ECE 454
Computer Systems Programming

Final Review

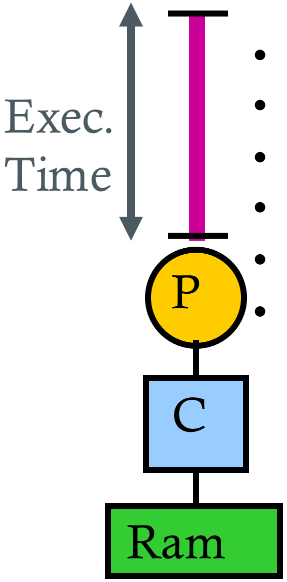
Jon Eyolfson
Courtesy: Ashvin Goel
ECE Dept, University of Toronto

Final Mechanics

- Final exam will cover all the material from the course
  - CPU architectures, compiler optimizations, performance optimization
  - Cache performance, dynamic memory and modern allocators
  - Threads and synchronization, parallel architectures and performance, better locking methods, avoiding locking
  - Map reduce
- Based upon lecture material and labs

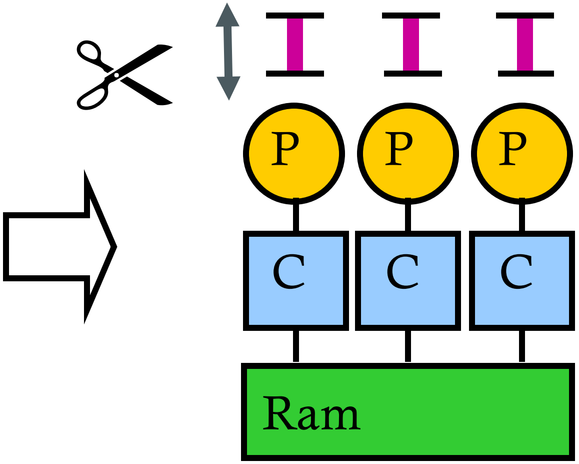
What we have learnt

Sequential program optimization:



- CPU architecture
- Profiling
- Compiler optimization
- Memory hierarchy
- Cache optimization
- Dynamic memory

Parallel programming on single machine:



- Threads
- Synchronization
- Parallel architecture and performance
- Better locking
- Avoiding locking

CPU Architectures

- Key techniques that make CPU fast
  - Pipelining
  - Branch prediction
  - Out-of-order execution
  - Instruction-level parallelism
  - Simultaneous multithreading

CPU architecture: Intel

Year	Processor	Tech.	CPI
1971	4004	no pipeline	*n*
1985	386	pipeline branch prediction	*close to 1* *closer to 1*
1993	Pentium	Superscalar	*< 1*
1995	PentiumPro	Out-of-Order exe.	*<< 1*
2000	Pentium IV	SMT	*<<<1*

Profiling

- Tools for profiling
  - gprof
  - gcov
  - unix time
  - perf
- Rationale behind profiling?
  - Amdahl's law
    - speedup = OldTime / NewTime
  - Implications of Amdahl's law?

Compiler optimizations

- Machine independent (apply equally well to most CPUs)
  - Constant propagation
  - Constant folding
  - Common subexpression elimination
  - Copy propagation
  - Dead code elimination
  - Loop invariant code motion
  - Function inlining

GCC -O1
(only inline very small func.)

- Machine dependent (apply differently to different CPUs)
  - Instruction Scheduling
  - Loop unrolling

GCC –O2
GCC –O3

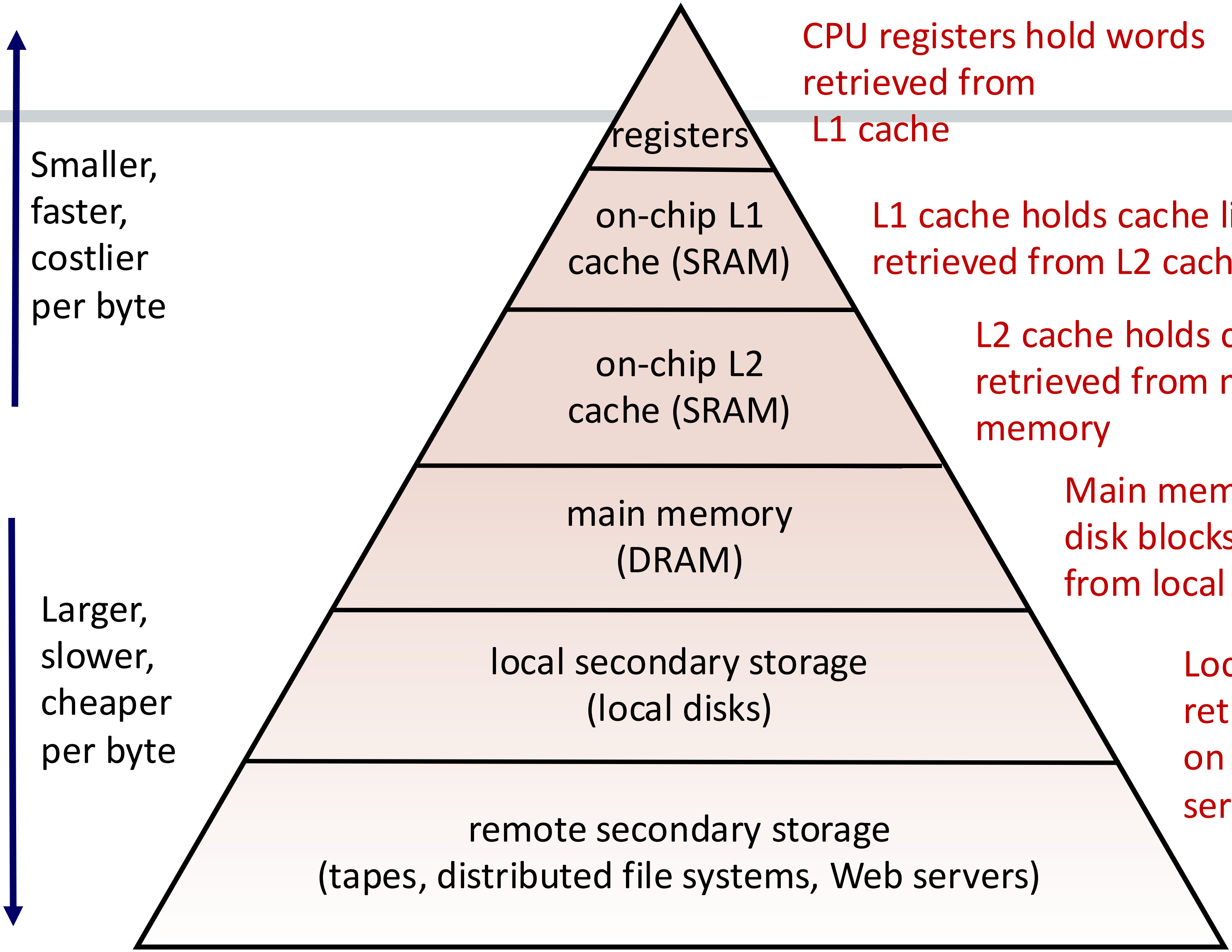
Might need to do manually.

Role of the Programmer

How should I write my programs, given that I have a good, optimizing compiler?

- Don’t: Smash Code into Oblivion
  - Hard to read, maintain, & assure correctness
- Do:
  - Select best algorithm
  - Write code that’s readable & maintainable
    - Procedures, recursion
    - Even though these factors can slow down code
  - Eliminate optimization blockers
    - Allows compiler to do its job
- Focus on Inner Loops
  - Do detailed optimizations where code will be executed repeatedly
  - Will get most performance gain here

Cache performance



Smaller,
faster,
costlier
per byte

registers

on-chip L1
cache (SRAM)

on-chip L2
cache (SRAM)

main memory
(DRAM)

local secondary storage
(local disks)

remote secondary storage
(tapes, distributed file systems, Web servers)

Larger,
slower,
cheaper
per byte

CPU registers hold words retrieved from L1 cache

L1 cache holds cache lines retrieved from L2 cache

L2 cache holds cache lines retrieved from main memory

Main memory holds disk blocks retrieved from local disks

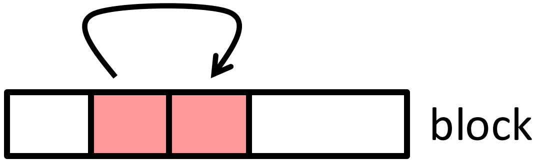
Local disks hold files retrieved from disks on remote network servers

Why Caches Work

- **Locality:** Programs tend to use data and instructions with addresses near or equal to those they have used recently

- **Temporal locality:**
  - Recently referenced items are likely to be referenced again in the near future

- **Spatial locality:**
  - Items with nearby addresses tend to be referenced close together in time

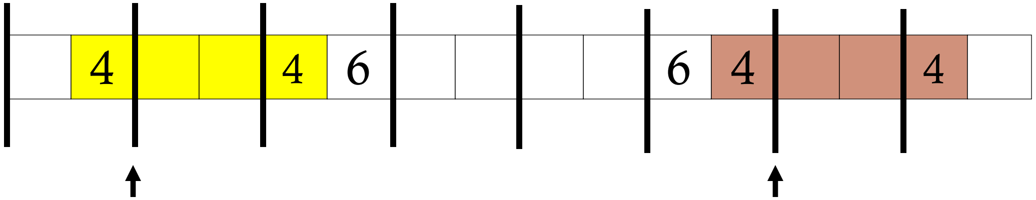


Optimize your program for cache performance

- Write code that has locality
  - Spatial: access data contiguously
  - Temporal: make sure access to the same data is not too far apart in time
- How to achieve locality?
  - Proper choice of algorithm
  - Loop transformations
    - Tiling

Dynamic memory management

- How do we know how much memory to free just given a pointer?



- How do we keep track of the free blocks?
  - Implicit list
  - Explicit list
  - Segregated free list
- How do we pick a block to use for allocation -- many might fit?
- How do we reinsert a freed block?
- How do phkmalloc and jemalloc work?

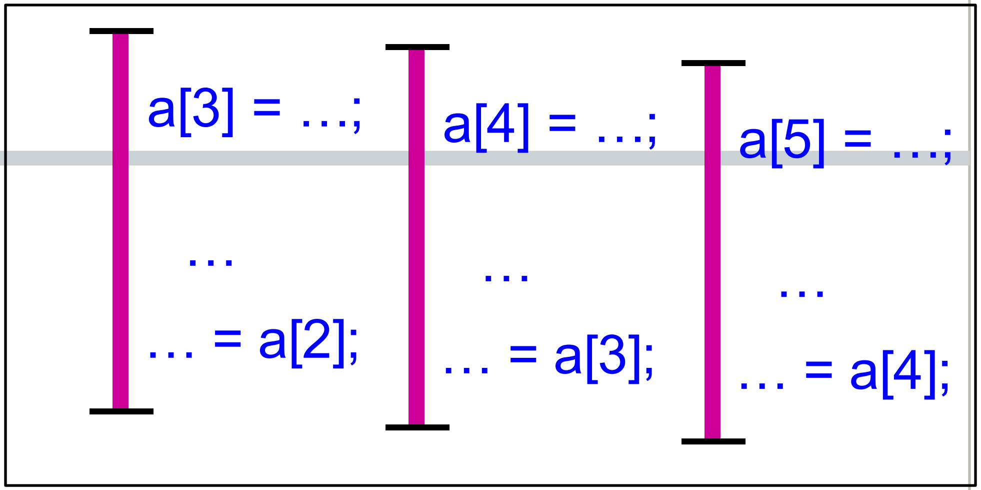
Multithreading

- What is multithreading?
- How do we share data across different threads?
- Communication and synchronization
    - ***Data race***
    - ***Deadlock***
- How to use `pthread` libraries to program
- Coarse-grain lock vs. fine-grain lock

Example: Parallelize this code

```
for( i=1; i<100; i++ ) {
        a[i] = …;
        …;
        … = a[i-1];
}
```

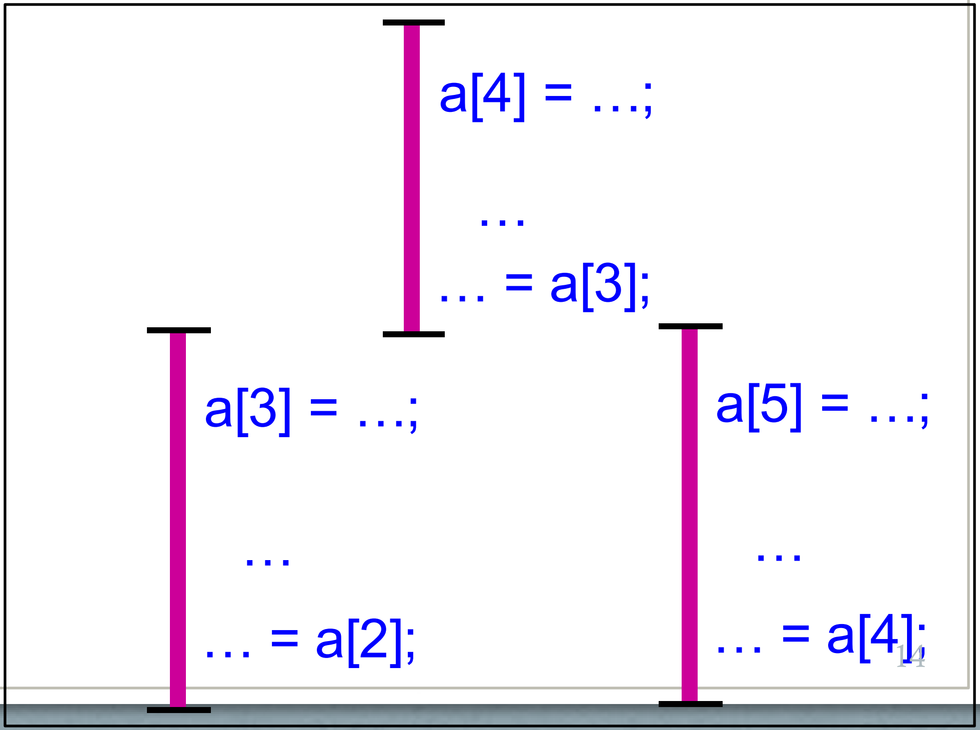
- Problem: each iteration depends on the previous
- Solution: appropriate synchronization



```
a[3] = …;
…
… = a[2];
```

```
a[4] = …;
…
… = a[3];
```

```
a[5] = …;
…
… = a[4];
```



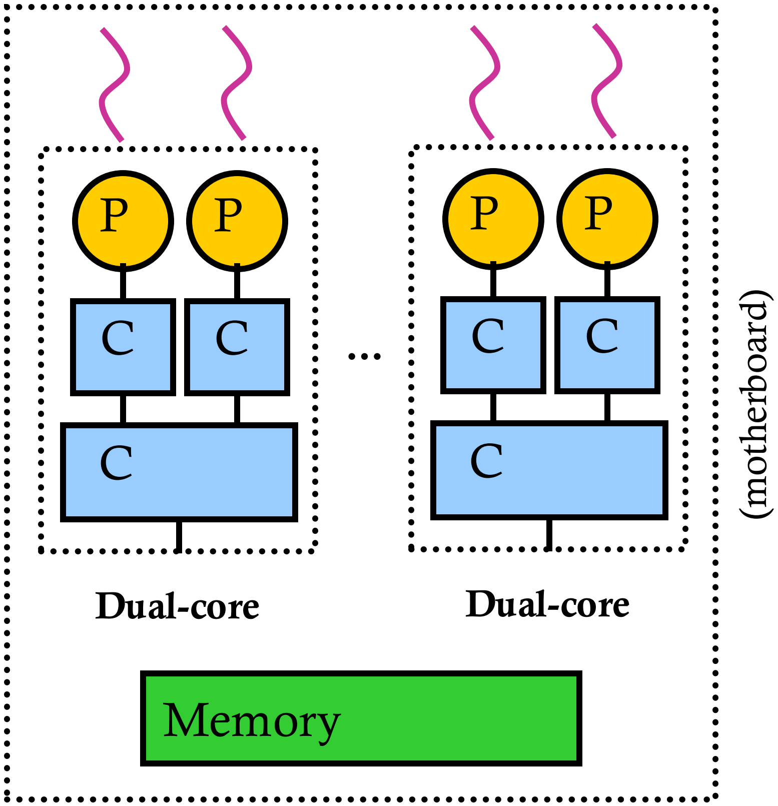
```
a[4] = …;
…
… = a[3];
```

```
a[3] = …;
…
… = a[2];
```

```
a[5] = …;
…
… = a[4];
```

Parallel architectures

- Cores have their private caches
- Cache lines may be duplicated
- Need protocol to ensure consistency



SMP
(Symmetric multiprocessing)

Cache coherence

- MESI
  - Modified
  - Exclusive
  - Shared
  - Invalid

  - Why is "Exclusive" needed?
- What is false sharing?
  - Why it is bad?

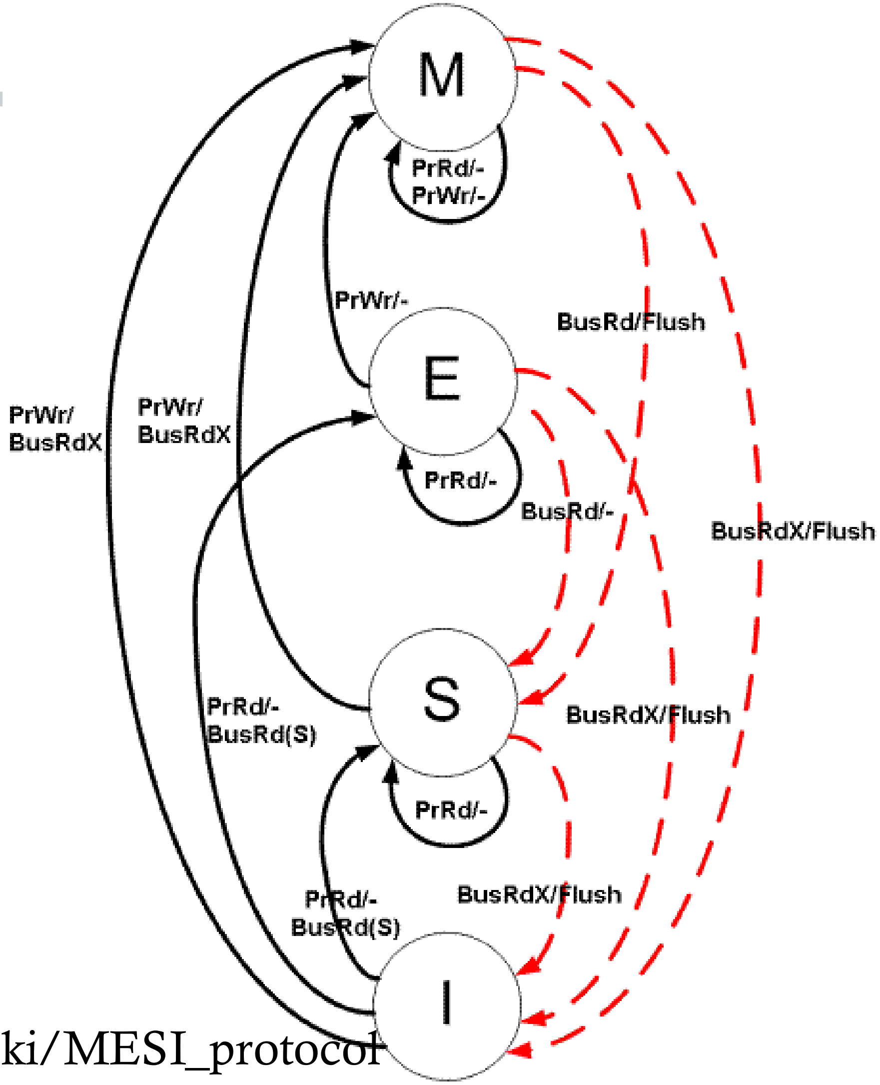


Image src.: http://en.wikipedia.org/wiki/MESI_protocol

Performance implications of parallel architecture

- Cache coherence is expensive (more than you thought)
  - Avoid unnecessary sharing (e.g., false sharing)
  - Atomic operations are expensive
    - Avoid unnecessary coherence (e.g., better locks)
- Crossing sockets is a killer
  - ***Can be slower than running the same program on single core!***
  - pthread provides CPU affinity mask
    - pin cooperative threads on cores within the same die

Memory Consistency

- Difference between memory coherence and consistency
- What is sequential consistency?
- Why is it expensive to implement sequential consistency
- Processor optimizations that lead to violating sequential consistency
- Using memory barriers for correct memory ordering

Better Locks

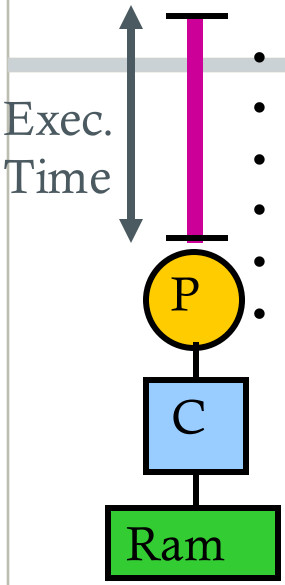
- How are locks implemented?
- Why is locking expensive?
- Why focus on spinlocks?
- Why are TTAS locks better than TAS spinlocks?
- Why are ticket locks better than TTAS locks?
- Why are queuing locks better than ticket locks?
- How are these locks implemented and what impact do they have on cache coherence?

Avoiding Locks

- What are the challenges with locking?
- What is non-blocking synchronization and how is it implemented?
- What is the ABA problem?
- What is RCU and how is it implemented? What are its various components and how do they interact with each other?

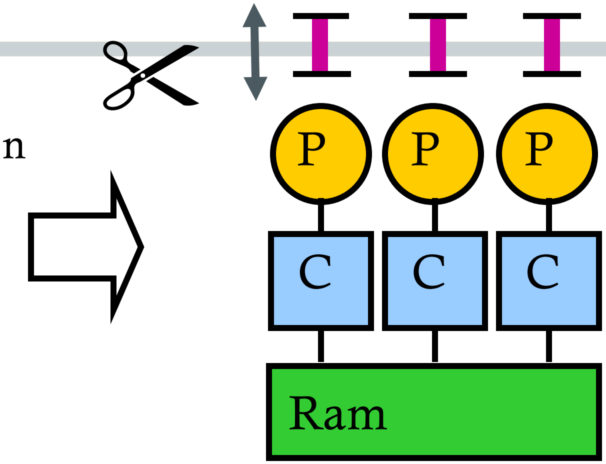
What we have learnt

Sequential program optimization:



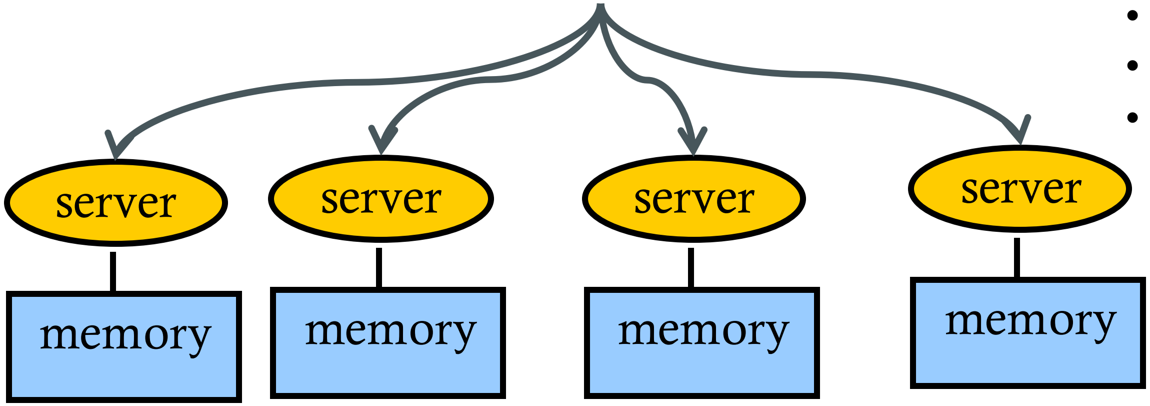
- CPU architecture
- Profiling
- Compiler optimization
- Memory hierarchy
- Cache optimization
- Dynamic memory

Parallel programming on single machine:



- Threads
- Synchronization
- Parallel architecture and performance
- Better locking
- Avoiding locking

Parallel programming on distributed system:



- MapReduce
- Distributed database
- Distributed memcache

MapReduce

- Why do we need MapReduce?
- What is MapReduce?
  - Programming model for big data analytics
  - Programmer writes two functions

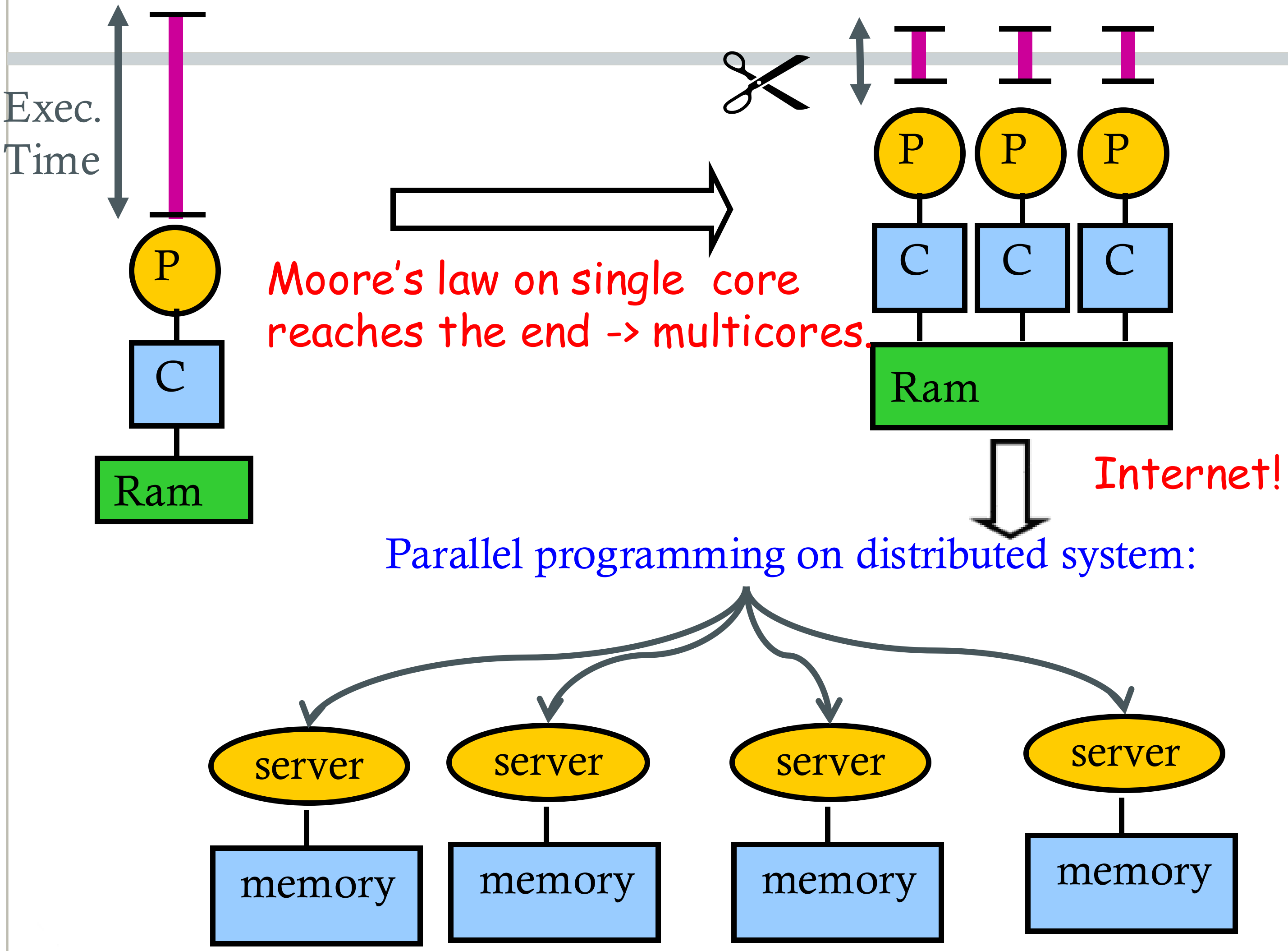
```
map (in_key, in_value) -> list(out_key, intermediate_value)
```

- Processes input key/value pair
- Produces set of intermediate pairs

```
reduce (out_key, list(intermediate_value)) -> list(out_key, outvalue)
```

- Processes a set of intermediate key-values

Technology is always changing



Sequential program optimization:

Parallel programming on single machine:

Moore's law on single core reaches the end -> multicores.

Internet!

Parallel programming on distributed system:

Is what we have learnt still going to be useful in 20 years?

- Why ask me now? Ask me in 2044…
- Technology is going to change …
  - Some techniques might not be relevant
  - Performance might not be very important at all
    - Correctness, easy-to-program, scalability, energy consumption…
- However, key ideas will still hold!
  - *"There is nothing new under the sun"*
  - *Amdahl's law: optimize for the bottleneck*
  - Cache: CPU cache -> memory cache (-> memcached -> CDN)
  - Parallelization
  - Avoid unnecessary computation (e.g., unnecessary sharing, sync., etc.)

More important: critical thinking

- “Why” is far more important than “how”
  - For each technique we have learnt, we discussed the “why”
    - E.g., why cache coherence impacts performance? why multi-core?
  - “How” is just a natural consequence of understanding “why”
  - The capability of asking the right “why” question and finding out the answer will keep you at the cutting edge of technology trends
- Skepticism + curiosity
  - Do we really need this technology?

The End

- Congratulations on surviving ECE 454!
  - It's a challenging course, but I hope you found it worthwhile
- Good luck, and thanks for a great class!
  - I really enjoyed it, and I hope the feeling is mutual

And if you haven't done so, please
submit your course evaluation, thanks!