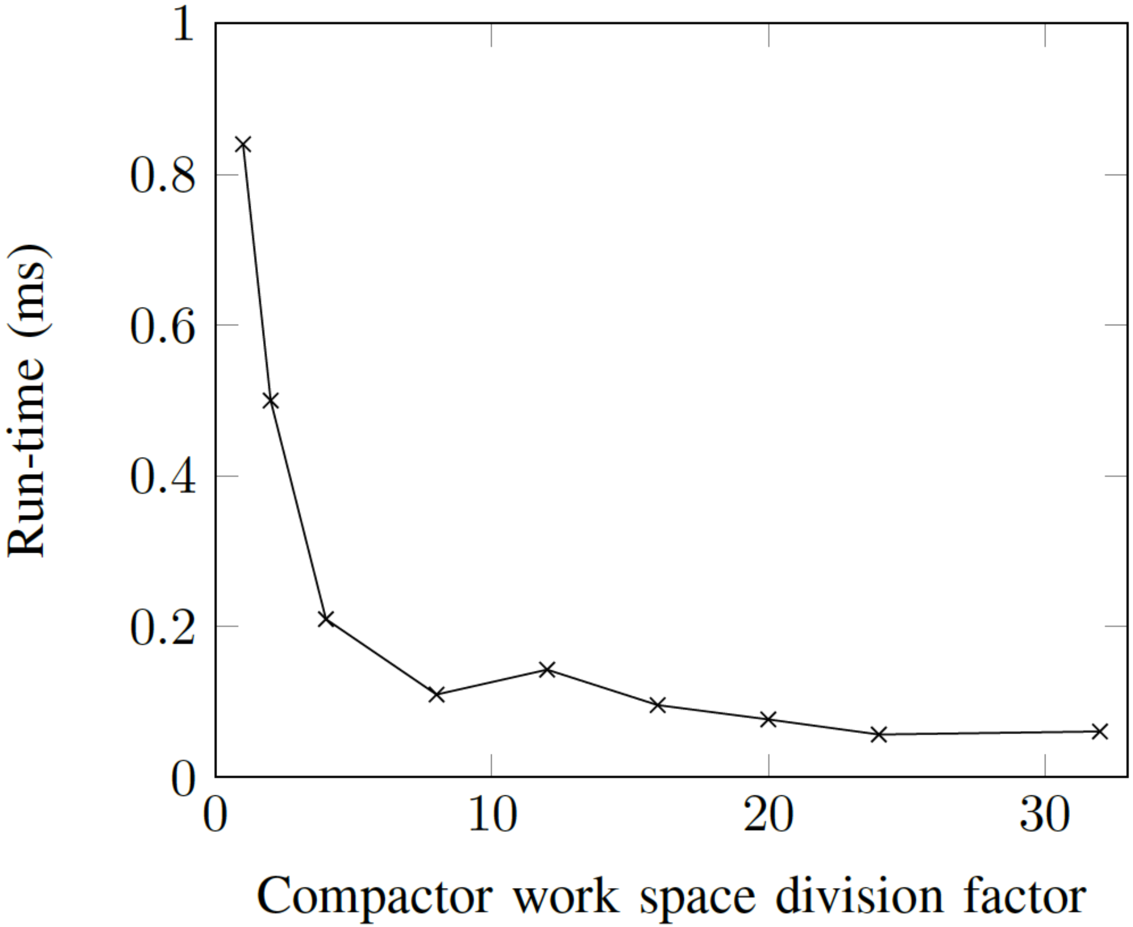
# Results: Compactor



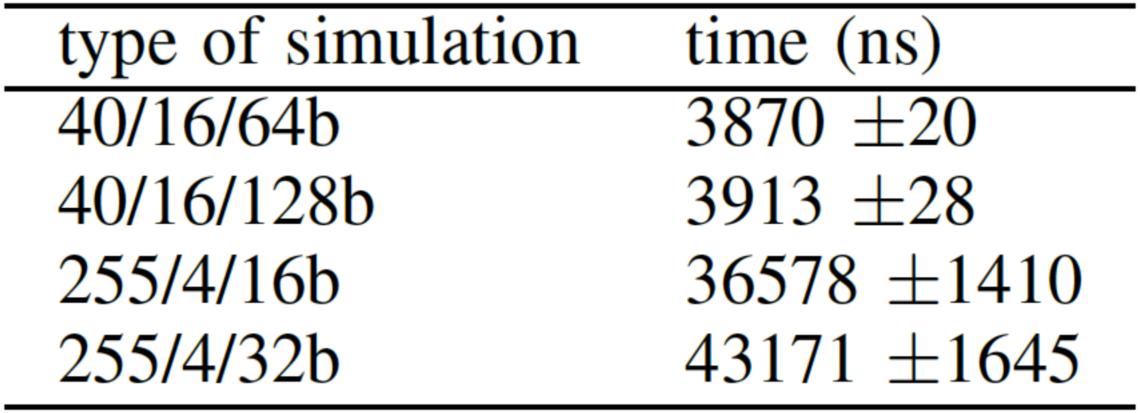
| type of simulation | ARM Block Copy | memcpy | DMA | sark_mem_cpy |
|---|---|---|---|---|
| 40/4/32b | 0.039 ±0.0013 | 0.044 ±0.0005 | 0.064 ±0.0 | 0.146 ±0.004 |
| 255/4/16b | 0.4 ±0.08 | 0.66 ±0.08 | 0.91 ±0.09 | 1.7 ±0.1 |
| 255/4/32b | 0.95 ±0.043 | 1.06 ±0.13 | 1.07 ±0.13 | 3.87 ±0.09 |

- Type of simulation: neurons per core / post traces per buffer / initial size of buffer (STDP rule controlled)
- Time expressed in mili-seconds
- In the last row, compactor run-time is nearing 1ms

# Reduce compactor run time



# Results: Scanner



| type of simulation | time (ns) |
|---|---|
| 40/16/64b | 3870 ±20 |
| 40/16/128b | 3913 ±28 |
| 255/4/16b | 36578 ±1410 |
| 255/4/32b | 43171 ±1645 |

- Type of simulation: neurons per core / post traces per buffer / initial size of buffer (STDP rule controlled)
- Time expressed in nano-seconds

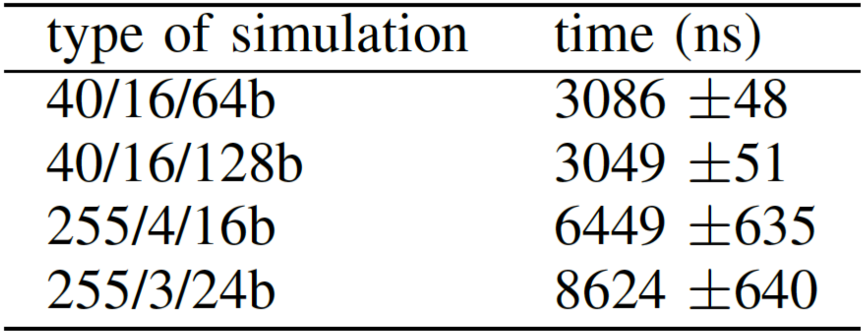
# Reduce scanner run-time: Generational garbage collector

- Main principle: Keep track of how old memory region is and how much it was garbage collected as demonstrated by Lieberman et al*.
- Old regions which were garbage collected a lot, are likely to hold more permanent objects so scan them less often

- For history traces, have two generations: old generation and new generation
- New generation is almost unlikely to have garbage, so do not scan it
- Old generation probably has garbage, scan it all

* Lieberman et al, A Real-Time Garbage Collector Based on the Lifetimes of Objects, 1983

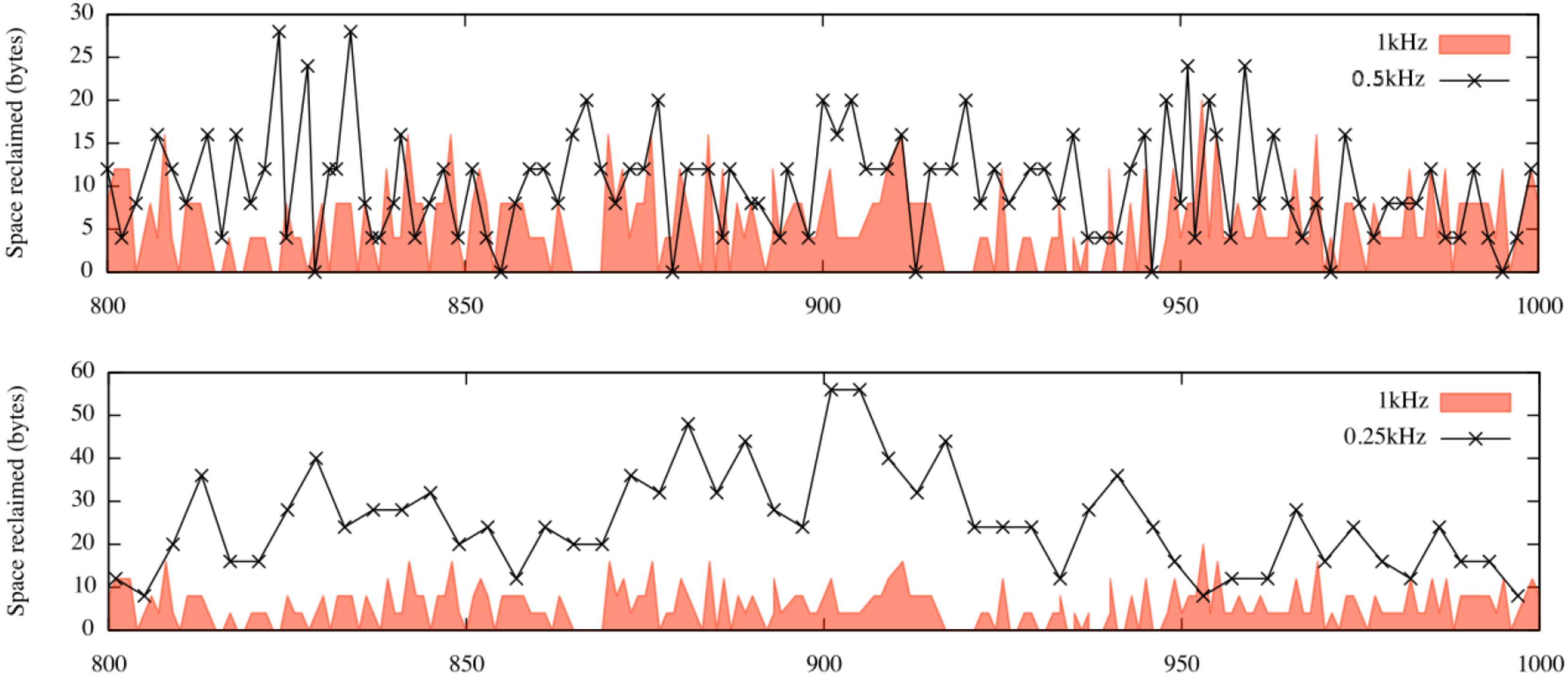
# Results: Scanner with generations



| type of simulation | time (ns) |
|---|---|
| 40/16/64b | 3086 ±48 |
| 40/16/128b | 3049 ±51 |
| 255/4/16b | 6449 ±635 |
| 255/3/24b | 8624 ±640 |

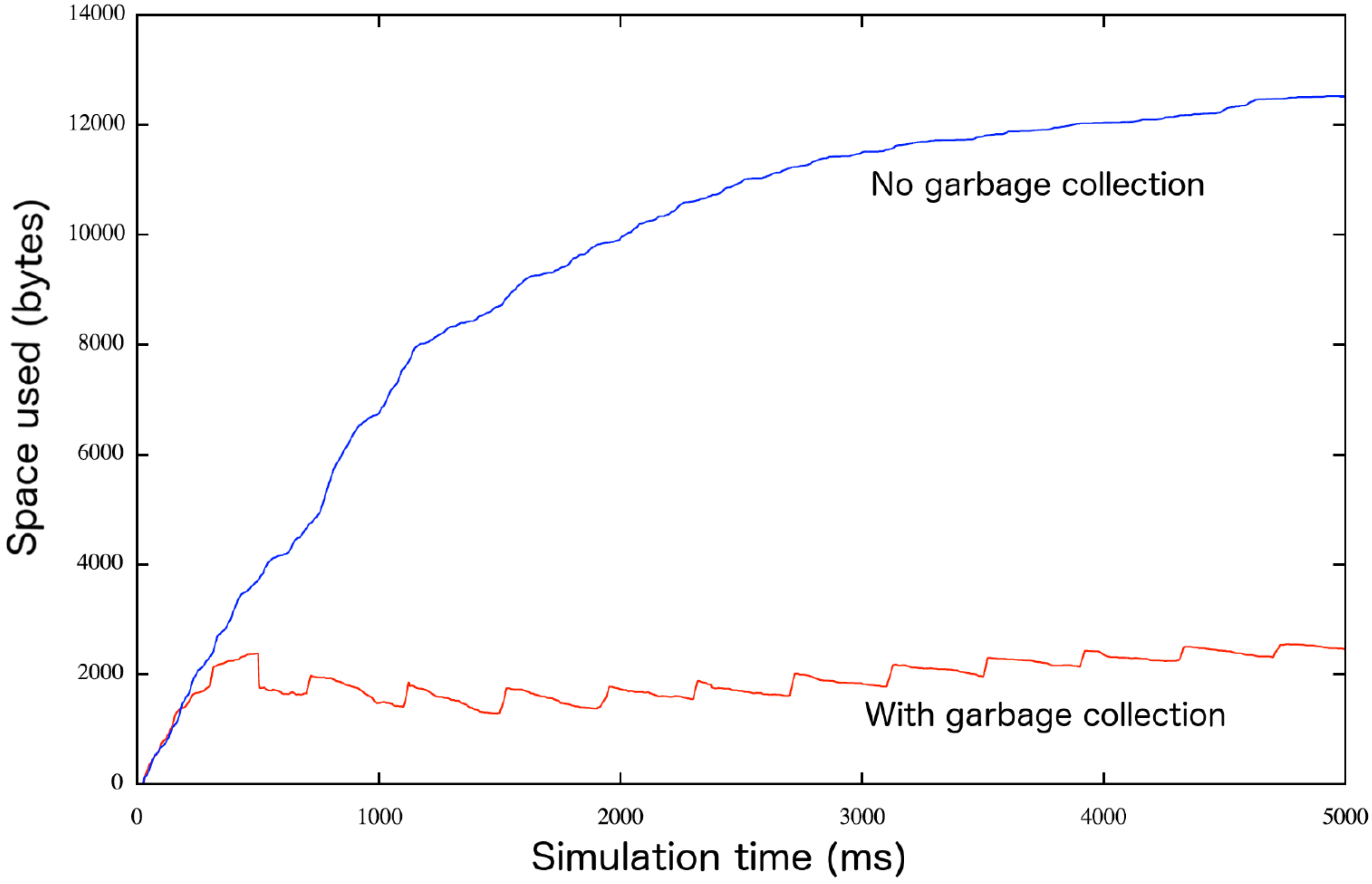
- Type of simulation: neurons per core / post traces per buffer / initial size of buffer (STDP rule controlled)
- Time expressed in nano-seconds
- Much less memory to scan reduces run-time ~5 times

# Results: Scanner



- Sometimes scanner does not find any garbage (Red trace often reaches 0 bytes)
- Collect less often to avoid wasted scan cycles (Lower illustration black curve is scanning every 4th timestep)

# Results: Total memory usage for plasticity



# Possible further work

- Garbage collector of general type objects for DTCM
- More sophisticated garbage collection rules from biological literature: When is history trace dead?
- Can garbage collector help reduce memory significantly to fit more neurons per core in plastic networks?

# Conclusion

- Garbage collection can help SpiNNaker avoid overflowing synaptic buffers and monitor real memory usage for plasticity history traces.
- Disadvantage is that it copies a lot of data around which is a slow process.
- Other known solutions: inject artificial ‘spike’ periodically that will clear plasticity history trace buffers.

Source code is available on garbage collection branches of sPyNNaker software package.

Acknowledgements:

- Project supervised by Dave Lester
- Jamie Knight allowed to use his run-time profiling tool
- Thanks to SpiNNaker software team for showing the ropes of the toolchain

**Questions?**