Computer Architecture
Crash course

Frédéric Haziza <daz@it.uu.se>

Department of Computer Systems

Uppsala University

Summer 2009

Conclusions

- The multicore era is already here
- cost of parallelism is dropping dramatically
  - Cache/Thread is dropping
  - Memory bandwidth/thread is dropping
- The introduction of multicore processors will have profound impact on computational software

> *"For high-performance computing (HPC) applications, multicore processors introduces an additional layer of complexity, which will force users to go through a phase change in how they address parallelism or risk being left behind."*

[IDC presentation on HPC at International Supercomputing Conference, ICS07, Dresden, June 26-29 2007]

Why multiple cores/threads?
(even in your laptops)

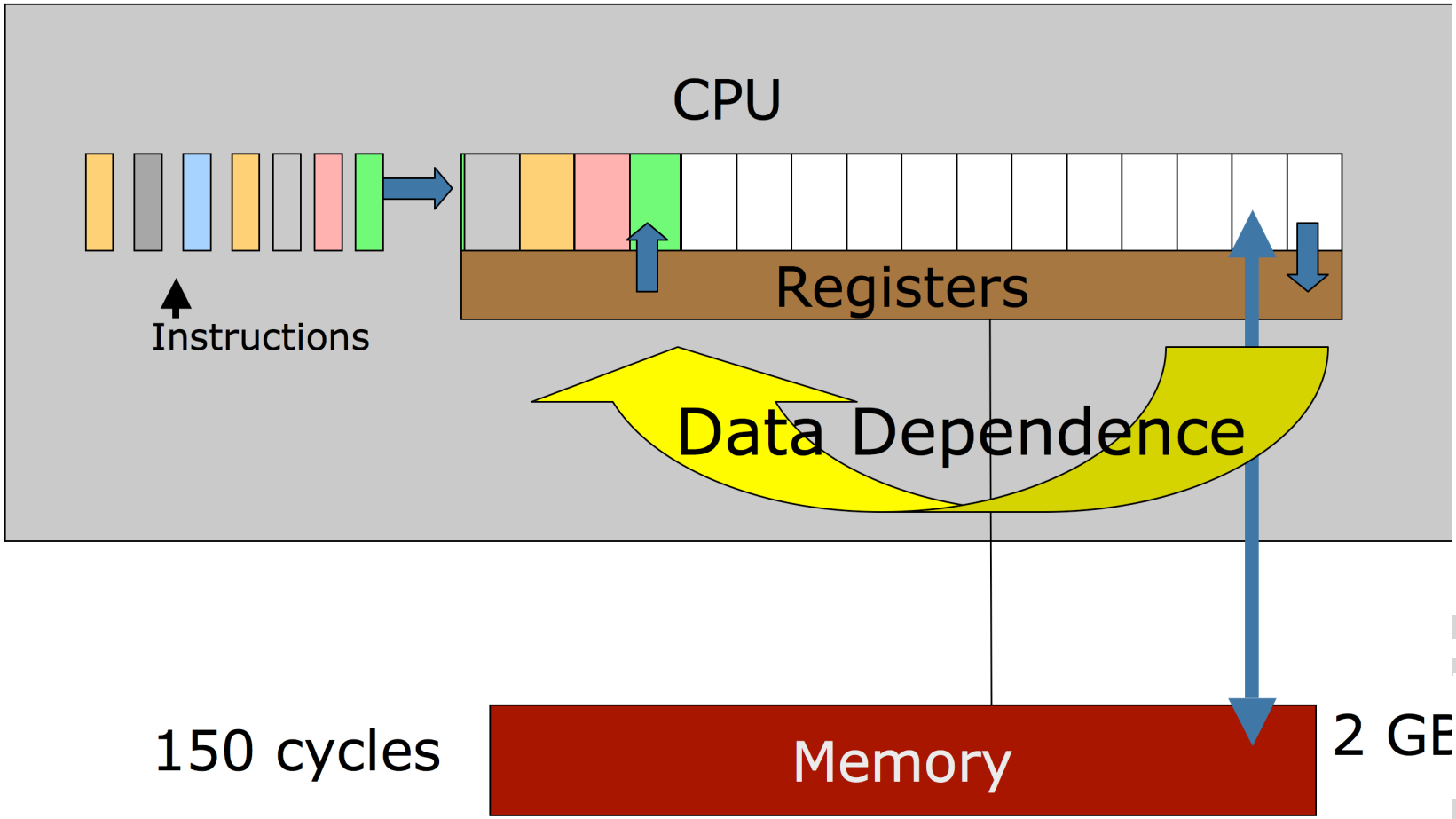
- Instruction level parallelism is running out
- Wire delay is hurting
- Power dissipation is hurting
- The memory latency/bandwidth bottleneck is becoming even worse

Focus: Performance

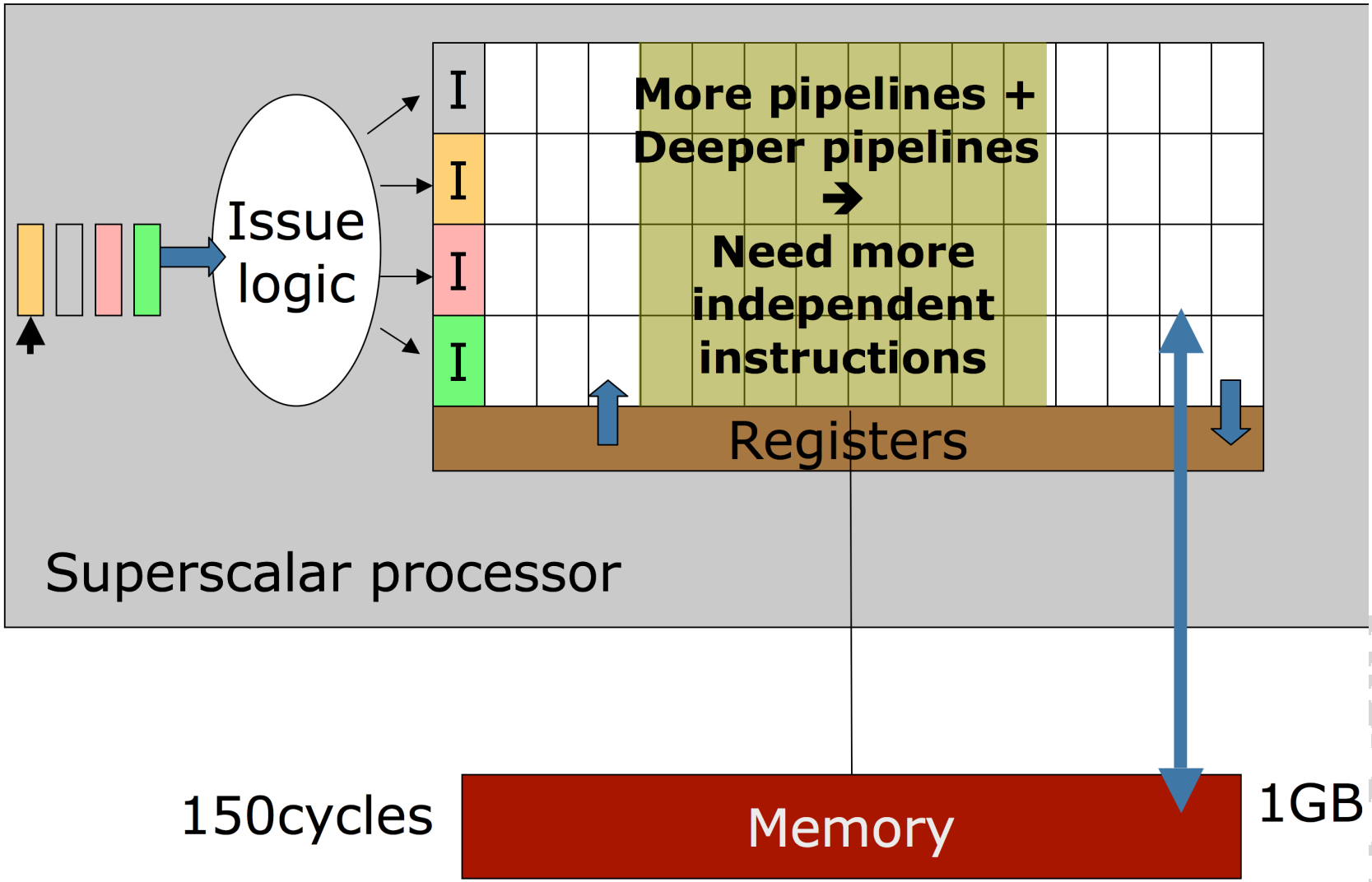
Create and explore:

1. Parallelism
   - Instruction level parallelism (ILP)
   - In some cases: Parallelism at the program/algorithm level ("Parallel computing")
2. Locality of data
   - Caches
   - Temporal locality
   - Cache blocks/lines: Spatial locality

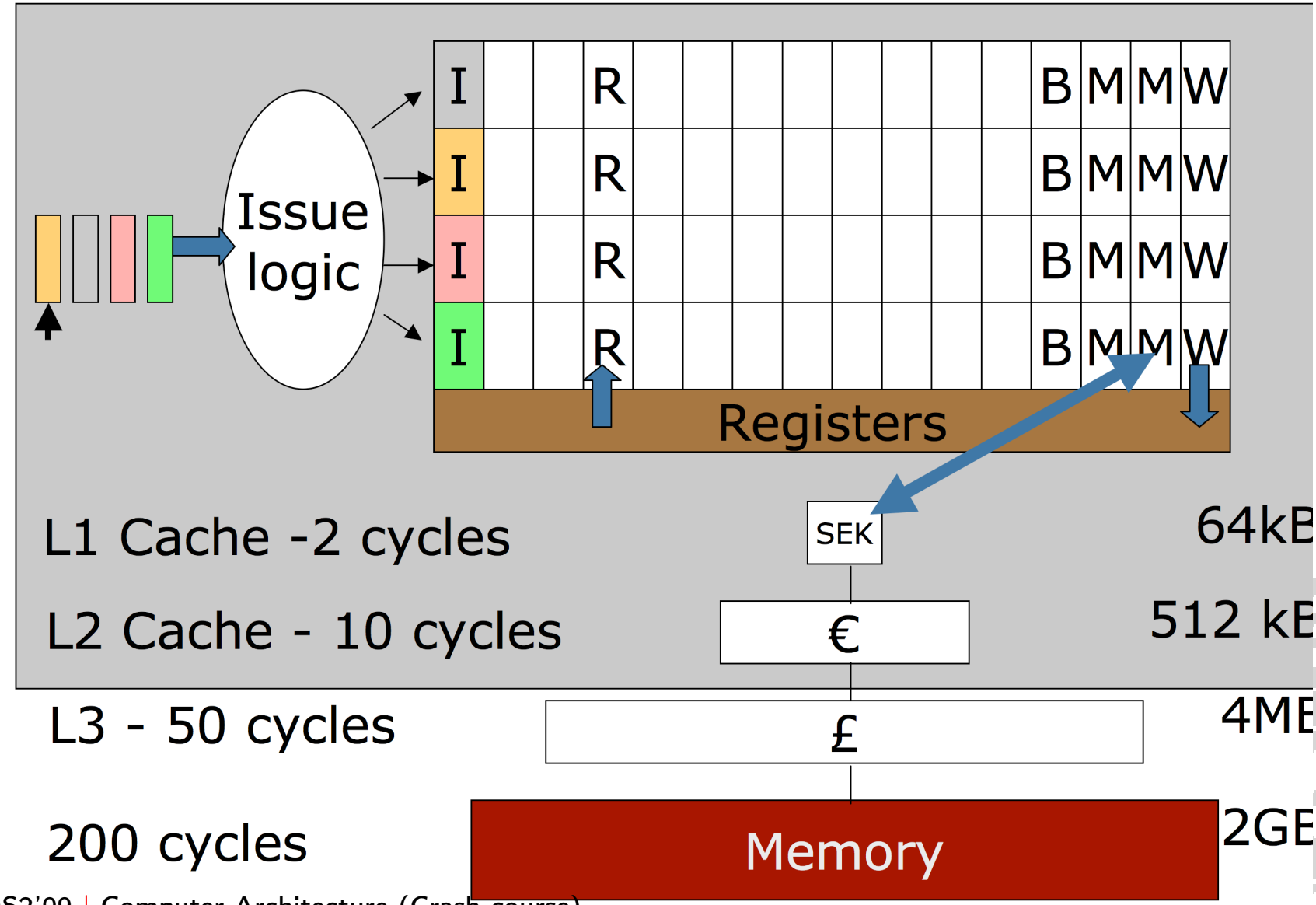
Old Trend 1: Deeper pipelines
(exploring ILP)



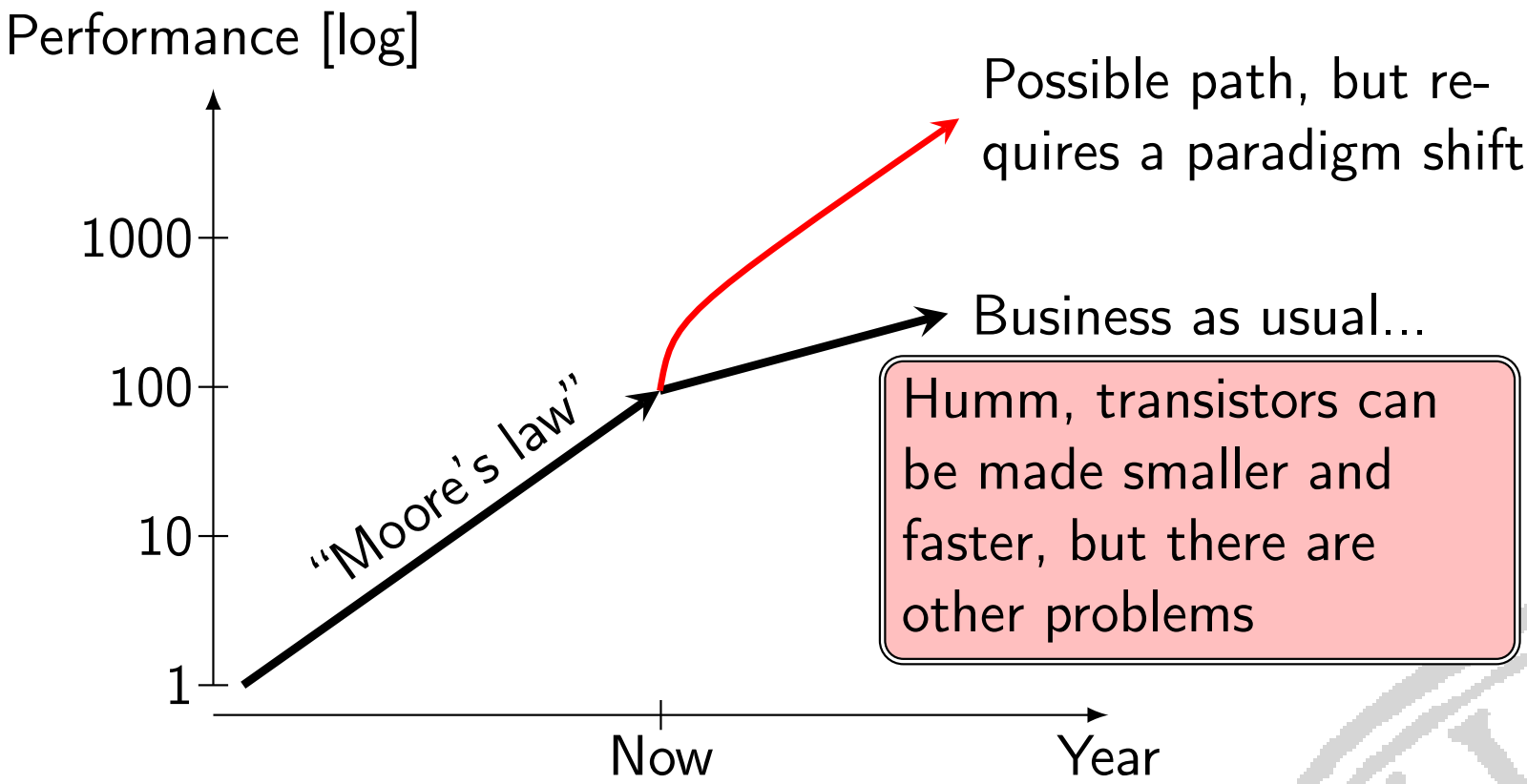
Old Trend 2: Wider pipelines
(exploring more ILP)



Old Trend 3: Deeper memory hierarchy
(exploring locality of data)



Are we hitting the wall now?



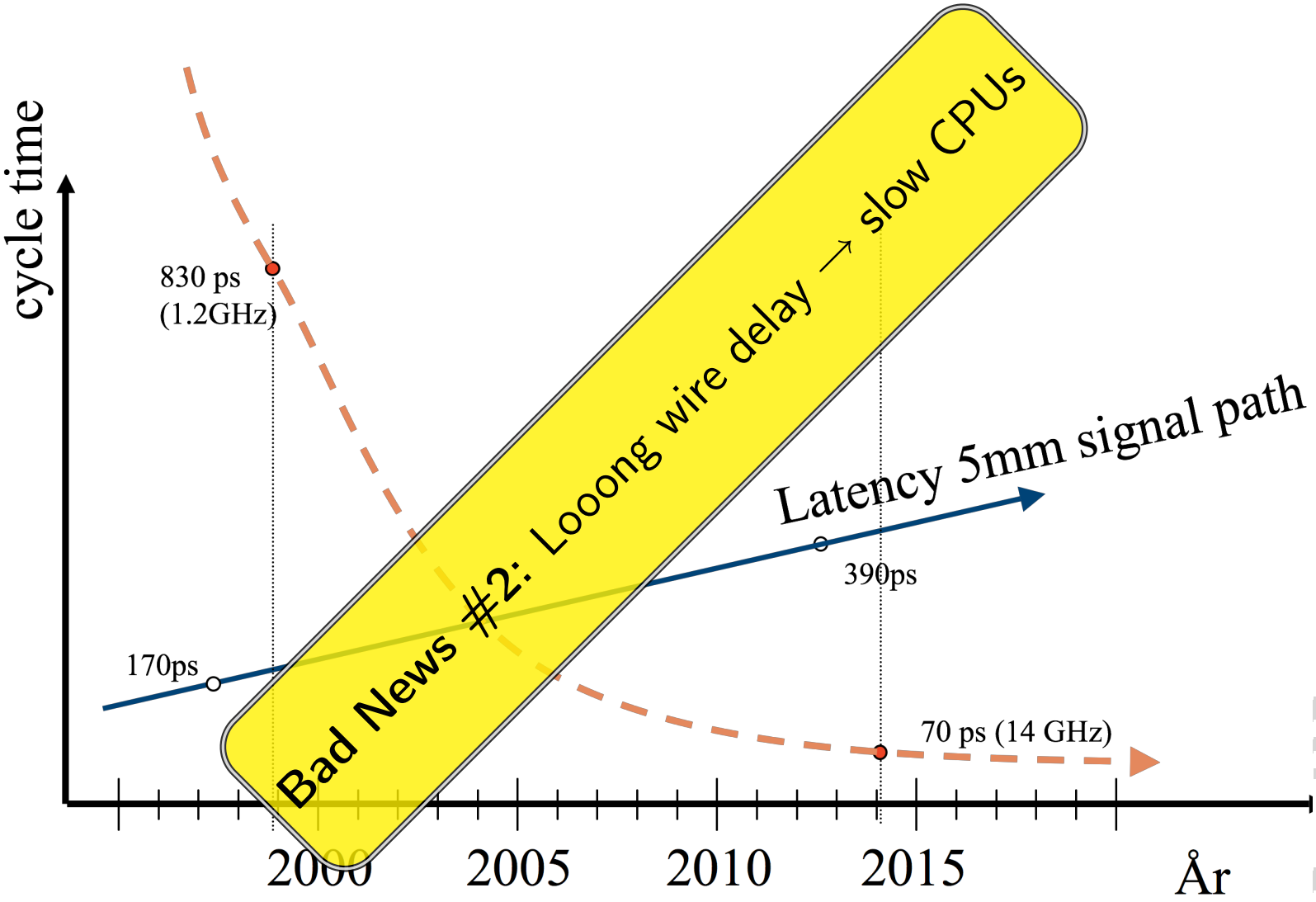
Classical microprocessors: Whatever it takes to run one program fast

Exploring ILP (instruction-level parallelism)

- Faster clocks → Deep pipelines
- Superscalar Pipelines
- Branch Prediction
- Out-of-Order Execution
- Trace Cache
- Speculation
- Predicate Execution
- Advanced Load Address Table
- Return Address Stack
- ...
-

Bad News #1: Already explored most ILP

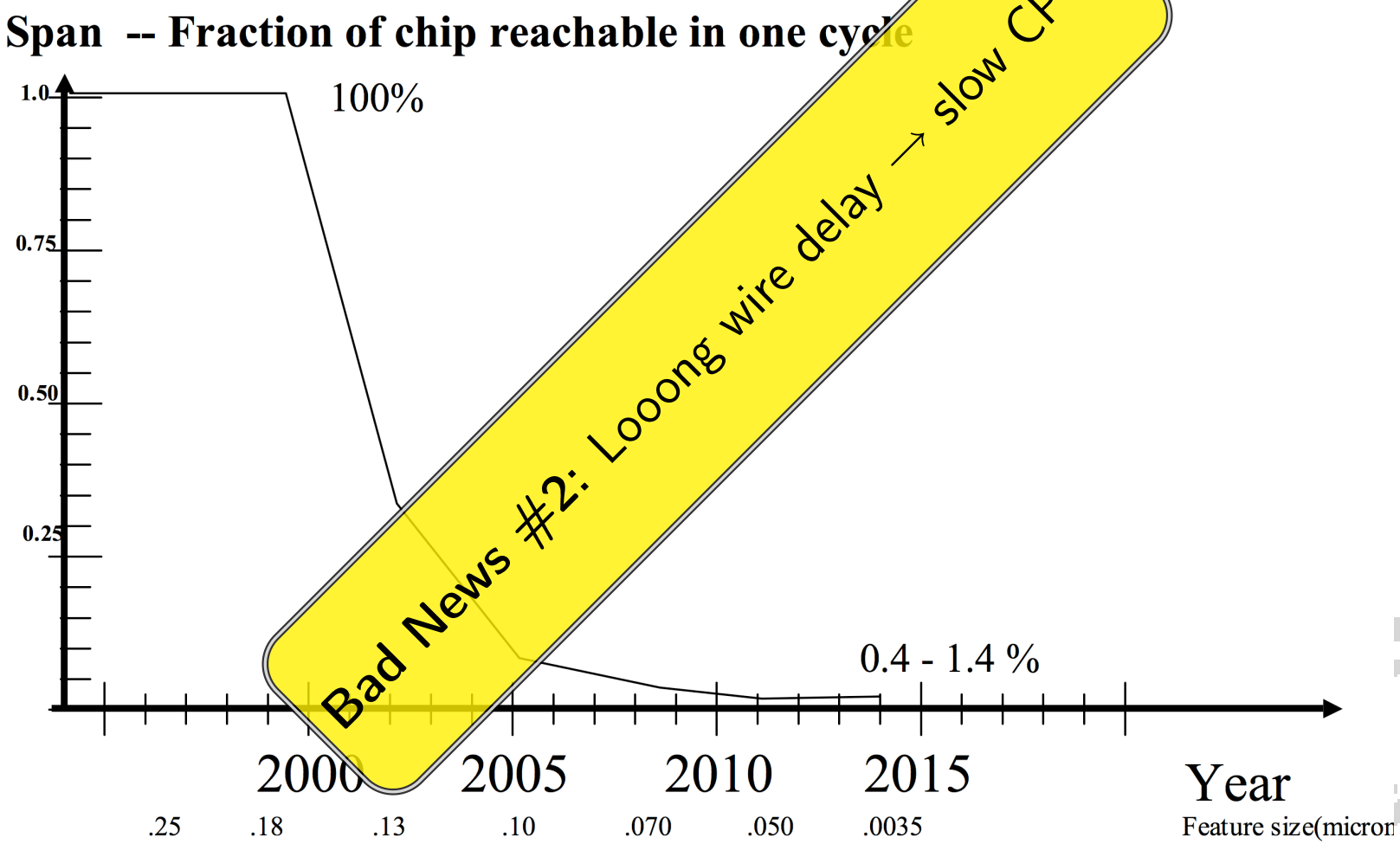
#2: Future technology



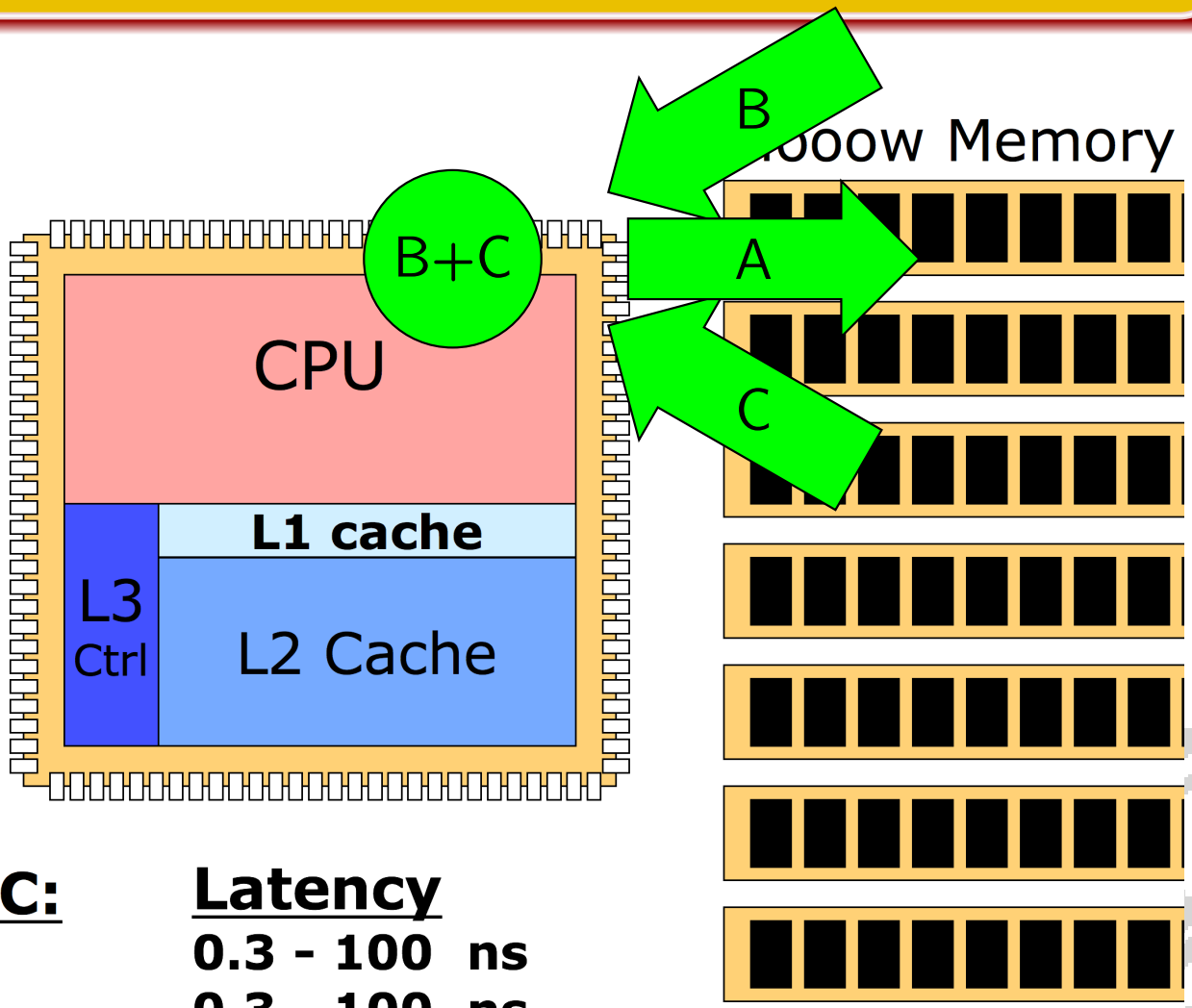
Quantitative data and trends according to V. Agarwal et al., ISCA 2000

***Based on SIA* (Semiconductor Industry Association) prediction, 1999**

#2: How much of the chip area can you reach in one cycle?



Bad News #3: Mem latency/bandwidth



A = B + C:	Latency
Read B	0.3 - 100 ns
Read C	0.3 - 100 ns
Add B & C	0.3 ns
Write A	0.3 - 100 ns

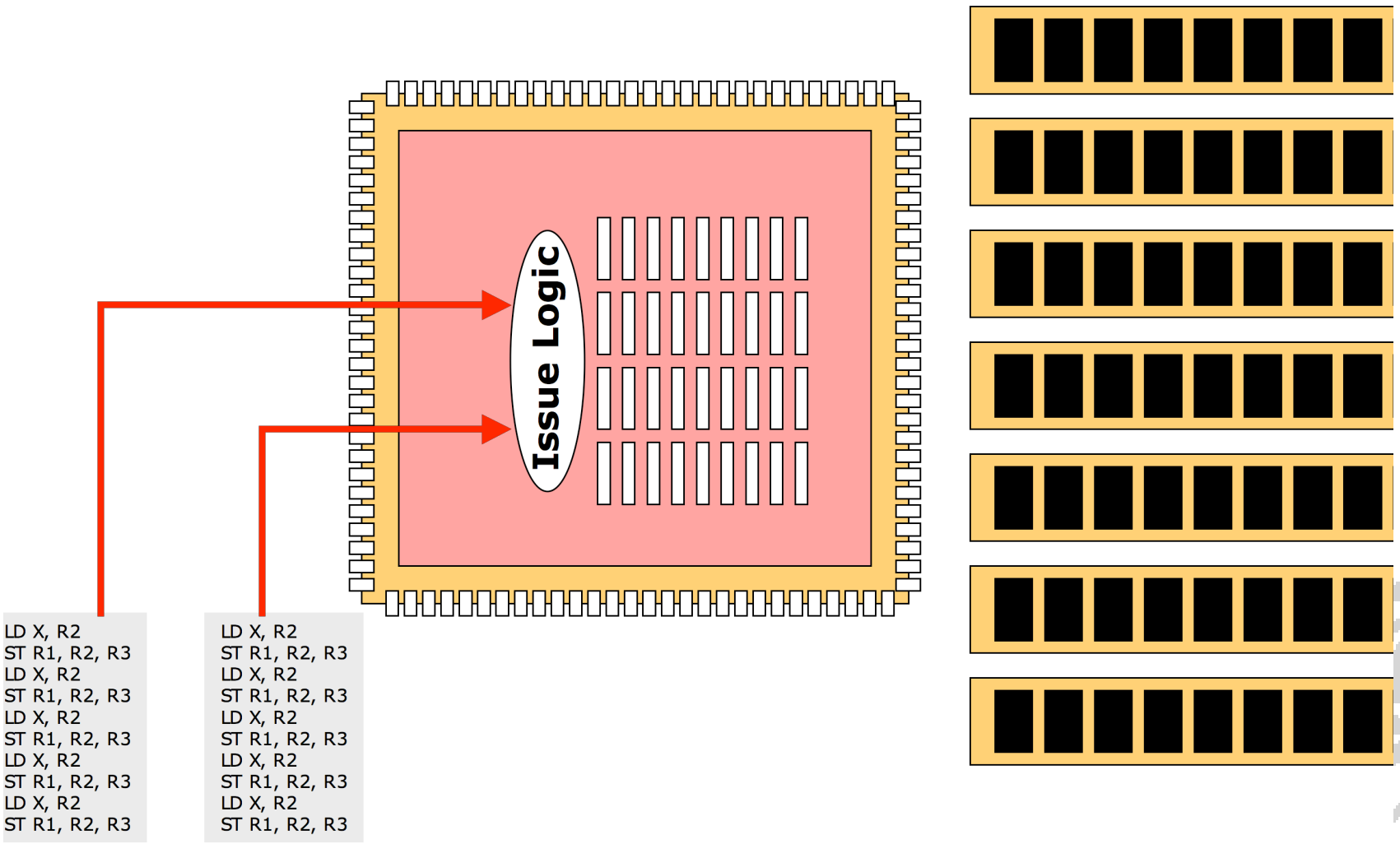
Bad News #4: Power is the limit

- Power consumption is the bottleneck
  - Cooling servers is hard
  - Battery lifetime for mobile computers
  - Energy is money
- Dissipated effect is proportional to
  - ~ Frequency
  - ~ Voltage^2

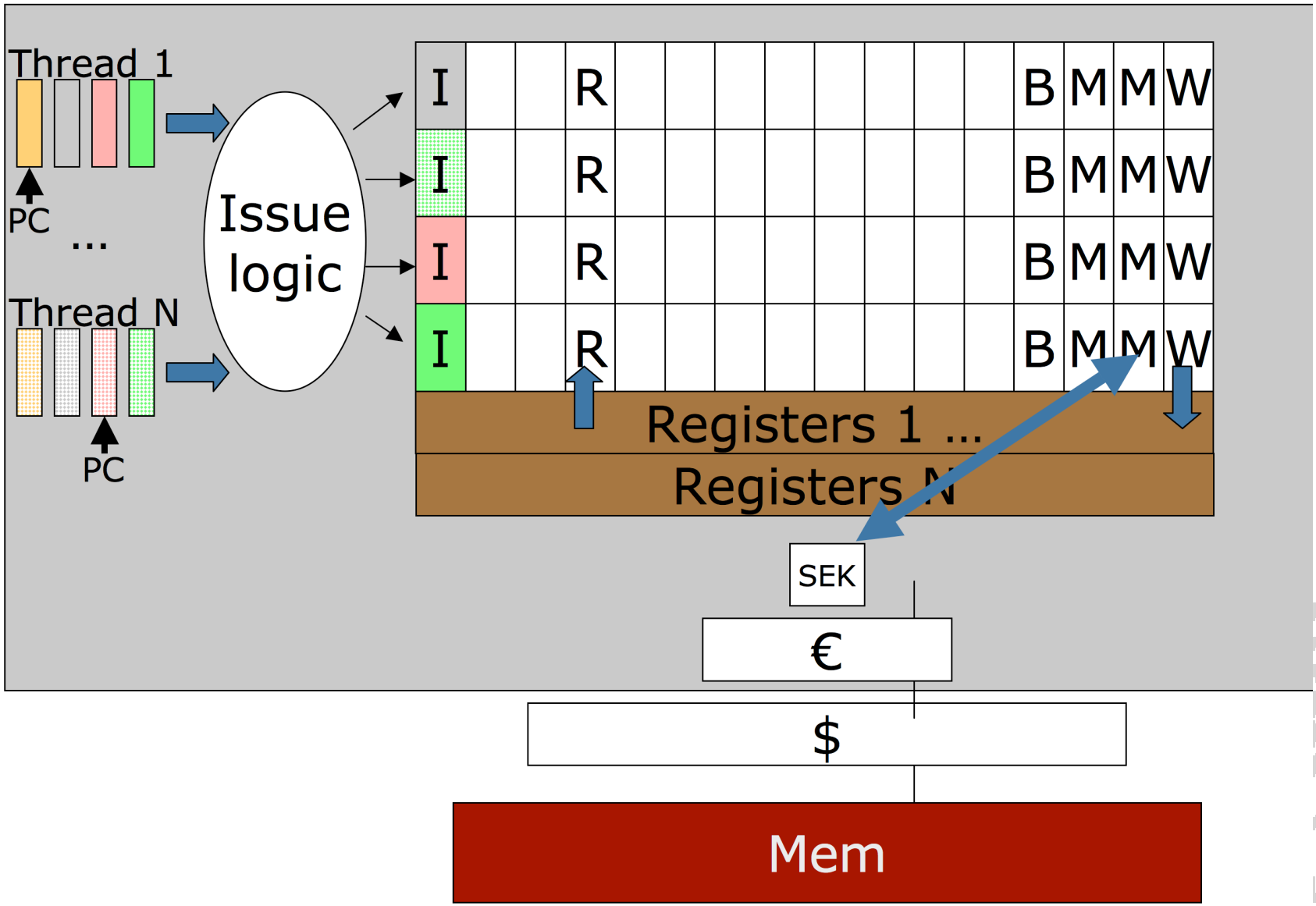
1. **Why mutliple threads/cores?**
   - Old Trends
   - Bad news
   - Solutions?
2. Introducing concurrency
   - Scenario
   - Definitions
   - Amdhal's law
3. Can we trust the hardware?

Bad News #1: Not enough ILP

→ feed one CPU with instr. from many threads

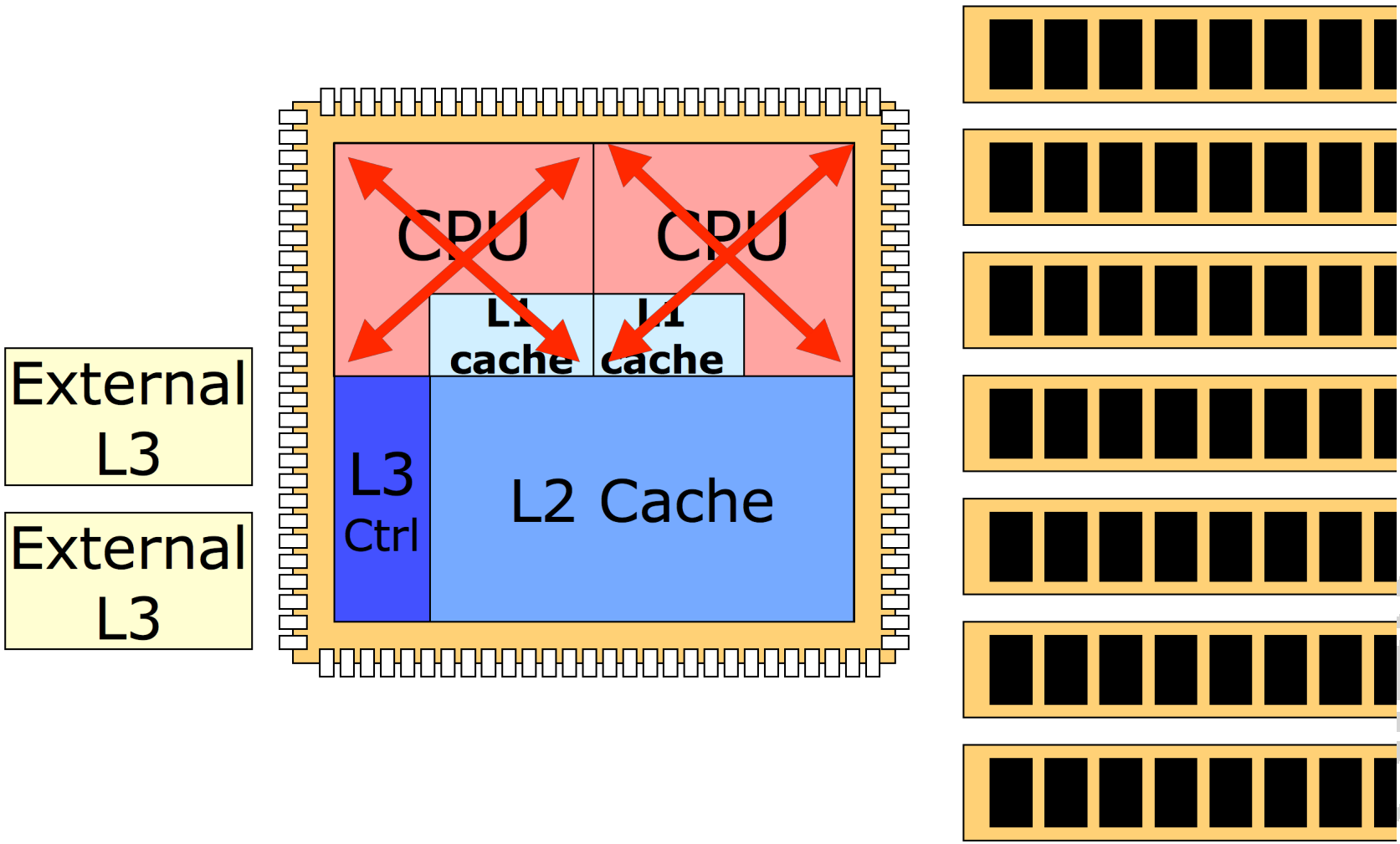


Simltaneous multithreading



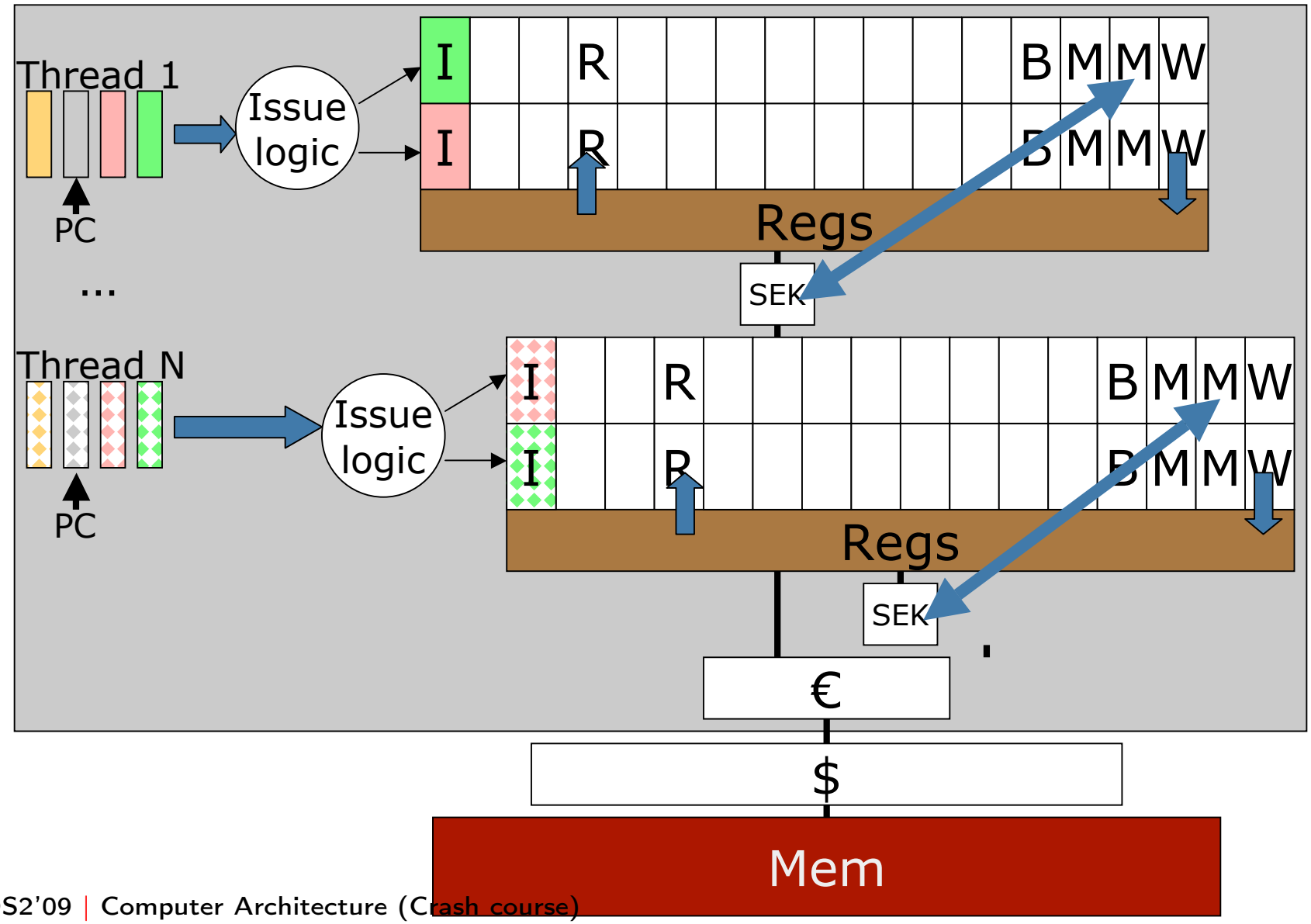
Bad News #2: Wire delay

→ Multiple small CPUs with private L1$



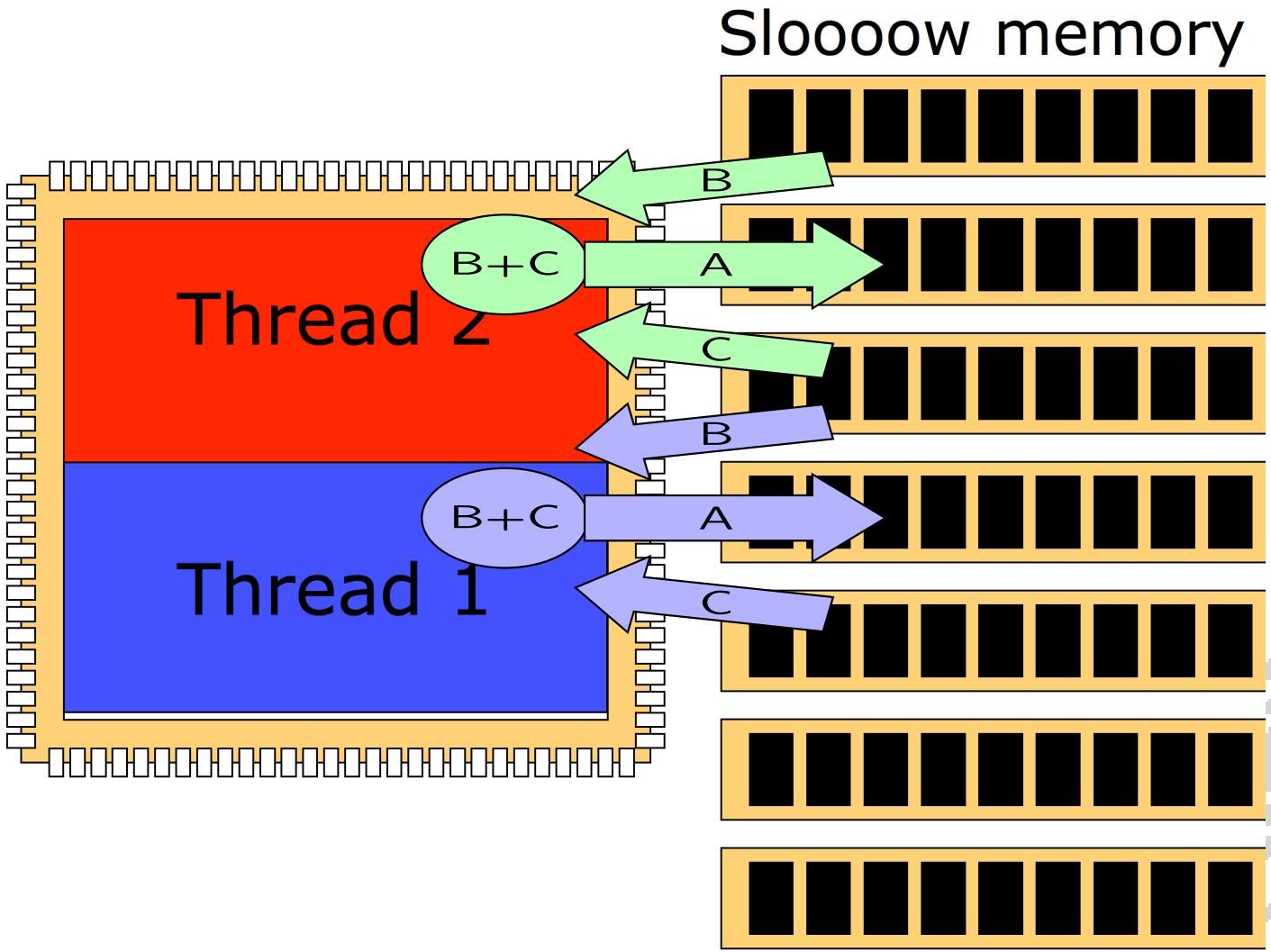
Bad News #2: Wire delay

→ Multiple small CPUs with private L1$



Bad News #3: Memory latency/bandwidth

→ memory accesses from many threads

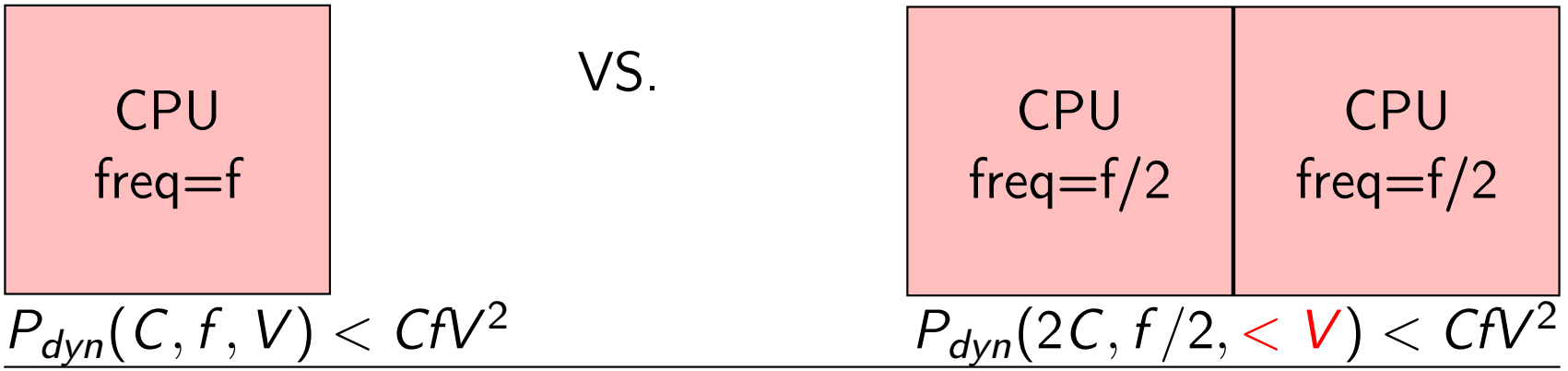


Thread-Level Parallelism → Memory-Level Parallelism (MLP)

Bad News #4: Power consumption

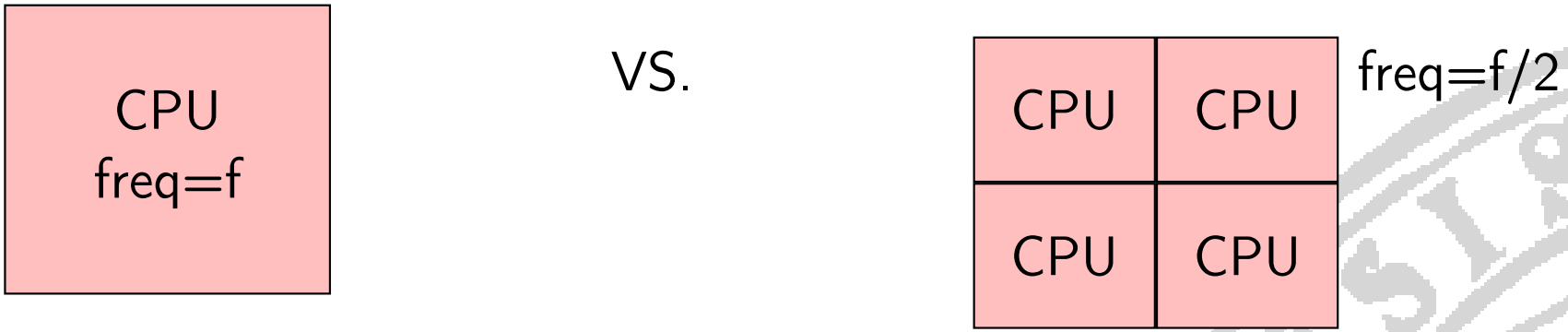
→ Lower the frequency → lower voltage

$$P_{dyn} = C * f * V^2 \approx area * freq * voltage^2$$



$P_{dyn}(C, f, V) < CfV^2$

$P_{dyn}(2C, f/2, < V) < CfV^2$



$P_{dyn}(C, f, V) < CfV^2$

$P_{dyn}(C, f/2, < V) < \frac{1}{2}CfV^2$

Solving all the problems (?):

Exploring thread parallelism

#1 Running out of ILP

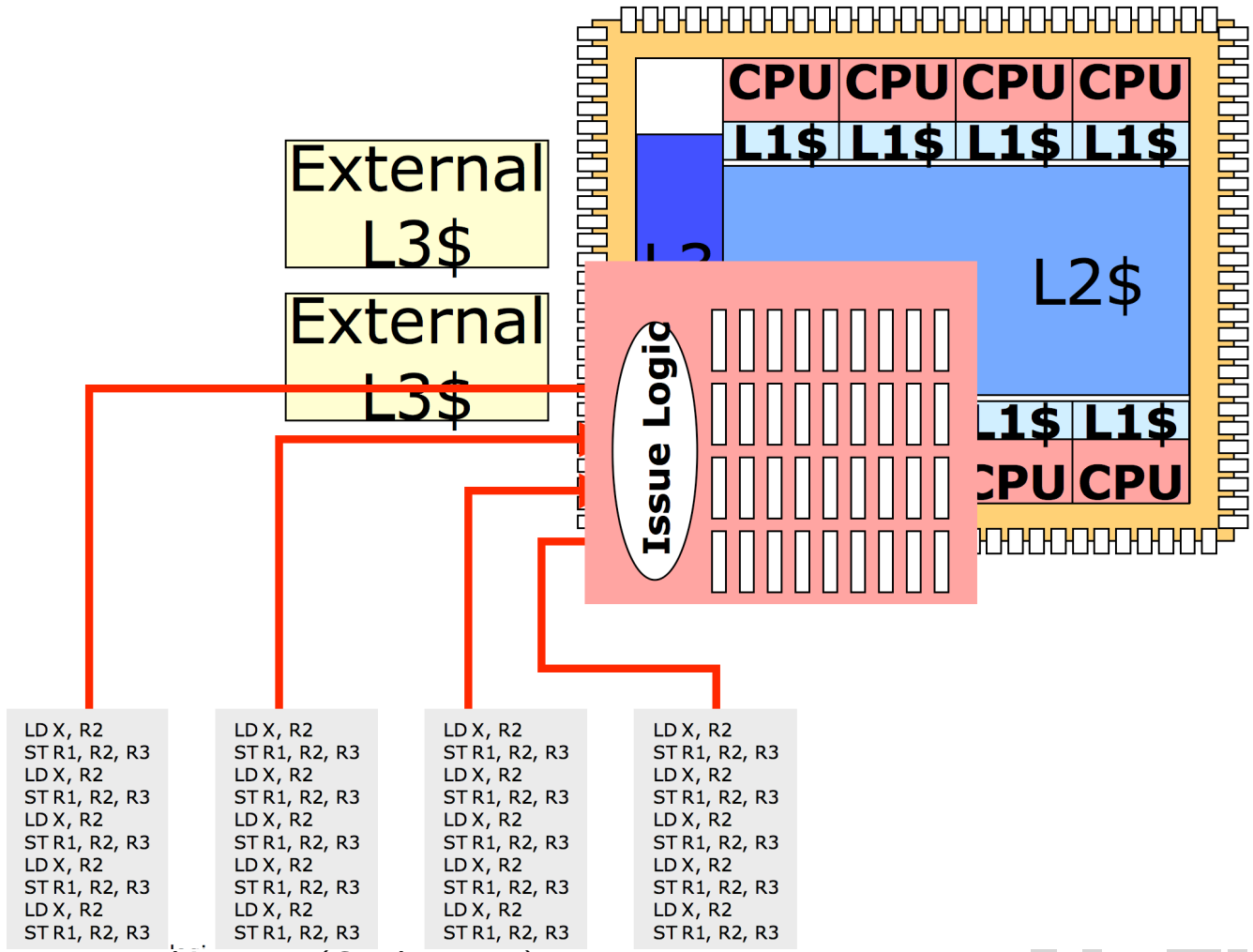
#2 Wire delay is starting to hurt

#3 Memory is the bottleneck

#4 Power is the limit

In all computers very soon...

Multicore processors, probably also with simultaneous multithreading



Scenario

Several cars want to drive from point A to point B.

They can compete for space on the same road
and end up either:

- following each other
- or competing for positions (and having accidents!).

Or they could drive in parallel lanes,
thus arriving at about the same time without getting in each other's way.

Or they could travel different routes, using separate roads.

Scenario

Several cars want to drive from point A to point B.

Programming

They can compete for space on the same road
and end up either:

- following each other
- or competing for positions (and having accidents!).

Programming

Or they could drive in parallel lanes,

thus arriving at about the same time without getting in each other's way.

Programming

Or they could travel different routes, using separate roads.

Definitions

Concurrent Program

- 2^+ processes working together to perform a task.
- Each process is a sequential program

  (= sequence of statements executed one after another)
- *Single thread of control* vs *multiple thread of control*

- Communication
- Synchronization

Correctness

Wanna write a concurrent program?

- What kinds of processes?
- How many processes?
- How should they interact?

Correctness

Ensure that processes interaction is properly synchronized

Ensuring the critical sections of statements do not execute at the same time

Delaying a process until a given condition is true

Our focus: imperative programs and asynchronous execution

Amdhal's law

- P is the fraction of a calculation that can be parallelized
- $(1 - P)$ is the fraction that is sequential

  (i.e. cannot benefit from parallelization)
- N processors

speedup

$$\frac{(1-P)+P}{(1-P)+P/N}$$

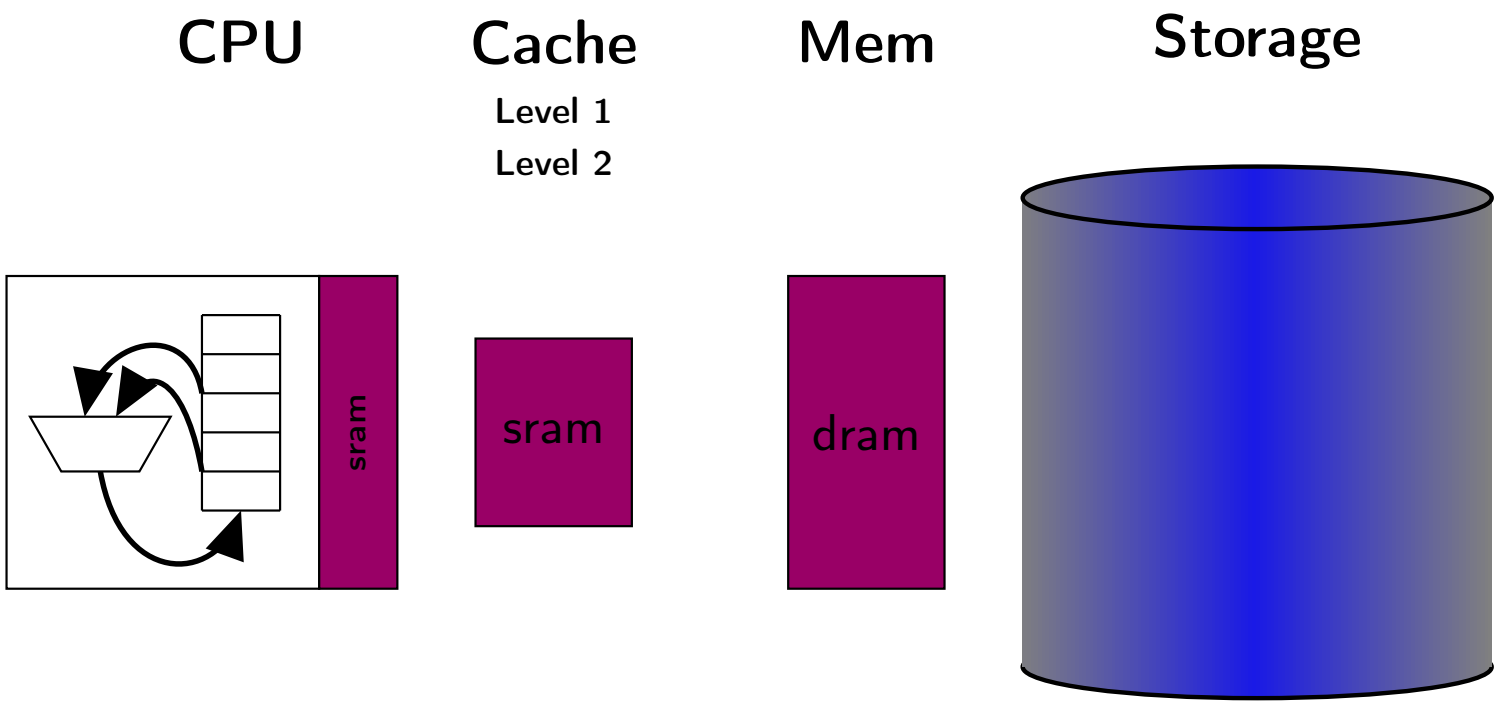
maximum speedup

Example

If $P = 90\% \Rightarrow$ max speedup of 10

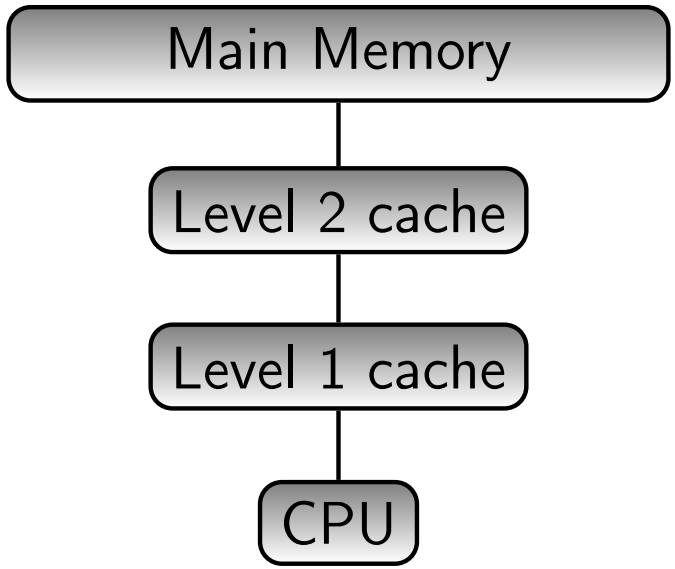
no matter how large the value of N used (ie $N \to \infty$)

Single-Processor machine



2000:	1ns 3ns	10ns	150ns	5 000 000ns
1982:	200ns	200ns	200ns	10 000 000ns

Memory Hierarchy



Why do we miss in the cache?

- miss

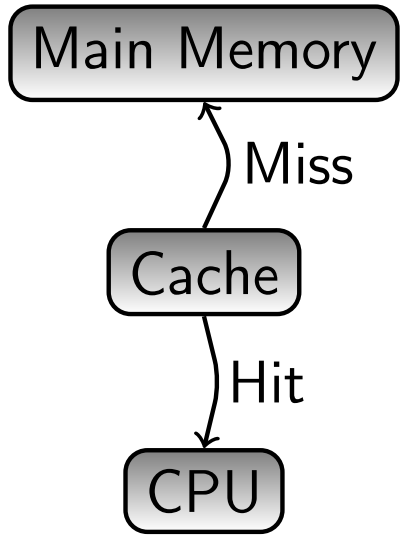
  Touching the data for the first time

- miss

  The cache is too small

- miss

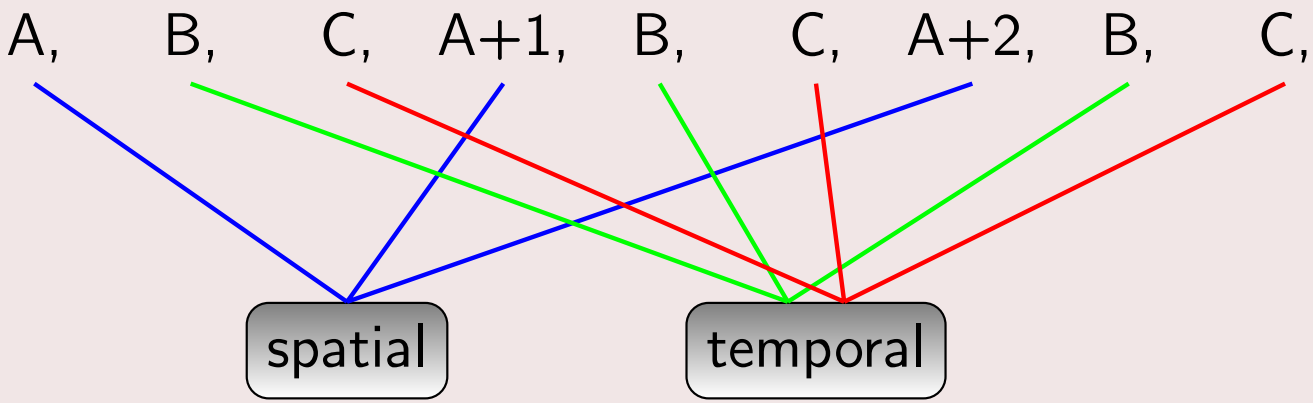
  Non-ideal cache implementation (data hash to the same cache line)



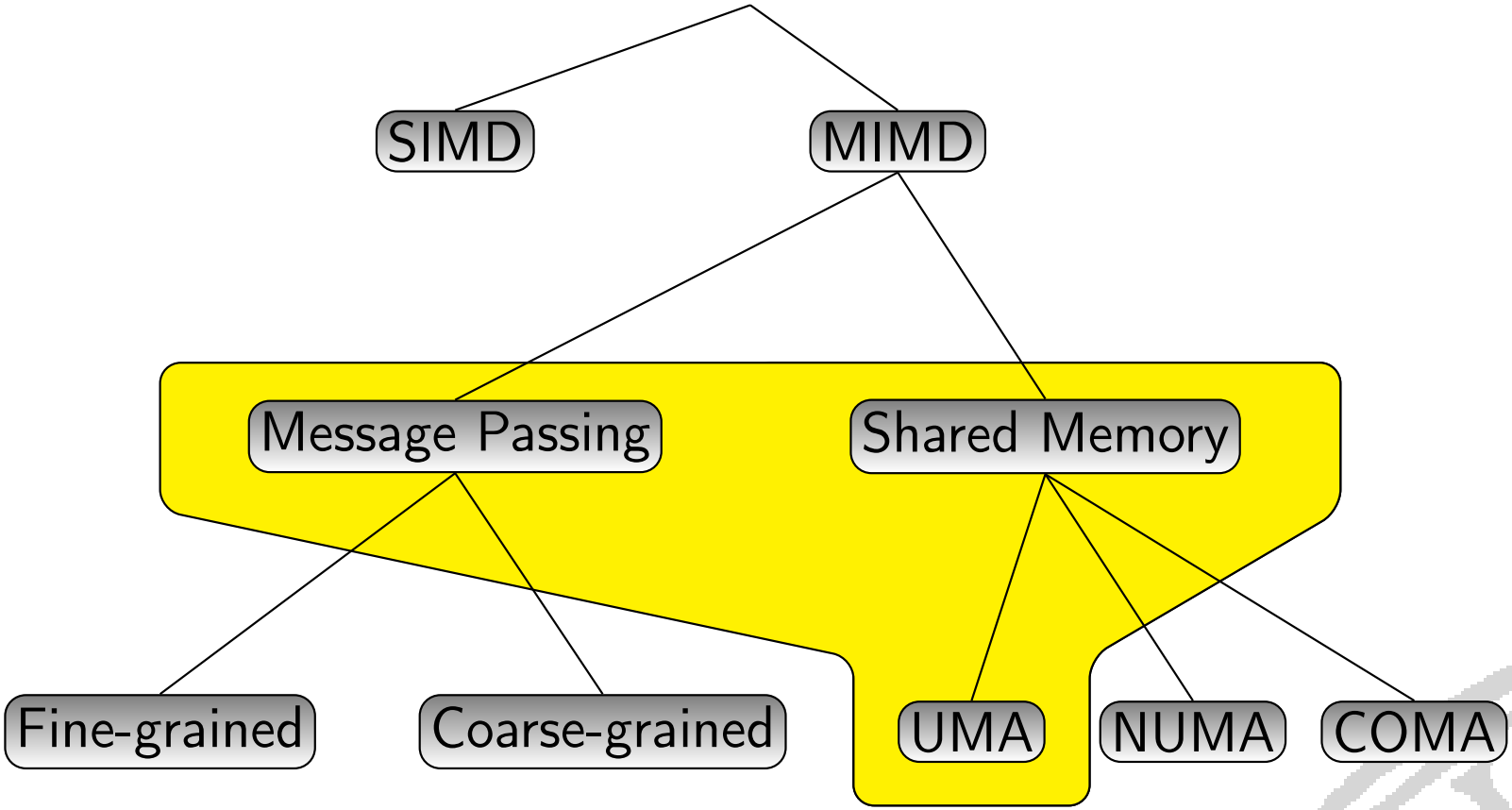
Locality

- locality
- locality

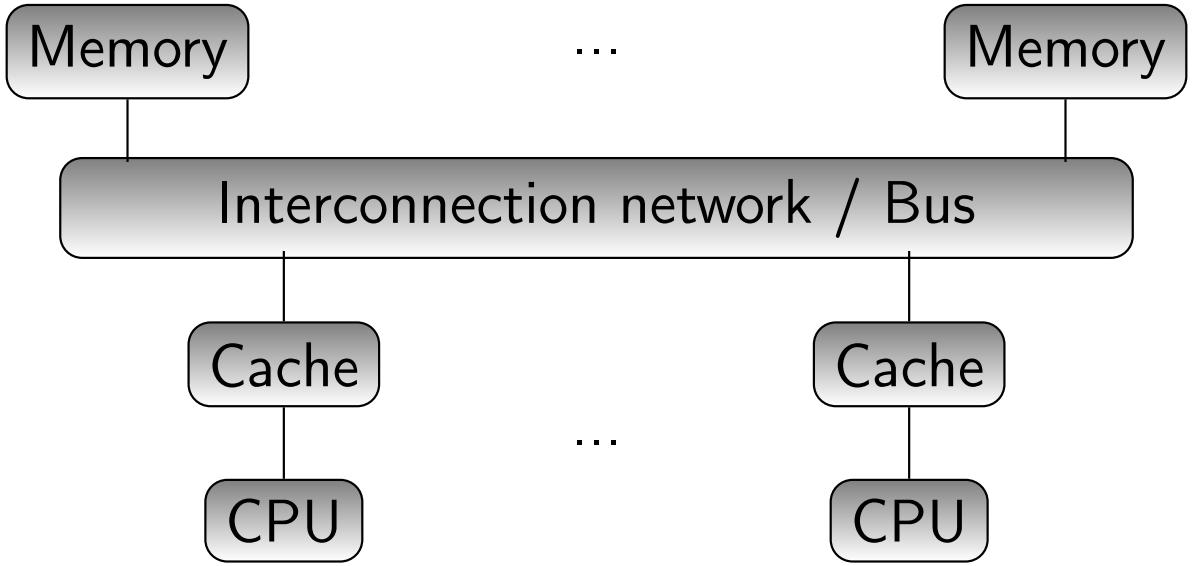
Inner loop stepping through an array



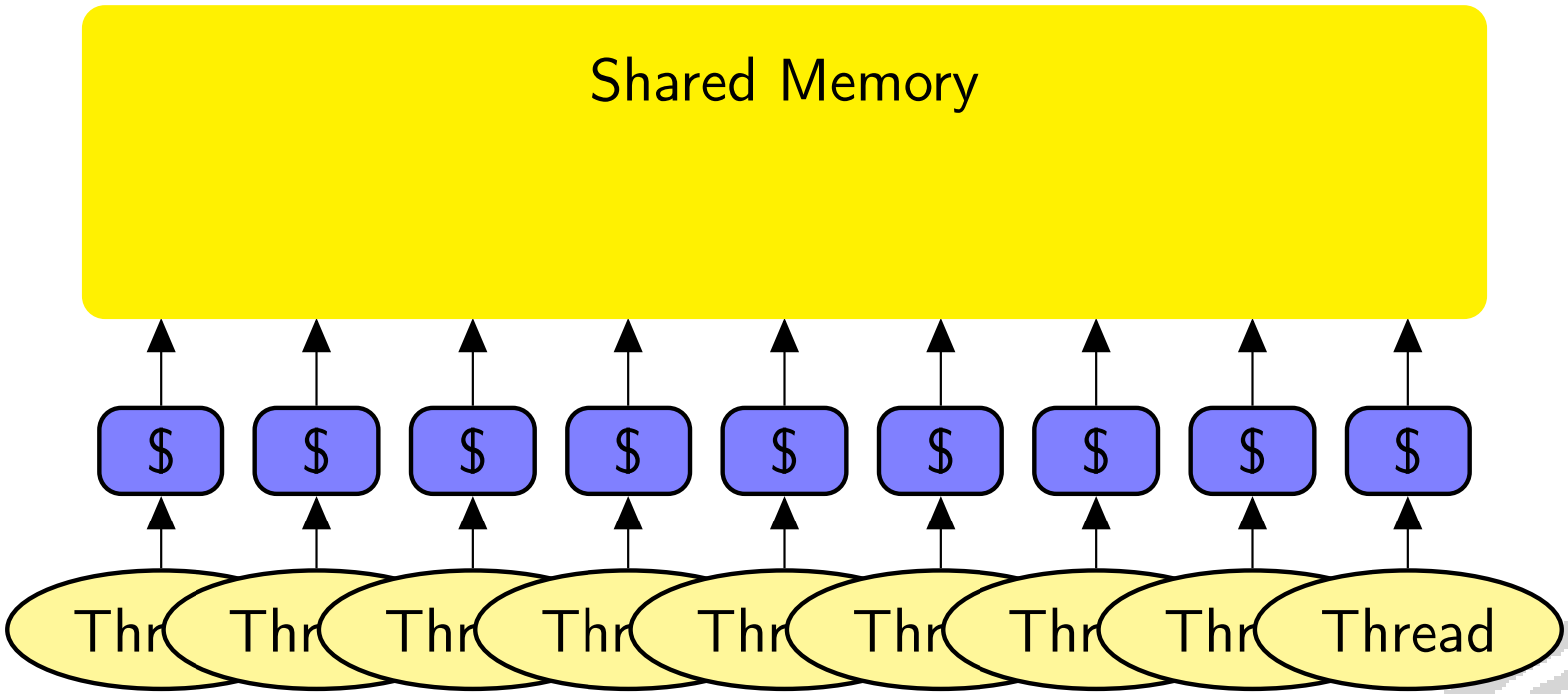
MultiProcessor world - Taxonomy



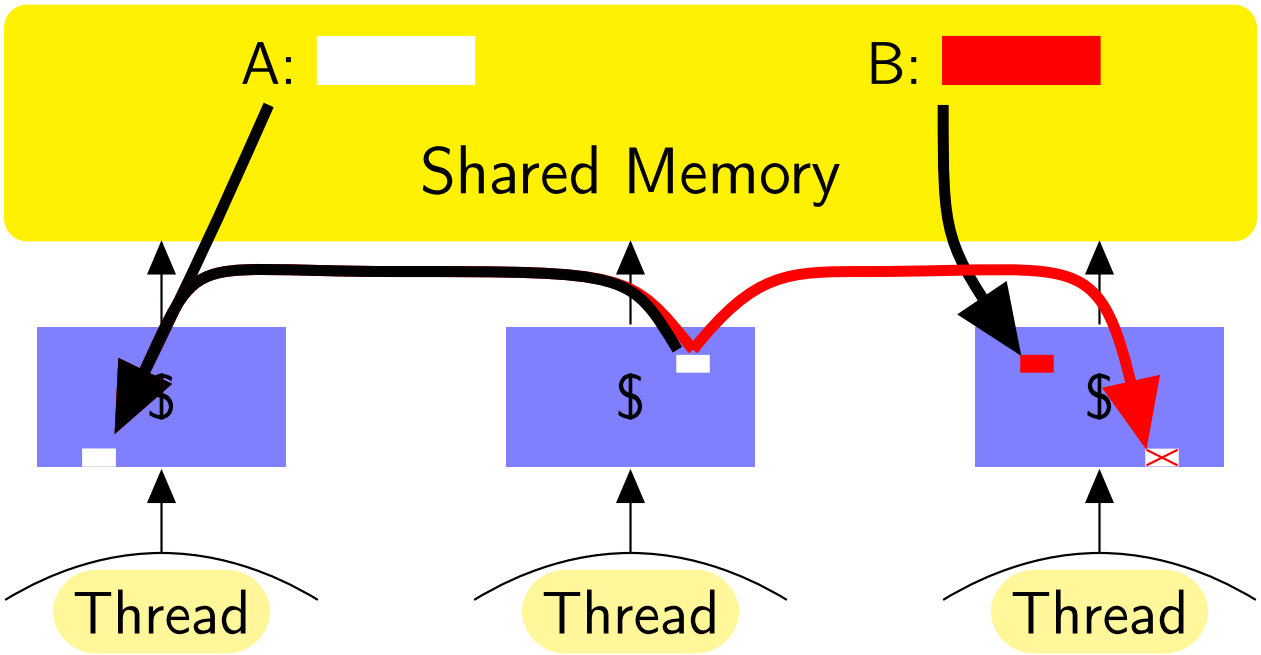
Shared-Memory Multiprocessors



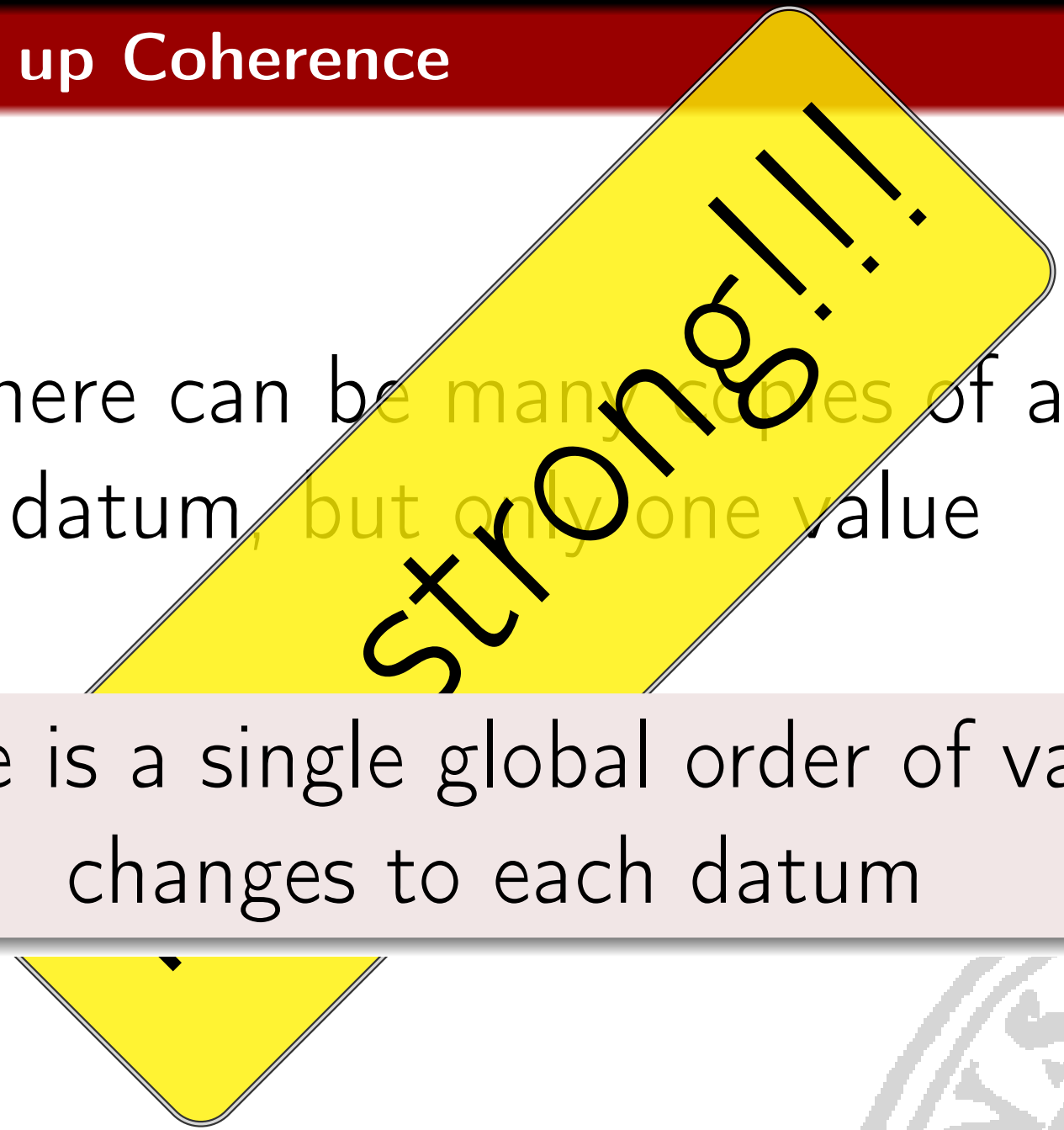
Programming Model



Cache coherency



Summing up Coherence



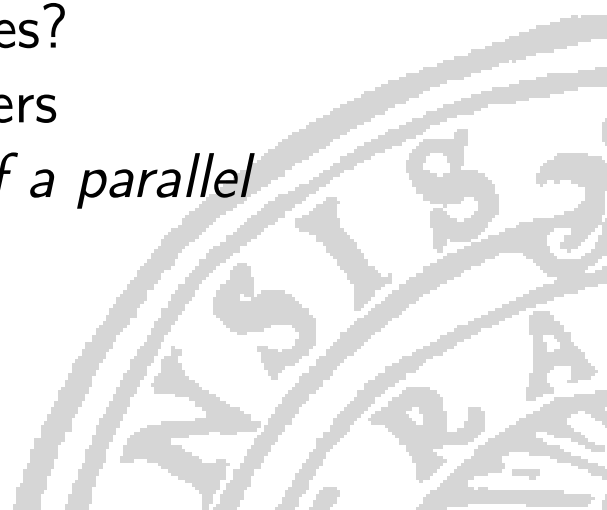
There can be many copies of a datum, but only one value

There is a single global order of value changes to each datum

Memory Ordering

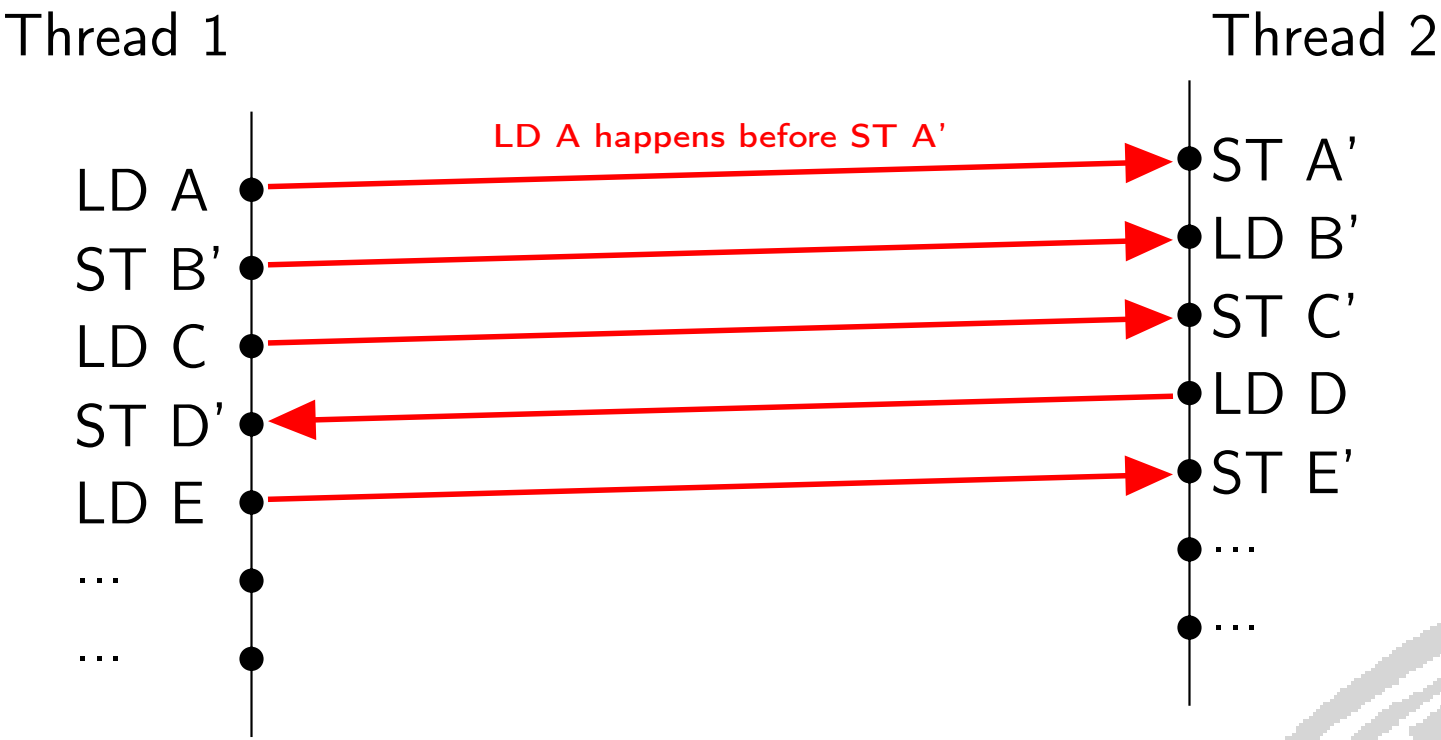
- The defines a per-datum order of value changes.
- The defines the order of value changes for all the data.

What ordering does the memory system guarantees?
"Contract" between the HW and the SW developers
Without it, we can't say much about the result of a parallel execution

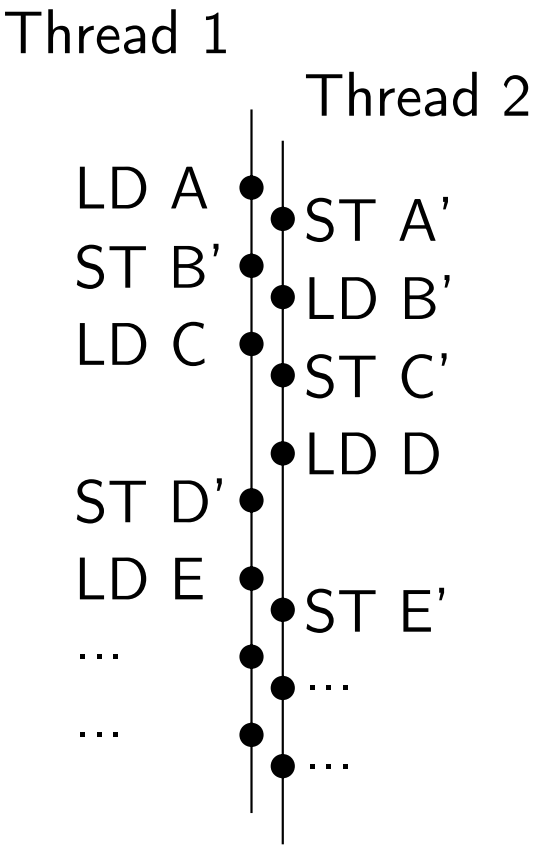


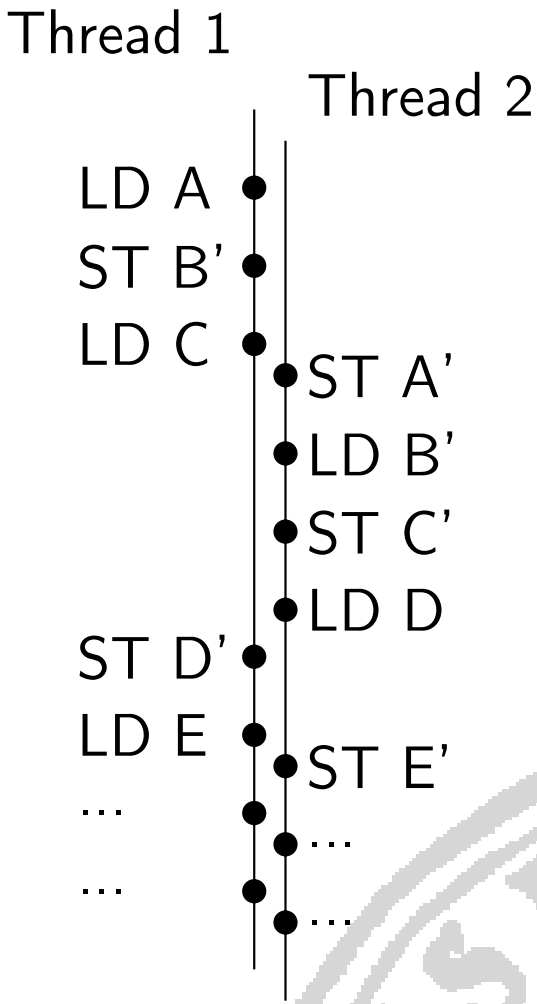
What order for these threads?

A' denotes a modified value to the datum at address A



Other possible orders?

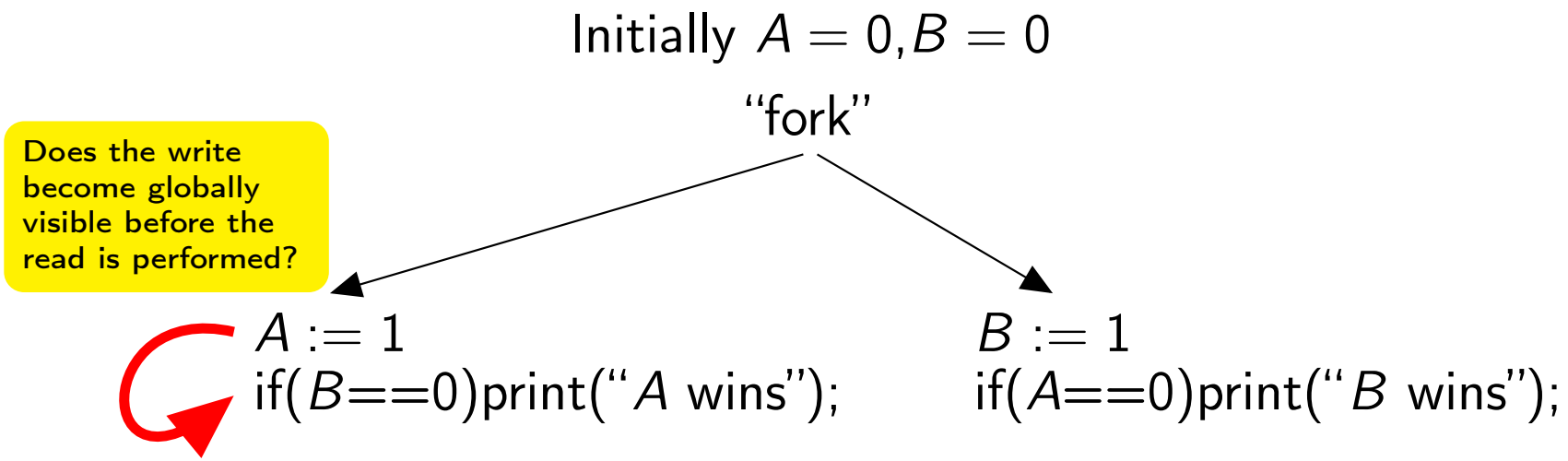




Memory model flavors

- : Programmer's intuition
- : Almost Programmer's intuition
- : No guaranty

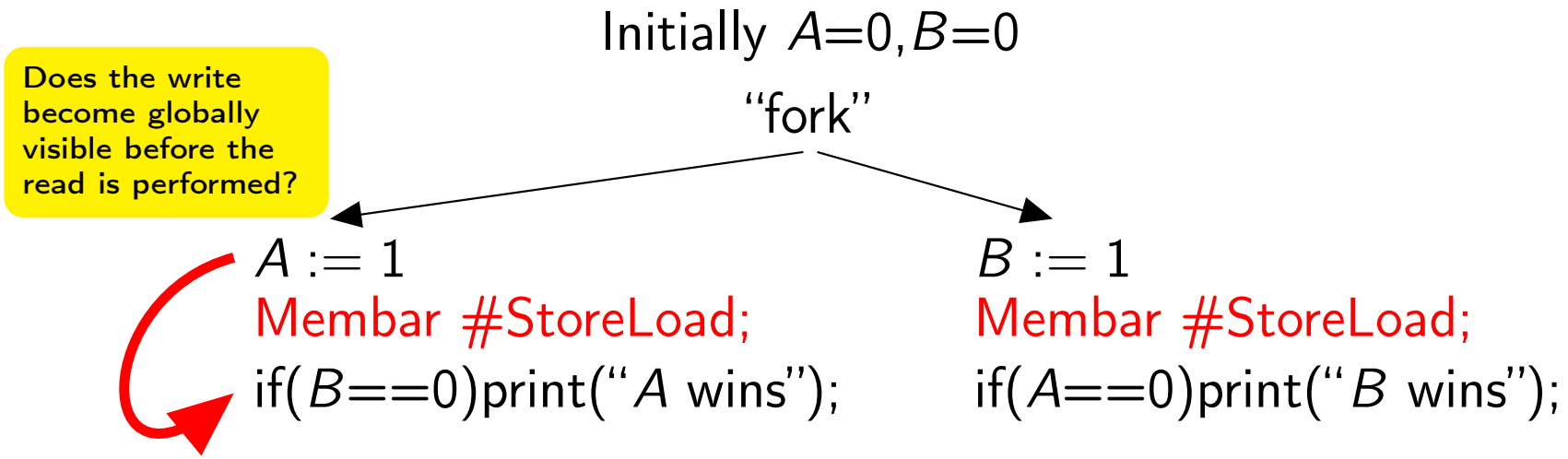
Dekker's algorithm



Can both A and B win?

Left: The read (ie, test if B==0) can bypass the store ($A := 1$)
Right: The read (ie, test if A==0) can bypass the store ($B := 1$)
$\Rightarrow$ Both loads can be performed before any of the stores
$\Rightarrow$ Yes, it is possible that both win!

Dekker's algorithm for Total Store Order



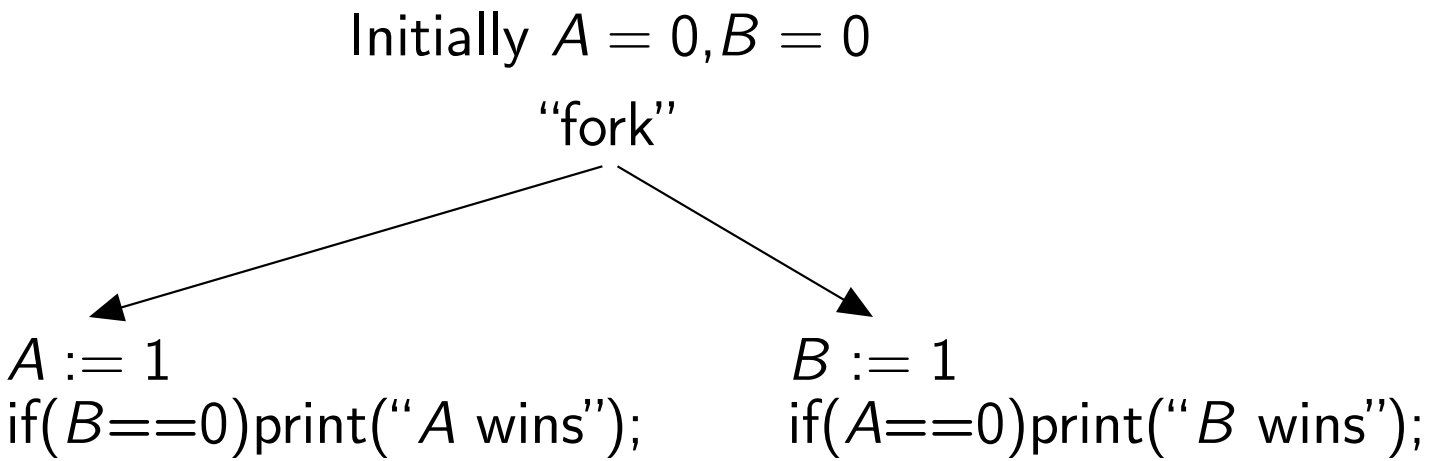
Can both A and B win?

Membar: the read is started after all previous stores have been "globally ordered"

$\Rightarrow$ Behaves like a sequentially consistent machine

$\Rightarrow$ No, they won't both win. Good job Mister Programmer!

Dekker's algorithm, in general



Can both A and B win?

The answer depends on the memory model

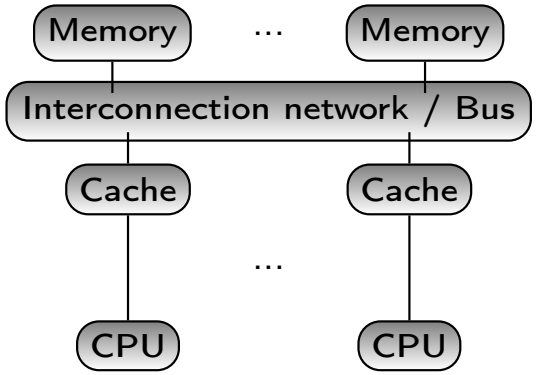
Remember? ...
Contract between the HW and SW developers.

So....

Memory Model
is a tricky issue

New issues

- Compulsory miss
- Capacity miss
- Conflict miss



-
  Cache-to-cache transfer
-
  Side-effect from large cache lines
- What about the compiler?

  Code reordering? *volatile* keyword in *C*...

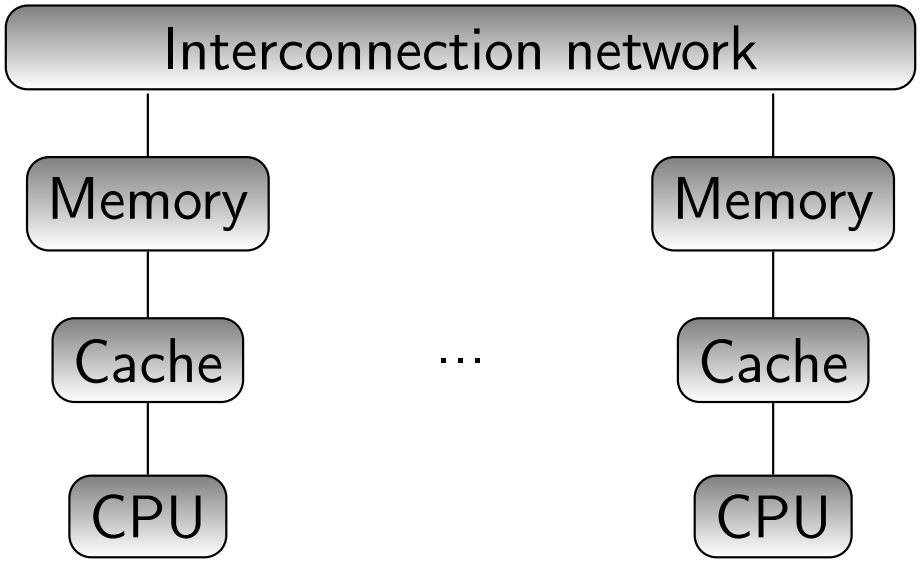
Good to know

Performance ⇒ Use of Cache
Memory hierarchy ⇒ Consistency problems

To get maximal performance on a given machine,

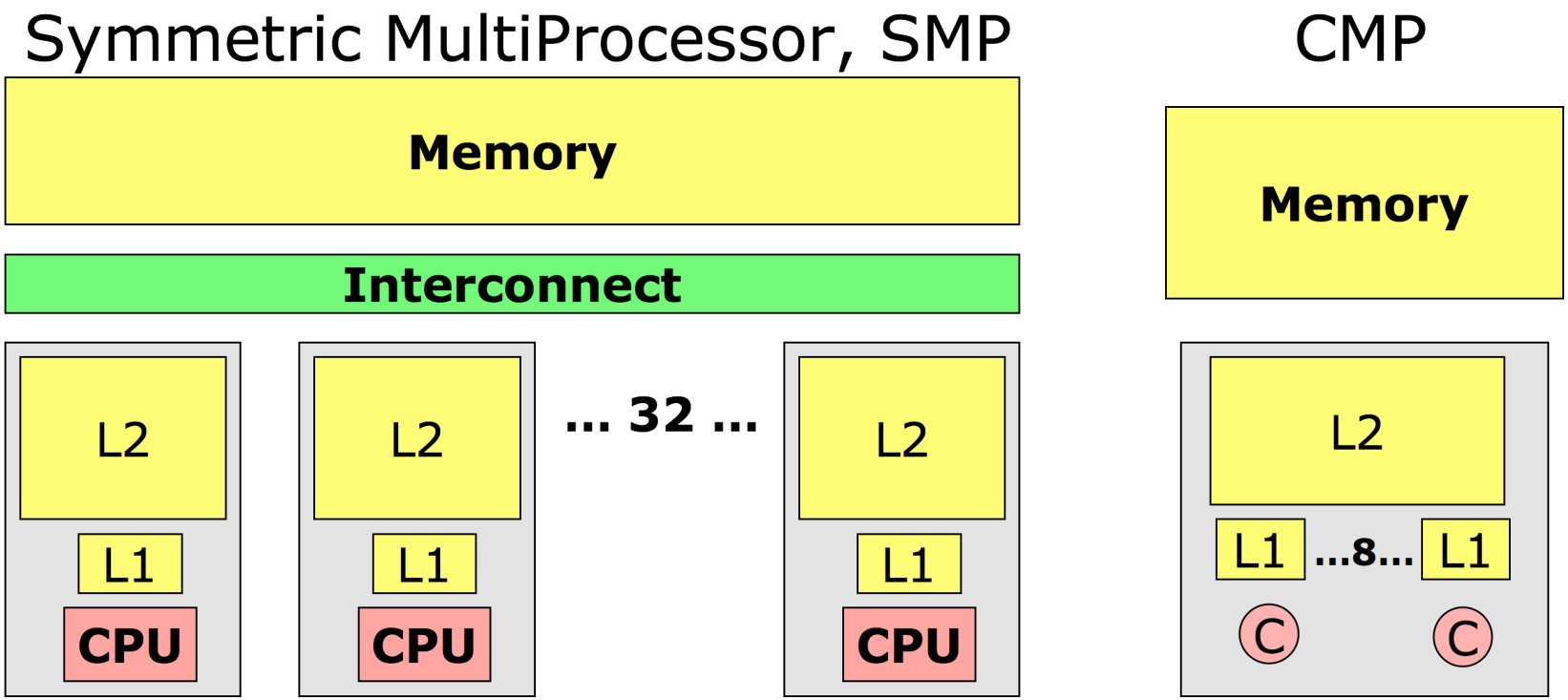
the programmer has to know about the characteristics of the memory system and has to write programs to account them

Distributed Memory Architecture

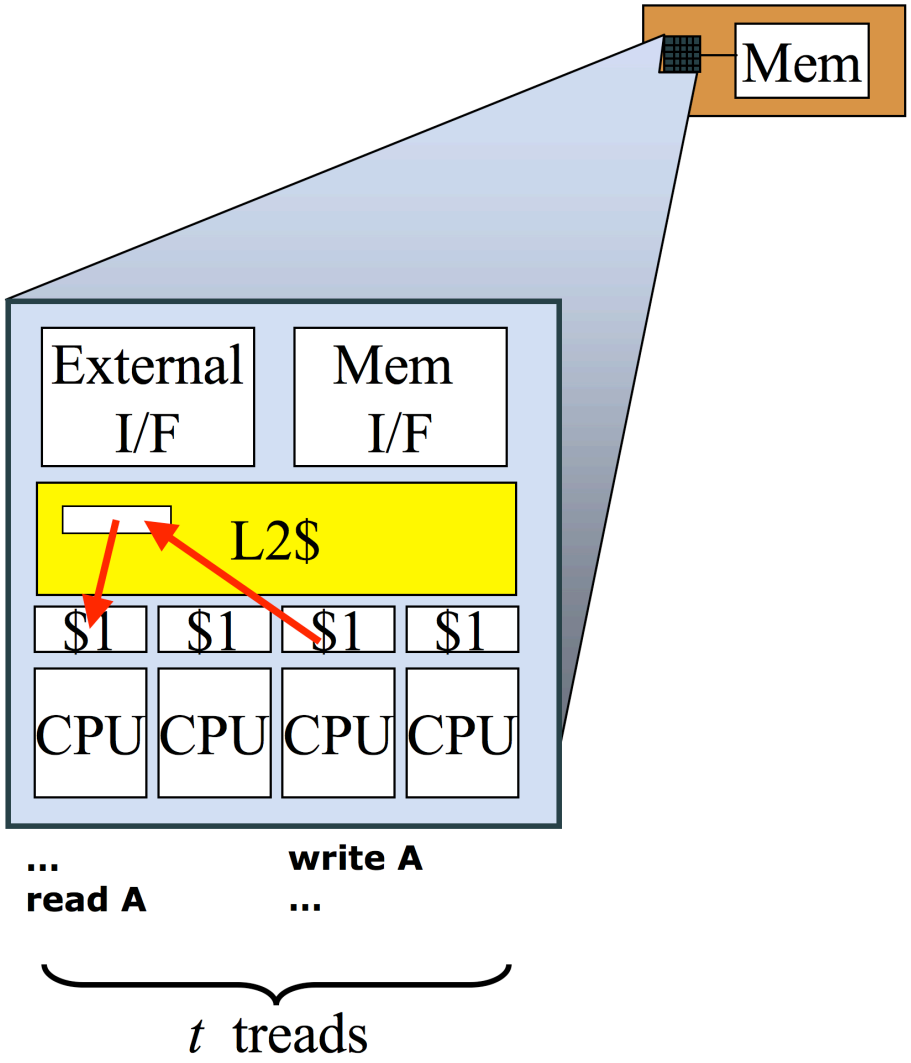


- Communication through Message Passing
- Own cache, but memory not shared
  $\Rightarrow$ No coherency problems

Isn't a CMP just a SMP on a chip?



Cost of communication?



Impact on Algorithms

For performance, we need to understand the interaction between algorithms and architecture.

The rules have changed

We need to question old algorithms and results!

Criteria for algorithm design

- Pre-CMP:
  - Communication is expensive: Minimize communication
  - Data locality is important
  - Maximize scalability for large-scale applications
- Within a CMP chip today:
  - (On-chip) communication is almost to free
  - Data locality is even more important
  - (SMT may help by hiding some poor locality)
  - Scalability to 2-32 threads
- In a multi-CMP system tomorrow:
  - Communication is sometimes almost free (on-chip), sometimes (very) expensive (between chips)
  - Data locality (minimizing of-chip references) is a key to efficiency
  - "Hierarchical scalability"