The University for world-class professionals

Manchester Metropolitan University

Machine Dependent Optimisation and Cache Architectures

Dr Paris Yiapanis

room: John Dalton E151

email: p.yiapanis@mmu.ac.uk

Where we are…

- ~~Admin and overview~~
- ~~Lexical analysis~~
- ~~Parsing~~
- ~~Semantic analysis~~
- ~~Machine-independent optimisation~~
- ~~Code generation~~
- ~~Hardware architectures~~
- **Machine-dependent optimisation**
- Review

Objectives

- Discuss further compiler optimisations
  - Instruction percolation
  - Loop unrolling
  - Loop fusion
  - etc
- Discuss ways compilers can optimise cache usage

Pipelining and Parallelisation

- Last week we looked at pipelining and parallelisation for different CPU architectures
- Some architectures use dynamic scheduling, others use static
- For static scheduling, code optimisation is essential
- For dynamic scheduling, code optimisation can facilitate further optimisation

Instruction Scheduling

- We would like to reorder the instructions in a way which
  - preserves the dependencies between those instructions (and hence the correctness of the program), and
  - achieves the minimum possible number of pipeline stalls
  - instruction Percolation
- We've seen an example last week

Loop manipulation for Parallelism

- Two common techniques for taking advantage of parallelism
  - Loop unrolling
  - Loop fusion

Loop Unrolling

- Make data independencies *explicit* through rewriting loop to do more work in each iteration
- Basic principle:
  - Change

```
for (i = 0; i < n; i++) {
        do something related to i
}
```

  - To

```
for (i = 0; i < n; i+=4) {
        do something related to i
        do something related to i+1
        do something related to i+2
        do something related to i+3
}
process remaining items /* if n not divisible by 4 */
```

This particular change is useful if we have *four* degrees of parallelism.

Loop Fusion

- Turn two (or more) loops into one
- Basic idea:
  - Change

```
for (i = 0; i < n; i++) {
        do somethingA related to i
}
for (i = 0; i < n; i++) {
        do somethingB related to i
}
```

  - To

```
for (i = 0; i < n; i++) {
        do somethingA related to i
        do somethingB related to i
}
```

This particular change is useful if we have *two* degrees of parallelism.

Branching Decisions

- If we have a choice, can speculatively execute *both* outcomes (assuming they are independent)
- Basic idea:
- Execute

```
if (x) {
        do a
} else {
        do b
}
```

as

```
test x
do a IN PARALLEL WITH do b
```

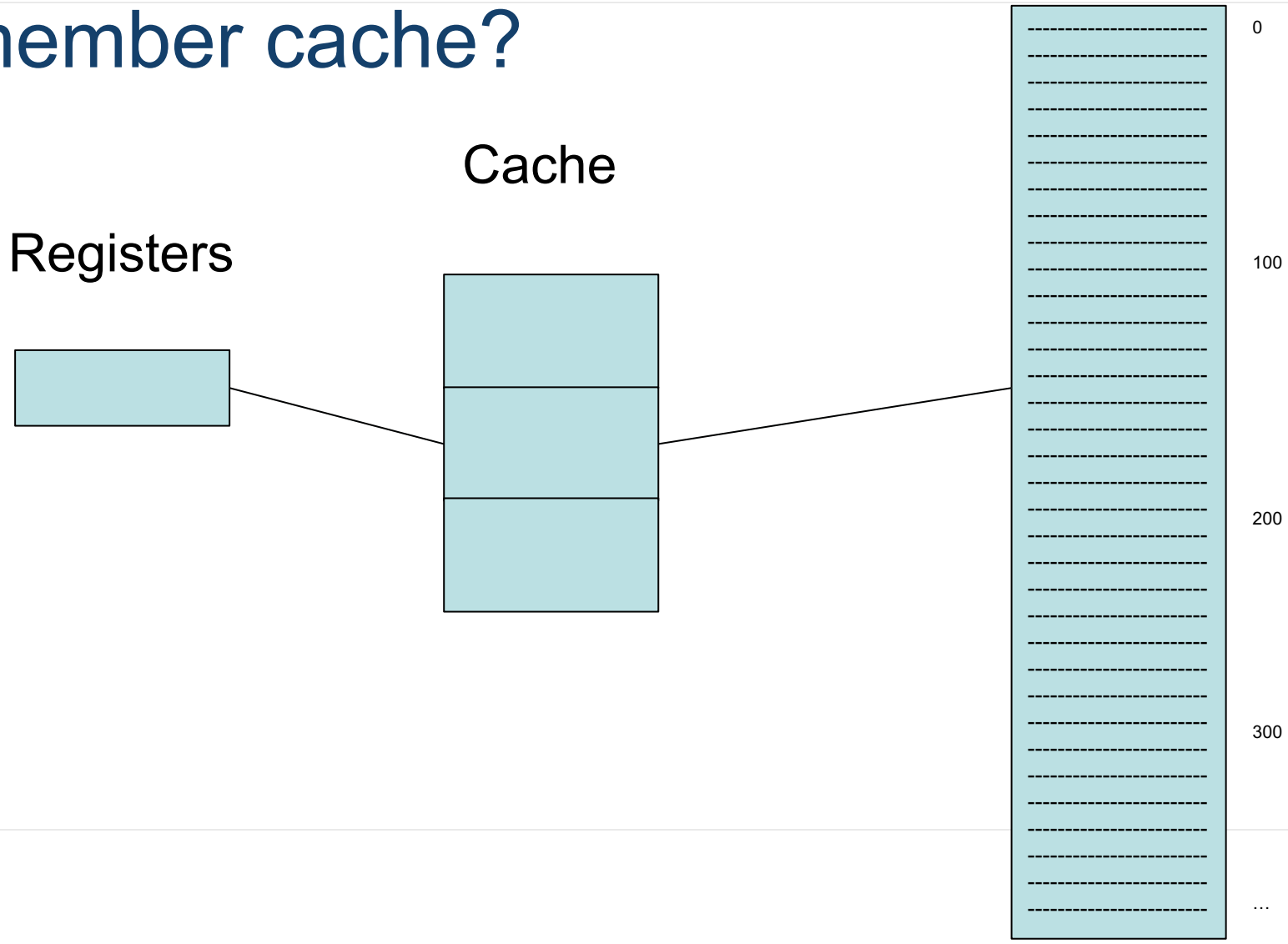
(throw away unneeded results)

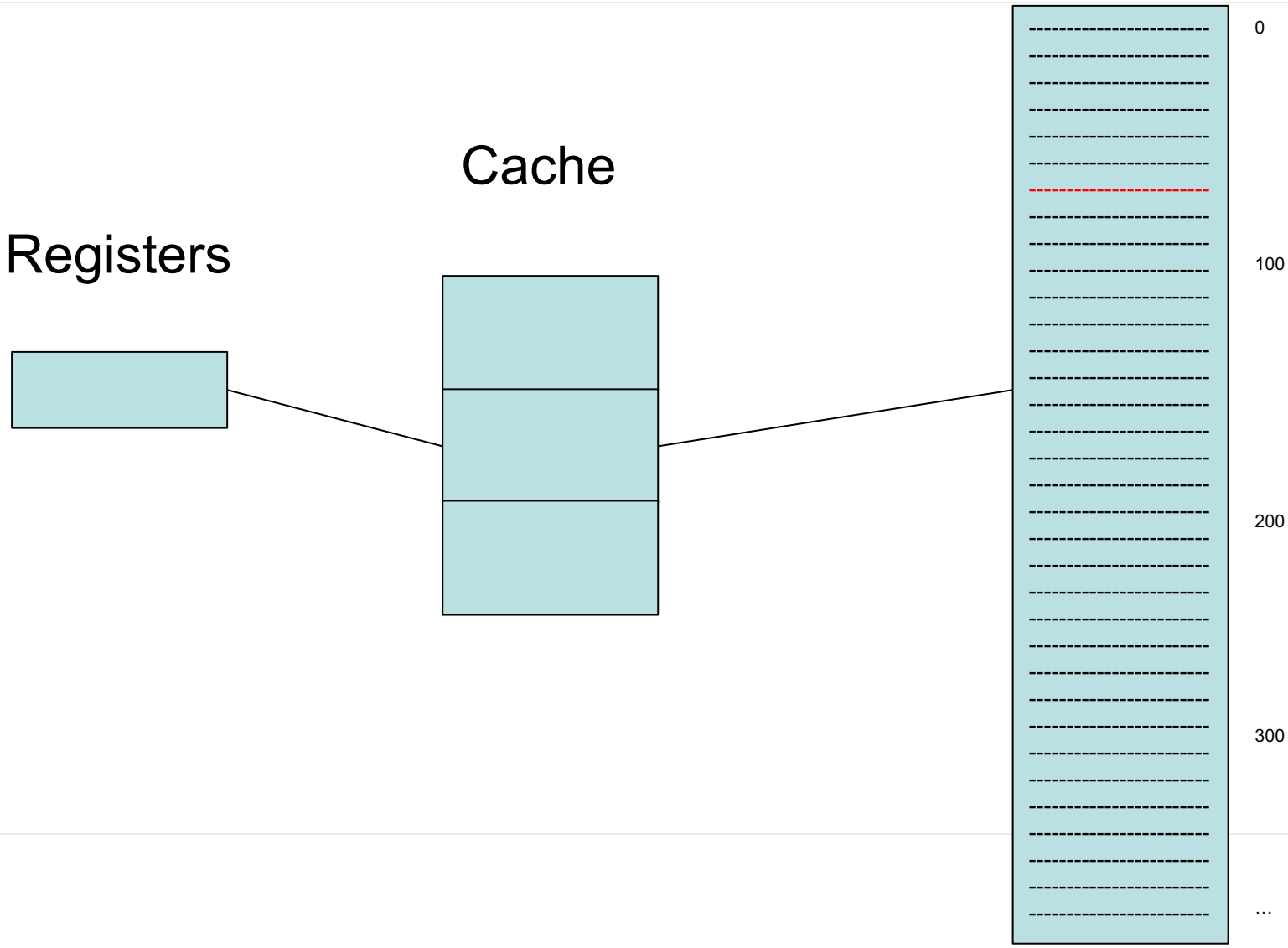
This gives *two* degrees of parallelism; more can be achieved by discovering opportunities within *do a* and *do b*.

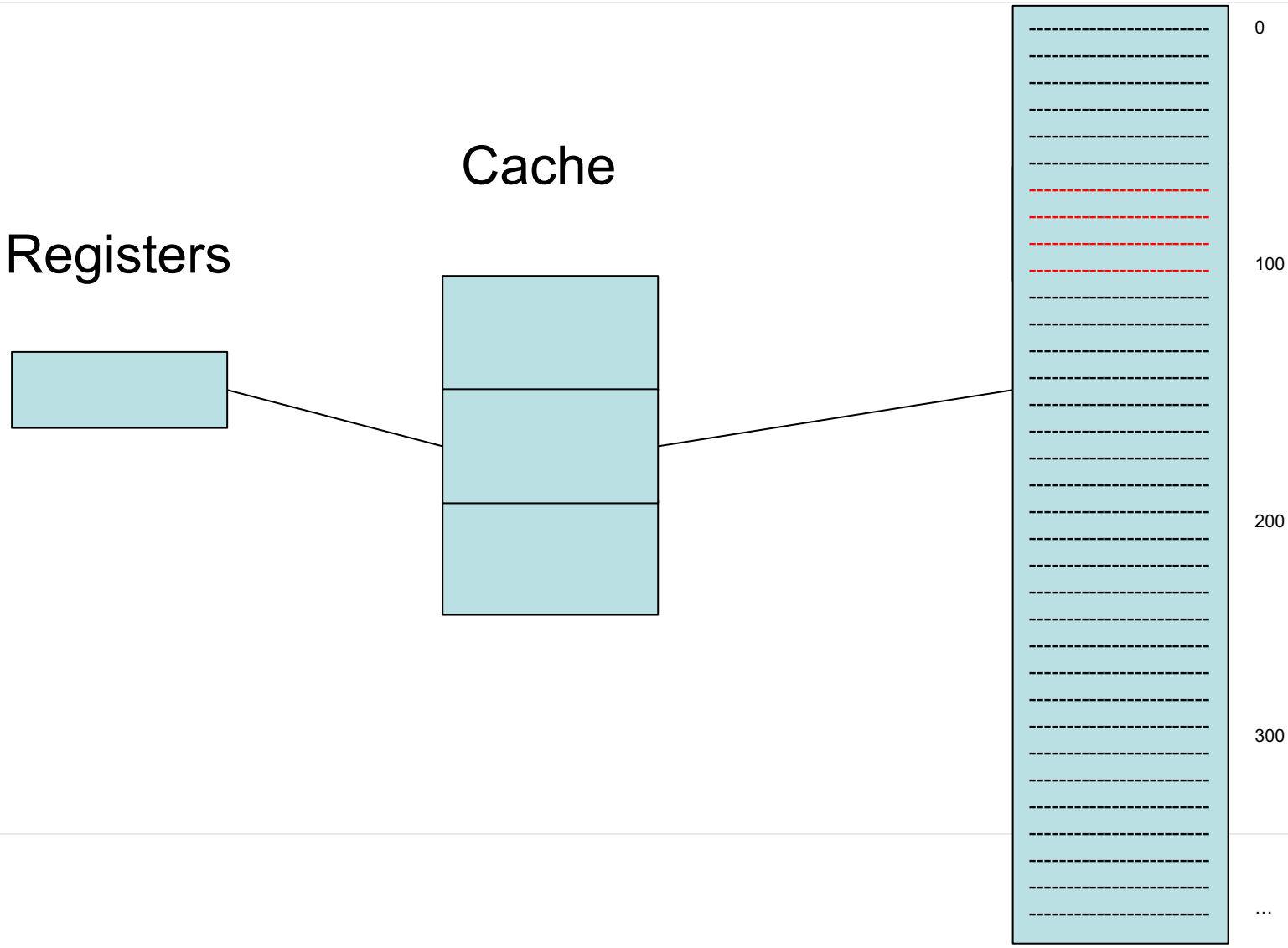
Cache Optimisation

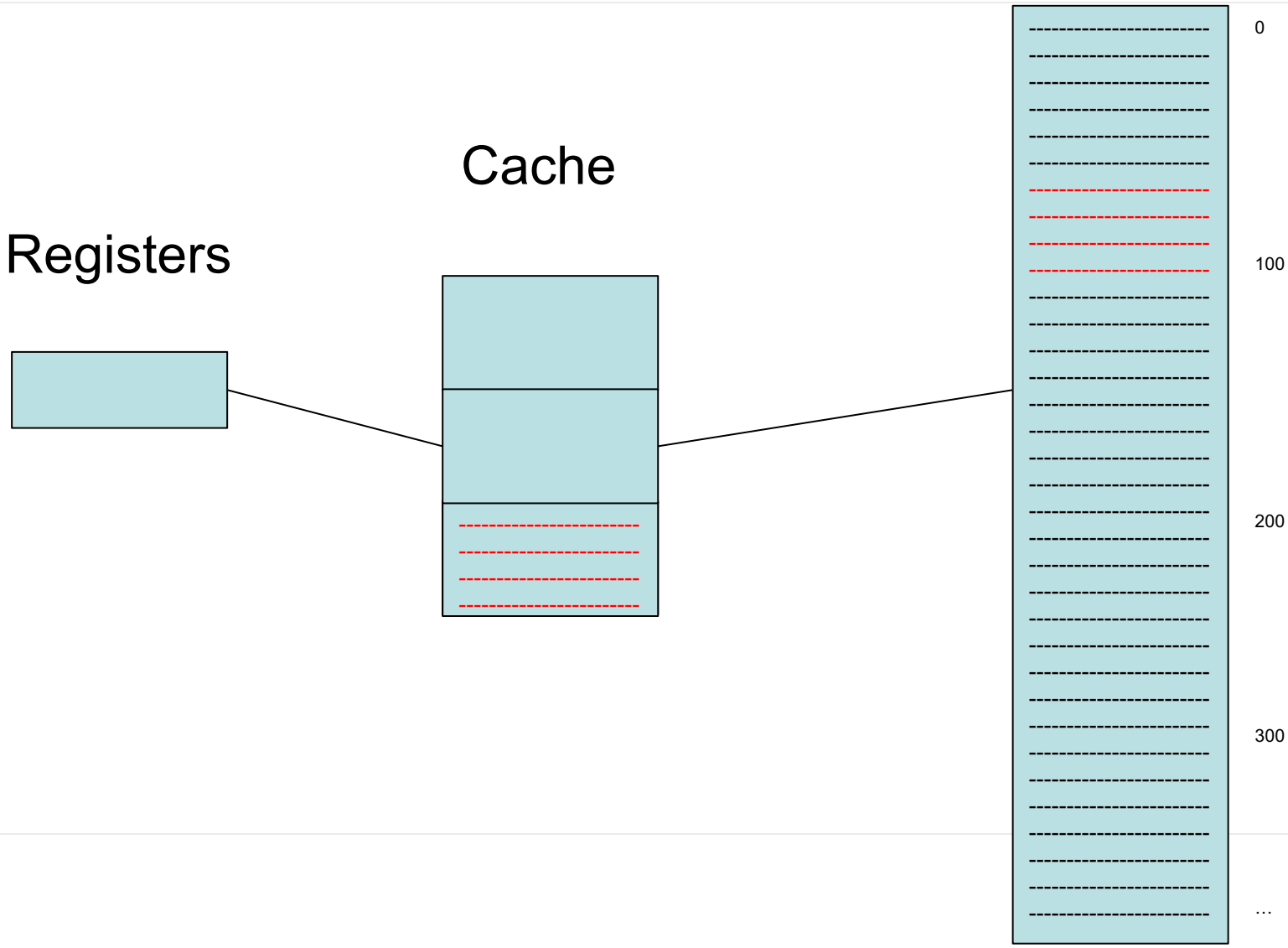
- Try to increase the chances that the needed memory locations are already in cache
- Reorder data accesses to improve data locality
- Cache is divided into **cache lines**
  - When memory is read into cache, a whole cache line is always read at the same time
  - Good if we have data locality: nearby memory accesses will be fast
  - Typical cache line size: 64-128 bytes (compiler needs to know exactly if it is going to optimise)

remember cache?




Main
Memory
Cache
Registers
0
100
200
300
…


Main
Memory
Cache
Registers
0
100
200
300
…


Main
Memory
Cache
Registers
0
100
200
300
…

Blocking or Tiling

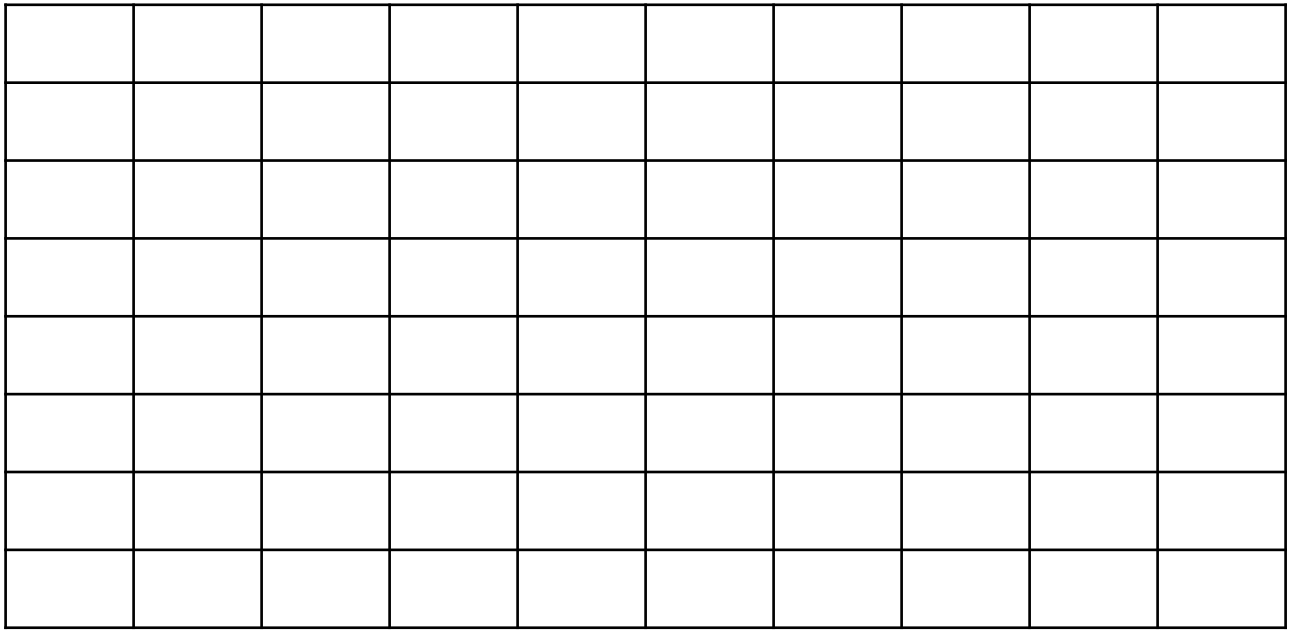
- Change

```
for (i = 0; i < n; i++)
    do something
```

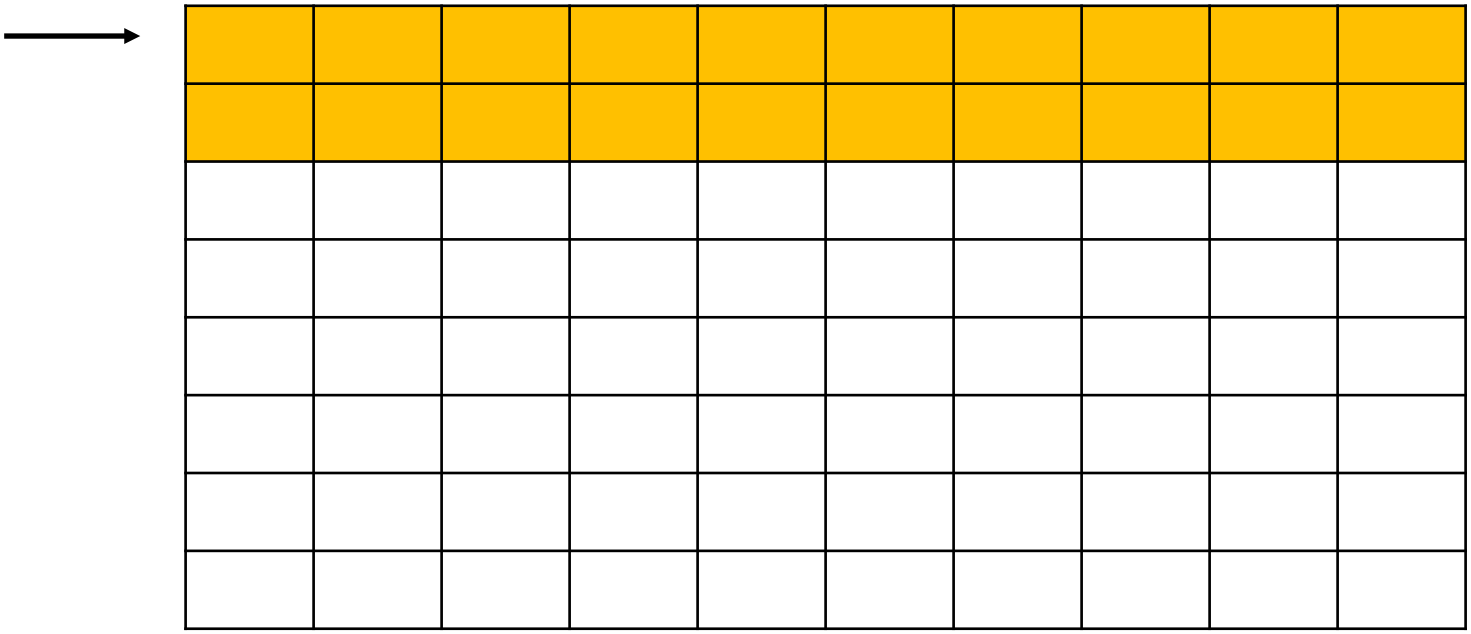
To

```
for (i = 0; i < n; i += B)
    for (j = i; j < min(N, i+B); j++)
        do something
```

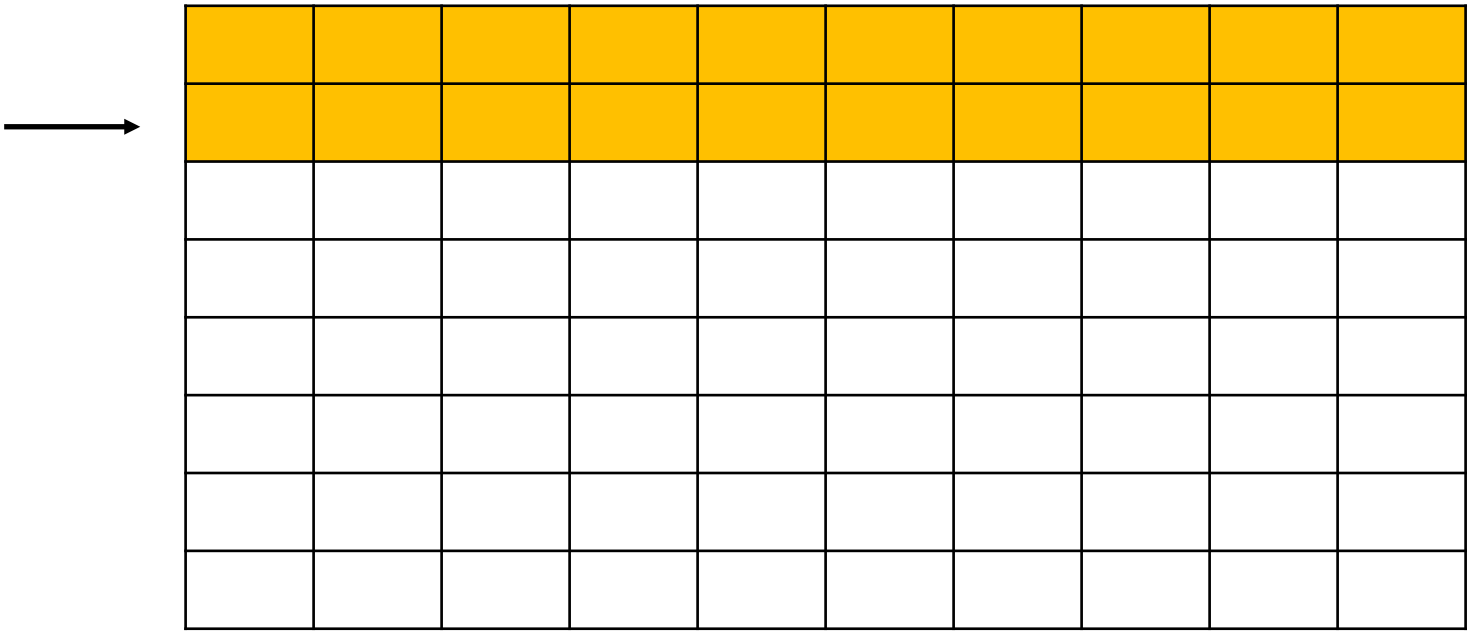
Blocking or Tiling



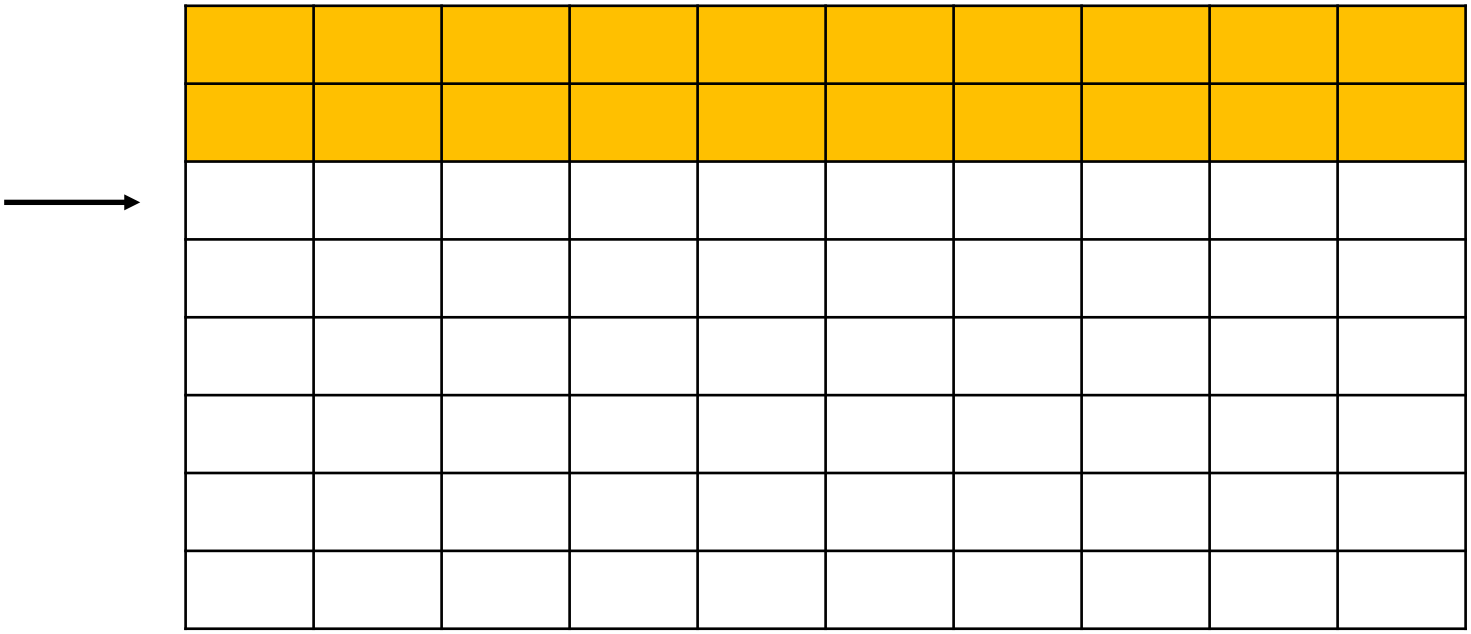
Blocking or Tiling



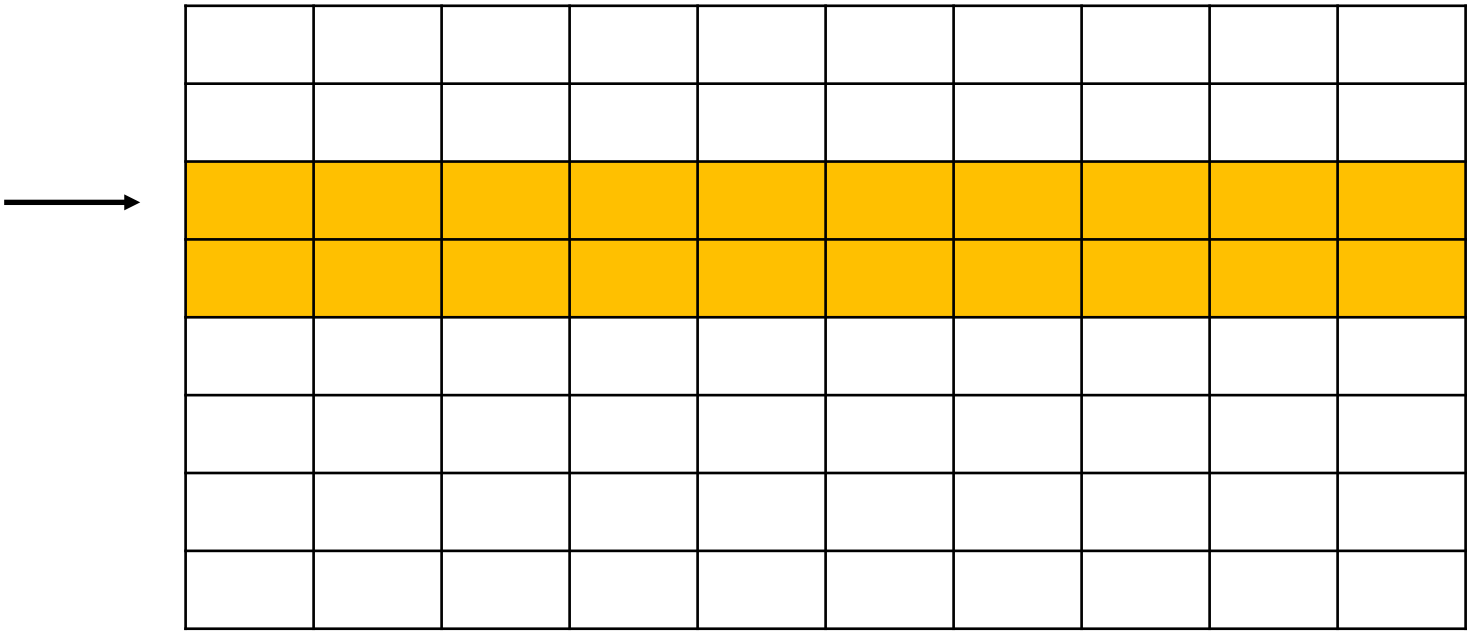
Blocking or Tiling



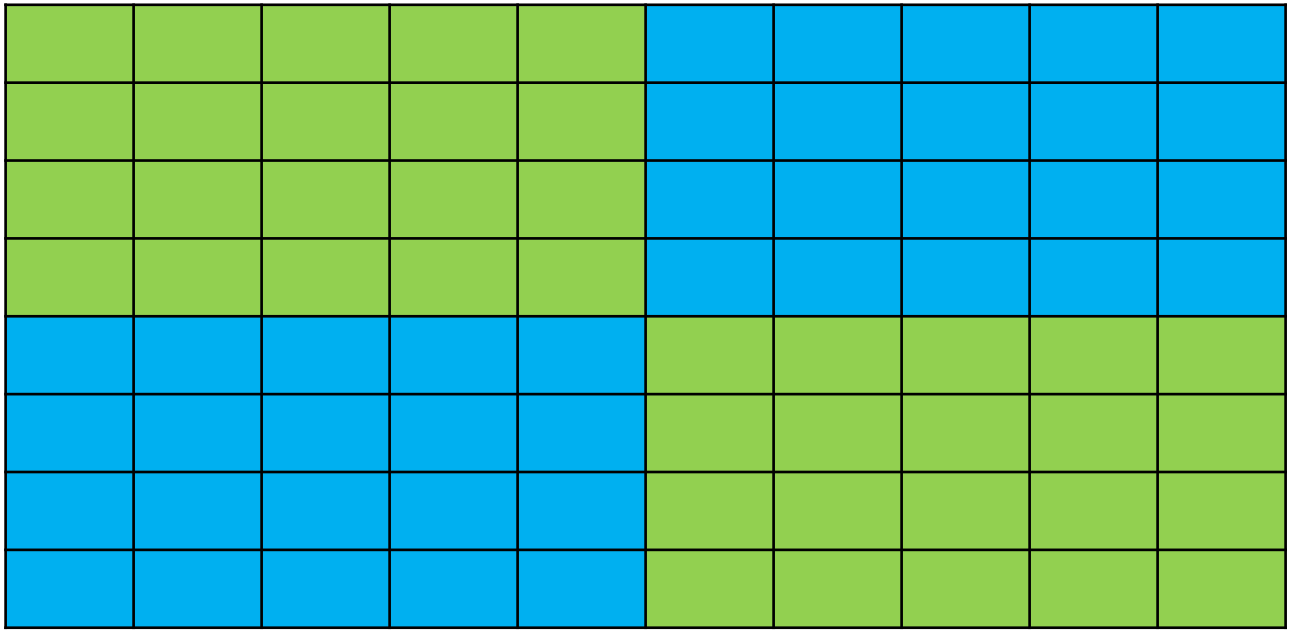
Blocking or Tiling



Blocking or Tiling



Blocking or Tiling



Summary

- Techniques to take advantage of some machine-dependent features for optimisation
  - Optimisation for pipelining
    - Instruction percolation
    - Loop unrolling
    - Loop fusion
  - Optimisation for cache
    - Blocking

Where we are…

- ~~Admin and overview~~
- ~~Lexical analysis~~
- ~~Parsing~~
- ~~Semantic analysis~~
- ~~Machine-independent optimisation~~
- ~~Code generation~~
- ~~Hardware architectures~~
- ~~Machine-dependent optimisation~~
- Review