Copyright © by Luke Seed, September 2013

# Cache Memory

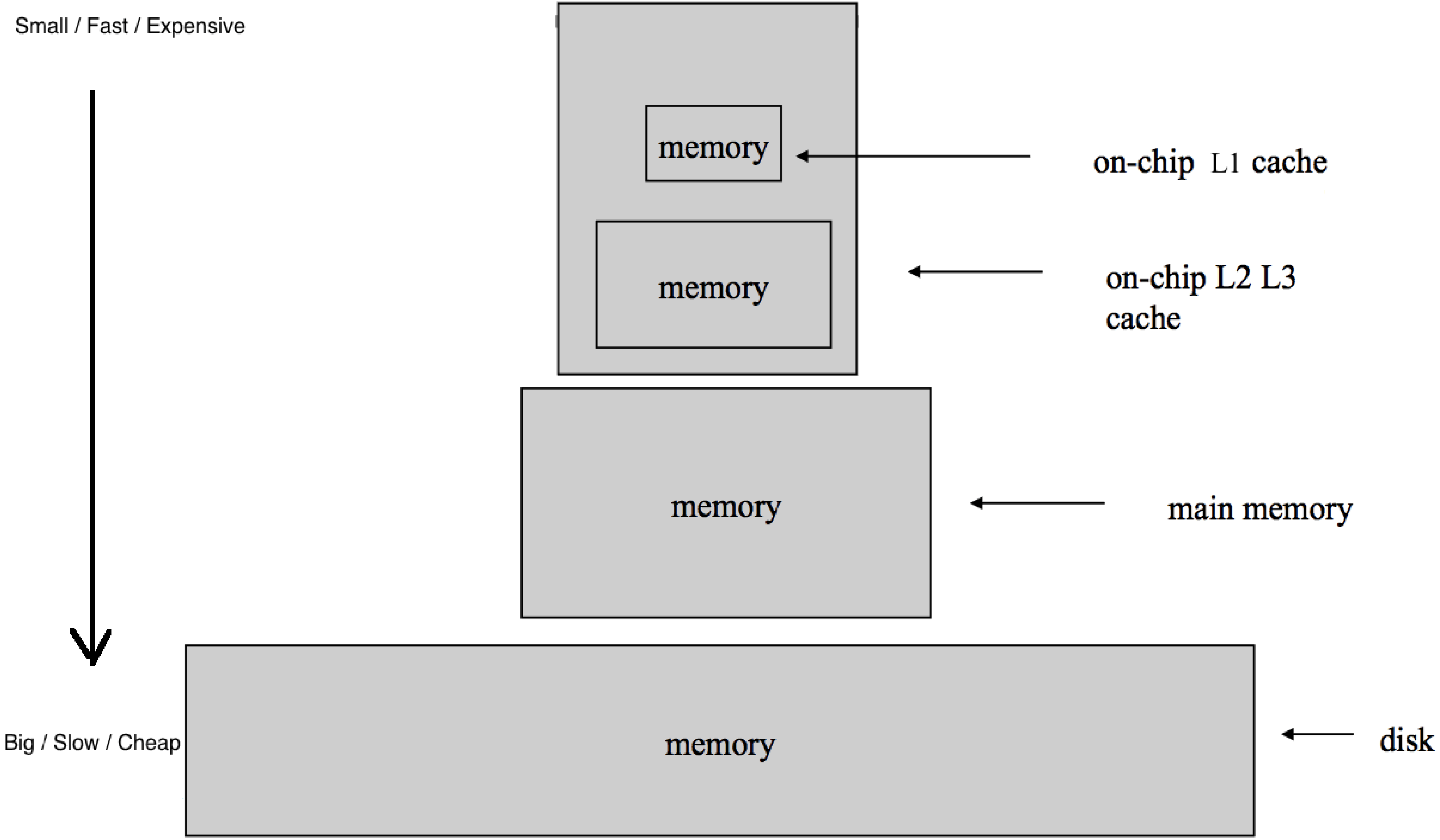
- What is cache memory?
- Cache Organisation
- Cache Implementation
- Cache Performance

Copyright © by Luke Seed, September 2013

# Cache Memory

- Early processors had an on-chip instruction queue, a modest amount of DRAM and a hard disk
- As processor speeds increased, memory access times did not keep pace and more creative solutions were required for memory interfaces.
- A multi-level memory architecture known as 'cache' provided a solution.

Copyright © by Luke Seed, September 2013



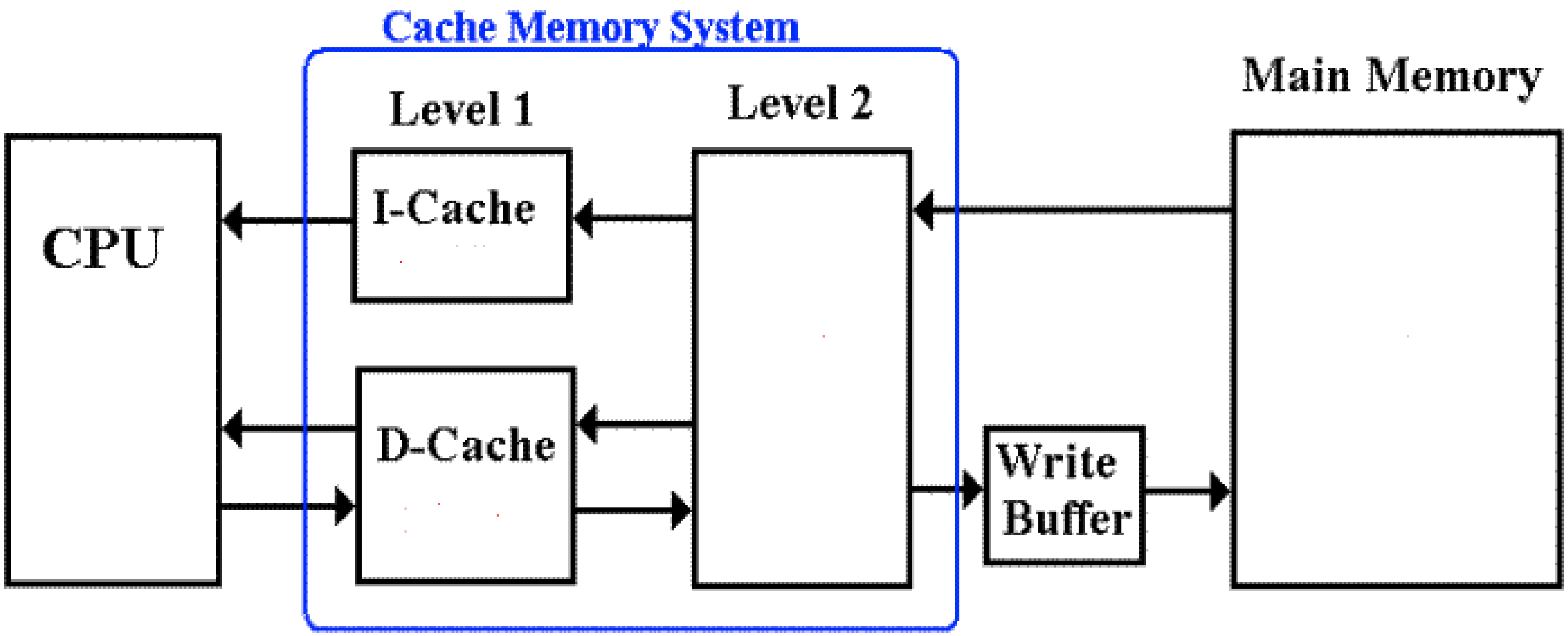
Copyright © by Luke Seed, September 2013

# Cache Memory

- Cache memory is a temporary store for data and instructions; it is faster to access than main memory
- L1 cache is a small memory on the processor chip which runs at processor speed
- L2 cache is much larger than L1 but slower. It is accessed if the required data is not found in the L1 cache; aim for L2 to have the data about 90% of the time
- Multi-core chips can have a L3 cache shared by all of the cores

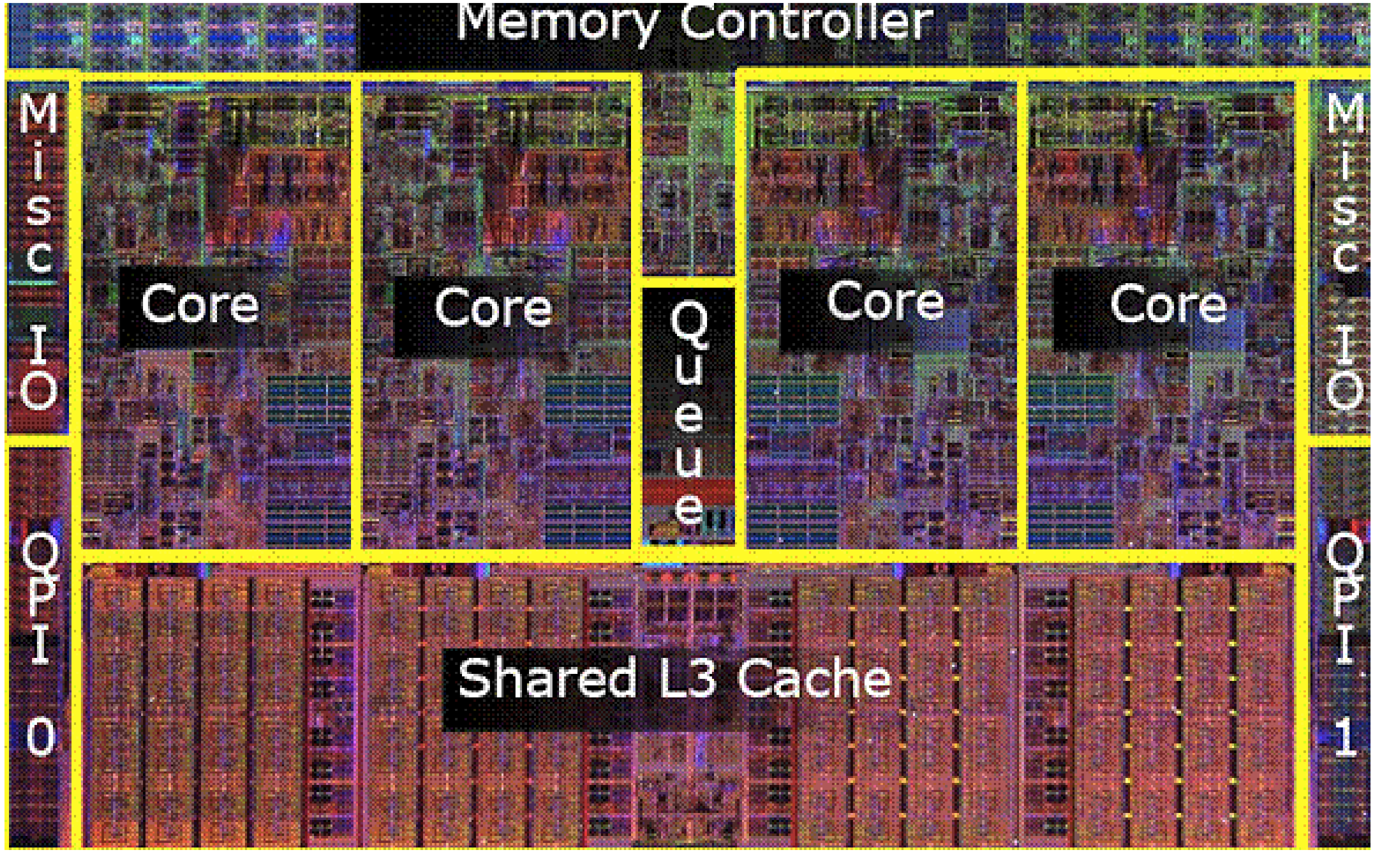
Copyright © by Luke Seed, September 2013

# Cache Memory



Copyright © by Luke Seed, September 2013

# Cache Memory



Copyright © by Luke Seed, September 2013

# Cache Definitions

- *cache hit* – cpu requested data is found in the next level cache
- *cache miss* – requested data is not in the cache and must be fetched from upper levels
- *hit time* -- time to access the cache (time to determine hit/miss + memory access time)
- *miss penalty* -- time to move data from upper levels to cpu
- *hit ratio* -- percentage of time the data is found in the cache
- *miss ratio* -- (1 - hit ratio)

Copyright © by Luke Seed, September 2013

# Types of Organisation

- Direct Mapped
- Fully Associative
- Set Associative
- + few others

- Caches are arranged in blocks (or lines) just like lower levels of the hierarchy

Copyright © by Luke Seed, September 2013

# Direct Mapped

- Simplest and cheapest
- Lowest performance
- Some significant drawbacks

Copyright © by Luke Seed, September 2013

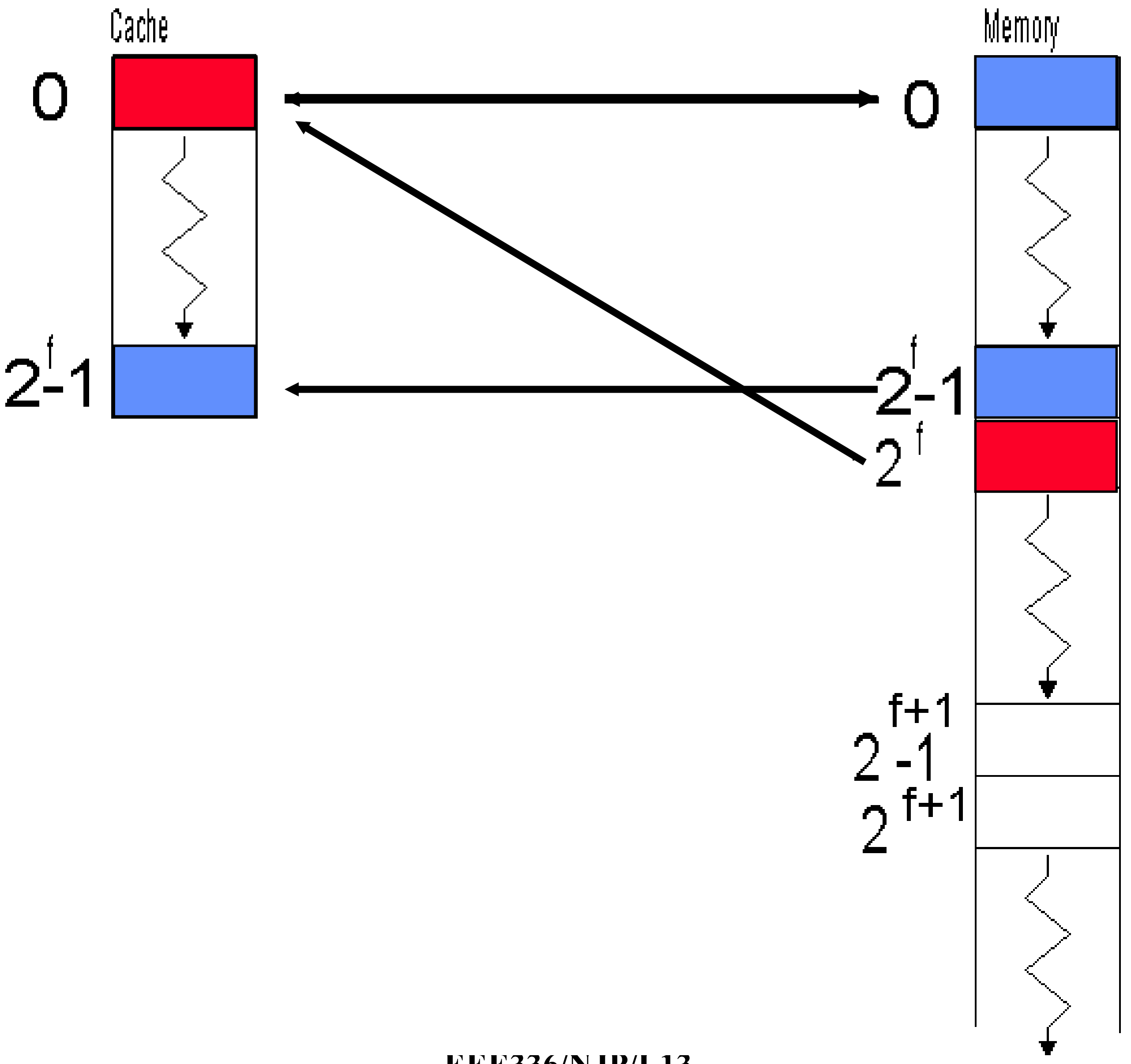
# How does it work

- Address to Mapper organised as follows

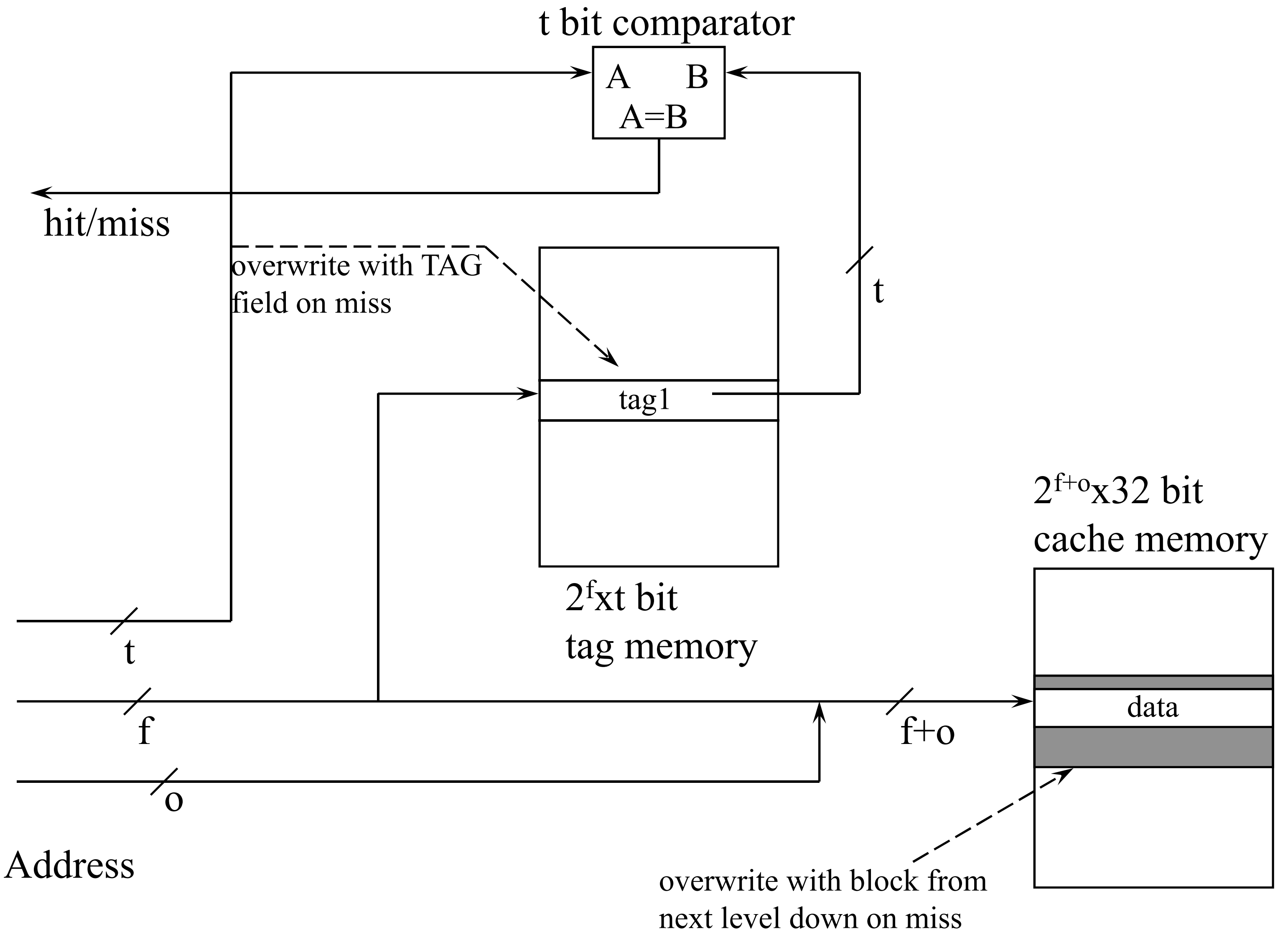
| | | |
|---|---|---|
| TAG | FRAME | OFFSET |

  - OFFSET is offset to the cache block and passes through unchanged
  - FRAME is address of block in cache and passes through unchanged
  - TAG is used to identify whether the requested block is in the cache

Copyright © by Luke Seed, September 2013



Copyright © by Luke Seed, September 2013



Copyright © by Luke Seed, September 2013

# Advantages/Disadvantages

- Low hardware overhead
- Fast on hit
  - read doesn' t add any time
  - On write overall access time is tag memory time + cache memory time
- Block from next level down can only be mapped into a *specific* block

Copyright © by Luke Seed, September 2013

# Fully Associative

- Block from next level down can be mapped into any block in the cache
  - Similar to Primary/Secondary memory interface

Copyright © by Luke Seed, September 2013

# Advantages/Disadvantages

- Very Flexible
- Does not suffer from problem due to blocks which cannot co-exist in the cache
- Needs full TLB to find block in cache
- Access time always incurs the overhead of accessing the TLB
- Big hardware overhead
- Needs memory management (RANDOM ?)

Copyright © by Luke Seed, September 2013

# Set Associative

- Combines advantages of both Direct Mapped and Fully Associative
  - Hardware is relatively simple
  - A Practical cache can be constructed which rivals the Fully Associative Cache in performance terms

Copyright © by Luke Seed, September 2013

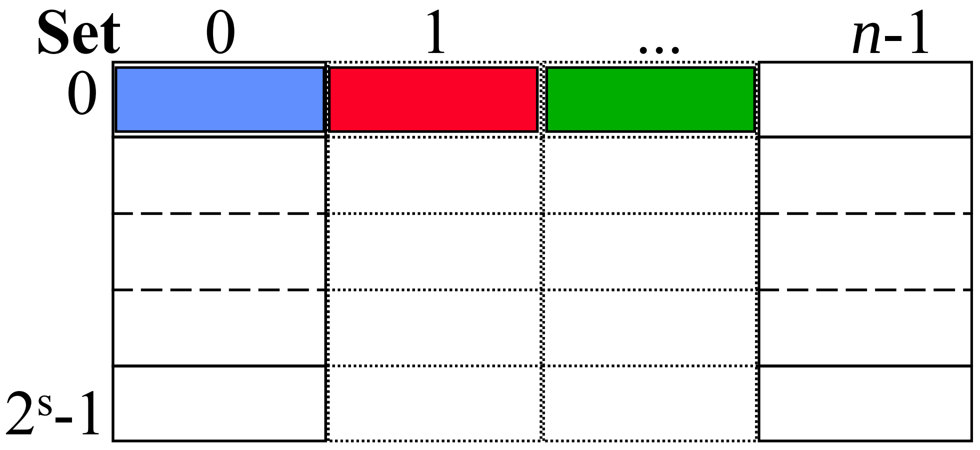
# How does it Work

- Address is as follows

| TAG | SET | OFFSET |
|---|---|---|

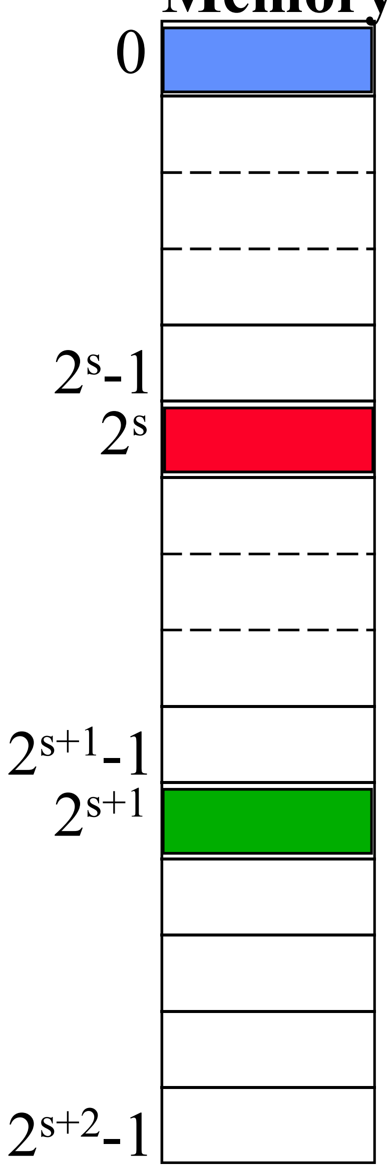
  - Cache is organised into sets
  - Each set contains a number of individual blocks or frames - say *n*
  - Cache is Direct Mapped with respect to sets (using SET and TAG fields)
  - Cache is Fully Associative within a set

Copyright © by Luke Seed, September 2013



Cache

Memory

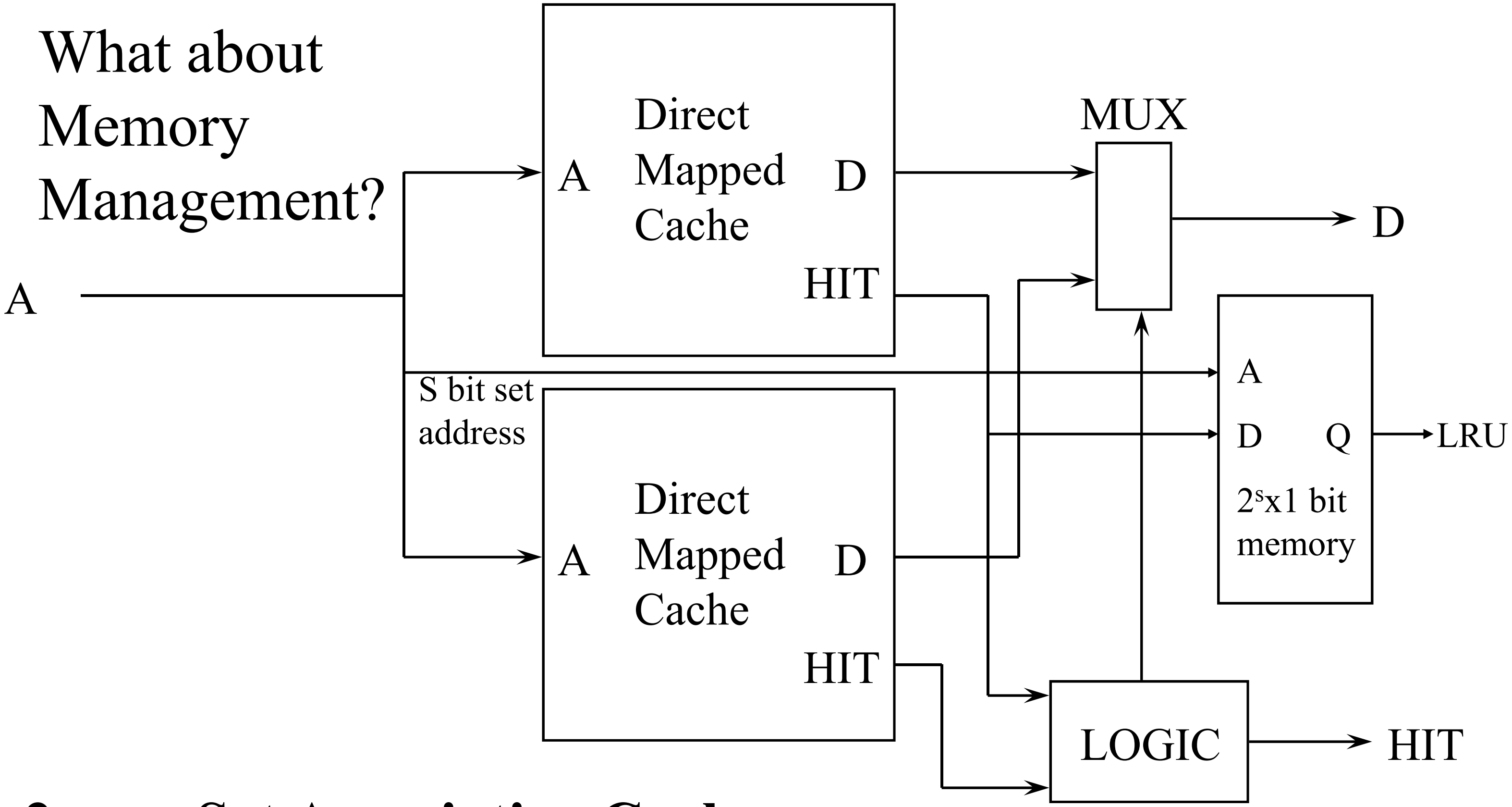


# n-way Set Associative Cache

Copyright © by Luke Seed, September 2013

# How is it Implemented?

What about Memory Management?



**2-way Set Associative Cache**

Copyright © by Luke Seed, September 2013

# Complexity w.r.t. *n*

- As n becomes bigger, the cache looks more like a Fully Associative Cache
- 2-way set associative is easy just 2 Direct Mapped Caches with a bit of extra hardware to select between them
- What about memory management?

Copyright © by Luke Seed, September 2013

# • Can you explain this?

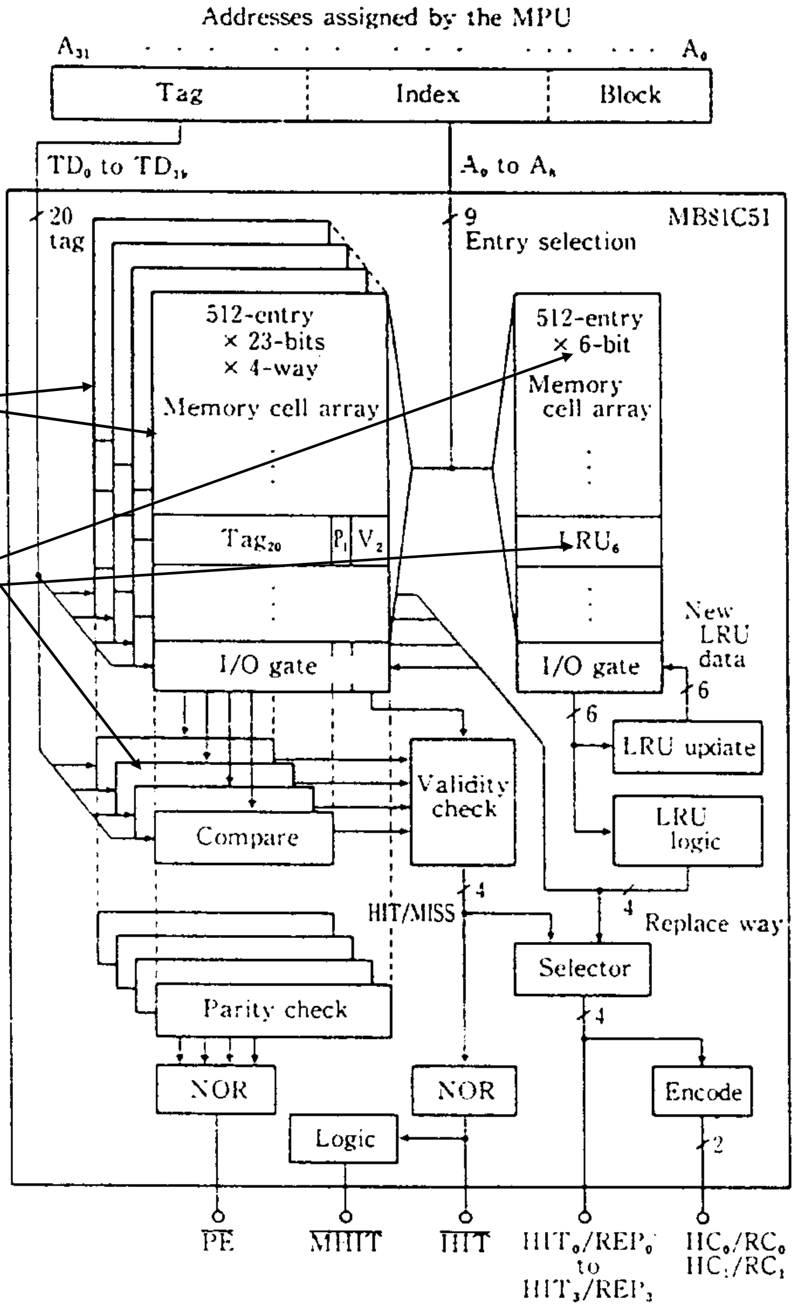
4-way SA (4 sets of LUTs)

Only one (or none) will HIT

LRU on each set

Why 6 bits / set

- 4 in each set
- 2 bit ID
- A Queue of 3 IDs is all that is necessary = 6 bits / set



Copyright © by Luke Seed, September 2013

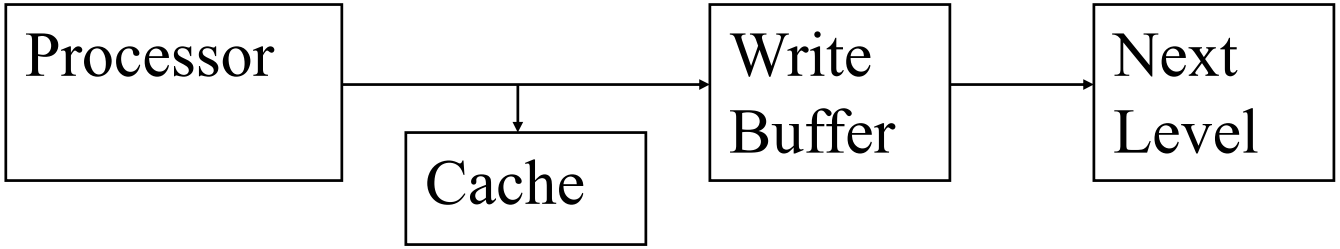
# Other Issues

- Can separate caches for program and data
  - Program cache is never written
  - Caches can be optimised for underlying behaviour of data
  - Disadvantage - decision about balance between program and data is predetermined
- Got to be able to invalidate cache contents
- Some memory must be *uncacheable*

Copyright © by Luke Seed, September 2013

# Writing to a Cache

- Two options on writing to data cache:
  - – Write back - When a block is discarded write it back to the next level down
  - – Write through - When a datum is written write it to the next level down as well (but don't wait for the write to complete



Copyright © by Luke Seed, September 2013

# Write-Through

- Means that blocks do not need to be written back but:
  - Can cause the processor to wait if a write is required before the previous write is finished
  - Two further options with write-through when there is a miss on a write
    - bring the block into the cache - With Allocate
    - don’t bring the block - With No Allocate

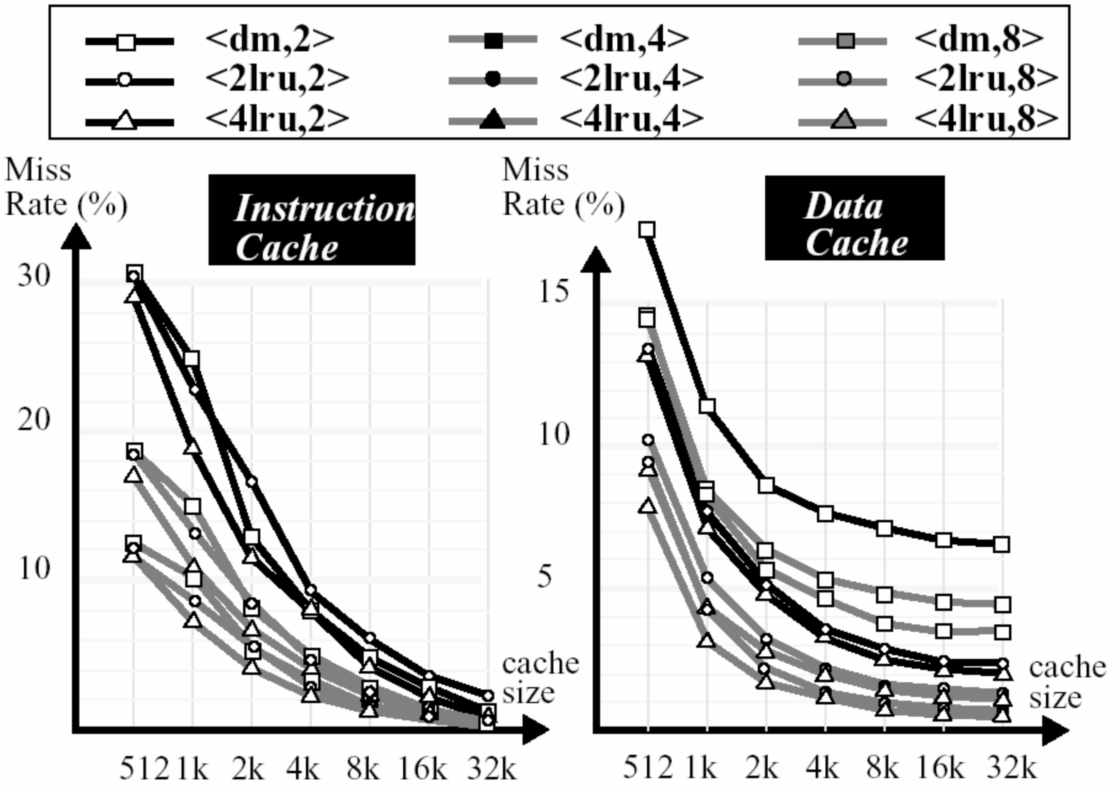
Copyright © by Luke Seed, September 2013

# Cache Performance

- There are analytical models but, generally, performance must be derived empirically for representative work-loads
- Performance depends on work-load, size, type, line-size, etc.

Copyright © by Luke Seed, September 2013

# Case Study (L1 Cache)



C-L Su, A M Despain, Cache Design Trade-Offs for Power and Performance Optimisation: A Case Study. Proc. ISLPED, pp63-68, 1995

Copyright © by Luke Seed, September 2013

# Execution Trace Cache

- Intended for instructions
- They do not store an image of the memory
- They store the sequence of instructions that have been executed
- This implies a greater efficiency
- Instructions can be partly decoded before being put in the TEC

Copyright © by Luke Seed, September 2013

# Example

- Code snippet

```
L1  ● move x,y
    ● cmp y,#0
    ● je L2
      inst 1
      inst 2
      inst 3
L2  ● cmp y,#9
    ● add x,#1
    ● jlt L3
      inst 4
      inst 5
L3  ● in x,23
    ● jmp L1
```

Normal cache contents

| | | | |
|---|---|---|---|
| ● move x,y | ● cmp y,#0 | ● je L2 | inst 1 |
| inst 2 | inst 3 | ● cmp y,#9 | ● add x,#1 |
| ● jlt L3 | inst 4 | ● in x,23 | ● jmp L1 |

Execution trace cache contents

| | | | |
|---|---|---|---|
| ● move x,y | ● cmp y,#0 | ● je L2 | ● cmp y,#9 |
| ● add x,#1 | ● jlt L3 | ● in x,23 | ● jmp L1 |